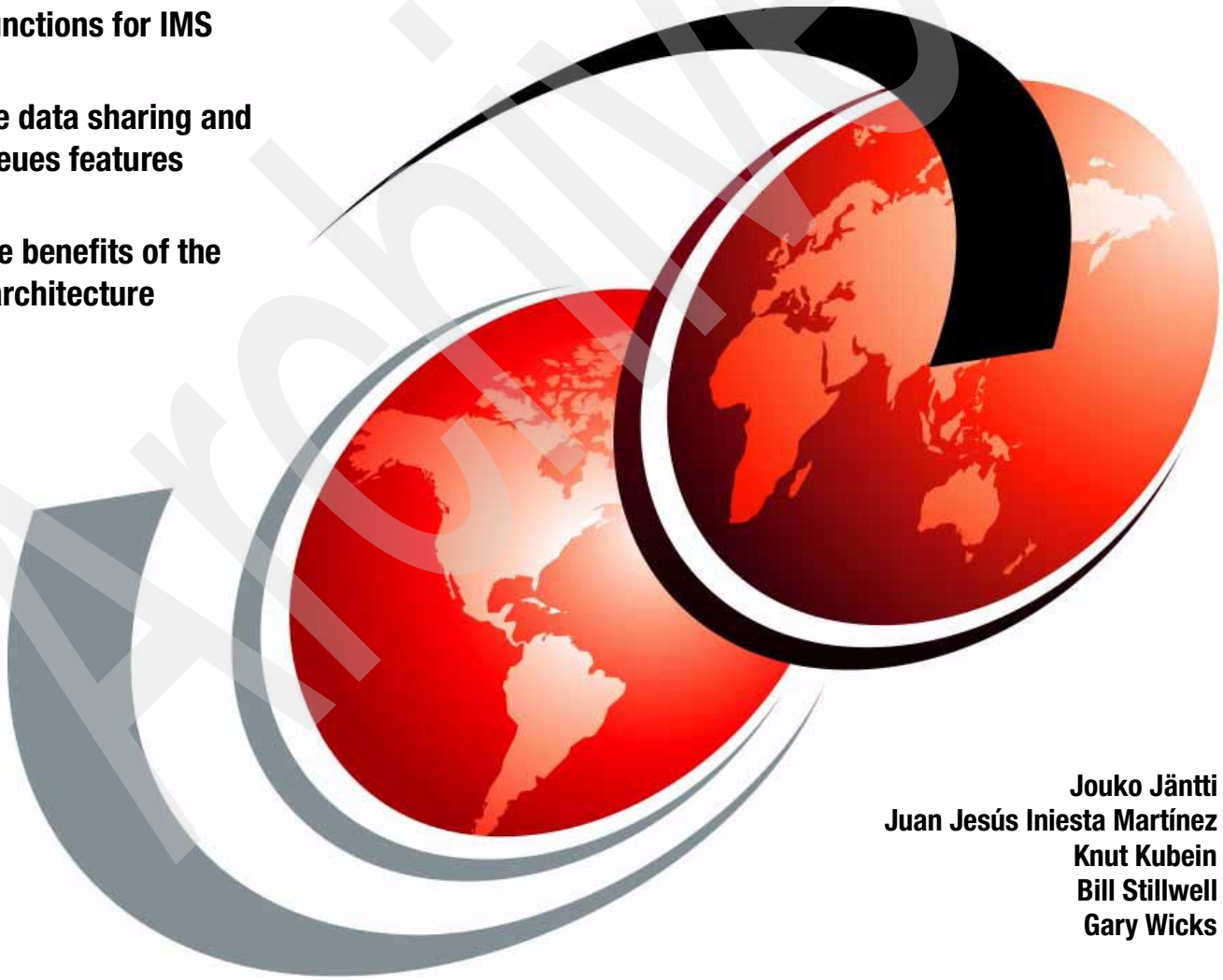
IBM

IMS in the Parallel Sysplex

Volume I: Reviewing the IMSplex Technology

- Become familiar with the Parallel Sysplex functions for IMS
- Review the data sharing and shared queues features
- Explore the benefits of the new CSL architecture



Jouko Jäntti
Juan Jesús Iniesta Martínez
Knut Kubein
Bill Stillwell
Gary Wicks

Archived

Redbooks

ibm.com/redbooks

Archived

International Technical Support Organization

IMS in the Parallel Sysplex
Volume I: Reviewing the IMSplex Technology

March 2003

Archived

SG24-6908-00

Note: Before using this information and the product it supports, read the information in “Notices” on page ix.

Archived

First Edition (March 2003)

This edition applies to IMS Version 8 (program number 5655-C56) or later for use with the OS/390 or z/OS Operating System.

© Copyright International Business Machines Corporation 2003. All rights reserved.
Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

© Copyright IBM Corp. 2003. All rights reserved.

Archived

Archived

Archived

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:
This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

© Copyright IBM Corp. 2003. All rights reserved.

Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

Advanced Peer-to-Peer Networking®
Affinity™
CICS®
DataPropagator™
DB2®
DFS™
DFSMSdss™

Extended Services®
ES/9000®
ESCON®
FICON™
IBM®
IMS™
IMS/ESA®
Language Environment®
MQSeries®
MVS™
MVS/ESA™
NetView®
OS/390®
Parallel Sysplex®
Perform™
Redbooks™
Redbooks(logo)™ 
RACF®
RAMAC®
RMF™
S/390®
SP™
VM/ESA®
VSE/ESA™
VTAM®
WebSphere®
z/OS™

The following terms are trademarks of other companies:

ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

C-bus is a trademark of Corollary, Inc. in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

SET, SET Secure Electronic Transaction, and the SET Logo are trademarks owned by SET Secure Electronic Transaction LLC.

Other company, product, and service names may be trademarks or service marks of others.

Archived

Preface

This IBM® Redbook is the first volume of a series of redbooks called IMS™ in the Parallel Sysplex®. These redbooks describe how IMS exploits the Parallel Sysplex functions and how to plan for, implement, and operate IMS systems working together in a Parallel Sysplex. We use the term *IMSplex* to refer to multiple IMSs which are cooperating with each other in a Parallel Sysplex environment to process a common shared workload. Although we generally think of an IMSplex in terms of online environments, an IMSplex can include batch IMS jobs as well as IMS utilities.

This redbook provides an overview of the Parallel Sysplex and the services offered to authorized programs such as IMS. It then continues with the description of the functionality that IMS provides in a Parallel Sysplex environment. Each function supported by IMS is presented in one or more chapters, and includes:

- Introduction to the Parallel Sysplex

  This is a brief, high level description of the objectives and benefits of a Parallel Sysplex, the hardware and software components, and the services provided by cross-system coupling facility (XCF) and cross-system extended services (XES). We also identify (again at a high level) which of these services IMS exploits to provide the user with the functionality described in the following chapters. We assume that you, as the IMS user, are responsible for the implementation of the IMSplex, not the Parallel Sysplex.

- Block level data sharing

  The first step in migrating to a Parallel Sysplex environment is usually to implement block level data sharing. When enabled, IMS batch and/or online applications running anywhere in the IMSplex can access all of the data in the shared IMS databases, allowing the user to distribute the workload according to the business requirements.

- Connectivity

  Once shared data is available from multiple IMS members in a Parallel Sysplex, the user must plan for and implement techniques for distributing the batch and transaction workload across the multiple IMSs in the IMSplex.

- Shared queues

  Shared queues is one technique for distributing the IMS transaction workload across multiple members of the IMSplex. When enabled, any transaction can be entered from the network to any IMS in the IMSplex and be executed by any IMS in the shared queues group. When one IMS is not available, or too busy to process a transaction, others can step in and pick up the work.

- Common Service Layer

  The Common Service Layer (CSL), introduced in IMS Version 8, improves the user's ability to manage an IMSplex. It includes improvements in both operations and recovery, as well as a capability to coordinate the online change process across all IMSs in the IMSplex.

The other volumes in this series are:

- *IMS in the Parallel Sysplex, Volume II: Planning the IMSplex,* SG24-6928
- *IMS in the Parallel Sysplex, Volume III: IMSplex Implementation and Operations,* SG24-6929

© Copyright IBM Corp. 2003. All rights reserved.

The team that wrote this redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization, San Jose Center.

Jouko Jäntti is a Project Leader specializing in IMS with the IBM International Technical Support Organization, San Jose Center. He holds a bachelor's degree in Business Information Technology from Helsinki Business Polytechnic in Finland. Before joining the ITSO in September 2001, Jouko worked as an Advisory IT Specialist at IBM Global Services in Finland. Jouko has been working on several e-business projects with customers as a Specialist in IMS, WebSphere®, and UNIX on the OS/390® platform. He has also been responsible for the local IMS support for Finnish IMS customers. Prior to joining IBM in 1997, he worked as a Systems Programmer and Transaction Management Specialist in a large Finnish bank for 13 years, and he was responsible for the bank's IMS systems.

Juan Jesús Iniesta Martínez is a Senior I/T Specialist with IBM Global Services in Spain. He has 15 years of experience in the IMS field. Prior to joining IBM in 1995, he worked for 10 years as a Console Operator and 8 years as an IMS Systems Programmer and Technical Support Specialist in the IMS team of a large Spanish bank. Since Juan joined IBM, he was assigned to "la Caixa", one of the largest financial entities in Spain. His areas of expertise and responsibilities include IMS installation and maintenance, IMS System Management, IMS problem determination and related products, Parallel Sysplex architecture, IMS data sharing, and IMS shared queues installation.

Knut Kubein is a Senior Advisory I/T Specialist with IBM Global Services in Germany. He has 30 years experience with IBM Large Systems Products working as a Technical Support Specialist, with 20 of those years devoted to IMS. He leads the IMS Support Team in EMEA. His areas of expertise include IMS data sharing, shared queues, and the Parallel Sysplex architecture. He supports large Bank accounts as well as other Industrial IMS users. He has written extensively about shared queues.

Bill Stillwell is a Senior Consulting I/T Specialist and has been providing technical support and consulting services to IMS customers as a member of the Dallas Systems Center for 20 years. During that time, he has developed expertise in application and database design, IMS performance, Fast Path, data sharing, shared queues, planning for IMS Parallel Sysplex exploitation and migration, DBRC, and database control (DBCTL). He also develops and teaches IBM Education and Training courses, and is a regular speaker at the annual IMS Technical Conferences in the United States and Europe.

Gary Wicks is a Certified I/T Specialist in Canada. He has 28 years of experience with IBM in software service and is currently on assignment as an IMS Development Group Advocate. He holds a degree in Mathematics and Physics from the University of Toronto. His areas of expertise include IMS enablement in Parallel Sysplex environments and he has written several IBM Redbooks™ on this subject over the last six years.

Thanks to the following people for their contributions to this project:

Rich Conway
Bob Haimowitz
International Technical Support Organization, Poughkeepsie Center, USA

Rose Levin
Jim Bahls
Mark Harbinski
Claudia Ho
Judy Tse

Sandy Stoob
Pedro Vera
Francis Ricchio
IBM Silicon Valley Laboratory, USA

Suzie Wendler
Rich Lewis
IBM USA

Alan Cooper
Pete Sadler
IBM UK

Rick Long
IBM Australia

Become a published author

Join us for a two- to six-week residency program! Help write an IBM Redbook dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You'll team with IBM technical professionals, Business Partners and/or customers.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you'll develop a network of contacts in IBM development labs, and increase your productivity and marketability.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our Redbooks to be as helpful as possible. Send us your comments about this or other Redbooks in one of the following ways:

- Use the online **Contact us** review redbook form found at:

  ibm.com/redbooks

- Send your comments in an Internet note to:

  redbook@us.ibm.com

- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. QXXE Building 80-E2
  650 Harry Road
  San Jose, California 95120-6099

Archived

Archived

1

Introduction to the Parallel Sysplex

This chapter provides an overview of the Parallel Sysplex and includes brief descriptions of the objectives and benefits of a Parallel Sysplex, its hardware components, including the Coupling Facility hardware itself, its software components, including XCF and XES, and the sysplex services provided by these components. The descriptions in this chapter, or even in this book, are by no means comprehensive, and at times the description of the sysplex services are abbreviated for ease of understanding. Some of the services available to sysplex users are not described at all if they do not play a role in the IMSplex.

Throughout the description of the Parallel Sysplex services, we have tried to use IMS's implementation of Parallel Sysplex services as examples.

Archived

© Copyright IBM Corp. 2003. All rights reserved.

1.1 Introduction

IMS/ESA® Version 5 was the first release of IMS to exploit the Parallel Sysplex. Version 5 dramatically improved the way block level data sharing was implemented by taking advantage of Parallel Sysplex services for communications and data sharing offered by the OS/390 components cross-system coupling facility (XCF), cross-system extended services (XES), and a new hardware component called the Coupling Facility (CF). This hardware is not to be confused with XCF, which is software.

Since that time, each release of IMS has more fully exploited Parallel Sysplex services. There is every indication that future releases of IMS will add even more Parallel Sysplex exploitation to enable a more dynamic, available, manageable, scalable, and well performing environment for database, transaction, and systems management. To be able to take advantage of these functions, and any future enhancements, it is important that you have at least a cursory understanding of what a Parallel Sysplex is, how it works, and why it is important to IMS and to your installation.

While this simply a very high level overview of the Parallel Sysplex, more detailed information is provided in a series of IBM publications on the Parallel Sysplex and Parallel Sysplex services. These publications can be found by searching for the term **sysplex** on the IBM publications center Web site:

http://ehone.ibm.com/public/applications/publications/cgibin/pbi.cgi

1.2 What is a Parallel Sysplex?

In 1990, IBM announced the *sysplex* as the strategic direction for large systems computing environments and described it as "... *a collection of MVS™ systems that cooperate, using certain hardware and software products, to process work*." The term sysplex is derived from the words ***SYS***tems com***PLEX***. At the time of this first announcement, and for several years thereafter, there was no Parallel Sysplex - only a *base sysplex* - which provided improvements in inter-processor communications between systems and any subsystems wishing to exploit those services, but no data sharing services.

The Parallel Sysplex was introduced later in the 1990s and added hardware and software components to provide for *sysplex data sharing*. In this context, data sharing means the ability for sysplex member systems and subsystems to store data into, and retrieve data from, a common area. *IMS data sharing* is a function of IMS subsystems which exploits sysplex data sharing services to share IMS databases. Since IMS did not exploit the base sysplex, from this point on we will use the term sysplex as shorthand notation for a Parallel Sysplex.

In this series of Redbooks, the term sysplex, when used alone, refers to a Parallel Sysplex. When the sysplex is not a Parallel Sysplex, and the distinction is important, we use the term base sysplex. We use the term IMSplex to refer to one or more IMSs cooperating in a single data sharing environment.

1.2.1 Some components and terminology of a Parallel Sysplex

Parallel Sysplex implementation requires a number of hardware, software, and microcode components, and has its own terminology. Here are some of the more visible components of a Parallel Sysplex, and some of the Parallel Sysplex terminology you will encounter.

- Central processing complex (CPC)

  A typical sysplex includes multiple CPCs (sometimes known as a CEC, or Central Electronic Complex), although it is possible to have a single-CPC sysplex. These include

selected models of the ES/9000®, S/390® and z900 processors. For the sake of brevity, in these redbooks we use the term S/390 to refer to any processor which is sysplex capable. As well, we use the term OS/390 to refer to an MVS/ESA™, OS/390, or z/OS™ operating system.

- Common time reference (sysplex timer)

  Because the OS/390 systems and subsystems running in the sysplex cooperate with other instances of the same subsystem, or with other subsystems and programs, it is critical that they all share a common time of day. For example, two IMS systems sharing data include the time that data is updated in their respective but separate logs. If those times were not synchronized, database recovery using these times would be difficult. All CPCs in a sysplex are connected to a sysplex timer (IBM 9037) which provides this synchronization across all CPC time-of-day (TOD) clocks. The sysplex extended TOD architecture also guarantees that two applications requesting a TOD value from any CPC in the sysplex will each receive unique values — also important for IMS data sharing recovery.

- Signaling paths

  Many of the services provided by the sysplex require communications between participating OS/390 systems and subsystems. To do this, at least two signaling paths are required between the CPCs in the sysplex - one inbound path and one outbound path. These paths may be defined using one or more of the following:

  – ESCON® channels operating in CTC mode
  – The 3088 Multisystem Channel Communication Unit
  – Coupling Facility with list structures

  Structures will be the subject of much discussion later in this document. The signaling structures are a Parallel Sysplex implementation requirement, not specific to IMS, and so no more is said about the signaling structures themselves.

- Coupling Facility (CF)

  The Coupling Facility hardware, not to be confused with cross-system coupling facility (XCF) software, is itself a processor which provides the medium on which shared data may be stored and retrieved within a Parallel Sysplex. The CF processor itself can be a logical partition (LPAR) in a S/390 processor but without the channels (and peripherals) which are part of the LPARs running your OS/390 operating systems.

  There are two basic types of Coupling Facility - *internal* and *stand-alone*. The stand-alone CF is a 9674 processor and is external to the CPC. During the last few years, most CPCs have been shipped with internal CFs which need only be activated to be used. A z900 box, for example, may come with multiple general use CPUs, plus additional CPUs which can be configured and activated only as Coupling Facility LPARs. It is also possible to configure one or more S/390 LPARs as a Coupling Facility. From an end-user (for example, IMS) functional point of view, there is no difference between the different types of Coupling Facility.

- Coupling Facility Control Code (CFCC)

  The Coupling Facility equivalent of an operating system is the Coupling Facility Control Code (CFCC) which is implemented only as microcode in the CF itself and runs only in LPAR mode. No software runs in the CF processor. Different releases of the CFCC are referred to as levels. For example, the latest CFCC level (at the time of this writing) is CFCC Level 12. Some data sharing functions have minimum CFCC level requirements, similar to some OS/390 or IMS functions that have minimum version and release level requirements.

- CPC-to-CF links

  The CF link is the functional equivalent of a CPC channel connecting the CPC to its peripheral devices, such as DASD. It provides the physical path for sending data to, and

receiving data from, the Coupling Facility. Sometimes you will see a physical link referred to as a *path* or a *coupling facility channel.* RMF™, for example, may report *path busy* conditions.

- Coupling Facility link adapters

  CF link adapters are microcode on either end of the CF link which provide the functionality to send and receive messages (data) between the CPC and the CF. CF link adapters have buffers into which data is place for sending. These buffers will often be referred to as *subchannels*. For example, some of the RMF reports may show *subchannel busy* conditions. This means that the buffers were full at the time a request was made and the requestor had to wait.

- CF-to-CF links

  Recent versions of the sysplex configuration may include another type of link called a CF-to-CF link. These links are used by some sysplex data sharing services, such as structure duplexing, to synchronize updates.

- Couple data sets (CDS)

  Although all CPCs have DASD containing many system data sets, there is one group of data sets required to define the sysplex and its services. These data sets are called *couple data sets* (CDSs). A base sysplex has only a single CDS (with an optional alternate for availability) called the sysplex couple data set. The sysplex CDS defines the sysplex and contains (for example) information related to the members of the sysplex and members of specific groups within the sysplex (called XCF groups). An IMS data sharing group is one example of an XCF group.

  A Parallel Sysplex must have a sysplex CDS but will also have as many as five additional CDSs, each with a specific function, or purpose. These are:

  - CFRM (coupling facility resource management) CDS
  - ARM (automatic restart management) CDS
  - SFM (sysplex failure management) CDS
  - WLM (Workload Manager) CDS
  - LOGR (System Logger) CDS

  Each of these CDSs contain user-defined *policies* which contain rules to tailor your Parallel Sysplex to your installation's unique requirements.

- Cross-system coupling facility (XCF)

  In 1990, MVS was enhanced to include a set of services provided by system software called cross-system coupling facility (XCF) services. XCF has since been a basic part of all supported MVS, OS/390 and z/OS systems ever since the base sysplex was made available and provides sysplex services to OS/390 and to subsystems such as IMS. Examples of such services include:

  - Group services
  - Signalling services
  - Monitoring services
  - Sysplex services for recovery

  These will be described in more detail later, but they all basically deal with communications between OS/390 components and/or subsystem with the sysplex.

- Cross-system extended services (XES)

  With the Parallel Sysplex came another set of OS/390 services known as cross-system extended services. XES provides OS/390 components and subsystems with sysplex services for data sharing — a comprehensive set of services to access shared data in the Coupling Facility structures. Examples, which will be discussed in a little more detail later, include:

- Connection services
- Lock services
- Cache services
- List services

- CF structures

  Within the Coupling Facility are blocks of storage referred to as structures. These structures contain all of the user-related data that is to be shared across the sysplex. These will be discussed in more detail later, but there are three types of structures available to the user, each offering a different set of functions:

  - Cache structures
  - Lock structures
  - List structures

  A full-blown IMSplex uses all three types of structures to store information about shared data, shared message queues, and IMSplex resources. Defining, sizing, and managing these structures is a critical part of implementing and managing the IMSplex.

- Hardware system area (HSA)

  Some sysplex services allocate and use storage within each CPC to signal the occurrence of certain events within the Coupling Facility. This storage is allocated as bit vectors in the hardware system area (HSA) by connectors to cache and list structures, with each bit signaling the occurrence of an event associated with that structure. There are two types of bit vectors:

  - *Local cache vectors* signal that a database buffer has been invalidated.
  - *List notification vectors* signal that a message has arrived on a list structure.

  IMS uses these bit vectors for data sharing and for shared queues.

- Policies

  Finally there are the sysplex *policies* mentioned above. These policies are in the various Parallel Sysplex CDSs and describe how the sysplex is to manage certain resources, conditions, or events. They are typically created by the sysplex administrator or systems programmer and are tailored to the needs of the installation. For example, an automatic restart management (ARM) policy in the ARM CDS describes the actions ARM (XCF's sysplex service for recovery) should take when a registered subsystem fails, or if the OS/390 system on which that subsystem is running fails. The coupling facility resource management (CFRM) policy in the CFRM CDS defines the structures which may be allocated in the Coupling Facilities. There are other policies for Workload Manager, System Logger, and System Failure Management, each in their own CDS.

1.2.2 OS/390 component and subsystem software

To take advantage of the XCF and XES services available in a Parallel Sysplex, OS/390 components and subsystems such as IMS must explicitly invoke these services. The number of sysplex exploiters has grown dramatically since it was first announced, and now includes, among many others:

- IMS, IRLM, CQS, IMS Connect, WebSphere MQ, DB2®, CICS® T/S,...
- XCF (uses XES services), JES2, RACF®, VSAM, VTAM®, System Logger,...

There are surely many more and each new product provided by IBM, as well as other software vendors, may take advantage of these sysplex services. How to use them is documented in a series of IBM publications. Examples include:

- *z/OS MVS Programming: Sysplex Services Guide*, SA22-7617
- *z/OS MVS Programming: Sysplex Services Reference*, SA22-7618

- *z/OS MVS Setting Up a Sysplex*, SA22-7625

Figure 1-1 shows some of the major components of a possible Parallel Sysplex configuration.

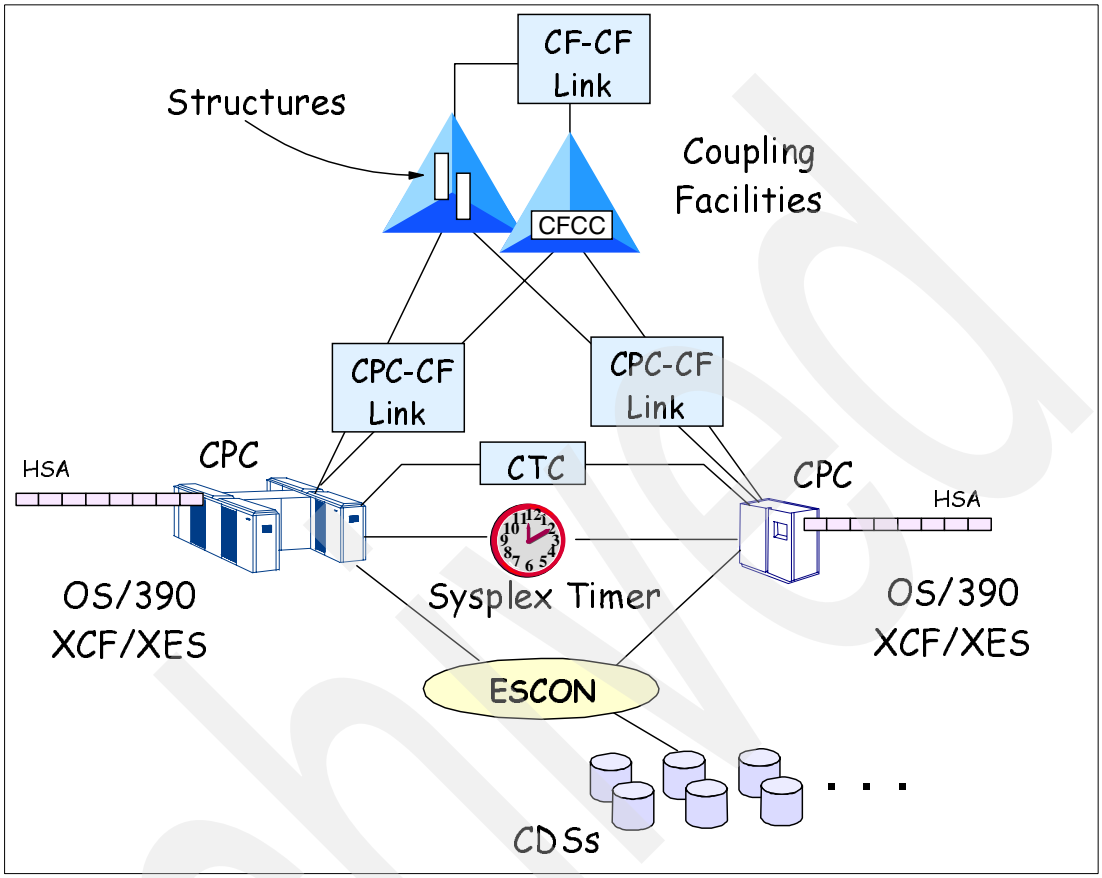


Figure 1-1 Parallel Sysplex configuration

1.3 Sysplex services

This section provides an overview of the sysplex services provided by XCF and XES, and identifies where IMS invokes these services. As always, a more complete and detailed explanation can be found in the IBM sysplex documentation, but many of the terms and concepts we discuss here, you will hear in discussions about the Parallel Sysplex with IMS. The following sections in this chapter should give you a good understanding of how IMS exploits these services, and help you in the planning, implementation, and operational phases of an IMS migration to the Parallel Sysplex environment.

We briefly discuss three types of services:

- Sysplex services for communications
- Sysplex services for recovery
- Sysplex services for data sharing

1.4 Sysplex services for communications (XCF)

Cross-system coupling facility (XCF) offers a set of communication services which allows multiple programs or subsystems, running on different systems in a sysplex, to share status

information and to communicate with each other. XCF communications services are available only between members of an XCF group. Before explaining what each of these services offers, it is first necessary to describe the concept of an XCF group.

1.4.1 XCF groups

According to *z/OS V1R3.0 MVS Sysplex Services Guide,* SA22-7617: "*A group is a set of related members defined to XCF by a multisystem application in which members of the group can communicate (send and receive data) between MVS systems with other members of the same group.*"

There are many XCF groups in a Parallel Sysplex, and IMS and IMSplex-related programs are members of many XCF groups. Groups are dynamically created when the first member joins the group. For example, when the IRLMs are initiated, they join an XCF group using the name specified in the IRLMGRP= parameter. The group is dynamically created by XCF when the first member joins. Information about the group is maintained in the sysplex couple data set. We refer to this as a data sharing group, but it is also an XCF group. Being a member of a group entitles that member to use XCF signaling and monitoring services.

There are many other groups in an IMSplex. Examples are:

- Data sharing group (IRLMs)
- Shared queues group (CQSs)
- OTMA group (IMSs and IMS Connect)
- Fast Database Recovery group (1 IMS and 1 FDBR)
- VTAM generic resource group (IMSs)
- IMSplex group (multiple address spaces running with a Common Service Layer)

In addition to these explicitly joined groups, each connector to a CF structure becomes a member of another group associated just with that structure. For example, each IRLM is not only a member of a data sharing group, but is also a member of a group associated with the IRLM lock structure. Being a members of the lock structure group allows XES to inform the all members of changes in structure status, such as a structure failure.

1.4.2 XCF group services

Group services allow authorized programs to join (create, if necessary) and leave XCF groups, and to be informed when other members join or leave the group. By belonging to a group, the member is authorized to invoke other XCF services, such as signaling and monitoring services. When joining a group, the member can identify a *group user routine*. When a new member joins the group, or an existing member leaves the group, this routine in each remaining member is driven by XCF to notify that member of the change in group status. For example, IMS and Fast Database Recovery (FDBR) join an XCF group using the name specified in the GROUPNAME parameter. If IMS fails, XCF drives FDBR's group user routine to inform FDBR that the IMS system it was monitoring failed so that it can back out in-flight database updates and release the locks held by the failed system. If an IRLM fails, other IRLMs are notified through their group user routine that one has failed and each can then take action to read the retained locks held by the failed IRLM.

1.4.3 XCF signaling services

Signaling services provide the means for members of a group to send messages to, receive messages from, and respond to messages received from, other *members of the same XCF group*. XCF macros are provided to communicate with other group members. The XCF macros invoke user message routines to send messages, receive messages, and respond to received messages. It is not necessary for any member to know which OS/390 system any

other member is on, in order to send a message. No special sessions are established between the group members. For example, IMS Connect and IMS both join an OTMA (XCF) group. When IMS Connect wants to send a message to IMS OTMA, it uses XCF to send the message without caring where that IMS is located. OTMA's message user routine is then driven to inform IMS that a message has arrived from IMS Connect. IMS sends the response to IMS Connect using the same techniques.

1.4.4 XCF monitoring services

Status monitoring services allow members to monitor the operational status of other members of the same XCF group. When a member joins a group, it identifies to XCF a *group user routine, a status user routine, a 32*-byte *member state field, and a status checking interval*. That member must update this status field periodically (and within the specified status checking interval). XCF monitors this status field based on the specified status checking interval and, if not updated within the specified interval, drives the status user routine to determine the status of the member. If the status user routine indicates that the member's status has changed, or if the status user routine just doesn't respond, then XCF drives the group user routines of the remaining members to inform them of the changed status of the member. Although there are five different states that can exist for a member, we will consider only two - ACTIVE and FAILED.

An example of this is the IRLMs in a data sharing group. Each IRLM updates a status field in CSA indicating that it is still alive and kicking (active). If one IRLM fails, XCF will detect that the IRLM has stopped updating its status field and (try to) drive that IRLMs status user routine. If the IRLM has truly failed, then the status user routine will not respond and XCF will notify the other IRLMs of the failure, at which time they will begin action to read the retained locks from the IRLM structure.

OS/390 also has a status field, but it is on the sysplex CDS instead of in CSA. If OS/390 fails, or the processor fails, then this status field will not be updated. XCFs running on other members of the sysplex monitor each other's status field and will detect the status missing condition and take action to isolate the failed system. XCF will then drive the group user routines of every group member on the surviving OS/390s that had a member on the failed OS/390 system. For example, if the OS/390 that IRLM1 was running on failed, all the other IRLMs would be informed.

1.5 Sysplex services for recovery (ARM)

A program such as IMS can enhance its recoverability by registering as a restart element to the automatic restart management (ARM) service of the Parallel Sysplex. ARM is activated in the sysplex by defining an ARM policy in the ARM CDS and then activating that policy. When a registered element fails, it will be restarted by OS/390 on the same logical partition (LPAR). When the OS/390 system itself fails, XCF running in other OS/390 systems detect this and can restart that element on another system (this is called a cross-system restart). Elements can be defined as part of a restart group. When a cross-system restart is required, all elements which are part of the same restart group will be restarted on the same LPAR. For example, if an IMS system is connected to a DB2 system, then it is important that both IMS and DB2 are restarted on the same LPAR. Another example is CICS and the corresponding DBCTL region.

1.6 Sysplex services for data sharing (XES)

Cross-system extended services (XES) is a set of sysplex services for data sharing which allows subsystems, system products, and authorized programs to use Coupling Facility structures for high performance, high availability data sharing. In this context, data sharing means the sharing of *any type of data* by programs running on any system in the sysplex. It does not mean data sharing only in the IMS (or DB2 or VSAM) sense of the term. IMS data sharing utilizes these sysplex services for data sharing to manage its own block level data sharing of IMS databases. Other examples of shared data used within the IMSplex include shared messages (IMS shared queues), shared log data (CQS and the System Logger), and shared information about IMSplex resources (IMS Resource Manager).

Shared data resides in one of three structure types in the Coupling Facility — cache structures, lock structures, and list structures. XES provides a set of services to connect to and define the attributes of these structures (connection services), and another set of services unique to each structure type (cache services, lock services, and list services). Connectors use these services to invoke the functions supported by that structure type.

The details of each structure type and the XES services supporting them is documented in the z/OS library of manuals — one good reference is *z/OS V1R3.0 MVS Sysplex Services Guide,* SA22-7617. The following is a very brief overview of the structure types, their components, what they are used for (by IMS), and what services are provided by XES.

1.6.1 Structures, connectors, and services

A *structure* is a Coupling Facility construct whose purpose is to support the sharing of data (for example, data in an IMS database, or messages in an IMS message queue) among multiple *connectors* to the structure through a set of *XES services* rather than require individual connectors to manage that storage directly. This not only simplifies and standardizes the access to the CF storage, but allows for enhancements to CF storage management functions without having to modify individual programs using that storage. All structures in a Parallel Sysplex must be defined in a Coupling Facility Resource Management (CFRM) *policy* in the CFRM couple data set.

There are three types of structures that can be allocated in a Coupling Facility - *cache structures, lock structures, and list structures*. Each type is designed to support certain data sharing functions and has its own set of XES services. Programs such as IMS, which want to access a structure and invoke these XES services for data sharing, connect to it using XES connection services. After connecting to the structure, the program then accesses data in the structure using XES cache, lock, or list services.

1.6.2 Connectors and connection services

XES provides connection services for a program to connect to (and disconnect from) a structure, and to define the type and attributes of the structure. That program is called a *connector* and, when connected to a structure, is authorized to use other XES services appropriate to the type of structure to which it is connected. The first connector to a structure determines the structure attributes. Other users can connect but cannot change those attributes. Some of the attributes of a structure are:

- Structure information, such as type and structure persistence
  - *Structure types* are cache, lock, and list structures. Note that while structures must be pre-defined in the CFRM policy, the type of structure is not determined until the first user connects to the structure.

- A *persistent structure* is one which, when all connectors have disconnected from the structure, the structure remains allocated in the Coupling Facility. It is still there the next time a program connects to it. Examples of persistent IMS structures are the shared queue structures, the IRLM lock structure, and the resource structure. OSAM, VSAM, and shared VSO structures are not persistent.

- Connection information, such as connection persistence
  - A *persistent connection* is one which persists if the connector fails. A non-persistent structure will not be deleted if there are any active or failed persistent connections to that structure. An IMS example is a shared VSO structure. It has connection persistence but not structure persistence. If all IMSs terminate their connection to a shared VSO structure normally, then it is deleted. However, if one IMS fails, then its connection becomes a failed persistent connection and the structure will not be deleted even if there are no other active connectors.
- Structure modification information determines how a structure may be modified after it is allocated. For example, does the connector support structure rebuild, structure alter, structure duplexing, and system managed processes.

Also during connection to a cache or list structure, each connector provides information about the *local cache vector* (required for cache structures) or *list notification vector* (optional for list structures). These vectors, located within HSA, define a relationship between each connector and the structure and are used to signal the occurrence of some structure event, such as the invalidation of a data buffer, or the arrival of a message on a list structure. For more information about structure attributes, refer to *MVS Programming: Sysplex Services Reference Guide*, SA2-7618.

1.6.3 Cache structures and cache services

A cache structure and its related cache services provide connectors with data consistency and (optionally) high speed access to data. Data consistency ensures that if one connector updates a block of data, any other connector with that same block of data in its buffer pool will be notified that its copy is invalid and should be refreshed if it is re-used. This is called *buffer invalidation*. If the connector opts to store data in the structure, then any connector can read that data from the cache structure (at CF speeds) instead of reading if from DASD (at the much slower DASD speeds). Whether or not a cache structure contains data entries is determined by the first user to connect to the structure — that is, the one that sets the structure type and attributes. There are three types of cache structures that may be allocated in the Coupling Facility. These are:

- **Directory-only** cache structures do not contain any data. They have only directory entries which are used to register interest by a connector in a particular block of data.
- **Store-through** cache structures contain both directory entries and data entries.
- **Store-in** cache structures contain both directory entries and data entries.

The difference between store-in and store-through structures is explained later.

Local cache vector

For cache structures, each connector identifies the length of the local cache vector for that structure in HSA. Each bit in the local cache vector indicates the validity of the contents of one local buffer in one connector's buffer pool(s). IMS uses one cache structure for VSAM and another cache structure for OSAM, each with its own local cache vector. For shared DEDB VSO AREAs, there is one cache structure for each AREA, each with its own local cache vector. One bit in the local cache vectors is associated with each VSAM, OSAM, or shared VSO buffer. Figure 1-2 is an illustration of an IMS buffer pool and the association of

each buffer with a bit in the local cache vector. Note that all cache structures have local cache vectors with relationships to local buffers whether there are data entries in the structure or not.

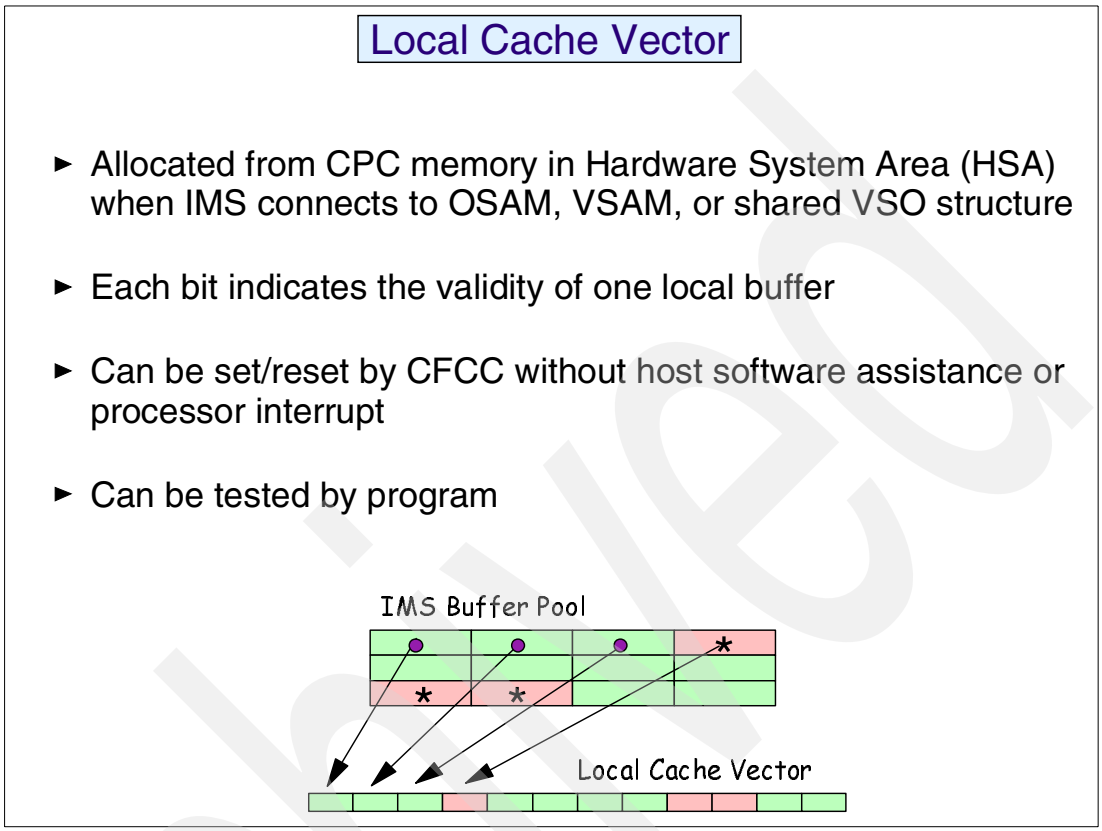


Figure 1-2 IMS local cache vector

Directory-only cache structures

Directory-only cache structures do not have data entries — only directory entries. *Each directory entry identifies a block of data* which is in a connector's buffer pool. For example, a block of IMS data is identified by its global DMB number (determined by DBRC), its data set ID (which data set within the database), and the RBA (relative byte address) within the data set. This is sufficient to uniquely identify that OSAM block or VSAM control interval (CI).

The directory also identifies which connectors (IMSs) have that block in their local buffer pool, and a *vector index* relating the directory entry to a buffer in the local buffer pool. When one connector updates a block of data on DASD, the CFCC uses the vector index in the directory entry to invalidate that buffer in other connectors having the same block in their buffer pools. Note that all cache structures have directory entries, and all directory entries perform the same functions.

Figure 1-3 is an illustration of a directory only IMS cache structure (for example, the VSAM cache structure). In this example, Block-A is in IMSX's buffer pool in a buffer represented by vector-index-1 in IMSX's local cache vector. Block-A is also in IMSY's buffer pool in a buffer represented by vector-index-4 in IMSY's local cache vector. The bits in the local cache vector were assigned to the buffers by each IMS when it connected to the structure.

IMS uses this type of structure for all shared VSAM databases, and has the option to use it for OSAM.

Archived

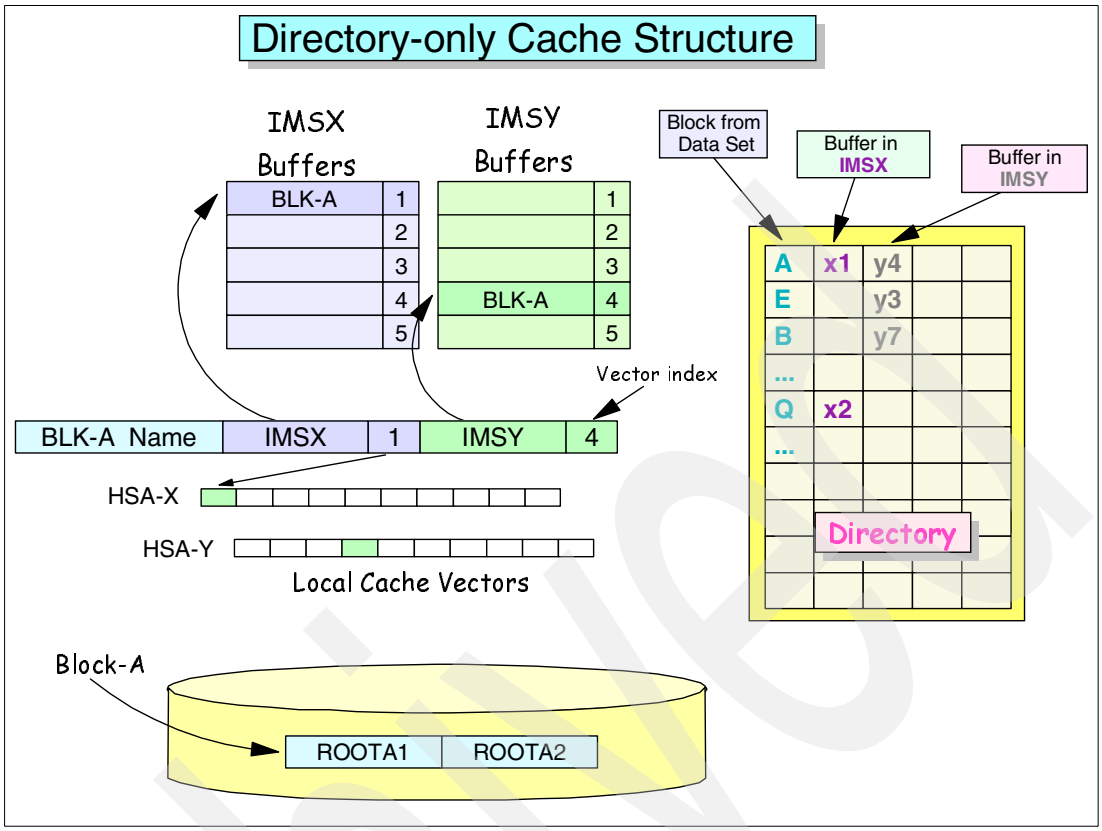


Figure 1-3 Directory-only cache structure

Store-through cache structures

A store-through cache structure contains both directory entries and data entries. The directory entry for the block indicates whether there is data for that block in the structure (not all directory entries have data in the structure) and where that data is in the structure. The data in the structure is considered *unchanged* data. That is, any updates to that block of data *already exist on DASD* before being written to the structure. IMS may use this type of structure for OSAM. The other option for OSAM is a directory-only cache structure. As a user of shared OSAM databases, you must choose between directory-only or store-through cache structures.

Figure 1-4 illustrates a store-through cache structure. Note that not all directory entries have data entries. This may happen when IMS reads a block of data into a buffer but does not write it to the structure. OSAM has an option, for example, to not write data to the OSAM cache structure unless it has been changed. If it is changed, OSAM will write it first to DASD and then to the structure.

Although there is only one OSAM caching structure, whether or not to cache data in that structure is an option set at the buffer pool level. When this type of structure is used, but not all OSAM data is cached, for blocks that are not cached there will be directory entries for the blocks in the buffers but no data entries.

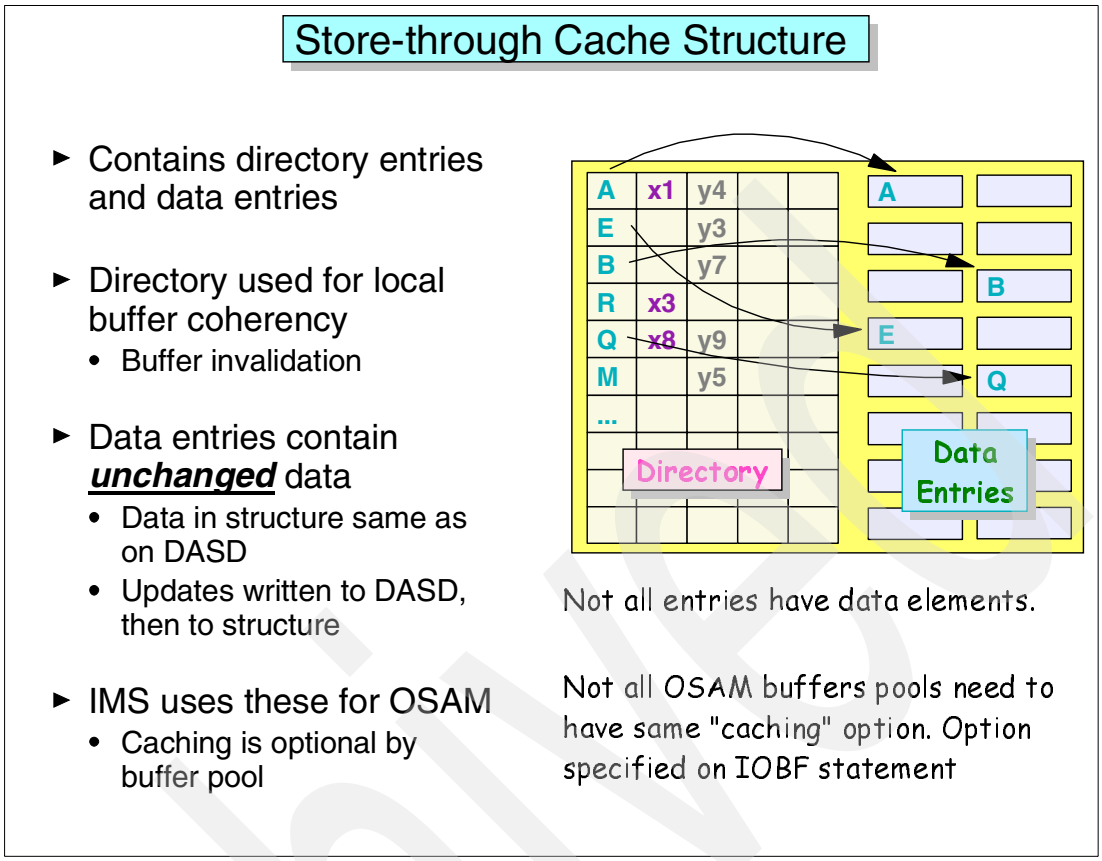


Figure 1-4 Store-through cache structure

Store-in cache structures

A store-in cache structure may contain *changed* data. Changed data is data which has been written to the structure but not yet written to DASD. When a connector writes a block of data to the structure, it indicates whether the data has been changed since it was last written to the structure. A flag in the directory entry indicates that the data entry contains changed data. Periodically, the connector requests a list of all changed data entries, reads them from the structure, and writes them out to DASD, turning off the changed flag in the directory entry. This process is called *cast-out processing*. IMS uses store-in cache structures for shared VSO AREAs. It invokes cast-out processing at every system checkpoint, and whenever the VSO AREA is closed.

IMS uses store-in cache structures only for shared VSO AREAs in a data entry database (DEDB). Store-in cache structures can offer a very significant performance benefit over store-through structures because there is no I/O. If a shared VSO CI is updated many times between two IMS system checkpoints, then with a store-through structure (such as, OSAM or VSAM), that block or CI is written to DASD each time it is committed — perhaps hundreds or thousands of times between system checkpoints. If it is in a store-in cache structure (such as, shared VSO), then it gets written to the structure each time it is committed, but only gets written to DASD once per system checkpoint, saving perhaps hundreds or thousands of DASD I/Os.

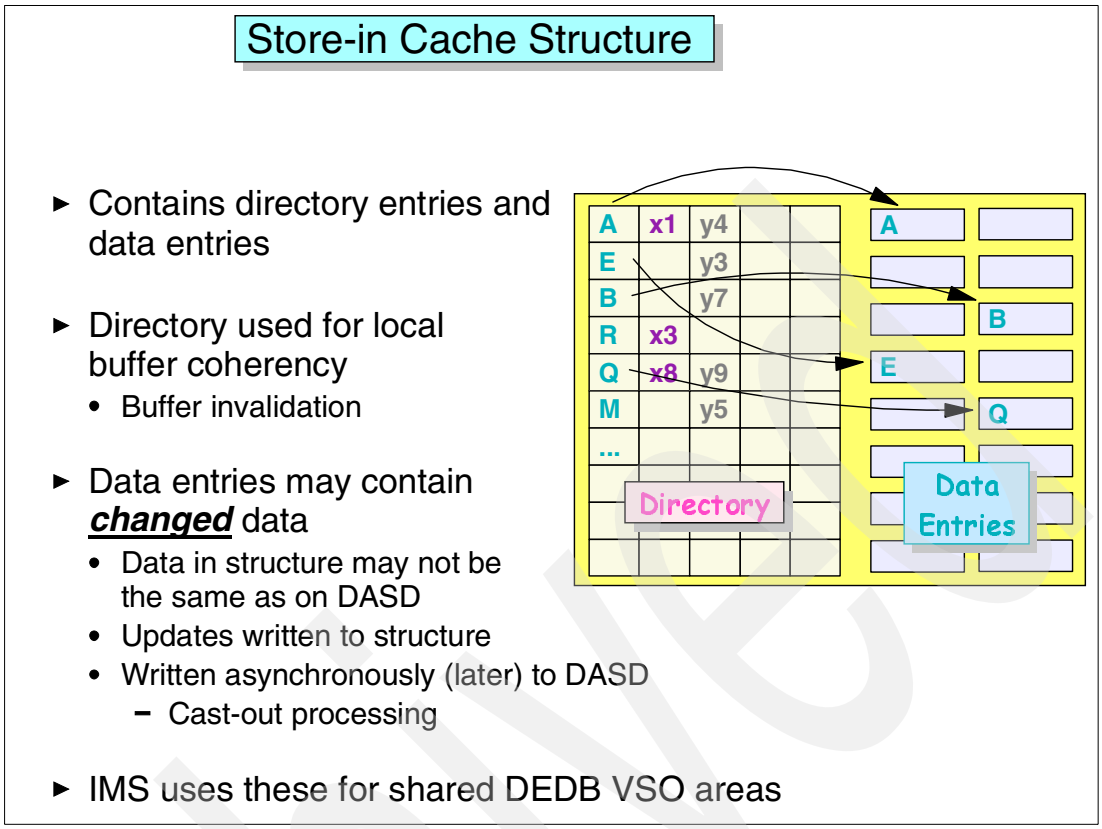


Figure 1-5 Store-in cache structure

Comparing store-through and store-in cache structures

Each type of cache structure has its advantages

- Directory only cache structures contain no data entries. They can be smaller and they do not have the overhead of writing data both to DASD and to the structure.
- Store-through cache structures contain data entries that represent unchanged data. They are larger than directory-only cache structures and there is the overhead of writing the updated data to the structure each time it is updated on DASD. Other connectors may benefit from the data written earlier by the same or another connector. If the structure fails, since the data is unchanged, there is no database recovery required.
- Store-in cache structures have the same size and overhead considerations as store-through structures. They have similar read advantages while allowing the connector to avoid much of the DASD I/O. However, if a store-in cache structure fails, then database recovery will be required (changed data was in the failed structure that is not on DASD). Because of this, Fast Path has the option of using dual structures for shared VSO. Beginning with z/OS 1.2 and CF Level 11, structure duplexing is supported which may be an alternative to Fast Path maintaining dual structures.

XES cache services

Connectors to cache structures invoke connection services to connect and define the type of structure and its attributes, but then use cache services to make use of the structure and its functions. Connectors request cache services using the IXLCACHE macro and specifying a specific function on the request. We talk briefly about four cache service request types.

Archive

READ_DATA request

Before IMS reads a block of data into its buffer pool, it must first tell the Coupling Facility which block of data it will read and which buffer it will put that data in. This is called *registering interest* in a data item and is necessary so that, if another IMS updates that block, the other IMS's buffer can be invalidated. This XES cache service is the READ_DATA request. It causes a directory entry to be created (or updated, if it already exists). If an existing directory entry points to a data entry in the structure when the READ_DATA request is made, that block will be automatically read from the CF into the buffer identified on the request without further action by the connector. For directory-only cache structures, of course, there is never a data entry in the structure, but the READ_DATA request is still issued to register interest.

WRITE_DATA request

IMS issues a WRITE_DATA request to write a block of data to a store-through or store-in cache structure. If IMS has updated that block of data, then the request will contain an indicator that the data has been changed. If any other IMS has registered interest in the data item, its buffer will be invalidated by flipping a bit in the local cache vector. Which bit to flip is determined by looking at the directory entry. A signal is then sent to the processor to flip that bit in the local cache vector, indicating the buffer is invalid. The IMS which had its buffer invalidated is removed from the directory entry. Note that this IMS does not know its buffer has been invalidated until it tests the bit setting in the local cache vector on a refetch (explained below).

CROSS_INVALidate request

When IMS updates a block of data on DASD and does not write it to the structure (for example, if it is a directory-only structure), then of course there is no WRITE_DATA request to invalidate the buffers. In this case, IMS will issue a CROSS_INVALidate request to tell the CFCC to invalidate any other IMS's buffer with registered interest in the data item.

TESTLOCALCACHE request

Before any IMS uses data in any buffer, it must first test to determine if the contents of that buffer are still valid. To do this, it must test the bit in the local cache vector to see if some other IMS has cross-invalidated it. This is a TESTLOCALCACHE request and must be issued before each access to the buffer. If a buffer is found to be invalid, then that IMS must re-register interest in the data item and, if not returned on the READ_DATA request, then reread it from DASD.

Figure 1-6 and Figure 1-7 are examples of IMS using cache services to read and update data.

Archive

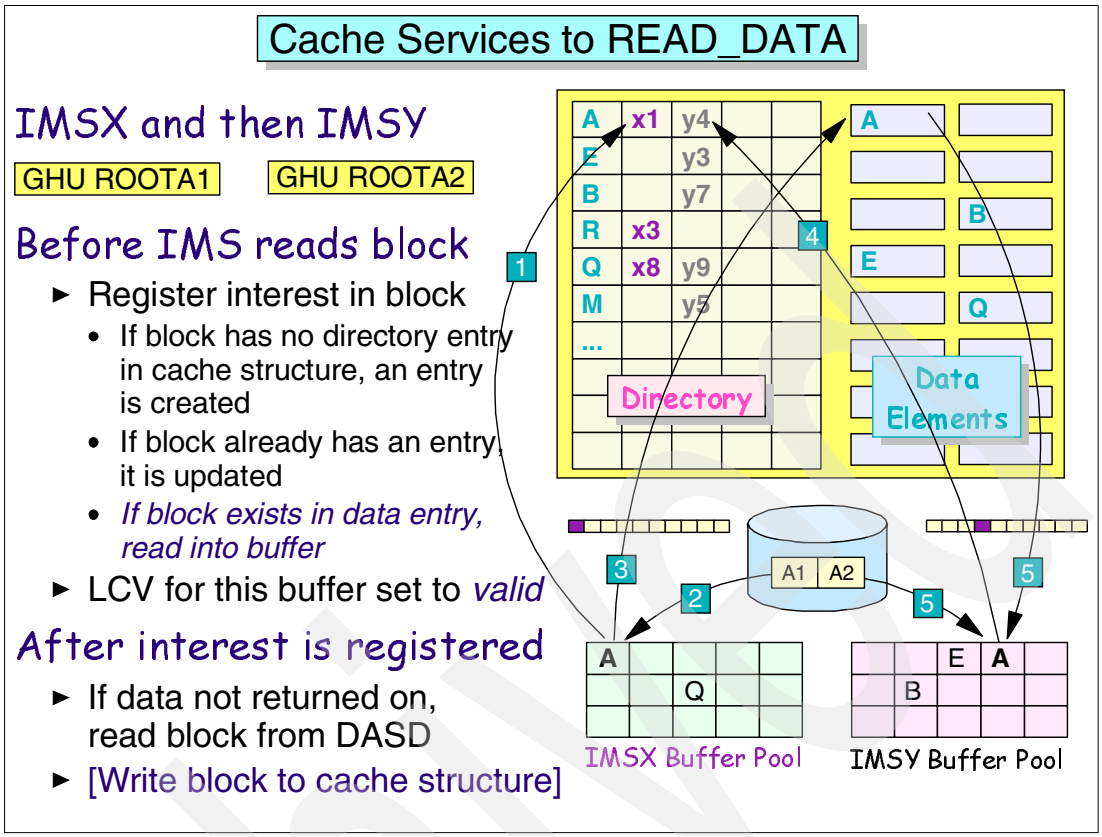


Figure 1-6 Example of a READ_DATA cache service request

1. A program in IMSX issues a get call to ROOTA1 which is in Block-A. IMSX issues a READ_DATA request to register interest in Block-A and provides the vector index into the local cache vector. A directory entry for Block-A is created in the cache structure.
2. IMSX reads Block-A from DASD.
3. IMSX writes Block-A to DASD using a WRITE_DATA request and indicating that the data is *not changed*.
4. A program in IMSY issues a call to ROOTA2, which is also in Block-A. IMSY must register interest in Block-A using a READ_DATA request. This will update the directory entry for Block-A to indicate IMSY's interest and its vector index.
5. If Block-A is in the structure when IMSY issues the READ_DATA request, it will be moved to IMSY's buffer. If it is not in the structure, then IMSY would read it from DASD.

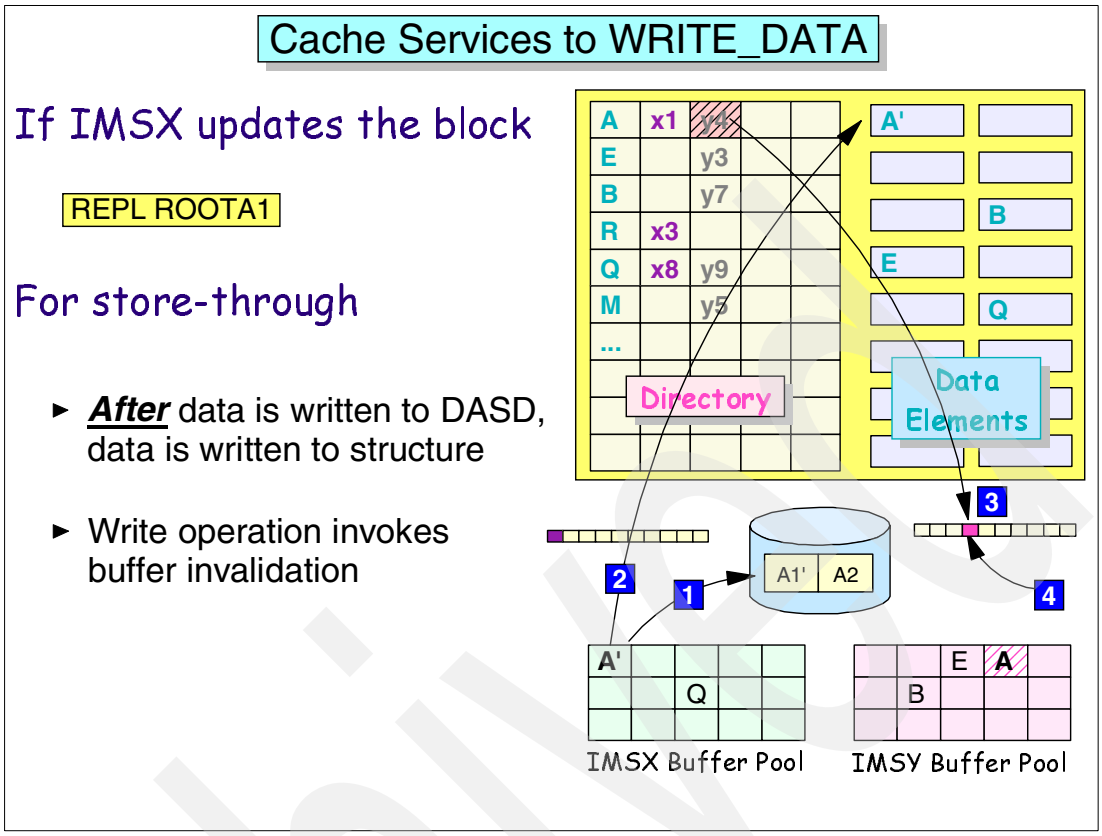


Figure 1-7 Example of a WRITE_DATA request to a store-through structure

1. If the program on IMSX updates ROOTA1, IMSX will first write it to DASD.
2. IMSX will then issue a WRITE_DATA request with the changed data indicator on.
3. The CFCC will send a signal to IMSY's processor to flip the bit pointed to by IMSY's vector index in the directory entry and remove IMSY's interest from the directory entry.
4. If IMSY needs to use the buffer again, it must test the validity of the buffer by issuing the TESTLOCALCACHE request. When it finds that it is invalid, it will re-register interest in Block-A. At this time, since Block-A (with the updated data) is in the structure, it will be read into IMSY's buffer.

If this were a store-in cache structure (shared VSO), the only difference would be that step-1 would be skipped. Updated blocks in the structure would be written to DASD later (at IMS system checkpoint time) using cast-out processing (another function of XES cache services).

Cache structure sizing considerations

It is important that the cache structure have at least enough space so that every buffer in every connected IMS can have a directory entry. If there are not enough directory entries to support all the buffers, then directory entries will be *reclaimed* when a new data item is registered and the associated buffer(s) will be invalidated.

When the cache structure also contains data, then enough space must be allocated to hold as much data as the user wants. Space for data entries is used just like space in a buffer pool - the least recently used data entry will be purged to make room for a new one. This is unavoidable, unless the database being cached is very small and will fit entirely within the cache structure, so like a buffer pool, the sizing of the structure will affect its performance.

CFSIZER, a tool for helping users size IMS structures, can be found on the Internet at:

http://www.ibm.com/servers/eserver/zseries/cfsizer

1.6.4 Lock structures and lock services

A lock structure can consist of two parts. The first part is a lock table — a series of lock entries which the system associates with resources. The second (optional) part is a set of record data entries containing a connected user's ownership interest in a particular resource and can be used for recovery if the connector fails.

IMS uses the IRLM to manage locks in a block level data sharing environment. The IRLM uses XES lock services to manage the locks globally. It is the IRLM, not IMS, that connects to the lock structure. In the context of the definition of the components of a lock structure, the connected user is an IRLM and the resource is an IMS database resource (for example, a record or a block).

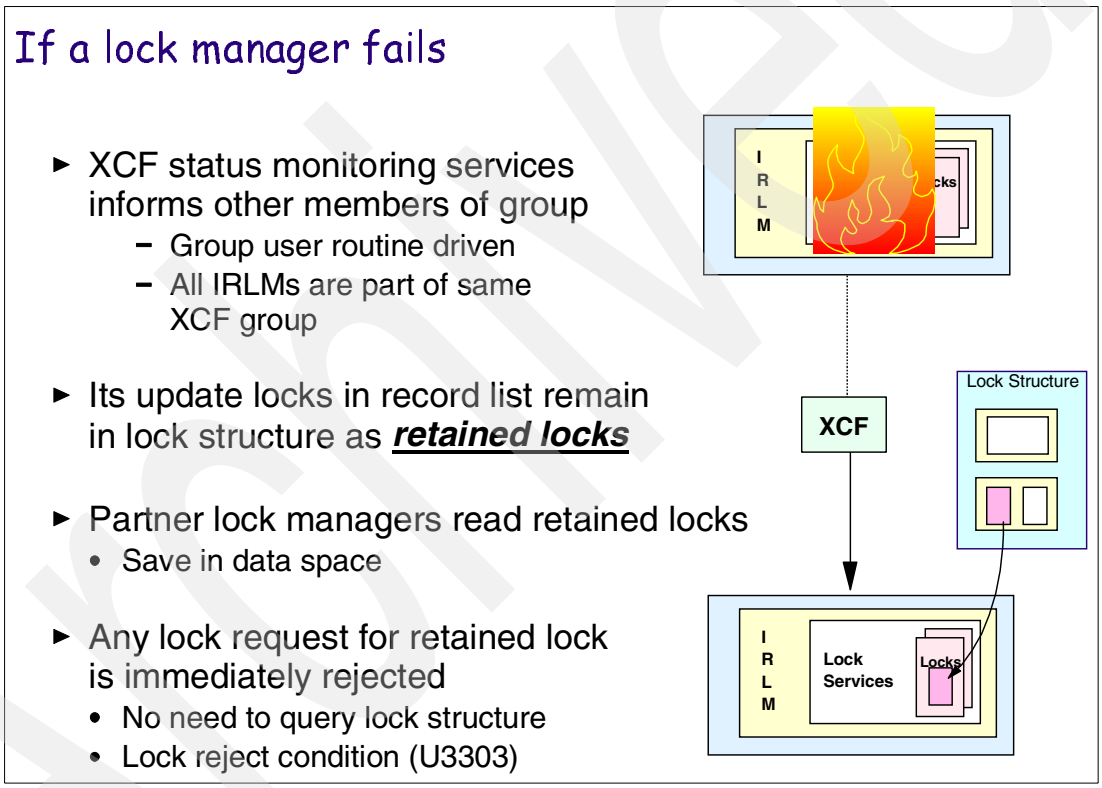


Figure 1-8 Lock table component of lock structure

The lock table consists of an array of lock table entries. The number and size of the lock table entries is determined by the first connector at connection time. Each lock table entry identifies connectors (IRLMs) with one or more clients (IMSs) that have an interest in a resource which has "hashed" to that entry in the lock table. The entry also identifies a *contention manager*, or *global lock manager*, for resources hashing to that entry. The purpose of the lock table is to provide an efficient way to tell whether or not there is a *potential contention* for the same resource among the data sharing systems. If there is no potential contention, a lock request can be granted immediately without the need to query other lock managers for each individual lock request. If the lock table indicates that there is potential contention, then additional work is needed to determine whether there is a conflict between the two requestors. This is the responsibility of the global lock manager.

In the example shown in Figure 1-9, a lock request issued by IMS system (let's say IMS3) to IRLM3 for RECORDA hashes to the third entry in the lock table (it has a hash value of 3). Interest is registered in that entry and IRLM3 is set as the *global lock manager* or *contention manager* for that entry. There has also been a lock request to IRLM1 for RECORDB. This is *potential lock contention* which must be resolved by the global lock manager. It is impossible to tell from the information in the lock table whether the two requests are compatible or not. It is the responsibility of the global lock manager to resolve this. Also in our example, an IRLM1 client has also requested a lock for RECORDC which hashed to entry 5 in the lock table. Since there is no other interest from any IRLM client for this entry, the lock can be granted immediately.

We will discuss how potential lock conflicts are resolved when we discuss XES lock services.

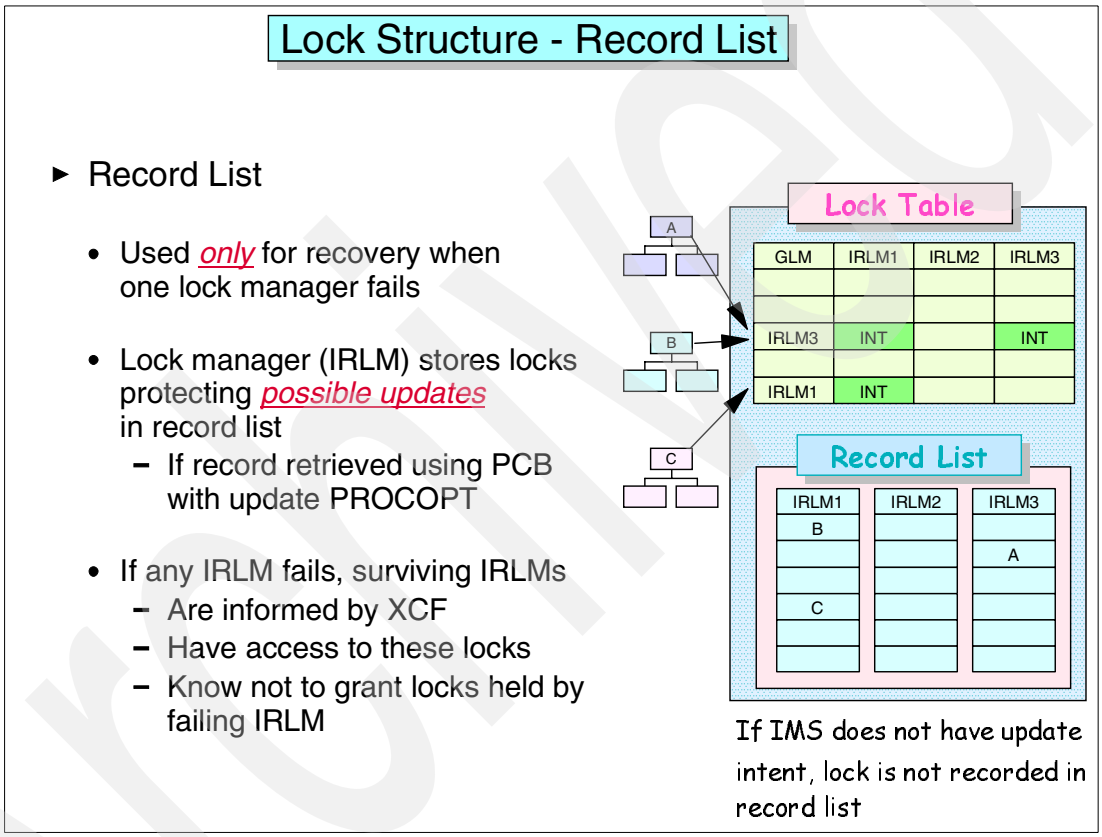


Figure 1-9 Record list component of lock structure

Figure 1-9 shows the record list component, or the list of record data entries showing the level of interest for specific resources. For the IMS data sharing environment, these entries represent record and block locks which IMS has acquired with *update intent*. It does not mean only those resources which IMS has updated, but those resources which IMS might have updated. If an IMS database record was retrieved with an update processing option (PROCOPT), then the lock for that resource goes into this record list. These record data entries in the record list are used only for recovery when a connector (IRLM) fails. They are not used for granting or rejecting locks at any other time. They are not used, for example, to resolve the potential conflict indicated by the lock table.

We will discuss how these are used when we discuss XES lock services.

XES lock services

The IRLM uses XES services to manage the global locks in an IMS data sharing environment. We talk about only two of these services, because these are necessary to understand the concepts of global data sharing.

Requesting ownership of a resource (OBTAIN)

When the IRLM wants to *obtain* a resource lock on behalf of one of its (IMS) clients, it issues an OBTAIN request (IXLLOCK REQUEST=OBTAIN). There are many parameters on this request, but the few that are significant to us are the resource name (what resource is being locked), the hash value (index into the lock table), the requested state (shared or exclusive), and the information to be put into the record data entry in the record list.

When the request is made, the hash value is used as an index into the lock table. The lock table entry will be updated to show interest by the requestor in this entry. If there is no other interest by any other lock manager, the lock table entry is updated with the ID of the requesting IRLM as the global lock manager and the requestor is informed that the lock may be granted. If there is other interest already in the lock table entry, then there might be a conflict with a lock held by some other requestor's client (by another IMS). The *contention exit* of the global lock manager already assigned to this entry is driven to resolve the potential conflict. If there is none, then the lock is granted; if the request is for the same resource, then the requestor must wait until the holder releases the lock.

Releasing ownership of a resource (RELEASE)

When the holder of a lock (IMS) is ready to release it, it issues the IXLLOCK REQUEST=RELEASE macro with the resource name, the hash value, and an indication to delete the record data. At this time, if there were waiters for the resource, the next one on the wait queue gets the lock and processing continues.

Global lock manager

One of the IRLMs is chosen as the global lock manager, or contention manager, for each lock table entry with some interest. It is usually the first one to request a lock that hashes to that entry, but in some cases it could be another IRLM. When a lock request is hashed to an entry with existing interest, and therefore with an existing global lock manager, that lock manager's contention exit is driven to determine whether the lock request is for the same resource, and if so, are the locks compatible (for example, two share lock requests are compatible). The global local lock manager will keep information about each resource (for example, the resource name and the held lock state) in a data space. When the contention exit is driven, that IRLM can just look in the data space to resolve the possible contention. It will add the new requestor's lock request information in the data space and, if a third requestor comes along, the global lock manager can make a decision without having to query any other lock manager.

Tracing a lock request

The next few figures illustrate what happens when first one IMS, and then a second IMS, issues lock requests for different resources that hash to the same value in the lock table.

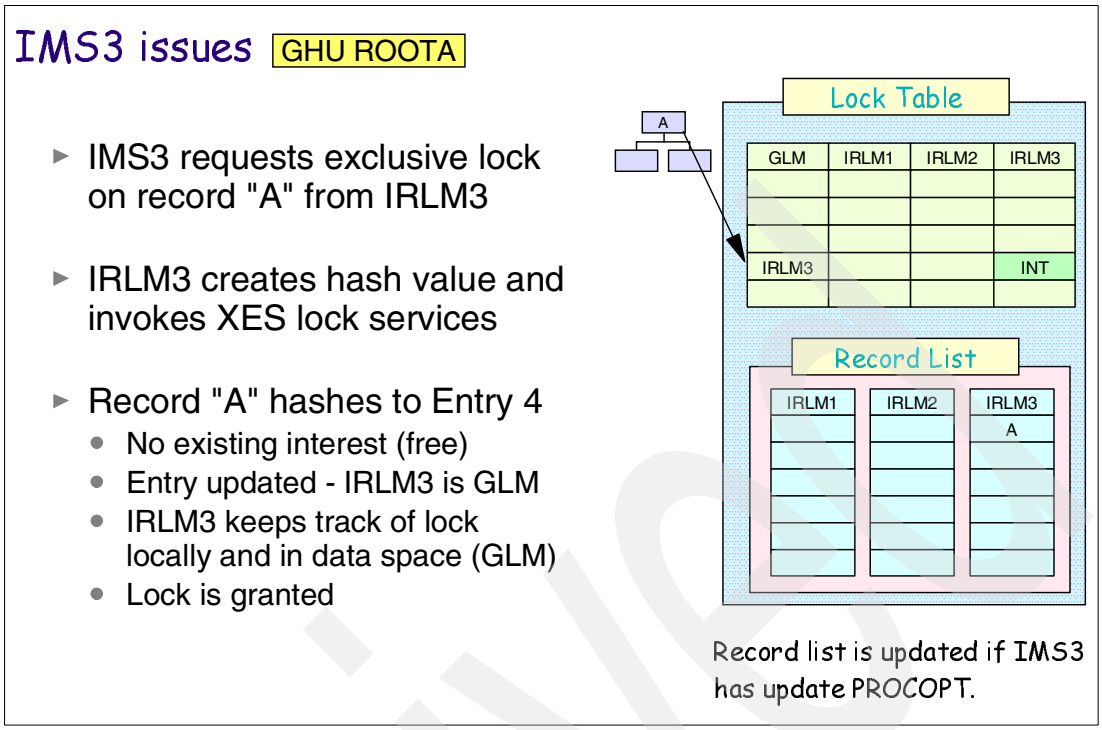


Figure 1-10 Requesting an exclusive lock - first requestor

In the example started in Figure 1-10, IMS3 has issued a lock request to IRLM3 for an exclusive lock on ROOTA. IRLM3 uses the resource name to create a hash value (HV=4) and then submits an XES lock services OBTAIN request. The hash table entry indexed by HV=4 shows no existing interest in any resource that would hash to this entry. The entry is updated to show interest by IRLM3 and sets IRLM3 as the global lock manager for this entry. Record data is added to the record list with information passed on the lock request, including the resource name and lock holder. The lock is granted.

Archive

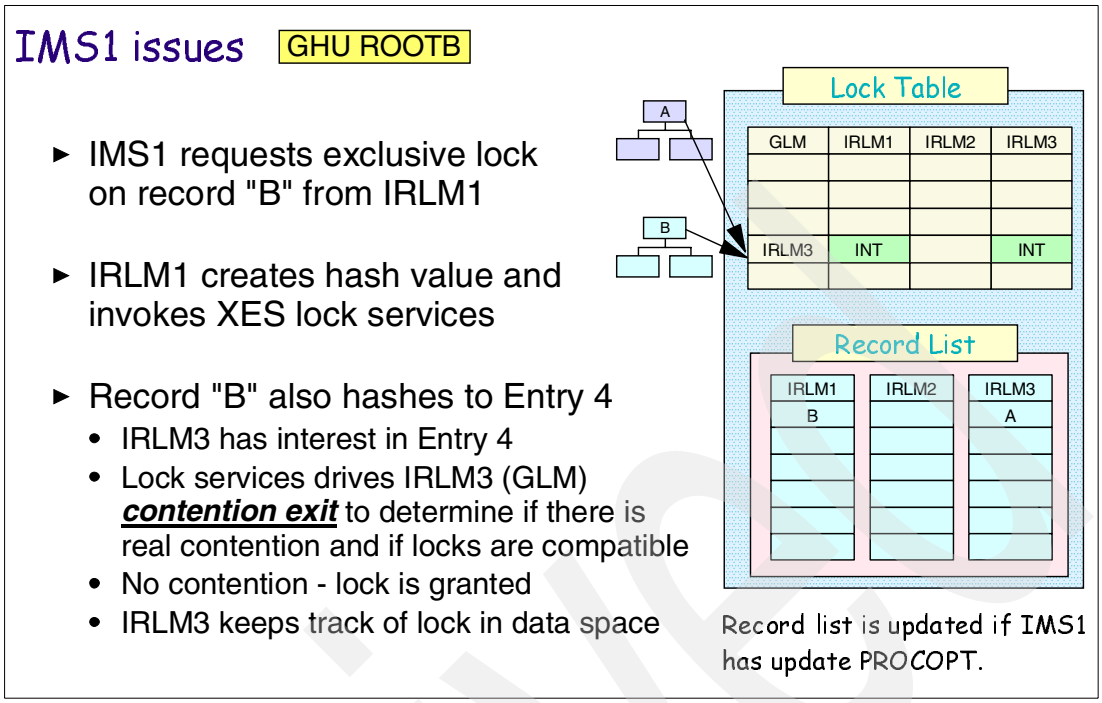


Figure 1-11 Requesting an exclusive lock - second requestor

Figure 1-11 continues the example. In this illustration, IMS1 has requested a lock on ROOTB. The resource name hashes to the same value as ROOTA (HV=4). When the lock request is issued to lock services for an exclusive lock on ROOTB, the lock table entry shows that there is already some interest in this entry, but we cannot tell from this whether it is for the same resource. IRLM3's contention exit is driven to resolve the potential conflict. IRLM3 has kept the information about the other lock in a data space and can determine immediately that the requests are for two different resources and the lock can be granted. IRLM1's interest is updated in the lock table entry, a record data entry is created, and IRLM3 keeps track of IRLM1's locked resource in a data space.

Handling an IRLM failure

If an IRLM fails, the locks that it held cannot be managed, and they won't be released until the IRLM is restarted. This may be immediately or it may be a long time. When an IRLM fails, other IRLMs which are part of the same data sharing XCF group are informed by XCF that one IRLM has failed. Each surviving IRLM reads from the lock structure the record data entries of the failed IRLM into a data space. These locks are considered retained locks and, if the IRLM gets a request for any of these locks, instead of invoking XES services, the IRLM will immediately reject the lock and application program will abend with a U3303.

Figure 1-12 illustrates what happens when an IRLM fails and surviving IRLMs are notified by XCF.

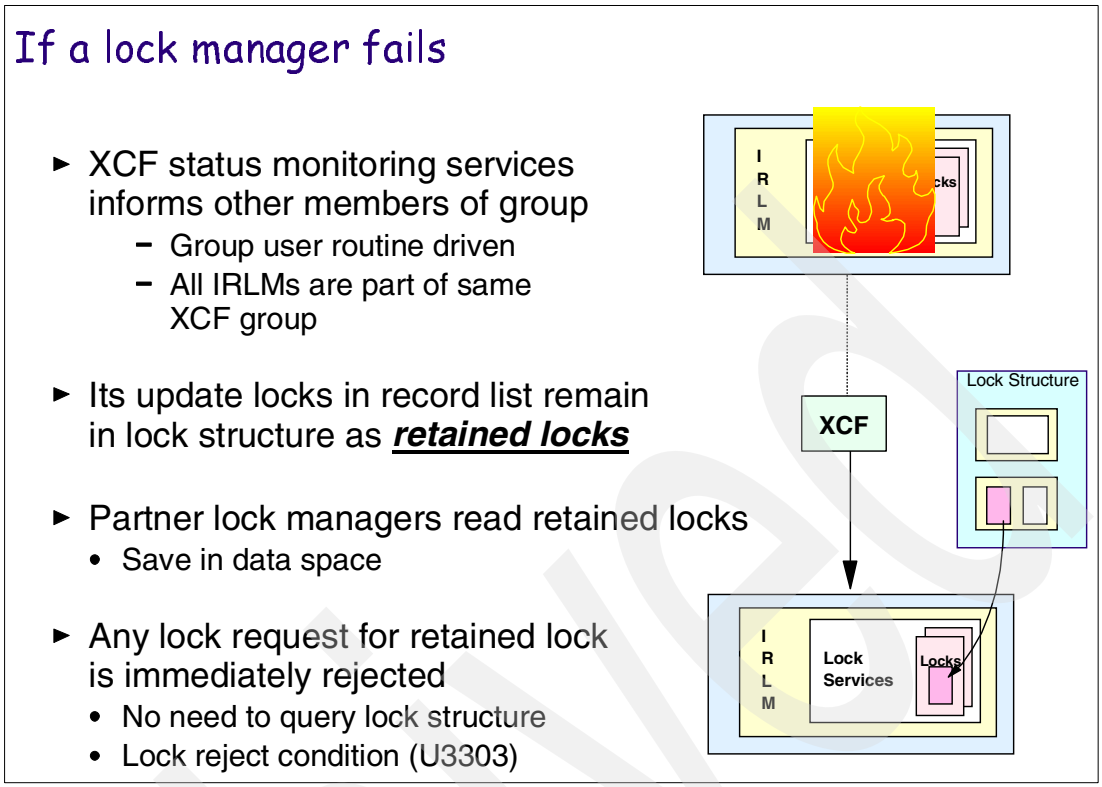


Figure 1-12 Managing IRLM failures

Lock structure sizing considerations

Locking services will work no matter how big the lock table is, but if there are a lot of false contentions, then there will be a lot of additional overhead to resolve the potential conflict. Ideally we would like to have no potential contentions, but since this is not really possible, we would at least like it to be a very, very small probability. Therefore, the size of the structure and the size of the lock table should be big enough to keep this probability low. A lock table entry may require two, four, or eight bytes, depending on how many IRLMs may be connected to it. A two-byte entry can handle up to six IRLMs. Assuming there are no more than six IRLMs in your data sharing group, a 32 MB structure with a 16 MB lock table will have 8 million entries. If there are seven to 23 IRLMs, then the entry size is four bytes and the 16 MB lock table will contain only four million entries. And 24 to 32 IRLMs will require 8-byte entries, producing only 2 million entries. The first connector sets the size of the lock table and the maximum number of connectors. It is important that the IRLM sets this at the lowest possible value.

Help with sizing the IRLM lock structure can be found at:

`http://www.ibm.com/servers/eserver/zseries/cfsizer`

1.6.5 List structures and list services

List services allow subsystems and authorized applications running in the same Parallel Sysplex to use a Coupling Facility to share data organized in a list structure. The list structure consists of a set of lists and an optional lock table. Information is stored on each list as a series of list entries. The list entries on one list may be kept in sorted order by key value. The lock table can be used to serialize on resources in the list structure, such as a particular list or list entry. A list structure with a lock table is called a serialized list structure.

While cache and lock services have very distinct purposes, the connector to a list structure may use it to keep *any kind of information* that needs to be shared. For example, in an IMSplex, list structures and list services are used for three functions:

- IMS uses the *resource list structure* for resource management. This structure is used to keep *status information* about the resources in the IMSplex, such as which nodes are logged on to which IMS, which users are in a conversation and what the conversation ID is, what the names are of the online change data sets, and other information related to the IMSplex.
- CQS uses up to four *shared message queue list structures* for shared message queues. These structures are used to keep *messages* (IMS full function and Fast Path EMH transactions and responses) that need to be shared among all the IMSs in a shared queues (XCF) group.
- CQS uses the System Logger to log activity to the shared queues structures. The System Logger uses one or two (full function and Fast Path) *logger list structures* to support a *shared logstream* among all the CQSs in the shared queues group.

Other users of list structures include XCF for signalling, JES2 checkpoint, VTAM Generic Resources and multinode persistent sessions, and just about anything that needs to be shared across the sysplex and which doesn't meet the specific requirements of cache or lock services.

Components of a list structure

The list structure has more components that either cache or lock structures. Figure 1-13 shows the major components of a list structure. Some of these components exist in every list structure, and some are optional. The first connector to a list structure determines its attributes and whether or not the structure will have these optional components.

Archived

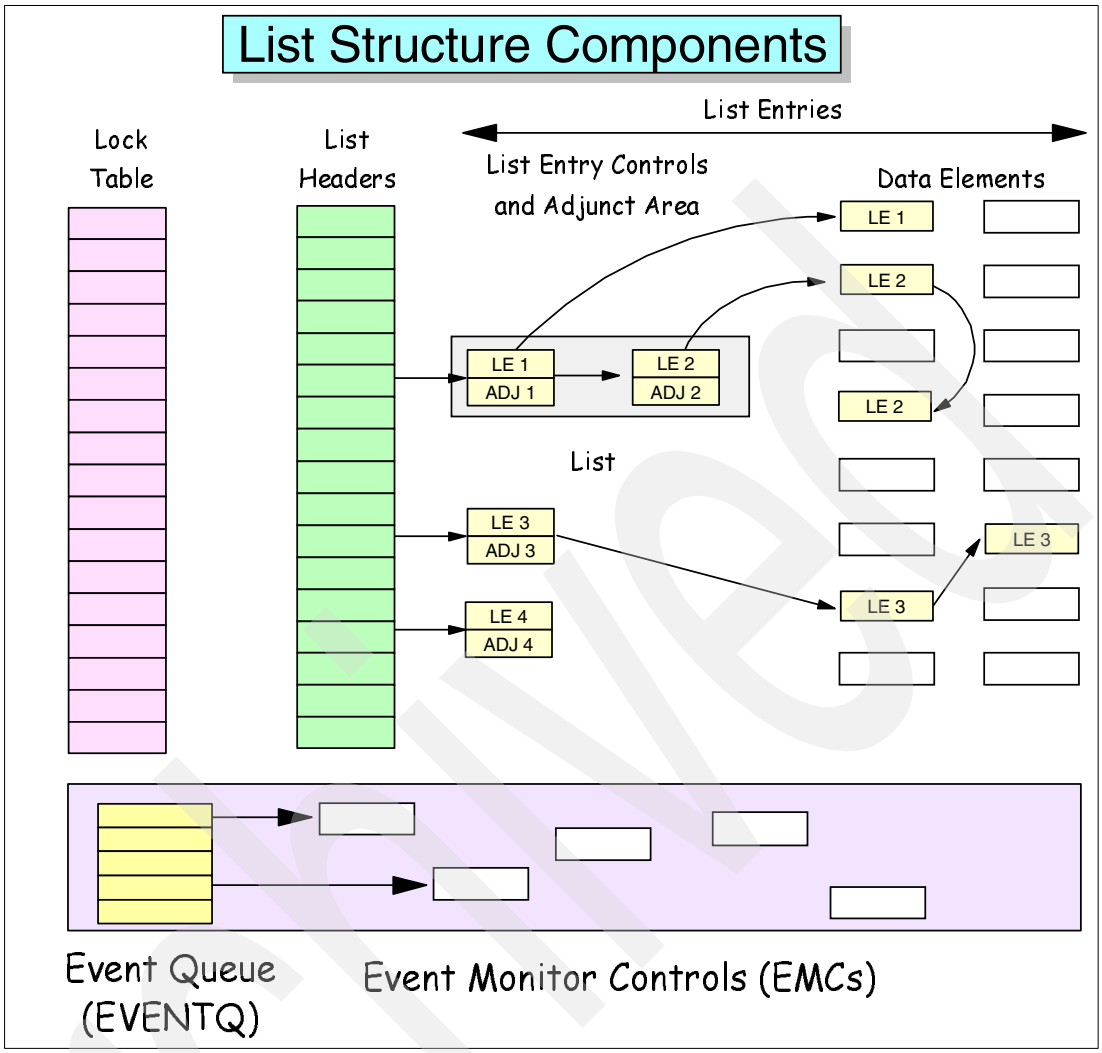


Figure 1-13 List structure components

The following describes these components and how they are used.

- List headers

  List headers are the anchor points for lists. The first connector to a list structure determines how many list headers the structure should have. Each list entry is chained off one list header. There is no preset meaning to a list header. The connector determines how the list headers are to be used. The Common Queue Server (CQS) for IMS shared queues allocates 192 list headers in a shared queue structure. It assigns some of them for transactions, some for messages to LTERMs, some for MSC links, and so on.

- List entries

  This is where the information is kept. A list entry can consist of one, two, or three components — list entry controls, an adjunct area, and a data entry. All list entries on a list header form a list. All list entries on a list header that have the same entry key form a sublist.

  – List entry controls

    Each list entry must have a list entry control (LEC) component. The LEC contains the identification information for the list entry, such as the list header number, the list entry key (ENTRYKEY), the list entry ID (LEID), and the list entry size (how many data

elements in the data entry, if any), and other attributes of the list entry. For CQS, the entry key includes the IMS destination, such as a transaction code or LTERM name. The data entry is the IMS message itself.

- Adjunct area

  Each list entry can have an optional adjunct area. Whether or not there are adjunct areas is determined by the first connector. If they exist, then every list entry has one. Each adjunct area is 64 bytes and can contain whatever information the connector wants. CQS allocates the shared queue structures with adjunct areas. The adjunct area is used by CQS for internal controls.

- Data entry and data elements

  The first connector determines whether the list structure is to be allocated with data elements, and the size of the data element. Each list entry may have a data entry associated with it. A data entry consists of one or more data elements, depending on how long the data is. For CQS, an IMS message is one data entry with one or more 512-byte data elements. There are also list entries with data entries for CQS control information.

- Lock table

  The first connector may optionally allocate a lock table. The lock table can be used to serialize a process, for example, to serialize access to a list. Access to an individual list entry is serialized by list services and the CFCC without having to serialize access to the list. CQS allocates a lock table and uses it to serialize access to its CONTROLQ (list header one).

- Event monitor area

  Space within the structure can be allocated for event monitoring. The first connector can allocate a percentage of the list structure to be used for an event queue (EVENTQ) and event monitor controls (EMCs). These are used to monitor events such as the arrival of a message on a particular list, or a message with a particular entry key (sub-list). When a monitored event occurs, the connector is informed. The event is that an event queue or a list has *transitioned* from empty (no entries) to non-empty (one or more entries). CQS uses *event queue monitoring*.

  - Event queue (EVENTQ)

    When event queue monitoring is requested by the connector, an EVENTQ is created in the structure with *one entry for each connector*.

  - Event monitor controls (EMCs)

    A connector indicates what it wants to monitor by *registering interest* in a list (list number) or a sublist (the entry key). When interest is registered, an EMC is created for that connector and list or sublist. IMS tells CQS which sublists to monitor (for example, a particular transaction code) and CQS registers interest in that sublist using XES list services. IMS registers interest in message destinations, such as transaction codes, LTERM names, and MSC links (MSNAMEs).

  It is not a requirement for a list structure to have an event monitor area. The resource structure in IMS V8, for example, does not invoke list or event queue monitoring and so does not have this part of the structure. When monitoring is not requested by the connector, it is the responsibility of the connector to determine what is on the structure - there is no notification of changes.

Event queue monitoring

When EVENTQ monitoring is requested by the connector, a *list notification vector* is created in the hardware system area (HSA) of the connector's CPC. This vector is used similarly to

the way the local cache vector is used for cache structures — to inform the connector that something has occurred. In this case, there is only one bit per connector (entry on the EVENTQ).

When an EMC is queued to an EVENTQ, the EVENTQ transitions from *empty to non-empty* and the list notification vector for that connector is set causing the *list transition exit* specified by the connector to be driven. It is the responsibility of the list notification exit to determine what the event was (that is, which EMC was queued to the EVENTQ). An XES service called DEQUEUE_EVENT will read the EMC on the event queue and dequeue it. The EVENTQ is now empty again and an EMC will not be queued again to the EVENTQ until the sublist becomes empty again and then non-empty.

Figure 1-14 and Figure 1-15 show an example of CQS monitoring of the shared queues structures.

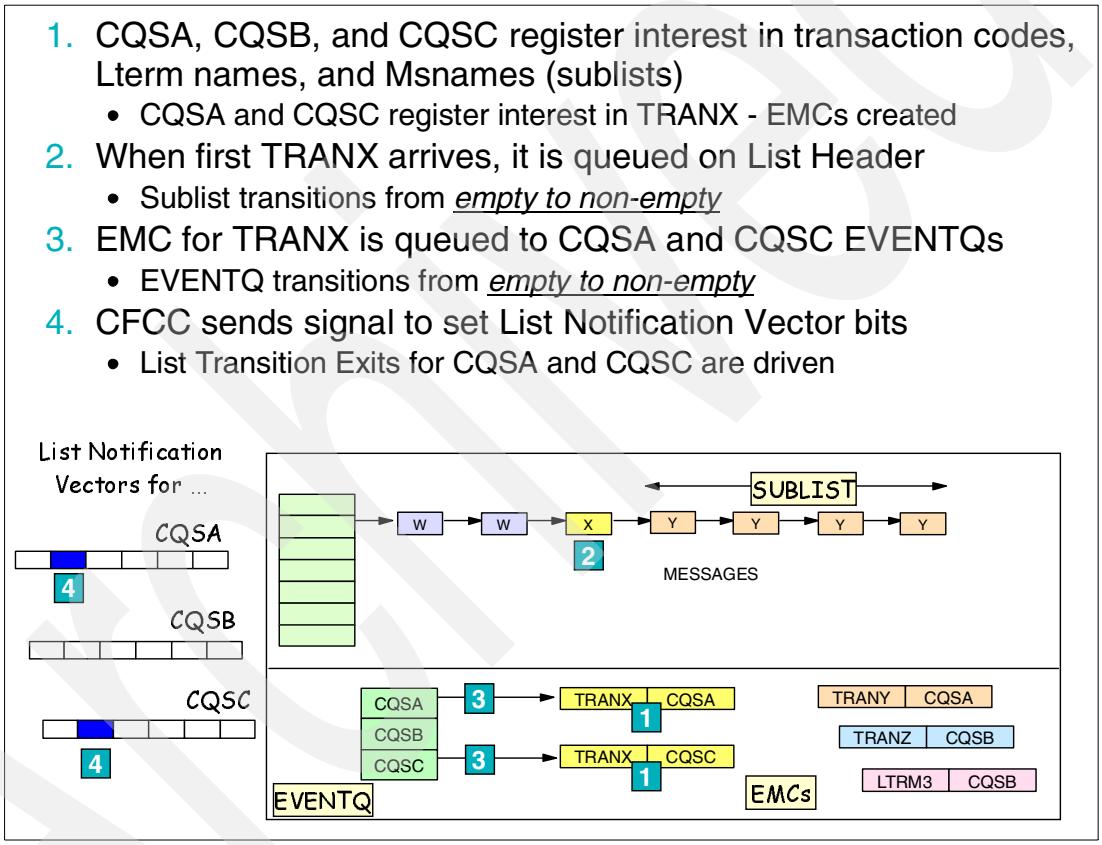


Figure 1-14 Example of CQS event queue monitoring

Note that, in the above example, the EMC for TRANY is not queued to the EVENTQ. This is because it only occurs when the TRANY sublist transitions from empty to non-empty. When the first TRANY arrived, this would have happened, the same as it is doing for TRANX. It will not occur again for TRANY unless all the messages for TRANY are read and dequeued by IMS and the sublist becomes empty and then non-empty again.

Note also that, although there are TRANW messages on the queue, no CQS has registered interest so there is no EMC for TRANW. When (if) any CQS does register interest, an EMC will be created and immediately queued to the EVENTQ.

5. List Transition Exit notifies CQS that "something" has arrived
   - Don't know what yet
6. CQS reads EVENTQ and finds EMC for TRANX
   - Dequeues EMC; EMC won't be requeued until sublist becomes empty and then non-empty; LNV bit turned off
7. CQS notifies IMS that TRANX message has arrived
8. When dependent region is available, IMS requests CQS to read message from list structure
   - If only one message, only one IMS will be successful

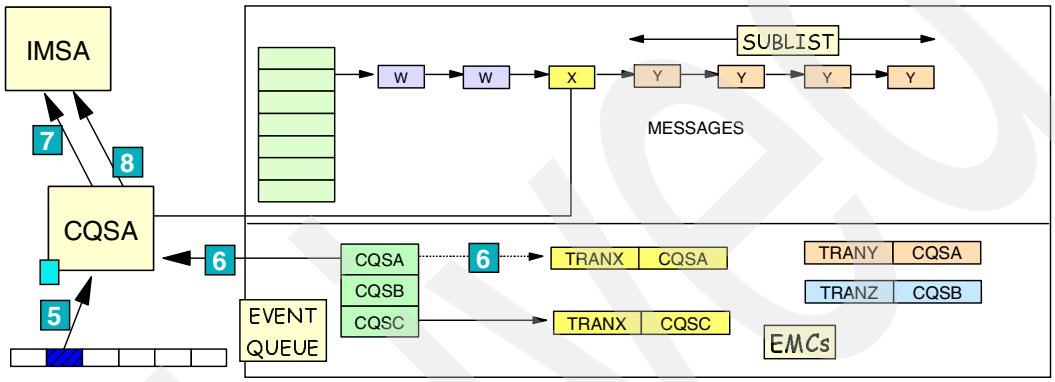


Figure 1-15 Example of CQS EVENTQ monitoring (continued)

The enqueuing of the EMC to CQSA's EVENTQ causes CQSA's list notification vector to be set, which in turn causes the list transition exit to be driven. All CQS knows at this time is that there is something on the queue, but doesn't know what. The exit will invoke list services to read the EMC and dequeue it from the EVENTQ. Now that CQS knows what was on the queue, it can notify IMSA. IMSA will, eventually, read the transaction from the shared queue, process it, and delete it. A similar sequence of events will occur for CQSC and IMSC, except that this time, when IMSC tries to read the message, it will already be gone (unless another arrived in the meantime). This is called a false schedule in IMS and incurs some wasted overhead. However, this technique is self-tuning as workload increases. The shared queues planning chapters in this book will discuss this and suggest ways to minimize the occurrence of false schedules.

Other list services

Besides the monitoring of lists, sublists, and event queues, other XES services allow the connector to:

- Read, write, move, or delete an entry in a list structure
- Combine operations
  - Read an entry and delete it
  - Move an entry and update it
  - Move an entry and read it
- Read or delete multiple entries with a single request
- Serialize a list entry (using the lock table) and perform a list operation

CQS uses all of these services in support of IMS shared queues and the Resource Manager.

1.7 Other connection services

Besides connecting to and disconnecting from a structure, XES connection services also support several other request types.

Structure delete

When the structure is no longer required, it can be deleted. This may be done automatically if the structure is not persistent when there are no more active or failed persistent connections. If the structure is persistent, and therefore not deleted when all connections are gone, it may be deleted using an operator command. The command will only work if there are no connections. When this command is entered, XCF will invoke XES connection services to delete the structure.

```
SETXCF FORCE,STRUCTURE,STRNAME=structure-name
```

Structure event notification

As mentioned earlier, all connectors to a structure belong to an XCF group. For example, all IRLMs connected to a lock structure belong to the same XCF group. When connection services are invoked to connect to the structure, an *event exit* is specified. Certain events related to the structure itself (as opposed to the data within the structure) cause this event exit to be driven. Examples of events that drive this exit are:

- New connection to a structure
- Existing connections
- Disconnection or failed connection
- Loss of connectivity by any connector
- Structure or Coupling Facility failure
- Several rebuild, alter, and duplexing events

Structure rebuild

Structure rebuild is the construction of a new instance, or a secondary instance, of a structure. When a connector connects to a structure, it may specify one or more of the following parameters.

- ALLOWREBLD=YES means that the connector will support *user-managed rebuild* of the structure.
- ALLOWDUPREBLD=YES means that the connector will support *user-managed duplexing* of the structure.
- ALLOWAUTO=YES means that the connector will support *system-managed processes*, which include system-managed rebuild and system-managed duplexing. These processes also require the CFRM couple data set to be formatted to support system-managed processes (SMREBLD and SMDUPLEX) specified in the CDS format utility.

User-managed rebuild

There must be at least one active connector to the structure and all connectors must agree to coordinate the rebuild process. Briefly, this means that the connectors coordinate to quiesce activity, to connect to the new instance, to propagate data to the new structure instance, and then to signal completion, at which time the original instance is deallocated (deleted).

User-managed rebuild can be used to move a structure from one Coupling Facility to another, or to recover from a loss of connectivity or structure failure. It may be initiated by one of the connectors, or by operator command:

```
SETXCF START,REBUILD,STRNAME=structure-name<,LOC=OTHER>
```

Archive

Specifying LOC=OTHER causes the structure to be rebuilt on another Coupling Facility and requires at least two Coupling Facilities to be specified in the PREFLIST in the CFRM policy.

System-managed rebuild

System-managed rebuild may be invoked when there are no active connectors to the structure. If there are active connectors, they only have to agree to quiesce activity until the system has rebuilt the structure.

System-managed rebuild will not rebuild a structure due to loss of connectivity, structure failure, or Coupling Facility failure. Such a rebuild requires an active connector and user-managed rebuild.

User-managed duplexing rebuild

Duplexing is a special form of rebuild where a secondary structure is allocated in another Coupling Facility that the primary structure and all operations are directed to both instances of the structure. The purpose of duplexed structures is that, if one structure or Coupling Facility fails, structure operations can continue using the remaining structure. No IMS-related connectors support user-managed duplexing.

System-managed duplexing rebuild

As its name implies, a structure can be duplexed without direct connector participation, other than to respond to the start-up event notification. A system-managed duplexing rebuild can be initiated even if there are no active connectors. Once the duplexing rebuild is complete, the connector issues requests only to the primary instance of the structure and the system propagates the operation to the secondary instance transparently to the connector. If one instance fails, for example because of a structure or Coupling Facility failure, connector operations continue with the remaining structure. When another suitable Coupling Facility becomes available, duplexing will be automatically reinstated. Except for the OSAM and VSAM cache structures, all IMS structures support duplexing.

Structure duplexing requires not only that the connector indicate support during connection, but it also requires the CFRM couple data set to be formatted with SMREBLD and SMDUPLEX, and for the structure definition in the CFRM policy to specify DUPLEX(ALLOWED) or DUPLEX(ENABLED). When duplexing is enabled, the structure will be duplexed automatically by the system when the first connector allocates the structure. When duplexing is allowed, it must be initiated, or terminated, by operator command.

```
SETXCF START,REBUILD,DUPLEX,STRNAME=structure-name
SETXCF STOP,REBUILD,DUPLEX,STRNAME=structure-name
```

Structure alter

The structure alter function allows a Coupling Facility structure to be reconfigured with minimal disruption to connectors. Alter can change the size of a structure, the entry-to-element ratio, and the alteration of the percentage of structure storage set aside for event monitor controls (EMCs). Structure alter requires that the connector specify the following when connecting to the structure:

- ALLOWALTER=YES

Structure size can be altered only within the maximum and minimum sizes specified on the structure definition statements in the CFRM policy (SIZE and MINSIZE). It may be initiated by a connector or by the alter command:

```
SETXCF START,ALTER,STRNAME=structure-name,SIZE=new-size
```

When the command is used to alter the size of a lock structure, the size of the lock table itself is not changed, only the amount of storage allocated for the record data entries.

Other structure characteristics, such as changing the entry-to-element ratio of a list structure, can only be initiated by a connector (or by the system if AUTOALTER is enabled). For IMS structures, only CQS will initiate an alter, and then only to alter the size if it detects that the structure has reached some user specified threshold of data elements in use.

AUTOALTER
When autoalter is enabled, the system may initiate a structure alter based on one of two factors:

- If the system detects that the storage in use in a structure has reached a structure full threshold specified in the policy, it may increase the size to relieve the constraint. At this time it may also alter the entry-to-element ratio if it determines that it does not represent the actual utilization of list entries and data elements.
- If the system detects that storage in the Coupling Facility is constrained, it may decrease the allocated storage for a structure that has a large amount of unused storage.

Autoalter is enabled only under the following conditions:

- ALLOWALTER=YES is specified during connect processing to enable the alter process.
- ALLOWAUTO=YES is specified during connect processing to enable system-managed processes.
- ALLOWAUTOALT(YES) is specified in the structure definition in the CFRM policy.

Also FULLTHRESHOLD(value) can be specified in the CFRM policy to override the default of 80% full.

1.8 Objective of Parallel Sysplex development

The primary objective when developing the sysplex, and later the Parallel Sysplex, was to provide a price-competitive computing facility for large commercial processing systems. It is only when you consider the parameters within which this objective was reached that you can appreciate the significance of the achievement of the Parallel Sysplex. These parameters were:

- Preserve a single system image
- Provide customer investment protection (hardware, software, and skills)
- Ensure that current applications benefit
- Provide considerable room for growth
- Provide continuous availability to corporate information resources
- Make it easy to use

1.8.1 Benefits of a Parallel Sysplex configuration

Since the base sysplex and Parallel Sysplex were announced, many benefits have been documented for its use. These include:

- Reduced costs

  Costs can be reduced, or the growth of costs limited, by continuing to use the hardware, software, operational, systems management, and application development technologies which are compatible with what you have been using for years. Among the sources of cost reduction are the ability of applications to continue to run without change, application developers and system operators to not have to be retrained, and end-users to be able to continue using an interface already familiar to them. Because the total workload, or a single application's workload, can be distributed across multiple CPCs, it may not be

necessary to have a single CPC capable of handling one application's peak workload and then be underutilized during less busy periods.

- Greater capacity

  Certainly when the sysplex was first introduced, one of the chief benefits was the increased throughput that could be attained by applying the power of two, three, or even thirty-two CPCs to a logically single workload. This is, of course, still true today. Even though the power of a single CPC has increased dramatically since 1990, so has the application workload. In fact, this increased power has led many businesses to develop new applications that were not possible with the smaller, slower, and multi-image processing capabilities without the sysplex.

- Better availability

  Because the application systems taking advantage of Parallel Sysplex capabilities can more closely function as a single system, and any work can process on any OS/390 image, the failure of a single instance of a sysplex member or application subsystem does not completely take down an entire application. Hardware and software systems can be rotated offline incrementally for maintenance while the workload processes on the remaining systems.

- Greater flexibility

  The Parallel Sysplex offers great flexibility in meeting your I/T requirements. The sysplex can be a mixture of older and newer technologies, additional systems can be dynamically and incrementally added (or removed) from the sysplex as business needs dictate, or a sysplex can be reconfigured to meet immediate and sometimes unforeseen requirements.

It is not our intention to present all of the benefits of the Parallel Sysplex. These have been documented in great detail in IBM documentation, trade magazines, trade shows, technical conferences, and so on. We did, however, want to set the stage for why the Parallel Sysplex environment is good for IMS.

1.8.2 Why move your IMS environment to a Parallel Sysplex?

The value of a new product or feature is often difficult to quantify, especially because customer installations vary so widely. However, we believe that Parallel Sysplex technology is the vehicle to carry large and not so large computer applications into the future. A look at the history of IMS since Version 5 will show that each release adds more IMS features that are based on Parallel Sysplex technology:

- Data sharing
- Shared queues
- VTAM Generic Resources
- VTAM multi-node persistent sessions
- Automatic restart management
- Sysplex communications
- Operations manager and single point of control
- Resource Manager and sysplex terminal management
- Coordinated global online change
- Automatic RECON loss notification

There is little reason to believe that future releases of IMS will not continue this progression into the Parallel Sysplex world — not only for capacity and availability, but for function as well.

2

Introducing IMS data sharing

Although the sysplex has been around since 1990 as a platform, it became an industrial strength computing facility only when OS/390 and its core products were fully integrated in their support of that platform, therefore delivering the true Parallel Sysplex experience.

In the following sections, we concentrate on the features that pertain specifically to IMS block level data sharing. Our objective is to acquaint you with these features, to show you how they work, and to give you a good perception of what is involved to plan for this migration. We have divided the contents logically into several chapters:

- This introductory chapter provides block level data sharing background and specific IMS feature introductions.
- Chapter 3, “Data sharing integrity components” on page 47 describes how IMS, DBRC, IRLM and OS/390 provide the necessary integrity for data sharing.
- Chapter 4, “Additional data sharing facilities” on page 77 contains more detailed information about other IMS facilities involved with Parallel Sysplex block level data sharing enablement.

Many IMS customers are currently enjoying the higher availability that IMS block level data sharing offers through improved fault tolerance, change control, and workload distribution.

© Copyright IBM Corp. 2003. All rights reserved.

2.1 Data sharing overview

The concept of data sharing is not new to IMS. It was first implemented in IMS/VS Version 1 Release 2 to share data between multiple IMS subsystems and two IRLMs in order to overcome the capacity constraints of the single-processor environments of the day. This database sharing support also provided a vehicle to create higher availability environments where the impact of outages caused by component failures, and planned change windows were minimized to the end user. Because the IRLM that IMS connects to must reside in the same logical partition, this limited the data sharing environment to two LPARs, therefore the name two-way data sharing.

In support of Parallel Sysplex, IMS Version 5 with IRLM Version 2.1 was the first version to expand the two-way database level data sharing limit of previous IMS and IRLM releases to N-way block level data sharing. A maximum of 32 IRLMs and 255 IMS data sharing subsystems can simultaneously access and update a single set of IMS databases with full data integrity and good performance still guaranteed. These IMS subsystems, known as a *data sharing group*, can support any required combination of IMS TM, DBCTL and batch applications running in z/OS environments.

With OS/390 APAR OW15447 and Coupling Facility Control Code (CFCC) CFLEVEL=2, 255 connectors to a cache structure are allowed. Since IRLM uses the lock structures, and the list and lock structures only allow 32 connectors, you can have only 32 IRLMs in a sysplex, although IRLMs can be shared.

2.2 Current IMS N-way data sharing components

IMS data sharing is based on base Parallel Sysplex architectures, but in addressing performance issues, new or modified high performance mechanisms are used for locking and communication. The components of a Parallel Sysplex containing IMS subsystems are:

- IMS data sharing subsystems
- The Coupling Facility (CF)
- Use of sysplex services (XCF and XES) for data sharing control and communication
- Use of Coupling Facility structures
- The Database Recovery Control (DBRC) component
- The REcovery CONtrol data set (RECON)
- Internal Resource Lock Manager (IRLM)

Figure 2-1 on page 35 introduces the major components of block level data sharing environments.

Archive

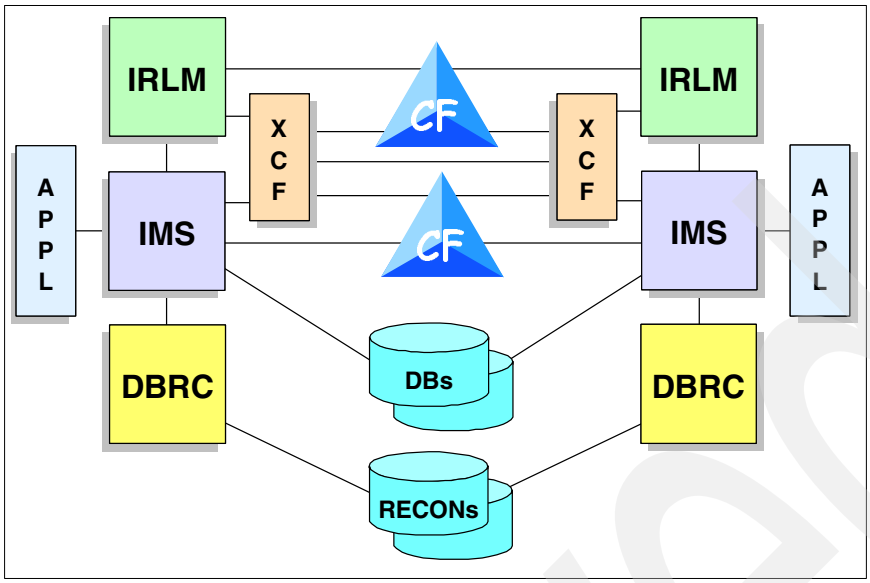


Figure 2-1 Components of data sharing environments

2.2.1 IMS data sharing subsystems

The IMS images in a single Parallel Sysplex are known collectively as a *data sharing group*. The group may consist of any combination of these IMS subsystems:

- IMS TM control regions
- IMS DB control (DBCTL) regions
- IMS batch jobs

Each of these tasks reflects one IMS image in the Parallel Sysplex since each controls its own local database buffer pool. Each batch job fills one of the 255 available data sharing slots, whereas one IMS TM or one DB control region and all BMPs under its control also fill one slot. This 255 connection limit is most advantageous for customers wanting to operate more batch regions involved in data sharing.

2.2.2 Coupling Facility

The technology that assists in making high performance sysplex data sharing possible is a combination of hardware and software products, collectively known as the *Coupling Facility*. The Coupling Facility is a microprocessor unit connected to OS/390 processor complexes by high bandwidth fiber optic links called *CF subchannels*. Coupling Facilities can also reside internally in the Central Processing Complex (CPC).

Instead of accessing data in a Coupling Facility by address, you allocate objects, called *structures*, and access data in the structures as logical entities (by name, for instance). Within the CF, storage is dynamically partitioned into structures of different types, each with a unique function. The ability to access data in this manner frees you from having to be concerned about the physical location or address of the data within the Coupling Facility.

Each type of structure provides a unique set of functions and offers different ways of using a Coupling Facility. The types of structures are:

- Cache structure
- List structure
- Lock structure

Details regarding these structures can be located in 1.6.1, “Structures, connectors, and services” on page 9.

2.2.3 Sysplex services for data sharing

Cross-system Coupling Facility (XCF) is a component of OS/390 and z/OS and was provided with the original base sysplex components. It is necessary in the block level data sharing environment to provide group, signalling, and status monitoring services to OS/390 components and authorized applications.

The OS/390 feature called Cross-system Extended Services® (XES) allows multiple instances of an authorized application or subsystem, running on different systems in a sysplex, to implement high performance, high availability data sharing by using a Coupling Facility. Applications can maintain and access data in three types of structures (list, lock, or cache) using XES macros to invoke services. Features of the different structures, available through the use of XES, include the ability to:

- Share data organized as a set of lists (list structure).
- Determine whether a local copy of cached data is valid (cache structure).
- Automatically notify other users when a data update invalidates their local copies of cached data (cache structure).
- Implement efficient, customized locking protocols, with user-defined lock states and contention management (lock structure).

2.2.4 IMS block level data sharing use of Coupling Facility structures

IMS uses two different types of structures in the Coupling Facility for data sharing support:

- **Lock structure**

  The IRLM lock structure is used by IRLM to serialize updates to the shared IMS database resources.

- **Cache structures**

  There are several IMS facilities that use cache structures:

  - **OSAM directory only buffer invalidate (XI) structure**

    Used by IMS data sharing partners to check whether or not an accessed buffer is currently valid. When an IMS subsystem makes a change to a buffer, each of the data sharing IMS partners that have accessed that buffered block have flags located in the hardware system AREA (HSA) in CPC storage set to invalid. Whenever an IMS partner attempts to access that buffer it is found to be invalid, and that IMS must read the buffer from DASD. That particular flag is then marked as valid.

  - **OSAM store-though structure**

    IMS Version 6 introduced an enhancement that allows OSAM database buffers to be read from or written to a cache structure in the Coupling Facility. Therefore, both buffer invalidations and data caching in the CF are supported. To enable OSAM database CF caching, caching options have been added to the IOBF subpool and CFOSAM statements.

  - **VSAM directory only buffer invalidate (XI) structure**

    This cache structure performs the same task as the OSAM Buffer XI structure but for VSAM buffered control intervals.

  - **VSO data entry database (DEDB) AREA store-in structure**

To perform block level data sharing of Virtual Storage Option (VSO) DEDB AREAs so that multiple IMS subsystems can read and update the data concurrently, IMS uses a cache structure in the Coupling Facility. Each CF cache structure is composed of a directory and data portion. In IMS Version 8, and in IMS Version 7 with the appropriate APAR (PQ50661), OS/390 system managed rebuild and automatic altering of structure size support is available for shared VSO structures.

2.2.5 IMS Database Recovery Control (DBRC)

Every IMS instance in the Parallel Sysplex has its own Database Recovery Control (DBRC) address space. DBRC controls the authorization of database access to application programs executing within its related IMS subsystem. Authorization is based on information kept in the RECON data set, in conjunction with the status of the ACCESS parameter in the DATABASE macro in IMS system definition or ACCESS value on the **/START DB** command for the database being referenced.

The ACCESS attribute is specific to each IMS, whereas the RECON status applies to all IMSs in the Parallel Sysplex. For example, if a database is registered with a SHARELVL of 0 in the RECON, only the first IMS to access that database will be authorized by DBRC. All other attempts, regardless of what the ACCESS parameter states, will be denied. On the other hand, if a database is registered with a SHARELVL of 3, indicating sharing by all IMSs, and one IMS has a definition of ACCESS=RD for that database, then any application program under control of that IMS will be granted READ access only.

2.2.6 The DBRC RECON data set

The Recovery Control data set (RECON) is maintained by the DBRC address space. The RECON is the primary control mechanism for database integrity and must be shared among all IMSs in the data sharing group. The RECON is updated by any DBRC in the data sharing group. Updates are generated from various sources:

- You can add, delete or change data in the RECON through the use of the Database Recovery Control utility (DSPURX00). You have to register a database and all its data sets by name, with its data sharing level, and its recovery control information in the RECON. Many of the DBRC utility functions can also be executed online through the **/RMxxxxxx** command.
- IMS itself, running in the control region or the DLISAS address space may change the status of a database or data set for integrity reasons, caused by commands, or hardware, software, or backout failures.
- When IMS utilities such as image copy or database recovery are executed, related status fields are updated in the RECON.

2.2.7 Internal Resource Lock Manager

Integrated Resource Lock Manager (IRLM) is a critical component in the Parallel Sysplex block level data sharing environment.

Each IMS subsystem participating in the data sharing group uses the services of IRLM Version 2 Release 1. The IRLMs manage data sharing global locks with the help of the Coupling Facility, which makes consistently high locking performance possible regardless of the number of sharing subsystems.

2.3 IMS features associated with block level data sharing

There are many facilities that have been introduced above the IMS Version 5 level that can be utilized in your block level data sharing environment. We have listed the major line items here.

2.3.1 IMS Version 6 features related to block level data sharing

IMS Version 6 introduced many major block level data sharing line items that customers have utilized in industrial strength production environments.

Fast Database Recovery (FDBR)

FDBR is a facility that addresses a data sharing problem known as the retained locks situation.

At the time a data sharing subsystem fails, it might hold non-sharable locks on database resources that cannot be released until the retained locks are released. Retained locks are usually released by dynamic backout during emergency restart of the failed IMS. The execution of a successful emergency restart typically takes a few minutes, but can take much longer depending upon the type of failure and nature of the emergency restart.

Even though a data sharing IMS fails, the other data sharing partners can still be executing. If application programs executing on the remaining IMS subsystems try to access a database resource that is protected by a retained lock associated with this failed IMS partner, the program is abended with a U3303 abend code, and the input transaction is placed on the suspend queue. If these access attempts to the retained locks result in many U3303 abends, then the impact on the application programs executing on the surviving IMSs can be severe (that is, application programs are unable to get their work done, and terminal operators might be hung in response mode).

Application program failures related to the access of resources held by retained locks can be avoided or minimized in two ways:

- Application programs can be changed to issue an INIT STATUS GROUPA call at initialization and to check for BA and BB status codes associated with unavailable databases. If these status codes are returned, the application can decide to terminate or take another processing path. These application changes eliminate the U3303 abend when a database call attempts to access a resource protected by a retained lock.
- FDBR provides a second way to address the retained lock problem by dynamically backing out in-flight units of work whenever an IMS subsystem in a data sharing group fails while holding retained locks. FDBR minimizes the impact of the problem by reducing the amount of time that retained locks are held.

In IMS Version 8, virtual storage constraint relief has been extended to FDBR modules. 378K has been moved from CSA to private storage, and 420K from extended CSA to extended private storage. The CSA/ECSA storage relief is received for each FDBR instance on the system.

For more information, refer to 4.4, “Fast Database Recovery (FDBR)” on page 89.

Shared sequential dependent segments

Use of the Fast Path DEDB sequential dependent (SDEP) segment function provides the user with a time-sequenced insert capability for a portion of a DEDB AREA that has SDEPs defined. The SDEP buffer is kept in main storage and is written by an output thread during Fast Path synchronization point processing after it is filled to capacity. At the same time, the next SDEP buffer is obtained on behalf of the subsequent CI.

With the implementation of shared SDEPs, the method of allocating SDEP CIs has changed. Each IMS subsystem image in a sysplex data sharing environment is known as an IMS partner and each IMS partner now manages its own SDEP CIs. The SDEP AREA is shared for read access, but the individual CIs are not shared for insert. A timestamp has been added to the end of each SDEP segment to allow SDEPs to be shared without changing time-of-insert processing. No Coupling Facility structures are used for shared SDEP support.

OSAM Coupling Facility data caching

OSAM CF caching allows OSAM blocks to be cached in a Coupling Facility structure. One of the costs of data sharing among IMS subsystems is known as *buffer invalidation*. Whenever a data sharing IMS (IMSA) updates a block in its local buffer pool, if the block is also in another system's buffer (IMSB), that buffer must be invalidated. A future reference by IMSB to the block in the invalidated buffer requires it to reread the block from the database data set on DASD.

With the OSAM caching capability using CF store-through cache structures, IMS users now have the option of storing updated blocks in a Coupling Facility structure such that references to invalidated blocks can be satisfied from the Coupling Facility, therefore avoiding a reread from a data set on DASD.

The OSAM caching capability has three options:

- Store all selected database data set blocks in the Coupling Facility or
- Store only blocks that have been changed in the Coupling Facility, or
- Do not store any OSAM blocks in the Coupling Facility

For more information, refer to 4.1, “OSAM Coupling Facility data caching” on page 78.

/STOP REGION ABDUMP command enhancements

In prior releases, completion of the **/STOP REGION ABDUMP** command was held if the dependent region to be aborted was waiting for an IRLM lock. If the wait for the command to execute is excessive, the operator wanting to terminate the dependent region can issue another form of the command, **/STOP REGION CANCEL**, to force the region to terminate. This form of the command causes the entire IMS online system to abnormally terminate if the dependent region were waiting on a lock.

This command has been enhanced to detect whether the region which is to be stopped is waiting on a lock request within the IRLM. If it is waiting on a lock when the command is issued, the waiting dependent region task is resumed and forced to return to its dependent region address space, where it is abended (U0474).

DBRC NOPFA option

This prevents the setting of the Prohibit Further Authorization flag when the GLOBAL form of a database command is issued, such as **/DBR, /STOP,** or **/DBD** with the GLOBAL keyword option. If the intent of issuing a command, **/DBR DB xxxx GLOBAL** for example, was to deallocate a database from multiple data sharing IMS subsystems so that stand alone batch processing could be started against the database, the use of the NOPFA keyword option command eliminates the need to turn off the Prohibit Further Authorization flag in DBRC before starting the stand alone batch processing.

BMP scheduling flexibility

APAR PQ21039 for IMS Version 6 simplifies the specification of the IMSID for users whose BMPs can execute under multiple control regions in a Parallel Sysplex. Without this enhancement, moving a BMP from one control region to another typically requires a change

in the BMP IMSID parameter. With this enhancement, no changes to BMP JCL are required even when a BMP is executed under different control regions.

Additional lock detection reporting

Hangs caused by lock contention in a sysplex data sharing environment may be difficult and time consuming to diagnose. The purpose of this enhancement is to provide the user with a report listing all BLOCKERs and WAITERs in the sysplex. The report will also provide the necessary information to identify the transaction causing the hang more easily, so that action can be taken to resolve the problem.

The IRLM command, MODIFY RUNTIMEO, causes IRLM to call an IMS time-out exit. SMF records are created for all IMS transactions holding locks and determined to be BLOCKERs or WAITERs at the time the IRLM modify command is issued. These SMF records can then be processed by an RMF exit to create a report.

Other IMS Version 6 features of interest to block level data sharing users

There are other IMS Version 6 features that are not directly related to BLDS but still of general interest to IMS users:

- **CI reclaim**

  Using block level data sharing or XRF prevented IMS from releasing empty KSDS CIs in releases earlier than IMS Version 6 even when all content in those CIs had been deleted. CI reclaim was turned off in prior releases, which had the potential for causing performance problems when using primary and secondary indexes.

- **Improved security**

  This is accomplished with IMS Version 6 by propagating the security environment to a back-end IMS when it is processing a transaction that requires signon security. This is an important enhancement in either a shared queues or MSC environment.

- **Online change for DEDBs**

  Since IMS Version 6, DEDB online change is consistent with the level of support for full function database online changes. Adding, deleting, and changing of DEDB databases and AREAs is fully supported. This allows changes to be made without bringing down IMS.

- **UCB VSCR/10,000 DD names**

  This feature provides virtual storage constraint relief for private address space storage below the 16 MB line and the ability to allocate as many as 10,000 data sets to an address space. Note: There is a limit of 8,189 full function database data sets that can be opened concurrently. For DEDBs, which are allocated to the control region, 10,000 AREA data sets can be opened as well as allocated. These enhancements allow the number of data sets that can be allocated and opened in the IMS control region and DL/I SAS address spaces to be greatly increased.

- **DBRC performance improvements**

  Changes have been made to reduce the number of DBRC requests and to otherwise reduce RECON I/O in order to provide some performance improvement.

  Another performance improvement has been made for DEDB AREA unauthorization and deallocation. Prior to IMS Version 6, DBRC was called twice for each AREA, once to unauthorize and again to deallocate. The process in IMS Version 6 has been changed so that IMS calls DBRC once with a list of AREAs to process. All AREAs in the list are unauthorized and deallocated at the same time.

- **Image Copy 2 utility**

Image Copy 2 utility (DFSUDMT0) is a new form of image copy that produces an image copy with high performance with no or minimal database data set outage. The fuzzy image copy option allows the database data set to be copied to remain fully accessible by the online system(s). A clean image copy does require a short outage (a few seconds). Image Copy 2 utility utilizes SMS concurrent copy and supports all types of database data set organizations (KSDS, ESDS, and OSAM) used by IMS.

- **Open Database Access (ODBA)**

  Open database access provides a callable DL/I interface from application execution environments outside the scope of IMS control, such as DB2 Stored Procedures, WebSphere Enterprise Java Beans (EJBs), and CICS Java programs. Open Database Access is not needed for IMS controlled regions, such as MPRs, BMPs or IFPs, for calls to locally controlled databases. The application execution region, which may be any address space in an OS/390 image, connects through a database resource adapter (DRA) interface to DBCTL or IMS TM/DB subsystems.

2.3.2 IMS Version 7 features related to block level data sharing

IMS Version 7 continues to improve high availability characteristics for data sharing users.

High Availability Large Database

The only reference to this major line item is that HALDB has been designed for full block level data sharing operation.

IMS Monitor enhancements

New functionality has been added to the IMS monitor to enable you to constrain whatever monitor records are written during your monitor run, to avoid writing unwanted output and to reduce the cost of running the monitor. The monitor is an important tool to use when benchmarking performance for your pre and post block level data sharing environments.

Change accumulation enhancements

In a block level data sharing environment, the change accumulation process produces a lot of spill records, as it has to run without having all the log data sets available. For some customers, the volume of spill records is very large and causes performance problems when the CA data sets are needed for recovery.

Database recovery cannot be run until all the spill records have been merged into the change accumulation data set. This requires that a last change accumulation be run just prior to recovery. Recovery must wait for this to complete.

IMS Version 7 tends to produce fewer spill records than IMS Version 6. This is because IMS Version 6 uses the DSSN value in the log records to determine which records to include. The utility does not accumulate any records if any log records have a lower DSSN value. IMS Version 7 uses the DSSN and the log sequence number (LSN) for its decision on whether to include a log record in the accumulation. The LSN is stored in the RECON and gives the Change Accumulation utility (DFSUCUM0) better information about the log records being processed. It accumulates all records up to the time for which the log records for the data set are included as inputs.

Online Recovery Service (ORS)

Although ORS is a separately price product, it was introduced with IMS Version 7. Its design goals are:

- Recover multiple DBDS and Fast Path AREAs in a single pass of the IMS log data sets.

- Timestamp recovery to allocation boundaries or any prior point in time.
- Perform™ as a function of the control region, executing in parallel with other online IMS activity.
- Be initiated through recovery related IMS online commands.

Fast Path DEDB enhancements

I/O error support improvements have been introduced for toleration of individual write I/O errors for a DEDB with one AREA data set (ADS). In a case where one or a few Control Intervals (CIs) suffer write I/O errors in a DEDB AREA that is implemented as a single AREA data set (ADS), the data in the CIs in error will remain available to all applications in the IMS system which experienced the write error, and the ADS may be recovered without any requirement for a data outage. Figure 2-2 explains this process.

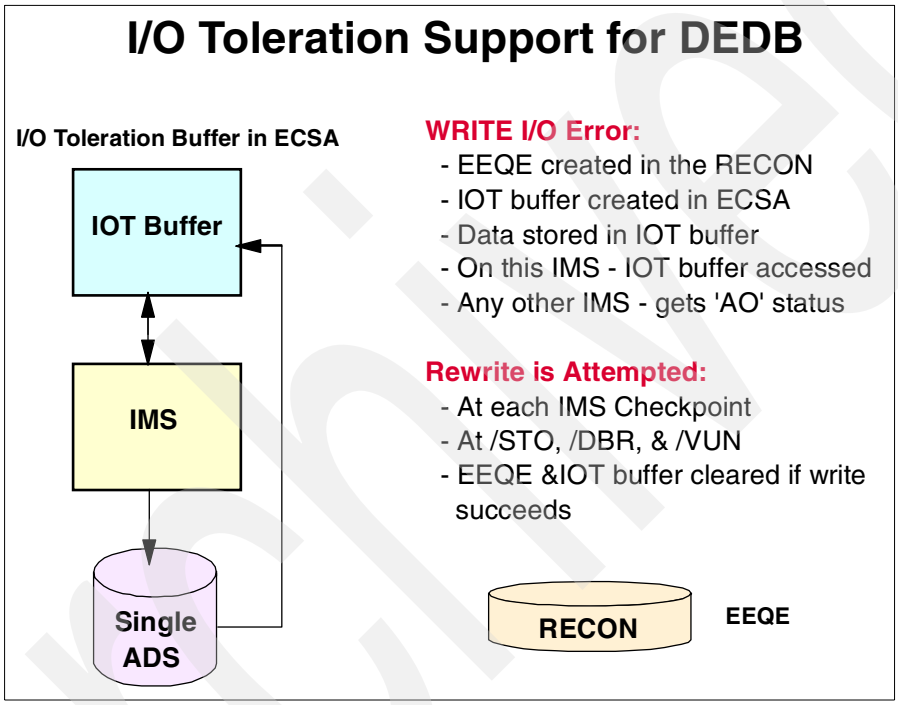


Figure 2-2 DEDB I/O toleration support

- **DEDBs MADS I/O timing**

  One or more of the AREA data sets (ADSs) implemented on RAMAC® or another RAID storage device may occasionally have extraordinarily prolonged I/O durations during data recovery processing within the DASD subsystem. If any I/O, read, or write is delayed beyond the user defined MADS I/O timing parameter, then IMS Version 7 will initiate a long busy event which involves the following steps:

  a. Prohibit further reads and writes to the ADS concerned, and thereby temporarily remove the ADS from the MADS.

  b. Abandon the I/O that timed out.

  c. Reissue the I/O if it was a read I/O. That read would then be performed against another ADS (since the ADS that suffered the long busy is no longer in the MADS) and can be expected to have a normal response time (presuming that you have configured the MADS so that they are not all subject to the same failure that caused the long busy situation).

d. If the I/O that timed out was a write I/O, build a list entry for a CF list structure identifying ADS I/O time out instances. The list entry will identify:

   - The AREA name
   - The ADS DDNAME
   - The RBA of the I/O request
   - The I/O requester's ID

   Note that since the ADS is a participant in a MADS, write I/Os to other ADS(s) are expected to have completed satisfactorily, since the fact that the MADS is still active means that at least one other ADS must still be active. Therefore, the CIs involved can still be accessed from the other ADS(s) even while this "long busy event" continues.

e. Issue an un-timed read I/O to the ADS to detect the end of the "long busy" situation.

f. If the end of the "long busy" situation is recognized, by a successful return from that untimed read, IMS will commence recovery of failed writes by processing each entry in the CF list (as discussed in point d above), until that list is empty.

   - For each entry in the CF list, IMS will read the relevant data from another ADS and write that data back to the ADS that is being recovered.
   - When the list is empty, the ADS will be returned to normal operation.

g. If the untimed read returns an I/O error, normal MADS write I/O error processing will occur:

   - The ADS will be flagged in error and stopped.
   - You will need to use an ADS Create process to recover that ADS from the other ADS(s) in the MADS.

This process is illustrated in Figure 2-3.

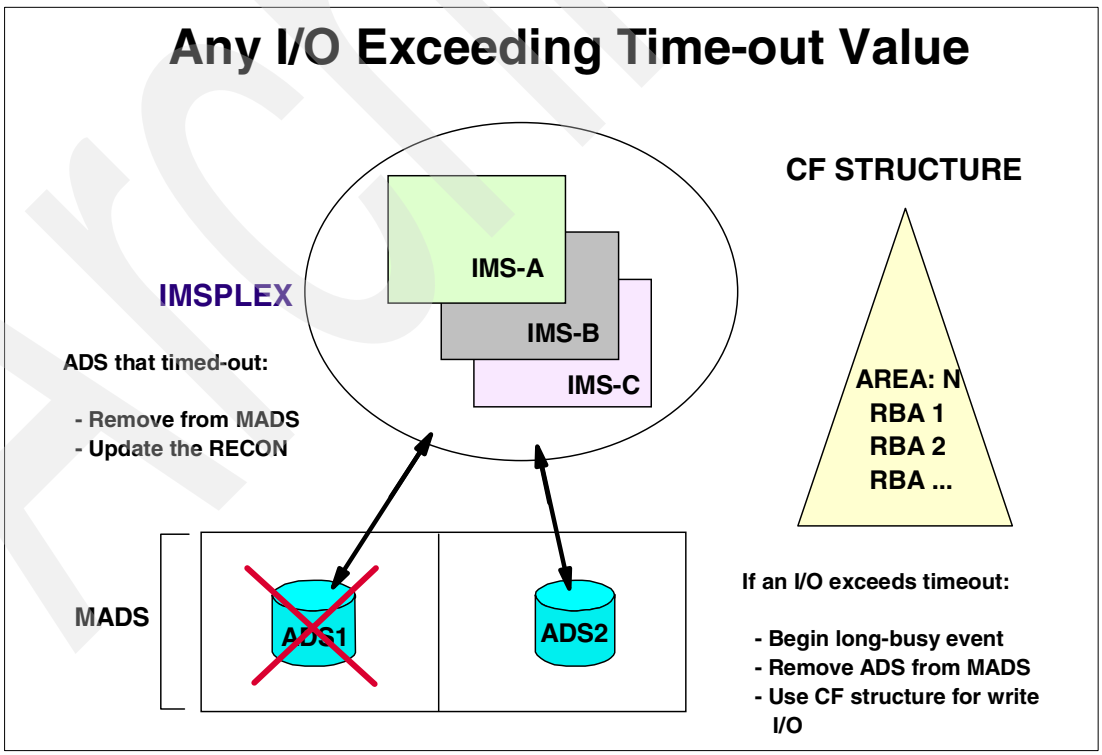


Figure 2-3 MADS I/O timing

Archive

The MADS I/O timing function is implemented by inclusion of a MADSIOT statement in DFSVSMxx, defining the CF structure name and the time-out value to be used:

```
MADSIOT = (structurename, time-out value)
```

The active CFRM policy for the Parallel Sysplex must, of course, include the structure name.

Note: You should ensure that multiple area data sets for one AREA are configured so that they have no single point of failure, and so that no item in the configuration could cause a long busy simultaneously for each of the ADSs.

Other IMS Version 7 features of interest to block level data sharing users

IMS Version 7 is a very rich version containing many other useful new line items like the following:

- **Image Copy 2 enhancements**

  Compression may be used for Image Copy 2 copies. This feature is invoked by control statements in the utility or a GENJCL.IC keyword. This invokes the COMPRESS parameter for the DFSMSdss™ DUMP statement. DFSMSdss RESTORE automatically expands data which is invoked by IMS Database Recovery utility and ORS. The benefits are the smaller space requirements.

- **DBRC enhancements**

  Previous IMS releases issued warning messages when the PRILOG record size approached its maximum capacity. However, very little recovery time was given before IMS was shut down. New DBRC parameters have been delivered which will allow the warning thresholds to be lowered, therefore giving more time for recovery.

  A new IMS message is issued when a RECON data set is lost. This is designed to allow your installation to automate the recovery, especially in a sysplex environment.

  IMS Version 7 has modified the scheme used to serialize access to the RECON data sets. This has been done to favor online systems. The intent is to lessen the rate of congestion of access requests to the RECON caused by a number of batch jobs using DBRC that are being started simultaneously.

- **New message DFS1929I at IMS initialization**

  As the number of IMS parameters needed increase in block level data sharing environments, it becomes more difficult to determine exactly what has been coded for each online IMS subsystem. The message DFS1929I displays the system parameter values that are set during IMS initialization. The values can come from the IMS system generation defaults, from the DFSPBxxx member of IMS PROCLIB, or from any final override specification in the JCL EXEC statement. This information provides an easy way to determine the values used in IMS initialization for tuning and problem determination purposes.

2.3.3 IMS Version 8 features related to block level data sharing

IMS Version 8 is, at the time of writing, the latest in IMS Versions to support data sharing environments.

CF structure management improvements

Enhancements are being made available to Coupling Facility structures in the recent releases of OS/390 and z/OS. Structures used in IMS Parallel Sysplex environments can take advantage of these enhancements.

The support available depends on the type of structure:

- **System managed rebuild** is supported for the following structures:
  - Shared VSO cache (support also available with IMS Version 7 and APAR PQ50661)
  - Shared message queues list
  - IRLM lock (support available for any IMS release using IRLM Version 2 Release 1 with APAR PQ48823)
  - CQS resource list
- **Autoalter** is supported for the following structures:
  - Shared VSO cache (support also available with IMS Version 7 and APAR PQ50661)
  - Shared message queues list (also available with IMS Version 6 and 7)
  - IRLM lock (support also available with any IMS release using IRLM Version 2 Release 1 with APAR PN67253)
  - CQS resource list
  - OSAM and VSAM cache (IMS Version 6 support with APAR PQ38946, IMS Version 7 support with APAR PQ53939)
- **System managed duplexing** is supported for the following structures:
  - Shared VSO cache (support also available with IMS Version 7 and APAR PQ50661)
  - Shared message queues list (support also available with IMS Version 7 and APAR PQ47642)
  - IRLM lock (support available for any IMS release using IRLM Version 2 Release 1 with APAR PQ48823)
  - CQS resource list

Fast Path DEDB enhancements

Several database enhancements have been introduced for Fast Path.

- Support for Fast Path DEDBs greater than 240 AREAs
- Support for nonrecoverable DEDBs
- Enhanced support for sysplex Coupling Facility management of DEDBs
- Unused IOVF count update

We will discuss the enhanced support for sysplex CF management of DEDBs in Chapter 4, “Additional data sharing facilities” on page 77.

Batch RRS support

OS/390 Resource Recovery Service (RRS) support may be invoked if more than one resource manager is involved in update processing, such as IMS DB and DB2. RRS will be invoked with the intention to operate as a system wide syncpoint manager to ensure that all commits or backouts will be kept consistent across all participants.

If you are running batch applications for updating IMS databases and resources managed and hosted by other resource managers, such as DB2 or MQSeries®, RRS support for the batch applications is now available. This functionality has been made available to IMS Version 7 through the service process (APAR number PQ51895). The batch RRS support is intended to simplify your administration and make your operations less error prone and easier to use.

Other IMS Version 8 features of interest to BLDS users

These features provide improved performance during online processing and database recovery periods.

- **Image Copy 2 enhancements**

  The enhancements for Image Copy 2 (IC2) in IMS Version 8 are intended to ease the coordination of your database administration and are based on a number of user requirements. With IMS Version 8, you are able to execute multiple utility control statements during one IC2 execution step of the DFSUDMT0 utility. This means, multiple copies are created in one IC2 step invoking DFSMSdss once.

  In overview, Image Copy 2 now supports:

  - Multiple DBDSs, as well as Fast Path AREA data sets copied in one execution step.
  - When multiple data sets are copied, IC2 issues multiple DFSMSdss DUMP commands preceded by a PARALLEL command. This requests that DFSMSdss initiate multiple concurrent tasks to process the DUMP commands.
  - IC data set group names.
  - User choice of the DFSMSdss OPTimize() specification to control the amount of data read from the database data set in one I/O request.
  - Use of the “Same Data Set” option to create (‘stack’) multiple output image copies in the same output data set.

- **Parallel database processing**

  The parallel database processing enhancement is provided for improving the performance of IMS after restart to reach a steady operational state faster. Multiple TCBs (using multiple threads) are used for database authorization, dynamic allocation, open, close, and end-of-volume processes, which were running as serialized processes before. The parallel database processing provides the following benefits:

  - Exploits available processor power
  - Reduces the elapsed processing times
  - Achieves faster steady state response times

In October 2002, the IMS performance group at the IBM Silicon Valley Laboratory performed an IMS Version 8 performance evaluation and Parallel Database Open processing was included in their review.

Identical BMPs were executed under IMS Version 8 and IMS Version 7 with all involved databases restored to their pristine state prior to each test. The database open times were compared using a batch message processing region to open all databases. IMS Version 8 Parallel Database Open provides the following improvements when compared to IMS Version 7:

- Normal restart times are reduced by up to 56%
- Emergency restart times are reduced by up to 39%
- Shutdown (/CHE FREEZE) times are reduced by up to 19%
- Database open times are reduced by up to 5%

Note: These comparisons are based on reaching a ‘ready for work’ state which IMS Version 8 achieves upon restart compared to IMS Version 7 which requires additional processing to initiate the database open process. Normal disclaimers related to distribution ‘AS IS’ without any warranty either expressed or implied, is associated with these test results.

Now that we have introduced some of the concepts and features associated with block level data sharing we describe this environment in more detail in the following chapters.

Archived

3

Data sharing integrity components

Database integrity is a fundamental design point of IMS and is defined as:

> Database integrity is maintenance of the correct internal database information (such as, pointers between segments), and the maintenance of the user data stored in the database. It uses database locks to protect the database while making it available to multiple application programs concurrently. Integrity includes the isolation of effects of concurrent updates to a database by two or more application programs.

To maintain integrity, database updates must occur only on a transaction consistent basis, meaning that:

- None of the effects of a transaction can be visible to other programs until the transaction reaches a commit point. The data items that will be affected by updates are inaccessible (locked) during the updating process.
- All the effects of a transaction (multiple updates to multiple databases) are made available to other applications and programs after the commit point for that transaction if the transaction is successful, or
- If the transaction fails and is abended or rolled back, all updates are removed by the backout processes.

With block level data sharing, this definition has been extended to include protection of data in situations where updates occur on multiple processors in a Parallel Sysplex environment.

© Copyright IBM Corp. 2003. All rights reserved.

3.1 Data sharing integrity

To maintain integrity and recoverability in a block level data sharing environment, IMS provides the following functions:

- Database authorization
- Interface with DBRC and its RECON data set
- Lock management
- Use of IRLM V2.1 for lock management
- Buffer invalidations
- Interface to notification processes

Figure 3-1 presents a pictorial overview of the major integrity elements available to the Parallel Sysplex data sharing environment.

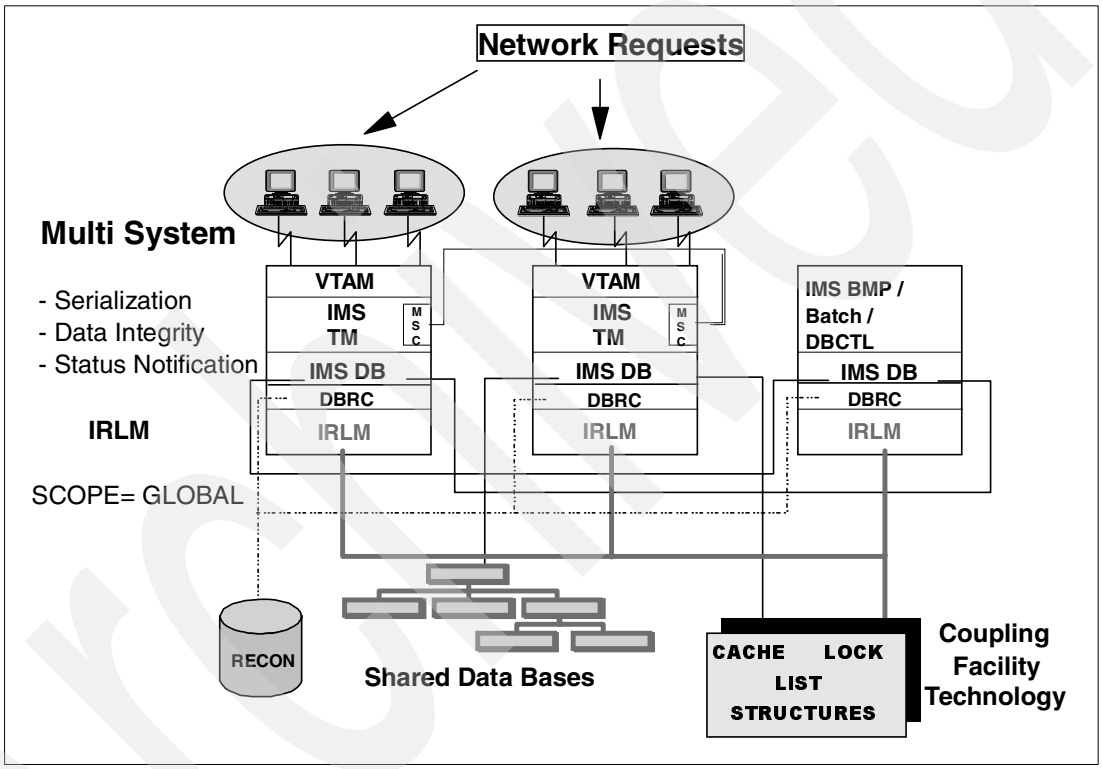


Figure 3-1 Block level data sharing data integrity components

We examine each of these major data sharing integrity components in this chapter.

3.2 Database authorization

Database authorization is a DBRC function, and is used to ensure that an IMS subsystem does not create an invalid data sharing environment. An IMS subsystem must ask DBRC for permission to access a database before using it. Authorization checks the concurrent users and environment and authorizes or refuses to authorize the database, depending on the current authorizations and access intent of the subsystem. Block level data sharing requires:

- Shared RECON data sets initialized as SHARECTL for IMS Version 6. SHARECTL is mandatory with IMS Version 7 and 8 and there are no other options.

- One IRLM or multiple communicating IRLMs indicated by the IRLMGRP parameter in the IRLM start procedure.
- SHARELVL=2 with one IRLM or SHARELVL=3 with multiple IRLMs.

The DBRC authorization process ensures that data sharing configuration rules are not violated.

3.2.1 When authorization occurs

Authorization for a full function database occurs the first time the IMS subsystem accesses the database at program specification block (PSB) schedule time, during application initialization with the exception of HALDB databases. For HALDB, the authorization does not occur at PSB schedule time, but later when it is referenced during partition selection processing. Authorization processing for Fast Path DEDB AREAs occurs at the first DL/I call.

Authorization is retained until the subsystem terminates, or a **/DBRECOVERY** command is entered on the IMS subsystem to which the database or partition was authorized. A **/DBRECOVERY** command entered in one data sharing subsystem directed toward a database or partition will not affect the authorization of another sharing subsystem for the subject database unless the GLOBAL keyword is added. But if the GLOBAL keyword is used then the database or partition will be closed on all online subsystems if possible and DBRC will prevent further authorization.

3.2.2 Access intent

Access intent is determined by DBRC when a subsystem tries to allocate a database.

- For a batch job, DBRC uses the processing option (PROCOPT) of the PSB for each database to determine the access intent. If the PSB has multiple PCBs for the same database, the highest intent for that database is used.
- For an IMS DC online system, the ACCESS parameter of the DATABASE macro sets the access intent.

There are four processing intent attributes. They are listed here from the most restrictive to the least restrictive:

1. Exclusive (EX)

   The subsystem requires exclusive access of the database and no sharing is allowed, regardless of the share options registered in DBRC.

   – PROCOPT of L or xE (batch), (where x = A,D,G,I,R)
   – ACCESS of Ex (online)

2. Update (UP)

   The subsystem may update the database. Even if no updates actually take place, the database is held in update mode. Any logs created with actual changes during this process are required for recovery or change accumulation.

   – PROCOPT of A,I,R,D (batch)
   – ACCESS of UP (online)

3. Read with integrity (RD)

   The subsystem only reads the database, but it also checks any enqueue or lock held by other subsystems. It waits for any conflicting lock to be released before proceeding.

   – PROCOPT of G (batch)
   – ACCESS of RD (online)

4. Read without integrity (RO)

   The subsystem only reads the database and it does not check any lock or enqueue held by other subsystems.

   – PROCOPT of GO (batch)
   – ACCESS of GO (online)

3.3 Database Recovery Control and the RECON

Database Recovery Control (DBRC) controls the authorization process for each IMS to access and update shared data. Information is stored in the RECONs to reflect system access to data and to store information needed for database recovery.

Designating the RECON as SHARECTL is a prerequisite to any data sharing and tells DBRC to:

- Enforce IMS integrity processing.
- Provide DBRC authorization for registered databases.

Under SHARECTL, DBRC will control authorization access to IMS databases by all IMS sharing subsystems, according to their processing intent. DBRC will record information in the shared RECON data sets for the IMS subsystems about:

- Allocation of databases to a subsystem, and subsystem status changes
- Log data on behalf of each sharing subsystem, including archiving, change accumulation and merge processing
- Image copies of database data sets or Fast Path AREA data sets

From this data and user defined skeleton JCL, DBRC can build GENJCL job streams for:

- IMS log archiving jobs
- IMS utilities including database recovery
- Any user defined job stream based on the GENJCL.USER function

3.4 Lock management

IMS uses locking to serialize processing. There are two prime reasons for this lock serialization. First, it ensures that no application program can access uncommitted updates. Secondly, it serializes updates to a OSAM block or VSAM control interval.

Without data sharing, locking is done for only one IMS system. Full function databases use either the program isolation (PI) lock manager, or the IRLM. Fast Path databases use the Fast Path lock manager in conjunction with either the program isolation lock manager or the IRLM. Knowledge of these locks does not have to be spread to other instances of these lock managers. Without data sharing, locking is known as *local locking*.

In the data sharing sysplex, locking is extended across all IMS subsystems in the data sharing group. This is known as *global locking*. Locks prevent the sharing subsystems from concurrently updating the same block or control interval (CI). The block or CI is locked before making the update and the lock is held until the buffer is written to DASD (and all other copies of the buffer have been invalidated in other subsystems if necessary), during sync point processing or buffer steal processing.

When a lock is requested, the request may not be compatible with a holder of the lock. This prevents the IRLM from granting the lock request. Typically, the requestor must wait until the

holder releases the lock. On the other hand, locks can be requested with a conditional attribute. Conditional lock requests don't wait for release by holders; instead, the IRLM tells the requestor that the lock is not available. The requestor cannot have the lock and must take some other action. The use of the conditional attribute is explained in 3.4.1, “Lock properties” on page 51.

With block level data sharing, the IRLM must be used. One IRLM can share locking information with other IRLM instances by means of a notification process. Notifications are messages sent from one IMS to other IMS systems in the data sharing group. These messages contain information that other IMS systems need. For example, when global commands are issued in one IMS system, notifications are sent to other IMS systems. Similarly, data set extension information is sent by notifications.

3.4.1 Lock properties

Lock requests have several properties. These properties are used to determine what is locked, who holds the lock, the compatibility of the holder with other potential concurrent holders, and recovery characteristics. These properties include the following:

- **Requestor**

  The lock requestor is almost always an instance of an IMS application program. It could be an IMS transaction manager message processing program, an IMS transaction manager Fast Path program, a batch message processing program (BMP), or an IMS batch job.

- **Resource**

  The resource is the item locked. This could be a database record, an OSAM block, a VSAM control interval, a DEDB unit of work, or some other entity.

- **Private attribute**

  The *private* attribute is used in conjunction with levels to determine compatibility. The designation *private* tells the lock managers to grant the lock only within one IMS subsystem. Even though lock requestors in different IMS subsystems request the lock at compatible levels, if one requests the lock with the private attribute, others cannot hold it concurrently.

- **Known attribute**

  The attribute *known* tells the lock manager that this lock may protect an update. If the system in which this requestor is running should fail, the lock should be preserved. The attribute *known* causes the lock to become a modified lock.

- **Conditional attribute**

  The attribute *conditional* indicates that the lock requestor does not wait for the lock. If another holder prevents the requestor from acquiring the lock, the IRLM returns to the requestor with an indication that the lock is not granted. If the conditional attribute is not used, a lock request is suspended if the lock cannot be granted. This suspension is ended when the lock can be granted.

  Locks are requested conditionally for one of two reasons:

  – The requestor is holding another resource, such as ownership of a full function database buffer, which should not be held if the requestor is suspended.

  – The requested resource is not specifically required. For example, full function hierarchic direct space management sometimes requests a conditional lock on a block. Instead of waiting for the lock on the requested block, it looks for another block to hold the segment.

Archived

- **Level or state**

  Locks are requested at a level. This is also called the *state.* The level is used to determine whether another lock holder can hold the lock concurrently. The IRLM uses a matrix to determine if two levels match. Levels typically vary according to what the requestor might do with the resource. For example, a reader requests a Level 4 lock for a database record. An updater requests a Level 6 record for a database record. Level 4 locks are compatible with each other, but not with Level 6 locks. This allows two readers to share a database record, while an updater does not share a record.

The lock compatibility matrix is shown in Table 3-1. *Yes* indicates the lock levels are compatible. *No* indicates that they are not compatible.

Table 3-1 Lock compatibility matrix

	Level Requested					
		2	3	4	6	8
Level Held	2	Yes	Yes	Yes	Yes	No
	3	Yes	Yes	No	No	No
	4	Yes	No	Yes	No	No
	6	Yes	No	No	No	No
	8	No	No	No	No	No

3.5 Full function locking overview

IMS provides integrity for full function databases by using several different types of locks. The most commonly used locks are database record and block locks.

Database record locks prevent programs from accessing uncommitted updates. They are used both with and without block level data sharing. Programs get a lock on a database record when they access any part of the record. Updaters get an unshared lock; others get a shared lock. This prevents any program from accessing uncommitted updates.

Data sharing adds block locks to the locking environment. These locks are used to serialize updates to a block by different IMS subsystems. Only one IMS system can update a physical block at any time and block locks are used to enforce this. Before a program can update any database record in a block, it must get a lock on the block. This lock prevents a program in another IMS system from updating the block concurrently.

Other lock types are used to coordinate or control processes such as data set opens, closes, extensions, and key sequenced data set (KSDS) control interval and control area splits. There are also locks used for IRLM notifications.

3.5.1 Database record locks

Database record locks are used to serialize access to database records while positioned on them. When the requestor changes position, the lock is released and a new one obtained. They prevent programs from accessing uncommitted updates. They also prevent updates to database records for which the Q command code has been used. The “Q” command code is used by a program to ensure that no other program modifies a segment while the first program moves its position to other database records. The segments locked by the “Q” command code are not freed (regardless of whether the update has occurred or not) until:

- An explicit DEQ call with the same class is issued
- A commit point is reached
- The program terminates normally

When the PROCOPT value for a program communication block has the O option, read without integrity occurs and database record locks are not requested.

The following describes the properties for the database record locks:

- **When requested**

  These locks are requested when an application program causes a database record to be accessed. The access may result from a call requesting a segment in the record or a call that must traverse a segment in the record. For example, a GN call may traverse many database records and request many database record locks.

- **When released**

  If an update is made to a database record or if the PROCOPT includes E, the lock is not released until application sync point time. The lock is released when the application program positions on a different database record using the same program communication block (PCB), if an update is not made, the PROCOPT does not include E, and the Q command code is not used for any segment in the database record.

  A dequeue call is used to release the Q command code. If a new position is not acquired, the lock is released at sync point time.

- **Requestor**

  The requestor is the program that issues a DL/I call causing the database record to be accessed.

- **Resource**

  The resource that is locked is the identification of the database record. For all databases, the resource includes the global DMB number to identify the database and the DCB number to identify the data set in the database. In addition, there is an identification of the record within the database:

  – For hierarchic indexed direct access method (HIDAM) databases, this is the relative byte address of the root segment.

  – For hierarchic indexed sequential access method (HISAM) databases and secondary indexes, this is a hashed value of the key of the record. However, if a secondary index is used as an alternate processing sequence, then the resource locked is the RBA of the root of the secondary index target segment.

  – For hierarchic direct access method (HDAM) databases, this is the relative byte address of the root anchor point from which the root is chained. With hierarchic direct access method databases, the database record lock is actually a lock on all of the records chained from this root anchor point.

- **Private attribute**

  The *private* attribute is used with level 6 database record lock requests. It is not used with level 4 lock requests.

- **Known attribute**

  The *known* attribute is used with level 6 database record lock requests. It is not used with level 4 lock requests.

- **Level**

  The level depends on the PROCOPT of the program communication block used to make the call. Level 4 is used with PROCOPT=G program communication blocks. Level 6 is used when the PROCOPT includes I, R, D, or A.

  Table 3-2 summarizes the lock level to the program communication block PROCOPT values.

Table 3-2 Database record lock levels

Program Communication Block PROCOPT Value	Lock Level
G only	4
I,R,D, or A	6
GO	none

3.5.2 Block locks

Block locks are used to serialize updates to OSAM blocks and VSAM control intervals. They prevent programs in different IMS subsystems from making concurrent updates to the same block or control interval. They are used for OSAM, VSAM entry sequenced data sets, and VSAM key sequenced data set data components.

For OSAM and VSAM entry sequenced data sets, block locks do not prevent concurrent updates to different database records in the same block or control interval by programs in the same IMS system.

For VSAM key sequenced data set data components, block locks may prevent concurrent updates to the same control interval by programs in the same IMS system. This depends on the operations being done by the program. Multiple concurrent replaces or inserts of logical records in the same control interval are allowed. Similarly, multiple concurrent erases of logical records in the same control interval are allowed. Erases are not allowed with concurrent replaces or inserts. This is done to ensure that a backout of an erase will have space available in the control interval for the reinsertion of the logical record.

Any program requiring a control interval split must have the lock on the control interval exclusively.

Erasure of logical records is controlled by the ERASE parameter on the DBD statement in the DFSVSMxx member or the DFSVSAMP data set for batch systems. Specifying ERASE=NO causes deleted records to be replaced, not deleted. These replaced records have a flag set in their prefix indicating that they have been deleted.

The following describes the properties for the block locks:

- **When requested**

  Block locks are requested when a block or control interval is about to be updated. The update may be an insert, delete, or replace of a segment, a change of a pointer or an index entry. Block locks are also requested when an application program uses a Q command code for any segment in the block or control interval.

- **When released**

  These locks are released at application program sync point time.

- **Requestor**

  The requestor is the program that issues a DL/I call causing the block or control interval to be updated.

- **Resource**

  The resource locked is an identification of the block or control interval. For all databases, the resource includes the global DMB number to identify the database and the DCB number to identify the data set in the database. There is also an identification of the block by its relative byte address (RBA).

- **Private attribute**

  The attribute *private* is always used with block locks. This prevents the sharing of these locks by requestors in different IMS systems.

- **Known attribute**

  The attribute *known* is always used with block locks.

- **Level**

  Level 4 is always used for OSAM and VSAM entry sequenced data sets. This allows the lock to be shared by users in the same IMS system. VSAM key sequenced data set data component control intervals are locked at one of following three levels:

  – Level 3 is used for an erase of a logical record.

  – Level 4 is used for a replace of a logical record. Level 4 is also used for an insert of a logical record which does not cause a control interval or control area split.

  – Level 6 is used for an insert of a logical record when it causes a control interval or control area split.

  Table 3-3 summarizes the block lock levels.

Table 3-3 Block lock levels

Data Set Type	Operation	Lock Level
OSAM	All updates	4
ESDS	All updates	4
KSDS	Erase of logical record	3
	Replace of logical record	4
	Insert of logical record	4
	CI/CA split	6

3.5.3 Busy locks

Busy locks are used to serialize certain activities to a database data set. First, they serialize the opening and closing of data sets and the creation of new blocks in them. Second, they prevent updating of a key sequenced data set by a program while another program is inserting into it. This is used to preserve integrity during potential control interval or control area splits.

The following describes the properties for the busy locks:

- **Requested**

  Busy locks are requested when an open, close, or extension of a data set is done. They are also requested when an update to a key sequenced data set is done.

Archived

- **Released**

  These locks are held only during the operation that requires the lock. For example, a lock for open processing is released when the open completes. Similarly, a lock for an insert is released when the insert completes.

- **Requestor**

  The requestor is the program that issues a DL/I call invoking the operation.

- **Resource**

  The resource locked identifies the database data set. This resource has the global DMB number to identify the database and the DCB number to identify the data set in the database. There is also an identification that this is a busy lock.

- **Private attribute**

  The attribute *private* is not used with busy locks.

- **Known attribute**

  The attribute *known* is not used with block locks.

- **Level**

  Lock level 2 is used for non insert updates to key sequenced data sets. Level 8 is used for the other operations. These include opens and closes of data sets, creations of new blocks, and inserts into key sequenced data sets.

  Table 3-4 summarizes the busy lock levels.

Table 3-4 Busy lock levels

Operation	Lock Level
Data set open	8
Data set close	8
New block creation	8
KSDS non insert update	2
KSDS insert	8

3.5.4 Extend locks

Data set extend locks are used to serialize the extension of OSAM and VSAM entry sequenced data set database data sets.

The following describes the properties for the data set extend locks:

- **When requested**

  Extend locks are requested during the data set extension process.

- **When released**

  For hierarchic direct access, and hierarchic indexed direct access method databases, extended locks are released when the extension is completed. For hierarchic indexed sequential access method databases, they are released at application sync point time.

- **Requestor**

  The requestor is the program that issues a DL/I call causing the extension.

Archived

- **Resource**

  The resource locked identifies the database data set. This resource has the global DMB number to identify the database and the DCB number to identify the data set in the database. There is also an identification that this is an extend lock.

- **Private attribute**

  The attribute *private* is always used with extend locks.

- **Known attribute**

  The attribute *known* is always used with extend locks.

- **Level**

  Level 2 is always used for extend locks.

3.5.5 Data set reference locks

Data set reference locks are not used for serialization. Instead, they are used in conjunction with notifications. The data set reference lock is used for notification of data set extensions and buffer invalidations.

The following describes the properties for the data set reference locks:

- **When requested**

  Data set reference locks are requested when a database data set is opened.

- **When released**

  They are released when the data set is closed.

- **Requestor**

  The requestor is the IMS system that opens the database data set.

- **Resource**

  The resource locked identifies the database data set. This resource has the global DMB number to identify the database and the DCB number to identify the data set in the database. There is also an identification that this is a data set reference lock.

- **Private attribute**

  The attribute *private* is not used with data set reference locks.

- **Known attribute**

  The attribute *known* is not used with data set reference locks.

- **Level**

  Level 2 is always used.

3.5.6 Command locks

The command lock is not used for serialization. Instead, it is used in conjunction with notifications. In particular, the command lock is used for notification of commands and write I/O errors.

The following describes the properties for the command locks:

- **When requested**

  The command lock is requested when an IMS system identifies itself to the IRLM. This occurs during IMS initialization.

Archived

- **When released**

  It is released when IMS quits the IRLM, at IMS termination.
- **Requestor**

  The requestor is the IMS system.
- **Resource**

  There is only one resource for the command lock. It identifies this as the command lock.
- **Private attribute**

  The attribute *private* is not used with the command lock.
- **Known attribute**

  The attribute *known* is not used with the command lock.
- **Level**

  Level 2 is always used.

3.6 Fast Path DEDB locking overview

Fast Path also uses locking to provide integrity, but it uses different locks. Its most commonly used locks are on control intervals and units of work (UOWs). They are used both to prevent access to uncommitted updates and to serialize updates to control intervals by different programs. Control interval and unit of work locks are used with and without block level data sharing. Data sharing adds the requirement that locks be known across multiple IMS systems.

When a program accesses a Fast Path control interval, it locks the control interval. Updaters have exclusive use of the control interval. This prevents other programs from accessing uncommitted updates. It also ensures that no other program is making a concurrent update to another copy of the control interval. Sometimes, locks on units of work are used in place of control interval locks. This occurs when certain sequential processes, such as high speed sequential processing (HSSP), are used.

When a request for DEDB data (a CI usually) is made, a lock is obtained. The Fast Path lock manager keeps track of the lock request. When another lock request, for example, from another dependent region, is made for the same resource (CI), the Fast Path lock manager will create another control block representing that request. It will also recognize that another request already exists for that resource. This means that we have contention. The Fast Path lock manager will then call the IMS lock manager (Program Isolation or IRLM) to deal with the contention. This will handle any possible deadlock contention. So IRLM or Program Isolation can be involved even with DEDBs registered at SHRLEVEL(1).

Other lock types are used to coordinate or control processes, such as opens and closes of AREAs and the use of overflow buffer allocations. Fast Path has its own set of locks used for notification.

Fast Path locking in the data sharing sysplex is relevant only for data entry databases. One option to share the data that currently resides in main storage databases (MSDB) is a conversion to DEDB VSO structures. First use the Main Storage Database to Data Entry Database Conversion utility, (DBFUCDB0) as part of the process to convert a main storage database to a data entry database AREA. Then, by initializing the DBRC RECON data set DBDS record parameter VSO, you specify that the DEDB AREA is to reside in virtual storage the next time the control region is initialized or when the next **/STA AREA** command is processed.

AREAs that are defined with SHARELVL(0 | 1) are read into and written from an OS/390 data space. AREAs defined with SHARELVL(2 | 3) use the Coupling Facility to share data between connected subsystems.

The basic unit of interaction with the Coupling Facility that contains the DEDB VSO AREA is the control interval. Segment level locking is not supported for SHARELVL(2/3) with the virtual storage option. Non-shared VSO AREAs still use segment level locking.

The following types of locks are used with data entry databases for block level data sharing.

3.6.1 Control interval locks

Control interval locks are used to prevent access to uncommitted updates and to serialize updates to control intervals.

Control interval locks are not used for:

- Control intervals containing sequential dependents. A different scheme is used to provide integrity with shared sequential dependent segments. The owning IMS gets an update lock on each CI it allocates, it holds the locks until the CI buffer has been filled and the CI is written to DASD. A new X'5958' log record just for SDEP CIs is cut by buffer release IMS task (ITASK): it allows restart not to look endlessly for SDEP CIs that are already on DASD.
- High speed sequential processing, High Speed DEDB Direct Reorganization utility, or the VSO preload process. Instead, unit of work locking is used.
- With program communication blocks whose PROCOPT value is GO. GO causes read without integrity to be performed.

The following describes the properties for the control interval locks:

- **When requested**

  Control interval locks are requested when an application program DL/I call results in a request for a segment in a control interval, or a call that must traverse a root anchor point or segment in the control interval. For example, a GN call may traverse many control intervals and request many control interval locks.

- **When released**

  Control interval locks are released in each of the following four instances:

  – Buffer stealing

    If the buffer containing the control interval is stolen by the buffer manager, the lock is released unless any of the following conditions are met:

    - The control interval contains the root of the current position for the program communication block and the program communication block PROCOPT allows updates.
    - The Q command code has been used to retrieve a segment in the control interval.
    - The control interval contains the root in the root anchor point chain, which precedes the current position, and the program communication block PROCOPT includes D, R, or A.

  – Dequeue call

    A dequeue call releases locks for control intervals unless any of the conditions listed for buffer stealing are met. Unlike full function calls, dequeue calls do not release locks for resources locked by the Q command code.

– Application program sync point

  If the program has not updated the control interval, the lock is released as part of application program sync point processing.

– Output thread completion

  If the program has updated the control interval, the lock is released when the output thread completes. The output thread writes the control interval to DASD or to a Coupling Facility cache structure. Cache structures are used for shared virtual storage option (SVSO) AREAs.

- **Requestor**

  The requestor is the program that issues a DL/I call causing the control interval to be accessed.

- **Resource**

  The resource locked identifies the control interval. The resource ID includes the global DMB number to identify the database, the DCB number to identify the AREA, and the relative byte address of the control interval.

- **Private attribute**

  The attribute *private* is not used.

- **Known attribute**

  The attribute *known* is used with level 8 control interval lock requests. It is not used with level 2 lock requests.

- **Level**

  The level depends on the PROCOPT of the program communication block used to make the call. Level 2 is used with PROCOPT=G program communication blocks. Level 8 is used when the PROCOPT includes either I, R, D, or A. When the PROCOPT includes H, high speed sequential processing is invoked and unit of work locking replaces control interval locking. Control interval locking is not used when the PROCOPT includes GO.

  Table 3-5 presents the control interval lock levels for different PROCOPT values.

Table 3-5 Control interval lock levels

PCB PROCOPT value	Lock Level
G only	2
I,R,D, or A without H	8
H	None
GO	None

3.6.2 Unit of work locks

Unit of work locks are used with high speed sequential processing, the High Speed Data Entry Database Direct Reorganization Utility, and the virtual storage option preload process. They are used in place of control interval locks. Like control interval locks, they prevent access to uncommitted updates and serialize updates to control intervals. Unit of work locks are used instead of control interval locks to reduce the number of locks requested.

Like control interval locks, unit of work locks are not used with sequential dependents and are not used when GO is used for the program communication block PROCOPT value. Unit of work locks are requested by four users:

- High speed sequential processing

  Unit of work locks are requested by application programs when using a high speed sequential processing program communication block. These are program communication blocks with PROCOPT values which include H.

- High Speed Data Entry Database Direct Reorganization utility

  They are requested by the reorganization utility.

- Virtual storage option preload

  They are requested by the preload process when a virtual storage option (VSO) AREA is opened with this option.

- Other programs, when high speed sequential processing, the reorganization utility, or the virtual storage option preload process is active.

  When a high speed sequential processing program, the reorganization utility, or virtual storage option preload is active for a data entry database AREA, all other programs accessing the AREA request unit of work locks for the AREA. These other programs also request control interval locks.

The following describes the properties for the unit of work locks:

- **When requested**

  A unit of work lock is requested when the program, utility, or preload process first accesses a control interval in the unit of work.

- **When released**

  High speed sequential processing programs release a unit of work lock when they would have released locks on all of the control intervals in the unit of work if high speed sequential processing were not being used. The reorganization utility and virtual storage option preload release a unit of work lock when their processing has moved past the unit of work. Other programs release a unit of work lock when they release all of their locks on control intervals in the unit of work.

- **Requestor**

  The requestor is the program that issues a DL/I call causing the unit of work to be accessed. For the reorganization utility, the requestor is the utility. For virtual storage option preload it is the preload process.

- **Resource**

  The resource locked identifies the unit of work. The resource designator includes the global DMB number to identify the database, the DCB number to identify the AREA, and the relative byte address of the first control interval in the unit of work.

- **Private attribute**

  The attribute *private* is not used.

- **Known attribute**

  The attribute *known* is used with level 8 lock requests. It is not used with level 2 lock requests.

- **Level**

  High speed sequential processing, the reorganization utility, and virtual storage option preload always request Level 8 for the unit of work lock. Other programs always request Level 2 for the unit of work lock.

  Table 3-6 presents the unit of work lock levels for different requestors.

Archived

Table 3-6 Unit of work lock levels

Requester	Lock Level
High speed sequential processing	8
Reorganization utility	8
Virtual storage option preloaded	8
Other programs	2
Programs with PROCOPT = GO	None

3.6.3 AREA locks

AREA locks serialize open and close processing for data entry database AREAs.

The following describes the properties for the AREA locks:

- **When requested**

  AREA locks are requested when an AREA is opened and closed. Whenever the second CI within the DEDB (where the DMAC control block resides), is modified, the AREA lock is obtained.

- **When released**

  These locks are released at the end of open and close operations.

- **Requestor**

  The requestor is the IMS system that opens or closes the AREA.

- **Resource**

  The resource locked identifies the AREA. The resource designator includes the global DMB number to identify the database and the DCB number to identify the AREA.

- **Private attribute**

  The attribute *private* is not used with AREA locks.

- **Known attribute**

  The attribute *known* is used with AREA locks.

- **Level**

  Level 6 is used.

3.6.4 AREA notify locks

AREA notify locks are used in conjunction with notifications. For example, they are used in sending notification of I/O errors.

The following describes the properties for the AREA notify locks:

- **When requested**

  AREA notify locks are requested when an AREA is opened.

- **When released**

  These locks are released when an AREA is closed.

- **Requestor**

  The requestor is the IMS system that opens the AREA.

- **Resource**

  The resource locked identifies the AREA. The resource designator includes the global DMB number to identify the database and the DCB number to identify the AREA.
- **Private attribute**

  The attribute *private* is not used with AREA notify locks
- **Known attribute**

  The attribute *known* is not used with AREA notify locks.
- **Level**

  Level 2 is used.

3.6.5 Fast Path command lock

The Fast Path command lock is not used for serialization. Instead, it is used in conjunction with notification. One example is for the IMS reconnection to a failed and restarted IRLM.

The following describes the properties for the Fast Path command locks:

- **When requested**

  The Fast Path command lock is requested when an IMS system with Fast Path capabilities identifies itself to the IRLM. This occurs during IMS initialization and reconnect.
- **When released**

  This lock is released when IMS quits the IRLM. Quit occurs at IMS termination.
- **Requestor**

  The requestor is the IMS system.
- **Resource**

  There is only one resource for the Fast Path command lock. It identifies this as the Fast Path command lock.
- **Private attribute**

  The attribute *private* is not used with the Fast Path command lock.
- **Known attribute**

  The attribute *known* is not used with the Fast Path command lock.
- **Level**

  Level 2 is always used.

3.6.6 Fast Path buffer overflow locks

Fast Path buffer overflow locks are used to serialize the use of overflow buffers in IMS systems. Only one region per IMS system may exceed its normal buffer allocation at any time. These locks are used in this serialization process.

The following describes the properties for the Fast Path buffer overflow locks:

- **When requested**

  These locks are requested when a region needs to exceed normal buffer allocation limit.
- **When released**

  These locks are released when the region gives up its overflow buffer allocation at sync point time.

Archived

- **Requestor**

  The requestor is the region that needs to exceed its normal buffer allocation limit.

- **Resource**

  The resource locked is an identification of the IMS system and an identification that this is the overflow buffer lock for this system.

- **Private attribute**

  The attribute *private* is not used with buffer overflow locks.

- **Known attribute**

  The attribute *known* is not used with buffer overflow locks.

- **Level**

  Level 8 is used.

3.6.7 Summary of Fast Path locking

Table 3-7 shows a summary of the locks that are obtained depending on the IMS call and processing option used in Fast Path environments.

Table 3-7 Fast Path locking, depending on the IMS calls and PROCOPT

IMS CALL	PROCOPT				
	G	**R,I,D,A**	**GOx**	**GH**	**H**
Get	Share level global control interval block Held until sync point, dequeue call, or buffer steal	Update level global control interval lock Held until sync point, dequeue call, or buffer steal	No locking	Share level global unit of work lock Held until sync point or dequeue call	Update level global unit of work lock Held until sync point or dequeue call
Insert Replace Delete	Not applicable	Update level global control interval lock Held until output thread processing completes	Not applicable	Not applicable	Update level global unit of work lock Held until output thread processing completes

3.7 Specific use of locks and their effect in data sharing

Additional locks are obtained by IMS to maintain recoverability and integrity. Locks are obtained when searching for free space, maintaining database pointers, communicating global notifications between subsystems, serializing control interval/control area splits, and extending databases.

Block locks for hierarchic direct space search

The block lock is used when searching for free space in a block level data sharing environment. If no free space is found, the lock is released immediately. If free space is found, the lock is held at Level 4 private until a sync point is reached.

When a bit map in hierarchic direct access method or hierarchic indexed direct access method databases is updated, a global lock is placed on the bit map block or control interval. Therefore, all access to a bit map is serialized across multiple systems. With the hierarchic

direct space search algorithm, even when the bit map is referenced, the block or control interval is locked.

Space reclaim and key sequenced data sets

In the block level data sharing environment, KSDS data component CIs for the HISAM primary data set, HIDAM index, and any secondary index are protected by global block locks:

- A level 3 private lock is obtained for the erase of a key sequenced data set record.
- A level 4 private lock is obtained for the insertion or replacement of a KSDS record.
- A level 6 private lock is obtained for a record insert resulting in a KSDS CI, or CA split.

High update activity to KSDSs

Updates to a KSDS require the busy lock for the data set. This lock is held only for the duration of the update, however, it can lead to a high volume of lock requests. Inserts require a level 8 (exclusive) lock and will be single threaded with other updates.

Updates also require a block lock on the data component CI. These are never shared across subsystems and are held until application sync point time.

Processes which do many updates to KSDSs should be investigated for potential increased lock processing. Batch job scheduling might need to be adjusted to avoid conflicts.

Block locks and hierarchic indexed access method pointers

If TWINBWD is specified on the POINTER = keyword in the SEGM DBD statement then physical twin forward and backward pointer fields are created and maintained. This option is recommended for HIDAM and PHIDAM database root segments, because it provides improved segment delete performance.

Because of the maintenance of twin backward and forward pointers, whenever a root is inserted or deleted in a data sharing environment, block locks are also obtained for the blocks for the preceding and following roots. This could become a data sharing performance issue.

Locks with the Q command code– full function databases

When a Q command code is issued for a root or dependent segment, a Q command code lock at share level is obtained for the segment. This prevents sharing subsystems from updating the buffer until the Q command code lock is released. The locks are released with a dequeue call or at sync point time.

3.8 IRLM Version 2 Release 1

Internal Resource Lock Manager (IRLM) component provides locking services to each IMS subsystem in a data sharing group.

Data sharing is implemented by the provision of IRLM, functioning with each IMS subsystem, and by the transfer of the responsibility for the locking of data during IMS updates from the IMS itself to the IRLMs. Each IRLM communicates with the other IRLMs by means of the Coupling Facility lock structure to ensure that lock usage is coordinated in intersystem block level data sharing environments. IRLM is responsible for not only the serialization of database updates but deadlock control environmental recovery in the event of a failure of a client of IRLM services.

Even if IRLM is only used with the SCOPE=LOCAL startup parameter, the following locking changes will occur:

- Segment locks for full function databases will be eliminated.
- Deadlock detection will be done on a timer basis. This will cause involved locks to be held longer and might increase the contention for those locks.

These changes could cause increased lock contention.

3.8.1 Deadlock control

A deadlock occurs in the following situation:

1. Program A holds a lock on ROOT A.
2. Program B holds a lock on ROOT B.
3. Program A now requests a lock on ROOT B and is required to wait for the lock.
4. While Program A is waiting for ROOT B, Program B requests a lock on ROOT A.
5. Program B is required to wait for the lock.
6. A deadlock occurs within a commit interval where neither program A or B will give up its locks, because neither program can commit its data until it gains access to the lock it is waiting for. So each program continues waiting unless the lock manager intervenes.

IRLM chooses one of the programs to rollback with a user Abend 777. This will release that program's locks on completion of the rollback process. Analyze your deadlocks using the DEADLOCK report produced by DFSERA10 with the DFSERA30 exit.

There can be many more than two programs involved in a deadlock ring. When choosing a deadlock victim, program isolation (PI) or IRLM looks at all of the programs or transactions involved in the deadlock. It attempts to choose the victim that will be least disruptive to the system:

- If the participants are of the same type, such as two non message driven BMPs, then one that has been running the shortest time since its last sync point or its start is chosen. This minimizes the amount of work that must be backed out and reprocessed.
- If the participants are of different types, program isolation or IRLM chooses the one closest to the top in the order of the following list:
  a. Fast Path message driven program
  b. Message processing program with MODE=SNGL
  c. Message processing program with MODE=MULT
  d. CICS task
  e. Batch (DLI or DBB) program, not updated
  f. Explicit APPC/IMS (CPI-C) program
  g. Message-driven batch message processing with MODE=SNGL
  h. Non-message-driven batch message processing
  i. Message-driven batch message processing with MODE=MULT
  j. Fast Path online utility or batch (DLI or DBB) update program. This is last to be selected as a deadlock victim.

When you are using program isolation for lock management, deadlock detection occurs every lock request. This is not true in the case of IRLM and its start up parameter DEADLOK.

In the following example, the DEADLOK parameter is defined as:

```
DEADLOK=(3,1)
```

The first number of the DEADLOK parameter represents the number of seconds to wait between local deadlock detection cycles. Using the DEADLOK=(3,1) sample, each IRLM performs deadlock detection once every 3 seconds instead of every lock request. The theory is that it is probably not a deadlock and if you just wait a few seconds it will resolve itself. If no deadlock is detected, then no abend will be issued. The fundamental difference between program isolation (PI) locking and IRLM is that PI prevents deadlocks, whereas IRLM detects them. It would be too expensive to do prevention across IRLMs. The drawback of setting a high local deadlock detection value is that if two requestors are really deadlocked, the period until the deadlock is resolved is lengthened.

All IRLMs check for deadlocks among all IMS subsystems in the data sharing sysplex, based on the global deadlock detection time, specified as the second value of the DEADLOK parameter in the IRLM start up procedure. The global value specifies the number of local cycles that must complete before global detection is initiated.

There have been two enhancements with IRLM Version 2.1 DEADLOK settings:

- The original design of IRLM forced the IRLMs participating in the data sharing group to synchronize their deadlock parameters to the highest DEADLOK values of any member of the group. This means that all IRLMs must be disconnected from the group in order to lower the values. With IRLM Version 2.1 APAR PN69636, however, the DEADLOK values used by all members of the group can be changed up or down by starting a new IRLM with the values desired. Deadlock processing will always use the values from the newest IRLM to join the group. Therefore, it is important to control the scheduling of IMS subsystems so that, for example, a batch region with different DEADLOK values in its associated IRLM start up procedure does not set deadlock cycle values that would degrade online operations.
- With IRLM Version 2.1 APAR PQ44791, IRLMPROC DEADLOK values for the local deadlock frequency can be specified in milliseconds. You can also use the following command to dynamically change the deadlock frequency:

  ```
  MODIFY irlmproc,SET,DEADLOK=
  ```

  With the speed of processors increasing there is a requirement to execute the deadlock detection more frequently. When average transaction times of 100 to 200 milliseconds exist, a deadlock can cause delays and large wait chains to occur.

IRLM V2.1 internally sets the first DEADLOK parameter at a maximum of 5 and the second only to a value of 1 regardless of what you code.

Recommendation: The same values for the DEADLOK parameter should be specified in all IRLM subsystems in the data sharing sysplex, if you don't want the highest values to become the default for all IRLMs. At APAR PN69636 level, if different DEADLOK values have been specified in supported IRLMs for each IMS subsystem, scheduling controls must be introduced since it is the latest IRLM to join the group that sets the DEADLOK values.

3.8.2 Storing lock information

The information pertaining to all locks (shared or update) granted by a particular IRLM is stored within that IRLM on the OS/390 image where the IRLM is running.

In addition, every lock granted by an IRLM in a data sharing group will have an entry in the lock table portion of the lock structure, in the Coupling Facility.

Locks protecting updates will be stored in the record list section of the lock structure.

3.8.3 The lock structure in the Coupling Facility

A locking structure consists of two portions:

- A lock table
- A record list

IRLM by default divides the storage allocated for a locking structure equally between the lock table and the record list.

NOTE: Other IMS documentation may refer to the *lock hash table* and the *modify lock table*, but we use the current terminology of *lock table* containing *lock table entries (LTE)* and *record list* containing *record list entries (RLE)*.

However, because the lock table size must be in a power of 2, the storage used for it may be much larger than necessary based on the rate of global false contentions. This leaves the remainder of the allocation to the RLE storage. Clients having large numbers of locks are then required to create very large structure allocations in order to get the required number of RLE entries. If they could dictate how many LTEs they wanted, the structure could be more efficiently used.

Via IRLM Version 2.1 APAR PQ44114, the HASH= parm in the IRLMPROC has been introduced to set the number of LTEs entries in the CF. This occurs when the first IRLM connects, causing structure allocation. Otherwise there is a 1:1 split between LTE and RLE storage allocations. The parm HASH= has a value between 0 and 1024 and represents the number of Lock Hash entries available in the CF. The value must either be zero or a power of two with each increment representing 1048576 entries.

Also the following command can be used to set the size of the lock table:

```
MODIFY irlmproc,SET,HASH=
```

The size of the lock structure, in comparison to the number of locks currently held by participating IRLMs, determines the number of synonyms in the lock structure. Synonyms are lock requests that hash to the same entry. Adequate sizing of the lock structure will minimize the number of false contentions. See 3.8.4, “Following a lock request” on page 69 for an explanation of false contentions.

The lock table portion of the lock structure

The lock table entries indicate which IRLMs have a lock hashing to the table and whether any IRLM has an update lock hashing to that entry. Lock granting and hash ownership checking both use the lock table.

Access to the CF structures is provided by a component of OS/390 called *cross-system extended services* (XES). Each entry in the lock table contains one byte plus one bit for each IRLM, and its XES locking services using the structure. The settings of the bits indicate whether the corresponding IRLM holds at least one lock which hashes to this entry in the table.

The lock table is used to minimize the amount of IRLM-to-IRLM communication needed to grant a lock. For example, if the lock table indicates that no IRLM has an update lock hashing to a particular lock table entry, then when an IRLM wants to grant a shared lock for a resource that hashes to the entry, it can determine that no IRLM-to-IRLM communication is required.

If, on the other hand, the lock table indicates that IRLM-to-IRLM communication is required to grant a lock, the lock table indicates which IRLMs need to be involved. This limits the number of IRLMs that need to communicate about a lock request.

The record list portion of the lock structure

The record list consists of record entries for uncommitted IMS updates.

Whenever an application program uses a program communication block with update intent, the lock request on the database record level lock is flagged with a modify attribute. If any segments are updated by this application, the block level lock will be flagged with a modify as well as a force attribute. This will cause IRLM to write modify locks in the record list portion of the lock structure. These include some full function database record locks, all full function block locks, and some Fast Path control interval locks.

The record list portion of the locking structure is not used to assist in the locking process; instead, it exists for availability reasons.

If the only copy of lock information protecting updated resources was in the IRLM, then that information would be lost in an IRLM failure. Since lock information would be lost, an IRLM failure would cause all IMS subsystems in the data sharing group to stop accessing portions of shared databases. The record list portion of the Coupling Facility lock structure is needed to prevent that loss.

In the event of a failure of a CPC, or IRLM, all surviving IRLMs read the record list entries for the failed IRLM and construct in-storage retained locks. Should an application request one of these retained locks, the lock request is rejected by the local IRLM with a U3033 pseudo-abend (unless the application has been coded to cater for unavailable data).

3.8.4 Following a lock request

Figure 3-2 presents the components of the Coupling Facility lock structure used by IRLM and a lock request activity flow.

Archived

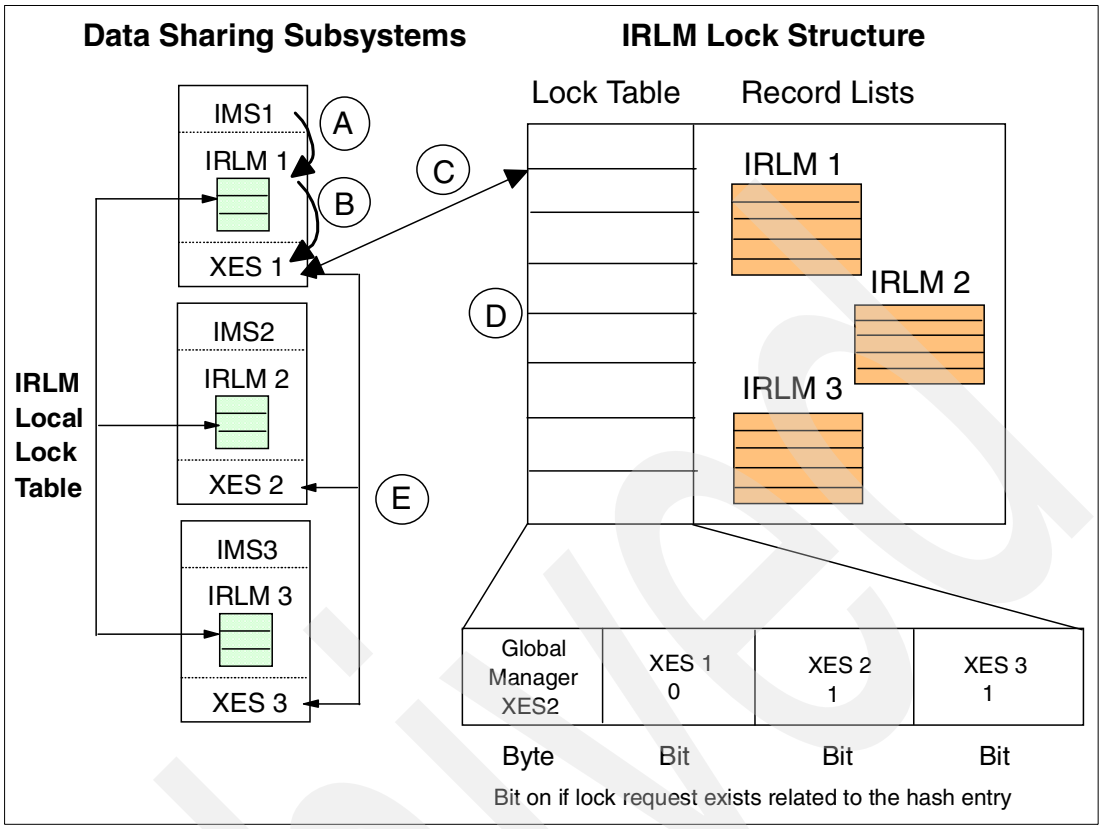


Figure 3-2 IRLM and Coupling Facility lock structure components and process flow

The following flow occurs when an IMS subsystem requests a lock from IRLM. The key letters A through E on IRLM and Coupling Facility lock structure components represent the stages of the lock request process.

- STAGE A

  The IMS1 subsystem issues a request to IRLM1.

- STAGE B

  IRLM1 checks compatibility with its own local locks. If there is a local locking conflict, the local IRLM will handle the resource request enqueue.

- STAGE C

  If there is no local locking conflict, IRLM1 passes the request to XES locking services to access the lock structure in the Coupling Facility to determine if another IRLMs lock ownership contends with this request.

- STAGE D

  Cross-system extended services accesses the Coupling Facility lock table and updates it if necessary. XES compares the LTE with the request and, if there is no contention, the lock is granted.

  When a block lock is obtained, IRLM writes RLEs within the record list portion of the lock structure, for the database record level lock, as well as for the block lock. This is a synchronous activity and the IMS task control block under which this activity is performing is set at 100% central processing unit state for the duration. Subsequent requests for the lock are handled locally by this IRLM unless another system has expressed an interest in the lock in the meantime.

- STAGE E

  If there is contention; in other words, if there is another lock that has randomized to this entry, XES uses cross-system coupling facility (XCF) communication with XESs in the other systems to determine if, in fact, an incompatible request (a request for a lock on the same resource) has occurred:

  If so, then the request is queued, and must wait.

  If it is determined that another system holds a lock for another database record that just happened to randomize to the same entry, then the lock request is granted. This is known as *false contention*.

  It is important to note that requests for a lock, whose lock table entry is in contention, are suspended until it is determined that there is no real lock contention on the resource.

The Coupling Facility is also accessed to release locks. A database record lock is individually released if the application moves to a new record without updating the previous record. This requires an update of the lock table in the Coupling Facility. Update locks and any remaining non update locks are released as a group at sync point time or when OTHREADs complete for Fast Path DEDB updates. This requires both the lock table and the record list table in the Coupling Facility to be updated.

3.9 The buffer invalidation process

The process of buffer invalidation prevents the use of old buffered copies of blocks or control intervals by IMS data sharing partners after the physical block or control interval has been updated by one IMS subsystem.

The OSAM or VSAM cache structures in the Coupling Facility are used to determine which IMS systems have a copy of the block or CI that needs to be invalidated. The relationship of databases, buffers, and Coupling Facility structures shown in Figure 3-3 illustrates the buffer invalidation components.

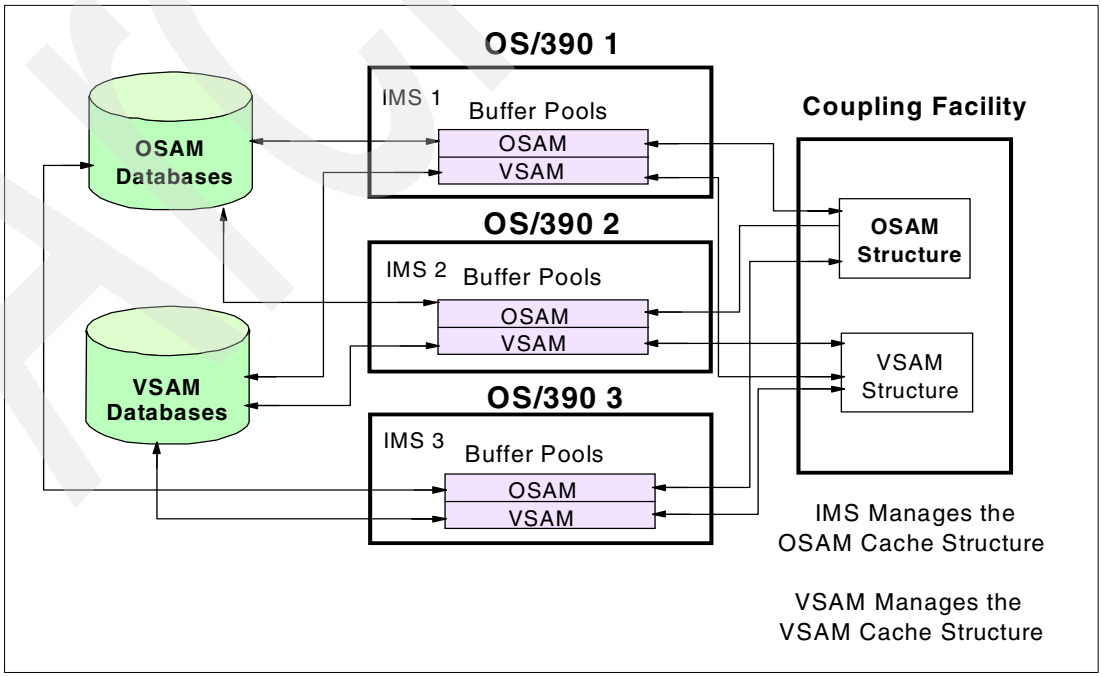


Figure 3-3 Relationship of databases, buffers, and Coupling Facility structures

3.9.1 OSAM and VSAM buffer invalidation using XES services

OSAM and VSAM buffer invalidation structures are built in the Coupling Facility. For each block or control interval of shared data read by an IMS from the sharing group, an entry is made in the OSAM or VSAM structure. Each entry consists of a field for the buffer ID and 255 slots for the IMS subsystems to register interest in an entry's buffer. When an IMS subsystem reads a block or control interval, a bit in the hardware system area (HSA) is turned to “valid” and the interest is registered in the Coupling Facility as a block-to-buffer relationship.

When an update to a block or control interval is committed to DASD, buffer cross invalidation takes place. For example, using Figure 3-3, if IMS1 updates a control interval in a VSAM database that IMS3 has read into its private buffers, buffer cross invalidation must occur:

1. IMS1 updates the control interval. This control interval is contained in both IMS1 and IMS3 buffer pools.
2. The buffered control interval in IMS3 will still be marked as valid but, because of IRLM, the updated block will be protected against updates. Other database records in this block are still available to IMS3.
3. IMS1 writes the updated control interval on DASD and then asks the Coupling Facility, via cross-system extended services, to invalidate the control interval in any IMS which has a copy of the control interval.
4. The Coupling Facility changes the bit structure in the hardware system area (HSA) to indicate that this control interval is invalid. It communicates with the necessary OS/390 images; OS/390 3 in this case, rather than all.
5. If IMS3 accesses the CIs data again, the updated CI will have to be read back into IMS3’s buffer pool when IMS3 attempts to access the CI.

The Coupling Facility updates the bit structures for the other IMS subsystems with the copy of the CI, even if they are on different LPARs. The IMS subsystem doing the invalidation has its task control block put into a “CPU Busy” state, but not put into a wait or suspended state.

The number of I/O operations per transaction in a single system image is a function of local buffer pool hits; the higher the hit ratio, the lower the I/O rate.

When moving to a Parallel Sysplex, I/O’s per transaction become a function of the local buffer pool hit ratio and the global buffer pool hit ratio. The expectation is that local buffer pool hit ratios will be lower in a Parallel Sysplex than in a single system image, primarily because of buffer cross invalidations caused by updates on other systems (an update on System B causes invalidation of the local buffer on System A). The number of misses to a local buffer pool will typically increase by 1% to 5% per system added to the Parallel Sysplex. This means an increase in I/O of between 1% and 5% per transaction.

3.9.2 Buffer invalidation and Fast Path DEDBs

Unlike full function OSAM or VSAM databases, shared non-VSO DEDBs do not require any cache structure. This is for two reasons. First, all locking for DEDBs is at the CI level, not the record level. This means that even a get call will request a CI lock and will wait if any other IMS already holds an incompatible lock on this CI. So, even if two programs in two different IMSs ask for different records in the same CI, one will wait. Second, shared non-VSO DEDBs do not use look-aside buffering as do OSAM and VSAM. So, whenever a requestor gets a lock on a CI, fast path does not look in the buffer pool for a copy that may already exist. Fast path always reads that CI from DASD, therefore always getting a valid copy. Another way of looking at this is that, whenever an application releases its lock on a fast path CI, that buffer is always invalidated locally — no need for a cache structure to do this.

Shared VSO, on the other hand, has its own buffer pools just for the use of the shared VSO AREAs. The user has the option of declaring these buffer pools to be look-aside or non-look-aside. Since buffer invalidation is really a function of XES, the Coupling Facility Control Code, and the structure, whenever an updated buffer is written to the SVSO structure, whether it is from a look-aside buffer pool or not, the corresponding bit in the local cache vector(s) is changed to indicate that the buffer is invalid.

3.10 The IMS interface to the notification process

The notification function is used for IMS subsystem to IMS subsystem data sharing communication. It is used to broadcast information pertaining to:

- Global database commands
- Database write errors
- Data set extensions

3.10.1 Global database commands

Database commands (**/START, /STOP, /DBRECOVERY, /DBDUMP**) have an optional parameter, GLOBAL. When this optional parameter is specified, all other IMS online subsystems participating in the data sharing sysplex will be sent a copy of the global command from the subsystem that issued the command using the XCF interface. The command will then automatically be issued on the receiving subsystem.

IMS Version 8 and the single point of control (SPOC) feature provides an ISPF application running under TSO that provides a location from which IMS commands may be entered by a person to one or more members of an IMSplex.

3.10.2 Database write errors

When a database write error occurs, the subsystem that sustained the write error must notify all other subsystems that the write error occurred, so that they can update their internal tables, and understand that any subsequent request for the block or control interval that is in error cannot be honored.

In a full function environment, the DBRC RECON DBDS record is updated with EEQE information, the recovery needed flag in the DBDS record is turned on, and the recovery needed counter in the DB record is incremented.

For Fast Path in IMS Version 6, each failing CI is marked by an error queue element (EQE). The EQE list will be updated in the AREA data set control block (ADSC). The IMS sharing partners are notified that the CI is read inhibited.

IMS Version 7, without MADS, exploits the I/O Toleration (IOT) function that was previously used only by full function databases, to improve the availability of data in a DEDB. IOT is used to store data from CIs that experience write I/O errors in IOT buffers which are kept in storage and maintained across IMS restarts until the AREA concerned is recovered. A rewrite of the CI is attempted:

- At each IMS checkpoint
- At every **/STO**, **/DBR**, or **/VUN** command

If the rewrite is successful, the EEQE indicator in the DBDS and IOT buffer is cleared.

In IMS Version 7, if MADS are available for the AREA, each failing CI is marked by an EQE, not an extended error queue element (EEQE), for the specific data set but the CI is not

flagged with an EEQE unless all copies are bad. An EQE is not converted to an EEQE until the last copy of the CI fails the write operation.

3.10.3 Data set extensions

An IMS subsystem determines the high-use RBA of a database data set at database OPEN time via information in the VSAM catalog or the format 1 DSCB field DS1LSTAR. If the high-used relative byte address is changed as a result of activities on another subsystem, this subsystem must be informed of the change so that it can maintain its own internal data set of related control blocks.

The data set extension notification process accomplishes this task. During this period, a level 2 data set reference lock is obtained.

3.10.4 The Coupling Facility and notifications

Notification processing continues to be handled by the IRLM. However, the IRLMs use cross-system coupling facility (XCF) signaling to communicate events to each other (notification process). The OS/390 configuration decides whether XCF will use the Coupling Facility. However, the use of the Coupling Facility by XCF is recommended, especially as the number of OS/390 images in the data sharing sysplex increases.

Recommendation: The use of channel-to-channel facilities by XCF is recommended to ensure better performance until the number of sysplex partners becomes very large. The management benefits associated with use of the Coupling Facility by XCF then begin to outweigh the channel-to-channel performance benefits.

3.11 Components involved in preserving database integrity

Now that the material in this chapter has been presented, we can follow the logical flow of multi layered integrity control, as DL/I calls from an application cause processing in several data sharing components.

Table 3-8 contains an example of how block level data sharing facilities work together to maintain integrity. The example is for an application program with a program communication block (PCB) with PROCOPT=A, accessing a full function database. The segments to be retrieved are not in buffers in this example and have to be retrieved from DASD.

At sync point time, locks are released once buffer invalidation has taken place.

Table 3-8 An example of block level data sharing integrity facilities

Related Activities				
IMS Call	**IRLM on OS/390 image**	**Lock structure in CF**	**VSAM/OSAM buffer pool**	**VSAM/OSAM cache structure in CF**
Get Unique ROOT A	Obtain database record lock on ROOT A	Create lock entry in lock table	Retrieve control interval/block from DASD Set bit associated with buffer to valid	Register interest in buffer

Related Activities				
Get Unique ROOT B	Release database record lock on ROOT A Obtain database record lock on ROOT B	Release lock hashed entry for ROOT A Create lock entry in lock table for ROOT B	Retrieve control interval/block from DASD Set bit associated with buffer to valid	Register interest in buffer
GHNP	No additional locks required	No additional lock entries required	Retrieve dependent segment CI/block from DASD Set bit associated with buffer to valid	Register interest in buffer
REPL DEP1	Obtain block lock for control interval or block containing DEP1	Create record list entry for database record lock, ROOT B Create record list entry for block lock, DEP1	Update control interval/block in buffer pool	No activity
SYNC POINT	Release database record lock for ROOT B Release block lock for DEP1	Release lock hashed entry for ROOT B Release lock hashed entry for DEP1 Release record list entry for database record lock on ROOT B Release record list entry for block lock for DEP1	Write updated control interval/block to DASD	Perform buffer invalidation if necessary

This concludes our overview discussion of data sharing integrity components.

Archived

Archived

4

Additional data sharing facilities

There are other block level data sharing enhancements that may be selected for use based on need. They are:

- OSAM Coupling Facility data caching
- Fast Path DEDB VSO data sharing (SVSO)
- Shared sequential dependent segments
- Fast Database Recovery (FDBR)

We explain how these features function to improve data availability in your sysplex environment.

Archived

© Copyright IBM Corp. 2003. All rights reserved.

4.1 OSAM Coupling Facility data caching

OSAM CF data caching provides the support to use a Coupling Facility for the writing and reading of database buffers. This support is an enhancement of the N-way data sharing processing implemented first in IMS Version 5 where the Coupling Facility was used for OSAM buffer invalidations and did not cache database buffers.

4.1.1 Why use OSAM CF data caching

The main reason for using CF data caching is to avoid performance problems associated with re-reading OSAM blocks following buffer invalidations. This is most useful for small databases with heavy update activity or any database with heavy update activity to hot spots. It offers little performance benefit to use OSAM Coupling Facility caching for databases with little or no re-read processing.

This support allows the user the option to specify the caching of OSAM database buffers. The user may request the caching of all database buffers, by writing the data buffer to the Coupling Facility after it has been acquired from DASD, or just the caching of changed data. The latter option writes data buffers, that have been altered and are being written back to DASD, to the Coupling Facility afterwards. The caching option is specified on a subpool basis. Databases or database data sets may be assigned to specific subpools which have been defined with or without a caching option.

4.1.2 Requirements for OSAM CF data caching

The following elements have to be present for OSAM CF data caching:

- Databases registered at SHARELVL (2|3)
- The use of IRLM Version 2 Release 1
- Available Coupling Facility structures of the proper size

4.1.3 The store-through cache structure

The space in this cache structure is divided between directory entries and data entries:

- Each directory entry is about 200 bytes long and contains information about data items shared among users:
  - The name of the data item (RBA + global DMB# + DCB#). The RBA is four bytes in length with the global DMB# allocated to two bytes and the DCB# of length one byte. There are nine bytes of zeros after the DCB#.
  - Which users registered interest in the data item.
  - If data is in the user's buffer, what buffer it is in.

The data item may be in the cache structure, in any data sharing group user's local buffers, or both, so the directory entries are required for buffer invalidation. The number of directory entries is determined by the size of the structure and the DIRRATIO and ELEMRATIO parameters on the CFNAMES statement. It is advisable to have an entry for each buffer, but this will not occur unless the user specifies it.

- The data entry is between 2 KB and 32 KB, because it is used to store the OSAM block data item:
  - There may be between 1 and 16 2-KB data elements in the data entry, depending on the OSAM block size.
  - Data is written to DASD before it is stored in the structure.

– Data items will be read from the cache instead of DASD.

The calculation of how large a portion of the cache structure to allocate for the data elements is based on how much data you want in the cache.

The size of the OSAM structure is calculated as shown in Figure 4-1.

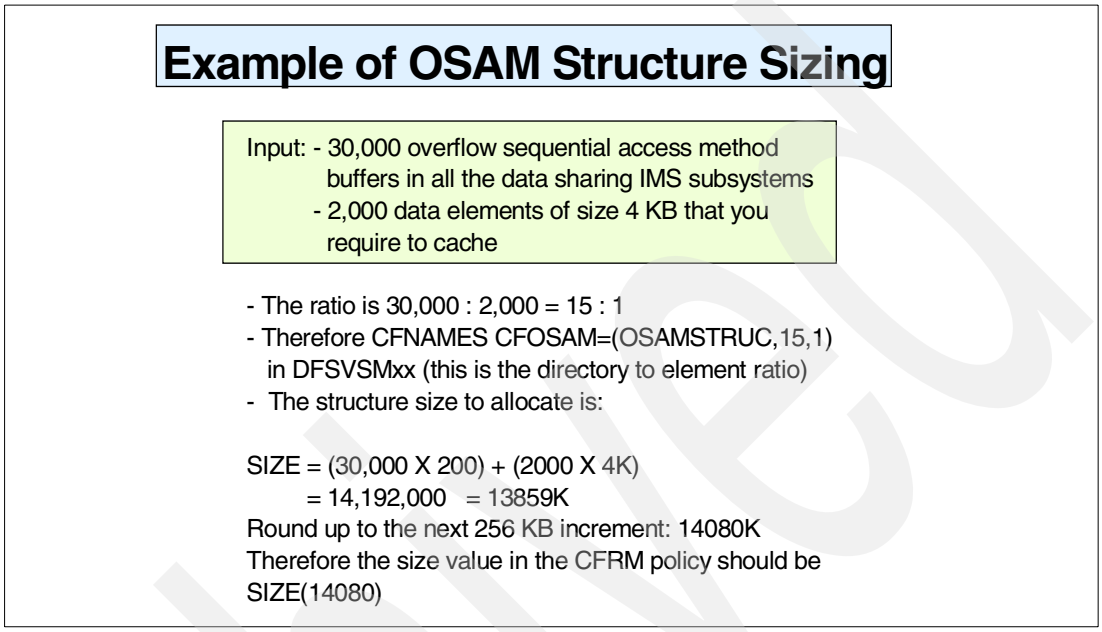


Figure 4-1 Example of OSAM structure sizing

4.1.4 Requesting data caching

Several parameters are necessary to invoke OSAM DB CF caching:

- In the IOBF subpool definition statement in DFSVSMxx, there is a caching option parameter CO, which can be set to:

  N = No data caching for this subpool.

  A = Cache all data when read from DASD and all changed data to the CF.

  C = Cache CHANGED data to the CF after it has been written to DASD.

  Specifying N for no data caching does not affect the allocation of storage in the OSAM structure. It affects only whether IMS writes data to the structure.
- A CFRM policy defines the OSAM cache structure to be activated by IMS.
- The structure names to be defined to IMS are contained in the CFOSAM parameter in DFSVSMxx as part of the CFNAMES statement. The CFOSAM calculation associated with setting these new parameters is presented in 4.1.3, “The store-through cache structure” on page 78.

The CFOSAM parameter in CFNAMES statement:

```
CFNAMES ....,CFOSAM=(OSAMSTRUC,dirratio,elemratio)
```

4.1.5 An OSAM cache modification example

To illustrate how OSAM buffer caching operates, we lay out the flow of a read and modify of an OSAM block with two IMS data sharing subsystems expressing interest in the data. Figure 4-2 is the first diagram to demonstrate how the OSAM CF structure operates with IMS data sharing partners accessing OSAM databases.

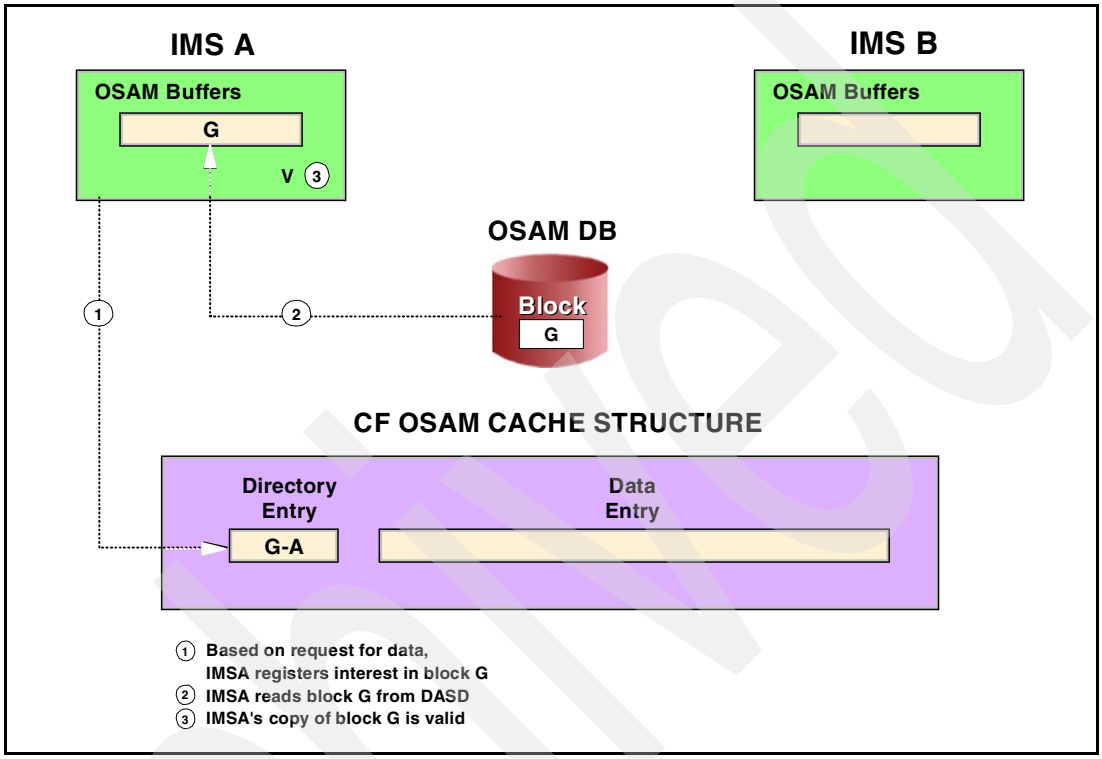


Figure 4-2 OSAM cache read and update data: diagram 1

The OSAM block has been read into IMSA private buffer pool and the CF OSAM structure directory entry has been updated to indicate that IMSA has interest in block G. IMSB has not yet performed any operations.

Figure 4-3 on page 81 shows IMSB accessing the block G and the Coupling Facility OSAM structure directory entry being updated.

Archive

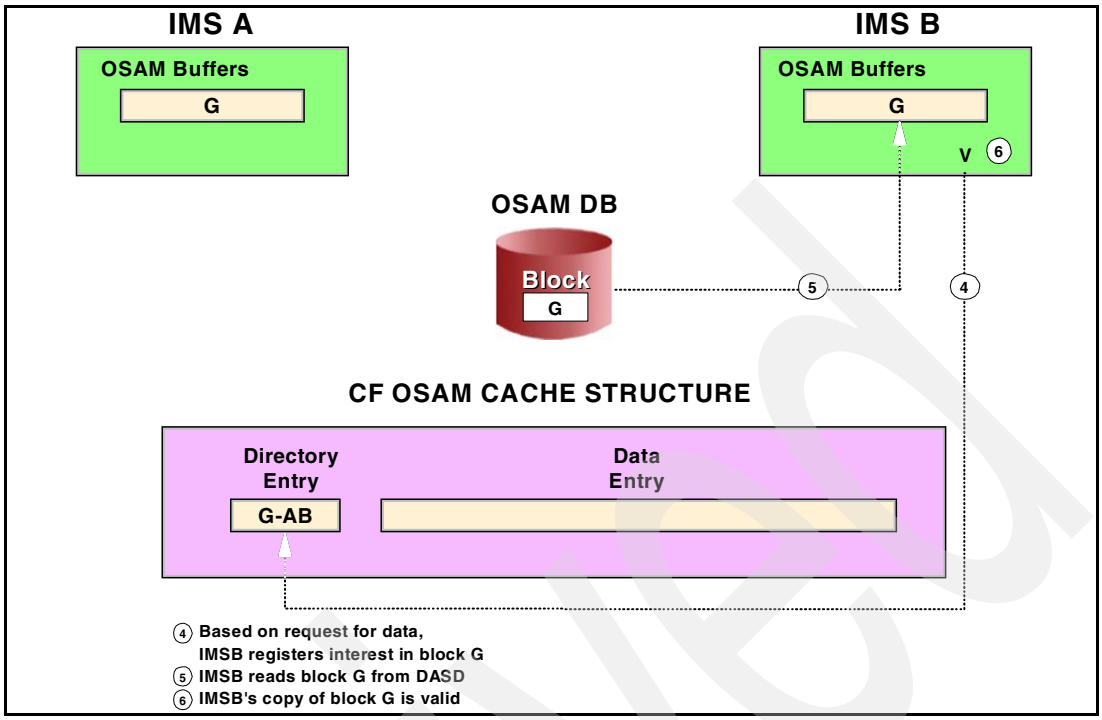


Figure 4-3 OSAM cache read and update data: diagram 2

At this point, block G is contained in both private buffer pools within IMSA and IMSB. The directory entry in the OSAM structure shows that there are two data sharing partners with interest in this resource.

Figure 4-4 shows IMSA updating block G and the CF OSAM structure directory entry being updated.

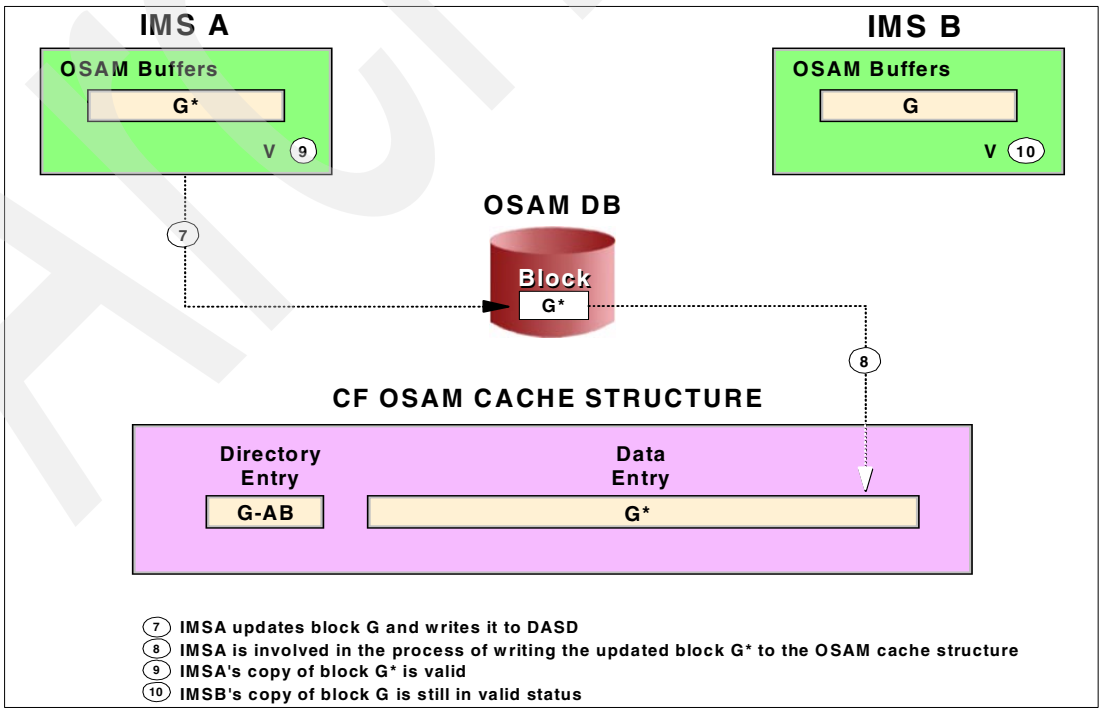


Figure 4-4 OSAM cache read and update data: diagram 3

Archived

IMSA has updated block G. The first activity is to commit the buffer to DASD before the updated data entry in the Coupling Facility is updated. At this moment IMSB is unaware that IMSA has updated the data in the block it has interest in.

Figure 4-5 shows IMSB accessing block G and the CF OSAM structure directory entry being updated.

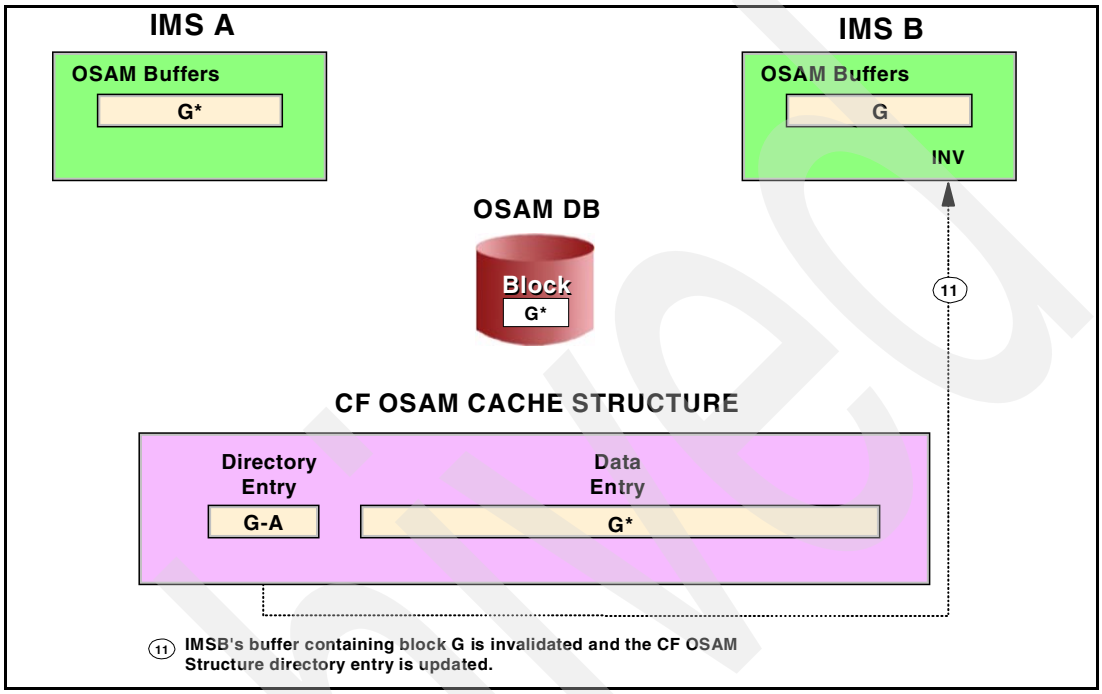


Figure 4-5 OSAM cache read and update data: diagram 4

Since the buffered image of block G in IMSB is invalid, it is marked as such and its valid interest in block G is removed for the moment from the CF OSAM cache directory entry for that buffer. If IMSB doesn't try to access G, no additional cost is incurred. If IMS3 does need access to G then it must be re-read from the cache.

Figure 4-6 illustrates that now IMSB has the correct level of the block loaded into its private buffer pool and placed back as a valid interested subsystem in block G.

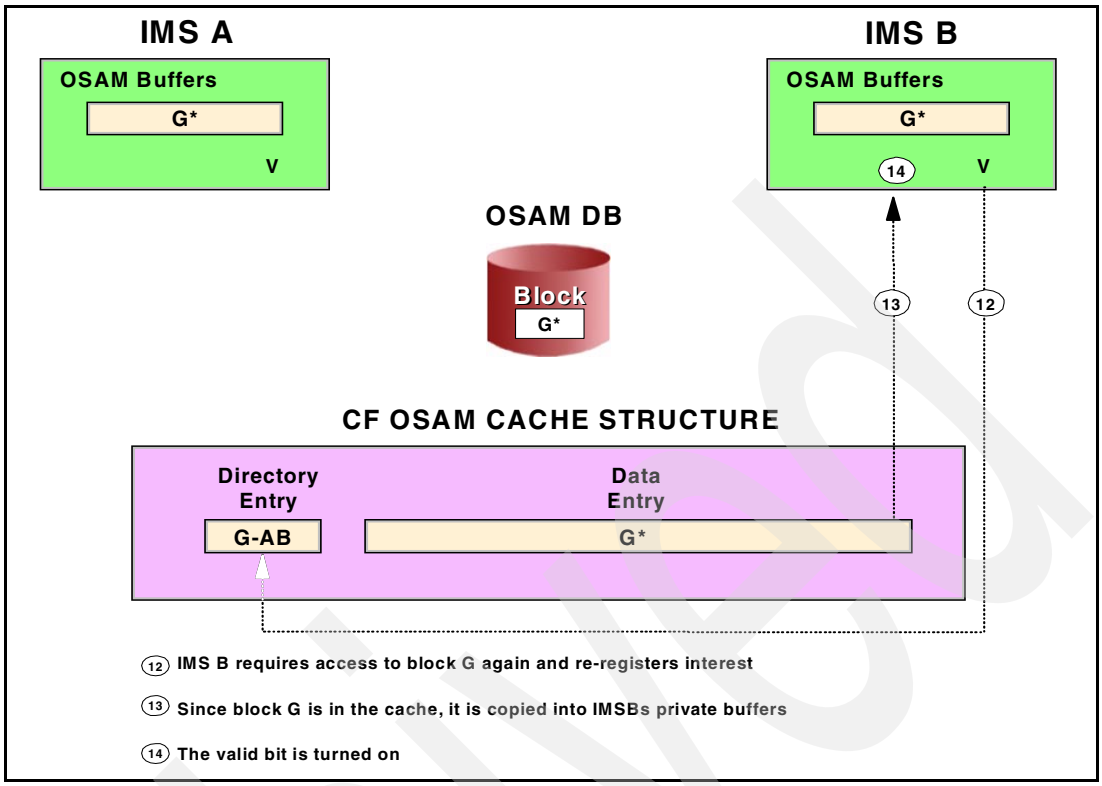


Figure 4-6 OSAM cache read and update data: diagram 5

The most important point to emphasize from this example is that IMSB did not have to access DASD to obtain the updated block G*. Instead, it was able to obtain the latest version of this block from the Coupling Facility structure.

4.1.6 General guidelines to improve performance with OSAM DB CF caching

There are four principal guidelines associated with OSAM DB CF caching:

- If you have no need to perform data caching at the moment, allow DIRRATIO and ELEMRATIO values to default to 999:1 in the DFSVSMxx member. This defines 999 directory entries of 200 bytes each for one data element. Therefore, most of the space will be allocated to directory entries rather than data entries. Another option is to override the default ratio with DIRRATIO set to 1 and ELEMRATIO set to 0 causing the structure to be allocated with no data elements.
- When IMS subsystems are cloned, OSAM caching will provide a good opportunity for performance improvements.
- Since the first IMS system to connect to the OSAM structure causes the build to occur, if a batch job is the first to share in the OSAM database, then it will be the batch rather than the online subsystem that causes space allocations to occur.
- Also there are improved statistics for OSAM pools that provide OSAM caching activity counts and sequential buffering counts using the OSAM extended STAT call. It should be exercised to assist in monitoring use and effectiveness of OSAM DB CF caching.

4.2 Fast Path DEDB VSO sharing

The virtual storage option (VSO) was introduced with IMS Version 5 to replace MSDBs. VSO provides similar benefits to main storage database users, because there is no call related I/O. Before shared VSO, basic VSO used OS/390 data spaces as its local cache storage area for DEDB AREA data. There are also a few additional enhancements included with the basic VSO feature:

- Insert and delete calls are supported.
- VSO Fast Path AREAs are supported by DBRC.
- Standard utilities are now available to the user.
- VSO allows designs to use the full DEDB hierarchical model.
- Field level (FLD) calls are supported, so that applications that access MSDBs using FLD calls do not have to be modified.

4.2.1 Components of the VSO data sharing solution

SVSO support was first introduced with IMS Version 6. There are several participants in the shared VSO support design.

DBRC and RECON data set updates

There are some mandatory and other optional features to be introduced into the database recovery control environment for shared VSO support:

- The INIT.DB command must be used to set the SHARELVL to a value of 2 or 3.
- The INIT.DBDS command must be used to set VSO on for the AREA.
- Use of the PRELOAD, PREOPEN, and SHARELVL(0|1) options must be reviewed for use.
- A new CFSTR1 parameter is used in the INIT.DBDS command to name the DEDB AREA VSO CF cache structure.
- If two structures for the AREA are to be used, the CFSTR2 parameter names the second.

The cache structure

A store-in cache structure is used to store data and invalidate buffers. Data is written to the store-in cache first and data is cast out to DASD periodically. Each VSO DEDB AREA is represented in the Coupling Facility shared storage by one or two failed-persistent cache structure. That is, cache structures are deleted after the last IMS subsystem sharing partner disconnects from the Coupling Facility. There are several elements that make up this type of cache system:

- Permanent DASD storage to house the DEDB AREA.
- The cache structure, which is allocated in the Coupling Facility and holds data to be shared by the data sharing partners.
- Local cache buffers, which contain a copy of the data in the cache structure, and are allocated through the IMS database buffer pools.
- Local cache vector, which indicates the validity of the data in the local cache buffers. It is located in the HSA, which is system storage.
- A Coupling Facility connection, which is the communication path between IMS and the structure, using high speed CF links.

The storage in the VSO store-in cache structure is divided into directory entries and data entries.

Archived

Directory entries

Directory entries are each about 200 bytes long and contain control information about data items shared among users:

- A 16-byte name of the data item, which is made up of the four byte ID 'VSOS', eight bytes for the AREA name, and four bytes for the relative byte address (RBA) of the CI in question.
- Which IMS registered interest in the data item.
- If data is in a user's buffer, where that buffer is located.

These directory entries are required for buffer invalidation and there is one directory entry per control interval. The data item may be in the cache structure, in each user's buffers, or in both locations.

Data entries

The data entries are used to store data ranging in size from 512 KB to 28 KB, which is the range of the DEDB control interval size. The data entry is composed of between 1 to 7 data elements depending on the DEDB CI size. The size of the data element also depends on the CI size. For example, an 8 KB control interval uses two 4 KB data elements.

Table 4-1 provides the data element breakdown by CI size.

Table 4-1 Data element breakdown by CI size

CI size (KB)	Data element size (KB)
512	512
1024	1024
2048	2048
4096	4096
8192 - 28672	4096

CF access to a single element data entry is synchronous, whereas multiple element data entry access is asynchronous. Therefore, there is a performance benefit in keeping the CI size to 4 KB or less.

Data is written to the CF and periodically the data is cast out to DASD. Therefore, data items are read from cache, if possible instead of from DASD.

The shared VSO DEDB caching structure, illustrated in Figure 4-7 shows a one-to-one relationship between directory entries and data entries.

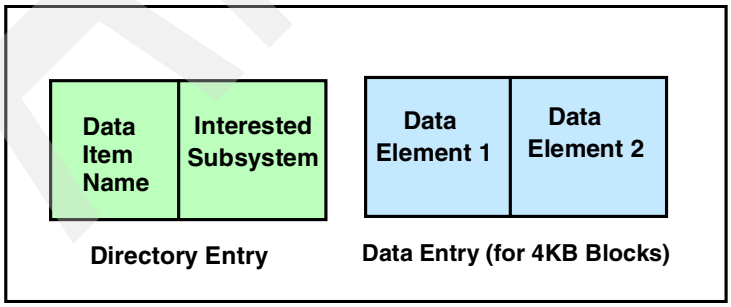


Figure 4-7 CF structure directory entry and data elements

Archived

The local cache buffer pool

This component is made up of two components: the local cache directory and the local cache buffers.

The local cache directory associates the buffer with data in it to the entry in the local cache vector which is built in the hardware system area (HSA).

The local cache buffer's contents may or may not be valid at any point in time.

The local cache vector is located in the hardware system area. Each local cache vector entry indicates the validity of the local cache buffer. Each buffer is identified as valid or invalid.

The CF connections

Each subsystem accessing a SVSO structure must connect to the structure by the use of the XES IXLCONN macro. The first user to connect to the CF allocates the structure. The size of the structure is determined by the CFRM policy, and the connection fails if the structure is not defined in the selected policy. CF connections are defined as persistent, which means that if a connection fails, it remains allocated as a failed persistent connection. Figure 4-8 shows the major elements of a SVSO DEDB environment.

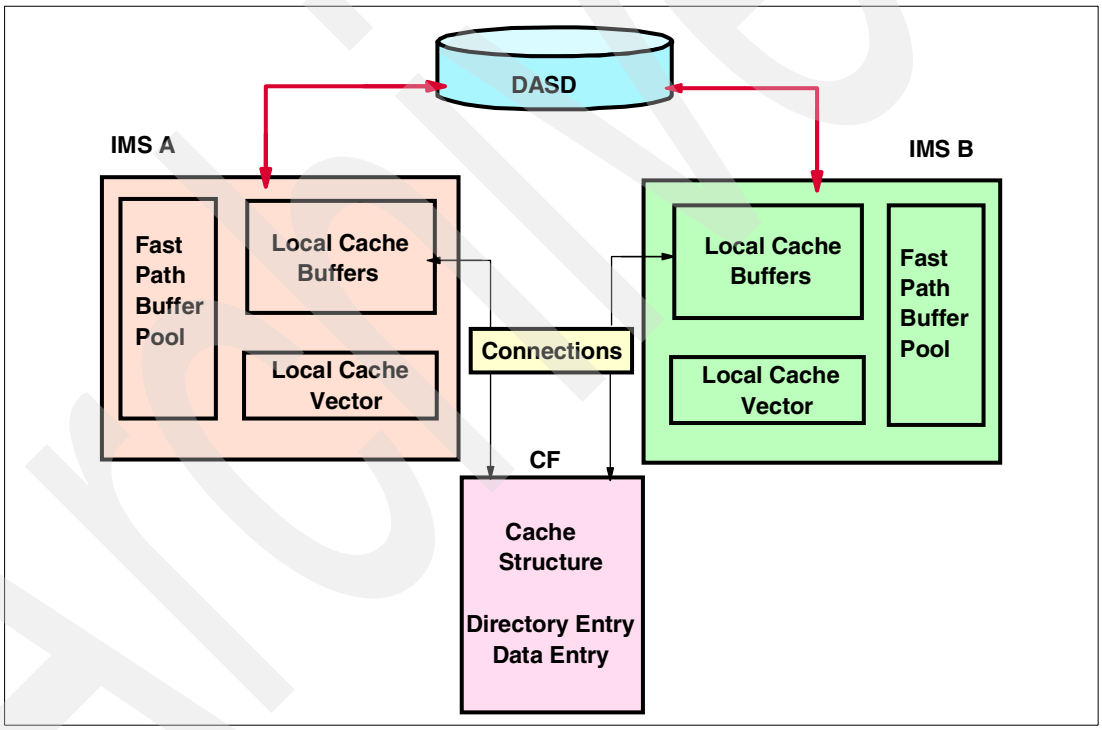


Figure 4-8 Elements of a store-in cache system

4.2.2 VSO store-in cache structure and host elements interface

Figure 4-9 presents the relationship between the host based cache directories, buffers and vector index entries and the CF cache structure for data sharing DEDB VSO.

Archived

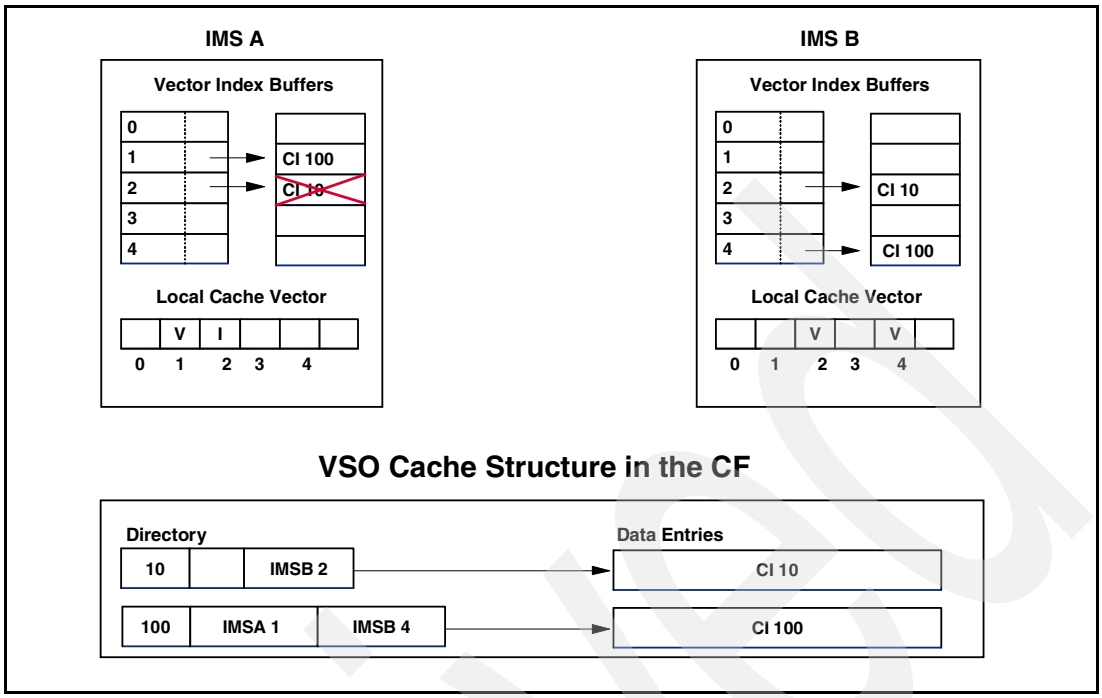


Figure 4-9 DEDB VSO AREA structure relationships

In Figure 4-9 both IMSA and IMSB have expressed interest in control interval 100 and both have valid images within the local private buffers. Figure 4-9 also indicates that IMSB has updated control interval 10, causing a buffer invalidate in IMSA which is reflected in IMSAs local cache vector table, vector index, and the CF cache structure directory entry.

4.2.3 IMS Version 8 additional Coupling Facility support for SVSO

Fast Path has been enhanced for IMS Version 8 so that shared DEDB VSO databases are also able to exploit the following Coupling Facility features:

- OS/390 system managed rebuild of a VSO structure. IMS Version 7 APAR PQ50661 provides this support as well. The command to initiate a system managed rebuild to copy a VSO structure from one CF to another is:

  ```
  SETXCF START,REBUILD,STRNAME=strname
  ```

- OS/390 alter and automatic alter of a VSO structure sizes. IMS Version 7 APAR PQ50661 provides this support as well. The automatic alter feature does not take effect unless users explicitly specify ALLOWAUTOALT(YES) for a SVSO structure in a CFRM policy.
- OS/390 system managed structure duplexing. Duplexed structures are duplicate structures of the same area. If you are using OS/390, you must define primary and secondary VSO structures to DBRC and in the CFRM policy. However, if you are using z/OS, you can take advantage of system-managed duplexing. At the time of publication, although IMS has supplied the support to enable this feature, the OS/390 maintenance necessary to activate is not available.

These enhancements provide the following benefits:

- You can keep the structure online during planned reconfiguration (rebuild).
- CF storage is reclaimed together with dynamic expansion and contraction of structures, based on actual CF storage usage (auto alter).

- System-managed duplexing allows you to have dual VSO structure support without defining both primary and secondary structures to DBRC and in the CFRM policy. Duplexing operation is transparent to you except for requesting duplex mode in a CFRM policy and allocating additional resources for a secondary structure instance.

4.3 Sharing Fast Path DEDB sequential dependent segments

DEDBs have a special type of dependent child segment called a sequential dependent, or SDEP. The SDEP segment type is defined within a DEDB by coding the TYPE=SEQ parameter on the SEGM statement in the DBD macro. SDEP segments are not stored near the root segment or other dependent segment types. Instead, they are stored in SDEP CIs at the end of each AREA data set (ADS). When an application issues an ISRT call for a SDEP, the segment is saved in a dependent region's normal buffer allocation (NBA) buffer until it is committed, at which time it is moved into the current sequential dependent buffer (CSDB). As each application program reaches commit, it acquires a lock on this CSDB and moves its SDEPs to the next available position. When the CSDB is full, it is written to the ADS and another buffer is allocated and filled. The net affect of this is that SDEPs are stored in the ADS in the time sequence in which they were inserted, without regard to which root segment they were stored under — therefore the name sequential dependent segment.

The SDEP Scan utility (DBFUMSC0) is used to retrieve SDEPs in time sequence and the SDEP Delete utility (DBFUMDL0) is used to delete SDEPs after they have been scanned. Because the CSDB buffers are not written until they are full, there is no way to let two or more IMSs put committed SDEPs in the same buffer. Putting them in different buffers would destroy the sequential order of the segments. This is why SDEP data sharing was not initially supported until IMS Version 6.

4.3.1 The shared SDEP enhancement

IMS Version 6 solved the SDEP sharing problem through a combination of IMS internal processing changes as well as SDEP CI and segment format changes. Although the SDEP segments are no longer physically in sequence, they now contain a commit timestamp (the 8-byte STCK value at commit time) which optionally allows them to be sorted as part of the scan process and presented to the end-user in chronological sequence — the same as when they were not shared. These changes are completely transparent to the application program inserting them, although there are some changes to the two SDEP utilities.

This feature has the following characteristics:

- Processing changes

  SDEP CIs are no longer allocated one at a time. Instead, each sharing IMS is preallocated a set of SDEP CIs which it proceeds to fill, and then asks for more. The number allocated for each request is dependent on how fast that IMS is filling them up. The CI preallocate process ensures that changes to the second CI is logged and that control CI is hardened before a IMS begins to use any preallocated CIs.

  Note that this allocation method means that each IMS inserts SDEPs in sequence within its own set of allocated CIs, but that when combined with the CIs allocated to other IMSs, the overall effect is that they are not physically in chronological sequence. This problem is addressed by the SDEP segment format change that is described in the next paragraph.

- SDEP segment format change

  An eight-byte timestamp is added to the end of each SDEP segment. This timestamp denotes the time that segment was committed. Since each IMS in the data sharing group is using a time synchronized by the external time reference (sysplex timer), these

timestamps are consistent across all IMSs and can be used by the SDEP Scan utility to sort the SDEPs after scanning them and before writing them to the output file. The net effect is that they appear to the end user to be in chronological sequence, just as they were when they were not shared.

- SDEP CI format changes

  There have also been some changes to the format of the SDEP CI itself. There are some new flags and the IMSID of the IMS system which filled that CI. In all, there are an additional eight bytes of overhead added to the SDEP CI.

- Use of the IRLM

  Prior to IMS Version 6, SDEP segments and CIs were never locked. The only locking came in the form of an internal CSDB latch which serialized the inserting of committed SDEPs into the CSDB by different dependent regions. When SDEPs are shared, you may see an IRLM lock on the CI currently in the CSDB. The purpose of this lock is not to protect the buffer (that is still done by the CSDB latch) but rather it is used by IMS to determine whether any particular CI is among those in its set of pre-allocated CIs. This in turn is used when a program running in one IMS needs access to a segment in a CI which is still in another IMS's buffer pool.

- SDEP scan and delete utilities

  Prior to IMS Version 6, the scan and delete utilities worked solely on RBA boundaries. There were no timestamps — everything was already in sequence.

  With IMS Version 6 and the possibility of data sharing, SDEP segments may not be physically in chronological sequence (at least, if they are shared). So establishing start and stop times for these utilities can no longer be done based solely on RBA. Instead these utilities now work on a timestamp basis. For example, the SDEP Scan utility may be directed (through control cards) to scan all segments inserted between 8:00 AM and 10:00 AM. In a data sharing environment, it is possible that some segments inserted at 10:01 AM may physically be interspersed among those between the two specified times. The SDEP Scan utility will ignore these SDEPs and return only those between the desired times. There are a number of control card changes made in IMS Version 6 (even when the SDEPs are not being shared) which you need to understand. Please refer to the manual *IMS Version 8: Utilities Reference: Database and Transaction Manager,* SC27-1308*,* for details of the changes. Note that many of these changes apply even if the SDEPs are not being shared.

- Transparent application program support

  Although the SDEP segment itself has been increased by eight bytes, these eight bytes are added to the segment by Fast Path when it is inserted, and removed by Fast Path when the segment is retrieved either by an application program or by the scan utility. The LL (length) field in the segment as seen by the application is the length excluding the eight bytes. In other words, to the application program, the segment looks just the same. No application changes are required.

- No changes to DEDB AREA root and direct dependent segment sharing
- No effect on XRF backup and takeover

  Backup tracking and takeover restart open processing are not affected in their function by this enhancement.

4.4 Fast Database Recovery (FDBR)

Fast Database Recovery (FDBR) is an optional function of IMS introduced in IMS Version 6. It is designed to quickly release locks held by an IMS subsystem when that subsystem fails.

FDBR monitors an IMS subsystem using the IMS log and XCF status monitoring services. When it determines that the IMS has failed, FDBR dynamically backs out uncommitted updates for full function databases and does the redo processing for DEDBs. These are functions that would be performed by emergency restart.

When FDBR completes the back outs and redos, it releases the locks held by the failed subsystem. This allows any sharing subsystems to acquire these locks. From the time of the IMS failure until the locks are released, other subsystems requesting the locks will receive lock rejects which usually result in U3303 abends of application programs.

FDBR is not a replacement for emergency restarts of IMS. Emergency restarts are still required to do the following:

- Recover the message queues
- Recover MSDBs
- Resolve in-doubt threads with CICS and DB2
- Change the DBRC subsystem record from “failed” to “normal”
- Release DBRC authorizations

The benefit of FDBR is its quick release of in-flight locks. This lessens the likelihood of lock rejects in other IMSs.

FDBR, creates a separate IMS control region running on the same or another OS/390 image. This control region monitors an IMS subsystem, detects failures, and automatically recovers any database resources that are locked by the failed IMS, making them available for other IMS subsystems.

4.4.1 Environments where FDBR is supported

Full function HDAM, HIDAM, HISAM, index databases, including HALDB databases along with Fast Path DEDBs (including shared VSO and SDEPs) are supported. XRF is not supported by FDBR and if IMS includes the HSBID parameter, FDBR is disabled. IMS remote site recovery is supported when FDBR is active.

Figure 4-10 presents the FDBR region in a separate OS/390 system and the resources it must interface with.

Archived

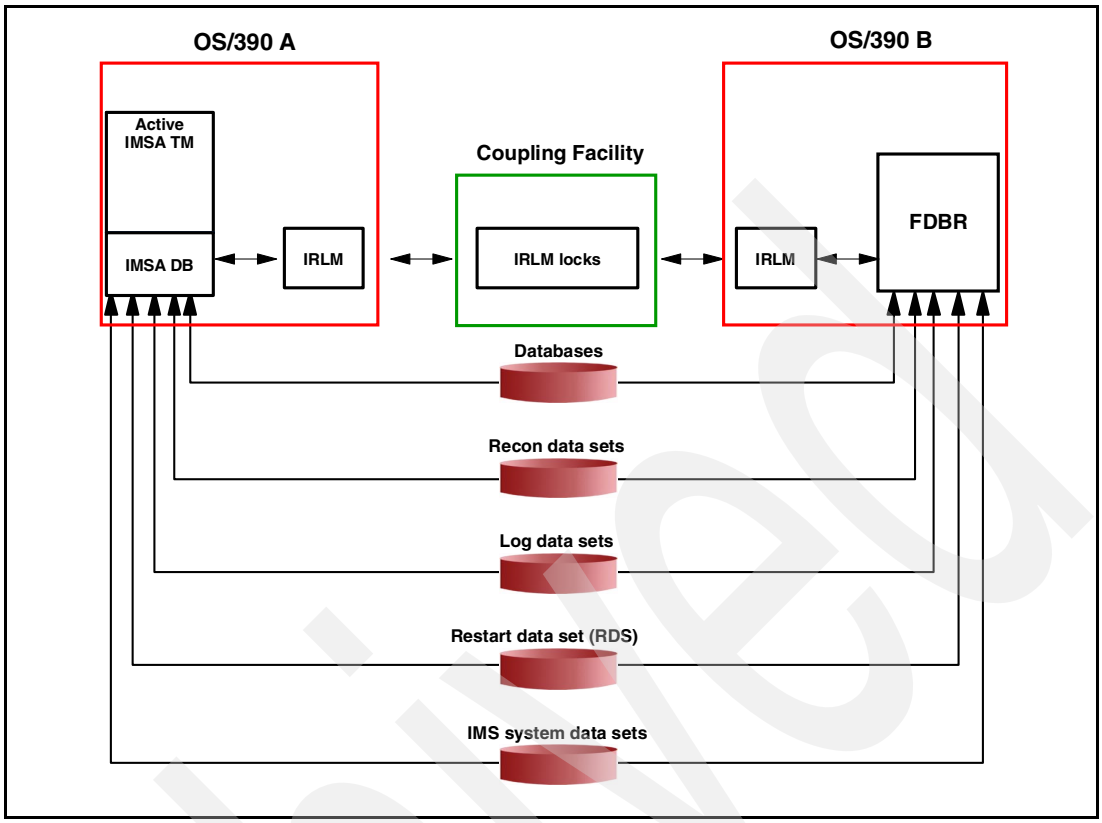


Figure 4-10 Fast Database Recovery elements

Each FDBR region tracks a single IMS subsystem in the XCF group and must have access to the following components to perform the surveillance functions and recovery of affected databases during failures:

- RECON data sets (can be dynamically allocated)
- IMS log data sets (OLDS and WADS, can be dynamically allocated)
- IMS system data sets (SDFSRESL, MODBLKS, MODSTAT, ACBLIB)
- RDS for checkpoint information
- The databases (can be dynamically allocated)

In subsequent chapters on the implementation, operations and recovery-restart of data sharing environments, we will discuss FDBR again.

Archive

Archived

5

IMS shared queues

In this chapter, we introduce shared queues, describe their design, components, and message flow, and provide some information to consider when planning and implementing a shared queues environment. We cover:

- An overview of IMS shared queues environment
- Components of a shared queues and how they work together
- Shared queues queue types
- Logging and security
- Overflow and full conditions

We assume that you have a general knowledge of IMS message processing.

Archived

© Copyright IBM Corp. 2003. All rights reserved.

5.1 Shared queues overview

Many IMS users are looking forward to managing their increasing application workloads efficiently now and in the future. In addition, they want to develop strategies to improve their current service levels and move toward continuous operations while maintaining or improving acceptable response times.

IMS users recognize the requirement to distribute and balance the total workload efficiently across multiple systems. IMS shared queues, a function introduced in IMS/ESA Version 6, assists users in meeting these requirements by:

- Providing dynamic load balancing among multiple IMS systems in a Parallel Sysplex environment
- Optimizing overall throughput across the sysplex, so that no single IMS remains under utilized while others are overloaded
- Achieving improvements in continuous availability for applications
- Improving reliability and providing better failure isolation, so that any remaining IMS can continue processing the shared workload
- Adding on new IMS systems as workload increases

Without shared queues, each IMS system has its own queues for both input and output messages, and its own expedited message handler for Fast Path messages. Messages are processed in the local IMS system, unless that IMS sends the message to another IMS system. With shared queues, all IMS systems in a Parallel Sysplex can share a common set of message queues stored in the Coupling Facility. Full function and Fast Path input and output messages can be processed by any IMS system that has access to the shared queues and is capable of processing the message.

Any IMS system running in a Parallel Sysplex can participate in processing application workload as defined in its IMS system definition according to available CPU capacity and resources. With a shared queues implementation, new IMS subsystems can easily be added to the sysplex as the workload increases or as extra loads must be processed (scalability). The content of shared queues is not affected by hardware or software failures. Remaining IMSs continue to schedule their applications and process messages stored in the Coupling Facility.

5.1.1 Description of facility and use of Parallel Sysplex

There are really two sets of shared queues - those that support IMS full function messages, and another set that supports IMS Fast Path expedited message handler (EMH) messages.

The queues themselves are really lists of messages that are kept in list structures in the Coupling Facility. There may be as many as four shared queues list structures for one IMSplex that is sharing the message queues:

- A primary structure for the full function messages (required)
- An overflow structure for the full function messages (optional)
- A primary structure for Fast Path messages (required only if the FPCTRL macro (enables Fast Path) is included in the IMS system definition)
- An overflow structure for Fast Path messages (optional)

As seen in Figure 5-1, a list structure consists of a number of list headers which act as anchor points for messages. There are list headers for transactions, LTERMs, MSNAMEs and other

resources. Individuals messages are called list entries in sysplex terminology. Note that long messages may require multiple list entries.

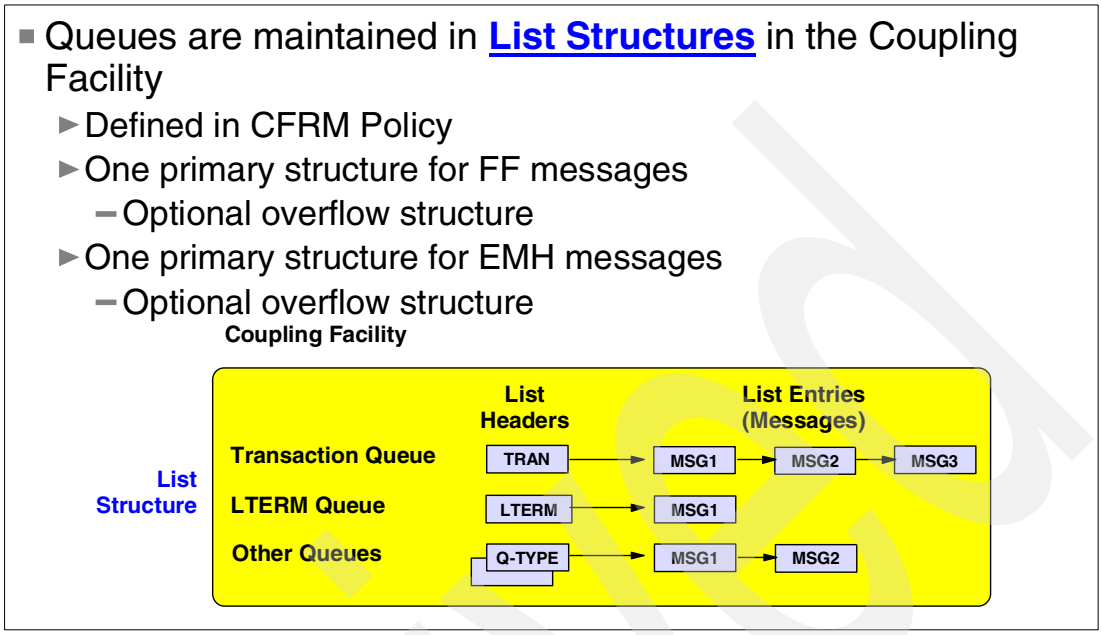


Figure 5-1 Shared queues structure

When IMS is initialized, it connects to these structures through an address space called the Common Queue Server (CQS).

Once connected, it registers interest in specific resources (for example, transaction code, LTERM name, MSNAME) which it is able to process (for example, process a transaction, deliver a message, send to a remote IMS).

- For the full function message input queue, IMS will register interest if the transaction is defined to that IMS, and it is not stopped (if it were stopped, it couldn't process it, so it doesn't register interest).
- For the EMH input queue, IMS will register interest in a PSB when the first IFP region for the PSB is started in the local IMS system.
- For LTERMs, the terminal must be in session with IMS before IMS will register interest. IMS cannot send a message to a terminal it is not in session with.

Note: Multiple IMSs will (probably) register interest in the same transactions, but only one IMS will register interest in an LTERM (the terminal is in session with only one IMS at a time).

5.2 Shared queue components

There are lots of new components to an IMS shared queues environment compared to an environment without shared queues. Figure 5-2 shows the main components in an IMS shared queues environment.

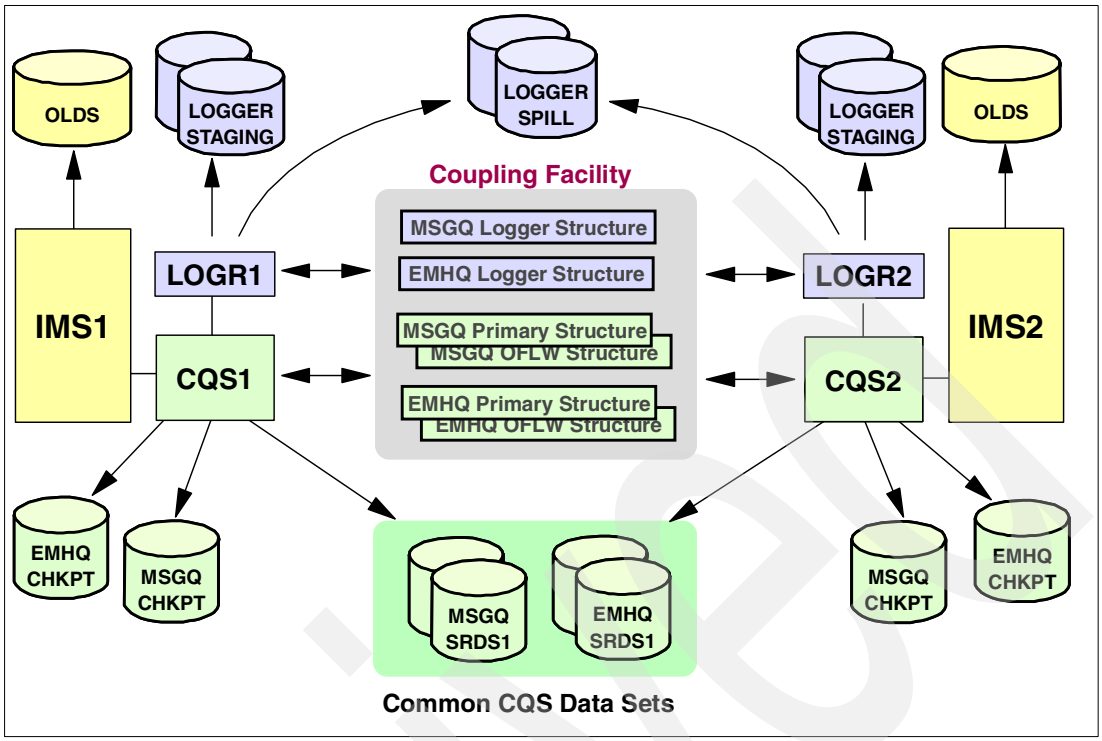


Figure 5-2 Shared queues components

Here is a short description of the main components in an IMS shared queues environment as shown in Figure 5-2:

- Coupling Facility

  IMS shared queues are defined in the Coupling Facility.

- Cross-system Coupling Facility (XCF)

  Cross-system Coupling Facility (XCF) is the OS/390 Software to communicate and monitoring services between the Coupling Facility and Common Queue Server (CQS) address space.

- XES and XES services

  All the accesses to CFs are handled by the OS/390 XES component. XES application programming interfaces (APIs), also called XES services, are the only means by which programs and subsystems can access structures in the Coupling Facility.

- Coupling Facility structures for IMS messages

  The following list type of structures can be defined in the CFRM policy for IMS shared queues:

  – Message queue structures (primary and an optional overflow)
  – EMH queue structures (primary and an optional overflow)
  – System Logger structures (MSGQ and EMHQ log streams)

- Common Queue Server (CQS) address space

  The Common Queue Server CQS is a generalized queuing facility that manages data objects within shared queues on behalf of its clients. One CQS address space is required per IMS in Version 6. It is possible to have multiple IMSs per CQS (within one OS/390 image) in Version 7. CQS is the interface between IMS and shared queue structures. In IMS Version 8, CQS is enhanced to support the Resource Manager (RM). RM uses the

CQS to access a new Coupling Facility list structure called a resource structure. The resource structure is optional and when used, it contains global resource information for uniquely named resources.

- SRDS data sets and CQS CHKPT data sets

  Structure recovery data sets (SRDS) are shared data sets that contain structure recovery information for shared queues on a structure pair the image copy of the queues. They are used with OS/390 log stream to recover structures. There are two SRDS data sets per structure pair. CHKPT data sets are for the CQS checkpoint. They are used for CQS restart.

- System Logger and LOGR policy

  System Logger is part of the OS/390 or z/OS and it is used by CQS to log all activity to and from shared queue structures. CQS writes log records to a LOGSTREAM defined in the LOGR policy. All CQSs in the shared queues group use the same log streams — log records from multiple CQSs are merged by the System Logger. LOGR policy defines the log streams used by CQS and identifies the structures used by the log streams. It also specifies other logger options, such as offload and duplexing options. Logger structures are defined in the CFRM policy. One logstream and one structure each for MSGQ and EMHQ is needed. Logger data sets consist of offload (spill) data sets used to archive old log data and staging data sets used to duplex data in structure (optional, but not desirable for performance reasons). Do not mix up the term spill data set with the spill records used in Change Accumulation.

5.3 Common Queue Server (CQS)

The Common Queue Server CQS is a generalized queuing facility that manages data objects within a shared queue on behalf of its clients.

CQS runs on either a z/OS or an OS/390 operating system. The CQS client must also run under the same z/OS or OS/390 operating system. CQS uses the Coupling Facility as a repository for data objects. Clients communicate with CQS to direct their CQS to manipulate client data residing on shared Coupling Facility structures. The CQS runs in a separate address space that can be started by the client system.

IMS DB/DC and DC Control (DCCTL) subsystems use the CQS to access the shared queues. The CQS address space is started during IMS initialization through parameters specified in the **CQSIPxxx** PROCLIB member. Each IMS in a shared queues environment accesses the shared queues through a related CQS address space.

CQS performs the following services on behalf of its clients:

- Notifies registered clients when work exists for them on the shared queues.
- Provides clients with an architected interface for accessing the shared queues and the CQS. Potentially, subsystems other than IMS DB/DC and DCCTL could use this interface.
- Writes CQS checkpoint information to a CQS checkpoint data set for restart of CQS.
- Writes structure checkpoint information to structure recovery data sets (SRDSs) for recovery of shared queue list structures.
- Provides structure recovery for the shared queue list structures.
- Performs overflow processing for the shared queue list structures.
- Drives numerous CQS and user exits. CQS exit routines inform the client when events, such as the following occur:

- CQS abnormal termination
- CQS restart completion
- Structure copies
- Structure recoveries
- Structure overflow
- Work available on a shared queue

5.3.1 Base Primitive Environment

The Base Primitive Environment (BPE) is a software layer on which new components of IMS are designed to be built on. It is the first step in the rearchitecting of IMS. BPE provides common services, such as subdispatching, storage management, tracing, serialization, and message formatting. BPE services are accessed through well-defined architected interfaces. CQS is one of the first components built on the BPE software layer.

For the most part, BPE is not visible. IMS components are built on top of BPE, but BPE itself has very few externals. The externals are:

- BPE modules in RESLIB
- Messages issued to the system console
- CQS JCL
- PROCLIB members
- BPE dump formatter

For information about setting up the BPE PROCLIB members, see *IMS in the Parallel Sysplex, Volume III: IMSplex Implementation and Operations,* SG24-6929.

5.3.2 Components of a list structure

All the structures used in the Parallel Sysplex must be defined in the Coupling Facility Resource Management (CFRM) policy, before they can be allocated. When the structures related to shared queues are allocated by CQS, they will be allocated as list structures. The type of the structure (cache, lock, list) is determined by the connector, not by the policy.

There are two pairs of structures in a shared queues environment — one pair for full function messages and another pair for Fast Path EMH messages. Each pair consists of a primary structure and an optional overflow structure. The overflow structure is used to offload messages from the primary structure when it is in danger of becoming too full.

The EMH structure pair is needed only if IMS system generation includes the FPCTRL macro, enabling Fast Path. If the IMS system definition includes the FPCTRL macro, the EMHQ structure is always required. There are cases where the IMS system definition has the FPCTRL macro and EMH is not being used to process transactions, such as with Fast Path DEDB databases being used. The EMHQ structure is still required because Fast Path transactions could be implemented through an online change if the system has been defined as Fast Path capable. If the system is defined as Fast Path capable, and you do not process EMH transactions, your EMHQ structure can be defined as the minimum size of 256K.

Note: When migrating CF levels, lock, list, and cache structure sizes might need to be increased to support new function. For example, when you upgrade from CFCC level 9 to level 10, the required size of the structure might increase by up to 768 K. Flexibility is a good convention to specify for both the initial and maximum sizes in CFRM.

Shared queues structures are persistent. They remain allocated even if all CQSs have disconnected, or all IMSs and their associated CQSs cold start.

There are four components of a list structure that should be understood to some extent:

- Lock table

  This is a construct used solely to serialize some resource within the list structure. It is optional from an OS/390 perspective, but CQS will build a list structure with a lock table. This type of list structure is called a *serialized list structure*. It is used by CQS to serialize structure operations, such as structure checkpoints and structure rebuild processing.

- List headers

  These are anchor points for the many lists that may be in the structure. Each message is a member of a list which starts at the list header.

- List entries

  These are the messages, and consist of list entry controls (information about the list entry), an adjunct area (user data used by CQS), and the data entry (the message itself — transactions, responses, other messages, includes IMS message prefix and data).

- Event monitor controls (EMCs)

  These are used by the Coupling Facility Control Code (CFCC) to monitor events, such as the arrival of a message, so that CQS and IMS can be informed that there is work on the queue.

Figure 5-3 shows a full function message queue list structure with two messages. Message 1 is stored as one data object, comprising a list entry control and one data element. Message 2 is stored as two data objects (list entry 2 and list entry 3), each comprising one list entry control and two data elements.

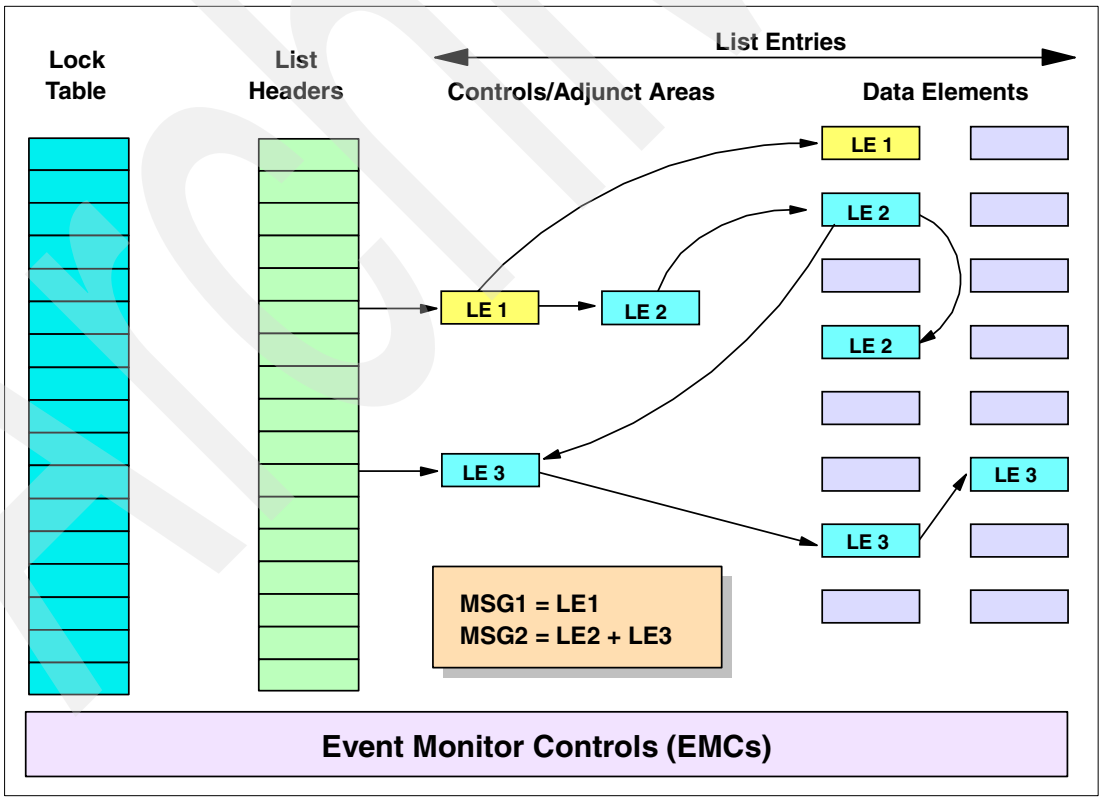


Figure 5-3 Components of a list structure

When a list structure is built, 256 list headers are created. The connector may decide how many list headers to use, and what the meaning of each is. CQS uses 192 of the available 256.

5.3.3 List headers

List headers are anchor points for CQS or IMS queues. A list header points to the first list entry in the queue. When a structure is allocated, 192 list headers are created; 71 for CQS private queues, and 121 for the IMS queues. There are 11 list headers per queue type. Figure 5-4 shows the details of the different types of queues and their identities, for example, the queue type for transaction ready queue is 01. For a MSGQ structure, 99 of the IMS list headers are used, and the remaining 22 are reserved for future use. For an EMHQ structure, 22 of the IMS list headers are used and the remaining 99 are reserved for future use.

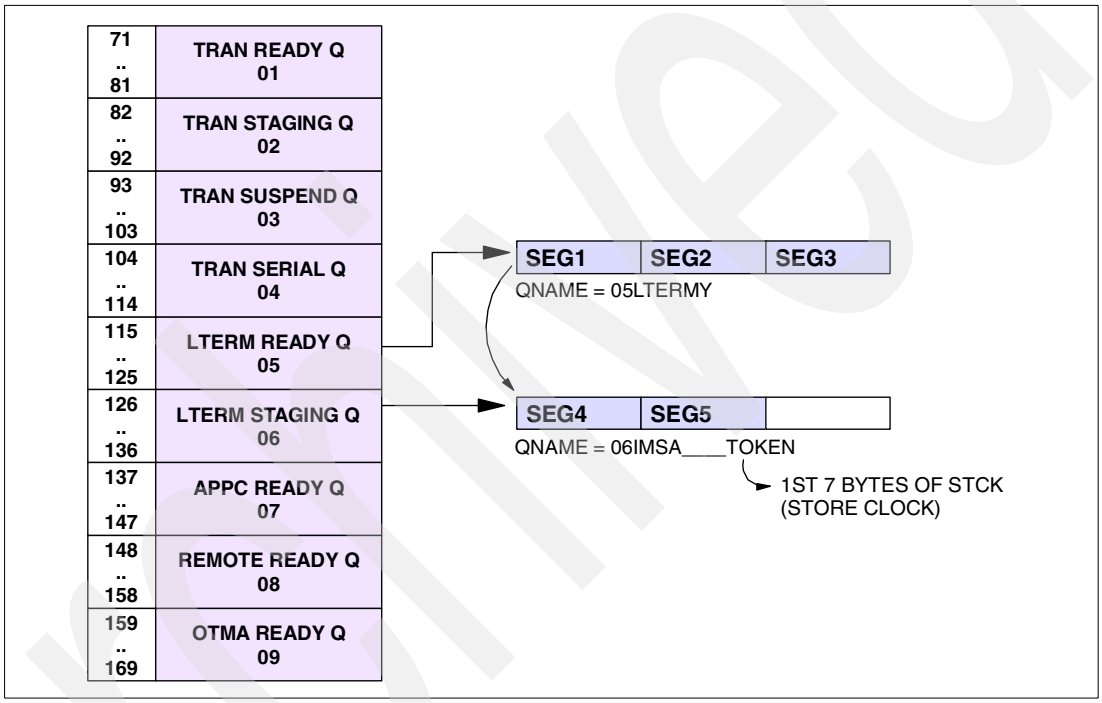


Figure 5-4 Full function message queue types

5.3.4 List entries

List entries are chained off list headers and are used to store messages. Each list entry is divided into two sections, a control area and one or more data elements. The control area contains the queue name, which is the object in which IMS registers interest if it is capable of processing that type of message. The control area is also the anchor point for one or more 512-byte data elements in which the message is stored. Each message is divided into 512-byte sections, with each section stored in a data element. The data elements are chained together to retain the layout of the message.

Components of a list structure in Figure 5-3 on page 99 demonstrates how the chaining occurs. In this figure, assume the size of the logical records are 1K (requiring two data elements per list entry). The first message has a single list entry with one 512-byte data element containing the message. The second message is 2K long and four 512-byte data elements are required to store the message. The logical records are 1K in length; to store the 2K message requires two list entries. When a message requires more than one logical record, the second and subsequent logical records are chained off the staging queue. The

first logical record is chained off the ready queue and points to the list entry in the staging queue. Figure 5-4 on page 100 shows how the queue manager divides the message across multiple data elements.

The list entries that hold the messages are called data entries. The data entry consists of one or more data elements (512 bytes each) and it is the contents of an IMS long or short message logical record in the queue pool, including the IMS message prefixes. There are an additional 16 bytes on the front of the message called the CQS prefix which contains a flag to indicate whether the message is recoverable, a timestamp, and the CQS version.

Note that it is this “recoverable” flag which indicates whether a message is to be recovered if the structure needs recovery. A data object is the IMS message (including IMS prefixes), that is the contents of one IMS LGMSG or SHMSG logical record. The maximum length of the logical record is 30,632 bytes.

IMS queue pool buffers may contain one or more logical records, depending on the size of the buffer and the sizes of SHMSG and LGMSG. Note also that each long or short message logical record may contain several IMS message segments for a single message as seen in Figure 5-5. The unit of transfer between IMS and CQS and the structure is the logical record, not a segment. Only the data content of the logical record is transferred, not the entire long or short message logical record (unless it were exactly full).

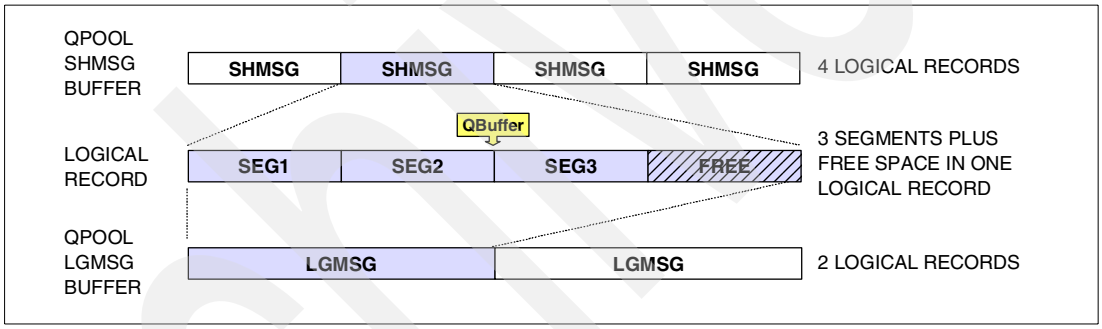


Figure 5-5 Data entry

One IMS long or short message logical record (data object) may require multiple data elements, for example:

- Long or short message logical record = 1000 bytes
- 3 IMS segments @ 300 bytes = 900 bytes
- 900-byte data object requires two 512-byte data elements

One IMS message may require multiple list entries:

- One data entry will contain the contents of a single long or short message logical record
- Long messages may require multiple long or short message logical records
- Therefore, long messages may require multiple list entries

The list entry controls contain a lot of information about the message, but for our purposes, the 16-byte QUEUE NAME is the most important. This is how IMS will identify a particular message (or type of message). Queue names do not have to be unique.

The 8-byte Version ID contains the IMSID of the IMS, which put the message on the queue, and is not further referenced in this book. It is used by CQS to identify messages which have been moved to its private queues.

The total size of the list entry controls is about 200 bytes. This information is useful in determining what the size of a structure should be. The diagram at the bottom of Figure 5-6

shows a LH with two messages queued — they are both input messages for transaction code TRANX, therefore the queue name of 01TRANX.

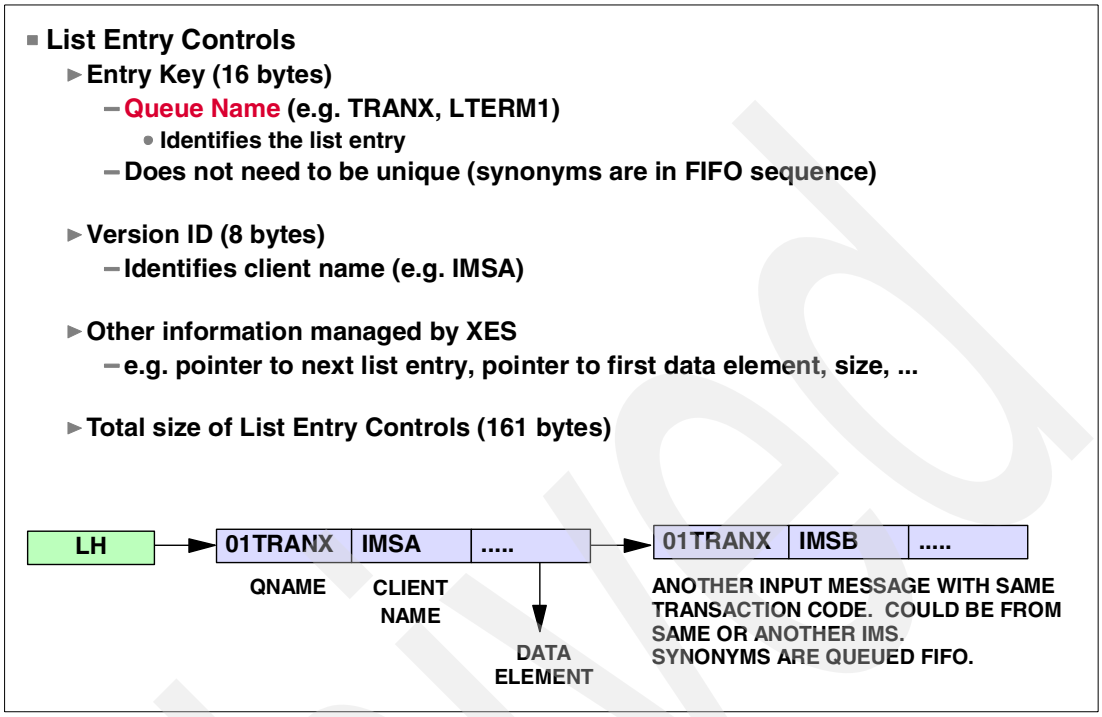


Figure 5-6 List entry controls

The adjunct area is optional for list structures (Figure 5-7), as is the lock table, but CQS does use them. They are allocated when a list entry is in use (that is, has a data entry) and are 64 bytes each (also useful when sizing structures).

The adjunct area is used by CQS to help manage the queues when messages get moved around. It contains the queue name of the original queue where the message came from (for example, 01TRANX) and something called a UOWID. The rest of the 64 bytes is currently reserved for future use.

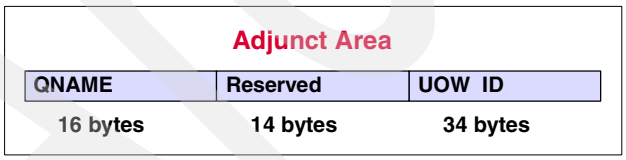


Figure 5-7 Adjunct area

There is one adjunct area per list entry. The adjunct area is 64 bytes long and it is part of 161 bytes of list entry controls. The first 16 bytes of the adjunct area contains the client queue name where the list entry resides. The queue name associates a message on a CQS private queue with a client queue and is used for resynchronizing queues after a failure.

5.3.5 Unit of work ID

The unit of work ID (UOWID) is a unique identifier of a single input or output message. It is 34 bytes long and has three components, as you can see in Figure 5-8. The UOWID can be found in CQS and IMS log records, and it can be used to associate multiple messages with a single original input transaction. The components are:

- Originating system message ID

  This is the ID of the original input message and consists of the IMSID, which put this message on the shared queues, and the STCK (store clock) value when it was put on there. This part of the UOWID never changes.

- Processing system message ID

  As a message works its way through the system, additional messages may be generated. This part of the UOWID identifies who processed the message and when. This will change with each new processing of a message, but all messages which are created as a result of that original input message will have the same originating system message ID.

- Flag

  This 2-byte field simply indicates, for logging and tracking purposes, whether the action being taken on the message is a PUT to the structure or a READ from the structure.

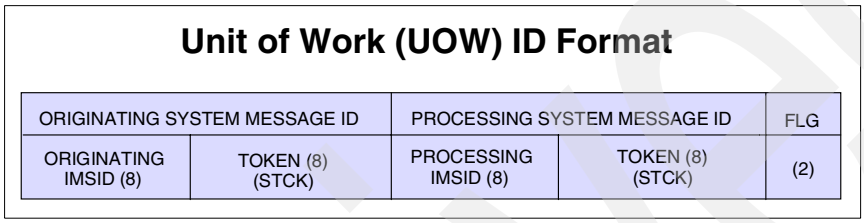


Figure 5-8 UOW

All list entries, with the same queue name, form a sublist. IMS requests the CQS to *put* a message on the end of the sublist if a new message arrives, or to *get* a message from the front of the sublist case when it wants to process a message. In Figure 5-9, seven list entries are anchored off two list headers. There are three TRANX transactions. The grouping of them together is referred to as a *sublist*. (The EMC area is described in 5.3.7, “Event monitor controls” on page 104.)

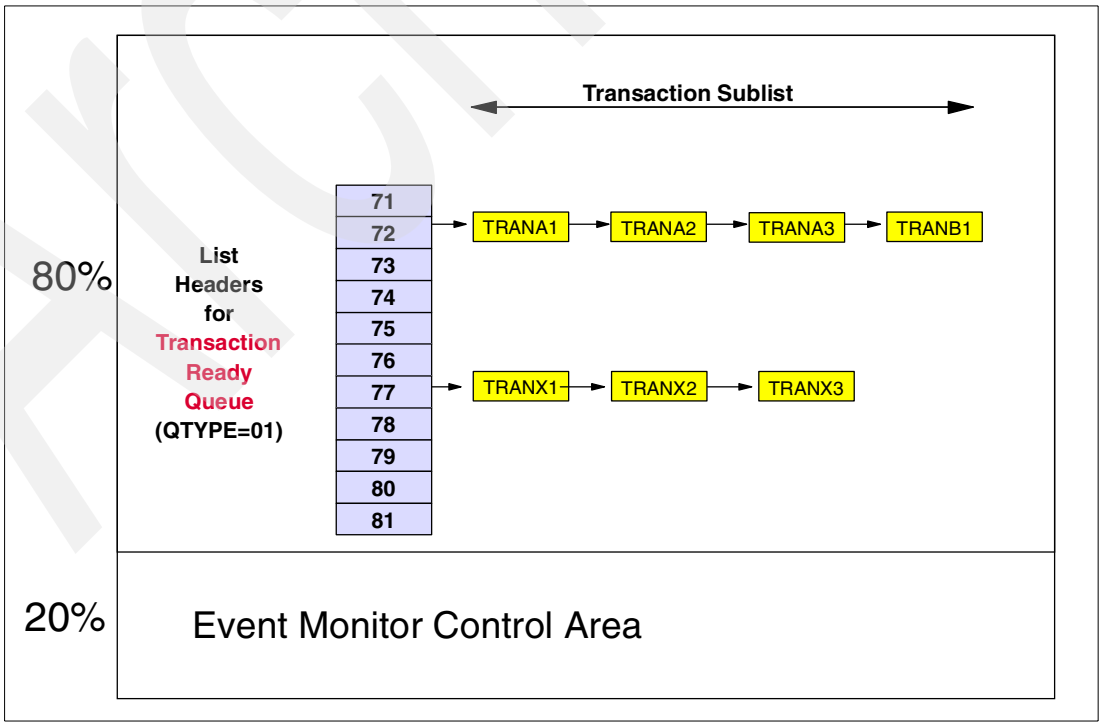


Figure 5-9 Transaction sublists in the shared queues

5.3.6 Lock table

The CQS uses the lock table to serialize access to the CQS control queue. The control queue is currently the only queue that is locked.

The control queue is a CQS private queue that has list entries for functions such as overflow processing, structure checkpoint, and rebuild processing. When a CQS wants to perform one of these functions, it must first obtain the lock for the control queue. When it obtains the lock, it proceeds to look at the list entry for the function that it needs to perform. If the list entry already contains the name of a CQS, the function is already being performed. Otherwise, this CQS adds its name, releases the lock on the control queue, and proceeds to perform the function. When a CQS has its name in the list entry for a particular function, it is referred to as the *master CQS*.

5.3.7 Event monitor controls

The event queue is located in the area of the EMCs and is used to post events of interest to the connected users. The event queue is maintained by XES in the list structure. There is one entry in the event queue for each connected user. When a monitored event occurs, an event monitor control (EMC) is queued to the connector's event queue. EMCs are used to inform the connector when an “event” occurs, for example, the arrival of a message on a registered queue.

In this case, the IMS subsystems, represented by CQSs, are the connected users, and the events they are interested in are the arrival and existence of messages on certain queues. When IMS registers interest in a queue name (for example, TRANX), TRANX is referred to as a “monitored event”. When there is work on the TRANX queue, event monitor control blocks (EMCs) representing the queue are posted to the event queue for the registered user(s), and the user is then informed there is work to be done.

The EMCs are 64-byte control blocks which represent a particular user's interest in a particular queue name. For example, when IMSA registers interest in TRANX, an EMC will be created for this relationship (IMSA-TRANX). If IMSB also registers interest in TRANX, then another EMC for IMSB-TRANX will be created.

When an EMC is first posted to the event queue, that client is informed that the queue has gone from “empty to non-empty”. The EMC will stay on the event queue for that user until he either deregisters interest (at which time the EMC is deleted) or the queue becomes empty (at which time the EMC is just removed from the event queue).

An EMC entry is a control block in the Coupling Facility list structure that is used to represent a CQS that has a registered interest in a queue. Each EMC entry occupies 64 bytes of storage. An EMC control block is created for each CQS that has an interest in a particular queue. If you have three CQSs that register an interest in queue TRANA, three EMCs are created. Care should be taken when sizing the shared queue structures to ensure that there is enough space to store the number of EMCs that are required.

CQS reserves 20% of the total structure size for EMCs. If CFCC Level 4 or greater is being used, the area can be dynamically increased in 5% increments up to a maximum of 50% if required. Use the **SETXCF ALTER** command to make this change. Available data elements are used to create additional space for EMCs. If the EMC area is increased, the amount of space available for list entries is reduced. If the EMC area runs out of space, any attempt by an IMS system to register an interest in a queue fails.

Here’s an example of how the EMC looks. When there are multiple entries with the same queue name (for example, multiple TRANXs) on the queue, they form a “sublist”. These

sublists are maintained in a FIFO manner, and are represented by a single EMC per registered client.

In the example in Figure 5-10, IMSA has registered interest in X and Y. Because there are occurrences of both X and Y in the structure, these EMCs have been posted to IMSA's event queue. IMSB has registered interest in Z. Because there are no Zs in the structure, the EMC is *not* posted to IMSB's event queue. If one arrives, it will be posted and IMSB will be informed.

IMSC has also registered interest in X. IMSC gets its own EMC, which is currently posted to IMSC's event queue. There are occurrences of Work on the queue, but because no one has registered interest, there is no EMC. If someone does register interest, an EMC will be created and immediately posted to that client's event queue.

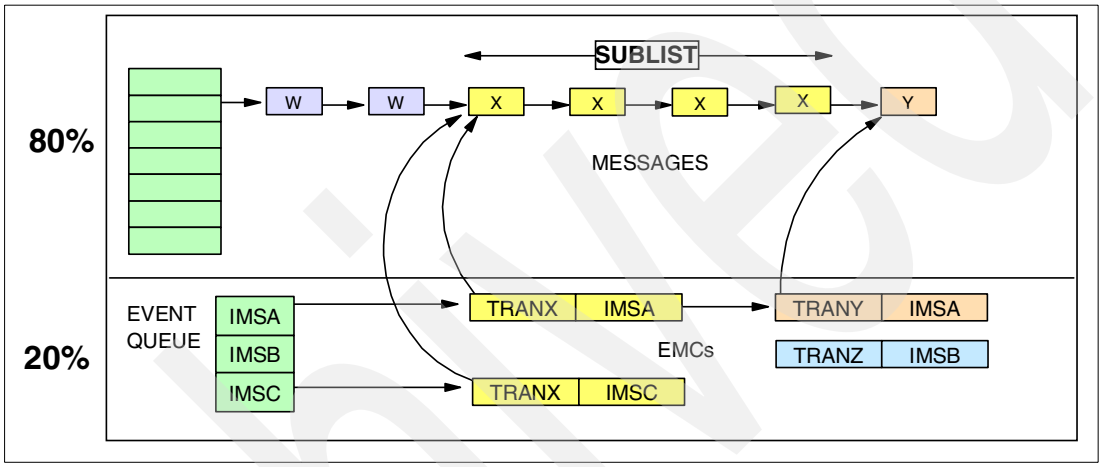


Figure 5-10 Event queue sample

5.3.8 CQS queue types

Various types of private and client queues are managed in an IMS shared queues environment. Each queue type is used for a different type of work. In a MSGQ structure, there are nine different client queue types, and in an EMHQ structure, there are two different client queue types. Eleven list headers are assigned to each client queue type, to reduce contention when accessing the client queues. Each queue type has a corresponding queue type number. The queue type number is passed by IMS to CQS with each request to put a message on a queue. The queue type number ensures that the messages are placed on the correct queue.

5.3.9 Private queue types

In the following definitions of the queue types, the queue type number is shown in parentheses after the queue name. There are six types of private queues, and they are described in Table 5-1.

Table 5-1 Private CQS queue types

Queue Type	Comment
Control queue (01)	Contains information to manage the shared queues environment. Global information about the structure, local information about each CQS, structure checkpoint information, structure rebuild information, and information about overflow processing are some of the details that are stored in the control queue.
Cold queue (02)	Contains messages that a CQS found on the lock queue that were being processed by the IMS that this CQS serves. Messages are left on the lock queue if an IMS system abends while transactions are being processed, and the IMS system is restarted cold. (To emergency restart an IMS system cold, specify either COLDCOMM or COLDSYS on the restart command.) The messages that IMS was processing at the time of the abend would have been on the lock queue. When the IMS system is restarted cold, all messages that were being processed are moved from the lock queue to the cold queue. They remain on the cold queue until they are manually dequeued. Messages are also moved to the cold queue if CQS was coldstarted (regardless of what IMS did).
Rebuild queue (03)	An intermediate queue used during recovery of an EMHQ structure with WAITRBLD=NO specified
Lock queue (08)	Contains messages that are locked. When IMS reads a message, the message is locked by requeuing it to the lock queue. The lock queue is used to prevent more than one IMS system from processing the same message. The message is later deleted when IMS completes processing it, or the message is unlocked if IMS cannot process it.
Move queue (09)	An intermediate queue used to save messages that are being moved from one queue to another
Delete queue (10)	An intermediate queue used to save messages which are being deleted

5.3.10 Full function queue types

There are nine types of full function message queues, and they are described in Table 5-2.

Table 5-2 Full function queue types

Queue Type	Comment
Transaction ready queue (01)	Contains the first logical record of messages destined for local or remote transactions
Transaction staging queue (02)	An internal queue that the IMS queue manager uses for holding those parts of input messages that exceed a single logical record
Transaction suspend queue (03)	Contains the first logical record of transactions that have been suspended
Transaction serial queue (04)	Contains the first logical record of serial type transactions (defined with SERIAL=YES on the TRANSACT macro)
LTERM ready queue (05)	Contains the first logical record of messages destined for LTERMs
LTERM staging queue (06)	An internal queue that the IMS queue manager uses for holding those parts of output messages that exceed a single logical record
APPC ready queue (07)	Contains the first queue buffer of output messages destined for APPC

Queue Type	Comment
Remote ready queue (08)	Contains the first logical record of messages destined for MSNAMES or LTERMS outside the shared queues group (MSC responses)
OTMA ready queue (09)	Contains first logical record of output messages destined for Open Transaction Manager Access (OTMA) devices

5.3.11 Fast Path queue types

Table 5-3 shows the two Fast Path (EMHQ) queue types.

Table 5-3 Fast Path (EMHQ) queue types

Queue Type	Comment
Program ready queue (01)	Messages destined for Fast Path programs
LTERM ready queue (05)	Messages destined for LTERMs that were generated by a Fast Path program

5.3.12 Queue names

IMS registers interest in queues by the queue name. The queue name is 16 bytes long and is made up of a 1-byte queue type, an 8-byte resource name, and in some cases, a 4-byte IMS identifier with three blanks. The types of resources are transaction names, LTERM names, and MSNAMEs. For APPC and OTMA transactions, a timestamp is used in place of a resource name to ensure that only the IMS that placed the transaction onto the queue can process the transaction.

If a message must be processed by a particular IMS, the identifier of the IMS that is to process the message is placed in the queue name. No other IMS has an interest in this unique queue name. For example, if TRANA has a requirement to be processed by IMSA, it would be added with a queue name of 'TRANA&blank.&blank.&blank.IMSA'.[1] IMSA is the only system that has registered interest in queue 'TRANA&blank.&blank.&blank.IMSA', so it is the only IMS system notified when one of these transactions is queued.

If it did not matter which IMS system processed TRANA, TRANA would be added with a queue name of 'TRANA'. Any IMS system that had registered interest in the queue name of 'TRANA' would be notified.

The full function queue names are made up as follows:

- Transaction ready queue X'01' + Transaction Code
- Transaction staging queue X'02' + Transaction Code
- Transaction suspend queue X'03' + Transaction Code
- Transaction serial queue X'04' + Transaction Code + IMSID
- LTERM ready queue X'05' + LTERM Name, for local LTERMs
- LTERM staging queue X'06' + LTERM Name
- APPC ready queue X'07' + Timestamp + IMSID
- Remote ready queueX'08' + MSNAME
- OTMA ready queueX'09' + Timestamp + IMSID

The Fast Path queue names are made up as follows:

- Program ready queueX'01' + PSB Name

[1] &blank. is the space character.

- LTERM ready queueX'05' + LTERM Name

5.3.13 IMS queue manager

All requests to process messages on the MSGQ structure go through the IMS queue manager. IMS messages are built in one or more queue manager buffers (long or short message logical records). If the message is longer than one long or short message logical record, the second through n^{th} logical records are placed on the staging queue. Partial messages in the staging queue are identified by a unique queue name that is generated by the queue manager. IMS does not register interest in the unique queue name assigned to the staging queue entries and therefore is not notified of partial messages placed on the staging queue.

The long or short message logical record containing the initial segment of the message is placed on the ready queue after the second through n^{th} logical records (if any) have been placed on the staging queue. Before placing the first logical record on the ready queue, the unique queue name used to place the second through n^{th} long or short message logical records on the staging queue is saved in the message prefix of the first segment of the message. This is required to be able to retrieve the segments of the message that were placed in the staging queue.

IMS registers interest in the ready queues and is notified when the queues go from empty to nonempty. Putting the second through n^{th} long or short message logical records on the staging queue before putting the first logical record on the ready queue prevents an IMS from retrieving a partial message.

5.4 Register interest

Each IMS system in the shared queues group has an associated CQS address space. CQS (the server) acts on behalf of its IMS system (the client) by placing messages on and removing messages from the shared queues. CQS does not actually know the content or meaning of the IMS messages. Instead, IMS tells CQS to place the messages on a particular queue in the shared queues structure. There are several queue types that are managed by CQS for transactions and LTERMs.

When an IMS system in the shared queues group starts or restarts, it informs CQS of the transactions defined in the system definition. CQS places these transaction names in a special table, the *interest area table*, in the Coupling Facility structure. When input messages are placed on the shared queues for a particular transaction, CQS checks whether the transaction is listed in its interest area table. If it is in the table and the queue goes from empty to nonempty, CQS notifies IMS that there are messages to be processed.

If the IMS system has an available dependent region with the proper class and priority to execute the transaction, IMS tells CQS to get the message from the shared queues. Because there could be many IMS systems in the shared queues group, the *first CQS* that requests the message, receives the message from the shared queues. In shared queues environments, the same transaction names can exist in different CQS interest area tables if the transaction is defined in more than one IMS system in the shared queues group.

Registration is the process of telling CQS (and CQS tells the CF) that IMS has an “interest” in any message with the queue name that is being registered. It means that IMS wants to be informed when there is a message on that queue, because IMS is capable of processing it. When IMS registers interest, an EMC is created in the structure.

IMS registers interest in a specific queue name at IMS initialization for statically defined transactions. At the completion of an IMS restart, IMS will register interest in transactions (for example, TRANX) that are defined to that IMS in the system generation. This is because it can immediately begin processing those transactions. Note that IMS does not register interest in LTERMs at this point because they have not yet logged on, and until they do, IMS could not deliver a message.

During normal operations, events occur that will cause IMS to register interest in additional resources when their status changes as follows:

- LU for static LTERM(s) logs on
- LU for dynamic (ETO) LTERM signs on
- MSC logical link started
- Stopped transaction has been started
- Transaction has been added by online change

In every case, it means that IMS is now capable of processing a message for one of those destinations, and so IMS will register interest in it. Note that for ETO, IMS does not register interest until the user signs on. There is no LTERM until signon is complete.

Deregistration is the opposite of registration, and means that IMS can no longer process the message. The EMC is deleted from the structure and IMS will no longer be informed when there are messages for that queue name.

During IMS termination, IMS will disconnect from CQS and CQS will deregister all of IMS's interest. During normal operations, deregistration can occur when IMS is no longer capable of processing the queue, that is when any of the following events occurs:

IMS does not register interest in queue names for every queue type — only the ones highlighted in Figure 5-11 (TRQ, TSerQ, LRQ, and RRQ). These contain messages that can be processed. Suspended transactions cannot be processed until they are removed from suspended status, messages on the staging queues will be read when their respective “first parts” have been read from the ready queue.

Technically, IMS does register interest in the APPC and OTMA queues, but it only does this when it puts a message on the queue and really knows it's there. No IMS will ever put a message on the APPC or OTMA queue for another IMS to deliver, but it is just as well to say that IMS does not register interest.

For Fast Path, there are only two queue types, and IMS may register interest in either of them. For the program ready queue, IMS will register interest in a PSB when the first IFP region for the PSB is started in the local IMS system. For the LTERM ready queue, IMS will not register interest until the first Fast Path message from that LTERM is PUT on the EMH queue. This is to reduce the number of registrations when there are few or no EMH transactions (remember, we will have an EMH structure even if we only access DEDBs — if we don't use the EMH, there is no sense in registering interest).

Archive

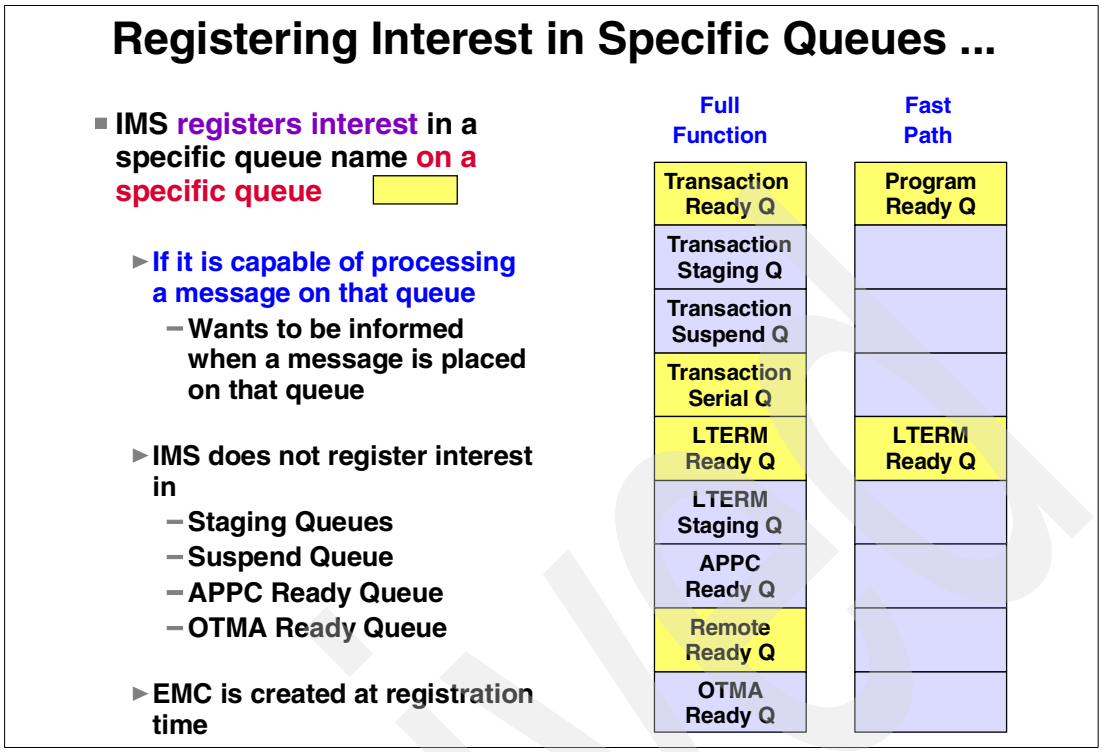


Figure 5-11 Register interest in specific queues

When interest is registered, an EMC is created. If the queue is empty at the time of registration, then the EMC is *not* put on the event queue. When the first message for that registered queue name arrives, the EMC is posted to the event queue(s) of all registered clients, and that client is informed that the queue has gone from empty to non-empty. As additional messages arrive for that queue name, there is no additional notification made, and when the queue name transitions from non-empty to empty, no notification is given. The only way that IMS knows a queue is empty is to ask for a message and then to be told there are none. It is similar to telling an MPP there are no more messages by letting it ask and then returning a QC status code.

IMS is informed of work by queuing an EMC to the event queue on a registered queue name, if there is work on the queue when IMS registers interest, or when the queue goes from empty to non-empty. When a queue goes from non-empty to empty, the EMC is dequeued but IMS is not informed. IMS finds out that the queue is empty the next time it asks for a message on that queue.

There are two ready queues for full function transactions:

- Transaction ready queue

  The transaction ready queue is used to queue messages for both local and remote transactions. If a transaction is defined as remote (TRANY in the picture), then only the IMS which has an active logical link to the remote SYSID (2) will register interest.

- Transaction serial queue

  The transaction serial queue is used to queue transactions defined with the SERIAL=YES parameter. There are special queue names for these transactions so that only the IMS which put the message on the queue will be notified there is work on the queue and will read and process it.

5.4.1 Registering interest in PSBs for Fast Path

When the first IFP region for a PSB is started in an IMS, IMS registers interest in the PSB to CQS. When an EMH message is put on the program queue for the PSB and causes the queue to go from empty to non-empty, CQS notifies every IMS that has previously registered interest in the PSB that there is a message on the queue.

If the IMS system has an available IFP region to execute the transaction, IMS tells CQS to get the message from the shared queues. Since there could be many IMS systems in the shared queues group, the *first CQS* to request the message receives the message from the shared queues. In shared queue environments, the same program names can exist in different CQS interest tables if the program is defined in more than one IMS system in the shared queues group.

Fast Path EMH transaction processing has changed with shared queues also. When any FP exclusive or FP potential transaction arrives in the front end IMS system, DBFHAGU0 is invoked. Without shared queues, this exit has several options about scheduling the transaction in the local IMS. With shared queues, there is additional information available to the exit, as well as some new options. These are passed to the exit in an extended parameter list (EPL).

To schedule an EMH transaction, the PSB to process that transaction must be started either in the local IMS or in a B-E IMS somewhere in the Parallel Sysplex. This information is passed to the exit in the exit parameter list.

There are also some new scheduling options that may be selected by the exit that determine whether the transaction should be processed locally or globally, and whether the transaction should be scheduled by PSB only without regard to the status of the routing code.

If the input message is Fast Path potential or Fast Path exclusive transaction, the message is passed to Fast Path Input Edit/Routing Exit (DBFHAGU0) and the new extended parameter list passed to exit. DBFHAGU0 knows which Fast Path PSBs are active in the shared queues group. DBFHAGU0 has new scheduling options:

- Sysplex processing code, that determines whether transaction is to be scheduled locally or globally
- Schedule by PSB name that overrides routing code scheduling

All of the new information and options are provided in the EPL. The address of the EPL is at the end of the SPL. If this address is all zeros, then this IMS is not running with shared queues. Based on that information, the user can code the exit to run in either environment, and because the EPL contains default sysplex processing code, users do not have to take special action if they default to use local first option. See Figure 5-12.

Archive

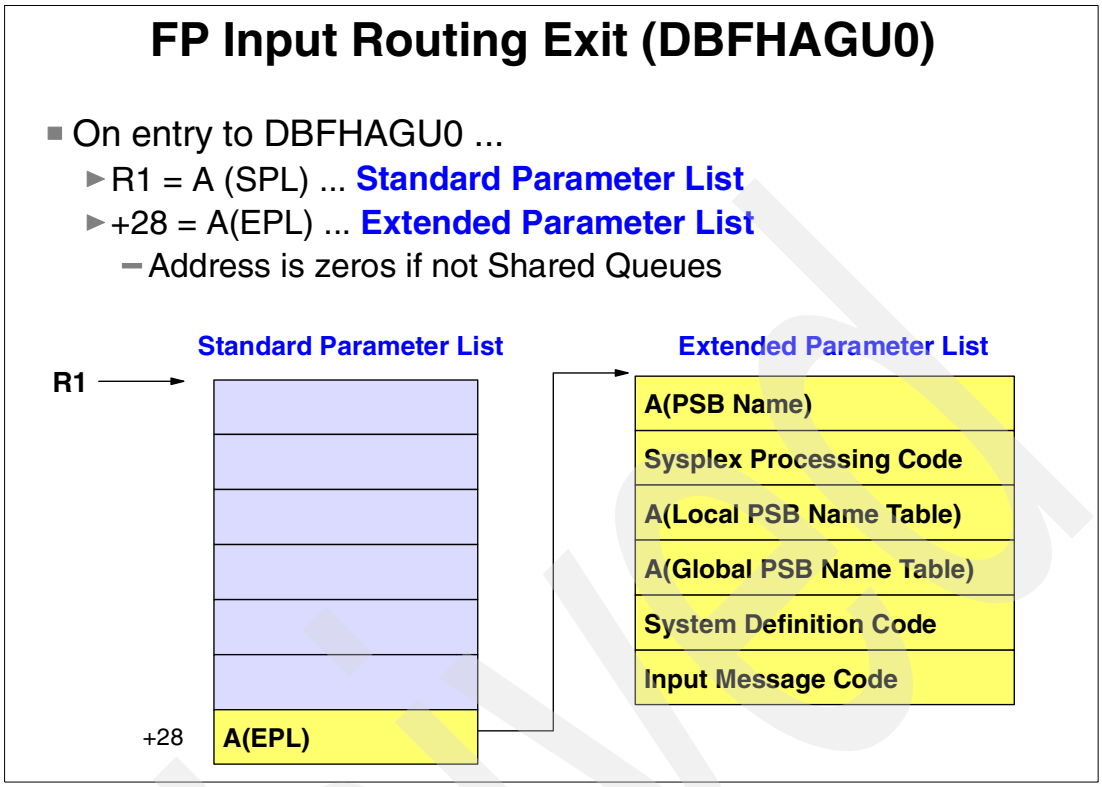


Figure 5-12 DBFHAGU0

5.4.2 Registering interest in terminals

LTERMs are also registered in the CQS interest area table. When an ETO user signs on or a user on a static terminal logs on, IMS notifies CQS to add an entry for the LTERM in the CQS interest area table. When messages are placed on shared queues for a specific LTERM, the CQS with that LTERM in the CQS interest table notifies IMS of the existence of a message, and the message is delivered to the IMS to which the terminal is connected. If the LTERM does not exist in any CQS interest area table, the message stays on the shared queues.

5.5 Significant status

An affinity is created whenever an LU connects to an IMS system. These affinities are usually deleted at session termination. Session termination can consist of user logoff, VTAM LOSTERM exit, IMS normal shutdown, or IMS failure. Affinities can remain after session termination, if the terminal has a significant status.

A significant status indicates that an affinity should remain after session termination, to maintain the consistency between a terminal and the IMS system. This is a list of significant status:

- Terminal in response mode
- Terminal in conversational mode
- Exclusive mode (set by **/EXCLUSIVE** command)
- Preset destination (set by **/SET xxxxxxxxx** command)
- MFS test (set by **/TEST MFS** command)
- 3600 Financial/SLUP (LU0) devices
- Operating system failure

An affinity remains in the case of OS/390 failure, because the IMS ESTAE exit is not executed in this situation. In the event of an IMS failure, affinities are deleted by the abend ESTAE routines.

For dynamic terminals, while a user stays logged on, the significant status is associated with that LU through the user. When the user signs off (which is done before session termination or logoff), the significant status is moved to the user structure, and the user structure is “disconnected” from the virtual terminal control block (VTCB). When the session terminates, there is no significant status associated with the session, and the affinity is always deleted.

For static terminal logoff, the entry in the table is also removed even if there is significant status for the terminal.

Existing affinities may cause problems in later session establishments. If there is an existing affinity for a terminal and a user attempts to connect to a specific APPLID, the session is created for that APPLID. Affinities are not globally known; they are held by the local IMS. When the user later re-establishes a session with the original IMS, significant status is restored.

Affinity™ deletion can be forced. Two IMS exits have been enhanced to allow deletion of affinities: the Signoff exit (DFSSGFX0) and the Logoff exit (DFSLGFX0). If the transaction manager portion of IMS is cold started, existing affinities are deleted.

Keep in mind that it is your responsibility in a shared queues environment to manage and control the deletion of significant status to keep terminal sessions consistent across IMS systems. If you have VTAM Generic Resources installed for static LTERMs, affinities are automatically maintained in VTAM's affinity table.

5.6 Accessing shared queues

Each IMS that has registered an interest in a queue is notified when the queue goes from empty to nonempty. IMS interface to CQS for full function services is through the IMS queue manager and for Fast Path services through the Expedited Message Handler. IMS can request CQS to perform one of six actions:

- PUT

  IMS puts messages on the shared queues when message is received by IMS:

  - Input from network
  - Input from MSC
  - Input from APPC or OTMA
  - Inserts by dependent regions

  The IMS that is putting the message does not need to have registered the interest to place the message on queue, but the destination (for example, transaction code) must be known by IMS (either statically or dynamically defined).

- READ and BROWSE

  IMS retrieves messages from the shared queues when IMS is ready to process the message:

  - Schedule a TRANSACTION
  - Send output to LTERM
  - Send output to MSNAME

- MOVE

  IMS may move a message from one queue to another, for example, in the following cases:

- Deadlock (U777)
- Suspended (U3303)
- Requeued by NDM Exit

- DELETE

  IMS deletes messages from the shared queues when all processing has completed. The input messages are deleted when they have been scheduled and committed and the output messages when they have been delivered and acknowledged. CQS batches deletes for performance therefore reducing the number of CF accesses.

- UNLOCK

  IMS may unlock the message if it cannot process it.

A read function gets the first list entry of a message from a queue, which causes it to be moved to the lock queue. If the message prefix contains a staging queue name, a browse would then be issued to get the remaining parts of the message from the staging queue. Partial messages on a staging queue do not have to be locked, because the only way to access them is from the first segment of the message on the lock queue.

The difference between a read and a browse is that a read gets the contents of the list entry and moves the list entry to the lock queue, whereas a browse only gets the contents of the list entry.

Although messages are stored on the structure and the structures are the basis for all message queue integrity and recovery, the queue pool still exists in a shared queues environment. It is used for the intermediate storage of messages. The message queue data sets are not used with shared queues.

For example, an input message is first put into the queue pool before being moved to the shared queues structure, and messages inserted by applications are put in the queue pool before being put on the shared queues structure. When IMS retrieves a message for processing, it queues it in the queue pool the same as in a non-shared queues environment — the difference being that there is usually just one message there instead of a whole long queue of them. Perhaps the biggest use of the queue pool in terms of storage is that the SPA and last output message of a conversational transaction is kept in the queue pool between iterations (a copy is also kept on the LOCKQ for recover ability). Occasionally, when there is a problem with CQS or with the structures, IMS will save messages in the queue pool until the problems can be resolved.

IMS Fast Path accesses the EMHQ through the EMH. New options have been added to the Fast Path Input Edit/Routing Exit (DBFHAGU0). These are discussed later.

5.6.1 Message queue overflow structure

When you define your message queue structure, you can define an optional overflow structure. The overflow structure is used to store the largest queues on the primary structure when the primary structure reaches a certain percentage of use. The percentage value is set in the OVFLWMAX parameter in the global CQS PROCLIB member. The largest queues are moved from the primary structure to the overflow structure to assist in preventing the primary structure from becoming full. The primary structure and the overflow structure together are known as a *structure pair*. This applies for both MSGQ and EMHQ structures.

If a printer was stopped and output was still being queued to it, rather than this one printer LTERM causing the MSGQ structure to fill up, the worst case scenario would be that it would fill up the overflow structure. All queues that are still in the primary structure are affected by the overflow structure filling up.

5.6.2 Overflow processing

When the overflow threshold is exceeded and the overflow structure is defined, structure activity is quiesced while the queue names that are using the largest number of data elements are identified. The identified queues are moved to the overflow structure until the data element usage drops by a minimum of 20%. The moving of one large queue could result in the drop being far greater than 20%. A queue cannot exist in both the primary structure and the overflow structure. If a queue is selected to be moved, the whole queue is moved from the primary structure to the overflow structure. A maximum of 512 queues can be moved, to reduce the data element usage by 20%. You can make use of a user exit if you would like to exclude specific queues from overflow processing. You can determine which queues are in the overflow structure by issuing the **/DISPLAY OVERFLOWQ STRUCTURE ALL** command. If the defined threshold is exceeded again, the same processing occurs, to reduce the data element usage by 20%.

The following tasks are performed during overflow processing:

1. CQS quiesces the structure.
2. CQS identifies the queue using the largest number of data elements.
3. CQS moves the identified queue to the overflow structure.
4. CQS checks to see what percentage of the structure is being used. If the number of data elements in use has not decreased by 20%, another queue is identified and moved.

This process continues until the number of data elements that are being used has decreased by at least 20%, or 512 queues have been moved. On completion of overflow processing, a structure checkpoint is taken.

The queue remains in the overflow structure until the number of entries on the queue gets to zero. Periodically, a scan is run to look at the queue names that are in the overflow structure. If these queues have zero entries queued, the queue name is moved back to the primary structure. Having a large queue in the overflow structure isolates and reduces the impact that it can have on the overall system. In the case where we have one stopped printer, the worst case scenario is that the overflow structure eventually fills up, and further messages cannot be put onto the queues in the overflow structure. The primary structure continues to function normally.

Overflow processing should be performed only during exceptional circumstances. If it is being performed regularly, consider making the primary structure larger and look to see whether there are any problems with message delivery (such as output queuing to a stopped printer).

5.6.3 Processing without an overflow structure

If a primary structure is defined without an overflow structure, CQS still goes through the process of determining which queues would have been moved to the overflow structure. All puts to the identified queues are rejected, and the queues remain in the primary structure. Overflow threshold checking is not performed again.

Note: If an overflow structure is not defined, you can add one only by cold starting the primary structure.

5.6.4 Full structure

If either the primary or the overflow structure becomes full, puts to that structure are rejected. A structure can become full if either all the list entries or all the data elements are used. A

structure-full condition is usually a temporary one, and IMS does not stop attempting to put messages on the queue. An application program receives an **A7** or **QF** status code if it attempts to do an ISRT call to a queue that is on a full structure. If IMS has committed output messages that have not yet been placed on the shared queue, they are saved in IMS queue buffers and moved to the shared queue when space becomes available. Input messages are rejected with the following messages:

```
DFSO7O UNABLE TO ROUTE MESSAGE (for non-EMH terminals)
DFS2194 SHARED EMHQ NOT AVAILABLE (for EMH terminals)
```

5.6.5 CQS checkpoint data sets

Each CQS maintains local VSAM entry sequenced data sets (ESDSs) for use as checkpoint data sets. There is always one for the MSGQ structure and, if Fast Path is defined, there is also one for the EMHQ structure. They are dynamically allocated to CQS when it initializes. When a system checkpoint is taken, CQS writes its control blocks and tables to the log stream, to support CQS restart. A CQS checkpoint is similar in concept to an IMS simple checkpoint. The location of the system checkpoint information within the log stream is recorded in the CQS checkpoint data set and in the structure on the control queue.

CQS performs a system checkpoint in the following situations:

- Automatically, based on CQS log volume. The SYSCHKPT parameter in the CQSSLxxx PROCLIB member defines how many CQS log records are to be written before a checkpoint is issued.
- Manually, when the **/CQCHKPT SYSTEM** command is entered.
- At the completion of a successful structure checkpoint.
- At CQS shutdown time.
- At the completion of a CQS restart.
- After a client resynchronizes with CQS.

Figure 5-13 shows two CQS address spaces, each with its own local pair of checkpoint data sets.

Archived

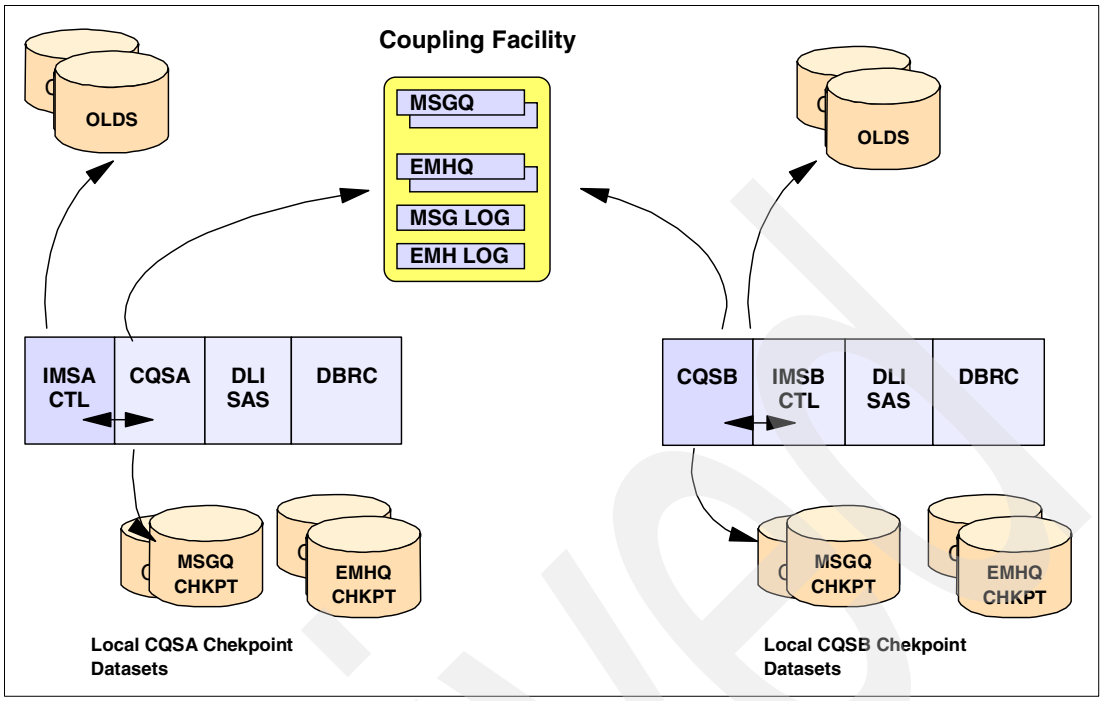


Figure 5-13 Checkpoint data sets

5.6.6 CQS structure recovery data sets

CQS uses a pair of SRDSs per Coupling Facility structure pair for its structure checkpoint processing. These data sets are shared across all CQS address spaces in the shared queues group. When a structure checkpoint is requested, CQS dynamically allocates an SRDS. Structure checkpoint requests alternate between the two SRDSs. After a successful structure checkpoint has been taken, and the data objects have been copied to the SRDS, that SRDS is used if a structure recovery is required. The alternate SRDS (the oldest) is the one to which the next structure checkpoint is written. The oldest SRDS is also used if a structure recovery is required and the most recent SRDS cannot be read. During structure checkpoint processing, all activity to the structure is quiesced while recoverable data objects on the structure are written to a data space, before they are written to the SRDS. The size of each SRDS should be at least the size of the primary structure plus the overflow structure to ensure that all objects can fit.

To minimize the length of time the structure is quiesced during structure checkpoint processing, CQS first writes the data objects to a data space. The structure is quiesced while the data objects are being written to a logger data space. Once the data objects have been written to the logger data space, activities against the structure are resumed. CQS then completes the structure checkpoint process in the background by copying the data objects from the CQS-owned data space to an SRDS. Because no other work for a structure can be processed while CQS is checkpointing the structure, we recommend that structure checkpoints be performed during nonpeak hours.

In addition to recoverable objects being copied from the structure to the SRDS, log records prior to the oldest structure checkpoint are deleted because they are not required for a structure recovery. If a CQS that is part of the shared queues group is not running while two structure checkpoints are taken, the log records for its last CQS system checkpoint will have been deleted. When the CQS is restarted, it must be brought up cold. To avoid this situation, bring the CQS straight back up after the IMS system that it supports is shut down.

To prevent a log-full condition, which is where the number of log data sets has reached the limit or there is no more DASD space for the allocation of log data sets, regularly initiate a structure checkpoint. A structure checkpoint reduces the time it takes to perform a structure recovery if required.

CQS performs structure checkpoints in the following situations:

- When the log stream approaches a full status
- After a successful structure rebuild (unless it was a structure copy initiated by an operator)
- After a successful overflow threshold process
- After the **/CQCHKPT SHAREDQ STRUCTURE structurename** command is entered to any CQS in the shared queues group
- At CQS shutdown if the **/CQSET SHUTDOWN SHAREDQ** command was previously issued

Figure 5-14 shows the CQS SRDSs that are shared by CQSs in the shared queues group.

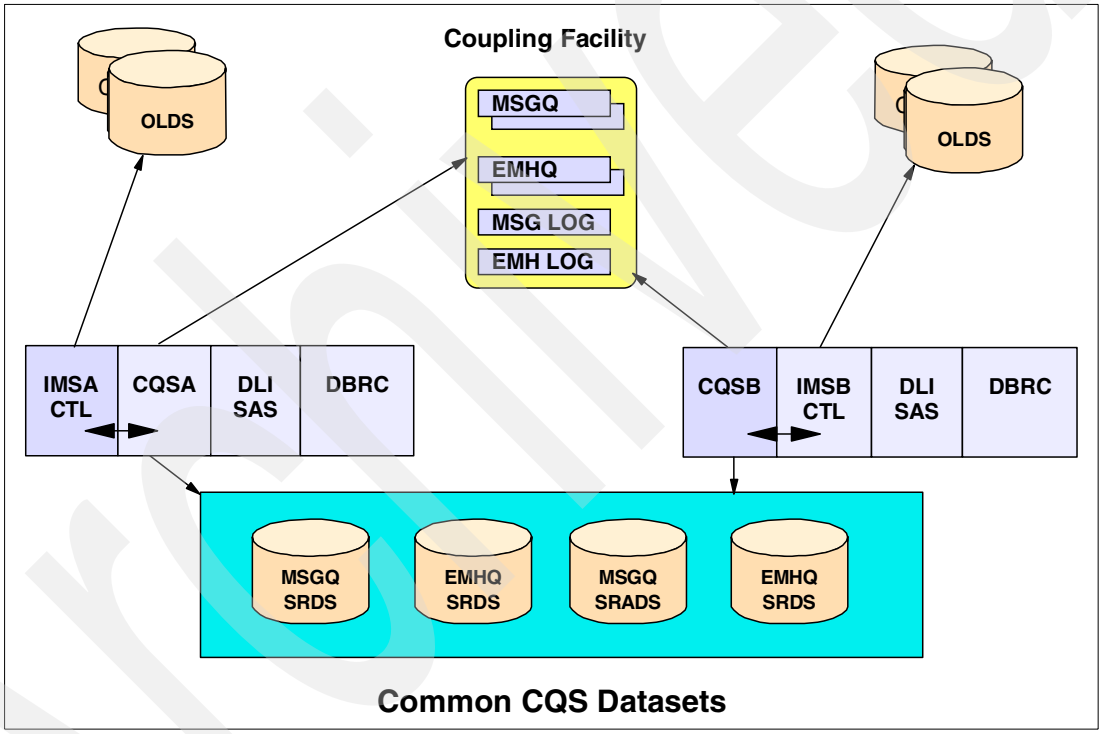


Figure 5-14 Structure recovery data sets

5.6.7 Structure integrity

Because the processing of messages is often done by multiple IMSs in the Parallel Sysplex, the integrity of these messages is the responsibility of CQS — the focal point for all messages across the sysplex. To accomplish this, CQS performs structure and system functions that have the same intent as equivalent IMS functions.

CQS logs all message and structure activity into and out of the shared queues. Because there are multiple CQSs, and just one structure (pair), all CQSs use the OS/390 System Logger to log this activity to a log stream defined in the LOGR policy. This log stream is a merged log of all CQS log records (unlike IMS logs in a data sharing environment which must be merged by Change Accumulation utility prior to using them for recovery).

Archived

CQS takes system checkpoints, with the log records also going to the System Logger. The log token of the beginning of the checkpoint log records is kept in a checkpoint data set for each CQS. Another copy is kept on the CONTROLQ in the structure. Think of the checkpoint records, such as the IMS x'40xx' system checkpoint records, and the checkpoint data set, such as the IMS restart data set (RDS).

CQS also takes structure checkpoints. These are logical copies of the contents of the structure at the time the structure checkpoint is initiated. You can call them logical copies because only the data content is copied, not the entire structure. Unused list entries, for example, are not copied. This is unlike IMS database image copy where a physical copy is taken, including empty blocks.

With the above data, any CQS can recover a structure if it is lost due to structure damage or CF failure. If a shared queue structure is lost, CQS can recover it from the last structure checkpoint and the appropriate log stream (full function and Fast Path have separate log streams). The OS/390 System Logger is described under 5.6.9, “OS/390 System Logger” on page 120.

5.6.8 CQS security

If RACF or another security product is installed at an installation, the security administrator can define profiles that control the ability of clients to connect to and access CQS shared queue structures. When an IMS client issues the CQSCONN request to connect to a queue structure, CQS issues a `RACROUTE REQUEST=AUTH` call to determine whether the client is authorized to access the structure. If the client's userid has at least update authority, the client can connect to the structure. The profile names must be of the form `CQSSTR.structure_name`, where structure_name is the name of the primary structure to be protected. The structure names are specified in the CQSSGxxx and CQSSLxxx PROCLIB members. CQS does not perform a separate check on the overflow structure name because the primary and overflow structures are considered to be one unit.

If a profile is not defined for a CQS structure, the structure is not protected, and any client can issue a CQSCONN request to access the structure. See *z/OS MVS Programming: Sysplex Services Guide,* SA22-7617, for more information about protecting access to Coupling Facility structures.

Note: If the RACF database is not shared by IMS systems belonging to the same shared queues group, RACF definitions (profiles) should be synchronized to achieve same results on security checking for all IMSs in the group.

In a shared queues environment when user exit routines are called in back-end systems because of an application program AUTH or CHNG call, the address of the CTB is zero, because that control block is not in the IMS subsystem making the call and because the security environment (ACEE) is built dynamically by IMS in the dependent region. IMS dynamically creates a security environment in the dependent region when CHNG or AUTH call issued from the back-end system or from the front-end system where the user has signed off. Dynamic creation of lots of ACEEs has performance implications and Build Security Environment Exit Routine (DFSBSEX0) may be coded to allow bypass of dynamic creation of ACEE in the dependent region for CHNG and AUTH calls and bypass some part of the security checking process.

In a shared queues environment, the CTB address contains zeroes in back-end systems. Exits using CTB address may have to be changed to work in a manner that does not require CTB to get to ACEE. Those exit routines may have to be changed to use RACF provided functions, such as RACROUTE REQUEST=EXTRACT macro to locate the user's ACEE.

Installations that use the same exit routines in shared queues and non-shared queues environments may need to change exit routines to function properly in both types of environments. The exits to be checked are:

- Command Authorization Exit Routine (DFSCCMD0)
- Transaction Authorization Exit Routine (DFSCTRN0)
- Security Reverification Exit Routine (DFSCTSE0)

5.6.9 OS/390 System Logger

The OS/390 System Logger is an OS/390 component that allows applications to log from different systems within a sysplex to a central point. The System Logger merges the log data from several systems in the sysplex into a single log stream. A log stream is essentially a collection of data in log blocks residing in a Coupling Facility structure, on DASD, or a combination of the two. The logging structures and the log streams are defined in the CFRM policy. For a full description of the System Logger, refer to *z/OS MVS Setting Up a Sysplex,* SA22-7625.

CQS uses the System Logger to record all information necessary for it to recover shared queue structures. CQS writes log records for each structure pair to a separate log stream. The same log stream is written to by all of the CQS address spaces in the one shared queues group. The System Logger manages the log stream and provides a merged log for all CQS address spaces that are connected to a message queue structure on a Coupling Facility. Figure 5-15 shows two CQS address spaces writing to a shared log stream.

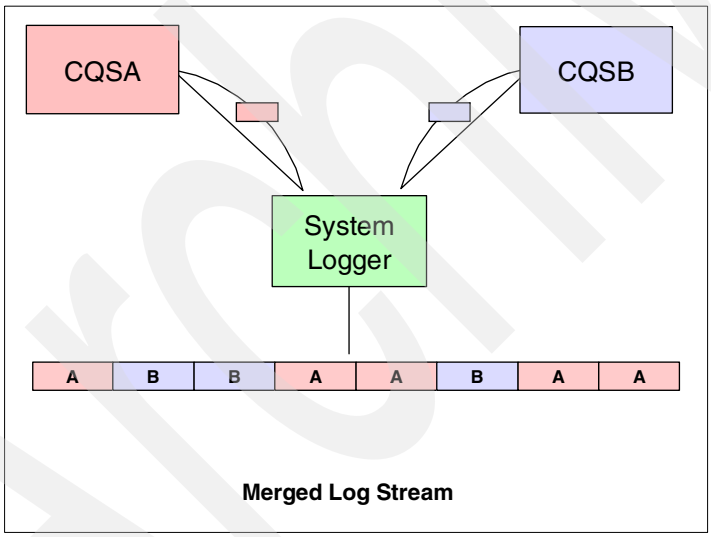


Figure 5-15 A merged log stream

The System Logger can use the following components:

- Logger data space
- Staging log data sets
- Log data sets

5.6.10 OS/390 logging structure

The logging structure stores the logging data written by each of the CQSs that are writing to the same log stream on a Coupling Facility. There is one logging structure for each shared queue structure pair.

5.6.11 System Logger data space

There is one OS/390 System Logger per OS/390 image. Each System Logger maintains a copy of the log data that it writes to the logging structure in its own logger data space unless the structure has a staging log data set defined. If a logging structure on a Coupling Facility needs to be recovered and a staging log data set was not being used, all of the System Loggers are required to participate because each logger data space has part of the data that is required to rebuild the log stream in the logging structure.

Figure 5-16 shows both the logger data space (1a) and the staging log data set (1b). Only one of the two methods of duplexing the log stream can be used. You cannot use both methods at the same time.

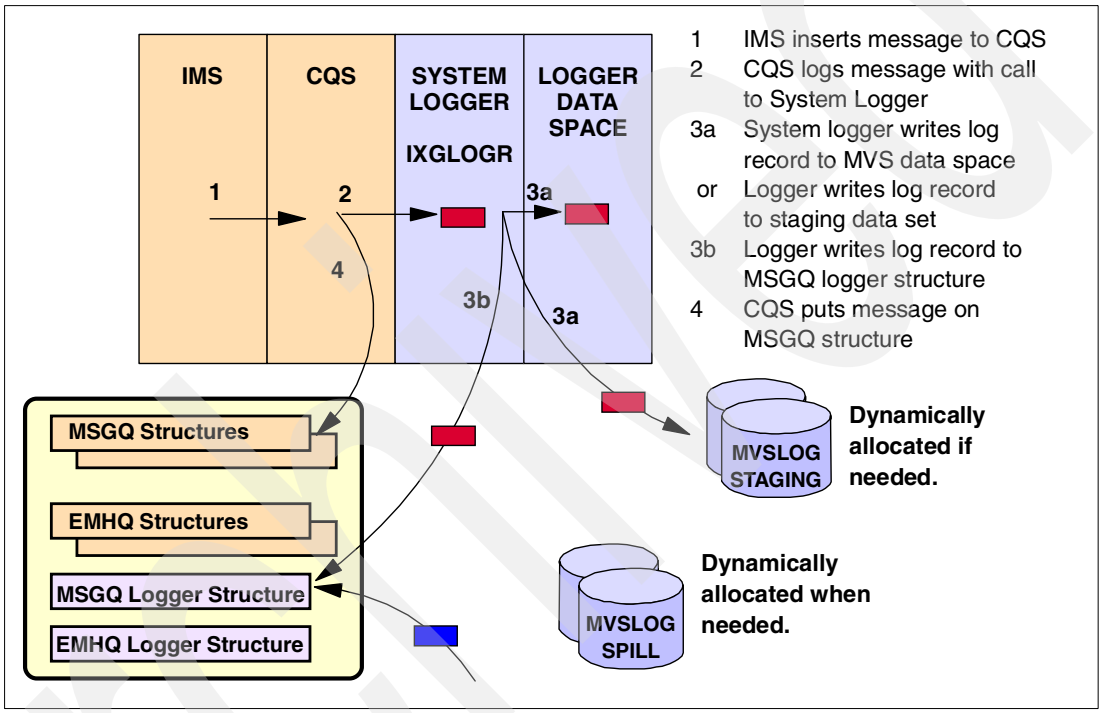


Figure 5-16 Logging components

5.6.12 OS/390 staging log data set

Staging log data sets can be used to store a copy of what is in the logging structure on the Coupling Facility. If your configuration contains a single point of failure, then you should use staging log data sets in preference to the logger data space. However, duplexing (where available) eliminates the single point of failure and is recommended.

In the case of the log stream, a single point of failure can be when the log stream is unable to be rebuilt after the failure of a single item. If your Coupling Facility is volatile (no backup power supply) or is on the same processor box as a system that is connecting to it, you have a single point of failure. In this case, losing power to the site or the processor box failing results in a loss of your logging structure. If you have a single point of failure, we recommend that you use staging log data sets. In a production environment, it is usually unacceptable to allow any single point of failure to interrupt your business processing.

A staging log data set contains a copy of the log data that is written to a logging structure by the local System Logger. The staging log data sets from all of the System Loggers that are writing to a common log stream are used to recover a logging structure. The staging log data

set should be on DASD that is accessible by all of the System Loggers in the shared queues group. Staging log data sets are required only if the logging structure needs to be recovered. The System Logger performing the recovery reads each of the staging log data sets to reconstruct the logging structure.

5.6.13 Log data sets

OS/390 System Logger log data sets are used when the space used in either the logging structure or any of the staging log data sets for a log stream exceeds the value of the HIGHOFFLOAD parameter. This parameter is defined in the LOGR policy. When the value of the HIGHOFFLOAD parameter is exceeded, the System Logger offloads data from the Coupling Facility to a log data set. The amount of data that is offloaded is determined by the LOWOFFLOAD parameter in the LOGR policy. The logging structure is not quiesced while the offload is occurring. The HIGHOFFLOAD parameter should not be set too high, because log data continues to be added while the offload is occurring, and, if the logging structure fills up, puts to the structure are rejected. We recommend that the LOWOFFLOAD parameter be set to 0, so that all of the data in the structure when the offload commences is written to the log data set through the logger data space. All data that is written to the structure after the commencement of the offload process is not offloaded.

The number of log data sets that you can allocate is 168. If you are using OS/390 Release 3 or later, you can increase the number of data sets by using the DSEXTENT parameter in the LOGR policy. For more information about the DSEXTENT parameter, refer to *z/OS MVS Setting Up a Sysplex,* SA22-7625.

Archives

6

Transaction flow in a shared queues environment

In this chapter we discuss the path taken by an IMS transaction from the time it is entered from a terminal, through to the response back to the terminal. We discuss both full function and Fast Path transactions.

Archived

© Copyright IBM Corp. 2003. All rights reserved.

6.1 Processing a full function transaction

A full function transaction is always passed to CQS to be put on the shared queue structure. A full function transaction can be processed by the local IMS system when it is first received, or any of the IMSs in the shared-queues environment that have a registered interest in the queue can bid for the right to process the transaction. The exception to this rule is when the transaction is a SERIAL type, in which case it can only be processed by the IMS system which received it (the front-end IMS). With IMS Version 6 also the APPC and OTMA originated messages are exceptions to this rule. IMS Version 7 began shared queues support for asynchronous APPC and OTMA messages, and IMS Version 8 supports all APPC and OTMA transactions.

When an IMS system is notified that work is available on the transaction ready queue, IMS updates a flag (new with shared queues) in the scheduler message block (SMB) control block to indicate that work is available for this transaction type. This flag is updated to indicate that there are no more transactions on a queue when a message processing program (MPP) on the IMS system performs a get unique (GU) call and receives a `QC` status code.

IMS must have the following resources available to process a transaction:

- A message processing region (MPR) that is waiting for work. This MPR must be able to process the class for which that transaction has been assigned.
- The PSB that runs the transaction must be started.

6.1.1 Local processing

When the local IMS system receives a full function transaction, IMS determines whether the transaction can be processed immediately in the local IMS system before it places the transaction on the transaction ready queue. The transaction can be processed immediately only if the resources are available. If the transaction can be processed immediately in the local IMS system, the message is queued to the lock queue rather than the transaction ready queue on the MSGQ structure. The local IMS system then processes the transaction. Because the transaction is not placed on the transaction ready queue, other IMS systems that have a registered interest in this queue are not notified. This is far more efficient in that only one IMS system performs scheduling, and the overhead for notifying all of the registered CQSs is removed. Where possible, you should try to process the majority of full function transactions within the local IMS system. The number of transactions processed by the local IMS system is affected by the MPR occupancy level. Low MPR occupancy levels increase the chance that the full function transaction can be processed by the local IMS system.

6.1.2 Global processing

If the local IMS system did not have the resources available to process the transaction immediately, CQS passes the transaction to the transaction ready queue. If there are no other entries on the queue for this transaction, all IMSs that have a registered interest in the queue are notified. Each notified IMS having the resources available performs all scheduling functions except passing control to the dependent region controller. IMS requests the message from CQS, and the message is passed to the MPP when the MPP issues the GU call. The first CQS to perform the read function retrieves the first list entry on the queue and passes the transaction to IMS to execute it. If there was only one of this type of transaction on the queue, all other CQSs attempting to retrieve the transaction are unsuccessful. The empty message queue is discovered during message priming and the schedule fails before the application is loaded and given control by the program controller. This is called a false schedule or a pseudo-scheduling failure.

6.1.3 Scheduling for full function transaction

Scheduling of transactions for processing within IMS is still very much the same as without shared queues, with few exceptions. Since it is the CQS that now manages the queues, IMS no longer knows how many messages are on a queue. IMS is notified by CQS that at least one message is on the queue (although it may be gone by the time IMS asks for it), but not how many. So the scheduling parameters which are dependent on the length of the queue are ignored when shared queues is active.

The PARLIM parameter in the TRANSACT macro says that, when the number of messages in the queue is greater than the number of dependent regions where this transaction is already scheduled times the value of PARLIM, then schedule another dependent region (unless the number of dependent regions has already reached MAXRGN). With shared queues, we don't know the first number — how many messages are in the queue. In the beginning IMS ignored the specification and if there were ANY messages in the shared queues, and if MAXRGN had not been reached, then another dependent region was scheduled. In other words, IMS scheduling with shared queues worked as though we coded PARLIM=0.

Parallel processing and the use of PARLIM changed significantly with APARs PQ39919 and PQ65633, and is now very much the same as with traditional DASD queueing. IMS now counts the number of successful GUs and when that number exceeds the PARLIM value times the number of started dependent regions, it starts another one.

The other difference is the limit priority and limit count on the PRTY parameter. In a non-shared queues environment, it means that when the number of transactions on the queue exceeds the limit count, increase the scheduling priority to the limit priority. In the shared queues environment again, because IMS doesn't know the count, it never increases the scheduling priority, and the limit count and limit priority are ignored. Transactions are always scheduled according to normal priority (10,20 are ignored) in Example 6-1.

Example 6-1 PRTY

```
APPLCTN PSB=PGMX,SCHDTYP=PARALLEL
TRANSACT CODE=TRANX,PARLIM=0,MAXRGN=5,
    PRTY=(7,10,20)
```

IMS is told when a queue transitions from empty to non-empty. This is flagged in the SMB (Scheduler Message Block) for the transaction. Until IMS finds out otherwise, it will continue to assume there is at least one message for that transaction in the queue, and will schedule a dependent region for it according to its standard scheduling algorithms (with the exceptions noted on the previous foil).

Scheduling is done in two phases. The first phase, called pseudo-scheduling consists of all scheduling steps up to but not including the passing of control to the dependent region. During this phase, the message is not in the queue pool, it is still on the structure (maybe).

IMS then issues a READ request to CQS for the transaction and gets the first list entry into one queue buffer. The message is requeued to the LOCKQ to prevent other IMSs from reading it. If the message prefix indicates more, then IMS will go back to CQS with a BROWSE request to the staging queue to get the rest of the message.

Once the entire message has been read into the queue buffers, the message is **primed** and the dependent region is given control, issues a GU IO-PCB and begins processing.

Note: If the message is gone when IMS tries to READ it (some other IMS got it first) then we have a pseudo-scheduling failure and the schedule process is stopped, the SMB is updated to indicate there are no more messages for this transaction, and IMS goes on to schedule something else.

To avoid some of the overhead of putting a message on the queue and then just pulling it off again, there is a process called LOCAL=YES processing. It is not an external parameter, but there are actions the user can take to make it more effective.

When a transaction arrives in the front end IMS system, if the following conditions are true:

- It can be scheduled immediately into a dependent region.
- The entire message fits into one queue buffer.
- The transaction is not defined as SERIAL=YES.
- The message is not originated from a program-to-program switch.

Then, IMS will process the transaction locally (LOCAL=YES) and avoid a significant amount of overhead. For LOCAL processing, IMS puts the message directly onto the LOCKQ (other IMSs are not notified), keeps a copy in the queue buffer, and schedules the transaction to be processed. The rest of the scenario is the same.

The overhead saved is by not putting it on the ready queue and then turning right around and reading it and moving it to the lock queue. Also, it potentially avoids pseudo-scheduling failures in other IMSs that were informed there was a message on the queue, only to find out it was gone when they ask for it.

The user should consider taking actions to increase the probability of this occurring. These might include increasing the number of dependent regions (decreasing region occupancy), decreasing the number of scheduling classes (increasing the number of regions eligible to run a transaction), and eliminating SERIAL processing wherever it is not required.

When the application ISRTs a message to the IO-PCB or an ALT-PCB, the message is placed into a queue buffer. When the application commits, the queue buffer holding the input message is freed, the input message is deleted from the structure, and the output message(s) are queued to their final destinations on the structure.

Application program output (ISRT) messages are placed (PUT) on the shared queue according to their destination, that can be any of the following:

- LTERM ready queue
- APPC/OTMA ready queue
- Remote ready queue (remote LTERM or MSNAME)
- Transaction ready queue (pgm-pgm switch)
- Transaction serial queue (pgm-pgm switch)

6.1.4 Full function transaction flow

In this section we describe how a full function transaction is processed. The transaction flow relates to the following four figures: Figure 6-1 through Figure 6-4 on page 130.

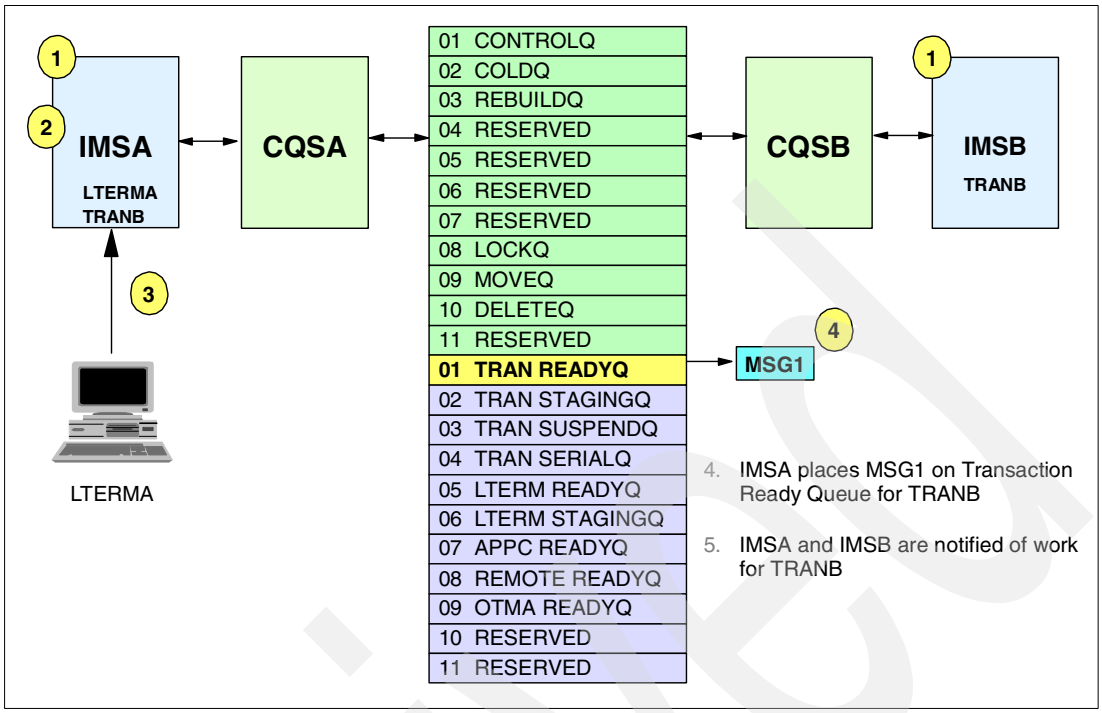


Figure 6-1 Full function transaction flow (part 1)

1. IMSA and IMSB register interest in the statically defined transaction, TRANB, when IMS is initialized or when the transaction is started.
2. IMSA registers an interest in statically defined LTERMA when the terminal logs on.
3. IMSA receives MSG1 from LTERMA.
4. IMSA checks to see whether it has resources available in the local IMS system to process the message immediately. If the resources are available, IMSA requests that CQSA put MSG1 on the lock queue and proceeds to process the message. In this example, the resources are not available, so IMSA requests that CQSA put MSG1 on the transaction ready queue for TRANB.

Archived

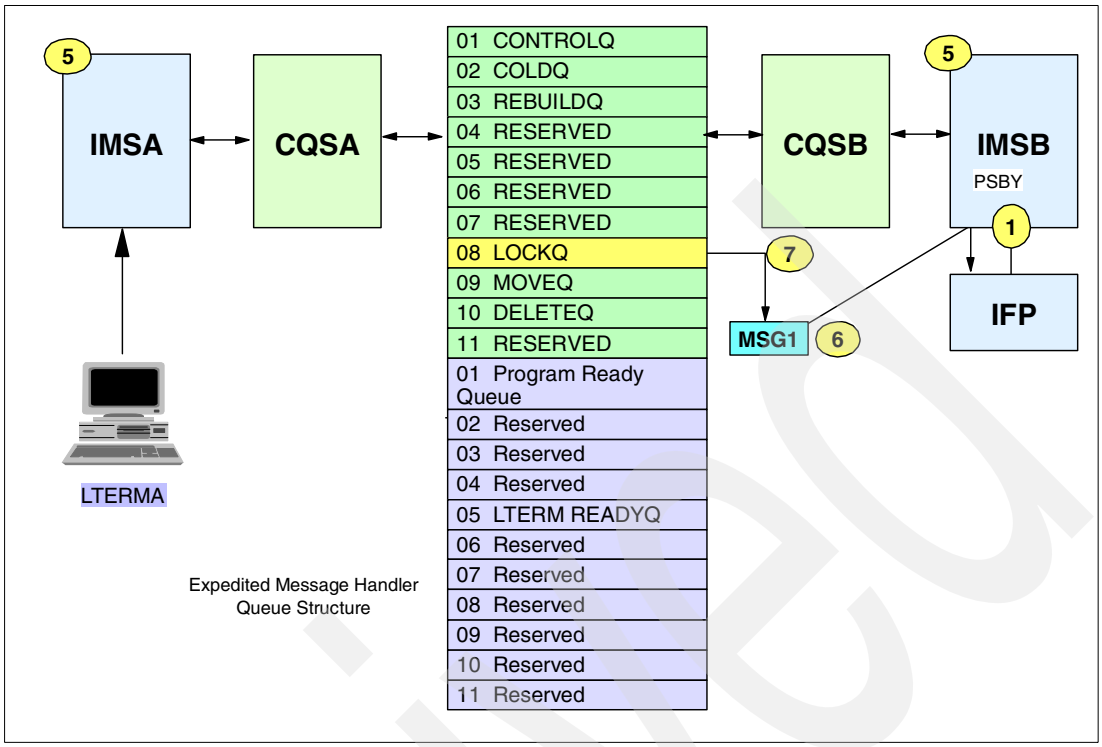


Figure 6-2 Full function transaction flow (part 2)

5. IMSA and IMSB are notified by the CQSs that the TRANB queue has gone from empty to non-empty.

   IMSB performs scheduling.

6. IMSB reads MSG1 for TRANB.
7. TRANB is moved from the transaction ready queue to the lock queue.
8. IMSB loads the program for TRANB.

Archive

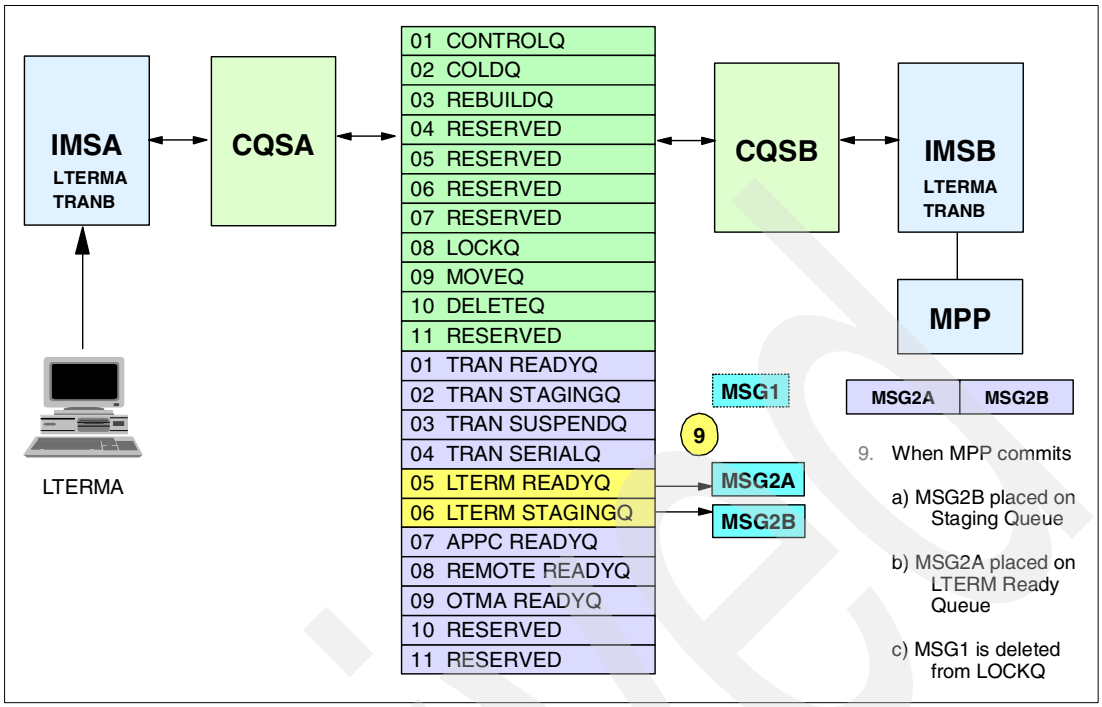


Figure 6-3 Full function transaction flow (part 3)

9. The MPP processes TRANB and inserts the reply (MSG2) in two queue buffers for LTERMA. The MPP then commits the processing. MSG2B is placed on the LTERM staging queue, and MSG2A is placed on the LTERM ready queue. CQSB puts MSG2B on the LTERM staging queue before MSG2A is put on the LTERM ready queue. When the MSG2B segment has been put on the LTERM staging queue, the MSG2A segment is put on the LTERM ready queue. Putting the message on the LTERM ready queue causes IMSA to be notified that the LTERMA queue has gone from empty to non-empty. The full message is then available for retrieval.

   IMSB then requests CQSB to delete MSG1 from the lock queue.

Archive

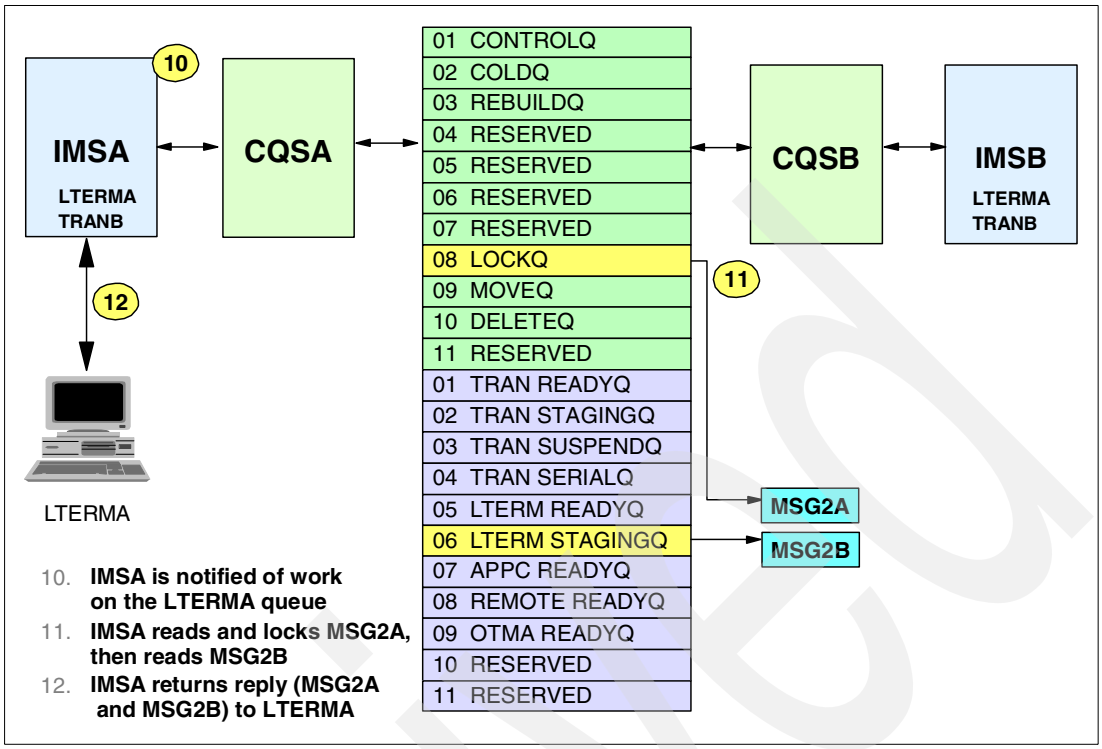


Figure 6-4 Full function transaction flow (part 4)

10. CQSA notifies IMSA of work for LTERMA.
11. IMSA passes a read request to CQSA. CQSA then locks MSG2A and reads the segments of the message. The message segments are then passed to IMSA.
12. IMSA returns reply messages to LTERMA.

    When the terminal has acknowledged receipt of the message, IMSA requests CQSA to delete MSG2A and MSG2B from the structure.

6.2 Processing a Fast Path transaction

Fast Path transactions can also take advantage of shared queues. Fast Path exit DBFHAGU0 can decide whether the transaction is to be processed locally or placed on the EMHQ structure. The default setting is local first. IMS attempts to process the transaction locally, and if that is not possible, the transaction is placed on the EMHQ structure.

Each Fast-Path-defined IMS system maintains a number of tables which are used to indicate the PSBs that are active within the shared-queues environment. When an IMS system receives a Fast Path transaction, it needs to determine whether the transaction can be processed by an active IFP region. This decision is made by doing a lookup in the tables. If there is no active IFP region in the shared-queues group that can process the transaction, a message is sent to the user indicating that the transaction cannot be processed.

A local PSB name table is created on the system where there is an active IFP region servicing that PSB. It is created only when the first IFP region servicing that PSB is started, and it is deleted when the last IFP region servicing that PSB is stopped. Each local PSB name table maintains a counter recording the number of IFP regions servicing the particular PSB on the local IMS system.

When the local PSB name table is created, a message indicating that a particular PSB is now being serviced by an IFP region is sent to all other IMSs in the shared-queues group. The message indicates the IMS system name that is servicing the particular PSB, and each IMS system in the shared queues group either creates or updates its global PSB name table. The global PSB name tables contain a list of the IMS system names where the PSB is being serviced by an active IFP. For each PSB that has an active IFP region servicing it, there is a local PSB name table on the local IMS system, and all other IMSs in the shared-queues group have a global PSB name table. It is possible and probable for an IMS system to have both a local PSB name table and a global PSB name table for a particular PSB. This is the case if the same PSB is being serviced by one or more IFP regions in its local IMS system as well as one or more IFP regions in its sharing IMS systems.

Figure 6-5 shows three IMS Fast Path systems in a shared-queues group. Each IMS system has a list of global and local PSB name tables. Each global PSB name table lists which IMS systems in the shared-queues group other than the local IMS system have IFP regions active that are servicing a PSB. Each local PSB name table shows which PSBs are being serviced by IFP regions in the local IMS system. The numbers in parentheses, in the local tables, represent the number of IFP regions in the local IMS system servicing the PSB. When a local PSB name table count goes from 1 to 0 the local IMS system notifies other IMS systems in the shared-queues group that it is no longer servicing the PSB. All other IMS systems update their global PSB name table.

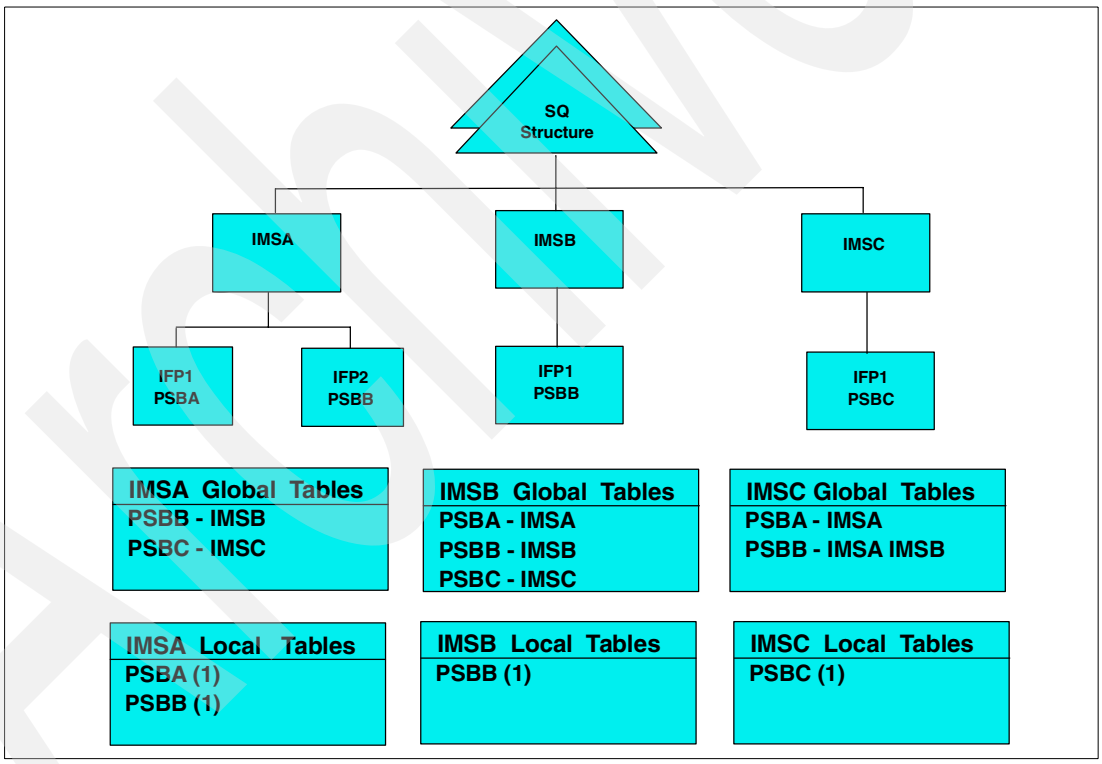


Figure 6-5 Global and local PSB name tables

If an EMHQ structure contains transactions that are waiting to run, and the last IFP region in the shared-queues group that can process them is stopped, the transactions are deleted from their queue name and the following message is sent to the inputting terminal to unlock it:

```
DFS2529I NO FAST PATH REGION IS ACTIVE
```

When an IMS system shuts down or abends, all other IMS systems in the shared-queues environment update their global PSB name tables for all PSBs that had an active IFP region

on the IMS system that has ended. The table is deleted if it was the only IMS system that was servicing the PSB (other than the local IMS system).

In the Fast Path input edit/routing exit routine, DBFHAGU0, you can specify the how the message is to be processed. This is known as the sysplex processing code (SPC), and there are three options:

- Local only
- Local first
- Global only

A design objective should be to process the message on the local IMS system wherever possible. If the message is processed on the local IMS system, no interaction is required with the EMHQ structure. This avoids the overhead for:

- Performing a CQSPUT to the EMHQ structure
- Each IMS system registered as interested in the PSB and LTERM being notified
- The IMS system that is to process the message having to read the message from the EMHQ structure
- Extra logging

6.2.1 Local only

The local only option requires the transaction to be processed only on the local IMS system. The message is never placed on the EMHQ structure. If the routing code is stopped or there are no IFP regions servicing the PSB in the local IMS system, the input is rejected with this message:

```
DFS2533 ROUTING CODE NOT ACTIVE.
```

Otherwise, the transaction is queued to the balancing group.

6.2.2 Local first

The local first option, which is the default, causes IMS to attempt to process the transaction on the local IMS system first, to avoid access to the EMHQ structure. The transaction is always processed locally, if there is at least one IFP region in the local system servicing the PSB that this transaction requires and the number of transactions queued to this balancing group is not greater than five. If more than five transactions are queued to the balancing group that this transaction requires, it is still processed locally if the number of IFP regions servicing this balancing group divided by four is greater than the number of transactions currently queued to this balancing group. If the transaction cannot be processed locally because of the above criteria, it is passed to CQS, which puts the message on the program ready queue.

For the transaction to be accepted, the local routing code must not be stopped, and there must be at least one program servicing this transaction within the shared-queues group, otherwise the input is rejected with this message:

```
DFS2533 ROUTING CODE NOT ACTIVE
```

The check as to whether a routing code is active is performed by a lookup on the local PSB name table and then, if required, the global PSB name table.

6.2.3 Global only

If the global only option is specified and the input transaction is accepted, the message is always placed on the EMHQ structure for processing. For the transaction to be accepted, the local routing code must not be stopped and there must be at least one program servicing this transaction within the shared-queues group, otherwise the input is rejected with this message:

```
DFS2533 ROUTING CODE NOT ACTIVE
```

The check as to whether a routing code is active is performed by a lookup on the local PSB name table and then, if required, the global PSB name table.

The global only option should be selected only for transactions that are not going to run on the front-end IMS system.

6.2.4 Fast Path transaction flow

In this section we describe how a Fast Path transaction is processed. PSBY is defined with an SPC option of Local First. The transaction flow relates to the following four figures: Figure 6-6 through Figure 6-9 on page 135.

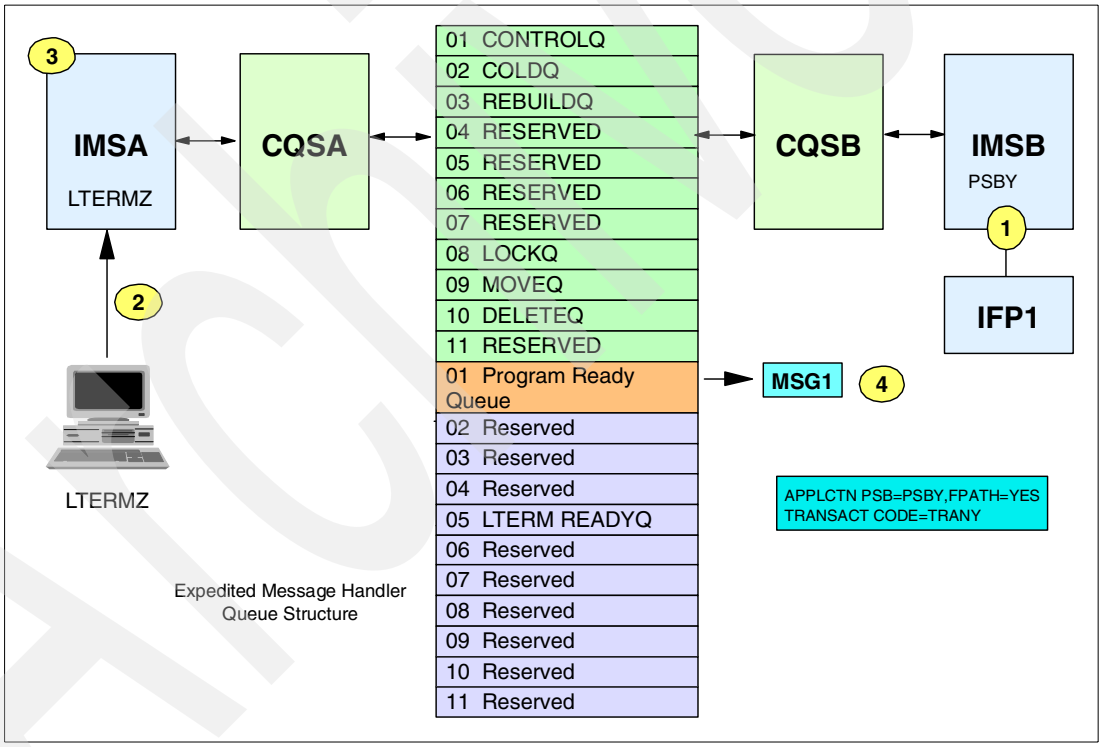


Figure 6-6 Fast Path transaction flow (part 1)

1. IMSB registers interest in PSBY when IFP1 is started. IFP1 is the only region servicing PSBY in the shared-queues group. IMSB creates a local PSB name table for PSBY and notifies IMSA, which causes IMSA to create a global PSB name table for PSBY.
2. IMSA receives MSG1 (TRANY) from LTERMZ.
3. IMSA registers an interest in statically defined LTERMZ when the first Fast Path transaction is put on the EMHQ structure. Registration has already occurred to the MSGQ structure when LTERMZ logged on. Registration to the EMHQ structure occurs only when the first Fast Path transaction is put on the EMHQ structure.

4. IMSA performs a lookup in the local PSB name table to determine whether it can be processed locally. In this case the lookup would not be successful. A lookup is then performed on the global PSB name table. The lookup finds an entry, and IMSA accepts the transaction from the user. IMSA then requests that CQSA put MSG1 on the program ready queue for PSBY.

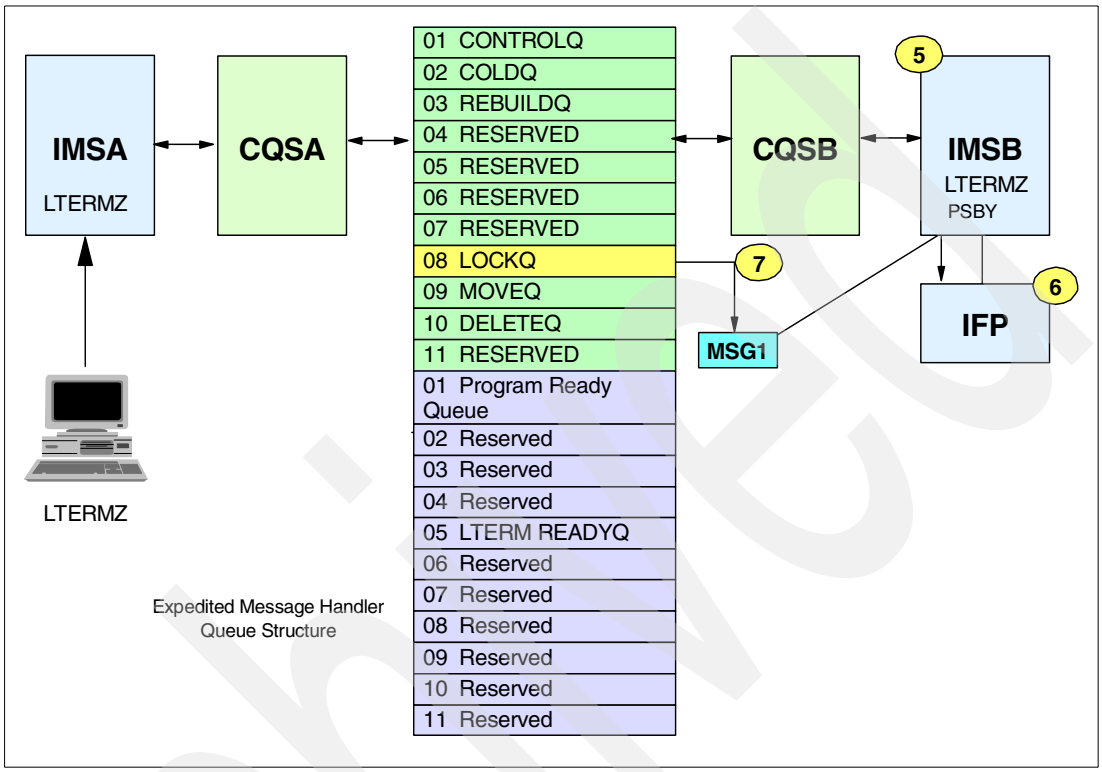


Figure 6-7 Fast Path transaction flow (part 2)

5. IMSB is notified by CQSB that the PSBY queue has gone from empty to non-empty.
6. IMSB reads TRANY from the PSBY queue and passes it to IFP1.
7. TRANY is moved from the program ready queue to the lock queue.

Archived

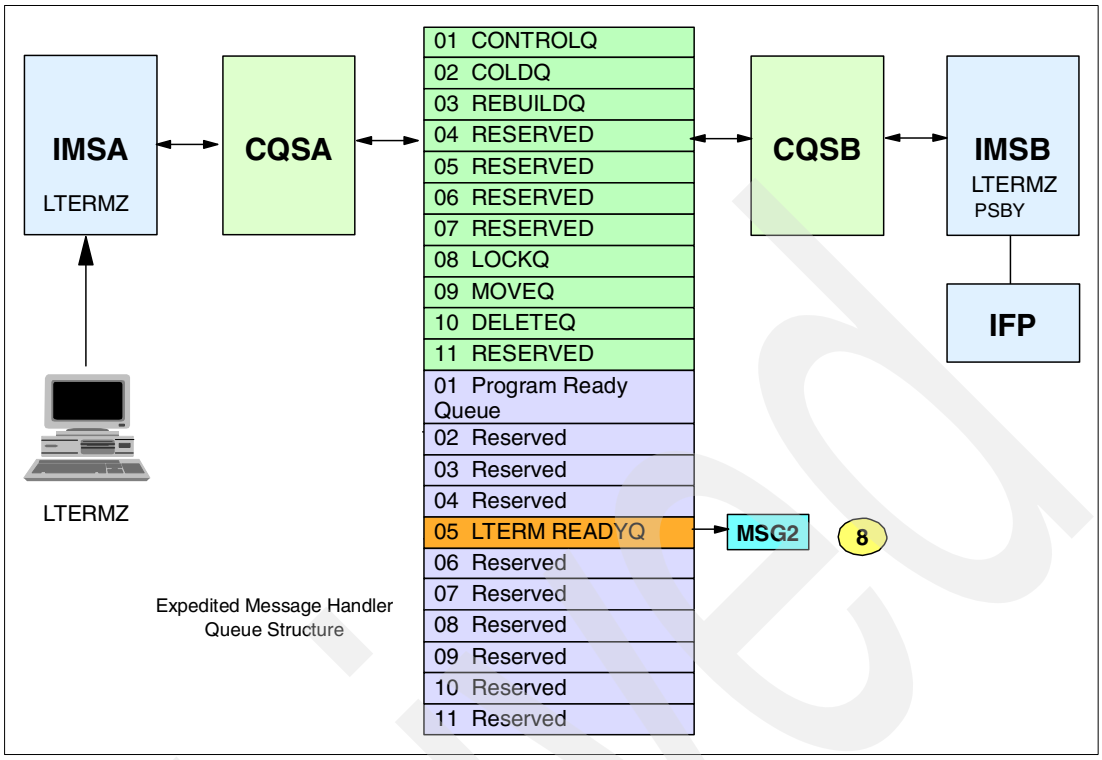


Figure 6-8 Fast Path transaction flow (part 3)

8. IFP1 processes TRANY, inserts a reply (MSG2) for LTERMZ, and commits the work. IMSB then requests that CQSB place the inserted reply message (MSG2) for LTERMZ on the LTERM ready queue and delete MSG1 from the lock queue.

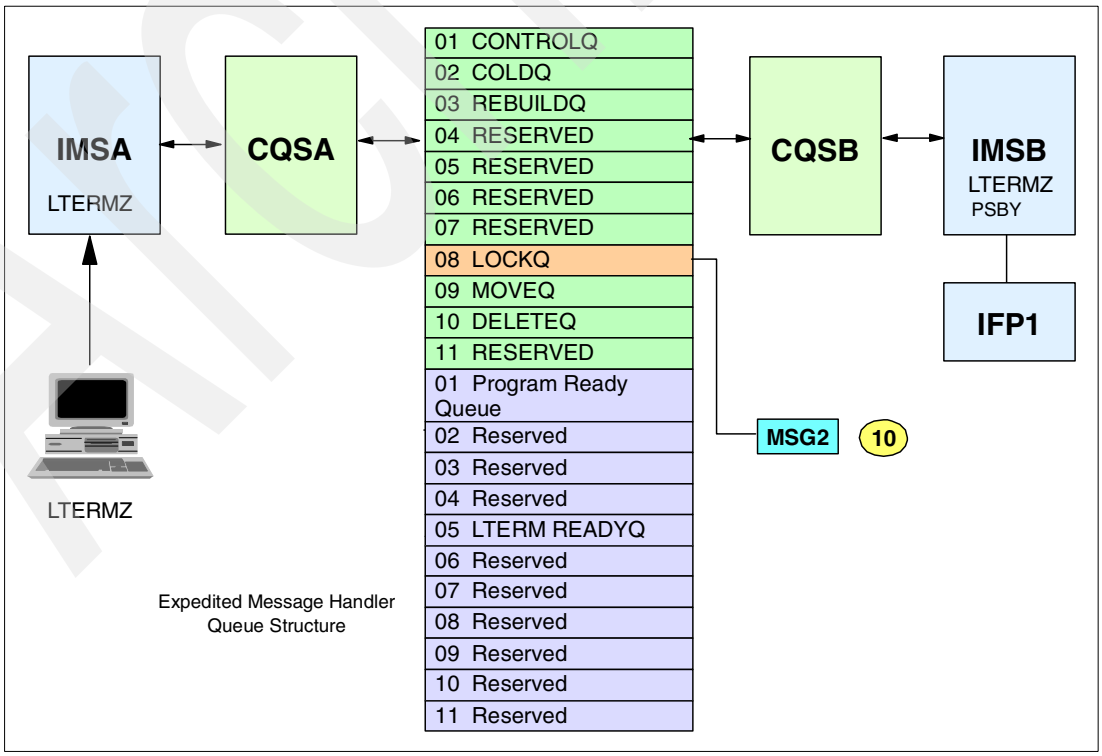


Figure 6-9 Fast Path transaction flow (part 4)

9. The queue for LTERMZ has gone from empty to non-empty. CQSA then notifies IMSA of work for LTERMZ.
10. IMSA passes a read request to CQSA. The message is retrieved. The reading of the message causes the message to be moved from the LTERM ready queue to the lock queue.
11. IMSA returns the reply message to LTERMZ.

    When the terminal has acknowledged receipt of the message, IMSA requests CQSA to delete MSG2 from the structure.

Archived

7

Common Service Layer (CSL) components

The Common Service Layer (CSL) is a major progression in the management and operation of the IMS Parallel Sysplex. This chapter describes the CSL architecture and the components which make it up. The purpose of the CSL is to allow the exchange of information between the components making up the IMSplex. The following is a list of the CSL components:

- IMS control region and DBRC
- Common Queue Server (CQS)
- Structured Call Interface (SCI)
- Operations Manager (OM)
- Resource Manager (RM)

CSL enables new functionality in the areas of operations and resource management and the communications between the components. Here is a list of the first functions utilizing the CSL that were introduced with IMS Version 8:

- Sysplex terminal management (STM)
- Global online change
- Single point of control (SPOC)
- Operation Manager user interface
- Automatic RECON loss notification (ARLN)
- Language Environment® (LE) dynamic run time options

These functions are described later in subsequent chapters.

© Copyright IBM Corp. 2003. All rights reserved.

7.1 IMS architecture overview

The IMS long term architecture was developed over a period of years in the 1980s and 1990s. There are several goals hoped to accomplish by the new architecture:

- Reduce complexity of product code
  - Isolated modular units (MUs) communicating through an architected interface
  - Design / development / test / release independence
  - Improved code quality
  - Less scope of knowledge required to work on IMS code
- Remove capacity and growth constraints
  - Modular units can be added to the sysplex as needed to meet demand
  - Protect OS/390 investment for both IBM and customers by allowing incremental growth as it is needed
- Simplify systems management of IMS
  - Provide a single point of control for all of the pieces of the IMS system
  - Integrate into any IBM-wide operations solution
  - Single place to define IMS resources, operate and tune the system
- Improve IMS availability characteristics
  - Modular units are independent from a failure isolation perspective
  - Spreading MUs throughout the sysplex reduces single-machine failure exposure

7.1.1 Architecture in IMS Version 6 and Version 7

Figure 7-1 shows the IMS architecture as it exists today with IMS Version 6 and Version 7 in a shared queues system. The top layer is the global services layer. This layer comprises many different services that are outside the scope of a single IMS image, or IMSplex. DBRC and IRLM are shown here, because they can be used by more than one IMS image at a time.

Inside of the box labeled "IMS image", we find the IMS control region as we know it today. This includes all of the IMS control address spaces, the DL/I address space, and the dependent regions. Also in this box is the CQS address space. CQS and IMS communicate via a structured interface, known as the CQS SCI (structured call interface).

One final layer shown here is the Base Primitive Environment (BPE). BPE provides the set of system services common to all new IMS address spaces. This includes dispatching, latching, storage management, address space initialization and termination, and many others.

Archived

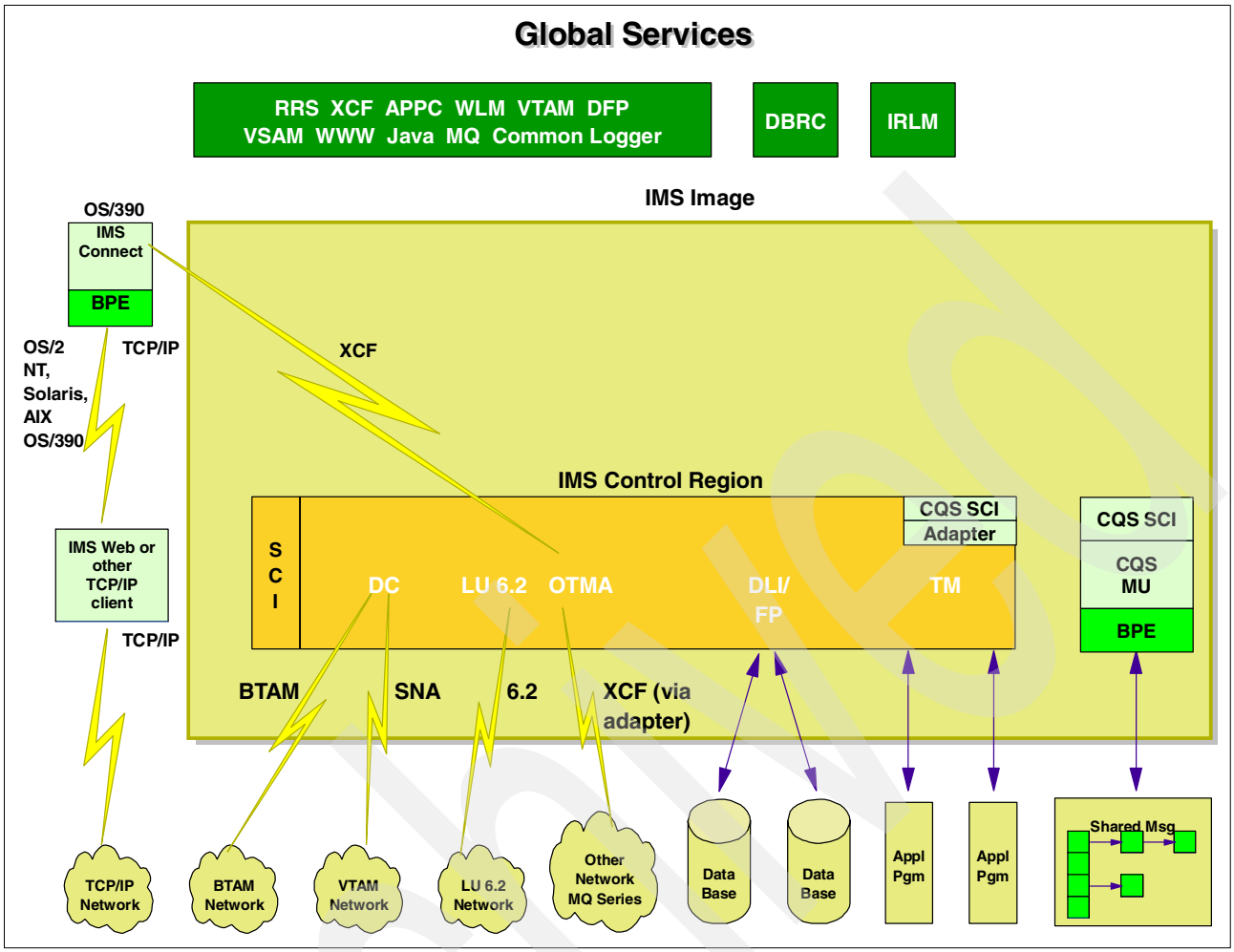


Figure 7-1 IMS architecture in IMS Version 6 and 7

7.1.2 Architecture in IMS Version 8

Figure 7-2 shows the IMS architecture as implemented in IMS Version 8. The big change between IMS Version 7 and IMS Version 8 is the introduction of the beginnings of the CSL, and the communications among IMS, CSL, and DBRC via the SCI.

In IMS Version 8, there are three new address spaces: Operation Manager (OM), Resource Manager (RM), and Structured Call Interface (SCI). Note that SCI is used to mean both the SCI address space, as well as the interface between address spaces.

All IMS address spaces register with SCI, so that they can be a part of the communications flowing around the IMSplex. In IMS Version 8, CQS still uses its own private SCI (the "CQS SCI") for performance reasons and to minimize the code change required. DBRC moves into the IMSplex scope by storing the IMSplex ID in the RECON data set. This ties the RECONs to one particular IMSplex.

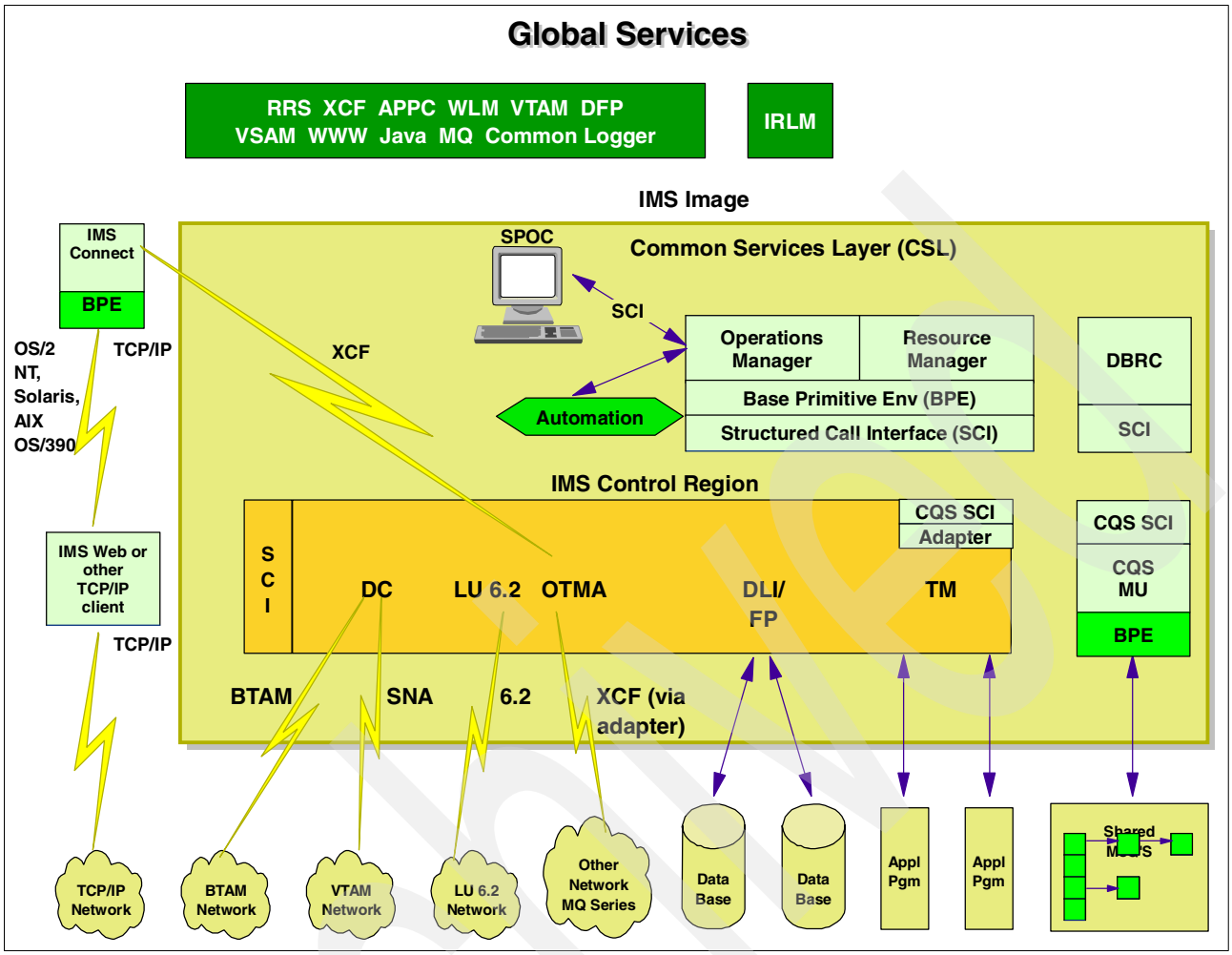


Figure 7-2 IMS architecture in an IMS Version 8 environment

7.2 Common Service Layer (CSL) architecture

The Common Service Layer is an evolving architecture for the IMSplex. It is an architecture, not an address space. The IMSplex consists of the three new CSL address spaces built on the Base Primitive Environment (BPE), plus existing address spaces.

7.2.1 CSL address spaces

In IMS Version 8, there are three new IMS CSL address spaces:

- Structured Call Interface (SCI)
- Operations Manager (OM)
- Resource Manager (RM)

The Common Service Layer is a set of address spaces which provide system management functions to one or more IMS systems within an IMSplex.

In IMS Version 8, the systems management functions implemented by the CSL are:

- Operations management

  The ability to control or operate an IMSplex from a single location (a SPOC).

- Resource management

  The ability to keep global name and status information about certain IMS resources (in Version 8, transactions, LTERMs, and users).

- Inter-address space communication

  The ability to send messages and requests to other IMSplex components, and to receive state change information about these components.

The Common Service Layer is one of the four layers in the long term IMS architecture, and is the basis for future IMSplex enhancements.

The intra-IMSplex communication between components are handled by the Structured Call Interface. For the resource management, a new resource structure can be defined in the Coupling Facility. Resource Manager uses CQS to manage the resource structure.

7.2.2 CSL servers and clients

The new CSL address spaces are *servers* that perform services for clients. This is equivalent to CQS performing message queuing services for IMS. A typical characteristic for these servers is that they provide specific functions; they are not sensitive to the data that is sent from their clients.

The users of CSL address spaces are *clients.* The most common clients are:

- IMS Version 8 control regions
- IMS-provided TSO/ISPF based single point of control (SPOC)
- DBRC

Because the interface to CSL is open, there are also other potential CSL clients. Vendors may write complementary products that use CSL services and IMS users may write local functions such as automation programs which utilize CSL services.

7.2.3 CSL configuration

Figure 7-3 describes the CSL components and shows their relationship to each other.

Archive

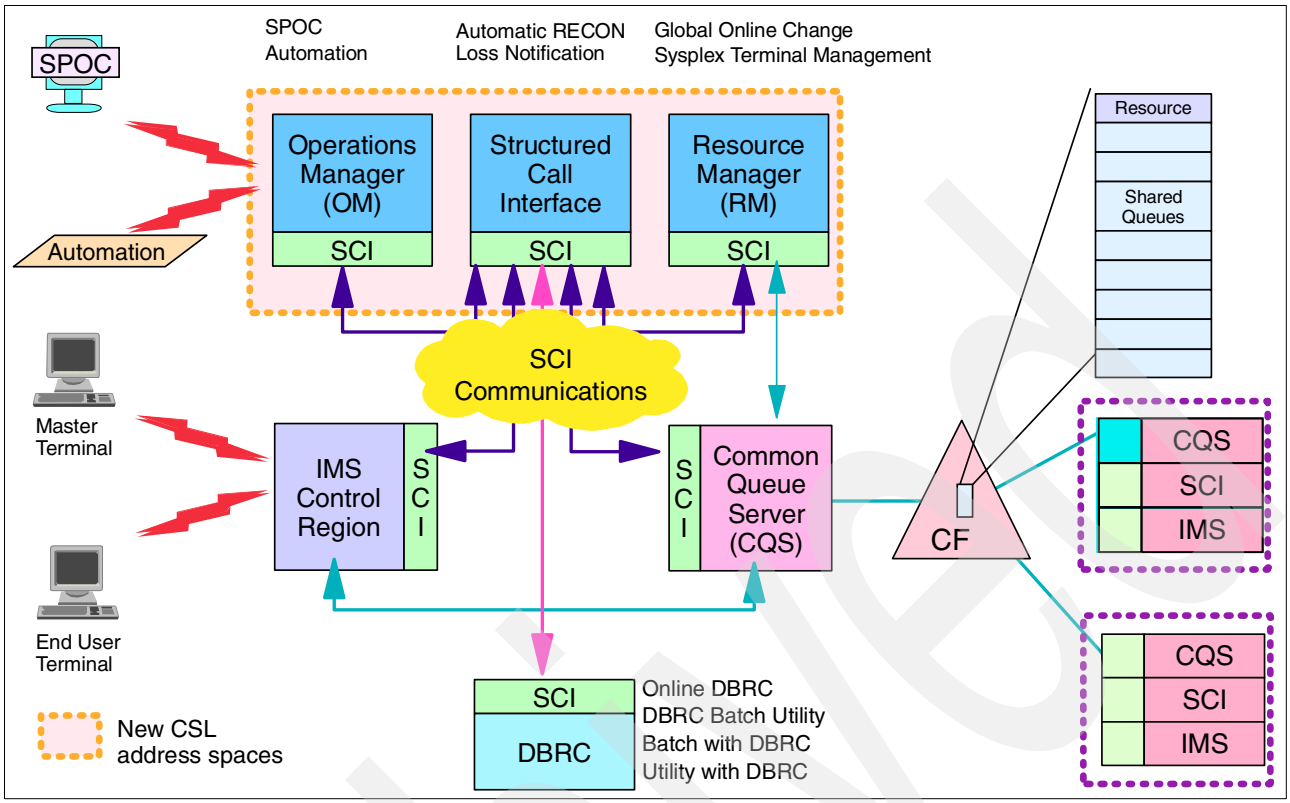


Figure 7-3 Common Service Layer components

The SCI provides the mechanism for the inter-address space communication, and the address spaces use the SCI to communicate as required between the components (clients and servers). One SCI address space is required per OS/390 or z/OS image which has one or more IMSs taking part in the IMSplex. Only one OM address space and one RM address space are required per IMSplex, because the SCI provides the communication between the operating system images. However, for performance and availability reasons, we recommend having one OM and one RM per OS/390 or z/OS image.

7.2.4 IMSplex configuration

An IMSplex is a collection of one or more IMS address spaces that work together as a unit. These IMSs typically share databases, resources, or message queues, or any combination of these. Usually, an IMSplex runs in a S/390 or z/OS Parallel Sysplex environment. An IMSplex must include an IMS Common Service Layer.

The following address spaces can participate in an IMSplex:

- The control region (DLI/SAS and dependent regions are included as part of the control region in this sense)
- IMS manager address spaces (OM, RM, and SCI)
- IMS server address spaces (CQS and DBRC)
- Any vendor- or customer-written address spaces that register with SCI. SCI has provisions for registering members with a type of *OTHER*, to allow for vendor and customer participation in an IMSplex.

These address spaces are referred to generically as IMSplex components. An IMSplex is made up of all IMS component address spaces that share the same IMSplex name. The IMSplex name is usually specified in a PROCLIB member for each address space.

Figure 7-4 depicts an IMSplex running on a four operating systems image sysplex, each with the CSL address spaces (SCI, OM, RM), a CQS address space, data sharing, and with VTAM Generic Resources.

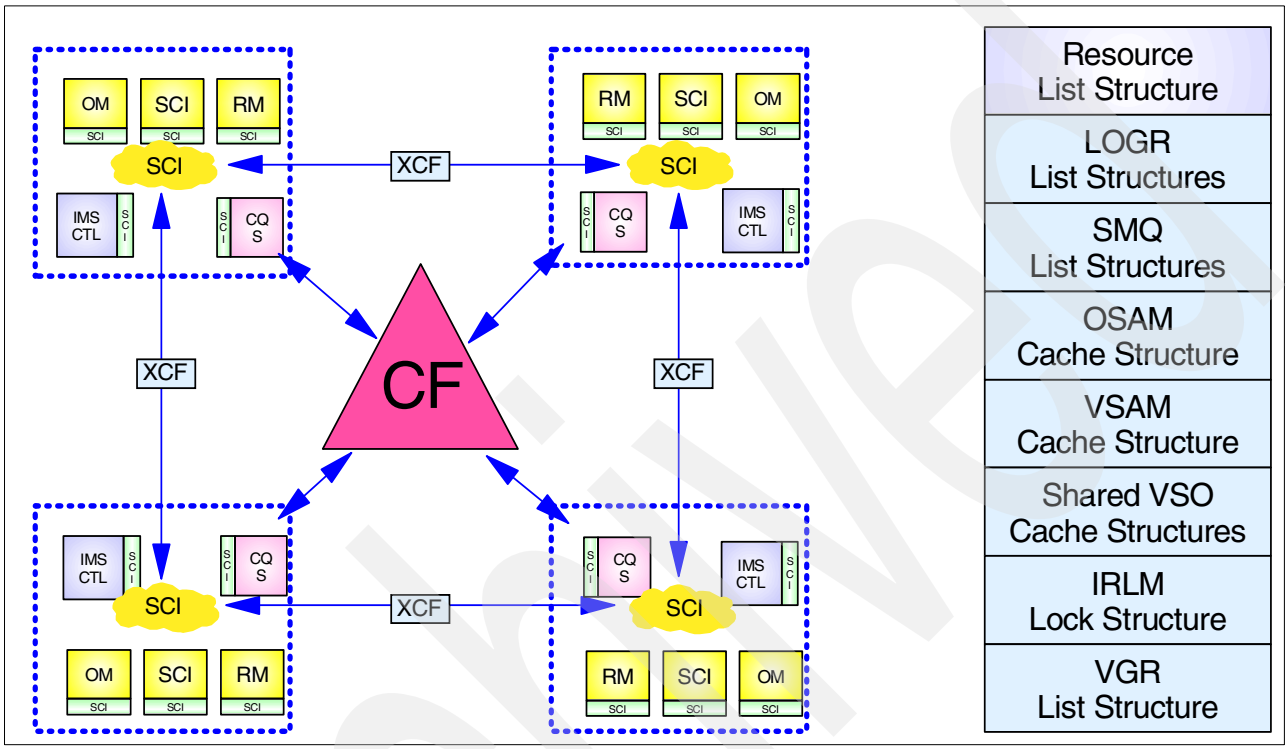


Figure 7-4 IMS Version 8 IMSplex configuration

7.2.5 IMSplex environmental factors

Some environmental considerations for IMSplex:

- If the IMSs in the IMSplex are sharing databases, then they must also share the same RECONs, and there can only be one set of RECONs.
- If the IMSs in the IMSplex are sharing message queues, then all of the IMSs must share the same CQS structures: one structure for full-function message queues, and one optional structure for EMH queues.
- Neither IMS nor the CSL enforce all IMSplex IMSs sharing the same message queues. It is possible to have some IMSs share the same queues and others not, but this is not recommended, and may lead to unexpected results for some IMSplex commands.
- Also note that IMSplex supports the minimal case, a single non-sharing IMS system with a CSL.

7.2.6 IMS Version 8 in a sysplex

An IMSplex is a collection of IMS modular units (MUs) that cooperate in a sysplex environment to process workloads. The IMS systems in the sysplex are sharing databases and/or sharing queues.

The Common Service Layer (CSL) is optional in IMS Version 8 and it is added to provide systems management in an IMSplex. The CSL consists of the new address spaces Operations Manager (OM), Resource Manager (RM), and Structured Call Interface (SCI):

- SCI provides communication between different IMS MUs in an IMSplex. SCI must reside on every system in the IMSplex.
- OM controls the operations of an IMSplex.
- RM provides the infrastructure for managing global resource information and IMSplex-wide processes in an IMSplex.

The CQS interface was built as a specific implementation of the SCI interface. The SCI extends the architecture to a more generic model. The existing CQS interface is not changed to use SCI. The resource structure is a new Coupling Facility list structure managed by CQS, used by RM to contain global resource information.

Single point of control, global online change, and sysplex terminal management are new functions provided by CSL. These new functions require the full CSL environment, that is in addition to the SCI, at least one OM and one RM is required in the IMSplex. Putting OM and RM on every system improves performance and reliability. Sysplex terminal management also ***requires*** a resource structure. A TSO/ISPF single point of control (SPOC) is provided as an interface to OM.

Global online change can be implemented without a resource structure, but to have consistency checking of the online change ACBLIB, FORMAT, MODBLKS, and MATRIX data sets, including all concatenations, the resource structure is needed. Otherwise you must ensure consistency of the data sets involved.

Another new function, automatic RECON loss notification (ARLN) requires at a minimum that there is an SCI on each OS/390 image and that each IMS DBRC or utility or batch job using the RECONs has specified the same “IMSPLEX=” value, either via an execution parameter or a user exit. If all you want is ARLN, then only SCI is required. If you want other new functions, then you need both OM and RM.

Archive

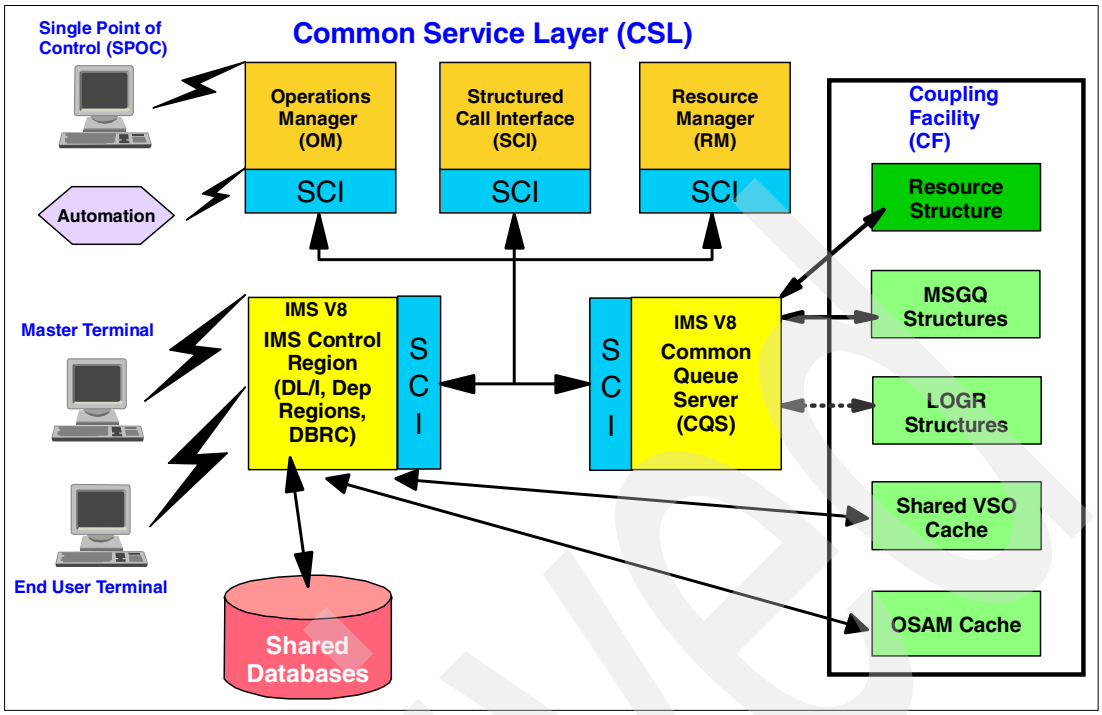


Figure 7-5 IMS Version 8 in Parallel Sysplex

7.3 Base Primitive Environment (BPE)

Base Primitive Environment (BPE) is a programming environment for developing new IMS component address spaces. It is a common system service base, upon which many other IMS components are built. Common Queue Server (CQS) is an example of this, and more and more IMS components are written to utilize the services provided by the BPE. A version of BPE has been shipped with IMS since IMS 5.1. For IMS Version 8, the BPE version shipped and required is 1.4. BPE 1.4 supports the Common Queue Server (CQS) and the new Common Service Layer (CSL) components.

BPE provides the following environment configuration and base system services to functional components:

- TCB structure and control block definitions
- Sub-dispatching
- Storage management
- Tracing and trace table definitions
- Optional user exits

The following IMS components use BPE:

- Common Queue Server (CQS)
- Operations Manager (OM)
- Resource Manager (RM)
- Structured Call Interface (SCI)

When an IMS component that uses BPE is started, the component loads a copy of the BPE service modules into its address space from the IMS program libraries.

The IMS component's modules are specific to that component; however, the BPE service modules are common across the various address spaces. The base system service functions are therefore identical in every address space that uses BPE.

7.4 Structured Call Interface (SCI)

Structured Call Interface (SCI), is an IMS address space that provides communications between IMS components that work together in a sysplex. The Structured Call Interface address space provides:

- Standardized intra-IMSplex communications between members of an IMSplex
- Security authorization for IMSplex membership
- SCI services to registered members

SCI allows one IMSplex component to communicate to another IMSplex component or components, without the components needing to know where the other components are running. SCI routes the communications locally within the same OS/390, and globally around the sysplex for components on different OS/390s. The sender and receiver of these communications do not need to know where the other partner exists; the routing is all handled by SCI.

The structured call interface services are used by SCI clients to register and deregister as members of the IMSplex, and to communicate with other members. The SCI client issues CSL macros to execute code in the SCI address space using cross memory services under its own TCB and an SRB is scheduled in either the SCI address space or the destination member's address space.

Figure 7-6 shows that SCI is involved with every member. Each individual member has a SCI interface which it invokes to register or communicate with other members. When communications are with a member on another OS/390 image, then XCF signalling services are used to send the message.

SCI is required for any IMS address spaces to communicate with each other with a few exceptions. The interface between IMS and CQS was developed for IMS Version 6 shared queues and is also used in Version 8, even when CQS is used for access to the resource structure. SCI is not used for communications between IMS and CQS or between the Resource Manager and CQS. The other exception is the communication between IMS and DBRC, where the SCI is not used.

Archived

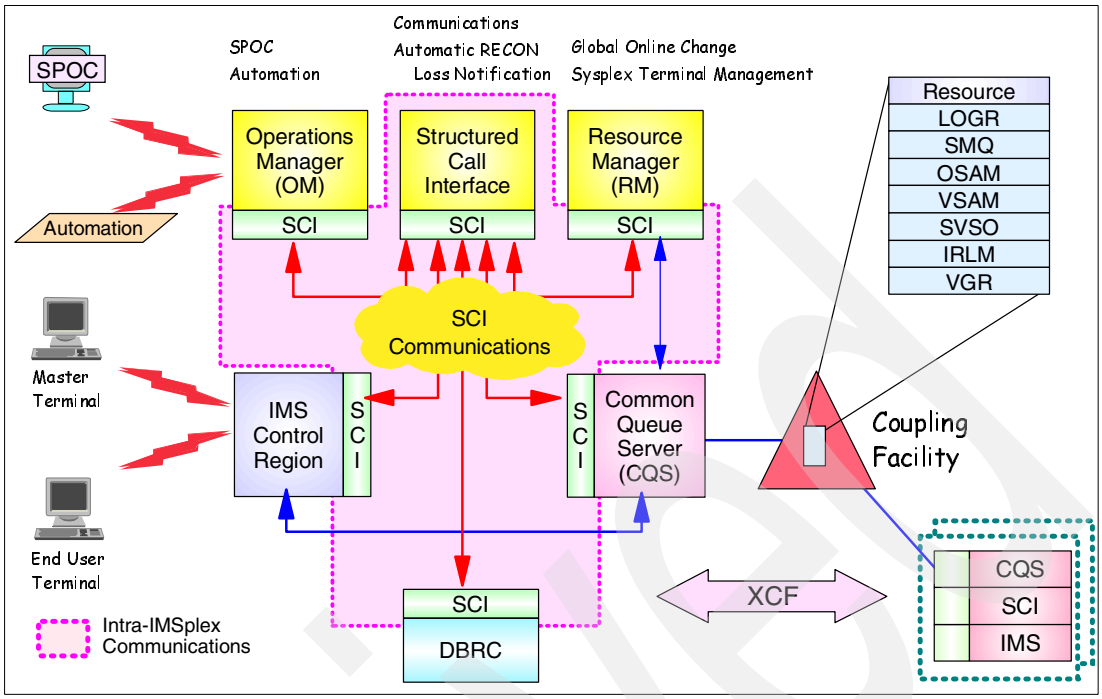


Figure 7-6 SCI infrastructure

7.4.1 SCI components

The SCI configuration is that one SCI address space is required on each OS/390 or z/OS image with IMSplex members.

The other address spaces in the IMSplex register with the SCI. The following components all register and interact with SCI.

- CSL address spaces — Operations manager (OM) and Resource Manager (RM)
- Common Queue Server (CQS)
- IMS - DB/DC, DBCTL, DCCTL, FDBR
- Automated Operator Programs (AOP) such as single point of control (SPOC)
- DBRC
- Batch with DBRC
- Others — The CSL (SCI) interface is documented, and may be accessed by vendor or user programs.

Note: Registrants may abend if SCI is not available when required. The individual registering member makes the decision on what to do with the 'No SCI' reason code. Different IMS components handle it differently. The TSO SPOC just displays the return and reason code, IMS issues a WTOR (DFS3306A) asking if the user wants to retry or cancel. If you reply CANCEL, IMS abends with a U3309. OM, RM, and CQS get a CSL0020I message indicating SCI is not ready, but then they retry six times at ten second intervals and then abend with a U0010.

7.4.2 SCI user exit routines

SCI user exits allow you to customize and monitor the SCI environment. All these exits are optional and they are written and supplied by the user. No sample exits are provided, but the provided exit points for SCI are the following:

- SCI client connection user exit
- SCI initialization/termination user exit
- BPE statistics user exit

SCI uses BPE services to call and manage its user exits. Since CSL components are built on the Base Primitive Environment, they can take advantage of BPE common services and interfaces. This aids in creating a common look and feel to the configuration of new IMS components. This is one of the areas where the three new address spaces (SCI, OM and RM) are the same.

BPE also provides a common user exit runtime environment for all user exits. This environment includes a standard user exit parameter list, callable services, static and dynamic work areas for the exits, and a recovery environment for user exit abends.

BPE enables you to externally specify the user exit modules to be called for a particular user exit type by using EXITDEF= statements in the BPE user exit list PROCLIB members. User exits are defined in the BPE user exit list PROCLIB member for SCI as shown in Example 7-1.

Example 7-1 Exit definition for SCI

```
EXITDEF=(TYPE=xxx,EXITS=(XXX,YYY),COMP=SCI)
```

Detailed information about these exits can be found in *IMS Version 8: Common Service Layer Guide and Reference,* SC27-1293, and *IMS Version 8: Base Primitive Environment Guide and Reference,* SC27-1290.

SCI client connection user exit

The SCI client connection user exit is called when a client connects (registers) or disconnects (deregisters) from SCI. It is also called when a client issues the CSLSCRDY (ready) and the CSLSCQSC (quiesce) requests.

This exit is called for the following events:

- After a client has successfully connected to SCI
- After a client has successfully completed the Ready request to SCI
- After a client has successfully completed the Quiesce request to SCI
- After a client has successfully disconnected normally or abnormally from SCI

This exit is defined as TYPE=CLNTCONN in the EXITDEF statement in the BPE user exit list PROCLIB member.

SCI initialization/termination user exit

The SCI initialization/termination user exit is called during SCI address space initialization, IMSplex initialization, SCI address space normal termination, or IMSplex normal termination. This exit is not called during SCI address space abnormal termination or IMSplex abnormal termination.

This exit is called for the following events:

- After SCI has completed initialization
- After each IMSplex has initialized
- When SCI is terminating normally
- When an IMSplex is terminating normally

This exit is defined as TYPE=INITTERM in the EXITDEF statement in the BPE user exit list PROCLIB member.

SCI statistics available through BPE statistics user exit

The BPE statistics user exit routine enables you, at regular intervals, to gather statistics related to an IMS component that is running with a BPE address space.

This exit is also called for the following events:

- First called soon after BPE initialization completes
- Also called a final time during normal shutdown of the address space

The exit is The BPE Statistics user exit routine is defined as TYPE=STATS,COMP=BPE in the EXITDEF statement in the BPE user exit PROCLIB member.

The interval between successive statistics exit calls is specified on the STATINTV parameter in the BPE configuration PROCLIB member.

7.4.3 SCI IMSplex member exit routines

The SCI IMSplex member exit routines allow an IMSplex member to:

- Monitor what address spaces are active members of the IMSplex
- Receive messages and requests from other members of the same IMSplex

SCI member exits are supplied by the IMSplex member address space (such as the IMS control region) and they are driven in the member address space. The user must write its own exit routines for each member tailored to the needs of that member product, to be supplied as part of the product. The exit points provided are the following:

- SCI input exit
- SCI notify client exit

SCI input exit routine

The SCI input exit routine is called whenever there is a message or a request for the IMSplex member.

The IMSplex member loads the exit routine and passes the exit routine address on the CSLSCREG request. The exit is driven in the member's address space, either as an SRB (for authorized members) or as an IRB (for non-authorized members).

SCI notify client exit routine

The SCI notify client exit routine is driven whenever there is a change in the SCI status of an IMSplex member. This allows a member to keep track of the status of other members in the IMSplex.

The IMSplex member loads the exit routine and passes the exit routine address on the CSLSCREG request. The exit is driven in the member's address space, either as an SRB (for authorized members) or as an IRB (for non-authorized members).

The exit is driven whenever an IMSplex member:

- Completes a successful CSLSCREG FUNC=REGISTER
- Completes a successful CSLSCRDY FUNC=READY
- Completes a successful CSLSCQSC FUNC=QUIESCE
- Completes a successful CSLSCDRG FUNC=DEREGISTER
- Terminates without issuing a CSLSCDRG FUNC=DEREGISTER request
- Is not reachable because the local SCI is not active

Archived

7.5 Operations Manager (OM)

Operations Manager (OM) is an IMS address space that provides command processing and a single point of control for command entry in a sysplex. The introduction of OM in IMS Version 8 is the first step of introducing a single point of control and operator interaction and management of all or selected members of the IMSplex.

Prior IMS Version 8 and operations manager, sysplex operations management does not provide an IMS single image. Commands can be routed to multiple IMS systems using the E-MCS console function that requires a common command recognition character (CRC) defined in IMS.PROCLIB member DFSPBxxx.

Also, to direct a command to a specific IMS system you need to know the OS/390 system name and the IMS name. The CMD and ICMD DL/I calls of the Automated Operator Interface only affect the IMS system they are using. They cannot manage other IMS systems. RACF can be used to secure commands entered from MCS/E-MCS but DFSCCMD0 is needed to secure command at the verb, keyword, and resource levels.

Commands may be entered by each MTO but most commands and automation processes today can only affect an individual IMS. Some commands have a GLOBAL parameter, but asynchronous responses (for example, DFS0488I message) from other IMSs are not returned to the MTO entering command. See Example 7-2.

Example 7-2 Command with both synchronous and asynchronous responses

```
/DBR DATABASE XYZ GLOBAL
DFS058I DBR COMMAND IN PROGRESS
DFS0488I DBR COMMAND COMPLETED DBN XYZ  RC=nn
```

Figure 7-7 gives an example of pre-IMS Version 8 IMSplex command activity.

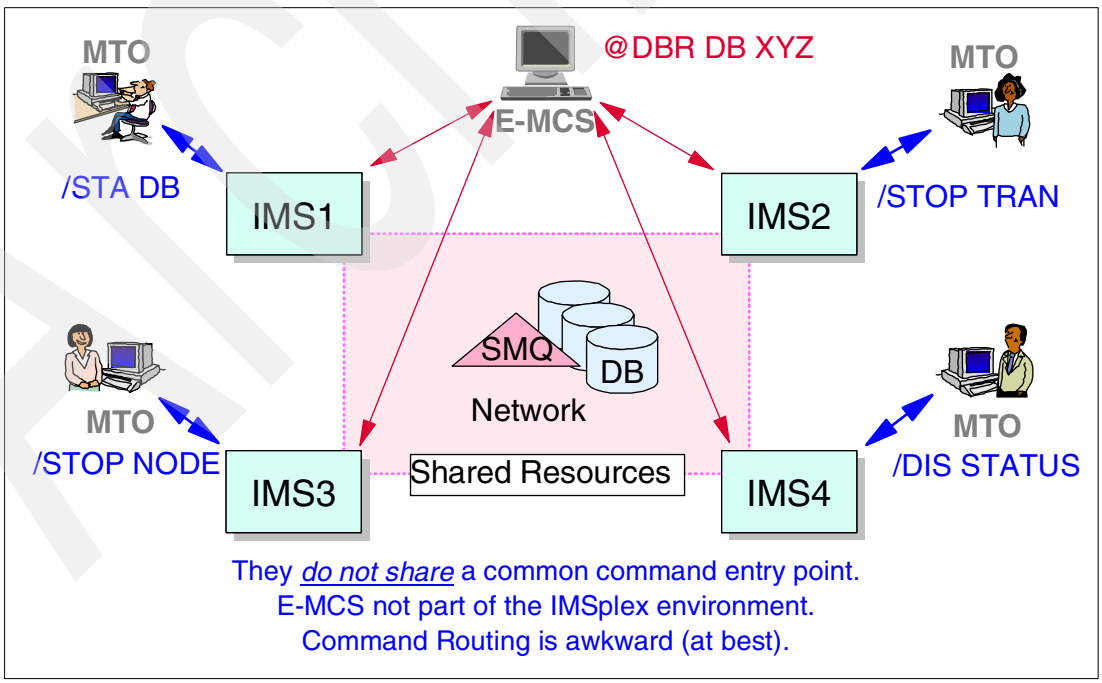


Figure 7-7 Pre Version 8 IMSplex command activity

7.5.1 IMSplex components related to OM

Several of the IMSplex components shown earlier are applicable to OM. The IMSplex components related to OM include the other CSL address spaces (RM and SCI), OM clients (IMS subsystems, TSO SPOC):

- OM address space
  - At least one OM required in IMSplex defined with CSL
  - one OM on each OS system best for performance
- Common Service Layer (CSL) address spaces
  - At least one Resource Manager (RM) required in IMSplex
  - Structured Call Interface (SCI) required on each OS system
- OM clients (TSO SPOC and IMS subsystems)
  - The TSO SPOC is an AOP (Automated Operator Program) client
  - Each IMS DB/DC, DCCTL, or DBCTL subsystem is a command processing client

7.5.2 OM infrastructure

The Operation Manager (OM) is a server address space in the IMSplex. It act as a common point of command entry into the IMSplex and has the following characteristics:

- Runs in a separate address space
- Communicates with other IMS address spaces via the SCI
- Uses the same address space design concepts as RM and SCI
  - Design concepts shared by OM, RM, and SCI where applicable
  - CSL address spaces have same look and feel
  - Purpose of sharing design concepts is to make CSL address spaces easier to understand, enhance, and service
- Supports an architected interface (OM requests) for clients
- Supports user exits to help user manage OM
- Uses Base Primitive Environment (BPE) for system services
- Supports one IMSplex, with plans to support multiple IMSplexes in a future release

7.5.3 OM services

The OM utilizes BPE services and provides an API allowing single point of command entry into the IMSplex. OM is a focal point for operations management and automation, and consolidates command responses from multiple IMSs.

The following services are provided to members and clients of an IMSplex:

- Routes commands to IMSplex members registered for the command
  - Does not understand specific IMS commands or resources
- Consolidates command responses from individual IMSplex members into a single response to present to the command originator
- Provides an API for IMS command automation
- Provides a general use interface for command registration to support any command processing client

Archived

- Provides user exit capability for command and response editing, and for command security
- Supports existing IMS commands (classic commands) and new IMSplex commands
- Supports a single point of control (SPOC) to present a single system image for the IMSplex by allowing the user to enter commands to IMSs in the IMSplex from a single console. IMS will provide the following SPOC applications:
  - 3270 TSO/ISPF application running on S/390
  - Workstation application
- Available for use by any IMS Version 8 system
  - IMS systems must register commands with OM. When a command is entered through OM API, it is sent to interested systems based on requested routing.
  - IMS Version 7 and prior systems will not be enhanced to communicate with OM.
- Does not support command recovery
  - Does not write to log, data sets or Coupling Facility
  - OM does not support restart
  - If OM fails, application is notified of failure but command response output is not returned
  - If command is processed successfully by IMS, IMS command recovery rules apply

The CSL configuration requires one OM address space per IMSplex, but one per OS/390 or z/OS image is recommended for performance and availability.

Figure 7-8 shows an IMS Version 8 CSL environment which has the ability to issue IMSplex wide commands and receive their responses from one place via SPOC.

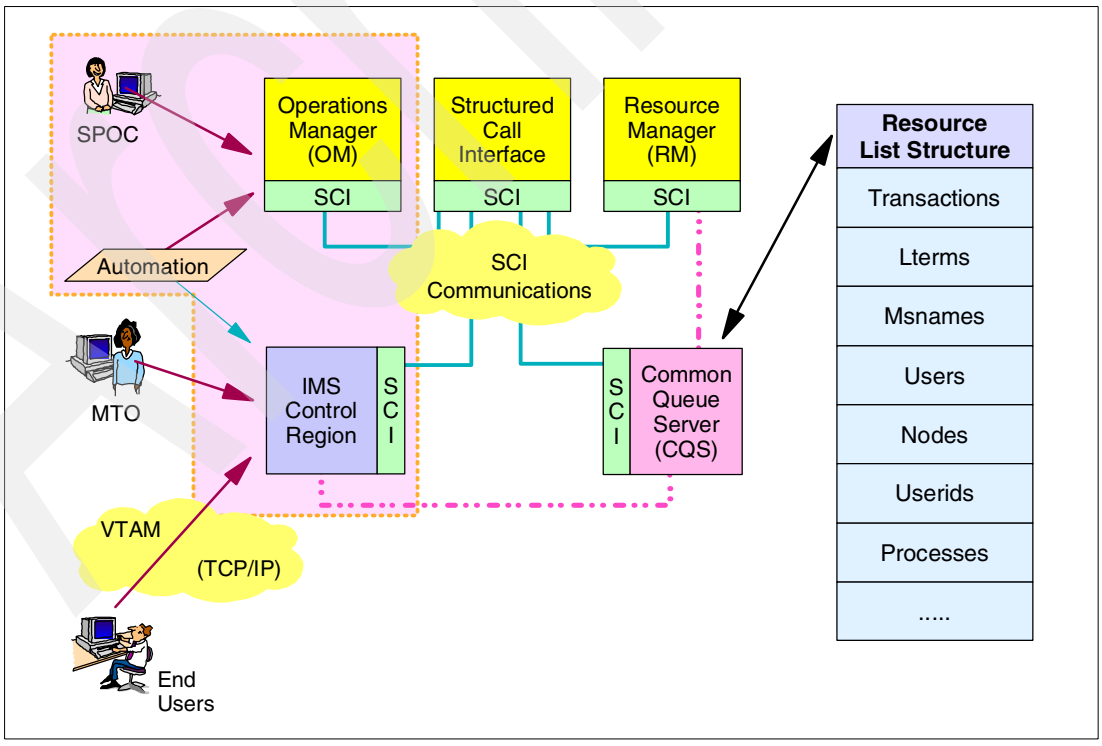


Figure 7-8 CSL with OM and SPOC issuing IMSplex wide commands

Figure 7-9 shows a much larger IMSplex with four z/OS images in the sysplex and one IMS on each of these OS/390s. OM is routing commands and consolidating responses centrally throughout the IMSplex.

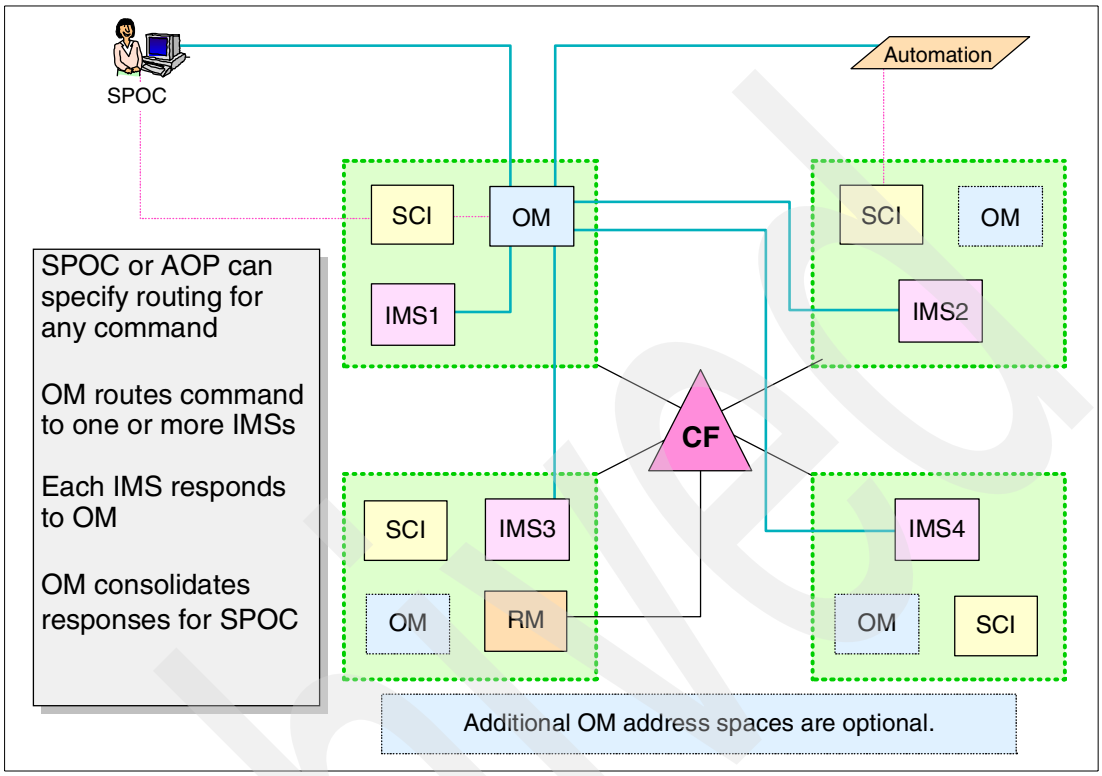


Figure 7-9 OM routing commands in a four member IMSplex

7.5.4 OM command entry routing and response consolidation

OM provides the ability to route commands to IMSplex members and consolidate responses from the members registered for the command. This can be done to all members of the IMSplex, or selected members. The OM API supports the routing of commands via the ROUTE parameter on the various macros. The SPOC application handles creation of the routing list from user input, or from the default preferences.

OM consolidates the command responses from the individual IMSplex members into a single response which is presented to the command originator. You can also specify a time-out value for the response which represents the maximum amount of time you want to wait for the response before the request times out.

7.5.5 OM command security

OM command security is optionally performed during command processing. Command security allows:

- The user to control which user IDs can enter IMS commands through OM
- The user ID to be associated with an application program address space
- The user ID to be the end user logged onto TSO SPOC

Authorization is performed within OM before sending the commands to IMS using RACF (or equivalent security product). There is also the option of calling a user written exit to perform command security.

Command authorization for a command entered through the OM interface can be checked by OM, by IMS, or by both. In the OM initialization PROCLIB member (DFSOIxxx), the CMDSEC= parameter tells OM whether to check command authorization, and if so, how. The choices are the same as for:

- Use RACF (or equivalent security product)
- Use a user-written security exit
- Use both RACF and the exit
- Use neither RACF nor exit

If the exit is to be used, then the exit must be defined in the BPE User Exit List PROCLIB member pointed to in the BPECFG PROCLIB member. The user exit list entry should identify the type of exit as TYPE=SECURITY, and any name the user wants (for example, OMSCTYX0).

If RACF is to be used, then the commands should be defined to RACF. The QRY command requires READ access, and the others require UPDATE access. In the new IMS PROCLIB member DFSCGxxx, used when IMS is running with CSL, the CMDSEC parameter identifies how IMS should handle security for commands entered through the OM interface. Presumable, if OM is performing security authorization, it can be skipped in IMS (CMDSEC=N).

Recommendation: Use OM command security rather than IMS command security.

RACF definitions

Command security in RACF is defined in the OPERCMDS class. The format of the resource is:

```
IMS.imsplexname.command-verb.resource-type
```

Note: This is one level deeper than command security in IMS, which only authorizes commands to the command verb level.

7.5.6 OM APIs

Commands may be entered on an IMS terminal, such as the MTO terminal or the end-user terminal, using the existing command format. Commands may also be entered through the SPOC and OM API. If the command is entered through the OM API, the command can be routed to all IMSs that can process the command.

Figure 7-10 shows how a command comes in from a SPOC/AOP to OM, is routed to interested IMSs which then send their output back to OM in a consolidated format. OM then sends the output back to the originator of the request.

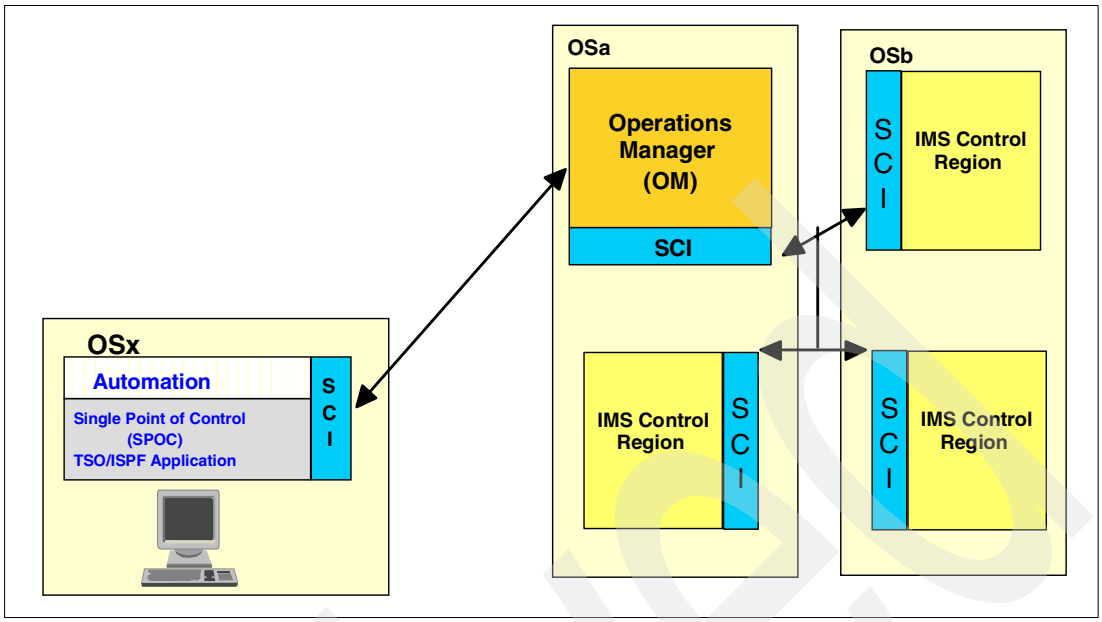


Figure 7-10 IMSplex operations in IMS Version 8

OM provides two types of API support for Automated Operator Program (AOPs).

- CSLOMI API

  The CSLOMI API is used to support command string built by workstations. IMS Connect could use CSLOMI. Command strings are passed to OM and command responses are returned to the client in XML format.

- CSLOMCMD API

  The CSLOMCMD API is used by IMS TSO/SPOC. Command keywords are passed to OM and command response are again returned in XML format.

OM command processing support

OM does not distribute unsolicited output messages to an AOP. You can use NetView® or another automation product to trap IMS messages and then send them to an automation program (communication external to IMS). The automation program could then issue a command on behalf of the event. Also, OM does call the output user exit routine passing the unsolicited message. The user exit routine can then provide its own technique to distribute the message. For example, an OS/390 write to operator (WTO) macro could be used to write the message to the system console. OM does not support command recovery, except in cases that will be changed for RM processing (discussed later in 7.6, “Resource Manager (RM)” on page 163), nor does OM support restart.

7.5.7 OM clients

The OM interface supports several types of clients and their specific interface into the IMSplex for command processing. Examples of OM clients are:

- IMS control region
  - Registers commands it can process if received by OM from another client
  - Receives commands from single point of control (SPOC) or user automation program (AOP)
  - Sends responses through OM to SPOC or AOP

- SPOC and automation programs
  - Send commands to one or more members of IMSplex
  - Receive consolidated responses
  - TSO SPOC provided with IMS Version 8
- Resource Manager
  - Registers with Operations Manager for new OM QUERY command:

    ```
    QUERY STRUCTURE NAME(...) SHOW(...)
    ```
- Vendor (and user) provided clients

All OM services are invoked by CSLOMxxx macros. OM macro coding and use is described in *IMS Version 8: Common Service Layer Guide and Reference,* SC27-1293. The following paragraphs presents the command processing client types, their roles, and the interface used to interact with OM.

7.5.8 Command processing (CP) clients

These are clients which process commands entered from other address spaces. IMS and RM are command processing clients.

Figure 7-11 shows where a CP client (IMS) fits in the IMSplex. IMS must register with SCI to become a member of the IMSplex and be eligible to receive commands from OM. OM must also register with SCI. IMS then registers commands with OM. These are commands which IMS is willing to accept from AO clients.

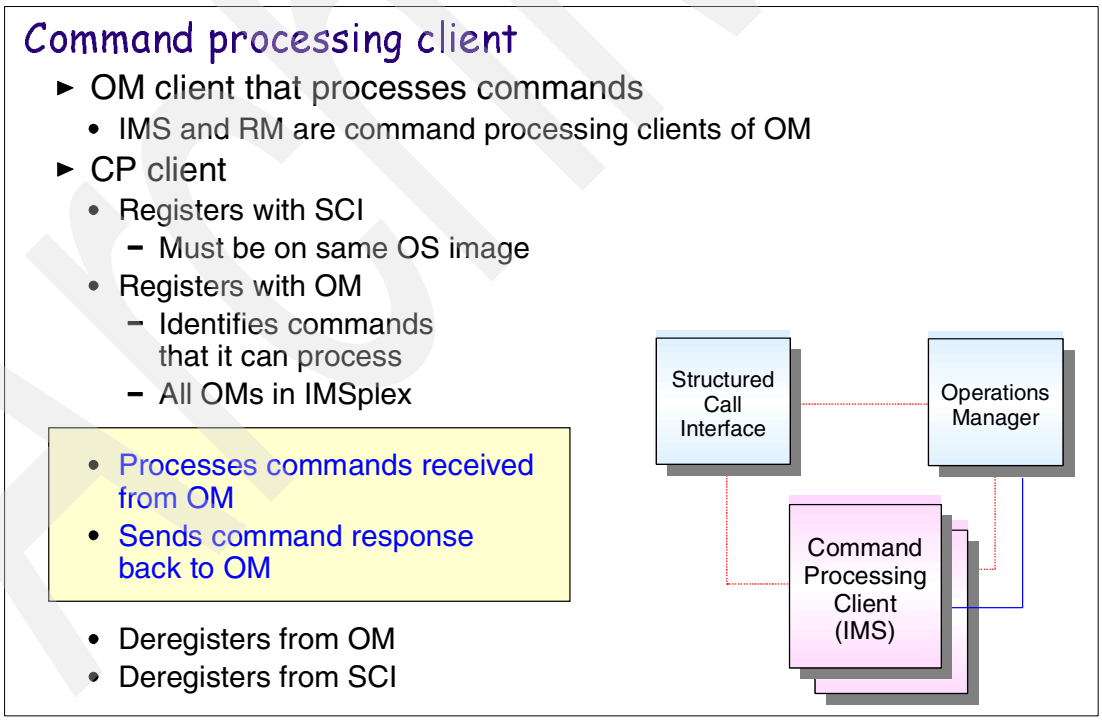


Figure 7-11 Command processing client

IMSs responsibility is to receive the command, process it, and respond to OM. IMS does not communicate directly with any AO client. A parameter in new PROCLIB member DFSCGxxx (CMDSEC) determines whether or not IMS should perform command authorization for

commands from OM. Command authorization can (and probably should be) performed by OM prior to sending the command to IMS.

7.5.9 AO clients

An AO client is one which enters the command to OM. Human operators are not in direct contact with OM. For example, a TSO SPOC is delivered with IMS and has an ISPF application which registers with SCI and uses SCI services to communicate with OM. The human operator may be working at a workstation and not in direct session with an OS/390 AO client. In this case, the AO client would merely receive the command message from the workstation and pass it along to OM. Figure 7-12 shows an AO client.

A new Data Management Tool, DB2 Control Center, supports a GUI interface to DB2 and through OM to IMS, giving that operator the ability to control both IMS and DB2 from the same terminal.

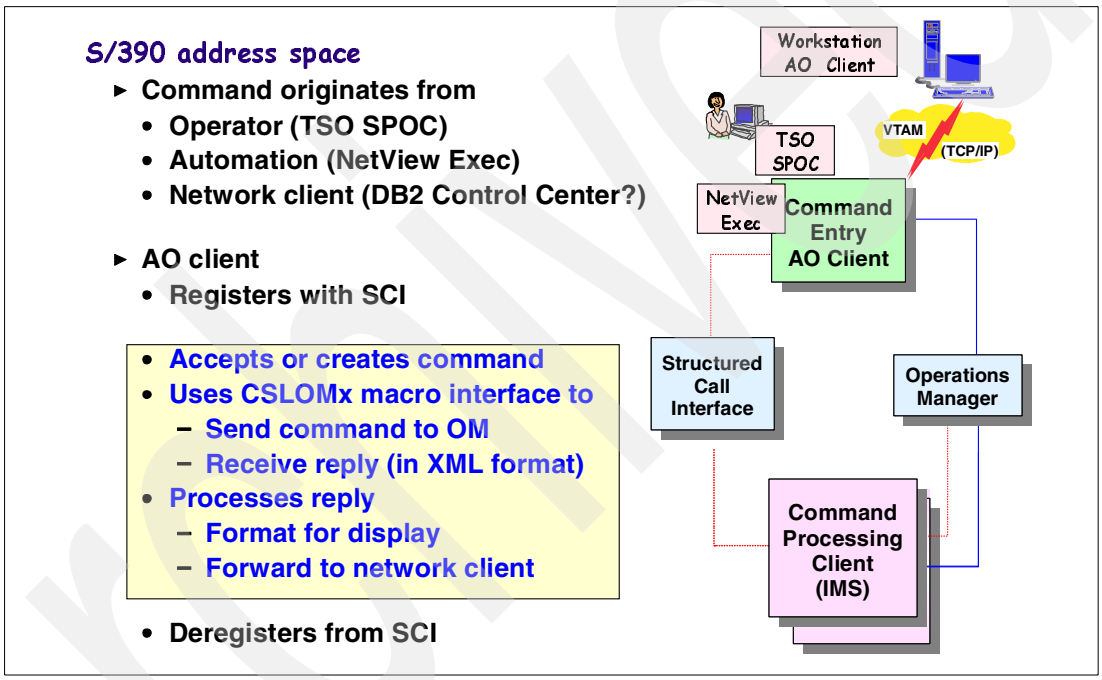


Figure 7-12 AO Client

Command entry (CE) AO clients

These are clients through which commands are entered to the OM and then to the command processing clients. TSO SPOC is an example of such a client.

For human operators, it is the responsibility of the AO client to format the IMS response for display. IMS will return all responses to OM encapsulated in XML tags. This makes it difficult for a human to read, the AO client should interpret the response and display it in a more readable format. The IMS Version 8 TSO SPOC does this.

These clients use the CSLOMCMD macro interface to OM.

Command forwarding (CF) AO clients

These are clients which are forwarding commands strings built elsewhere, probably by a workstation SPOC (GUI) and processing the returned responses.

OM forwards these commands to a CP client, and then returns the responses to the AO client. AO clients may be people sitting at a terminal, or they may be automation programs, such as NetViews EXECs.

Another type of AO client is an automation client. In this case, and automation program, such as a NetView EXEC, can join the IMSplex and enter commands to IMS. For example, if a message indicating that a transaction has been stopped is captured, a **/START TRAN** command could be issued to that IMS.

These clients used the CSLOMI macro interface to OM.

7.5.10 Commands from the OM API

With the new CSL architecture, new commands and command syntax have been created to operate on IMS systems at the IMSplex level. The OM interface supports both the new IMSplex commands and the classic IMS commands. Actually, OM supports any kind of command as long as someone registers it. OM's services include command entry and response, and command authorization.

IMSplex commands or commands from the OM API work differently from classic IMS commands. Classic IMS commands are those that are in the original or classical IMS command syntax, that is the command format allowed from the IMS master terminal.

Following are the general differences between IMSplex commands and classic IMS commands:

- IMSplex commands are commands that apply to the entire IMSplex. They can be entered only from the OM API.
- Some classic IMS commands are not supported by the OM API.
- Some classic IMS commands are not recoverable.
- Pre-IMS Version 8 Automated Operator Interface (AOI) application programs that issue **/DISPLAY** commands will not see some status information that is kept in the Resource Manager.
- The OM API does not support all of the variations in syntax that were acceptable before. IMS has to register with OM and indicate which commands it can process. Only a few of the variations are registered by IMS. Refer to *IMS Version 8: Command Reference,* SC27-1291, for the list of the command verbs and primary keywords that can be issued through the OM API.

Tip: The OM input exit can be used to check the syntax (spelling) of IMS commands and change those that used to be valid (for example, DATABASE or /DISP) to valid forms (DB and /DIS).

IMSplex commands can only be entered through the OM interface, they cannot be entered directly to IMS. The command responses returned are in XML format. XML has been chosen as a programming interface. The responses must be translated from XML to a suitable display format.

IMSplex commands also support filters and wildcards for resource name selection. The percent sign (%) wildcard is used to represent a single character substitution, and the asterisk (*) wildcard is used for multi-character substitution.

Archived

7.5.11 IMSplex commands

The following commands are new with IMS Version 8 and can only be processed via the OM interface:

- **INITIATE**

  INITiate an IMSplex global process. In IMS Version 8, the only global process available is global online change.

- **INIT OLC**

  Starts a global online change (OLC).

- **TERMINATE**

  TERMinate process. This command is used to terminate a global process when something goes wrong. For global online change, this would be the equivalent of the **/MODIFY ABORT** command.

- **TERM OLC**

  Stops a global online change that is in progress.

- **DELETE**

  DELete resource. This command is used to delete a resource. In IMS Version 8, the only resources that can be deleted are the dynamic LE runtime parameters added earlier with the UPD LE command:

  - **DEL LE** deletes runtime LE options.

- **UPDATE**

  UPDate resource. This command is used to update a resource. In IMS Version 8, two resources may be updated — the dynamic LE runtime parameters and some TRANSACTion attributes:

  - **UPD LE** updates runtime LE options.
  - **UPD TRAN** updates selected TRAN attributes.

- **QUERY**

  QueRY resource. This command has several parameters. All of the QRY commands include a SHOW parameter to specify what fields you want returned, or you can say SHOW(ALL):

  - **QRY IMSPLEX** returns the names of the members in the IMSplex and displays information about the IMSplex itself.
  - **QRY LE** returns runtime LE options.
  - **QRY MEMBER** returns status and attributes of the IMS members in the IMSplex and displays information about a specific member by member type.
  - **QRY OLC** displays information from the new OLCSTAT data set used for global online change.
  - **QRY TRAN** returns info similar to **/DIS TRAN**, it displays transaction attributes.
  - **QRY STRUCTURE** returns structure information of the RM resource structure.

For complete details on the new IMS commands, refer to *IMS Version 8: Command Reference,* SC27-1291.

Archived

7.5.12 Classic IMS commands

Most classic IMS commands can be entered through the OM API interface. Each IMS will register with OM those commands which it will accept. In general, all IMS classic commands can be entered through the OM interface except those that apply to the inputting LTERM, such as those in Example 7-3.

For example, it would not make sense to enter the **/SIGN ON** command from the SPOC, because this is not an IMS terminal.

Example 7-3 Classic commands which cannot be issued through OM

```
/EXC, /EXIT, /FORMAT
/HOLD, /(UN)LOCK node, pterm,lterm
/SET, /SIGN
/TEST MFS, /RCL, /REL
```

7.5.13 IMS asynchronous command response

When a command response includes a synchronous DFS058I message followed by one or more asynchronous messages (for example, DFS0488I), the synchronous DFS058I message is not returned, only the messages indicating the command completed are returned. This applies only to commands routed through the OM API and does not apply to existing interfaces.

7.5.14 Presence of resource structure

Some IMS commands have global impact if sysplex terminal management is active and a resource structure is present, such as the command in Example 7-4.

Example 7-4 Commands with Global impact if resource structure present

```
/STOP NODE ABC
```

NODE XYZ is flagged as stopped in the resource structure, now NODE ABC cannot log on to any IMS in IMSplex.

7.5.15 Commands indifferent to IMSplex

Some classic commands execute on every IMS which receives it. They are not aware of the IMSplex, such as the command in Example 7-5.

These are commands for resource status recovery and are not supported by STM. For example, **/DIS TRAN** will execute on every IMS. Because part of the command output is the global queue count (in a shared queues environment), each IMS will query CQS, which queries the shared queues structure, for this count.

Example 7-5 Command indifferent to IMSplex

```
/DIS TRAN TRX1 QCNT
```

Most commands depend on several factors:

- The command source
- If RM is active with a resource structure
- The effect of significant status
- If the resource exists on the resource structure

- If the resource is owned by this IMS
- If the resource owned by another IMS
- If the command is for display or update

Those commands displaying or changing the status of an STM-support resource (for example, node, LTERM, user), execution of the command depends on several factors. For example, if a NODE is logged on to IMS1, the **/STOP NODE** command can only be executed by IMS1. For display commands, only one IMS will return global information. For example, for **/DIS LTERM XYZ QCNT**, only one IMS will display the global queue count. Each IMS will display other local LTERM attributes. For these command, OM selects one of the IMSs as the MASTER for that command. Only the MASTER will perform global operations.

There are a significant number of changes to the way IMS commands work in this environment. They are all documented in the *IMS Version 8 Command Reference* manual, and should be studied when migrating to IMS Version 8 with a CSL (even without a CSL, there are some changes).

7.5.16 OM user exit routines

The user may write user exits for the OM address space. All OM exits are optional, no samples are provided. Exits may be called for connection, initialization, termination, input, output, and security.

Like SCI, OM uses BPE services to call and manage its user exits. Since CSL components are built on the Base Primitive Environment, they can take advantage of BPE common services and interfaces. This aids in creating a common look and feel to the configuration of new IMS components. This is one of the areas where the three new address spaces (SCI, OM and RM) are the same.

BPE also provides a common user exit runtime environment for all user exits. This environment includes a standard user exit parameter list, callable services, static and dynamic work areas for the exits, and a recovery environment for user exit abends.

The OM user exit routines receive control in the OM address space in an authorized state. BPE enables you to externally specify the user exit modules to be called for a particular user exit type by using EXITDEF= statements in the BPE user exit list PROCLIB members. User exits are defined in the BPE user exit list PROCLIB member for OM as shown in Example 7-6.

Example 7-6 User exit definition for OM in BPE

```
EXITDEF=(TYPE=xxx,EXITS=(XXX,YYY),COMP=OM)
```

OM supports the following user exits:

- OM client connection user exit
- OM initialization/termination user exit
- OM input user exit
- OM output user exit
- OM security user exit
- BPE statistics user exit for OM

OM client connection user exit

OM client connection user exit is called when client registers or deregisters commands with OM and indicated it is ready to accept commands. It is called again when client deregisters from OM. This exit is defined in BPE user exit list PROCLIB member via EXITDEF statement specified with TYPE=CLNTCONN

OM initialization/termination user exit

OM initialization/termination user exit is called:

- After OM has completed initialization
- Called after each IMSplex has initialized
- Called when OM is normally terminating
- Called when an IMSplex is normally terminating

This exit is defined in BPE user exit list PROCLIB member via EXITDEF statement specified with TYPE=INITTERM

OM input user exit

OM input user exit is called when OM receives a command, before OM processes the command, which allows the command to be modified or rejected. This exit is defined in BPE user exit list PROCLIB member via EXITDEF statement specified with TYPE=INPUT.

OM output user exit

OM output user exit is called to allow a user to view and manipulate output from OM. This exit is called for the following events:

- A command has been processed and is ready to be delivered to the originator of the command. The exit can modify the command response text before the response is delivered.
- When an unsolicited message is received from a client, for example, an IMS control region (late reply), via the CSLOMOUT API.

This exit is defined in BPE user exit list PROCLIB member via EXITDEF statement specified with TYPE=OUTPUT.

OM security user exit

Use the OM Security user exit to perform security checking during command processing. This exit is given control after the OM Input exit. This exit is invoked when the CMDSEC= parameter on the OM procedure is specified as A or E as follows:

- When A is specified, both this exit and RACF (or equivalent) are used for OM command security.
- When E is specified, only this exit is called for OM command security.

This exit has the ability to accept or reject commands, and it can override RACF definitions. This exit is defined in BPE user exit list PROCLIB member via EXITDEF statement specified with TYPE=SECURITY.

BPE statistics user exit for OM

The BPE statistics user exit routine enables you, at regular intervals, to gather statistics related to an IMS component that is running with a BPE address space. The exit is also called a final time during normal shutdown of the address space. The BPE statistics user exit routine is optional.

The statistics exit is called on a time-driven basis:

- The interval between successive statistics exit calls is specified on the STATINTV parameter in the BPE configuration PROCLIB member.
- The exit is first called soon after BPE initialization completes.
- Subsequent calls occur every STATINTV seconds after the previous call returns.

- The BPE statistics exit is also called one final time during normal address space shutdown processing.

This exit is defined in BPE user exit list PROCLIB member via EXITDEF statement specified with TYPE=STATS.

Detailed information about these exits can be found in *IMS Version 8: Common Service Layer Guide and Reference,* SC27-1293, and *IMS Version 8: Base Primitive Environment Guide and Reference,* SC27-1290.

7.6 Resource Manager (RM)

RM is an IMS address space that manages global resources and IMSplex-wide processes in a sysplex on behalf of its clients. IMS is one example of an RM client that uses RM to manage:

- Global message destination and terminal resources
- Global online change

The Resource Manager allows IMS to begin to address the task of resource management and coordination across the IMSplex.

To provide the global resource management services, a resource structure is required. In this structure, RM maintains information about the IMSplex and its members, as well as information about the Sysplex Terminal resources. It may also contain information about global processes, however, the structure is not required for global online change.

The resource structure is optional in this environment. If the structure has not been defined, sysplex terminal management (IMS's exploitation of the global resource management services of RM) is not enabled and only a single RM address space is allowed in the IMSplex.

Prior to IMS Version 8 and the Resource Manager, IMSplex-wide resource management was the responsibility of operations. It was possible that these resources could be defined inconsistently such that the results of an operation differed between IMSs. For example, a resource name could represent an LTERM on one IMS and a transaction on another. Depending on where the message arrived, the message would be queued differently. Or an LTERM could be assigned to different nodes on different IMSs.

Although the Resource Manager does not itself provide any management or coordination functions, it does provide the infrastructure which allows IMSs in the IMSplex to implement some resource management functions. Figure 7-13 shows an example of an IMSplex and its resources.

Archived

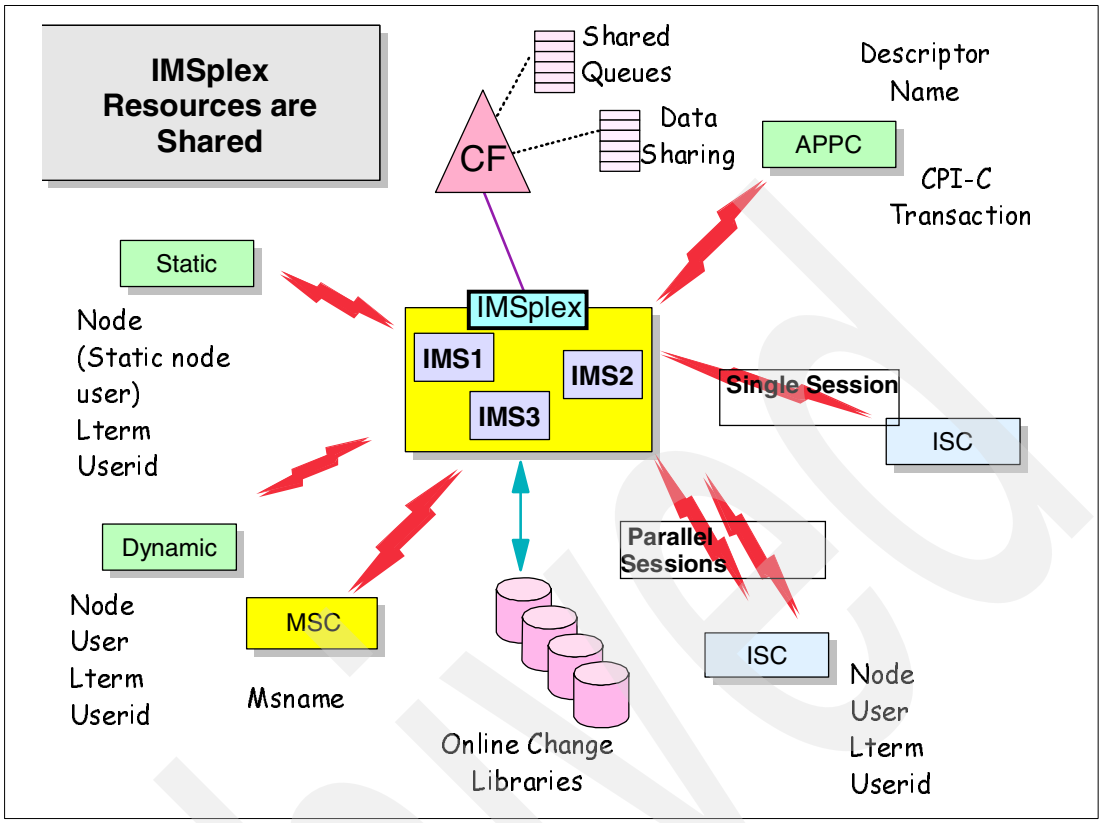


Figure 7-13 Resources in an IMSplex

7.6.1 IMSplex components related to RM

The IMSplex components related to RM include:

- RM address space
  - At least 1 RM required in IMSplex defined with CSL
  - 1 RM on each z/OS system best for performance
  - Only one RM allowed in IMSplex with no resource structure
- Common Service Layer address spaces
  - At least 1 Operations Manager (OM) required in IMSplex
  - Structured Call Interface (SCI) required on each z/OS system
- RM clients (IMS subsystems)
  - Each IMS DB/DC, DCCTL, or DBCTL subsystem is a client
- Coupling Facility list structure (resource structure)
  - Resource structure contains global resources (optional)
- Common Queue Server (CQS)
  - Required if resource structure is defined
  - RM registers to one CQS that must reside in the same system (LPAR)

7.6.2 Resource management functions

IMS Version 8 begins the process of exploiting the RM infrastructure by providing the following resource management functions:

- Sysplex terminal management (STM) consists of three subfunctions:
  - Resource type consistency: Guaranteeing that the same resource name is not used to define different resource types by different IMS systems
  - Resource name uniqueness: Guaranteeing that the same resource is not active on multiple IMSs at the same time
  - Resource status recovery: Restoring sysplex terminal and user status following a logoff/logon, signoff/signon, or IMS restart
- Global online change coordinates the online change process across all active IMSs in the IMSplex. This function requires the Operations Manager and an AOP (for example, the TSO SPOC) to submit the commands to initiate online change.
- Global callable services allows an IMS exit using FIND or SCAN callable services to retrieve global status when a node, LTERM, or user is not active locally but does have some global status on the resource structure.

7.6.3 Resource management infrastructure

Figure 7-14 shows a typical single image configuration of one IMS in a CSL configuration. It includes the IMS control region, the new CSL address spaces (SCI, OM, and RM), a resource structure (optional), and a Common Queue Server (CQS) to access the resource structure.

The complete architecture of Resource Management in the IMSplex is shown here. Full service requires an IMS control region, a Resource Manager, the Common Queue Server, and a resource structure. If no resource structure is defined, then CQS is not required (unless, of course, shared queues is active).

With the exception of the IMS control region, all of these address spaces are built on the Base Primitive Environment (BPE) — not shown in the figure. When additional IMSs on other OS/390 images are part of the IMSplex, each image must include an SCI address space and, if using a resource structure, a CQS address space.

RM and OM only need to be started on one z/OS image in the IMSplex, although multiples are recommended for availability and performance.

Archive

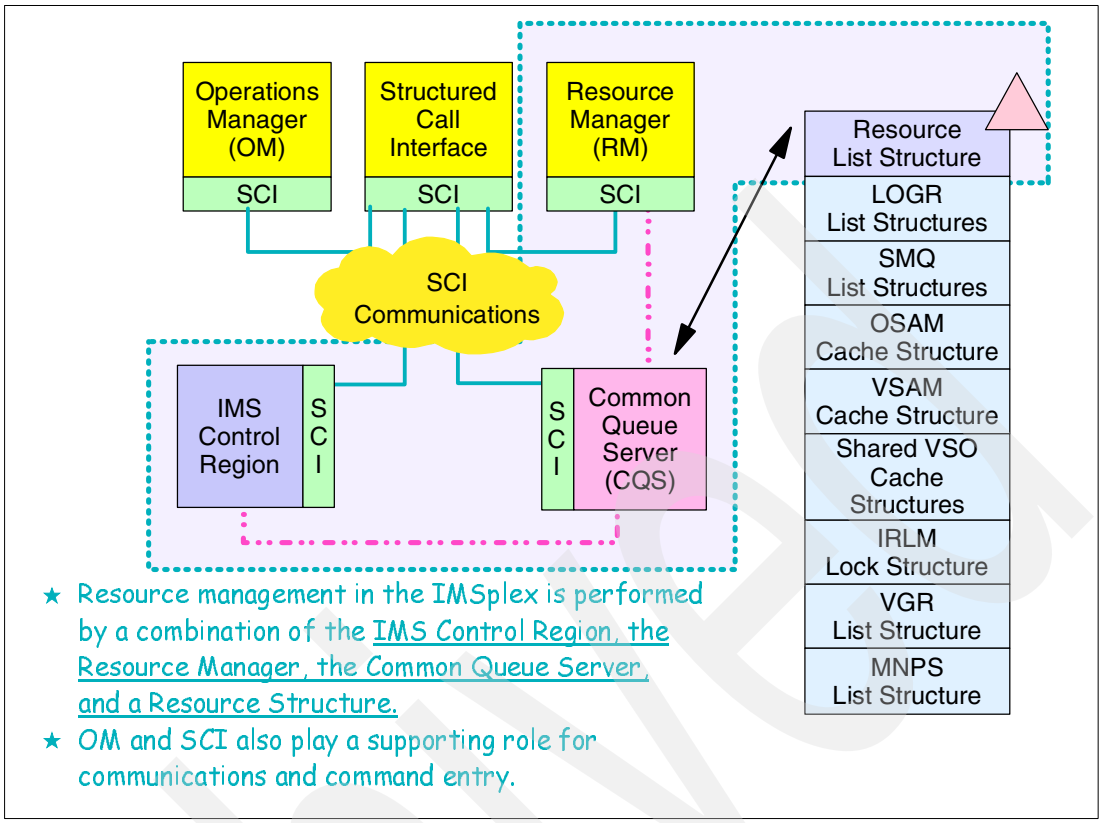


Figure 7-14 Resource Manager, Common Queue Server, and resource structure

7.6.4 RM clients and their roles

Figure 7-15 shows how these different components communicate with each other. IMS and RM communicate using SCI services. If IMS and RM are on the same OS/390 image, SCI uses cross-memory calls. If they are on different OS/390 images, SCI uses XCF signalling.

Archive

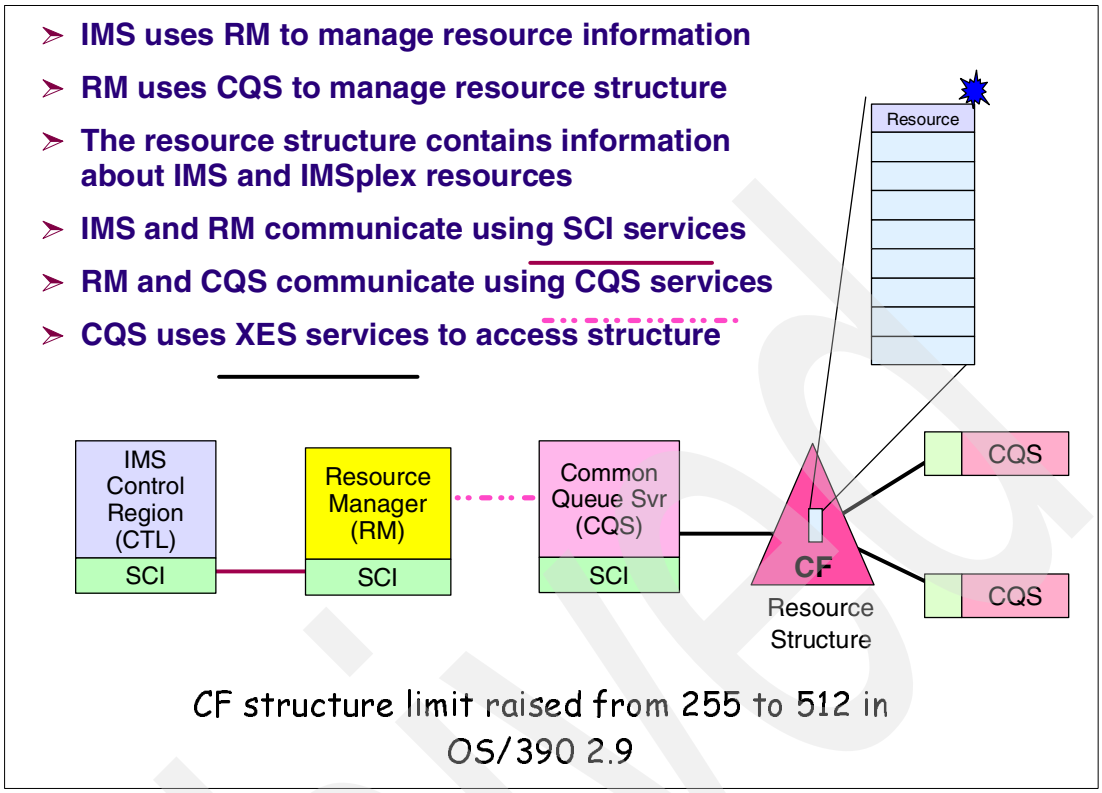


Figure 7-15 RM clients and their roles

Because the CQS interface was developed in IMS Version 6 for shared queues (before SCI), all address spaces communicating with CQS use this interface instead of the SCI interface. CQS still registers with SCI to join the IMSplex but does not use its communication services. CQS use OS/390 cross-system extended services (XES) to connect to and to access the resource structure. Note that a single CQS address space per OS/390 image can be used to access both the resource structure and the shared queue structures.

Resource Manager provides services to its clients. Currently, IMS is the only RM client, but the interface is open and documented, so that vendors and users may write their own RM clients. The client is responsible for utilizing RM services, that means that clients use CSL macros to maintain global resource information. When IMS acts as a client to RM, it registers with RM at IMS initialization time and deregisters from RM at IMS termination. During normal operations IMS adds or updates resource entries on the structure, may query resources and delete entries form the structure.

7.6.5 Resource structure

The resource structure is a CF list structure used by RM to contain global resource information, IMSplex-wide process information, and information about the IMSplex and its members. The resource structure is defined in the CFRM policy and allocated by CQS as a list structure. It is optional in a CSL environment, but it if exists, it contains information about IMSplex resources. Each resource has an entry in the structure with information about that resource.

For example, a node resource entry would identify the IMS system to which that node is logged on. A user resource entry would contain information about any active or held conversations for that user. The following is a list of the types of entries found in the resource structure:

- Transactions
- Nodes, LTERMs, MSNAMEs, APPC descriptors, users, user IDs
- Global processes such as online change
- IMSplex global and local information

Although the structure is optional, if it does not exist, then there is no global repository for sysplex terminal information and the sysplex terminal management function of IMS is disabled. Global online change, while it makes use of the structure for recovery from some error conditions, does not require the structure.

7.6.6 Resource Manager (RM) address space

The Resource Manager address space is a required component of the Common Service Layer and an integral part of the resource management infrastructure. While the RM does not in itself provide any of the resource management functionality, it does offer services to its clients (IMS control regions) to enable them to provide these functions.

IMS exploits RM services to provide sysplex terminal management (if a resource structure has been defined) and global callable services. At least one RM address space is required within the IMSplex. If a resource structure exists, then multiple RMs can be started and is recommended for performance and availability. If there is no resource structure then there can be only one RM in the IMSplex.

Like other IMSplex components, RM registers with SCI and uses SCI to communicate with its clients (IMS control regions). A single RM can support multiple IMS Version 8 control regions if they are in the same IMSplex. This is true even if those control regions are on different OS/390 images. Figure 7-16 shows one RM providing support for two IMS control regions on the same OS/390 image and one IMS control region on another OS/390 image. A total of 32 IMS control regions can be supported by a single RM. Note that, if a resource structure is being used, each RM requires a local (same OS/390 image) CQS to access the structure.

Archived

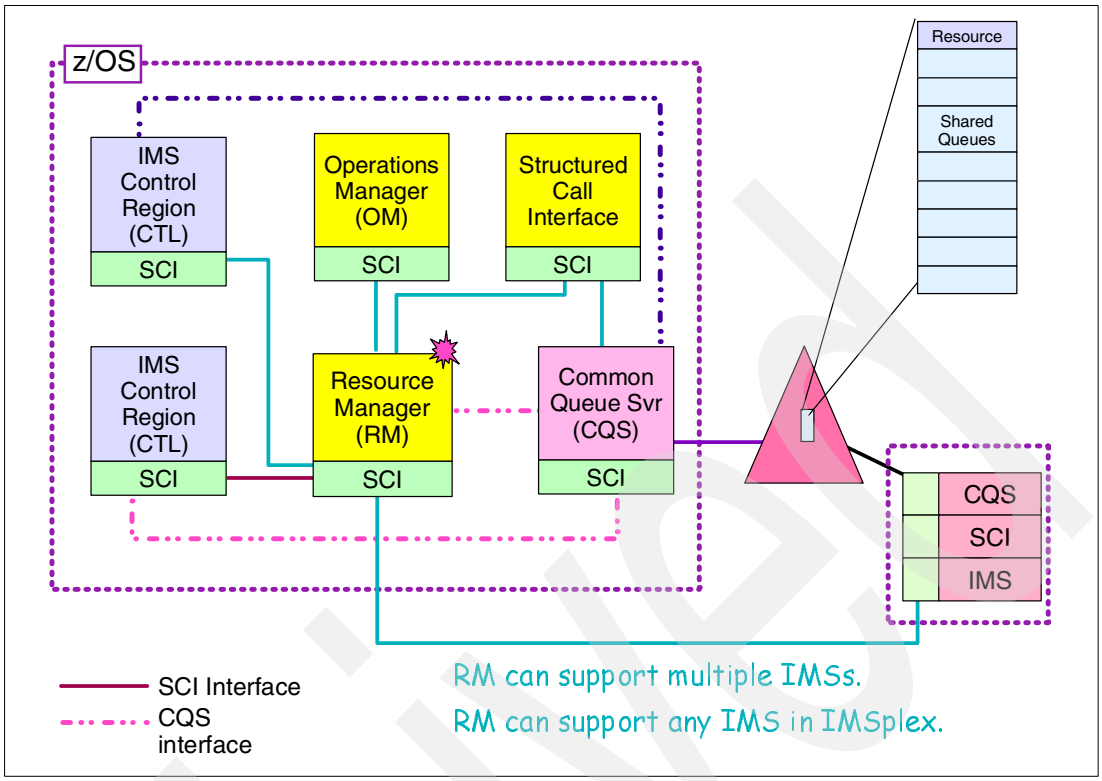


Figure 7-16 RM supporting IMSs

7.6.7 RM user exit routines

RM user exit routines enable you to customize and monitor your RM environment. All the RM user exit routines are optional and no samples are provided.

Like SCI and OM, RM uses BPE services to call and manage its user exits. Since CSL components are built on the Base Primitive Environment, they can take advantage of BPE common services and interfaces. This aids in creating a common look and feel to the configuration of new IMS components. This is one of the areas where the three new address spaces (SCI, OM and RM) are the same.

BPE also provides a common user exit runtime environment for all user exits. This environment includes a standard user exit parameter list, callable services, static and dynamic work areas for the exits, and a recovery environment for user exit abends.

The RM user exit routines receive control in the RM address space in an authorized state. BPE enables you to externally specify the user exit modules to be called for a particular user exit type by using EXITDEF= statements in the BPE user exit list PROCLIB members. User exits are defined in the BPE user exit list PROCLIB member for RM as shown in Example 7-7.

Example 7-7 User exit definition for RM

```
EXITDEF=(TYPE=xxx,EXITS=(XXX,YYY),COMP=RM)
```

RM supports the following user exits:

- BPE initialization/termination user exit for RM
- RM client connect user exit
- BPE statistics user exit for RM

Detailed information about these exits can be found in *IMS Version 8: Common Service Layer Guide and Reference,* SC27-1293, and *IMS Version 8: Base Primitive Environment Guide and Reference,* SC27-1290.

RM client connection user exit

This exit is called when a client connects (registers) to RM or disconnects (deregisters) from RM. This exit is called for the following events:

- After a client has successfully connected to RM
- After a client has successfully disconnected normally or abnormally from RM

This exit is defined as TYPE=CLNTCONN in the EXITDEF statement in the BPE user exit list PROCLIB member.

RM initialization/termination user exit

This exit is called for the following events:

- After RM has completed initialization
- After each IMSplex has initialized
- When RM is terminating normally
- When an IMSplex is terminating normally

This exit is not called during RM address space abnormal termination or IMSplex abnormal termination. This exit is defined as TYPE=INITTERM in the EXITDEF statement in the BPE user exit list PROCLIB member.

BPE statistics user exit for RM

The BPE statistics user exit can be used to gather both BPE and RM statistics. BPE statistics user exit provides statistics about BPE system functions and optionally, statistics about the IMS component running with BPE:

- BPE statistics user exit can gather statistics about any BPE-based address space
- Available to CQS, OM, RM, and SCI address spaces
- RM provides statistics
- BPE-owned user exit

BPE statistics user exit is driven based upon a timer and it is defined in BPE user exit list PROCLIB member via EXITDEF statement specified with TYPE=STATS. BPE statistics user-supplied exit parameter list (BPESTXP) field BPESTXP_COMPSTATS_PTR contains a pointer to RM statistics header. RM statistics header is mapped by CSLRSTX macro.

7.6.8 Common Queue Server (CQS)

Common Queue Server (CQS) is a generalized server that manages data objects on a Coupling Facility list structure, such as a queue structure or a resource structure, on behalf of multiple clients.

CQS receives, maintains, and distributes data objects from shared queues on behalf of multiple clients. Each client uses its CQS for accessing the data objects on the Coupling Facility list structure. IMS is one example of a CQS client that uses CQS to manage both its shared queues and shared resources.

Common Queue Server (CQS) was introduced in IMS Version 6 to provide an interface between IMS and the shared queue structures. In IMS Version 6, one CQS could support only a single IMS control region. In IMS Version 7 CQS was enhanced to support multiple IMS

control regions as long as they are on the same OS/390 image and belong to the same shared queues group.

In IMS Version 8 CQS has been further enhanced to also support the interface between Resource Manager and the resource structure while continuing to support shared queues. That is, a single CQS can support both shared queues and the RM. However, CQS shipped with IMS Version 8 does not support IMS Version 6 or IMS Version 7 shared queues. For example, if you have both IMS Version 7 and IMS Version 8 in the same shared queues group, they require different CQSs. Figure 7-17 shows one CQS supporting both IMS shared queues and RM.

Some of the CQS enhancements delivered with IMS Version 8 are listed here:

- Support for list structures with programmable list entry ids (LEIDs) to guarantee uniqueness
- Support for structure alter and autoalter, system-managed rebuild, and system-managed duplexing
- Ability to initiate structure repopulation to rebuild a failed resource structure

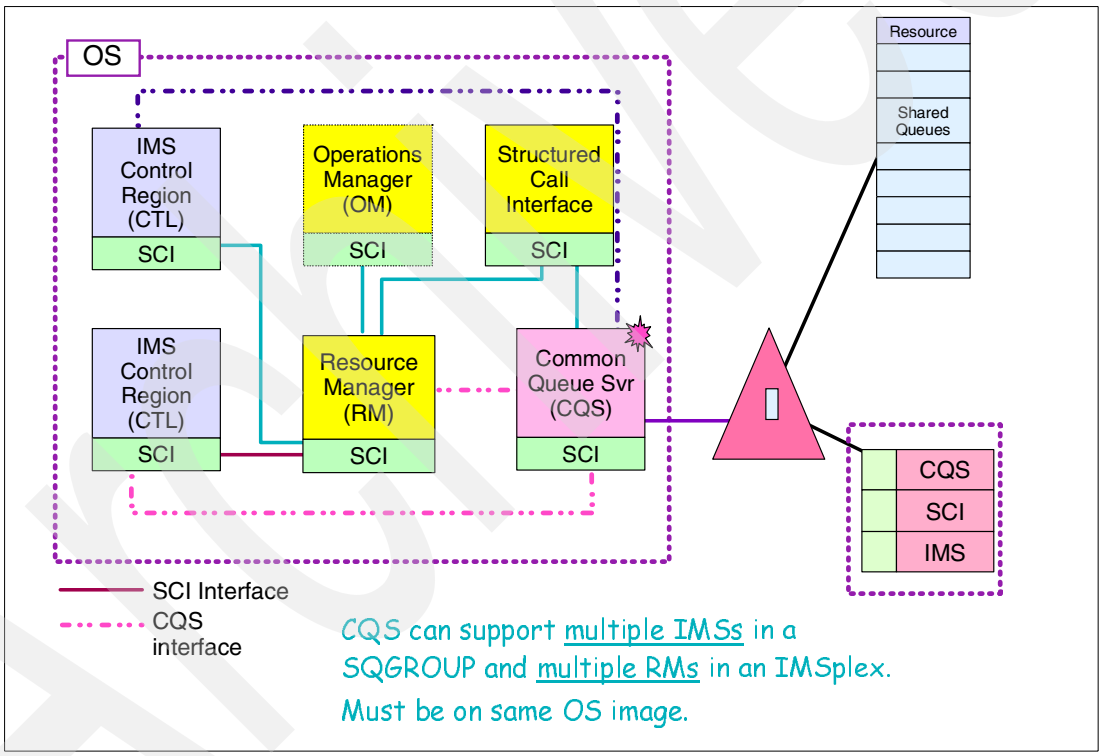


Figure 7-17 CQS supporting shared queue and RM

7.6.9 CQS requirements

A CQS is required on every OS image where RM exists and a resource structure is being used:

- CQS for RM not required if no RM structure
  - CQS required for shared queues
  - Same CQS can support both RM and IMS shared queues
- CQS registers with SCI

- RM registers with CQS
- RM uses CQS services to manage RM structure
- If using BPEINI00, CQS initialization module
  - BPEINIT=CQSINI00
  - Module must be in //STEPLIB
  - Can execute CQSINIT0 instead of BPEINI00
- RM initialization PROCLIB member CQSIPxxx
  - CQSINIT=xxx
  - Member must be in //PROCLIB

7.6.10 CQS components

CQS maintains the following components:

- Primary message queue structure
  - OS/390 Coupling Facility list structure that contains shared queues.
- Overflow message queue structure
  - A Coupling Facility list structure that contains shared queues when the primary structure reaches an installation-specified overflow threshold. The overflow structure is optional.
- Resource structure
  - An OS/390 Coupling Facility list structure that contains uniquely named resources.
- OS/390 log stream
  - A shared OS/390 log stream that contains all CQS log records from all CQSs connected to a structure pair. This log stream is important for recovery of shared queues, if necessary. Each structure pair has an associated log stream.
- Checkpoint data sets
  - Local data sets that contains CQS system checkpoint information.
- Structure recovery data sets (SRDS)
  - Shared data sets that contain structure checkpoint information for shared queues on a structure pair. Each structure pair has two associated SRDSs.

Archived

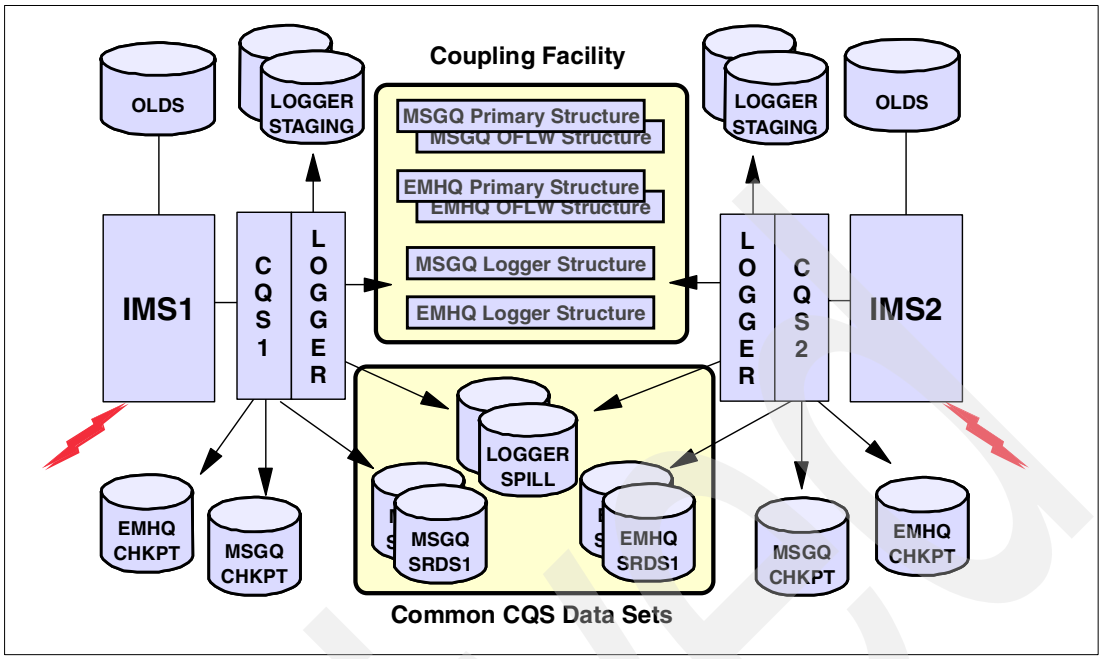


Figure 7-18 Shared queues components

The CQS address space can support both the shared queues structures as well as the resource structure. To support the resource structure, new parameters have been added to CQSIPxxx (initialization PROCLIB member) and CQSSGxxx (global PROCLIB member). There are no changes to CQSSLxxx (local PROCLIB member).

7.6.11 Supporting multiple clients

You can use one CQS address space to support multiple clients. Examples of clients are IMS and RM.

As many as 32 different clients on the same LPAR can connect to Coupling Facility structures through a single CQS by using the CQSCONN request.

Archive

Archived

8

Sysplex terminal management (STM)

In an IMSplex, IMS operations and users are faced with the complexities of managing resources which may have been defined to multiple IMSs within the IMSplex, but which must be managed with the same level of control as if there were a single IMS.

For example, in a single IMS, a single session NODE can log on only once, an LTERM can be assigned to only one NODE at a time and therefore can be active only once, optionally, a USER can be signed on only once, and so on. A single IMS can enforce this because that single IMS is aware of all instances of that resource's activity.

IMS is a queuing system. When a message arrives or is created in IMS, IMS uses a process called ***find destination*** (FINDDEST) to determine how to handle (queue) the message. Destinations (typically) are transactions codes, logical terminal names, or MSC logical link names (MSNAMEs). When IMS queues a message to one of these destinations, there is never any confusion about the type or end-point of that destination.

Similarly, an end-user can have some significant status, such as being in a conversation, in MFS TEST mode, or have several other statuses which we might call ***significant status***. If that user's session fails, the IMS to which the user was connected can remember that significant status and restore it the next time the user signs on.

Note: In all of these cases, when multiple IMSs are involved, it is necessary for each IMS to know the status of a resource on other IMSs in the IMSplex.

Sysplex terminal management (STM) addresses these and similar resource management issues of the IMSplex. This chapter discusses STM.

© Copyright IBM Corp. 2003. All rights reserved.

8.1 STM objectives

Sysplex terminal management objectives are to:

- **Provide for resource type consistency**

  Do not allow message destination resources to be defined as different resource types on different IMSs.

  Message destinations are names which IMS uses to queue a message to its destination, and include transaction codes, LTERM names, MSC logical link names (MSNAMEs), and APPC descriptor names.

  For example, do not allow the same resource name to be defined as a transaction on IMS1 and as an LTERM on IMS2. This is the only STM feature supported for APPC.

- **Provide for resource name uniqueness**

  Do not allow a resource to be active on more than one IMS at a time.

  Resources managed by this function are single session VTAM NODEs, Extended Terminal Option (ETO) USERs, LTERMs, and (if requested) USERIDs.

  For example, do not let LTERMA be active on IMS1 and IMS2 at the same time. Do not allow USERA to be signed on to both IMS1 and IMS2.

- **Support global resource status recovery**

  Allow a terminal user to terminate a session on one IMS (normally or abnormally) and resume (or recover) that user's status on another session with another IMS. A status for which recovery is supported is called significant status.

  For example, if USERX is in a conversation on IMS1, and IMS1 fails, that user may resume the conversation after re-establishing a new session on IMS2.

- **Support global callable services**

  User exits can access terminal and user information across IMSplex.

  Allow an IMS exit using callable services to determine the status of a resource anywhere within the IMSplex.

  For example, the Signon Exit (DFSSGNX0) should be able to determine if a USER is signed on anywhere within the IMSplex.

8.2 STM environment

Sysplex terminal management is an optional function of IMS. It is enabled only when IMS is executing in a Common Service Layer (CSL) environment with shared queues. It utilizes the services of the Resource Manager with a resource structure to save information about, and manage the status of, IMSplex VTAM-connected terminal and user resources, and to some extent, transactions.

8.2.1 STM configurations

Figure 8-1 shows a possible configuration of a single IMS in a CSL environment with shared queues. The shaded area indicates those components used directly by STM. Additional IMSs in the IMSplex would utilize the same resource and shared queue structures.

Other IMSs on other OS/390 images would have similar configurations. Each OS/390 image would require its own SCI address space and each may have a local Resource Manager

address space, although only one RM is required within the IMSplex. There is, of course, a single resource structure and a single set of shared queues structures.

Wherever there is a Resource Manager accessing the resource structure, there must also be a Common Queue Server. A single CQS address space can provide client services to RM for the resource structure and to IMS for the shared queue structures. CQS must reside on the same OS/390 image as its clients (RM and IMS).

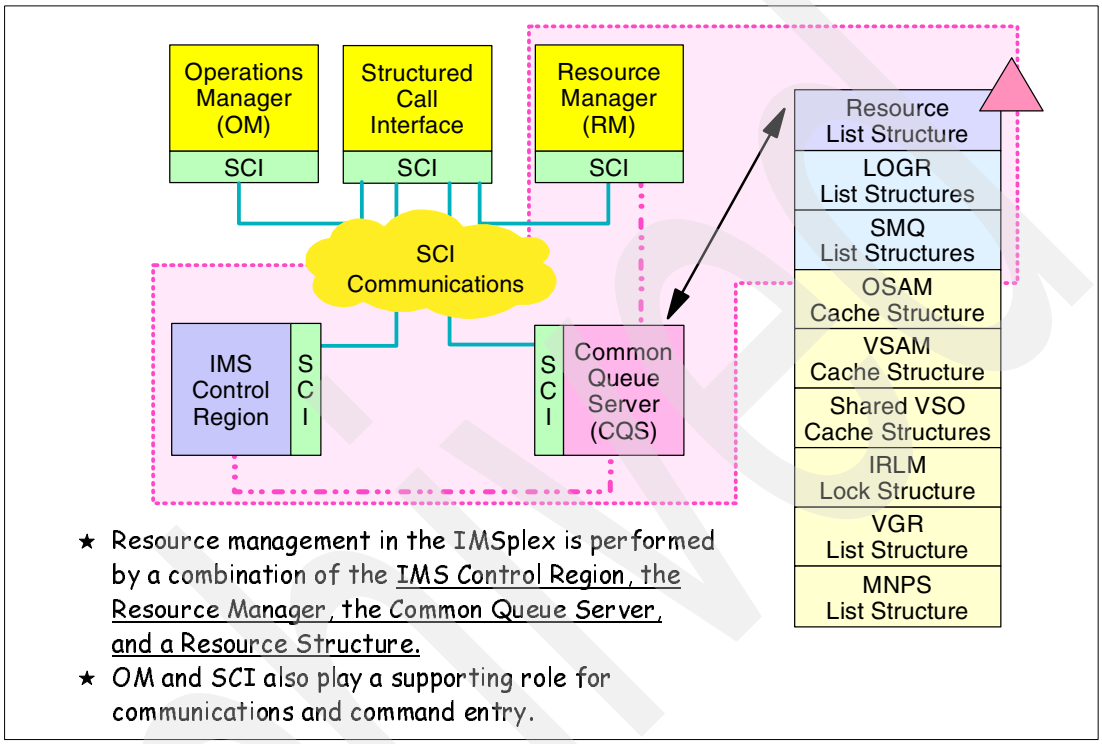


Figure 8-1 Sysplex terminal management configuration

Sysplex terminal management (STM) applies only to VTAM terminals and users. BTAM and OTMA resources are not supported. APPC is supported only minimally (described later — see 8.4.1, “Resource type consistency” on page 182).

Global resource sharing requires the Resource Manager address space, a resource structure, and the IMSs must be running with shared queues enabled.

Without a structure, the user can opt for local status recovery (same as previous versions) or no status recovery (new in IMS Version 8).

8.3 IMSplex resources

IMS resources in an IMSplex include many different types, including IMS system address spaces, IMS system data sets, IMS defined databases, transactions, and applications, dependent regions running application programs, VTAM and OTMA network resources, batch and utility address spaces, and probably several more. Most of these resources have names by which they are known to IMS. When IMS systems are cloned, or have similar definitions, many (or all) of these names are the same throughout the IMSplex and can form the basis for IMSplex-wide system management functions.

Sysplex ***terminal*** management addresses the management of a subset of these resources, primarily those defined as part of the VTAM network. These resources, and the names they are known by, are shown in Figure 8-2. Note that STM supports neither BTAM nor OTMA resources.

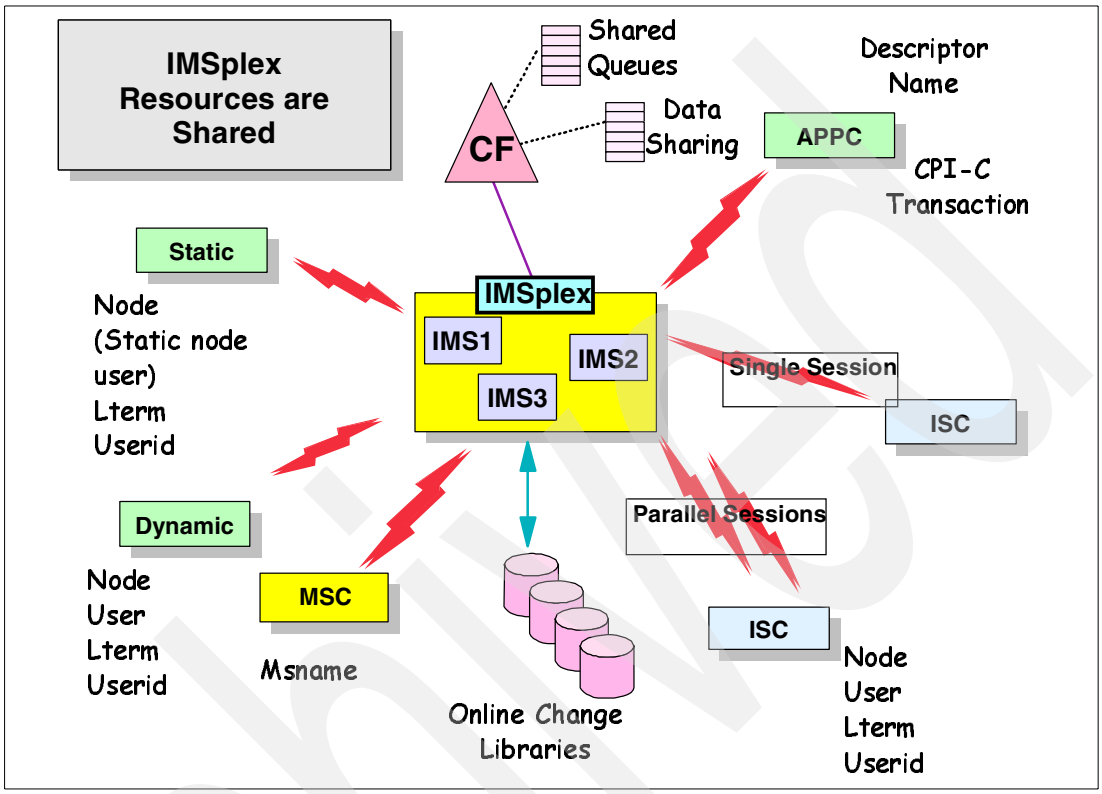


Figure 8-2 Resources in an IMSplex

This figure identifies the resources managed by sysplex terminal management. Each of these resources can be the source or destination of an IMS message, and has one or more names associated with it. Each name represents an IMS control block. How IMS handles these messages is determined solely by its ***message destination*** name, which usually represents an anchor block from which to queue the message. Each named resource may be represented by an entry in the resource structure.

8.3.1 Statically defined VTAM resources (not parallel-session ISC)

Statically defined VTAM resources are defined in the IMS system generation using the TYPE, TERMINAL, and NAME macros, the NODE name being defined on the TERMINAL macro and the LTERM name being defined on the NAME macro:

```
TYPE      UNITYPE=SLUTYPE2 (for example)
TERMINAL  NAME=NODEname
NAME      LTERMname
```

These names may be used to create entries in the resource structure. For parallel session ISC terminals, or for NODEs created dynamically using ETO, another control block (the SPQB) is created representing the ETO USER. Because there is no USER equivalent for statically defined single session VTAM resources, a new name is invented to be used strictly for creating resource structure entries. This new name is the STATIC NODE USER and has the same name as the NODE name. There is no equivalent IMS control block, and it is used

ONLY for resource structure entries. No IMS command or log record will ever refer to a static NODE user.

If a user signs on from a static NODE, a USERID is associated with that session for security (authorization) processing. Although this is optional for static terminals, many IMS shops require a user to sign on. Users are, by default, prevented from signing on to more than one session unless the SGN= parameter is coded in PROCLIB member DFSPBxxx. By coding SGN=M, the user is allowed to sign on multiple times.

8.3.2 Dynamic (ETO) resources

When a user logs on to IMS using the Extended Terminal Option (ETO), a control block structure (VTCB) representing that NODE is created in IMS. When that user signs on, a user structure is created consisting of a user control block (SPQB) and one or more LTERMs (CNTs). The names of the USER and LTERM(s) associated with the signed on user may be decided by default or by the Signon Exit (DFSSGNX0). Additionally, since the user is required to sign on, a USERID is also associated with the user.

When a dynamic user with significant status (for example, in a conversation) signs off or logs off, the user structure (USER + LTERM) is disconnected from the terminal structure but is not deleted. If that same user then signs on from another NODE, the user control block structure is connected to the new NODE. Any status and messages queued for that user follow the user to the new NODE (messages are queued by LTERM name). To be consistent, sysplex terminal management keeps much recoverable status information about a user in the USER entry in the resource structure, therefore the reason for the static NODE user invention mentioned above for statically defined resources.

8.3.3 Single session ISC resources

ISC, or LU 6.1, is an SNA program-to-program protocol that allows IMS to communicate with other subsystems. ISC is normally used to send transactions and replies back and forth between IMS and CICS systems, but could also be used, in lieu of MSC, for IMS to IMS communications.

Single session ISC resources are defined just like single session non-ISC resources, with the same name types. These are NODE, (STATIC NODE) USER, LTERM, and USERID.

```
TYPE      UNITYPE=LUTYPE6
TERMINAL  NAME=NODEname
NAME      LTERMname
```

8.3.4 Parallel session ISC resources

Parallel session ISC definitions, whether statically or dynamically defined, always have a SUBPOOL (SPQB) associated with them to represent the USER. An example of a statically defined parallel session ISC NODE with a maximum of two parallel sessions may be defined as:

```
TYPE      UNITYPE=LUTYPE6
TERMINAL  NAME=NODEname,SESSION=2   (defines the NODE resource)
SUBPOOL   NAME=username1            (defines the first USER resource)
NAME      LTERMname1                (defines the first LTERM resource)
SUBPOOL   NAME=username2            (defines the second USER resource)
NAME      LTERMname2                (defines the second LTERM resource)
```

NODE name, USER name, and LTERM name are defined during system definition, or during the ETO logon/signon process. The USERID is also provided during the logon/signon

process. Because parallel sessions have been defined, there may be multiple sessions between a NODE and IMS, with the NODE logged on multiple times. In this case, there must be a different USER (SUBPOOL) and LTERMs (NAMEs) associated with each logon. Each USER may have a distinct signon USERID.

8.3.5 MSC logical links (MSNAMEs)

MSNAMEs representing MSC logical links are defined during the system definition process by the MSNAME macro:

```
        MSPLINK   (defines the physical link - not in the resource structure)
        MSLINK    (defines the logical link - not in the resource structure)
name    MSNAME    (defines the MSNAME resource)
```

Like LTERMs and TRANSACTions, MSNAMEs represent message destinations and, in a shared queues environment, have their own queue type on the shared queue structure. IMS must be able to distinguish between this destination type and another.

8.3.6 Static transactions

Transactions are defined to IMS with the TRANSACT macro and represent message destinations. The TRANSACT macro generates an SMB control block which may be used in a non-shared queues environment for queuing input messages for scheduling. In a shared queues environment, the transaction message is queued on the transaction ready queue in the shared queues structure. Again, because they are a destination for queuing messages, they must be distinguishable from the other destinations.

Other than keeping track of defined transactions, there is no other support for transactions in STM. Transaction characteristics are not kept in the resource structure.

8.3.7 APPC CPI-C driven transactions

CPI-C driven transactions are used by APPC terminals to enter transactions directly to IMS application programs without going through normal transaction scheduling. The transaction code itself is not (must not be) defined to IMS with the TRANSACT macro. Instead, the CPI-C driven transaction is defined in the TP_Profile. Like static transactions, only the transaction code itself is managed by STM.

8.3.8 APPC output descriptors

APPC output descriptors are defined in IMS.PROCLIB member DFS62DTx as follows:

```
U descname parms
```

IMS treats the APPC descriptor name in the same way it treats an LTERM name — to determine the destination of a message entered, for example, from a program issuing a CHNG call. It is therefore necessary to be able to distinguish between a real LTERM and an APPC descriptor LTERM.

Except for keeping track of the defined APPC descriptor names, there is currently no other support in STM for APPC sessions. No status is kept for APPC descriptors.

8.3.9 Message destinations

Not all of these names are message destination anchor blocks. For example, although a message may be sent to a NODE, the message is queued off the control block (the CNT)

representing an LTERM assigned to that NODE. The following names are considered message destinations for purposes of queuing messages:

- Transaction codes
  - Static transactions defined in the IMS system generation
  - CPI-C transactions defined in the TP_PROFILE
  - Dynamic transactions defined by the Output Creation Exit (DFSINSX0)
- Logical terminal names (LTERMs)
  - Static LTERMs defined in the system generation
  - Dynamic LTERMs created in an ETO environment
- Logical link names (MSNAMEs)
  - Defined by the MSNAME macro in the IMS system generation
- APPC descriptor names
  - Defined in IMS.PROCLIB member DFS62DTx. Although messages are not queued by APPC descriptor name, this name is used to determine, from the descriptor definition, what the message destination is.

8.3.10 Summary of IMS resources managed by STM

The above resources are managed by the sysplex terminal management function of IMS using the resource structure as a repository for resource-related information.

8.4 STM terms and concepts

Sysplex terminal management introduces some new terms and concepts that need to be understood before discussing the functionality:

- Resource
  - VTAM terminal and user (static/dynamic)
  - Transaction (static/dynamic/CPI-C)
- Resource type consistency
  - For message destinations
- Resource name uniqueness
  - Single active resource
  - Single signon enforceable
- Resource status
  - Non-recoverable
  - Recoverable
  - Significant
- Significant status
  - Command
  - End-user
- Status recovery mode
  - Global (recover anywhere)
  - Local (recover on local only)
  - None (not recoverable)

Archived

- Ownership and Affinities
  - Resource ownership
  - RM affinity
  - VGR affinity

8.4.1 Resource type consistency

Resource type consistency is based on the concept of an IMS ***message destination*** as described in 8.3.9, “Message destinations” on page 180. For purposes of resource type consistency, a message destination is any named resource which may also be the name of a queue of messages for that resource.

For example, in a shared queues environment, transactions are queued off the (serial) transaction ready queue in the shared queue structure. Messages destined for logical terminals are queued off the LTERM ready queue. Output messages destined for MSC remote destinations are queued off the remote ready queue. When a message arrives in IMS, or an application program issues a CHNG or ISRT call to a TP-PCB, IMS must analyze the name in the message (call FINDDEST) to determine how to queue the message.

Message destinations have a name type of x’01’ as shown in Figure 8-18 and include:

- Transaction codes
- LTERM names
- MSNAMEs
- APPC descriptor names

Other resource types are not checked for consistency since they are not used for message queuing. Resource type consistency does not apply to:

- Nodes, users, userids

These are not message queue destinations. For example, it is perfectly alright to have the same name for a NODE, a USER, an LTERM, and a USERID.

In an IMSplex, each IMS system has its own system definition, although certainly a single system definition can be shared by all IMSs. When this is done, we say these IMSs are cloned, and the message destinations are obviously consistent. However, because it is not a requirement, and because each IMS might have its own system definition, it is important that, when resources are defined to these separate IMSs, resources of the same type are defined consistently. For example, it would be problematical if IMS1 defined a resource named PRSNL as a transaction, and IMS2 defined a resource named PRSNL as an LTERM. Because both transaction codes and LTERM names are message destinations, IMS1 would queue a message with destination PRSNL on the transaction ready queue, and IMS2 would queue it on the LTERM ready queue. Very confusing.

In a list structure (such as the resource structure) which has been allocated with user managed list entry IDs (LEIDs), no two entries can have the same LEID - they must be unique within the structure. When IMS wants the RM to create an entry on the structure, it provides a LEID consisting of the ***name type + resource name***. This may also be referred to as the ***resource ID***. In the above example, IMS1 would create a transaction entry with an LEID of 01PRSNL. If IMS2 later tried to create an entry for an LTERM named PRSNL, it would also provide a LEID of 01PRSNL. Because this LEID already exists in the structure, and because it must be unique, IMS2 would not be allowed to create a LTERM named PRSNL, and the logon would fail.

Figure 8-3 shows an example of the same resource name being defined by IMS1 as a transaction and IMS2 as an LTERM:

- With STM, when IMS1 is initialized, it will create a Transaction entry in the resource structure, including PRSNL. This entry will never be deleted unless the structure itself is deleted.
- If a user tries to log on to IMS2 from an LTERM named PRSNL, IMS2 would attempt to create an LTERM entry in the resource structure but would fail because it already exists as a Transaction. The logon would be rejected.

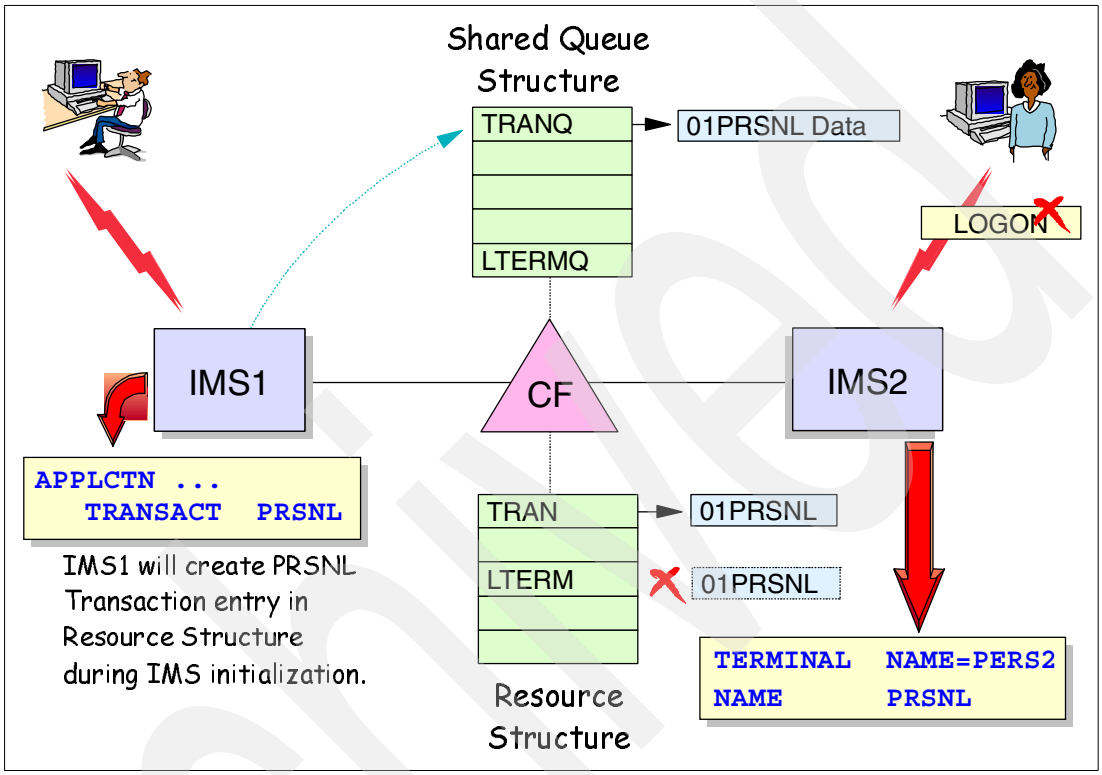


Figure 8-3 Resource type consistency

There is a difference here between statically defined terminals and dynamic ETO terminals. With a static terminal, the static NODE users (SNUs) and LTERMs are created and activated at logon time. If any one of the LTERMs defined for a static NODE is not consistent, then the logon is rejected. Because the SNU name is the same as the NODE name, if the NODE is valid, so is the SNU.

For ETO terminals, the logon process creates the terminal structure (VTRB) and must be successful before signon is attempted, therefore, before any USER or LTERM entry is created. If *any* LTERM is valid (consistent), then the logon is accepted, but any invalid LTERM is not created. If all LTERMs are invalid, then an attempt is made to create a default LTERM having the same name as the USER. If this is also rejected, then the signon is rejected. It may then be necessary for the end-user to log off, signon again with a different user descriptor, or otherwise correct the problem.

8.4.2 Resource name uniqueness

Resource name uniqueness guarantees that the same resource name will not be active on more than one IMS within the IMSplex at the same time. An LTERM named PRSNL, for example, cannot be active on both IMS1 and IMS2. Resource name uniqueness applies to the following resource types:

- Single session NODEs (static or dynamic)

- LTERMs (static or dynamic)
- USERs (including static NODE users, ETO users, and parallel session ISC subpools)
- USERIDs if the user wants to enforce single user signon (unless SGN=G, Z, or M in DFSPBxxx)

It does not apply to these resource types, which may be active on multiple IMSs concurrently:

- Transactions
- MSNAMEs
- Parallel session ISC NODEs
- APPC descriptor names

When a resource (for which the uniqueness requirement is enforced) becomes active anywhere within the IMSplex, an entry is created in the resource structure identifying the resource and its ***owner***. The owner is the IMS system on which that resource is active. If that same resource (or another resource of the same name and type) were to attempt to become active on this or another IMS in the IMSplex, it would fail. Note that a USERID may be allowed to be active on multiple IMSs at the same time by coding SGN=G, Z, or M in its DFSPBxxx PROCLIB member. This is a global (IMSplex-wide) parameter, and the first IMS to join the IMSplex will set this value in the IMSplex global entry. It then applies to all IMSs joining the IMSplex later, regardless of the value of SGN set in their DFSPBxxx PROCLIB members.

Figure 8-4 shows an example with two IMSs both having defined an LTERM resource named PRSNL. In this example, IMS1 is the first to log on (from NODE PERS1). As a result, the NODE, SNU, and LTERM resource entries are created in the resource structure (as shown) with an owner of IMS1. If a user on a different NODE (PERS2) but with the same LTERM name defined (PRSNL) were to try to log on, that logon would be rejected. This would be true even if other unique LTERMs were defined for PERS2.

Similar results would happen for ETO terminals, except that if multiple LTERMs are created and any are successful, then the signon would be successful. Like with resource type consistency, if no LTERMs are unique, then a default LTERM equal to the USER name is attempted. If that also fails, then the signon is rejected. If the USER is not unique, then the signon is rejected. No attempt is made to create a default USER.

Archived

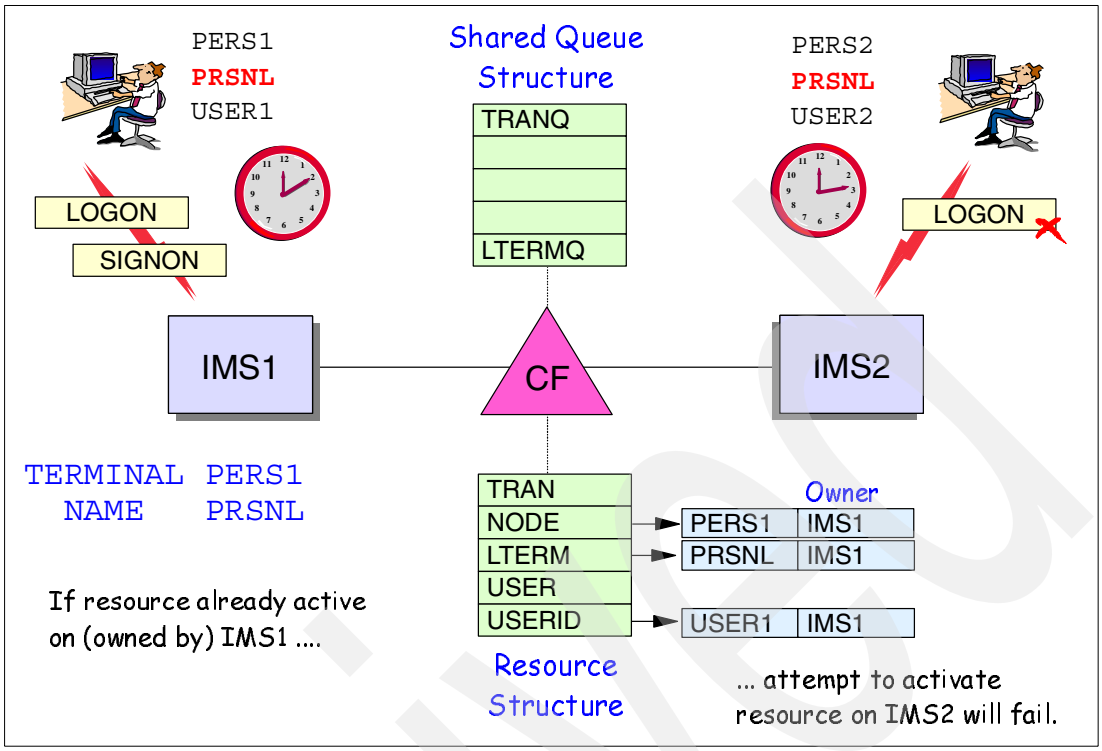


Figure 8-4 Resource name uniqueness

Note that for statically defined terminals, the USERID entry is not created until the end-user signs on. In an ETO environment, the USER (instead of the SNU), LTERM, and USERID entries are created at signon time.

8.5 Resource status

Most IMS terminal/user resources have some kind of status that can be associated with them. For example, they may be in conversation, in response mode, in TEST MFS mode, and many others. Some of this status is recoverable and some is not. Some is significant and some is not. Some is related to a command having been entered from a terminal/user, and some is related to the work a user is doing.

The distinctions between these different status types is important in understanding how sysplex terminal management deals with resource status recovery. The following paragraphs describe these distinctions. For purposes of the following discussion, and for brevity's sake, the term *user* will be used to describe the terminal/user end of an IMS session. In some cases, the status associated with this user really applies to the physical terminal.

8.5.1 Command status

Command status is that user status which is set by the entry of an IMS command. Examples are:

```
/STOP NODE | LTERM | USER
/TEST
/TEST MFS
/EXCLUSIVE NODE | USER
/ASSIGN LTERM | USER .... SAVE | NOSAVE
/LOCK LTERM | NODE
```

8.5.2 End-user status

End-user status is a status that is the result of work being done by the user. For example:

- Response mode: When a user enters a response mode transaction
- Conversational mode: When the user has entered an IMS conversational transaction
- STSN mode: Sequence numbers updated every time an input message is received from, or an output message is sent to, an STSN device (for example FINANCE, SLUTYPEP, or LUTYPE6/ISC)

8.5.3 Recoverable status

Recoverable status is that command or end-user status which, following a successful session or IMS restart, will be restored by IMS *if IMS knows what that status was*. Sometimes recoverable status may not be known to IMS at restart and so is not restored. For example, if the status was only known locally (that is, not saved in the resource structure) and if IMS is cold started, then the status would not (could not) be recovered.

This definition applies whether or not IMS is running with STM enabled. When STM is not enabled, recoverable status is saved and restored from local control blocks and log records only.

Examples of recoverable status include:

- Conversational mode
- TEST MFS mode
- LTERM assignments made with the SAVE keyword
- Fast Path response mode

Note: When a user enters a Fast Path (EMH) transaction, that terminal is put into response mode. If the session or IMS fails, when it is re-established, the terminal will be put back into response mode. See below for the description of non-recoverable full function response mode.

8.5.4 Non-recoverable status

Non-recoverable status is that command or end-user status which is maintained by IMS only as long as that user is active. In simple terms, it is whatever status is not in the recoverable category. If the session terminates, normally or abnormally, that status is discarded by IMS. If IMS fails, the status will be discarded during emergency restart. This definition applies whether or not IMS is running with STM enabled. Even when STM is enabled, this status will not be recovered.

Examples of non-recoverable status include:

- /TEST mode (except for /TEST MFS)
- /LOCKed
- /ASSIGNed LTERMs and USERs without the SAVE option
- Response mode for full function transactions

Note: This means, if a user is in response mode after entering a full function (not Fast Path) transaction, and the session (or IMS) fails, then when the user re-establishes the session, that terminal will *not* be in response mode.

Archived

8.6 Significant status

In the context of sysplex terminal management, significant status applies to recoverable status which will prevent a resource entry from being deleted from the resource structure. That is, if a resource has significant status at the time of session or IMS failure, then it will prevent that resource entry from being deleted at session or IMS termination. There are two types of significant status:

- Command significant status
- End-user significant status

8.6.1 Command significant status

There are six types of command significant status that can exist for a user in a STM environment. The command significant status is set by the entry of an IMS command and it is always global. The following sentences for each list item identify the information saved for each status.

- **/TEST MFS**

  Applies to NODEs and users; indicates that the terminal is in TEST MFS mode.

- **/STOP NODE | LTERM | USER xxx**

  Applies to NODEs, LTERMs, and users; indicates that the resource is stopped.

- **/EXCLUSIVE NODE | USER xxx**

  Applies to NODEs and users; identifies the NODE or user which is to be used exclusively for input or output from this terminal.

- **/TRACE SET ON NODE xxx**

  Applies to NODEs; indicates that the NODE is being traced.

- **/CHANGE USER xxx AUTOLOGON ... SAVE**

  Applies to ETO users; identifies the changed autologon information for that ETO user. This is not significant unless the SAVE option is specified.

- **/ASSIGN LTERM | USER xxx TO yyy SAVE**

  Applies to LTERMs and users; identifies the assignments for the user or LTERM; this is not significant unless the SAVE option is specified.

8.6.2 End-user significant status

There are three types of end-user significant status applicable to STM:

- Conversational status

  When a user enters a conversational transaction, that user is said to be in conversation with IMS. Recoverable information about that conversation includes a conversation ID and, of course, a transaction code. Conversation IDs and transaction codes for HELD conversations are also significant and are saved. A conversational input in-progress flag indicates that a conversational transaction has been entered and queued, but that no output has yet been delivered.

- Fast Path response mode

  When a user enters a Fast Path transaction (one which is scheduled into an EMH region), that user is in Fast Path response mode. Note that, while this is considered significant, full function response mode is not. Recoverable information consists of a Fast Path input

Archived

in-progress flag that indicates that a user has entered a Fast Path transaction and no output response has yet been delivered.

- STSN status

  STSN devices are those that use the *Set and Test Sequence Number* instruction to maintain accountability for input and output messages. Each time an input message arrives from a STSN device, the input sequence number is increased. A similar function exists for output messages. STSN devices include terminal defined to IMS (statically or dynamically) as FINANCE, SLUTYPEP, LUTYPE6 (ISC). The recoverable information consists of the input and output sequence numbers.

For each of these, the installation may choose whether or not to save the recoverable information.

8.7 Resource status recovery

Resource status recovery is the most complex of the STM functions. The status is rrecoverable, if it is known to IMS when the resource reconnects. That means that the status has to be saved in the resource structure. The resource can be node, LTERM, or user entry in the structure.

8.7.1 Significant status

When the session terminates, IMS will not delete the resource entry from the resource structure, if it has:

- End-user significant status
  - Conversation, Fast Path response mode, STSN
- Command significant status
  - STOP, EXC, TEST MFS, TRACE
  - ASSIGN or CHANGE USER with SAVE keyword

Note that not all recoverable status is significant, for example LTERM assignments made without the SAVE keyword. The user may elect, for each resource, whether or not end-user status will be saved in the resource entry.

8.7.2 Status recovery mode (SRM)

Section 8.6.2, “End-user significant status” on page 187 described the three types of end-user significant status.

Status recovery mode (SRM) identifies if, and how, that status is to be maintained. There are three choices, SRM=GLOBAL, LOCAL or NONE.

SRMDEF=GLOBAL End-user significant status is to be maintained on the resource structure. Status is also maintained locally while the resource is active. When the resource becomes inactive, local status is deleted.

It is recoverable across a session termination/restart or IMS restart from information on the resource structure on any IMS in the IMSplex.

SRMDEF=LOCAL End-user significant status is to be maintained in local IMS control blocks only. When the resource becomes inactive, its status is maintained in local control blocks and in the IMS system checkpoint log records.

The status is recoverable only on the local system from the local control blocks or, in the case of an IMS restart, from the system checkpoint records.

SRMDEF=NONE End-user significant status is to be maintained in local IMS control blocks only as long as the resource is active on that IMS. When the resource is no longer active, for any reason, that status is deleted.

It is not recoverable on any IMS.

Note: When a resource structure exists, resource entries with their SRM is maintained in that structure regardless of the SRM setting. That is, even if SRM=LOCAL or NONE, the SRM setting will be kept in the resource entry in the structure. When no resource structure exists, the SRM setting is kept in the local control blocks.

There is no SRM option for command significant status. It is always maintained globally if there is a resource structure, and always maintained locally if there is no structure.

8.7.3 Status recoverability (RCVYxxxx)

In addition to SRM, there is another parameter that is more granular — recoverability, set by the RCVYxxxx parameter. This parameter allows the user to determine whether specific end-user statuses should be recoverable.

RCVYCONV=YES | NO When set to YES, information required to recover a conversation will be kept across a session termination/restart and IMS termination/restart. Where this information is kept depends on the status recovery mode.

When set to NO, conversational information is kept locally and only as long as the session is active. When the session terminates, even without explicitly exiting the conversation (/EXIT command), the conversational information will be discarded by IMS and the conversation will be exited.

RCVYFP=YES | NO When set to YES, Fast Path (EMH) response mode is recoverable. This means that if a session terminates (or IMS terminates) while the terminal is in Fast Path response mode, when that session is re-established, it will be returned to response mode. When the response is available, it can be delivered to the terminal.

When set to NO, Fast Path response mode is not restored after session termination and restart. Note that for Fast Path transactions, when RCVYFP=NO, the Fast Path response message is also not recoverable. When the response is discovered by IMS, it will be discarded.

RCVYSTSN=YES | NO When set to YES, STSN sequence numbers for both input and output messages will be saved and are recoverable.

When set to NO, they are not recoverable. When a session terminates and then is restarted, the sequence numbers will revert to zero (cold start). This may have particular significance for ETO STSN devices.

- When a resource structure exists, the RCVYxxxx settings are maintained in that structure regardless of the SRM setting. That is, even if SRM=LOCAL or NONE, the RCVYxxxx settings will be kept in the resource entry in the structure.

Archived

- When no resource structure exists, these settings are kept in the local control blocks. Note that the default for all is YES.

System defaults for these are set in PROCLIB member DFSCGxxx and can be overridden for each resource at logon or signon time by the Logon Exit or the Signon Exit.

8.8 Status recovery examples

These are status recovery examples for SRM=GLOBAL and SRM=LOCAL.

8.8.1 Status recovery (SRM=GLOBAL)

This is an example of STM Global status recovery. Figure 8-5 shows two IMSs (IMS1 and IMS2) which each have NODEA and LTERMA defined. These IMSs are part of a data sharing group, a shared queues group, and a VTAM Generic Resources group (GRSNAME=IMSX).

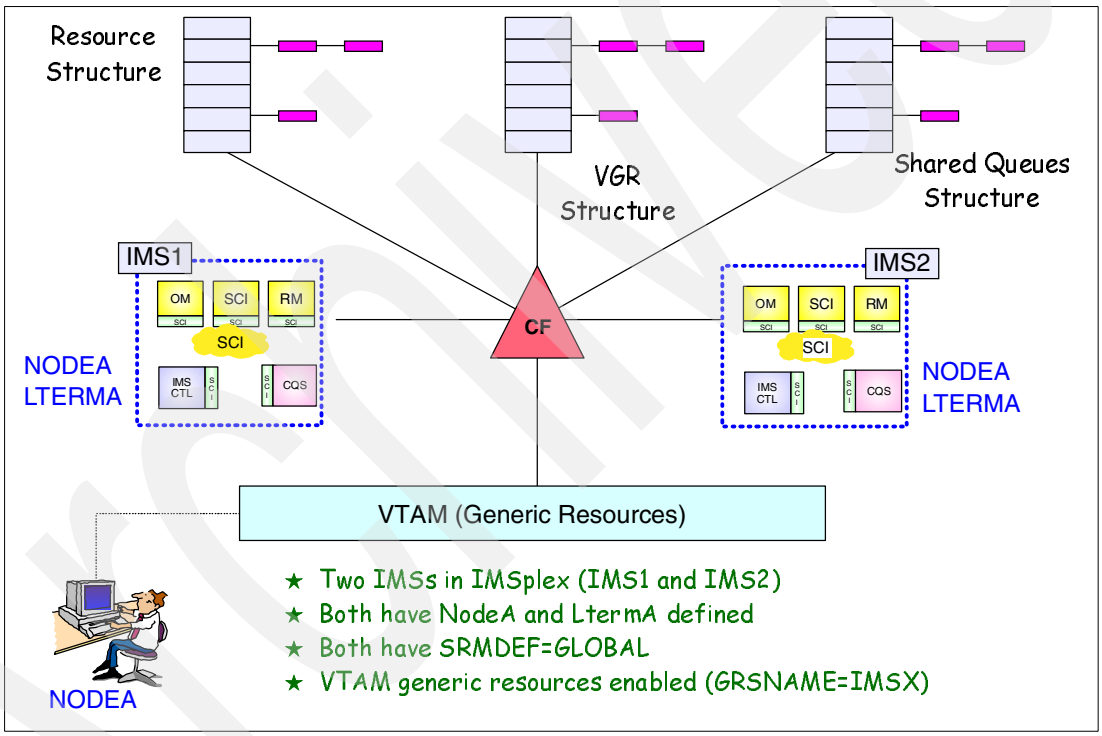


Figure 8-5 Status recovery example

User log on

In Figure 8-6, a user logs on from NodeA using the generic resource name IMSX. VTAM will check the VGR structure and determine that this node does not currently have an affinity for any application in the IMSX generic resource group. It selects one of the active IMSs to route the logon request to. In this example, it is IMS1.

IMS1 creates entries in the resource structure for NodeA, Static-Node-UserA (also named NodeA), and LTERMA. If the user signed on, and SGN was not equal to M, then an entry for the userid would also be created. All have OWNER=IMS1 and SRM=GLOBAL. If any of these resources already exist, then the create would fail (“resource name uniqueness”) and IMS1 would determine the reason why. If it is because that user is currently active on (owned by) another IMS, then logon would be rejected. In this case the logon is successful. VGR affinity

is set to IMS1 and, because SRM=GLOBAL, IMS1 will tell VTAM to manage the VGR affinity (session level affinity is enabled only with z/OS 1.2 and later). This means, VTAM should delete the VGR affinity when the session terminates.

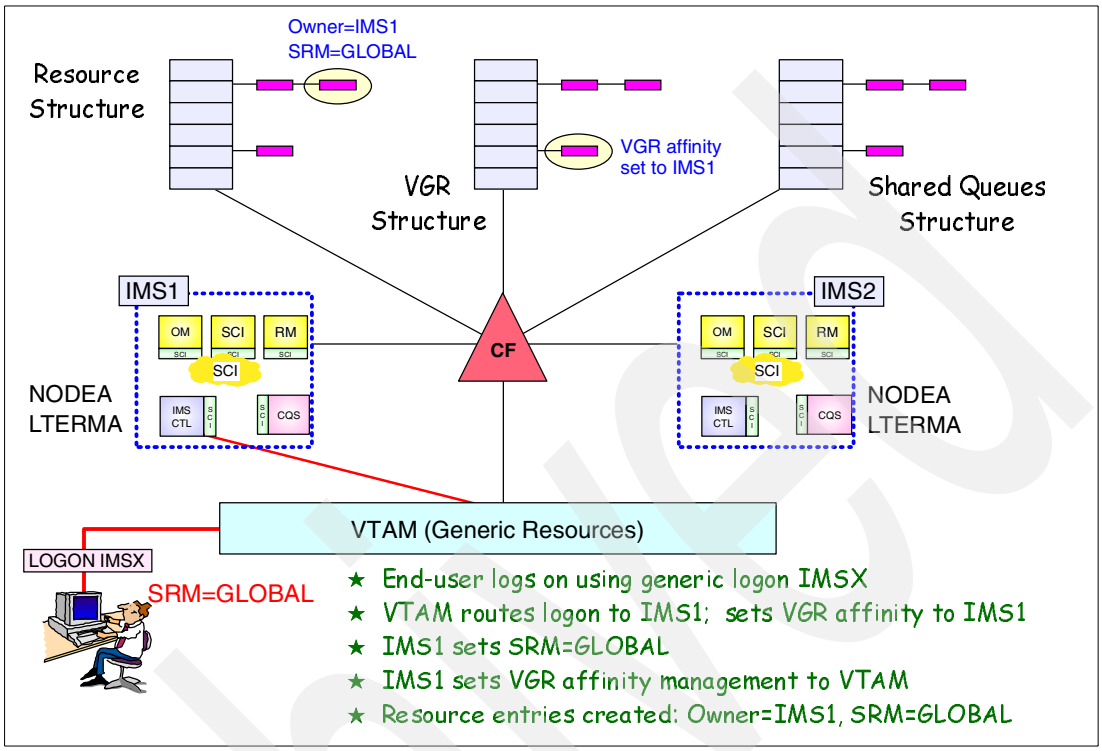


Figure 8-6 Status recovery (SRM=GLOBAL)

Enter a conversational transaction

In Figure 8-7,the user then enters a conversational transaction. IMS1 will do several things:

- Create a new conversation by creating a CCB for that user. This user is now in conversation status on IMS1.
- Put the SPA+Input-Transaction on the transaction ready queue.
- Update the Static-Node-UserA resource in the resource structure to indicate the conversation ID and set a flag indicating `input conversational transaction in progress`. The conversational status is also maintained in IMS1s control blocks.

This continues for several iterations of the conversation, with the CONV-IP flag being set and reset for each transaction.

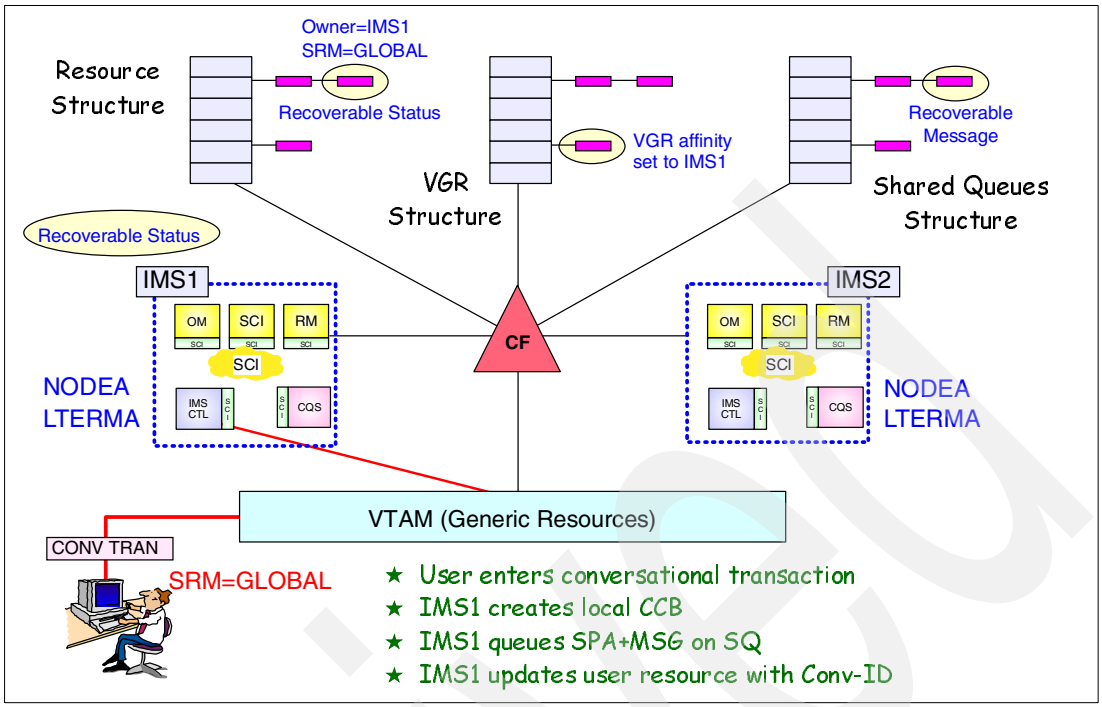


Figure 8-7 Status recovery (SRM=GLOBAL)

IMS failure

Now, in Figure 8-8, IMS fails while the user is preparing to enter the next transaction. The previous SPA+OUT-MSG are still on the shared queue (LOCKQ). VTAM will delete the VGR affinity.

IMS2 is informed that IMS1 has failed and initiates some clean-up activities.

- Queries RM for entries owned by IMS1.
- Finds entries for NodeA, Static Node UserA, and LTERMA. Entries indicate that SRM=GLOBAL and than the user is in a conversation (end-user significant status).
- Since end-user status is in the structure and available to any other IMS, IMS2 resets ownership to “not owned” meaning user is allowed to log on to any IMS.
- IMS2 will also requeue the SPA+OUT-MSG locked by IMS1 to the LTERM ready queue to make it available to any IMS that the user logs on to. (Locked messages are only available to the IMS that locked them. LRQ messages are available to any IMS.)

Archive

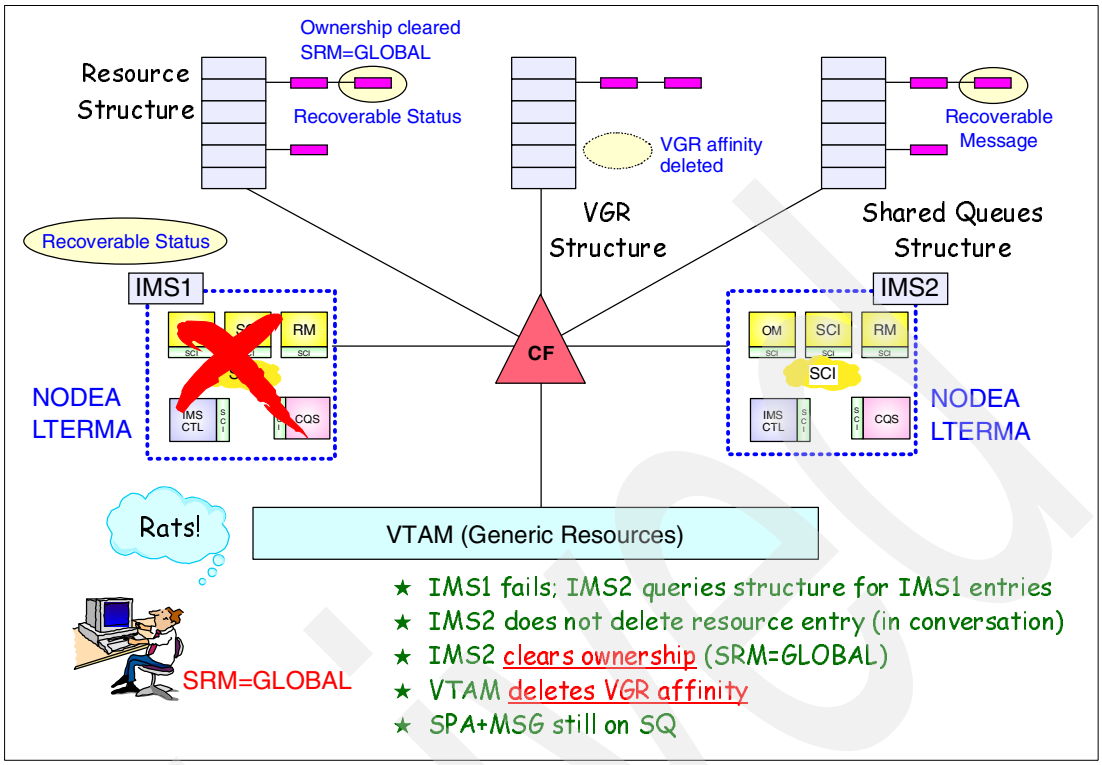


Figure 8-8 Status recovery (SRM=GLOBAL)

User log on to another IMS

Then, in Figure 8-9, the user decides not to wait for IMS1 to be restarted and logs back on using IMSX. VTAM checks the affinity table, sees not affinity for NodeA, and routes the logon request to the only active IMS in the group — IMS2.

IMS2 tries to create resource entries in the structure and fails because they are already (still) there. IMS2 checks and discovers that the resource has significant status but is not owned. IMS2 accepts the logon and sets ownership to IMS2. IMS2 also relocks the SPA+OUT-MSG that it found on the LTERM ready queue (put there during cleanup processing after IMS1 failed).

The user can continue the conversation by holding and then releasing the conversation. This causes IMS2 to go back to the shared queue, retrieve the last SPA+OUT-MSG from the LOCKQ, and resend the last output to the user terminal.

The user then continues the conversation as normal.

Archive

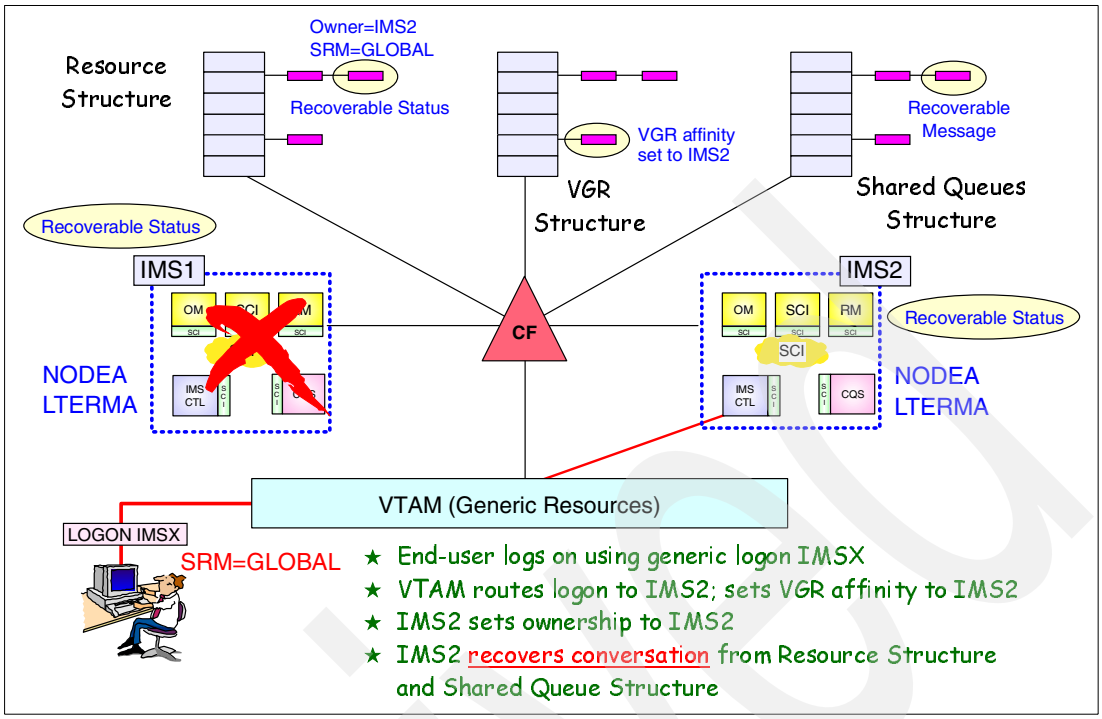


Figure 8-9 Status recovery (SRM=GLOBAL)

Restart IMS failed

In Figure 8-10, when IMS1 is restarted, its control blocks will indicate that the user was in a conversation with SRM=GLOBAL.

Since the status was on the structure at the time of failure, IMS1 knows that the user may have logged on to another IMS and continued the conversation. IMS1 deletes his local status.

Even if the user had not logged on to another IMS, the status is still on the structure and IMS1 can safely delete the local status.

Archive

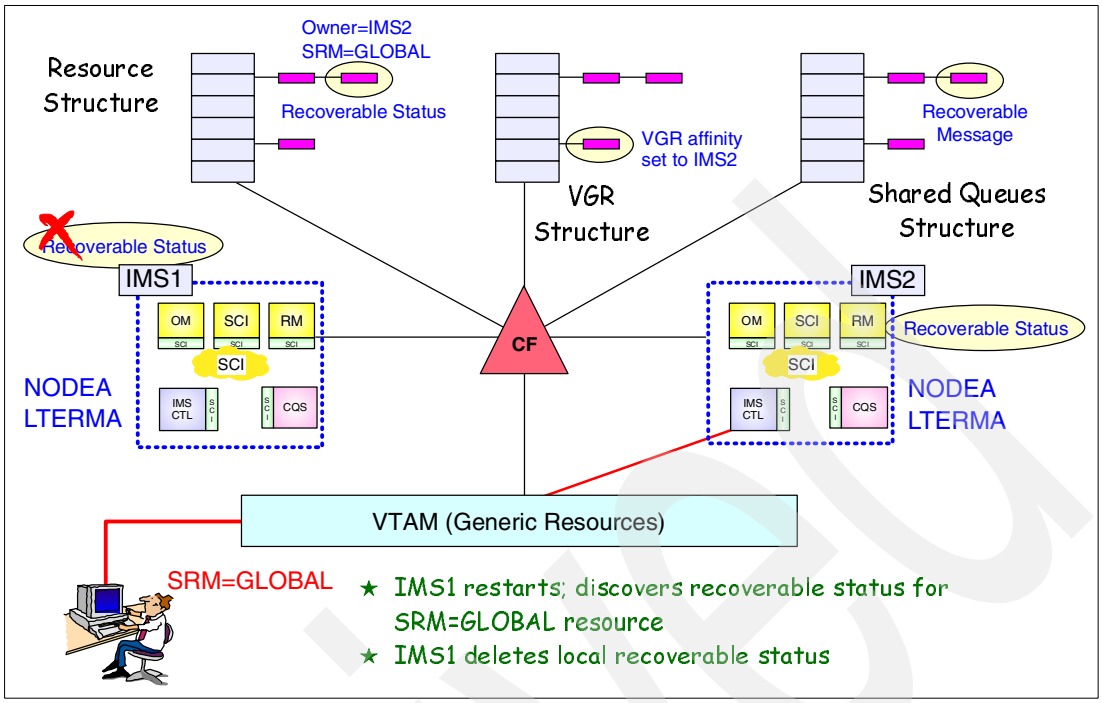


Figure 8-10 Status recovery (SRM=GLOBAL)

8.8.2 Status recovery (SRM=LOCAL)

In Figure 8-11, SRM=LOCAL, there are a few differences:

- The VTAM GR affinity is set to IMS-managed instead of VTAM-managed
- The SRM value in the resource entry is set to LOCAL

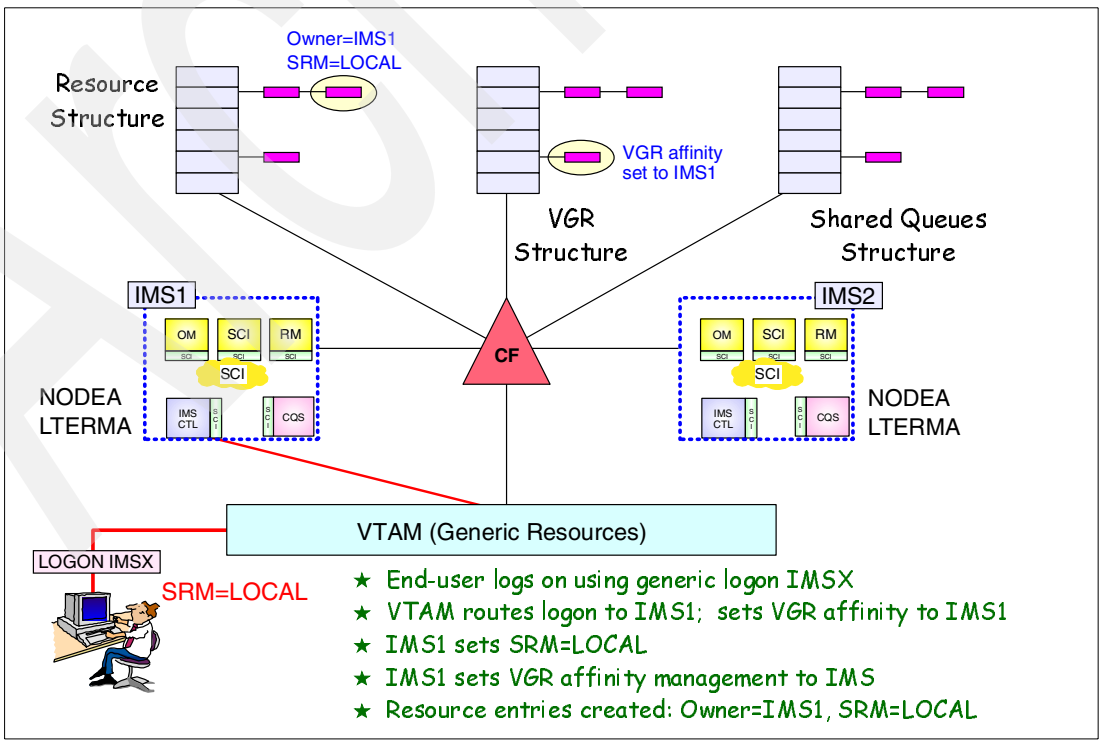


Figure 8-11 Status recovery (SRM=LOCAL)

Enter a conversational transaction

Figure 8-12 shows that when the user enters a conversational transaction, the conversational status is *not* maintained in the resource structure. It is known only to IMS1.

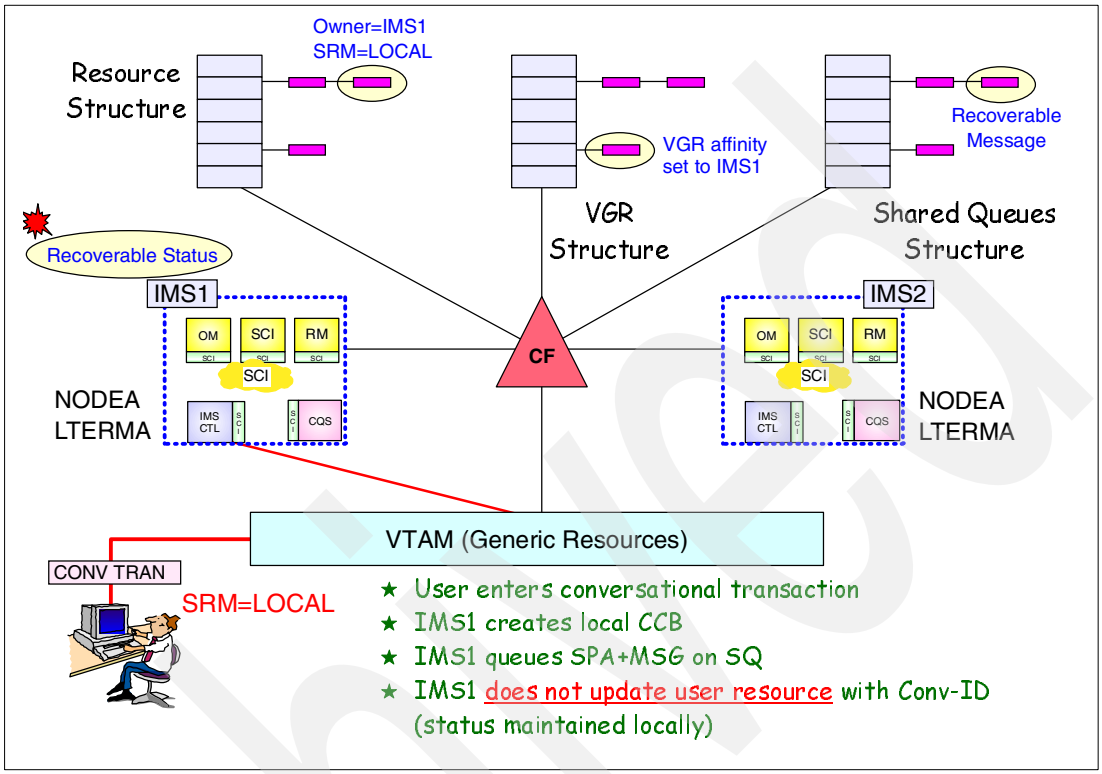


Figure 8-12 Status recovery (SRM=LOCAL)

IMS failure

In Figure 8-13, when IMS1 fails and IMS does cleanup, IMS2 sees that the SRM=LOCAL for the resource and does not know whether there is any end-user status. So, IMS2 does not delete the resource entry, and IMS2 does *not* clear ownership.

Also, the VGR affinity, which is IMS managed, is *not* cleared by IMS1 since the terminal is in conversational mode with SRM=LOCAL. IMS1 is the only IMS that knows anything about the conversation.

Archived

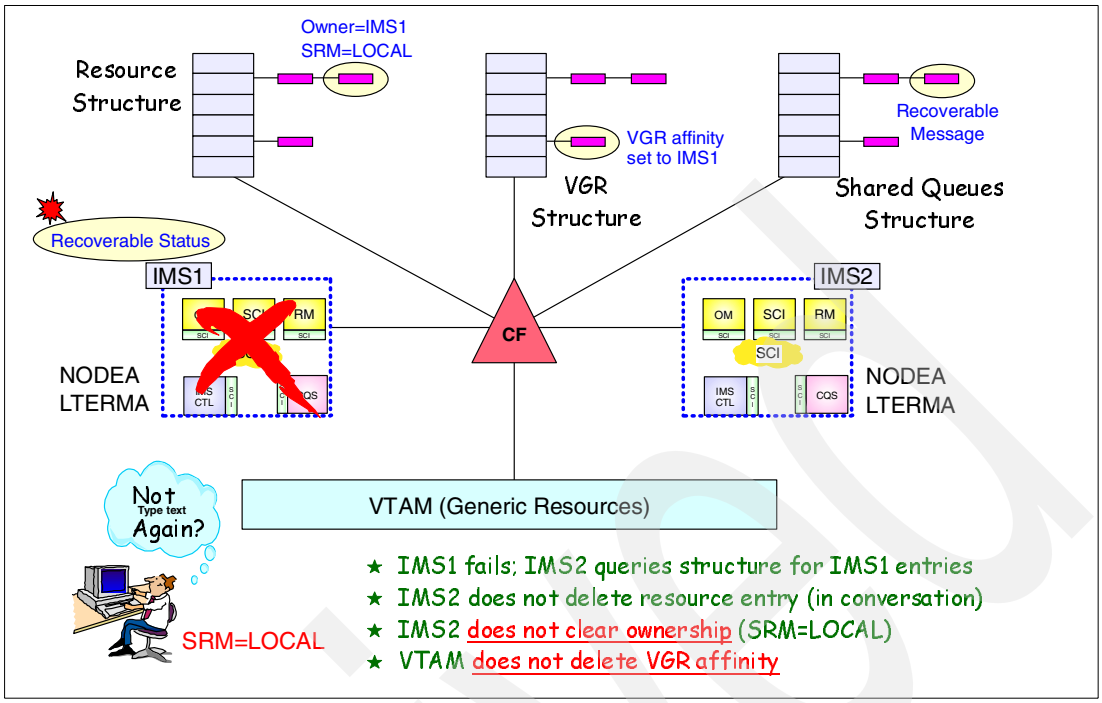


Figure 8-13 Status recovery (SRM=LOCAL)

User log on to another IMS

In Figure 8-14, if the user tries to log on again using IMSX, VTAM will discover that NodeA has an affinity for IMS1. Since IMS1 is not active, the logon will fail. The user must wait for IMS1 to restart.

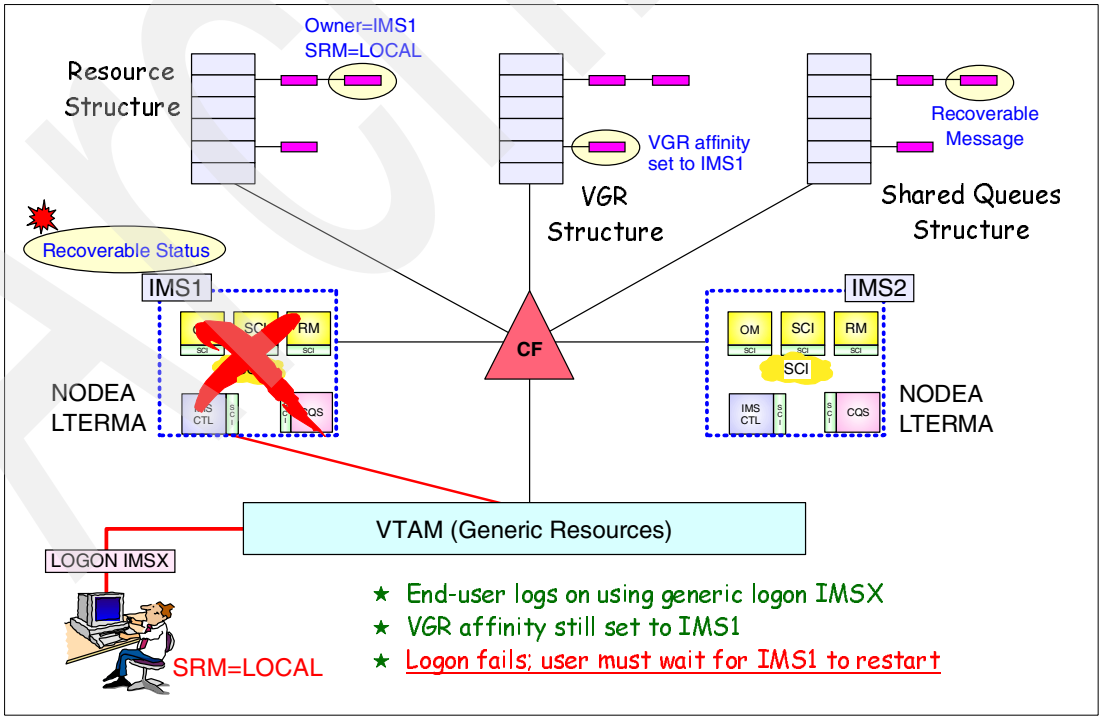


Figure 8-14 Status recovery (SRM=LOCAL)

Logon rejected

Then, in Figure 8-15, if the user gets tired of waiting (maybe the user knows that IMS1 will be down for an extended period of time) and logs on directly to IMS2, IMS2 will check the NodeA entry and find that NodeA is owned by IMS1 and reject the logon.

Too bad; *however*, IMS2 can steal the node if the logon exit (DFSLGNX0) says it is OK. A clever user shop will write a logon exit to recognize some user data entered at logon as a request to steal the node. For example, the user data might say, "`I really mean it!`". If this happens, then the conversation is not recoverable. When IMS1 is restarted and finds the resource with recoverable status and SRM=LOCAL, IMS1 will check the resource structure and discover that NodeA/UserA/etc. are no longer owned by IMS1 and delete the status. That conversation is gone.

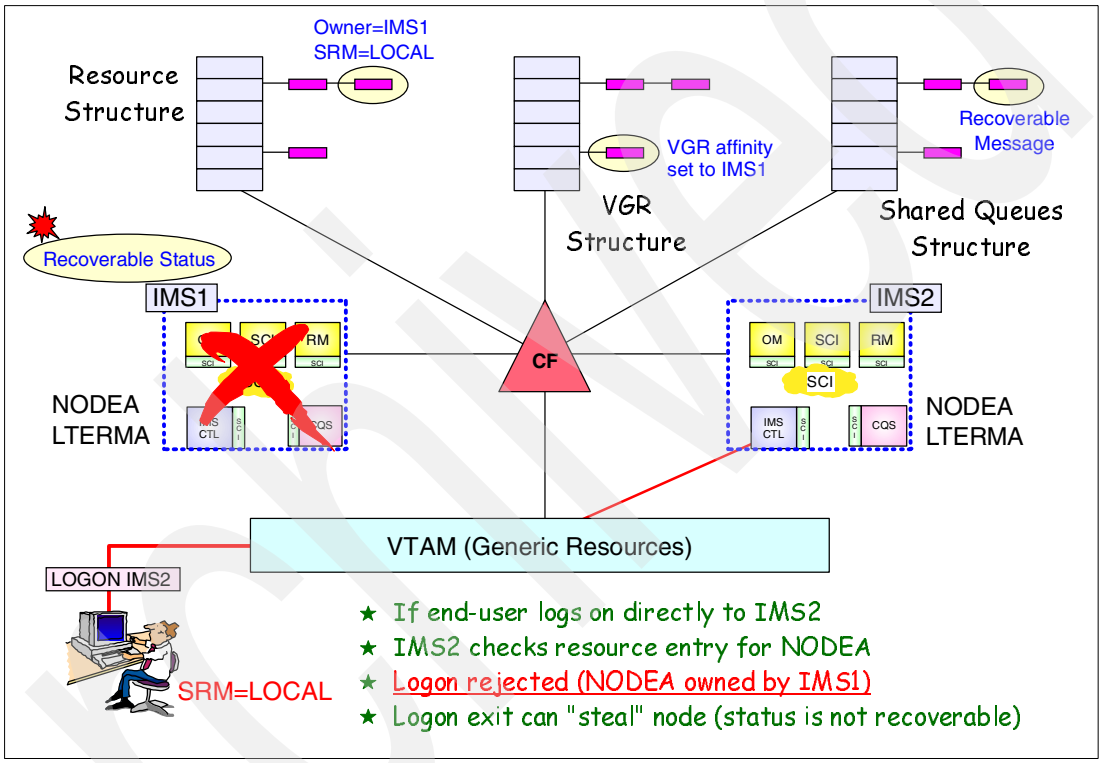


Figure 8-15 Status recovery (SRM=LOCAL)

8.8.3 Status recovery (SRM=NONE)

SRM=NONE (or RCVYxxxx=NO) means that end-user status is *not* to be recovered, even locally. When a session terminates with SRM=NONE, all end-user status is deleted. See Figure 8-16.

This might be useful with ETO, which will not delete control blocks if there is any status. For example, an ETO STSN device ALWAYS has status (always has a sequence number). SRM=NONE for these devices would delete the status, and then the resources themselves could be deleted.

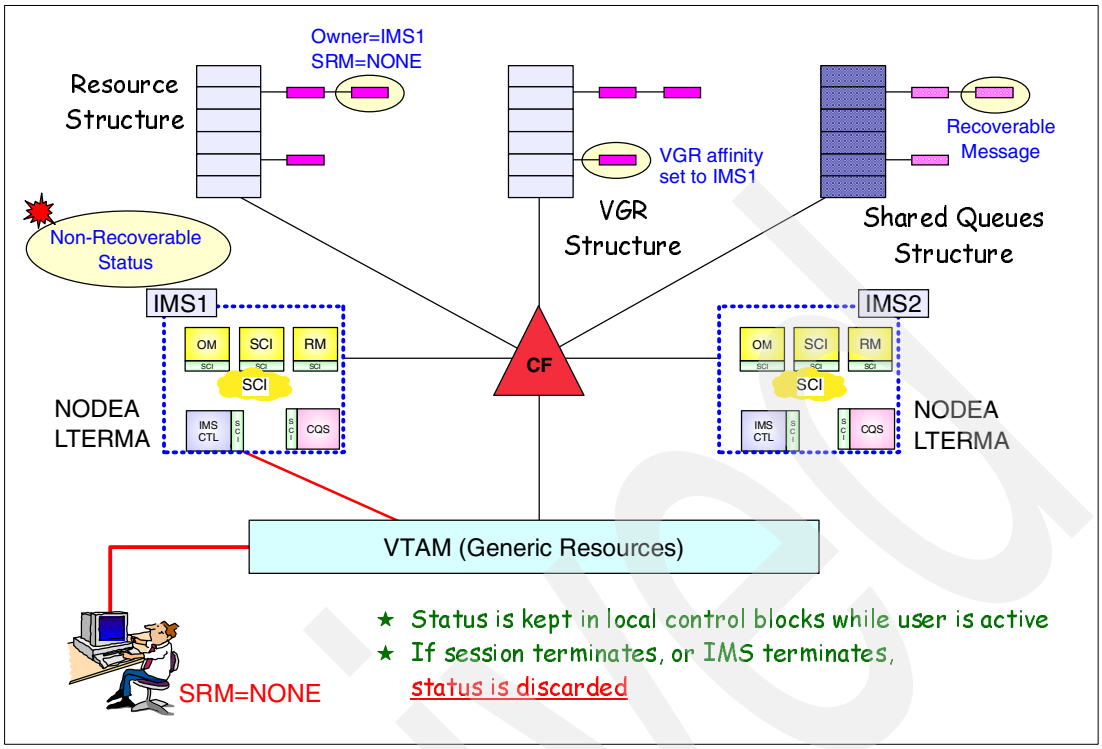


Figure 8-16 Status recovery (SRM=LOCAL)

8.9 Ownership and affinities

There are two types of affinities that are possible with STM:

- VTAM Generic Resources affinity is used by VTAM to route terminal logon requests to a particular IMS which holds significant status for that resource. Refer to 11.4, “Generic resources affinity” on page 242 for more information.
- Resource Manager (RM) affinity is used by STM to prevent a user from logging on to one IMS when its end-user significant status may exist only in another (SRM=LOCAL).

8.9.1 Resource ownership and RM affinity

It has been mentioned several times that when a resource becomes active, a resource entry is created in the resource structure and ***ownership*** is set to that IMS. For example, when a NODE logs on to IMS1, a resource entry for that NODE is created in the resource structure and the ownership of that NODE is set to IMS1. That NODE is said to have an ***RM affinity*** for IMS1. The two terms are synonymous.

RM affinity (ownership) applies to NODEs, (Static NODE) Users, and LTERMs. It also applies to Userids if single signon is being enforced (that is, if SGN is not equal to G, Z, or M). When a resource is owned by one IMS, it cannot become active on another IMS. Examples later on will show how this can happen and how IMS reacts.

8.10 Resources and the resource structure

The resource structure is used to keep information about the IMSplex and its members (IMS, CQS, and the RM), about global IMSplex processes (such as the global online change process), and about individual terminal and user resources. This section describes those resource entries used for the sysplex terminal management function, including when they are created, what data they contain, how they are used, and when they are deleted.

8.10.1 Resource structure components and characteristics

When a connector connects to a structure defined in the CFRM policy, that connector identifies the type of structure (list, cache, or lock) and its characteristics. For the resource structure (and for the shared queues structures), the Common Queue Server (CQS) is the connector. Some of the characteristics of the resource structure specified by CQS are:

- How many list headers (LH)

  The resource structure is allocated with 192 list headers. A list header is an anchor point for list entries.

- What type of list entry (LE)

  Each list entry represents one IMSplex or terminal/user resource on the structure. Each list entry is composed of several parts.

  - List entry controls (LEC)

    Header for each list entry. The LEC contains information about list entry, such as:

    - Entry Key (ENTRYKEY)

      For RM, the entry key is the “resource type” plus “resource name” (for example, x’01’ + trancode). When RM asks CQS to create an entry, CQS uses the resource type to identify a range of 11 list headers, and the resource name to determine which LH in that range on which to place the list entry. The entry key does not have to be unique, although for the resource structure, RM keeps them unique.

    - User-managed list entry ID (LEID)

      The LEID is the “name type” plus “resource name” (for example, x’01’ plus LTERM name). The LEID must be unique within the structure. STM uses this characteristic to enforce resource type consistency.

    - Version

      Number increased by one each time the resource entry is updated. This is used by RM to ensure that updates to the structure are serialized.

    - Other

      There are several other fields in the list entry controls that are used by CQS, XES or the CFCC to manage the structure. They are not of interest for this discussion.

  - Adjunct area (ADJ)

    This is optional for list structures in general, but for the resource structure, every list entry has one. The adjunct area is 64 bytes and contains the resource “owner” and a small amount (up to 24 bytes) of client data (DATA1).

  - Data elements (DE)

    Data elements are also optional for a list structure, but when defined, are used to contain user data that is too large to fit in the adjunct area. The resource structure is defined with 512 byte data elements. Each list entry may have zero or more data

elements, containing recoverable resource data (DATA2) that is too large to fit in DATA1.

- Other characteristics of the resource structure include:
  - Structure is persistent

    Will not be deleted when all connectors are gone
  - Connection is persistent

    Will not be deleted if connector abends
  - Supports (new) structure management enhancements
    - Alter and autoalter
    - System managed rebuild
    - System managed duplexing

Figure 8-17 is a simple illustration of the resource list structure. Each resource entry on the structure is composed of one list entry control (LEC), one adjunct area (ADJ) and zero or more data elements (DE). Whether a data element is needed depends on how much recoverable data is kept for that resource.

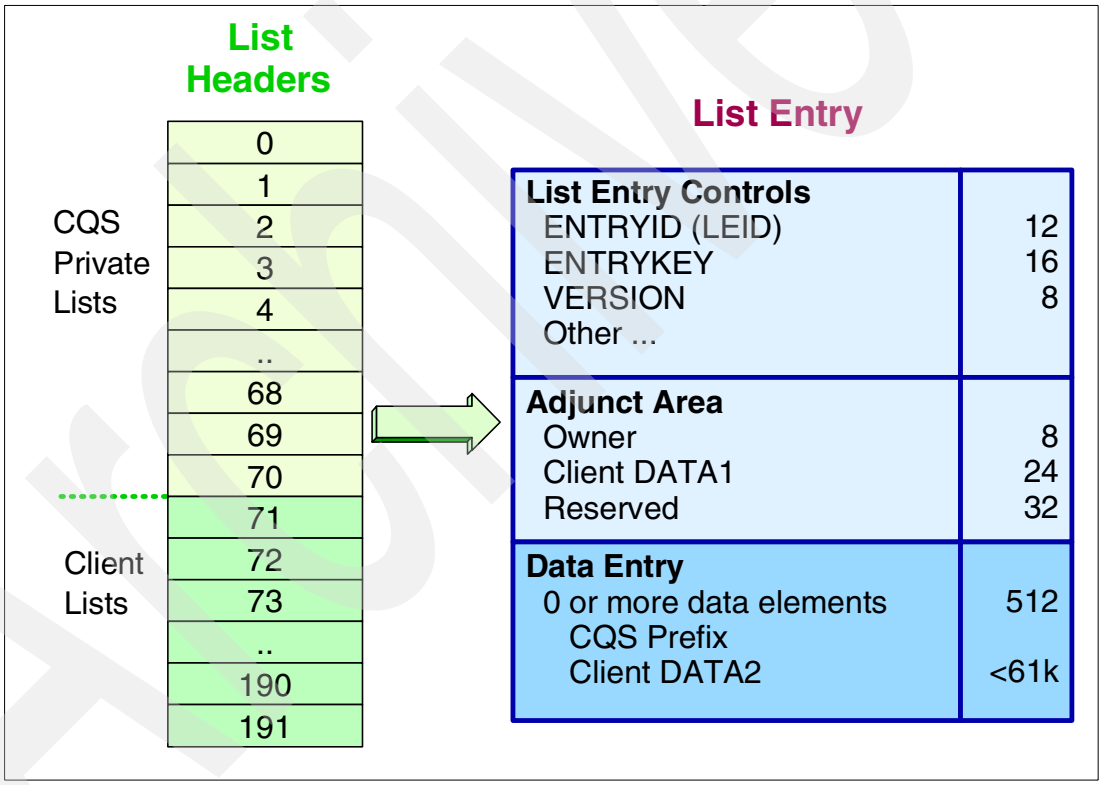


Figure 8-17 Resource structure components

CQS reserves list headers 0-70 for its own use. Presently, the only LH used by CQS in the resource structure is list header zero (LH0). LH71 through LH191 are available for use by the CQS client (Resource Manager). The ENTRYKEY is used by CQS to determine which list header any particular resource entry (list entry) is to be placed.

Archived

8.11 Resource entries in the resource structure

Figure 8-18 identifies the IMSplex resources, resource types, and name types. In each case, the resource entry identifies the existence of a resource in the IMSplex, whether or not it is active, and other information about that resource, some of which is recoverable status. The ENTRYKEY (resource type + resource name) identifies the resource. Information about that resource is kept in the adjunct area (OWNER and DATA1) and, when DATA1 is not large enough, in one or more data elements (DATA2).

Resource Type	Name Type	Definition	Meaning
01	01	Transaction	Application program message destination
02	01	Lterm	Logical terminal message destination
03	01	Msname	MSC logical path name message destination
04	04	User	Person signed on to an ETO terminal or parallel session ISC SUBPOOL
05	05	Node	VTAM terminal (not APPC)
10, 21, 32, (+11),...,252	10, 21, 32, (+11), ..., 252	Vendor and Customer	Reserved for vendor & customer use
11	11	IMSplex	IMS global information
12	01	CPI-C Transaction	CPI-C transaction
14	01	APPC Descriptor	APPC descriptor name - message destination
15	15	Userid	User ID for security checking
26	26	Static Node User	User of a statically defined terminal or single session ISC node
242	242	RM Process	RM global process status
253	253	RM Global	RM global information

Figure 8-18 STM resource types and name types

8.11.1 IMSplex entries

Several of the resource entries apply to the IMSplex itself. Some of these are global (only one entry for the IMSplex) and some are local (multiple entries). Only two of these entries play a significant role in STM. They are the DFSSTMGBL entry (one per IMSplex) and the DFSSTMLimsid entries (one for each IMS). The others are not discussed here.

DFSSTMGBL

Contains the multiple signon indicator (SGN=). This entry is created when the first IMS joins the IMSplex. The signon indicator cannot be changed by the second or succeeding IMSs. This is used to determine whether to enforce resource name uniqueness for Userids. To change this value for the IMSplex, all IMSs must be shut down. Then the first to rejoin can reset this value. The entry is never deleted.

DFSSTMLimsid

Created for each IMS that joins the IMSplex. When the entry is created, or when IMS rejoins the IMSplex, then the ownership of this resource is set to that IMS. When IMS shuts down normally, the ownership is released. If IMS fails, then it can't release ownership, so the entry remains with its ownership set to the failed IMS. This is used by other IMS systems in the IMSplex to determine whether an IMS system has failed, and is important for cleanup activities after an IMS failure. This entry is deleted only if IMS is shut down with the LEAVEPLEX keyword.

```
/CHE FREEZE LEAVEPLEX
```

When XRF is enabled, both IMSs (active and alternate) have entries. They contain a flag indicating whether this IMS instance is an XRF active or alternate system, and the RSENAME. The RSENAME is used as the "owner" of a terminal resource instead of the IMSID. If the alternate must take over, then it "owns" those resources which had been owned by the old active IMS.

8.11.2 Sysplex terminal entries

Most of the entries in a resource structure represent the STM-managed resources associated with the terminal/user environment. These entries are used to identify a resource and to maintain information about the status of that resource. Although these entries are always created when they are activated (for example, when a NODE logs on), their deletion is usually determined by the existence of significant status and the values of SRM and RCVYxxxx.

The following paragraphs identify each type of resource and some of the information about these resources.

Static transaction entry

Represents a transaction code defined statically to IMS during the system generation process. One is also created for transactions dynamically defined by the Output Creation Exit (DFSINSX0).

- Entry key = x'01' + transaction code; LEID = x'01' + transaction code
- When created
  - During IMS initialization
  - When added by global or local online change
- Usage and comments
  - Enforce resource type consistency
  - No recoverable information
- When deleted
  - Never deleted; must delete (SETXCF FORCE) structure and create new one to delete a transaction

CPI-C driven transaction entry

Represents a transaction code defined in APPC TP_PROFILE and entered from an APPC device. The transaction code is used to schedule a dependent region program to process CPI-C driven transaction.

- Entry key = x'0C' + transaction code; LEID = x'01' + transaction code
- When created
  - When first CPI-C transaction driven on any IMS

- Usage
  - Enforce resource type consistency
  - The IMSID of each IMS where the transaction is driven is added to DATA1 and then, if more than two IMSs, to DATA2.
  - When IMS terminates normally, it removes its IMSID from this entry. When IMS terminates abnormally, another IMS is informed (by XCF) and removes the IMSID.
  - No recoverable information
- When deleted
  - When all IMSIDs in the resource entry are gone, the last IMS deletes the resource.

NODE entry

Represents a static or dynamic (ETO) NODE.

- Entry key = x'05' + NODEname; LEID = x'05' + NODEname
- When created
  - When NODE first becomes active on any IMS. It will also be created when a command sets significant status (for example, /STOP NODE). If the NODE does not exist when this command is entered, it will be created with the "stopped" status.
- Usage and comments
  - Enforce resource name uniqueness for single session VTAM NODEs. This is not enforced for parallel session ISC NODEs.
  - Maintain recoverable status related to the NODE (for example, NODE is stopped)
  - Used, along with (Static NODE) User entry, to restore recoverable status across session termination/restart (including IMS termination/restart)
- When deleted
  - When NODE is no longer active and has no significant status

User entry

Represents an ETO user or a parallel session ISC SUBPOOL.

- Entry key = x'04' + username (subpool name); LEID = x'04' username
- When created
  - For ETO, when user structure (SPQB) is created at signon time.
  - For parallel session ISC, when parallel session (subpool) becomes active. There will (probably) be multiple user entries for each NODE.
  - When command sets significant status (for example, **/STOP USER**).
- Usage and comments
  - Enforce resource name uniqueness
  - Maintain user-related recovery information. Most significant status, such as conversational, Fast Path, and command status is kept here. Also, this entry contains "segments" in DATA2 with information about each assigned LTERM and its status. For example, a **/STOP LTERM** command would set significant status in the LTERM segment of this entry, not in the LTERM entry.
  - Used, along with NODE entry, to restore recoverable status across session termination/restart (including IMS termination/restart)

- When deleted
  - When the user is no longer active and contains no significant status. Note that, when SRM=LOCAL and IMS fails, surviving IMSs will not know whether there is significant status for this entry in the failed (local) IMS, so the resource will not be deleted.

Static NODE user entry

Represents the user associated with a static NODE. This “resource” does not exist in the IMS environment (that is, no definitions or control blocks). It is “invented” by STM to provide a similar construct for static NODEs, ETO NODEs, and parallel session ISC NODEs and their users.

- Entry key = x’1A’ + NODEname; LEID = x’1A’ + NODEname
- When created
  - Same time the NODE is created
- Usage and comments
  - This entry has the same usage and content as the User entry described above. One difference between this entry and the User entry is that this entry cannot exist without the NODE entry. For ETO, the User entry can exist independently of the NODE.
- When deleted
  - Same time the NODE is deleted

LTERM entry

Represents a static or dynamic logical terminal (LTERM)

- Entry key = x’02’ + LTERMname; LEID = x’01’ + LTERMname
- When created
  - For static terminals, the same time the NODE is created
  - For ETO terminals and ISC sessions, the same time the USER is created
- Usage and comments
  - It is used only to enforce resource type consistency and resource name uniqueness.
  - Contains no recovery related information. Any significant status for an LTERM is kept in the associated (Static NODE) User entry.
- When deleted
  - For static LTERMs, deleted when the NODE and Static NODE User entries are deleted
  - For ETO LTERMs, or LTERMs assigned to parallel session ISC NODEs, deleted when the User entry deleted

USERID entry

Represents a signed on user.

- Entry key: x’0F’ + USERID; LEID = x’0F’ + USERID
- When created
  - When user signs on, if single signon is being enforced; otherwise, not created
- Usage and comments
  - Enforce resource name uniqueness when single signon specified
  - There is no recoverable information
- When deleted
  - When user is no longer signed on; even if (Static NODE) User has significant status

Archived

MSNAME entry

Represents an MSC logical link as defined on MSNAME macro.

- Entry key = x'03' + MSNAME; LEID = x'01' + MSNAME
- When created
  - During IMS initialization
- Usage and comments
  - Enforce resource type consistency. Resource name uniqueness is not enforced for MSNAMEs. They can be simultaneously active on multiple IMSs in the IMSplex.
  - There is no recoverable information.
- When deleted
  - Never deleted; must delete (SETXCF FORCE) structure and create new one to delete a MSNAME.

APPC descriptor name

Represents an APPC descriptor name defined in DFS62DTx. This name is used by applications on the CNHG call to direct output to an APPC destination. It is, therefore, one of the "message destination" name types.

- Entry key = x'0E' + descriptor name; LEID = x'01' + descriptor name
- When created
  - During IMS initialization for each descriptor in DFS62DTx; also created when descriptors are dynamically added using /START L62DESC x command.
- Usage and comments
  - Enforce resource type consistency. Resource name uniqueness is not enforced for this resource.
  - As each IMS is initialized, its IMSID is added to DATA1 or to DATA2 (DATA1 can accommodate two IMSIDs).
  - When IMS terminates normally, it removes its IMSID from this entry; when IMS terminates abnormally, a surviving IMS is informed (by XCF) and removes the failing IMS's IMSID from the entry.
  - No recoverable information.
- When deleted
  - When all IMSIDs have been deleted from the resource entry.

Archived

9

Other functions utilizing CSL

CSL enables new functionality in the areas of operations and resource management and the communications between the components. Here is a list of the first functions utilizing the CSL that were introduced with IMS Version 8:

- Sysplex terminal management (STM)
- Global online change
- Single point of control (SPOC)
- Operation Manager user interface
- Automatic RECON Loss Notification (ARLN)
- Language Environment (LE) dynamic run time options

STM was discussed in the previous chapter. This chapter describes the other functions mentioned in the list.

Archived

© Copyright IBM Corp. 2003. All rights reserved.

9.1 Online change

It is possible to make system definition changes without shutting down the online system. A request to add an application or to modify the current set of programs, transactions, and database usage does not need to force a complete system definition and an IMS restart, with possible interruption for the online users.

You can examine the request to see if an online change can be made. If the request does not involve changes to the IMS network or the use of static terminals as defined in the current IMS system definition, you can arrange for the changes to be made while the IMS system is executing online.

9.1.1 Changing resources by online change

Example 9-1 lists the allowable changes to various resources that can be made via a MODBLKS type of system definition and an online change. The list is based on the stage 1 system definition macros. Also changes to ACBLIB, FMTLIB, and SMU security matrixes can be introduced using the online change.

Table 9-1 System defined resources and permissible online changes

System definition macro	Permissible online changes
APPLCTN	Add a PSB (application) and its attributes Change attributes Delete a PSB
DATABASE	Add a database and its attributes Change attributes Delete a database
RTCODE	Add a routing code and inquiry attributes Delete a routing code
TRANSACT	Add a transaction and its attributes Change attributes Delete a transaction

9.1.2 Review of local online change

IMS online change has been available with previous IMS releases. It allows changes to be made to IMS resources without bringing down the IMS system. It is invoked and controlled with the various **/MODIFY** commands:

```
/MODIFY PREPARE
/MODIFY COMMIT
/MODIFY ABORT
/DISPLAY MODIFY
```

Online change switches online libraries between an active library and an inactive library that contains the changes to be implemented for the following libraries:

- IMSACBA and IMSACBB
- MODBLKSA and MODBLKSB
- MATRIXA and MATRIXB
- FORMATA and FORMATB

The libraries in use by the running IMS (active libraries) are recorded in MODSTAT data set.

In an IMS Parallel Sysplex environment, online change poses some operational problems. The online change must be coordinated between the multiple systems in the sysplex.

The IMS resources are generated with the appropriate utility process:

- DBDGEN, PSBGEN, and ACBGEN
- MODBLKS system generation
- MFS utility
- Security generation using the Security Maintenance Utility (SMU)

Figure 9-1 shows the various preparation of resources for the online change process. The staging libraries are updated with the changed resources.

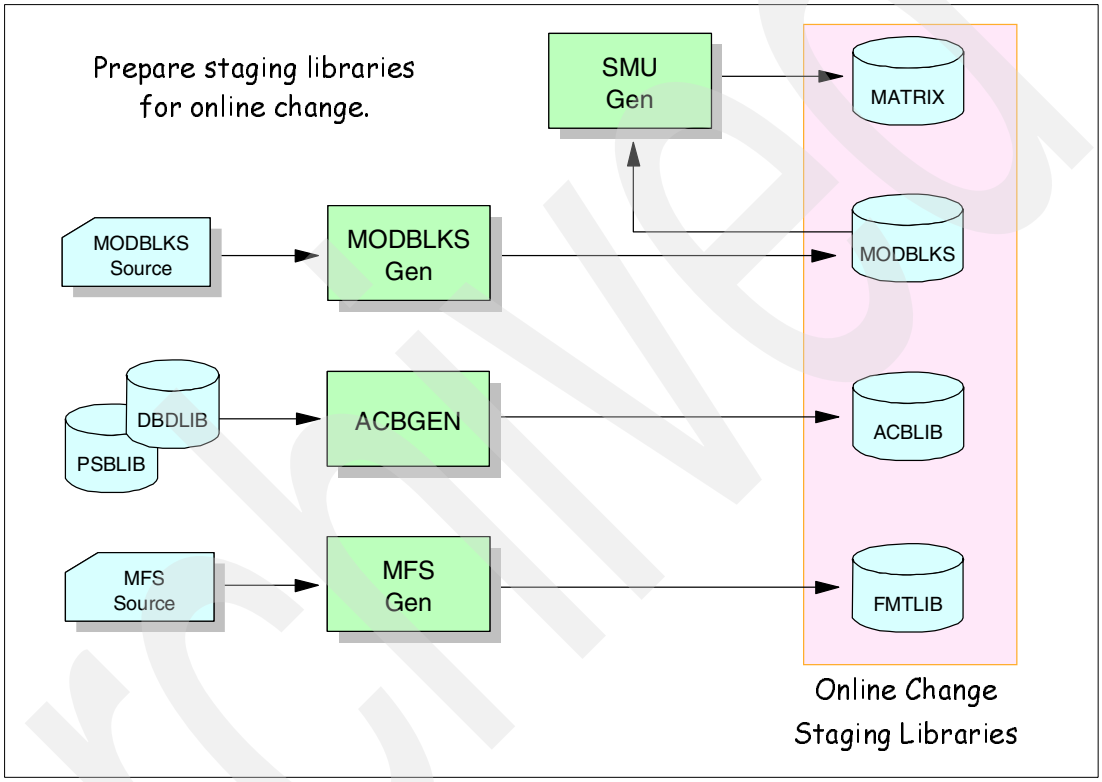


Figure 9-1 Online change preparation

Before the online change may be done, the updated staging libraries must be copied to the inactive libraries. This is done with the Online Change Copy utility (DFSUOCU0). The utility reads a MODSTAT status data set to determine the inactive libraries to updated. The changed resources are then copied to the inactive library.

Figure 9-2 shows the staged changes, in a Parallel Sysplex. After the changes have been staged to the inactive IMS libraries, a **/MODIFY PREPARE** is issued for IMS1, to prepare the changes for implementation in the online system. A **/DISPLAY MODIFY** command is issued to display work in progress for resources to be changed or deleted. If there is no work in progress, a subsequent **/MODIFY COMMIT** on IMS1 completes the online change. The inactive and active libraries are switched, the MODSTAT data set is updated to indicate the new active libraries, and processing continues.

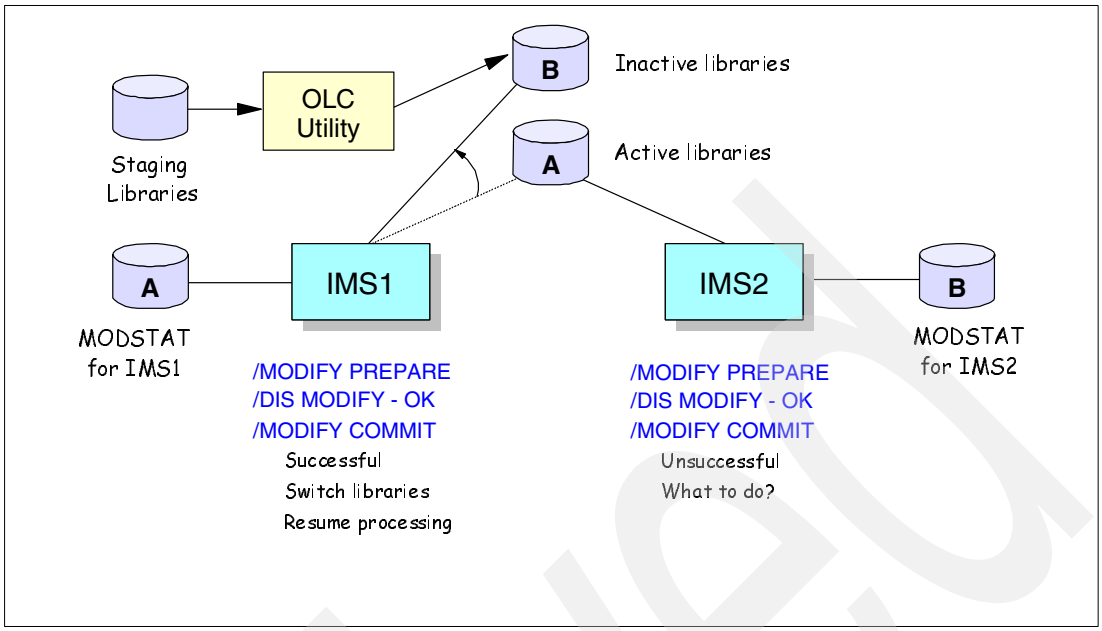


Figure 9-2 Manually coordinated online change in a sysplex environment

To coordinate the online change with IMS2 in this sysplex environment, a **/MODIFY PREPARE**, **/DISPLAY MODIFY**, and **/MODIFY COMMIT** are issued on IMS2. Unfortunately the online change can fail and leave you with the IMS systems in the example using different libraries and therefore different resources as shown in Figure 9-3.

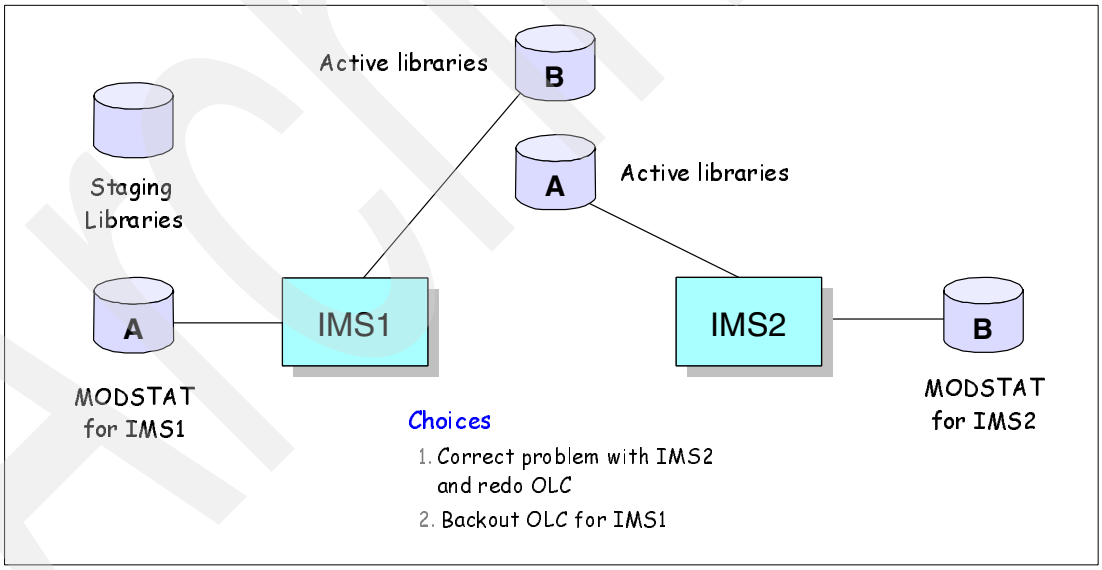


Figure 9-3 In-consistent libraries in a sysplex environment

This leaves you with two options:

- Correct the problem with IMS2, and reissue the online change command.
- Backout the successful online change that was done to IMS1.

As you can see trying to successfully coordinate online change in a data sharing environment can be cumbersome and problematic.

IMS Version 8 offers an alternative to the traditional online change — global online change, using the new Operations Manager to process commands and the new Resource Manager to coordinate the online change across the IMSplex.

9.1.3 Overview of global online change

The processing of global online change is similar to that of local online change. The resource preparation and staging is the same as in the past with local online change. For IMS Version 8, modifications have been made to coordinate the change across multiple IMS systems.

The online change status (OLCSTAT) data set is shared by all of the IMS systems. Its function is similar to that of the MODSTAT data set with local online change. OLCSTAT contains the DDNAMEs of the active libraries. An active library is either an "A" or "B" library. For example, the active ACBLIB is either IMSACBA or IMSACBB. We will see later that OLCSTAT data set also contains some other information.

With global online change, typically all of the IMS systems share the same libraries. They each use the same DDNAME. For example, if one IMS is using its IMSACBA DD for ACBLIB, all other IMSs will also be using this DD for ACBLIB. We will see later that there is an option to use different data sets.

Global online change is invoked by IMSplex commands, such as the INITIATE OLC command. Global online change is an IMS function to change resources online for all of the IMSs in an IMSplex. Global online change has the following characteristics:

- Global online change is enabled with DFSCGxxx OLC=GLOBAL in IMS PROCLIB.
- IMS uses Resource Manager to coordinate the phases of online change with the other IMSs in the IMSplex.
- IMS uses a shared OLCSTAT data set to contain the global online change status. The Global Online Change utility (DFSUOLC0) is used to initialize the OLCSTAT data set and only needs to be run once before the first IMS is cold started for the first time.
- One INITIATE OLC PHASE(PREPARE) command coordinates the online change prepare phase across all of the IMSs in the IMSplex.
- One INITIATE OLC PHASE(COMMIT) command coordinates the online change commit phase 1, commit phase 2, and commit phase 3 across all of the IMSs in the IMSplex.
- IMS commits the global online change by writing to the OLCSTAT data set, after all of the IMSs in the IMSplex have committed the online change by completing the commit phase 1 successfully.
- One TERMINATE OLC PHASE(COMMIT) command coordinates the online change abort phase across all of the IMSs in the IMSplex.
- QUERY MEMBER command shows online change status of IMSs.
- Offline preparation steps needed for local online change also apply to global online change, such as DBDGEN, PSBGEN, ACBGEN, MODBLKS system generation, Online Change Copy utility, and MFS format utility.
- The Online Change Copy utility (OLCUTL) must be run to copy staging libraries to inactive libraries determined by OLCSTAT data set.
- IMS restart performs consistency checking of global online change resources across the IMSplex, such as online change library data set names and the OLCSTAT data set name.
- IMS does not support an online switch between local online change and global online change.

9.1.4 Components

The global online change components are:

- OLCSTAT data set
  - Global online change status data set (like MODSTAT data set)
- RM address space
  - At least 1 RM required in IMSplex defined with CSL
  - Only one RM allowed in IMSplex with no resource structure
- Common Service Layer address spaces
  - OM required for new online change commands
  - SCI required on each z/OS image
- RM clients (IMS subsystems)
  - Each IMS DB/DC, DCCTL, or DBCTL subsystem is a client
- Coupling Facility list structure (resource structure)
  - Resource structure contains IMSplex-wide process information and global resources
  - Resource structure is optional
- Common Queue Server (CQS)
  - Required if resource structure used

9.1.5 OLCSTAT data set

The OLCSTAT data set contains information about the global online change environment. It must be initialized once before the first IMS is cold started for the first time. The initialization is done by the Global Online Change utility (DFSUOLC0) which will set the initial online change library suffixes to A or B in a header record.

The header record identifies the global online change suffixes for each library, the modify-id of the last (current) online change, the type of online change last performed, and a flag indicating that global online change is in-progress. The flag is to keep someone from trying to start another global online change while one is in-progress and prevent any IMS from initializing during a global online change.

The OLCSTAT dataset contains the list of IMSs that participate in global online change and are in sync with the current online change libraries. If IMS misses a global online change because it was not active when global online change was executed, then that IMS might have to cold start, depending on how many global online change processes it missed, and what the last type was. For example, if the last type was FORMAT only, then a warm start is OK. Otherwise, a cold start is required.

Also, the OLCSTAT data set is:

- Dynamically allocated by IMS
- Initialized by Global Online Change utility (DFSUOLC0)
- Updated by IMS restart, global online change, and /CHE FREEZE LEAVEPLEX command
- Defined by IMSs with DFSCGxxx OLCSTAT= parameter
- Not compatible with previous releases of IMS

Archived

9.2 Single point of control (SPOC)

A major new function being delivered with IMS Version 8 is the ability to manage a group of IMSs (an IMSplex) from a single point of control (SPOC). The purpose of the single point of control is to ease IMSplex management by: providing a single system image of the IMSplex, and allow commands to be entered and routed to all, or selected IMSs in the IMSplex. Prior to IMS Version 8, most commands and automated processes only affected a single IMS.

This functionality is provided by components of the new Common Service Layer (CSL).

9.2.1 CSL components

The Operations Manager provides an application programming interface (API) for application programs that perform automated operator actions. These programs are called automated operator programs (AOPs).

An AOP issues commands that are embedded in OM API requests. The responses to these commands are returned to the AOP (through the OM API) embedded in XML tags. An AOP can be written in assembler using the macro interface documented in *IMS Version 8:Common Service Layer Guide and Reference,* SC27-1293. An AOP can also be written in REXX enabling it to run TSO, or in the NetView environment. A REXX AOP uses the REXX SPOC API shipped with IMS Version 8. The REXX SPOC API supports a subset of the functions supported by the OM API.

Before an AOP can start issuing OM requests, it must register with SCI. The OM uses SCI to route the commands or requests to the IMSplex components on behalf of the AOP. When the AOP completes its work with the IMSplex, it must deregister from SCI.

One example of an AOP is the TSO SPOC application described in 9.3, “TSO SPOC application” on page 215, that is shipped with IMS Version 8.

IMS Version 8 does not require the use of SPOC, as the existing command interfaces for the WTOR, MTO, and E-MCS console continue to be supported. The new format IMSplex commands, however, can only be entered from a SPOC.

Figure 9-4 shows an overview of the CSL environment and the command interfaces provided by the new SPOC facilities using the SCI and the existing classic interfaces that are still available.

Archived

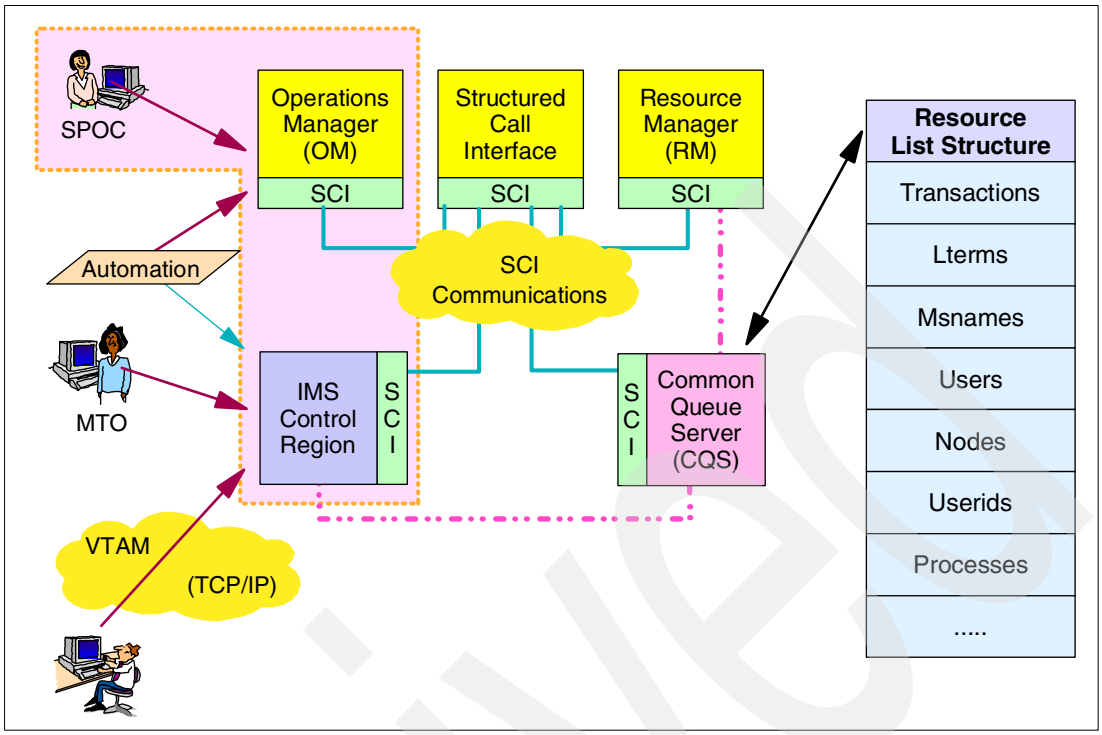


Figure 9-4 IMS command access in a CSL environment

The SPOC application may or may not be on the same system as the OM, but it must be on same system as the SCI, as the SCI is used to communicate with OM.

You can write your own SPOC applications, use the IBM-provided TSO/ISPF SPOC application, or write a REXX SPOC application that uses the REXX SPOC API. Using a SPOC application, you can:

- Issue commands to all the IMS subsystems in an IMSplex.
- Display consolidated responses from those commands.
- Send a message to an IMS terminal that is connected to any IMS in the IMSplex using the BROADCAST command.

The limitations (commands not supported) of the TSO SPOC application are the same as those for the OM API. The command supported by the OM API can be found in the *IMS Version 8: Release Planning Guide,* GC27-1305. Complete information on all of the IMS commands, can be found in the *IMS Version 8: Command Reference,* SC27-1291.

9.2.2 Command behaviors

In an IMSplex environment, IMS commands issued through OM can behave differently than when those same commands are issued to an individual IMS subsystem. As noted, the new “IMSplex commands” can only be issued through the OM API. Existing IMS commands can be issued through the OM API or to individual IMSs. These commands are called classic commands hereafter.

Commands that are issued to OM are, by default, routed to all the IMSplex components that are active and have registered interest in processing those commands. If you want to route a command to one or more specific IMSs in the IMSplex, use the ROUTE() parameter on the command request.

Depending on whether an IMSplex is defined with a Resource Manager (and there is a resource structure available to RM), command behavior can be affected. When a resource structure is not defined, resource status is maintained on local IMSs in the IMSplex. In this case, commands only have a local effect.

If RM is defined with a resource structure in the IMSplex, RM maintains global resource information, including resource status. So, in this scenario, resource status is maintained both globally and locally. Usually, if a user signs off or a client shuts down, resource status is maintained globally but deleted locally.

Another behavior that is worth noting is how command processing clients process classic commands that are routed to the entire IMSplex. In general, OM chooses one of the command processing clients in the IMSplex to be the “master” to coordinate the processing of the classic commands.

Whether the master (or a non-master) client will process a classic command depends on where the command resource status is kept:

- If the command resource status is kept in a resource structure, the classic command will usually be processed by a non-master client where the command resource is active.
- If the command resource is not active on any of the command processing clients in the IMSplex, OM will route the classic command to the master client. If the classic command is being routed to all the clients in the IMSplex, command processing clients where the command resource is not active will reject the classic command.

9.3 TSO SPOC application

IMS provides the TSO/ISPF based single point of control (SPOC) application from which a user can manage operations of all IMS systems within a single sysplex (IMSplex).

Although it is called the single point of control, it does not imply that there can be only a single SPOC. Nor does it imply that a single SPOC can completely control an IMSplex. There can be more than one TSO SPOC in an IMSplex.

There are many cases where commands must be entered into IMS as a result of an operator or automation program seeing a DFS™ message, or the commands are entered at a scheduled time or time interval.

The TSO SPOC does NOT see any IMS messages other than those returned in response to command entry by that SPOC.

9.3.1 SPOC registers with local SCI

The TSO SPOC uses an ISPF panel interface and communicates with a single Operations Manager (OM) address space. OM then communicates with all of the other address spaces in the IMSplex (for example, IMS) as required for operations.

As shown in Figure 9-5 TSO SPOC will register with a Structured Call Interface (SCI), which must be on the same OS/390 or z/OS image as the SPOC. It will also register with OM which may be on any image in the IMSplex. After registration, the SPOC will use SCI communications to send commands through OM to any or all IMSs in the IMSplex.

Every SPOC must register with SCI. The SPOC does not have to be on the same OS/390 image as OM or IMS, but it does have to be on the same OS/390 image as SCI. After

registering, the SPOC will use SCI to communicate with OM, send commands to IMS, and receive responses.

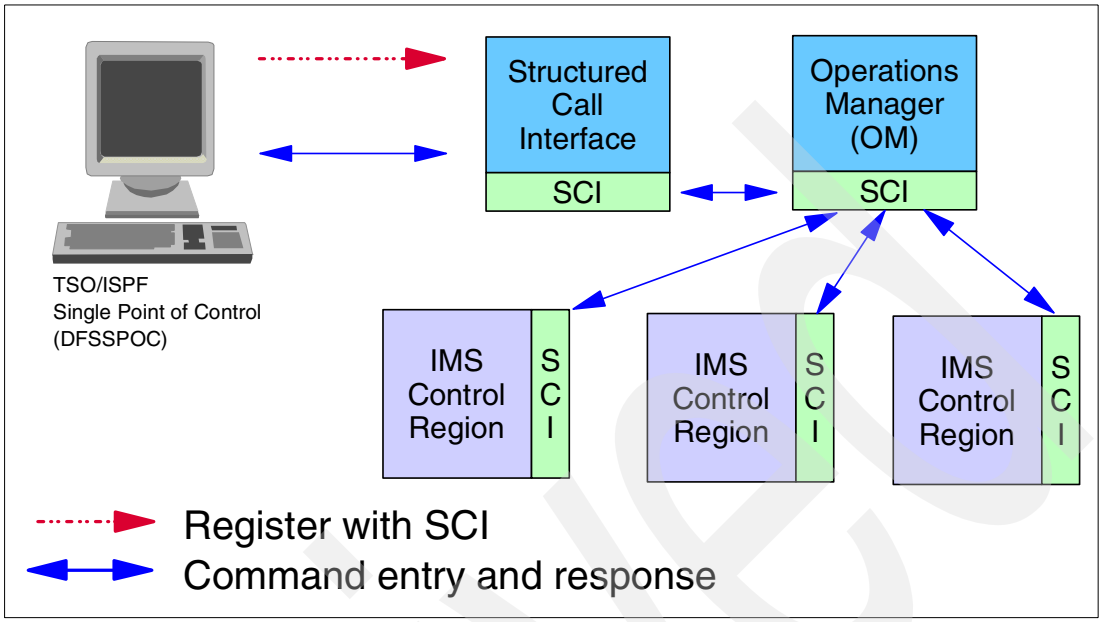


Figure 9-5 TSO SPOC registration and communication

9.3.2 IMS provided TSO/ISPF single point of control (SPOC)

The IMS Version 8 provides a SPOC application that can be invoked using TSO/ISPF. The TSO SPOC must be on the same OS/390 as SCI, but not OM.

The DFSSPOC application enters commands to IMS through OM using the CSLOMxxx interface and receives responses in XML format. It then formats these responses and displays them on the screen similarly to the way IMS would display them.

The TSO SPOC does not perform security checking on its own. This is done by OM and/or IMS using the TSO user ID for authorization.

9.3.3 TSO SPOC functions to an IMSplex

The TSO SPOC provides the following functions to an IMSplex:

- The user can set preferences for command entry, such as the IMS or IMSs to route the command to, whether or not to allow short cuts and how to process them, a default OM wait time, and so on.
  - Shortcuts are abbreviated ways of entering commands. A command which is entered frequently can be abbreviated to just a few characters.
- Presents a single system image for an IMSplex by allowing the user to enter commands to all IMSs in the IMSplex from a single console.
- IMSs can be part of a user-defined group. For example, in an 8-way IMSplex, four could belong to one group and four to another. Commands can then be routed to groups instead of individual IMSs.
- Displays consolidated command responses from multiple IMS address spaces.
- Sends a message to an IMS terminal connected to any IMS control region in the IMSplex by using the IMS /BROADCAST command.

- Allows the user to enter commands.
- Receives command responses and formats the XML tagged responses into readable format.
- DFSSPOC application collects the consolidated responses from OM and displays them, identifying which IMS each response is from.
- Displays IMSplex and classic IMS command responses.
- Allows the user to sort IMSplex command response by column.
- Allows the user to set default command parameters.
- DFSSPOC will keep a short history of recently entered commands and the responses, allowing the user to go back and browse early command responses and even to edit and reenter the same command.
- Allows the user to enter long commands perhaps with many database names.
- Allows user specified grouping of IMSplex members.

As an ISPF application, it also provides other typical ISPF application features:

Print The command responses and log information can be put to the ISPF list file.

Save Information can be saved to a user specified data set.

Find Command responses can be searched for text strings.

Help Help dialogs are available for application use information.

Scrolling Where applicable, data can be scrolled left and right, or up and down.

Split A split screen can be used to start multiple DFSSPOC applications.

Retrieve Get previously issued commands back into the command line.

9.3.4 DFSSPOC inputs and outputs

There are a few inputs and outputs from the TSO SPOC. See Table 9-2 and Figure 9-6

Table 9-2 DFSSPOC inputs and outputs

Component	Description
3270 terminal	Using ISPF services, accepts commands from user and displays the response
IMSplex	Sends command to IMSplex and manages the command responses
ISPF profile	Saves various variables for next session
Command status table	Keeps a history of commands issued so they can be issued again
SAVE AS file	Specified by the user to have command response output
ISPF list file	Holds results of print requests
User defined group table	Keeps the group name and group members that the user has defined

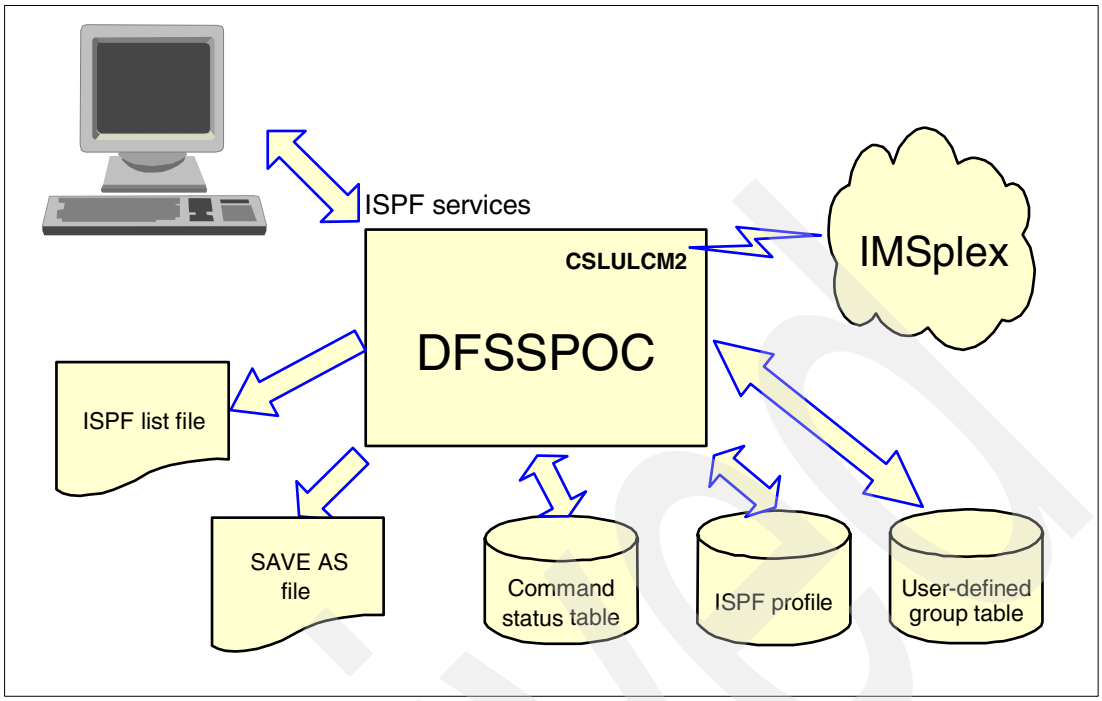


Figure 9-6 DFSSPOC inputs and outputs

9.4 Operations Manager programming interface

The interface to all of the new CSL services provided by the new components, SCI, OM, and RM, are documented in *IMS Version 8:Common Service Layer Guide and Reference,* SC27-1293.

The Operations Manager (OM) provides services to the IMSplex by allowing user supplied Automated Operator Programs (AOPs) to send commands to CSL components and then consolidating the responses to those commands before returning them to the AOP. OM can optionally provide command security using RACF or a user written command authorization exit.

The AOP (for example, the SPOC) can start and join the IMSplex with only SCI started on the local system. However, it cannot issue any commands (including QRY IMSPLEX) unless there is an OM available within the IMSplex. So, the AOP can assist in the startup of RM or IMS, for example it can be used to issue the /NRE command to IMS. IMS will have registered with OM for the commands that it can process (includes both classic commands and the new IMSplex commands). IMS is called the ***command processing (CP) client***.

9.4.1 AOP starting steps

When an AOP is started, it must join the IMSplex by registering with the local SCI address space. This is done with a CSL (CSLSCREG) macro described in the referenced manual. SCI may invoke RACF to authorize that AOPs user ID to join the IMsplex. Once the AOP has registered with SCI, it can begin submitting commands from end-users and receiving responses from the CP client(s) through OM using other CSL macros. There can be two type of clients: a command entry client and a command forwarding client. The most important difference between them is that a command forwarding client should be an authorized program. If it is not, it will not be able to pass the user ID from the network connected

command entry client, and the user ID associated with the command forwarding client will be used for security authorization. Here are typical examples of the steps that an AOP might take.

1. Register with SCI; join the IMSplex as an AOP.
2. Receive a request from an AOP client to submit a command to one or more IMSs in the IMSplex. This request might be from a directly connected z/OS user (for example, a TSO user), from an external (network attached) client, or from another system address space such as NetView.
3. Submit the command to OM using CSL provided macros (CSLOMI or CSLOMCMD). Included in the request are the command and any routing (which IMSs) and timeout (WAIT) values.
4. When OM receives the command, it will (optionally) authorize the AOP user ID for the command. It will then forward the command to the CP client(s) according to the routing information provided by the AOP. It will wait for a response for a period of time specified on the input from the AOP.
5. The CP client will execute the command and return the response to OM encapsulated in XML tags.
6. When OM has all the responses, or the WAIT interval has expired, it will forward the responses to the AOP, still encapsulated in XML.
7. The AOP would process the response depending on where the command originated.
   a. If it came from a locally (z/OS) attached user with a display device, the AOP should format the response for display, interpreting the XML tags and putting the response in a format understandable by a person. An example of this type of AOP is the TSO SPOC provided with IMS Version 8 and described in detail in 9.2, “Single point of control (SPOC)” on page 213.
   b. If it came from a network client, then the AOP can simply forward the response, XML tags and all, to the network client and let that client format the response.
   c. A third possibility is a user written automation program running, for example, as a NetView EXEC which would in turn invoke a function to register with the local SCI and issue commands through OM to any or all IMSs in the IMSplex. In this case, the client is the NetView EXEC and there is no external client to display a response to. The EXEC merely examines the response to determine whether the command was successful or not.
8. The AOP would continue receiving, forwarding, and responding to requests from its clients, then deregister from SCI.

Figure 9-7 shows how each of the first two of these AO client types might be configured in an IMSplex.

Archive

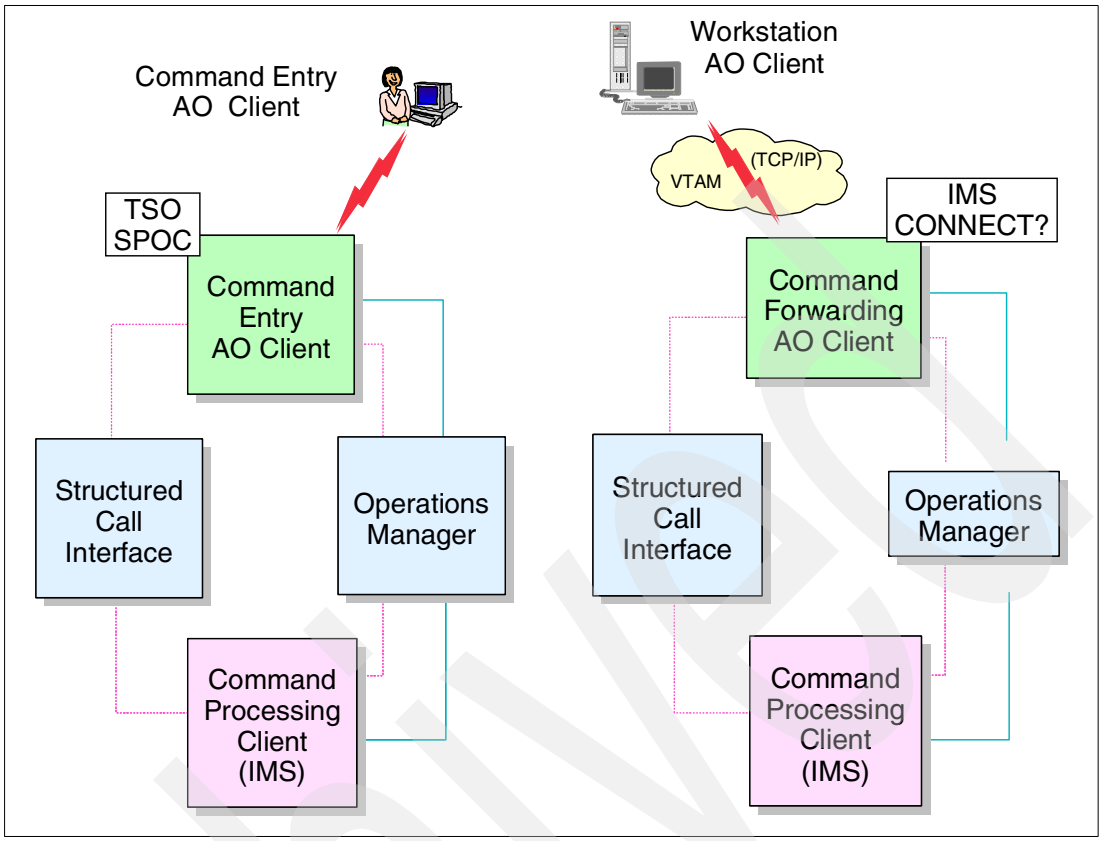


Figure 9-7 Typical AOP configurations

The following section describes in detail how a REXX EXEC might be written to register with the IMSplex and invoke the OM interface.

9.5 Automatic RECON loss notification (ARLN)

The intention for automatic RECON loss notification (ARLN) is that the first DBRC instance detecting that one of the RECON data sets has had an error will not only force a reconfiguration to use and activate the spare copy by copying the good RECON into it, but it will also automatically notify all other DBRC instances about the reconfiguration and the discarded RECON data set.

9.5.1 Process notification

The notification will be routed to DBRC instances which have been registered with the Structured Call Interface (SCI) when they are started. This includes those DBRC instances which are indicated by message DSP0388I (added in IMS Version 7) and includes online, batch with DBRC, utilities with DBRC and the DBRC utility itself (DSPURX00). The SCI is used for the communication between all registered DBRC instances across your IMSplex.

9.5.2 Redefining the RECON discarded

The unavailable RECON data set gets discarded instantly from all notified DBRC instances. All notified DBRC instances will issue message DSP1141I in response to the notification. As long as you are using dynamic allocation for your RECON data sets, all involved DBRC

instances will also deallocate the discarded RECON data set. Now you can delete and redefine this RECON data set. The next access to your RECONs will propagate the newly defined RECON data set as the new available spare RECON data set.

The propagation of an unavailable RECON data set across your sysplex (SCI registered DBRC participants) prevents your IMS Version 8 subsystems from holding the allocation of the original RECON data set, possibly for a long time. Prior to IMS Version 8, the deallocation was usually done by the next access to the RECON so you had to wait or it was forced by any command against the RECON issued from every active online subsystem. Figure 9-8 gives you an overview how DBRC is cooperating with SCI.

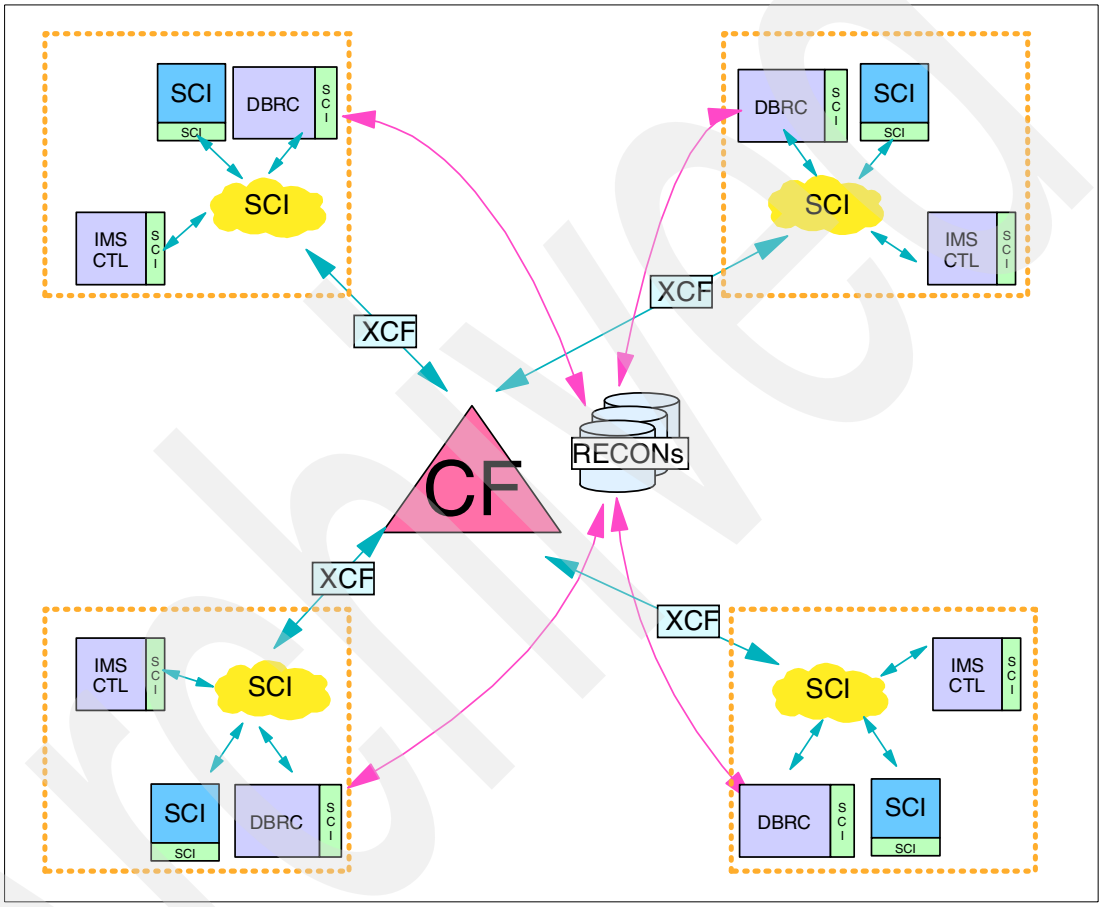


Figure 9-8 DBRC and SCI

9.6 Language Environment (LE)

Before Language Environment, the high-level languages run time services were distributed with the compiler language product. Each high-level language had its own run time library to perform these and other functions, as shown in Figure 9-9.

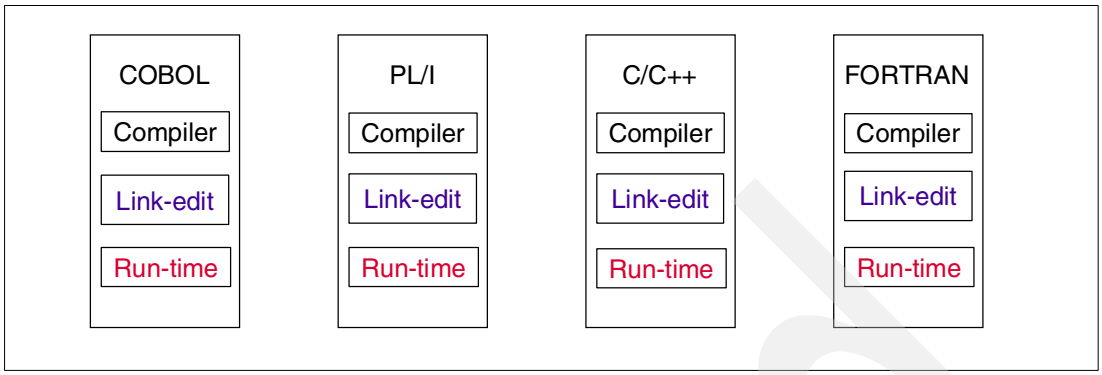


Figure 9-9 Language specific run time services

Language Environment (LE) is the replacement product of older language-specific run time libraries and is now a base element of the z/OS, OS/390, VM/ESA®, and VSE/ESA™ operating systems.

9.6.1 New run time LE services

Today, LE has become IBM's key language product for all new run time services as shown in Figure 9-10. These services are available to the application developer from multiple high-level languages including COBOL, PL/I, C, C++, and FORTRAN. Even Assembler programs can use these run time services (as long as they conform to the LE conventions).

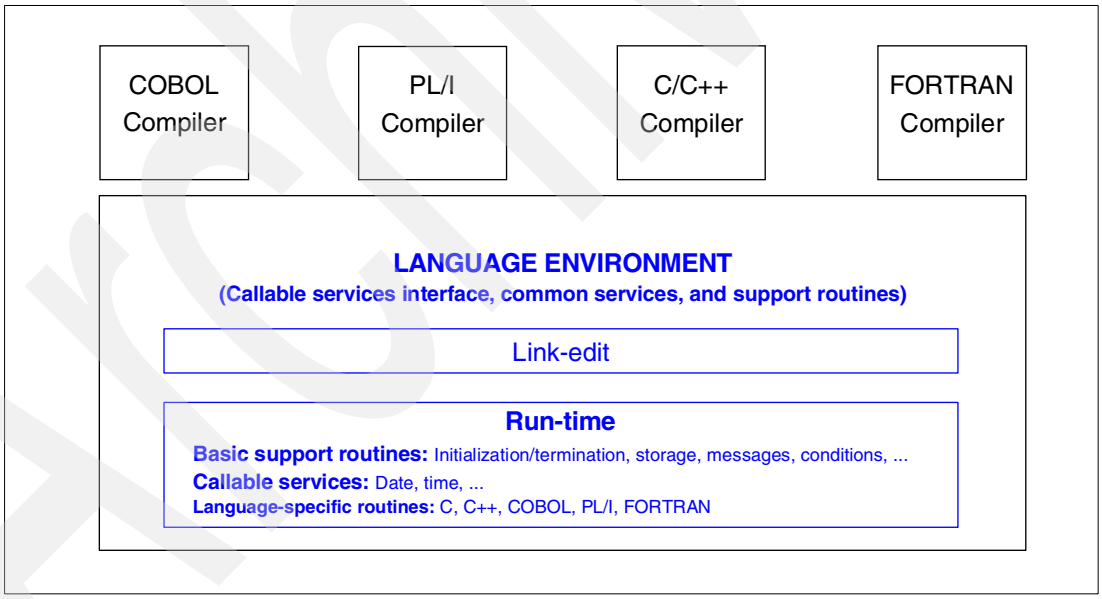


Figure 9-10 LE as a common run time environment

The CSL architecture enables the Language Environment (LE) dynamic run time options, and introduces new Operation Manager commands for this purpose, but because this is not a specific feature for IMS Parallel Sysplex only, it is not described here in detail.

Archived

10

Introduction to IMSplex connectivity

This publication uses the term IMSplex to represent a collection of IMS TM subsystems that are related, either by sharing common databases and/or message queues or by providing common services to the networked end user community. In effect, the goal of an architected IMSplex environment should allow the end user to view the IMSplex as a single entity, a provider of IMS service.

In general, one could describe the objectives of an IMSplex as:

- Presenting a single image (provider of service) to the end user community.
- Providing greater availability and/or capacity to the end user community.
- Allowing your organization to take advantage of new technology with improved price and performance characteristics.
- Allowing existing applications to function without change.

We introduce the various factors that must be taken into account, along with suggested solutions, to implement an IMSplex that comes as close as possible to achieving your objectives of high availability, coupled with efficient workload balancing and distribution.

© Copyright IBM Corp. 2003. All rights reserved.

10.1 IMS connections

When migrating from a single to a multiple IMS environment, provisions must be made for the subsystems, intelligent workstations and other connectors to the existing IMS to be able to communicate seamlessly and efficiently. As part of migration planning, these connections must be identified for the existing environment so that provisions can be made for satisfying their requirements in the target environment.

Because the goal is no longer to have a single IMS communicating with its connectors, some changes might be required not only to the IMS systems but also to the connectors. For example, if the existing IMS were connected to a DB2 subsystem, each clone of it will have to be connected to a DB2 subsystem. For APPC conversations, SIDE_INFO and TP_PROFILE data sets might require updating. Each connection should be evaluated for its requirements in the target environment.

Table 10-1 identifies the more common types of connections found in IMS environments. In this table, the type of connector and the type of connections are identified.

Table 10-1 Summary of IMS connections

Connector type	Connection type	Description
IMS	MSC	Existing IMS is connected through MSC to another IMS.
IMS	ISC	Existing IMS is connected through ISC to another IMS.
CICS	ISC	Existing IMS is connected through ISC to a CICS region.
CICS	DBCTL	Existing IMS is providing DBCTL services to a CICS region.
DB2	ESS	Existing IMS is connected to a DB2 subsystem.
LU62	APPC-IMP-IN	Existing IMS receives implicit APPC transactions from APPC applications.
LU62	APPC-IMP-OUT	Existing IMS runs applications which issue CHNG/ISRT calls to APPC destinations
LU62	APP-EXP-IN	Existing IMS runs applications which ACCEPT explicit conversations from APPC applications.
LU62	APPC-EXP-OUT	Existing IMS runs applications which ALLOCATE explicit conversations with APPC applications.
MQSeries	ESS and BMP-IN	Existing IMS receives MQSeries input using MQ calls and the trigger monitor BMP.
MQSeries	ESS - OUT	Existing IMS runs applications which explicitly send messages through MQ calls
MQSeries	OTMA	Existing IMS receives and sends MQSeries messages through MQSeries IMS Bridge using OTMA.
Web Server	OTMA	Existing IMS receives transactions from the Web through MQSeries or IMS Connect to OTMA
Web Server	APPC	Existing IMS receives transactions from the Web through an APPC connection.
TCP/IP Sockets	BMP	Existing IMS uses a BMP to provide a sockets interface.

Archived

Connector type	Connection type	Description
TCP/IP	OTMA	Existing IMS uses IMS Connect for sockets connections.
DCE RPC	ISC	Existing IMS receives RPC requests through DCE Application Server's ISC interface.
DCE RPC	APPC	Existing IMS receives RPC requests through DCE Application Server's APPC interface.
DCE RPC	OTMA	Existing IMS receives RPC requests through DCE Application Server's OTMA interface.
ODBA	DRA	Provides OS/390 batch application programs, DB2 Stored Procedures, and WebSphere EJBs access to IMS.

10.2 IMS configuration considerations

There are several manners in which an IMSplex can be configured. The decisions are based on both technical and business rationale. We present a few options in the following material.

10.2.1 Cloning

One way to configure an IMSplex is to replace an existing IMS system with multiple IMS systems, each capable of providing the same services and functions as the original IMS. This is called *cloning*. When cloning is used, the IMSplex contains multiple servers, each able to provide similar functions to the end user community.

The advantages of cloning are:

- Increased availability

  If multiple servers exist, the loss of one server does not cause a complete loss of IMS service. In addition, preventive or emergency maintenance; either hardware or software, can be introduced into the IMSplex, a server at a time.

- Increased capacity

  As the workload requirements of the enterprise grow, cloning allows additional IMS systems to be brought into the IMSplex, as required, in a non-disruptive manner.

- Ability to benefit from new hardware technology

  Distributing the workload across multiple processors in the IMSplex, while preserving the image of a single IMS system to the end user community, allows the enterprise to gain the benefits (price and performance) of new processor technology without having to be overly concerned about the power of a single processor.

Multiple cloned IMS systems could be very similar but, perhaps, not identical. Factors that might force systems to differ depend on several conditions:

- There may be affinities within applications, Client file use, special operating processes or specific hardware attachment that might restrict complete cloning to be achieved.
- Communication among IMS systems which use traditional queuing (for example, to achieve workload balancing) can be achieved using MSC and/or ISC. These MSC and/or ISC definitions must be different and, therefore, cannot be cloned. Distributing workload among IMS systems can be automatically achieved with shared queues, therefore eliminating the need for unique MSC and/or ISC definitions.

Archive

10.2.2 Joining

One can combine multiple, somewhat independent, IMS systems into an IMSplex. This situation might arise when these multiple IMS systems need to share a subset of the enterprise's databases. The combining of multiple IMS systems is called *joining*. When joining is used, the IMSplex is not viewed as a single provider of service to the end user community. Rather, it continues to be viewed as multiple, independent providers of IMS service.

An example of joining could be a situation where there are three IMS systems on three different OS/390 systems, each one handling the IMS processing requirements for three different manufacturing plants. The vast majority of the databases in each system are independent (unique to the specific manufacturing facility). However, there are a few databases that represent corporate, rather than plant, data. Therefore only the limited cloning of funtionality is required.

If systems are to be joined rather than cloned, one gives up several advantages of a Parallel Sysplex environment:

- Application availability. When an application can only execute on a single system, the loss of that system stops all processing for that application until the failed system is restarted.
- Workload balancing. By restricting an application's processing to a single IMS, one cannot take advantage of excess processing capacity that might be available in the other IMS systems in the sysplex.

10.2.3 Front-end, back-end

Another configuration option is to replace an existing IMS system with one or more *front-end and back-end* systems.

The design is that the front-end IMS systems have the IMS network connected to them but do not have any dependent regions to process application programs. The back-end IMS systems have dependent regions to process application programs but do not have the IMS network connected. Transactions and messages flow between the front-end and back-end systems through MSC and/or ISC links or shared queues.

VTAM Generic Resources can also be used to balance the network workload across the joined systems, while shared queues enable the transaction workload to be routed to the correct system for processing.

10.2.4 IMS Version 6 features related to IMSplex connectivity

IMS Version 6 provided major enhancements to the manner that IMS subsystems can merge into very powerful IMSplex groups.

Shared queues

This is a replacement for the manner in which input transactions and output messages are physically managed and queued for processing. With shared queues, the use of the message queue data sets has been eliminated and replaced by queuing messages to list structures in one or more CFs.

Since a CF can be accessed by multiple IMS subsystems, the messages queued in a CF can be accessed and processed by any IMS that is interested in processing a particular resource name. Shared queues enable:

- Workload balancing. Multiple IMS subsystems can balance dependent region application workloads among themselves automatically.

- Improved availability. If one IMS fails, the surviving IMSs can continue processing.
- Incremental growth. IMS subsystems can be added to the Parallel Sysplex environment without disruption.

VTAM Generic Resources

VTAM Generic Resources (VGR) support allows terminals to log on using a generic name rather than a specific IMS APPLID when requesting a session with one of the IMSs in a Generic Resource Group. VTAM Generic Resource support selects one of the IMS subsystems in the generic resource group to which the session request will be routed.

VTAM's session selection algorithm enables the session requests to be evenly distributed across the IMS subsystems to achieve network workload balancing while simplifying the terminal end user interface to the IMSs in the Parallel Sysplex. Figure 10-1 presents a sample VGR configuration using shared queues facilities.

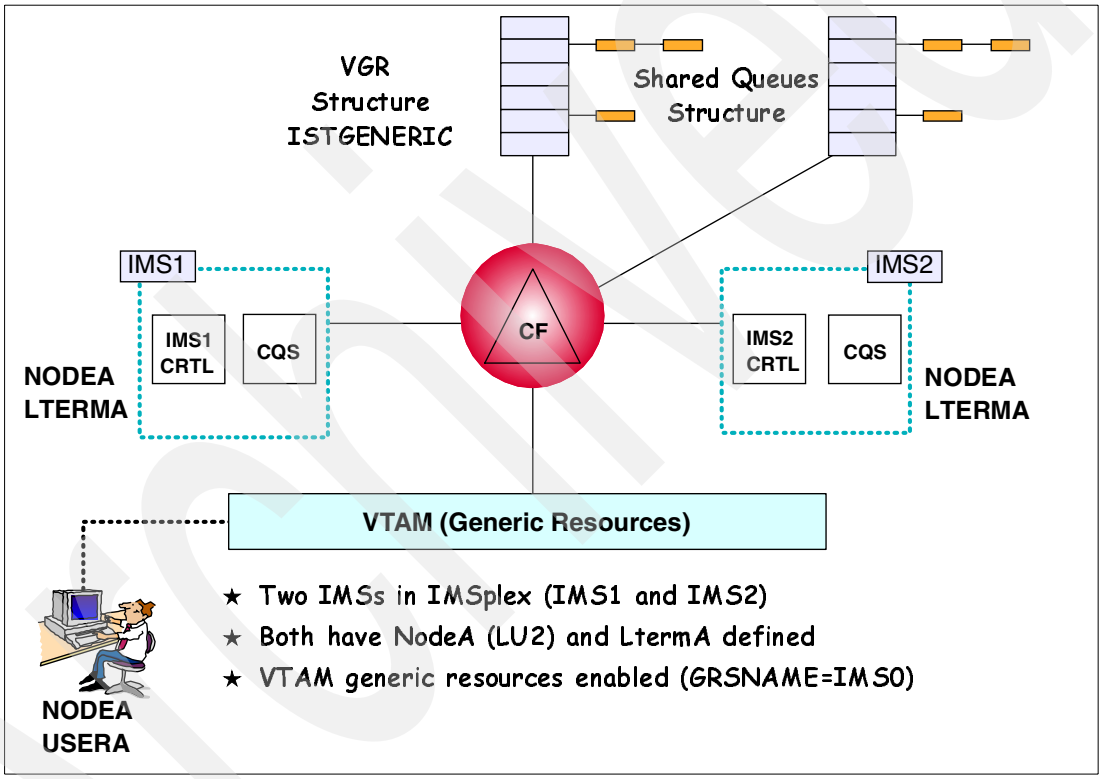


Figure 10-1 IMSplex configuration using VGR

VTAM Generic Resources and shared queues coexistence

The major difference between shared queues and VGR is that IMS shared queues balances the application workload across IMSs, while VGR balances the network workload between IMS and its logged on users. Both are valid requirements, and shared queues and VGR can be used either separately or together.

Besides the application workload balancing, shared queues offers the user an easy way to incrementally increase or decrease the capacity of a Parallel Sysplex without disturbing existing logged on users. This may be done by just starting a new IMS when the workload gets heavy. The new IMS does not need to have network and users connected to it. When the workload decreases again, just quiesce and normally terminate that IMS. No end users are affected.

There are certainly some availability improvements with shared queues. When one IMS fails, the work that it put on the queue can be processed by other IMSs while the failed one is being restarted. And, if it is necessary or desirable to cold start one (or all) IMSs, the shared queues are not lost. They remain in the CF for access by the restarted IMSs when the are ready.

Sysplex communications

Communication among the IMSs in a Parallel Sysplex environment has been enhanced by:

- Allowing commands to be submitted to any IMS TM, DCCTL, or DBCTL subsystem in a Parallel Sysplex environment from a single MCS or E-MCS console in the sysplex.
- Routing synchronous and asynchronous responses to a command to the MCS or E-MCS console which entered the command.
- Providing RACF (or equivalent product) and/or Command Authorization Exit (DFSCCMD0) security for commands entered from MCS or E-MCS consoles.

These improvements in sysplex communications capability allow users to control multiple IMSs within a Parallel Sysplex from a single OS/390 console rather than from multiple OS/390 consoles or IMS master terminals.

Primary and secondary master terminal support

One of the advantages of a Parallel Sysplex environment is the capability to clone the IMS system definitions for all of the IMSs, therefore making the definition process less error prone. The master and secondary master LTERM definitions cannot be cloned in IMS Version 5, because the LTERMs are directly associated with the IMS system that had output queued to them.

IMS Version 6 and shared queues allows the physical terminal and LTERM name definitions for the primary and secondary master terminals defined in an IMS system definition to be overridden at execution time through a new, unique proclib member (DFSDCxxx) associated with each IMS subsystem.

MSC dynamic MSNAMES and shared local SYSIDS

This is a function designed explicitly to assist in the maintenance of IMS systems in a shared queues group when one or more of the IMSs in the group has external Multiple System Coupling (MSC) links to other IMS systems outside the group or when such MSC links are added to one or more of the IMSs in the group.

Dynamic MSNAMEs, as implemented, remove the requirement for each IMS in a shared queues group to include the definitions of the MSC network in the system definition.

Shared local SYSIDs among the members of a shared queues group allow all members of the group to properly handle MSC related messages that are received or sent by members of the group.

Generic /START REGION support

A new generic `/START REGION` command allows a single PROCLIB member to be used to start a dependent region under any IMS system. This is useful when starting Message Processing Regions (MPRs) and Interactive Fast Path Regions (IFPs) and can have some use when starting Batch Message Processing (BMP) Regions.

BMP scheduling flexibility

APAR PQ21039 for IMS Version 6 simplifies the specification of the IMSID for users whose BMPs can execute under multiple control regions in a Parallel Sysplex. Without this enhancement, moving a BMP from one control region to another typically requires a change

in the BMP IMSID parameter. With this enhancement, no changes to BMP JCL are required even when a BMP is executed under different control regions.

10.2.5 Other IMS Version 6 connectivity features of interest

The following IMS connectivity related enhancements were also introduced in IMS Version 6.

Increased number of MSC links and SYSIDS

This feature provides the ability to define more MSC resources to an MSC network. These changes remove the concern of the addition of multiple IMS TM systems to an MSC network in a Parallel Sysplex environment.

Improved security

This is accomplished by propagating the security environment to a back-end IMS when it is processing a transaction that requires signon security. This is an important enhancement in either a shared queues or MSC environment.

Conversational SPA truncated data option

This feature supports a variable length SPA size when conversational processing involves multiple program-to-program switches. This support is useful when the processing IMS is not the owner of the conversation, such as in a shared queues environment.

10.2.6 IMS Version 7 ehancements associated with IMSplex connectivity

The following IMS Parallel Sysplex connectivity related enhancements were introduced with IMS Version 7.

Rapid Network Reconnect

VTAM persistent support was introduced in VTAM Version 3 Release 4. This allows user sessions to survive an application failure and then be restored as soon as the application had been successfully restarted. This support was limited initially to application failures and restarts on the same system.

However, VTAM persistent support was enhanced in Version 4 Release 4 and provides session recovery from any failure including OS/390, VTAM, or IMS in a Parallel Sysplex environment. The enhancements allow the failed application (IMS) to be restarted on a different system in the sysplex.

Rapid Network Reconnect (RNR) support in IMS Version 7 implements VTAM persistent session support. When recovery is required on the same system, IMS will need to be defined as Single-Node Persistent Session (SNPS) capable. When recovery to another system in a Parallel Sysplex, IMS will need to be defined as Multi-Node Persistent Session (MNPS) capable. These definitions are made within the VTAM APPL definition.

RNR support can be setup for automatic reconnect of terminals (ARNR) or no reconnect (NRNR). System-wide defaults can be set up within IMS, but these can be overridden in ETO or logon exits for terminals and terminal types. The override can only override ARNR to NRNR or vice versa, not turn the use of RNR on or off. Specification of the parameter RNR= without a value will attempt to turn on the support to a default on NRNR. APAR PQ55118 in IMS Version 8 provides the capability to specify RNR=NONE.

RNR addresses the user needs for higher availability and reduced CPU overhead following a failure restart. Higher availability is achieved because the VTAM sessions are maintained. Therefore, following a failure, it is quicker to re-establish the sessions. Because the significant

work for session setup and cleanup is reduced after a failure, there is less CPU overhead in these instances. Additionally, network traffic is minimized.

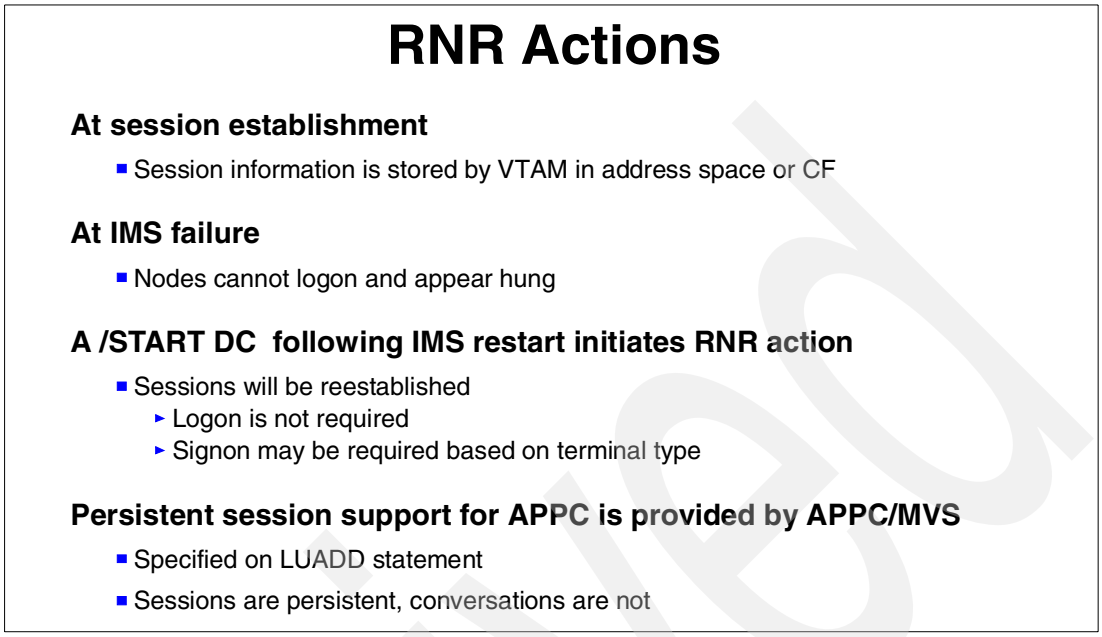


Figure 10-2 RNR actions

VTAM Generic Resource enhancements

To enhance availability to all IMS systems in a Parallel Sysplex, there is a requirement to allow VTAM to reset affinity when there is a processor, OS/390, or IMS failure when IMS cannot control the shutdown.

Two new options for PROCLIB member DFSDCxxx have been provided: GRAFFIN and GRESTAE. These are delivered as part of IMS Version 7 and for IMS Version 6 they are available as APAR PQ18590.

Valid values for GRAFFIN are IMS and VTAM, and their meaning is as follows:

- GRAFFIN=IMS

  GRAFFIN = IMS requests IMS to manage the affinities and is the default.

- GRAFFIN=VTAM

  GRAFFIN = VTAM states that VTAM will manage the affinities. This will impact all IMS terminal sessions except MSC & ISC. MSC does not use VGR as it always logs onto IMS using the specific IMS APPLID. ISC affinities will be managed by IMS.

The GRESTAE parameter is meaningful only when IMS controls the affinities with GRAFFIN=YES. Valid values for GRESTAE are Y (default) or N, as follows:

- GRESTAE=Y

  When GRAFFIN=IMS is specified, IMS ESTAE processing will attempt to delete affinities during IMS abend where no status remains before closing the IMS VTAM ACB. There are circumstances, such as I/O timeouts, where this can delay the termination and subsequent restart of IMS.

- GRESTAE=N

  GRESTAE=N specifies that IMS should close the IMS VTAM ACB immediately and leave affinity for all nodes set.

ETO enhancements

Several enhancements have been made to improve ETO ease of use and capacity. The enhancements include the following:

- ETO descriptor limit

  Prior to IMS Version 7, a descriptor that contained more than 50 records was ignored and an error message issued. This limited, for example, the number of remote LTERMs that could be defined on a single MSC link in an environment that used ETO MSC descriptors. In IMS Version 7, a new algorithm is used to increase the number of records that can be defined for a descriptor. At system initialization, 500 records is set as the new initial limit. When the value of 500 is reached, additional storage to hold 4 x the limit is obtained. The data in the old records is copied to the new space and the old space is freed. The action of increasing the limit by 4 x is repeated as more storage is needed until the absolute limit of 512,000 records is reached.

- Autologon

  The autologon capability in ETO allows IMS to dynamically acquire a NODE as a result of queued output. It is applicable to all terminal types but primarily used for printers. When a printer fails, any autologon users queued to that printer must be manually moved to another printer to allow the messages to be delivered. In the past, this was not always easy because when a user structure was moved to another printer interactively, for example, using remote logon or **/OPNDST**, this temporarily disabled the autologon capability for other users of the printer until the active user was signed off via a **/SIGN OFF** command or via an ASOT timeout. The enhancement in IMS Version 7 allows automatic autologon to occur for queued users regardless of how the terminal was started. Once the queue for the first user is drained, that user is automatically signed off and the next user in the queue is signed on. The process is continued until all users have their output delivered.

  Another autologon enhancement allows the autologon parameters to be changed dynamically via the IMS **/CHANGE** command. The IMS **/DISPLAY** command has also been enhanced to display the auto logon parameters.

- LTERM assignment

  In a static environment, LTERMS are easily assigned from one node to another. In IMS Version 7, the ETO support provides an enhancement that allows IMS to deal with ETO terminals in a way that is similar to static terminals for specific situations, for example, allowing output to be sent to a working terminal when the original terminal is broken. When the **/ASSIGN** command is issued and specifies a nonexistent USER, IMS dynamically creates the control blocks. IMS builds a complete user structure including the user and LTERM control blocks based on the user descriptor definition (either DFSUSER or one specific to the user). If an LTERM is not available, for example, it is already assigned to another user, the user structure will be created without any LTERMs. Additionally, this assignment can optionally be defined as a persistent assignment and, if so, will be remembered across: session failure and restart, system failure and restart, and CLSDST due to printer sharing.

- Associated printer support

  ETO associated printer support allows an end user at signon to specify or to be associated with a specific printer. The associated printer can have many user and LTERM structures, each of which is unique to an individual end user. IMS Version 7 enhances the use of associated printer support in a shared queues environment. When an end-user signs on to an IMS system, the associated print (ASP) user structures, if specified, are created. For a shared queues environment, this IMS is considered the front-end for the end-user. The programs that process the input messages, however, may run in either the front-end IMS or in a back-end IMS system.

If the transaction processes in the front-end, IMS knows which printer LU is to be associated with the ASP structures. If the transaction processes in the back-end, IMS creates the USER and LTERM structures for the printer as a result of an application CHNG call. The creation of the USER structure can specify autologon but does not have to do so. If autologon is specified, then at syncpoint, the back-end IMS can issue a simlogon, establish a session with the printer, and send the message. If the naming scheme for the ASP USER and LTERM structures are unique to the interactive end-user, for example, variation of USERID or LTERM name, then the output creation exit may not have a way to determine what printer LU name to specify for autologon (remember this was specified by the end user in the front-end). The messages will not be delivered until such time when a session is created with a printer that is associated with the ASP structures, for example, a transaction processes on the front-end with output messages for the associated printer.

With the enhancement, registration of interest for the ASP LTERMs will be done to assure more timely delivery of the printer output.

TM and MSC message routing and control user exit: DFSMSCE0

Figure 10-3 on page 232 presents the selection of routing exits available to pre IMS Version 7 users.

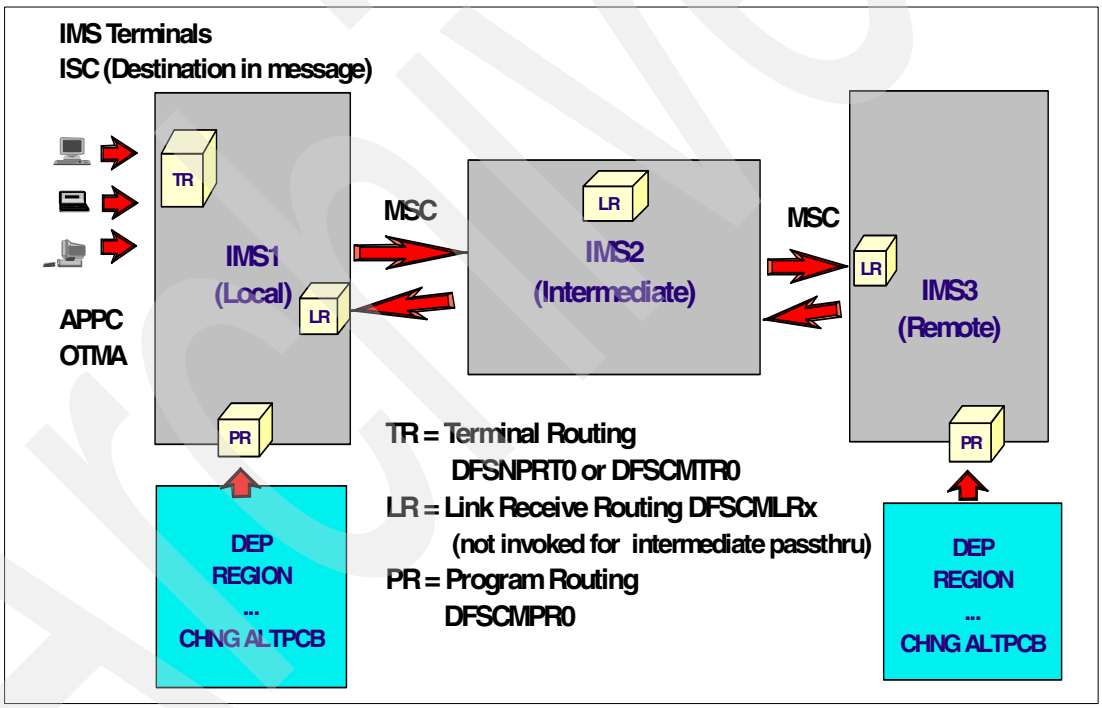


Figure 10-3 View of pre IMS Version 7 routing exit selections

With IMS Version 7, the new TM and MSC message routing and control user exit (DFSMSCE0) combines, replaces, and enhances several services of the message routing exits available in IMS Version 6. Figure 10-4 illustrates some of the tasks that DFSMSCE0 can be applied to.

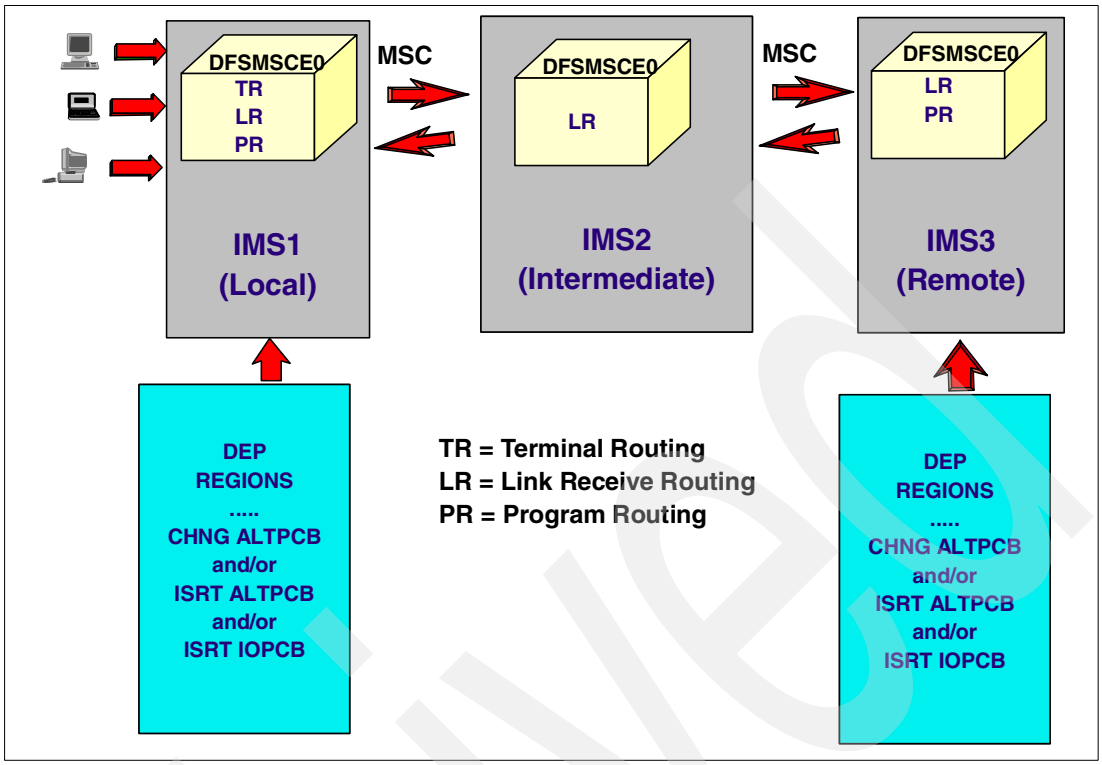


Figure 10-4 Potential uses of the DFSMSCE0 exit

10.2.7 Other IMS Version 7 connectivity features of interest

IMS Version 7 introduced the following enhancements that have impacted IMS connectivity.

RACF PassTicket

The RACF PassTicket is a one-time-only password that is generated by a requesting product or function. At user sign-on it is an alternative to the RACF password that removes the need to send RACF passwords across the network in clear text. RACF Version 2 Release 1, or a functionally equivalent security manager, is required on both the front-end and back-end systems.

SLUP/Finance session cold termination

When IMS is cold started, every attached SLUP session is also cold started. Thereafter, every restart is then a warm start of the terminal. There is the possibility that these sessions cannot be warm started, and that an IMS cold start is required to resolve this problem.

The IMS command **/CHANGE NODE** has a new keyword COLDSESS to allow IMS to reset the terminal to a cold state. This allows SLUP or Finance terminals to be restarted “cold” without requiring an IMS cold start. This is applicable to both ETO and static terminals.

Userid clarification

The userid clarification enhancement provides a method by which application programs can determine the nature of the value contained in the USERID field of the I/O PCB.

This provides a method that allows IMS application programs or exits to determine whether a user was signed on at the time a transaction is entered.

Deferred ACB open

When IMS is started, the VTAM ACB is opened and logon requests flow. These requests will be queued until a **/START DC** command is issued.

In some emergency restart scenarios, the **/START DC** may not be effective until all the necessary log records are processed. Some devices like ATMs may logon to IMS, and these sessions will queue until the **/START DC** has completed. This increases the possibility that these remote devices will time-out before IMS can accept the logon request.

To resolve this, the IMS parameter VACBOPEN within the DFSDCxxx member of PROCLIB has been provided. This parameter determines whether IMS will open the VTAM ACB at startup, or defer it until processing of the **/START DC** command has completed.

10.2.8 IMS Version 8 enhancements associated with IMSplex connectivity

IMS Version 8 introduced many features that are related to IMS Parallel Sysplex connectivity.

Systems management enhancements

As IMS systems are joined together into sharing groups (sharing databases, network resources, or message queues) in a sysplex environment, system management becomes more complex. Prior to IMS Version 8, the IMSs that were in sharing groups had to be managed individually. IMS Version 8 builds upon the idea of an IMS sysplex (known hereafter as an IMSplex) to help reduce the complexity of managing multiple IMSs in a sysplex environment.

An IMSplex can be defined as one or more IMS address spaces (control, manager, or server) that work together as a unit. Typically (but not always), these address spaces:

- Share either databases or network resources or message queues (or any combination)
- Run in a S/390 sysplex environment
- Include an IMS Common Service Layer (CSL - new for IMS Version 8)

Examples of IMSplexes are:

- A set of IMS control regions at the IMS Version 6, 7 or 8 level without a CSL that are sharing data or sharing message queues.
- A set of IMS control regions at the IMS Version 8 level with a CSL that are sharing data or sharing message queues.
- A single IMS control region at the IMS Version 8 level with a CSL. This still qualifies as an IMSplex because it is a set of IMS address spaces (IMS control, CQS, SCI, OM, RM) working together.

To support the IMSplex, a number of IMS functions have been enhanced and a number of new functions have been added.

- The Base Primitive Environment (BPE) has been enhanced.
- The Common Queue Server (CQS) has been enhanced.
- A new component, the Common Service Layer (CSL), is introduced consisting of the following three new address spaces:
  - Operations Manager (OM)
  - Resource Manager (RM)
  - Structured Call Interface (SCI)
- Optional resource structure.

- A TSO/ISPF based single point of control (SPOC) application program and a REXX API is shipped with IMS Version 8.
- The IMS terminal management function of IMS TM has been enhanced.
- A new global online change function has been added to coordinate global online change activities of all the IMSs in the IMSplex.

Sysplex terminal management (STM) addresses the management of a subset of IMSplex resources, primarily those defined as part of the VTAM network. Chapter 8, “Sysplex terminal management (STM)” on page 175 describes STM in detail.

Common Service Layer

The Common Service Layer (CSL) is new for IMS Version 8. The components of CSL, Operations Manager (OM), Resource Manager (RM), and Structured Call Interface (SCI), provide the infrastructure for an IMSplex. Each OM, RM, and SCI runs in a separate address space.

Structured Call Interface

The Structured Call Interface (SCI) is the part of the Common Service Layer (CSL) that provides the communications infrastructure of the IMSplex. Using the SCI, IMSplex components can communicate with each other within a single OS/390 image or across multiple OS/390 images. Individual IMSplex members do not need to know where the other members are running.

The SCI is responsible for routing requests and messages between the IMS control regions, Operations Managers (OMs) and Resource Managers (RMs). SCI also provides support for automatic RECON loss notification.

Resource Manager (RM)

For IMS Version 8, IMS Transaction Manager (TM) has been enhanced to use the new Resource Manager (RM) facility to maintain IMS resource information in a sysplex environment. By having the resource information available to other IMSs in the sysplex, the following is achievable:

- Resume work for VTAM terminals and users if their local IMS fails.
- Eliminate VTAM Generic Resources terminal affinities.
- Provide resource type consistency.
- Provide name uniqueness.
- Provide global callable services for NODE, LTERM, and user resources.

Operations Manager (OM)

The Operations Manager (OM) provides the interface for a single system image for system operations in an IMS Version 8 IMSplex. The OM provides an application programming interface (API) for the distribution of commands and responses within the IMSplex. The OM:

- Routes IMS commands to IMSplex members that are registered to process those commands.
- Consolidates command responses from individual IMSplex members into a single response for presentation to the command originator.
- Provides user exits for command and response edit and command security.

Common Queue Server (CQS) enhancements

CQS has been enhanced to support the Resource Manager access to the new (optional) resource structure. Resource Manager (RM) is a new address space introduced in IMS Version 8 and uses CQS with a new Coupling Facility list structure called a resource structure to maintain global resource information for IMSplex resources.

The CQS is also exploiting the enhancements made available to CF list structures in the recent releases of OS/390 and z/OS. These enhancements include:

- System managed rebuild of the structures
- Structure alter and autoalter
- System managed duplexing of the structures
- Structure full monitoring capability
- In addition to the functional enhancements for IMS Version 8, the Common Queue Server (CQS) information is now in a separate book from that of BPE information. For IMS Version 8, you will find all the information pertaining to CQS in the *IMS Version 8: Common Queue Server Guide and Reference,* SC27-1292.

Single point of control application

One of the new functions delivered with IMS Version 8 is the ability to manage a group of IMSs (an IMSplex) from a single point of control (SPOC). With IMS Version 8, IBM is delivering a TSO/ISPF based SPOC application. Using the ISPF SPOC application, you can:

- Issue commands to any or all IMSs in an IMSplex
- Display consolidated responses from those commands

IMS Version 8 also provides the REXX Application Programming Interface (API) as an interface to SPOC for user written automation. For details on the SPOC enhancement for user written automation, refer to *IMS Version 8:Common Service Layer Guide and Reference,* SC27-1293.

Global online change

One of the complexities of running multiple (cloned) IMS systems in an IMSplex is coordinating online change processing for all IMS systems in the IMSplex. Prior to IMS Version 8, it was necessary to perform online change on each individual IMS in the IMSplex. An important part of managing an IMSplex from a single point of control is to be able to coordinate global online change (also referred to as coordinated online change) processing among all the IMSs in the IMSplex.

For IMS Version 8, modifications have been made to coordinate the online change across multiple IMS systems, invoked by IMSplex commands, such as the INITIATE OLC.

APPC and OTMA enhancements

Prior to IMS Version 8, implicit synchronous APPC and OTMA messages could only be processed by the front-end IMS in a shared queue environment. In IMS Version 8, all implicit APPC and OTMA transactions are eligible for sysplex wide processing. Figure 10-5 presents an overview of this support.

Archive

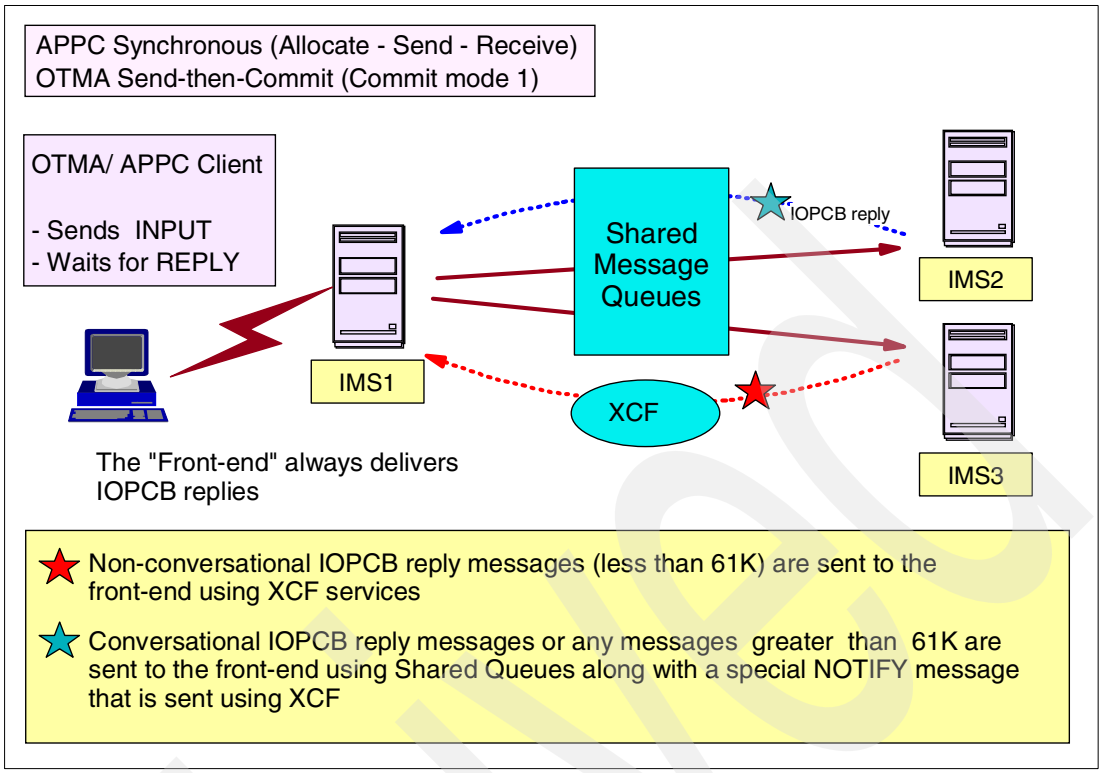


Figure 10-5 General flow of synchronous APPC or OTMA transactions with shared queues

IMS MSC FICON™ CTC support

The multiple systems coupling (MSC) feature of IMS provides a reliable, high bandwidth host-to-host communications between IMS systems. One choice for the physical host-to-host connection is to define the MSC to utilize the channel-to-channel (CTC) hardware support.

On z/Series 900 processors, CTC bandwidth can be enhanced by implementing the S/390 Fiber Connection (FICON) channel support for CTCs. This FICON support could be significantly faster than the Enterprise System Connection (ESCON) support for large blocks of data transferred. It is estimated that one FICON CHPID can do the work of a number of ESCON CHPIDs. This increased bandwidth is the result of faster data transfer rates, I/O rates, and CCW and data pipelining. The distance between hosts can also be increased.

The IMS MSC FICON CTC support increases the volume of IMS messages that can be sent between IMS systems when using the IMS MSC facility. This enhancement is activated by changing the IMS online procedure so that the DD cards specifying the CTC address for the link specifies the CTC FICON address. No changes are needed to the IMS system definition to convert existing CTC links to FICON. z/OS Version 1 Release 2 is needed for IMS MSC FICON CTC support. This capability is also enabled for IMS Version 7 by APAR PQ51769.

10.2.9 IMS Version 8 connectivity features of interest for APPC/IMS

IMS Version 8 included several enhancements for the management of APPC/IMS resources:

- A new parameter (CPUTIME) has been added to the TP_Profile data set (which is maintained by the operating system) to specify the number of CPU seconds that a CPI-C program is allowed to run before being terminated. This limits the time that resources are held due to a possible error in the program causing it to loop endlessly.

- LU 6.2 descriptors can now be added or deleted dynamically using the **/START** and **/DELETE** commands.
- New OUTBND= parameter has been added to IMS PROCLIB member DFSDCxxx. This allows you to specify an APPC/IMS LU, other than the BASE LU, to be used when establishing new outbound conversations. Before, if the BASE LU status was disabled or unavailable, outbound conversations could not be allocated even if there were other APPC/IMS LUs defined for this IMS system that were active and available.
- The **/CHANGE** command has been enhanced to allow you to change the outbound LU using the new OUTBND keyword.

Archived

11

VTAM Generic Resources

One of many benefits in a Parallel Sysplex environment is the ability to present a single system image to the end user. One of the ways to achieve this simplification is through the use of a facility introduced in ACF/VTAM Version 4 Release 2 called VTAM Generic Resources (VGR). For this facility to be used, a Parallel Sysplex environment is required as well as the use of Advanced Peer-to-Peer Networking® (APPN) within the Parallel Sysplex.

In a Parallel Sysplex environment it is possible, with VGR, to allow the end user a generic logon to IMS. The user is presented with a single view of all IMS systems in the sysplex. VGR determines the destination IMS, based on existing relationship information, availability, workload, and/or user-defined criteria such as those specified in a user exit.

Archived

© Copyright IBM Corp. 2003. All rights reserved.

11.1 VTAM Generic Resources

VGR allows the user to have a generic logon to IMS. This supplies a single view of all the IMS systems in the sysplex, with VGR determining the IMS system that the user is connected to.

This has a number of benefits. Network load balancing across the IMS systems can be achieved, and it allows for changes to the number of IMS systems in a sysplex. IMS systems can be added without impact and removed with only the users having to logoff and logon.

Figure 11-1 presents the major components of VGR that we discuss in this chapter.

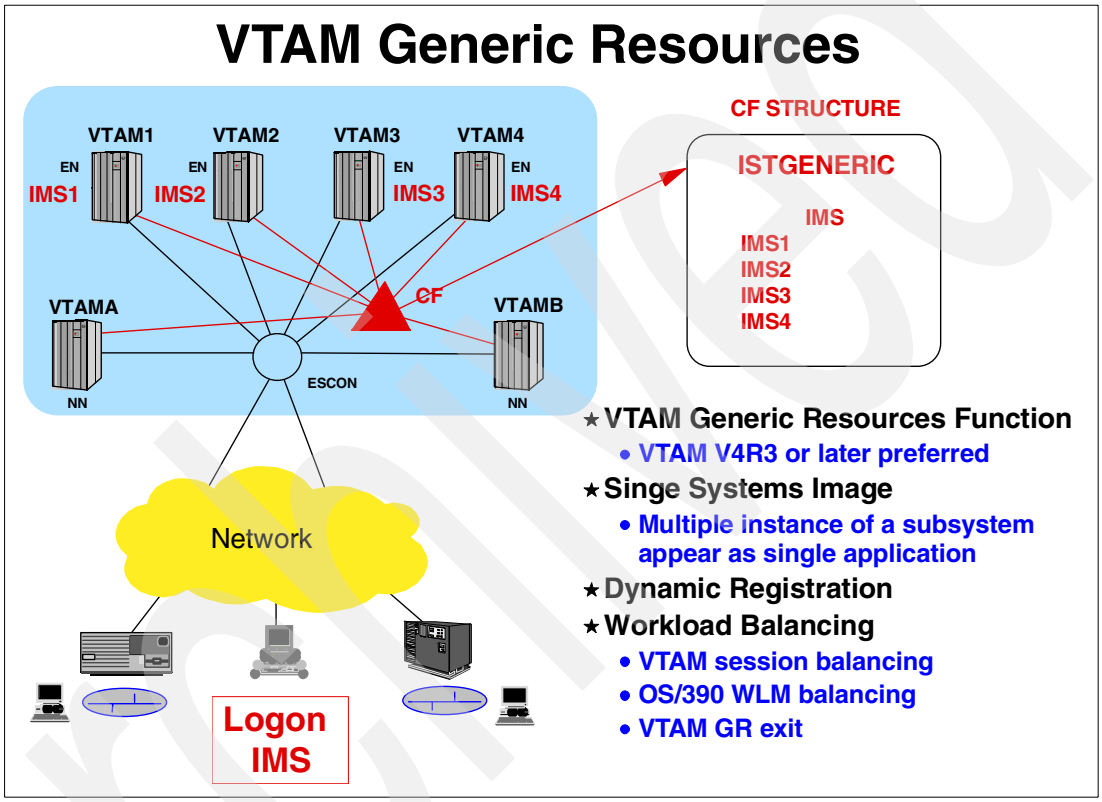


Figure 11-1 VGR overview

11.2 Terminology used for VTAM Generic Resources

Here is some of the essential terminology you need to know when talking about VTAM Generic Resources:

- Generic resource name

  A generic resource name (GR name) is defined in the IMS startup parameters, and is the name used when logging on to the generic resource group.

- Generic resource group

  A generic resource group (GRG) is made up of one or more IMS systems that have the same generic resource name. Any IMS system that has the same generic resource name is said to be a member of that generic resource group.

- Generic resource member

  A generic resource member (GR member) is an IMS system belonging to a generic resource group.

- Affinity

  A terminal currently in session with a system in a generic resource group is said to have an affinity to that system. A terminal with a significant status in IMS, such as a terminal operating in conversational mode, is said to have an affinity in VTAM to that IMS, even if the terminal is not currently in session with IMS.

  A terminal that has a requirement to connect to a particular IMS due to protocol or integrity is said to have a session affinity to that IMS. The affinity remains after session termination.

- ACB

  ACB is the term used to describe the application program network node name to VTAM. It is the name that is used to establish a session with a specific IMS system in a non-VGR environment. Even in a VGR environment, if an end user chooses to establish a connection with a specific IMS then they can do so by using the ACB name rather than the VGR name in session establishment. The IMS ACB is defined using the APPLID parameter in the system generation which can be overridden by the APPLID1 parameter in IMS startup.

When a user logs onto the GRSNAME, VTAM decides which IMS to connect the user to and adds this information into an affinity table within the ISTGENERIC Coupling Facility structure.

11.3 VTAM Generic Resources in a sysplex environment

VTAM provides a single image view of multiple subsystems across OS/390 images in a sysplex environment. VGR is a VTAM facility that allows logon through a generic name that maps to one or more VTAM applications.

VGR allows the user to use a single VTAM logon name or APPLID to access a group of IMS systems in a sysplex called the Generic Resource Name or GRSNAME. The GRG is created by VTAM when the first IMS system is started in the sysplex. As more IMS systems get started, they join the GRG.

VTAM maintains a GRG member list in a list structure, ISTGENERIC, within the Coupling Facility. The ISTGENERIC structure is defined in the CFRM policy. Roughly 300 bytes per session is required for defining the structure. It is possible for a single LU to have multiple IMS sessions and therefore to use more storage in the structure.

The ISTGENERIC structure keeps track of which terminal is connected to which real ACB by means of a partner or affinity list. This structure contains:

- A member list that consists of group name, member name (APPLID), session counts, and other VTAM information.
- An affinity table is also contained within the structure defining the member name and LU name.

Figure 11-2 shows a conceptual layout of the ISTGENERIC structure containing information about VTAM Generic Resources. The structure is logically divided into two sections: a group member list and an affinity table.

Archive

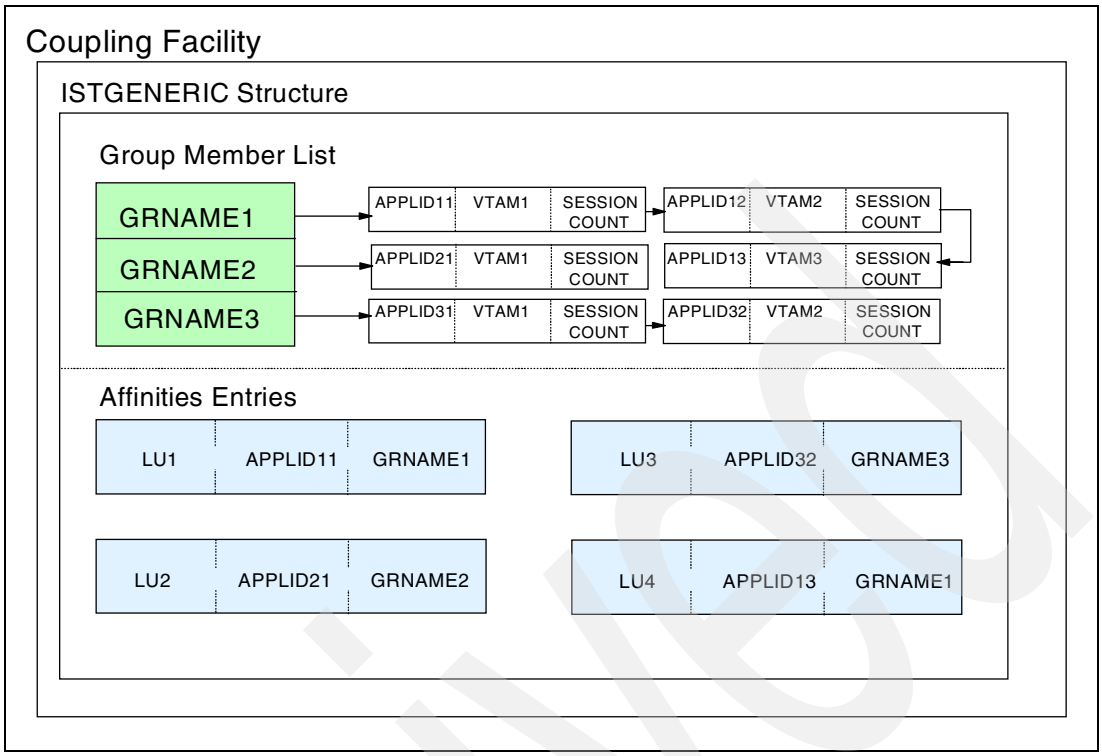


Figure 11-2 ISTGENERIC structures

The group member list is made up of three generic resource groups, with the names of GRNAME1, GRNAME2 and GRNAME3. The generic resource group GRNAME1 has three generic resource members, APPLID11, APPLID12 and APPLID13. GRNAME2 has one member, APPLID21, and GRNAME3 has two members, APPLID31 and APPLID32. The session counts of all application programming network nodes are recorded in the structure to be used by VTAM for determining which generic resource member a session should be established with, if the session was requested with a generic resource name. The local VTAM name is associated with each member of the generic resource group.

LU1 requested a session using the generic resource name GRNAME1. VTAM could connect the session to any generic resource member of that group; namely APPLID11, APPLID12, or APPLID13, based on the information in the group member list. VTAM elected to connect LU1 to APPLID11 based on the session selection criteria used, and an entry was created in the affinity list with the relevant session establishment information. Similarly:

- LU2 requested a session with generic resource group GRNAME2 and the session was connected to APPLID21.
- LU3 requested a session with GRNAME3 and was connected to APPLID32.
- LU4 requested a session with GRNAME1 and was connected to APPLID13.

11.4 Generic resources affinity

With the original VGR implementation in IMS, IMS takes the responsibility of managing the affinity. So, at session termination, which can occur for a number of reasons, including user logoff, IMS shutdown or failure, network or OS/390 failure, IMS decides whether the affinity is to be deleted. The affinity is deleted at session termination except for these connected sessions:

- ISC parallel sessions
- SLUP terminals
- ETO users
- Any LU with significant status. This can include:
  - Terminal in response mode
  - Terminal in conversational mode
  - Exclusive (**/EXC**)
  - Test (**/TEST**)
  - Preset destination (**/SET xxxxxxxx**)
  - MFS test (**/TEST MFS**)
  - Financial / SLUP (LU0) devices
  - OS/390 image failure

Therefore, if IMS fails, without being able to control the shutdown (OS/390 or processor failure), IMS will not delete any of these affinities. So if there is an attempt to logon following this type of failure, then the logon would be directed towards the failed IMS, and this would also fail. The user has the option of waiting for that IMS to come back up, or to log on directly to another IMS.

If the user logs on directly to another IMS, then that user's status will not be known and cannot be restored. While this still might be desirable for availability reasons (don't want to wait for a failed IMS to be restarted), not only is the status lost but when that user later logs back on to the original IMS, the original status will be restored, possibly causing great confusion.

Therefore, there is a requirement to allow VTAM to reset affinity and allow a logon to another IMS in the sysplex.

Affinities are deleted during normal or abnormal session termination unless one or more of the following is true:

- The terminal retains significant status. Significant status is set for a terminal if it is in response or conversational mode or when the terminal is in some preset state or mode of operation, such as MFSTEST, test, preset, or exclusive. Significant status is always maintained for a static terminal whether its session is active or not. It is maintained for a dynamic (ETO) terminal when the user is signed on and is reset when the user is signed off (normally or abnormally).
- Session termination is for a SLUTYPEP or FINANCE terminal.
- Session termination is for the last ISC parallel session with a given node and not all parallel sessions to that node have been quiesced (including this last session).

The type of system failure, IMS or OS/390, also determines whether affinities are released when the failure occurs. When the failure is an IMS failure, IMS's ESTAE processing releases all affinities unless the terminal has significant status, is a SLUTYPEP or FINANCE device, or is associated with an ISC connection where all parallel sessions have not been quiesced.

If sessions are lost because of an OS/390 failure, IMS's ESTAE routine does not get control, and affinities are not deleted regardless of whether significant status is held. Two courses of action are available for session re-establishment when sessions are terminated because of an OS/390 failure:

- First, the end user can log on using the specific IMS APPLID of one of the surviving IMSs.
- Or, the end user must wait until the failed IMS is restarted and then log on with the generic resource IMS name. This IMS restart can occur on any of the surviving OS/390s in the

Parallel Sysplex. SLUTYPEP and FINANCE terminals, most likely, will not be able to request a session using a specific IMS APPLID if they are already programmed to request a session using a generic resource name.

The advantage of this implementation of VTAM Generic Resources by IMS is that affinities are retained if significant status is held, and will be returned to the state it was in prior to session termination.

11.4.1 VTAM Generic Resources affinity management

Prior to IMS Version 8, VGR affinity management was determined by the values of the GRAFFIN and GRESTAE parameters in DFSDCxxx at IMS initialization.

GRAFFIN has two possible values:

- **GRAFFIN=IMS**

  When set to IMS (which is the GRAFFIN default), IMS decides whether or not to delete the VGR affinity when the session terminates. IMS makes this decision based on whether or not the terminal has significant status at the time of session termination. The intent is to force (or at least encourage) the user to log back on to the same IMS which has knowledge of some significant status, such as conversational status.

  Table 11-1 distinguishes different terminal types and when or if affinities are deleted.

Table 11-1 Affinity actions with GRAFFIN = IMS at the IMS Version 7 level

Terminal Type	Exceptions	Affinity deleted at	If
Static terminals	SLUP/Finance/ISC	Session termination --- never	No significant status (see note) --- Status exists
ETO terminals	SLUP/Finance/ISC	Session termination	----
ISC sessions	----	Last parallel session --- never	Cold termination --- If no cold termination
SLUP/Finance (static and ETO)	/CHA NODE xxx COLDSESS	never	----
SLUP/Finance (static and ETO)	----	/CHA NODE xxx COLDSESS	Out of session and idle
All	----	IMS shutdown	IMS shutdown with LEAVEGR
All	----	IMS restart	Cold start of COLDCOMM

Note: User signoff (DFSSGFX0) or logoff (DFSLGFX0) exits can request that IMS reset any significant status. This includes the Fast Path response mode status.

- **GRAFFIN=VTAM**

  When set to VTAM, the following occurs on session termination:

  – IMS automatically insures that all terminal status conditions are reset for sessions using the IMS generic resource name before the signoff or logoff exits are driven, or in the case of a failed or restarted IMS, before the next logon or signon.

- ISC sessions are an exception to resetting, since IMS continues to manage affinities for ISC sessions.
- Since MSC does not use VGR because sessions specify APPLIDs, there is no association to MSC links.
- SLUTYPEP and FINANCE terminal affinities, significant status, and sequence numbers are reset upon session termination. These terminal types are free to immediately log back on using the generic resource name no matter what the cause of the session termination. Session re-establishment, however, will be cold.

Table 11-2 An affinity activity chart for GRAFFIN = VTAM users

Terminal type	Exceptions	Affinity deleted at
IMS terminals	ISC	Session termination
ISC	----	IMS decision since affinity is not managed by VTAM

- **GRESTAE=N or Y**

  This option only applies if (GRAFFIN=IMS) has been set. It defines whether or not IMS should reset affinities where no status remains and issue CLSDSTs for all nodes during ESTAE processing before closing the VTAM ACB (GRESTAE=Y), or close the VTAM ACB immediately to expedite IMS termination and leave the affinity for all nodes set (GRESTAE=N).

The planning consideration here is possibly slow IMS termination (GRESTAE=Y), versus continued terminal affinity to the failed IMS (GRESTAE=N).

11.4.2 VTAM Generic Resources affinities and IMS Version 8

VGR affinities have the same purpose in IMS Version 8 as in prior versions — to force (encourage) a user logging on to return to that IMS which has knowledge of that user's significant status. However, in IMS Version 8, with sysplex terminal management and status recovery mode (SRM) of GLOBAL, when a resource structure is available, significant status is known by all the IMSs in the IMSplex. It is therefore unnecessary to use VGR affinity to direct a user's logon request back to the same IMS where the session was terminated with significant status.

When SRM=LOCAL, only the original IMS knows that status and it is necessary to return to the original IMS to restore that status. When SRM=NONE, we don't care about the status, so again it is not necessary to return to the same IMS.

VTAM Generic Resources support in IMS Version 8 depends on the level of the operating system — OS/390 or z/OS.

VGR support when running with OS/390 V2 R10

The installation still specifies the GRAFFIN parameter, and this parameter is still a system-wide value determining whether IMS or VTAM will manage all of the VGR affinities. When GRAFFIN=VTAM, all affinities will be deleted when a session terminates. When GRAFFIN=IMS, IMS will determine whether or not to delete the VGR affinity. But in IMS Version 8, this decision is based not just on whether significant status exists, but also on the SRM setting.

- SRM=GLOBAL or SRM=NONE
  - GRAFFIN=IMS or GRAFFIN=VTAM

With any combination of the above, the VGR affinity will be deleted at session termination.

- SRM=LOCAL
  - GRAFFIN=IMS

    If significant status exists at session termination, IMS will not delete the VGR affinity. The next generic logon will go back to the same IMS and its recoverable status will be restored from local control blocks.
  - GRAFFIN=VTAM

    With or without significant status, VGR affinity will be deleted by VTAM, because we don't know that significant status exists. It will be shown later, however, that if significant status does exist for the user, even without VGR affinity, a Resource Manager (RM) affinity will still exist.

VGR support when running with z/OS V1R2 or higher

Beginning with z/OS Version 1 Release 2, VTAM supports session level affinities. This means that, as each NODE logs on, IMS can decide whether VTAM or IMS is to manage the VGR affinities. The GRAFFIN parameter is no longer required and will be ignored if still included. Like with OS/390, there are some rules about how IMS sets the affinity management attribute.

VTAM managed affinities

When this is set, VTAM will delete the affinity whenever the session terminates. The next time the user logs on, VTAM may route the logon request to any available IMS in the VGR group. VTAM management of affinities occur for:

- All terminals with SRM=GLOBAL or NONE

  In this case, we either have the status in the resource structure, or we don't care about recovering the status. So, we don't care which IMS the user logs on to.
- All ETO non-STSN terminals

  For ETO non-STSN terminals, the SRM is not set until the user signs on. But the VGR affinity management attribute must be set at logon time. Not knowing what the SRM value will be, IMS assumes GLOBAL (or NONE) and sets it to VTAM. When the session terminates, IMS will disconnect the user structure from the terminal structure and VTAM will delete the affinity. Any end-user significant status stays with the user, not the NODE. If SRM had been set to LOCAL, that user could log on to another. This would be allowed because there is no RM affinity for the NODE. However, when the user tries to sign on, IMS would find the RM affinity for the user (and LTERM) and reject the signon.

IMS managed affinities

When this is set, IMS will decide at session termination whether or not to delete the affinity. IMS uses end-user significant status to make this determination. If it exists, then IMS will not delete the VGR affinity. The next generic logon will be routed to the same IMS.

- All static terminals with SRM=LOCAL

  When SRM=LOCAL, end-user significant status is known only to the local IMS.
- Dynamic STSN terminals with SRM=LOCAL

  Unlike ETO non-STSN terminals, for ETO STSN terminals, the SRM is set at logon time, so IMS knows whether SRM is LOCAL or not. If it is LOCAL, then IMS will want to manage the VGR affinity, because it will be the only IMS to know whether the user has end-user significant status.

Table 11-3 on page 247 summarizes how IMS sets VGR affinity management.

Table 11-3 VGR affinity management

Terminal Type	SRM	Managed by
Static	Global, None	VTAM
Dynamic STSN	Global, None	VTAM
Dynamic non-STSN	Any	VTAM
Static	Local	IMS
Dynamic STSN	Local	IMS

The existence of command significant status when a resource structure exists does not affect the deletion of a VGR affinity. Because command significant status is always GLOBAL when a structure exists, command significant status can always be recovered on another IMS in the IMSplex. When no resource structure exists, then command significant status is always LOCAL and will prevent the deletion of a VGR affinity.

The use of VTAM Generic Resources by IMS has proven to be a very effective facility in the IMSPlex environment.

Archived

Archived

12

Using Automatic Restart Manager (ARM) for IMS

This chapter describes the Automatic Restart Manager (ARM) and how it can be used with IMS in the Parallel Sysplex environment.

Automatic Restart Manager (ARM) is a facility of OS/390 introduced in MVS/ESA SP™ 5.2. It can be used to automatically restart failed IMS, online subsystems and certain other jobs and tasks in a Parallel Sysplex.

In an IMS shared queues environment, it is important to restart failed IMS subsystems as quickly as possible. Failed subsystems may hold locks protecting updates. Requests for these locks result in lock reject conditions, which usually cause abends of application programs. Emergency restart releases these locks when it backs out in-flight work. These backouts eliminate any further lock rejects. ARM may be used to invoke emergency restarts more quickly and, therefore, provide greater availability.

ARM distinguishes between abends of a job or task and failures of an entire OS/390 system. If a job or task fails, ARM may be used to automatically restart it on the same system. If a system fails, ARM may be used to restart the work on other systems in the Sysplex.

Only those jobs or started tasks which register to ARM are eligible to be restarted by ARM. IMS control regions, IRLMs, FDBR, and CQS may do this registration. For control regions, FDBR, and CQS, it is optional. For the IRLM, it always occurs when ARM is active. IMS dependent regions (MPR, BMP, and IFP), IMS batch jobs (DLI and DBB), IMS utilities, and the online DBRC and DL/I SAS regions do not register to ARM.

There is no need for ARM to restart online DBRC and DL/I SAS regions, because they are started internally by the control region.

ARM is optional. If used, it must be defined in an ARM policy, and the policy must be activated. The policy is defined using the OS/390 Administrative Data Utility. It is activated by a SETXCF command. ARM support in IMS was implemented through APAR PN71392 for IMS/ESA Version 5.1. With this support, IMS defaults to the use of ARM when run under an OS/390 using ARM.

© Copyright IBM Corp. 2003. All rights reserved.

12.1 ARM considerations

IMS regions that support ARM have an ARMRST parameter. If ARMRST=N is specified, the region does not register with ARM.

When ARM restarts an IMS online system, IMS will always use AUTO=Y. This is true even if AUTO=N is specified. The use of AUTO=Y eliminates the need for the operator to enter the **/ERE** command.

When ARM restarts an IMS online system, IMS specifies the OVERRIDE parameter when appropriate. This eliminates the requirement for an operator action during automatic restarts.

12.1.1 Exceptions to automated restarts of IMS

An IMS online system will not be automatically restarted by ARM in the following situations:

- An ARM policy is not active.
- There is no available system for the restart.
- ARMRST=N is specified for the IMS control region.
- IMS is normally terminated. This is done by a **/CHE PURGE**, **FREEZE**, or **DUMPQ** command.
- IMS is terminated by an OS/390 cancel unless ARMRESTART is specified on the CANCEL or FORCE command.
- Any of the following IMS abends occur:
  - U0020 - MODIFY
  - U0028 - /CHE ABDUMP
  - U0604 - /SWITCH SYSTEM
  - U0758 - QUEUES FULL
  - U0759 - QUEUE I/O ERROR
  - U2476 - CICS TAKEOVER
- IMS abends before it completes restart processing. This is used to avoid recursive abends since another restart would, presumably, also abend.

12.1.2 Restart conditions

The ARM policy controls under which conditions a job or task is restarted. An installation may choose to have the job or task restarted on both abends and system failures or only on abends. The TERMTYPE parameter is used for this specification. TERMTYPE(ALLTERM) specifies that restarts will occur for either abends or system failures. TERMTYPE(ELEMTERM) specifies that restarts will occur only after abends. ALLTERM is the default.

12.1.3 Restart groups

An installation defines jobs or tasks in a restart group. They are called elements. When a system failure occurs, ARM restarts all elements in a restart group on the same LPAR. Of course, only those elements for which TERMTYPE(ALLTERM) has been specified will be restarted. A restart group might be composed of an IMS DBCTL control region and all of the CICSs which attach to it. Another example is an IMS DB/DC control region and the DB2 to which it attaches.

Archived

Restart groups are defined in the ARM policy for the Administrative Data Utility. The policy identifies which elements are in a restart group, where they may be restarted, and the command or JCL used to restart them.

12.1.4 Specifying the restart method

ARM provides three choices for method used to restart an element. They are:

- Use the same JCL that previously started the element.
- Restart the element as a batch job using JCL in a data set specified in the policy.
- Restart the element by issuing the command that is specified in the policy.

The RESTART_METHOD parameter for the element is used to specify the choice.

For most installations, using the same JCL will be appropriate. This is the PERSIST specification.

12.1.5 A restart group example

Example 12-1 is an example of statements used to define the elements in a restart group and the OS/390 systems on which they may execute. PD11 is an IMS DBCTL system. PA11, PA12, and PA13 are CICS AORs that use PD11. PT11 and PT12 are CICS TORs which use these CICS AORs. This installation has named its restart groups using the OS/390 system on which the group normally runs and an identification of the data-sharing group. The OS/390 system is SYSA and the data-sharing group is IMSP. This leads to name SYSA_IMSP for the restart group.

The restart group includes an IRLM; however, it is only restarted after abends. It is not moved with the group when its system fails.

Example 12-1 Restart group definitions for ARM

```
RESTART_ORDER
LEVEL(1)
ELEMENT_NAME(PD11)IMS
LEVEL(2)
ELEMENT_NAME(PA1*)CICS AORS
LEVEL(3)
ELEMENT_NAME(PT1*)CICS TORS
RESTART_GROUP(SYSA_IMSP)
TARGET_SYSTEM(SYSA,SYSB,SYSC,SYSD)
ELEMENT(PD11)IMS
RESTART_METHOD(BOTH,PERSIST)
ELEMENT(PA1*)CICS AORS
RESTART_METHOD(BOTH,PERSIST)
ELEMENT(PT1*)CICS TORS
RESTART_METHOD(BOTH,PERSIST)
ELEMENT(PI01)IRLM
TERMTYPE(ELEMTERM)
RESTART_METHOD(BOTH,PERSIST)
```

If SYSA fails, PD11, PA11, PA12, and PA13 will be restarted on either SYSB, SYSC, or SYSD. They will be restarted using the same JCL that was used for the failed execution. PD11 will be restarted before any of the CICS AORs, and the AORs will be restarted before the TORs. This installation has an IRLM with the same name executing on each of the systems. It does not need to restart the IRLM on OS/390 failures. It has specified TERMTYPE(ELEMTERM) to avoid these restarts. Other installations may have other circumstances. If a group may be

restarted on an LPAR where an IRLM is not already executing, a specification of TERMTYPE(ALLTERM) for the IRLM may be used to restart the IRLM also.

12.1.6 Other ARM capabilities

ARM has options to control the order in which elements are started, the number of attempts made per element, the timing of these attempts, and other actions taken upon failures. A complete description of ARM policy specifications, including these options, appears in *z/OS MVS Setting Up a Sysplex,* SA22-7625.

12.1.7 Restarting dependent regions

ARM will not restart IMS dependent regions (MPRs, BMPs, and IFPs). These must be started by some other means. Other automated operations facilities, such as NetView or IMS Time-Control Option (TCO) may be used.

12.1.8 ARM with IRLM

IRLM added ARM support via APAR PQ06465. If an ARM policy is active, IRLM always registers once this maintenance is applied. When using ARM to restart IRLM, module DXRRL0F1 must be in the OS/390 linklist. If it is not, restarts of IRLM by ARM will fail.

12.1.9 Restarting after IRLM abends

If an IRLM abends, ARM will restart it on the same LPAR. In the meantime, IMS will quiesce. This includes terminating any in-flight units of work and discontinuing scheduling. When this completes, IMS will attempt to reconnect to its IRLM. Since restarting the IRLM is a very quick activity, ARM will have restarted the IRLM before IMS attempts the reconnection. So, the reconnection of IMS to its IRLM will be automatic. If the same IRLM name is used by all IRLMs in a data-sharing group, restarting IRLMs for system failures is not required. The installation may always have an IRLM executing on each target system for the restart group. This will ensure that an IRLM with the correct name is available for any IMS control region restarts. In this case, the ARM policy should include a TERMTYPE(ELEMTERM) parameter for the IRLMs. This will cause ARM to restart the IRLMs only on abends.

If different IRLM names are used by IRLMs in a data-sharing group, an installation should place each IMS and its IRLM in a restart group. This will ensure that they are restarted on the same LPAR. Of course, each pair of IMS and IRLM systems would have its own restart group. In this case, the ARM policy should include a TERMTYPE(ALLTERM) parameter for each IRLM. This will cause ARM to restart the IRLMs on both abends and system failures.

12.1.10 ARM with CQS

CQS includes ARM support. If ARM is active and ARMRST=N is not specified in the CQS procedure, CQS registers with ARM. ARM will restart it if CQS abends or if its OS/390 system fails. There are some exceptions to ARM restarts after CQS abends. Module CQSARM10 contains a table of CQS abends for which ARM restarts will not be done. Users may modify this table. The table is shipped with the following abends:

- 0001 - ABEND during CQS initialization
- 0010 - ABEND during CQS initialization
- 0014 - ABEND during CQS initialization
- 0018 - ABEND during CQS restart
- 0020 - ABEND during CQS restart

Because a CQS must execute with its IMS system, it should be included in a restart group with its IMS.

12.1.11 Information for ARM policies

The Element name is the IMSID

- May be used in ELEMENT(imsid) and ELEMENT_NAME(imsid) in ARM policy.

The Element Type is SYSIMS

- May be used in ELEMENT_TYPE(SYSIMS) in ARM policy.

Note: For IMS Version 8, the CQS ARM element name is changed from cqssn +'CQS' to 'CQS' + cqsssn + 'CQS'. Any installation that defines an Automatic Restart Manager Policy for CQS must change CQS's ARM element name in the ARM policy. For example, in IMS Version 7, if SSN=CQS1A, the ARM element name is CQS1ACQS. In IMS Version 8, the ARM element name is CQSCQS1ACQS.

12.1.12 ARM and CSL

A CSL address space (OM, RM, SCI), if requested, can register with the OS/390 Automatic Restart Manager (ARM). The ARM is an OS/390 recovery function that can improve the availability of started tasks. When a task fails or the system on which it is running fails, the ARM can restart the task without operator intervention. IBM provides policy defaults for automatic restart management. You can use these defaults, or you can define your own ARM policy to specify how CSL address spaces should be restarted. The ARM policy specifies what should be done if the system fails or if a CSL address space fails. ARMRST=Y is the default for all three address space. To enable or disable the ARM explicitly, you can specify ARMRST=Y or N in one of two ways:

- In the CSL address space initialization PROCLIB member:
  - CSLSIxxx for SCI
  - CSLRIxxx for RM
  - CSLOIxxx for OM
- As an execution parameter

When ARM is enabled, the CSL address spaces register to ARM with an ARM element name. The ARM element names are defined in Table 12-1.

Table 12-1 ARM element names

CSL Address Space	ARM element name
OM	"CSL" + omname + "OM"
RM	"CSL" + rmname + "RM"
SCI	"CSL" + sciname + "SC

Note: The name of the CSL address space is the name defined either as an execution parameter, or in the initialization PROCLIB member of that CSL address space. For example, if OMNAME=OM1A in the CSLOIxxx PROCLIB member, the ARM element name is CSLOM1AOM.

Archived

12.1.13 ARM and FDBR

ARM support is provided for both IMS and FDBR.

ARM support for IMS with FDBR active

ARM support is available for IMS through the “ARMRST=Y” execution parameter. When an FDBR system using ARM tracks an IMS system, it notifies ARM that it is doing the tracking and that ARM should not restart this IMS if it fails. FDBR uses the ASSOCIATE function of ARM to make this notification. So, even if IMS registers to ARM, IMS will not be restarted after its failures when FDBR is active and using ARM.

If FDBR is terminated normally, it notifies ARM. This tells ARM to restart the tracked IMS if this IMS has previously registered to ARM. This is appropriate since FDBR is no longer available to do the recovery processes.

When ARM is used for IMS, an installation can choose either to use or not to use ARM for FDBR. If FDBR does not use ARM, it cannot tell ARM not to restart IMS. In this case, a failure of IMS will cause FDBR to do its processing and ARM to restart IMS. This could be advantageous. One would expect FDBR processing to complete before the restart of IMS by ARM completes. If so, locks would be released quickly by FDBR and a restart of IMS by ARM would occur automatically. Since these actions depend on the timing of FDBR processing and IMS restart processing, they cannot be guaranteed.

ARM support for FDBR

FDBR can register to ARM. This is controlled by the “ARMRST” execution parameter. The default is to register. “ARMRST=N” causes FDBR not to register.

When FDBR registers to ARM, it is restarted automatically if it fails. Failures include abends of FDBR and failures of the OS/390 system on which it is executing.

When IMS registers with ARM, it uses its “IMSID” for its ARM element name. Similarly, when FDBR registers with ARM, it uses its IMSID for its ARM element name. Because ARM element names must be unique, the IMSIDs of an IMS system and an FDBR region cannot be the same.

12.1.14 Comparison of FDBR, XRF, and ARM

FDBR, XRF, and ARM can be used to back out the in-flight work of a failed IMS system and release its locks. Each has its own advantages.

12.1.15 FDBR advantages and disadvantages

FDBR has the following advantages:

- FDBR begins its backout processing sooner than an IMS restart initiated by ARM. FDBR is tracking the active IMS by reading its log. This eliminates most of the log reading that is required in an ARM-initiated restart.
- FDBR requires no operator intervention when IMS runs as a started task. XCF monitoring is used to inform FDBR that IO prevention is complete.
- FDBR uses less CPU and storage than XRF during the tracking phase.
- With IMS Version 8, FDBR modules were moved from CSA to private storage, or from extended CSA to extended private storage. This CSA/ECSA storage relief is received for each FDBR product on the system. 378 KB has been moved from CSA to private and 420 KB has been moved from ECSA to extended private.

- FDBR registers and interacts with the IMS Version 8 Structured Call Interface (SCI).

FDBR has the following disadvantages:

- FDBR does not restart the failed IMS.
- FDBR does not invoke the following recovery processes:
  - Resolution of in-doubt threads with CICS and DB2
  - Recovery of IMS message queues
  - Recovery of MSDBs
  - Release of DBRC database authorizations

12.1.16 XRF advantages and disadvantages

XRF has the following advantages:

- XRF begins its database recovery processing sooner than an IMS restart initiated by ARM. XRF is tracking the active IMS by reading its log. This eliminates most of the log reading that is required in an ARM-initiated restart.
- XRF can resolve in-doubt threads with CICS and DB2. This requires that the CICS and DB2 subsystems be restarted on the OS/390 where the XRF alternate executes.
- XRF handles the recovery of IMS message queues.
- XRF moves terminal sessions to the new active system.
- XRF recovers MSDBs.
- XRF accepts new work on the new active system.
- XRF handles DBRC database authorizations by having the alternate assume the authorizations of the failed active.

XRF has the following disadvantages:

- XRF requires more CPU and storage than FDBR while in tracking phase.
- Takeover on another CPU requires operator intervention to indicate that IO prevention is complete.

12.1.17 ARM advantages and disadvantages

ARM has the following advantages:

- ARM requires no resources for tracking.
- ARM-initiated restarts can resolve in-doubt threads with CICS and DB2. ARM can automatically move IMS, CICS, and DB2 in a group to the same OS/390.
- ARM-initiated restarts do not require operator intervention.
- ARM-initiated restarts handle the recovery of IMS message queues.
- ARM-initiated restarts recover MSDBs.
- ARM-initiated restarts accept new work on the new active system.
- ARM-initiated restarts release DBRC database authorizations during emergency restart processing.

ARM has the following disadvantages:

- ARM-initiated restarts begin the database recovery processes later than FDBR or XRF would.

Archived

Each installation will have to weigh the advantages and disadvantages of these three components before making the selection that is right for them.

Archived

Abbreviations and acronyms

ACB	application control block
AGN	application group name
AOI	automated operator interface
APAR	authorized program analysis report
APF	authorized program facility
APPC	advanced program to program communication
ARLN	automatic RECON loss notification
ARM	Automatic Restart Manager
AWE	Asynchronous Work Element
BMP	batch message program
BPE	Base Primitive Environment
BSDS	boot strap data set
CBPDO	custom built product delivery offering
CI	control interval
CICS	Customer Information Control System
CQS	Common Queue Server
CSA	common system area
CSI	consolidated software inventory
CSL	Common Service Layer
CST	Consolidated Service Test
CTC	channel-to-channel
DASD	direct access storage device
DB/DC	database/data communications
DB2	DATABASE 2
DBA	database administrator
DBCTL	database control
DBD	database description
DBDS	database data set
DBRC	database recovery control
DEDB	data entry database
DL/I	Data Language/I
DLI/SAS	DL/I separate address space
DLT	database level tracking (RSR)
DRA	database resource adapter
ECSA	extended common system area
EMCS	extended multiple consoles support
EMEA	Europe, Middle East and Africa
ESAF	external subsystem attach facility
ESCON	Enterprise System Connection
ESO	Extended Service Offering
ETO	Extended Terminal Option
EX	execution (IVP)
FDBR	Fast Database Recovery
FICON	Fiber Connection
FMID	function modification identifier
FT	file tailoring (IVP)
FTP	File Transfer Protocol
GSAM	generalized sequential access method
HALDB	High Availability Large Database
HFS	hierarchical file system
HLQ	high-level qualifier
HTML	Hyper Text Markup Language
HTTP	Hyper Text Transfer Protocol
IBM	International Business Machines Corporation
IFP	IMS Fast Path program
ILS	Isolated Log Sender
IMS	Information Management System
IPCS	Interactive Problem Control System
IPL	initial program load
IRLM	Internal Resource Lock Manager
ISC	intersystem communication
ISD	independent software delivery
ISPF	Interactive Systems Productivity Facility
ITSO	International Technical Support Organization
IVP	installation verification program
J2C	J2EE Connector Architecture
J2EE	Java 2 Platform, Enterprise Edition
JBP	Java batch processing region
JCL	job control language
JDBC	Java database connectivity
JDK	Java Development Kit
JES	job entry subsystem (JES2 or JES3)
JMP	Java message processing region
JVM	Java Virtual Machine

© Copyright IBM Corp. 2003. All rights reserved.

KSDS	key sequenced data set
LE	Language Environment
LMOD	load module
LPA	link pack area
LPAR	logical partition
LTERM	logical terminal
LU	logical unit
LU2	Logical Unit 2
MCS	multiple consoles support
MFS	message format services
MLPA	modifiable link pack area
MNPS	Multi-Node Persistent Sessions
MOD	message output descriptor (MFS)
MOD	module (SMP/E)
MPP	message processing program
MPR	message processing region
MSC	multiple systems coupling
MSDB	Main Storage Database
MVS	Multiple Virtual System
ODBA	open database access
OLDS	online log data set
OM	Operations Manager
ORS	Online Recovery Service
OSAM	overflow sequential access method
OTMA	open transaction manager access
OTMA C/I	OTMA callable interface
PCB	program communication block
PDS	partitioned data set
PDSE	partitioned data set extended
PIA	package input adapter
PIC	Product Introduction Centre
PMR	problem management record
PPT	program properties table
PSB	program specification block
PSP	preventive service planning
PST	program specification table
PTF	program temporary fix
QPP	Quality Partnership Program
RACF	Resource Access Control Facility
RDS	restart data set
RIM	related installation material
RLDS	recovery log data set
RLT	recovery level tracking (RSR)
RM	Resource Manager
RMF	Resource Measurement Facility
RNR	Rapid Network Recovery
RRS	Resource Recovery Service
RSR	Remote Site Recovery
RSU	recommended service upgrade
SAF	Security Authorization Facility
SCI	structured call interface
SDM	System Data Mover
SDSF	spool display and search facility
SLDS	system log data set
SMP/E	System Modification Program/Extended
SMQ	shared message queues
SMU	security maintenance utility
SPOC	single point of control
SRDS	structure recovery data set
SSA	sub-system alias
SVC	supervisor call
SVL	Silicon Valley Laboratories
TCB	task control block
TCO	time controlled operations
TCP/IP	Transmission Control Protocol/Internet Protocol
TMS	Transport Manager System
TPNS	Teleprocessing Network Simulator
TSO	Time Sharing Option
USS	Unix System Services
USS	unformatted system services (SNA)
VG	variable gathering (IVP)
VOLSER	volume serial (number)
VSAM	virtual storage access method
VSCR	Virtual Storage Constraint Relief
VSO	Virtual Storage Option (DEDB VSO)
VTAM	virtual telecommunication access method
WADS	write ahead data set
WAS	WebSphere Application Server
WWW	World Wide Web
XML	eXtensible Markup Language
XRC	eXtended Remote Copy
XRF	eXtended Recovery Facility
WLM	Workload Manager

Archived

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

IBM Redbooks

For information on ordering these publications, see “How to get IBM Redbooks” on page 260. Note that some of the documents referenced here may be available in softcopy only.

- *IMS Primer,* SG24-5352
- *IMS/ESA V6 Parallel Sysplex Migration Planning Guide for IMS TM and DBCTL,* SG24-5461
- *IMS Version 7 High Availability Large Database Guide,* SG24-5751
- *IMS Version 7 Release Guide,* SG24-5753
- *A DBA’s View of IMS Online Recovery Service,* SG24-6112
- *IMS Version 7 Performance Monitoring and Tuning Update,* SG24-6404
- *IMS e-business Connectors: A Guide to IMS Connectivity,* SG24-6514
- *IMS Version 7 Java Update,* SG24-6536
- *Ensuring IMS Data Integrity Using IMS Tools,* SG24-6533
- *IMS Installation and Maintenance Processes,* SG24-6574
- *IMS Version 8 Implementation Guide - A Technical Introduction of the New Features,* SG24-6594
- *IMS DataPropagator™ Implementation Guide,* SG24-6838
- *Using IMS Data Management Tools for Fast Path Databases*, SG24-6866

Other publications

These publications are also relevant as further information sources:

- *IMS Version 8:Common Service Layer Guide and Reference,* SC27-1293
- *IMS Version 8: Base Primitive Environment Guide and Reference,* SC27-1290
- *IMS Version 8: Command Reference,* SC27-1291
- *IMS Version 8: Release Planning Guide,* GC27-1305
- *IMS Version 8: Utilities Reference: Database and Transaction Manager,* SC27-1308
- *Parallel Sysplex Overview: Introducing Data Sharing and Parallelism in a Sysplex,* SA22-7661
- *z/OS MVS Programming: Sysplex Services Guide,* SA22-7617
- *z/OS MVS Programming: Sysplex Services Reference,* SA22-7618
- *z/OS MVS Setting Up a Sysplex,* SA22-7625

© Copyright IBM Corp. 2003. All rights reserved.

Online resources

These Web sites and URLs are also relevant as further information sources:

- IMS Web site

  http://www.ibm.com/ims

- CF sizer tool

  http://www.ibm.com/servers/eserver/zseries/cfsizer

- IBM publications

  http://ehone.ibm.com/public/applications/publications/cgibin/pbi.cgi

How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

ibm.com/redbooks

Archived

Index

© Copyright IBM Corp. 2003. All rights reserved.

C

D

Archived

Archived

Archived

S

Archived

T

U

V

W

X

Archived

IMS in the Parallel Sysplex, Volume I: Reviewing the IMSplex Technology

(0.5” spine)
0.475”<->0.873”
250 <-> 459 pages

Redbooks

Archived

Archived

IMS in the Parallel Sysplex

Volume I: Reviewing the IMSplex Technology

Redbooks

Become familiar with the Parallel Sysplex functions for IMS

Review the data sharing and shared queues features

Explore the benefits of the new CSL architecture

This IBM Redbook is the first volume of a series of redbooks called IMS in the Parallel Sysplex. These redbooks describe how IMS exploits the Parallel Sysplex functions and how to plan for, implement, and operate IMS systems working together in a Parallel Sysplex.

Volume 1 provides an overview of the Parallel Sysplex and the services offered to authorized programs such as IMS. It then continues with the description of the functionality that IMS provides in a Parallel Sysplex environment. Each function supported by IMS is presented in one or more chapters. The topics include:

- Introduction to the Parallel Sysplex
- Block level data sharing
- Connectivity and workload balancing
- Shared queues
- Common Service Layer

The other volumes in this series are:

IMS in the Parallel Sysplex, Volume II: Planning the IMSplex, SG24-6928

IMS in the Parallel Sysplex, Volume III: IMSplex Implementation and Operations, SG24-6929

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks

SG24-6908-00 ISBN 0738430005