#SHAREorg

SHARE
Technology · Connections · Results

Session 11861: MQ on z/OS - Vivisection

Lyn Elkins – elkinsc@us.ibm.com

IBM Advanced Technical Skills

SHARE in Anaheim 2012

#SHAREorg

SHARE
Technology · Connections · Results

Agenda

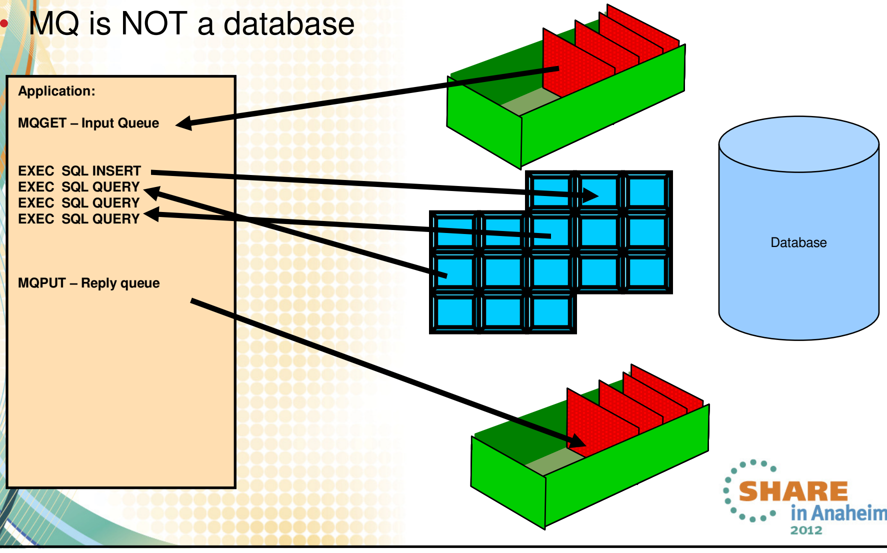
- One of these things is not like the other
- How are messages stored?
  - Private Queues
  - Shared Queues
- First line managers - the components of a z/OS queue manager
- What happens on a API call?
- Summary

SHARE in Anaheim 2012

#SHAREorg

One of these things is not like the other

- MQ is NOT a database



SHARE in Anaheim 2012

#SHAREorg

Not a database…we are NOT a database

N O T E S

Queue Managers and Databases provide different capabilities and have different application access profiles.

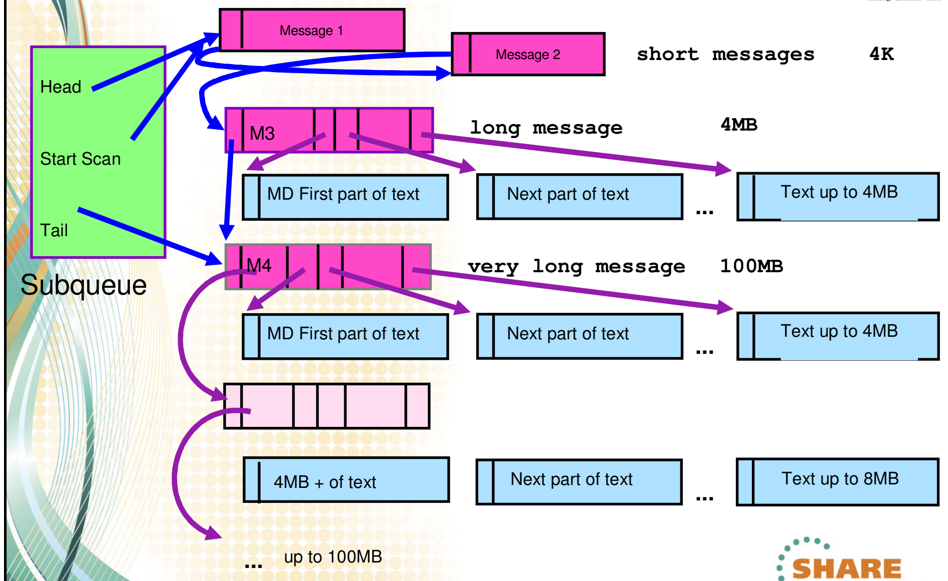
- In a Queue manager *most* operations are disruptive. This is because MQGET and MQPUT operations change the contents of a queue. This is in contrast to a database where most operations are non-disruptive. This is because a query of a database table does not change the contents of a table and is generally issued much more frequently than inserts. The READ WRITE ratio for these technologies is completely inverted. In addition messages may or may not have a persistence associated with them. Table entries tend to be long lived objects.

- Messages are an example of data flow, database tables (and, at times, files) are data endpoints.
  - In many business transactions we expect both information to flow between users where it is retrieved and stored. One of the most important aspects of WebSphere MQ is the way it "extends the transaction". This means that a local transaction can be made available outside its normal environment by the use of a WebSphere MQ message to supply input and deliver the response.
- MQ is designed for ease of message movement and concurrency.
  - As many applications are putting and getting messages at the same time, and these operations are all disruptive, the Queue managers needs to provide a very high degree of update concurrency. Since a queue is only a *temporary* store for data (remember, those messages represent important information flow!), there should be opportunities to optimise the flow of data through a queue manager in the knowledge that data isn't going to reside on a queue for a very long time.
- In terms of delivering a valuable messaging function to applications there are several requirements.
  - **A Simple and portable API (MQI) ensures that the simple concepts of messaging and queuing are available and implementable throughout the enterprise.**
  - **An independent queuing subsystem** ensures that all environments (CICS, IMS, Batch, TSO) have democratic access to messaging and queuing.
  - **Participation in the application environment's recovery ensures** that robust messaging is provided. This includes transactional support and unexpected failure of application, environment or hardware.
  - **High concurrency queue access and high throughput rate are vital to enable messaging to be used as part of the organisational infrastructure by all users.**
  - **Interplatform communications** extend access to WebSphere MQ enabled transactions throughout the enterprise and between enterprises.

SHARE in Anaheim 2012

#SHAREorg

Private Queue Message Storage



SHARE in Anaheim 2012

#SHAREorg

Private Queue Message Storage

N O T E S

Queues store messages on pages.

- Each page is 4kB in size.
- Each of the 20 subqueues has a different anchor point identifying first and last message pages.
- A start position is held to enable more efficient MQGET processing.
- According to their size, messages are defined as being short, long or very long.
- Different sized messages have different page representations.

Short messages are contained completely within a page.

- When a getter is traversing a queue and locates an eligible short message, it can directly copy all the data from the page. The message is then marked as deleted.

Long and Very Long Messages have a hierarchical structure.

- A message greater than 4K has a reference in the traversal chain. These references point to message text, or lower level references, and so on. Reference structure means no architected maximum to very long message size.

Pages are deallocated as all messages are removed from a page.

- Messages that reside completely within a page, or the high level reference to long or very long messages do not cause page synchronous page deallocation at MQGET time. This would cause too much contention.
- Asynchronous deallocation is performed by the Scavenger process, who acquires exclusive page locks for pages with all messages deleted.
- Text pages do not suffer contention, and are deallocated by synchronously during MQGET processing.

Large messages *may* have performance impacts.

- Merely enabling message large message support does not impact performance.
- The structure of very large messages does not degrade the traversal characteristics of the queue. Notice how long and very long messages are no more difficult to traverse at the high level.
- The getter of a 100MB message has to traverse an increased number of pages.
- Putting a very large message can cause a large amount of logging.

Objects are stored in a similar way to messages.

- All WebSphere MQ objects are held on pageset 0. They are anchored from page 0.
- Page 0 contains an array of object types, and objects are chained in a similar way to messages.
- Short and Long Objects (Namelist is an example of the latter) have similar structure to short and long messages.

SHARE in Anaheim 2012

#SHAREorg

Shared Queue Message Storage

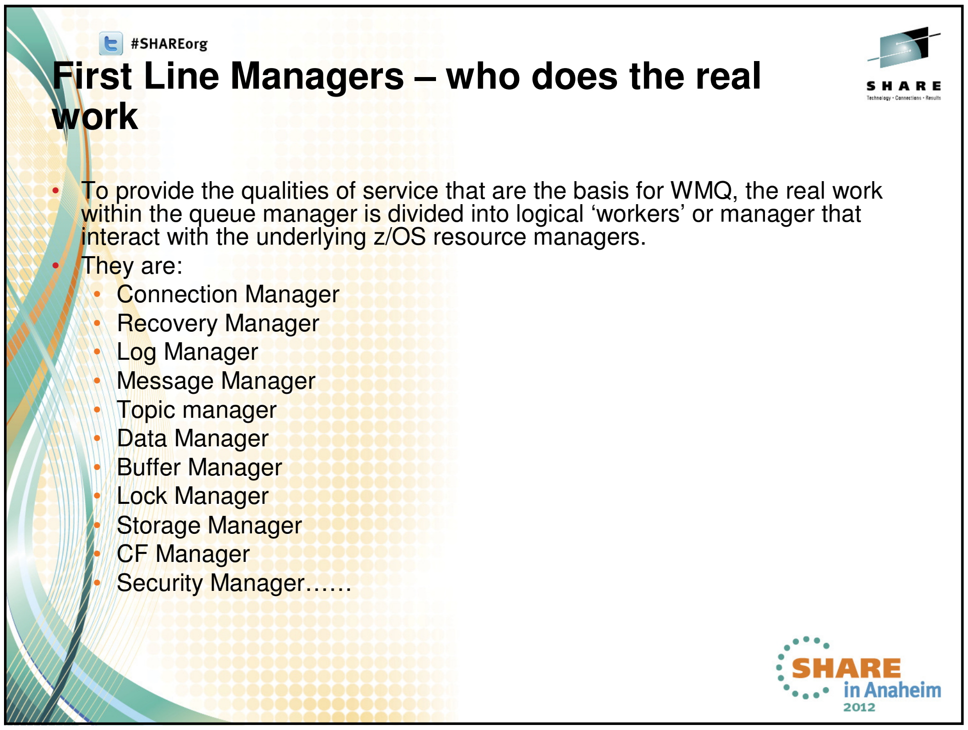
- To be covered in detail in session 11514 on Thursday by Paul Dennis

SHARE in Anaheim 2012

#SHAREorg

Alignment

SHARE in Anaheim 2012

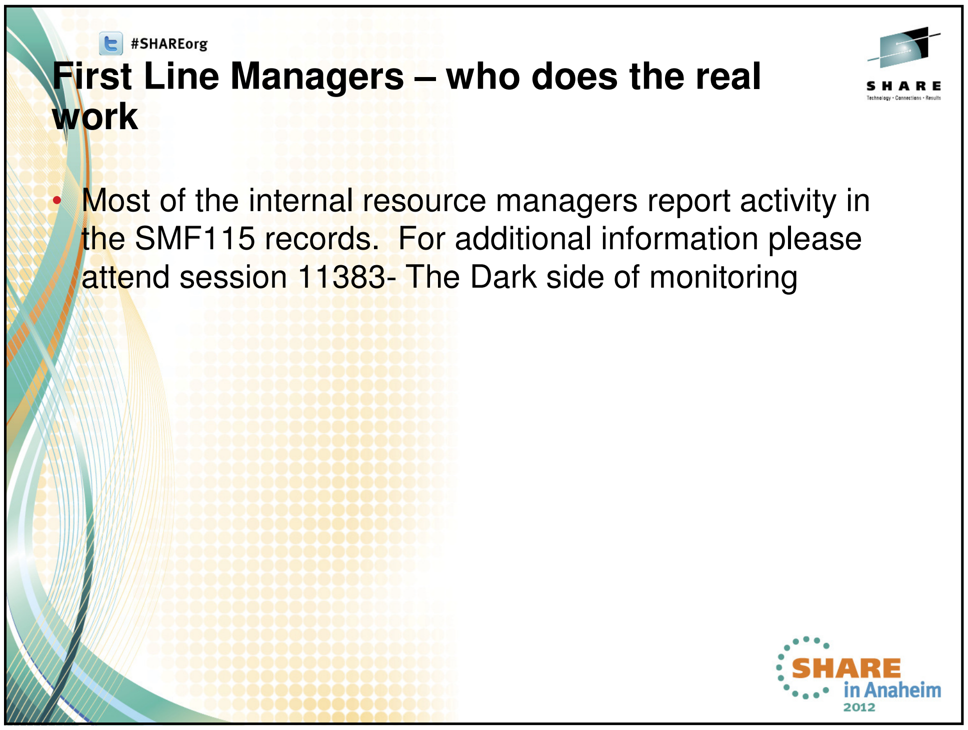


#SHAREorg

First Line Managers – who does the real work

- To provide the qualities of service that are the basis for WMQ, the real work within the queue manager is divided into logical 'workers' or manager that interact with the underlying z/OS resource managers.
- They are:
  - Connection Manager
  - Recovery Manager
  - Log Manager
  - Message Manager
  - Topic manager
  - Data Manager
  - Buffer Manager
  - Lock Manager
  - Storage Manager
  - CF Manager
  - Security Manager……

SHARE in Anaheim 2012



#SHAREorg

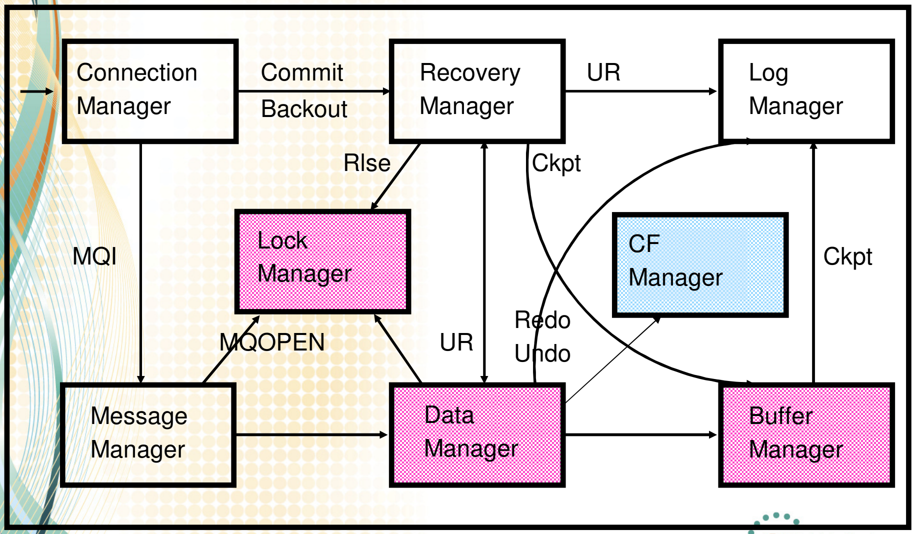
First Line Managers – who does the real work

- Most of the internal resource managers report activity in the SMF115 records. For additional information please attend session 11383- The Dark side of monitoring

SHARE in Anaheim 2012

#SHAREorg

Building Blocks - Resource Managers



SHARE in Anaheim 2012

#SHAREorg

Building Blocks - Resource Managers

N O T E S

The Queue manager is built from many resource managers (RM).

- A resource manager is a code package that encapsulates operations relating to a logically consistent set of operations on a given resource (e.g. the log). If a party wishes to modify or query a resource, it must access it through the appropriate resource manager.
- Resource managers interact with each other to provide the MQI verb set.

The Queue Manager operation can be by understood by decomposing it into the interactions between these resource managers. The most important are:

- Connection Manager
  - **Process the application thread as it enters the Queue manager.**
- Message Manager
  - **Handle operations relating to messages and objects and triggering.**
- Data Manager
  - **Process objects and the physical storage of messages on disk pagesets, coupling facility or DB/2, begin transactions.**
- Buffer Manager
  - **Handles the physical I/O to pagesets and ensures efficient buffering of data for data manager. Handle checkpoints.**
- Recovery Manager
  - **Handle transactional operations (commit, backout). Restart.**
- Log Manager
  - **Provide interfaces to read from and write to the log efficiently. Offloading of active logs, and the management of the archive logs.**
- Lock Manager
  - **Provide locks to enable transactional isolation.**
- Coupling Facility Manager
  - **Provide Queue manager relevant interfaces to the coupling facility. Equivalent function to combination of RMs.**

Now we discuss these Queue manager resource managers in detail.

- We will also discuss the most important data structures, namely how data objects and messages are stored on disk pagesets and Coupling Facility structures.

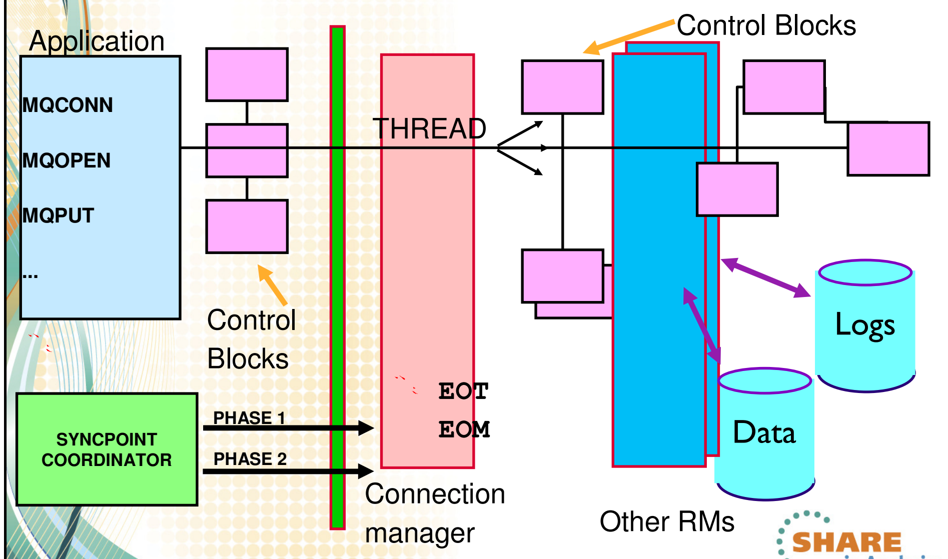
Distributed Queuing is not part of the Queue Manager.

- Moving messages between Queue managers is the responsibility of a separate address space called the channel initiator ("Mover").

SHARE in Anaheim 2012

#SHAREorg

Handling Applications - Connection Manager



SHARE in Anaheim 2012

#SHAREorg

Handling Applications - Connection Manager

N O T E S

Connection Manager handles all MQI requests.

- Connection manager can be considered the "front door" to the queue manager.
- There is sufficient complexity in task management and recovery, and application environments, to justify a separate resource manager just for these functions.
- Connection manager understands the adapter tasking scheme with reference to thread management.

However, MQCONN is not necessarily passed to Connection manager.

- In environments with relatively simple tasking schemes, an application issuing an MQCONN verb causes its adapter to identify with Connection manager.
- In more complex environments where the adapters are separate units of execution (TCBs), the adapters identify with Connection manager at application environment initialisation. An application performing an MQCONN in these environments does not cause interaction with Connection manager. This is why an application doesn't need to issue MQCONN in these environments, most notably CICS.

MQCMIT and MQBACK requests are passed to Recovery Manager.

- When a transaction is using the Queue manager to request WebSphere MQ resource co-ordination, these requests are passed to Recovery manager.
- Such recovery requests are treated differently to the rest of the MQI since they do not relate explicitly to WebSphere MQ resources, but rather to the Unit of Work (UOW) state.
- In two phase commit scenarios (e.g. co-ordinating message input/output with database INSERTs in CICS, IMS, RRS), Connection Manager is not called from the application, but from the Syncpoint co-ordinator.

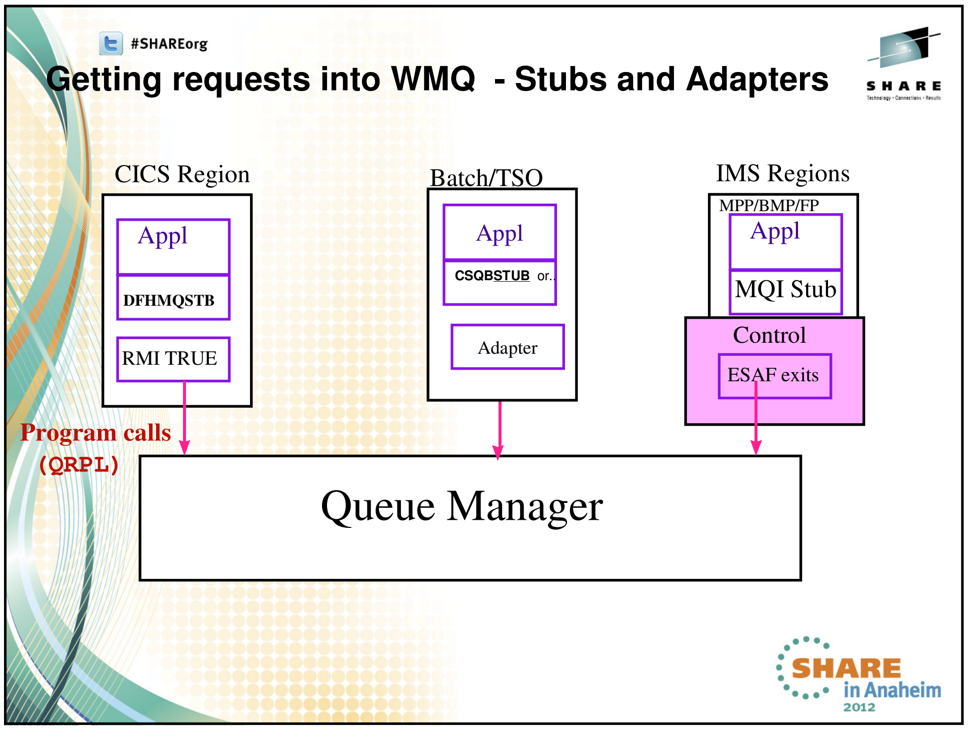
All other MQI verbs are passed to Message Manager.

- It is Message manager who processes all requests relating to messages and queues.

Connection manager handles task termination.

- Normally and abnormally ending applications have transactional implications. The former have their work committed; the converse is true for the latter.
- As the Queue manager uses the Subsystem Interface (SSI) it is notified upon task termination, and can thus implement the appropriate transactional semantics. Note that this can be extremely complex for environments in which the unit of execution moves (e.g. RRS with DB2 stored procedures) and is usually discussed under the subject "Context Management". This is beyond the scope of this presentation.

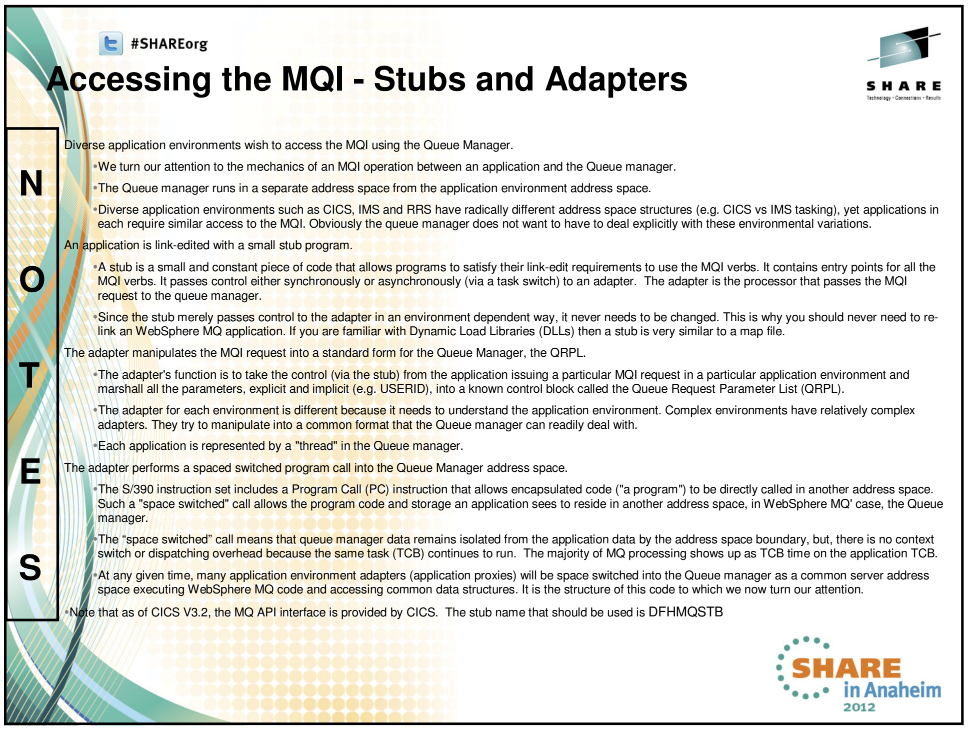
SHARE in Anaheim 2012



#SHAREorg

Getting requests into WMQ - Stubs and Adapters

SHARE in Anaheim 2012



#SHAREorg

Accessing the MQI - Stubs and Adapters

NOTES

Diverse application environments wish to access the MQI using the Queue Manager.

- We turn our attention to the mechanics of an MQI operation between an application and the Queue manager.
- The Queue manager runs in a separate address space from the application environment address space.
- Diverse application environments such as CICS, IMS and RRS have radically different address space structures (e.g. CICS vs IMS tasking), yet applications in each require similar access to the MQI. Obviously the queue manager does not want to have to deal explicitly with these environmental variations.

An application is link-edited with a small stub program.

- A stub is a small and constant piece of code that allows programs to satisfy their link-edit requirements to use the MQI verbs. It contains entry points for all the MQI verbs. It passes control either synchronously or asynchronously (via a task switch) to an adapter. The adapter is the processor that passes the MQI request to the queue manager.
- Since the stub merely passes control to the adapter in an environment dependent way, it never needs to be changed. This is why you should never need to re-link an WebSphere MQ application. If you are familiar with Dynamic Load Libraries (DLLs) then a stub is very similar to a map file.

The adapter manipulates the MQI request into a standard form for the Queue Manager, the QRPL.

- The adapter's function is to take the control (via the stub) from the application issuing a particular MQI request in a particular application environment and marshall all the parameters, explicit and implicit (e.g. USERID), into a known control block called the Queue Request Parameter List (QRPL).
- The adapter for each environment is different because it needs to understand the application environment. Complex environments have relatively complex adapters. They try to manipulate into a common format that the Queue manager can readily deal with.
- Each application is represented by a "thread" in the Queue manager.

The adapter performs a spaced switched program call into the Queue Manager address space.

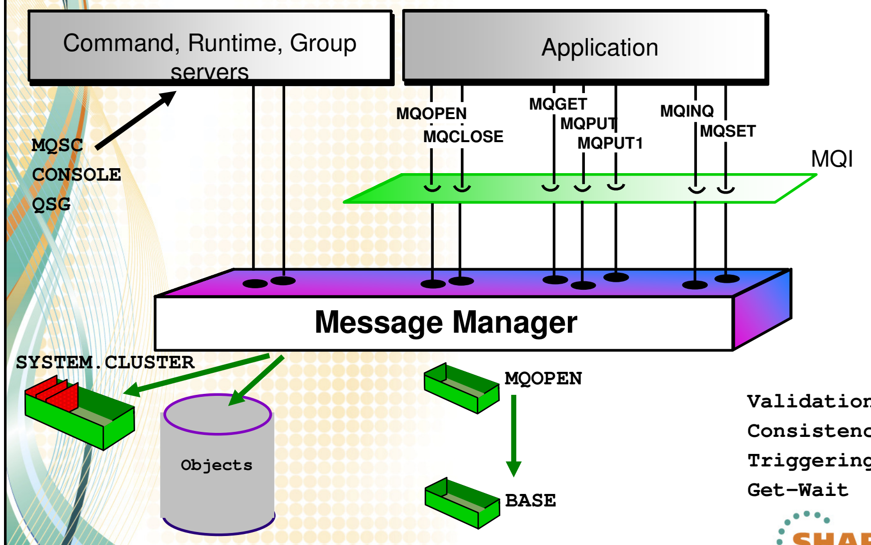
- The S/390 instruction set includes a Program Call (PC) instruction that allows encapsulated code ("a program") to be directly called in another address space. Such a "space switched" call allows the program code and storage an application sees to reside in another address space, in WebSphere MQ' case, the Queue manager.
- The "space switched" call means that queue manager data remains isolated from the application data by the address space boundary, but, there is no context switch or dispatching overhead because the same task (TCB) continues to run. The majority of MQ processing shows up as TCB time on the application TCB.
- At any given time, many application environment adapters (application proxies) will be space switched into the Queue manager as a common server address space executing WebSphere MQ code and accessing common data structures. It is the structure of this code to which we now turn our attention.

- Note that as of CICS V3.2, the MQ API interface is provided by CICS. The stub name that should be used is DFHMQSTB

SHARE in Anaheim 2012

#SHAREorg

Controlling the MQI and MQSC - Message Manager



Validation
Consistency
Triggering
Get-Wait

SHARE in Anaheim 2012

#SHAREorg

Controlling the MQI and MQSC - Message Manager

N O T E S

Message manager manages all the WebSphere MQ resource related MQI.
- It is the component that is most oriented towards applications. Other components deal with more abstract concepts (from the application's perspective), such as disk pagesets, buffers and logging.
- It controls the external and internal application environment APIs.

Message manager manages all MQSC requests.
- It processes all MQSC requests related to queues, processes, namelists and storage classes. These originate from the Command, Runtime and Group servers.
- If a command or resource modification relates to clustering, Message manager notifies the Repository manager through using the SYSTEM.CLUSTER.COMMAND.QUEUE.

Validation of MQI requests and MQSC commands
- It checks validity and consistency of MQI requests (e.g. MQOO_ value, can't open for shared and exclusive input).
- Similar validity and consistency checking is performed for MQSC.

Name resolution for Remote, Alias and Cluster queues.
- Open processing resolves the requested queue name to a base queue name.
- At Put time, any appropriate headers are added.

Manages security processing.
- Using userid information extracted from the QRPL, MQMD, and application environment control blocks, calls Security manager (not further discussed) to verify access to opened queue.

Implements shared/exclusive access to queues.
- At MQOPEN it creates locks that allows single or multi user input access to a queue. Uses Lock manager services for these locks. These locks are released when the queue is closed, so called "Allocation duration locks".

Performs processing to enable triggering and Get-Wait processing.
- After a successful MQPUT to a queue, it
  - **checks triggering rules to see if a trigger message is needed.**
  - **checks for any MQGET waiters that could be satisfied.**
- In some shared queue environments, there is Coupling Facility assistance for trigger and Get-Wait processing.

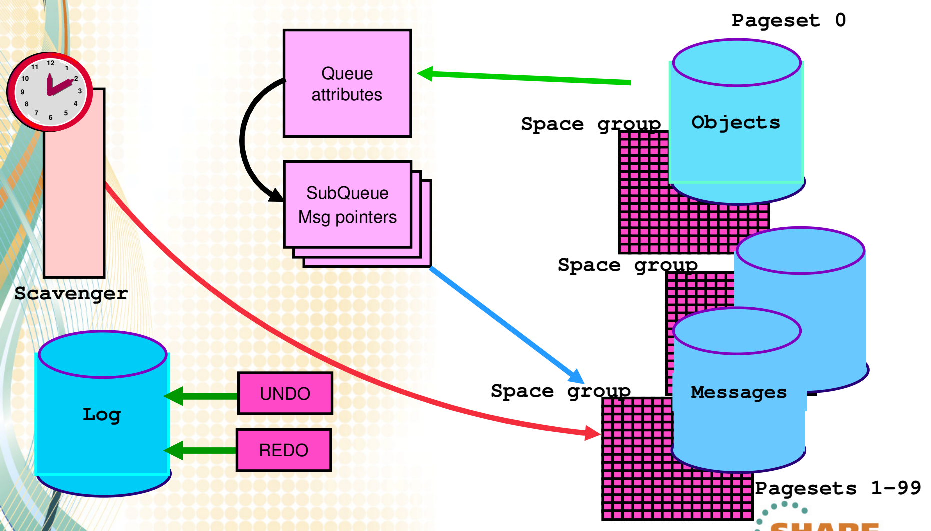
Uses Data manager access method to process messages and objects.

Put to waiting Getter is implemented in the message manager

SHARE in Anaheim 2012

#SHAREorg

Controlling Messages and Objects - Data Manager



SHARE in Anaheim 2012

#SHAREorg

Controlling Messages and Objects - Data Manager

NOTES

Data manager relates WebSphere MQ messages and objects to buffers in pagesets.

- Data manager maps every object (queue, namelist...) and message to a buffer on a page in a pageset.
- Pagesets are defined using the DEFINE PSID(*pageset id*) BUFFPOOL(*buffer pool id*). Uses DSN CSQP00xx.
- There are several types of buffer corresponding to message or object size (see later).
- All WebSphere MQ object definitions are held on pageset 0.
- For messages held in shared message queues, the Coupling Facility is used rather than pagesets (see later).

Data manager manages space in pagesets.

- The usage of pages in a page set is recorded in the "space map" within each pageset.
- The space map is used to allocate new pages for new and updated objects or MQPUT operations.
- MQGET operations result in the deallocation of pages.
- Page deallocation can also be performed asynchronously by the scavenger process.

Data manager begins the transaction and logs all transactional operations.

- A transaction is implicitly started when the first MQPUT or MQGET is performed within syncpoint.
- It writes REDO and UNDO operations to the log that can be used to complete or rollback the transaction in the event of failure.
- Pageset recovery can be performed at restart to bring pagesets to a consistent point. Data manager uses "before" and "after" images (position, length and content) of a pageset change recorded on the log to do this.

Data manager acquires locks on pages and messages and queues.

- It acquires exclusive locks on messages that are not released until commit time, to ensure message isolation. This is necessary for both MQPUT and MQGET, since changes are not visible until commit (unless same UOW).
- It acquires shared locks on pages to ensure that pages are not deallocated when it is using them.
- It acquires shared locks on queues to prevent deletion for in-doubt transactions.

Data Manager uses Buffer manager to optimally access pages.

- Performance can be greatly enhanced by buffering input and output to pagesets. Data manager relies on Buffer manager to read and write virtual storage representations of the pagesets.

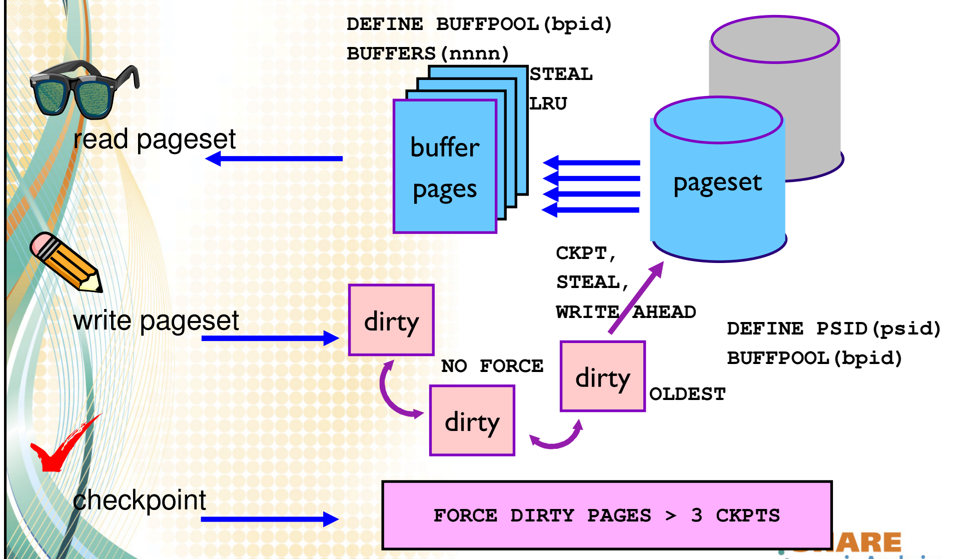
A queue is implemented as 20 subqueues.

- These subqueues correspond to the persistence and priority of messages. This improves performance of messages of a given priority at put time, though typically all messages on a queue have the same priority.
- Having separate persistent and non-presistent messagepages improves restart time, since non-persistent message pages can be marked as deallocated in the space map at restart.

SHARE in Anaheim 2012

#SHAREorg

Buffer Manager – High Performance storage and retrieval



SHARE in Anaheim 2012

#SHAREorg

Private Queue Message Storage – Buffers & Pagesets

NOTES

Messages are initially stored on buffer pages, in virtual storage. Ideally they remain there for their lives.

One of the 'sub-managers' within a queue manager, the buffer manager, performs I/O operations to pagesets.

- The access methods provided for manipulation of logical pagesets result in I/O operations being performed to pagesets.
- Buffer manager caches I/O in virtual storage pages to improve performance.
- Buffers are defined for a pageset using the DEFINE BUFFPOOL MQSC command. Buffer poools are mapped to pagesets using DEFINE PSID.

Buffer manager caches pageset read operations.

- It caches most recently referenced pages. This includes some read-ahead processing for pages. Applications performing unspecified MQGETs benefit most from this.
- When buffer pools become full, Buffer manager may use a "steal" policy to write out least recently used (LRU) pages to disk. These stolen pages are now available for the new data.

Buffer manager caches pageset write operations.

- As all recoverable MQI operations are written to the log, Buffer manager can defer writing pageset changes. This "no-force" policy greatly improves throughput.
- Deferred writes relating to committed transactions will require REDOing at restart after an abnormal termination, since the change never made it to the pageset.

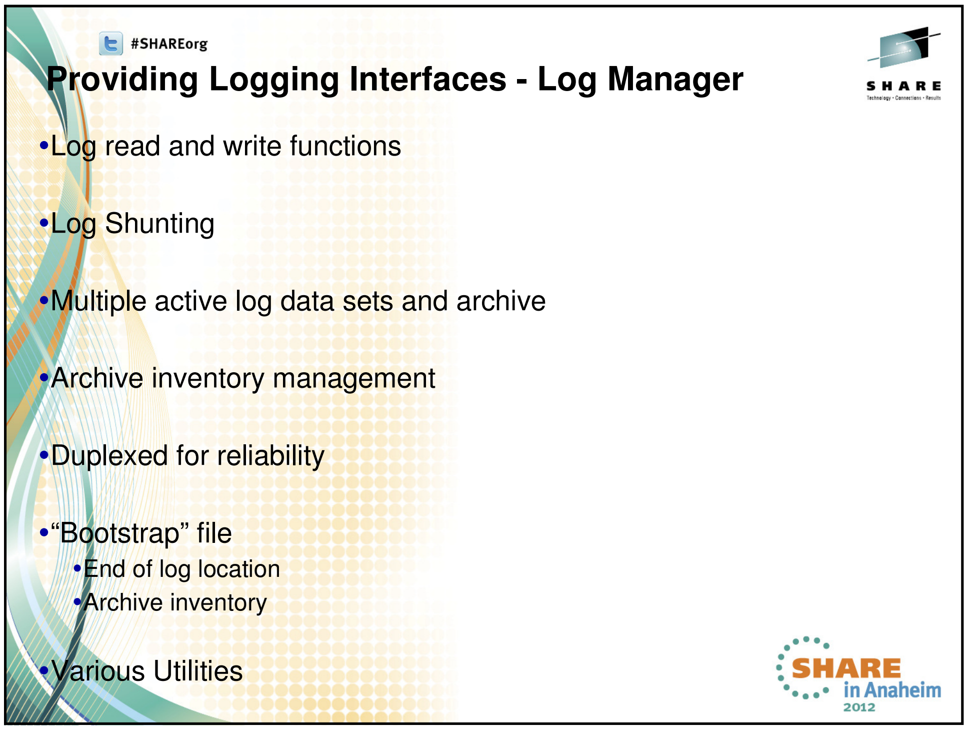
Buffer manager enforces "log write-ahead rule".

- Buffer manager ensures that any pages written out to pagesets have their corresponding log records forced out to the log. This means the log must always be ahead of the oldest deferred page.
- Ensures that any written buffers (including stolen) are consistently recovered with the log.

Recovery Manager broadcasts checkpoint requests to Buffer manager.

- Checkpoint processing is driven by Recovery manager according to the number of log operations. The LOGLOAD parameter defaults to checkpointing every 10K log operations. It s also performed at log switching.
- Whenever a buffer page becomes more than three checkpoints old, it is forced out to disk.
- Long checkpoint intervals increase Queue manager restart time after an abnormal termination, since more of the log needs to be analysed to bring the pagesets up to date.

SHARE in Anaheim 2012

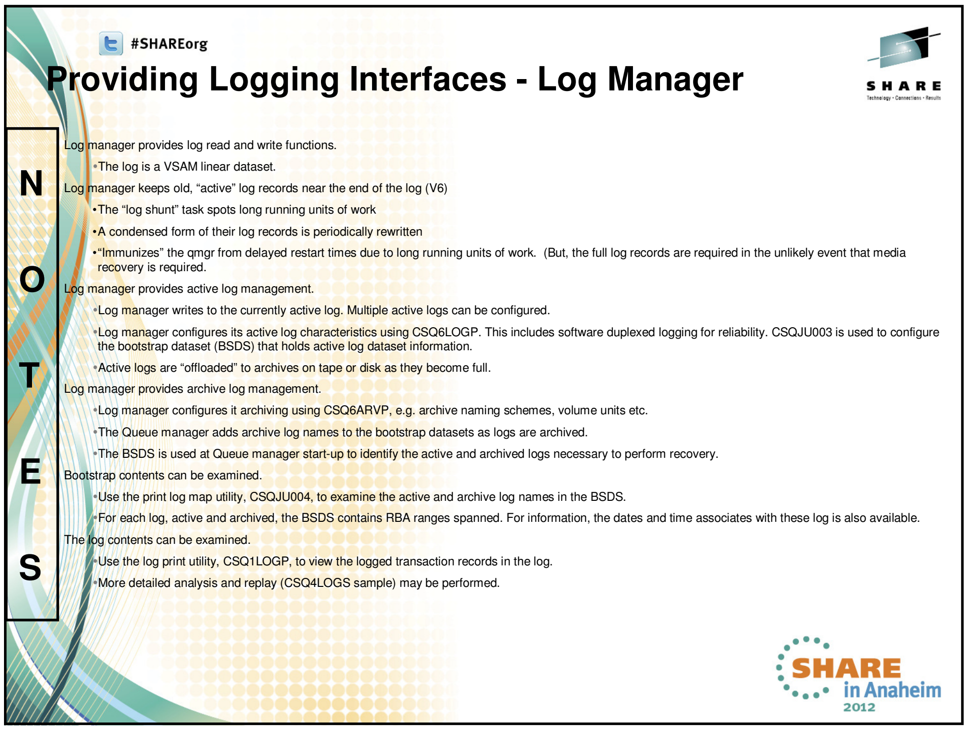


#SHAREorg

Providing Logging Interfaces - Log Manager

- Log read and write functions
- Log Shunting
- Multiple active log data sets and archive
- Archive inventory management
- Duplexed for reliability
- “Bootstrap” file
  - End of log location
  - Archive inventory
- Various Utilities

SHARE in Anaheim 2012



#SHAREorg

Providing Logging Interfaces - Log Manager

N O T E S

Log manager provides log read and write functions.
- The log is a VSAM linear dataset.

Log manager keeps old, “active” log records near the end of the log (V6)
- The “log shunt” task spots long running units of work
- A condensed form of their log records is periodically rewritten
- “Immunizes” the qmgr from delayed restart times due to long running units of work. (But, the full log records are required in the unlikely event that media recovery is required.

Log manager provides active log management.
- Log manager writes to the currently active log. Multiple active logs can be configured.
- Log manager configures its active log characteristics using CSQ6LOGP. This includes software duplexed logging for reliability. CSQJU003 is used to configure the bootstrap dataset (BSDS) that holds active log dataset information.
- Active logs are “offloaded” to archives on tape or disk as they become full.

Log manager provides archive log management.
- Log manager configures it archiving using CSQ6ARVP, e.g. archive naming schemes, volume units etc.
- The Queue manager adds archive log names to the bootstrap datasets as logs are archived.
- The BSDS is used at Queue manager start-up to identify the active and archived logs necessary to perform recovery.

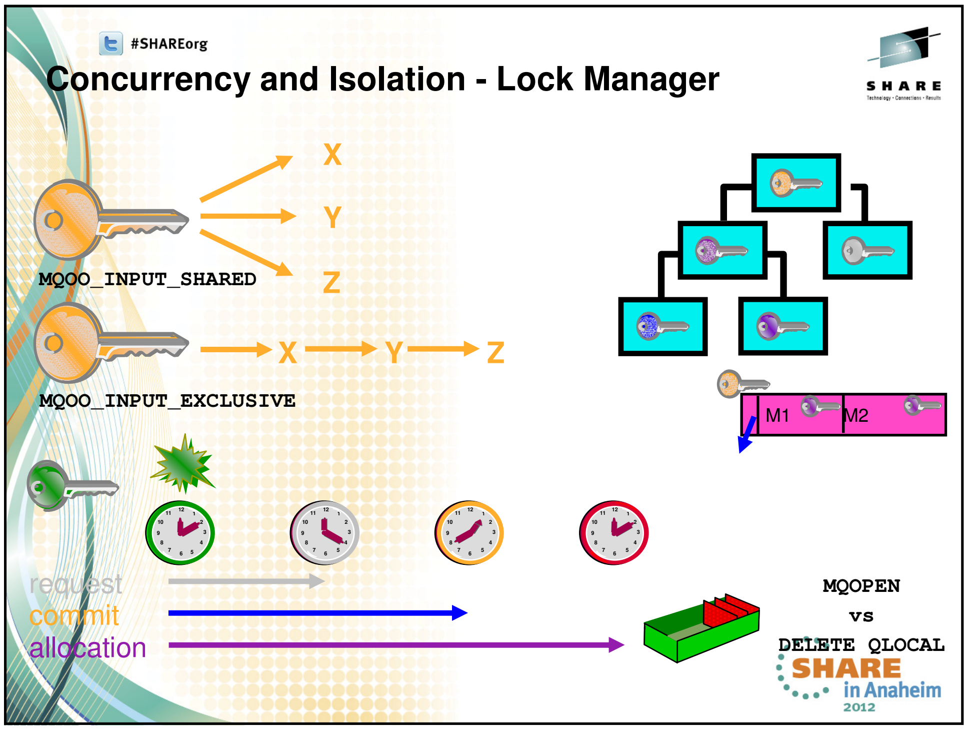
Bootstrap contents can be examined.
- Use the print log map utility, CSQJU004, to examine the active and archive log names in the BSDS.
- For each log, active and archived, the BSDS contains RBA ranges spanned. For information, the dates and time associates with these log is also available.

The log contents can be examined.
- Use the log print utility, CSQ1LOGP, to view the logged transaction records in the log.
- More detailed analysis and replay (CSQ4LOGS sample) may be performed.

SHARE in Anaheim 2012

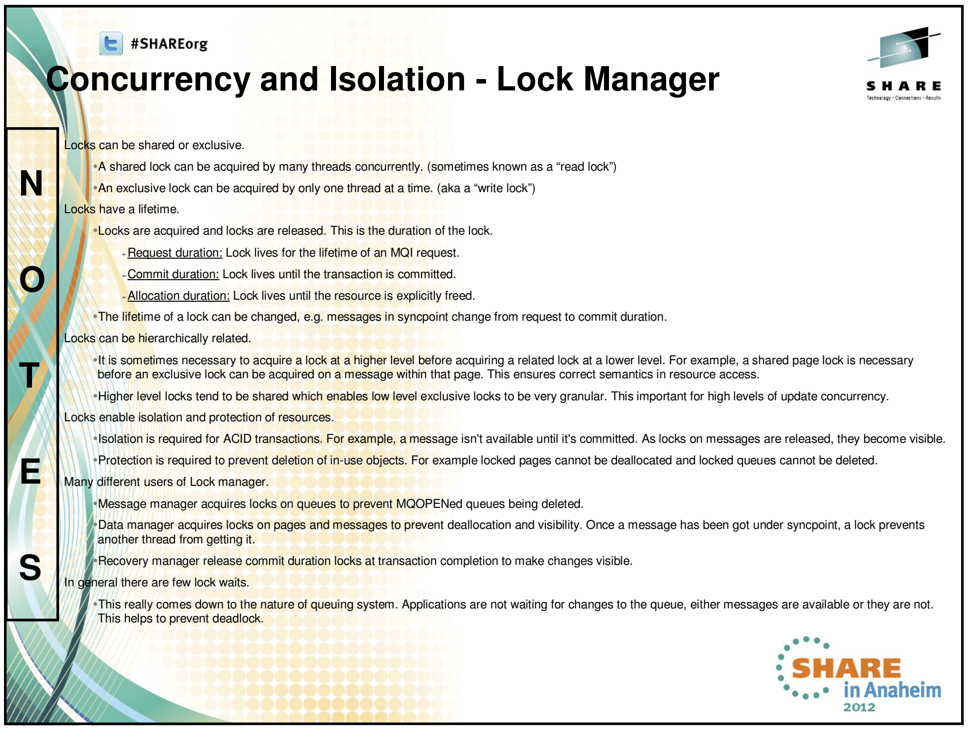
#SHAREorg

Concurrency and Isolation - Lock Manager



#SHAREorg

Concurrency and Isolation - Lock Manager



NOTES

Locks can be shared or exclusive.

- A shared lock can be acquired by many threads concurrently. (sometimes known as a “read lock”)
- An exclusive lock can be acquired by only one thread at a time. (aka a “write lock”)

Locks have a lifetime.

- Locks are acquired and locks are released. This is the duration of the lock.
  - Request duration: Lock lives for the lifetime of an MQI request.
  - Commit duration: Lock lives until the transaction is committed.
  - Allocation duration: Lock lives until the resource is explicitly freed.
- The lifetime of a lock can be changed, e.g. messages in syncpoint change from request to commit duration.

Locks can be hierarchically related.

- It is sometimes necessary to acquire a lock at a higher level before acquiring a related lock at a lower level. For example, a shared page lock is necessary before an exclusive lock can be acquired on a message within that page. This ensures correct semantics in resource access.
- Higher level locks tend to be shared which enables low level exclusive locks to be very granular. This important for high levels of update concurrency.

Locks enable isolation and protection of resources.

- Isolation is required for ACID transactions. For example, a message isn't available until it's committed. As locks on messages are released, they become visible.
- Protection is required to prevent deletion of in-use objects. For example locked pages cannot be deallocated and locked queues cannot be deleted.

Many different users of Lock manager.

- Message manager acquires locks on queues to prevent MQOPENed queues being deleted.
- Data manager acquires locks on pages and messages to prevent deallocation and visibility. Once a message has been got under syncpoint, a lock prevents another thread from getting it.
- Recovery manager release commit duration locks at transaction completion to make changes visible.

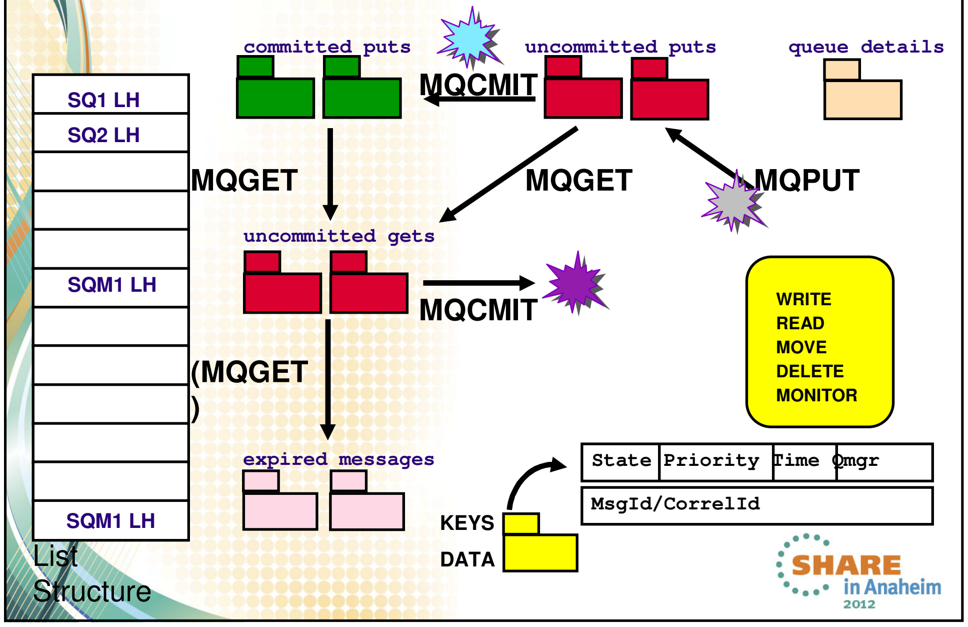
In general there are few lock waits.

- This really comes down to the nature of queuing system. Applications are not waiting for changes to the queue, either messages are available or they are not. This helps to prevent deadlock.

SHARE in Anaheim 2012

#SHAREorg

Shared Message Queue Storage Using CF List Structures



SHARE in Anaheim 2012

#SHAREorg

Shared Message Queue Storage Using CF List Structures

N O T E S

What is a List structure?

- A List structure consists a of a set of list headers (queues). A List header anchors the list and contains information associated with the list. The most interesting are the count of list entries (queue depth) and maximum number of list entries (maximum queue depth).
- A List entry consist of
  - **List entry controls contain information such as an entry identifier, (maps to MsgId) and a primary and secondary key. The list can be monitored according to the contents of the primary and secondary key.**
  - **List elements contain user data which corresponds to the message data (including MQMD). The CF architecture defines a maximum number of elements associated with a list entry, and this limits the message size that may be stored in the CF.**

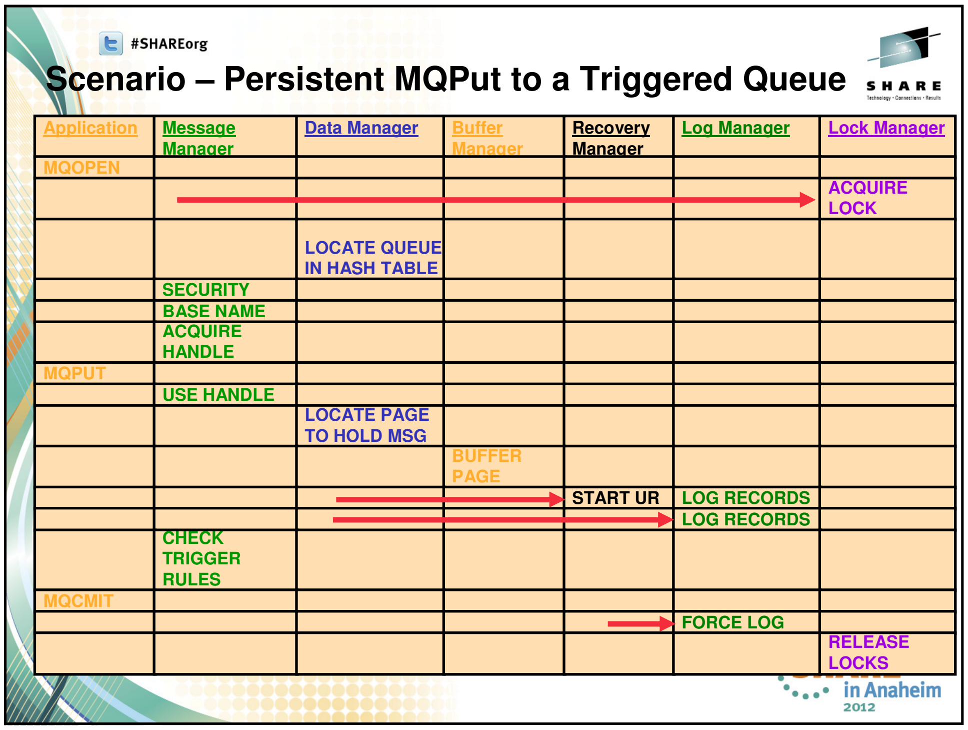
What functions does the List structure provide?

- A rich variety of access methods are provided to manipulate CF list structures.
- Basic functions and their mapping to the MQI include:
  - **READ: Read a list entry; used for MQGET browse a message from a queue.**
  - **WRITE: Create a new list entry; MQPUT a message to a queue.**
  - **DELETE: Delete a list entry; Careful - MQGET is not fully deleted until committed.**
  - **MOVE: Move a list entry to another list; used to get a message under syncpoint, where it is invisible.**
- More complex functions include
  - **READ_LIST: Read list entries with specific key; First pass in MQGET with MsgId/CorrelId, list then searched.**
  - **READ_LCONTROLS: Read list header controls; Used for determining queue depth.**
  - **MONITOR_KEYRANGE: Notify when count of list entry with key range exists; Used for MQGET-wait, triggering.**

How are messages stored on a shared queue?

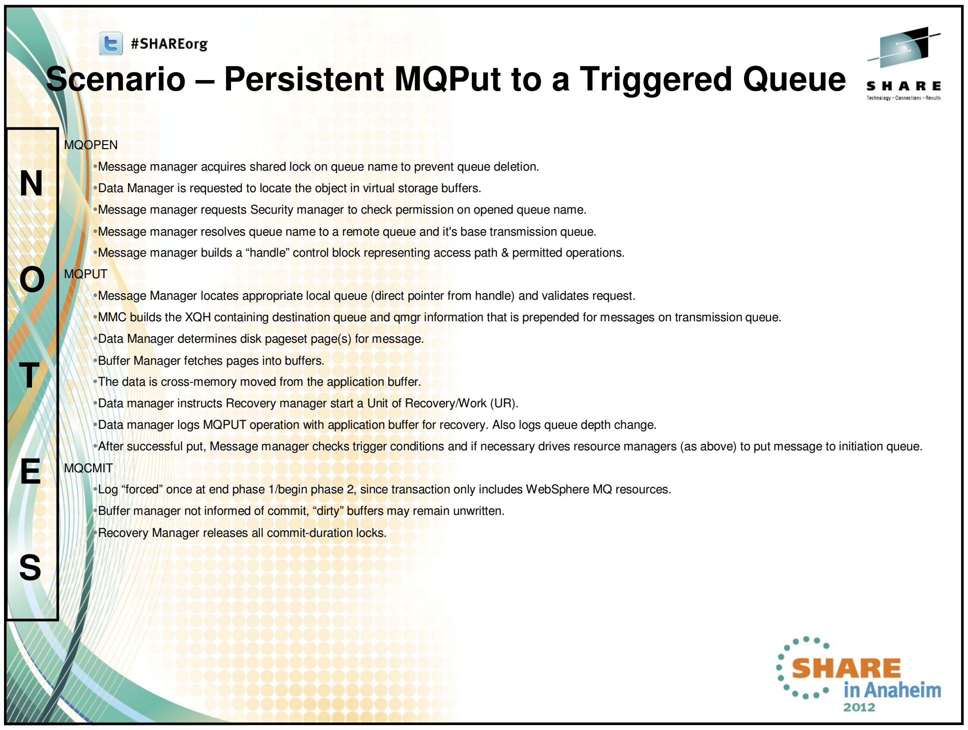
- A message on a queue is represented by a list entry in a list.
- The primary key is used to order the list entries according to WebSphere MQ rules as follows:
  - Highest priority oldest committed message to lowest priority newest committed message.
  - Indoubt uncommitted MQPUTs ordered by putting Qmgr, then priority and put time. Qmgr for peer recovery.
  - Uncommitted MQPUTs ordered by putting Qmgr, then priority and put time. Qmgr for peer recovery.
- Last list entry holds queue definition information and dynamic queue data.
- Uncommitted MQGETs are moved to a separate list. These are deleted at commit time or returned at backout.
- List entry monitoring used to detect committed message arrival. Used for get wait and triggering (first and depth).

SHARE in Anaheim 2012



Scenario – Persistent MQPut to a Triggered Queue

Application	Message Manager	Data Manager	Buffer Manager	Recovery Manager	Log Manager	Lock Manager
MQOPEN						
	→	→	→	→	→	ACQUIRE LOCK
		LOCATE QUEUE IN HASH TABLE				
	SECURITY					
	BASE NAME					
	ACQUIRE HANDLE					
MQPUT						
	USE HANDLE					
		LOCATE PAGE TO HOLD MSG				
			BUFFER PAGE			
		→	→	START UR	LOG RECORDS	
		→	→	→	LOG RECORDS	
	CHECK TRIGGER RULES					
MQCMIT						
				→	FORCE LOG	
						RELEASE LOCKS



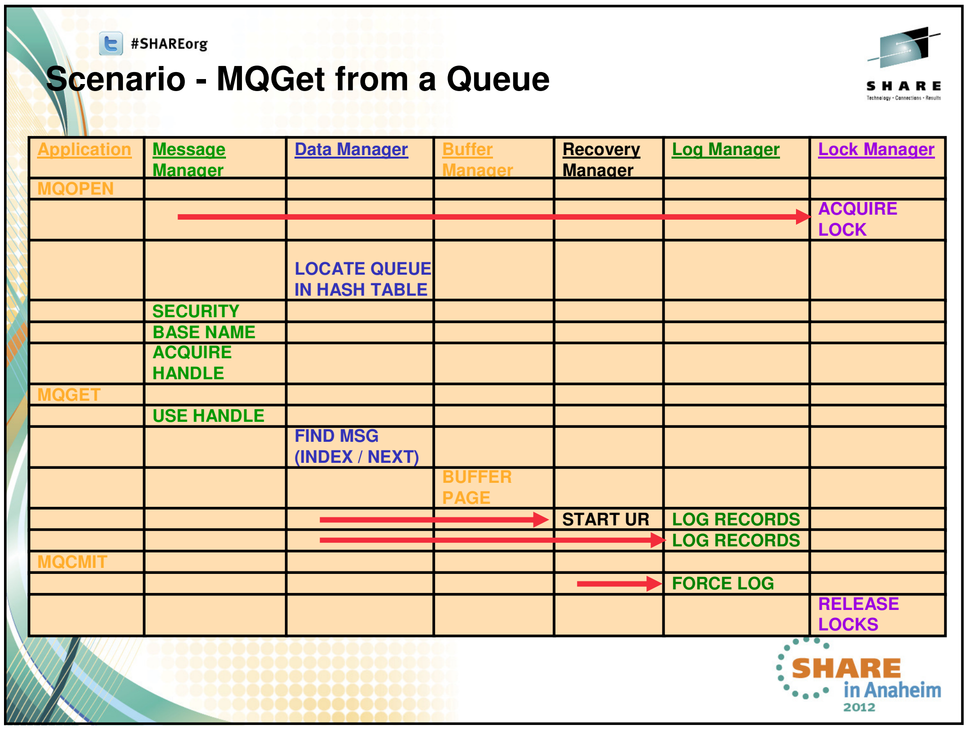
Scenario – Persistent MQPut to a Triggered Queue

N O T E S

MQOPEN
- Message manager acquires shared lock on queue name to prevent queue deletion.
- Data Manager is requested to locate the object in virtual storage buffers.
- Message manager requests Security manager to check permission on opened queue name.
- Message manager resolves queue name to a remote queue and it's base transmission queue.
- Message manager builds a "handle" control block representing access path & permitted operations.

MQPUT
- Message Manager locates appropriate local queue (direct pointer from handle) and validates request.
- MMC builds the XQH containing destination queue and qmgr information that is prepended for messages on transmission queue.
- Data Manager determines disk pageset page(s) for message.
- Buffer Manager fetches pages into buffers.
- The data is cross-memory moved from the application buffer.
- Data manager instructs Recovery manager start a Unit of Recovery/Work (UR).
- Data manager logs MQPUT operation with application buffer for recovery. Also logs queue depth change.
- After successful put, Message manager checks trigger conditions and if necessary drives resource managers (as above) to put message to initiation queue.

MQCMIT
- Log "forced" once at end phase 1/begin phase 2, since transaction only includes WebSphere MQ resources.
- Buffer manager not informed of commit, "dirty" buffers may remain unwritten.
- Recovery Manager releases all commit-duration locks.

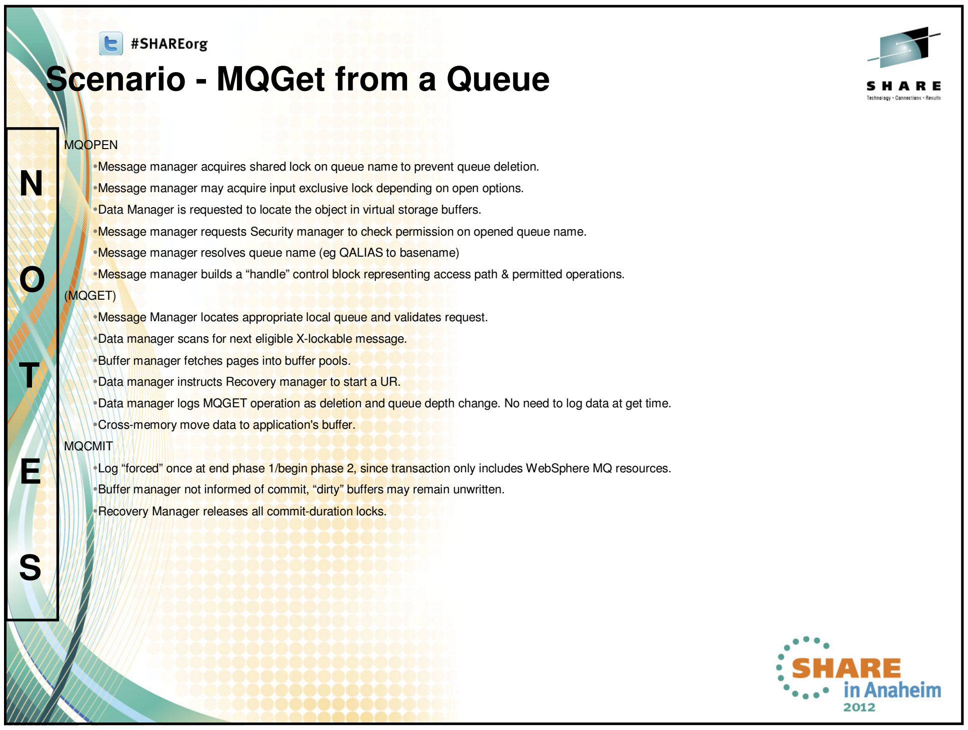


#SHAREorg

Scenario - MQGet from a Queue

Application	Message Manager	Data Manager	Buffer Manager	Recovery Manager	Log Manager	Lock Manager
MQOPEN						
	→	→	→	→	→	ACQUIRE LOCK
		LOCATE QUEUE IN HASH TABLE				
	SECURITY					
	BASE NAME					
	ACQUIRE HANDLE					
MQGET						
	USE HANDLE					
		FIND MSG (INDEX / NEXT)				
			BUFFER PAGE			
		→	→	START UR	LOG RECORDS	
		→	→	→	LOG RECORDS	
MQCMIT						
				→	FORCE LOG	
						RELEASE LOCKS

SHARE in Anaheim 2012



#SHAREorg

Scenario - MQGet from a Queue

NOTES

MQOPEN

- Message manager acquires shared lock on queue name to prevent queue deletion.
- Message manager may acquire input exclusive lock depending on open options.
- Data Manager is requested to locate the object in virtual storage buffers.
- Message manager requests Security manager to check permission on opened queue name.
- Message manager resolves queue name (eg QALIAS to basename)
- Message manager builds a "handle" control block representing access path & permitted operations.

(MQGET)

- Message Manager locates appropriate local queue and validates request.
- Data manager scans for next eligible X-lockable message.
- Buffer manager fetches pages into buffer pools.
- Data manager instructs Recovery manager to start a UR.
- Data manager logs MQGET operation as deletion and queue depth change. No need to log data at get time.
- Cross-memory move data to application's buffer.

MQCMIT

- Log "forced" once at end phase 1/begin phase 2, since transaction only includes WebSphere MQ resources.
- Buffer manager not informed of commit, "dirty" buffers may remain unwritten.
- Recovery Manager releases all commit-duration locks.

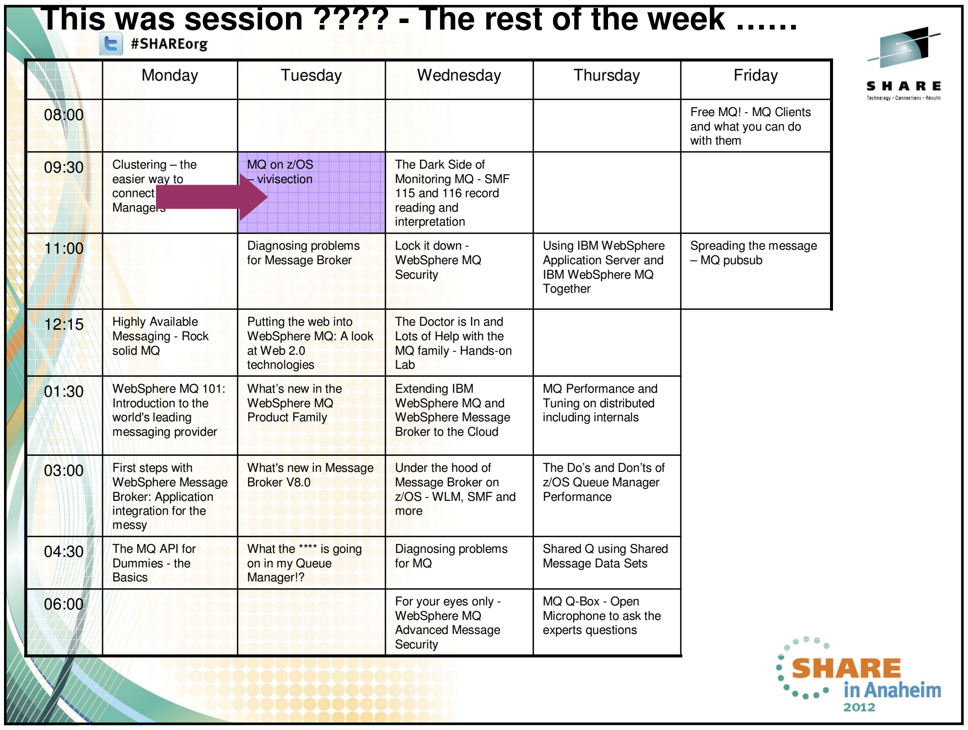
SHARE in Anaheim 2012

#SHAREorg

Summary

- Delivers transactional messaging
  - Enables robust business applications
- Complex, but well organised
  - Adapters, Address spaces, Resource Managers
- Designed for throughput, availability and scalability
  - Logging, Buffering, Locking, Communications

SHARE in Anaheim 2012

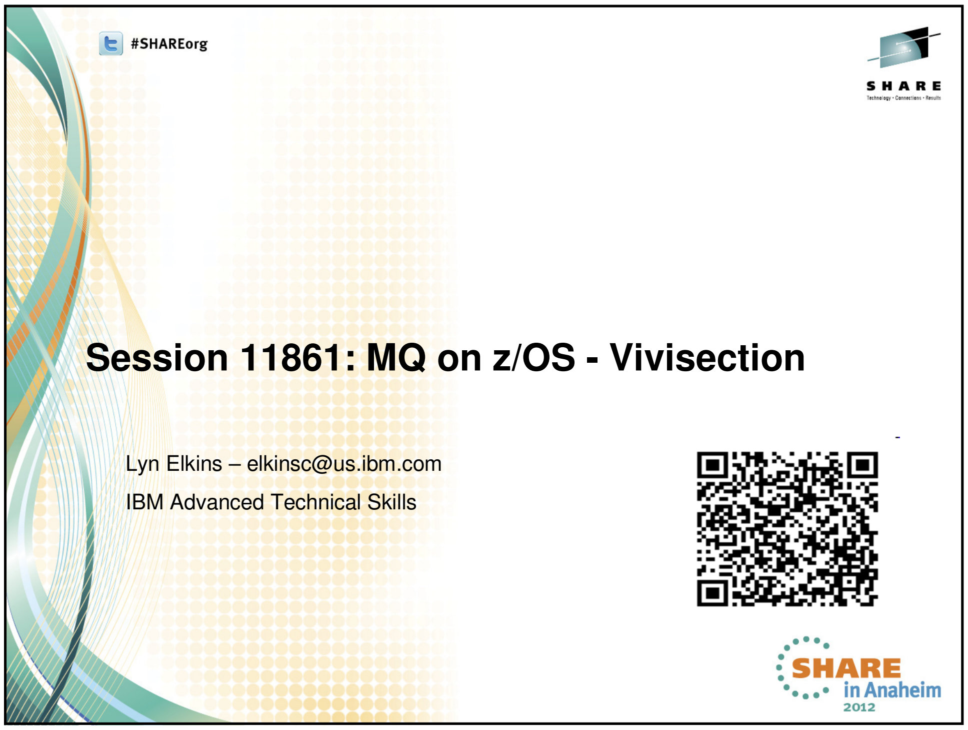


This was session ???? - The rest of the week ……

#SHAREorg

	Monday	Tuesday	Wednesday	Thursday	Friday
08:00					Free MQ! - MQ Clients and what you can do with them
09:30	Clustering – the easier way to connect Managers	MQ on z/OS – vivisection	The Dark Side of Monitoring MQ - SMF 115 and 116 record reading and interpretation		
11:00		Diagnosing problems for Message Broker	Lock it down - WebSphere MQ Security	Using IBM WebSphere Application Server and IBM WebSphere MQ Together	Spreading the message – MQ pubsub
12:15	Highly Available Messaging - Rock solid MQ	Putting the web into WebSphere MQ: A look at Web 2.0 technologies	The Doctor is In and Lots of Help with the MQ family - Hands-on Lab		
01:30	WebSphere MQ 101: Introduction to the world's leading messaging provider	What's new in the WebSphere MQ Product Family	Extending IBM WebSphere MQ and WebSphere Message Broker to the Cloud	MQ Performance and Tuning on distributed including internals	
03:00	First steps with WebSphere Message Broker: Application integration for the messy	What's new in Message Broker V8.0	Under the hood of Message Broker on z/OS - WLM, SMF and more	The Do's and Don'ts of z/OS Queue Manager Performance	
04:30	The MQ API for Dummies - the Basics	What the **** is going on in my Queue Manager!?	Diagnosing problems for MQ	Shared Q using Shared Message Data Sets	
06:00			For your eyes only - WebSphere MQ Advanced Message Security	MQ Q-Box - Open Microphone to ask the experts questions	

SHARE in Anaheim 2012



#SHAREorg

SHARE
Technology · Connections · Results

Session 11861: MQ on z/OS - Vivisection

Lyn Elkins – elkinsc@us.ibm.com
IBM Advanced Technical Skills

SHARE in Anaheim 2012