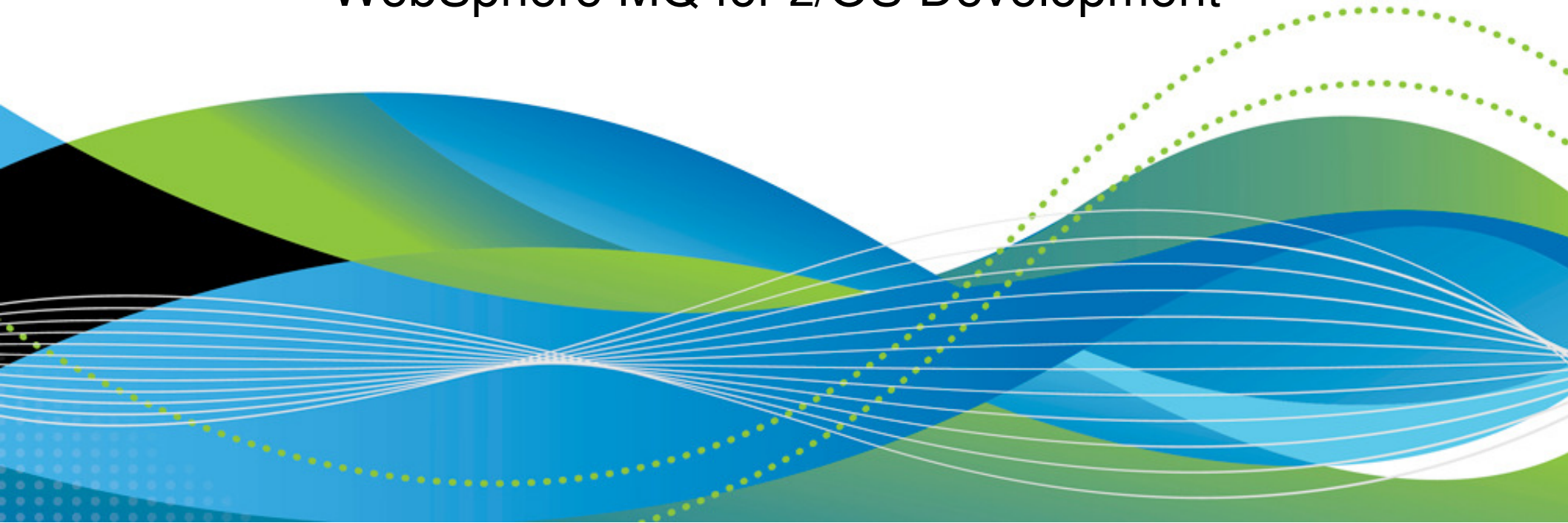
#SHAREorg

SHARE
Technology · Connections · Results

MQ Performance and Tuning

Paul S Dennis
WebSphere MQ for z/OS Development



Agenda

- Application design
- Queue manager configuration
- Mover (chinit) configuration
- Shared queues
- Version 7 updates
- Accounting and statistics
- For more information

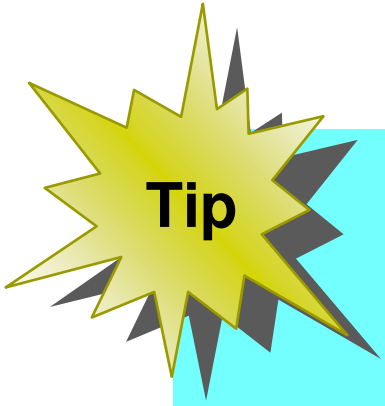
Application Design

Application design

WebSphere MQ is not a database

- ✓ MQ is optimized for temporary storage of in-transit messages
- ✗ MQ is not optimized for long term storage of bulk data

- ✓ MQ is optimized for FIFO retrieval of messages
- ✗ MQ is not optimized for keyed-direct retrieval of data



If you want a database, use DB2

WebSphere MQ is not a database

NOTES

- It is possible to use WebSphere MQ as a database, storing records as "messages" and retrieving specific records by specifying various combinations of queue name, MessageID, CorrelIiD, Groupid, MsgSeq Number and MsgToken.
- Sometimes this is simply poor application design, sometimes it is a conscious decision to avoid the cost of a proper database manager.
- Whatever the reason it is a bad idea:
- WebSphere MQ will not give the operational or performance characteristics of a proper database manager.
- Abusing WebSphere MQ in this way will adversely impact other “legitimate” users of WebSphere MQ.
- If you need a database, use a database manager.

Persistence

Nonpersistent messages

- ✓ Message will be delivered except:
  - ✗ Queue manager fails between PUT and GET (n/a for SQs)
  - ✗ Medium fails between PUT and GET
- ✓ Message will not be delivered twice

Persistent messages

- ✓ Message will be delivered:
  - ✓ Queue manager fails
  - ✓ Medium fails
- ✓ Message will not be delivered twice

Persistence

NOTES

- Chart reminds us that WebSphere MQ provides two classes of service for messages:
- Persistent messages are preserved across software and hardware failures.
- Nonpersistent messages can be lost if is there is a software or hardware failure.
- For both classes of service, WebSphere MQ does not deliver the same message more than once.

Persistence – the choice

Persistent messages are *much* more expensive to process than nonpersistent

So why do we use persistent messages?

Cost of losing the messages 

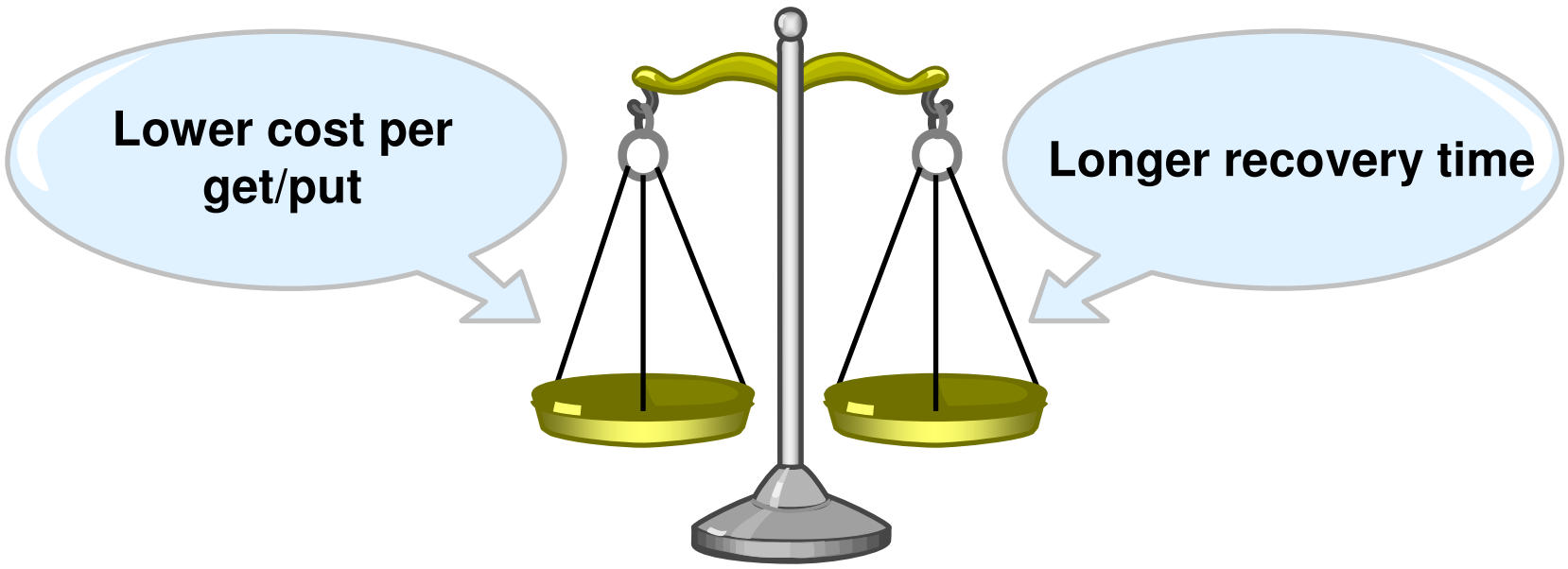
Cost of developing the application 

Persistence – the choice

NOTES

- Many people assume, incorrectly, that you must use persistent messages for "important" information and nonpersistent is for when you don't care
- The real reason for WebSphere MQ persistent message support is to *reduce application complexity*.
- If you use persistent messages your application does not need logic to detect and deal with lost messages.
- If your application (or operational procedures) *can* detect and deal with lost messages, then you do not need to use persistent messages.
- Consider:
  - A bank may have sophisticated cross checking in its applications and in its procedures to ensure no transactions are lost or repeated.
  - An airline might assume (reasonably) that a passenger who does not get a response to a reservation request within a few seconds will check what happened and if necessary repeat or cancel the reservation.
- In both cases the messages are important but the justification for persistence may be weak.

Long units of work



- Small messages (<100KB) -- max 100 per UOW
- Large messages (>100KB) -- max 10 per UOW
- Life of UOW << time to log archive
  - ✘ Do not prompt the user when unit of work in flight!

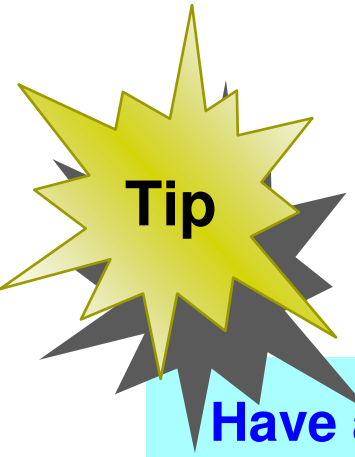
Long units of work - example

NOTES

- Following pseudocode example shows how to split a large UOW which contains many MQPUTs into a number of smaller UOWs, each of which contains 100 MQPUTs.
- The example uses a separate queue (FTP_index_queue) to record how many MQPUTs have been done.

```
Do forever
   Get current line record from FTP_index_queue
   If not found then set current line to first record
   Do i = 1 to 100
     Read file
     MQPUT message
   End
   MQPUT current line record to FTP_index_queue
   Commit
End
```

Tip – long units of work



Have automation check for messages CSQJ160I and CSQR026I

- Use DISPLAY CONN(*) WHERE(QMURID EQ qmurid) to find out where they are coming from

Use MAXUMSGS to limit the number of in-syncpoint operations in a unit of work

For client channels, use DISCINT, HBINT and KAINT.

Tip – long units of work

NOTES

- CSQJ160I is issued for a UOW spanning two or more active log switches.
- CSQR206I is issued when a UOW has been active for 3 checkpoints.

Long queues

Getter is permanently overloaded	You need more compute power -- talk to IBM
Queue is not a queue -- it's a database	You need a database -- talk to IBM

But sometimes:

- **Getter is temporarily down**
- **Getter is temporarily overloaded**
- **Getter is a batch job**

Long queues

FIFO queues (generic gets):

- **Make sure the gets really are generic -- clear MessageID and CorrelliD to nulls (not blanks)**

Indexed queues (eg:get by MessageID or Correllid):

- **Not having an index can be expensive:**

	Indexed	Not indexed
Get 1000th message	600	6300
Get 10 000th message	600	48 000

- **Having an index can be expensive:**

Index rebuild time:	1 millisecond per message 30 minutes per million messages

Don't you just *hate* long queues?

Long Queues and Index Usage

NOTES

- If you are using an indexed queue which contains significant numbers of messages , then it is a good idea to try to have keys (Msgid or CorrelIiD) which vary.
- If all the messages on the index queue have the same Msgid or Correlid then why was the queue indexed? If the keys don't vary then a get with a specific key effectively behaves like a vanilla GET NEXT request.....but it is slower!
- So, if you don't need an index don't specify one.

Heavyweight MQI Calls

- MQCONN is a ‘heavy’ operation
  - Don’t let your application do lots of them
  - Lots of MQCONNs can cause throughput to go from 1000s Msgs/Sec to 10s Msgs/Sec
- MQOPEN is also ‘heavy’ compared to MQPUT/MQGET
  - It’s where we do the security check
  - If you’re only putting one message, consider using MQPUT1

Application design - summary

- ✗ Do not use WebSphere MQ as a database
- ✓ Use nonpersistent messages if you can
- ✗ Avoid long units of work
- ✗ Avoid long queues

Queue manager configuration

Queue manager configuration

Buffer pools and pagesets

Objects

- **Buffer pool 0**
- **Pageset 0**

Queues with short-lived messages

- **Large buffer pool**

Queues with long-lived messages

- **Small buffer pool**

Buffer pools and pagesets

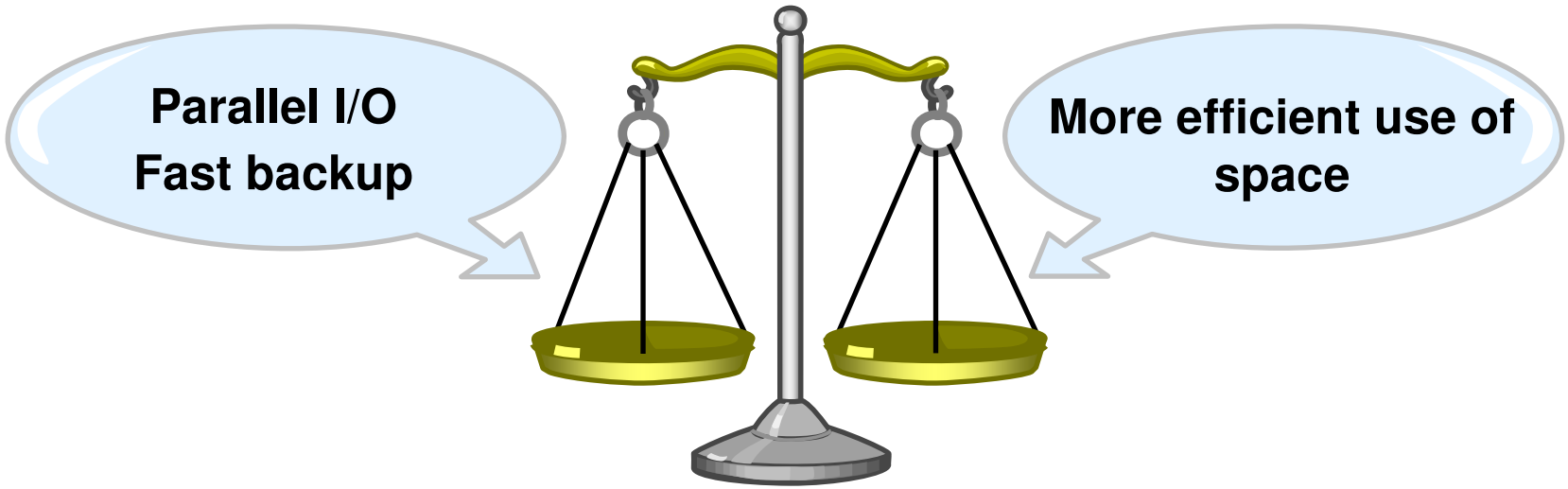
NOTES

- We recommend that you use pageset 0 and buffer pool 0 exclusively for "system" use -- that is, for objects. This reduces any possible impact on the queue manager itself from application activity (the queue-manager needs to update queue objects when messages are put to queues).
- For short-lived messages, we recommend using a large buffer pool to maximize the chance that a message will remain in a buffer for its life (from put to get).
- For long-lived messages we recommend using a different (smaller) buffer pool -- there is not much point in keeping a message in a buffer if we cannot keep it until get time.
- This leaves a number of "spare" buffer pools which you can use for any queues with special characteristics or requirements.

Buffer pools

- Buffer pool 0 only for pageset 0
- Short-lived messages
  - Large buffer pool
  - Deferred pageset writes (QPSTDWT) = 0
  - Pageset reads (QPSTRIO) = 0
- Long-lived messages
  - Small buffer pool so messages move to disk
  - Expect deferred pageset writes > 0
- Make them too big and tune down
  - Buffer exhaustion (QPSTSOS) always 0
- To print SMF stats see MP15 Support pack
- Leave some storage for MVS!

Pagesets

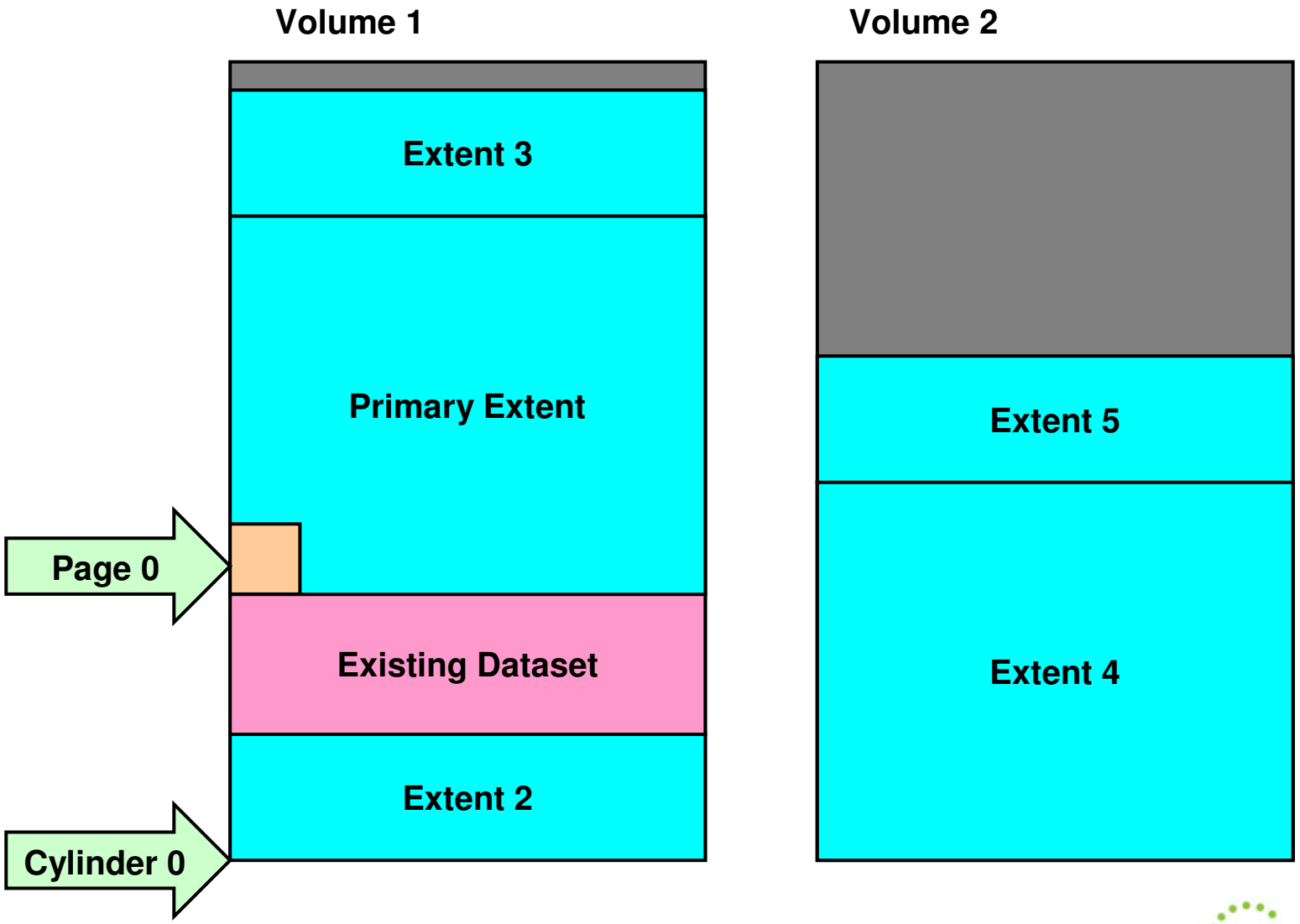


Be sure to allocate secondary extents to allow pageset expansion

Pageset expansion

- Allocate secondary extents for pageset expansion
  - New extent is obtained and formatted when pageset is 90% full
  - Applications waiting for expansion may be suspended
  - If space becomes free in page set then other tasks can use it
  - Can specify EXPAND(SYSTEM) on MQ PSID object if no secondary extents
- Guaranteed Space attribute
  - Allocates primary extent size on each volume specified
  - When page set expansion occurs it fills one volume before using next volume
    - Possible big impact when new volume used
    - Page set expansion must format all of this space

How pageset expansion works



How pageset expansion works

NOTES

- When the data set is allocated and it has the Guaranteed Space attribute then the primary allocation is allocated on each volume (primary extent and extent four).
- When the page set is formatted then only the primary extent is formatted by default.
- As page set expansion occurs, extent 2 and extent 3 are allocated and formatted. When there is no more space on the volume the large extent on the second volume (extent 4) is formatted. When this fills up then extent 5 is allocated and used.
- Do not use a full volume backup of this volume to restore the page set as it may be unusable, and you may not know for weeks that you have corrupted your page set. This is because extent 2 will be backed up before the primary extent, and page 0 of the page set (in the primary extent) *must* be backed up first.

Pageset backup and restore

REPRO of dataset	330 cylinders 3390 per minute
DFDSS DUMP/RESTORE	760 cylinders 3390 per minute
RAMAC snapshot Shark FlashCopy	"Instant"

Example: Recovery time for a 1389 cylinder pageset (1GB)

- **Run 1 million request/reply transactions (90 minutes)**
  - **Batch jobs put to batch server, get reply**
- **Restore page set (DFDSS 2 minutes, REPRO 6 minutes)**
  - **Consider rename**
- **Restart queue manager (log replay 20 minutes)**
  - **50 000 transactions processed per minute**
  - **10 hour's worth of data -> additional 2 hours**

Backup at least daily

Pageset backup and restore

NOTES

- We recommend you backup each pageset *at least* daily.
- Chart shows some sample times for different backup techniques.
- You must also consider the type and amount of storage you use for backups.
- Chart also shows example times for recovery from a lost pageset.
- Note that log replay is a significant part of the recovery time (much the largest part in our example).
- You can reduce the log-replay part of your recovery time by taking more frequent backups.

Active log datasets

- You do not want to lose your active logs:
  - Your queue manager won't work if it can't write to them
  - Your recovery won't work if it can't read from them

✓ Duplex the logs

- You do not want your queue manager to wait for archiving
- You do not want to use archive logs for recovery

✓ Allocate enough space

- You do not want to be slowed by contention:
  - You will have contention if archiving is using the same volume(s) as the active log
  - You will have contention if dual logs are on the same volume

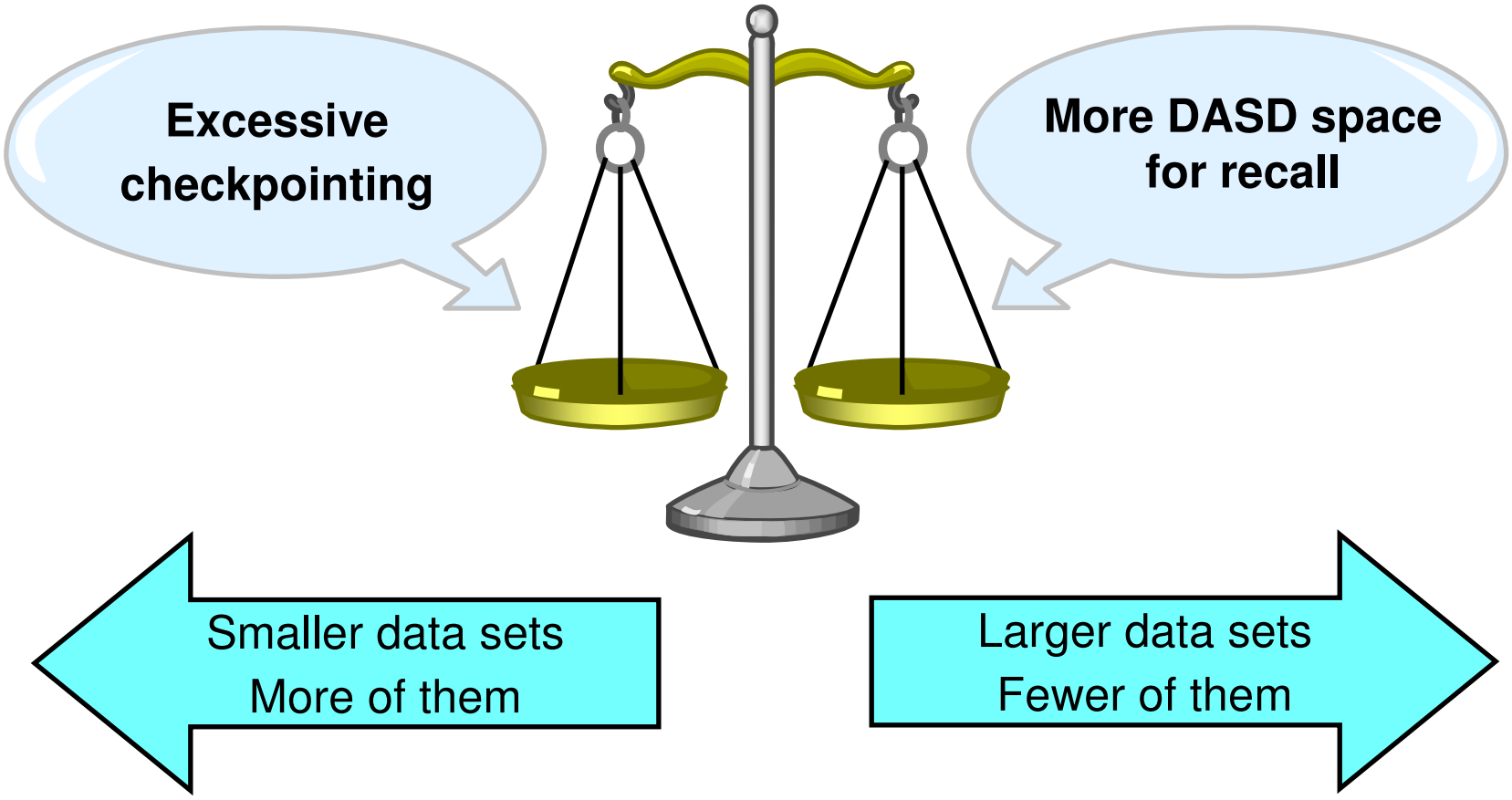
✓ Use separate volumes

Active log datasets

NOTES

- There is very little performance difference between single and dual logging (IO takes place in parallel) unless the total IO load on your DASD subsystem becomes excessive.

Active logs – how many?



- Tune checkpoint frequency with LOGLOAD
- 1000 cylinders of 3390 fit on a 3490 tape

Active logs – how much space

UOW recovery without accessing archive logs:

- Minimum of 4 to allow long-shunting to prevent UOW from going onto archive

Pageset recovery without accessing archive logs:

- Enough space (= time) for plenty of checkpoints
- Enough space (= time) for backups

General recommendation: Allocate at least enough active log space for one day's data

Comfort factors:

- Operational problems can delay archiving
- Unexpected peaks can fill logs quickly

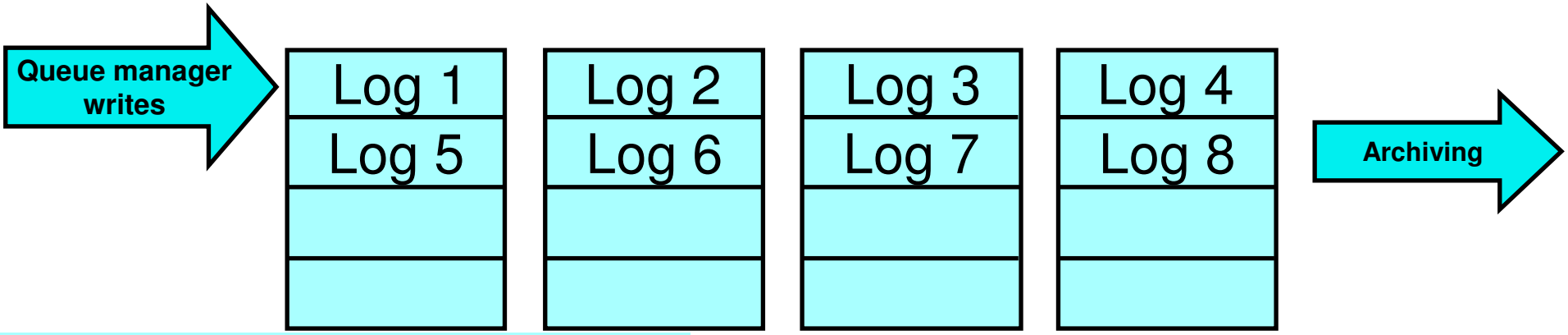
Active logs – how much space

NOTES

- Chart emphasizes that you want to provide enough active log data set space so that you never need to access archived log data sets for recovery -- either "normal" recovery following queue manager termination, or recovery following loss of a pageset. Shunting occurs every three checkpoints for a unit of work, so you need four logs in the ring to be sure that shunting will work, and the active log is not overwriting the records it needs to copy.
- We recommend that you allocate *at least* enough active log space to hold one day's worth of logging.
- For many installations the best arrangement is to use as much DASD space as possible for active log data sets and archive to magnetic tape -- the more active log space you have, the less likely you are to need archived logs for recovery.
- If you sometimes experience long units of work from the chinit, caused by failure of a TCP/IP link, then be sure to specify TCPKEEP = YES which ensures the chinit quickly recognizes a failed TCP/IP link (and backs-out its units of work).

Active logs - contention

- Keep consecutive logs on separate volumes
- Ensure archiving does not use same volume(s) as active log



Some DASD have multiple logical volumes stored on one physical disk:

- Ensure different physical volumes
- Reduce disk contention between logical volumes
- Avoid single point of failure

Shark has one logical volume stored on many small disks -- no physical volume:

- Can use UCB aliasing -- up to 4 paths to one logical volume per MVS image

Active logs - contention

NOTES

- Chart shows a way to organize log data sets across several volumes so that you do net get contention for the same volume when switching from one active log data set to the next, and so that you do not get contention for the same volume between logging by the queue manager and archiving.
- If your DASD maps several logical volumes to the same physical volume then using different logical volumes does not provide adequate protection from DASD contention -- ensure you allocate on different *physical* volumes.
- Shark UCB aliasing allows you to allocate data sets on the same logical volume without incurring contention. This is automatic if you use workload manager goal-mode.

Archive logs

- Ensure you don't lose your archive logs:
  - Ensure retention date long enough (otherwise data sets get deleted automatically!)
  - Keep enough records in BSDS (MAXARCH should be large)
  - Keep two copies (perhaps one on disk, one on tape)
  - Decide when to send data sets off-site (possibly needed for restart after a failure)
- Same size as active logs
- Don't use software compression for tapes

Archive logs

NOTES

- You should protect archived log data sets in much the same way you protect active log data sets (in particular keep two copies) until you are sure you will never need them for recovery (or anything else!).
- We recommend that you avoid using software compression for archived logs because of the extra time required to decompress if you need to use the data for recovery.

CSQZPARM

CSQ6SYSP

- Tune LOGLOAD so not checkpointing more than 10 times per hour (see QJSTLLCP)

CSQ6LOGP

- Increase output buffer space (OUTBUFF) if buffer waits (QJSTWTB) > 0

CSQ6ARVP

- Make archive allocation (PRIQTY) same size as active logs
- Make archive retention period (ARCRETN) large so archives do not get deleted

CSQZPARM

NOTES

- QJSTWTB indicates the number of times the log output buffer was full and caused a log record to wait for I/O to complete. However, QJSTBPAG might indicate that the log output buffer is too large in relation to the demand for real storage.

Use LLA for frequent connects

- DB2 stored procedure issues MQCONN, MQPUT1, MQDISC
- High costs loading MQSeries modules at MQCONN
- Put CSQBCON and CSQBSRV into LLA(Library Lookaside):
  - 17 connects/disconnects per second -> 65 per second
- Put 'STEPLIB' into LLA to reduce BLDL costs:
  - 65 connects/disconnects per second -> 300 per second
- SYS1.PARMLIB(CSVLLA*xx*):

```
LIBRARIES(MQM.Vvrm.SCSQAUTH,
          CUSTOMER.APPS.LOAD,
          CEE.SCEERUN)
FREEZE(MQM.Vvrm.SCSQAUTH,
       CUSTOMER.APPS.LOAD,
       CEE.SCEERUN)
```

Queue manager throughput

Example application:

Request/reply, 1KB messages, local requester

Nonpersistent messages**:**

- **Throughput limited by queue contention**
- **Example application, single server, zSeries 900:**
  - **At 2200 round trips per second, need more queues or servers**

Persistent messages, two phase commit:

- **Throughput limited by log contention**
- **Example application, single server, Shark DASD:**
  - **At 110 round trips per second, need more servers**
- **Example application, 16 servers, Shark DASD:**
  - **At 770 round trips per second, need more queue managers**

Queue manager throughput

NOTES

- In general we do not expect the throughput of the queue manager to limit the throughput of your system -- processing the messages (in your application) is likely to be much more significant than transporting the messages (in the queue manager). But:
- If you have a single server (more precisely, a single thread), you will be processing messages serially and will probably be able to increase throughput by adding more servers (or threads) to allow parallel processing.
- At high message rates, processing the queues can be limited by contention -- for the queue in the case of nonpersistent messages or for the log in the case of persistent messages.
- You can relieve contention for a queue by using more than one queue.
- If you are limited by contention for the log then you must consider using more than one queue manager.

Put to a waiting getter

- Only takes place when there is a getter waiting for a message that is being put
- Only for non-persistent out of syncpoint messages
- The message is passed straight from the putter to the getter without being placed onto the queue first
  - Removes a lot of processing of placing the message onto the queue
- Significantly reduces CPU cost and improves throughput
  - 40% CPU saving for 1K messages
  - Also applies to shared queues if there is a waiting getter on the same queue manager
  - Particularly significant for large shared queue messages

Put to a waiting getter

NOTES

- "Put to a waiting getter" is a technique whereby a message may not actually be put onto a queue if there is an application already waiting to get the message. Certain conditions must be satisfied for this to occur, in particular the message must be non-persistent and the putter and getter must be processing the message outside syncpoint control. If these conditions are met then the message will be transferred from the putters buffer into the getters buffer without actually touching the MQ queue. This removes a lot of processing involved in putting the message on the queue and therefore leads to increased throughput and lower CPU costs.

Mover (chinit) configuration

Mover (chinit) configuration

Chinit recommendations

Maximum 9000 clients or 4500 channel pairs

Dispatchers issue communication requests

We recommend: 20 dispatchers if more than 100 channels

Adapters issue MQ requests

We recommend: 30 adapters

Chinit recommendations

NOTES

- Chart shows approximate upper limits for chinit capacity (clients or channel pairs). If you are approaching these limits you should consider adding more queue managers.
- Performance of the chinit is potentially constrained if you do not have enough dispatchers or adapters. Chart shows some guidelines -- in general it does not hurt to use higher values.
- The first MIN(MAXCHL/CHIDISP),10) channels are associated with the first dispatcher TCBs and so on until all dispatcher TCBs are used. Therefore, if only a small number of channels are in use, make sure MAXCHL is accurate for even distribution across dispatcher TCBs.

Fast nonpersistent messages

Channel attribute:NPMSPEED(FAST)

- **No end-of-batch commit:**
  - Less CPU and logging
  - Still an end-of-batch confirm flow
- **Processed out of syncpoint:**
  - Messages appear sooner at remote end
  - Consider 100 messages in 1 second, batch size = 100:
    - **A fast message is available every 0.01 second**
    - **Non fast messages, 100 available after 1 second**
- **Risk of losing message if channel fails**

Chinit - batching

- **Many channels have an *achieved batch size* of 1**
  - **Specify BATCHSZ = 1 -- saves 5% CPU cost**
  - **Use DISPLAY CHSTATUS(channel) XBATCHSZ to see the size of batches achieved by the channel (need MONCHL to be set)**
- **For persistent messages, if response time is not critical**
  - **Specify BATCHINT > 0 -- increases messages per batch**
  - **Reduces commits and end-of-batch flows**
  - **Too large a value and you may impact response times**

See *Intercommunication Guide*

Chinit - batching

NOTES

- Batching in the chinit can reduce overheads by spreading the cost of end-of-batch flows and commits over a number of messages (a batch).
- The batch size you specify (BATCHSZ) is a maximum -- if the channel is fast compared to the message rate then the chinit will often have only one message to send -- giving an *achieved batch size* of one.
- If you have an achieved batch size of one, consider setting BATCHSZ to 1 -- this reduces overheads (the chinit does not need to look for a second message to include in the batch).
- If you can tolerate the delay, you can reduce overheads by using BATCHINT to force the channel to wait before ending a batch (and so get a higher achieved batch size).
- *WebSphere MQ Intercommunication* provides more detailed information about batching.

Channel message compression

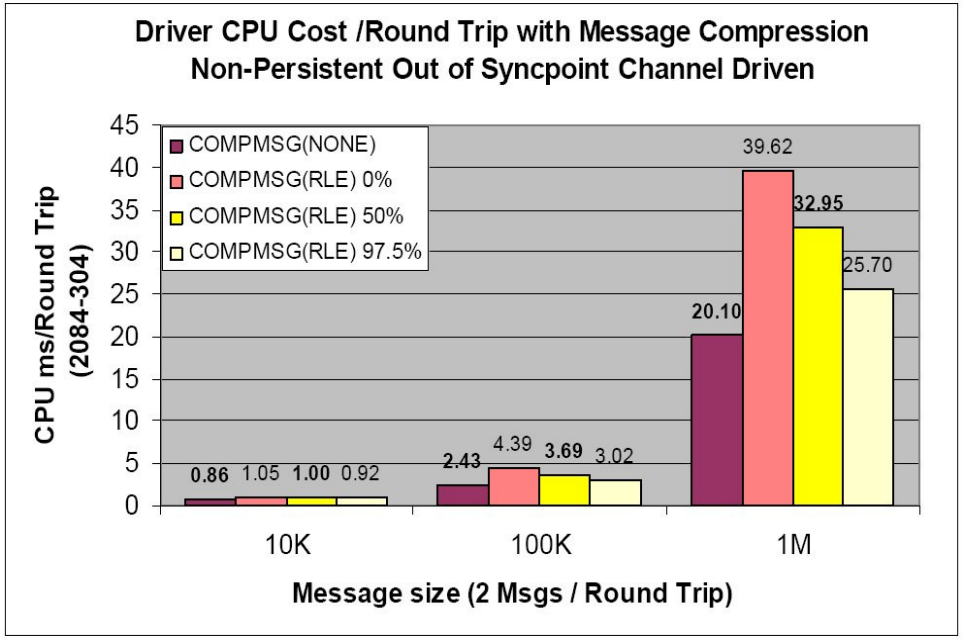
- Optional compression on data that flows across a channel
  - None
  - Message headers
  - Data compression
- Choice of compression techniques
  - RLE
  - ZLIB
- Performance dependant on numerous factors
  - Compressibility of messages
  - Elapsed time to compress messages
  - CPU cost for compression
  - Available network bandwidth

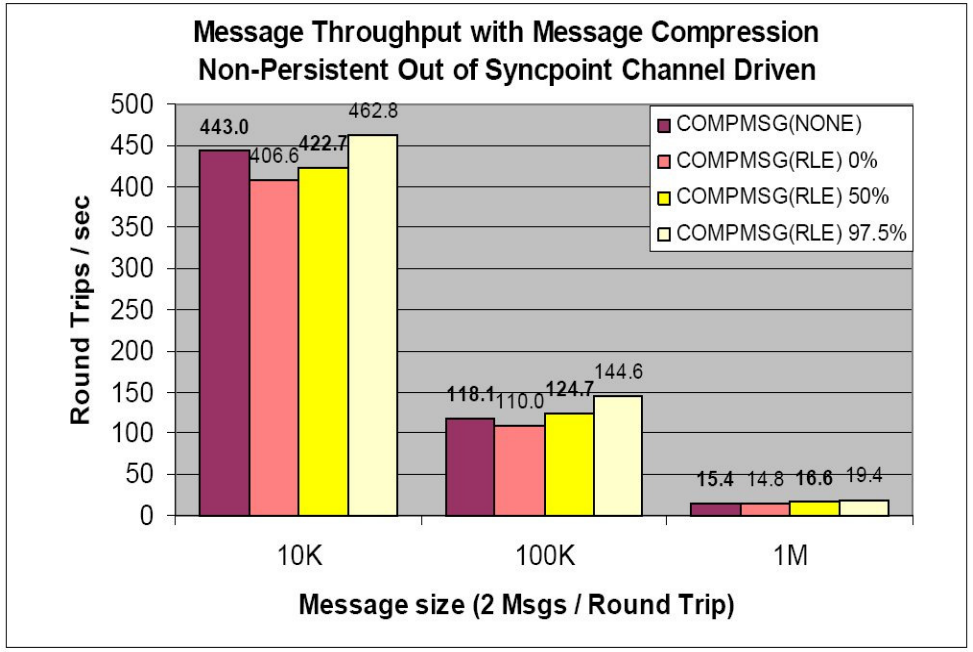
Channel message compression

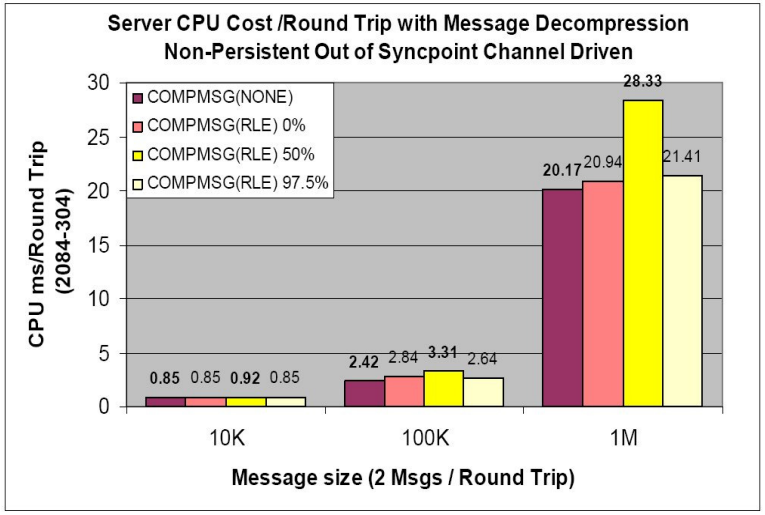
NOTES

- WebSphere MQ for z/OS V6.0 introduced new channel attributes to enable the compression of message headers and/or message data. A choice of compression techniques is available for message data compression; for message headers the choice is simply to enable compression or not. Compression of channel data reduces the amount of network traffic and can therefore improve the performance of channels.
- The benefits of channel compression will vary greatly depending on a number of factors: the compressibility of the messages; the elapsed time spent in performing compression; the amount of CPU consumed during compression and decompression; the available bandwidth of the network. All of these factors will differ from one installation to the next, and may vary from day to day within a single installation.

Channel message compression







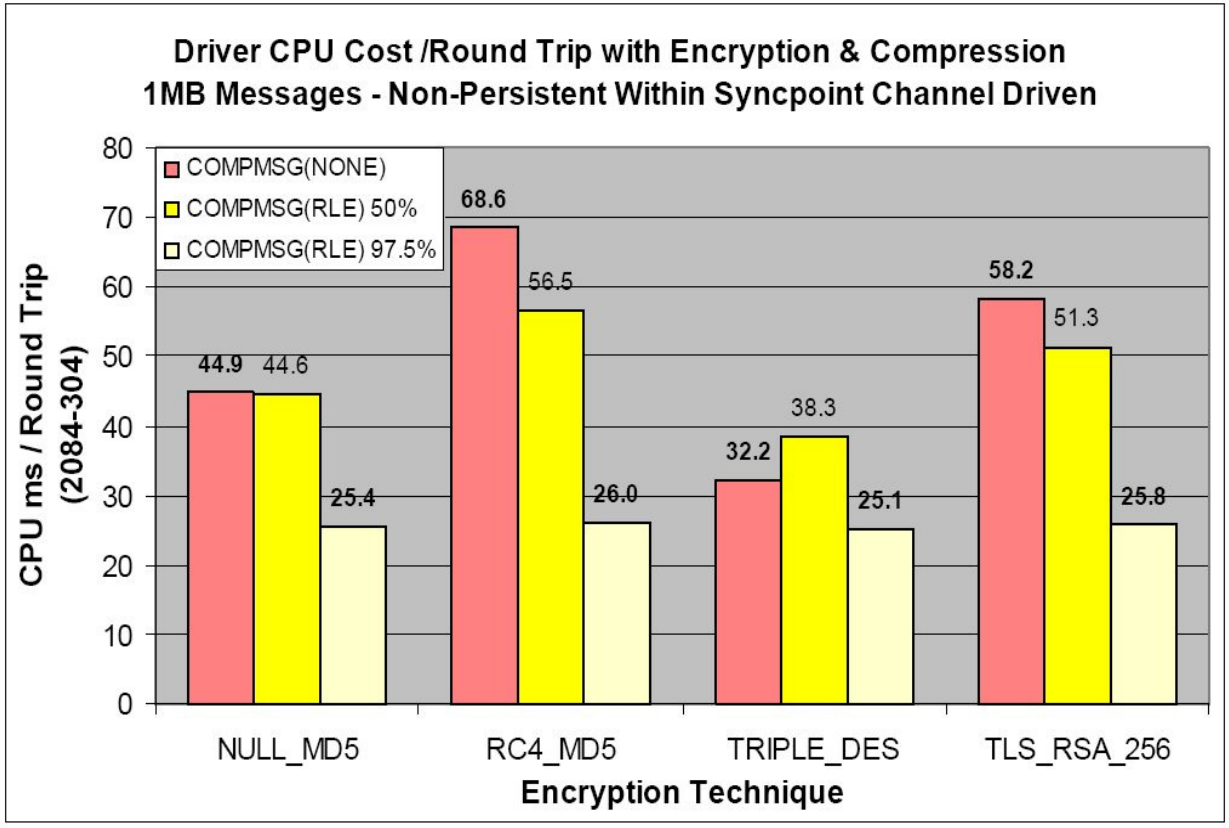
Channel message compression

NOTES

- We measured the effects of compression using a remote request reply model. A message is MQPUT by an application running against one MQ Queue Manager (Queue Manager A) to a remote queue hosted by a second MQ Queue Manager (Queue Manager B). A reply application running against Queue Manager B MQGETs the message and then MQPUTs a reply message of the same size to a reply queue hosted by Queue Manager A. Upon receipt of the reply message, Queue Manager A then MQPUTs another message, and the process continues for a defined number (usually tens of thousands) of cycles. This task is performed first with no compression, and then with compression of the request message only. In this way we can determine the cost of compression at Queue Manager A, the cost of decompression at Queue Manager B, and changes to the number of round trips per second. All CPU figures quoted relate to the total CPU (MQ Queue Manager + MQ Channel Initiator + TCP/IP + Application) per transaction.
- The charts show the effects on CPU costs at the compressing (driver) end and the decompressing (server) end of the transaction, and the effect on throughput. Note that, as described above, only the request message undergoes compression (or attempted compression); the reply message does not pass through the compression routines. The CPU costs displayed on the charts are the total costs (MQ Queue Manager + MQ Channel Initiator + TCP/IP + Application) for servicing two messages (request and reply), but any differences in CPU cost are the result of compressing request messages at the driver or decompressing them at the server.

Channel message compression with SSL

- Benefits seen when using compression with SSL
- Cost of compressing data offset again cost of encrypting data

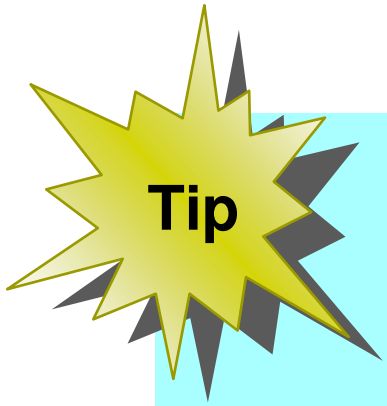


Channel message compression with SSL

NOTES

- For some installations it is possible the cost of encrypting messages can be partially offset using message data compression; although there is a cost associated with compression, there may be a greater gain through encrypting a smaller amount of data.
- The chart gives an indication of how compression can affect the CPU costs of sending and receiving encrypted messages. The value of enabling compression routines on the channel will be highly dependent upon the compressibility of the data, and in many cases the introduction of compression will result in an additional processing cost.
- As would be expected, the more expensive encryption techniques show the greatest potential for improvement.

Tip – VTAM RU size



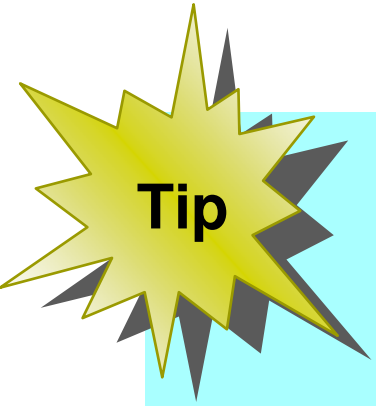
Use largest possible RU size for network

Message size	RU size 256	RU size 3480
1000	66ms	44ms
10 000	332ms	88ms

Tip – channel exits

Exits within chinit often use GETMAIN and FREEMAIN

- **Impact at high throughput**
- **Doubles the cost**



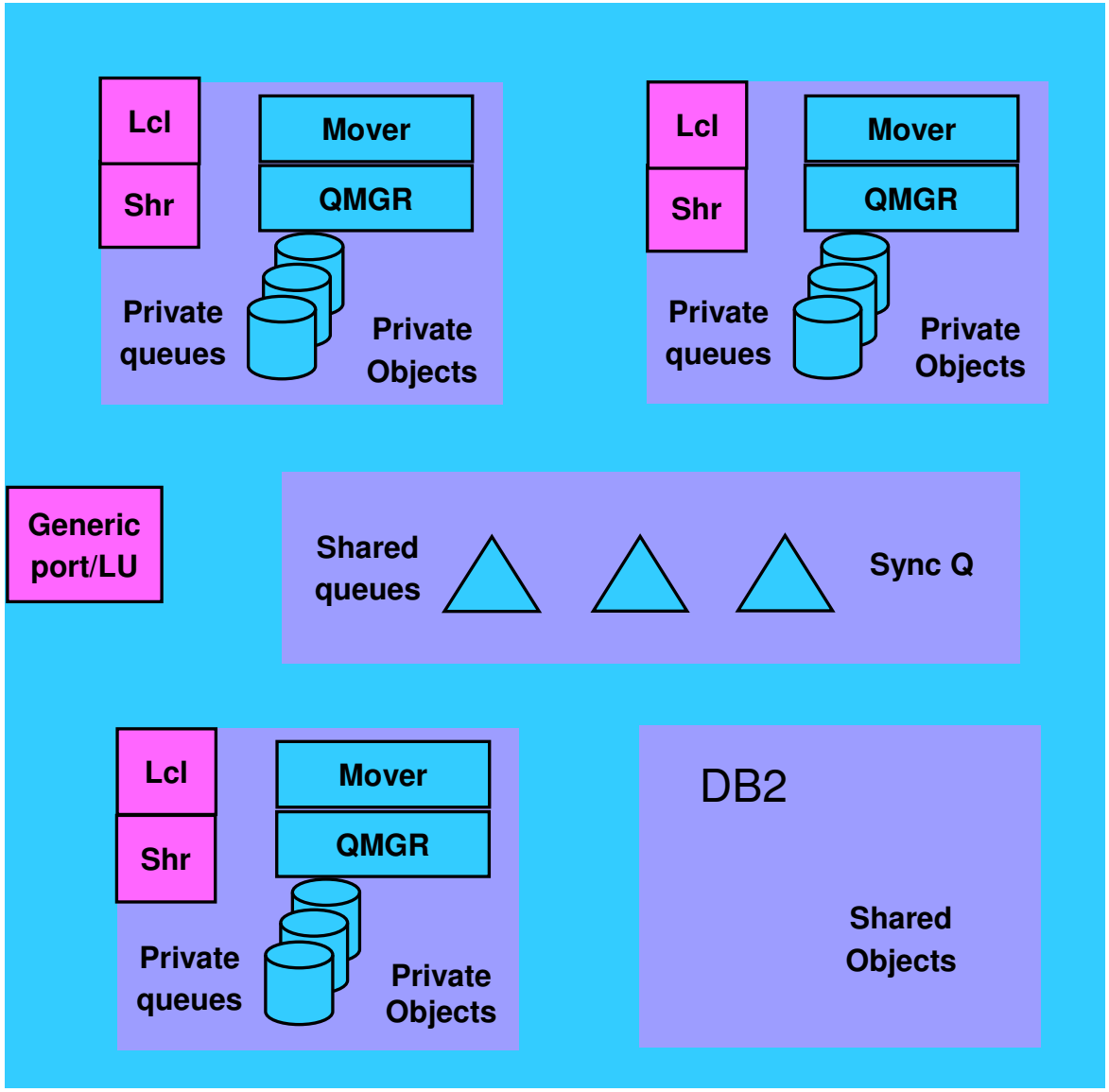
Exploit ExitBuffer:

- **Allocate buffer on initial call -- returns ExitBufferAddress**
- **ExitBufferAddress gets passed to every exit call -- so reuse the storage**
- **Free buffer on term call**

Shared Queues

Shared Queues

Shared Queue

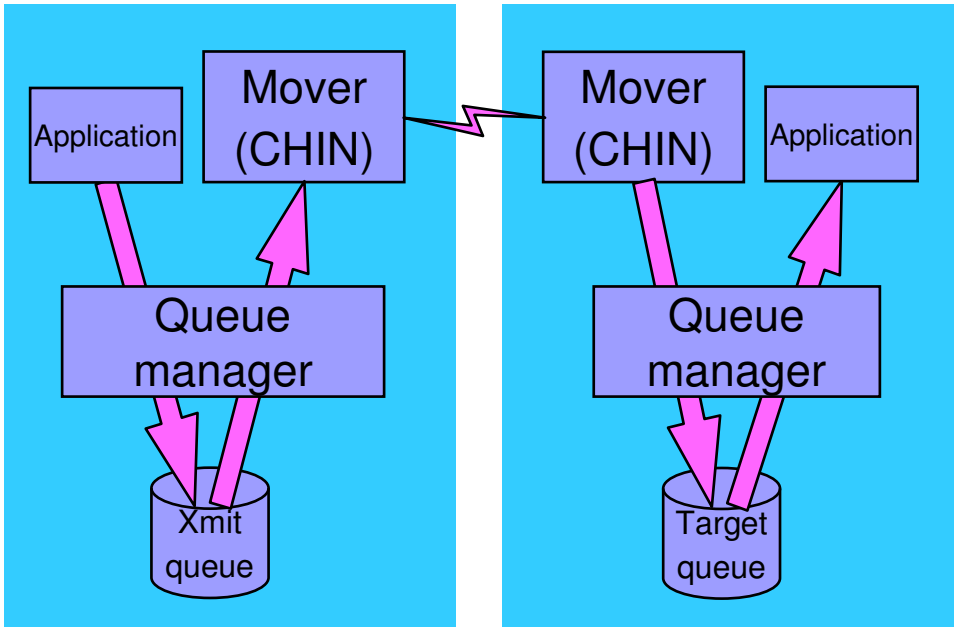


Shared Queue

NOTES

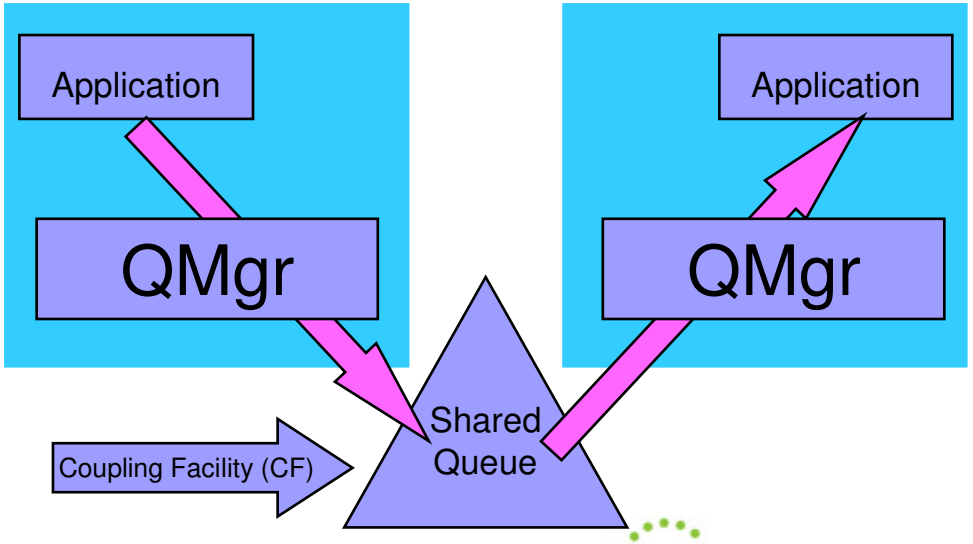
- Chart shows a queue-sharing group of three queue managers in the same sysplex (outer square).
- Each queue manager functions as a "traditional" queue manager with its own private queues and objects.
- Each queue manager also functions as a member of the queue-sharing group, sharing queues held on coupling facilities (CFs, shown as triangles).
- Shared object definitions for the queue-sharing group are held in DB2.
- Define once, available everywhere
- Generic port/LU for mover
- Benefits of shared queues include:
  - Availability
  - Systems Management
  - Scalability
  - Workload balancing (pull model rather than push)
  - Non persistent messages are available after restart

Shared Queue – Cost Savings



Elimination of message flows saves costs

Simplified administration saves costs



Shared Queue – Cost Savings

NOTES

- Chart contrasts moving messages between queue managers using the chinit (top left) with using shared queue (bottom right).
- Shared queue avoids the processing costs of the chinit and greatly simplifies system administration.

Shared Queue – tuning and setup

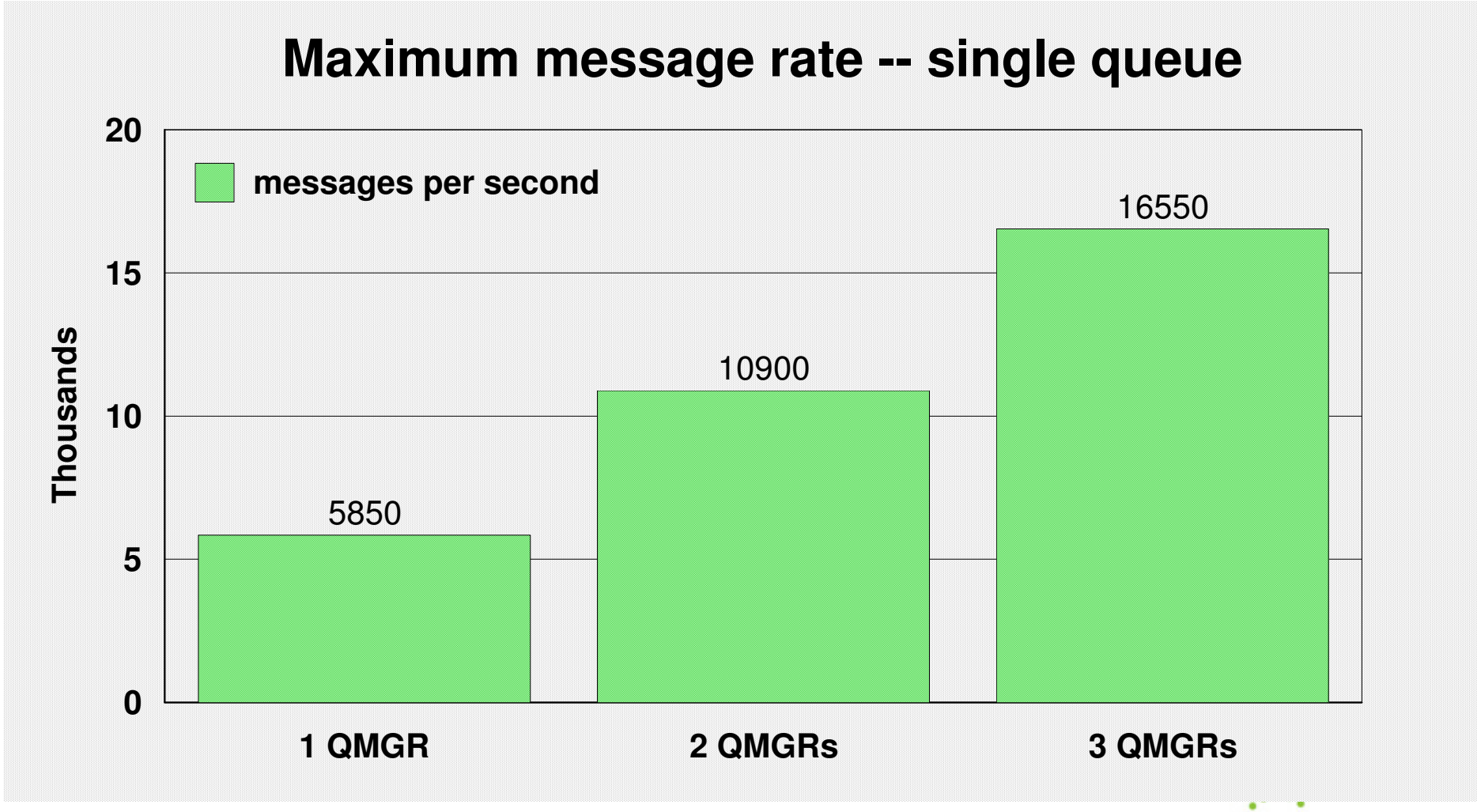
- Make CF structure(s) big enough:
  - 7500 1KB message = 16MB
- Use single structure rather than multiple in same unit of work
- Shared queue performance depends on CF link type:
  - ICP link (same box) – fastest
  - CIB link (150 metres)
  - CBP link (10 meters)
  - CFP link (27km) -- slowest (100MB/sec technology)
- Number of DB2 tasks -- use default but keep an eye on Q5ST->DHIGMAX (not more than 10)

Shared Queue – tuning and setup

NOTES

- Chart provides some guidelines for configuring shared queues to provide optimum performance.
- Where possible, keep related queues in the same application structure -- additional overheads are incurred if a unit of work affects queues in more than one structure.
- You may need more than one structure because any one structure can contain a maximum of 512 queues. Also you will need more than one structure if you want to use more than one coupling facility.
- There are several types of coupling facility links:
- Internal coupling (ICP) can only be used with a CF which is an LPAR of the same processor. It uses very fast memory-to-memory technology.
- Infiniband (CIB) is the fastest technology for an external CF, 6GB between z10 ECs
- Slower coupling facility links (CFP) can connect a CF up to 27 kilometers distant.

SQ non-persistent message throughput scales



Shared queue - scaling

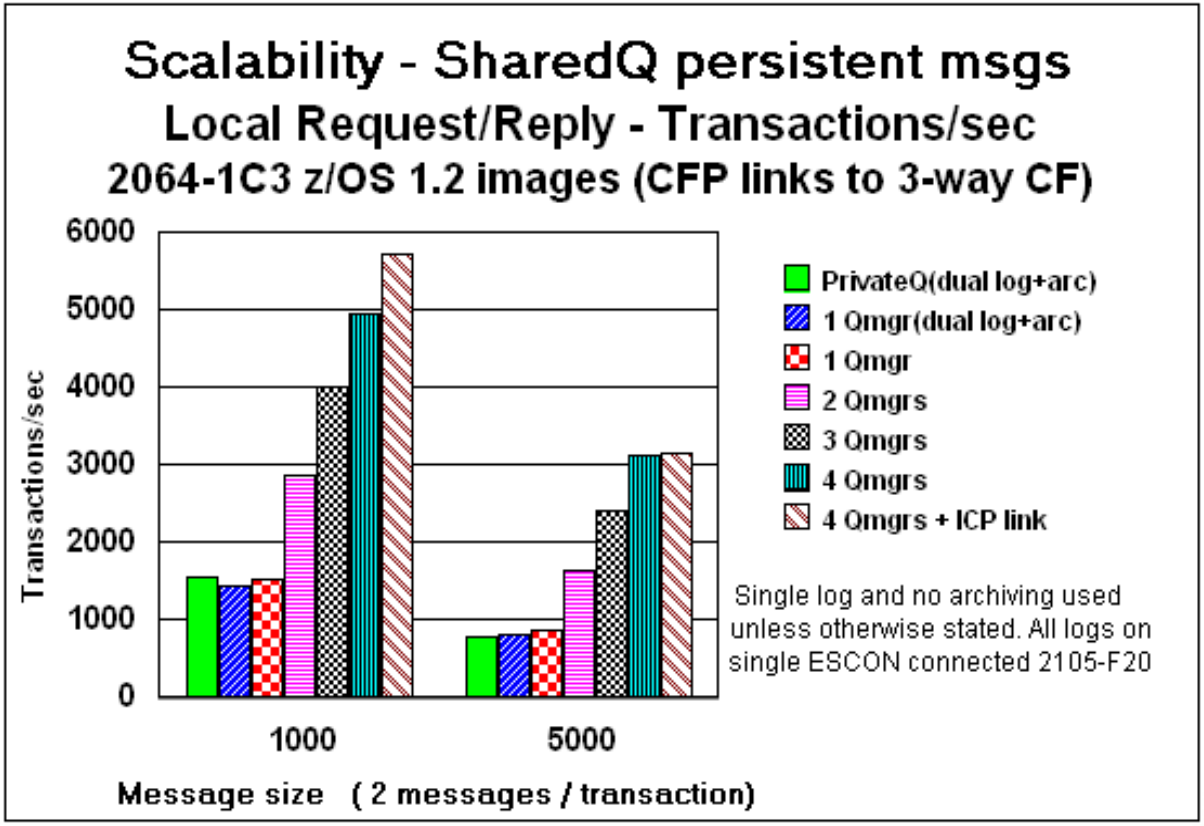
NOTES

- Chart shows measured results on lab setup -- actual numbers of messages processed will vary depending on the equipment and configuration used.
- Lab setup does not include business logic.
- In all cases one queue manager per z/OS image.

SQ persistent message throughput scales

- 11,000+ persistent messages/sec using 4 qmgrs

- Log DASD still likely to be first limit on throughput



CF Link types / z/OS XCF heuristics

NOTES

- Throughput and CPU cost depend on load
- CFP slowest, most expensive - up to 27 Km
- CBP 'ClusterBus' - 10 metres
- ICP only to CF LPAR'd within same box
- CBP performs more like ICP than CFP? (did on 9672 anyway)
- CF calls can be synchronous or async according to z/OS heuristics
- Async likely to mean order of 130+ microsec wait
- Sync means z/OS CPU is busy while call data sent, CF processes, return data received

  typically 50+ microsecs
- Technology trends - CPU speed increasing faster than link speed

SQ backup and recovery

- BACKUP of 200,000 1KB SQ persistent messages takes 1 CPU secs (2084-303)
- RESTORE of that fuzzy backup takes 31 secs using DS8000 DASD
- Re-apply log data since backup taken @ 9MB/sec on this DASD
- Logs read (backwards) in parallel by single qmgr
  - so longest log determines recovery processing time (15,660MB in 29 mins)
- How often should CF structure be backed up for 30min recovery time ?

1KB persistent msgs/sec to longest log	1KB persistent msgs/sec to 3 evenly loaded logs	MB/sec to longest log	Backup interval in minutes (based on reading logs backwards at 9MB/sec)
1000	3000	2.27	115
2000	6000	4.61	56
6400*	19200	15	18
(*6400 is the maximum for this DASD with 1KB msgs)			

Version 7 Updates

Version 7 Updates

Version 7 Updates

- Small messages
- Log compression
- Large shared queue messages (using SMDS)
- Some z196 performance figures

Small messages

- Small (< 4KB) message storage changed
  - One message per page instead of fitting multiples into single page
  - May increase DASD use, especially if messages not quickly retrieved
  - But typical workloads show improved performance and reduced CPU
    - One customer has reported a 50% reduction in processing time between a workload running on MQ V6 and the same workload running on MQ V7.0.
  - Note this is the DEFAULT behaviour in V7.0.1 but it can be reset by the RECOVER QMGR (TUNE MAXSHORTMSGS 0) command as described in MP1G

Log compression

- Can increase the throughput possible for persistent messages
- May reduce the size of your logs
  - Dependent on your message content
  - Useful if you are DASD constrained.
- RLE (run-length encoding) of “insert” log records for private queue messages
  - Will not compress shared queue log records
  - SMF 115 records updated to show compression rates achieved etc
- Controlled via ZPARM option at queue manager level.
  - COMPLOG(NONE) or COMPLOG(RLE) in CSQ6LOGP
  - Can also be viewed/controlled via DISPLAY LOG / SET LOG
- By compressing a 100KB message that contained 90% compressible data, it was possible to increase the transaction rate by over 3 times (see SupportPac MP1G).

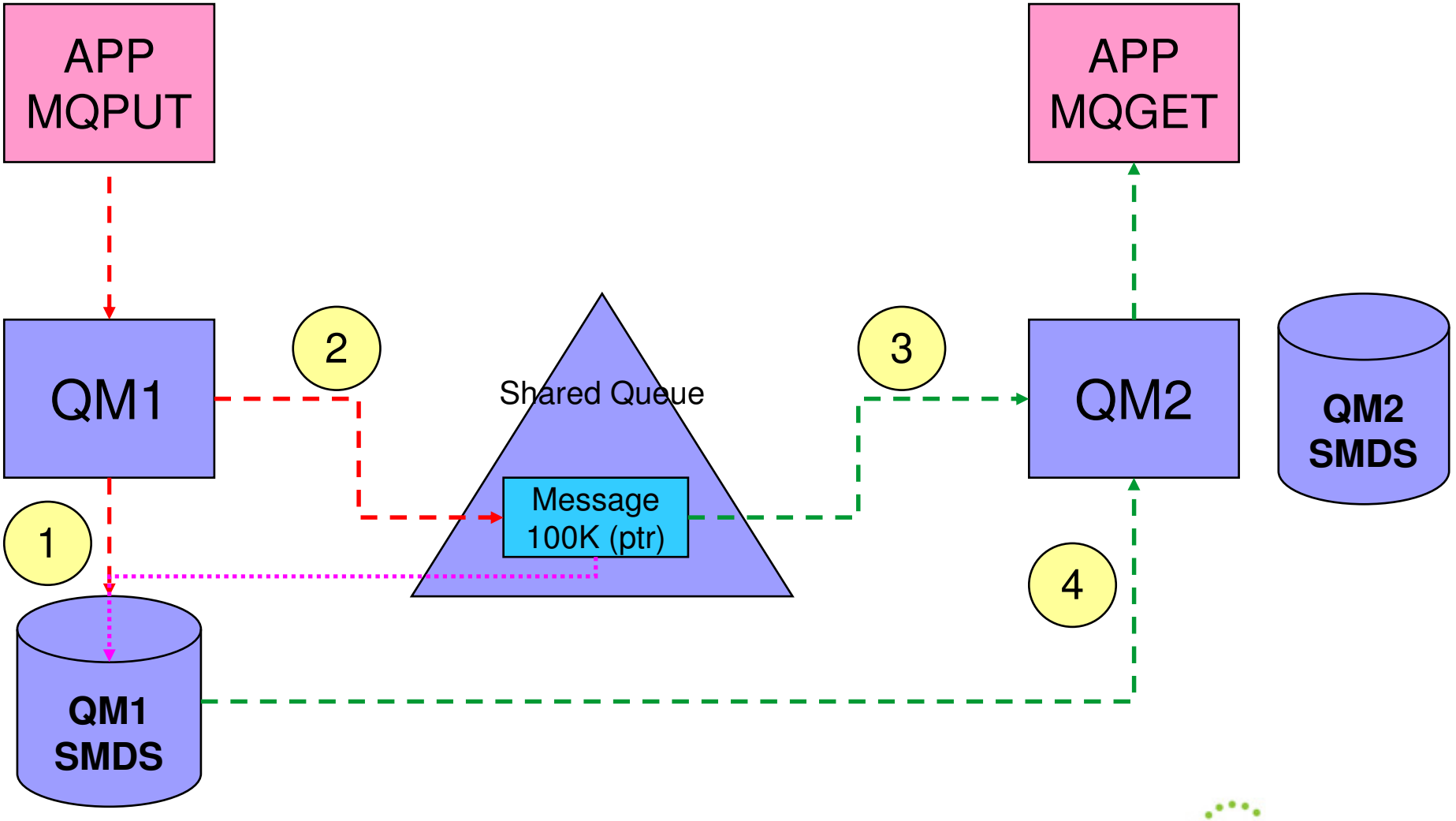
Log compression

NOTES

- Log compression improves the I/O processing, reducing the bandwidth required to log persistent messages. Clearly there is some CPU cost associated with the compression (and, if required during recovery, decompression). But this may be offset by the I/O savings – and for some customers, it's the I/O usage that was most critical in limiting their workload.
- Compression is done using the same run-length encoding algorithm that is an option on channels.

Large Shared Queue Messages (using SMDS)

V7.1



Shared message data set concepts

NOTES

Offloaded message data for shared messages is stored in data sets.

Each application structure has an associated group of shared message data sets, with one data set per queue manager.

Named using DSGROUP parameter on CFSTRUCT definition.

Each queue manager owns a data set for each structure, opened for read/write access, which it uses to write new large messages.

Each queue manager opens the data sets for the other queue managers for read-only access, so it can read their message data.

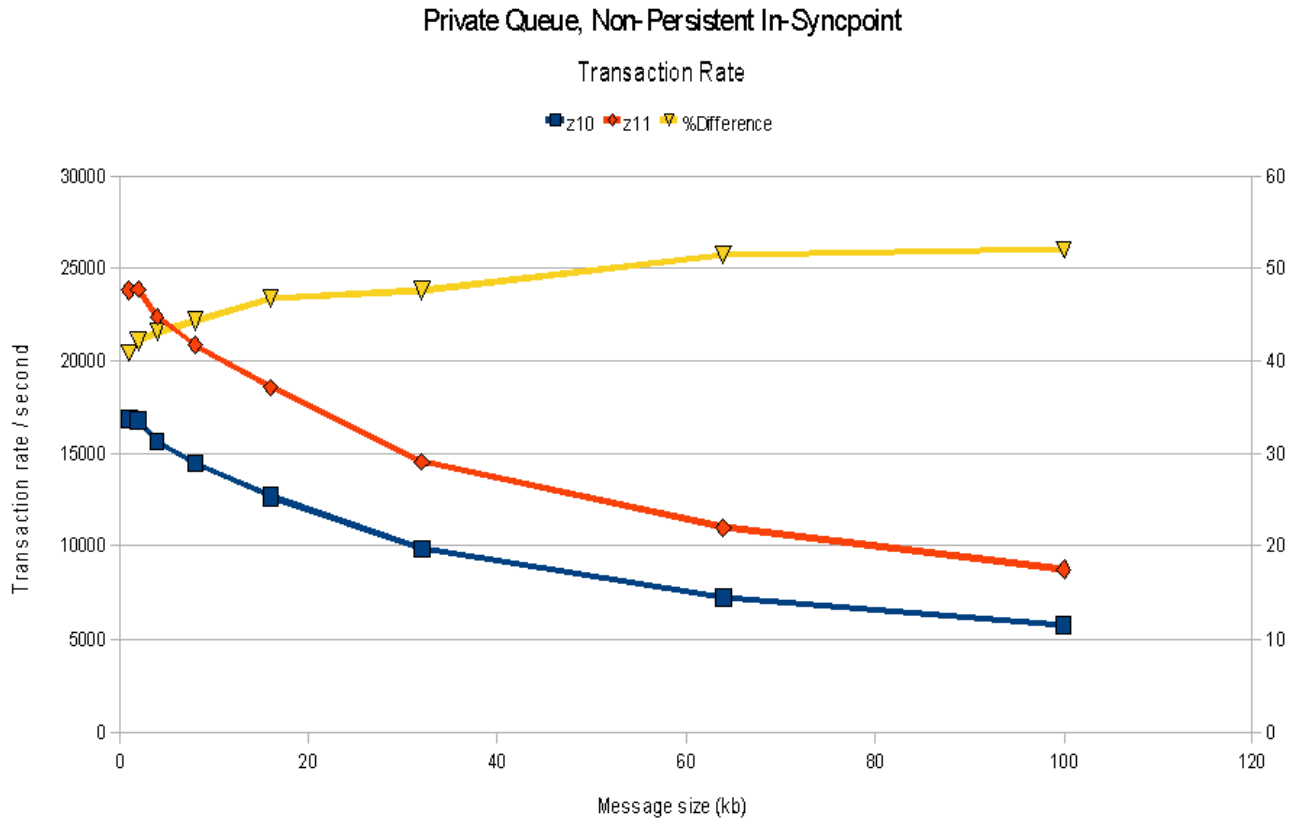
When a message with offloaded data needs to be deleted, it is passed back to the queue manager which originally wrote it, so that the queue manager can free the data set space when it deletes the message.

Large Shared Queue Messages: SMDS

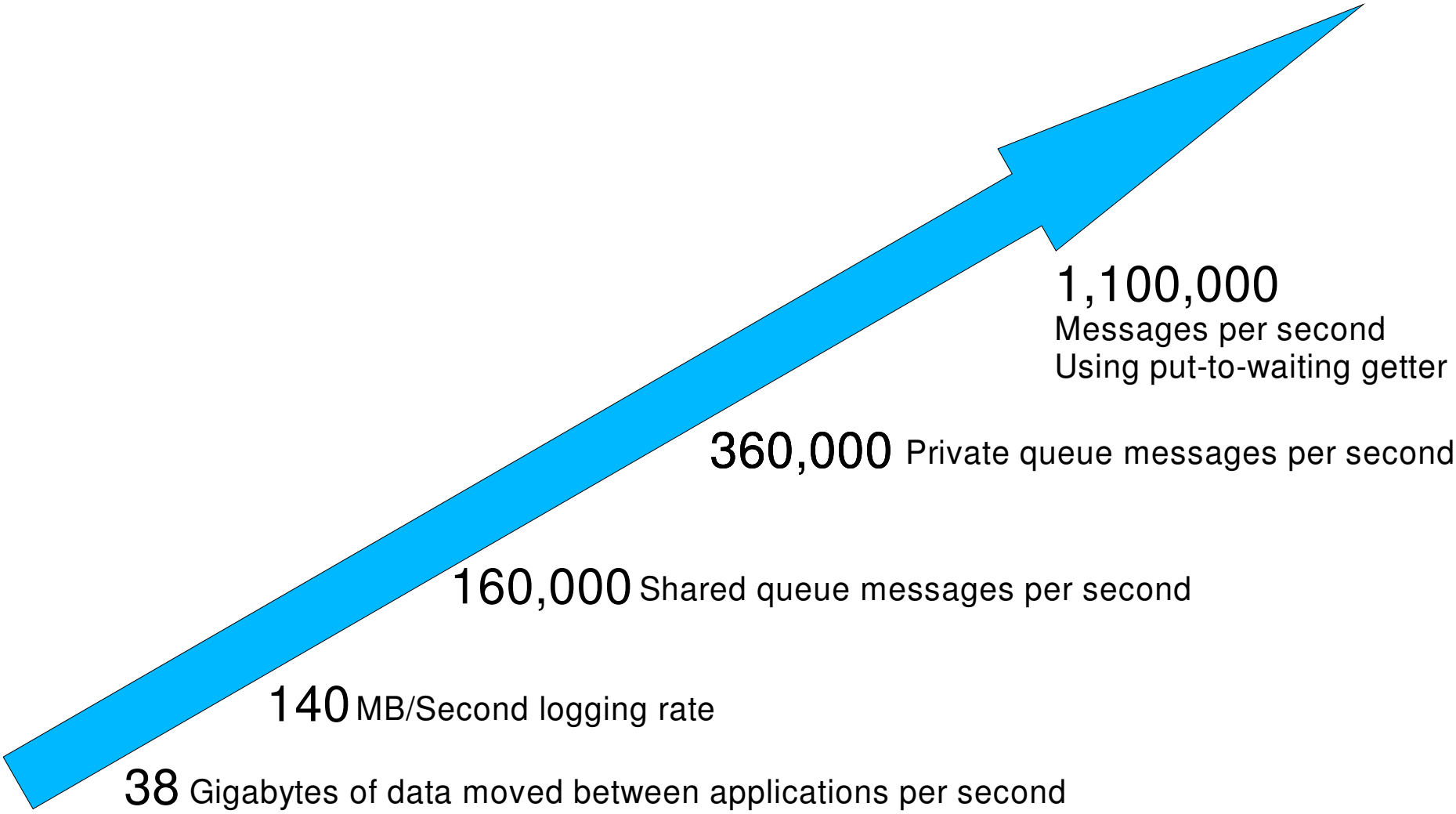
- Using DB2 BLOBs to store large (>63KB) messages is expensive
  - Both CPU and pathlength
- Shared Message DataSets (SMDS) removes DB2 for large message storage
  - DB2 still needed for storing shared definitions
  - CF still holds small messages and pointers for offloaded messages
- Shared VSAM datasets increase shared queues capacity and performance
  - All queue managers in the QSG can access the dataset
- CF Structure message reference still controls locking, ordering, deletion etc.
  - So every message still has a “pointer” in the CF
- Rules control offload message size and % Structure-full offload trigger
  - Set per CF structure
  - Offloading messages at 63K gives 1.25M messages on a 100GB structure
  - Offloading all messages at 1KB gives 45M messages on same structure
- All QSG members must be at new level to access queues with this capability

MQ V7.0.1 z10 / z196 performance comparisons

- Tests run in Hursley for a variety of workloads show performance improvements in line with LSPR predictions
  - The transaction cost is between 28 and 34% LESS on the z196 compared to tl



MQ V7.1.0 z196 performance highlights



MQ V7.1.0 z196 performance highlights

NOTES

- The MQ V7.1.0 measurements were run in the following configuration
  - 1x64-way LPAR with 12-way CF
  - 2x30-way LPAR with 16-way CF
  - 2x24-way LPAR with 16-way CF
  - 3x8-way LPAR with 16-way CF
- Measurements were run using up to 3 queue managers for private queue messaging and up to 12 queue managers for shared queue messaging
- As well as using batch applications, ran Mover, CICS, IMS Bridge and Client measurements.

Accounting and statistics

Accounting and statistics


Why didn't anyone tell me?

What you can do with accounting

- Which applications used which queue, and what did they do with it!
- CPU cost per MQ verb per queue
- Is “Put to a waiting getter” being used
- How many messages were read from disk
- *Persistent v Non Persistent message counts*
- How many MQGETs were by msgid/correlid and how many message were skipped to find the right message
- Expired messages
- Which were the active queues
- Why is today slower than yesterday
- Id by Jobname, transaction, or channel name
- 'C' program source to process them – MP1B
- Costs 5-10%

What you can see with statistics

- Number of checkpoints (QJSTLLCP)
- Reads from active/archive logs - indicative of backouts (QJSTRACT/QJSTRARH)
- Amount of data being logged (QJSTCIWR)
- Is log buffer big enough (QJSTWTB)
- How effective is log compression (QJSTCmpUncmp/QJSTCmpComp)
- Number of MQ verbs being issued (QMSTGET – number of MQGETs)
- Buffer pool usage – is buffer pool being stressed (QPSTDWT)
- Queuing on DB2 server TCBs (Q5ST. DHIGMAX)

For more information

http://www.ibm.com/software/integration/support/supportpacs/

MP1B	Interpreting Accounting and Statistics Data
MP1G	WebSphere MQ for z/OS V7.0.1 Performance Report
MP1E	WebSphere MQ for z/OS V6.0 Performance Report
MP15	Program to Print SMF Statistics
MP16	Capacity Planning and Tuning -- Updates
MD16	Getting the Best from MQSeries on OS/390

Questions?

Copyright Information