#SHAREorg

SHARE

S12622 – Are you running too Many Queue Managers or Brokers?

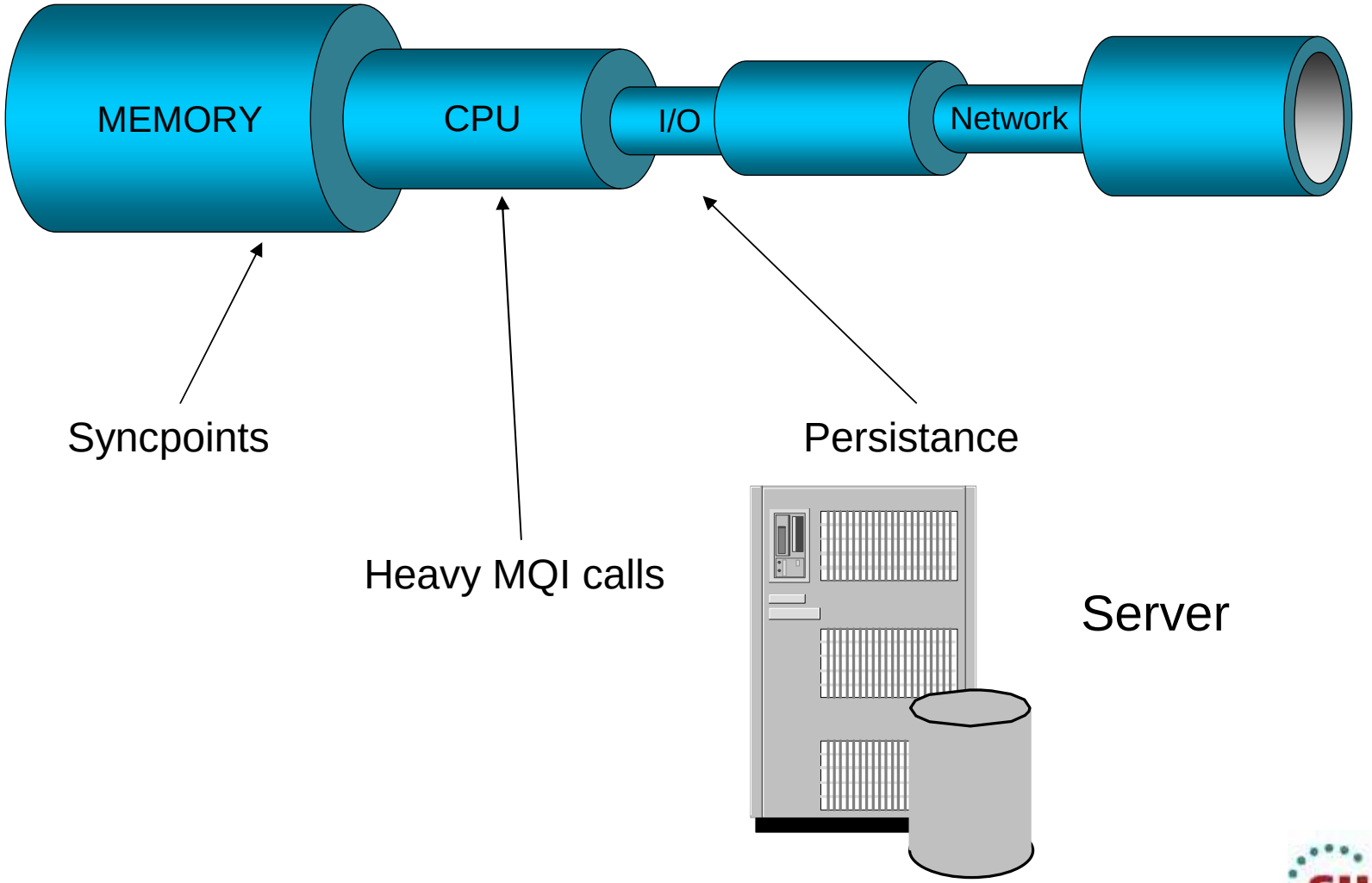
Ralph Bateman and David Gorman
IBM Hursley

S12622

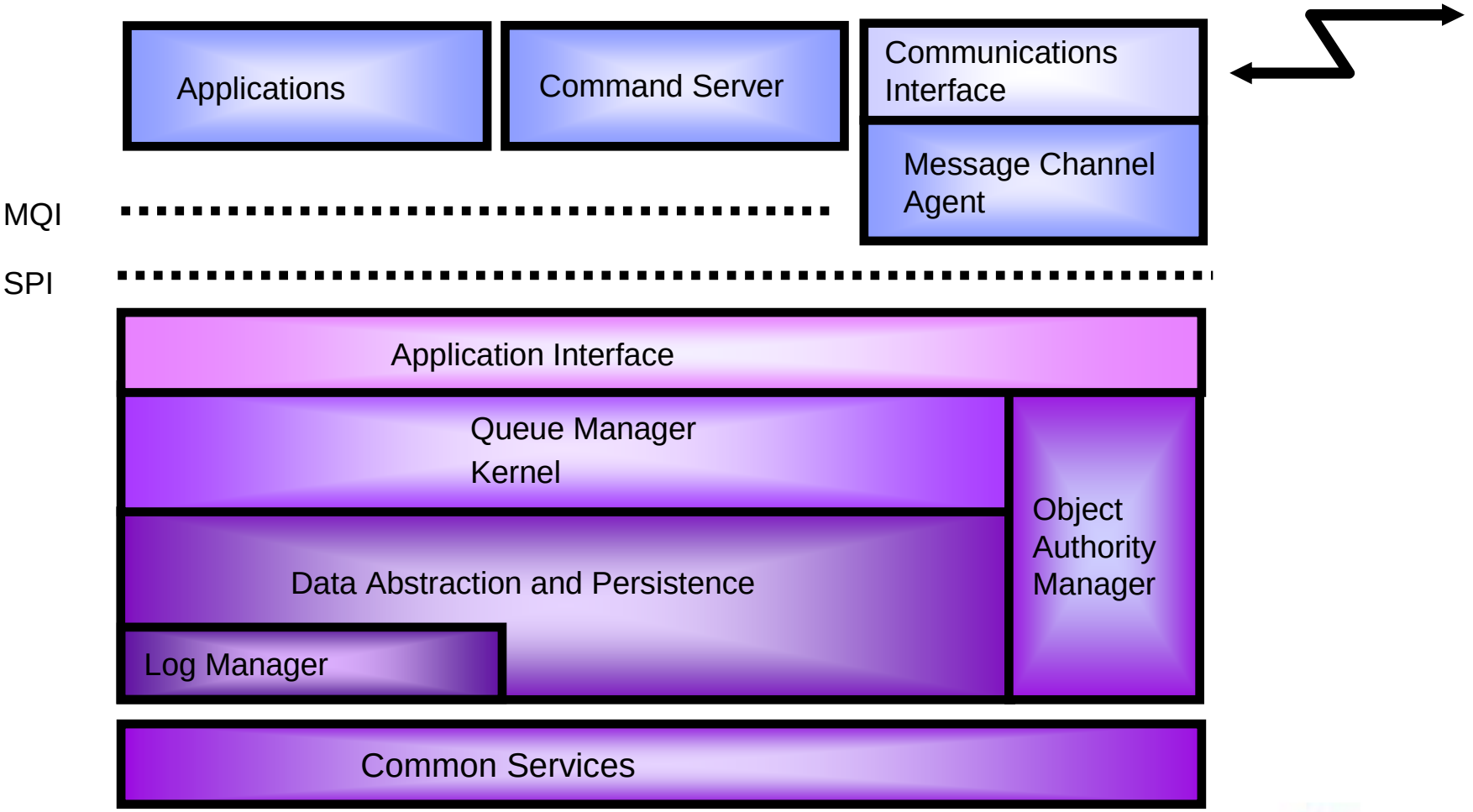
SHARE in San Francisco 2013

Agenda

What should we look at first



Queue Manager: Functional View



Queue Manager: Functional View

- The queue manager has the following major components:
- **Application Interface** provides the environment and mechanism for execution of MQI calls.
- **Queue Manager Kernel** provides most of the function of the MQI. For example, triggering is implemented here along with message location.
- **Object Authority Manager** provides access control for the queue manager and its resources. It allows specification of which users and groups are permitted to perform which operations against which resources.
- **Data Abstraction and Persistence** provides storage and recovery of the data held by the queue manager.
- **Log Manager** maintains a sequential record of all persistent changes made to the queue manager. This record is used for recovery after a machine crash and for remembering which operations were in which transaction.
- **Message Channel Agents** are special *applications* using the SPI for the majority of their operations. They are concerned with reliable transmission of messages between queue managers.
- **SPI** is a lower-level API similar to MQI but only available to Queue Manager processes offering greater performance and functionality.
- **Command Server** is a special application. It is concerned with processing messages containing commands to manage the queue manager.
- **Common Services** provides a common set of operating system-like services such as storage management, NLS, serialisation and process management. This isolates the Queue Manager from platform differences.

Syncpoint – Are they that important?

- Do you need it?
  - Yes, when a set of work needs to either all be performed, or all not performed
- Maximum Size of UOW can be limited
  - QMGR MAXUMSGS parm
  - Set to sensible value to avoid runaway applications
- Make sure you keep the size of your UOWs small
  - Don’t forget to end the UOW
- Cheaper to process in syncpoint for persistent messages
  - Up to a point, not huge UOWs
  - Log not forced after every MQPUT/MQGET
- Useful even when only a single message inside syncpoint
  - And running multiple parallel applications

Syncpoint

- The size of a UOW (number of messages in it) can be controlled via the MAXUMSGS queue manager parameter. This however has no impact on the duration of the UOW, it simply controls the maximum number of messages that a UOW can contain. It prevents runaway applications

- It can be considerably cheaper to process multiple persistent messages inside syncpoint rather than processing them outside syncpoint. This is because if persistent messages are being used outside of syncpoint, it is necessary to force them to the log as soon as they are put, to ensure that they are available if a failure occurs. If they are processed inside syncpoint it is only necessary to force the log when the UOW is committed. This means that we will spend less time waiting for the pages to be forced out to disk. In effect the cost of forcing the UOW out to disk is shared between all of the messages put and got, rather than each one having to bear the cost. Syncpoint should not be considered as an ‘overhead’.

Heavyweight MQI Calls

- MQCONN is a “heavy” operation
  - Don’t let your application do lots of them
  - Wrappers and OO interfaces can sometimes hide what’s really happening
  - Lots of MQCONNs can drop throughput from 1000s Msgs/Sec to 10s Msgs/Sec
- MQOPEN is also ‘heavy’ compared to MQPUT/MQGET
  - Depends on the type of queue and whether first use
    - Loading pre-existing queue; creating dynamic queue
  - It’s where we do the security check
    - Try to cache queue handles if more than one message
  - If you’re only putting one message consider using MQPUT1
    - Particularly client bindings
- Try to avoid exclusive access to the Queue
  - Makes it harder to scale the solution
    - For example adding more instances of application
  - Implies that reliance on message order is not required
    - Partition the data to allow parallel processing?

Heavyweight MQI Calls

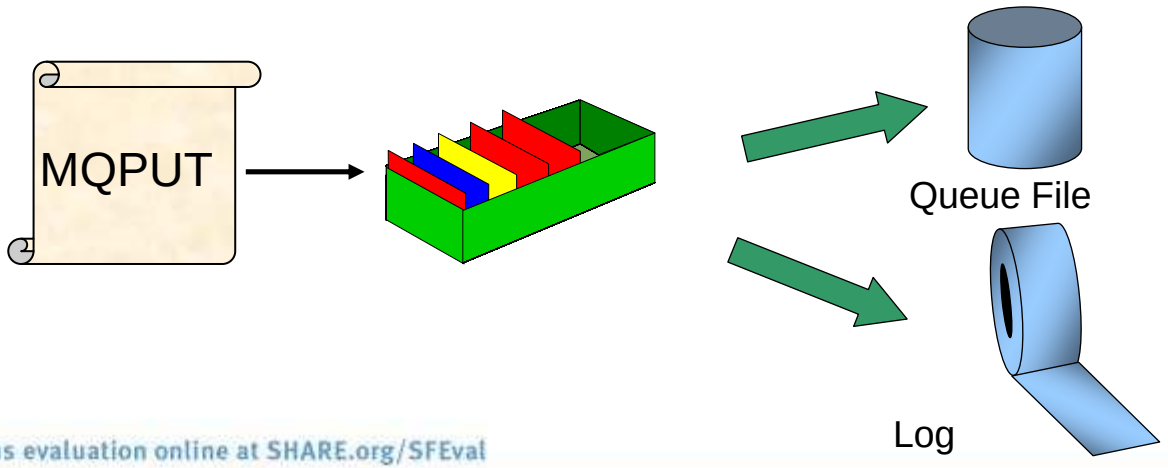
- MQCONN is a very heavyweight operation. Doing lots of these calls could cause throughput to suffer. Make sure that you don't connect and disconnect a lot in your application, rather, connect once and then use this connection for all subsequent operations. Think carefully about any encapsulation you might do in your OO applications, make sure that the encapsulation does not cause you to do lots of MQCONNs and MQDISCs.

- MQPUT1
  If just putting a single message to a queue, MQPUT1 is going to be cheaper than MQOPEN, MQPUT and MQCLOSE. This is because it is only necessary to cross over from the application address space into the queue manager address space once, rather than the three times required for the separate calls. Under the covers inside the queue manager the MQPUT1 is implemented as an MQOPEN followed by MQPUT and finally the MQCLOSE. The cost saving for a single put to a queue is the switching from application to queue manager address space. Of course, if multiple messages need to be put to the queue then the queue should be opened first and them MQPUT used. It is a relatively expensive operation to open a queue.

- Exclusive use of queues
  Opening queues for exclusive use can help with sequencing issues, but it is a good idea to investigate whether other solutions are available. Exclusive use will make it harder to add extra tasks to process more work if needed in the future. Possible solutions are partitioning the data on the queue so that different tasks can work on different parts of the queue data (get by CorrelID can be used for this). This will enable more tasks to process the queue while maintaining ordering within the partitioned part of the data.

Persistence – do you need it?

- Log bandwidth is going to restrict throughput
  - Put the log files on the fastest disks you have
- Persistent messages are the main things requiring recovery after an outage
  - Can significantly affect restart times
- Why use persistence?
  - False assumption that persistence is for "important" data and nonpersistent for when you don't care
  - The real reason for persistent messages is to reduce application complexity
  - With persistent, apps do not need logic to detect and deal with lost messages
  - If your app (or operational procedures) can detect and deal with lost messages, then you do not need to use persistent messages



Persistence

- If persistent messages are used then the maximum rate that messages can be put is typically going to be limited by the logging bandwidth available. This is normally the over riding factor as to the throughput available when using persistent messages.
- As persistent messages need to be made available when a queue manager is restarted, they may need to be recovered if there has been a failure (could be queue manager or system etc). The persistent workload that has been done is the main key as to how long it is going to take to restart the queue manager after a failure. There are other factors involved which include the frequency of checkpoints etc, but ultimately it all comes down to the fact that persistent messages have been used. If there has been a failure then no recovery is required on non-persistent messages, the pages that contained them are simply marked as not used.
- 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.

What can you look at ?
Statistics and Accounting

- MQ collects sets of data to be written as messages to predefined queues
  - Very similar concepts to SMF 116 reports for z/OS but using MQ facilities
  - Data can be post-processed to give information on the activity of the system
- Statistical Data collection is split into 3 classes
  - QMGR statistics - statistics based on the activity of the whole system
  - Queue statistics - statistics on the activity of the queue (per queue)
  - Channel statistics - statistics on the activity of the channel (per channel)
- Accounting Data collects information about MQI applications
  - PCF message is written upon MQDISC
  - Or at regular intervals for long running tasks
- Collection of data for each class may be selected independently

Formatting Statistics/Accounting Events

- Sample program amqsmon to format
- Source code provided

```
amqsmon –m SHARE –t accounting
```

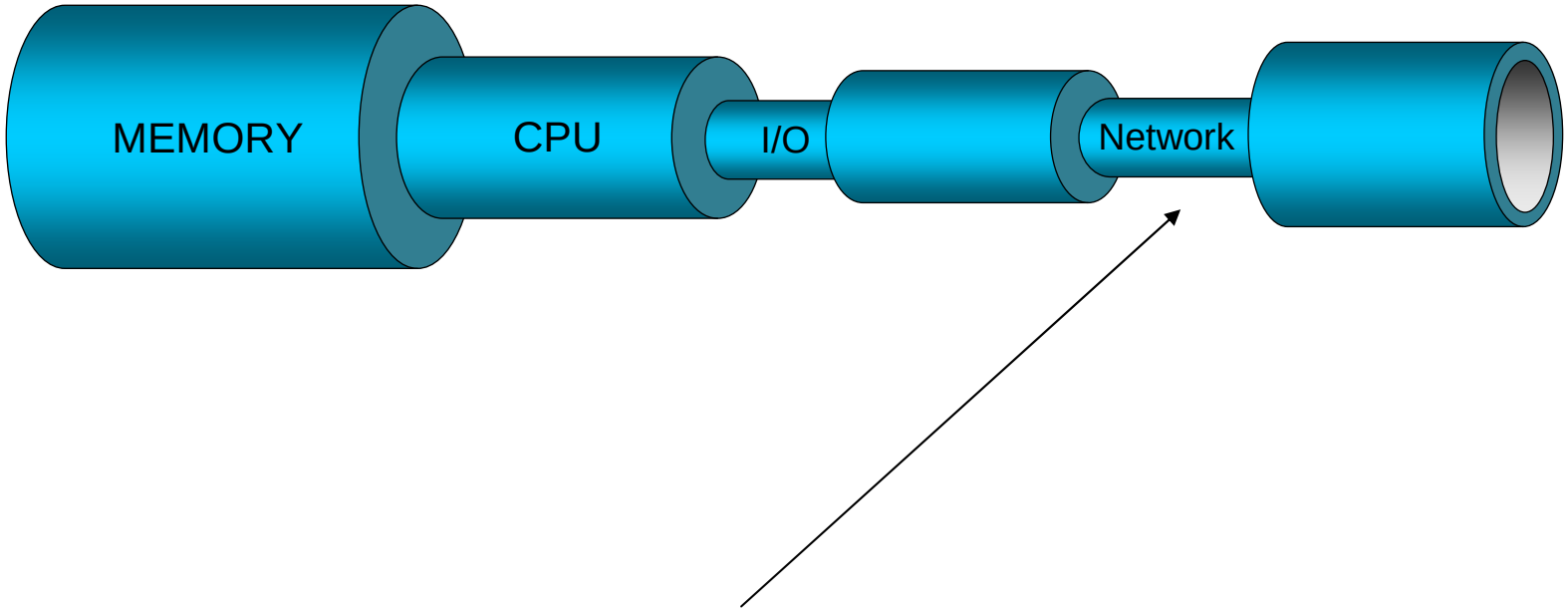
QueueManager: SHARE
IntervalEndDate: 2013-01-10
IntervalEndTime: 14:39:50
CommandLevel: 600
SeqNumber: 0
ApplName: **amqsput.exe**
ApplicationPid: 9408
ApplicationTid: 1
UserId: 'metaylor'
ObjectCount: 1

OBJECTS:
QueueName: 'APP.QUEUE.X'
QueueType: Predefined
QueueDefType: Local
OpenCount: **1**
OpenDate: 2013-01-10
OpenTime: 14:39:49
CloseCount: 1
CloseDate: 2013-01-10
CloseTime: 14:39:50

PutCount: [0, 1]*
PutFailCount: 0
Put1Count: [0, 0]
Put1FailCount: 0
PutBytes: [0, 4]
PutMinBytes: [0, 4]
PutMaxBytes: [0, 4]

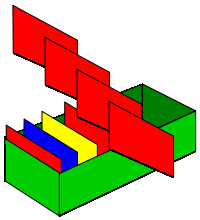
*array shows non-persistent, persistent

What should we look at next?

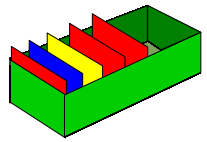


Network

Single or Batch of Messages



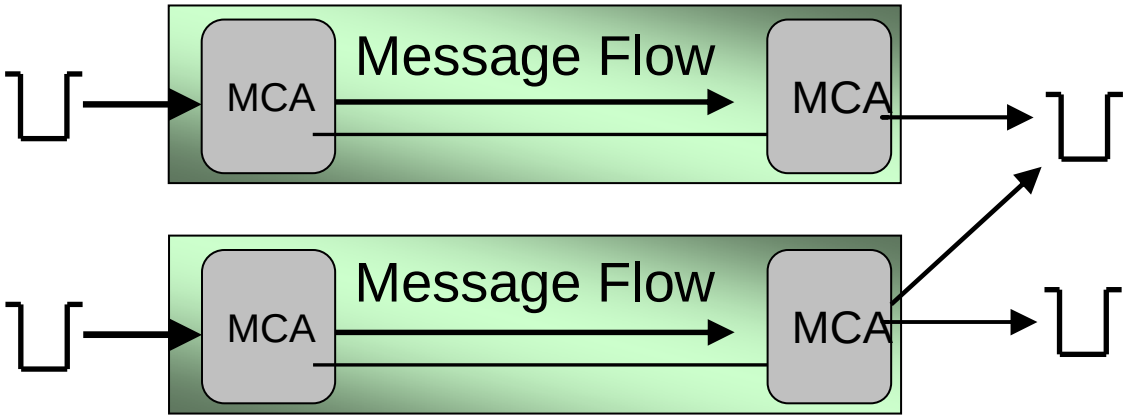
- Batchsize parameter determines maximum batch size
  - Actual batchsize depends on
    - Arrival rate
    - Processing speed
- Low Batchsize may cause XmitQ to build up
  - Scratchpad housekeeping uses equiv of 3 Persistent messages
  - High Batchsize will make little difference - self throttling
- Use high (~50) Batchsize unless there are issues with:
  - Data visibility - throughput
  - Communication link reliability
  - Message Size: Can use BatchLimit with V7.1
- Batchint can increase effective batch size towards Batchsize
  - Delays message availability to application
  - Reduces CPU cost per message



Channel Batch Size

- Setting an appropriate Batchsize (BATCHSZ) is a difficult issue and is related to the following factors:
  - Applications write messages to XMIT queues for moving over channels to remote systems. Channels take batches of messages from XMIT queues and move to destination. The overhead per batch (commit, CPU and disk activity) is divided by the #messages in the batch to give the cost per msg. If the commit process occurs less often then the message transfer rate is increased.
  - A batch is ended when one of two things happen - either the number or size of messages transferred has reached the maximum allowed for the batch or the transmission queue is empty and the Batchint has expired. If the message arrival rate is lower than the message transfer rate then the effective batch size is dynamically reduced as a drained transmission queue implies 'end of batch'. This reduction in batch size will reduce the message transfer rate as commit processing is more frequent and increases the cost of processing each message by the channel.
  - Messages arriving at the receiving MCA are placed on the target application queues under syncpoint control. This means that they are not visible to any receiving applications until the commit is performed. If the batch size is large, messages may not be made available to receiving applications for some time which may have a severe impact on the message throughput of the overall system.
- Because the batch size is so greatly influenced by the message arrival rate on the transmission queue, it is generally recommended to set the Batchsize quite high(ie leave at default of 50) - unless there are contrary factors, which are:
  - Data visibility - due to (outstanding) commit processing.
  - Unreliable, slow or costly communication links, making frequent commit processing a necessity.
  - Large Messages. Upon restart it may be necessary to resend the last batch.
- Entirely non-persistent message batches do not use disk for hardening batch information (NPMSpeed(FAST)) but still cause a line turnaround.
- With MQ V7.1 you can also use BatchLimit as an additional control on the amount of data transferred in each batch. This can be helpful when the size of messages on a transmission queue varies significantly.

One or Multiple Channels

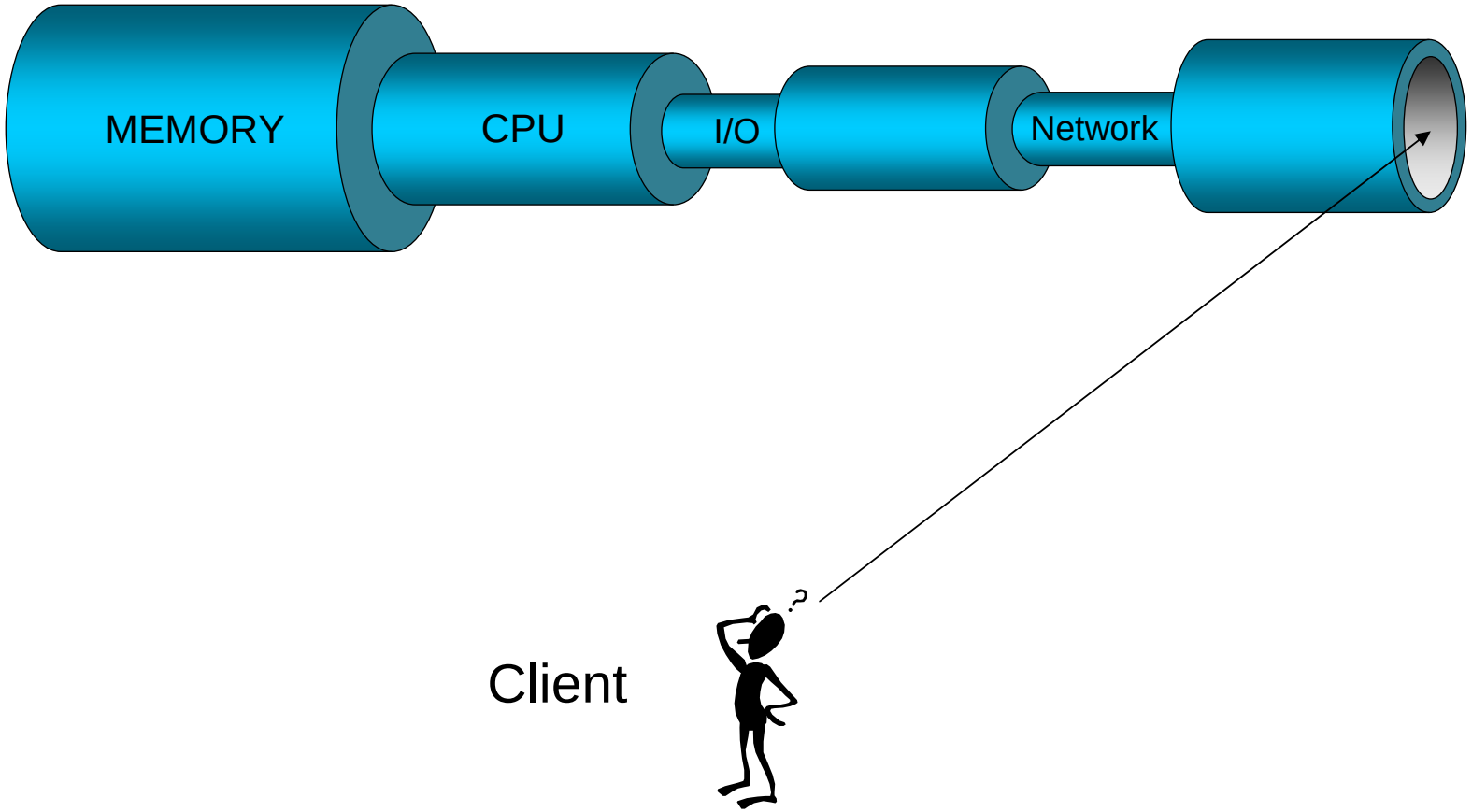


- Use one channel if it can handle the throughput
  - Monitor depth of XMIT queue
  - One high-use channel is more efficient than two low-use channels
    - The actual batch size will be larger
  - Multiple channels can yield some performance benefit
    - Depends on network and arrival rate
- Multiple channels for separate classes of work
  - Large messages only delay large message
  - Encryption cost on taken on worthwhile messages
  - Small interactive messages do not get delayed

One or Multiple Channels

- For the reasons previously outlined, it is most appropriate to have as high a channel utilization as possible. This means that it is appropriate to have as few channels as can handle the load between any two queue managers.
- However, where there are different classes of message being moved around the MQ network, it may be appropriate to have multiple channels to handle the different classes.
- Messages deemed to be 'more important' may be processed by a separate channel. While this may not be the most efficient method of transfer, it may be the most appropriate. Note that a similar effect may be achieved by making the transmission queue a priority order queue and placing these message at the head of the transmission queue.
- Very large messages may hold up smaller messages with a corresponding deterioration in message throughput. In this instance, providing a priority transmission queue will not solve the problem as a large message must be completely transferred before a high priority message is handled. In this case, a separate channel for large messages will enable other messages to be transferred faster.
- If it is appropriate to route message traffic through different parts of the underlying network, multiple channels will enable those different network routes to be utilized.

What should we look at next?



Clients

- Section on MQ Clients

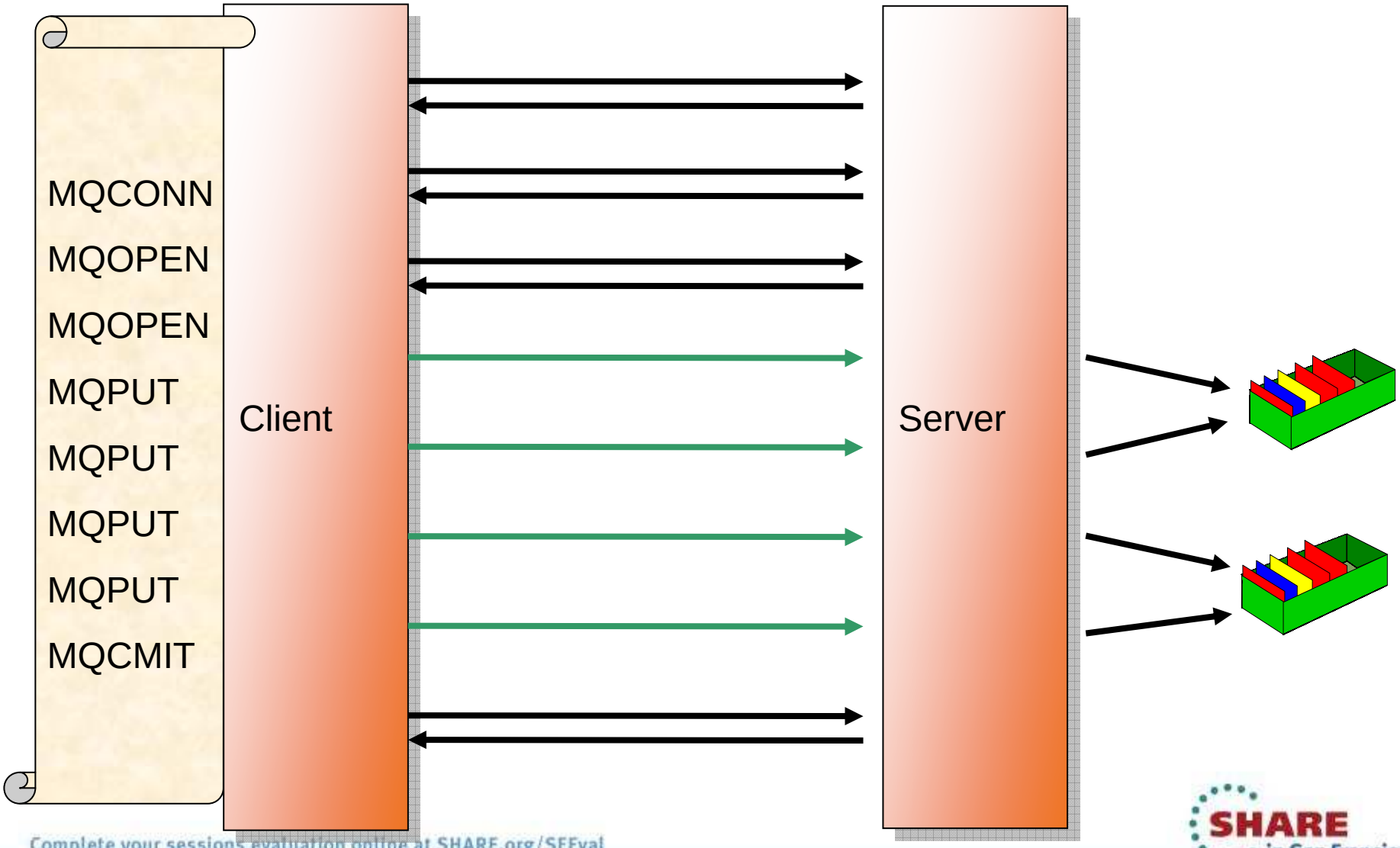
MQ Client Architectures

- Large systems have been built with clients
  - 50,000 clients per server
- Similar considerations to synchronous architectures
  - Request / response time critical
  - MQ Transport cost can be < 1 milli second
  - 256 byte user message causes 1420 bytes flow with serialised MQI parameters
- Scalability considerations
  - Large number of processes on server
  - Trusted bindings (Channel programs)
  - Overheads of per-client ReplyToQ
    - Share queues, using CorrelId/MsgId to select correct response
  - Recovery processing
    - If the queue manager fails, all clients reconnect at the same time

MQ Client Architectures

- MQ clients offer lightweight, low overhead, low cost and low administration access to MQ services. Clients reduce the requirements for machine resources on the client machine, but there are tradeoffs: Resources on the server are required for the MCAs to handle the client connections - 1 per client connection (MQCONN).
- Application architectures built around thin clients often feature large numbers of connections. MQ has been proven with large configurations of up to 50,000 clients concurrently attached to a single AIX server. However, there are some points to consider to achieve the best performance with thin clients:
  - Large configurations (ie many client attachments) result in a large number of MQ processes:
  - Each client connection requires a channel. Each channel requires a receiver and an agent.
- The number of processes can be reduced by using trusted bindings for the receiver, eliminating the agent processes.
- Since each queue requires control structures in memory, having a ReplyToQ for each client will result in a large number of queues and high memory usage. You can reduce the number of queues, and therefore memory requirements, by sharing a ReplyToQ between some (or all) of the clients, and referencing reply messages using MsgId and/or CorrelId.
- Each API call is transferred (without batching) to the server, where the call is executed and the results returned to the client. The MQMD has to be passed on input and output flow. Similarly the MQGMO/MQPMO.

Asynchronous Put Response



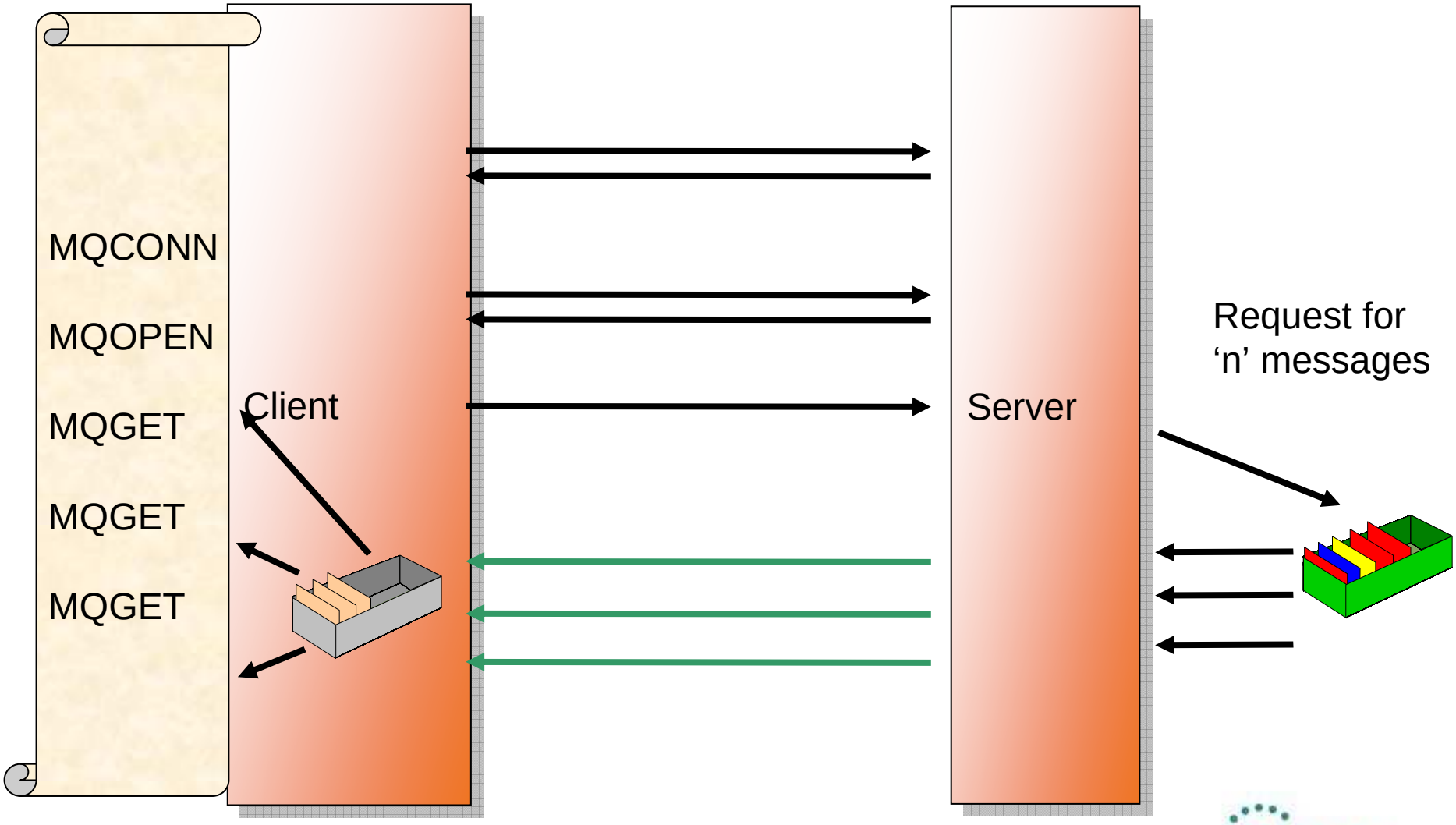
Asynchronous Put Response

- Asynchronous Put (also known as 'Fire and Forget') is a recognition of the fact that a large proportion of the cost of an MQPUT from a client is the line turnaround of the network connection. When using Asynchronous Put the application sends the message to the server but does not wait for a response. Instead it returns immediately to the application. The application is then free to issue further MQI calls as required. The largest speed benefit will be seen where the application issues a number of MQPUT calls and where the network is slow.

- Once the application has competed its put sequence it will issue MQCMIT or MQDISC etc which will flush out any MQPUT calls which have not yet completed.

- Because this mechanism is designed to remove the network delay it currently only has a benefit on client applications.

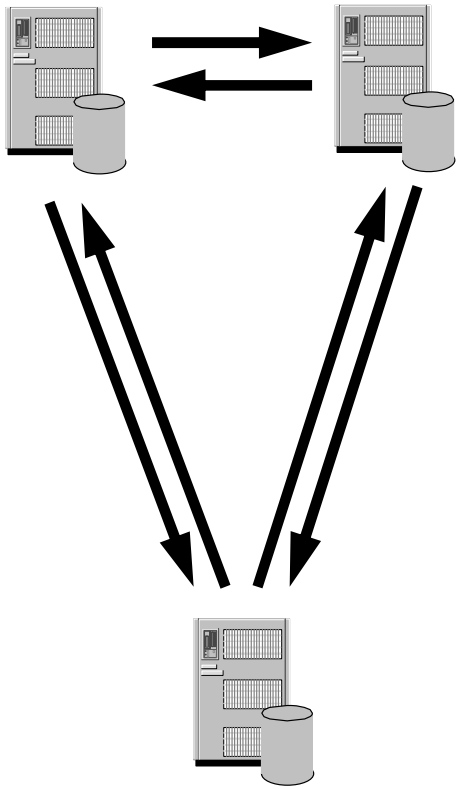
Read-ahead of messages



Read-ahead of messages

- Read Ahead (also known as 'Streaming') is a recognition of the fact that a large proportion of the cost of an MQGET from a client is the line turnaround of the network connection. When using Read Ahead the MQ client code makes a request for more than one message from the server. The server will send as many non-persistent messages matching the criteria (such as MsgId) as it can up to the limit set by the client. The largest speed benefit will be seen where there are a number of similar non-persistent messages to be delivered and where the network is slow.
- Read Ahead is useful for applications which want to get large numbers of non-persistent messages, outside of syncpoint where they are not changing the selection criteria on a regular basis. For example, getting responses from a command server or a query such as a list of airline flights.
- If an application requests read ahead but the messages are not suitable, for example, they are all persistent then only one message will be sent to the client at any one time. Read ahead is effectively turned off until a sequence of non-persistent messages are on the queue again.
- The message buffer is purely an 'in memory' queue of messages. If the application ends or the machine crashes these messages will be lost.
- Because this mechanism is designed to remove the network delay it currently only has a benefit on client applications.

Clustering



Clustering – How Many FRs

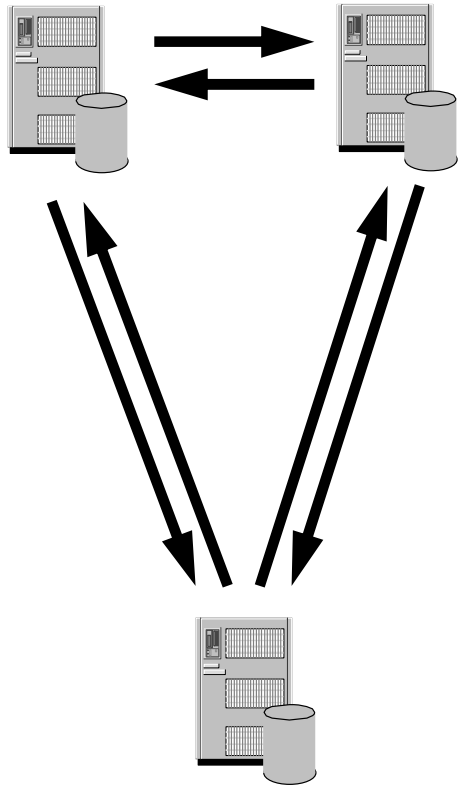
- Whenever an application tries to use a queue, that queue manager will have to register its interest with the full repositories.
  - The more of these ‘subscriptions’ there are in the system, the bigger the overhead when changes occur
  - Minimise unnecessary traffic and FR load by hosting similar applications (those that work with same queues) in the same location
- Already mentioned admin overhead of many overlapping clusters.
  - Also has a cluster performance implication, subscriptions have to be made in every cluster (even if these end up flowing to the same place)
- **Sometimes** an advantage to having ‘parallel’ channels within a cluster…
  - But don’t rush to do that. As well as adding complexity, may result in channels being underutilised which will actually reduce performance
  - Generally keep separate cluster receivers for separate QOS
    - For example separating security domains, see later

Considerations for Full Repositories (FRs)

- More often than not, problems hit by administrators are caused by the interactions between multiple WLM parameters.
- This is another area where a large number of overlapping clusters can cause issues – probably confusion in this case.
- On the other hand, remember that most balancing is per-channel. As well as the points on the main slide, this means that separating applications into separate channels (therefore clusters) will stop messages from one altering balancing for another.

Considerations for Full Repositories (FRs)

- FRs should be highly available
  - Avoid single point of failure - have at least 2
  - **Recommended to have exactly 2 unless you find a very good reason to have more**
  - Put them on highly available machines



- FRs must be fully inter-connected
  - Using manually defined cluster sender channels

- If at least one FR is not available or they are not fully connected
  - Cluster definition changes via FRs will not flow
  - User messages between Partial Repositories over existing channels will flow

Considerations for Full Repositories (FRs)

- Full Repositories must be fully connected with each other using manually defined cluster sender channels.
- You should always have at least 2 Full Repositories in the cluster so that in the event of a failure of a Full Repository, the cluster can still operate. If you only have one Full Repository and it loses its information about the cluster, then manual intervention on all queue managers within the cluster will be required in order to get the cluster working again. If there are two or more Full Repositories, then because information is always published to and subscribed for from 2 Full Repositories, the failed Full Repository can be recovered with the minimum of effort.
- Full Repositories should be held on machines that are reliable and highly available. This said, if no Full Repositories are available in the cluster for a short period of time, this does not affect application messages which are being sent using the clustered queues and channels, however it does mean that the clustered queue managers will not find out about administrative changes in the cluster until the Full Repositories are active again.
- For most clusters, 2 Full Repositories is the best number to have. If this is the case, we know that each Partial Repository manager in the cluster will make its publications and subscriptions to both the Full Repositories.
- It is possible to have more than 2 Full Repositories.

Considerations for Full Repositories (FRs)

- The thing to bear in mind when using more than 2 Full Repositories is that queue managers within the cluster still only publish and subscribe to 2. This means that if the 2 Full Repositories to which a queue manager subscribed for a queue are both off-line, then that queue manager will not find out about administrative changes to the queue, even if there are other Full Repositories available. If the Full Repositories are taken off-line as part of scheduled maintenance, then this can be overcome by altering the Full Repositories to be Partial Repositories before taking them off-line, which will cause the queue managers within the cluster to remake their subscriptions elsewhere.

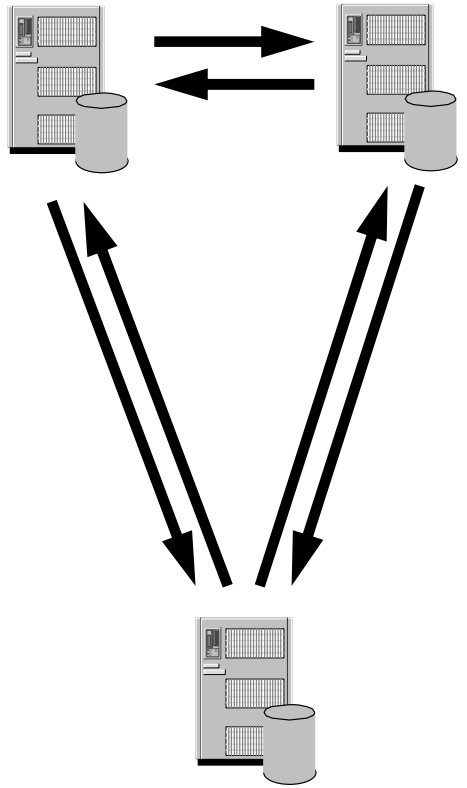
- If you want a Partial Repository to subscribe to a particular Full Repository queue manager, then manually defining a cluster sender channel to that queue manager will make the Partial Repository attempt to use it first, but if that Full Repository is unavailable, it will then use any other Full Repositories that it knows about.
- Once a cluster has been setup, the amount of messages that are sent to the Full Repositories from the Partial Repositories in the cluster is very small. Partial Repositories will re-subscribe for cluster queue and cluster queue manager information every 30 days at which point messages are sent. Other than this, messages are not sent between the Full and Partial Repositories unless a change occurs to a resource within the cluster, in which case the Full Repositories will notify the Partial Repositories that have subscribed for the information on the resource that is changing.

- As this workload is very low, there is usually no problem with hosting the Full Repositories on the server queue managers. This of course is based on the assumption that the server queue managers will be highly available within the cluster.

- This said, it may be that you prefer to keep the application workload separate from the administrative side of the cluster. This is a business decision.

Considerations for Full Repositories (FRs)

- Should applications run on full repositories? (Should they host ‘data’ queues?)
  - Best Practice hat on: No
  - consider the risks (see Notes) and decide on what is appropriate given your environment
- What if I need to take them down for maintenance?
  - Use the fact that you have two!
- What if I need to move them?
  - It will depend on what is changing, see next section



Considerations for Full Repositories (FRs)

- The previous slide gave the ‘standard’ rules and reasons for working with full repository, but here are some tips based on the way people really tend to work with them and some common issues:

- There is no reason applications cannot happily run on a queue manager which is acting as a full repository, and certainly the original design for clustering assumes this will probably be the case. HOWEVER, many people actually prefer to keep FRs dedicated to just maintaining the cluster cache, for various reasons:
  - When any application in the cluster wants to use new features, can upgrade FRs without having to test ALL co-located applications
  - If for some reason you need to apply urgent maintenance to your full repositories they can be restarted or REFRESHed without touching applications
  - As clusters grow and demands on cache maintenance become heavier, there is no risk of this affecting application performance (through storage, CPU demands for example)
  - Full repositories don’t actually need to be hugely powerful – a simple Unix server with a good expectation of availability is sufficient.

- Maintenance:
  - This is precisely the sort of reason you want 2 full repositories. The cluster will continue to function quite happily with one repository, so where possible bring them down and back up one at a time. Even if you experience an outage on the second, running applications should be completely unaffected for a minimum of three days

- Moving full repositories
  - Is a bit trickier than moving a regular queue manager. The migration foils look into this further

Publish / Subscribe

- Location of Publishers
- Location of Subscribers
- Clustered pubsub

NEW: Publish Subscribe Cluster controls

- Publish Subscribe clusters introduced in WMQ Version 7 give a powerful way to combine the ‘topic space’ – pub/sub domain - across multiple queue managers
- However, publish subscribe is a much more dynamic environment than traditional point to point messaging, and this means in large or tightly stretched deployments the additional overhead can be undesirable
- Therefore for existing large point to point clusters the recommendation has been to avoid suddenly introducing clustered Topic objects (instead creating new cluster networks for any pub/sub activity), and many customers have taken this advice on board in their internal processes.
- However, until now there has been no way to enforce this…

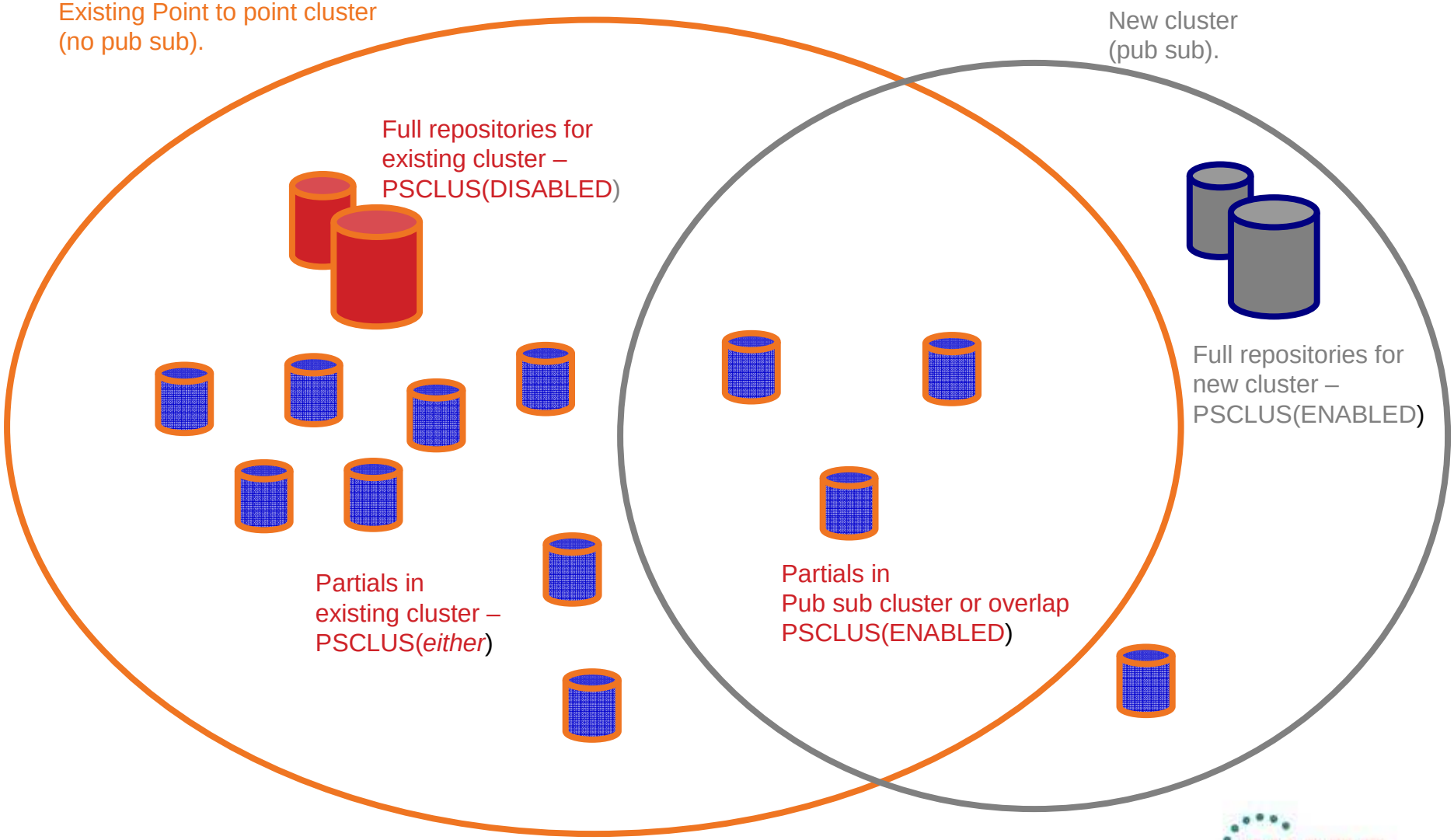
Notes: Pub sub cluster controls

- Conference sessions previously have advised against using topics in ‘large’ clusters. 100 queue managers is a very approximate suggestion for ‘large’, but in reality what problems will be seen varies hugely depending on the usage and load on the system – 5 QMs may be ‘large’, or 500 may run acceptably.
- Things to consider
  - Rate of change of subscriptions – huge factor if lots of new subscriptions constantly trigger new proxy subs.
  - Existing interconnectivity (from p2p traffic). Full repositories probably already connected to everyone for example!
  - Publication rates
  - Number of channels supported / required. Any QM hosting subscriber will need sender channel to every other QM.
- Strong recommendation to create new pub sub infrastructure and scale gradually bearing all these considerations in mind.

Publish Subscribe Cluster controls

- New attribute at queue manager level controls whether or not this QM will participate in pub/sub clustering.
  - PSCLUS (ENABLED/DISABLED)
- Disables the definition of cluster topic objects, and the sending/receiving of proxy subscriptions.
- Cannot be disabled if cluster topics already present (even defined elsewhere in the cluster).
  - This is an up front ‘policy enforcement’ measure, not a quick fix!
- **Ideally** set on every queue manager if no pub sub to be used.
  - **However** configuring at least full repositories gives majority of protection. This also disables the ‘everybody learns about everybody’ aspect of a publish/subscribe cluster.

Publish Subscribe Cluster controls

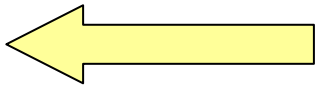


Notes: Pub sub cluster controls

- This slide gives an example of how an administrator could choose to overlay a new pub sub cluster on their existing point to point infrastructure. Key points are:
  - Separate full repositories for existing and new clusters. If desired to keep entirely as a subset of existing cluster, the new FRs _could_ also be in the point to point cluster – but they must be separate queue managers to the existing FRs.
  - PSCLUS disabled on non-pub/sub full repositories
  - PSCLUS enabled on all queue managers participating in pub sub clusters. NOT recommended to belong to a pub sub cluster and disabled, this is an 'error' and will trigger warning messages in logs.
  - PSCLUS can be disabled if desired on non pub sub partial repositories in the non pub sub cluster. Doing so is desirable as it will make errors more obvious more quickly and directly to the responsible administrator. If this is not practical, it is not mandatory.
  - Partial repositories which need to participate in pub sub are added to the pub sub cluster over time, and PSCLUS enabled. Some queue managers may only be participating in the pub sub cluster.

Message Broker

Agenda

- Planning your Message Broker deployment topology 
  - Broker modes
  - Execution Groups vs Additional instances
  - Scaling
  - Applications and Libraries
  - Operational environment
  - HA multi instance
  - Global cache
- Understanding your broker's behaviour
  - Other tools
  - Performance statistics
  - Resource Statistics
  - Activity Log
- Making changes

Why should you care about the topology

- Optimising your broker's performance can lead to real $$ savings… Time is Money!
  - Lower hardware requirements, lower MIPS
  - Maximised throughput… improved response time
  - Ensure your SLAs are met
- WebSphere Message Broker is a sophisticated product
  - There are often several ways to do things… which is the best?
  - Understanding how the broker works allows you to write more efficient message flows and debug problems more quickly
- It's better to design your message flows to be as efficient as possible from day one
  - Solving bad design later on costs much more
  - Working around bad design can make the problem worse!

Broker modes

The operation mode that you use for your broker is determined by the license that you purchase.

The following modes are supported:

- **Trial Edition**
  - All features are enabled, but you can use the product for only 90 days after installation.
- **Express Edition**
  - Limited number of features are enabled for use within a single execution group. Message flows are unlimited.
- **Standard Edition**
  - All features are enabled for use with a single execution group. The number of message flows that you can deploy are unlimited.
- **Advanced mode**
  - All features are enabled and no restrictions or limits are imposed. This mode is the default mode, unless you have the Trial Edition.
- **Remote Adapter Deployment**
  - Only adapter-related features are enabled, and the types of node that you can use, and the number of execution groups that you can create, are limited.

Message Flow Deployment Algorithm

How many Execution Groups should I have?
How many Additional Instances should I add?

Execution Groups

- Results in a new process/address-space
- Increased memory requirement
- Multiple threads including management
- Operational simplicity
- Gives process level separation
- Scales across multiple EGs

Additional Instances

- Results in more processing threads
- Low(er) memory requirement
- Thread level separation
- Can share data between threads
- Scales across multiple EGs

Recommended Usage

- Check resource constraints on system
  - How much memory available?
  - How many CPUs?
- Start low (1 EG, No additional instances)
- Group applications in a single EG
- Assign heavy resource users to their own EG
- Increment EGs and additional instances one at a time
  - Keep checking memory and CPU on machine
- Don't assume configuration will work the same on different machines
  - Different memory and number of CPUs

Ultimately have to balance Resource Manageability Availability

Consider how your message flows will scale

- How many **execution groups**?
  - As processes, execution groups provide isolation
  - Higher memory requirement than threads
  - Rules of thumb:
    - One or two execution groups per application
    - Allocate all instances needed for a message flow over those execution groups
    - Assign heavy resource users (memory) to specialist execution groups

- How many **message flow instances**?
  - Message flow instances provide thread level separation
  - Lower memory requirement than execution groups
  - Potential to use shared variables
  - Rules of thumb:
    - There is no pre-set magic number
    - What message rate do you require?
    - Try scaling up the instances to see the effect; to scale effectively, message flows should be efficient, CPU bound, have minimal I/O and have no affinities or serialisation.

Applications and Libraries

- WMB V8 introduces the concepts of Applications and Libraries as means of developing, deploying and managing your integration solutions:

Application

- means of encapsulating resources to solve a specific connectivity problem
- application can reference one or more libraries

Library

- a logical grouping of related routines and/or data
- libraries help with reuse and ease of resource management
- library can reference one or more libraries

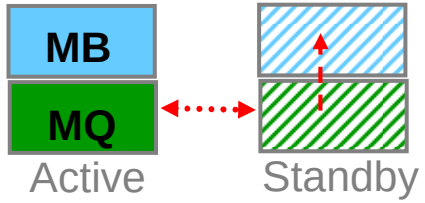
- These concepts span all aspects of the Toolkit and broker runtime, and are designed to make the development and management of WMB solutions easier.

Consider your operational environment

- Network topology
  - Distance between broker and data
- What hardware do I need?
  - Including high availability and disaster recovery requirements
  - IBM can help you!
- Transactional behaviour
  - Early availability of messages vs. backout
  - Transactions necessitate log I/O – can change focus from CPU to I/O
  - Message batching (commit count)
- What are your performance requirements?
  - Now… and in five years…

High Availability with MQ Multi-Instance Brokers

- **MB Exploits MQ Multi-instance queue manager capability**
  - MQ provides basic failover without HA coordinator
  - HACMP, VCS, HA Linux no longer required in many scenarios to restart MQ and MB!
  - MB SAP Input node exploits for state management to give multi-broker and HA SAP support



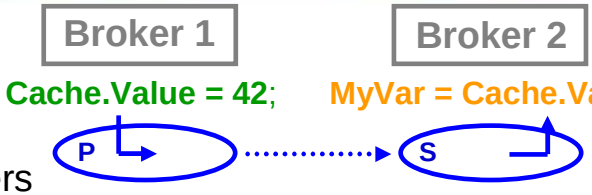
- **Active and Stand-by Queue Manager and Brokers**
  - Start multiple instances of a queue manager on different machines
  - One is “active” instance; other is “standby” instance
  - Shared data is held in networked storage (NAS, NFS, GPFS) but owned by active instance
  - Automatic MQ Client reconnect will attempt to make failures transparent as possible

V53	Unknown	Unknown	Unknown	Stopped	Unknown
V7	2414	V7_2009-04-21_12.3...	Yes	Running	rockall(Active)
V7B	2415	V7B_2009-04-21_12....	Yes	Running	rockall(Active)
V7C	Unknown	Unknown	Unknown	Running as standby	llareggub(Active), rockall(Standby)

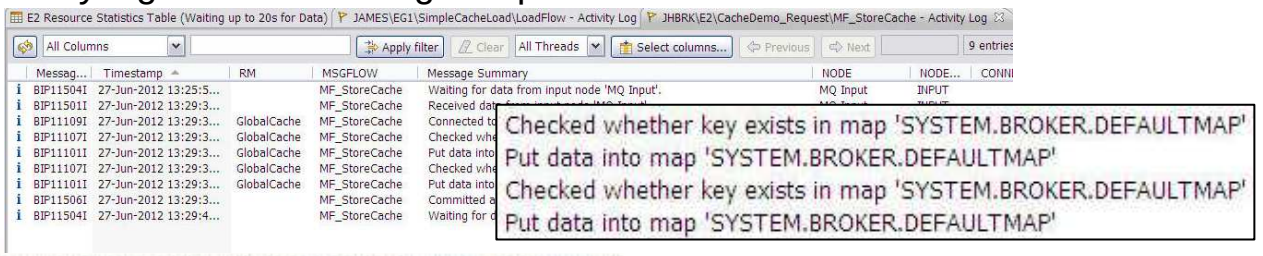
  - **MB Exploitation**

1. Standby MB not running; an MQ service can restart MB once MQ recovery complete
2. Standby MB is running, but not fully initialized until MQ recovery complete

Global Cache

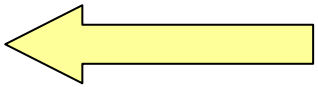


- New Built-in facility to share data between multiple brokers
  - Global cache shared between flows, execution groups & brokers
  - Seamless access to global cache using from all message broker flows and nodes
  - Typical scenarios include multi-broker request-reply and multi-broker aggregation
  - Improve mediation response times and dramatically reduce application load
  - Uses WebSphere Extreme Scale coherent cache technology to replace existing IA91 Support Pac
- Easy Read and Write Access to Global Cache
  - Accessed via global map data type corresponding to MB global cache
  - Read and write access in same way as other data types e.g. **`LocalEnvironment, MyVariable`**
  - Broker has own system cache for inter-broker information sharing
  - Support for external caches through embedding WXS JARs in JavaCompute node
- Simple to Configure and Manage
  - MB Cache completely contained within MB OS processes – no extra moving parts to configure
  - Flow write to global cache is replicated to other cache members using WXS technology
  - MB local cache can be administered via Explorer, command line & management API
- Monitoring and Reporting
  - Full resource manager statistics shows cache interactions and other relevant statistics
  - Activity log shows cache agent operations for write and read



Agenda

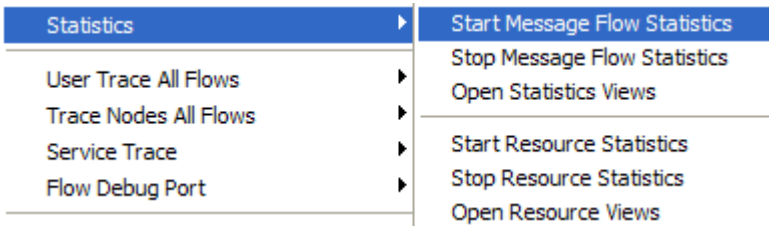
- Planning your Message Broker deployment topology
- Understanding your broker's behaviour
  - Other tools
  - Performance statistics
  - Resource Statistics
  - Activity Log
- Making changes

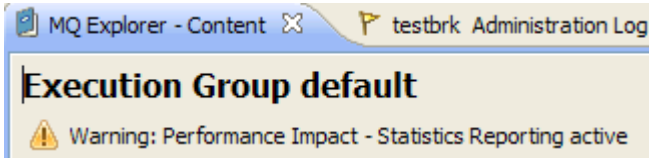


Some tools to understand your broker's behaviour

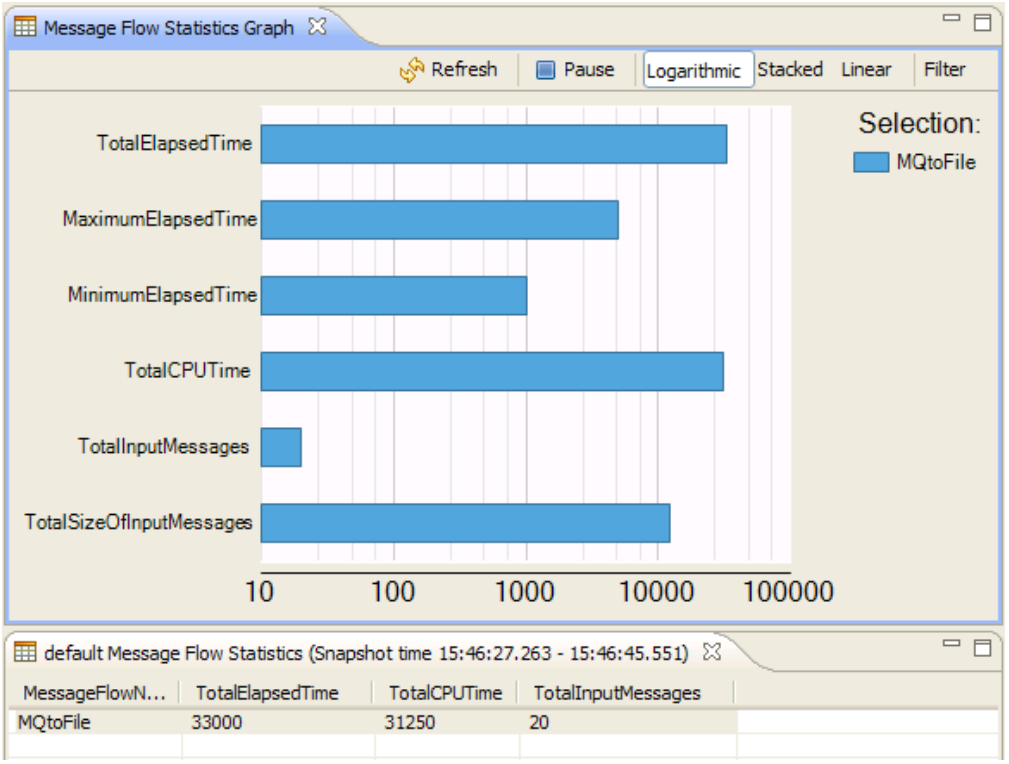
- *PerfHarness* – Drive realistic loads through the broker
  - http://www.alphaworks.ibm.com/tech/perfharness
- OS Tools
  - Run your message flow under load and determine the limiting factor.
  - Is your message flow CPU, memory or I/O bound?
  - e.g. "perfmon" (Windows), "vmstat" or "top" (Linux/UNIX) or "SDSF" (z/OS).
  - This information will help you understand the likely impact of scaling (e.g. additional instances), faster storage and faster networks
- Third Party Tools
  - *RFHUtil* – useful for sending/receiving MQ messages and customising all headers
  - *NetTool* – useful for testing HTTP/SOAP
  - *Filemon* – Windows tool to show which files are in use by which processes
  - *Java Health Center* – to diagnose issues in Java nodes
- MQ Explorer
  - Queue Manager administration
  - Useful to monitor queue depths during tests
- Message Broker Explorer
  - Understand what is running
  - Offload WS-Security processing onto XI50 appliance
  - performance and resource statistics

Broker performance statistics

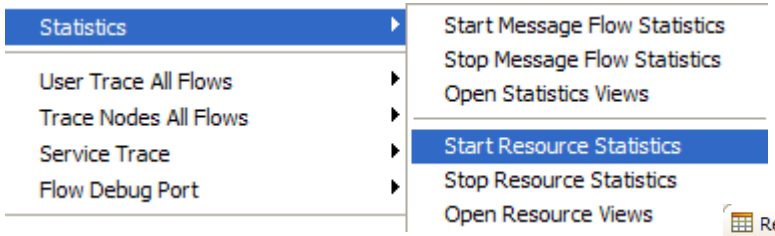


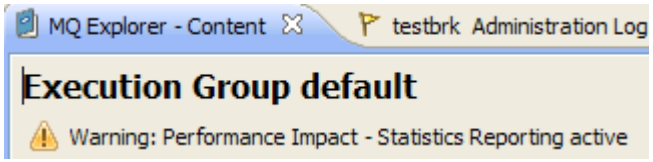


- The Message Broker Explorer enables you to start/stop message flow statistics on the broker, and view the output.
- Warnings are displayed advising there may be a performance impact (typically ~3%)

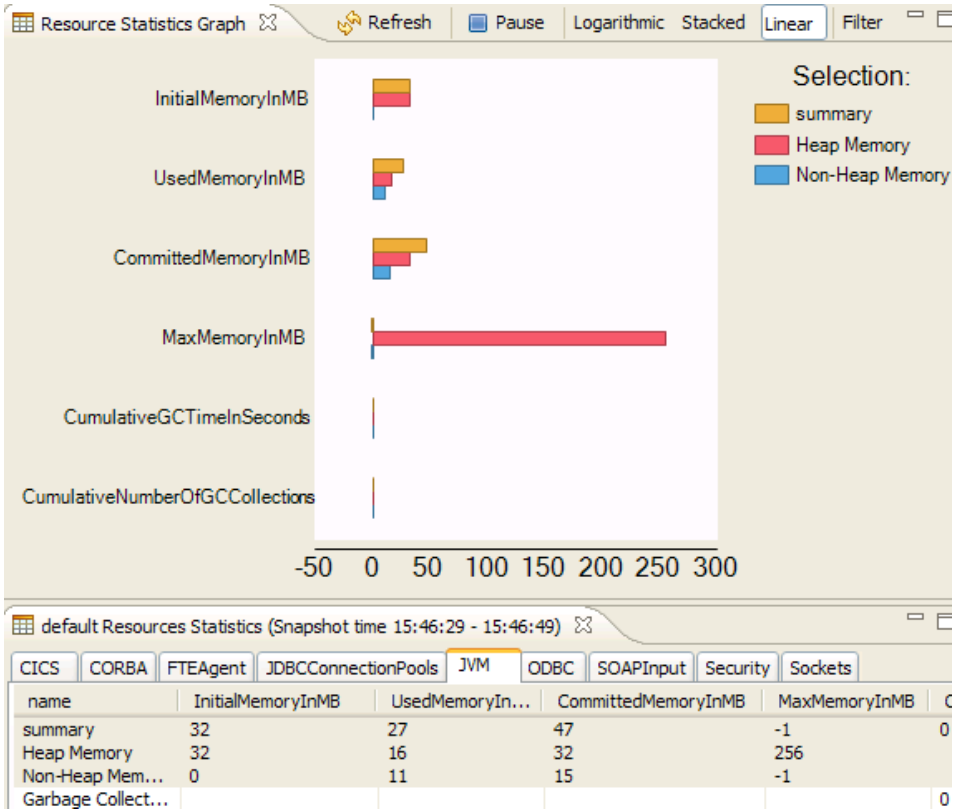


Broker resource statistics





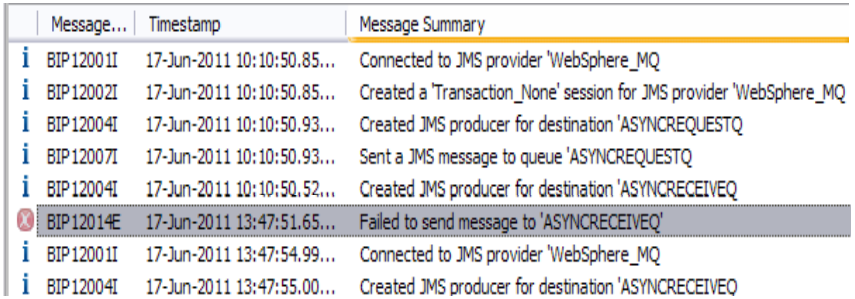
- The Message Broker Explorer enables you to start/stop resource statistics on the broker, and view the output.
- Warnings are displayed advising there may be a performance impact (typically ~1%)



Activity Log

- **New Activity Logging Allows users to understand what a message flow is doing**
  - Complements current extensive product trace by providing end-user oriented trace
  - Can be used by developers, but target is operators and administrators
  - Doesn't require detailed product knowledge to understand behaviour
  - Provides qualitative measure of behaviour

- **End-user oriented with external resource lifecycle**
  - Focus on easily understood actions & resources
  - "GET message queue X", "Update DB table Z"…
  - Complements quantitative resource statistics



	Message...	Timestamp	Message Summary
i	BIP12001I	17-Jun-2011 10:10:50.85...	Connected to JMS provider 'WebSphere_MQ
i	BIP12002I	17-Jun-2011 10:10:50.85...	Created a 'Transaction_None' session for JMS provider 'WebSphere_MQ
i	BIP12004I	17-Jun-2011 10:10:50.93...	Created JMS producer for destination 'ASYNCREQUESTQ
i	BIP12007I	17-Jun-2011 10:10:50.93...	Sent a JMS message to queue 'ASYNCREQUESTQ
i	BIP12004I	17-Jun-2011 10:10:50.52...	Created JMS producer for destination 'ASYNCRECEIVEQ
✗	BIP12014E	17-Jun-2011 13:47:51.65...	Failed to send message to 'ASYNCRECEIVEQ'
i	BIP12001I	17-Jun-2011 13:47:54.99...	Connected to JMS provider 'WebSphere_MQ
i	BIP12004I	17-Jun-2011 13:47:55.00...	Created JMS producer for destination 'ASYNCRECEIVEQ

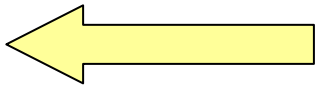
- **Flow & resource logging**
  - User can observe all events for a given flow
    - e.g. "GET MQ message", "Send IDOC to SAP", "Commit transaction"…
  - Users can focus on individual resource manager if required
    - e.g. SAP connectivity lost, SAP IDOC processed
  - Use event filters to create custom activity log
    - e.g. capture all activity on JMS queue REQ1 and C:D node CDN1
  - Progressive implementation as with resource statistics, starting with JMS, C:D and SAP resources

- **Comprehensive Reporting Options**
  - Reporting via MB Explorer, log files and programmable management (CMP API)
  - Extensive filtering & search options, also includes save data to CSV file for later analysis

- **Log Rotation facilities**
  - Rotate resource log file when reaches using size or time interval

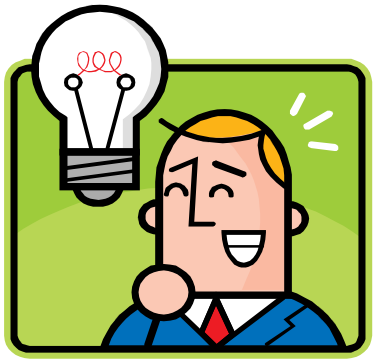
Agenda

- Planning your Message Broker deployment topology
- Understanding your broker's behaviour
- Making changes



Testing and Optimising Message Flows

- Run driver application (e.g. *PerfHarness*) on a machine separate to broker machine
- Test flows in isolation
- Use real data where possible
- Avoid pre-loading queues
- Use a controlled environment – access, code, software, hardware
- Take a baseline before any changes are made
- Make ONE change at a time.
- Where possible retest after each change to measure impact
- Start with a single flow instance/EG and then increase to measure scaling and processing characteristics of your flows



Thank You!

This was session 12622 - The rest of the week ……

	Monday	Tuesday	Wednesday	Thursday	Friday	
08:00					Are you running too many queue managers or brokers?	
09:30		What's New in WebSphere Message Broker			Diagnosing Problems for MQ	CICS and WMQ - The Resurrection of Useful
11:00		Extending IBM WebSphere MQ and WebSphere Message Broker to the Cloud	WMQ - Introduction to Dump Reading and SMF Analysis - Hands-on Lab	BIG Data Sharing with the cloud - WebSphere eXtreme Scale and WebSphere Message Broker integration	Getting the best availability from MQ on z/OS by using Shared Queues	
12:15						
01:30	Introduction to MQ	MQ on z/OS – Vivisection	Migration and maintenance, the necessary evil	The Dark Side of Monitoring MQ - SMF 115 and 116 Record Reading and Interpretation		
03:00	First Steps With WebSphere Message Broker: Application Integration for the Messy	BIG Connectivity with WebSphere MQ and WebSphere Message Broker	WebSphere MQ CHINIT Internals	Using IBM WebSphere Application Server and IBM WebSphere MQ Together		
04:30	WebSphere MQ application design, the good, the bad and the ugly	What's New in the WebSphere MQ Product Family	MQ & DB2 – MQ Verbs in DB2 & Q-Replication	WebSphere MQ Channel Authentication Records		
06:00			Clustering - The Easier Way to Connect Your Queue Managers			