WebSphere Software

WebSphere Integration User Group
Hursley Park, June 2010

WebSphere MQ Clustering Best Practices

Anthony Beardsmore
WebSphere MQ Development

WebSphere Software

Agenda

- Recap
  - What is clustering
  - Why do we use it

- Best Practices
  - General cluster 'hygiene'
  - Performance
  - Avoiding problems before they arise

- Q&A

What this presentation is(n't) about

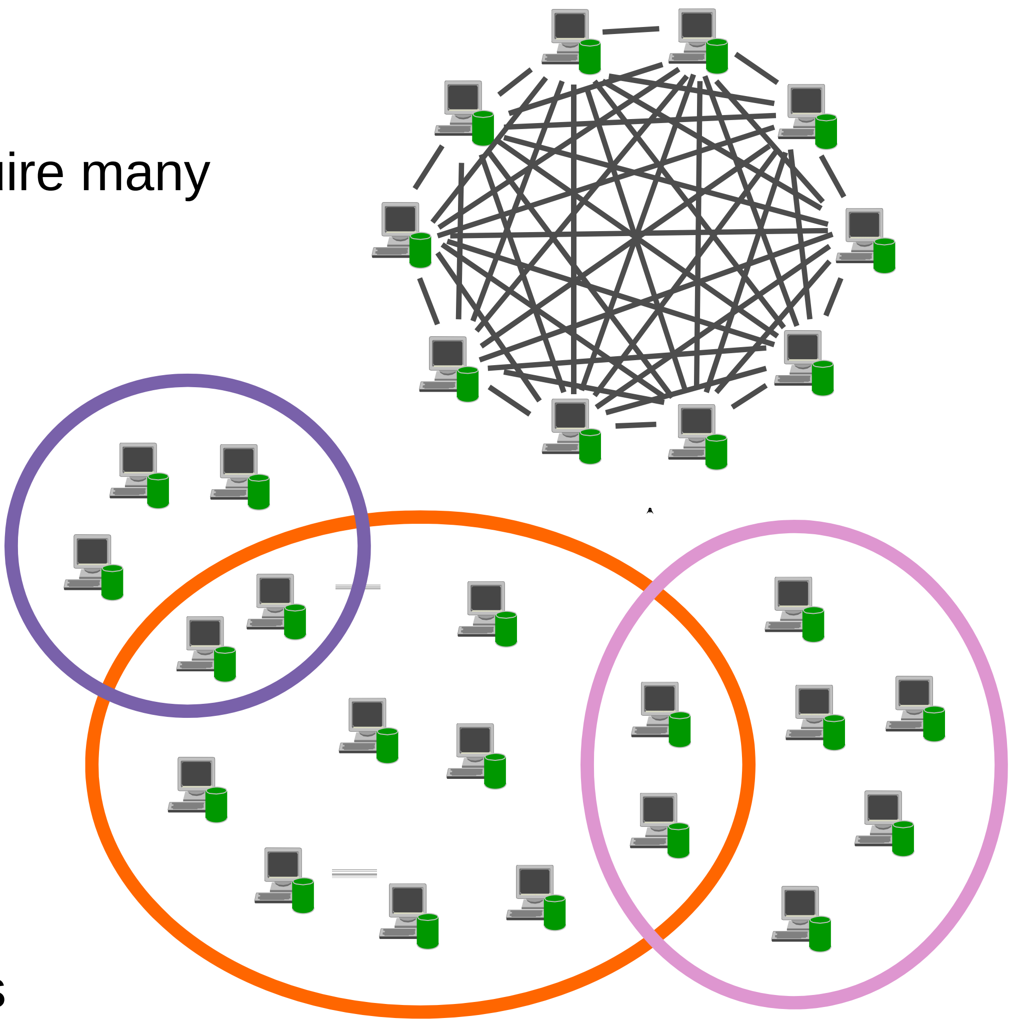
- Assumes a basic understanding of the nuts and bolts of WMQ clusters
  - (but don't be afraid to ask questions)

- Trying to share some common approaches to common problems
  - Not laying down rules, and not one size fits all

- If you have further best practices you'd like to share, feel free! Also, come along to birds of a feather sessions etc.

The purpose of clustering

- Simplified administration
  - Large WMQ networks require many object definitions
    - Channels
    - Transmit queues
    - Remote queues
- Workload balancing
  - Spread the load
  - Route around failures
- Flexible connectivity
  - Overlapping clusters
  - Gateway Queue managers
- Pub/sub Clusters



How can we process more messages?

N O T E S

- It would be nice if we could place all the queues in one place. We could then add processing capacity around this single Queue manager as required and start multiple servers on each of the processors. We would incrementally add processing capacity to satisfy increased demand. We could manage the system as a single entity. A client application would consider itself to be talking to a single Queue manager entity.
- Even though this is highly desirable, in practice it is almost impossible to achieve. Single machines cannot just have extra processors added indefinitely. Invalidation of processor caches becomes a limiting factor. Most systems do not have an architecture that allows data to be efficiently shared between an arbitrary number of processors. Very soon, locking becomes an issue that inhibits scalability of the number of processors on a single machine. These systems are known as "tightly coupled" because operations on one processor may have a large effect on other processors in the machine cluster.
- By contrast, "loosely coupled" clusters (e.g. the Internet) have processors that are more or less independent of each other. Data transferred to one processor is owned by it and is not affected by other processors. Such systems do not suffer from processor locking issues. In a cluster solution, there are multiple consumers of queues (client queue managers) and multiple providers of queues (server queue managers). In this model, for example, the black queue is available on multiple servers. Some clients use the black queue on both servers, other clients use the black queue on just one server.
- A cluster is a loosely coupled system. Messages flow from clients to servers and are processed and responses messages sent back to the client. Servers are selected by the client and are independent of each other. It is a good representation of how, in an organization, some servers provide many services, and how clients use services provided by multiple servers.
- The objective of WebSphere MQ clustering is to make this system as easy to administer and scale as the Single Queue Manager solution.

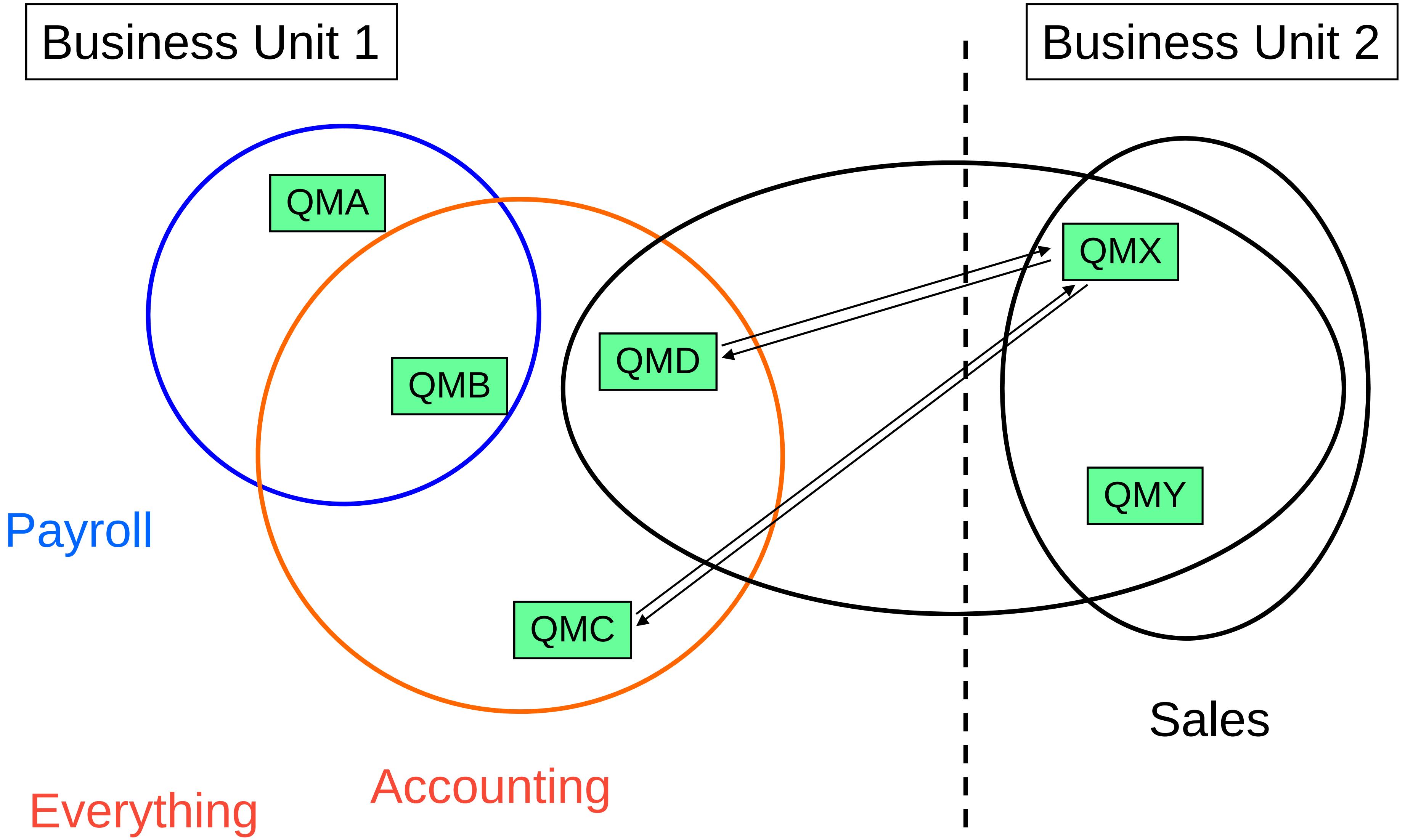
Some general good ideas

- Use the manuals / InfoCenters
  - In particular, when making administrative changes such as adding and removing queue managers, make sure you've understood and followed the process described
- Less is more
  - Clusters make doing a lot, a lot easier. Try and resist the temptation to overcomplicate them therefore!
- Queues work best when they're empty, and channels work best when they're busy.
  - Bear that in mind when designing your topologies

Areas we can look at

- Conventions
  - Who owns what, naming
- Topologies
  - Gateways, improving Performance, WLM tips
- Full repositories
  - Where and what
- Migration (platform, host)
  - Moving servers and changing roles
- Migration (release, fixpack)
  - Scheduling and preparing for maintenance
- Staying available
  - Working with HA, preparing for DR
- Security
  - Transmit queues, Aliases, exits
- Monitoring

Conventions – Coping with Sprawl



Coping with Sprawl – some solutions

- Don’t have too many cooks
  - Treat a single cluster as a single point of administration
  - Have well defined policies for the gateway queue managers

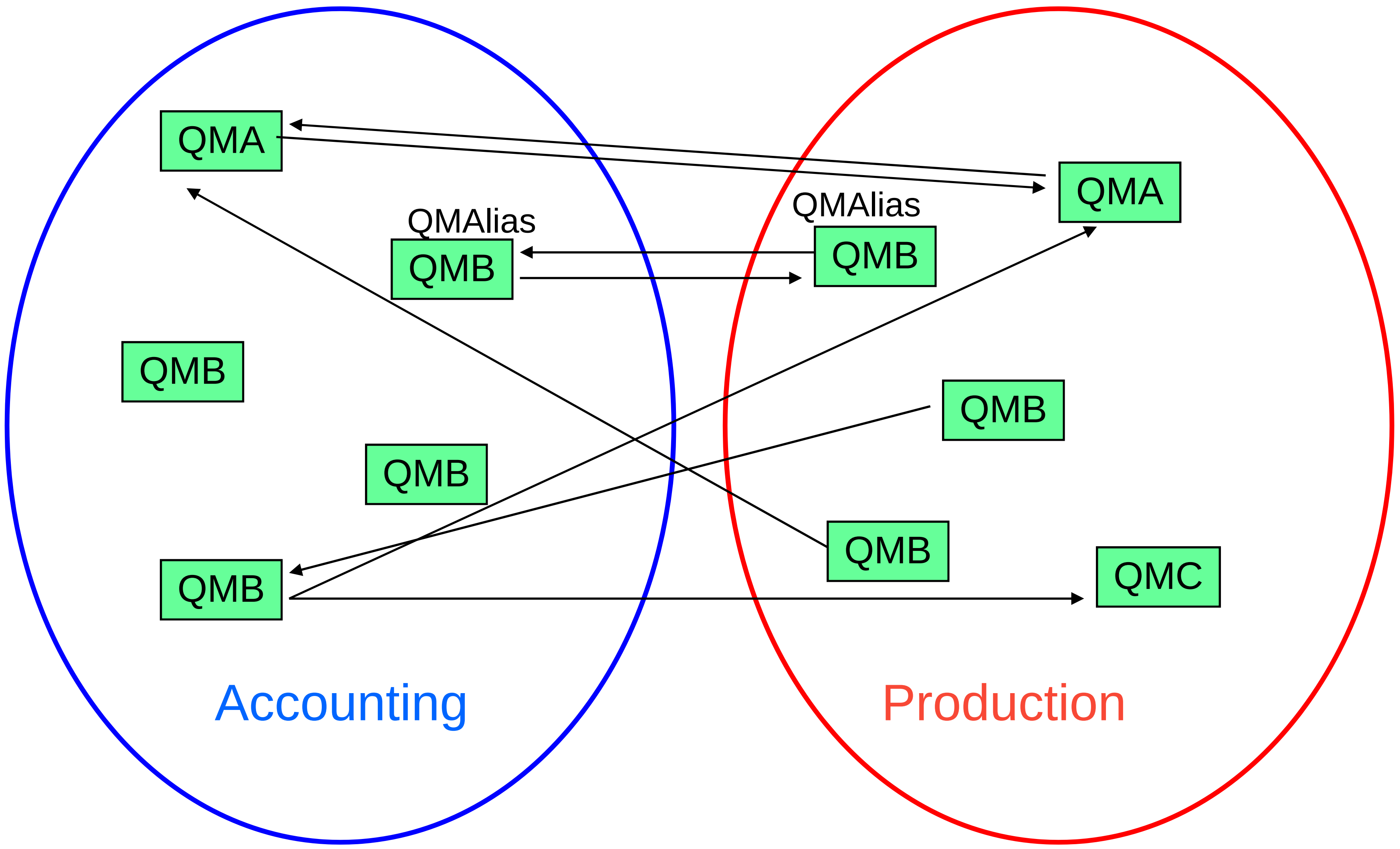
- Treat all overlapping clusters as a single Namespace
  - Channel names, and therefore queue manager names, should be unique throughout.

Notes – Coping with Sprawl

N
O
T
E
S

- KISS (Keep it simple, sysadmin)
- Clusters work best seen as a self contained administrable unit
- Sometimes cooperation is essential / unavoidable (for example between organisations)
- Clear understanding of who owns what, and rules/conventions will help things run smoothly.

Gateways

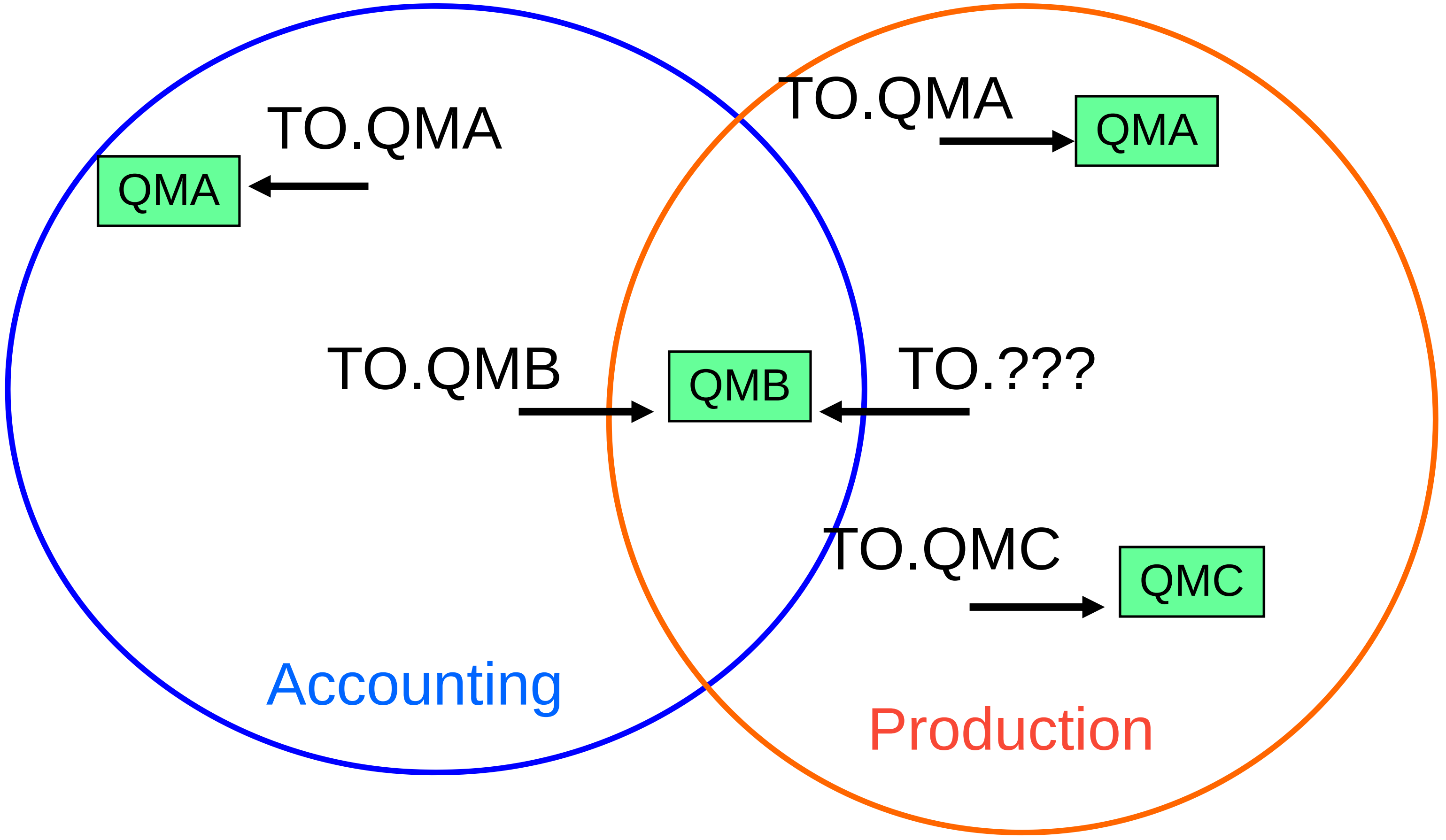

QMA
QMAlias
QMB
QMAlias
QMB
QMA
QMB
QMB
QMB
QMB
QMB
QMB
QMC
Accounting
Production

Gateways

N O T E S

- Help to isolate problems – where clusters are genuinely separated
  - Refer to rule one – don't separate them unless there's a good reason
- Where both clusters managed by same team, may be easier to have a single gateway queue manager which exists in both clusters
  - Use clustered QRemotes to forward messages
- Can have multiple paths between clusters if really needed
  - Can workload balance across QM aliases to choose different routes

Conventions – Naming (The problem)



Conventions: Naming – the Solution

- CLUSNAME.QMGRNAME

- Note that that means only one cluster per channel…
  - Useful side effect!
  - One cluster per channel means if need to separate out qualities of service by cluster later, haven't created a problem that could have been avoided.

Notes – Naming

N
O
T
E
S

- With the previous ownership / scope considerations in mind, naming can help significantly to reduce confusion
- Channels are probably the most significant (because XXX.TO.YYY is such an ingrained idea)
  - No hard rules, but CLUSTER.TO.XX or CLUSTER.X are good contenders
  - Cluster.Qmgr is probably best to save precious chars!
- Other objects can only benefit from sensible house rules
  - LOB.PROJECT.QNAME
  - LOB.CLUSTER.ALIAS.NAME

Performance

- 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 (e.g. separating security domains, see later)

Workload Management

- The workload balancing algorithm gives you a huge set of tools
  - Resist the urge to use all of them
  - Quickly get very complicated interactions with more than a few in play

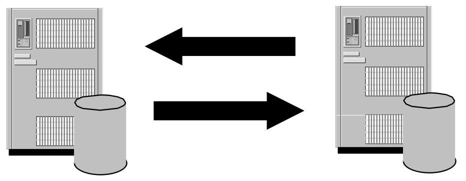
- May not be immediately obvious how important channels are to the selection process
  - Multiple routes to a destination = multiple destinations (from WLM round robin point of view)
  - Stopped local receiver = local destination less favoured

Considerations for Full Repositories (FRs)

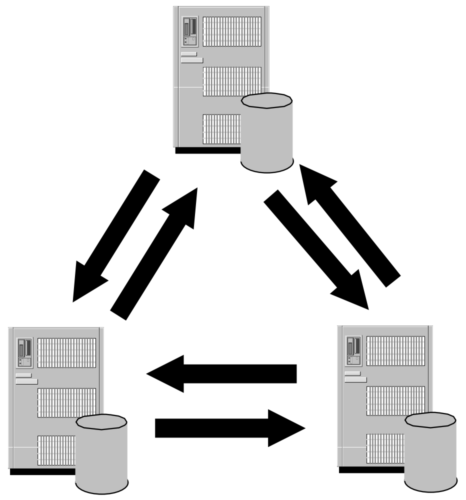
N O T E S

- 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)

NOTES

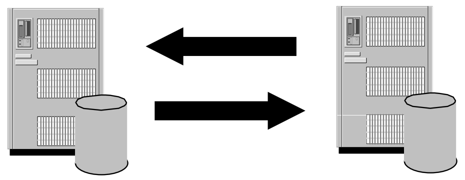
- Full Repositories must be fully connected with each other using manually defined cluster sender channels.
- You should always have at least 2 Full Repositories in the cluster so that in the event of a failure of a Full Repository, the cluster can still operate. If you only have one Full Repository 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)

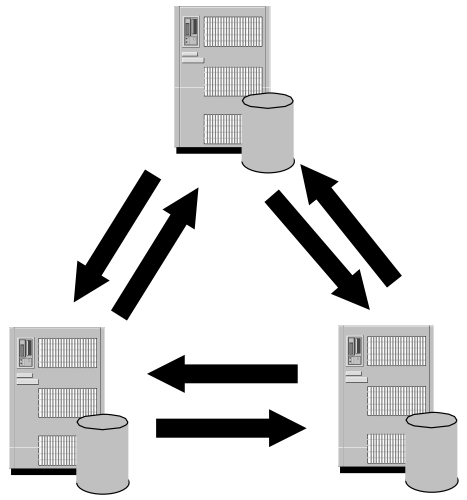
N O T E S

- 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



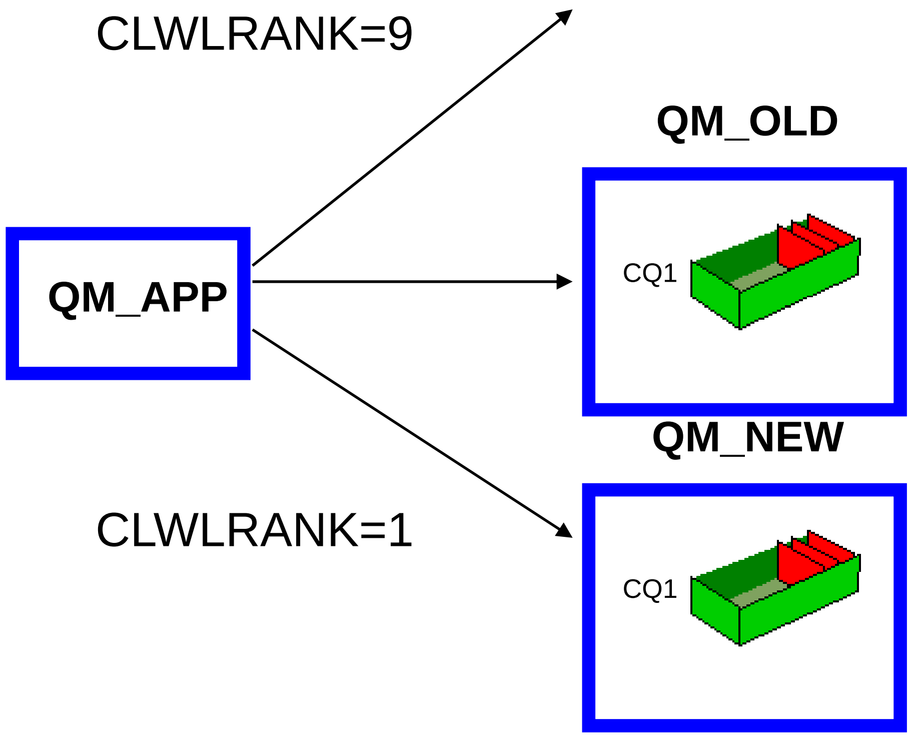
Considerations for Full Repositories (FRs)

N O T E S

- 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

Migration Type 1 (Moving things around in a cluster) Applications and their Queues

- Where you have the option, work with Clustering, not against it!
  - Use multiple destinations with WLM to ‘roll’ applications from the old choice to the new
  - CLWLPRTY or CLWLRANK on Queues or Channels are ideal
  - This gives advantage of easy roll back if needed



Moving Queues and Applications

N O T E S

- CLWLRANK will ignore channel status. Use this when you don’t want the secondary option (the destination not yet in use or no longer in use) to be taken unless other destination removed completely or disabled
- CLWLPRTY has similar behaviour, but will fall back to the secondary destination if the channel is unavailable

Migration Type 1 (Moving things around in a cluster) Entire Queue Managers (partial repositories)

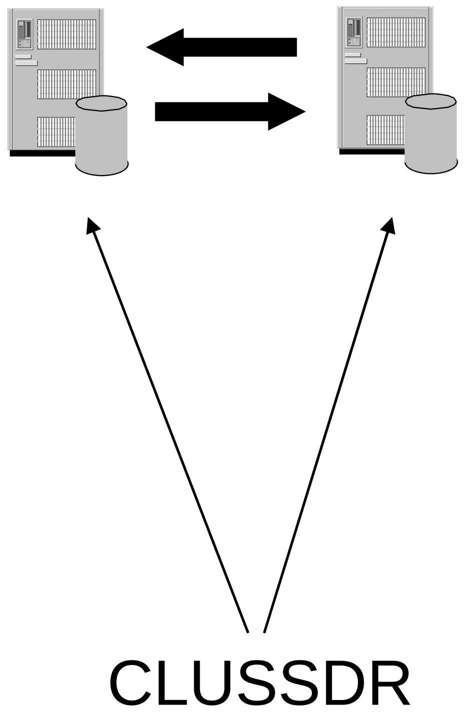
- If actually staying on same host, but IP address changing, this is quite simple.
  - DNS is the way forward
  - Suspend queue manager can help avoid traffic build up
  - As long as can reconnect to full repositories, channel autodefinitions will sort themselves out

- If want to move a queue manager wholesale to a new host, you **can** copy the entire system and restore as from a backup. Would only suggest considering this if no other options
  - Treat as a restore
  - Will need REFRESH CLUSTER

- **Never 'pretend' that two different installations are the same queue manager (by trying to give a new installation the same QMGR name, IP address etc)**
  - This is one of the most common mistakes in working with clusters. The cache knows about QMID and state may end up corrupted
  - If accidentally end up with this scenario, RESET QMGR CLUSTER is your friend

Migration Type 1 (Moving things around in a cluster) Entire Queue Managers (Full repositories)

- As before, DNS used correctly can help simplify things
  - Avoid more than one manually defined sender from each partial, and make them to an FR that is unlikely to change CONNAME
- When you do have to move, give things plenty of time to 'settle' afterwards before making further changes
  - Make sure any manual senders are quickly moved to a valid full repository
- Remember that all full repositories must have senders manually defined to all others (if more than 2)
- **See Previous slide's warning regarding 'pretend' moves, only more so!**



Moving things around in a cluster – Summary

- Best practice is nearly always to use ‘swing’ hardware
- Clustering is good at having multiple options concurrently available. So why get rid of the working setup before the replacement is up and running?
- When creating the replacement, be sure to use a different name!

Moving applications and Queue Managers

N
O
T
E
S

- ‘Time to settle’. A good hint that updates have been processed is no messages waiting on the SYSTEM.CLUSTER.COMMAND.QUEUE on any repositories (full or partial)

Migration Type 2 (Upgrades and installations)

- Avoid the big bang scenario
  - Clusters are designed to be very comfortable with multiple versions of queue manager coexisting
- Have a fall back plan
  - On z/OS, have you applied backwards migration PTFs
  - Have you taken back-ups?
  - Avoid using new function immediately
- Full repositories first
  - Although they can forward information they do not understand, they cannot persist it

Migration Part 2 (Upgrades and installations)

N
O
T
E
S

- http://www.ibm.com/developerworks/websphere/library/techarticles/0910_beardsmore/0910_beardsmore.html
- If taking a queue manager down temporarily (e.g. for this kind of maintenance) remember to use ‘SUSPEND’ first.

Availability

- Clustering isn't an HA solution
  - It can keep things up and running for you though used appropriately
- Loss of a destination queue means any messages there stranded – possibly gone for ever in worst case
  - One of the reasons avoiding ordering requirement at application level preferable
- 2 FRs means loss of one not critical to smooth running of cluster
  - 60 day grace period for existing knowledge if both down

(Image from Wikimedia, © Henrik Thorburn, Licensed under Creative Commons Attribution 3.0)

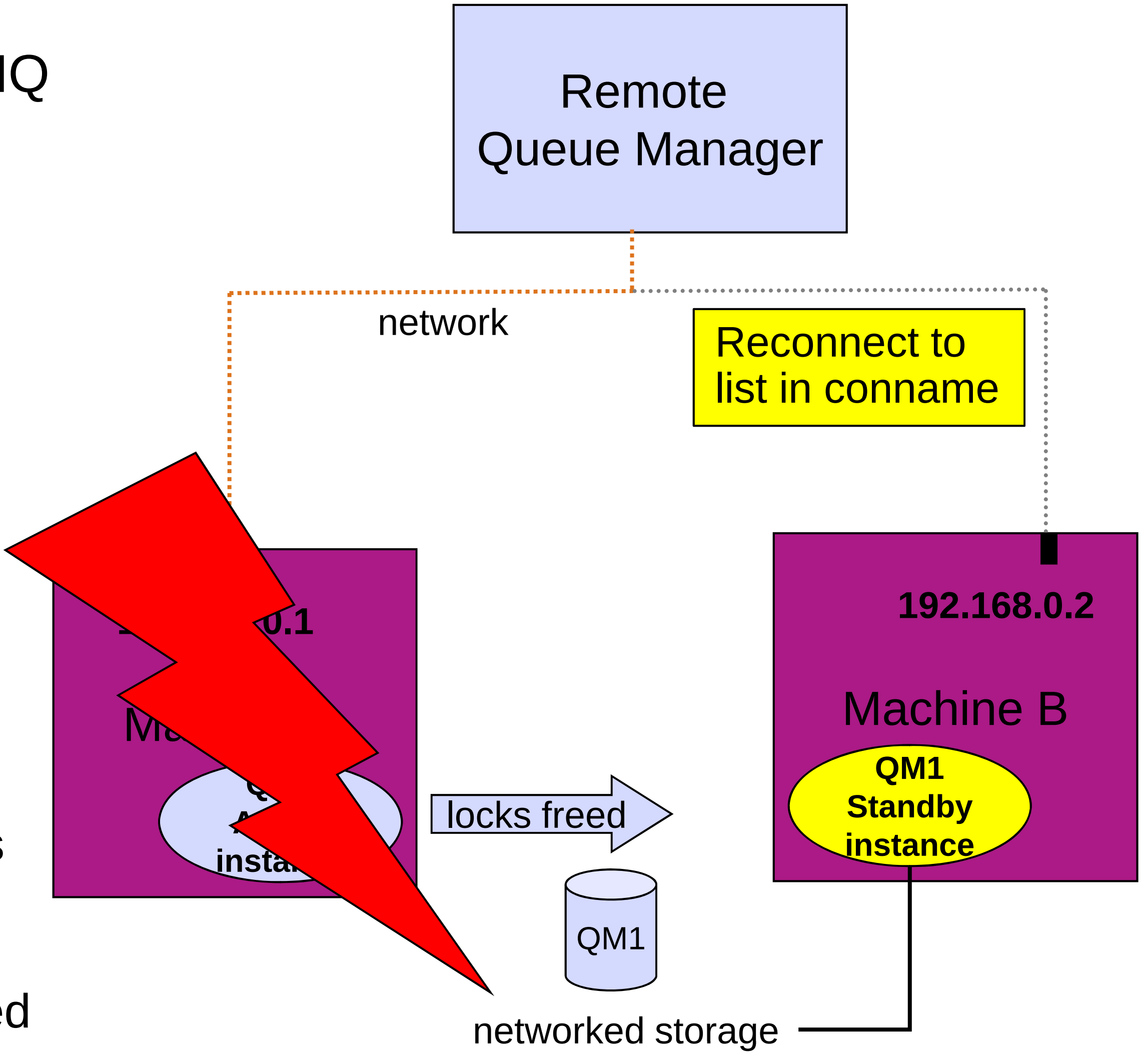
Availability

N
O
T
E
S

- HA is having 2, DR is having them a long way apart
  - More seriously, HA is keeping things running, DR is recovering when HA has failed.
- WMQ, and particularly clustering, are good at getting you back up and running (dealing with new messages) rapidly when there is a failure. Having multiple destinations in a cluster (prioritized if preferred) gives you this almost for free!
- Software HA is the best built in offering for keeping your existing messages available

Availability – Multi Instance Queue Managers

- Software HA – built into MQ
- Hot / Warm
- File system lock based
  - NFS4 or similar
- Relies on highly available NAS
- Cannot help with external dependencies (e.g. DB)
  - Not full replacement for HACMP etc.
- Fully compatible with clustering
  - Including full repositories
  - **Except**… back level queue managers will not respect comma seperated connames



Availability – Multi Instance Queue Managers

N
O
T
E
S

- Given that as a basic rule we think it’s a good idea to have highly available Full Repositories, should we use Software HA / Multi instance to achieve that
- Yes, this is a perfectly workable solution, and may be a good idea.
- Points to remember:
  - Standby repository will only kick in if hot QM loses connection to storage (filesystem/SAN).
  - Network problems on the ‘out-facing’ side might still mean partials cannot reach the full repository
  - Result: Don’t think this means only need one FR!

Availability – Disaster Recovery

- DR is hard to do well, WMQ can only help. The only ‘true’ disaster recovery option in WMQ (excluding underlying replication technologies) is restoration from a backup

- Be VERY careful testing disaster recovery scenario
  - Easy to accidentally join the real cluster and start stealing messages!
  - Ideally keep network separated, but can help by:
    - Not giving backup ‘live’ security certs
    - Not starting chinit address space (z/OS)
    - Not allowing channel initiators to start (distributed)

- Backup will be out of sync with the cluster
  - REFRESH CLUSTER() to resolve updates, get sequence numbers in sync.

- (The hardest one) applications need to deal with replay or loss of data.
  - Decide whether clearing queues down to a known state, or enough information elsewhere to manage replays

Refresh Cluster and the History Queue

- REFRESH CLUSTER considered harmful?
  - Sledgehammer approach
  - Review processes first and see if step missed

- SYSTEM.CLUSTER.HISTORY.QUEUE
  - Version 7.0.1
  - Snapshot captured at refresh time
  - Servicability enhancement

Refresh Cluster and the History queue

N
O
T
E
S

- Refresh cluster is a very powerful tool but can sometimes do more harm than good.
  - By forcing a queue manager to forget all the ‘state’ it knew about, it can flush bad changes throughout your system before they are caught.
- In the past, this has also caused problems for IBM service in trying to help diagnose PMRs (where the first corrective action has been to attempt a REFRESH)
- In version 7, this is addressed by keeping a snapshot of previous cluster state if the S.C.H.Q is defined.
  - This queue will be defined by default on distributed queue managers on startup. On z/OS migration a conscious decision is needed. Messages will expire after 3 months (general cluster object grace period).

Security - Overview

- Use SSL (TLS) or security exits as a baseline
  - Without that, all bets are off (see next slide)
    - Unwanted queue managers joining your cluster
    - Man in the middle
    - Manage certificates appropriately – use tools to help

- Use Queue Aliases!
  - put to local queue alias QA.CQ.DUBLIN (targeting remote cluster queue CQ.DUBLIN)...
    - **setmqaut -m CORK -n QA.CQ.DUBLIN -t queue -p myuser +put**
  - NOTE: No access required to CQ.DUBLIN or the cluster transmit queue
  - Good practice anyway for reasons of flexibility

- z/OS provides checking at object name level (RACF etc.)

Channel security with SSL

- External CAs, Internal CAs or self-signed?
- Alter existing channels or define new channels?
  - NETPRTY
  - Must delete non-SSL channels
- Set SSLCAUTH to REQUIRED
  - Optional still checks received certificate
- SSLPEER
  - “CN=QM*, OU=MIDDLEWARE, O=IBM, C=UK”

- Basic process
  - Setup key repositories and certificates
  - Ensure full repositories and cluster channels are healthy
  - Recommended order of DEFINE/ALTER CHANNEL commands
    - Cluster receivers on all full repositories
    - Cluster senders on all full repositories
    - Cluster receivers on all partial repositories
    - Cluster senders on all partial repositories
  - Channels must be restarted to pick up changes
    - REFRESH SECURITY TYPE(SSL) prior to channel restart.

‘Local’ Security

- Incoming messages –
  - MCAUSER on the channel will help a lot (consider it an absolute minimum, as best practice.)
    - Needs access to the SYSTEM.CLUSTER.COMMAND.QUEUE
  - Can use separate cluster (on separate receiver chl) per class of service/application domain, to restrict access to a subset of queues.
- Certificate name filtering can be a powerful tool (z/OS)
- Some ‘off the shelf’ security exits can provide other useful features – for example BlockIP (BlockIP2)

Incoming message Security

N O T E S

- Without setting an MCAUSER on your cluster channels, you open the queue manager up to the possibility that an application connecting to any other queue manager in the cluster could administer the local Qmgr.

- ALWAYS use MCAUser and channel security (e.g. SSL) as an absolute minimum

‘Local’ Security – Part 2

- If want to avoid aliasing everything, use PUTAUT(CTX) on the receiver channel
  - IDs in message will be used to put messages incoming off the cluster channel
  - IDs remote repositories running under must have access to SYSTEM.CLUSTER.COMMAND.QUEUE
  - Remote queue manager administrators must be trusted to be running a tight ship (no spoofed user IDs)
  - Issues around platform/user name compatibility
- NEVER give +setall to cluster transmit queue (or any other for that matter)

Monitoring

- Remember that all messages going off-queue manager will pass through the SYSTEM.CLUSTER.TRANSMIT.QUEUE
  - Monitor depth appropriately
  - DIS CHSTATUS(*) WHERE(XQMSGSA GT 1)

- SYSTEM.CLUSTER.COMMAND.QUEUE should tend to depth 0
  - Lots of messages indicates churn in the cluster state
  - If making big changes, allow to settle in between (e.g. moving respository – allow to reach 0 before moving second)

Monitoring

- Clustering can't see everything
  - Monitor your other components
  - If you load balance to 2 broker queues and one broker is offline, clustering will happily send messages to a queue that is not being read
  - Can use existing tools or hand crafted code to help - for a crude example of how, see article

http://www-03.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP101552

- All other good monitoring practices apply – for example read the logs
  - Common issue: ignoring 'No update received' messages – can indicate a real problem underlying your cluster which MQ is currently doing its best to work around…

Monitoring

N O T E S

- Although generally administrators should not be too concerned about the state of SYSTEM.XXX queues it can sometimes be useful to monitor the depths of the SYSTEM.CLUSTER queues as described

- SYSTEM.CLUSTER.REPOSITORY.QUEUE should not cause alarm however, it is usual for some number of messages to remain on this queue (containing state information) at any given time.

- SYSTEM.INTER.QMGR.PUBS build up may indicate problems in the publish/subscribe engine (incoming publications from the cluster or hierarchy)

Publish Subscribe

- Clustered Topics make extending your pub sub domain between queue managers easier than it's ever been, but…
- Start small
- Don't put the root node (/) into a cluster
  - Make global topics obvious (e.g. /global or /cluster)
  - Unless migrating…
- Be careful mixing traditional clusters
  - Large clusters and cluster topic objects leads to large number of channels
  - Admin controls on topic objects

Copyright and Trademarks