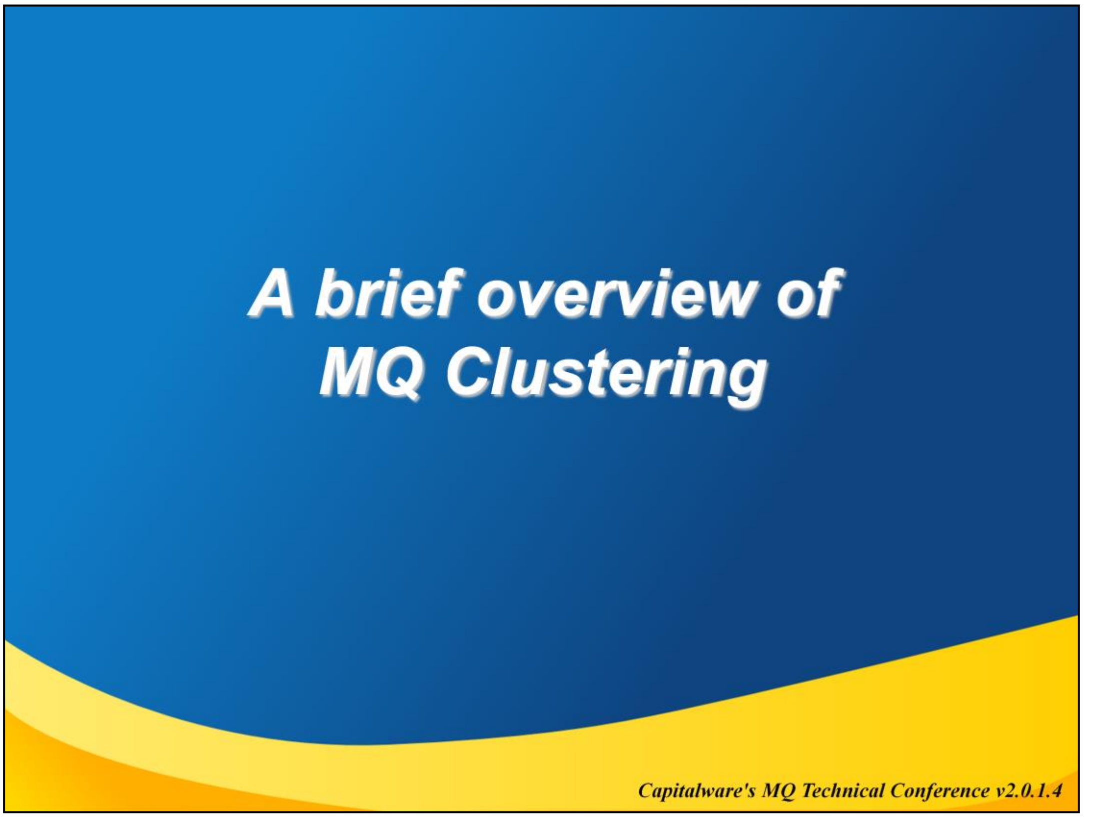
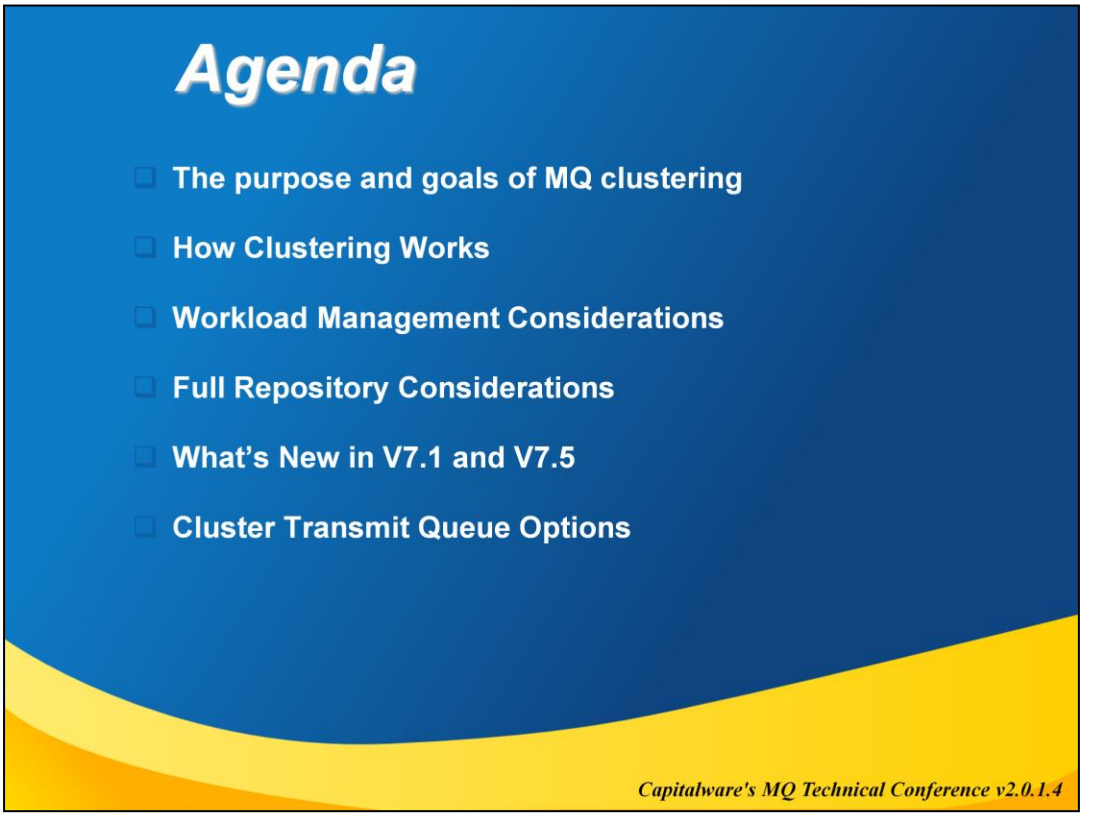
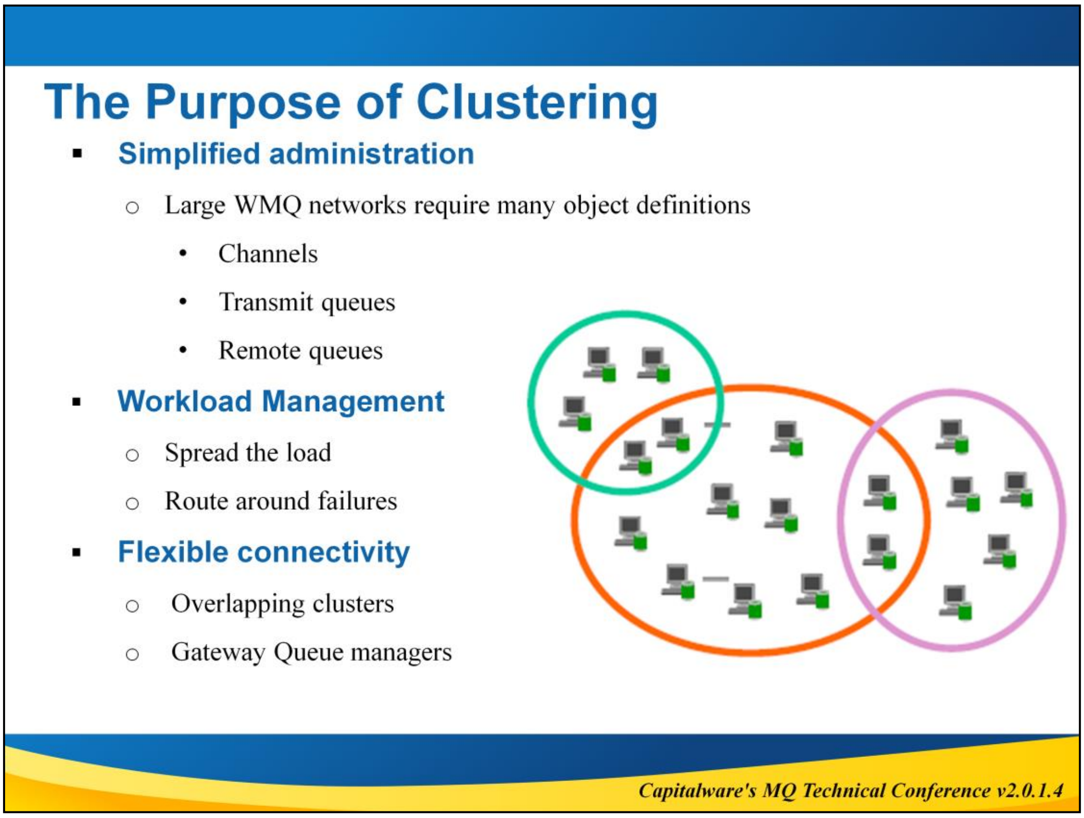

A brief overview of
MQ Clustering
Capitalware's MQ Technical Conference v2.0.1.4


Agenda
The purpose and goals of MQ clustering
How Clustering Works
Workload Management Considerations
Full Repository Considerations
What's New in V7.1 and V7.5
Cluster Transmit Queue Options
Capitalware's MQ Technical Conference v2.0.1.4



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.

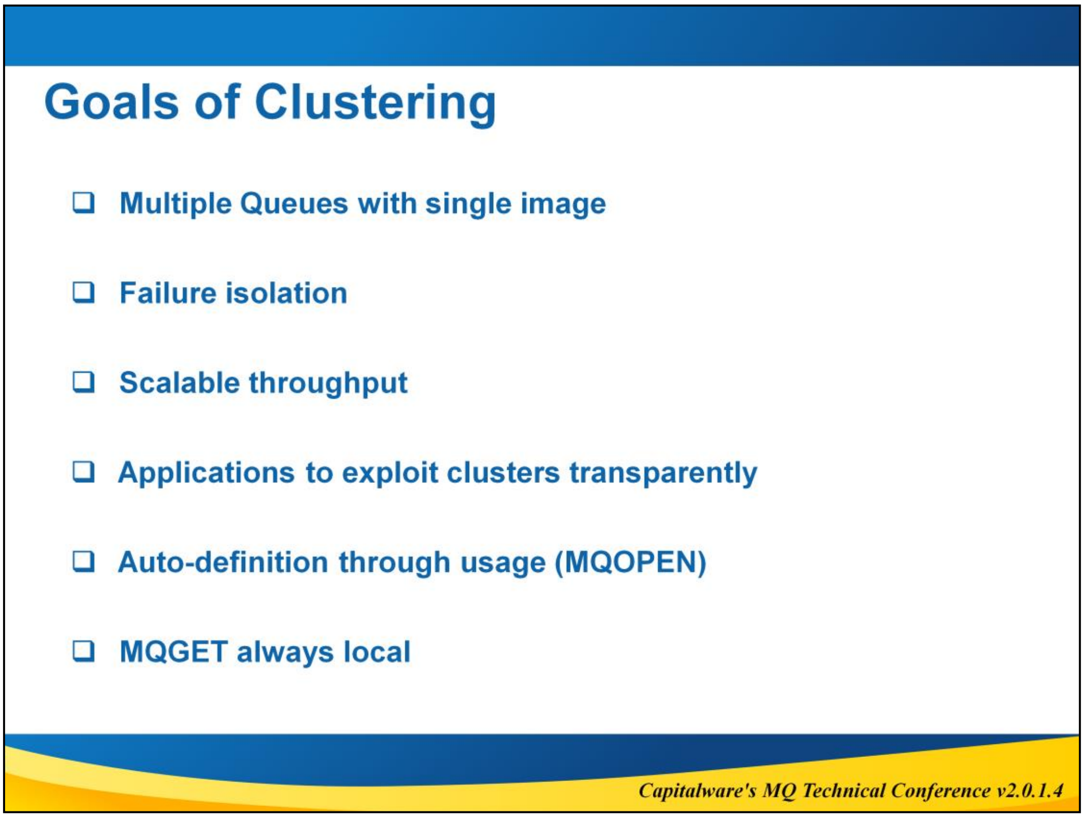
Most systems do not have an architecture that allows data to be efficiently shared between an arbitrary number of processors. Very soon, locking becomes an issue that inhibits scalability of the number of processors on a single machine. These systems are known as "tightly coupled" because operations 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.

What do we mean by Clustering?



Consider a client using the queue that is available in the cluster on three server queue managers. A message is MQPUT by the client and is delivered to *one* of the servers. It is processed there and a response message sent to a ReplyToQueue on the client queue manager.

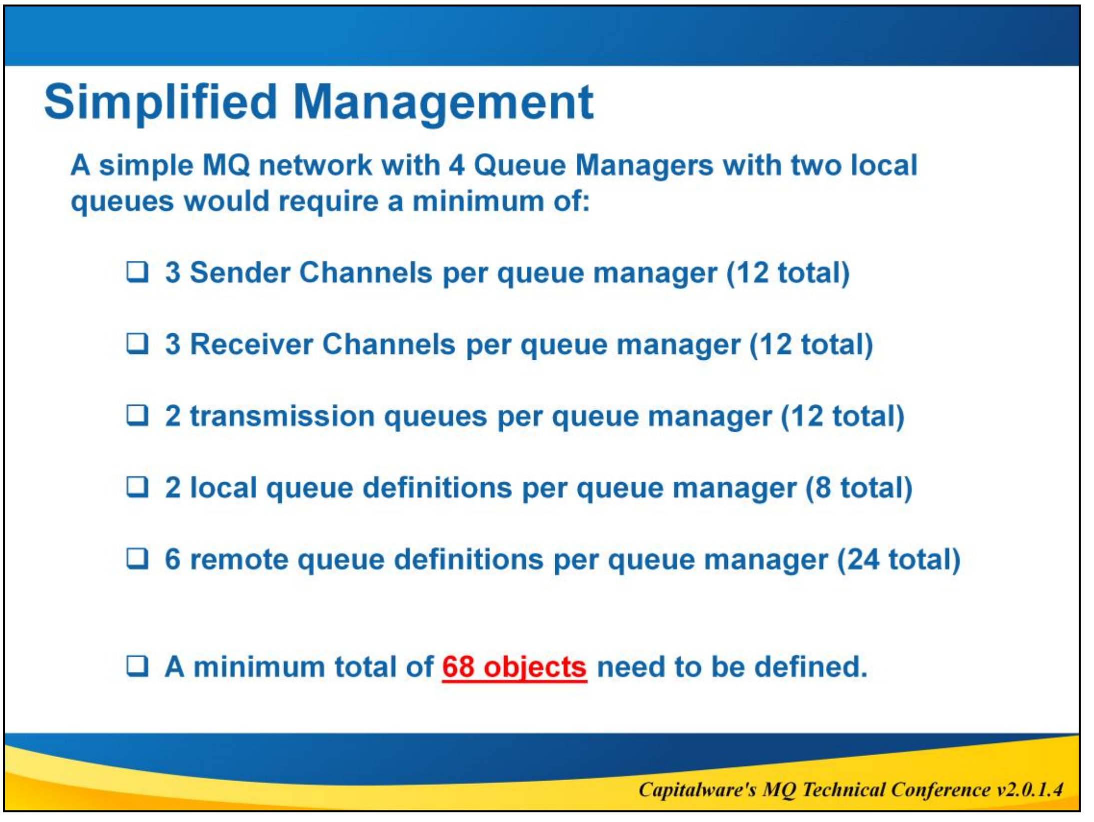
In this system, if a server becomes unavailable, then it is not sent any further messages. If messages are not being processed quickly enough, then another server can be added to improve the processing rate.

It is important that both these behaviors are achieved by existing MQI applications, i.e. without change. It is also important that the administration of clients and servers is easy. It must be straight forward to add new servers and new clients to the server.

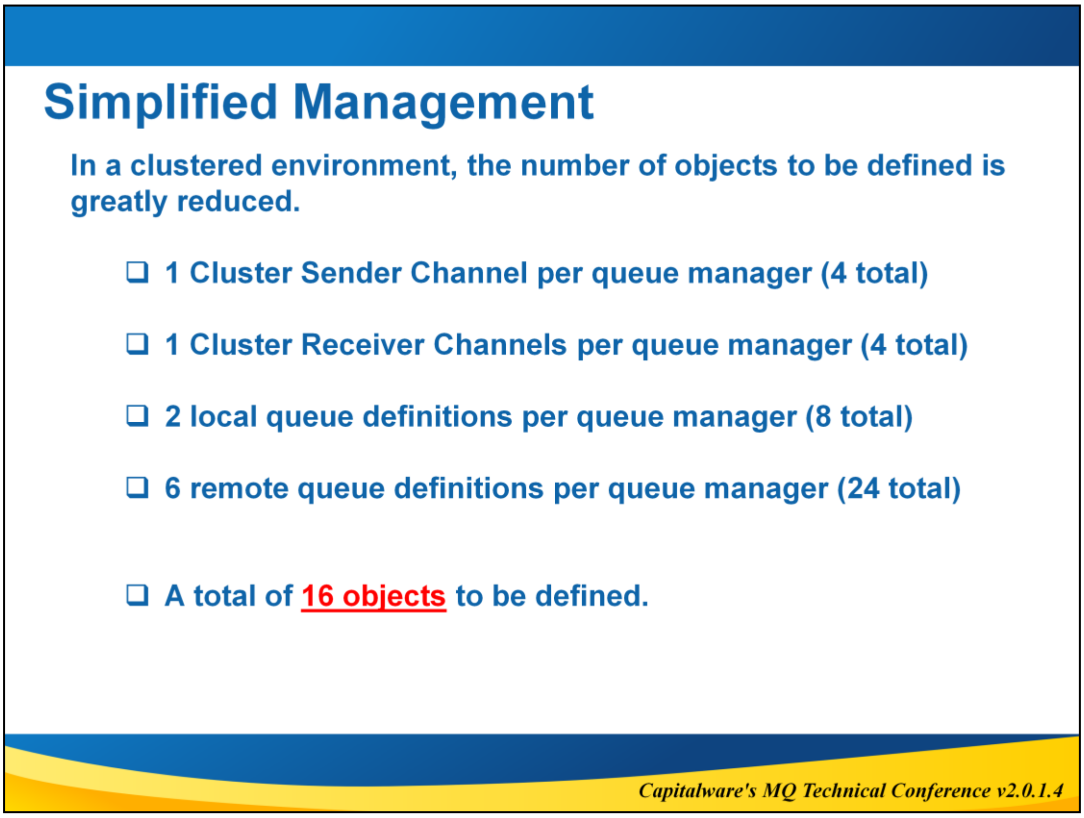
We see how a cluster can provide a highly available and scalable message processing system.

The administration point in processing is MQOPEN as this is when a queue or queue manager is identified as being required by an application.

Note that only one message is sent to a server; it is not replicated three times, rather a specific server is chosen and the message sent there. Also note that MQGET processing is still local, we are not extending MQGET into the network.

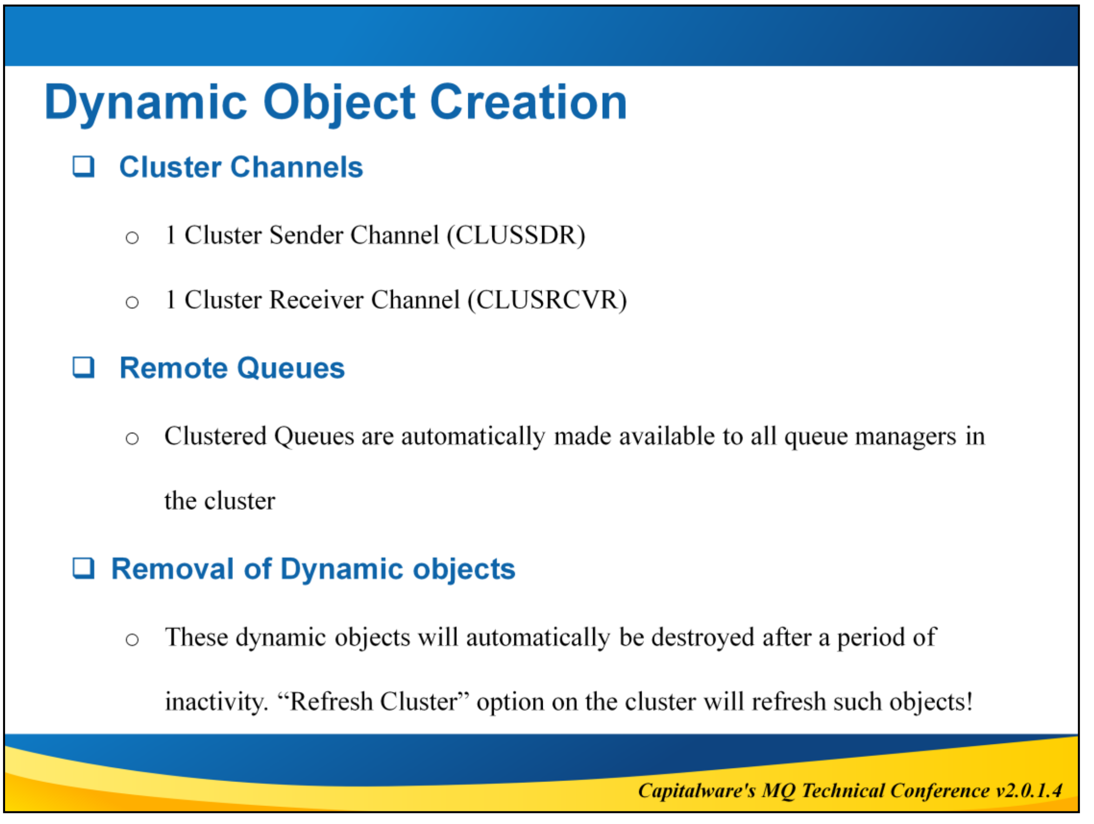


Any MQ admin will agree that managing the MQ objects in a complex network can be very cumbersome. The few the objects, the easier it is to manage both a static MQ network and a growing one.



In a clustered environment, the number of objects to be defined is reduced to 16. This is due to the fact that the only one cluster sender and cluster receiver channel is needed, and no transmission queues or remote queue definitions are necessary.

With this type of configuration, the risk of making errors in defining transmission queues and remote queue definitions is eliminated.



MQ will automatically create and destroy certain objects, as they are needed for clustering. The TWO main type of objects created are cluster channels and cluster queues.

MQ Channels are the number one cause of administrative frustration. MQ Clustering relieves some of those burdens by managing them for us.

To reduce the level of complexity, we only need one CLUSSDR and one CLUSRCVR.

Once these channels are defined, MQ will automagically create the other subsequent channels to the other qmgrs in the cluster

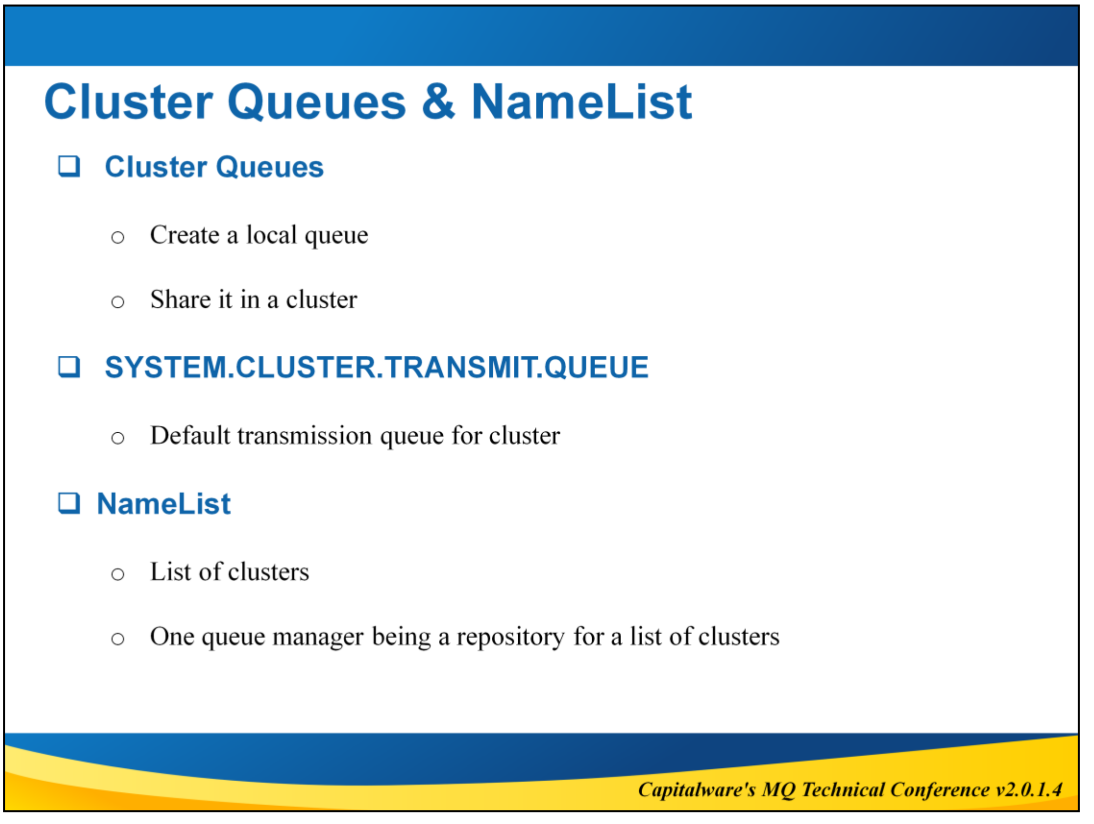
REMOTE QUEUES:

Queues that are defined as cluster queues are automatically made available to the all the queue managers in the cluster. What happens locally is that MQ dynamically creates a remote queue definition on the local qmgr. These queues are can be displayed using MQ Explorer and will appear as “local cluster queues”.

Repository Queue Manager

- **Full Repository**
  - At least 2 FULL repository queue managers need to be defined.
  - Full Repository queue managers are the keepers of cluster information
- **Partial Repository**
  - Contains information about the local system needs to exchange messages with.
- **SYSTEM.CLUSTER.REPOSITORY.QUEUE**
  - Cluster repository information is retained here
- **SYSTEM.CLUSTER.COMMAND.QUEUE**
  - Used to send/receive repository updates from repository queue managers

At least 2 qmgrs are needed for to be FULL repos qmgrs for availability purproses. A full repository cntains information about all the qmgrs in a cluster, including the qmgr names, locations, their channelswhat queues they host, etc.



Cluster queues can be defined by users. Create as local and add it to a cluster.

In addition, a system default "cluster transmission queue' is automatically created when you create a a new qmgr. This queue is similar to the user-defined transmission queue but it does take precedence over it in cases where the target queue is resolved via the repository.

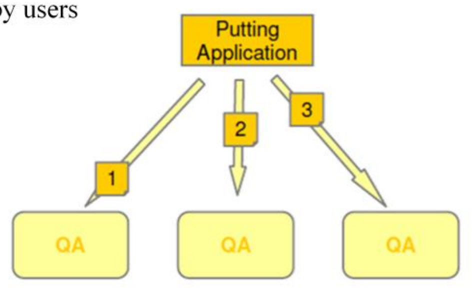
Cluster Objects Summary

- **SYSTEM.CLUSTER.REPOSITORY.QUEUE** – the queue that holds repository information
- **SYSTEM.CLUSTER.COMMAND.QUEUE** – the queue used to send repository update messages
- **SYSTEM.CLUSTER.TRANSMIT.QUEUE** – the default transmission queue
- **SYSTEM.DEF.CLUSSDR** – the default definition for a cluster sender channel
- **SYSTEM.DEF.CLUSRCVR** - the default definition for a cluster receiver channel

Why is it called “Workload Balancing”?

Workload Management

- ❑ **MQ Clusters often implemented to do workload-balancing**
- ❑ **Workload Management algorithm used to accomplish this**
  - Provides a means of distributing workload across set of queues
  - Default values intended to achieve an even distribution of messages
  - Distribution can be configured by users

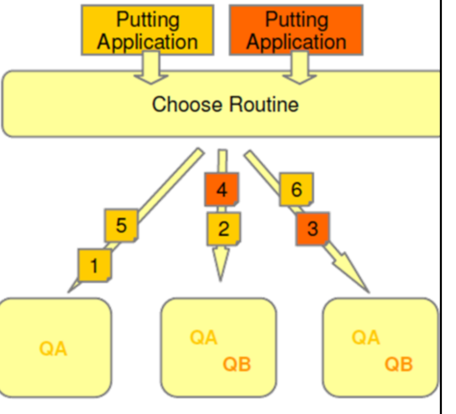


Why Isn't My Workload "Balanced"?

- Once implemented, distribution of messages may not be as expected
- Messages are being distributed across the queues, but not evenly
- May work fine in test, but in production a different behavior is observed
- Why is this?

Multiple applications sharing cluster channels

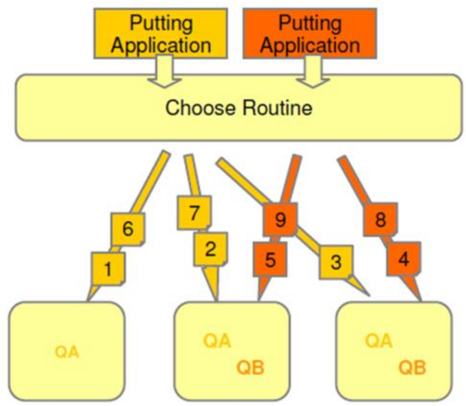
- Distribution of messages may not be as expected
- Seen when more than one putting application in the same queue manager is using cluster queues
- This can occur when frequency of puts between them follows a particular pattern
  - Channels Occurs because the queue manager round-robin's messages based on the channel being used, not the queue that the message is sent to.
- Doesn't always occur; depends on the sequence of puts
- For ease of understanding…
  - Picture two queues, A and B, on two queue managers
  - If the put always happen A, B, A, B, A, B, A, B…
    - Then all messages for A would go to one of the queue managers and all for B go to the other.
  - This is due to the fact that MQ is round-robining across the two channels

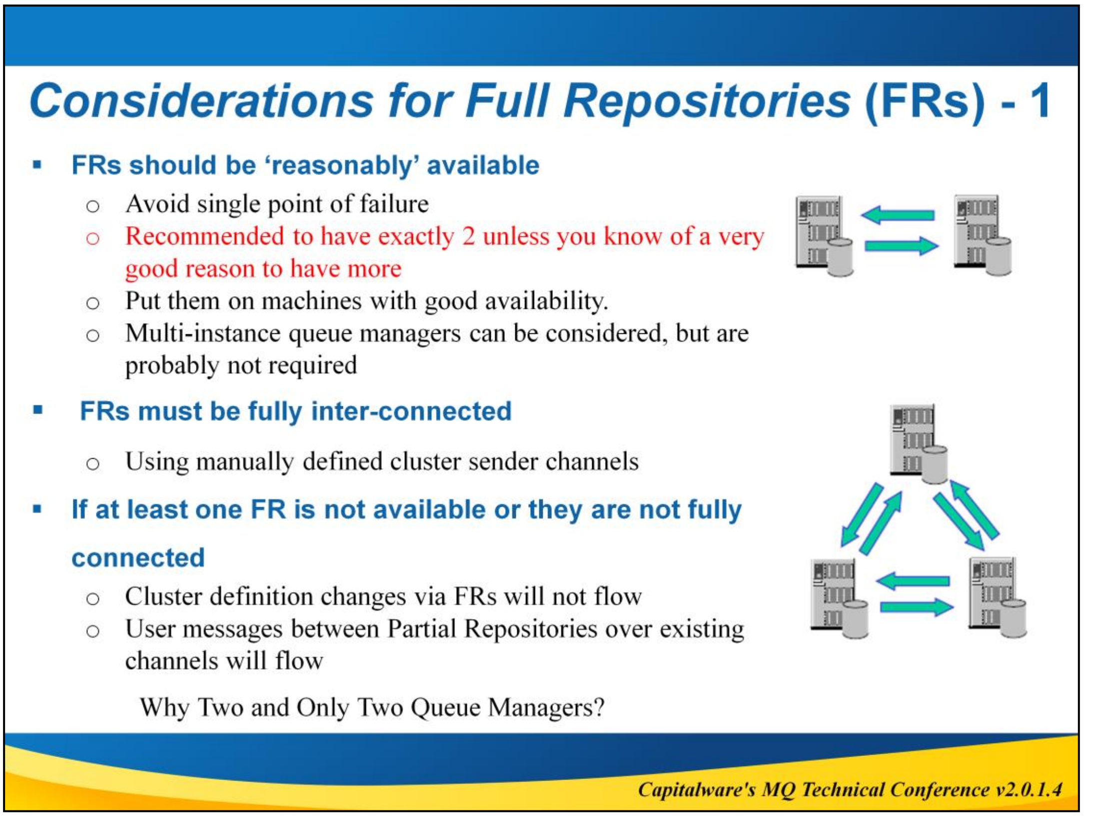


What to do about it?

Use Clusters to Separate Applications

- A new CLUSRCVR can be used to separate applications
- Queues A and B are hosted in different clusters to achieve this





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.

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.

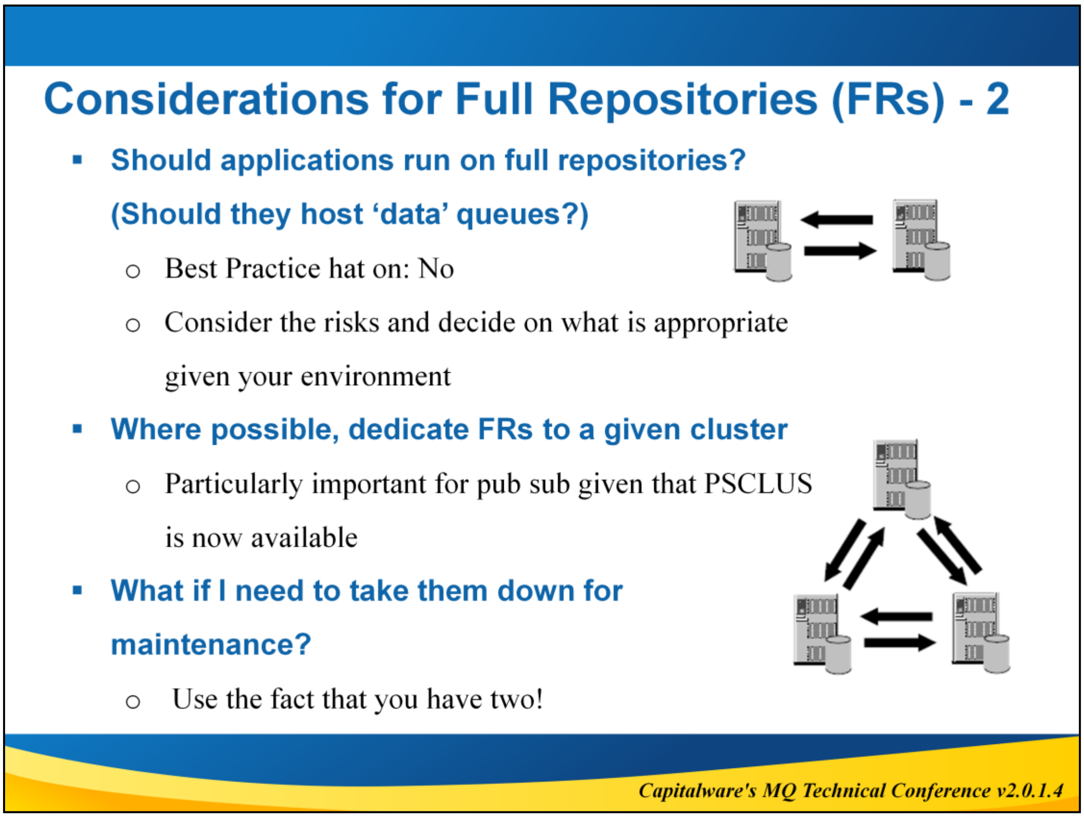
If the Full Repositories are taken off-line as part of scheduled maintenance, then this can be overcome by altering the Full Repositories to be Partial Repositories before taking them off-line, which will cause the queue managers within the cluster to remake their subscriptions elsewhere.

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.



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.

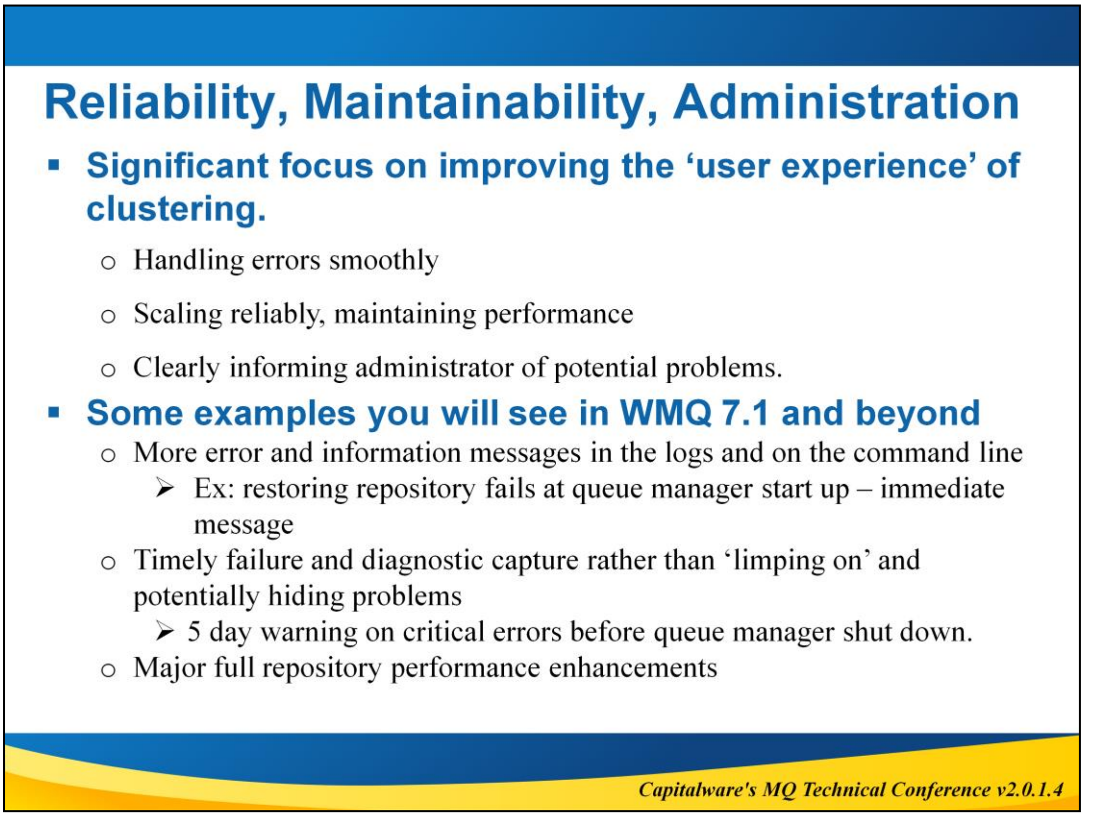
Maintenance:

– 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.

What's New with Clustering in MQ V7.1?



One of the big shifts to note here is from allowing the queue manager to continue without a running repository manager. When the repos manager encounters a critical error it will now go into 'retry' for a few days, issuing warnings with a countdown timer, before shutting down the

queue manager.

This is important because without a repository manager the cluster cache will grow 'stale' and applications may suddenly experience outages sometime in the future, when the root cause may be near impossible to diagnose.

Get-Inhibiting the SYSTEM.CLUSTER.COMMAND.QUEUE suspends this process allowing the administrator to go away and resolve the issue (e.g. fix underlying hardware problem, contact IBM service).

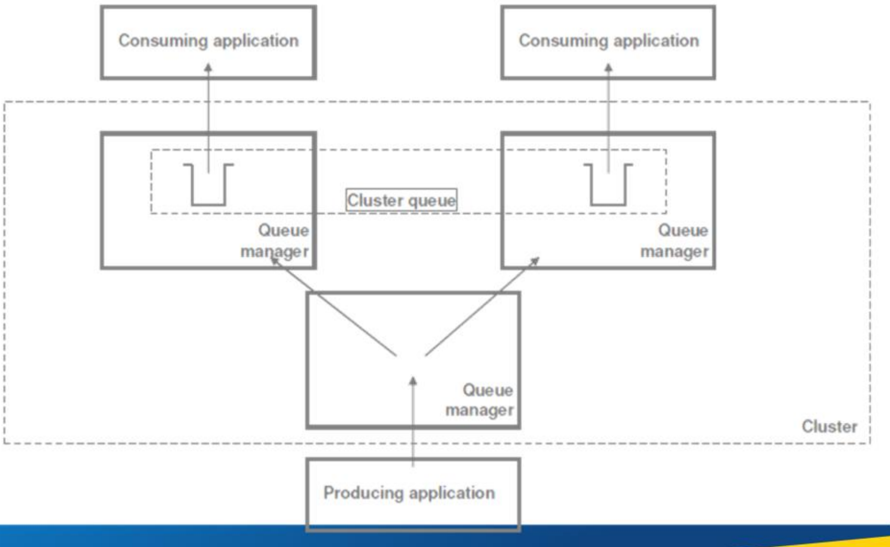
Changes to Workload Management

- **Prior to MQ V7.1, when MQ queue manager clusters were used to distribute messages across multiple cluster queue instances, two "bind" options were available**
  - BIND_ON_OPEN which ensures that all messages for the lifetime of the object handle are routed to the same cluster queue instance
  - BIND_NOT_FIXED which allows 'spraying' across multiple instances.
- **These dictate when the workload management algorithm is invoked**
  - One, when the object is opened
  - Continually, when each message is put
- **"All or Nothing" options did not allow for "in-between" situations, message groups being the primary example**
  - Handling these situations requires application programming involvement

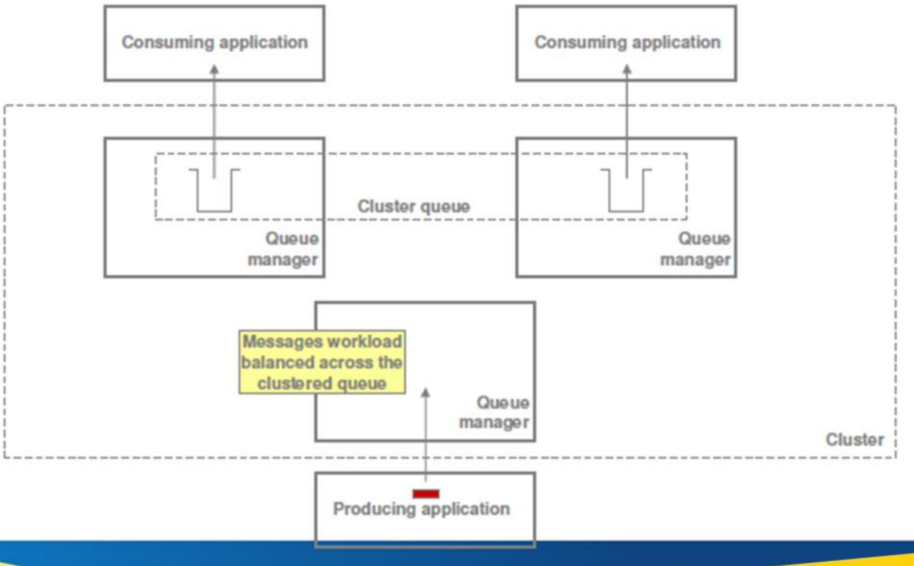
Workload Management With MQ V7.1

- **MQ V7.1 introduces a third option into the mix:**
  - BIND_ON_GROUP allows workload distribution while maintaining the integrity of groups of messages
- **Note that the Workload management algorithm itself is unchanged**
  - What is changed is when the workload management algorithm is invoked
  - With BIND_ON_GROUP, the algorithm will only be driven between complete groups of messages

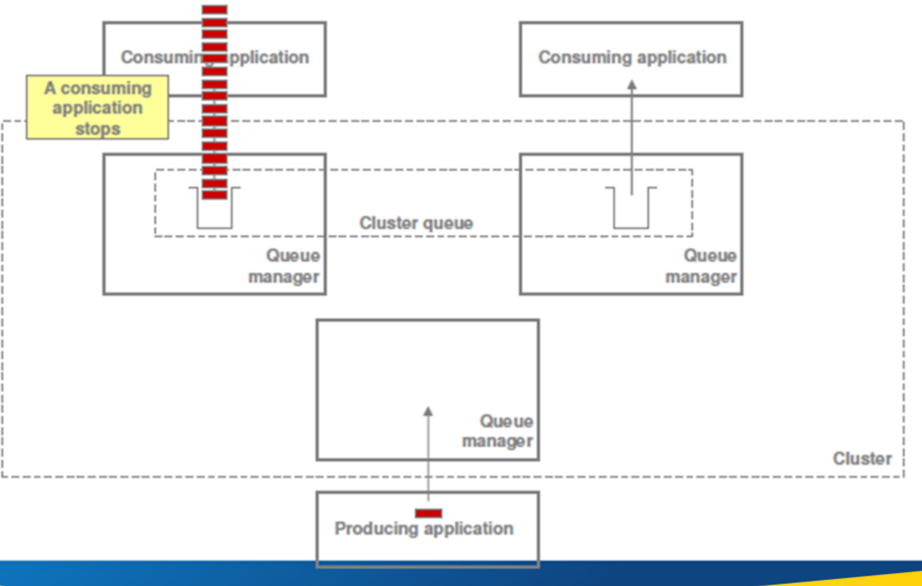
NEW: Cluster Monitoring and Rebalancing
A typical workload balancing scenario

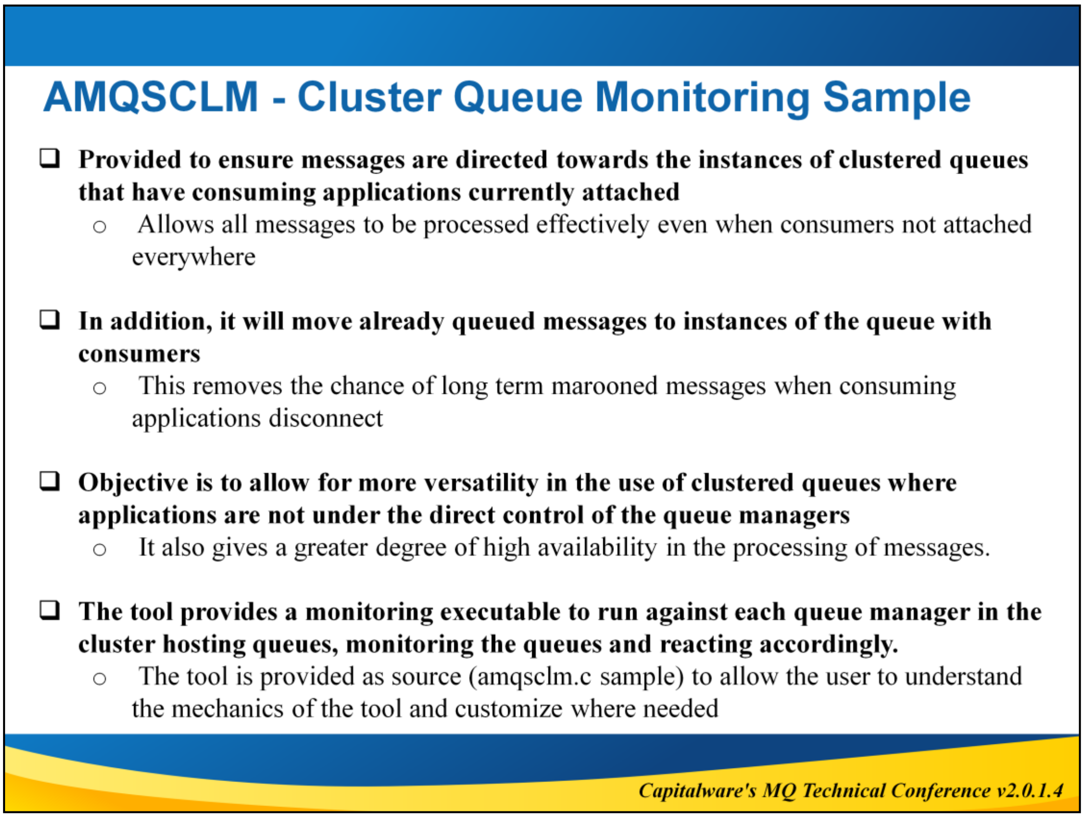


Under normal running…



When things go wrong…





A new tool **amqsclm** is provided to ensure messages are directed towards the instances of clustered queues that have consuming applications currently attached. This allows all messages to be processed effectively even when a system is asymmetrical (i.e. consumers not attached everywhere).

– In addition it will move already queued messages from instances of the queue where no consumers are attached to instances of the queue with consumers. This removes the chance of long term marooned messages when consuming applications disconnect.

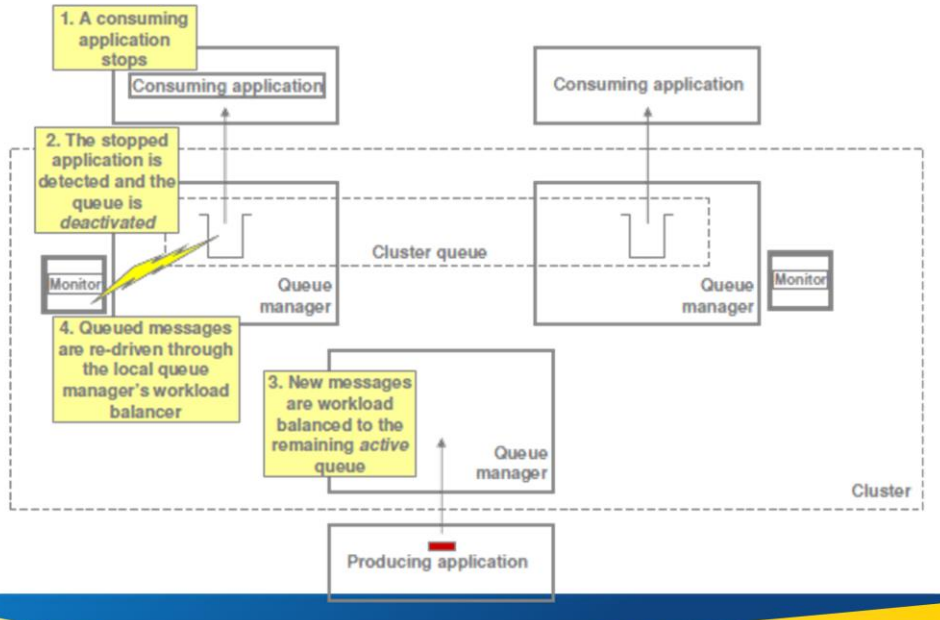
The above allows for more versatility in the use of clustered queue topologies where applications are not under the direct control of the queue managers. It also gives a greater degree of high availability in the processing of messages.

The tool provides a monitoring executable to run against each queue manager in the cluster hosting queues, monitoring the queues and reacting accordingly.

– The tool is provided as source (amqsclm.c sample) to allow the user to understand the mechanics of the tool and customise where needed.

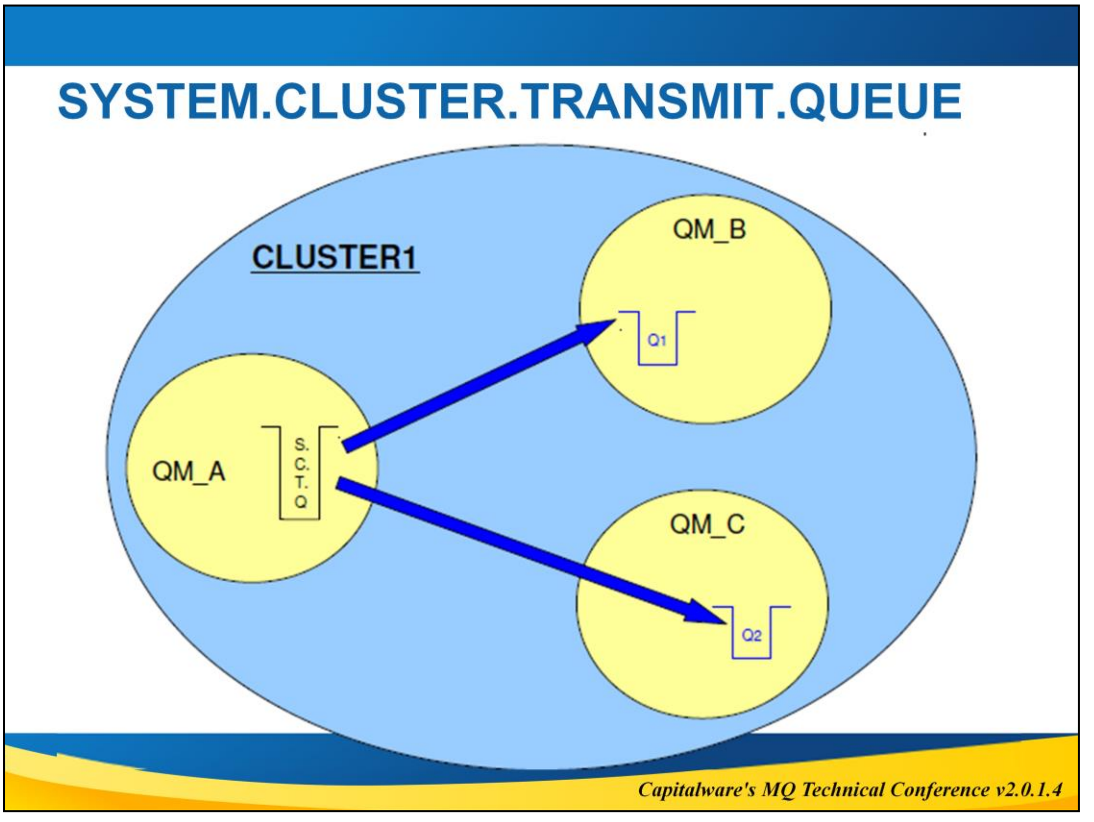
This sample was introduced in WMQ 7.1 (distributed platforms), but has been backported to WMQ 7.0.1 fixpack 8.

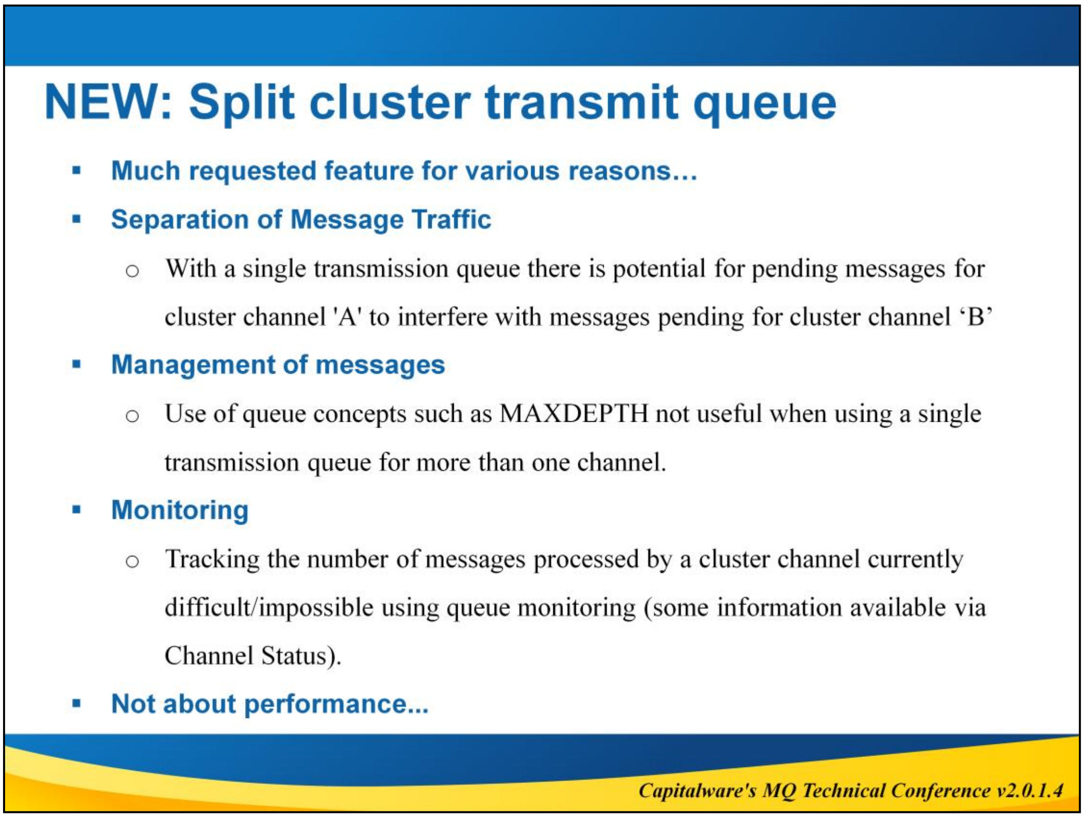
When the queues are monitored…



Split the Cluster Transmit Queue

Capitalware's MQ Technical Conference v2.0.1.4


SYSTEM.CLUSTER.TRANSMIT.QUEUE
CLUSTER1
QM_A
S.
C.
T.
Q
QM_B
Q1
QM_C
Q2
Capitalware's MQ Technical Conference v2.0.1.4



This has been a very long standing requirement from a number of customers

All the reasons on this slide are valid, but the number one reason often quoted in requirements was ‘performance’

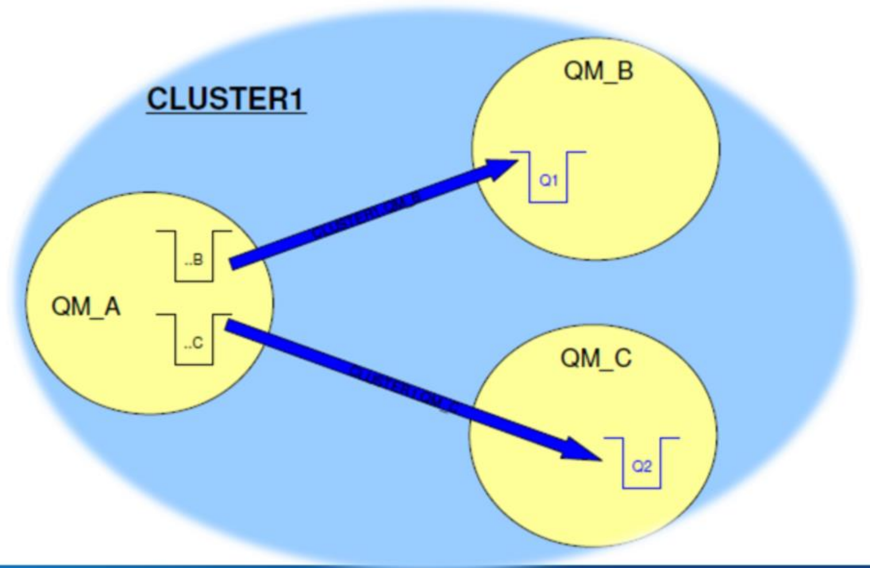
– In reality splitting out the transmit queue does not often buy much here, hence often other solutions (e.g. improving channel throughput) were really needed.

Main reason for delivery now is to allow application separation

Split cluster transmit queue - automatic

- **New Queue Manager attribute which effects all cluster-sdr channels on the queue manager**
  - ALTER QMGR DEFCLXQ (SCTQ | CHANNEL)
- **Queue manager will automatically define a PERMANENT-DYNAMIC queue for each CLUSSDR channel.**
  - Dynamic queues based upon new model queue
    - **"SYSTEM.CLUSTER.TRANSMIT.MODEL"**
  - Well known queue names:
    - **"SYSTEM.CLUSTER.TRANSMIT.<CHANNEL-NAME>"**
- **Authority checks at MQOPEN of a cluster queue will still be made against the SYSTEM.CLUSTER.TRANSMIT.QUEUE even if CLUSSDR is selected.**

Splitting out the S.C.T.Q. per channel



Split cluster transmit queue - manual

- **Administrator manually defines a transmission queue and using a new queue attribute defines the CLUSSDR channel(s) which will use this queue as their transmission queue.**
  - DEFINE QLOCAL(APPQMGR.CLUSTER1.XMITQ) CHLNAME(CLUSTER1.TO.APPQMGR) USAGE(XMITQ)
- **The CHLNAME can include a wild-card at the start or end of to allow a single queue to be used for multiple channels. In this example, assuming a naming convention where channel names all start with the name of the cluster, all channels for CLUSTER1 use the transmission queue CLUSTER1.XMITQ.**
  - DEFINE QLOCAL(CLUSTER1.XMITQ) CHLNAME(CLUSTER1.*) USAGE(XMITQ)
  - Multiple queues can be defined to cover all, or a subset of the cluster channels.
- **Can also be combined with the automatic option**
  - Manual queue definition takes precedence.

Questions & Answers