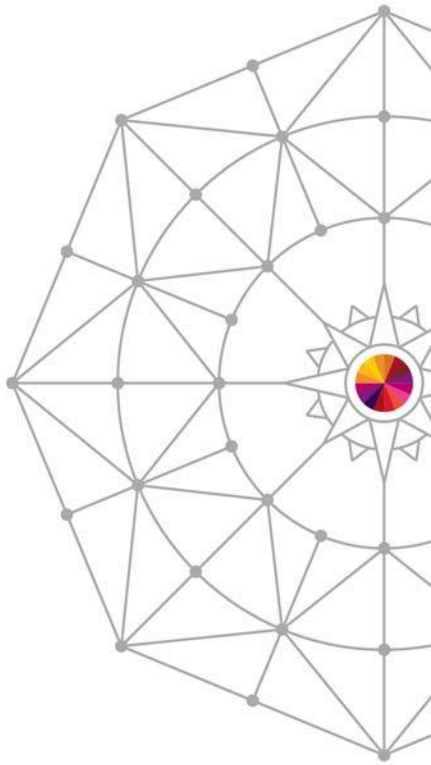
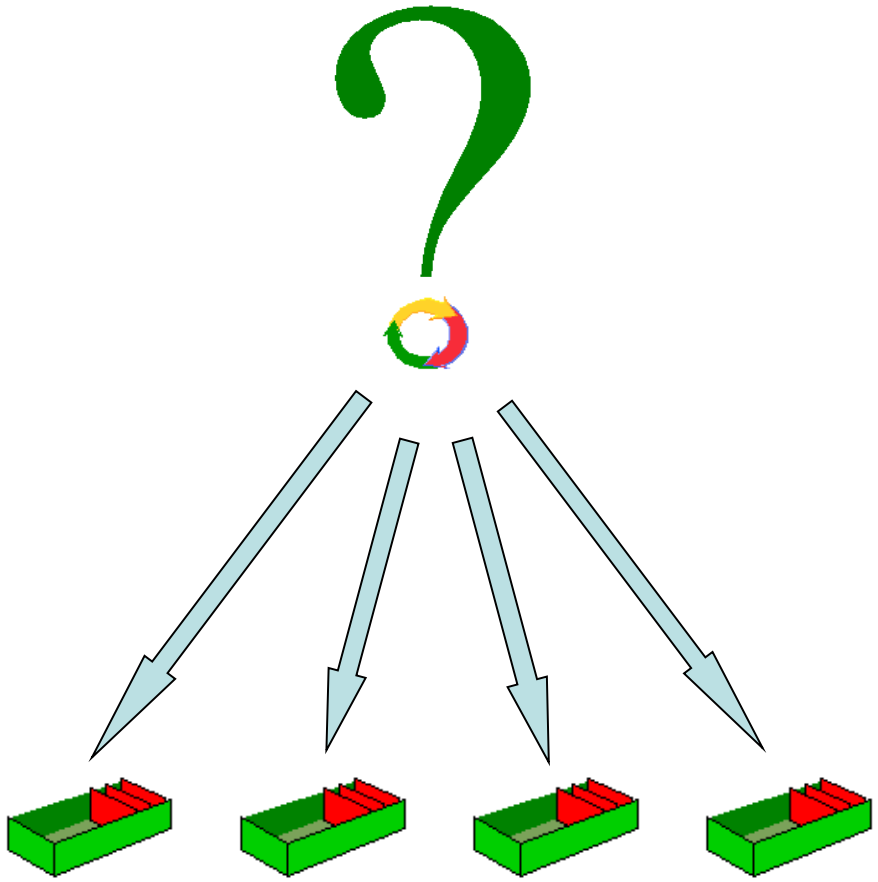
Session 16196
MQ Clustering - The Basics, Advances and What's New in V8



Neil Johnston
IBM UK
neilj@uk.ibm.com

Agenda

- The purpose of clustering
- Defining a cluster
- Lifting the Lid on Clustering
- Workload Balancing
- Flexible topologies - routing
- Further Considerations
- Recommendations
- What's New

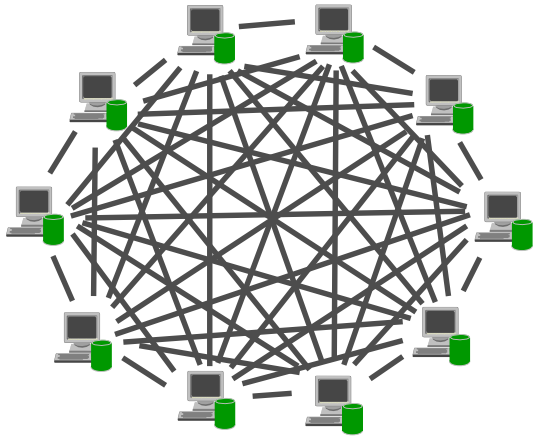


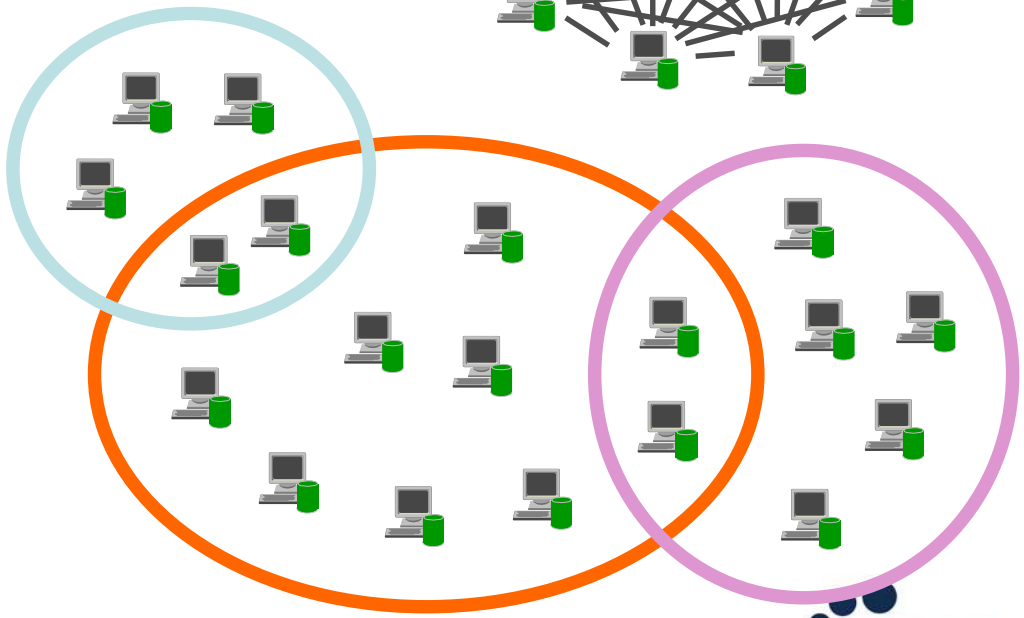
What are clusters?

- The term clusters means has different meaning for different people as there are a large number of technologies that use the term and a large number of products that offer “clustering”.
- Clustering is usually associated with parallel processing or high availability technologies.
- There exist many different types of parallel processing architectures. The amount of symmetry between the processing nodes varies between architectures.
- Although WMQ clustering does provide a level of high availability, its raison d’etre is as a WMQ parallel processing feature.
- The most symmetric systems are similar to SP2. These are essentially a set of identical processors (RS/6000) connected together using a high speed switch which allows fast communication between the individual nodes.
- Systems like z/OS Parallel Sysplex consist of complexes of processors, each complex comprising numerous, but same type processors. The complex is connected together using a coupling facility, which, as well as allowing high speed communication, also allows efficient data sharing. Most parallel architectures do not have this type of data sharing capability.
- The most generalized parallel architectures involve processors of different types, running different operating systems connected together using different network protocols. This necessarily includes symmetric systems like SP2 and Parallel Sysplex.
- A WebSphere MQ cluster is most similar to the most generalized parallel architecture. This is because WebSphere MQ exploits a wide variety of platforms and network protocols. This allows WebSphere MQ applications to naturally benefit from clustering.
- WebSphere MQ clusters are solve a requirement to group queue managers together (i.e. to increase processing power of the WMQ network) whilst minimising the administration costs associated with WMQ queue manager intercommunication.

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?

- 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.

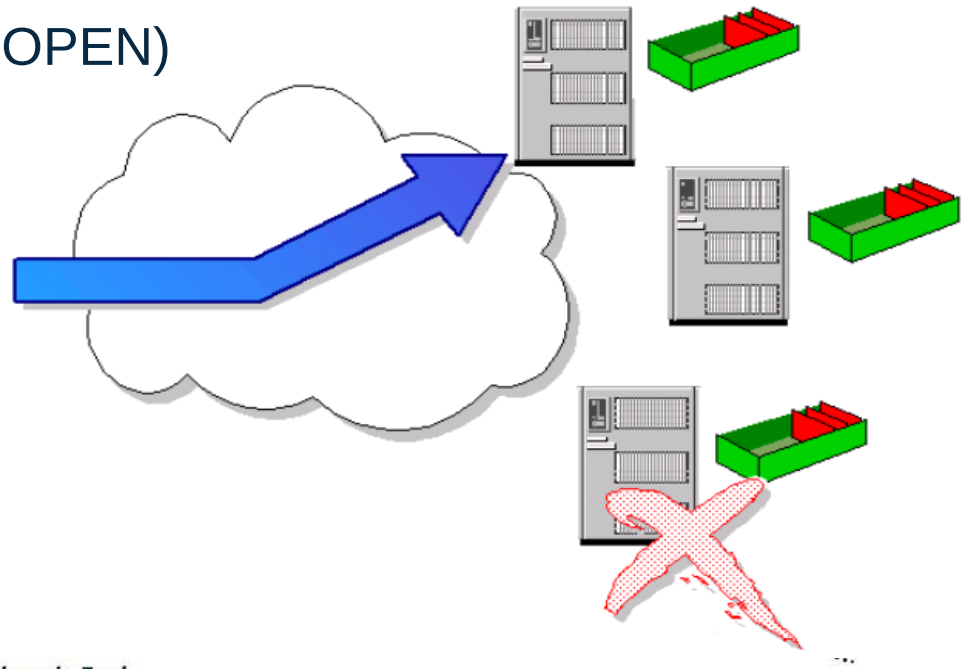
- By contrast, "loosely coupled" clusters (e.g. the Internet) have processors that are more or less independent of each other. Data transferred to one processor is owned by it and is not affected by other processors. Such systems do not suffer from processor locking issues. In a cluster solution, there 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.

Goals of Clustering

- Multiple Queues with single image
- Failure isolation
- Scalable throughput
- MQI applications to exploit clusters transparently
- Definition through usage (MQOPEN)
- MQGET always local



Goals of Clustering

- Consider a client using the black 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.

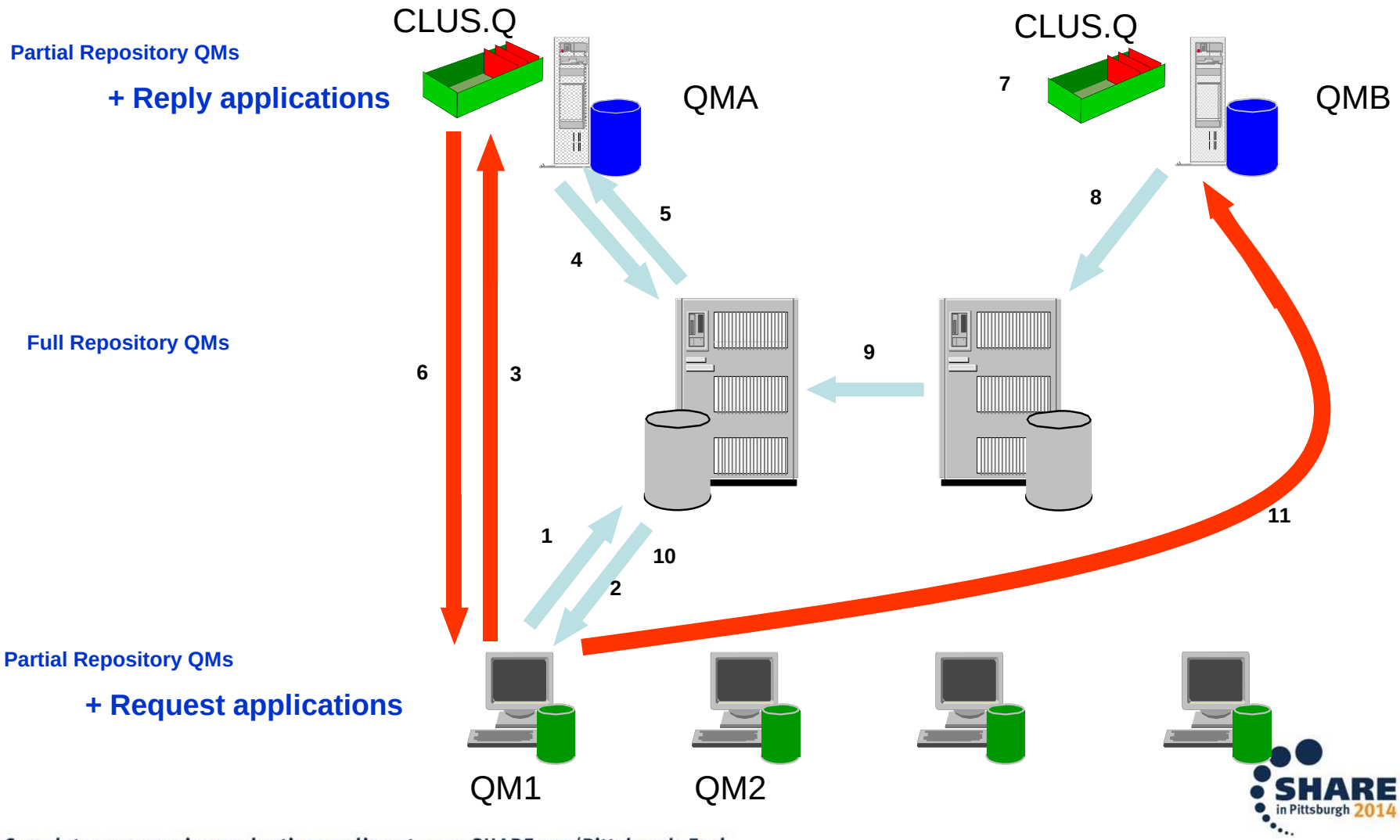
- We see how a cluster can provide a highly available and scalable message processing system. The administration point in processing is MQOPEN as this is when a queue or queue manager is identified as being required by an application.

- 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.

Repositories – Full and Partial

- Each queue manager in a cluster is a repository:
  - It stores information about objects in the cluster
- Full repository (usually 2 per cluster):
  - Stores all information about all objects in the cluster
- Partial repository:
  - Only stores information that it needs to know about

WMQ Cluster Architecture



WMQ Cluster Architecture

- Reply queue manager hosts applications that send request and receive reply.
- Request queue manager hosts applications that receive request and send reply.
- Let us walk through the flow of a message from a request queue manager to a reply queue manager and the response message that is returned. We assume that the request app has never used the queue providing the service previously, and that the request app has not communicated with the request app previously.
- Clustering introduces a new architectural layer, the Full Repository and Partial Repository queue managers, purely for the sake of explanation. Full Repository queue managers are not separate queue managers (contrast to DNS servers), and their role is to serve as a global directory of queues and queue managers in the cluster. There are a small number of these Full Repositories. Each request and reply queue manager have a Partial Repository. In practice, Full Repository queue managers often host applications too.

WMQ Cluster Architecture

- At MQOPEN, the queue manager to which the application is connected detects that this queue has not been used by this queue manager previously. The queue manager sends an internal message (1) requesting the location of the servers for the green queue and channel definitions to get to these servers. The returned definitions (2) are installed on the request queue manager, a channel is automatically defined to the reply queue manager (3).

- Similar processing occurs at the request side, to locate the requestor's ReplyToQueue manager.

- The most frequent method used by a reply app to send a response message to a requestor is to use the routing information in the message descriptor, namely the ReplyToQ and ReplyToQMgr. The reply app's requirement is slightly different to the original request, since the originating application's ReplyToQ is normally private to its Queue manager, i.e. it is not visible to the whole cluster. In this case the server needs to be able to locate the ReplyToQMgr rather than the ReplyToQ.

WMQ Cluster Architecture

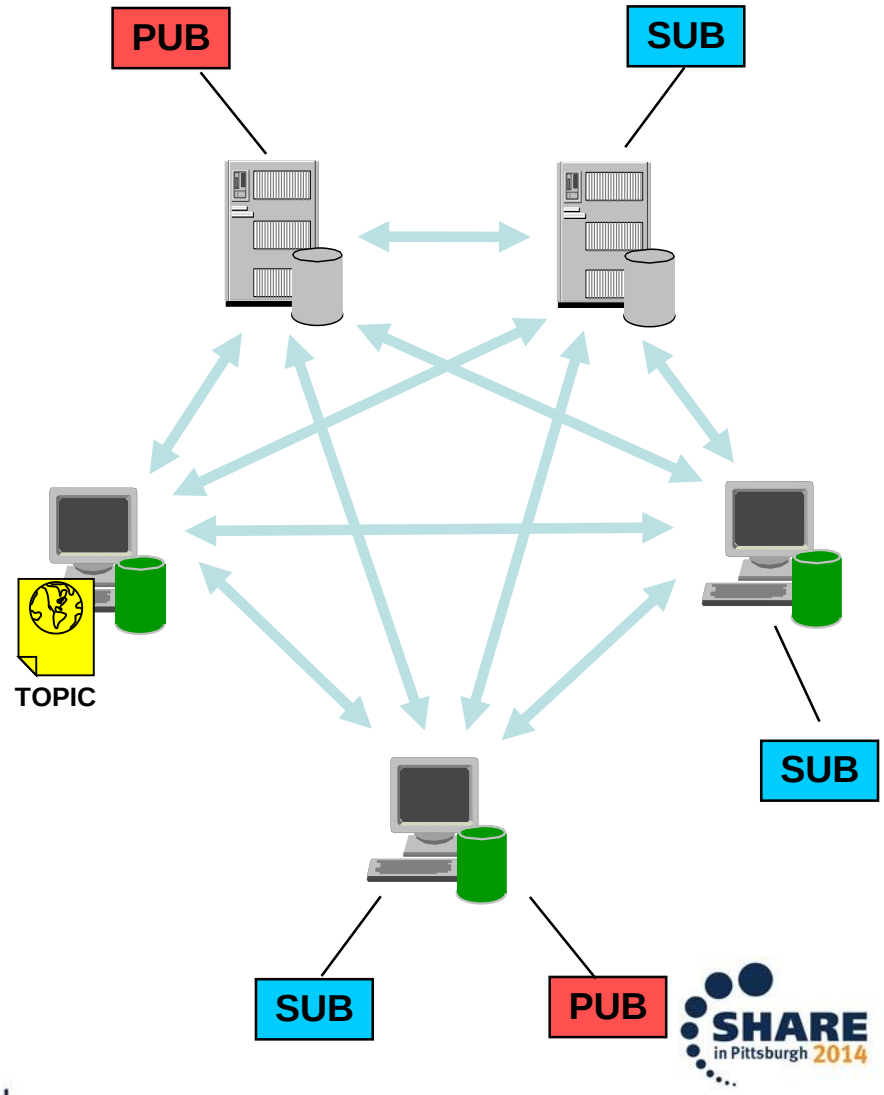
- This happens as follows. When an MQPUT1 or MQOPEN request is made to send a message to a ReplyToQMgr for the first time, the queue manager sends an internal message (4) to the repository requesting the channel definition required to reach the client. The returned definition (5) is used to automatically define a channel to the request queue manager (6) to get the message back to the request queue manager where local queue resolution puts the message on the ReplyToQ.

- Finally, we see what happens when some attributes of a cluster queue or cluster Queue manager change. One interesting case is the creation of a new instance of a cluster queue manager holding a cluster queue being used by a request (7). This information is propagated from the reply queue manager (8). The Full Repository propagates the definition to the other Full Repository (9) and the Full Repository propagates it to any interested request QMs through the repository network (10), allowing this new instance to be used during an MQOPEN, MQPUT or MQPUT1 call (11).

- Note that channels are only automatically defined once on the first MQOPEN that resolves to a queue on the remote queue manager, not for every MQOPEN or MQPUT.

Pub/sub Clusters

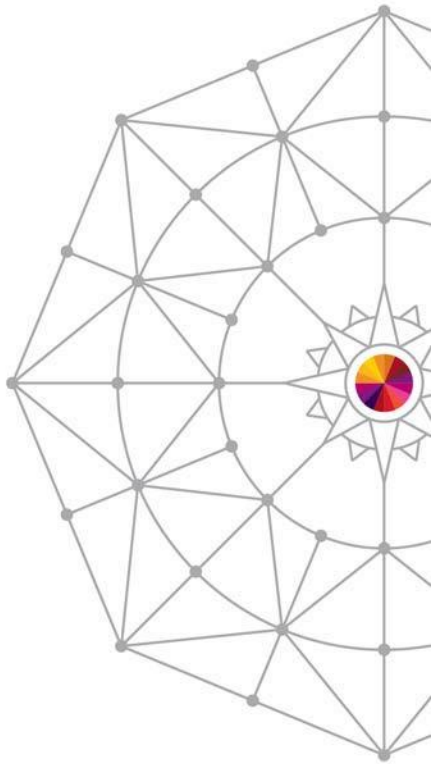
- For distributed pub/sub
- Based on clustered topic objects
  - Hosted on one or more queue managers in the cluster
- Based on clustering for object auto-definition
  - All to all queue manager connectivity
  - Channel auto-definition on first cluster topic definition
- How large?
  - No more than100 queue managers



Pub/sub Clusters

- WebSphere MQ V7 introduced Pub/sub Clusters which are used for distributed pub/sub, allowing publishers to send publications to remote subscribers.
- Pub/Sub clusters use the underlying clustering features to provide automatic connectivity between queue managers.
- The point of control for Pub/sub clusters is the topic object which can be administratively shared in the cluster, just as queues can be shared in a cluster.
- We will look at this in more detail a little later.

Cluster configuration



Resource definition for a Partial Repository

```
DEFINE CHANNEL(TO.QM1) CHLTYPE(CLUSRCVR) TRPTYPE(TCP)
CONNAME(MACHINE1.IBM.COM) CLUSTER(DEMO)
```

- CLUSRCVR definition provides the information to the cluster that allows other queue managers to automatically define sender channels

```
DEFINE CHANNEL(TO.QM2) CHLTYPE(CLUSSDR) TRPTYPE(TCP)
CONNAME(MACHINE2.IBM.COM) CLUSTER(DEMO)
```

- **CLUSSDR definition must direct the queue manager to a Full Repository where it can find out information about the cluster**

```
DEFINE QLOCAL(PAYROLLQ) CLUSTER(DEMO)
```

- **Queues can be advertised to the cluster using the CLUSTER() keyword**

```
DEFINE TOPIC(SPORTS) TOPICSTR(/global/sports) CLUSTER(DEMO)
```

- **…and so can Topics**

Resource definition for Partial Repository QM1

- The cluster channels are the backbone of the cluster, and it is worth taking a little time to understand what they are used for. The use varies slightly depending on whether the queue manager is going to be used as a Full Repository or a Partial Repository.

Partial Repository

- To get a Partial Repository queue manager running in the cluster, you will need to define a cluster sender channel and a cluster receiver channel.

- The cluster receiver channel is the most important of the two as it has 2 roles. (1) It is used as a standard receiver channel on the queue manager on which it is defined. (2) The information it contains is used by other queue managers within the cluster to generate automatically defined cluster sender channels to talk to the queue manager on which the cluster receiver is defined. It is for this reason that the cluster receiver channel definition contains a number of attributes usually associated with sender channels such as the conname.

- The cluster sender channel is used to point the Partial Repository at a Full Repository queue manager from which it can then find out about other Full Repository queue managers and resources in the cluster. The cluster sender channel on a Partial Repository queue manager could be considered to be a boot-strap. Once the Partial Repository has exchanged some initial information with the Full Repository, the manually defined cluster sender channel is no longer required, but can be used to preferentially choose a Full Repository to publish and subscribe for cluster resources as mentioned previously.

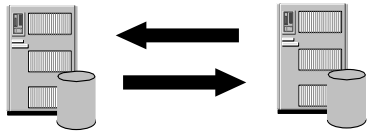
Resource definition for Full Repositories

Full Repository

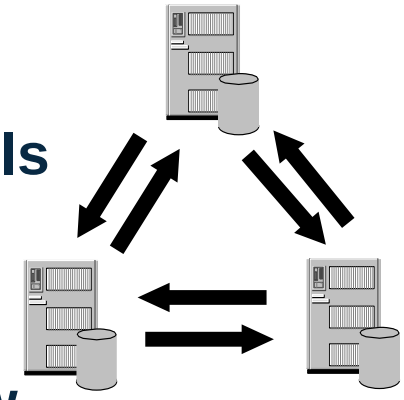
- A full repository is created by issuing alter qmgr(CLUSNAME)
- On a Full Repository, the role of the cluster receiver channel is the same :- it provides the information required by other queue managers in the cluster to talk to the Full Repository. The difference is in the use of the cluster sender channels. Manually defined cluster sender channels must be used to link all of the Full Repositories within the cluster. For the Full Repositories to communicate correctly, these channels need to connect the Full Repositories into a fully connected set. In a typical scenario with 2 Full Repositories, this means each Full Repository must have a cluster sender channel to the other Full Repository. These manual definitions mean that the flow of information between the Full Repositories can be controlled, rather than each one always sending information to every other Full Repository.
- The 2 key points to remember are:
- The Full Repositories must form a fully connected set using manually defined cluster sender channels.
- The information in the cluster receiver will be propagated around the cluster and used by other queue managers to create auto defined cluster sender channels. Therefore it is worth double checking that the information is correct.
- Note: If you alter a cluster receiver channel, the changes are not immediately reflected in the corresponding automatically defined cluster sender channels. If the sender channels are running, they will not pick up the changes until the next time they restart. If the sender channels are in retry, the next time they hit a retry interval, they will pick up the changes.

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 effect 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.

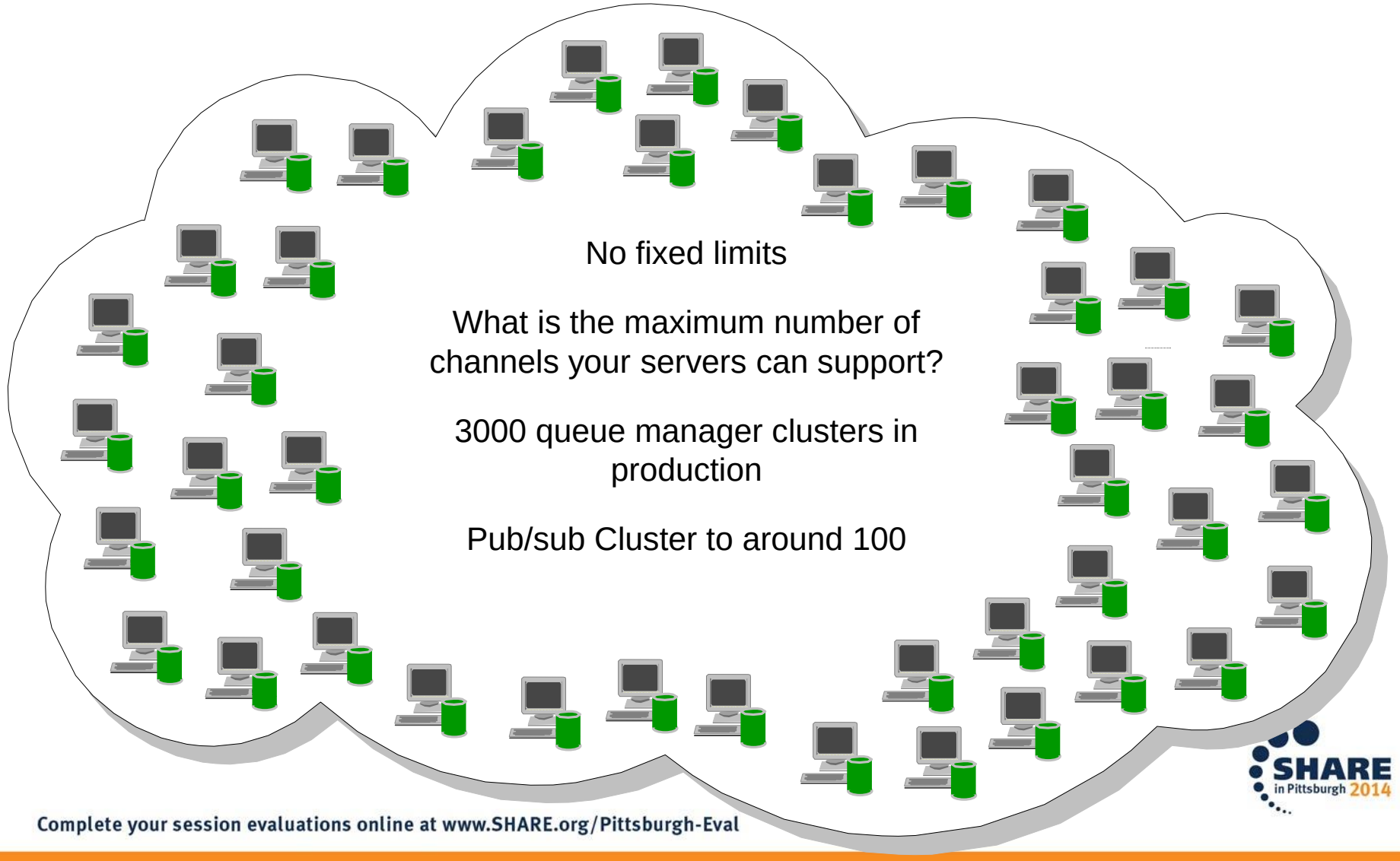
- If you want a Partial Repository to subscribe to a particular Full Repository queue manager, then manually defining a cluster sender channel to that queue manager will make the Partial Repository attempt to use it first, but if that Full Repository is unavailable, it will then use any other Full Repositories 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.

How big can a cluster be?



How big can a cluster be?

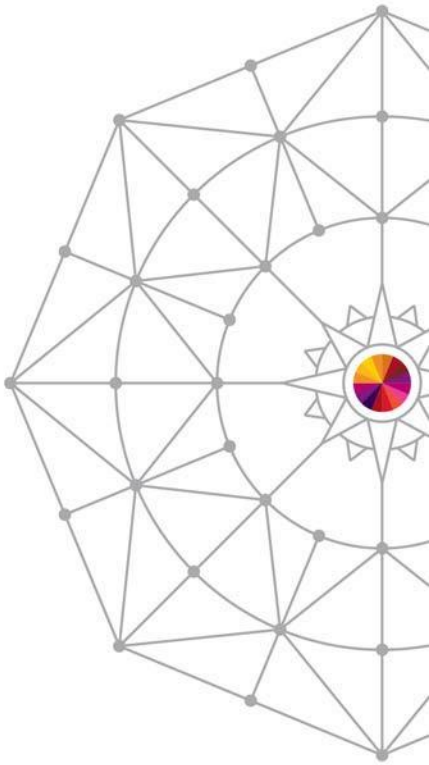
- There are no predefined limits to the size of a cluster. The main thing to consider is the maximum number of channels that will need to connect to your server queue managers (and also to the Full Repositories). If for instance, you are hosting cluster queues on a WMQ z/OS queue manager, then the maximum number of channels is about 9000. This would give a maximum theoretical cluster size of 4500 queue managers, if each queue manager in the cluster had one channel going to the server queue manager and one channel back from the server queue manager.

- If you require more queue managers than this, there are ways that this can be achieved. For example you could have more than one backend server and a customized workload balancing exit could be written to partition the work across the different servers. The internal workload algorithm will always round robin across all the servers so they would each need to cope with channels from all the clients. We will look closer at workload balancing later in the presentation.

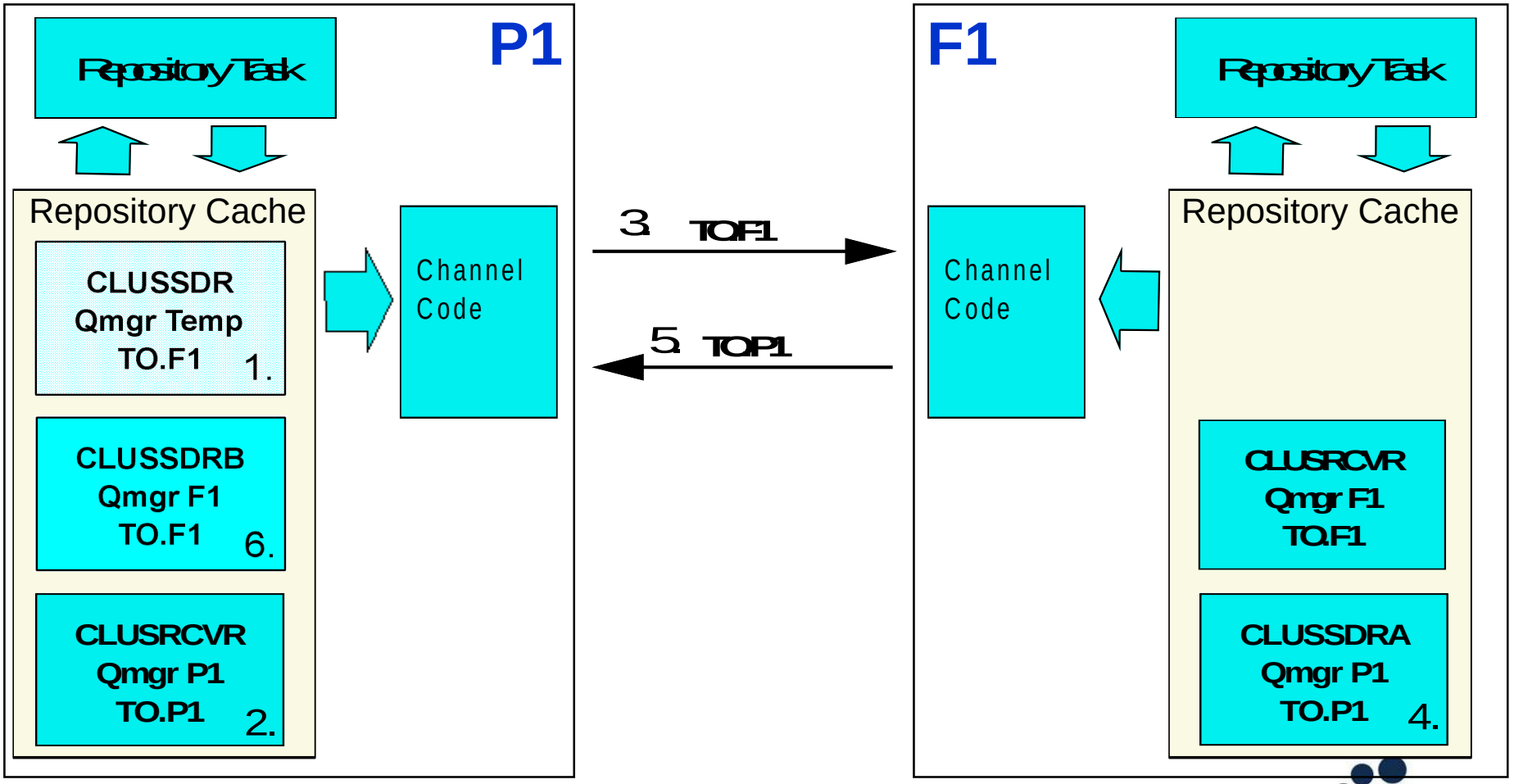
- One question we are regularly asked is whether it is better to have one large cluster or multiple smaller clusters. This should usually be a business decision only. It may be for example that you wish to separate secure requests coming from branches of the business over channels using SSL from un-secure requests coming from the internet. Multiple clusters would allow you to achieve this whilst still using the same backend servers to handle both request types.

- Pub/subs clusters scale into the 100s of queue managers, not the 1000s due to the all to all connectivity and overhead of proxy-subscriptions.

Lifting the Lid on Clustering



Initial definitions: Adding a Partial Repository



Initial definitions

- The previous diagram shows the communications that occur between a Partial Repository and a Full Repository queue manager when the Partial Repository is added to the cluster.

- A cluster sender channel is defined on the Partial Repository. The queue manager name seen on a DISPLAY CLUSQMGR command, shows a name that starts SYSTEM.TEMPQMGR. This name will be changed when the Partial Repository has communicated with the Full Repository and found out the real name of the Full Repository.

- A cluster receiver channel is defined on the Partial Repository.

- Once both a cluster sender channel and a cluster receiver channel for the same cluster are defined, the repository task starts the cluster sender channel to the Full Repository.

- The information from the cluster receiver channel on the Partial Repository is sent to the Full Repository which uses it to construct an auto defined cluster sender channel back to the Partial Repository.

- The repository task on the Full Repository starts the channel back to the Partial Repository and sends back the information from its cluster receiver channel.

- The Partial Repository merges the information from the Full Repositories cluster receiver channel with that from its manually defined cluster sender channel. The information from the Full Repositories cluster receiver channel takes precedence.

Checking initial definitions

- On the Full Repository display the new Partial Repository queue manager

```
DISPLAY CLUSQMGR(<partial repos qmgr name>)
```

- If the Partial Repository queue manager is not shown, the Full Repository has not heard from the Partial Repository
  - **Check channel status from the Partial to the Full**
  - **Check CLUSRCVR and CLUSSDR channels CLUSTER name**
  - **Check for error messages**
- On the Partial Repository display all cluster queue managers

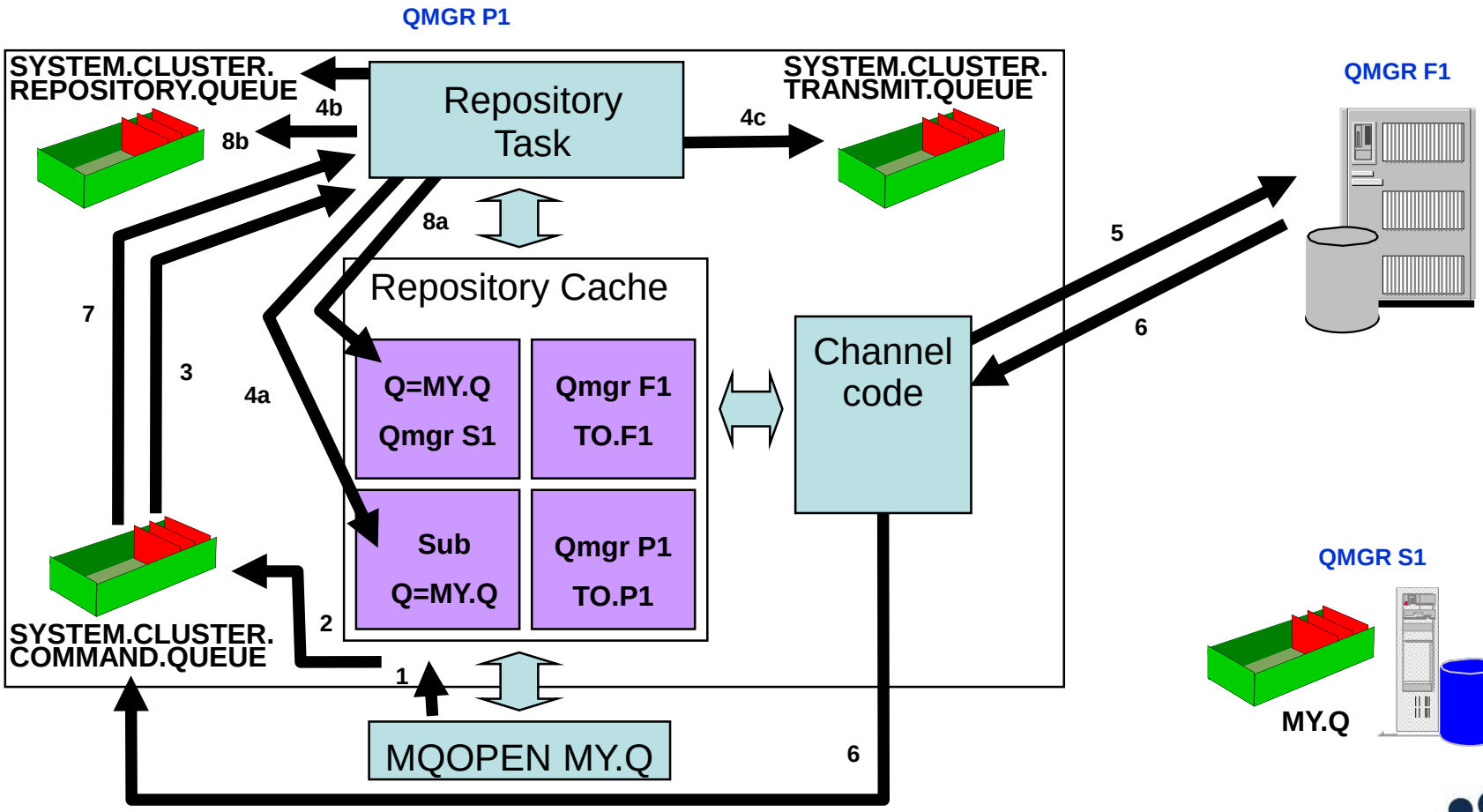
```
DISPLAY CLUSQMGR(*)
```

- If the channel to the Full Repository has a queue manager name starting SYSTEM.TEMPQMGR, the Partial Repository has not heard back from the Full Repository
  - **Check channel status from the Full to the Partial**
  - **Check for error messages**

Checking initial definitions

- On the Partial Repository, issue DISPLAY CLUSQMGR(*) to display all the cluster queue managers that the Partial Repository is aware of. What we want to check is that the manually defined cluster-sender channel to the Full Repository no longer has a CLUSQMGR name starting SYSTEM.TEMPQMGR. If it does, then we know that the initial communications between the Partial Repository and the Full Repository have not completed. The checklist below should help to track down the problem:
- Check the CLUSTER name defined on the CLUSRCVR and CLUSSDR channels
  - **The Partial Repository will not attempt to talk to the Full Repository unless both the CLUSRCVR and the CLUSSDR channels are defined with the same cluster name in the CLUSTER keyword.**
- Check the status of the channel to the Full Repository.
  - **The CLUSSDR channel to the Full Repository should be RUNNING. If it is in status RETRYING, check the CONNAME is correct, the network is ok and that the listener on the Full Repository is running.**
- On Full Repository, issue DISPLAY CLUSQMGR(<partial repos name>)
  - **If the Partial Repository has established a connection with the Full Repository, it will have sent the information from its CLUSRCVR channel to the Full Repository which will have used it to create an auto-defined CLUSSDR channel back to the Partial Repository. We can check if this has occurred by issuing the DISPLAY CLUSQMGR command on the Full Repository. Check channel status of the channel back from Full Repository.**
  - **The Full Repository should send the information from its CLUSRCVR channel back to the Partial Repository, thus the channel back to the Partial Repository should be RUNNING. If the channel is RETRYING, check the CONNAME on the Partial Repository's CLUSRCVR is correct (remember this is where the Full Repository got the CONNAME from) and check that the listener on the Partial Repository is running.**
- Check for error messages
  - **If an error occurs in the repository task on either the Partial Repository queue manager or the Full Repository queue manager, error messages are produced. If the checklist above does not provide the answer to the problem, it is worth examining the error logs from the queue managers as this may aid the problem diagnosis.**

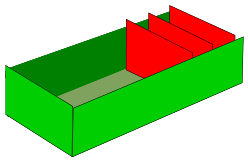
First MQOPEN of a Queue



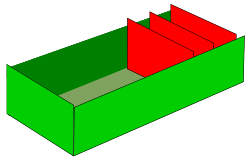
First MQOPEN of a Queue

- The previous diagram shows what happens when a cluster queue is opened for the first time on a queue manager.
- The MQOPEN request is issued and as part of the MQOPEN, the repository cache is examined to see if the queue is known about. No information is available in the cache as this is the first MQOPEN of the queue on this qmgr.
- A message is put to the SYSTEM.CLUSTER.COMMAND.QUEUE requesting the repository task to subscribe for the queue.
- When the repository task is running, it has the SYSTEM.CLUSTER.COMMAND.QUEUE open for input and is waiting for messages to arrive. It reads the request from the queue.
- The repository task creates a subscription request. It places a record in the repository cache indicating that a subscription has been made and this record is also hardened to the SYSTEM.CLUSTER.REPOSITORY.QUEUE. This queue is where the hardened version of the cache is kept and is used when the repository task starts, to repopulate the cache. The subscription request is sent to 2 Full Repositories. It is put to the SYSTEM.CLUSTER.TRANSMIT.QUEUE awaiting delivery to the SYSTEM.CLUSTER.COMMAND.QUEUE on the Full Repository queue managers.
- The channel to the Full Repository queue manager is started automatically and the message is delivered to the Full Repository. The Full Repository processes the message and stores the subscription request.
- The Full Repository queue manager sends back the information about the queue being opened to the SYSTEM.CLUSTER.COMMAND.QUEUE on the Partial Repository queue manager.
- The message is read from the SYSTEM.CLUSTER.COMMAND.QUEUE by the repository task.
- The information about the queue is stored in the repository cache and hardened to the SYSTEM.CLUSTER.REPOSITORY.QUEUE
- At this point the Partial Repository knows which queue managers host the queue. What it would then need to find out is information on the channels that the hosts of the queue have advertised to the cluster, so that it can create auto-defined cluster sender channels too them. To do this, more subscriptions would be made (if necessary) to the Full Repositories.

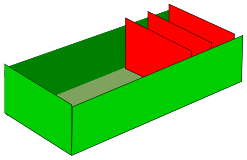
The SYSTEM.CLUSTER queues



- SYSTEM.CLUSTER.COMMAND.QUEUE
  - Holds inbound administrative messages
  - IPPROCS should always be 1
  - CURDEPTH should be zero or decrementing
  - If not, check repository task and error messages



- SYSTEM.CLUSTER.REPOSITORY.QUEUE
  - Holds hardened view of repository cache
  - CURDEPTH should be greater than zero
  - CURDEPTH varies depending on checkpoints. This is not a problem.

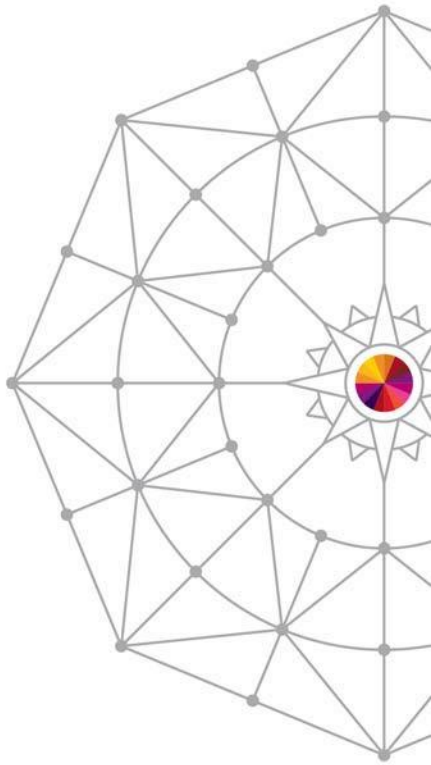


- SYSTEM.CLUSTER.TRANSMIT.QUEUE
  - Holds outbound administrative messages
  - Holds outbound user messages
  - CorrelId in MQMD added on transmission queue will contain the name of the channel that the message should be sent down

The SYSTEM.CLUSTER queues

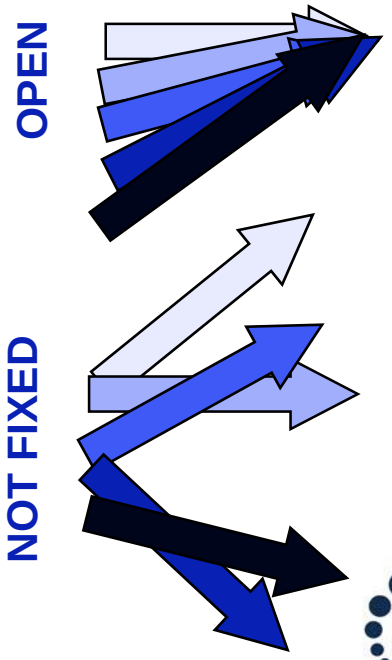
- These queues are for internal use only. It is important that applications do not put or get messages from the SYSTEM.CLUSTER queues.

Workload Balancing



Workload balancing - Bind Options

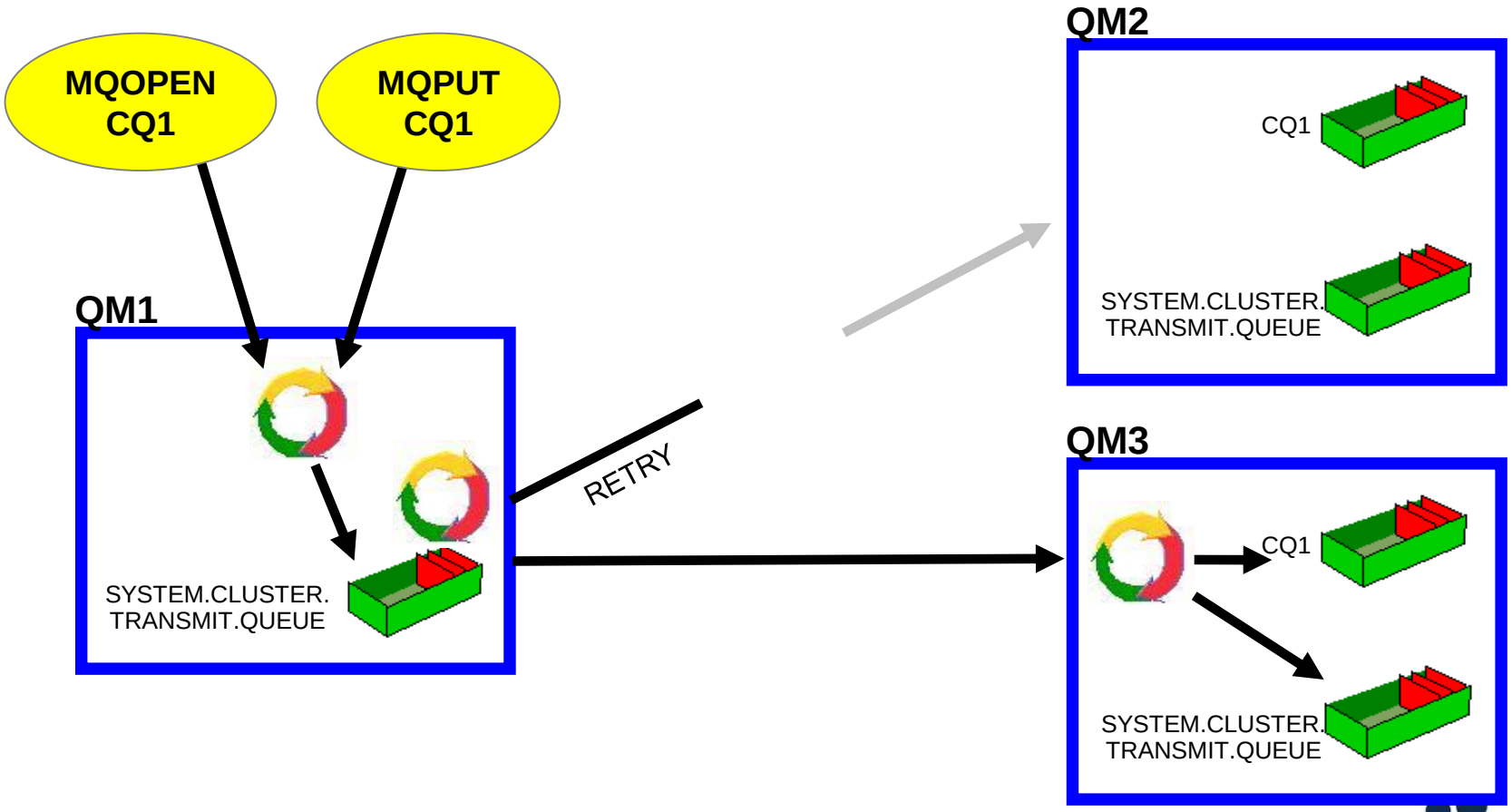
- Bind on open
  - Messages are bound to a destination chosen at MQOPEN
  - All messages put using open handle are bound to same destinat
- Bind not fixed
  - Each message is bound to a destination at MQPUT
  - Workload balancing done on every message
  - Recommended - No affinities are created (SCTQ build up)
- Application options
  - MQOO_BIND_ON_OPEN
  - MQOO_BIND_NOT_FIXED
  - MQOO_BIND_AS_Q_DEF (Default)
- DEFBIND Queue attribute
  - OPEN (Default)
  - NOTFIXED



Workload balancing - Bind Options

- Affinity is the need to continue using the same instance of a service, with multiple messages being sent to the same application process. This is often also known as conversational style applications. Affinity is bad news for parallel applications since it means there is a single point of failure. It can also inhibit scalability (no longer shared nothing, long locks needed). Historically, affinities have caused many problems.
- When writing new applications, one of your design goals should be for no affinities.
- If you cannot get around the need for an affinity in your application, however, MQ will help you.
- If you require all the messages in a conversation to be targeted to the same queue manager, you can request that using the bind options, specifically the “Bind on open” option. Using this option, a destination is chosen at MQOPEN time and all messages put using that open handle are bound to same destination. To choose a new destination, perhaps for the next set of messages, close and reopen the destination in order to re-drive the workload balancing algorithm to choose a new queue manager.
- If you do not have any affinities in your applications, you can use the opposite option, the “Bind not fixed” option.
- This behaviour can either be coded explicitly in your application using the MQOO_BIND_* options or through the default attribute DEFBIND on a queue definition. This attribute defaults to “Bind on open” to ensure applications not written with parallelism in mind still work correctly in a cluster.

When does workload balancing occur?



When does workload balancing occur?

- In a typical setup, where a message is put to a queue manager in the cluster, destined for a backend server queue manager also in the cluster, there are three places at which workload balancing may occur.
- The first of these is at either MQOPEN or MQPUT time depending on the bind options associated with message affinity. If bind_on_open is specified, then this means all messages will go to the same destination, so once the destination is chosen at MQOPEN time, no more workload balancing will occur on the message. If bind_not_fixed is specified, the messages can be sent to any of the available destinations. The decision where to send the message is made at MQPUT time, however this may change whilst the message is in transit.
- If a channel from a queue manager to a backend server goes into retry, then any bind_not_fixed messages waiting to be sent down that channel will go through workload balancing again to see if there is a different destination that is available.
- Once a message is sent down a channel, at the far end the channel code issues an MQPUT to put the message to the target queue and the workload algorithm will be called.

Workload management features

QUEUES

- Put allowed
  - **PUT(ENABLED/DISABLED)**
- Utilising remote destinations
  - **CLWLUSEQ**
- Queue rank
  - **CLWLRANK**
- Queue priority
  - **CLWLPRTY**

QUEUE MANAGER

- Utilising remote destinations
  - **CLWLUSEQ**
- Availability status
  - **SUSPEND/RESUME**
- Most recently used
  - **CLWLMRUC**

CHANNELS

- Channel status
  - **INACTIVE, RUNNING**
  - **BINDING, INITIALIZING, STARTING, STOPPING**
  - **RETRYING**
  - **REQUESTING, PAUSED STOPPED**
- Channel network priority
  - **NETPRTY**
- Channel rank
  - **CLWLRANK**
- Channel priority
  - **CLWLPRTY**
- Channel weighting
  - **CLWLWGHT**

EXIT

- A cluster workload exit can be used

Workload management features

- There are a number of features in WebSphere MQ which affect the way the default workload balancing algorithm works. With all these attributes set to their default values, the workload balancing algorithm could be described as “round robin, excluding failed servers”. More control is required in some scenarios to allow more flexible interconnected cluster, or to allow greater scalability, and some of these attributes will then need to be altered to achieve the behaviour required.

- Only some attributes on a clustered queue or channel have an impact on the workload balancing algorithm. In other words, not all attributes are propagated around the cluster. Those that have an impact are shown on this page.

- If a queue is altered to be disabled for put, this change is propagated around the cluster and is used by other queue managers in the cluster when choosing which queues to send their messages to. A queue that is disabled for put is not a good choice!

- A queue manager may be suspended from the cluster. This is another way of removing queues on that queue manager from being chosen by the workload balancing algorithm. Note, that if all queue managers in a cluster are suspended, then they are all equally ‘good’ choices and the effects will be nullified.

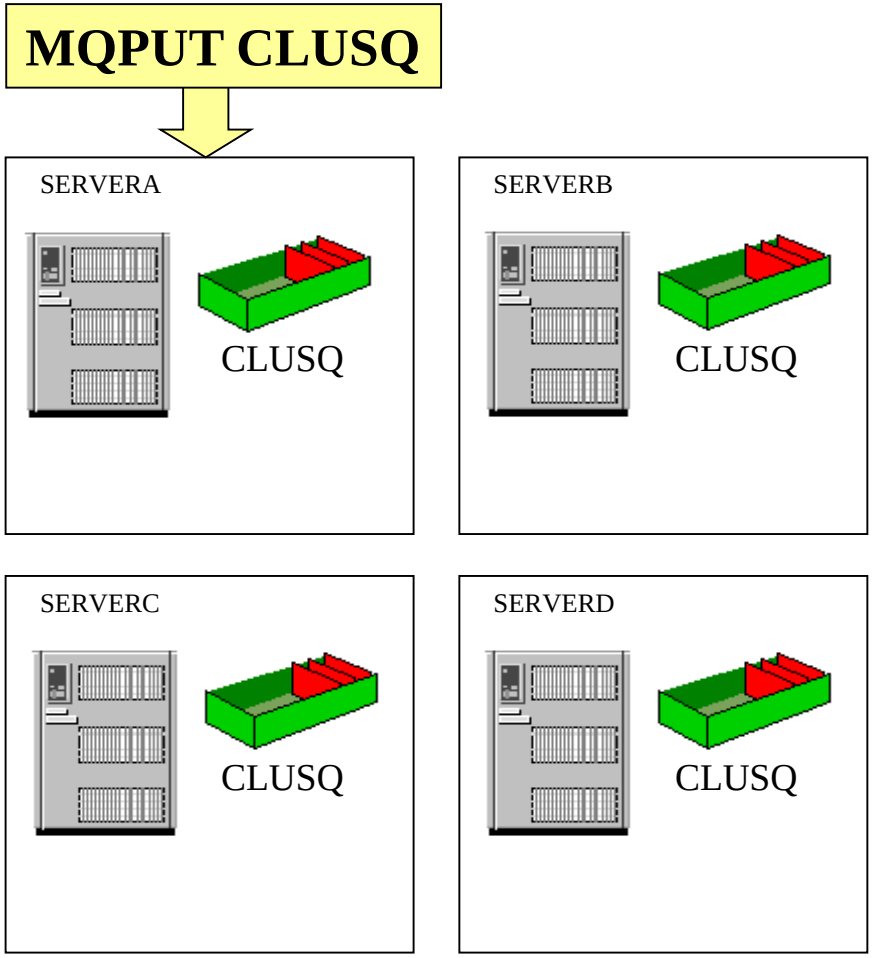
Workload management features

- Channel status plays a part in the workload balancing algorithm, by indicating when a server has failed. ‘Bad’ channel status suggests a problem and makes that queue manager a worse choice. If the entire network is having a problem and all channels are in RETRY state, then all queue managers are equally ‘good’ choices and the effects again will be nullified – where-ever the messages are targeted, they will have to wait on the transmission queue for the network to come back up.

- Network priority provides a way of choosing between two routes to the same queue manager. For example, a TCP route and a SNA route, or an SSL route and a non-SSL route. The higher network priority route is always used when it’s channel is in a good state.

- All of the choices made by the queue manager using it’s default workload balancing algorithm can be over-ridden by the use of a cluster workload exit. This exit is provided with all the same information that the queue manager uses to make its choices, and is also told which choice the default algorithm made, and can then change that choice if it wishes. It can make decisions based on message content, or other information that the queue manager is not aware, such as business routing decisions.

Utilising remote destinations with CLWLUSEQ

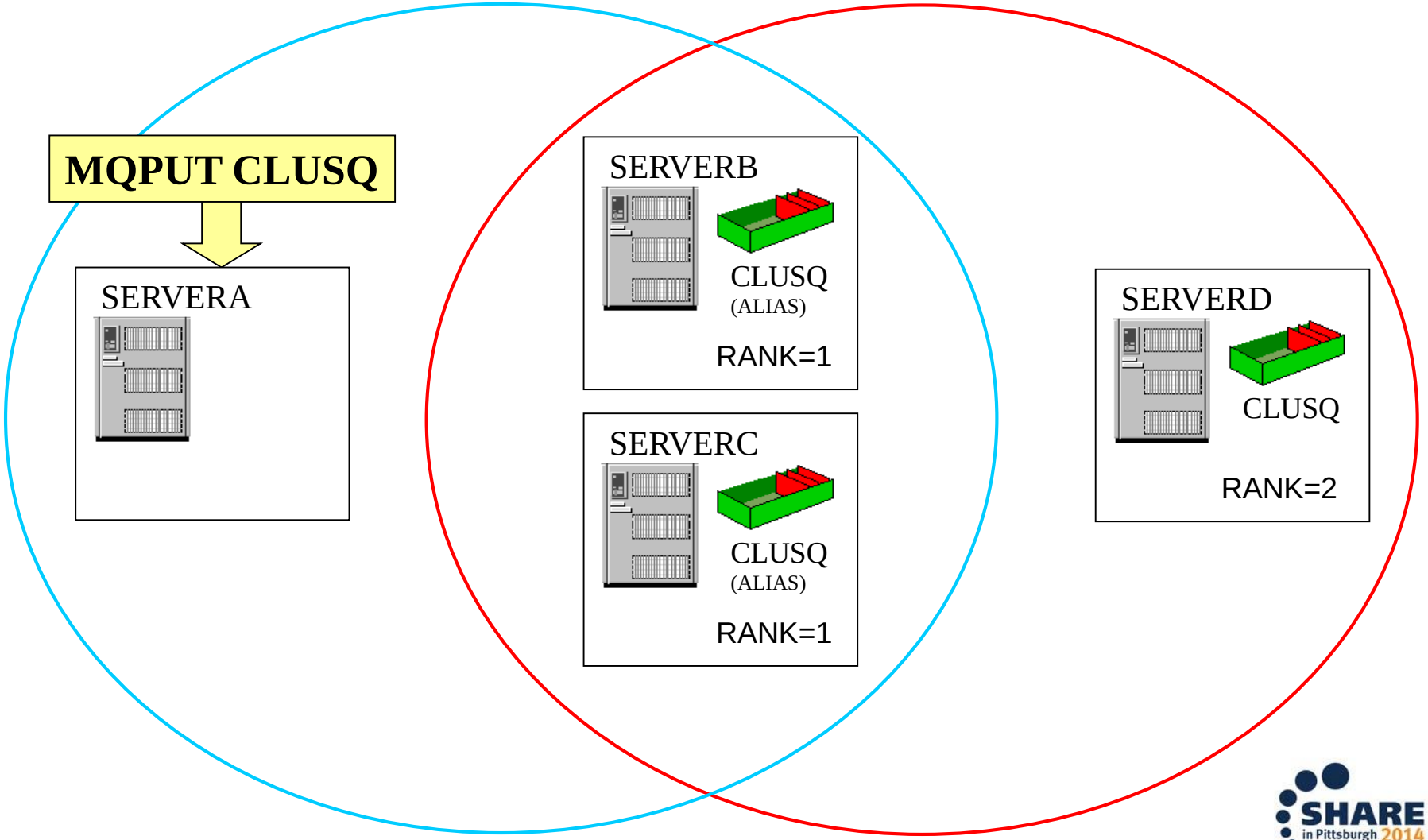


- Allows remote queues to be chosen, when a local queue exists
- MQSC
- DEFINE QL(CLUSQ) CLUSTER(IVANS) CLWLUSEQ( )
  - **LOCAL = If a local queue exists, choose it.**
  - **ANY = Choose either local or remote queues.**
  - **QMGR = Use the queue manager's use-queue value.**
- ALTER QMGR CLWLUSEQ( )
  - **LOCAL = If the queue specifies CLWLUSEQ(QMGR) and a local queue exists, choose it.**
  - **ANY = If the queue specifies CLWLUSEQ(QMGR) choose either local or remote queues.**
- Messages received by a cluster channel must be put to a local queue if one exists

Utilising remote destinations

- Prior to WebSphere MQ V6, if a local instance of the named cluster queue existed, it was always utilised in favour of any remote instances. This behaviour can be over-ridden at V6 by means of the CLWLUSEQ attribute, which allows remote queues to be chosen, when a local queue exists.
- The CLWLUSEQ attribute is available on queue definitions, to allow only specific named queues to utilise this over-ride, or on the queue manager to over-ride this behaviour for all cluster queues.
- DEFINE QL(CLUSQ) CLUSTER(IVANS) CLWLUSEQ( )
  - **LOCAL If a local queue exists, choose it.**
  - **ANY Choose either local or remote queues.**
  - **QMGR Use the queue manager's use-queue value.**
- ALTER QMGR CLWLUSEQ( )
  - **LOCAL If the queue specifies CLWLUSEQ(QMGR) and a local queue exists, choose it.**
  - **ANY If the queue specifies CLWLUSEQ(QMGR) choose either local or remote queues.**
- Any messages received by a cluster channel will be put to a local queue if one exists in order to avoid looping round the cluster.

Rank and Overlapping Clusters



Rank and Overlapping Clusters

- Rank allows greater control of the routing of messages through overlapping clusters.

- Without using rank, messages put to CLUSQ by applications connected to SERVERA could bounce between the alias queues on SERVERB and SERVERC. Using rank solves this “bouncing” problem because once messages arrive on SERVERB or SERVERC, the CLUSQ instance on SERVERD is available to the cluster workload management algorithm. As CLUSQ on SERVERD is ranked higher it will be chosen.

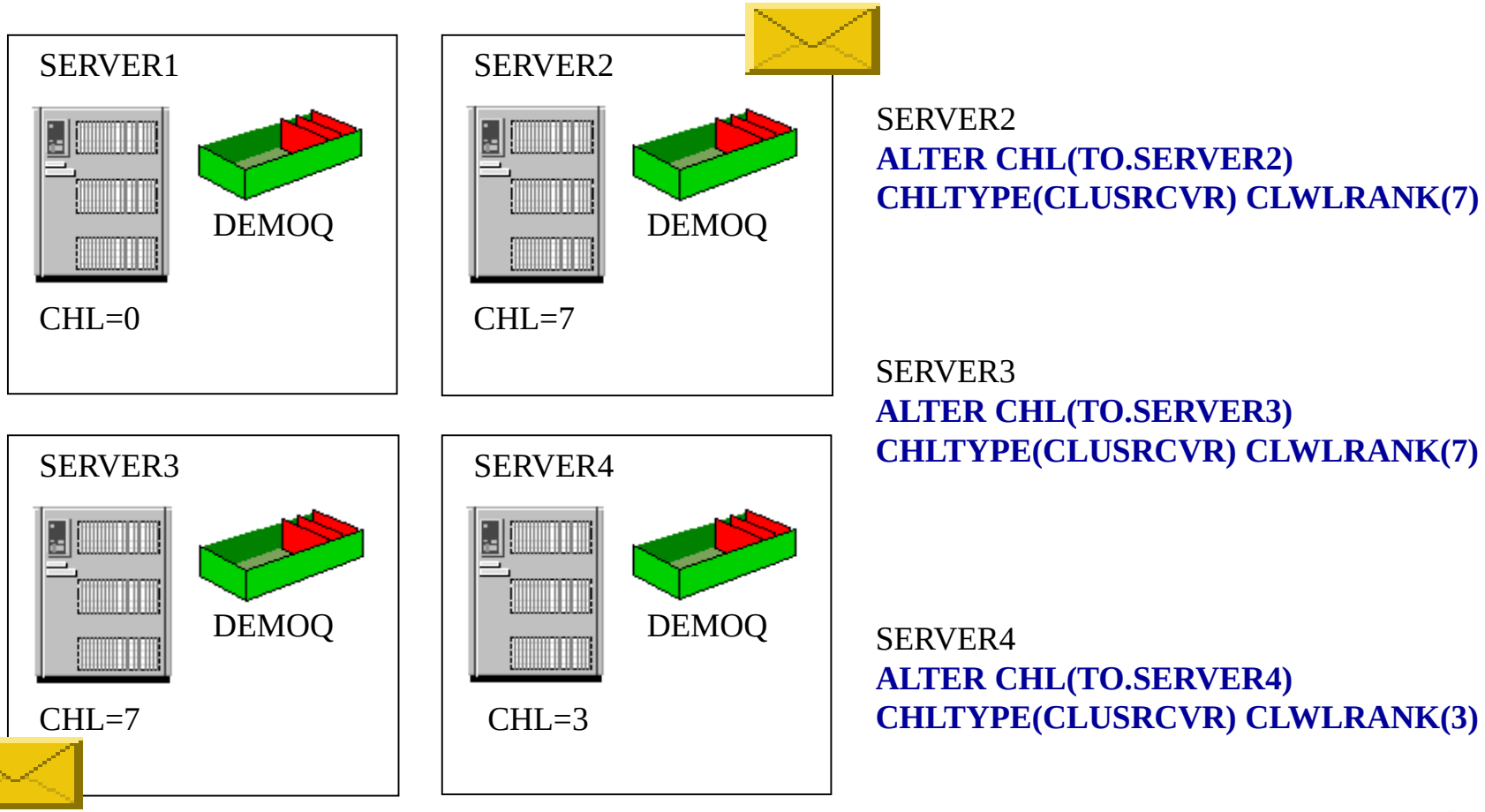
Rank - basics

- Channels and queues with the highest rank are chosen preferentially over those with lower ranks.

- MQSC
  - DEFINE CHL(TO.ME) CHLTYPE(CLUSRCVR) CLUSTER(IVANS) … CLWLRANK()
    - Range 0 – 9
    - Default 0
  - DEFINE QL(CLUSQ) CLUSTER(IVANS) CLWLRANK( )
    - Range 0 – 9
    - Default 0

- Channel rank checked before queue rank

Channel Rank example



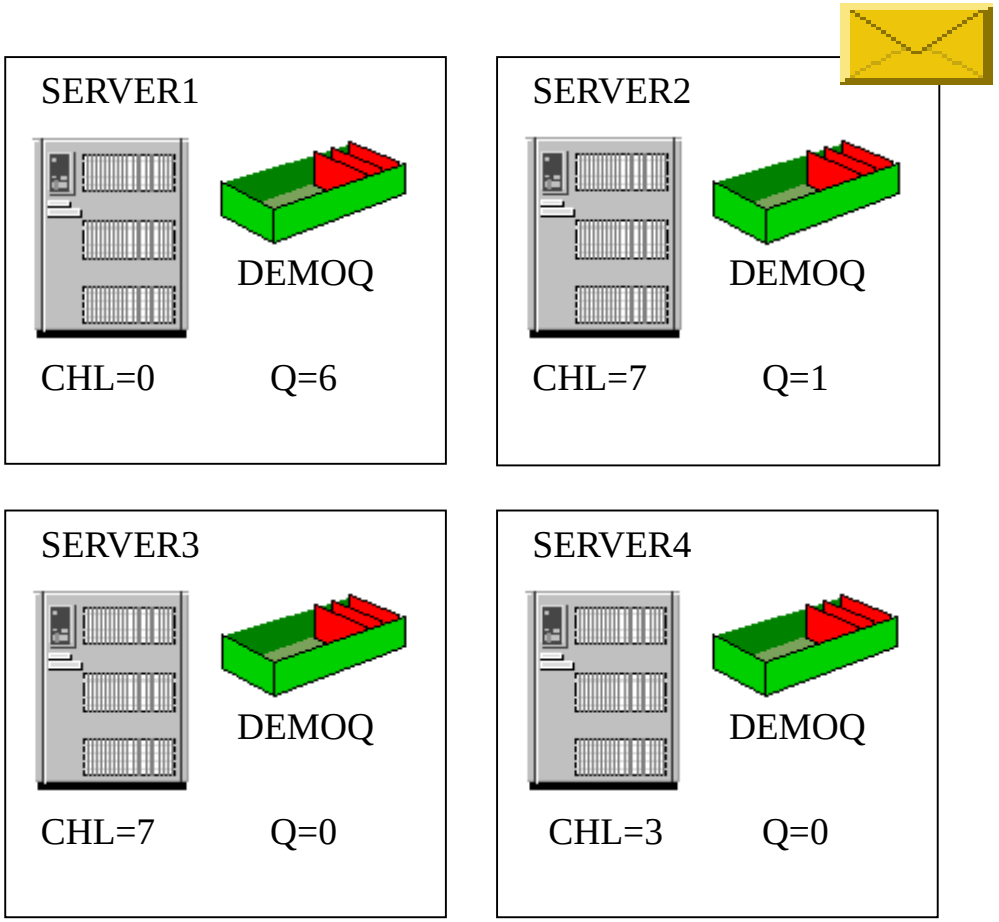
- Messages are equally distributed between the channels with the highest rank

Channel Rank example

- The channel rank for SERVER2 and SERVER3 are set higher than SERVER4. As the default channel rank is zero, SERVER1 has the lowest rank.

- Once the ranks for channels on SERVER2, SERVER3 and SERVER4 have been altered the messages are distributed equally between the highest ranked destinations (SERVER2 and SERVER3).

Queue Rank example



SERVER1
ALTER QL(DEMOQ)
CLWLRANK(6)

SERVER2
ALTER QL(DEMOQ)
CLWLRANK(1)

- Messages are equally distributed between the queues with the highest rank, channel rank has priority

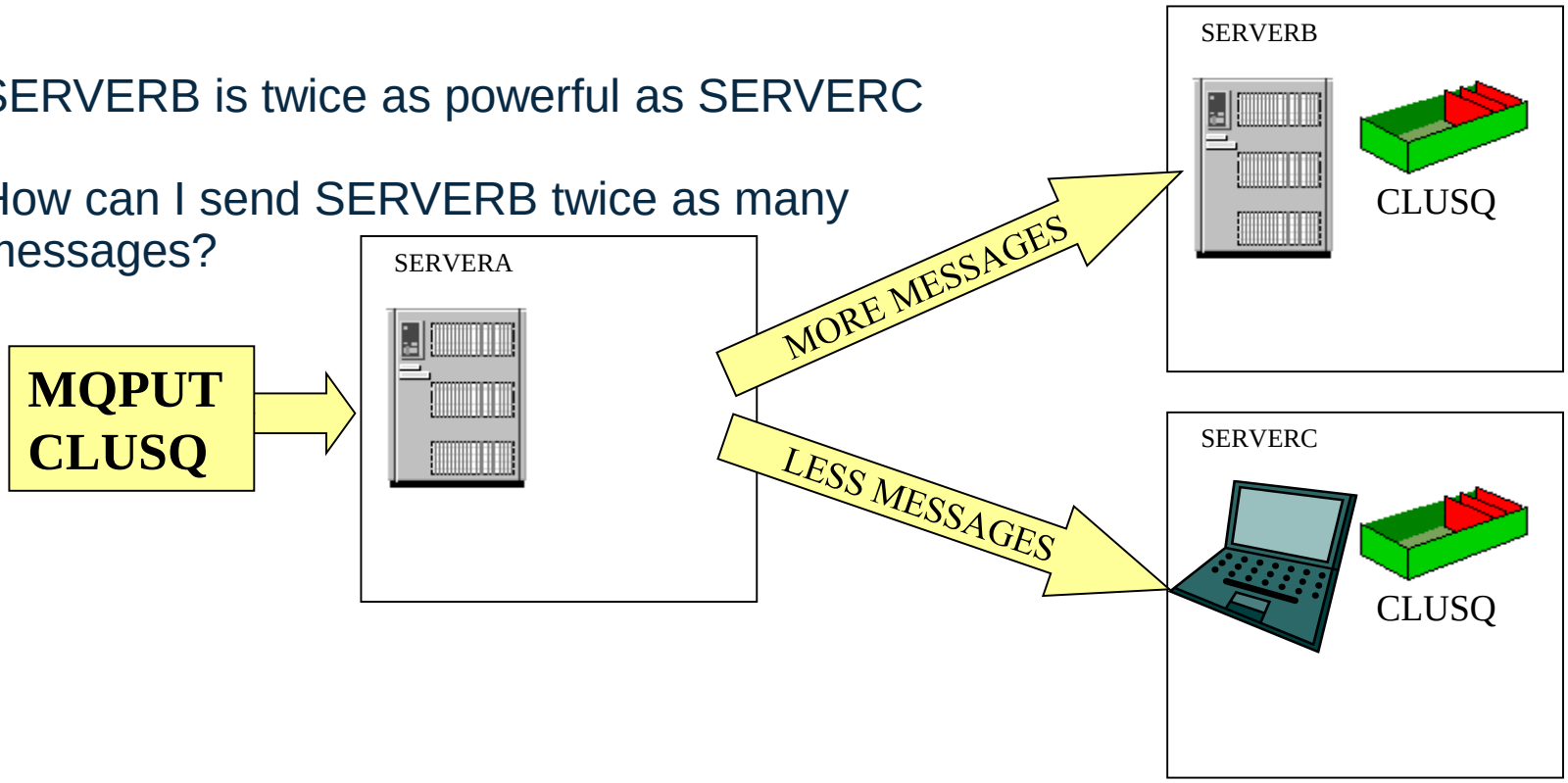
Queue Rank example

- Once the ranks for queues on SERVER1 and SERVER2 have been changed the all messages are delivered to SERVER2. This is because the cluster workload management algorithm checks channel ranks before queue rank. The channel rank check leaves SERVER2 and SERVER3 as valid destinations and because the queue rank for DEMOQ on SERVER2 is higher than that on SERVER3 the messages are delivered to SERVER2. Note that as channel rank is more powerful than queue rank, the highest ranked queue (on SERVER1) is not chosen.

- It is important to note that destinations with the highest rank will be chosen regardless of the channel status to that destination. This could lead to messages building up on the SYSTEM.CLUSTER.TRANSMIT.QUEUE.

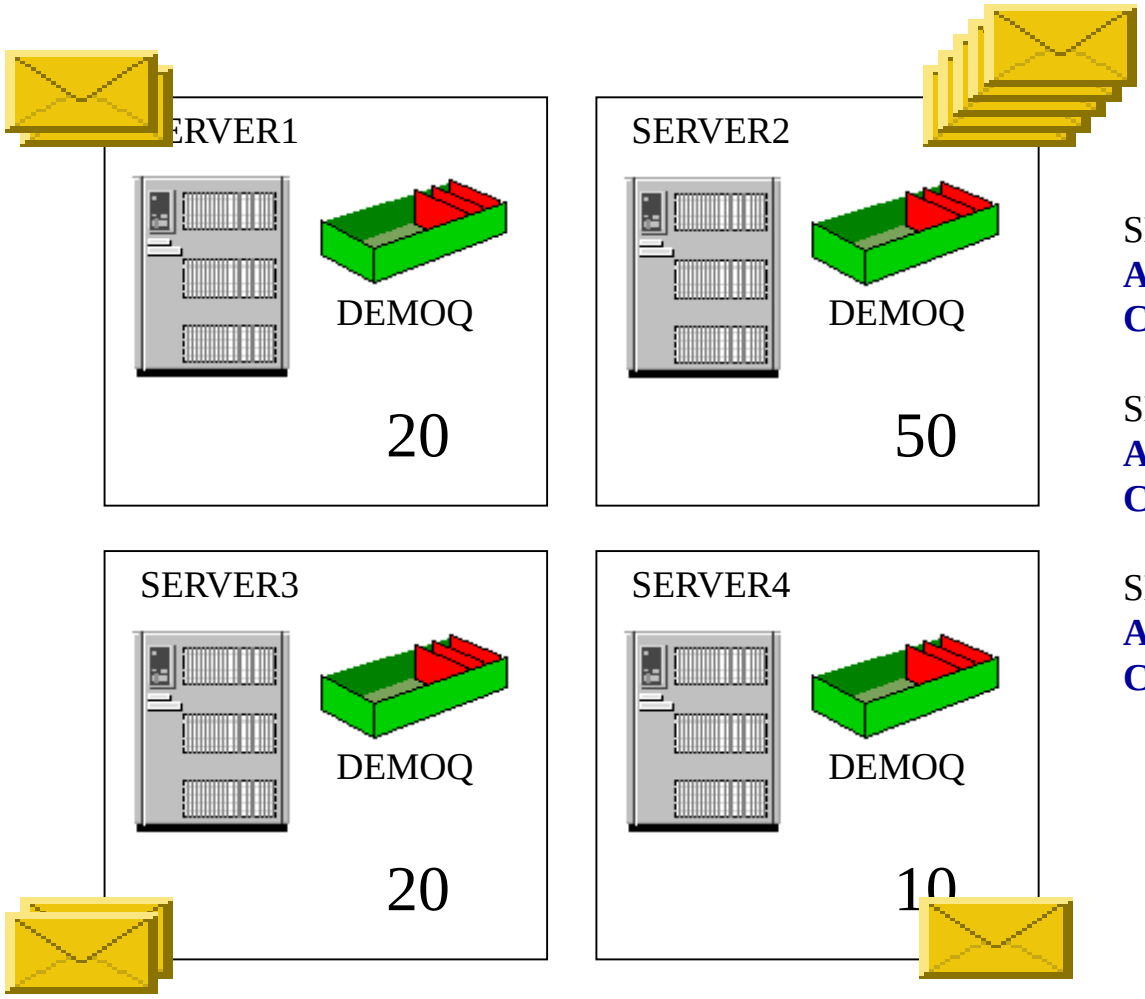
Utilising processing power

- SERVERB is twice as powerful as SERVERC
- How can I send SERVERB twice as many messages?



- If more than one destination is valid, the round robin algorithm sends messages in numbers proportional to channel weights
- MQSC

Channel Weight example



SERVER1
ALTER CHL(TO.SERVER2)
CHLTYPE(CLUSRCVR) CLWLWGHT(20)

SERVER3
ALTER CHL(TO.SERVER3)
CHLTYPE(CLUSRCVR) CLWLWGHT(20)

SERVER4
ALTER CHL(TO.SERVER4)
CHLTYPE(CLUSRCVR) CLWLWGHT(10)

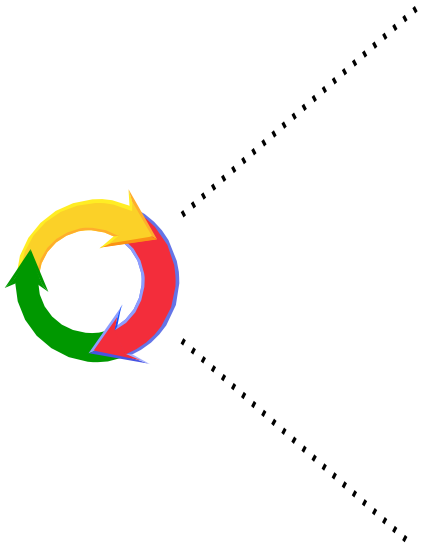
Channel Weight example

- In this example, the approximate percentage of messages distributed to each queue manager are as follows…

- SERVER1 20%
- SERVER2 50%
- SERVER3 20%
- SERVER4 10%

- Weight enables the cluster workload management algorithm to favour more powerful machines

Cluster Workload Algorithm



- Queue PUT(ENABLED/DISABLED)
- Local instance (CLWLUSEQ)
- Channel rank (CLWLRANK)
- Queue rank (CLWLRANK)
- Channel Status
  - INACTIVE, RUNNING
  - BINDING, INITIALIZING, STARTING, STOPPING
  - RETRYING
  - REQUESTING, PAUSED, STOPPED
- Channel net priority (NETPRTY)
- Channel priority (CLWLPRTY)
- Queue priority (CLWLPRTY)
- Most recently used (CLWLMRUC)
- Least recently used with channel weighting (CLWLWGHT)

Cluster Workload Algorithm

- The full algorithm (taken from the Queue Manager Clusters manual) is as follows…

- 1. If a queue name is specified, queues that are not PUT enabled are eliminated as possible destinations. Remote instances of queues that do not share a cluster with the local queue manager are then eliminated. Next, remote CLUSRCVR channels that are not in the same cluster as the queue are eliminated.

- 2. If a queue manager name is specified, queue manager aliases that are not PUT enabled are eliminated. Remote CLUSRCVR channels that are not in the same cluster as the local queue manager are then eliminated.

- 3. If the result above contains the local instance of the queue, and the use-queue attribute of the queue is set to local (CLWLUSEQ(LOCAL)), or the use-queue attribute of the queue is set to queue manager (CLWLUSEQ(QMGR)) and the use-queue attribute of the queue manager is set to local (CLWLUSEQ(LOCAL)), the queue is chosen; otherwise a local queue is chosen if the message was not put locally (that is, the message was received over a cluster channel). User exits are able to detect this using the MQWXP.Flags flag MQWXP_PUT_BY_CLUSTER_CHL and MQWQR.QFlags flag MQQF_CLWL_USEQ_ANY not set.

- 4. If the message is a cluster PCF message, any queue manager to which a publication or subscription has already been sent is eliminated.

- 4a. All channels to queue managers or queue manager alias with a rank (CLWLRANK) less than the maximum rank of all remaining channels or queue manager aliases are eliminated.

Cluster Workload Algorithm (cont)

- 5. If only remote instances of a queue remain, resumed queue managers are chosen in preference to suspended ones.
- 6. If more than one remote instance of a queue remains, all channels that are inactive (MQCHS_INACTIVE) or running (MQCHS_RUNNING) are included.
- 7. If no remote instance of a queue remains, all channels that are in binding, initializing, starting or stopping state (MQCHS_BINDING, MQCHS_INITIALIZING, MQCHS_STARTING, or MQCHS_STOPPING) are included.
- 8. If no remote instance of a queue remains, all channels in retrying state (MQCHS_RETRYING) are included.
- 9. If no remote instance of a queue remains, all channels in requesting, paused or stopped state (MQCHS_REQUESTING, MQCHS_PAUSED and MQCHS_STOPPED) are included.
- 10. If more than one remote instance of a queue remains and the message is a cluster PCF message, locally defined CLUSSDR channels are chosen.
- 11. If more than one remote instance of a queue remains to any queue manager, channels with the highest NETPRTY value for each queue manager are chosen.
- 11a. If a queue manager is being chosen: all remaining channels and queue manager aliases other than those with the highest priority (CLWLPRTY) are eliminated. If any queue manager aliases remain, channels to the queue manager are kept.

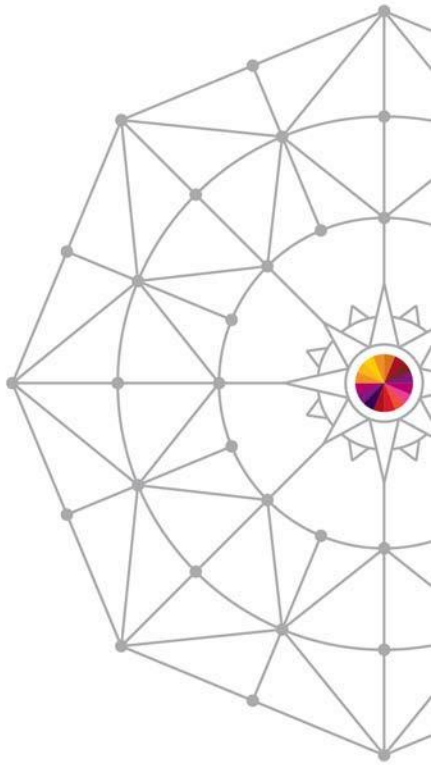
Cluster Workload Algorithm (cont)

- 11c. All channels except a number of channels with the highest values in MQWDR.DestSeqNumber are eliminated. If this number is greater than the maximum allowed number of most-recently-used channels (CLWLMRUC), the least recently used channels are eliminated until the number of remaining channels is no greater than CLWLMRUC.

- 12. If more than one remote instance of a queue remains, the least recently used channel is chosen (that is, the one with the lowest value in MQWDR.DestSeqFactor). If there is more than one with the lowest value, one of those with the lowest value in MQWDR.DestSeqNumber is chosen. The destination sequence factor of the choice is increased by the queue manager, by approximately 1000/(Channel weight) (CLWLWGHT). The destination sequence factors of all destinations are reset to zero if the cluster workload attributes of available destinations are altered, or if new cluster destinations become available. Also, the destination sequence number of the choice is set to the destination sequence number of the previous choice plus one, by the queue manager.

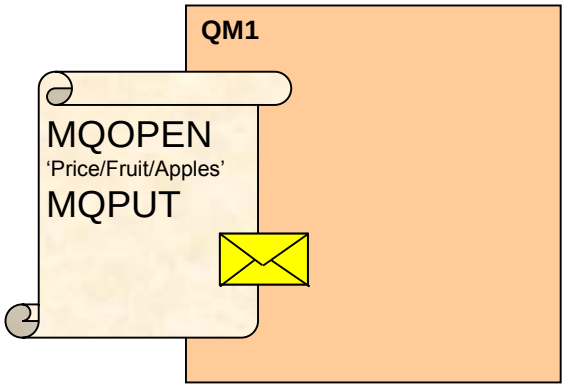
- Note that the distribution of user messages is not always exact, because administration and maintenance of the cluster causes messages to flow across channels. This can result in an apparent uneven distribution of user messages which can take some time to stabilize. Because of this, no reliance should be made on the exact distribution of messages during workload balancing.

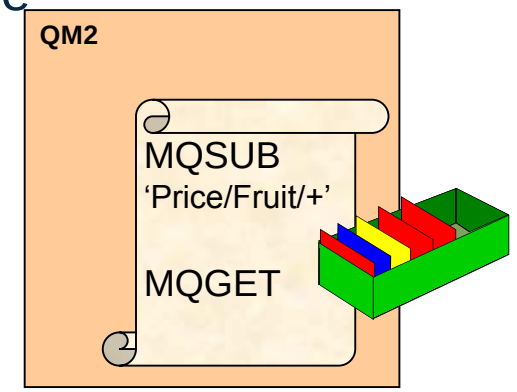
Pub/Sub Clusters

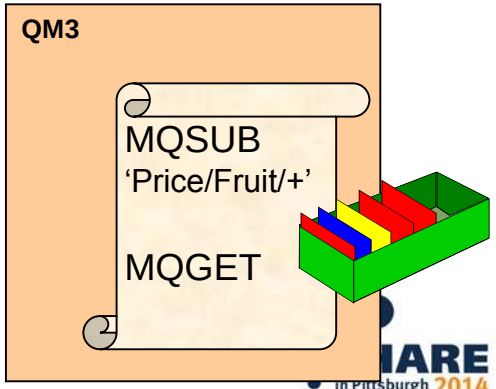


Publish/Subscribe Topologies

- Local Queuing -> Distributed Queuing
- Publish/Subscribe -> Distributed Publish/Subscribe
- Application API calls remain the same
- Administration changes have the effect



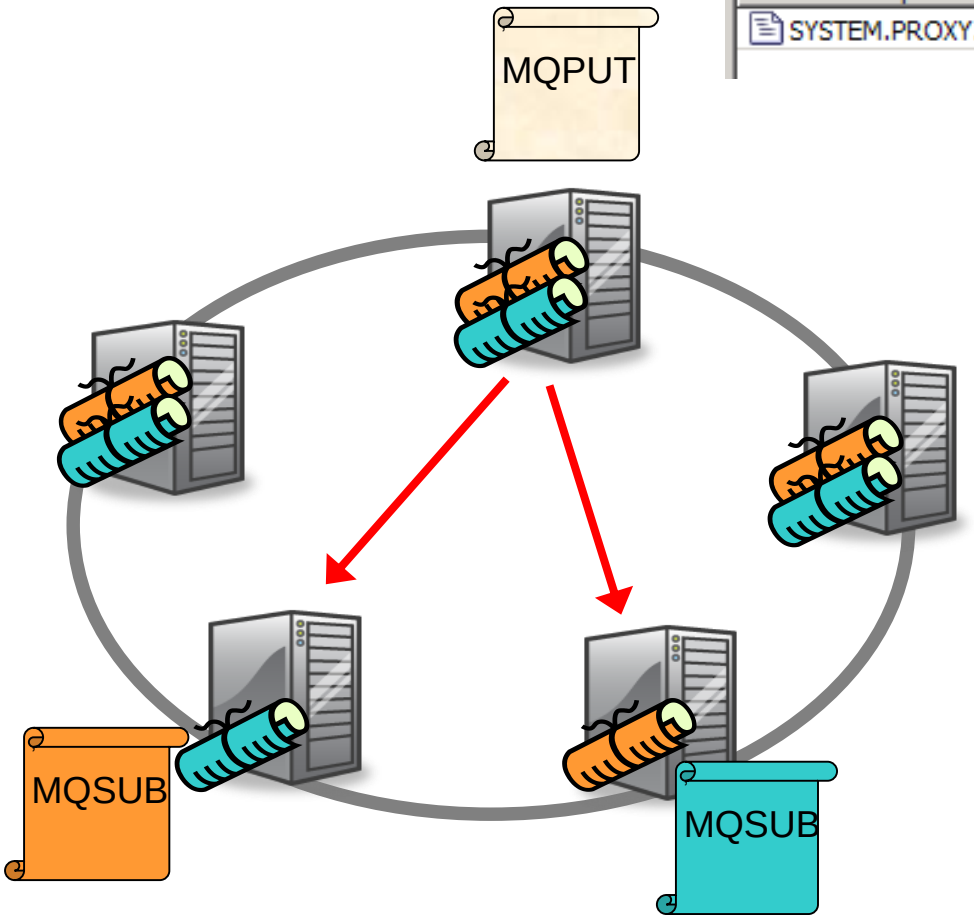




Publish/Subscribe Topologies

- Just as queuing can be done on a single queue manager or can be done by moving messages between multiple queue managers – known as distributed queuing – so can publish/subscribe. We call this distributed publish/subscribe.
- This is the concept (and the features to implement it) where an application may be publishing to a topic on QM1 and other applications may be subscribing on that topic on others queue managers, here QM2 and QM3, and the publication message needs to flow to those other queue managers to satisfy those subscribers.
- The application code stays the same, you still call MQSUB or MQOPEN and MQPUT, the difference, as with distributed queuing is in the administrative set-up of your queue managers.
- We are going to introduce the way clustering can be used to set up your queue managers to publish messages to another queue manager.

Pub/Sub Clusters



Subscription name	Topic string	Type
SYSTEM.PROXY.QM2 DEMO Price/Fruit/Apples	Price/Fruit/Apples	Proxy

TOPIC attributes

CLUSTER
SUBSCOPE
PUBSCOPE
PROXYSUB

```
Command Prompt - runmqsc TEST1

Starting MQSC for queue manager TEST1.

DEFINE TOPIC(APPLES)
       TOPICSTR('Price/Fruit/Apples')
       CLUSTER(DEMO)

DISPLAY SUB(*) SUBTYPE(PROXY) ALL
    1 : DISPLAY SUB(*) SUBTYPE(PROXY) ALL
AMQ8096: WebSphere MQ subscription inquired.
   SUBID(414D5120514D312020202020202020204F57864820000F02)
   SUB(SYSTEM.PROXY.QM2 DEMO Price/Fruit/Apples)
   TOPICSTR(Price/Fruit/Apples)              TOPICOBJ( )
   DEST(SYSTEM.INTER.QMGR.PUBS)              DESTQMGR(QM2)
   DESTCLAS(PROVIDED)                        DURABLE(YES)
   SUBSCOPE(ALL)                             SUBTYPE(PROXY)
```

Pub/Sub Clusters

- A pub/sub cluster is a cluster of queue managers, with the usual CLUSRCVR and CLUSSDR definitions, but that also contains a TOPIC object that has been defined in the cluster.
- With a cluster you have “any-to-any” connectivity. There are direct links between all queue managers in the cluster. This provides good availability for the delivery of messages, if one route is unavailable, there may well be another route to deliver the messages to the target subscription.
- With a TOPIC object defined in the cluster, an application connected to one queue manager in the cluster can subscribe to that topic or any node in the topic tree below that topic and receive publications on that topic from other queue managers in the cluster.
- This is achieved by the creation of proxy subscriptions on the queue managers in the cluster, so that when a publication to the topic in question happens on their queue manager, they know to forward it to the appropriate other members of the cluster.
- You can view these proxy subscriptions through the same commands we saw earlier. By default proxy subscriptions are not shown to you because the default value for SUBTYPE is USER. If you use SUBTYPE(ALL) or SUBTYPE(PROXY) you will see these subscriptions.
- There are a few attributes that are specifically related to Distributed Publish/Subscribe. PUBSCOPE and SUBSCOPE determine whether this queue manager propagates publications to queue managers in the topology (pub/sub cluster or hierarchy) or restricts the scope to just its local queue manager. You can do the equivalent job programmatically using MQPMO_SCOPE_QMGR / MQSO_SCOPE_QMGR.
- PROXYSUB is an attribute that controls when proxy subscriptions are made. By default it has value FIRSTUSE and thus proxy subscriptions are only created when a user subscription is made to the topic. Alternatively you can have the value FORCE which means proxy subscriptions are made even when no local user subscriptions exist.

Cluster Topics

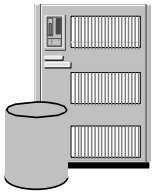
- Required for publishing to remote Pub/Sub Cluster QMs
- Pub/Sub Cluster exists when there are one or more cluster topics
  - Cluster topic is pushed to ALL queue managers in the cluster
  - Channels are automatically defined between ALL queue managers in the cluster
- Not to be confused with a subscriber
- Define
  - DEFINE TOPIC(FOOTBALL) TOPICSTR(/global/sports/football) CLUSTER(SPORTS)
- Display
  - DISPLAY TOPIC(FOOTBALL)
  - DISPLAY TOPIC(FOOTBALL) TYPE(LOCAL)
  - DISPLAY TOPIC(FOOTBALL) TYPE(ALL)
    - Local objects only
  - DISPLAY TOPIC(FOOTBALL) TYPE(CLUSTER)
  - DISPLAY TOPIC(FOOTBALL) CLUSINFO
  - DISPLAY TCLUSTER(FOOTBALL)
    - Cluster objects only
  - DISPLAY TOPIC(FOOTBALL) TYPE(ALL) CLUSINFO
    - Both local and cluster objects

Cluster Topics

- Cluster TOPICS are regular TOPIC objects that are advertised to the cluster. When a cluster TOPIC is defined, the cluster in which it is defined becomes a Pub/sub Cluster.
- Like traditional clusters, Pub/sub clusters are designed for many-many queue manager connectivity. In traditional clusters, cluster objects are automatically defined based on usage (e.g. put to a queue). The usage model is based on a putter using a set of cluster queues (not necessarily defined on all queue managers in the cluster). Therefore in traditional clusters it is unlikely that all queue managers will actually be connected to all other queue managers by auto-defined channels.
- In Pub/sub clusters, cluster objects are automatically defined before usage, at the time the first cluster TOPIC is defined in the cluster. This is because the usage model is different to that of traditional clusters. Channels are required from any queue manager to which a subscriber (for a cluster topic) is connected to all other queue managers so that a proxy-subscription can be fanned out to all the queue managers. Channels are also required back from any queue manager where a publisher (for a cluster topic) is connected to those queue managers which have a connected subscriber. Therefore in Pub/sub Clusters it is much more likely that all queue managers will actually be connected to all other queue managers by auto-defined channels. It is for this reason that, in Pub/sub clusters, cluster objects are automatically defined before usage.
- To define a cluster TOPIC, you must provide a topic object name, topic string, and cluster name. The queue manager on which the topic is defined should be a member of the specified cluster.
- When displaying cluster topics, there are two types:
  - Local: Directly administrable objects.
  - Cluster: Cluster cache records, based upon local objects.

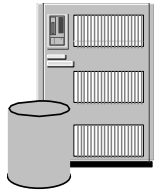
Cluster Topic Hosts

- Cluster Topics can be defined on any queue manager in the cluster
  - Full or Partial Repositories



- Cluster topics can be defined on more than one queue manager
  - E.g.
    - QM1 - DEF TOPIC(SPORTS) TOPICSTR(/global/sports) CLUSTER(DEMO)
    - QM2 - DEF TOPIC(SPORTS) TOPICSTR(/global/sports) CLUSTER(DEMO)

  - Multiple definitions ok
    - But not necessarily concurrent
      - If topic host is lost, cluster topic will remain usable for up to 30 days
      - RESET CLUSTER the lost queue manager



  - … but definitions should be identical
    - Behaviour is undefined where attributes conflict
    - Conflict reported
      - AMQ9465 / AMQ9466

Cluster Topic Hosts

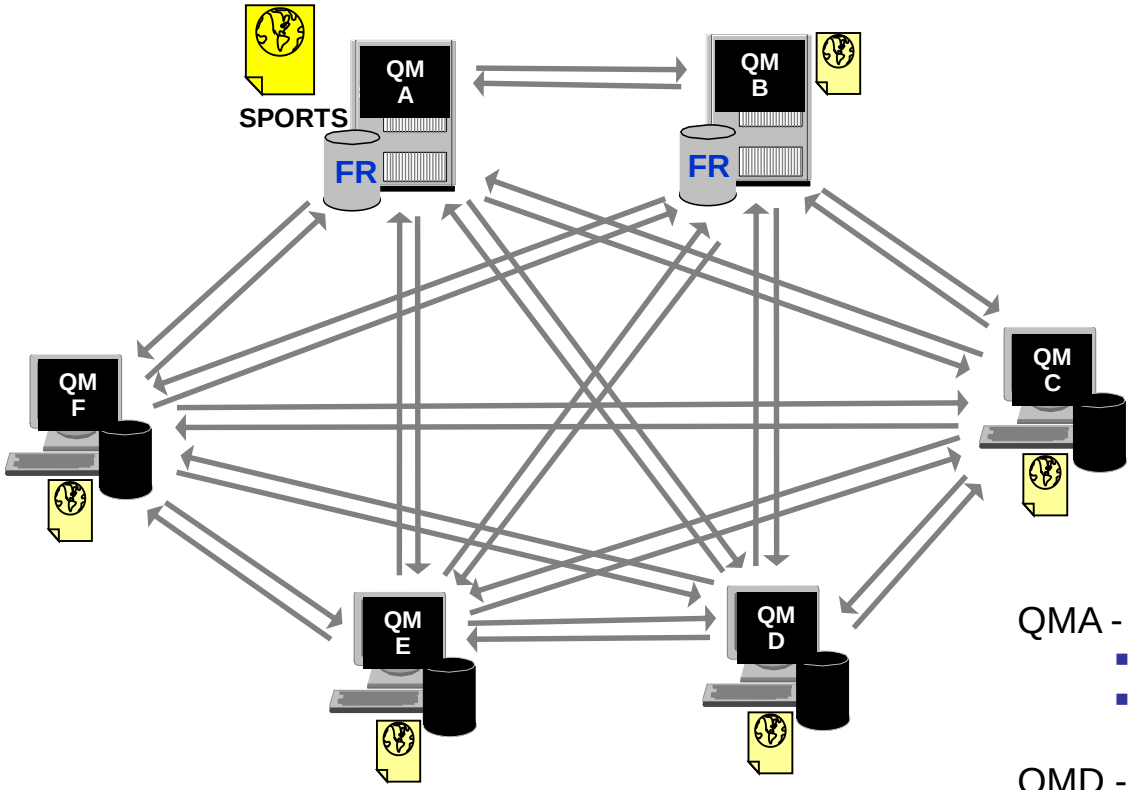
- A cluster topic host is a queue manager where a clustered TOPIC object is defined. Clustered TOPIC objects can be defined on any queue manager in the pub/sub cluster. When at least one clustered topic exists within a cluster the cluster is a pub/sub cluster. It is recommended that all clustered TOPIC objects be identically defined on two queue managers and that these machines be highly available. If a single host of a clustered TOPIC object is lost (e.g. due to disk failure), any cluster topic cache records based on the clustered topic object that already exist in the cluster cache on other queue managers, will be usable within the cluster for a period of up to 30 days (or until the cache is refreshed). The clustered TOPIC object can be redefined on a healthy queue manager. If a new object is not defined, up to 27 days after the host queue manager failure, all members of the cluster will report that an expected object update has not been received (AMQ9465 / AMQ9466 or CSQX465I / CSQX466I).

- There is no requirement that full repositories and topic hosts overlap, or indeed that they are separated. In pub/sub clusters that have just two highly available machines amongst many machines, it would be recommended to define both the highly available machines as full repositories and cluster topic hosts. In pub/sub clusters with many highly available machines it would be recommended to define full repositories, and cluster topic hosts on separate highly available machines, so that the operation and maintenance of one function can be managed without affecting the operation of other functions.

- Although there is nothing wrong with having multiple identical definitions for a topic within a cluster, it adds administration overhead when changing, so is probably a bad idea. Definitions must be kept consistent or it could result in unpredictable behavior (as indicated by warning messages in logs).

Pub/Sub Cluster Architecture

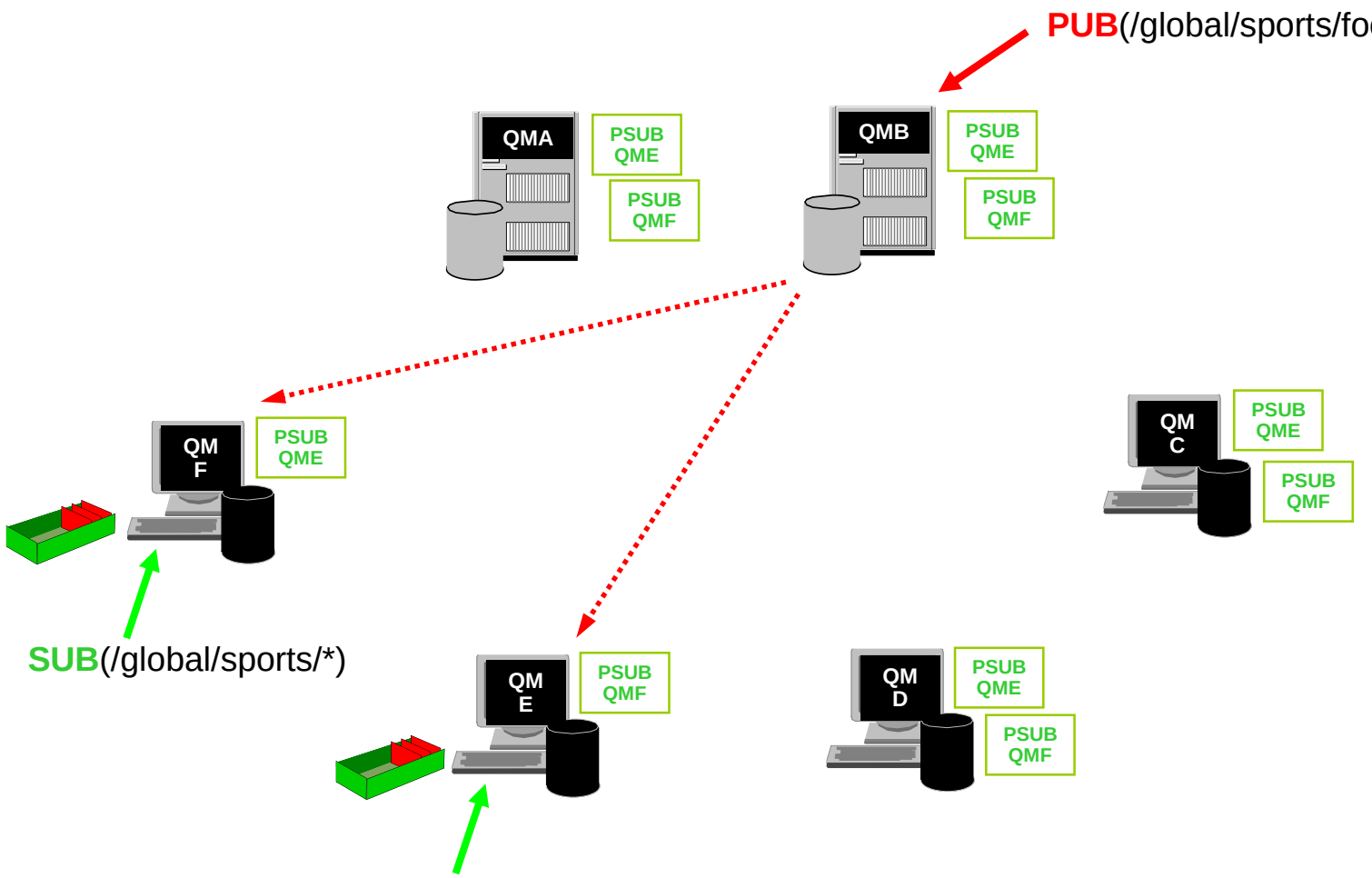
QMA - **DEFINE TOPIC(SPORTS) TOPICSTR(/global/sports) CLUSTER(DEMO)**



QMA - **DIS TOPIC(SPORTS) TYPE(ALL)**
- Local topic
- Cluster topic

QMD - **DIS TOPIC(SPORTS) TYPE(ALL)**
- Cluster topic

Pub/Sub Cluster Architecture 2



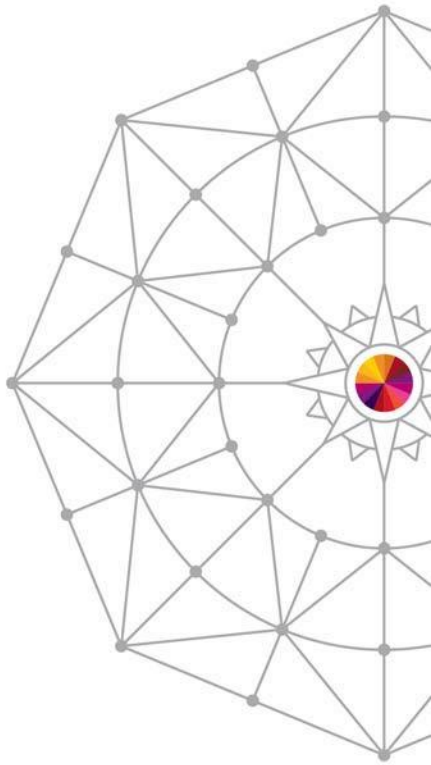
Pub/Sub Cluster Architecture 2

- We start this foil assuming that Pub/sub Cluster with cluster topic with topic string /global/sports already exists.

- The proxy-subscriptions are fanned out from the queue manager to which the subscriber is connected to all other queue managers.

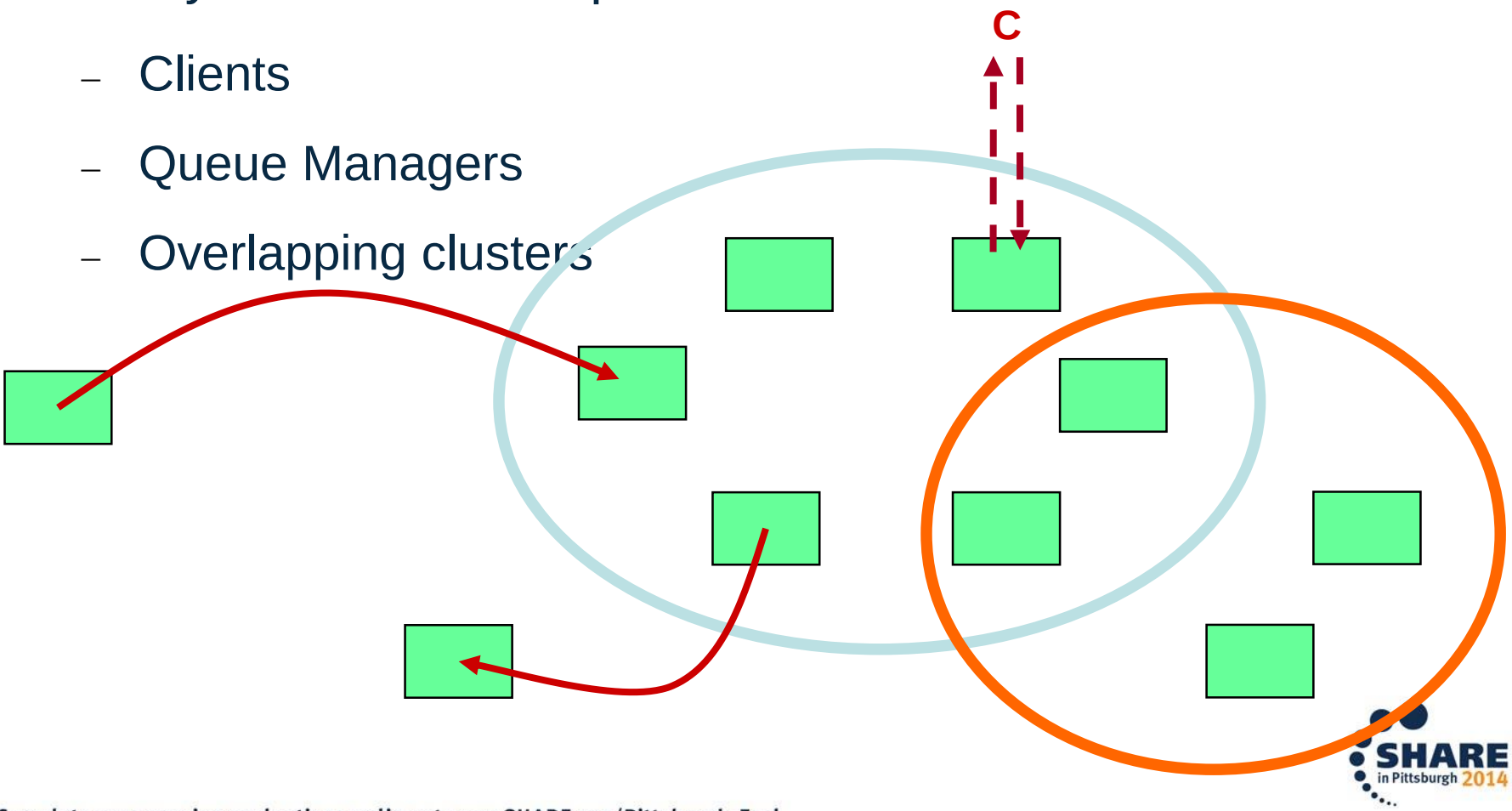
- Notice that…
  - the cluster topic string is /global/sports.
  - the subscribers' topic strings are /global/sports/* and /global/sports/football/*
    - Proxy-subscriptions are sent to other queue managers in the Pub/sub cluster for any topic string below the cluster topic string.
  - the publisher's topic string is /global/sports/football/news
    - Publications are sent based on local proxy-subscriptions

Flexible Topologies
Inter-Cluster routing



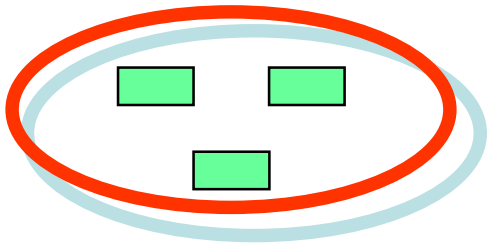
Routing messages in and out of clusters

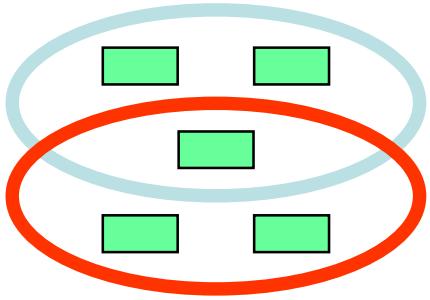
- A very flexible set of options
  - Clients
  - Queue Managers
  - Overlapping clusters



Overlapping clusters

- Why use them?
  - Geographic/network separation
  - To allow different organizations to administer their own
  - To allow independent applications to be administered separately
  - To create classes of service
  - To create test and QA environments (production usually kept separate)





Overlapping cluster channel and queues

- Channel configurations
  - Separate channels per cluster
    - **DEF CHL(TO.QM1.BLUE) CHLTYPE(CLUSRCVR) CLUSTER(BLUE)**
    - **DEF CHL(TO.QM1.RED) CHLTYPE(CLUSRCVR) CLUSTER(RED)**
    - **Finer admin control (e.g. Can stop channels for specific cluster, Security)**
  - Combined channels for multiple clusters
    - **DEF NAMELIST(CLNL) NAMES(BLUE,RED)**
    - **DEF CHL(TO.QM1) CHLTYPE(CLUSRCVR) CLUSNL(CLNL)**
    - **Fewer channel definitions**
- Queues configurations
  - Separate queues per cluster – more common
    - **DEF QL(Q1.BLUE) CLUSTER(BLUE)**
    - **DEF QL(Q1.RED) CLUSTER(RED)**
  - Queues in multiple clusters
    - **DEF NAMELIST(CLNL) NAMES(BLUE,RED)**
    - **DEF CHL(Q1) CLUSNL(CLNL)**
    - **Rarely seen**

Routing Into and Out of clusters

- See the Infocenter for theory and examples. See the next few charts for some examples
- Three key objects available…
  - Remote queue (QREMOTE)
    - Used to map remote queue names
    - E.g. DEFINE QREMOTE(LOCALQ) RNAME(REMQ) RQMNAME(QMB)
  - Queue manager alias (QREMOTE)
    - Used to map queue manager names
    - E.g. DEFINE QREMOTE(LOCALQMGR) RNAME(‘ ‘) RQMNAME(QMB)
  - Queue alias (QALIAS)
    - Used to map queue names
    - E.g. DEFINE QALIAS(Q1) TARGQ(Q2)
  - QREMOTE is used for both remote queues and queue manager aliases

Routing Into and Out of clusters

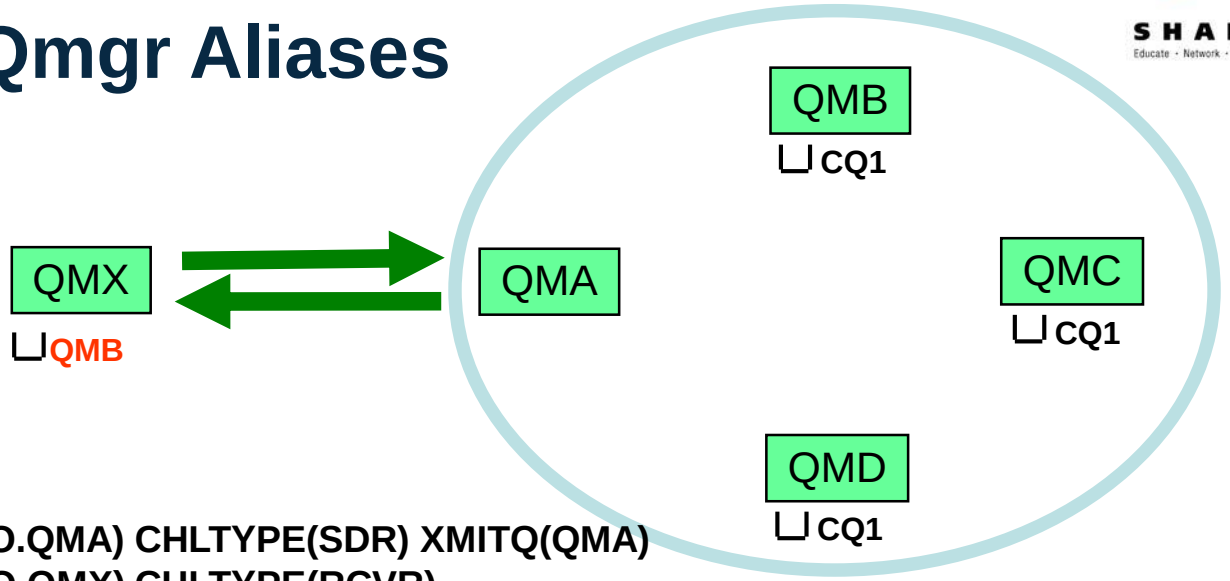
Questions to Consider:

- Do you want the message workload balanced or routed to a specific target?
- Do you want a fixed or flexible architecture.
- Flexible architectures -> Respond better to change
- Fixed architectures -> Give greater control
- “If I add another backend queue manager, what other queue managers need object changes?”
- Security requirements?
- Aliasing for security
- Different objects require different object authorities
- Different approaches require differing numbers of objects
- How much object architectural “awareness” is required in applications
- Do you want apps to specify a Queue Manager?
- Do you want apps to use local or remote queue objects?
- Do we want to change the code or application parameters?
- Can we change them?
- Can we remap with aliasing?

Scenario 1 – Qmgr Aliases

Application 1

```
MQCONNX(QMX)
MQOPEN(CQ1, QMB )
MQPUT
…
```



QMX

```
DEFINE CHL(QMX.TO.QMA) CHLTYPE(SDR) XMITQ(QMA)
DEFINE CHL(QMA.TO.QMX) CHLTYPE(RCVR)
DEFINE QL(QMA) USAGE(XMITQ)
DEFINE QREMOTE(QMB) RNAME(‘ ‘) RQMNAME(QMB) XMITQ(QMA)
```

QMA

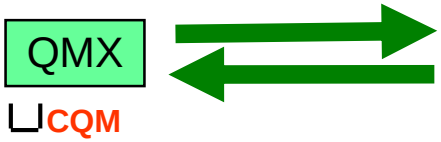
```
DEFINE CHL(QMA.TO.QMX) CHLTYPE(SDR) XMITQ(QMX)
DEFINE CHL(QMX.TO.QMA) CHLTYPE(RCVR)
DEFINE QL(QMX) USAGE(XMITQ)
```

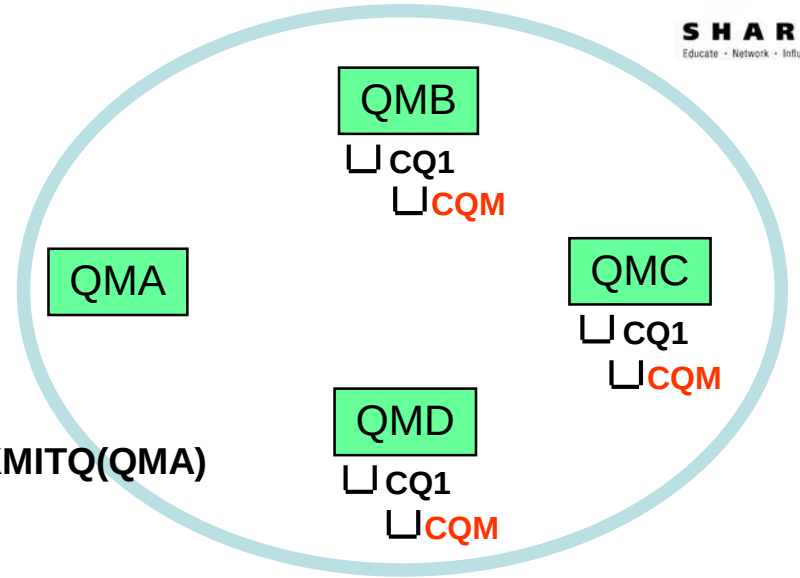
Now our app can put to CQ1 on QMB.
Without admin changes our app could put to any cluster queue on QMB.
But… can’t put to queues on QMC or QMD

Scenario 2 – Qmgr Aliases

Application 1

```
MQCONNX(QMX)
MQOPEN(CQ1, CQM)
MQPUT
…
```





QMX

DEFINE CHL(QMX.TO.QMA) CHLTYPE(SDR) XMITQ(QMA)
DEFINE CHL(QMA.TO.QMX) CHLTYPE(RCVR)
DEFINE QL(QMA) USAGE(XMITQ)
DEFINE QREMOTE(CQM) RNAME(‘ ‘) RQMNAME(CQM) XMITQ(QMA)

QMA

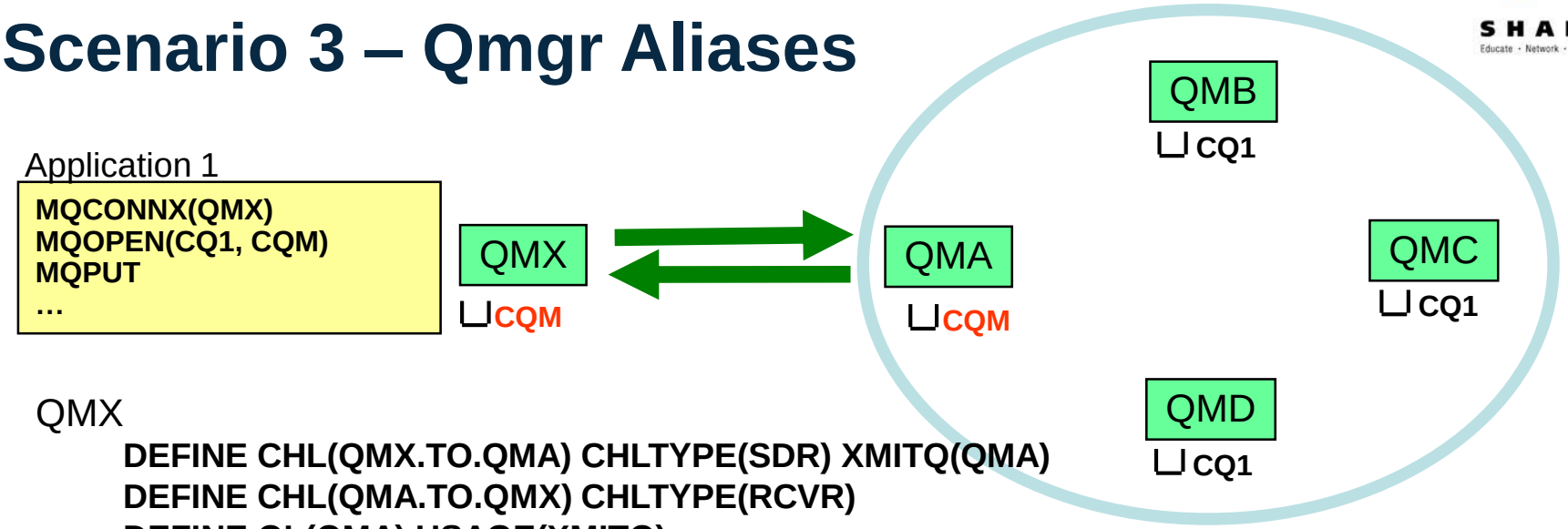
DEFINE CHL(QMA.TO.QMX) CHLTYPE(SDR) XMITQ(QMX)
DEFINE CHL(QMX.TO.QMA) CHLTYPE(RCVR)
DEFINE QL(QMX) USAGE(XMITQ)

QMn (n is either B, C or D)

DEFINE QREMOTE(CQM) RNAME(‘ ‘) RQMNAME(QMn) CLUSTER(BLUE)

Now our app can put to CQ1 on QMB, QMC, and QMD.
Without admin changes our app could put to any clustered queue on QMB, QMC, and QMD
But… If we want to workload balance to all queue managers in the cluster, it seems like a lot of effort to define so many queue manager aliases

Scenario 3 – Qmgr Aliases



QMX

DEFINE CHL(QMX.TO.QMA) CHLTYPE(SDR) XMITQ(QMA)
DEFINE CHL(QMA.TO.QMX) CHLTYPE(RCVR)
DEFINE QL(QMA) USAGE(XMITQ)
DEFINE QREMOTE(CQM) RNAME(‘ ‘) RQMNAME(CQM) XMITQ(QMA)

QMA

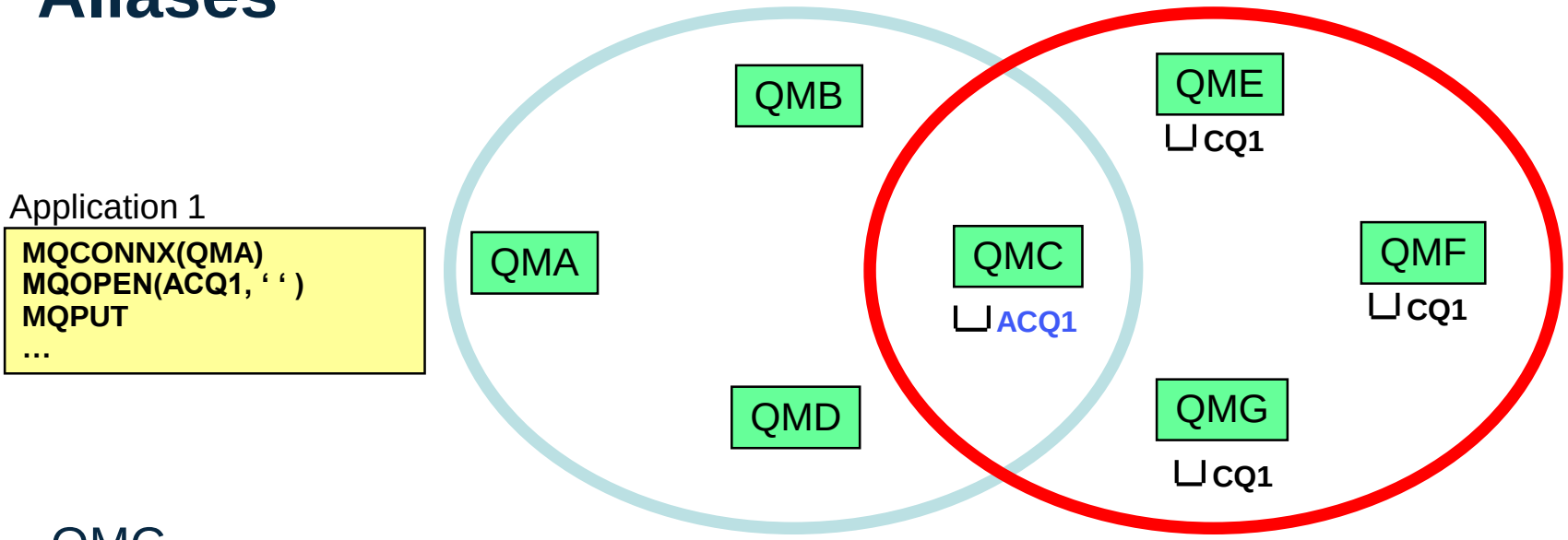
DEFINE CHL(QMA.TO.QMX) CHLTYPE(SDR) XMITQ(QMX)
DEFINE CHL(QMX.TO.QMA) CHLTYPE(RCVR)
DEFINE QL(QMX) USAGE(XMITQ)
DEFINE QREMOTE(CQM) RNAME(‘ ‘) RQMNAME(‘ ‘)

Now our app can put to CQ1 on QMB, QMC, and QMD.

Without admin changes our app could put to any queue (clustered or not) on QMA, QMB, QMC, and QMD

We can add queue managers and queues to the cluster without needing to add more queue manager aliases

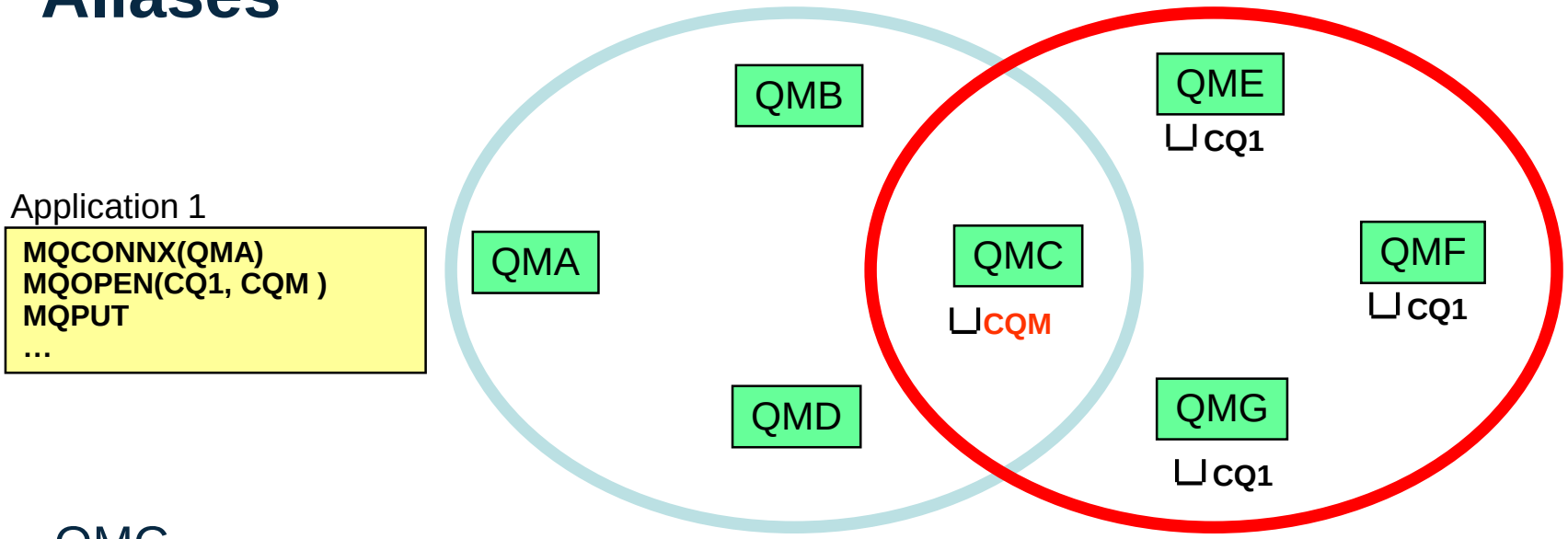
Scenario 4 – Overlapping Clusters & Q Aliases



- QMC
  - DEFINE QALIAS(ACQ1) TARGQ(CQ1) CLUSTER(BLUE)

- Now our app can put to any CQ1 in cluster RED
- But… need an alias for each queue

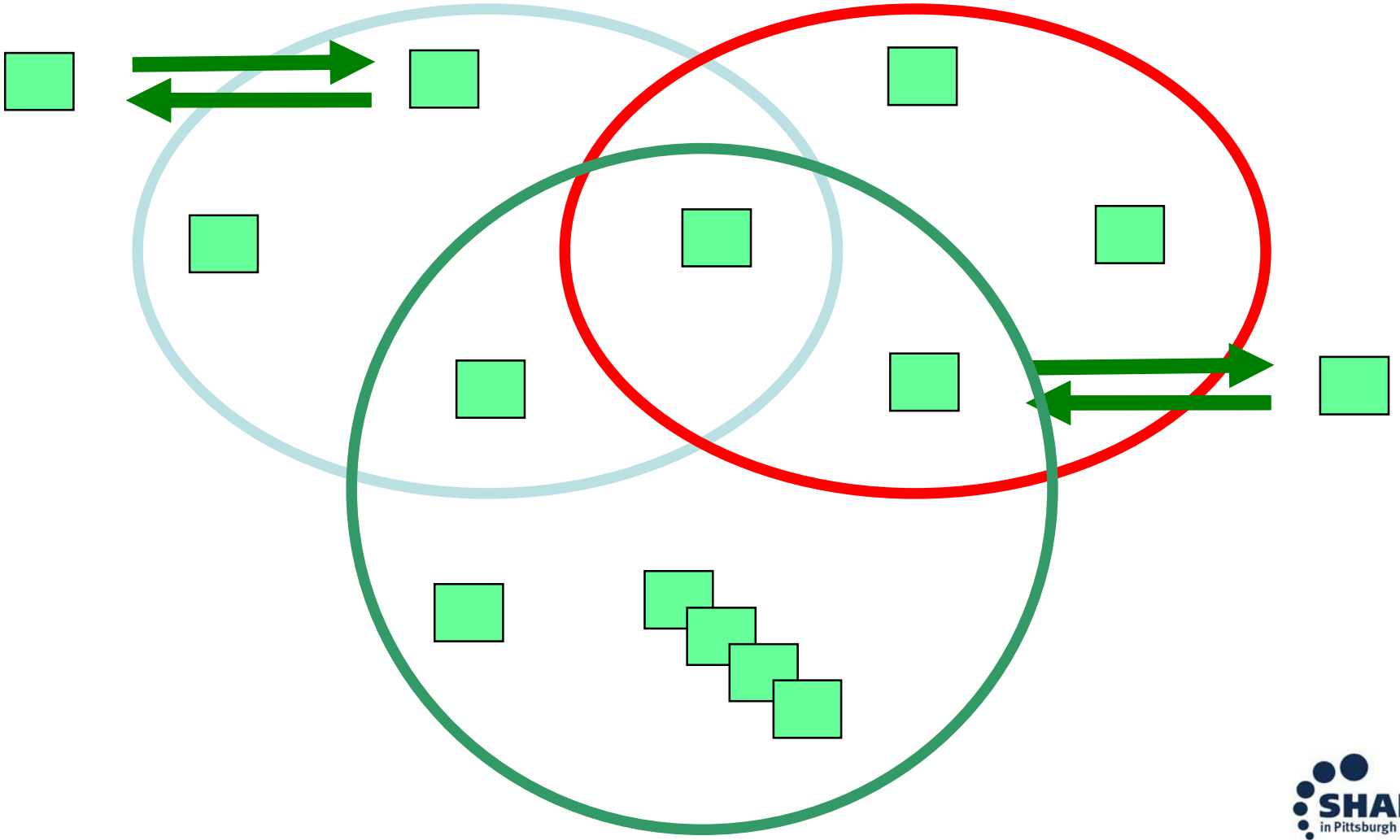
Scenario 5 - Overlapping Clusters & Qmgr Aliases



- QMC
  - DEFINE QREMOTE(CQM) RNAME(‘ ‘) RQMNAME(‘ ‘) CLUSTER(BLUE)

- Now our app can put to any cluster queue in cluster RED

Lots possible



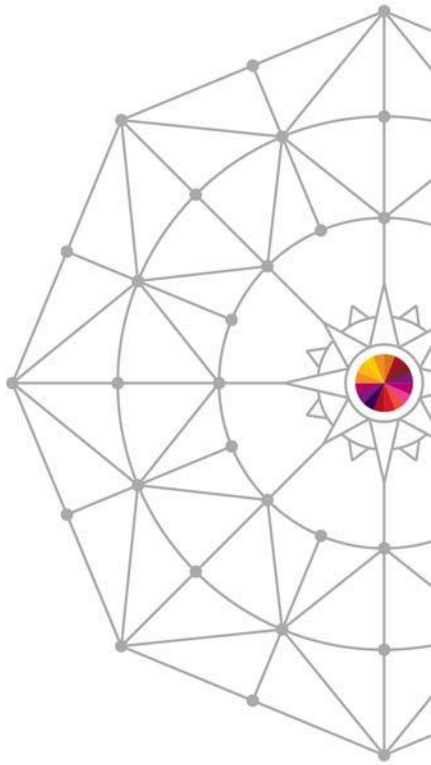
Remote Qs, QM aliases, Q aliases… Which should I use????

- Here are some considerations
  - Security
  - How flexible is the architecture
    - More queue managers? More queues?
  - Keep checking…. Is this as simple as it could be?
  - What knowledge of the system does the app require?
    - Queue manager name? Target queue name or some intermediate alias?
  - Should messages be workload balanced?
    - Or just using clustering for connectivity

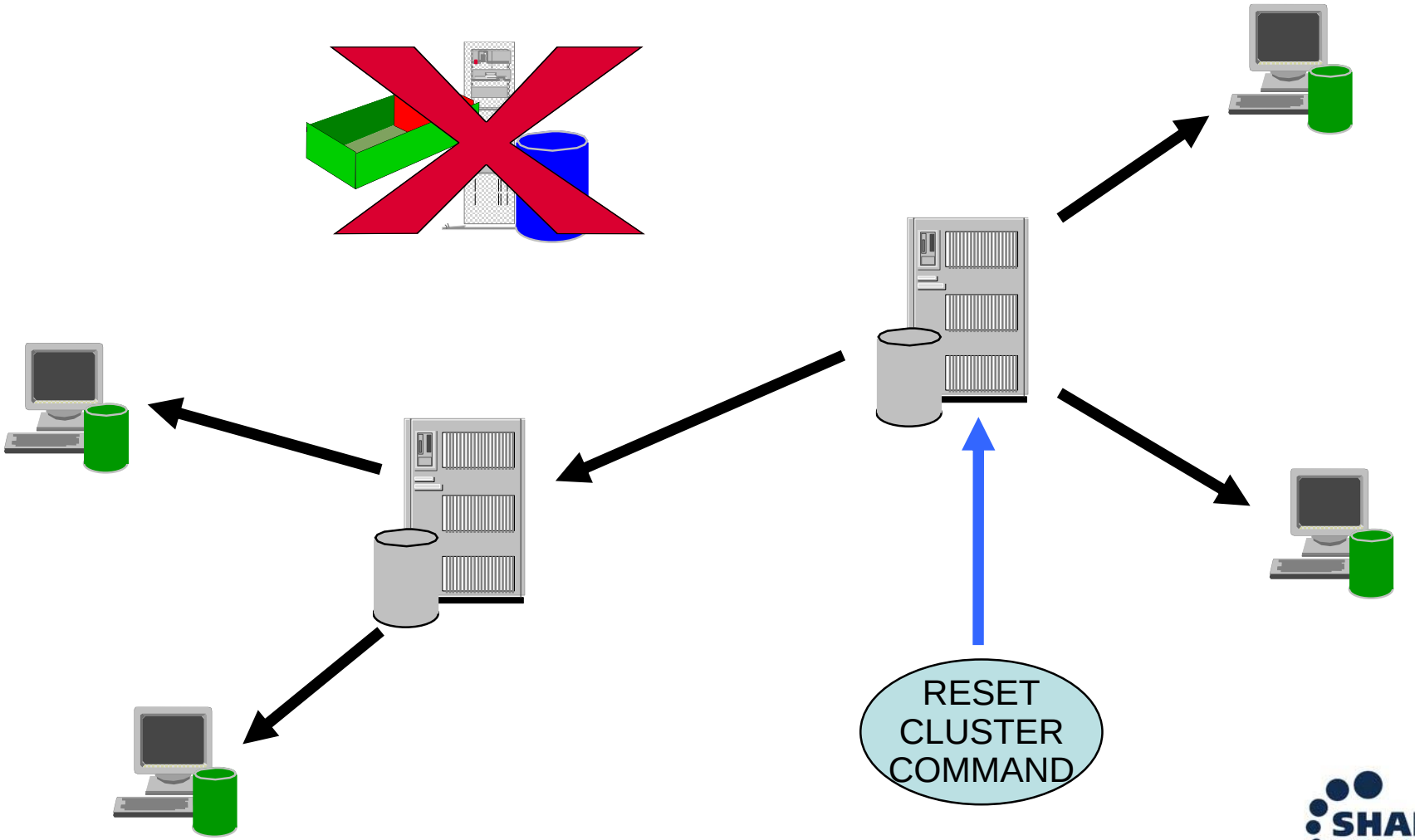
Remote Qs, QM aliases, Q aliases… Which should I use????

- With all these different options available, which should I use? Well the answer does depend. The considerations for your set-up will help make your choice. Here are some things to consider when making the choice.
- When thinking about security, be aware that queue manager aliases cannot be secured by named object (instead you use the SYSTEM.CLUSTER.TRANSMIT.QUEUE name). Queues, remote queues, and queue aliases can be secured by named object. This provides a more granular security model which is useful when a single queue manager is used by multiple users/applications/workloads.
- Take into account how much change is likely in the future in your architecture. For example, do you need to add more queue manager aliases if you add another cluster queue manager? Do you need to add more queue aliases if you add another cluster queue?
- What knowledge of the system does the application require. For example, does it need to specify a queue manager name? Does it need to specify the target queue name or some intermediate alias? MQ is very forgiving here, with a strong set of aliasing available.
- Are you simply trying to get a message from outside the cluster to one particular queue manager OR do you want to use the workload balancing function of clustering?
- Some useful manual references
  - **'Using aliases and remote-queue definitions with clusters' section of the WebSphere MQ Queue Manager Clusters manual**
  - **'Name resolution' section of the WebSphere MQ Application Programming Guide**

Further Considerations and Recommendations



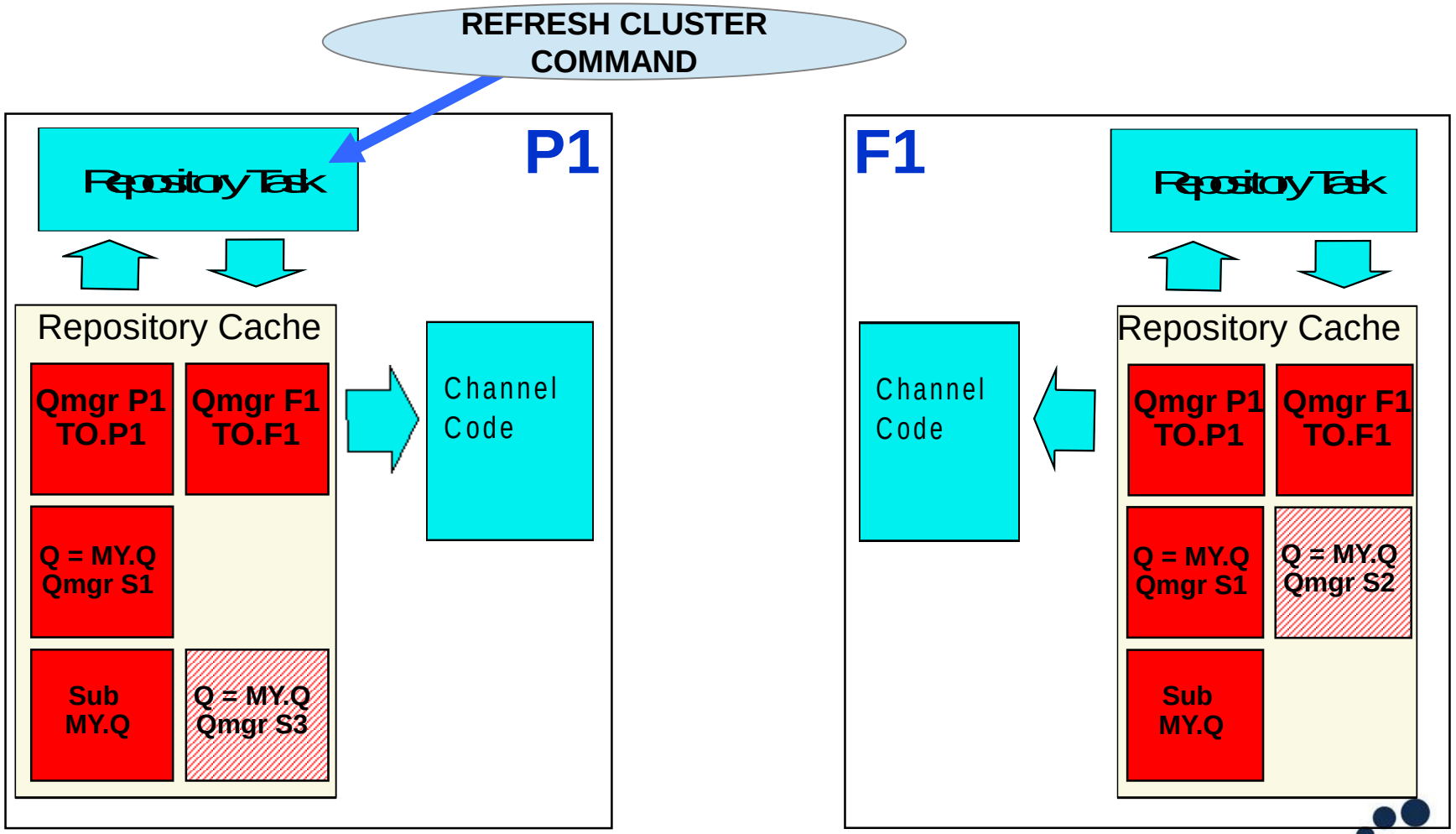
Incorrectly deleted queue managers



Incorrectly deleted queue managers

- If a queue manager is deleted without first removing it from the cluster, then the rest of the queue managers in the cluster will still believe that the queue manager exists and will retain this information for up to 90 days.
- Another way is required to tell the cluster to remove the information about the deleted queue manager. The RESET CLUSTER command can be used to force a queue managers definitions to be deleted from the cluster. It has to be run on a Full Repository, as the Full Repositories know about all the cluster resources owned by the queue manager being removed and can then inform the other queue managers in the cluster to delete them.
- Quite often we see a situation where the deleted queue manager is recreated using a script, however the cluster treats this as a different queue manager, as each queue manager is assigned a unique queue manager identifier when it is created. This can lead to duplicate queue managers in the cluster. Once again the RESET CLUSTER command can be used to remove the old instance of the queue manager by deleting it by its queue manager identifier (QMID). The ability to force remove a queue manager by QMID was introduced in V5.3.

Refreshing repository information



Refreshing repository information

- The diagram on the previous page shows a situation where the information held in the Partial Repositories cluster cache has become out of sync with the information held on the Full Repository. In this example, the Partial Repository believes that a queue called MY.Q is hosted by queue managers S1 and S3, but the Full Repository knows it is actually hosted by S1 and S2.
- This discrepancy can be fixed by issuing the REFRESH CLUSTER command on the Partial Repository. The command tells the Partial Repository to delete the information it has built up about the cluster. This information will then be rebuilt over time, by requesting it from the Full Repositories.
- It is not normally necessary to issue a REFRESH CLUSTER command except in one of the following circumstances:
  - Messages have been removed from either the SYSTEM.CLUSTER.COMMAND.QUEUE, or from another queue manager's SYSTEM.CLUSTER.TRANSMIT.QUEUE, where the destination queue is SYSTEM.CLUSTER.COMMAND.QUEUE on the queue manager in question.
  - Issuing a REFRESH CLUSTER command has been recommended by IBM® Service.
  - The CLUSRCVR channels were removed from a cluster, or their CONNAMEs were altered on two or more Full Repository queue managers while they could not communicate.
  - The same name has been used for a CLUSRCVR channel on more than one queue manager in a cluster, and as a result, messages destined for one of the queue managers have been delivered to another. In this case, the duplicates should be removed, and then a REFRESH CLUSTER command should be issued on the single remaining queue manager that has the CLUSRCVR definition.
  - RESET CLUSTER ACTION(FORCEREMOVE) was issued in error.

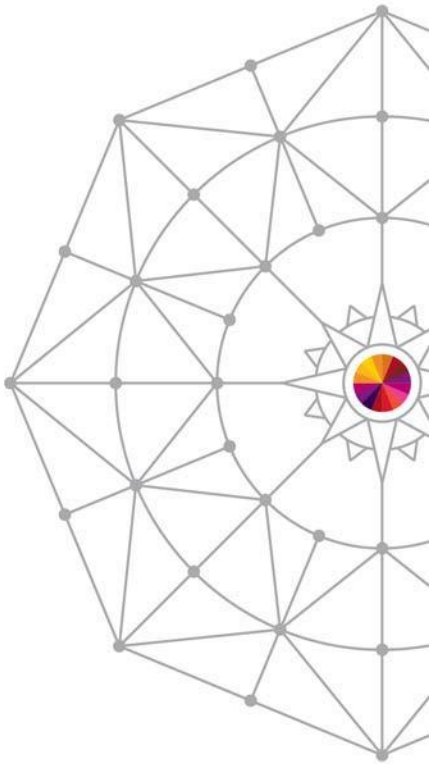
Implementation recommendations

- Be clear about your requirements
  - Reduced system administration?
  - Workload balancing?
- Read the Queue Manager Clusters manual
  - The tasks sections are useful – Especially how to remove a queue manager from a cluster
- Helps to experiment in a development/test environment
  - Before using commands (especially REFRESH CLUSTER) read the whole section for that command in the MQSC manual
- Usage notes contain important info
- Naming conventions
  - No one recommended convention - consistency is the key
- Document processes for production system changes
  - Add/remove queue manager to/from cluster
  - Take queue manager offline for maintenance
- Start small
- Have two Full Repositories
  - Be careful about connecting them
  - Consider where you should host them

Implementation recommendations (continued)

- Be careful defining cluster topics in existing clusters
- Monitor the SYSTEM.CLUSTER.TRANSMIT.QUEUE
  - Per channel CURDEPTH (V6) – DIS CHSTATUS XQMSGSA
- When debugging check that channels are healthy
  - Definition propagation: Path from one QM to another via FRs
  - Application: Path from application to queue manager hosting queue.
- Consider how you will administrate and debug
  - Monitor CSQX4.../AMQ94.. messages
  - MQRC_UNKNOWN_OBJECT_NAME
  - “Where’s my message?” (when there are now n cluster queues)
- Bind-not-fixed gives better availability
  - By default, bind-on-open is used
- Further Information in the Infocenter
  - Queue Manager Clusters
  - Script (MQSC) Command Reference
  - Publish/Subscribe User’s Guide

What's new?



Workload Balancing by Group: Background

- Previously, when WMQ queue manager clusters have been used to balance messages across multiple instances of a queue hosted on different queue managers, two ‘binding options’ have been provided: BIND_ON_OPEN which ensures that all messages for the lifetime of the object handle are routed to the same instance, and BIND_NOT_FIXED which allows ‘spraying’ across multiple instances.
- A third option is now available to maintain integrity of groups of messages, while still exploiting workload balancing.
- Workload balancing algorithm unchanged, but will only be driven between complete groups when using the new option
- Allows exploitation of groups in usual manner on MQGET:
  MQGMO_ALL_MSGS_AVAILABLE
  MQGMO_LOGICAL_ORDER

Workload Balancing by Group: Usage

- New MQOPEN option in the MQI
  - MQOO_BIND_ON_GROUP (0x00400000, 4194304)
- 'Out of order/application managed groups not supported, so:
  - Must specify MQPMO_LOGICAL_ORDER at put time
  - Must NOT manually specify group ID (leave as MQGI_NONE).
- Ungrouped messages, or messages which do not comply with above, will fall back to BIND_NOT_FIXED
  - Except where group options would give an error today (e.g. MQRC_INCOMPLETE_GROUP)
- Use usual grouping flags within the MD of messages:
  Last message in group: MQMF_LAST_MSG_IN_GROUP
  All other messages in group: MQMF_MSG_IN_GROUP
- *Note: As with all usage of groups on z/OS, must specify: INDXTYPE(GROUPID)*

Workload Balancing by Group: Example

- **MQOPEN QX@QM1**

  (MQOO_BIND_ON_GROUP)

- **MQPUT Msg1**

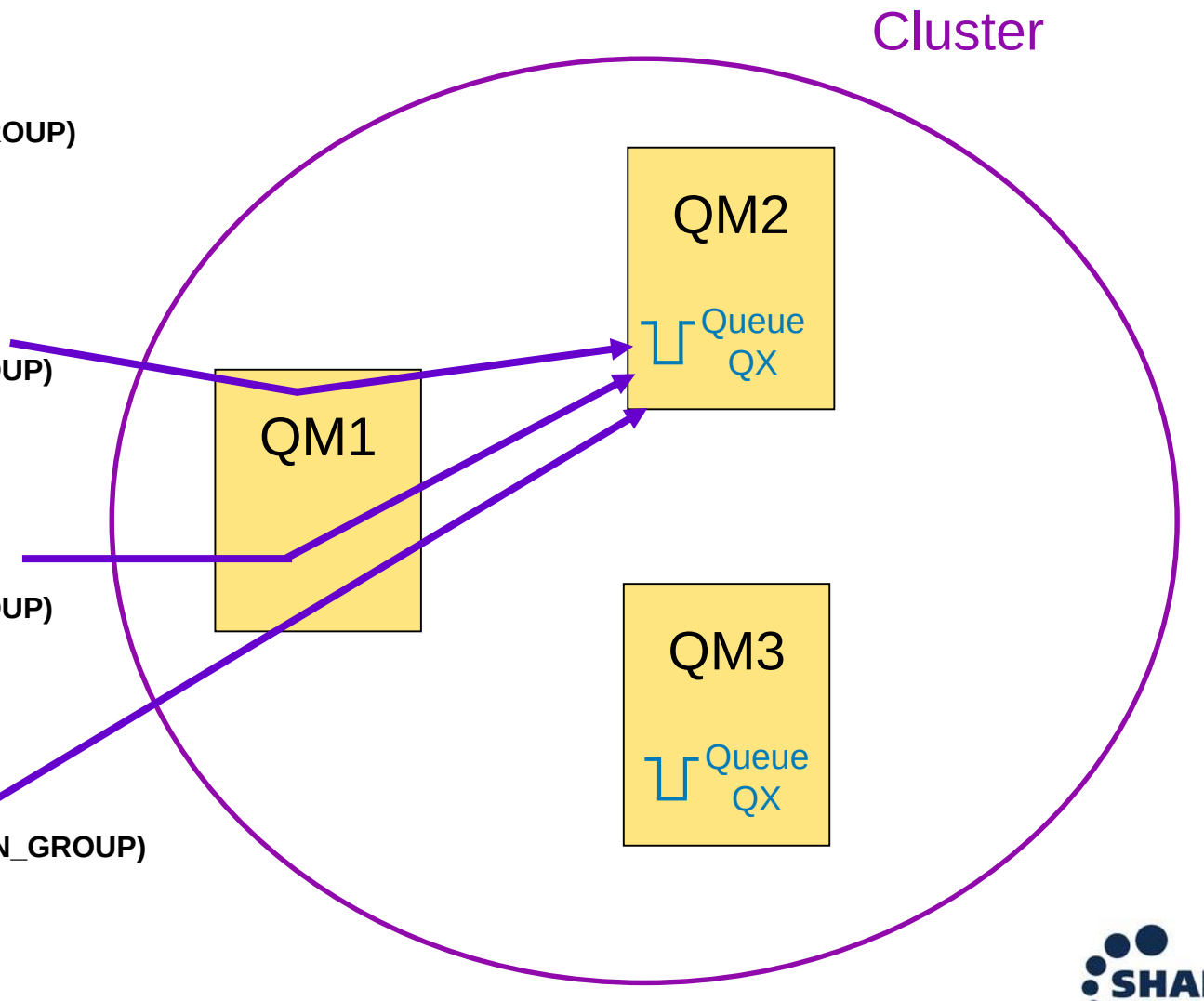
  (MQMF_MSG_IN_GROUP)

- **MQPUT Msg2**

  (MQMF_MSG_IN_GROUP)

- **MQPUT Msg3**

  (MQMF_LAST_MSG_IN_GROUP)



Workload Balancing by Group: Example

- **MQOPEN QX@QM1**

  (MQOO_BIND_ON_GROUP)

- **MQPUT Msg4**

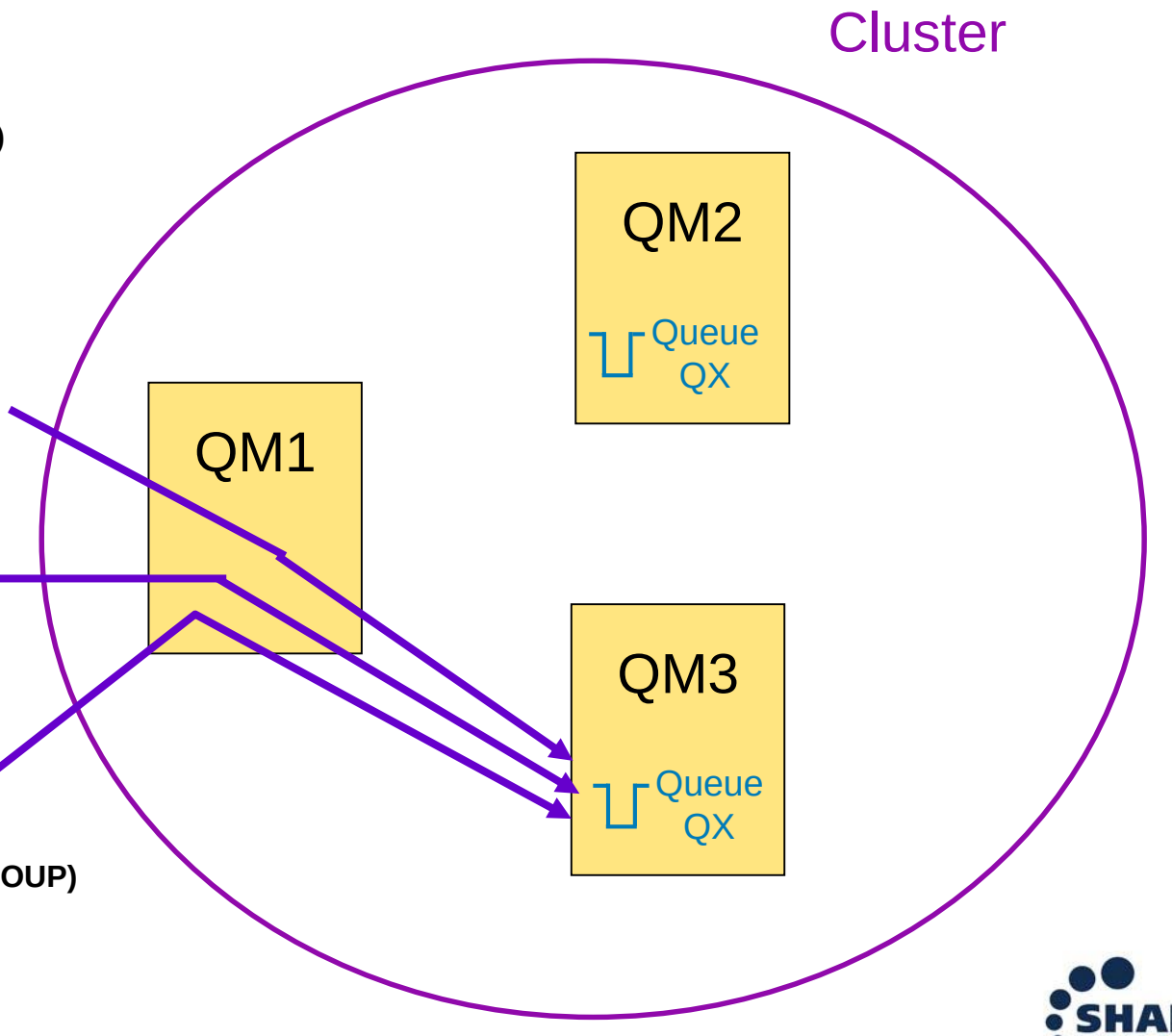
  (MQMF_MSG_IN_GROUP)

- **MQPUT Msg5**

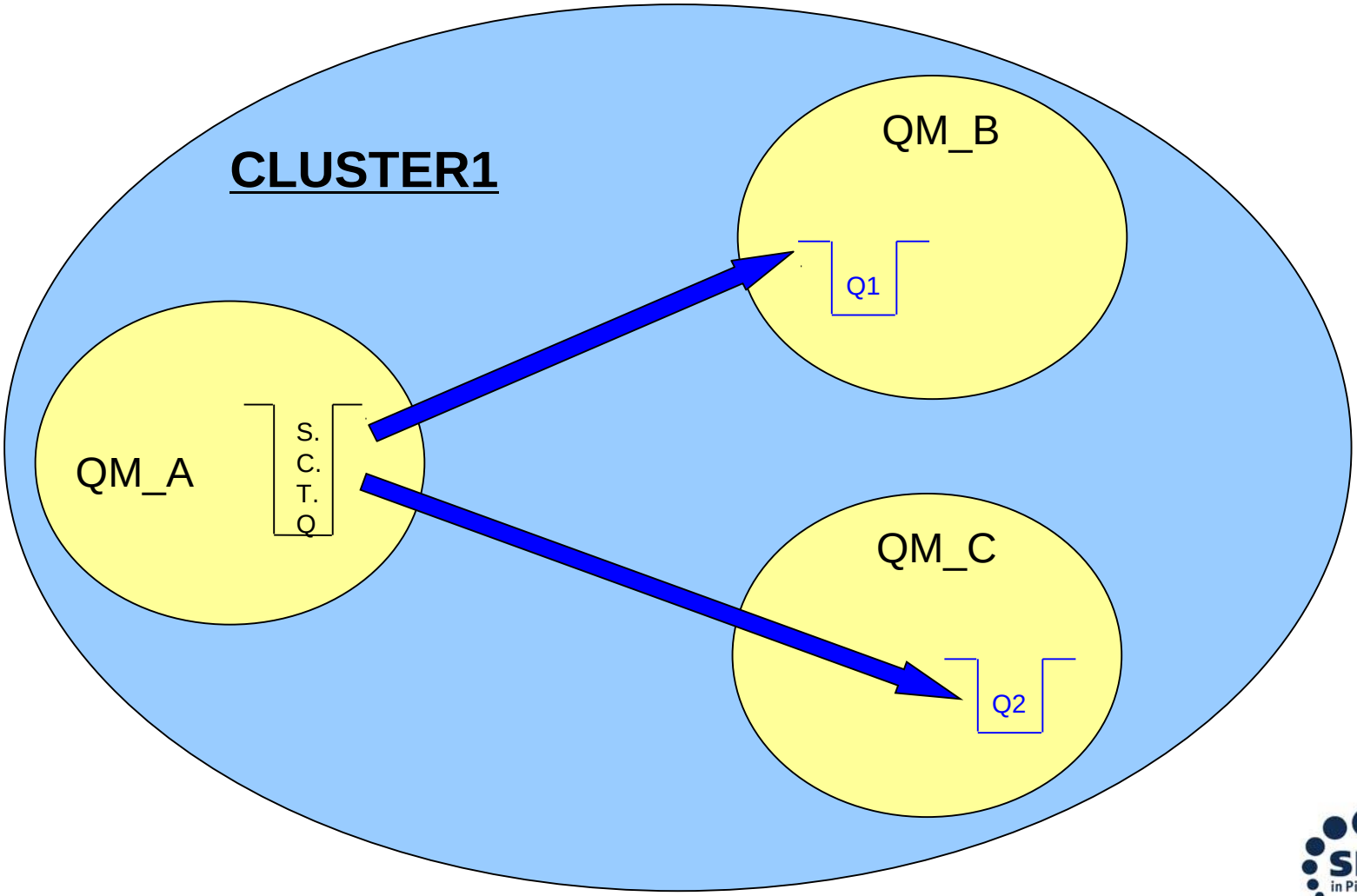
  (MQMF_MSG_IN_GROUP)

- **MQPUT Msg6**

  (MQMF_LAST_MSG_IN_GROUP)



The SYSTEM.CLUSTER.TRANSMIT.QUEUE



Split Cluster Transmit Queue

- Much requested feature for various reasons…
- **Separation of Message Traffic**
  - With a single transmission queue there is potential for pending messages for cluster channel 'A' to interfere with messages pending for cluster channel 'B'
- **Management of messages**
  - Use of queue concepts such as MAXDEPTH not useful when using a single transmission queue for more than one channel.
- **Monitoring**
  - Tracking the number of messages processed by a cluster channel currently difficult/impossible using queue monitoring (some information available via Channel Status).
- **Not** about performance...

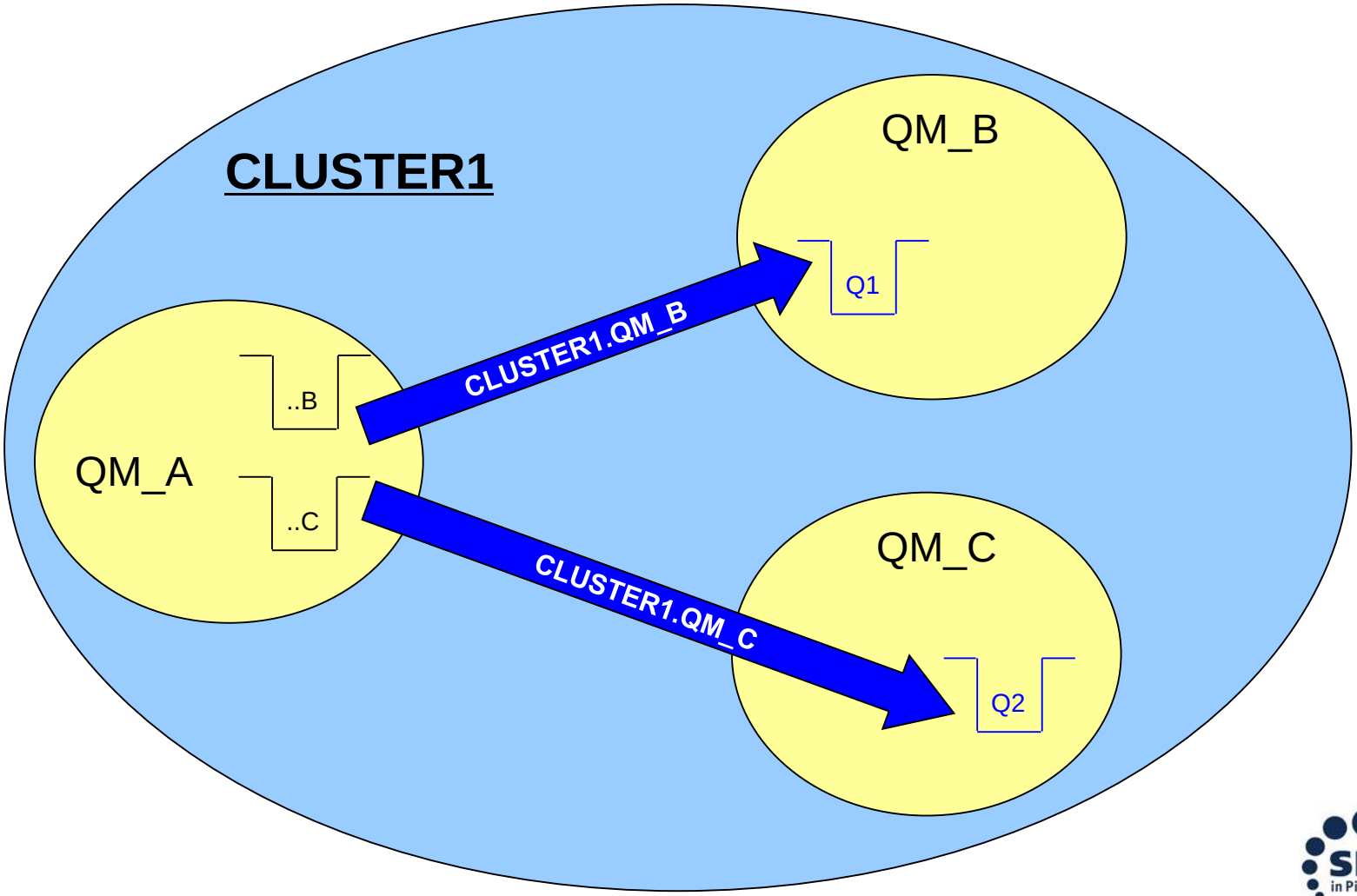
Notes: Split Cluster Xmit Q Background

- 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 affects 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 CHANNEL is selected.

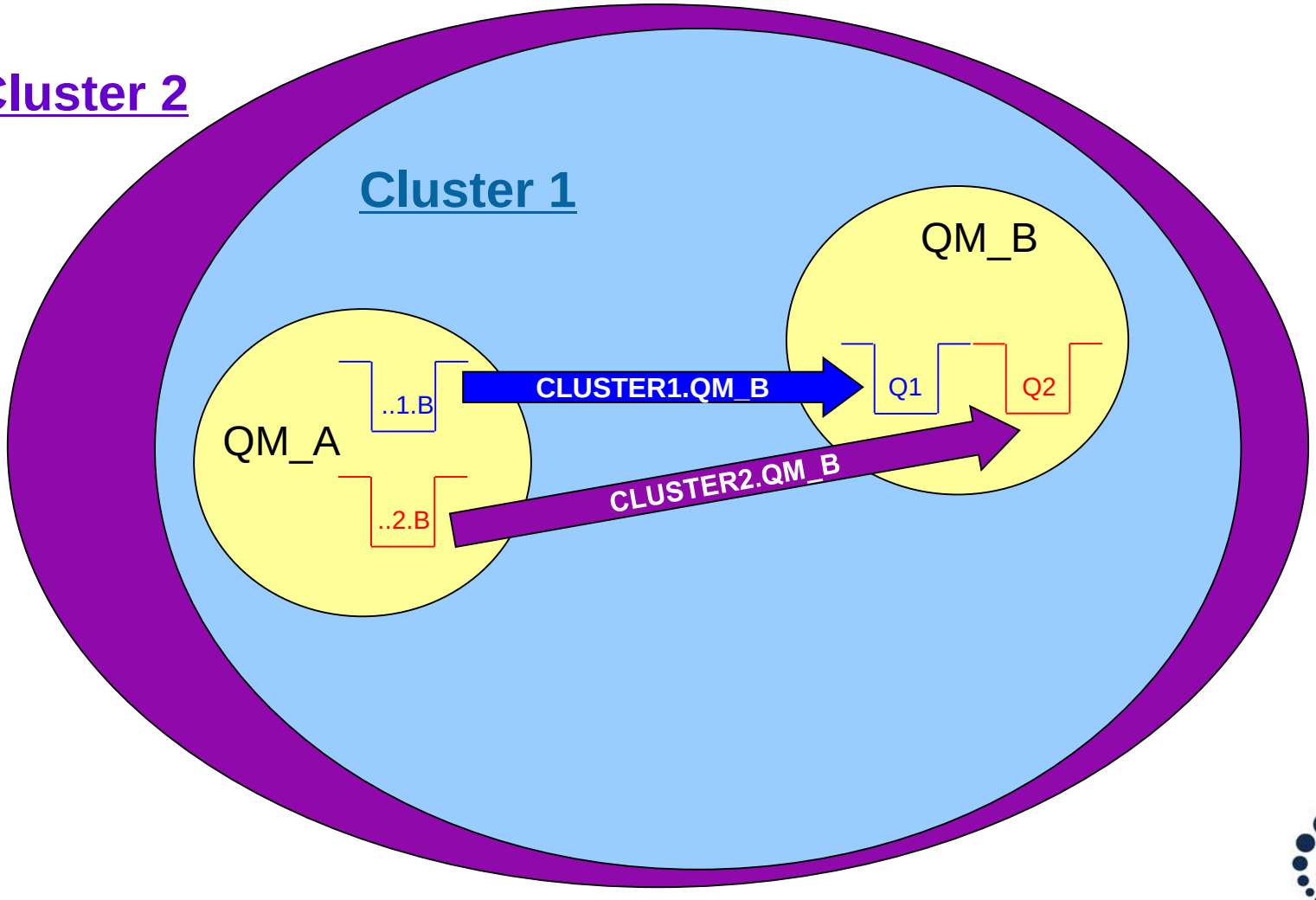
Splitting Out the S.C.T.Q. Per Channel



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.

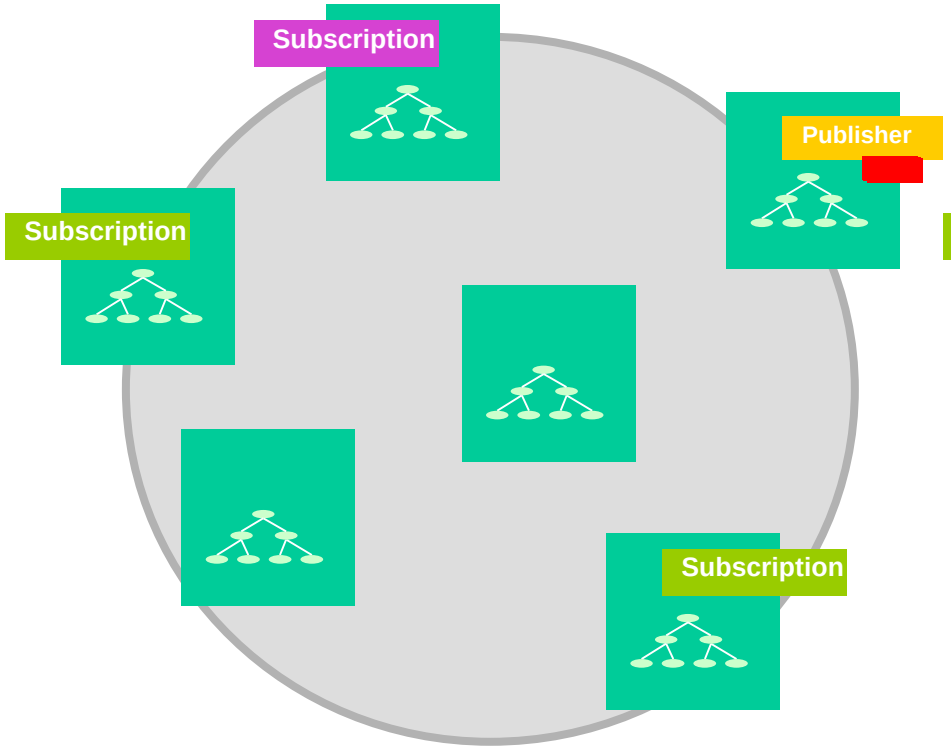
Splitting Out by Cluster (or Application)

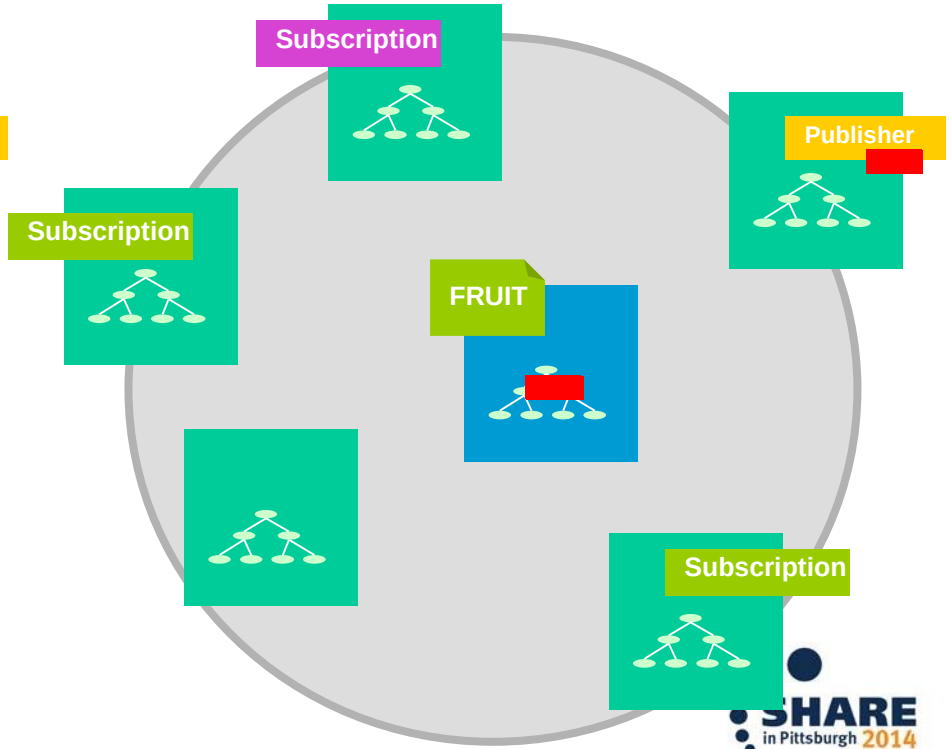


Routed Pub/Sub

- Publish/subscribe in clusters now has two modes of routing publications.
  - **DIRECT**: Publications are sent directly from the publisher's queue manager to every queue manager in the cluster with a matching subscription.
  - **TOPIC HOST**: Publications are routed via one or more queue managers in the cluster. These are the queue managers where the clustered topic definitions were defined.

V8





Cluster routing - Notes

WebSphere MQ V8 introduces the new topic object attribute of CLROUTE *(ClusterPubRoute)*. This takes two values:

- DIRECT
  - All queue managers in the cluster are aware of everyone else and will send publications directly to other queue managers in the cluster with matching subscriptions. This is the default.
- TOPICHOST
  - Only the queue manager(s) where the topic object is defined are aware of everyone else. Publications are forwarded to these topic hosting queue managers, who then forward onto queue managers with matching subscriptions.

To use topic host routing in a cluster, the following queue managers must be at V8 or above:

- The topic hosting queue managers
- The full repository queue managers
- The queue managers where subscriptions exist
- The queue managers where publishers connect

Any older queue managers will not be aware of the topic host routed topic definition and continue to behave as if it was not defined in the cluster.

How it works: subscription propagation

- A subscription for a topic string is created on a queue manager.
- That queue manager tells others that it is interested in that topic string.
- Those queue managers register ***proxy*** subscriptions to represent the interest.
- On a queue manager, when a message is published to the topic string, a copy of the message is sent, over WebSphere MQ channels, to each queue manager that a proxy subscription is held for.
- The receiving queue manager processes the message and gives a copy to each subscription it holds *(including any proxies…).*

Proxy subscriptions

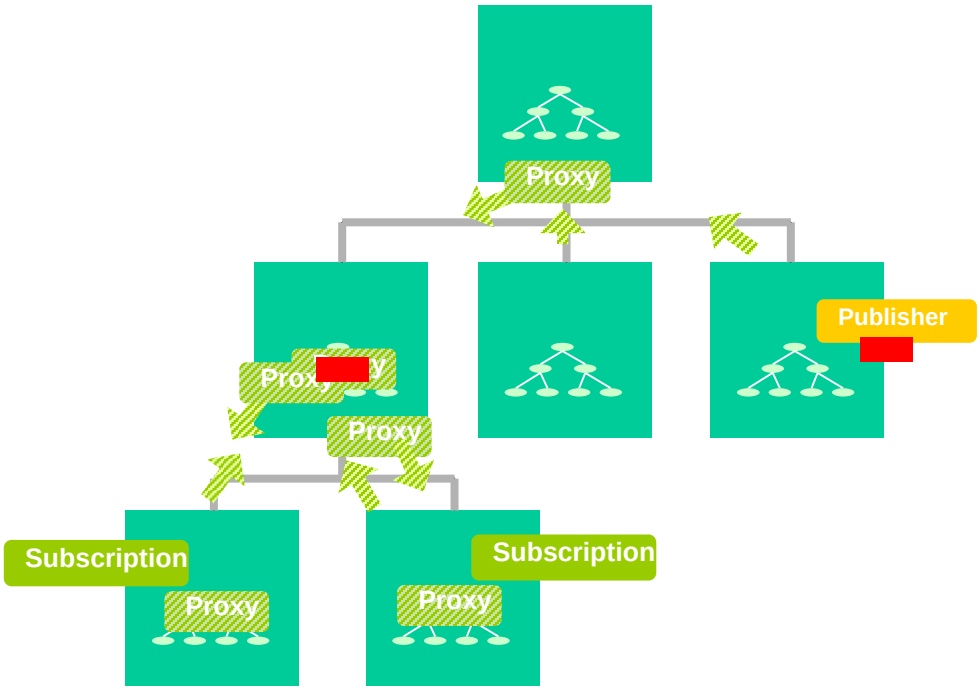
- Proxy subscriptions are a special type of subscription that tells one queue manager that another queue manager needs a copy of any matching publications to be sent to it.
- Proxy subscriptions can be viewed on a queue manager using MQSC or MQ Explorer. This shows which queue managers will receive a publication for which topic.

```
DISPLAY SUB(*) SUBTYPE(PROXY) TOPICSTR DESTQMGR


AMQ8096: WebSphere MQ subscription inquired.
   SUBID(414D5120514D4752312020202020202021123C5320000E02)
   SUB(SYSTEM.PROXY.QMGR2 CLUSTER1 /X/Y)   TOPICSTR(/X/Y)
   DESTQMGR(QMGR2)                         SUBTYPE(PROXY)
AMQ8096: WebSphere MQ subscription inquired.
   SUBID(414D5120514D4752312020202020202021123C5320000E05)
   SUB(SYSTEM.PROXY.QMGR2 CLUSTER1 /X/Z)   TOPICSTR(/X/Z)
   DESTQMGR(QMGR2)                         SUBTYPE(PROXY)
```

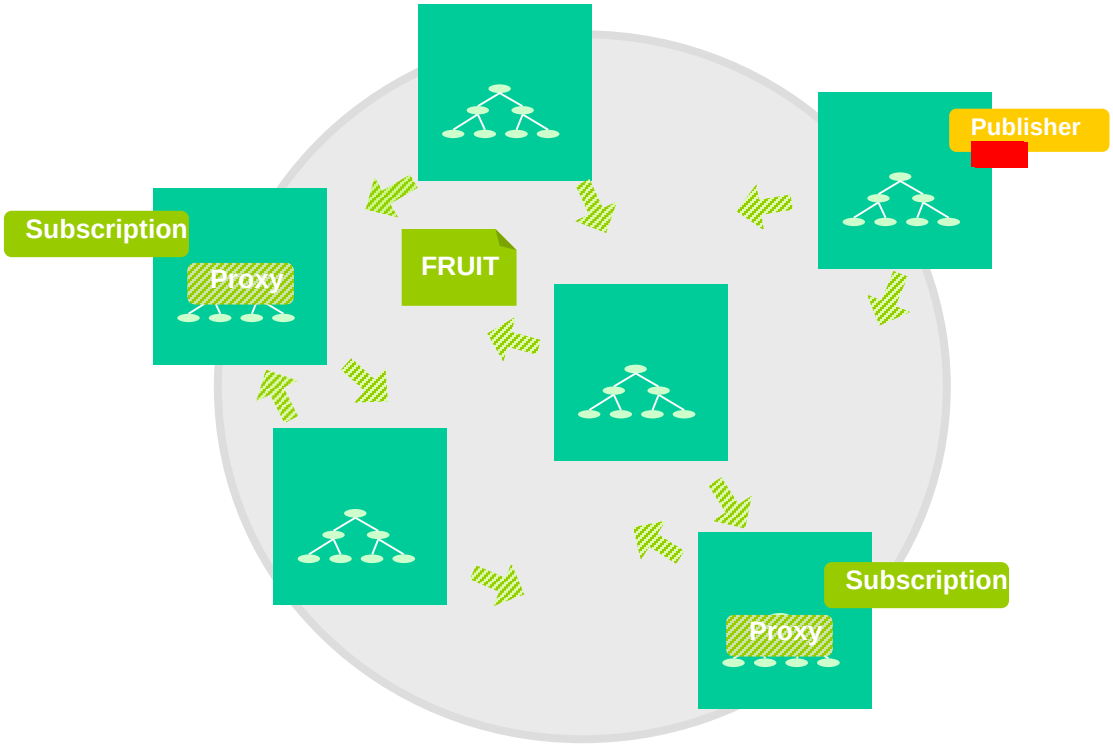
PubSub Cluster Routing: In a Hierarchy

- Proxy subscriptions are sent from a queue manager to every ***directly*** connected queue manager in the hierarchy.
- They in turn send proxy subscriptions onto any further relations.
- Until everyone in the hierarchy is aware of the topic string being subscribed to.
- Publications are then sent back down the path of proxy subscriptions.



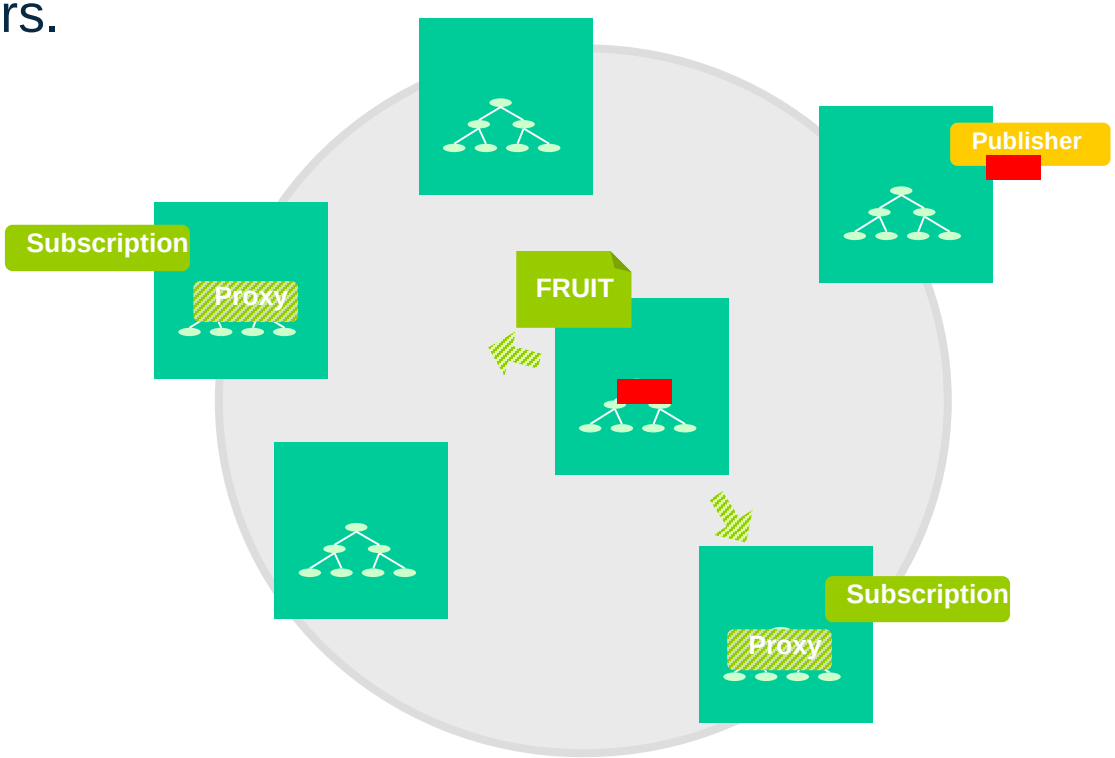
PubSub Cluster Routing: Direct Routed Clustered Topics

- Proxy subscriptions are sent from a subscribing queue manager to ***every*** other queue manager in the cluster.
  - *Every queue manager is aware of everyone else in the cluster.*
- Any publication from a queue manager is sent directly to those queue managers which sent proxy subscriptions.



How it works: Topic host routed clustered topics

- Proxy subscriptions are sent from a subscribing queue manager ***only*** to queue managers that host a definition of the clustered topic.
  - Only the topic hosts are aware of everyone else.
- Any publication from a queue manager is ***always*** sent to one of the topic hosts, who then forwards the message to any subscribing queue managers.



V8

Any questions?

This was session 16196 - The rest of the week ……

	Monday	Tuesday	Wednesday	Thursday	Friday
08:30			Application programming with MQ verbs	The Dark Side of Monitoring MQ - SMF 115 and 116 Record Reading and Interpretation	CICS and MQ - Workloads Unbalanced!
10:00					
11:15	Introduction to MQ	What's New in IBM Integration Bus & WebSphere Message Broker	MQ – Take Your Pick Lab	Using IBM WebSphere Application Server and IBM WebSphere MQ Together	
12:15					
01:30		All about the new MQ v8	MQ Security: New v8 features deep dive	New MQ Chinit monitoring via SMF	
03:00	MQ Beyond the Basics	MQ & DB2 – MQ Verbs in DB2 & InfoSphere Data Replication (Q Replication) Performance	What's wrong with MQ?	IIIB - Internals of IBM Integration Bus	
04:15	First Steps with IBM Integration Bus: Application Integration in the new world	MQ for z/OS v8 new features deep dive	**MQ Clustering - The Basics, Advances and What's New in v8**