Recording available at:
http://www.slideshare.net/DavidWare1/ame-2273-mq-clustering-pdf

What can you achieve with MQ clusters?

Matt Leming
lemingma@uk.ibm.com

Agenda

- **Why a cluster?**
  - Starting to scale and setting one up
- **Service availability**
  - Routing around failures
- **Location dependency**
  - Smarter routing
- **Levels of separation**
  - What are we sharing?

NOTES: Terminology

Lots of different terminology floats around when you get into application infrastructure discussions… Clients, applications, servers, services, requesters, responders…
For the purposes of this session:

Client – a ‘client’ in the general sense (whether connected locally or remotely), uses IBM MQ to send one off datagrams or initiates requests to services and waits for replies.

Service – a process which consumes messages and takes some action, often needing to reply to the requesting **Client.**

Note that there may be more than one **Instance** of a client or service, either connected to a given queue manager or the infrastructure as a whole.

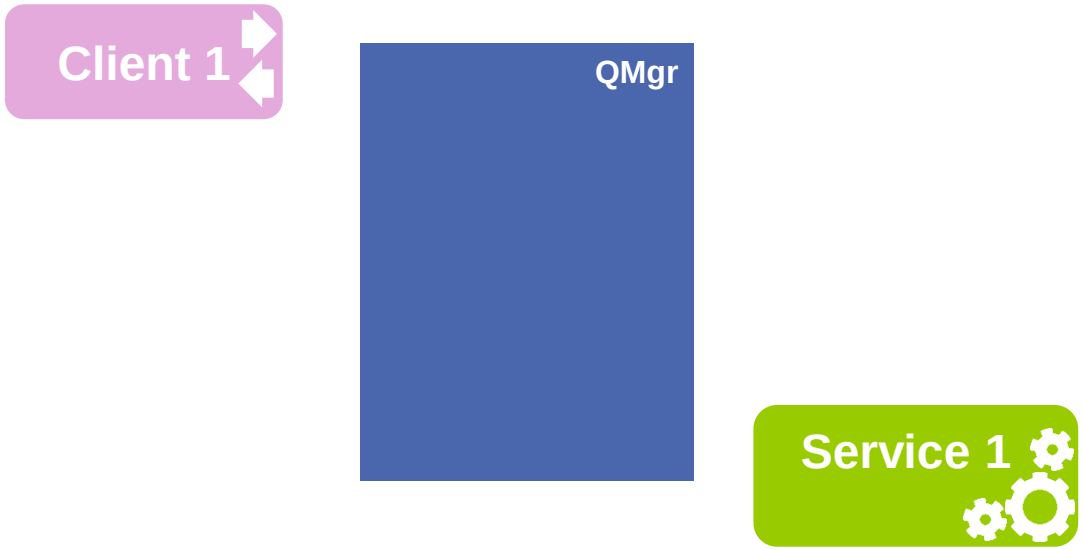
A set of clients and services working together to achieve some useful end goal make up an ‘application’.

Queue Manager

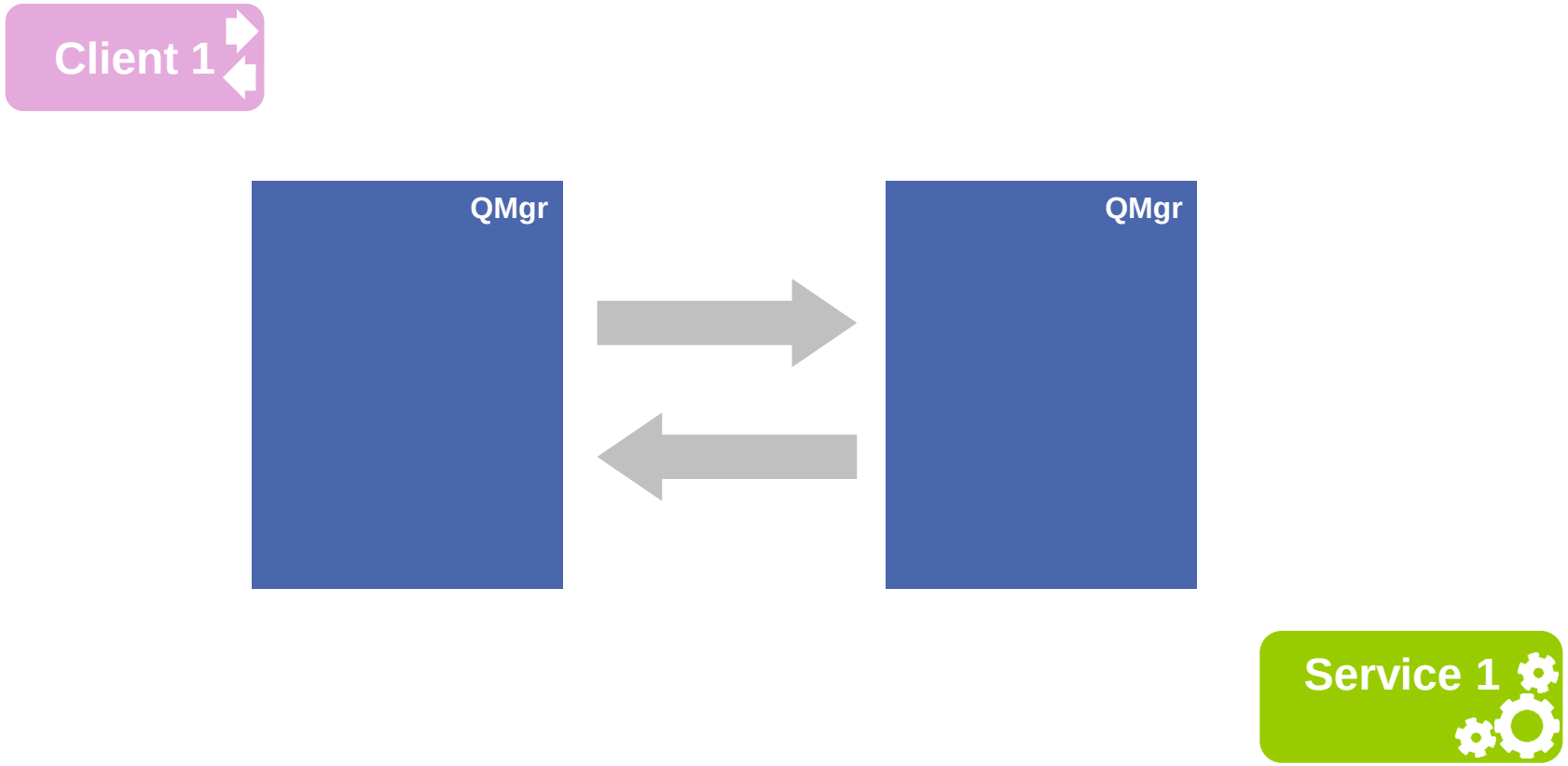
Full repository Queue Manager

Scaling your administration

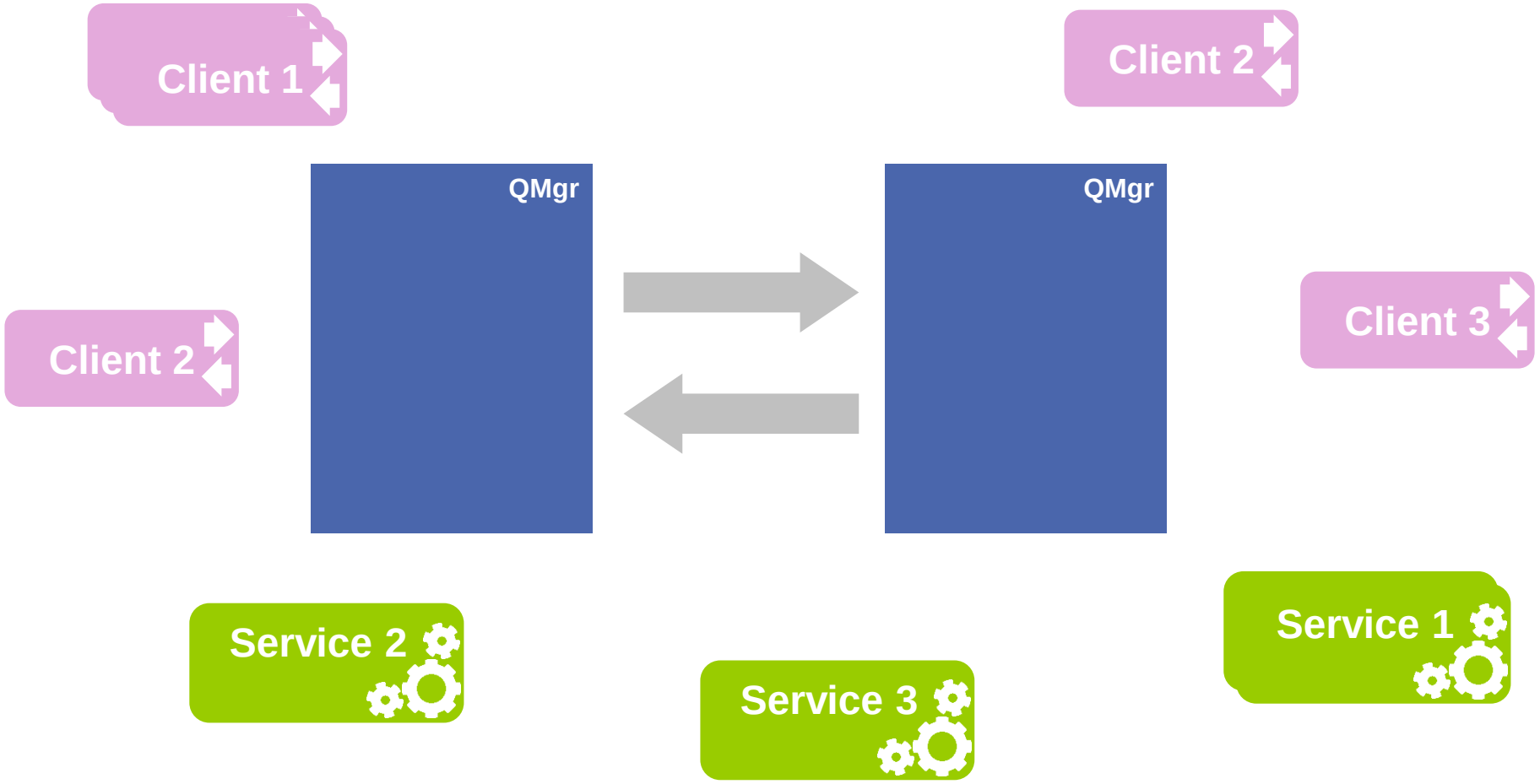
Where it all begins…



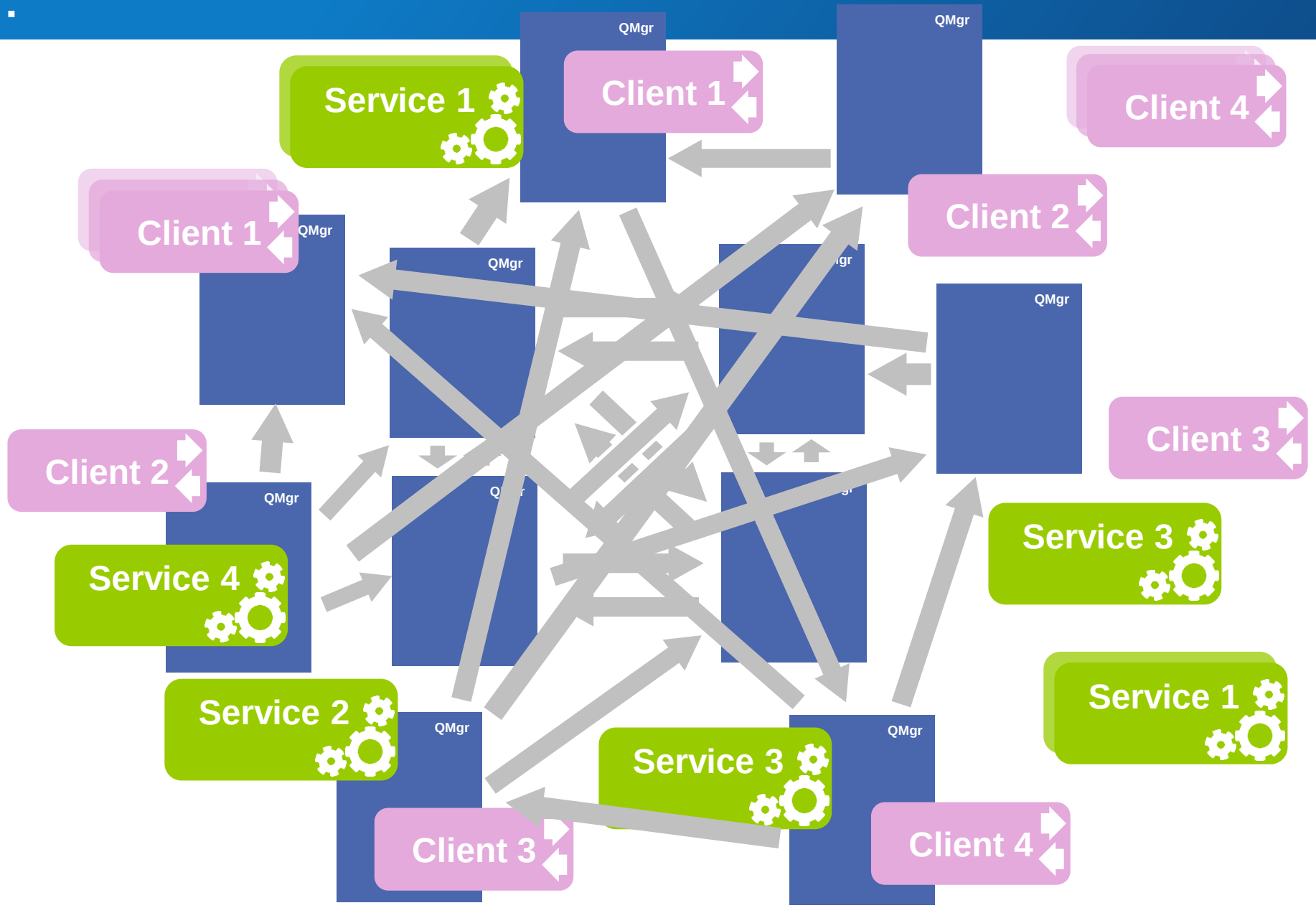
NOTES: Where it all begins…



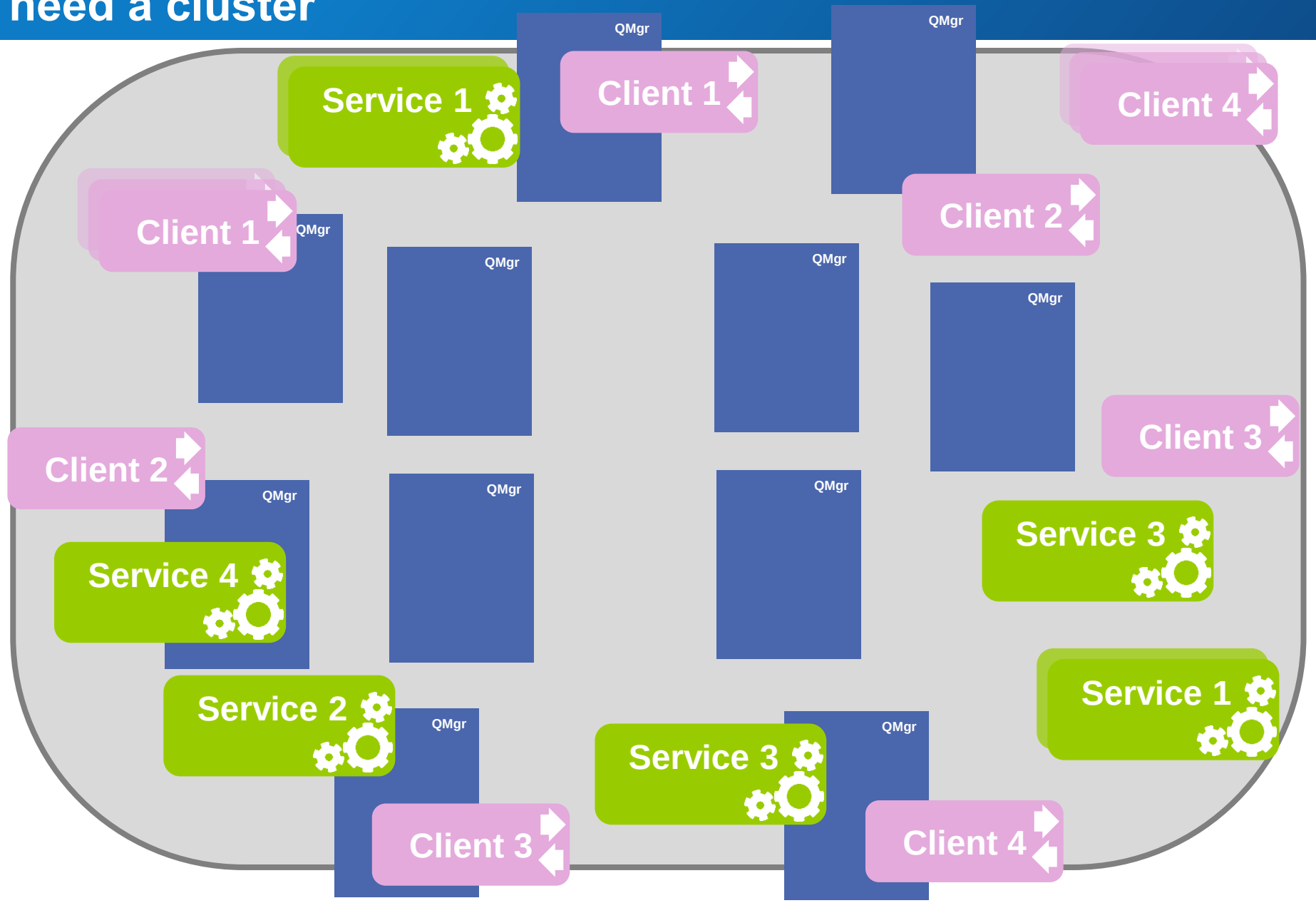
Over time…



Over time…



…Now you need a cluster

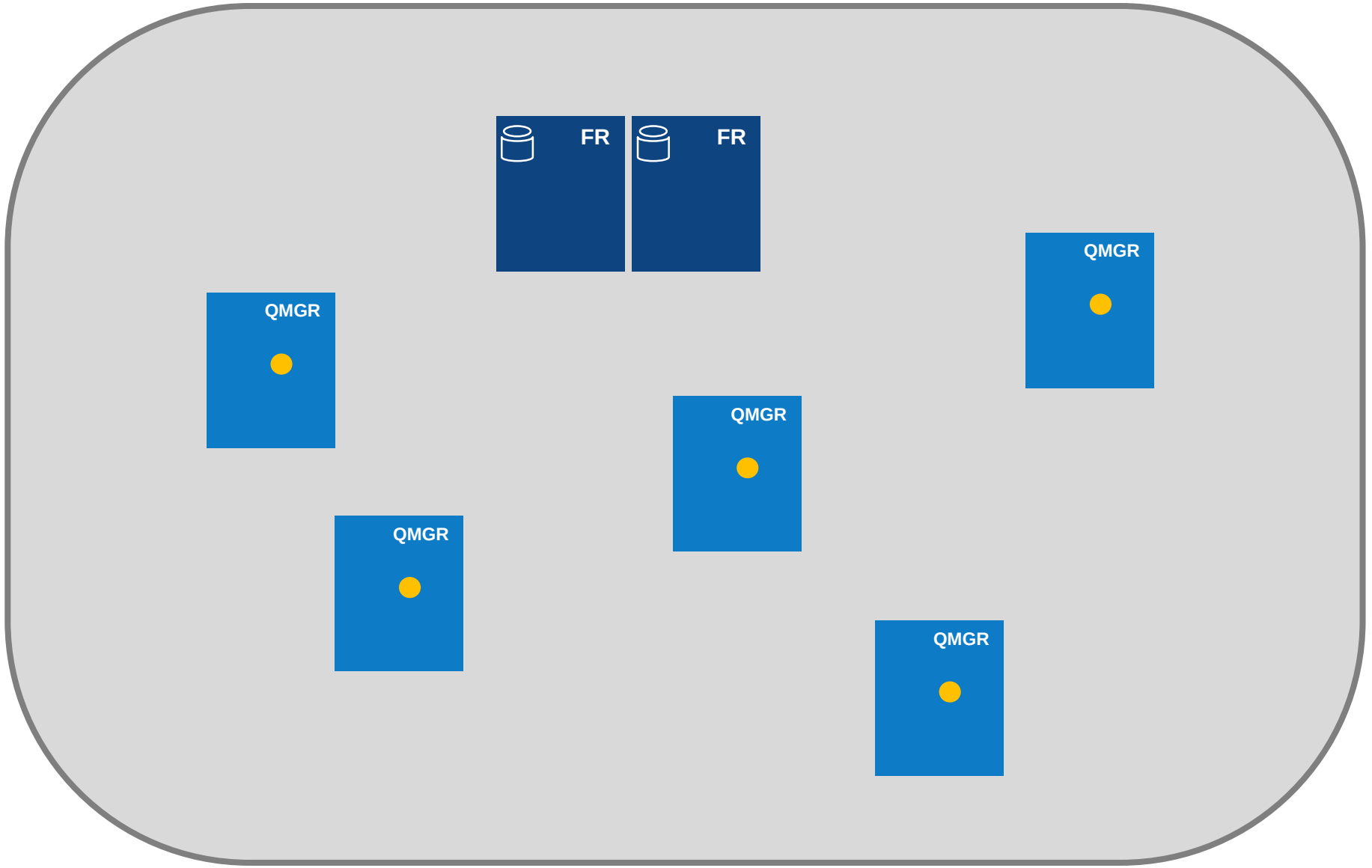


NOTES: Basic cluster

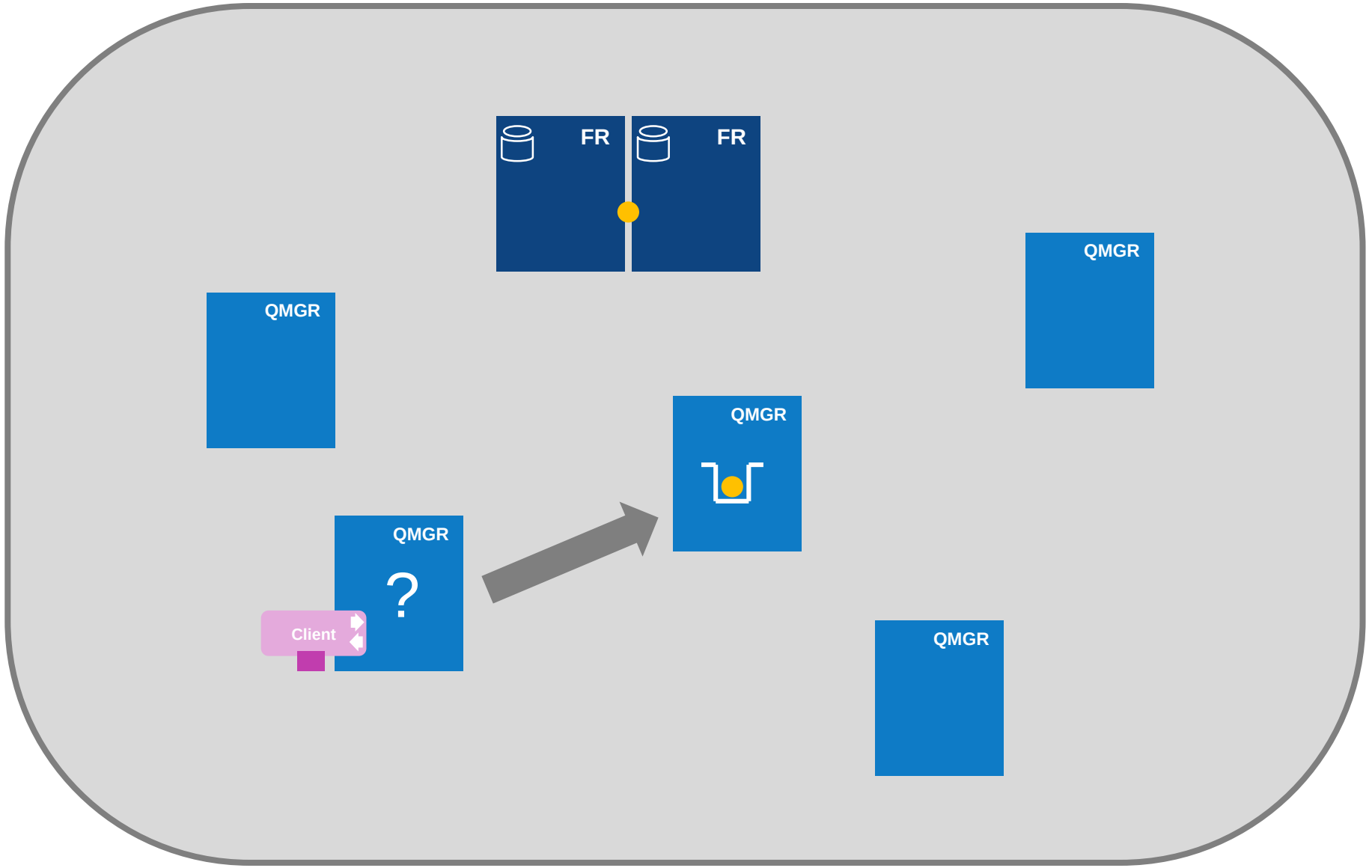
- This illustrates the first reason we may move to a IBM MQ cluster – simplified administration. Channel definitions - and, if we choose, the addressing of individual queues within the cluster - are tasks which no longer have to be manually carried out by the administrator.
- At this point we still only have single instances of ‘service’ queues.
- A degree of vertical scaling can be achieved by adding instances of the Service processes connecting to the single queue manager hosting a particular queue, if the application can cope with ‘interleaved’ processing.

How it works

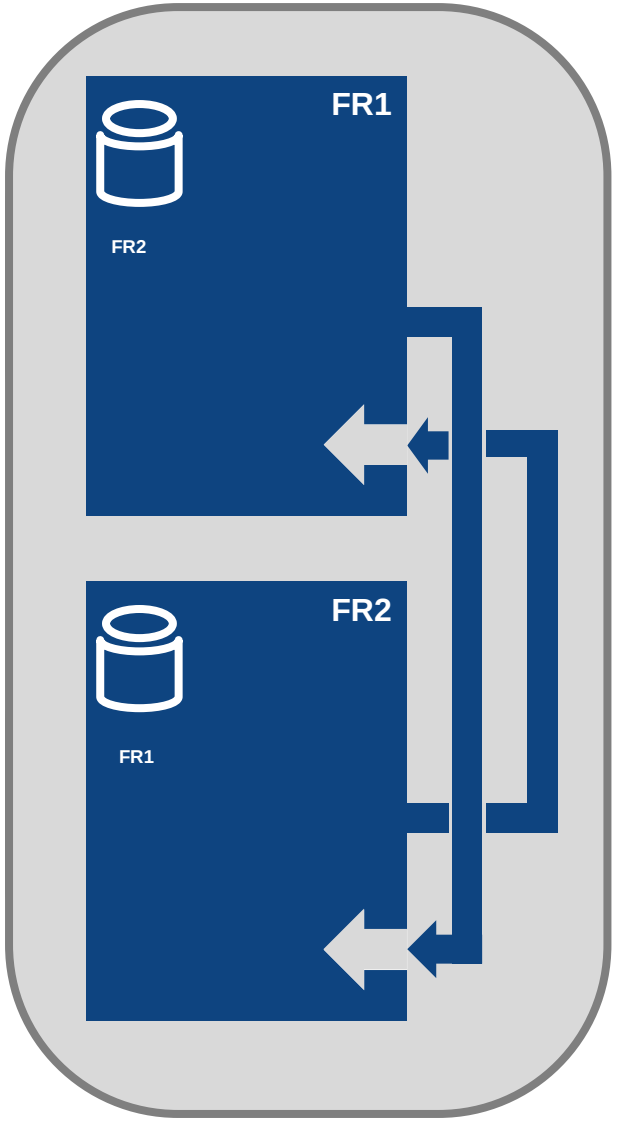
Building a cluster



Building a cluster



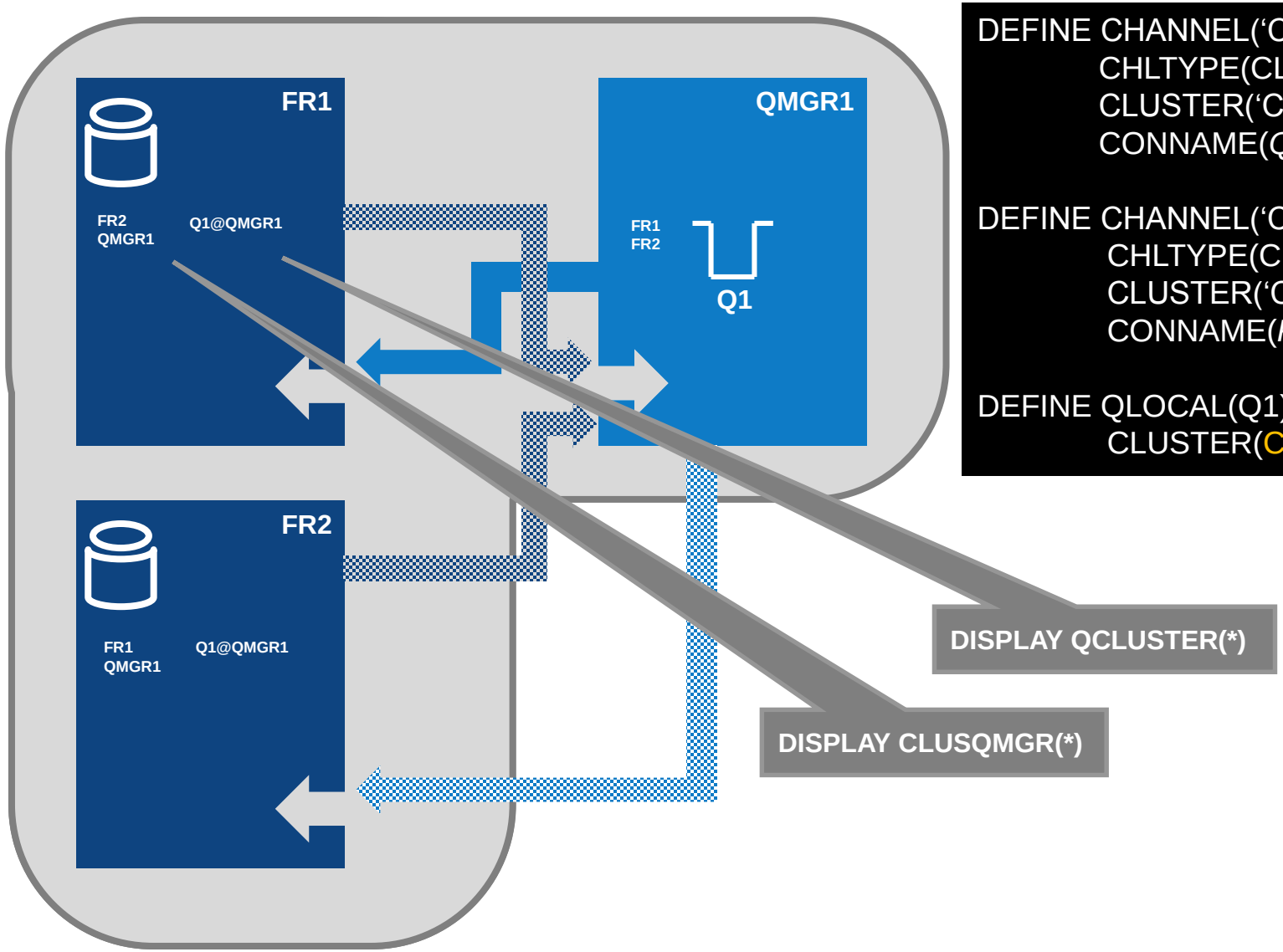
Step 1: Create your two full repositories




```
ALTER QMGR REPOS('CLUS1')

DEFINE CHANNEL('CLUS1.FR1') CHLTYPE(CLUSRCVR)
       CLUSTER('CLUS1') CONNAME(FR1 location)

DEFINE CHANNEL('CLUS1.FR2') CHLTYPE(CLUSSDR)
       CLUSTER('CLUS1') CONNAME(FR2 location)
```

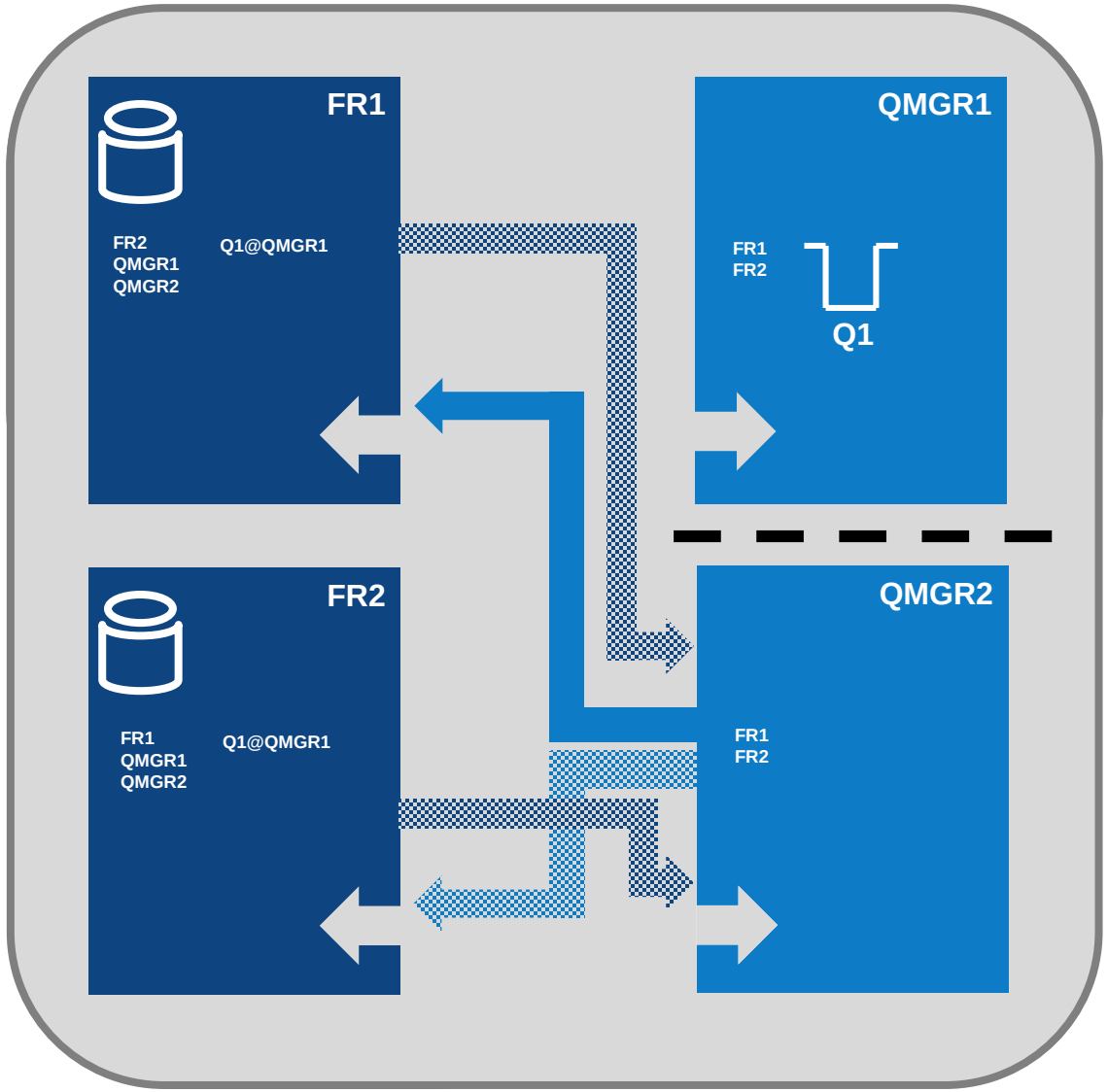
```
ALTER QMGR REPOS('CLUS1')

DEFINE CHANNEL('CLUS1.FR2') CHLTYPE(CLUSRCVR)
       CLUSTER('CLUS1') CONNAME(FR2 location)

DEFINE CHANNEL('CLUS1.FR1') CHLTYPE(CLUSSDR)
       CLUSTER('CLUS1') CONNAME(FR1 location)
```

Step 2: Add in more queue managers




```
DEFINE CHANNEL('CLUS1.QMGR1')
       CHLTYPE(CLUSRCVR)
       CLUSTER('CLUS1')
       CONNAME(QMGR1 location)

DEFINE CHANNEL('CLUS1.FR1')
       CHLTYPE(CLUSSDR)
       CLUSTER('CLUS1')
       CONNAME(FR1 location)

DEFINE QLOCAL(Q1)
       CLUSTER(CLUS1)
```

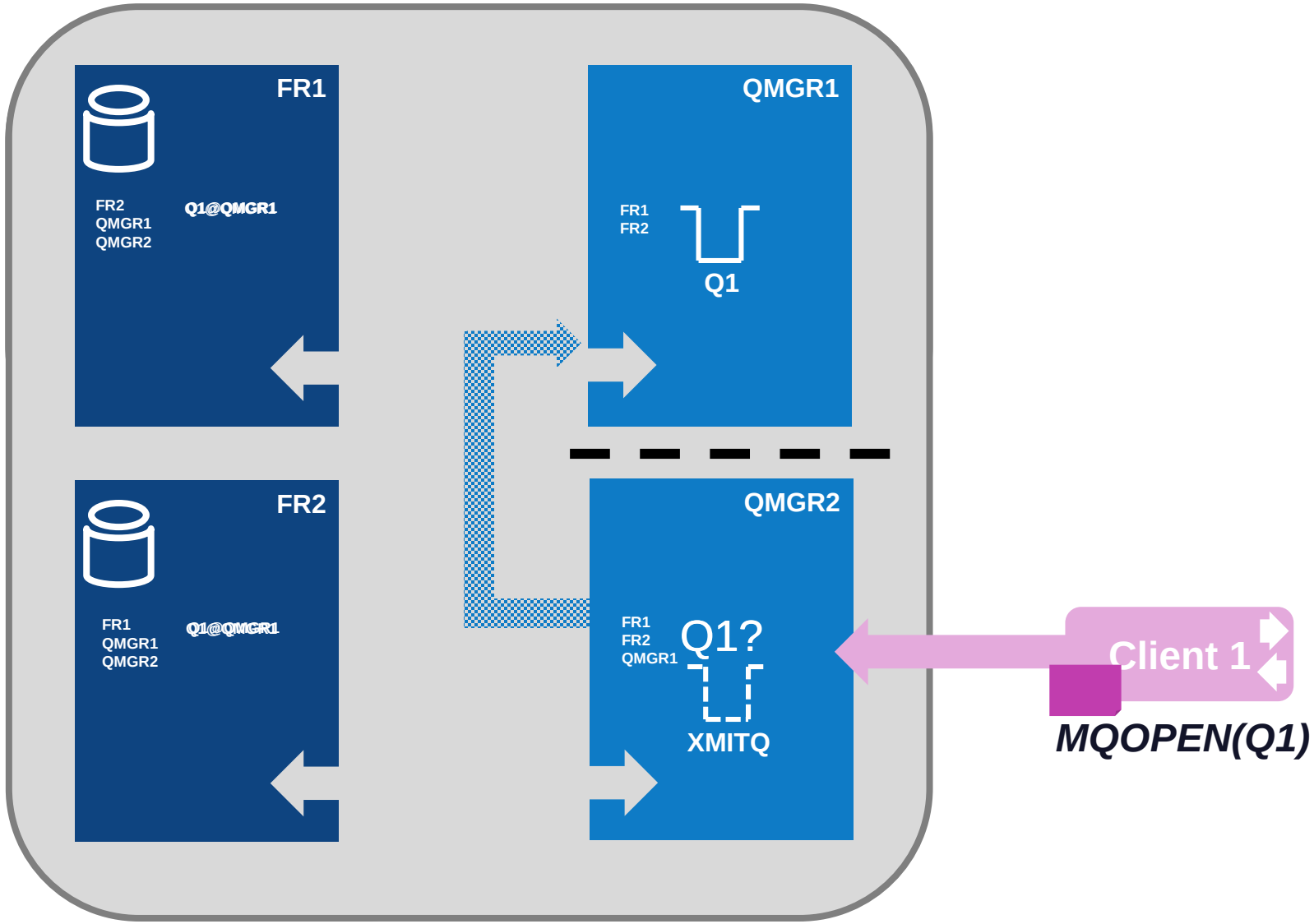
Step 2: Add in more queue managers



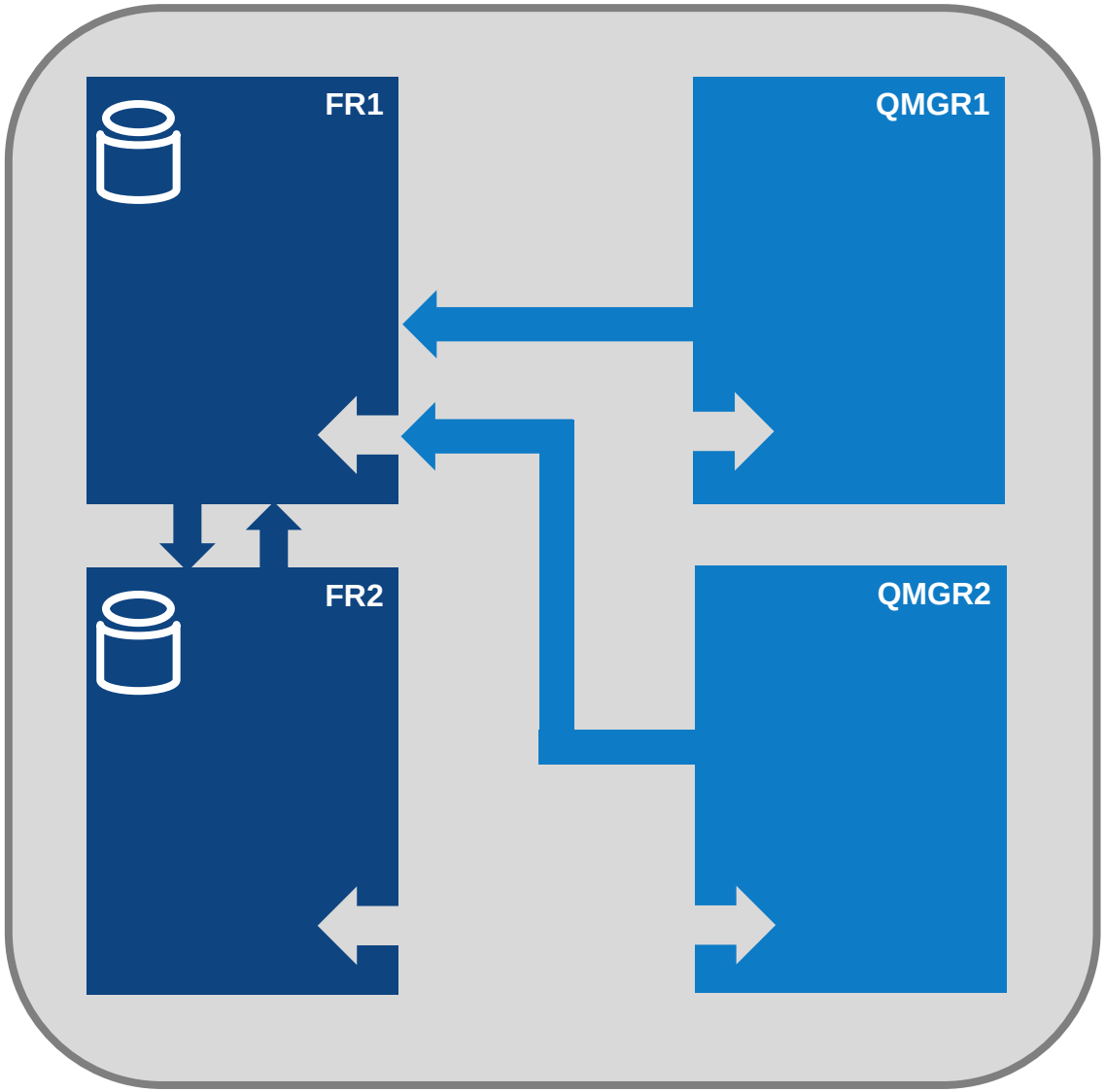

```
DEFINE CHANNEL('CLUS1.QMGR1')
       CHLTYPE(CLUSRCVR)
       CLUSTER('CLUS1')
       CONNAME(QMGR1 location)

DEFINE CHANNEL('CLUS1.FR1')
       CHLTYPE(CLUSSDR)
       CLUSTER('CLUS1')
       CONNAME(FR1 location)

DEFINE QLOCAL(Q1)
       CLUSTER(CLUS1)
```

```
DEFINE CHANNEL('CLUS1.QMGR2')
       CHLTYPE(CLUSRCVR)
       CLUSTER('CLUS1')
       CONNAME(QMGR2 location)

DEFINE CHANNEL('CLUS1.FR1')
       CHLTYPE(CLUSSDR)
       CLUSTER('CLUS1')
       CONNAME(FR1 location)
```

Step 3: Start sending messages



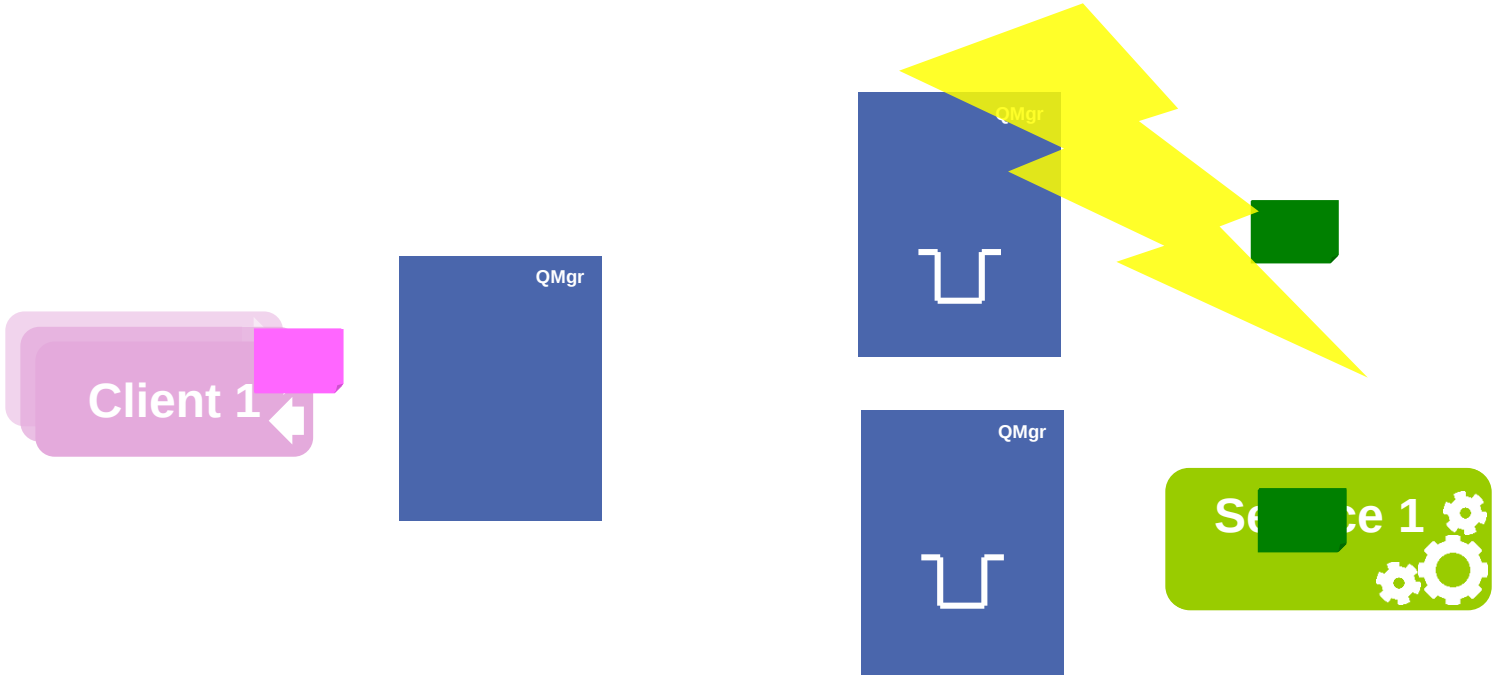
So all you needed…



- Two full repository queue managers
- A cluster receiver channel each
- A single cluster sender each
- No need to manage pairs of channels between each queue manager combination or their transmission queues
- No manual starting of channels
- No need for remote queue definitions or transmission queues

But that's just for starters…

But what else do you get with a cluster?

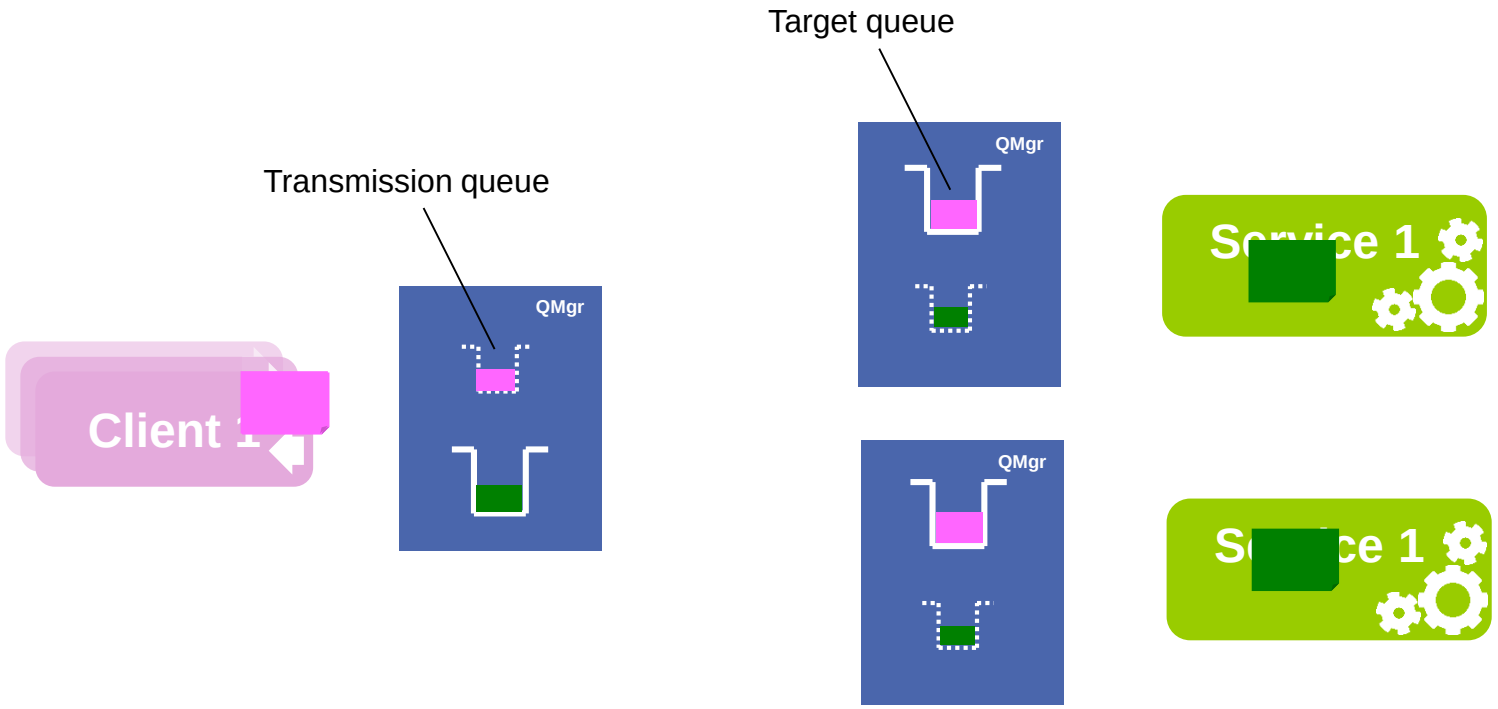


- Workload Balancing
- Service Availability

NOTES: Starting to scale horizontally

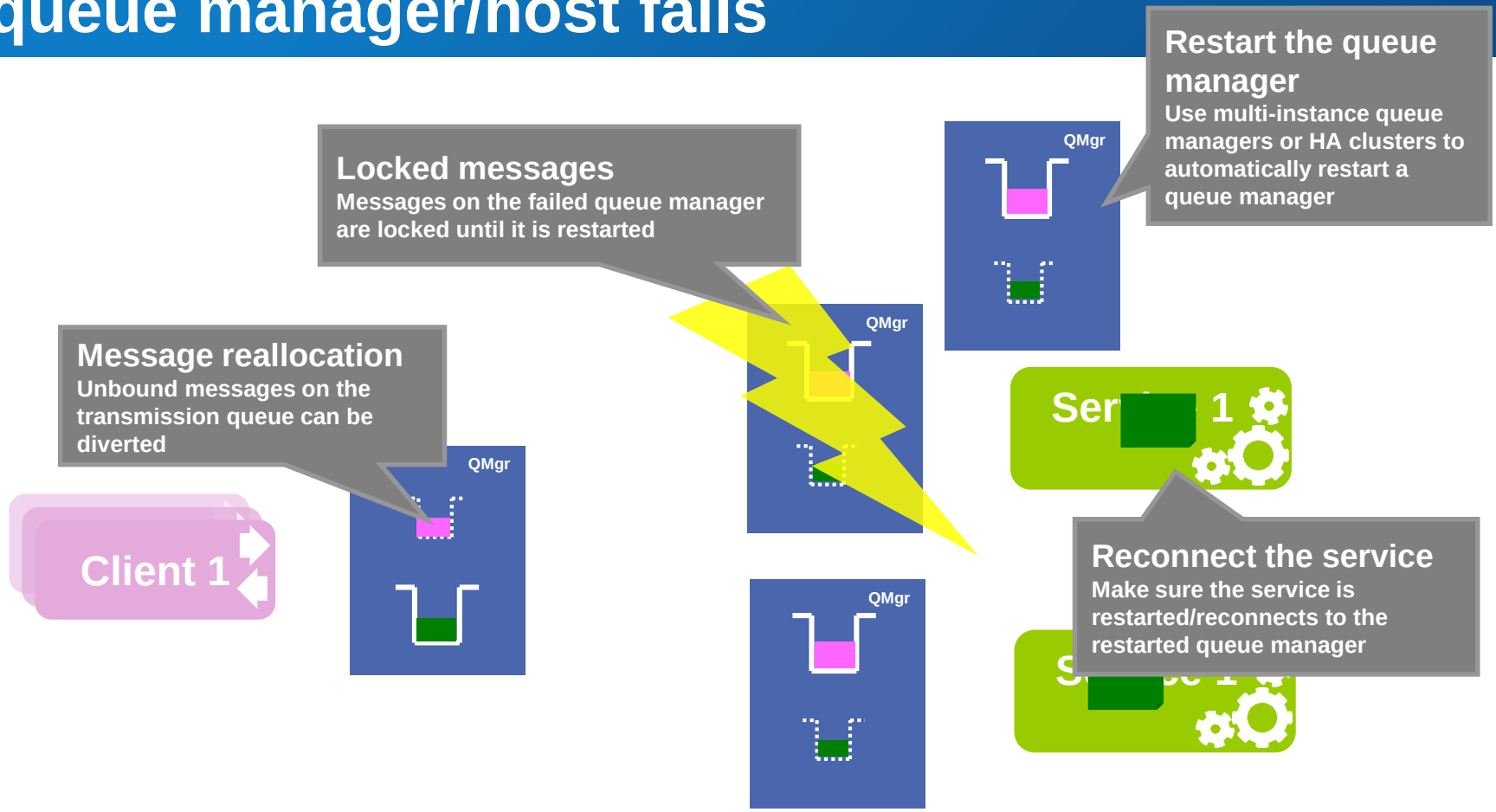
- **By adding instances of ‘service’ queues, we can start to scale applications across multiple hosts and beyond the ‘vertical’ scaling on a single (albeit maybe multi-CPU) system.**
- **Using IBM MQ Clustering allows instances to be added and removed without modifying the client or server applications**
  - But may require some thinking ahead to design with this in mind – e.g. avoiding hard coded queue manager names
- **As soon as we have multiple instances, questions arise about ‘choosing’ an instance for a request, so as a natural consequence workload balancing becomes available**
- **Location transparency for ‘the service’ has been achieved at this point, but there is still a strong coupling between ‘an instance’ and a particular queue manager – for example for reply routing (see next section)**

Where can the messages get stuck?



- Target queues
- Transmission queues

The service queue manager/host fails



When a queue manager fails:

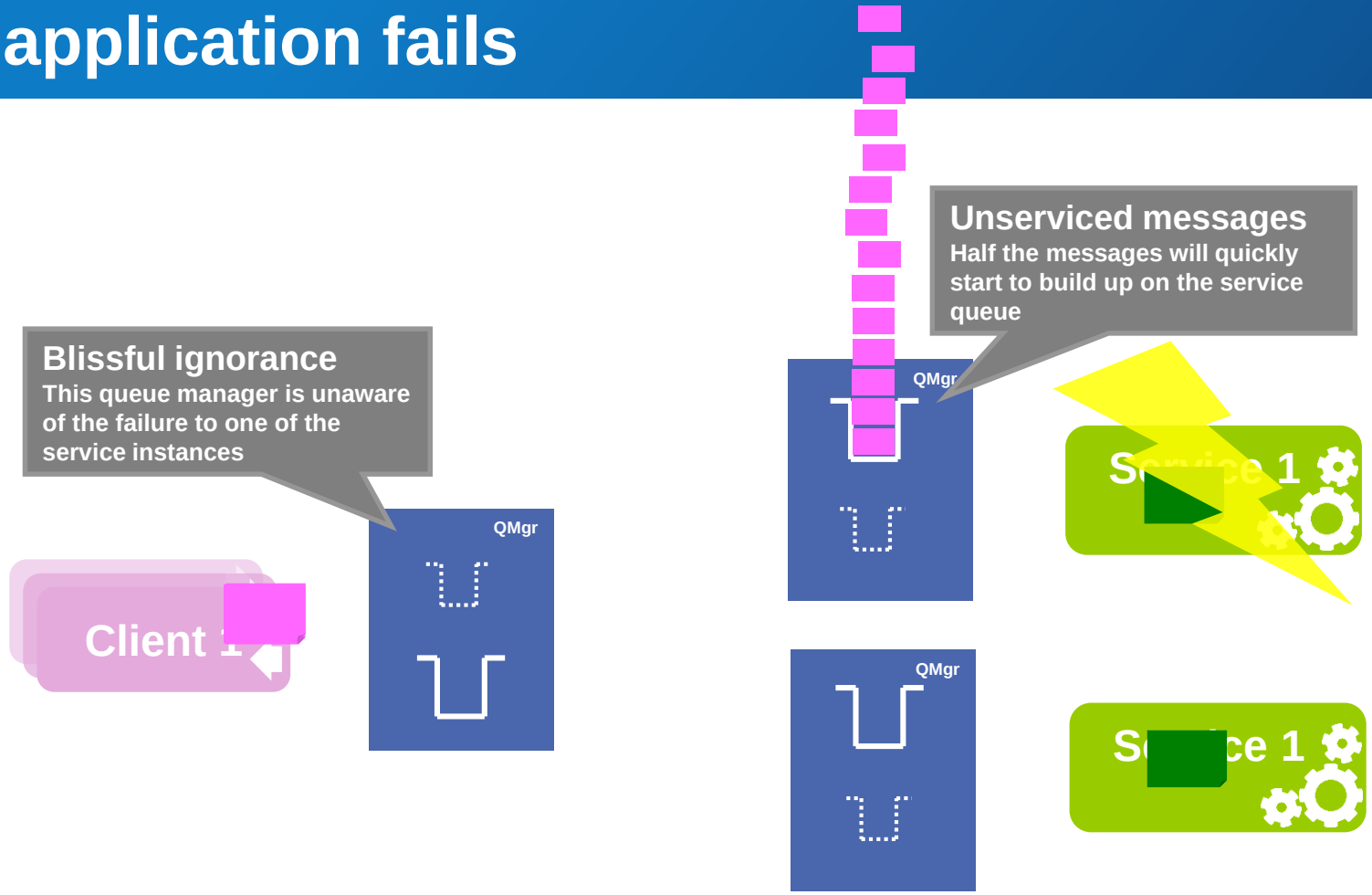
- Ensure messages are not **bound** to it
- Restart it to release queued messages

NOTES: Failed service queue manager

- When a ‘service’ queue manager fails, request messages which have reached that queue manager or responses on transmission queues are inevitably lost until that queue manager can be restarted.
- However – service availability can be maintained by making sure that there is sufficient capacity in other hosts to cope with all requests being loaded onto them.
- This will be smoother and give higher availability if client applications can be designed to avoid server affinity and strict message ordering requirements – BIND_NOT_FIXED. Reallocation will then mean that even in-flight requests can be re-routed.
- To avoid the trapped request problem, consider HA clustering technology or multi-instance queue managers.

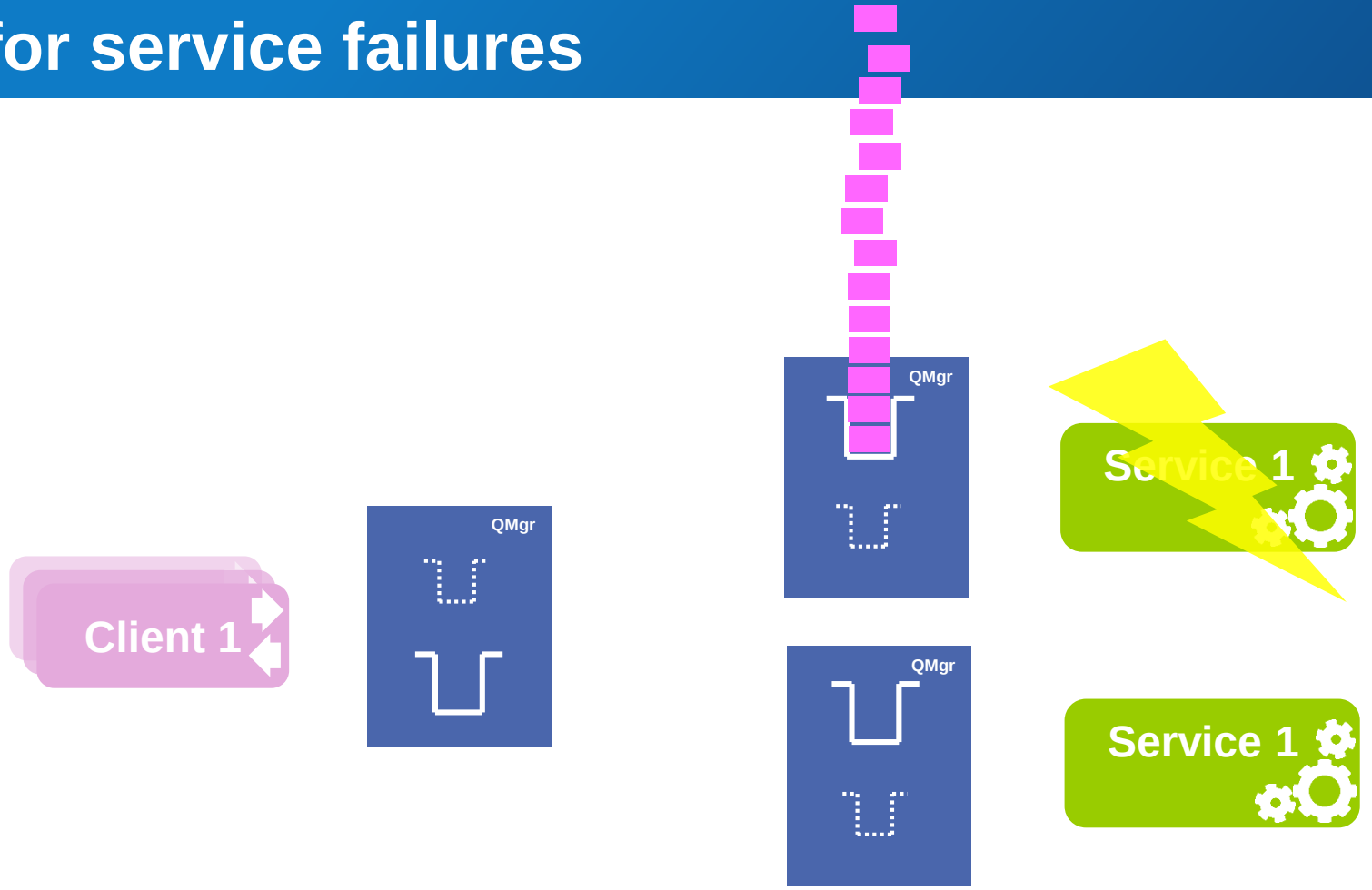
Service application availability

The service application fails

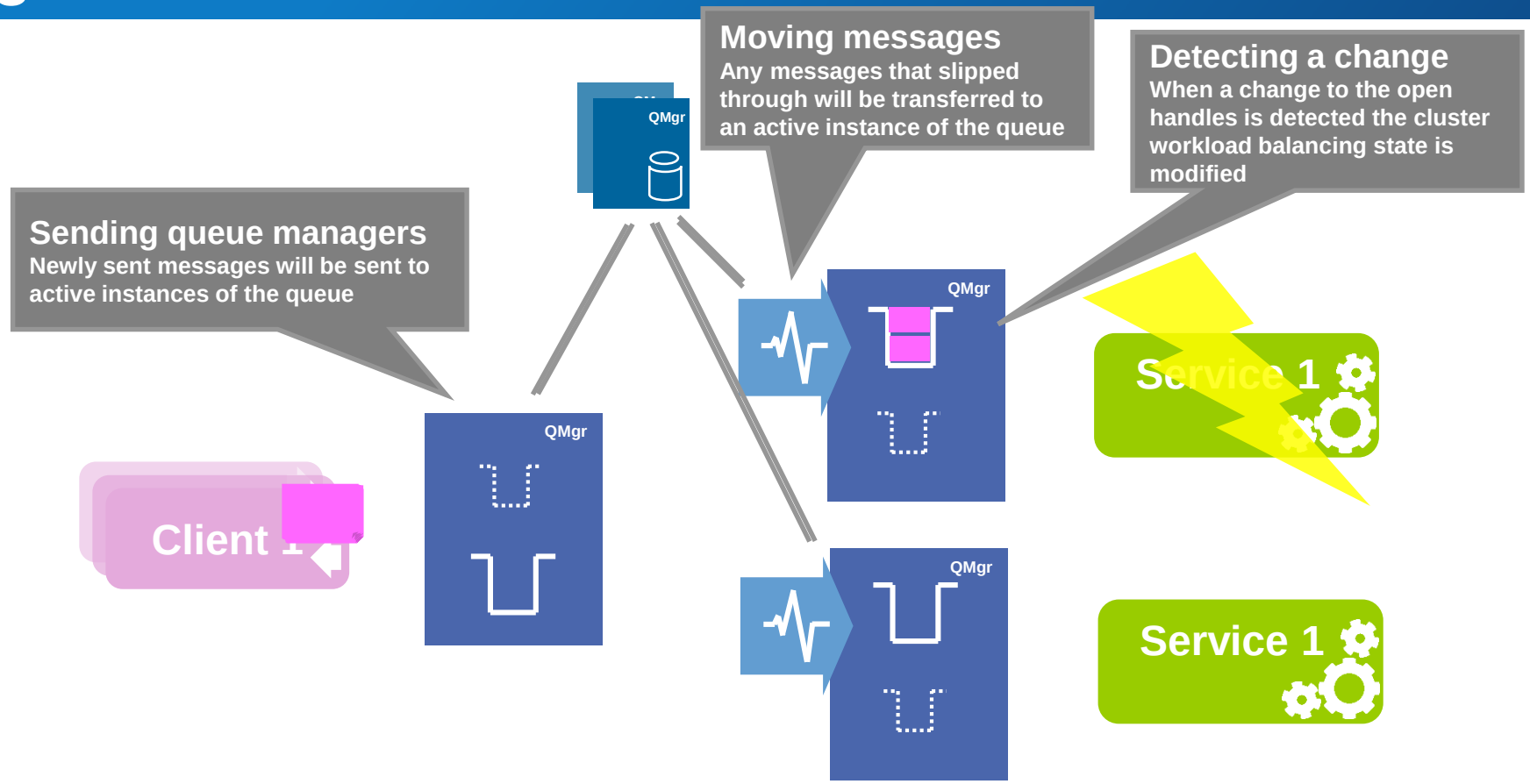


- Cluster workload balancing does not take into account the availability of receiving applications
- Or a build up of messages

Monitoring for service failures



Monitoring for service failures



- MQ provides a sample monitoring service
- Regularly checks for attached consuming applications
- Generally suited to ***steady state*** service applications

NOTES: Cluster queue monitoring sample

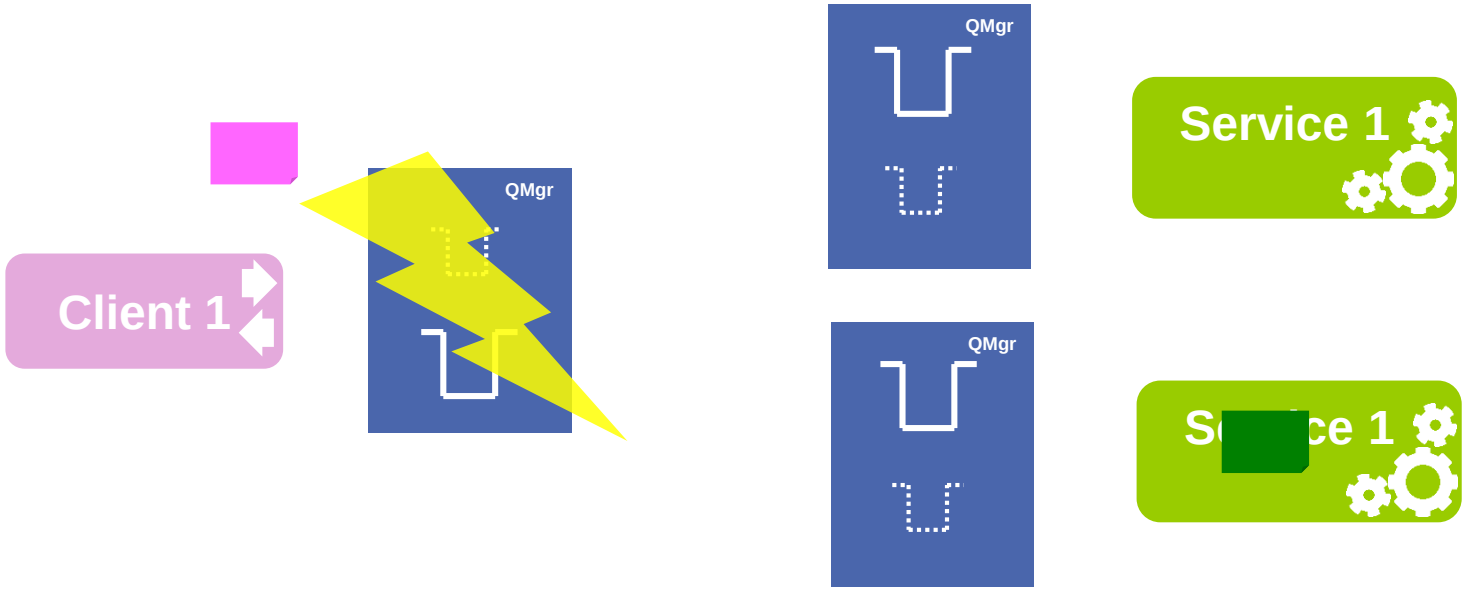
- **amqsclm, is provided with MQ to ensure messages are directed towards the instances of clustered queues that have consuming applications currently attached. This allows all messages to be processed effectively even when a system is asymmetrical (i.e. consumers not attached everywhere).**
  - **In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

NOTES: AMQSCLM logic

- **Based on the existing MQ cluster workload balancing mechanics:**
  - Uses cluster priority of individual queues – all else being equal, preferring to send messages to instances of queues with the highest cluster priority (CLWLPRTY).
  - Using CLWLPRTY always allows messages to be put to a queue instance, even when no consumers are attached to any instance.
  - Changes to a queue's cluster configuration are automatically propagated to all queue managers in the cluster that are workload balancing messages to that queue.
- **Single executable, set to run against each queue manager with one or more cluster queues to be monitored.**
- **The monitoring process polls the state of the queues on a defined interval:**
  - If **no** consumers are attached:
    - CLWLPRTY of the queue is set to zero (if not already set).
    - The cluster is queried for any active (positive cluster priority) queues.
    - If they exist, any queued messages on this queue are got/put to the same queue. Cluster workload balancing will re-route the messages to the active instance(s) of the queue in the cluster.
  - If consumers **are** attached:
    - CLWLPRTY of the queue is set to one (if not already set).
- **Defining the tool as a queue manager service will ensure it is started with each queue manager**

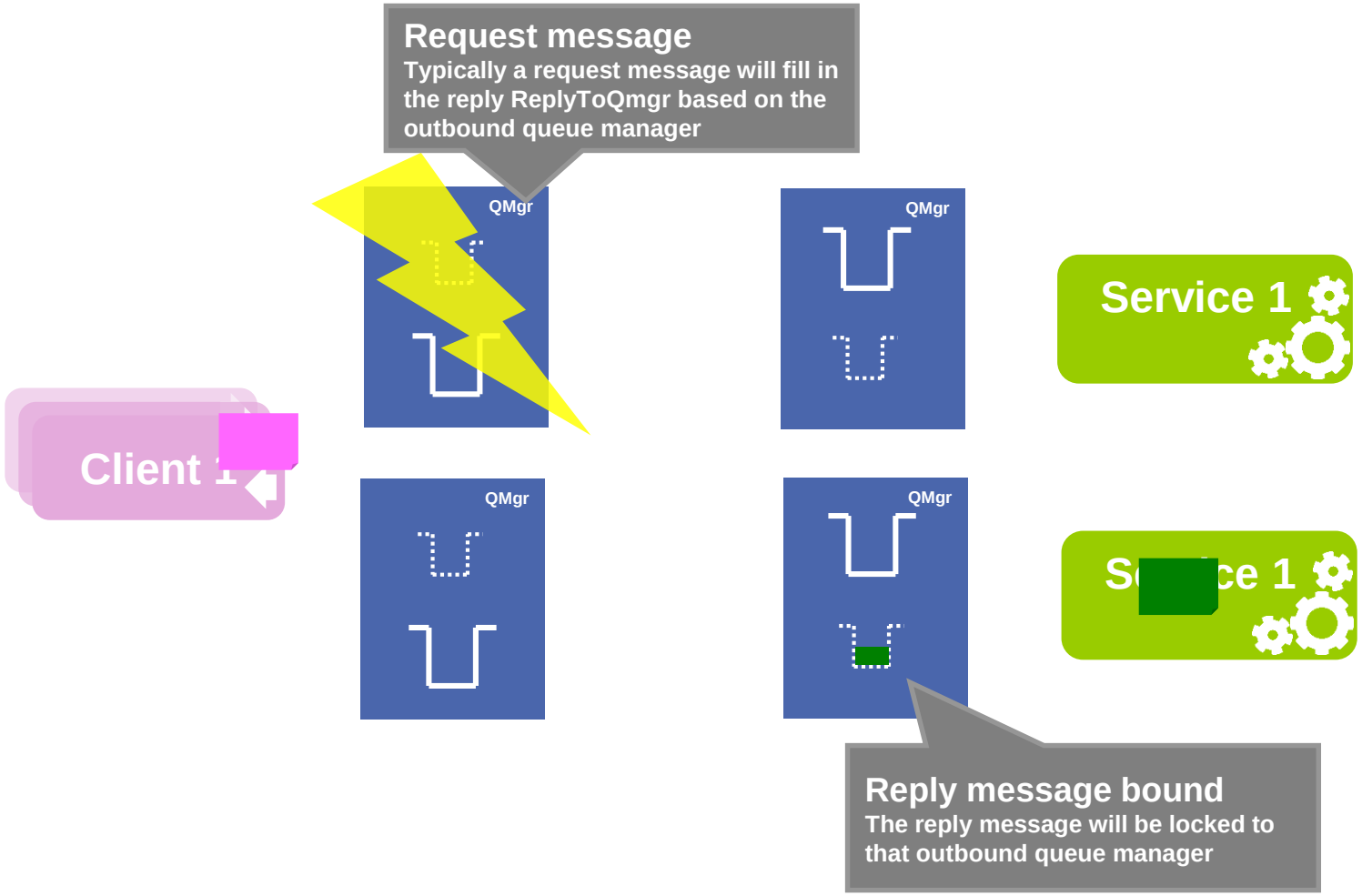
Client failures

Client availability



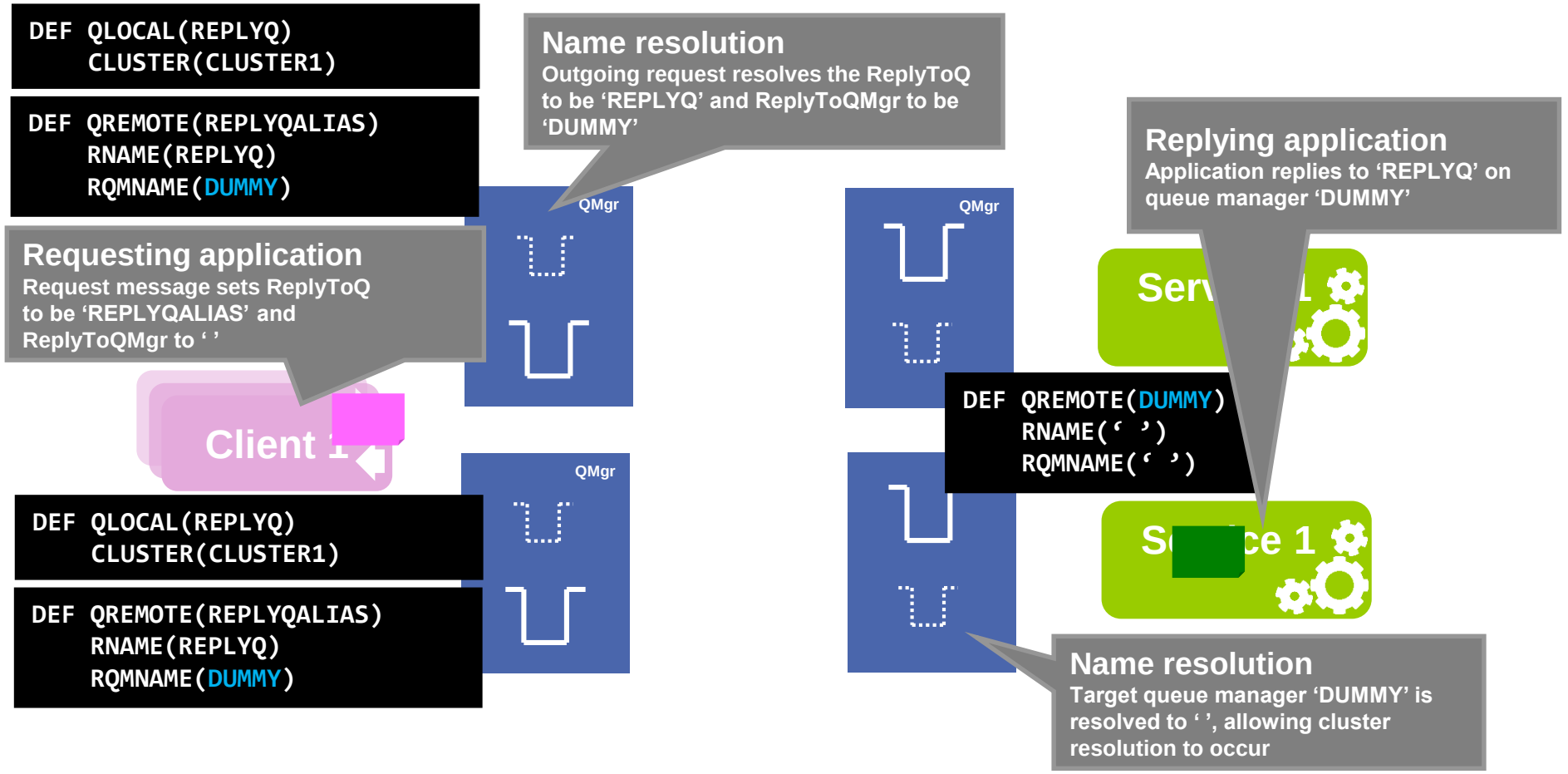
- Multiple locations for a client to connect to
  - Allows new requests when one queue manager is unavailable.
- Replies can be automatically routed back to the originating queue manager.

NOTES: Client host failure with an in flight request/response



- Reply messages are bound to the originating queue manager, with no ability to redirect.

NOTES: Client host failure with an in flight request/response



- ***Reply-to queue aliases*** and ***reply-to queue manager aliases*** can be used to blank out the outbound resolution of the ReplyToQMgr field.
- *Typically, under normal running, you require the originating queue manager to receive the replies, cluster workload balancing configuration from before can help to provide this.*

NOTES: Failed ‘client’ queue manager

- Traditional applications will use ReplyToQMgr which has been set on outgoing request. So may need to consider ReplyToQueueAlias to route response through workload balancing.

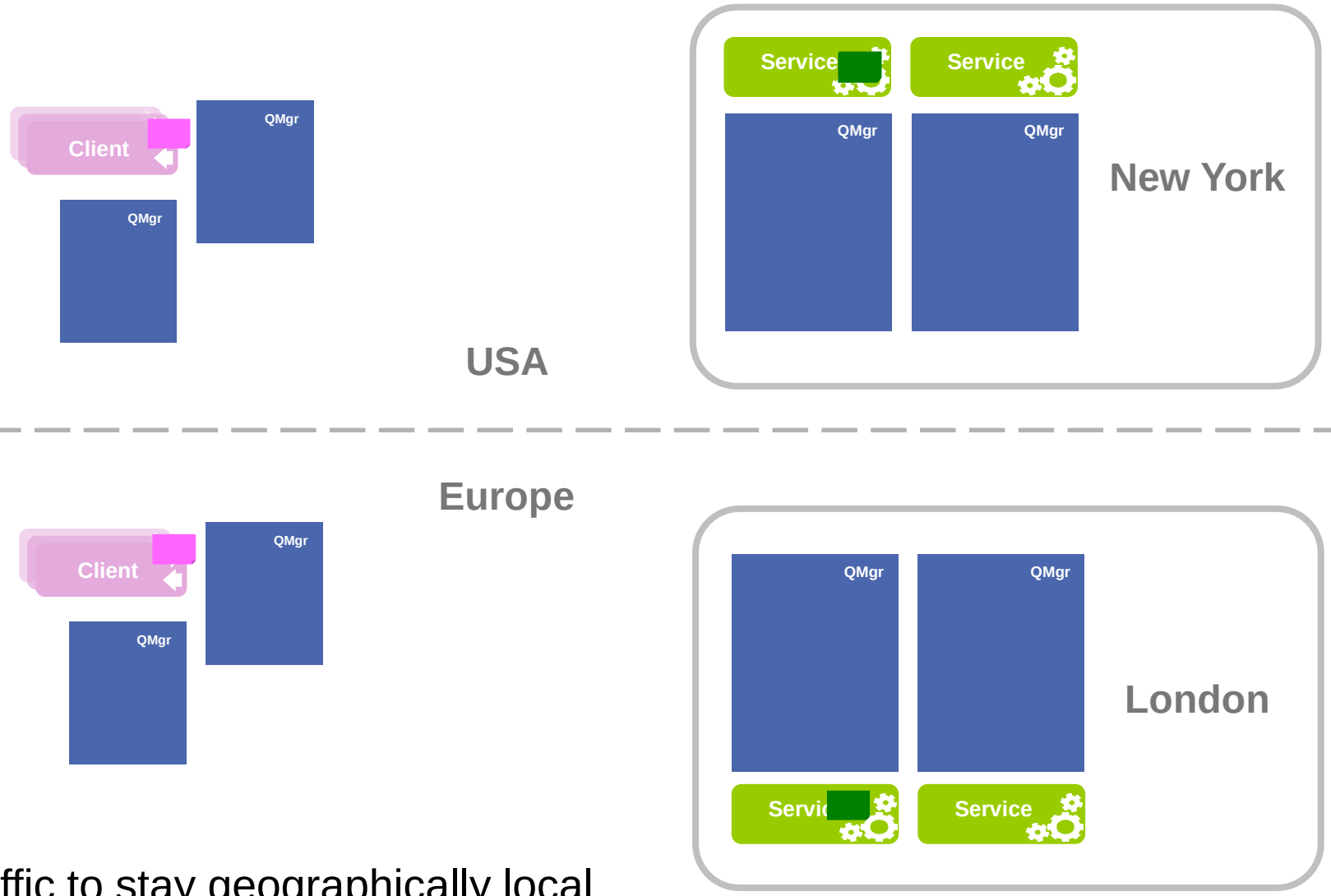
- Managing reconnection beyond scope of this session, and in ideal world will reconnect to same queue manager (may involve HA clusters, multi instance queue managers), however…
- Clustered reply queues give various possibilities. Simplest case is ‘shared responses’ but not really worth discussing further… lets assume need to get back to particular client ‘instance’.

1) Can use priority to prefer ‘usual’ location. Using some form of polling perhaps, ensure client connects / reconnects to particular queue manager whenever it is up. If down, client and replies fail over to backup.

2) OR: can use AMQSCLM again to get replies to follow connection

Smarter routing

Global applications

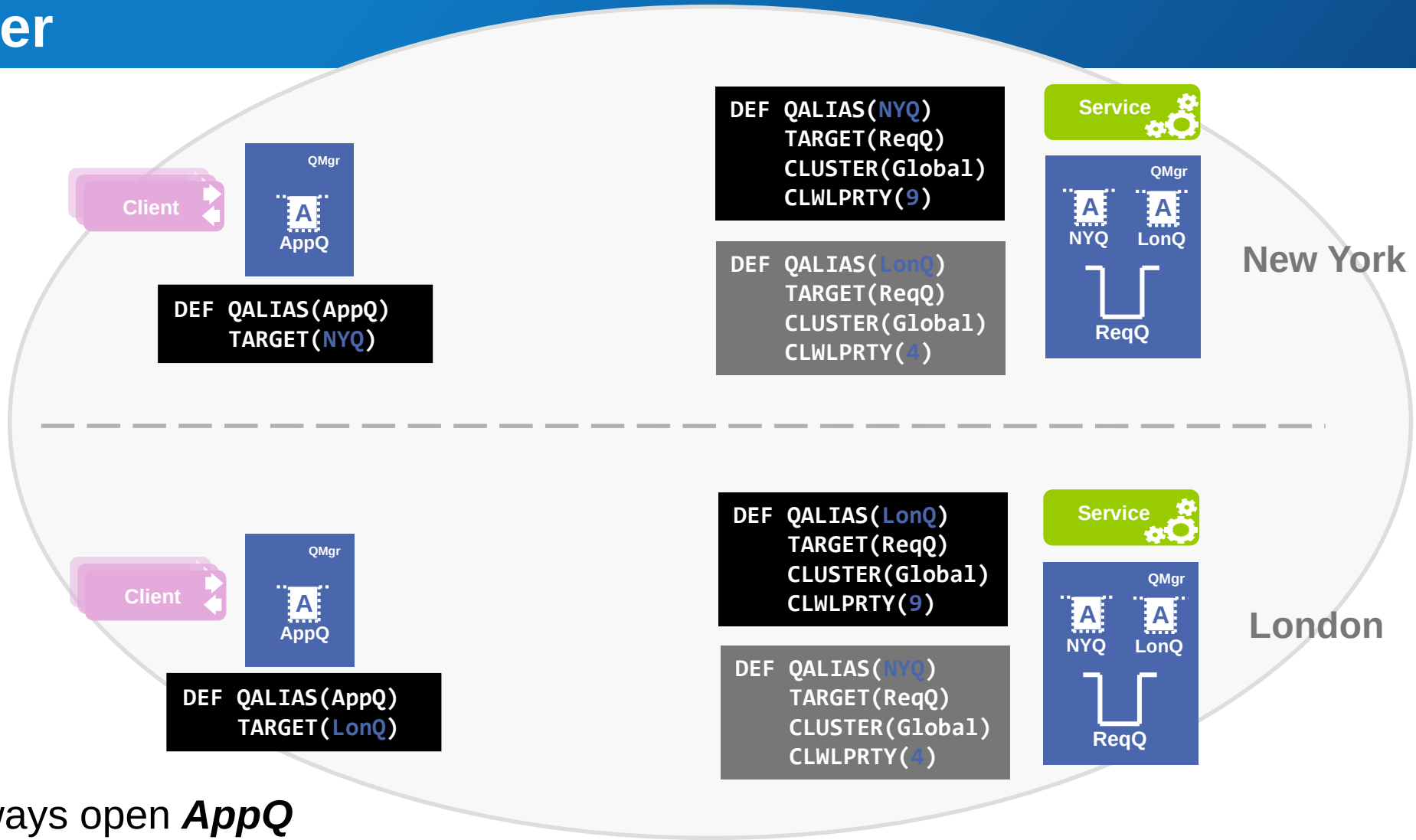


- Prefer traffic to stay geographically local
- Except when you have to look further afield
- How do you achieve this with clusters?

NOTES: One cluster

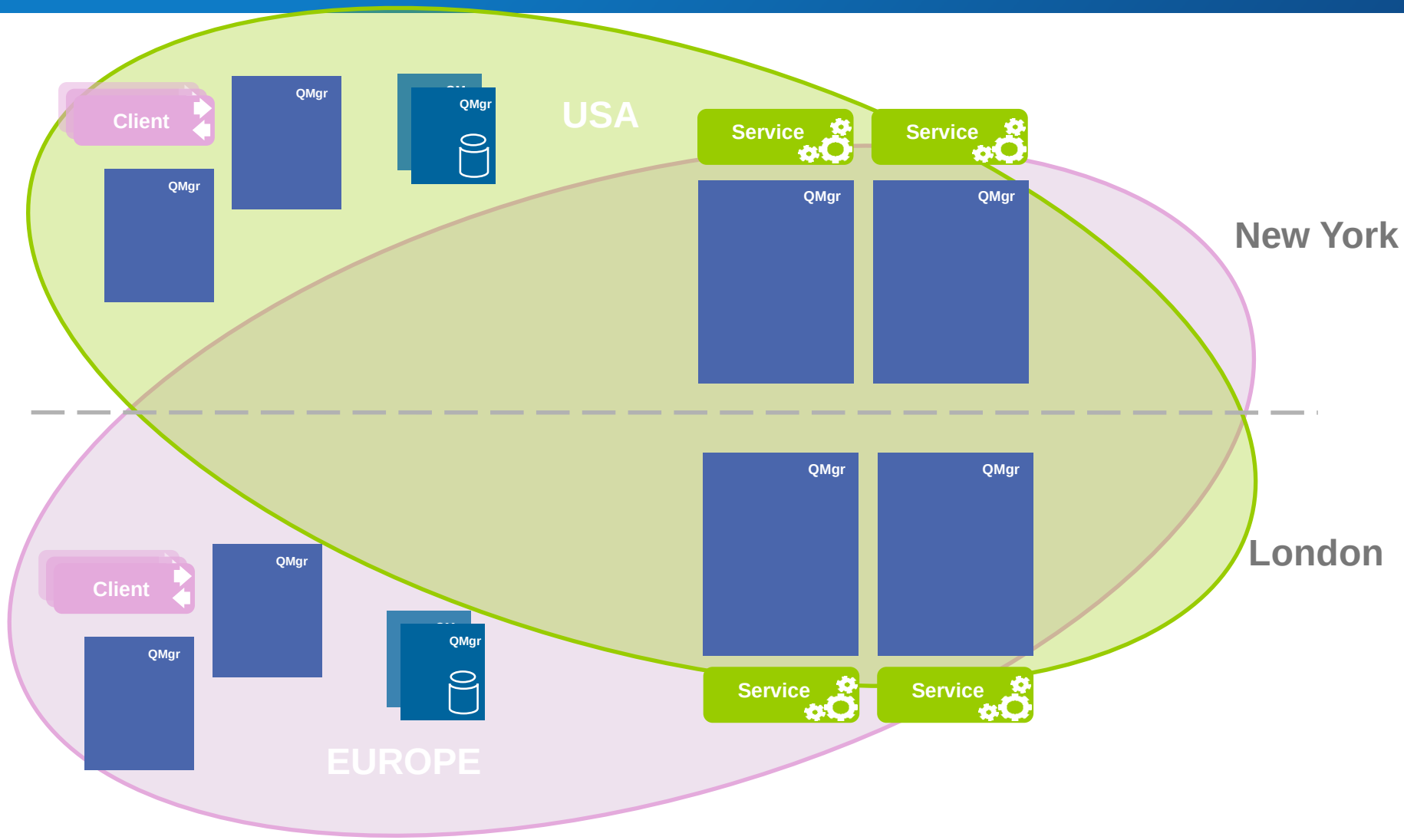
- A single cluster is often the simplest and best approach even when large distances are involved – for example, cluster certainly doesn't have to be limited to a particular datacenter.
- However, often for obvious reasons we would rather keep as much traffic as possible 'local', and we would like to know that if we lose our connection to the outside world for a period of time, things can keep running.
- Conversely though, if a particular service is down locally, we'd like to make use of the remote instance (even if it may be a bit slower than usual).
- Finally, we'd like our applications to 'look the same' wherever we connect – the deploying administrator might know this instance is running in London, but does the application really have to be coded to cope with that?

One cluster



- Clients always open ***AppQ***
- Local alias determines the preferred region
- Cluster workload priority is used to target geographically local cluster aliases
- Use of CLWLPRTY enables automatic failover (based on status of channel)
  - CLWLRANK can be used for manual failover (not based on status of channel)

The two cluster alternative



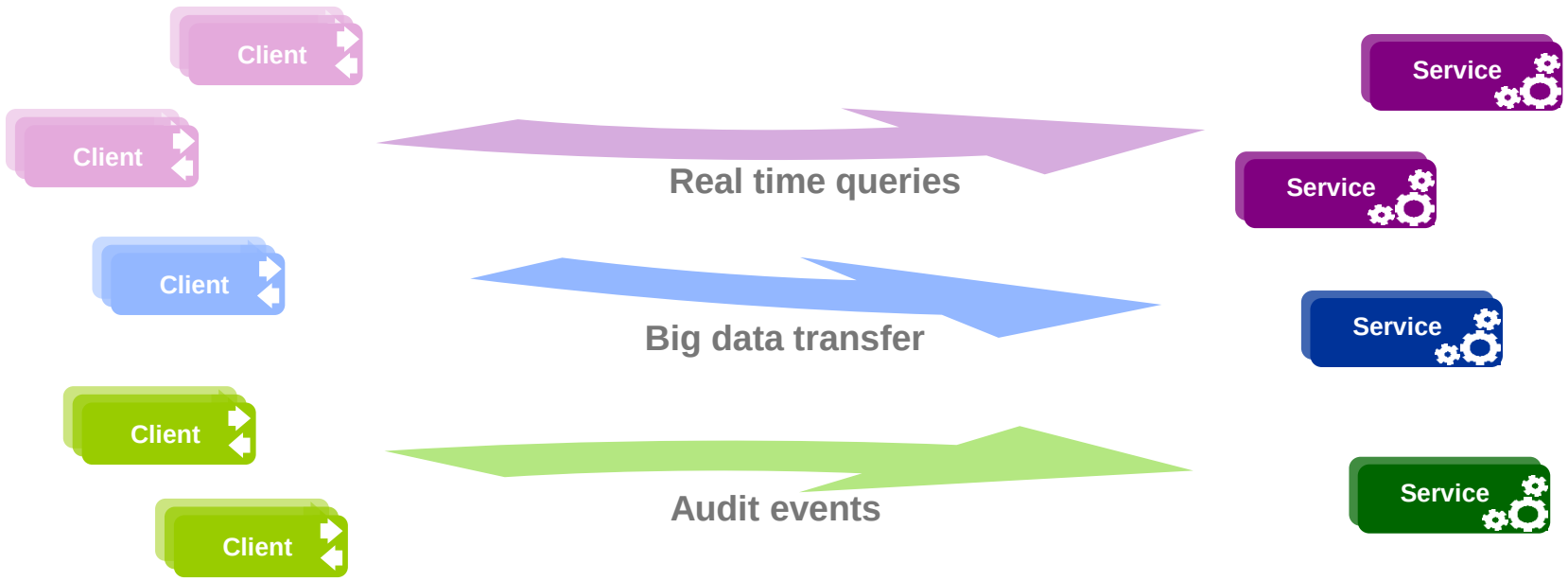
- The ***service*** queue managers join ***both*** geographical clusters
  - Each with separate cluster receivers for each cluster, at ***different cluster priorities***. Queues are clustered in ***both*** clusters.
- The ***client*** queue managers are in their ***local*** cluster only.

NOTES: Two cluster approach

- **Define two clusters.**
  - **USA** and **EUROPE**
- **For *service* queue managers, define separate cluster receiver channels, one for each cluster. Set the CLWLPRTY high on the one for the *local* cluster and low for the *remote* one.**
  - For service queue managers in New York:
    - DEFINE CHANNEL(**USA.NYxx**) CHLTYPE(CLUSRCVR) …. CLUSTER(**USA**) CLWLPRTY(**9**)
    - DEFINE CHANNEL(**EUROPE.NYxx**) CHLTYPE(CLUSRCVR) …. CLUSTER(**EUROPE**) CLWLPRTY(**4**)
  - For service queue managers in London:
    - DEFINE CHANNEL(**EUROPE.LONxx**) CHLTYPE(CLUSRCVR) …. CLUSTER(**EUROPE**) CLWLPRTY(**9**)
    - DEFINE CHANNEL(**USA.LONxx**) CHLTYPE(CLUSRCVR) …. CLUSTER(**USA**) CLWLPRTY(**4**)
- **Define a namelist of each service queue manager that contains both clusters and use this when clustering queues.**
    - DEFINE NAMELIST(**GLOBAL**) NAMES(**USA,EUROPE**)
    - DEFINE QLOCAL(QUEUE1) CLUSNL(**GLOBAL**)
- ***Client* queue managers only join the cluster that is local to them.**
- **The client queue managers will choose the instances of queues that are on queue managers with the highest CLWLPRTY on the channel.**
  - For example, a queue manager in the EUROPE cluster will only see the EUROPE.* channels. So London queue managers will have a CLWLPRTY of 9 and New York queue managers only 4, so preferring London whilst it is available.

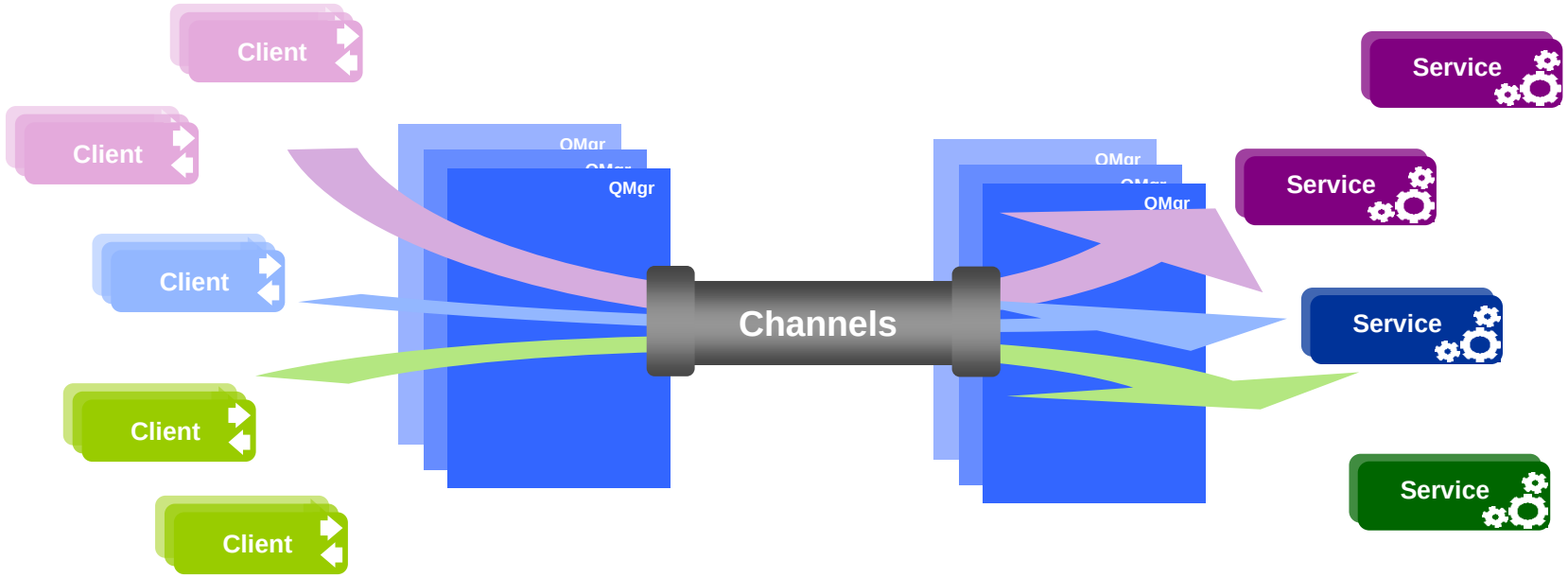
Separation of traffic

Multiple types of traffic



- Often a IBM MQ backbone will be used for multiple types of traffic

Multiple types of traffic



- Often a IBM MQ backbone will be used for multiple types of traffic
- When using a single cluster and the same queue managers, messages all share the same channels
- Even multiple cluster receiver channels in the same cluster will not separate out the different traffic types

NOTES: The cluster as the ‘pipe’ - problems

- **Mice and elephants**
  - Large non real time data is contending for resources with small ‘live’ request response transactions.
  - with due attribution to T-Rob:

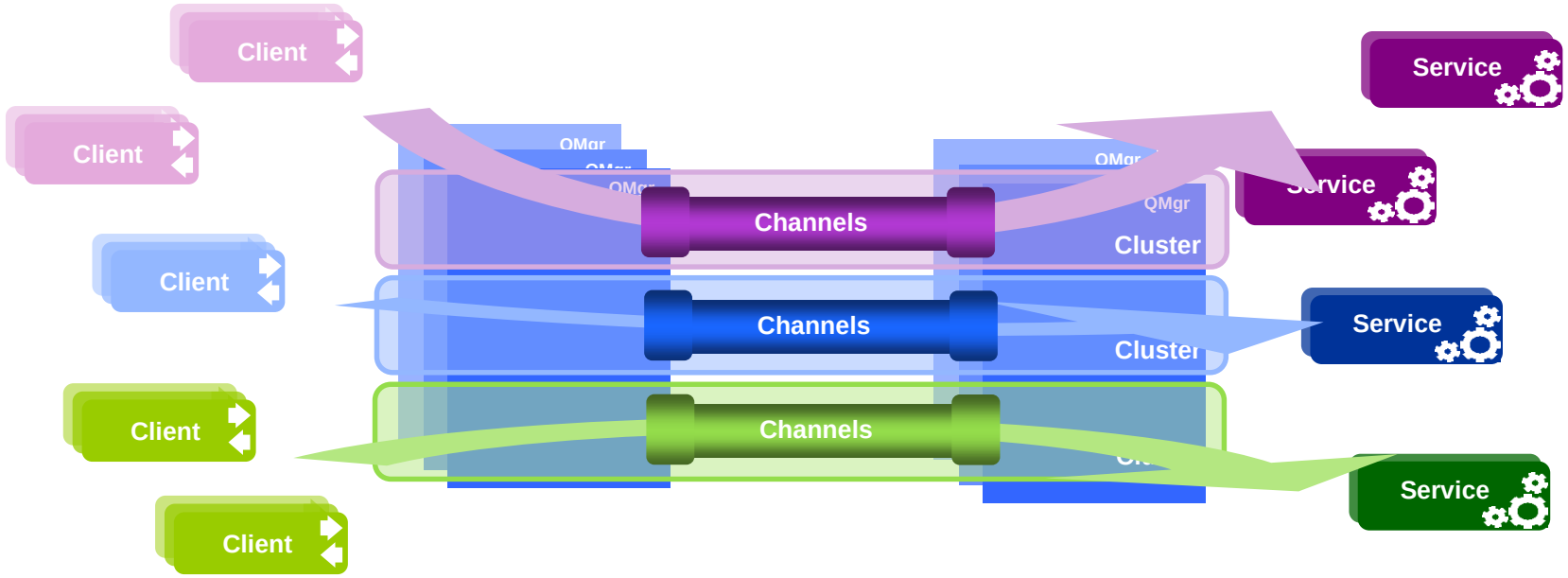
  https://www.ibm.com/developerworks/websphere/techjournal/0804_mismes/0804_mismes.html

- **All workload balancing at the messaging / channel level**
  - No distinction between a request that needs a week of CPU at the other end, and one which needs 1 ms.

- **Pub sub requires high ‘meshing’ – all queue managers aware of whole cluster**
  - Potentially lots of channel work for hosts not interested in pub sub activity when superimposed on existing cluster

- **Denial of service potential**
  - One application out of control = full cluster Transmit queue until someone can manually intervene

Multiple types of traffic



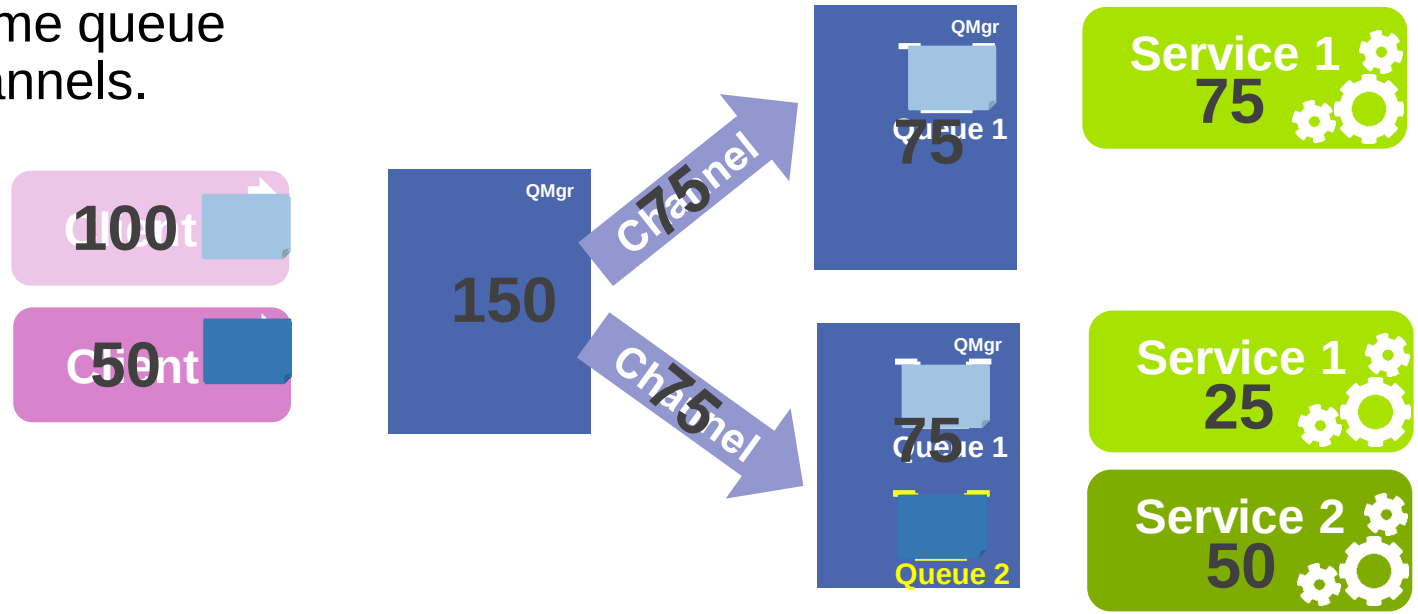
- Often a IBM MQ backbone will be used for multiple types of traffic
- When using a single cluster and the same queue managers, messages all share the same channels
- Even multiple cluster receiver channels in the same cluster will not separate out the different traffic types
- Multiple overlaid clusters with different channels enable separation

NOTES: The cluster as the ‘pipe’

- **Putting application in a separate cluster gives option of also giving it its own channel**
- **Applications with a need for a strictly controlled WLM ratio can be given their own clusters for this reason. However, bear in mind cost of too many overlapping clusters**
  - RFE 23391
- **In general, try and group applications with similar requirements rather than ending up with channel for every application**
  - Real time / Batch / Pub sub
- **Applications don’t need to know which cluster their resources are in as long as configuration is managed correctly on their behalf**
- **New in WebSphere MQ 7.1: Pub sub can be limited to specialised clusters / queue managers using PSCLUS attribute**
- **New in WebSphere MQ 7.5 (V8 for z/OS and IBM i): Channels for different clusters can also be separated at transmission queue level**

Workload balancing level interference

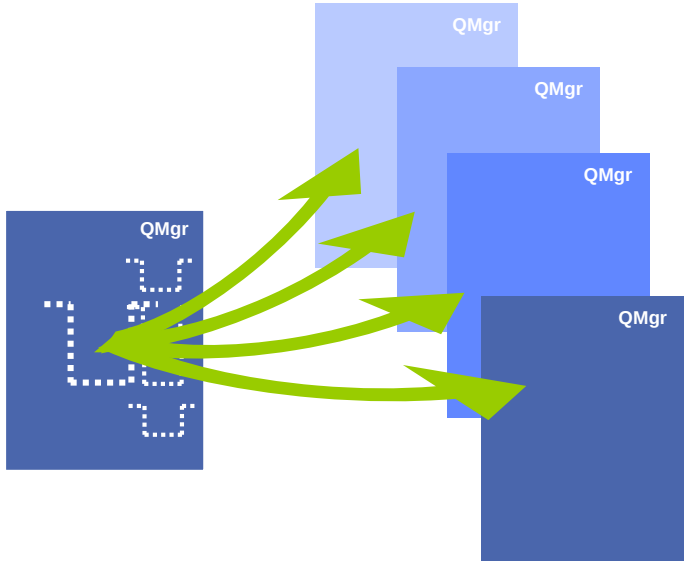
- Multiple applications sharing the same queue managers and the same cluster channels.



- Cluster workload balancing is at the **channel** level
  - Messages sharing the same channels, but to different target queues will be counted together.
- The two channels here have an even 50/50 split of messages…
- *…but the two instances of Service 1 do not!*
- *Split Service 1 and Service 2 queues out into separate clusters, queue managers or customise workload balancing logic.*

Cluster transmit queues

- **The default is a single shared transmission queue for all of a queue manager's cluster traffic**
- **MQ V7.5/V8.0 added multiple cluster transmission queue support**



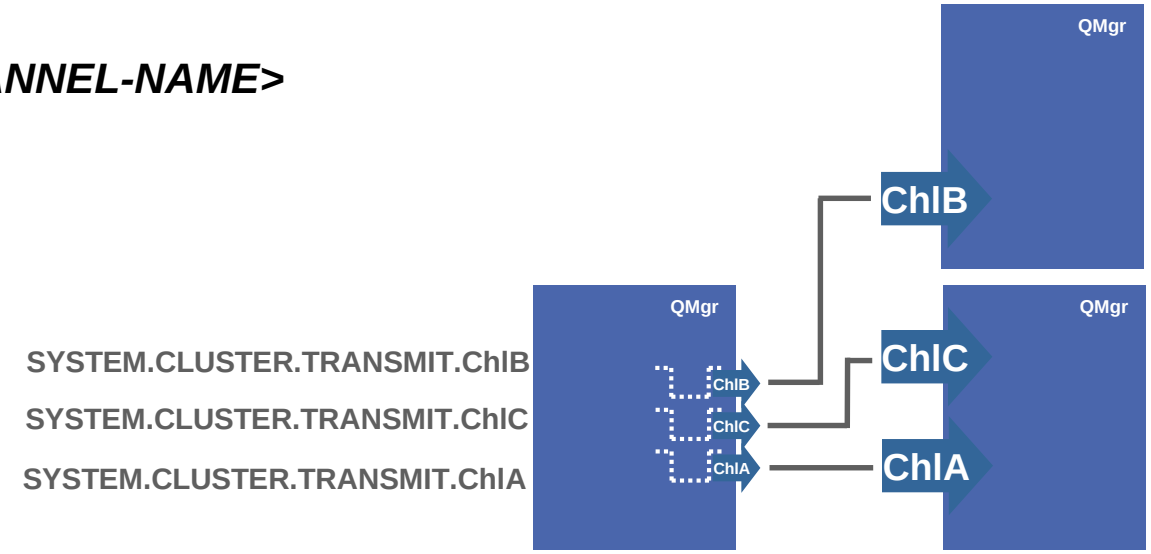
- **Separation of message traffic**
  - With a single transmission queue there is potential for pending messages for cluster *ChannelA* to interfere with messages pending for cluster *ChannelB. For example if the target qmgr is down*
- **Management of messages**
  - Use of queue concepts such as *MAXDEPTH* 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.
- **Performance?**
  - In reality a shared transmission queue is not always the bottleneck, often other solutions to improving channel throughput *(e.g. multiple cluster receiver channels)* are really what's needed.

Multiple cluster transmit queues: automatic

- **Configured on the *sending queue manager*, not the owners of the cluster receiver channel definitions.**

- **Queue manager switch to automatically create a permanent-dynamic transmission queue per cluster sender channel.**
  ALTER QMGR DEFCLXQ(CHANNEL)
- **Dynamic queues based upon model queue. SYSTEM.CLUSTER.TRANSMIT.MODEL**
- **Well known queue names.**

  SYSTEM.CLUSTER.TRANSMIT.*<CHANNEL-NAME>*



Multiple cluster transmit queues: manual

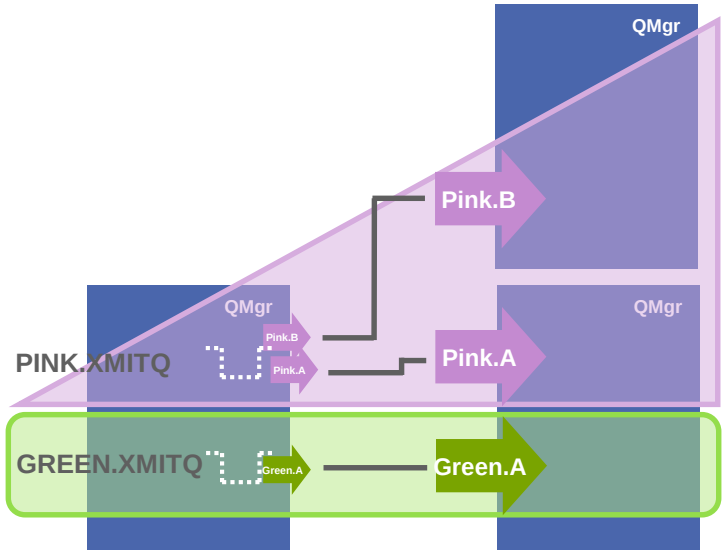
- ***Still* configured on the *sending queue manager*, not the owners of the cluster receiver channel definitions.**

- **Administratively define a transmission queue and configure which cluster sender channels will use this transmission queue.**

  DEFINE QLOCAL(GREEN.XMITQ) CLCHNAME(GREEN.*) USAGE(XMITQ)

  - Set a channel name *pattern* in CLCHNAME
  - Single/multiple channels (wildcard)
    - E.g. all channels for a specific cluster (assuming a suitable channel naming convention!)

- **Any cluster sender channel not covered by a manual transmission queue defaults to the DEFCLXQ behaviour**



Summary

- **Why a cluster?**
  - Starting to scale and setting one up
- **Service availability**
  - Routing around failures
- **Location dependency**
  - Smarter routing
- **Levels of separation**
  - What are we sharing?

Questions?