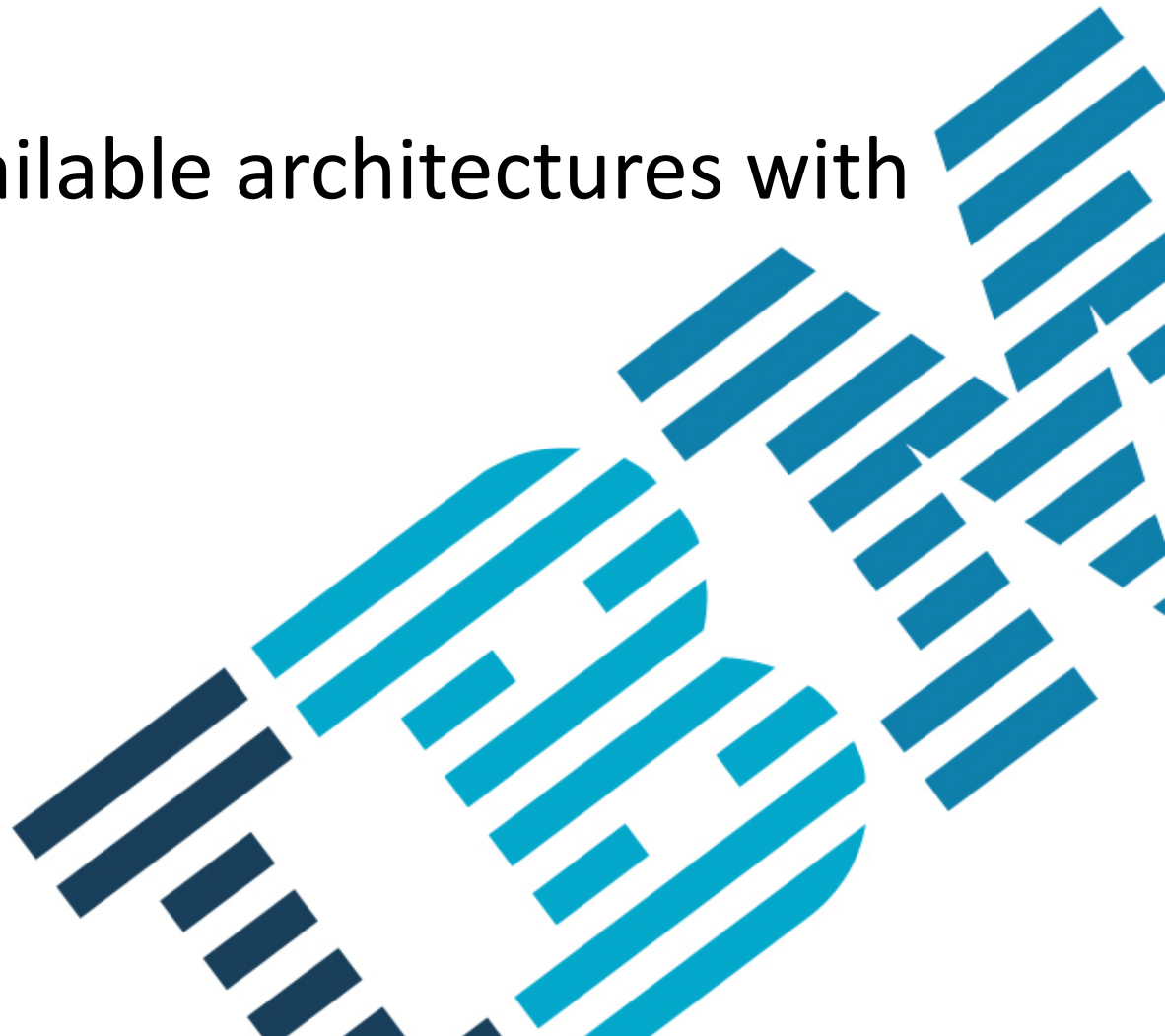# Building highly available architectures with WAS and MQ

Please Note

IBM's statements regarding its plans, directions, and intent are subject to change or withdrawal without notice at IBM's sole discretion.

Information regarding potential future products is intended to outline our general product direction and it should not be relied on in making a purchasing decision.

The information mentioned regarding potential future products is not a commitment, promise, or legal obligation to deliver any material, code or functionality. Information about potential future products may not be incorporated into any contract. The development, release, and timing of any future features or functionality described for our products remains at our sole discretion.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput or performance that any user will experience will vary depending upon many factors, including considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve results similar to those stated here.

Abstract and Aims

Abstract:

'This talk will look at architectures in which IBM MQ can be configured with the IBM WebSphere Application Server (and Liberty profiles) to give a highly-available scenario.

The basis be some of the scenarios that are documented in the developerWorks series "A flexible and scalable WebSphere MQ topology pattern". '

Aims:

- Outline some of the technologies and features that can be used for High Availability

- Consider some of the implications of technology choices

- Provide references for further study

- Find out what scenarios and concerns are of most interest

   i.e. what should be developing next!

..small warning...

- Designing, Implementing and Managing availability solutions is complex

- This presentation outlines some of the ideas, technologies and some points to keep in mind

- Coming from the development organization...

- So is just the 'tip of the HA iceberg'
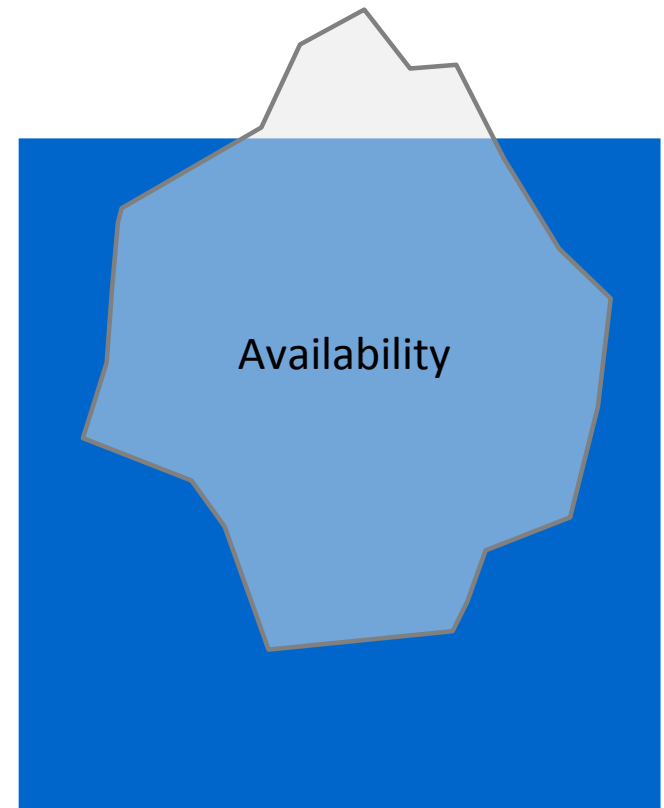
Availability

Agenda

- Introduction to HA concepts

- Product Technologies of Interest

- MQ – Multi-Instance QueueManagers

- Auto-reconnect from JavaEE

- Transactional Considerations

1 Slide introduction to High Availability

- **High Availability** – Ability of a system or component to be operational when required

- **Continuous Availability** - … accessible at all times for both planned and unplanned events

- **Redundancy** – eliminating single points of failures

- **Fail-over Strategies**
  Cold Standby - Warm Standby - Hot Standby

- **Software Clustering**
  Vertical - Horizontal

- **Hardware Redundancy**
  …not covered here

- **Costs of High Availability**
  …not covered here

Messaging design affecting availability

- Fire and Forget vs Request-Response
  Think about how the response is going to get back – is the route important?

- Synchronous vs Asynchronous
  Is a response expected immediately? How is that response getting back

- Affinity
  Message – relationship between messages
  Server – relationship between connections
  Session
  Transaction

- Message Ordering
  Can get difficult with a HA solution; even transaction recovery can happen concurrently with delivery

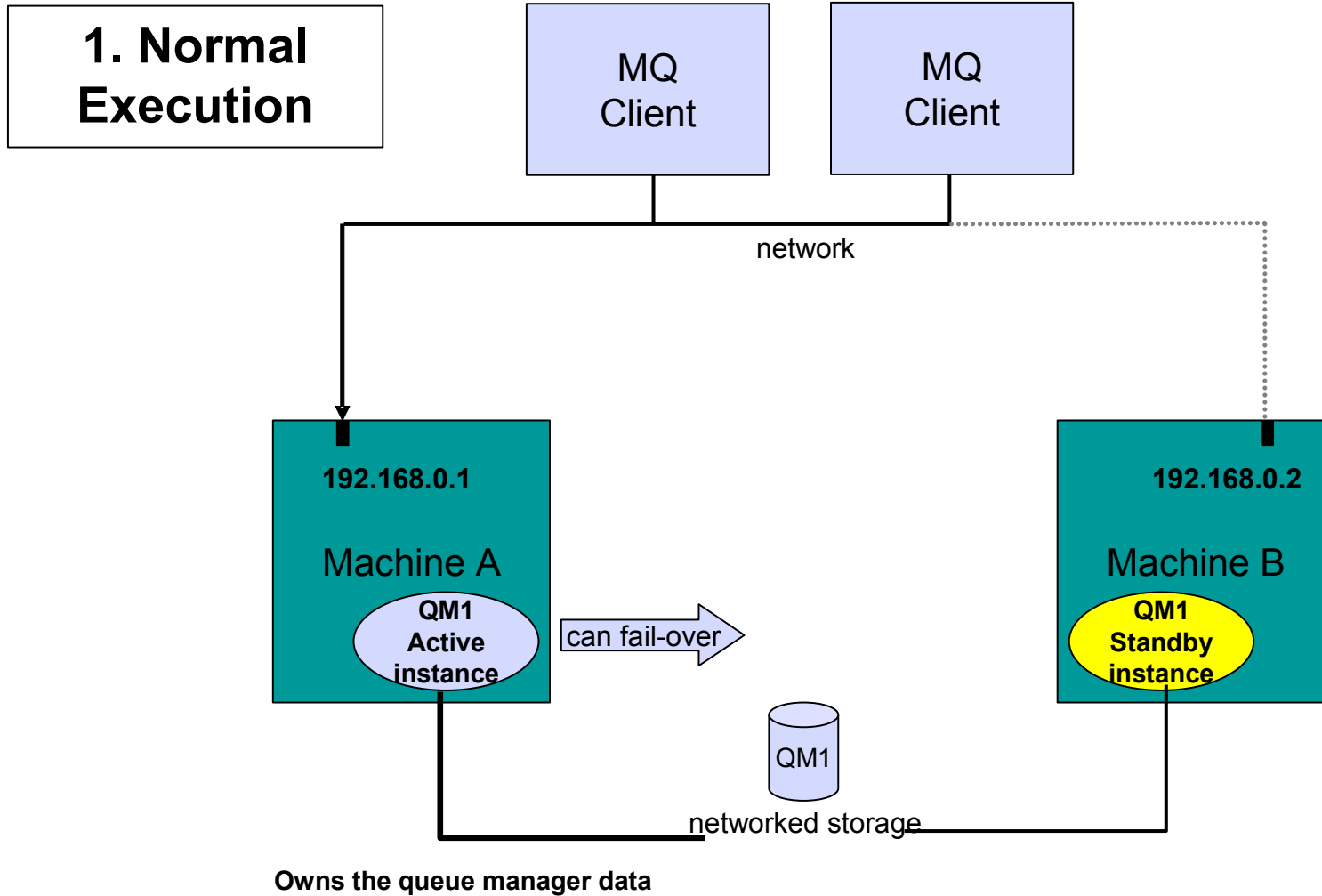- Transactional Concerns
  Where is transaction state held?

Product Technologies

- Hardware Clustering

- Network Spraying

- **IBM MQ**
  Multi-instance Queue Managers
  Queue Manager Clusters
    Workload balancing
  Queue Sharing Groups
  Client – connectivity
    Reconnect, Connection Name Lists, CCDTs

- **WAS**
  Clustering
  Network deployment, nodes
  Application availability

IBM MQ – Multi-instance Queue Managers

- Basic failover support without HA coordinator
  Faster takeover: fewer moving parts
  Cheaper: no specialised software or administration skills needed
  Windows, Unix, Linux platforms

- Queue manager data is held in networked storage
  NAS, NFS, GPFS etc so more than one machine sees the queue manager data

- Multiple (2) instances of a queue manager on different machines
  One is "active" instance; other is "standby" instance
  Active instance "owns" the queue manager's files and will accept app connections
  Standby instance does not "own" the queue manager's files and apps cannot connect
  If active instance fails, standby performs queue manager restart and becomes active

- Instances share data, so it's the SAME queue manager
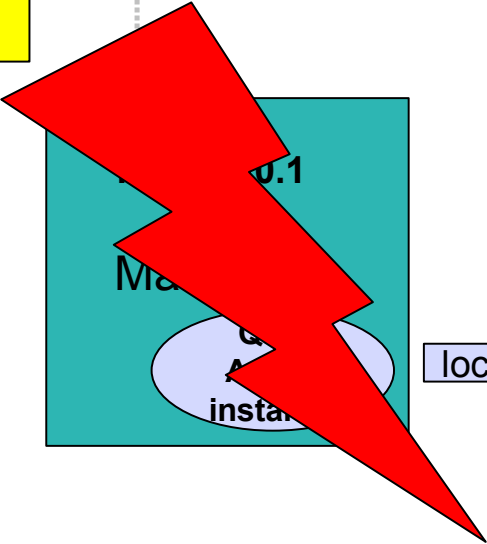 Including transactional logs

Multi-Instance QMs

1. Normal Execution

MQ Client

MQ Client

network

192.168.0.1

Machine A

QM1 Active instance

can fail-over

192.168.0.2

Machine B

QM1 Standby instance

QM1

networked storage

Owns the queue manager data

2. Disaster Strikes

MQ Client

MQ Client

network

Connections broken from clients

192.168.0.1

192.168.0.2

Machine A

Machine B

QM1 Active instance

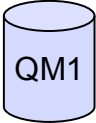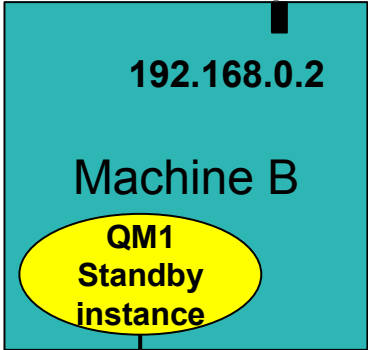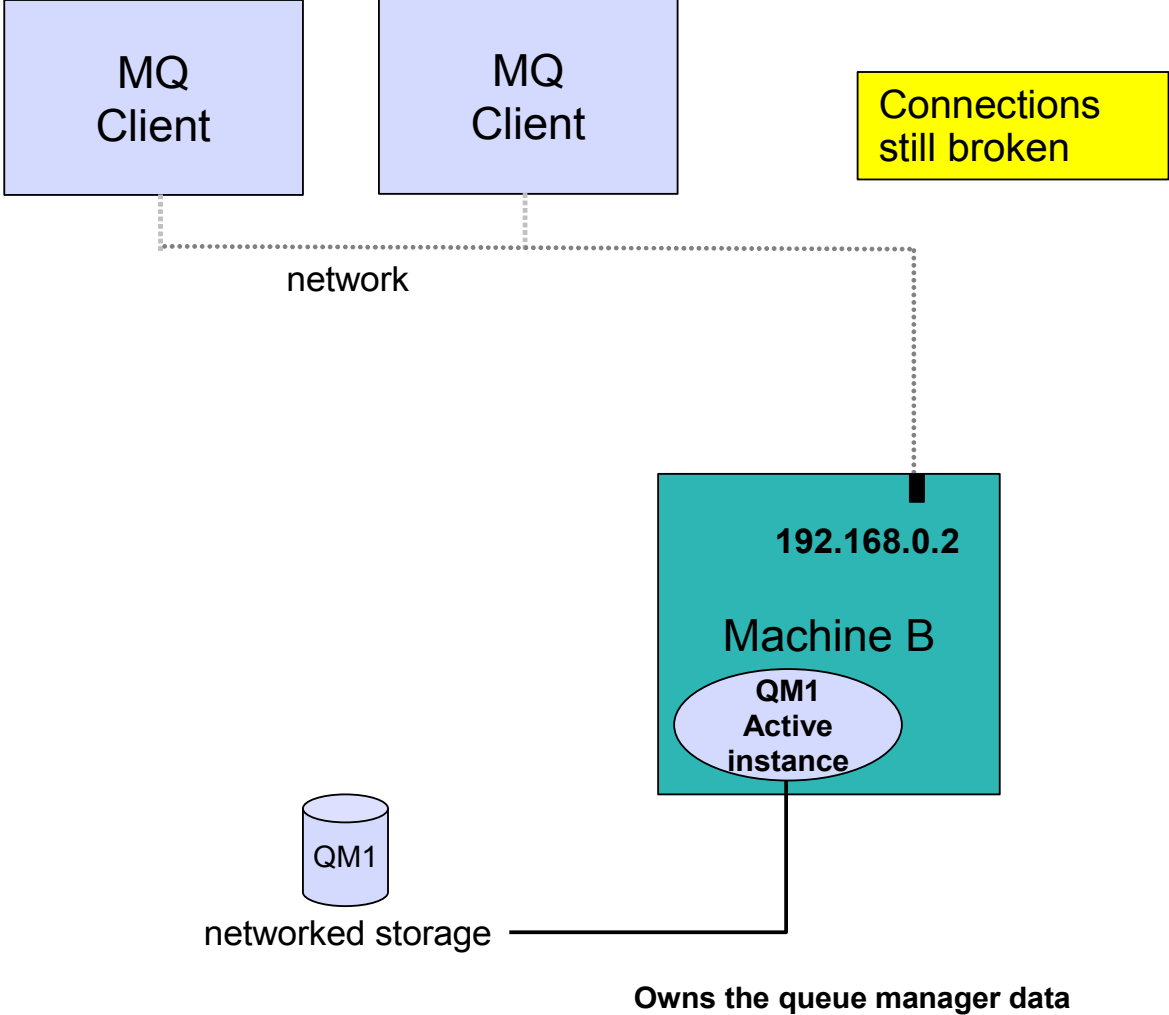QM1 Standby instance

locks freed

QM1

networked storage

2. Disaster Strikes

MQ Client

MQ Client

network

Connections broken from clients

192.168.0.1

192.168.0.2

Machine A

Machine B

QM1 Active instance

QM1 Standby instance

locks freed

QM1

networked storage

3. Standby Comes to Life

MQ Client

MQ Client

network

192.168.0.2

Machine B

QM1 Active instance

QM1

networked storage

Owns the queue manager data

4. Recovery Complete

MQ Client

MQ Client

Clients reconnected. Processing continues.

network

192.168.0.2

Machine B

QM1 Active instance

QM1

networked storage

Owns the queue manager data

Using Multi-instance Queue Managers from JavaEE
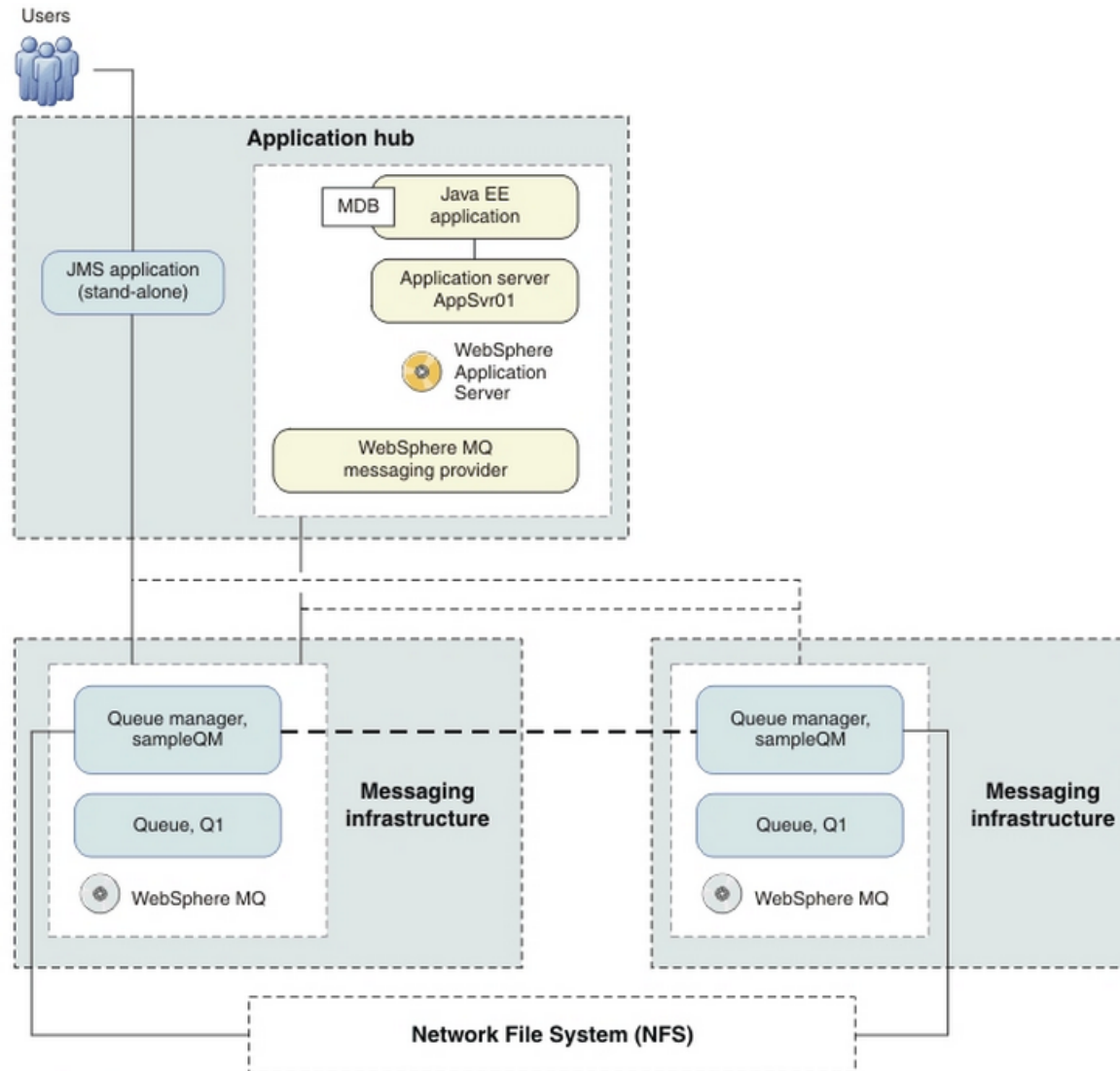
Autoreconnect in JavaEE

- Separate out the MQ concepts of
  'client auto-reconnect'
  'connection name list'
  And the role of the CCDT

- Client auto-reconnect is the ability of the MQ remote FAP transport to re-establish a client connection in the case of failure
  Controlled by ClientReconnectOptions

- Connection Name List provides the list of alternate host:port values of a queue manager

- CCDT provides a list of client definitions with weightings – and can also specify
  Connection name list, and ClientReconnectOptions

- **<u>Auto-Reconnect: NOT SUPPORTED in any MANAGED JavaEE Container from ANY vendor</u>**

Container Summary

	Connection Name List	**CCDT**	**Reconnect Options**	**Alternatives**
Activation Specifications	Supported (with restrictions)	Supported (with restrictions)	NOT supported	Act Specs own reconnect logic
Listener Ports	Supported (with restrictions)	Supported (with restrictions)	NOT supported	WAS's own reconnect logic
Web and EJB applications	Supported (with restrictions)	Supported (with restrictions)	NOT supported	Application own re-connection logic
Client Container	Supported	Supported	Supported	N/a

Activation Specifications – Connection Name List

- On Start-up - 1st entry in `ConnectionNameList` tried, then 2nd, 3rd, etc..

- Start-up retry properties defined on the Resource Adapter

- **startupReconnectionRetryCount** specifies the number of times that the WMQ RA will attempt to connect the endpoint, and the

- **startupReconnectionRetryInterval** property defines the time between reconnection attempts.

- During message processing the Java EE environment will detect the failure and then try to reconnect the Activation Specification.

- 1st entry in `ConnectionNameList` tried, then 2nd, 3rd, etc..

- After trying all of the entries it will wait for the period of time **reconnectionRetryInterval** before trying again.

- **reconnectionRetryCount** defines the number of consecutive reconnection attempts before an Activation Specification is stopped and will require a manual restart.

Activation Specifications - CCDT

- Very similar – the CCDT is, in effect, a list of entries that can be selected from.

- Entries can contain connection name list

- Best not to mix for "sanity's sake"

Web and EJB Containers

- Application, alongside normal error handling, needs to determine if wants to retry

- Application can
    - (1) Fail completely – and get re-driven later
    - (2) Re-drive the `createConnection()` call

- Re-drive the create Connection call …
    - .. this critically will re-drive the scan along the connection name list
    - .. or CCDT if one has been specified

- Depending on app server connection pools might be in use

- These may or may not purge themselves if a connection broken exception occurs

- Plus a connection pool might end up with different connections in the same pool

Transactional Considerations

- In a recovery situation the Transaction Co-ordinator needs to be able to get info on in doubt transactions from Resource Managers

  Implies...

- WAS needs to be able to connect to the QueueManager that has the logs

- Connection Factories *may* not deterministic as to the connection made
    c.f. Load balanced CCDTs, Connection Name List or IP Sprayers.

- Connecting to a different QM will give incorrect transaction state to WAS

- Transactions really in-doubt may be committed

- <u>Anything that alters where connections go may affect XA recovery</u>

RFE 53793: http://www.ibm.com/developerworks/rfe/execute?use_case=viewRfe&CR_ID=53793

Group-Level units of recovery - zOS

- A client's two-phase/global transaction can now be owned by a QSG

- *Instead of by individual queue managers*

  Implies...

- These in-doubt transactions can be resolved on any QMGR in the QSG.

- <u>Therefore having a z/OS queuemanager provides extra support for HA/Transactions</u>

Load Balancing

- Horizontal Scalability – implies that some way of balancing load across the components is required

- Can bring in WAS and MQ Clustering technologies
    Achieves balancing for those servers

- Can also use IP or hardware based balancing as well

- WAS → MQ balancing can be achieved using CCDT based weighting
- Co-located servers are also used to dedicate resource
- Or handle via administrative actions
- What is considered the *Single Point of Failure*?

WLM Comparison

	CONNAME list	CCDT (multi-QMGR)	Load balancer	Code stub
Scale of code change required for existing apps that connect to a single QM	+ive `MQCONN("QMNAME")` to `MQCONN("*QMNAME")` QMName might be in JNDI config for Java EE apps. Otherwise requires a one character code-change.			-ive Replace existing JMS/MQI connection logic with code stub.
Support for different WLM strategies	-ive Prioritized only	= Prioritized + Random	+ive Any, including per-connect round-robin	+ive Any, including per-message round-robin.
Performance overhead while primary QM is down	-ive Always tries first in list	+ive Remembers last good	+ive Port monitoring avoids bad QMs	+ive Can remember last good, and retry intelligently
XA Transaction Support	-ive The transaction manager needs to store recovery information that reconnects to the same QM resource. An MQCONN that resolves to different QMs generally invalidates this. e.g. in Java EE, a single Connection Factory should resolve to a single QM when using XA.			+ive Code stub can meet the XA transaction manager's requirements. e.g. multiple Connection Factories.
Connection rebalancing on failback. **e.g. when a QM restarts after a failure or planned outage, how long till apps use it again**	-ive Connection pooling in Java EE will hold onto connections indefinitely, unless connections are configured with an aged timeout. Using an aged timeout might drive exceptions in some cases. An aged timeout also introduces a performance overhead during normal operation. Conversation sharing might need to be disabled (SHARCNV=1) with an aged timeout to ensure reconnects always establish a new socket. The 'remembers last good' CCDT behaviour might also delay failback.			+ive Code stub can handle failback flexibly, with little/no performance overhead.
Admin flexibility to hide infrastructure changes from apps	-ive DNS only	= DNS and/or shared file-system / CCDT file push	+ive Dynamic Virtual IP address (VIP)	= DNS or single-QMGR CCDT entries

Active – Active Scenarios

- Often get asked for *active-active* configuration
  <u>What exactly does this mean?</u>

- Typically this 2 application servers connected to 2 QM s
  Often done for load balancing to give horizontal scaling

- Question:
  What fail over characteristics are required?
  What is the affinity of the applications and instructure
  QM workload can be re-distributed by CCDT
   Connection pool re-balancing on recovery
  Application

Practical Scenario:
"A flexible and scalable MQ topology Pattern"

- DeveloperWorks series by Peter Broadhurst

- *Pattern discussed in detail here:* *http://ow.ly/vrUUV*

✓ Continuous availability to send MQ messages, with no single point of failure

✓ Linear horizontal scale of throughput, for both MQ and the attaching applications

✓ Exactly once delivery, with high availability of individual persistent messages

✓ Three messaging styles: Request/response, fire-and-forget, and publish/subscribe

✓ A hub model, with a centralized MQ infrastructure scaled independently from the application

✓ Standalone, JavaEE and SCA environments

Overview – architecture view
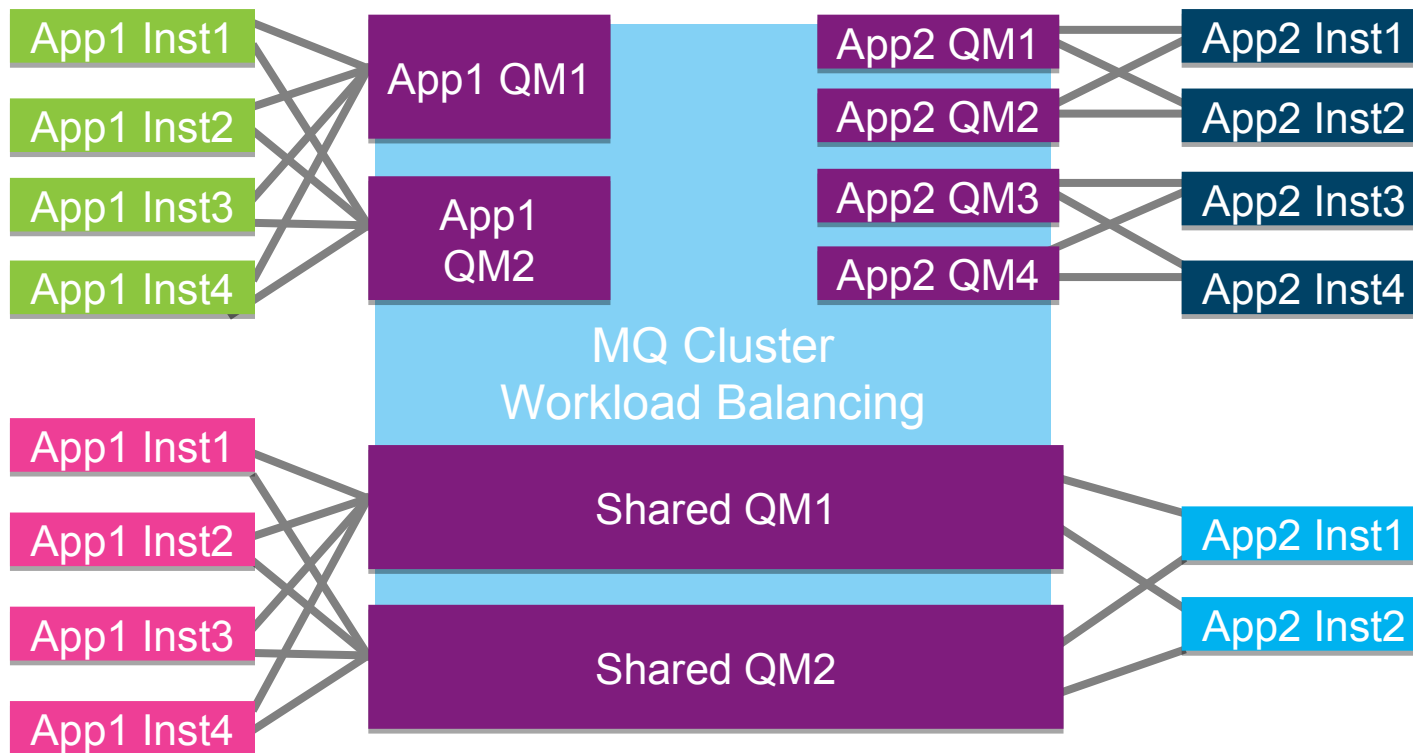
- Every sender/requester uses **two connections**
- Every receiver/service has **two listeners**
- Make each Queue Manager HA to recover persistent messages
- Simple to interoperate with co-located Queue Managers
- Simple to interoperate with z/OS Queue Sharing Groups

Overview – infrastructure view
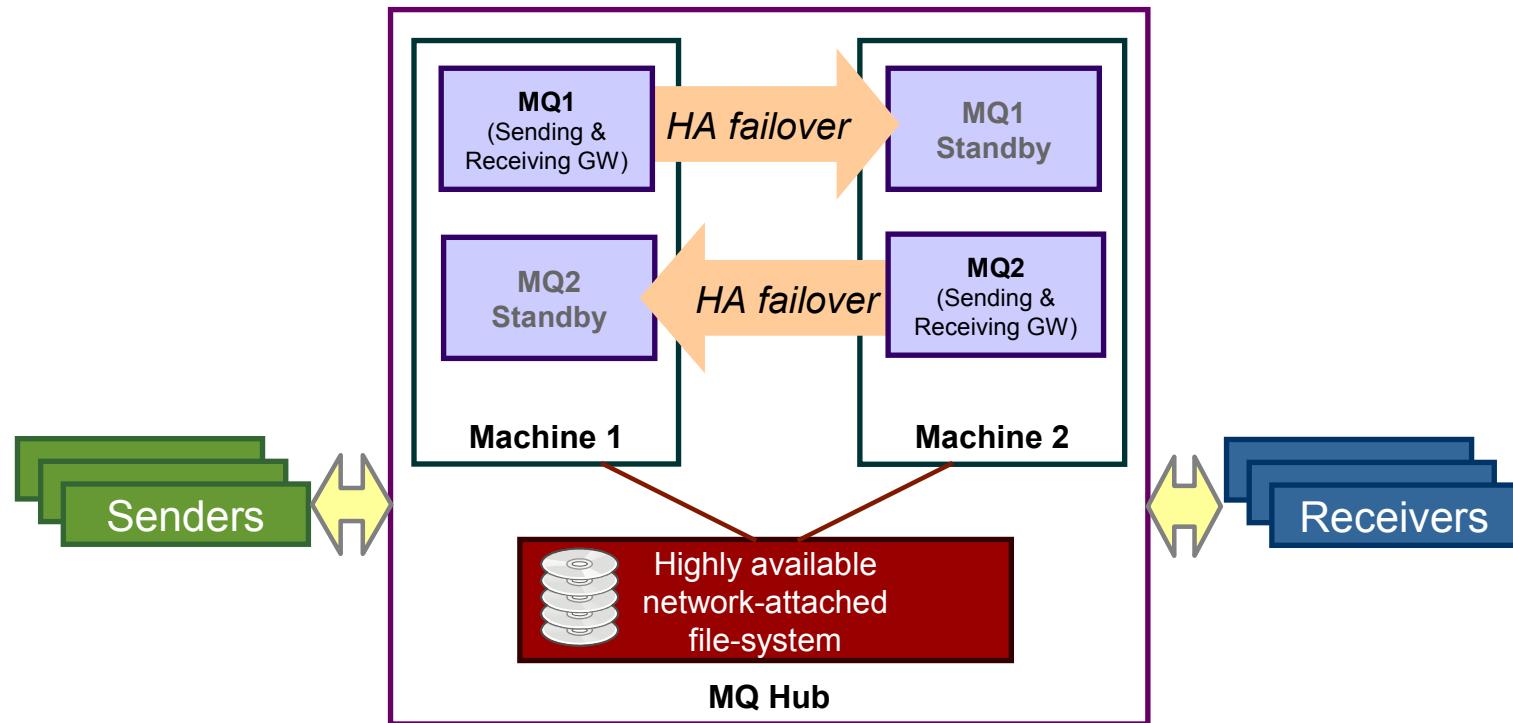
- Principal design philosophy is active/active
  - Continuous availability of the service
- Minimum number of queue managers is 2
  - Sending and receiving gateway roles can be fulfilled by the same qmgr
- HA failover is optional
  - If you have persistent messages that you need to recover quickly after a failure

Overview – 2 is the magic number
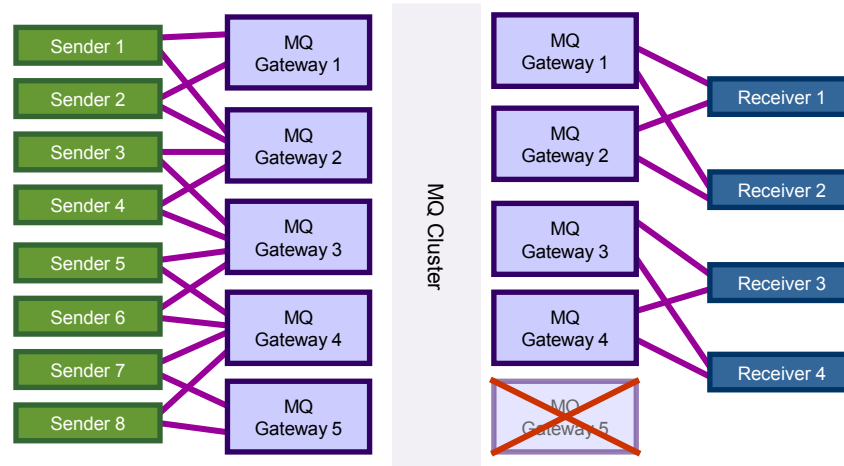
- **Every sender sends to two queue managers**
  - No single point of failure for sending messages
  - Not too many places to look for messages
- **Every receiver listens to two queue managers** *concurrently*
  - Every queue manager has two app instances listening for messages
  - Every app instance listens to two queue managers
  - *Note: cannot have more receiving gateways than receiving app instances*
- **No single point of failure**
  - Any single component can fail, and all other components continue processing
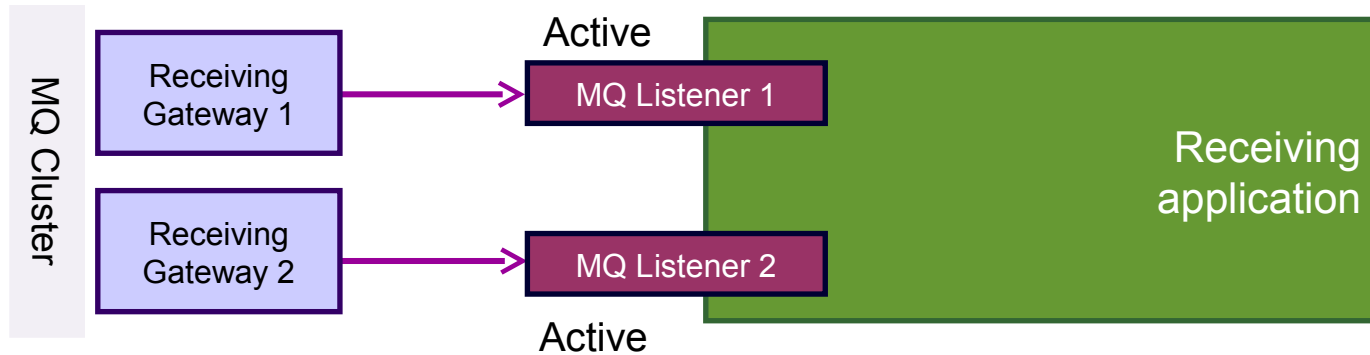
Sending messages
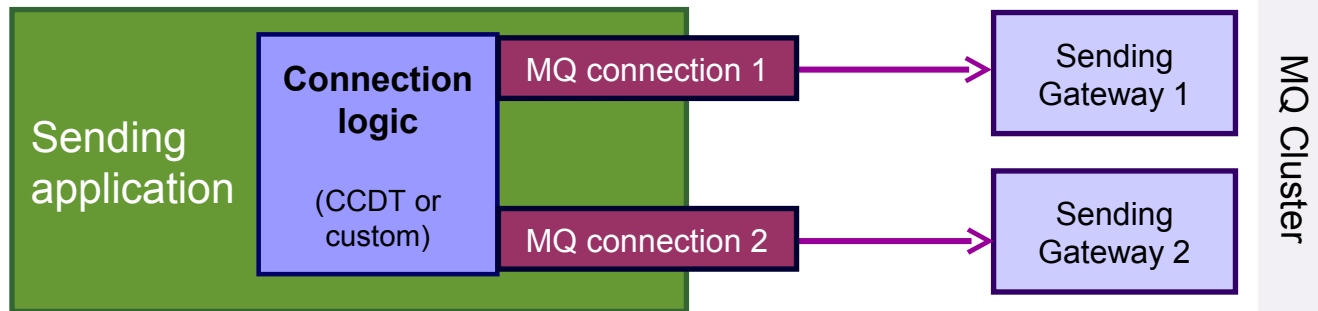
- Each app instance sends to two different queue managers

- Need a workload management strategy
  - Prioritised
  - Random
  - Round robin – *my personal preference*

- Biggest practical concern for customers:

*How do I create/change my app code to connect
to two different remote queue managers*

Receiving messages

- The application needs **two active** listeners
  - Random/prioritised attachment can lead to stranded messages
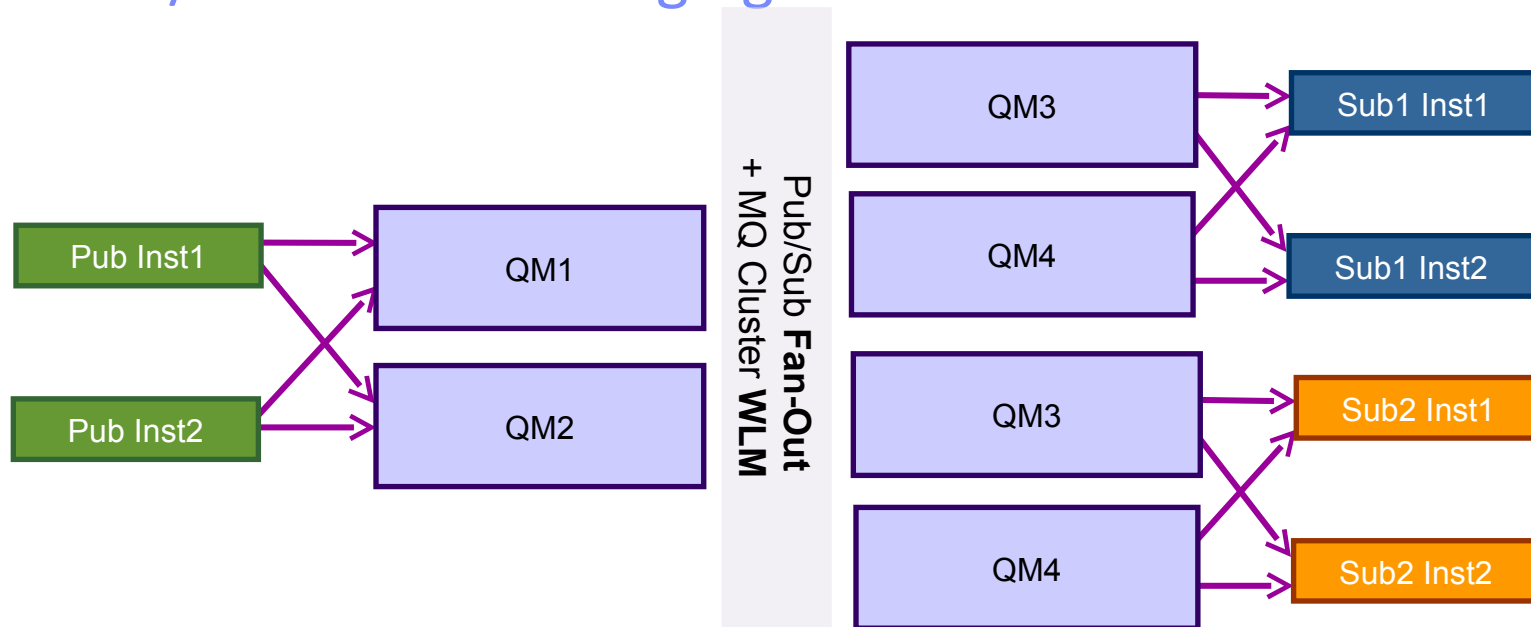 - AMQSCLM is an alternative

- For Java EE this means two MDB endpoints
  - EJB 2.1 style deployment descriptors
    - Add a second endpoint to the XML
  - EJB 3.0 style annotations
    - Create a code hierarchy
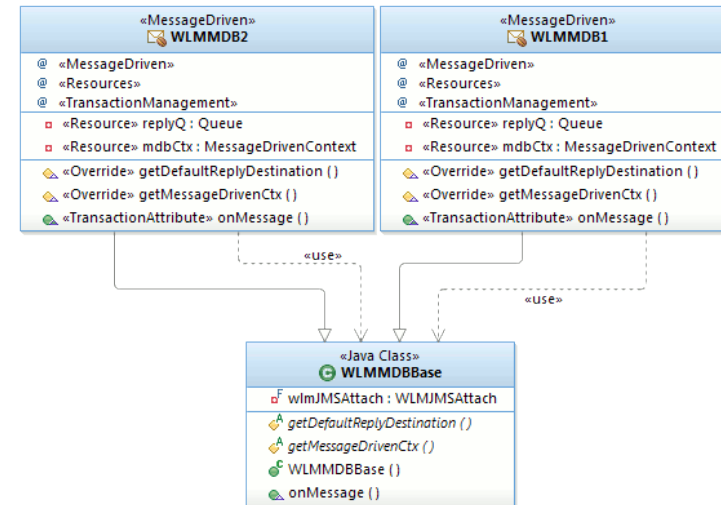
EJB 2.1 & 3.0 samples on github:
https://github.com/ibm-messaging/mq-wlm-client
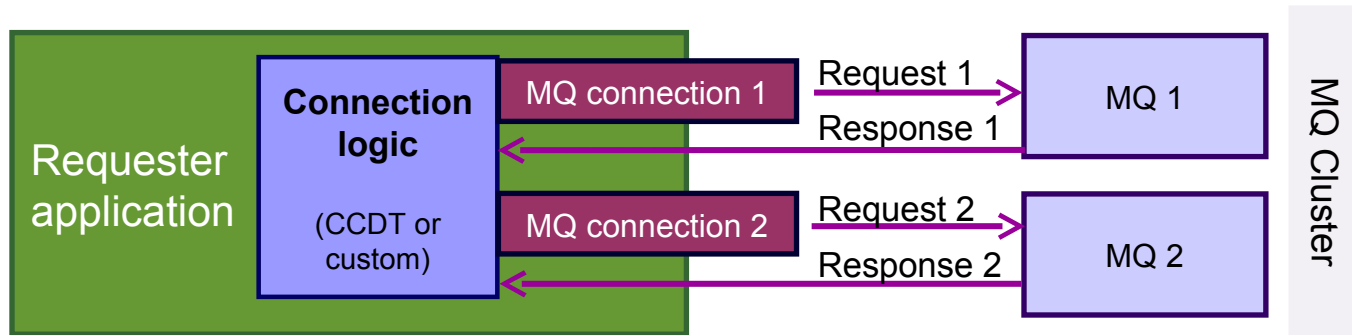
Publish/subscribe messaging

- MQ gives the same QoS for pub/sub as for P2P
  - Fan out messages one-to-many
  - WLM across multiple subscriber instances
- Achieved by bridging *durable* subscriptions to cluster queues
  - Define subscriptions on queue managers where *publishers* connect
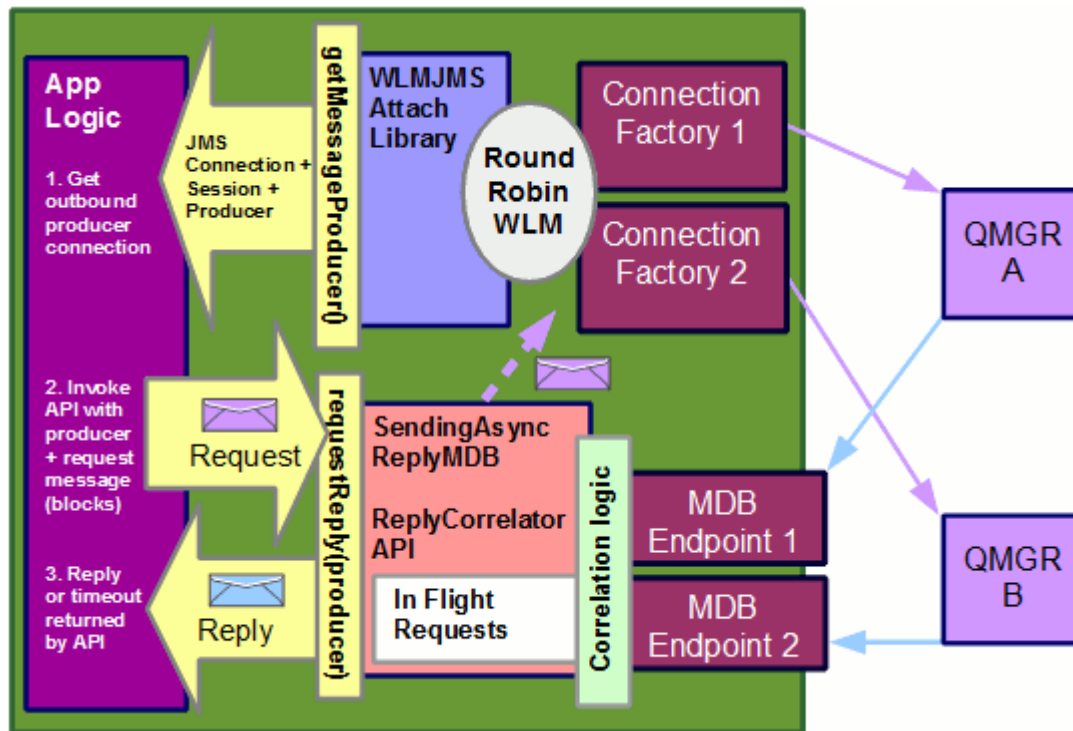
Synchronous request/response

Requester application — Connection logic (CCDT or custom) — MQ connection 1 — Request 1 / Response 1 — MQ 1 — MQ Cluster

MQ connection 2 — Request 2 / Response 2 — MQ 2

▶ Use **same MQ connection** to receive the response
  - e.g. the same JMS Session

▶ MQ fills in the MQMD.ReplyToQMgr on send
  - Back-end app **must** honour this when sending the response

Sample JavaEE applications
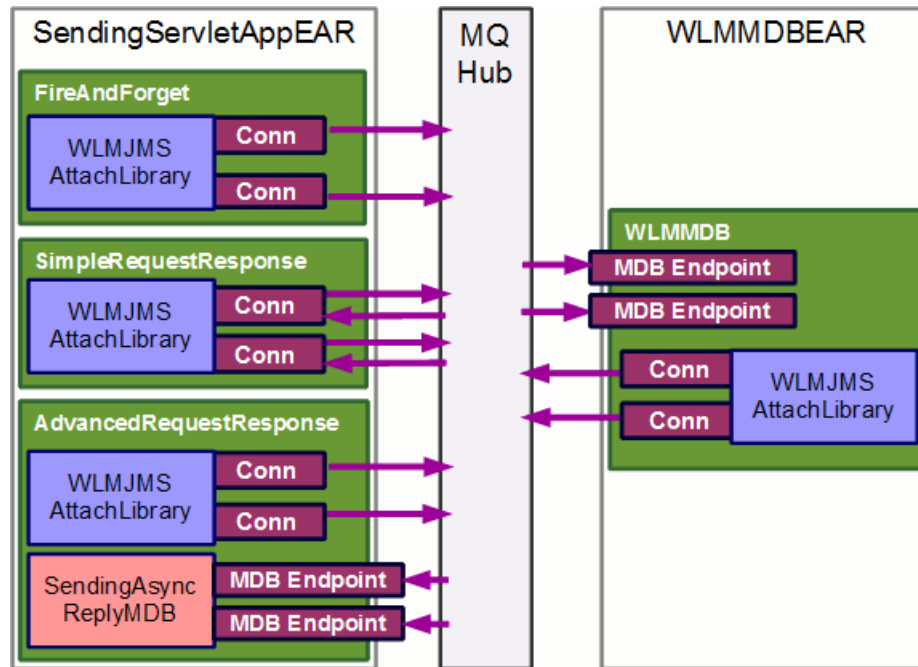
- **WLMJMSAttachLibrary**
- Code library used within all of the applications to establish workload-balanced outbound connections. In the example projects and deployment, this library is bundled individually within each EAR that depends on it.

Synchronous request/response – advanced case

- A clustered reply queue for *synchronous* request/response
  - Lets replies take a different route back from requests
  - Only adds resilience for highly parallel nonpersistent request/reply, where requests build up on back-end threads in business logic rather than in MQ
- Requires correlation logic to pass responses back to waiting threads
- *This is not the same as two-way asynchronous messaging*

Code samples: https://github.com/ibm-messaging/mq-wlm-client

Two-way asynchronous messaging
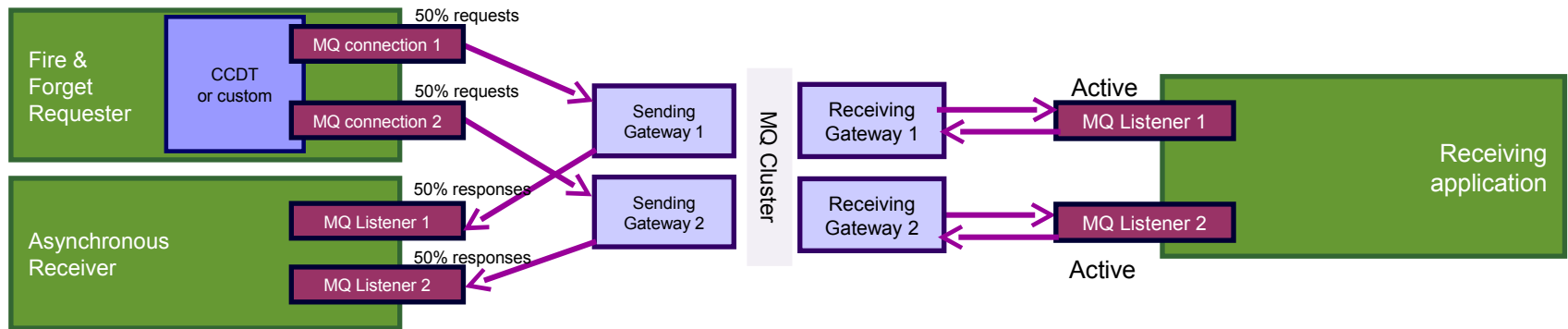
- The optimal use of messaging is fully asynchronous
- Requests are sent "fire & forget", as are responses
  - Critical requests are sent as persistent within a transaction that updates a DB
  - Transactional state update + persistent send = *exactly once delivery*
- Responses are handled by any app instance at any time
  - No thread is left 'hung' in the requesting application
  - If responses need to be correlated with requests, then a state store is used
    - A Database – DB2 etc.
    - An elastic cache – WebSphere eXtreme Scale
- Must be designed into the application
  - Can revolutionize **responsiveness**
  - Truly **decouples** applications

Limitations for messaging ordering

- No active/active solution is provided here for ordered messages
   MQ only assures order when there is one path from
   producing thread to consuming thread

- The simplest solution, *and as far as this presentation goes*
   Allocate individual queue managers with HA Failover for ordered messages

Can be the **same queue manager**.
Might be in different hubs.

References

- "High Availability in WebSphere Messaging Solutions"
http://www.redbooks.ibm.com/abstracts/sg247839.html

- "IBM WebSphere Application Server v8 Concepts, Planning and Design Guide"
http://www.redbooks.ibm.com/abstracts/sg247957.html

References

- "A flexible and scalable WebSphere MQ topology pattern"
  http://www.ibm.com/developerworks/websphere/library/techarticles/1303_broadhurst/1303_broadhurst.html

-

- "Workload Balancing "
  https://www.ibm.com/developerworks/community/blogs/messaging/entry/ccdts_connection_namelists_load_balancers_and_code_stubs?lang=en

- "Scenario: Using a multi-instance queue manager for high availability with WebSphere Application Server"
  http://www-01.ibm.com/support/knowledgecenter/prodconn_1.0.0/com.ibm.scenarios.wmqwasha.doc/topics/scenario_overview.htm