So You Think You Understand Multi-Instance Queue Managers?

Christopher Frank

IBM WebSphere Connectivity

chrisfra@us.ibm.com

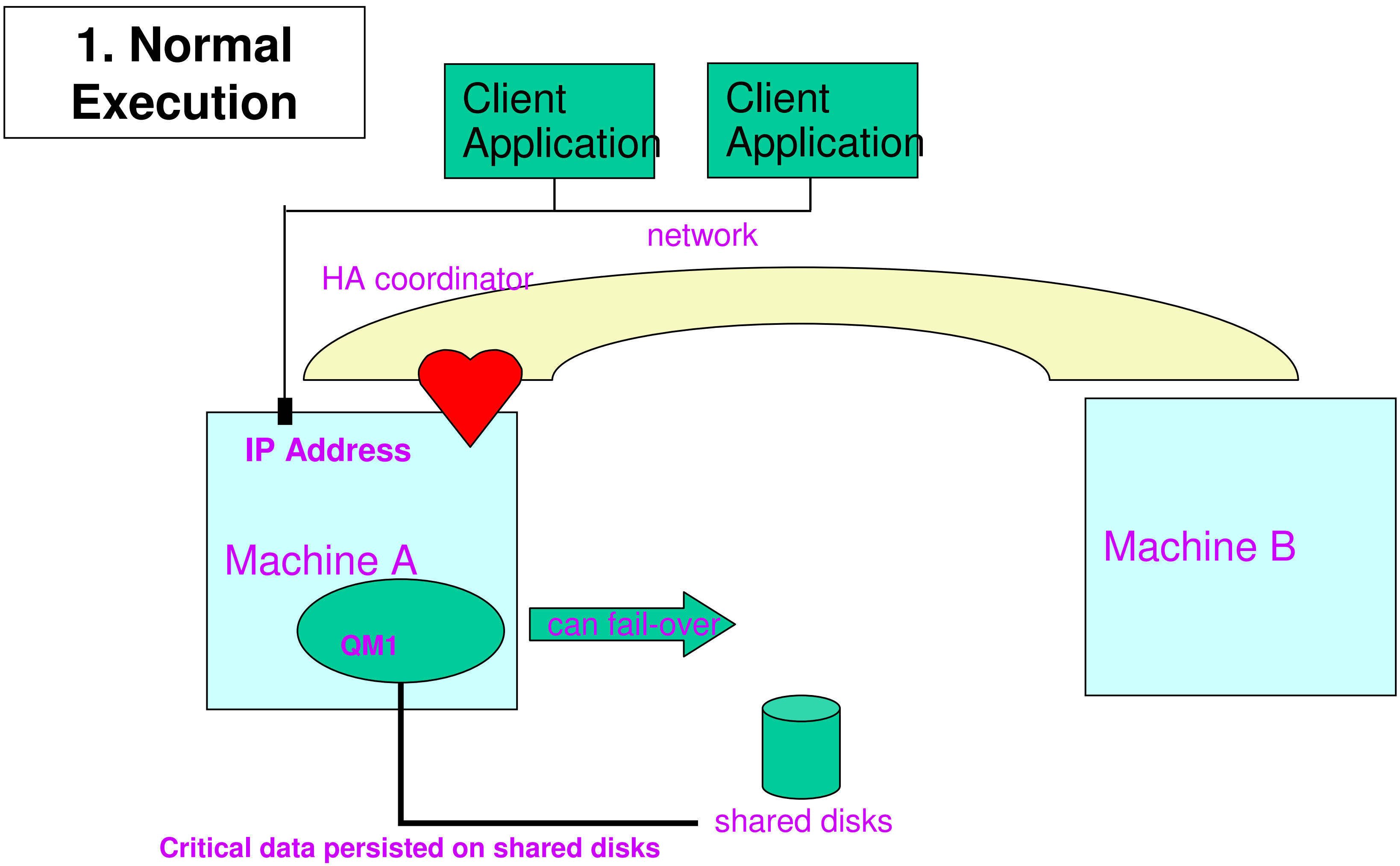
Introduction

- **Topics to be covered in this presentation:**
  - Brief overview of multi-instance queue manager support and its role as an HA solution
  - How to validate the file system for use with multi-instance queue managers
  - File system requirements and why they are what they are
  - Queue manager status
  - How file locking is used
  - Liveness checking
  - Troubleshooting
  - Summary

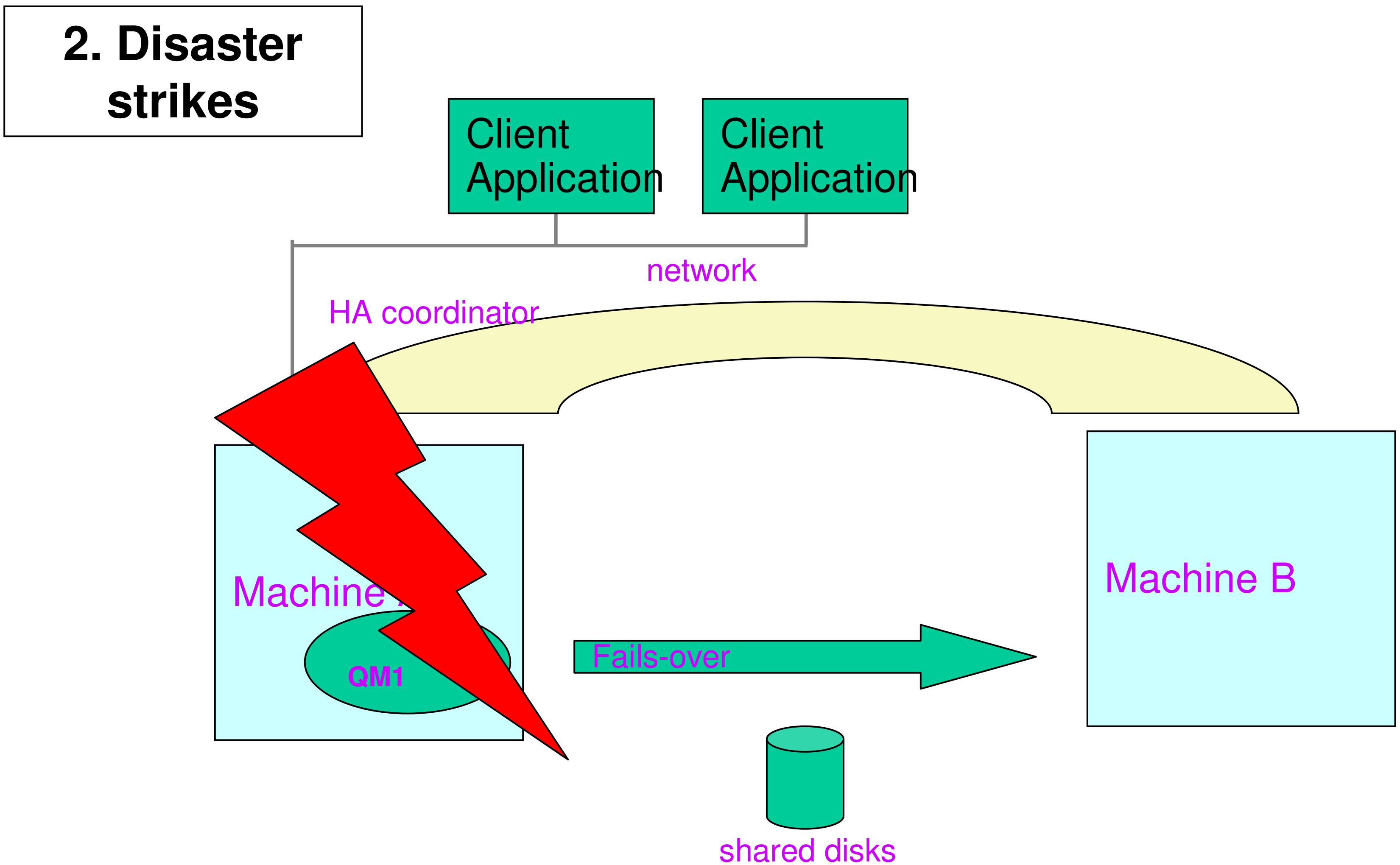
High Availability Options for MQ

- **High availability managers**
  - Products designed to provide comprehensive system high availability
  - Can cover multiple products – MQ, IIB, DB2, Oracle, WAS etc.
  - Requires an HA manager such as
    - HACMP for AIX
    - ServiceGuard for HP
    - Solaris Cluster
    - Veritas Cluster Server
    - MSCS for Windows Server
    - Linux-HA
- **Multi-instance support for MQ and IBM Integration Bus**
  - Provides *basic* failover for MQ and WMB/IIB only
  - Software only
  - Comes out of the box – no external HA coordinator needed

HA Cluster Coordination behavior (1)

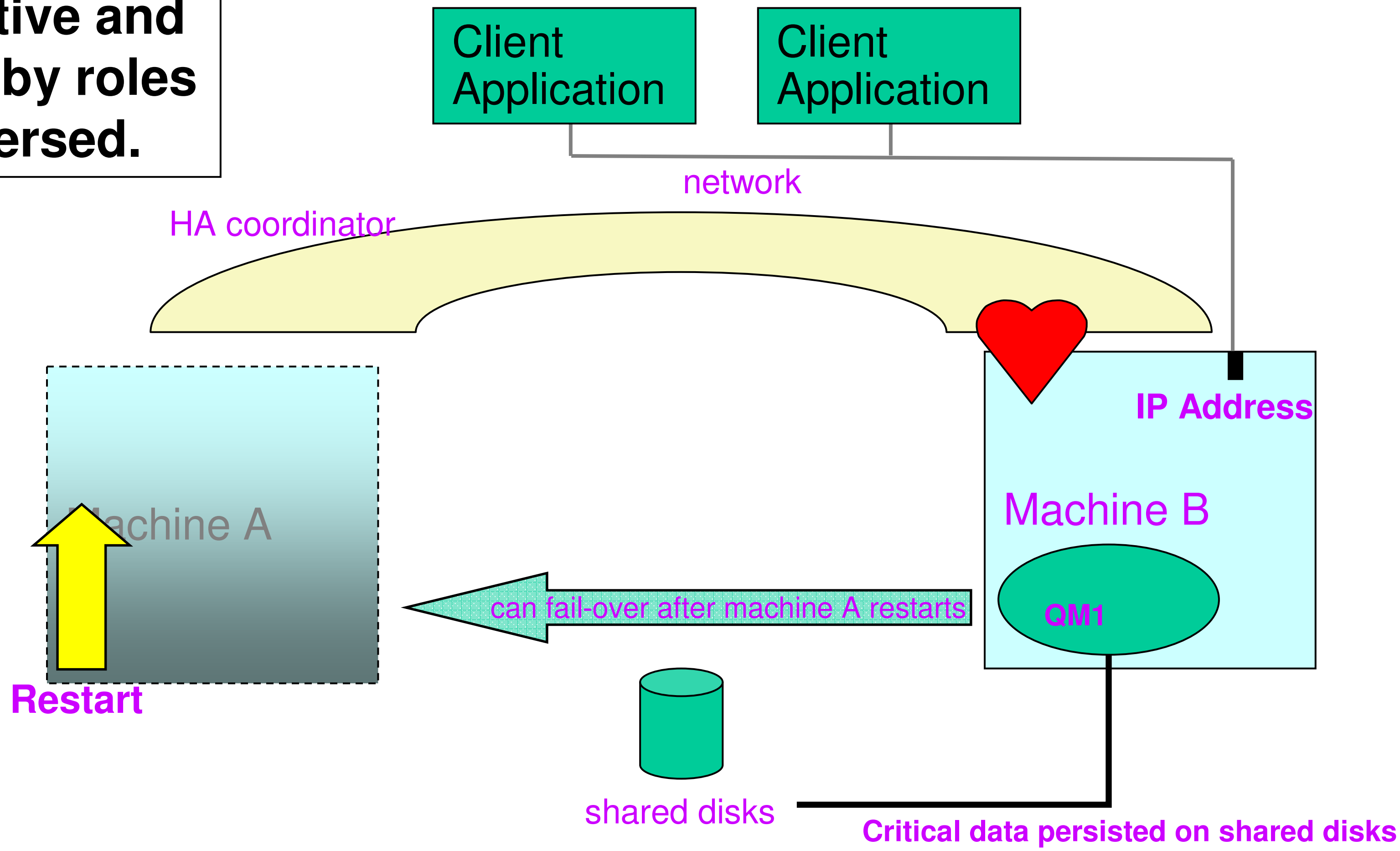


HA Cluster Coordination behavior (2)

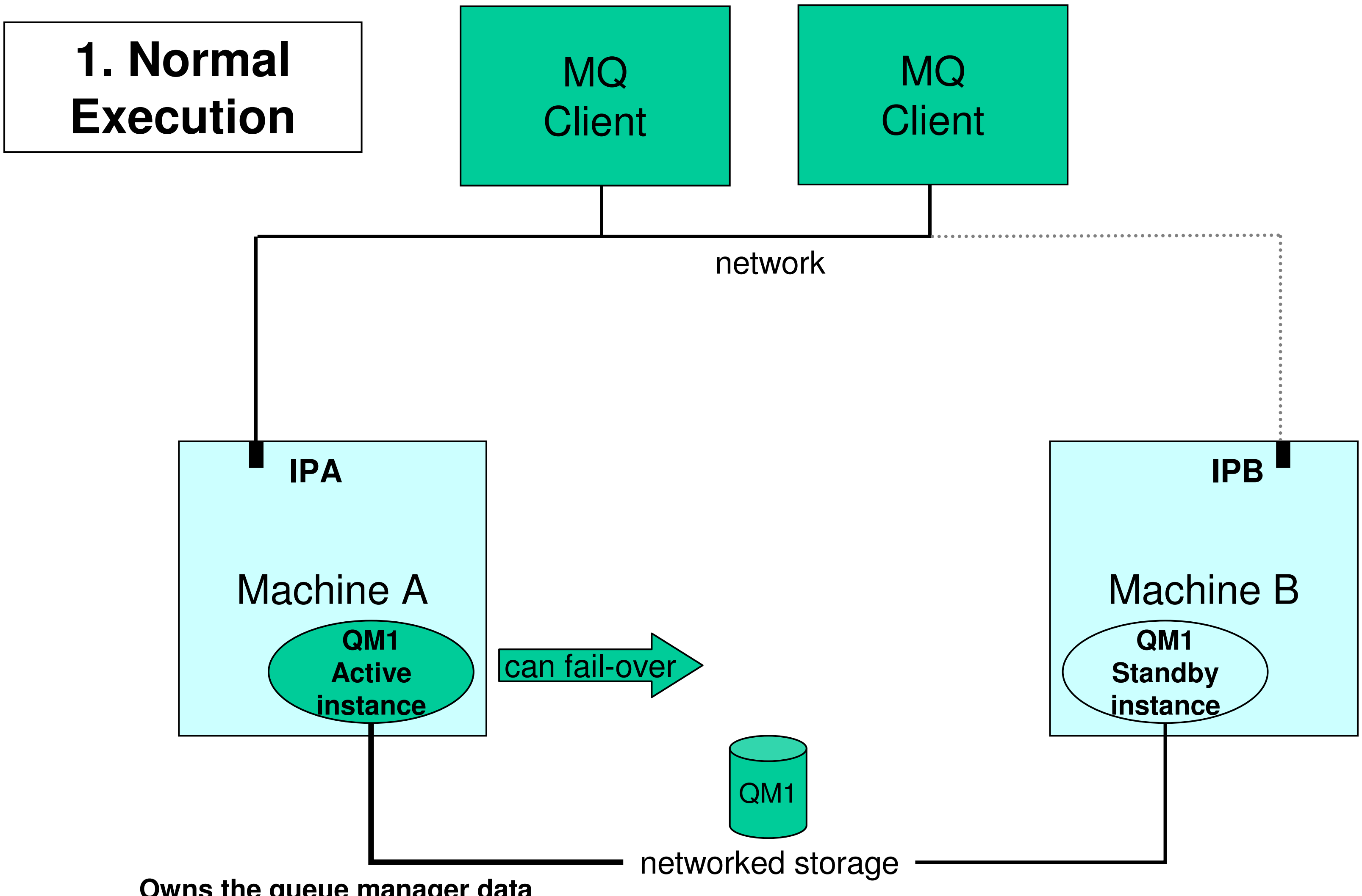


HA Cluster Coordination behavior (3)

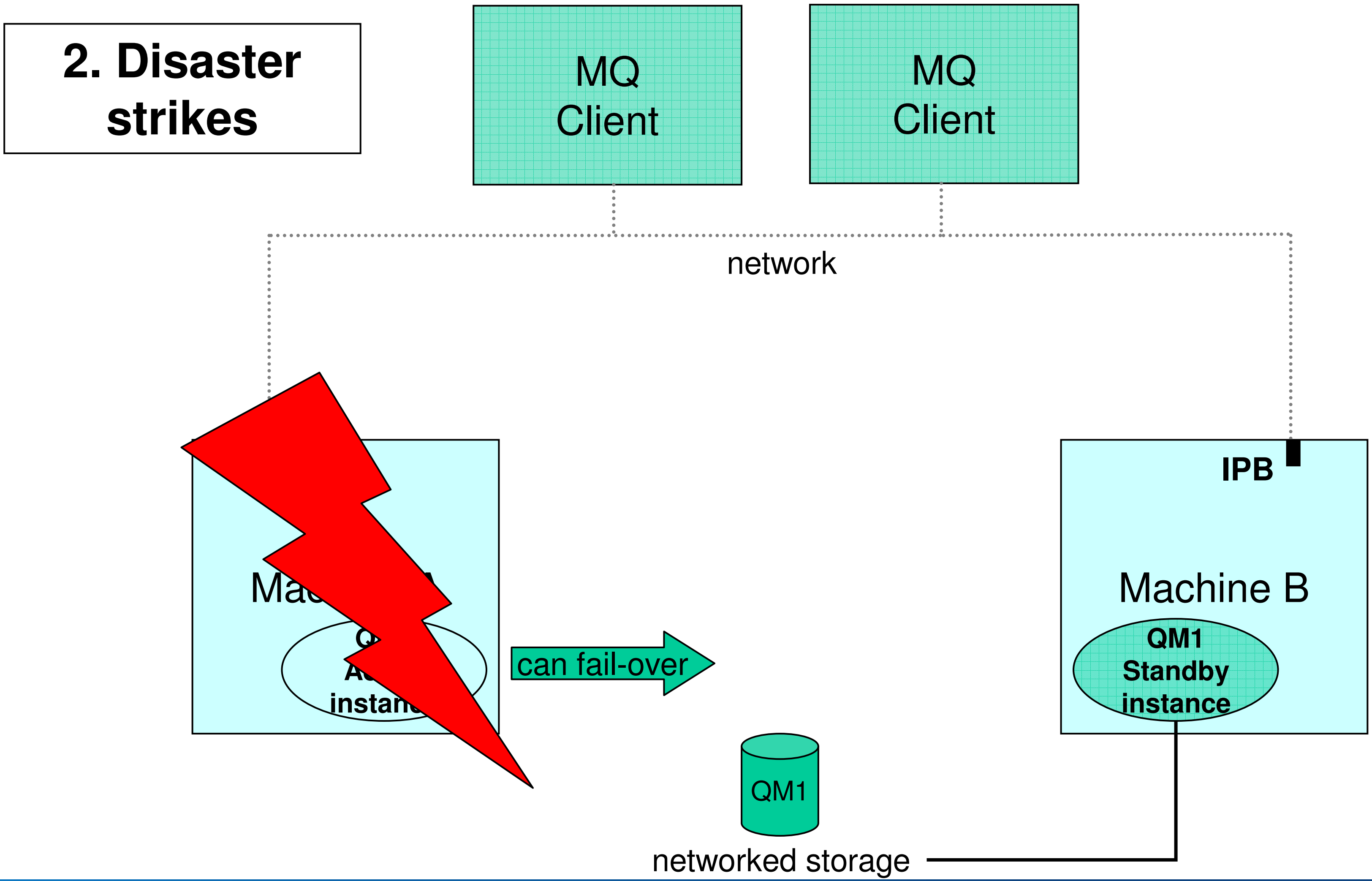
3. Active and Standby roles reversed.



Multi-instance queue manager behavior (1)

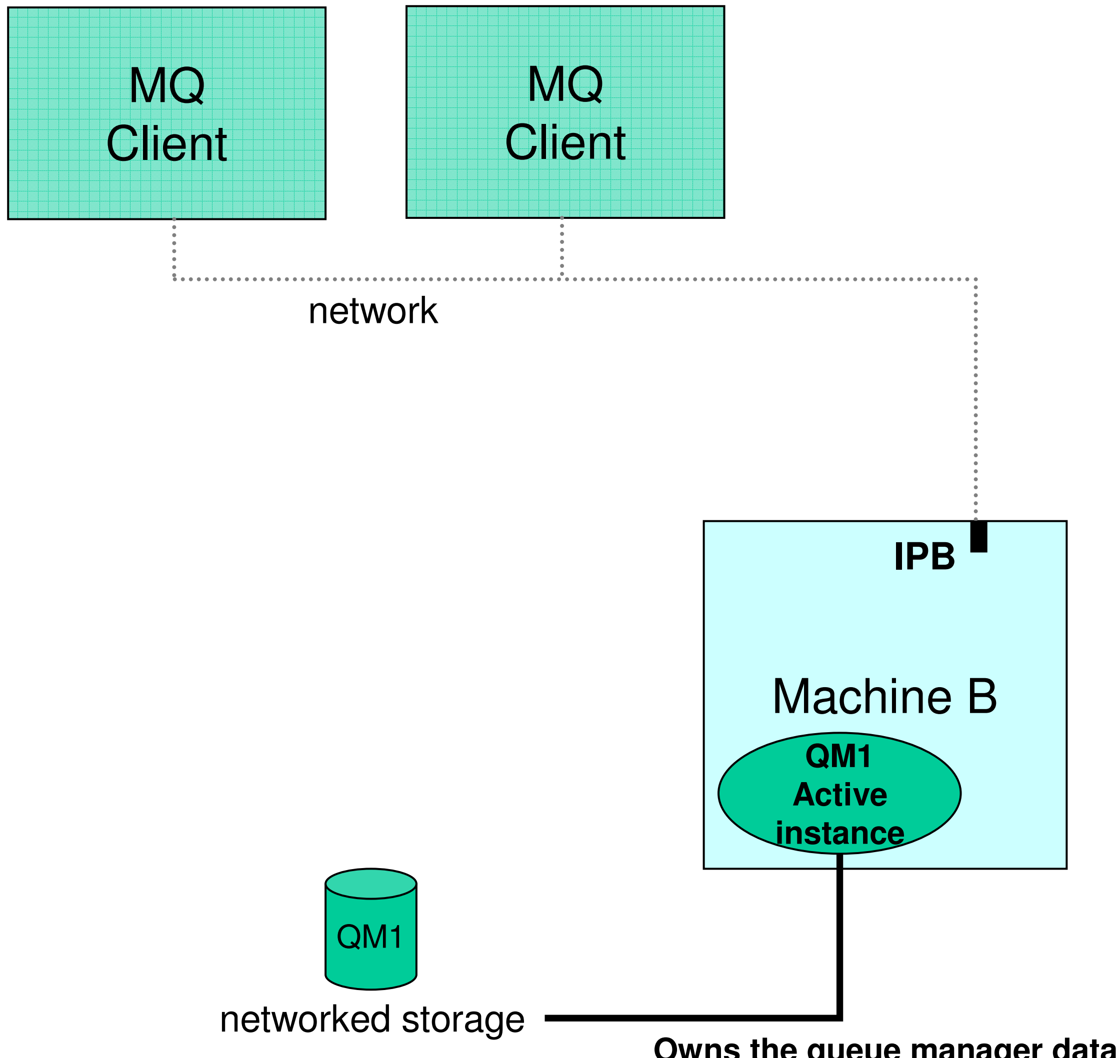


Multi-instance queue manager behavior (2)



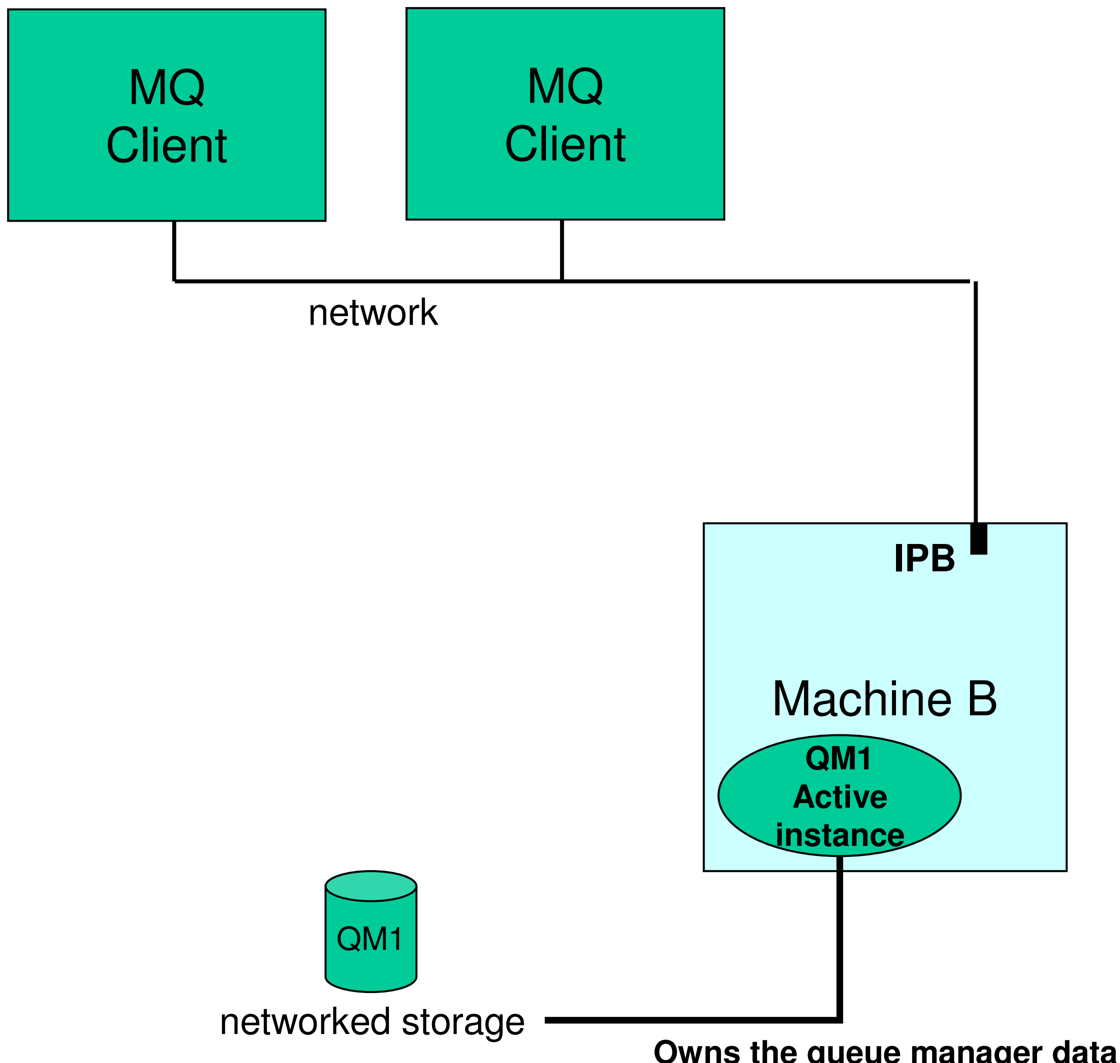
Multi-instance queue manager behavior (3)

3. Standby instance leaps into action



Multi-instance queue manager behavior (4)

4. Recovery complete – clients reconnect



What Multi-instance queue managers provide

- **<u>Basic</u> failover support without separate HA coordinator**
  - Failover support for queue manager only
  - No data or IP failover

- **Queue manager data is held in networked storage (NAS, not SAN)**
  - Multiple machines see the queue manager data
  - Multi-instance support requires *<u>lease-based file locking</u>*
    - NFS v4, GPFS, GFS2, CIFS (Windows only)

- **Allows starting multiple (two) instances of a queue manager on different machines**
  - One instance is “active” – the other instance is “standby”
  - Active instance “owns” the queue manager’s files
    - Will accept connections from applications
  - Standby instance does not “own” the queue manager’s files
    - Applications cannot connect to standby instance
    - If the active instance fails, performs queue manager restart and becomes active
  - Instances share the data, so it’s the SAME queue manager

What is a "Standby Instance"?

- **A standby queue manager instance is essentially a queue manager paused in the early stages of queue manager startup**

- **It does not "own" the queue manager's files and therefore is not capable of doing message processing**

- **"strmqm –x" is used to start an instance of a multi-instance queue manager**
  - The first instance will be the active instance
  - The second instance will be the standby instance
  - Additional instances are not permitted

- **A standby instance:**
  - Polls file locks held by the active instance every 2 seconds
    - Tuning Parameter available to alter this if needed
  - A standby instance also is responsive to requests to end ("endmqm –x")
  - A standby instance is *responsive* to requests by applications trying to connect, but it *rejects* them

Support for Network Filesystems

- **As of MQ V7.0.1 support for network filesystems was properly integrated**
  - Any “modern” network filesystem protocol with “proper” semantics supported
    - NFS v4 (not v3), CIFS (Windows only), GPFS, GFS2, etc
- **File systems such as NFS V4 provide *leased-based file locking***
  - Can detect failures and then release locks following a failure.
  - Older file systems such as NFS V3 do not have a reliable mechanism to release locks after a failure
    - Thus NFS V3 must not be used with multi-instance queue managers
    - NFS v4 and also GPFS, GFS2, CIFS (for Windows only) can be used
- **NFS v3 will generally work for MQ**
  - But it’s not adequate for multi-instance queue managers
  - So NFS v3 is NOT SUPPORTED (no, not ever) by multi-instance queue managers
- **Not all NFS v4 implementations are supported**
  - They must behave strictly according to Posix rules
  - They must meet certain configuration requirements
  - A tool is provided to validate configuration (amqmfsck)

Validating the filesystem for MIQM (1)

- **amqmfsck is a tool which checks out the filesystem**

- **The minimum steps to validate the file system are:**
  - `amqmfsck /shared/qmdata`
    - This checks basic POSIX file locking behavior
  - `amqmfsck -w /shared/qmdata`
    - Use on two machines <u>at the same time</u> to ensure that locks are handed off correctly when a process ends.
  - `amqmfsck -c /shared/qmdata`
    - Use on two machines <u>at the same time</u> to attempt concurrent writes.

- **The following can be used to shows whether the logger can guarantee data integrity.**
  - `amqmfsck [-f NumberOfPages] -i /shared/qmdata`
    - Use on two machines <u>at the same time</u>, then do something dreadful to the first one, then run a third instance to analyse the wreckage.

- **The top three steps are the *<u>minimum</u>* checks that should be performed**

- **Where we have put a restriction in the SOE, one of these tests fails.**

Validating the filesystem for MIQM (2)

- **If one or more tests *fail*, the file system is not capable of supporting multi-instance queue managers**
  - Run the tests using the *verbose* option ("-v") to help you interpret the errors
    - This will help you understand why the command failed, and whether the file system can be reconfigured to address the problem.
  - Failures caused by access control problems are not uncommon
    - These can usually be addressed by changing directory ownership or permissions.
  - Failures can also result from specific file system configuration options
    - These can often be addressed by reconfiguring the file system to behave in a different way.
      - File system performance options can fall into this category
      - Resolving usually requires working closely with team that understands the underlying file system

Validating the filesystem for MIQM (3)

- **If the tests are successful, the following is returned:**

  "*The tests on the directory completed successfully*"

- **Note that this is no guarantee!**
  - The file system can pass the checks but problems can still occur when doing so.
  - Also, environments not listed as supported in the Testing and Support statement for multi-instance queue managers can sometimes pass these tests.
  - So it is important that you verify that your environment is not excluded from the testing and support statement (http://www.ibm.com/support/docview.wss?&uid=swg21433474)

- **Be as thorough as possible with your tests**
  - Plan and run a variety of tests to satisfy yourself that you have covered all foreseeable circumstances.
  - Some failures are intermittent, and there is a better chance of discovering them if you run the tests more than once.
  - More detailed guidance on using the amqmfsck command can be found in the Technote at: http://www.ibm.com/support/docview.wss?uid=swg21446194.

Shared File System Requirements

- **Data write integrity**
  - The queue manager must know that written data is successfully committed to the physical device
  - Transactional system like MQ require that some writes be safely committed before continuing with other processing

- **Guaranteed exclusive access to files**
  - In order to synchronize multiple queue manager instances, a mechanism for obtaining an exclusive lock on a file is required

- **Release of locks on failure**
  - If a queue manager fails, a file system or network error to the file system occurs, etc, files locked by the queue manager need to be unlocked and made available to other processes
  - Must be possible without waiting for a failing queue manager to be reconnected to the file system.

- **A shared file system must meet these requirements for WebSphere MQ to operate reliably**
  - If it does not, the queue manager data and logs get corrupted when using the shared file system
  - These are fundamental requirements in order to ensure that messages are reliably written to the recovery log
  - These are *requirements* (NOT recommendations or suggestions)!

- **Requirements if you are using the NFS V4 as the shared file system:**
  - Hard mounts, synchronous writing and write caching must be disabled

Why Hard Mounts?

- **Soft versus Hard Mounting**
  - Govern the way the NFS client handles a server crash or network outage
  - Key advantage of using NFS is that it can handle this gracefully
  - Allow an application (MQ in this case) to KNOW the state of a failed write

- **Hard Mounts**
  - When accessing a file on an NFS hard mount, if the server crashes MQ will hang
    - This is the good (for us) effect of a hard mount
    - When the NFS server is back online the NFS client can reconnect and continue
      - Or if MQ fails the other instance can have a go at it

- **Soft Mounts**
  - If a file request fails, the NFS client will not hang; it will (maybe) report an error
  - But there is no guarantee that the file request did not actually write some data
    - This is a recipe for corrupted files and lost data
  - You can only use soft mounts safely if you don't care that you might lose some data
    - MQ does not (cannot) tolerate this

- **For this reason, multi-instance will not tolerate soft mounts**

Why Sync (rather than async)?

- **These options determine how data is written to the server on a client request**

- **Whatever you do on an NFS client is converted to an RPC equivalent operation**
  - So that it can be send to the server using RPC protocol
  - How these calls are handled differ when using async vs sync

- **Using async permits the server to reply to client requests as soon as it has processed the request and handed it off to the local file system**
  - Without waiting for the data to be written to stable storage
  - This yields better performance, but at the cost of possible data corruption
    - e.g.if the server reboots while still holding unwritten data and/or metadata in its cache
  - Async basically instructs the server to "lie" to the client, telling it the data is hardened when it is not
  - If async is used MQ may continue to run apparently fine
    - Because the possible data corruption may not be detectable at the time of occurrence
    - But there might be a "hole" in the log, potentially making recovery impossible

- **Using sync does the reverse**
  - The server will reply only after a write operation has successfully completed
  - Which means only after the data is completely written to the disk

- **You should NEVER use the async option when dealing with critical data**
  - Data loss happens with async because the client thinks data was committed (server reports that the write is committed) before it actually is
  - If the server crashed before actually committing any data, this would not be known by MQ
  - With sync, we KNOW the state of the data on the server, and so can recover cleanly after a failure

Why intr (rather than nointr)?

- **In NFS V4, a file operation will normally continue until an RPC error occurs, or until it has completed successfully**
  - And if mounted hard, most RPC errors will not prevent the operation from continuing
    - Even if the server is down, the process making the RPC call will hang until the server responds
- **intr permits NFS RPC calls to be interrupted**
  - Forcing the RPC layer to return an error
  - For MQ to fail over to a standby instance, RPC calls must be interruptible
- **nointr will cause the NFS client to ignore signals**
  - Including those that would allow a queue manager to fail over

What about Attribute Caching?

- **noac (no attribute caching) is recommended**
  - Suppresses attribute caching and forces file attributes to be kept updated in the NFS client
  - This will guarantee that on a read/write the NFS client will always have the most recent state of the file attributes
- **Under normal circumstances MQ will operate correctly with attribute caching**
  - But issues can arise when multiple NFS clients are contending for write access to the same file
    - Such as the NFS clients associated with the active and standby MQ instances
- **Cached attributes used by each NFS client for a file might differ**
  - An example of files accessed in this way are queue manager error logs
  - Error logs might be written to by both an active and a standby instance
  - Result can be that the error logs grow larger than expected before they roll over
- **Because of this, noac is recommended**
  - You can use the NFS ac* options to try and fiddle with this
  - But it's probably more trouble than it's worth

NFS Mount Example (and it's only an example)

- **A typical NFS mount will look something like this:**

  us-0a00-nas01t:/mqrt_reg01 /nas/mqm/mqrt_reg01 nfs
  rw,bg,**sync**,**hard**,**intr**,rsize=131072,wsize=131072,tcp,**noac**,vers=4

- **Critical to note:**
  - Hard (required)
  - Sync (required)
  - intr (required)
  - noac (recommended)

- **An NFS mount can have many other options**
  - These can vary from vendor to vendor
  - So there is no "standard" or "recommended" configuration beyond those required
  - Work with your file system staff and vendor(s) to get the best performance and stability

Checking Queue Manager Status (1)

- **The dspmq command will identify instance status and mode:**

```
C:\> dspmq -x
QMNAME(chris)                              STATUS(Running)
   INSTANCE(MPLS1A) MODE(Active)
   INSTANCE(MPLS1B) MODE(Standby)
```

- **This is a multi-instance queue manager with two instances**
  - The active instance on MPLS1A and the standby instance on MPLS1B

Checking Queue Manager Status (2)

- **A multi-instance queue manager has additional status values that can be reported. Some examples:**
  - **"Running as standby"**
    - The queue manager is defined here
    - There is a standby instance running locally
    - It holds the "standby lock", polling the master and active locks in anticipation of the failure of the active instance
  - **"Running elsewhere"**
    - The queue manager is defined here
    - There is no instance running here
    - There's an active instance on another machine
      - The master and active locks are held
      - qmstatus.ini reports "Running"

Queue Manager States
Starting
Running
Running as standby
Running elsewhere
Quiescing
Ending immediately
Ending pre-emptively
Ended normally
Ended immediately
Ended unexpectedly
Ended pre-emptively
Status not available

- **dspmq queries the lock files in order to report the status**

Queue manager status – More detail

- **qmstatus.ini contains several values related to multi-instance:**
  - PermitStandby = Yes | No
    - Indicates whether the active instance was started permitting standby instances
    - This is checked when the execution controller wants to become a standby instance
  - PermitFailover = Yes | No
    - Indicates whether a standby instance is permitted to failover when active crashes
    - This is used to prevent a queue manager which crashes as it starts up from doing it again
  - PlatformSignature = <numeric>
    - Indicates which platform owns the data
    - Prevents failover between different architectures and OSes
  - PlatformString = <string>
    - A string version of the platform signature used when reporting a mismatch between the running code and the qmstatus.ini

Lock Files (1)

- **Three files are used to ensure single activation and report status:**
  - **master**
    - Held in exclusive mode by the Execution Controller of the active instance
  - **active**
    - Held in shared mode by multiple queue manager processes, plus fastpath applications
  - **standby**
    - Held in exclusive mode by the Execution Controller of the standby instance

- **The lock files are used to coordinate the instances and by dspmq to report status for a multi-instance queue manager**

- **The master and active locks are held even by a normal queue manager**
  - Prevents accidental dual activation, even if multi-instance not being used

Lock Files (2)

- **An undocumented flag ("f") on dspmq lets you see the state of the file locks:**

```
C:\> dspmq -xf
QMNAME(chris)                          STATUS(Running)
    INSTANCE(MPLS1A) MODE(Active)
    INSTANCE(MPLS1B) MODE(Standby)
        master(MPLS1A,1249388074)
        active(MPLS1A,1249388074)
        standby(MPLS1B,1249814329)
```

- **The master, active and standby files contain a little data about the lock holder:**
  - Hostname
  - Lock id (Identifies the queue manager instance)
  - Lock time

- **When an instance starts, it calculates the lock id which it writes into the lock files that it owns**

How are Queue Manager Lock files used?

- **Periodically, in a multi-instance queue manager, lock files are reread and if the lock id doesn't match, the lock has been stolen**
  - A lock file should never be stolen, and NFS should renew its leases automatically without MQ having to repeatedly use the locked files.
    - But a queue manager won't notice a lease expiring unless it periodically rereads its lock file
    - So a "verify" thread reads the contents of the master file lock every 10 seconds
      - A Tuning Parameter is available to change this if needed
  - Because reading a file can block during a network outage, a "monitor" thread ensures that the verify thread is making progress checking the file
  - If the verify thread stalls for 20 seconds, or the reading of the file lock fails, or the lock owner in the file changes, the queue manager "commits suicide"

```
AMQ7279: WebSphere MQ queue manager '&3' lost ownership of data lock.
Explanation: The instance of queue manager &4 has lost ownership of a lock on its
data in the file-system due to a transient failure. It was not able to re-obtain the
lock and will stop automatically to prevent the risk of data corruption.
User response: Check that another instance of the queue manager has become active.
Restart this instance of the queue manager as a standby instance. If this problem
recurs, it may indicate that the file-system is not sufficiently reliable to support
file locking by a multi-instance queue manager.
```

Other files that are locked

- **A multi-instance queue manager takes file locks on other files too:**
  - The log control file and log extents (exclusive locks)
  - The files for queues and other MQ objects (exclusive locks during restart, otherwise shared locks)

- **These locks are an important part of the data integrity of the queue manager**

- **Also, NFS V4 performs better when these locks are held**
  - By holding a lock, data is written more eagerly to the filesystem (less buffering)
  - The implication of the lock is that the data is shared between machines

- **By holding a lock, you can tell whether a network outage occurred during which a conflicting lock was granted by the filesystem**
  - Without these locks, queue manager files (log, etc) could be corrupted

Health checking

- **Health-checking also takes place between queue manager processes**
- **The aim is to prevent orphaned processes for a failed queue manager**
  - Eliminate need for manual cleanup after a failure
  - MQ processes and utility managers monitor the health of the Execution Controller
- **MQ Processes don't try and continue on after a failure**
  - Some of these would just not die
  - Effect was often to make failures last longer, rather than avoid them

Liveness Checking

- **Multi-instance queue managers also have a liveness checking thread**
  - Only multi-instance queue managers have this
  - Ensures that the queue manager is able to do productive work
    - e.g. That the logger is making progress with writing log records
  - Checks are very carefully handled to ensure QM doesn't just blow up when it's very busy (e.g. when using an external HA solution like Veritas)
  - Checks every 60 seconds by default
    - A Tuning Parameter is available to change this if needed
  - The liveness checking runs on a separate thread and shoots the process issuing the actual I/O requests if it takes too long
    - This results in the queue manager "committing suicide"

AMQ7280: WebSphere MQ queue manager '&3' appears unresponsive.
Explanation: The queue manager is monitoring itself for responsiveness. It is not responding sufficiently quickly and will automatically stop if it continues to be unresponsive.

Problem Diagnosis – File systems

- **The first problem that most people encounter is setting up the networked storage**
  - uid/gid mismatch
  - Incorrect file permissions
  - amqmfsck will diagnose these with a sensible message

- **It's vital that file locking actually works as advertised**
  - amqmfsck –w is your best friend here (tests waiting and releasing locks)
  - It can be used to check that locks are handed off between processes and machines
  - Make sure your file system is supported!
    - http://www.ibm.com/support/docview.wss?&uid=swg21433474

- **File system and network tuning are important!**
  - NFS client, NAS server, network, etc
  - Poor performance can result in stalls and spurious fail-overs
  - NAS remote backup, ETL jobs, etc can also trigger spurious fail-overs

Problem Diagnosis – File integrity

- **The MQ code has been carefully designed to eliminate file integrity problems during failover**
  - However it does depend on the file system behaving correctly
  - Some file systems do not pass because they've been found to permit a failed write() call issued before a network outage to manage to write some data after the outage, even though the call failed
    - Can result in log corruption (characterised by a "hole" in the log)
    - May never be noticed, but media recovery will stop with "Media image not available"
    - May result in queue corruption if restart processing reads the mangled data

- **amqmfsck –i can be used to diagnose this**
    - It's essentially the same sequence of calls as the logger and will diagnose an integrity problem caused by a network outage

Problem Diagnosis – "Spurious failovers"

- **Occasionally, customers report spurious queue manager failovers**
  - Stand-alone queue managers on the same infrastructure would be unaffected
- **Could be triggered by the liveness-checks failing**
  - Stand-alone queue managers do not have this
- **Cause is often poor file system performance**
  - Someone running an ETL job, remote file back-up, etc

Summary

- The Multi-instance feature has been around some time now (5 years)
- File system must be NFS v4, with hard mounting, sync writes, I/O interruptible and caching disabled
- Control commands enhanced to report status of multi-instance queue managers
- File locking used to coordinate between instances on separate machines
- File locking also used to protect queue manager file integrity
- Configuration, monitoring and tuning of underlying file system important
- Problems usually involve file system issues

Questions?