MQ Performance Tuning

or

How fast is fast enough!

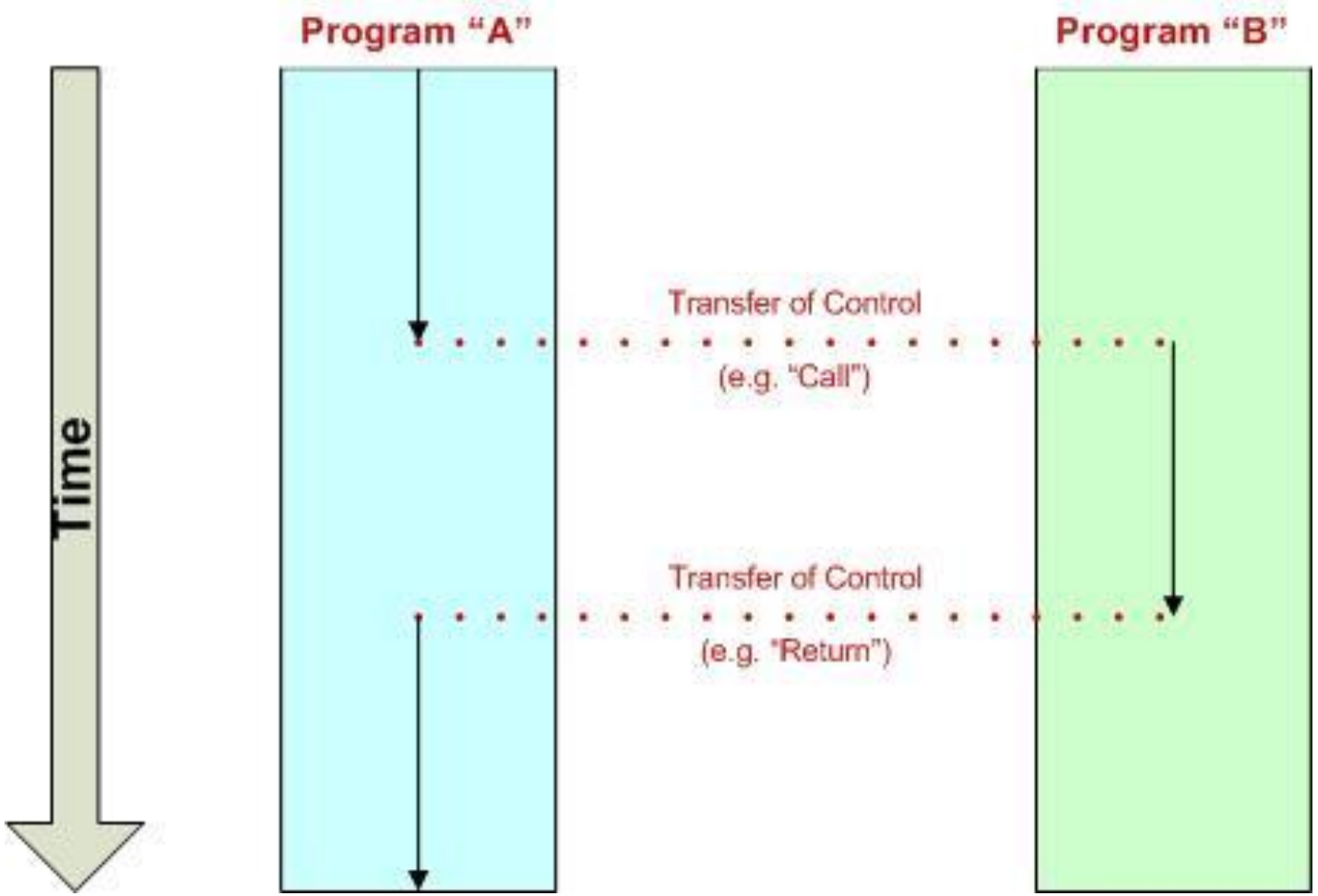
Table of Contents

Background Information

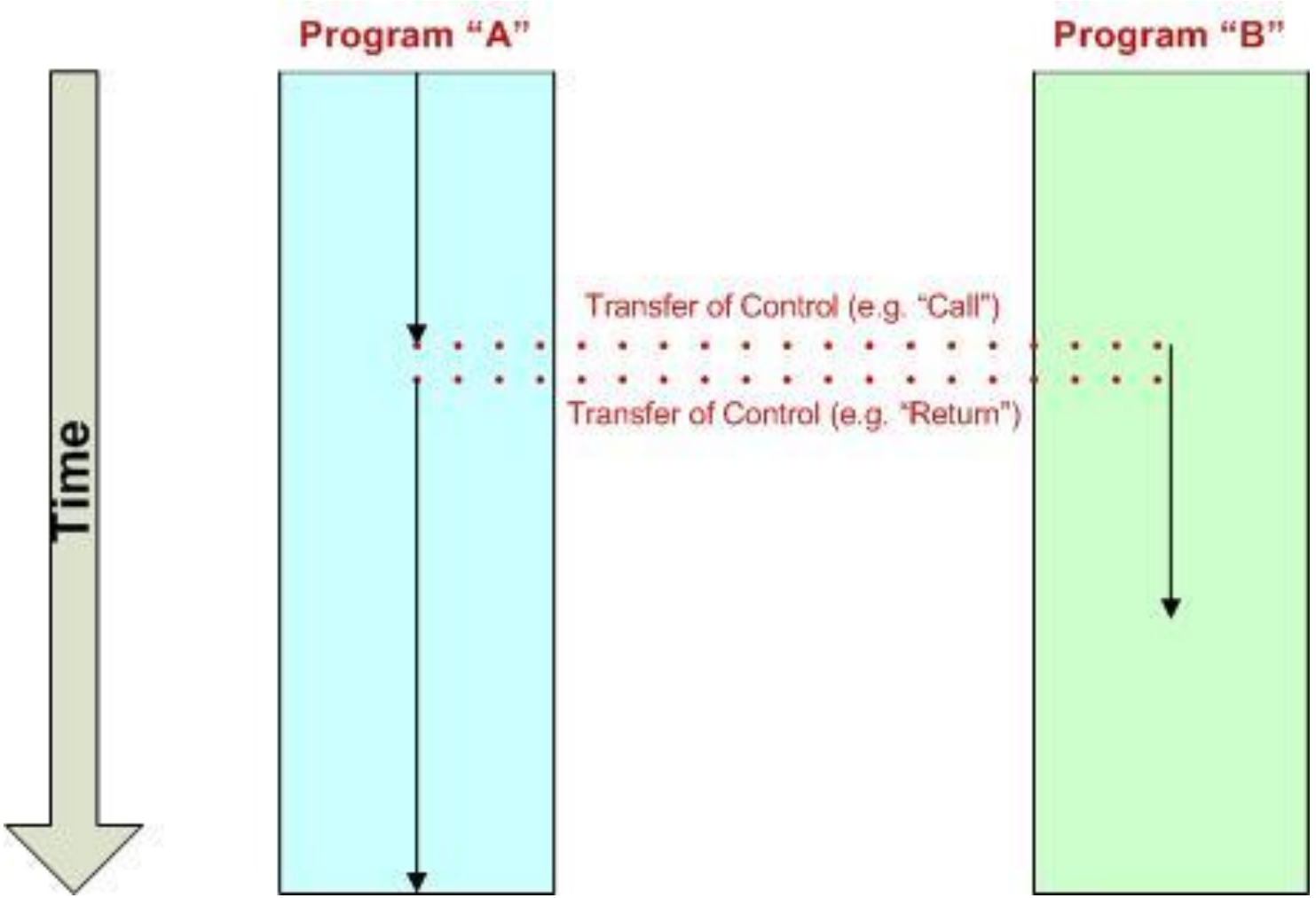
Synchronous versus Asynchronous

- **Synchronous Processing**
  - Communicating programs (e.g. Application & Database) tightly coupled in time
  - Calling program is blocked from executing while it waits for called program to complete
  - Delays in processing by called program are experienced by the calling program.
  - No backlogs, just increased latency!
- **Asynchronous Processing**
  - Communicating programs (e.g. Application and WMQ) loosely coupled in time
  - Calling program IS NOT blocked from executing while called program completes its work
  - Delays in processing by called program ARE NOT experienced by the calling program
  - No increased latency, just backlogs!
- **WMQ processing is always asynchronous!**
- **Programming patterns may simulate synchronous behavior**
  - Link two asynchronous calls together by the calling program. For example:
    - MQPut (Write a Message)
    - MQGet with a wait for response (Read a Message)

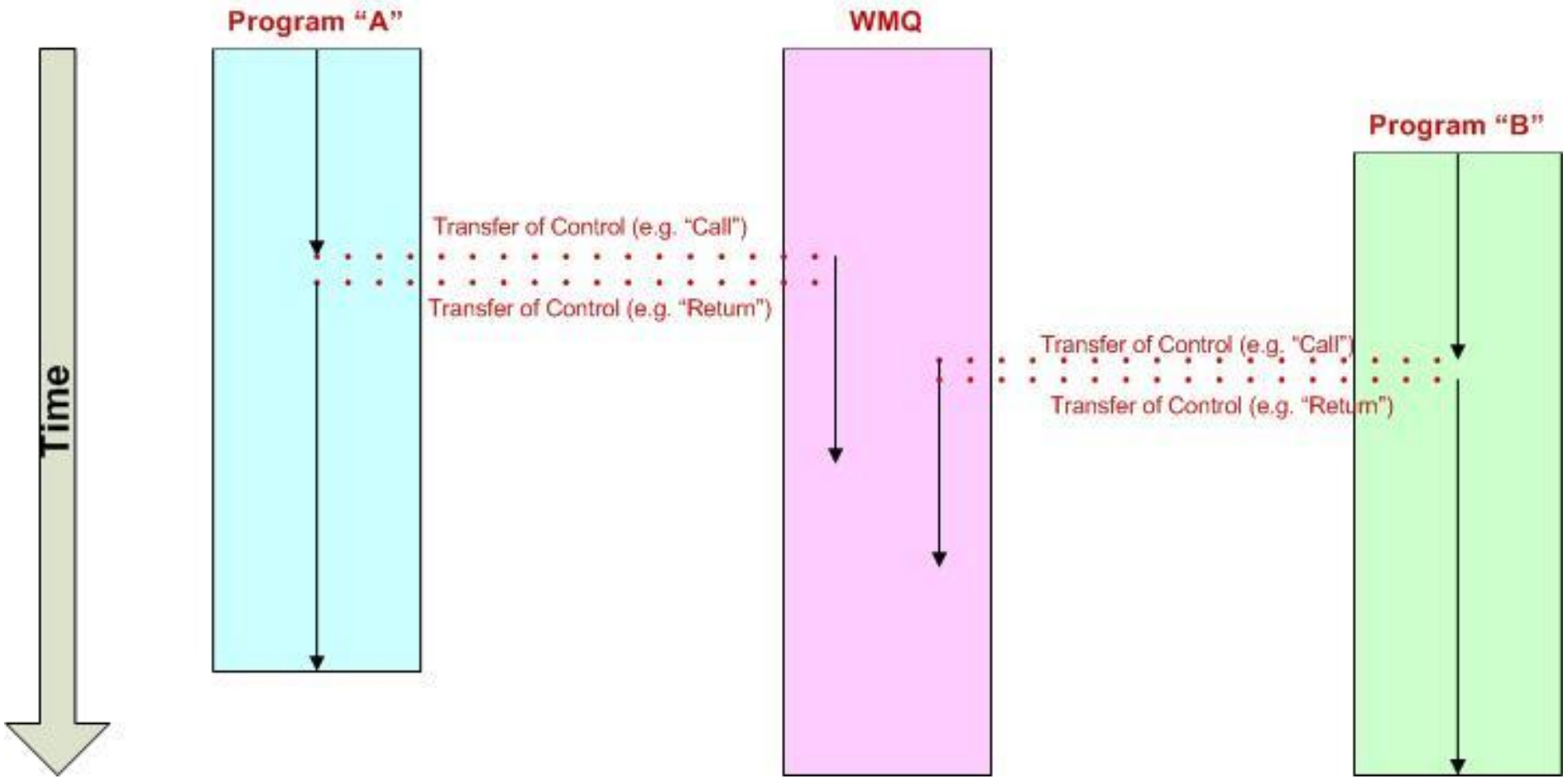
Synchronous Processing Diagram



Asynchronous Processing Diagram



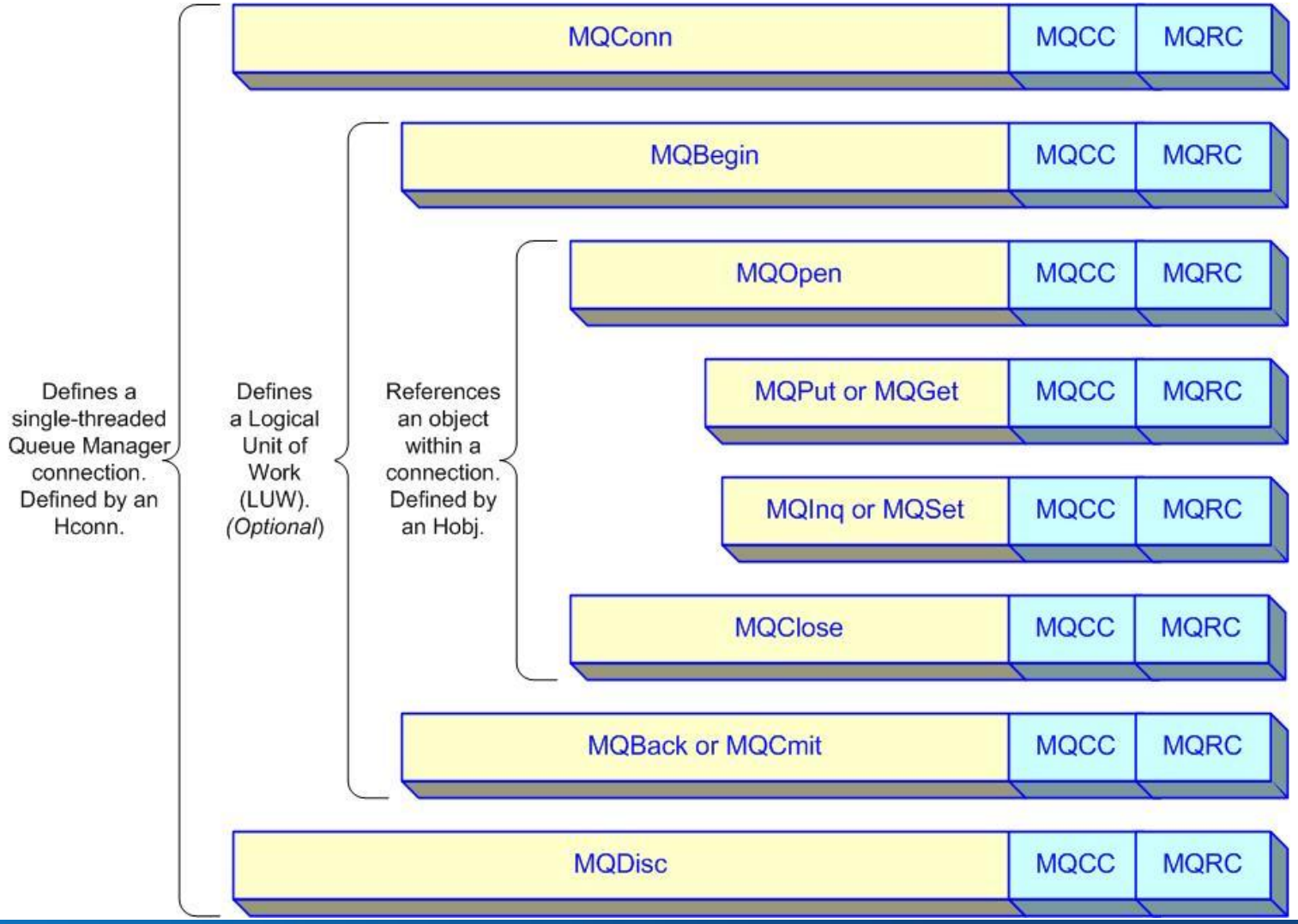
Multiple Asynchronous Processes Diagram



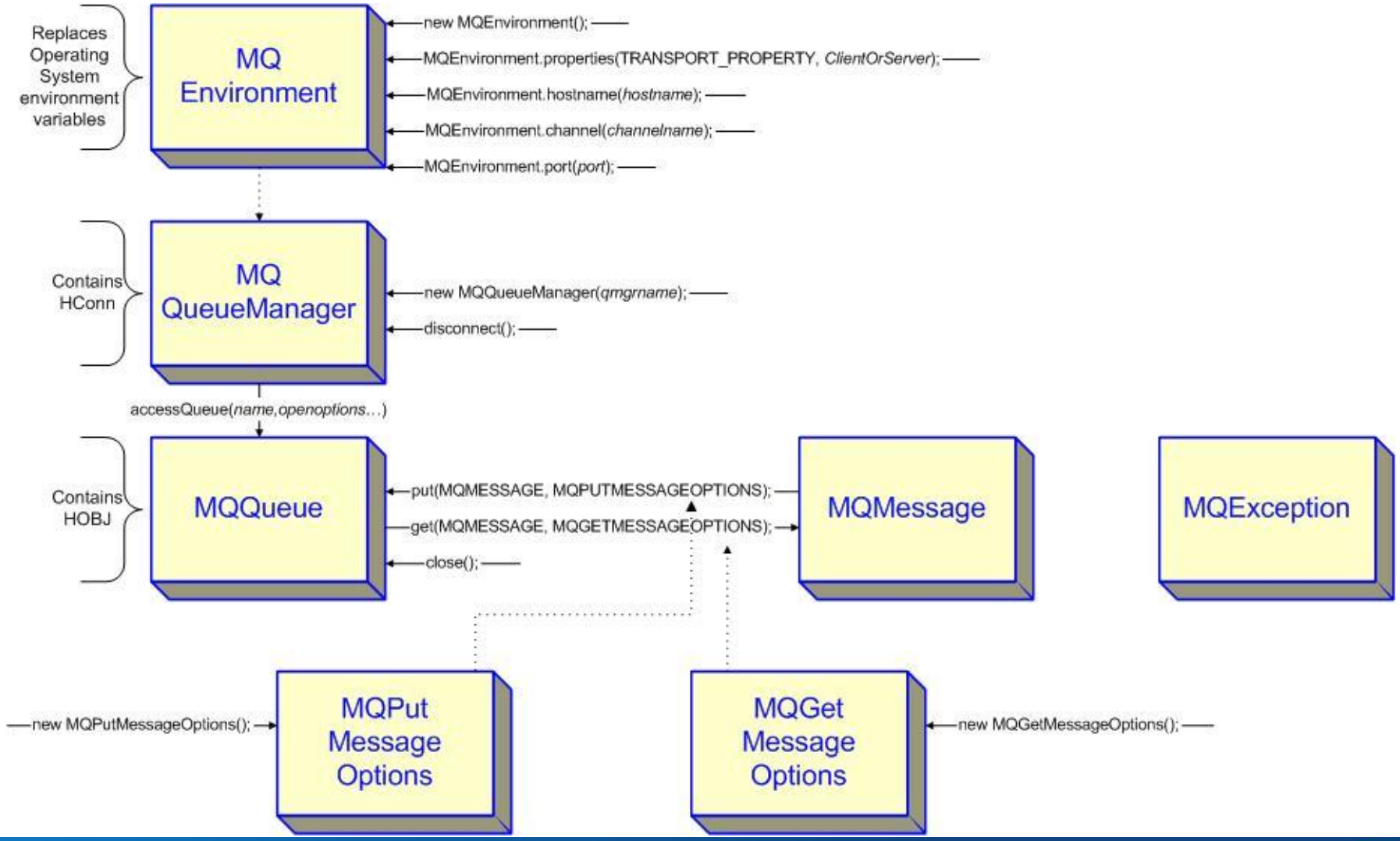
WMQ Programming Interfaces

- **MQI**
  - Original API
  - For Procedural languages (C, COBOL, RPG, Visual Basic)
- **WMQ Classes for Java**
  - First Java API for MQSeries
    - Designed by IBM, initially as SupportPac MA88
    - Predates JMS
- **WMQ Classes for JMS**
  - Second Java API for MQSeries
    - Designed by Sun Microsystems
    - Intended to be platform agnostic, but heavily influenced by the WMQ Classes for Java
    - Designed for feature set, not necessarily for performance
      - Message Selectors
- **WMQ Classes for .NET, ActiveX, and C++**
  - Supplied by IBM to support specific language and runtime environments

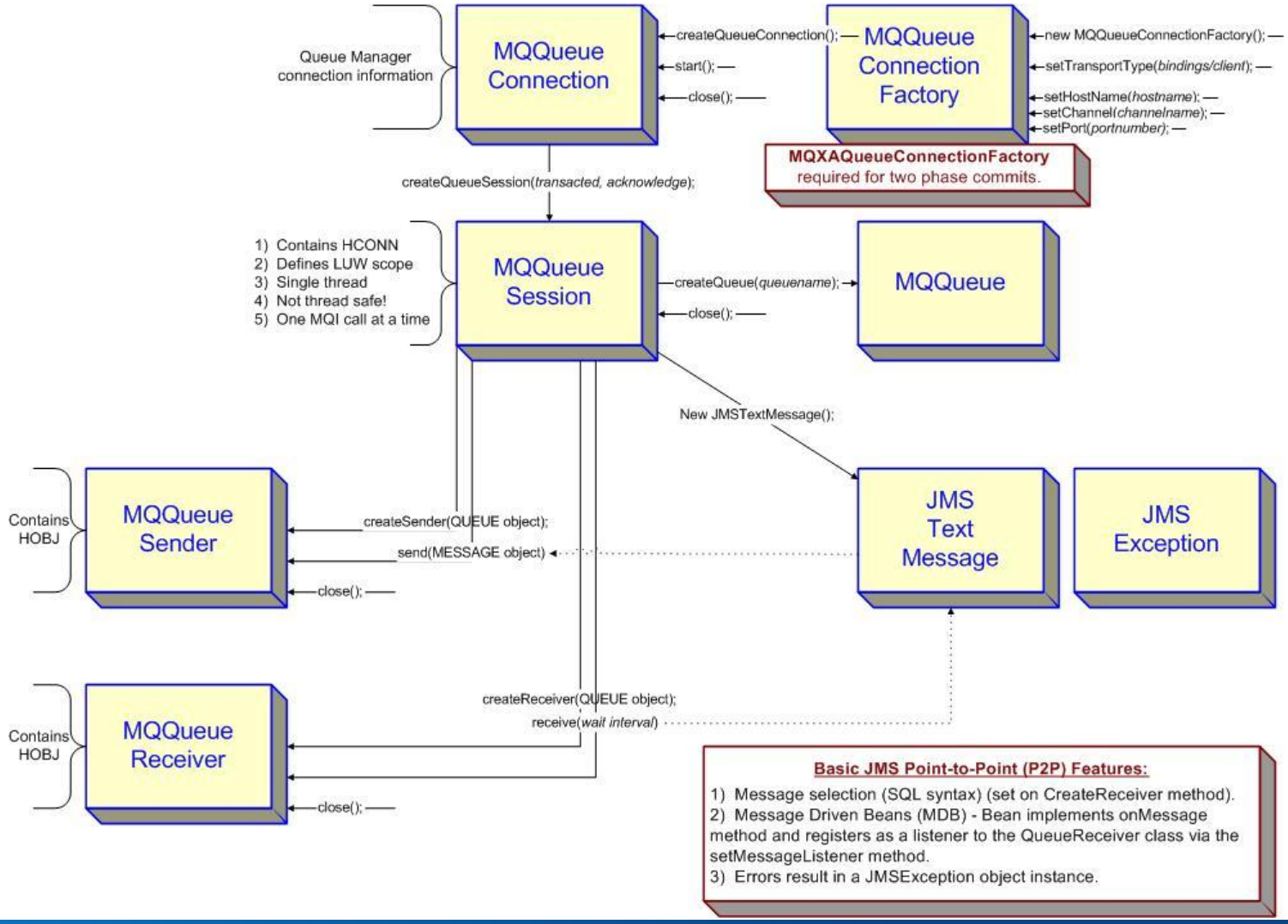
MQI (Message Queue Interface)



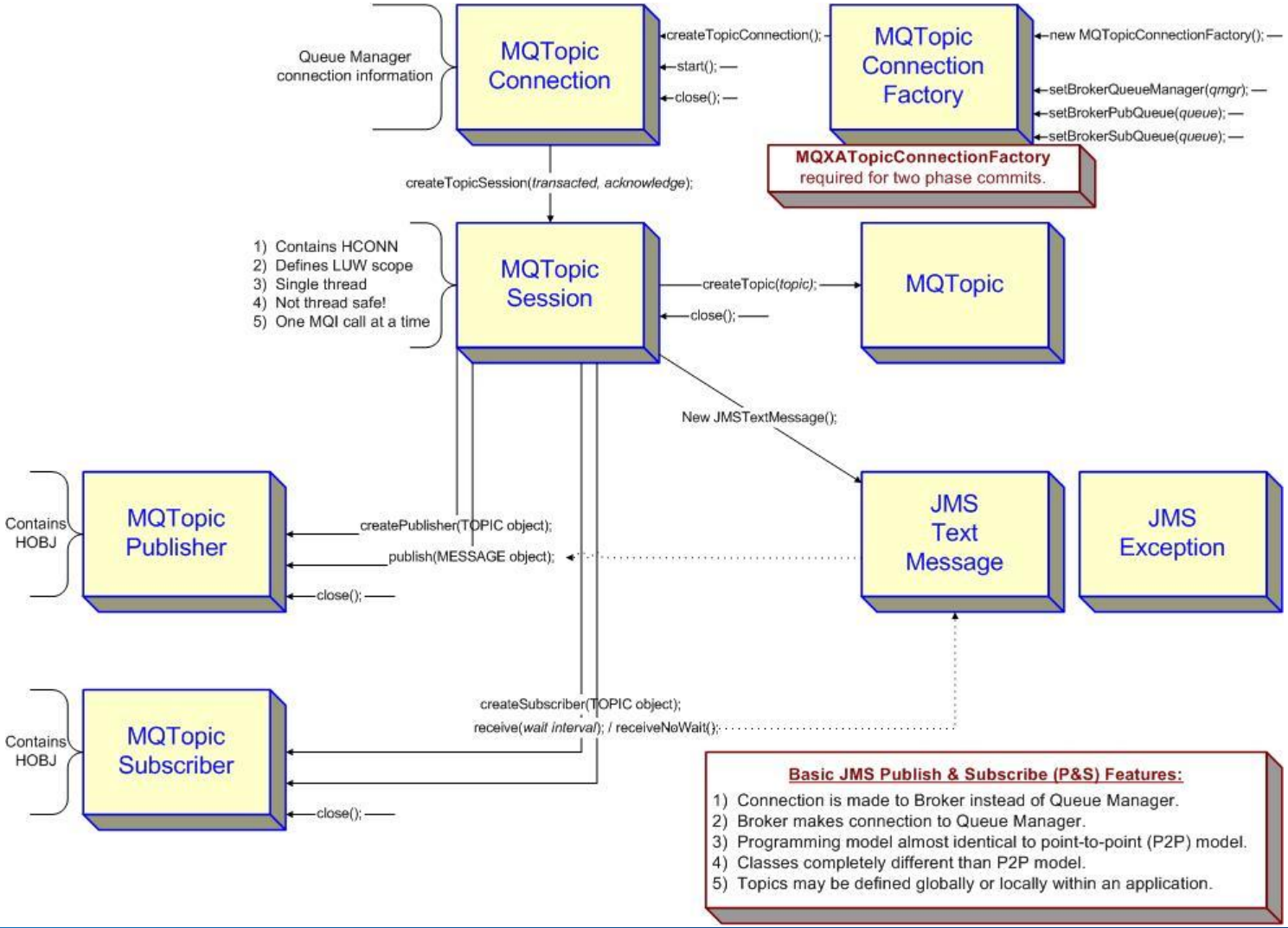
WMQ Classes for Java



WMQ Classes for JMS (Point to Point)



WMQ Classes for JMS (Publish/Subscribe)



Key API Points

- **Different APIs will have different performance characteristics**
- **Different API calls have different costs**
  - "Connect "is the most expensive call (in terms of latency)
  - "Get" calls have things to consider
    - Message Filters – response times degrade as queue depths increase
    - Lock Contention – response time degrades as number of "Readers" increases
- **API Calls are one of the WMQ Bottlenecks!**
  - Maximum number of API calls / second based upon the API call path length
  - Application Architects and WMQ Administrators should know this number!
  - Easy to determine, use the "Q" program: SupportPac MA01 (Thank you Paul Clarke)
    - crtmqm TempQmgr
    - strmqm TempQmgr
    - echo "define qlocal('TempQueue')" | runmqsc TempQmgr
    - date
    - echo "#!1000000/1024" | /…path…/q -m TempQmgr -ap –p1 -O TempQueue
    - date
    - The preceding commands write 1,000,000 messages of 1K size

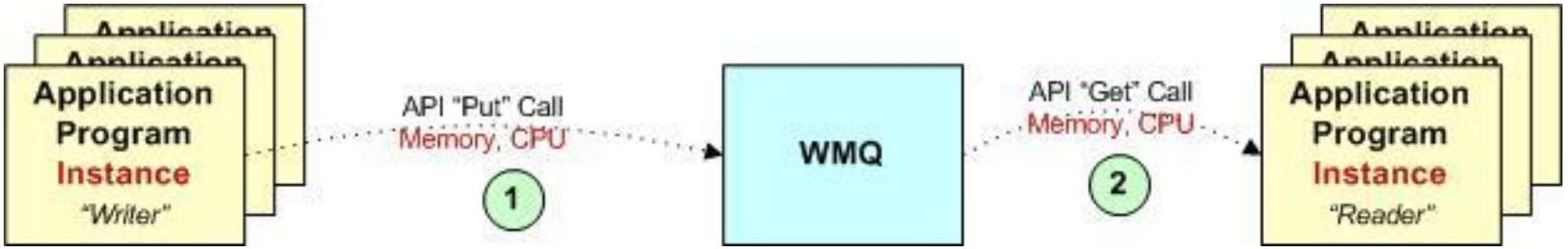
WMQ Message Processing

Key WMQ Internal Processing Points

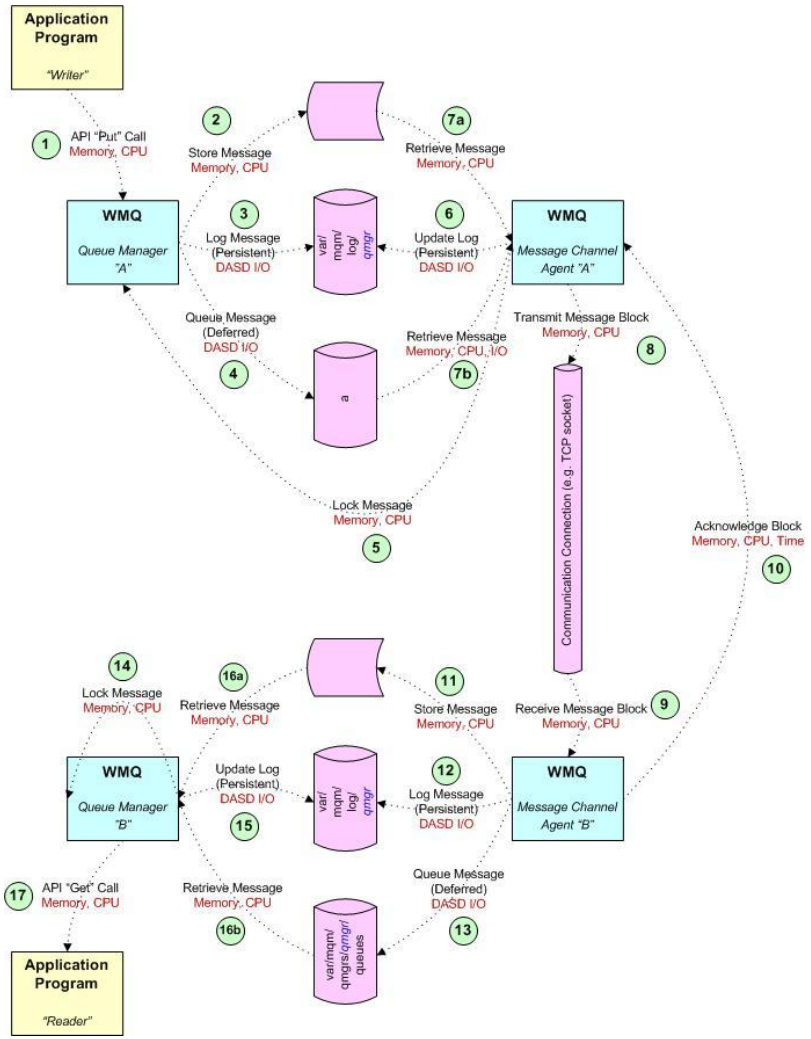
- **API Calls**
  - Each Connection Handle (HCON) is associated with a single thread!
  - API calls through the same Connection Handle are single-threaded!
  - API Path Length is roughly 5 ms, resulting in about 200 MQ calls per second.
- **Persistent messages are written to the log**
  - Message cannot be released to the application until the log write completes.
  - Non-persistent messages are roughly 10 times faster than persistent messages!
- **WMQ channel protocol is a blocking protocol**
  - MCA waits for an acknowledgement after each block is transmitted.
    - Impacted by Batch Size (BATCHSZ) parameter.
    - Impacted by the Batch Interval (BATCHINT) parameter.
  - MCA agents on each Queue Manager must update the log for persistent messages.
  - Multiple channels between Queue Manager pairs will significantly increase throughput.
  - Message delivery sequence is generally “First In First Out” (FIFO)
    - Separating large from small messages can yield significant QoS improvements

Message Processing – Application View

- Application view of WMQ
  - WMQ is a “Black Box”
  - Puts go in, Gets come out
- Application view of Puts
  - Due to asynchronous nature of the “Put”, application unaware of the impacts
  - From application perspective, more “Puts” equals more throughput
- Horizontal Scaling
  - Applications will add additional instances and/or threads
  - Often without regard, understanding, or consultation with MQ administrators

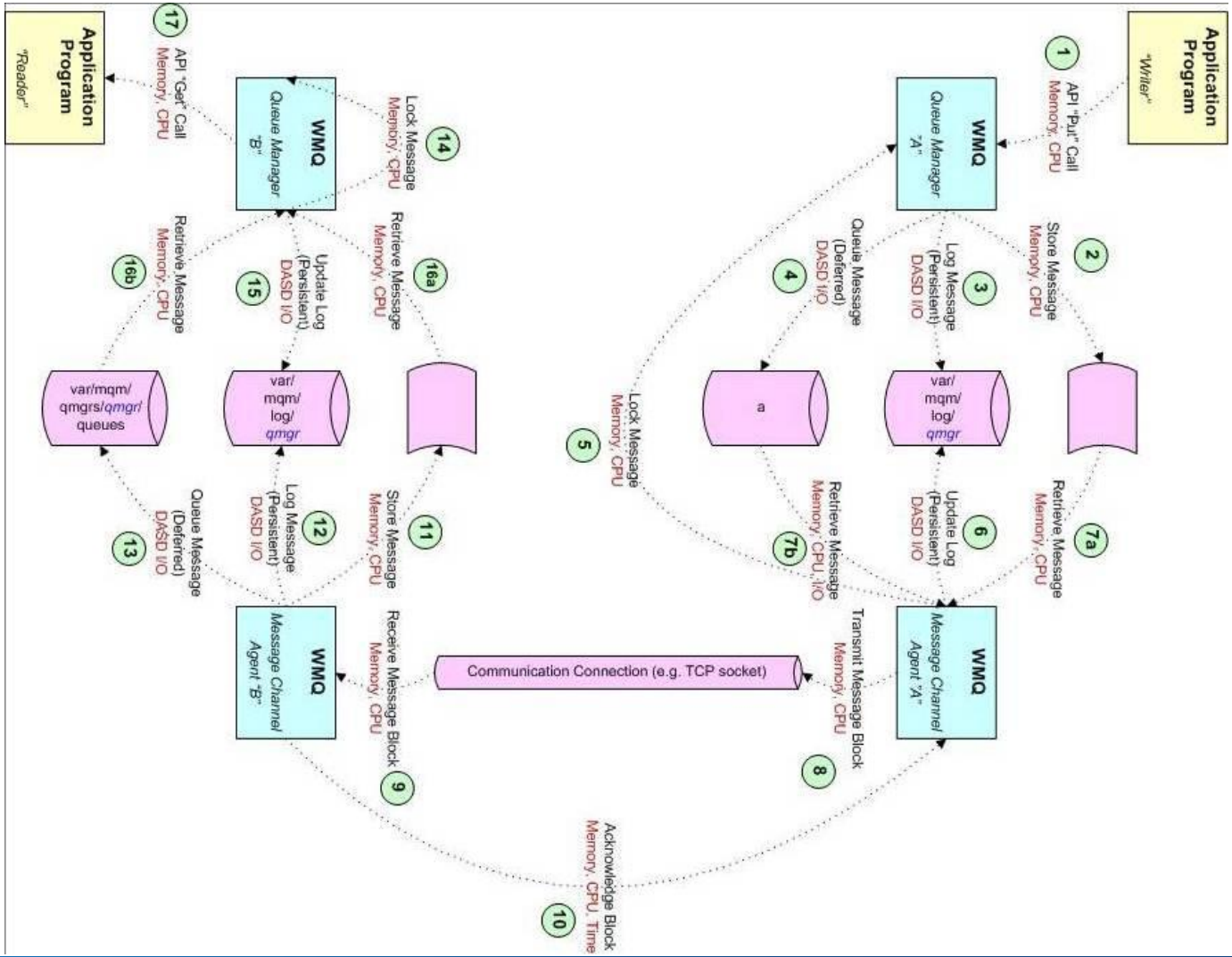


Message Processing – MQ View #1



- (01) MQPut by application
- (02) Move message to WMQ memory
- (03) Write persistent message to log
- (04) Move message from RAM to Disk
- (05) Acquire message lock
- (06) Update Log
- (07) Retrieve message (RAM or Disk)
- (08) Transmit message block
- (09) Receive message block
- (10) Acknowledge receipt of block
- (11) Move message into WMQ memory
- (12) Log Message
- (13) Move message from RAM to DISK
- (14) Acquire message lock
- (15) Update Log
- (16) Retrieve message (RAM or Disk)
- (17) MQGet & commit by application

Message Processing – MQ View #2



WMQ Performance Tuning

What does Performance Tuning Require?

- **Understanding your specific WMQ Infrastructure**
  - Infrastructure Topology
    - Clusters, Queue Managers, Channels
    - Brokers, Execution Groups
  - Application Resources
    - Servers, Languages, External Software (WMQ, WMB/IIB, DB2/Oracle, etc.)
- **Understanding WMQ Processing at a Resource Level**
  - CPU, Memory, Disk, Network
- **Creating a Performance Model**
  - Abstracting detailed WMQ internal processing into key steps & bottlenecks
  - Identifying key measurement points
    - Processing steps that can be directly measured
    - Processing steps that can be inferred
- **Key Measurement Points**
  - Identifying what metrics can be measured
  - Identifying how those metrics can be gathered

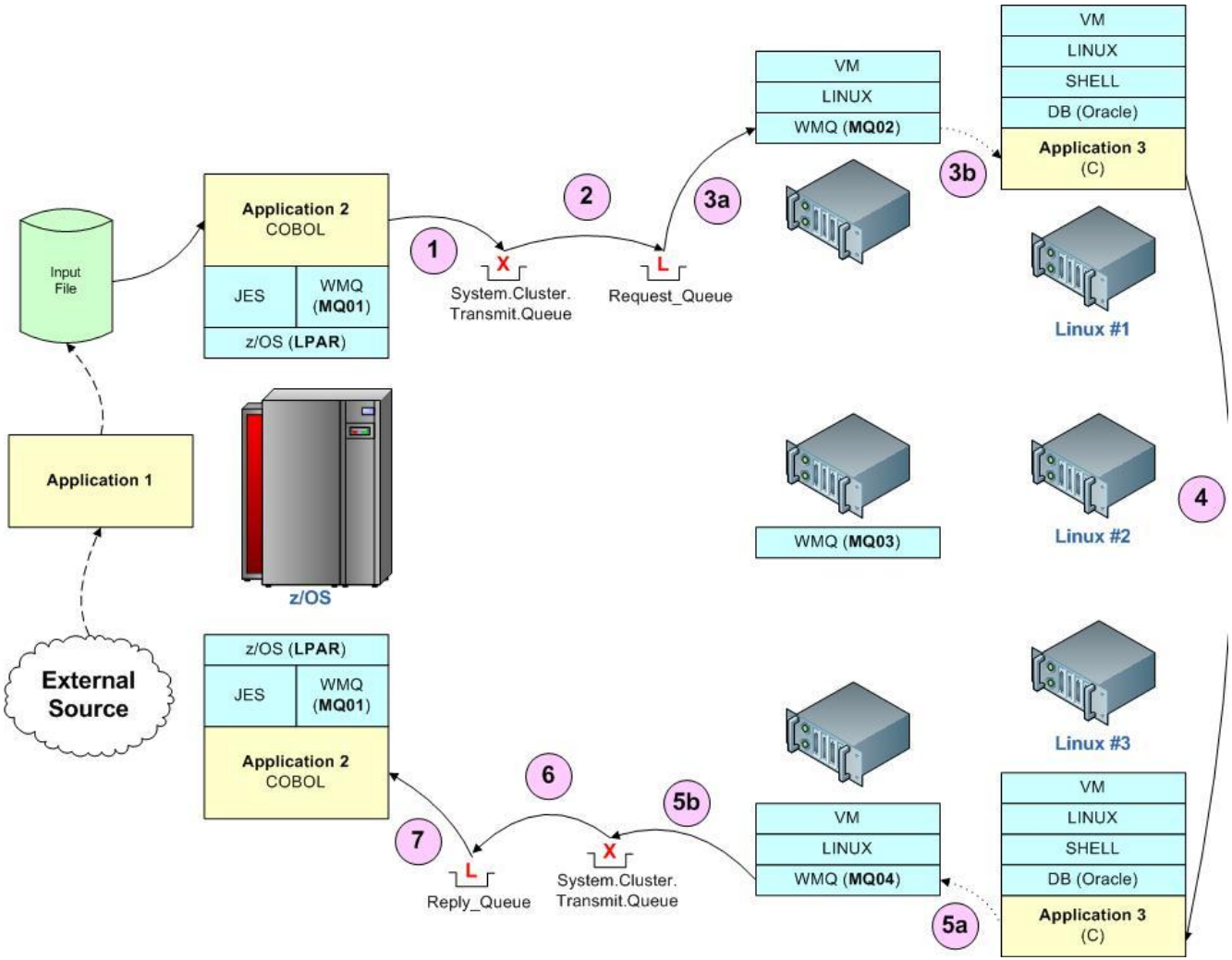
Key Concepts

- **Memory versus Disk**
  - Processing messages in RAM is MUCH FASTER than processing them from DASD
  - WMQ has a limited amount of memory available to each queue.
    - This can be altered but normally should not be!
    - Once the memory for the queue is exhausted, messages must be written to disk.
      - Not Logging
      - /var/mqm/qmgrs/*QmgrName*/queues
      - Once this happens, all future messages will be read from disk until processing catches up
- **Impedence**
  - Enqueue (arrival) rates versus Dequeue (consumption) rates:
    - If messages arrive faster than they can be processed they must be queued!
      - Messages will first be queued to memory,
      - When memory is full, then they will be written to disk.
    - The amount of memory buffering should be thought of in time (e.g. seconds).
    - If arrival rates exceed comsumption rates, no amount of RAM will be enough.

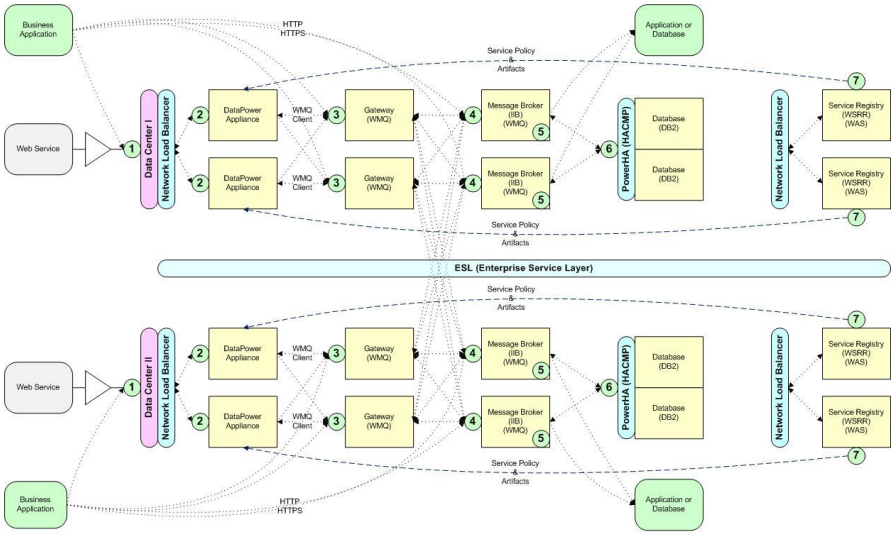
Key Concept Implications

- **System must be visualized as two separate functional components**
  - “Writers” generate messages.
  - “Readers” consume messages.
  - “Reader” capacity, on average, MUST exceed “Writer” capacity, on average.
- **Horizontal Scaling**
  - Both “Readers” and “Writers” are single-threaded through the Connection Handle
  - The API Path length limits the number of Gets or Puts per second per thread
  - The only way to increase these twin limitations is additional threads:
    - Additional instances of the application (e.g. more servers)
    - A multi-threaded application
- **You can’t tune without information**
  - Application topology
  - WMQ topology
  - Benchmark data

WMQ Process Model – Sample #1



WMQ Process Model – Sample #2



1. Benchmark load into a NLB URL. Web Calls per second from a single source (e.g. single TCP/IP socket).
   Benchmark load into a NLB URL. Web Calls per second from multiple sources (e.g. multiple TCP/IP sockets).

2. Benchmark load into a single DataPower appliance. Web Calls per second from a single source (e.g. single TCP/IP socket).
   Benchmark load into a single DataPower appliance. Web Calls per second from multiple sources (e.g. multiple TCP/IP sockets).

3. Benchmark asynchronous load (Datagram) into a single Gateway Queue Manager MQI Channel. WMQ Puts per second from a single source/thread (e.g. single Backend Handler).
   Benchmark load into a single Gateway Queue Manager MQI Channel. WMQ Puts per second from multiple sources (e.g. multiple Backend Handlers).
   Benchmark load into a multiple Gateway Queue Manager MQI Channels. WMQ Puts per second from multiple sources (e.g. multiple Backend Handlers).
   Repeat tests for synchronous Request/Reply loads (Put & Get with wait).

4. Benchmark asynchronous load into a single Message Broker Queue Manager (One channel). WMQ Puts per second from a single source (Transmit Queue). (1K payload).
   Benchmark asynchronous load into a single Message Broker Queue Manager (Two channels). WMQ Puts per second from a multiple sources (Transmit Queues). (1K payload).
   Benchmark asynchronous load into a two Message Broker Queue Managers (Two channels). WMQ Puts per second from a multiple sources (Transmit Queues). (1K payload).
   Benchmark asynchronous load between a single Gateway and Broker across the Data Centers. WMQ Puts per second from a single source (Transmit Queue) (1K payload).
   Benchmark asynchronous load between a single Gateway and Broker across the Data Centers. WMQ Puts per second from multiple sources (Transmit Queues) (1K payload).

5. Benchmark asynchronous load (Datagram) into a single Message Broker Execution Group. WMQ Gets per second from a single source (Local Queue).
   Benchmark asynchronous load (WMQ Datagram) into multiple Message Broker Execution Groups. WMQ Gets per second from a single source (Local Queue).
   Benchmark synchronous load (Request/Reply) into a single Message Broker Execution Group. WMQ Get with Puts per second from a single source (Local Queue).
   Benchmark synchronous load (Request/Reply) into multiple Message Broker Execution Groups. WMQ Get with Puts per second from a single source (Local Queue).
   Benchmark asynchronous load (SOAP Calls) into a single Message Broker Execution Group. Calls per second from multiple sources (TCP/IP Sockets).
   Benchmark asynchronous load (SOAP Calls) into a single Message Broker Execution Group. Calls per second from a single source (TCP/IP Socket).

6. Benchmark SQL Calls into a single table. SQL operations per second from a single source (thread). (Create, Read, Update, Delete).
   Benchmark SQL Calls into a single table. SQL operations per second from a multiple sources (threads). (Create, Read, Update, Delete).

7. Benchmark WSRR Calls to a single node. Calls per second from a single appliance (thread).
   Benchmark WSRR Calls to a single node. Calls per second from multiple appliances (threads).

Performance Benchmarking

Infrastructure Benchmarking – Execution 1

- **Benchmarking Goals**
  - Identify Key Bottlenecks at the component/thread level
  - Evaluate horizontal scaling solutions for bottlenecks
  - Establish baseline performance numbers
    - Infrastructure
    - Application
- **Benchmark Infrastructure First**
  - WMQ & Message Broker Capacities
  - Test Load Generation
    - Begin with single thread generating messages
    - Add threads until backlog develops (e.g. Queue Depth > 0)
  - Horizontally Scaling Behavior
    - Adding Queue Managers (Clustering)
    - Adding Channels (Parallel Processing)
  - Traffic Behavior
    - Client Bindings versus Server Bindings
    - Persistent versus Non-persistent messages
    - Small Message sizes versus Large Message sizes

Infrastructure Benchmarking – Execution 2

- **Test Execution**
  - Leave End-to-End tests for last (unless a “smoke test” is needed first)
  - Test Performance Model one hop at a time
    - Tests are simpler to understand and execute
    - Results illustrate basic capacities and bottlenecks
    - This approach tends to more easily identify tuning opportunities
- **Test Tools**
  - WebSphere MQ Settings
    - MonQ and MonChl
  - MA01 SupportPac (“Q” program) by Paul Clarke
    - A “must have” tool.
    - Easily generates single-threaded test loads.
    - Can act as a back-end application for Request/Reply testing.
  - MH04 SupportPac (“xmqstat” program) by Oliver Fisse
    - Another “must have” tool.
    - Summarizes queue statistics over duration of test.
  - SoapUI & LoadUI
    - Generate Web Service requests.

SupportPac MA01 – “q”

- **Overview**
  - Queue I/O tool
  - Category 2 SupportPac (“As Is” – no official IBM Support)
  - Authored by Paul Clarke of MQGem (formerly of IBM Hursley Laboratory)
  - Single executable to download; available for most Windows and UNIX platforms
  - More options than you will ever need (The Swiss Army knife of WMQ)
- **Usage**
  - Processing controlled by flags
    - WMQ Connection (Client or Server bindings)
    - Input and Output (File, Queue, Stdin, Stdout)
      - Each record from stdin equates to either one command or one message
    - MQ API Options (e.g. Persistence, Priority, etc.)
  - Input Data
    - Messages to be processed
    - Commands to the ‘q’ program
    - Input data may be “piped” into the command (Stdin)
      - echo “#!100000/1024 Test Message” | q –m *qmgrName* –O *queueName*

SupportPac MA01 – "q" (continued)

- **Invocation Examples**
  - q -m *QmgrName* -I *InQueue* -O *OutQueue* *(Queue → Queue)*
  - q -m *QmgrName* -f *InFile* -O *OutQueue* *(File → Queue)*
  - q -m *QmgrName* -I *InQueue* > *OutFile* *(Queue → File)*
  - q -m *QmgrName* -O *QueueName* *(Stdin → Queue)*

- **Input Commands**
  - "#" → First character indicates this is a command and not a message
  - ! → Second character indicates not to echo the command to ouput (optional)
  - 9999 → Number of messages to generate
  - /9999 → Size of message to be generated (Optional)
  - xxx → Text of message

- **Queue name specification**
  - Queue Name may be preceded by a Queue Manager name
  - Name separator characters (use only one)
    - : "#", "/", "\", or ","

SupportPac MA01 – "q" (continued)

- **Request Reply – Generating Request messages**
  - q -m *QmgrName* -O *RemoteQmgr#RequestQueue* -r *ReplyToQmgr#ReplyToQueue* -apR
    - Puts message to a request type queue on a remote Queue Manager
      - Messages are put as Request messages (-aR)
      - Messages are put as Persistent (-ap)
      - Messages specify to the "Reply To" Qmgr and Queue (-r)
- **Request Reply – Generating Reply messages**
  - q -m *QmgrName* -I *RequestQueue* -E –apr -w 300
    - Gets request message and generates a reply message
      - Messages are put as Reply messages (-ar)
      - Messages are put as Persistent (-ap)
      - Messages are written to the "Reply To" Qmgr and Queue (-E)
      - Process will wait for incoming messages 5 minutes (-w 300)
- **See slide notes for many additional parameters**
- **Better yet, see the MA01 readme.txt file!**

SupportPac MH04 - xmqstat

- **Overview**
  - Queue Statistics monitoring and reporting tool
  - Category 2 SupportPac (“As Is” – no official IBM Support)
  - Authored by Oliver Fisse of IBM Software Group (ISSW)
  - Some minor configuration is required.

- **Reported Data**
  - Time → Current Time
  - OIC → Open Input Count (Number of input handles; e.g. reading threads)
  - OOC → Open Output Count (Number of output handles; e.g. writing threads)
  - MxML → Maximum Message Length
  - MEC → Message Enqueue count (Number of messages written)
  - MDC → Message Dequeue count (Number of messages read)
  - UNC → Uncommited messages (at end of monitoring interval)
  - QCD → Current Queue Depth (at end of monitoring interval)
  - MxQD → Maximum Queue Depth (during monitoring interval)
  - GET → Get Enabled/Disabled
  - PUT → Put Enabled/Disabled

SupportPac MH04 – xmqstat (continued)

- **Extended Reported Data (-e option)**
  - PQF → Percentage Queue Full (during monitoring interval)
  - TQF → Time to Queue Full (at present enqueue rate)
  - TQE → Time to Queue Empty (at present dequeue rate)
  - The following extended data requires Queue Monitoring (MonQ) to be turned on
    - QOM → Queue Oldest Message (Age of oldest message in queue)
    - OQTS → Output Queue Time (Short) – Average time messages spent in queue
    - OQTL → Output Queue Time (Long) – Average time messages spent in queue
- **Application Handle Information Reported (-h option)**
  - Data displayed as per DIS QS(*queue*) TYPE(HANDLE)
- **Key Parameters**
  - -d Duration to collect statistics (in Seconds)
  - -e Extended statistics (some require MONQ enabled)
  - -f File name to write statistics to (default is stdout)
  - -h Display information about Application Handles
  - -i Statistics collection interval (in Seconds)
  - -m Queue Manager name
  - -q Queue name
  - -s Suppress display if no activity during interval
  - -t Display time

SupportPac MH04 – xmqstat (continued)

- **Queue Manager Connection Parameters**
  - -v Use the *MQSERVER* environment variable for client connection
  - -c Channel name to use for Client Connection
  - -x ConnectionName (”address(port)”)

- **Invocation Examples**
  - xmqqstat -m *Qmgr* -q *Queue* -d 300 -i 60 -h -e -s -t
    - Connect to local Queue Manager using Server bindings
    - Collect statistics on *Queue* on *Qmgr* (-m and –q)
    - Collect statistics for 5 minutes (-d)
    - Reportstatistics every minute (-i)
    - Display Handle information (-h)
    - Collect extended statistics (-e)
    - Don’t report an interval if there is no activity (-s)
    - Display the time (-t)
  - xmqqstat -c *SYSTEM.DEF.SVRCONN* -x *hostname(1414)* -m *Qmgr* -q *Queue* …
    - Connect to server hostname using port1414 (-x)
    - Use SYSTEM.DEF.SVRCONN channel (-c)

- **Notes**
  - Use Ctrl-C to stop execution

SupportPac MH04 – xmqstat (continued)

```
C:\MQ>xmqqstat -m TEST -q TEST -i 1 -s -t -h
Xmqqstat v1.1 - Developed by Oliver Fisse (IBM)

Connected to queue manager 'TEST'

PLATFORM(WINDOWS NT) LEVEL(701) CCSID(437)
MAXHANDS(256) MAXMSGL(4194304) MAXPRTY(9) MAXUMSGS(250000) MONQ(HIGH)

Processing LOCAL queue 'TEST'

DESC()
CRDATE(2010-09-09) CRTIME(15.29.02) ALTDATE(2010-10-03) ALTTIME(09.14.32)
CLUSTER() CLUSNL() DEFBIND(OPEN)
BOTHRESH(0) BOQNAME()
MONQ(QMGR) USAGE(NORMAL) NOTRIGGER

 Dumping 1 handle(s)...

   PID    TID    AT   CHL/APPL TAG/CONN     USER ID                    B INP I O S
 ------------------------------------------------------------------------------
    7968     0 USER                         Administrator@IBM-6AE723B N  NO N Y N
                ere MQ\java\jre\bin\java.exe

   Time         MxML       MxQD G P    OIC    OUC      MDC      MEC    UNC      CQD
 ------------------------------------------------------------------------------
 10:19:09    4194304    2500000 E E      0      1        0     6300      0     6300
 10:19:10    4194304    2500000 E E      0      1        0      350      0     6650
 10:19:11    4194304    2500000 E E      0      1        0        0      0     6650
 10:19:12    4194304    2500000 E E      0      1        0      350      0     7000
 10:19:14    4194304    2500000 E E      1      1     7000        0      0        0
 10:19:15    4194304    2500000 E E      1      1      350      350      0        0
 10:19:16    4194304    2500000 E E      1      1        0        0      0        0
 10:19:17    4194304    2500000 E E      1      1      350      350      0        0
 10:19:18    4194304    2500000 E E      1      1        0        0      0        0
 10:19:19    4194304    2500000 E E      1      1      350      350      0        0
 10:19:20    4194304    2500000 E E      1      1        0        0      0        0
 10:19:21    4194304    2500000 E E      1      1      350      350      0        0
 10:19:22    4194304    2500000 E E      1      1        0        0      0        0
 10:19:23    4194304    2500000 E E      1      1      303      316      0       16
 10:19:24    4194304    2500000 E E      1      1       47       34      0        0
 10:19:25    4194304    2500000 E E      1      1       18       62      0       40

Control-C caught. Shutting down...

Disconnected from queue manager 'TEST'
Xmqqstat v1.1 ended.
```

Application Benchmarking

- **Far more difficult than Infrastructure testing!**
  - Requires co-ordination with one or more Application teams
    - Communications
    - Scheduling
  - May require data setup and/or cleanup for each test
  - More resource intensive; fewer iterations
- **End-to-End Testing**
  - Different groups and tools collecting data
  - Difficult to correlate all of the different data
  - Frequently, too many cooks in the kitchen!
  - Not very useful for fine grained analysis and tuning
  - However, essential to benchmark application throughput and latency
- **Latency versus Capacity**
  - Latency → The round trip time of a single transaction
  - Capacity → The number of transaction per period of time (Seconds, Minutes, Days, etc)
- **IBM Performance Reports**
  - Don't forget to compare your results against IBMs!

Key Performance Indicators (KPIs)

- **API Puts - Calls/Second; Bytes/Second**
  - First set of tests with no threads reading messages.
  - Second set of tests with threads reading messages.
    - Keep queue depth close to or equal to zero.
  - Single Thread – 1K message size.
  - Multiple Threads – 1K message size.
  - Run Tests with Large Messages (e.g. 10 MB).
    - Keep an eye on disk space utilization.
  - Evaluate horizontal scaling (e.g. adding threads).
  - Keep an eye on queue depths and disk space usage.
  - Clean up messages after the test.
- **API Gets - Calls/Second; Bytes/Second**
  - Single Thread – 1K message size.
  - Multiple Threads – 1K message size.
  - Run Tests with Large Messages (e.g. 10 MB).
- **Message Channels**
  - Messages/Second; Bytes/Second
  - Add additional channels and transmission queues to test scaling options

MQ Performance Tuning

Summary

Next Steps

- **MoreTuning**
  - Iterative process
  - Infrastructure Tuning (Channels) & Application Tuning (Architecture, Design, and Programming)
  - Requires considerable WMQ knowledge
    - Application Design and Programming
    - WMQ Internal Processing
    - WMQ Data Gathering (measurements) & Testing tools
  - Results are sometimes counter-intuitive
- **Capacity Planning**
  - With benchmarks in place, overall system capacity can be estimated
  - Will capacity meet business requirements and/or SLAs?
  - Will capacity handle peak loads?
- **Monitoring**
  - Now that you know how it will break, monitor to determine when it will break!
  - Proactive upgrades before the Production outage takes place.

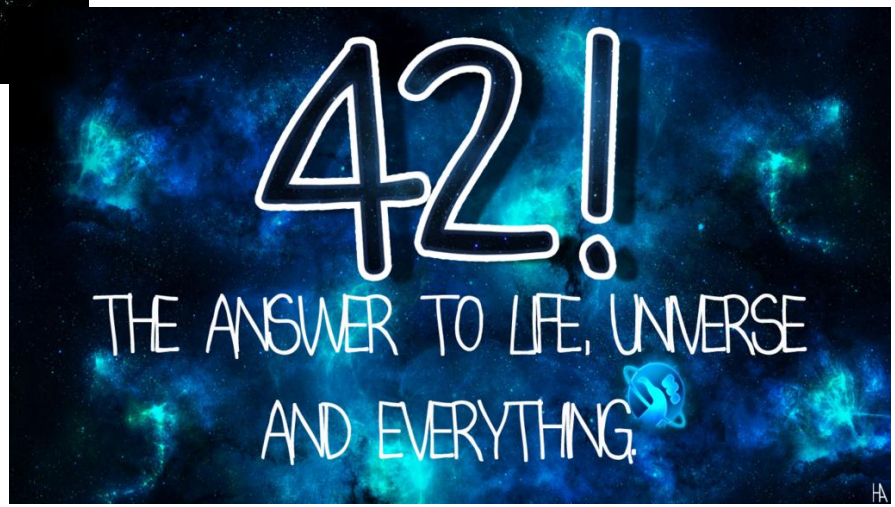
Key Takeaways

- **WMQ processing is asynchronous**
  - Some application processes are “Writers”
  - Some application processes are “Readers”
  - Speed of writers unrelated to speed of readers
- **Applications Scale Horizontally**
  - Applications increase capacity by adding additional instances
    - Application Instances (e.g. Application Servers)
    - Application threads
- **When capacity of “Readers” exceeds capacity of “Writers”**
  - Performance is at maximum throughput
  - Messages are processed in memory
  - Queue Depths are at or near zero
- **When capacity of “Writers” exceeds capacity of “Readers”**
  - Performance is at minimum throughput
  - Messages are processed from disk
  - Queue Depths are increasing

Reference Material

- IBM Developer Works
  - Tuning for Performance
    - http://www.ibm.com/developerworks/websphere/library/techarticles/0712_dunn/0712_dunn.htmlText
- IBM SupportPacs
  - http://www-01.ibm.com/support/docview.wss?rs=977&uid=swg27007205
  - Performance Reports (MPxx)
    - READ THESE!!! They have lots of information NOT FOUND ELSEWHERE!
    - http://www-01.ibm.com/support/docview.wss?rs=171&uid=swg27007150&loc=en_US&cs=utf-8&lang=en
  - MA01 SupportPac (“q”)
    - http://www-01.ibm.com/support/docview.wss?rs=171&uid=swg24000647&loc=en_US&cs=utf-8&lang=en
  - MH04 SupportPac (“xmqstat”)
    - http://www-01.ibm.com/support/docview.wss?rs=171&q1=Xa02&uid=swg24025857%20&loc=en_US&cs=utf-8&lang=en

Deep Thoughts



Questions & Answers

Presenter

- Glen Brumbaugh
  - Glen.Brumbaugh@TxMQ.com
- Computer Science Background
  - Lecturer in Computer Science, University of California, Berkeley
  - Adjunct Professor in Information Systems, Golden Gate University, San Francisco
- WebSphere MQ Background (20 years plus)
  - IBM Business Enterprise Solutions Team (BEST)
    - Initial support for MQSeries v1.0
    - Trained and mentored by Hursley MQSeries staff
  - IBM U.S. Messaging Solutions Lead, GTS
  - Platforms Supported
    - MVS aka z/OS
    - UNIX (AIX, Linux, Sun OS, Sun Solaris, HP-UX)
    - Windows
    - iSeries (i5OS)
  - Programming Languages
    - C, COBOL, Java (JNI, WMQ for Java, WMQ for JMS)


Thank
You