



Introduction to MQ: Can MQ Really Make My Life Easier?

Chris Leonard

IBM UK

ChrisL@uk.ibm.com

Session 17885

Monday 10th August 2015



SHARE is an independent volunteer-run information technology association that provides **education, professional networking and industry influence.**

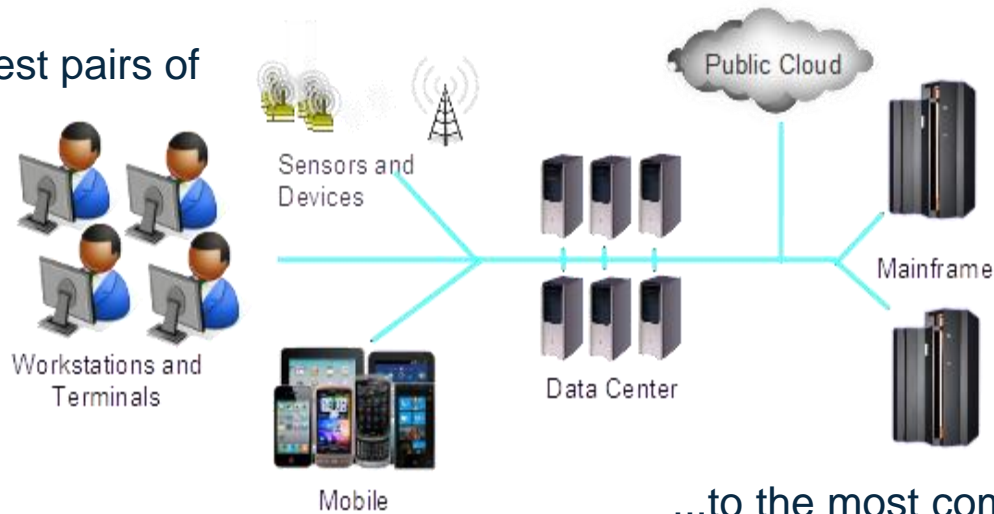


Agenda

- Why use messaging?
- Fundamentals of MQ
- Using the MQ API
- Other key features
- Extensions and related products
- Putting it all together...
- Summary

Why use messaging...?

From the simplest pairs of applications...

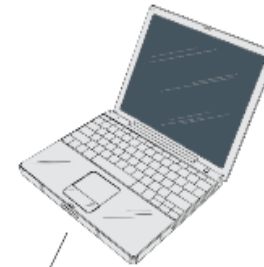
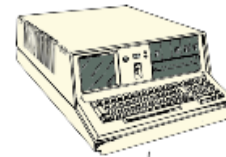


...to the most complex business processes.

- Messaging simplifies the challenges of connecting systems:
 - Extended Reach – Connecting anywhere to anywhere
 - Reliability – Assured delivery of data, securely and performant
 - Flexibility – Ease of application change
 - Scalability – Incremental growth of applications and capacity

Extended Reach – Universal Connectivity

Payroll have a program to run to add a one-time payment to an employee's pay packet



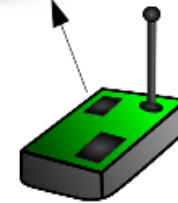
Research have a program to calculate annual review scores



Engineers monitor problem reports



Warehouse sensors monitor temperature and humidity

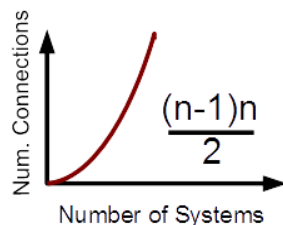
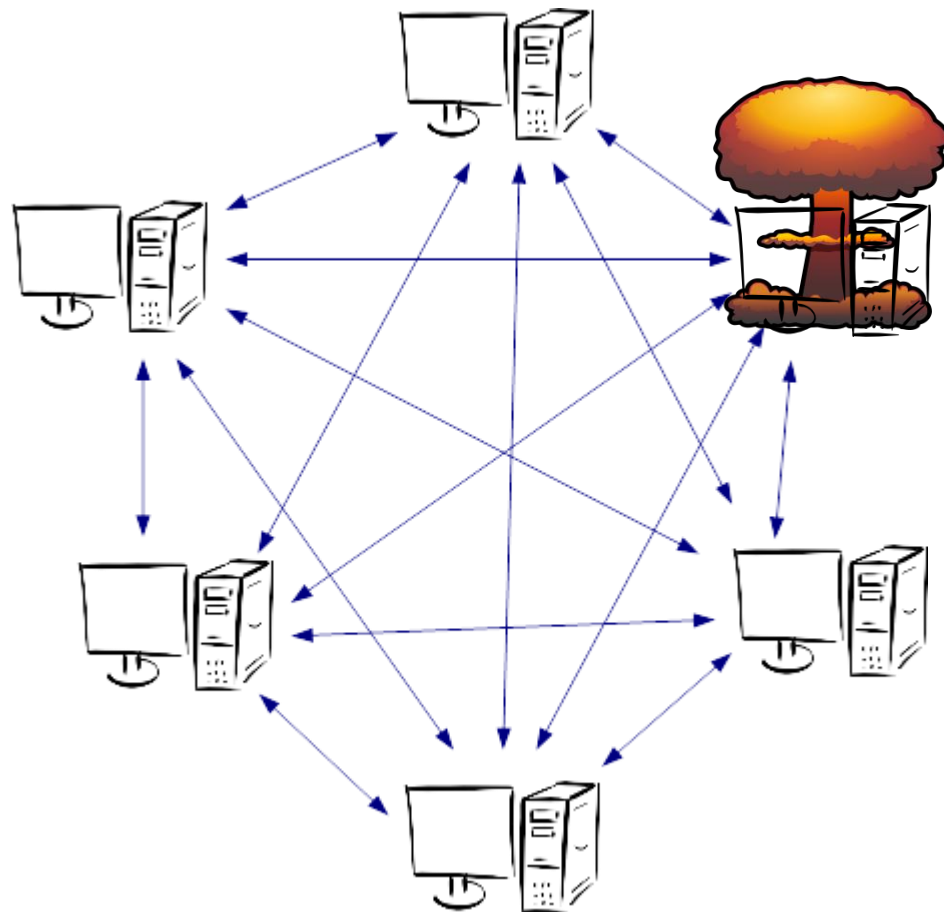


We want to connect applications running on different hardware and operating systems, and written in different programming languages.

Complete your session evaluations online at www.SHARE.org/Orlando-Eval

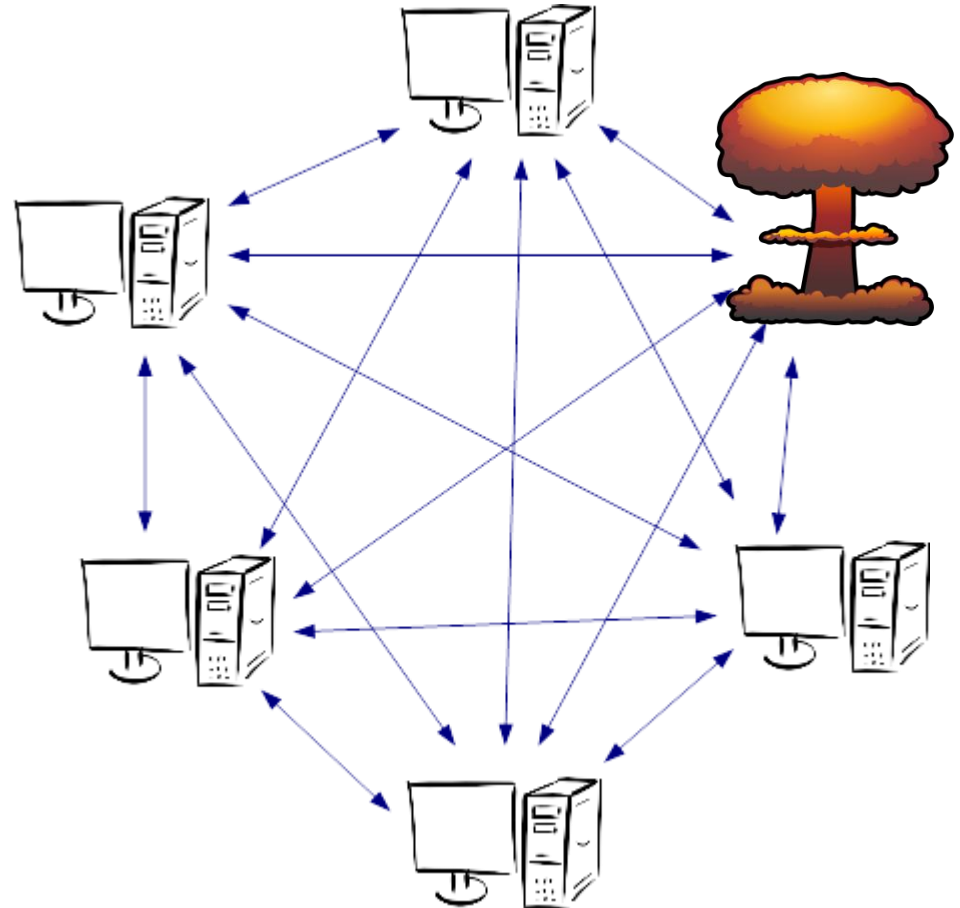
Reliability

- As systems become more tightly coupled, their reliance on each other increases.
 - The cost of a failure of a process increases
- Maximum number of connections goes up with the square of the number of systems



Reliability

- The risk of failure can be reduced by:
 - Removing dependencies
 - Introducing redundancy
 - Assuring data delivery
 - Providing robust security



Flexibility

- A process was originally designed for one purpose...
- ... It then needed to change to meet new requirements
- Being able to respond rapidly to internal and external challenges by rapidly modifying existing services gives a competitive advantage.



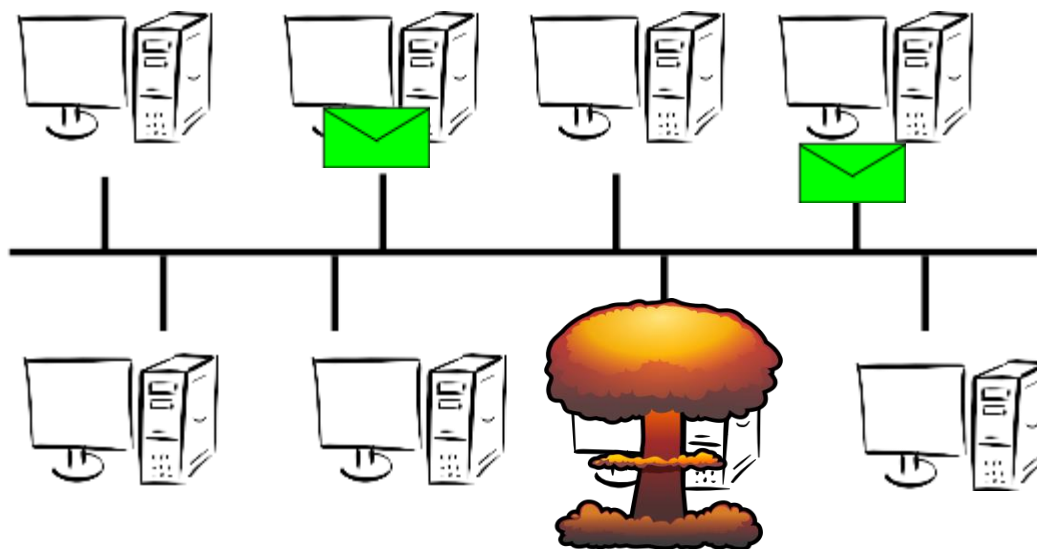
Process Scalability

- Many applications and processes start out on a single system.
- The business grows, and the capacity of the system can no longer cope with the workload demand.
- A scalable architecture enables the capacity to be incrementally grown to meet increasing workloads



Decoupling Systems

These issues can be overcome by choosing an alternative communication architecture:



The interdependence between systems can be decoupled through the use of a common messaging system, providing a scalable environment which is more tolerant of individual system outage.

Complete your session evaluations online at www.SHARE.org/Orlando-Eval

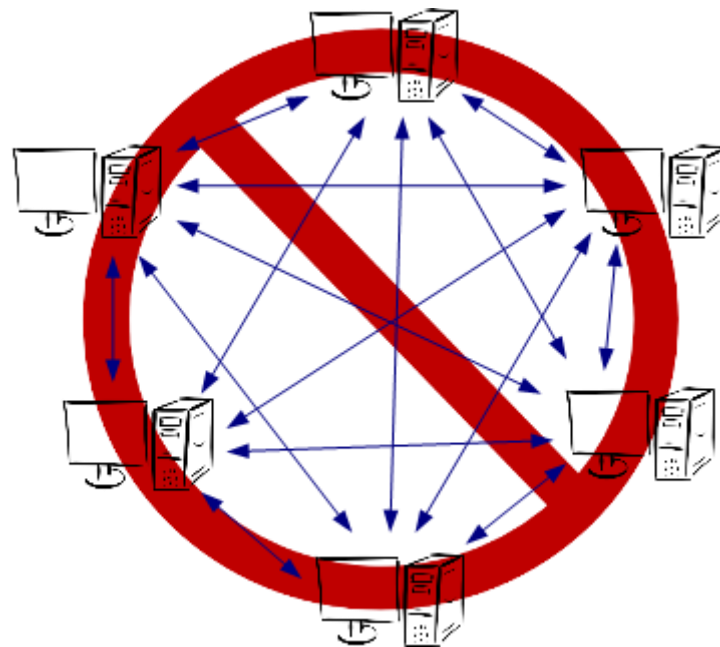
Agenda

- Why use messaging?
- **Fundamentals of MQ**
- Using the MQ API
- Other key features
- Extensions and related products
- Putting it all together...
- Summary

Fundamentals of MQ

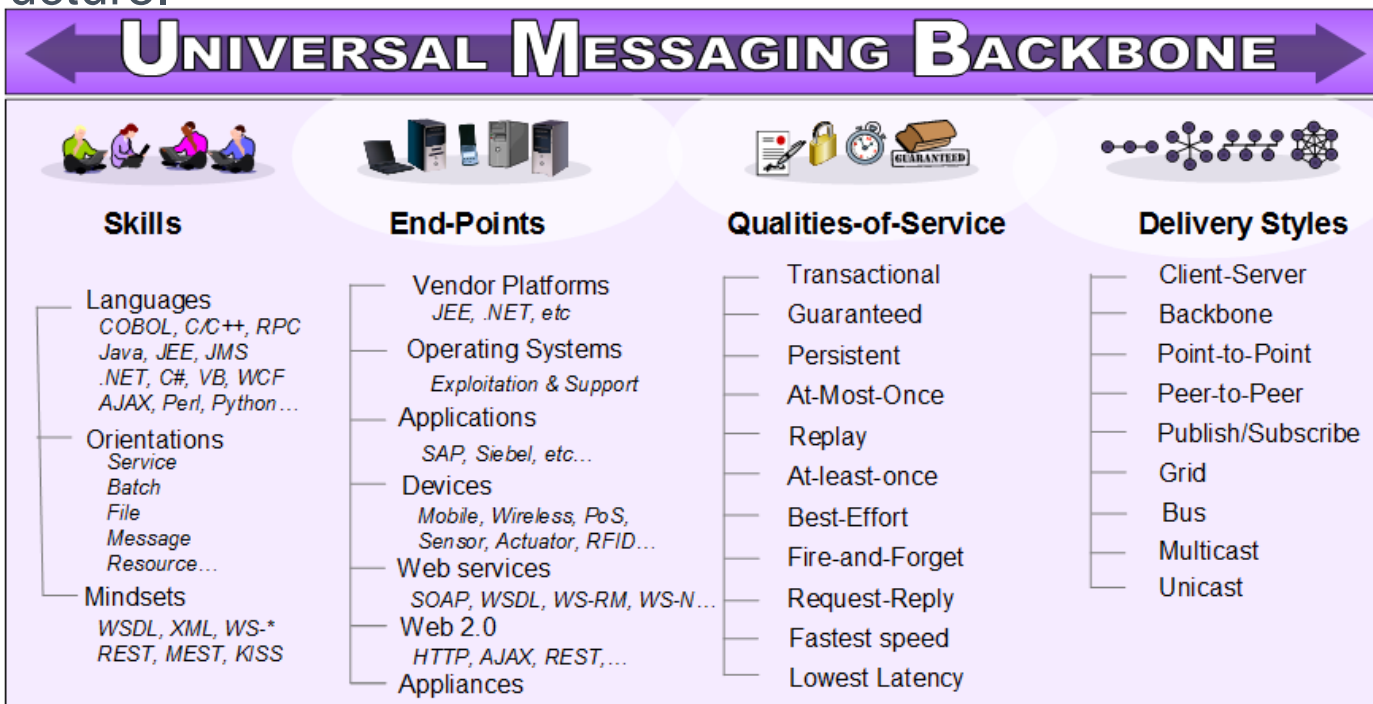
- **Reliability**
 - Assured message delivery
 - Performance
- **Ubiquitous**
 - Breadth of support for platforms, programming languages and API
- **Loose application coupling**
 - Location transparency
 - Time independence
 - Data transparency (with IBM Integration Bus)
 - Platform independence
- **Scalability**
 - Incremental growth

- **Rapid development**
 - Standards
 - Reduce Complexity
 - Ease of use



The Vision – The Universal Messaging Backbone

The vision for MQ is that it provides a range of capabilities, making it suitable to be a transport backbone across all environments in an IT Infrastructure.



MQ does not provide all these capabilities today. It evolves with new technologies as they develop and become widely adopted.

MQ is not a substitute for:

- Well written applications
- Robust network
- Good operational procedures
- Well managed systems

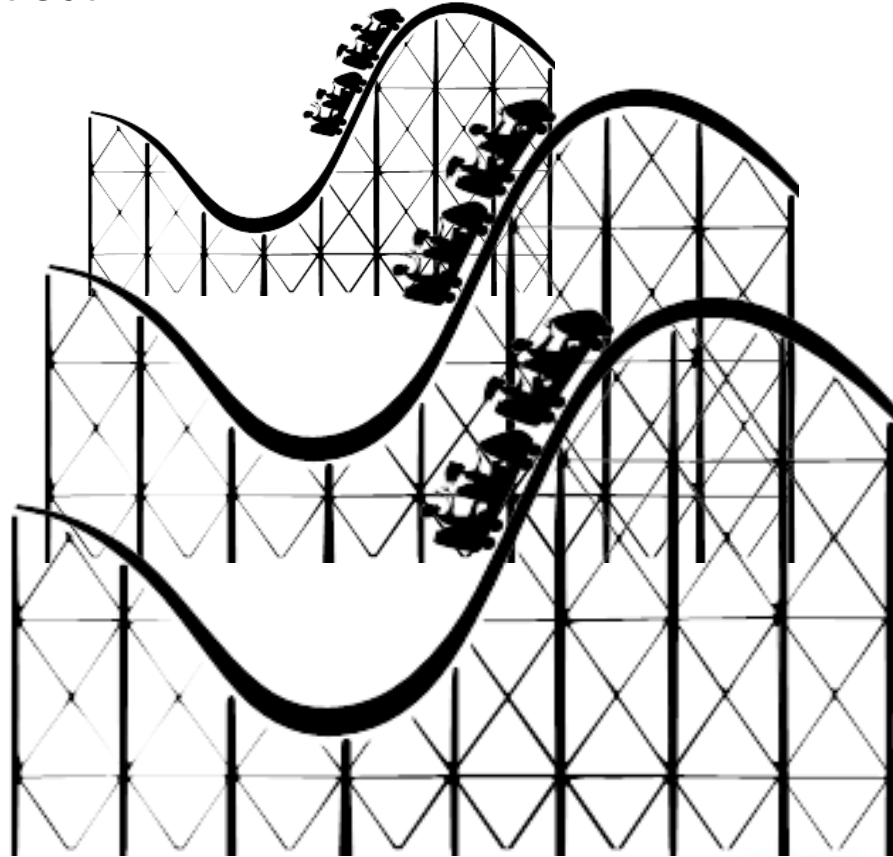


What is Asynchronous Messaging?

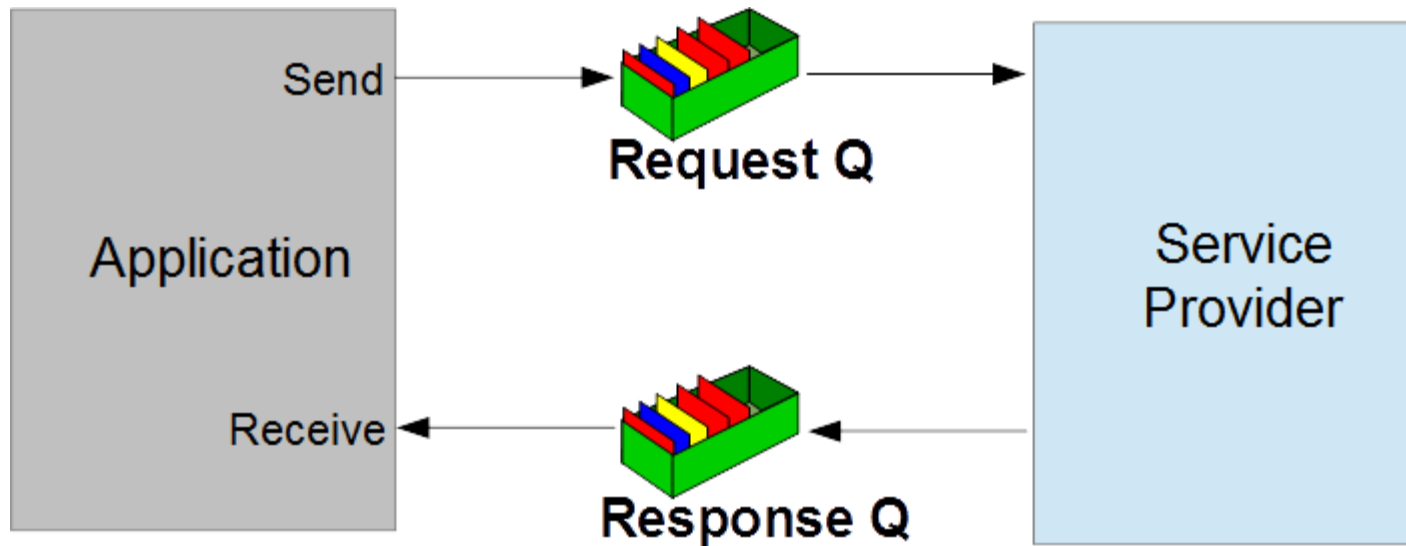
- Paradigm 1: Point to Point
- Paradigm 2: Publish Subscribe

Messaging Paradigm 1: Point to Point Messaging

- FIFO – First In, First Out
- One object in, one object out



Asynchronous Messaging – Point to Point



- Messages can be created from many sources:
 - Data, Messages, Events, Files, Web service requests / responses

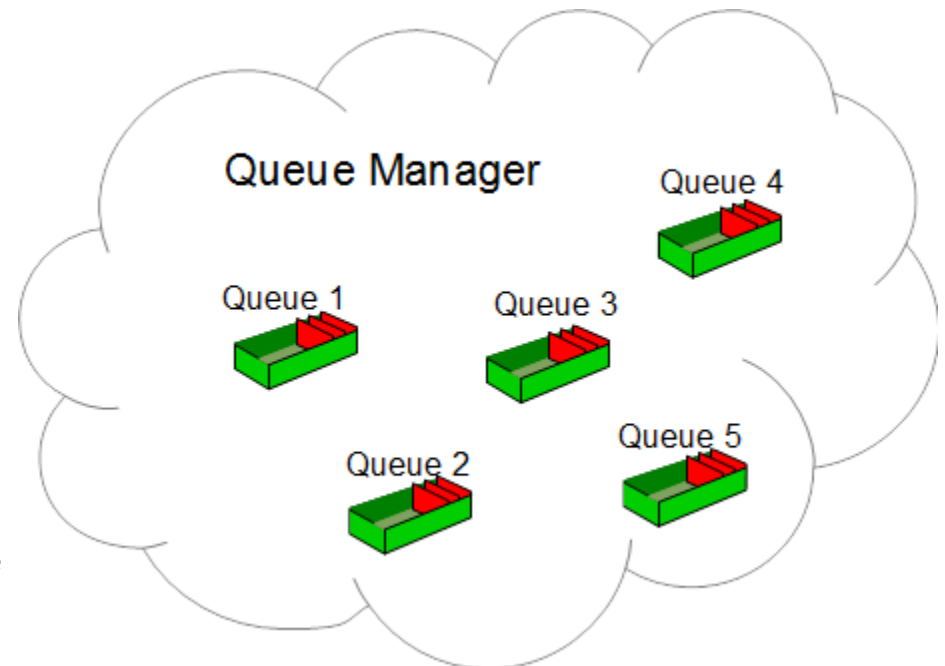
MQ – Point to Point Messaging

- The physical world is frequently organised in queues. Consider for a moment just how many queues you have been involved in today alone. We queue at the Post Office, Supermarket checkout, at traffic lights. We write shopping lists and to do lists. We use the postal service, voice mail, and of course, the ever present e-mail.
- The truth is that queuing is a natural model that allows us to function efficiently. Perhaps not surprisingly therefore it turns out that it is also a very useful model in which to organise our applications.
- Instead of application A talking synchronously to Application B have Application A 'send a message' to a queue which Application B will read.
- Messages can be of any form, the content is not restricted, so they could contain:
 - - General data
 - - Data packaged as messages
 - - It might be notification events
 - - Files being moved in a Managed File Transfer application
 - - SOAP messages for invoking services

What is a Queue?

- **A queue holds messages**
 - Various Queue Types
 - Local, Alias, Remote, Model
- **Queue creation**
 - Predefined
 - Dynamically defined
- **Message Access**
 - FIFO
 - Priority
 - Direct
 - Selected by Property (V7+)
 - Destructive & non-destructive access
 - Transacted

- **Parallel access by applications**
 - Managed by the queue manager

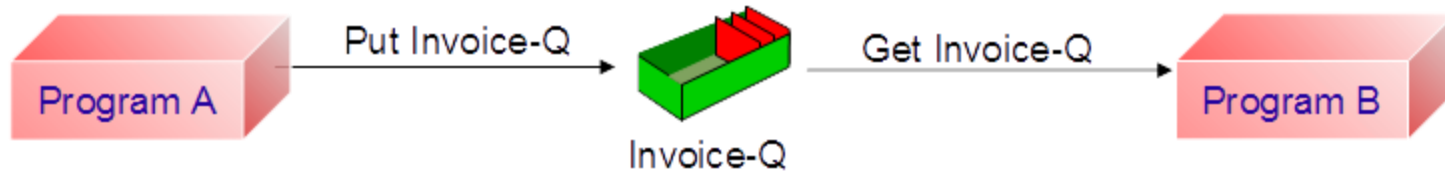


What is a Queue?

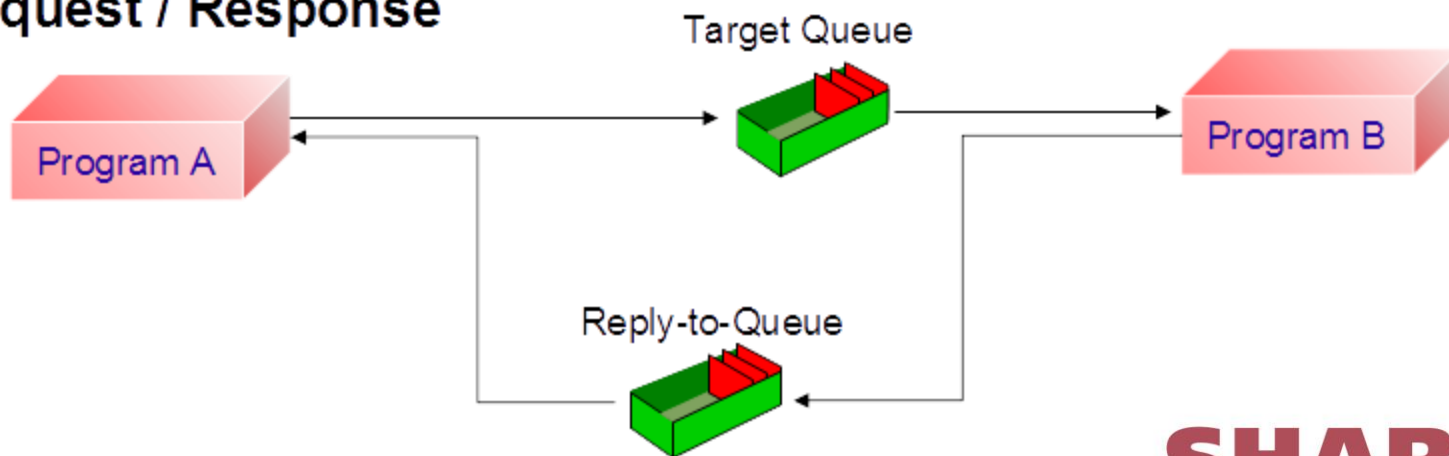
- A Queue is a named object (up to 48 characters) which is defined with a queue type.
 - Local Only queue type which can actually hold messages
 - Alias A queue name which 'points' to another queue
 - Remote A queue which 'points' to a queue on a remote machine
 - Model A template definition which when opened will create a local queue with a new name
-
- Applications open queues, by name, and can either put or get messages to/from the queue. Messages can be got from the queue either in FIFO order, by priority or directly using a message identifier or correlation identifier.
 - As many applications as required can open a queue either for putting or for getting making it easy to have single server responding to many clients or even n servers responding to many clients.
 - A key feature of MQ is its transaction support. Messages can be both put and got from queues in a transaction. The transaction can be just local, involving just messaging operations or global involving other resource managers such as a database. A classic example, is an application which gets a message, updates a database and sends a reply message all within a single transaction. If there is a failure before the transaction commits, for example a machine crash, both the database update and the received message will be rolled back. On machine restart the request message will still be on the queue allowing the application to reprocess the request.

Example application architectures (1)

'Send and Forget'



Request / Response

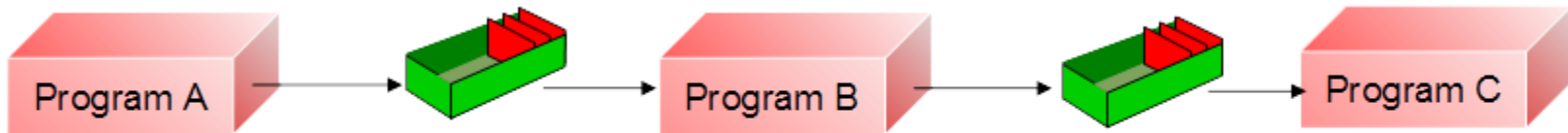


Example application architectures (1)

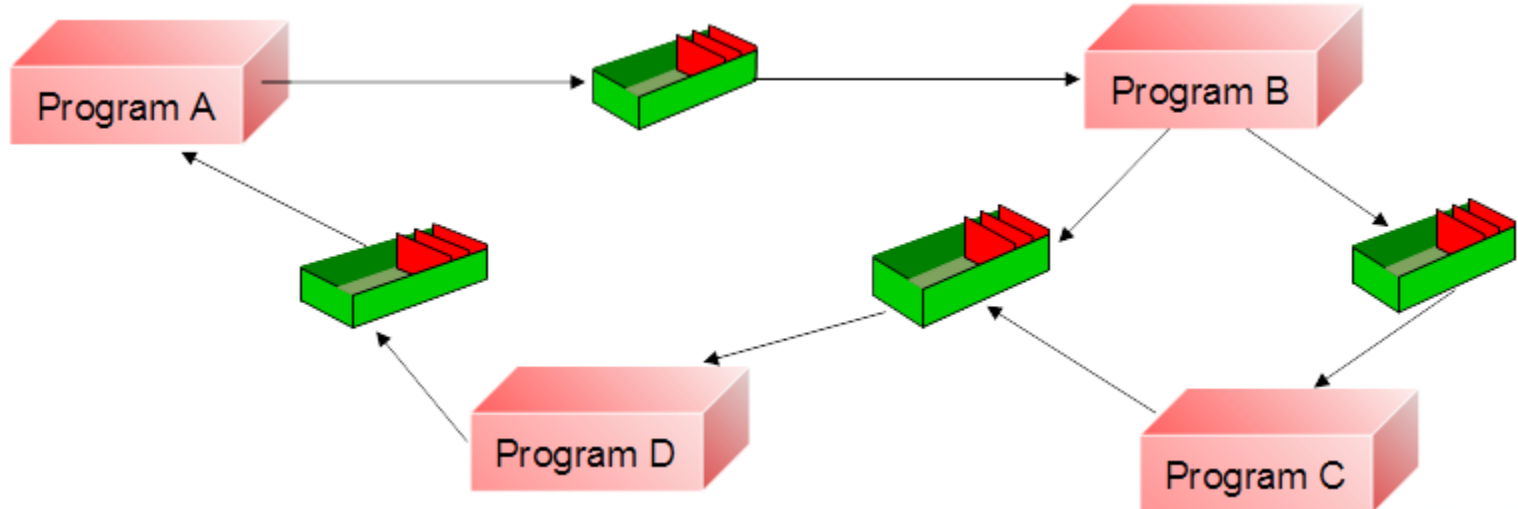
- These examples show some of the ways in which MQ queues can be used and, thereby, shows some of the styles of applications that may benefit from the use of a message/queuing model.
- 'Send and Forget'
- This style is one where there is no (direct) response required to a message. The message/queuing layer will guarantee the arrival of the data without the application having to solicit a response from the receiver.
- Request/Response
- This style is typical of many existing synchronous applications where some response is required to the data sent. This style of operation works just as well in an asynchronous environment as in a synchronous one. One difference is that - in this case - the sender does not have to wait for a response immediately. It could pick up the response at some later time in its processing. Although this is also possible with the synchronous style, it is less common.

Example application architectures (2)

Chain



Workflow



Example application architectures (2)

- **Chain**

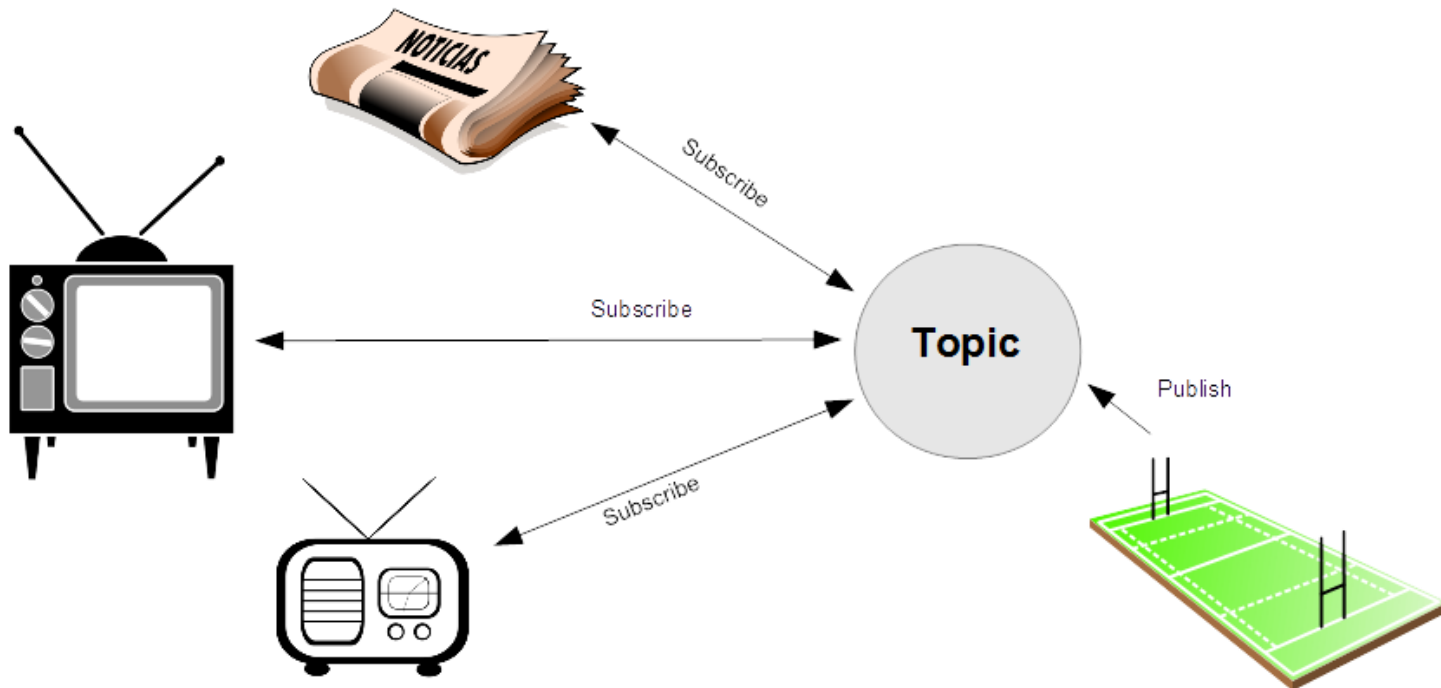
Data does not have to be returned to the originating application. It may be appropriate to pass a response to some other application for processing, as illustrated in a chain of applications.

- **Workflow**

There may be multiple applications involved in the processing before a response comes back to the originating application. These various modes of interaction may be arbitrarily combined to provide as complex/sophisticated topology as is necessary to support a particular application. The loosely coupled nature of the message queuing model makes it ideal for this style of interaction. Furthermore, it makes it straightforward to develop applications in an iterative style.

Messaging Paradigm 2: Publish / Subscribe

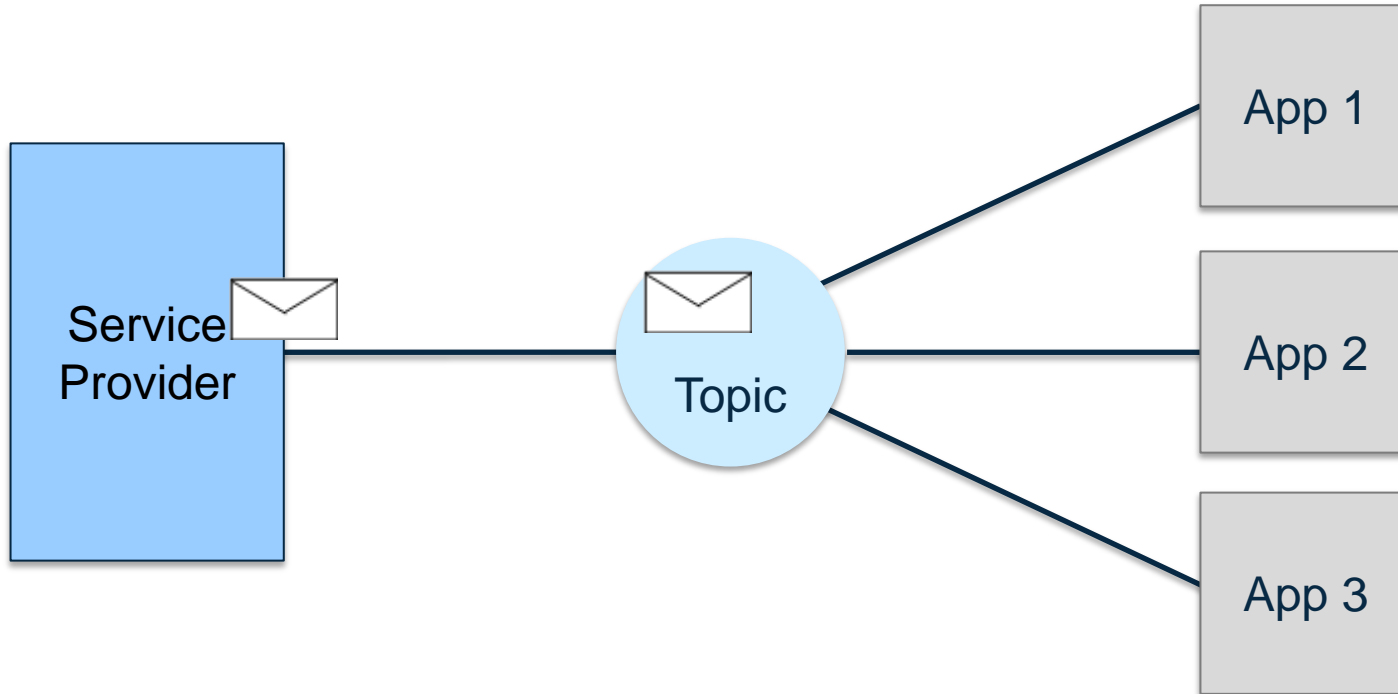
- One message is published, several messages are produced, one for each subscriber.
 - One to many relationship



MQ – Messaging Topology 2

- Our daily life is full of examples of requests for information on a given topic and providing information about a given topic.
- Let us consider an example:-
- You have installed a piece of software on your computer and you would like to know when there are updates available for it. The software provider has a service to inform you of updates when they occur.
- You ask the software provider to let you know when there are updates available for the software (a topic) in which you are interested (a subscription)
- The software provider informs you when updates become available (a publication)
- The software provider can use a message to provide this information
- You can provide a destination (queue) to which the information is published

Asynchronous Messaging – Publish/Subscribe



What is a Topic?

- A Topic is defined by a “**Topic String**”. This is a case sensitive character string, where the following characters have a special meaning:
 - '/' The topic level separator – provides structure to topic trees
 - '#' The wildcard character
 - '+' The single-level wildcard character
- Example:
Price/Fruit/Apple
- The Topic can be defined in a number of ways:
 - Predefined by the MQSC command
 - Predefined by the PCF interface (as used by the MQ Explorer)
 - Subscribing or Publishing to the Topic object

Topic Trees

By arranging Topic strings in a tree hierarchy, a 'Topic Tree' is created. Every node in the tree is a Topic.

Topic Trees provide two benefits:

- Wildcard characters can be used to subscribe to multiple Topics.
- Security policies can be established

For example, to subscribe to both Topics:

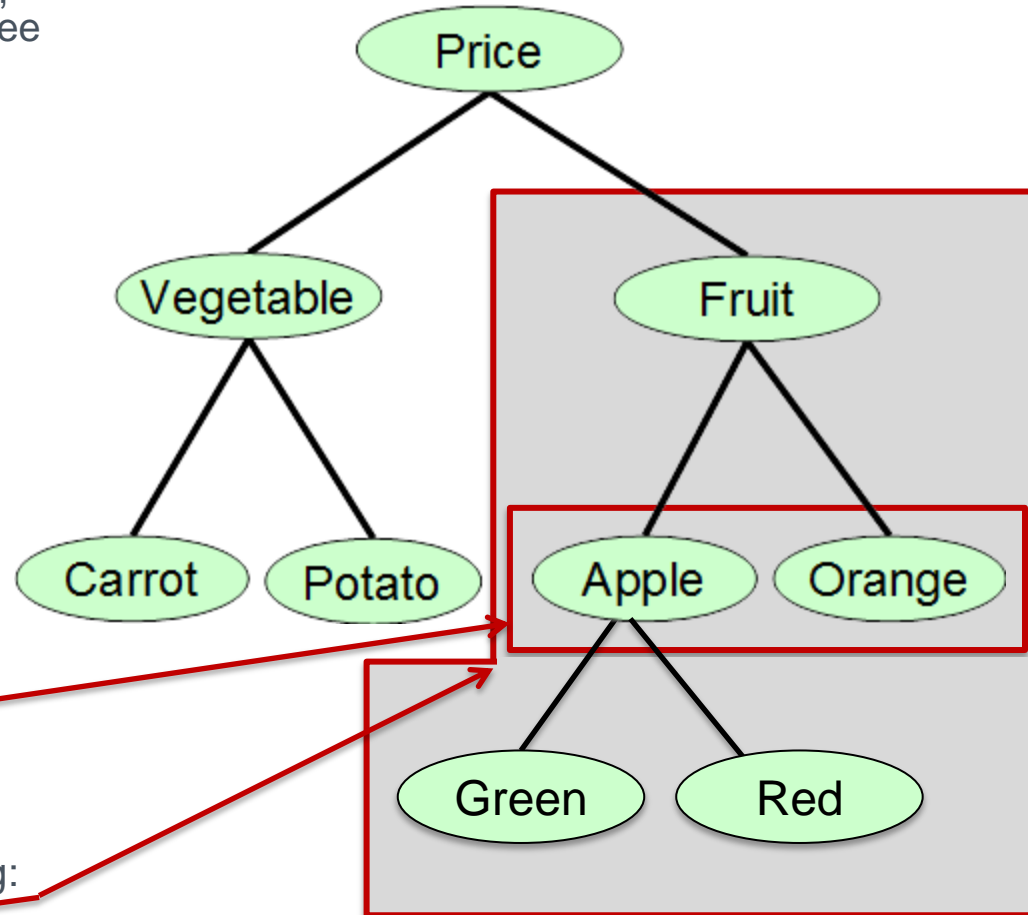
Price/Fruit/Apple
Price/Fruit/Orange

The subscription string is:

Price/Fruit/+

Note this is different to the subscription string:

Price/Fruit/#

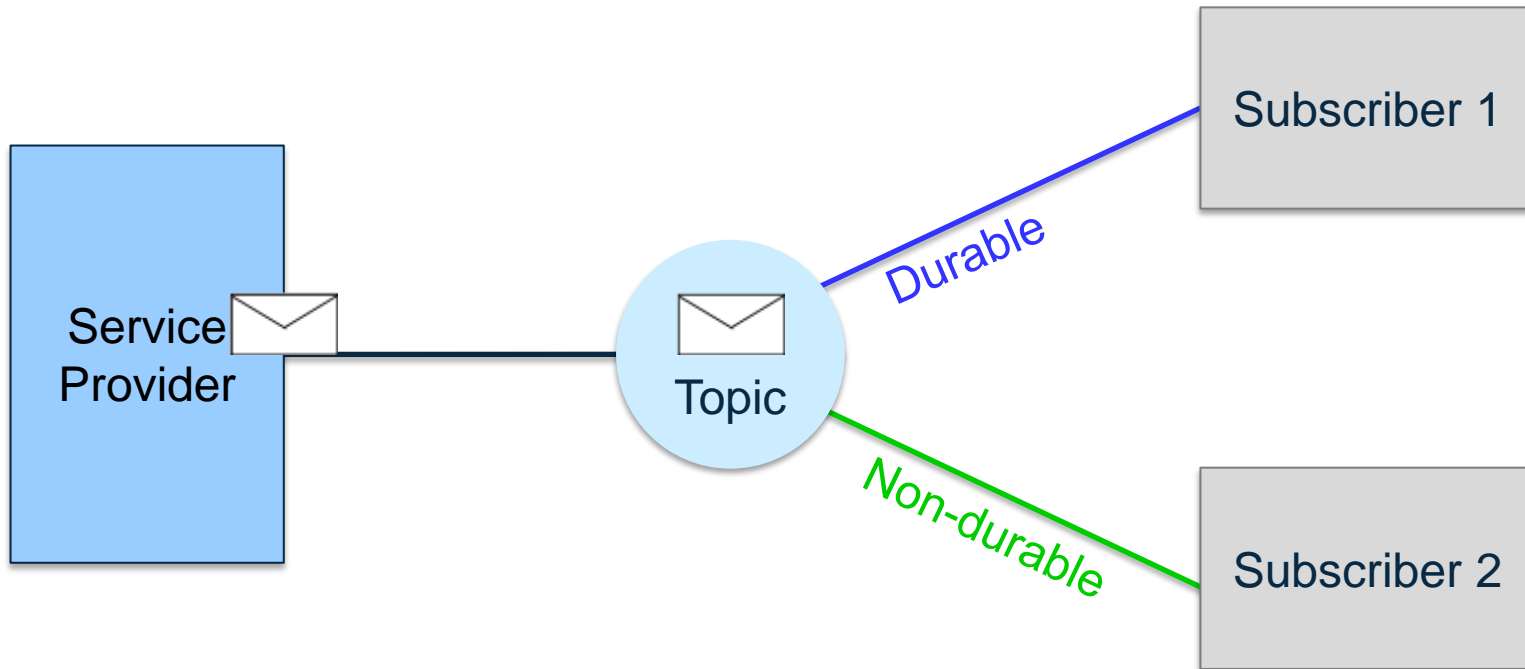


What is a Topic?

- A topic is a character string that describes the nature of the data that is published in a publish/subscribe system. Topics are key to the successful delivery of messages in a publish/subscribe system. Instead of including a specific destination address in each message, a publisher assigns a topic to the message. The queue manager matches the topic with a list of subscribers who have subscribed to that topic, and delivers the message to each of those subscribers.
- Note that a publisher can control which subscribers can receive a publication by choosing carefully the topic that is specified in the message.
- Each topic that you define is an element, or node, in the topic tree. The topic tree can either be empty to start with or contain topics that have been defined by a system administrator using MQSC or PCF commands. You can define a new topic either by using these create topic commands or by specifying the topic for the first time in a publication or subscription.
- Although you can use any character string to define a topic's topic string, choose a topic string that fits into a hierarchical tree structure. Thoughtful design of topic strings and topic trees can help you with the following operations:
 - Subscribing to multiple topics.
 - Establishing security policies.
 - Although you can construct a topic tree as a flat, linear structure, it is better to build a topic tree in a hierarchical structure with one or more root topics.
 - Topics can be defined by a system administrator using MQSC or PCF commands. (Topic objects)
 - However, the topic of a message does not have to be defined before a publisher can use it; a topic is created when it is specified in a publication or subscription for the first time.

Durable Publish/Subscribe in action

Durable subscriptions results in published messages being retained when the subscriber is not connected.

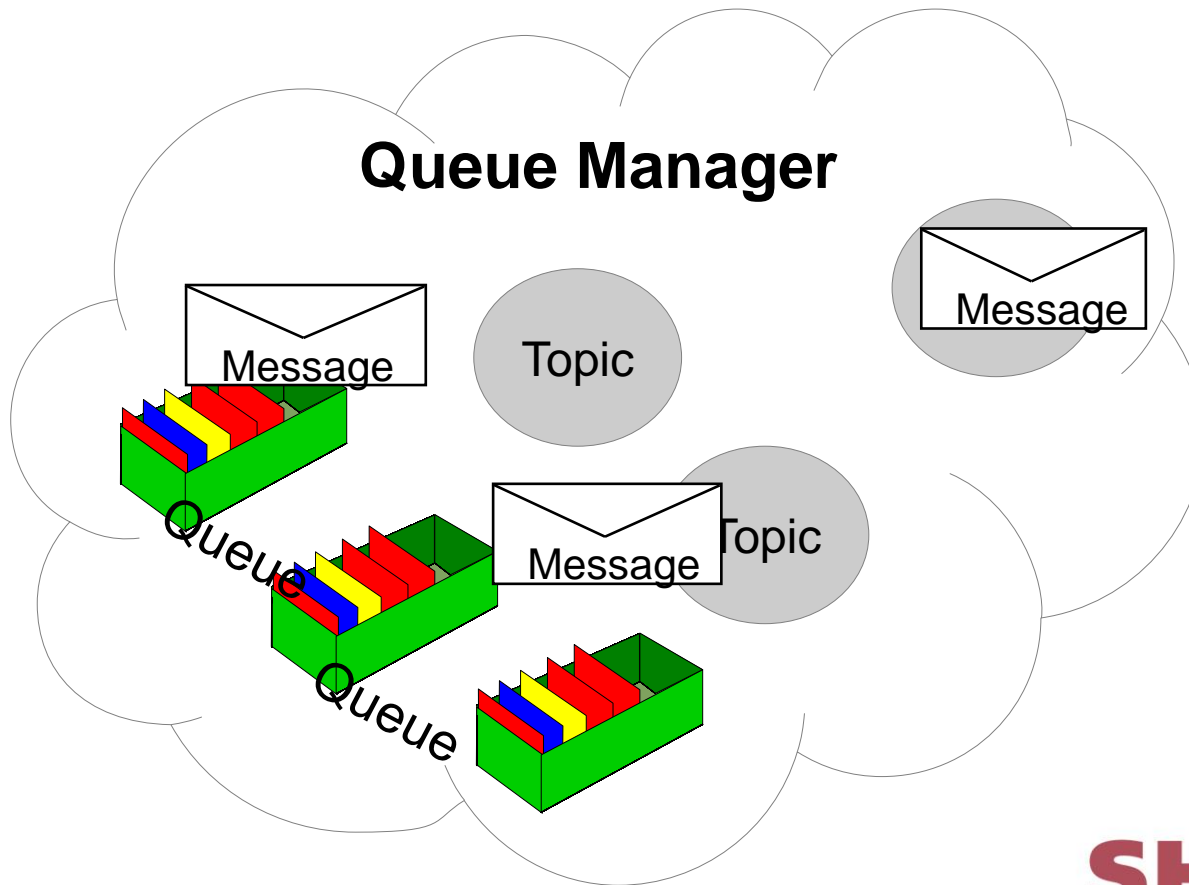


What is Publish/Subscribe?

- In this environment, the receiving applications notify an intermediary of their interest in particular sets of information. The receiving (or subscribing) application provides a subject and a queue where messages matching this subject may be delivered.
- The sending (publishing) applications generate information, together with a subject name, and sends the information to the pub/sub engine. The pub/sub engine contains a matching service which determines the subscribing applications interested in receiving this information.
- Note that the publish/subscribe model provides for the situation where a message may be published by an application using a subject which has no subscribers. In this instance the message data is discarded.
- There are many publish/subscribe products available in the marketplace today. MQ publish/subscribe differentiates itself by providing support for the publish/subscribe model and combining it with the exactly once delivery model of MQ message/queuing.

The MQ Queue Manager

The Queue Manager is the process which controls the storage and flow of messages



What is a message?

- Message = Header + User Properties + User Data



A Series of Message Attributes
Understood and augmented by the Queue Manager

- Message Id
- Correlation Id
- Routing information
- Reply routing information
- Message priority
- Message codepage/encoding
- Message format
-etc.

- Any sequence of bytes
- Private to the sending and receiving programs
- Not meaningful to the Queue Manager

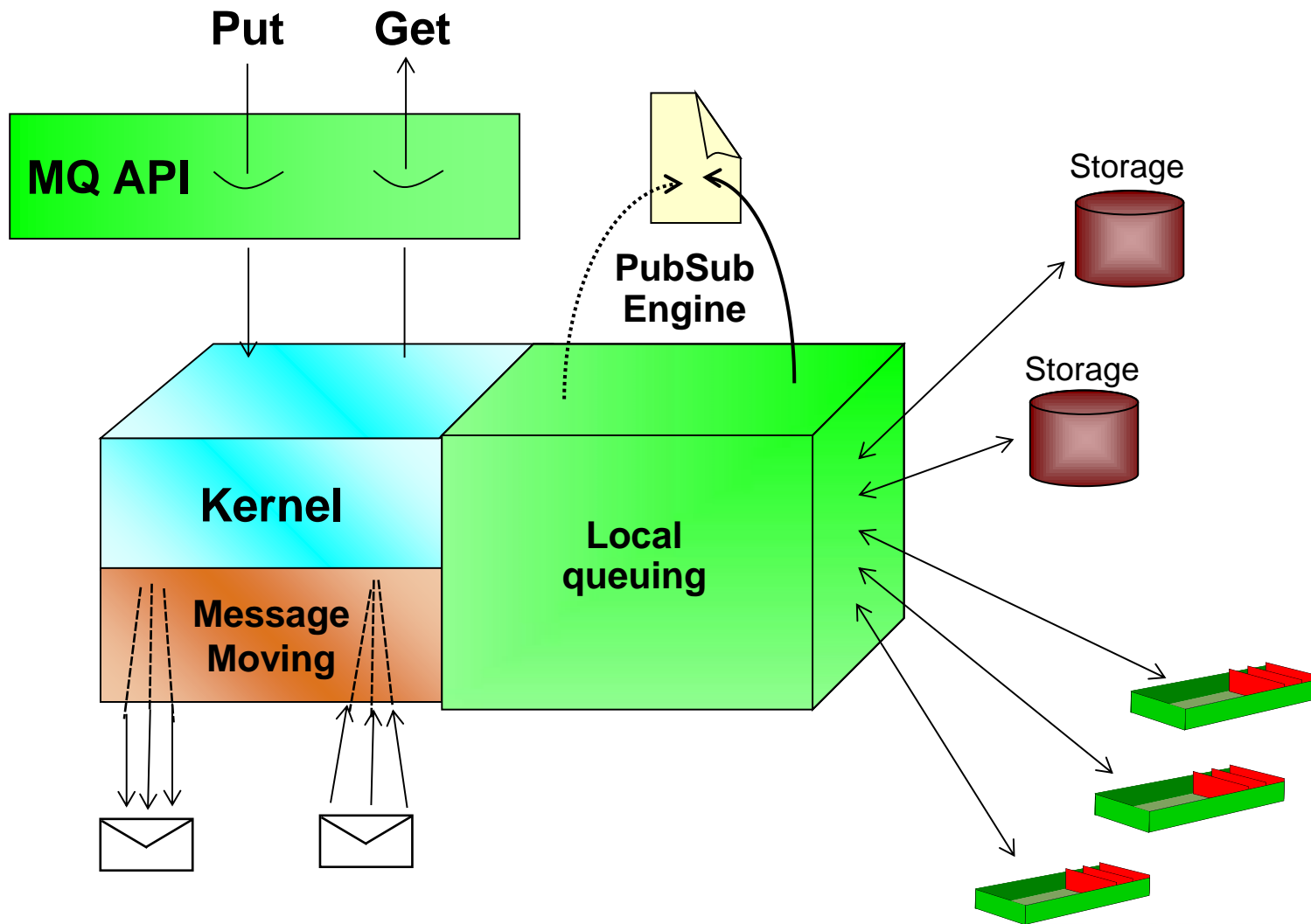
- User Properties require WMQ V7
Emulated for JMS in older versions of WMQ
- Arbitrary properties
For example, this is a "green" message

- Message Types
 - Persistent ... recoverable
 - Non Persistent
- Up to 100MB message length

What is a Message?

- A message in MQ is merely a sequence of bytes in a buffer of a given length. The current products support up to 100MB in a single message although the vast majority of messages are in the order of a few thousand bytes.
- Messages have various attributes associated with them such as their identifier, their priority and their format. Each application is free to define its own format for messages although there are a number of predefined formats. One common format for messages is XML for example.
- A key attribute of a message is its persistence. A message is either persistent or non-persistent. This attribute tells the Queue Manager how important the message is.
- Persistent: persistent messages are written to disk and are logged. The Queue Manager will ensure that the messages are recovered in the case of a system crash or network failure. These messages are delivered once and only once to the receiving applications.
- Non-persistent: The messages are identified by the application as non-critical. The Queue Manager will make every effort to deliver these messages but since they are not necessarily written to disk they will be lost in the case of a system crash or network failure. Clearly with no disk IO involved these messages are much faster than persistent ones.

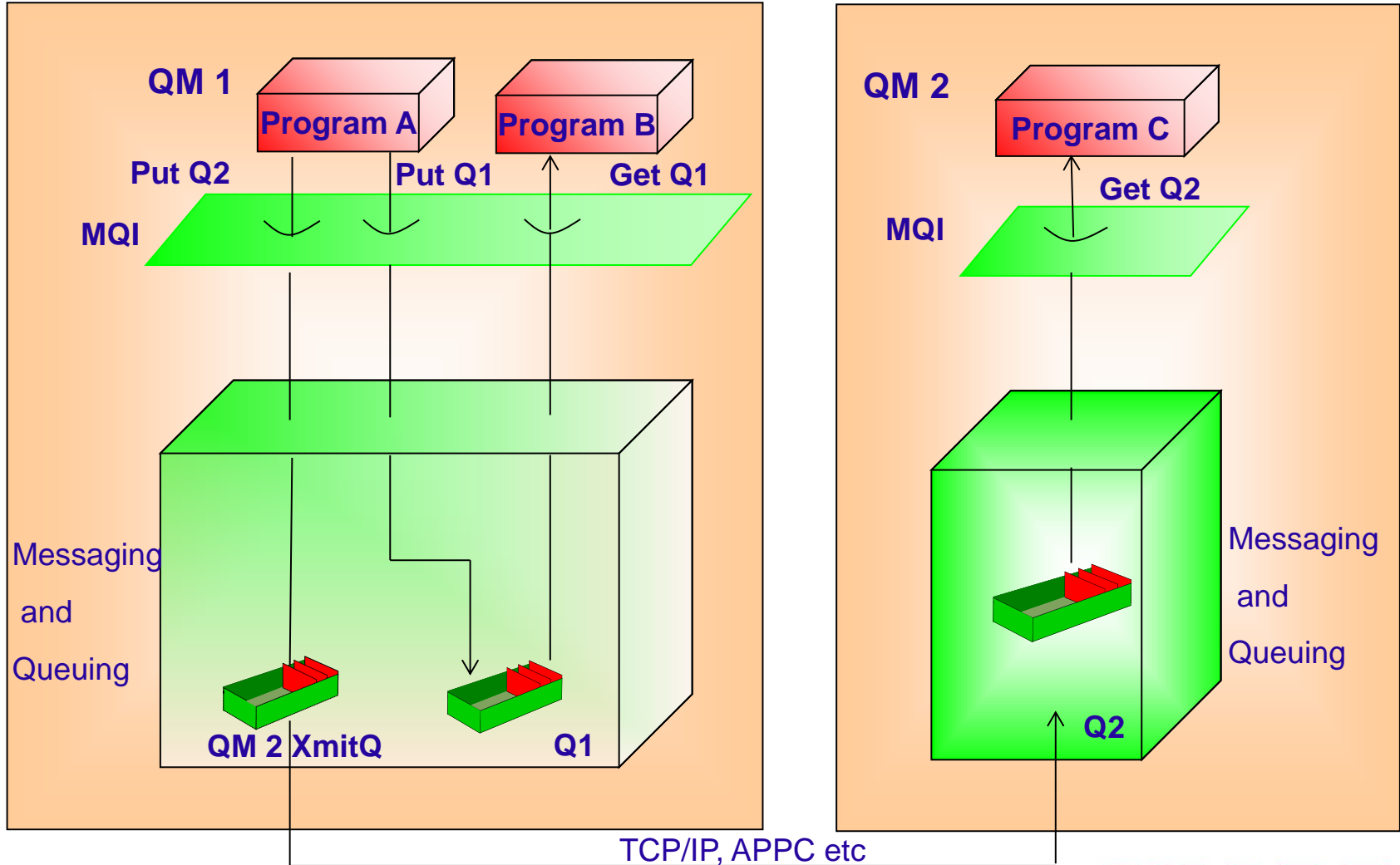
The Queue Manager



What is a Queue Manager?

- A queue manager may - generally - be thought of as 3 components:
- The Kernel is the part of the queue manager that understands how to implement the MQ APIs. Given that the APIs are common across the queue manager family, it stands to reason that the Kernel is mostly common code across the set of queue managers. (The primary exception to this is the z/OS queue manager where the same functions are implemented differently to support the same APIs).
- The Local Queuing component is the part of the queue manager responsible for interacting with the local operating system. It manages memory, the file system and any operating system primitives such as timers, signals, etc. This component insulates the Kernel from any considerations of how the underlying operating system provides services and so enables the Kernel to be operating system independent.
- The Message Moving component is responsible for interacting with other queue managers and with MQI clients. For environments where all of the message queuing activity is local to a system then this component is unused - though this is a very rare case.
- The message moving functions are provided by specialised MQ applications, called Message Channel Agents

Local and Cross-System Communication with MQ

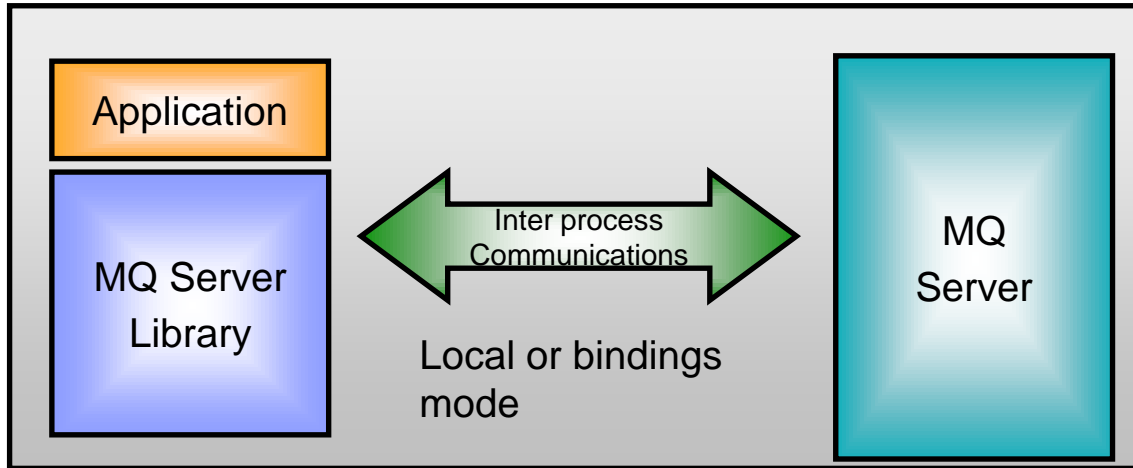


Cross-System Communication with MQ

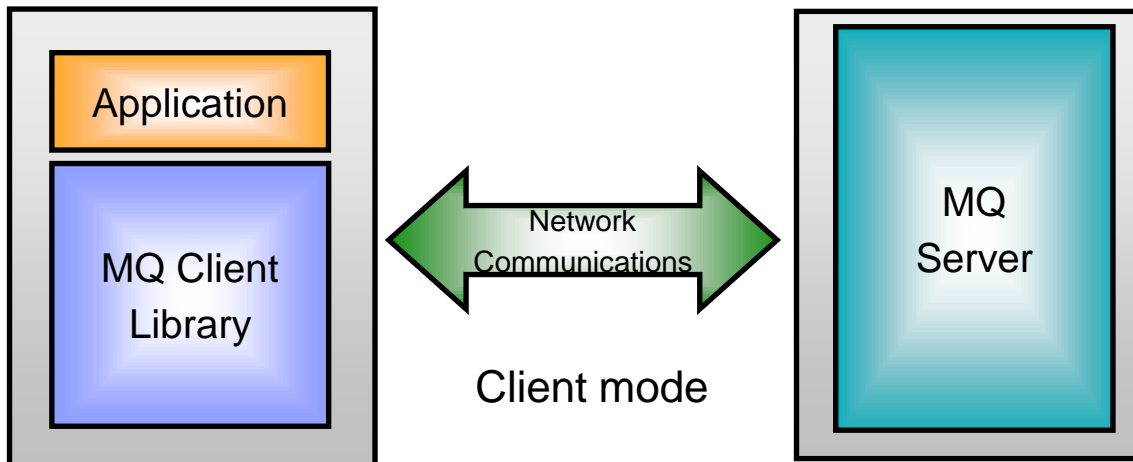
- In the diagram we see Program A sending messages to two other programs.
- To Program B: in this case the actual physical queue that both applications access are the same. This therefore does not require any network communication.
- To Program C: in this case Program A wants to put a message to queue Q2 on Queue Manager QM2. It can't do this directly without requiring that the network and QM2 Queue Managers are available so instead the message is put to a 'holding' queue called a transmission queue. Asynchronously, another part of MQ called a channel will read this transmission queue and deliver any messages to the queues on QM2.
- Any number of applications running on QM1 can send messages to QM2 via the same transmission queue and channel.

Communicating with the Queue Manager

Server Model



Client Model



Application code is independent of the client to queue manager connection mode

Complete your session evaluations online at www.SHARE.org/Orlando-Eval

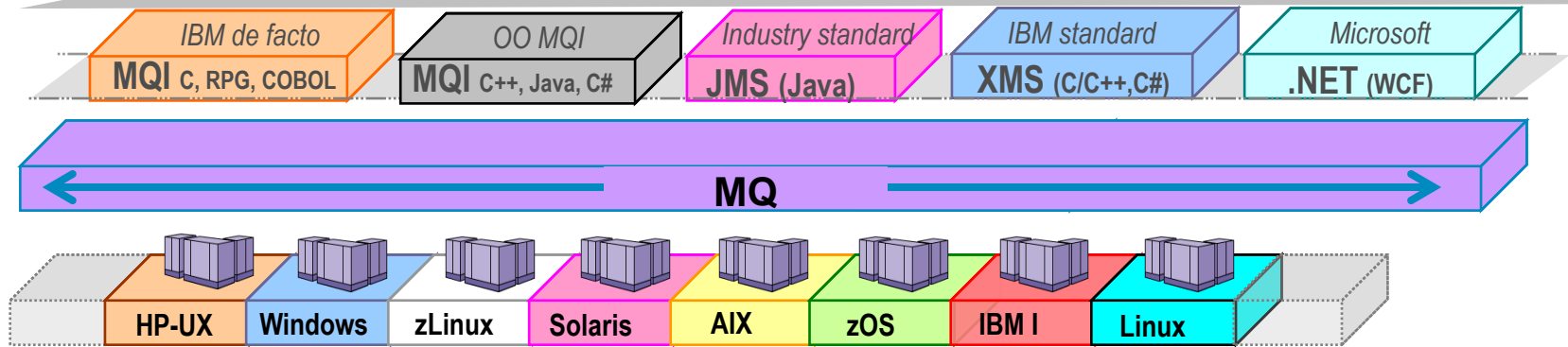
What is an MQ Client?

- MQ clients provide a low cost, low resource mechanism to gain access to MQ facilities. The client provides a remote API facility, allowing an MQ application to run on a machine that does not run a queue manager.
- Each MQ API command is passed to a Server queue manager where a proxy executes the required API command. The connection between client and server is entirely synchronous providing an 'rpc-like' mechanism - though NO regular (well-known) rpc mechanism is used !
- The client machine does not own any MQ resources - all resources are held by the Server machine. Thus, if local queuing capability is required then a server (rather than a client) must be used.
- The MQ Client support is part of the MQ product that can be installed and used separately from the MQ server. It provides a set of libraries which can be linked with your applications to provide access to MQ queues without requiring the application to run on the same machine as the queues.
- Generally speaking an application is linked either with the client libraries or with the server libraries (often called 'local' or 'bindings' mode). In bindings mode the application communicates with the Queue Manager via an inter-process communications link of some kind. In client mode the application communicates via a network connection. However, as can be seen from the diagram, the two models are logically equivalent. For this reason the functionality provided at the client is almost identical to that provided by local applications.

Agenda

- Why use messaging?
- Fundamentals of MQ
- **Using the MQ API**
- Other key features
- Extensions and related products
- Putting it all together...
- Summary

Programming API



- Broad support for:
 - Programming languages, messaging interfaces, application environments and OS platforms.

Programming API

- One of MQ's key strengths is its breadth. It can run on virtually any commercially available platform and is accessible through a wide number of programming languages and API. The MQ Interface (MQI) is the de facto API for MQ, providing simple common access across all platforms. It has both a procedural implementation and an object oriented implementation.
- Standards based interfaces such as JMS, and its IBM equivalent for C, C++ and .NET, XMS are also available.

The MQ API (MQI)

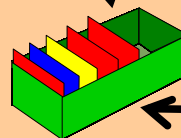
Sending Application

1. MQCONN (QM1)
2. MQOPEN (APP.Q for PUT)
3. MQPUT (MSG1)
4. MQCLOSE
5. MQDISC



Queue Manager

QM1



APP.Q

Receiving Application

1. MQCONN (QM1)
2. MQOPEN (APP.Q for GET)
3. MQGET
4. MQCLOSE
5. MQDISC

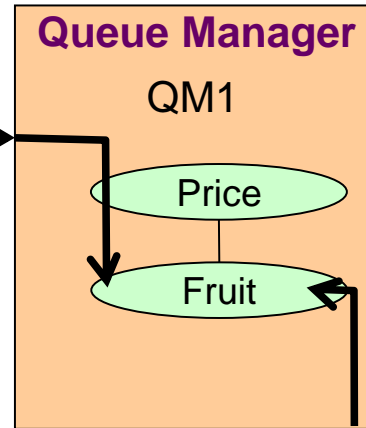
The MQ API (MQI)

- The most common verbs are MQOPEN, MQCLOSE, MQPUT and MQGET which are concerned with the processing of messages on queues. The first example shows an application putting a message to a queue and another getting the message off the queue. We refer to this as the Point-to-Point application model. The second example shows an application publishing a message to a topic and another subscribing to messages about that topic. We refer to this as the Publish/Subscribe application model.
- There are many, many options associated with these verbs. However, in general, most of these options may be left to take their default values - and MQ provides a set of default structures to allow for easy assignment of these default values.
- There are 24 verbs in total in the MQ API, known as the MQI. We have briefly illustrated the most common ones. The rest have less frequent use and we have summarised them in a table.
- To use the MQ verbs in your application you link with the MQ library provided with MQ, which will send your call to the MQ queue manager to process.

The MQ API – Publish/Subscribe

Sending Application

1. MQCONN (QM1)
2. MQOPEN (“Price/Fruit”)
3. MQPUT (MSG1)
4. MQCLOSE
5. MQDISC



Receiving Application

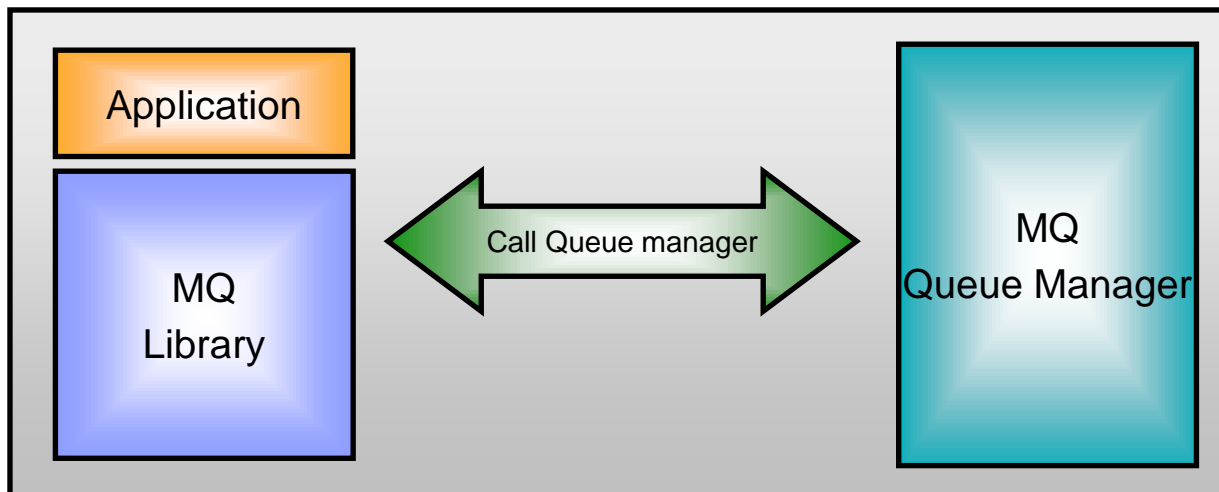
1. MQCONN (QM1)
2. MQSUB (“Price/Fruit”)
3. MQGET
4. MQCLOSE
5. MQDISC

The MQ API (MQI) – Summary of all verbs

IBM de facto

MQI C, RPG, COBOL

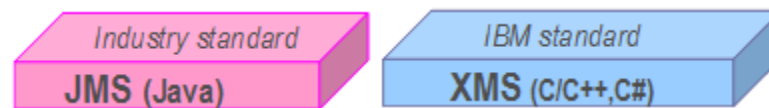
Connection	Resource Use	Messages	Object attributes	Transactions	Message Properties
MQCONN	MQOPEN	MQPUT	MQINQ	MQBEGIN	MQCRTMH
MQCONNX	MQSUB	MQPUT1	MQSET	MQCMIT	MQDLTMH
MQCTL	MQSUBRQ	MQGET		MQBACK	MQSETMP
MQDISC	MQCLOSE	MQCB			MQINQMP
					MQDLTMP
					MQMHBUF/MQBUFMH



Complete your session evaluations online at www.SHARE.org/Orlando-Eval

Java Message Service (JMS) and XMS

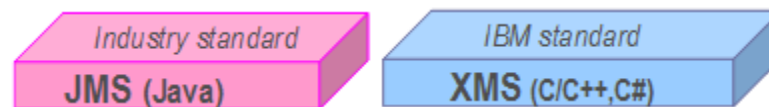
- JMS is the standard Java API for messaging
 - Point-to-point and Publish/subscribe messaging
 - Enables greater portability between messaging providers
 - Vendor-independent messaging API in Java
 - Managed by The Java Community Process
 - Expert Group includes IBM
 - MQ supports all Java Enterprise Edition (JEE) 1.4+ application servers
 - Features such as message-driven beans greatly simplify creation of messaging applications



Complete your session evaluations online at www.SHARE.org/Orlando-Eval

Java Message Service (JMS) and XMS

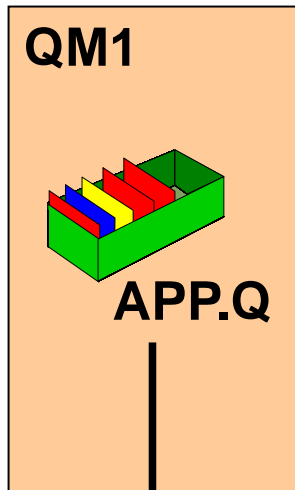
- IBM Message Service Clients (XMS) renders a JMS-like API in non-Java languages
 - (Almost) full compatibility with JMS 1.1 API
 - Full interoperability with IBM JMS implementations on MQ and WPM
 - Shared administered objects in JNDI with JMS
 - Current implementations include: C, C++ and .NET



Complete your session evaluations online at www.SHARE.org/Orlando-Eval

Example JMS receiving application

- Some of the client APIs need no MQI programming knowledge!



```
// Lookup the MQ specific objects in JNDI
Context jndiContext = new InitialContext();
ConnectionFactory cf =
    (ConnectionFactory) jndiContext.lookup("jms/QM1");
Destination dest =
    (Destination) jndiContext.lookup("jms/APP.Q");

// Establish a connection with the queue manager
Connection conn = cf.createConnection();
conn.start();
Session session =
    conn.createSession(false, Session.AUTO_ACKNOWLEDGE);

// Get a message
MessageConsumer consumer = session.createConsumer(dest);
Message msg = consumer.receive();
```

Java Message Service (JMS) and XMS

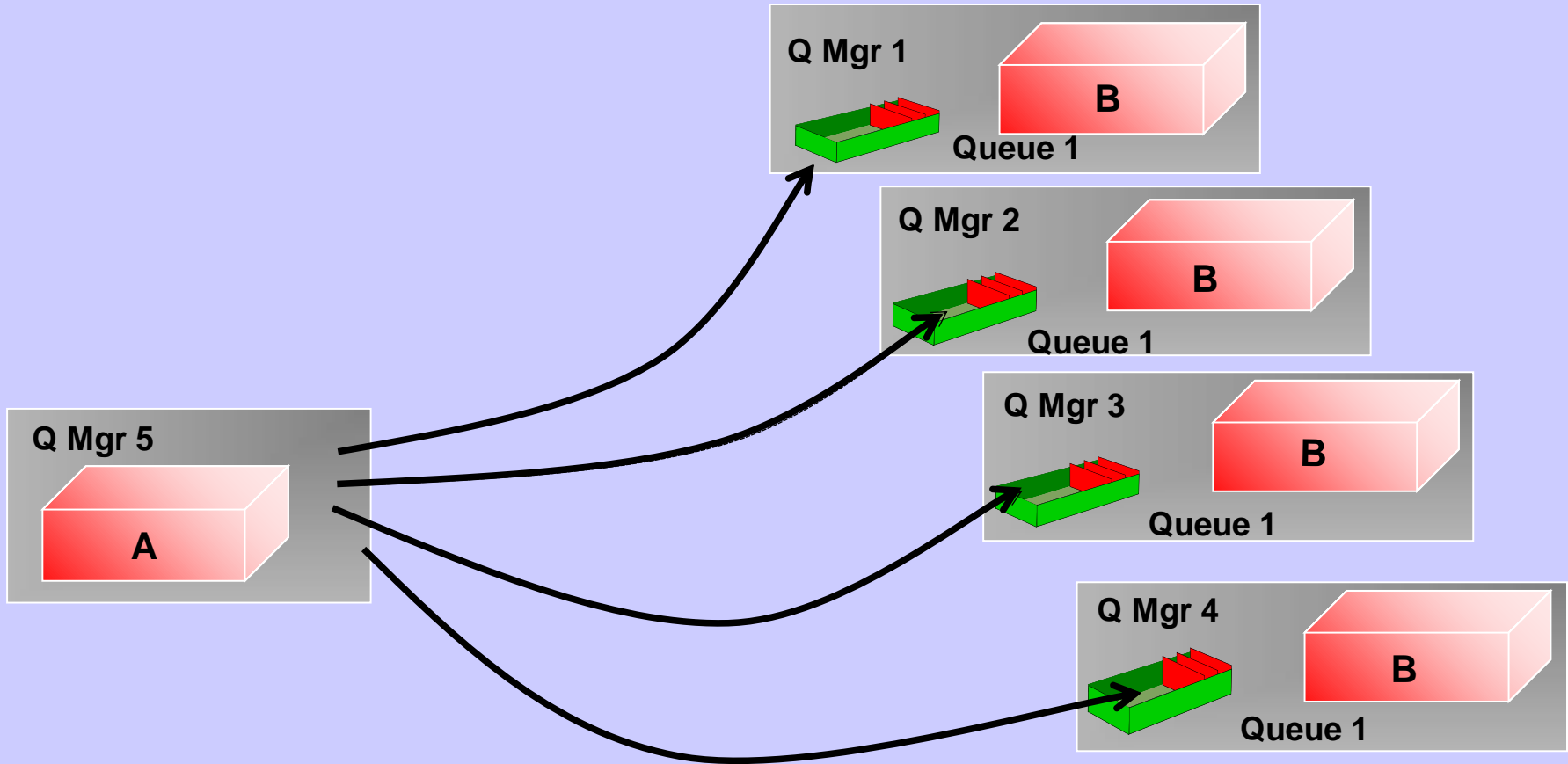
- JMS is part of the J2EE specification and is supported by all J2EE compliant applications servers including; WAS, WebLogic etc. If you are working in Java or a J2EE environment inside an app. server, then you will almost certainly use JMS to access your messaging infrastructure. JMS 1.1 is the current version of the standard and is fully supported by MQ.
- It simplifies programming – providing simple to use Pub/Sub messaging in addition to point-to-point, although there are many similarities with the MQI (Connection = MQCONN(), Session = UOW)
- XMS syntactically the same as JMS V1.1, but for C, C++ and C#. It offers good interoperability between JMS & non-Java applications, and they share administration models – it is ideal for sending message to JMS application running in an Application Server.

Agenda

- Why use messaging?
- Fundamentals of MQ
- Using the MQ API
- **Other key features**
- Extensions and related products
- Putting it all together...
- Summary

Example application architectures - Clustering

Cluster A



Example application architectures – Clustering

- In order to enable highly scalable applications, MQ queue managers provide support for MQ Clusters. In this environment, there are several copies (or clones) of a particular target queue and each message is sent to exactly one of the possible choices.
- MQ Cluster support also defines and manages all MQ resources, such as channels, automatically and provides automatic notification of failed or new queue managers in the environment.

MQ Transactions

- Message level inclusion/exclusion in unit of work
- Single UoW active per connection at any one time
- MQ local units of work
 - MQCMIT and MQBACK control the unit of work
- Messages and other resources in a global unit of work
 - Managed by a Transaction Manager
 - *WebSphere Application Server, CICS, IMS, z/OS RRS*
 - *Microsoft Transaction Server*
 - *Any XA or JEE App Server Transaction Manager*
- Managed by MQ
 - *MQ is an XA Transaction Manager*
 - *MQBEGIN, MQCMIT and MQBACK control the unit of work*

IBM de facto

MQI C, RPG, COBOL

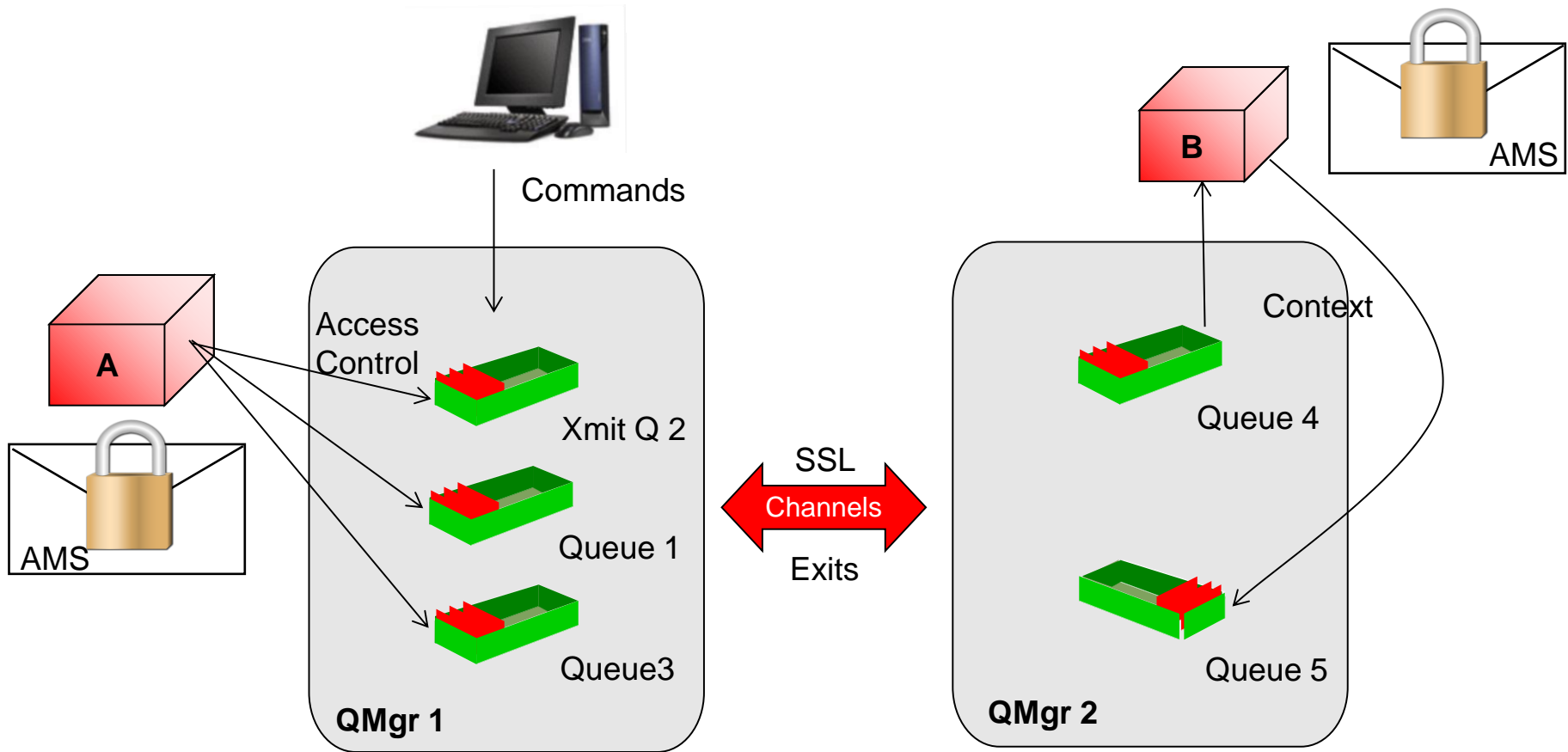
Transactions

MQBEGIN
MQCMIT
MQBACK

MQ Transactions

- MQ supports units of work (UoW) where a set of resource updates may be considered as an atomic unit - either all of the changes are made or none of the changes are made. This support is particularly important when using MQ in a commercial environment (it's primary focus) as transactions play a major part in this arena.
- MQ allows messages to be included/excluded from a UoW at the message level. This differs from some other environments where a UoW starts and all subsequent actions are included in the UoW. Thus, a set of messages may be considered to be a UoW. Often, it is necessary to include both MQ messages and some other recoverable resources (typically database updates) in a UoW. Typically, this has required the use of some Transaction Monitor and MQ works well with CICS and IMS on z/OS and with any XA compliant Transaction Manager. In situations where a Transaction Manager product is not available/suitable, MQ itself may be used as the Transaction Manager. This does not mean that MQ is transforming itself into a Transaction Monitor, it is just providing the Transaction Manager aspect of a Transaction Monitor product.
- The API used in handling transactions differs according to the environment. MQ provides some verbs to handle UoWs. If a Transaction Monitor is used, however, its UoW verbs are used in place of the MQI.

MQ Security



MQ Security

- There are several aspects to MQ security.
- Control of MQ commands :Access to MQ commands, like creating and starting queue managers, can be controlled through operating system facilities and also by MQ facilities; it is necessary to be in a particular authorisation group to be allowed to use these commands.
- Access to Queue Manager objects:There is an access control component that is provided by the MQ Queue Manager, called the Object Authority Manager (OAM), which controls access to Queue Manager objects, particularly queues. The OAM can control access to resources at a very granular level, allowing access for different actions, such as GET, PUT, INQ, SET, etc. This access is (generally) based upon group memberships.
- This security service is a pluggable component of MQ. Thus, if the OAM does not meet the requirements of the environment it is possible to provide a different (or additional) component. Note that the OAM is used for all queue managers except for the z/OS queue manager which uses any SAF compliant security manager.
- Encryption of message data through Advanced Message Security (AMS)

MQ Security (contd)

- Channel Security (Authentication)
- MQ 6.0 provides built-in SSL link level security
- MQ also provides a number of exit points during the transfer of messages between systems. The key exits concerned with security are :- Security Exit : This exit allows for (mutual) authentication of partner systems when they connect to one another. Message Exit: This exit allows for customisation at the message level, allowing individual messages to be protected, in terms of message integrity, message privacy and non-repudiation

MQ Security (contd)

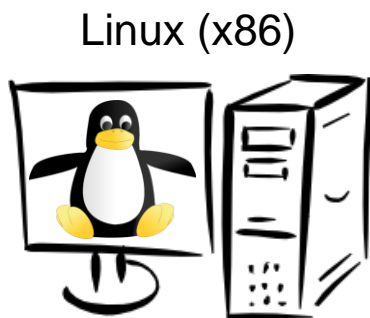
- Application Security
- This level of security is not implemented directly by the Queue Manager but such facilities may be implemented at the application level, outside of the direct control of MQ.
- Advanced Message Security
- Provides end to end security, enabling messages to be encrypted from the time they are PUT by the sending application to when they are GET by the receiving application, so messages are help encrypted when at rest on queues as well as when in transit.

Data Conversion

When receiving messages, MQ can convert the message payload data. This is most commonly used to convert character data so that it is in a format which is consumable by the receiving application.

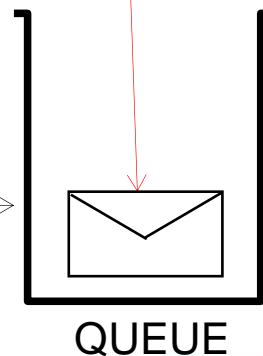
CCSID 500 (EBCDIC Latin-1-charset)
Data: H e l l o w o r l d !
Hex: C8 85 93 93 96 40 A6 96 99 93 84 4F

CCSID 1208 (UTF-8)
Data: H e l l o w o r l d !
Hex: 48 65 6C 6C 6F 20 77 6F 72 6C 64 21



MQGET

MQGMO_CONVERT
CCSID=1208

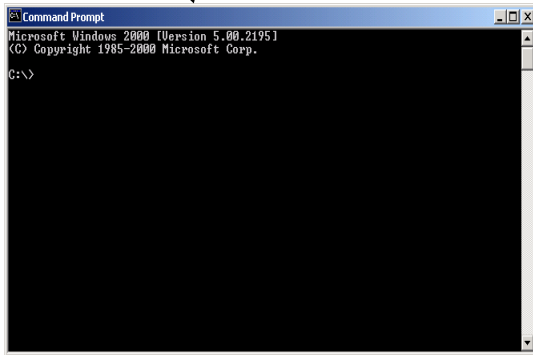


MQ Systems Management

MQ Explorer

Scripting

MQSC

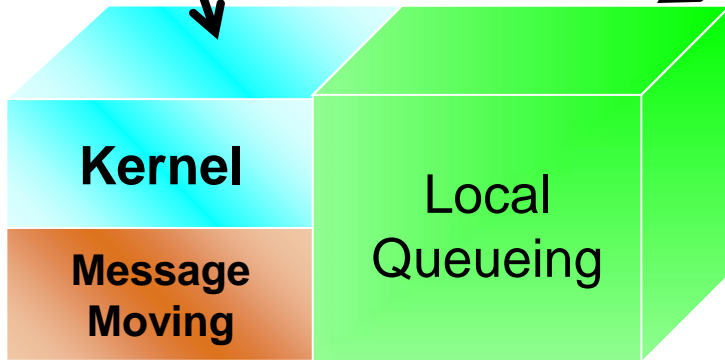


MQ Application

Programmable Command Format (PCF)



Queue name	Queue type	Open input count	Open output count
SYSTEM.ADMIN.ACCOUNTING.QUEUE	Local	0	0
SYSTEM.ADMIN.ACTIVITY.QUEUE	Local	0	0
SYSTEM.ADMIN.CHANNEL.EVENT	Local	0	0
SYSTEM.ADMIN.COMMAND.EVENT	Local	0	0
SYSTEM.ADMIN.COMMAND.QUEUE	Local	1	1
SYSTEM.ADMIN.CONFIG.EVENT	Local	0	0
SYSTEM.ADMIN.LOGGER.EVENT	Local	0	0
SYSTEM.ADMIN.PERFM.EVENT	Local	0	0
SYSTEM.ADMIN.PUBSUB.EVENT	Local	0	0
SYSTEM.ADMIN.QMGR.EVENT	Local	0	0
SYSTEM.ADMIN.STATISTICS.QUEUE	Local	0	0
SYSTEM.ADMIN.TRACE.ACTIVITY.QUEUE	Local	0	0
SYSTEM.ADMIN.TRACE.ROUTE.QUEUE	Local	0	0
SYSTEM.AUTH.DATA.QUEUE	Local	1	1
SYSTEM.BROKER.ADMIN.STREAM	Local	1	1
SYSTEM.BROKER.CONTROL.QUEUE	Local	3	0
SYSTEM.BROKER.DEFAULT.STREAM	Local	1	0
SYSTEM.BROKER.INTER.BROKER.COMM	Local	1	0
SYSTEM.CHANNEL.INITQ	Local	1	0
SYSTEM.CHANNEL.SYNCQ	Local	0	0
SYSTEM.CHL.AUTH.DATA.QUEUE	Local	0	0
SYSTEM.CICS.INITIATION.QUEUE	Local	0	0
SYSTEM.CLUSTER.COMMAND.QUEUE	Local	1	0

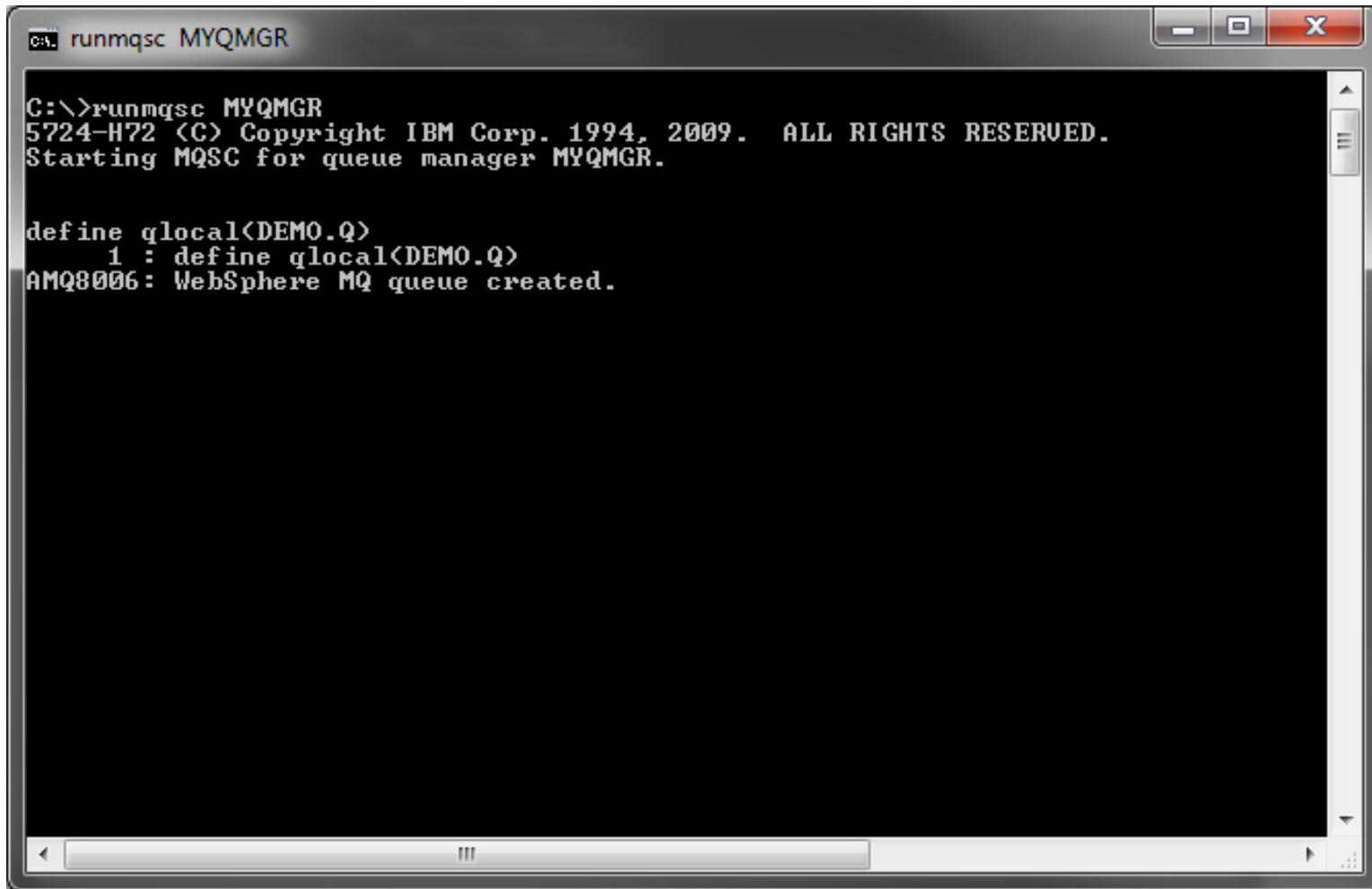


System Management Applications:

BMC, CA,
Landmark,
RYO, Tivoli



MQ Systems Management - runmqsc



```
runmqsc MYQMGR

C:\>runmqsc MYQMGR
5724-H72 (C) Copyright IBM Corp. 1994, 2009.  ALL RIGHTS RESERVED.
Starting MQSC for queue manager MYQMGR.

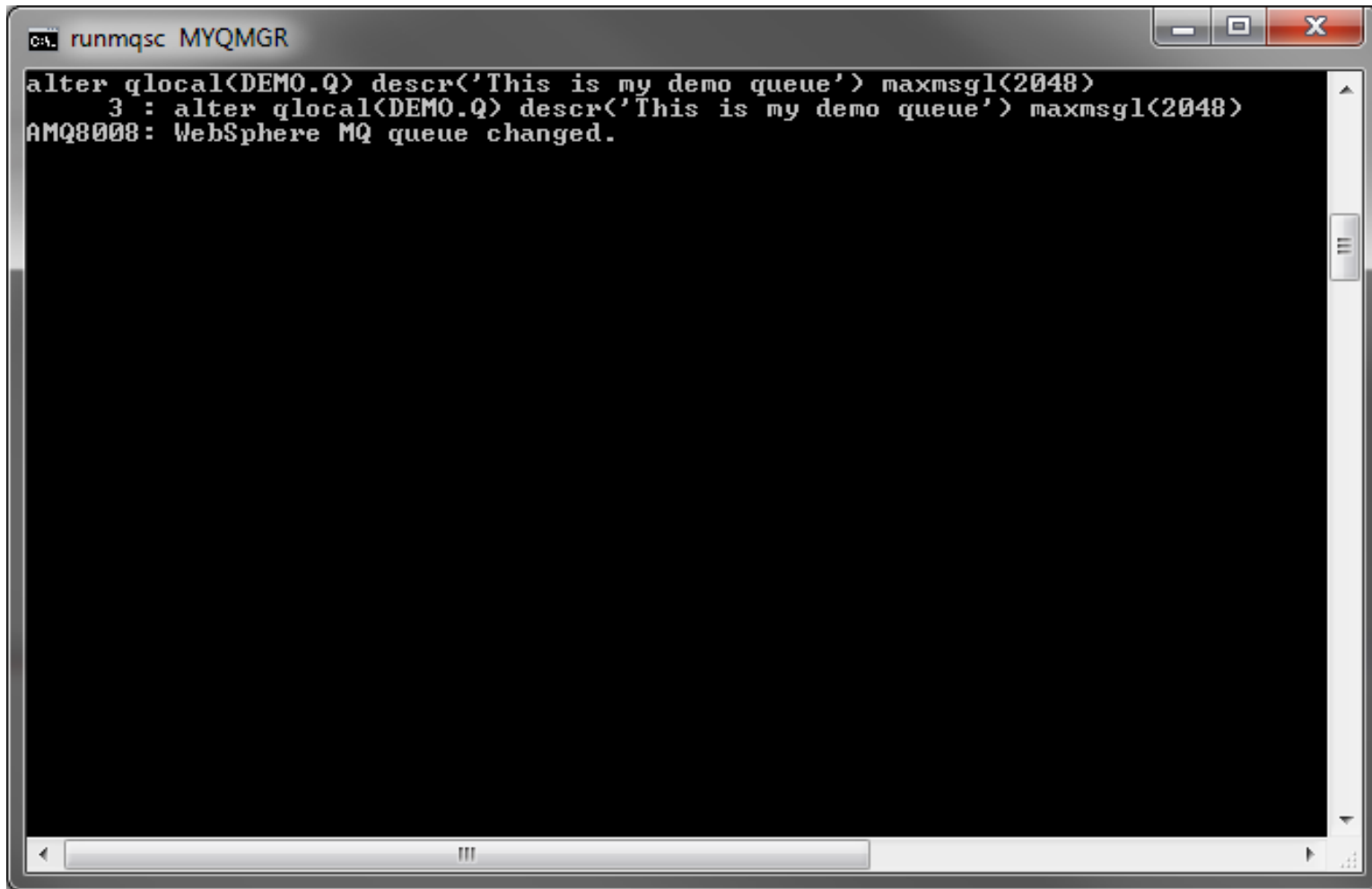
define qlocal(DEMO.Q)
  1 : define qlocal(DEMO.Q)
AMQ8006: WebSphere MQ queue created.
```

MQ Systems Management - runmqsc

```
runmqsc MYQMGR

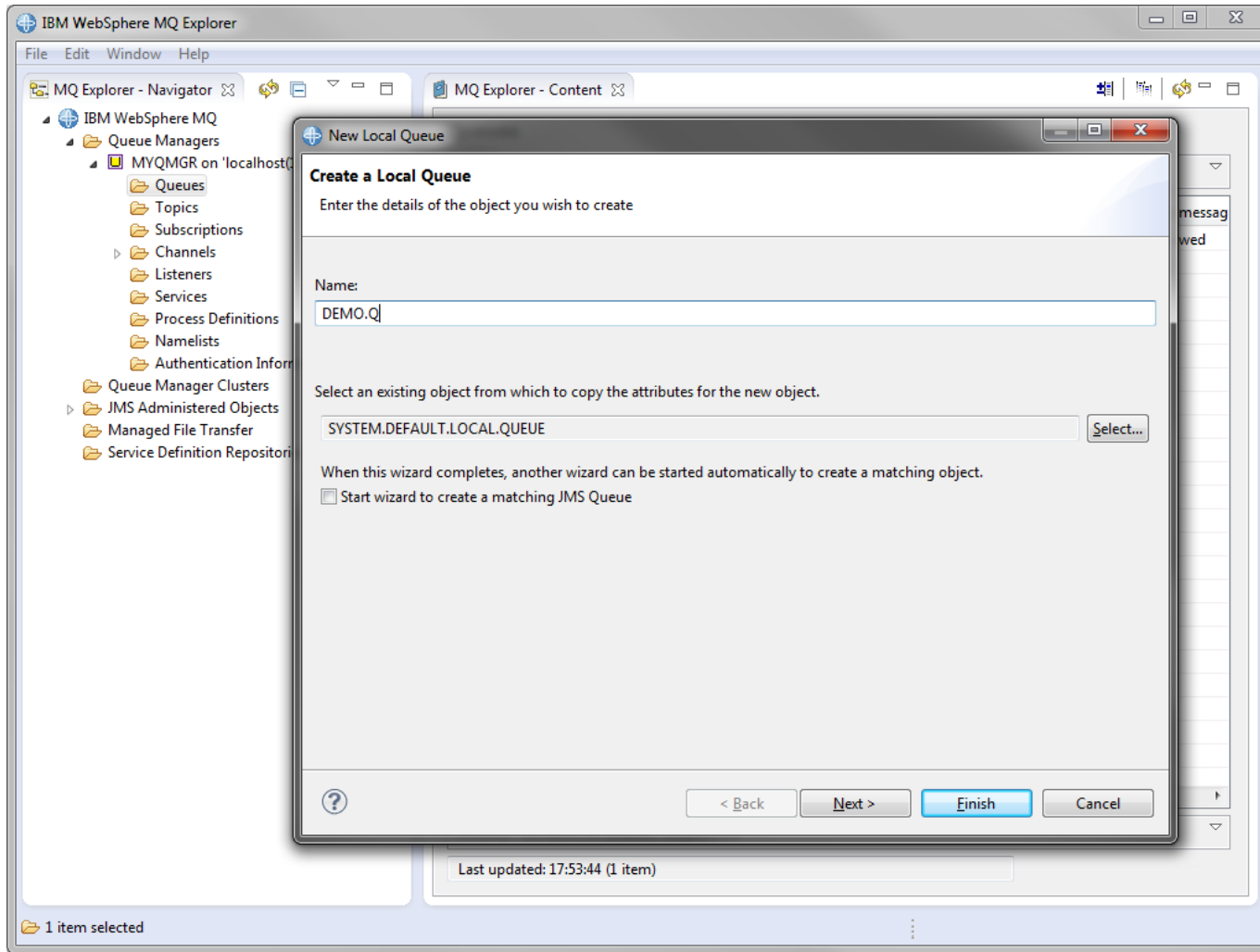
display qlocal(DEMO.Q)
  2 : display qlocal(DEMO.Q)
AMQ8409: Display Queue details.
  QUEUE<DEMO.Q>
  ACCTQ<QMGR>
  ALTTIME<13.40.46>
  BOTHRESH<0>
  CLUSTER< >
  CLWLRANK<0>
  CRDATE<2015-02-22>
  CURDEPTH<0>
  DEFPRTY<0>
  DEFPRESP<SYNC>
  DEFSOPT<SHARED>
  DESCR< >
  GET<ENABLED>
  INITQ< >
  MAXDEPTH<5000>
  MONQ<QMGR>
  NOTRIGGER
  OPPROCS<0>
  PUT<ENABLED>
  QDEPTHHI<80>
  QDPHI EU<DISABLED>
  QDPMAX EU<ENABLED>
  QSUCINT<999999999>
  SCOPE<QMGR>
  STATQ<QMGR>
  TRIGDPTH<1>
  TRIGTYPE<FIRST>
  TYPE<QLOCAL>
  ALTDATE<2015-02-22>
  BOQNAME< >
  CLUSNL< >
  CLWLPRTY<0>
  CLWLUSEQ<QMGR>
  CRTIME<13.40.46>
  DEFBIND<OPEN>
  DEFPSIST<NO>
  DEFREADA<NO>
  DEFTYPE<PREDEFINED>
  DISTL<NO>
  HARDENBO
  IPPROCS<0>
  MAXMSGL<4194304>
  MSGDLUSQ<PRIORITY>
  NPMCLASS<NORMAL>
  PROCESS< >
  PROPCTL<COMPAT>
  QDEPTHLO<20>
  QDPLOEU<DISABLED>
  QSUCIEU<NONE>
  RETINTUL<999999999>
  SHARE
  TRIGDATA< >
  TRIGMPRI<0>
  USAGE<NORMAL>
```


MQ Systems Management - runmqsc

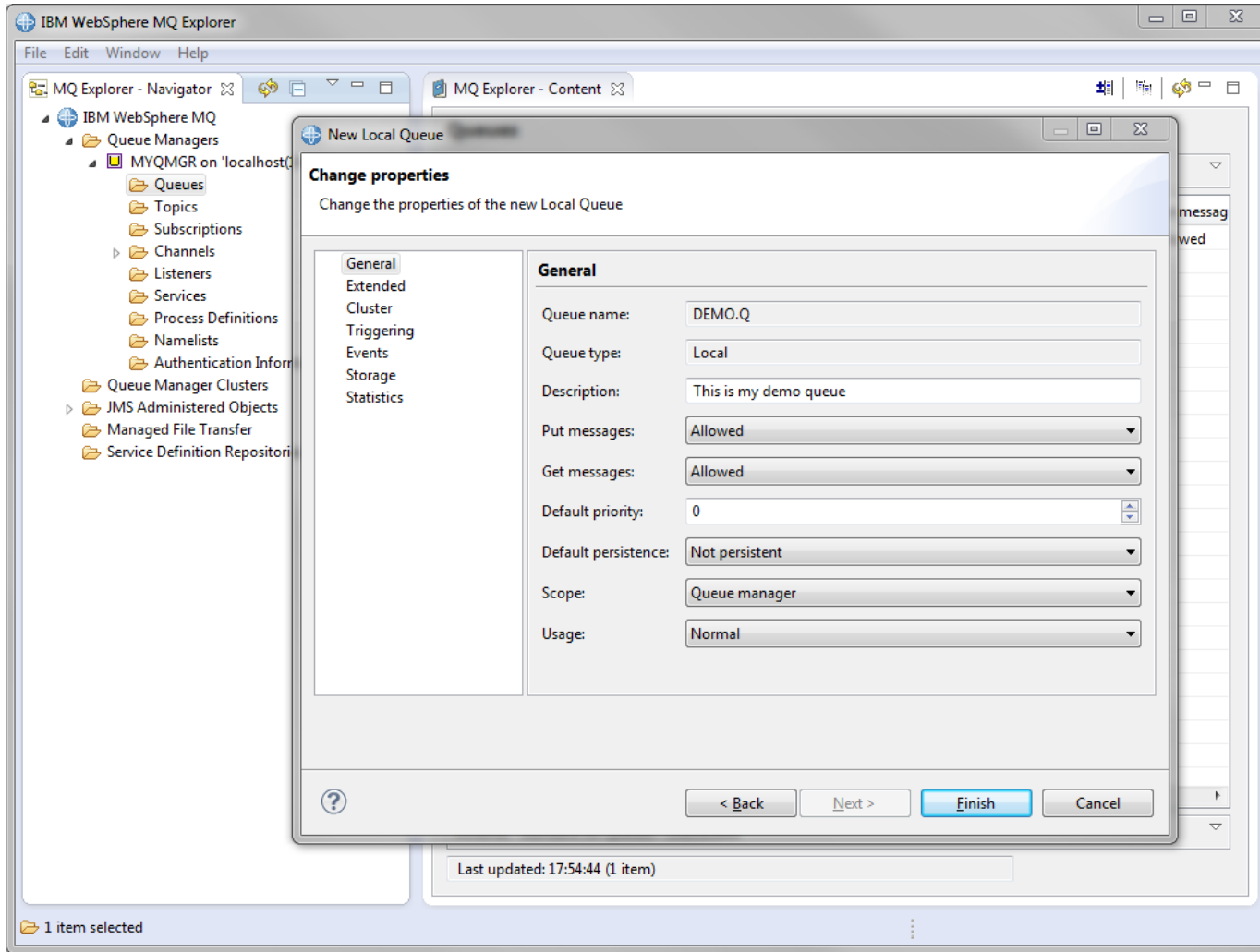


```
cmd: runmqsc MYQMGR
alter qlocal(DEMO.Q) descr('This is my demo queue') maxmsgl(2048)
3 : alter qlocal(DEMO.Q) descr('This is my demo queue') maxmsgl(2048)
AMQ8008: WebSphere MQ queue changed.
```

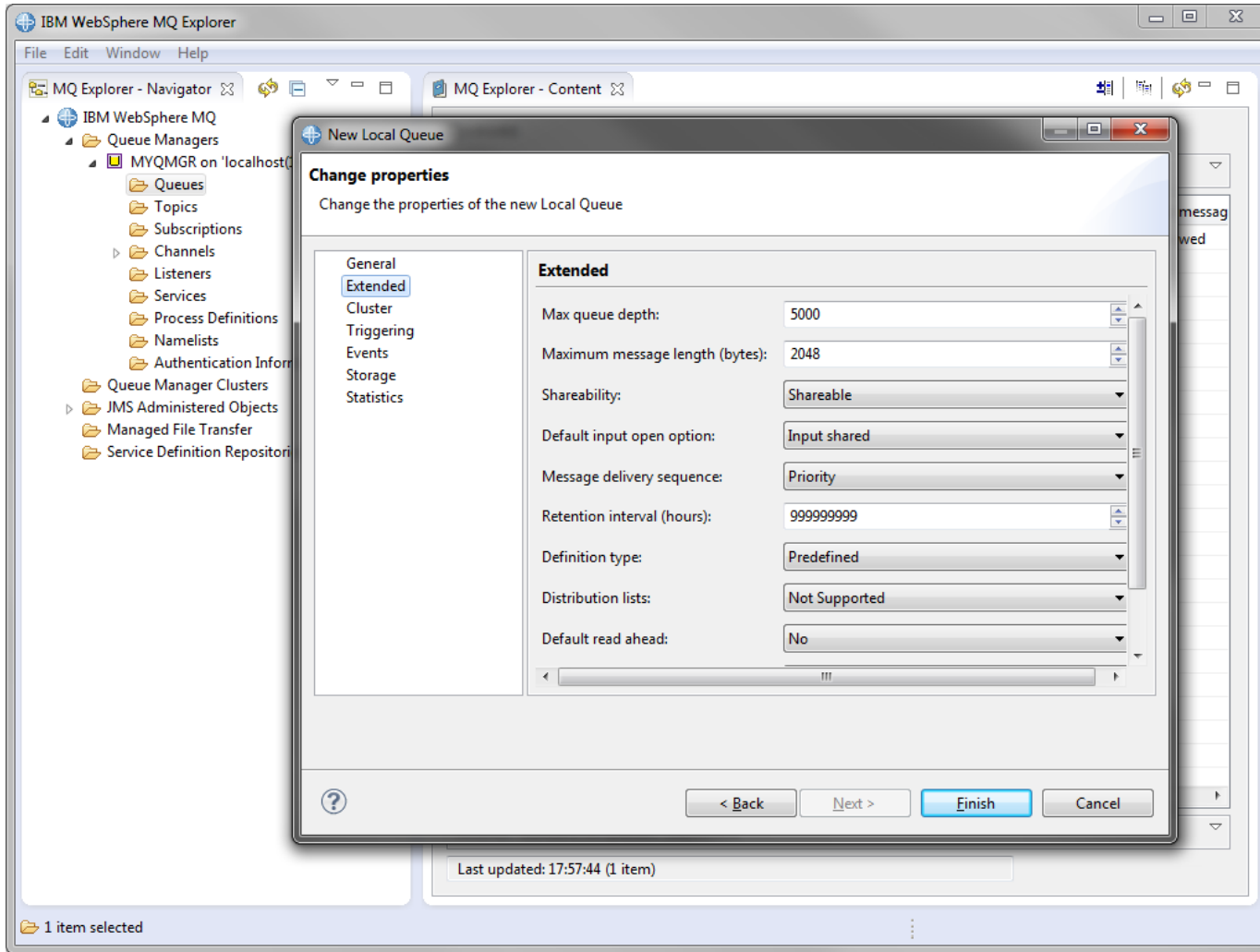
MQ Systems Management – MQ Explorer



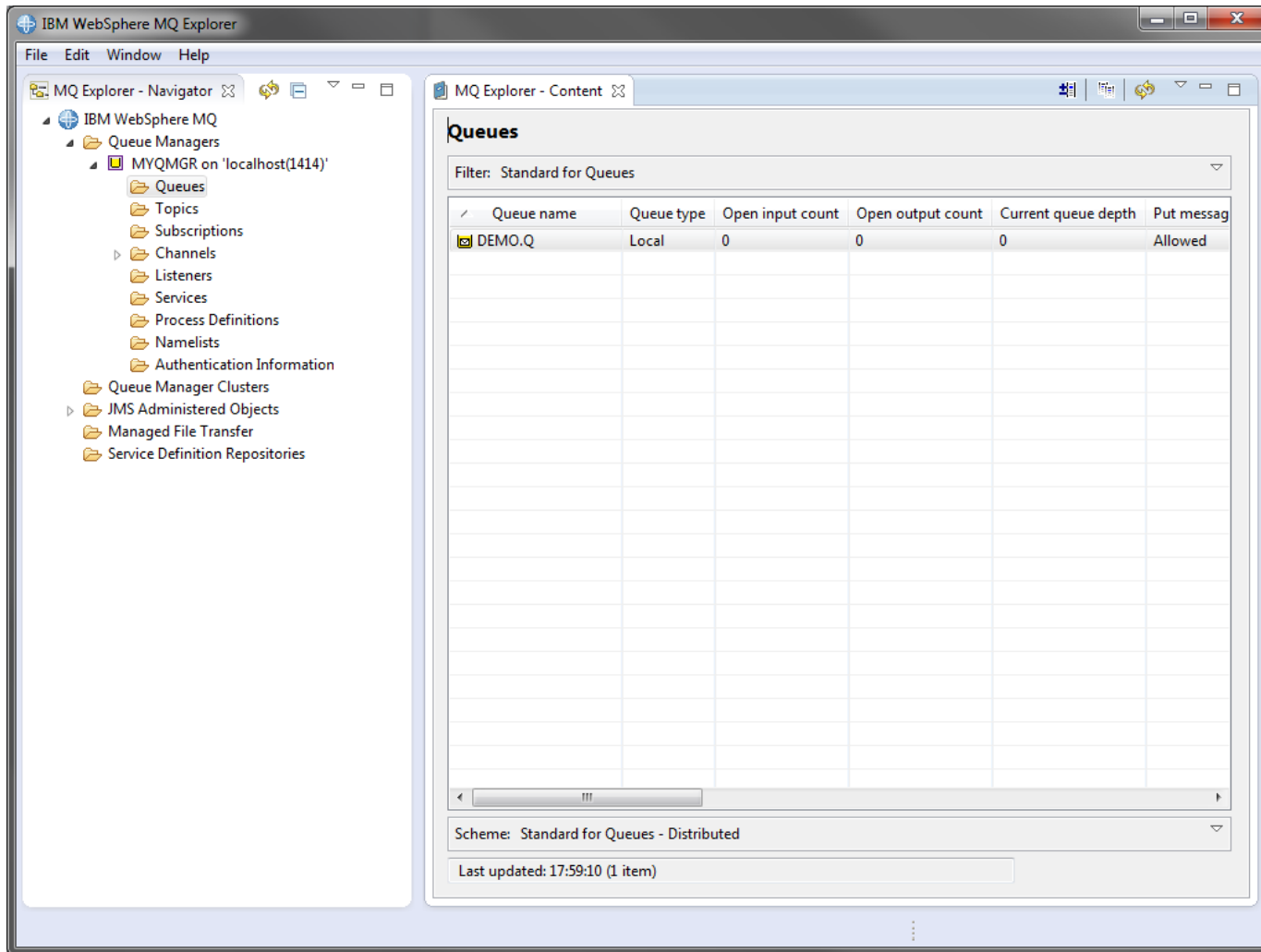
MQ Systems Management – MQ Explorer



MQ Systems Management – MQ Explorer



MQ Systems Management – MQ Explorer



The screenshot shows the IBM WebSphere MQ Explorer application. The left pane displays a tree view of the MQ environment, including Queue Managers, Channels, and various objects. The right pane shows the 'Queues' view for a selected queue manager, displaying a table of queue details.

Queue name	Queue type	Open input count	Open output count	Current queue depth	Put message
DEMO.Q	Local	0	0	0	Allowed

Additional details from the interface:
- Filter: Standard for Queues
- Scheme: Standard for Queues - Distributed
- Last updated: 17:59:10 (1 item)

Agenda

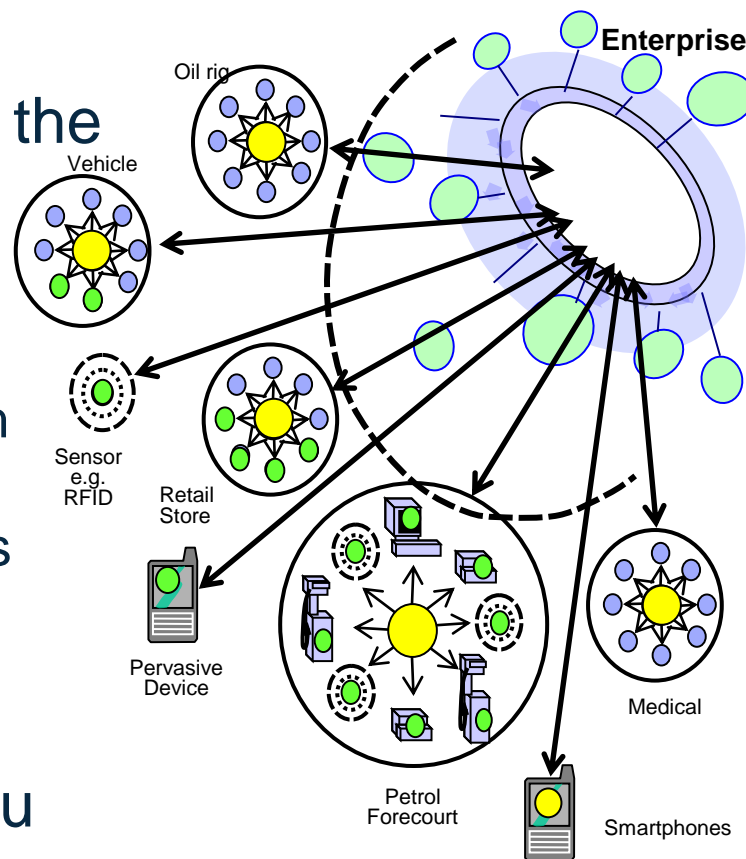
- Why use messaging?
- Fundamentals of MQ
- Using the MQ API
- Other key features
- **Extensions and related products**
- Putting it all together...
- Summary

MQ and the wider world

- For a messaging engine to be really useful it should allow access to the messages from many different environments. We have already discussed MQs programming language and API support but what about the environments.
- The complexity of overall business applications is increasing every year as more and more applications are linked together in some way. MQ dramatically reduces an individual applications complexity by providing a consistent, reliable and transactional method of communicating between applications from hundreds of different environments.
- We are now going to look briefly at some of the other WebSphere Business Integration products that make up the portfolio, and how MQ fits in

MQ Telemetry

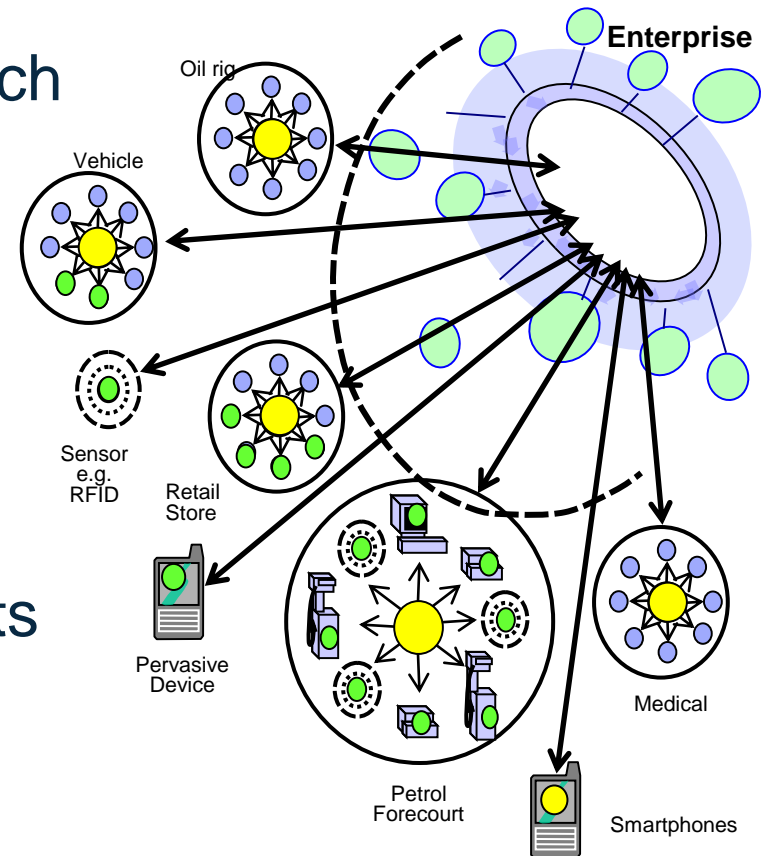
- Product extension included from MQ 7.1 (MQXR), supporting mass connectivity for smart devices to the enterprise
- Utilises MQTT protocol
 - a lightweight, public, low bandwidth messaging protocol for scenarios where enterprise messaging clients are too big or bandwidth intensive.
 - Established for >10 years
- Java and C API provided, but you can “roll your own”



MQ Telemetry

Ideally suited to:

- Fragile / Expensive networks such as “sometimes connected” devices / satellite phones
- Niche platforms such as tiny sensors, personal devices, edge/small servers
- Mass Scalability (> 50,000 clients per queue manager)



MQ Advanced Message Security

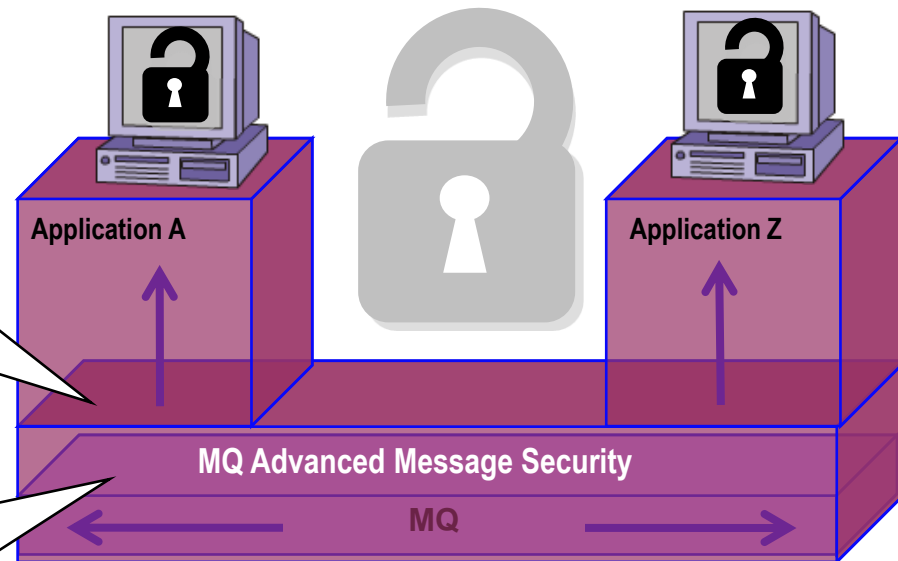
- Secures application data even before it is passed to MQ
- Upgrade from base MQ – No changes to existing applications or network required

MQ standard security:

- Industry standard SSL channels (256-bit)
- Certified for Common Criteria
- Authentication is based on Operating System identifier of local process
- Message data can be encrypted in transport but not when it resides in the queues

MQ Advanced Message Security adds:

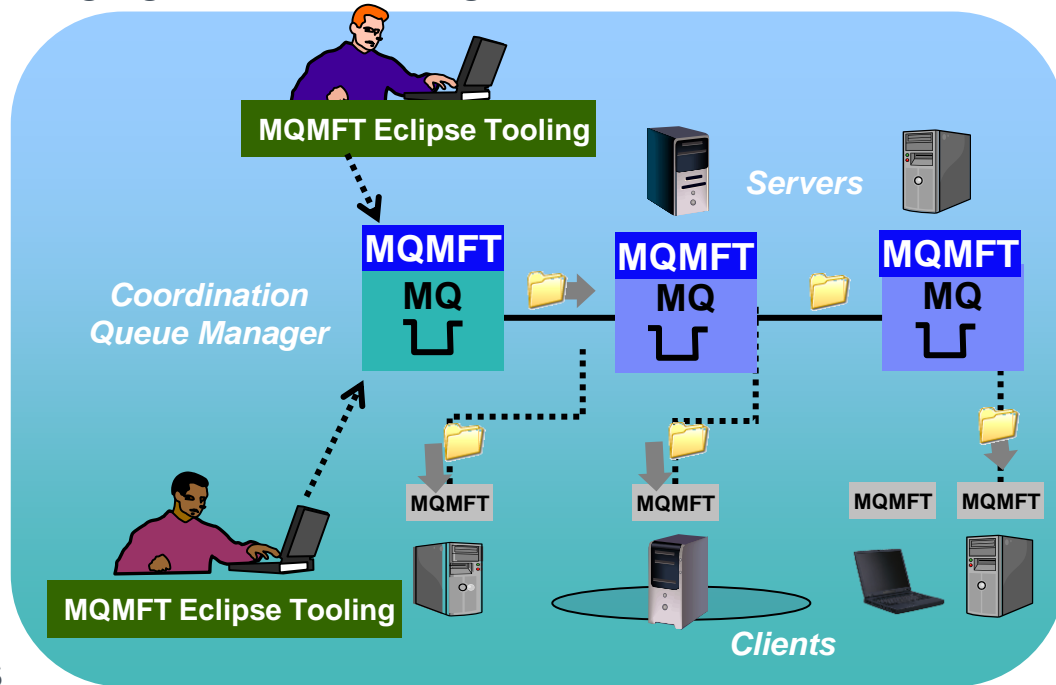
- + Authentication policies are based on certificates associated with each application
- + Message data is protected end-to-end – including when it resides in queues
- + Much finer granularity in security policies
- + No changes needed to applications or queues



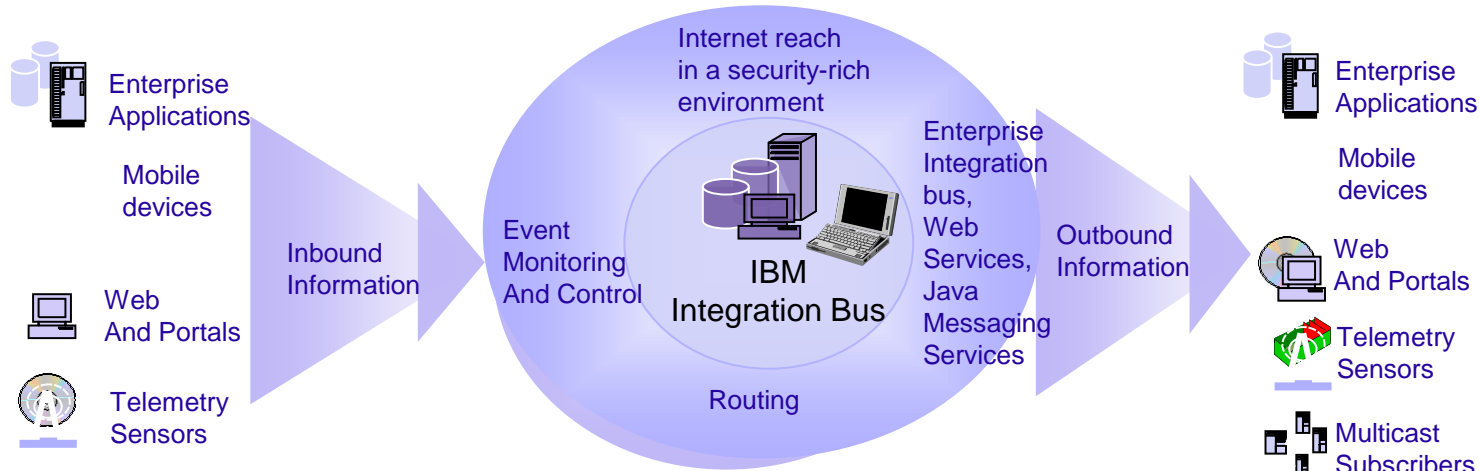
*Securing the data
and the applications*

MQ Managed File Transfer

- MQ MFT/FTE solves problems of auditing, monitoring, scheduling and securing file transfer operations
 - Automated bulk data transfer between distributed heterogeneous systems.
 - Capabilities for integrating, managing, and controlling data movement.
- Built on MQ
 - Assured delivery of data backbone
- Simplicity and flexibility
 - MQ Explorer Integration
 - Scriptable
 - Scheduled, or Triggered transfers
 - Complements MB File Nodes



WebSphere Message Broker / IBM Integration Bus



IBM Integration Bus

- Formerly known as WebSphere Message Broker
- Message transformation (mediations)
- Combine data sources: databases, files, etc.
 - Update other data stores: databases, files, etc.
 - Adapters - SAP, PeopleSoft, ORACLE, Files, e-mail...
 - Content based filtering and routing

Complete your session evaluations online at www.SHARE.org/Orlando-Eval

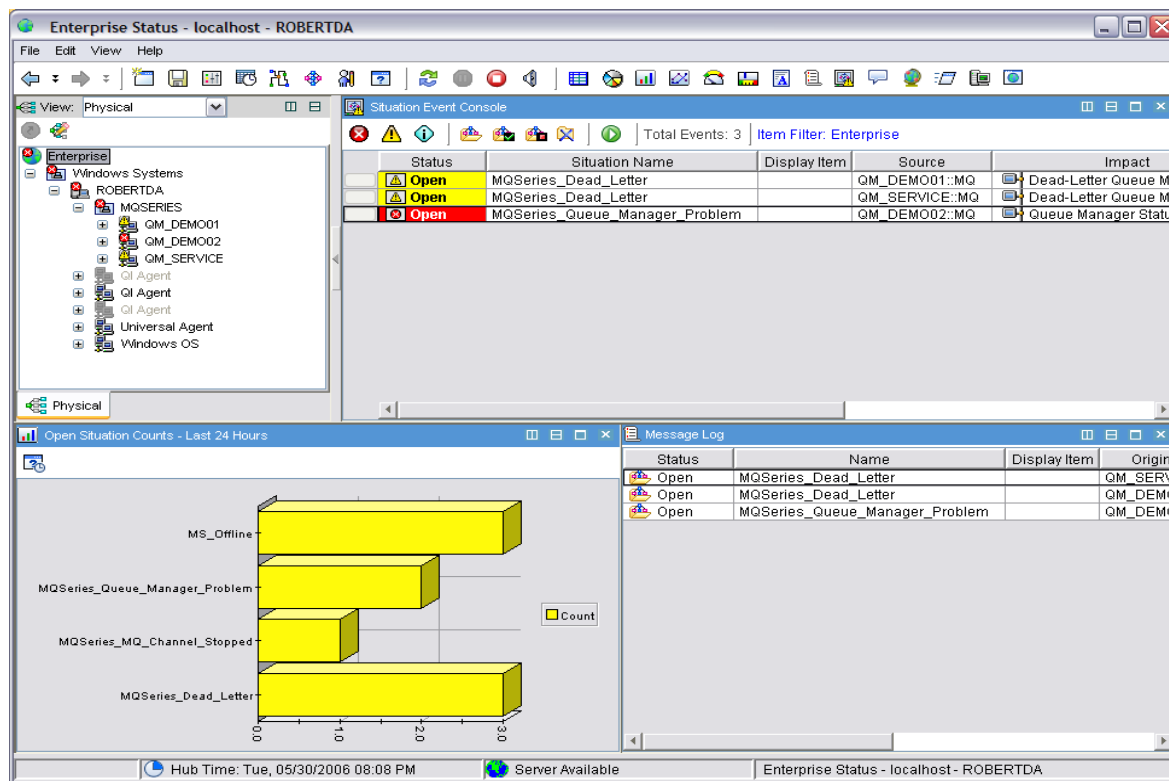
IBM Integration Bus

- MQ provides the assured delivery backbone to an Enterprise Service Bus. The queue managers are message content agnostic. Consequently, any data may be exchanged between applications. However, many applications are dependent upon their data being routed to particular destinations and are dependent upon particular data formats. So, the fact that applications may exchange data (via MQ and/or WebSphere Adapters) does not solve all possible problems. For the general case of any to any application integration, an intermediary is required to handle message routing issues and to handle (both simple and very complex) message transformation issues.

- Message Broker provides the function that enables complex message routing and transformation functions to be encapsulated outside of applications, in a (logically) central component.

Tivoli Omegamon and ITCAM

- Range of IBM products for monitoring and managing common core technologies with product-specific integration
 - eg Omegamon for Messaging deals with MQ and Message Broker



omegamon

Explorer will handle
single views
both MQ and DB2

nt team
res

e product/OS

Getting MQ : Free Trial

<http://www.ibm.com/software/products/en/ibm-mq>



Industries & solutions Services Products Support & downloads My IBM

IBM Software > Products > Connectivity, integration and SOA > Messaging > WebSphere MQ >

IBM MQ

IBM Software
See what smarter software
can do for you.



Accelerate growth by exploiting reliable, robust, industry-leading messaging solution

IBM MQ 90 day no-charge trial

[Download software](#)



IBM® MQ is robust messaging middleware that simplifies and accelerates the integration of diverse applications and business data across multiple platforms. IBM MQ facilitates the assured, secure and reliable exchange of information between applications, systems, services and file by sending and receiving message data via messaging queues, thereby simplifying the creation and maintenance of business applications. It delivers Universal Messaging with a broad set of offerings to meet enterprise-wide messaging needs, as well as connectivity for the internet of things and mobile devices.

MQ Advanced for developers is available for all development purposes.

IBM MQ provides:

Not in United States?

Select another country.

Considering a purchase?



Contact IBM

[Email IBM](#)

[Request a quote](#)

[Or call us at: 1-877-426-3774](#)
Priority code: WebSphere

[Product support](#)

Complete your session evaluations online at www.SHARE.org/Orlando-Eval

Agenda

- Why use messaging?
- Fundamentals of MQ
- Using the MQ API
- Other key features
- Extensions and related products
- Putting it all together...
- Summary

Putting it all together...

The sales department scenario



- The sales department has been using a legacy quote application for a number of years
 - The application runs standalone, with various products, prices and rules hardcoded
 - Updating this information means rebuilding the application with the new parameters, and deploying to each workstation out-of-hours
- This approach has many drawbacks
- Luckily, the CIO has commissioned a new solution to address this!

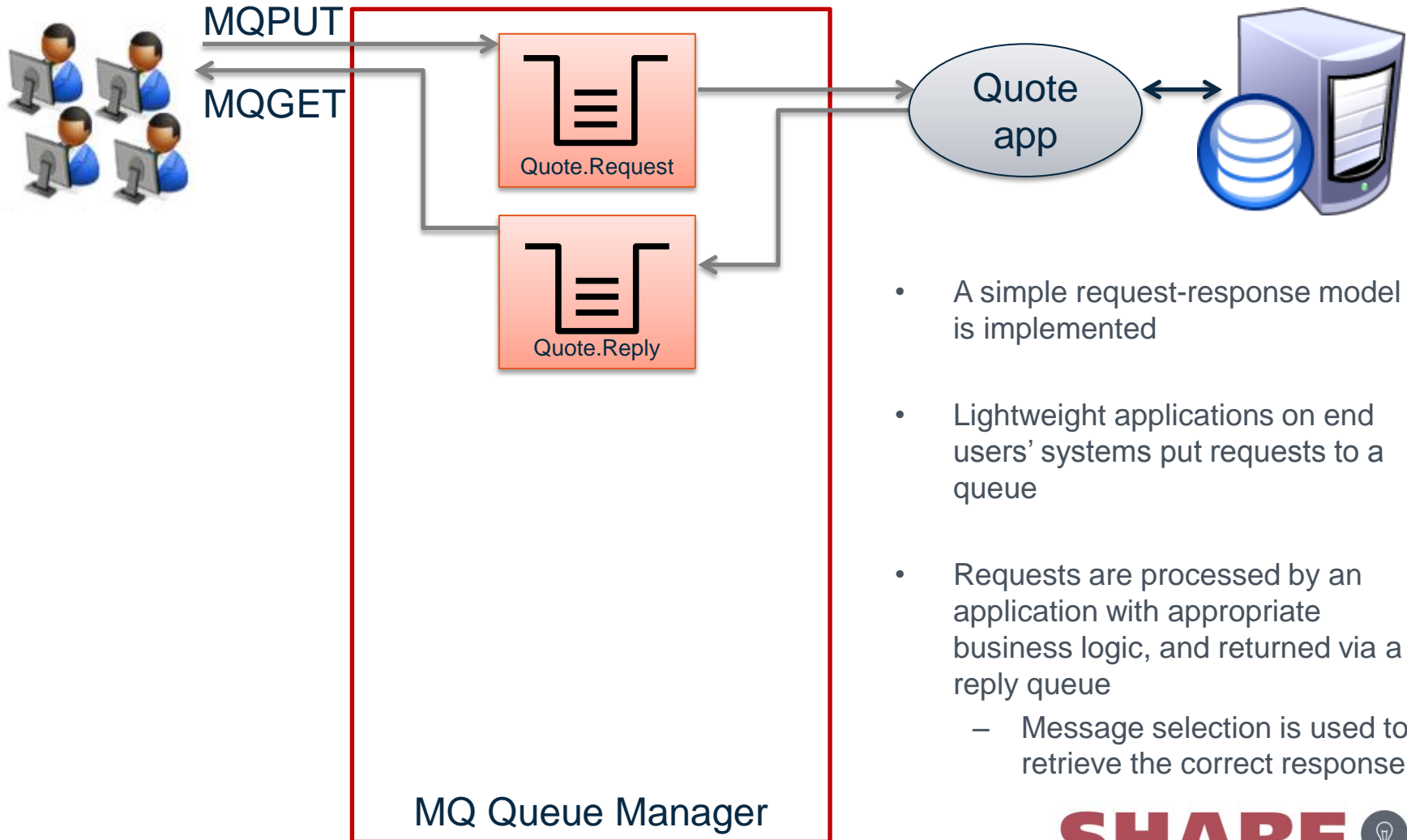
Requirements phase 1



- Agreement is reached that all dynamic information should be stored in a database, rather than in application logic
- Concerns are raised about how to manage end users' access to the database, as well as how best to handle multiple concurrent requests
- A messaging backbone solution is proposed



Solution phase 1



- A simple request-response model is implemented
- Lightweight applications on end users' systems put requests to a queue
- Requests are processed by an application with appropriate business logic, and returned via a reply queue
 - Message selection is used to retrieve the correct response

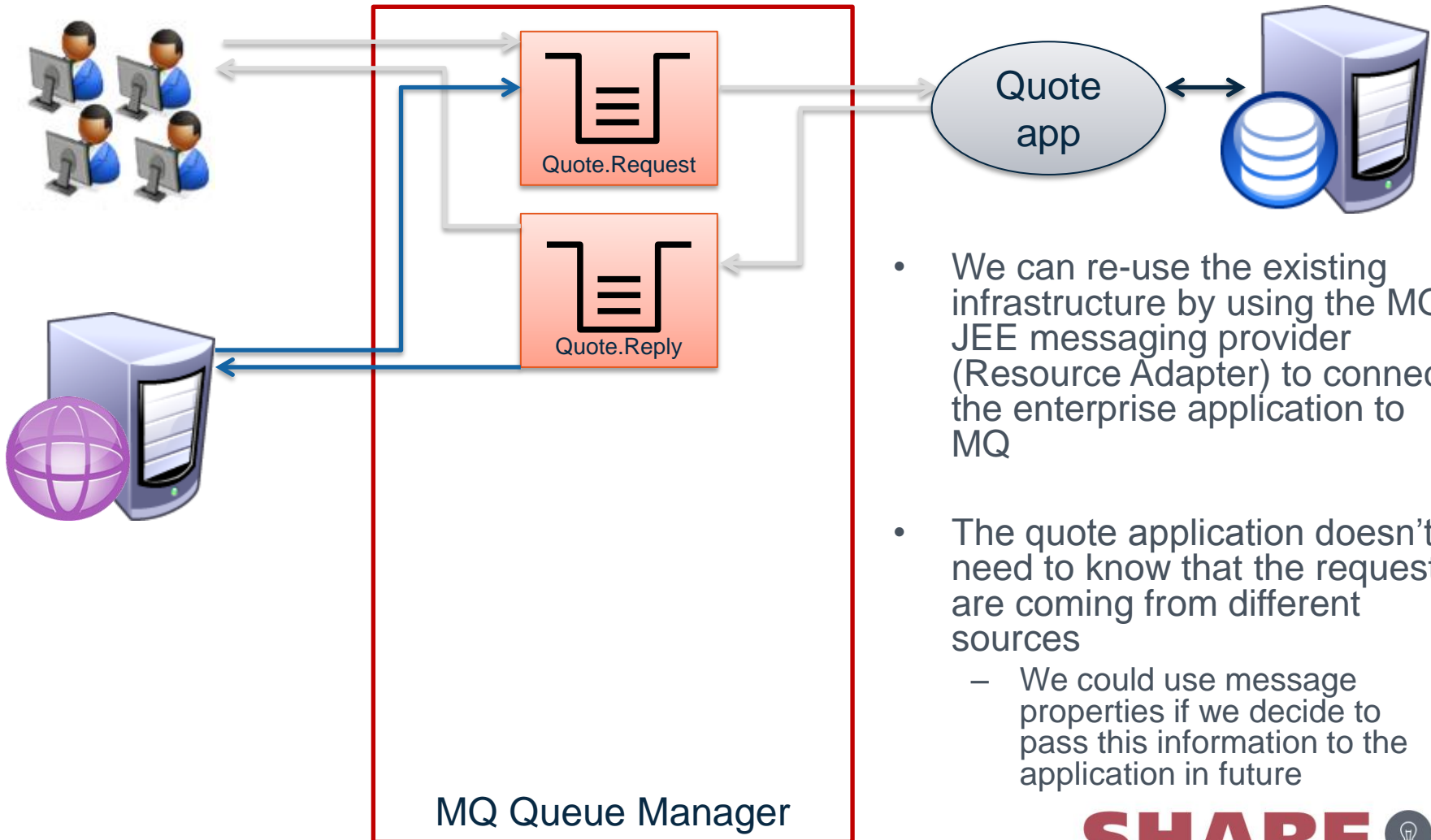
Requirements phase 2



- The CIO wants to deploy a new company website, which will allow users to request their own quotes without needing to speak with a sales representative
- An application server is chosen to host the web content and enterprise applications to provide this new capability
- How can the prior investment in the messaging backbone assist with this?



Implementation phase 2



- We can re-use the existing infrastructure by using the MQ JEE messaging provider (Resource Adapter) to connect the enterprise application to MQ
- The quote application doesn't need to know that the requests are coming from different sources
 - We could use message properties if we decide to pass this information to the application in future

Requirements phase 3



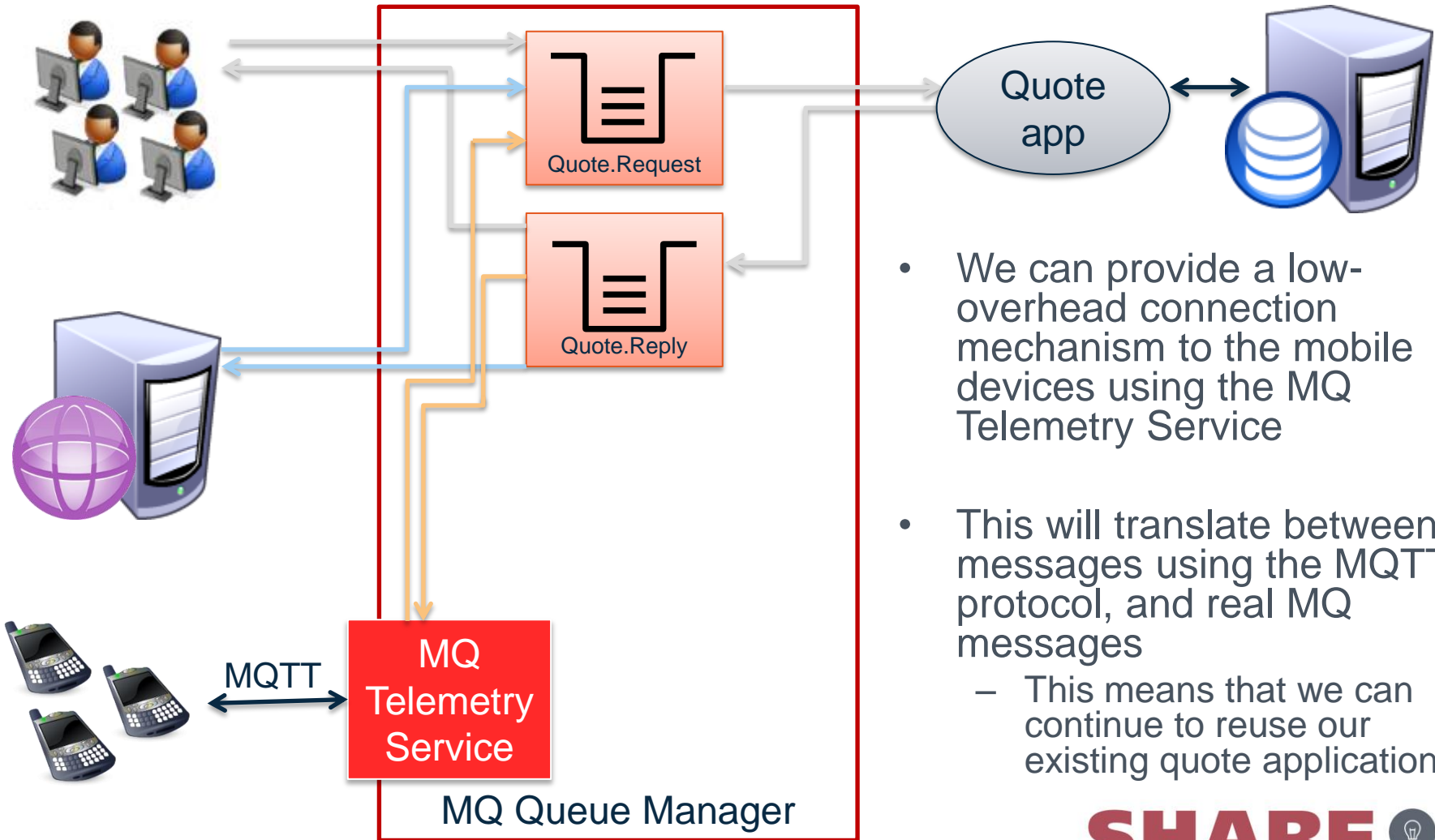
- The new website was a success
- Now, the sales team want a solution which will allow their sellers in the field to obtain a quote directly from their mobile device



- The CIO has cautioned that he wants to keep mobile data spend as low as possible

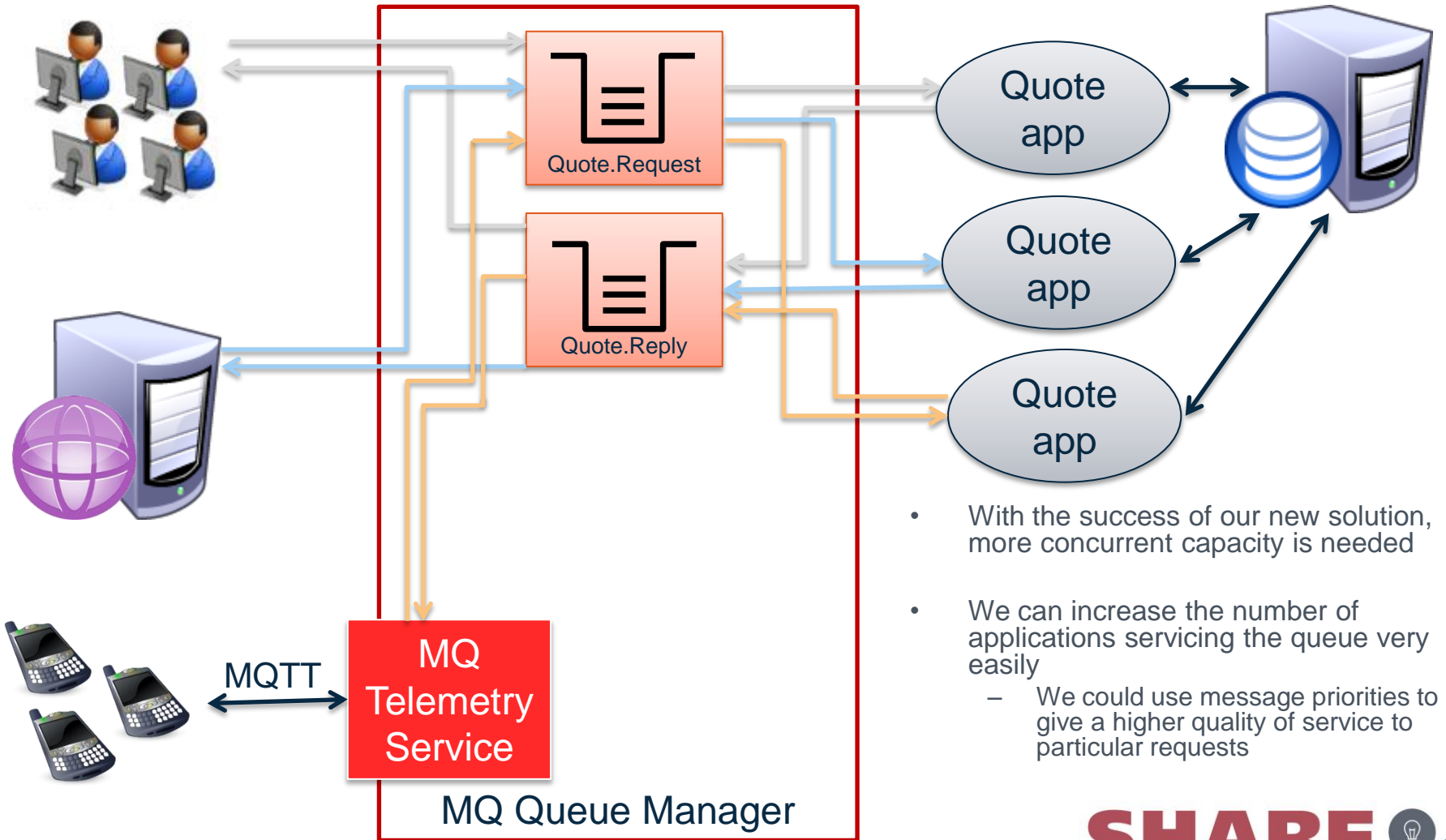


Implementation phase 3



- We can provide a low-overhead connection mechanism to the mobile devices using the MQ Telemetry Service
- This will translate between messages using the MQTT protocol, and real MQ messages
 - This means that we can continue to reuse our existing quote application

Phase 4



- With the success of our new solution, more concurrent capacity is needed
- We can increase the number of applications servicing the queue very easily
 - We could use message priorities to give a higher quality of service to particular requests

Phase 5



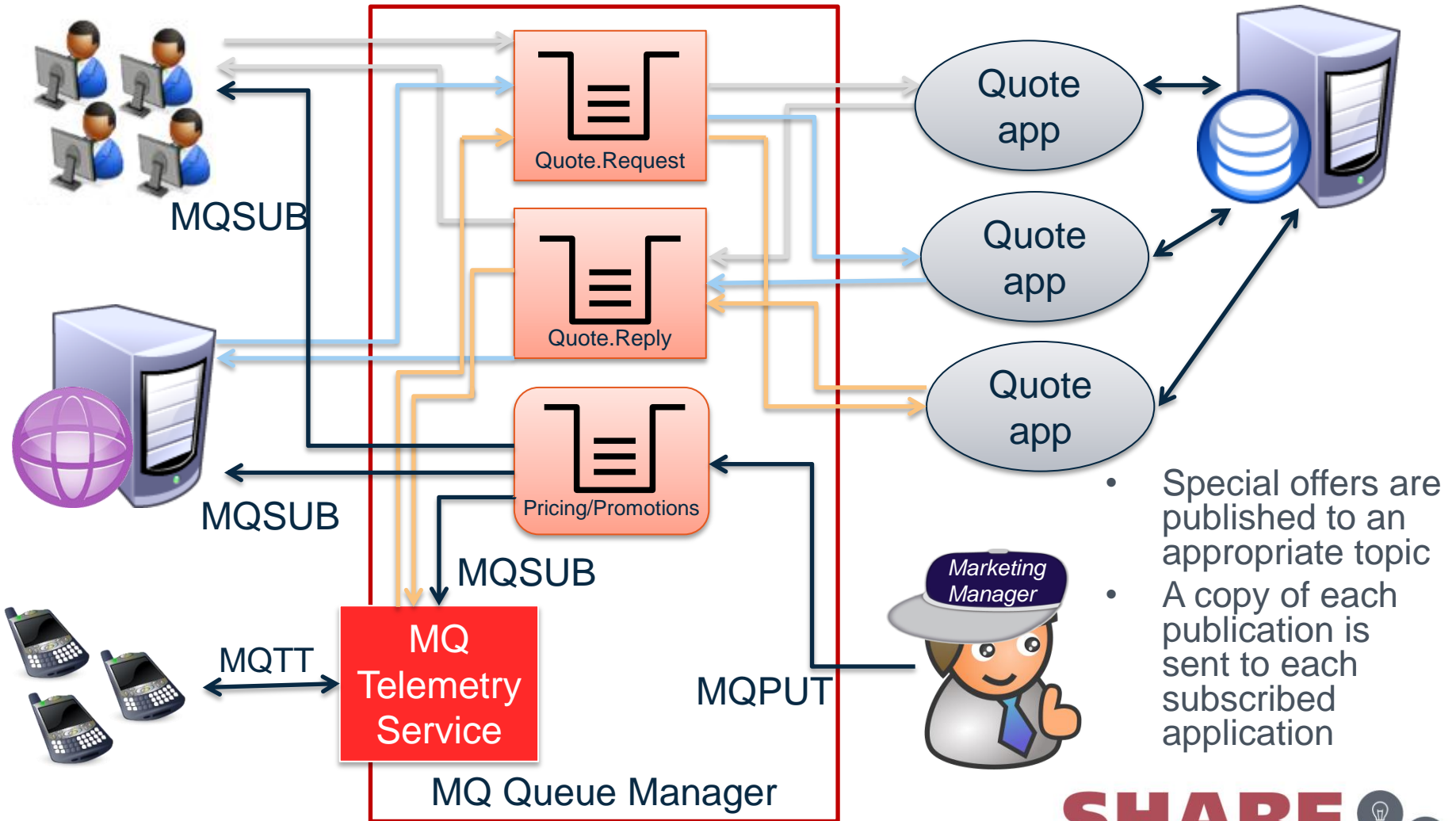
- The Marketing Manager asks for new function to provide feedback such as special offers to users in real time



- Given the broad audience, this sounds like an ideal use-case for publish/subscribe messaging



Phase 5



Future enhancements

So far we've only used a small fraction of MQ's capabilities. Some other considerations we could make in future enhancements to our solution are:

- MQ multi-instance capability to provide a backup system in case of failure
- MQ clustering to provide more advanced workload balancing options
- MQ Managed File Transfer to provide a reliable, auditable transport for data files
- MQ Advanced Message Security to sign and encrypt message bodies

Agenda

- Why use messaging?
- Fundamentals of MQ
- Using the MQ API
- Other key features
- Extensions and related products
- Putting it all together...
- **Summary**


Summary

- MQ - World leader in messaging technology
- Runs everywhere your applications do
- Simplifies application communication
 - From simple connectivity.....
 - to complex workload balancing, transformation and routing
- Provides secure, reliable and high-speed infrastructure

NEW MQ Labs!

- Wednesday August 12 @ 11:15 in the Dolphin Room Asia 5
 - The Dolphin is not the Hursley pub but the Orlando Hotel
- The MQ project including the developers will be there for Q&A as well
- YOU have the opportunity to be the test subjects for two NEW MQ labs:
 - Developing and Deploying JMS Enabled CICS Applications
 - Explore this brand new feature of MQ and CICS
 - Introduction to Channel Authorization on z/OS
 - You too can keep BOGUS clients off your queue manager
- In addition all the ‘regular’ MQ V8 and V7.1 labs are available if you want to try things out!

This was Session #17885 The rest of the week ...

	Monday	Tuesday	Wednesday	Thursday	Friday
08:30			MQ for z/OS, Using and Abusing New Hardware and the New v8 Features	Nobody Uses Files Any More Do They? New Technologies for Old Technology, File Processing in MQ MFT and IIB	Monitoring and Auditing MQ
					Securing MQ Initiated CICS Workload
10:00	Introduction to MQ - Can MQ Really Make My Life Easier?	MQ for z/OS: The Insider Story	IBM Integration Bus MQ Flexibility	Common Problems and Problem Determination for MQ z/OS	IBM MQ and IBM Integration Bus - from Migration and Maintenance to Continuous Enhancements, How and Why to Stay Current
11:15	Introduction to IBM Integration Bus on z/OS	Introduction to the New MQ Appliance	MQ V8 Hands-on Labs! MQ V8 with CICS and COBOL! MQ SMF Labs!		
12:15					
1:45	What's New in the Messaging Family - MQ v8 and More		Getting Started with Performance of MQ on z/OS	IBM MQ: Are z/OS & Distributed Platforms Like Oil & Water?	
3:15	What's New in IBM Integration Bus	Live!: End to End Security of My Queue Manager on z/OS	Digging into the MQ SMF Data	MQ Parallel Sysplex Exploitation, Getting the Best Availability from MQ on z/OS by Using Shared Queues	
		Application Programming with MQ Verbs			
4:30	MQ Security: New v8 Features Deep Dive	Live!: What's the Cloud Going to Do to My MQ Network?	Giving It the Beans: Using IBM MQ as the Messaging Provider for JEE Applications in IBM WebSphere Application Server	Challenge the MQ & IIB Experts	
		The Do's and Don'ts of IBM Integration Bus Performance			

Copyright and Trademarks

© IBM Corporation 2015. All Rights Reserved.

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at “Copyright and trademark information” at www.ibm.com/legal/copytrade.shtml.