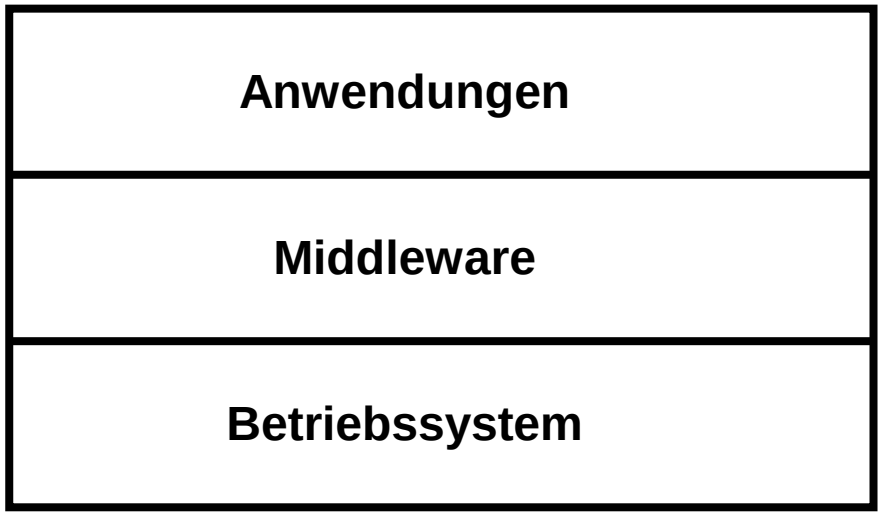
Einführung in z/OS
Enterprise Computing

Prof. Dr. Martin Bogdan
Dr. rer. nat. Paul Herrmannn
Prof. Dr.-Ing. Wilhelm G. Spruth

WS 2008/2009

Teil 12

MQSeries,, WebSphere MQ



Middleware

Software, die zwischen Anwendungen und Betriebssystem liegt, wird häufig als Middleware bezeichnet.

Ein Beispiel für Middleware sind Transaktionsmonitore wie CICS, IMS/DC, Tuxedo oder der SAP/R3 Transaktionsmonitor.

Andere Beispiele sind RPC, DCE, CORBA, RMI sowie Message Based Queuing (MBQ).

Message Based Queuing (MBQ)
Message oriented Middleware (MOM)

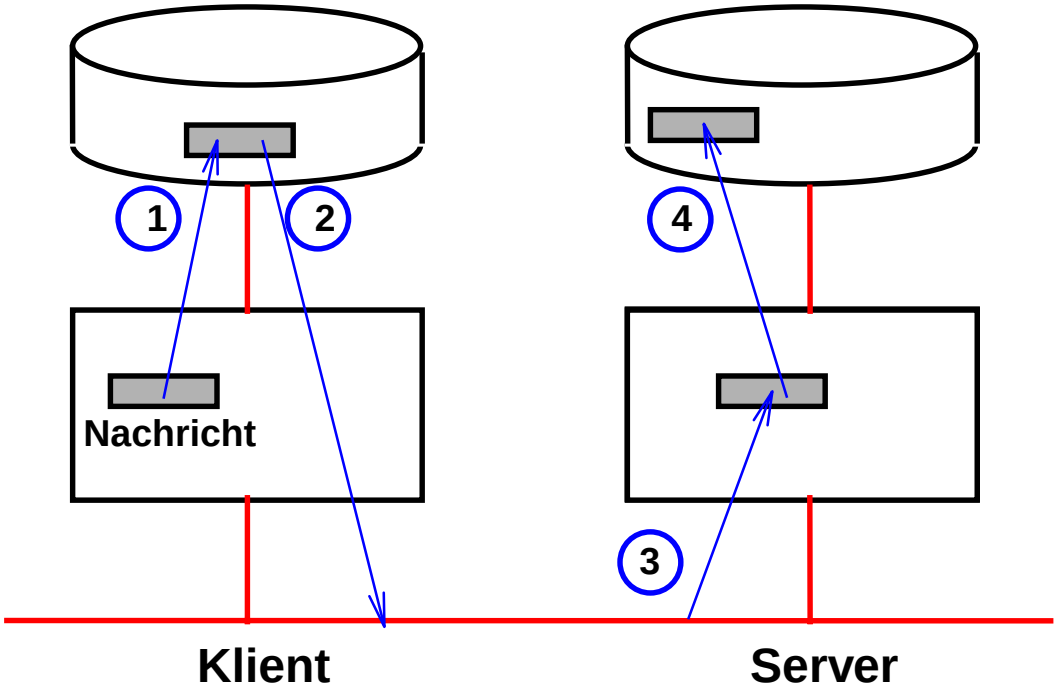
Führende Hersteller:

- **BEA** **MessageQ**
- **IBM** **MQSeries**
- **Microsoft** **MS Message Queue Server (MSMQ)**
- **Oracle** **Advanced Queuing (AQ),**

Message Based Queuing ist besser geeignet, wenn es um die Integration unterschiedlicher, heterogener Hardware, Software und Anwendungen geht.

SMTP und X.500 electronic Mail sind einfache Message orientierte Anwendungen

Literatur: B. Blakeley: “Messaging & Queuing using the MQI”. MvGrawHill, 1995.



MBQ - Message Based Queuing,
Message Oriented Middleware (MOM

Alternative zum RPC, Ersatz für Batch file transfers

Eigenschaften:

- **Nachrichten werden bis zur endgültigen Auslieferung an die Zielanwendung in Warteschlangen zwischengespeichert**
- **Asynchron (im Gegensatz zum RPC, Store-and-Foreward Prinzip)**
- **Recovery Mechanismen beim Versagen von Knoten oder Verbindungen**
- **Auslieferung wird garantiert**
- **Message Tracking (lokale Platte, entfernte Platte, Annahme der Nachricht durch Anwendung)**

Die Operation wird durch einen *Queue Manager* auf jedem Rechner gesteuert

Message Based Queuing (MBQ)
Message Oriented Middleware (MOM)

Message Queuing ist eine Methode der Programm-zu-Programm-Kommunikation. Programme sind in der Lage, Informationen zu senden und zu empfangen, ohne dass eine direkte Verbindung zwischen ihnen besteht. Die Programme kommunizieren miteinander durch Ablegen von Nachrichten in Message-Queues und durch Herunternehmen der Nachrichten von den Message-Queues.

- **Zeit-unabhängige (asynchrone) Kommunikation**
- **Verbindungslose Kommunikation**

Messaging bedeutet, dass Programme durch Senden von Daten in Nachrichten (Messages) kommunizieren und nicht durch wechselseitiges, direktes Aufrufen.

Message Based Queuing (MBQ)
Message Oriented Middleware (MOM)

Queuing heißt, dass Programme über Queues miteinander kommunizieren. Dadurch ist es nicht notwendig, dass diese Programme zeitlich parallel ausgeführt werden.

Eine Queue bezeichnet eine Datenstruktur, die Nachrichten speichert. Auf einer Queue können Anwendungen oder ein Queue-Manager Nachrichten ablegen. Queues existieren unabhängig von den Anwendungen, die Queues benutzen. Als Speichermedien für eine Queue kommen Hauptspeicher (wenn sie temporär ist), Platte bzw. ähnliche Zusatzspeicher oder beides in Frage.

Beim asynchronen Messaging führt das sendende Programm seine eigene Verarbeitung weiter aus, ohne auf eine Antwort seiner Message zu warten. Im Gegensatz dazu wartet beim synchronen Messaging der sendende Prozeß auf eine Antwort, bevor er seine Verarbeitung fortsetzt.

Bei einer Anwendung ist es nicht erforderlich, dass sich der Programmierer um das Ziel-Programm kümmert. Es ist belanglos, ob der Server momentan nicht im Betrieb ist oder keine Verbindung zu ihm besteht.

Message

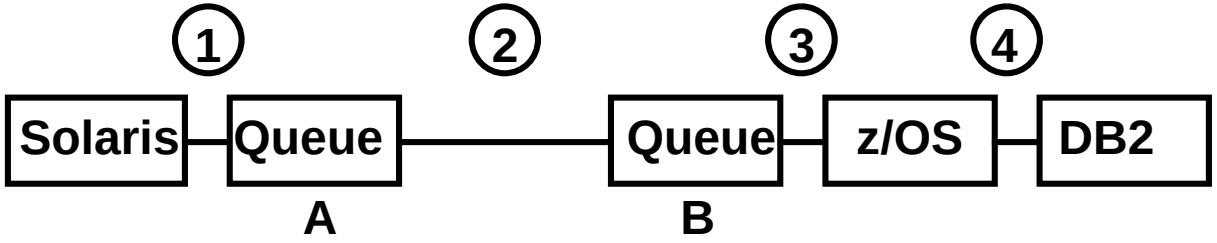
Message besteht aus zwei Teilen:

- **Message-Deskriptor oder Message-Header**
- **Daten, die vom einem Programm zu einem anderen gesendet werden**

Der Message-Deskriptor identifiziert die Message (Message-ID) und enthält Steuerinformationen (Attribute), wie z.B. Message-Type, Zeit-Ablauf, Korrelations-ID, Priorität und Namen der Antwort-Queue.

Daten können sein: Übertragung von Datenbank Reports, Transaktionen, Audio, Video.

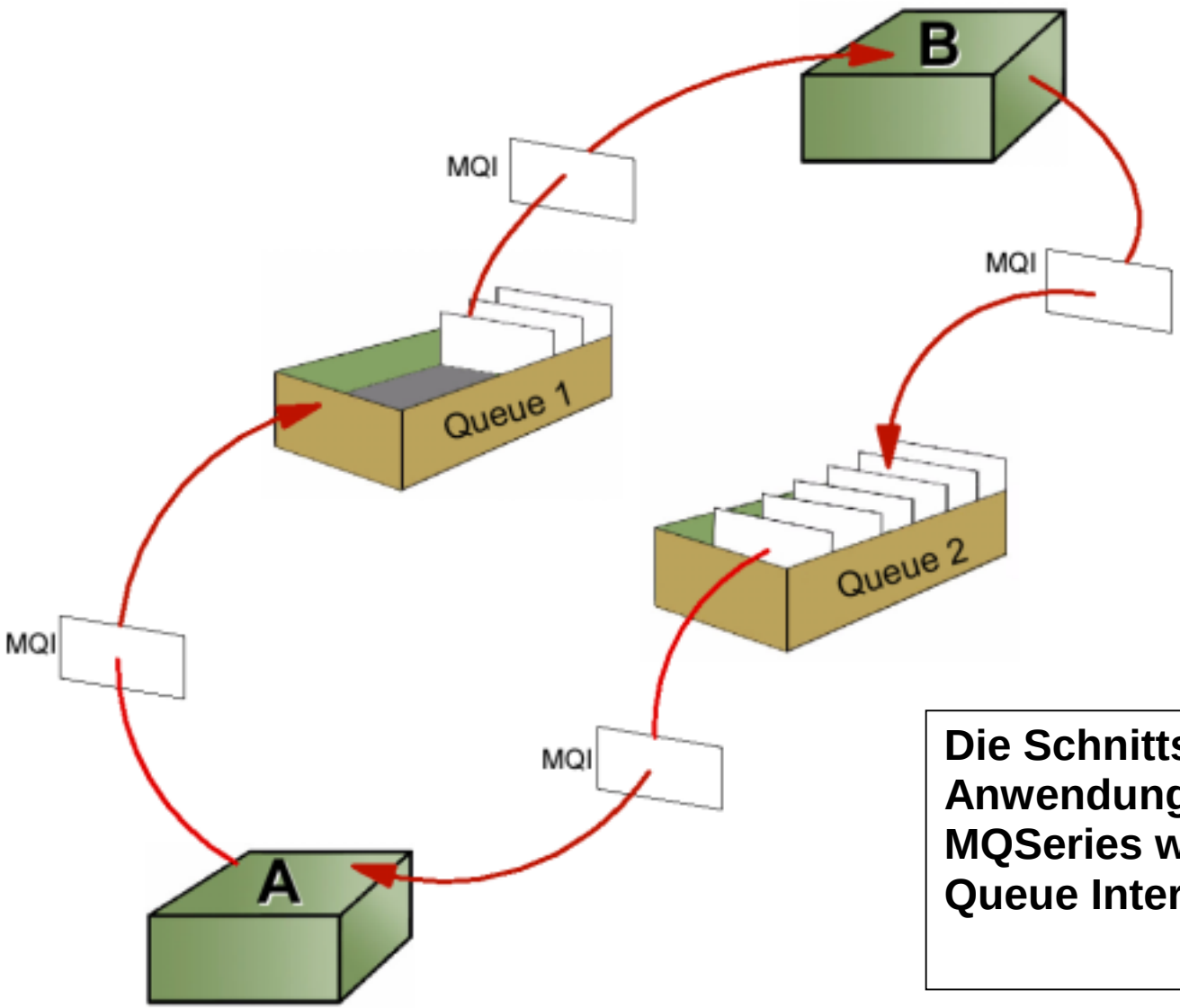
MQSeries Arbeitsweise



Schritte 1 - 3 : Wenn erfolgreich, Antwort senden

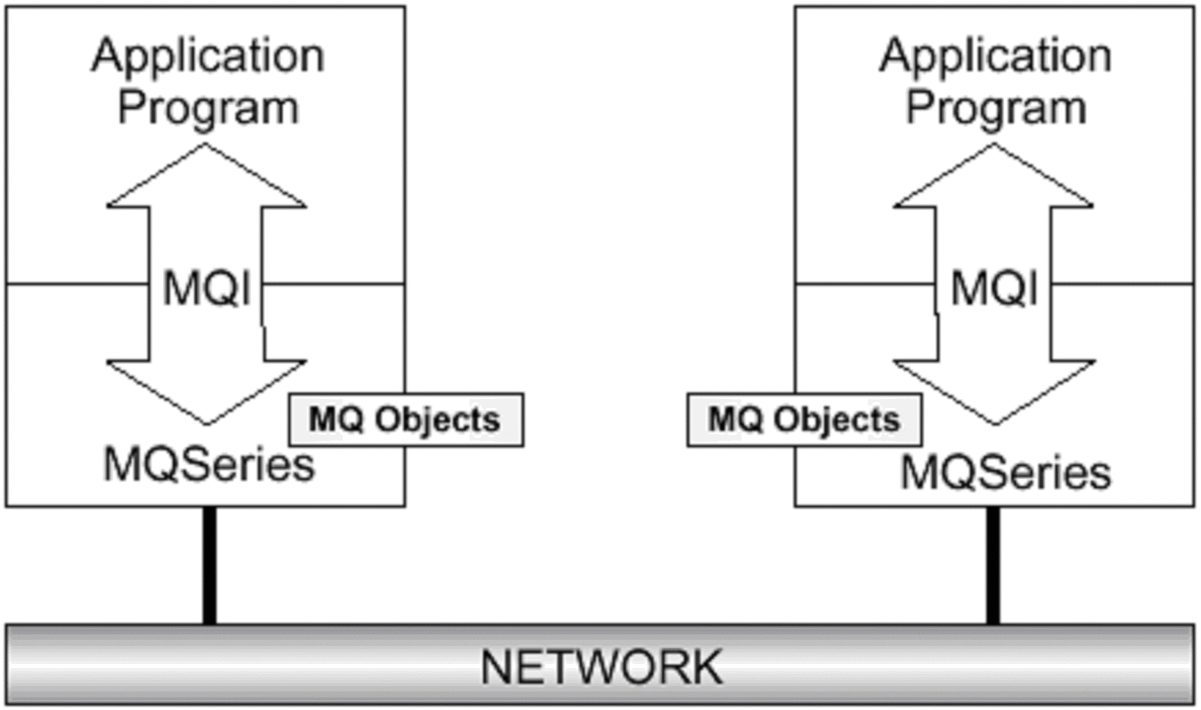
Step 4: Wenn z/OS nach DB2 versagt: roll back nach Queue B, dann recover

IBM MQSeries ist Marktführer. Für praktisch alle Betriebssysteme verfügbar: AIX, DG/UX, HP-UX, Windows, OS/390, OS/400, Psion Pervasive SOD, Sequent, Sinix, Solaris, Tandem, TPF, TrueUnix, VMS/VAX.



Die Schnittstelle zwischen Anwendungsprogramm und MQSeries wird als MQI (Message Queue Interface) bezeichnet.

MQSeries



IBM MQSeries

Vier Message-Typen:

Datagram:
Eine Message enthält Informationen, auf die keine Antwort erwartet wird.
Request:
Eine Message, für die eine Antwort angefordert wird.

Reply:
Eine Antwort auf eine Request-Message

Report:
Eine Message, die einen Event beschreibt (z.B. Fehlermeldung oder Bestätigung beim Eintreffen der Nachricht)

Message-Queue-Manager

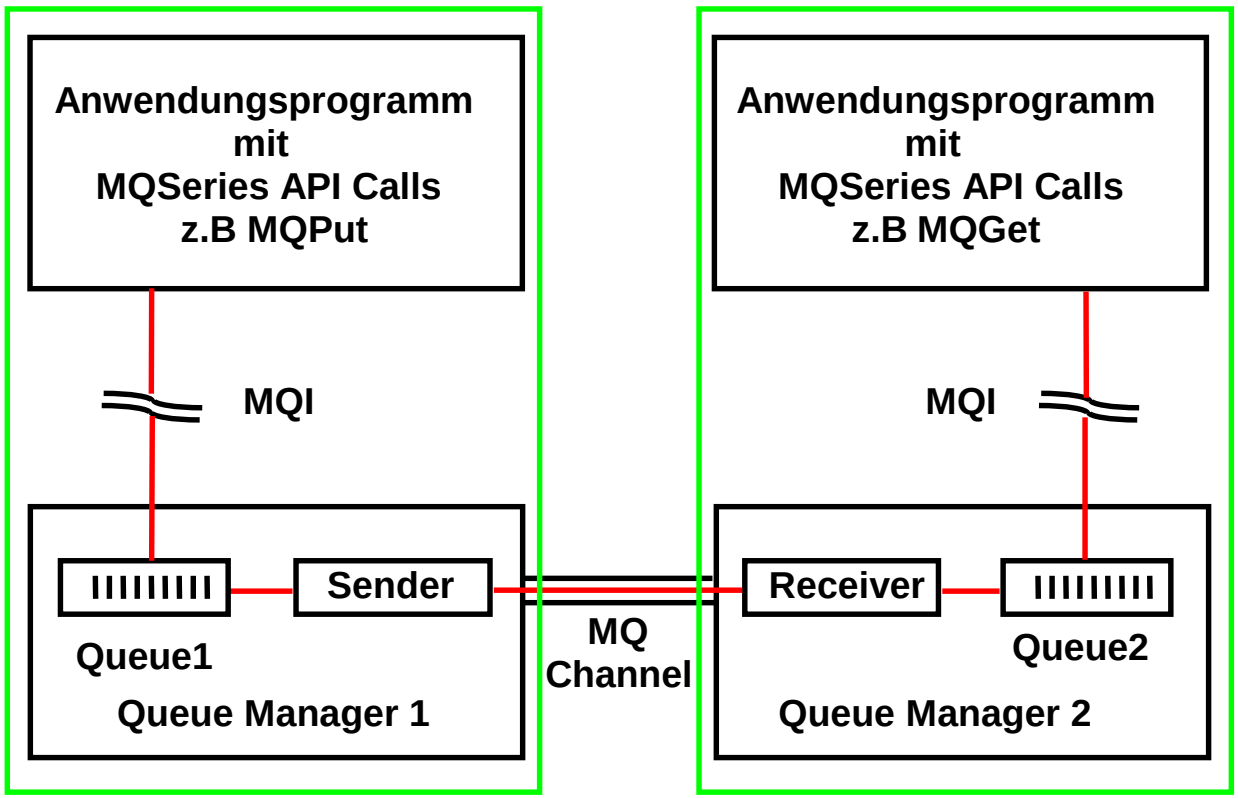
- **Message-Queue-Manager (MQSeries-Run-Time Programm) verwaltet die Queues und Messages zu verwalten.**
- **Stellt das Message-Queuing-Interface für die Kommunikation mit den Anwendungen zur Verfügung. Die Applikations-Programme rufen Funktionen des Queue-Managers durch Ausgabe von API-Call´s auf.**
- **MQPUT-API-Call z.B. legt eine Message auf eine Queue, die von einem anderen Programm mit Hilfe eines MQGET-API-Call´s gelesen werden soll.**

Message Based Queuing
Message oriented Middleware

Der Queue-Manager überträgt Nachrichten zu einem anderen Queue-Manager über Channels, wobei bestehende Netzwerk-Facilities wie TCP/IP, SNA oder SPX verwendet werden.

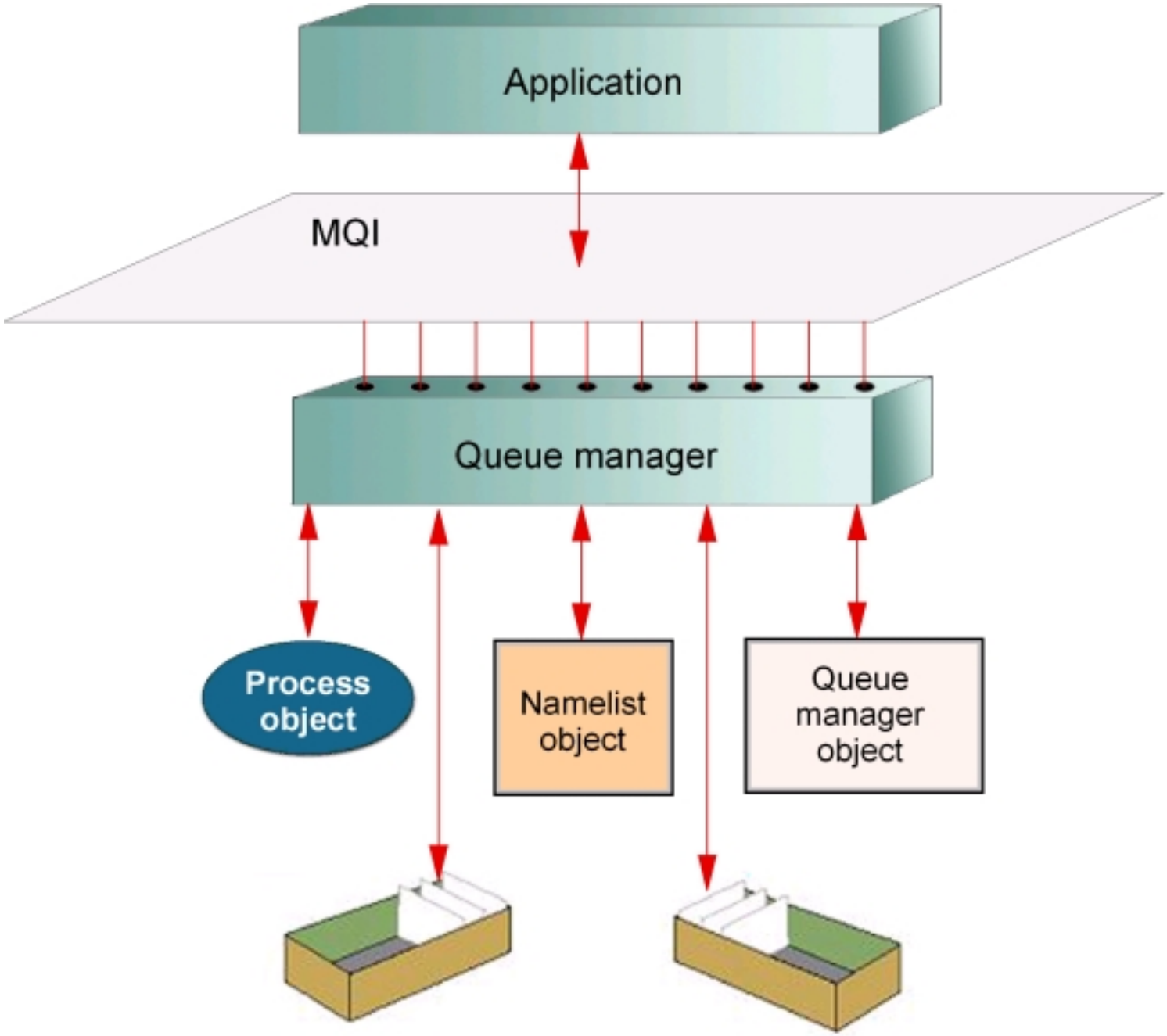
Um die Operationen des Queue-Managers zu verfolgen, können die MQSeries-Instrumentations-Events benutzt werden. Diese Events generieren spezielle Nachrichten (Event Messages)

Beim Eintreffen auf einer Queue können die Nachrichten (Messages) automatisch eine Anwendung starten. Dabei wird ein Mechanismus benutzt, der als Triggering bezeichnet wird. Die Anwendungen können wenn notwendig auch in Abhängigkeit von dem Verarbeitungszustand der Nachrichten gestoppt werden.



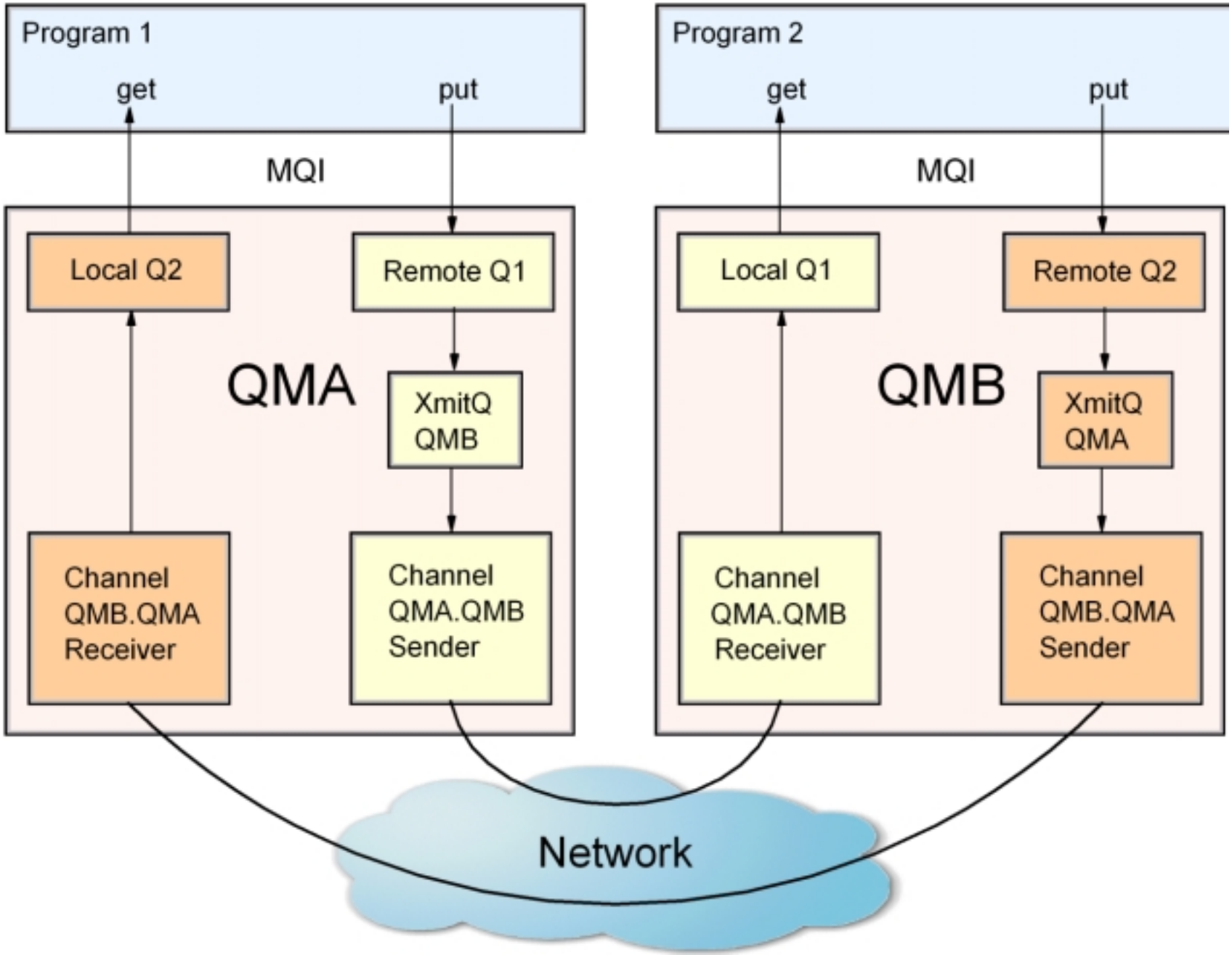
MQSeries Arbeitsweise

MQI Message Queue Interface



- Major calls
  - MQCONN
  - MQCONNX
  - MQOPEN
  - MQCLOSE
  - MQPUT
  - MQPUT1
  - MQGET

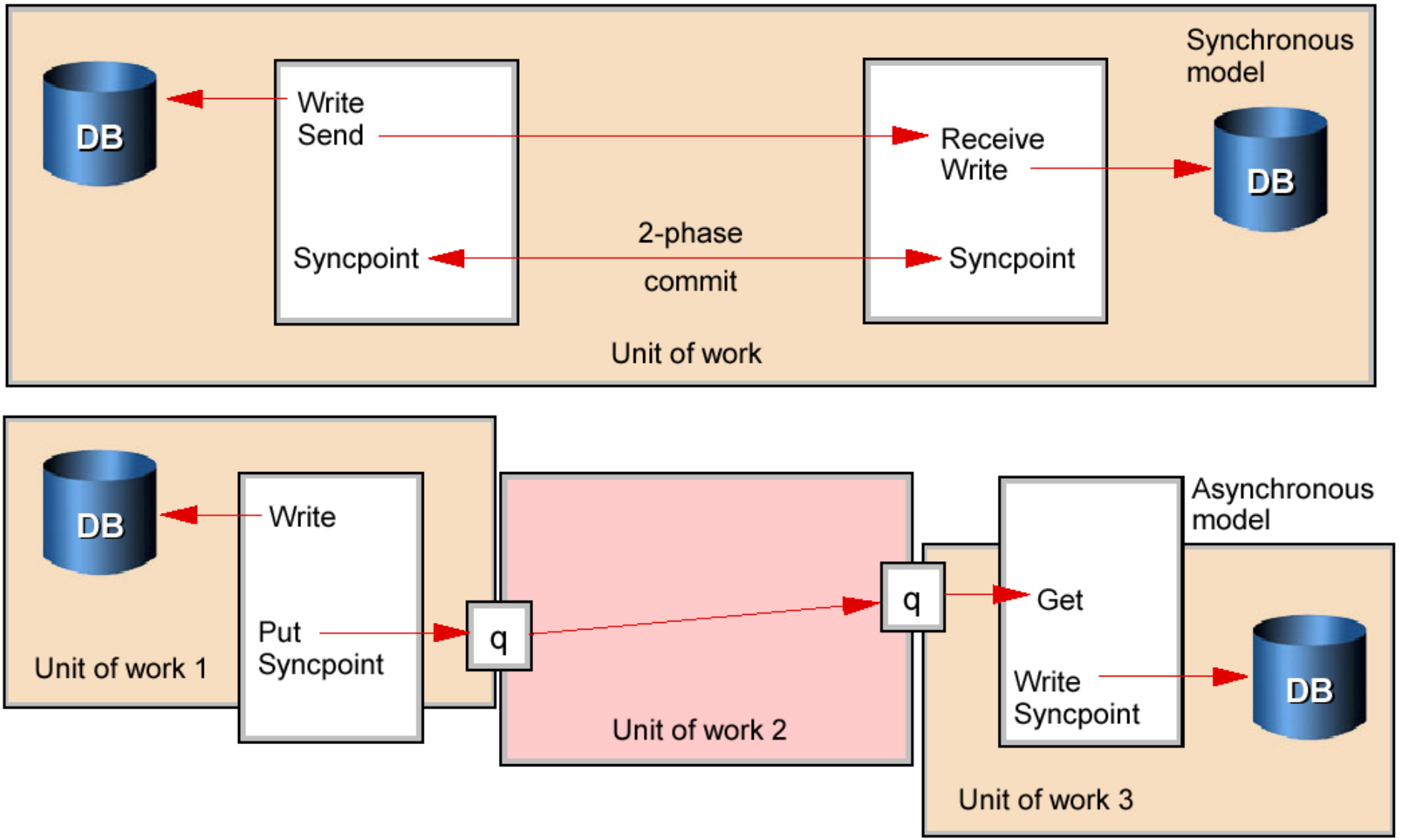
- Minor calls
  - MQBEGIN
  - MQCMIT
  - MQBACK
  - MQINQ
  - MQSET


Program 1
get
put
MQI
Local Q2
Remote Q1
QMA
XmitQ
QMB
Channel
QMB.QMA
Receiver
Channel
QMA.QMB
Sender
Program 2
get
put
MQI
Local Q1
Remote Q2
QMB
XmitQ
QMA
Channel
QMA.QMB
Receiver
Channel
QMB.QMA
Sender
Network

MQSeries Arbeitsweise

Das Anwendungsprogramm auf Rechner 1 stellt mit Hilfe eines *Message Queue Interface* (MQI) API Aufrufs (z.B. MQPut) eine Nachricht in Queue 1 . Die als *Queue Manager* (QM) bezeichnete MQSeries Laufzeitumgebung (Runtime) bewirkt unter ACID Bedingungen den Transport der Nachricht nach Queue 2 auf Rechner 2. Der Transport erfolgt über einen *MQ-Channel* (Message Queue Channel). Hierfür unterhält Queue Manager 1 einen *Channel Sender* und Queue Manager 2 einen *Channel Receiver* (auch als Message channel Agent bezeichnet). Queues, Queue Manager, Channel Sender/Receiver und Channels repräsentieren Schicht 5 Middleware. Der Channel kann, z.B. mit Hilfe von Sockets, in der Schicht 4 das TCP/IP Protokoll verwenden.

Das Anwendungsprogrogramm auf Rechner 2 kann nun asynchron mit Hilfe eines MQI Aufrufs MQGet die Nachricht aus Queue 2 auslesen. Evtl. unterhält Queue Manager 2 eine Trigger Funktion, welche die Anwendung beim Eintreffen einer Nachricht informiert.



MQSeries Arbeitsweise

Queue manager

The component of software that owns and manages queues is called a queue manager (QM). In WebSphere MQ, the message queue manager is called the MQM, and it provides messaging services for applications, ensures that messages are put in the correct queue, routes messages to other queue managers, and processes messages through a common programming interface called the Message Queue Interface (MQI).

The queue manager can retain messages for future processing in the event of application or system outages. Messages are retained in a queue until a successful completion response is received through the MQI. There are similarities between queue managers and database managers. Queue managers own and control queues similar to the way that database managers own and control their data storage objects. They provide a programming interface to access data, and also provide security, authorization, recovery and administrative facilities.

There are also important differences, however. Databases are designed to provide long-time data storage with sophisticated search mechanisms, whereas queues are not designed for this. A message on a queue generally indicates that a business process is incomplete; it might represent an unsatisfied request, anunprocessed reply, or an unread report. Figure 5-4 on page 105 shows the flow of activity in queue managers and database managers.

Types of message queues

Several types of message queues exist. In this text, the most relevant are the following:

Local queue

A queue is local if it is owned by the queue manager to which the application program is connected. It is used to store messages for programs that use the same queue manager. The application program doesn't have to run on the same machine as the queue manager.

Remote queue

A queue is remote if it is owned by a different queue manager. A remote queue is not a real queue; it is only the *definition* of a remote queue to the local queue manager. Programs cannot read messages from remote queues. Remote queues are associated with a transmission queue.

Transmission queue

This local queue has a special purpose: it is used as an intermediate step when sending messages to queues that are owned by a different queue manager. Transmission queues are transparent to the application; that is, they are used internally by the queue manager initiation queue.

This is a local queue to which the queue manager writes (transparently to the programmer) a trigger message when certain conditions are met on another local queue, for example, when a message is put into an empty message queue or in a transmission queue. Two WebSphere MQ applications monitor initiation queues and read trigger messages, the trigger monitor and the channel initiator. The *trigger manager* can start applications, depending on the message. The *channel initiator* starts the transmission between queue managers.

Dead-letter queue

A queue manager (QM) must be able to handle situations when it cannot deliver a message, for example:

- **Destination queue is full.**
- **Destination queue does not exist.**
- **Message puts have been inhibited on the destination queue.**
- **Sender is not authorized to use the destination queue.**
- **Message is too large.**
- **Message contains a duplicate message sequence number.**

When one of these conditions occurs, the message is written to the dead-letter queue. This queue is defined when the queue manager is created, and each QM should have one. It is used as a repository for all messages that cannot be delivered.

What is a channel?

A *channel* is a logical communication link. The conversational style of program-to-program communication requires the existence of a communications connection between each pair of communicating applications. Channels shield applications from the underlying communications protocols. WebSphere MQ uses two kinds of channels:

Message channel

A message channel connects two queue managers through message channel agents (MCAs). A message channel is unidirectional, comprised of two message channel agents (a sender and a receiver) and a communication protocol. An MCA transfers messages from a transmission queue to a communication link, and from a communication link to a target queue. For bidirectional communication, it is necessary to define a *pair* of channels, consisting of a sender and a receiver.

MQI channel

An MQI channel connects a WebSphere MQ client to a queue manager. Clients do not have a queue manager of their own. An MQI channel is bidirectional.

How transactional integrity is ensured

A business might require two or more distributed databases to be maintained in step. WebSphere MQ offers a solution involving multiple units of work acting asynchronously, as shown in Figure 5-3.

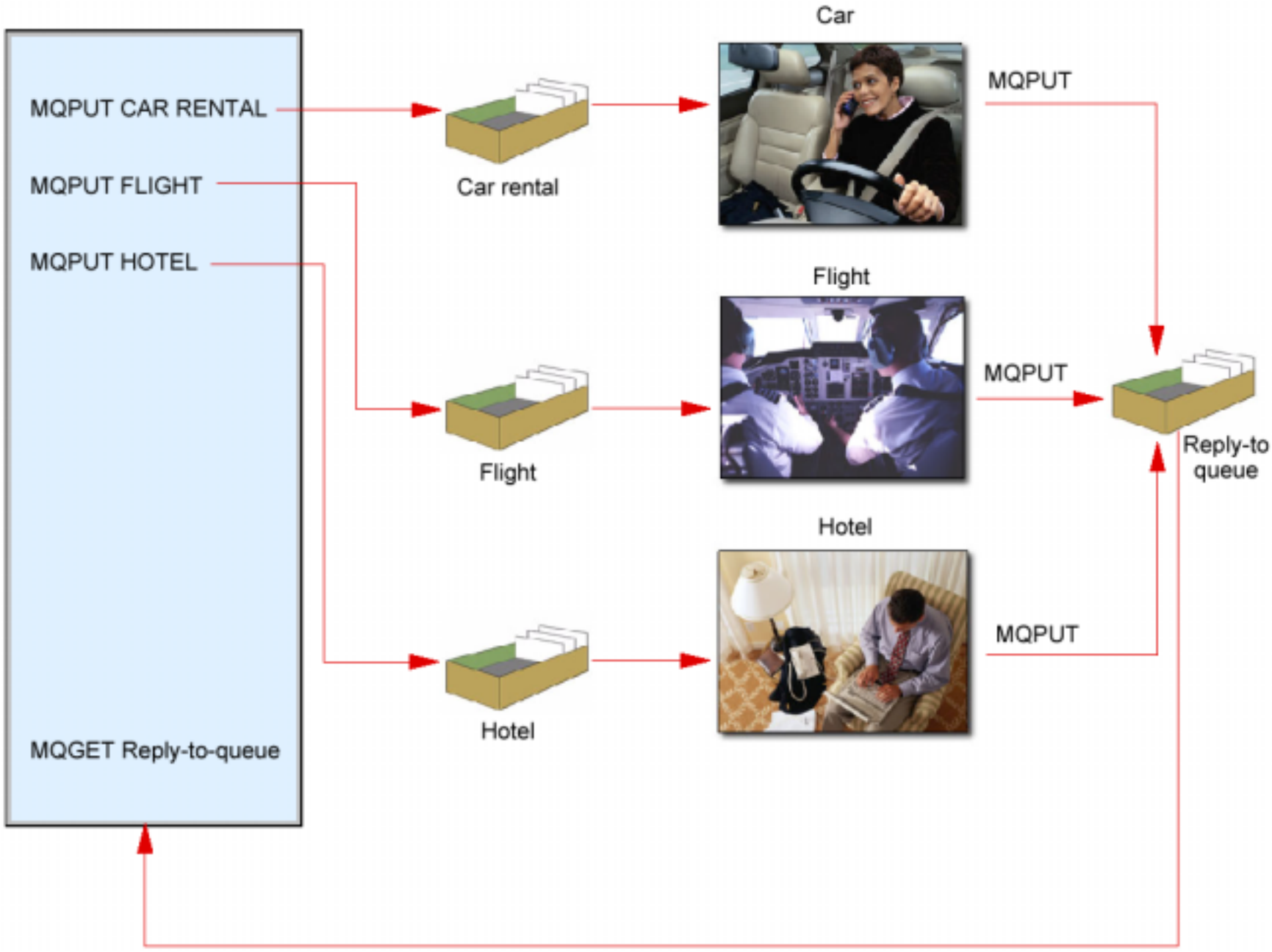
The top half of the Figure shows a two-phase commit structure, while the WebSphere MQ solution is shown in the lower half, as follows:

- **The first application writes to a database, places a message on a queue, and issues a syncpoint to commit the changes to the two resources. The message contains data which is to be used to update a second database on a separate system. Because the queue is a remote queue, the message gets no further than the transmission queue within this unit of work. When the unit of work is committed, the message becomes available for retrieval by the sending MCA.**

- **In the second unit of work, the sending MCA gets the message from the transmission queue and sends it to the receiving MCA on the system with the second database, and the receiving MCA places the message on the destination queue. This is performed reliably because of the assured delivery property of WebSphere MQ. When this unit of work is committed, the message becomes available for retrieval by the second application.**

- **In the third unit of work, the second application gets the message from the destination queue and updates the database using the data contained in the message.**

- **It is the transactional integrity of units of work 1 and 3, and the once and once only, assured delivery property of WebSphere MQ used in unit of work 2, which ensures the integrity of the complete business transaction. If the business transaction is a more complex one, many units of work may be involved.**

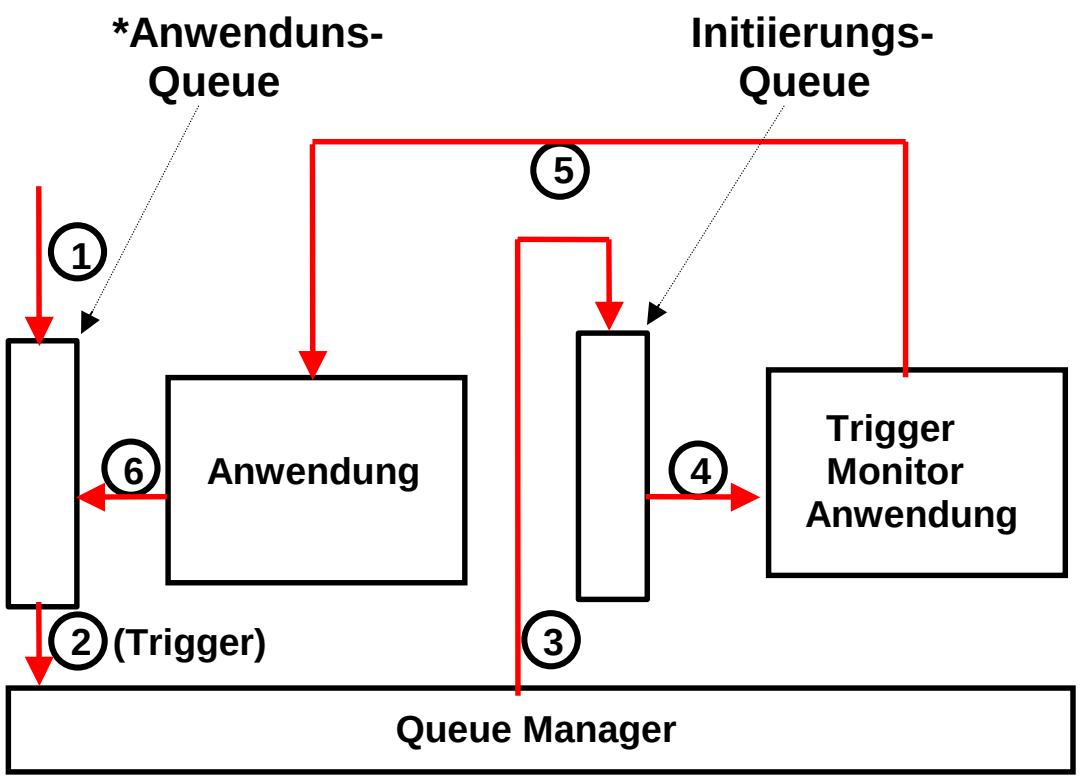

MQPUT CAR RENTAL
MQPUT FLIGHT
MQPUT HOTEL
MQGET Reply-to-queue
Car rental
Flight
Hotel
Car
Flight
Hotel
MQPUT
MQPUT
MQPUT
Reply-to queue

Example of messaging and queuing

Let us assume a travel agency to see how messaging facilities play a role in booking a vacation. Assume that the travel agent must reserve a flight, a hotel room, and a rental car. All of these reservations must succeed before the overall business transaction can be considered complete

With a message queue manager such as WebSphere MQ, the application can send several requests at once; it need not wait for a reply to one request before sending the next. A message is placed on each of three queues, serving the flight reservations application, the hotel reservations application, and the car rental application. Each application can then perform its respective task in parallel with the other two and place a reply message on the reply-to queue. The agent's application waits for these replies and produces a consolidated answer for the travel agent.

Designing the system in this way can improve the overall response time. Although the application might normally process the replies only when they have all been received, the program logic may also specify what to do when only a partial set of replies is received within a given period of time.

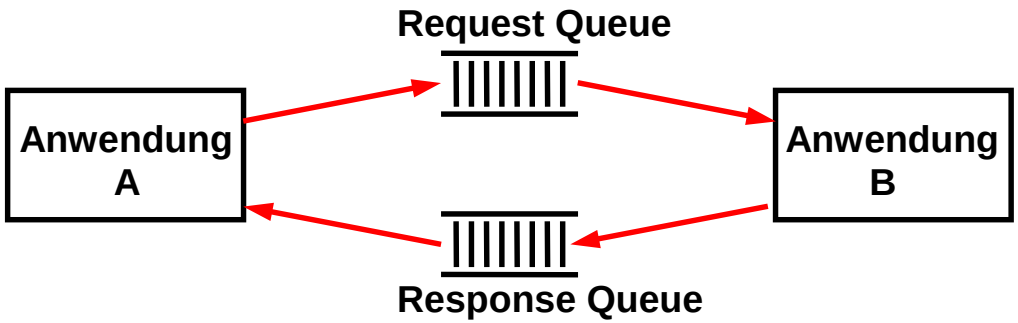


Trigger Nachrichten

Als Triggering wird der Mechanismus bezeichnet, mit Hilfe dessen eine Anwendung bei Bedarf gestartet werden kann. Der Queue Manager erkennt die Ankunft neuer Nachrichten (für die Triggering enabled ist). Er benachrichtigt eine spezielle Anwendung, den Trigger Monitor. Der Trigger Monitor startet die Anwendung, die für die Nachricht zuständig ist.

Da mehrere Nachrichten gleichzeitig eintreffen können, werden die Trigger in einer separaten Queue, der Initiierungsqueue, zwischen-gespeichert. Die Trigger Nachricht enthält die „Prozess Definition“ - Information, mit deren Hilfe der Trigger Monitor die gewünschte Anwendung starten kann.

MQSeries Betrieb



Nachrichtenkopf enthält: „Reply to ....“

Asynchroner Betrieb: Request/reply Modus kann synchronen Betrieb simulieren. Wird häufig so benutzt.

MQ-CICS Bridge: Receiving queue lädt Nachricht in die CICS Input Queue. CICS behandelt es wie eine 3270 Nachricht, gibt eine 3270 Nachricht zurück. Response Queue B übersetzt Nachricht in das MQSeries Format.

Keine Daten Konvertierung, nur binäre Daten werden übertragen.

Firmen wenden 40 % ihres eigenen Budgets für Software Entwicklung für Integrationsaufgaben.

Wird Software gekauft, kostet die Integration das 9fache des Kaufpreises.

MQSeries Kommandos

Jede Nachricht verfügt über einen eindeutigen 24 Byte Indentifier

Die MQI API wird über 10 Kommandos realisiert („Verben“ im IBM-Sprachgebrauch)

MQCONN, MQDISC	**Verbindung zu Queue Manager herstellen und löschen. Anwendungen „melden sich an“.**
MQOPEN, MQCLOSE	**Verbindung zu einer spezifischen Queue aufbauen und lösen.**
MQGET, MQPUT	
MQCMT, MQBACK	**für Transaktionverarbeitung**
MQINQ	**Status einer Queue abfragen**
MQSET	**spezifische Queue Parameter setzen**

MQSeries API
MQI

MQCONN stellt eine Verbindung mit einem Queue-Manager mit Hilfe von Standard-Links her.

MQBEGIN startet eine Arbeitseinheit, die durch den Queue-Manager koordiniert wird und externe XA-kompatible Ressource-Manager enthalten kann. Dieses API ist mit MQSeries Version 5 eingeführt worden. Es wird für die Koordinierung der Transaktionen , die Queues (MQPUT und MQGET unter Syncpoint-Bedingung) und Datenbank-Updates (SQL-Kommandos) verwenden.

MQGET

MQPUT

MQPUT1 öffnet eine Queue, legt eine Message darauf ab und schließt die Queue wieder. Dieser API-Call stellt eine Kombination von MQOPEN, MQPUT und MQCLOSE dar.

MQINQ fordert Informationen über den Queue-Manager oder über eines seiner Objekte an, wie z.B. die Anzahl der Nachrichten in einer Queue.

MQSET verändert einige Attribute eines Objekts.

MQCMIT gibt an, dass ein Syncpoint erreicht worden ist.

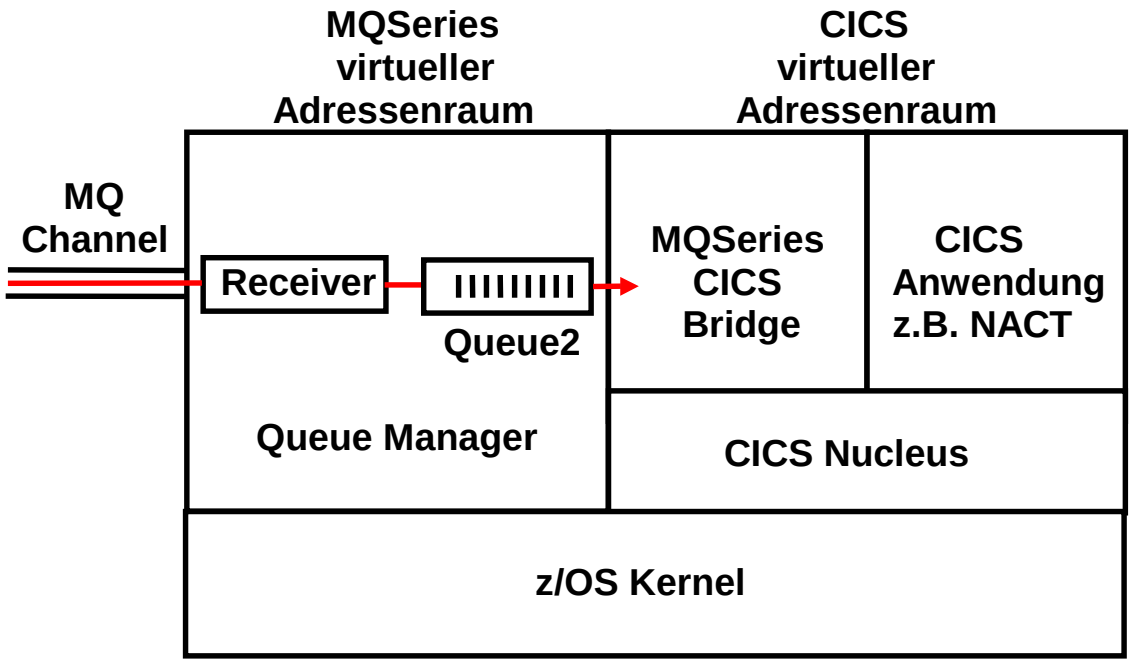
MQBACK teilt dem Queue-Manager alle zurück gekommenen PUT´s- und GET´s-Nachrichten seit dem letzten Syncpoint mit.

MQDISC schließt die Übergabe einer Arbeitseinheit ein. Das Beenden eines Programms ohne Unterbrechung der Verbindung zum Queue-Manager verursacht ein "Rollback" (MQBACK).

```
MQHCONN  HCon;                  // Connection handle
MQHOBJ   HObj1;                 // Object handle for queue 1
MQHOBJ   HObj2;                 // Object handle for queue 2
MQLONG   CompCode, Reason;      // Return codes
MQLONG   options;
MQOD     od1 = {MQOD_DEFAULT};  // Object descriptor for queue 1
MQOD     od2 = {MQOD_DEFAULT};  // Object descriptor for queue 2
MQMD     md  = {MQMD_DEFAULT};  // Message descriptor
MQPMO    pmo = {MQPMO_DEFAULT}; // Put message options
MQGMO    gmo = {MQPMO_DEFAULT}; // Get message options
:
// 1 Connect application to a queue manager.
strcpy  (QMName,"MYQMGR");
MQCONN  (QMName, &HCon, &CompCode, &Reason);

// 2 Open a queue for output
strcpy  (od1.ObjectName,"QUEUE1");
MQOPEN  (HCon,&od1, MQOO_OUTPUT, &Hobj1, &CompCode, &Reason);

// 3 Put a message on the queue
MQPUT   (HCon, Hobj1, &md, &pmo, 100, &buffer, &CompCode, &Reason);

// 4 Close the output queue
MQCLOSE (HCon, &Hobj1, MQCO_NONE, &CompCode, &Reason);

// 5 Open input queue
options = MQOO_INPUT_AS_Q_DEF;
strcpy  (od2.ObjectName, "QUEUE2");
MQOPEN  (HCon, &od2, options, &Hobj2, &CompCode, &Reason);

// 6 Get message
gmo.Options = MQGMO_NO_WAIT;
buflen  = sizeof(buffer - 1);
memcpy (md.MsgId, MQMI_NONE, sizeof(md.MsgId);
memset (md.CorrelId, 0x00, sizeof(MQBYTE24));
MQGET (HCon, Hobj2, &md, &gmo, buflen, buffer, 100, &CompCode, &Reason);

// 7 Close the input queue
options = 0;
MQCLOSE (HCon, &Hobj2,options, &CompCode, &Reason);

// 8 Disconnect from queue manager
MQDISC  (HCon, &CompCode, &Reason);
```

Figure 21. A Code Fragment

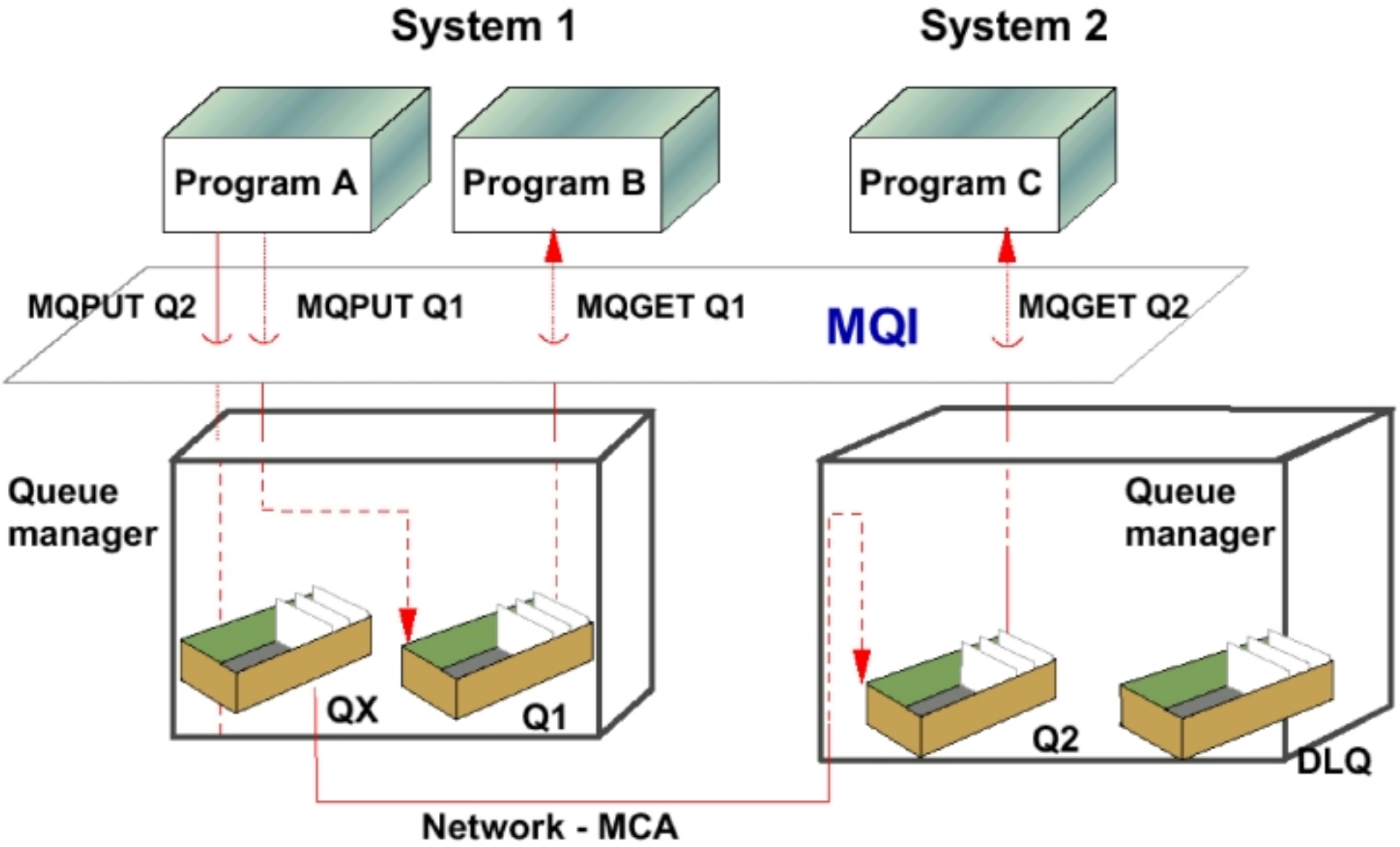
MQ Series Code Fragment



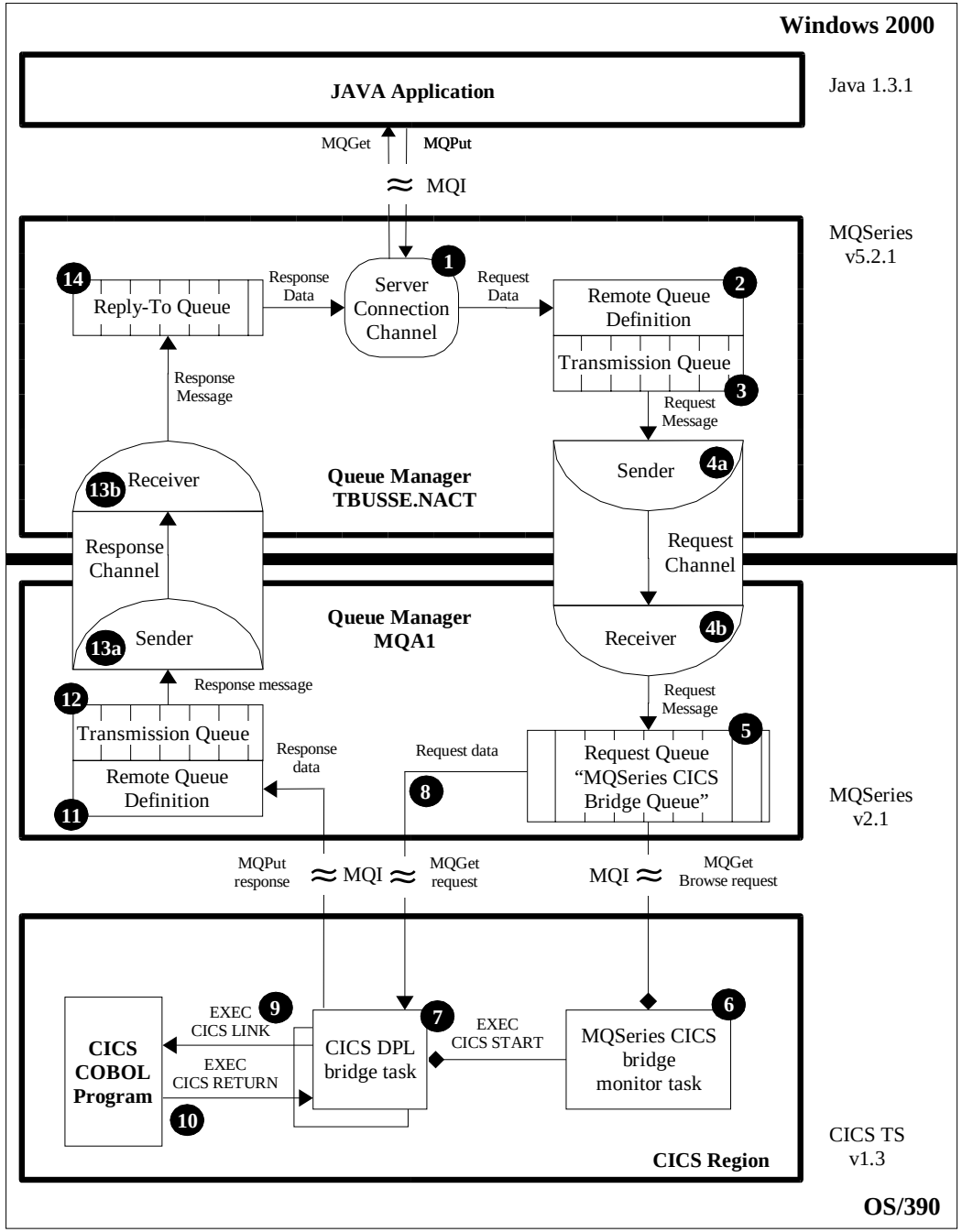
MQSeries Server
Anschluss an CICS

Beispiel: MQSeries Zugriff auf die COMMAREA der NACT Anwendung.
Klienten-Programm implementiert Präsentations-logik in Java

Diplomarbeit Tobias Busse



Die MQSeriers Laufzeitumgebung auf einem Rechner kann mehrere Anwendungen gleichzeitig bedienen



MQSeries Zugriff auf CICS

Java Präsentationslogik greift mit Hilfe von MQSeries auf CICS COMMAREA zu

MQ Summary

Common application programming interface (MQI).

Assured delivery: messages do not get lost and they arrive only once.

No synchronous access needed.

Message driven application.

Quicker development due to shielding of the network.