Professional Leadership Technical Exchange

AP090: Integrating WebSphere and MQSeries

Ed Fletcher

Europe, Middle East, & Africa

20 - 23 May 2001

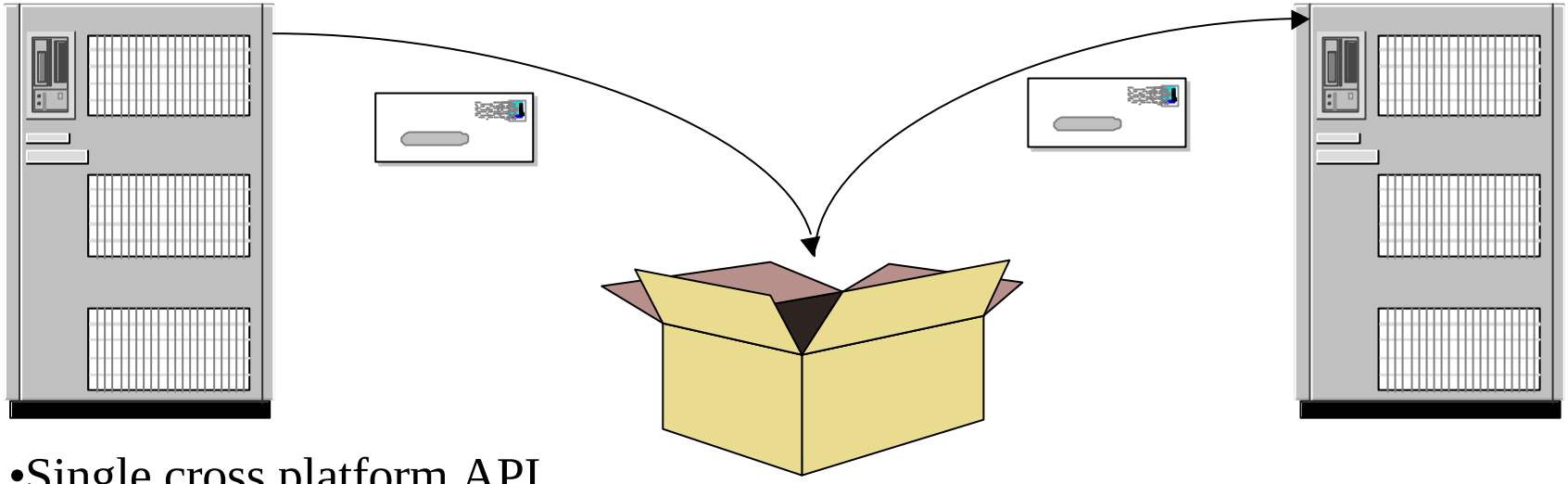
Lyon, France

Agenda

- Why integrate?
- What you can do
- How you can do it
- Integrating with legacy systems
- What's new?
- Acknowledgements and References

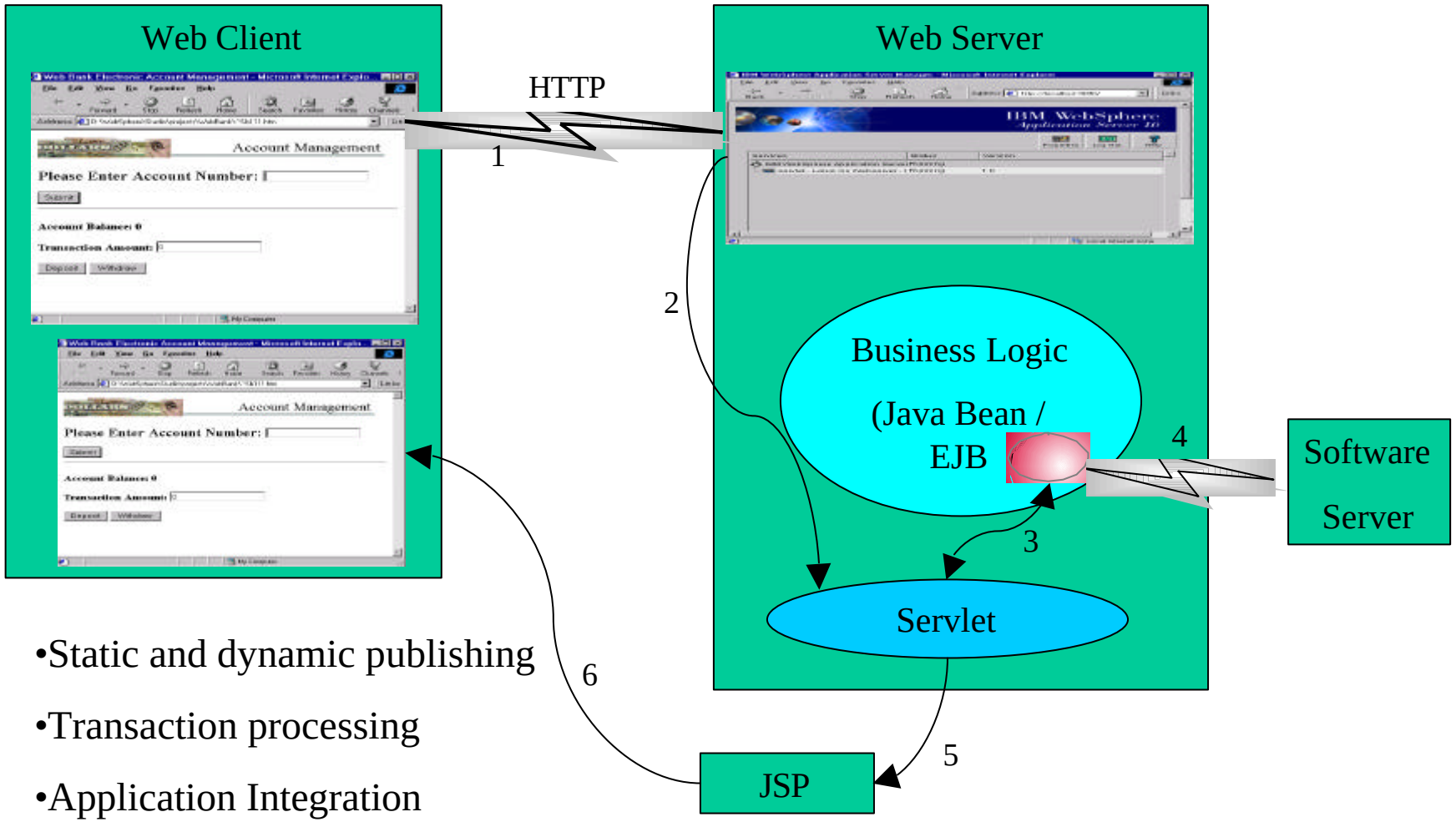
Why Integrate?

MQSeries



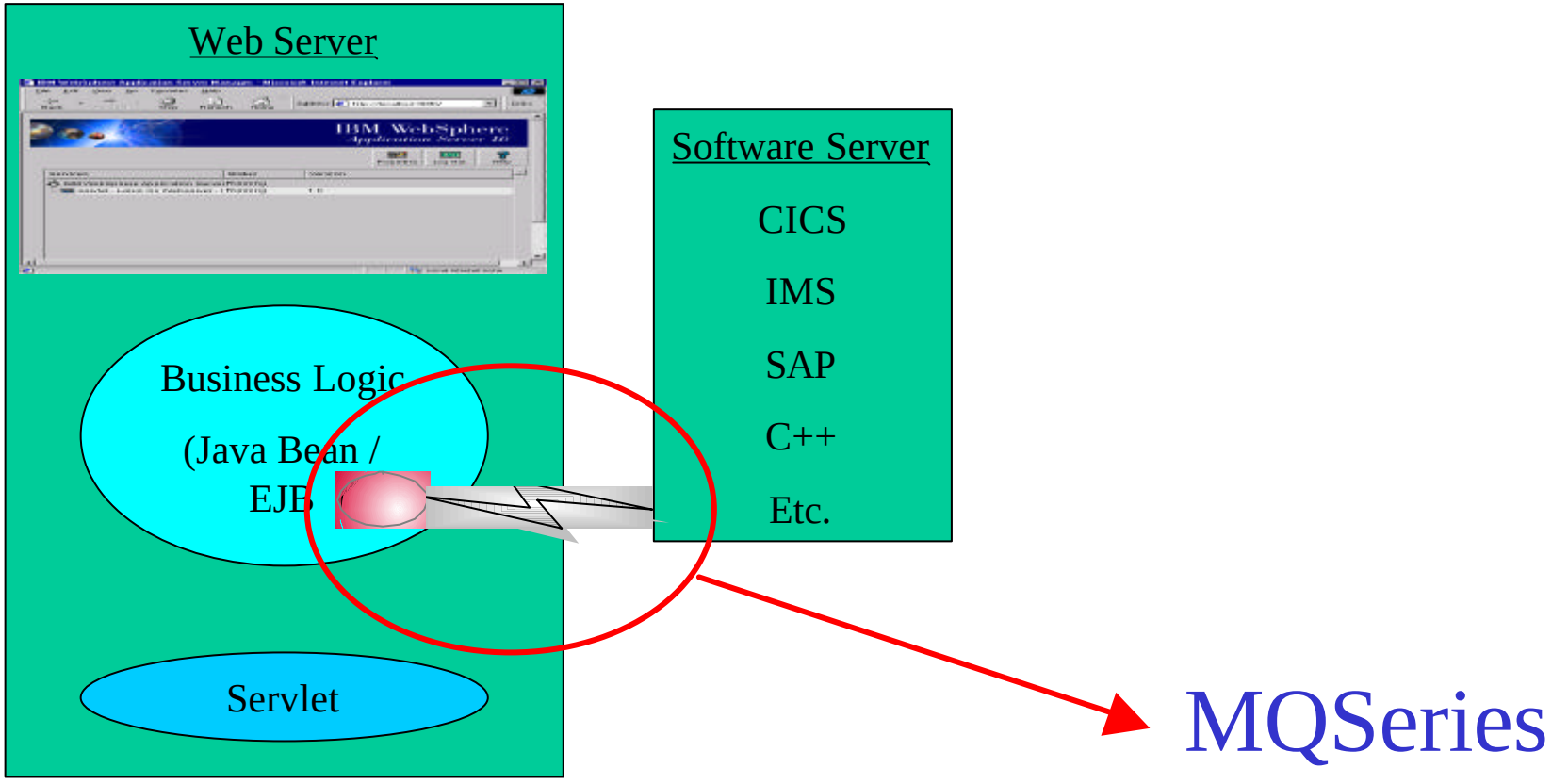
- Single cross platform API
- Overlapped (parallel) operations
- Freedom from partner program availability
- Assured message delivery and message recovery
- Application location transparency
- Easier cross-platform communication
- Communication protocol independence
- Fewer network connections

WebSphere Application Server

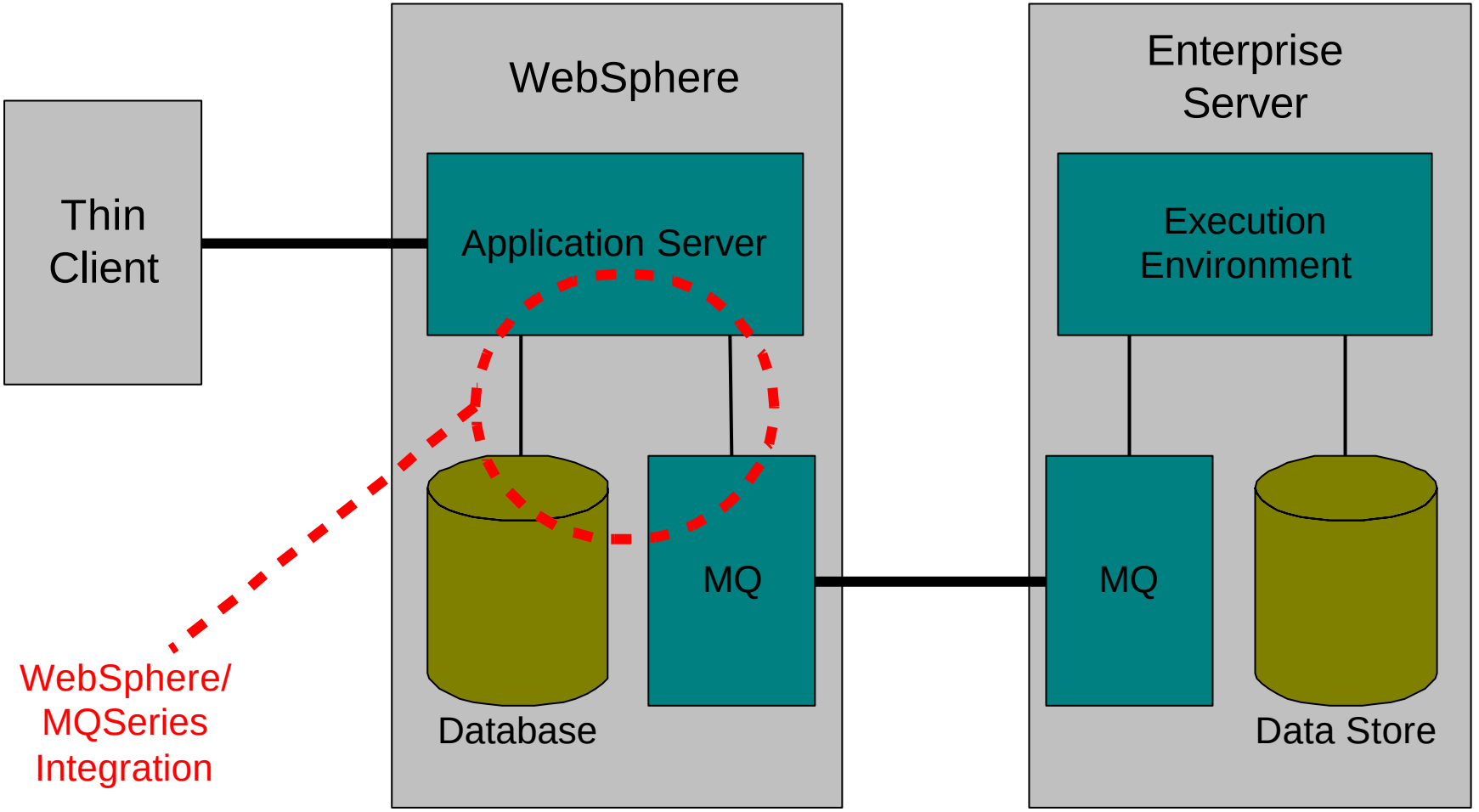


- Static and dynamic publishing
- Transaction processing
- Application Integration

EJB to Software Server



WAS Meets MQ


WebSphere
Enterprise Server
Thin Client
Application Server
Execution Environment
MQ
MQ
Database
Data Store
WebSphere/ MQSeries Integration

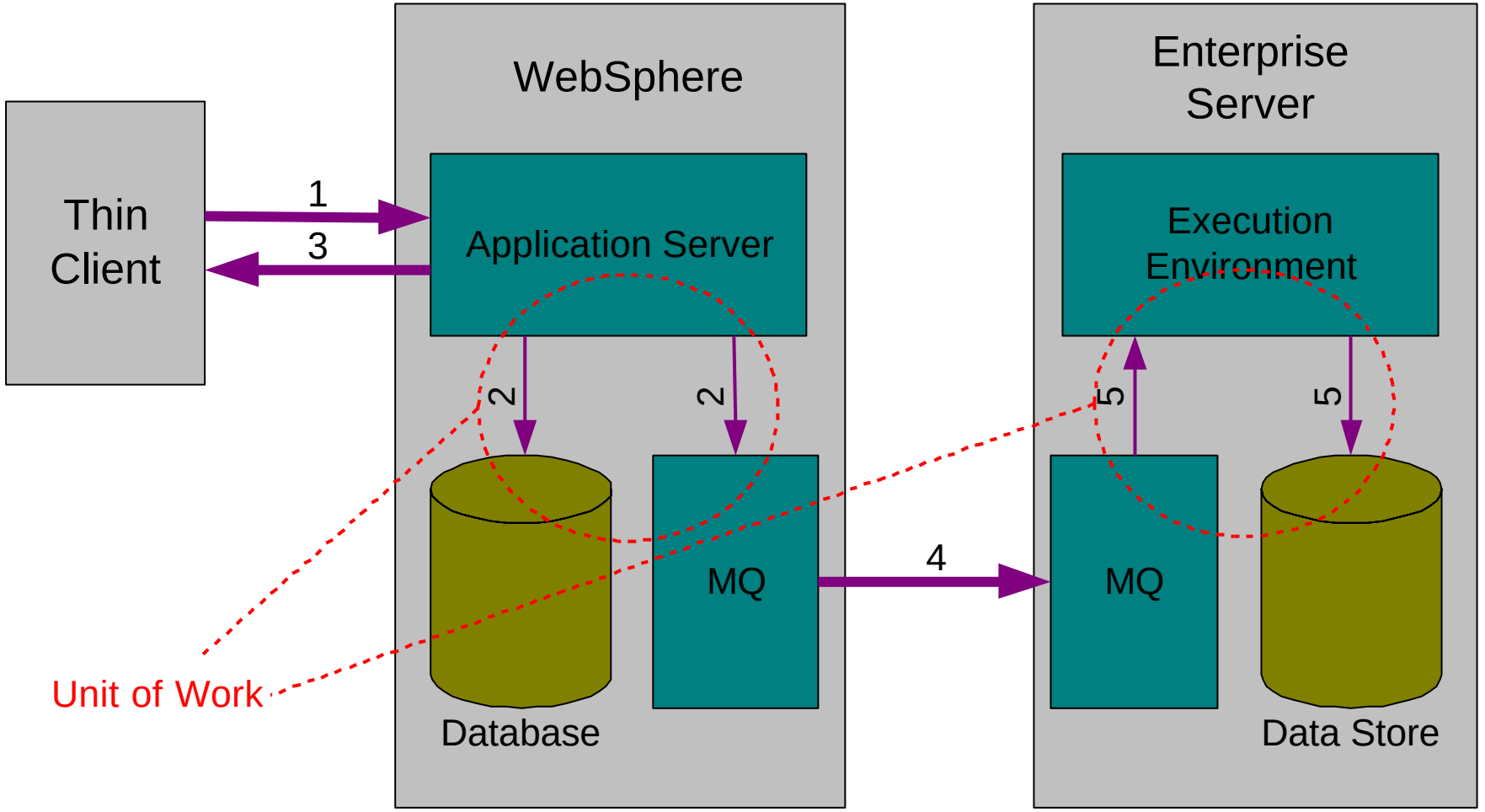
Agenda

- Why integrate?
- What you can do
- How you can do it
- Integrating with legacy systems
- What's new?
- Acknowledgements and References

WAS Sends Datagram

- E.g. register for online banking, submit order
- Usually persistent
  - It may be non-persistent, e.g. request statement by post
- WebSphere state says what the user did
  - “We have received your request and it is being processed”
- Requires unit of work to include WebSphere and MQSeries resources

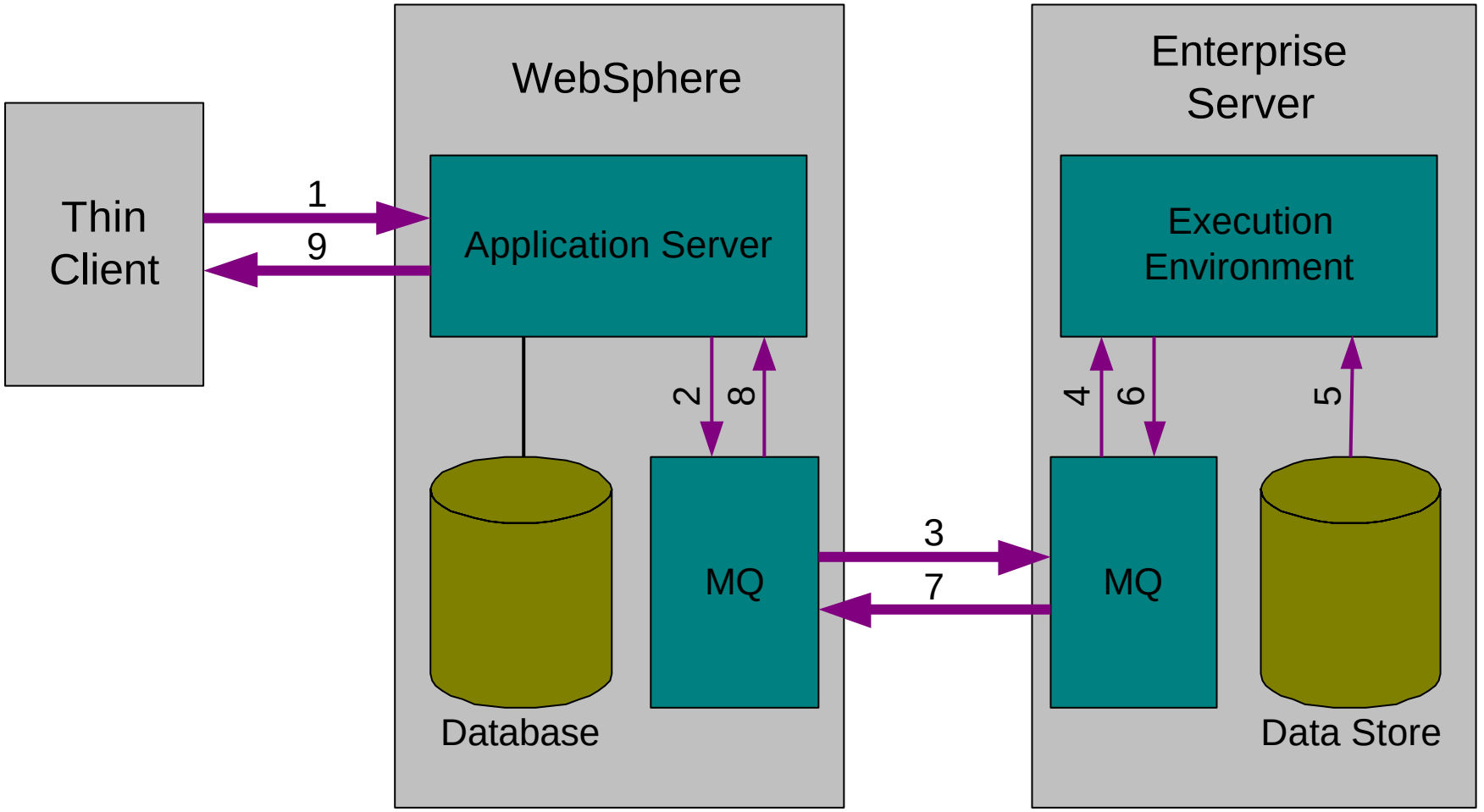
WAS Sends Datagram



WAS Sends Request/Reply (Enquiry)

- E.g. Balance enquiry
- Use non-persistent messages
  - Faster throughput
- WebSphere sends request, blocking wait for reply
- Wait may (should) timeout, therefore need message expiry on reply
  - Response to user is “It failed, please try again”

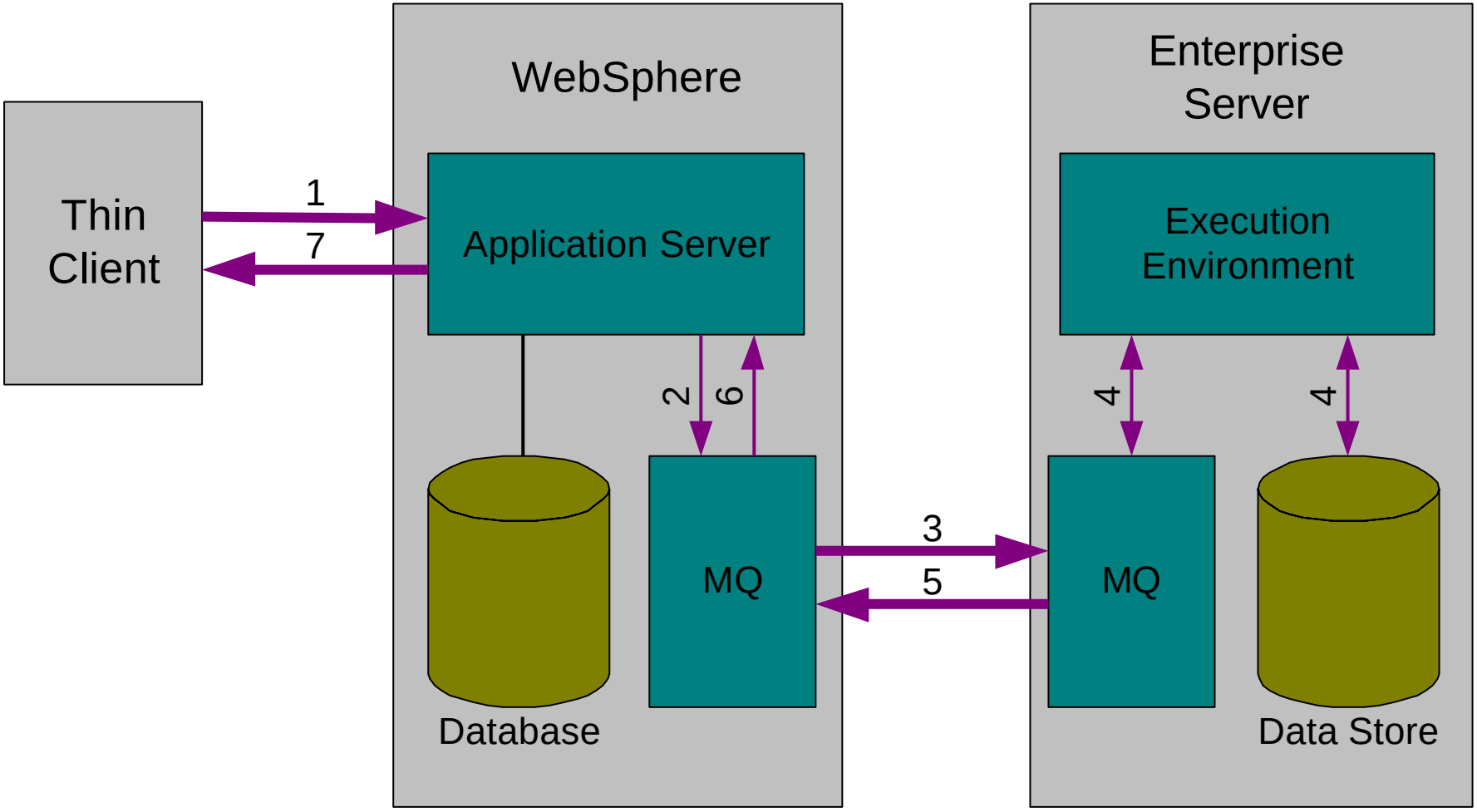
WAS Sends Request/Reply (Enquiry)



WAS Sends Request/Reply (Update)

- E.g. Intra-bank account transfer
- Perform update, expect immediate confirmation
- Can use persistent messages, but…
  - MQGet for reply message may time out, even though back-end operation succeeded
    - What do you tell the user when web app doesn't receive the reply?
- Web User Interface is synchronous, MQSeries is asynchronous

WAS Sends Request/Reply (Update)



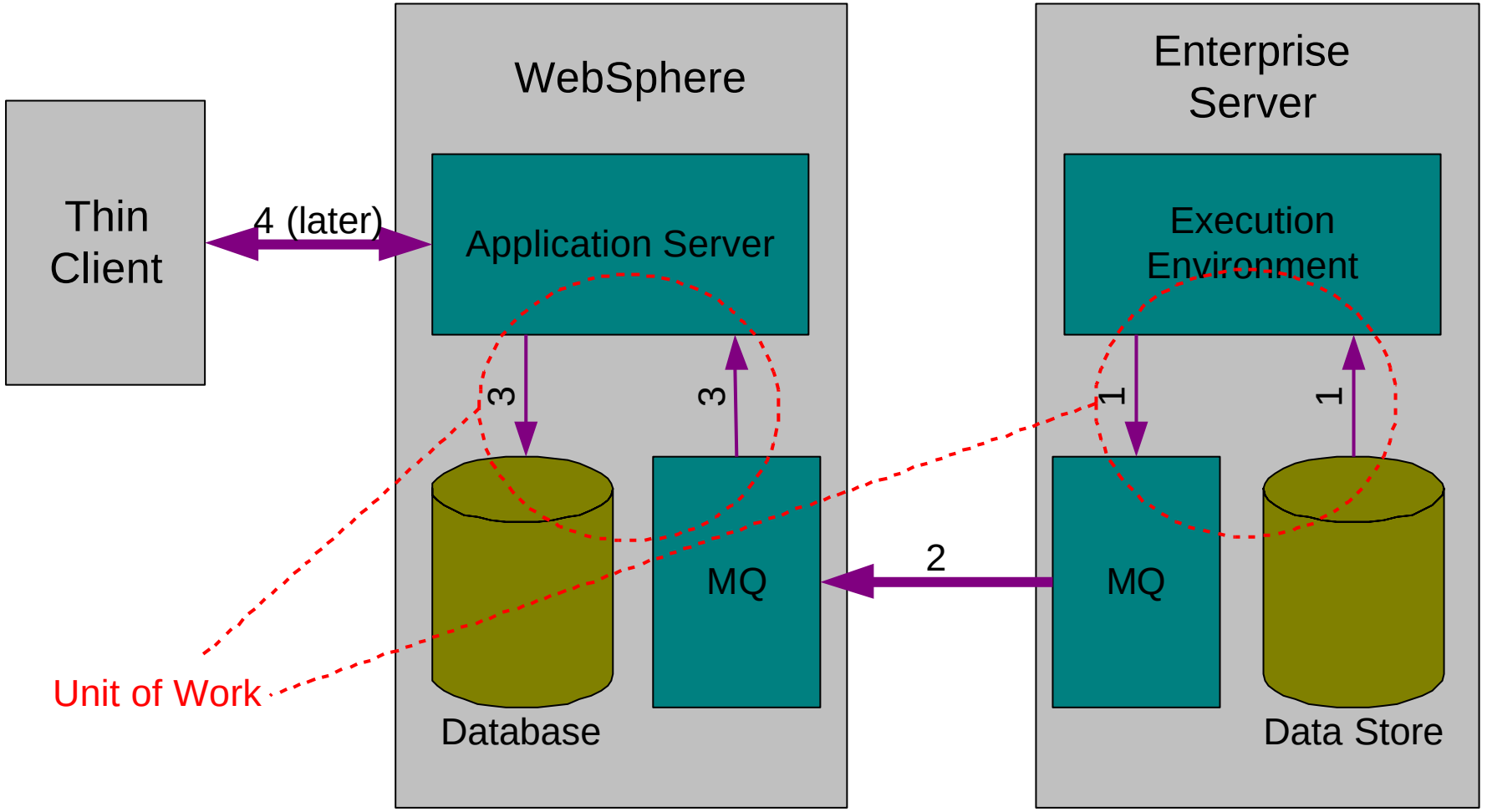
Persistent Request/Reply: Alternatives

- Can perform appropriate validation first
  - E.g. check account balances before attempting transfer
- Use persistent message to send request
- Provide response to user to say request is being processed
- Notify result of processing by other means:
  - Let user assume it worked, discover otherwise
  - Return URL of status page
  - Send email/SMS on completion
  - Assume it worked, manual notification on failure

WAS Receives Datagram

- E.g. Update user data cached at web channel or non-user data such as product catalogue
- An alternative to persistent request/reply
- Usually persistent
  - may be non-persistent, e.g. update to product catalogue
- Web user may become aware of change to data as result of a later enquiry
  - Typically this will be via other means such as e-mail and SMS
- Requires unit of work and “trigger monitor”
- Exceptions must be handled locally

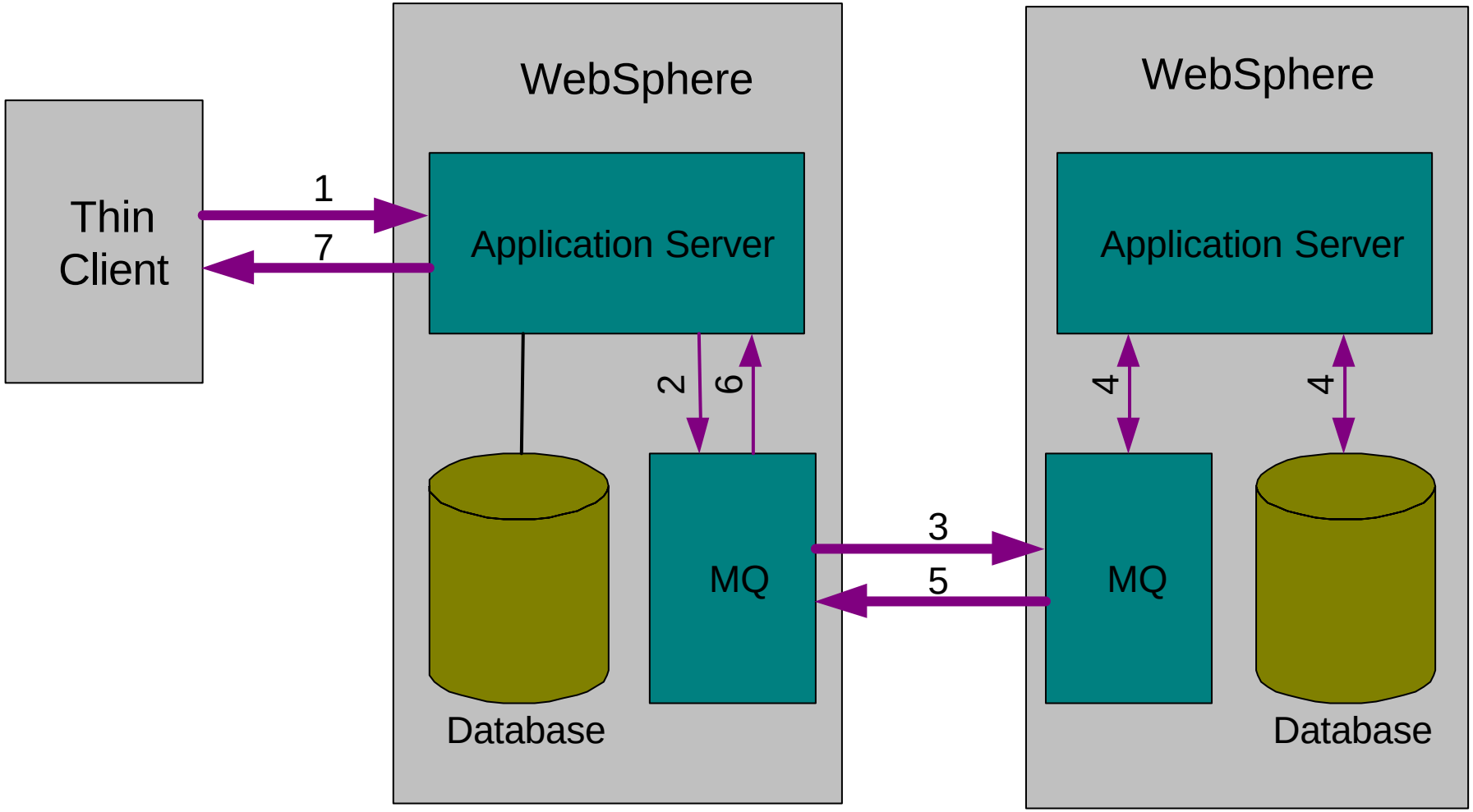
WAS Receives Datagram



WAS Receives Request/Reply

- WebSphere is used as the Enterprise System
- Use persistent messages
- Can model as two datagrams
- May route reply to different system
  - E.g. e-mail to say something went wrong
- Reply may be a return value or exception

WAS Receives Request/Reply



Agenda

- Why integrate?
- What you can do
- How you can do it
- Integrating with legacy systems
- What's new?
- Acknowledgements and References

MQSeries and Java

- Product extension (category 3 supportpac)
  - Last updated 01/03/2001
- Develop MQSeries applications in Java
- Comprises:
  - MQSeries classes for Java v5.2.0
  - MQSeries classes for Java Message Service (JMS) v5.2
- Platform coverage:
  - NT, AIX, AS400, HP-UX, Solaris, Linux, Windows 95/98/2000
  - Currently on OS/390, there only exists the MQSeries classes for Java v5.0
  - Currently on OS/400, the version is 5.1.2.

MQSeries classes for Java

- Provides Java language support for MQSeries applications
- implements an object oriented model for accessing MQSeries resources

- Supports various connection (transport) options:
  - client connection (TCP/IP, CORBA/IIOP)
  - bindings connection (JNI)

- Gives parity with support for C/C++, COBOL

MQSeries Classes for JMS

- MQSeries JMS implementation
  - Utilizes the MQSeries classes for Java to access MQSeries resources.
- Provides extra function over the MQSeries classes for Java:
  - Asynchronous message delivery
  - Message Selectors
  - Structured message classes
  - Transactional Support
- Provides an administration tool for defining administration objects and storing them in an enterprise directory service (JNDI namespace)

Client or Bindings?

- Client:
  - MQ client implementation, connects to queue manager over TCP/IP via the MQ listener
  - pure Java solution
  - obviously useful for connecting to remote queue managers
  - maybe used for local queue managers, consider loopback adapter

- Bindings:
  - MQ server application
  - Queue manager must be local to application
  - utilizes JNI to call the C MQI
  - performance benefits over client
  - application and queue manager more closely coupled

MQSeries Java vs. JMS

- MQSeries Java
  - traditional MQSeries object model approach
  - greater flexibility, more control
  - relatively straight forward to use
  - requires an understanding of the MQ API (MQI)

- MQSeries JMS
  - MQSeries implementation of a Java standard
  - simpler programming model, introduces a higher level of abstraction
  - application programmer may not need MQ specific skills.
  - reduced control and flexibility (from a traditional perspective)
  - extra functionality not available in MQSeries Java
  - abstraction layer has associated performance overhead

Sample JMS code using the JNDI

```
//retrieve connection factory from context (JNDI)
QueueConnectionFactory factory = (QueueConnectionFactory)context.lookup(conn_name);
// create connection
QueueConnection connection = factory.createQueueConnection();
connection.start(); //start connection otherwise no messages can be received
//Obtain session
QueueSession session = connection.CreateQueueSession(transacted, acknowlegeMode);
//retrieve destination from context
Queue queue = (Queue)context.lookup(queue_name);
//create sender/receiver
QueueSender queueSender = session.CreateSender(queue);
QueueReceiver queueReceiver = session.CreateReceiver(queue);
//send and receive message
TextMessage outMessage = session.createTextMessage();
queueSender.send(outMessage);
Message inMessage = queueReceiver.receive();
```

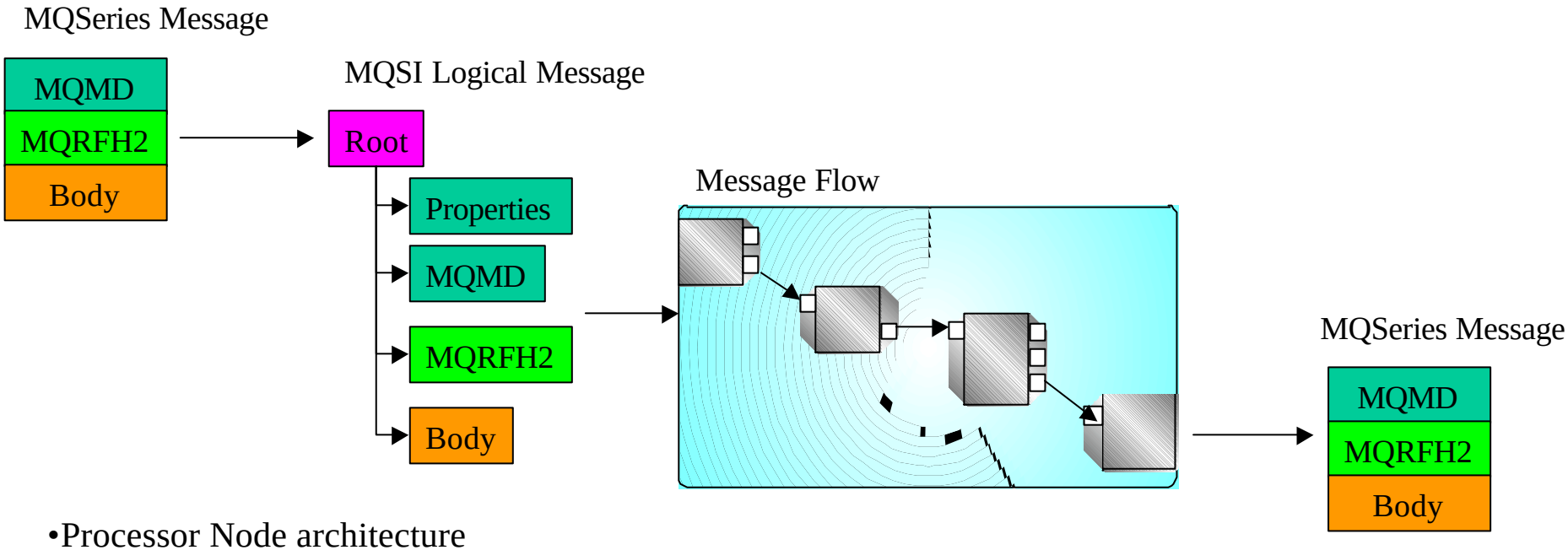
JMS Messages

- Header
  - JMSMessageID, JMSDestination,JMSDeliveryMode,
- Properties
  - JMSXUserID, JMSXApplID, JMSXDeliveryCount
- Data
  - 5 Subclasses for different content styles:
    - JMSBytesMessage: Unformatted binary data
    - JMSTextMessage: Character data
    - JMSStreamMessage: Sequence of typed data fields
    - JMSMapMessage: Collection of typed data fields
    - JMSObjectMessage: Serialized Java object

Agenda

- Why integrate?
- What you can do
- How you can do it
- Integrating with legacy systems
- What's new?
- Acknowledgements and References

MQSeries Integrator



- Processor Node architecture
- Significant extension and synergy with database
- Message parsing and formatting services
- Graphical tools for ease of use

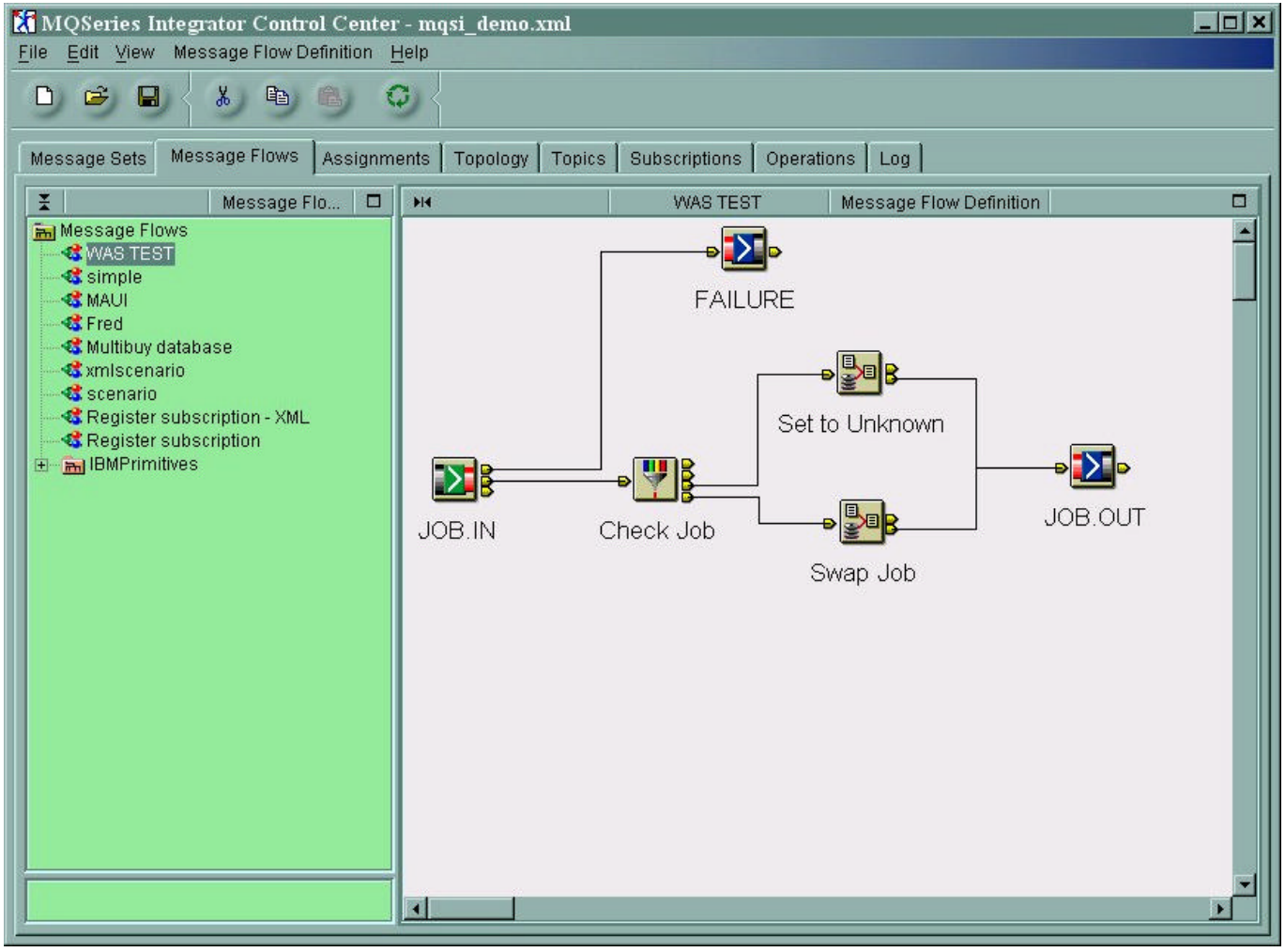
MQRFH2 Header

- Contains properties of message
  - Message Service Domain (Msd)
    - MRM
    - NEON
    - XML
    - none
  - Message Set (set)
  - Message Type (type)
  - Message Format (fmt)

JMS Meets MQSI

- MQSI uses MQRFH2.mcd.Msd to identify message domain
  - Must be set to “xml” or “jms_text” to invoke XML parser
- IBM JMS implementation uses MQRFH2.mcd.Msd to identify JMS message type
  - Use “jms_xxx” in MQRFH2.mcd.Msd for message type
  - Otherwise message is treated as TextMessage or BytesMessage according to MQMD.format

Example Message Flow



1. Read in a JMS text message
2. Manipulate it
3. Put it back out to a queue for a JMS application to pick up

This can result in a JMS Message Format Exception

Option 1: remove the MQRFH2 header

- You can add a compute node to remove the RFH2 header:
  - SET OutputRoot.MQRFH2=NULL
- JMS now assumes the message coming in is in byte code (jms_byte)
- You now have to re-format it to something useful

Option 2: Change the message format in MQSI to jms_text

- Use a *Reset Content Descriptor* Node to set the Format to jms_text
- The Msd in the MQRFH2 header is now set to jms_text
- The receiving JMS application can now format the message OK

Agenda

- Why integrate?
- What you can do
- How you can do it
- Integrating with legacy systems
- What's new?
- Acknowledgements and References

JTA Transaction to JMS

- MQSeries can be an XA resource manager for a WAS coordinated transaction
  - A single unit-of-work that includes WebSphere and Database and/or MQSeries resources
    - Implies 2-phase commit between WebSphere Application Server and MQSeries
- Must carry JTA transaction context from WebSphere onto the JMS Session object
- Honours the EJB transaction attributes (TX_REQUIRED, TX_NOT_SUPPORTED)

Prerequisites

- WebSphere Application Server 3.5
  - Advanced edition
  - Needs fixpack 3
- MQSeries 5.2
- MQSeries classes for JMS 5.2
  - MA88 supportpac
  - Use “WSQCF”, a modified QueueConnectionFactory
  - Use “WSTCF”, a modified TopicConnectionFactory

Using JTA/JMS 2PC

- Use WebSphere-specific versions of JMS administered objects
  - JMSWrapXAQueueConnectionFactory
  - JMSWrapXATopicConnectionFactory
- Connections obtained using these objects are *always* transactional
  - Use normal JMS QCF for non-transactional message handling

Note: 2PC only supported with JMS

- Base classes *do not support* 2PC with WebSphere

WebSphere Application Server 4.0

- GA target: 3Q/01
  - Beta running now
- JMS Listener
  - Start a transactional EJB
    - Uses the *onMessage* method
  - Additional to the EJB 1.1 specification
  - No need to modify the Deployment Descriptor
    - Separate side-file
    - XML
    - Describes how the EJB will be deployed to WebSphere

Message-Driven Beans

- Require the WAS equivalent of the CICS MQSeries Trigger Monitor
  - A “listener” that is part of the application server execution environment
  - Adapter function to get messages and make EJB method calls
- EJB 2.0 defines Message-Driven Beans
  - Container calls *onMessage* method of your bean when a message arrives
  - Bean is associated with a queue or topic via the Deployment Descriptor

Container-Managed Messaging

- Message-driven beans will either
  - Implement business logic
  - Act as an “adapter” to existing or other “messaging-ignorant” EJB
- Container-Managed Messaging provides the functionality for such an adapter bean
  - Parse message format based on definition in repository
  - Map to method invocation
- Targeted for WebSphere Application Server 4.5 (2Q/02)

Agenda

- Why integrate?
- What you can do
- How you can do it
- Integrating with legacy systems
- What's new?
- Acknowledgements and References

Acknowledgements and References

- MA88: MQSeries classes for Java and MQSeries classes for JMS 5.2
  - http://www-4.ibm.com/software/ts/mqseries/txppacs/ma88.html
- WebSphere JMS/JTA support for MQSeries Overview
  - A paper by Joanna Hodgson et al.
  - Available on the internet at:

    http://www-4.ibm.com/software/webservers/appserv/whitepapers.html