

14

Asynchronous Communication With Message Driven Beans

Objectives

After completing this lesson, you should be able to do the following:

- Identify the features of a messaging system
- Describe the Java Message Service (JMS) architecture
- Configure a Java Message Service
- Create a message-driven bean (MDB)
- Create a JMS/MDB client



Messaging Systems

A messaging system is a system that:

- Uses messages to communicate between components (rather than direct method invocations)
- Enables loosely coupled components
- Is peer-to-peer in nature

Message-Oriented Middleware

- Message-oriented middleware refers to an infrastructure that supports messaging.
- Typical message-oriented middleware architectures define these elements:
 - Message structure
 - The way to send and receive messages
 - Scaling guidelines

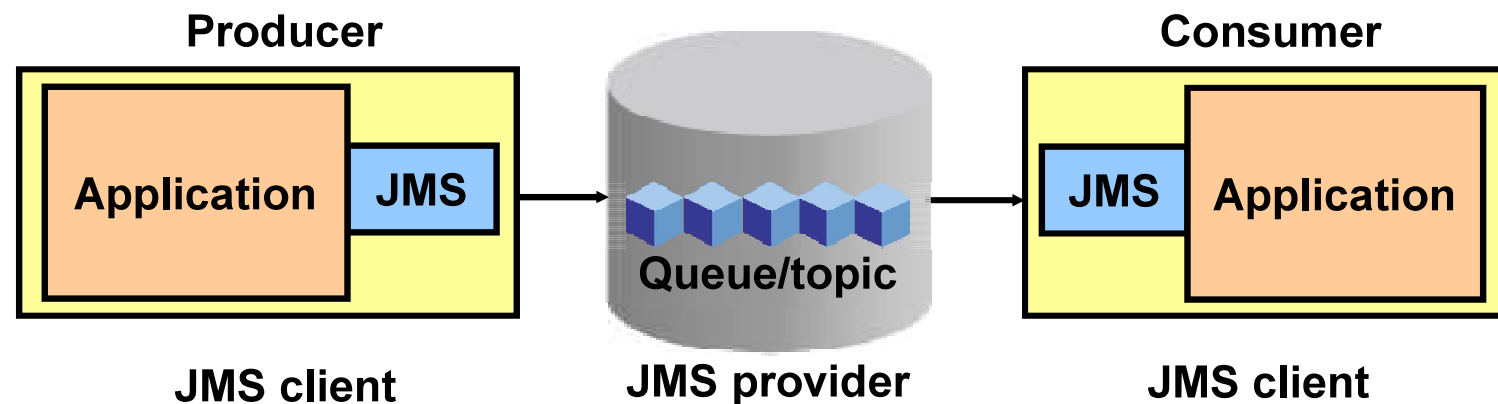
Java Message Service (JMS):

- Is a Java EE standard for developing messaging systems
- Provides an architecture for loosely coupled integration of systems through the use of messages
- Supports both point-to-point and publish/subscribe models for message delivery
- Is asynchronous in nature
- Enables reliable communication

JMS Application Architecture

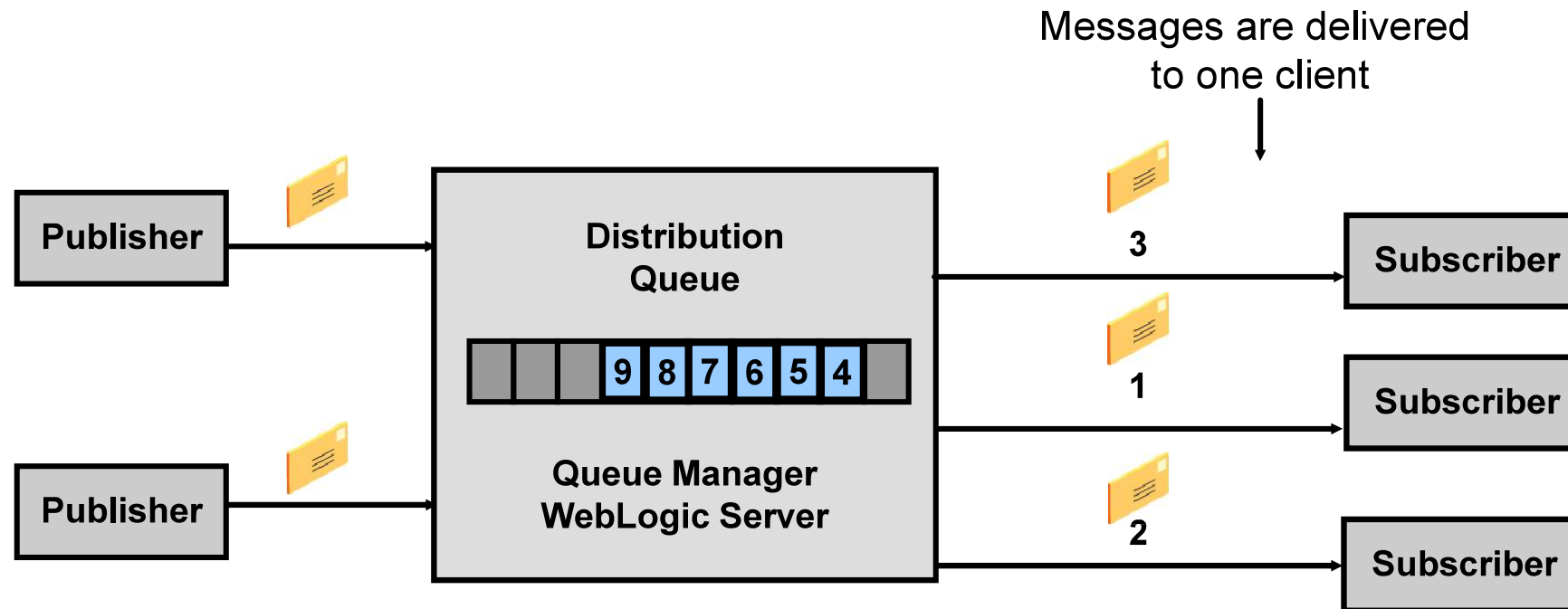
The main components of a JMS application:

- JMS clients (producer and consumer)
- JMS provider
- Administered objects
 - Connection factories
 - Destinations (queues or topics)



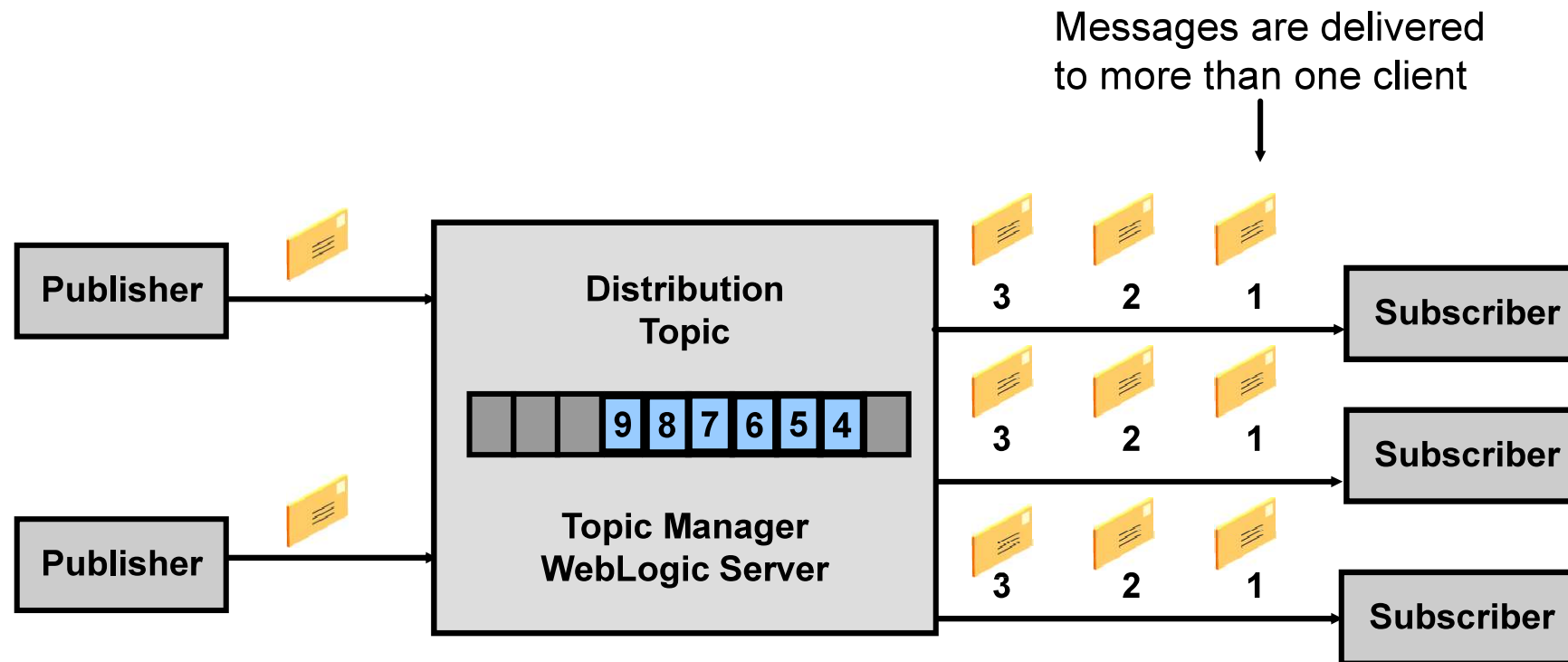
Point-to-Point Queue

Many producers can serialize messages to multiple receivers in a queue.



Publish-Subscribe Topics

Publishing and subscribing to a topic decouples producers from consumers.



Identify the correct statements about the point-to-point or queuing model.

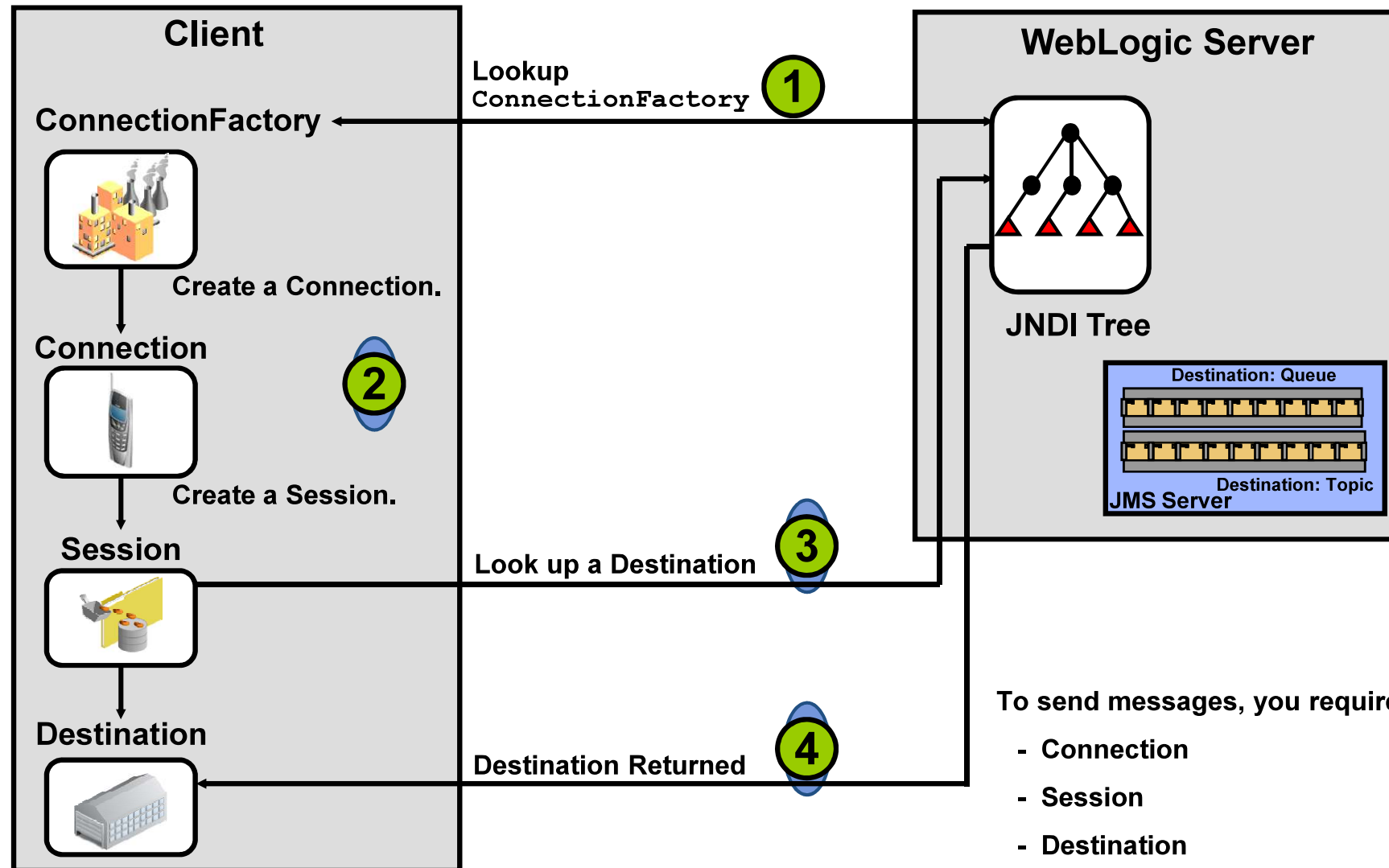
1. Multiple consumers will get the message.
2. The producer does not have to be running at the time the consumer consumes the message, nor does the consumer need to be running at the time the message is sent.
3. Every message successfully processed is acknowledged by the consumer.

WebLogic Server JMS Features

WebLogic Server JMS supports:

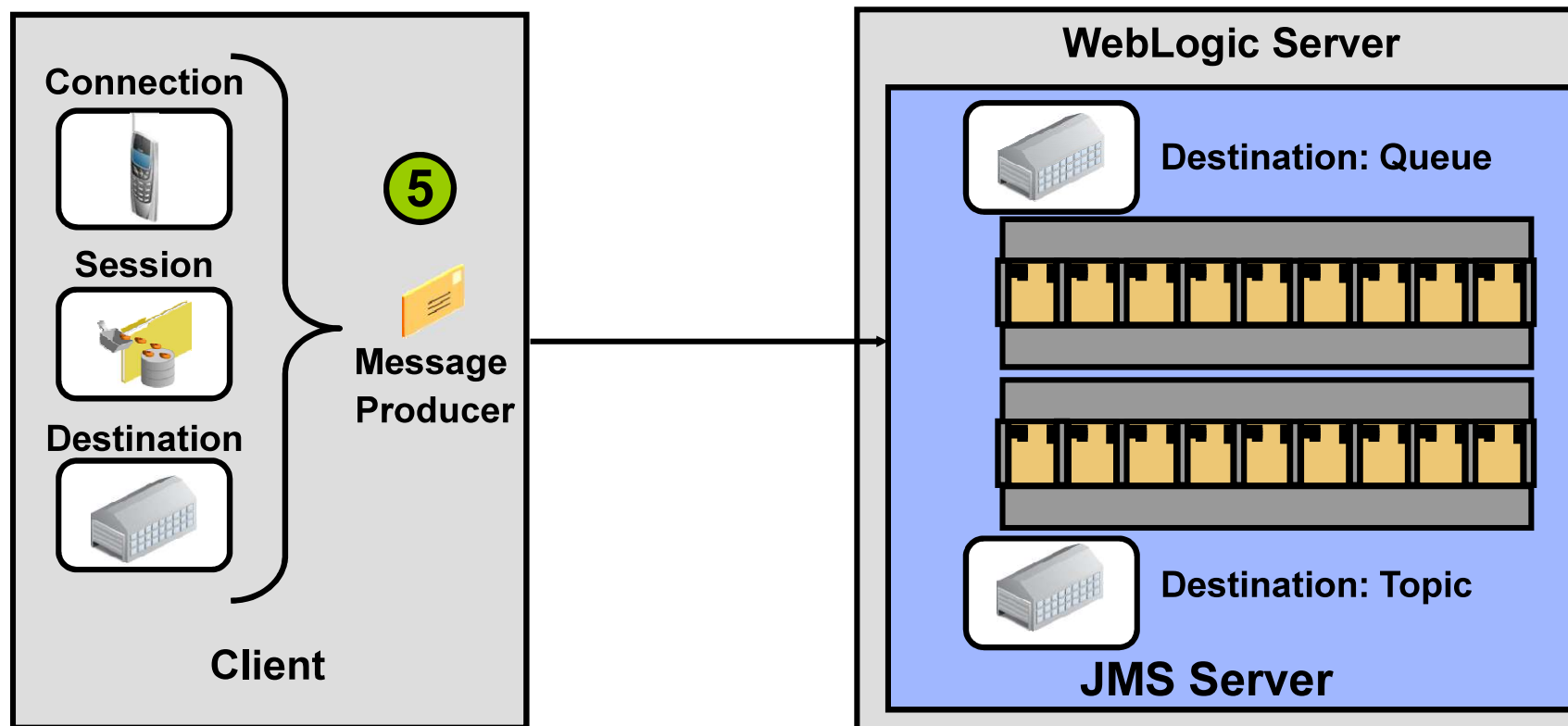
- PTP and pub/sub domains
- Guaranteed and transactional message delivery
- Durable subscribers
- Distributed destinations
- Recovery from failed servers

JMS Architecture: Connecting



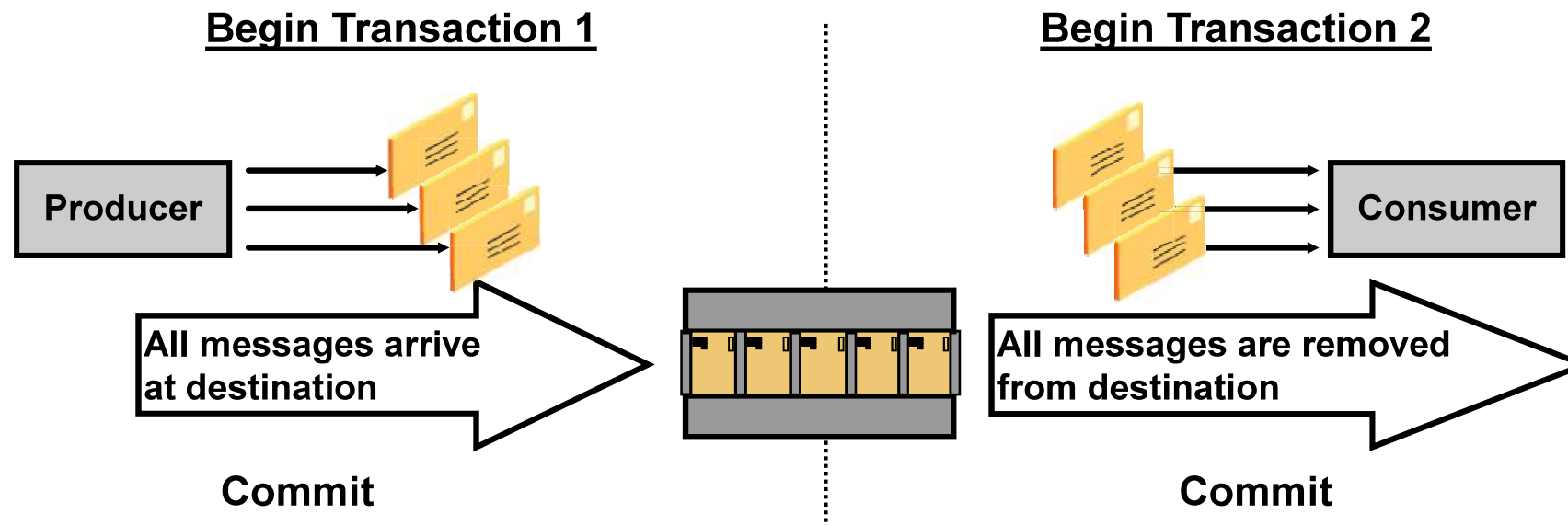
JMS Architecture: Sending Messages

Connection, Session, and Destination are used to send a message.



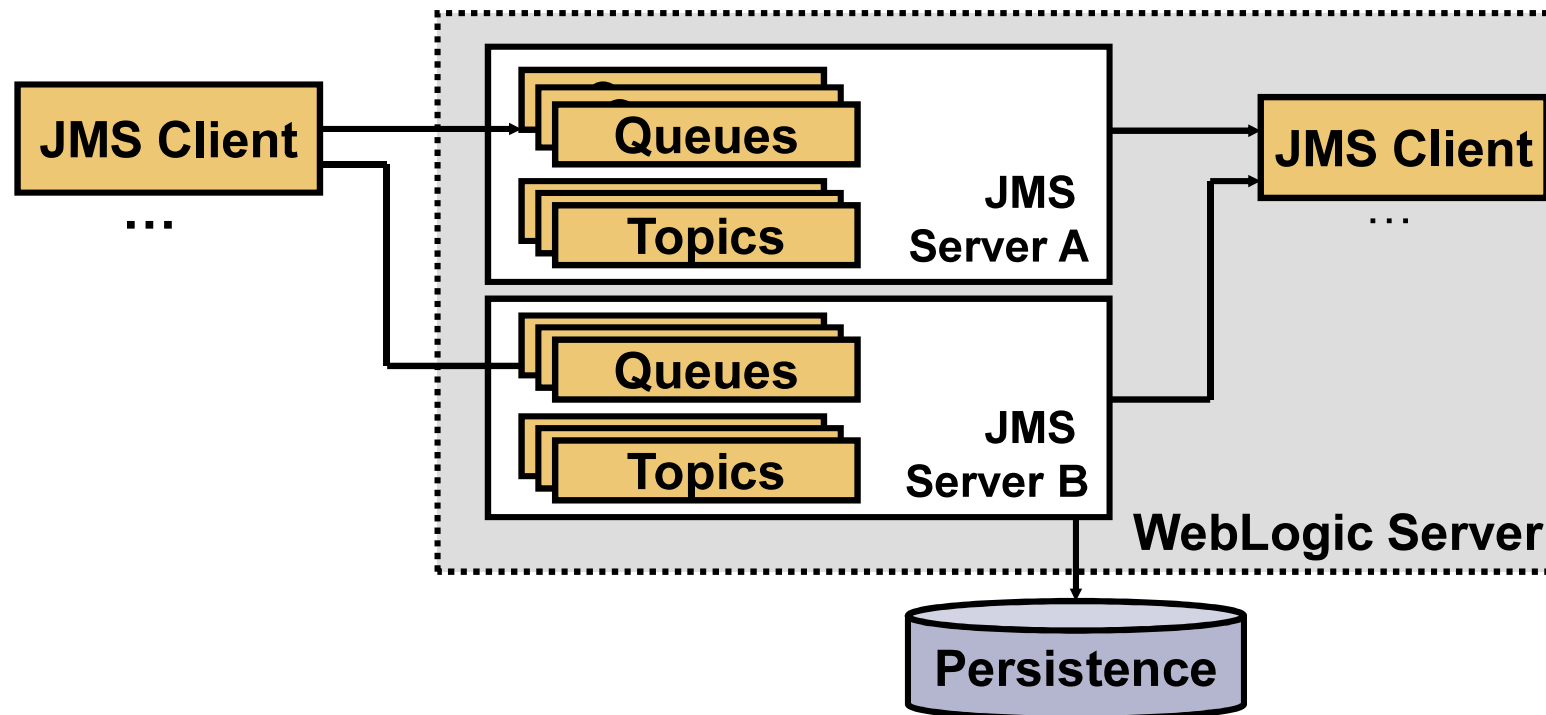
Transacted Messaging

- A JMS client can use JTA to participate in a distributed transaction.
- Alternatively, a JMS client can demarcate transactions local to the JMS Session, through a transacted session.
- Participation in a transaction is optional.



WebLogic Server JMS Server

- In WLS, the messaging service is implemented through a JMS server.
- A JMS server receives and distributes messages.



Creating a JMS Server

Domain Structure

- wl_server
 - Environment
 - Deployments
 - Services
 - Messaging
 - JMS Servers**
 - Store-and-Forward Agents
 - JMS Modules
 - Bridges
 - JDBC
 - Persistent Stores
 - Path Services
 - Foreign JNDI Providers
 - Work Contexts
 - XML Registries
 - XML Entity Caches
 - jCOM
 - Mail Sessions
 - File T3
 - JTA
 - Security Realms
 - Interoperability
 - Diagnostics

JMS Servers

Name	Persistent Store	TargetName
examplesJMS Server	exampleJDBCStore	examplesServer
WseeJMS Server	WseeFileStore	examplesServer

Create a New JMS Server

JMS Server Properties

The following properties will be used to identify your new JMS Server.

What would you like to name your new JMS Server?

Name: MyJMS Server

Specify persistent store for the new JMS Server.

Persistent Store: (none) Create a New Store

Connection Factory

- A connection factory:
 - Encapsulates connection configuration information
 - Is used to create preconfigured connections
 - Is stored in JNDI
 - Can be targeted to servers or clusters
- WLS provides a default connection factory that is bound in JNDI to `weblogic.jms.ConnectionFactory`.
- When a new configuration is required, a new connection factory can be created.

JMS Destination

- A JMS destination is a lightweight object stored in JNDI.
- It is the target on a JMS server for sending or receiving messages.
- The JMS destination types are:
 - Queue
 - Topic

Creating a Queue Destination

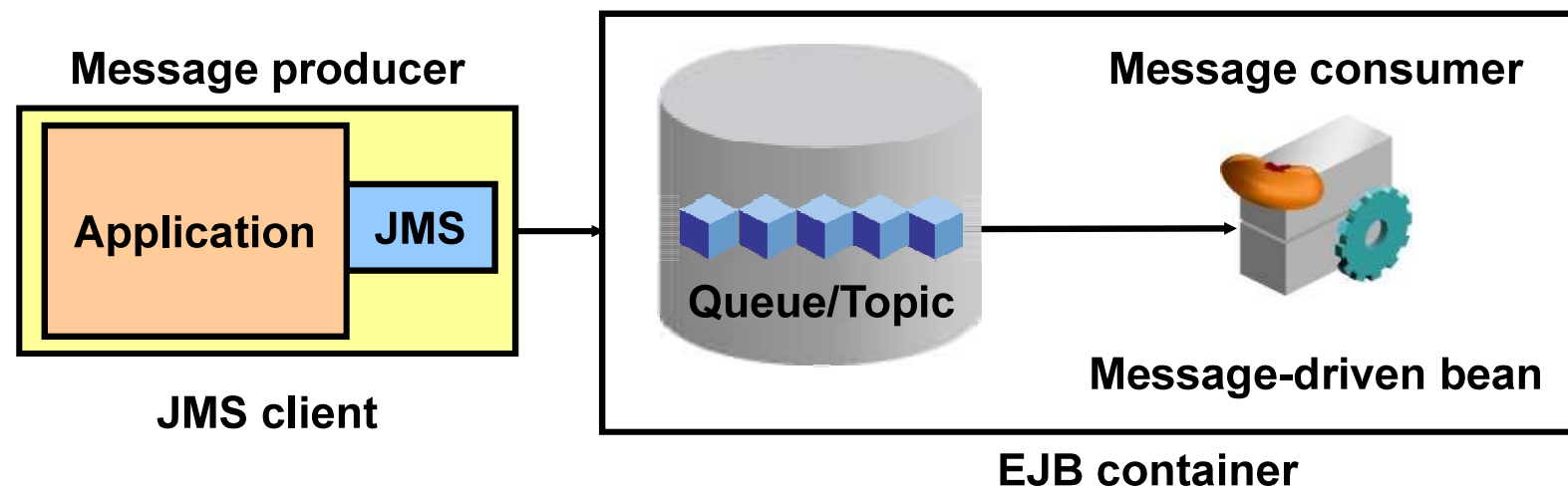
The image displays three overlapping screenshots from the JBoss IDE, illustrating the process of creating a new subdeployment for a JMS module.

- Left Screenshot (A):** Shows the "Settings for MyJMSModule" window with the "Subdeployments" tab selected. A "New" button is highlighted in the "Subdeployments" table.
- Middle Screenshot (B):** Shows the "Create a New Subdeployment" dialog. The "Targets" section is visible, and the "MyJMSServer" checkbox is selected.
- Right Screenshot (C):** Shows the "Settings for MyJMSModule" window with the "Subdeployments" tab selected. The "Summary of Resources" table is empty, indicating no items to display.

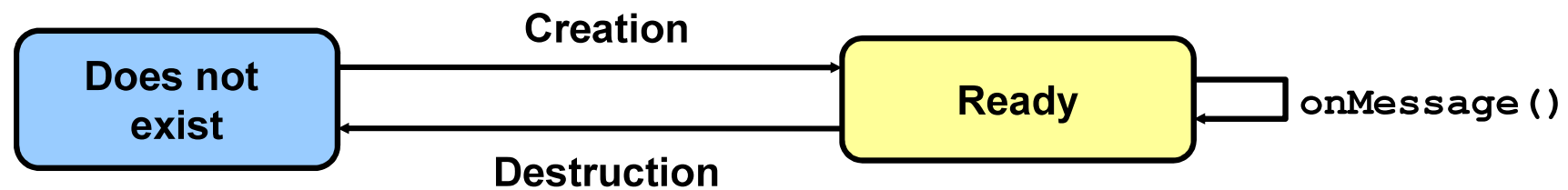
Message-Driven Beans

A message-driven bean (MDB):

- Is a stateless EJB that asynchronously consumes JMS messages
- Is never called directly by a client
- Has an `onMessage()` method that is invoked by the EJB container when a message arrives



Life Cycle of a Message-Driven Bean



Creating a Message-Driven Bean

```
@MessageDriven(mappedName = "jms/demoTopic", ①
    name = "MessageBean",
    activationConfig = { ②
        @ActivationConfigProperty(
            propertyName="connectionFactoryJndiName",
            propertyValue="jms/TopicConnectionFactory"),
        @ActivationConfigProperty(
            propertyName="destinationName",
            propertyValue="jms/demoTopic"),
        @ActivationConfigProperty(
            propertyName="destinationType",
            propertyValue="javax.jms.Topic"),
        @ActivationConfigProperty(
            propertyName="messageSelector",
            propertyValue="RECIPIENT = 'MDB'") } )
public class MDBBean implements MessageListener {
    ③
    @Resource javax.ejb.MessageDrivenContext mc;
    public void ④
        onMessage(Message message) {
            ... }
}
```


Creating a JMS/MDB Client

```
public class JMSClient {  
    public static void main(String [] args) {  
        try {  
            Context context = new InitialContext();  
            ConnectionFactory connectionFactory =  
                (ConnectionFactory) new  
                    InitialContext().lookup("  
                        jms/TopicConnectionFactory");  
            Connection connection =  
                connectionFactory.createConnection();  
            connection.start();  
            Session topicSession =  
                connection.createSession(false,  
                    Session.AUTO_ACKNOWLEDGE);  
            Destination topic = (Topic) new  
                InitialContext().lookup("jms/demoTopic");
```

1

2

3

4

Creating a JMS/MDB Client

```
MessageProducer publisher =  
    topicSession.createProducer(topic);  
Message message = topicSession.createMessage();  
message.setJMSType("theMessage");  
message.setLongProperty("time",  
    System.currentTimeMillis());  
message.setStringProperty("subject",  
    "Test EJB 3.0 MDB");  
message.setStringProperty("message",  
    "Test message sent to EJB 3.0 MDB");  
message.setStringProperty("RECIPIENT", "MDB");  
publisher.send(message);  
publisher.close();  
topicSession.close();  
connection.close();  
} catch(Throwable ex) { ex.printStackTrace(); }  
}  
}
```

5

6

7

8

Summary

In this lesson, you should have learned how to:

- Identify the features of a messaging system
- Describe the Java Message Service (JMS) architecture
- Configure a Java Message Service
- Create a message-driven bean (MDB)
- Create a JMS/MDB client



Practice: Overview

These practices covers the following topics:

- Creating a message-driven bean to send emails
- Modifying the session bean to produce messages for the MDB to send as emails
- Configuring JMS resources in WebLogic Server
- Testing the email functionality using an email client