Spring JMS

CS544: Enterprise Architecture

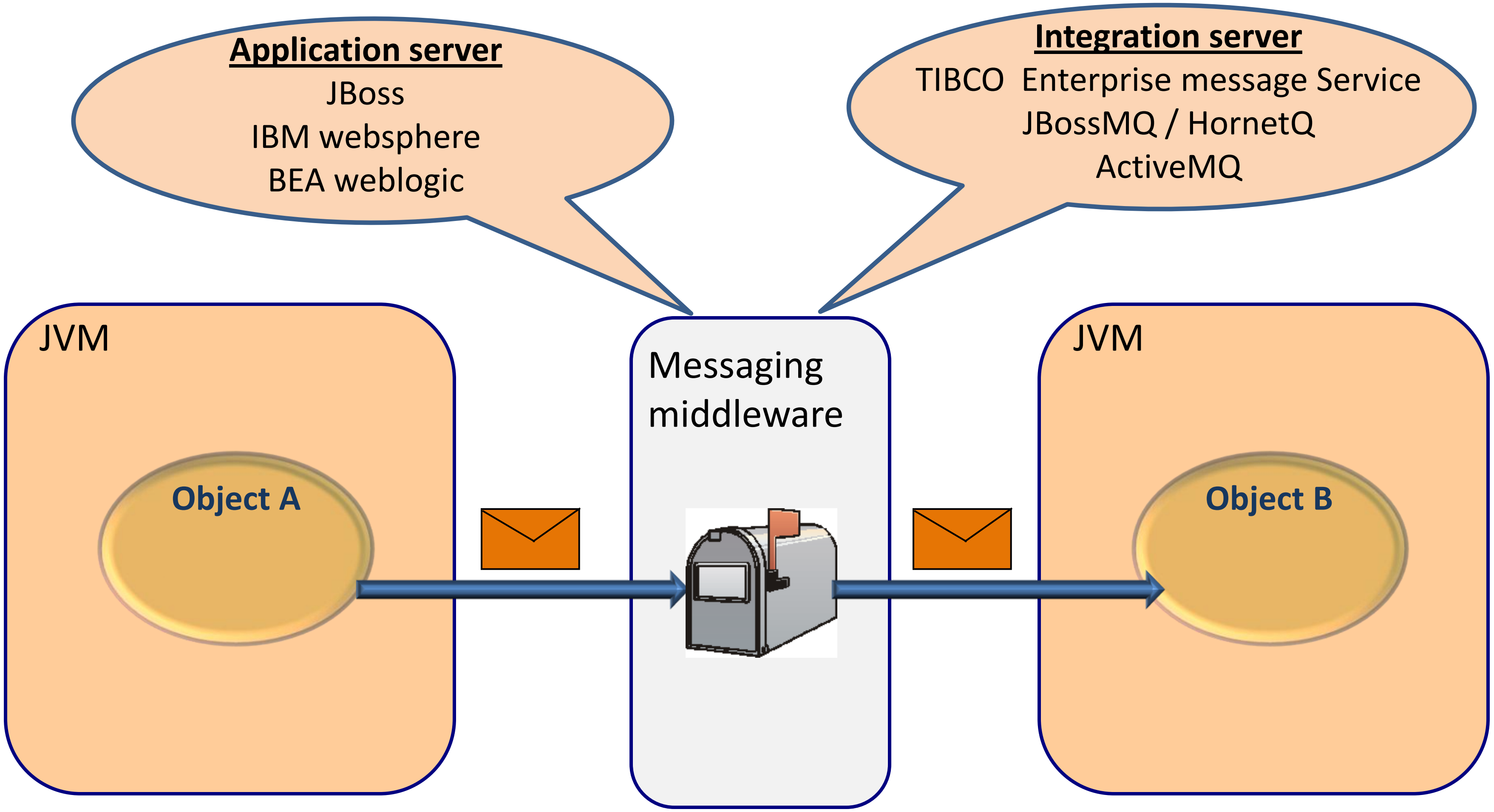
Spring JMS

- In this module we will start by going over some of the basic JMS terminology, after which we will compare a JMS application with and without using the Spring Template.
- At the end of the module there is also a short discussion on JMS and concurrency, and how Spring can easily create object pools to mitigate any concurrency issues.

Spring JMS:

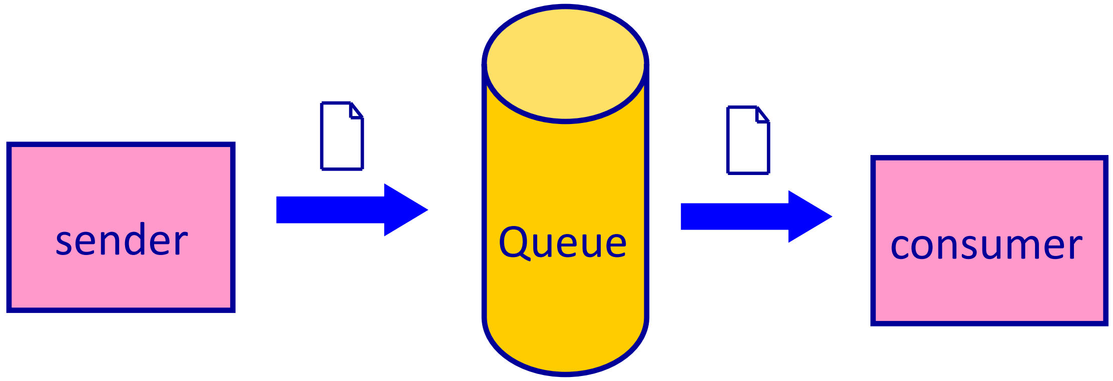
JSM BASICS

Java Message Service (JMS)



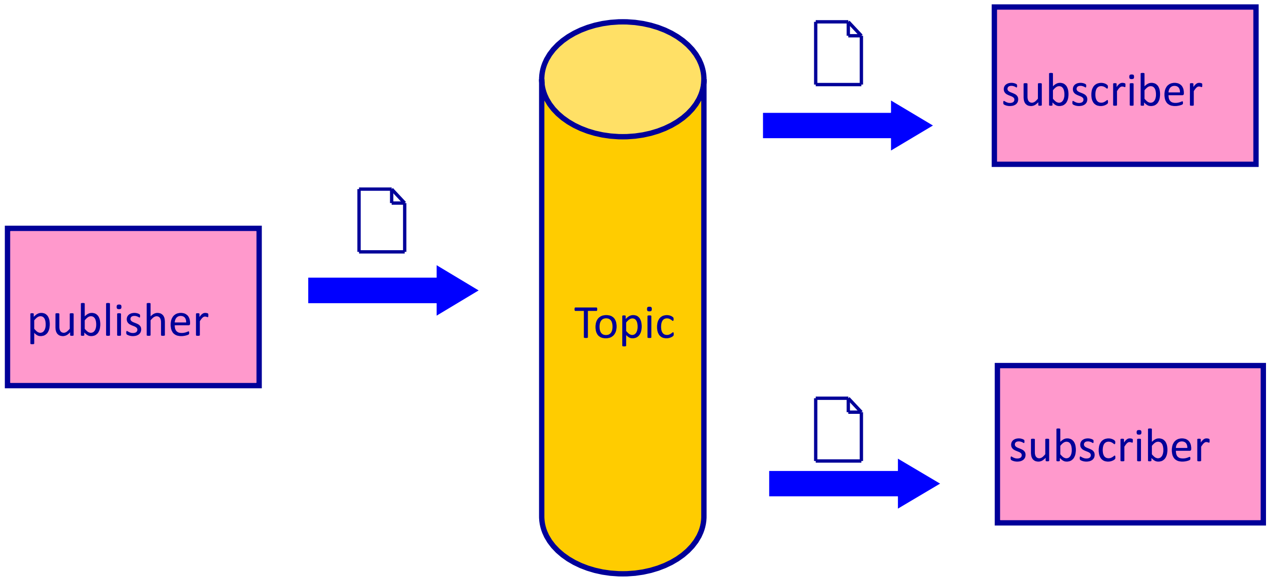
Point-To-Point (PTP)

- A dedicated consumer per Queue message



Publish-Subscribe (Pub-Sub)

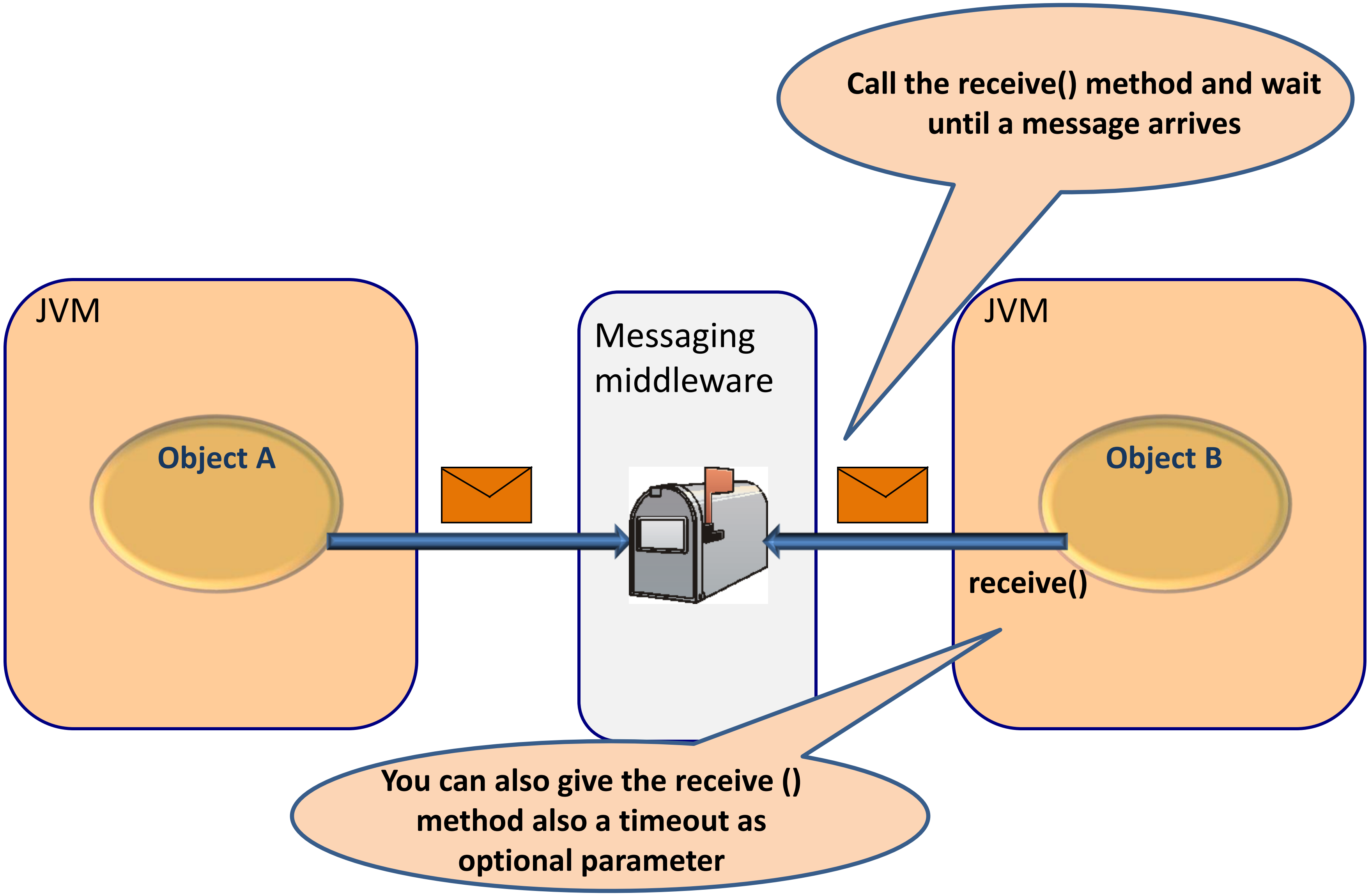
- A message channel can have more than one '*consumer*'
  - Ideal for broadcasting



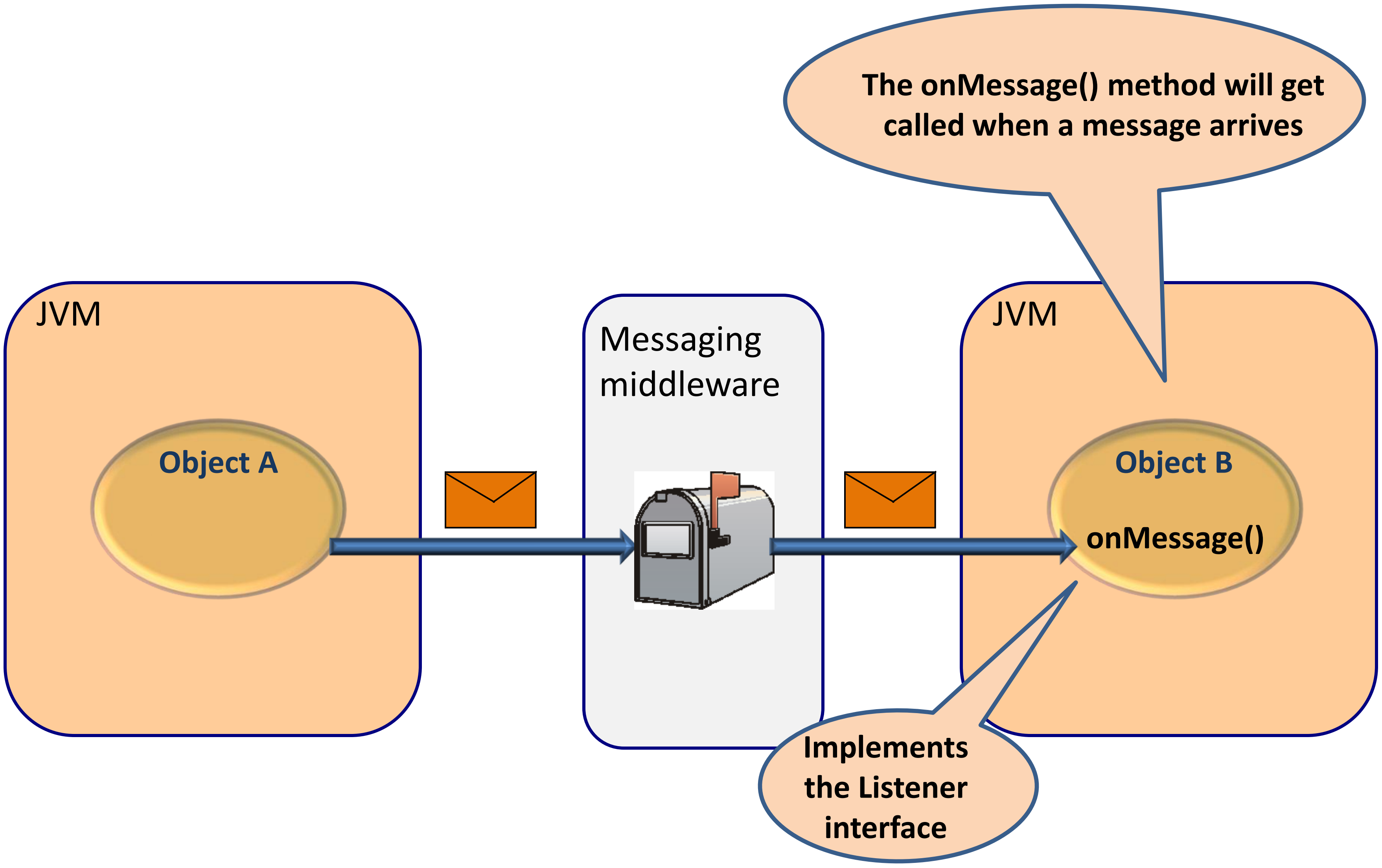
JMS receiver

- JMS has two types of receivers
  - Synchronous receiver
  - Asynchronous receiver

Synchronous receiver


Call the receive() method and wait until a message arrives
JVM
Object A
Messaging middleware
JVM
Object B
receive()
You can also give the receive () method also a timeout as optional parameter

Asynchronous receiver


The onMessage() method will get called when a message arrives
JVM
Object A
Messaging middleware
JVM
Object B
onMessage()
Implements the Listener interface

JMS Basics

- JMS itself is an asynchronous protocol
  - Although JMS receivers can pickup messages either synchronously or asynchronously
- Messages can either be sent point-to-point or through a publish-subscribe system
- JMS requires a JMS middle ware server
  - Either a full Java Enterprise Application Server, or a stand alone JMS server

Spring JMS:

SPRING JMS

JMS sender

jndiContext creation not shown

```
//Lookup a ConnectionFactory with JNDI
QueueConnectionFactory queueConnectionFactory = (QueueConnectionFactory)
jndiContext.lookup("MyJMS Connection Factory");
// Lookup a Destination with JNDI
Queue queue = (Queue) jndiContext.lookup("MyJMSQueue");
// Use the ConnectionFactory to create a Connection
QueueConnection queueConnection = queueConnectionFactory.createQueueConnection();
// Use the Connection to create a Session
QueueSession queueSession =
                queueConnection.createQueueSession(false,Session.AUTO_ACKNOWLEDGE);
// Use the Session to create a MessageProducer for this queue
QueueSender queueSender = queueSession.createSender(queue);
// Use the Session to create a Message
TextMessage message = queueSession.createTextMessage();
message.setText("Hello World");
// Use the MessageProducer to send the Message
queueSender.send(message);
```

Spring JMS sender (PTP)

```java
public class PTPSenderApplication {
  public static void main(String[] args) {
    ApplicationContext context = new ClassPathXmlApplicationContext("springconfig.xml");
    SpringQueueSender sender = context.getBean("springQueueSender",SpringQueueSender.class);
    sender.send("Hello World");
  }
}
```

```java
public class SpringQueueSender {
    private JmsTemplate jmsTemplate;

    public void send(final String text) {

        jmsTemplate.send(new MessageCreator() {
              public Message createMessage(Session session) throws JMSException {
                return session.createTextMessage(text);
              }
        });
        System.out.println("Sending message: " + text);
    }

    public void setJmsTemplate(JmsTemplate jmsTemplate) {
        this.jmsTemplate = jmsTemplate;
    }
}
```

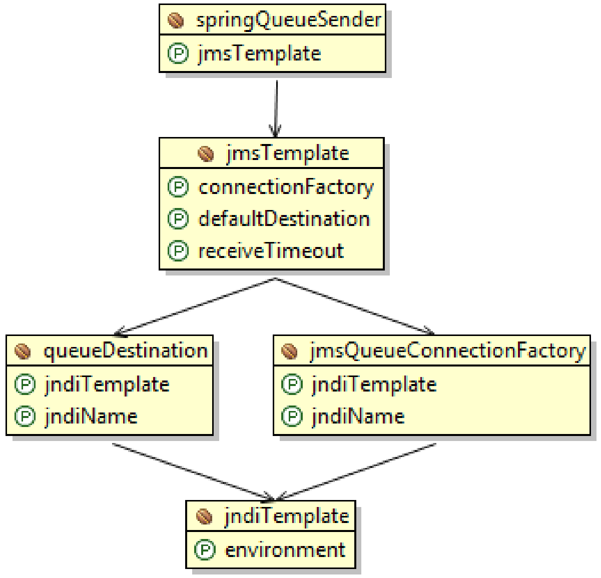
Will be injected

The spring configuration file(1/2)

```
<?xml version="1.0" encoding="UTF-8"?>
<beans …
    <bean id="queueDestination" class="org.springframework.jndi.JndiObjectFactoryBean">
        <property name="jndiTemplate" ref="jndiTemplate"/>
        <property name="jndiName" value="queue/testQueue"/>
    </bean>
    <bean id="jmsTemplate" class="org.springframework.jms.core.JmsTemplate">
        <property name="connectionFactory" ref="jmsQueueConnectionFactory" />
        <property name="defaultDestination" ref="queueDestination"/>
        <property name="receiveTimeout" value="5000" />
   </bean>
   <bean id="springQueueSender" class="springjms.SpringQueueSender">
        <property name="jmsTemplate" ref="jmsTemplate"/>
   </bean>
```

The name of the queue

Inject the jmsTemplate



The spring configuration file(2/2)

```
  <bean id="jmsQueueConnectionFactory" class="org.springframework.jndi.JndiObjectFactoryBean">
      <property name="jndiTemplate" ref ="jndiTemplate"/>
      <property name="jndiName" value="ConnectionFactory" />
  </bean>
  <bean id="jndiTemplate" class="org.springframework.jndi.JndiTemplate">
      <property name="environment">
          <props>
              <prop key="java.naming.factory.initial">
                  org.jnp.interfaces.NamingContextFactory
              </prop>
              <prop key="java.naming.provider.url">
                  localhost
              </prop>
              <prop key="java.naming.factory.url.pkgs">
                  org.jnp.interfaces:org.jboss.naming
              </prop>
          </props>
      </property>
  </bean>
</beans>
```

JMS Server specific value

JMS Server specific values

JMS Synchronous Receiver

```
//Lookup a ConnectionFactory with JNDI
QueueConnectionFactory queueConnectionFactory = (QueueConnectionFactory)
jndiContext.lookup("MyJMS Connection Factory");
// Lookup a Destination with JNDI
Queue queue = (Queue) jndiContext.lookup("MyJMSQueue");
// Use the ConnectionFactory to create a Connection
QueueConnection queueConnection = queueConnectionFactory.createQueueConnection();
// Use the Connection to create a Session
QueueSession queueSession =
             queueConnection.createQueueSession(false,Session.AUTO_ACKNOWLEDGE);
// Use the Session to create a MessageReceiver for this queue
QueueReceiver queueReceiver = queueSession.createReceiver(queue);
//Start the connection such that messages get delivered
queueConnection.start();
//Receive the message
Message m = queueReceiver.receive(1);
TextMessage message = (TextMessage) m;
System.out.println("Receiving message: " +message.getText());
```

Spring JMS synchronous receiver (PTP)

```
public class PTPReceiverApplication {
  public static void main(String[] args) {
    ApplicationContext context = new ClassPathXmlApplicationContext("springconfig.xml");
    SpringQueueReceiver receiver = context.getBean("springQueueReceiver",
                                                        SpringQueueReceiver.class);
    receiver.receiveMessage();
  }
}
```

```
public class SpringQueueReceiver {
    private JmsTemplate jmsTemplate;

    public void setJmsTemplate(JmsTemplate jmsTemplate) {
        this.jmsTemplate = jmsTemplate;
    }

    public void receiveMessage() {
      Message msg = jmsTemplate.receive();
      if (msg != null) {
        try {
          TextMessage message = (TextMessage) msg;
          if (message != null) {
            System.out.println("Receiving message: "+ message.getText());
          }
        } catch (Exception e) {
          System.out.println("Exception in SpringQueueReceiver receiveMessage(): " + e);
        }
      }
    }
}
```

Will be injected

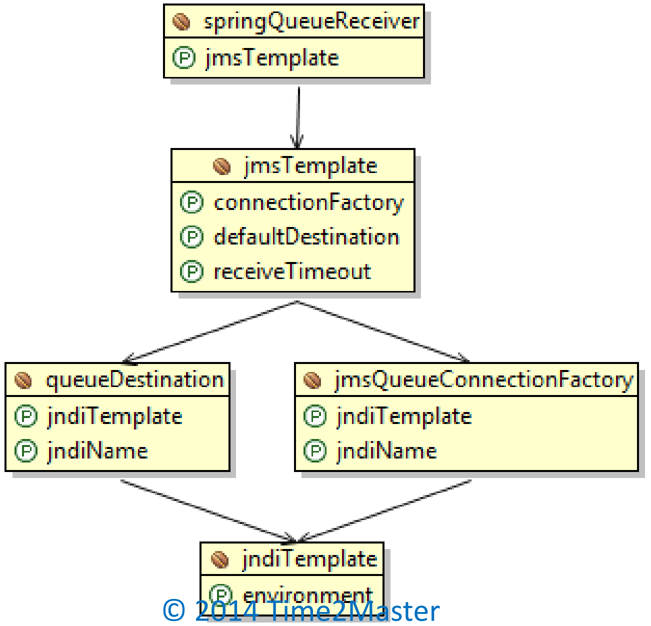
Call the receive() method

The spring configuration file(1/2)

```
<?xml version="1.0" encoding="UTF-8"?>
<beans …
    <bean id="queueDestination" class="org.springframework.jndi.JndiObjectFactoryBean">
        <property name="jndiTemplate" ref="jndiTemplate"/>
        <property name="jndiName" value="queue/testQueue"/>
    </bean>
    <bean id="jmsTemplate" class="org.springframework.jms.core.JmsTemplate">
        <property name="connectionFactory" ref="jmsQueueConnectionFactory" />
        <property name="defaultDestination" ref="queueDestination"/>
        <property name="receiveTimeout" value="5000" />
    </bean>
    <bean id="springQueueReceiver" class="springjms.SpringQueueReceiver">
        <property name="jmsTemplate" ref="jmsTemplate"/>
    </bean>
```

The name of the queue

Inject the jmsTemplate



The spring configuration file(2/2)

```xml
  <bean id="jmsQueueConnectionFactory" class="org.springframework.jndi.JndiObjectFactoryBean">
      <property name="jndiTemplate" ref ="jndiTemplate"/>
      <property name="jndiName" value="UIL2ConnectionFactory" />
  </bean>
  <bean id="jndiTemplate" class="org.springframework.jndi.JndiTemplate">
      <property name="environment">
          <props>
              <prop key="java.naming.factory.initial">
                  org.jnp.interfaces.NamingContextFactory
              </prop>
              <prop key="java.naming.provider.url">
                  localhost
              </prop>
              <prop key="java.naming.factory.url.pkgs">
                  org.jnp.interfaces:org.jboss.naming
              </prop>
          </props>
      </property>
  </bean>
</beans>
```

Spring JMS asynchronous receiver

Implements the MessageListener interface

The onMessage() is called when a message arrives in the message box

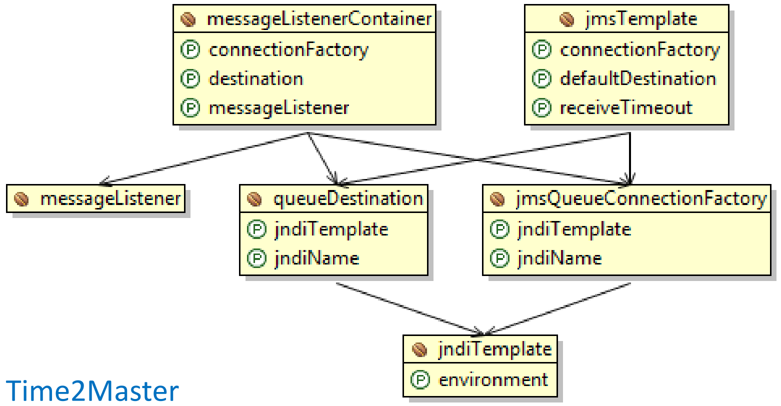
```
public class MessageListenerImpl implements MessageListener {

    public void onMessage(Message message){
        try {
            TextMessage textMessage = (TextMessage)message;
            System.out.println("message received: " + textMessage.getText());
        } catch (JMSException e) {
             System.out.println("JMSException in MessageListenerImpl omMessage()" + e);
        }
    }
}
```

Spring JMS Assynchronous receiver

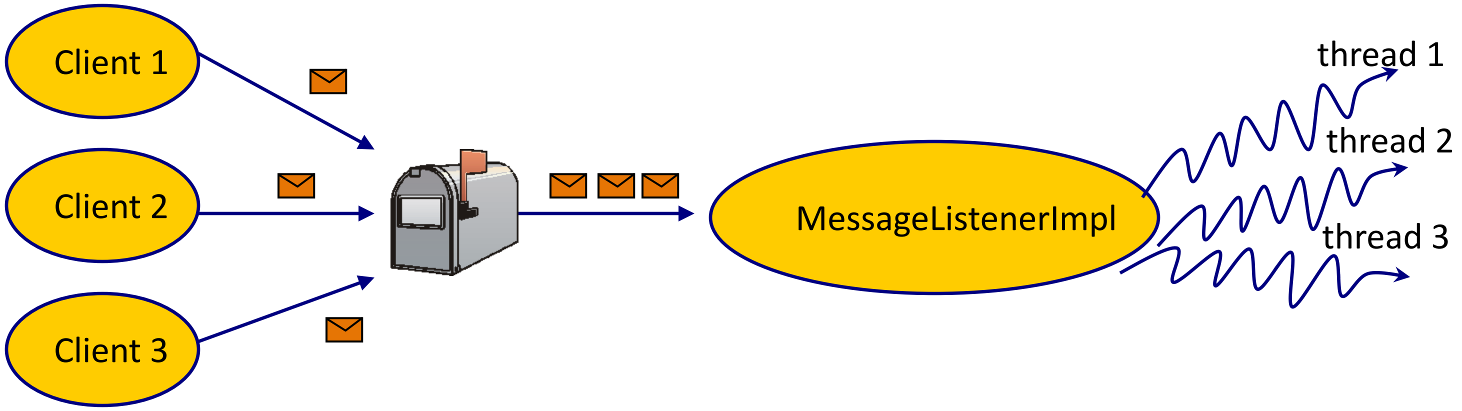
```
<?xml version="1.0" encoding="UTF-8"?>
<beans …
    <!-- JMS Queue Connection Factory -->
    <bean id="jmsQueueConnectionFactory" class="org.springframework.jndi.JndiObjectFactoryBean">
        …
    </bean>
    <bean id="jndiTemplate" class="org.springframework.jndi.JndiTemplate">
        …
    </bean>
    <bean id="queueDestination" class="org.springframework.jndi.JndiObjectFactoryBean">
        …
    </bean>
    <bean id="messageListenerContainer"
            class="org.springframework.jms.listener.DefaultMessageListenerContainer">
     <property name="connectionFactory" ref="jmsQueueConnectionFactory"/>
     <property name="destination" ref="queueDestination"/>
     <property name="messageListener" ref="messageListener"/>
   </bean>

   <bean id="messageListener" class="springjms.MessageListenerImpl"/>
</beans>
```

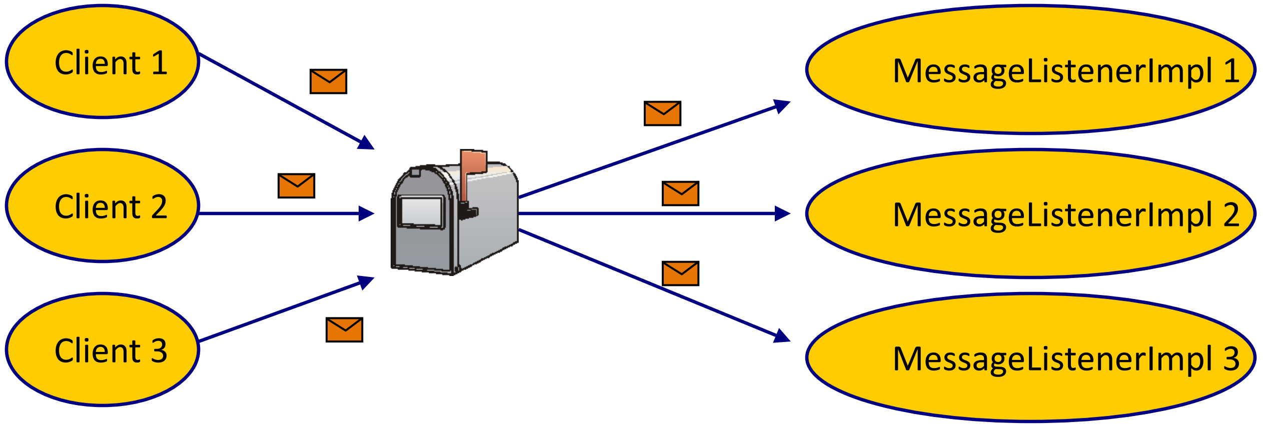


JMS and concurrency

- Every OnMessage() methode executes in its own thread



- Another option: pooling



JMS and pooling

```
<?xml version="1.0" encoding="UTF-8"?>
<beans …
    <!-- JMS Queue Connection Factory -->
    <bean id="jmsQueueConnectionFactory" class="org.springframework.jndi.JndiObjectFactoryBean">
        …
    </bean>
    <bean id="jndiTemplate" class="org.springframework.jndi.JndiTemplate">
        …
    </bean>
    <bean id="queueDestination" class="org.springframework.jndi.JndiObjectFactoryBean">
        …
    </bean>
    <bean id="messageListenerContainer"
              class="org.springframework.jms.listener.DefaultMessageListenerContainer">
     <property name="connectionFactory" ref="jmsQueueConnectionFactory"/>
     <property name="destination" ref="queueDestination"/>
     <property name="messageListener" ref="messageListener"/>
   </bean>
   <bean id= "mdpojo" class="springjms.MessageListenerImpl" scope="prototype" />
   <bean id="poolTargetSource" class="org.springframework.aop.target.CommonsPoolTargetSource">
     <property name="targetBeanName" value="mdpojo"/>
     <property name="maxSize" value="25"/>
   </bean>
   <bean id="messageListener" class="org.springframework.aop.framework.ProxyFactoryBean">
     <property name="targetSource" ref="poolTargetSource"/>
   </bean>
</beans>
```

prototype

Active Learning

- Are JMS messages sent Synchronously, Asynchronously or are both an option?

- Is concurrency a problem for JMS synchronous receivers?

Summary

- Spring makes it easy to send a JMS message
  - All JMS details are declared in the XML file
  - Use the JMSTemplate.send() method
- Spring makes it easy to receive a JMS message
  - All JMS details are declared in the XML file
  - Use JMSTemplate.receive() method for a synchronous receiver
  - Use a MessageListener for a asynchronous receiver
- By default, the OnMessage() method of the MessageListener is multithreaded
- You can also use pooling very easily with Spring by just configuring a pool in the XML file

Main Point

- JMS is an asynchronous message protocol, Spring provides a template to simplify the programming API.
- Science of Consciousness: Purification leads to Progress, simplifying the API makes it easier to create and maintain JMS applications