

Lecture 10: Java Message Service

Olivier Liechti
AMT

heig-vd

Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

Multi-Tiered Applications

heig-vd

Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

Frameworks



APIs



servlet

EJB

JDBC



JSP

JAX-RS

JPA



JAX-WS



Java EE
Application Server

JMS

Tools



Jenkins



Selenium Webdriver



Planning

heig-vd

Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

Part 1					Part 2					Part 3					
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

Week 12: JMS (1)

Week 13: JMS (2)

Week 14: Spring framework (1)

Week 15: Spring framework (2)

Presentations
(Tue and Fri)

Project sub. 3

Fri 15.01.2016, 20h

Project, phase 3

- In the last phase, we want to evaluate your **creativity**, your **attention to details** and your **presentation skills** (we want to see that you are **proud** of what you have achieved!).
- Pick **one of the topics** presented on the next slide.
- **Presentations**
 - 20 minutes presentation + demo
 - 10 minutes Q&A
- For the topics where you improve one aspect of your platform, make it very clear what was the status of your project at the end of phase 2.

#1	Evaluate the performance and scalability impact of a making the /events API asynchronous .
#2	Implement and demonstrate a more sophisticated rule engine for your platform.
#3	Implement additional and more sophisticated UI widgets for your platform.
#4	Build a complete demonstration system and "gamify" the application of your choice (e.g. GitHub, instagram, etc.)
#5	Build a UI to enable the interactive configuration of rules.

Message Oriented Middleware

- **Message Oriented Middleware (MOM)**

- What is MOM and when to use it?
- Point to Point vs. Publish-Subscribe

- **Java Message Service API**

- Producing messages
- Consuming messages
- JMS and transactions

- **Enterprise Integration Patterns**

- **JMS & transactions**

Message Oriented Middleware

What is MOM?

HR System

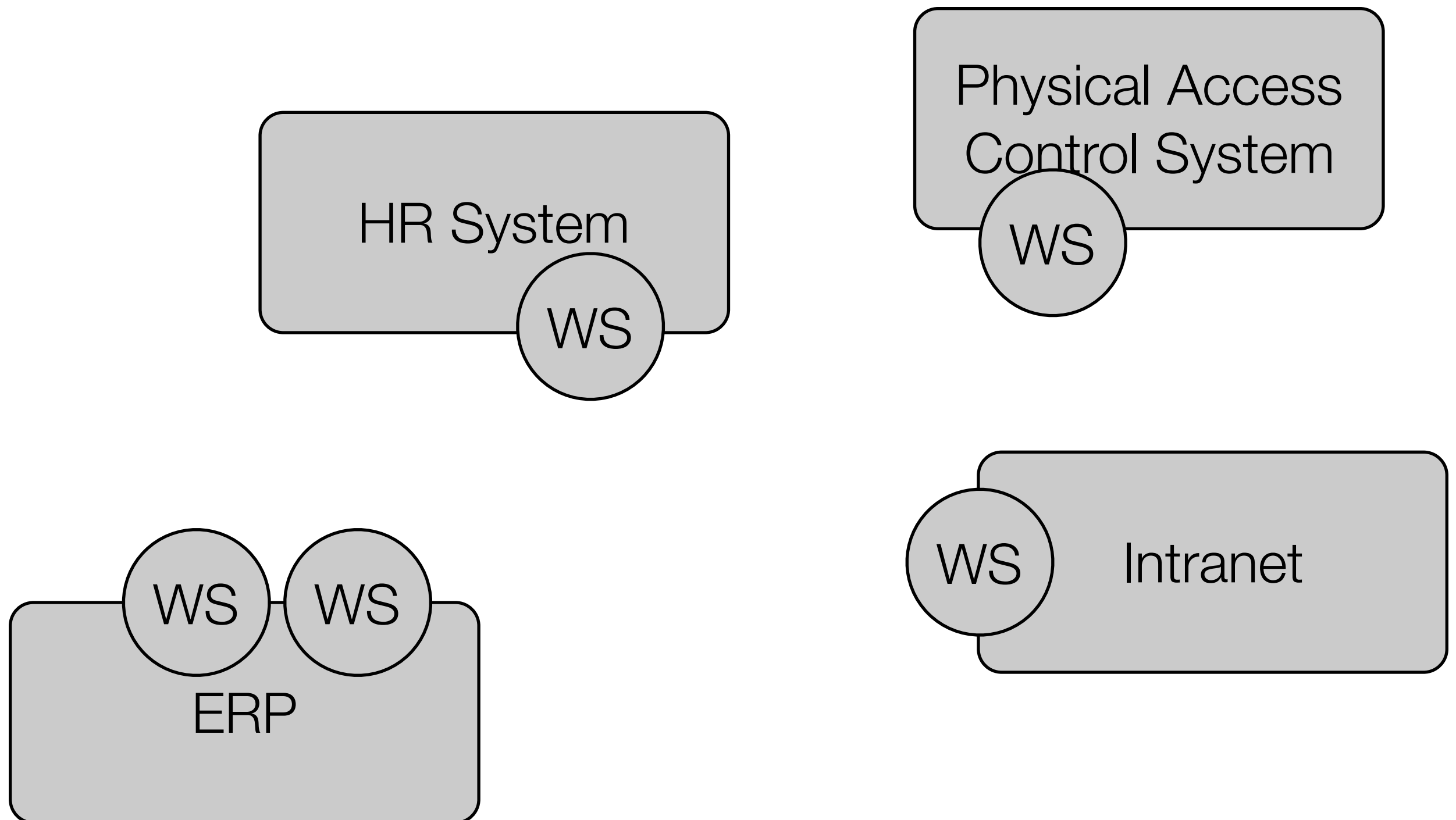
Physical Access
Control System

How do I
integrate these
systems?

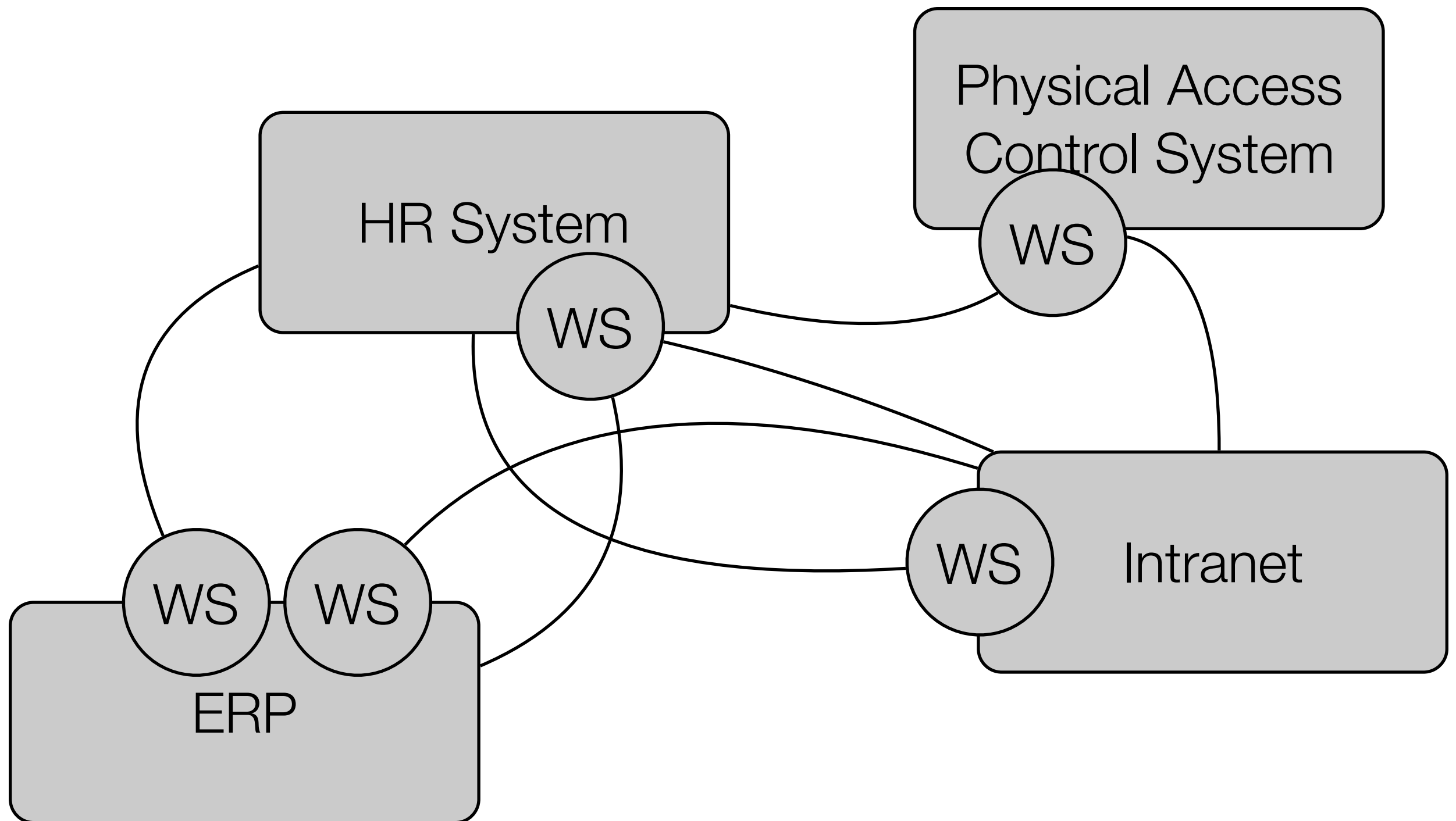
ERP

Intranet

Synchronous, one-to-one integration?



Synchronous, one-to-one integration?



Synchronous, one-to-one integration?

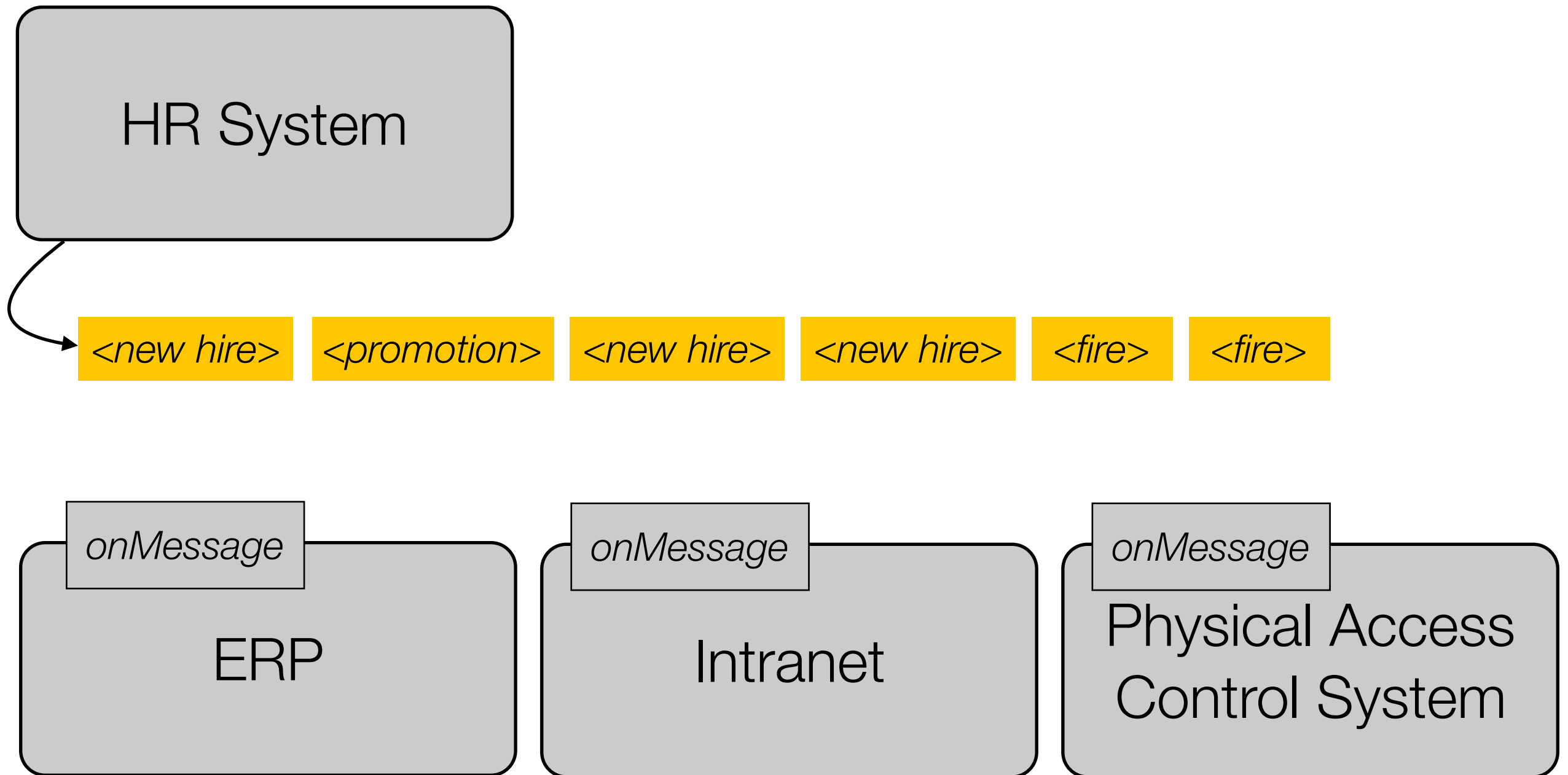


http://www.flickr.com/photos/russelljsmith/448298603/sizes/l/#cc_license

Message Oriented Middleware

- A MOM provides the **infrastructure** for exchanging **messages** between heterogeneous systems.
- Using a MOM makes it possible to integrate the systems in a **loosely coupled** manner.
 - Two systems integrated through a MOM do not know/see/talk directly. They only share a **common message format**.
- Using a MOM means that the integration patterns are **asynchronous**.
 - Two systems integrated through a MOM do not have to be up and running at the same time.

What is MOM?



What do we mean by “infrastructure”?

- A MOM provides the “**plumbing**” for integrating systems.
- We want solid plumbing - we hate leaks!
- This means that a MOM has to provide:
 - reliable message delivery
 - transactions
 - scalability
 - availability
 - security
- You also have tools to manage and configure the MOM.



Messaging Models

Point to Point

1 consumer

Publish-Subscribe

n consumers

Point to Point

- There are message **producers** and message **consumers**.
- Producers post messages in **queues**.
- Consumers read messages from **queues**.
- Every message is consumed **only once**, by one consumer.
- It is possible to **connect several consumers to the same queue** (which enables load balancing!)

Publish-Subscribe

- There are message **producers** and message **consumers**.
- Consumers subscribe to **topics**.
- Producers publish messages on **topics**.
- Every message published on a topic is **delivered to all consumers** who have subscribed to the topic.
- Very often, if a consumer goes away for some time, he will not receive the messages published during this period.
- It is however possible to use **persistent topics**: in that case, when the consumer comes back, it will receive the messages published while it was away.

Message Oriented Middleware & Java: the Java Message Service

What is JMS?

JDBC

JMS

RDBMS

MOM

What is JMS?

- JMS is an API that applications can use to **interact with a MOM**.
- JMS exposes **abstractions** related to point-to-point and publish-subscribe communication models:
 - Message
 - Queue
 - Topic
 - MessageConsumer, Message Producer
 - Connection, Session
- You can use different types of messages: text messages, serialized java objects, etc.
- JMS is one of the most stable Java APIs: version 1.1 dates from 2002.
- JMS is part of Java EE: every Java EE application server must include a MOM with a JMS interface.

Using JMS (1)

- JMS can be used in **standalone Java applications**.
- You include a **.jar** file provided by the MOM vendor in your **classpath**.
- In **your code** (with JMS 1.1), you:
 - Open a **connection** with the messaging service.
 - Initiate a **session**
 - Create a message **producer** (or a **consumer**)
 - **Create** and **send** (or **receive**) **messages**.
- In your code (with JMS 2.0), you:
 - Create a **JMS context**
 - Create a message **producer**
 - Send **messages** (or receive) messages

Sending a message with JMS 1.1

```
public void sendMessage(ConnectionFactory connectionFactory, Queue queueString text) {  
    try {  
        Connection connection = connectionFactory.createConnection();  
        try {  
            Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);  
            MessageProducer messageProducer = session.createProducer(queue);  
            TextMessage textMessage = session.createTextMessage(text);  
            messageProducer.send(textMessage);  
        } finally {  
            connection.close();  
        }  
    } catch (JMSException ex) {  
        // handle exception (details omitted)  
    }  
}
```

Sending a message with JMS 1.1

```
public void sendMsg(ConnectionFactory connectionFactory, Queue queue, String text) {  
    try (JMSContext context = connectionFactory.createContext());{  
        context.createProducer().send(queue, text);  
    } catch (JMSRuntimeException ex) {  
        // handle exception (details omitted)  
    }  
}
```

Receive a message with JMS 1.1

```
public String receiveMessage(ConnectionFactory connectionFactory, Queue queue) {  
    String body=null;  
    try {  
        Connection connection = connectionFactory.createConnection();  
        try {  
            Session session =connection.createSession(false,Session.AUTO_ACKNOWLEDGE);  
            MessageConsumer messageConsumer = session.createConsumer(queue);  
            connection.start();  
            TextMessage textMessage = (TextMessage)messageConsumer.receive();  
            body = textMessage.getText();  
        } finally {  
            connection.close();  
        }  
    } catch (JMSException ex) {  
        // handle exception (details omitted)  
    }  
    return body;  
}
```


Receive a message with JMS 2.0

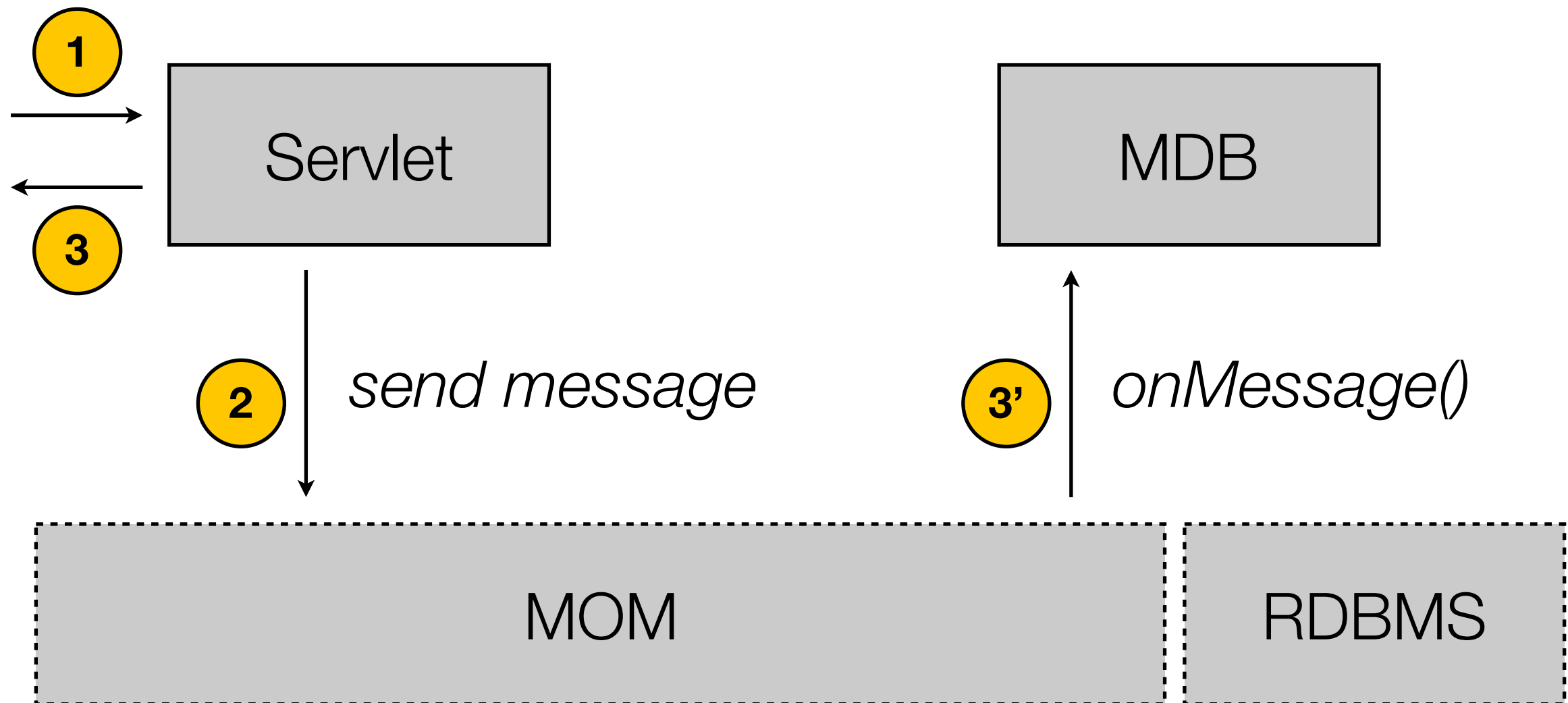
```
public String receiveMessage(ConnectionFactory connectionFactory, Queue queue){
    String body=null;
    try (JMSContext context = connectionFactory.createContext());{
        JMSConsumer consumer = session.createConsumer(queue);
        body = consumer.receiveBody(String.class);
    } catch (JMSRuntimeException ex) {
        // handle exception (details omitted)
    }
    return body;
}
```

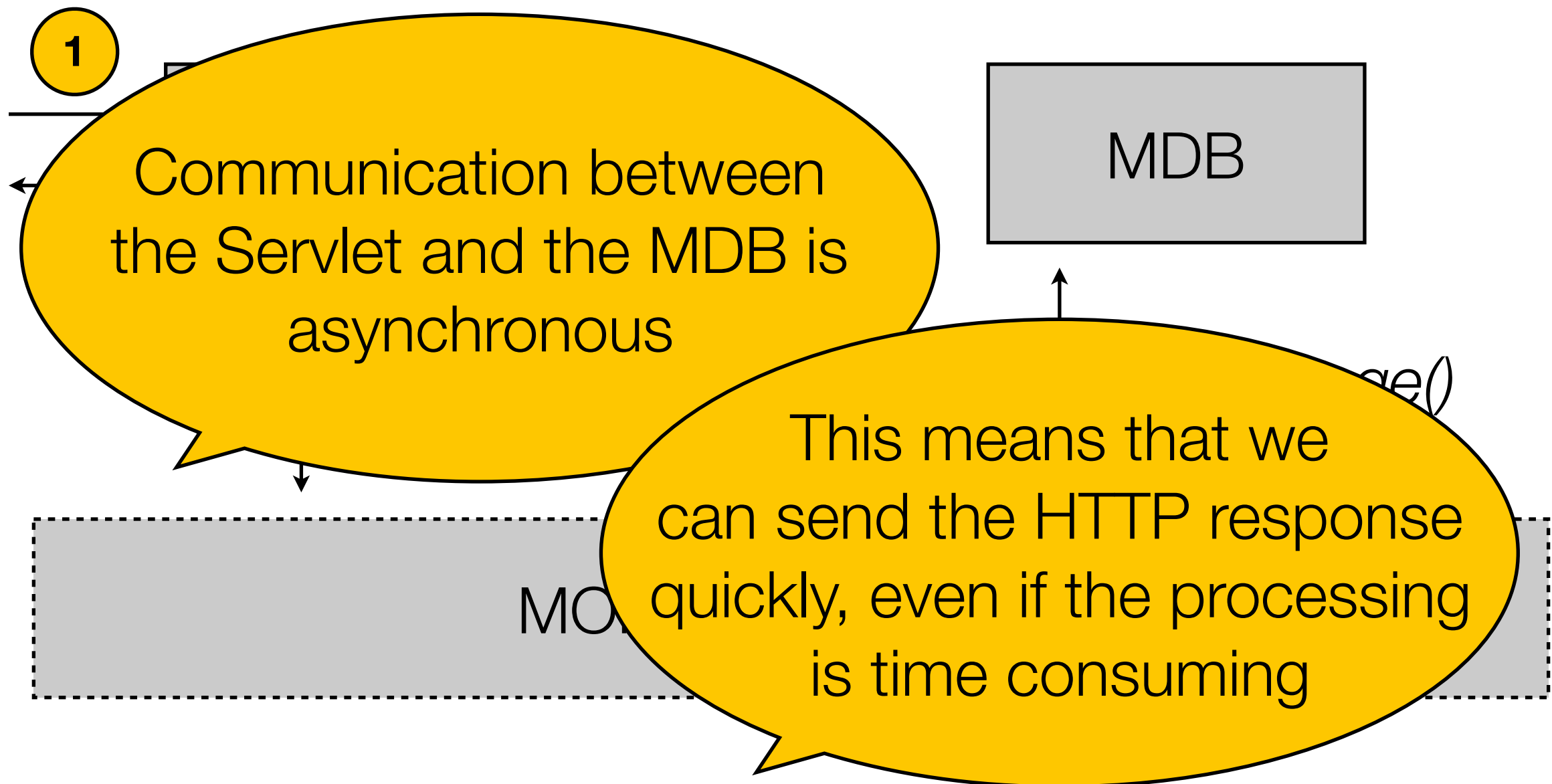
<http://www.oracle.com/technetwork/articles/java/jms20-1947669.html>

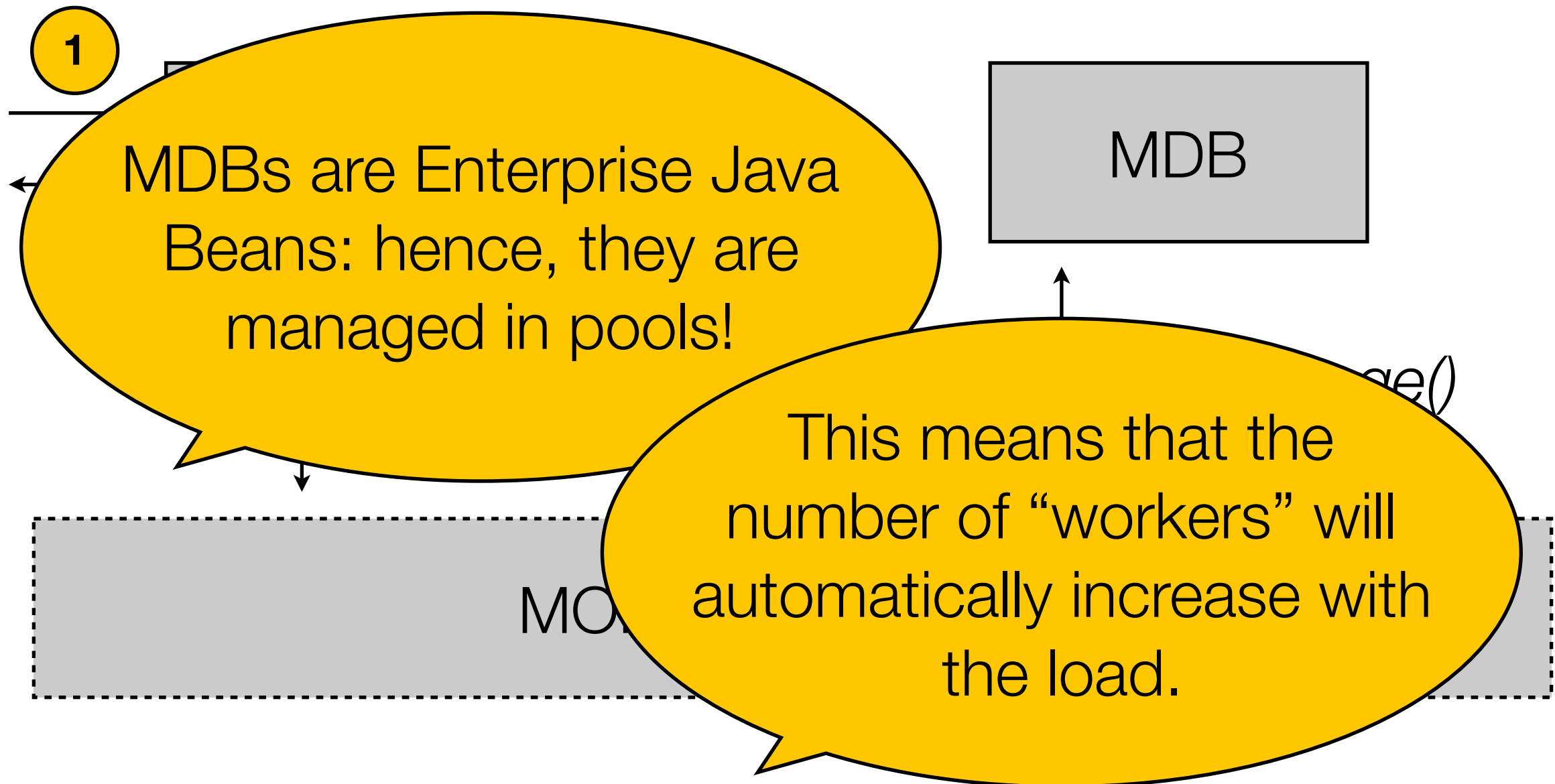
Using JMS with Java EE

- JMS can be used in **Java EE applications**.
- You often produce messages from components such as **servlets, session beans, managed beans**, etc.
- You may also consume messages from these types of components, BUT you generally use a particular type of EJB for that: the **Message Driven Beans**.
- A Message Driven Bean (**MDB**) must implement a `onMessage(Message m)` method.
- With JMS, you can exchange messages:
 - **Within** a single application (e.g. servlet to MDB)
 - **Between** applications (e.g. web front-end to computation back-end)

JMS & Java EE







Sending messages from a SLSB (JMS 1.1)

```
package ch.heigvd.osf.jms;

import javax.annotation.Resource;
import javax.ejb.Stateless;
import javax.jms.*;

@Stateless
public class MsgProducerBean implements MsgProducerRemote {

    @Resource(mappedName = "jms/MsgConsumerBeanFactory")
    private ConnectionFactory msgConsumerBeanFactory;

    @Resource(mappedName = "jms/MsgConsumerBean")
    private Topic msgConsumerBean;

    public void sendMsg() {
        try {
            Connection connection = msgConsumerBeanFactory.createConnection();
            Session session = connection.createSession(true, 0);
            MessageProducer messageProducer = session.createProducer(msgConsumerBean);
            TextMessage tm = session.createTextMessage();
            tm.setText("MESSAGE TEXT");
            messageProducer.send(tm);
            connection.close();
        } catch (JMSException ex) {
            ex.printStackTrace();
        }
    }
}
```



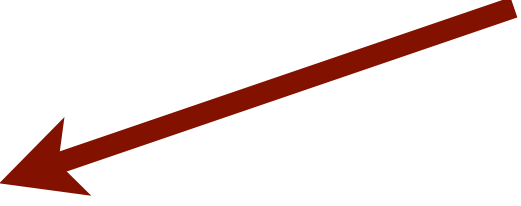
Consuming a message in a MDB (JMS 1.1)

```
package ch.heigvd.osf.jms;

import javax.ejb.ActivationConfigProperty;
import javax.ejb.MessageDriven;
import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.TextMessage;
import javax.jms.MessageListener;

@MessageDriven(mappedName = "jms/MsgConsumerBean")
public class MsgConsumerBean implements MessageListener {

    public void onMessage(Message message) {
        TextMessage msg = (TextMessage)message;
        try {
            System.out.println("MDB MsgConsumerBean received: " + msg.getText());
        } catch (JMSException ex) {
            ex.printStackTrace();
        }
    }
}
```



Send message in Java EE 7

```
@Inject
@JMSConnectionFactory("jms/MyConnectionFactory")
private JMSContext context;

@Resource(lookup="jms/MyQueue")
private static Queue queue;

public void sendMsg(String text) {
    context.createProducer().send(queue, text);
}
```

<https://docs.oracle.com/javaee/7/tutorial/jms-concepts005.htm>

Receive messages in Java EE 7

```
@MessageDriven(activationConfig = {
    @ActivationConfigProperty(propertyName = "destinationLookup",
        propertyValue = "jms/MyQueue"),
    @ActivationConfigProperty(propertyName = "destinationType",
        propertyValue = "javax.jms.Queue")
})
public class SimpleMessageBean implements MessageListener {

    @Resource
    private MessageDrivenContext mdc;

    static final Logger logger = Logger.getLogger("SimpleMessageBean");

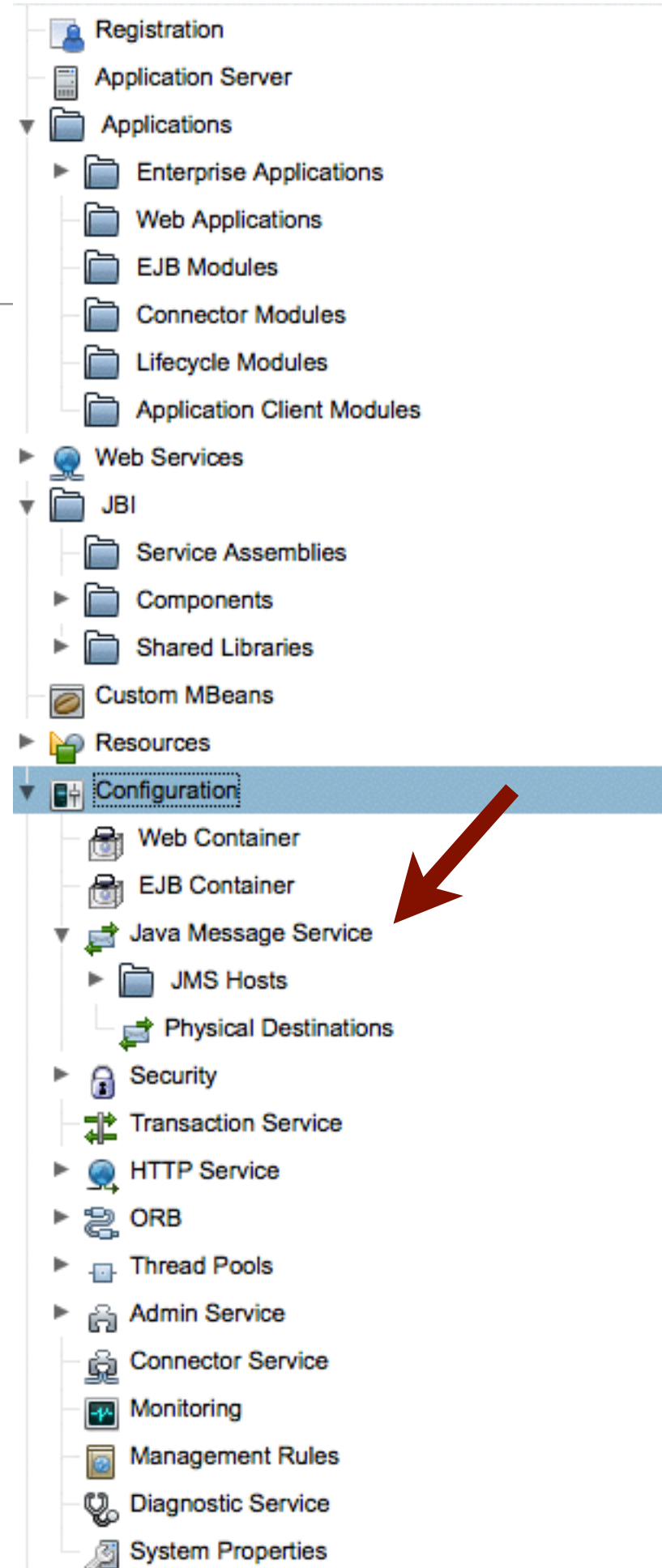
    public void onMessage(Message inMessage) {
        logger.log(Level.INFO, "Message: {0}", inMessage.getBody(String.class));
    }

}
```

<https://docs.oracle.com/javaee/7/tutorial/jms-concepts005.htm>

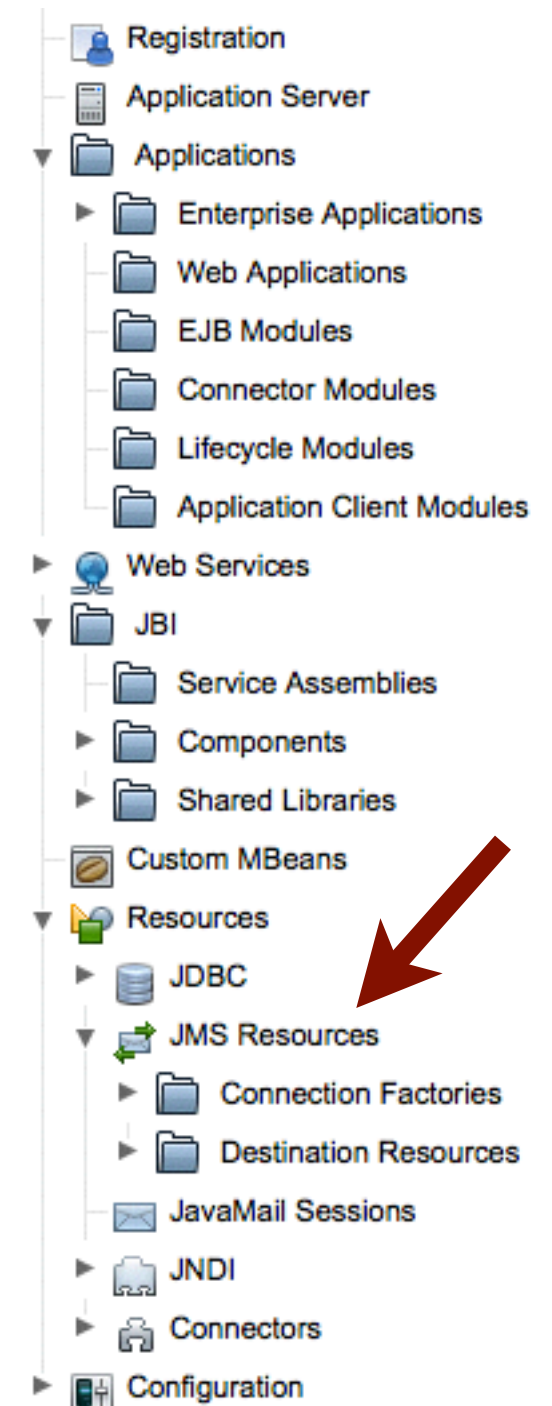
JMS and Glassfish

- The first thing you need to do is to configure the MOM **service**.
- The second thing you need to do is to define JMS resources:
 - Connection factories
 - Queues and/or Topics



heig-vd

Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud



Use cases for JMS

Use cases for JMS

- **Integration of multiple applications**
 - You can use queues and topics to create communication channels between several applications.
 - This even works outside of the Java EE realm: many MOM provide multiple APIs, in multiple languages (e.g. IBM mqseries).

Use cases for JMS

- **Improve response time and scalability with asynchronous processing**
 - In the standard model, an HTTP client sends a request, which is processed synchronously by a servlet, one or more SLSBs, the entity manager, the database, a JSP. This can take a lot of time.
 - **JMS used to be the recommended way to improve this process**, if the client does not need a response immediately. The servlet receives the request, posts a JMS message and sends a response. One MDB then consumes the message and resumes the work (SLBS > entity manager > database).
 - Note: starting with EJB 3.1, it is possible to **invoke SLSB asynchronously** (with a special annotation). This is easier to setup than JMS, but you will encounter legacy applications that still use the pattern.

Use cases for JMS

- **Scalability & application-level load balancing**
 - When you use a **JMS queue**, every message is consumed once and only once.
 - However, it is possible to connect **several consumers** on the same queue. When this is the case, the consumers read messages in a **round-robin** fashion.
 - This feature can be used to implement "compute grids":
 - **task messages** are placed in a JMS queue
 - **several nodes** (app servers on multiple machines) consume these messages and handle the time-consuming work
 - the **results** are either placed in a DB or sent in other JMS messages.

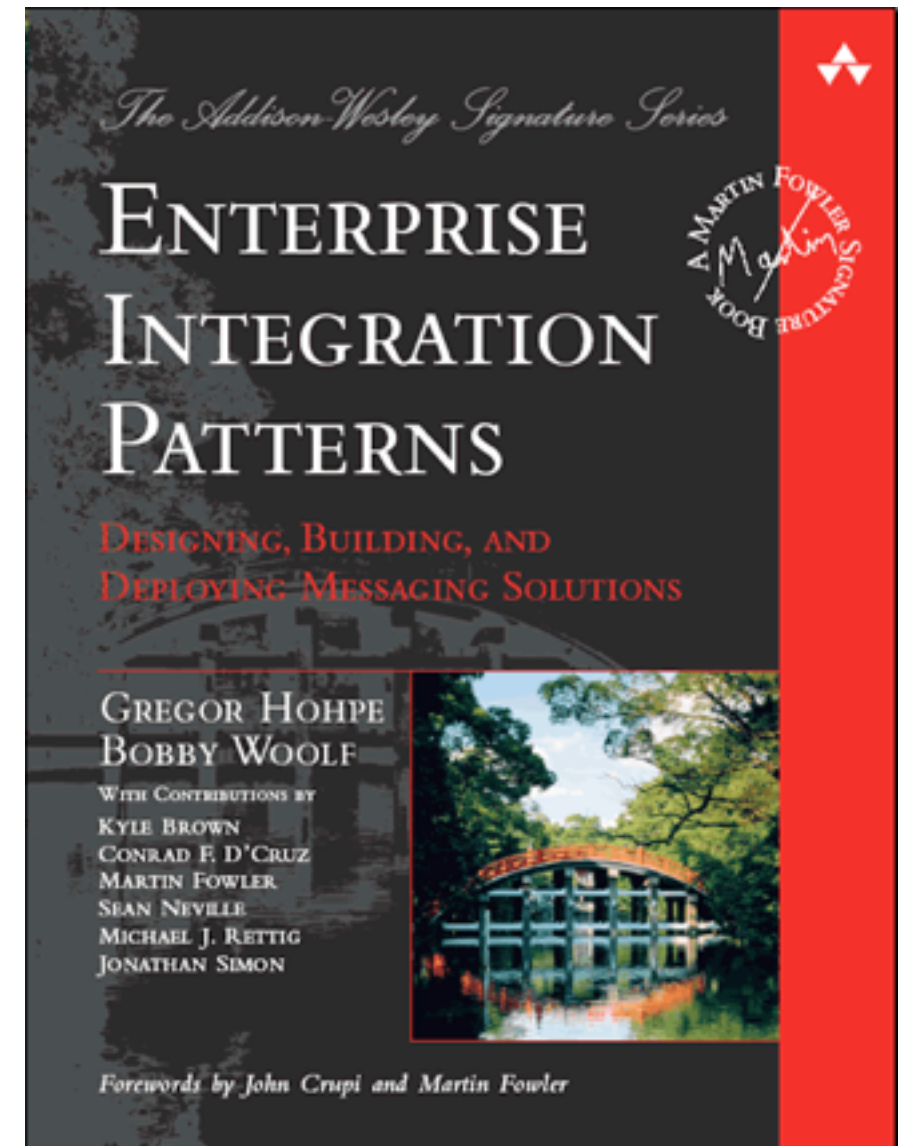
Enterprise Integration Patterns

Enterprise Integration Patterns

heig-vd

Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

- With JMS, we have two core communication mechanisms (point to point, publish-subscribe).
- How do use these communication mechanisms to deal with system integration problems?
- The same problems occur over and over again... patterns have emerged over time and have been documented.
- Some frameworks and tools have been developed to make it easier to instantiate the patterns.

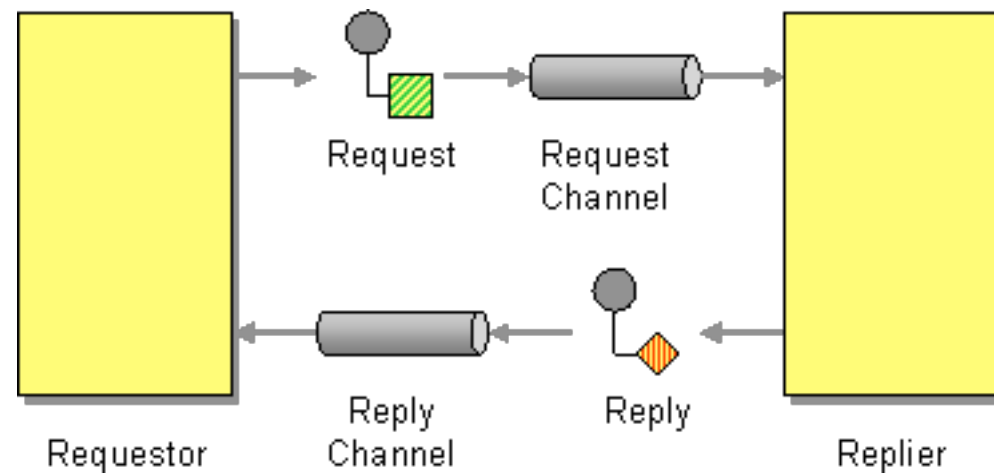


<http://www.eaipatterns.com/>

Enterprise Integration Patterns

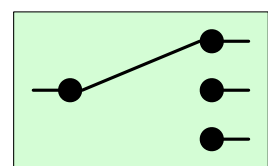
heig-vd

Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

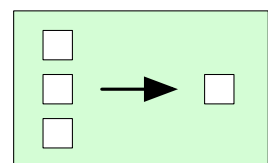


When two applications communicate via Messaging, the communication is one-way. The applications may want a two-way conversation. When an application sends a message, how can it get a response from the receiver?

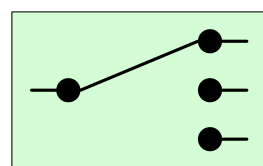
Messaging Channels



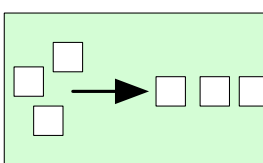
Message Router



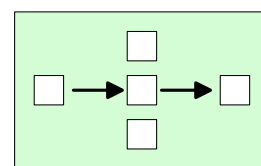
Aggregator



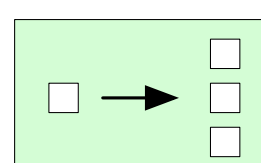
Content Based Router



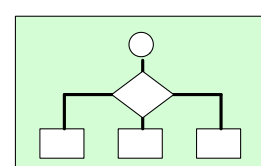
Resequencer



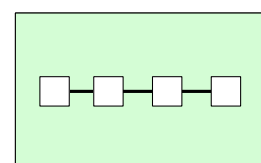
Composed Message



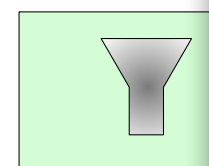
Splitter



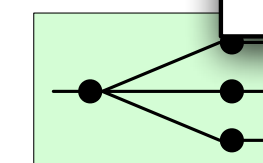
Process Manager



Routing Slip

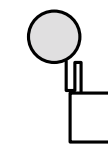


Message Filter

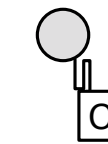


Recipient List

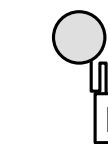
Message Construction



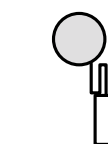
Message



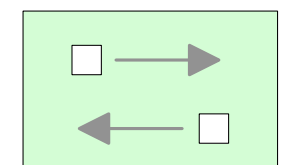
Command Message



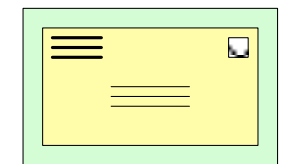
Document Message



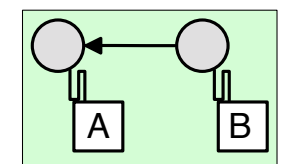
Event Message



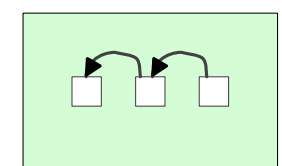
Request Reply



Return Address



Correlation ID



Message Sequence

Apache Camel



Apache Camel is a powerful **open source integration framework** based on known Enterprise Integration Patterns with powerful Bean Integration.

Camel lets you create the Enterprise Integration Patterns to implement routing and mediation rules in either a **Java based Domain Specific Language (or Fluent API)**, via Spring based Xml Configuration files or via the Scala DSL. This means you get smart completion of routing rules in your IDE whether in your Java, Scala or XML editor.

Spring Integration



Spring Integration is a new addition to the Spring portfolio. It provides an extension of the Spring programming model to support the well-known Enterprise Integration Patterns while building on the Spring Framework's existing support for enterprise integration.

It enables simple messaging within Spring-based applications and integrates with external systems via simple adapters. Those adapters provide a higher-level of abstraction over Spring's support for remoting, messaging, and scheduling.

Spring Integration's primary goal is to provide a simple model for building enterprise integration solutions while maintaining the separation of concerns that is essential for producing maintainable, testable code.

JMS and transactions

```
transaction.start();
```

```
accountA.debit(100);
```

```
try {
```

```
    accountB.credit(100);
```

```
} catch (AccountFullException e) {
```

```
    transaction.rollback();
```

```
}
```

```
transaction.commit();
```

transaction.start();

messageService.receiveMessage();

database1.doSomething();

database2.doSomethingElse();

messageService.sendMessage();

transaction.commit();

Distributed Transactions

transaction.start();

messageSer

database1.

database2.

messageSer

commit with messageService

commit with database1

commit database2

transaction.commit();

Distributed Transactions

transaction.start();

messageSer

database1.

database2.

messageSer

commit with messageService

commit with database1

DATABASE2 CRASHES!!!

commit database2

transaction.commit();

How do we deal with system failure at commit time?

Two-Phase Commit Protocol

Two-Phase Commit Protocol

Voting Phase

- The coordinator asks every resource to **prepare a commit**.
- The coordinator ensures that he will be able to commit even after he crashes (writes to a persistent logs)
- Every resource replies either with an **ack** or an **abort**

Completion Phase

- If every resource has sent an ack, then the coordinator sends a commit message to every resource.
- If at least one resource has sent an abort, then the coordinator sends a rollback message to every resource.

Java Transaction API (JTA) specifies standard Java interfaces between a transaction manager and the parties involved in a distributed transaction system: the resource manager, the application server, and the transactional applications.

ACID

JMS

What does it mean
to **rollback** a
transaction when there
is a JMS resource?

- Remember: JMS is probably the most common reason to encounter **distributed transactions**.
- Common examples:
 - A component receives a message via JMS and, as a result of the processing, modifies records in a RDBMS.
 - A component does some processing, modifies records in a RDBMS, and sends a message via JMS when it is done.
- Again... what does it mean to rollback a transaction in these scenarios?

- **For senders**

- **Goal:** if we rollback the transaction, the message should not be seen by consumers on the other side of the JMS transaction.
- **Behavior:** the messages are actually placed in the destination at transaction commit time!

- **For receivers**

- **Goal:** if we rollback the transaction, we don't want to have lost the message. We want to be able to re-process it later.
- **Behavior:** if we rollback, the message is placed back into the destination.

- **Caveats**

- **Poison messages** (what if we can never process a message without producing an exception?)
- **Scalability** (interacting with the JMS broker consumes resources and the release of these resources is not trivial... closing a connection to the broker does not release the connection immediately).