JMS and Transactions in Java EE

Olivier Liechti
Open Source Frameworks
Master Of Science in Engineering (MSE)

MSE | MASTER OF SCIENCE IN ENGINEERING

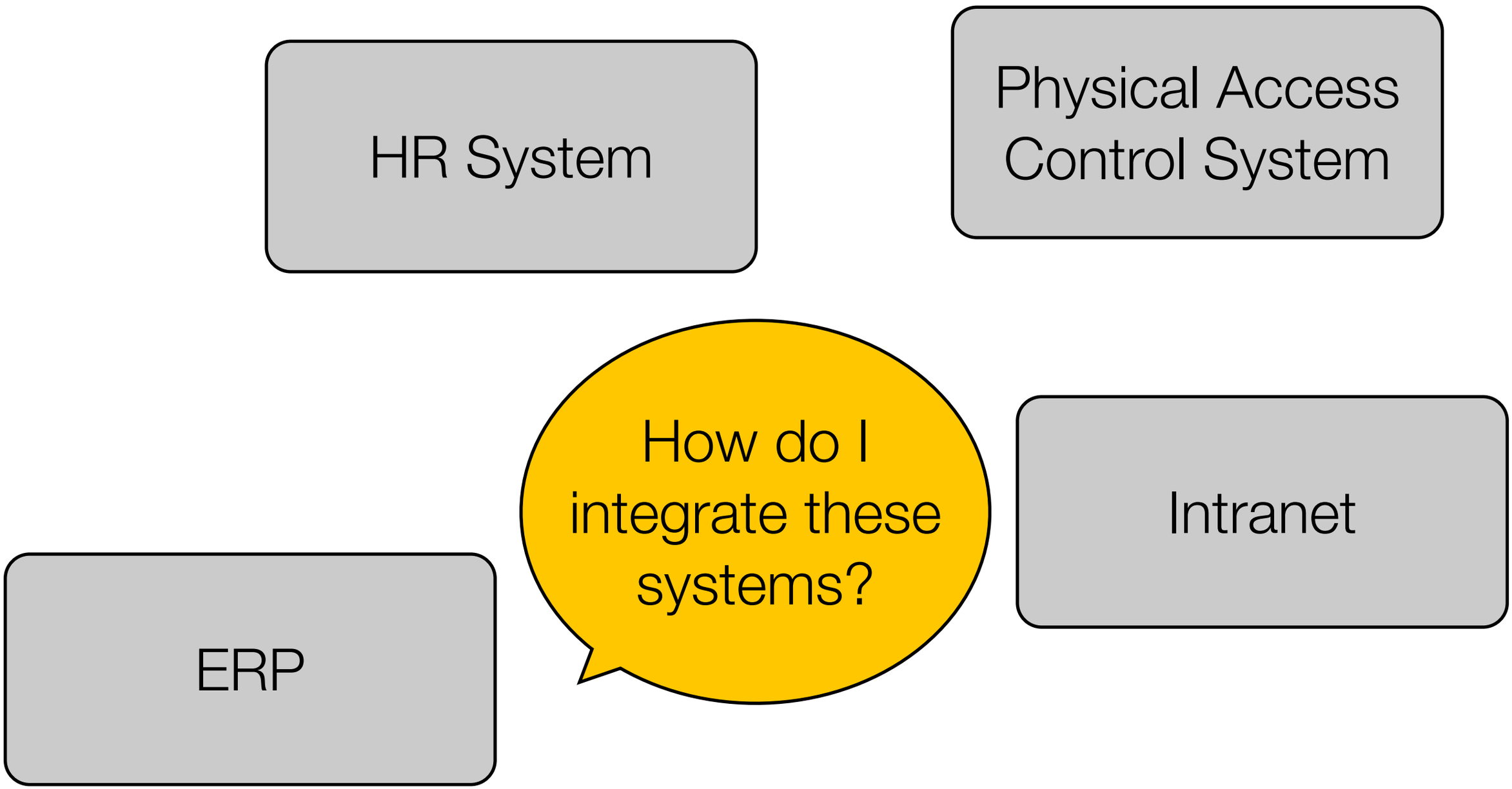
Message Oriented Middleware

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

- **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**
- **Transactions management with Java EE**

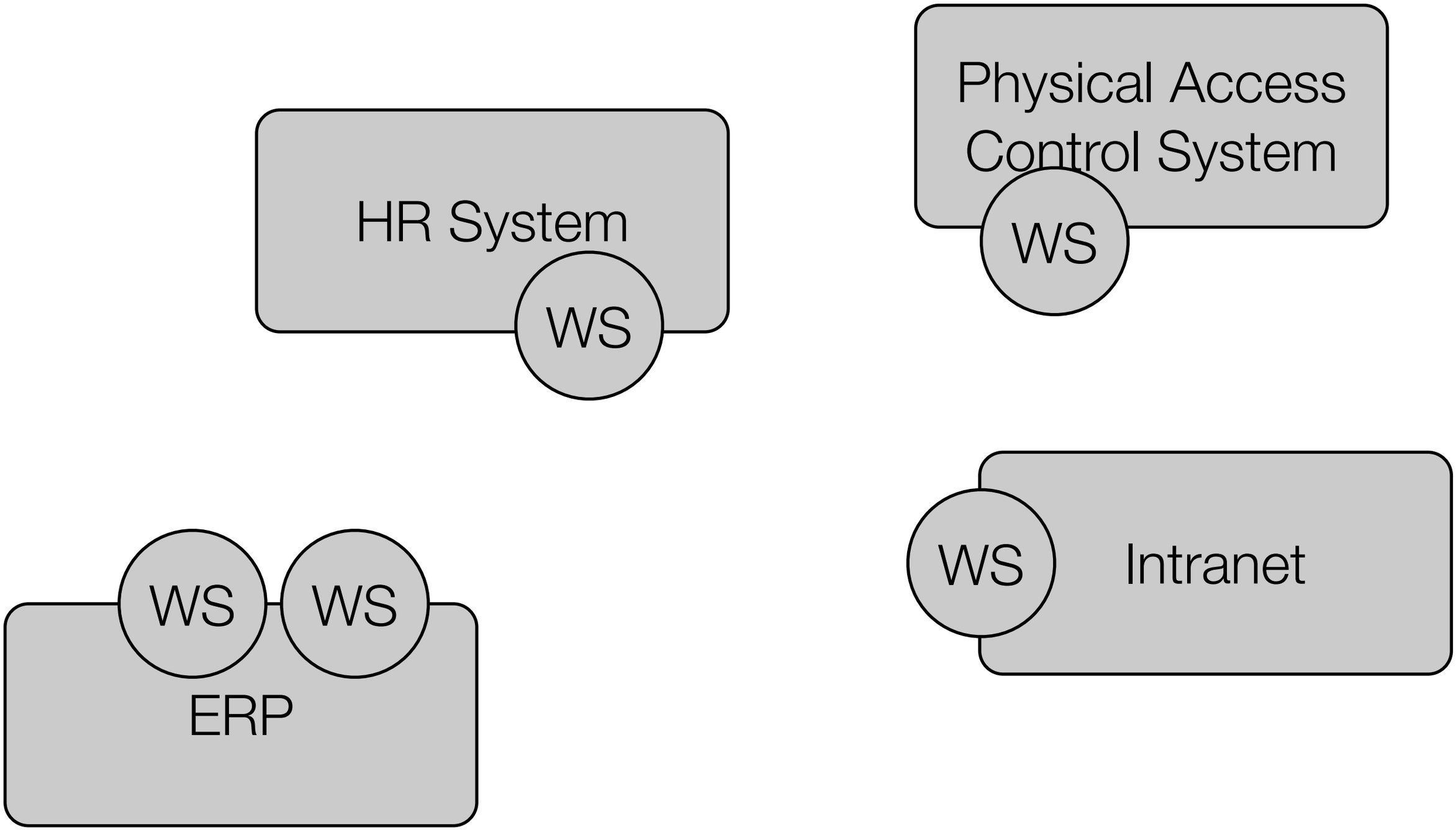
Message Oriented Middleware

What is MOM?


HR System
Physical Access Control System
How do I integrate these systems?
Intranet
ERP

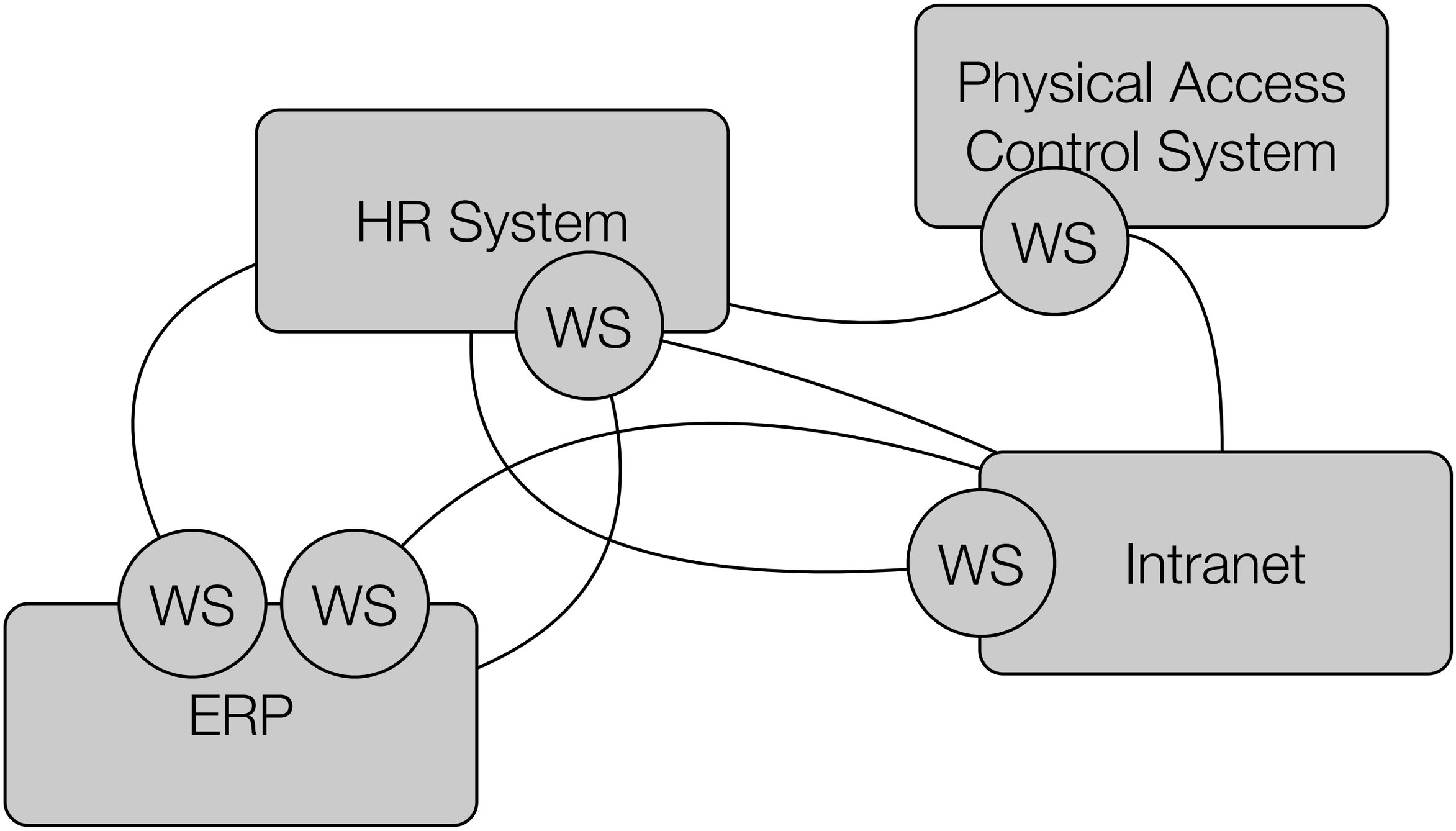
Synchronous, one-to-one integration?

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud



Synchronous, one-to-one integration?

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud



Synchronous, one-to-one integration?

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

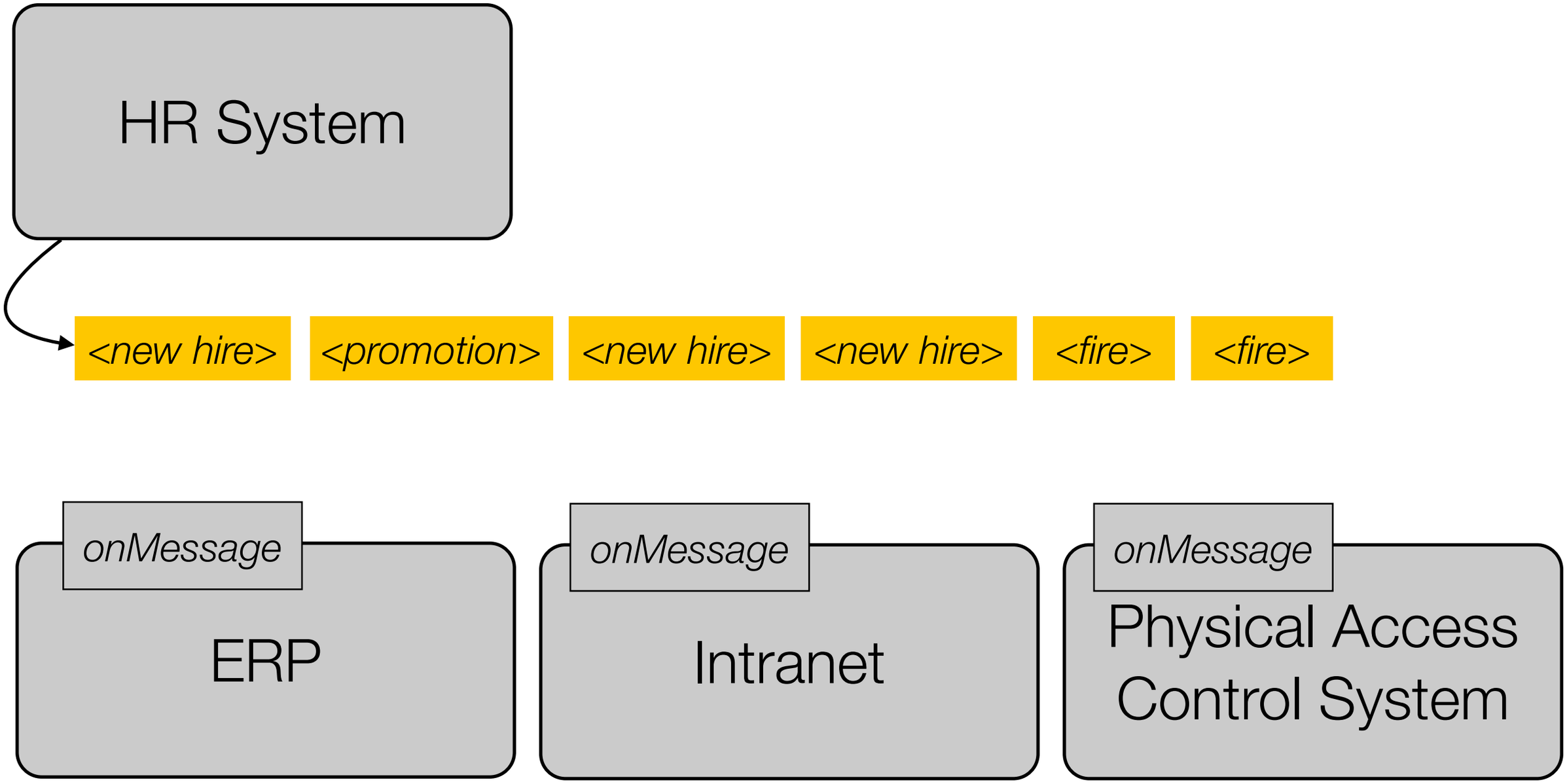
http://www.flickr.com/photos/russelljsmith/448298603/sizes/l/#cc_license

Message Oriented Middleware

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

- 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?


HR System
<new hire>
<promotion>
<new hire>
<new hire>
<fire>
<fire>
onMessage
ERP
onMessage
Intranet
onMessage
Physical Access Control System

What do we mean by “infrastructure”?

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

- 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.

http://www.flickr.com/photos/elsie/185811819/sizes/l/#cc_license

Messaging Models

Point to Point

1 consumer

Publish-Subscribe

n consumers

Point to Point

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

- 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

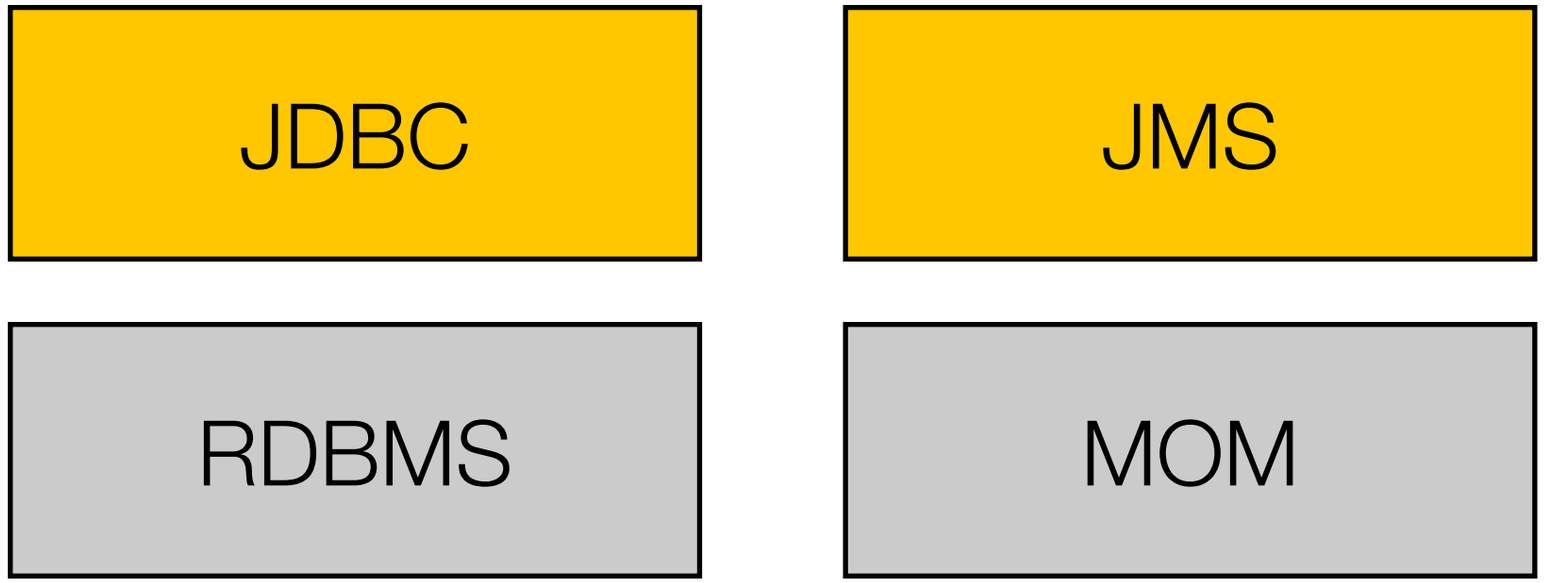
heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

- 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?

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud



What is JMS?

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

- 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)

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

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

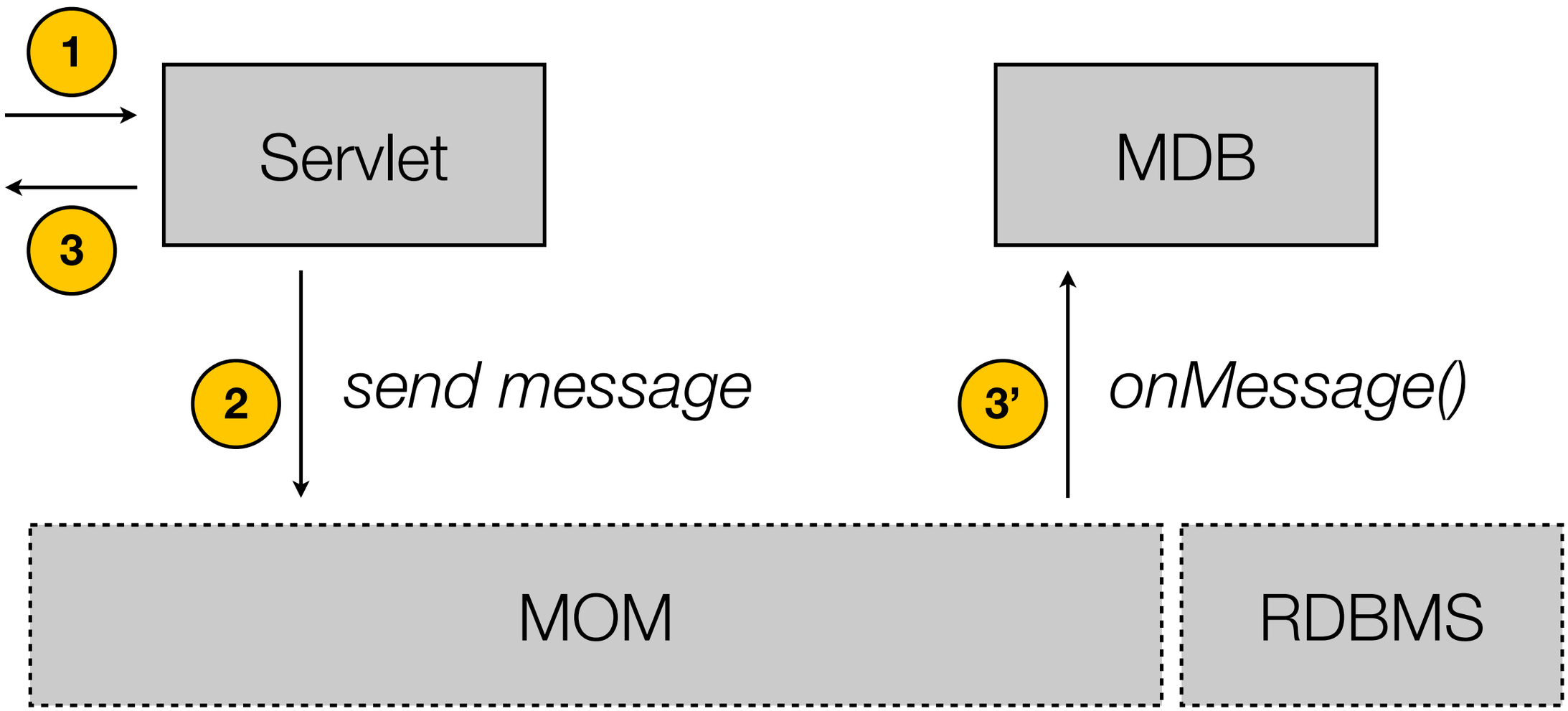
Using JMS (2)

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

- 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.

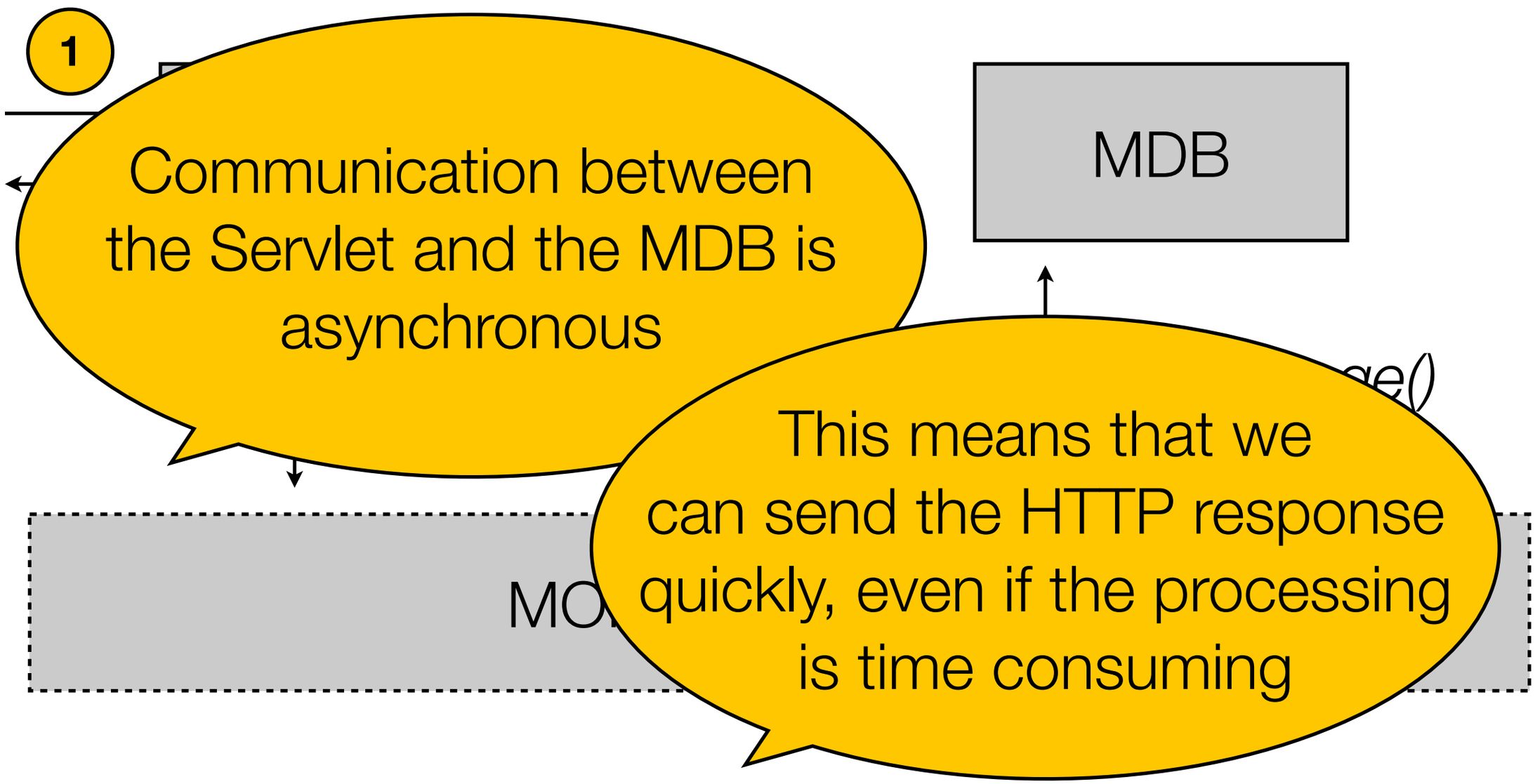
JMS & Java EE

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud



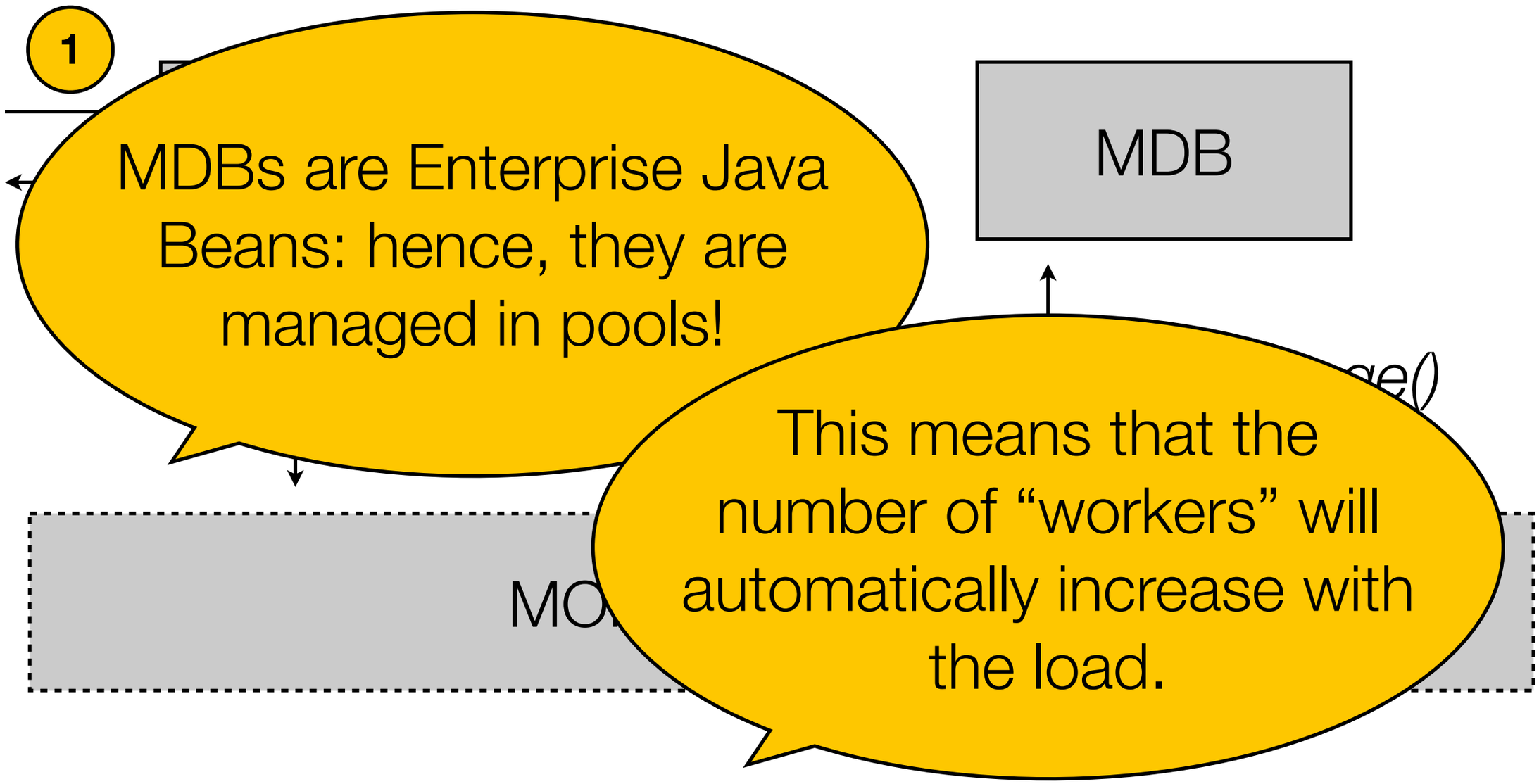
JMS & Java EE

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud



JMS & Java EE

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud



Sending a message from a SLSB

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

```
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

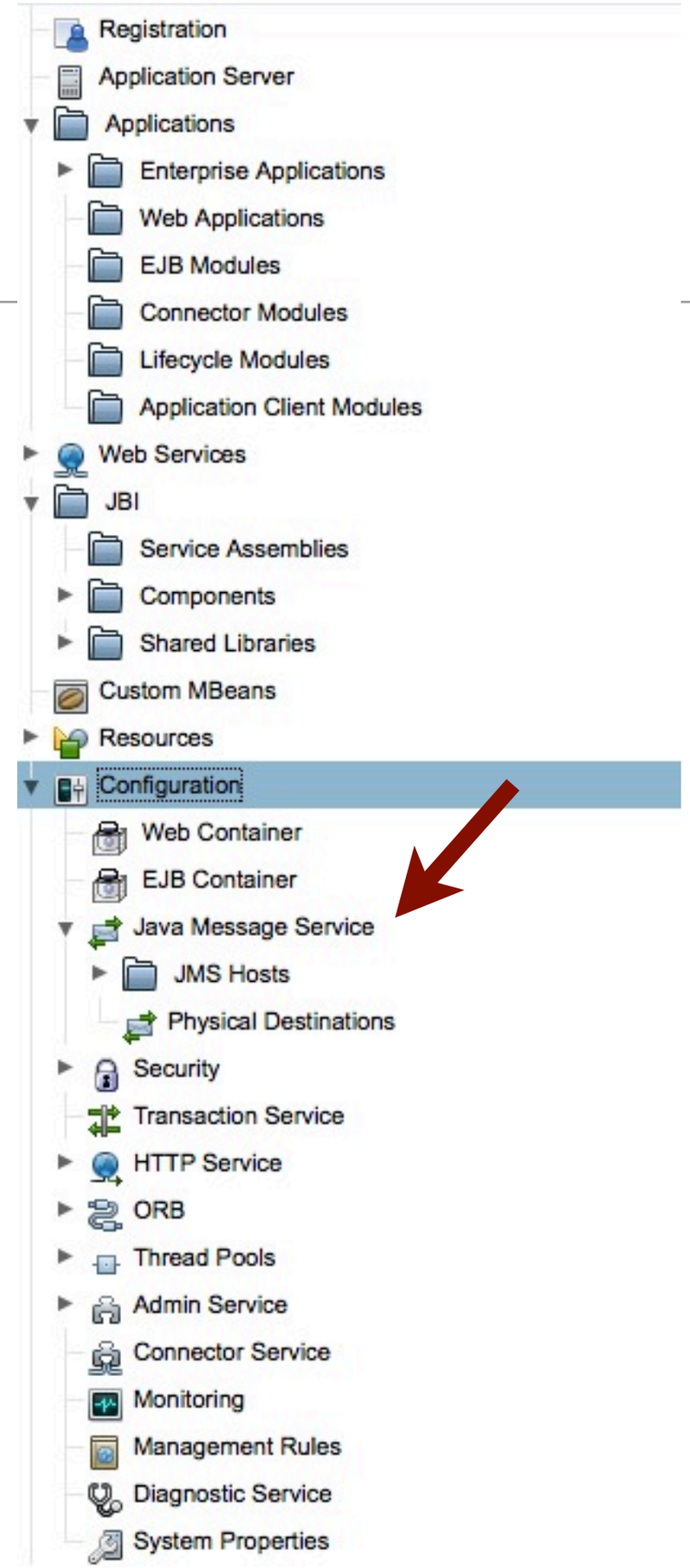
heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

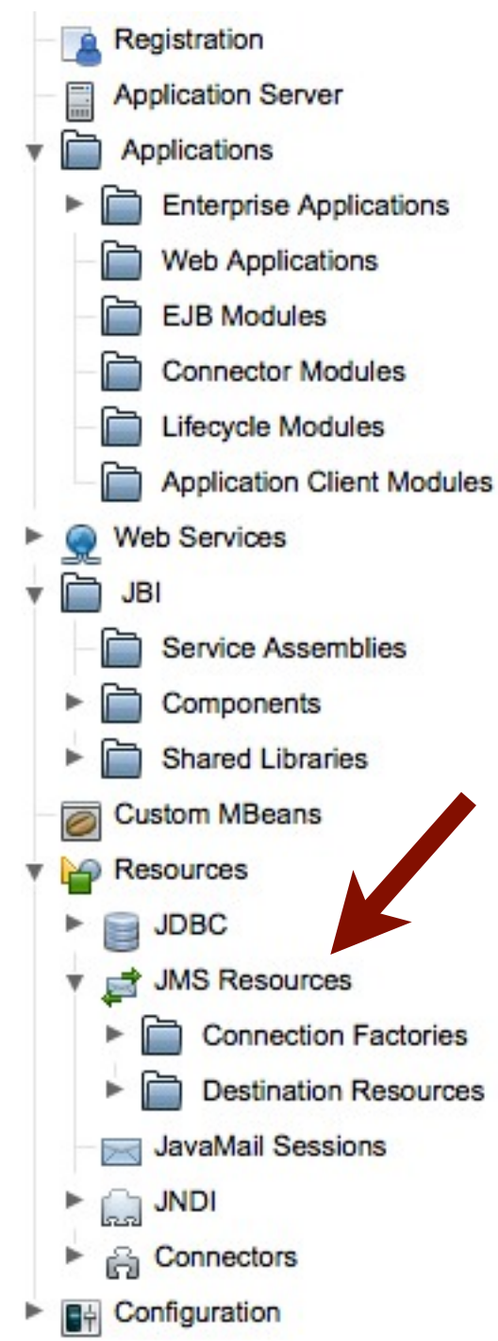
```
package ch.heigvd.osf.jms;

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();
        }
    }
}
```

JMS and Glassfish

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

- 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



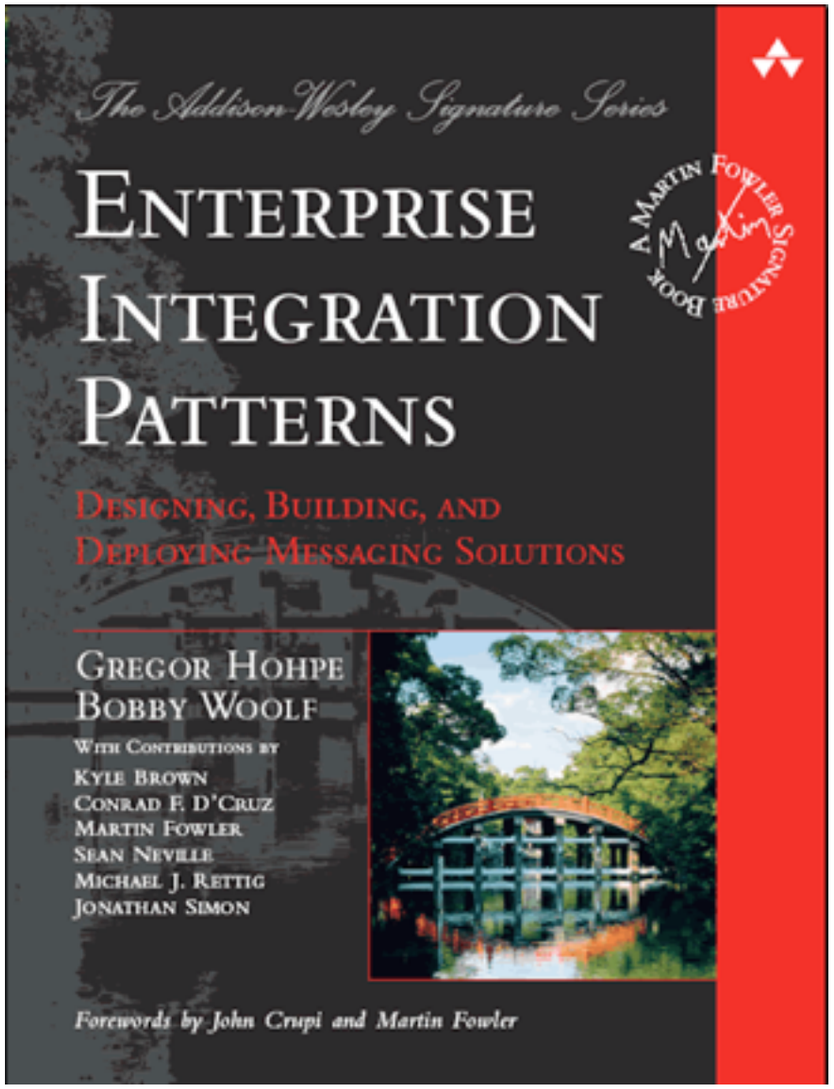


Enterprise Integration Patterns

Enterprise Integration Patterns

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

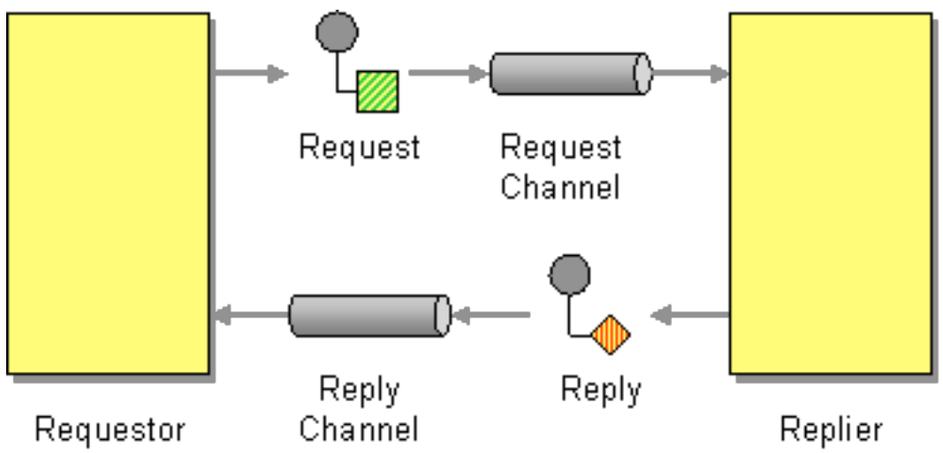
- With JMS, we have two core communication mechanisms (point to point, publish-subscribe).
- How do use these communication mechanisms to deal with system integration problems?
- The same problems occur over and over again... patterns have emerged over time and have been documented.
- Some frameworks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



http://www.eaipatterns.com/

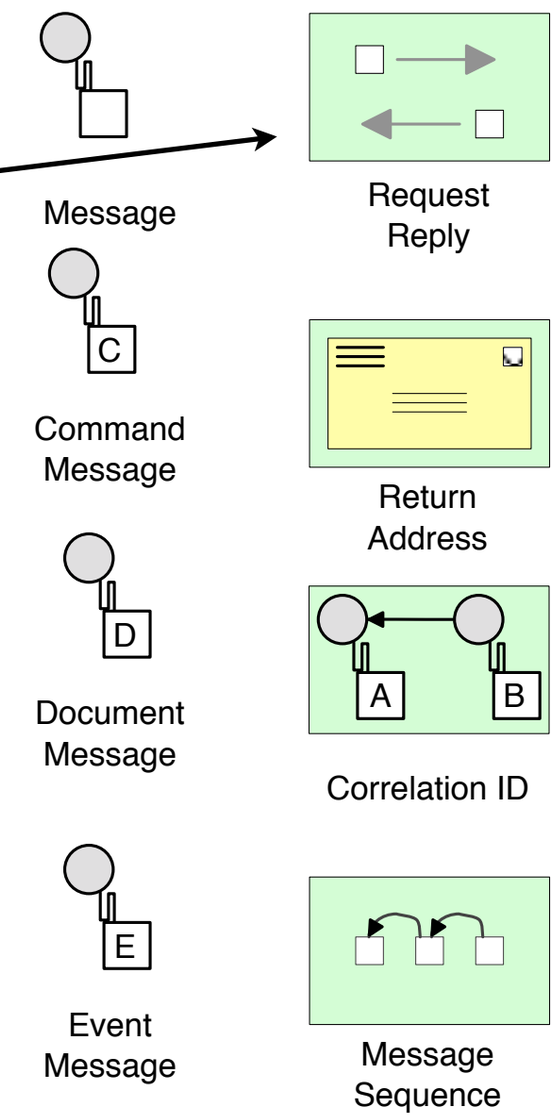
Enterprise Integration Patterns

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

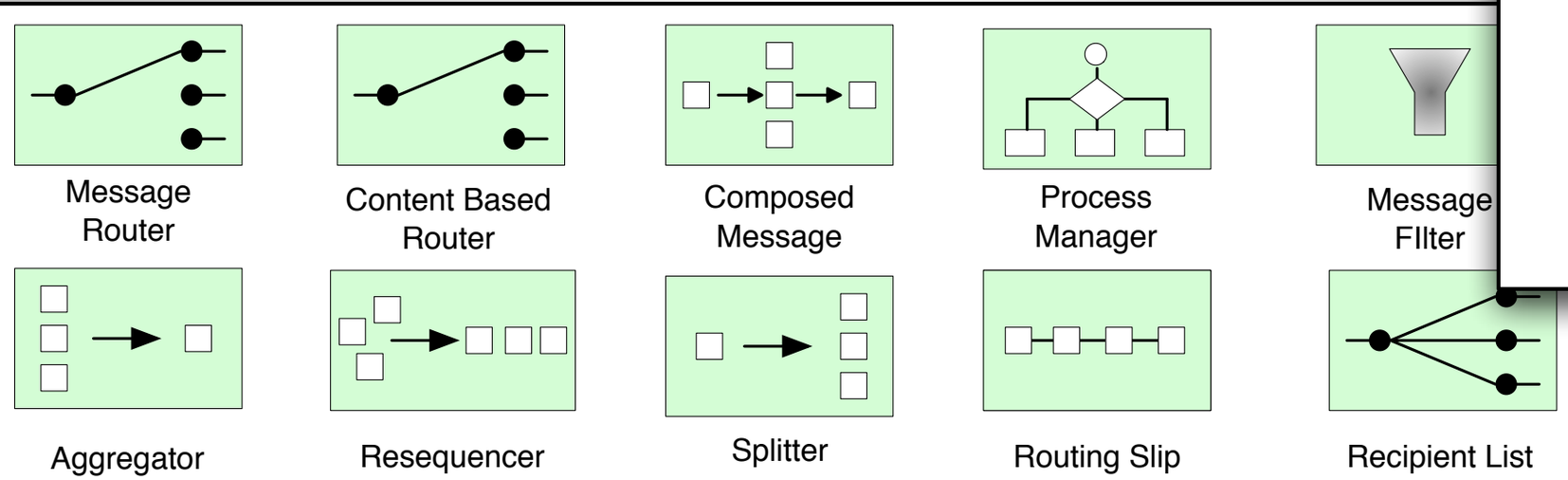


When two applications communicate via Messaging, the communication is one-way. The applications may want a two-way conversation. When an application sends a message, how can it get a response from the receiver?

Message Construction



Messaging Channels



Apache Camel

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

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

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

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.

Transactions

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

- **Transactions**
  - ACID
  - Distributed transactions
- **Transactions in Java EE**
  - Container managed transactions
  - Transactions and annotations
  - Transactions and JMS

Transactions

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

```
transaction.start();

    accountA.debit(100);
    accountB.credit(100);

    transaction.commit();
```

Transactions


```
transaction.start();

accountA.debit(100);
try {
  accountB.credit(100);
} catch (AccountFullException e) {
  transaction.rollback();
}

transaction.commit();
```

ACID

ACID

Atomicity: “all or noting”

ACID

Consistency: “business data integrity”

ACID

Isolation: “deal with concurrent transactions”

ACID

Durability: “once it’s done, it’s done”

A typical enterprise application accesses and stores information in one or more databases. Because this information is critical for business operations, it must be accurate, current, and reliable.

Data integrity would be lost if ***multiple programs*** *were allowed to update the same information simultaneously. It would also be lost if a system that failed while processing a business transaction were to leave the affected data only* ***partially updated****. By preventing both of these scenarios, software transactions ensure data integrity.*

Transactions control the ***concurrent access*** *of data by multiple programs. In the event of a system failure, transactions make sure that after recovery the data will be in a* ***consistent*** *state.*

Distributed Transactions

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

```
transaction.start();

messageService.receiveMessage();
database1.doSomething();
database2.doSomethingElse();
messageService.sendMessage();

transaction.commit();
```

Distributed Transactions

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

```
transaction.start();

messageSer
database1.
database2.
messageSer

transaction.commit();
```

commit with messageService

commit with database1

commit database2

Distributed Transactions

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

```
transaction.start();

messageSer
database1.
database2.
messageSer

transaction.commit();
```

commit with messageService

commit with database1

DATABASE2 CRASHES!!!

commit database2

How do we deal with system failure at commit time?

Two-Phase Commit Protocol

Distributed Transactions: 2PC

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 a nack, then the coordinator sends a rollback message to every resource.

Distributed Transactions & Java

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

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.

Transaction Demarcation

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

```
transaction.start();

accountA.debit(100);
accountB.credit(100);

transaction.commit();
```

Who does the “start” and the “commit” and **where**?

Container Managed Transaction

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

- The EJB container handles calls to `commit` and `rollback`.
- Methods defined on EJBs provide demarcation points.
- This is the **default behavior**.

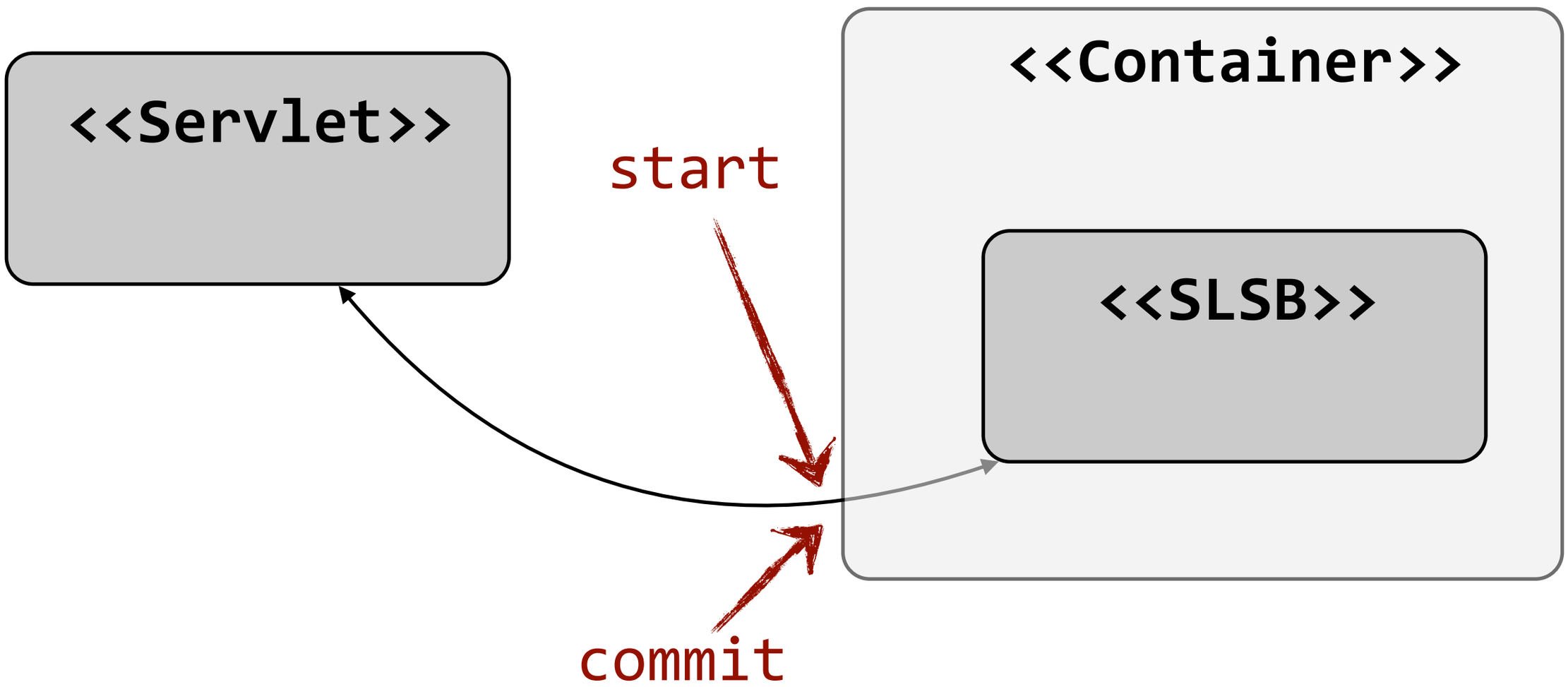
```
<<Servlet>>
CustomerController
 ...
 cm.invoice(29, 2000);
 ...
```

```
<<SLSB>>
CustomerManager

public invoice(
  long id, int amount);
```

Container Managed Transaction

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

- The EJB container handles calls to `commit` and `rollback`.
- Methods defined on EJBs provide demarcation points.
- This is the **default behavior**.



Transaction Scope

What happens when a **client** calls a method on a session bean, which calls a method on a session bean, which calls a method on a session bean, which calls a method on a session bean, which calls a method on a session bean, which calls a method on a session bean, which calls a method on a session bean, which calls a method on a session bean, which **throws an exception**?

Transaction Scope

What h[illegible]hen a **client** calls a method on [illegible]calls a method on a [illegible]ls a method on a session bea[illegible] a method on a session bean, which calls a method on a se[illegible] which calls a method on a se[illegible] method on a sessio[illegible] **exception**?

Everything should be rolled back!

No! Only changes incurred by the last method should be rolled back!

Transaction Scope

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

What h[illegible] when [illegible]

Everything should [illegible] rolled back!

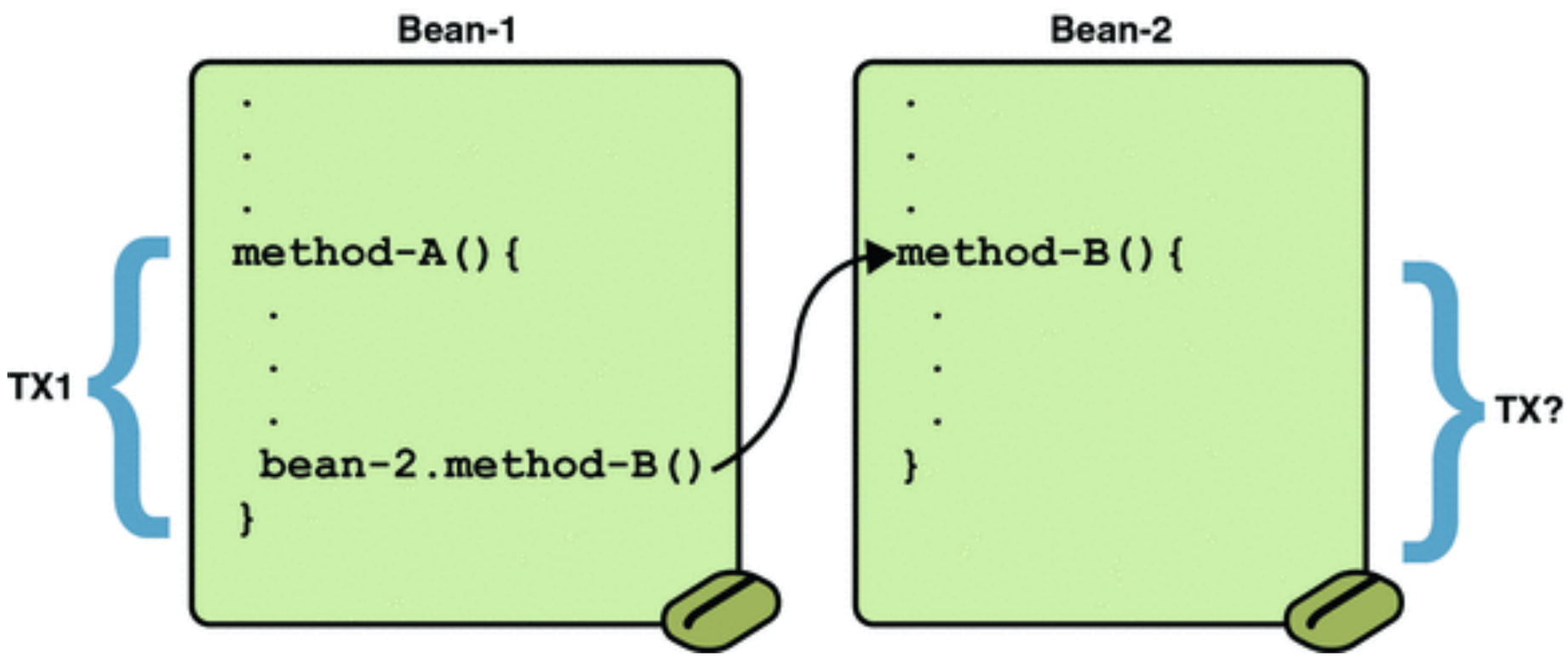
bea[illegible]
which calls a method [illegible]
calls a method on a se[illegible]
method on a session [illegible]
exception?

It is **up to the application** to specify intended behavior. The developer must specify transaction scope, typically with **annotations**.

No! Only changes incurred by the last method should be rolled back!

Transaction Scope

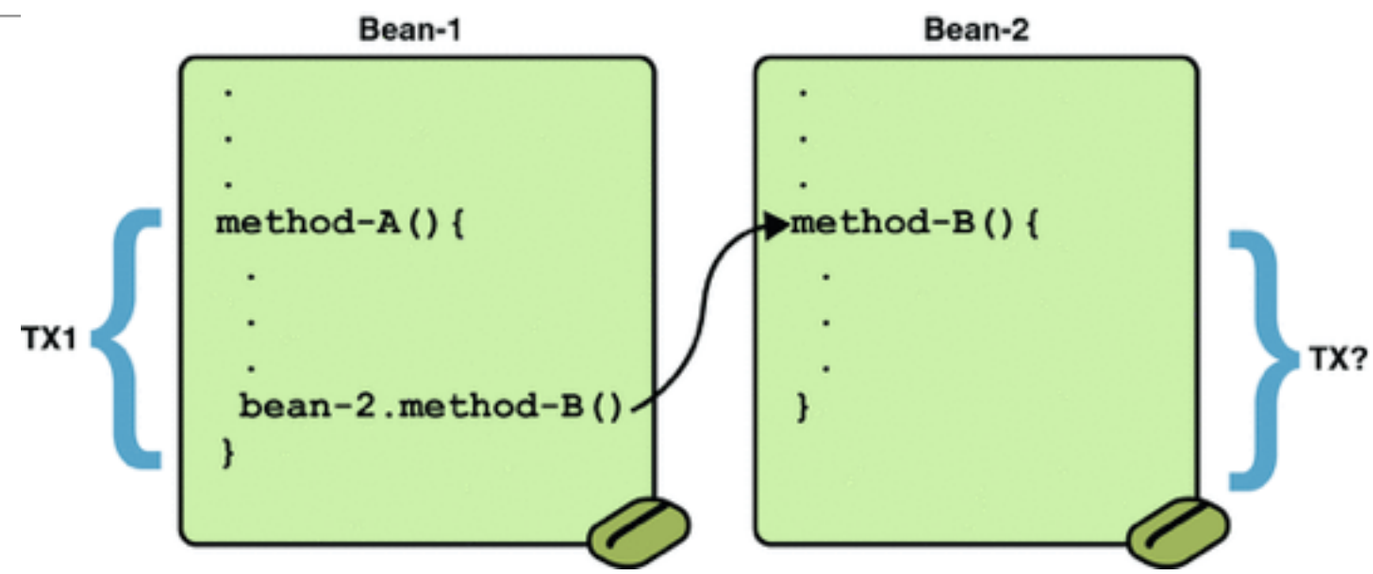
heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud



http://java.sun.com/javaee/5/docs/tutorial/doc/bncij.html

Transaction Scope

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud




```
@TransactionAttribute(NOT_SUPPORTED)
@Stateful
public class TransactionBean implements
Transaction {
...
    @TransactionAttribute(REQUIRES_NEW)
    public void firstMethod() {...}

    @TransactionAttribute(REQUIRED)
    public void secondMethod() {...}

    public void thirdMethod() {...}

    public void fourthMethod() {...}
}
```

Transaction Attribute	Client's Transaction	Business Method's Transaction
Required	None	T2
	T1	T1
RequiresNew	None	T2
	T1	T2
Mandatory	None	error
	T1	T1
NotSupported	None	None
	T1	None
Supports	None	None
	T1	T1
Never	None	None
	T1	Error

Transactions & Exceptions

There are two ways to roll back a container-managed transaction.

First, if a **system exception** is thrown, the container will automatically roll back the transaction.

Second, by invoking the **setRollbackOnly** method of the EJBContext interface, the bean method instructs the container to roll back the transaction. If the bean throws an application exception, the rollback is not automatic but can be initiated by a call to setRollbackOnly.

Note: you can also annotate your Exception class with `@ApplicationException(rollback=true)`

Bean Managed Transactions

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

If you have special needs, you can control the transaction demarcation yourself. You will start and commit the transactions, and possibly rollback them.

To do that, you need to use Bean Managed Transactions.

```
Stateful
@TransactionManagement(BEAN)
public CartSession {
  CartEnt cart;
  @PersistenceContext EntityManager em;
  @Resource UserTransaction ut;

  @PostConstruct public startCart() {
       ut.begin();
       cart = new CartEnt();
  }

  public addItem (String itemid, int qty) {
      em.persist(new CartItem(itemid, qty, cart.getId());
      cart.setItemQuantity(cart.getItemQuantity() + 1);
  }

  public checkOut() {
      em.merge(cart);
      ut.commit();
  }
}
```

JMS and Transactions

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

ACID

JMS

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

JMS and Transactions

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

- 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?

JMS and Transactions

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

- **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).

Questions?