J2E++: Interceptors, MOMs & Java Server Faces

Sébastien Mosser

Lecture #5, 12.04.2018

pictures: sxc.hu

Presentation Layer:
Java Server Faces

among others ...

1

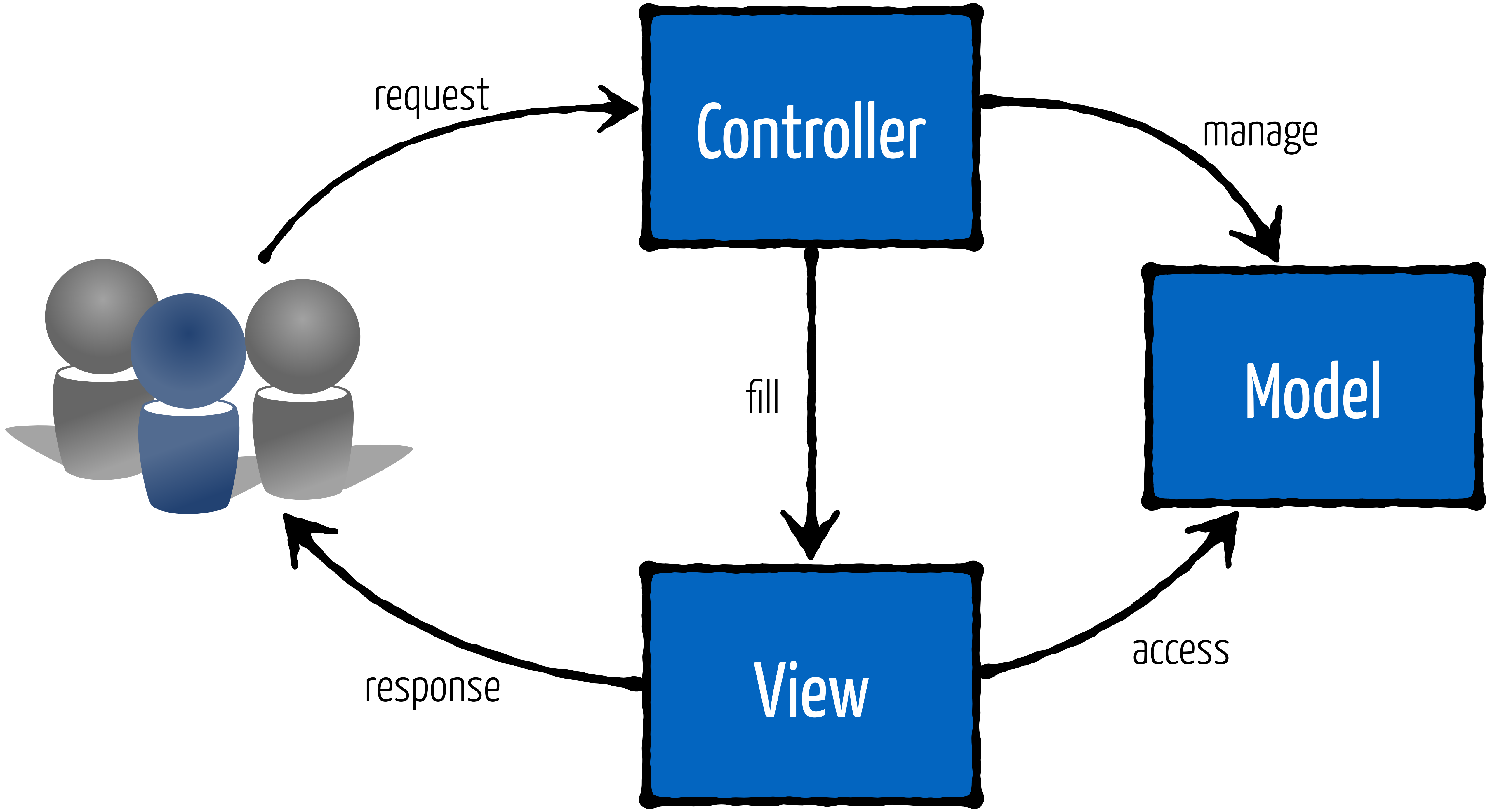
Principles

Artefacts

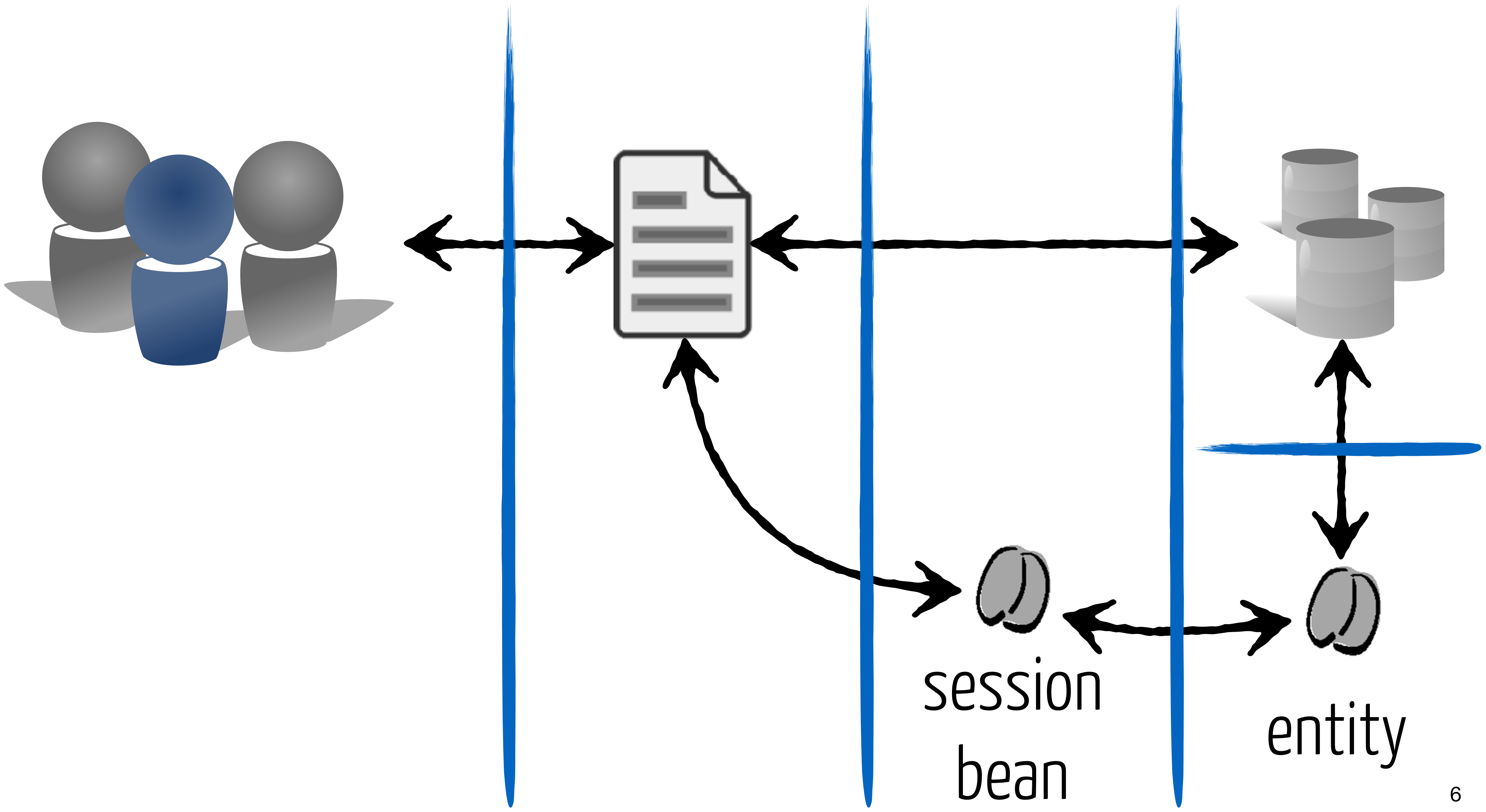
2

1 Principles

Architecture: Model-View-Controller



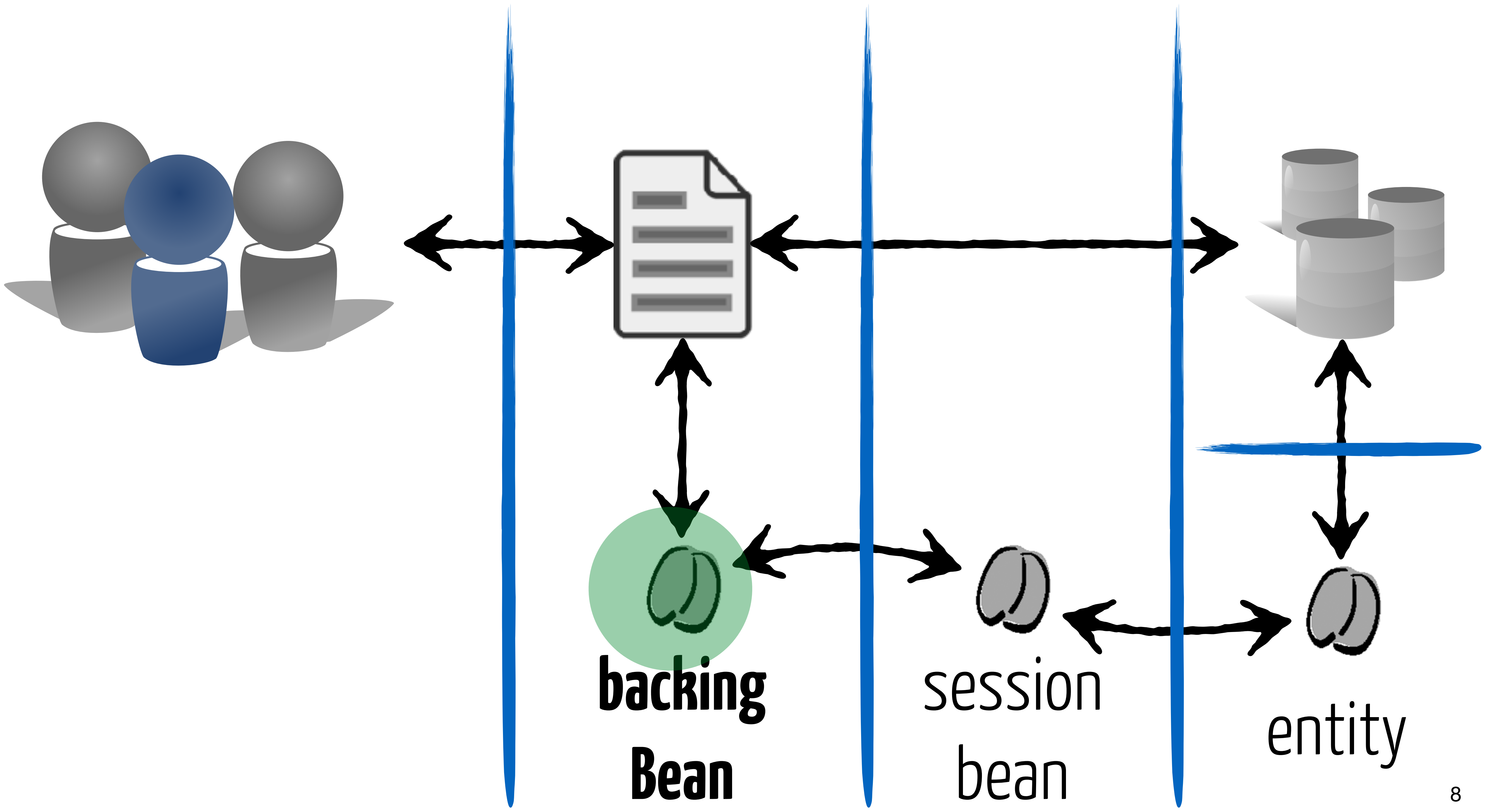
Interacting with the system

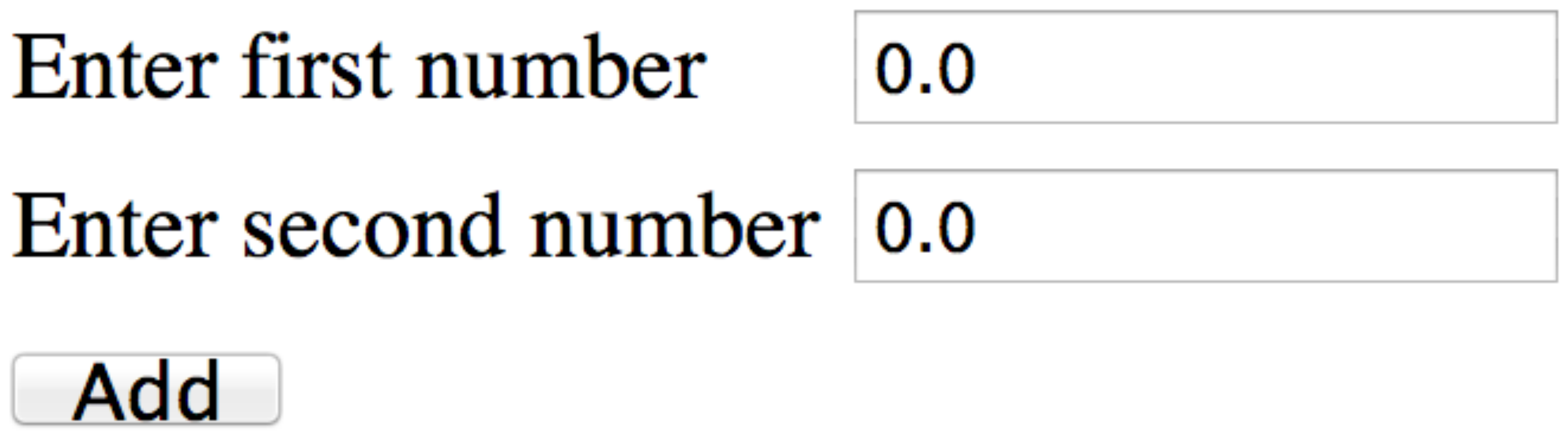


Rule of Thumb

Do not **pollute** your **Domain** layer with **presentation-specific concerns**

Introducing BackingBeans

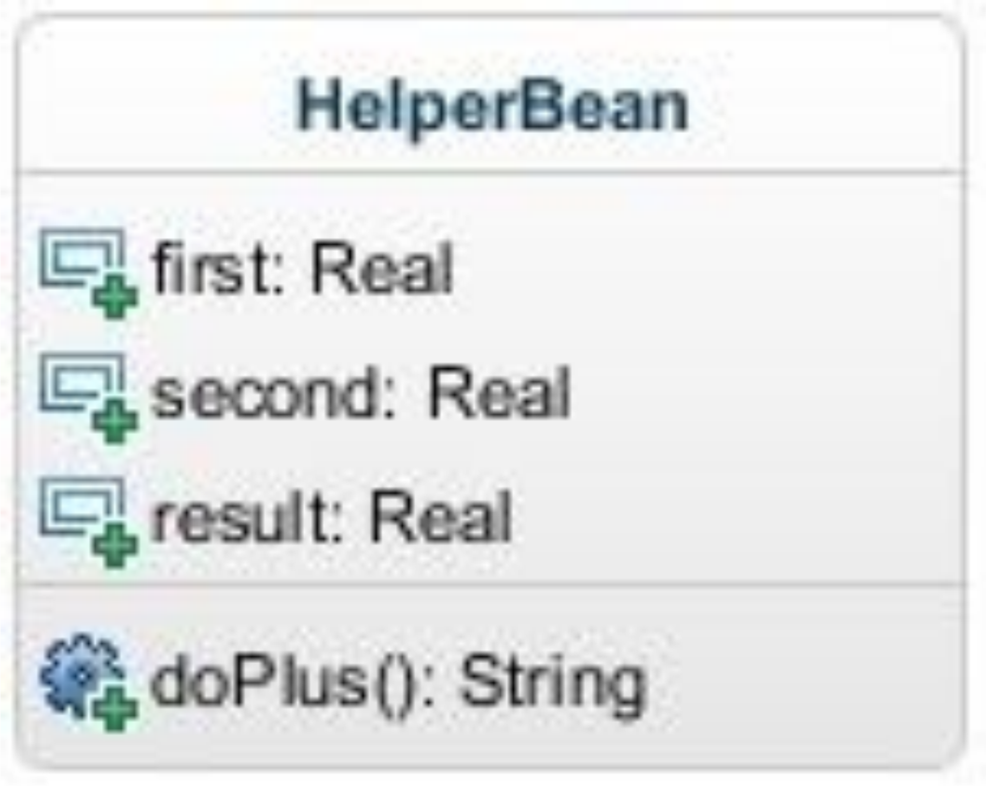




Enter first number 0.0

Enter second number 0.0

Add

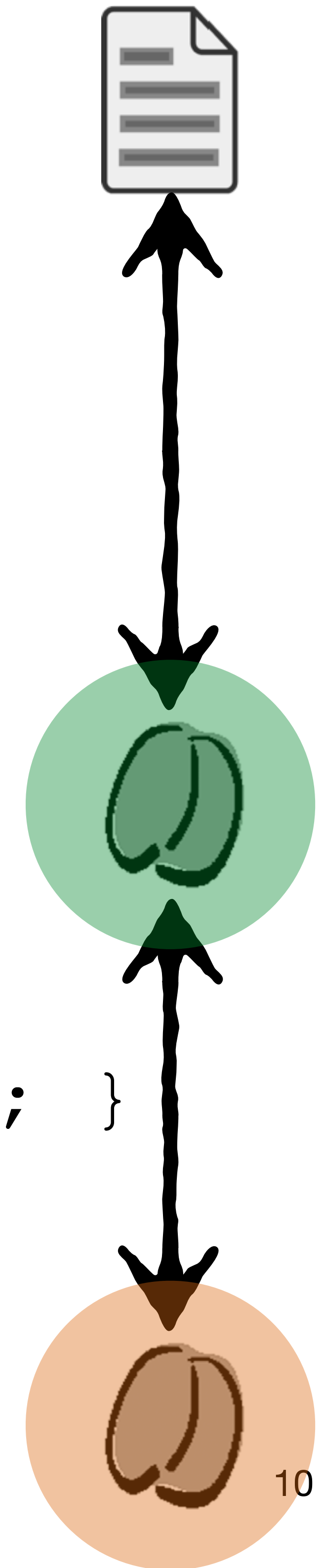


Data model for the web page

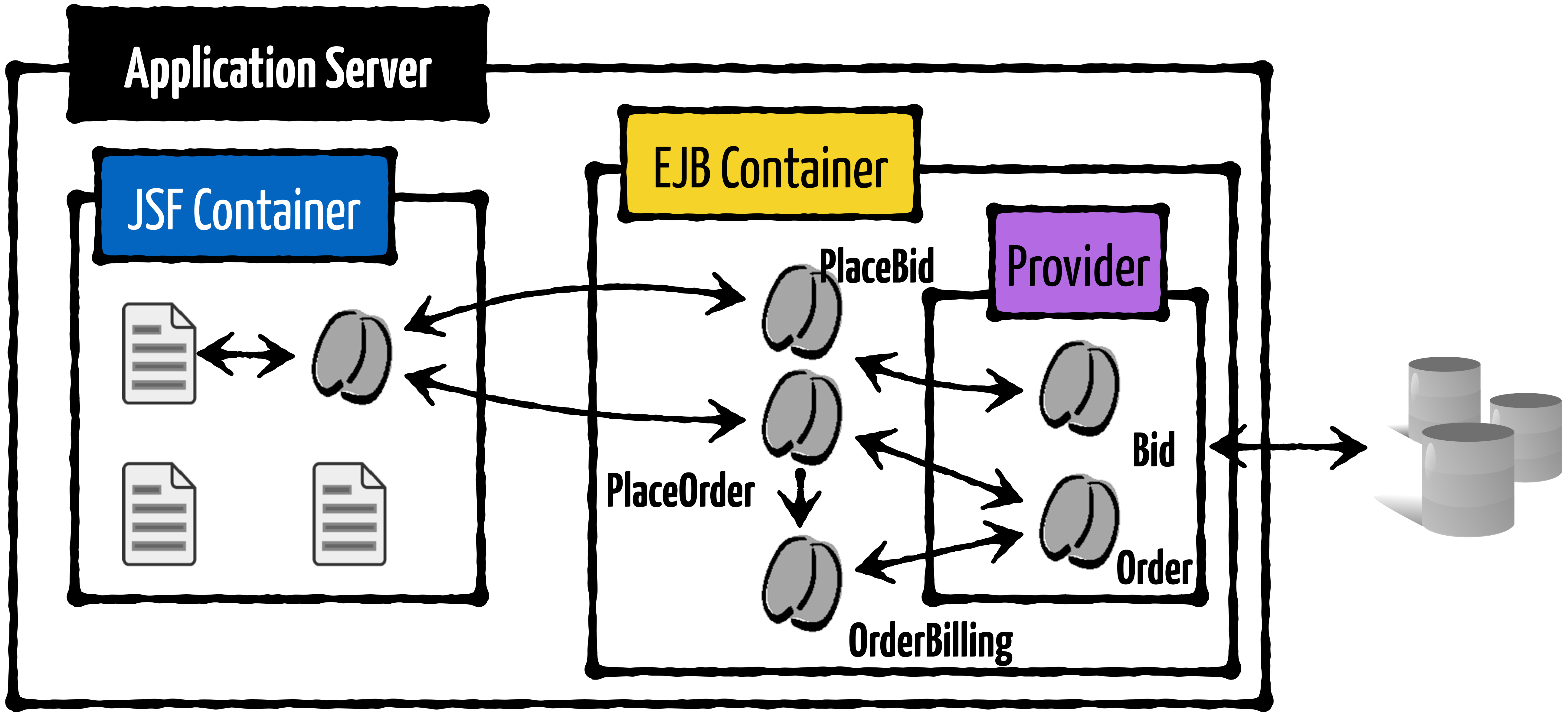
```
public double add(double x, double y);
```

Backing beans: Insulate processing!

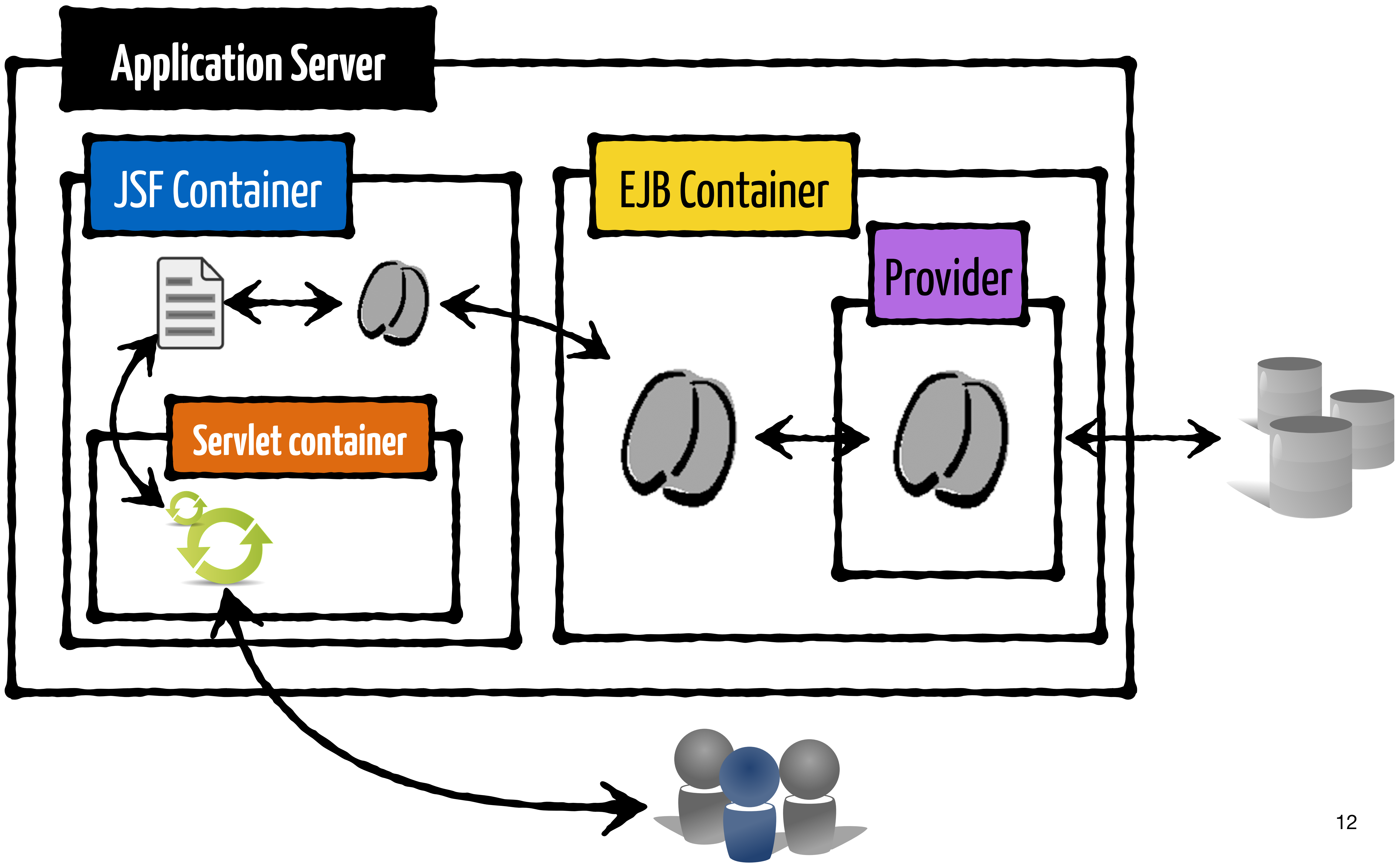
```
…
<h:inputText value='#{helper.first}'/>
…
<h:commandButton action="#{helper.plus}" …/>

@ManagedBean(name="helper")
public class CalculatorBean {

@EJB
Calculator calculator;

public void setFirst(double d) { this.first = d; }
public String doPlus() {
  result = calculator.add(first,second);
  return "success";
}
```



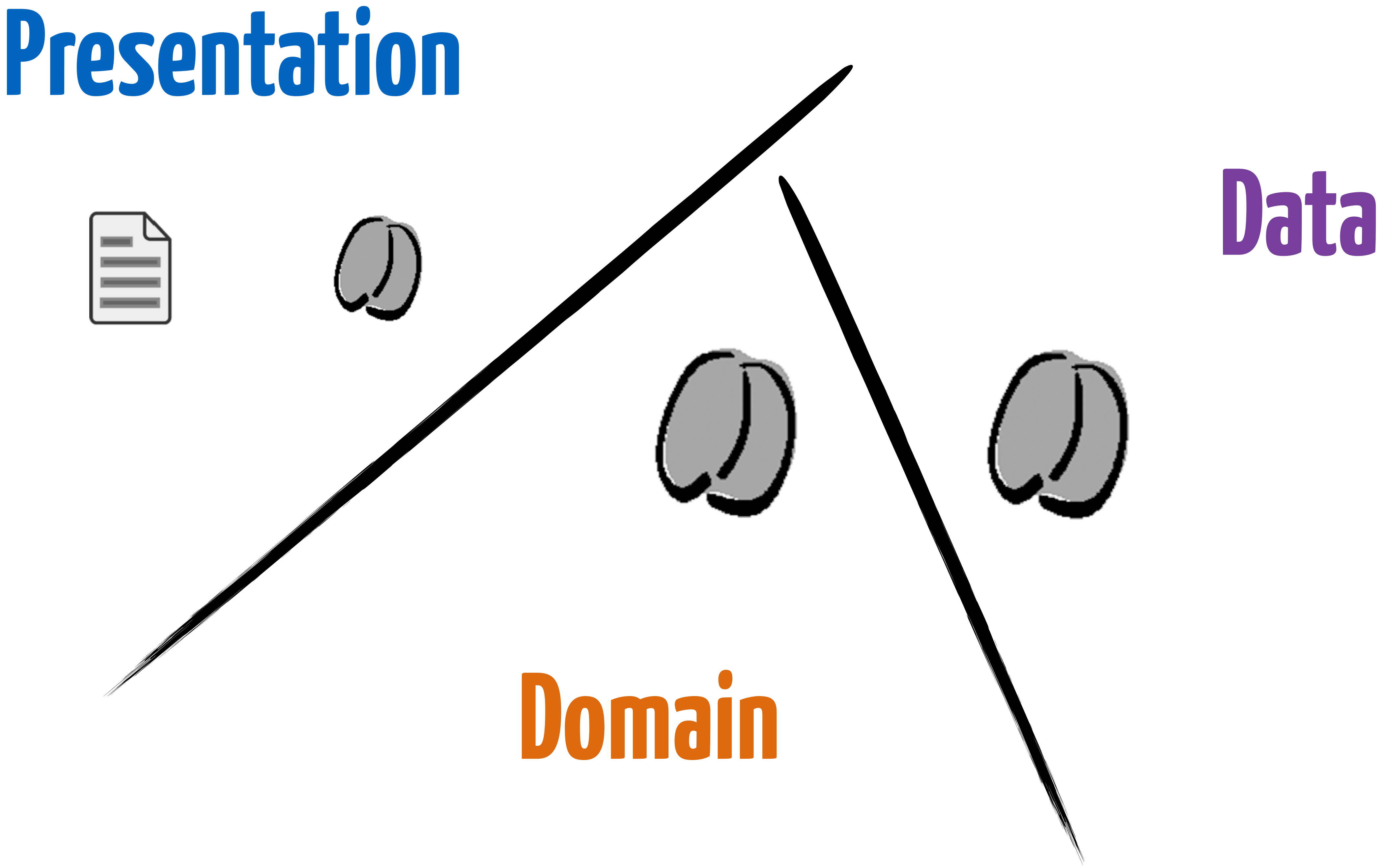
Relation with "Application Server" ?



It’s even more complex!



Main advantage: **We don't care!**



2 Artefacts

JSF Dependencies (e.g., Maven)

```
<dependency>
  <groupId>org.apache.myfaces.core</groupId>
  <artifactId>myfaces-api</artifactId>
  <version>2.1.8</version>
  <scope>provided</scope>
</dependency>

<dependency>
  <groupId>org.apache.myfaces.core</groupId>
  <artifactId>myfaces-impl</artifactId>
  <version>2.1.8</version>
  <scope>provided</scope>
</dependency>
```

```xml
<dependency>
  <groupId>org.apache.myfaces.core</groupId>
  <artifactId>myfaces-api</artifactId>
  <version>2.1.8</version>
  <scope>provided</scope>
</dependency>
```

Needed to **compile**

Will be **provided** at runtime
by the container

View

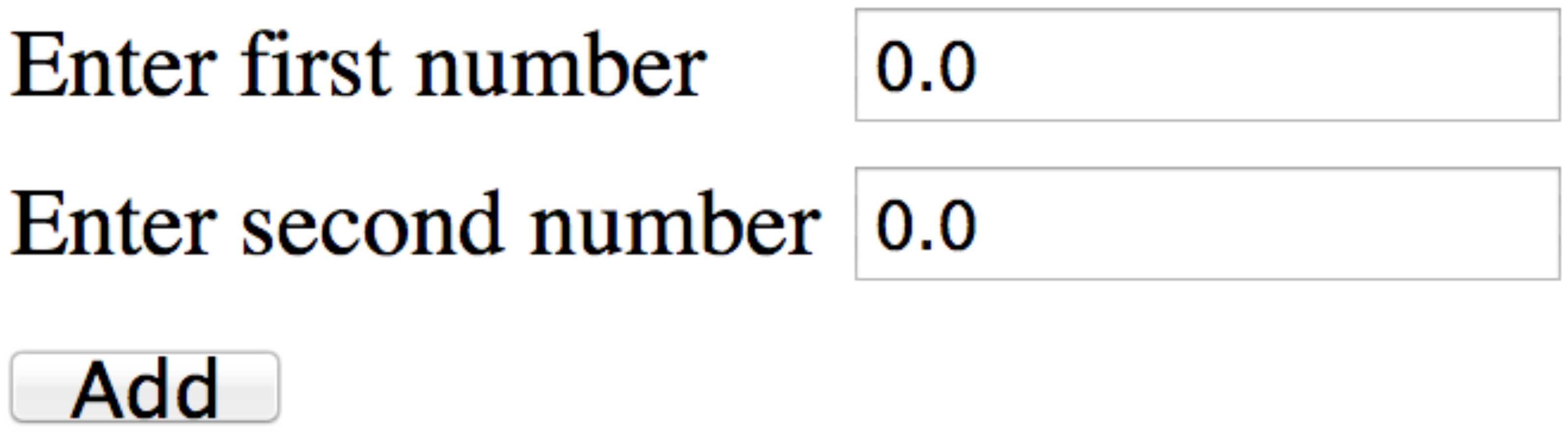
```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:h="http://java.sun.com/jsf/html">

<h:body bgcolor="white">
  <f:view>
  <h:form>
      <h:panelGrid columns="2">
        <h:outputText value='Enter first number'/>
        <h:inputText value='#{back.first}'/>
        <h:outputText value='Enter second number'/>
        <h:inputText value='#{back.second}'/>
        <h:commandButton action="#{back.doPlus}"
                         value="Add"/>
      </h:panelGrid>
    </h:form>
  </f:view>
</h:body>
</html>
```

off-the-shelf components

Backing bean

calculator.xhtml

BackingBean



Enter first number 0.0

Enter second number 0.0

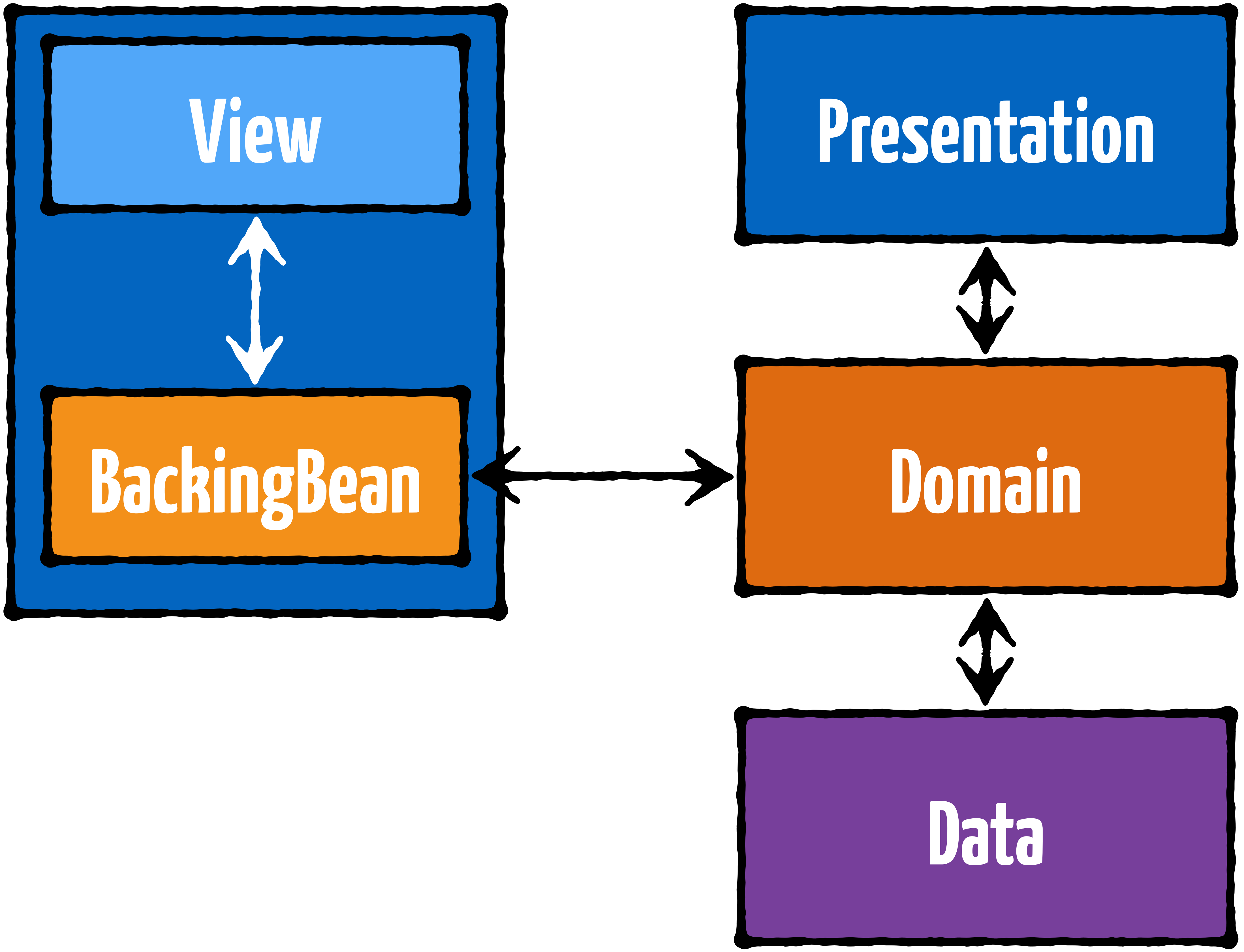
Add

```
@ManagedBean(name = "back")
public class CalculatorBackingBean {

   @EJB
   Calculator calculator;

   private double first;
   private double second;
   private double result;

   public void doPlus() {
      result = calculator.add(first, second);
   }
}
```

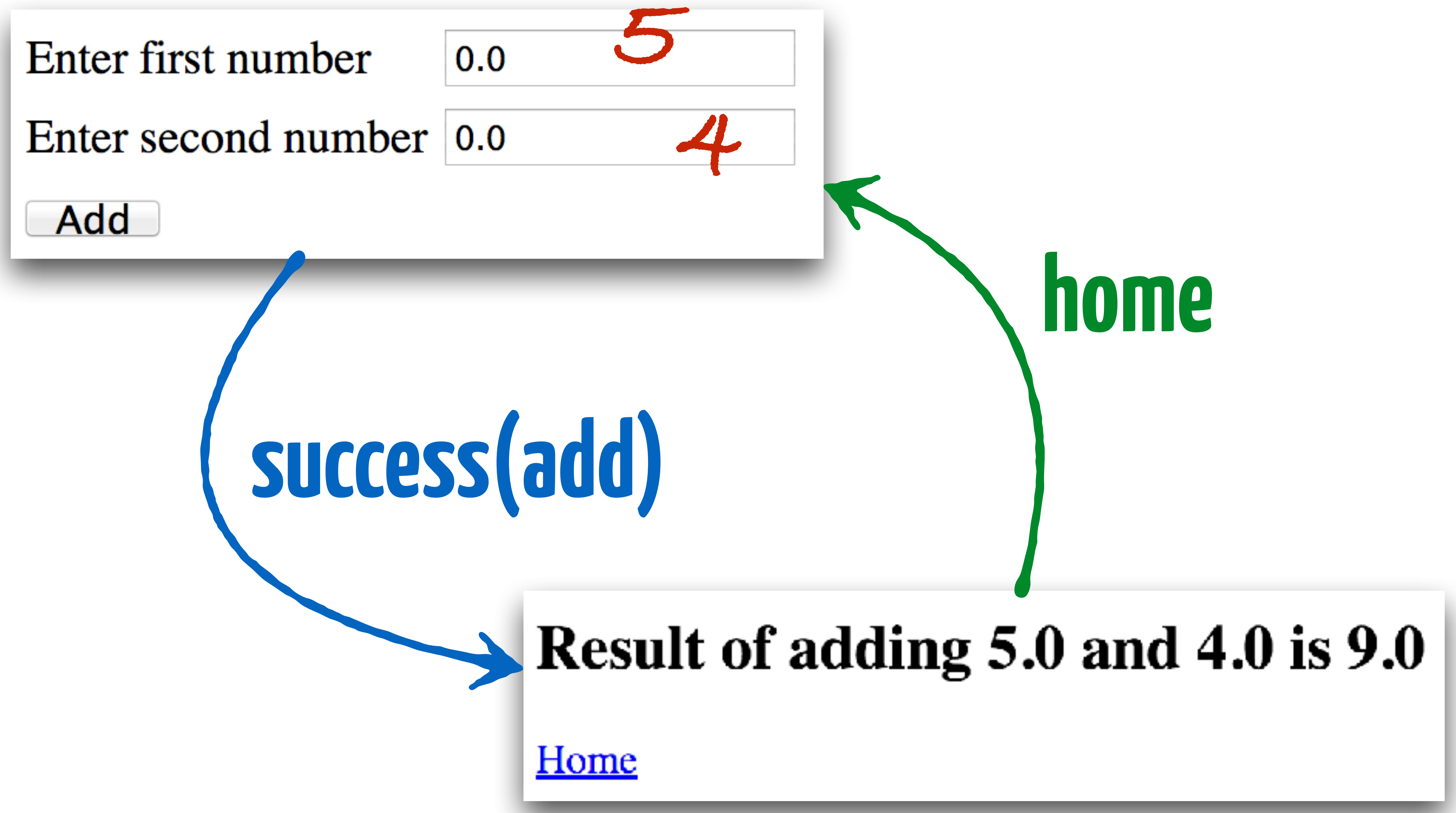
binds to domain layer (injected)

Presentation "data model"

Presentation "domain model"


View
BackingBean
Presentation
Domain
Data

Navigation flow as an automaton



success(add) ?

```
@ManagedBean(name = "back")
public class CalculatorBackingBean {

  public void doPlus() {
    result = calculator.add(first, second);
  }
}
```

```
@ManagedBean(name = "back")
public class CalculatorBackingBean {

  public String doPlus() {
    result = calculator.add(first, second);
    return "success";
  }
}
```

Implementing the automaton

```xml
<navigation-rule>
  <from-view-id>/calculator.xhtml</from-view-id>
  <navigation-case>
    <from-outcome>success</from-outcome>
    <to-view-id>/result.xhtml</to-view-id>
  </navigation-case>
</navigation-rule>
```

```xml
<navigation-rule>
  <from-view-id>/result.xhtml</from-view-id>
  <navigation-case>
    <from-outcome>back</from-outcome>
    <to-view-id>/calculator.xhtml</to-view-id>
  </navigation-case>
</navigation-rule>
```

webapp/WEB-INF/faces-config.xml

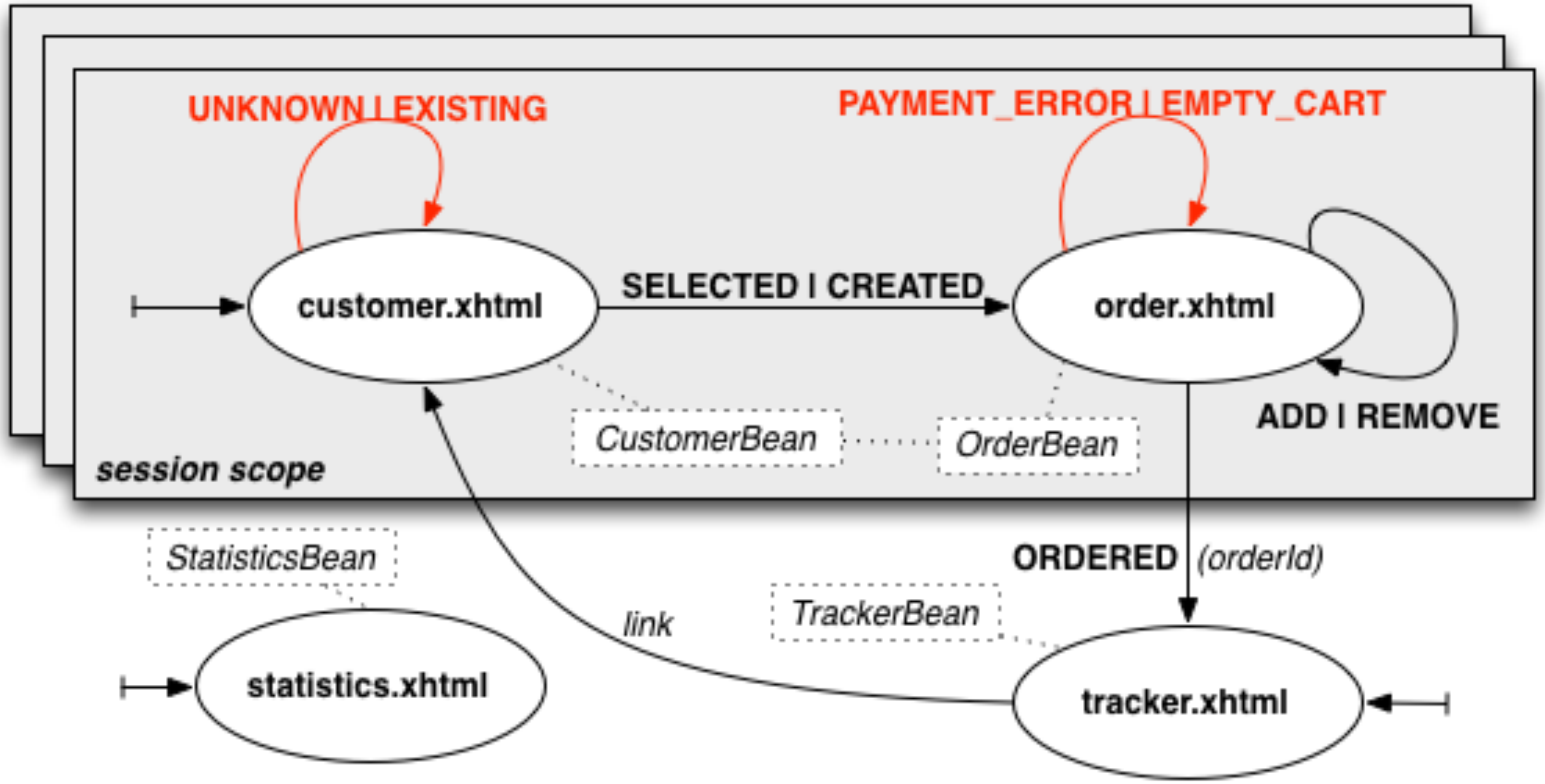
Processing the Views

JSF servlet to handle incoming requests

```
<servlet>
  <servlet-name>Faces Servlet</servlet-name>
  <servlet-class>
    javax.faces.webapp.FacesServlet
  </servlet-class>
  <load-on-startup>1</load-on-startup>
</servlet>
```

catch requests to *.jsf

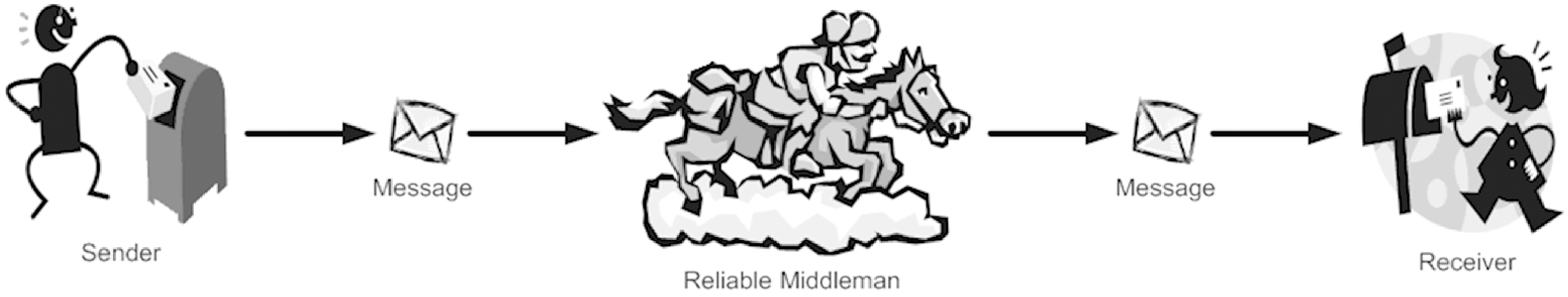
```
<servlet-mapping>
  <servlet-name>Faces Servlet</servlet-name>
  <url-pattern>*.jsf</url-pattern>
</servlet-mapping>
```


UNKNOWN | EXISTING
PAYMENT_ERROR | EMPTY_CART
customer.xhtml
SELECTED | CREATED
order.xhtml
ADD | REMOVE
CustomerBean
OrderBean
session scope
StatisticsBean
ORDERED (orderId)
TrackerBean
link
statistics.xhtml
tracker.xhtml

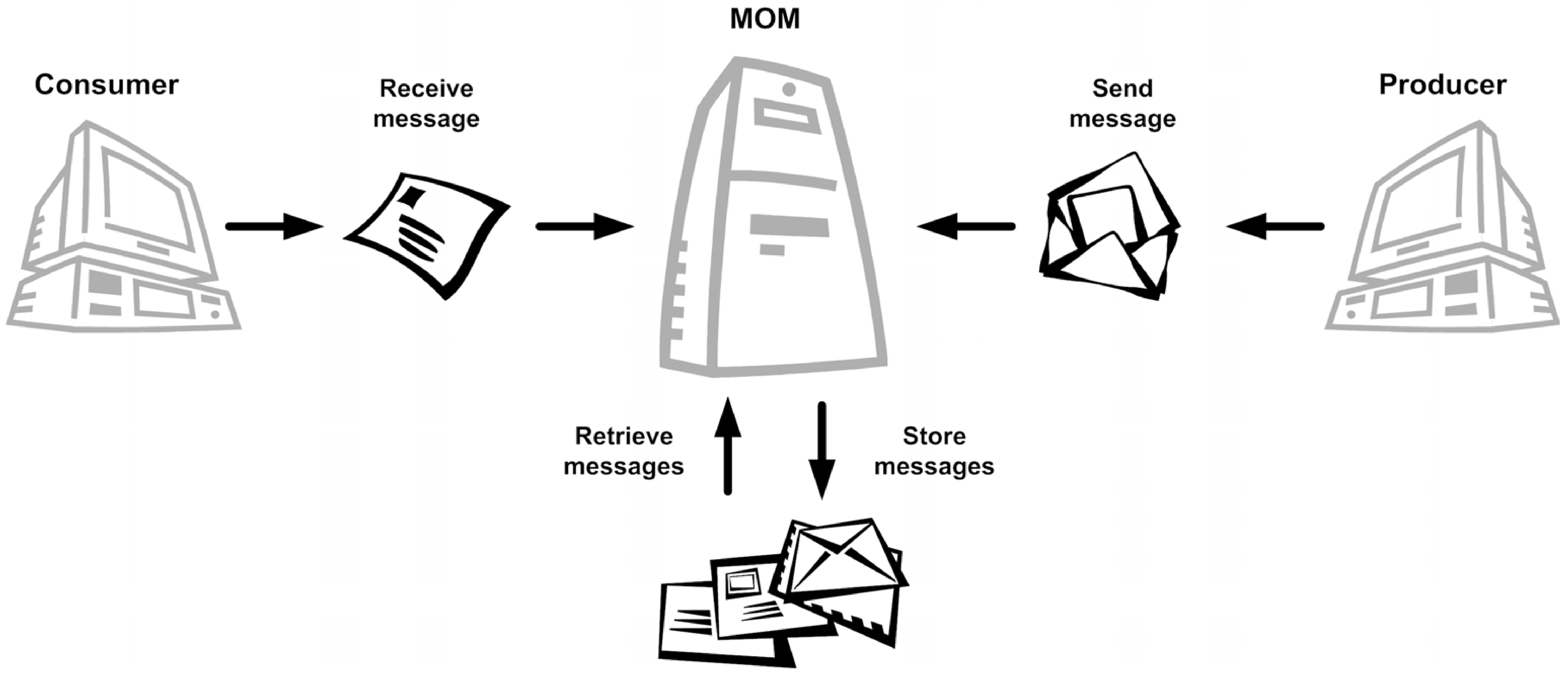
Message-oriented Middleware

EJB Messages

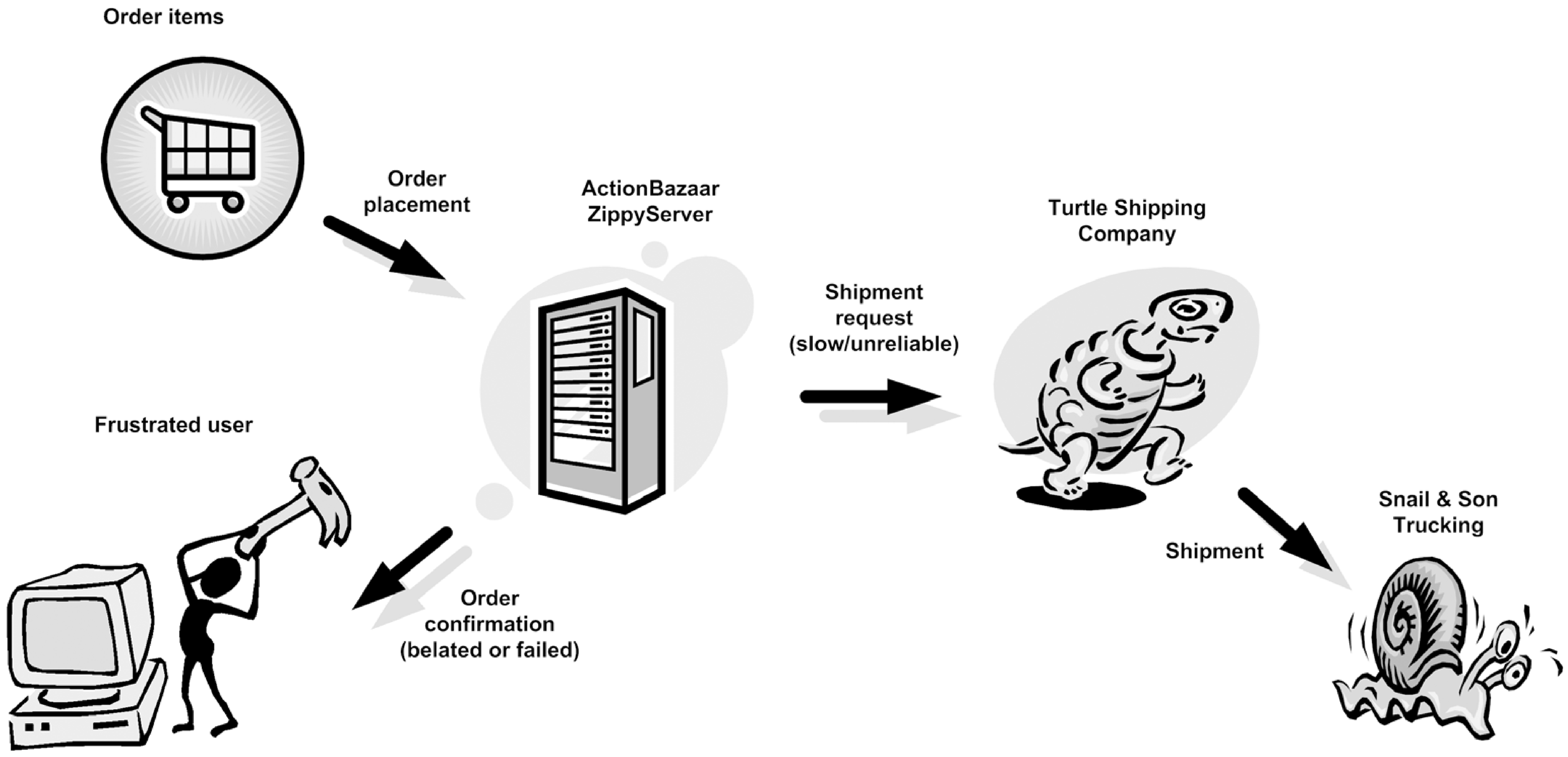
The **Messaging** paradigm



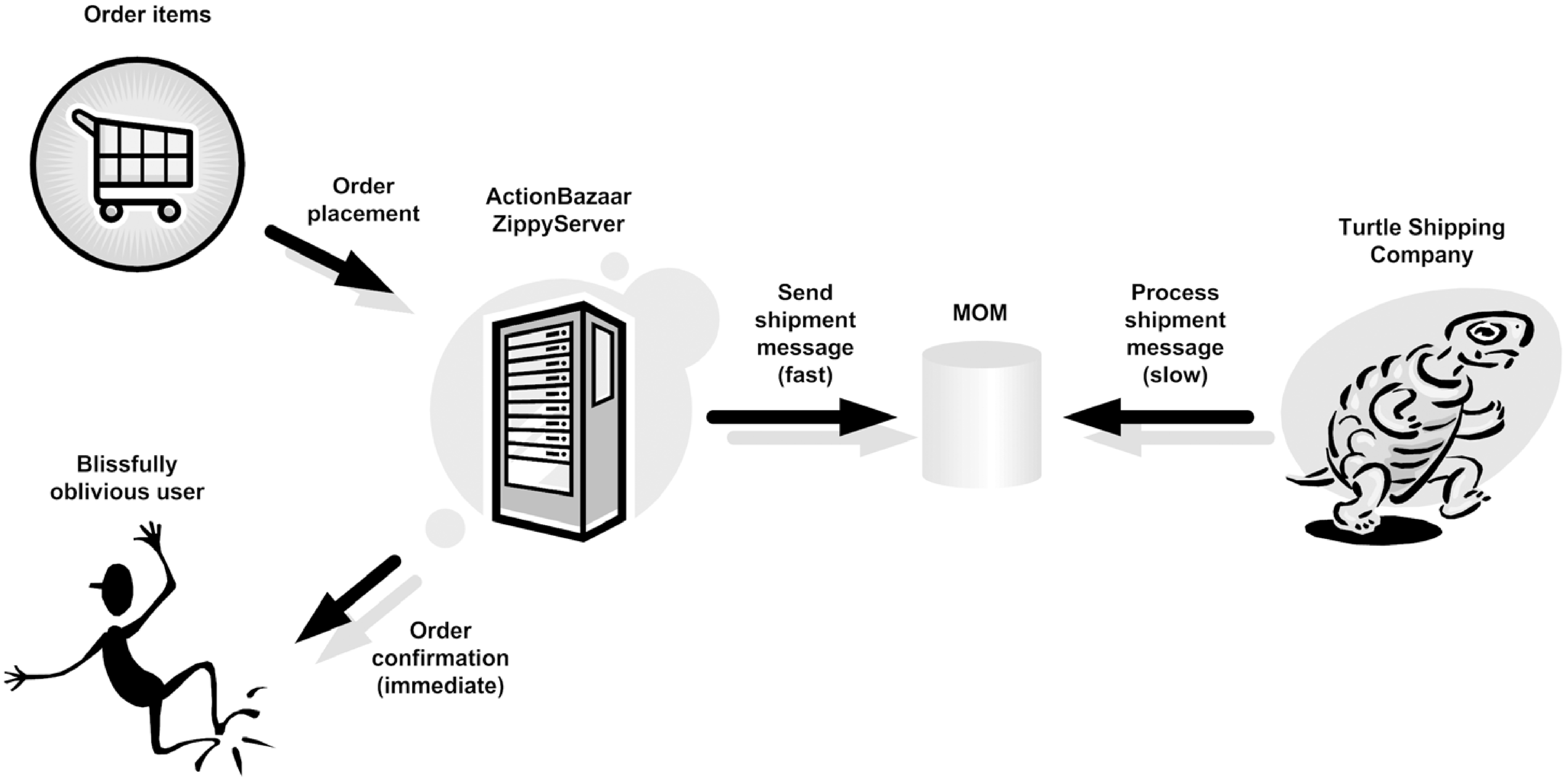
MOM: Message-oriented Middleware



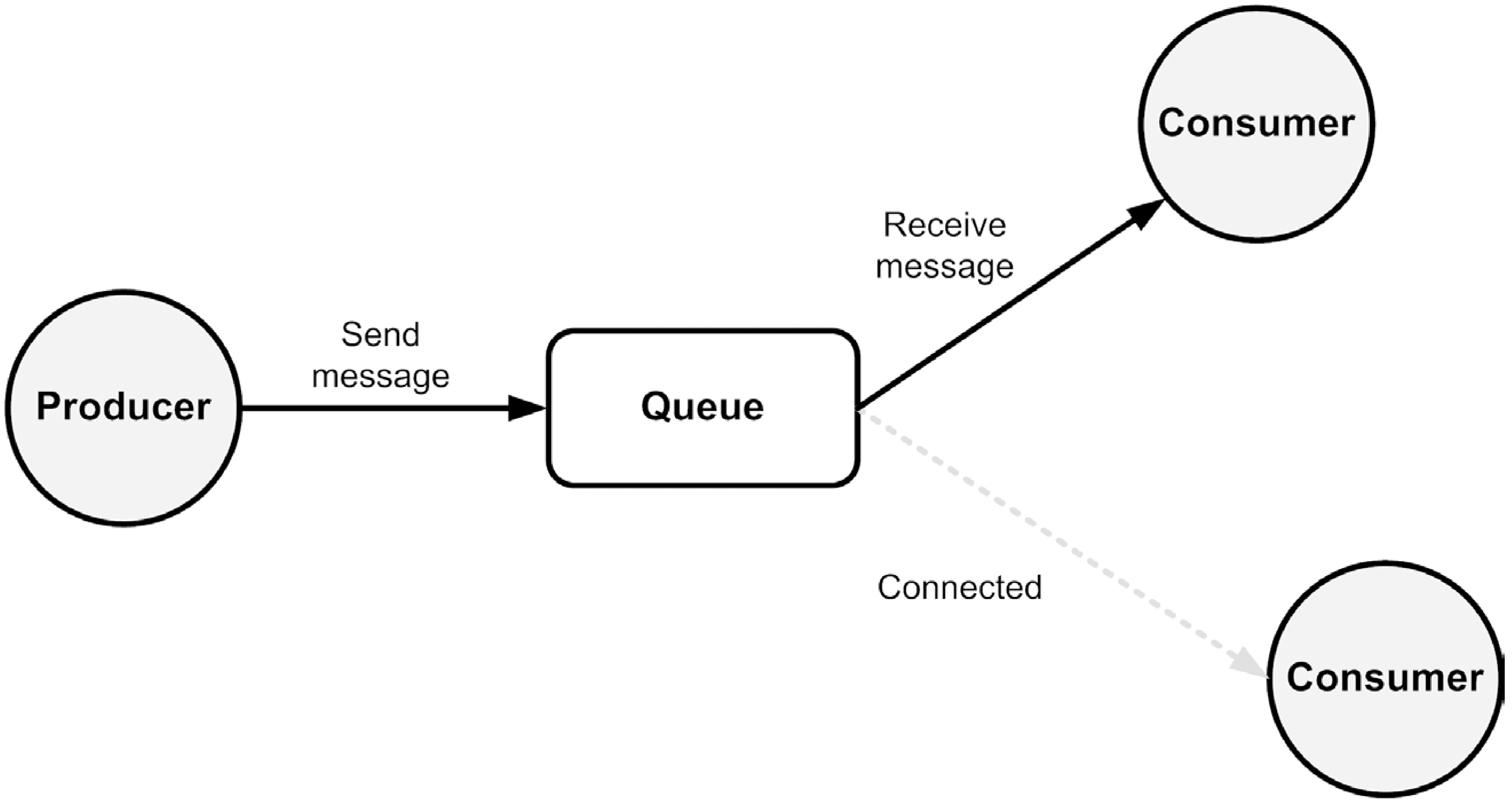
The ActionBazaar example



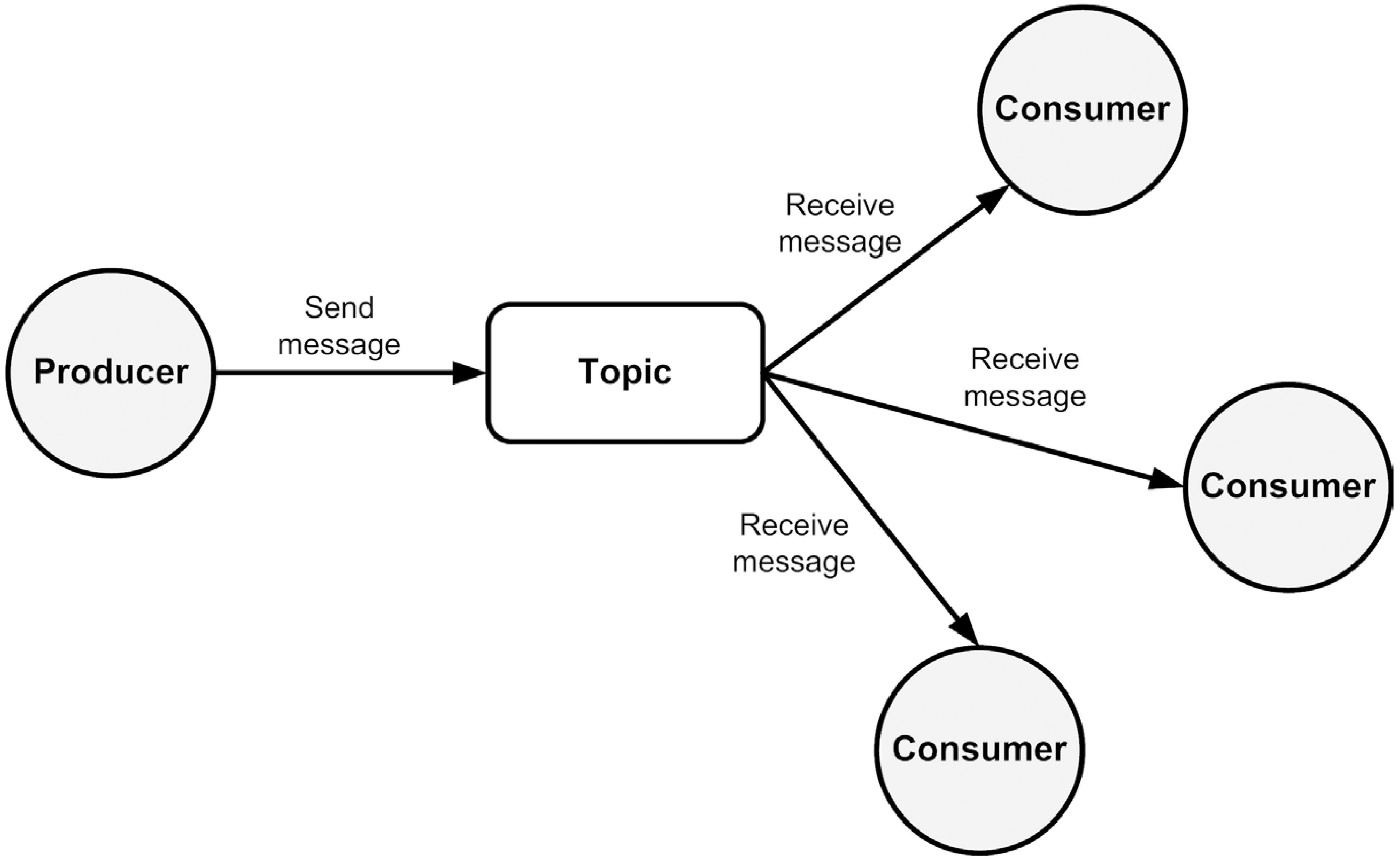
Introducing messaging


Order items
Order placement
ActionBazaar ZippyServer
Send shipment message (fast)
MOM
Process shipment message (slow)
Turtle Shipping Company
Blissfully oblivious user
Order confirmation (immediate)

Model: Point-to-Point



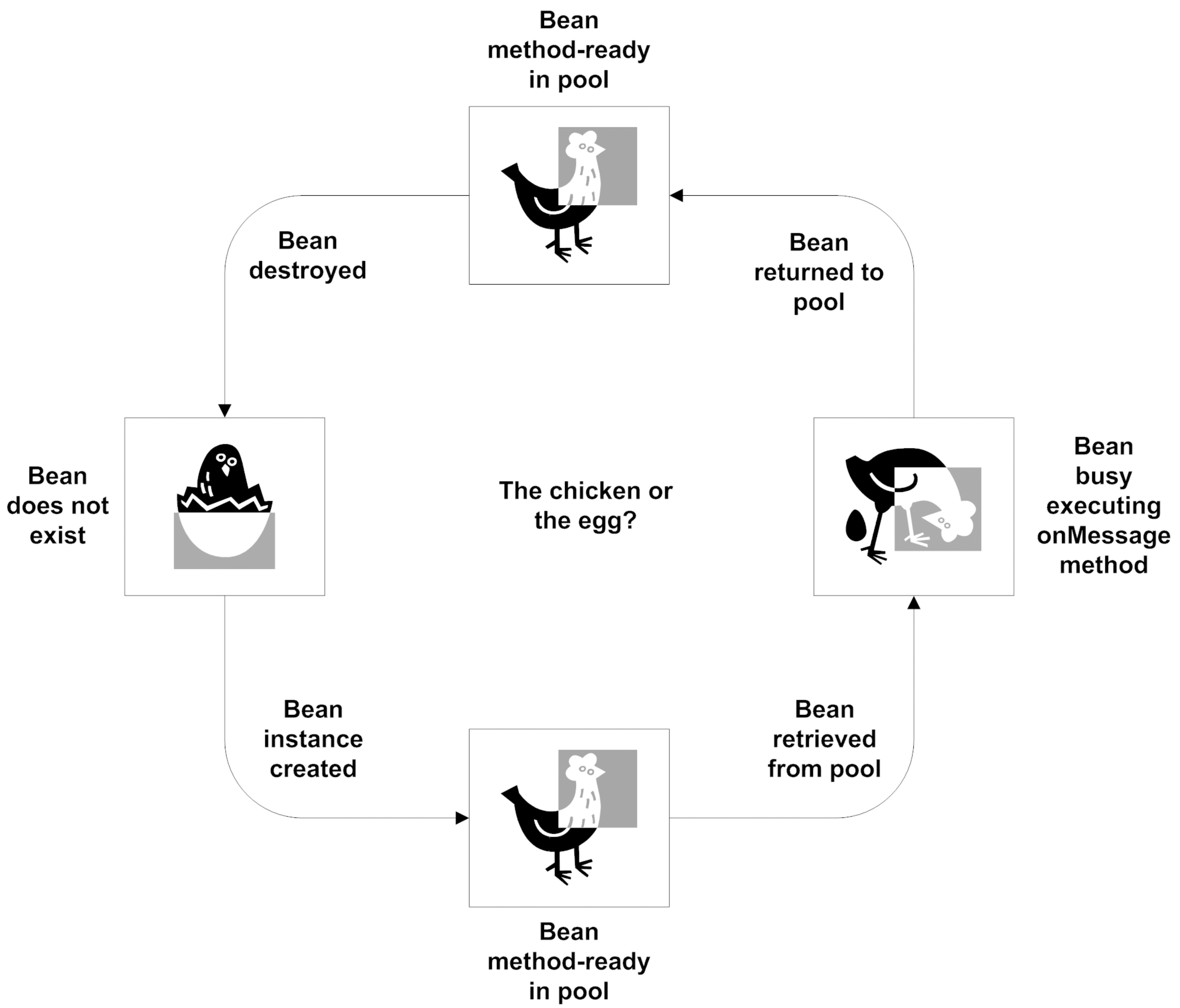
Model: Publish-Subscribe



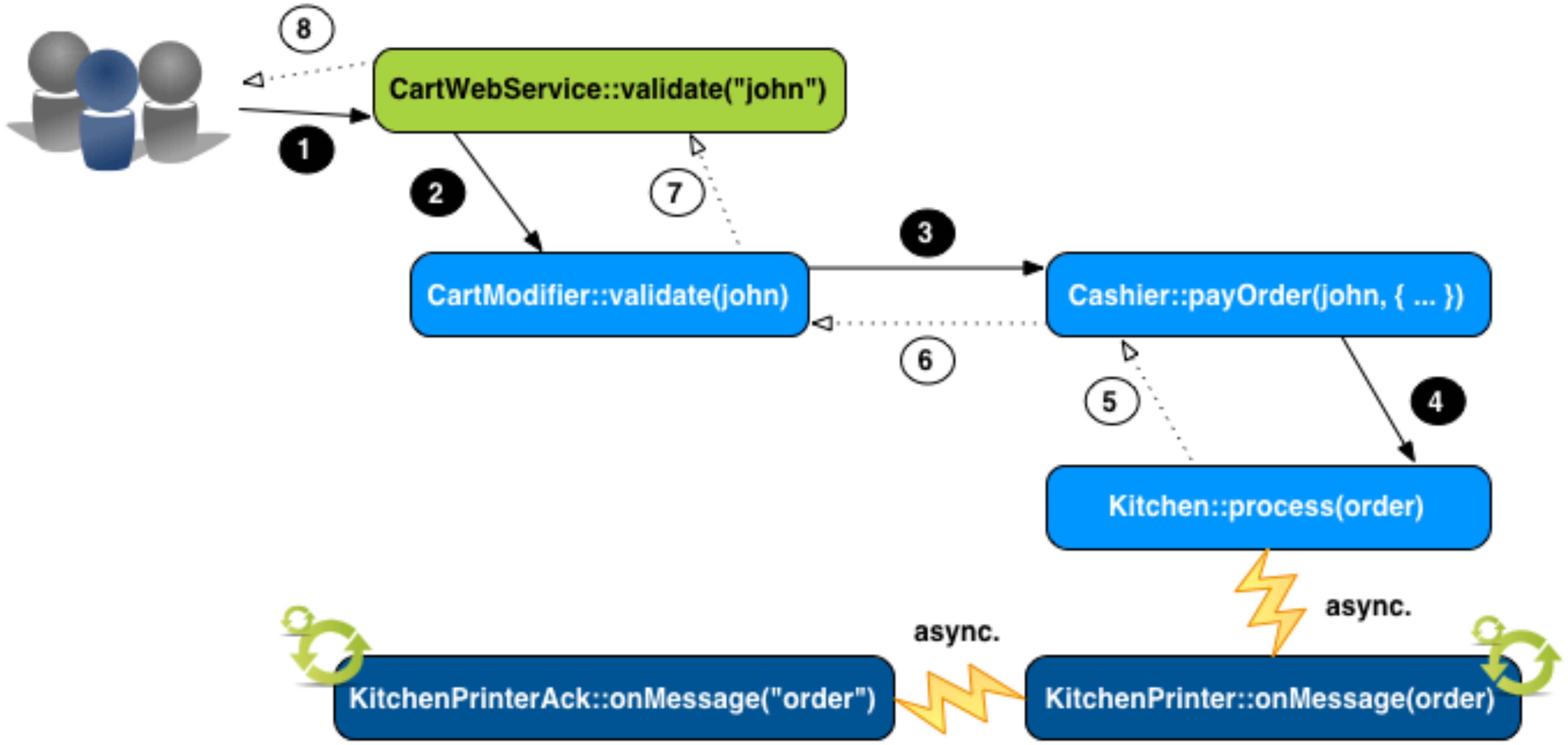
Implementation: onMessage(Message m)

```
public void onMessage(Message message) {
    try {
        ObjectMessage objectMessage = (ObjectMessage)message;
        ShippingRequest shippingRequest =
            (ShippingRequest)objectMessage.getObject();
        processShippingRequest(shippingRequest);
    } catch (JMSException jmse) {
        jmse.printStackTrace();
        context.setRollBackOnly();();
    } catch (SQLException sqle) {
        sqle.printStackTrace();
        context.setRollBackOnly();
    }
}
```

Message-driven bean lifecycle



Example: The Cookie Factory


8
CartWebService::validate("john")
1
2
7
3
CartModifier::validate(john)
Cashier::payOrder(john, { ... })
6
5
4
Kitchen::process(order)
async.
async.
KitchenPrinterAck::onMessage("order")
KitchenPrinter::onMessage(order)

Example: Text-based receiver

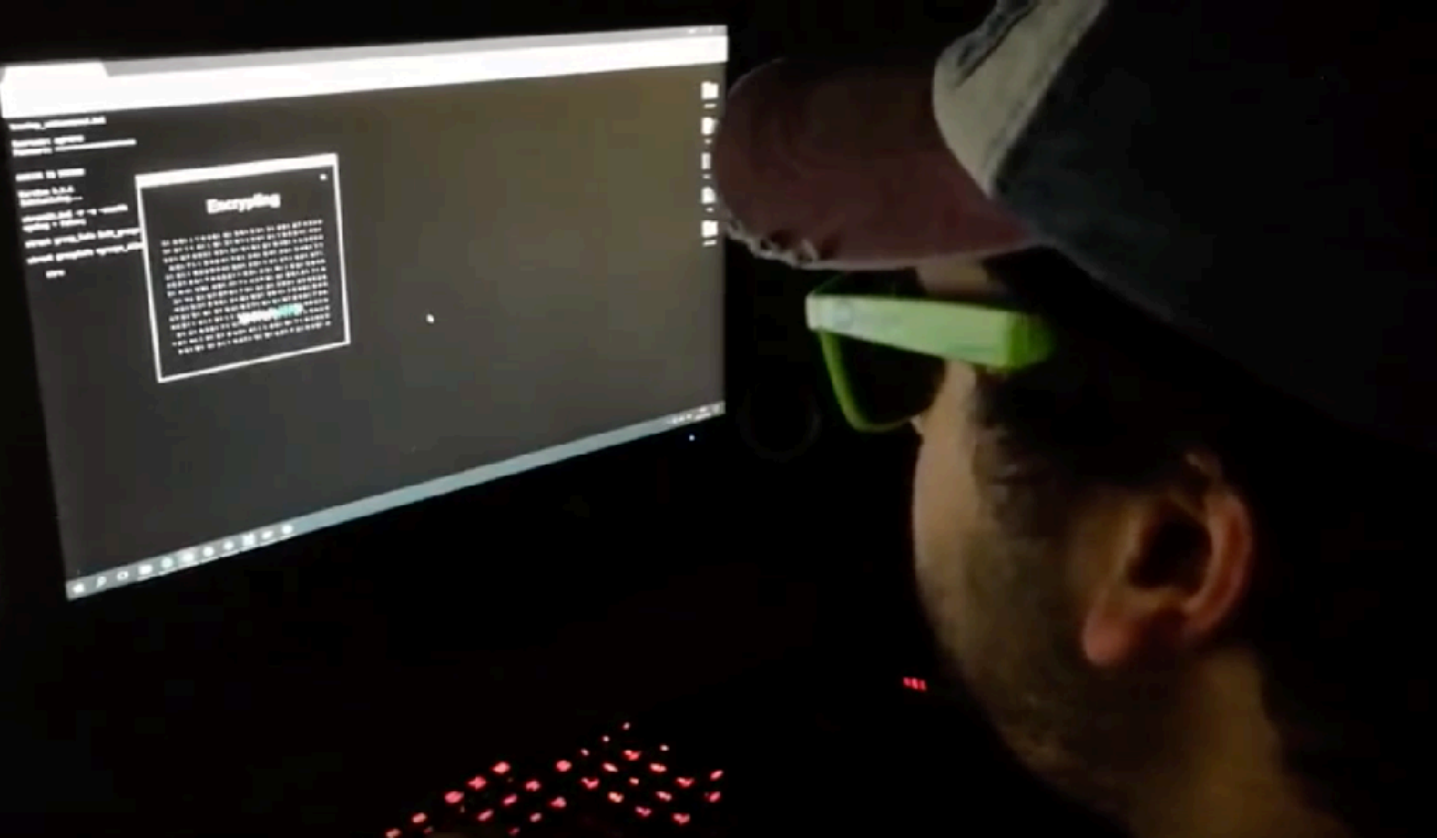
```
@MessageDriven
public class KitchenPrinterAck implements MessageListener {

    // ...

   public void onMessage(Message message) {
     try {
       String data = ((TextMessage) message).getText();
       System.out.println("\n\n****\n** ACK: " + data + "\n****\n");
    } catch (JMSException e) {
       throw new RuntimeException("Cannot read the received message!");
    }
  }
}
```

Sending a message to a MDB

```java
@Resource private ConnectionFactory connectionFactory;
@Resource(name = "KitchenPrinterAck") private Queue q;

private void acknowledge(int orderId) throws JMSException {
  Connection connection = null; Session session = null;
  try {
    connection = connectionFactory.createConnection();
    connection.start();
    session =
      connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
    MessageProducer producer = session.createProducer(q);
    producer.setDeliveryMode(DeliveryMode.NON_PERSISTENT);
    producer.send(session.createTextMessage(orderId + ";PRINTED"));
  } finally {
    if (session != null) session.close();
    if (connection != null) connection.close();
  }
}
```

Handling objects

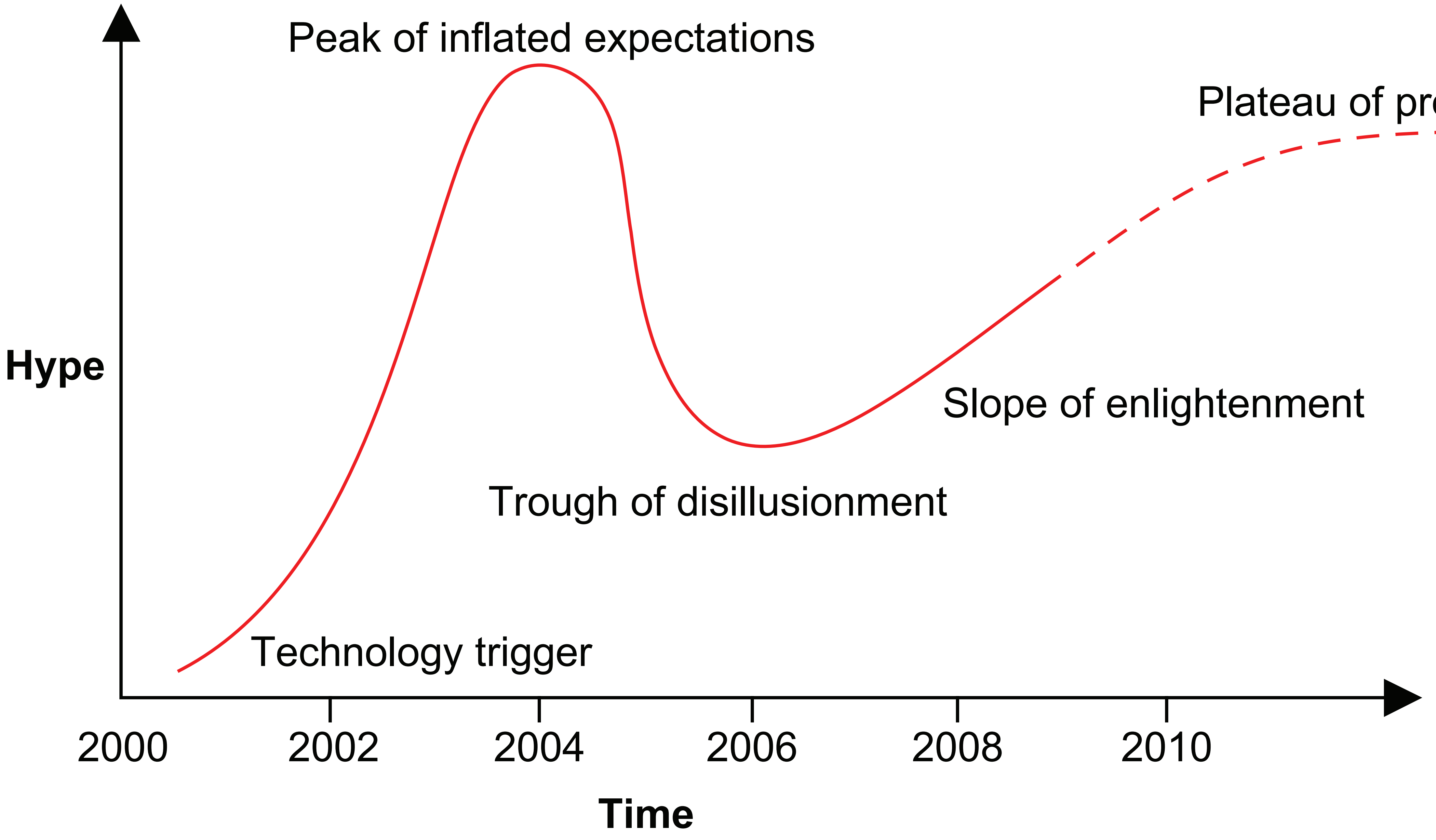
```
public void onMessage(Message message) {
  try {
    Order data = (Order) ((ObjectMessage) message).getObject();
    handle(data);
  } catch (JMSException e) {
    throw new RuntimeException("Cannot print …”);
  }
}

private void handle(Order o) throws IllegalStateException {
  Order data = entityManager.merge(o);
  // ...
}
```



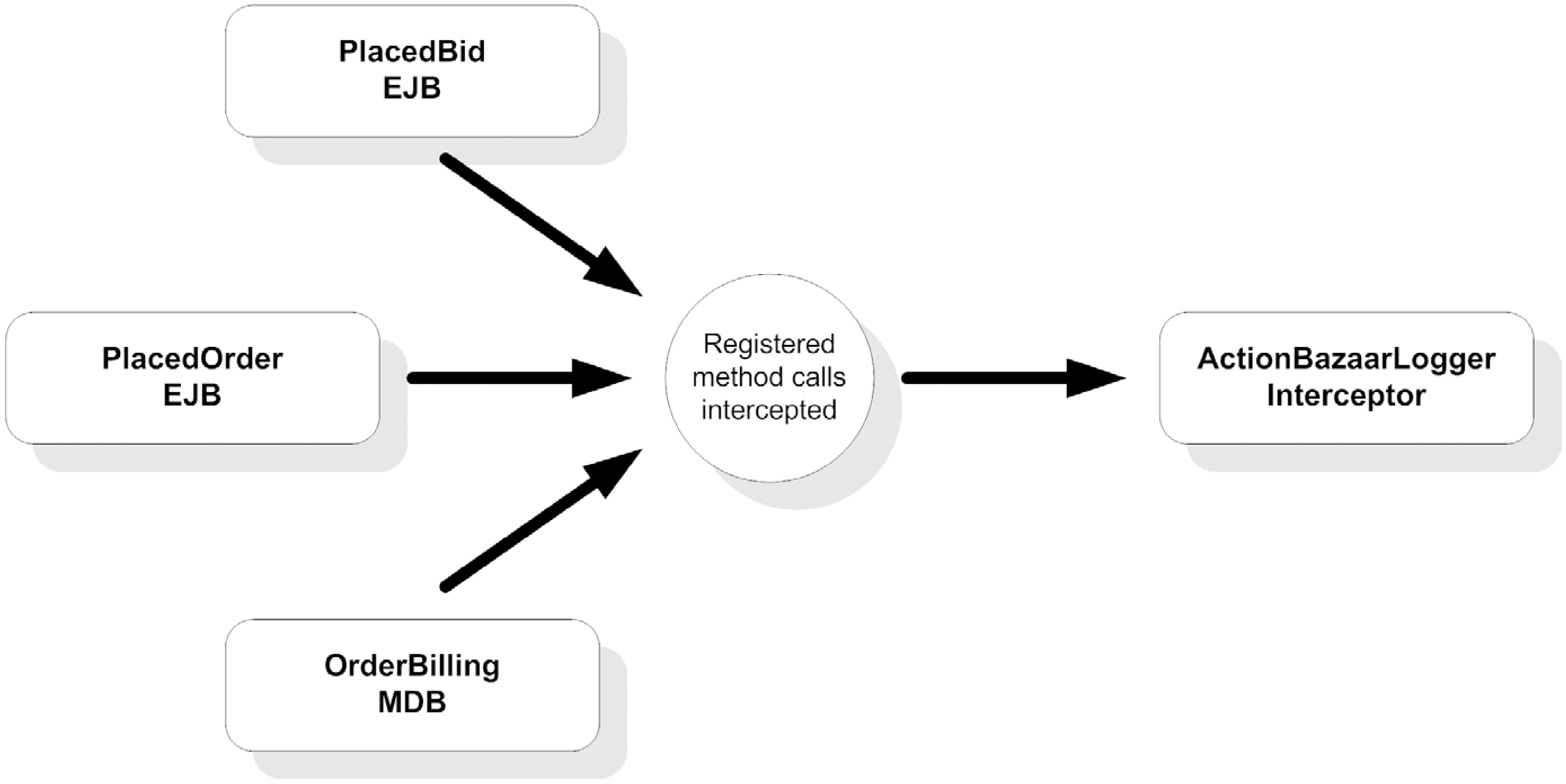
Intercepting Messages

Man in the middle
(without the hoody)

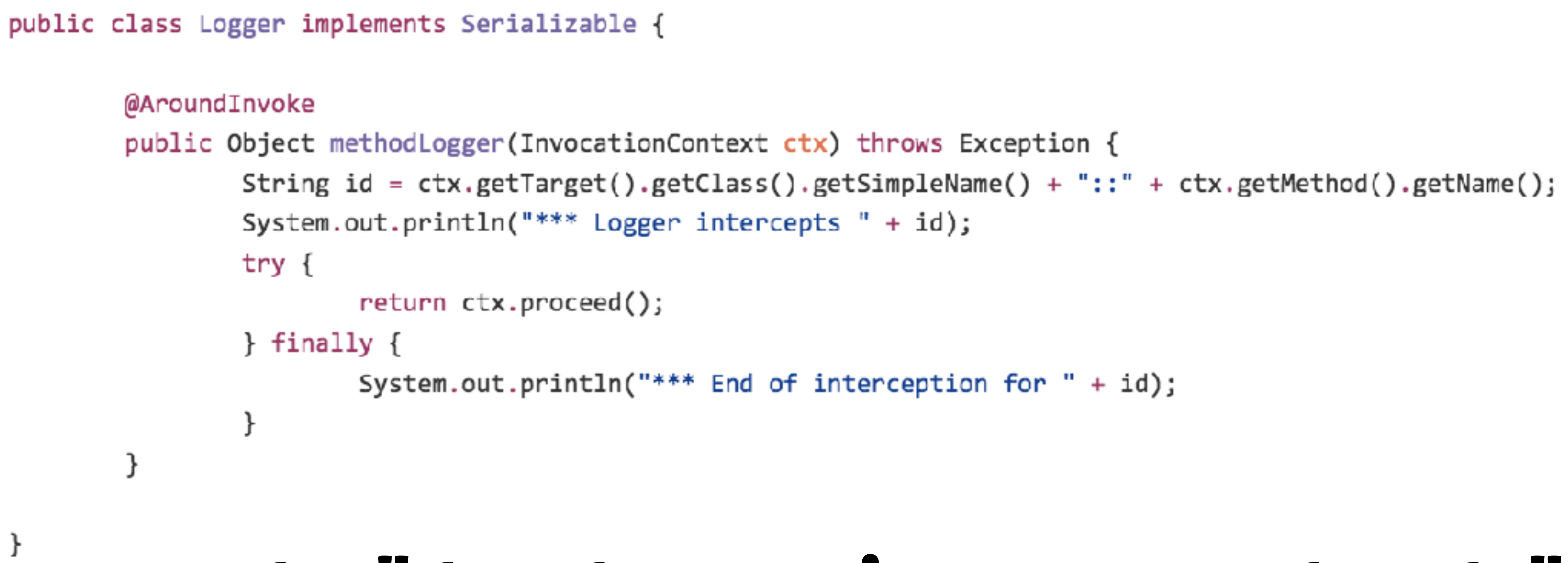
Roots: Aspect-oriented Programming



[Gartner Hype Cycle]

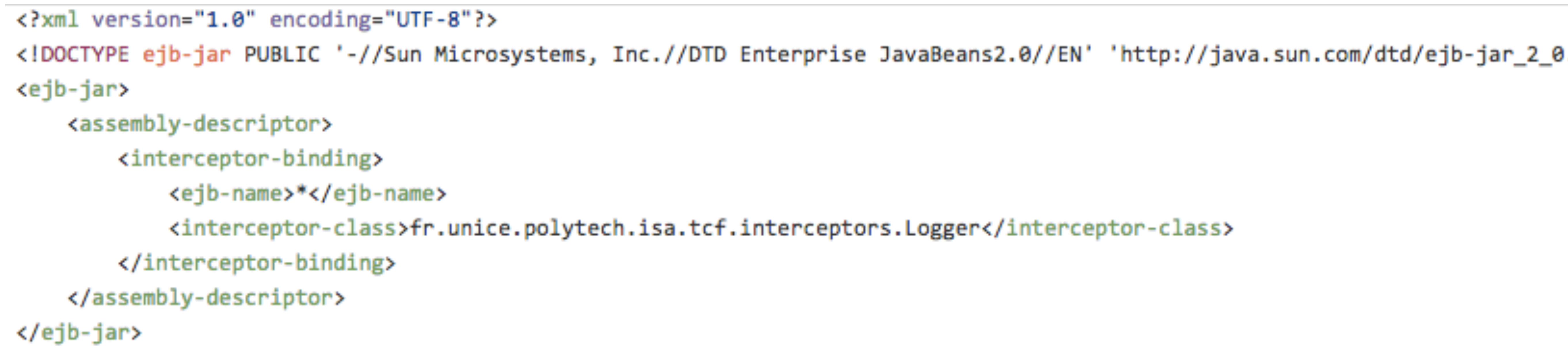

PlacedBid
EJB
PlacedOrder
EJB
OrderBilling
MDB
Registered method calls intercepted
ActionBazaarLogger
Interceptor

```
@Stateless
public class PlaceBidBean implements PlaceBid {
    ...
    @Interceptors(ActionBazaarLogger.class)
    public void addBid(Bid bid) {
        ...
    }
}

public class ActionBazaarLogger {
    @AroundInvoke
    public Object logMethodEntry(
        InvocationContext invocationContext)
            throws Exception {
        System.out.println("Entering method: ”
            + invocationContext.getMethod().getName());
        return invocationContext.proceed();
    }
}
```

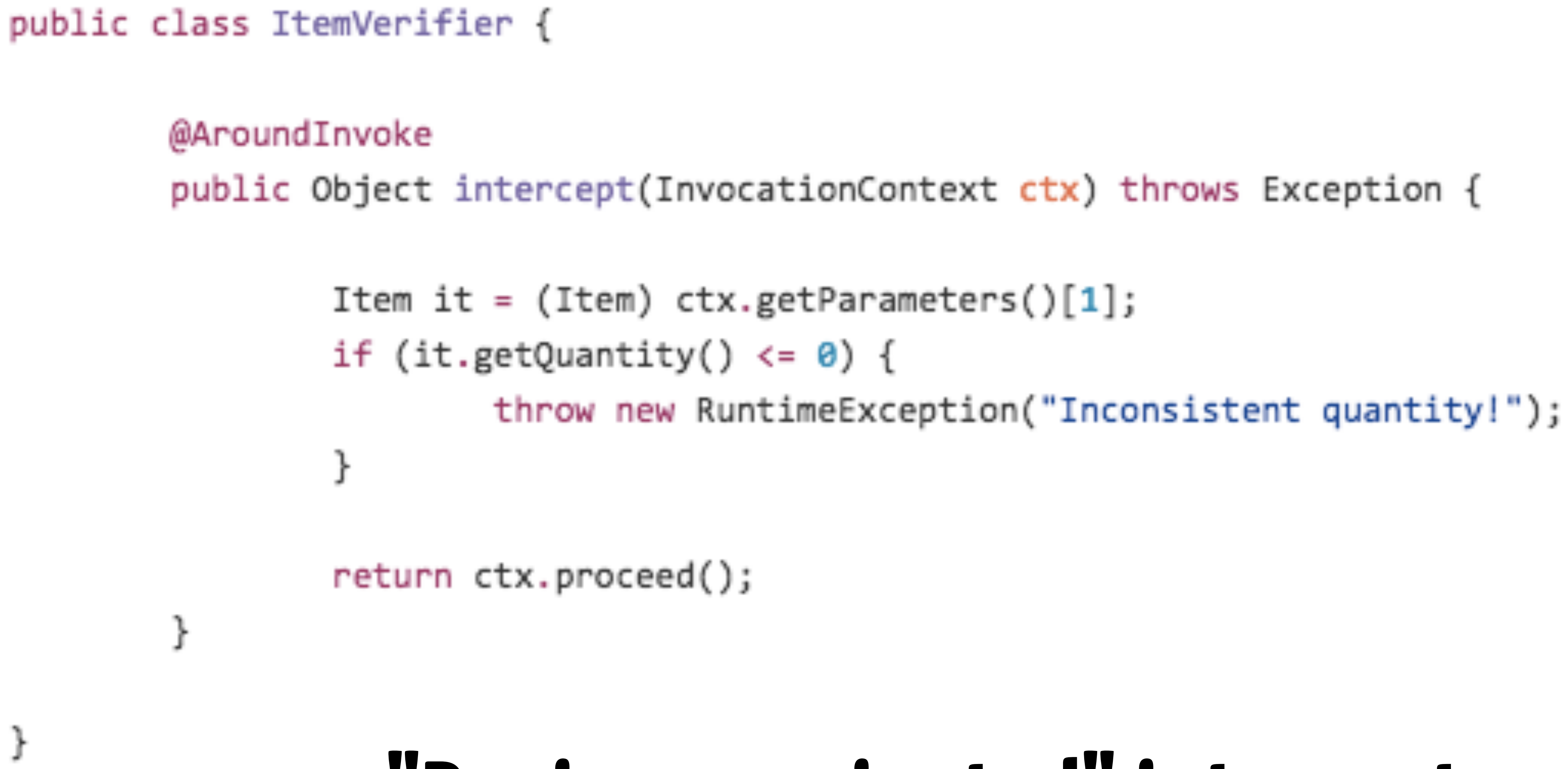


```java
public class Logger implements Serializable {

	@AroundInvoke
	public Object methodLogger(InvocationContext ctx) throws Exception {
		String id = ctx.getTarget().getClass().getSimpleName() + "::" + ctx.getMethod().getName();
		System.out.println("*** Logger intercepts " + id);
		try {
			return ctx.proceed();
		} finally {
			System.out.println("*** End of interception for " + id);
		}
	}

}
```

proceed = "do what you're supposed to do"

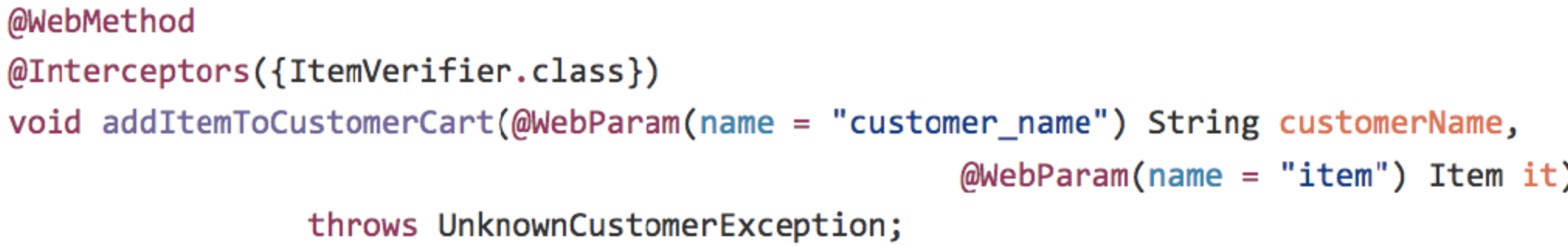


```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ejb-jar PUBLIC '-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans2.0//EN' 'http://java.sun.com/dtd/ejb-jar_2_0
<ejb-jar>
    <assembly-descriptor>
        <interceptor-binding>
            <ejb-name>*</ejb-name>
            <interceptor-class>fr.unice.polytech.isa.tcf.interceptors.Logger</interceptor-class>
        </interceptor-binding>
    </assembly-descriptor>
</ejb-jar>
```



```
public class ItemVerifier {

        @AroundInvoke
        public Object intercept(InvocationContext ctx) throws Exception {

                Item it = (Item) ctx.getParameters()[1];
                if (it.getQuantity() <= 0) {
                        throw new RuntimeException("Inconsistent quantity!");
                }

                return ctx.proceed();
        }

}
```

"Business-oriented" interceptors



```
@WebMethod
@Interceptors({ItemVerifier.class})
void addItemToCustomerCart(@WebParam(name = "customer_name") String customerName,
                                                  @WebParam(name = "item") Item it
            throws UnknownCustomerException;
```

```java
public class CartCounter implements Serializable {

        @EJB private Database memory;

        @AroundInvoke
        public Object intercept(InvocationContext ctx) throws Exception {
                Object result = ctx.proceed();  // do what you're supposed to do
                memory.incrementCarts();
                System.out.println("  #Cart processed: " + memory.howManyCarts());
                return result;
        }

}
```