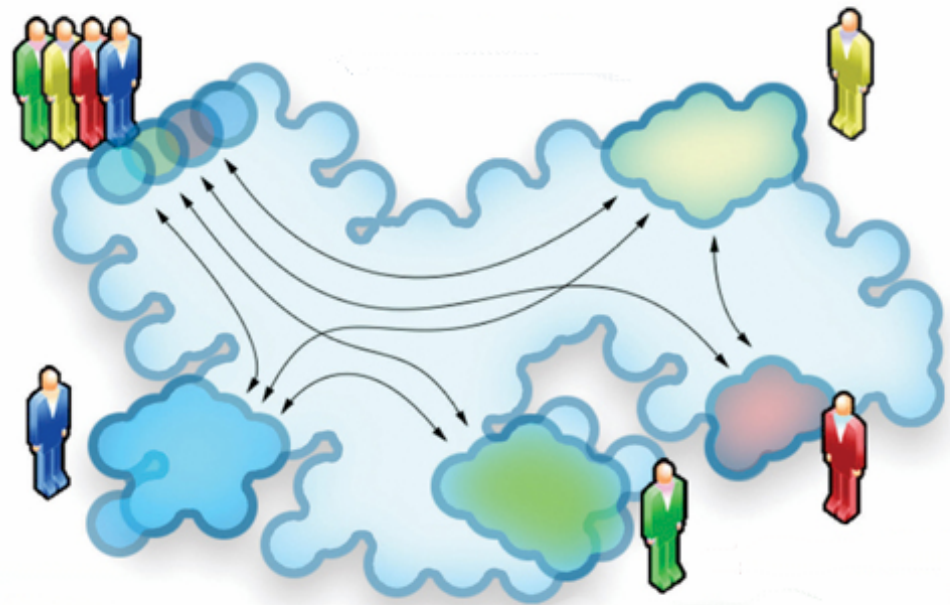


# Building Standard-Based Business Processes with Web Services

**Josef Schiefer**

**Vienna, November 2004**



# Agenda

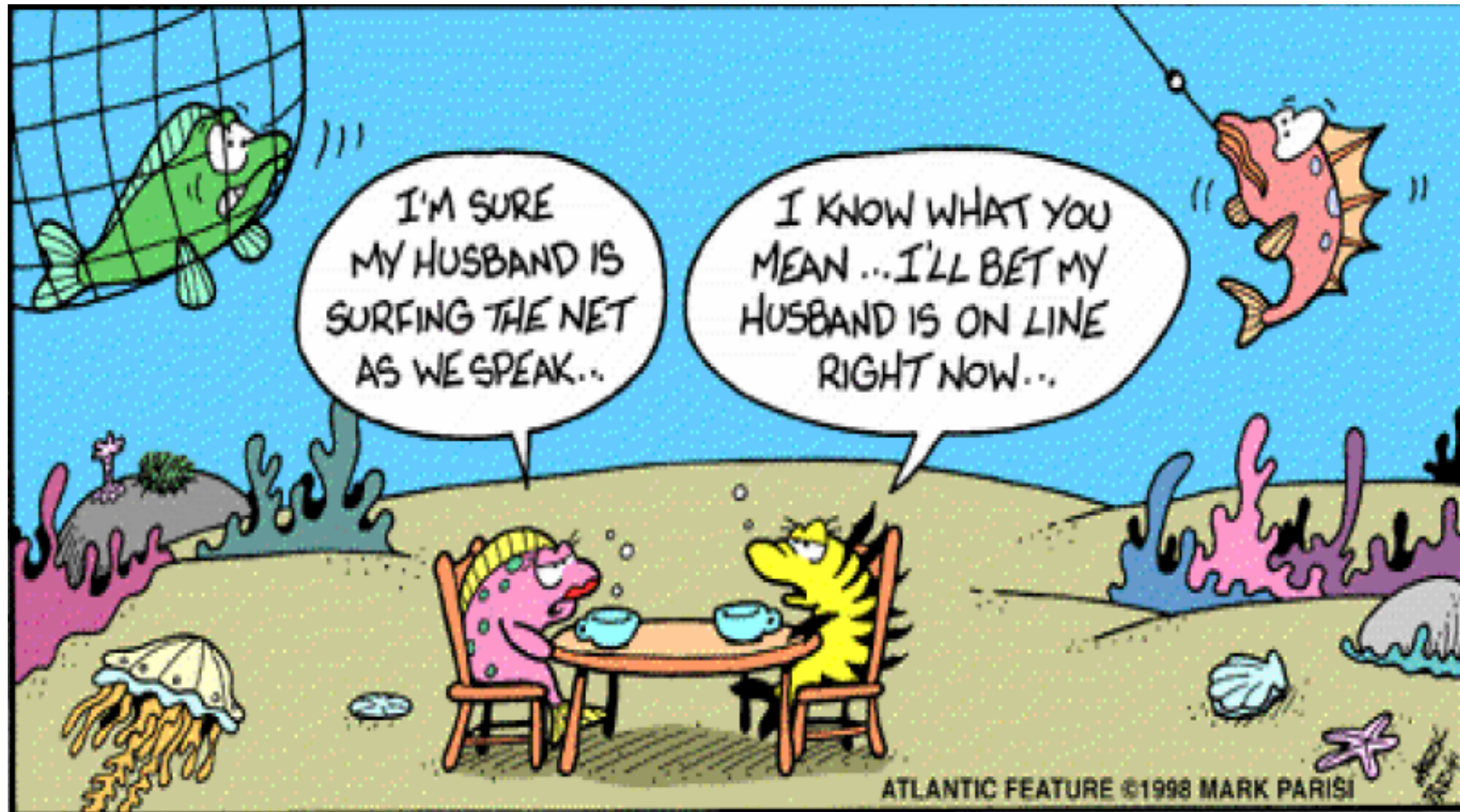
## Block 1

- » Motivation/Introduction
- » Orchestration vs Choreography
- » BPEL4WS - Basic Constructs
  - › Partner Links
  - › Main Flow Constructs
  - › Message Correlation
  - › Compensation Handlers
  - › Fault Handlers
  - › Event Handlers
- » Q&A

## Block 2

- » Details to all BPEL4WS Constructs
- » Demo with Oracle BPEL Process Manager
- » Conclusion + Future Trends
- » Business Process Monitoring with Senactive InTime
- » Q&A
  
- » “Diplomarbeitsthemen” in the area of business process management & monitoring

Please interrupt me if you have questions!!



# Integration...



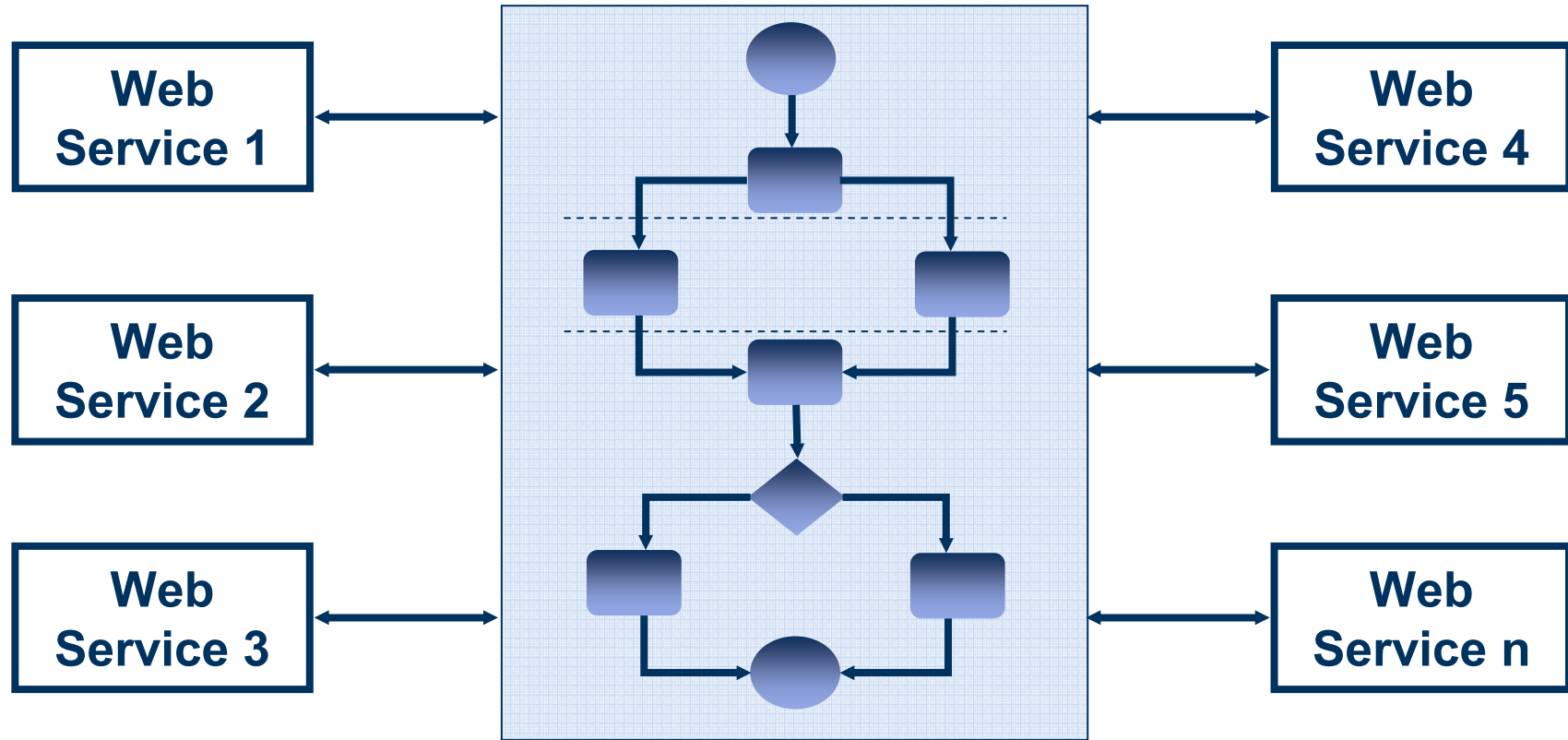
# Coordination...



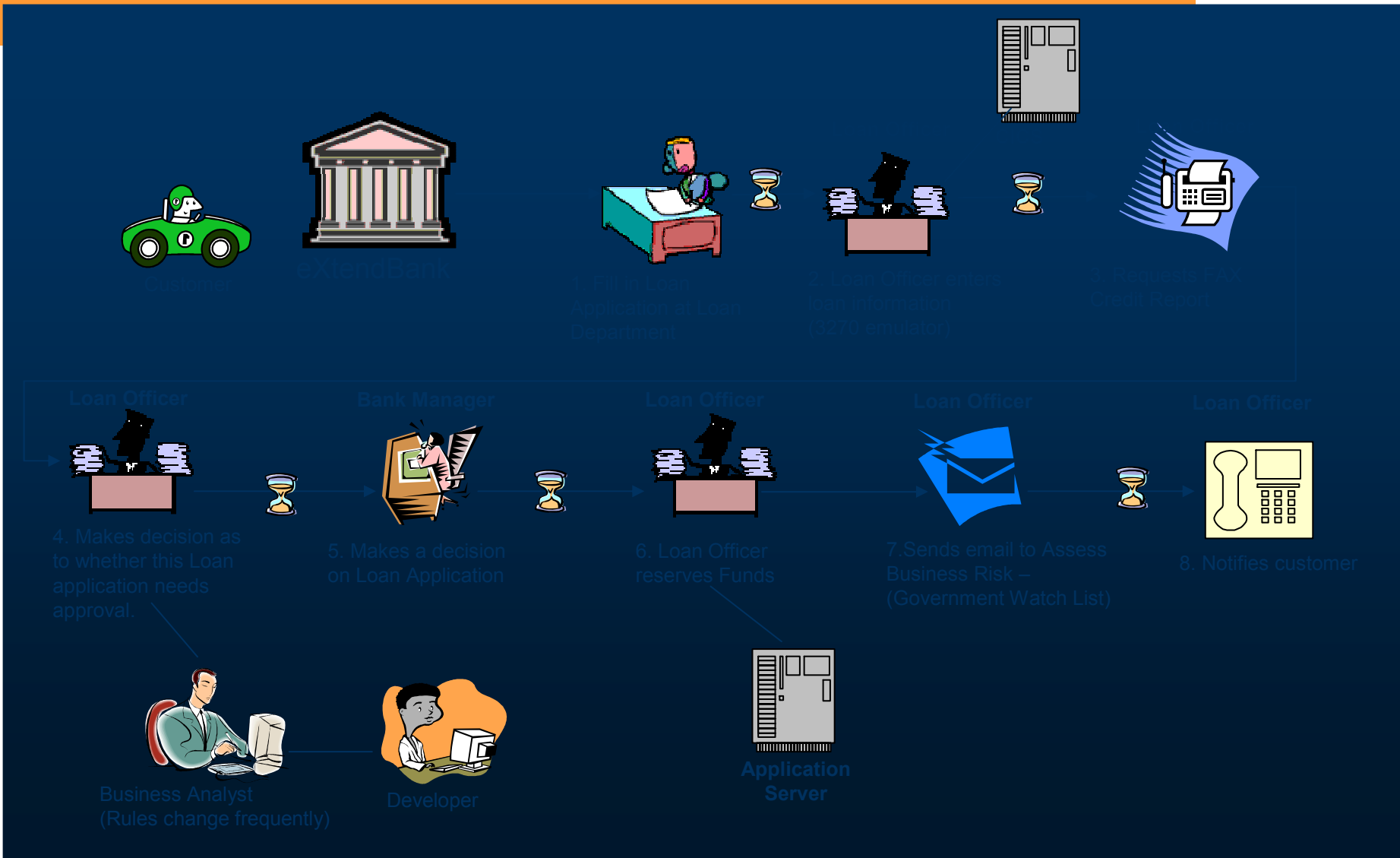


# Motivation/Introduction

# Web Services Meet Business Processes

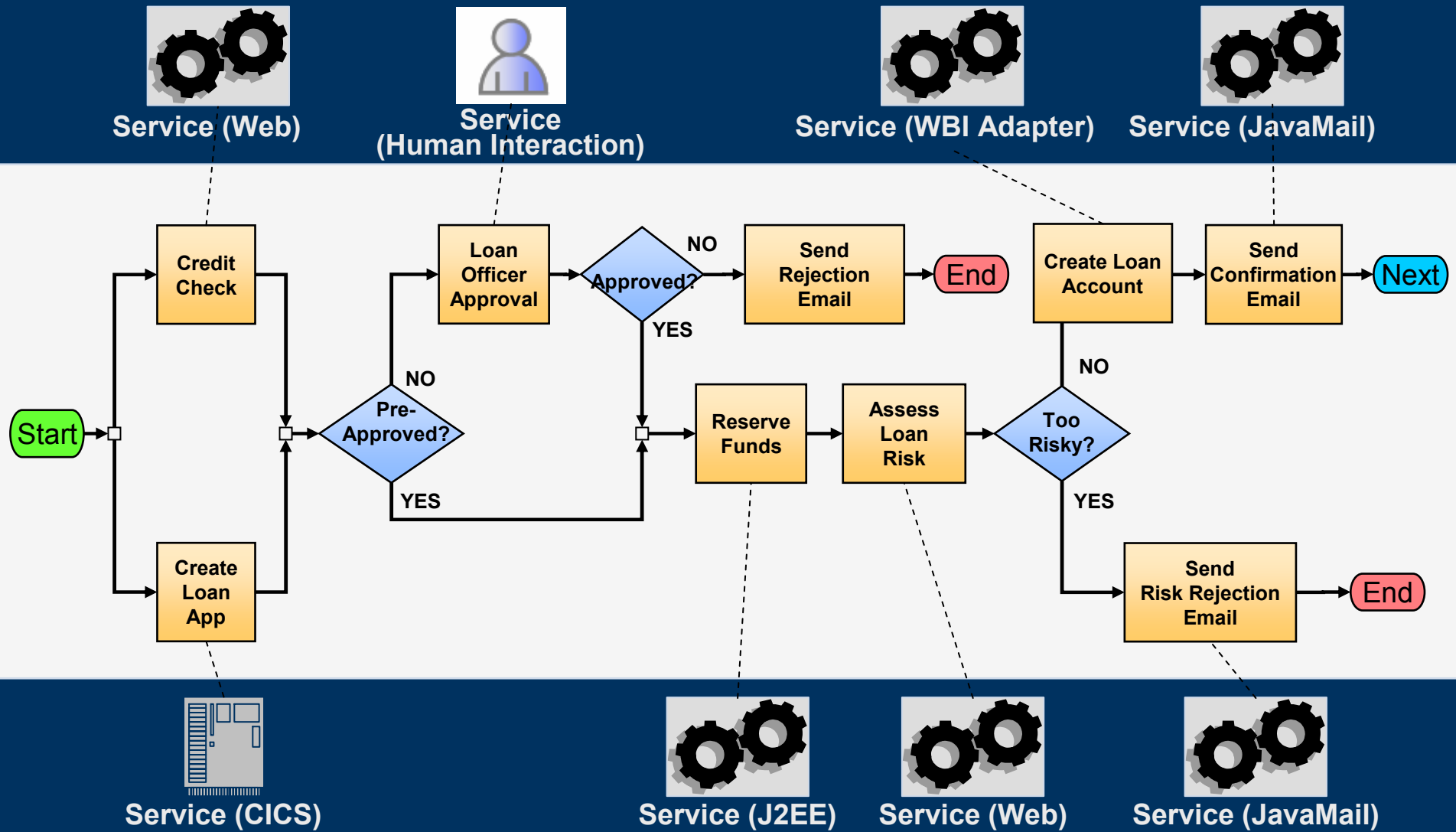


# eXtendBank – The OLD Loan Application System





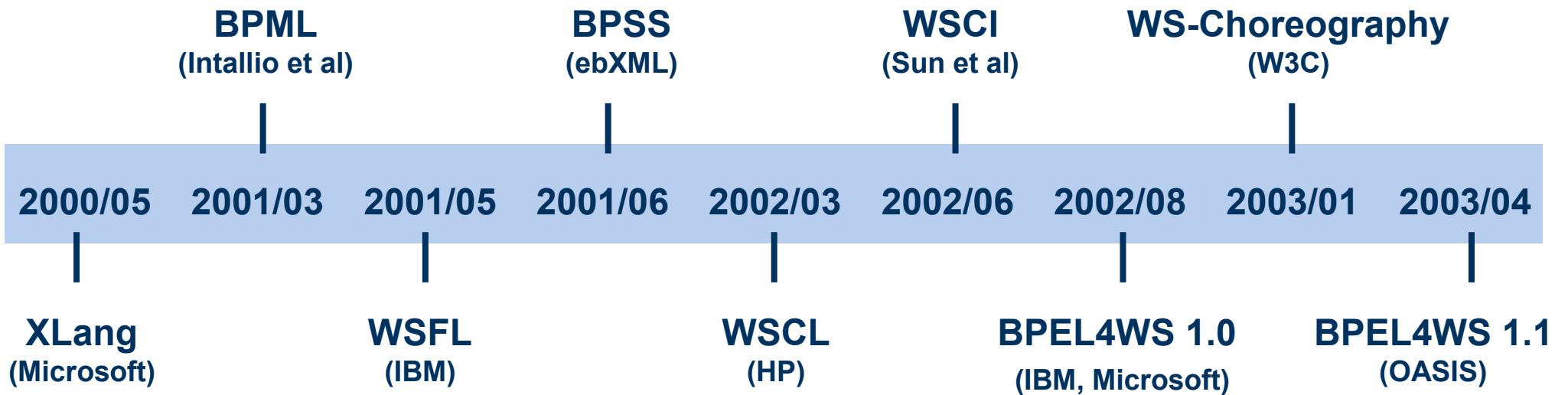
# eXtendBank: The new QuickLoan Process



# Business Process Challenges

- » Coordinate asynchronous communication between services
- » Correlate message exchanges between parties
- » Implement parallel processing of activities
- » Manipulate/transform data between partner interactions
- » Support for long running business transactions and activities
- » Provide consistent exception handling
- » ...

# Recent History of Business Process Standards



# Business Process Execution Language for Web Services (BPEL4WS)

Version 1.0 released by IBM, Microsoft and BEA in August 2002

- Accompanied by WS-Coordination, WS-Transaction

Version 1.1 submitted to OASIS April 2003

- BPEL4WS → WS-BPEL

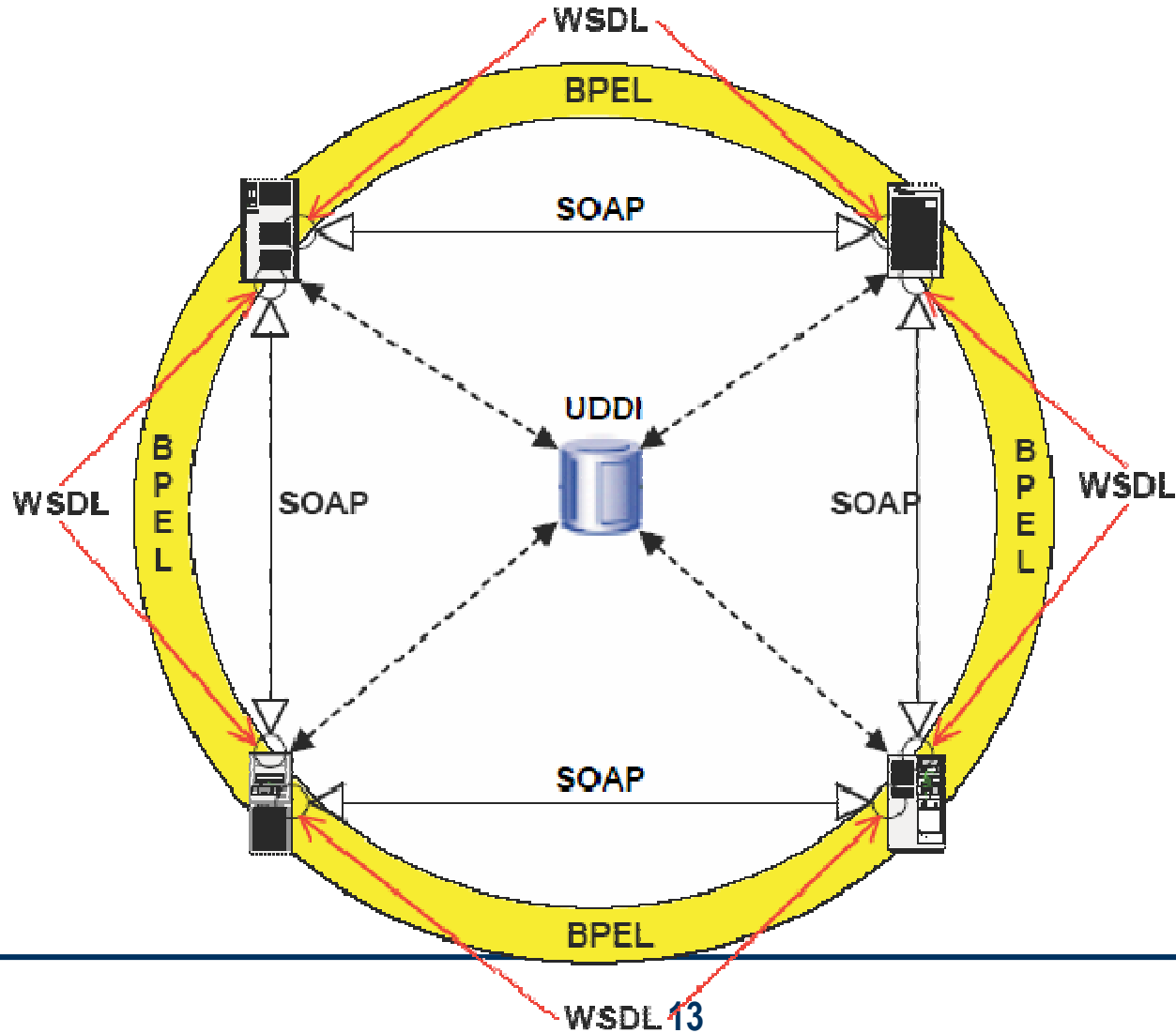
XML language for describing business processes based on Web services

- Convergence of XLANG (Microsoft) and WSFL (IBM)

Unprecedented industry consensus

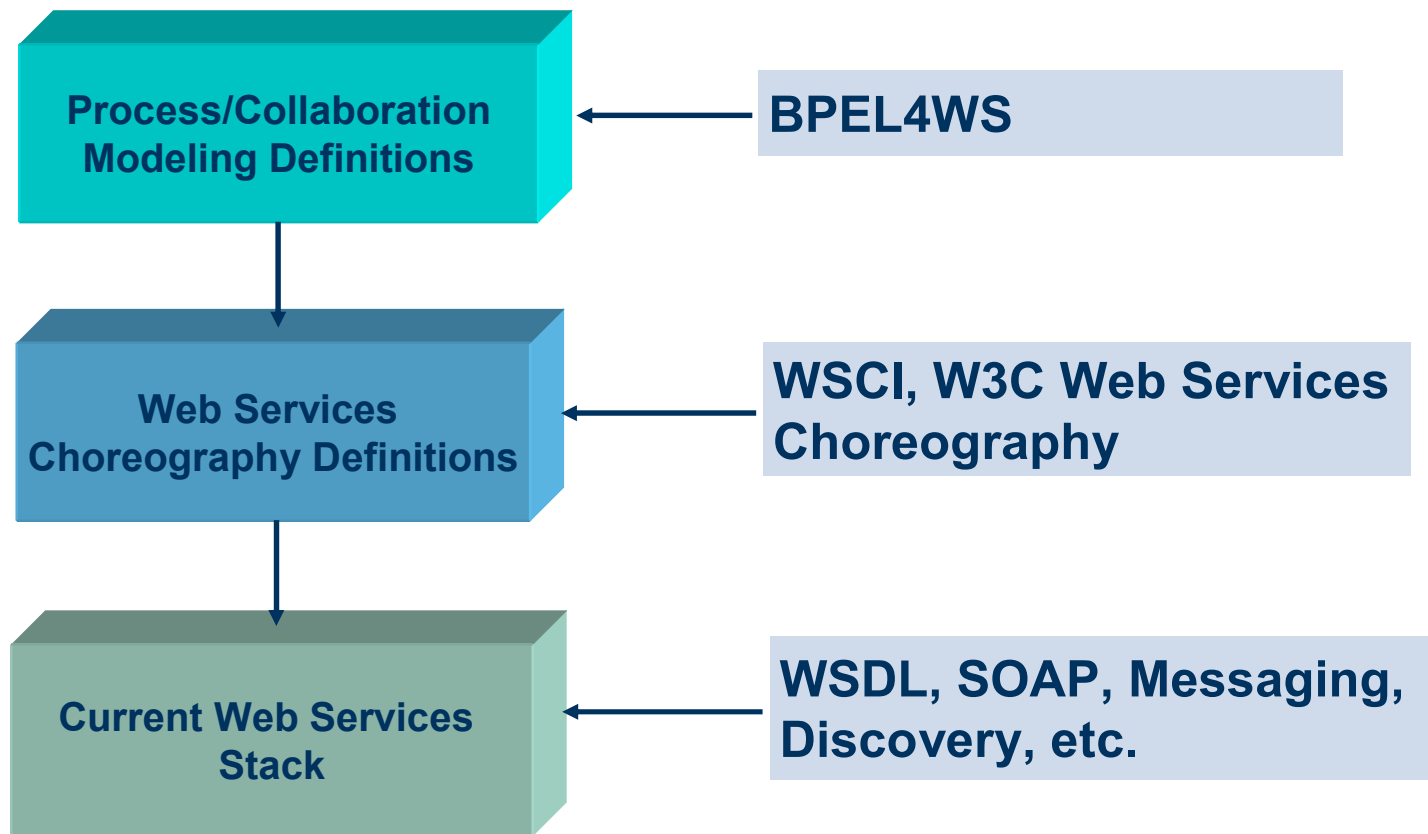
- IBM, Microsoft, Oracle, Sun, BEA, SAP, Siebel ...

# Interplay of BPEL4WS, Web Service, UDDI, WSDL, SOAP



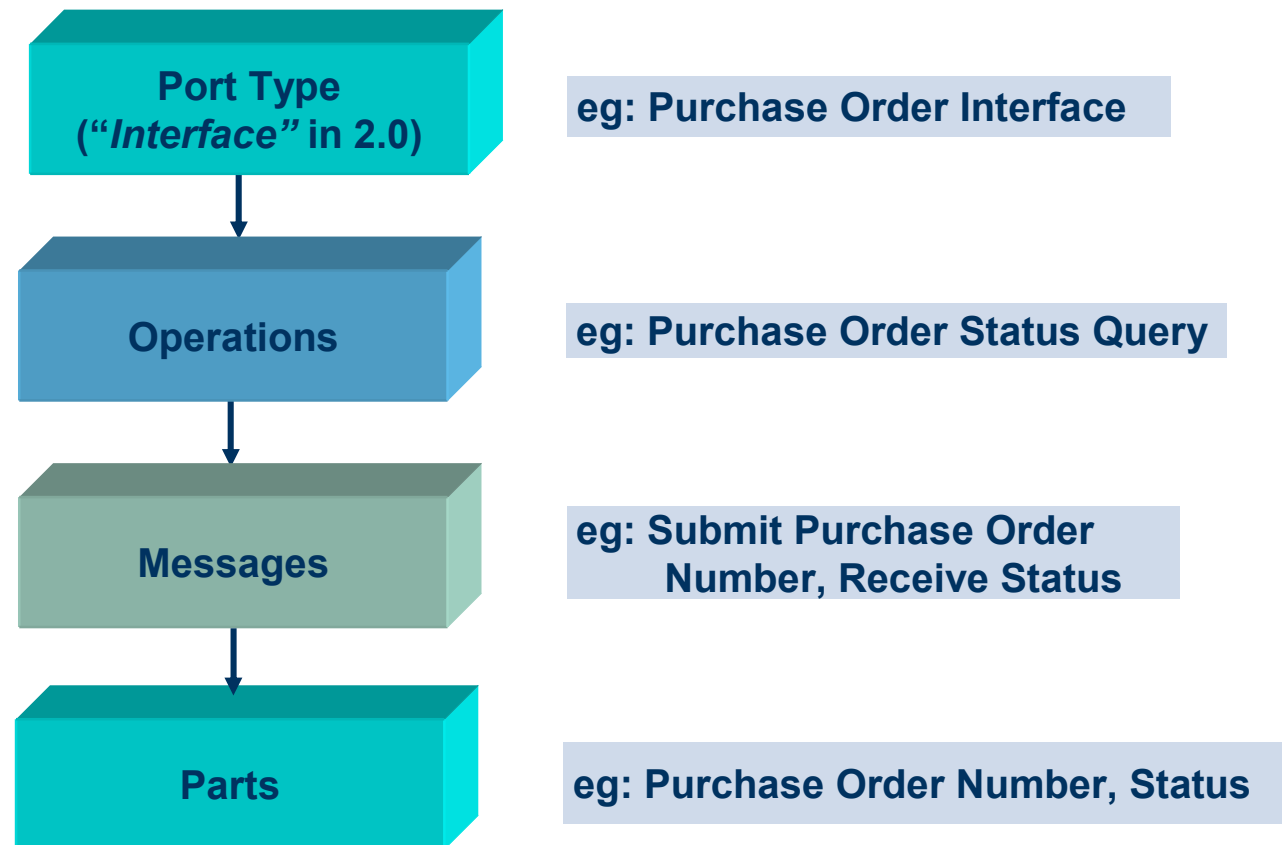
# Web Service Stack

## BPEL4WS is on Top of the Web Service Stack



# Das BPEL4WS Prozessmodell basiert auf dem Service-Modell von WSDL 1.1

WSDL specifies a hierarchy for describing Web Services characteristics in an abstract form:



# Standards Building Blocks of BPEL

Management	Orchestration – BPEL4WS			Business Processes
	Choreography – WSCI			
	WS-Reliability	WS-Security	Transactions	Quality of Service
			Coordination	
			Context	
	UDDI			Discovery
	WSDL			Description
	SOAP			Message
	XML			
	HTTP, IIOP, JMS, SMTP			Transport



# Value Proposition

## Portable business processes

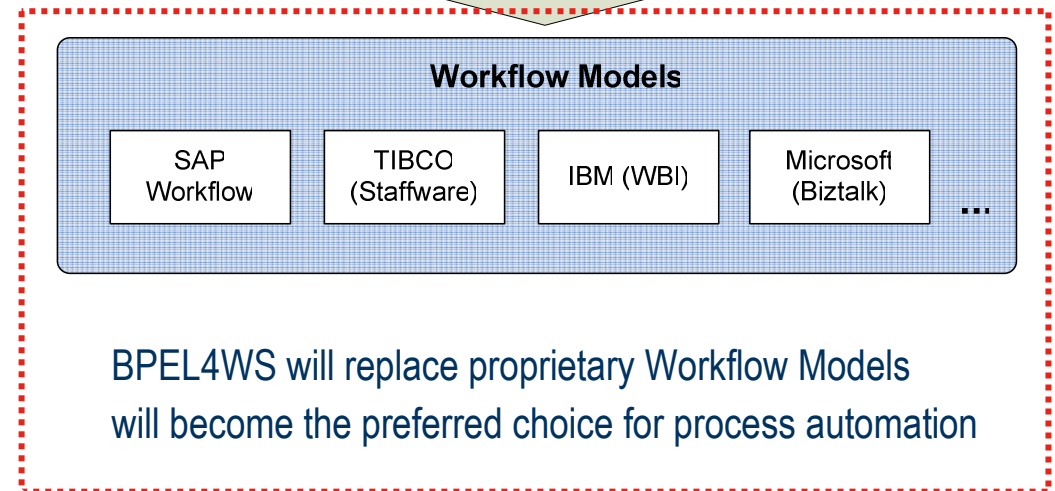
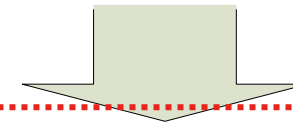
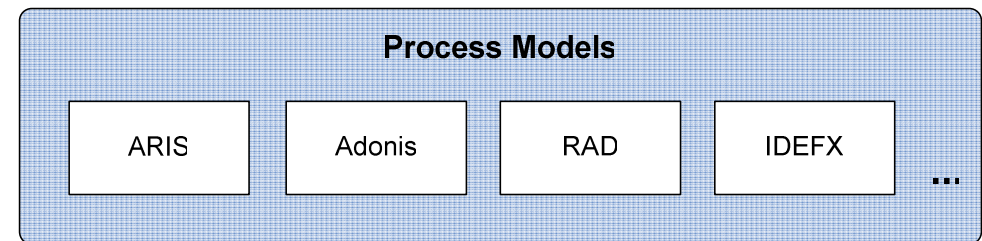
- » Built on top of an interoperable infrastructure of Web Services


## Industry wide language for business processes

- » Common skill set and language for developers

## Choice of process engines

- » Standards lead to competitive offerings





# Orchestration vs Choreography

# Orchestration vs Choreography

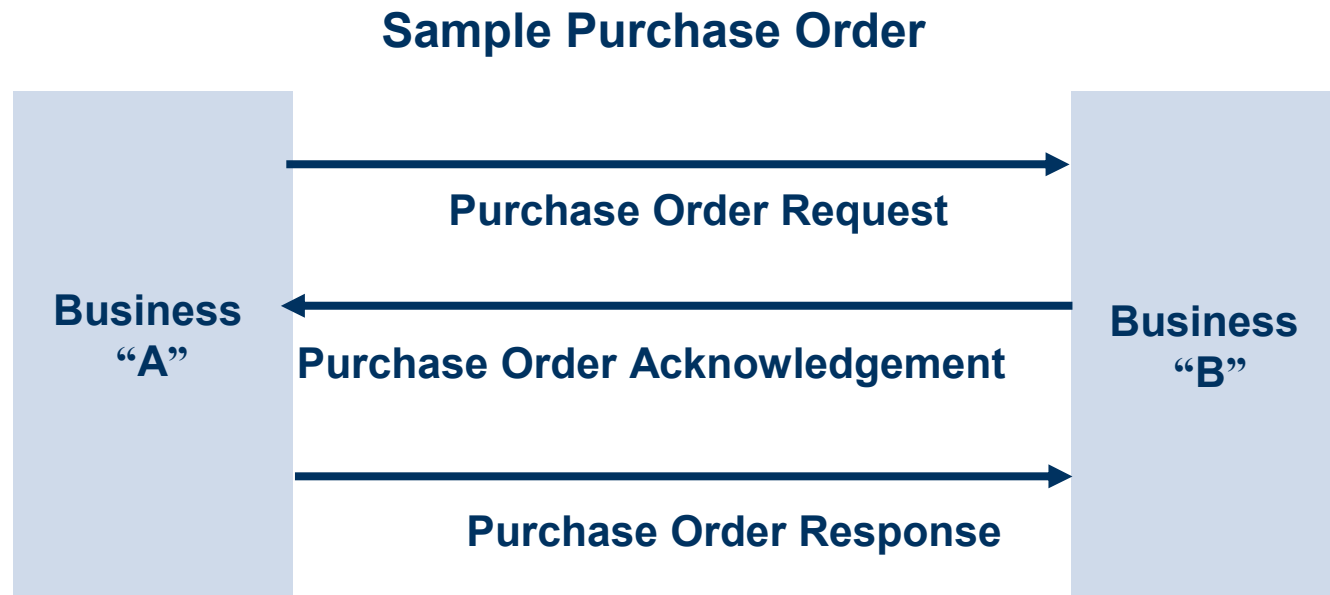
## Orchestration

- » An executable business process describing a flow from the perspective and under control of a single endpoint (commonly: Workflow)

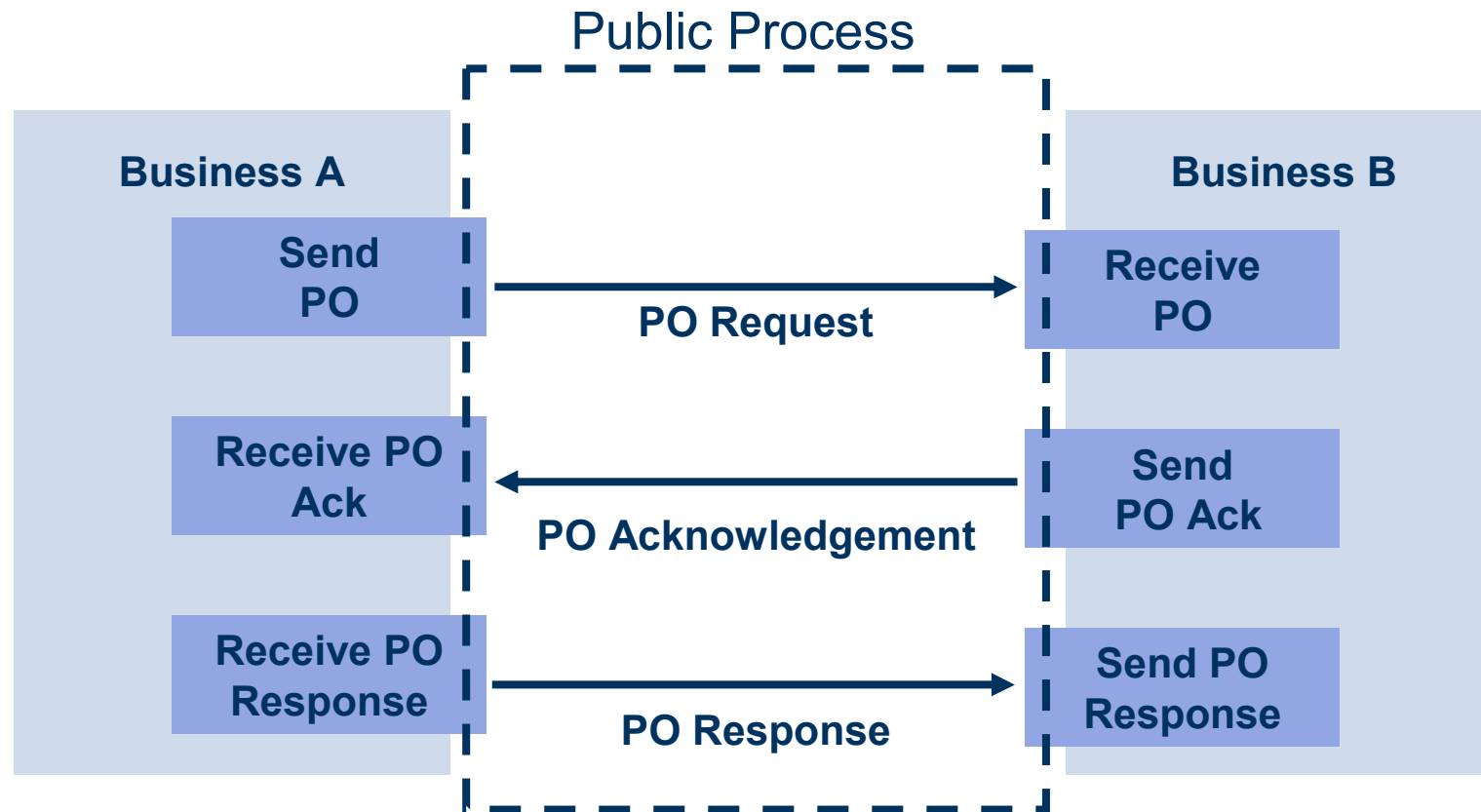
## Choreography

- » The observable public exchange of messages, rules of interaction and agreements between two or more business process endpoints

# Sample Business Process: Purchase Order

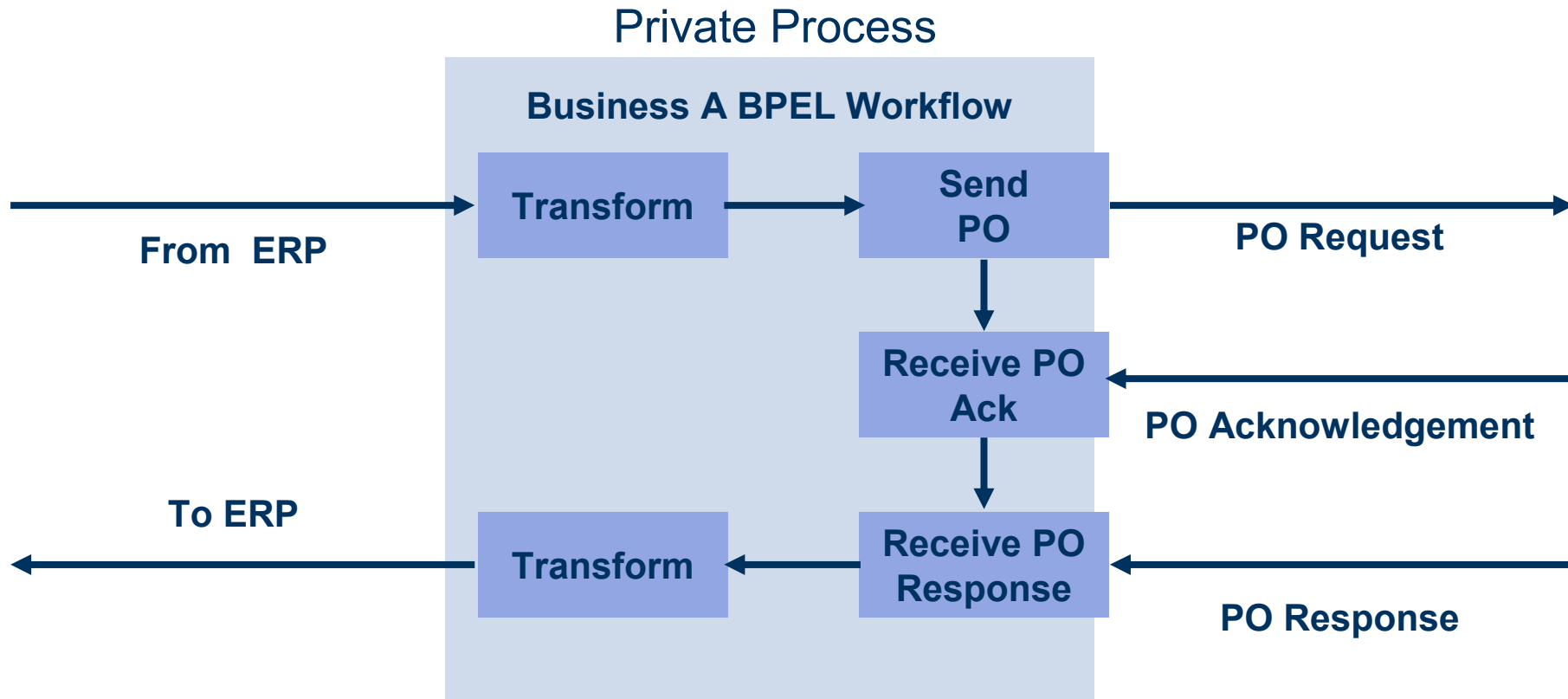


# From a Choreography Perspective



Choreography – The observable public exchange of messages

# From an Orchestration Perspective



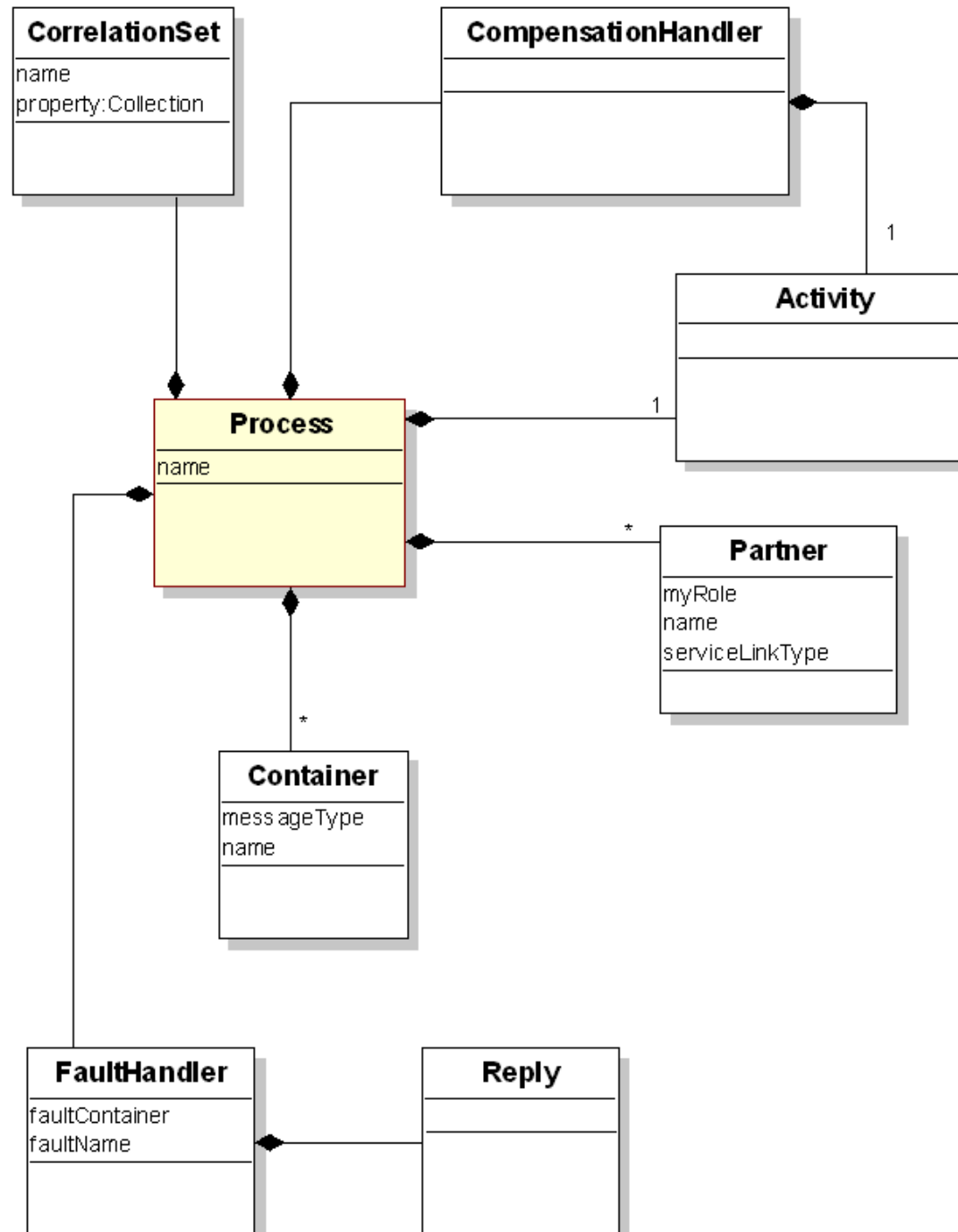
Orchestration – A private executable business process



# **BPEL4WS**

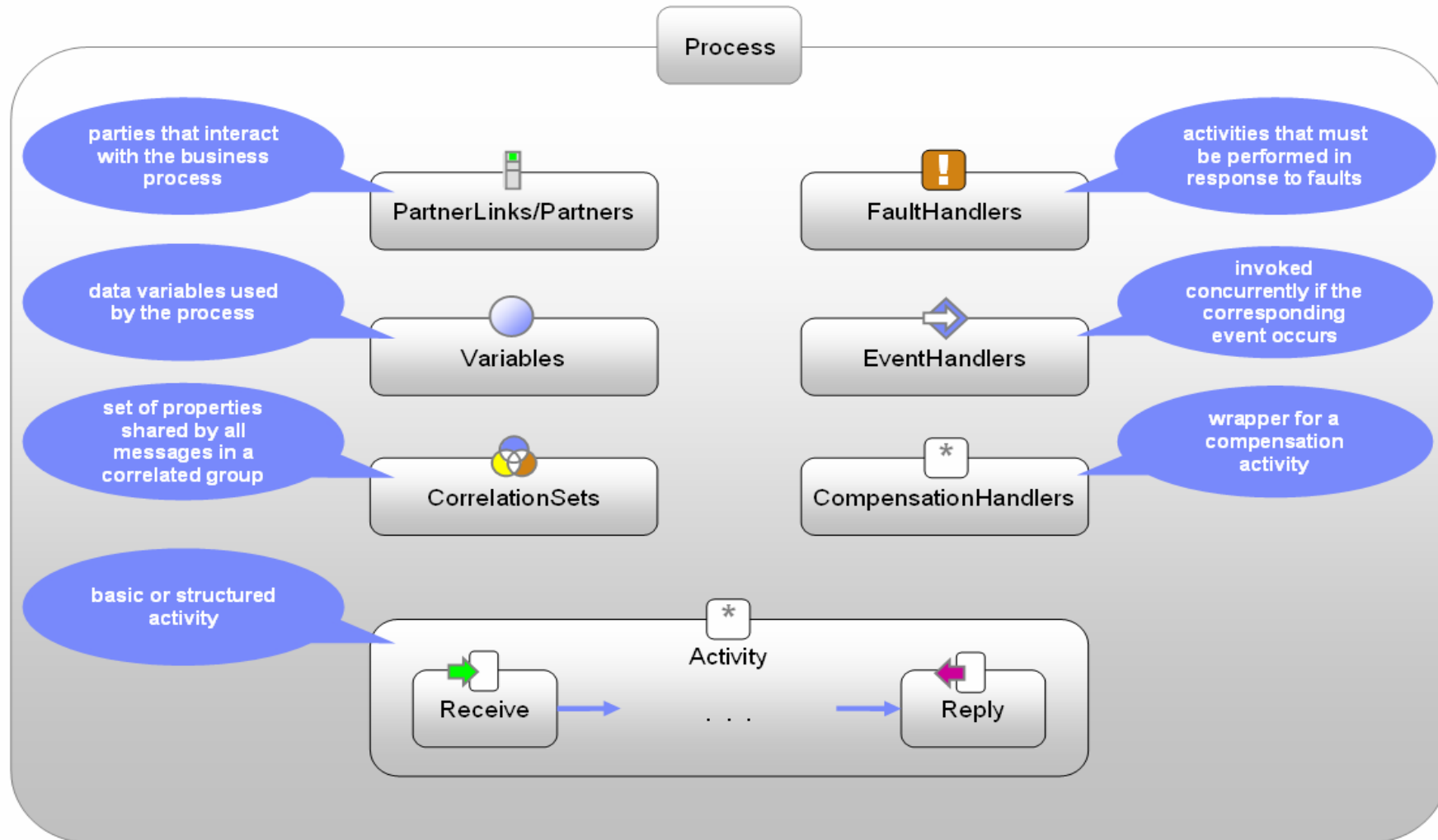
## **Basic Constructs**

# BPEL Process Meta Model





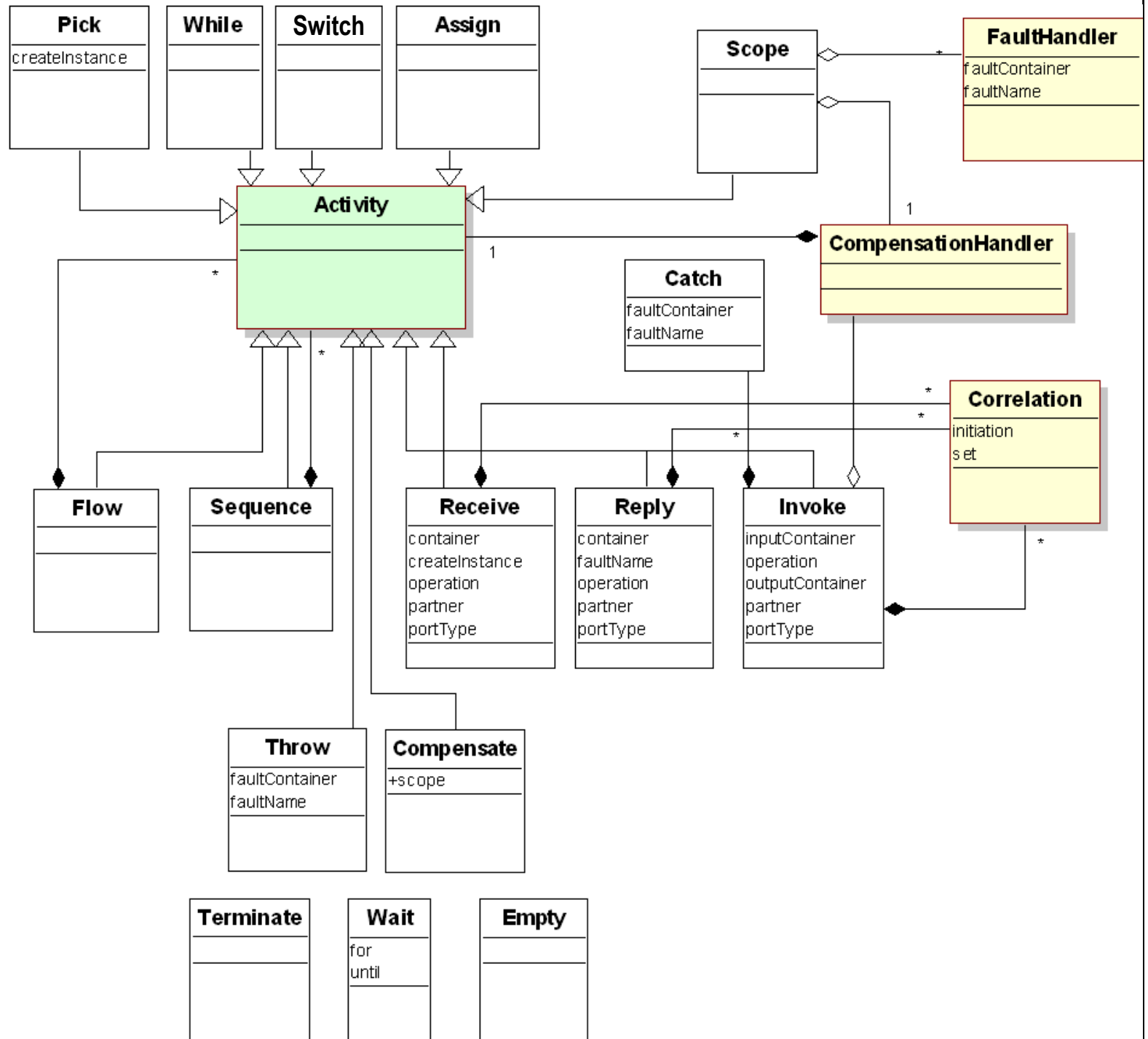
# BPEL4WS Overall Structure



# BPEL Scenario Structure

```
<process>
  <!-- Definition and roles of process participants -->
  <partnerLinks> ... </partnerLinks>
  <!-- Data/state used within the process -->
  <variables> ... </variables>
  <!-- Properties that enable conversations -->
  <correlationSets> ... </correlationSets>
  <!-- Exception handling -->
  <faultHandlers> ... </faultHandlers>
  <!-- Error recovery - undoing actions -->
  <compensationHandlers> ... </compensationHandlers>
  <!-- Concurrent events with process itself -->
  <eventHandlers> ... </eventHandlers>
  <!-- Business process flow -->
  (activities)*
</process>
```

# BPEL Activity Meta Model



# BPEL4WS Basic Activities



Do a blocking wait for a matching message to arrive



Generate a fault from inside the business process



Send a message in reply to a message that was received through a Receive



Immediately terminate the behavior of a business process instance



Invoke a one-way or request-response operation on a portType offered by a partner



Wait for a given time period or until a certain time has passed

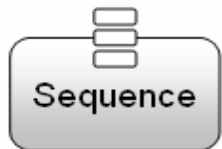


Update the values of variables or partner links with new data



Insert a "no-op" instruction into a business process

# BPEL4WS Structured Activities



Collection of activities to be performed sequentially in lexical order



Block and wait for a suitable message to arrive or for a time-out alarm to go off



Select exactly one branch of activity from a set of choices



Specify one or more activities to be performed concurrently



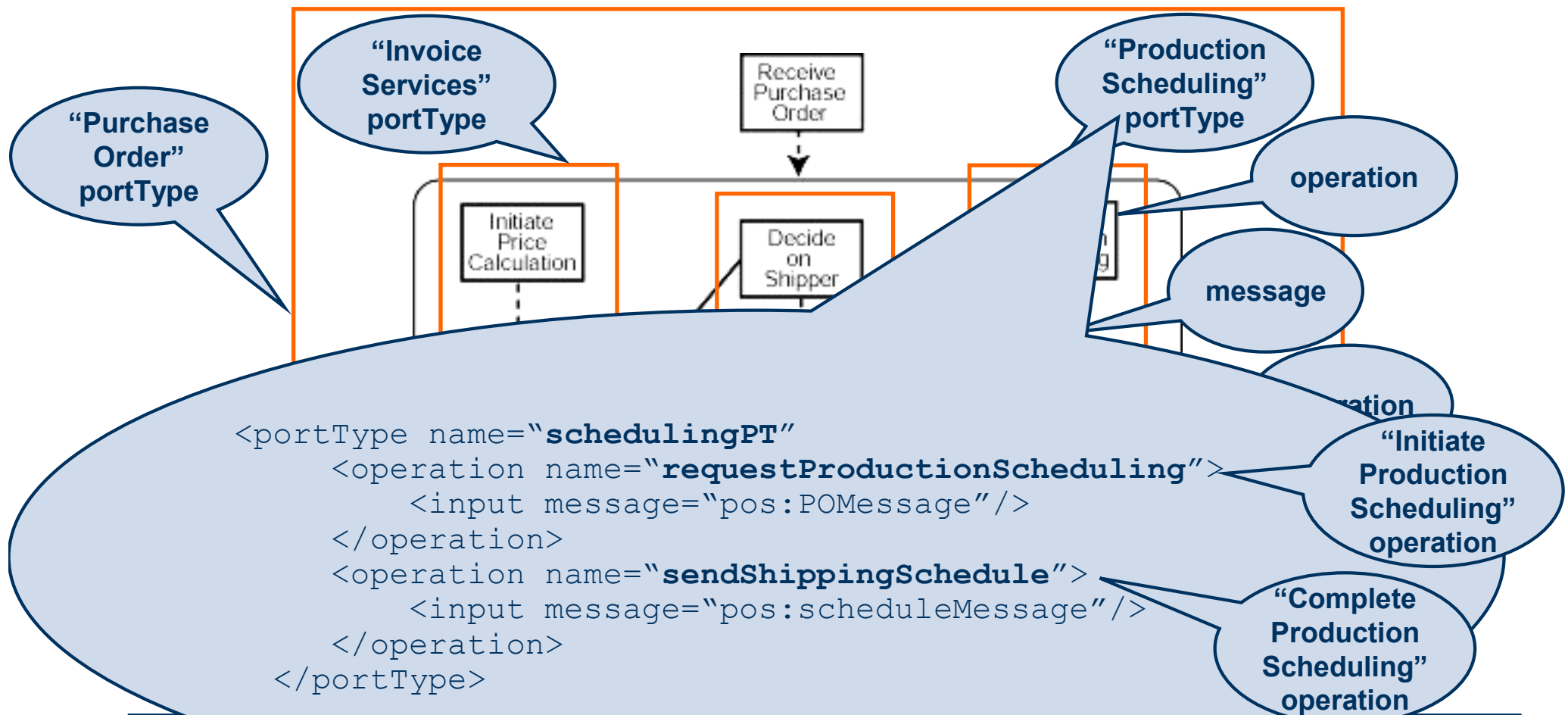
Indicate that an activity is to be repeated until a certain success criteria has been met



Define a nested activity with its own associated variables, fault handlers, and compensation handler

# BPEL4WS is capable of modeling complex business processes

The following is a BPEL4WS process for handling a **purchase order**:





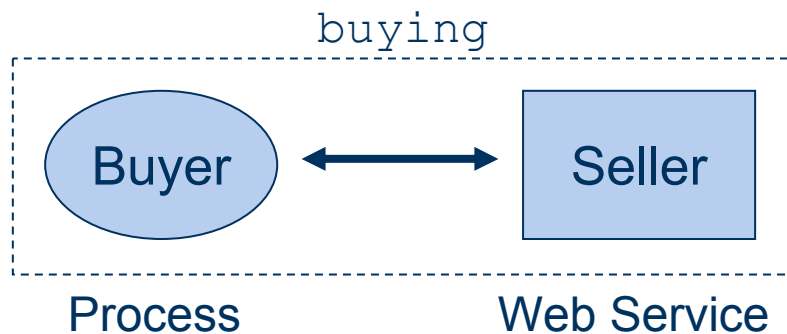
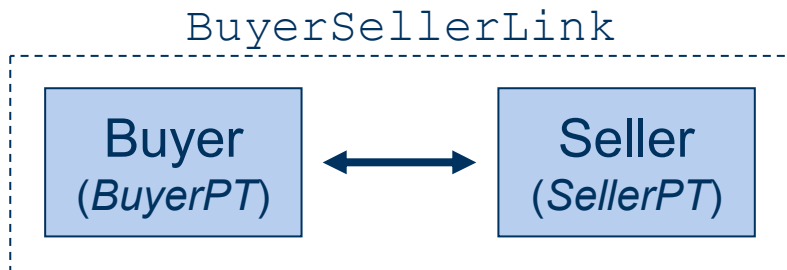
# Partner Links

## Partner, Partner Links, Partner Link Types, Endpoint References

- » Model peer-to-peer conversational relationships with partners
- » Define interaction channels between partners
- » **Partner Link Types:** Characterize relationships between two services by defining the „*roles*“ played by each of the services and specifying the *portType* provided by each service
- » **Partner Links:** Are used to represent interactions between a service and each of the parties with which it interacts
- » **Endpoint Reference:** Selection of service providers and invocation of their operations. Can be used in Partner Links.
- » **Partners:** A subset of the partner links of the process



# Partner Link Types, Partner Links



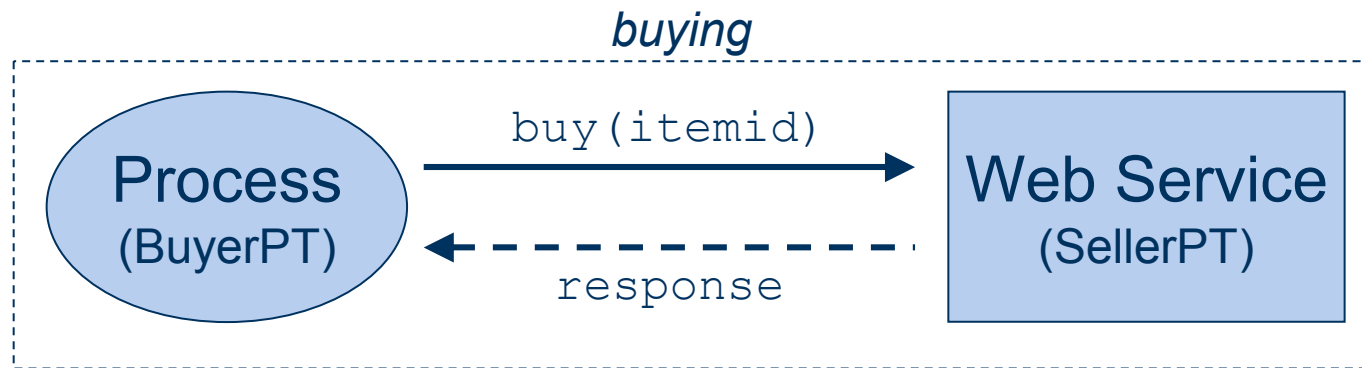
## Partner Link Types

```
<partnerLinkType
  name="BuyerSellerLink">
  <role name="Buyer">
    <portType name="BuyerPT"/>
  </role>
  <role name="Seller">
    <portType name="SellerPT"/>
  </role>
</partnerLinkType>
```

## Partner Links

```
<partnerLinks>
  <partnerLink
    name="buying"
    partnerLinkType="BuyerSellerLink"
    myRole="Buyer"
    partnerRole="Seller"/>
</partnerLinks>
```

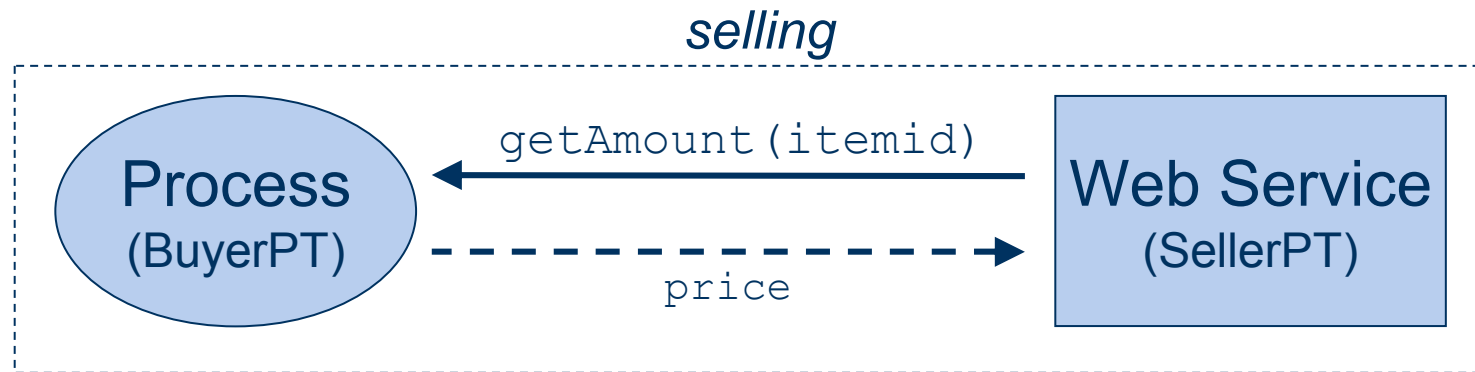
# Partner Links



Using partner links in the <invoke> activity:

```
<invoke  
  partnerLink="buying"  
  portType="SellerPT"  
  operation="buy"  
  inputVariable="itemid"  
  outputVariable="response" />
```

# Partner Links



## Incoming calls with blocking <receive> activity

» Creates a new process instance

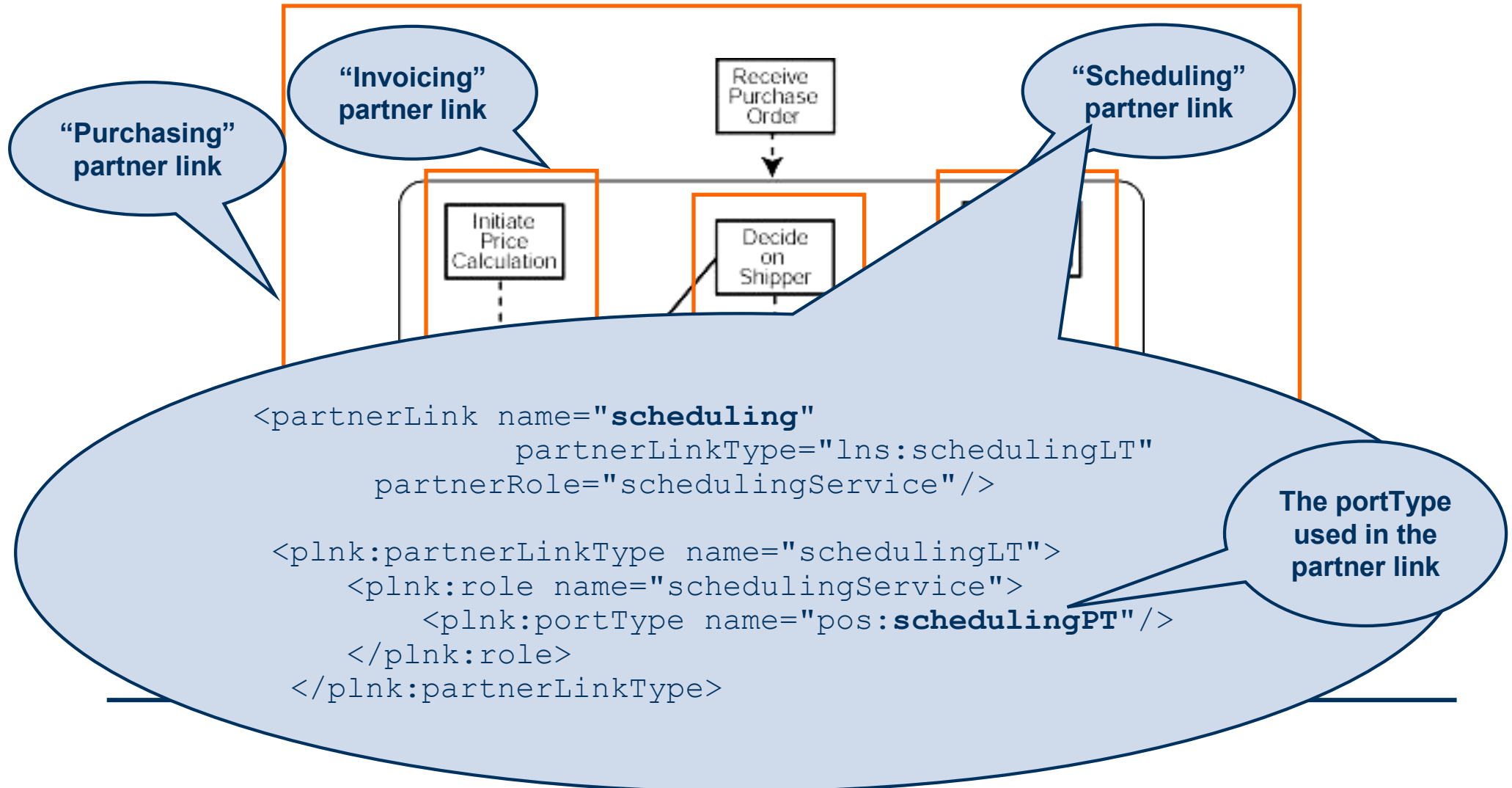
```
<receive partnerLink="selling" portType="SellerPT"
  operation="getAmount" variable="itemid"
  createInstance="yes"/>
```

## Result via <reply> activity

```
<reply partnerLink="selling" portType="SellerPT"
  operation="buy" variable="price"/>
```

# Partner Links

Partner links define the **messages** and **port types** used in the interactions in both directions, along with role names



# Endpoint References

BPEL4WS uses “endpoint references” for dynamic selection of service providers and invocation of their operations

The relevant information about a partner service can be set up as part of business process deployment

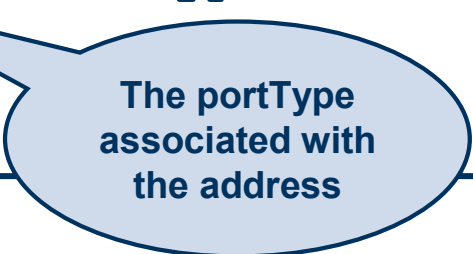
- This is a more “static” approach

However, it is also possible to **select and assign partner services dynamically**

BPEL4WS leverages the **WS-Addressing** specification for this capability

- WS-Addressing defines a **standard representation for endpoint references** that incorporates information from a WSDL description as well as policy information:

```
<wsa:EndpointReference xmlns:wsa="...">  
  <wsa:Address>http://www.someendpoint.com</wsa:Address>  
  <wsa:PortType>PurchaseOrderPortType</wsa:PortType>  
</wsa:EndpointReference>
```



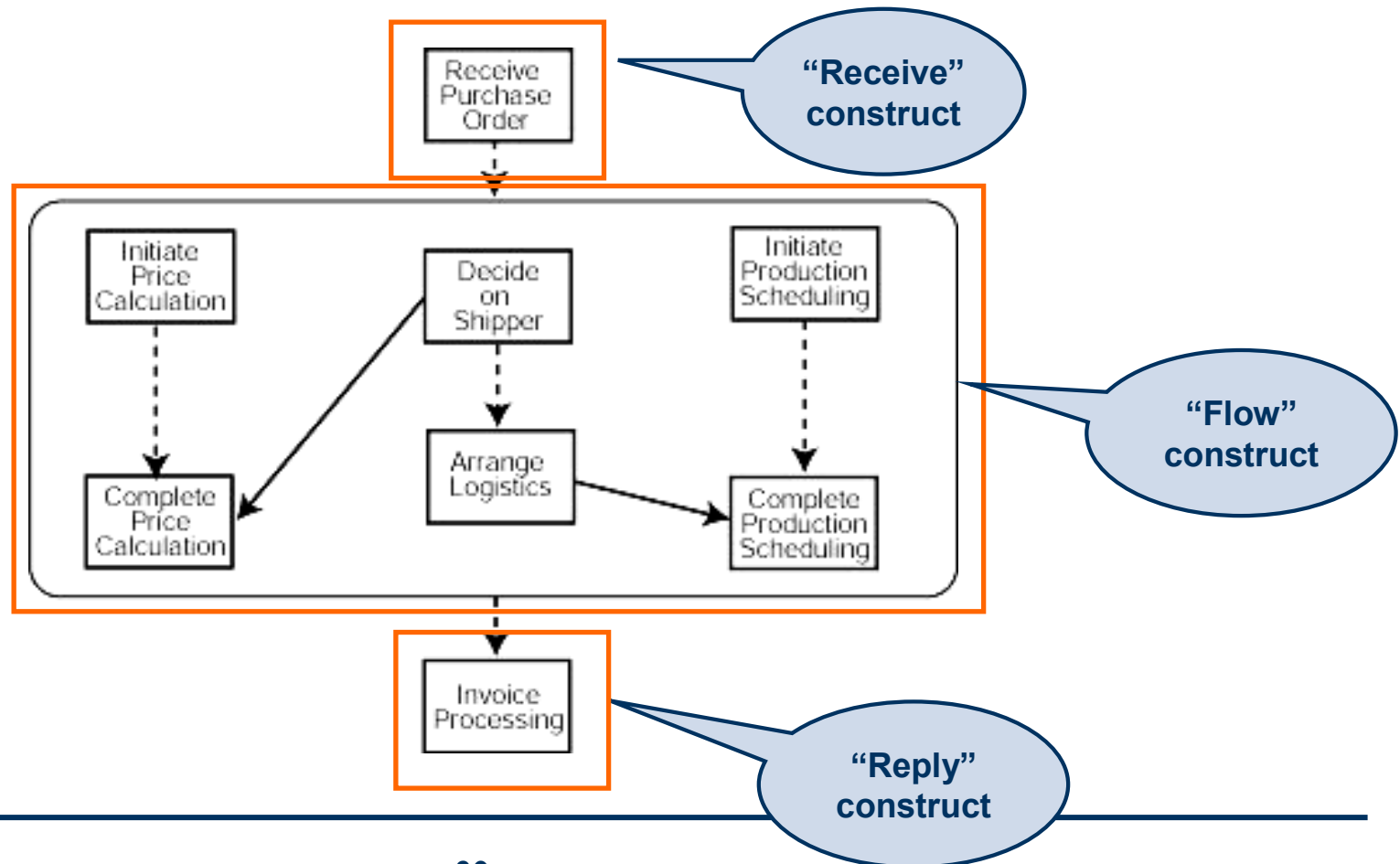
The portType  
associated with  
the address



# Main Flow Constructs

# Main BPEL4WS constructs: “Receive”, “Flow” and “Reply” 1/3

The purchase order example uses **all three constructs**



## Main BPEL4WS constructs: “Receive”, “Flow” and “Reply” 2/3

The **receive** construct allows a process to do a **blocking wait** for a matching message to arrive

```
<receive partnerLink="purchasing"  
  portType="lms:purchaseOrderPT"  
  operation="sendPurchaseOrder"  
  variable="PO">  
</receive>
```

Wait to receive a purchase order on the “Purchasing” partner link

Represents the purchase order message

The **flow** construct allows one or more activities to be **performed concurrently**



## Main BPEL4WS constructs: “Receive”, “Flow” and “Reply” 3/3

The **reply** construct allows a process to **send a message in reply** to a message that was received through a **<receive>**

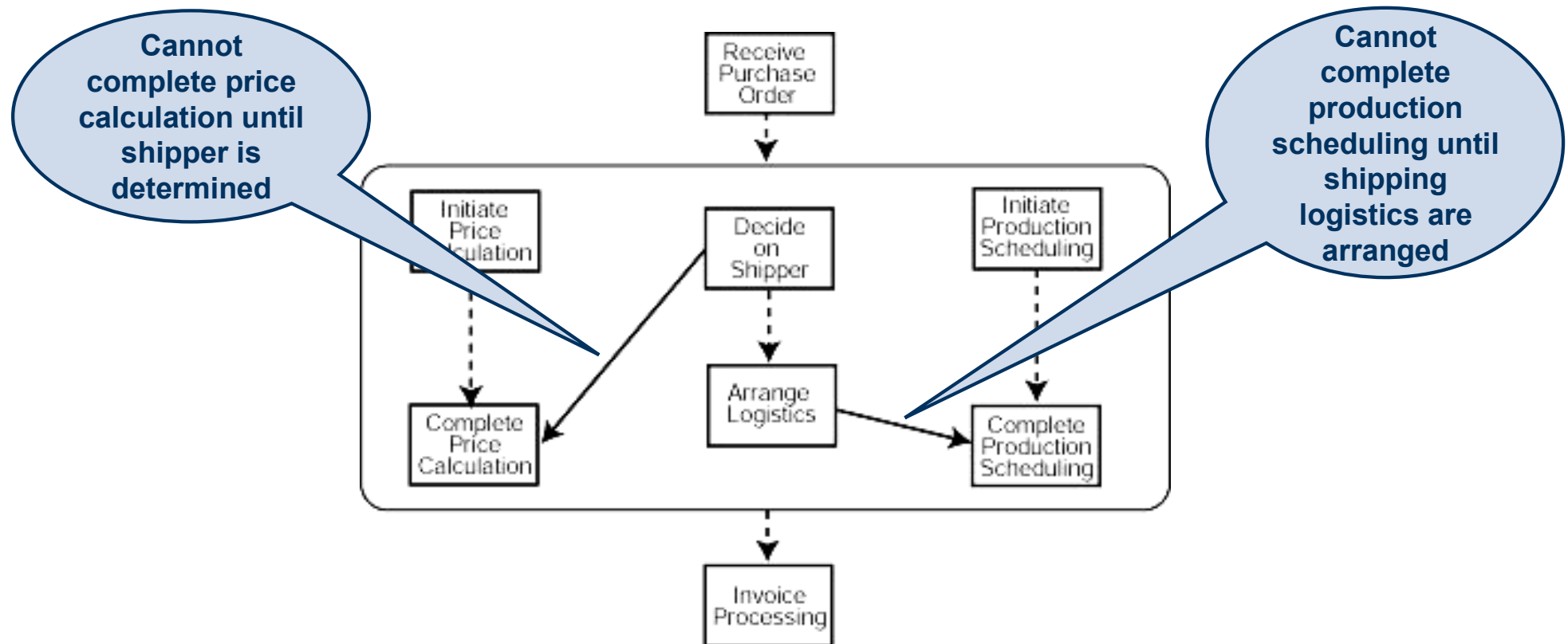
```
<reply partnerLink="purchasing"  
      portType="lns:purchaseOrderPT"  
      operation="sendPurchaseOrder"  
      variable="Invoice">  
</reply>
```

Send invoice  
on the  
“Purchasing”  
partner link

Represents  
the invoice  
message

# Modeling dependencies between activities

There are **several dependencies** in the purchase order example



# The synchronization dependencies between concurrent tasks are expressed by using “links” to connect them 1/2

Dependency links have to be defined in the <links> section:

```
<flow>
  <links>
    <link name="ship-to-invoice"/>
    <link name="ship-to-scheduling"/>
  </links>
```

... activities use the links as source and targets

```
</flow>
```

# The synchronization dependencies between concurrent tasks are expressed by using “links” to connect them 2/2

The following represents the **dependency of the price calculation on the shipper selected**:

```
<invoke partnerLink="shipping"
  portType="lns:shippingPT"
  operation="requestShipping"
  inputVariable="shippingRequest">
  outputVariable="shippingInfo">
  <source linkName="ship-to-invoice" />
</invoke>
```

This represents the “Decide on Shipper” activity

```
<invoke partnerLink="invoicing"
  portType="lns:computePricePT"
  operation="sendShippingPrice"
  inputVariable="shippingInfo"
  <target linkName="ship-to-invoice" />
</invoke>
```

The common link name represents a dependency between the two activities

This represents the “Complete Price Calculation” activity

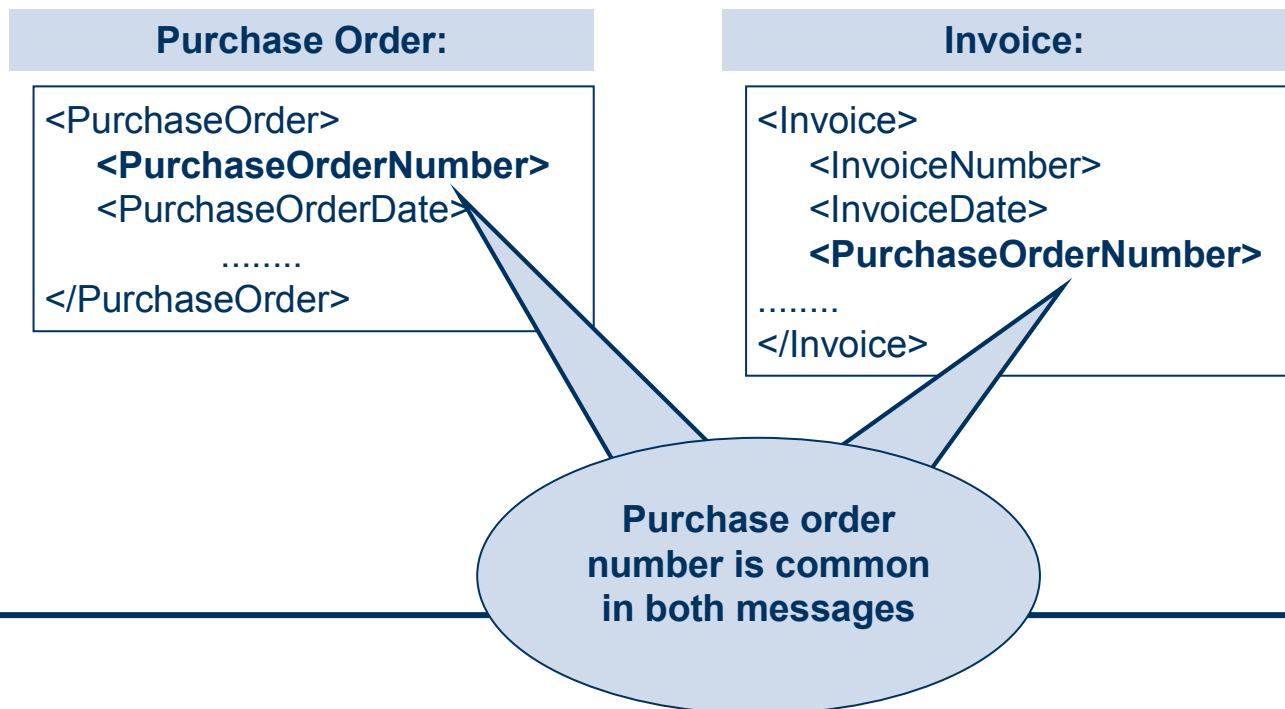


# Message Correlation

# Message correlation involves the association of two or more messages with each other in an asynchronous environment

This may be done by **associating contents** in a given message with its correlating message

- For example, in a purchase order/invoice scenario, the invoice may contain the corresponding **purchase order number**



# BPEL4WS represents message correlations using “correlation sets”

A **correlation set** contains a set of properties shared by **all messages in a correlated group**

```
<receive partnerLink="Buyer" portType="SP:PurchasingPT"
  operation="AsyncPurchase" variable="PO">
  <correlations>
    <correlation set="PurchaseOrder" initiate="yes">
  </correlations>
</receive>

<invoke partnerLink="Buyer" portType="SP:BuyerPT"
  operation="AsyncPurchaseResponse" inputVariable="POResponse">
  <correlations>
    <correlation set="PurchaseOrder" initiate="no" pattern="out">
    <correlation set="Invoice" initiate="yes" pattern="out">
  </correlations>
</invoke>

<correlationSet name="PurchaseOrder"
  properties="cor:customerID cor:orderNumber"/>

<correlationSet name="Invoice"
  properties="cor:vendorID cor:invoiceNumber"/>
```



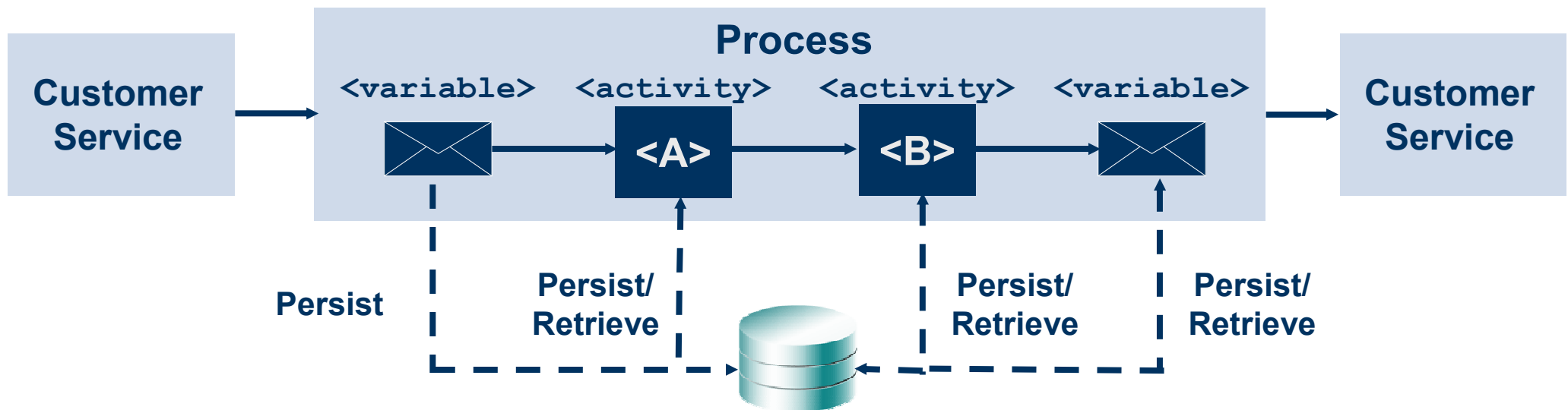
# Variables



# Variables

Messages sent and received from partners

- » Persisted for long running interactions
- » Defined in WSDL types and messages



# Variables in BPEL

## BPEL:

```
<variables>
  <variable name="PO" messageType="lns:POMessage"/>
  <variable name="Invoice" messageType="lns:InvMessage"/>
  <variable name="POFault" messageType="lns:orderFaultType"/>
</variables>
```

## Purchase Process WSDL:

```
<message name="POMessage">
  <part name="customerInfo" type="sns:customerInfo"/>
  <part name="purchaseOrder" type="sns:purchaseOrder"/>
</message>
<message name="InvMessage">
  <part name="IVC" type="sns:Invoice"/>
</message>
<message name="orderFaultType">
  <part name="problemInfo" type="xsd:string"/>
</message>
```

## How is Data Manipulation Done?

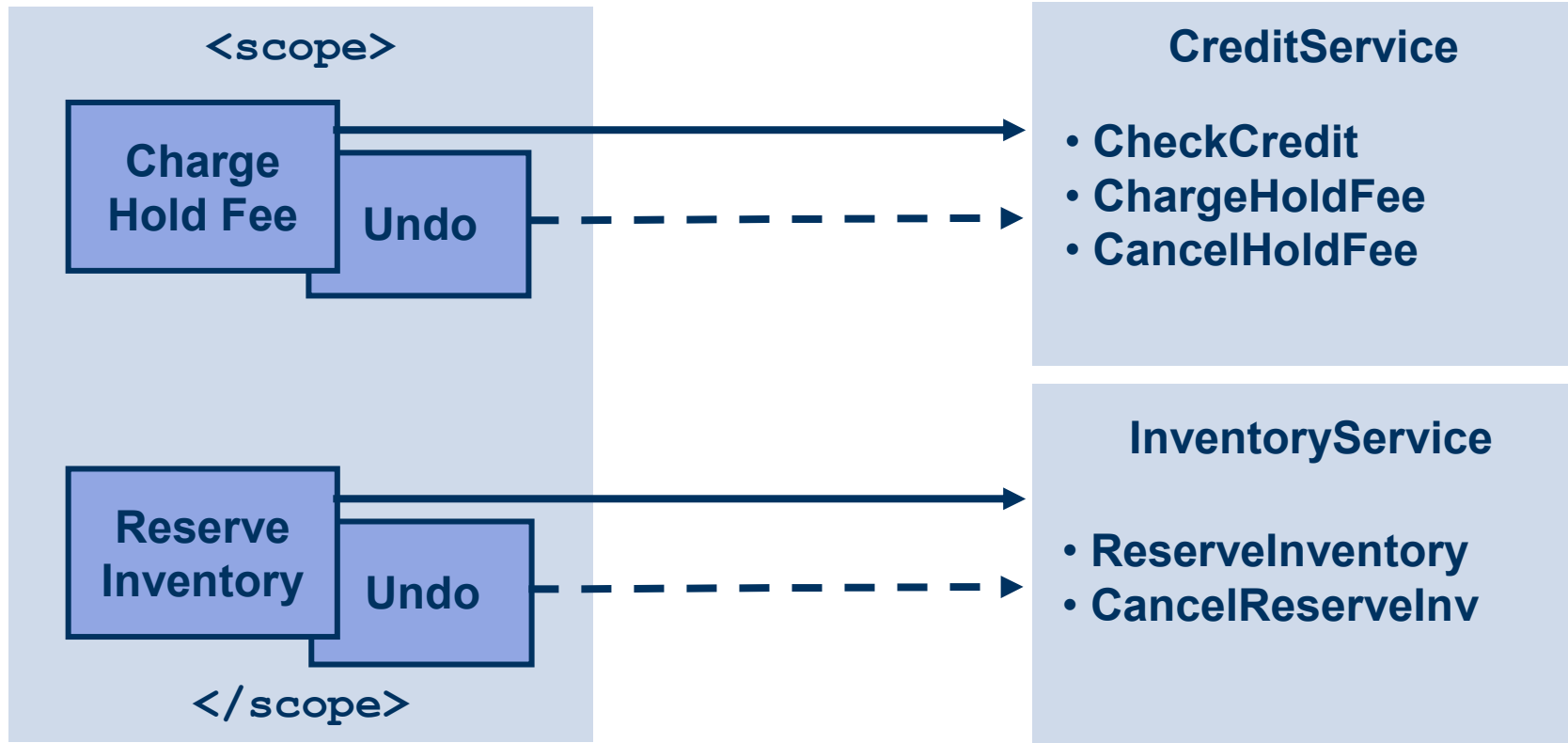
Using `<assign>` and `<copy>`, data can be copied and manipulated between variables  
`<copy>` supports XPath queries to sub-select data

```
<assign>  
  <copy>  
    <from variable="PO" part="customerInfo" />  
    <to variable="creditRequest" part="customerInfo" />  
  </copy>  
</assign>
```



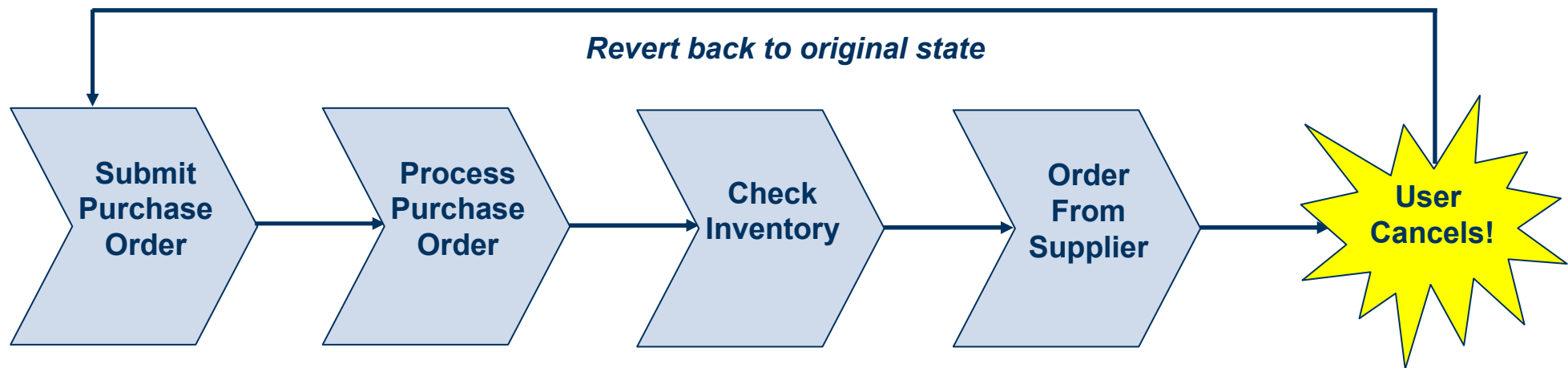
# Compensation Handlers

# Long Running Transactions and Compensation



# Long Running Transactions and Compensation Handlers

Consider a situation in which a user **cancels a purchase order**:



In this situation, it is **not possible to lock system resources** (ex: database records) for extended periods of time

- Therefore, the **partial work must be undone** as best as possible

# Compensation Handlers in BPEL

```
<scope>
  <compensationHandler>
    <invoke partnerLink="Seller" portType="SP:Purchasing"
      operation="CancelPurchase"
      inputVariable="getResponse"
      outputVariable="getConfirmation">
      <correlations>
        <correlation set="PurchaseOrder" pattern="out"/>
      </correlations>
    </invoke>
  </compensationHandler>
  <invoke partnerLink="Seller" portType="SP:Purchasing"
    operation="SyncPurchase"
    inputVariable="sendPO"
    outputVariable="getResponse">
    <correlations>
      <correlation set="PurchaseOrder" initiate="yes" pattern="out"/>
    </correlations>
  </invoke>
</scope>
```



# Fault Handlers



# Exception Handling in BPEL

**<faultHandlers>** catch exceptions based on a fault name and fault variables  
Fault Handlers can perform arbitrary activities upon invocation

```
<process>
  ...
  <scope>
    <faultHandlers>
      <catch faultName="lns:cannotCompleteOrder"
            faultVariable="POFault">
        <reply partnerLink="customer"
              portType="lns:purchaseOrderPT"
              operation="sendPurchaseOrder"
              variable="POFault"
              faultName="cannotCompleteOrder"/>
      </catch>
      <catchAll>
        <empty/>
      </catchAll>
    </faultHandlers>
    ... other activities
  </scope>
</process>
```

```
<throw faultName="lns:cannotCompleteOrder" variable="POFault"/>
```





# Event Handlers

# Event Handlers in BPEL

`<eventHandlers>` are invoked concurrently when certain events occur

There are two types of events: **Message Events** and **Alarm Events**

**Message Events:** Event that waits for a message to arrive

```
<process name="orderCar">
  ...
  <eventHandlers>
    <onMessage partnerLink="buyer" portType="car" operation="cancel"
      variable="cancelDetails">
      <terminate/>
    </onMessage>
    ...
  </eventHandlers>
  ...
</process>
```

# Event Handlers in BPEL

## Alarm Events: Define timeout events

```
<process name="orderCar" xmlns:def="http://www.example.com/wsdl/example" ...>
  ...
  <eventHandlers>
    <onAlarm for=
      "bpws:getVariableData(orderDetails,processDuration)">
      ...
    </onAlarm>
    ...
  </eventHandlers>
  ...
  <variable name="orderDetails" messageType="def:orderDetails"/>
  </variable>
  ...
  <receive name="getOrder"
    partnerLink="buyer"
    portType="car"
    operation="order"
    variable="orderDetails"
    createInstance="yes"/>
  ...
</process>
```

# BPEL Lifecycle Management

## Creating a process instance

- » BPEL4WS business processes represent stateful long-running interactions
- » The creation of a process instance in BPEL4WS is always implicit (e.g. with first <invoke>)
- » Activities that receive messages (<receive> activities or <pick> activities) can be annotated to indicate that the occurrence of that activity causes a new instance (createInstance = „yes“)

## Terminating a process instance

- » When the **activity that defines the behavior of the process as a whole** (in most cases <sequence> or <flow>) **completes**.
- » When a **fault reaches the process scope**, and is either handled or not handled
- » When a process instance is **explicitly terminated** by a <terminate> activity.
- » If a **compensation handler** is specified for the business process as a whole, a business process instance can be compensated after normal completion.

# What is missing...

- » Details on BPEL4WS activities (there are a lot of them...)
- » Many examples that show how these activities „really“ work
- » Demo with Oracle BPEL Process Manager (former Collaxa)
- » The Future of BPEL + Conclusion
- » Brief Introduction in Business Process Monitoring with Senactive InTime

**→ Upcoming Week**

# Building Standard-Based Business Processes

with Web Services



Josef Schiefer

Vienna, November 2004

