

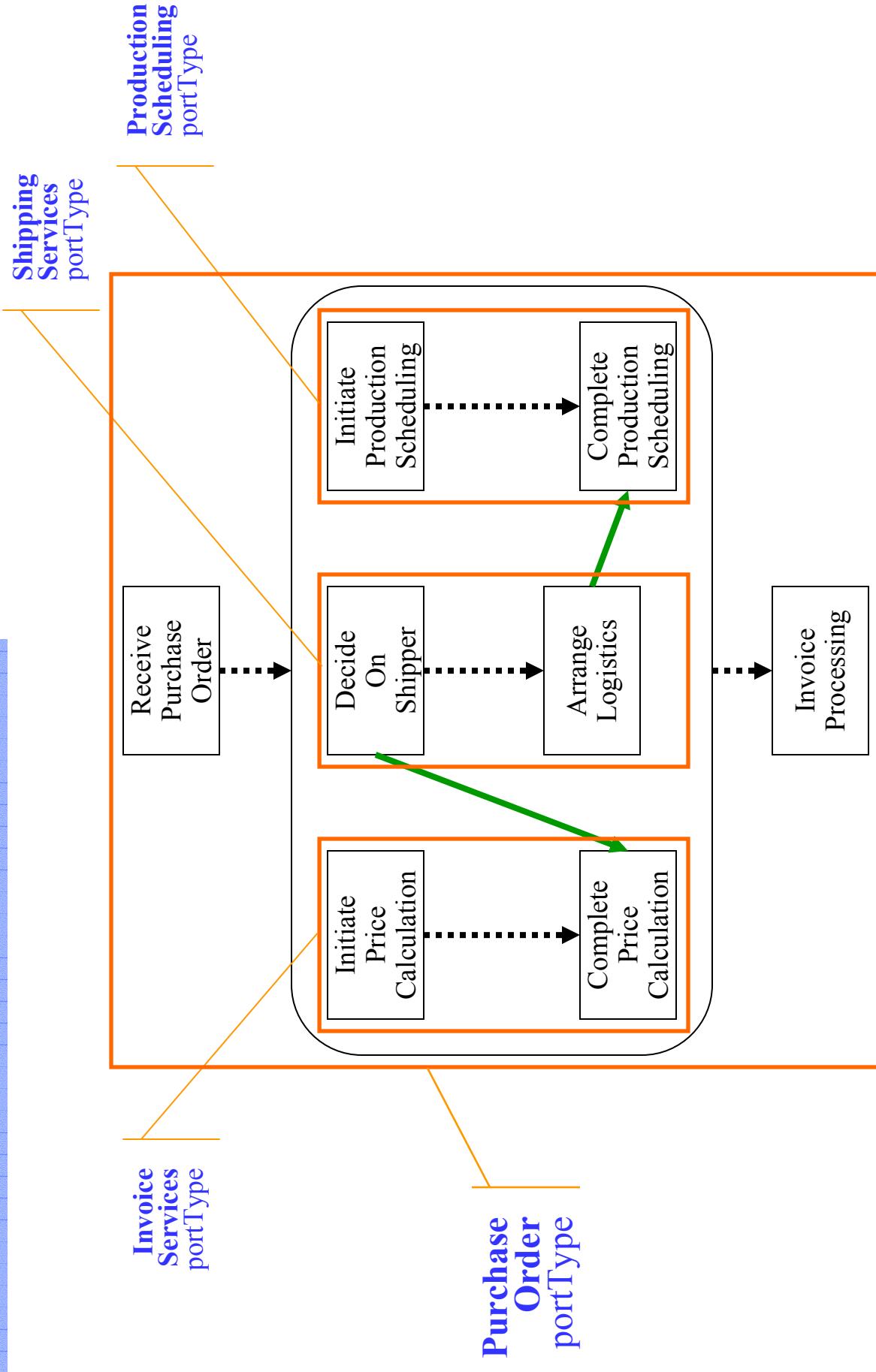
Business Process Execution Language



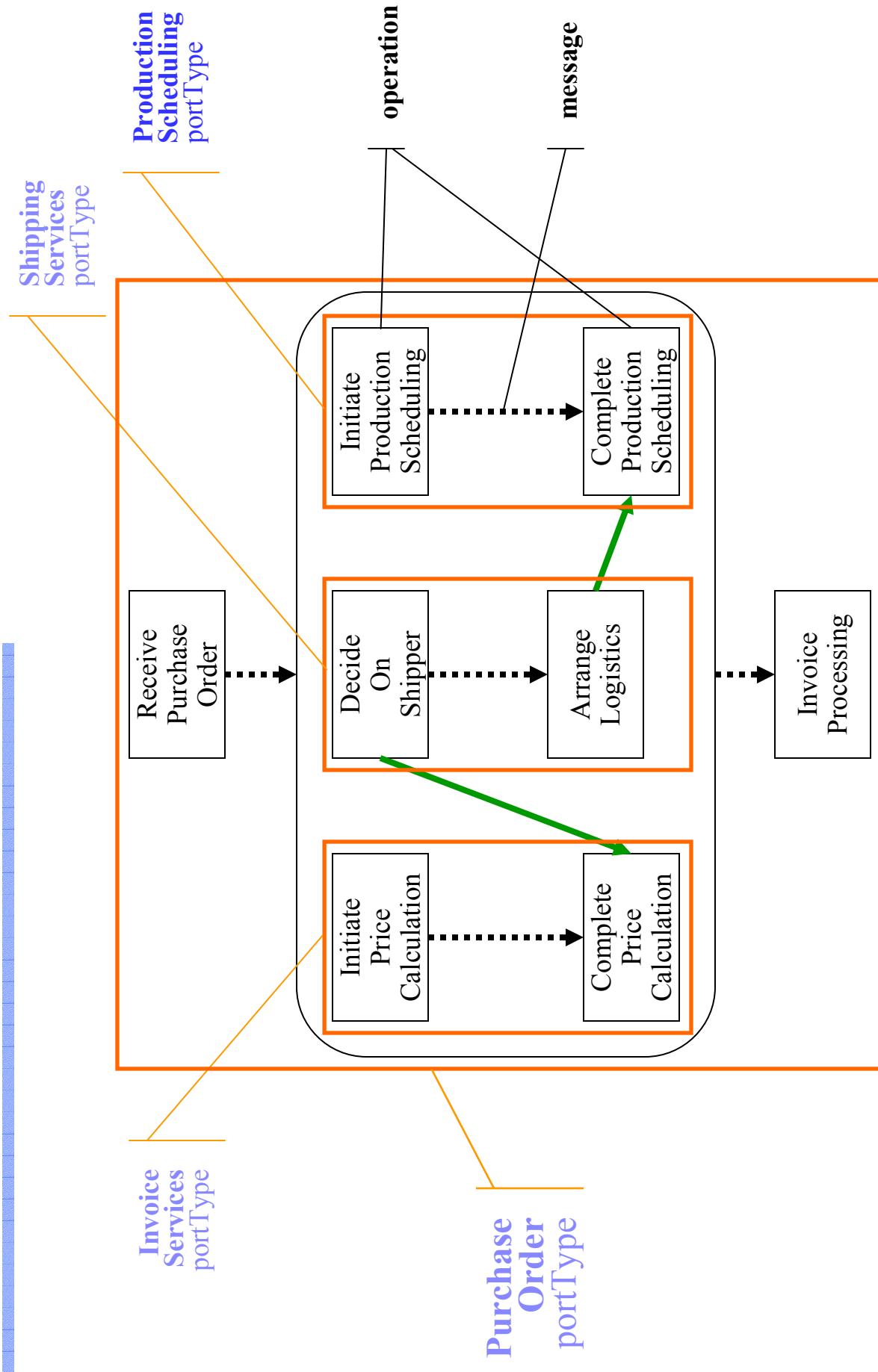
Business Process Execution Language

- Define business processes as coordinated sets of Web service interactions
- Define both abstract and executable processes
- Enable the creation of compositions of Web services
- Where it comes from:
 - Strong roots in traditional flow models
 - Concepts from structured programming languages
 - On top of WSDL and core XML specifications
 - Merges WSFL and XLANG concepts
- The OASIS WS BPEL Technical Committee is advancing the BPEL4WS Specification

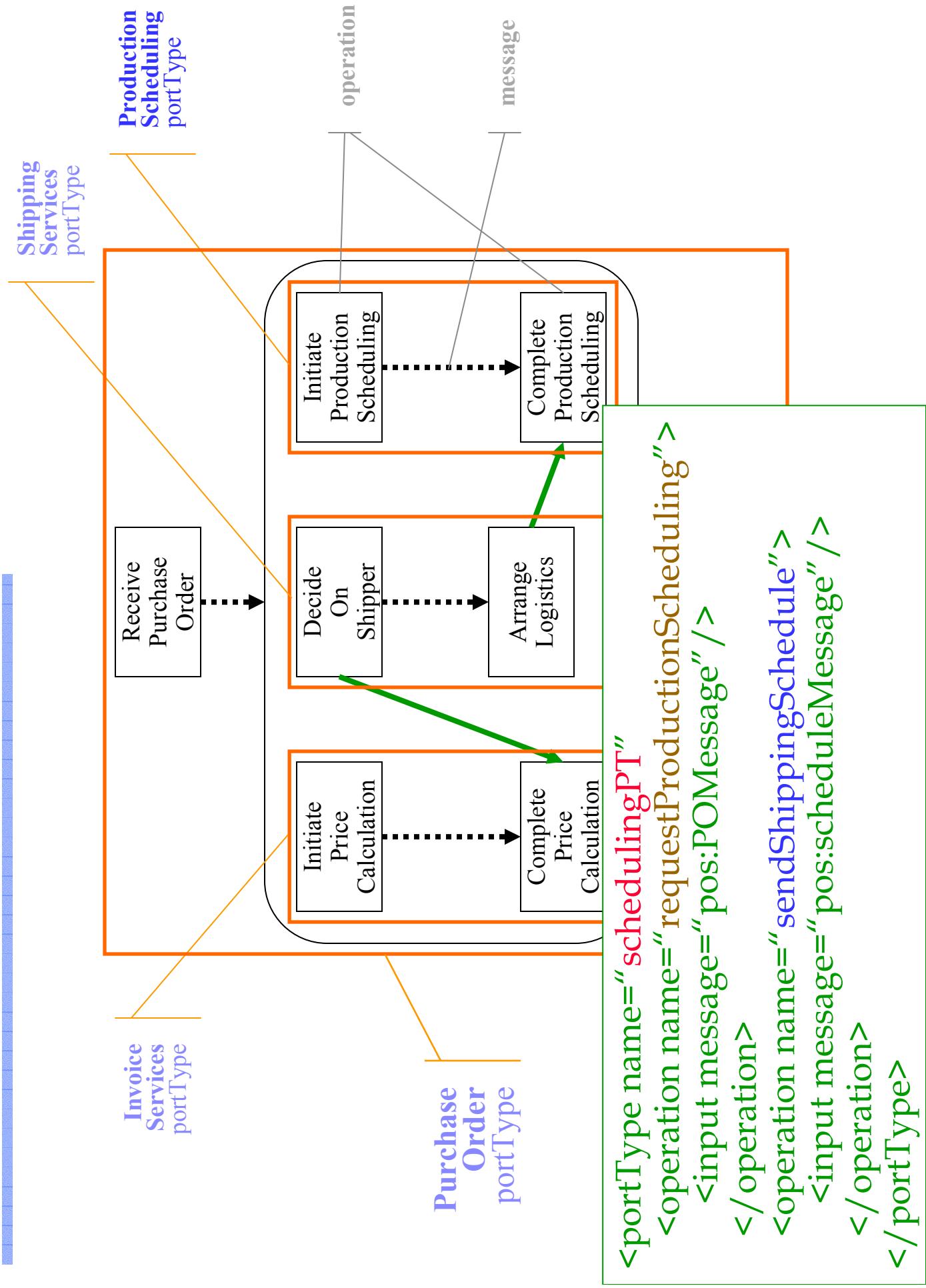
BPEL & WSDL



BPEL & WSDL



BPEL & WSDL

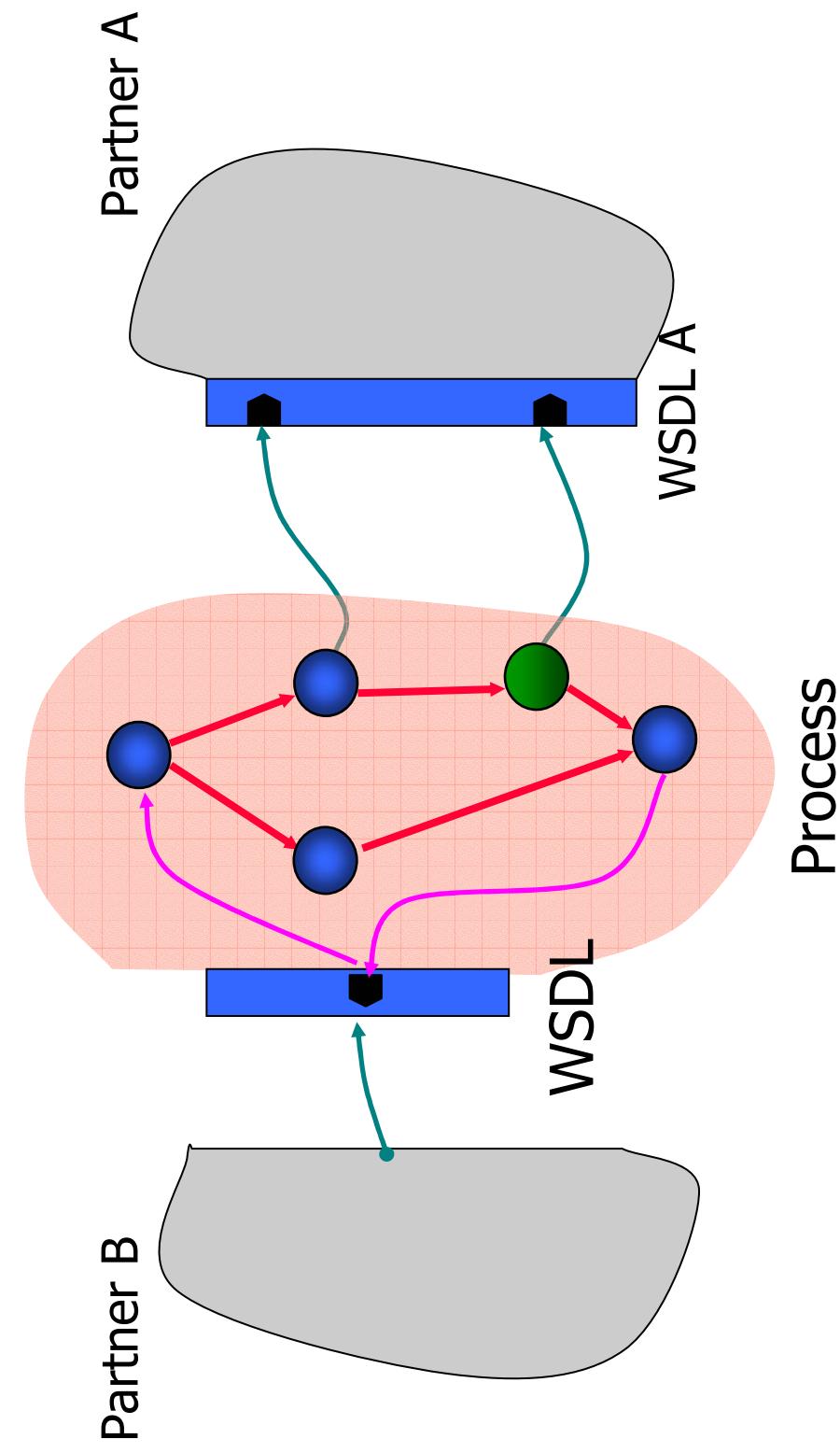


Structure of a BPEL Process

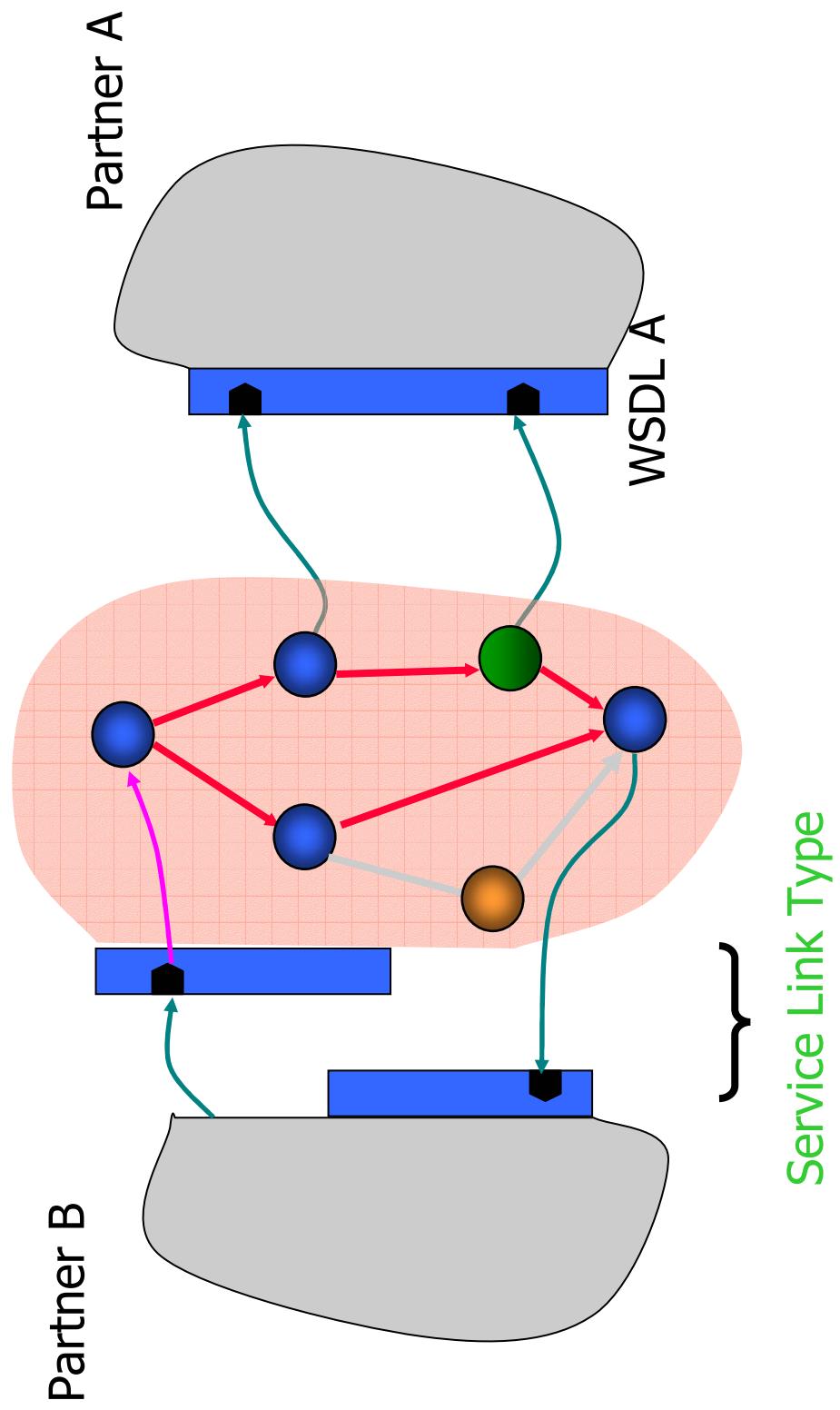
```
<process ...>
  <partners> ...
  </partners>
  <containers> ...
  </containers>
  <correlationSets> ...
  </correlationSets>
  <faultHandlers> ...
  </faultHandlers>
  <compensationHandlers> ...
  </compensationHandlers>
  ...
  (activities)*
</process>
```

- Web services the process interacts with
 - Data used by the process
 - Used to support asynchronous interactions
- Alternate execution path to deal with faulty conditions
 - Code to execute when “undoing” an action
- What the process actually does

BPEL and WSDL Partners



BPEL and WSDL Partners



Partner Links

- Partner links are used to represent interactions between a service and each of the parties with which it interacts
- Partner links define the **messages** and **port types** used in the interactions in both directions, along with role names

Partner Definitions and Links

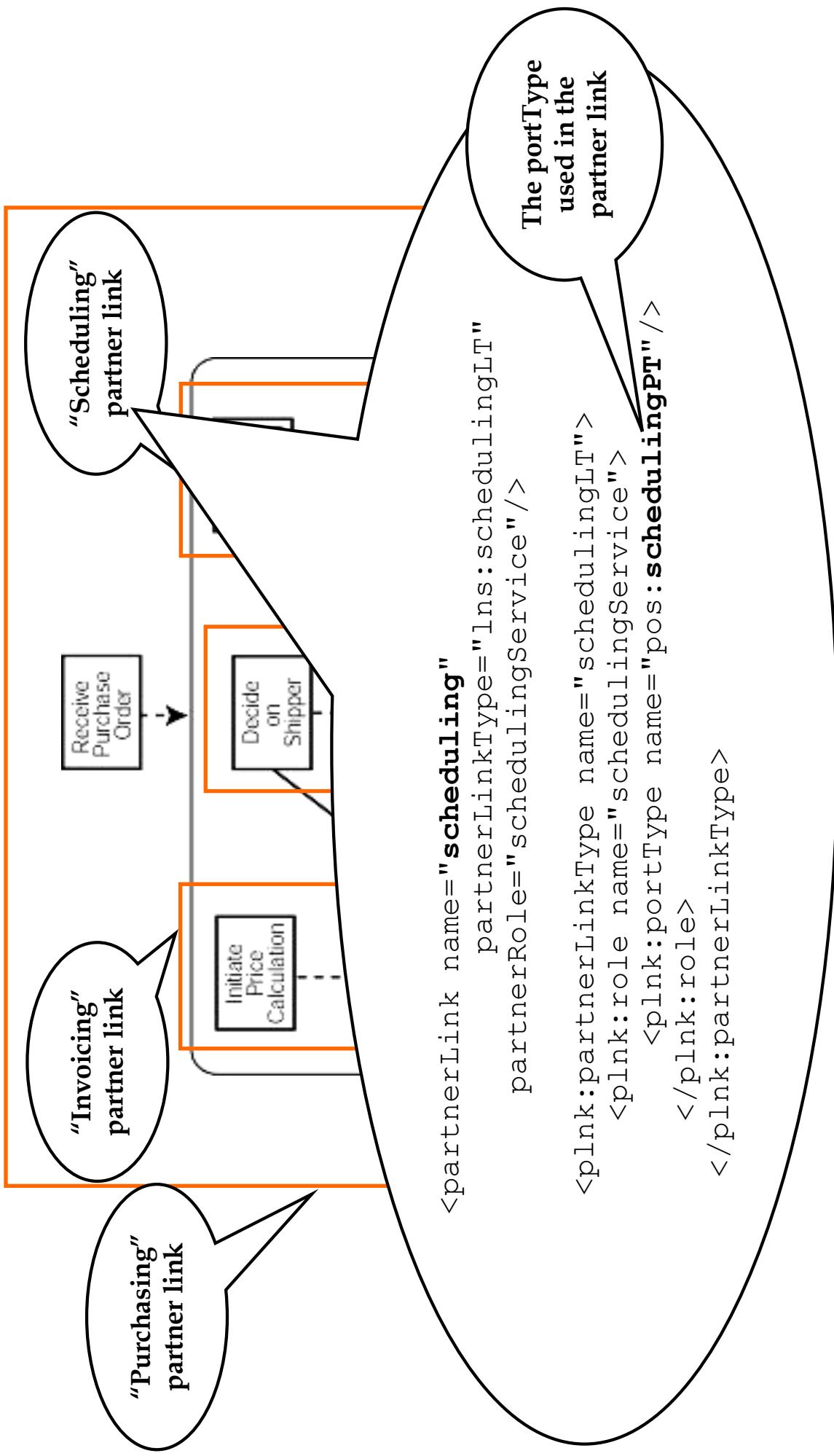
- A partner is accessed over a WS “channel”, defined by a service link type

```
<partner name="..." servicelinkType="..."  
partnerRole="..." myRole="..."/>
```

- A SLT defines two roles and the portTypes that each role needs to support

```
<serviceLinkType name="...">  
<role name="...">  
  <portType name="..." />*  
</role>  
<role name="...">  
  <portType name="..." />*  
</role>  
</serviceLinkType>
```

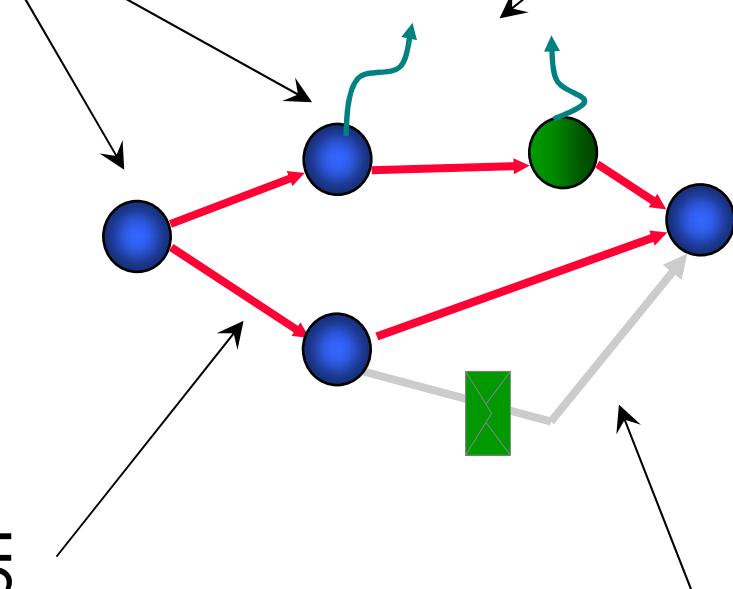
Partner Links



Traditional Flow Models

Control links define execution flow as a directed acyclic graph

Activities represent units of processing



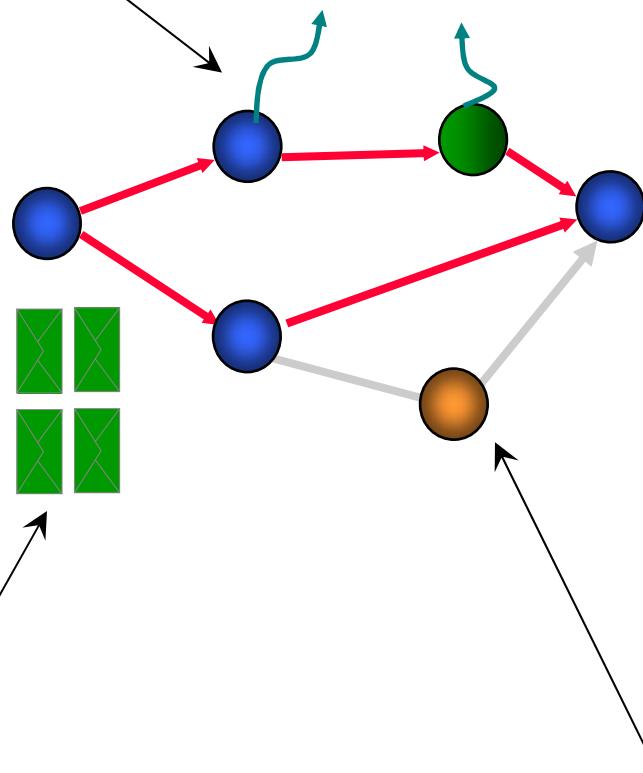
Flow of data is explicitly modeled through data links

Activities are mapped to application invocations or human actions

BPEL Data Model

Globally scoped data
variables typed as WSDL
messages

Activities input /
output is kept in
global variables



Assignment
activities move
data around

<container name="..." message="..." />

BPEL Basic Activities

- Invokes an operation on a partner

```
<invoke partner="..." portType="..." operation="..."  
inputContainer="..." outputContainer="..." />
```

- Receives invocation from a partner

```
<receive partner="..." portType="..." operation="..."  
container="..." [createInstance="..."] />
```

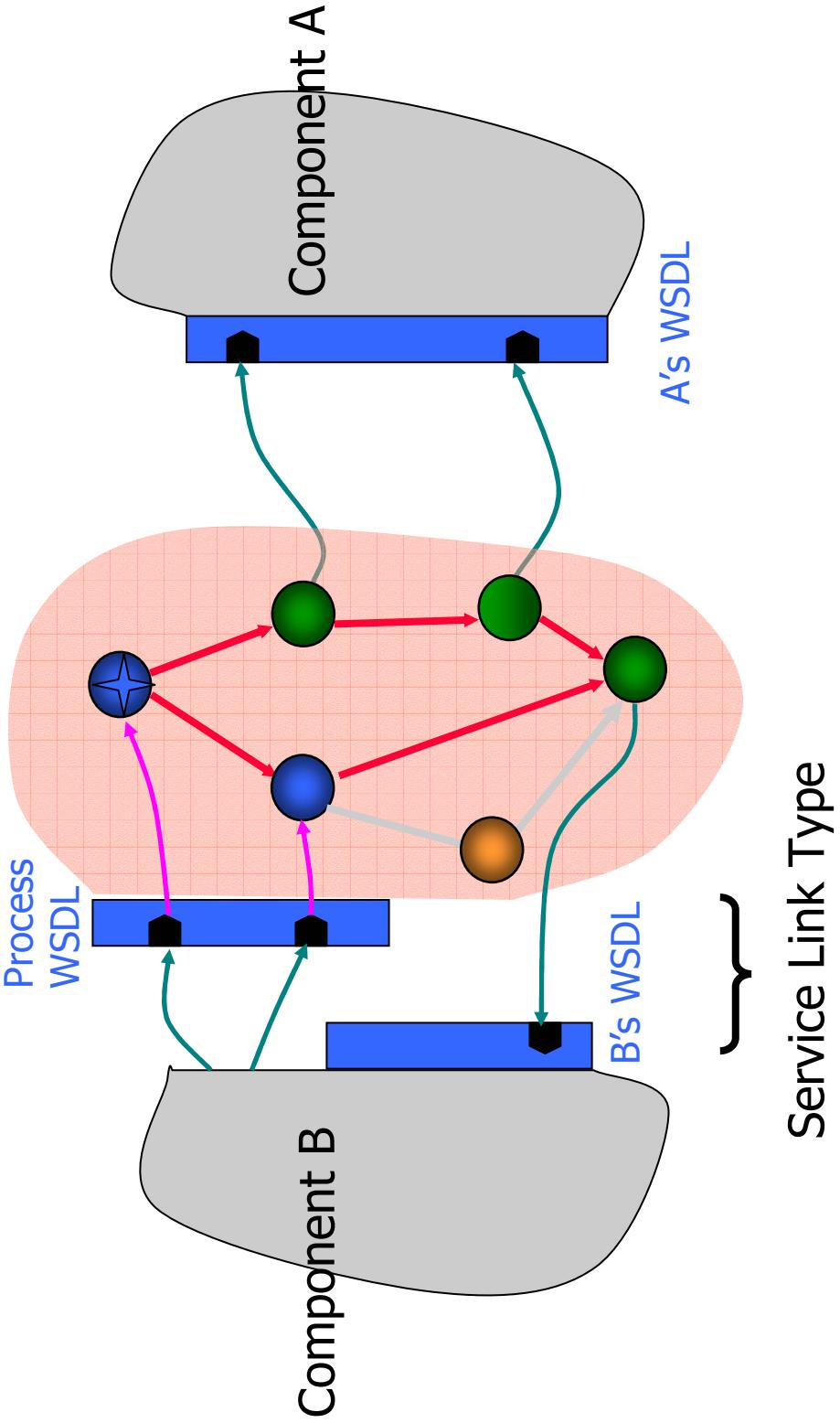
- Sends a reply message in partner invocation

```
<reply partner="..." portType="..." operation="..."  
container="..." />
```

- Data assignment between containers

```
<assign>  
<copy>  
<from container="..." /> <to container="..." />  
</copy>+  
</assign>
```

BPEL Composition of Web Services



More Basic Activities

- Detects processing error and switches into fault processing mode

```
<throw faultName="..." faultContainer="..." />
```

- Pull the plug on this instance

```
<terminate/>
```

- Execution stops for a specified amount of time

```
<wait for="..." until="..." ? />
```

- Do nothing; a convenience element

```
<empty>
```

BPEL Structured Activities



<sequence>
execute activities sequentially



<flow>
execute activities in parallel



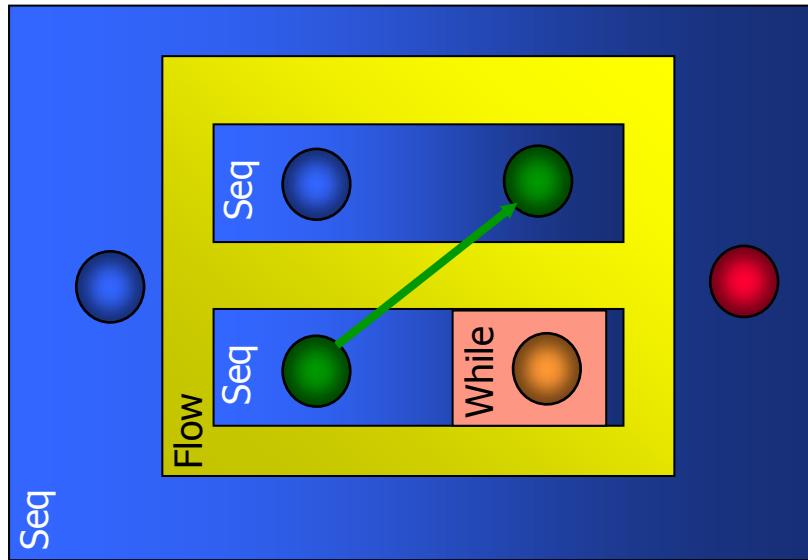
<while>
iterate execution of activities until condition is violated
<pick>

several event activities (receive message, timer event)
scheduled for execution in parallel; first one is selected
and corresponding code executed

<link ...>
defines a control dependency between
a source activity and a target



Nesting Structured Activities: Example



```
<sequence>
  <receive .../>
  <flow>
    <sequence>
      <invoke ... />
      <while ... >
        <assign> ... </assign>
        </while>
    </sequence>
    <sequence>
      <receive ... />
      <invoke ... />
    </sequence>
    </flow>
    <reply ... />
  </sequence>
```

Asynchronous Interactions in BPEL

- BPEL can model many types of interactions:
 - Simple stateless interactions
 - Stateful, long running, asynchronous interactions
- For the latter case, how to ensure that two (or more) messages are referring to the same “session”?

Message Correlation

- Associating two or more messages with each other in an asynchronous environment
- 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**

Purchase Order:

```
<PurchaseOrder>
  <PurchaseOrderNumber>.....</PurchaseOrderNumber>
  <PurchaseOrderDate>.....</PurchaseOrderDate>
  .....
</PurchaseOrder>
```

Invoice:

```
<Invoice>
  <InvoiceNumber>
  <InvoiceDate>
  <PurchaseOrderNumber>.....</PurchaseOrderNumber>
  .....
</Invoice>
```

common in both messages

Correlation Sets

- What is a correlation set?
 - A set of business data fields that capture the state of the interaction ("correlating business data"), e.g., a "purchase order number", a "customer id", etc.
 - Each set is initialized once
 - Its values do not change in the course of the interaction
- CSS : the data used to maintain the state of the interaction (a "conversation")
 - At the process end of the interaction, CSS allow incoming messages to reach the right process instance

Defining Correlation Sets

- A **CS** is a named set of properties. Properties are defined as WSDL extensibility elements

```
<correlationSet name="..." properties="..." />
```

- A **property** has a simple XSD type and a global name

```
<bpws:property name="..." type="..." />
```

Properties

- A property is “mapped” to a field in a WSDL message type

```
<bpws:property Alias  
    propertyName="..."  
    messageType="..." part="..."  
    query="..."/>
```

- The property can thus be found in the messages actually exchanged
- Typically a property will be mapped to several different message types and carried on many interactions, across operations and portTypes

Using Correlation

- An input or output operation identifies which correlation sets apply to the messages received or sent

```
<receive partner="..." operation="..." portType="..."  
container="...">  
<correlations>  
  <correlation set="PurchaseOrder" initiation="yes" />  
</correlations>  
</receive>
```

- That CS will be used to assure that the message is related to the appropriate stateful interaction
- A CS is initialized once, in an interaction where the set appears with the “initiation” attribute set to “yes”. Its value may never be changed afterward

Example: Defining Correlation Sets

- A customer ID and order number represent a unique purchase order

```
<correlationSet name="PurchaseOrder"
```

```
    properties=" cor:customerID cor:orderNumber" />
```

```
<correlationSet name="Invoice"
```

```
    properties=" cor:vendorID cor:invoiceNumber" />
```

- A vendor ID and invoice number represent a unique invoice

Example: Using Correlation Sets

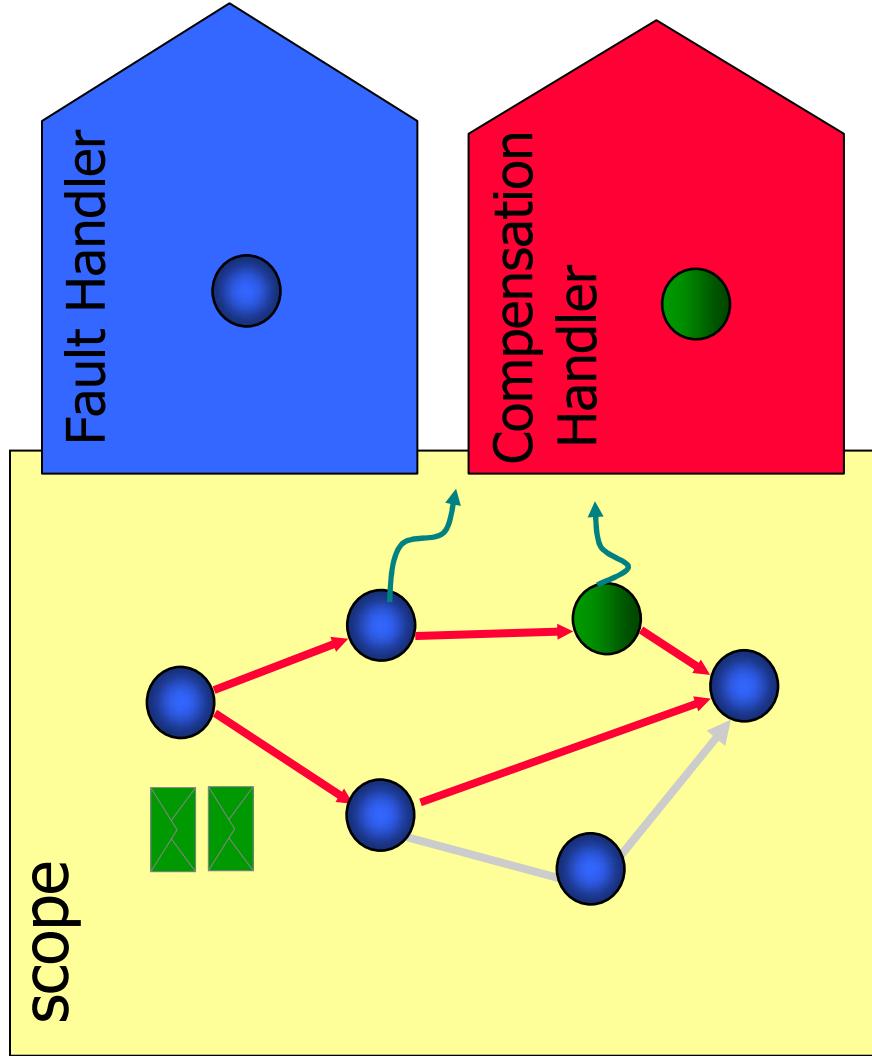
- Declares correlation between purchase order and invoice

```
<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>
```

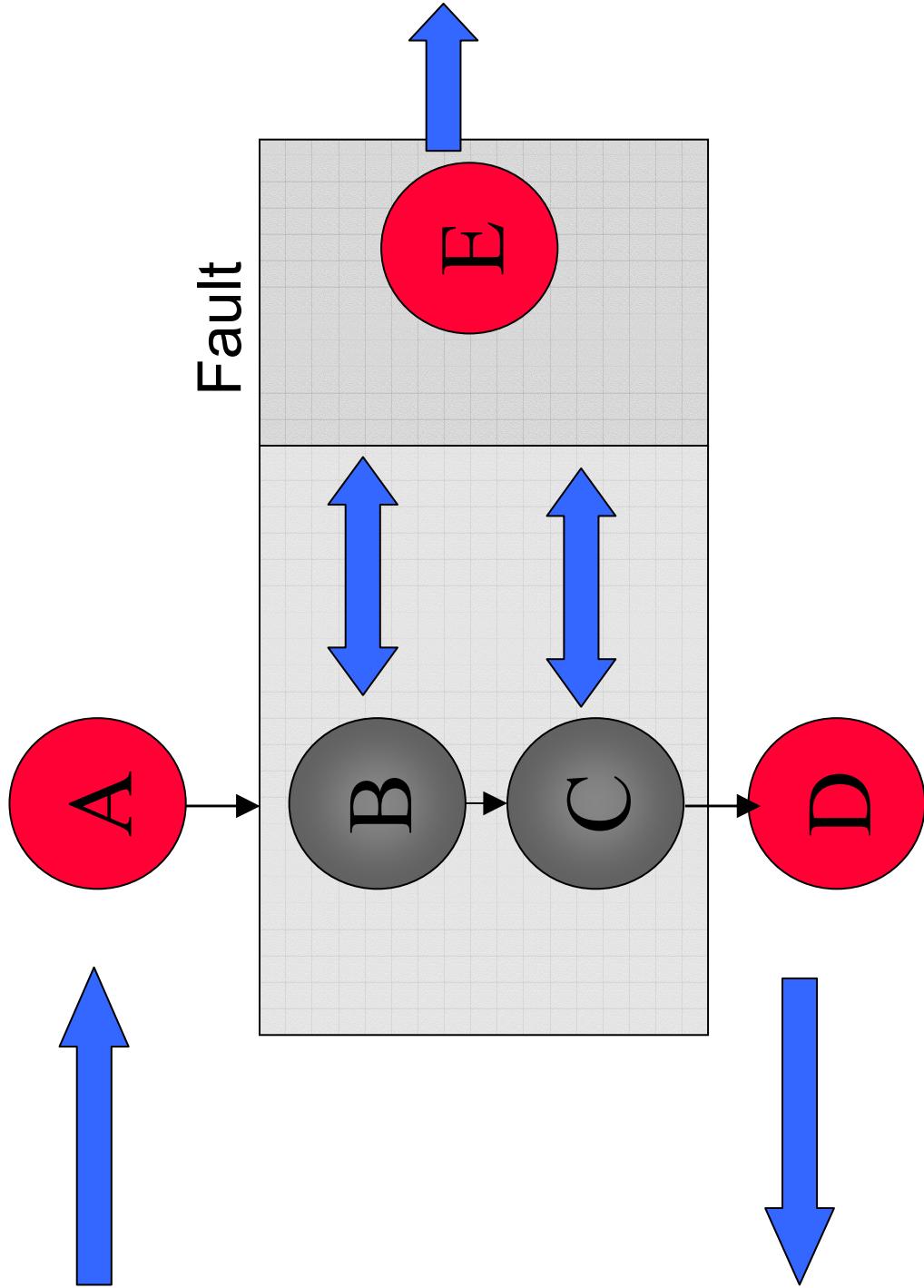
BPEL Handlers and Scopes

- A **scope** is a set of (basic or structured) activities

- Each scope can have two types of handlers associated:
 - **Fault handlers**
 - Many can be attached, for different fault types
- **Compensation handler**
- A single compensation handler per scope



Scope and Fault Example



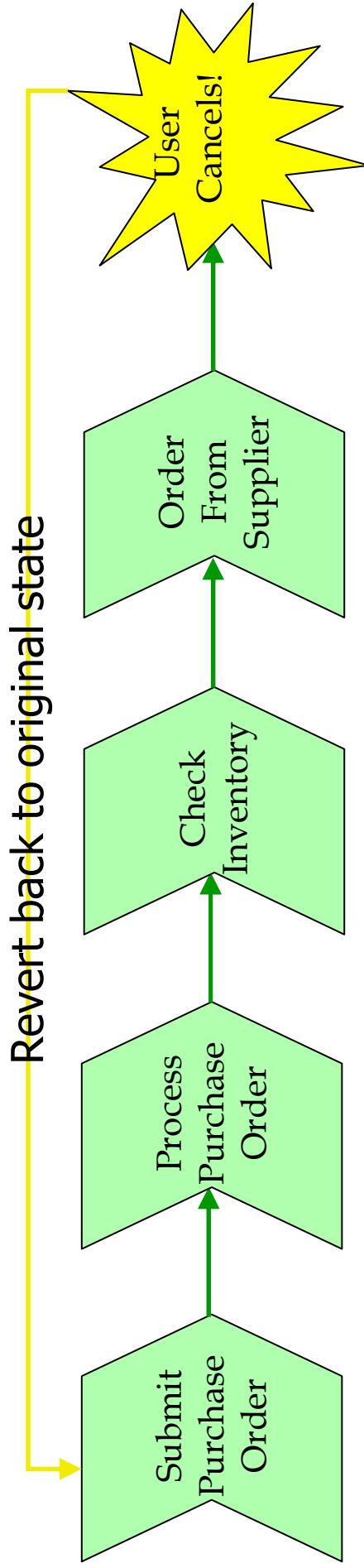
How Handlers Work

- A fault handler defines alternate execution paths when a fault occurs within the scope
- Typical scenario:
 1. Fault is thrown (retuned by invoke or explicitly by process)
 2. Execution of scope is terminated
 3. Appropriate fault handler located (with usual propagation semantics)
 4. Main execution is compensated to “undo” business effects of unfinished work
- A compensation handler is used to reverse the work performed by an already **completed** scope
 - A compensation handler can only be invoked by the fault handler or compensation handler of its immediate enclosing scope

Partial Work Unavoidable

- Business processes are often of long duration, which means that a business process may need to be cancelled after many transactions have been committed during its progress

- 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

- Invoked to perform compensation activities — a “wrapper” for compensation activities:
 - Specifies a compensating operation on a given portType for a given partner link:

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

The `CancelPurchase` operation invokes a cancellation

The response to the purchase request is used as input

Dynamic Service Selection and Invocation

- 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
- BPEL uses **endpoint references** defined in the WS-Addressing specification for this capability

<http://msdn.microsoft.com/ws/2003/03/ws-addressing>

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

PortType and **Address** association