BPEL: Building Standards-Based Business Processes with Web Services

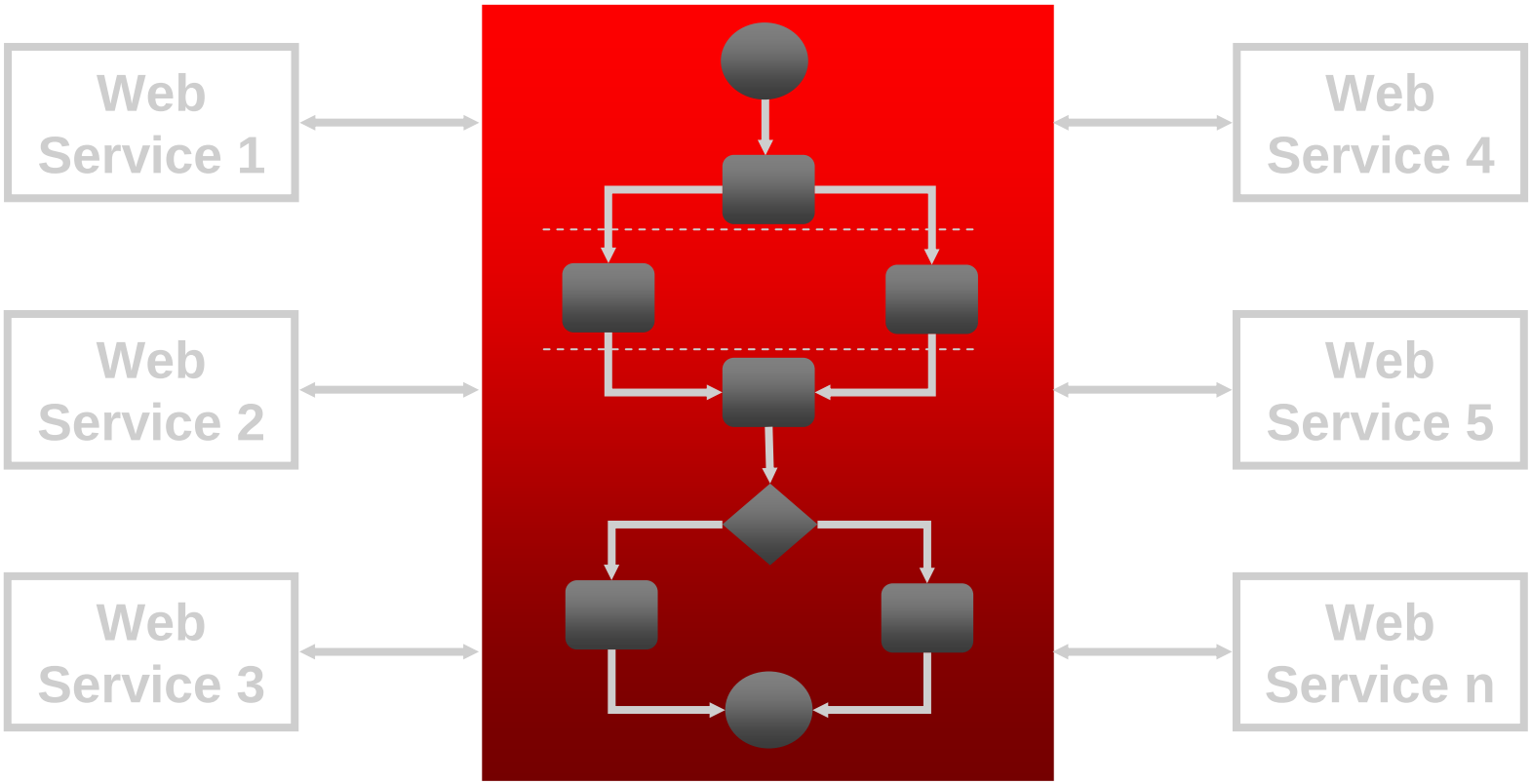
BPEL?

“B E E P L E”?

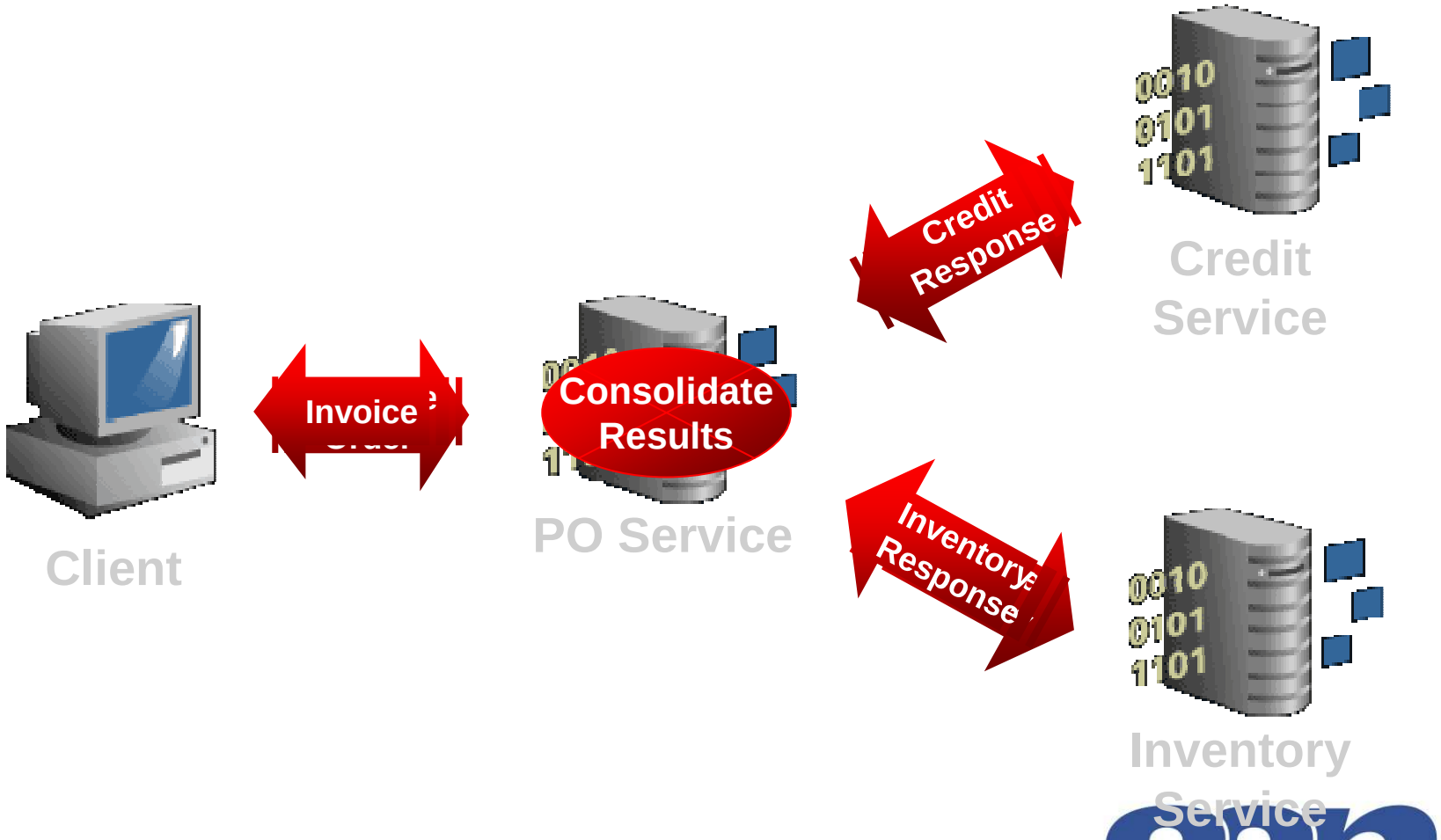
“B E E – P E L L”?

“B I P P L E”?

Web Services Meet Business Processes



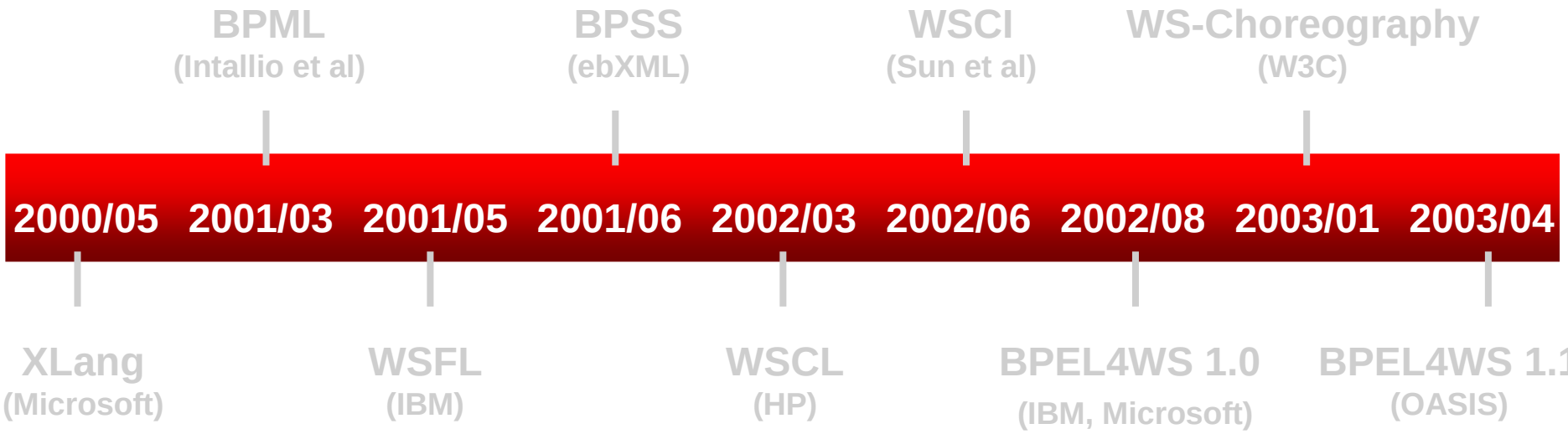
Example Problem Space



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



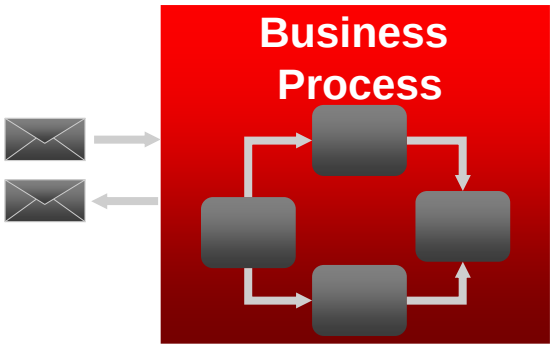
Orchestration vs. Choreography

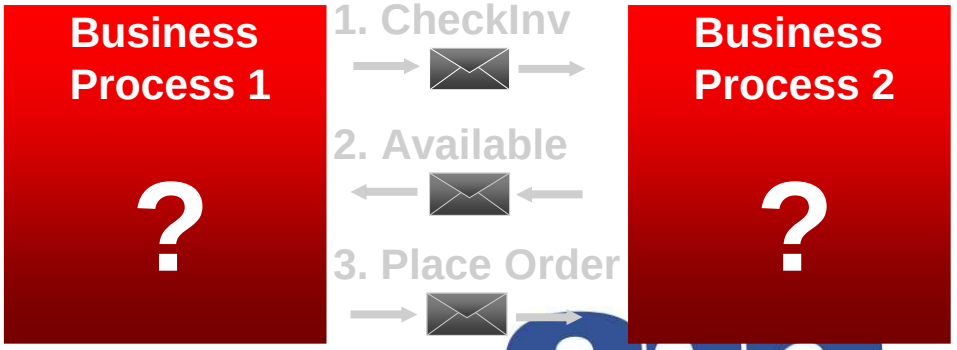
Orchestration

- Private process
- Steps of an executable workflow
- Process controlled by one party

Choreography

- Public (abstract) process
- Sequence of observable messages
- Conversation made up of equals





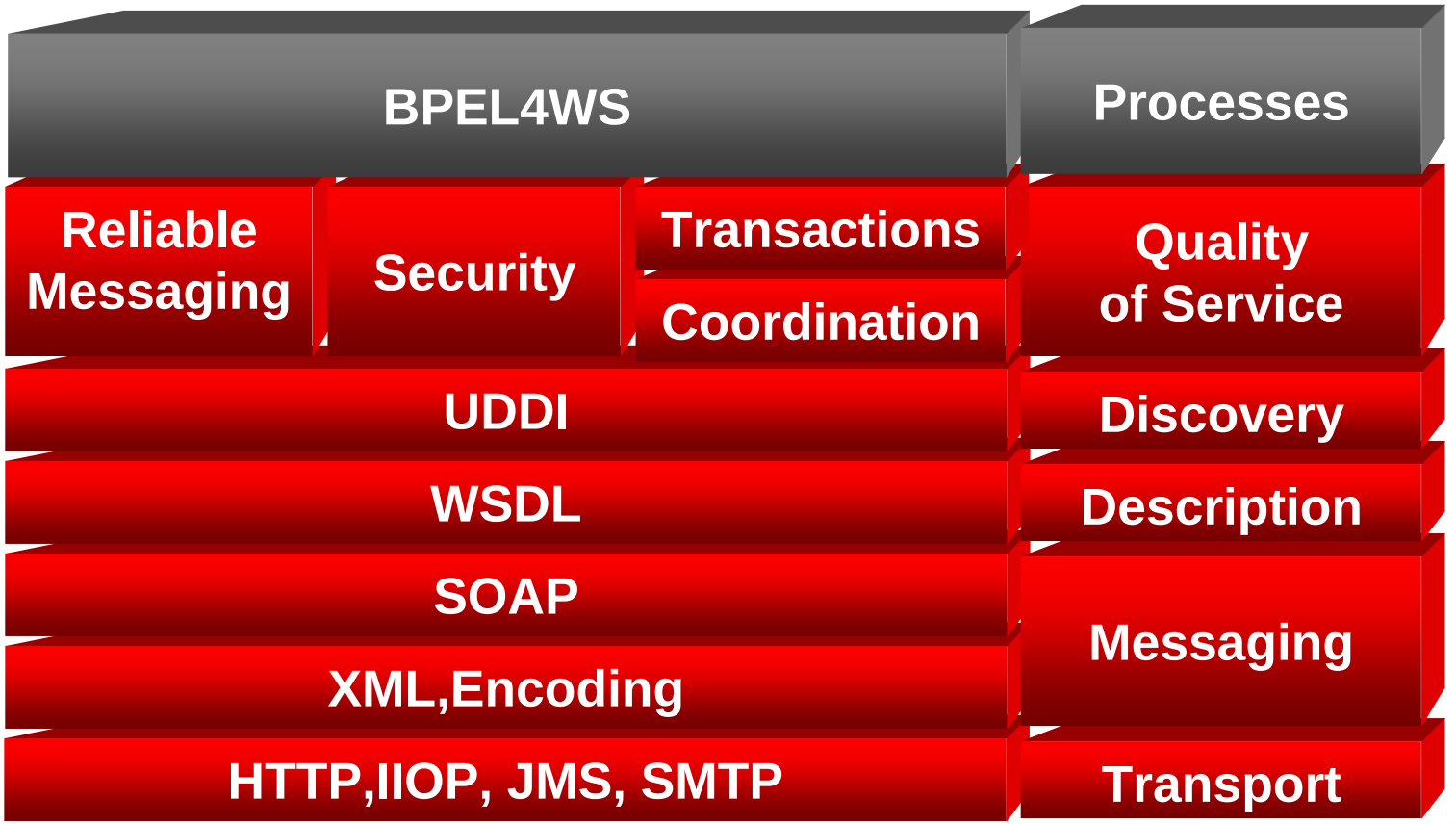
Business Process Execution Language for Web Services

- Version 1.0 released by IBM, Microsoft and BEA in August 2002
  - Accompanied by WS-Coordination, WS-Transaction which remain unsubmitted to standards bodies
- Version 1.1 submitted to OASIS April 2003
- XML language for describing business processes based on Web services
  - Convergence of XLANG (Microsoft) and WSFL (IBM)
- Amazing industry “consensus” in the last 6 months
  - IBM, Microsoft, Oracle, Sun, BEA, SAP, Siebel …

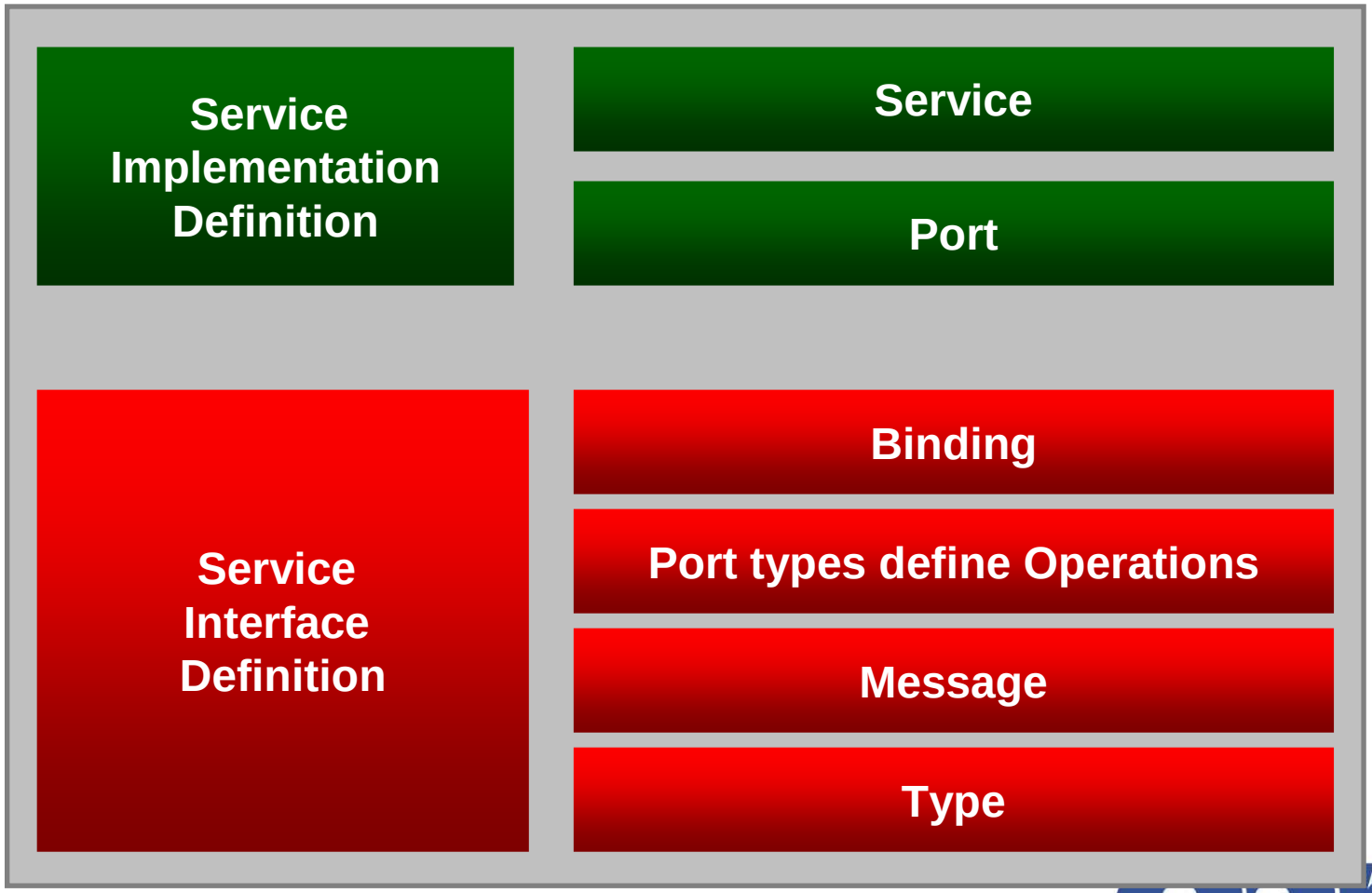
Value Proposition

- 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

Standards Building Blocks of BPEL



BPEL Depends on WSDL and WSDL Extensions



BPEL Scenario Structure

```
<process>
   <!– Definition and roles of process participants -->
   <partners> ... </partners>
   <!- 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 Activities

Primitive Activities

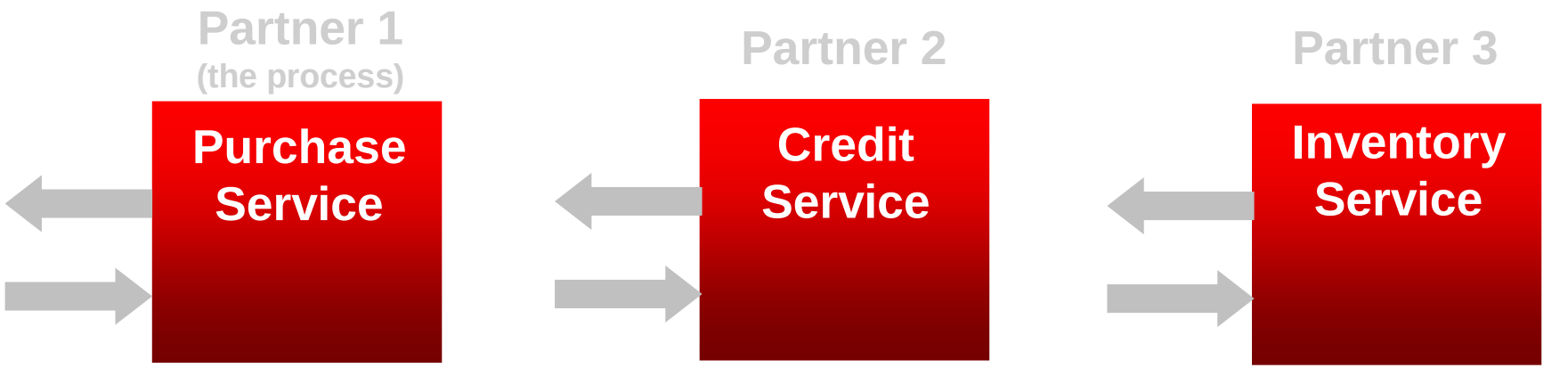
- `<invoke>`
- `<receive>`
- `<assign>`
- `<reply>`
- `<throw>`
- `<terminate>`
- `<wait>`

Structured Activities

- `<sequence>`
- `<switch>`
- `<pick>`
- `<flow>`
- `<link>`
- `<while>`
- `<scope>`

Partners

- Declare the Web services and roles used by the process
- Tied to WSDL of the process itself and the participating Web services by service link types



Partners in BPEL

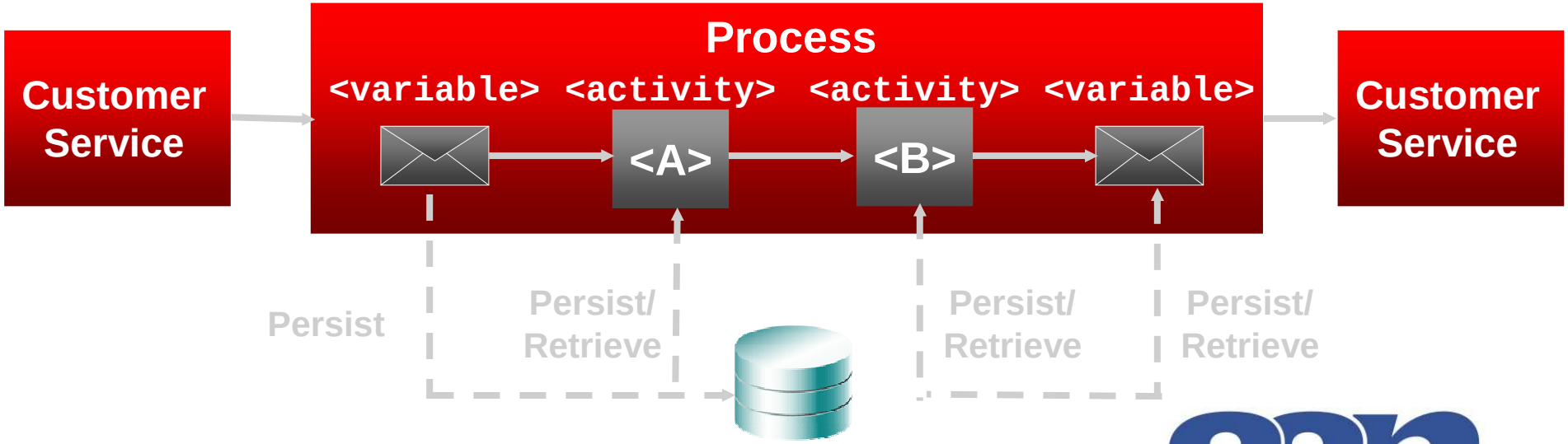
```
<partners>
 <partner name="customer" serviceLinkType="lns:purchaseSLT"
    myRole="purchaseService"/>
 <partner name="inventoryChecker" serviceLinkType="lns:inventorySLT"
    myRole="inventoryRequestor" partnerRole="inventoryService"/>
 <partner name="creditChecker" serviceLinkType="lns:creditSLT"
    myRole="creditRequestor" partnerRole="creditService"/>
</partners>
```

```
<slt:serviceLinkType name="purchaseSLT">
 <slt:role name="purchaseService">
  <slt:portType name="tns:purchasePT"/>
 </slt:role>
</slt:serviceLinkType>
```

```
<portType name="purchasePT">
 <operation name="sendPurchase">
 </operation>
</portType>
```

Variables

- Messages sent and received from partners
  - Persisted for long running interactions
  - Defined in WSDL types and messages



Variables in BPEL

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

```
<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?

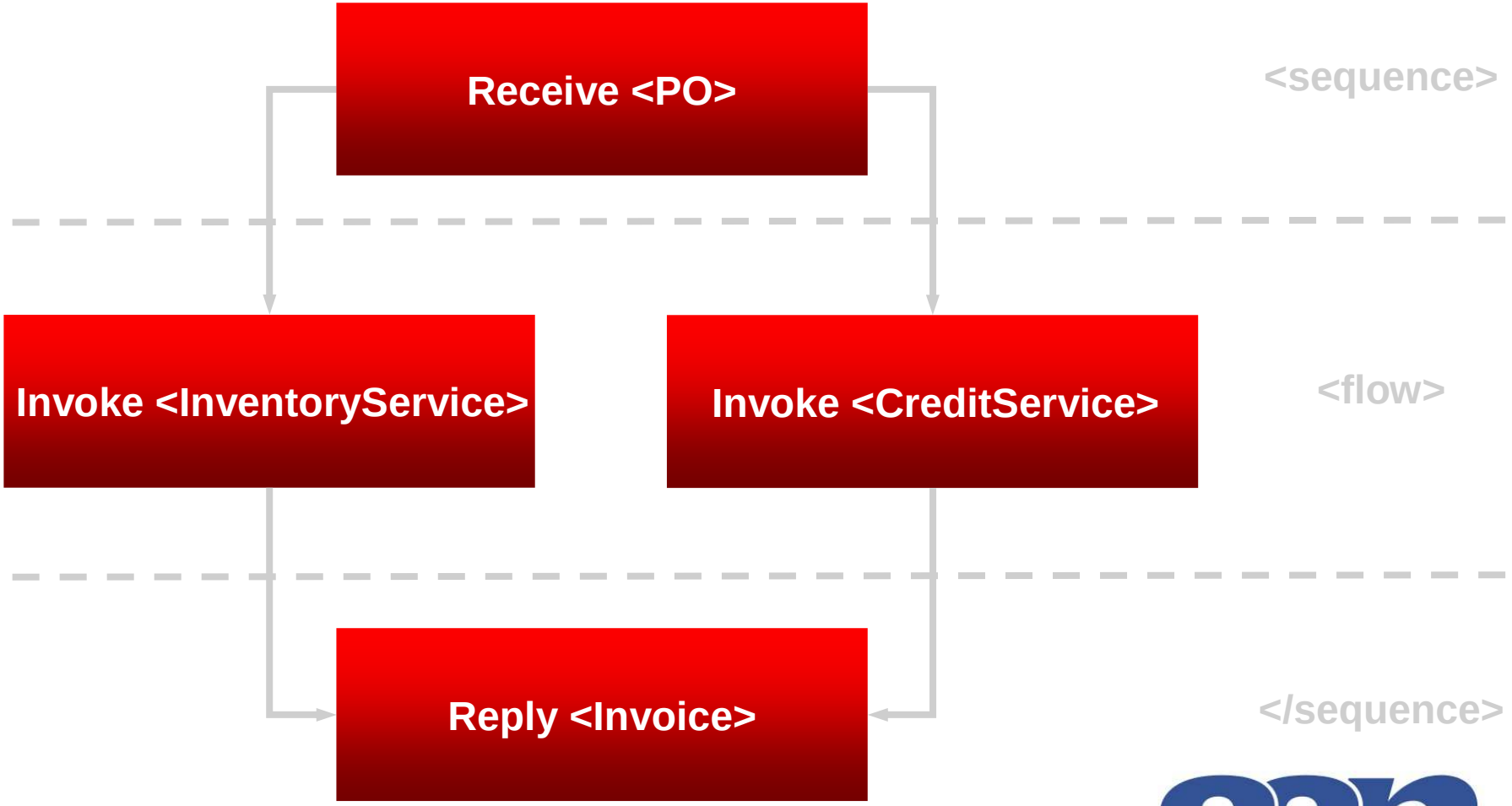
- Using <assign> and <copy>, data can be copied and manipulated between variables
- <copy> supports XPath queries to sub-select data

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

Simple Activities

- Receive
  - Wait for a partner inbound message
  - Can be the instantiator of the business process
- Reply
  - Synchronous response to a receive activity
  - Response to the inbound receive from a partner
- Invoke
  - Issue a request synchronously *or* asynchronously
- Pick
  - Specify an inbound set of messages
  - Can be the instantiator of the business process
  - Activity completes when one of the messages arrives

Simple Activities Combined with Structured Activities

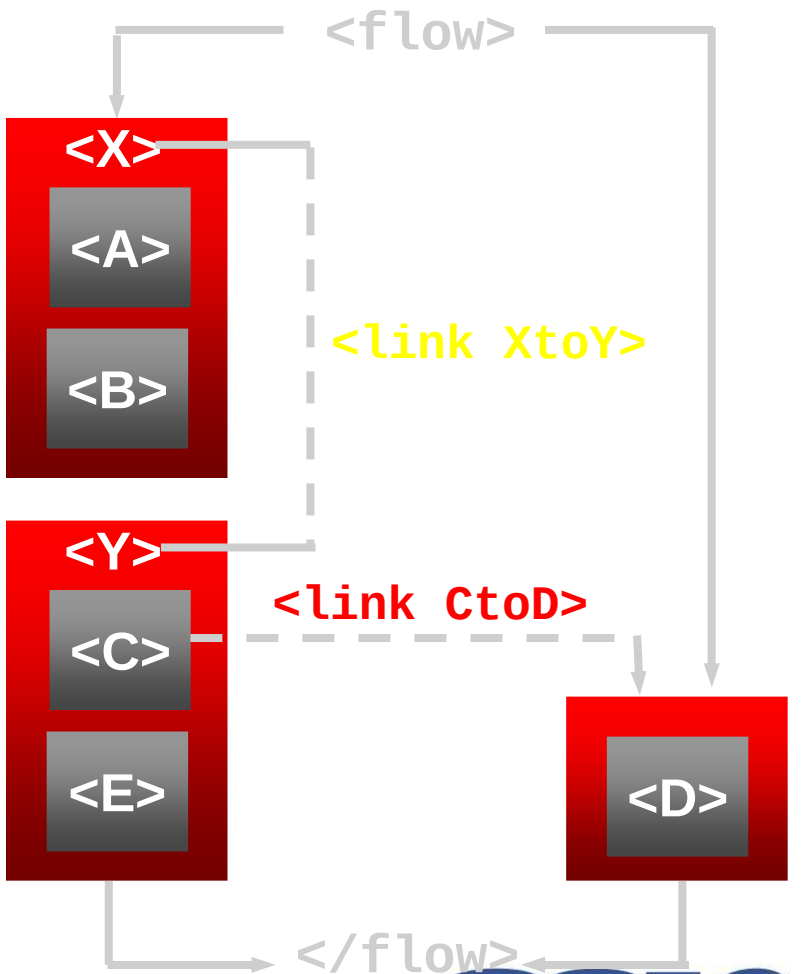


Sample Activities in BPEL

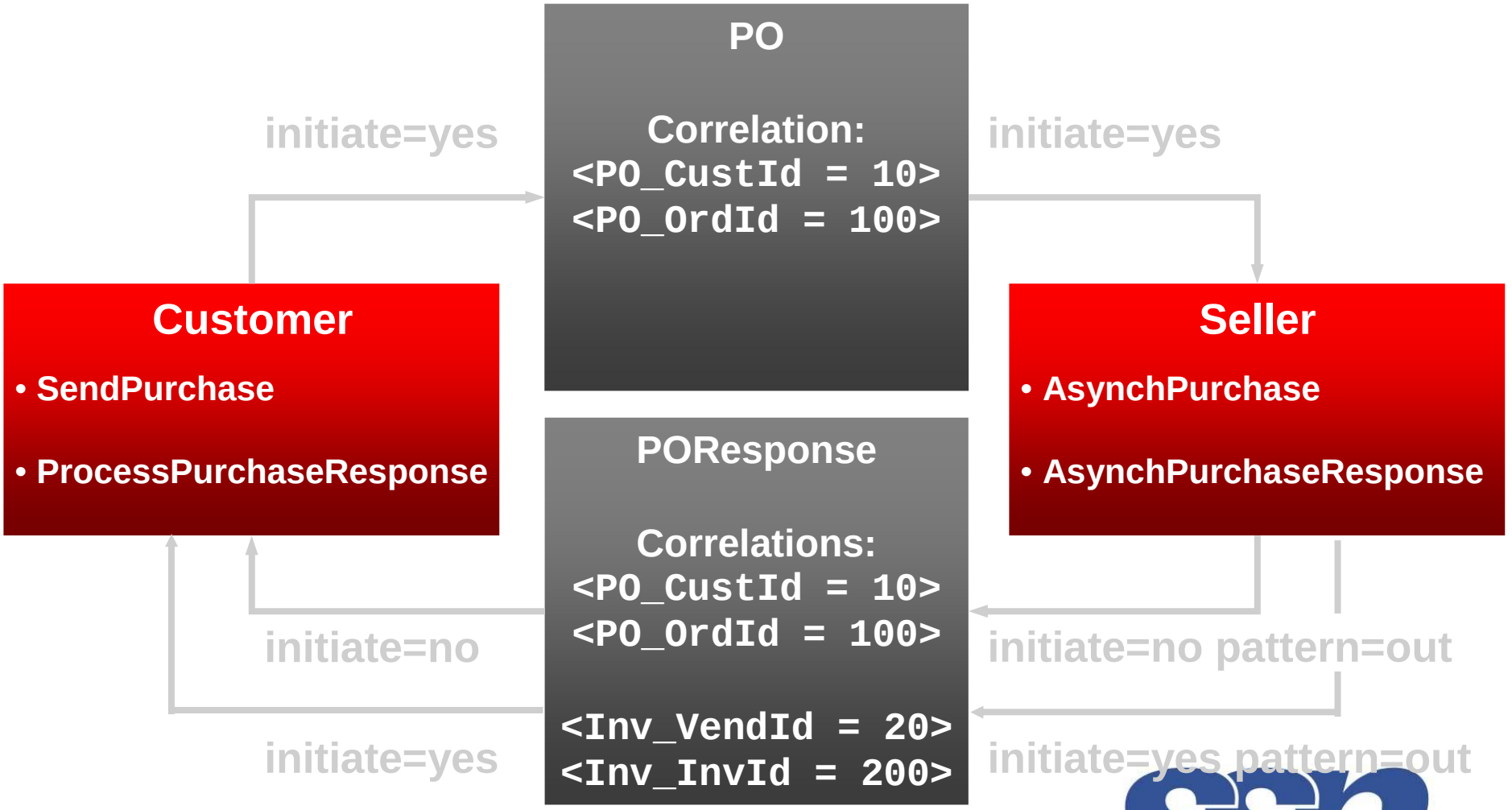
```
<sequence>
 <receive partner="customer" portType="lns:purchaseOrderPT"
          operation="sendPurchaseOrder" variable="PO"
          createInstance="yes" />
 <flow>
  <invoke partner="inventoryChecker" portType="lns:inventoryPT"
          operation="checkINV" inputVariable="inventoryRequest"
          outputVariable="inventoryResponse" />

  <invoke partner="creditChecker" portType="lns:creditPT"
          operation="checkCRED" inputVariable="creditRequest"
          outputVariable="creditResponse" />
 </flow>
 ...
 <reply partner="customer" portType="lns:purchaseOrderPT"
        operation="sendPurchaseOrder" variable="invoice"/>
</sequence>
```

Links – Control Flow

```
<flow>
 <links>
  <link name="XtoY"/>
  <link name="CtoD"/>
 </links>
 <sequence name="X">
   <source linkName="XtoY"/>
   <invoke name="A" .../>
   <invoke name="B" .../>
 </sequence>
 <sequence name"Y">
   <target linkName="XtoY"/>
   <receive name="C"/>
     <source linkName="CtoD"/>
   </receive>
   <invoke name="E" .../>
 </sequence>
 <invoke partner="D">
 <target linkName="CtoD"/>
 </invoke>
</flow>
```



Correlation



Correlations in BPEL

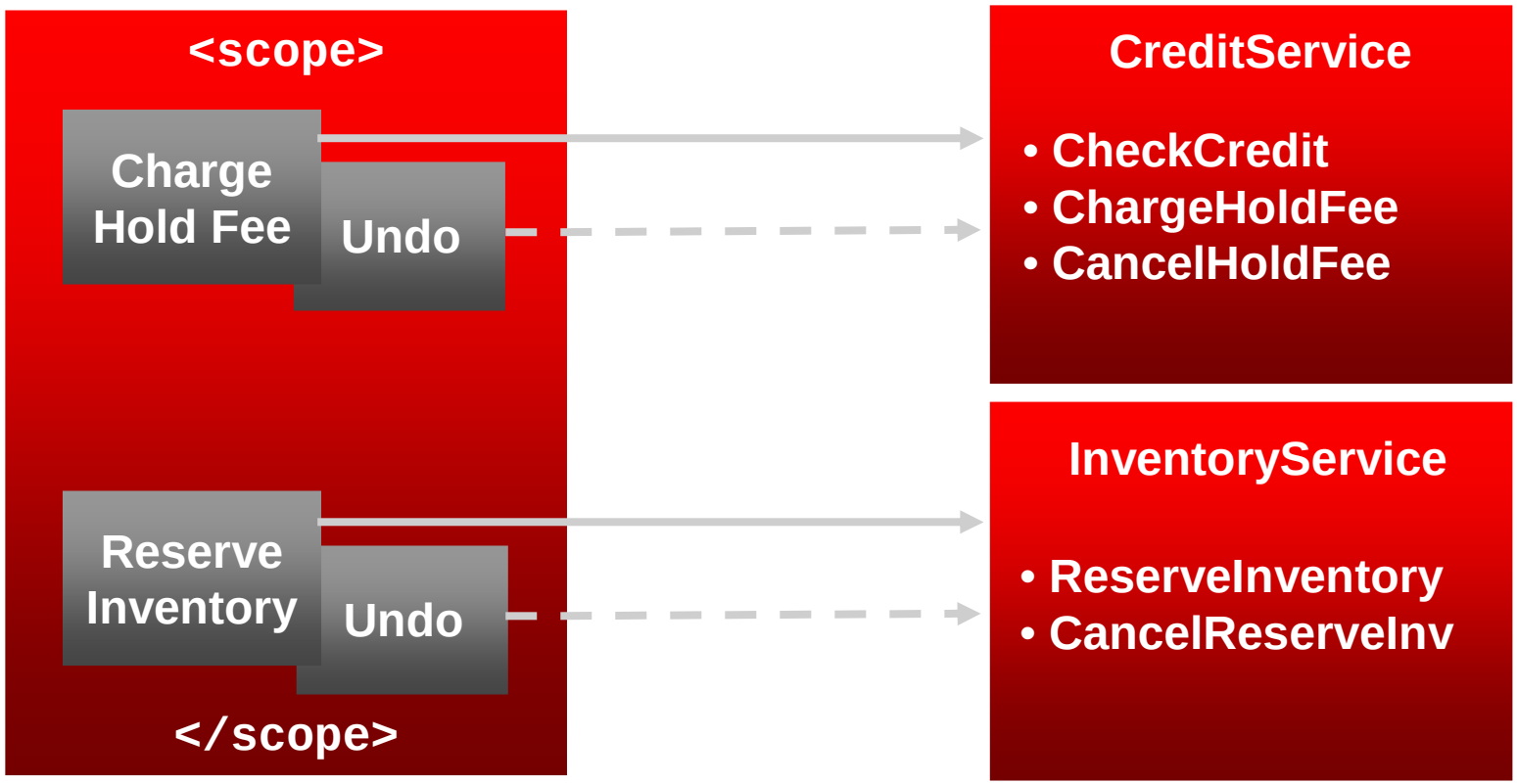
```
<correlationSets>
 <correlationSet name="POCorr" properties="cor:custId cor:ordId"/>
 <correlationSet name="InvoiceCorr" properties="cor:vendId cor:invId"/>
</correlationSets> ...

<receive partner="Customer" portType="SP:PurchaseOrderPT"
    operation="AsynchPurchase" variable="PO">
  <correlations>
    <correlation set="POCorr" initiate="yes">
  </correlations>
</receive> ...

<invoke partner="Customer" portType="SP:CustomerPT"
  operation="ProcessPurchaseResponse" inputVariable="POResponse">
 <correlations>
  <correlation set="POCorr" initiate="no" pattern="out">
   <correlation set="InvoiceCorr" initiate="yes" pattern="out">
 </correlations>
</invoke> ...
```

Scopes in BPEL

- Provide a shared context for subset of activities
- Can contain
  - fault handlers
  - event handlers,
  - compensation handler variables
  - correlation sets
- Can serialize concurrent access to variables

```
<scope
variableAccessSerializable="yes|no“
 ...>

<variables>
</variables>

<correlationSets>? ...
</correlationSets>

<faultHandlers>
</faultHandlers>

<compensationHandler>? ...
</compensationHandler>

<eventHandlers>
</eventHandlers>
(activities)*

</scope>
```

Long Running Transactions and Compensation



Compensation Handlers in BPEL

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

Exception Handling in BPEL

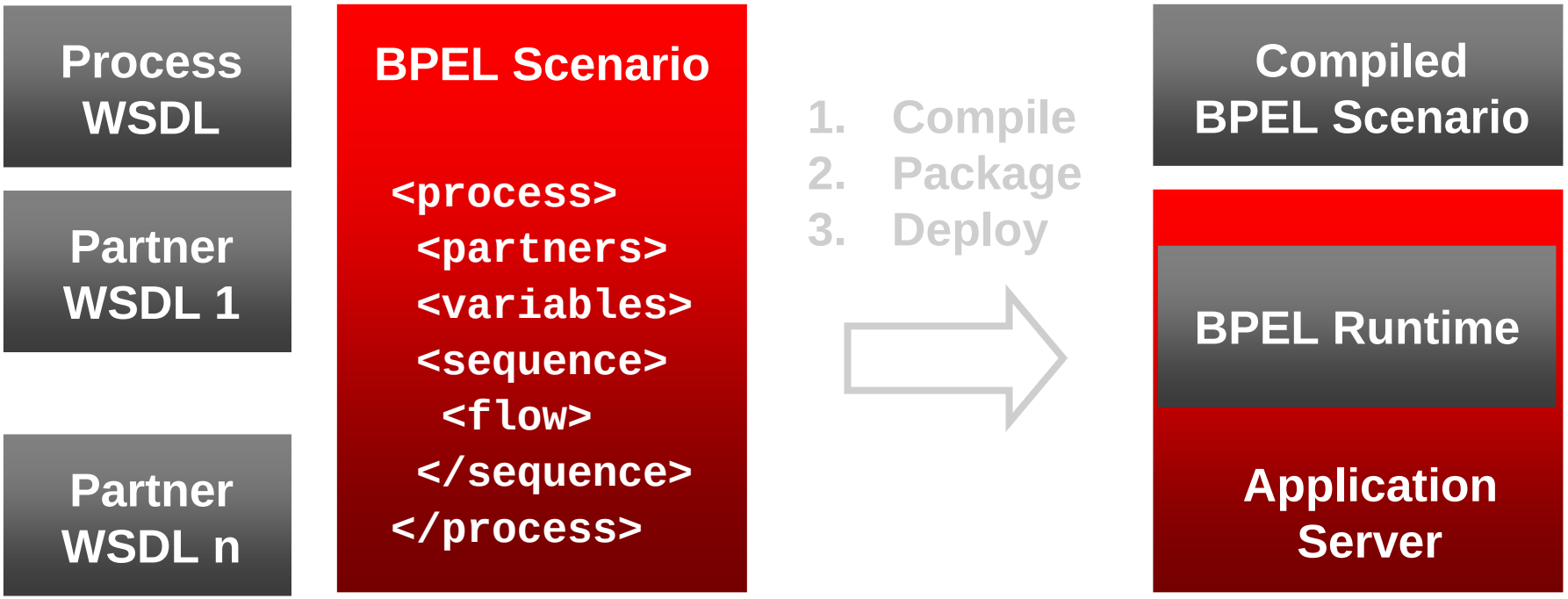
- `<faultHandlers>` catch exception

  Based on WSDL port defining fault

- `<faultHandlers>` can perform activities upon invocation

```
<faultHandlers>
    <catch faultName="lns:cannotCompleteOrder"
        faultVariable="POFault">
  <reply partner="customer"
         portType="lns:purchaseOrderPT"
         operation="sendPurchaseOrder"
         variable="POFault"
         faultName="cannotCompleteOrder"/>
 </catch>
</faultHandlers>
```

Just Show Me How to Do it!



Tooling Requirements

- IDE – build your Web services
- WSDL authoring – model your interfaces
- Schema authoring – model your messages
- Process modeling – model your orchestration
- Packaging and deployment
- Debugging
- Monitoring
- Analyzing