Unit 2

Software Development by Composition and Integration

Unit 2-4

Business Process and Execution

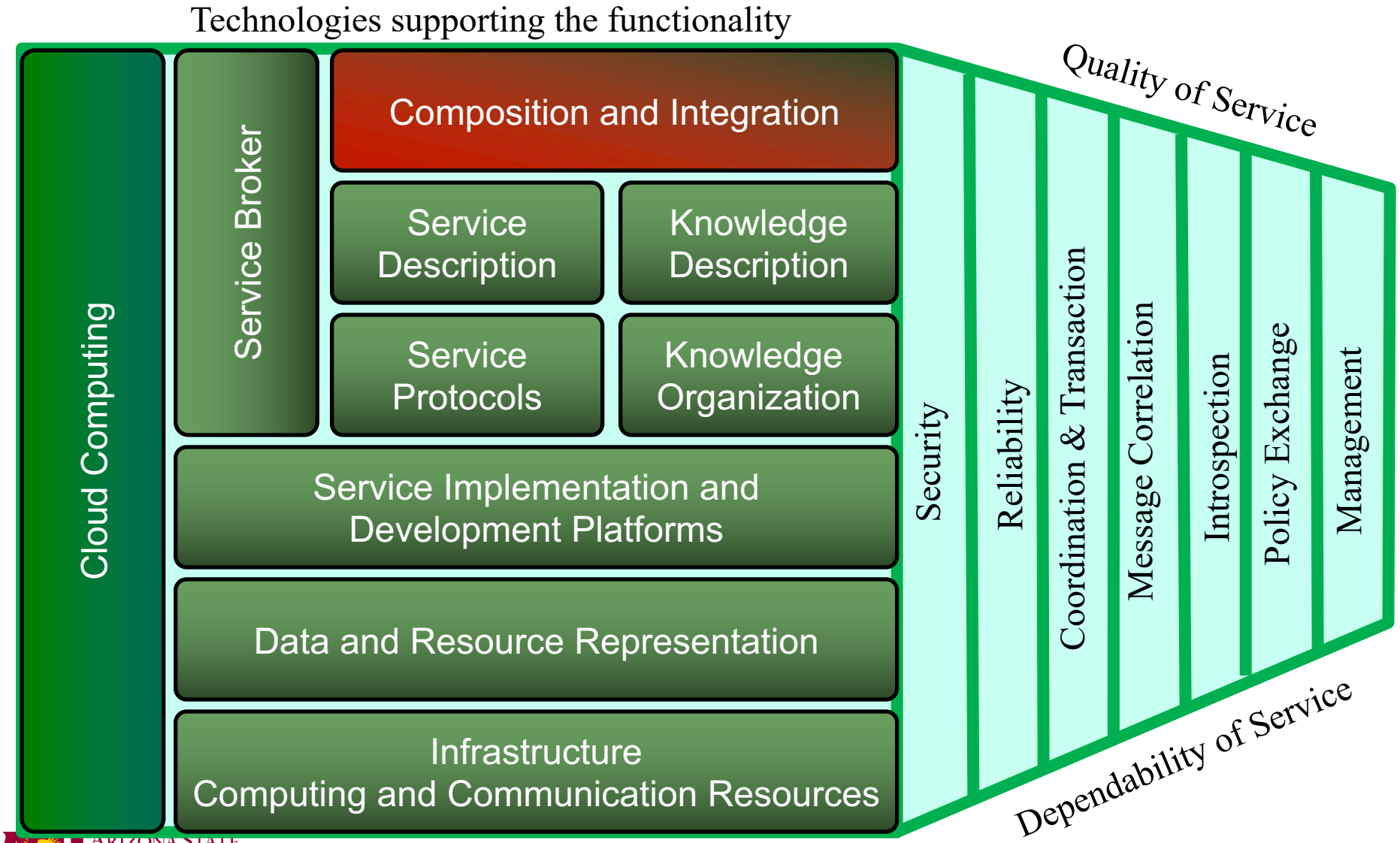
Dr. Yinong Chen

Unit 2 Outline

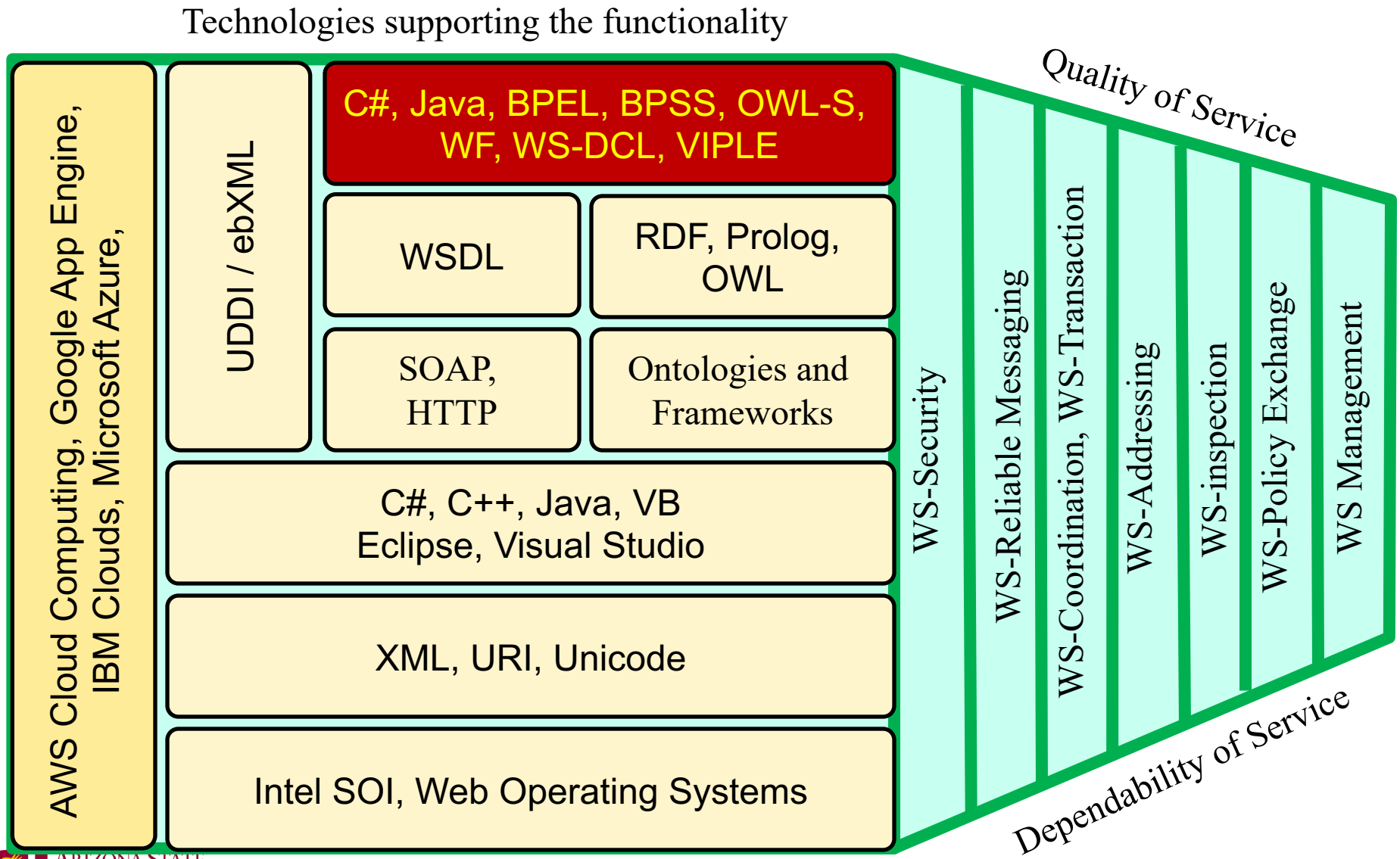
- 2-1 Enterprise Architecture and Business Process
- 2-2 Workflow Foundation 1: Concepts
- 2-3 Workflow Foundation 2: Case Study
- 2-4 **BPEL (Business Process Execution Language)**
  - Overview of Workflow and Orchestration
  - WSDL in BPEL
  - BPEL constructs and BPEL Process Definition
- 2-5 A Case Study of BPEL Application
- 2-6 Stateful Services
- 2-6 Development Frameworks Supporting BPEL
  - Oracle SOA Suite, BPMN, and BizTalk
- 2-7 Message-Based Integration
- 2-8 Web Caching and Recommendation

Unit Test 2

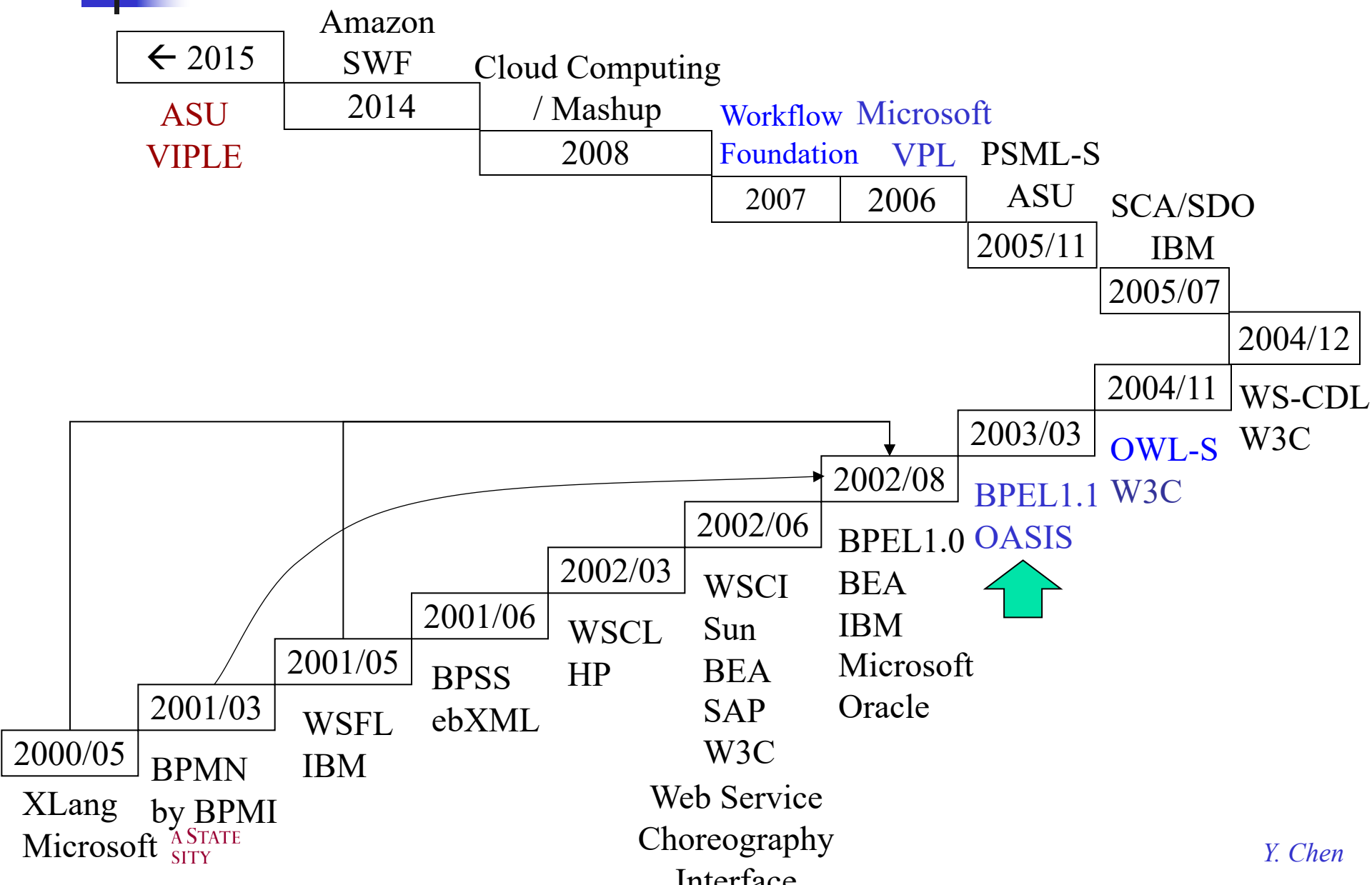
Organization of SOC-Enabling Technologies



Instances of the SOC-Enabling Technologies



Composition and Business Process Languages



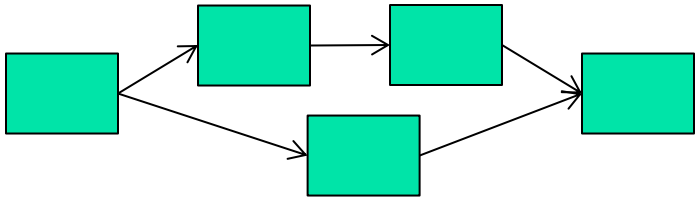
BPEL vs. other Composition Languages

Language	Architecture Style	Other features
BPEL 2002	orchestration	Based on SOAP, WSDL, UDDI directory, widely used by large corps
BPSS 2001 *BP Spec. Schema*	choreography	Business Process Specification Schema Based on SOAP, ebXML repository, CPP/CPA collaboration, for small biz
BPMN 2001 *BP Modeling Notation*	orchestration	A superset of BPEL, supports advanced semantics & complex structures, by BPMI which merged with OMG in 2005
WSCI 2002	choreography	WS Choreography Interface: Complementary to BPEL, submitted to W3C, not widely used
WS-CDL 2004	choreography	Complementary to BPEL, W3C own proposal
PSML-S 2005	choreography	From ASU. Focus on dynamic behaviors: dynamic discovery, matching, cooperation

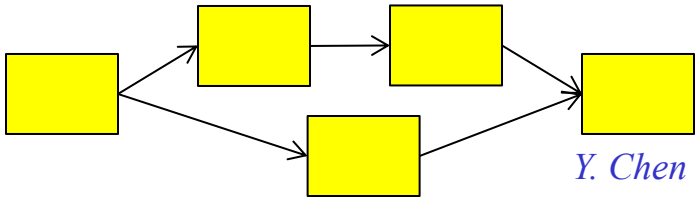
Business Process vs. Workflow

- A business process is a sequence or a collection of business activities, involving actors (humans or machines) and actions, to perform on business inputs and generated business results.
- A workflow is a composition (programming) or flow management technology that organizes the components and services and defines their order of execution.
- Workflow frameworks are designed to describe business processes, so that business process can be
  - precisely described or specified
  - executed automatically

Business Process

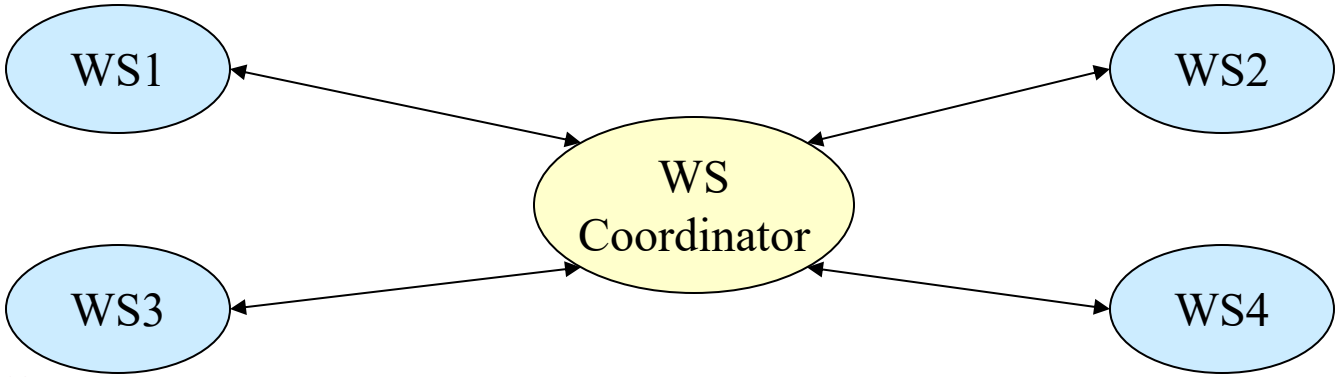


Workflow from Developer's view



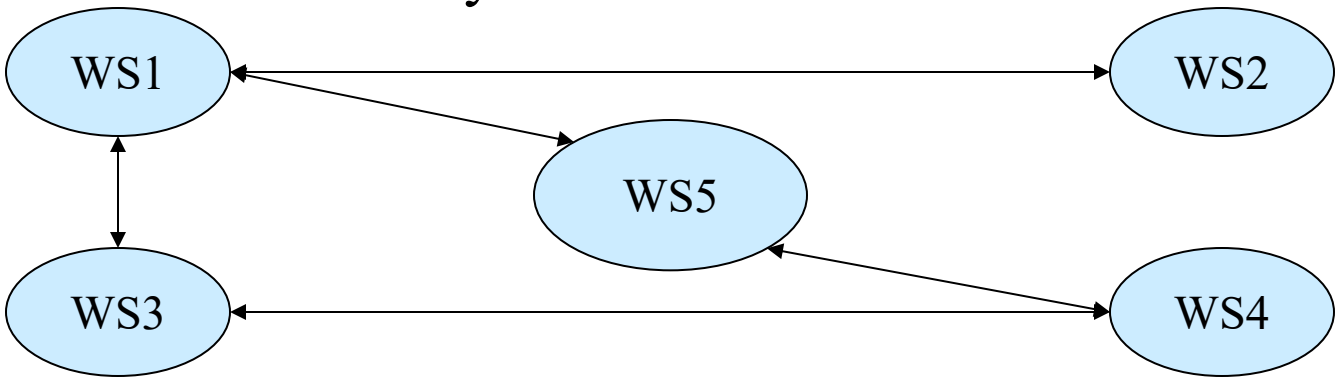
Composition Style: Orchestration

- A central process, which can be a service itself, takes control over the involved services and coordinates the execution of different operations
  - Involved services communicate with the central process only, within the application;
  - How service functionality is achieved by aggregating other Web services
  - Useful for private business process, using independent services
  - BPEL uses this style



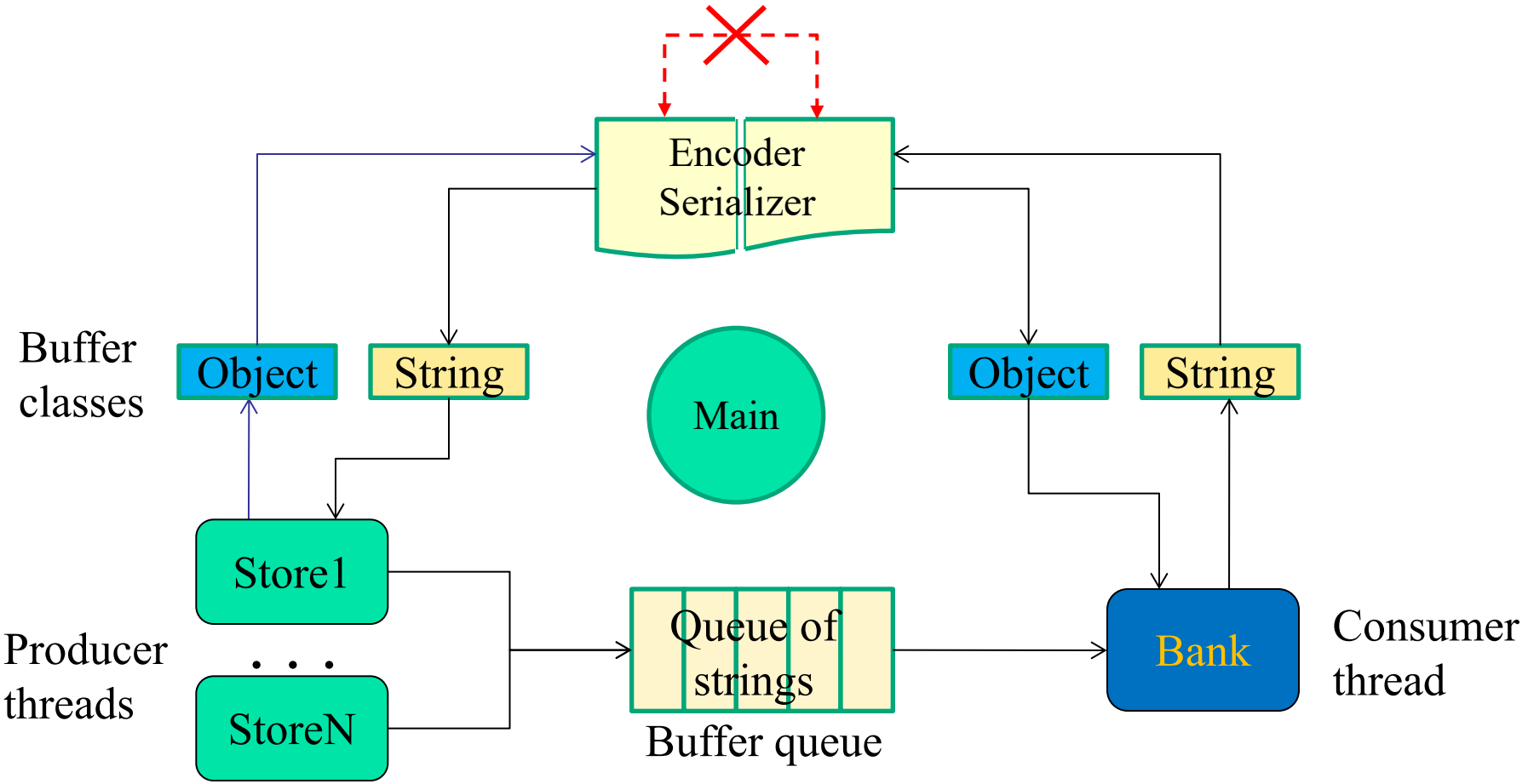
Composition Style: Choreography

- There is no central coordinator.
  - Each service involved can communicate with multiple partners within the application;
  - How to interact with individual services to consume its functionality
  - Useful for public business process involving coordinated design of distributed services
  - WS-CDL uses this style



- Increasing complexity of composition, the line between the two styles is vanishing, and the styles are not emphasized.

Orchestration Vs. Choreography



Amazon Simple Workflow (SWF)

http://docs.aws.amazon.com/amazonswf/latest/awsflowguide/

- Amazon SWF provides a high-level way for developers to compose distributed and asynchronous workflow applications.
- It is a programming environment that simplifies the process of defining workflow and thus simplifies the implementation of a business process using Amazon SWF.
- It can be used for implementing a broad range of applications, including business processes, media encoding, long-running tasks (persistent), and background processing.

Amazon SWF Application Structure

5. SWF monitors, manages, coordinates, synchronizes the execution

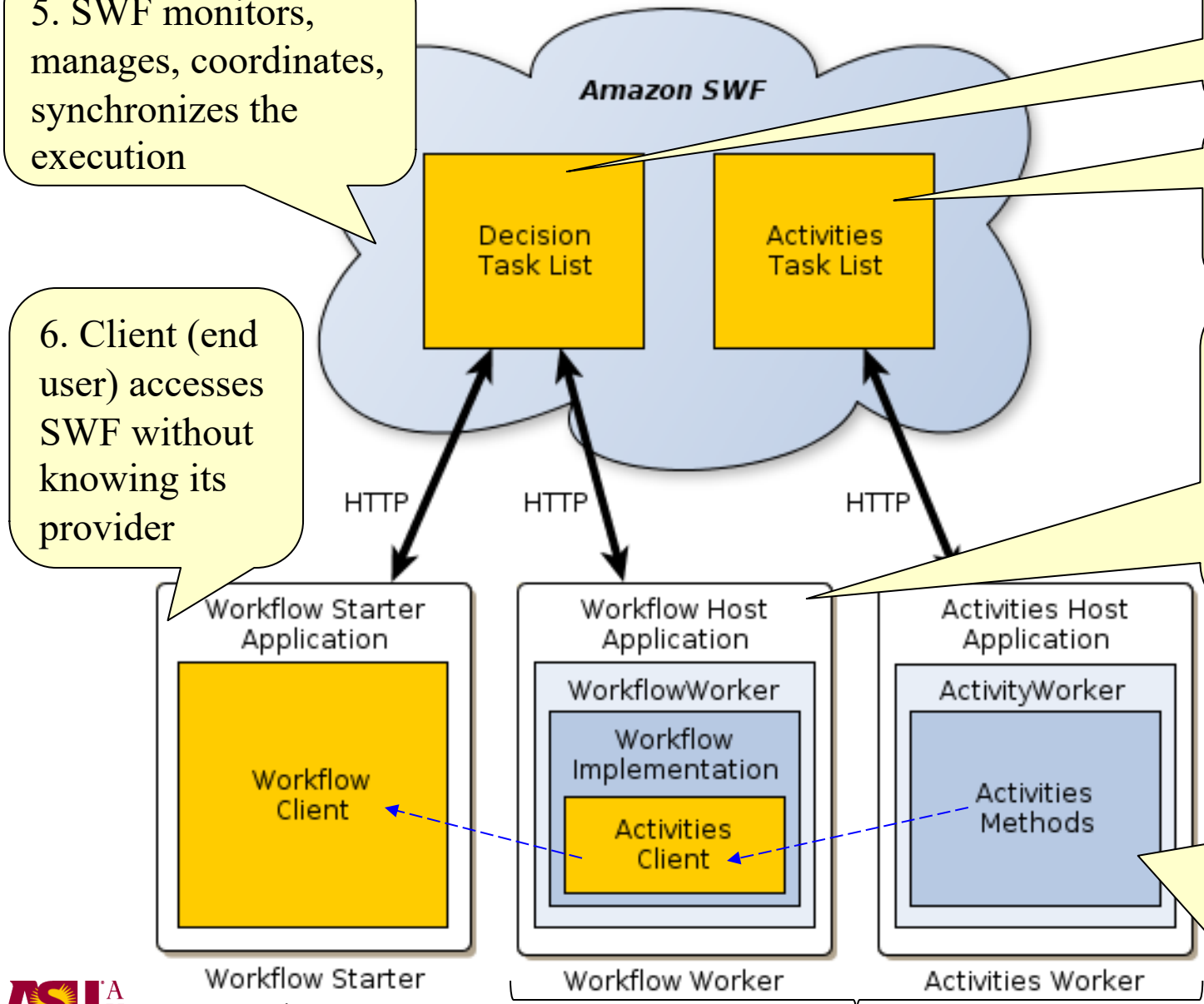
4. Workflow developer registers app to SWF

2. Activity developer registers activities to SWF

6. Client (end user) accesses SWF without knowing its provider

3. Workflow developer writes the workflow that connects your activities, and host them as app

1. Activity developer writes the activity methods and host them anywhere on workers as services



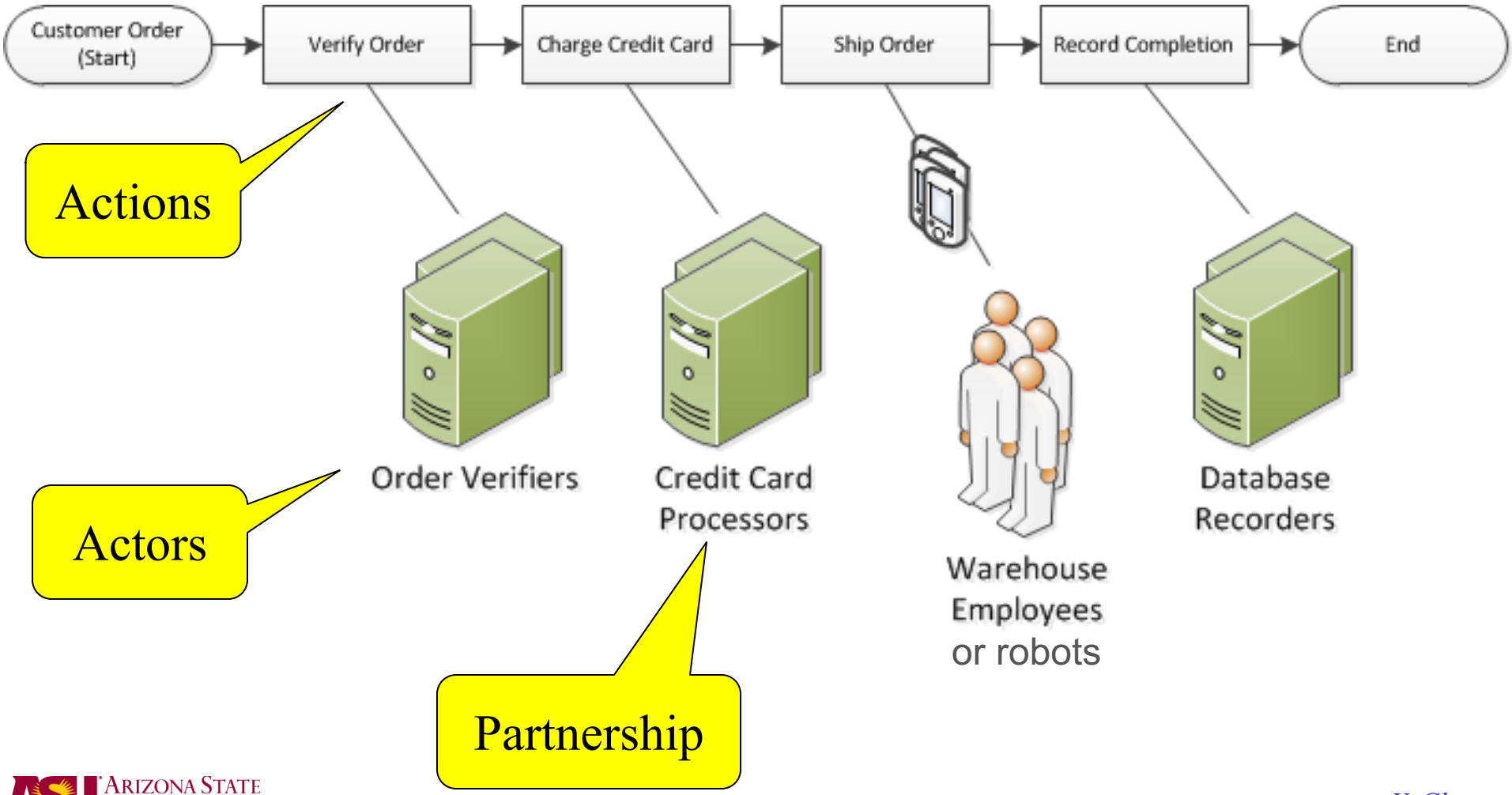
Creating Your Workflow in Amazon SWF

1. Write or discover existing activity methods that are needed in your workflow. Host your own services.
2. Write a decider to implement the coordination logic of your workflow.
3. Register your activities and workflow with Amazon SWF. You can do this step programmatically or by using the AWS Management Console.
4. Start your activity workers and decider.
5. Start one or more executions of your workflow.
6. View workflow executions using the AWS Management Console.

Note: The above process shows that you develop both activities and the workflow. Normally, you do not do both.

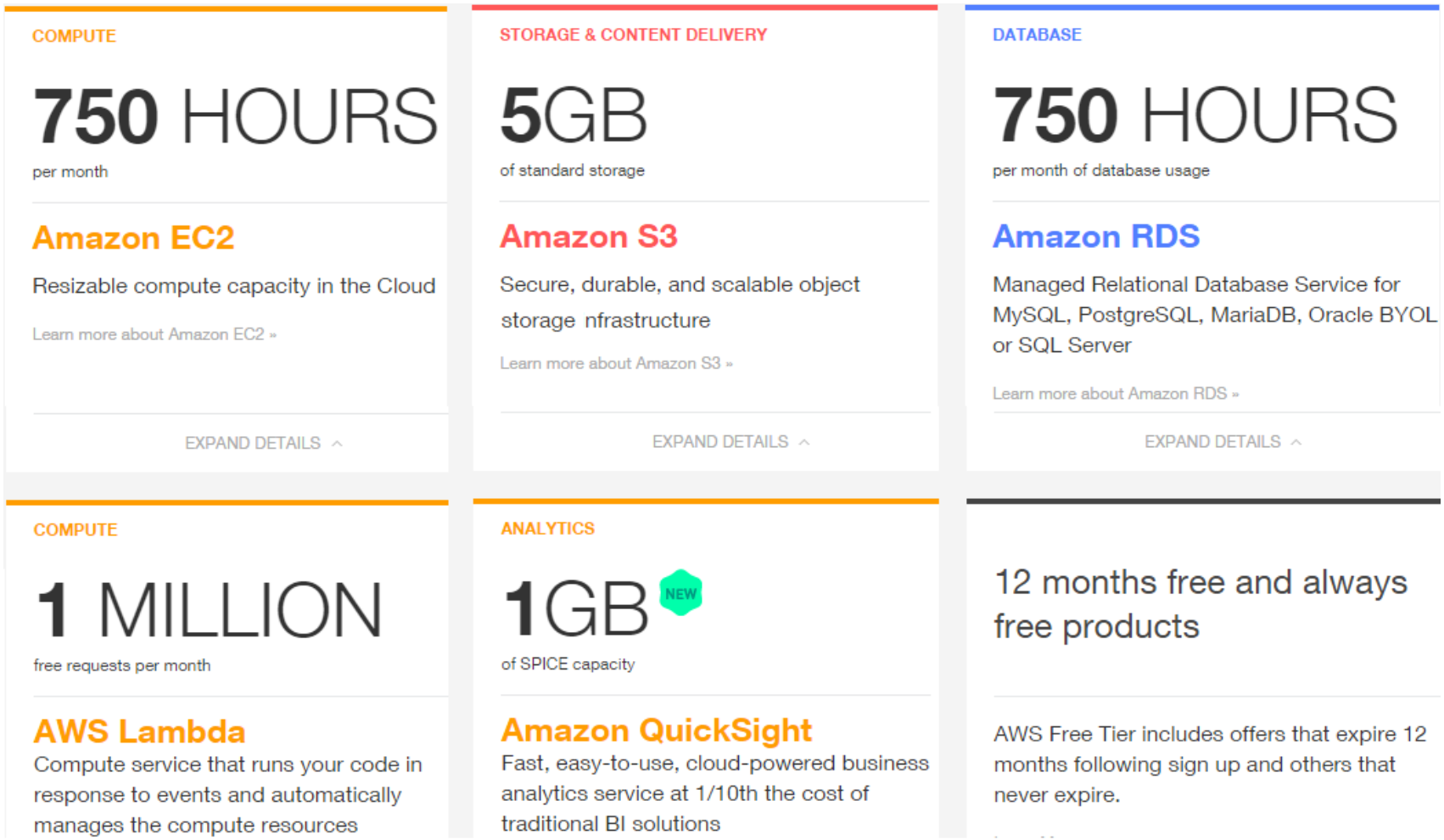
Amazon Workflow Example in E-Commerce

http://docs.aws.amazon.com/amazonswf/latest/developerguide/swf-dev-about-workflows.html



AWS Free Tier

https://aws.amazon.com/cn/s/dm/optimization/server-side-test/free-tier/free_np/



BPEL versus Workflow Foundation

- Similarities (Considering the graphic view of BPEL)
  - Architecture-driven approach
  - Add another layer of abstraction to programming
  - Workflow/Process languages
  - Graphic tools for visual development, e.g., Oracle SOA Suite
  - Similar constructs
  - Executable
- Differences
  - BPEL is more towards enterprise level software integration and is closer to the business architecture;
  - WF is closer to programming languages and allows to mix with programming languages, e.g., CodeActivity.

BPEL, BPEL4WS, WS-BPEL

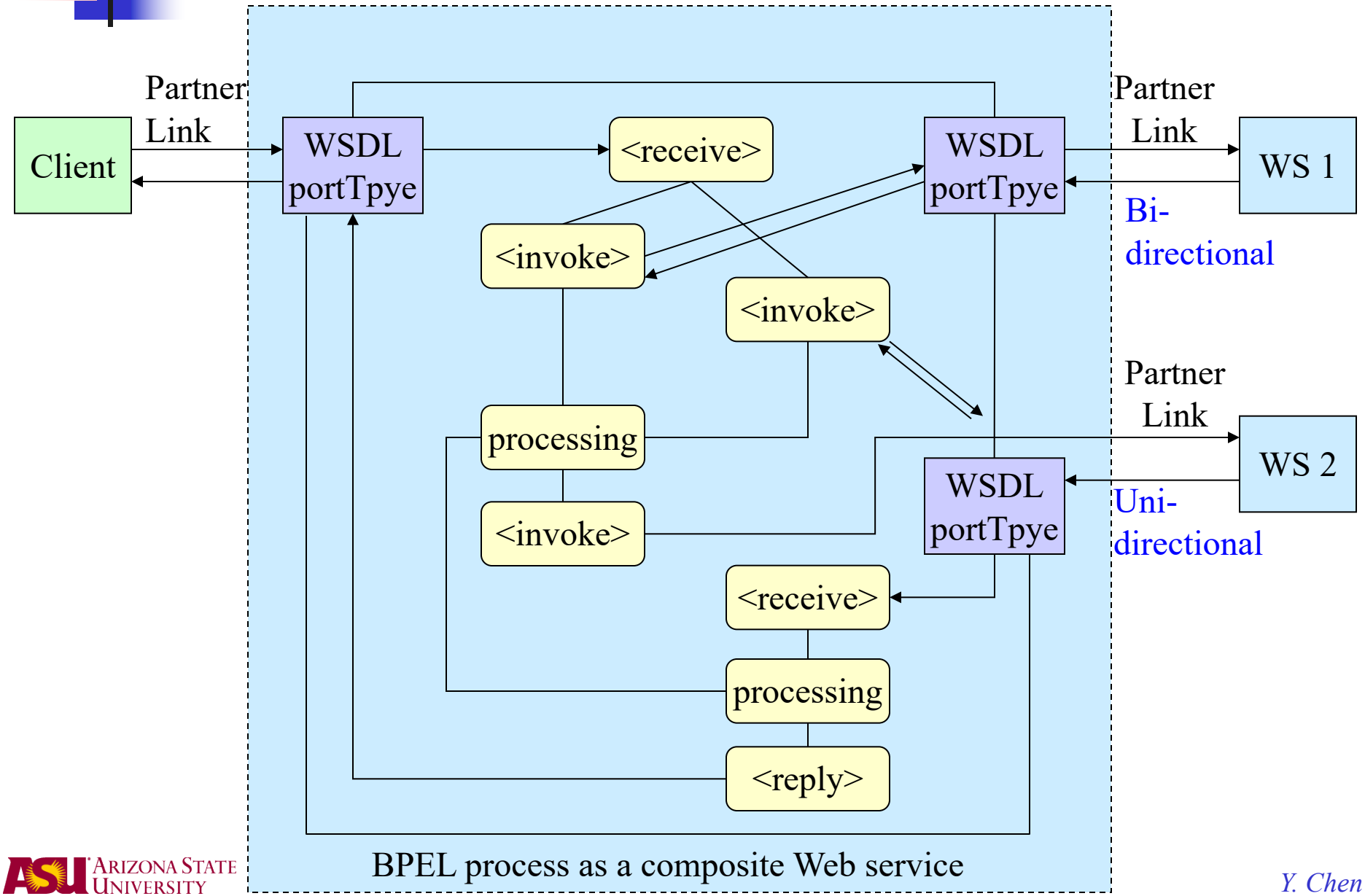
http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-bpel/ws-bpel.pdf

- A BPEL process defines the structure of the interaction in terms of
  - participant services (partners)
    - Characterize partners
    - Provide support to partner conversation
  - business logic.
    - Data
    - Control flow
    - Error handling and recovery mechanisms

BPEL Partners: Input / Output of a Process

- Partners:
  - A composition defines a new service that interacts with one or more partners.
  - A partner is characterized by the WSDL interface portType
- Interactions between partners can be unidirectional or bidirectional. You may define
    - Synchronous and asynchronous interactions.
    - Stateless and Stateful.
- How is the state maintained in stateful services?
  - BPEL correlation mechanism uses business data to maintain the state of the interaction.
  - Other middleware mechanisms are possible.

Developing a Business Process with BPEL



BPEL Basic Activities

A BPEL process consists of steps, each step is an activity:

<invoke>	Invoking a service
<receive>	Waiting for receiving a message
<reply>	Generating response to a synchronized service call
<assign>	Modifying data variables
<throw>	Jumping to an exception handler
<catch>	Handling an exception
<wait>	Waiting for a certain amount of time
<terminate>	Terminating the entire process

BPEL Constructs

A BPEL process can use constructs to define complex activities:

<sequence>	Define a set of activities that will be executed in the given order of sequence.
<flow>	Define a set of activities that will (can) be executed in parallel.
<switch>	Define a selection (if-then-else)
<while>	Define a while-loop

<while condition="expr" standard-attributes>

standard-elements

activities

</while>

<pick>	Define a list of activities. Execute the first activity that is available, e.g., can be used to block wait for a response, or timeout, whichever comes first.
<partnerLink>	Define a partner link
<variable>	Declare a variable

Process Example: Select the Best Insurance

```
<?xml version= "1.0" encoding="utf-8"?>
    <process name="insuranceSelectionProcess"
        targetNamespace="http://bpelexample.com/bpel/example/"
        xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
        xmlns bpws="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
        xmlns:ins="http://bpelexample.com/bpel/insurance/"
        xmlns:com="http://bpelexample.com/bpel/company/">
    <partnerLinks>
①       <partnerLink name = "client"
                partnerLinkType = "com:selectionLT"
                myRole = "insuranceSelectionService"/>
②       <partnerLink name = "insuranceA"
                partnerLinkType ="ins:insuranceLT"
                myRole = "insuranceSelectionRequester"
                partnerRole = "insuranceSelectionService"/>
③       <partnerLink name = "insuranceB"
                partnerLinkType= "ins:insuranceLT"
                myRole = "insuranceSelectionRequester"
                partnerRole = "insuranceSelectionService"/>
    </partnerLinks>
```

Root element

One role

Two roles

Two roles

This business process has 3 partners: Client, insureanceA, and insuranceB

Example (contd.): Variables

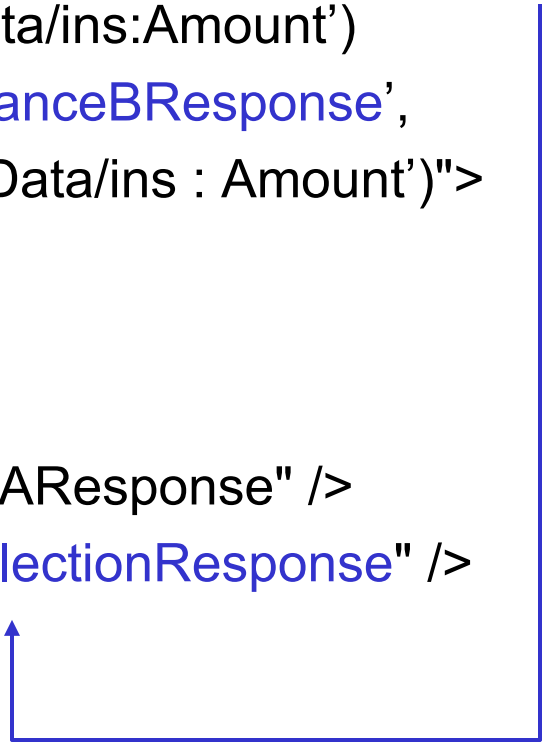
```
<variables>
    <!-- input for BPEL process -->
    <variable name="InsuranceRequest"
        messageType="ins:InsuranceRequestMessage"/>
    <!-- output from insurance A -->
    <variable name="InsuranceAResponse"
        messageType="ins:InsuranceResponseMessage"/>
    <!-- output from insurance B -->
    <variable name="InsuranceBResponse"
        messageType="ins:InsuranceResponseMessage"/>
    <!-- output from BPEL process -->
    <variable name="InsuranceSelectionResponse"
        messageType="ins:InsuranceResponseMessage"/>
</variables>
```

Example (contd.): Sequence and Flow

```
<sequence>
    <!-- Receive the initial request from client -->
    <receive partnerLink="client"
        portType="com:InsuranceSelectionPT"
        operation="SelectInsurance"
        variable="InsuranceRequest"
        createInstance= "yes" />
    <!-- Make concurrent invocations to Insurance A and B -->
    <flow>
        <!-- Invoke Insurance A web service -->
        <invoke partnerLink="insuranceA"
            portType="ins:ComputeInsurancePremiumPT"
            operation="computeInsurancePremium"
            inputvariable= "InsuranceRequest"
            outputVariable="InsuranceAResponse"/>
        <!-- Invoke Insurance B web service -->
        <invoke partnerLink="insuranceB"
            portType="ins:ComputeInsurancePremiumPT"
            operation="computeInsurancePremium"
            inputvariable= "InsuranceRequest"
            outputVariable="InsuranceBResponse"/>
    </flow>
```

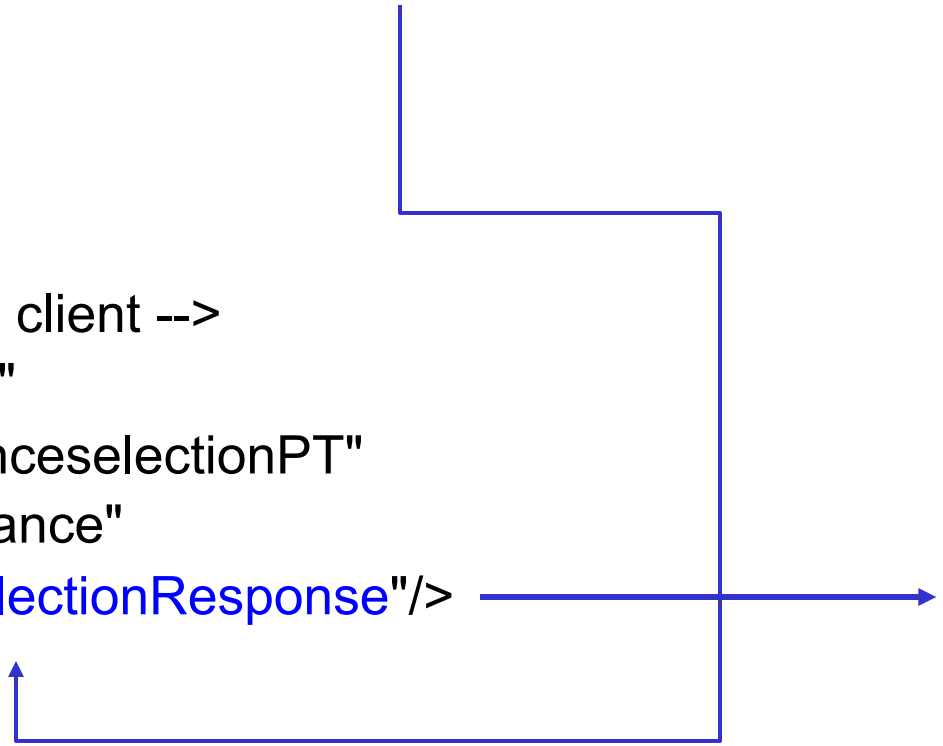
Example (contd.): Switch (if-then-else)

```
<!-- select the best offer and construct the response -->
  <switch>
    <case condition="bpws:getvariableData('InsuranceAResponse',
    'confirmationData', '/confirmationData/ins:Amount')
      &lt;= bpws:getvariableData('InsuranceBResponse',
      'confirmationData', '/confirmationData/ins : Amount')">
      <!-- select Insurance A -->
      <assign>
        <copy>
          <from variable="InsuranceAResponse" />
          <to variable="insuranceSelectionResponse" />
        </copy>
      </assign>
    </case>
```



Example (contd.): Switch

```
            <otherwise>
                <!-- select Insurance B -->
                <assign>
                    <copy>
                        <from variable="InsuranceBResponse" />
                        <to variable="InsuranceSelectionResponse" />
                    </copy>
                </assign>
            </otherwise>
        </switch>
        <!-- send a response to the client -->
        <reply   partnerLink="client"
            portType="com:insuranceselectionPT"
            operation="selectInsurance"
            variable="InsuranceSelectionResponse"/>
    </sequence>
</process>
```



The Root Element "process"

Define the qualifiers and namespaces of the process

- Target namespace
- Default namespace
- Additional namespaces in which other names are defined

Read text chapter 4 if not familiar with XML

```
<?xml version="1.0" encoding="utf-8"?>

  <process name="insuranceSelectionProcess"
    targetNamespace="http://bpelexample.com/bpel/example/"
    xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
    xmlns bpws="http://schemas.xmlsoap.org/ws/2003/03/business-
    process/"
    xmlns:ins="http://bpelexample.com/bpel/insurance/"
    xmlns:com="http://bpelexample.com/bpel/company/">
```

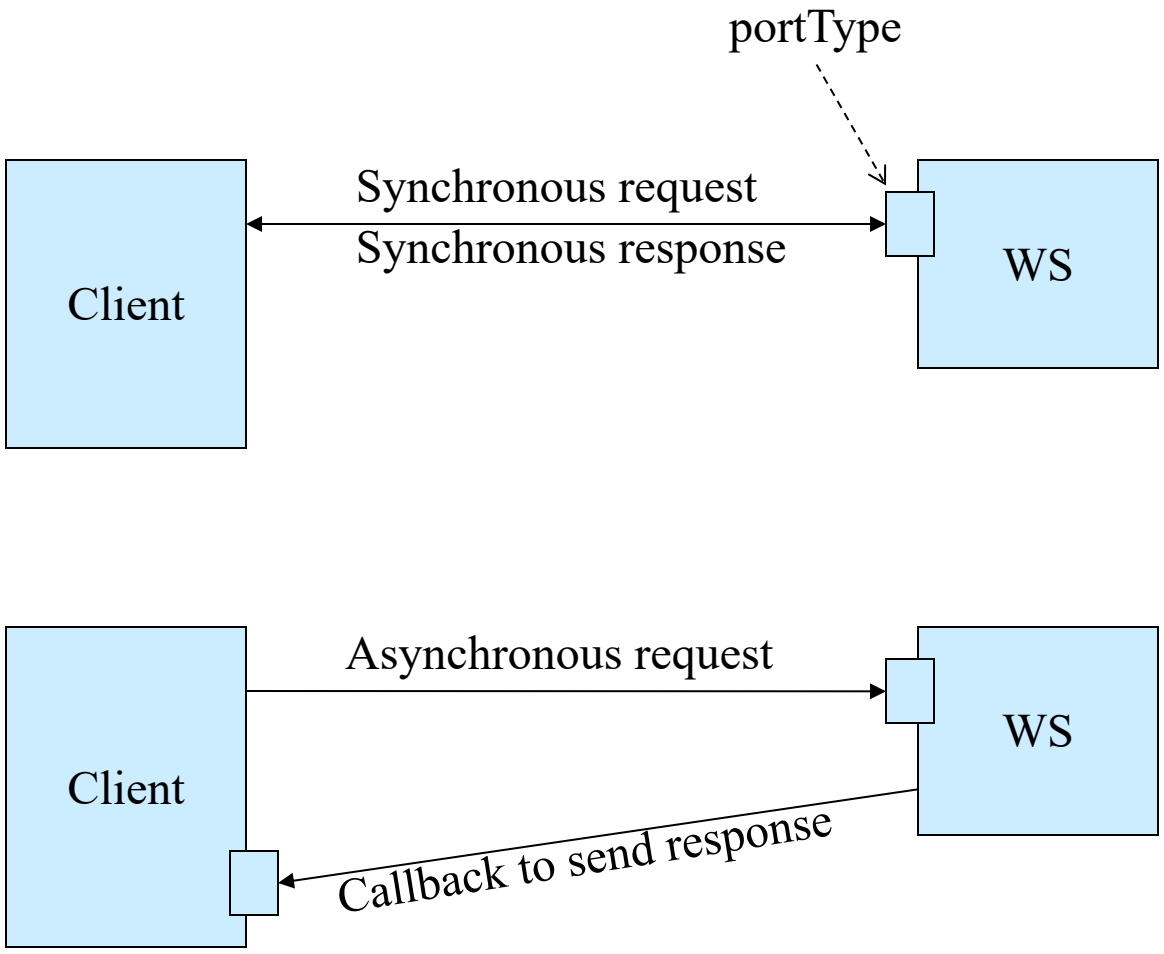
Default

Invoking Web Services in BPEL Process

BPEL supports two methods of two-way communication:

- Synchronous:
  - Send a request to a WS, and block-wait for receiving the response;
  - Suitable for the situation where the result is expected in little time.
- Asynchronous:
  - Send a request to a WS and start to take other activities.
  - Let the callee WS to call back;
  - Suitable for the situation where the result may take a long time
  - The caller must be a service, with a portType

Invoking Web Services in BPEL Process



Invoking WS Sequentially

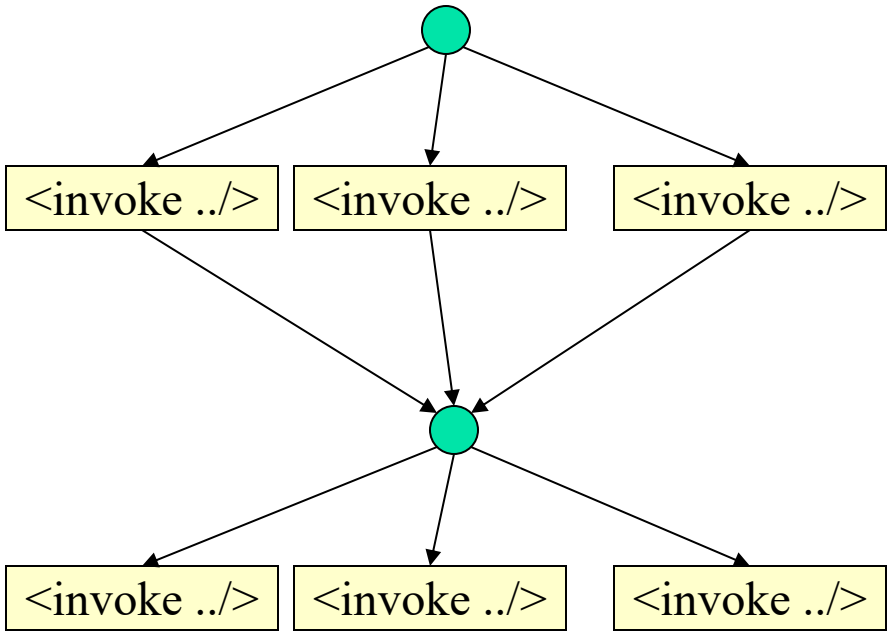
```
<process>
    <!– waiting for request to start processing -->
    <receive … />
    . . .
    <!– Start to process the request by calling multiple WS one by one -->
    <sequence>
        <!-- Invoke Insurance web services -->
        <invoke  …/>
        <invoke  …/>
        <invoke  …/>
         . . .
    </sequence>
</process>
```

Invoking WS in Parallel

```
<process>
    <!– waiting for request to start processing -->
    <receive … />
     . . .
     <!– Start to process the request by calling multiple WS in parallel -->
     <flow>
        <!-- Invoke Insurance web services -->
        <invoke  …/>
        <invoke  …/>
        <invoke  …/>
         . . .
     </flow>
</process>
```

Invoking WS Sequentially and in Parallel

```
<process>
    <!-- waiting for request to start processing -->
    <receive … />
    . . .
    <!-- Start to process the request by calling multiple WS sequentially and in
       parallel -->
   <sequence>
        <flow>
                <invoke  …/>
                <invoke  …/>
                . . .
        </flow>
        <flow>
                <invoke  …/>
                <invoke  …/>
                . . .
        </flow>
    </sequential>
</process>
```

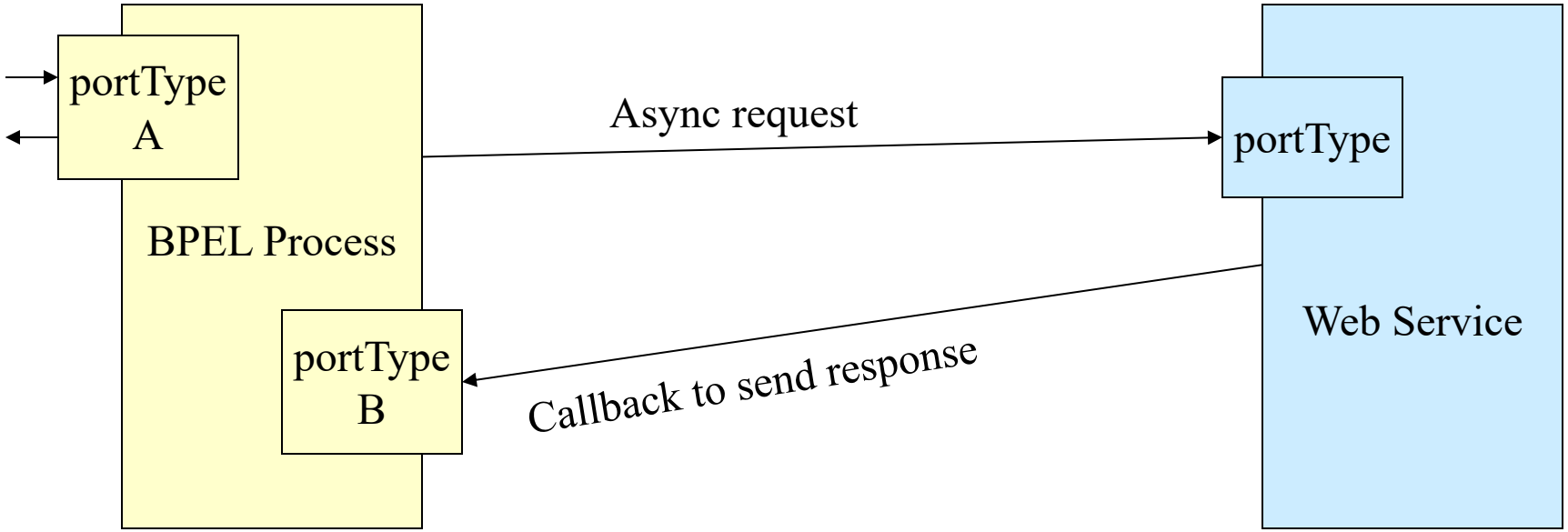


Each Business Process is a Web Service

Each BPEL process is typically a composite Web service, with the following features

- Clients will call the process as a WS;
- At least one WS will be called by the process (otherwise, it is not a composite WS)
- A process will have a WSDL file to define how the clients or other WS can call the process
- A process can offer synchronous and asynchronous communication methods to the clients or other WS.

WSDL portType of the BPEL Process

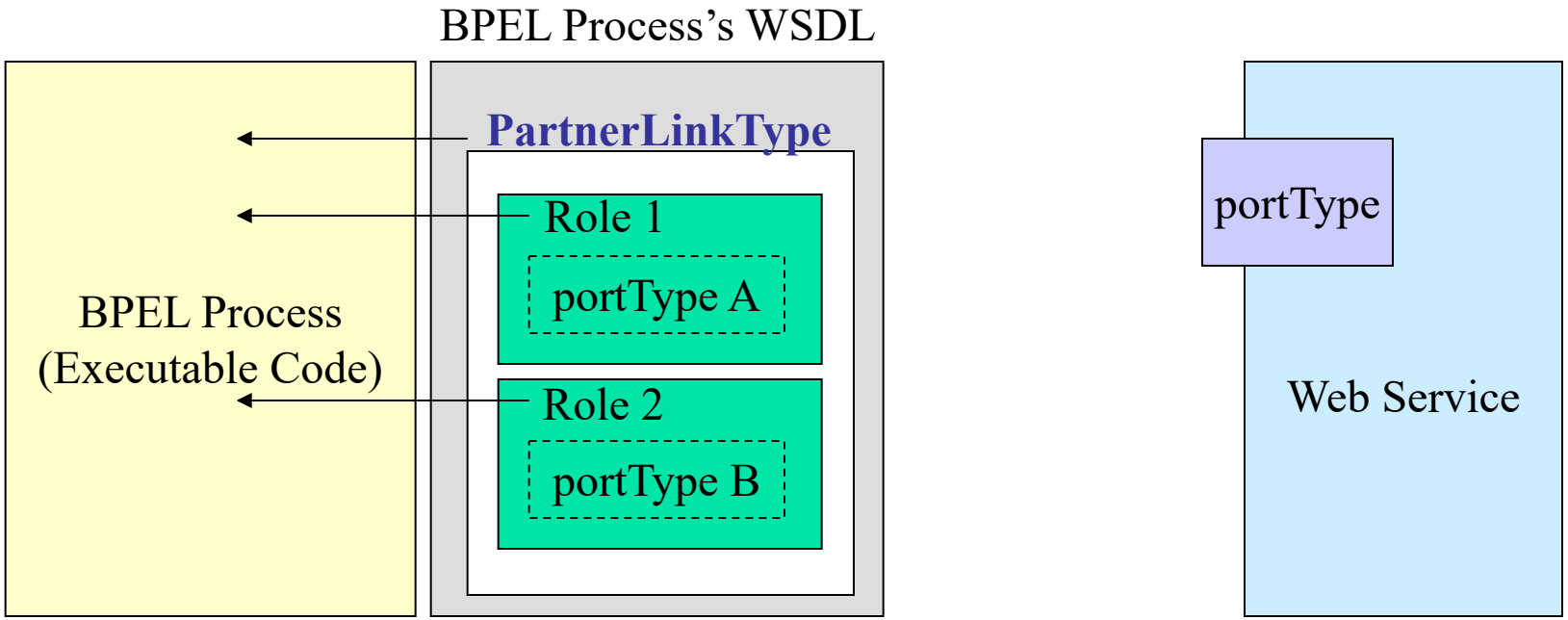


Consider the WSDL portType: A and B

- The BPEL process offers portType A to the client
- The BPEL process offers portType B for the WS to callBack
- The BPEL process will access the portType offered by WS

PartnerLinkType for BPEL Process in WSDL

A new element, called PartnerLinkType is added into the WSDL file of the process, using the **extensibility** of WSDL.
This element does not have to be in the services that interact with the BPEL process.



WSDL File of the BPEL Process

```
<?xml version= "l.0" encoding="UTF-8"?>
    <definitions
        xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
        xmlns:soap="http://schemas xmlsoap.org/wsdl/soap/"
        xmlns:xs= "http://www.w3.org/2001/XMLschema"
        xmlns: soapenc="http://schemas.xml soap.org/soap/encoding/"
        xmlns :ins="http://bpelexample.com/bpel/insurance/"
        xmlns : com="http://bpelexample.com/bpel /company/"
        targetNamespace ="http://bpelexample.com/bpel/company/"
        xmlns="http://schemas.xmlsoap.org/wsdl/"
        xmlns : plnk= "http://schemas.xmlsoap.org/ws/2003/05/partner-link/" >
        <types>
            <xs:schema ... >
            </xs:schema>
        </types>
        <message ... >
            <part ... />
        </message>
```

Default

Additional namespace

WSDL File of the BPEL Process (contd.)

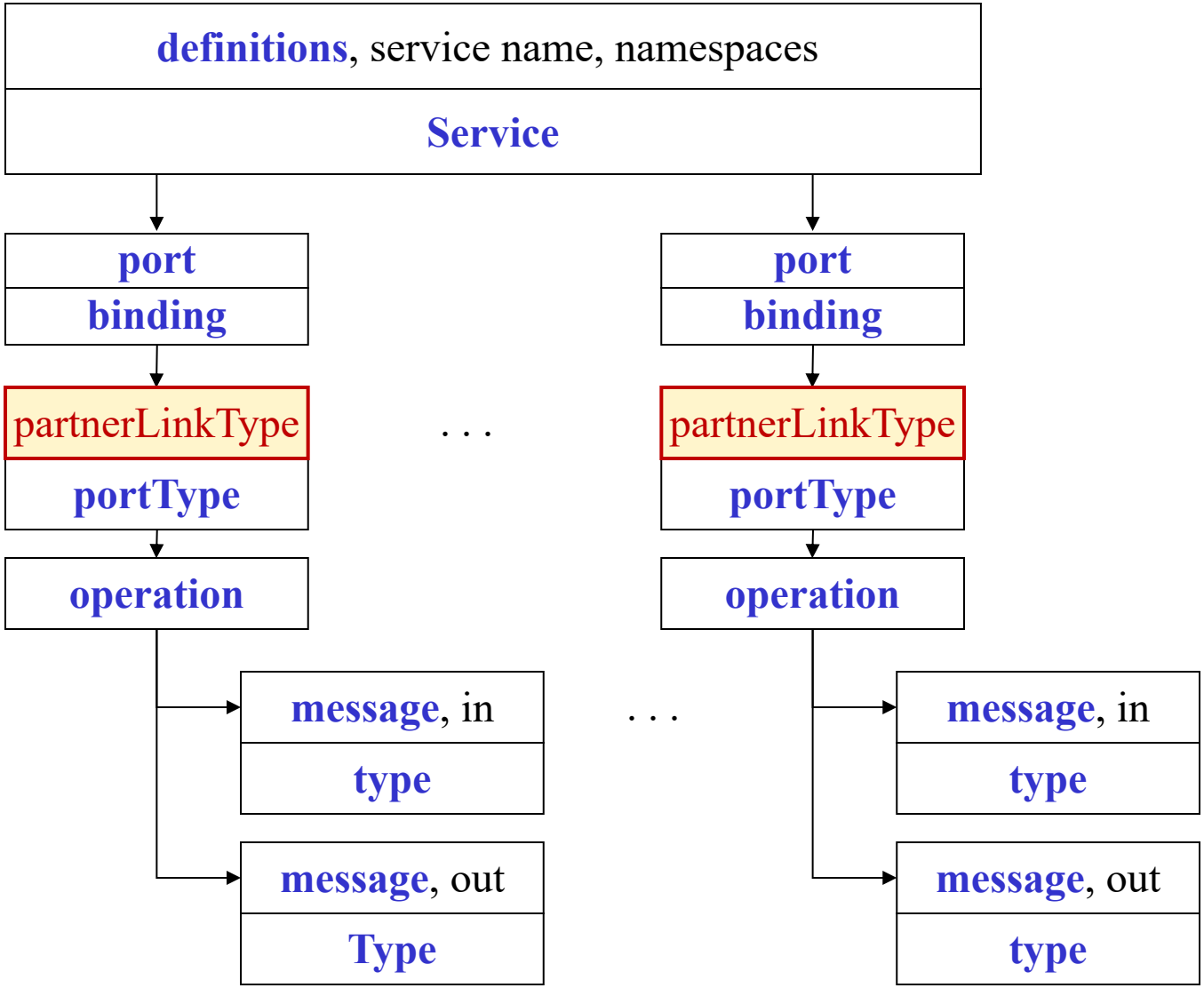
```
<portType name="computeInsurancePremiumPT">
    <operation name="...">
        <input message="..." />
        <output message="..." />
    </operation>
</portType>
<portType name="ComputeInsurancePremiumCallbackPT">
    <operation name="...">
        <input message="..."/>
    </operation>
</portType>
<plnk:partnerLinkType name=insuranceLT>
    …
</plnk: partnerLinkType>
```

Next page

WSDL File of the BPEL Process (contd.)

```
<plnk:partnerLinkType name=insuranceLT>
    <plnk:role name= "insuranceService">
      <plnk:portType name="ins:computeInsurancePremiumPT"/>
    </plnk: role>
    <plnk: role name="insuranceRequester">
      <plnk:portType name="ins:ComputeInsurancePremiumCallbackPT"/>
    </plnk: role>
</plnk: partnerLinkType>
</definitions>
```

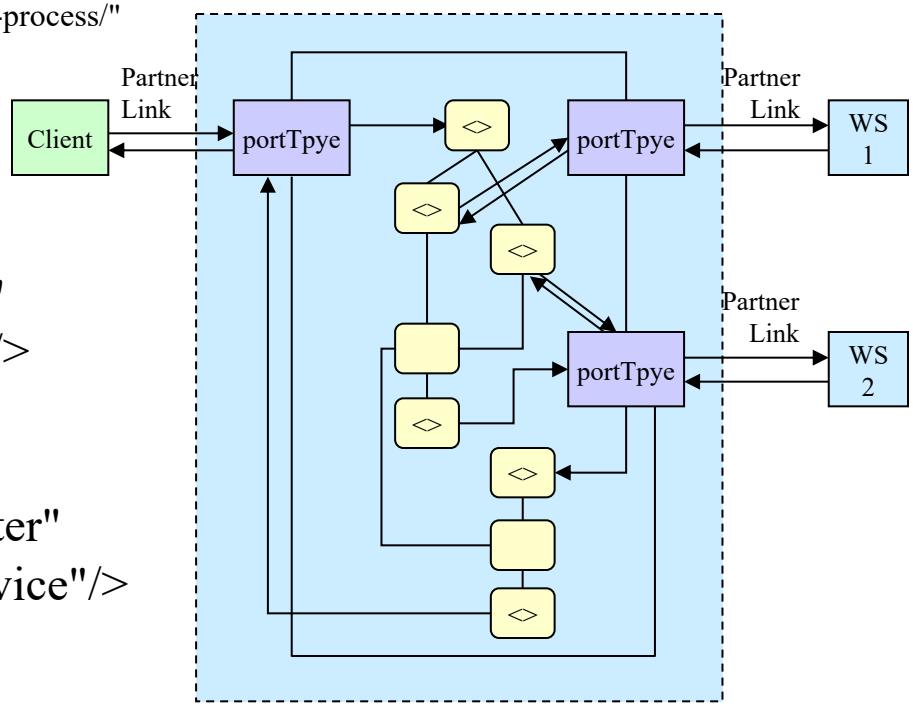
WSDL with the plnk extension



Define the process with the partnerLink using partnerLinkType

```
<?xml version="1.0" encoding="utf-8"?>
<process name="insuranceSelectionProcess"
        targetNamespace="http://bpelexample.com/bpel/example/"
        xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
        xmlns bpws="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
        xmlns:ins="http://bpelexample.com/bpel/insurance/"
        xmlns:com="http://bpelexample.com/bpel/company/">

    <partnerLinks>
        <partnerLink name= "client"
            partnerLinkType = "com:selectionLT"
            myRole="insuranceSelectionService"/>
        <partnerLink name = "insuranceA"
            partnerLinkType = "ins:insuranceLT"
            myRole = "insuranceSelectionRequester"
            partnerRole = "insuranceSelectionService"/>
        <partnerLink name = "insuranceB"
            partnerLinkType = "ins:insuranceLT"
            myRole = "insuranceSelectionRequester"
            partnerRole = "insuranceSelectionService"/>
    </partnerLinks>
```



Using the partnerLink in the BPEL Process

```
<receive partnerLink="client"
    portType="com:InsuranceSelectionPT"
    operation="SelectInsurance"
    variable="InsuranceRequest" createInstance= "yes" />

<flow>
    <invoke partnerLink= "insuranceA"
        portType= "ins:ComputeInsurancePremiumPT"
        operation= "computeInsurancePremium"
        inputvariable= "InsuranceRequest"
        outputVariable= "InsuranceAResponse"/>

    <invoke partnerLink= "insuranceB"
        portType= "ins:ComputeInsurancePremiumPT"
        operation= "computeInsurancePremium"
        inputvariable= "InsuranceRequest"
        outputVariable="InsuranceBResponse"/>
</flow>
```

BPEL Variables

- BPEL business processes model the exchange of messages between involved web services.
- Messages are exchanged as operations are invoked.
- When the business process invokes an operation and receives the result, we often want to store that result for subsequent invocations, use the result as is, or extract certain data.
- BPEL provides variables to store and maintain the state.
- Variables can also hold data that relates to the state of the process, but will never be exchanged with partners.

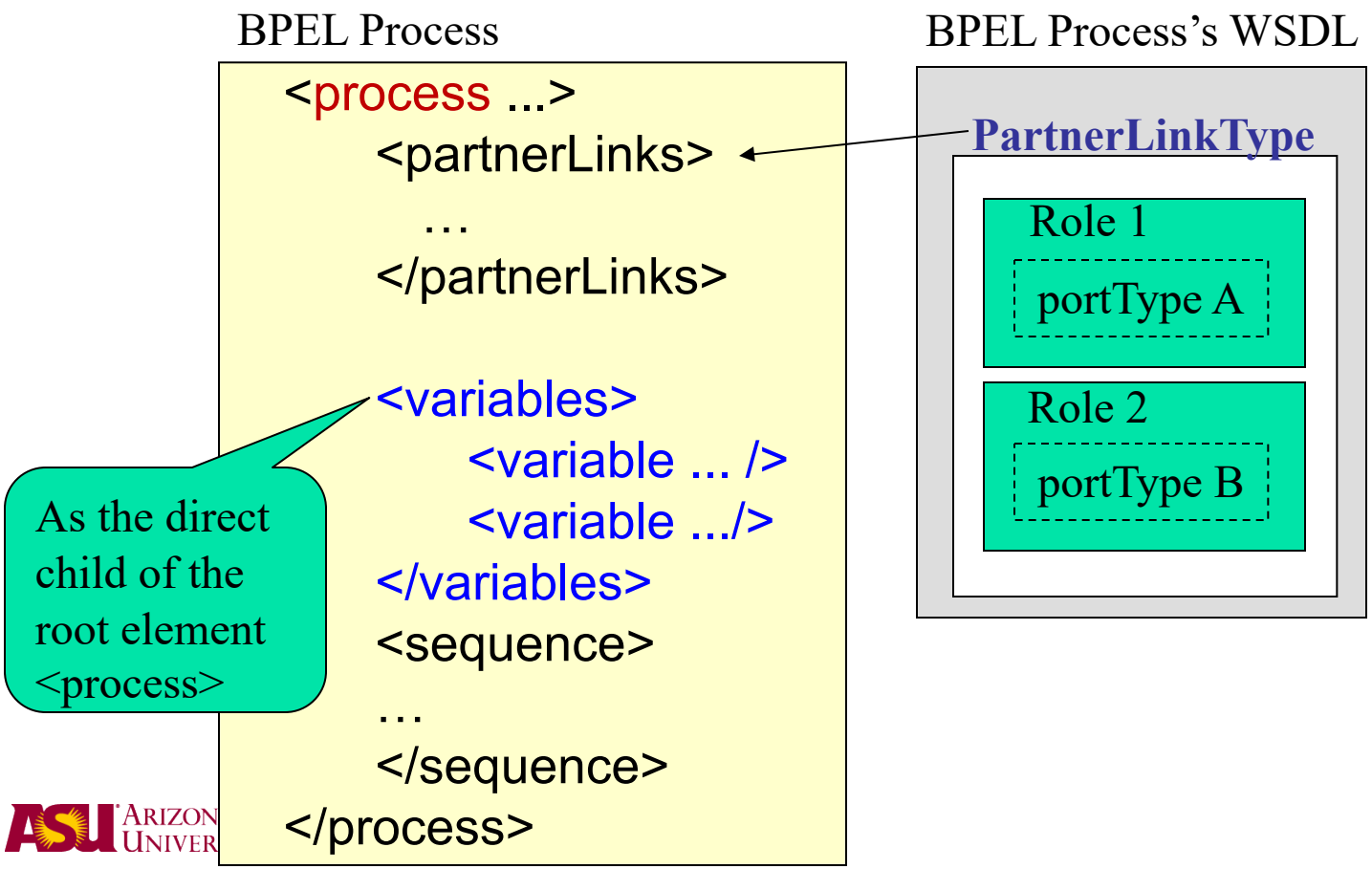
Declaration of Variables

- BPEL Variables are typed. Variables can be declared in one of the following kinds:
  - *messageType*: A variable that can hold a WSDL message;
  - *element*: A variable that can hold an XML Schema element;
  - *type*: A variable that can hold an XML Schema simple type.

```
<variables>
    <variable name = "InsuranceRequest"
                messageType = "ins:InsuranceRequestMessage"/>
    <variable name = "PartialInsuranceDescription"
                element ="ins:InsuranceDescription"/>
    <variable name= "LastName"
                type="xs:string" />
</variables>
```

Global Variables

- You can declare variables globally at the beginning of a BPEL process declaration document.
- The following example shows the structure of a BPEL process that uses variables:



Scope and Local Variables and Handlers

Scopes

- provide a way to divide a complex business process into hierarchically organized parts;
- provide behavioral contexts for activities;
- allow us to define different fault handlers for different activities;
- can be defined within <sequence> or <flow>.
- provide a way to declare local variables that are visible only within the scope.
- allow us to define local correlation sets, compensation handlers, and event handlers

Local variables within the given "scope"

```
<sequence>
    <scope>
        <variables>
            <!-- variable definitions local to the scope -->
        </variables>
        <correlations>
            <!-- correlation sets -->
        </correlations>
        <compensationHandler>
            <!-- Compensation handlers local to the scope -->
        </compensationHandler>
        <eventHandlers>
            <!-- Event handlers local to the scope -->
        </eventHandlers>
    </scope>
    <faultHandlers>
        <!-- Fault handlers local to the scope -->
    </faultHandlers>
    …
</sequence>
```

The scope does not apply to these faultHandlers

Exception Handlers

```
<process>
    <partnerLinks>
        …
    </partnerLinks >
    <variables>
        …
    </variables>

    <faultHandlers>
        <catch ...>
            <!– perform an activity -->
        </catch>
        <catch ...>
            <!– perform an activity -->
        </catch>
        <catchAll>
            <!– perform activity.
                catchAll is optional -->
        </catchAll>
    </faultHandlers>
    …
</process>
```

```
<faultHandlers>
    <catch faultName="trv:TicketNotApproved" >
        <!-- First fault handler -->
        <!-- Perform an activity -->
    </catch>
    <catch faultName="trv:TicketNotApproved"
                    faultvariable="TravelFault" >
        <!-- Second fault handler -->
        <!-- Perform an activity -->
    </catch>

    <catch faultvariable="TravelFault" >
        <!-- Third fault handler -->
        <!-- Perform an activity -->
    </catch>
    <catchAll>
        <!-- Perform an activity -->
    </catchAll>
</faultHandlers>
```