

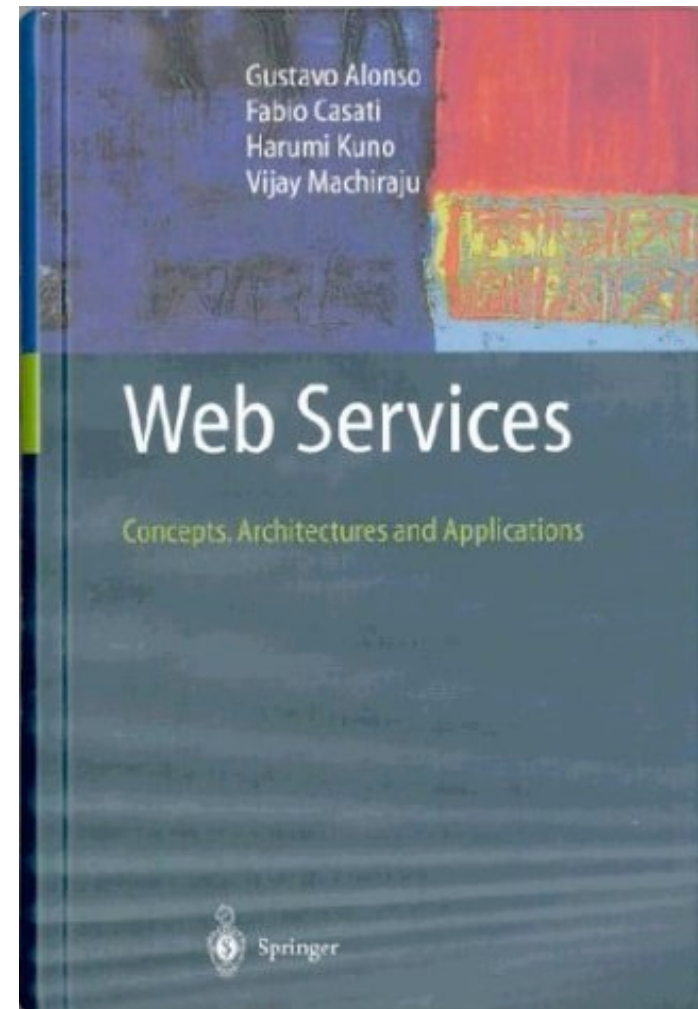
Standard Software for Enterprise Resource Planning

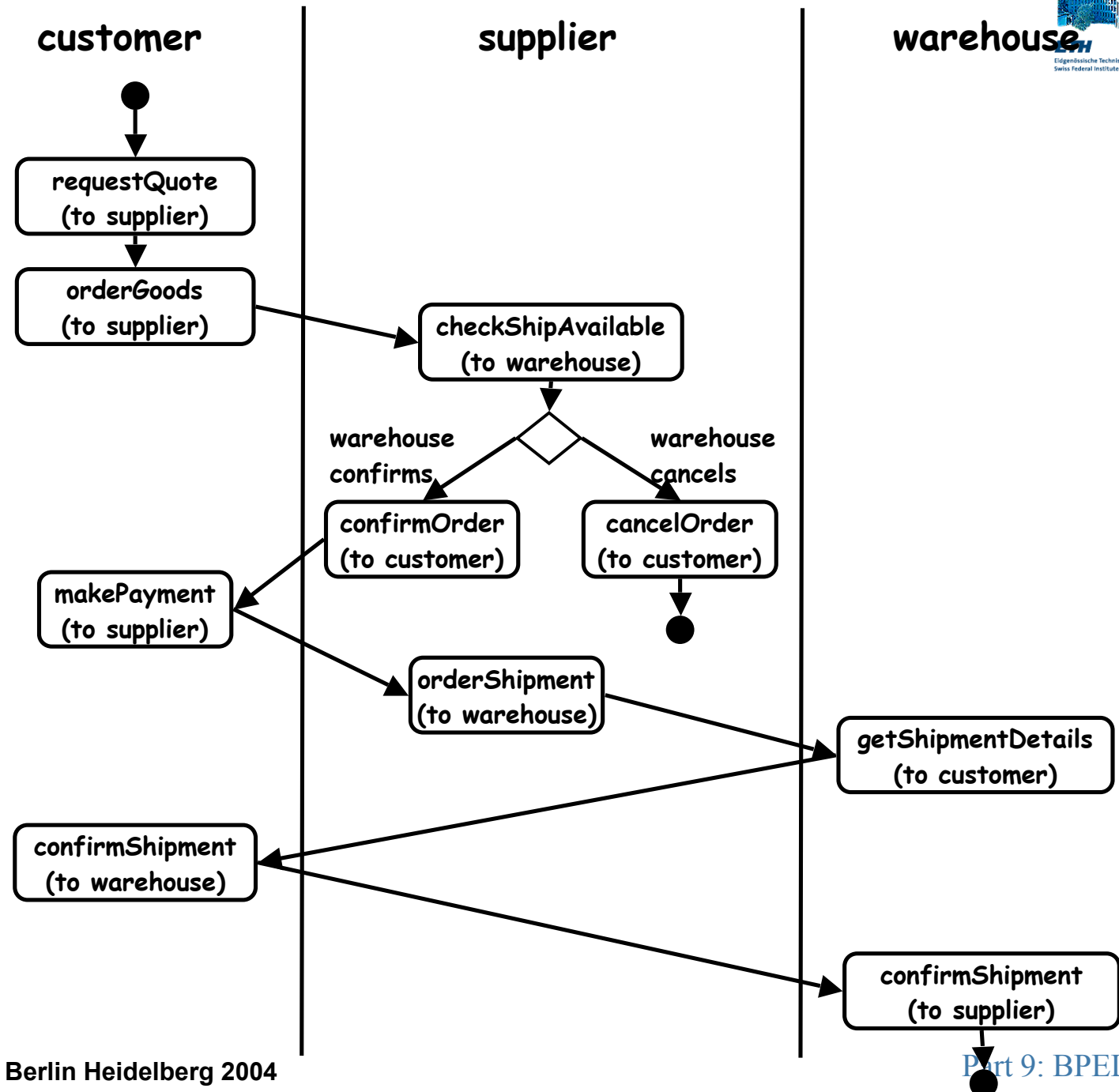
Lecturer: Prof. Dr. Ralf Möller
Lab classes: Rainer Marrone, Michael Wessel

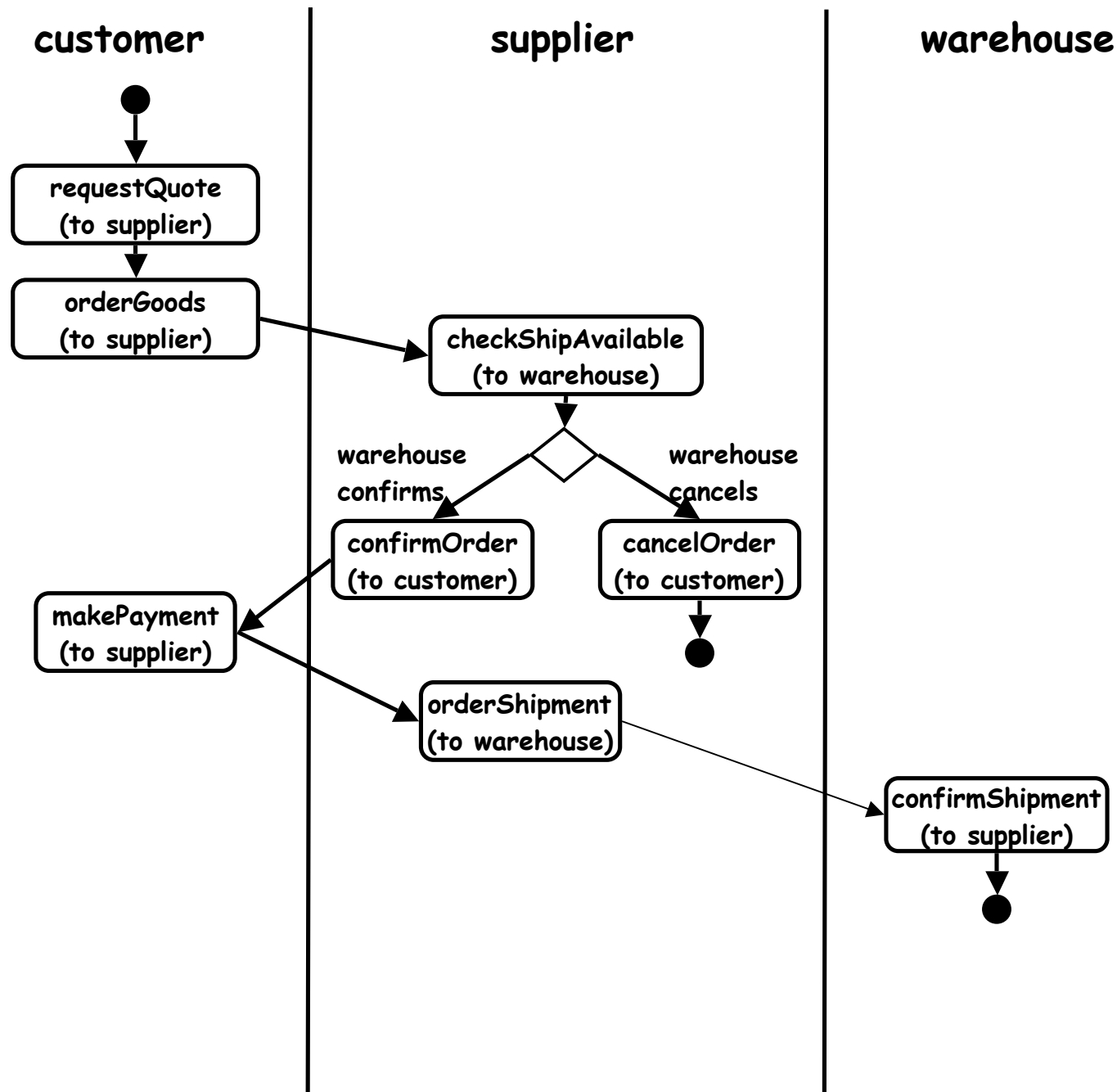
Lecture: Thursdays (90 minutes)
Lab classes: Fridays (60 minutes)

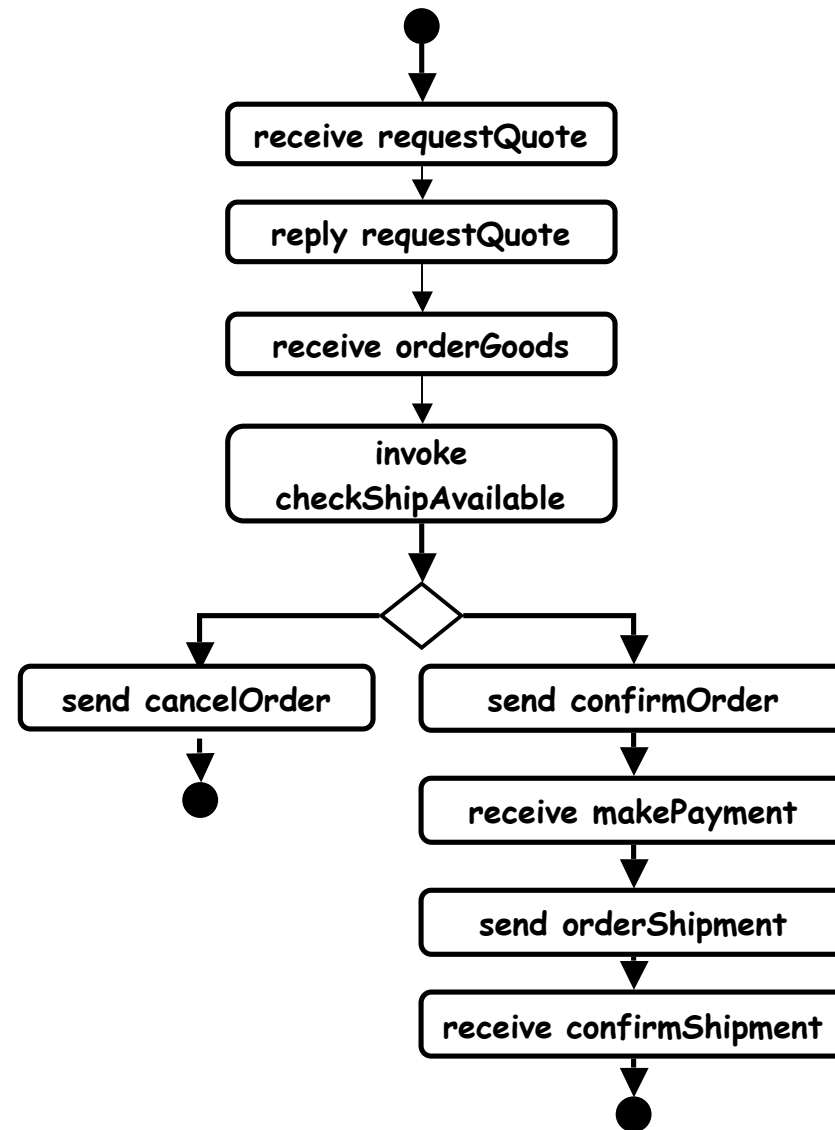
Prerequisite:
Lecture on ECommerce

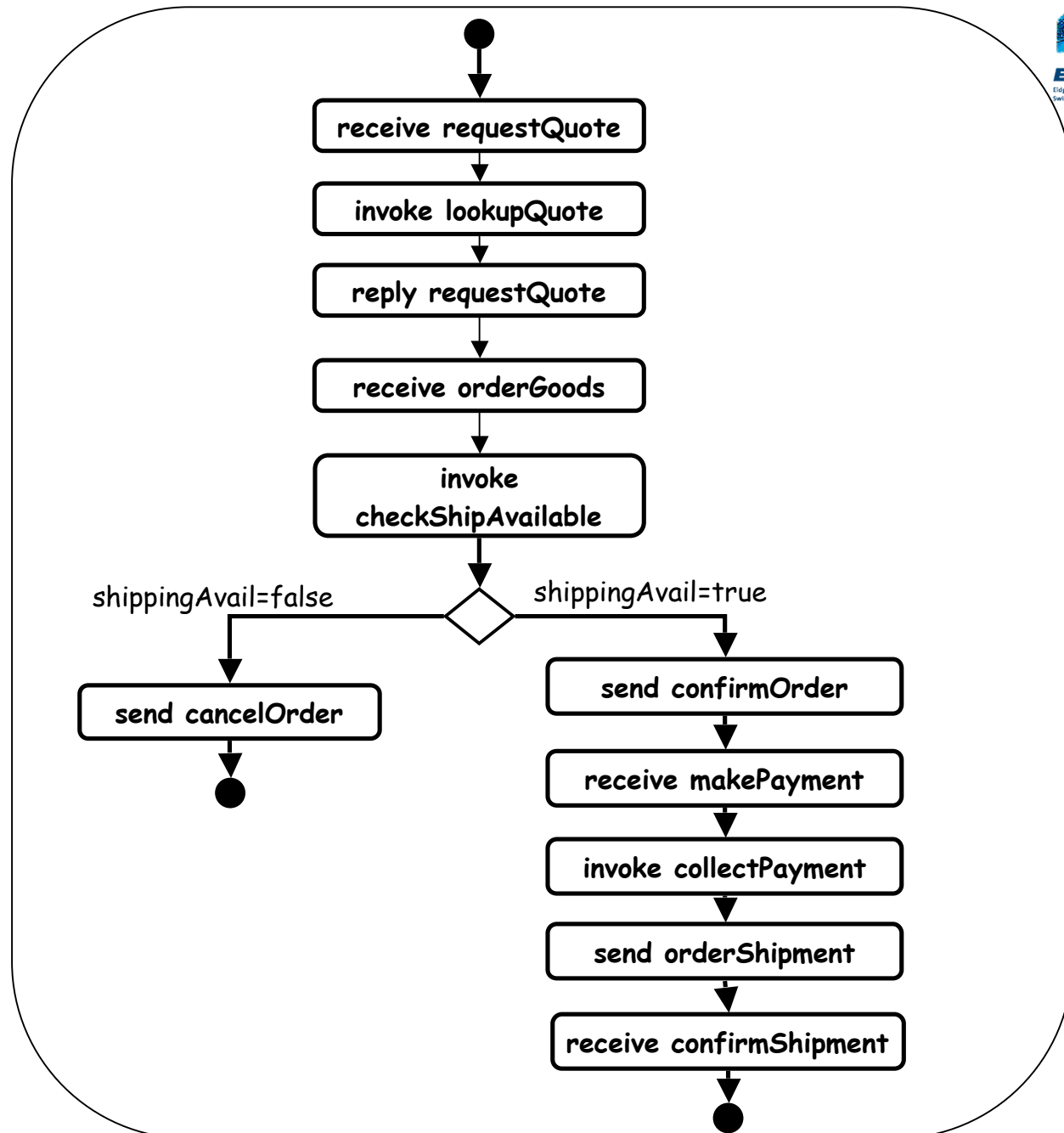
This lecture is based on:



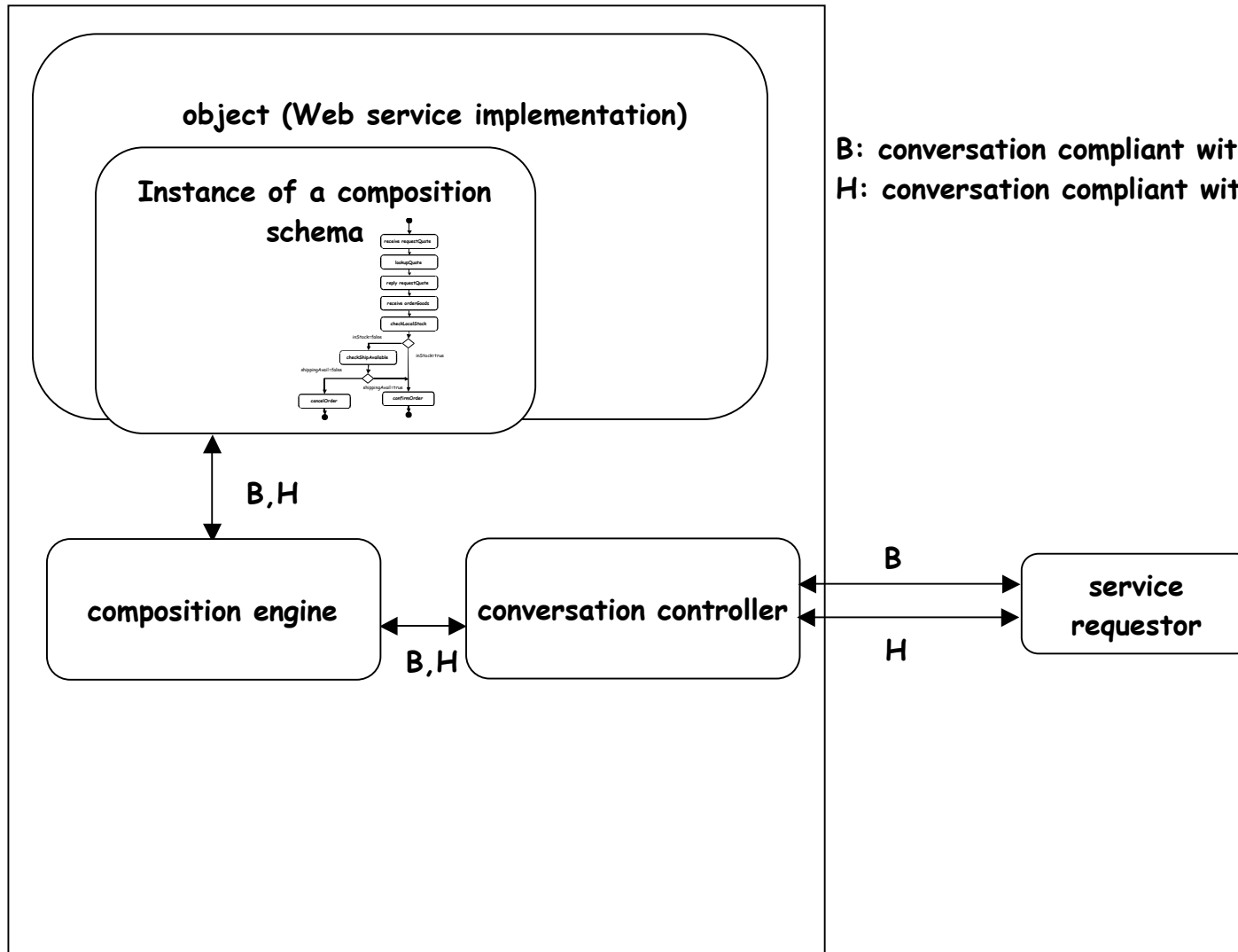








service provider



B: conversation compliant with a business protocol

H: conversation compliant with an horizontal protocol



ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Web Services - Concepts, Architecture and Applications

Part 9: BPEL4WS

Gustavo Alonso, Cesare Pautasso
Computer Science Department
ETH Zürich
alonso@inf.ethz.ch
<http://www.inf.ethz.ch/~alonso>

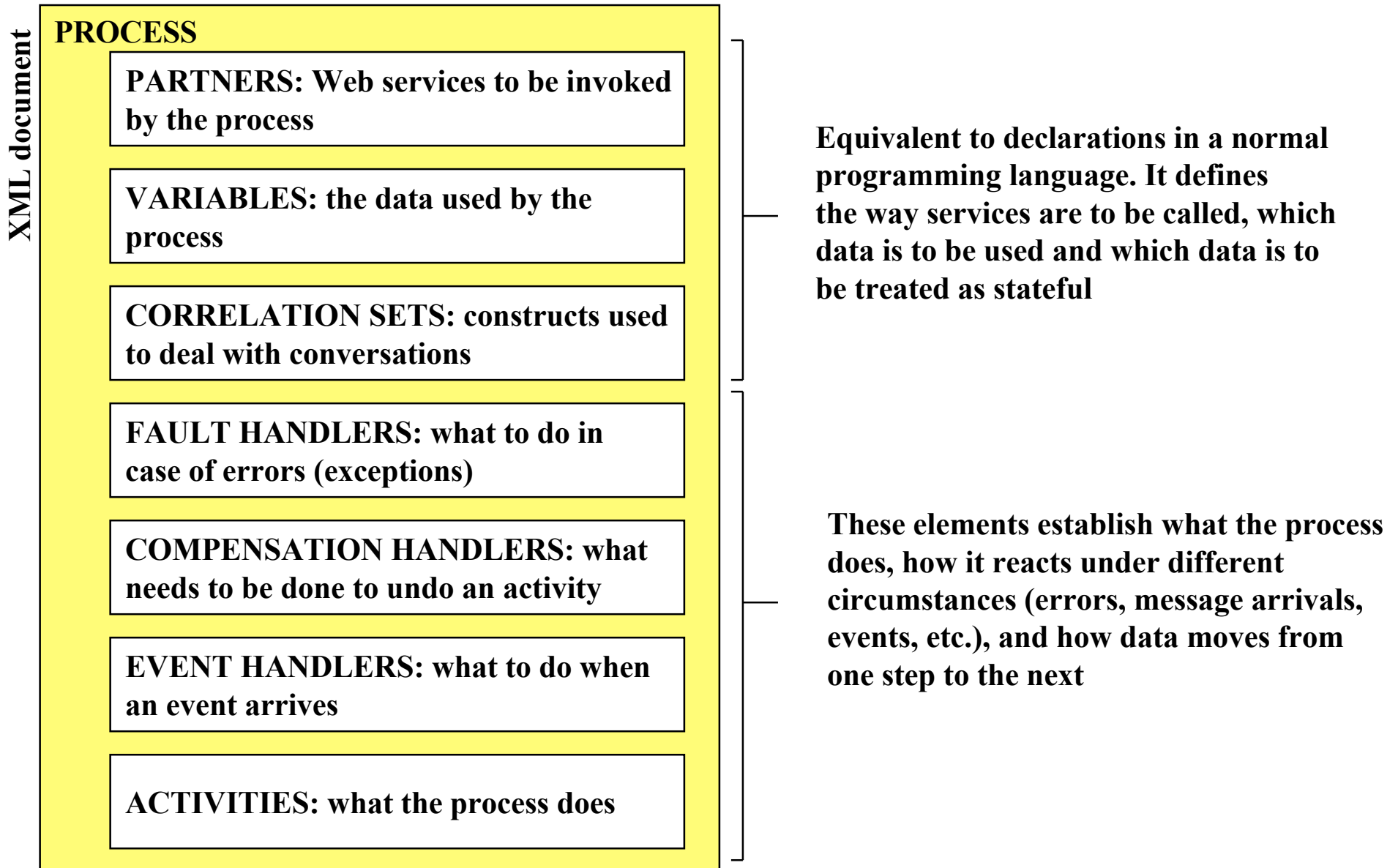
What is BPEL

- ❑ BPEL4WS = Business Process Execution Language for Web Services
- ❑ The latest version of the specification is 1.1
- ❑ Currently, it is the dominant specification for defining composite Web services, conversations and business protocols
- ❑ It does not directly deal with implementation of the language but only with the semantics of the primitives it provides. The emphasis is on interoperability between systems rather than portability of specifications
- ❑ Used to define:
 - Abstract processes: conversations and protocols for how to use a given service or between different services
 - Executable processes: essentially workflows extended with Web service capabilities

Elements of BPEL

- a brief introduction -

Basic elements of BPEL



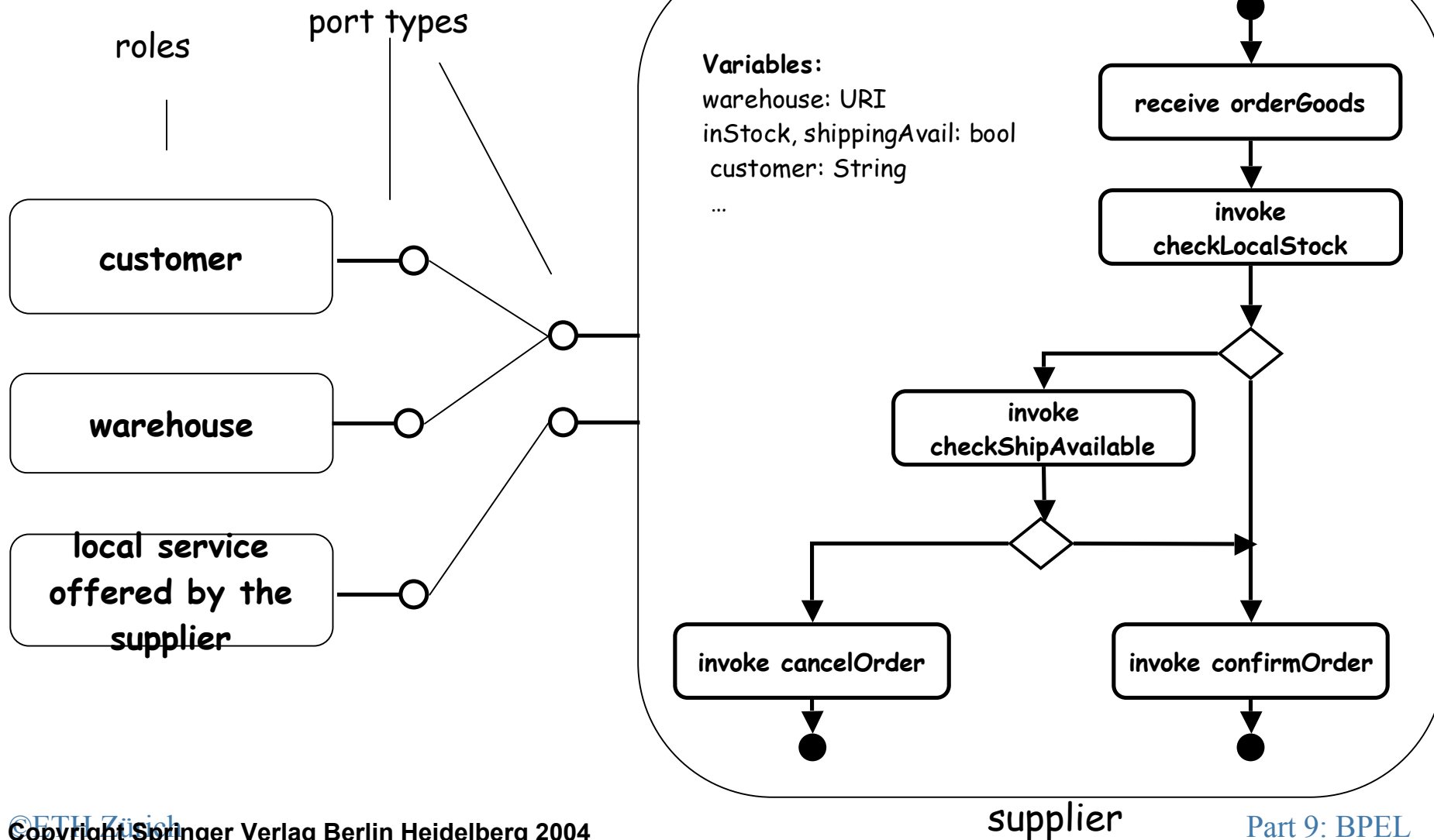
Partners

- ❑ The concept of partners is used to define the Web services that are to be invoked as part of the process. It is based on three elements:
 - Partner Link Type: it contains two PortTypes (see WSDL), one for each of the roles in the partner entry (i.e., one portType is the portType of the process itself, the other one is the portType of the service being invoked).
 - Partner Link: the actual link to the service. This is where the actual assignment to a binding is made (outside the scope of BPEL). Several partner links may share the same partner link type
 - Partners: a group of Partner Links (this is an optional element). A partner link can only appear in one partner.

- ❑ The notion of Partner Link Type reflects a peer-to-peer relation between the process and each one of the services the process calls

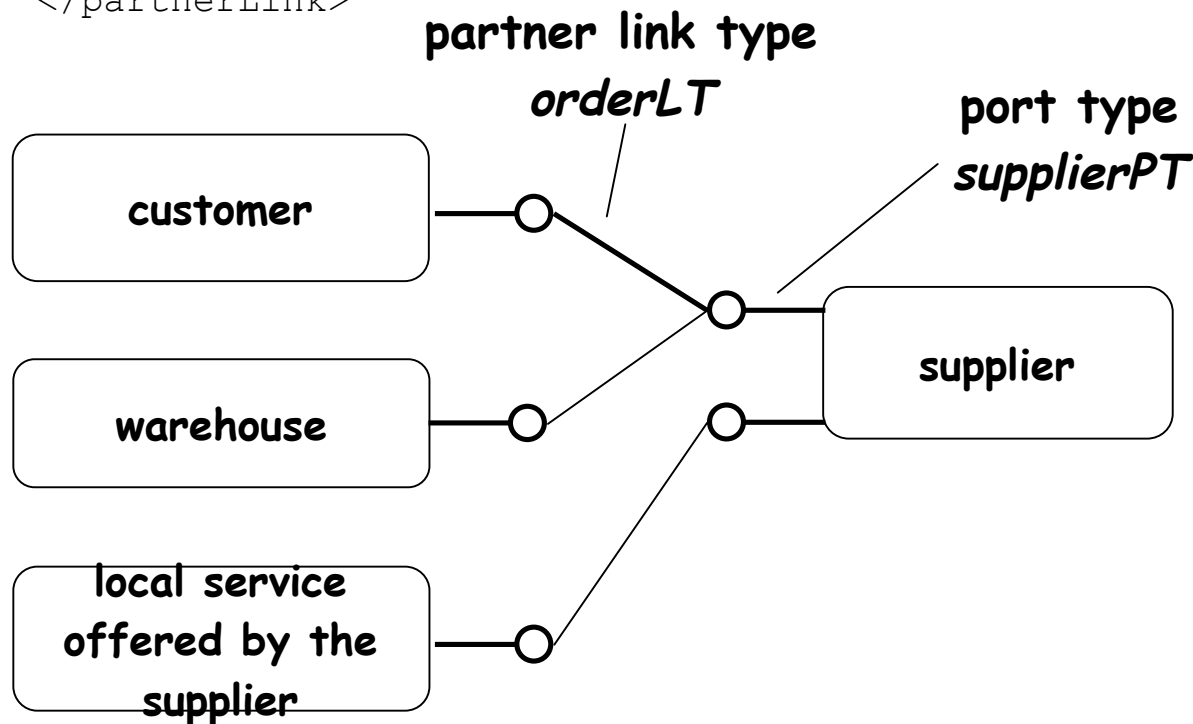


Abstract and/or executable process
orchestration,
variables and data transfers,
exception handling,
correlation information (for instance routing)



partner link definition: it further qualifies the interactions occurring through a partner link type. Its definition refers to a partner link type and specifies the role played by the composite service as well as the one played by the other partner

```
<partnerLink name="customerP"
  partnerLinkType="orderLT"
  myRole="supplier"
  partnerRole="customer">
</partnerLink>
```



Understanding BPEL partners



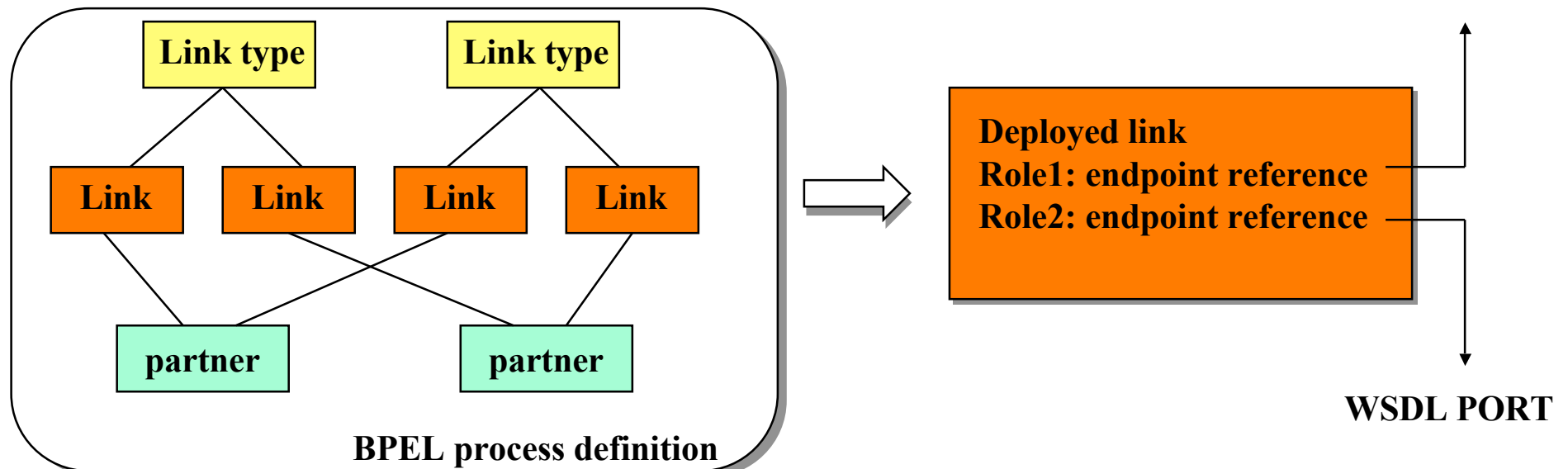
- ❑ In principle, the partner element is used as a vehicle for defining links:
 - Partner Link Type: similar to a class definition in object oriented languages, it is a generic link to a given category of web service (e.g., Bookstore_purchase, Bookstore_search). The partner link type does not really specify the partner but the nature of the link and the roles played by the process and the called Web service
 - Partner Link: the actual instantiation of the link type (i.e., an object of class partner link type). The partner link describes the actual Web service that is going to be called (in an abstract manner since BPEL does not cover binding). Examples: Amazon_purchase (partner link of type Bookstore_purchase) or Barnes&Noble_search (partner link of type Bookstore_search)
 - Partners: this is an optional element that can be used to tie together all the links that correspond to the same physical partner. Example: partner “Amazon” contains partner links “Amazon_purchase” and “Amazon_search”.

Finding the partner information

- ❑ Partner Link Types information can be found in a variety of places:
 - In the WSDL description of the actual service (BPEL specifies how to use WSDL extensibility to define the partner link type as part of the WSDL service description document; the partner link type is part of the abstract service description)
 - In their own WSDL document, independent of the WSDL description of the services involved
 - Partner link types are not included in the BPEL description of the process
- ❑ Partner Links and Partners:
 - Are defined in the BPEL document describing the process

From definition to enactment

- ❑ BPEL does not say anything about how a process is to be executed and how to map a process description to an actual running system. This is the reason why partner link types use only the PortTypes of WSDL descriptions (i.e., contain only abstract information) and not the actual ports for those services (i.e., the binding information that allows contact with and usage of those services)
- ❑ The mapping of a partner link to an actual service takes place through the notion of “Endpoint References”. Every partner role in a link is assigned an endpoint reference at deployment time (this can also happen dynamically for an activity in the process). This endpoint reference is what maps to the actual port.



Variables

- ❑ Variables are used in BPEL to hold data used within the process
- ❑ Variables typically contain two basic forms of data:
 - Entire messages (defined in the WSDL description of a service)
 - Process specific data (counters, state variables, etc.)
- ❑ Variables are defined in the BPEL description of the process without specifying their type (e.g., what message they correspond to). Like partner link types, the idea is that these definitions are to be found in separate WSDL documents or in the WSDL descriptions of the services to be invoked by the process.
- ❑ Variables can be:
 - A message (their type is a WSDL message, which can be found in the WSDL description of the service using that message)
 - An XML type (f.i. integer; typically used for internal operations within the process)
 - An XML element (used to refer to complex XML types)

the orderID can be used
for correlating the two
messages

message checkAvailability

orderID

requestedDeliveryDate

deliveryLocation

...

warehouse

supplier

message availability

orderID

shippingAvail

Correlation Sets

- ❑ Correlation is an advanced feature of BPEL that is intended to help in mapping an abstract specification (which is what BPEL supports) to running instances of that specification
- ❑ The problem: an abstract process describes what to do in general, while each running instance of the process must work only on its own data (f.i. on the messages that correspond to a particular purchase order). The correlation problem is how to specify in BPEL the way each running instance can identify the messages it has to process according to that abstract description of the process.
- ❑ Correlation sets are mappings between data, messages and properties that help a process instance identify the messages that are intended for itself. Properties are additional WSDL specifications that give a unique name to parts of a message. These names are then used to identify data that is used to route messages, f.i. a customer id, or a social security number
- ❑ Correlation sets can be rather complex but their basic use is simple: assign a name to a part of a message (f.i. a purchase order number) by defining a property, and then look for that property in all messages to identify the ones relevant to the process

Understanding correlation sets

- ❑ Correlation sets are a controversial part of BPEL since they are very much implementation specific, while BPEL does not get into implementation details
- ❑ Correlation sets are included into BPEL because of the predominant bottom-up approach followed by the designers of the specification:
 - BPEL assumes only SOAP for message exchanges
 - BPEL assumes the process is responsible for its own state
 - BPEL has many implicit assumptions about how it should be implemented (some of them are rather questionable)
- ❑ Correlation sets are not needed in many other systems (and it is not clear that even vendors supporting BPEL follow the execution model implicit in correlation sets):
 - ebXML assumes there is a messaging system already capable of identifying messages and sorting them according to the intended destination process instance
 - Most middleware solutions (rather than direct compilation to Java, as most current implementations do) would not use correlation sets in the way BPEL describes them
 - The problem can be solved in a much easier manner than through correlation sets (f.i. using a case identification number generated at the start of the process)

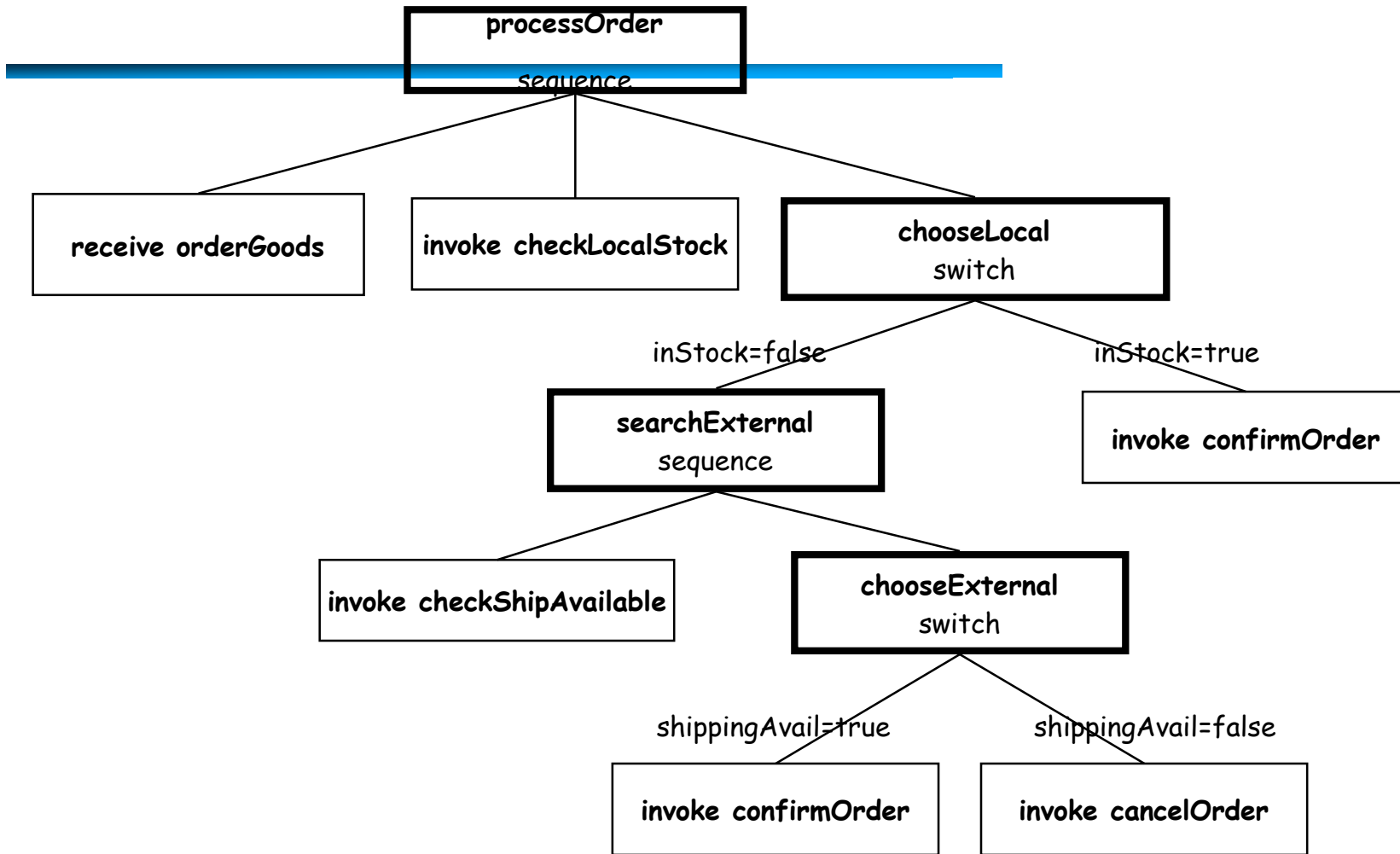
- ❑ BPEL has three types of handlers:
 - Fault handlers (executed when an exception is thrown)
 - Compensation handlers (that are executed to undo the effects of operations)
 - Event handlers (executed when a particular message arrives or an alarm is raised)
- ❑ All handlers must be associated with a scope (similar to a code block in programming languages):
 - If an exception is raised within a given scope, then the corresponding fault handler for that exception is called (identical to try-catch statements in Java)
 - If the effects of an scope need to be undone, then the corresponding compensation handler is called. If there are nested scopes, then the compensation handlers of the low level scopes are called in reverse order of execution
 - Event handlers are active for as long as the control flow remains within the corresponding scope. Event handlers are executed either when a message arrives or a given alarm condition (f.i. a timer) is raised

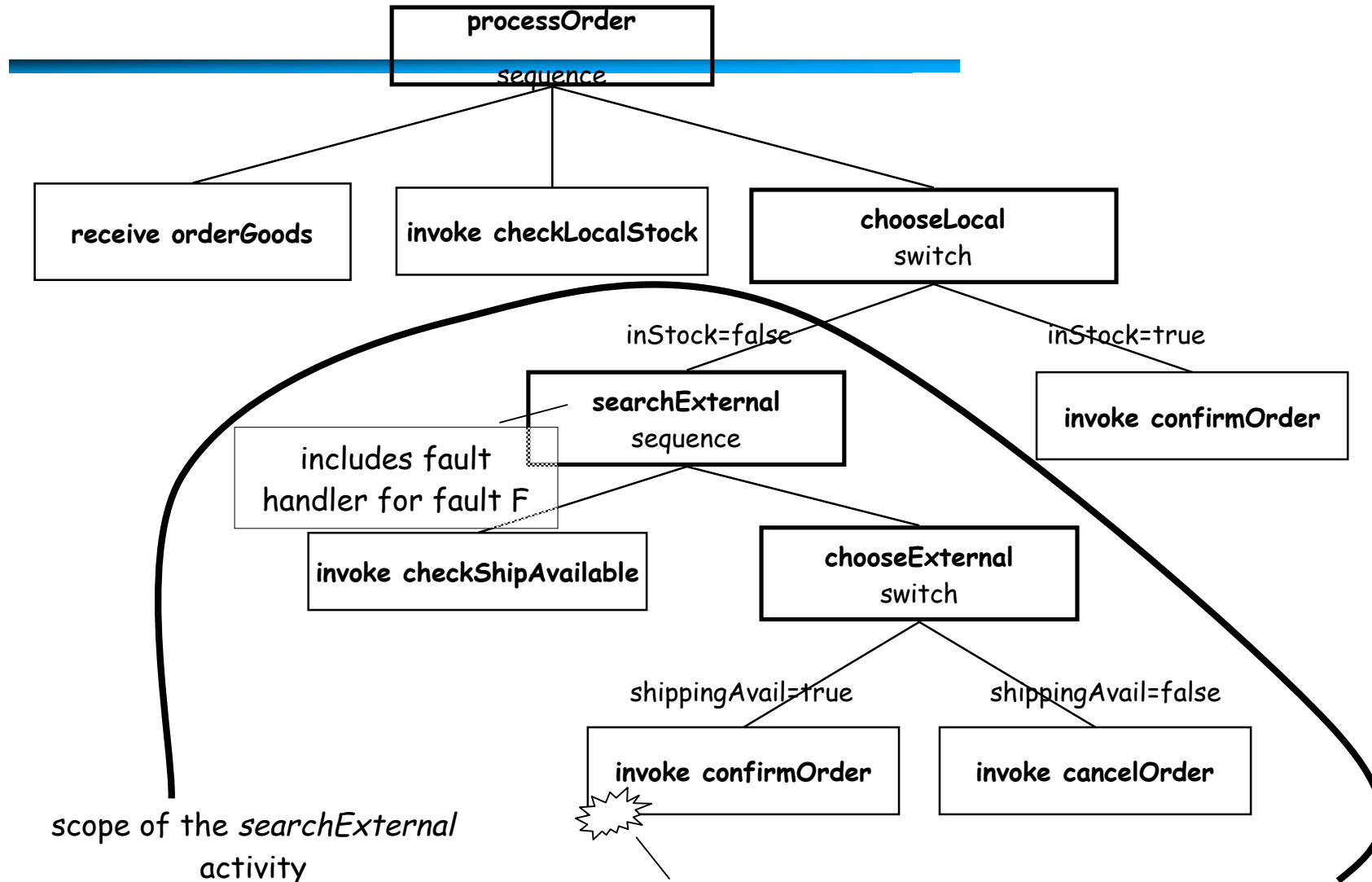
Understanding handlers

- ❑ Handlers in BPEL are intended to cover many features, not all of them equally relevant or important:
 - Fault handlers are a basic component for fault tolerance and fulfil the same role as exception handlers in normal programming languages. Every process needs to have this type of handlers
 - Compensation handlers are a legacy of advanced transaction models and map very well to the notion of transactional coordination espoused by WS-Coordination and WS-Transactions. Compensation handlers assume the process contains the entire logic of what is to be done, something that it is rarely true in complex processes
 - Event handlers fulfill two roles:
 - Dealing with initialisation (the process starts upon the arrival of a message that triggers the corresponding handler). The problem is that it is not clear whether this should be part of the process itself (similar to correlations sets)
 - Dealing with asynchronous messages (in the middle of the process, the customer sends a cancellation message). This is a very useful feature for implementing realistic processes but the problem is typically the implementation of asynchronous behaviour, rather than its specification

Activities

- Activities are the actual operations the process will complete:
 - <receive> blocks until a message is received
 - <reply> sends a message in response to a *received* message
 - <invoke> sends a message to invoke an operation on a remote service
 - <assign> updates the value of variables
 - <throw> raises a fault for a fault handler to catch
 - <terminate> finishes the process
 - <wait> suspends execution for a given time period
 - <empty> no-op used for synchronization purposes
 - <scope> defines a block of activities
 - <sequence> executes a set of activities one after another
 - <flow> executes in parallel a set of activities
 - <while> repeats an activity depending on certain conditions
 - <switch> chooses between a set of activities
 - <pick> waits for a message or an alarm
 - <compensate> defines the activities of a compensation block





due to the behavior of the default handler, implicitly associated with each activity, a fault F occurring in activity *send confirmOrder* would propagate up until activity *searchExternal*, where the handler resides

More on activities

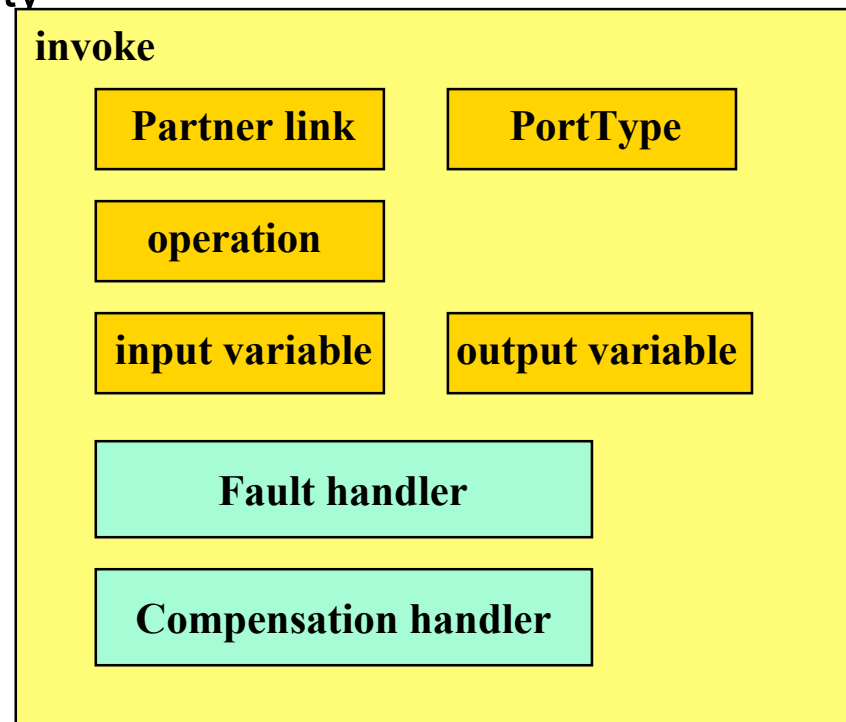
- ❑ Activities in BPEL can be seen as nodes in a graph where implicit or explicit links determine the control flow between activities
- ❑ Activities are of three types:
 - Web service related (receive, reply, invoke)
 - Control flow commands (throw, terminate, compensate, etc.)
 - Data manipulation (assign)
- ❑ Activities have two optional elements used to establish synchronization dependencies through links in the flow construct:
 - Source: the activity is the source of this link
 - Target: the activity is the target of this link
- ❑ Activities have three attributes:
 - Name of the activity
 - JoinCondition: used in flow constructs
 - SuppressJoinCondition: used in flow constructs

Understanding activities

- ❑ BPEL is very close to conventional workflow management systems and follows a similar modeling approach
- ❑ BPEL is also implicitly very close to Java: existing implementations compile BPEL to Java code
- ❑ The activities in BPEL have been designed to allow for almost any form of process modeling that one can imagine and to facilitate the mapping of graphical workflow modeling tools to BPEL
 - Structured workflows with explicit control flow activities map directly to BPEL constructs
 - Unstructured workflows with activity graphs can be modelled with the flow construct
- ❑ BPEL also includes constructs for handling asynchronous activities (a feature missing in most workflow systems) and time events
 - Event handlers are used in both cases
- ❑ The goal is to have tools that translate graphical representations into BPEL (as it is not feasible to program BPEL directly in XML)

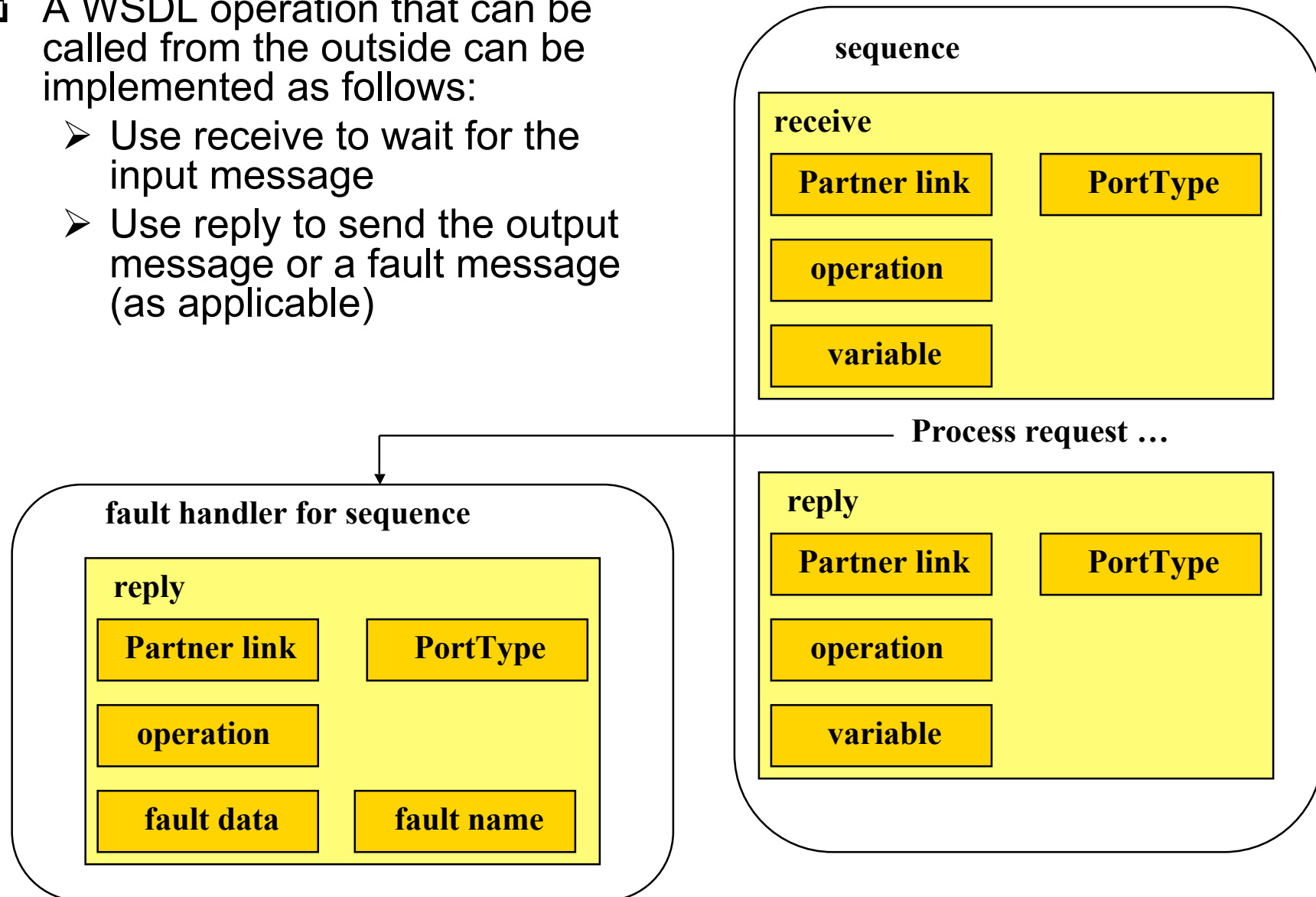
WSDL operations in BPEL

- A call to an operation in WSDL can be mapped to BPEL as follows:
 - Use invoke to call the operation
 - An input variable with the request (the input message of the WSDL operation)
 - An output variable for the response (the output message of the WSDL operation)
 - A WSDL fault can be handled by using a fault handler attached to the invoke activity



WSDL operations in BPEL

- ❑ A WSDL operation that can be called from the outside can be implemented as follows:
 - Use receive to wait for the input message
 - Use reply to send the output message or a fault message (as applicable)



Some available implementations



- There are several implementations of BPEL already available:
 - Collaxa: <http://www.collaxa.com/>
 - IBM: <http://www.alphaworks.ibm.com/tech/bpws4j>