Modeling Business Processes with BPMN

DIPARTIMENTO DI INFORMATICA
E SISTEMISTICA ANTONIO RUBERTI

SAPIENZA
UNIVERSITÀ DI ROMA

Andrea Marrella

`marrella@dis.uniroma1.it`

Presentation Outline

- This seminar introduces **business process modeling** using the **BPMN** (*Business Process Model and Notation*) standard.

- Visit http://www.bpmn.org/ for downloading the complete specification of BPMN 2.0 and some interesting examples.

- Web site of the course : http://www.dis.uniroma1.it/~bpm

- Download slides at :

    http://www.dis.uniroma1.it/~marrella/teaching.html

- <u>Reference Books</u> :

    Marlon Dumas, Marcello La Rosa, Jan Mendling, Hajo Reijers
    Fundamentals of Business Process Management

    Stephen A. White PhD, Derek Miers
    BPMN Modeling and Reference Guide

    Thomas Allweyer
    BPMN 2.0 - Introduction to the Standard for Business Process Modeling

2

Topics

- **Process Modeling**
- BPMN Background
- Basic Concepts
- Advanced Concepts
- Conclusions

Business Processes

- **Business Process [1] :** "*A business process consists of a set of activities that are performed in coordination in an organizational and technical environment. These activities jointly realize a business goal*"

  - A **business goal** is the target that an organization aims to achieve by performing correctly the related business process.

- Currently, business processes are the core of most information systems:

  - production line of a car manufacturer, procedures for buying tickets on-line…

- This requires that organizations specify their **flows of work** for the **orchestration** of participants, information and technology for the realization of products and services**.**

- In order to manage Business Processes, they have to be described and documented in terms of **process models**.

A first BPMN Example

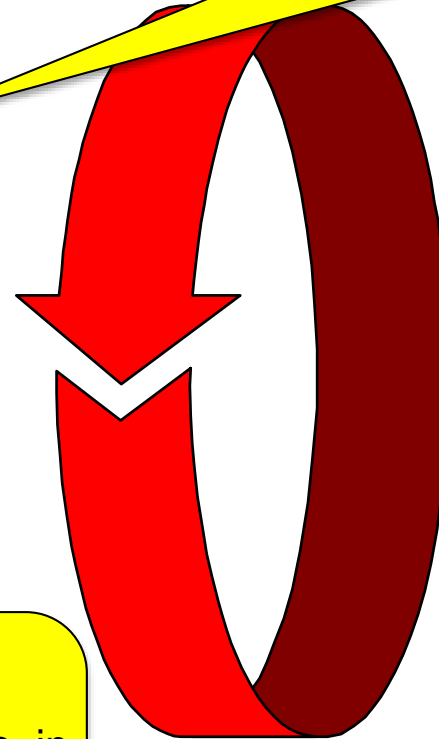Business process concerning the book loans service provided by a library.
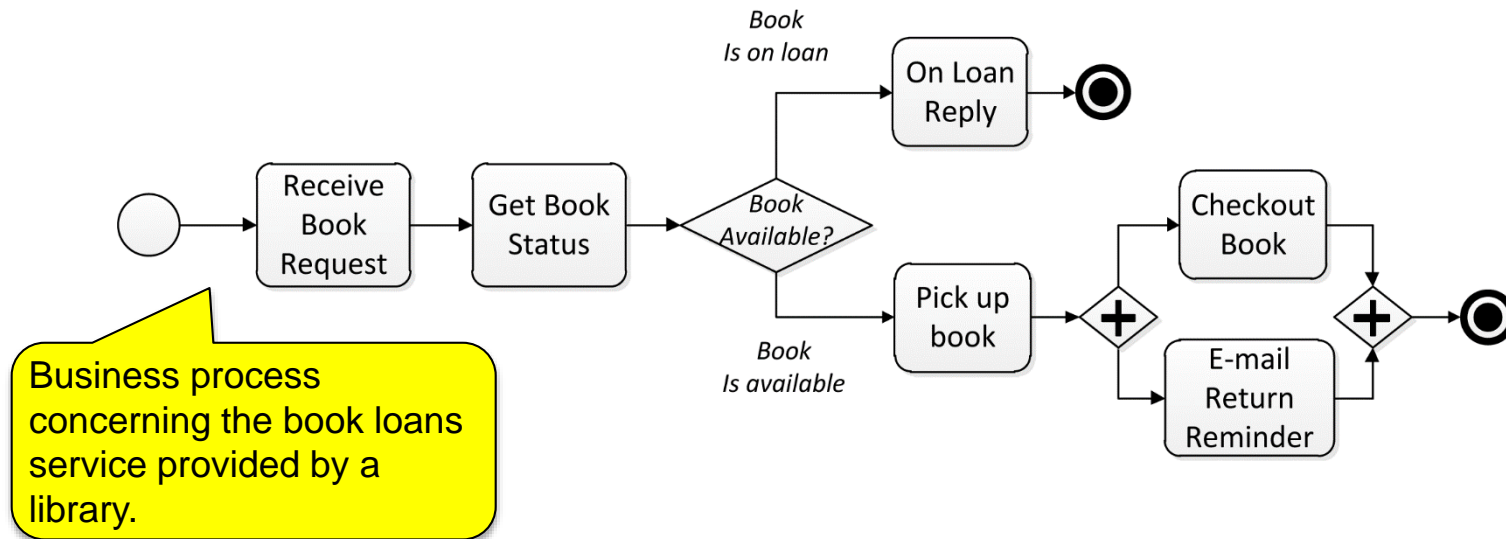
- A notation for business process modeling defines the **symbols** for the various process elements, their correct **meaning** as well as their possible **combinations**.
- Thus, a **notation** is a **standardized language** for the description of business processes.

The Process Life-Cycle

1. Process identification
2. **Process discovery (as-is)**
3. Process analysis
4. Process redesign (to-be)
5. Process implementation
6. Process monitoring

Process Modeling Tools

Process Management Systems

The purpose of Process Discovery is to capture the main components of a process (how the things have to be done) and to **document a process** with manuals, policies, procedures.

After having documented the process, there is the need to capture **its essence**, in the form a business process diagram (process model), and this can be done through the **Process Modeling** activity.
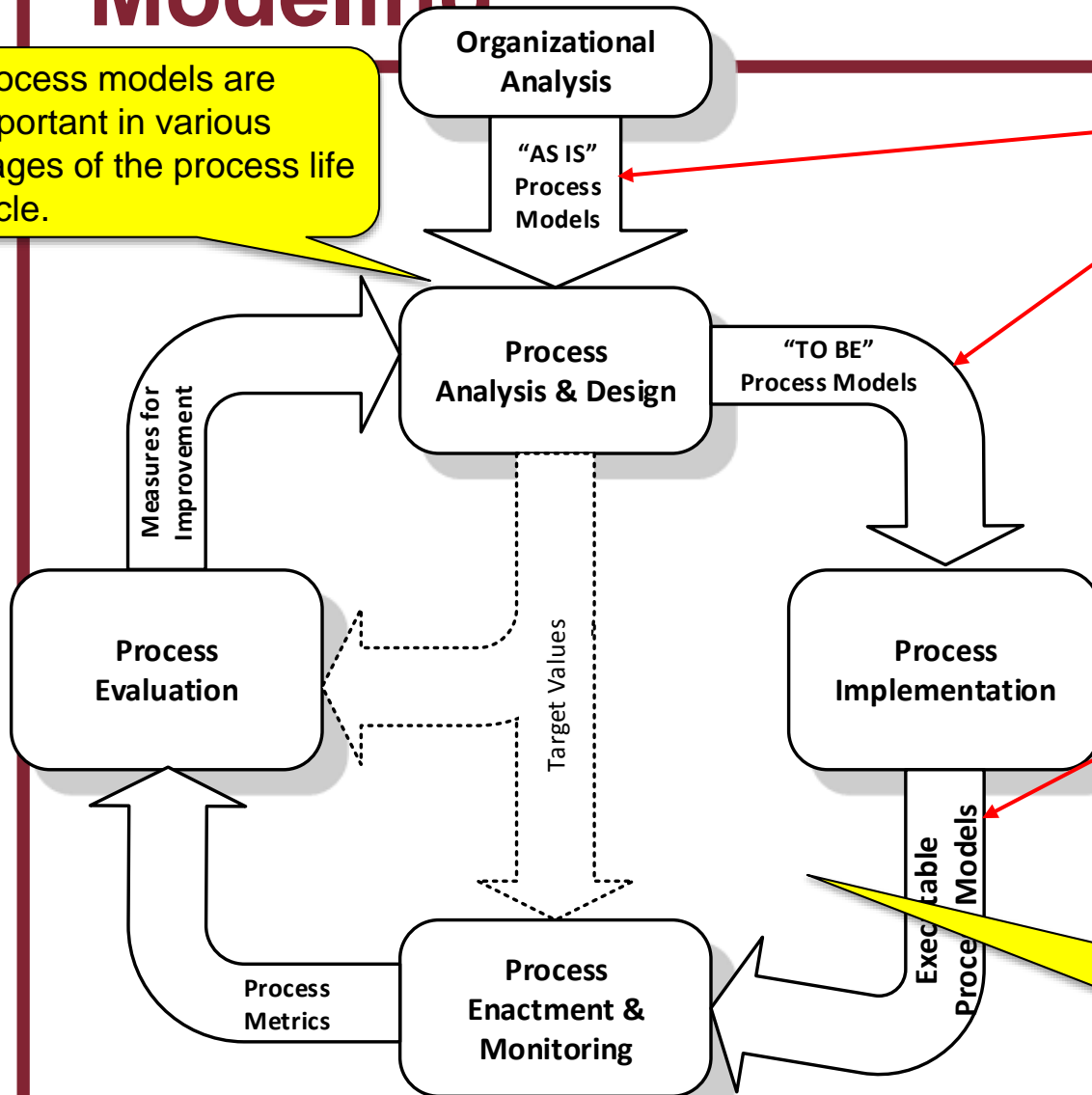
Purposes of Process Modeling

Organizational Analysis

"AS IS" Process Models

Process Analysis & Design

"TO BE" Process Models

Measures for Improvement

Process Evaluation

Target Values

Process Implementation

Executable Process Models

Process Enactment & Monitoring

Process Metrics

High-level Models including

Communication, simulation, activity-based costing…

Detailed Models including

Data types, conditions, data mappings, fault handling…

Integration, testing, deployment…

Process models are important in various stages of the process life cycle.

The models produced will look quite differently depending on the reason for modeling them in the first place.
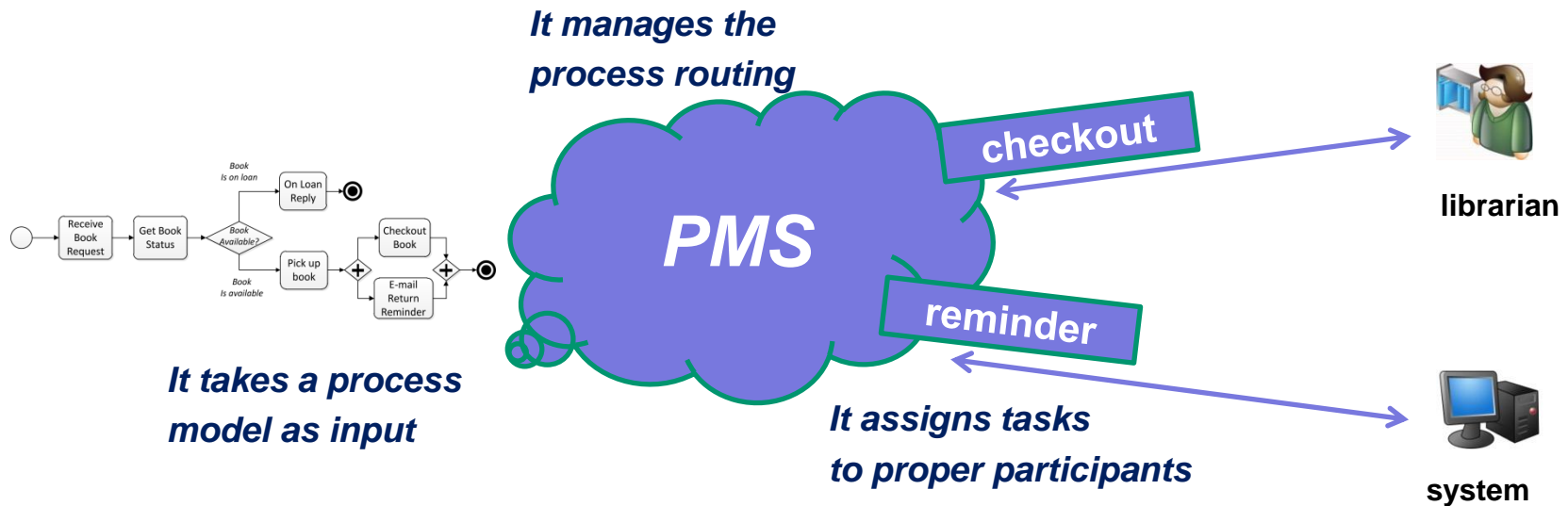
Executable Process Models

- Executable Process Models carry the instructions on how work should happen, who should do it, links to the other systems, etc.

- They provide a ***direct method of translating strategical and tactical intent into operational processes***.

- For being executed, process models have to meet very strict demands, because they are not converted into a computer program by a human being, but **directly processed by a machine**.

- Currently, many standards for executable process descriptions have been established, such as:
    – **XPDL** (XML Process Definition Language) [2]
    – **BPEL** (Business Process Execution Language) [3]

    …but such descriptions have no graphical notations, and the main range of application is the definition of automatic processes.

- YAWL (Yet Another Workflow Language) [7] is a modeling language that allows for a graphical representation of executable process models.

Process Management Systems

- **Process Management System (PMS) [1] :** "*A PMS is a generic software system that is driven by explicit process representations to coordinate the enactment of business processes* ".
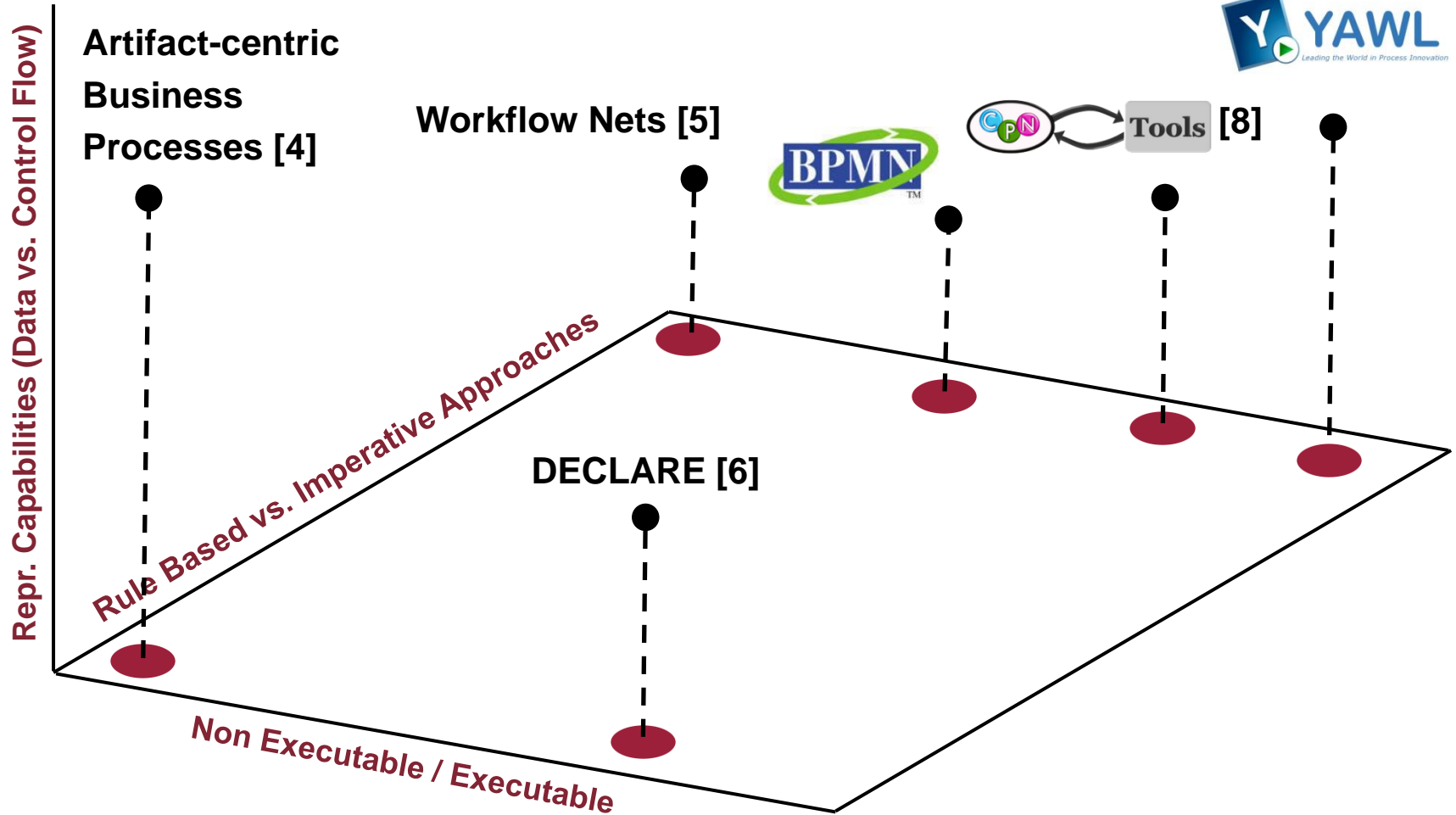

It manages the process routing

PMS

checkout

librarian

reminder

system

It takes a process model as input

It assigns tasks to proper participants

 – <u>A PMS is driven by a specific business process model</u>.

The Importance of Process Modeling

- To model a process leads to a number of questions :
    - Which steps are really necessary?
    - Who should do them?
    - Should they be kept in house or outsourced?
    - How they should be done?
    - What capabilities are needed?
    - What results do we expect and how will they be monitored?

- While the answers to these questions are always situation-specific…

- …without the backdrop of a **commonly agreed description** of the business process in question, such answers are often *vague* and *wooly*.

Modeling Languages for Business Processes

Repr. Capabilities (Data vs. Control Flow)

Artifact-centric Business Processes [4]

Workflow Nets [5]

YAWL **[7]**

Tools **[8]**

BPMN

Rule Based vs. Imperative Approaches

DECLARE [6]

Non Executable / Executable

Topics

- Process Modeling
- **BPMN Background**
- Basic Concepts
- Advanced Concepts
- Conclusions

Business Process Modeling Notation(BPMN)

- BPMN is the OMG standard for representing business processes.
  - Other standards, such as activity diagrams of UML, were not accepted for process modeling in practice, because their use is restricted to the area of object-oriented software design.

- There are many tools for designing a BPMN process:
  - **Bizagi Process Modeller** (it also provides an execution engine)
  - JBPM (Eclipse plugin)
  - Signavio
  - TIBCO Business Studio (free download, quite large)
  - IBM Websphere Business Modeler
  - ARIS
  - Oracle BPA
  - Business Process Visual Architect (Visual Paradigm)
  - Progress Savvion Business Modeller

Why BPMN?

- *"The primary goal of BPMN is to provide a notation that is **readily understandable by all business users**, from the business analysts that create the initial draft of the processes, to the technical developers responsible for implementing the technology that will perform these processes, and finally to the business people who will manage and monitor those processes."*

- *"The idea is to create a **standardized bridge for the gap between the business process design and process implementation**."* [BPMN 2.0 spec.]

The history of BPMN

The vendors realized immediately there was a need of a graphical representation for the language **oriented towards the needs of business users**. Not a notation that directly represents the precise execution language under development.

| 1994 | 1998 | 2000 | 2001 | 2002 | 2003 | 2004 | 2005 | 2006 | | 008 | 2009 | 2010 |

WfMC

Ref Model

WPDL → XPDL 1.0 → XPDL 2.0 → XPDL 2.1 → XPDL 2.2

BPMI

BPMN 1.0

XML

OMG

BPMN 1.0 → BPMN 1.1 → BPMN 1.2 → BPMN 2.0
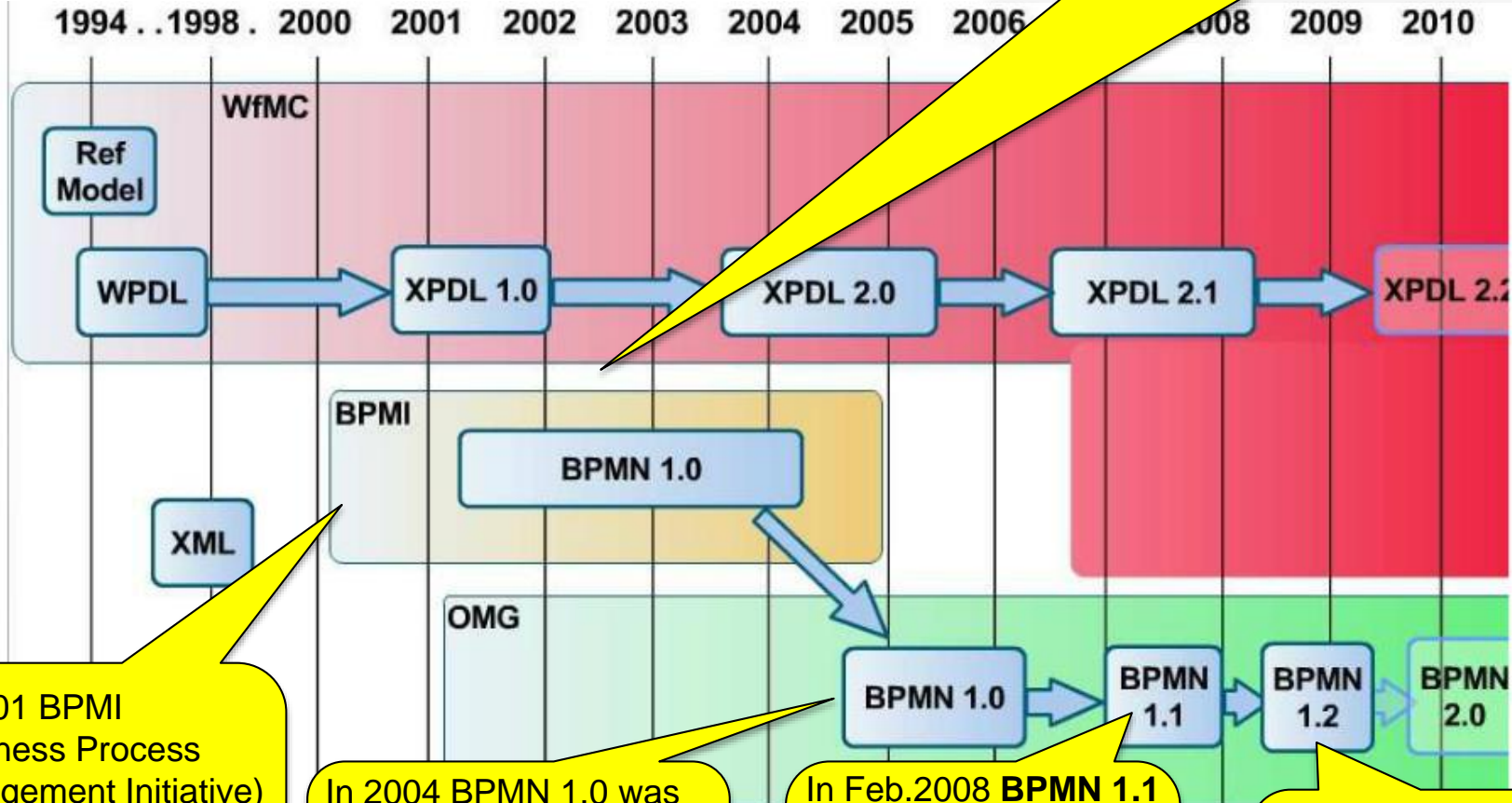
In 2001 BPMI (Business Process Management Initiative) developed BPML as an XML process execution language.

In 2004 BPMN 1.0 was released to the public and **in 2006 it was adopted as OMG standard**.

In Feb.2008 **BPMN 1.1** was released to the public, making the **meaning of the notation more explicit**.

BPMN 1.2 does not include any significant graphical changes; modifications were merely editorial.

BPMN 1.2 vs BPMN 2.0

- **BPMN 1.2** provides a mapping from a "valid" BPMN diagram to BPEL, such that an engine can execute the process.
  - The 1.2 specification provides only **contained verbal descriptions** of the graphic notations elements and modeling rules. This leads to misleading and confusions in the translation process.
- **BPMN 2.0 beta 2** was introduced in June 2010.
  - It represents the biggest revision of BPMN since its inception.
- BPMN 2.0 received a **formal definition** in the form of a *metamodel*, that is a precise definition of the constructs and rules needed for creating specific models.

What is a Metamodel?

- The BPMN 2.0 metamodel defines all BPMN entities with all their attributes and relations.
  - Entities could be visual shapes like activities or gateways, but also invisible stuff, such as web service operations or data structures.
- BPMN 2.0 introduces a **MOF-based metamodel** that defines the abstract syntax and semantics of the modeling constructs.
- The metamodel enables interchange, interoperability and execution of models

MOF-based metamodel
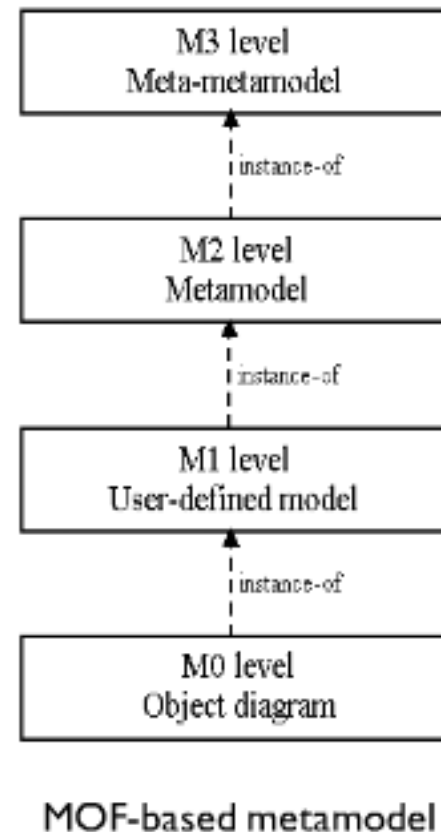
BPMN 2.0 Metamodel

- Metamodelling provides a number of benefits :
  - It formalizes the definition of models and entities.
  - It formalizes the relationship between elements.
  - It enables interoperability.
- The new version's specification document has got comprehensive **UML class diagram** that graphically show the features of the different BPMN constructs and their relationships.
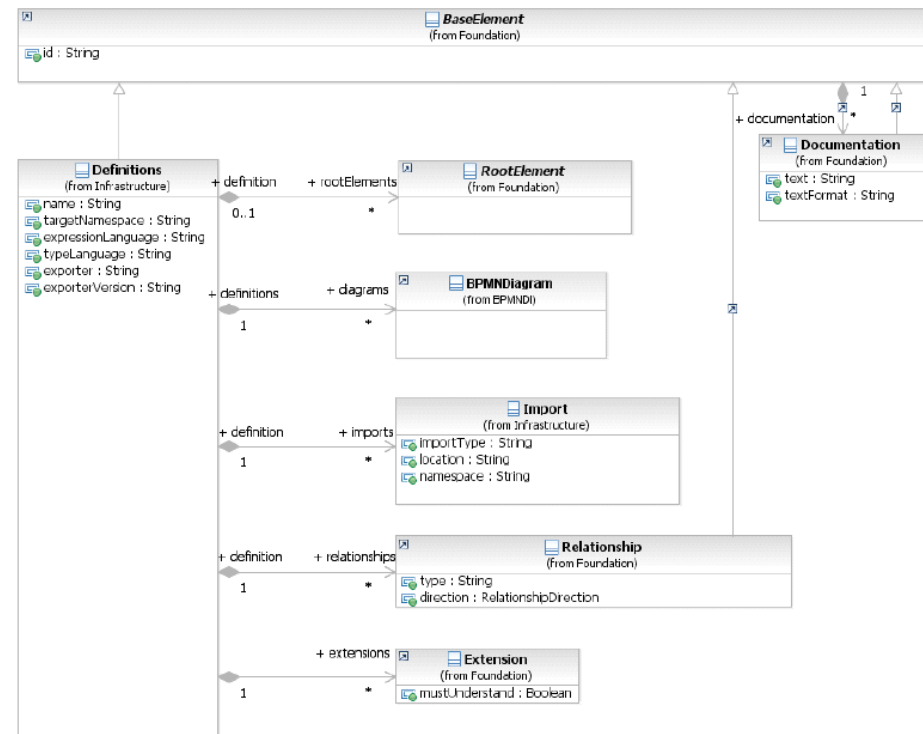
BPMN 1.2 vs BPMN 2.0

- The metamodel also has got **additional language constructs** that cannot be represented in the graphic models.

  – Such constructs are required by process engines to capture the necessary additional information for process execution.

- Moreover, the metamodel was the basis for the development on an **exchange format for BPMN models**.

  – Up to now, it was almost impossible to transfer BPMN models from one tool into another.

  – Some tools have got import and export interfaces for the exchange of BPMN models by means of the XPDL format, but the use of XPDL for this purpose is not widely accepted yet.

  – Moreover, XPDL has not been implemented uniformly by all vendors, so that in practice there are quite often problems with model exchange.

Current BPMN 1.2 problems
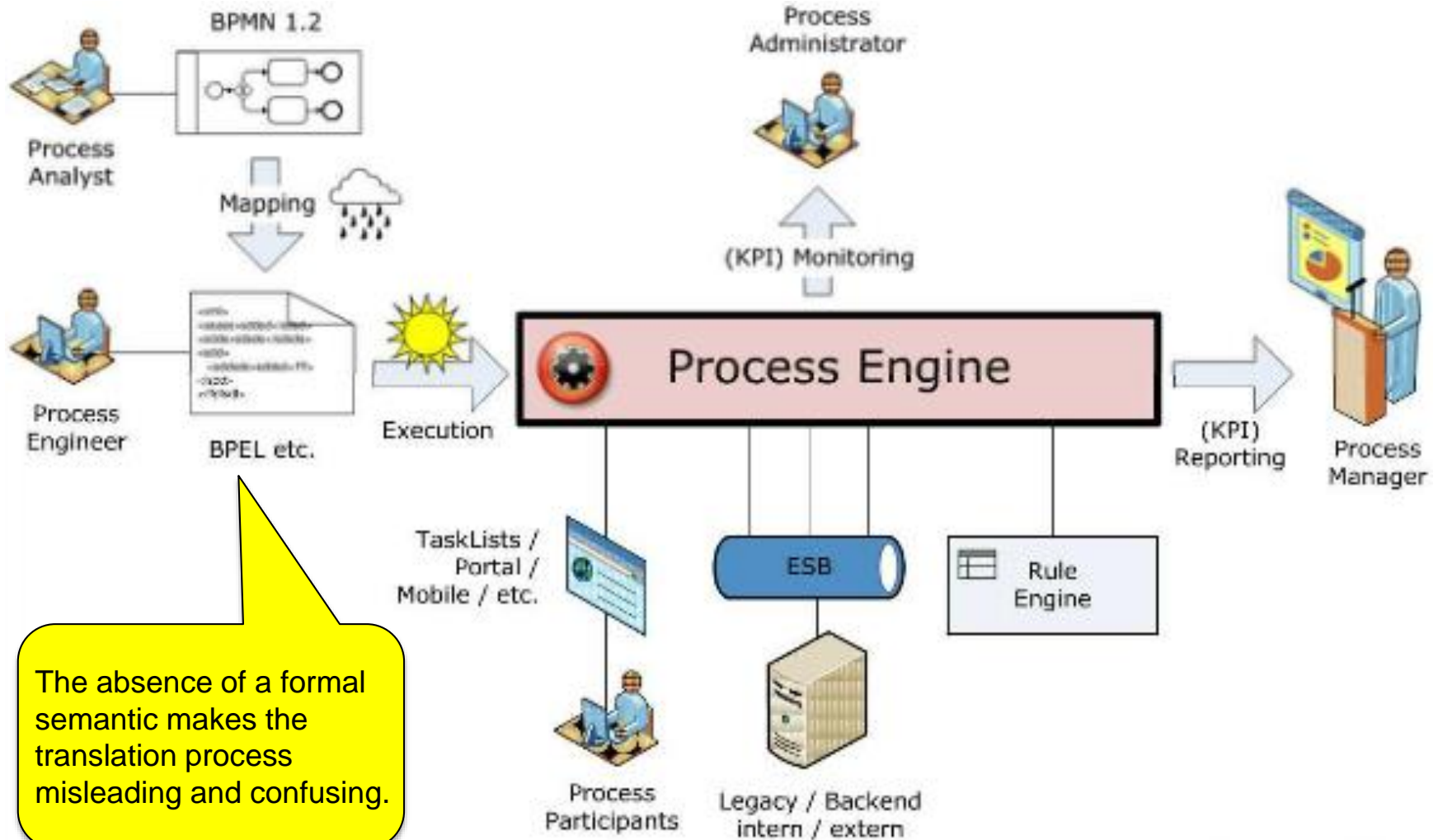
The absence of a formal semantic makes the translation process misleading and confusing.

Solving BPMN 1.x problems with BPMN 2.0

BPMN 2.0 has now its explicit execution semantic, and it is possible to directly execute detailed BPMN models.

BPMN 2.0
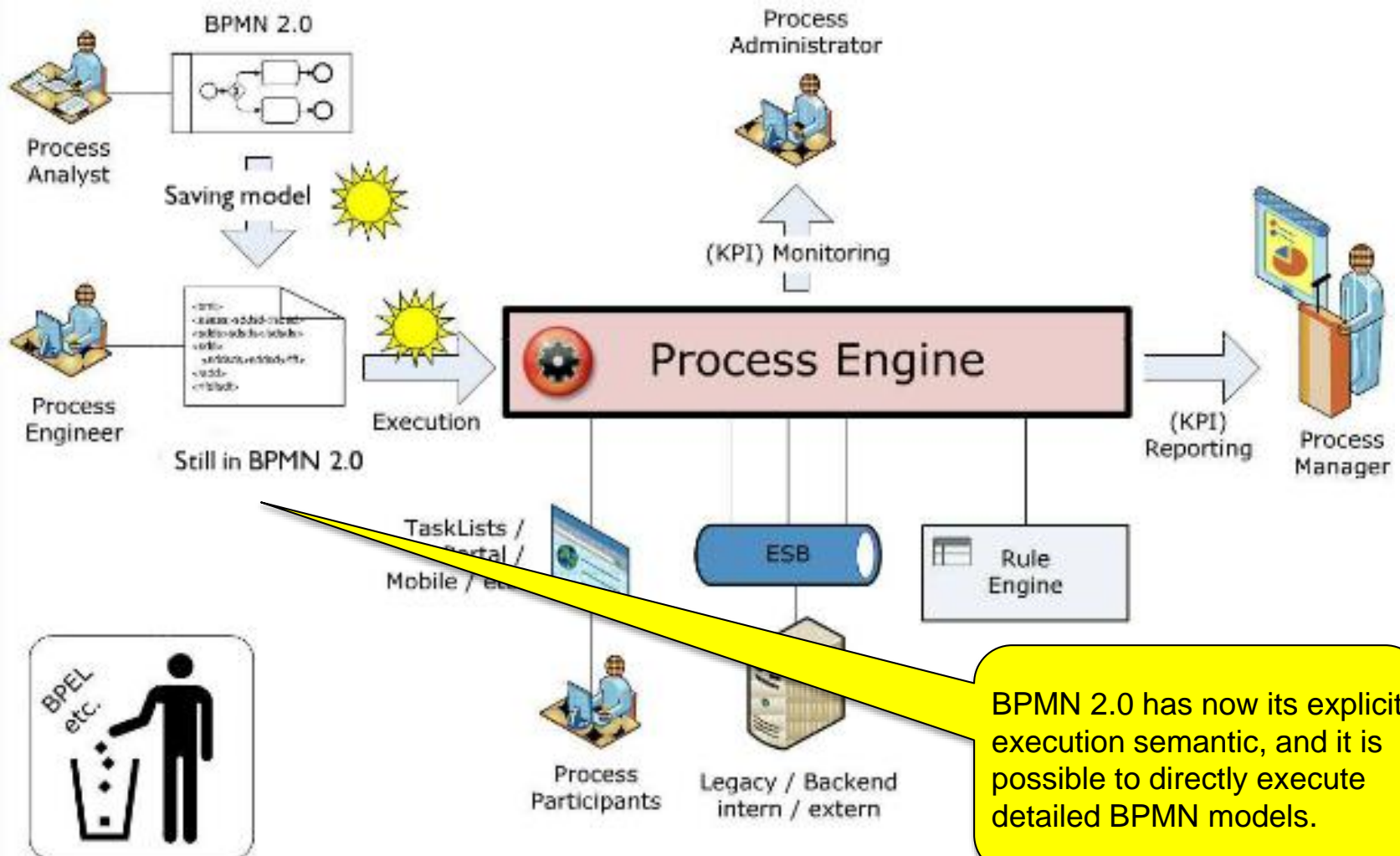
- **The typical modeler does not need to work with the metamodel**. Normally, s\he uses a modeling tool that only allow the creation of models complying with the specification, and thus with the metamodel.

- It is rather the vendors of modeling tools and process engines who have to deal with the metamodel.

- BPMN 2.0 supports 3 different levels of process modeling:
  - **Process Maps**: simple flow charts of the activities.
  - **Process Descriptions**: flow charts extended with additional information, but not enough to fully define actual performance.
  - **Process Models**: flow charts extended with enough information so that the process can be analyzed, simulated, and/or executed.

Topics

- Process Modeling
- BPMN Background
- **Basic Concepts**
- Advanced Concepts
- Conclusions

BPMN Basic Elements

- **4 basic elements**, that usually cover the 80% of modeling needs.

It is an ***atomic unit of work*** that has a duration.
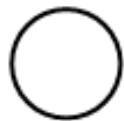
Event

Task

Flow

Gateway

Events represent things that ***happen instantaneously***.

Arcs impose ***temporal constraints*** between flow objects.

Elements that ***control the flow*** of execution of the process.

Connecting Activities

- The *sequence flow d*efines the *order of flow objects in a process* (activities, events and gateways). Each activity can have one or more *incoming sequence flow* and one or more *outgoing sequence flow.*

- Typically, an activity tends to have a *single incoming and a single outgoing sequence flow.*
- Each process must have always at least a **start event** (a circle with a thin border), that shows where a *process instance can start*, and a **end event** (a circle with a thick border), for indicating when a *process instance completes*.
- Starting from a process model, an organization runs a number of independent instances of this process.

Activity Behaviour

- Once a process instance has been created, we use the notion of **token** to identify the progress (or **state**) of that instance.

- A token *is a "**theoretical object**"* used to create a descriptive "simulation" of the behavior associated to each BPMN element (it is not currently a formal part of the BPMN specification).

- A token is created in the start event, traverses the sequence flow and is destroyed in a end event. That is, <u>there is no time associated with the token travelling down a sequence flow</u>.

The Task becomes Ready

The Task is Active

An illustration that the Task is active

The Task is Complete
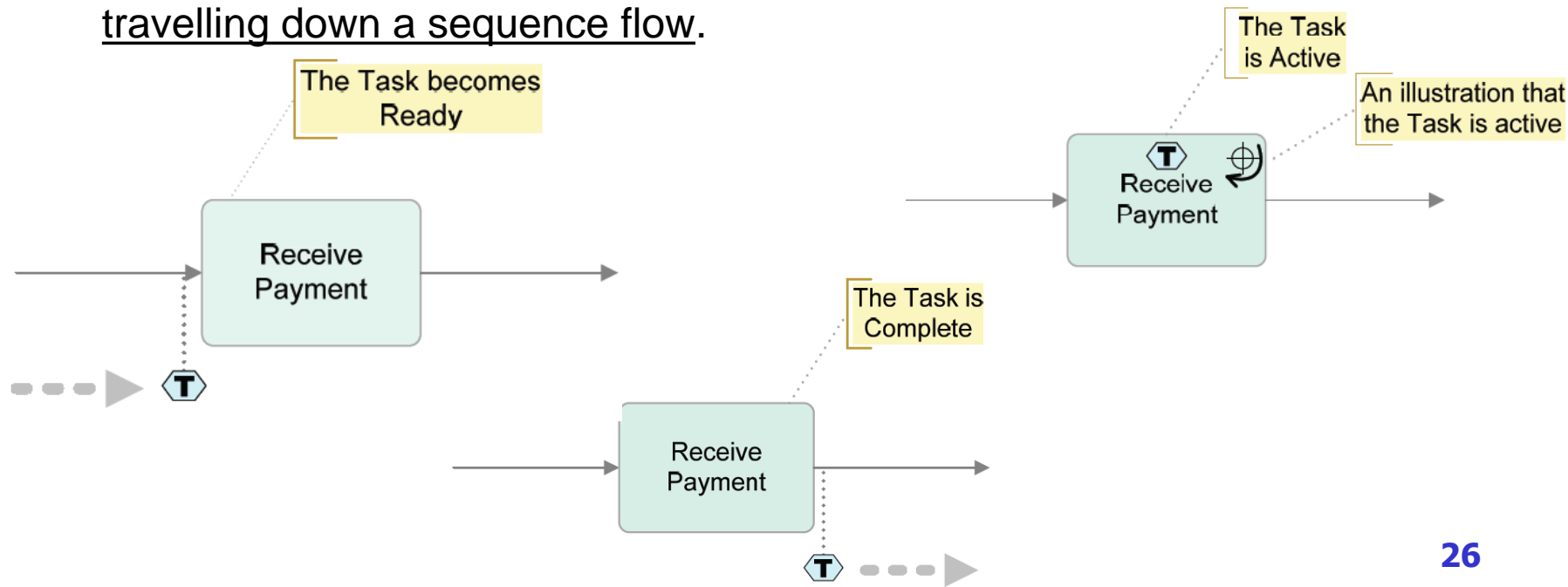
A first example: an order management process

These activities are **mutually exclusives.**

Activities

Start event

End event. BPMN adopts the *implicit termination semantics.*

Reject order

Order rejected

Send invoice

Purchase order received

Check stock availability

X

Confirm order

+

Ship goods

+

Order fulfilled

Every start/end event should be labelled, in order to communicate **what triggers an instance of the process o**r **what is the outcome when a process instance completes**.

Exclusive Gateways

Parallel Gateways

These activities can be **executed concurrently.**

A process instance completes only when each token flowing in the model reaches an end event.

A bit more on gateways

- A *gateway* implies that there is a gating mechanism that either allows or disallows passage of tokens through the gateway.

- As tokens arrive at a gateway, they can be merged together on input, or split apart on output depending on the gateway type.

- A *split gateway* represents a point where the process flow diverges, while a *join gateway* represents a point where the process flow converges.
  - Splits have <u>one incoming sequence flow</u> and <u>multiple outgoing sequence flows</u> (representing the branches that diverge).
  - Joins have <u>multiple incoming sequence flows</u> (representing the branches to be merged) and <u>one outgoing sequence flow</u>.

Types of gateways

- **Exclusive Gateways (XOR)**
    - Indicates locations within a business process where the sequence flow can take two or more alternative paths.
    - **Only one** of the paths can be taken.
    - Depicted by a diamond shape that *may* contain a marker that is shaped like an "X".
    - We use a ***XOR-join*** to merge two or more alternative branches that may have previously been forked with a ***XOR-split***.
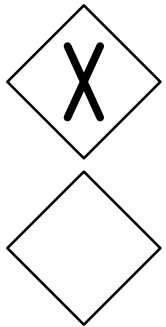
- **Parallel Gateways (AND)**
    - Provide a mechanism to synchronize parallel flows (***AND-join***) and to create parallel flows (***AND-split***), with activities that can be executed concurrently.
    - Depicted by a diamond shape that *must* contain a marker that is shaped like a plus sign.
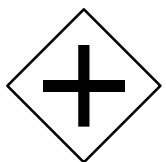
Exclusive Gateways – Splitting Behaviour
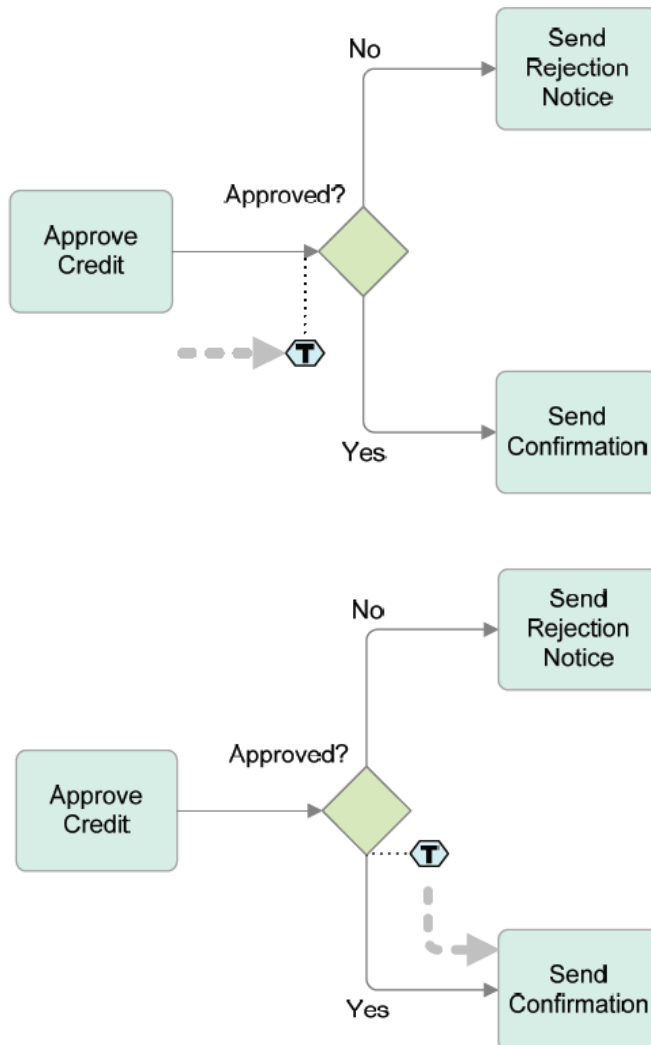
- **Exclusive gateways** are locations within a process where there are two or more alternative paths.
- The criteria for the decision, which the exclusive gateway represents, exist as ***conditions on each of the outgoing sequence flow***.
- When a token arrives at an exclusive gateway, there is an immediate evaluation of the conditions that are on the gateway's outgoing sequence flow. One of those conditions must always evaluate to true.

Default Conditions

No → Send Payment Problem

Payment OK?

No, Exceeded Retry Limit → Cancel Order

Default Condition

Yes → Send Confirmation
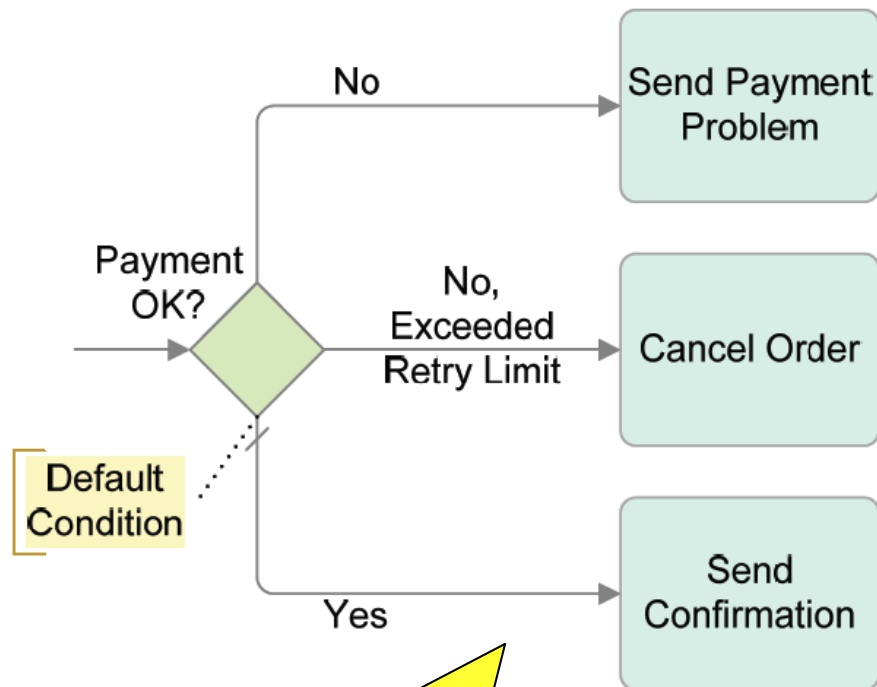
The **default condition** has the meaning of "*otherwise*", and it can be left unlabeled.

- One way for the modeler to ensure that the process does not get stuck at an exclusive gateway is to use a ***default condition*** for one of the outgoing sequence flow.

- The *default condition* can complement a set of standard conditions to provide an **automatic escape mechanism** in case *all the standard conditions evaluate to false.*

- The *default* is chosen if all the other sequence flow conditions turn out to be false.

Exclusive Gateways – Merging Behaviour

- When a token arrives at the exclusive gateway, there is no evaluation of conditions (on the incoming sequence flow), and immediately moves down the outgoing sequence flow.

The Token is "passed-through." There is no delay, incoming Condition evaluation, or synchronization
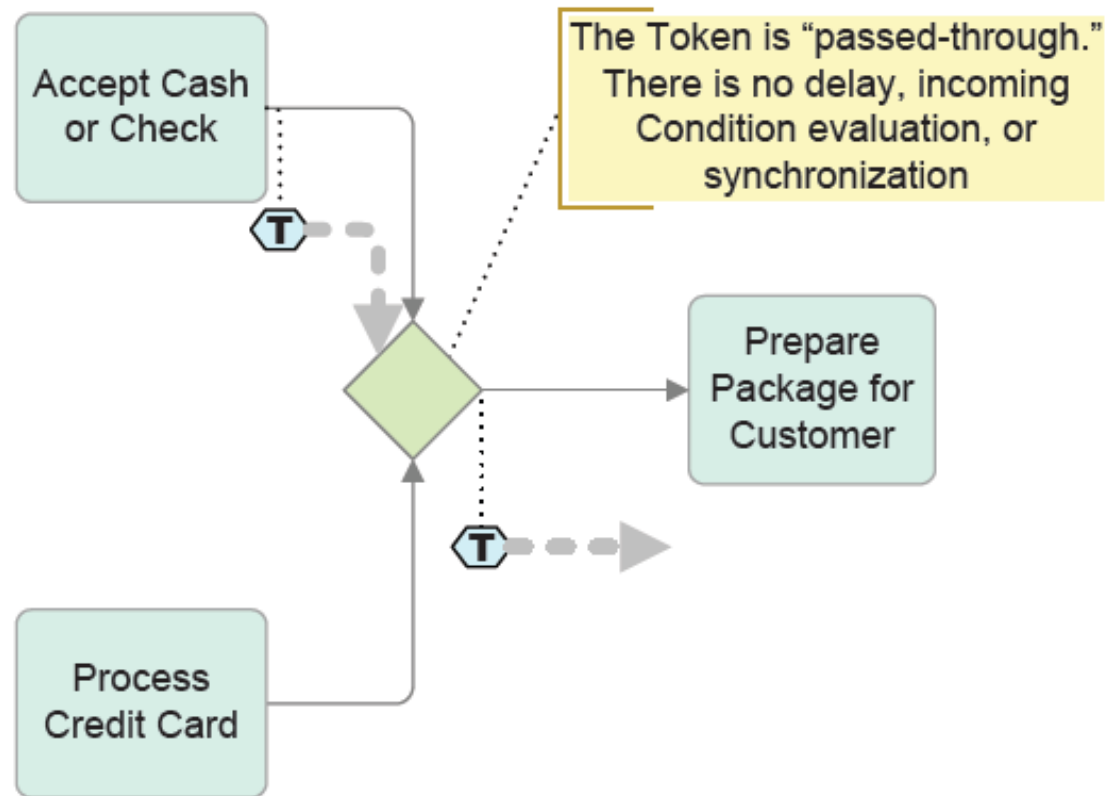
Exercise

- As soon as an invoice is received from a customer, it needs to be checked for mismatches.

- The check may result in either of these three options:
  - i) there are no mismatches, in which case the invoice is posted;
  - ii) there are mismatches but these can be corrected, in which case the invoice is re-sent to the customer; and
  - iii) there are mismatches but these cannot be corrected, in which case the invoice is blocked.

- Once one of these three activities is performed the invoice is parked and the process completes.

Solution
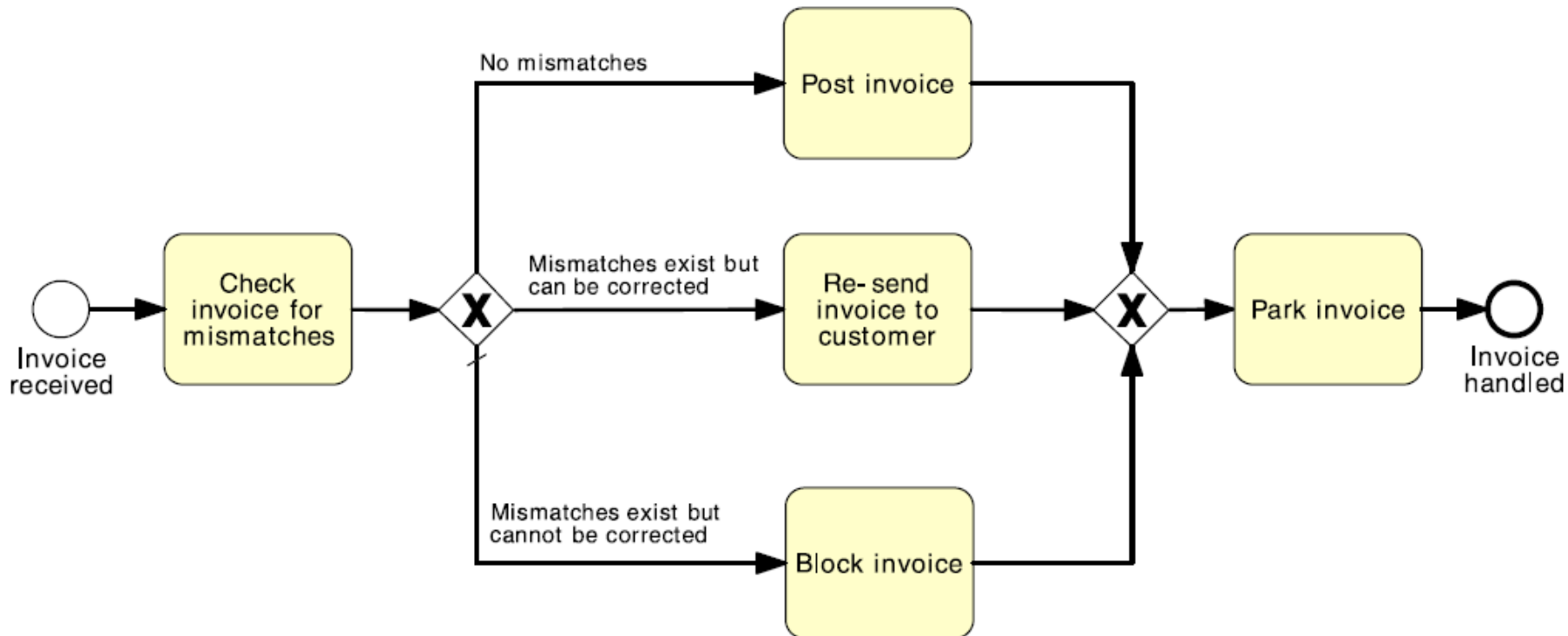
Parallel Gateways – Splitting Behaviour
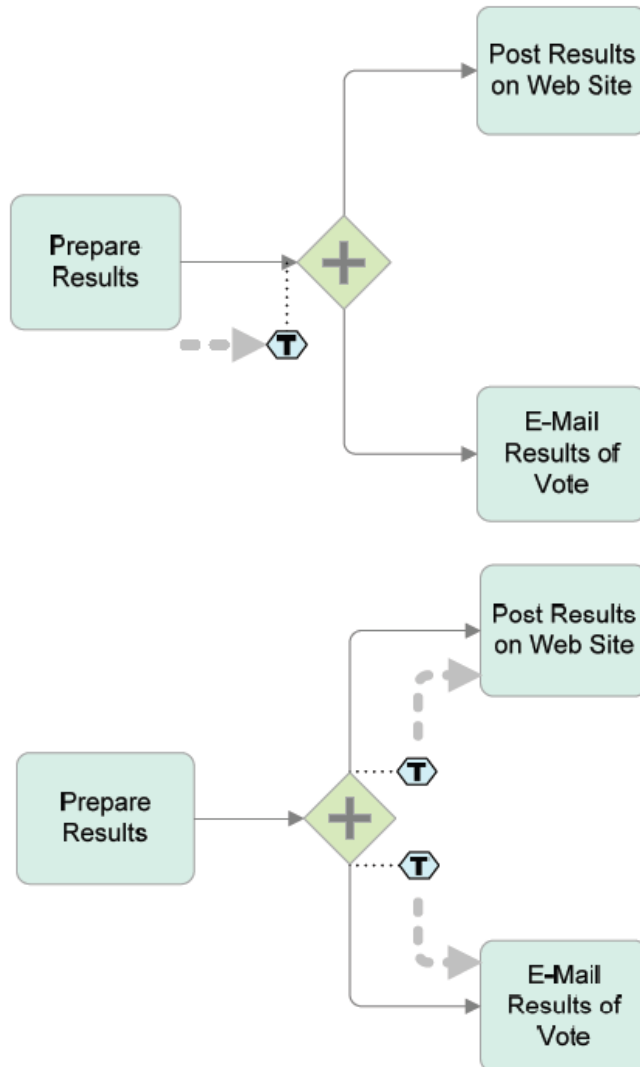
- When a *token arrives at a parallel gateway,* **there is no evaluation** of any *conditions* on the *outgoing sequence flow.*

- The parallel gateway will create parallel paths.

- This means that the gateway will **create a number of tokens that are equal to the number of outgoing sequence flow**. One token moves down each of those outgoing sequence flow.

Parallel Gateways – Merging Behaviour

- To synchronize the flow, the parallel gateway will wait for a token to arrive from each incoming sequence flow.

- When the first token arrives, there is no evaluation of a condition for the incoming sequence flow, but the **token is "held" at the gateway and does not continue**.

- When all the tokens are arrived, then they are merged and one token moves down the outgoing sequence flow.
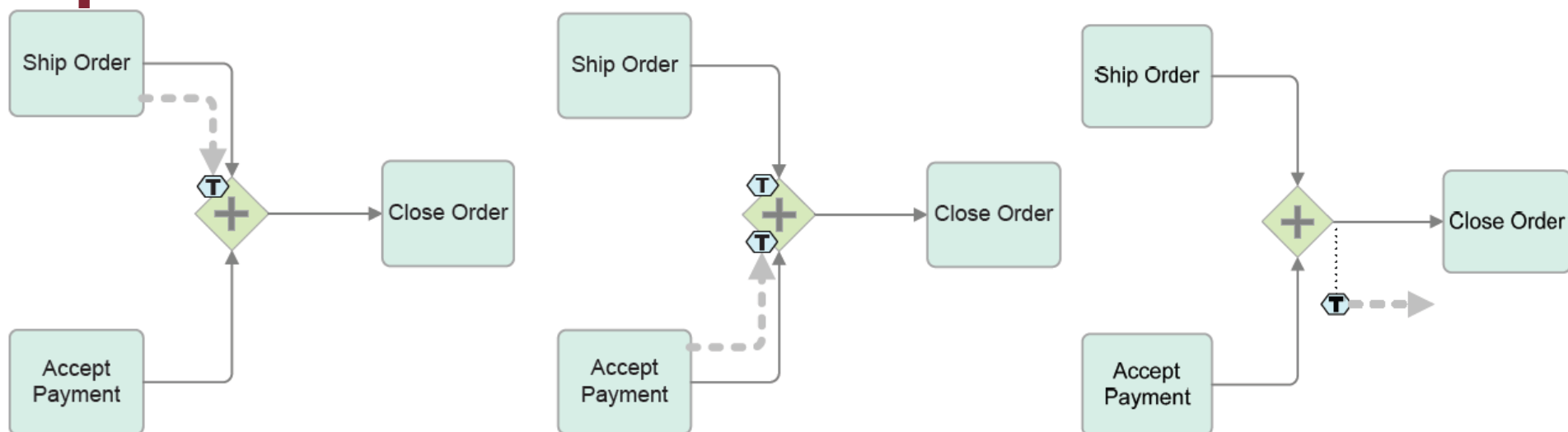
Exercise

- Once the boarding pass has been received, passengers proceed to the security check. Here they need to pass the personal security screening and the luggage screening. Afterwards, they can proceed to the departure level.
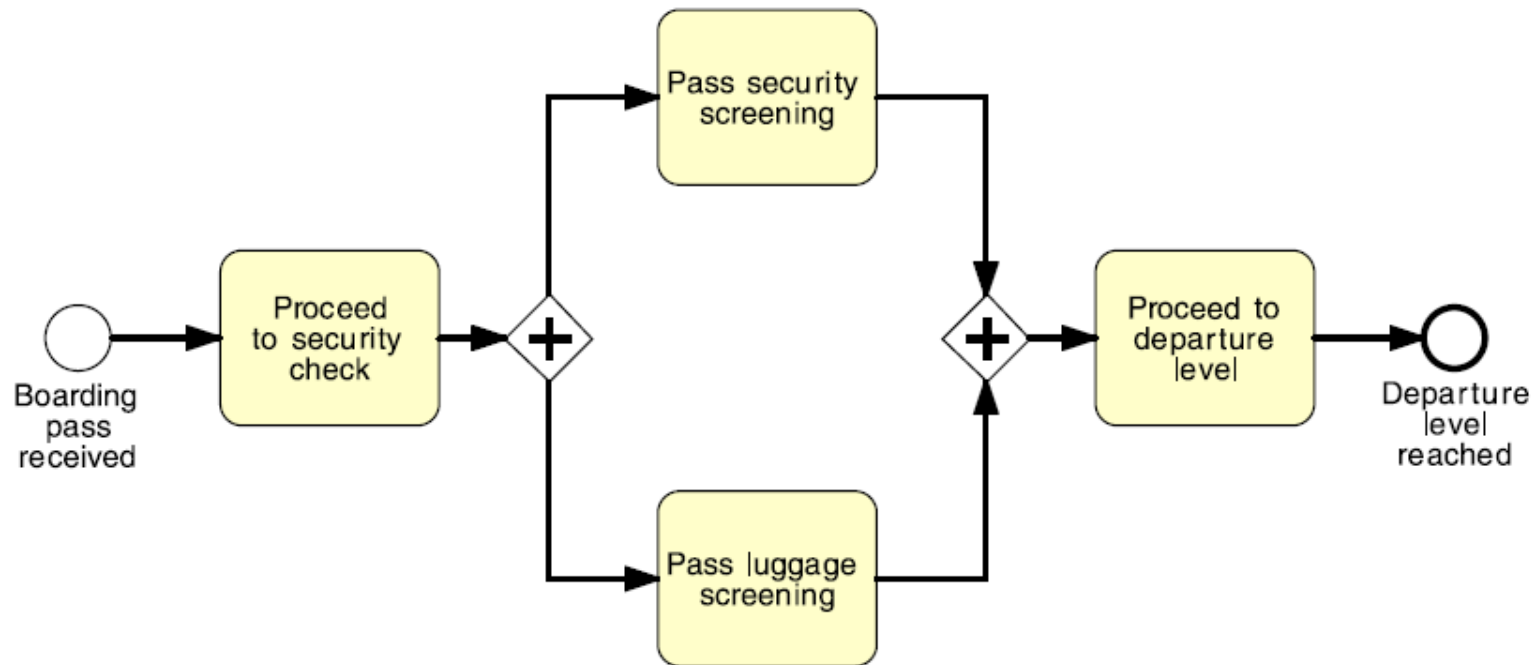
Exercise

- Describe the behavior of this process.

Only two of three paths will be used at any one time. Thus, the Process will be stuck waiting for the third path.
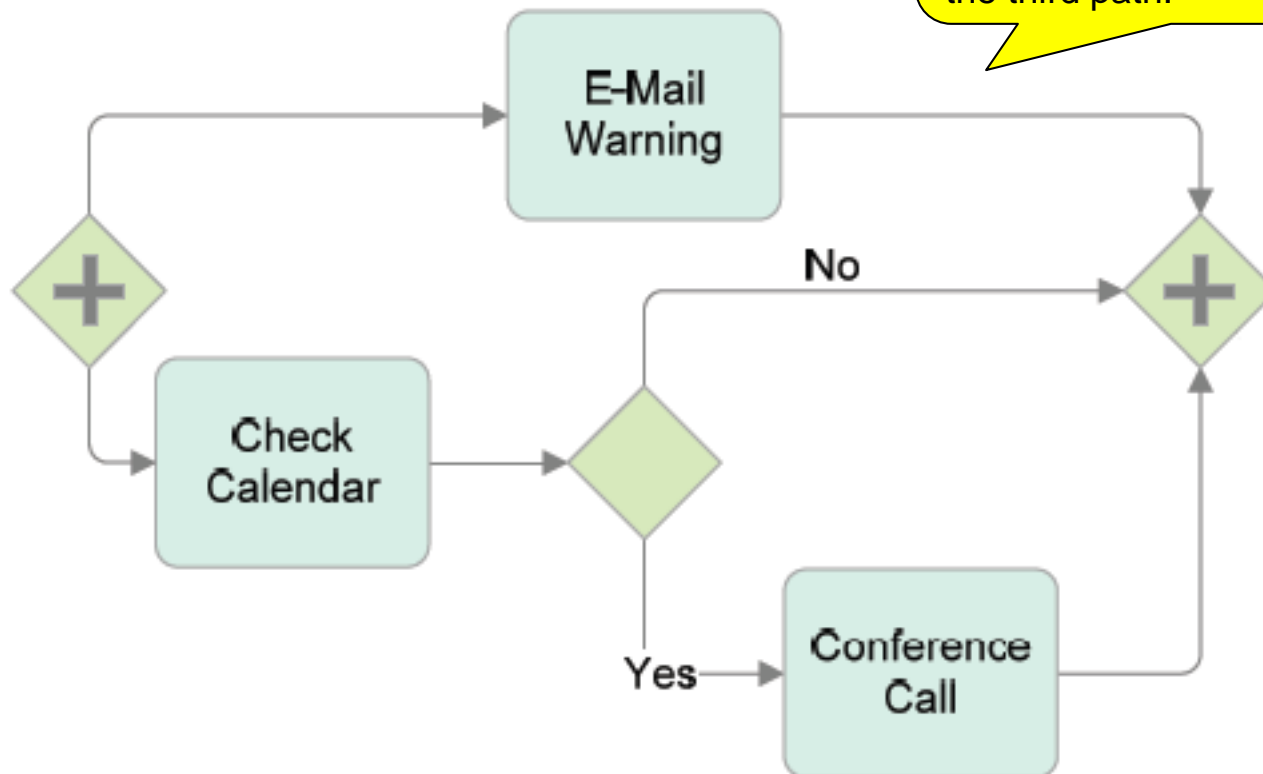
Solution
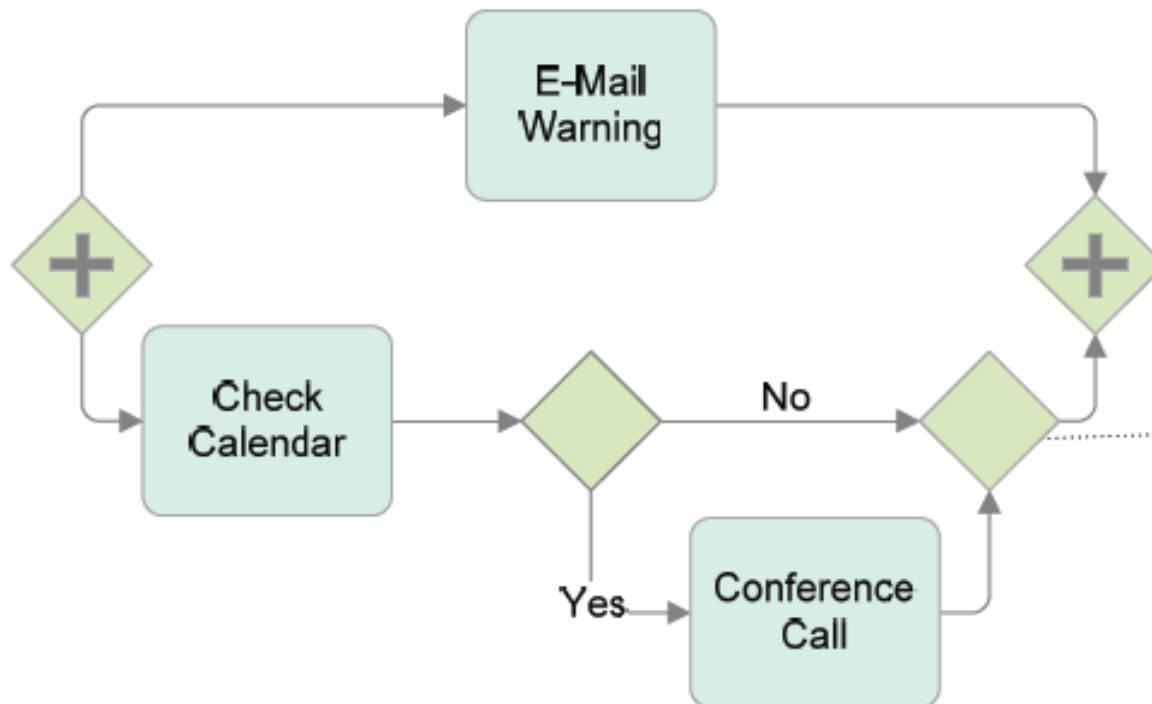
The intervening **Exclusive Gateway** reduces the number of incoming paths to the **Parallel Gateway**.

Inclusive gateways

- **Inclusive Gateways (OR)**
  - Sometimes we may need to take **one or more branches** after a decision activity.
  - To model situations where a decision may lead to one or more options being taken at the same time, we need to use an *inclusive (OR) split gateway*.
  - An *OR-split* is similar to the XOR-split, but the conditions on its outgoing branches ***do not need to be mutually exclusive***, i.e., more than one of them can be true at the same time.
  - When we encounter an OR-split, we thus take one or more branches depending on which conditions are true.

Inclusive Gateways – Splitting Behaviour
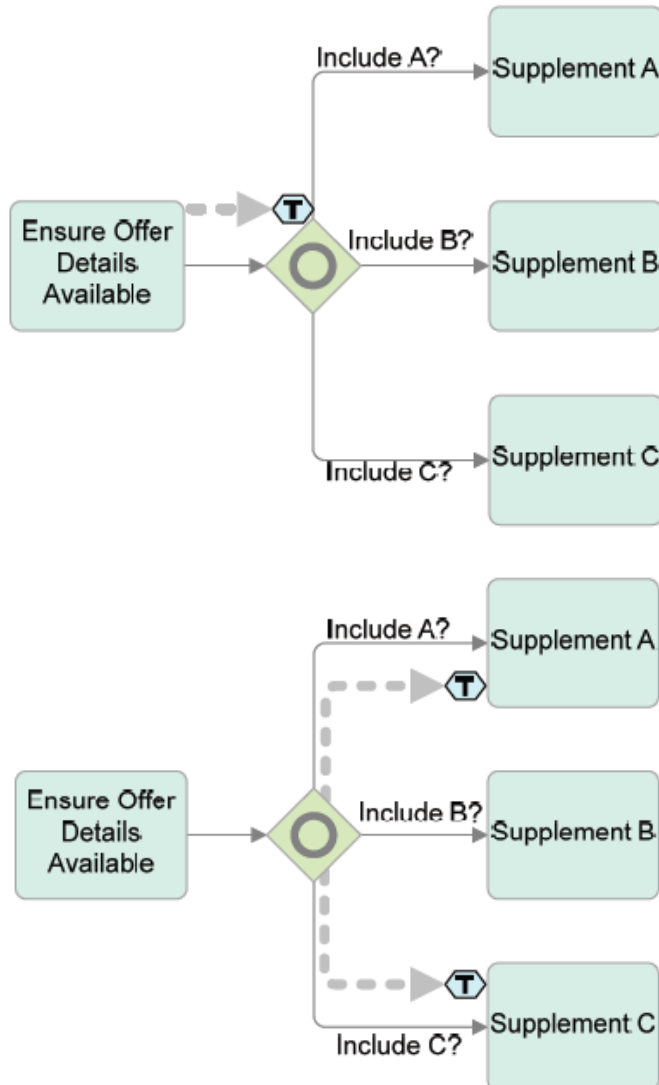
- Inclusive gateways support decisions where **more than one outcome is possible** at the decision point.

- Inclusive gateway with multiple outgoing sequence flows creates **one or more paths** based on the conditions on those sequence flow.

- In terms of token semantics, this means that the OR-split takes the input token and ***generates a number of tokens equivalent to the number of output conditions that are true.***

- Every condition that evaluates to true will result in a token moving down that sequence flow.

- At least one of those conditions *must evaluate to true.*

Inclusive Gateways – Merging Behaviour
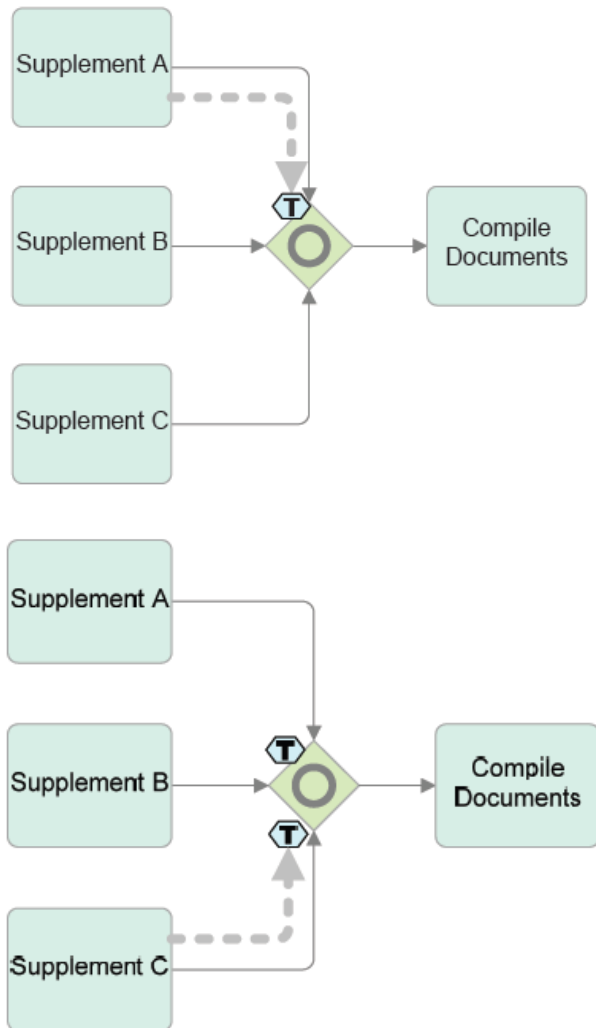
- When the first token arrives at the gateway, the gateway will "look" upstream for each of the other incoming sequence flow to see if there is a token that might arrive at a later time.

- Thus, the gateway will hold the first token that arrived in the upper path until the other token from the lower path arrives.

- When **all the expected tokens have arrived** at the gateway, the process flow is synchronized (the incoming tokens are merged) and then a token moves down the gateway's outgoing sequence flow.

Exercise

A company has two warehouses that store different products: Amsterdam and Hamburg.

When an order is received, it is distributed across these warehouses: if some of the relevant products are maintained in Amsterdam, a sub-order is sent there; likewise, if some relevant products are maintained in Hamburg, a sub-order is sent there.

Afterwards, the order is registered and the process completes.

A first solution

order only contains
Amsterdam products

Forward sub-
order to
Amsterdam
warehouse

Check order
line items

Order
received

order only contains
Hamburg products

Forward sub-
order to
Hamburg
warehouse

Register
order

Order
completed

order contains
both Amsterdam
and Hamburg
products

Forward sub-
order to
Amsterdam
warehouse

Forward sub-
order to
Hamburg
warehouse

Some activities represented in
the process model have to be
duplicated.

A second solution

order does not contain
Amsterdam products

order contains
Amsterdam products

Forward sub-order to Amsterdam warehouse

Order received

Check order line items

Register order

Order completed

order does not contain
Hamburg products

order contains
Hamburg products

Forward sub-order to Hamburg warehouse
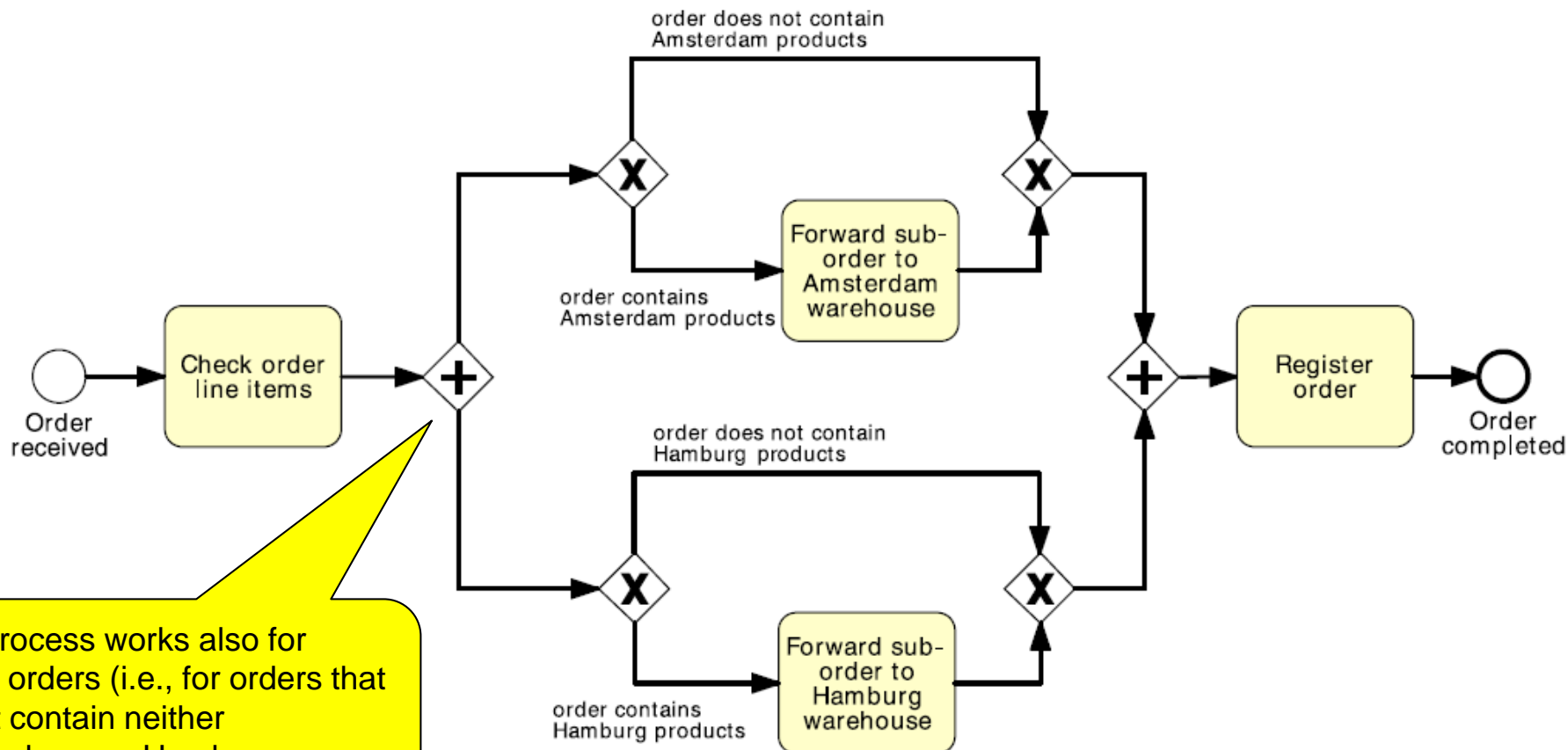
This process works also for empty orders (i.e., for orders that do not contain neither Amsterdam nor Hamburg products)

A third solution with OR gateways
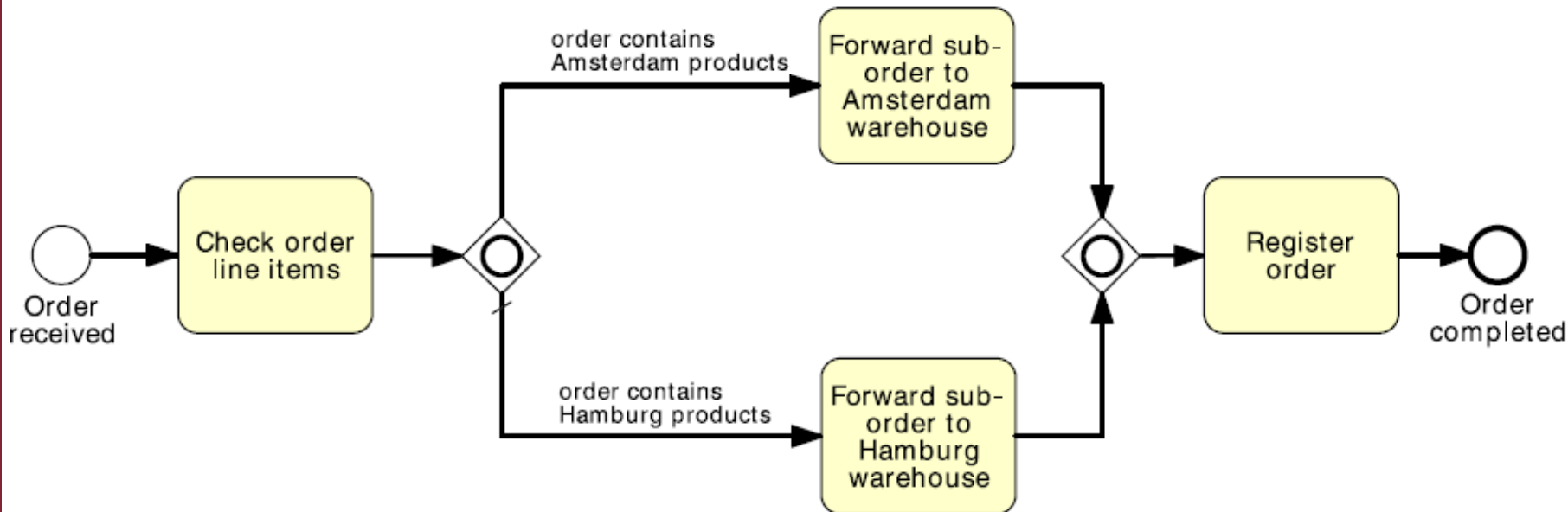
What type should we assign to this join?
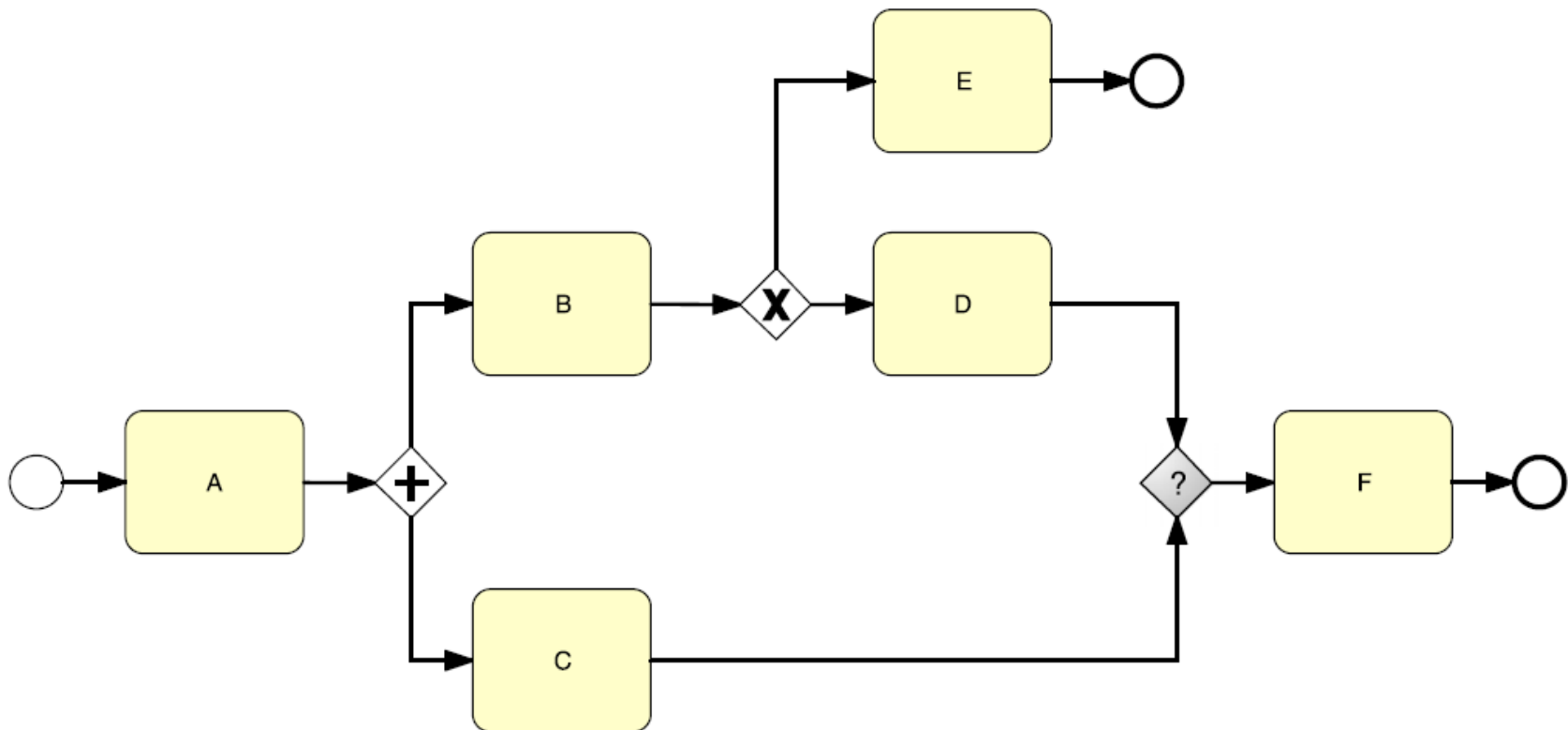
What type should we assign to this join?

Let us try an **AND-join** to match the preceding AND-split.

We recall that an AND-join waits for a token to arrive from each incoming branch. While the token from the branch with activity "C" will always arrive, the token from the branch with activities "B" and "D" may not arrive if this is routed to "E" by the XOR-split.

So if activity "D" is not executed, **the AND-join will wait indefinitely for that token**, with the consequence that the process instance will not be able to progress any further. This behavioral anomaly is called *deadlock* and should be avoided.

What type should we assign to this join?

Let us try an **XOR-join**. We recall that the XOR-join works as a pass-through by forwarding to its output branch each token that arrives through one of its input branches.

In our example this means that we may execute activity "F" once or twice, depending whether the preceding XOR-split routes the token to "E" (in this case "F" is executed once) or to "D" ("F" is executed twice).

While this solution may work, we have the problem that we do not know whether activity "F" will be executed once or twice, and **we may actually not want to execute it twice**. Moreover, if this is the case, we would signal that the process has completed twice, since the end event following "F" will receive two tokens. And this, again, is something we want to avoid.

What type should we assign to this join?

The only join type left to try is the **OR-join**.

An OR-join will wait for all incoming active branches to complete. If the XOR-split routes control to "E", the OR-join will not wait for a token from the branch bearing activity "D", since this will never arrive.

Thus, it will proceed once the token from activity "C" arrives. On the other hand, if the XOR-split routes control to "D", the OR-join will wait for a token to also arrive from this branch, and once both tokens have arrived, it will merge them into one and send this token out, so that "F" can be executed once and the process can complete normally.

When should we use an OR-join?

Since the OR-join semantics is not simple, the presence of this element in a model may confuse the reader.

Thus, we suggest to use it only when it is strictly required.

Clearly, it is easy to see that an OR-join must be used whenever we need to synchronize control from a preceding OR-split. Similarly, we should use an AND-join to synchronize control from a preceding AND-split and an XOR-join to merge a set of branches that are mutually exclusive.

Process Modelling Viewpoints
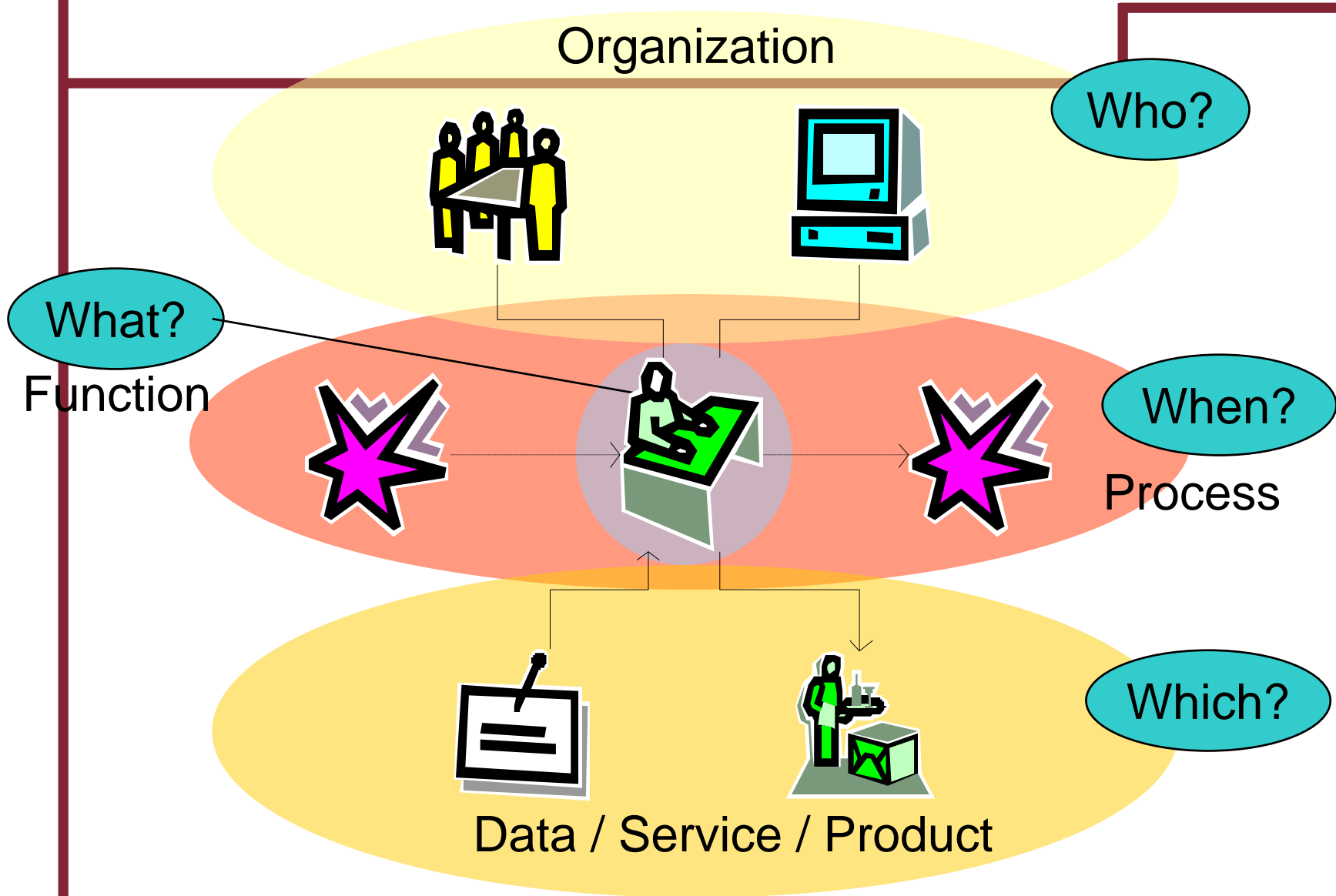
Organisational Elements in Process Models

Two basic abstractions:

- **Resource:** Anything or anyone involved in the performance of a process activity. It can be a human actor, an equipment (e.g. a printer) or a software system.
  - The resource perspective of a process is interested in **active resources**.
- **Resource class:** Set of resources with shared characteristics, e.g. Clerks, Managers, Insurance Officers

A *resource class* may represent either a:

- **Role** (skill, competence, qualification)
  Classification based on what a resource can do or is expected to do (e.g., a clerk is a role).
- **Group** (department, team, office, organizational unit)
  Classification based on the organization's structure (e.g., the administration department in an organization).

Resource Modelling in BPMN

- In BPMN, resource classes are captured using:
  - **Pools**
    - represent **independent organizational entities** in a collaborative business process diagram, e.g., Customer is **independent** from the Supplier.
    - **Independent** means they do not share any common system that allows them to communicate implicitly. Hence, they have to communicate explicitly through the use of messages.
  - **Lanes**
    - represent **multiple resource classes in the same organizational space** (i.e., in the same pool) and sharing common systems.
    - The Sales Department and the Marketing Department of the same company may be represented in the same pool, but in different lanes. They can communicate directly.
    - Clerks and Managers can be modelled in two separate lanes, but in the same pool representing the bank.
      - The clerk creates a new loan application...
      - A manager evaluates the loan application...

...ools — ...

In practice, **each Pool represents a distinct Process** and **each set of resource classes has its own Pool**.

Pools act as containers for the flow objects of a specific set of resource classes and communicate by means of messages.(dotted lines).

Pool1

This is another

message 1

message 2

message 3

This is one process

Pool2

Lane1

Lane 2
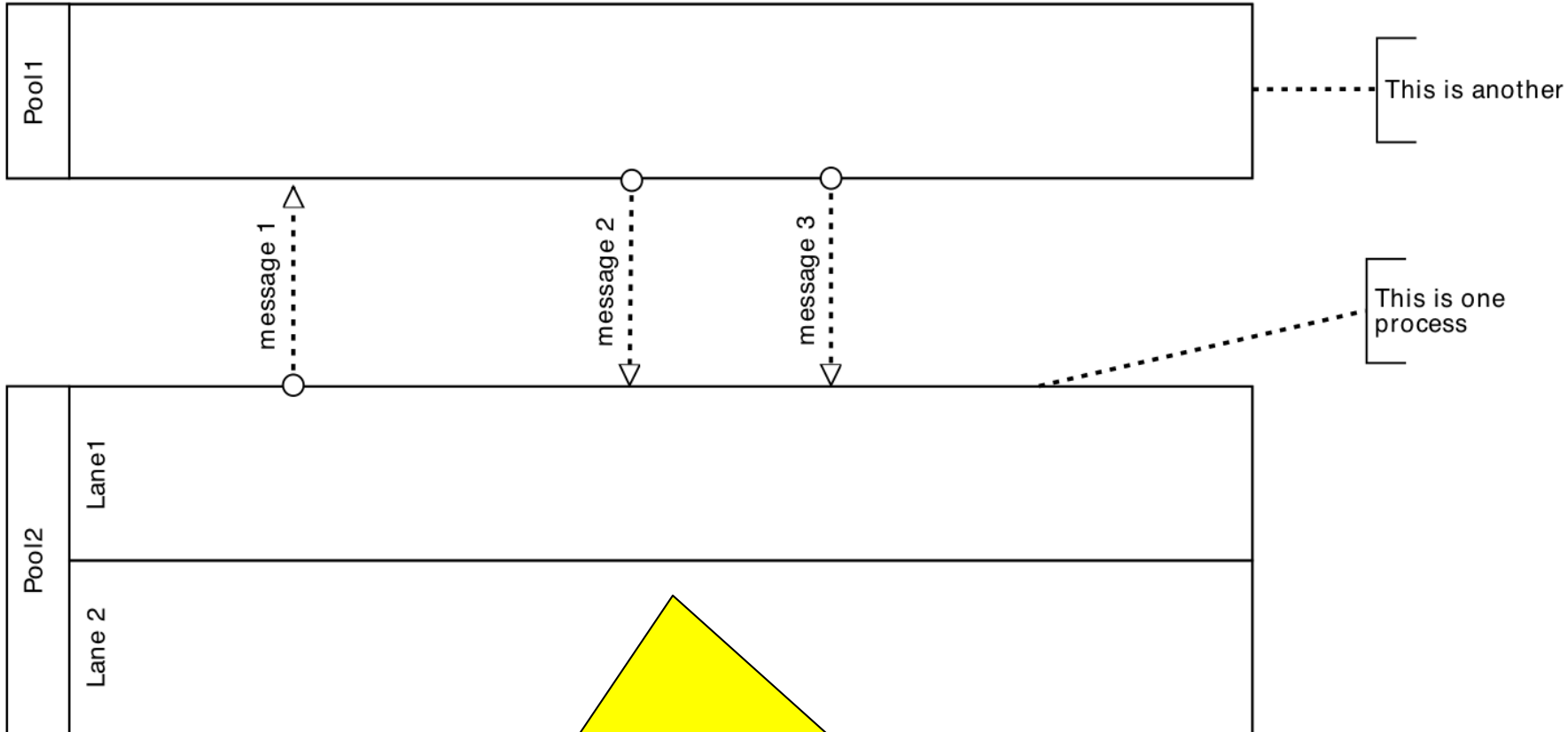
In each pool we may have multiple lanes, often assumed to represent **internal business roles** within a Process. Lanes actually provide a generic mechanism for partitioning the objects within a Pool.

Message Flows

- Message flow defines the messages/communications between **two separate participants** (shown as pools) of the diagram.

- Message flow must always occur between two separate pools and **cannot connect two objects within a single pool**.
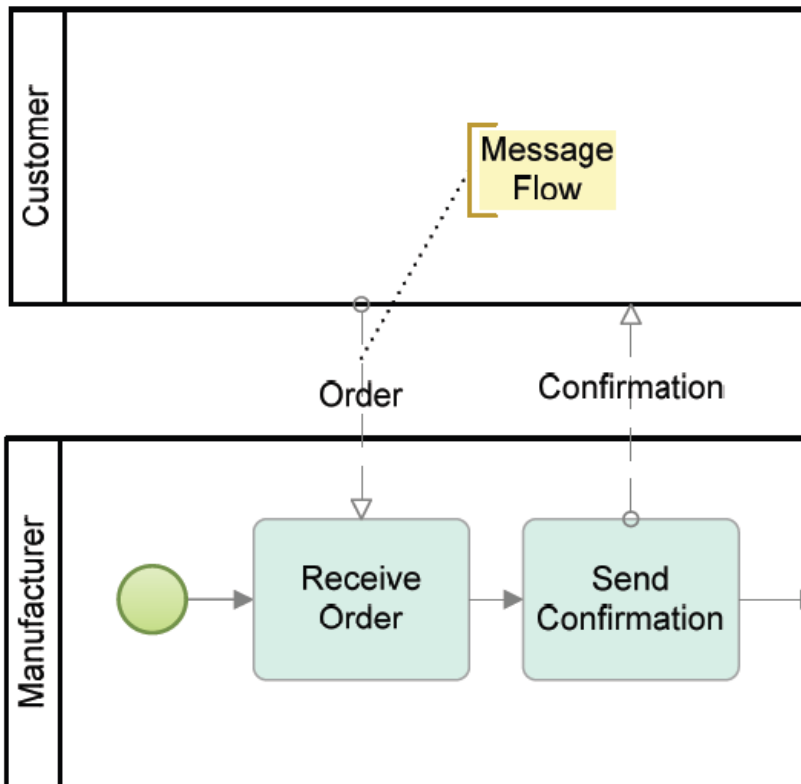
- Thus, message flow is only used in collaborations diagrams with two or more pools.

- Sequence flow cannot cross a pool boundary - i.e., a Process is fully contained within a pool.

Order Management Process with Pools
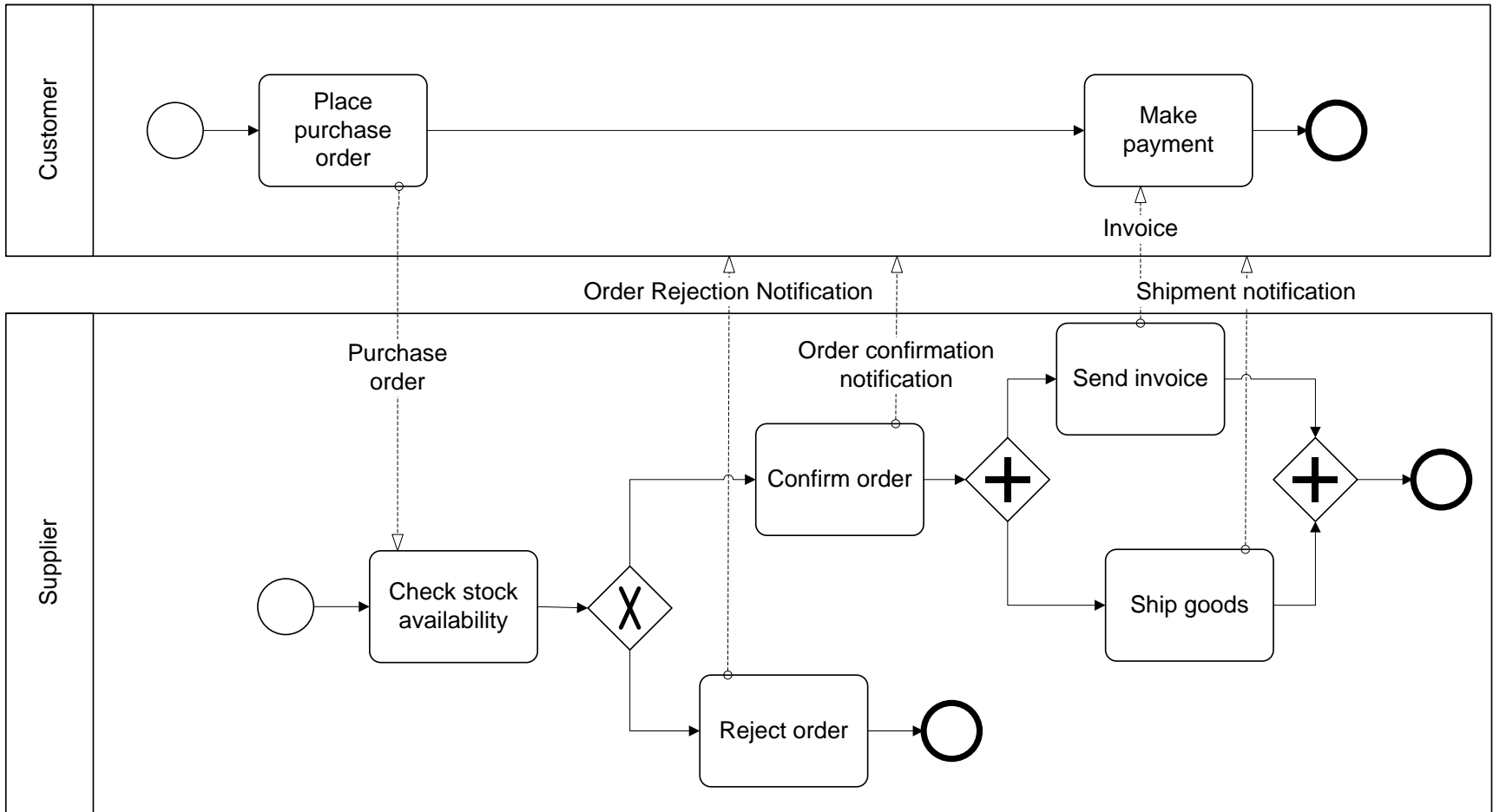
Order Management Process with Lanes
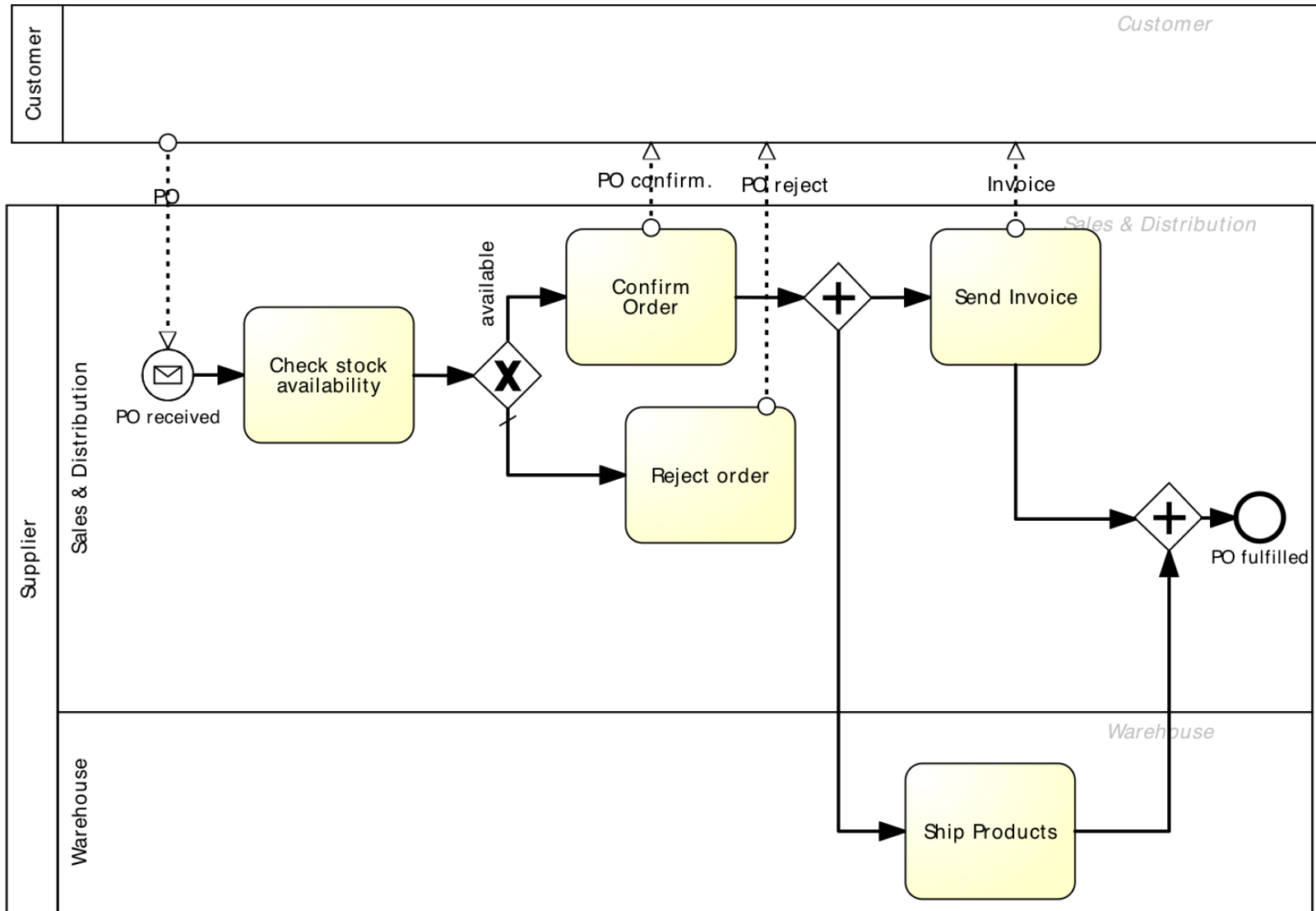
Lanes

- Lanes **create sub-partitions for the objects within a pool**.
- These partitions are used to group process elements (showing how they are related), or which roles have responsibility for carrying out the activities.

- Lanes often represent organization roles (e.g., Manager, Administration, Associate, etc), but can represent any desired classification (e.g., underlying technology, organizational departments, company products, etc).
- Sequence flow can cross Lane boundaries.
- Message flow is not used within or across lanes of a pool.
- Lanes can be nested.
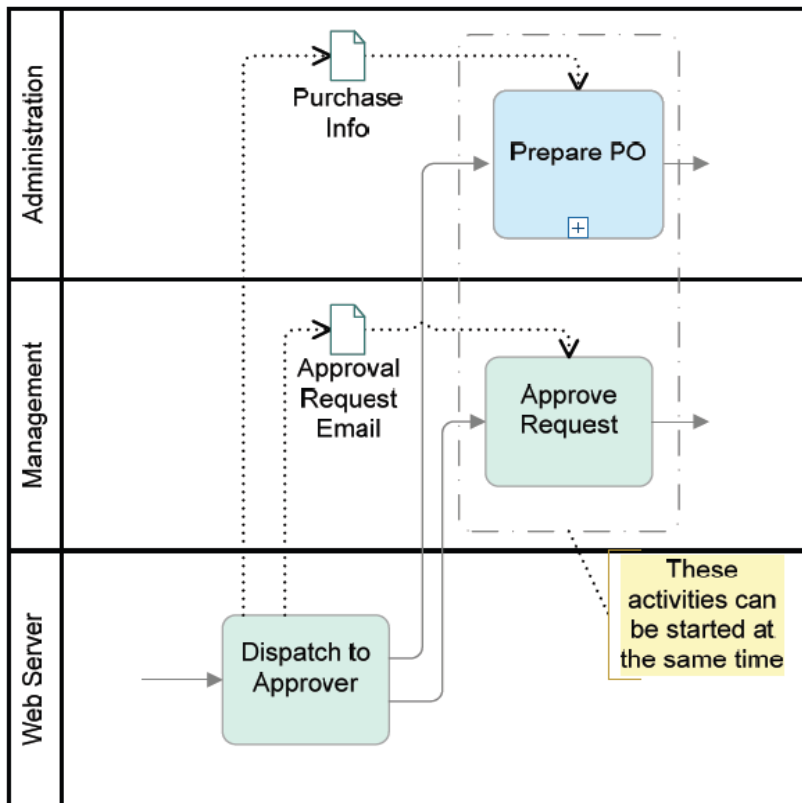
Pools – black box

- A pool **is not required to contain a process**. Known as a "***black box", these pools do not show activities* or sequence flow inside its boundary.

- In this example, the "Customer" Pool is a *black box (as far as Mortgage Co is concerned, they have no knowledge of the Processes of their Customer*).
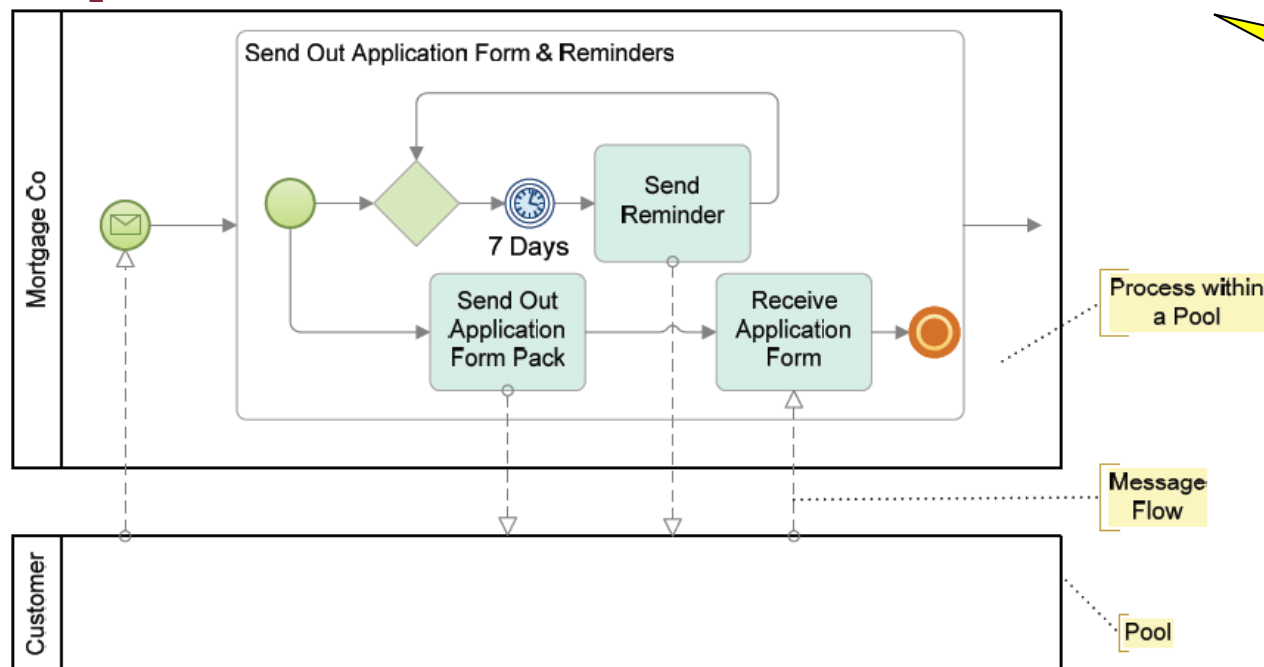
When the Pool is black box, Message Flow connects to its boundary.

Artifacts

- *Artifacts* provide a mechanism to capture **additional information** about a process. This information **does not directly impact** the flow chart characteristics of a process.

- Three different kinds of artifacts are available : *Groups, Text Annotations, Data Objects*.

- Modelers and tool vendors can extend BPMN through the addition of new types of artifacts.

A *Text Annotation* provides the modeler with the ability to add further descriptive information or notes about a process or its elements.
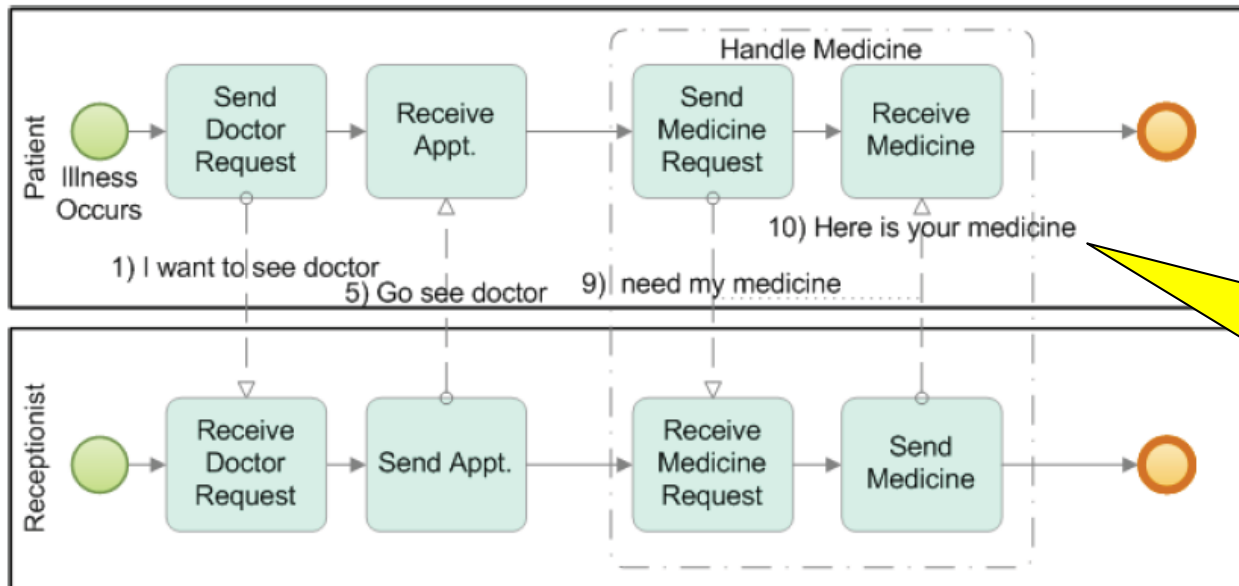
Groups cannot be interrupted by Intermediate Events

A *Group* is used to surround a group of flow objects in order to highlight and/or categorize them.

Data *Artifacts*

Data Object

Data Store
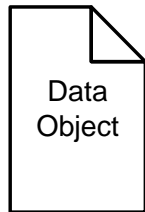
Directed association

Undirected association

- *Data Objects* are a mechanism to show how data is required or produced by activities.
  - They are depicted by a rectangle that has its upper-right corner folded over.
  - Represent *input* and *output* of a process activity.

- *Data Stores* are containers of data objects that need be persisted beyond the duration of a process instance.
- **Associations** are used to link data objects and data stores with flow objects (e.g. activities and, sometimes, events).
- **Directed associations** are used to understand when process activities can read/write data objects from/to data stores.

Data Objects

- Data objects are used to show how data and documents are used within a Process as *inputs* and *outputs* of activities.

- Data objects may also have "**states**" that depict how the object (document) is updated within the Process. The state is usually shown under the name of the data object and is placed between brackets.
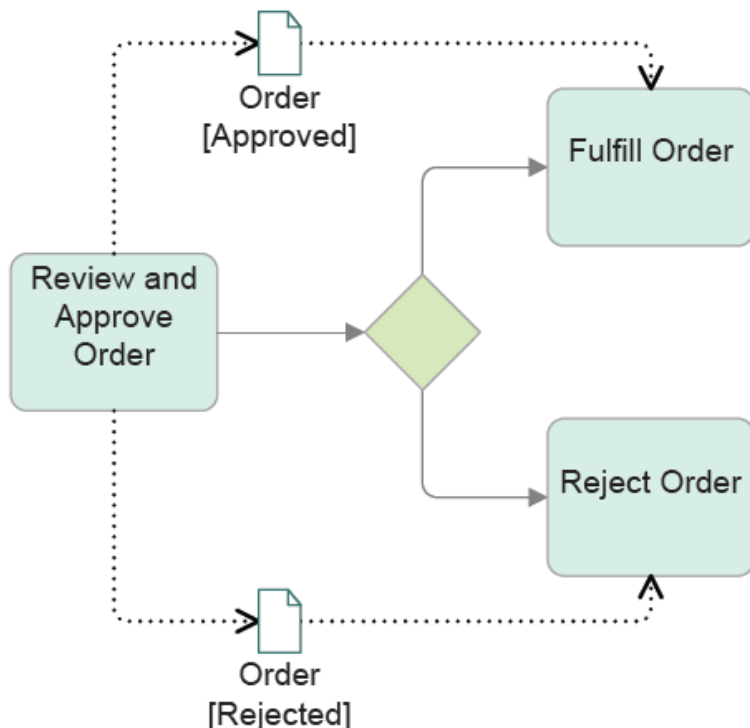
- By using the state of a data object and placing it within multiple locations within a diagram, the modeler can document the changes that a data object will go through during the Process.

- **Data flow** represents the movement of data objects from into and out of activities.

- In BPMN, data flow is decoupled from the sequence flow.

Extending the Order Mgt Process
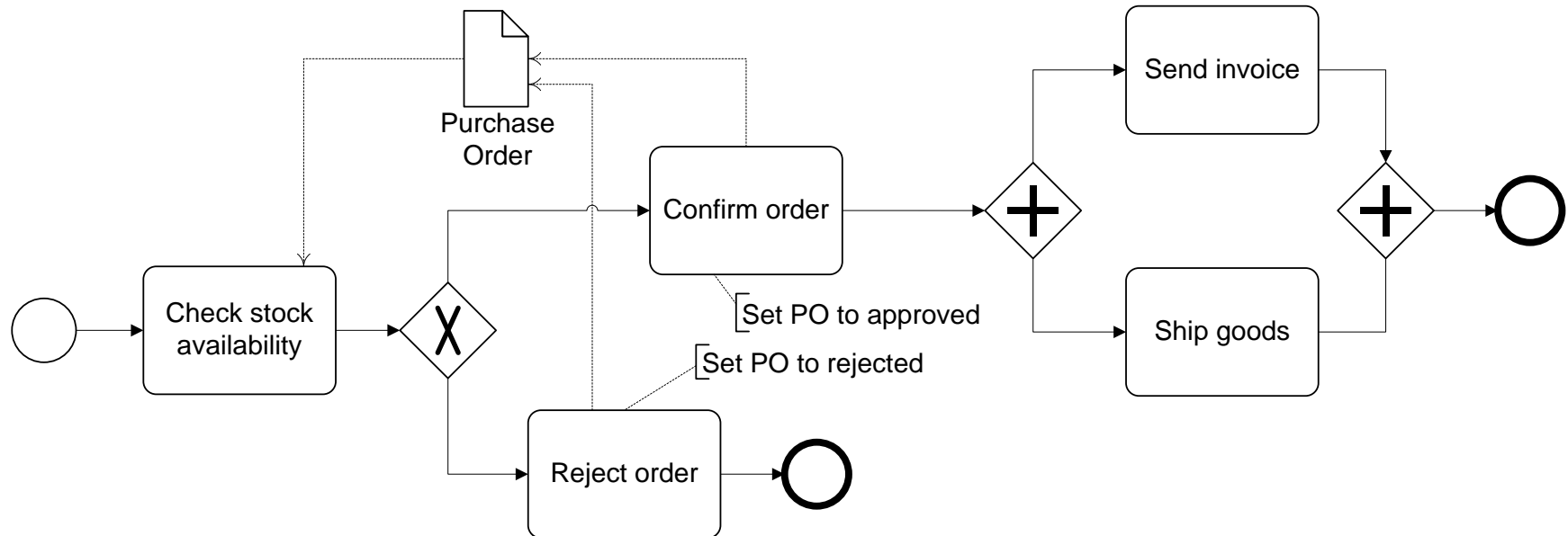
- The Purchase Order document serves as an input to the stock availability check. Based on the outcome of this check, the status of document is updated, either to "approved" or "rejected".

- Input data objects **are required for an activity to be executed**. Even if a token is available on the incoming arc of that activity, the latter cannot be executed until all input data objects are also available.

Exercise

Insert data objects in the following process model :
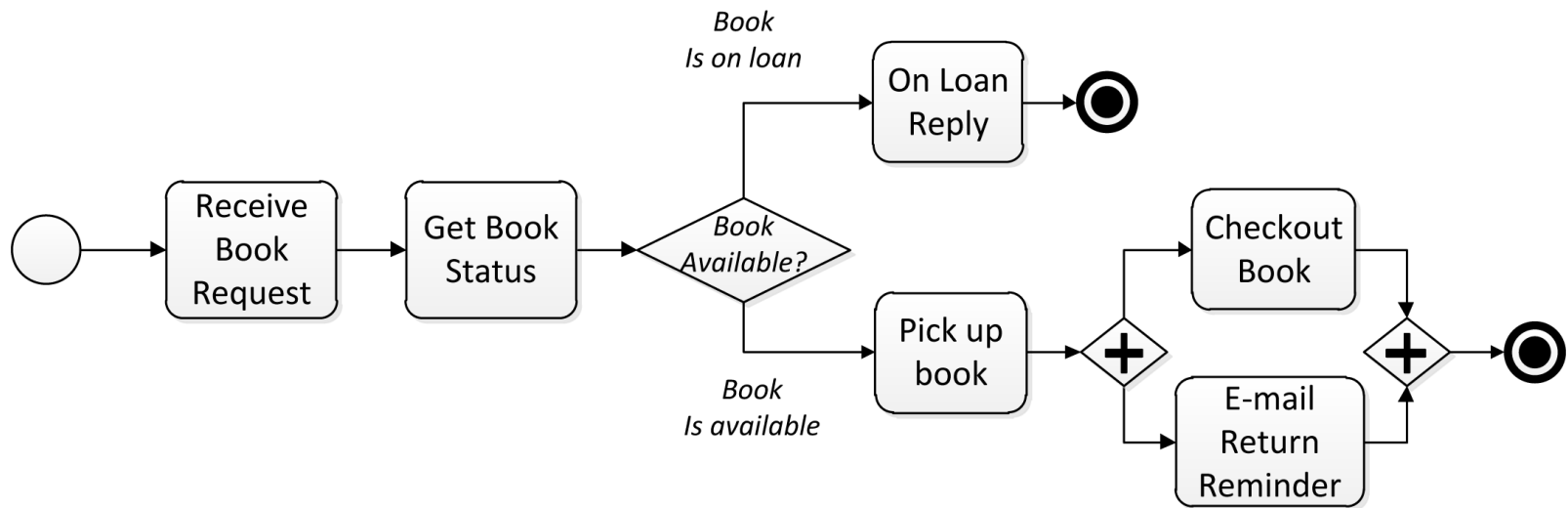
Solution

Insert data objects in the following process model :

Books DB

Book Information

Book
Is on loan

On Loan
Reply

Receive
Book
Request

Get Book
Status

Book
Available?

Book
Is available

Pick up
book

Checkout
Book

E-mail
Return
Reminder

Customer Info
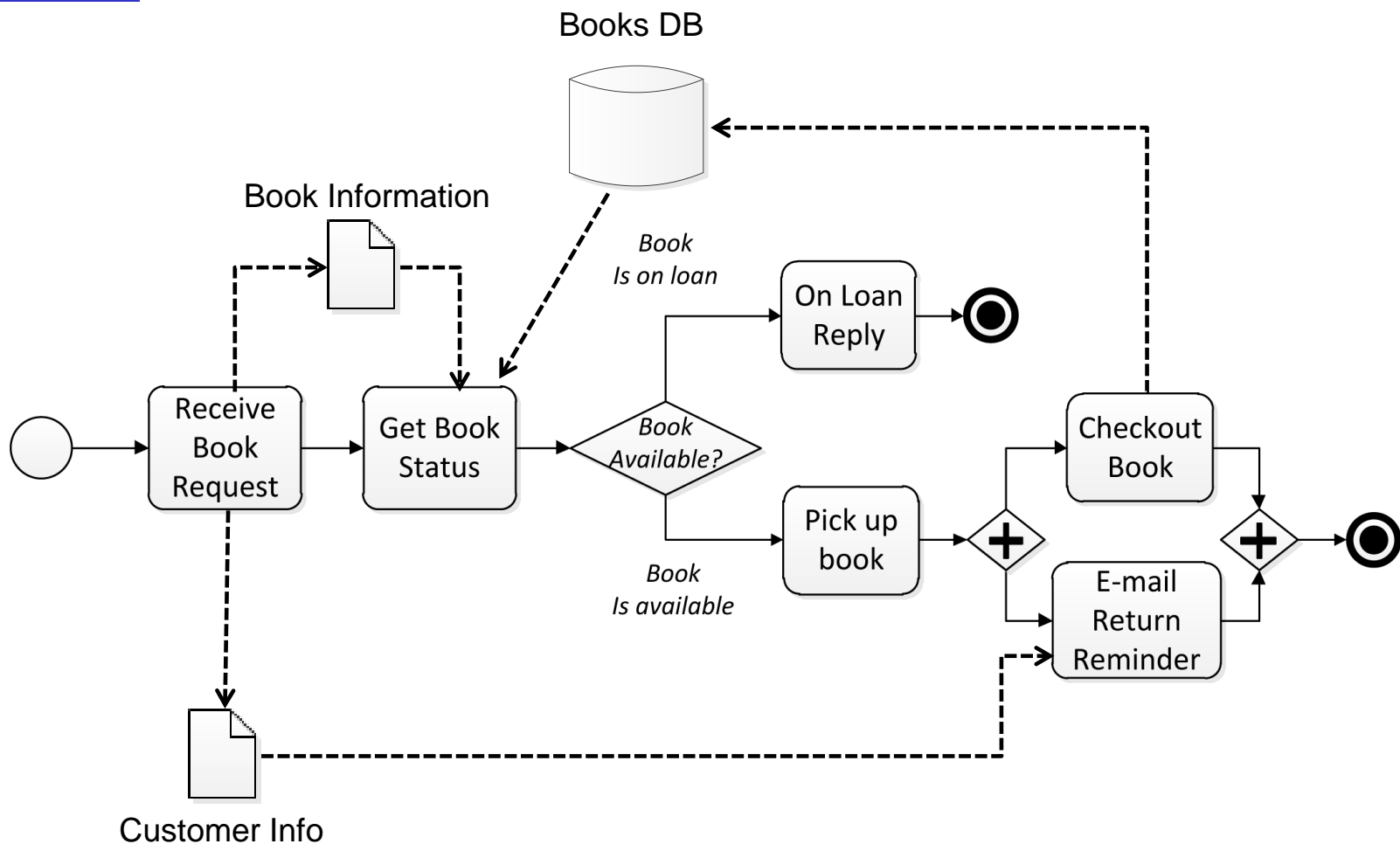
New Data Objects in BPMN 2.0
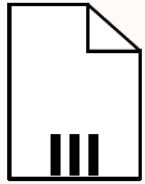
- A **Collection Data Object** represents a collection of information, e.g., a list of order items.
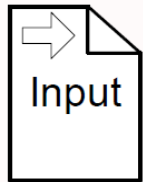
- A **Data Input** is an external input for the entire process. A kind of input parameter.
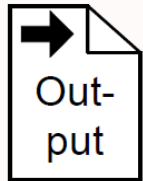
- A **Data Output** is data result of the entire process. A kind of output parameter.

- A **Data Store** is a place where the process can read or write data, e.g., a database or a filing cabinet. It persists beyond the lifetime of the process instance.

BPMN Elements – Recap

Connections
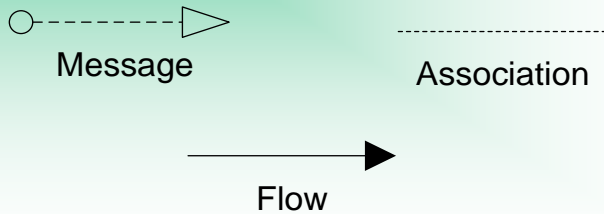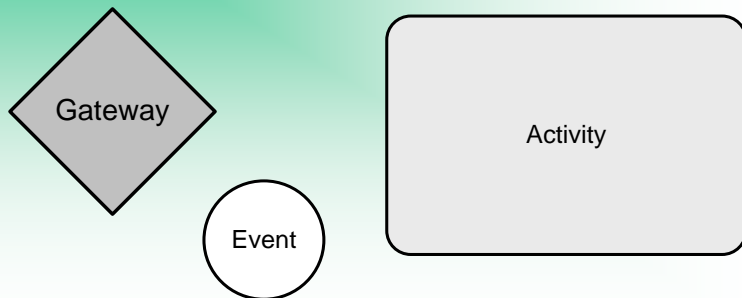
Message

Association

Flow

Swimlanes

Pool

Lane

Flow Objects

Gateway

Activity

Event

Artifacts

Text Annotation

Data Object

Data Store
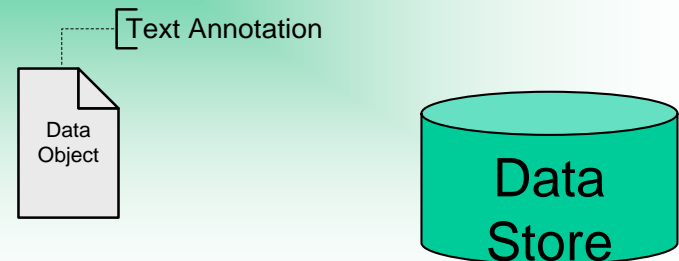
Topics

- Process Modeling
- BPMN Background
- Basic Concepts
- **Advanced Concepts**
- Conclusions

Categories of Processes

- BPMN 2.0 supports four main categories of Processes

    1. **Orchestration** : They represent a *specific business or **organization's view of the process***. It describes how a single business entity (i.e., a process *participant,* such as a buyer, seller, shipper, or supplier) goes about things. **A BPMN diagram may contain more than one orchestration**. If so, each orchestration appears within its own *pool*. ***Each orchestration can only represent one participant***.

    2. **Collaboration** : It is merely a *collection of participants* and their *interaction*.

    3. **Choreography** : They represent the *expected behavior* between two or more business participants.

   4. **Conversation** : The logical relation of message exchanges.

Specialized types of tasks

User	• There are 7 specialized types of tasks (with different markers):
Receive	• **None** : A generic or undefined task.
Send	• **User** : A task where a human performer carries out the task with the assistance of a software application.
Service	• **Receive** : Waits for a message to arrive from an external participant (relative to the Business Process). Once received, the Task is complete.
Script	• **Send** : Dispatches a message to an external participant.
Manual	• **Service** : Links to some sort of service, which could be a web service or an automated application.

- There are 7 specialized types of tasks (with different markers):
- **None** : A generic or undefined task.
- **User** : A task where a human performer carries out the task with the assistance of a software application.
- **Receive** : Waits for a message to arrive from an external participant (relative to the Business Process). Once received, the Task is complete.
- **Send** : Dispatches a message to an external participant.
- **Service** : Links to some sort of service, which could be a web service or an automated application.
- **Script** : Performs a modeler-defined script.
- **Manual** : A non-automated task that a human performer undertakes outside of the control of the workflow or PMS engine.

BPMN Activities

Task
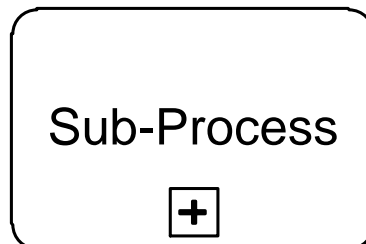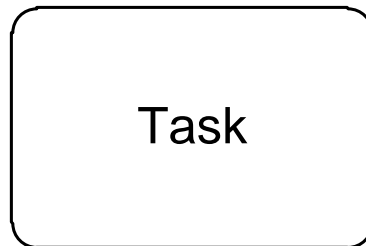
Sub-Process
+

A sub-process has a "*plus sign*" placed in the lower center of the shape, that indicates it can be opened for more details.

- An **activity** is work that is performed within a business process.
- It can take some time to perform, and involves one or more resources from the organization.
- It usually requires some type of *input* and produces some sort of *output.*
- An activity can be **atomic** (known also as a *task*) or **compound** (*non atomic*, in the sense you can **drill down to see another level of the process below**).
- A **task** is used **when the work in the Process is not broken down to a finer level of detail**.
- The compound type of an Activity is called a *sub-process.*

72

Types of sub-processes

Collapsed
sub-process
+

Manage
Payment
+
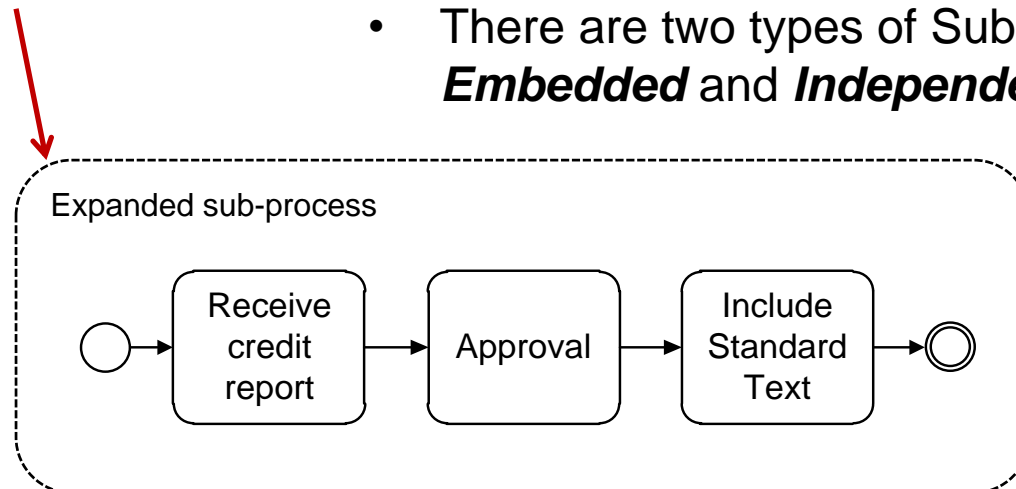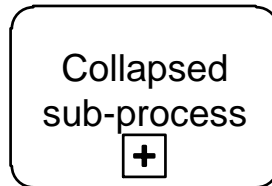
- Sub-processes enable hierarchical process development.
- We refer to a Process that contains a Sub-Process as the **Parent Process** for the Sub-Process. Conversely, the Sub-Process is the **child Process** of the Process that contains it.
- For an *expanded* version of a sub- process, the details (i.e., another process) are visible within its boundary.
- There are two types of Sub-Processes: *Embedded* and *Independent*

Expanded sub-process

Receive credit report → Approval → Include Standard Text

73

Embedded sub-processes

- A modeled process that is actually part of the *Parent Process.* Embedded Sub-Processes ***are not reusable*** by other processes. All "process relevant data" used in the parent process ***is directly accessible by the embedded sub-process*** (since it is part of the parent).
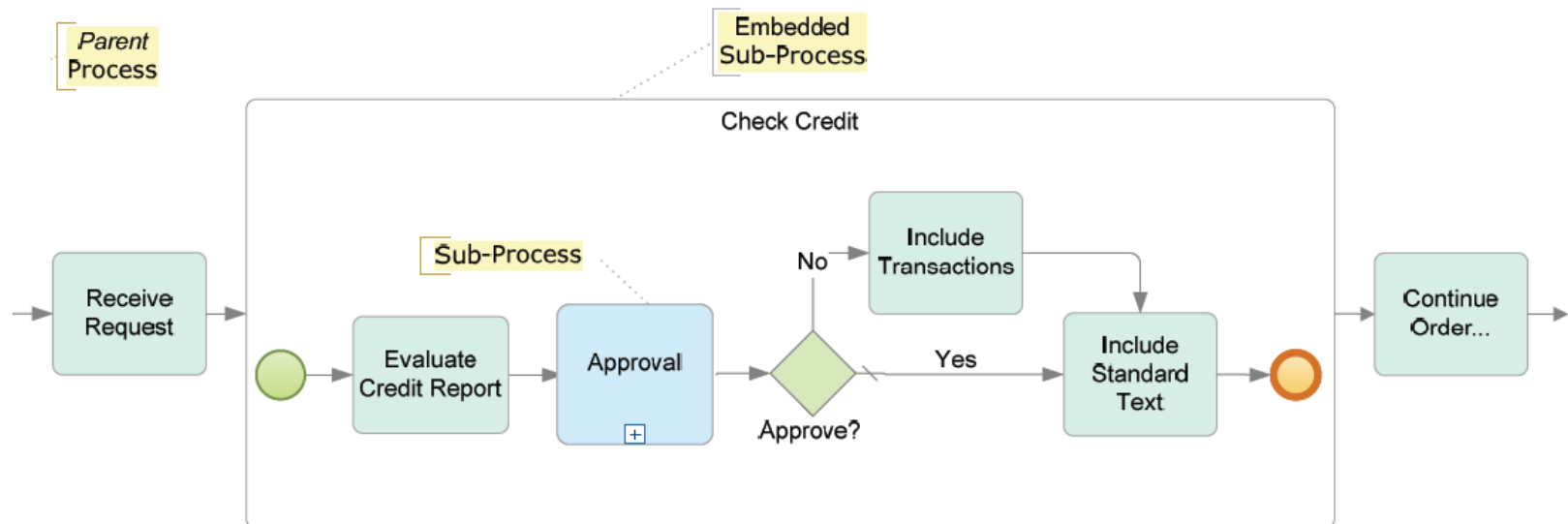
- An important characteristic of an embedded sub-process is that it can only begin with a None Start Event—i.e., without an explicit *trigger* such as a message.
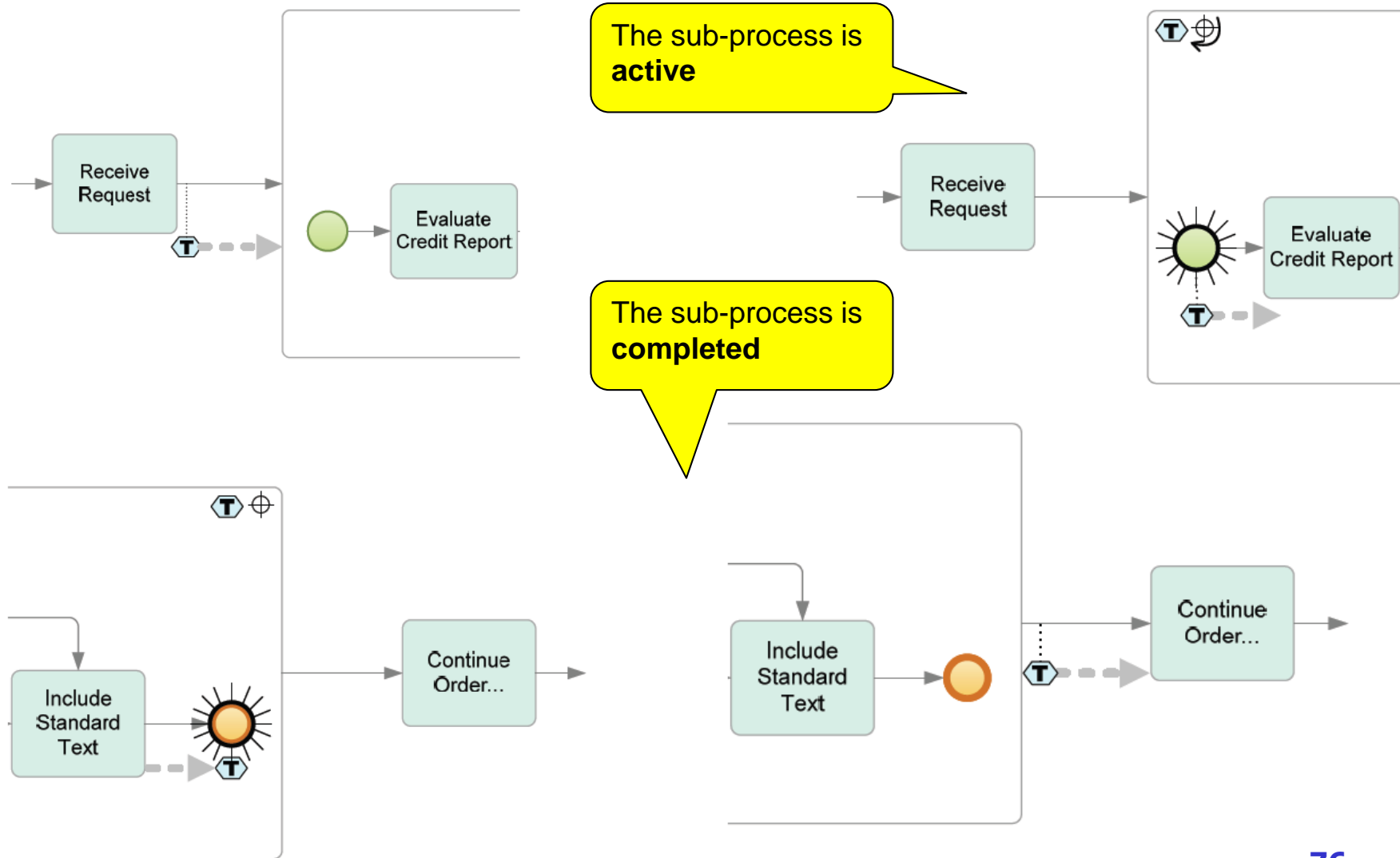
Independent sub-processes

- A separately modeled process that could be used in multiple contexts (e.g., *checking the credit of a customer*). The "process relevant data" of the parent (calling) process is not automatically available to the sub-process. ***Any data must be transferred specifically***, sometimes reformatted, ***between the* parent** *and* **sub-process***.*
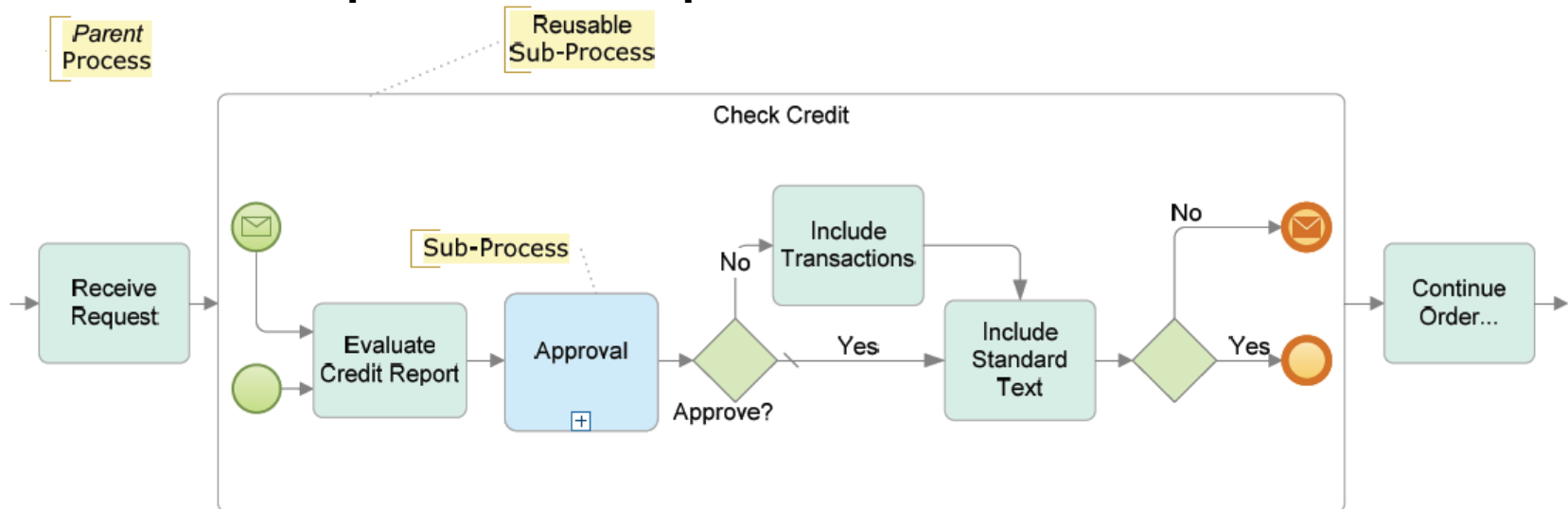
- Transferring data from the parent Process to the reusable sub-process will rely on a "mapping" between the data elements of the two levels.
- Just like an embedded sub-process, an independent sub-process **must have** a None Start Event. **Independent sub-processes maximize reuse**.

Behaviour across process levels

The sub-process is **active**
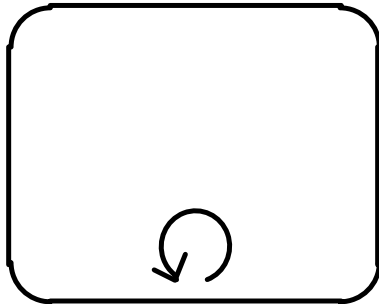
The sub-process is **completed**

We can simplify the model by hidding the content of its subprocesses. We are **collapsing** the sub-processes in compound activities.
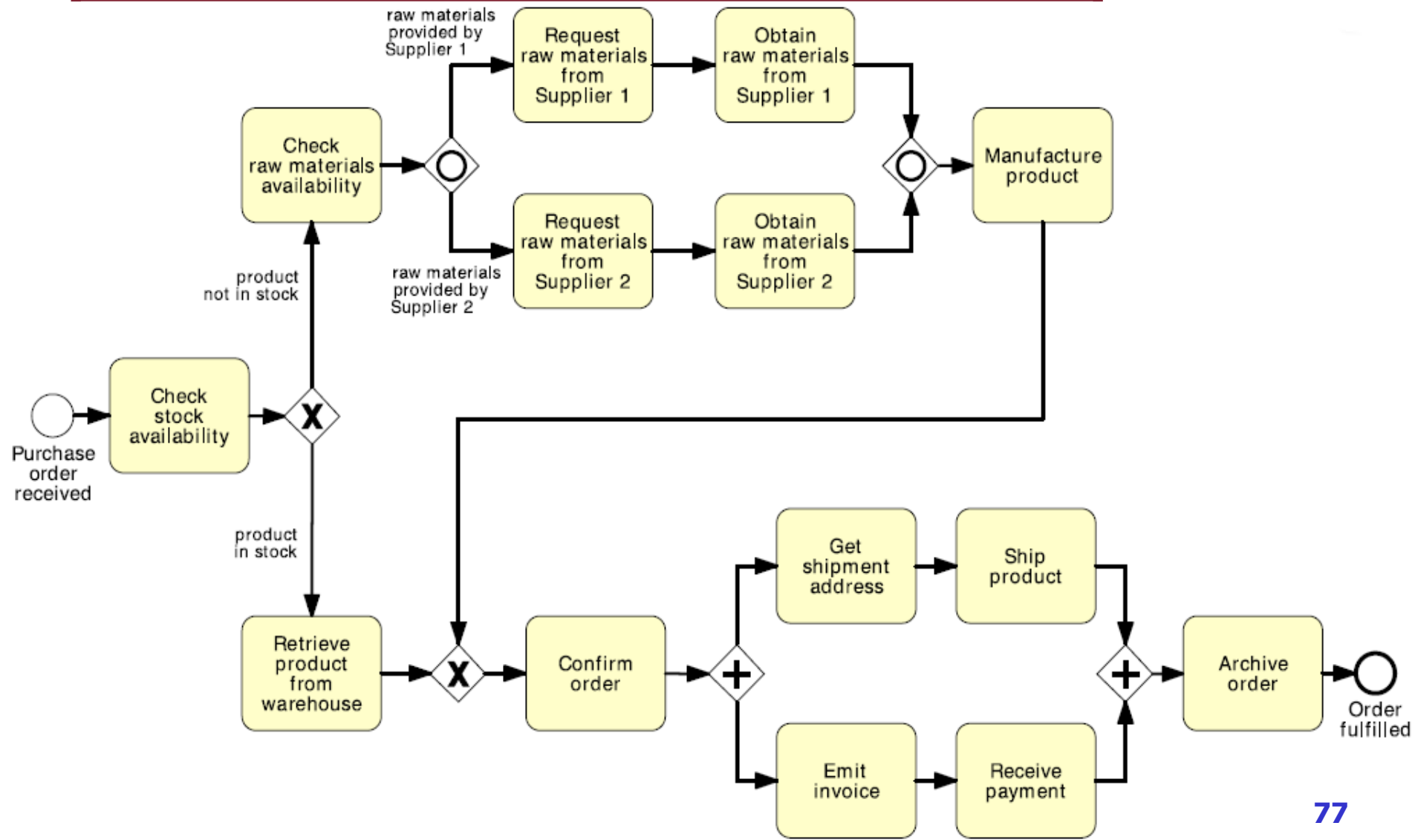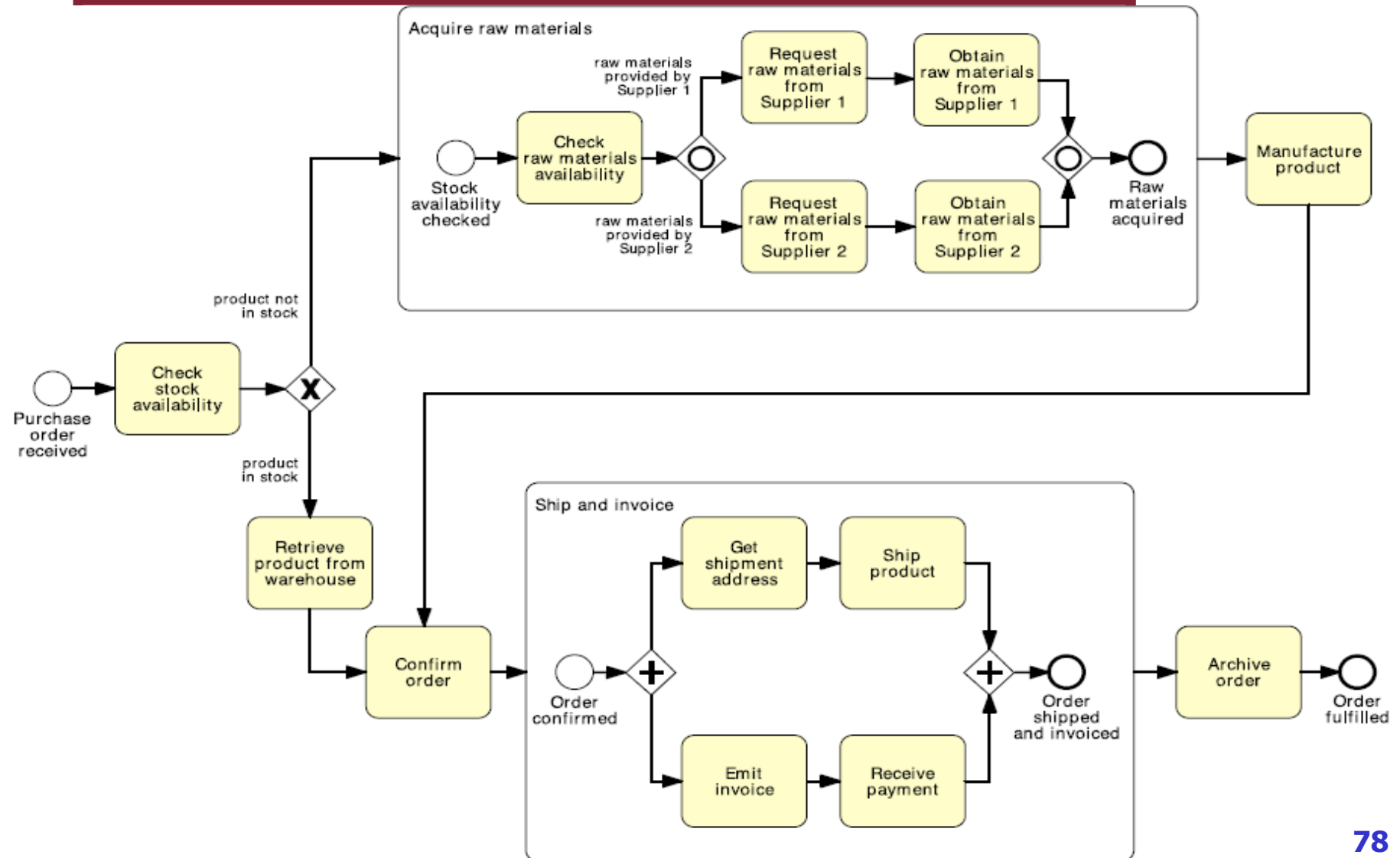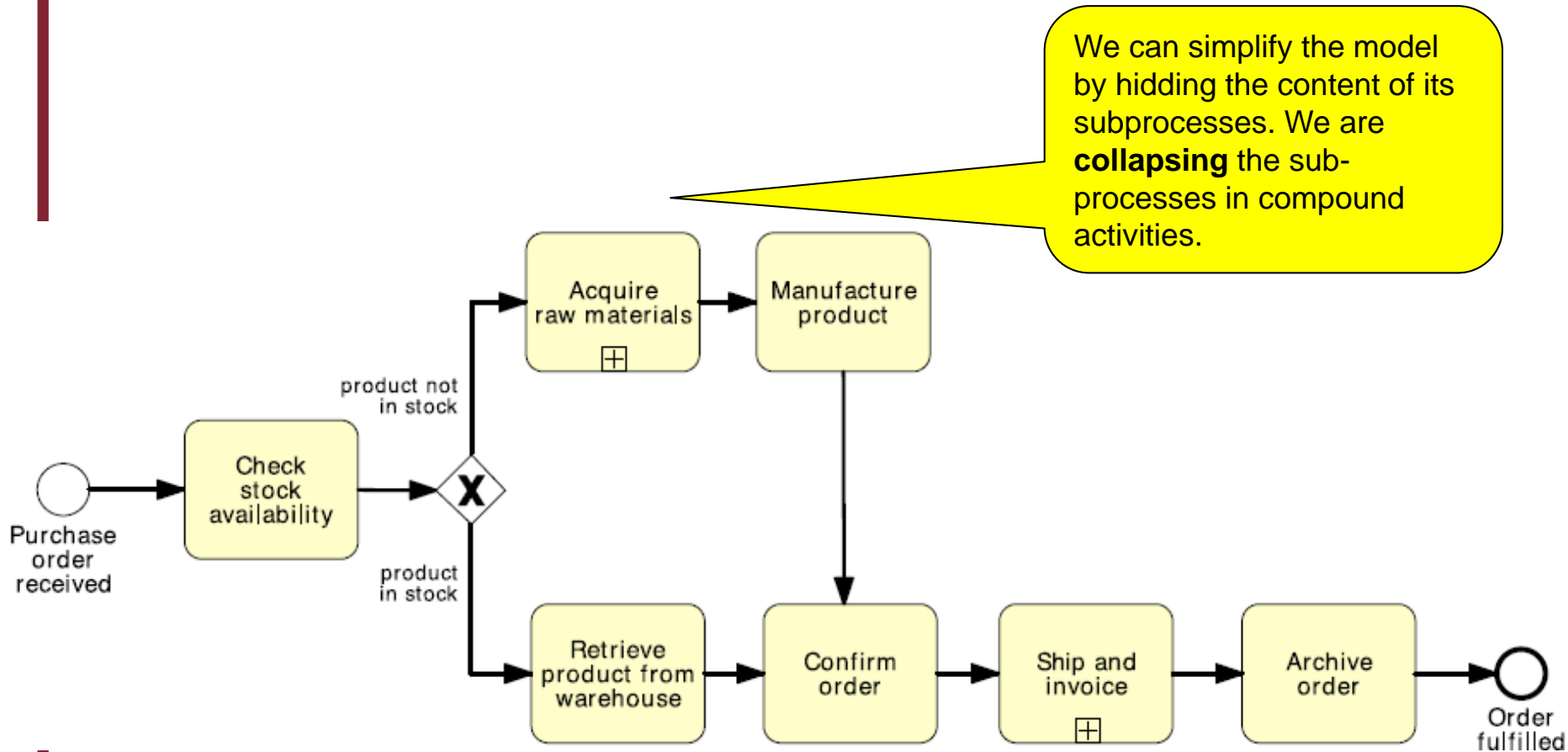
Looping

- On an activity, it is possible to define a **loop condition** that determines the number of times to perform that activity. There are two variations for activity looping :
  - *While Loop (or **While-Do**) -* The loop condition is checked *before* the activity is performed. If the loop condition turns out to be *true*, then the activity is performed. *If not*, the activity completes and the Process continues (a token moves down the outgoing sequence flow), even if the activity was never performed. The cycle of checking the loop condition and performing the activity continues until the loop condition is *False*.
  - ***Until Loop (or Do-While) -*** The loop condition is checked *after* the Activity is performed.

- Using activity attributes, it is possible to set the ***maximum number of loops (loop maximum)*** for both while and until loops. After the activity has reached the loop maximum, it will stop (even if the loop condition is still true).

Example
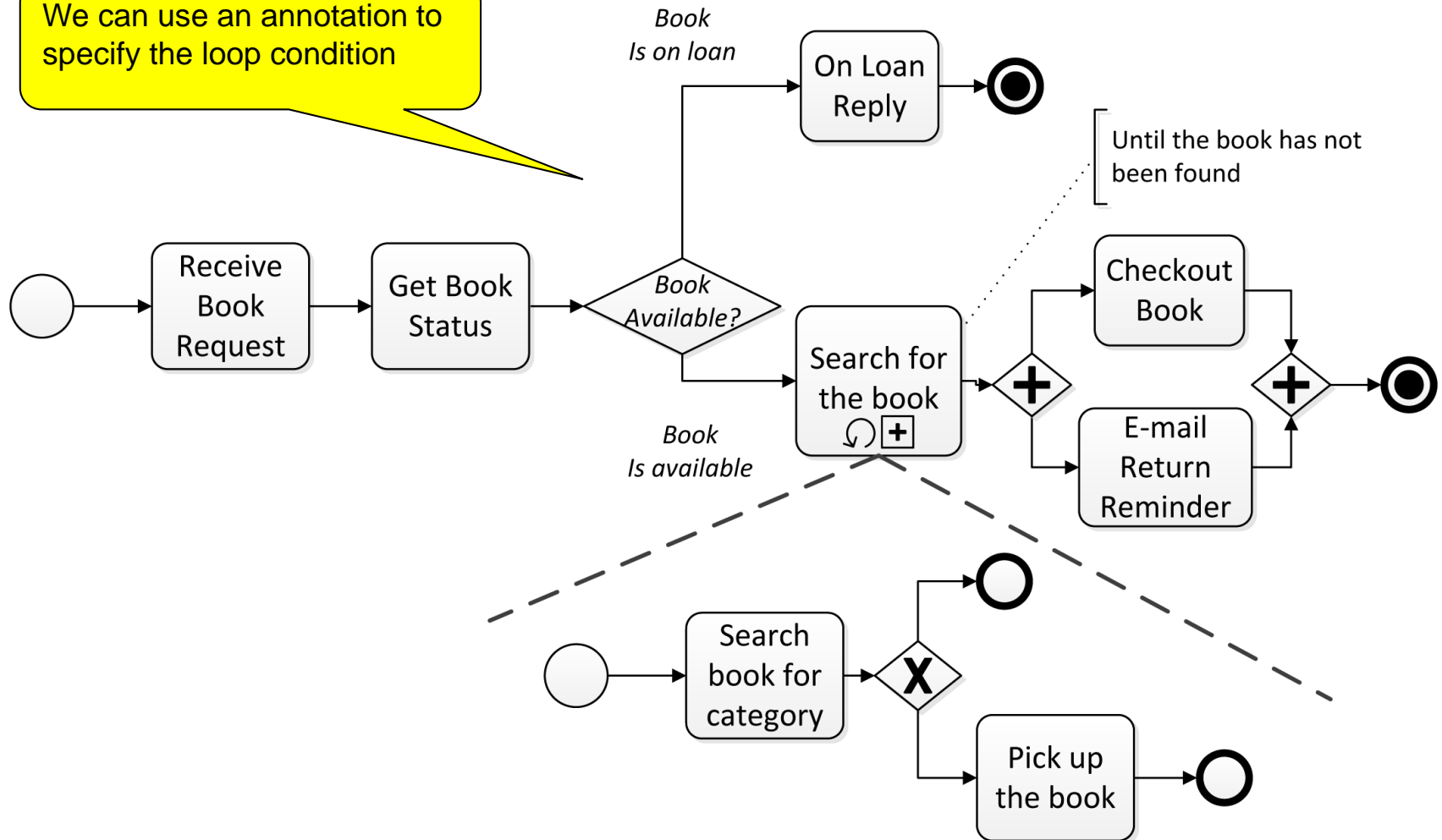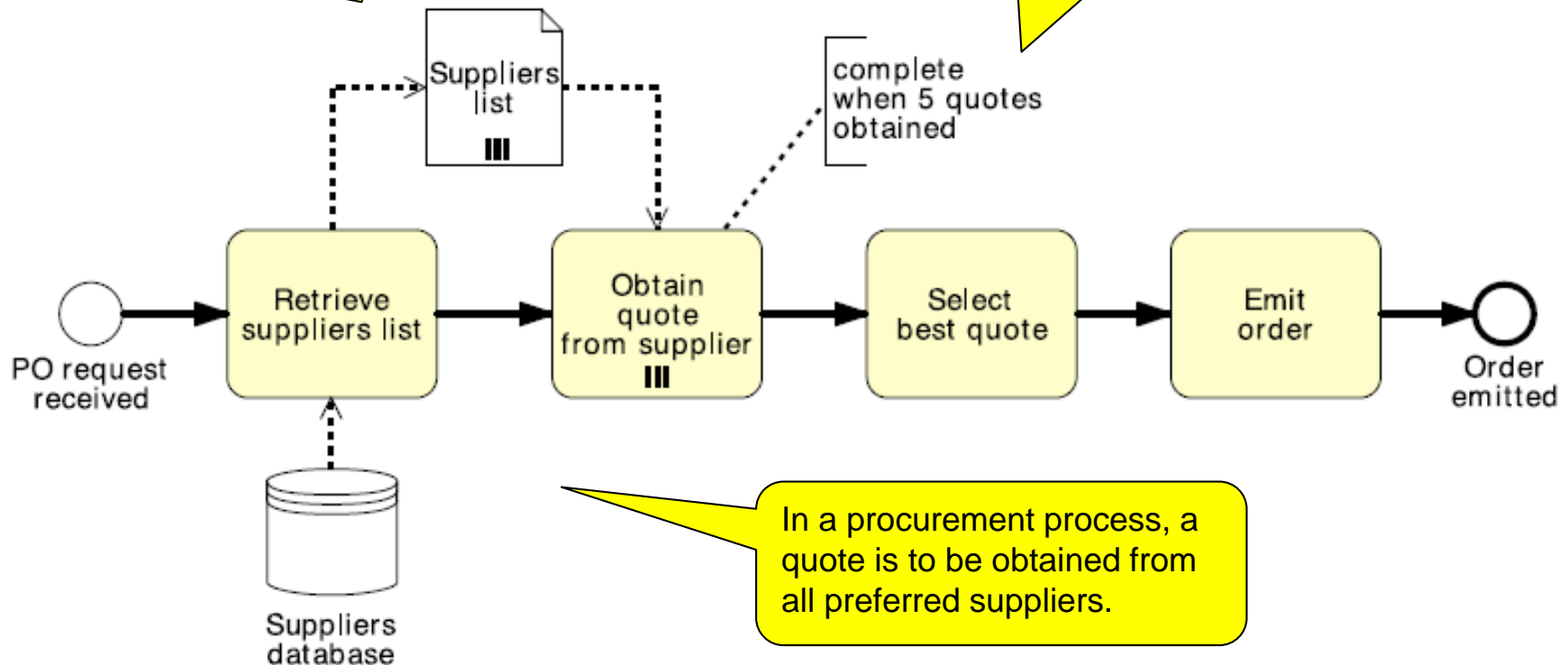
Multi-Instance Activities
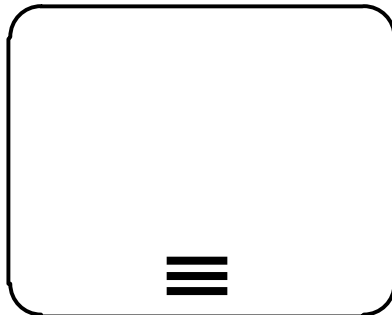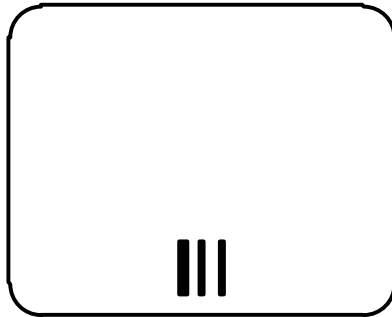
- Activity to be performed **many times** **concurrently** with **different data sets**.
- The value of the loop condition attribute determines the number of times that the activity is performed.
- The key point to understand that the activity **does not cycle around**; **each activity execution is distinct from the others**.
- The individual instances of a Multi-Instance Activity might occur in *parallel* or in *sequence*.
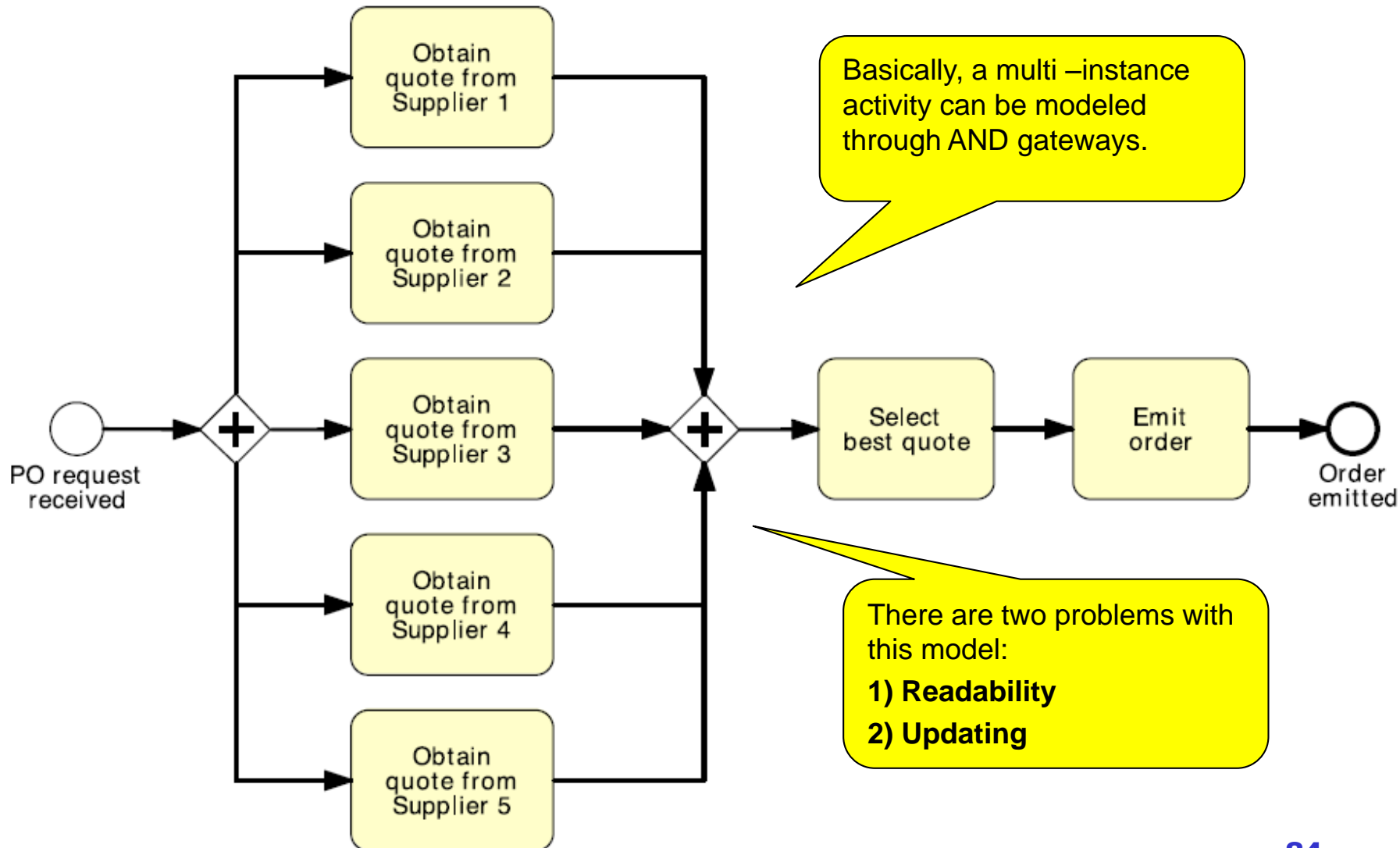
82

Example

This is a collection of *similar* data objects (a list of suppliers, in our case). When a collection is used as input to a multi-instance activity, **the number of items in the collection determines the number of activity instances to be created.**

After all quotes have been received, they are evaluated and the best quote is selected.

In a procurement process, a quote is to be obtained from all preferred suppliers.

Example

Obtain quote from Supplier 1

Obtain quote from Supplier 2

Obtain quote from Supplier 3

Obtain quote from Supplier 4

Obtain quote from Supplier 5

PO request received

Select best quote

Emit order

Order emitted

Basically, a multi –instance activity can be modeled through AND gateways.

There are two problems with this model:
1) **Readability**
2) **Updating**

After a car accident, a statement is required from two witnesses out of the five that were present, in order to lodge the insurance claim.

As soon as the first two statements are received, the claim can be lodged with the insurance company without waiting for the other statements.
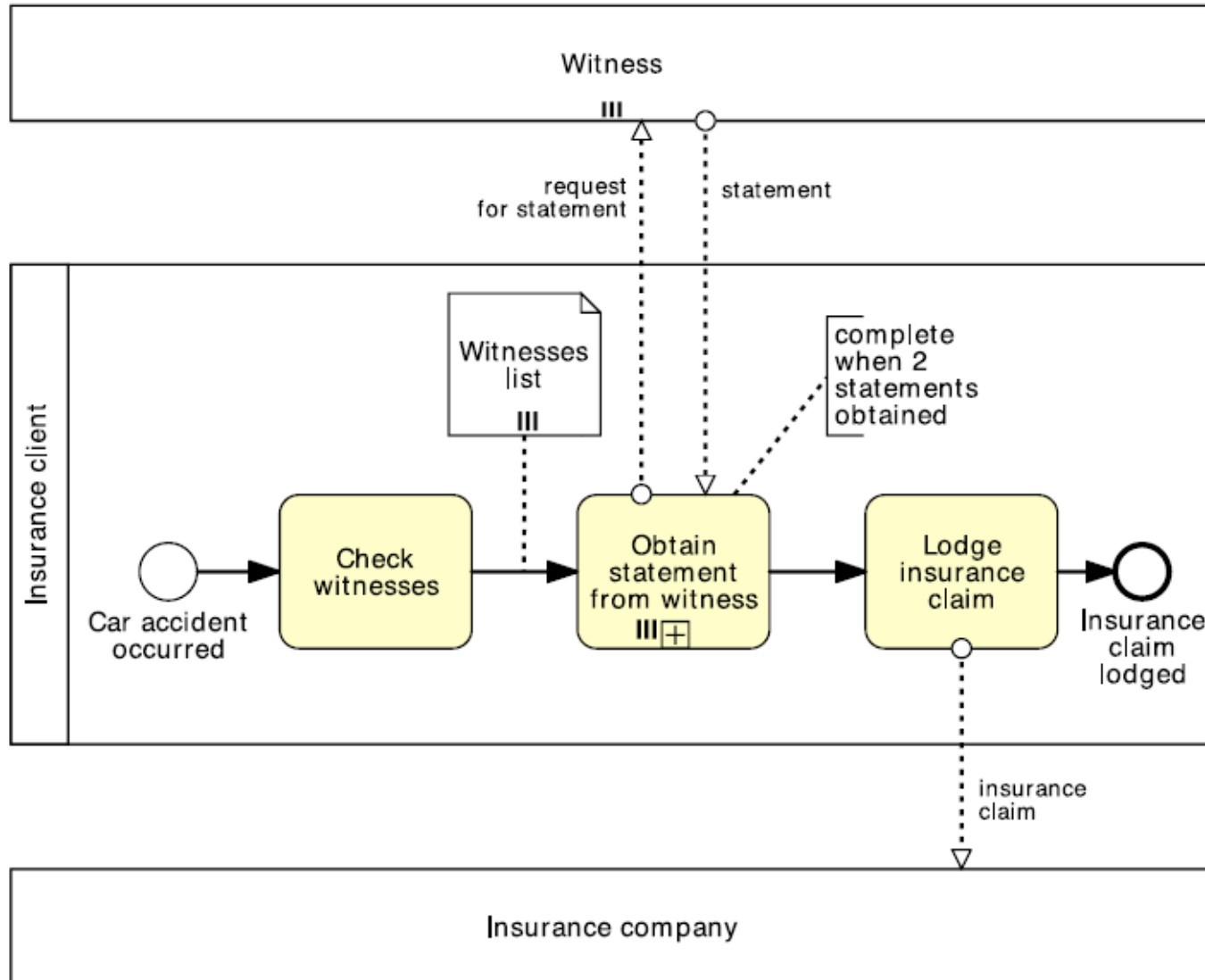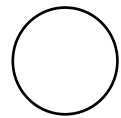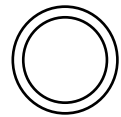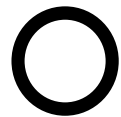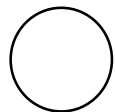
Events

Start

Intermediate

End

- An event is something that "happens instantaneously" during the course of a business process.
- An event may affect the flow of the Process and usually have a *trigger* or a *result*.
- They can start, delay, interrupt, or end the flow of the process.
  – Events are circles and the type of boundary determines the type of Event.
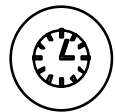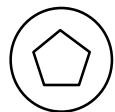
Start Events

None

Message

Timer

Conditional

Signal

Multiple

Parallel
Multiple

- A Start Event shows where a Process can begin.
- A Start Event is a small, open circle with a single, thin lined boundary.
- There are different types of Start Events to indicate the varying circumstances that can trigger the start of a Process.
- These circumstances, such as the arrival of a message or a timer "going-off," are called **triggers**.
- A Start Event can only have **outgoing sequence flows.**
- Trigger-based Start Events can only feature in top-level processes (hence they are never used in sub-processes).

Start Events Behaviour

An illustration of an Event being "Triggered"

Test Complete Design
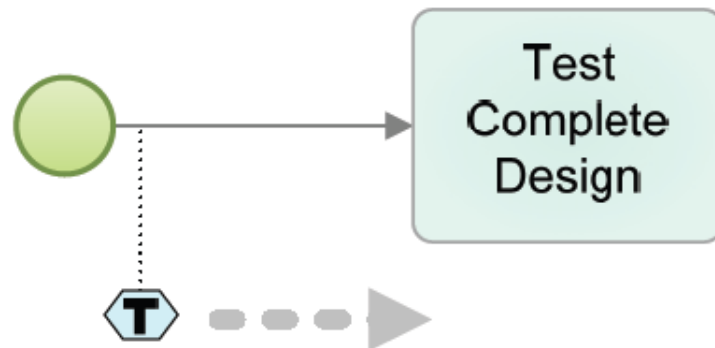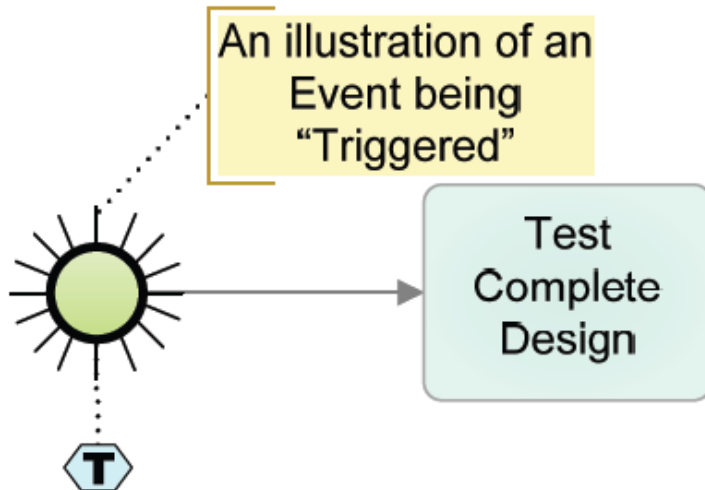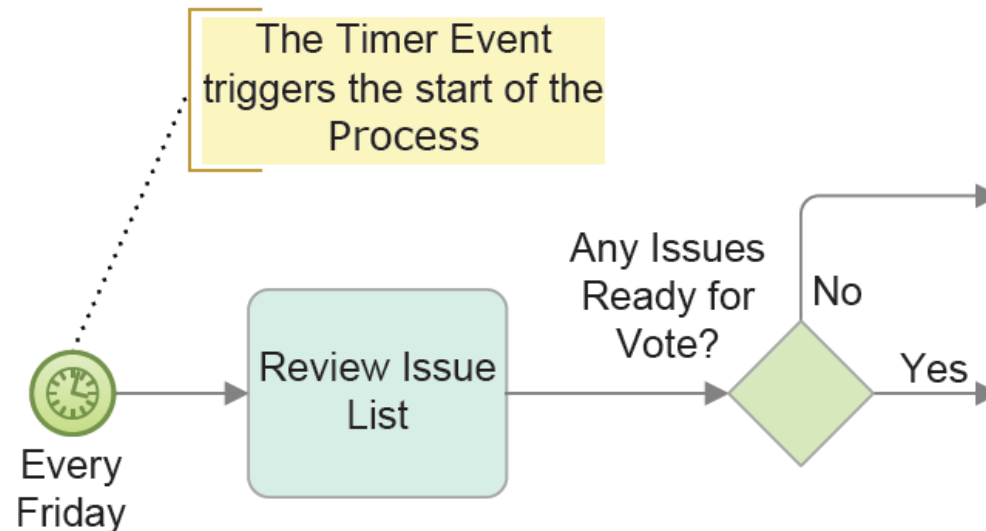
- Start Events are where the flow of a Process starts, and, thus, are where **tokens are created.** When a Start Event is triggered, the token is generated.

Test Complete Design

- Immediately after the Start Event triggers and the token generated, the token will then exit the Start Event and travel down the outgoing Sequence Flow.

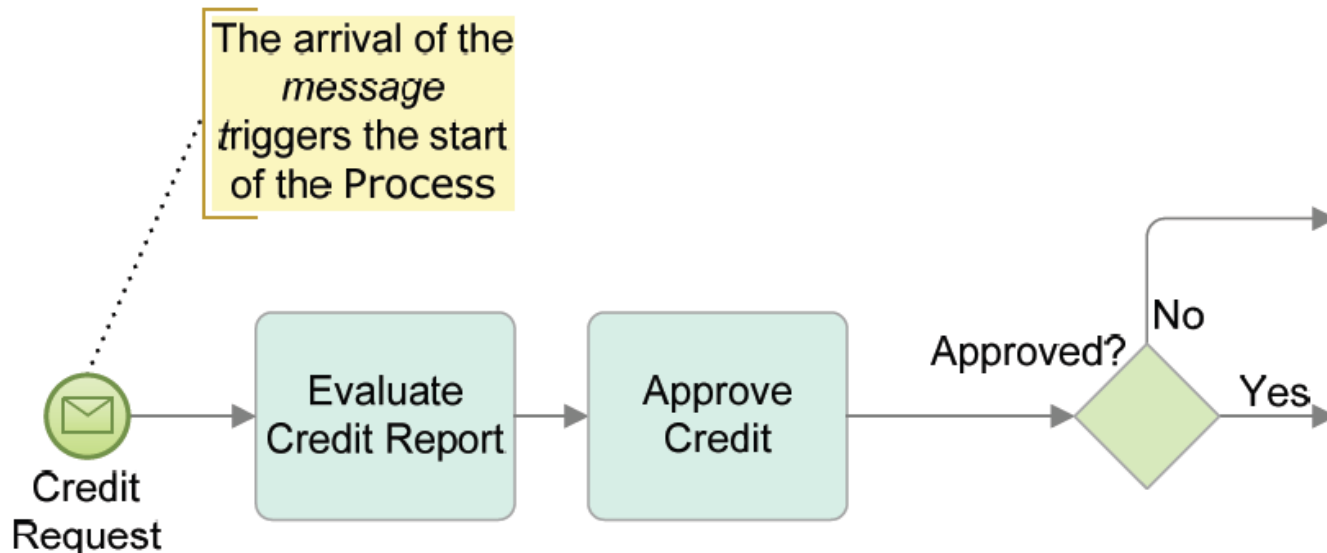# Timer Start Event

- The **Timer Start Event** indicates that the Process is started (i.e., triggered) when a specific time condition has occurred.
    - This could be a specific date and time (e.g., January 1, 2009 at 8am) or a recurring time (e.g., every Monday at 8am).

The Timer Event triggers the start of the Process

Every Friday → Review Issue List → Any Issues Ready for Vote? — No / Yes

Message Start Event

- The **Message Start Event** represents a situation where a Process is initiated (i.e. triggered) by the **reception of a message**.

- A message is a direct communication between two business participants. **These participants must be in separate pools** (they cannot be sent from another lane inside a single pool).

The arrival of the *message* triggers the start of the Process

Credit Request → Evaluate Credit Report → Approve Credit → Approved? — No / Yes

Signal Start Event

- It indicates that the Process is started (i.e. triggered) when a signal is detected.

- This signal was a broadcast communication from a business participant or another Process. Signals have no specific target or recipient - i.e., all processes and participants can see the signal and it is up to each of them to decide whether or not to react.

  - Unlike messages, signals can operate within a Process (perhaps between a Sub-Process and its parent calling Process).

The detection of the Signal Triggers the Start of the Process
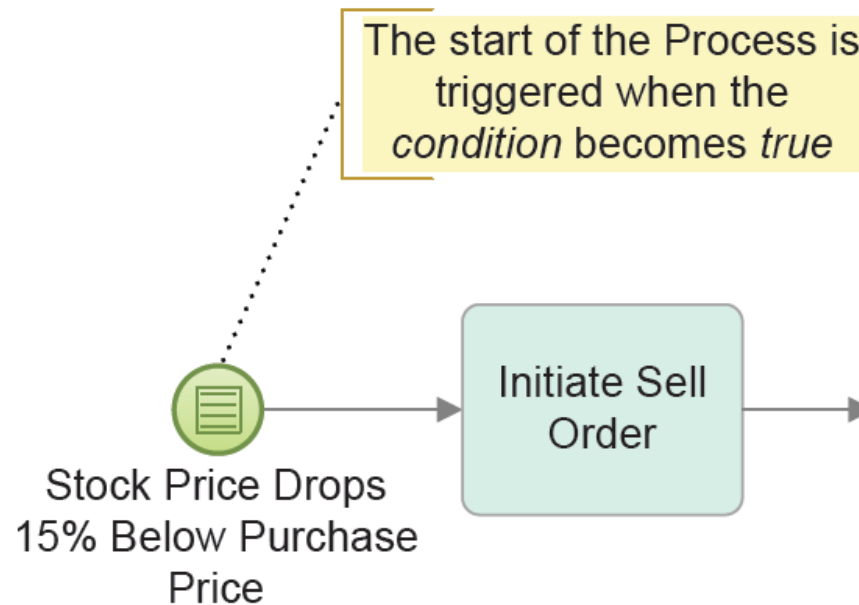
Accept Order → Validate Order → Order Validated → Accept / Reject
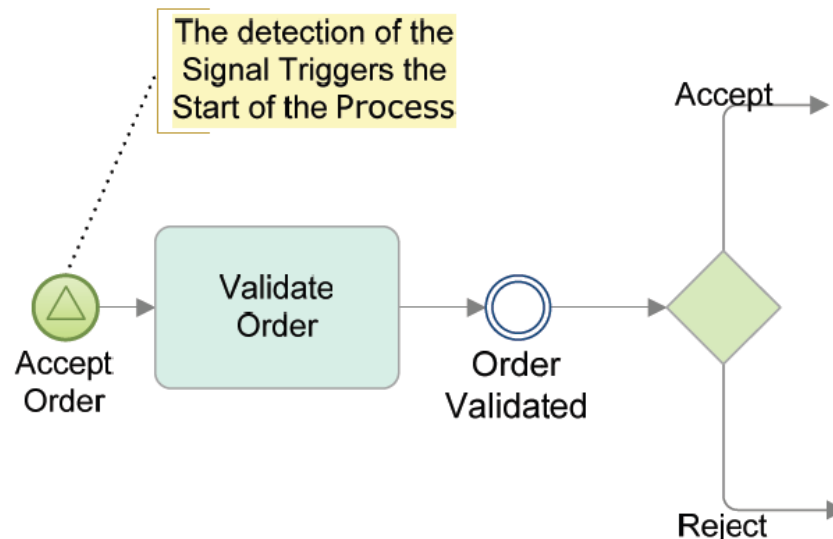
Conditional Start Event

- The Conditional Start Event represents a situation where a Process is started (i.e., triggered) when a pre-defined *condition* becomes *true.*
- A condition is used to define the details of the change in data that is expected.
  - The condition for the Event must become false and then true again before the Event can be triggered again.
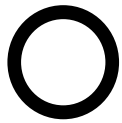
The start of the Process is triggered when the *condition* becomes *true*

Stock Price Drops 15% Below Purchase Price

Initiate Sell Order

Multiple and Parallel Multiple Start Events

- The **Multiple Start Event** represents a collection of two or more Start Event *triggers.* The triggers can be any combination of messages, timers, conditions, and/or signals.
  - Any one of those triggers will instantiate the Process.
  - **If one of the other triggers occurs**, or the same trigger occurs again, then another Process instance is generated.

- For the **Parallel Multiple Start Event** a combination of *triggers* is required before the process can be instantiated*.

End Events

None

Message

Signal

Multiple

Terminate

- There are different types of End Events that indicate different categories of *results* for the Process.
  - A result is something that occurs at the end of a particular path of the Process (for example, a message is sent, or a signal is broadcast).
  - All End Events are ***throw results***.
- Only incoming sequence flow is permitted – (i.e. sequence flow cannot leave from an End Event).
- A None End Event is always used to mark the end of Sub-Processes (moving from one level up to the next).

End Events Behaviour
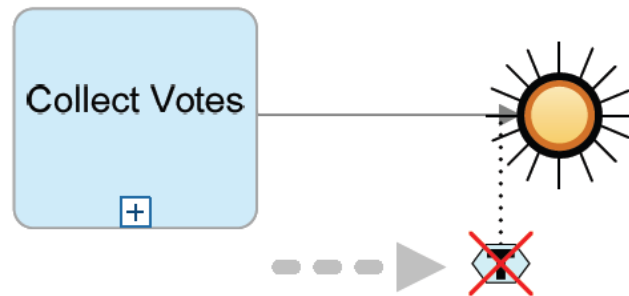
- When a token arrives at an End Event, the result of the event, if any, occurs and **the token is consumed**.
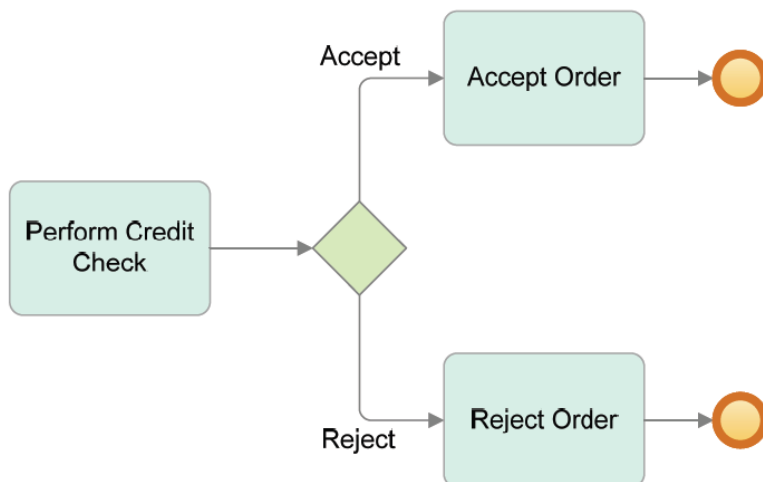
- it is possible to have one or more paths (threads) that continue even after the *token* in one path has reached an End Event and has been consumed.

- If the Process still contains an unconsumed *token, then the* Process is still "active." **After all active paths have reached an End Event, the Process is then complete**.

96

Terminate End Event

- The Terminate End Event will cause the **immediate cessation of the Process instance at its current level and for any Sub-Processes** (even if there is still ongoing activity), but it will not terminate a higher-level parent Process.

When the lower path reaches the Terminate End Event, the work of the upper path will be stopped, thereby stopping the infinite loop.

7 Days

Send Reminder

Send Out Application Form Pack

Receive Application Form

Terminate End Event
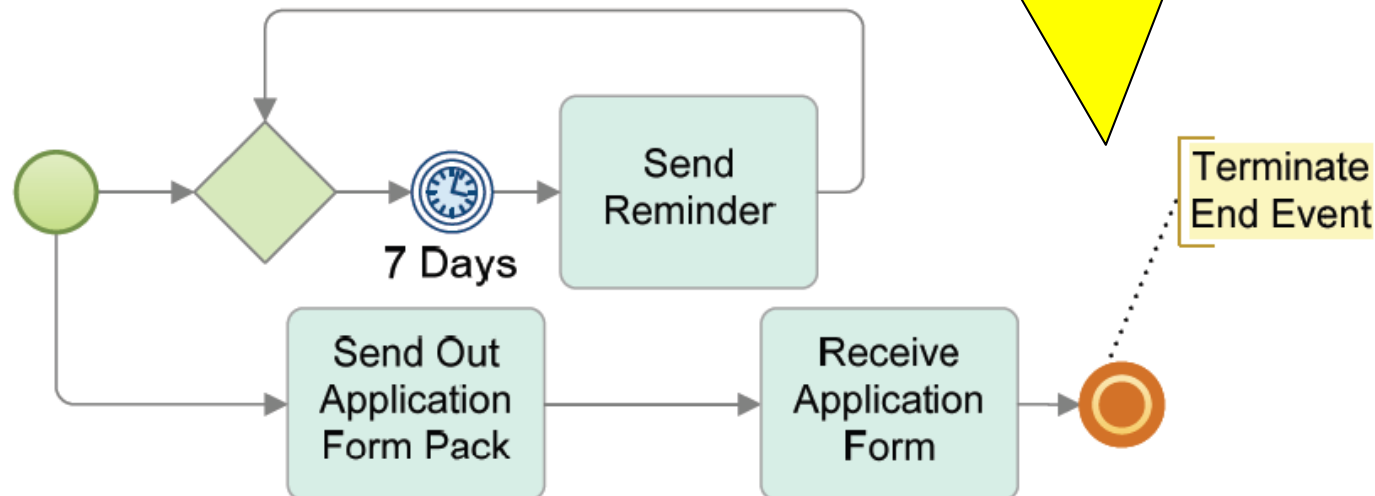
Intermediate Events

Catching Throwing
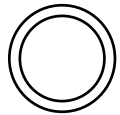
None

Timer

Message

Signal

Conditional
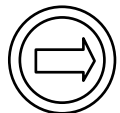
Link
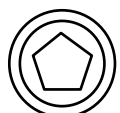
Multiple

Parallel
Multiple

- An Intermediate Event indicates where something happens/occurs after a Process has started and before it has ended.

- **They may also interrupt the normal processing of an activity.**

- Each type of Intermediate Events can either *throw* or *catch* the event.

- A *catching Intermediate Event* waits for something to happen (i.e., wait for the circumstance defined on the *trigger*).

- A *throwing Intermediate Event* immediately fires (effectively creating the circumstance defined on the trigger).

98

Catch Intermediate Events Behaviour
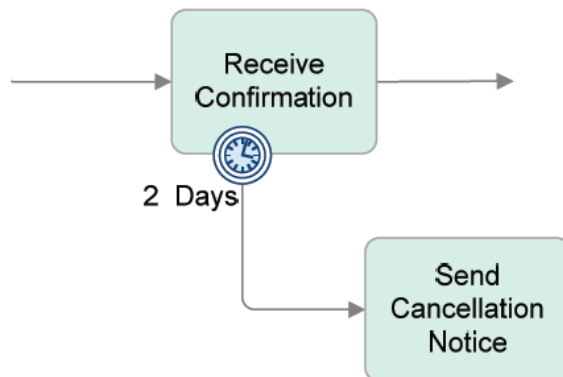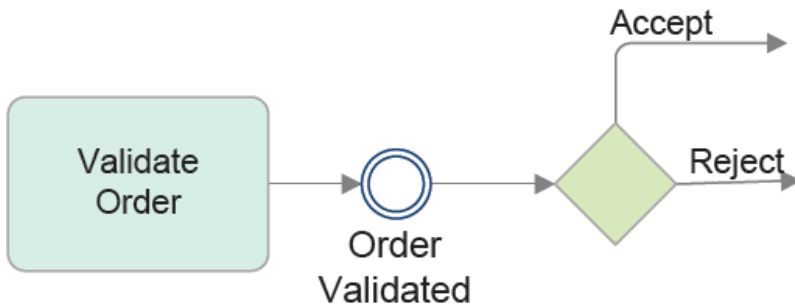
- A token arriving at a catch Intermediate Event **would wait until the trigger occurs**. Then the token would leave immediately and move down the outgoing Sequence Flow.

- **A Catching Intermediate Event** (*except than the Link Event*) **can also be attached to the boundary of an Activity.**

- When the activity starts, so does the timer.

  - **If the activity finishes first**, then it completes normally and the Process continues normally.

  - **If the timer goes off before the Activity is completed**, the Activity is immediately interrupted and the Process continues down the sequence flow from the Timer Intermediate Event.

Throw Intermediate Events Behaviour
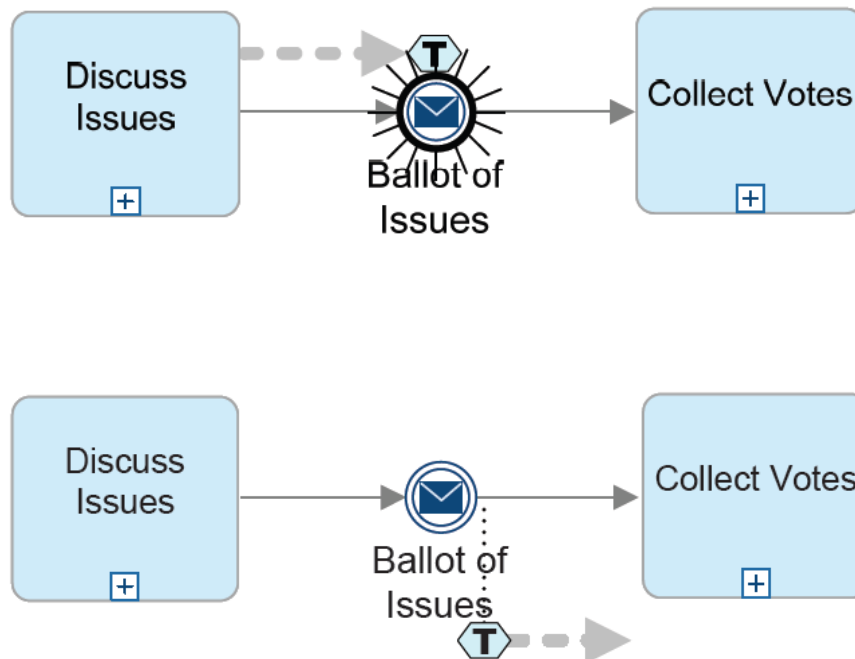
- A token arriving at a throw Intermediate Event would immediately fire the trigger. It would then leave immediately and travel down the outgoing Sequence Flow.

- A Throwing Intermediate Event **can not be attached** to the boundary of an Activity.

Intermediate Events Behaviour
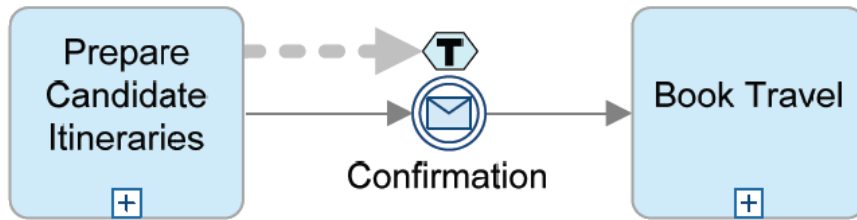
- *When a token arrives at a catching Message Intermediate Event, the Process pauses until a message arrives.*

- *When a token arrives at a throwing Message Intermediate Event, it immediately triggers the Event, which sends the message to a specific participant.*
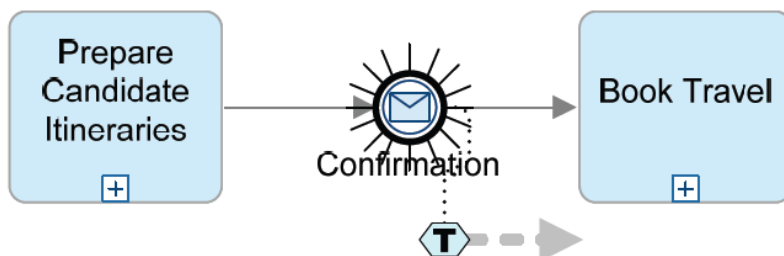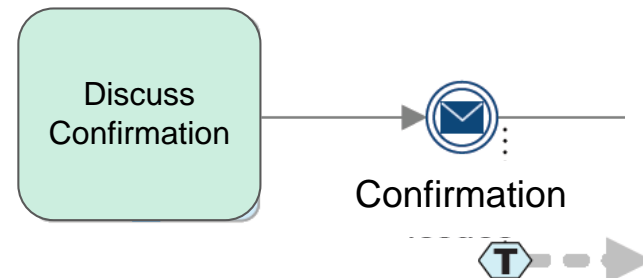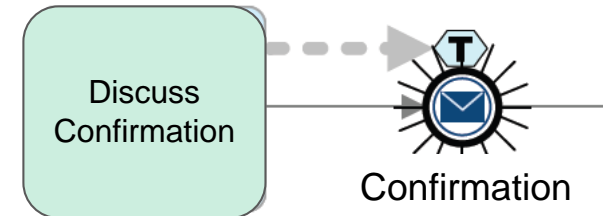
If the *token is waiting at the Intermediate Event and the message* arrives, then the Event triggers.

An Example
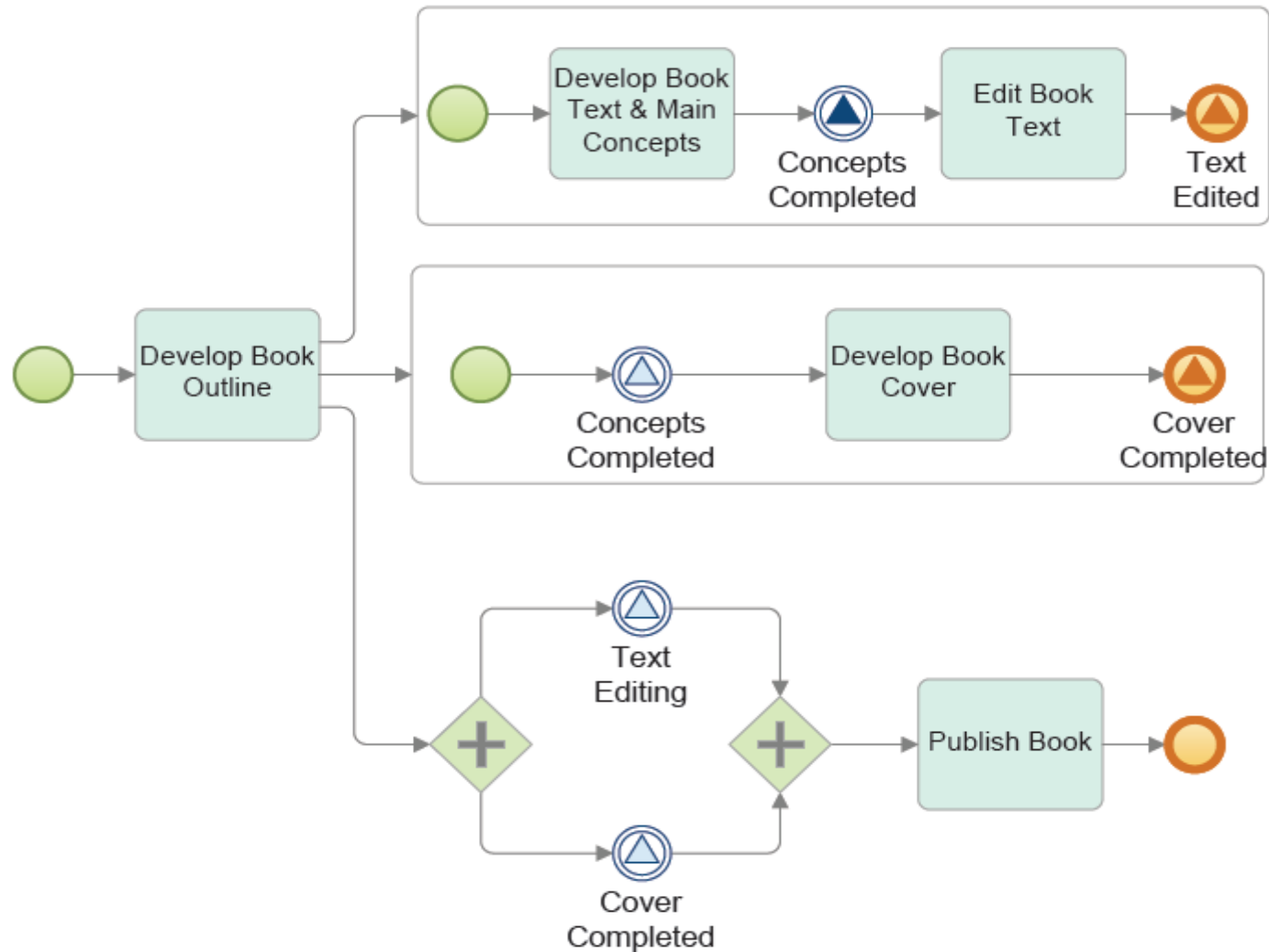
Link Intermediate Event

- Link Intermediate Events are always used in pairs, with a *source and a target Event.* To ensure the pairing, both the source and target Link Events must have the same label.

- Using a pair of Link Events creates a *virtual Sequence Flow.*

- There can be only one Target Link Event, but there may be multiple Source Link Events paired with the same catching Link Event.

- Once the Source Link Event is triggered (the throw), the token immediately jumps to the catching (Target) Link Event.

- The arrival of the token at the Target Link Event immediately triggers the Event..
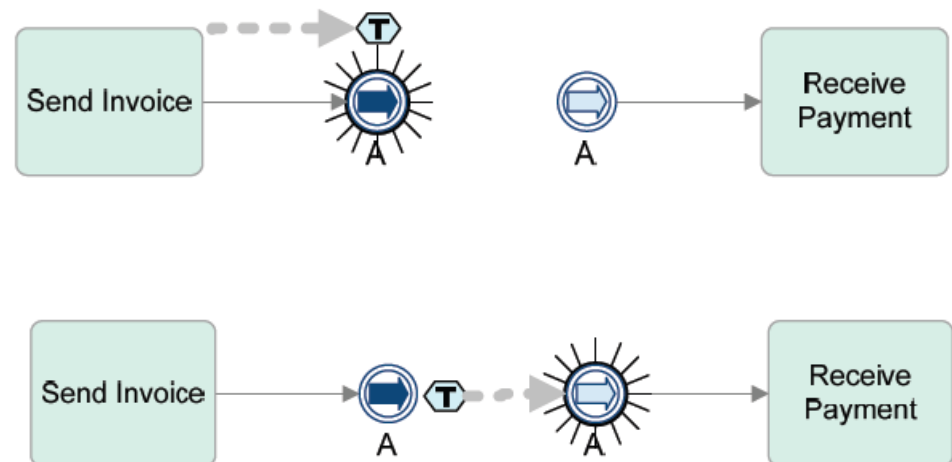
Example of a Multiple Intermediate Event
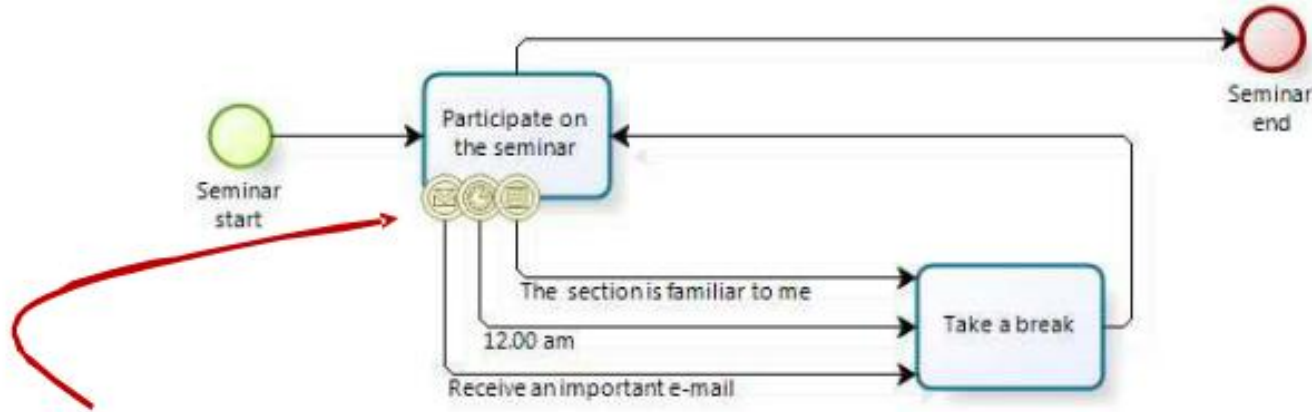
Different triggers or events can cause a break

Is equal to:
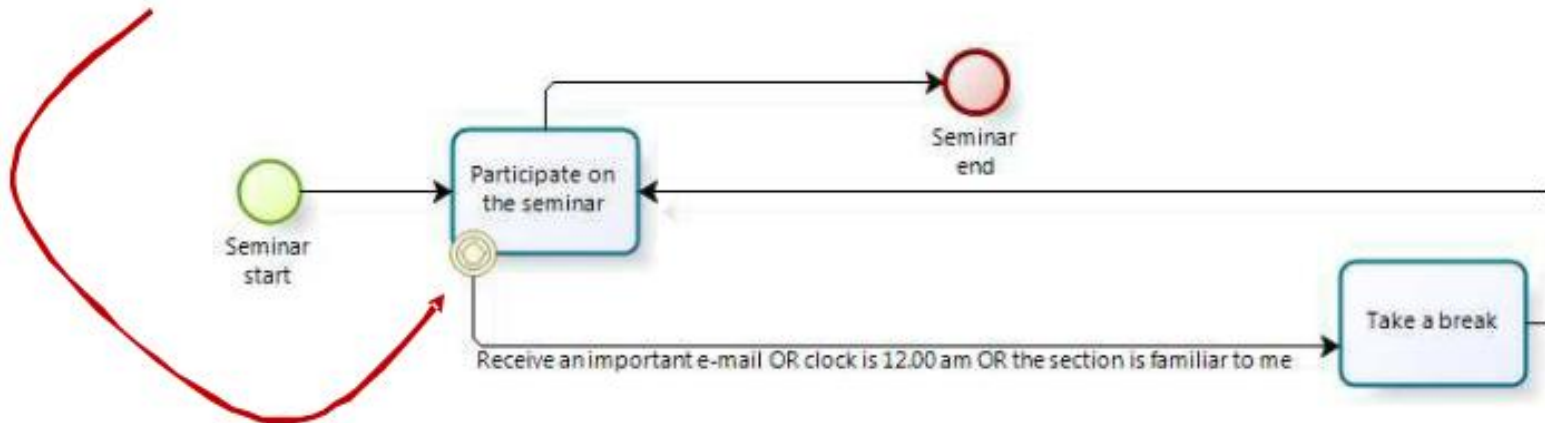
Gateways
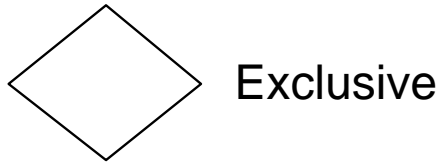
Exclusive
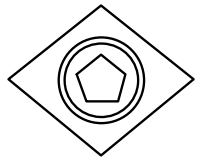
Event
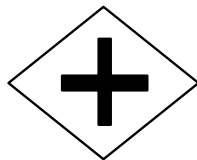
Parallel
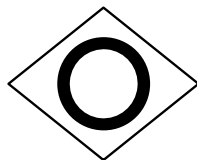
Inclusive
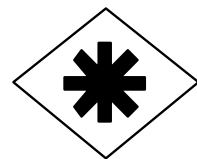
Complex

- Gateways are modeling elements that control how the Process diverges or converges.
- They represent points of control for the paths within the Process.
- They split and merge the flow of a Process (through Sequence Flow).
- Since there are different ways of controlling the Process flow, there are different types of Gateways.
- The type (*splitting and merging) for a* single Gateway **must be matched** - i.e. *a Gateway cannot be Parallel on the input side, and Exclusive on the output side*.

Event-based Exclusive Gateways
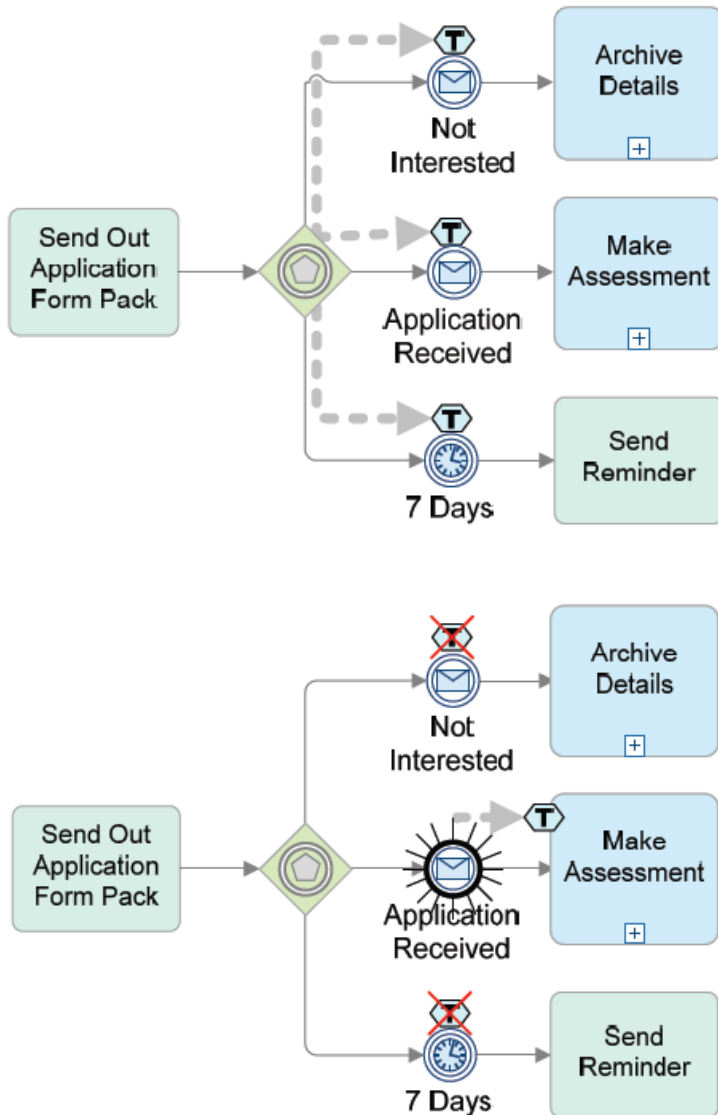
- Event-Based Exclusive Gateways represent an alternative branching point where the decision is based on two or more Events that might occur, rather than data-oriented conditions (as in an Exclusive Gateway).

- These Events, which must be of the **catch variety,** are the first objects connected by the Gateway's outgoing Sequence Flow. The tokens will wait there until one of the Events is triggered.

- The Intermediate Events that are part of the Gateway configuration become involved in a *race condition*. Whichever one finishes first (fires) will win the race and take control of the Process with its token.

- Then the token will immediately continue down its outgoing Sequence Flow, by disabling the other paths.

106

Complex Gateways – Splitting Behaviour

- With a Complex Gateway, Modelers provide their *own expressions that determine the merging and/or splitting behavior* of the Gateway.

- Complex Gateway uses a single outgoing ***assignment*** within the Gateway, rather than a set of separate conditions on the outgoing Sequence Flow.

- An assignment has two parts: a *condition* and an *action*. When an assignment is performed, it evaluates the condition and if *true*, it then performs the *action* such as updating the value of a Process or Data Object property.

- The outgoing assignment may send a token down one or more of the Gateway's outgoing Sequence Flow. The outgoing assignment may refer to data of the Process or its Data Objects and the status of the incoming Sequence Flow.

- For example, an outgoing assignment may evaluate Process data and then select different sets of outgoing Sequence Flow, based on the results of the evaluation.
    - However, the outgoing assignment should ensure that at least one of the outgoing Sequence Flow will always be chosen.

Complex Gateways – Merging Behaviour

- There are many patterns that can be performed with the Complex Gateway, such as typical Inclusive Gateway behavior, batching of multiple tokens, accepting tokens from some paths but ignoring the tokens from others, etc.

- The Gateway looks the same for each of these patterns, so the modeler should use a Text Annotation to inform the reader of the diagram how it is used.

- The Complex Gateway uses an ***incoming assignment*** when tokens arrive. The condition may refer to Process or Data Object information and the status of the incoming Sequence Flow.

  - If the condition is false, nothing happens other than the token is held there.

  - If the condition is true, then the action could be set to pass the token to the output side of the Gateway, thereby activating the outgoing assignment, or the action could be set to consume the token.

Design a a sample expense reimbursement process. This process provides for reimbursement of expenses incurred by employees for the company. For example buying a technical book, office supplies or software. In a normal day there are several hundreds of instances of this process created. Concentrate on the basic flow of the Process…

After the reception of a meeting remainder, a new account must be created if the employee does not already have one. The report is then reviewed for automatic approval. Amounts under $200 are automatically approved, whereas amounts equal to or over $200 require approval of the supervisor. In case of rejection, the employee must receive a rejection notice by email. The reimbursement goes to the employee's direct deposit bank account. If the request is not completed in 7 days, then the employee must receive an "approval in progress" email If the request is not finished within 30 days, then the process is stopped and the employee

Exercise - Solution
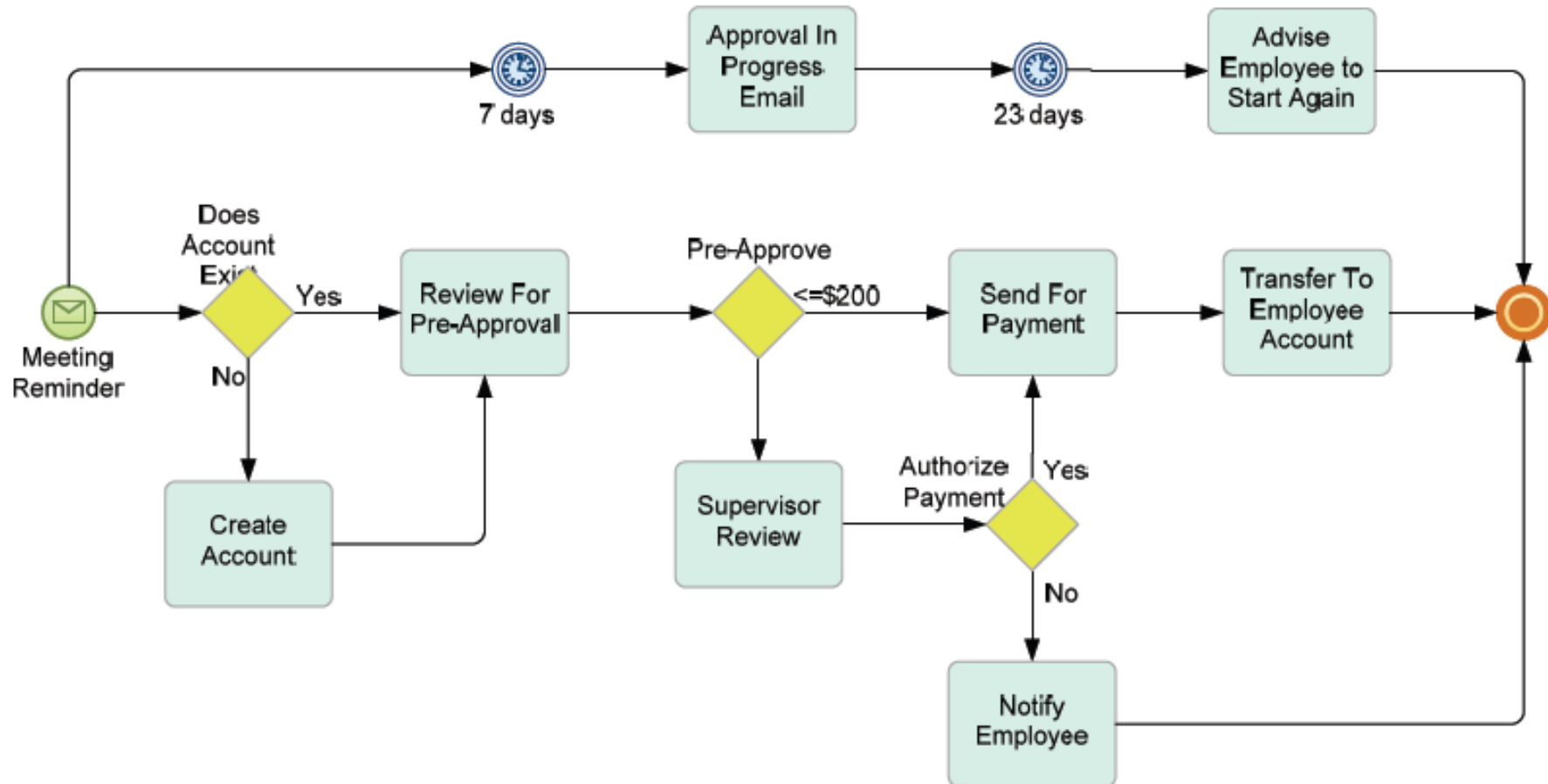
Further Advanced Concepts

- Error Events and Exception Handling
- Cancel Events, Compensation Events and Transactions
- Ad Hoc Processes
- Conversation Diagrams
- Coreography Diagrams

Error Events

Error End Event – Catching

Error Intermediate Event - Throwing

- The Error Intermediate Event is used to handle the occurrence of an *error* that needs the interrupting of an Activity (to which it is attached).

- An *error is generated* by applications or systems involved in the work (which are transparent to the Process) or by End Events.

- The Error End Event is used to ***throw an error***.

- The Error Intermediate Event can only be used when attached to the boundary of an Activity, thus it can only be used to ***catch an error***.

- When an error occurs ***all work will stop for that Process***.

- However, these Events do not interrupt the Activity since they are only operational after an Activity has completed.

Exception Handling

Handle Shipment
Request Shipment → Get Shipment Ack → Get Shipment Report → Error End Event
Shipment Failed

Handle Order
Handle Payment
Handle Shipment
Shipment Failed — Payment Failed
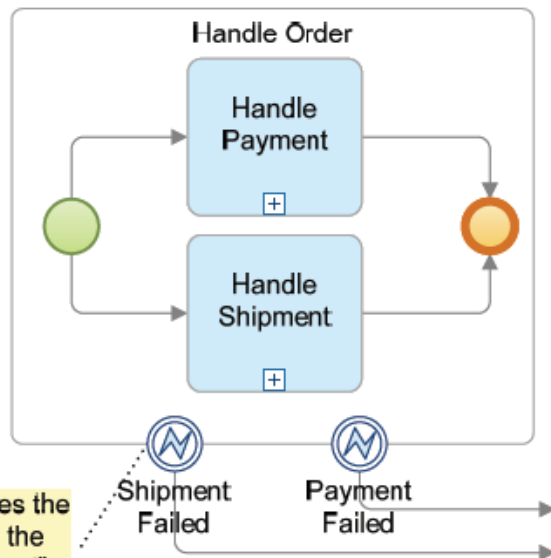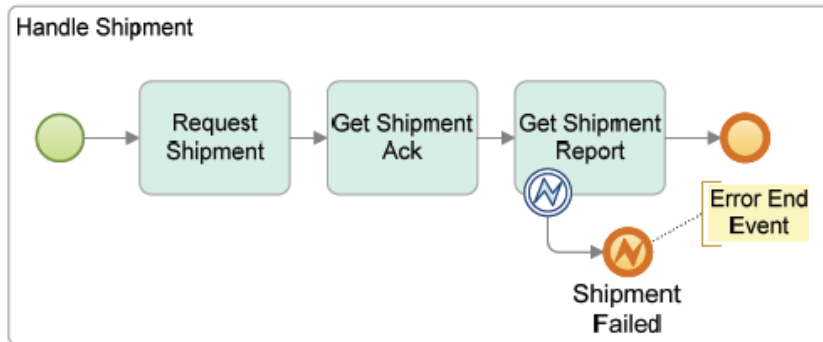
This Event catches the *error* thrown in the "Handle Shipment" Sub-Process

- The error thrown by the Event will be caught by an Intermediate Event at a higher level.

- Errors have a **specific scope of visibility**. <u>An error can only be seen by a parent Process</u>. Other Processes at the same level or within different Pools cannot see the error.

- ***Errors only move upward in the Process hierarchy***.

- If there happens to be more than one Process level higher than the Error End Event, then first level that has a catch Error Intermediate Event attached to its boundary will be interrupted, even if there are higher levels that could possible catch the same error.

113

Exception

The token leaves the previous flow object and arrives at the Activity with the attached Intermediate Event.
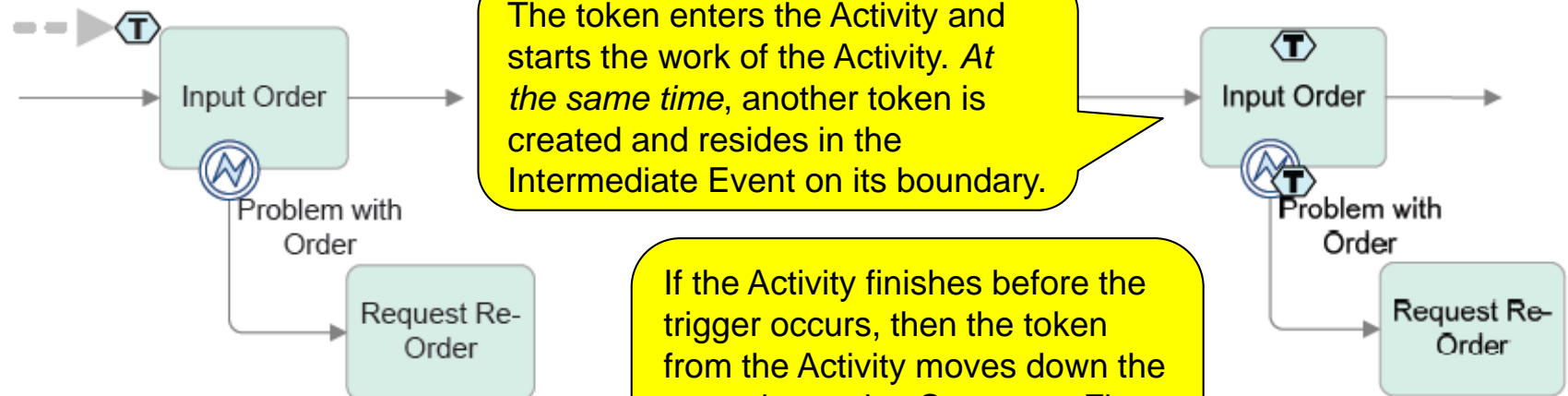
The token enters the Activity and starts the work of the Activity. *At the same time*, another token is created and resides in the Intermediate Event on its boundary.

Input Order

Problem with Order

Request Re-Order

Input Order

Problem with Order

Request Re-Order

If the Activity finishes before the trigger occurs, then the token from the Activity moves down the normal outgoing Sequence Flow of the Activity and the additional token is consumed.

Input Order

Problem with Order

Request Re-Order
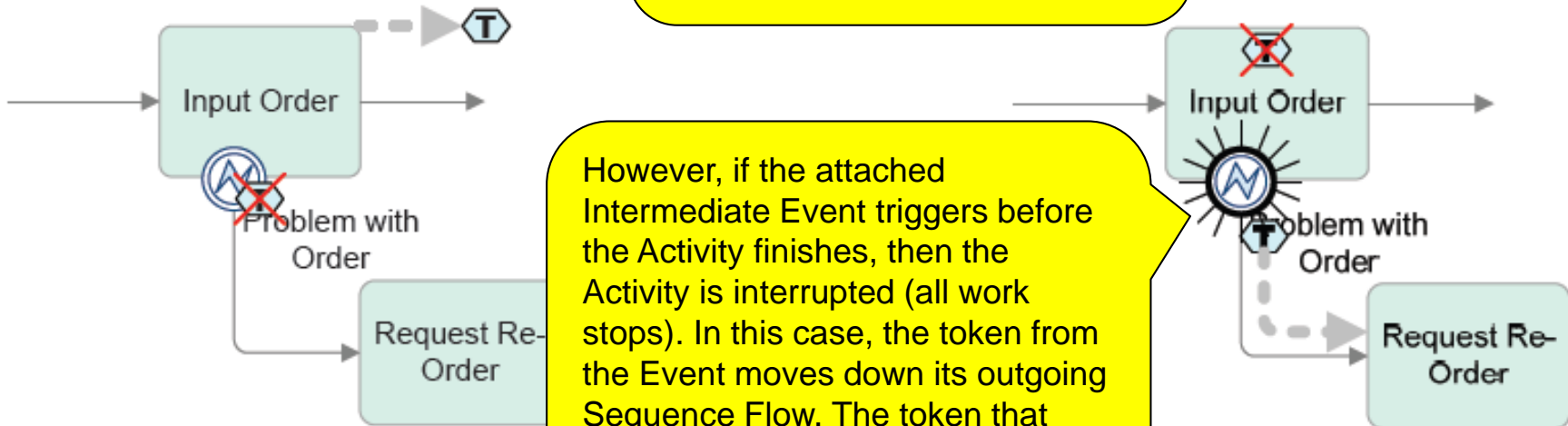
However, if the attached Intermediate Event triggers before the Activity finishes, then the Activity is interrupted (all work stops). In this case, the token from the Event moves down its outgoing Sequence Flow. The token that was on the Activity is consumed.

Input Order

Problem with Order

Request Re-Order

114

Transactions

A Process model (i.e., within one Pool), shows the Activities of the Transaction Sub-Process for just one of the participants.

Bookings

Book Flight

Booking Error

Cancel Flight

Book Hotel

Booking Error

Cancel Hotel

Successful Bookings

Charge Buyer

Transaction

Failed Bookings

Send Unavailability Notice

Handle through Customer Service

Exceptions (Hazards)

A Transaction is a set of activities that **logically belong together**.

- In BPMN, a Transaction is *a formal business relationship and agreement between two or more participants*.

- For a Transaction to succeed, all parties involved have to perform their own Activities and reach an agreement point.

- If <u>any one of them </u>withdraws or fails to complete, then the Transaction cancels and all parties need to undo all the work that has completed.
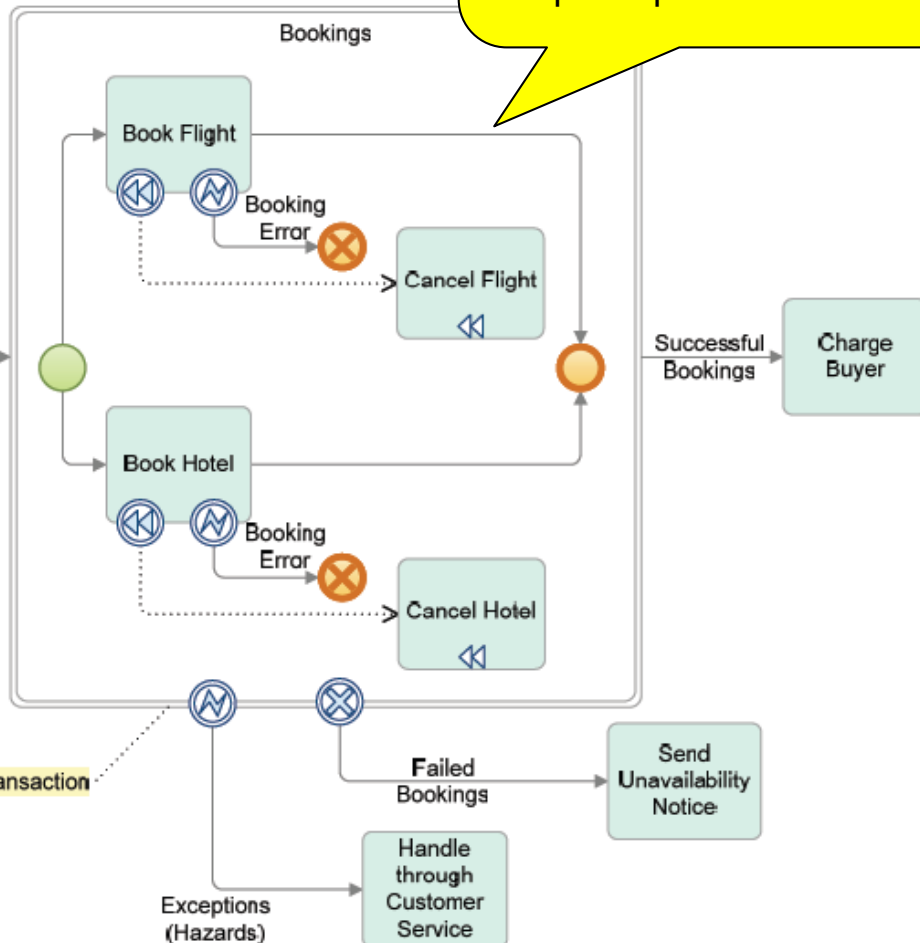
Transactions

- Transaction Sub-Processes have special behaviors.

- *Firstly*, they are associated with a ***Transaction Protocol***. This means that the companies involved in the Transaction must be able to send and receive all the handshaking messages between the participants.
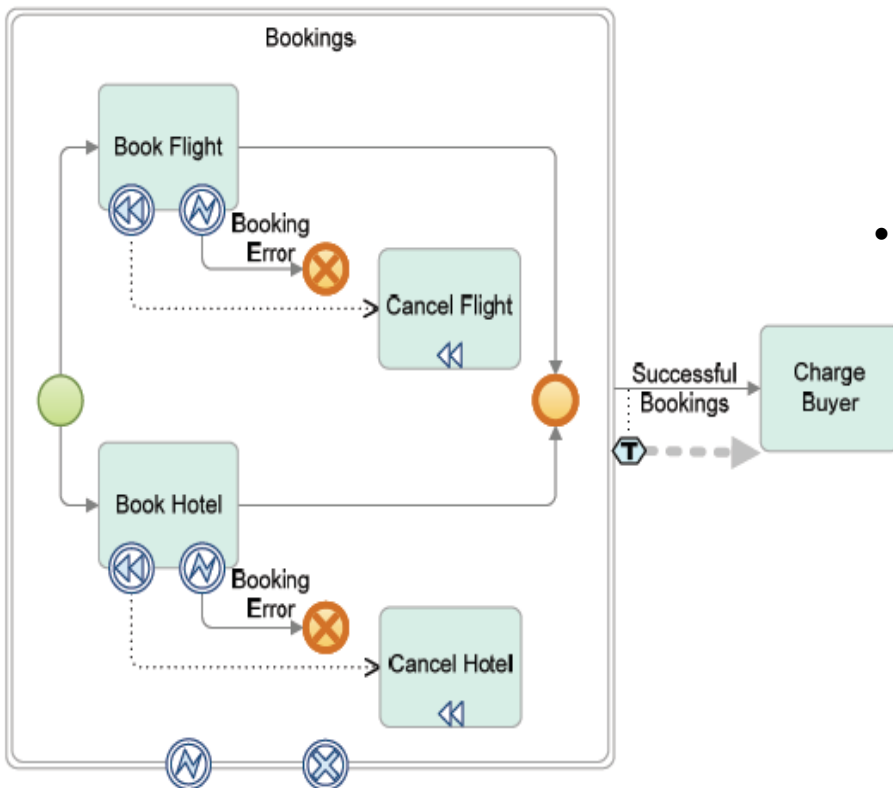
- *Secondly*, if the work of all the Activities in the Transaction Sub-Process complete normally and all the tokens reach an End Event, the Sub-Process is still not complete.

- *Thirdly,* if a processing or technical error occurs for one of the participants of the Transaction, then there are two possibilities for interrupting the Transaction Sub-Process:
  - An attached Error Intermediate Event is triggered (often called a hazard) and the Transaction Sub-Process is interrupted.
  - An attached Cancel Intermediate Event is triggered and the Transaction Sub-Process is cancelled.

Cancel Events

- The Cancel Intermediate Event is designed to handle a situation where a *transaction is canceled.*

- Cancel Intermediate Events can only **catch** a transaction cancellation. The Cancel End Event **throws** the cancellation.

- The Cancel Intermediate Event can only be attached to the boundary of a Transaction Sub-Process. It can be triggered by a Cancel End Event *within* the Sub-Process, or through a cancellation received through the *transaction protocol* assigned to the Transaction Sub-Process.

- When triggered, the Transaction Sub-Process is interrupted (all work stops) and the Sub-Process is *rolled-back,* which may result in the compensation of some of the Activities within the Sub-Process.

- To cancel the Transaction Sub-Process, the Cancel End Event must be contained within the Sub-Process or within a lower level child Sub-Process.

117

Compensation Events
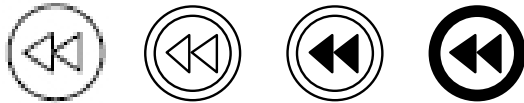
- There are two types of Compensation Intermediate Events: **throwing** and **catching** - i.e. sending and receiving.

- The *catch Compensation Intermediate Event* can only be used by attaching them to the boundary of an Activity. However, the *throw Compensation Intermediate Event* is used in normal flow.

- The Compensation End Event indicates that the ending of a Process path results in the triggering of a compensation.

- In the definition of the Compensation End Event the name of an Activity can be identified as the Activity that should be compensated. The Activity must be within the Process, either at the top-level Process or within a Sub-Process.

- If the named Activity was completed and it has an attached Compensation Intermediate Event, then that Activity will be compensated.

Hazard in a Transaction Sub-Process
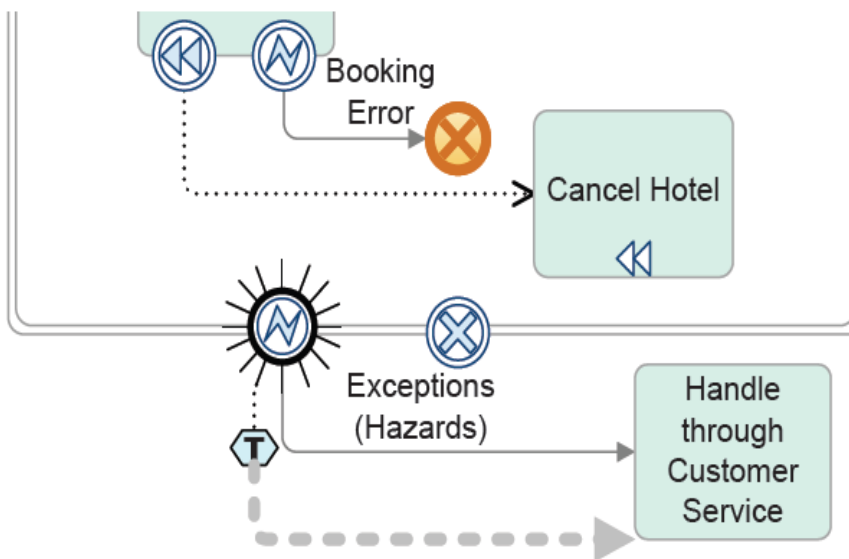
- When there is a hazard, a normal cancellation and compensation are not sufficient to fix the situation.

- The transaction is then interrupted.

- The error can happen within the Transaction Sub-Process or within a Process (unseen) of one the other participants in the transaction.

- The error from one of the other participants will be sent through the transaction protocol.

- When Error Intermediate Event triggers, all work within the Sub-Process is terminated immediately - there is no compensation.

- The token then is sent down the outgoing Sequence Flow of the Error Event to reach Activities that will deal with the situation.

Cancellation in a Transaction Sub-Process
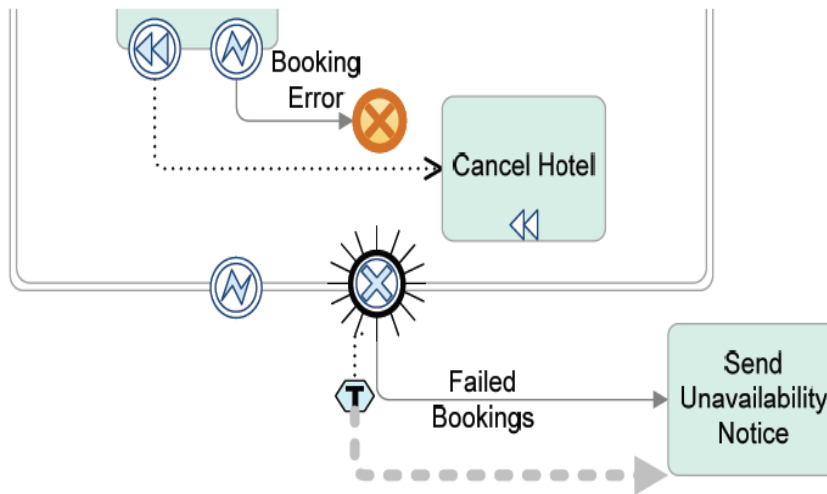
- a Transaction Sub-Process can be **cancelled** through an Event internal to the Sub-Process or through a cancellation sent through a transaction protocol.

- When a Transaction Sub-Process is cancelled, the Cancel Intermediate Event attached to its boundary is triggered.

- The token will eventually continue down the Cancel Intermediate Event's outgoing Sequence Flow, but the behavior of the Transaction Sub-Process involves more than just interrupting the work in the Sub-Process.

- Indeed, all ongoing work within the transaction *is cancelled.*

Compensation in a Transaction Sub-Process
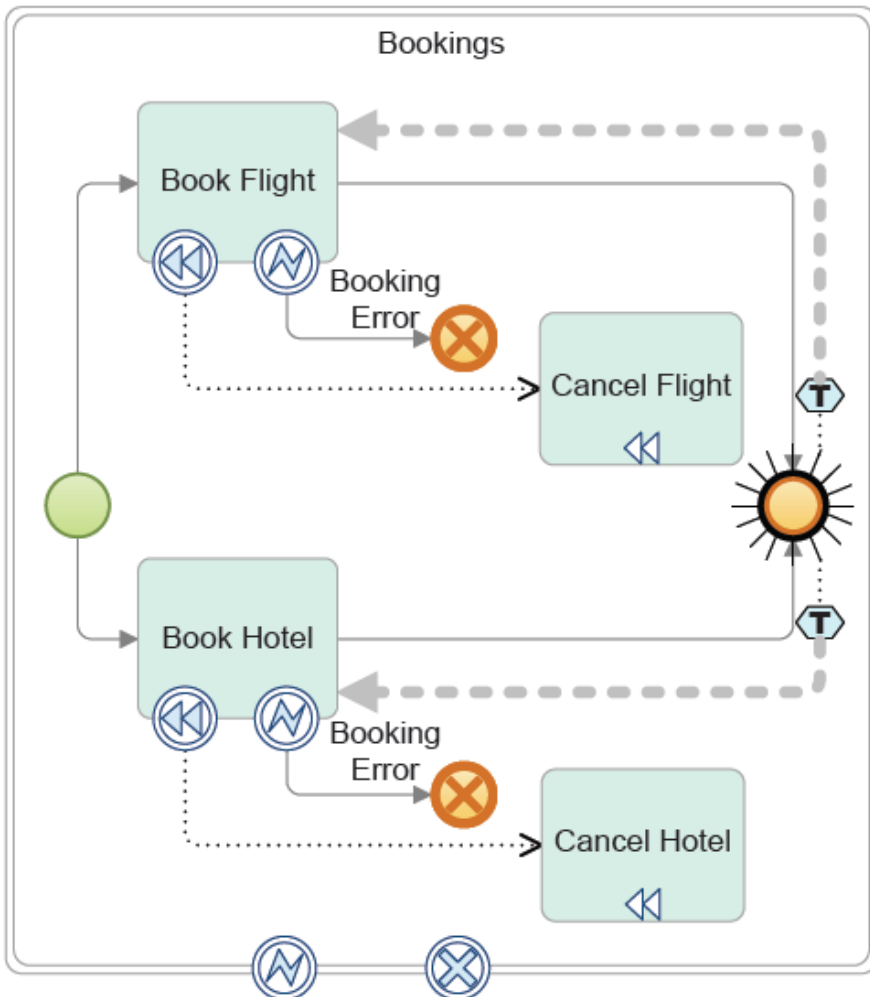
- However, completed work (in the Transaction Sub-Process) may need to be undone, which requires a "rolling back" before the parent Process can continue.

- This means that each Activity in turn, in reverse order, is checked to see whether or not it requires compensation. Compensation is the undoing of work that has been completed.

- A token can be used to trace this rolling back as it travels backward through the Process after a Transaction Sub-Process has been cancelled.

121

Compensation in a Transaction Sub-Process

- Compensation does not just happen automatically. Another Activity is required to undo the work of the original Activity.

- The Compensation Activity links to each Activity via the Compensation Intermediate Event attached to its boundary.

- The link between the normal Activity and the Compensation Activity is done through an **Association** rather than a Sequence Flow.

- The Compensation Intermediate Event is **never triggered during the normal flow of the Process**. It only can be triggered during the roll-back of the Transaction Sub-Process. Only one Compensation Activity can be associated with the Compensation Intermediate Event.

- When the reversal of the token reaches an Activity that has an attached Compensation Intermediate Event, that Compensation Event fires and the token is then sent to the associated Compensation Activity.

Compensation in a Transaction Sub-Process
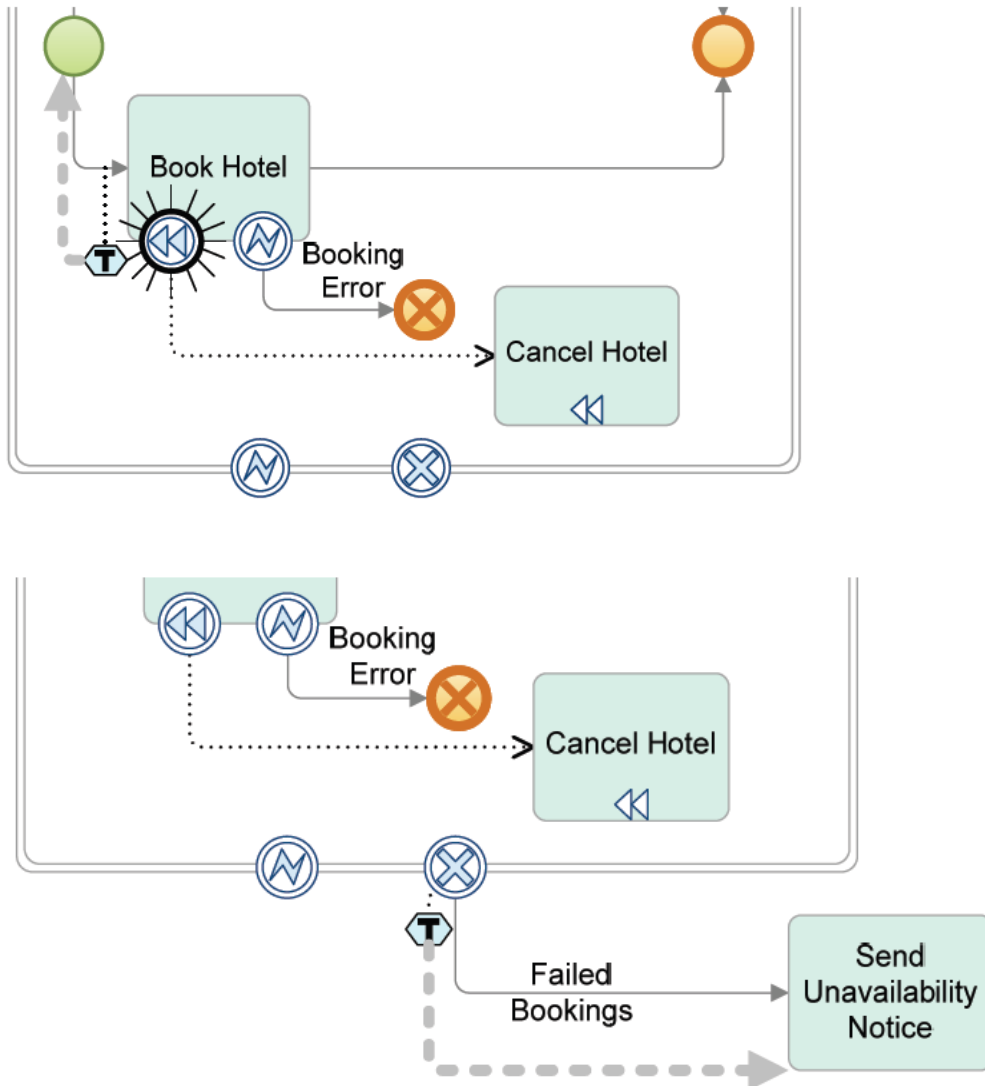
- When the Compensation Activity has completed, the token continues its backward journey through the Transaction by leaving the Activity whose work was just undone.

- When all the Activities of the Transaction Sub-Process have been checked and, if necessary, compensated, then the cancellation of the Transaction is completed.

- This allows the token in the parent Process to travel down the outgoing Sequence Flow of the attached Compensation Intermediate Event.

Ad Hoc Processes

- The Ad Hoc Process represents Processes where the Activities might occur in any order, and in any frequency- there is no specific ordering or obvious decisions.

Write a Book Chapter
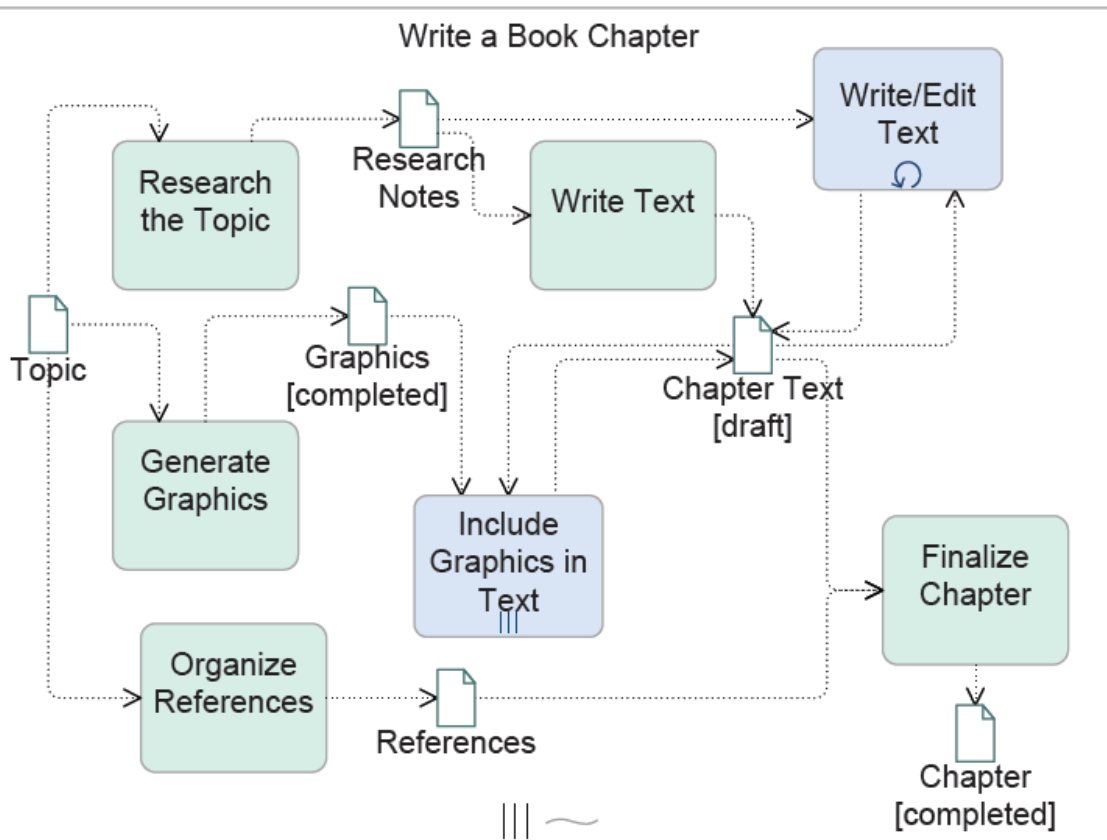
- Typically, the Activities in an Ad Hoc Process involve human performers who make the decisions as to what Activities to perform, when to perform them, and how many times.

- The Ad Hoc Process has a non-graphical completion condition attribute that is used to determine if the work of the Process is complete.

Conversation Diagrams

- Conversation diagram models conversations between participants (pools)
- Conversation allows a modeler to group collaboration interactions between two or more participants, which together achieve a common goal, e.g. "negotiate delivery" → abstraction of common interactions.
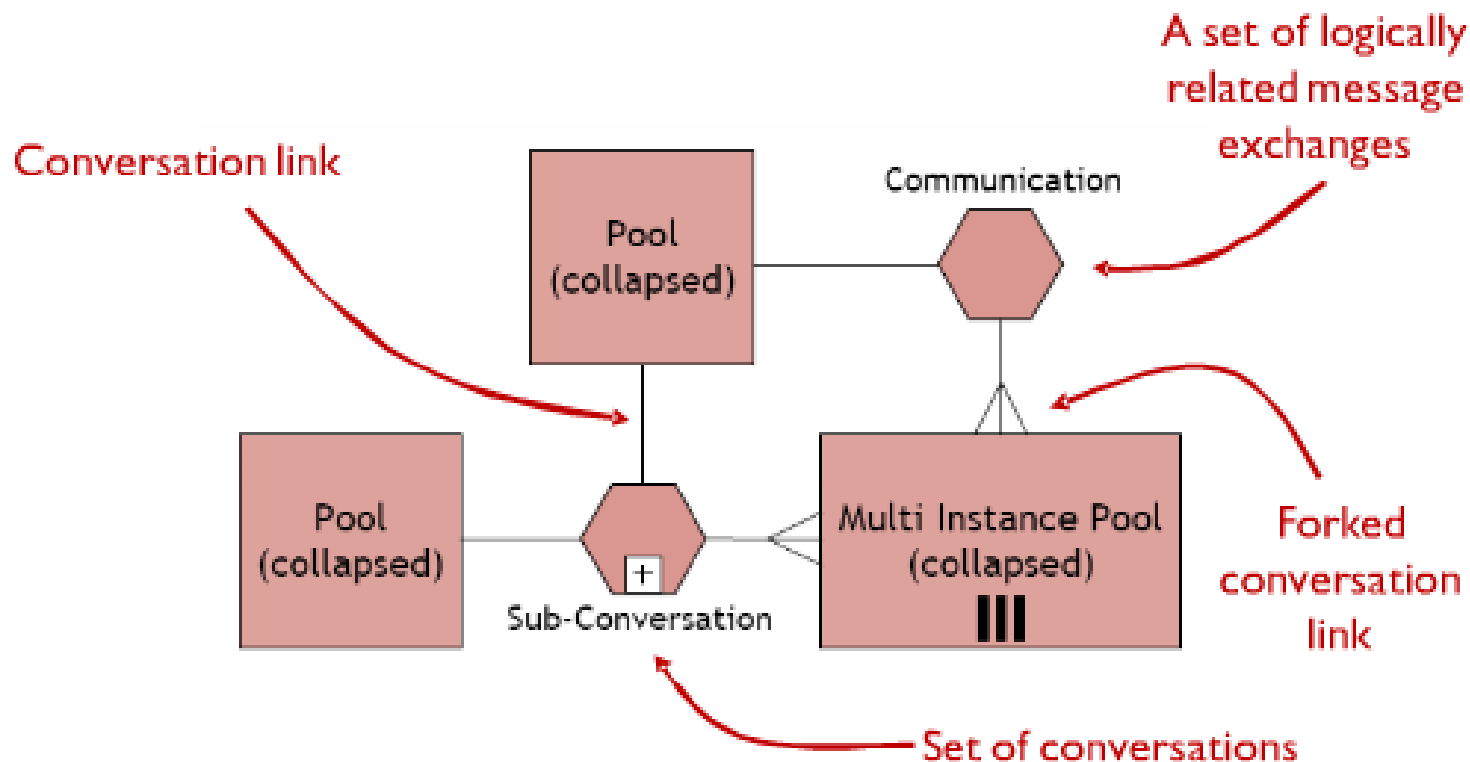
Conversation Elements

- Communication or conversation defines a set of logically related message exchanges.
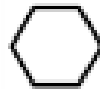
- When marked with a "+" it indicates a Sub-Conversation, a compound conversation element.

Is equal to:

Participant A — Conversation — Participant B
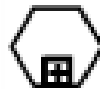
Is equal to:
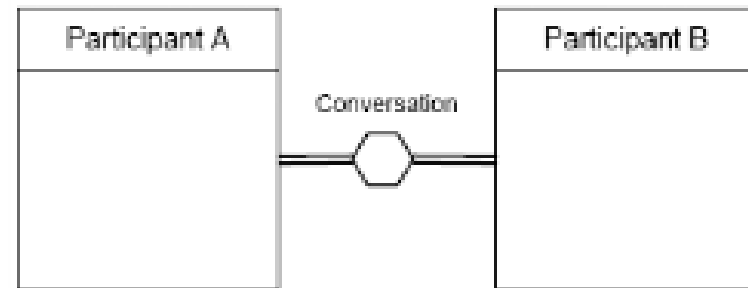
Participant A — — — — Participant B
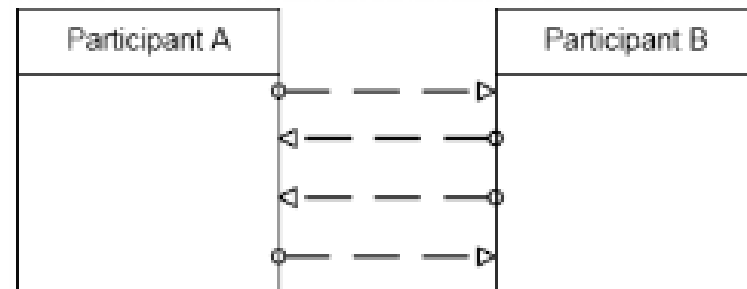
Coreography Diagram

- **Choreography** provides a flowchart view to sequence the interactions between participants.
- A choreography defines the sequence of interactions between participants.
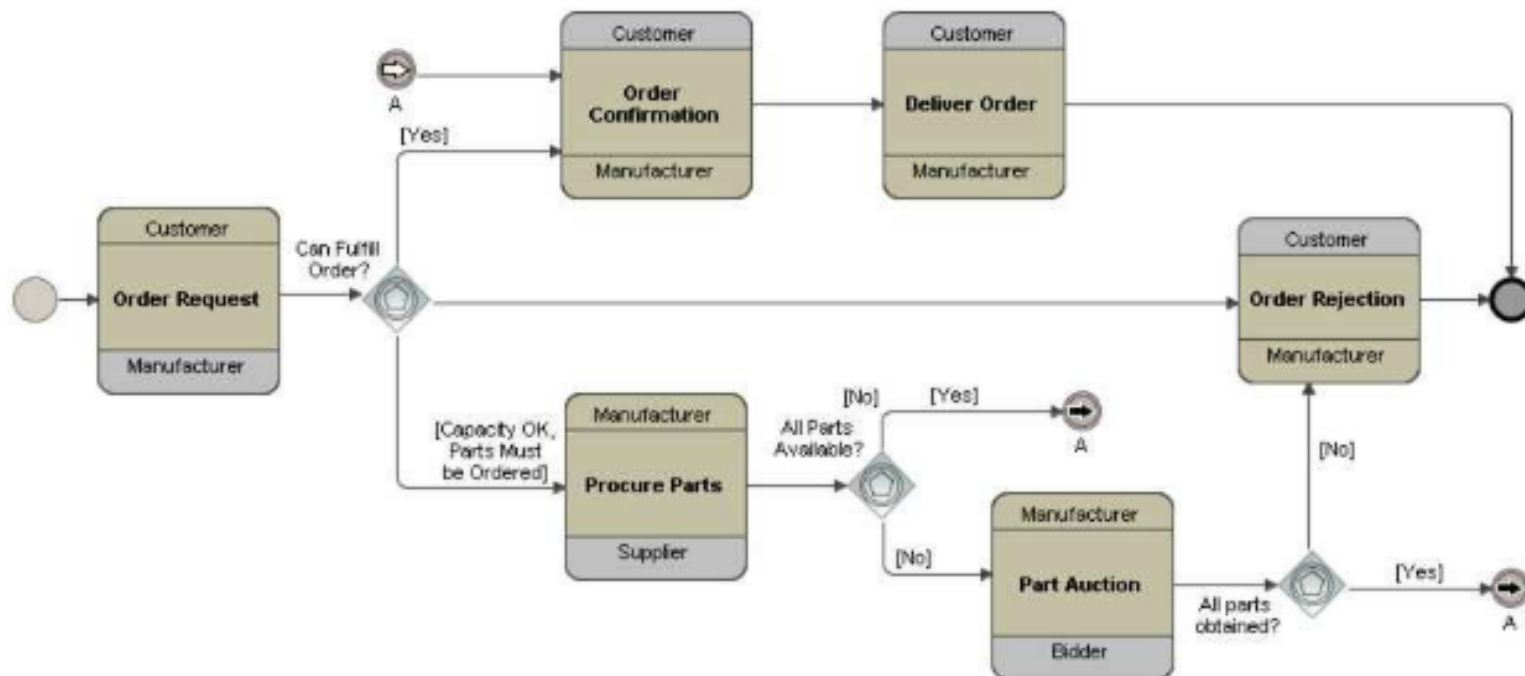
Coreography Elements

Choreography Task – defines a set of participants that exchange messages to complete a task.

The unshaded participant is the initiator of the activity

A Choreography Sub-Process – contains a refined choreography with several Interactions

The shaded participant is NOT the initiator
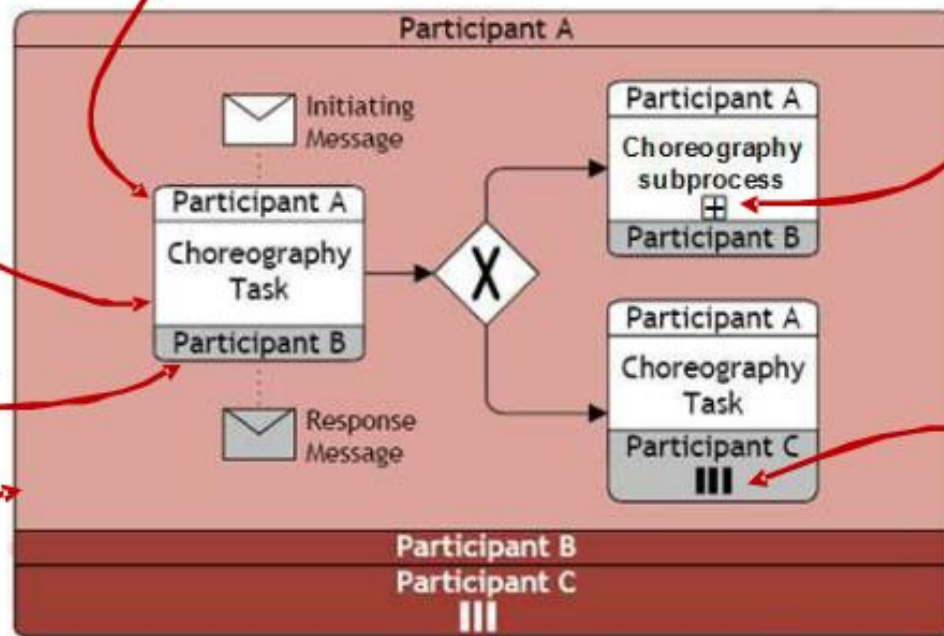
Expanded Choreography Sub Process – contains a detail of interactions (Message Exchange) between participants.

Multiple Participants Marker – Applied to one or both of the Choreography Task participant identifiers to show there's a set of participants of the same kind

Participant A

Initiating Message

Participant A
Choreography Task
Participant B

Response Message

Participant A
Choreography subprocess

Participant B

Participant A
Choreography Task
Participant C

Participant B
Participant C

128

Topics

- Process Modeling
- BPMN Background
- Basic Concepts
- Advanced Concepts
- **Conclusions**

BPMN 2.0 - Business Process Model and Notation

Activities

Task — A Task is a unit of work, the job to be performed. When marked with a ⊞ symbol it indicates a Sub-Process, an activity that can be refined.

Transaction — A Transaction is a set of activities that logically belong together; it might follow a specified transaction protocol.

Event Sub-Process — An Event Sub-Process is placed into a Process or Sub-Process. It is activated when its start event gets triggered and can interrupt the higher level process context or run in parallel (non-interrupting) depending on the start event.

Call Activity — A Call Activity is a wrapper for a globally defined Task or Process reused in the current Process. A call to a Process is marked with a ⊞ symbol.

Activity Markers
Markers indicate execution behavior of activities:

- ⊞ Sub-Process Marker
- ↻ Loop Marker
- ‖‖‖ Parallel MI Marker
- ☰ Sequential MI Marker
- ∼ Ad Hoc Marker
- ◁ Compensation Marker

Task Types
Types specify the nature of the action to be performed:

- ✉ Send Task
- ✉ Receive Task
- 👤 User Task
- 🖐 Manual Task
- ▦ Business Rule Task
- ⚙ Service Task
- 📜 Script Task

Sequence Flow — defines the execution order of activities.

Default Flow — is the default branch to be chosen if all other conditions evaluate to false.

Conditional Flow — has a condition assigned that defines whether or not the flow is used.

Gateways

Exclusive Gateway — When splitting, it routes the sequence flow to exactly one of the outgoing branches. When merging, it awaits one incoming branch to complete before triggering the outgoing flow.

Event-based Gateway — Is always followed by catching events or receive tasks. Sequence flow is routed to the subsequent event/task which happens first.

Parallel Gateway — When used to split the sequence flow, all outgoing branches are activated simultaneously. When merging parallel branches it waits for all incoming branches to complete before triggering the outgoing flow.

Inclusive Gateway — When splitting, one or more branches are activated. All active incoming branches must complete before merging.

Complex Gateway — Complex merging and branching behavior that is not captured by other gateways.

Exclusive Event-based Gateway (instantiate) — Each occurrence of a subsequent event starts a new process instance.

Parallel Event-based Gateway (instantiate) — The occurrence of all subsequent events starts a new process instance.

## Conversations

A **Conversation** defines a set of logically related message exchanges. When marked with a ⊞ symbol it indicates a Sub-Conversation, a compound conversation element.
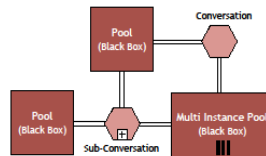
A **Call Conversation** is a wrapper for a globally defined Conversation or Sub-Conversation. A call to a Sub-conversation is marked with a ⊞ symbol.

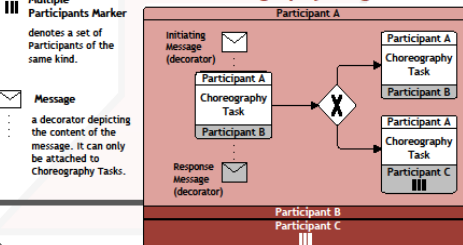A **Conversation Link** connects Conversations and Participants.
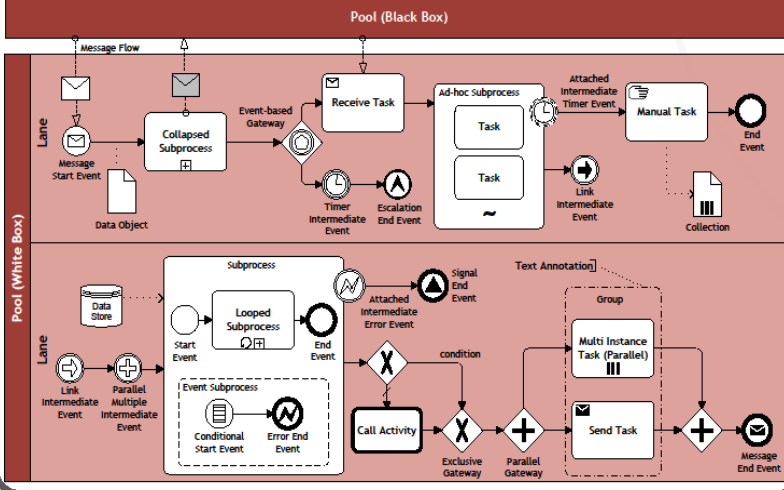
Conversation Diagram

Pool (Black Box) — Conversation — Pool (Black Box) — Sub-Conversation — Multi Instance Pool (Black Box)

Choreographies

Participant A — Choreography Task — Participant B

Participant A — Sub-Choreography ⊞ — Participant B — Participant C

Participant A — Call Choreography — Participant B

A **Choreography Task** represents an Interaction (Message Exchange) between two Participants.

A **Sub-Choreography** contains a refined choreography with several Interactions.

A **Call Choreography** is a wrapper for a globally defined Choreography Task or Sub-Choreography. A call to a Sub-Choreography is marked with a ⊞ symbol.

‖‖‖ Multiple Participants Marker denotes a set of Participants of the same kind.

✉ Message a decorator depicting the content of the message. It can only be attached to Choreography Tasks.

Choreography Diagram

Participant A — Initiating Message (decorator) — Response Message (decorator) — Participant A / Choreography Task / Participant B — Participant A / Choreography Task / Participant C — Participant A / Choreography Task / Participant B — Participant B — Participant C

Collaboration Diagram

Pool (Black Box)

Pool (White Box)

Lane — Message Flow — Message Start Event — Data Object — Collapsed Subprocess ⊞ — Event-based Gateway — Receive Task — Ad-hoc Subprocess (Task / Task ∼) — Attached Intermediate Timer Event — Manual Task — End Event — Timer Intermediate Event — Escalation End Event — Link Intermediate Event — Collection

Lane — Data Store — Start Event — Looped Subprocess — End Event — Attached Intermediate Error Event — Signal End Event — condition — Group — Multi Instance Task (Parallel) ‖‖‖ — Text Annotation — Link Intermediate Event — Parallel Multiple Intermediate Event — Event Subprocess — Conditional Start Event — Error End Event — Call Activity — Exclusive Gateway — Parallel Gateway — Send Task — Message End Event

Swimlanes

Pool — Lane — Task — Lane — Task

Pools (Participants) and Lanes represent responsibilities for activities in a process. A pool or a lane can be an organization, a role, or a system. Lanes subdivide pools or other lanes hierarchically.

Message Flow symbolizes information flow across organizational boundaries. Message flow can be attached to pools, activities, or message events. The Message Flow can be decorated with an envelope depicting the content of the message.

The order of message exchanges can be specified by combining message flow and sequence flow.

Events

	Start	Start Event Sub-Process Interrupting	Start Event Sub-Process Non-Interrupting	Intermediate Catching	Intermediate Boundary Interrupting	Intermediate Boundary Non-Interrupting	Intermediate Throwing	End
	Standard							Standard
None: Untyped events, indicate start point, state changes or final states.	○						○	○
Message: Receiving and sending messages.	✉	✉	✉	✉	✉	✉	✉	✉
Timer: Cyclic timer events, points in time, time spans or timeouts.	🕐	🕐	🕐	🕐	🕐	🕐		
Escalation: Escalating to an higher level of responsibility.		⌃	⌃		⌃	⌃	⌃	⌃
Conditional: Reacting to changed business conditions or integrating business rules.	▤	▤	▤	▤	▤	▤		
Link: Off-page connectors. Two corresponding link events equal a sequence flow.				→			→	
Error: Catching or throwing named errors.		⚡		⚡	⚡			⚡
Cancel: Reacting to cancelled transactions or triggering cancellation.					⊗			⊗
Compensation: Handling or triggering compensation.		◁◁		◁◁	◁◁		◀◀	◀◀
Signal: Signalling across different processes. A signal thrown can be caught multiple times.	△	△	△	△	△	△	▲	▲
Multiple: Catching one out of a set of events. Throwing all events defined.	⬡	⬡	⬡	⬡	⬡	⬡	⬟	⬟
Parallel Multiple: Catching all out of a set of parallel events.	✚	✚	✚	✚	✚	✚		
Terminate: Triggering the immediate termination of a process.								●

Data

A Data Object represents information flowing through the process, such as business documents, e-mails, or letters.

A Collection Data Object represents a collection of information, e.g., a list of order items.

A Data Input is an external input for the entire process. A kind of input parameter.

A Data Output is data result of the entire process. A kind of output parameter.

A Data Association is used to associate data elements to Activities, Processes and Global Tasks.

A Data Store is a place where the process can read or write data, e.g., a database or a filing cabinet. It persists beyond the lifetime of the process instance.

References

[BPMN 2.0 spec] OMG. *BPMN 2.0 specification.* (January 2011)

[1] M. Weske. *Business Process Management: Concepts, Languages, Architectures.* Springer-Verlag (2007).

[2] Workflow Management Coalition. *XPDL 2.1 Complete Specification* (2008).

[3] OASIS. *Web Services Business Process Execution Language Version 2.0.* http://docs.oasis-open.org/wsbpel/2.0/Primer/wsbpel-v2.0-Primer.pdf (2007).

[4] R. Hull. *Artifact-Centric Business Process Models: Brief Survey of Research Results and Challenges.* In Proceedings of the OTM 2008 Confederated International Conferences, CoopIS (2008).

[5] W.M.P .van der Aalst, C. Stahl. *Modeling Business Processes A Petri Net Oriented Approach.* The Mit Press (2011).

[6] M. Pesic. *DECLARE: Full Support for Loosely-Structured Processes.* In Proceedings at the 11th IEEE International Enterprise Distributed Object Computing Conference (EDOC) (2007) .

References

[7] A.H.M. ter Hofstede, W. van der Aalst, M. Adams, N. Russell. *Modern Business Process Automation: YAWL and its Support Environment.* Springer-Verlag (2009).

[8] S. Christensen, N. Damgaard Hansen. *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use.* Jensen (1997).

THANKS FOR THE ATTENTION