

# *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** (***B**usiness **P**rocess **M**odel and **N**otation*) standard.
- This lesson shows how BPMN can support different methodologies as well as different modeling goals (e.g., orchestration and choreography), using actual business processes as examples.
- Visit <http://www.bpmn.org/> for downloading the complete specification of BPMN 2.0 and some interesting examples.

- Reference Books :

*Stephen A. White PhD, Derek Miers*

***BPMN Modeling and Reference Guide***

*Thomas Allweyer*

***BPMN 2.0 - Introduction to the Standard for Business Process Modeling***



# Topics

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



# Business Processes

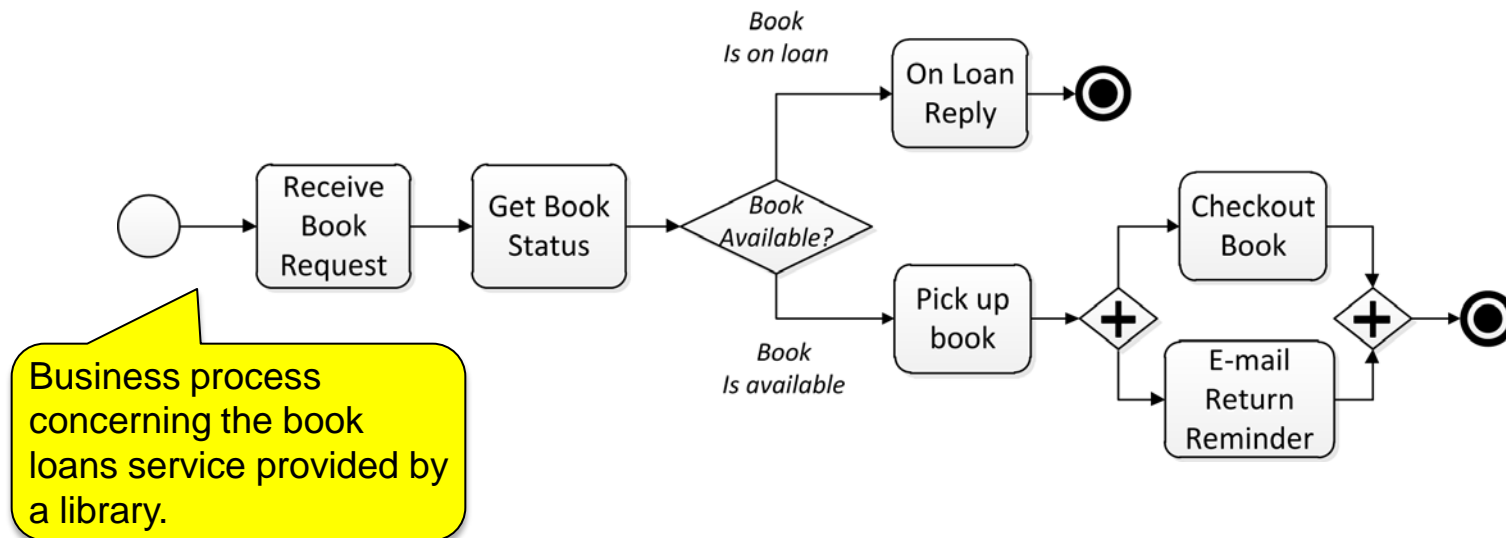
- **Business Process [1]** : *“A business process consists of a set of activities that are performed in coordination in an organizational and technical environment. These activities jointly realize a business goal ”*
  - A **business goal** is the target that an organization aims to achieve by performing 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** (their business processes) for the **orchestration** of participants, information and technology for the realization of products and services.

# The Importance of Process Modeling



- This 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.

# A first BPMN Example

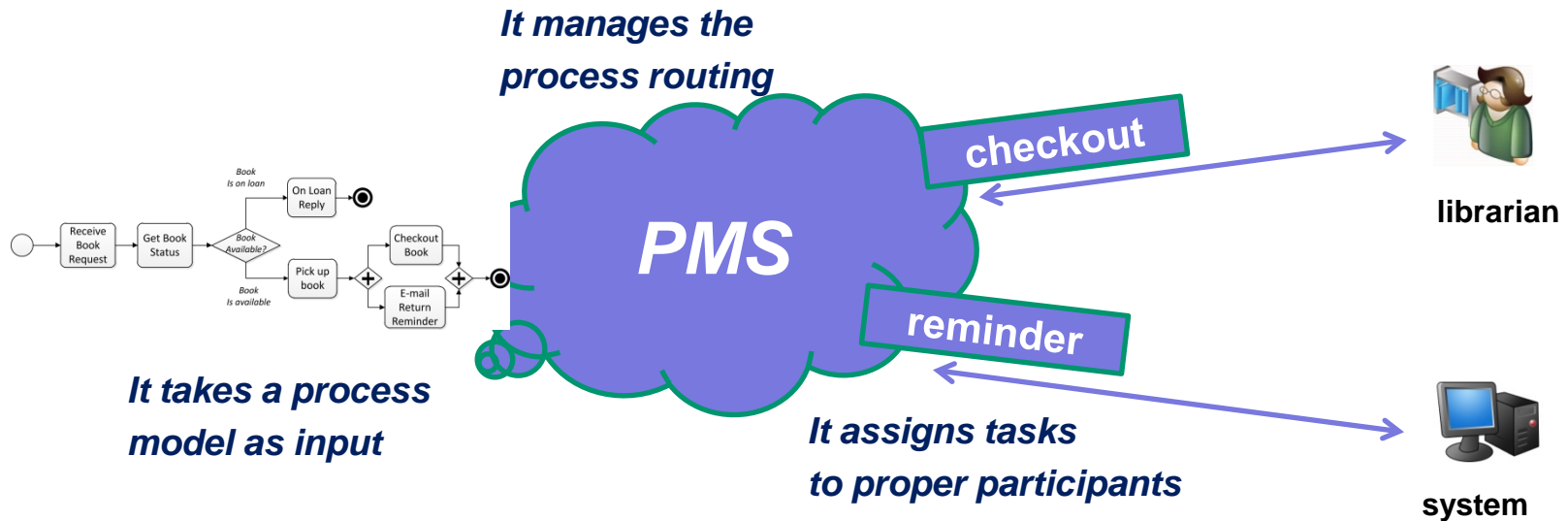


- A notation for graphic business process modeling defines the **symbols** for the various process elements, their correct **meaning** as well as their possible **combinations**.
- Thus, a notation is a standardized language for the description of business processes.

# 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 ”.



- A PMS is driven by a specific business process model.

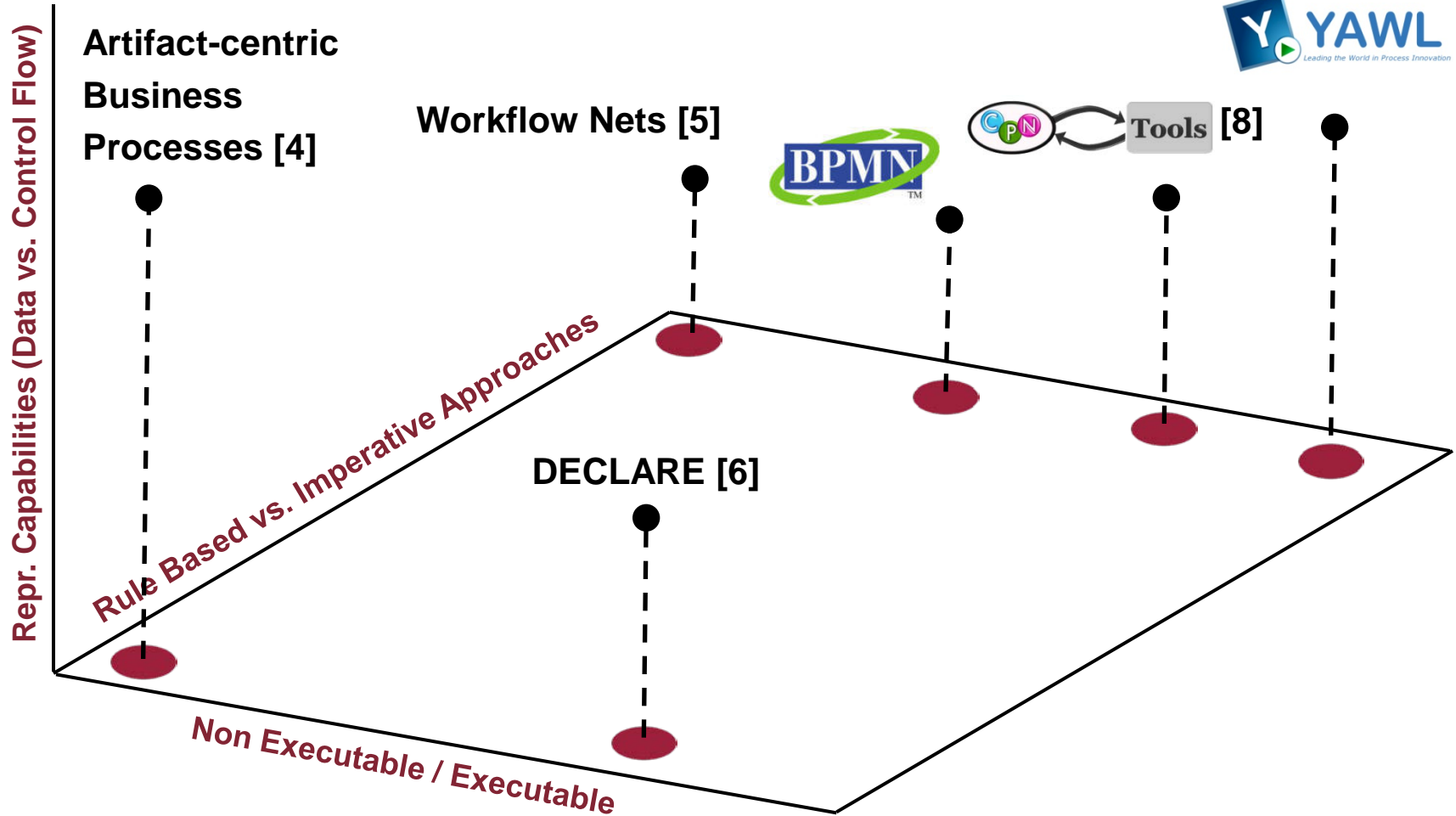
# Executable Process Models



- 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.**
  - By now, standards for executable process descriptions have been established, as for example :
    - **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.



# Modeling Languages for Business Processes





# Topics

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

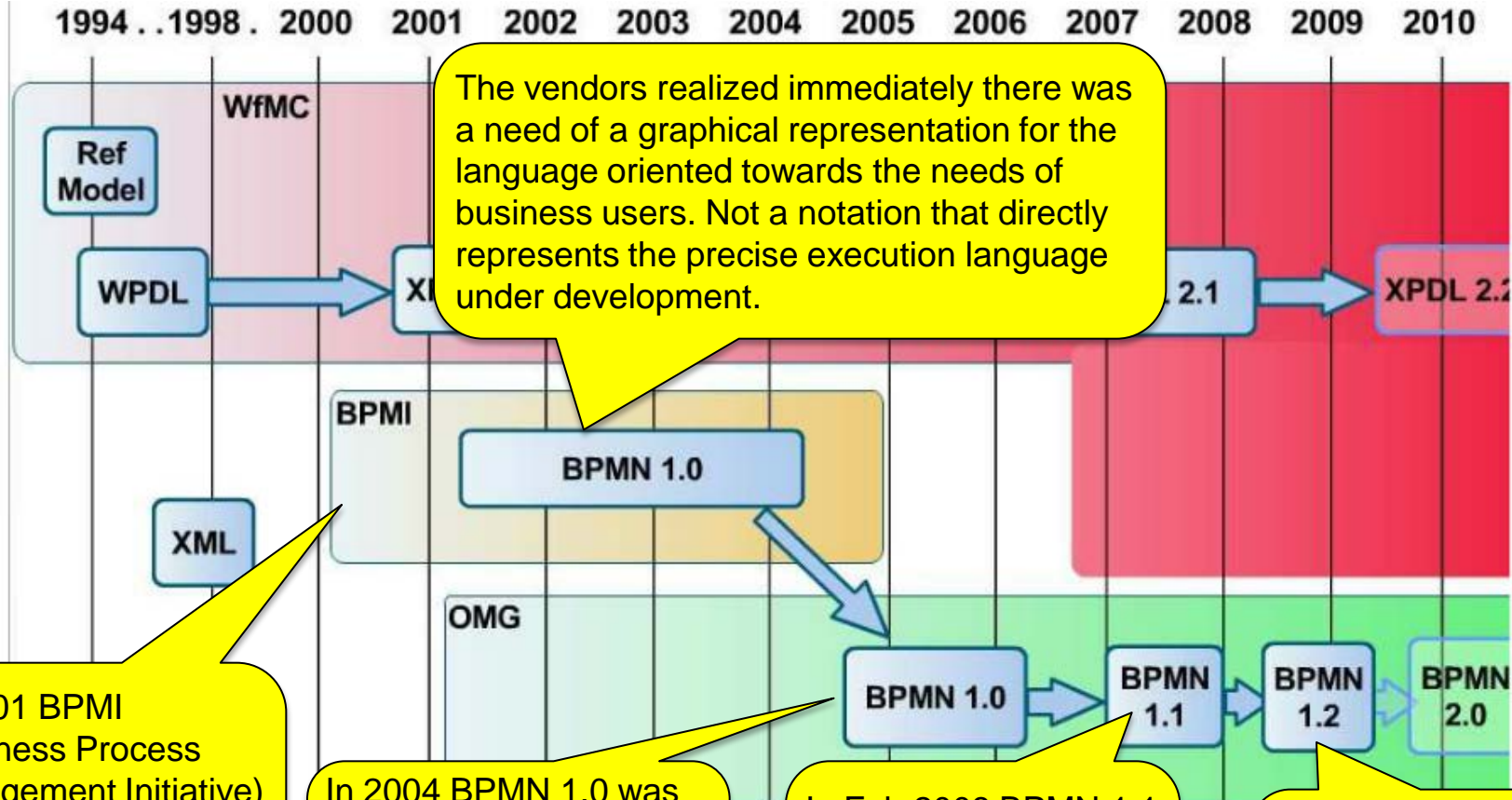


# Why BPMN?

- *“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.

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.

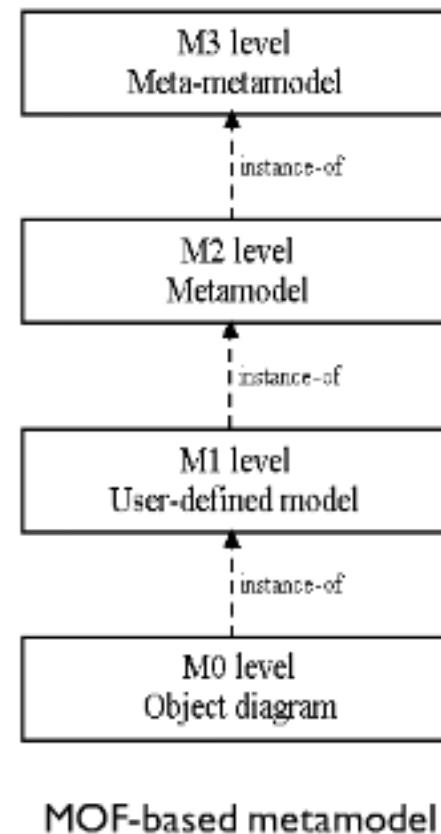


# BPMN 1.2 vs BPMN 2.0

- BPMN 1.2 provided a mapping from a “valid” BPMN diagram to BPEL, such that an engine can execute the process.
  - This 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.

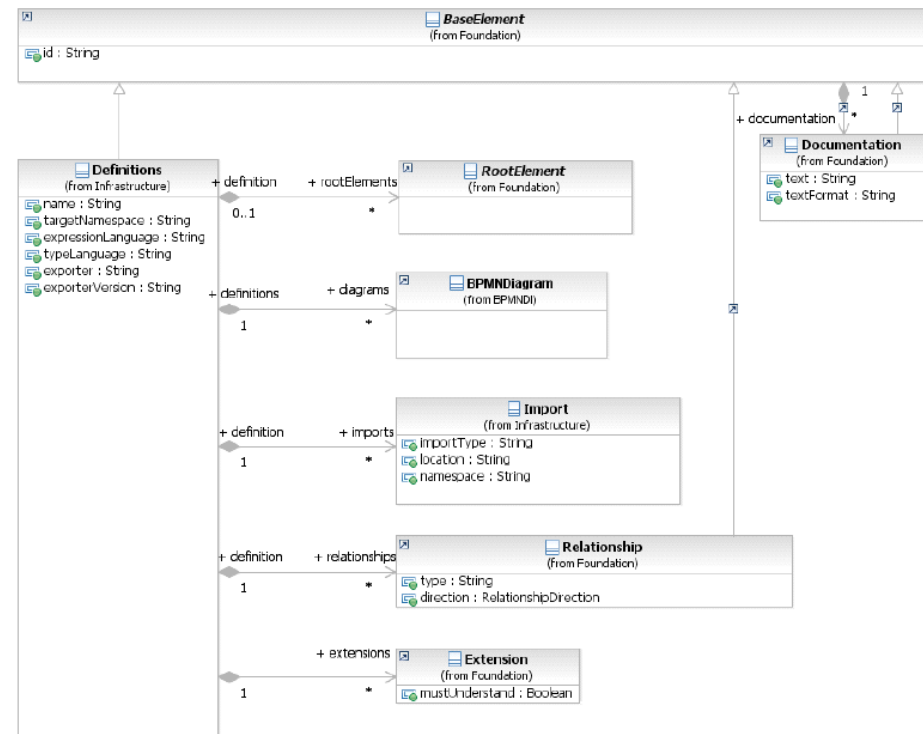
# What is a Metamodel?

- 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
- A valid metamodel is an **ontology**.



# 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.



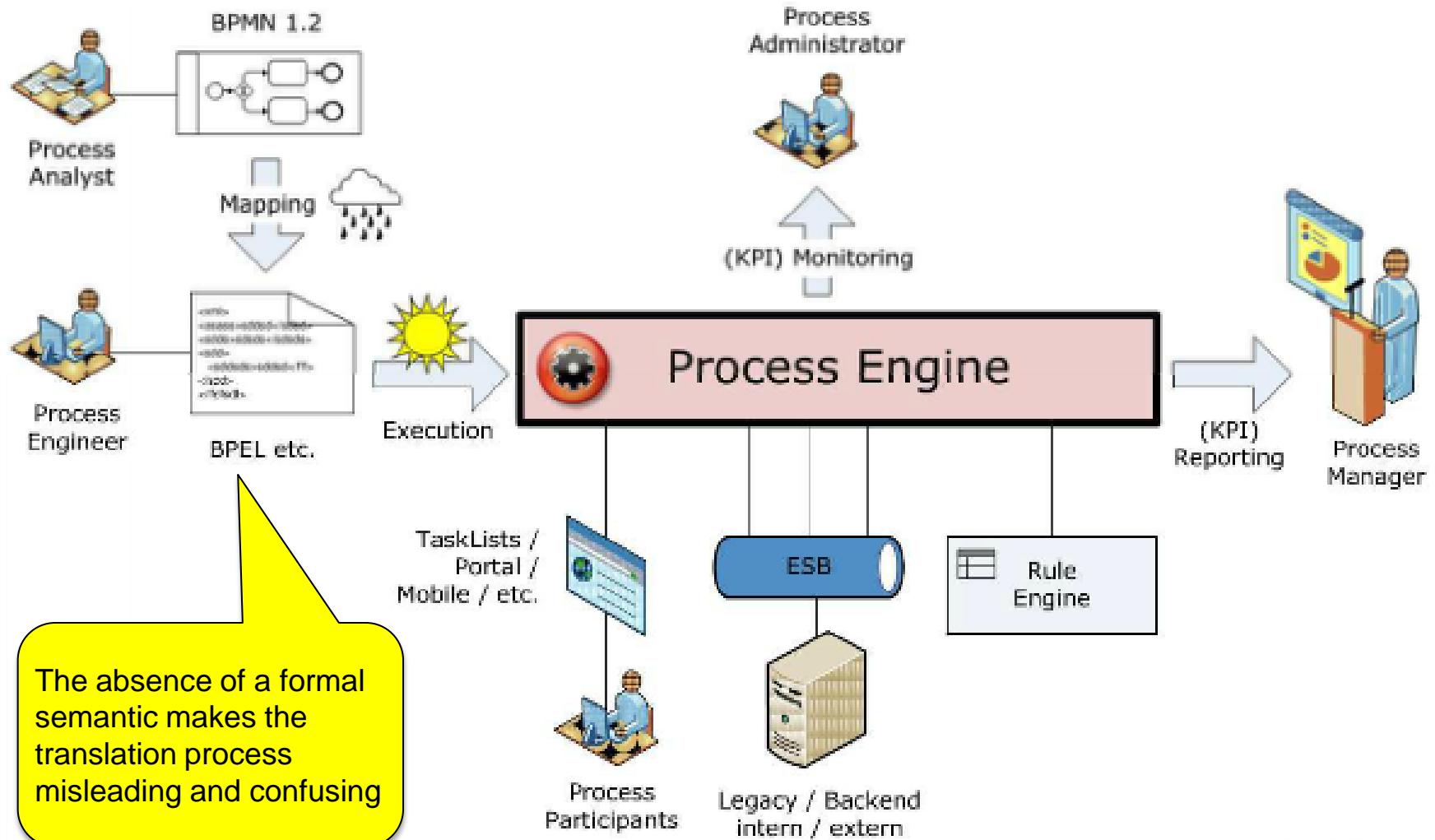


# BPMN 1.2 vs BPMN 2.0

- 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 XPD L format, but the use of XPD L for this purpose is not widely accepted yet.
  - Moreover, XPD L has not been implemented uniformly by all vendors, so that in practice there are quite often problems with model exchange.

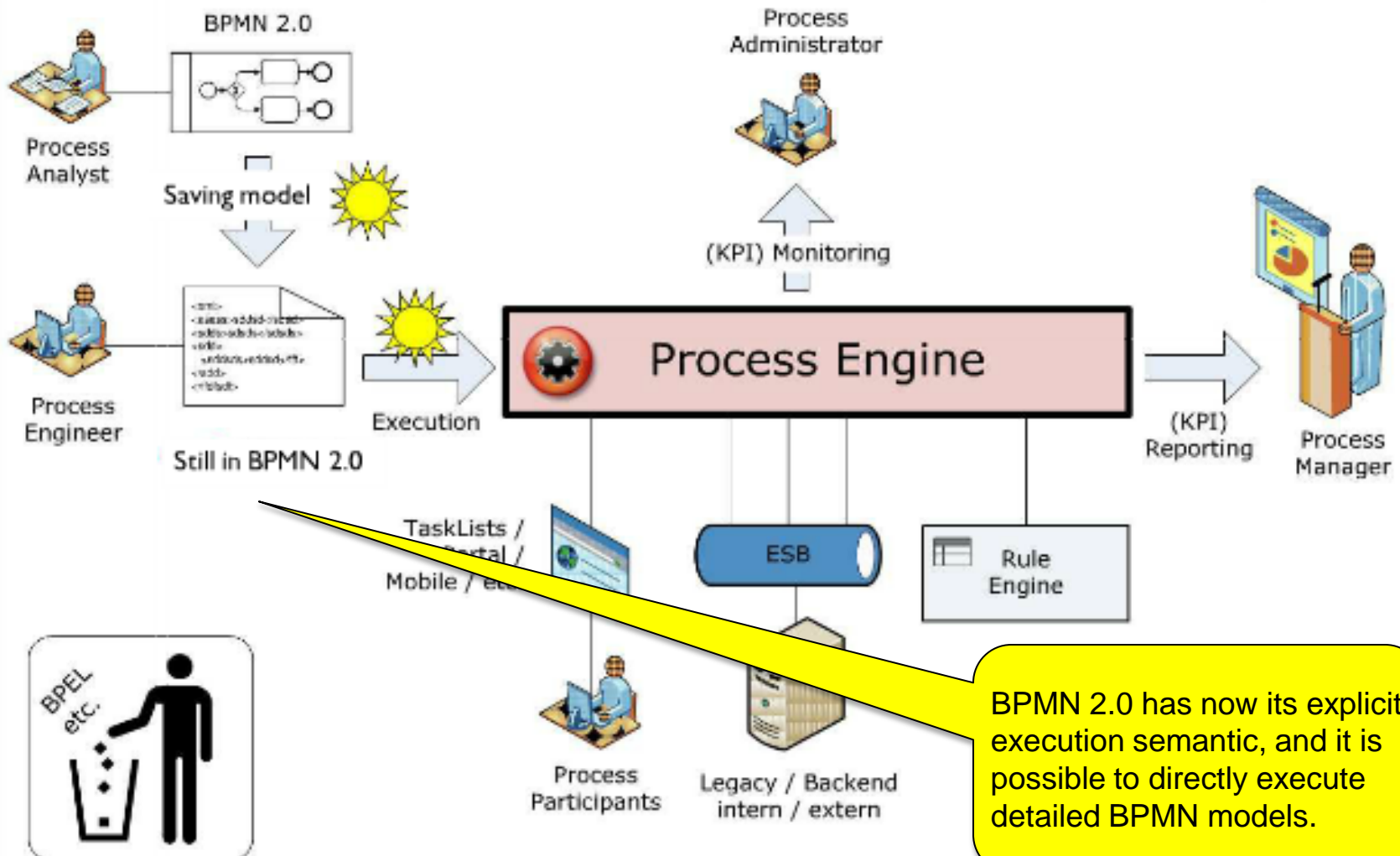


# Current BPMN 1.x problems



The absence of a formal semantic makes the translation process misleading and confusing

# Solving BPMN 1.x problems with 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.

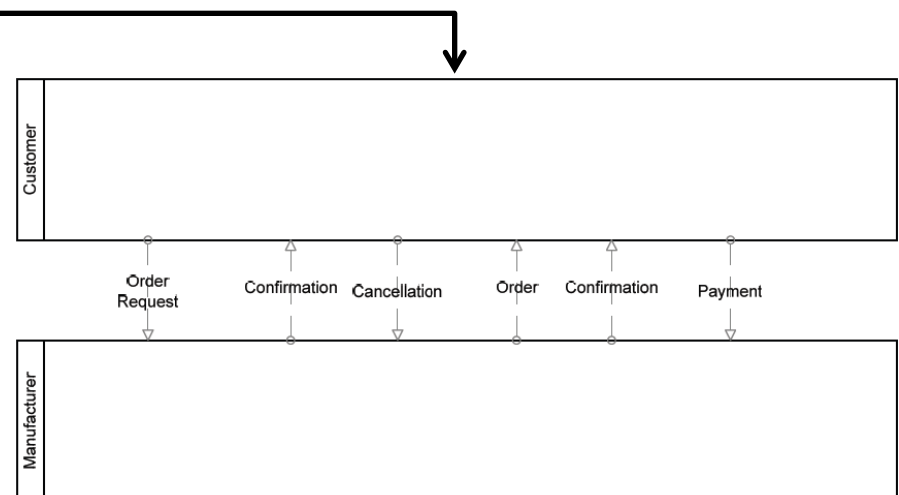


# Topics

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

# 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 container - called a *Pool*. *Each Pool 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.



# Basic elements in a BPMN process



## Flow Objects

### Events



### Activities



### Gateways



## Connectors

### Sequence Flow



### Message Flow



### Association



## Artifacts

### Data Object



Name  
[State]

### Text Annotation



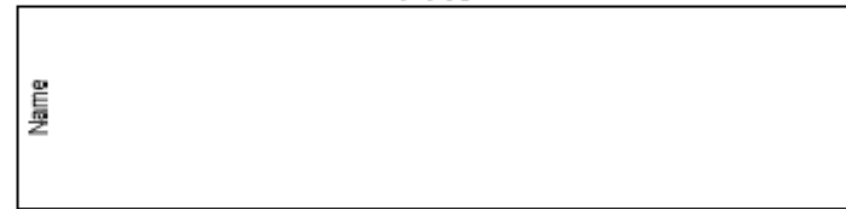
Add Text Here

### Group

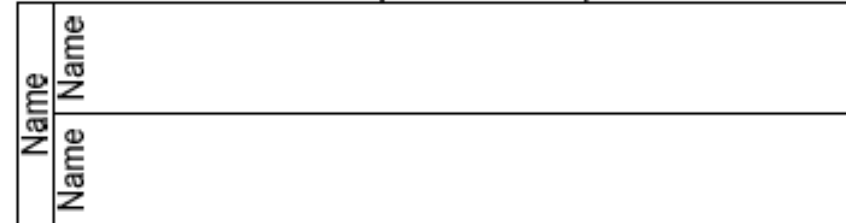


## Swimlanes

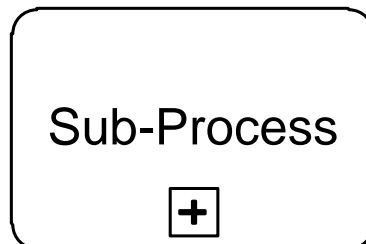
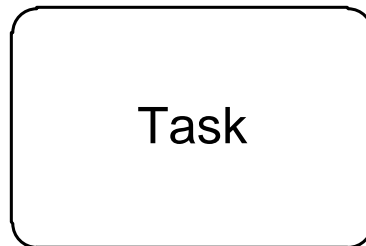
### Pool



### Lanes (within a Pool)

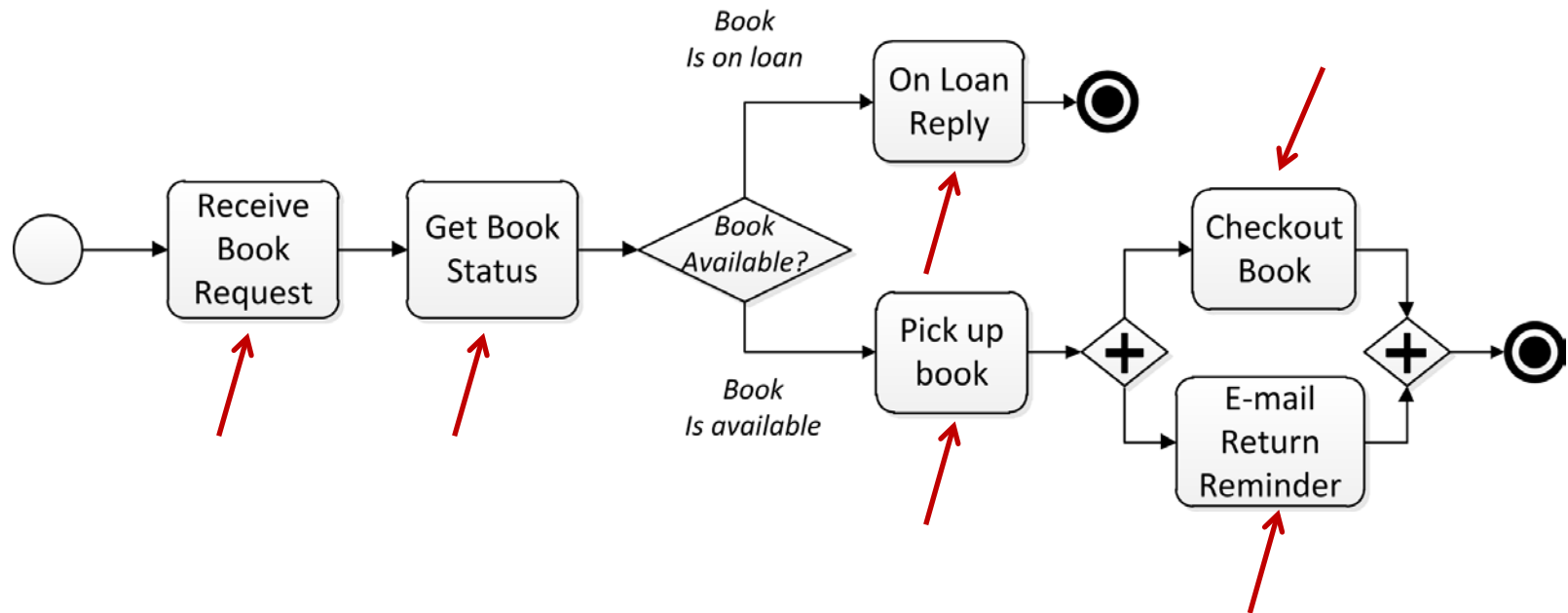


# BPMN Activities



- 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).
- The compound type of an Activity is called a **Sub-Process**.

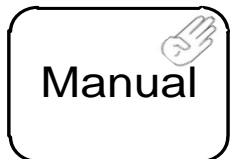
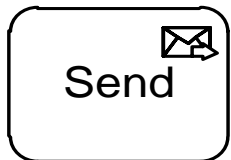
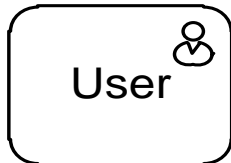
# Tasks



- A **Task** is an atomic activity that is included within a Process. A Task is used when the work in the Process is not broken down to a finer level of detail.



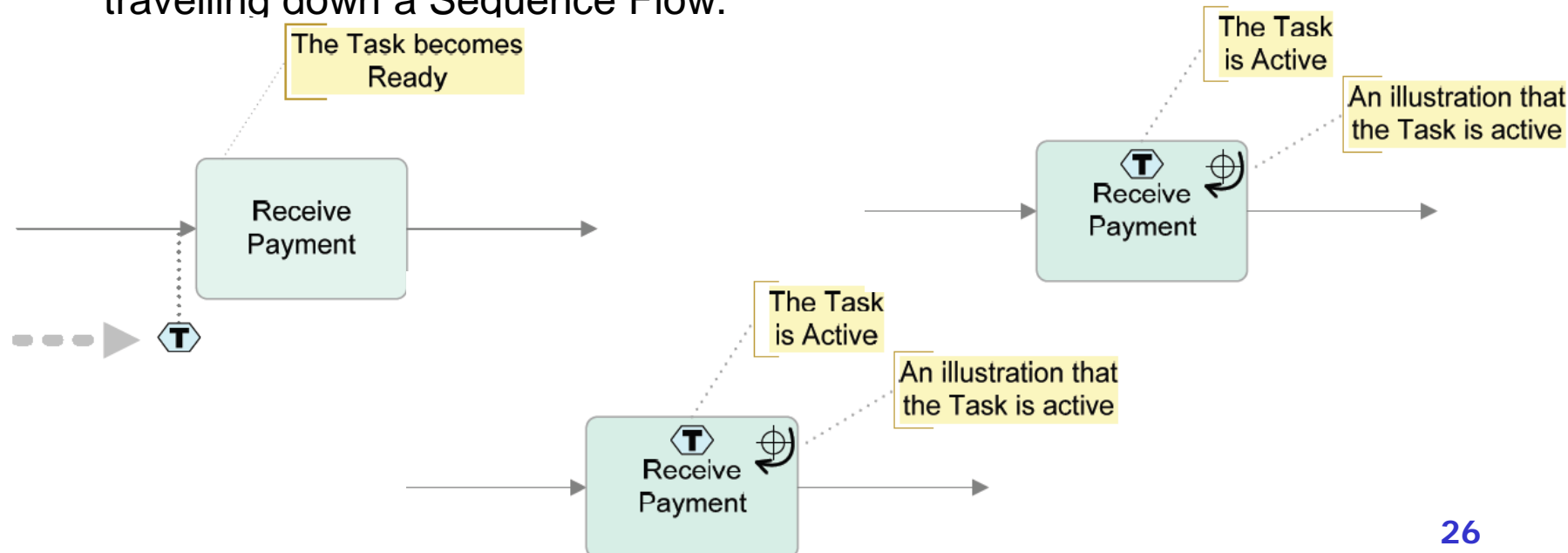
# Specialized types of tasks



- There are 7 specialized types of tasks :
- **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.

# Activity Behaviour

- In order to explain some of the underlying behavior of a BPMN model, we use the concept of **token**.
- 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 traverses the sequence flow, from the start to the end instantaneously. That is, there is no time associated with the token travelling down a Sequence Flow.



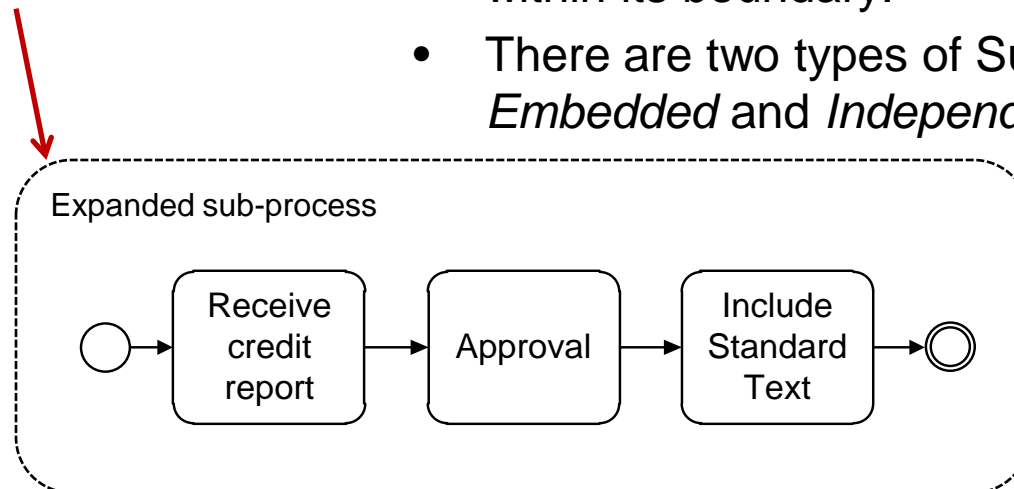
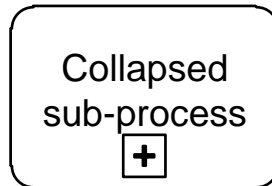
# Connecting Activities

- The *Sequence Flow* defines 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 will tend to have a single *incoming and a single outgoing Sequence Flow*.
- Sequence Flows cannot cross a process boundary (a Pool) or a Sub-Process boundary.

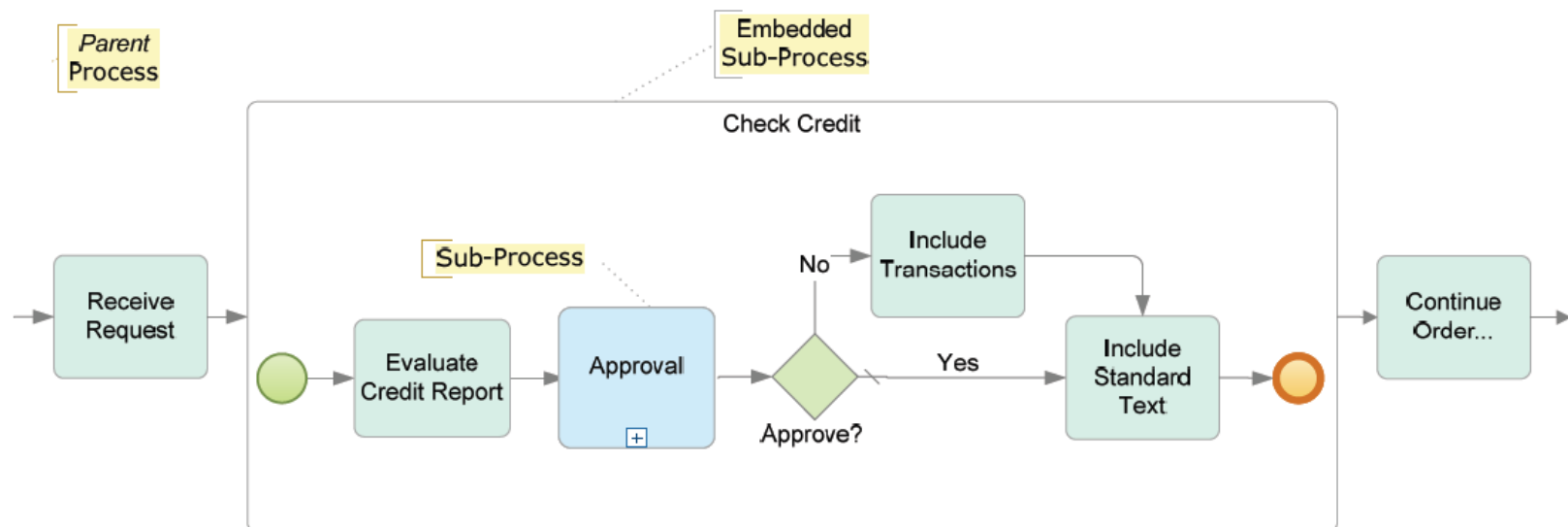
# Types of sub-processes



- Sub-processes enable hierarchical process development.
- The details of the sub-process are not visible in the diagram. A “plus” sign in the lower-center of the shape indicates that the activity is a sub-process and has a lower-level of detail.
- 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*

# 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).

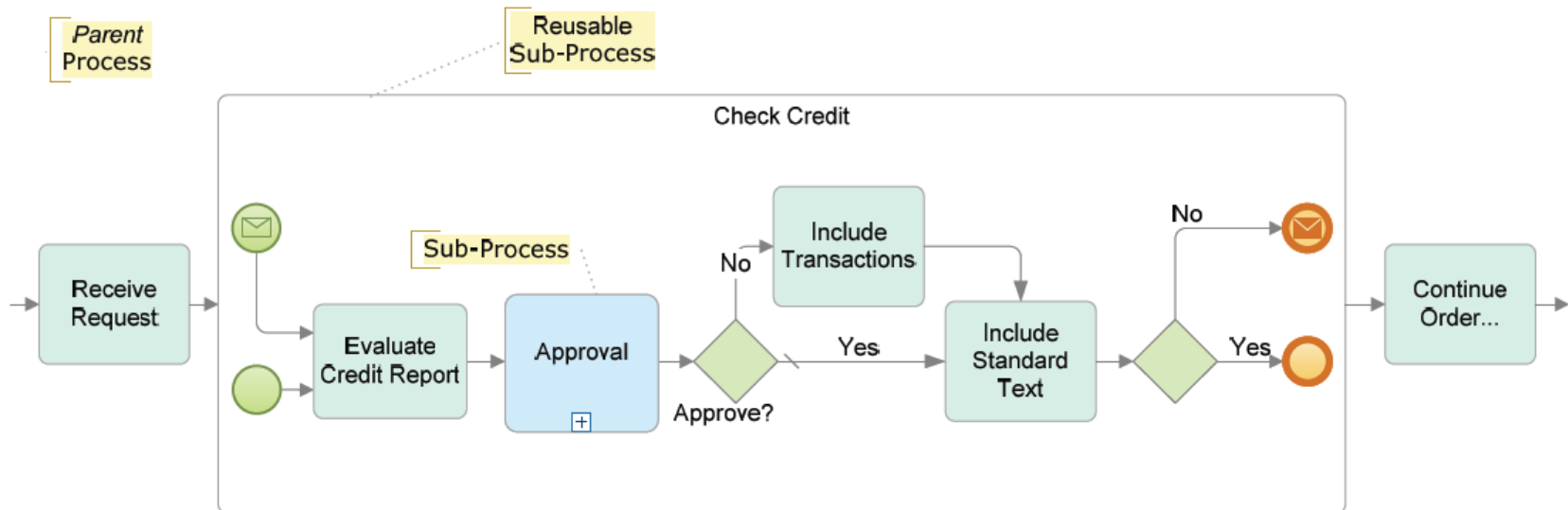


- An important characteristic of an Embedded Sub-Process is that it can only begin with a None Start Event—i.e., without an explicit *trigger* such as a message.

# Independent sub-processes

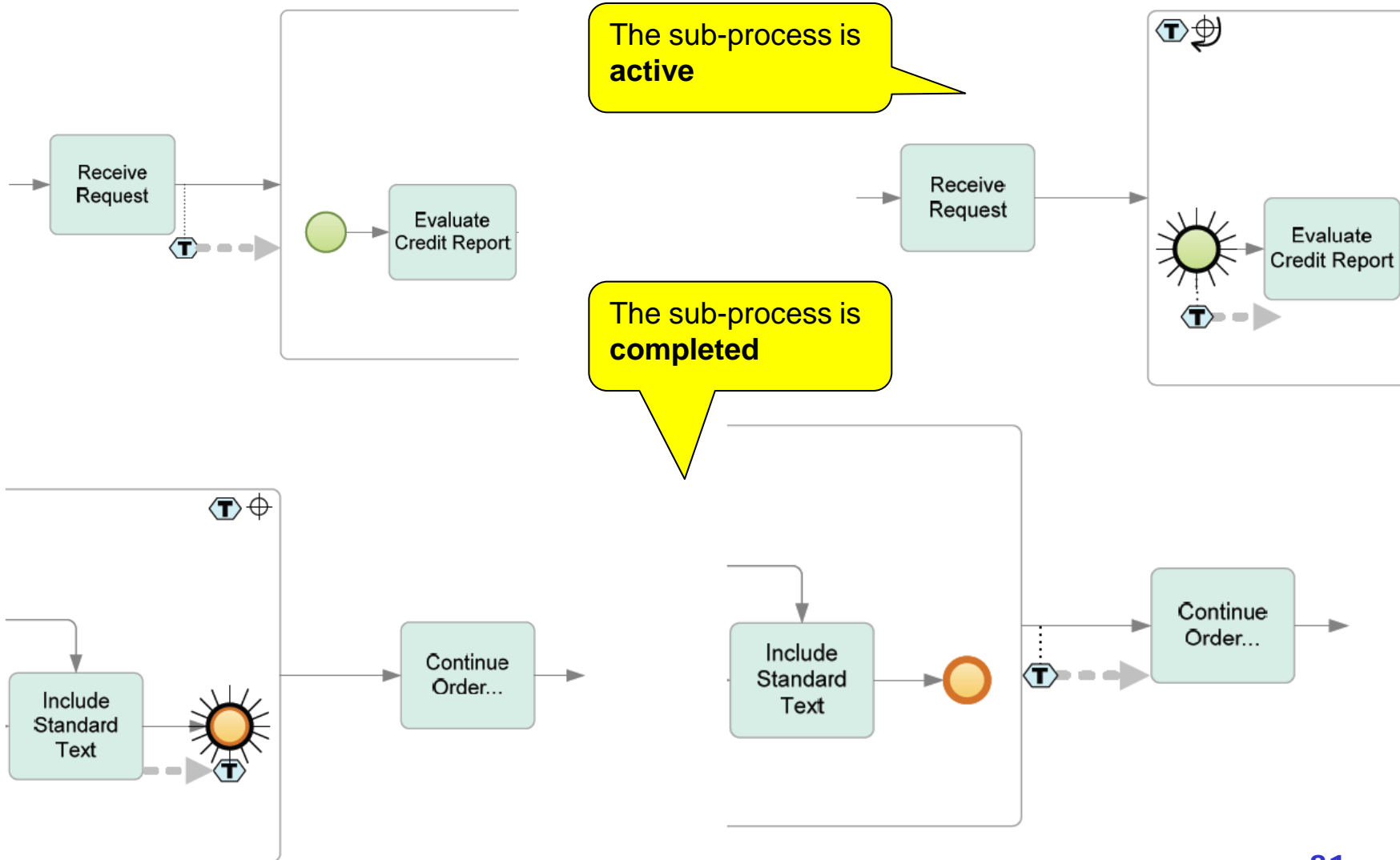


- A separately modeled process that could be used in multiple contexts (e.g., checking the credit of a customer). The “process relevant data” of the parent (calling) process is not automatically available to the sub-Process. **Any data must be transferred specifically**, sometimes reformatted, **between the Parent and Sub-Process.**

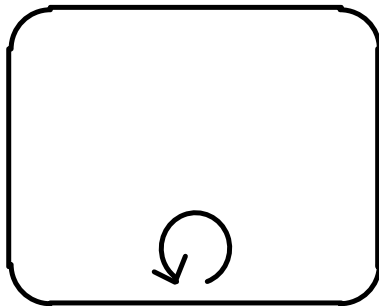


- Transferring data from the parent Process to the reusable Sub-Process will rely on a “mapping” between the data elements of the two levels.
- Just like an embedded Sub-Process, an Independent Sub-Process **must have** a None Start Event.

# Behaviour across process levels



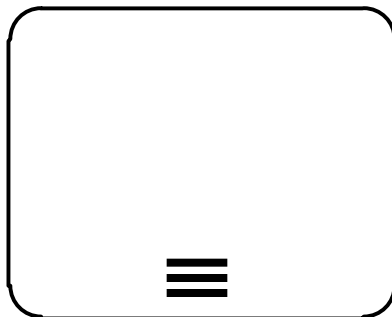
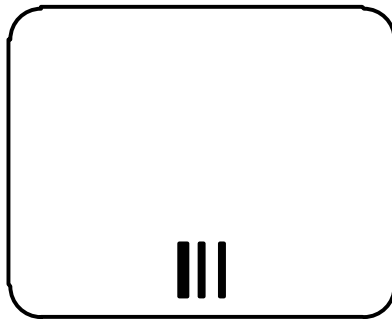
# Looping



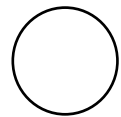
- On an Activity, it is possible to define a loop condition that determines the number of times to perform that Activity. There are two variations for Activity looping :
  - **While Loop (or While-Do)** - The loop condition is checked *before* the Activity is performed. If the loop condition turns out 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).



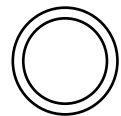
# Multi-Instance Activities



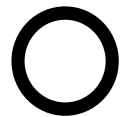
- Activity to be performed many times with different data sets.
- The value of the loop condition attribute determines the number of times that the Activity is performed. The condition must resolve to an integer.
- The individual instances of a Multi-Instance Activity might occur in *parallel* or in *sequence*.



Start



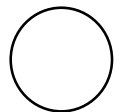
Intermediate



End

- An Event is something that “happens” 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.

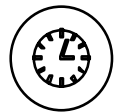
# Start Events



None



Message



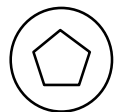
Timer



Conditional



Signal



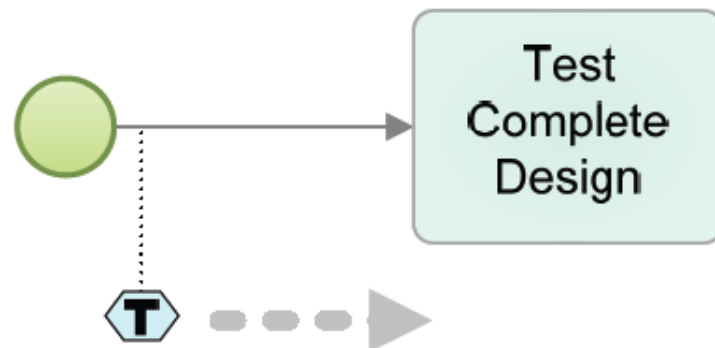
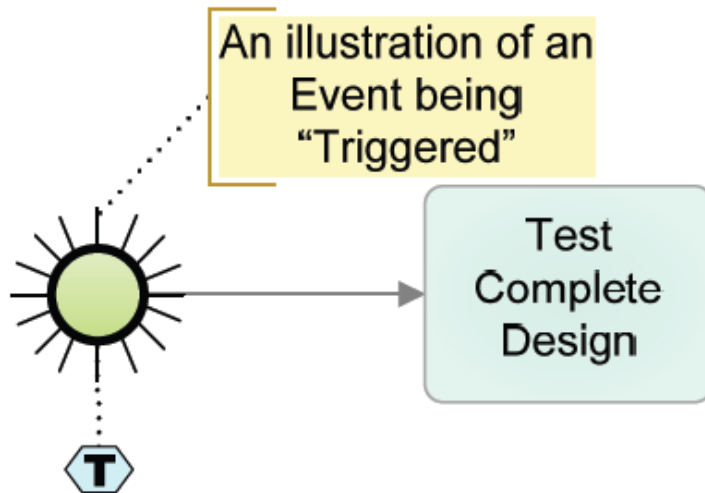
Multiple



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).

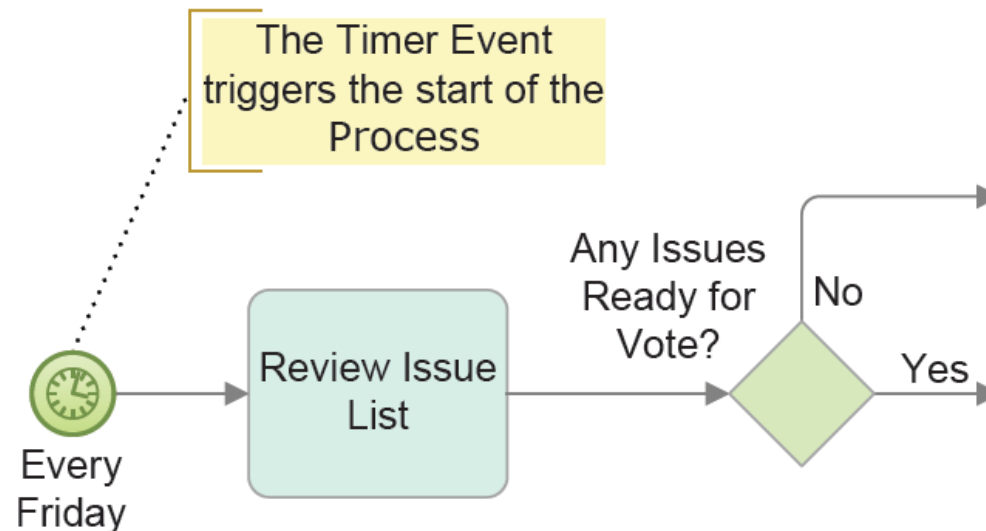
# Start Events Behaviour



- 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.
- 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.

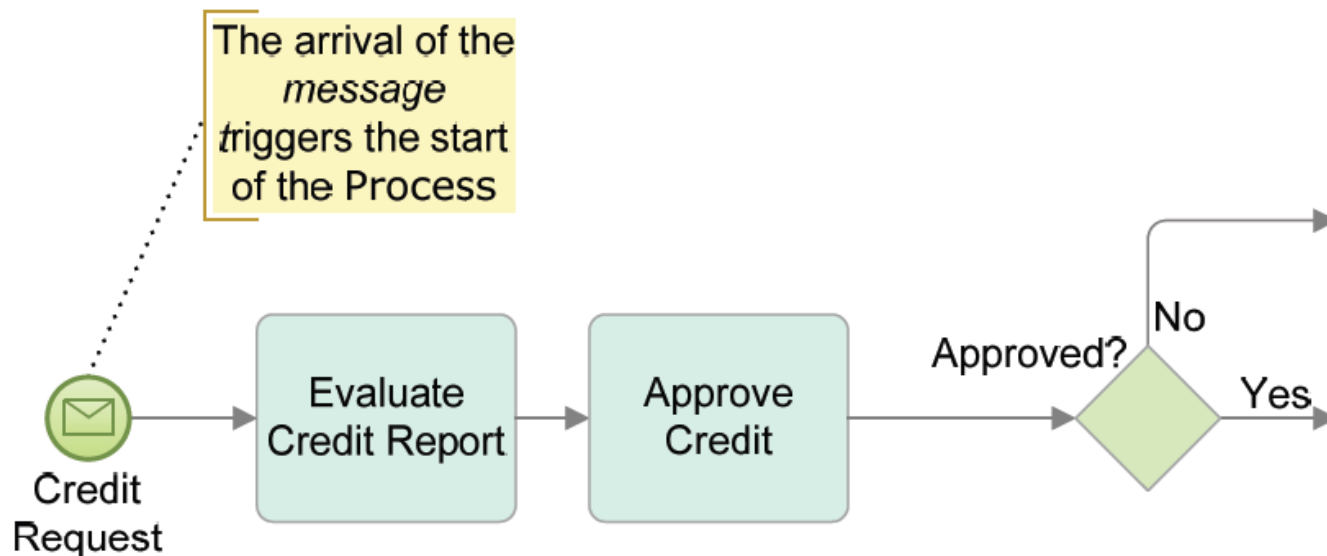
# Timer Start Event

- 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).



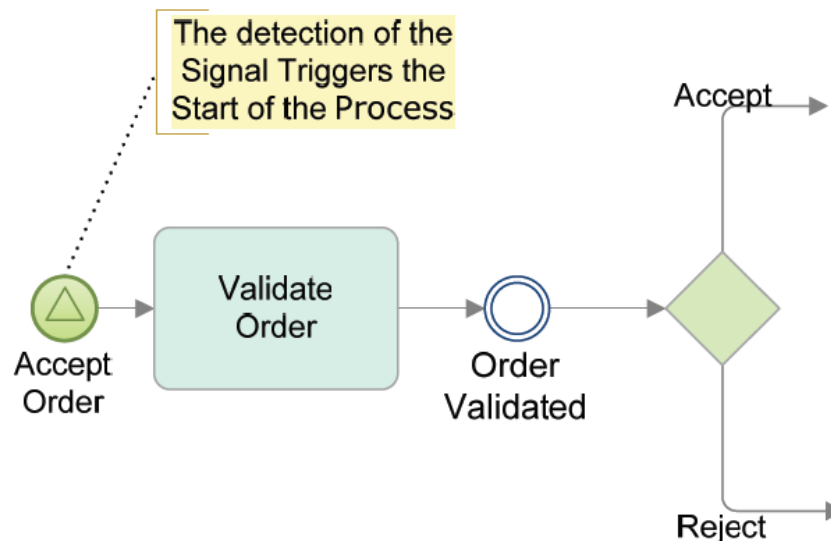
# Message Start Event

- 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)



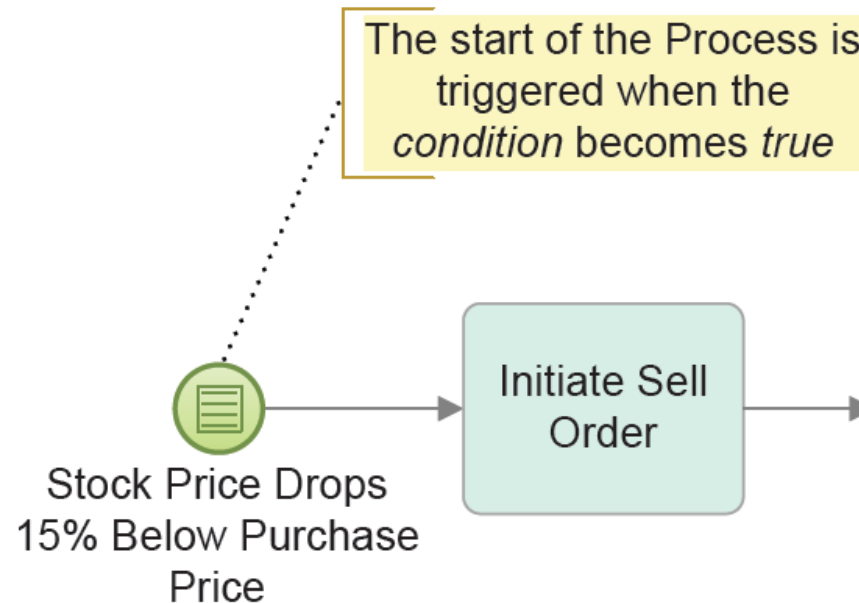
# Signal Start Event

- 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).



# 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.





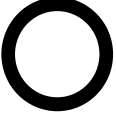




# Multiple and Parallel Multiple Start Events



- 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.

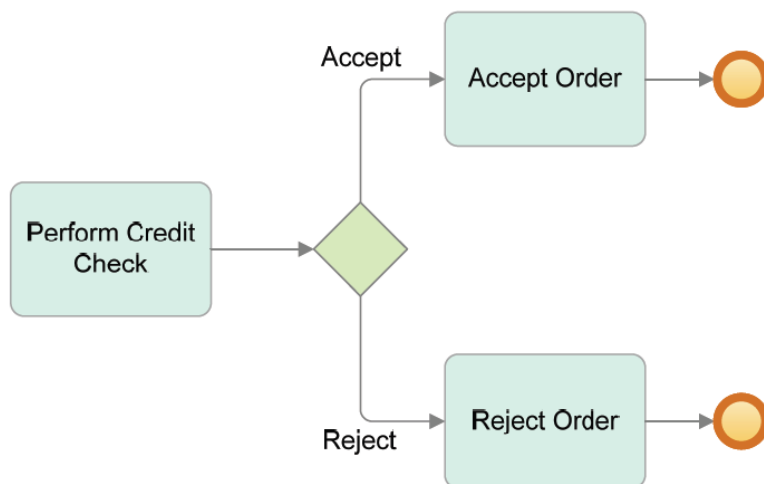
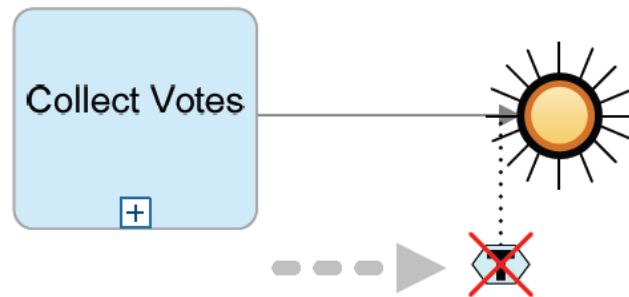
# End Events



-  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).

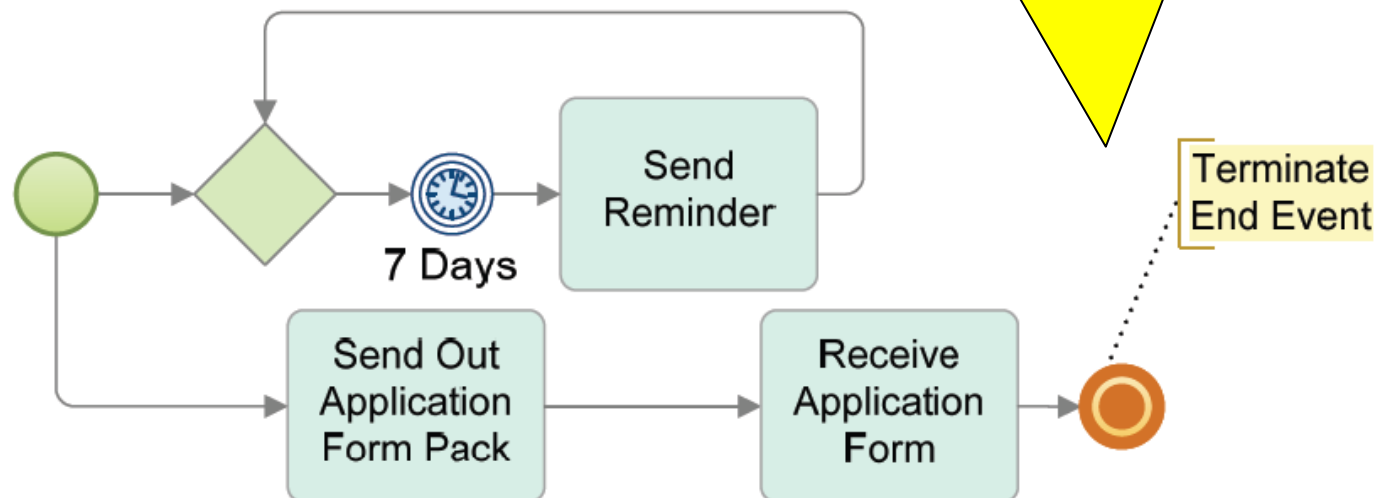
# End Events Behaviour



- 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.

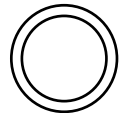
# Terminate End Event

- 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.



# Intermediate Events

## Catching Throwing



None



Timer



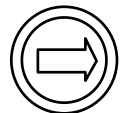
Message



Signal



Conditional



Link



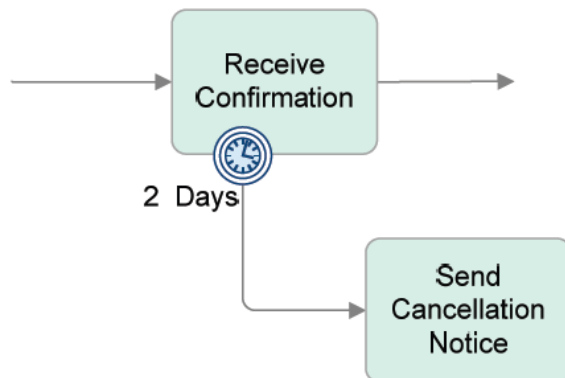
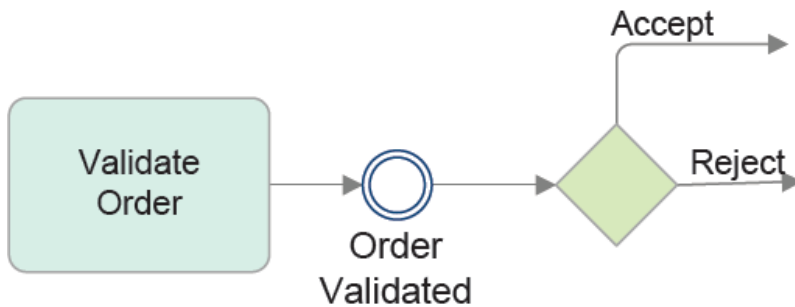
Multiple



Parallel  
Multiple

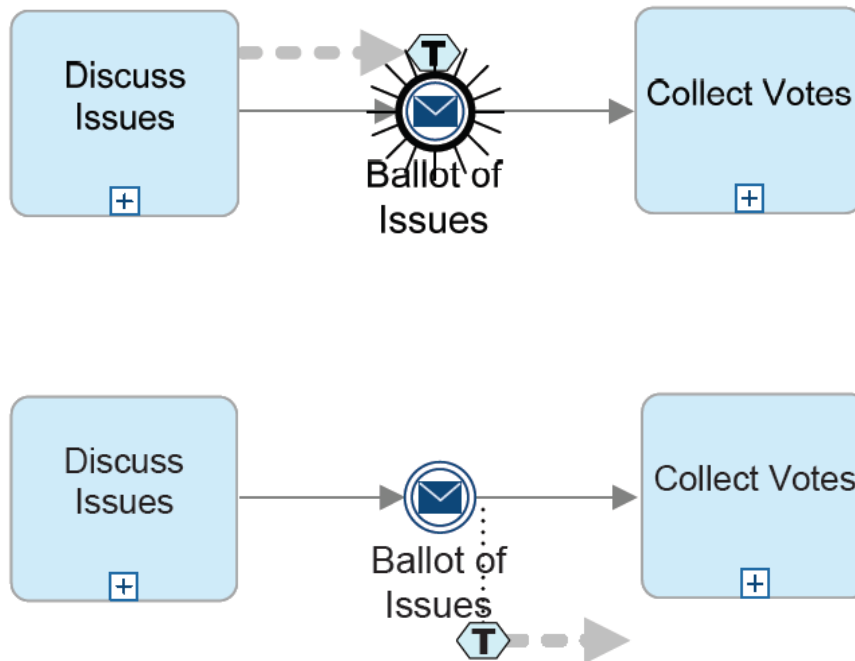
- An Intermediate Event indicates where something happens/occurs after a Process has started and before it has ended.
- They may also interrupt the normal processing of an Activity.
- Each type of Intermediate Events can either *throw* or *catch* the event.
- A *catching Intermediate Event* waits for 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).

# Catch Intermediate Events Behaviour



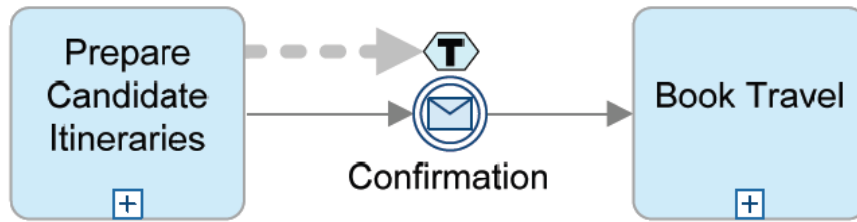
- A token arriving at a catch Intermediate Event would wait until the trigger occurs. Then the token would leave immediately and move down the outgoing Sequence Flow.
- A Catching Intermediate Event (except than the Link Event) can also be attached to the boundary of an Activity.
- When the Activity 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



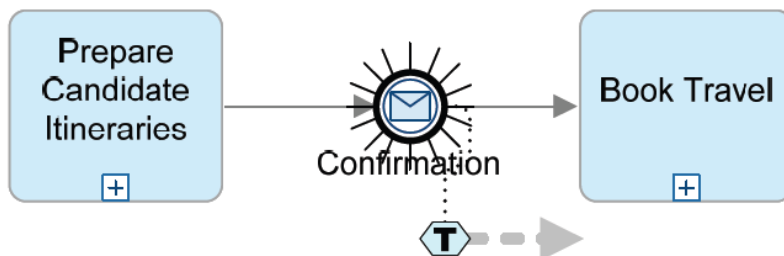
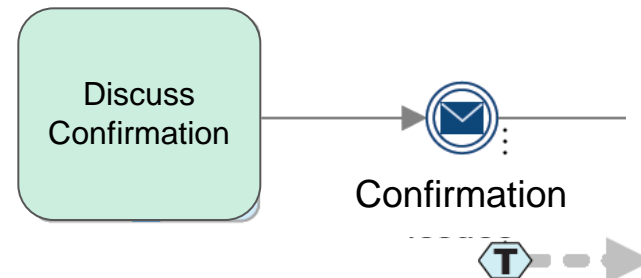
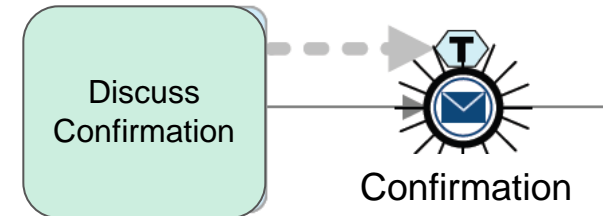
- A token arriving at a throw Intermediate Event would immediately fire the trigger. It would then leave immediately and travel down the outgoing Sequence Flow.
- A Throwing Intermediate Event **can not be attached** to the boundary of an Activity.

# Intermediate Events Behaviour



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

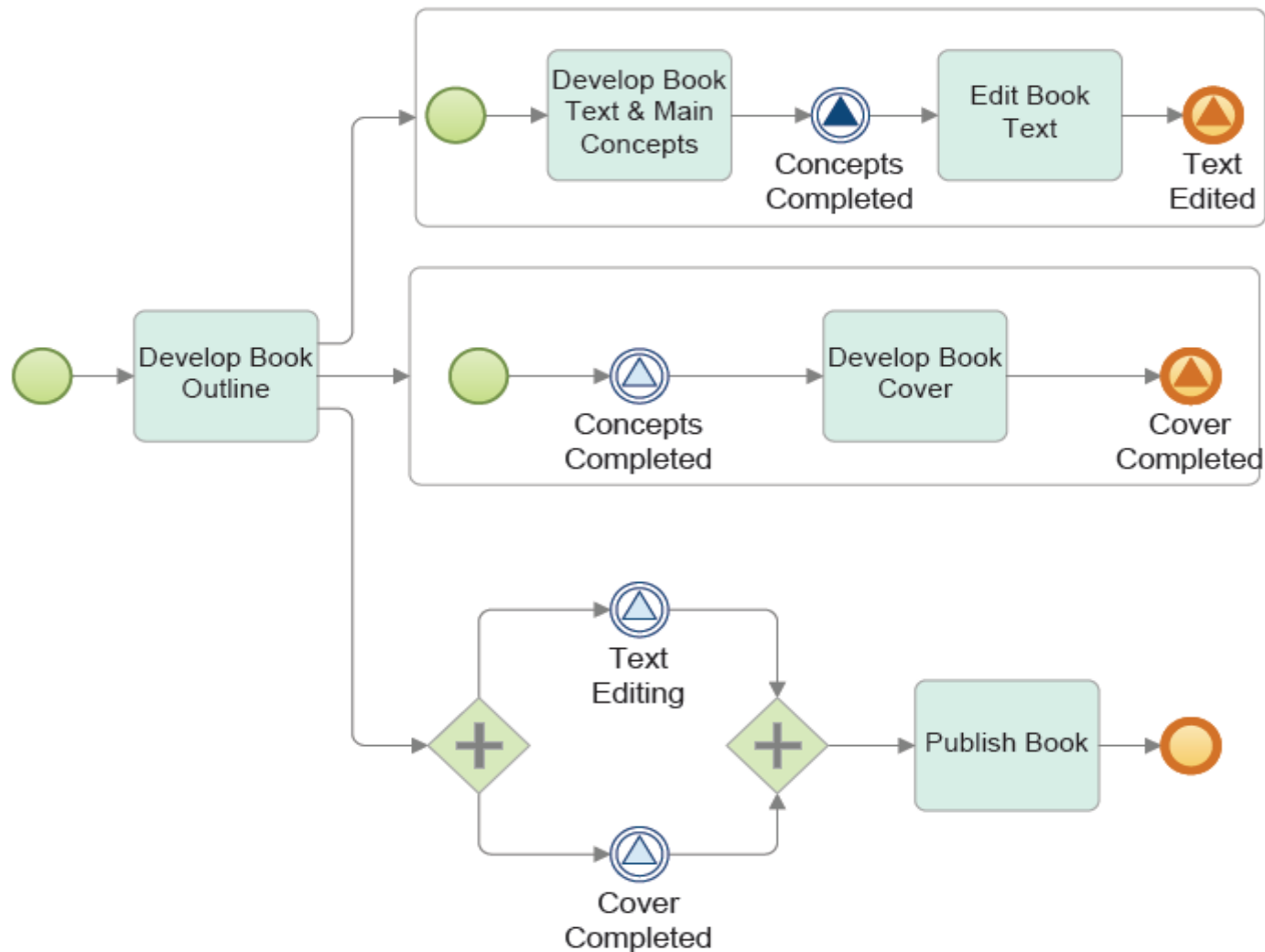
- When a token arrives at a throwing Message Intermediate Event, it immediately triggers the Event, which sends the message to a specific participant.



If the token is waiting at the Intermediate Event and the message arrives, then the Event triggers.

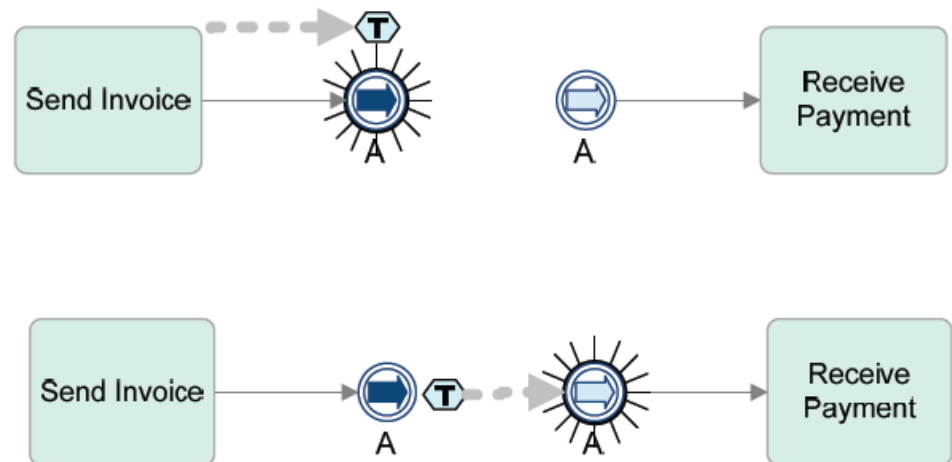


# An Example

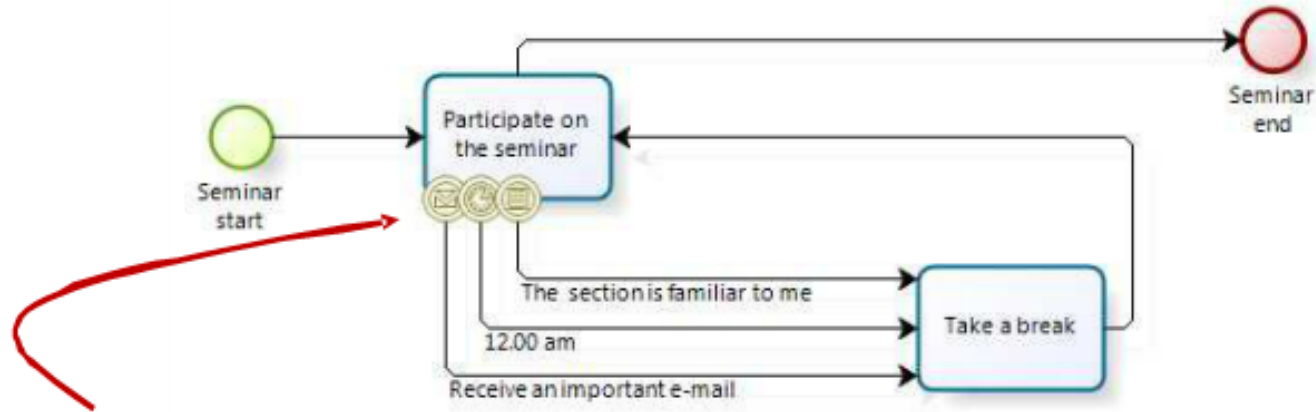


# Link Intermediate Event

- 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..

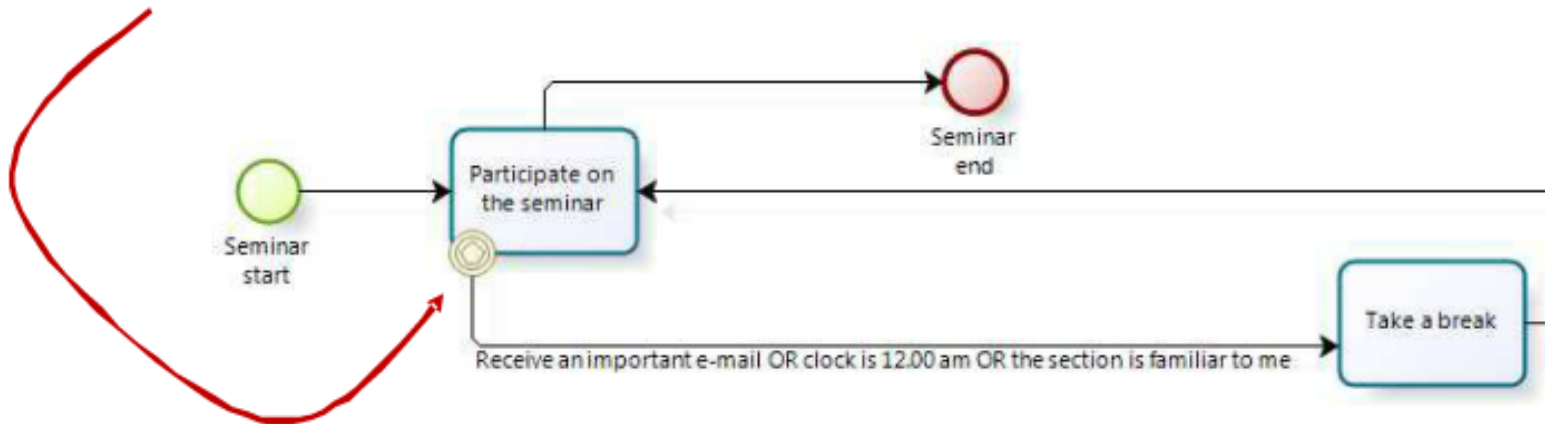


# Example of a Multiple Intermediate Event

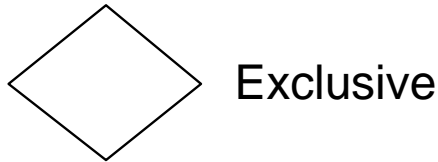


Different triggers  
or events  
can cause a break

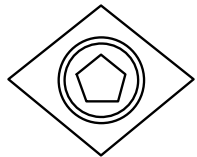
Is equal to:



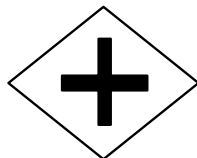
# Gateways



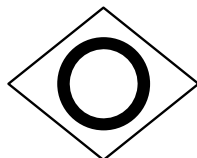
Exclusive



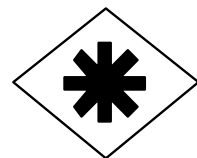
Event



Parallel



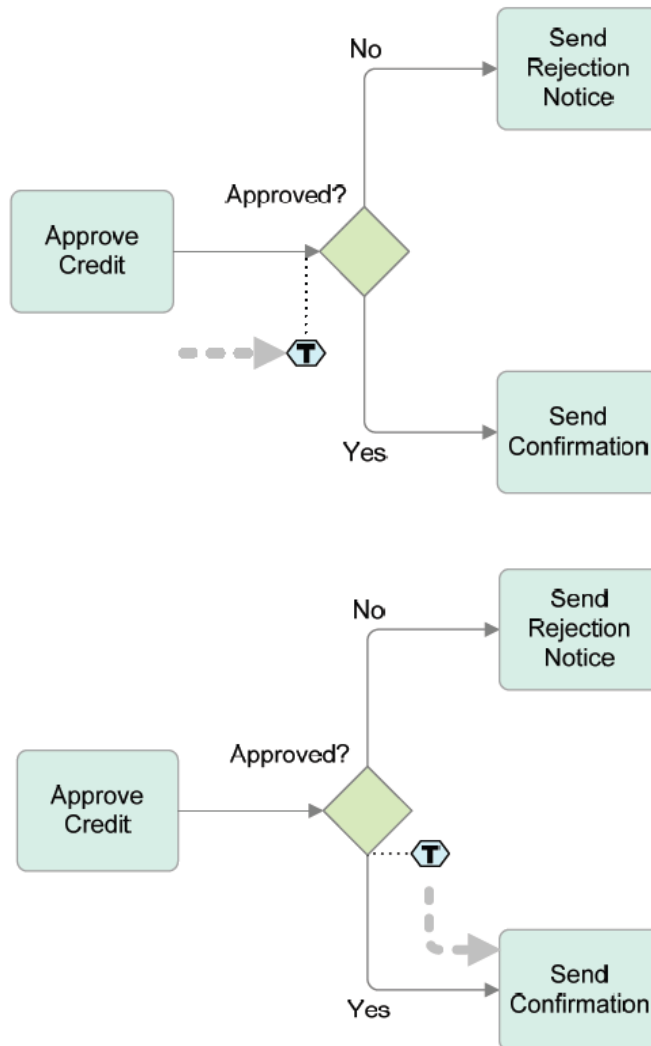
Inclusive



Complex

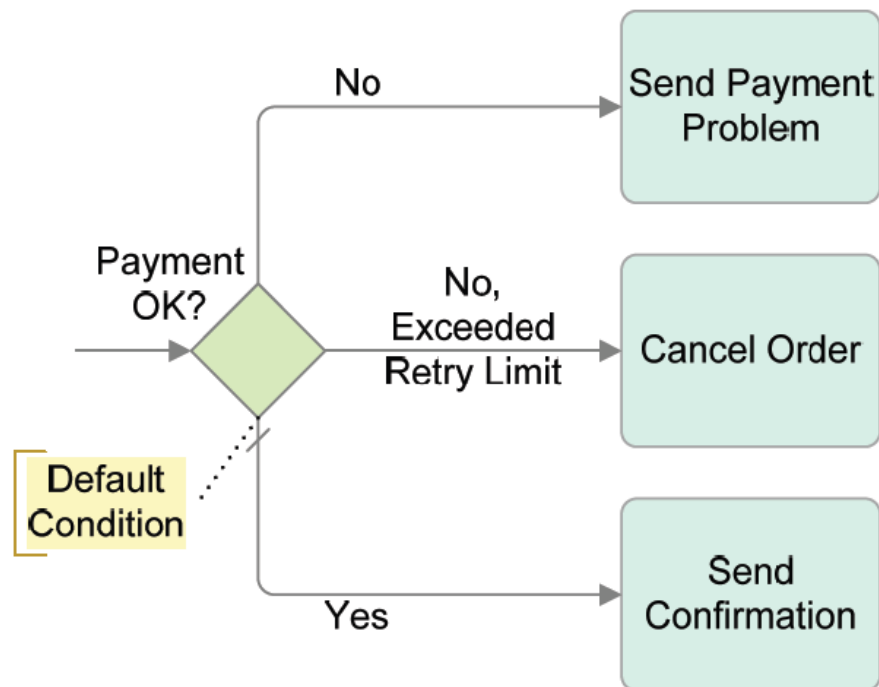
- Gateways are modeling elements that control how the Process diverges or converges.
- They represent points of control for the paths within the Process.
- They split and merge the flow of a Process (through Sequence Flow).
- Since there are different ways of controlling the Process flow, there are different 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*.

# 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

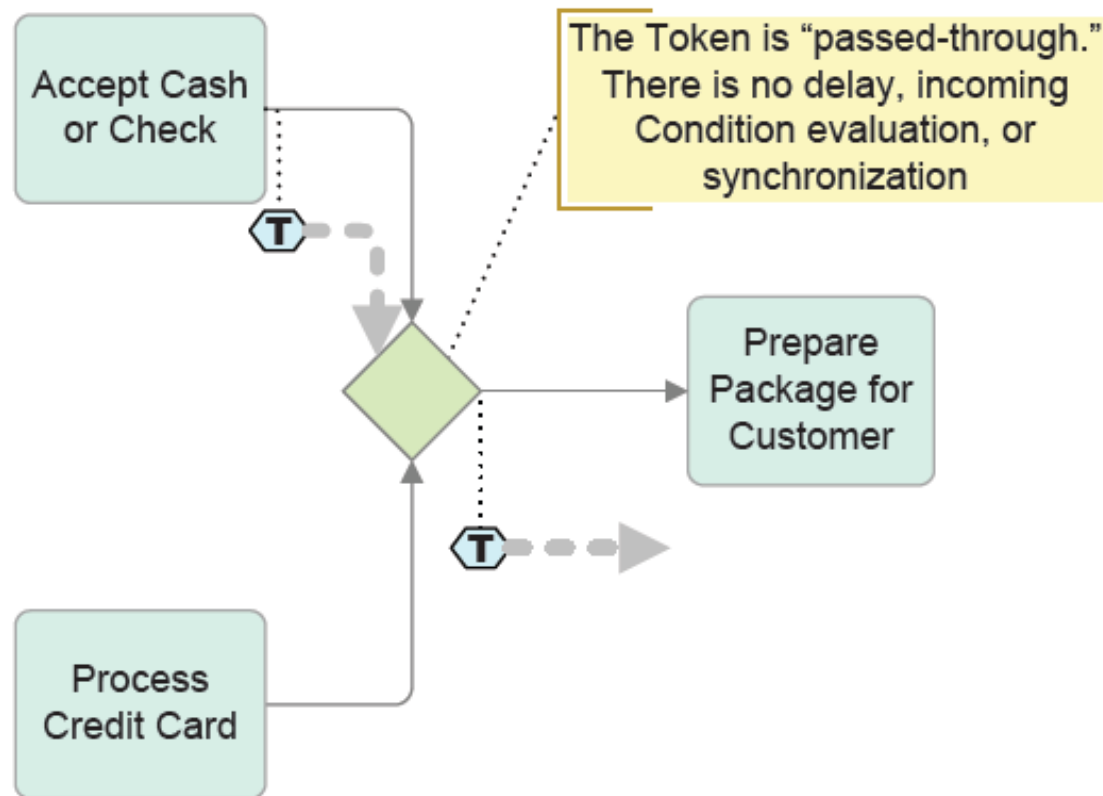


- One way for the modeler to ensure that the Process does not get stuck at an Exclusive Gateway is to use a *Default Condition* for one of the outgoing Sequence Flow.
- The default condition can complement a set of standard conditions to provide an automatic escape mechanism in case all the standard conditions evaluate to false.
- The Default is chosen if all the other Sequence Flow conditions turn out to be false.

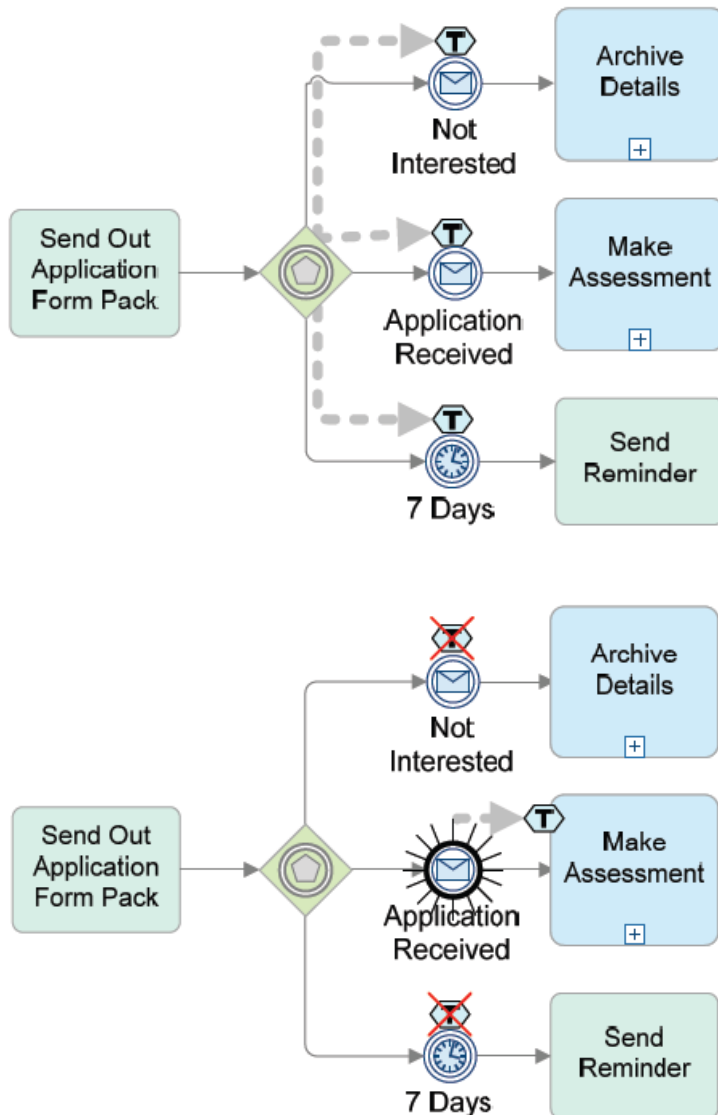
# Exclusive Gateways – Merging Behaviour



- 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.



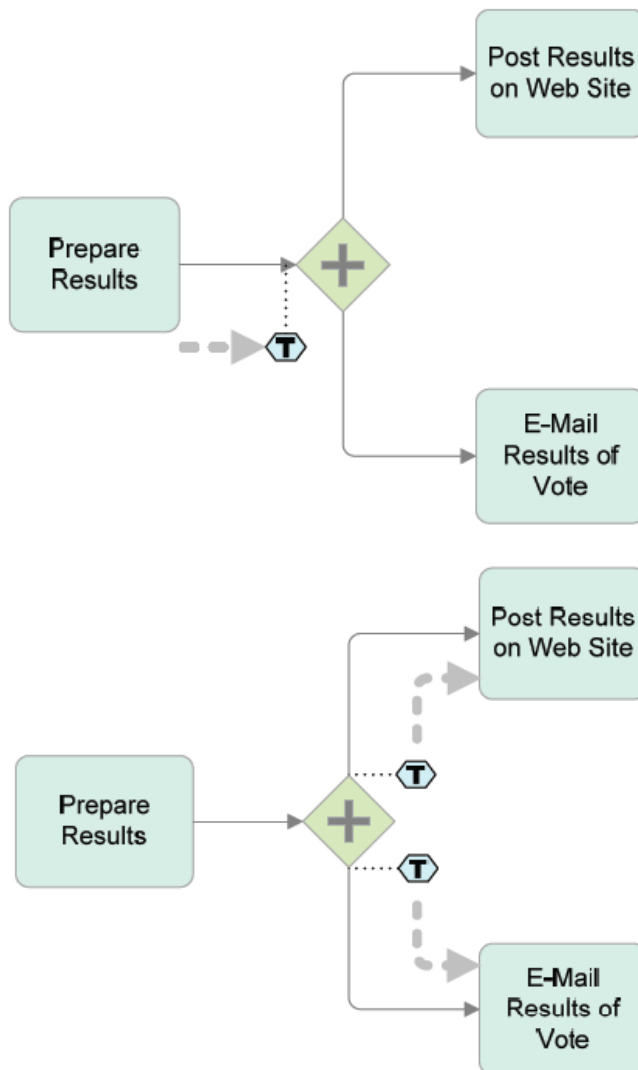
# Event-based Exclusive Gateways



- 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.



# Parallel Gateways – Splitting Behaviour

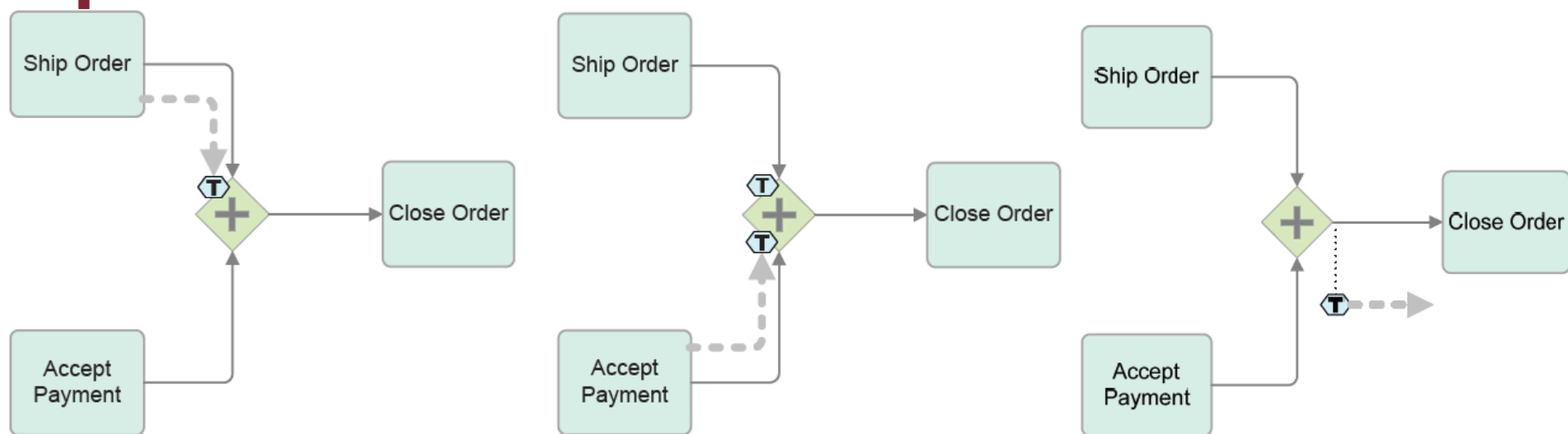


- When a *token arrives at a Parallel Gateway*, there is no evaluation of any *conditions on the outgoing Sequence Flow*
- The Parallel Gateway will create parallel paths.
- This means that the Gateway will create a number of tokens that are equal to the number of outgoing Sequence Flow. One token moves down each of those outgoing Sequence Flow.

# Parallel Gateways – Merging Behaviour



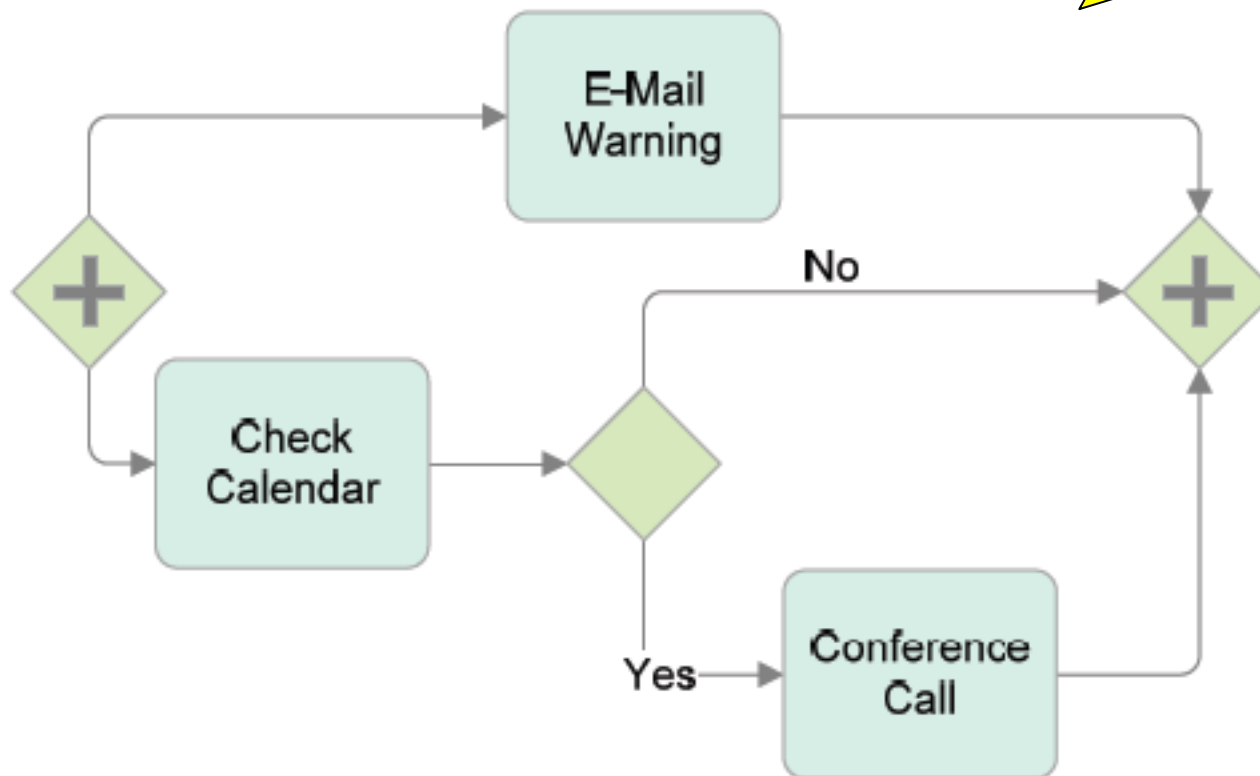
- To synchronize the flow, the Parallel Gateway will wait for a token to arrive from each incoming Sequence Flow.
- When the first token arrives, there is no evaluation of a condition for the incoming Sequence Flow, but the token is “held” at the Gateway and does not continue.
- When all the tokens have arrived, then they are merged and one token moves down the outgoing Sequence Flow



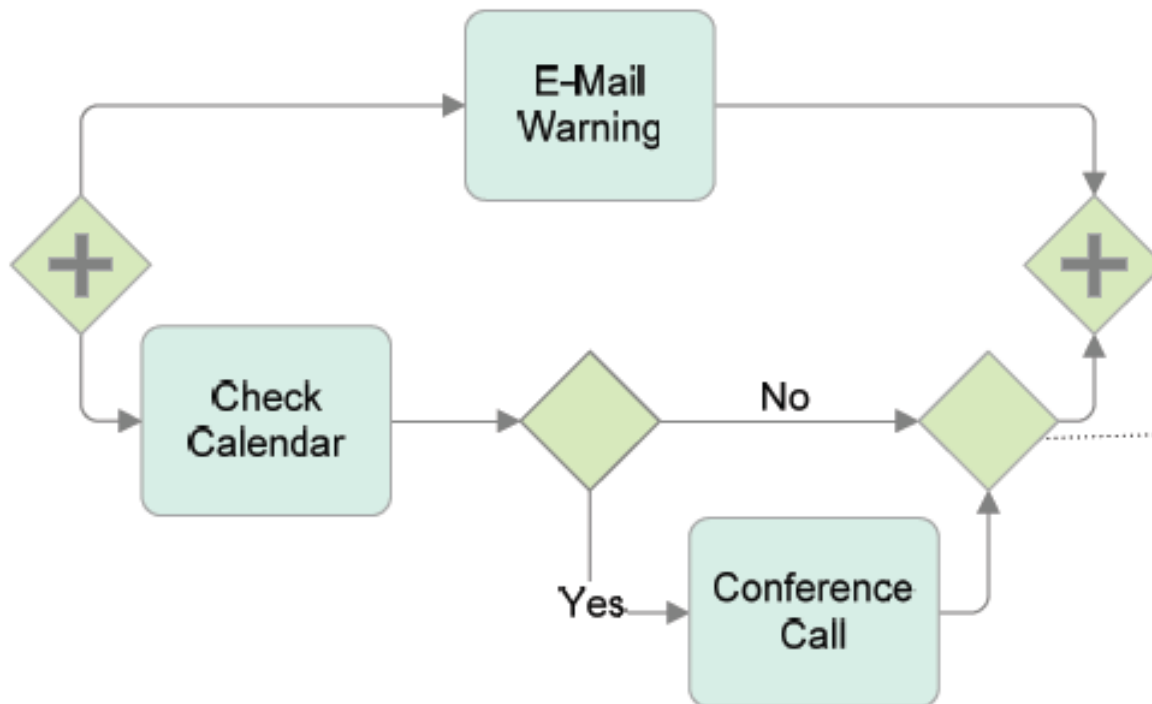
# Exercise

- 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

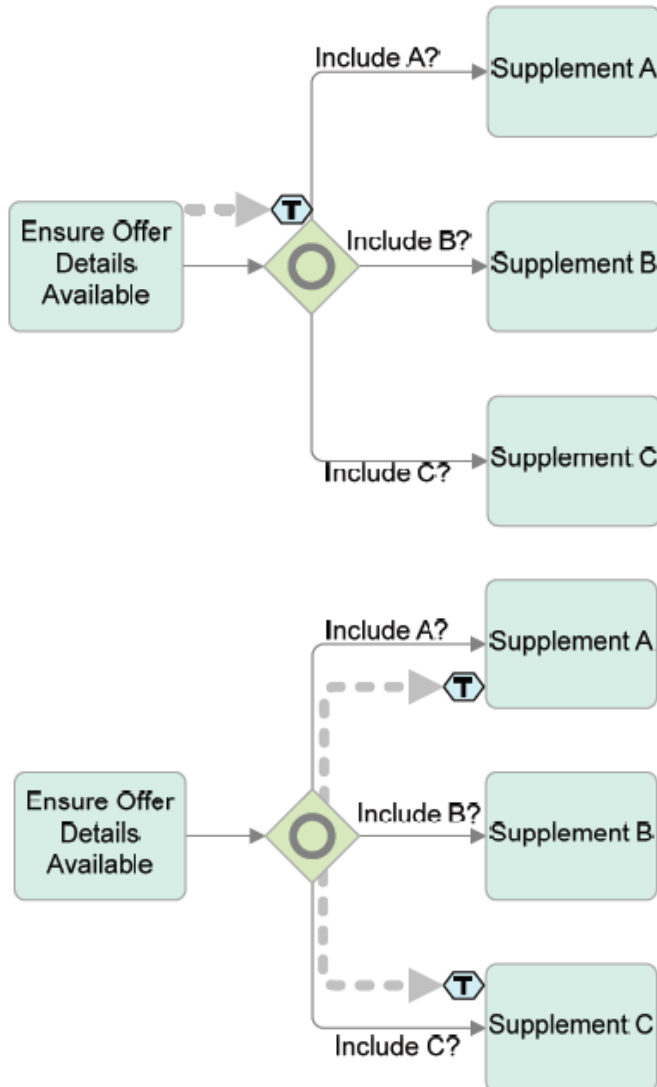


# Exercise



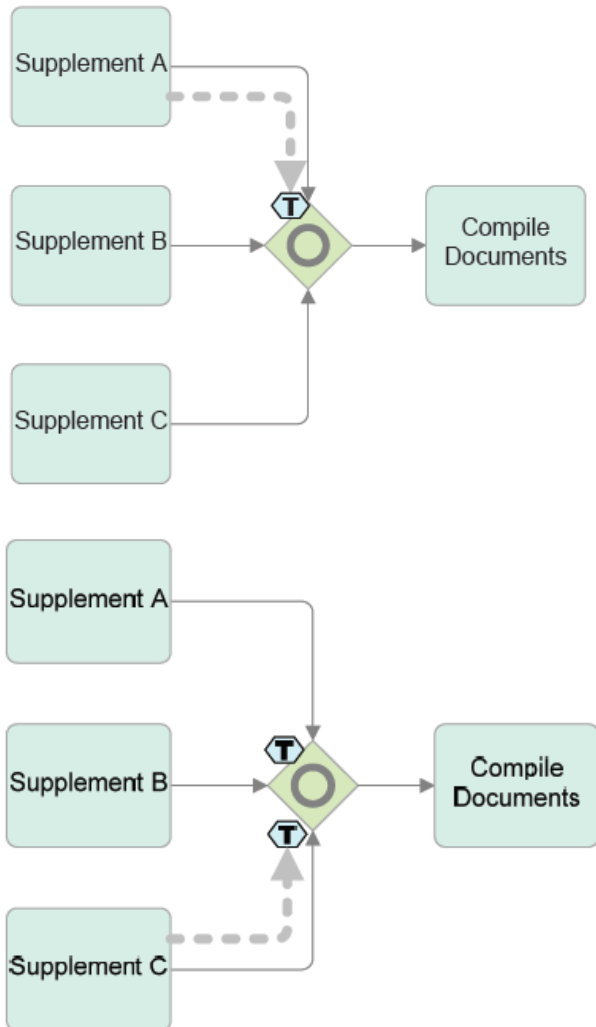
The intervening Exclusive Gateway reduces the number of incoming paths to the Parallel Gateway.

# 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 Flow creates one or more paths based on the conditions on those Sequence Flow.
- When a token arrives at an Inclusive Gateway, there is an immediate evaluation of all the conditions that are on the Gateway's outgoing Sequence Flow.
- 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.

# 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.



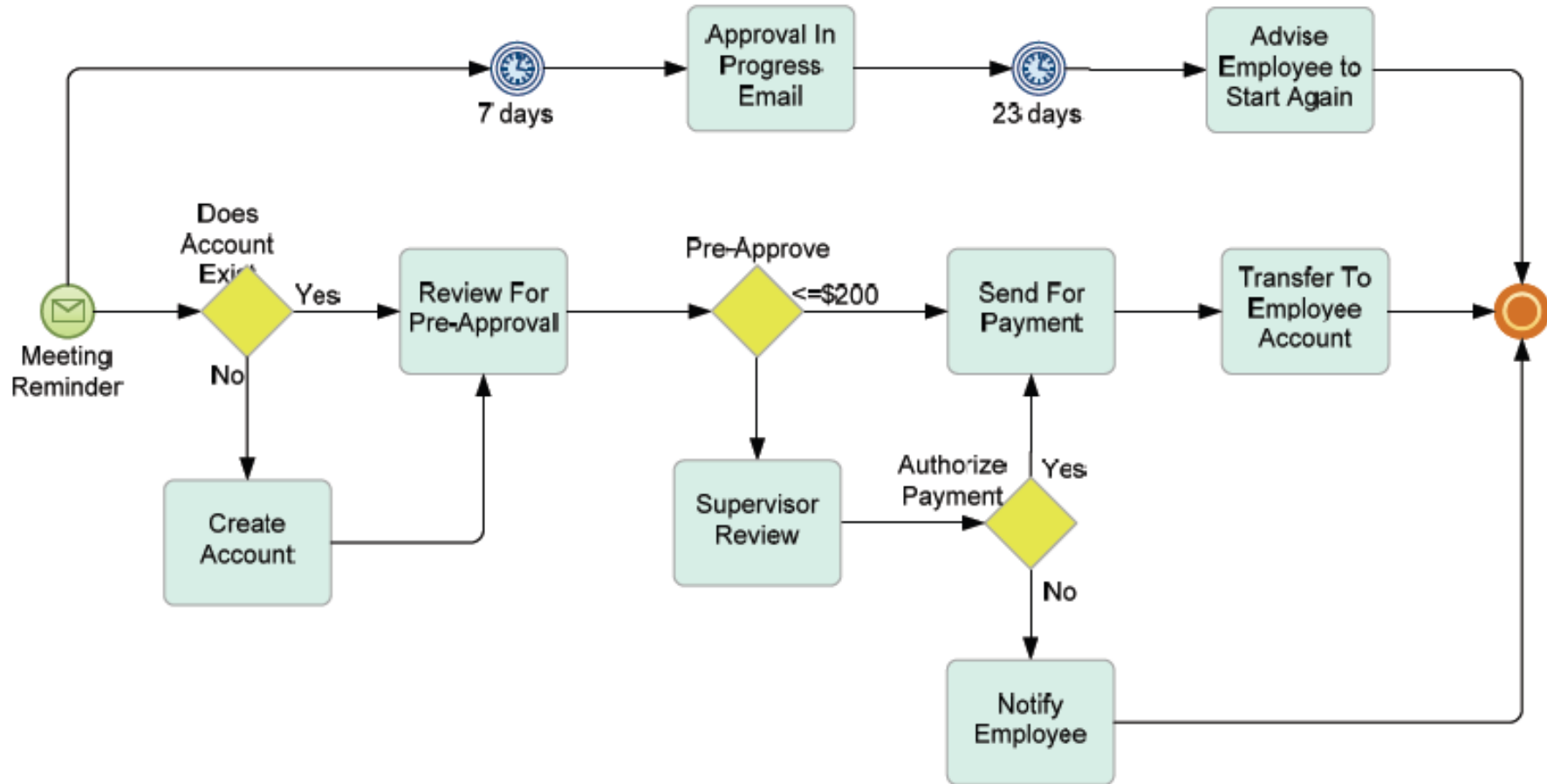


# Exercise

*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





# Topics

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



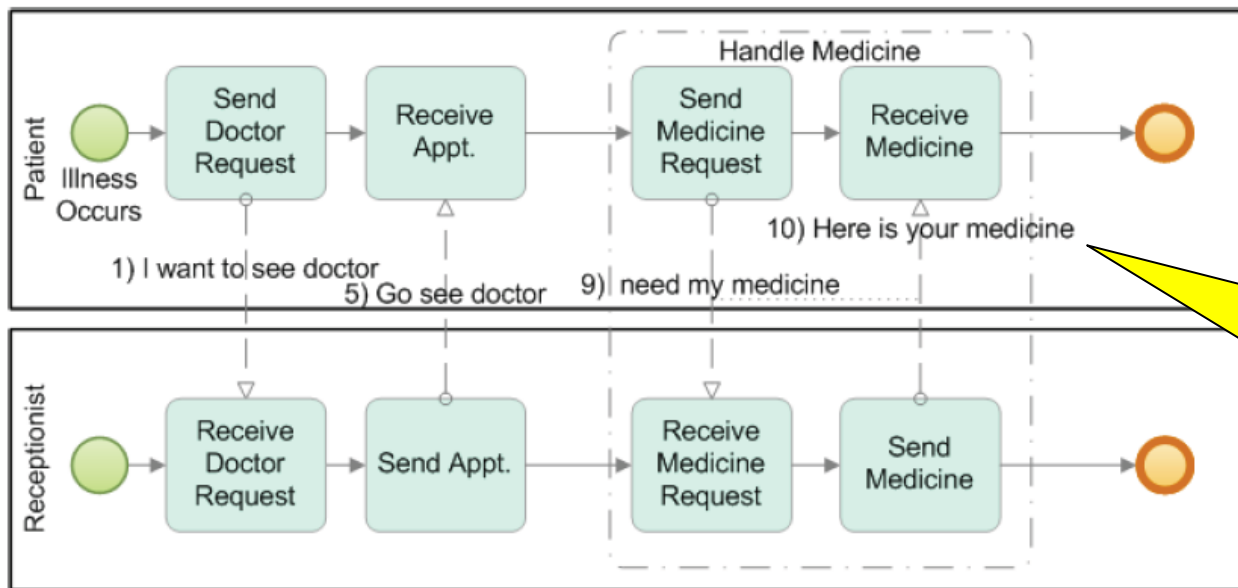
# Advanced Concepts

- Artifacts
  - Text Annotations, Groups, Data Objects
- Error Events and Exception Handling
- Cancel Events, Compensation Events and Transactions
- Ad Hoc Processes
- Swimlanes
- Conversation Diagrams
- Coreography Diagrams

# Artifacts

- **Artifacts** provide a mechanism to capture additional information about a Process, beyond the underlying flow-chart structure. 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.

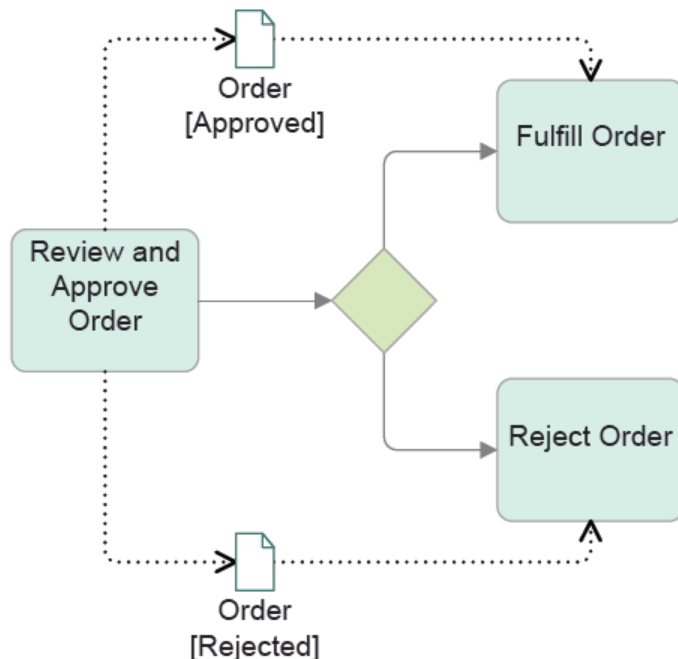


Groups cannot be interrupted by Intermediate Events

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

# Data Objects

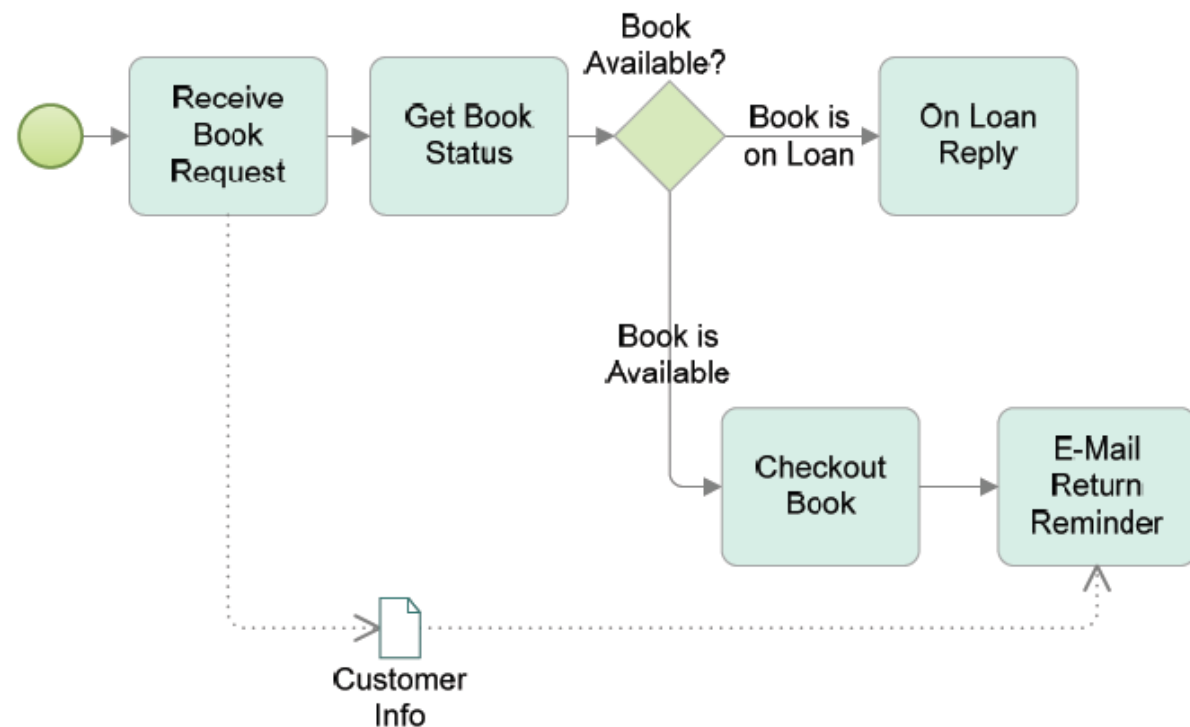
- Data Objects are used to show how data and documents are used within a Process. While Data Objects do not affect the structure and flow of the Process, they are intimately tied to *performance of Activities*.
- Data Objects can be used to define *inputs* and *outputs* of activities.



- Data Objects 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

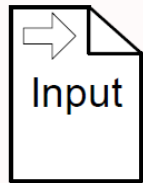
- **Data flow** represents the movement of Data Objects from into and out of Activities.
- In BPMN, data flow **is decoupled** from the Sequence Flow.
- It is possible to combine the Sequence Flow and the data flow when they coincide, but they are natively separated to allow modeling flexibility.



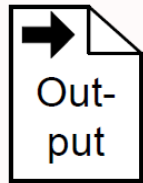
# New Data Objects in BPMN 2.0



- 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.



# Error Events



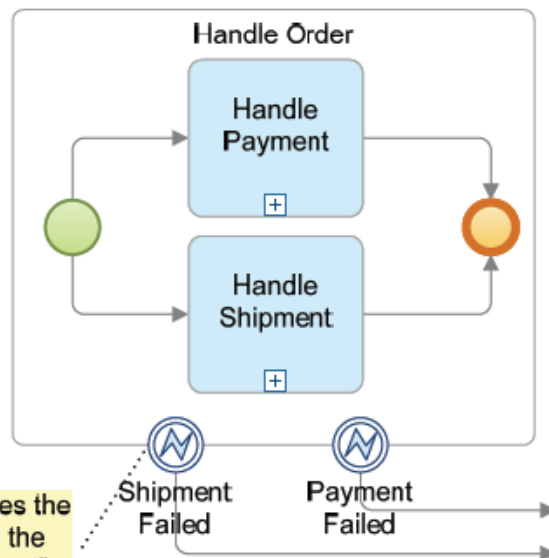
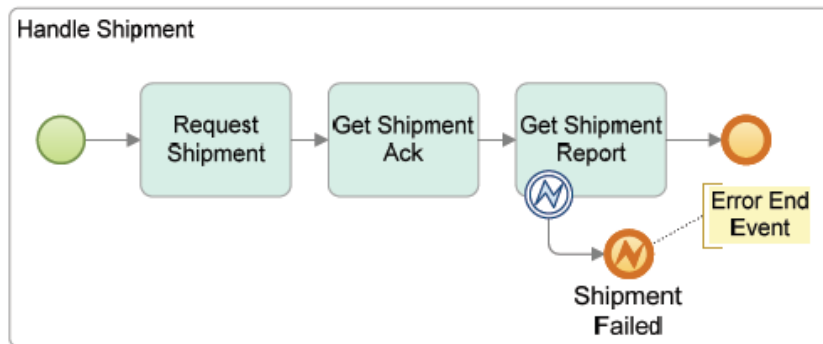
Error End Event –  
Catching



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



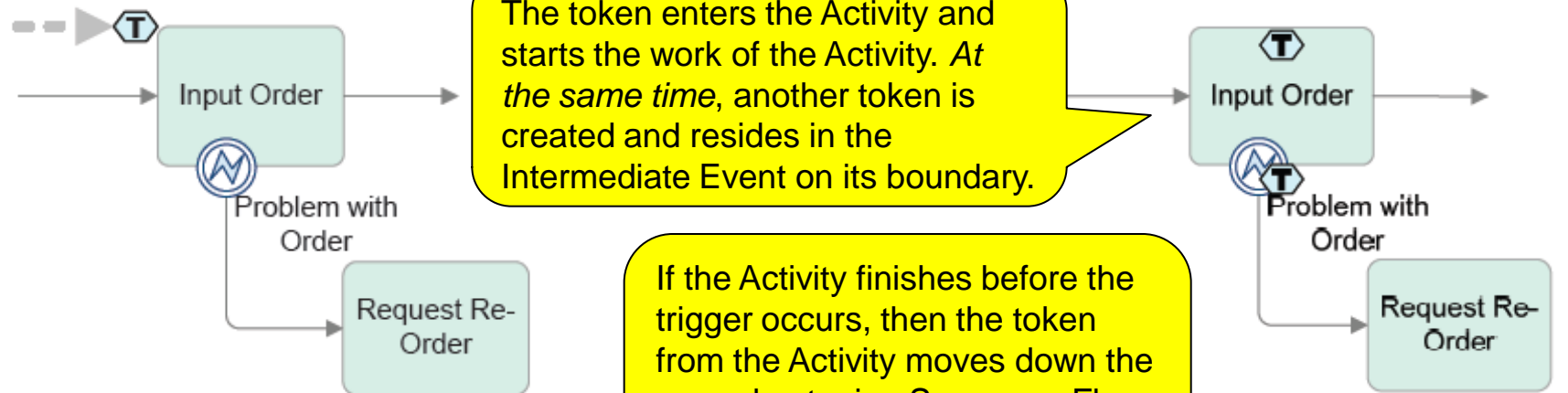
This Event catches the error thrown in the "Handle Shipment" Sub-Process

- The error thrown by the Event will be caught by an Intermediate Event at a higher level.
- Errors have a **specific scope of visibility**. An error can only be seen by a parent Process. 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.

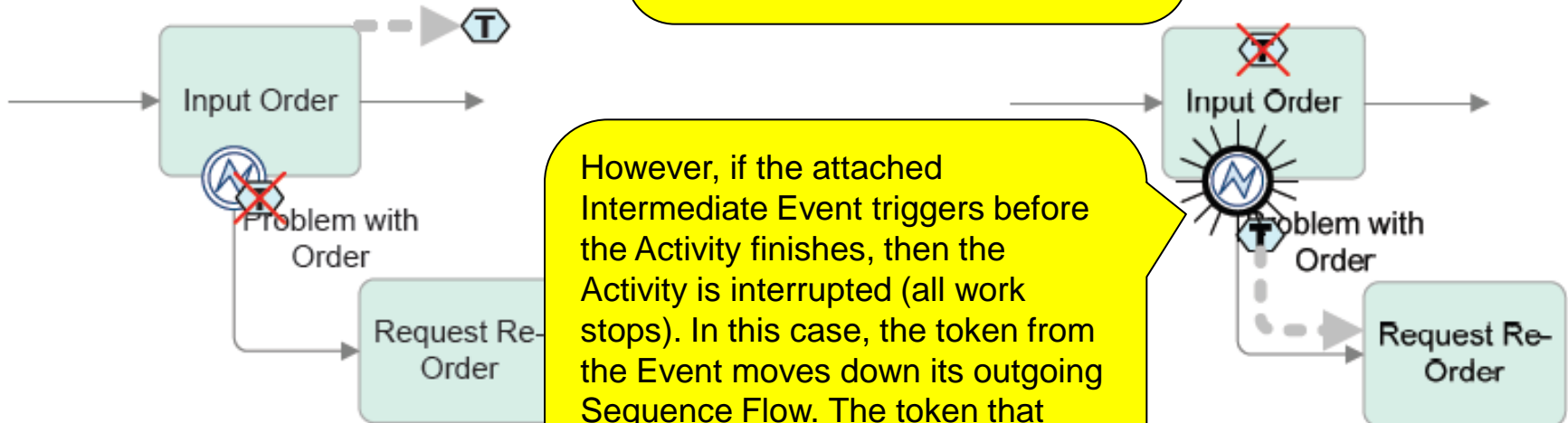


# Exception

The token leaves the previous flow object and arrives at the Activity with the attached Intermediate Event.



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.

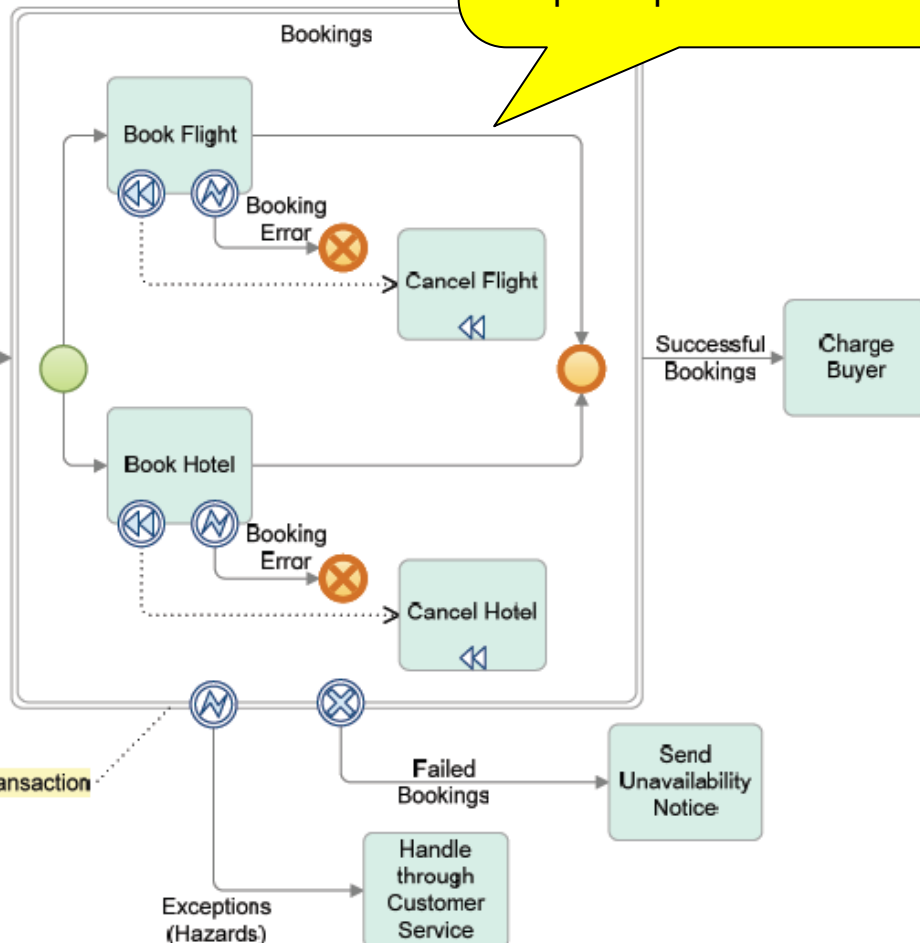


# Transactions

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

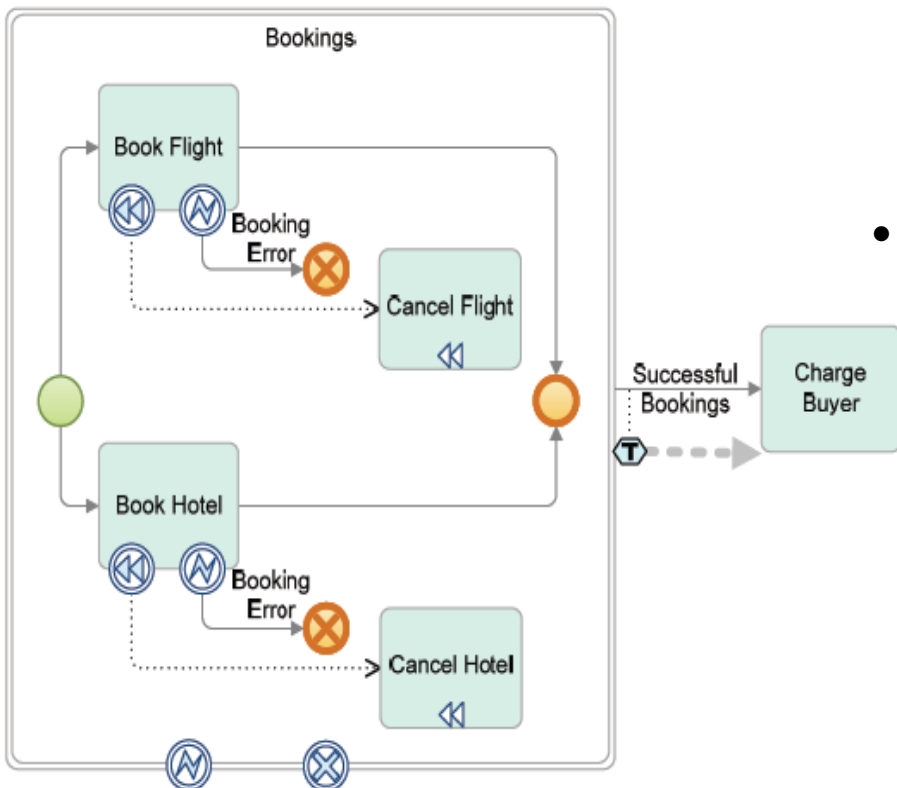
A Transaction is a set of activities that **logically belong together**.

- 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 any one of them 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.



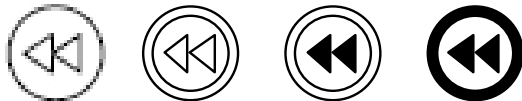
- *Secondly*, if the work of all the Activities in the Transaction Sub-Process complete normally and all the tokens reach an End Event, the Sub-Process is still not complete.
- *Thirdly*, if a processing or technical error occurs for one of the participants of the Transaction, then there are two possibilities 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



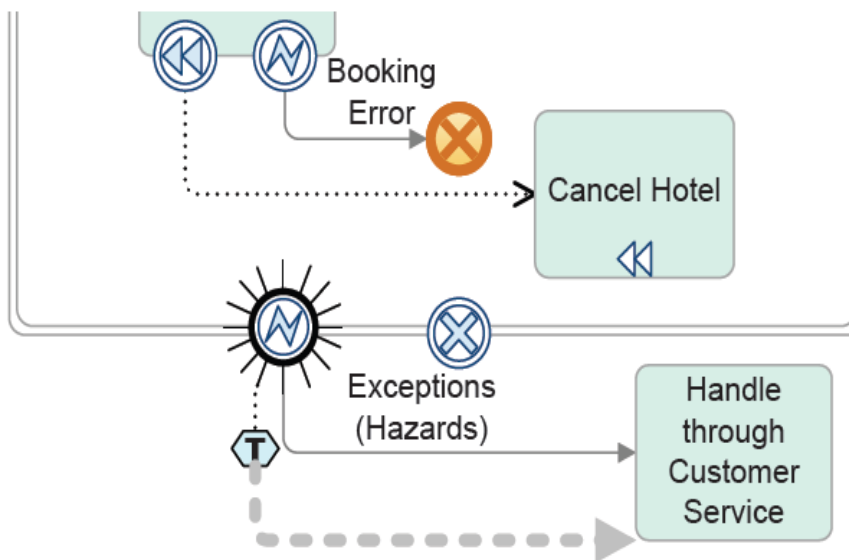
- The Cancel Intermediate Event is designed to handle a situation where a *transaction is canceled*.
- Cancel Intermediate Events can only **catch** a transaction cancellation. The Cancel End Event **throws** the cancellation.
- The Cancel Intermediate Event can only be attached to the boundary of a 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.

# 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.

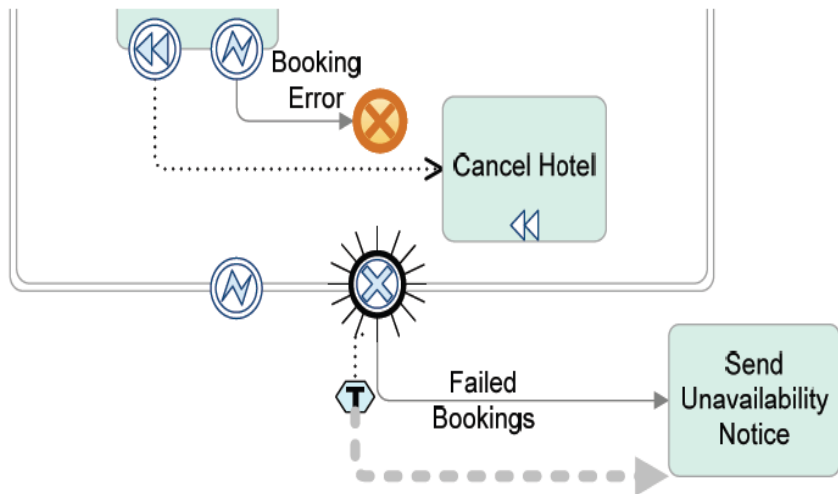
# Hazard in a Transaction Sub-Process



- 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 of 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.

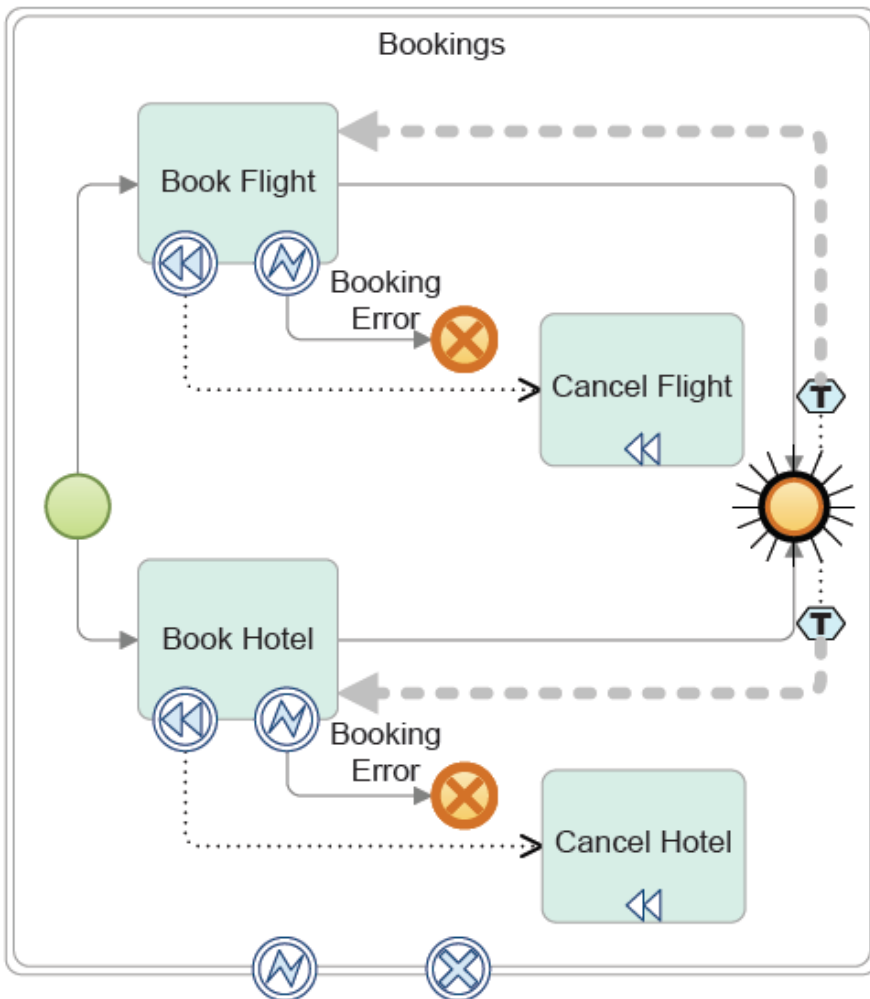


# Cancellation in a Transaction Sub-Process



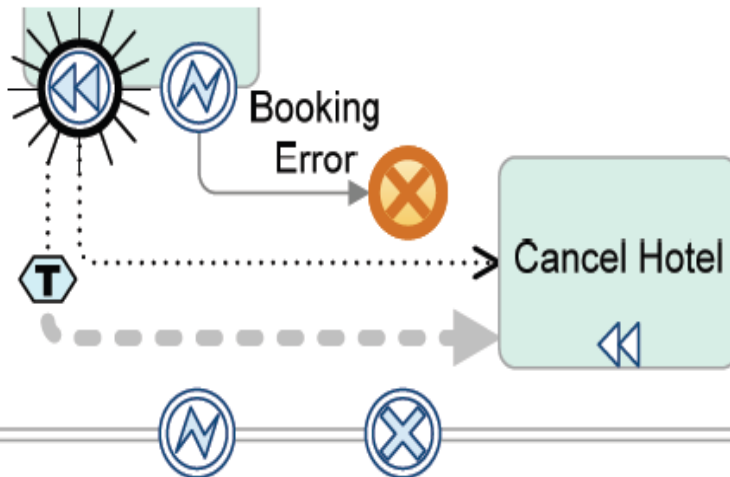
- a Transaction Sub-Process can be **cancelled** through an Event internal to the Sub-Process or through a cancellation sent through a transaction protocol.
- When a Transaction Sub-Process is cancelled, the Cancel Intermediate Event attached to its boundary is triggered.
- The token will eventually 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.

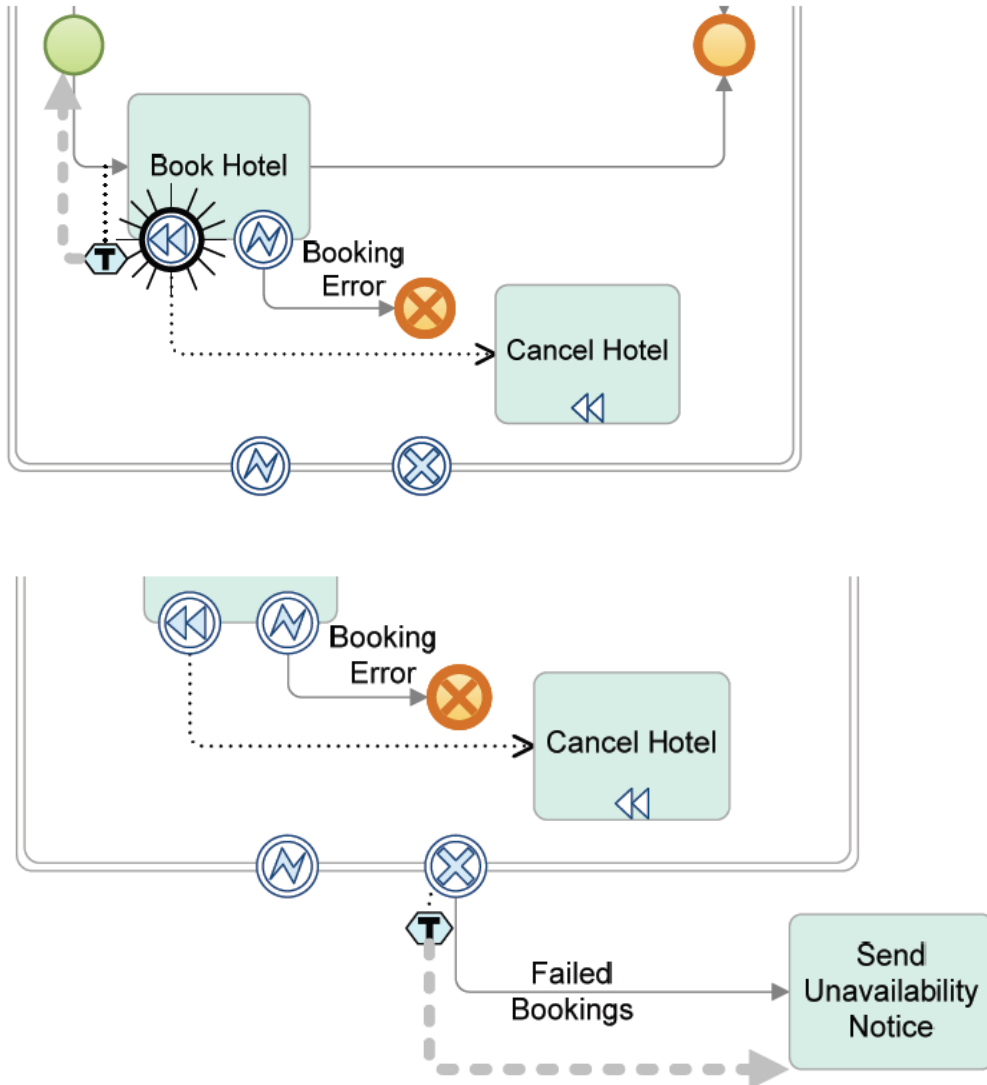
# Compensation in a Transaction Sub-Process



- 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.

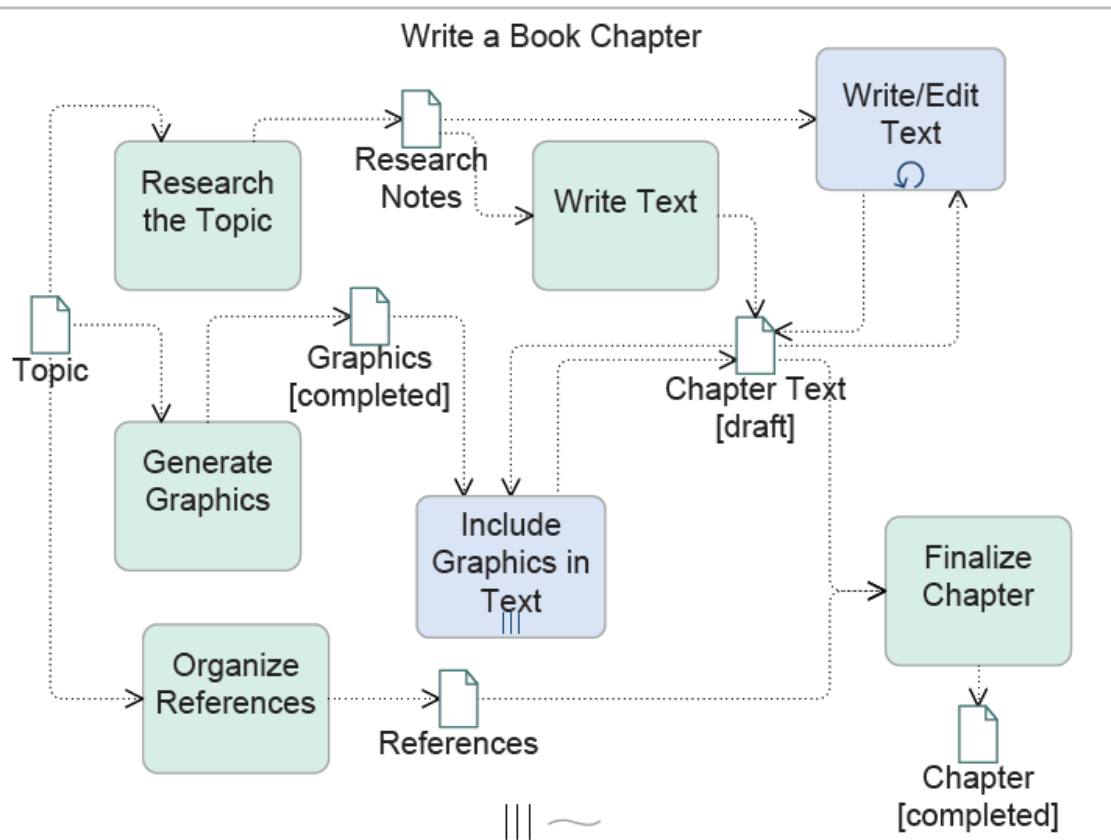
# Compensation in a Transaction Sub-Process



- 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.



- 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.

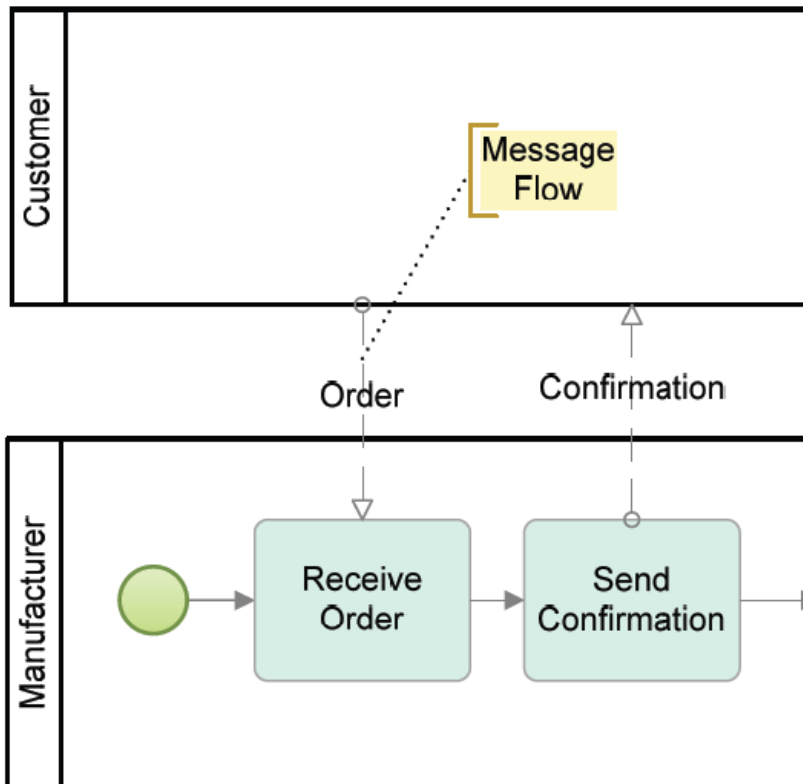


# Swimlanes

- BPMN uses “swimlanes” to help partition and/organize activities in a diagram. There are two main types:
- **Pools** - act as containers for a Process, each one representing a participant in a collaborative Business Process Diagram.
- **Lanes** - often assumed to represent internal business roles within a Process, Lanes actually provide a generic mechanism for partitioning the objects within a Pool based on the characteristics of the Process or elements.

# Message Flows

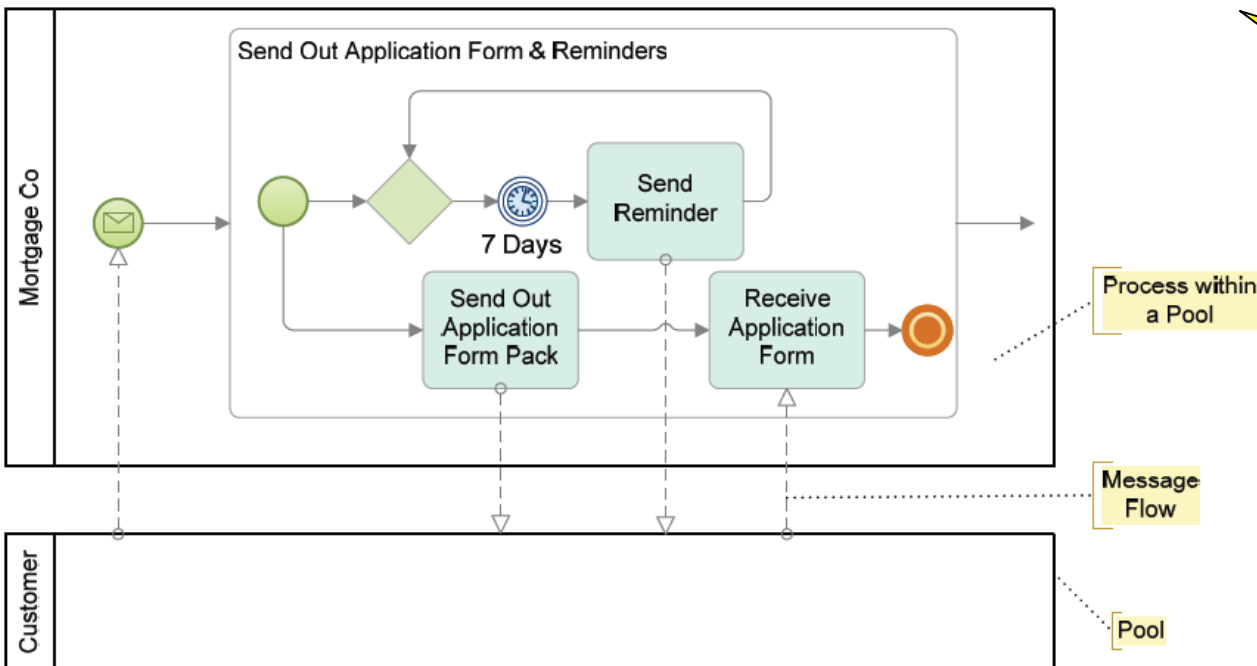
- Message Flow defines the messages/communications between two separate participants (shown as Pools) of the diagram.
- Message Flow must always occur between two separate Pools and cannot connect two objects within a single Pool.



- Thus, Message Flow is only used in collaborations (diagrams with two or more Pools).
- Where a Pool has Process elements, the Message Flow connects to those elements
- Sequence Flow cannot cross a Pool boundary - i.e., a Process is fully contained within a Pool.

# 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*).



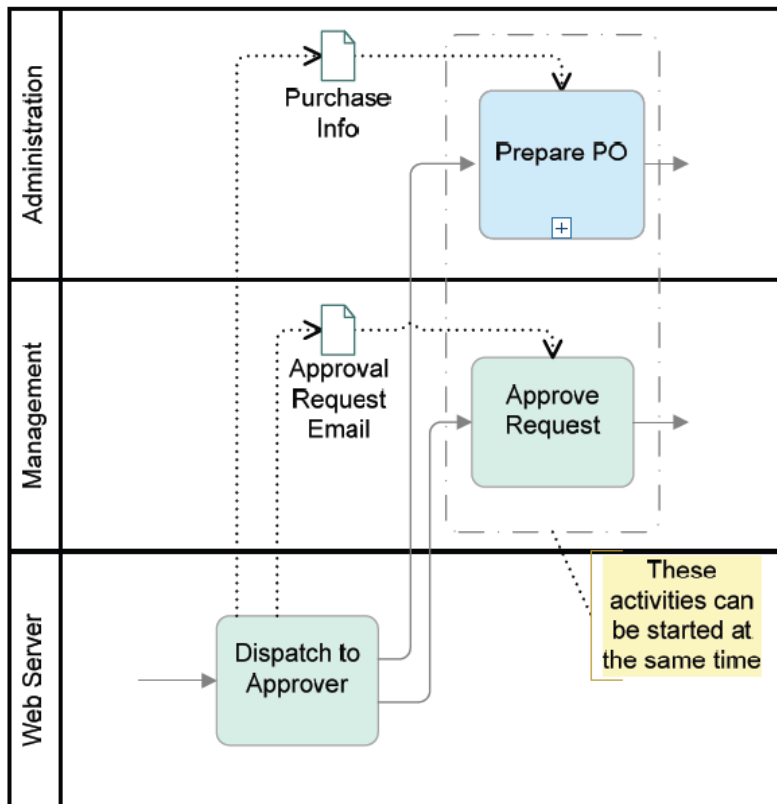
When the Pool is black box, Message Flow connects to its boundary.



# Lanes

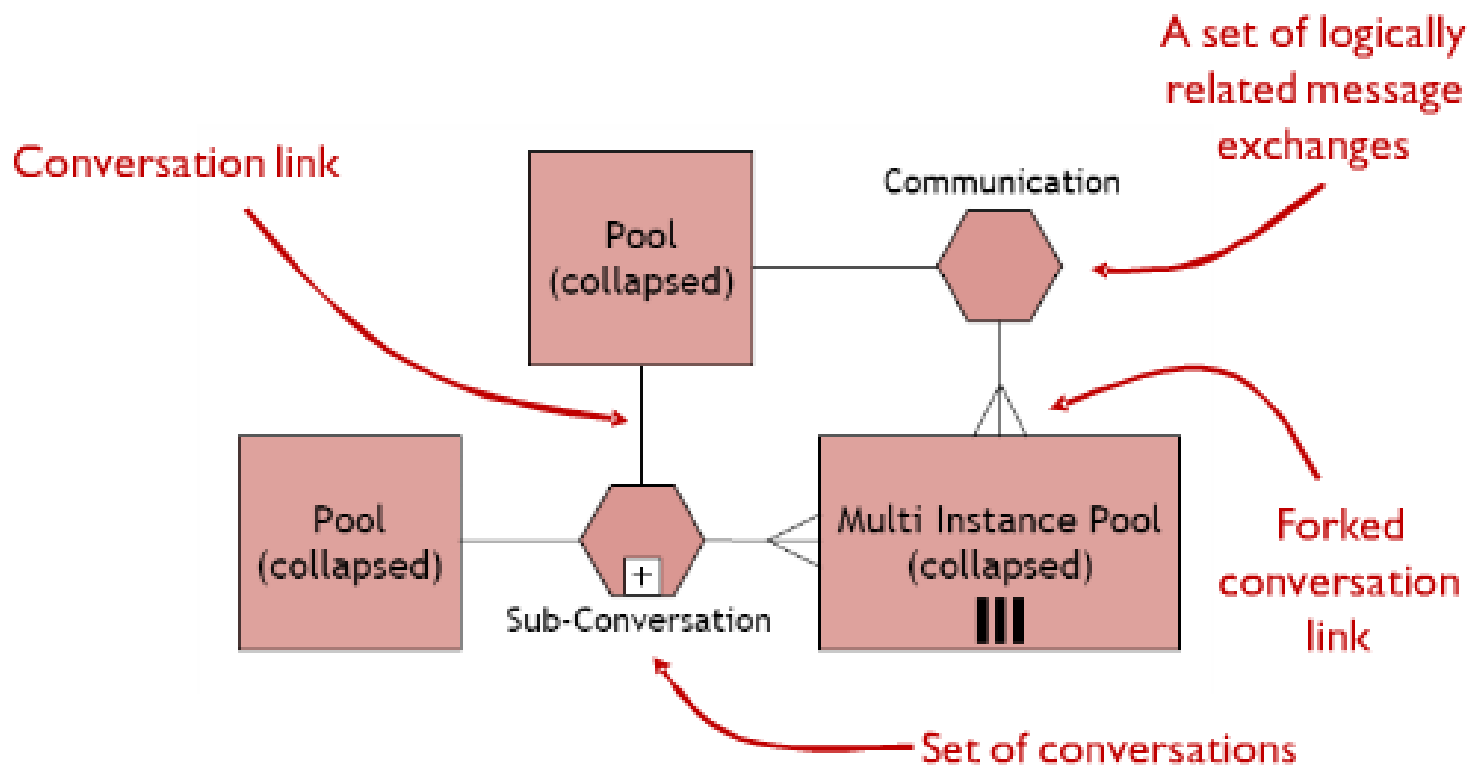
- Lanes create sub-partitions for the objects within a Pool.
- These partitions are used to group Process elements (showing how they are related), or which roles have responsibility for carrying out the Activities.

- 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.



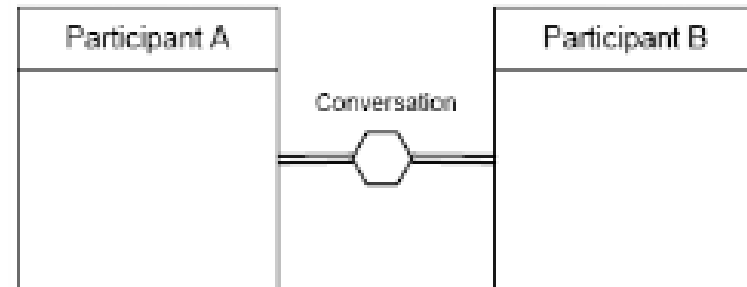
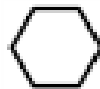
# Conversation Diagrams

- 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**.

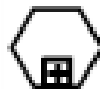


# Conversation Elements

- Communication or conversation defines a set of logically related message exchanges.



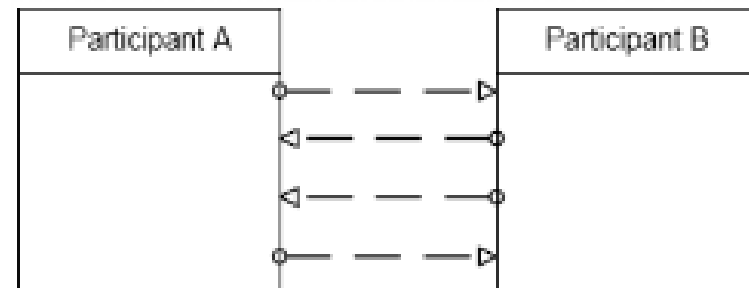
- When marked with a “+” it indicates a Sub-Conversation, a compound conversation element.



Is equal to:

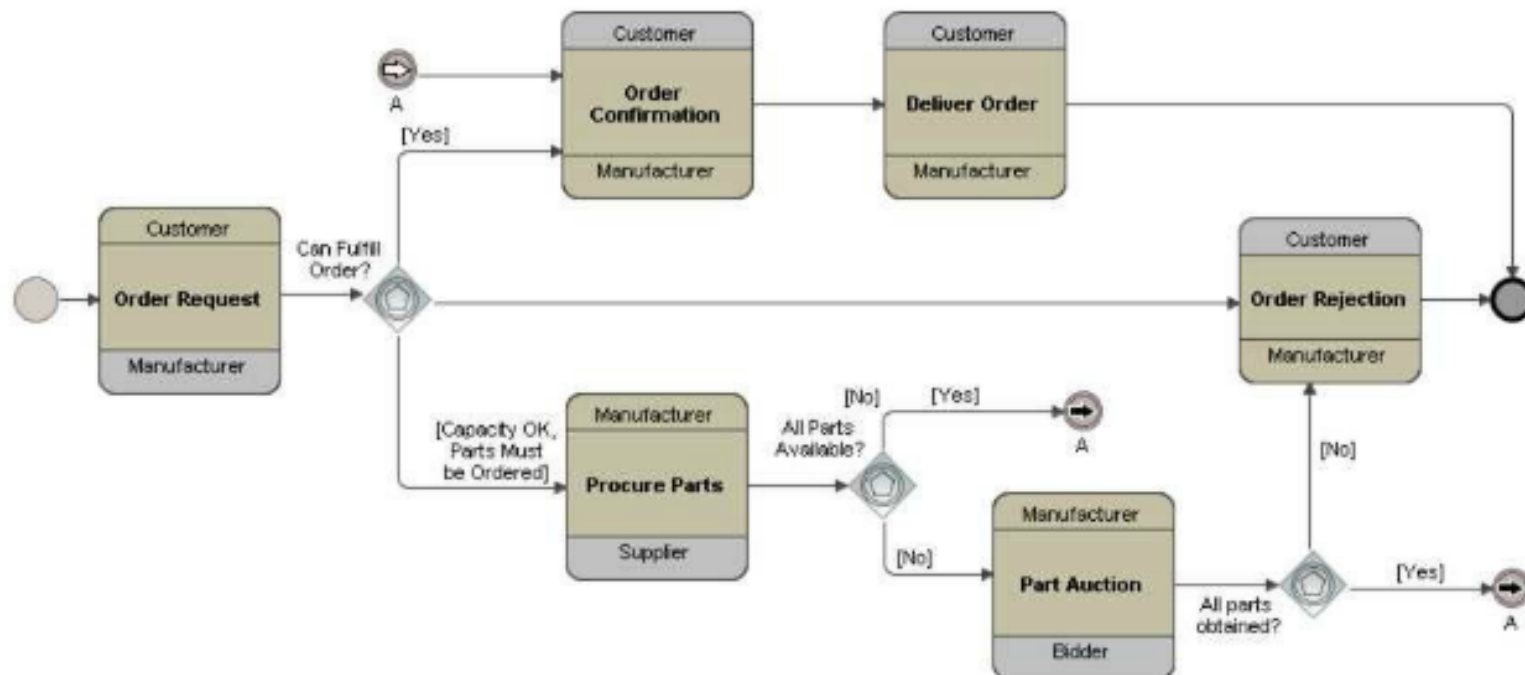


Is equal to:



# Coreography Diagram

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



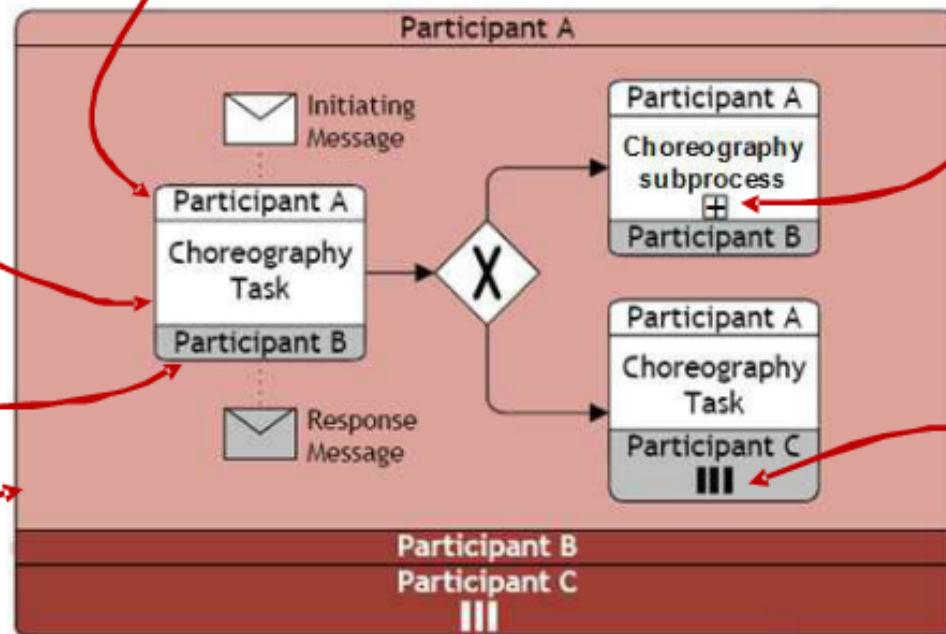
# Coreography Elements

**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



# Topics

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

## 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.

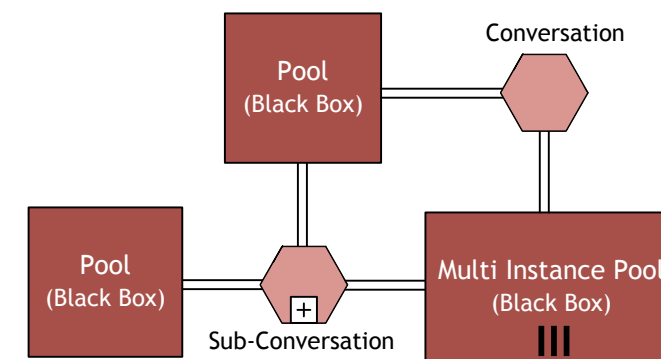
- Activity Markers**  
Markers indicate execution behavior of activities:
- Sub-Process Marker
  - Loop Marker
  - Parallel MI Marker
  - Sequential MI Marker
  - Ad Hoc Marker
  - Compensation Marker
- Task Types**  
Types specify the nature of the action to be performed:
- Send Task
  - Receive Task
  - 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.

## 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.

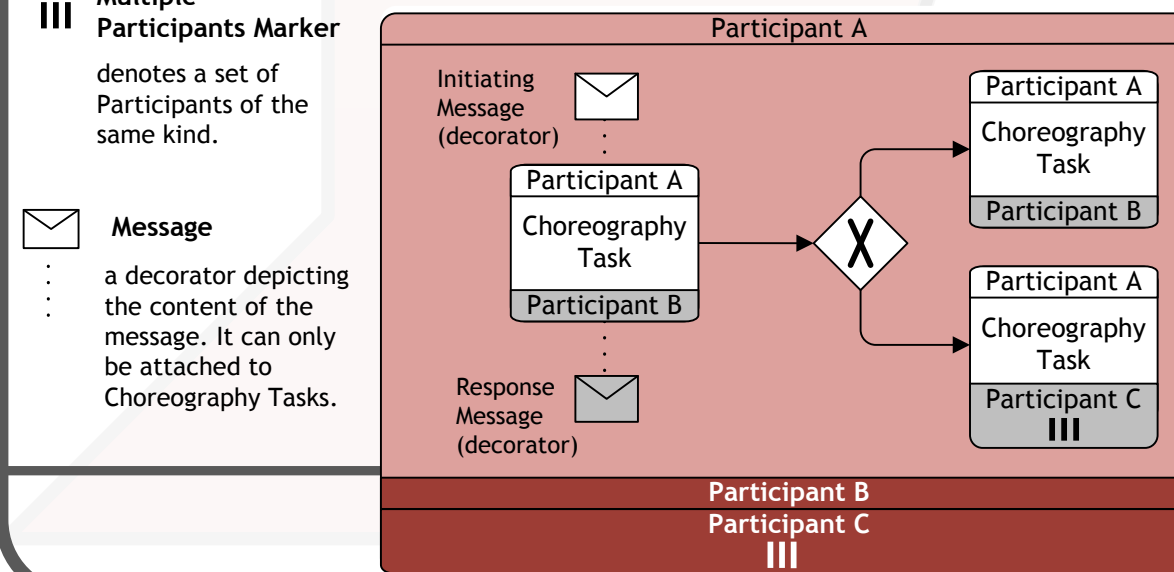
### Conversation Diagram



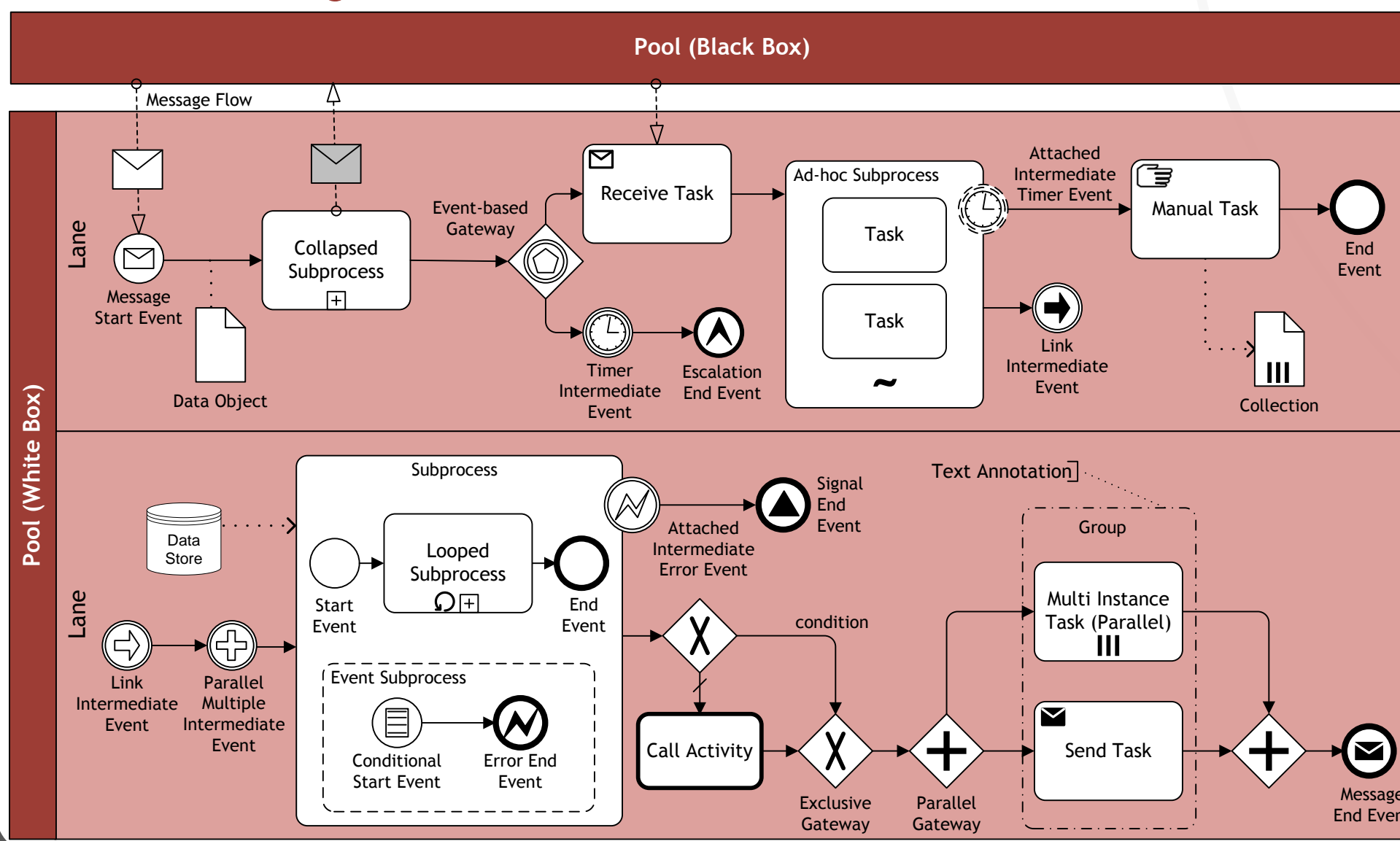
## Choreographies

- 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.

### Choreography Diagram



## Collaboration Diagram



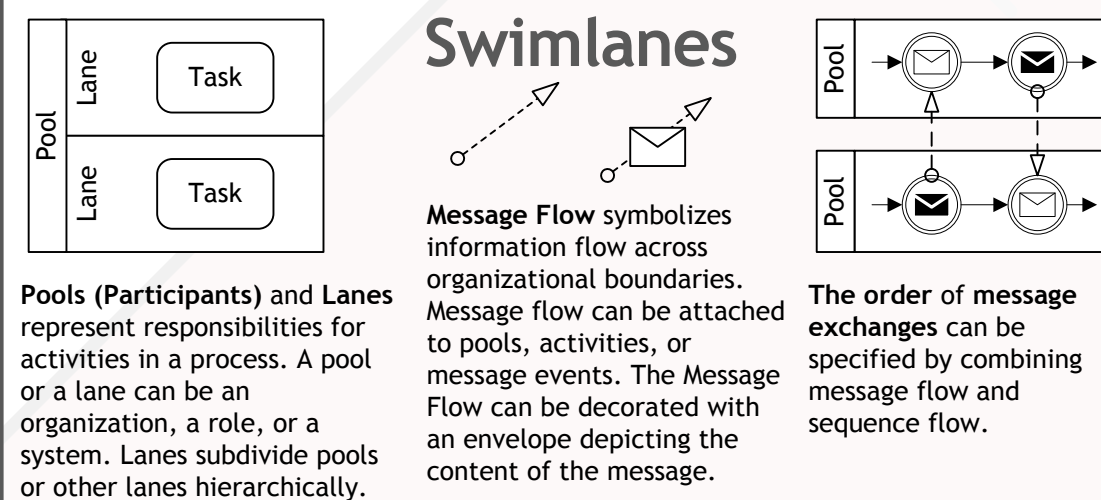
## Events

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

## Swimlanes





# 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

DIPARTIMENTO DI INFORMATICA  
E SISTEMISTICA ANTONIO RUBERTI



SAPIENZA  
UNIVERSITÀ DI ROMA