


# LAB-04

## BPMN

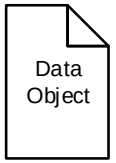
## Advanced Concepts

Lecturer: Andrea MARRELLA


# Objectives of this lecture

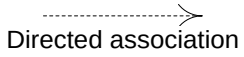
- Recap:
  - Data Artifacts in BPMN
  - Embedded and Independent Sub-processes
  - Multi-Instance, Loops, etc.
  - Error Events and non-interrupting boundary events
- Classroom exercises

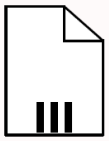
# Recap: Data Artifacts



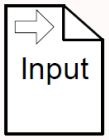

- **Data objects** are used to show how data and documents are used within a process as ***inputs*** and ***outputs*** of activities.


- **Data Stores** are ***containers of data objects*** that persist beyond the lifetime of the process instance.
- **Associations** are used to ***link*** data objects and data stores with flow objects.

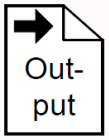


- A **Collection** of data objects represents a collection of information, e.g., a list of ordered items.



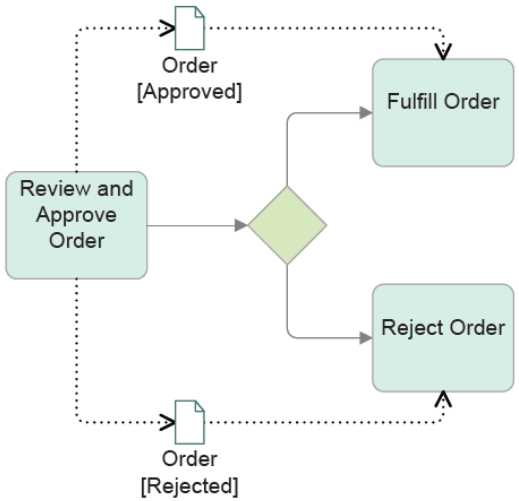

- A **Data Input** is an external input for the entire process. A kind of input parameter.




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

# Recap: Data Flow in BPMN

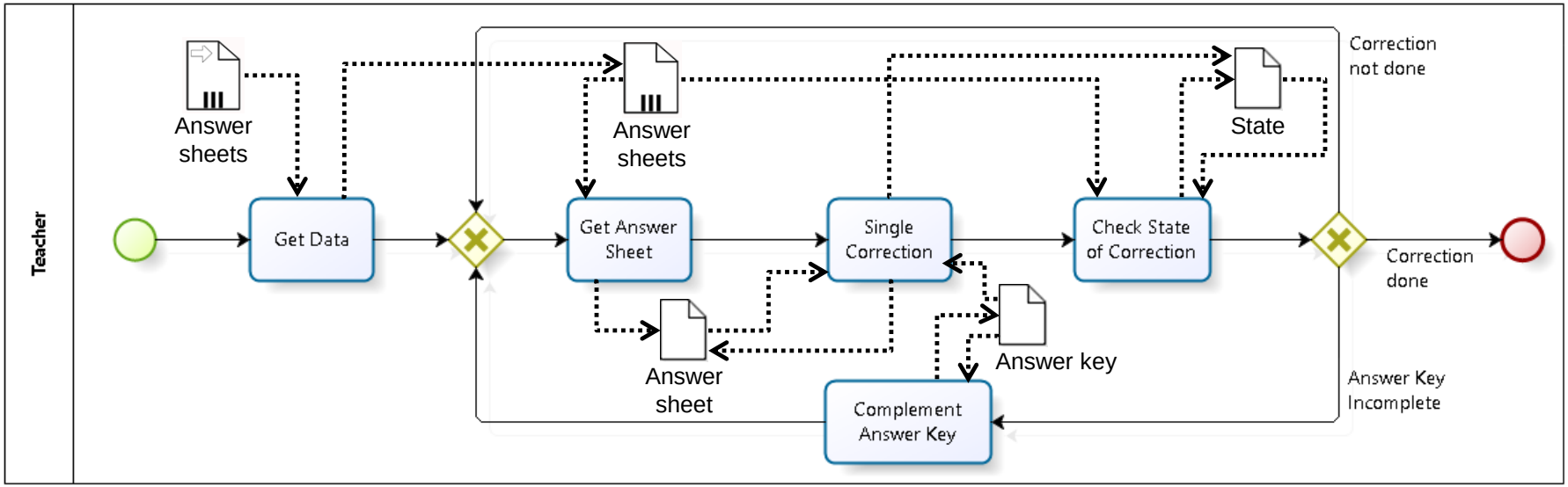
- Data objects may have "**states**" that depict how the object (document) is updated within the process.




- **Data flow** represents the movement of data from into and out of activities.
  - In BPMN, data flow is decoupled from the sequence flow.

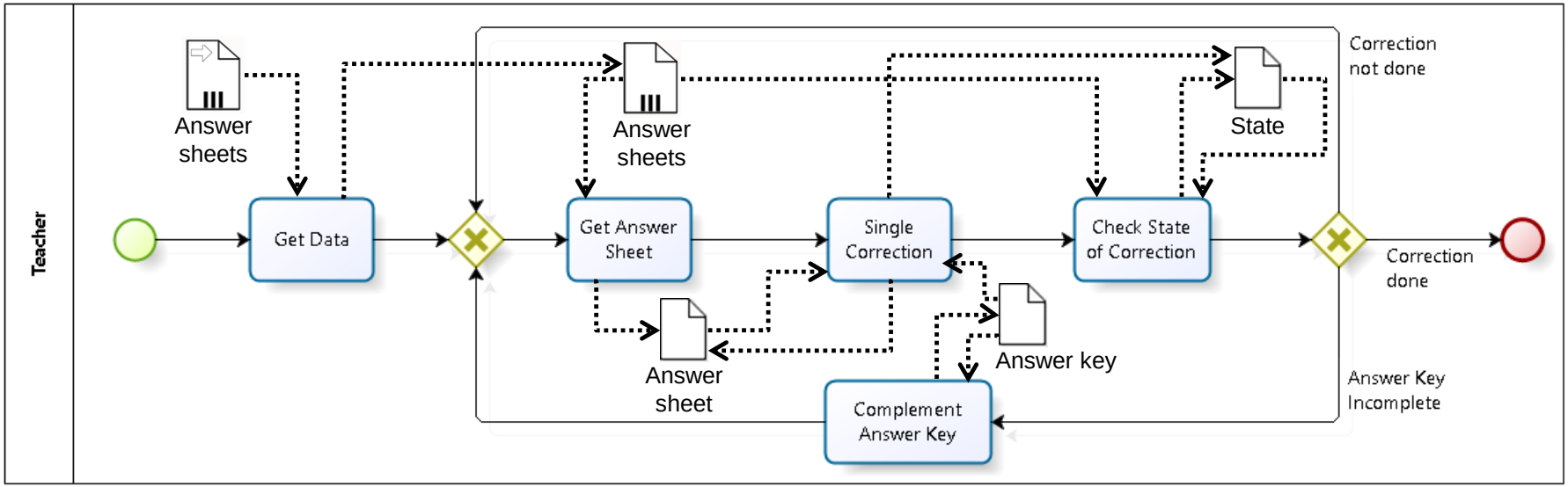
# Exam Correction Process

- At the beginning, the Data Input Collection ***Answer sheets*** of the process contains a set of answer sheets (i.e., filled out solutions of the exam). Task ***Get answer sheet*** selects one element of a local copy of this collection and writes it to the Data Object ***Answer sheet***.
- A performer of task ***Single Correction*** marks this ***Answer sheet*** using correction guidelines given in Data Object ***Answer key***.
- Task ***Check state of correction*** reviews whether all ***Answer sheets*** have been corrected or not. The task writes the status of the correction process in Data Object ***State***. Dependent on this value, the process continues the correction and turns back to the first gateway (condition ***Correction not done***), or it completes (condition ***Correction done***).
- The performer of task ***Single correction*** may identify a solution in an ***Answer sheet*** which is not part of the correction guidelines in Data Object ***Answer key*** up to now. In this case, ***Single correction*** writes the status Answer key incomplete to the Data Object ***State***, and the process runs task ***Complement answer key*** later on to update the answer keys.



# Errors and Warnings

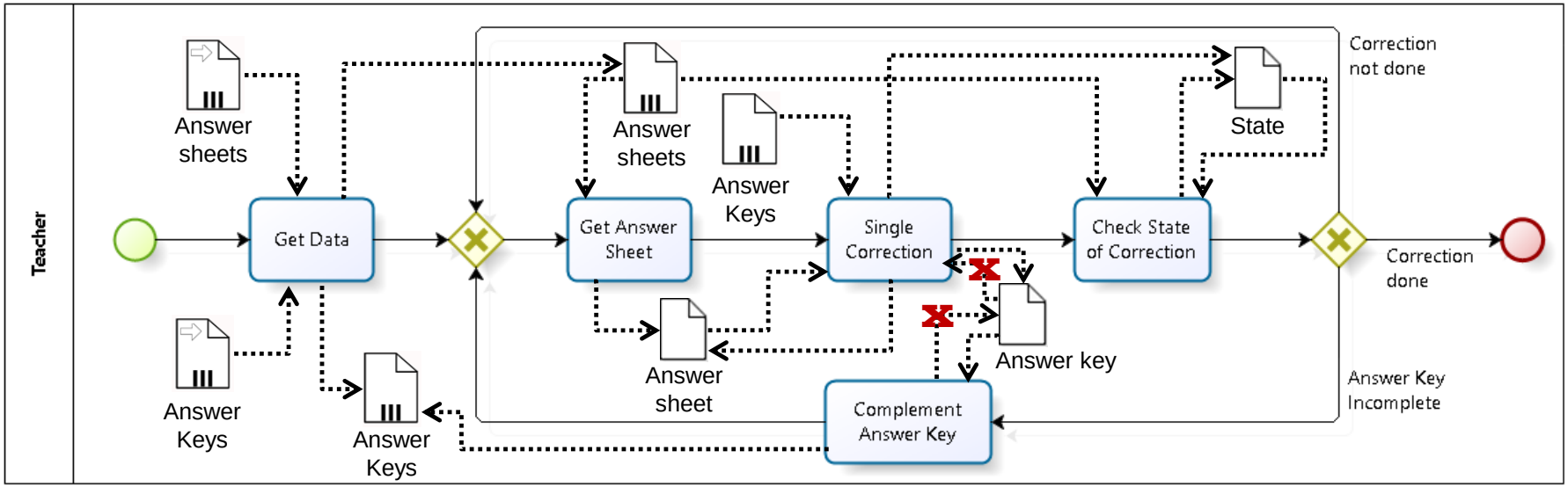
**WARNING**: Task **Single Correction** initializes ***State*** only optionally.




**WARNING**: There is no task using ***Answer sheet*** that has been updated by the task **Single Correction**.

**ERROR**: ***Answer key*** is **uninitialized** in the first run of the task **Single Correction**.

# Fixing Errors in the Data Flow



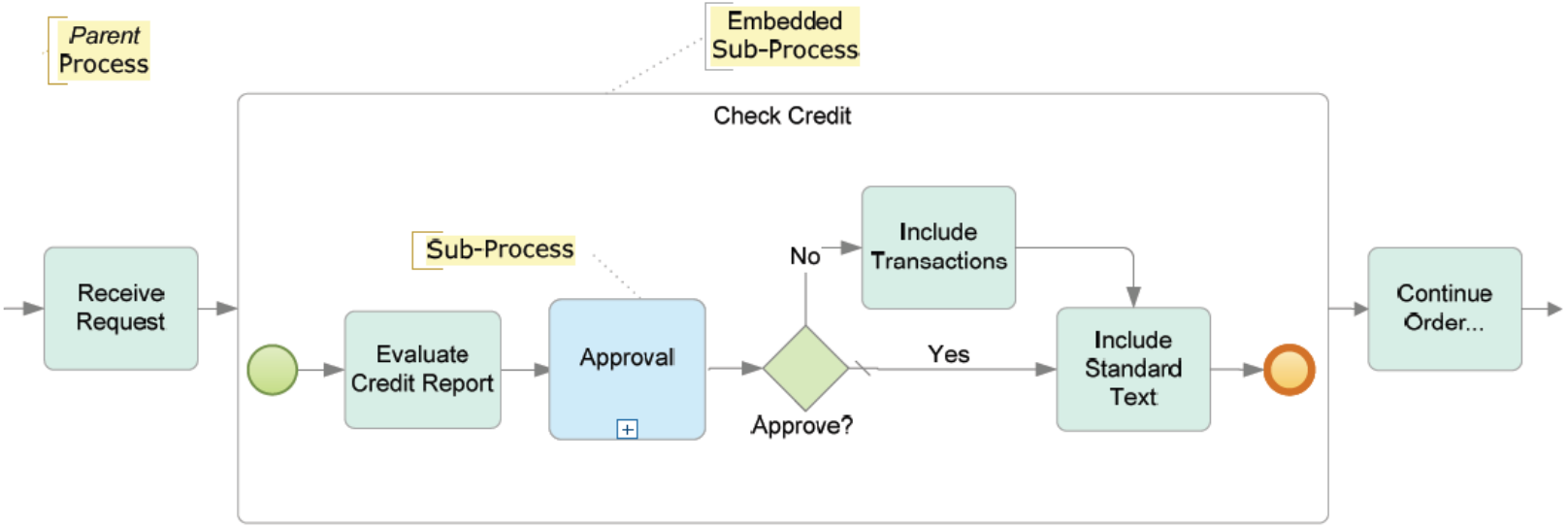

**The Simulation tool of Bizagi does not capture properly the behaviour of data flow.**

# Recap: Sub-processes

- 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***.
- Sub-processes enable hierarchical process development. There are two types of Sub-Processes: ***Embedded*** and ***Independent***.
  - By default, a sub-process is "**embedded**" into its parent process (i.e. it is stored within the same file).
  - In order to **maximize reuse**, it is possible to "extract" the sub-process and store it as a separate file in the process model repository. Such a sub-process is called "**independent**", and is invoked via a "**call**" activity

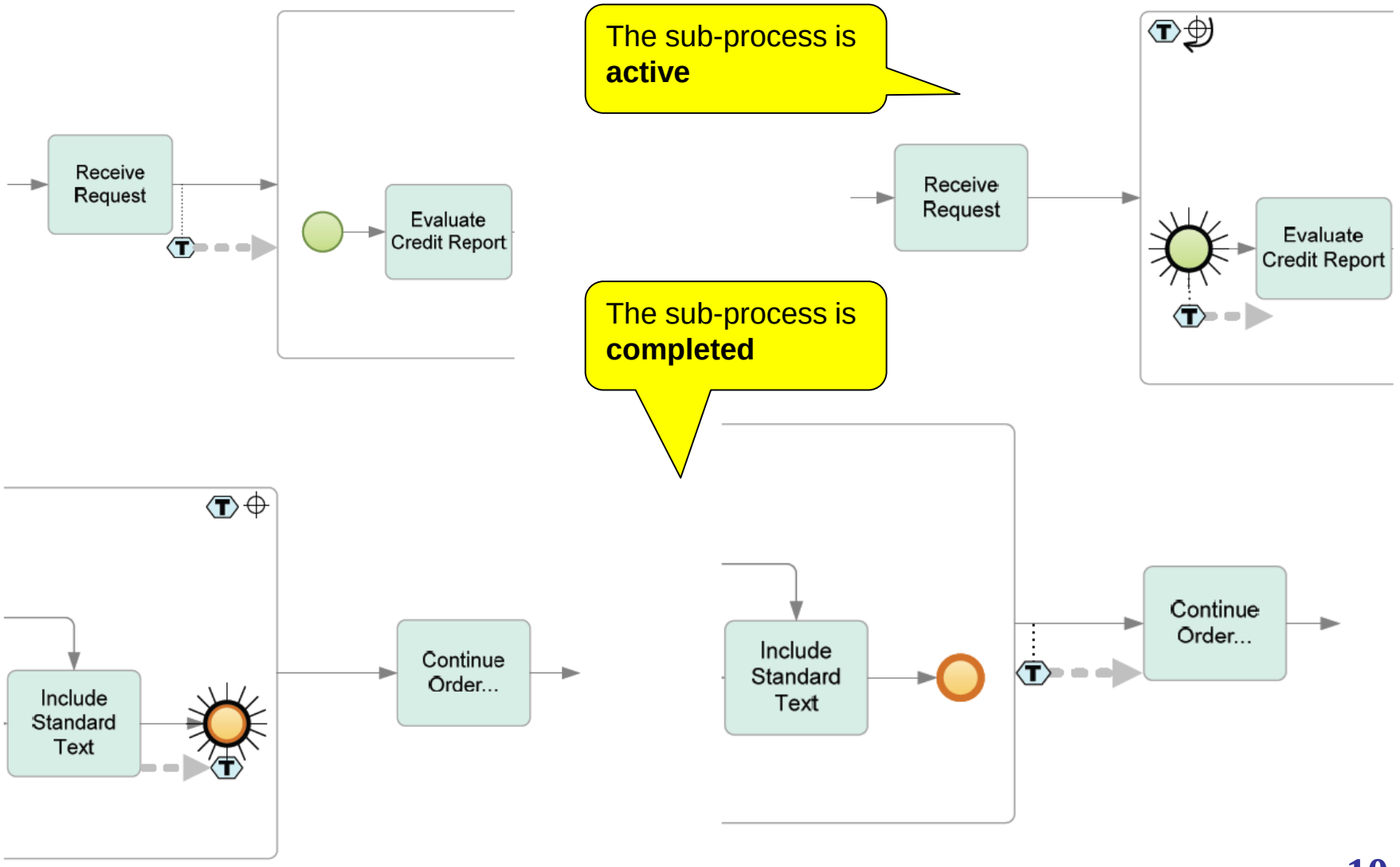
# Recap: Embedded Sub-processes

- **Embedded Sub-Processes** are ***part of a parent process*** and ***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).



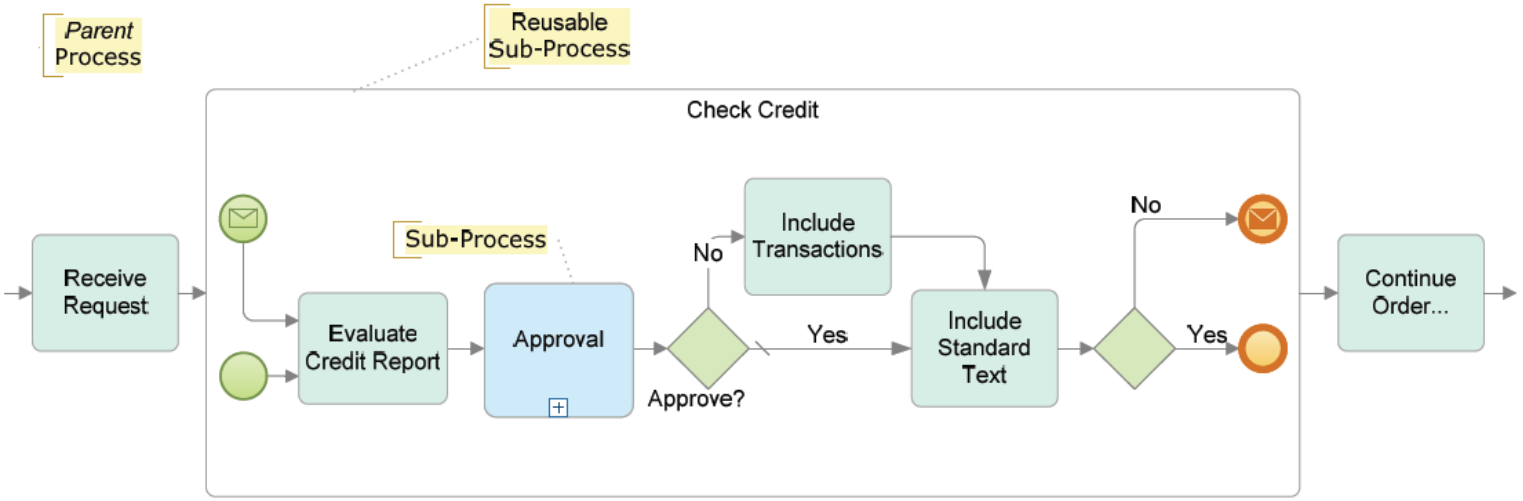

- An important characteristic of an embedded sub-process is that **it can only begin with a None Start Event**.

# Recap:
# Behaviour across process levels

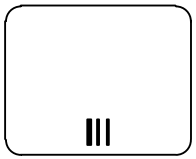


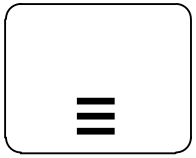
# Recap: Independent Sub-processes

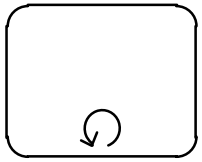
- An ***Independent Sub-Process*** is a separately modeled process that could be used in multiple contexts (e.g., *checking the credit of a customer*).
- The "process relevant data" of the parent (calling) process is not automatically available to the sub-process.
  - ***Any data must be transferred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

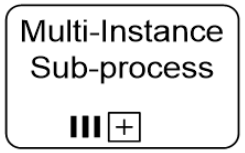


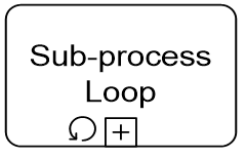

- Just like an embedded sub-process, an independent sub-process **must have at least None Start Event**.
  - Independent sub-processes **maximize reuse**.









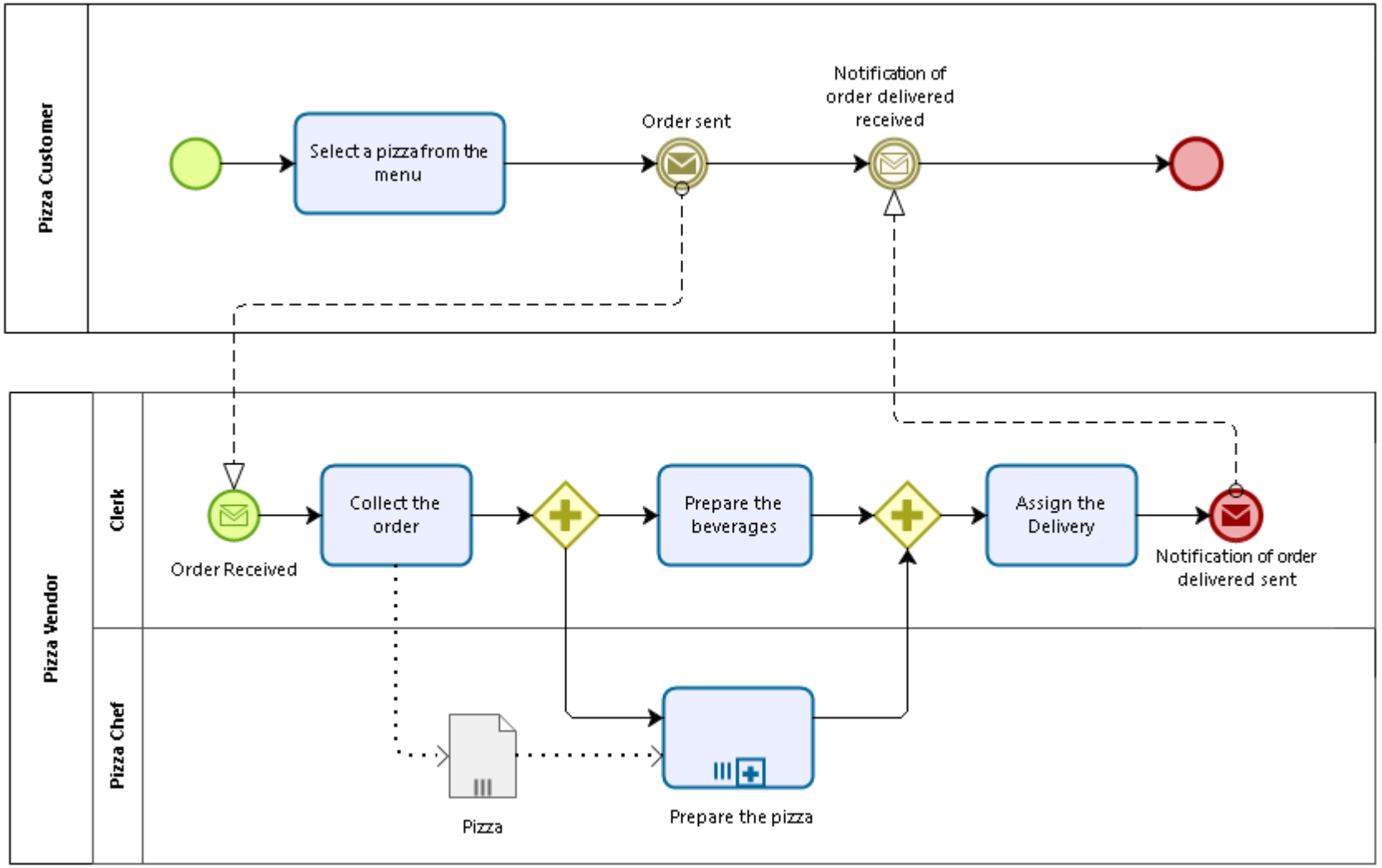

# Recap: Loop and Multiple Activities

- **Multi-Instance Activities**:
  - Activities to be performed **many times** concurrently with **different data sets**.
  - The individual instances of a multi-instance activity might occur in ***parallel*** or in ***sequence***.
  - The key point to understand that the activity does not cycle around; **each activity execution is distinct from the others**.
- **Loop Activities:**
  - On an activity, it is possible to define a **loop condition** that determines the number of times to perform that activity.
- They can be used both for labeling tasks and compound activities.

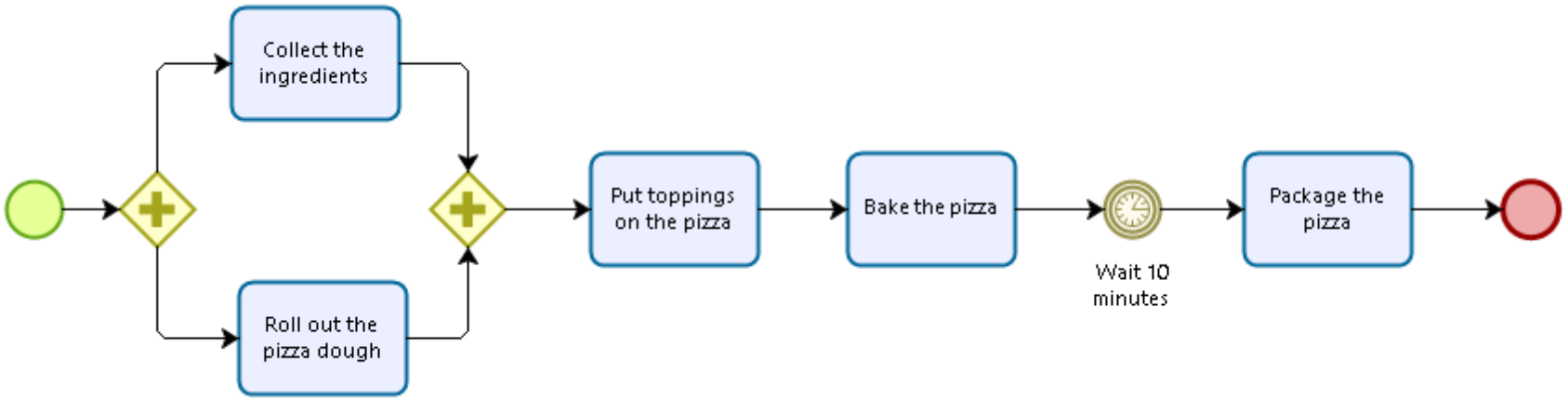

# Exercise – Pizza Delivery

- The exercise consists in modeling the interaction between a pizza customer and a pizza vendor.
- The customer selects some pizzas and beverages from the menu of the pizza vendor and orders them. After that, the customer waits for the notification that the pizza has been properly assigned to a “delivery boy”.
- The process of the pizza vendor is triggered by the order of the customer. After a clerk has collected the order, the preparation of the pizzas is delegated to the pizza chef. In the meanwhile the clerk starts to retrieve the beverages.
- The pizza chef follows a specific process to prepare each pizza associated to a specific order. Specifically, for each pizza, he prepares the ingredients and rolls out the pizza dough, puts the toppings on top of the pizza and bakes it. After 10 minutes, he packages the pizza and provides it to the clerk.
- When the clerk has collected all the beverages and the packages containing the pizzas of a specific order, he assigns them to a delivery boy and notifies the customer that the pizzas will be delivered soon.

# Solution 1/2
# Pizza Delivery Process



# Solution 2/2
# Subprocess «Prepare the pizza»



# Recap: Error Events

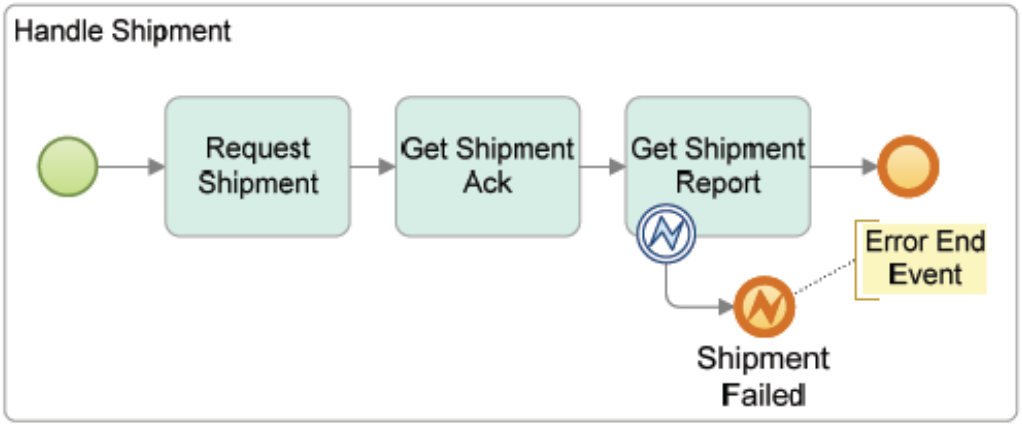
Error End Event – Catching

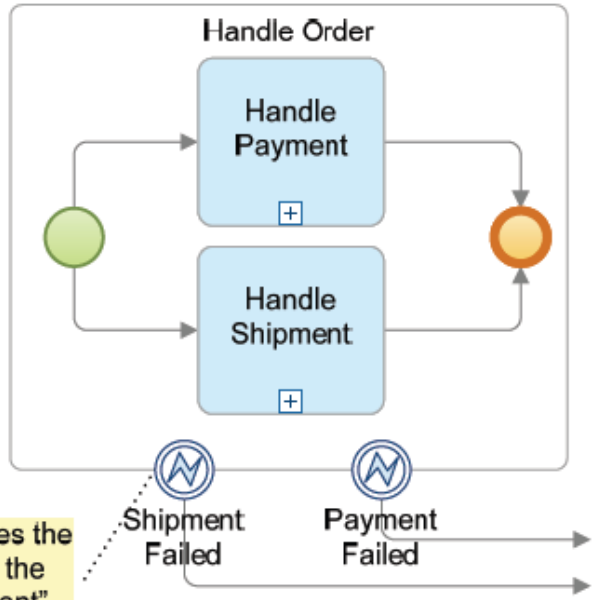
Error Intermediate Event - Throwing

- 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***.

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

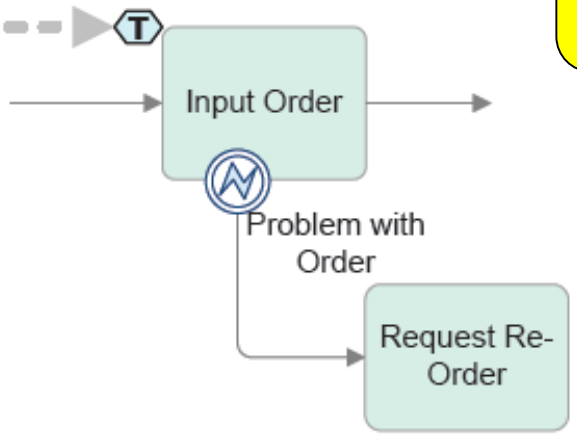
**An error can only be seen by a parent process**. Other processes at the same level or within different Pools cannot see the error.

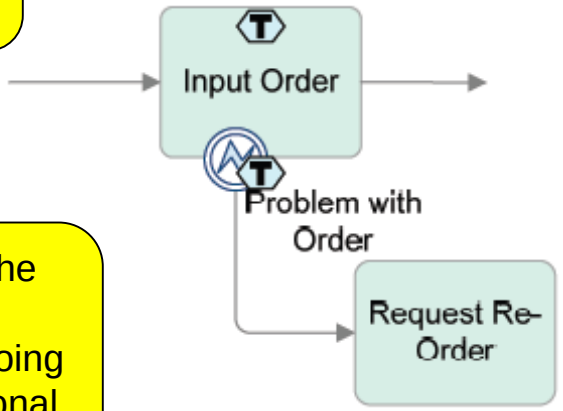


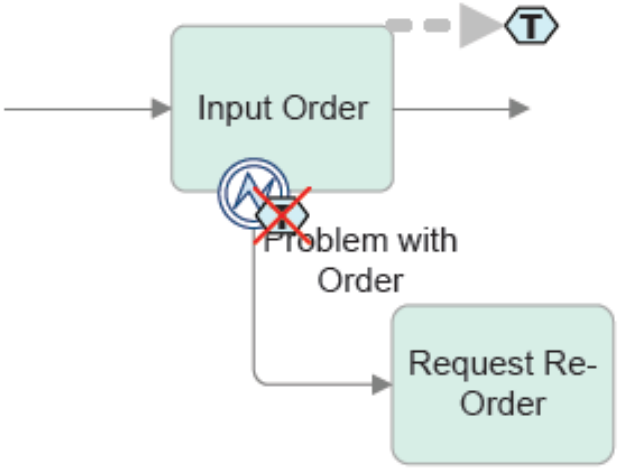
# Recap: Error Events Behaviour

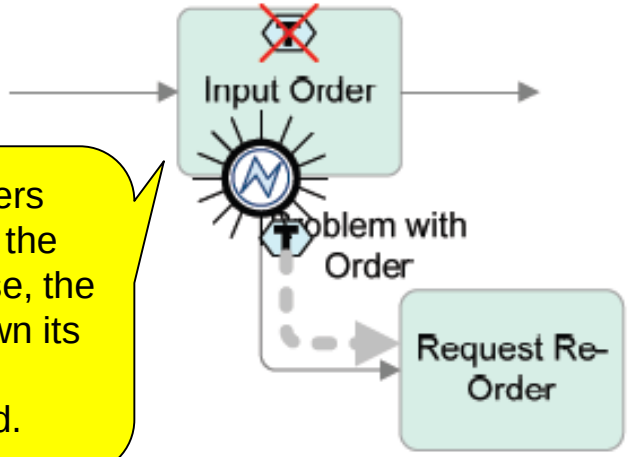
When a token arrives at the activity with the attached Intermediate Error Event, **another token is created** and **placed** in the Error Event on its boundary.




If the activity finishes before the trigger occurs, then the token moves down the normal outgoing sequence flow and the additional token is consumed.



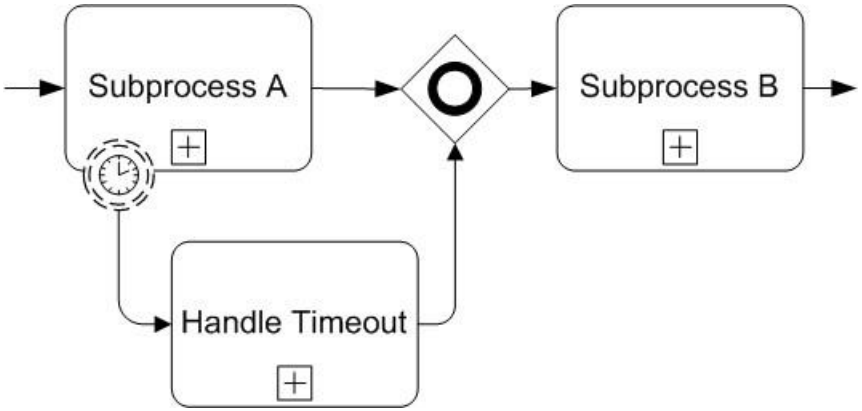




If the attached Error Event triggers before the activity finishes, then the activity is interrupted. In this case, the token from the event moves down its outgoing sequence flow and the token in the activity is consumed.

# Recap: Non-interrupting events

- If we need to trigger an activity **in parallel** to the normal flow, i.e. without interrupting the normal flow, a ***non-interrupting*** boundary event should be used.

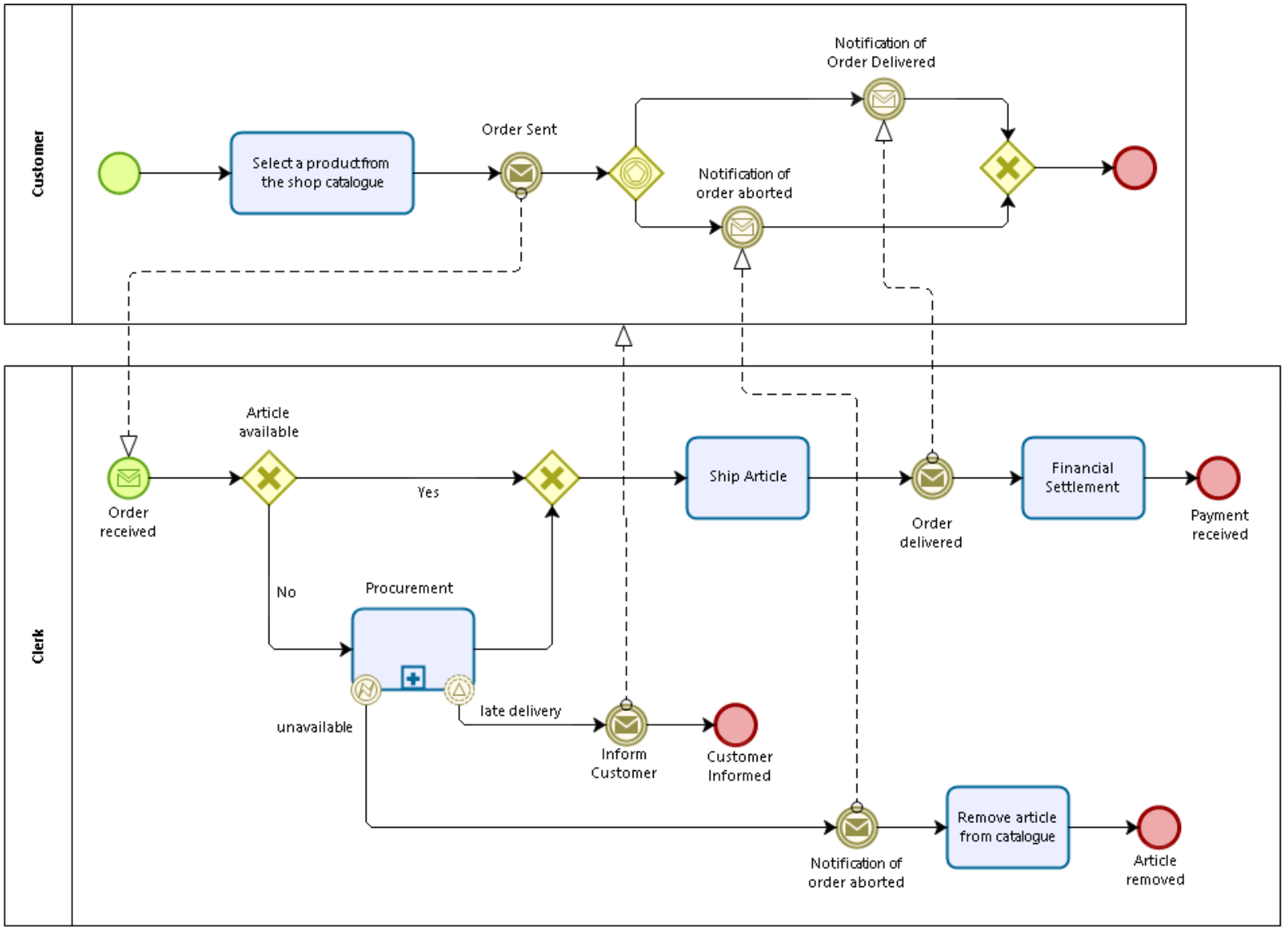
- In the case of a Non-Interrupting Event, the activity that was being performed will continue in parallel along with the new flow that was initiated by the boundary event.
  - **The current activity will NOT be cancelled or interrupted**.




# Exercise - Order Fulfillment and Procurement

- The order fulfillment and procurement process starts when a customer selects an article from the catalogue of a shop and performs an order.
- The process of the shop starts after receiving an order message and a clerk checks if the ordered article is available or not. An available article is shipped to the customer (that is also notified via email) followed by a financial settlement.
- In case that an article is not available, it has to be procured by calling the procurement sub-process. Such sub-process can be stopped in any moment during its execution if an article is discovered to be undeliverable. If this is the case, the customer is notified about the unavailability of the article, which is then removed by the catalogue of the shop. In this case, finally the execution of the Procurement sub-process stops immediately.
- The first task in the Procurement sub-process is to check whether the article to procured is available at the supplier. If not, a “unavailability”-exception is thrown and the sub-process ends. In case that the delivery lasts more than 2 days, a signal event is thrown by the sub-process telling the referencing top-level-process that the delivery will be late. In this last case, and if the delivery will last less than 2 days, the order is performed at the supplier.
- At this point, the Procurement sub-process continues its execution by waiting for the delivery. When the procurement sub-process finishes, the Order Fulfillment process continues with the shipment of the article and the financial settlement.

# Solution 1/2
# Order Fulfillment and Procurement



# Solution 2/2
# Subprocess «Procurement»