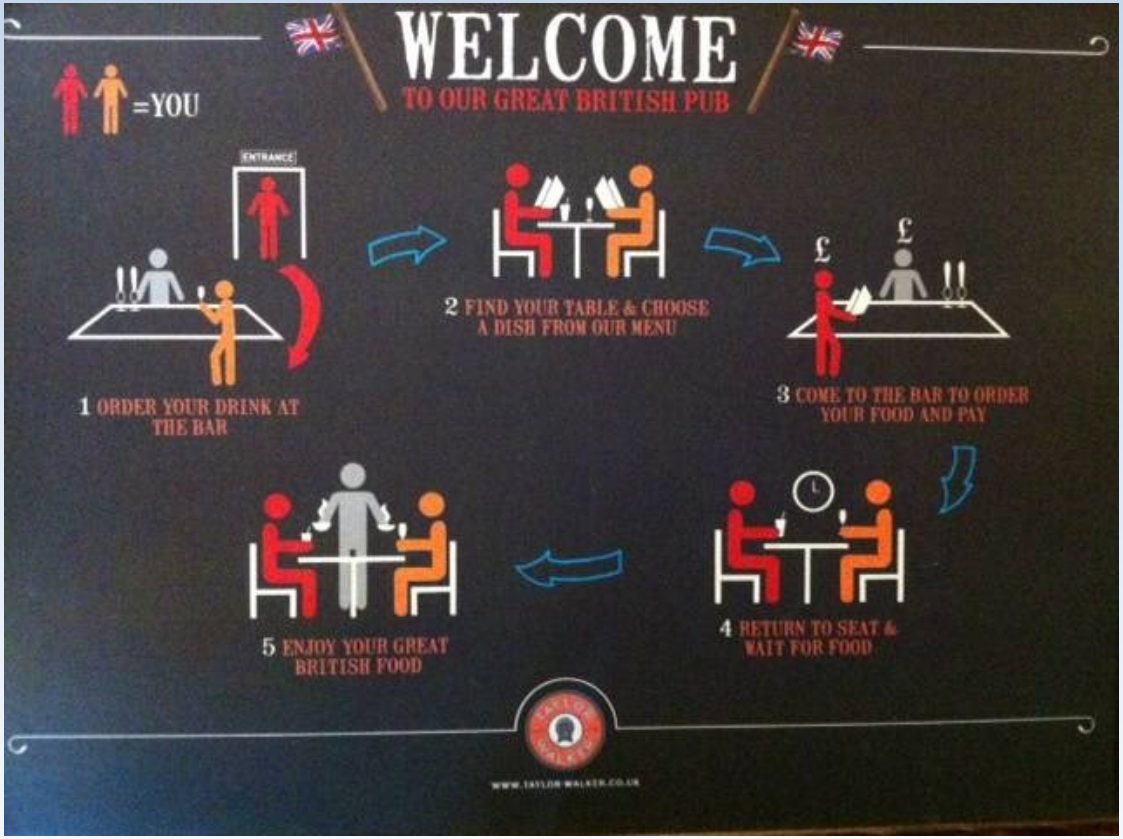
BPMN Diagrams

http://www.bpmn.org/



"Classic BPMN, British Pub Menu Notation"…

https://www.storm-consulting.com/media/BPMN-British-Pub-version.jpg

…and the expectations concerning the process outcome…

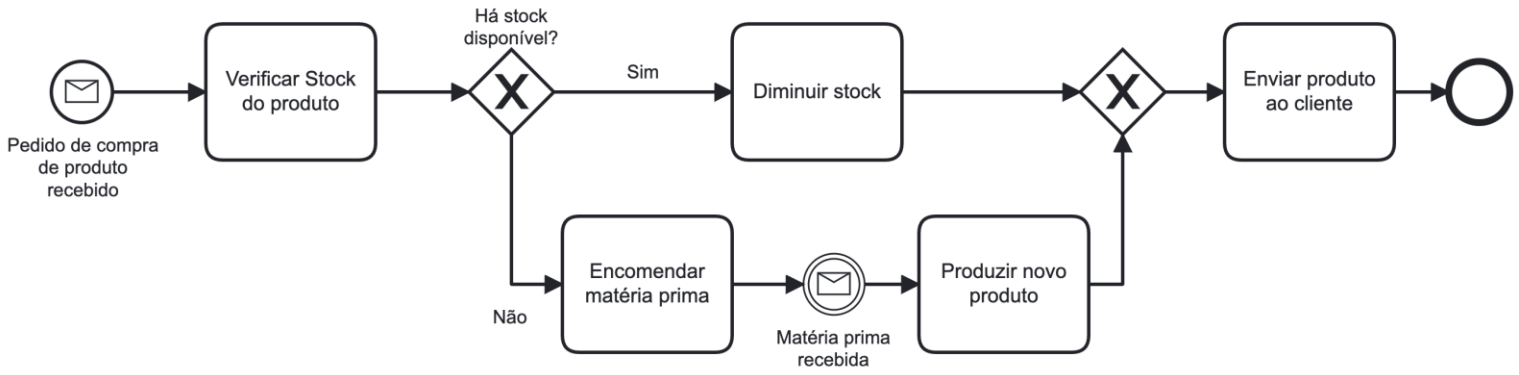
https://www.coolpun.com/images/coolpun/4f/4f5f99404d3f38fcada6e50c45f67edd.jpeg

Hello World! BPMN

Quando um vendedor recebe um pedido de compra de produto verifica sempre o stock disponível.

Se houver stock disponível, atualiza-o e envia o produto ao cliente.

Se não houver stock disponível, encomenda matéria-prima e espera pela sua receção. De seguida, produz o novo produto e envia-o ao cliente.

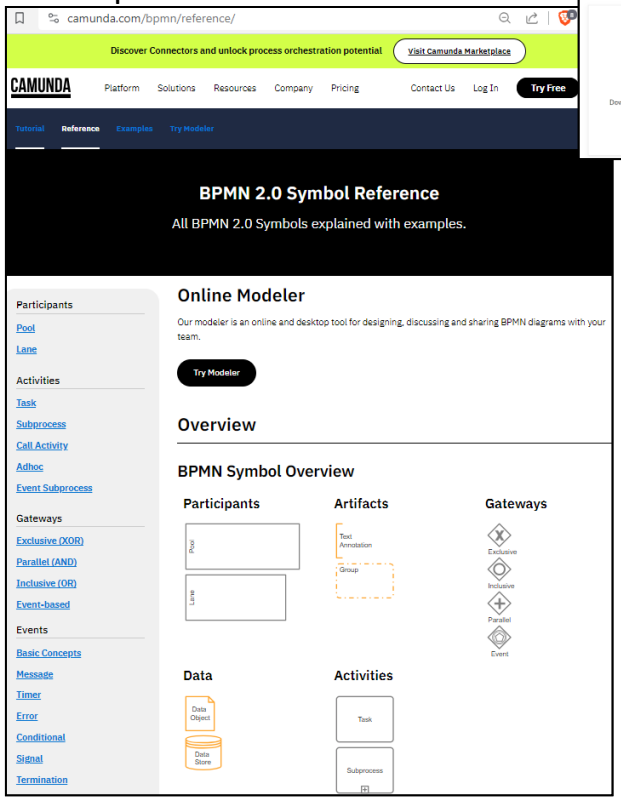


“Business” Process Model and Notation (BPMN)

http://www.omg.org/bpmn/

- BPMN is a OMG notation for Business Process Modelling.
- BPMN allows the graphical specification of business processes.
- BPMN can be mapped to execution languages and SOA environments…

- Timeline:
  - 2004: BPMI (Business Process Management Initiative) publishes BPMN 1.0
  - 2005: BPMI is integrated into OMG (Object Management Group)
  - 2006: OMG officially adopted the notation and publishes BPMN 1.0
  - 2008: BPMN 1.2 published
  - 2009: BPMN 2.0 draft release
  - 2011: BPMN 2.0 published
  - 2014: BPMN 2.0.2 published



…all about BPMN:

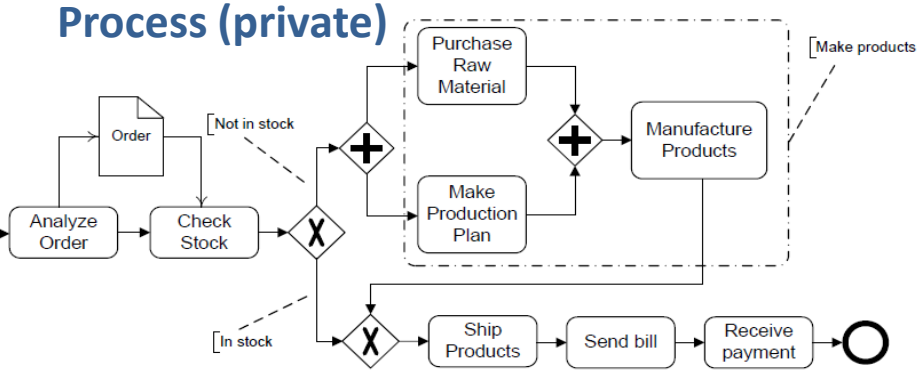
- http://www.bpmn.org/
- https://www.bpmn.org/
- https://camunda.com/bpmn/reference/
- …

Types of BPMN Diagrams

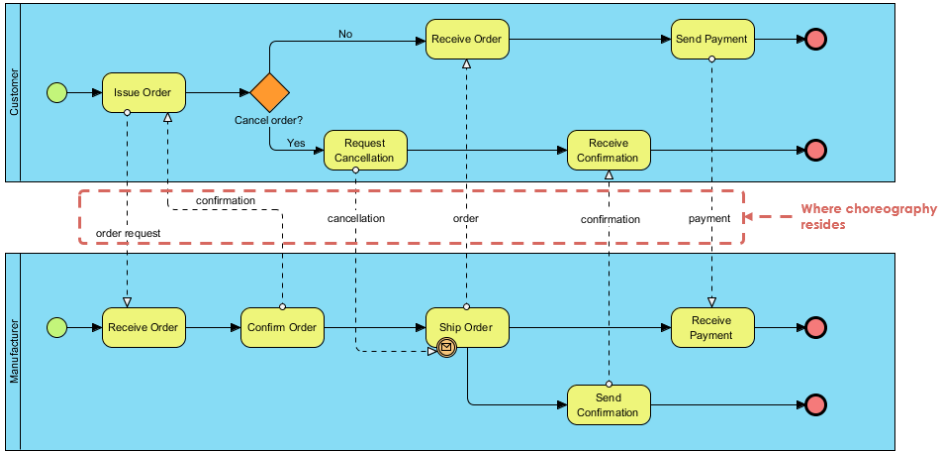
- **Process**
  - Represents the public or private processes of a participant.
  - Focus on representing the (internal) orchestration of a process.
  - The participant can be subdivided into multiple Lanes.
  - All external pools (if any) must be black-box.
- **Collaboration**
  - Represents the message exchange between two or more participants .
  - Focus on representing the orchestration of a process across multiple participants.
- **Choreography not addressed in "AMS")**
  - Represent the information pertaining to each participant in the choreography.
  - The focus is not on orchestrations of the work performed within these Participants, but rather on the exchange of information between Participants.
- **Conversation (not addressed in "AMS")**
  - Conversation diagrams visualize messages exchange between pools.
  - Design workflow with business process diagram and visualize communications with BPMN conversation diagrams.

Types of BPMN Diagrams

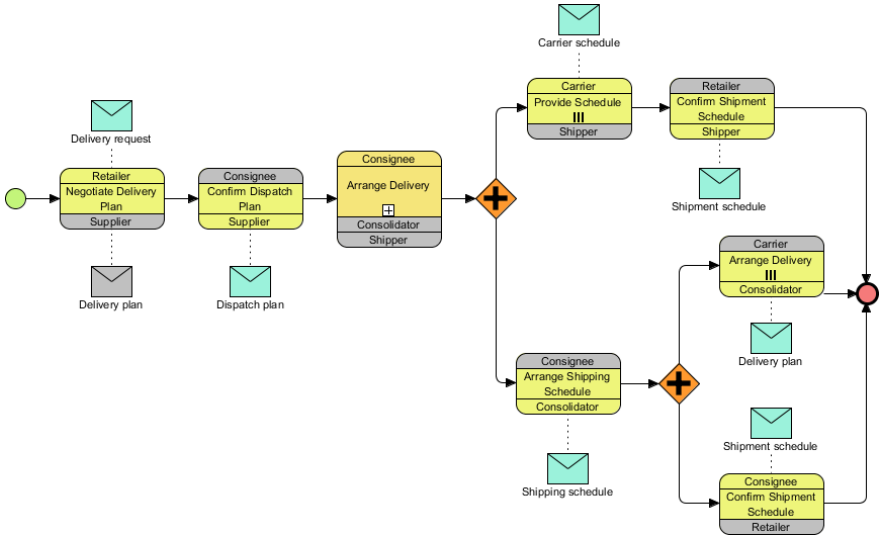
Process (private)



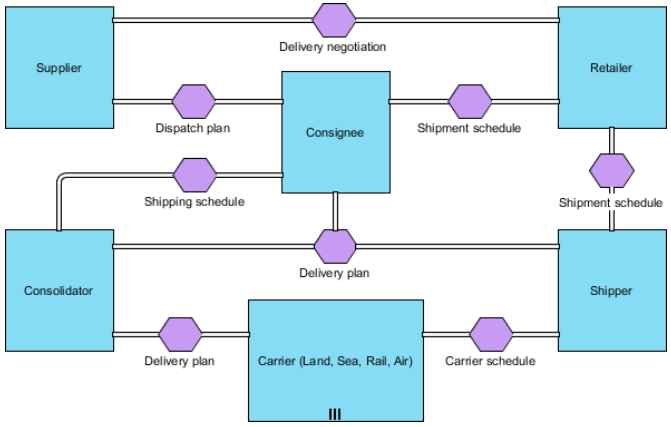
Collaboration



Choreography (not addressed in "AMS")



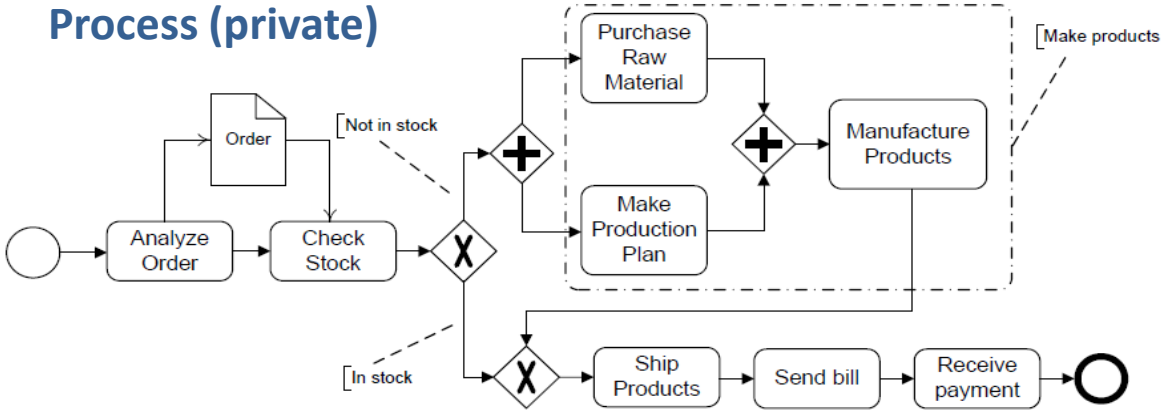
Conversation (not addressed in "AMS")



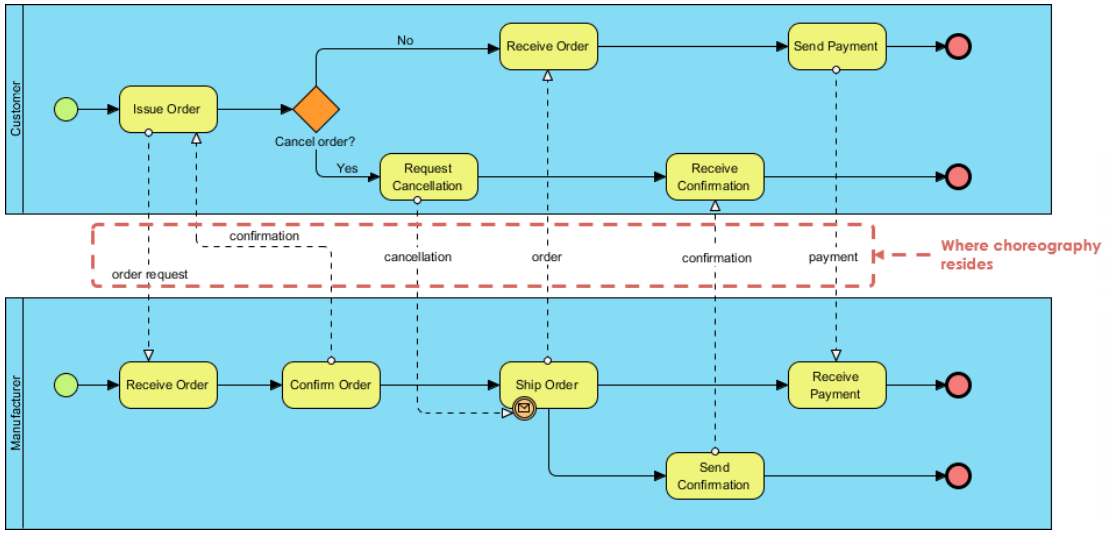
Examples of Collaboration, Choreography and Conversation from:
https://www.visual-paradigm.com/guide/bpmn/bpmn-orchestration-vs-choreography-vs-collaboration/

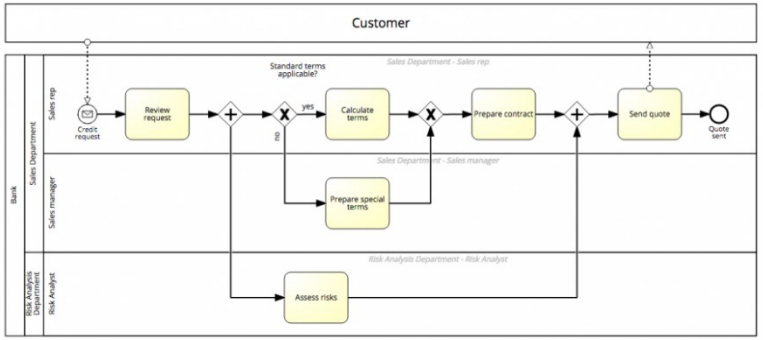
Types of BPMN Diagrams we will address

Process (private)



Collaboration





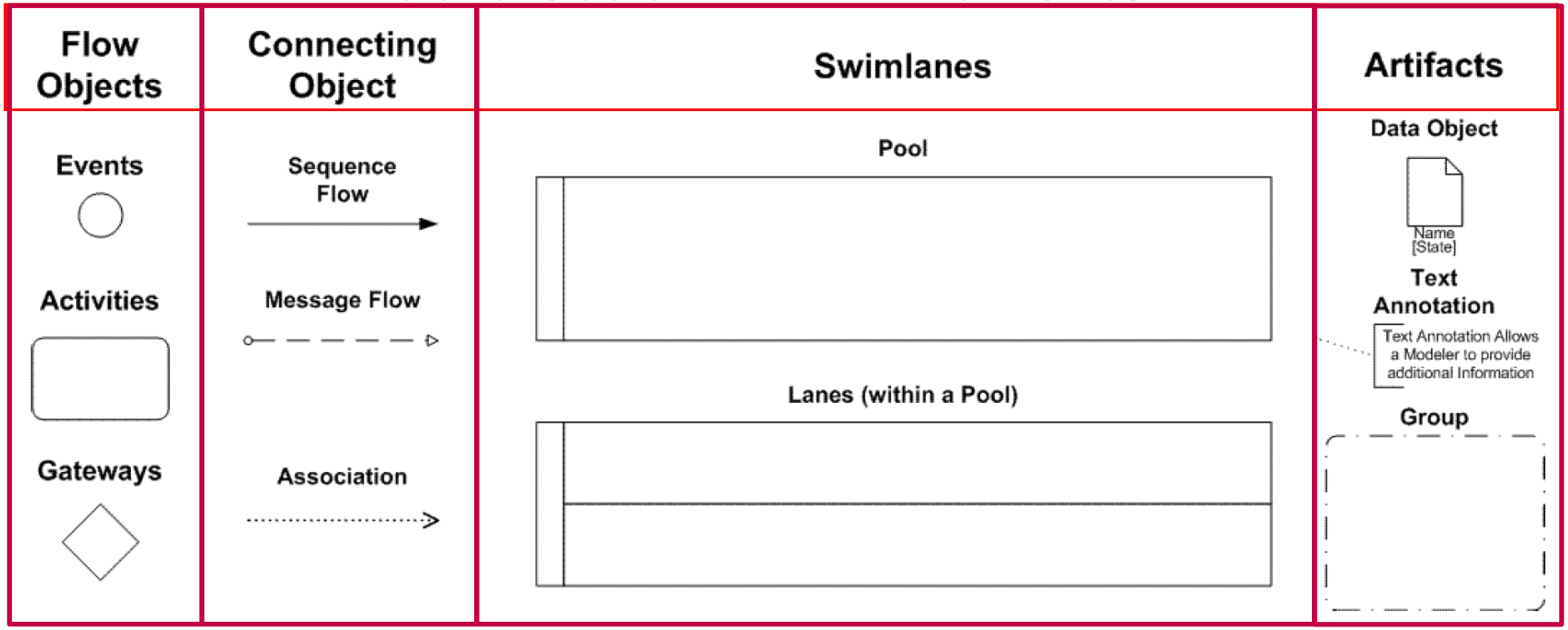
https://www.visual-paradigm.com/guide/bpmn/bpmn-orchestration-vs-choreography-vs-collaboration/

BPMN Core Elements

Core BPMN Elements

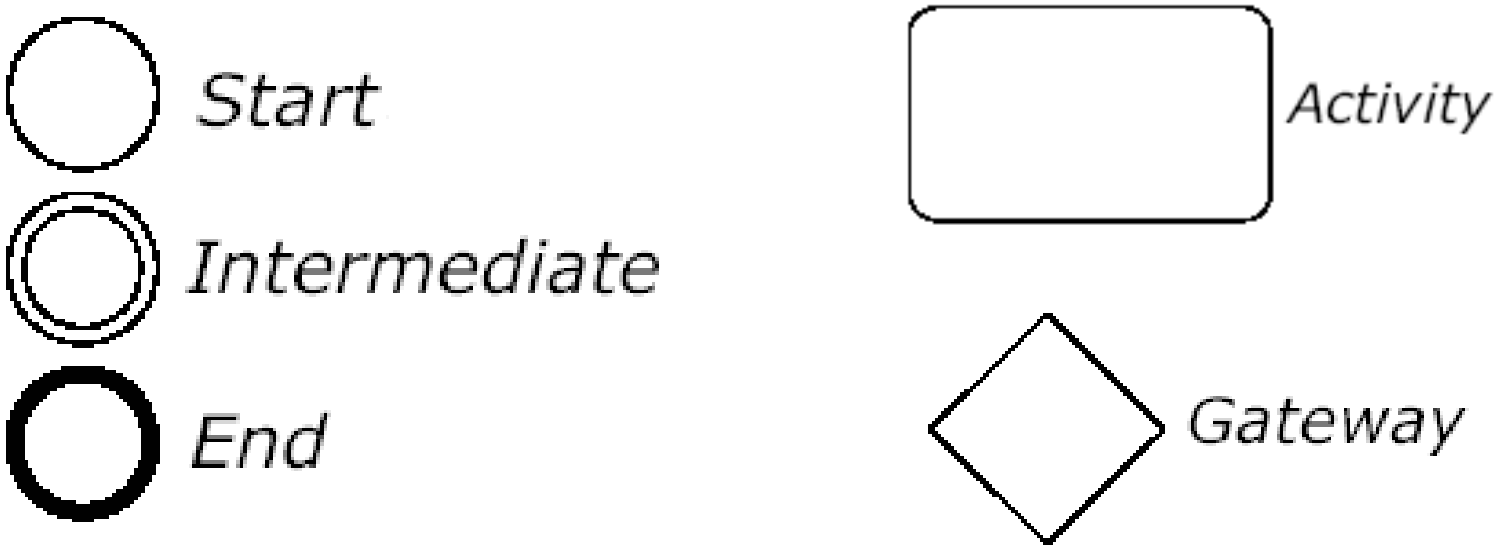
1. Flow	Event	Activity	Gateway
2. Connectors	Sequence	Message	Association
3. Data	Input	Output	Data Object
4. Swimlanes	Pool	Lane	
5. Artefacts			

Core Set of BPMN Elements



Flow Objects

- **Activity**: a unit of work.
- **Event**: an occurrence during a business process.
- **Gateway**: controls the flow of activities.



Connectors

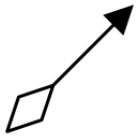
Sequence Flow

defines the execution order of activities.

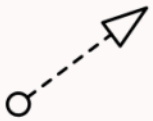
Default Flow

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

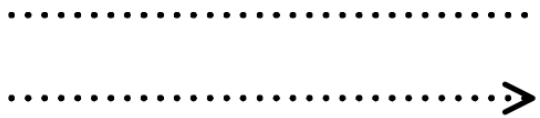
Conditional Flow



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



Message Flow
symbolizes information flow across organizational boundaries. Message flow can be attached to pools, activities, or message events.



An Association is used to link information and Artifacts with BPMN graphical elements (see page 67). Text Annotations (see page 71) and other Artifacts (see page 66) can be Associated with the graphical elements. An arrowhead on the Association indicates a direction of flow (e.g., data), when appropriate.

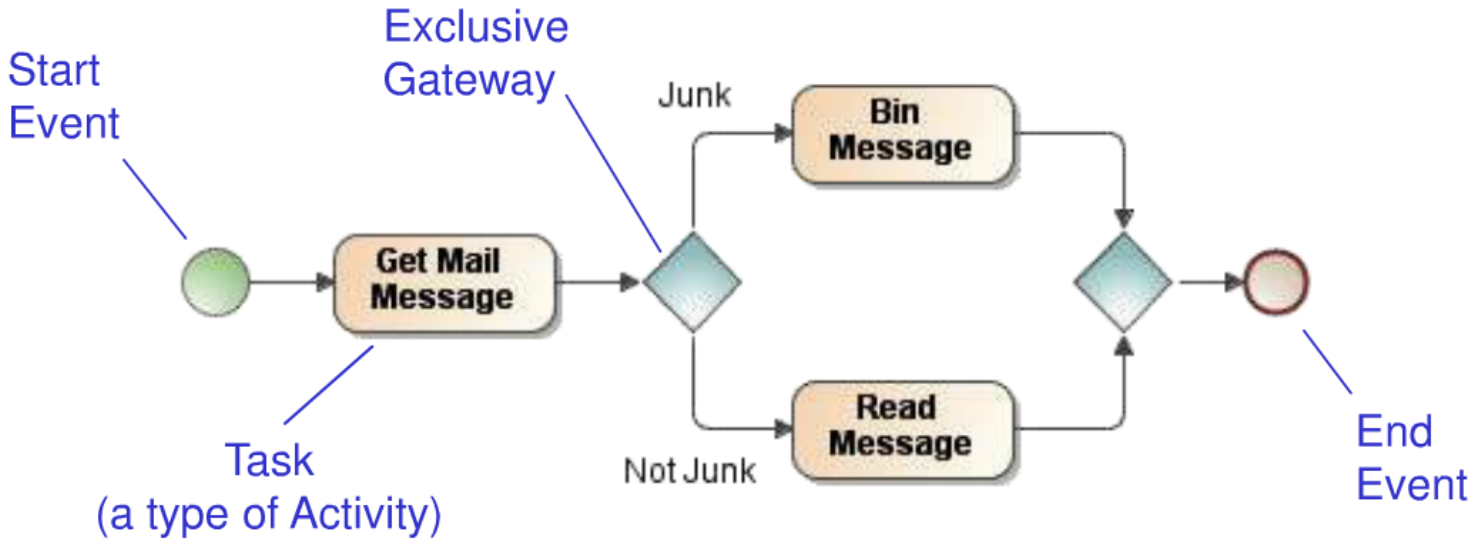
A Basic Sequence Flow

A Process as a set of Flow Objects

A process is defined as a sequence of Flow Objects:

- Events – something that happens during the process
- Activities – work performed in the process
- Gateways – split/merge flow through the process

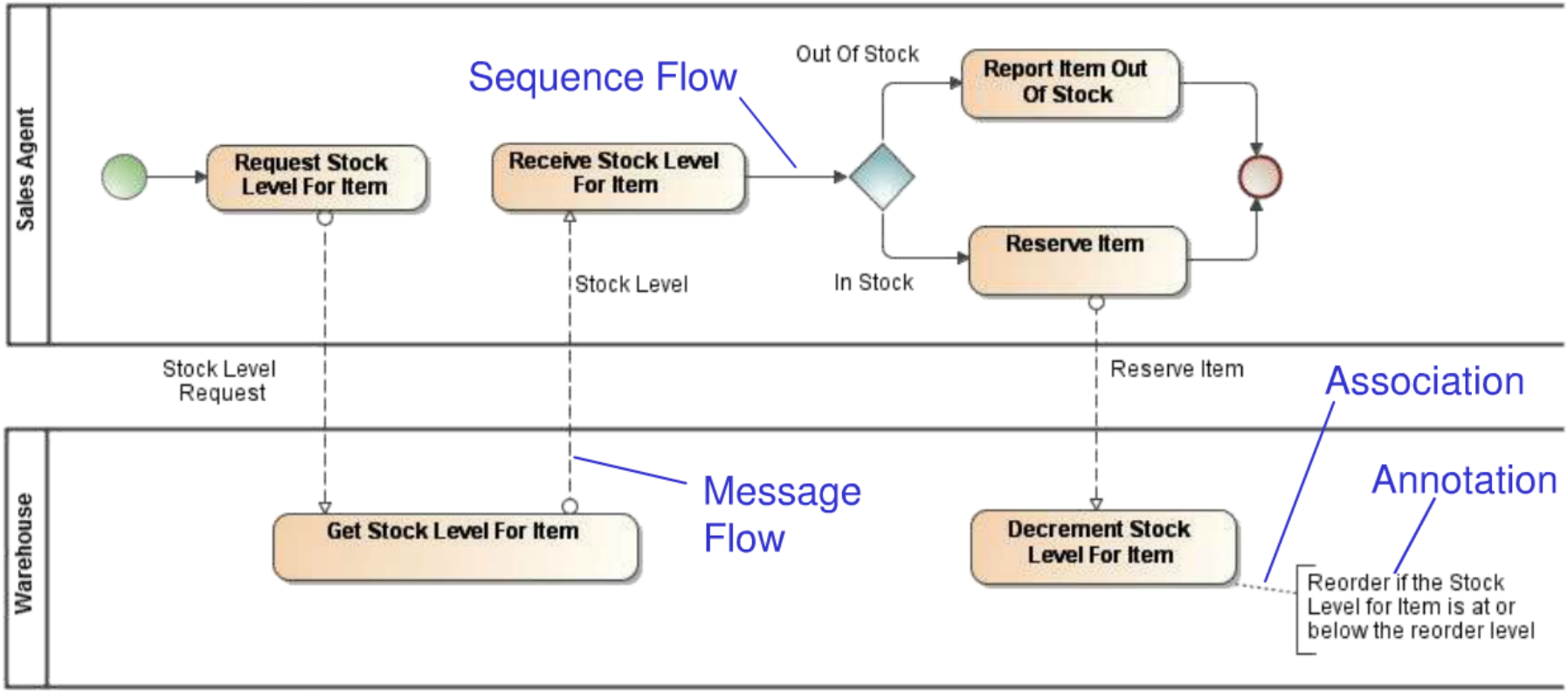
There are many types of Flow Object!



Connection Objects (with a WRONG diagram…)

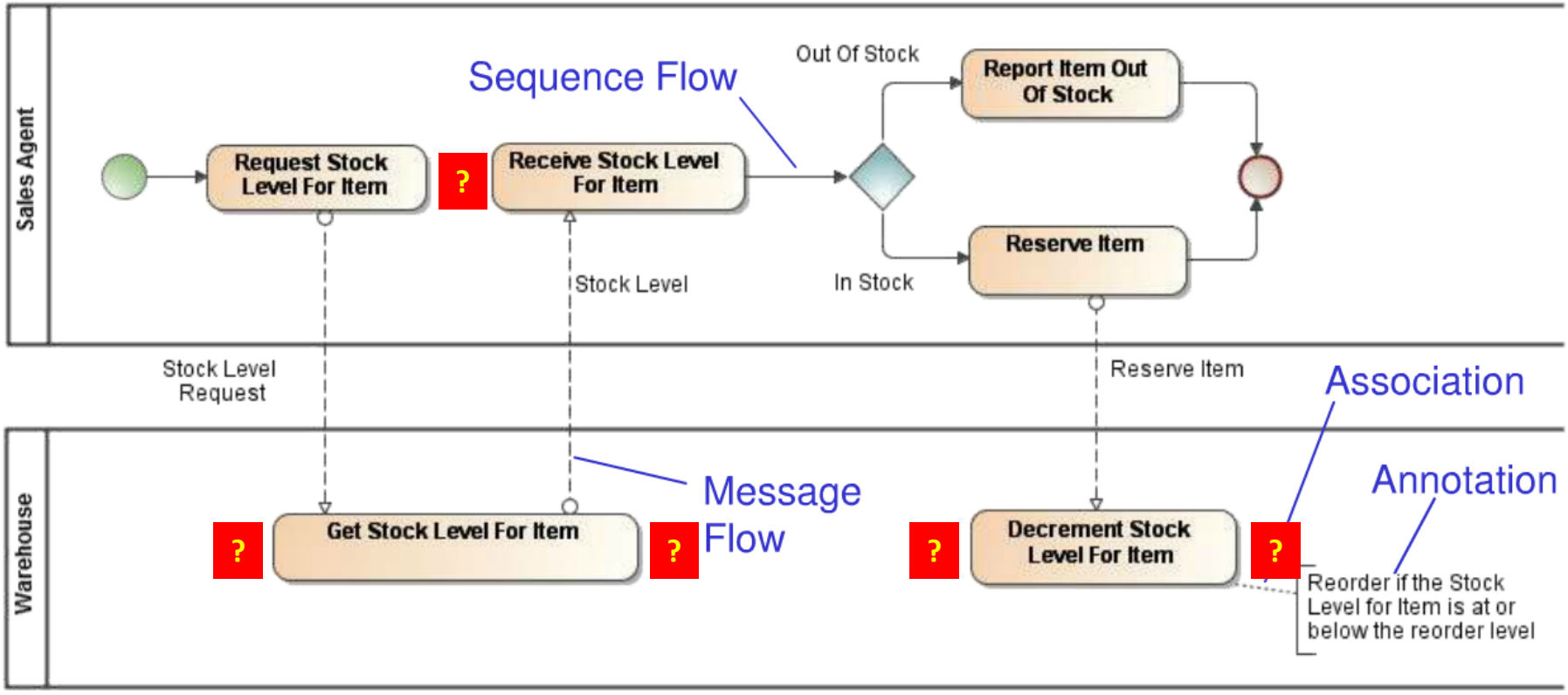
Connect Flow Objects together:

- Sequence Flows – determine the sequence of Activities
- Message Flows – messages between process participants
- Associations – associate text or data with modelling elements



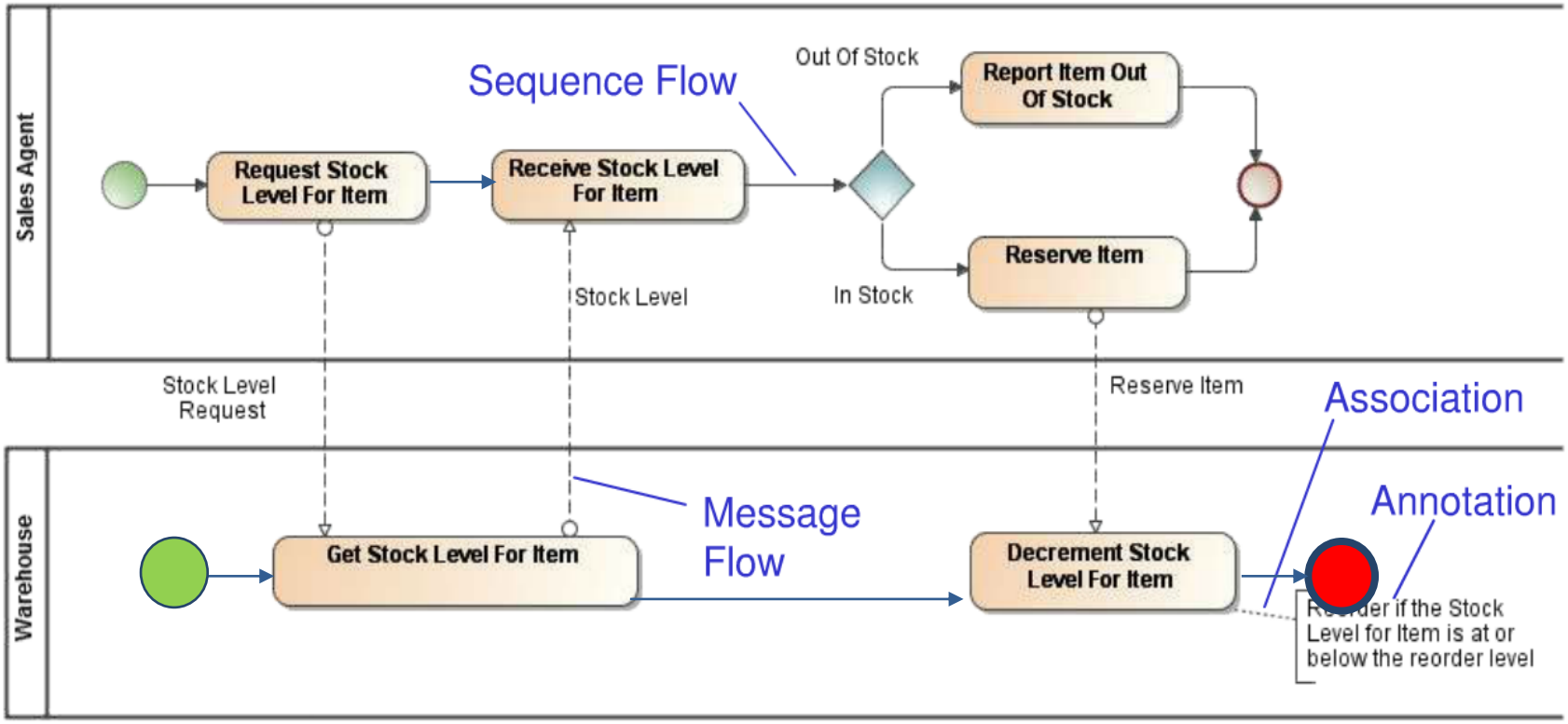
Connection Objects (WHAT IS WRONG…)

NOTE: BTW, Please be aware that this is not a complete diagram! There are still missing elements, at least in the places indicated by ?



Connection Objects (now it all seems to be OK ;-)

Below, one possible example of how this diagram could be corrected (only in the sense of making it a “legal BPMN diagram”, ignoring the eventual real semantics of the business, for what real specific requirements should be considered…)



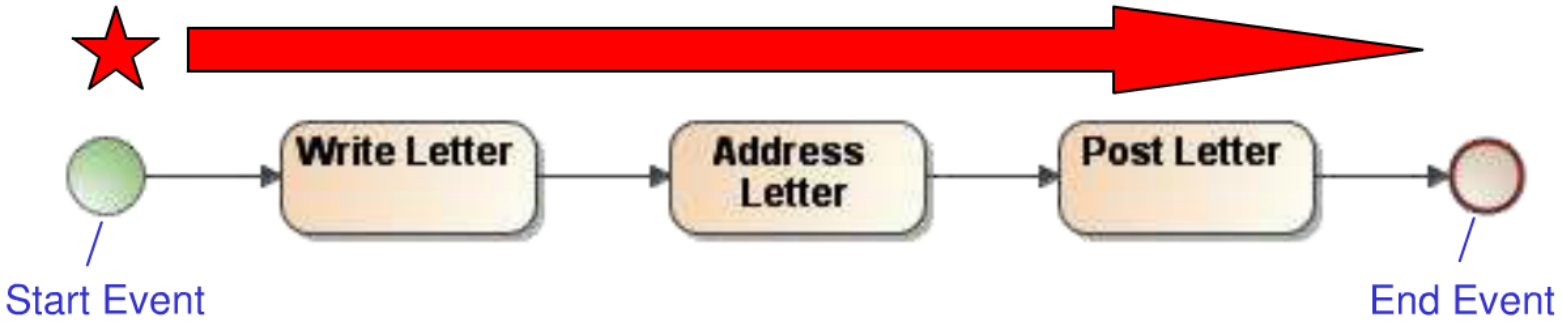
Process Semantics (the Token Game)

Token game – a token is an imaginary focus of control that you imagine flowing around the Process

Tokens traverse from a source Flow Object to a target Flow Object via a Sequence Flow

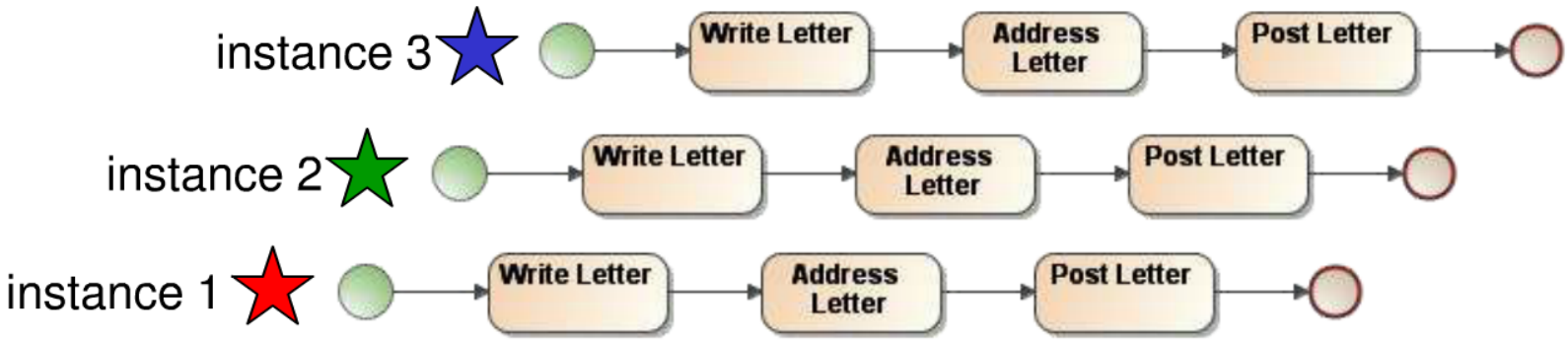
A Flow Object executes when it has tokens on one or more of its input flows

- When a Flow Object starts to execute it takes tokens off its input flows
- When a Flow Object has finished executing it offers tokens on one or more of its output flows

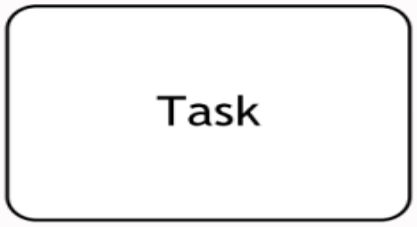


Process vs. Process Instance

- Each time a process receives a new Start Event, a new instance of that process begins executing
- We say that a process may have many *process instances*



Activities



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.

Activity Markers

Markers indicate execution behavior of activities:

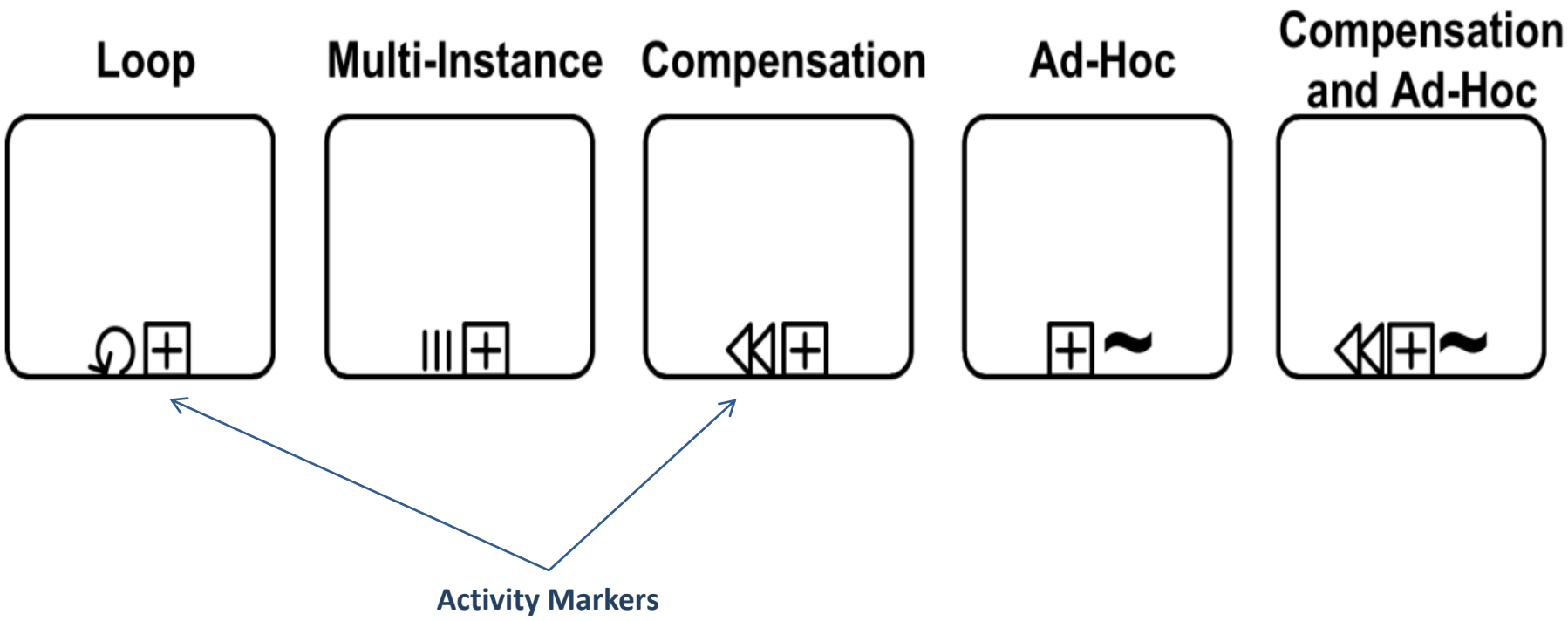
- Sub-Process Marker
- Loop Marker
- Parallel MI Marker
- Sequential MI Marker
- Ad Hoc Marker
- Compensation Marker

Task Types

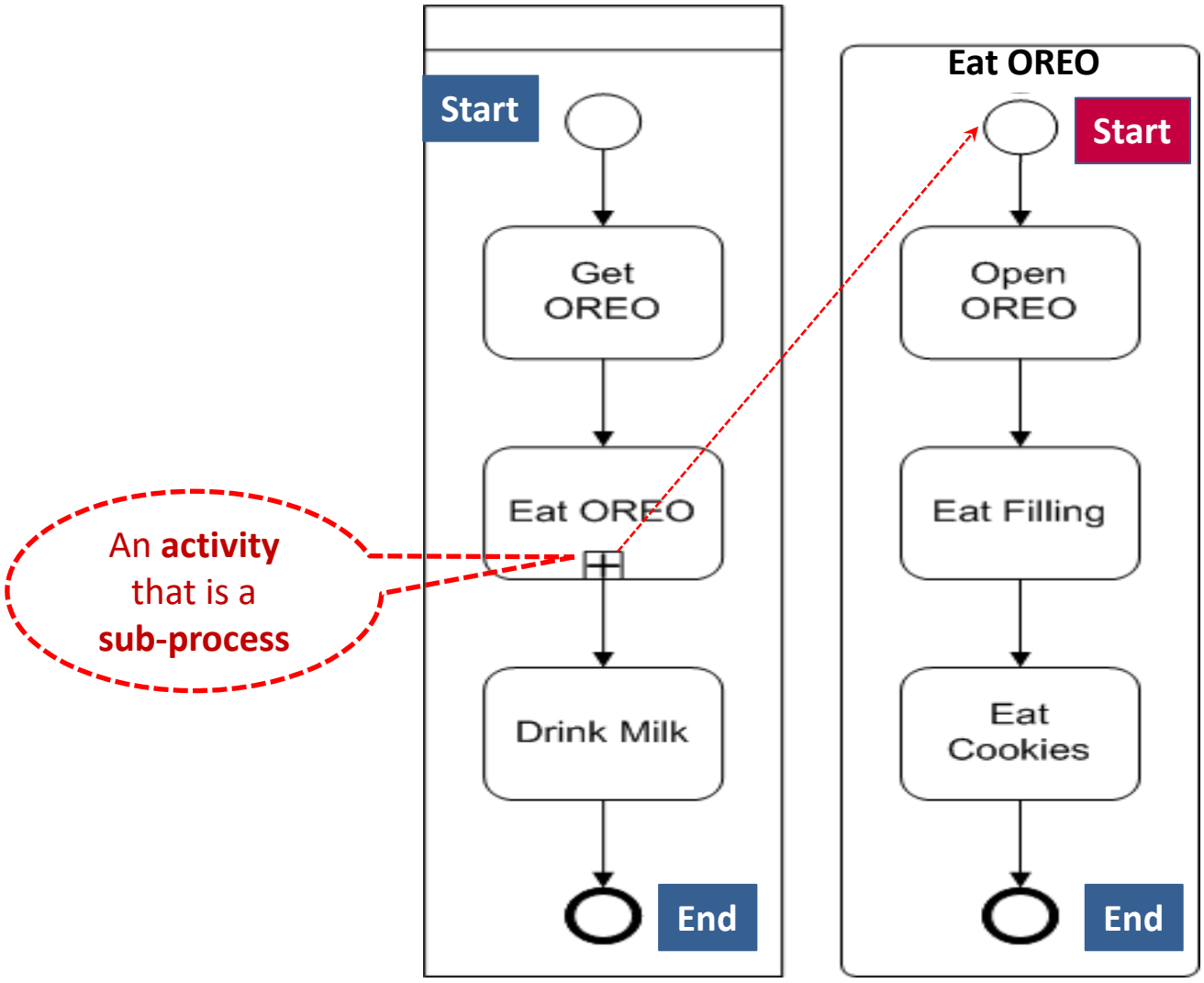
Types specify the nature of the action to be performed:

- Send Task
- Receive Task
- User Task
- Manual Task
- Business Rule Task
- Service Task

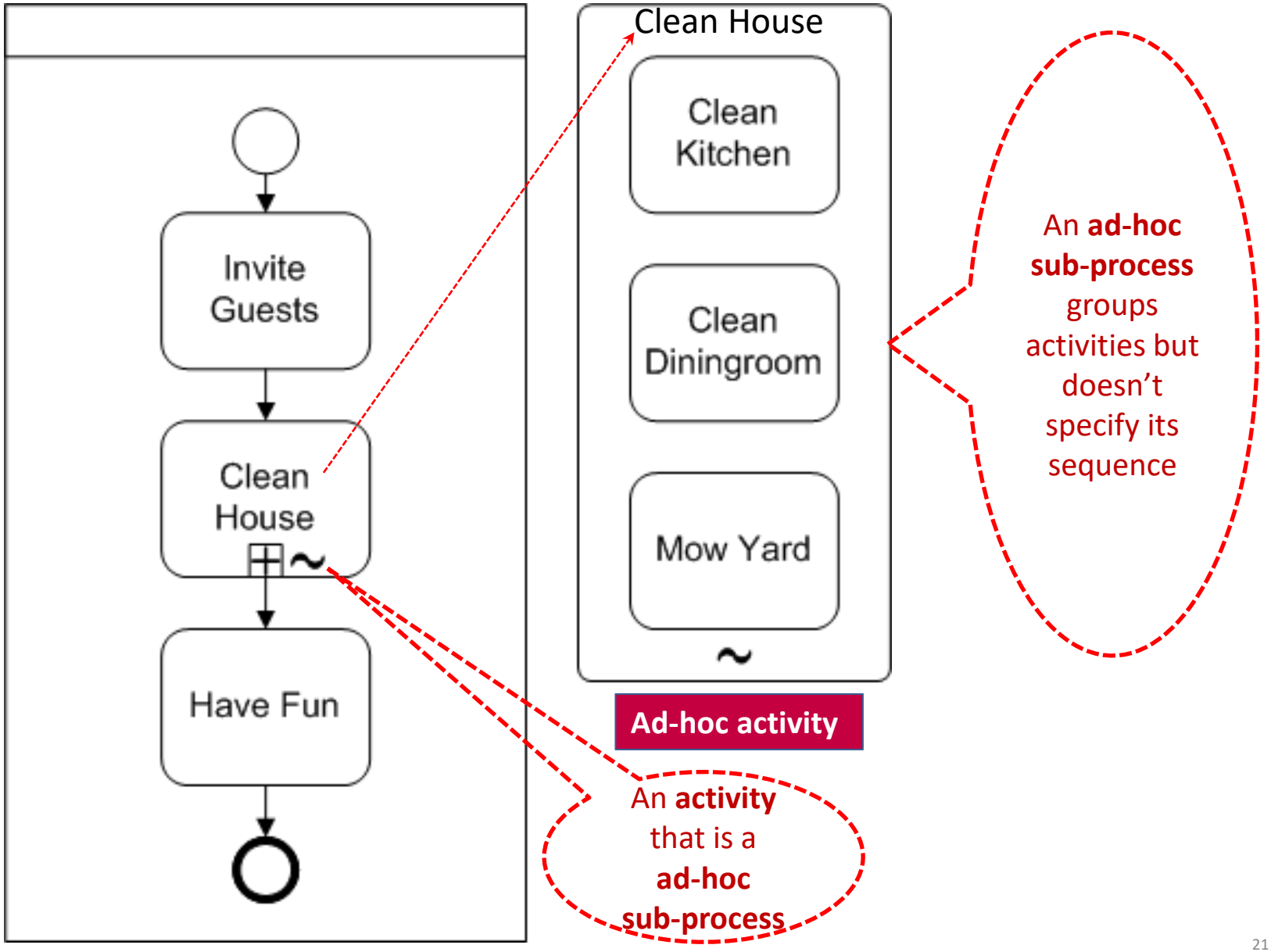
Activity Types



Activities


Start
Get OREO
Eat OREO
Drink Milk
End
An **activity** that is a **sub-process**
Eat OREO
Start
Open OREO
Eat Filling
Eat Cookies
End

Activities


Invite Guests
Clean House
Have Fun
Clean House
Clean Kitchen
Clean Diningroom
Mow Yard
Ad-hoc activity
An ad-hoc sub-process groups activities but doesn't specify its sequence
An activity that is a ad-hoc sub-process

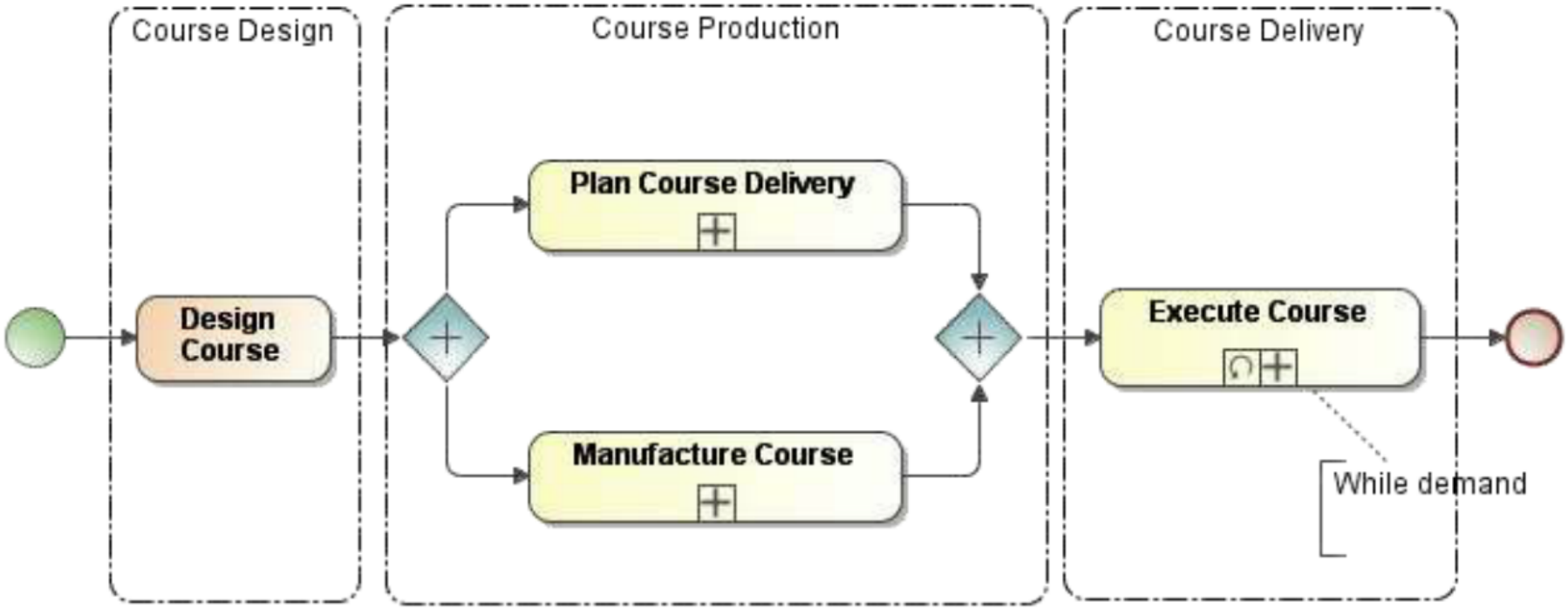
BPMN Elements: Artefacts and Data

Artifacts

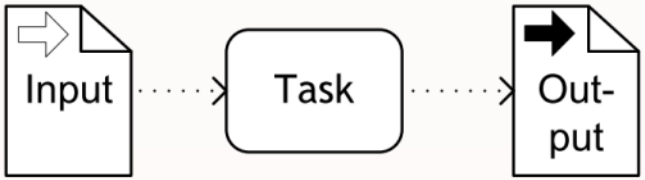
- **Group**: groups activities; used to better understanding the diagram; doesn't change flow sequence or process semantics (similar to UML package).
- **Annotation**: allows to add comments/notes to diagrams.

Groups

- Groups are simply a way to organize and highlight parts of the model in order to increase its comprehensibility
  - They have no semantics beyond a simple organizing role
- In the example below, the Groups indicate phases in the process

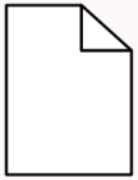


Data



A **Data Input** is an external input for the entire process. It can be read by an activity.

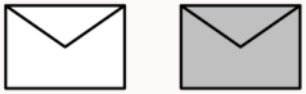
A **Data Output** is a variable available as result of the entire process.



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 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.

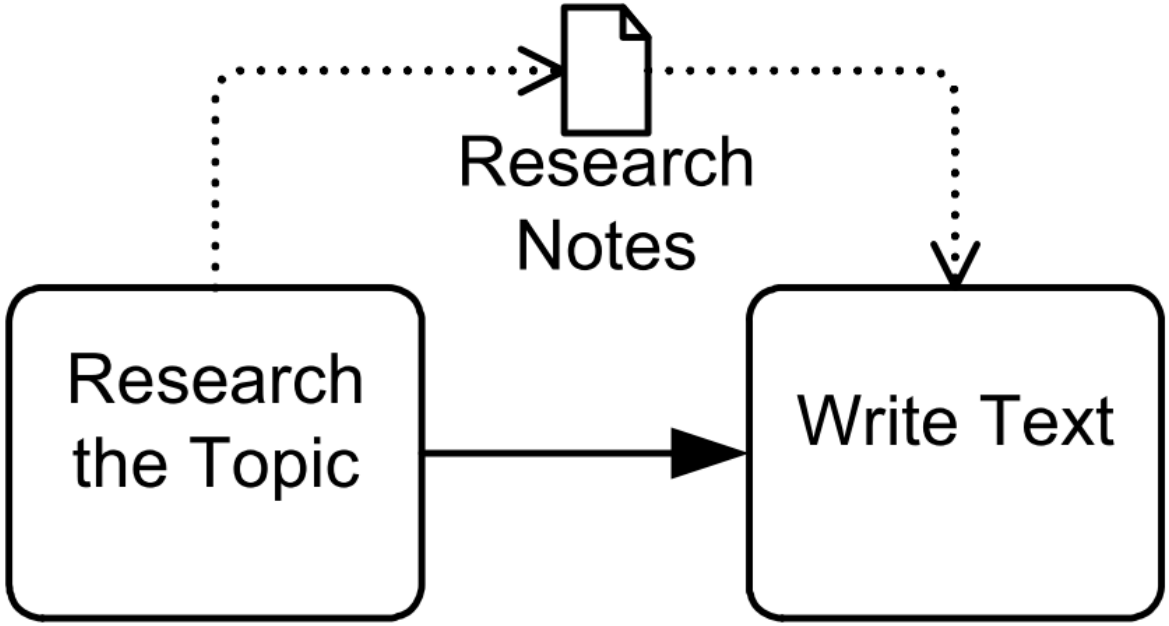


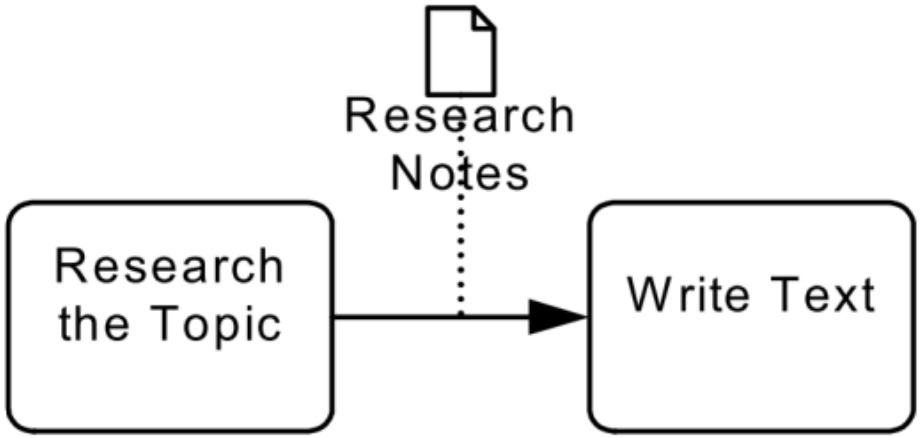
A **Message** is used to depict the contents of a communication between two Participants.

Data Objects

- Data Objects may have a state –
  "A condition or situation during the life of an object during which it satisfies some condition, performs some activity or waits for some event“
  - In UML the states an object may go through are modelled by creating State Machine
  - The example on the previous slide shows the states the Product Specification document goes through
- Data Objects have no effect on the flow!
  - They are connected to other elements by Associations *not* by Sequence Flows

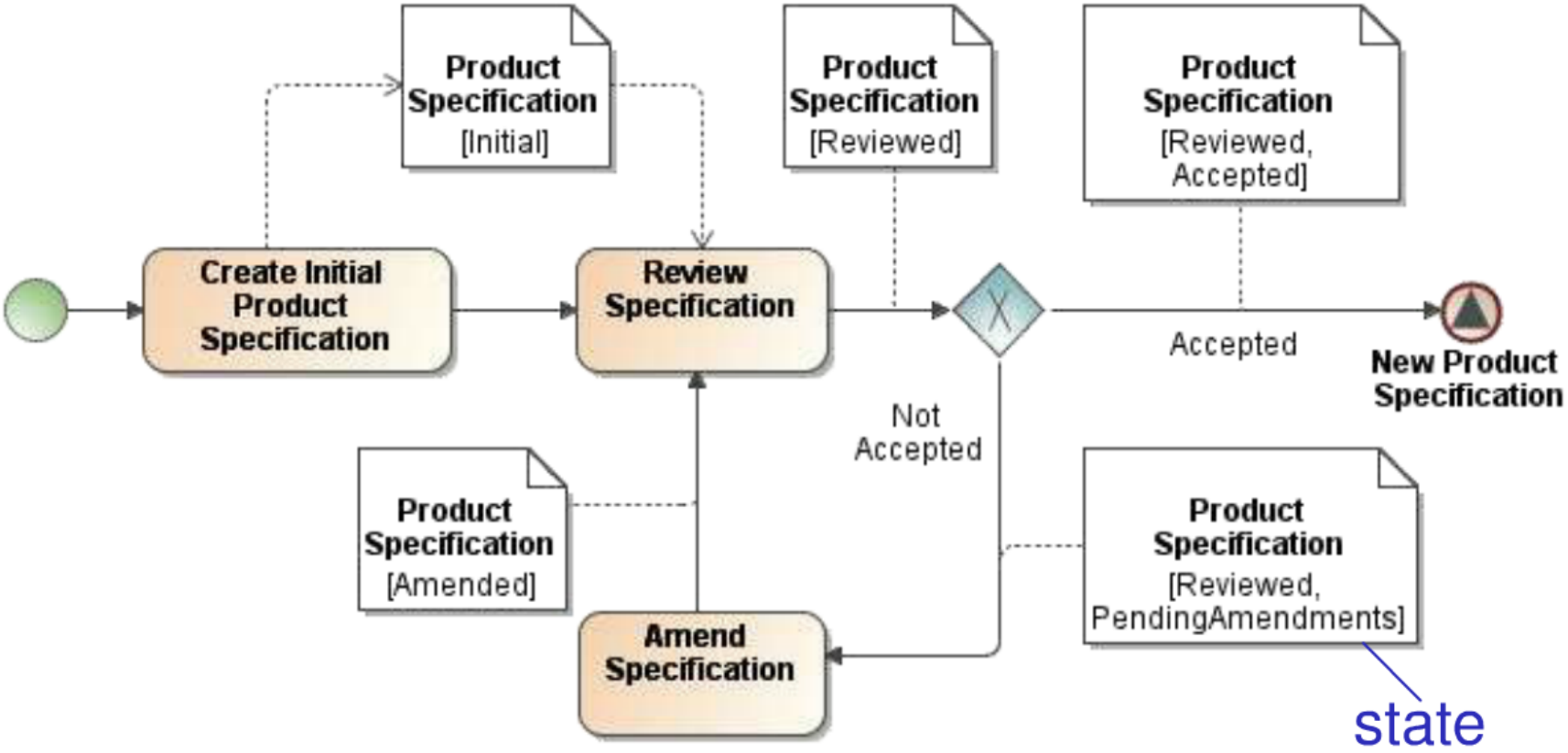
Data Association within a Process

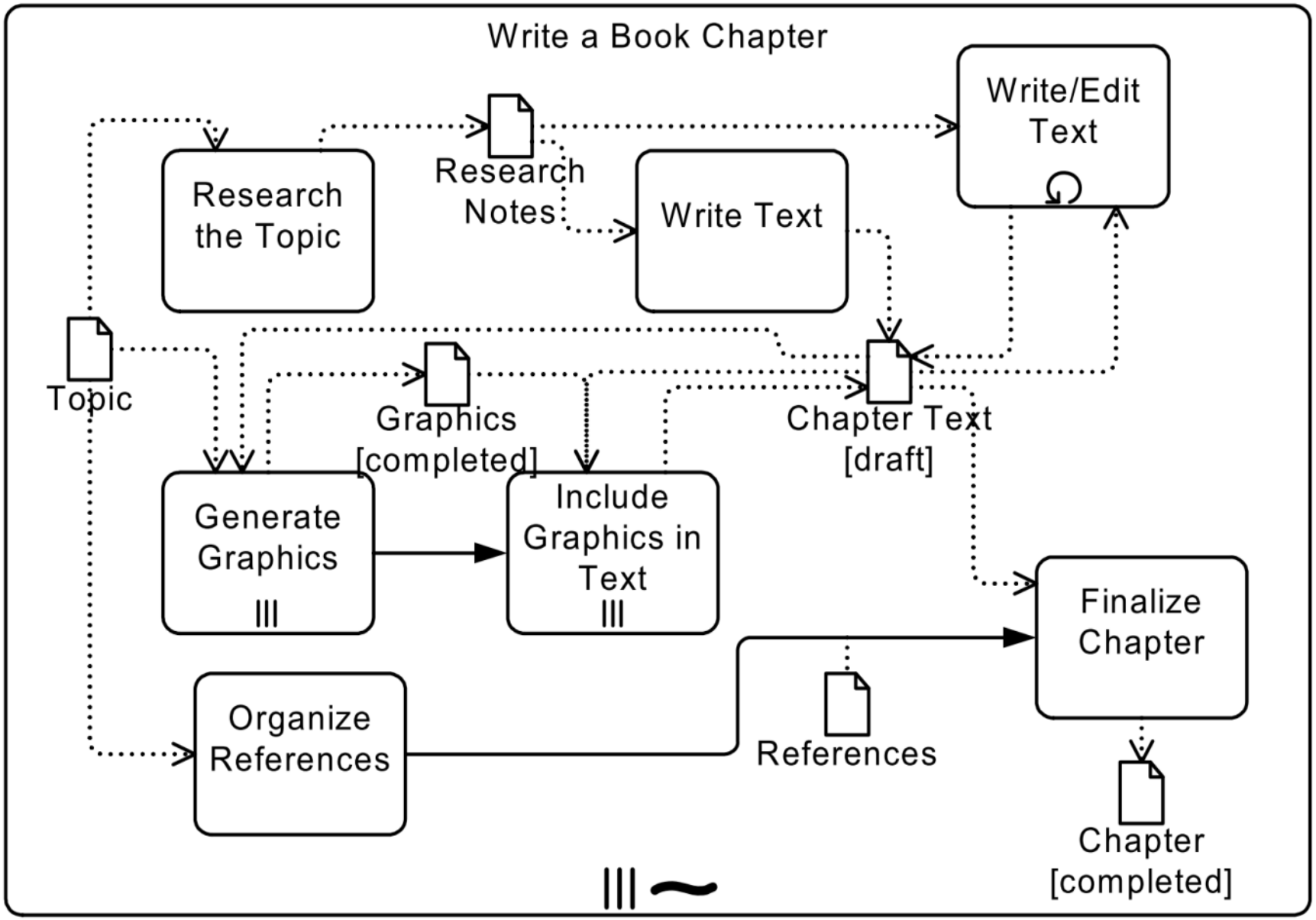




Data Objects

- Data Objects indicate data (often documents) flowing through the process, and possibly being transformed by it
- They may have a state



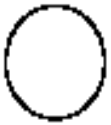

Write a Book Chapter
Research the Topic
Research Notes
Write Text
Write/Edit Text
Topic
Graphics [completed]
Chapter Text [draft]
Generate Graphics
Include Graphics in Text
Finalize Chapter
Organize References
References
Chapter [completed]

BPMN Elements: Events

Events

- An **Event** is a significant occurrence within a business.
- **Events** may affect the sequence or timing of a process.
- **Events** may have **Triggers** that cause the Event to happen and **Results**. **Triggers** and **Results** have special notations (see later).

- There are three **categories** of **Event**:



Start events begin a process (i.e. create a process instance and the corresponding token).

Intermediate events occur within a process instance.

End events terminate all or part of a process.

Event Triggers

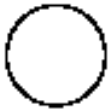
- A Trigger specifies what causes the Event.
- The Trigger is shown as an icon inside the Event symbol.
- BPMN defines several types of Triggers:
  - None (i.e. no Trigger) and
  - 12 other (Message, Timer, Error, Signal,...)
- A Trigger may have two different **behaviours**:

Throw a Trigger  (throw/send a message)

Catch a Trigger  (catch/receive a message)

Event Behaviour (throw/catch)

- When **throwing** a Trigger the Event **waits for a token** and then ***produces* the Trigger**.
  The notation for throw is a *filled* Trigger.
  
- When **catching** a Trigger the Event **waits for a token** and then ***waits* for the Trigger**.
  The notation for catch is an *outlined* Trigger.
  
  "Catch" is *blocking* while it waits for the Trigger and the Token.


Start events: can only **catch**


Intermediate events: can **catch** and **throw**

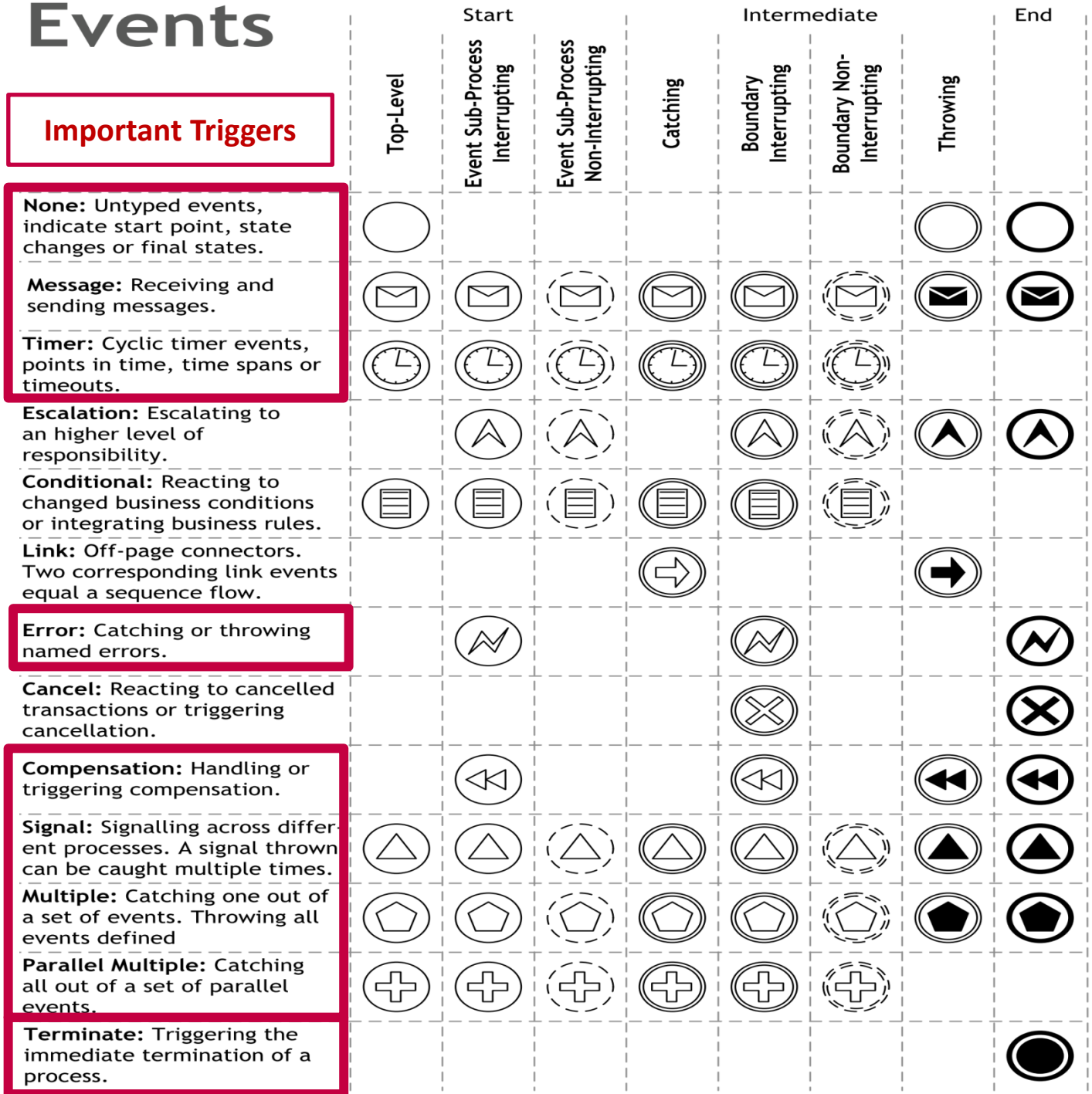

End events: can only **throw**

Example: Events with a Message Trigger

	Trigger	Category	Behaviour
	Message	Start	Catch
	Message	Intermediate	Catch
	Message	Intermediate	Throw
	Message	End	Throw

Detailed explanation can be found on the BPMN 2 standard, section 8.3.5

Events



Important Event Triggers

- **None/Untyped** (throw/catch)

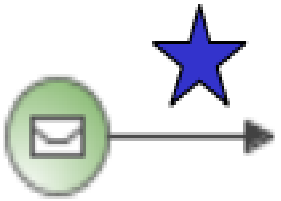
- **Timer** (/catch)
- **Message** (throw/catch)
- **Signal** (throw/catch)

- **Error** (throw/catch)
- **Compensation** (throw/catch)
- **Terminate** (throw/)

Start Event Semantics

- A Start Event begins a new instance of a business process
  - It has a single outgoing Sequence Flow and *no* incoming flows
- When triggered, it emits a token on it's single outgoing flow
  - The Trigger is the *cause* of a Start Event e.g. a timer going off, a message or signal being received or None

e.g. a Message Start Event emits a Token on receipt of a specific Message



Timer Start Event

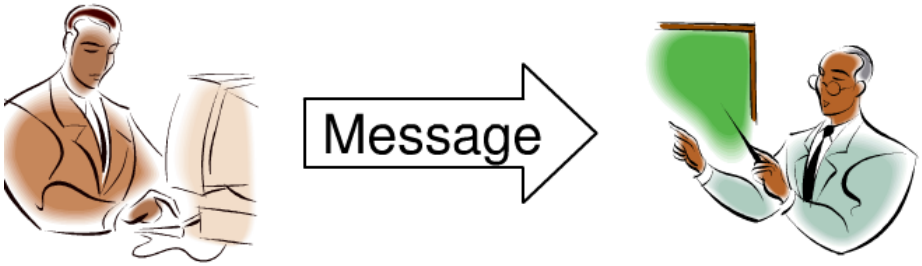
Timer Start Events are triggered when a time condition becomes true e.g.

- "At End Of Week" - a specific point in time
- "After 2 Weeks" - after a duration in time
- "Every Tuesday" - a repeating condition
  - Note: using a specific time or date (e.g. 1st Feb. 2009) may inhibit the reusability of the process

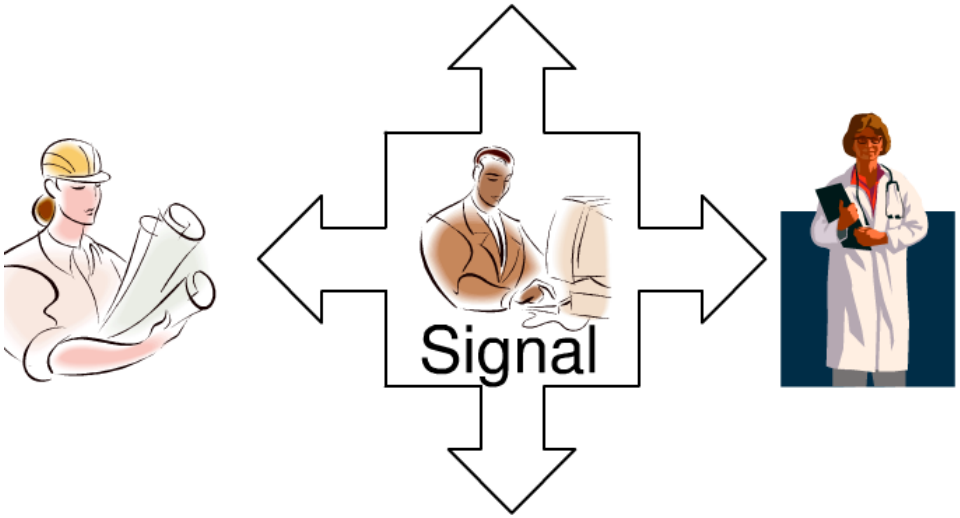
It is crucial to be very clear and precise when stating time conditions!

Messages vs. Signals

Messages and Signals are packets of data stored as Properties (see notes)



Messages are sent 1 to 1 and include references to the sender and receiver

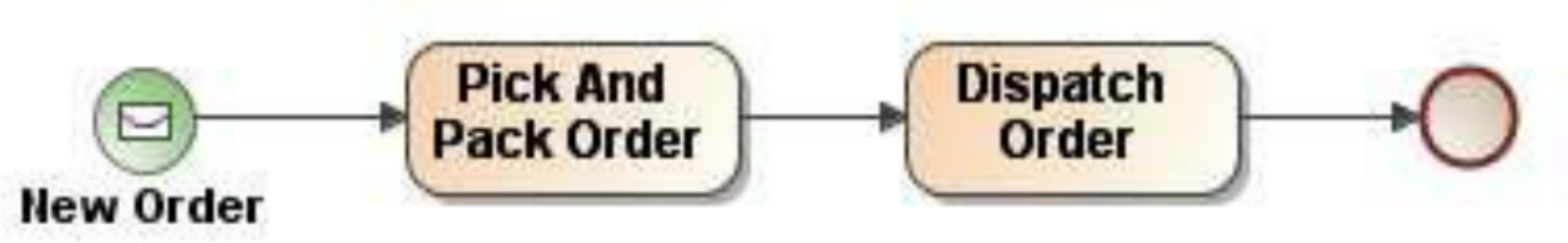


Signals are broadcast 1 to many and neither the sender nor the receiver are specified

Message Start Event

Message Start Events are triggered by receipt of a message from a business participant *external* to the current process

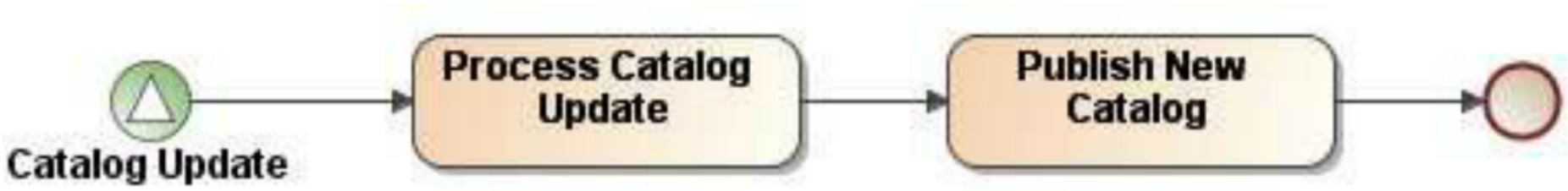
- A message is simply a direct communication between *exactly two* business participants - we'll see how to send messages later



Signal Start Event

Signal start events are triggered by receipt of a signal from a business participant or process *external* to the current process

- A signal is broadcast from an external business participant or process to *many* others



End Event Semantics

End Events may:

- End a *particular path* through a business process
- End the *whole process* (Terminate End Event)
- Generate a Result (e.g. a Message or Signal being sent)

They have incoming flows and *no* outgoing flows because they consume their input tokens

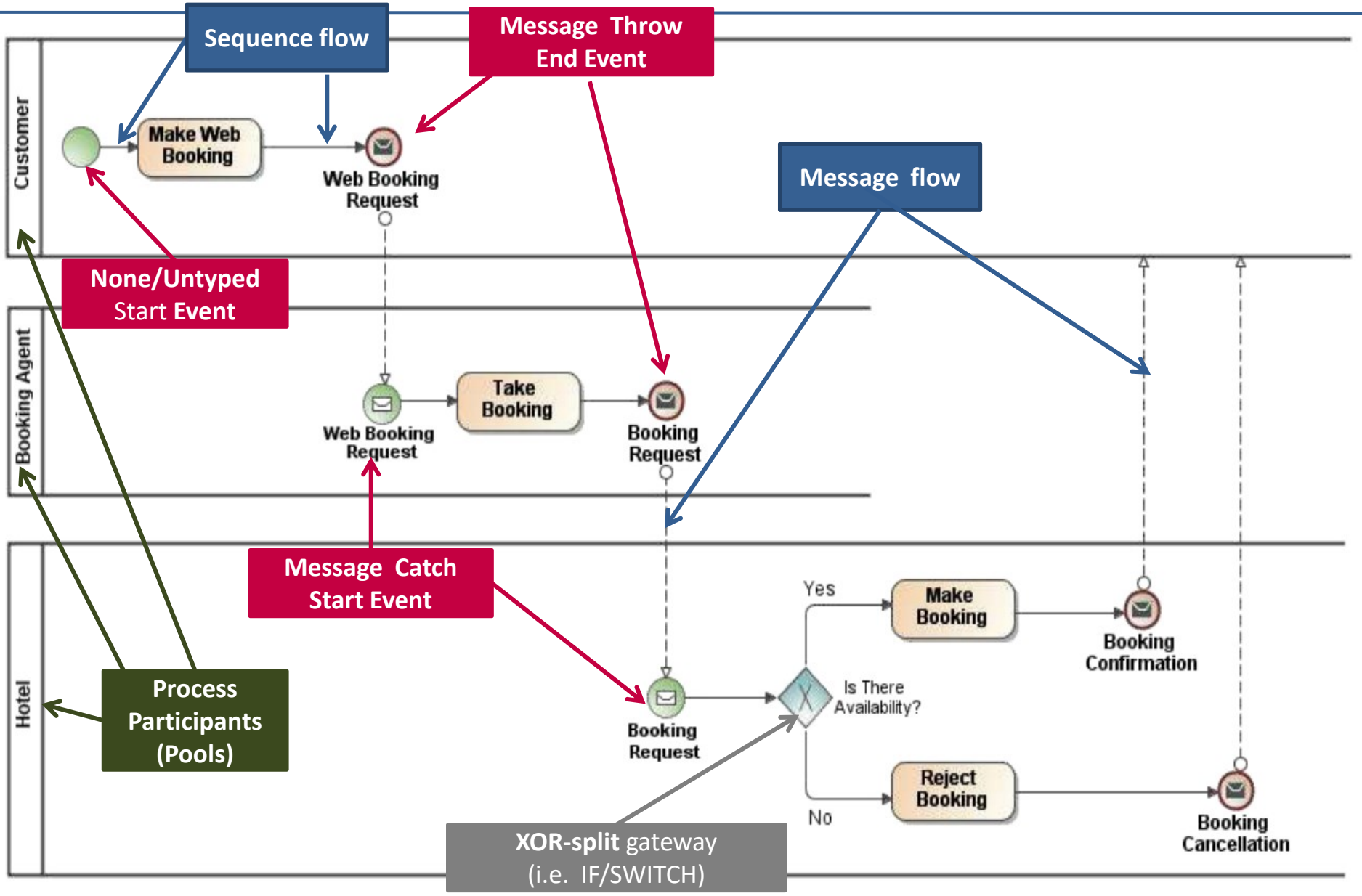
Message End Event

- When this Event receives a Token it emits a **Message** that is sent to a particular participant or process.

- Can only send the Message to a different participant or process (different Pools)! If the target is within the same Pool then Signals must be used instead.

We will go back to this later on when talking about collaboration and process diagrams!

Message Start & End Event



Signal End Event

When this Event receives a Token it broadcasts a Signal to every process set up to receive it

- Unlike a Message, a Signals attributes specify neither it's Source nor it's Target

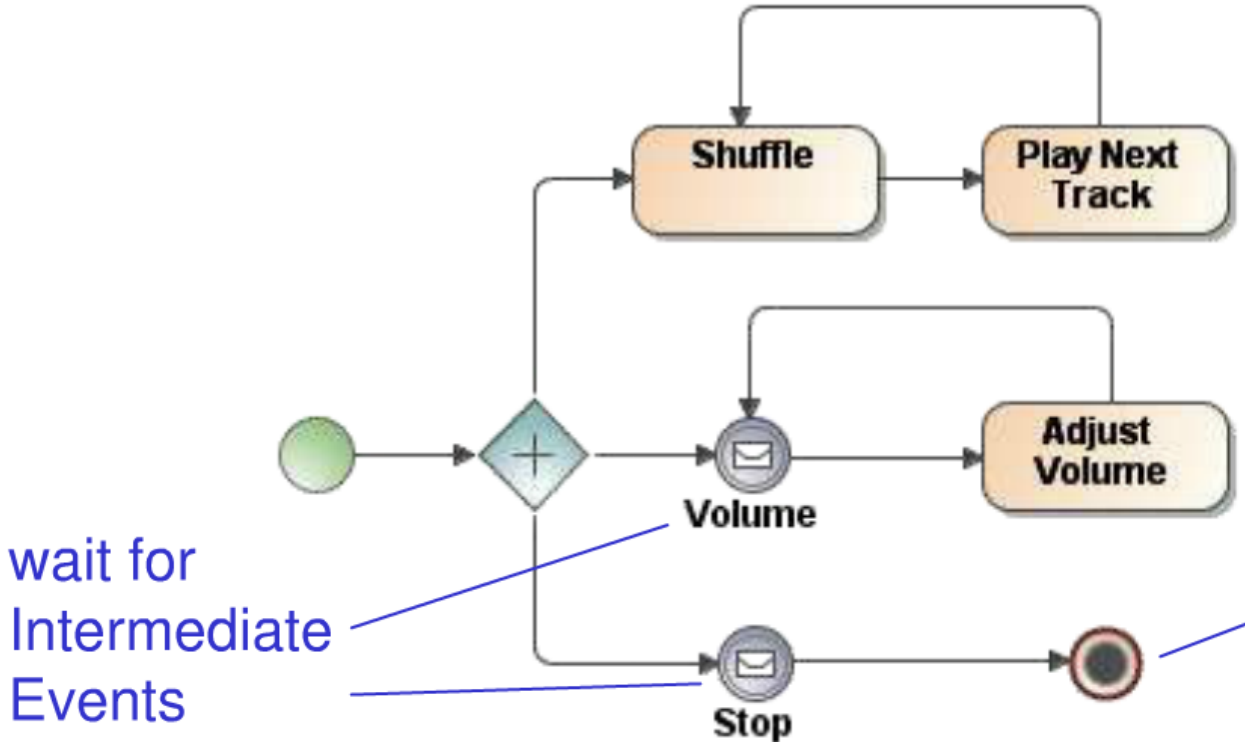
Sends the Newsletter

Any process that is waiting for a Newsletter can receive this Signal

Recall that a **Message** has a specific Recipient whereas a **Signal** is caught by any Process waiting for it

Terminate End Event

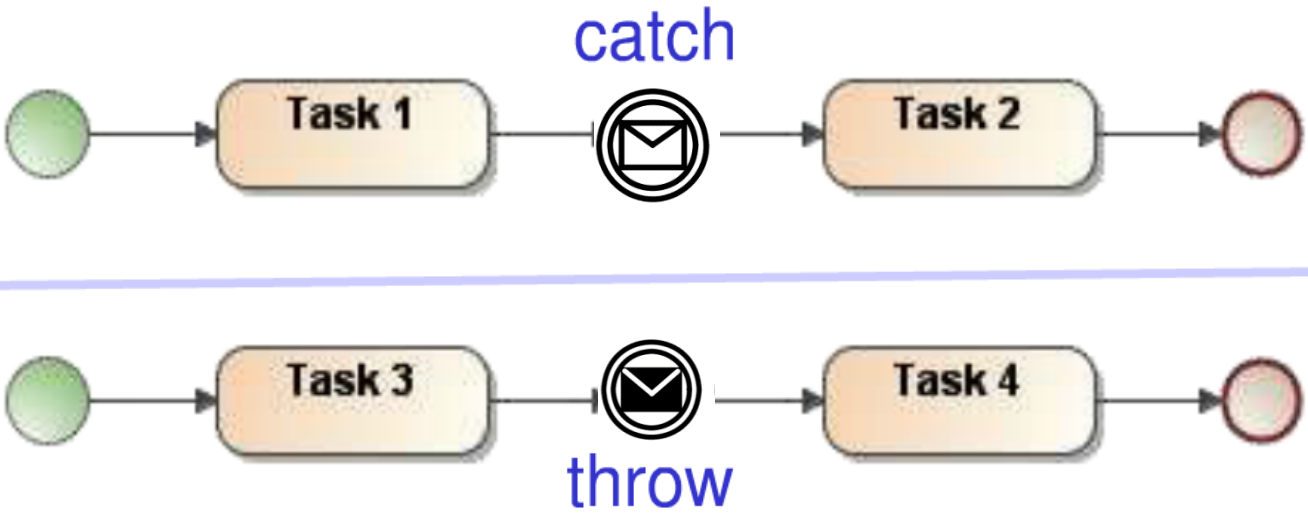
- When this Event receives a Token *all* Activities in the process stop!
  - A common use for a Terminate End Event is to stop a process after an interrupt or non-recoverable error



Terminate End Event and **None End Event** are similar iff a Process is not concurrent

Intermediate Events

- Intermediate Events occur *during* a business process
- They do *not* spawn a new process instance!
- Intermediate Events may catch *or* throw:
  - Catch - waits for the Event Trigger then emits a token
  - Throw - throws the Event Trigger then emits a token



Intermediate Event Semantics

- Catch:
  - If there is an input flow:
    - Wait for a token
  - Wait for the Event Trigger
  - Emit a token on the single output flow

Recall that a *catch* waits (blocks the token) until a Trigger is received!

- Throw:
  - Wait for a token on the single input flow
  - Emit the Event Trigger
  - If there is an output flow:
    - Emit a token on the single output flow

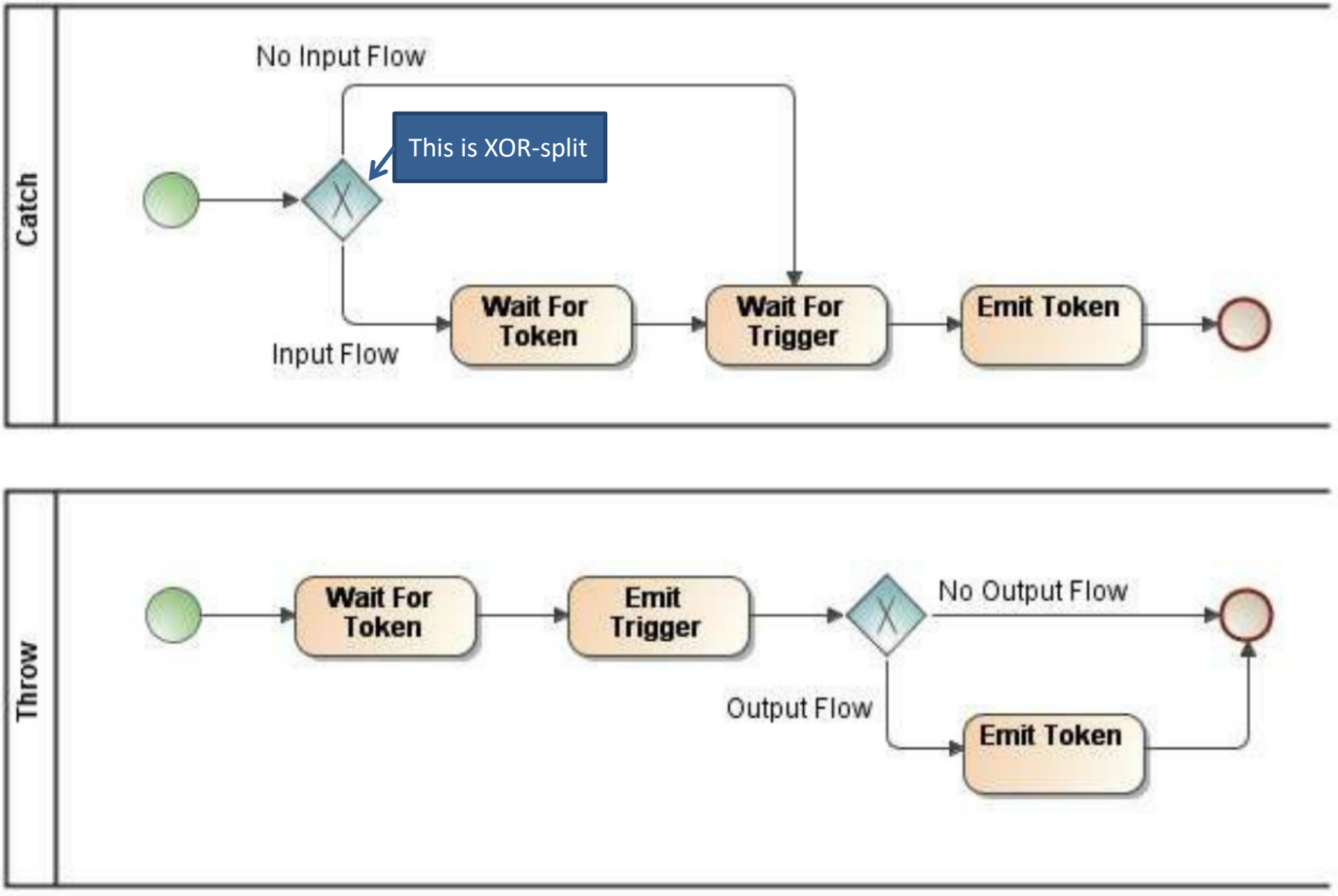
BPMN Exercise

- Use BPMN to model the semantics of Intermediate Events.

  - Catch:
    - If there is an input flow:
      - Wait for a token
    - Wait for the Event Trigger
    - Emit a token on the single output flow
  - Throw:
    - Wait for a token on the single input flow
    - Emit the Event Trigger
    - If there is an output flow:
      - Emit a token on the single output flow

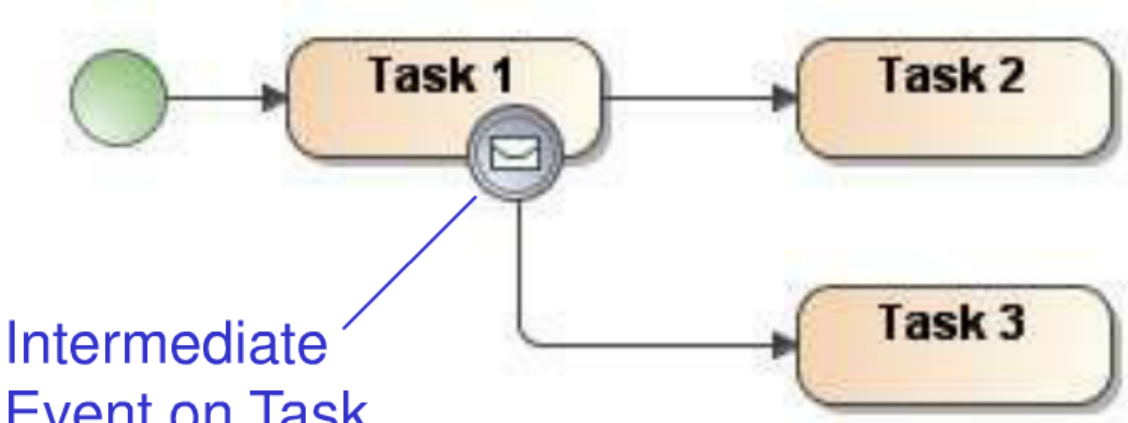
*(This is called **meta-modelling**, i.e. creating a model of a model)*

BPMN Intermediate Event semantics modelled with BPMN

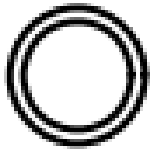


Intermediate Events as Interrupts

- Intermediate Events are often placed on Activity boundaries to act as interrupts
- If the Event Trigger occurs *before* the Activity is completed, the Activity is interrupted, and the Intermediate Event emits a token



"None" Intermediate Event

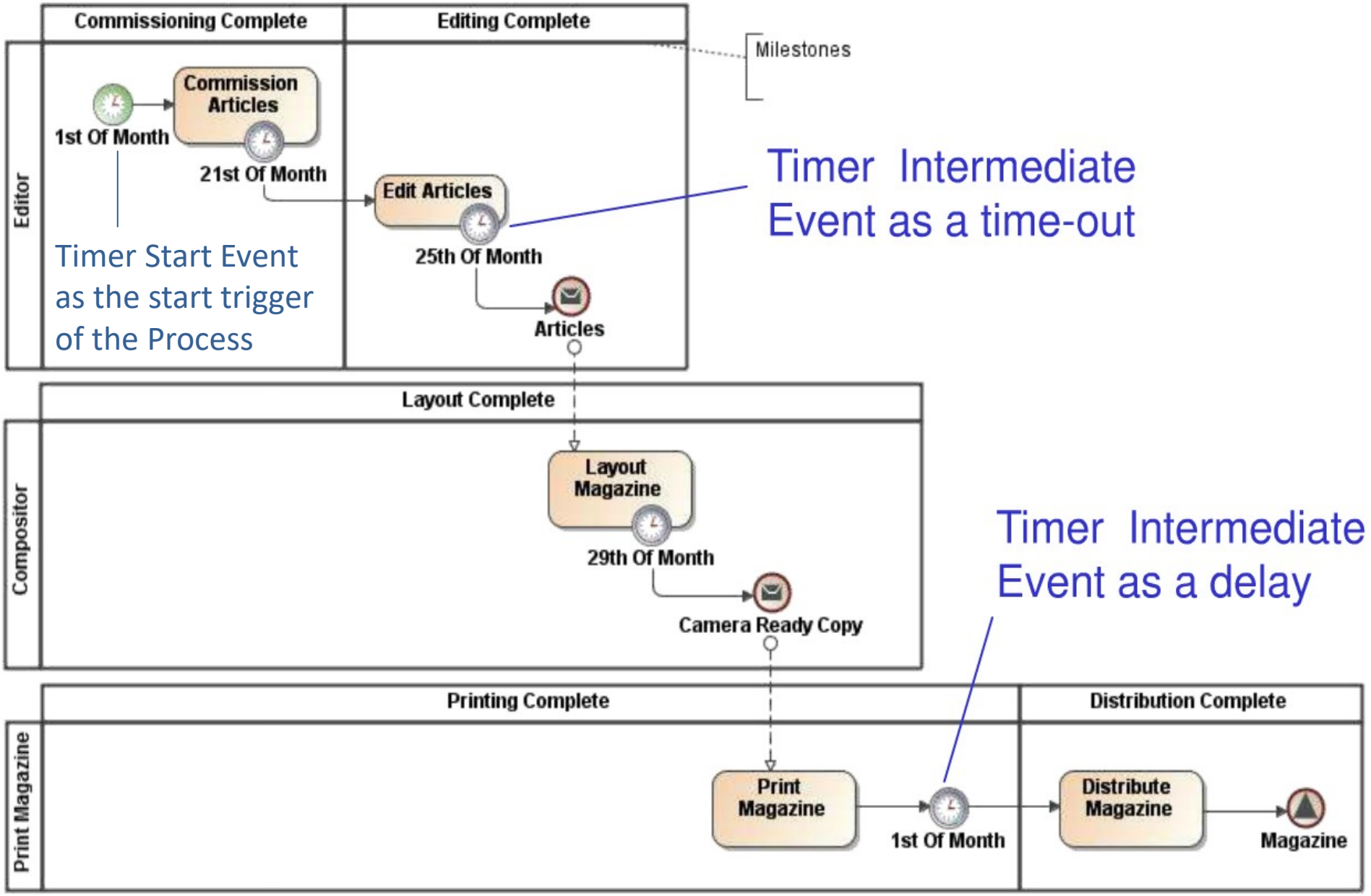


- This doesn't really *do* anything – it has no Trigger, so it fires immediately on receiving a token
- Use it as documentation to indicate that the process has reached an important intermediate state or milestone

Timer Intermediate Event

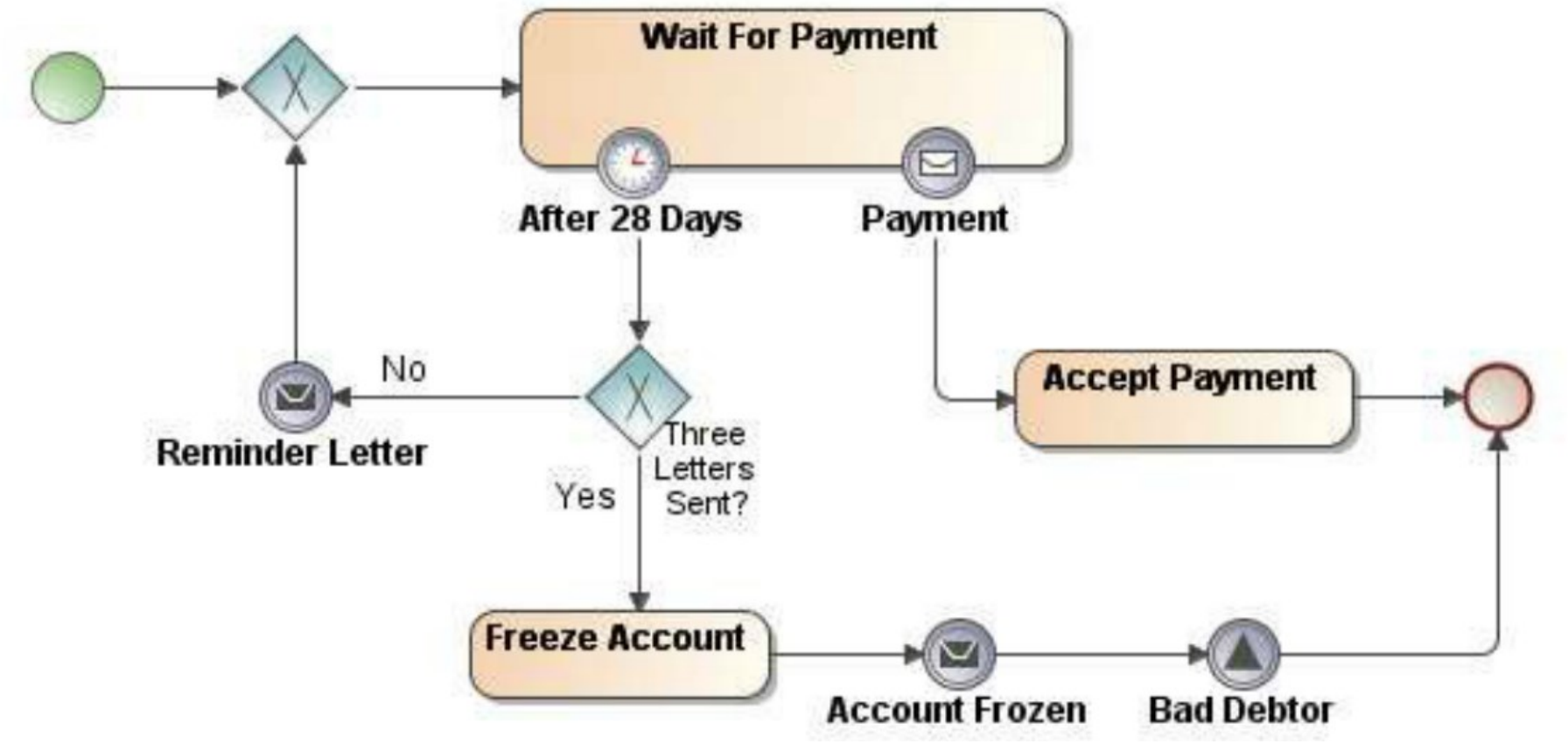
- Triggers when a time condition becomes true
- They can be used in two ways:
  - Time-out – when placed on an Activity boundary, they act as a time-out because the Activity is interrupted when the time condition becomes true
  - Delay – when they are placed inline in a Sequence Flow, they cause the flow to pause until the time condition becomes true

Timer Intermediate Event



Message & Signal Intermediate Events

Message and Signal Intermediate Events send and receive Messages and Signals

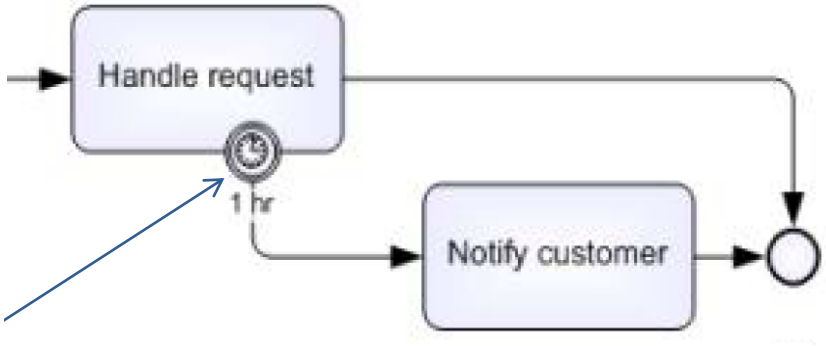


Interrupting and Non-Interrupting Events

Interrupting

Interrupting: when the event is thrown or caught the corresponding activity is interrupted and is not completed.

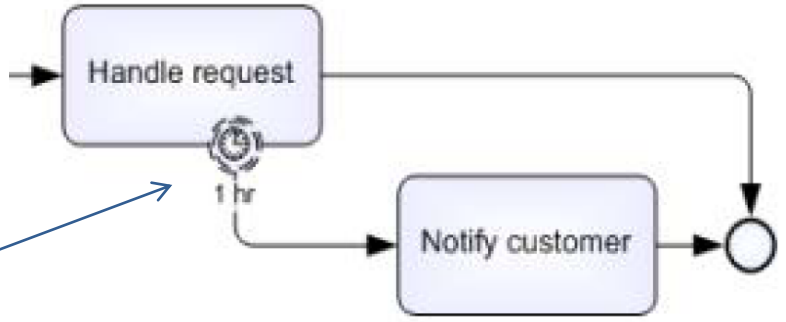
Notation is a **solid line**.

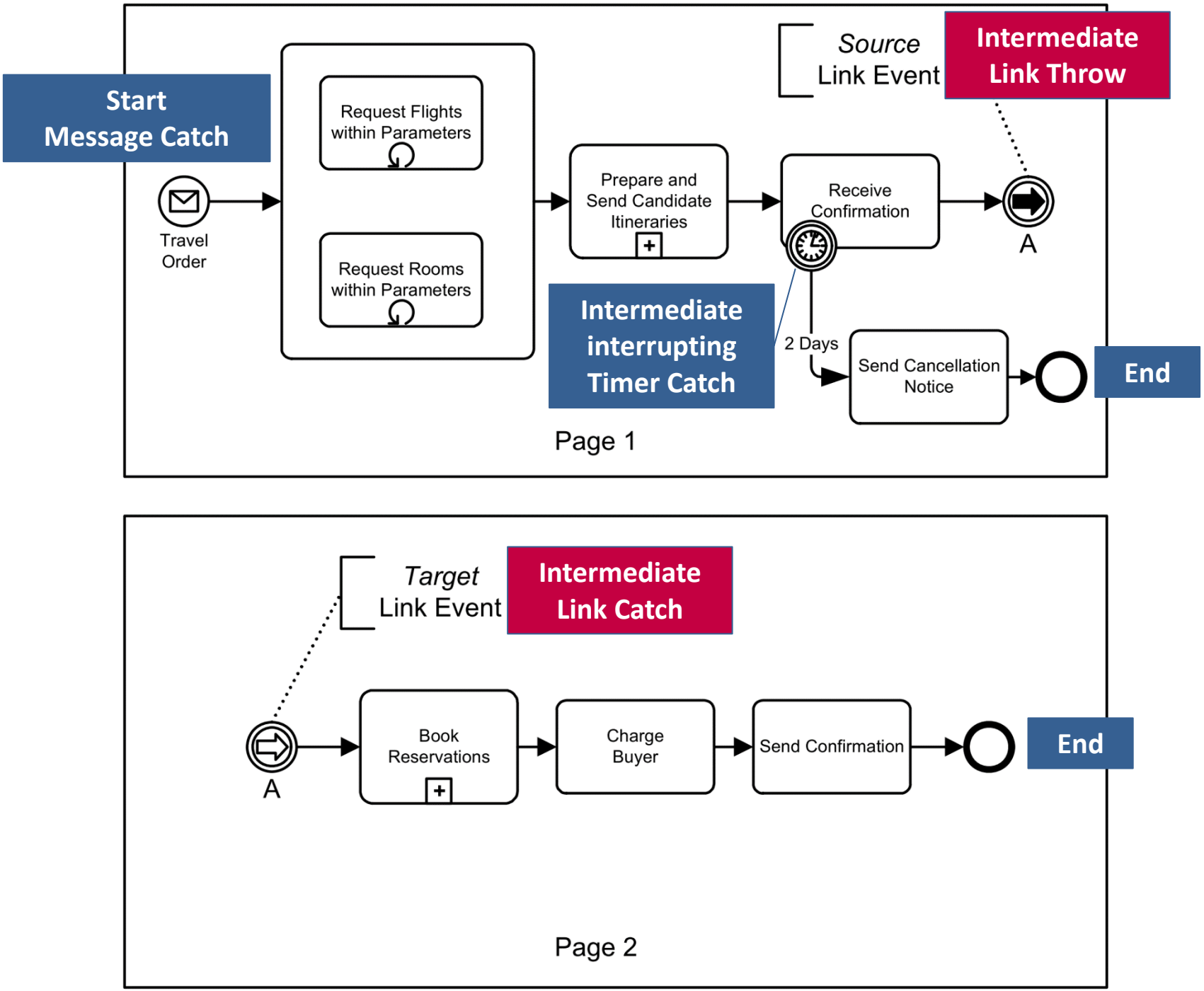


Non-Interrupting

Non-Interrupting: when the event is thrown or caught the corresponding activity continues its execution.

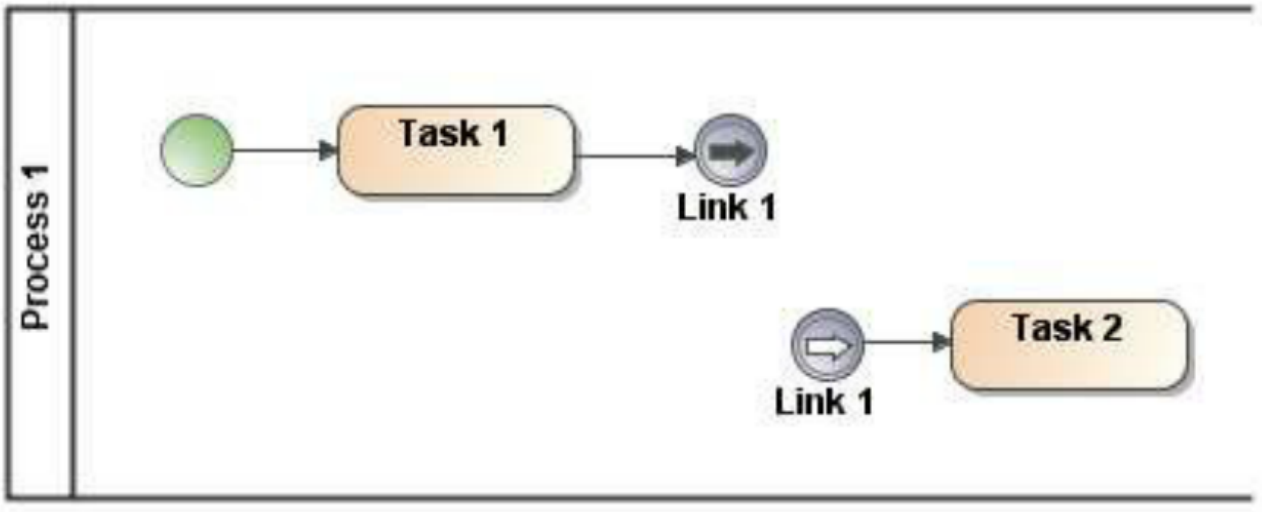
Notation is a **dotted line**.




Start
Message Catch
Travel
Order
Request Flights
within Parameters
Request Rooms
within Parameters
Prepare and
Send Candidate
Itineraries
Source
Link Event
Intermediate
Link Throw
Receive
Confirmation
A
Intermediate
interrupting
Timer Catch
2 Days
Send Cancellation
Notice
End
Page 1
Target
Link Event
Intermediate
Link Catch
A
Book
Reservations
Charge
Buyer
Send Confirmation
End
Page 2

Link Events

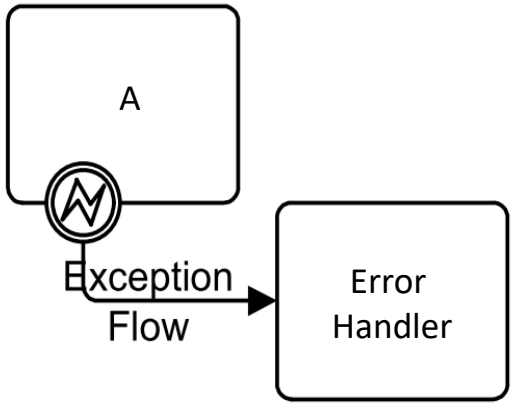
- Link Events allow the connection of two or more parts of the *same* process at the *same* level - they can't cross process boundaries e.g. Process to Sub-Process
- Multiple source Link Events (throw) can link to a single target Link Event (catch)
- Use them to break long Sequence Flow lines and connect parts of a Process across different pages of a print-out



Error Event (and Exception Flow)

Error End and Intermediate Events work in pairs to provide a simple error handling mechanism:

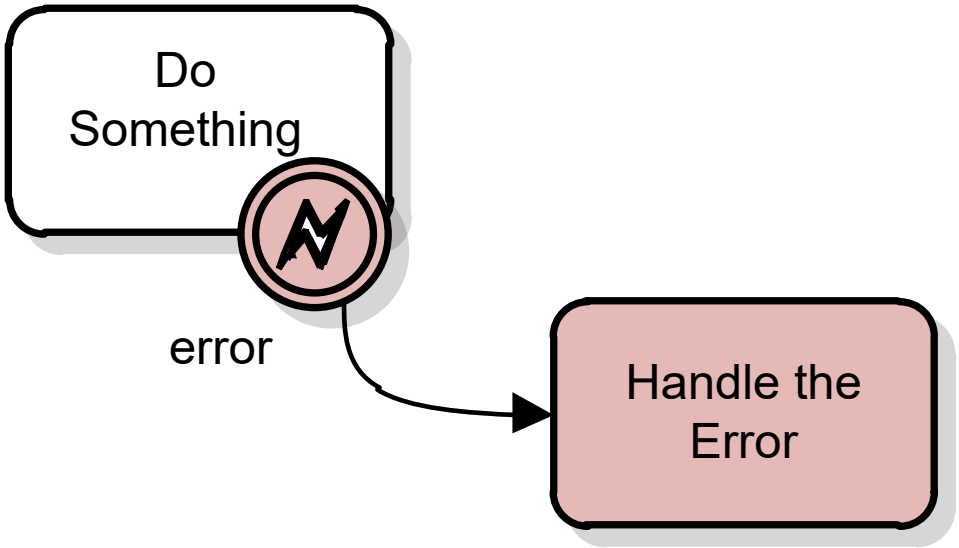
- Use an Error End Event to end an Activity and raise a named error
- Catch the error with an Error Intermediate Event *attached as an interrupt to the Activity boundary*
- This Error Intermediate Event emits a token to another Activity that handles the error

Exception Flow	*Exception flow* occurs outside the *normal flow* of the Process and is based upon an Intermediate Event attached to the boundary of an Activity that occurs during the performance of the Process (see page 287).	

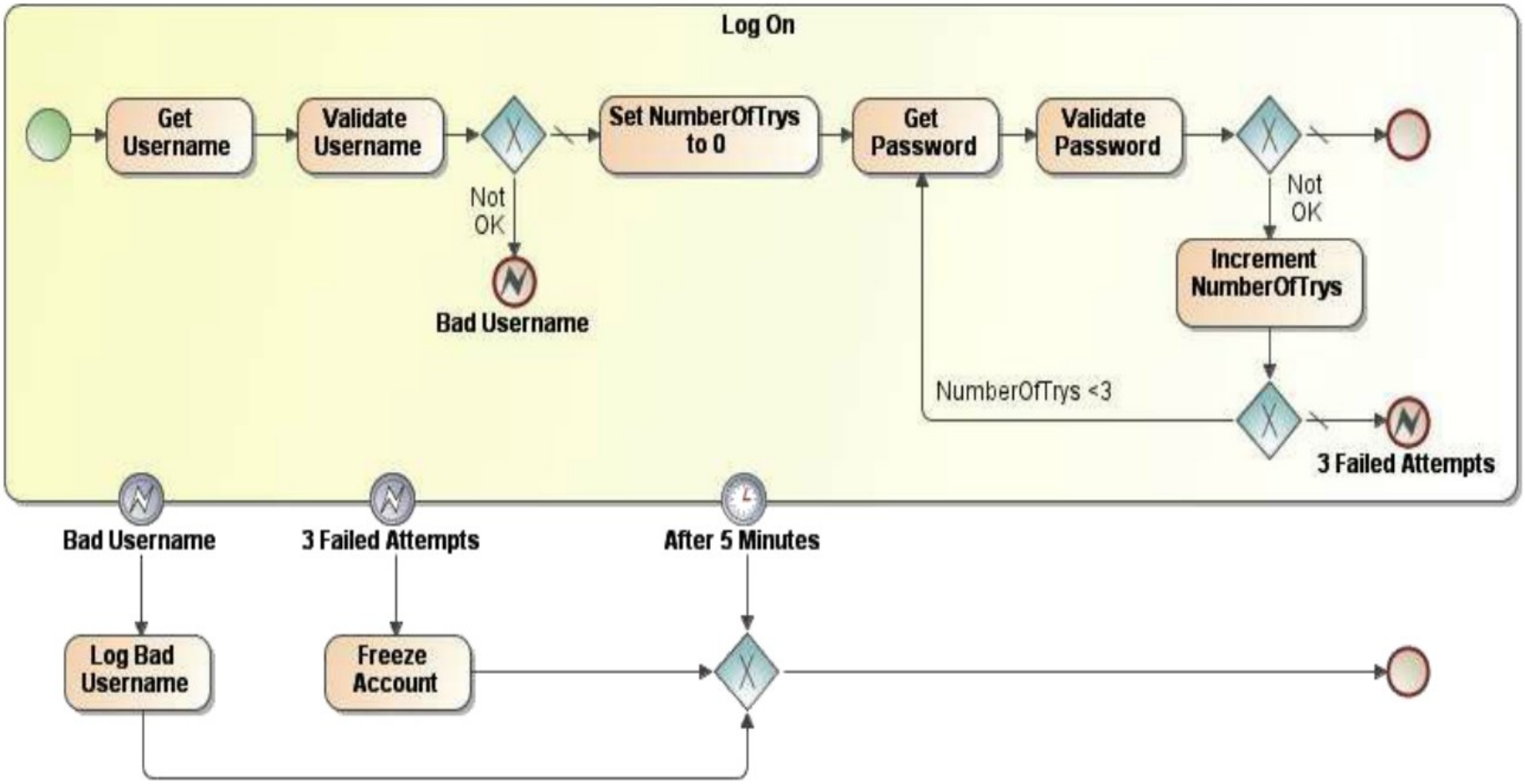
Error Event

Intermediate Errors are always **interrupting** (i.e. after an Error the activity must abort). If an activity intends to report some situation and continue then a non-interrupting Signal can be used.

Figure 10.79 – Error Events



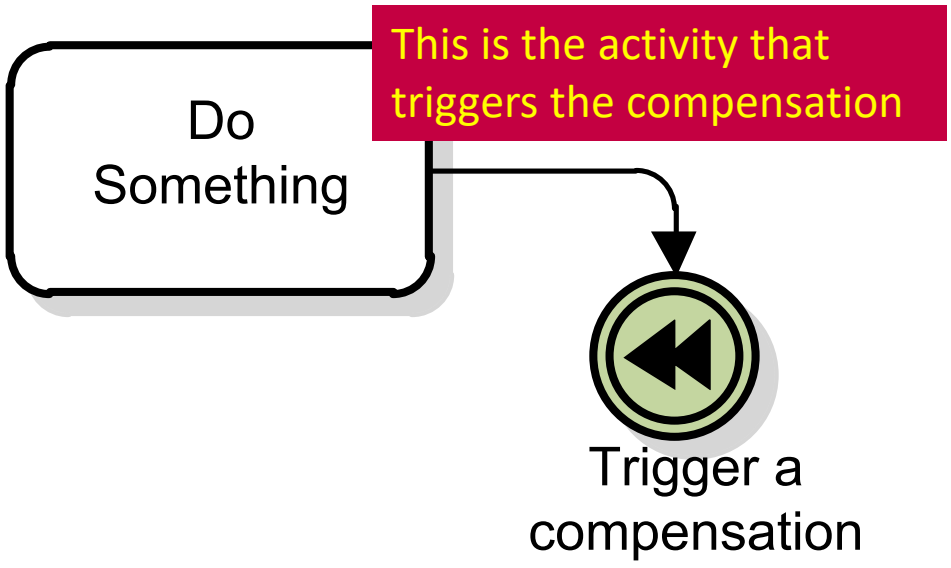
Error Event


Log On
Get Username
Validate Username
Not OK
Bad Username
Set NumberOfTrys to 0
Get Password
Validate Password
Not OK
Increment NumberOfTrys
NumberOfTrys <3
3 Failed Attempts
Bad Username
3 Failed Attempts
After 5 Minutes
Log Bad Username
Freeze Account

Compensation

- Sometimes you need to undo the effects of a business process. You do this by rolling it backwards, one completed Activity at a time
- There are three options for undoing each completed Activity:
  1. Do nothing - there are no data changes to undo
  2. Overwrite - restore all data to its original state
  3. Undo - perform a specific Activity to undo the data changes - this is known as *compensation*

Compensation



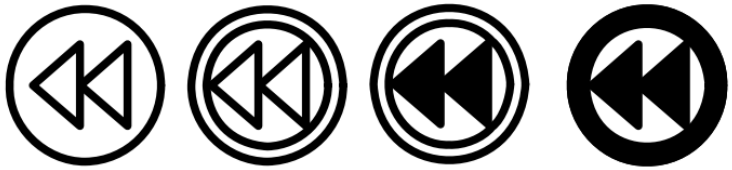
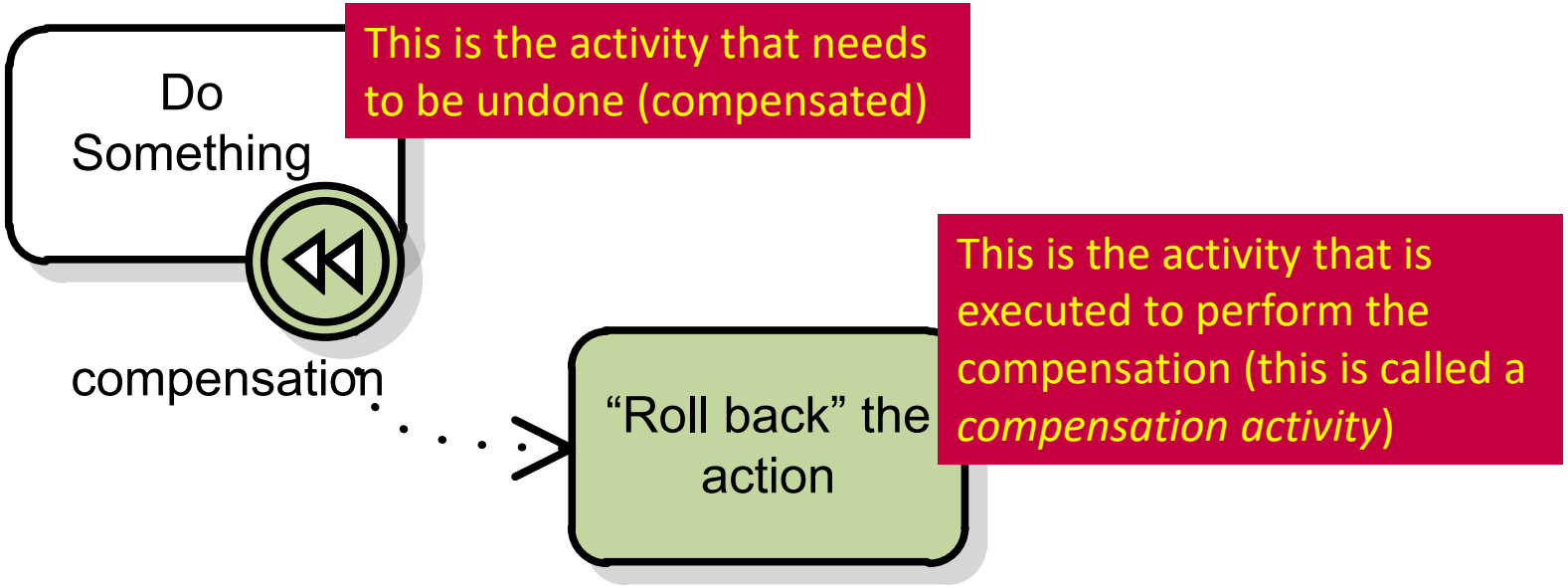
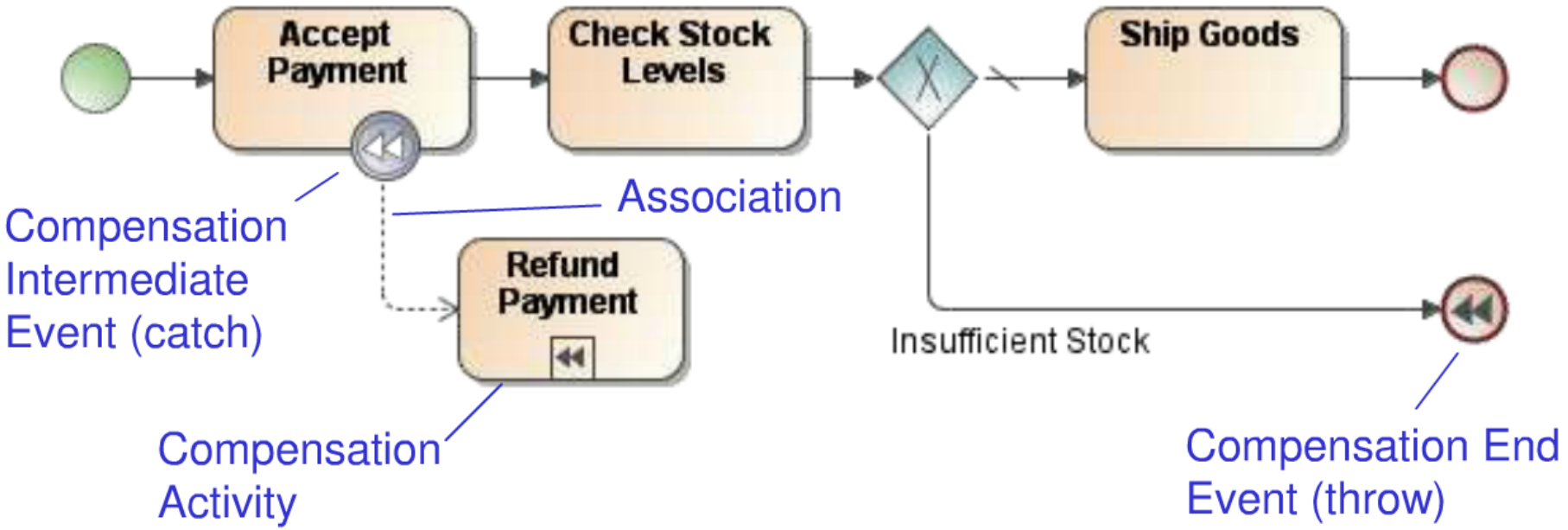


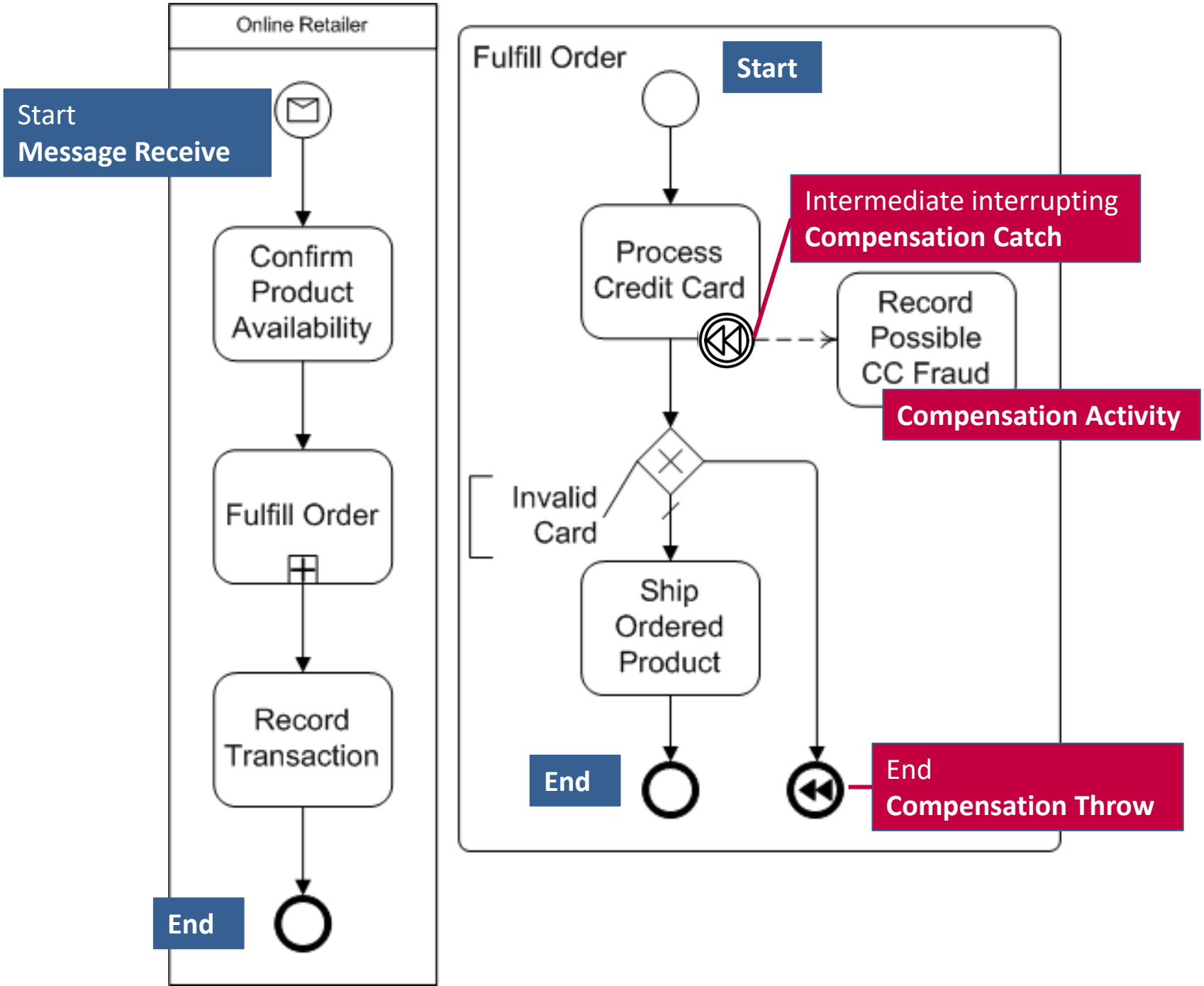
Figure 10.75 – Compensation Events

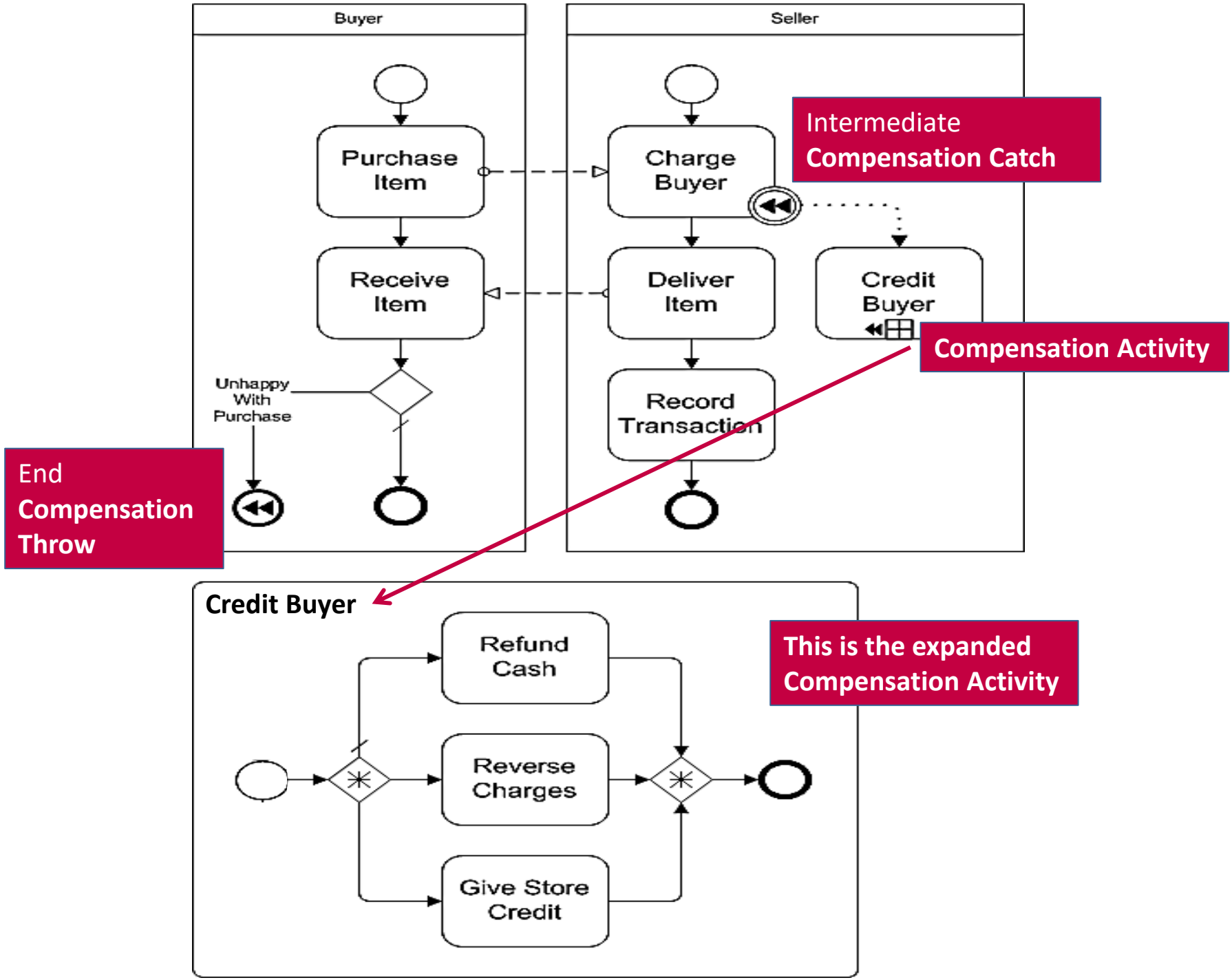


Compensation Example

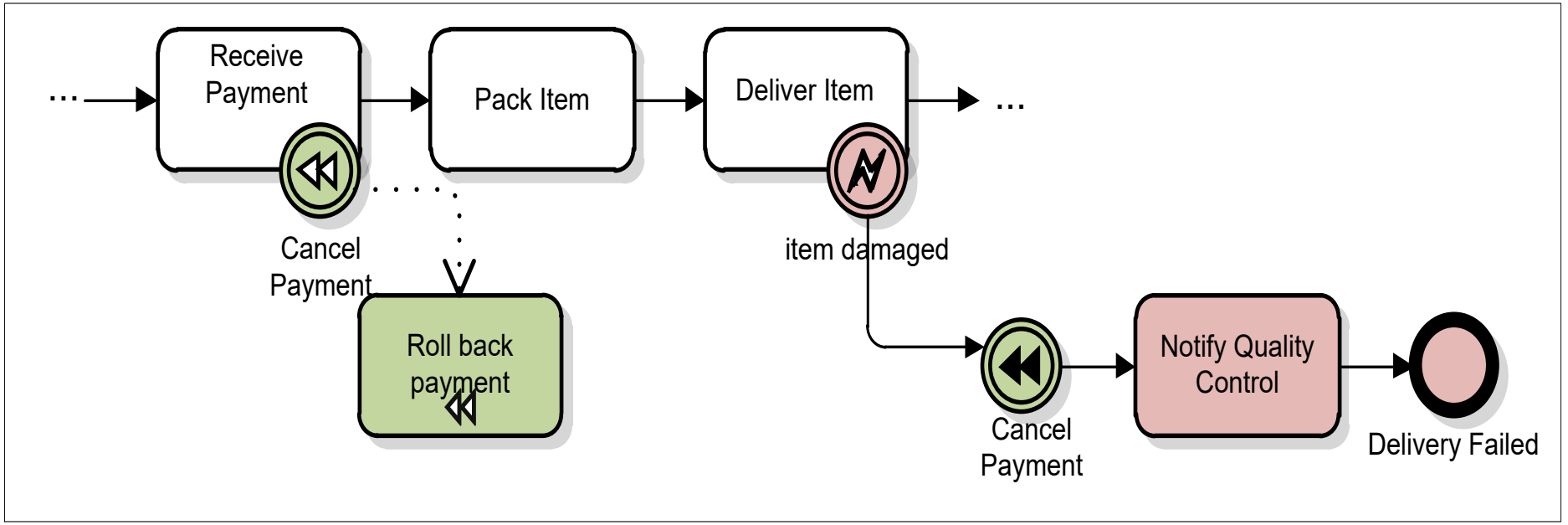
- If the goods are Out Of Stock, the Accept Payment Activity *must* be compensated so that the customer gets their refund!
- If the Compensation Event specifies a compensatable Activity *only* that Activity is compensated. Otherwise, *all* compensatable Activities in the process are compensated



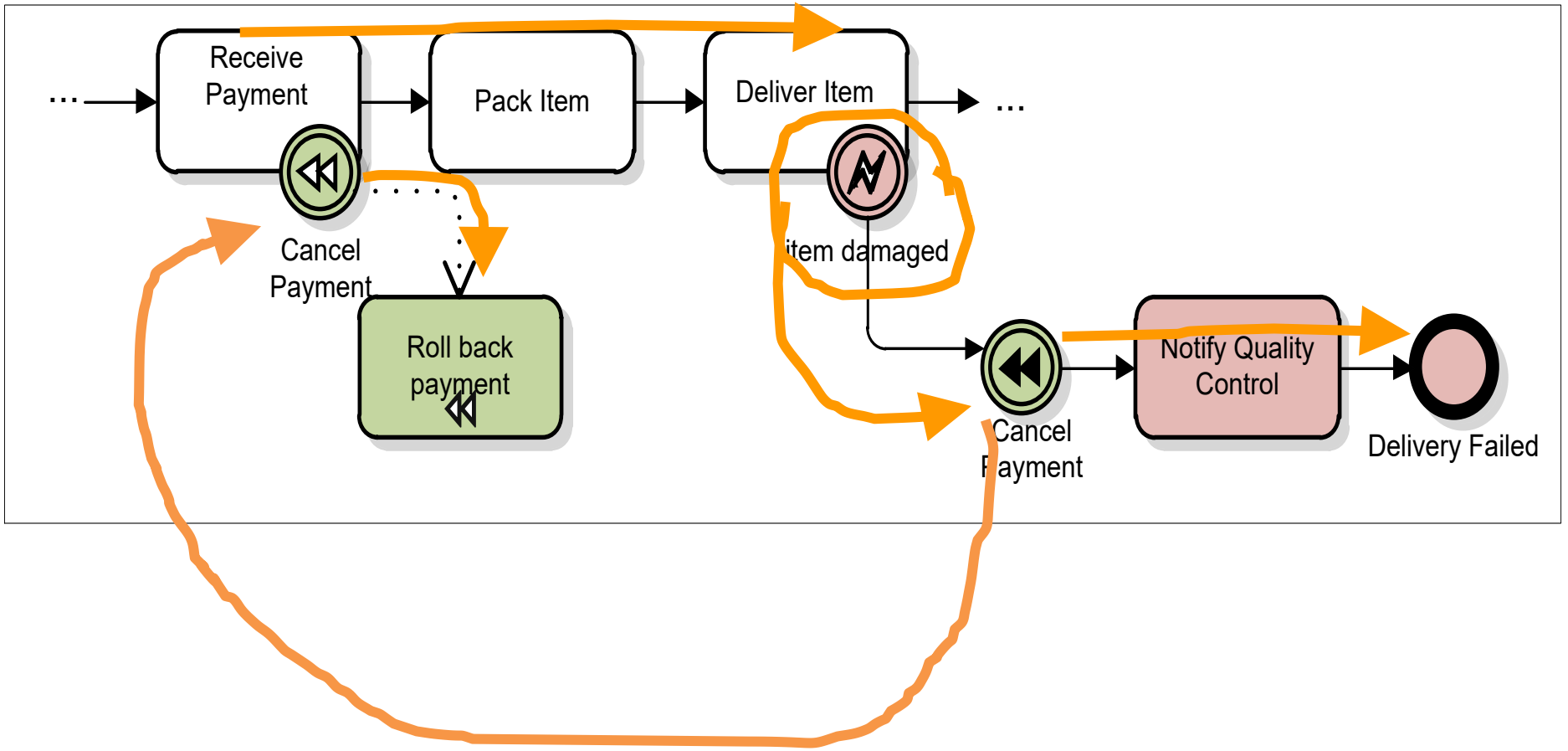

Online Retailer
Start
Message Receive
Confirm Product Availability
Fulfill Order
Record Transaction
End
Fulfill Order
Start
Process Credit Card
Intermediate interrupting
Compensation Catch
Record Possible CC Fraud
Compensation Activity
Invalid Card
Ship Ordered Product
End
End
Compensation Throw


Buyer
Seller
Purchase Item
Charge Buyer
Intermediate
Compensation Catch
Receive Item
Deliver Item
Credit Buyer
Compensation Activity
Unhappy With Purchase
Record Transaction
End
Compensation
Throw
Credit Buyer
Refund Cash
This is the expanded
Compensation Activity
Reverse Charges
Give Store Credit

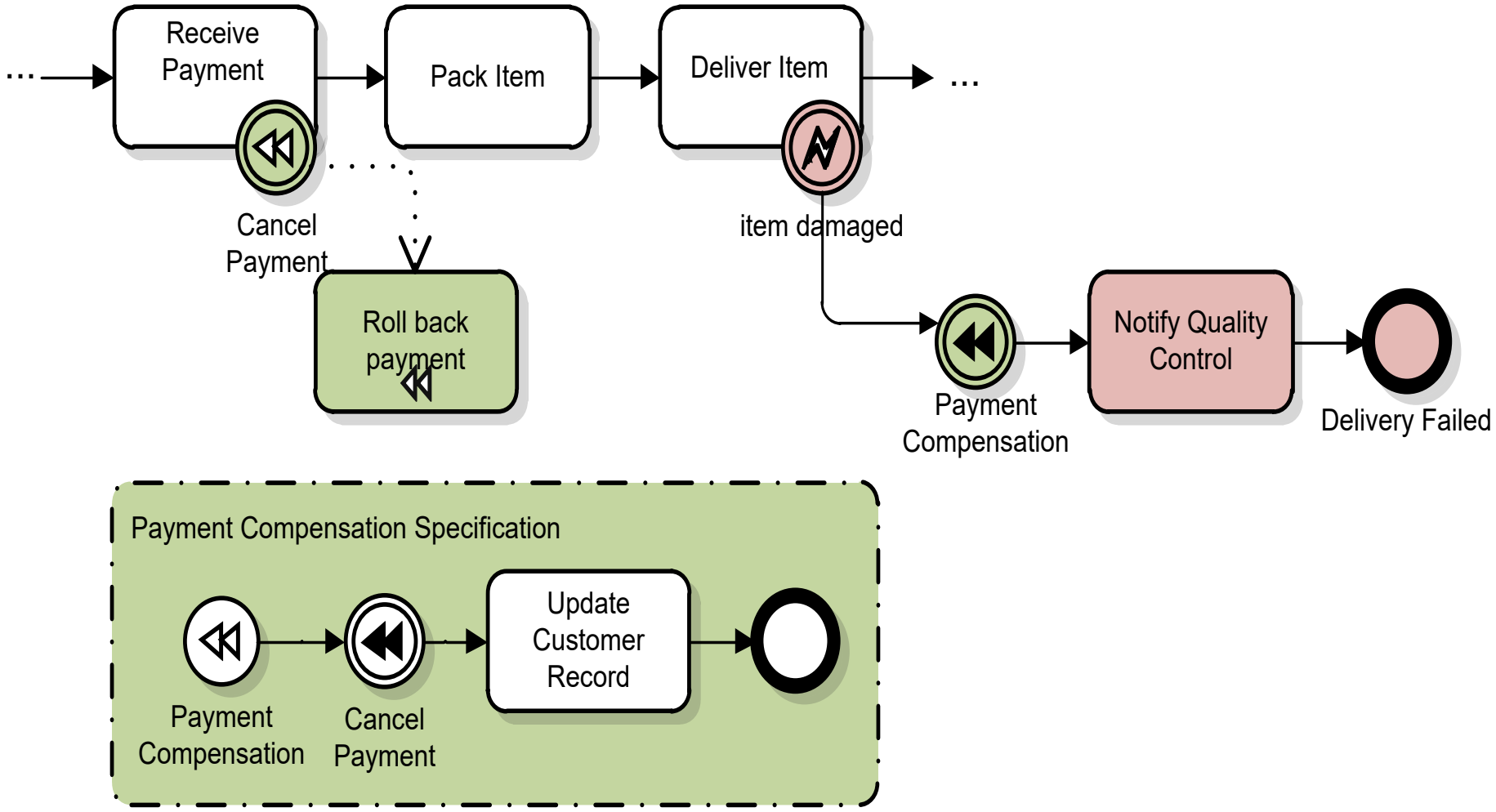
Errors and Compensation



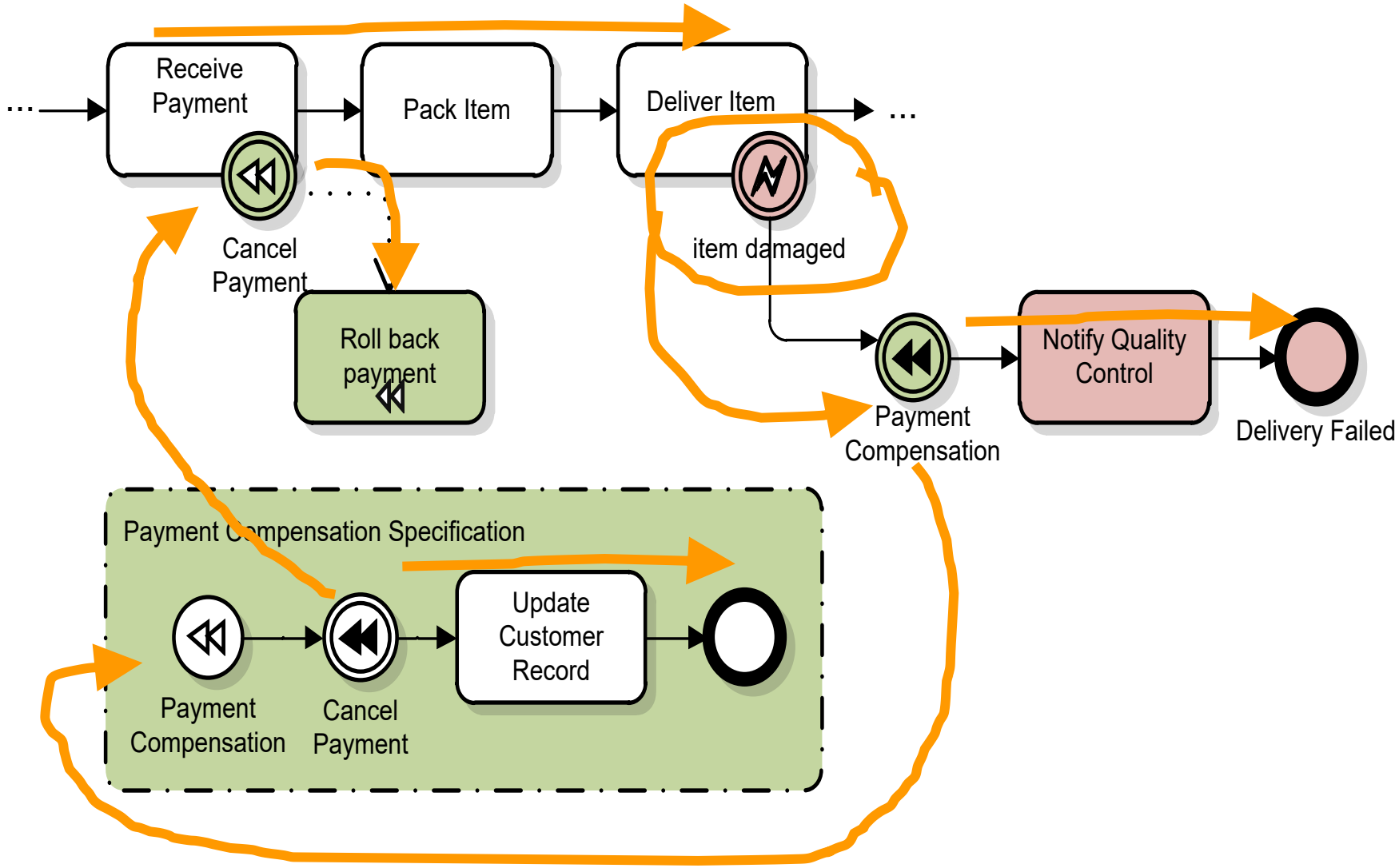
Errors and Compensation



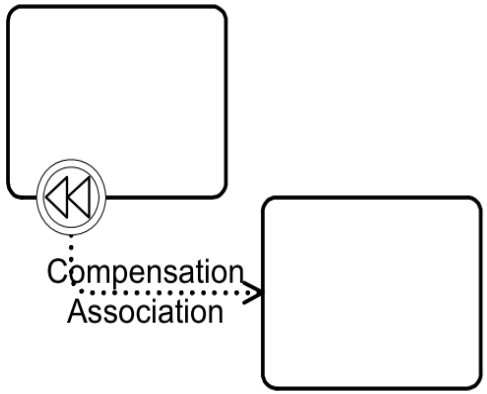
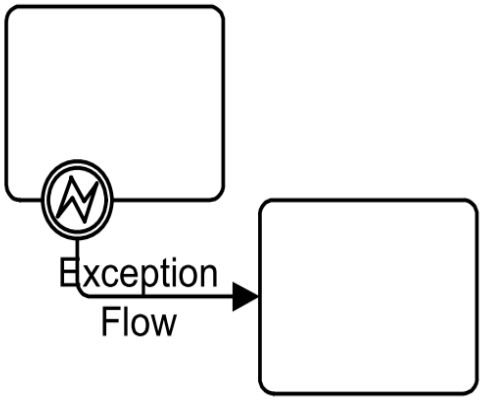
Errors and Compensation



Errors and Compensation



Errors and Compensation

Compensation Association	*Compensation* Association occurs outside the *normal flow* of the Process and is based upon a Compensation Intermediate Event that is triggered through the failure of a *transaction* or a *throw* Compensation Event (see page 302). The target of the Association MUST be marked as a Compensation Activity.	
Exception Flow	*Exception flow* occurs outside the *normal flow* of the Process and is based upon an Intermediate Event attached to the boundary of an Activity that occurs during the performance of the Process (see page 287).	

Errors and Compensation

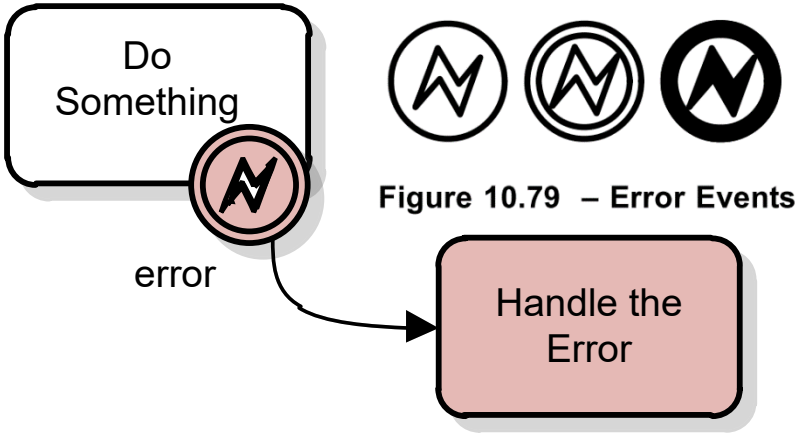
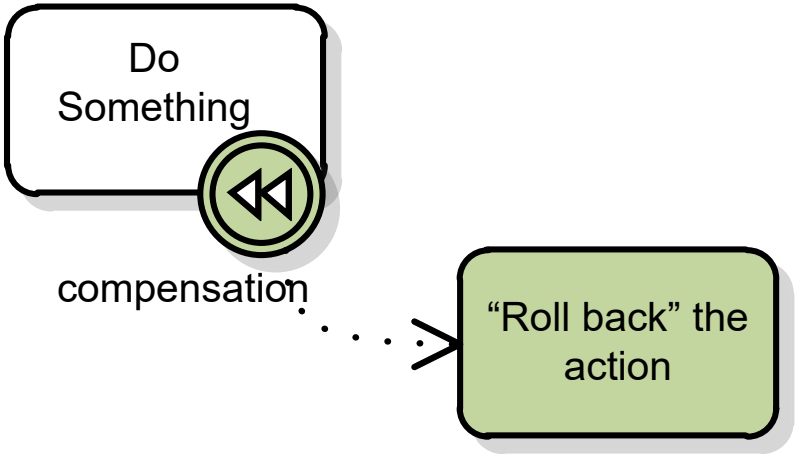


Figure 10.79 – Error Events



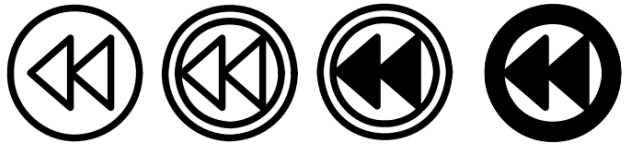
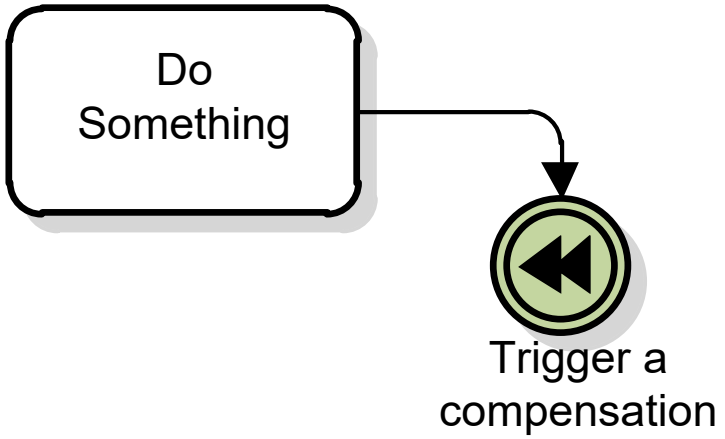


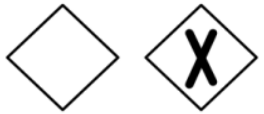
Figure 10.75 – Compensation Events



BPMN Elements: Gateways

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.

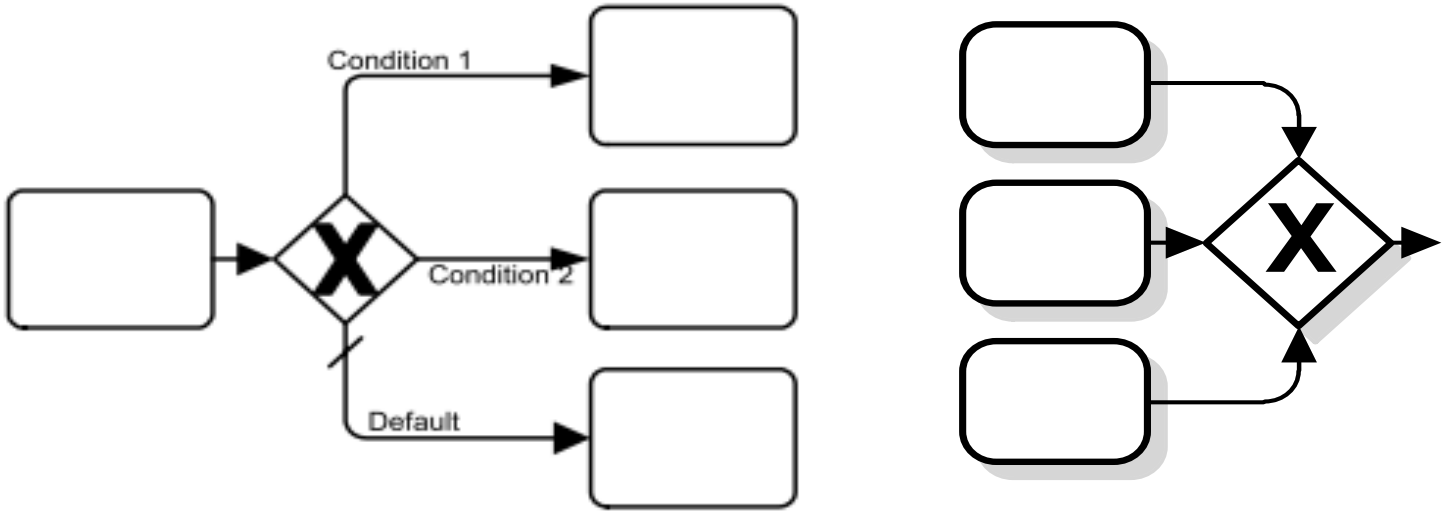
Parallel Gateway

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 Gateway (XOR-split/merge)

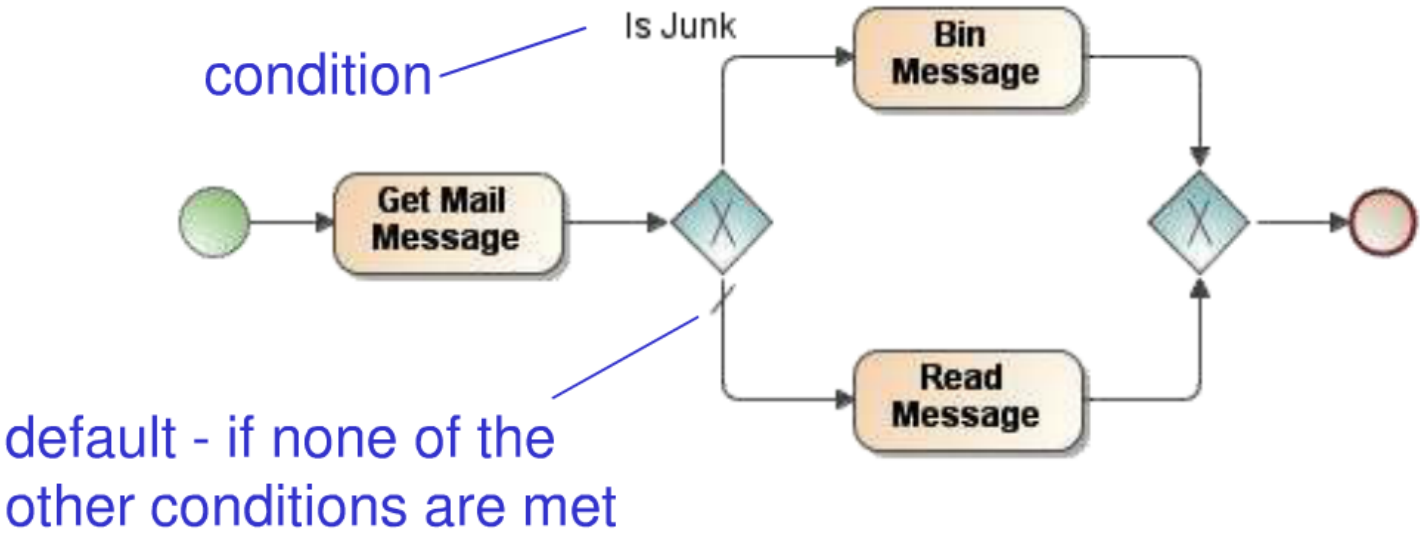


- XOR-split selects one and only one of the 2..* output flows.
- The conditions are evaluated top-to-bottom; the output flow associated to first condition evaluated true is selected.
- Optionally, a *default* flow (--/--) may be included. The default flow is unconditional and is selected whenever all the other conditions are evaluated false.
- The output flows of a XOR-split may be merged using a XOR-merge that continues as soon as one input flow arrives.

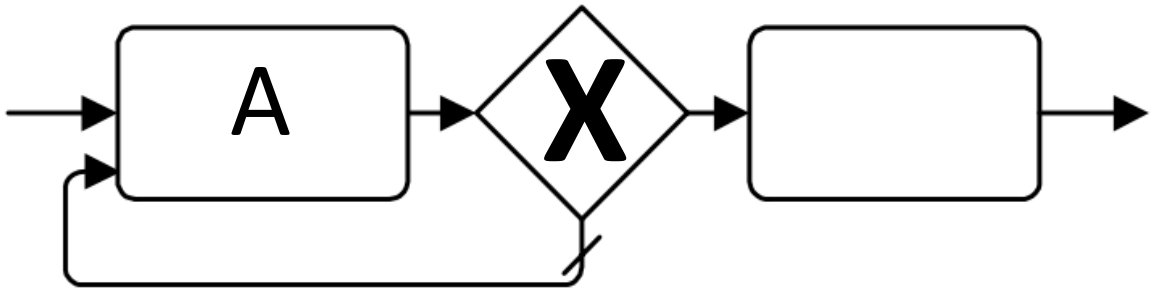
Exclusive Gateway

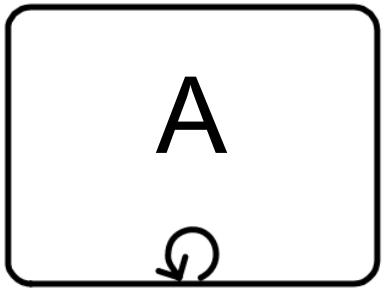
The Exclusive Gateway has one or more input flows, and two or more output flows. When it accepts a token on one of it's input flows it emits a single token on a *single* output flow:

- Each output flow has a *condition* and the set of conditions must be mutually exclusive
- A token is emitted on the output flow whose condition is true

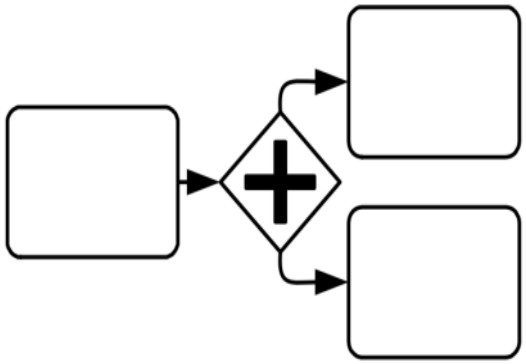


Exclusive Gateway to create a Loop

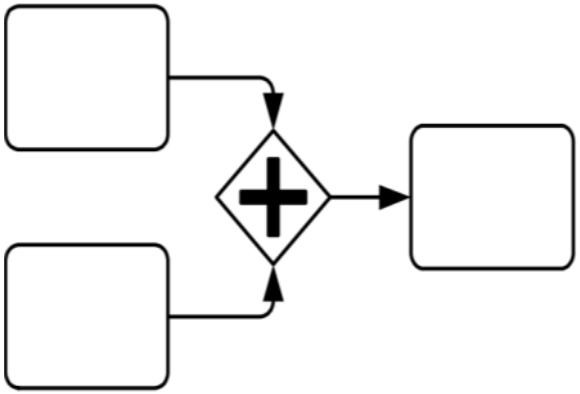




Parallel (AND-split/merge)



AND-split (fork)

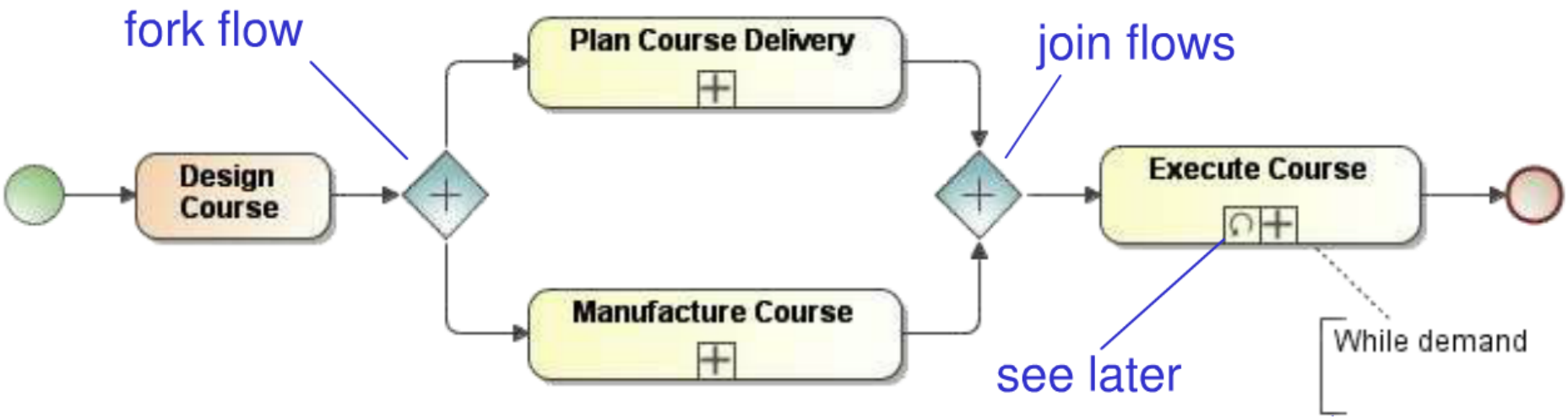


AND-merge(join)

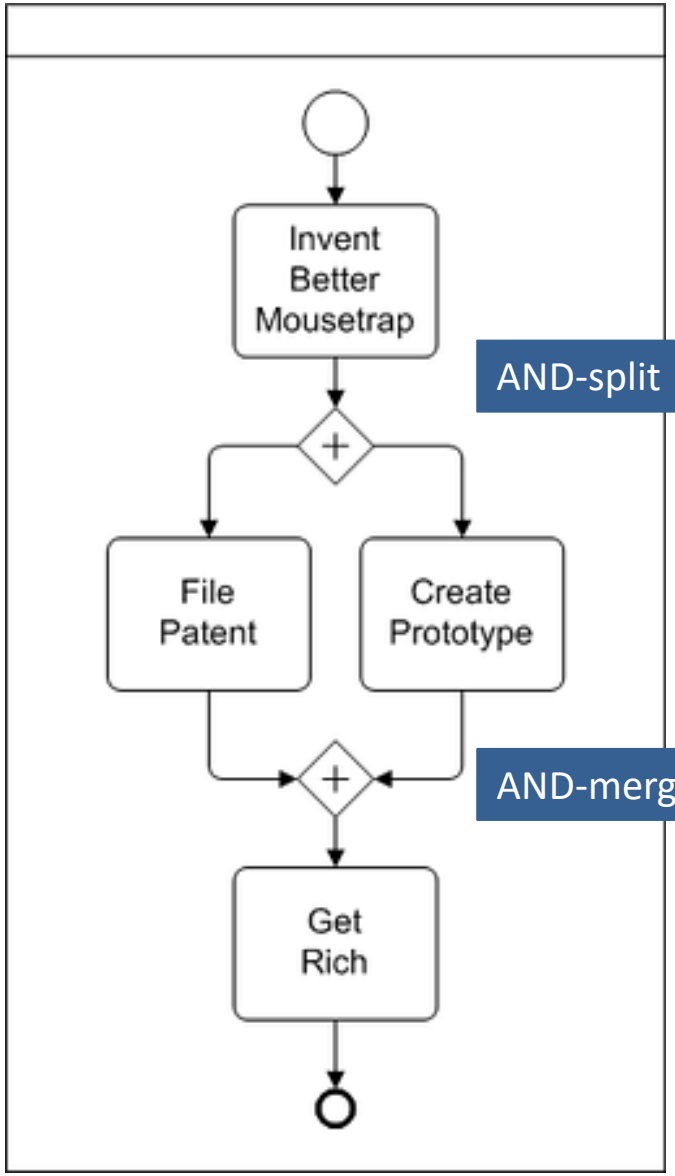
- AND-split forks one input flow into 2..* parallel (concurrent in time) output flows.
- AND-merge waits for *all* input flows before joining them into a single output flow.

Parallel Gateway

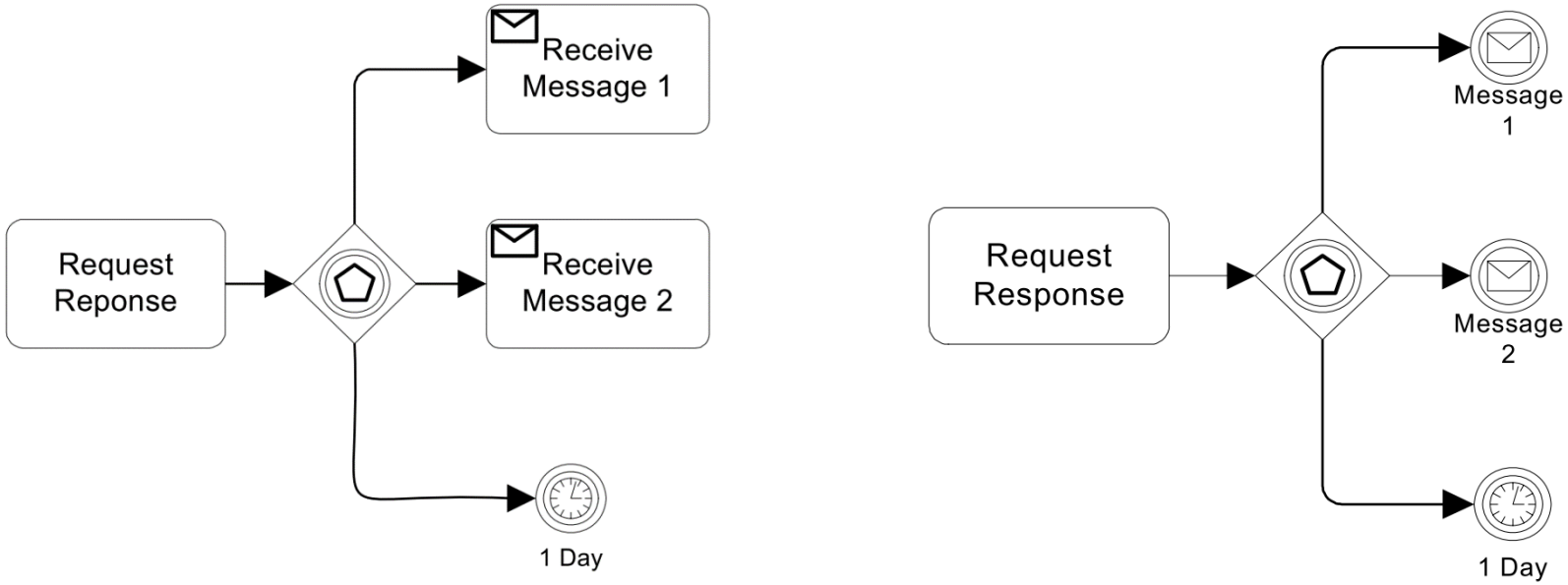
- The Parallel Gateway has one or more input flows, and two or more output flows
- It waits for a token on *all* of it's input flows (join), then emits a token on *all* of it's output flows (fork)



Parallel Gateway



Event-Based Gateway



- The Event-Based Gateway represents a branching point in the Process where the alternative paths that follow the Gateway are based on Events that occur, rather than the evaluation of Expressions using Process data (as with an Exclusive or Inclusive Gateway).
- After the first Event is received its output flow is followed. The remaining paths will be no longer valid (i.e. this is an event-based exclusive gateway).
- An Event-Based Gateway can also start a process if no incoming flows exist.

Event-Based Gateway

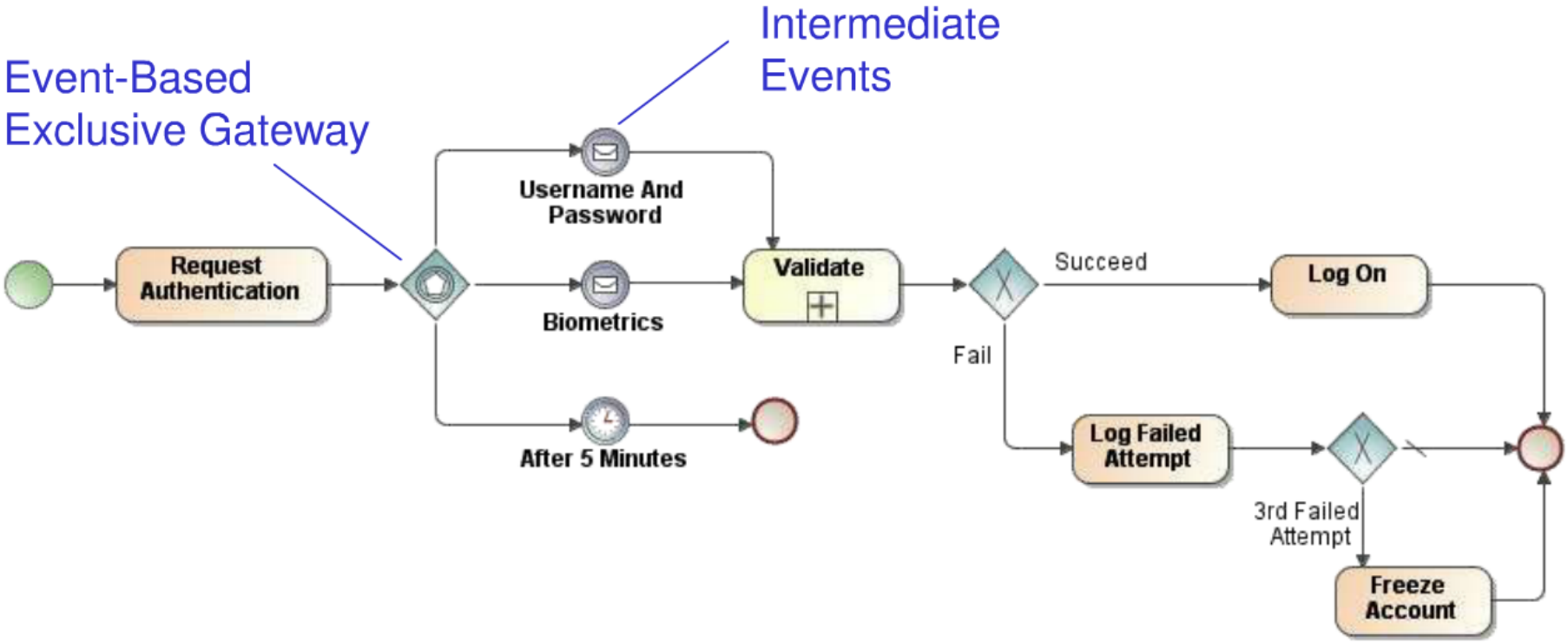
This is a type of Exclusive Gateway where, instead of conditions on its two or more outgoing flows, it has Intermediate Events (see later)

When it accepts a token on one of it's input flows it emits a token on *each* of its output flows:

- Each of these tokens is received by an Intermediate Event which *waits* for its trigger
- Whichever of these Events is triggered *first* "wins", and the token passes through that Event - all the other waiting tokens are consumed

Event-Based Gateway

- This example shows a simple log on process that illustrates the Event-Based Exclusive Gateway (Event Gateway for short)



Event-Based Gateway

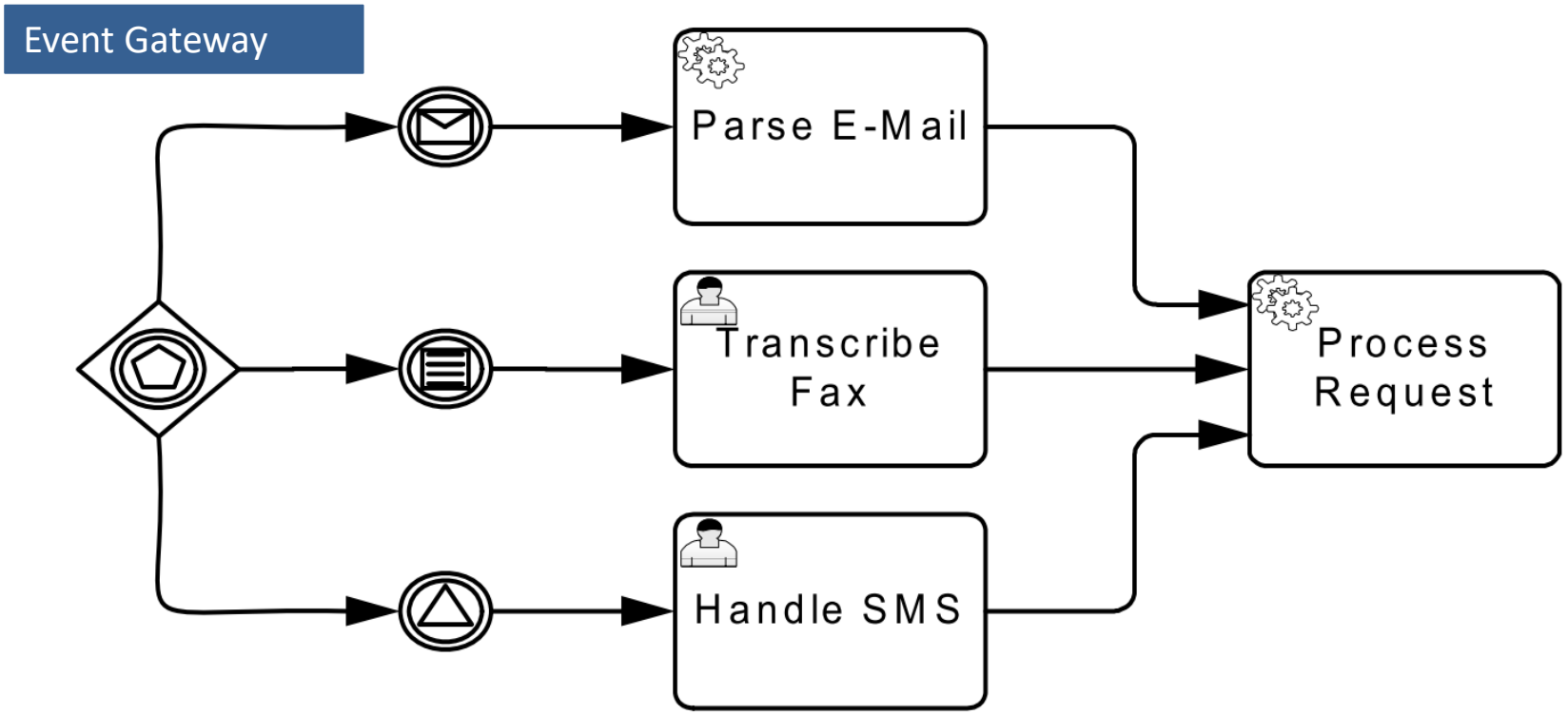
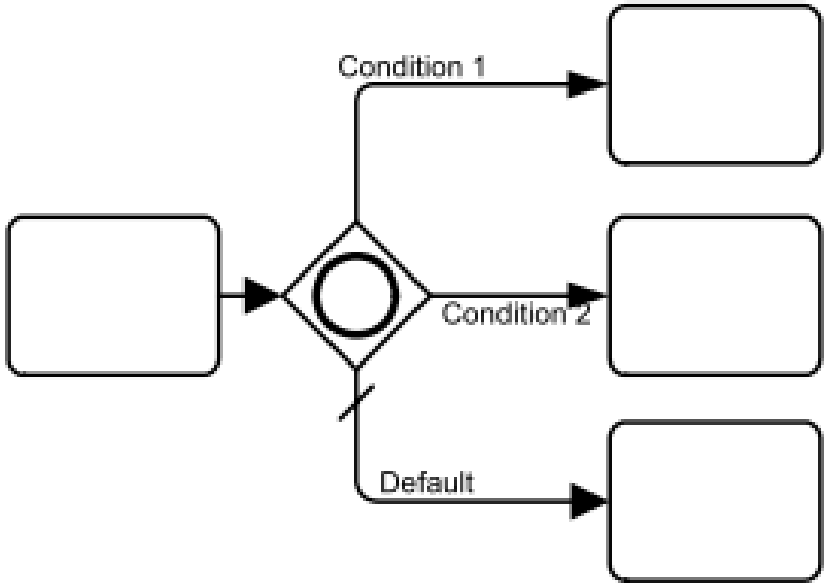


Figure 10.98 - A Process initiated by an Event-Based Gateway

Inclusive Gateway (OR-split/merge)

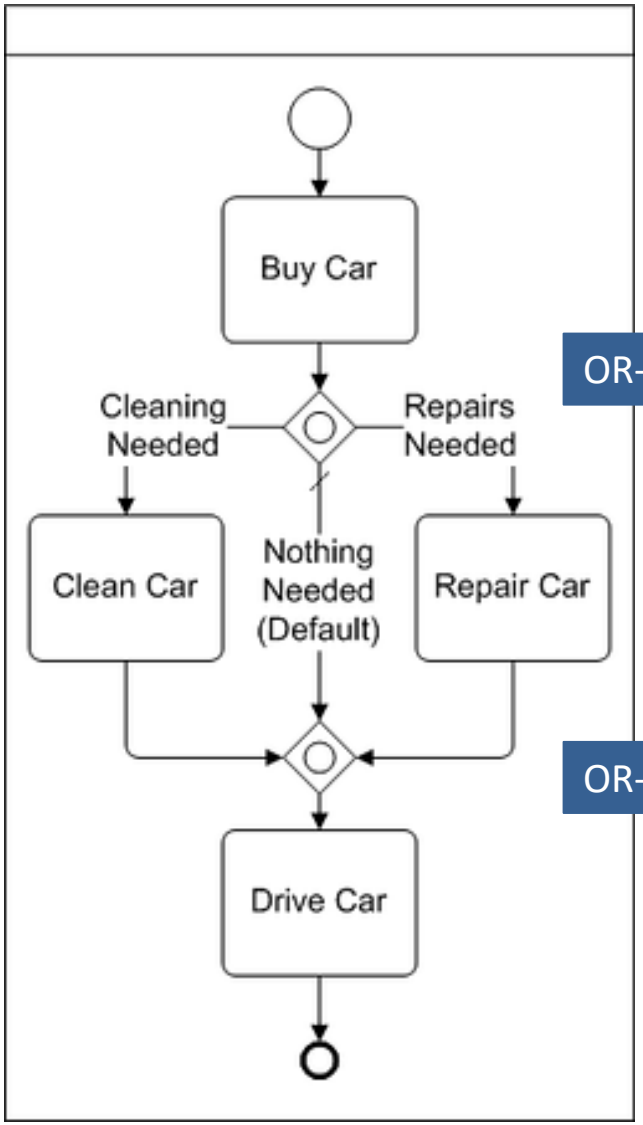


- OR-split selects all of the 0..* output flows where the corresponding condition is evaluated true.
- The result output flows are parallel.
- Optionally, a *default* flow (--/--) may be included. The default flow is unconditional and is selected when all the other conditions are evaluated false.
- OR-merge synchronizes a subset of its input flows.

Inclusive Gateway

- The Inclusive Gateway has a complex splitting and merging behaviour
  - Because of this, Inclusive Gateways are generally only used in pairs with each split being closely followed by a corresponding merge
- Splitting - each output flow has a condition and a token is emitted on *all* the output flows whose conditions are true
  - This is a logical inclusive OR
- Merging - all the tokens generated by a *corresponding* upstream Inclusive Gateway split are synchronized and merged

Inclusive Gateway

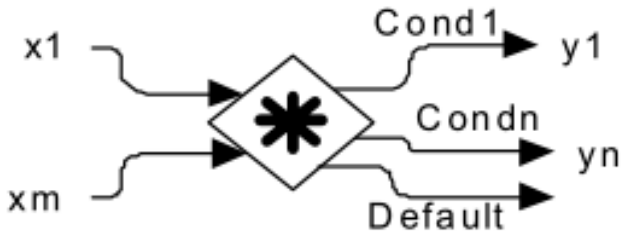


OR-split

OR-merge

Complex Gateway

- These have no default semantics!
  - Splitting - modeller provides an IncomingCondition
  - Merging - modeller provides an OutgoingCondition
- They *must* be supported by Text Annotations that describe their semantics, otherwise the BPD is unreadable!

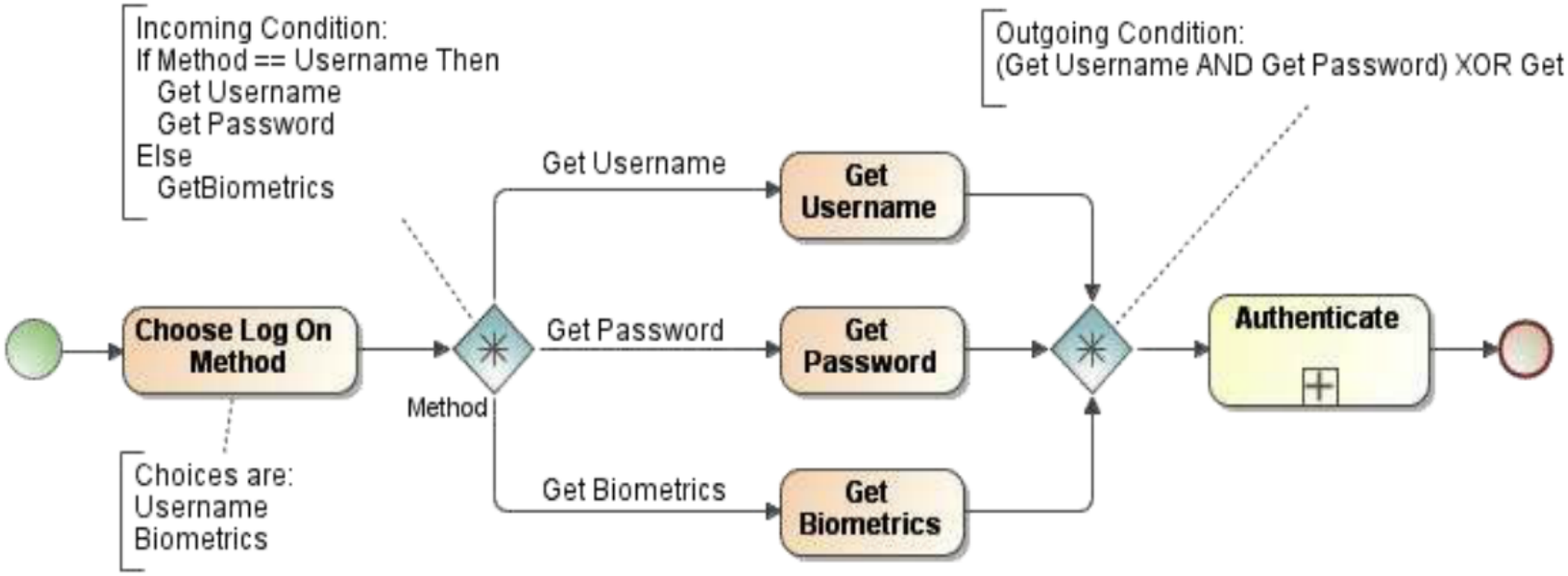


cond1 := x2 < 5
cond2 := x1 > 10 and x2 = 0
…

The **Complex Gateway** can be used to model complex synchronization behavior. An `Expression activationCondition` is used to describe the precise behavior. For example, this `Expression` could specify that *tokens* on three out of five *incoming* **Sequence Flows** are needed to activate the **Gateway**. What *tokens* are produced by the **Gateway** is determined by conditions on the *outgoing* **Sequence Flows** as in the split behavior of the **Inclusive Gateway**. If *tokens* arrive later on the two remaining **Sequence Flows**, those *tokens* cause a reset of the **Gateway** and new *token* can be produced on the *outgoing* **Sequence Flows**. To determine whether it needs to wait for additional *tokens* before it can reset, the **Gateway** uses the synchronization semantics of the **Inclusive Gateway**.

Complex Gateway

- Notice the IncomingCondition on the Complex Gateway split and OutgoingCondition on the Complex Gateway merge



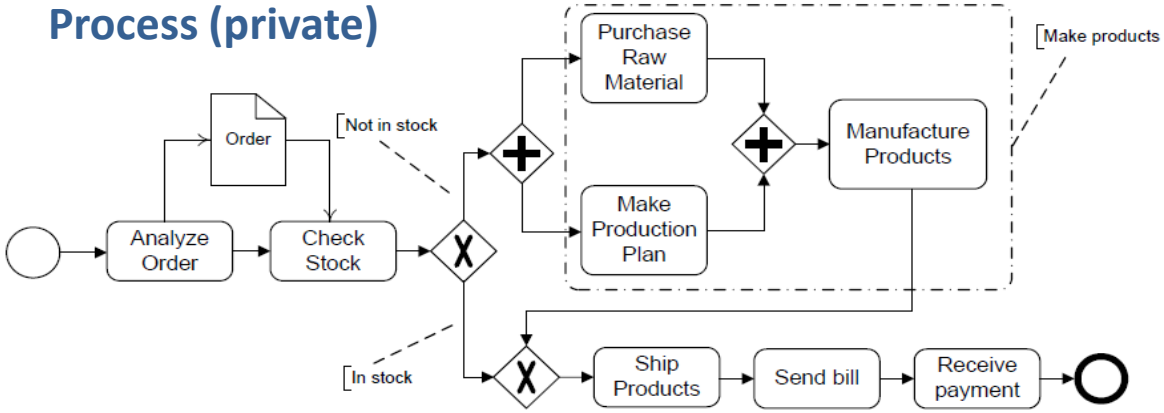
More in detail: Process and Collaboration Diagrams

Types of BPMN Diagrams

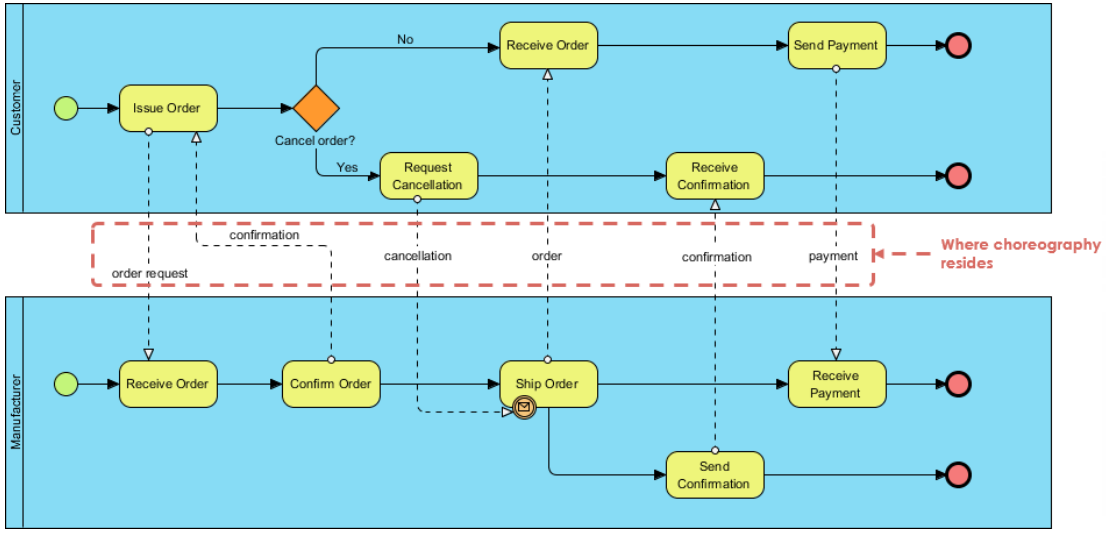
- **Process**
  - Represents the public or private processes of a participant.
  - Focus on representing the (internal) orchestration of a process.
  - The participant can be subdivided into multiple Lanes.
  - All external pools (if any) must be black-box.
- **Collaboration**
  - Represents the message exchange between two or more participants .
  - Focus on representing the orchestration of a process across multiple participants.
- **Choreography not addressed in "AMS")**
  - Represent the information pertaining to each participant in the choreography.
  - The focus is not on orchestrations of the work performed within these Participants, but rather on the exchange of information between Participants.
- **Conversation (not addressed in "AMS")**
  - Conversation diagrams visualize messages exchange between pools.
  - Design workflow with business process diagram and visualize communications with BPMN conversation diagrams.

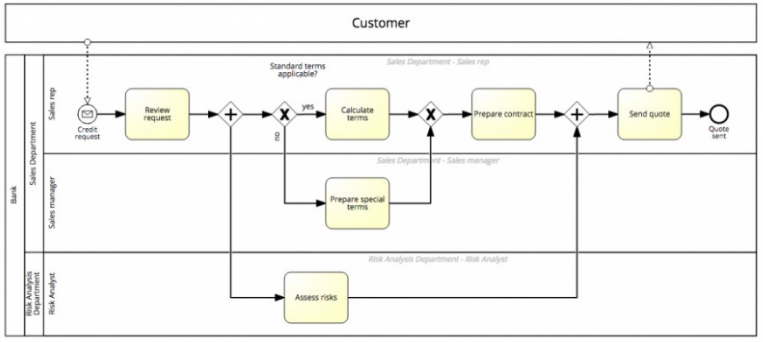
Types of BPMN Diagrams we will address

Process (private)



Collaboration





https://www.visual-paradigm.com/guide/bpmn/bpmn-orchestration-vs-choreography-vs-collaboration/

Pools and Lanes

- **Pool**: represents a process Participant.

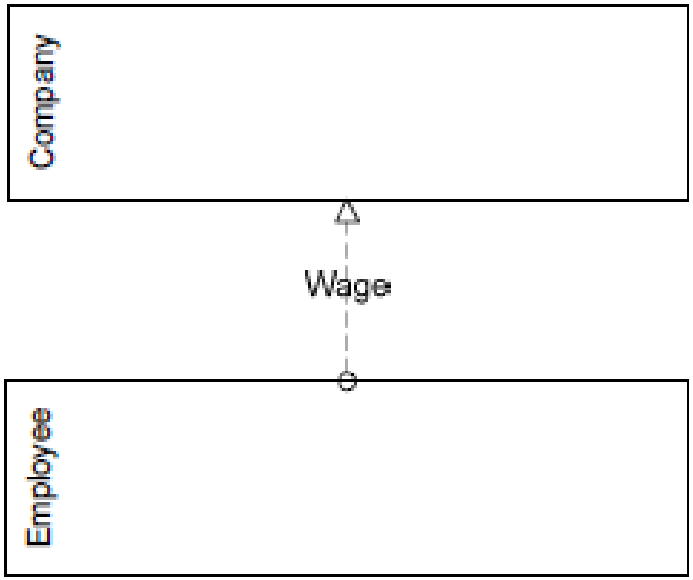
  Sequence flows cannot cross Pool boundaries

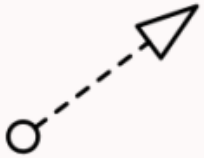
  Message flows can cross Pool boundaries

- **Lane**: a sub-division of a Pool. Used to organize and categorize the activities of a Participant.

  Sequence flows can cross Lane boundaries

Message Flow

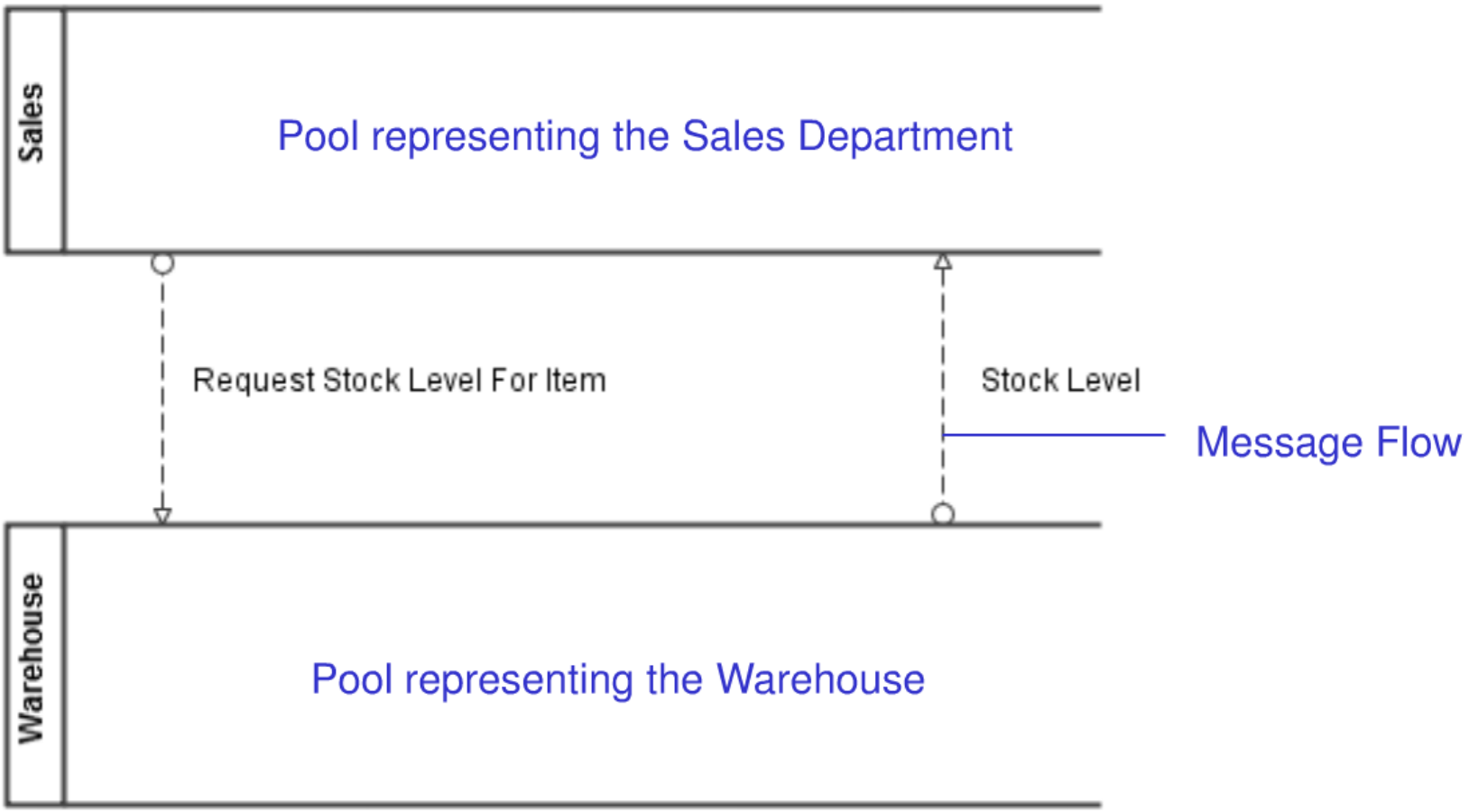




Message Flow symbolizes information flow across organizational boundaries. Message flow can be attached to pools, activities, or message events.

Message Flow

Use Pools in conjunction with Message Flows to show communication between two or more business participants:



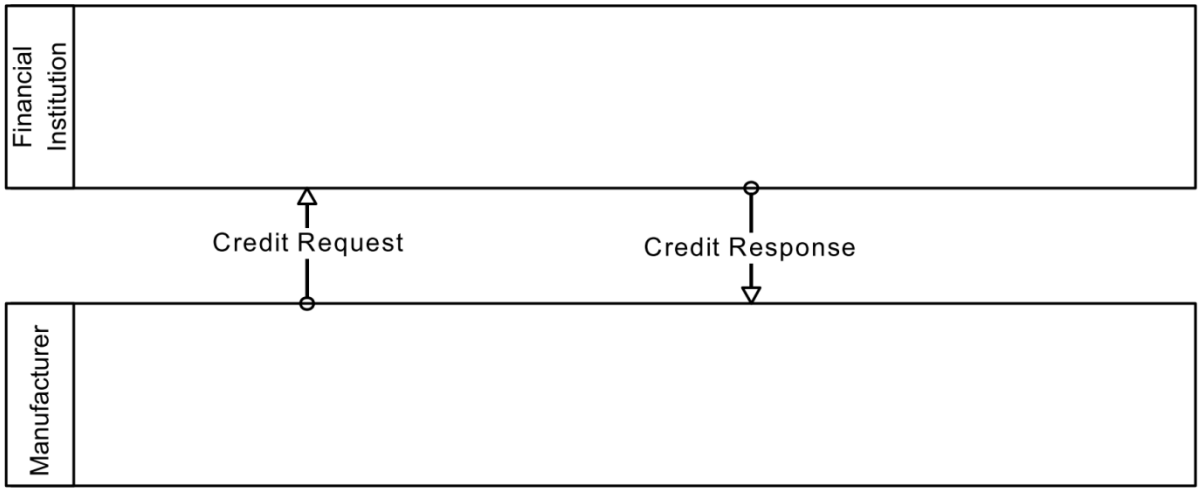


Figure 9.3 - Message Flows connecting to the boundaries of two Pools

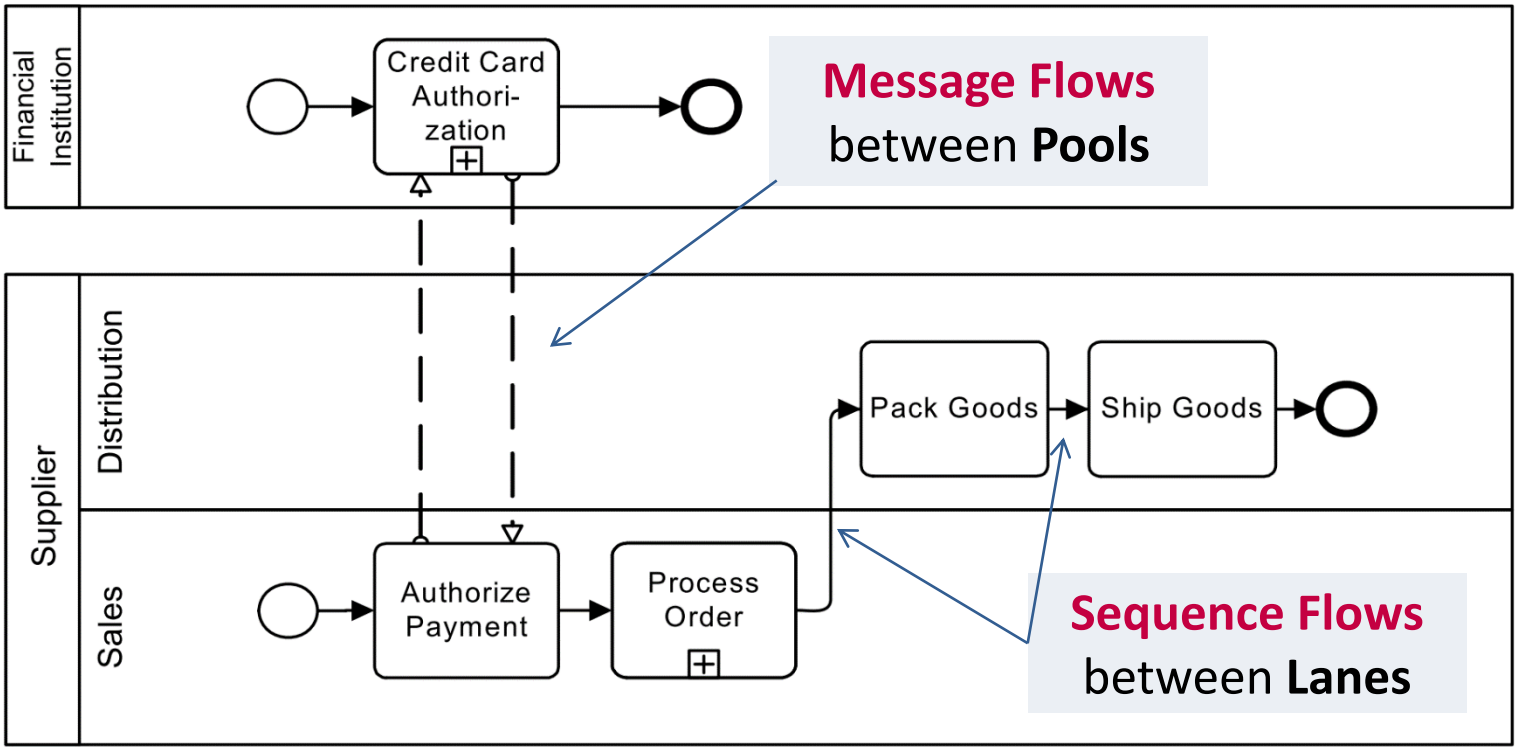
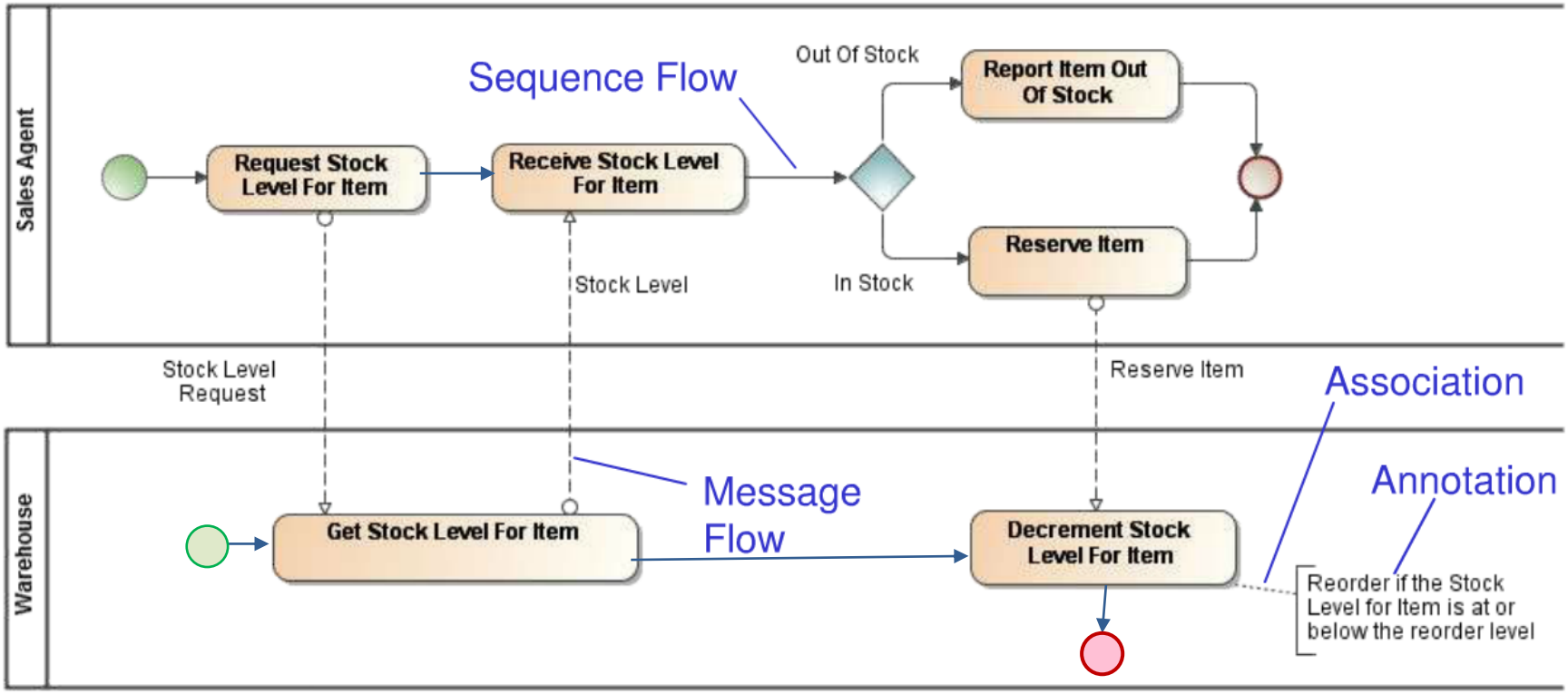
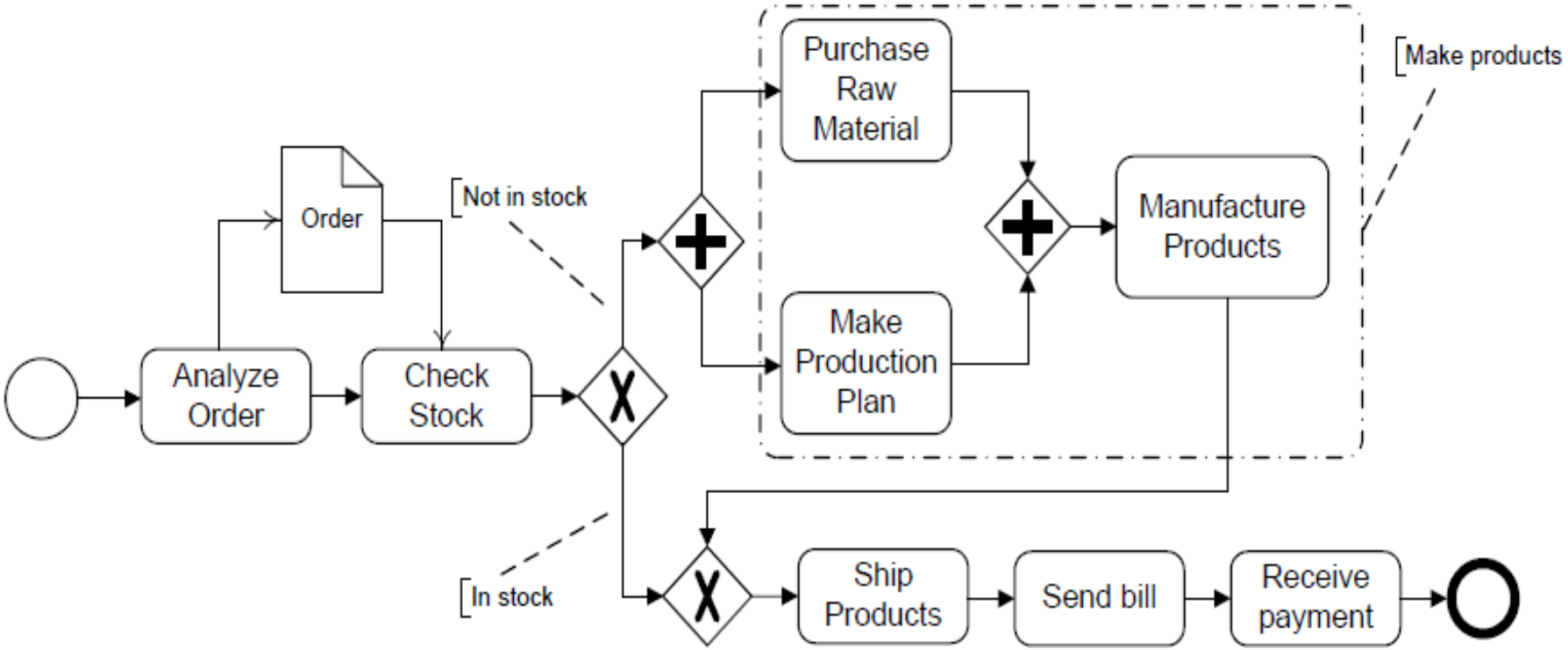


Figure 9.4 - Message Flows connecting to Flow Objects within two Pools

Collaboration Diagram Example



Private Process Diagram (without pools)



- A **Process** focuses on a single **Participant**.
- A **Private Process** focuses on a **Participant** internal to the organization.

Private Process Diagram (one pool)

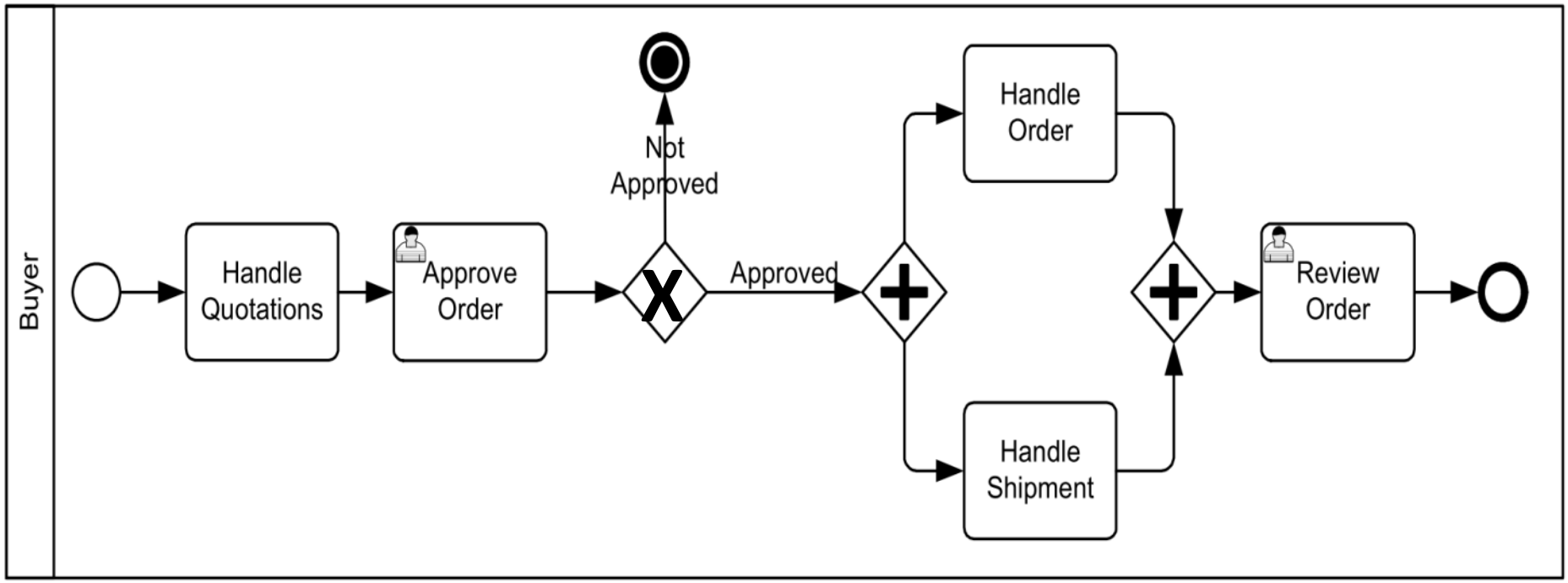


Figure 10.24 - Procurement Process Example

Private Process with one Pool and nested Lanes

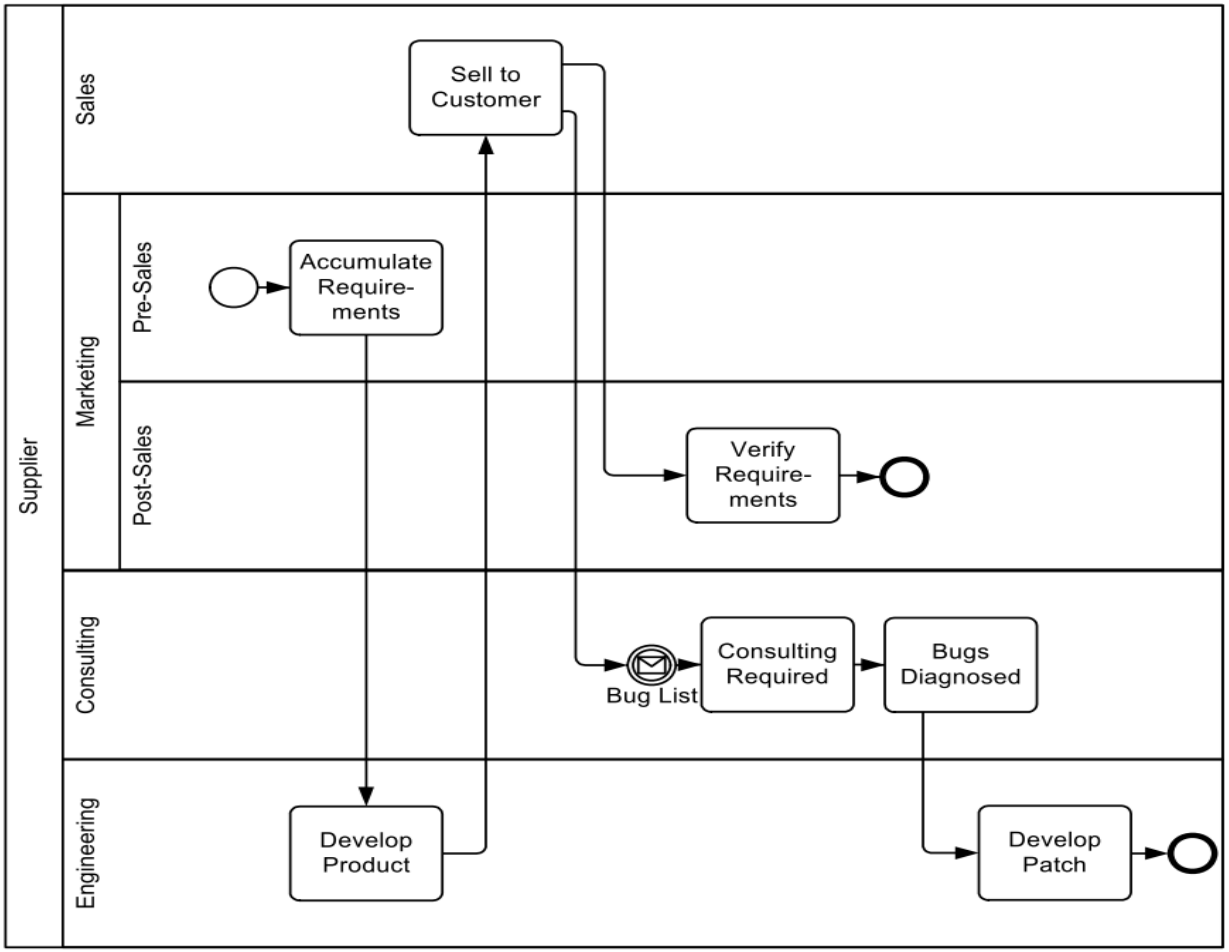


Figure 10.125 - An Example of Nested Lanes

- This **Process** focuses on a single **Participant** with multiple Lanes. Therefore it is a **private Process.**

Public Process

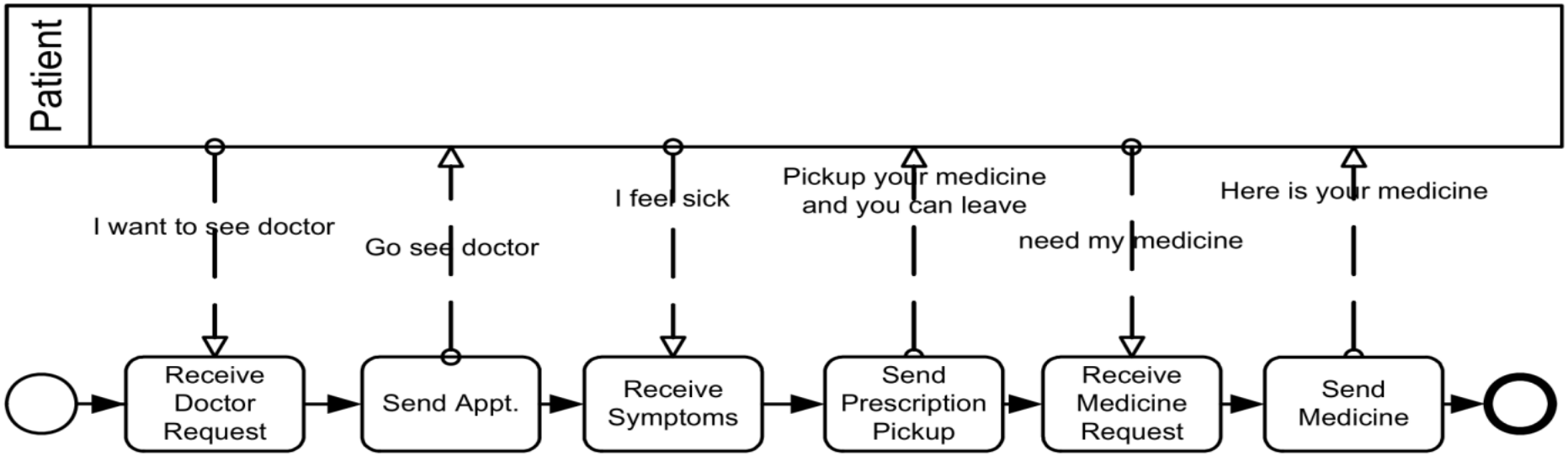


Figure 7.2 - Example of a *public* Process

- A **Public Process** focuses on the interaction between different **Participants** and/or **Processes**.
- Only those Activities that are used to communicate to the other Participant(s) are included. All other "internal" Activities of the private Business Process are not shown in the public Process. Thus, the public Process shows to the outside world the Message Flows and the order of those Message Flows that are needed to interact with that Process. Public Processes can be modelled separately or within a Collaboration to show the flow of Messages between the public Process Activities and other Participants.

Public vs. Private Process

Figure 7.1 - Example of a *private* Business Process

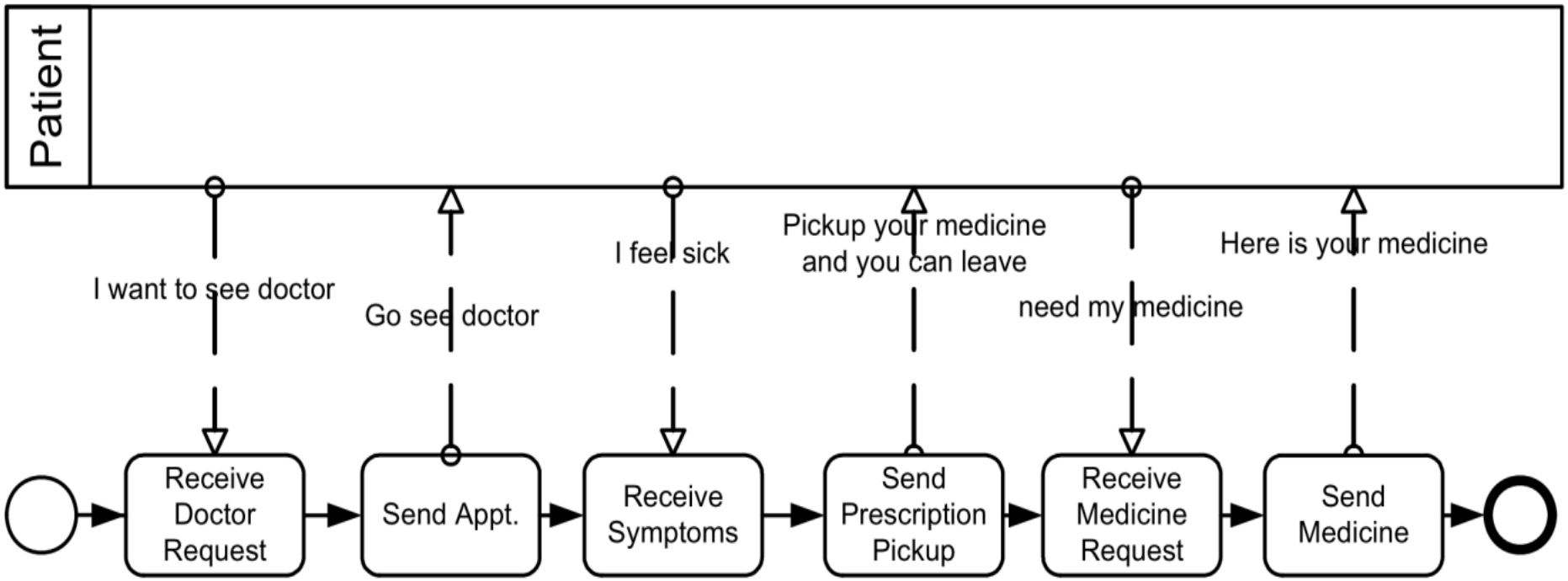


Figure 7.2 - Example of a *public* Process

Collaboration

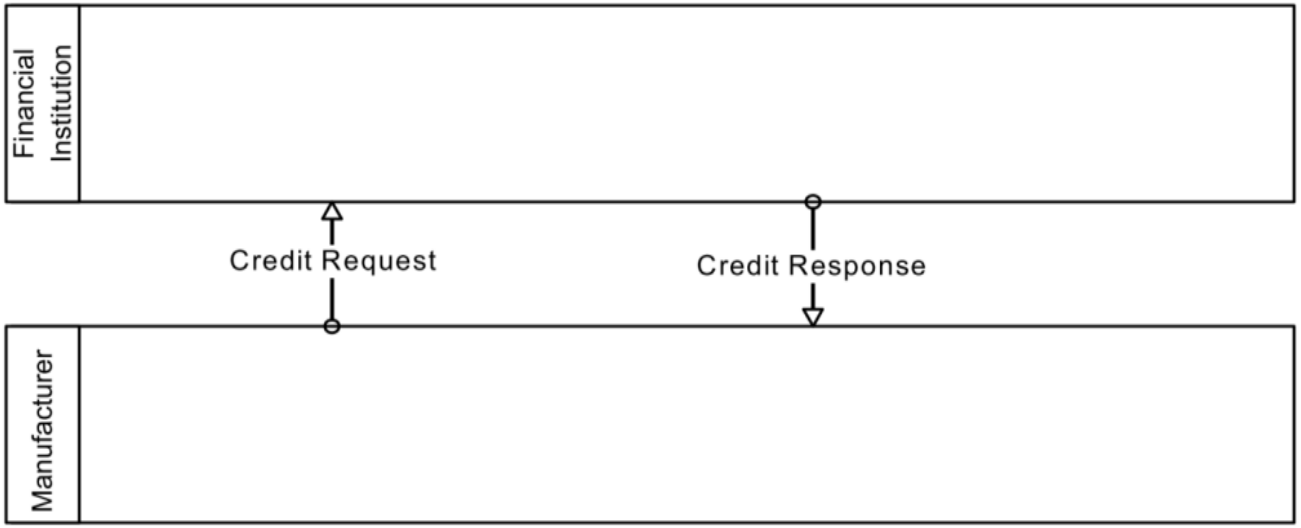


Figure 9.3 - Message Flows connecting to the boundaries of two Pools

- **Collaboration** depicts the interactions between two or more business entities. A Collaboration usually contains two or more Pools, representing the Participants in the Collaboration.
- The Message exchange between the Participants is shown by a Message Flow that connects two Pools. The Messages associated with the Message Flows can also be shown. The Collaboration can be shown as two or more public Processes communicating with each other. With a public Process, the Activities for the Collaboration participants can be considered the “touch-points” between the participants. The corresponding internal (executable) Processes are likely to have much more Activity and detail than what is shown in the public Processes.

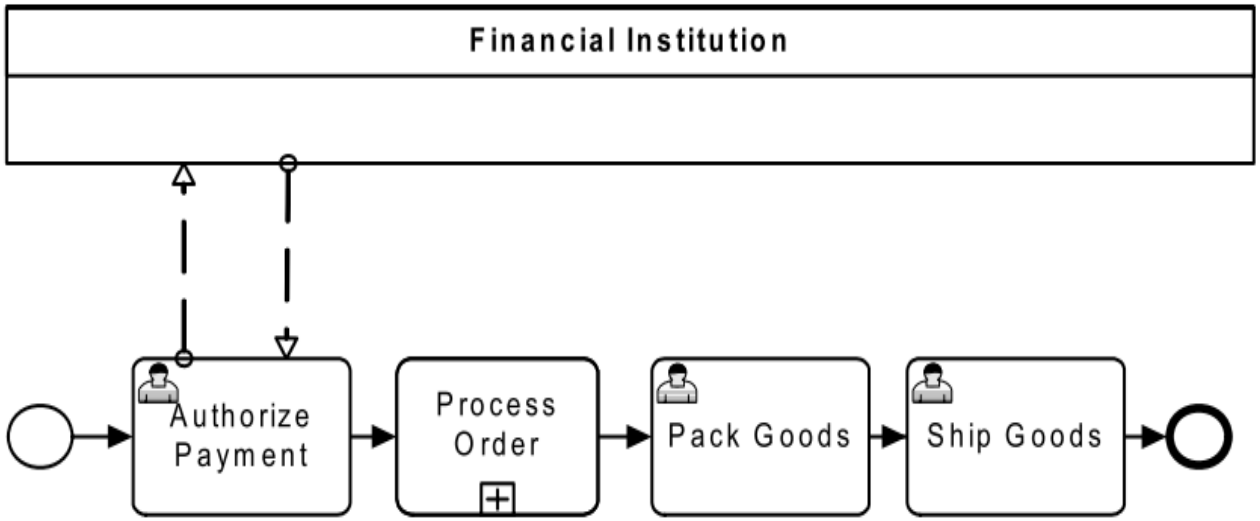


Figure 9.5 - Main (Internal) Pool without boundaries

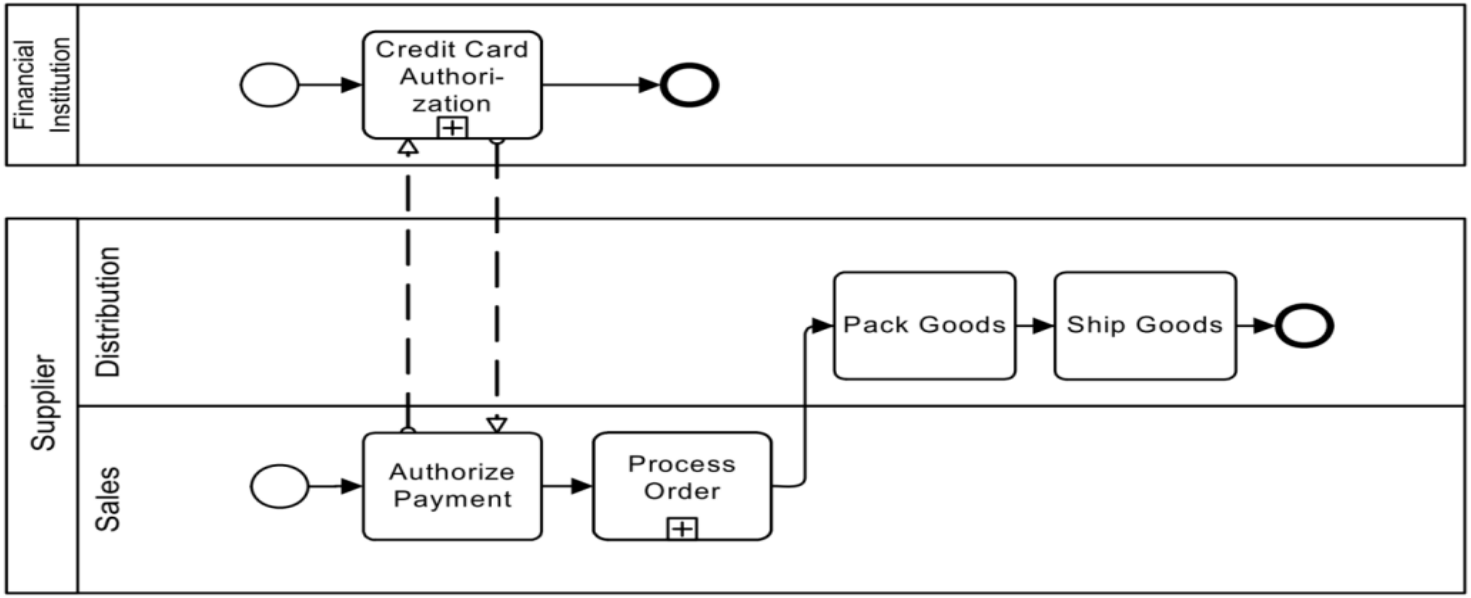


Figure 9.4 - Message Flows connecting to Flow Objects within two Pools

Collaborative Process

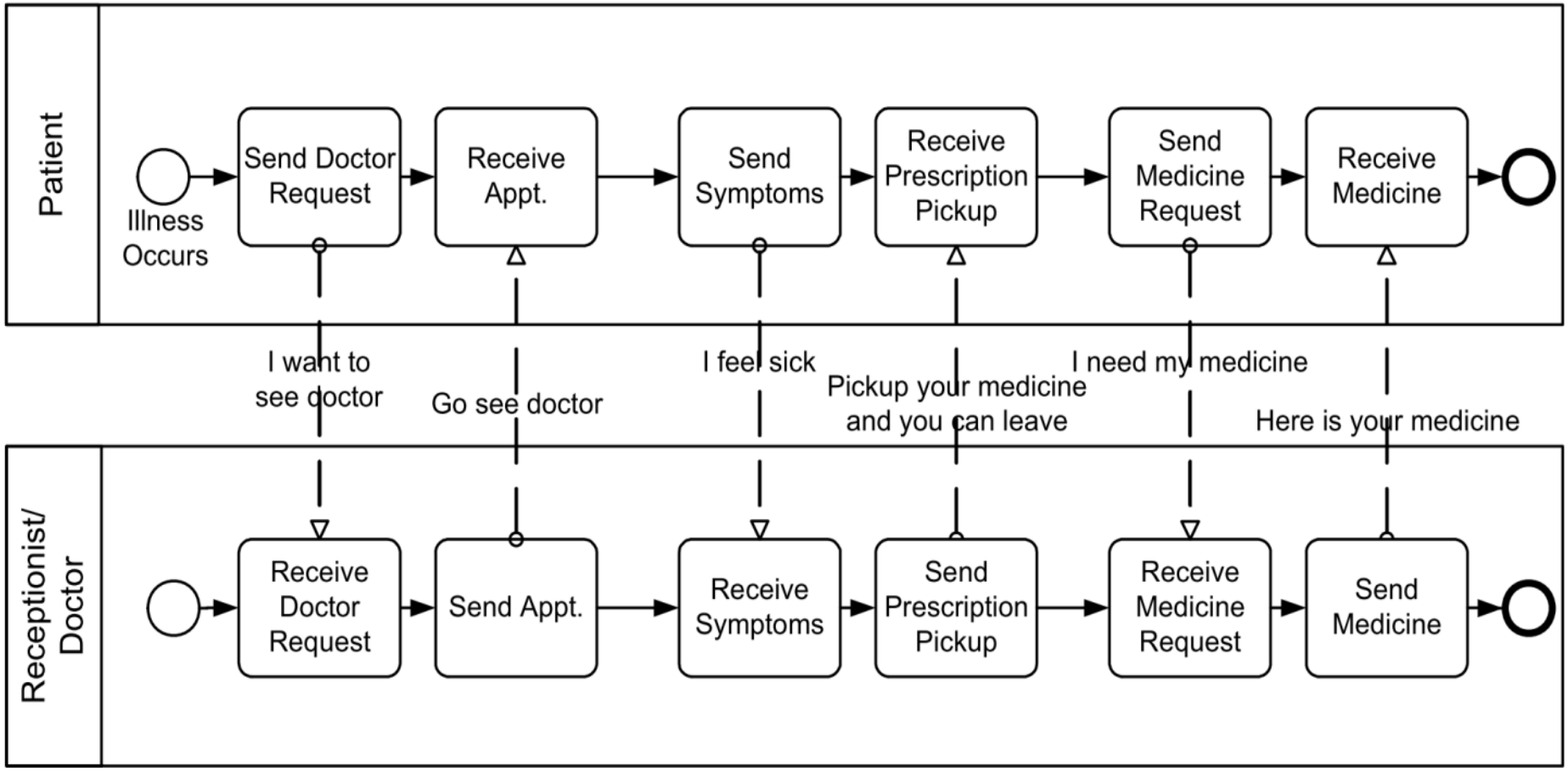


Figure 7.3 - An example of a Collaborative Process