Business Processes Modelling

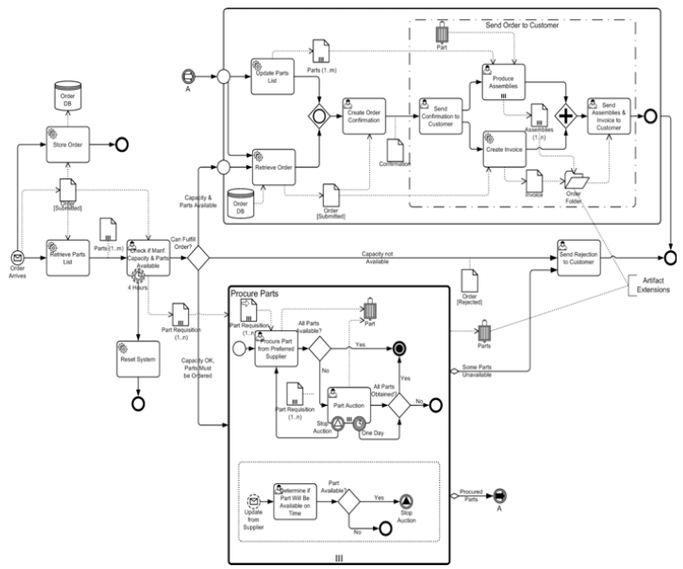
MPB (6 cfu, 295AA)

Roberto Bruni

http://www.di.unipi.it/~bruni

22 - Business process modelling notation

Object



We overview BPMN and their analysis based on Petri nets

Ch.4.7, 5.7 of Business Process Management: Concepts, Languages, Architectures
Ch.3, 4 of Fundamental of Business Process Management. M. Dumas et al.

BPMN

Main goal:

to define a **graphical notation**
that is **readily understandable:**

by **business analysts** (initial drafts of processes)

by **technical developers** (process implementation)

by **business people** (process management)

Short history

2000 - **Business Process Management Initiative** (BPMI.org)
(independent organization, studying **open specifications** for the management of **e-Business processes**)

2005 - BPMI and the Object Management Group™ (OMG™) merge their activities on BPM forming the
Business Modeling & Integration Domain Task Force (**BMI -DTF**)

2006 - **BPMN 1.0 approved**
2007 - BPMN 1.1 approved
2009 - BPMN 1.2 approved

2009 - BPMN 2.0 Beta 1 proposed
2010 - BPMN 2.0 Beta 2 proposed
2011 **- BPMN 2.0 Final delivered**

A joint effort!

Standardisation

In the context of graphical models for business processes

the development of BMPN is an important step in:

reducing the fragmentation that existed
with myriad of process modelling tools and notations

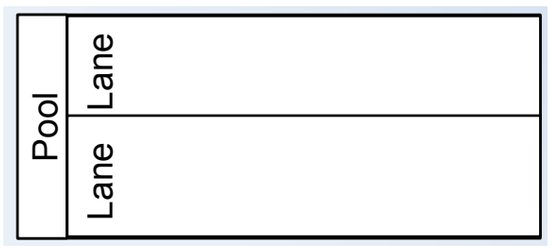
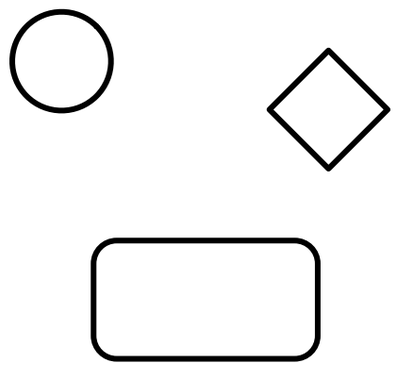
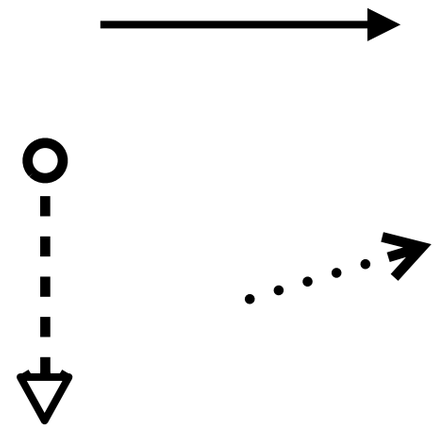
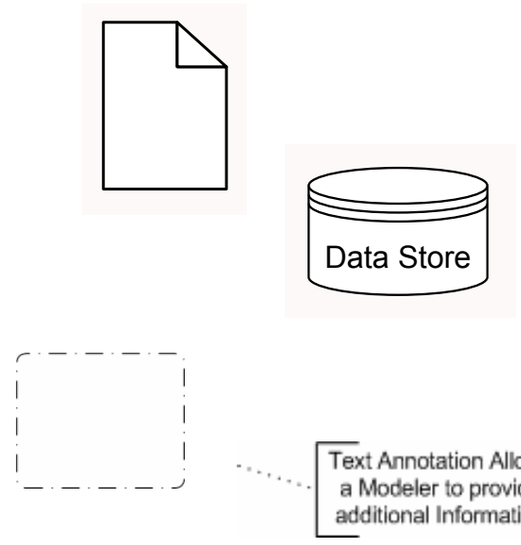
exploiting experiences with many divergent proposals to
consolidate the best ideas

supporting the adoption of **inter-operable**
business process management systems

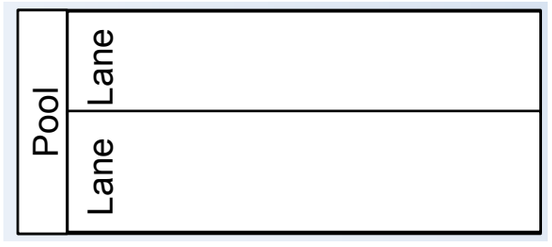
Business process diagrams

BPMN defines a standard for
Business Process Diagrams (**BPD**)
based on **flowcharting technique**

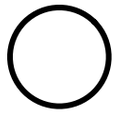
Four categories of elements

swimlanes	flow objects	connecting objects	artefacts
			

BPMN vs EPC (roughly)

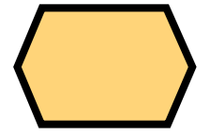


swimlanes



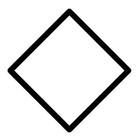
event

event



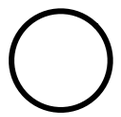
activity

function



gateway

connector



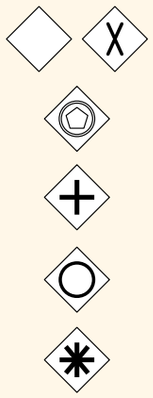
sequence flow

control flow

message flow

BPMN - Business Process Modeling Notation

Gateways



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

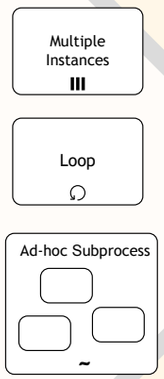
Event-based Exclusive Gateway
Is always followed by catching events or receive tasks. Sequence flow is routed to the subsequent event/task which happens first.

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 based on branching conditions. When merging, it awaits all active incoming branches to complete.

Complex Gateway
It triggers one or more branches based on complex conditions or verbal descriptions. Use it sparingly as the semantics might not be clear.

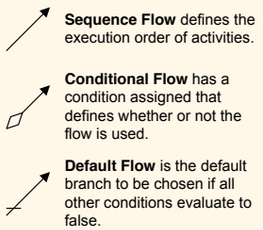
Activities



Multiple Instances of the same activity are started in parallel or sequentially, e.g. for each line item in an order.

Loop Activity is iterated if a loop condition is true. The condition is either tested before or after the activity execution.

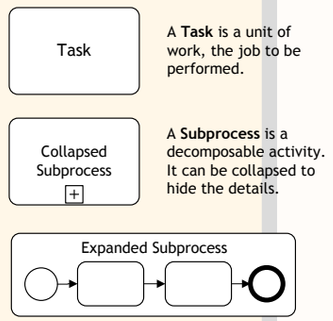
Ad-hoc Subprocesses contain tasks only. Each task can be executed arbitrarily often until a completion condition is fulfilled.



Sequence Flow defines the execution order of activities.

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

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

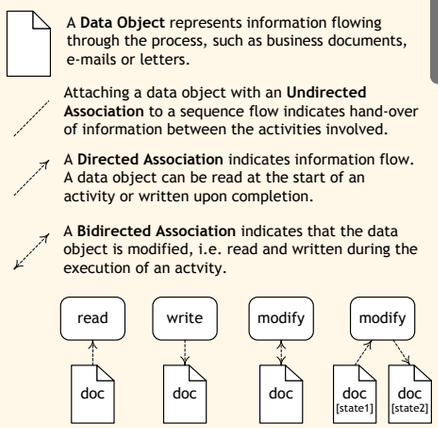


A **Task** is a unit of work, the job to be performed.

A **Subprocess** is a decomposable activity. It can be collapsed to hide the details.

An **Expanded Subprocess** contains a valid BPMN diagram.

Data



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

Attaching a data object with an **Undirected Association** to a sequence flow indicates hand-over of information between the activities involved.

A **Directed Association** indicates information flow. A data object can be read at the start of an activity or written upon completion.

A **Bidirected Association** indicates that the data object is modified, i.e. read and written during the execution of an actvity.

Events

	Start		Intermediate		End	
	Catching		*Throwing*			
Plain	○	◎			◉	Untyped events, typically showing where the process starts or ends.
Message	✉	✉	✉		✉	Receiving and sending messages.
Timer	◷	◷				Cyclic timer events, points in time, time spans or timeouts.
Error		↯			↯	Catching or throwing named errors.
Cancel		⊗			⊗	Reacting to cancelled transactions or triggering cancellation.
Compen-sation		◀◀	◀◀		◀◀	Compensation handling or triggering compensation.
Conditional	☰	☰				Reacting to changed business conditions or integrating business rules.
Signal	△	△	▲		▲	Signalling across different processes. One signal thrown can be caught multiple times.
Multiple	⬠	⬠	⬟		⬟	Catching or throwing one out of a set of events.
Link		⇨	➡			Off-page connectors. Two corresponding link events equal a sequence flow.
Terminate					●	Triggering the immediate termination of a process.

Catching

Start Event: Catching an event starts a new process instance.

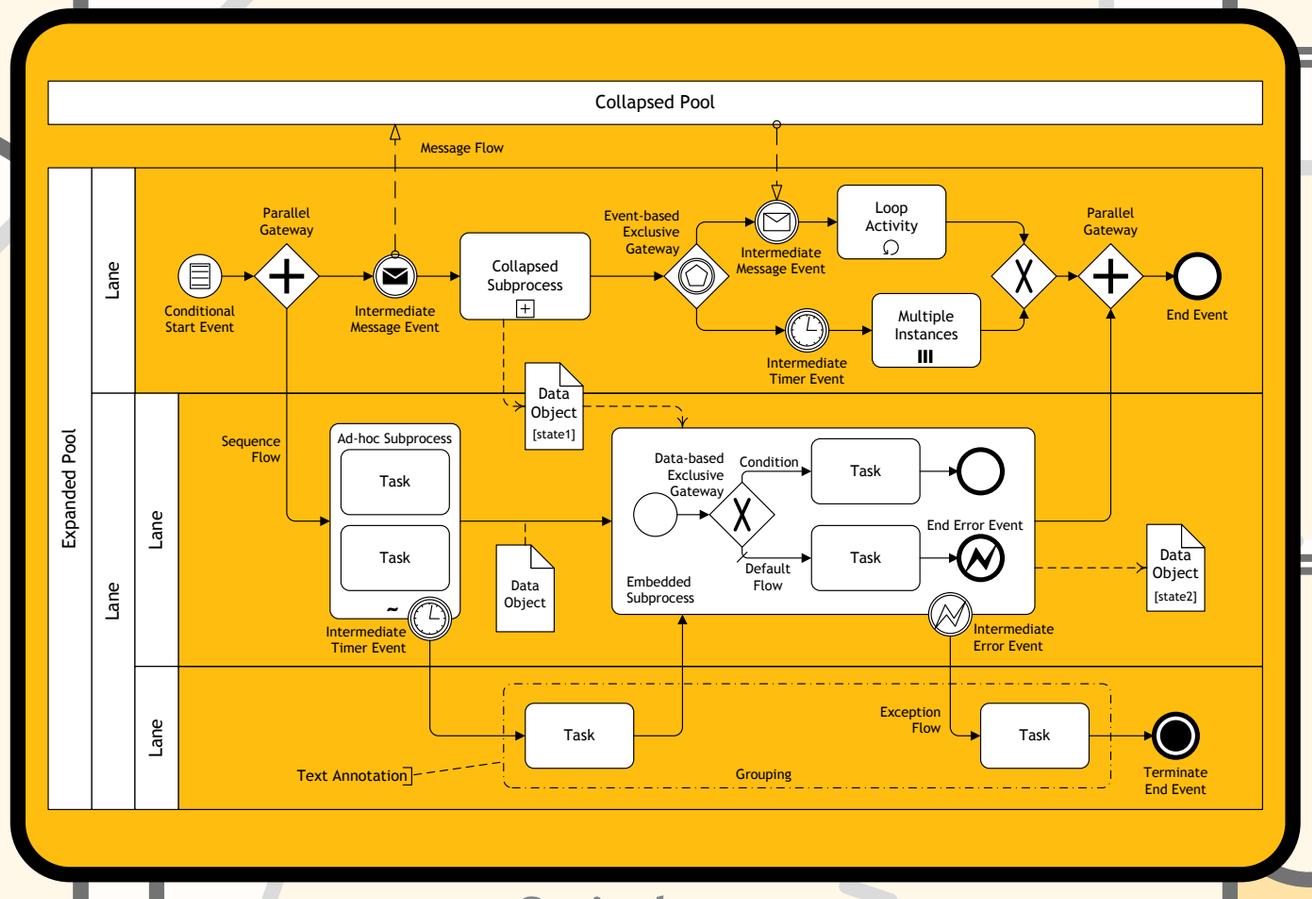
Intermediate Event (catching): The process can only continue once an event has been caught.

Throwing

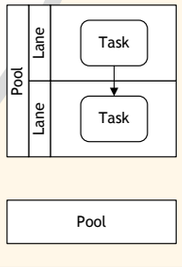
End Event: An event is thrown when the end of the process is reached.

Intermediate Event (throwing): An event is thrown and the process continues.

Attached Intermediate Event: The activity is aborted once an event is caught.

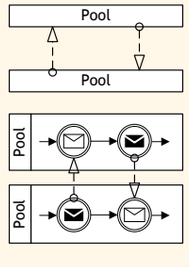


Swimlanes



Pools and **Lanes** represent responsibilities for activities in a process. A pool or a lane can be an organization, a role, or a system. Lanes sub-divide pools or other lanes hierarchically.

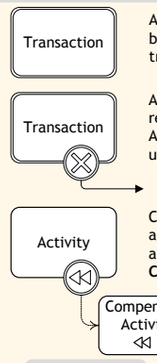
Collapsed Pools hide all internals of the contained processes.



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

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

Transactions

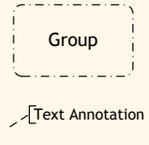


A **Transaction** is a set of activities that logically belong together; it might follow a specified transaction protocol.

Attached **Intermediate Cancel Events** indicate reactions to the cancellation of a transaction. Activities inside the transaction are compensated upon cancellation.

Completed activities can be compensated. An activity and the corresponding **Compensate Activity** are related using an attached **Intermediate Compensation Event**.

Documentation



An arbitrary set of objects can be defined as a **Group** to show that they logically belong together.

Any object can be associated with a **Text Annotation** to provide additional documentation.

Business **P**rocess **T**echnology
Prof. Dr. Mathias Weske

Web: bpt.hpi.uni-potsdam.de
Oryx: oryx-project.org
Blog: bpmn.info
BPMN Version 1.2

Authors

Gero Decker
Alexander Grosskopf
Sven Wagner-Boysen

HPI Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam

ORYX

BPMN 2.0 vs 1.0

Updated (new markers):

Tasks/SubProcesses

Events

Gateways

Artefacts

Added:

Choreographies

Full metamodel

XML Serialization

Diagram Interchange

BPMN Execution Semantics (verbal)

BPMN 2.0 - Business Process Model and Notation

http://bpmb.de/poster

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 Sub-Process or Task that is reused in the current process.

Activity Markers

Markers indicate execution behavior of activities:

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

Task Types

Types specify the nature of the action to be performed:

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

Sequence Flow
defines the execution order of activities.

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

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

Gateways

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

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

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

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

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

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

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

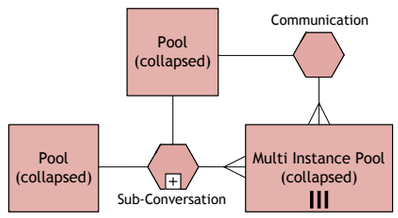
Conversations

A **Communication** defines a set of logically related message exchanges. When marked with a [+] symbol it indicates a Sub-Conversation, a compound conversation element.

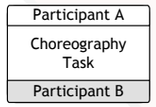
A **Conversation Link** connects Communications and Participants.

A **Forked Conversation Link** connects Communications and multiple Participants.

Conversation Diagram

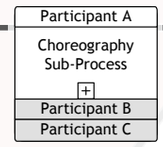


Choreographies



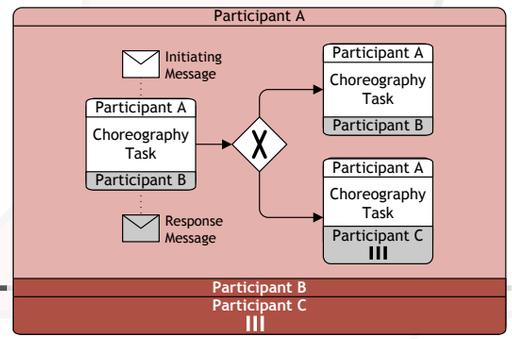
A **Choreography Task** represents an Interaction (Message Exchange) between two Participants.

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

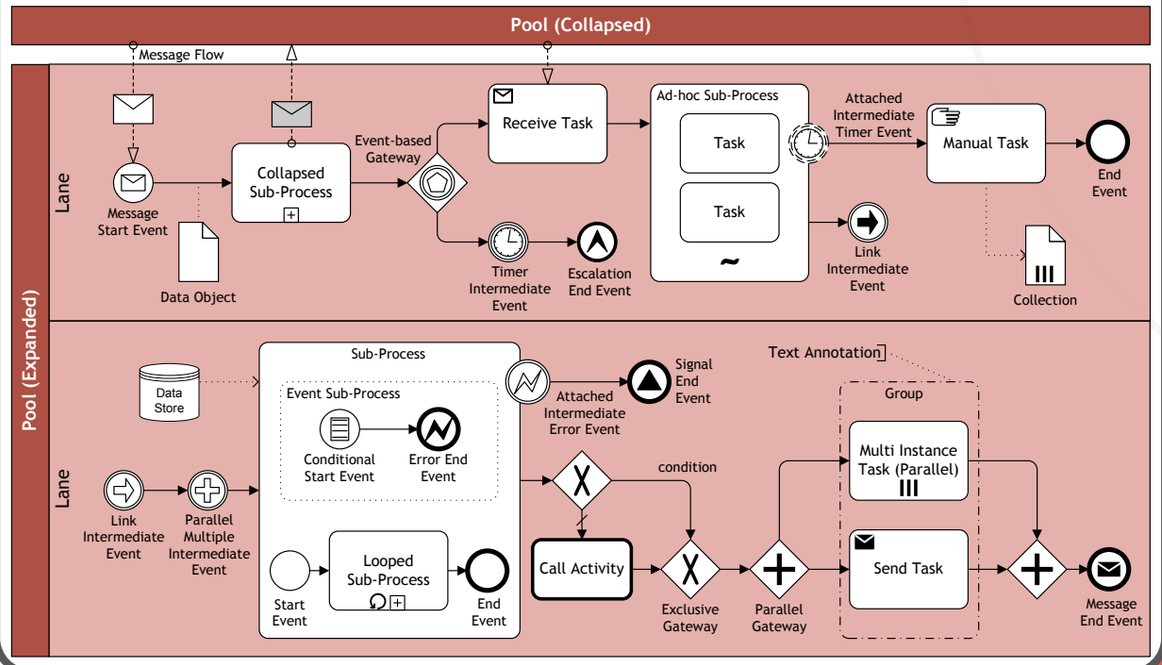


A **Choreography Sub-Process** contains a refined choreography with several Interactions.

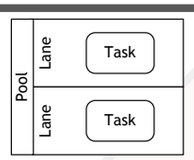
Choreography Diagram



Collaboration Diagram

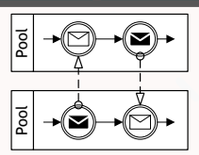


Swimlanes



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

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

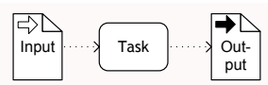


The order of message exchanges can be specified by combining message flow and sequence flow.

Events

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

Data



A **Data 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.

BPM Offensive Berlin

HPI Hasso Plattner Institut

camunda the business process company

inubit integrating your business and IT

SIGNAVIO simply professional

BPMN 2.0 - Business Process Model and Notation

http://bpmb.de/poster

Tradotto da: dexea

Attività

Task

Un ***task*** è un unità di lavoro, cioè il lavoro da svolgere. Quando si annota con il simbolo [+] indica un sottoprocesso, cioè un'attività che può essere perfezionata.

Transazione

Una **transazione** è un insieme di attività che si legano logicamente; essa potrebbe seguire uno specifico protocollo.

Sottoprocesso basato su eventi

Un **sottoprocesso basato su eventi** si trova all'interno di un processo o sottoprocesso. Si avvia quando il suo evento di inizio viene attivato e può interrompere il processo di livello superiore oppure eseguire in parallelo (senza interruzioni) in base all'evento di inizio.

Call Activity

Una ***call activity*** è un contenitore di un sottoprocesso definito globalmente o un task che può essere riusato nel processo attuale.

Simboli per attività

I seguenti simboli indicano il comportamento di esecuzione delle attività:

- Sottoprocesso
- Loop
- Esecuzione in parallelo
- Esecuzione sequenziale
- Ad hoc
- Compensazione

Tipologie di tasks

Le tipologie specificano la natura dell'azione da eseguire

- Task di invio
- Task di ricezione
- Utente
- Task manuale
- Regole di business
- Service
- Script

Flusso sequenziale

definisce l'ordine di esecuzione delle attività.

Flusso predefinito

è il ramo predefinito da scegliere se tutte le altre condizioni vengono valutate come false.

Flusso condizionale

ha una condizione assegnata che definisce se usare o meno il flusso.

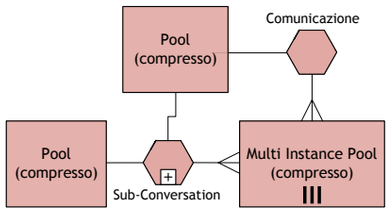
Conversazioni

Una **comunicazione** definisce un insieme di scambi di messaggi collegati logicamente. Se annotati con un simbolo [+] indicano una comunicazione interna ad un'altra conversazione.

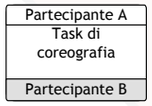
Un ***conversation link*** connette le comunicazioni ed i partecipanti.

Un ***forked conversation link*** connette le comunicazioni e molteplici partecipanti.

Diagramma di conversazione

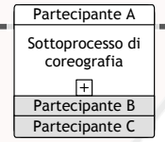


Coreografie



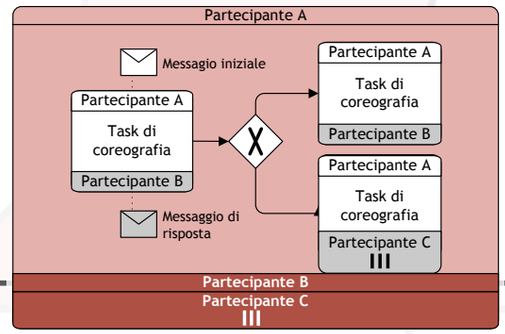
Un **Task di coreografia** rappresenta un'interazione(scambio di messaggi) tra due partecipanti.

Il simbolo ***Multiple Participants*** denota un insieme di partecipanti della stessa tipologia.

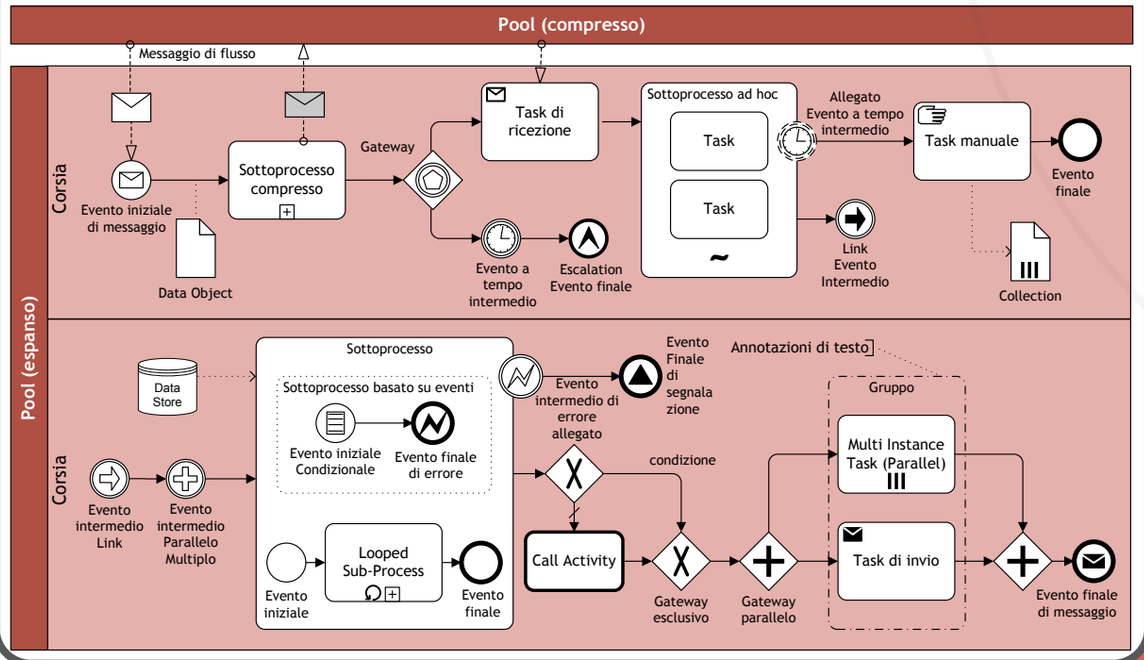


Un **Processo di coreografia** contiene una coreografia rifinita con molte interazioni.

Diagramma di coreografia



Collaboration Diagram



Eventi

	Inizio			Intermedio				Fine
	Alto livello	Interruzione di sottoprocessi	Non-interruzione di sottoprocessi	Catching	Boundary Interrupting	Boundary Non-Interrupting	Throwing	
Non definiti: punti di inizio, cambi di stato, o stati finali.	○						◎	●
Messaggio: invio e ricezione di messaggi	✉	✉	✉	✉	✉	✉	✉	✉
Timer: eventi a tempo.	🕒	🕒	🕒	🕒	🕒	🕒		
Escalation: passa ad un livello più alto di responsabilità.		▲	▲		▲	▲	▲	▲
Condizionale: reagisce a condizioni di business cambiate o integra regole di business.	☰	☰	☰	☰	☰	☰		
Link: Due corrispondenti *link events* sono uguali ad un flusso sequenziale.				⇨			⇨	
Errore: attiva o si occupa di un errore.		⚡			⚡			⚡
Cancel: reagisce a delle transazioni cancellate o causa una cancellazione.					⊗			⊗
Compensazione: gestisce o innesca la compensazione.		◁◁			◁◁		◁◁	◁◁
Signal: comunica con più processi. Lo stesso segnale può essere intercettato più volte.	△	△	△	△	△	△	▲	▲
Multiplo: intercetta uno tra vari eventi. Gestisce tutti gli eventi definiti.	⬠	⬠	⬠	⬠	⬠	⬠	⬟	⬟
Parallelo Multiplo: intercetta tutti gli eventi.	✚	✚	✚	✚	✚	✚		
Terminate: causa la fine immediata di un processo.								●

Gateways

Esclusivo(xor)

Basato su eventi

Parallelo

In caso di *splitting*, il flusso sequenziale viene diretto esattamente verso uno dei rami in uscita. In caso di *merging*, il flusso aspetta che un ramo in entrata arrivi a termine prima di andare avanti.

Questo simbolo è sempre seguito da intercettazioni di eventi o *tasks* di ricezione. Il flusso sequenziale prosegue verso il sucessivo *task*/evento che accade per primo.

Quando viene usato per dividere il flusso sequenziale, tutti i rami in uscita sono attivati simultaneamente. Invece quando viene usato per unire rami paralleli, il flusso aspetta il completamento di tutti i rami in entrata prima di andare avanti.

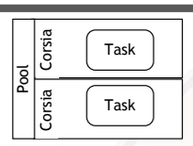
Inclusivo
In caso di *splitting*, uno o più rami sono attivati. Il flusso va avanti solamente quando l'esecuzione di tutti i rami è terminata.

Complesso
Gestioni di *merging* e *branching* che non sono gestite da altri gateways.

Esclusivo basato su eventi
All'attivazione di ogni evento successivo, viene avviata una nuova istanza di processo.

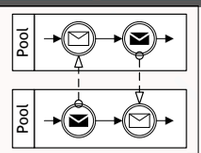
Parallelo basato su eventi
All'attivazione di tutti gli eventi successivi, viene avviata una nuova istanza di processo.

Swimlanes



Pools (Partecipanti) e **Lanes(corsie)** rappresentano le responsabilità per le attività in un processo. Esse possono essere un'organizzazione, un ruolo o un sistema. Le corsie suddividono le *pools* o altre *corsie* gerarchicamente.

Flusso di messaggi rappresenta il flusso di informazioni. Un flusso di messaggi può essere unito a pools, attività, o eventi di messaggi.

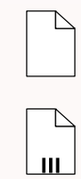


L' **ordine** degli **scambi di messaggi** può essere specificato associando il flusso di messaggi e il flusso sequenziale.

Data

Un ***Data Input*** è un input esterno usato all'interno del processo. Può essere letto da un'attività.

Un ***Data Output*** è una variabile disponibile come risultato di un intero processo.



Un ***Data Object*** rappresenta le informazioni che attraversano l'intero processo, come ad esempio documenti di business, e-mails, o lettere.

Un ***Collection Data Object*** rappresenta una collezione di informazioni, come ad esempio una lista di elementi ordinati.

Un ***Data Store*** è un luogo dove il processo può leggere oppure scrivere dati, ad esempio un database. Esso si mantiene oltre la durata dell'istanza del processo.

Un **messaggio** è usato per rappresentare i contenuti di una comunicazione tra due partecipanti.

BPM Offensive Berlin

Hasso Plattner Institut — IT Systems Engineering | Universität Potsdam

camunda — the business process company

inubit — integrating your business and IT

SIGNAVIO — simply professional

BPMN 2.0 (2009/11)
FAQ

What is BPMN?

BPMN is a graphical notation that depicts the steps (end to end flow) in a business process.

Specifically designed to coordinate
the sequence of processes and
the messages that flow between participants
in a related set of activities.

BPMN 2.0 (2009/11)

FAQ

Why is BPMN important?

The world of business processes has changed dramatically over the past few years. Processes can be coordinated from behind, within and over organizations boundaries. A business process now spans multiple participants and coordination can be complex.

Until BPMN, there has not been a standard modelling technique developed that addresses these issues.
BPMN provides users with a **royalty free notation**.

This will benefit users in a similar manner in which UML standardised the world of software engineering.
There will be training courses, books and a body of knowledge that users can access in order to better implement a business process.

BPMN 2.0 (2009/11) FAQ

Will there be a major rewrite?

Not for 2 or 3 years…

(good work! 10+ years and still no revision is planned)

Strong points of BPMN

Simplicity: A small set of basic symbols

Extensibility: many decorations available
(new ones can be added in the future)

Graphical design: **intuitive**

Generality: **orchestration** + **choreography**

Tool availability: **`.bpmn` exchange format**

Weaknesses of BPMN

over 100 graphical elements

verbose description (500 pages)

difficult to learn comprehensively:
different readings of the same diagram are possible

different BPMN vendors implement the execution of BPMN diagrams in different ways (and for different subsets)

1 - BPMN basics

Swimlanes (pools, lanes)

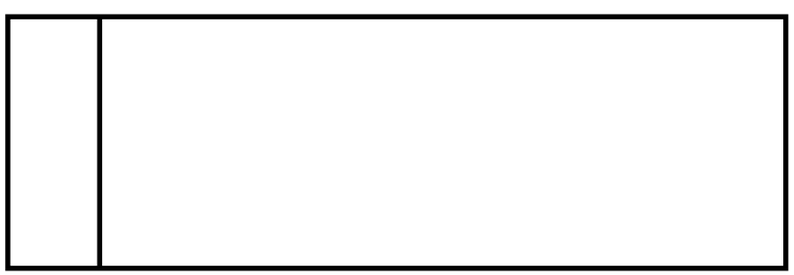
Swimlanes

A **swimlane** is a mechanism
to **organise activities into separate visual categories**
to illustrate different capabilities or **responsibilities**

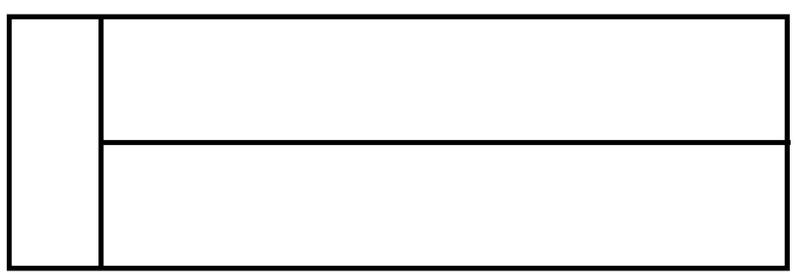
Present in many process modelling methodologies

BPMN supports two main swimlane objects:

pool

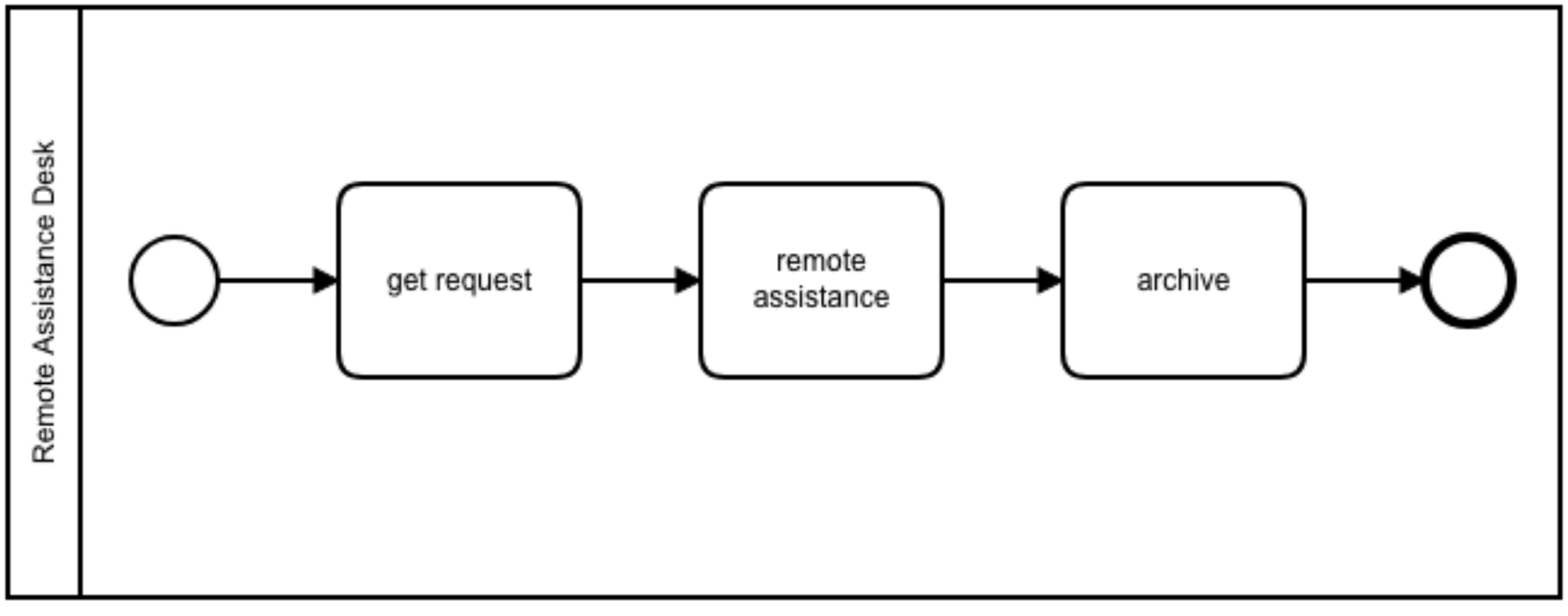


lanes



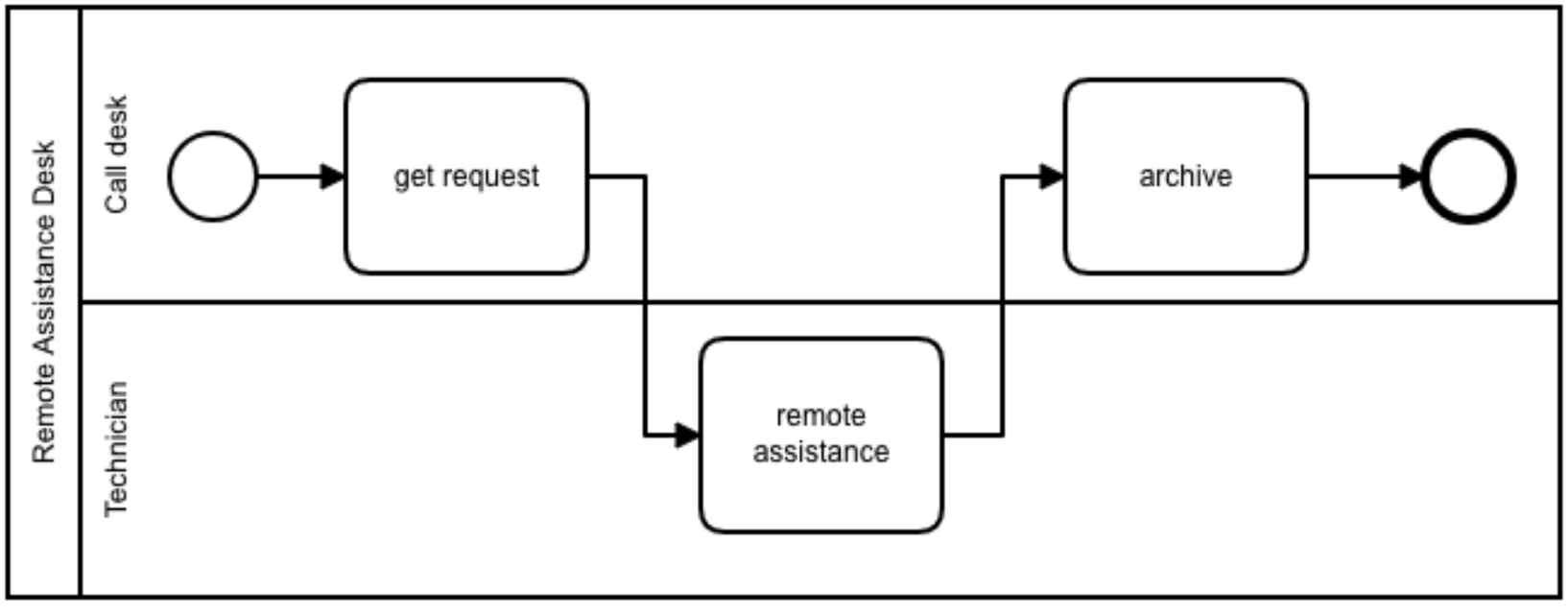
Pools

A **pool** represents a participant (or role) in a process
(represented as a rectangle with a name)



Lanes

A **lane** is a hierarchical sub-partition within a pool
that is used to organise and categorise activities
(inner rectangle that extends to the entire length of the pool



Collapsed pools

Internal process is not exposed
(like a black-box)

Constraints

A Pool MUST contain 0 or 1 business process.

A Pool can contain 0 or more lanes.

Two pools can only be connected with message flows.

Naming conventions

Process models:
a noun possibly preceded by an adjective

the label is often obtained by ``nominalizing'' the verb that describe the main action in the process
(e.g., claim handling, order fulfillment)

Avoid long labels
Articles are often omitted

Flow Objects

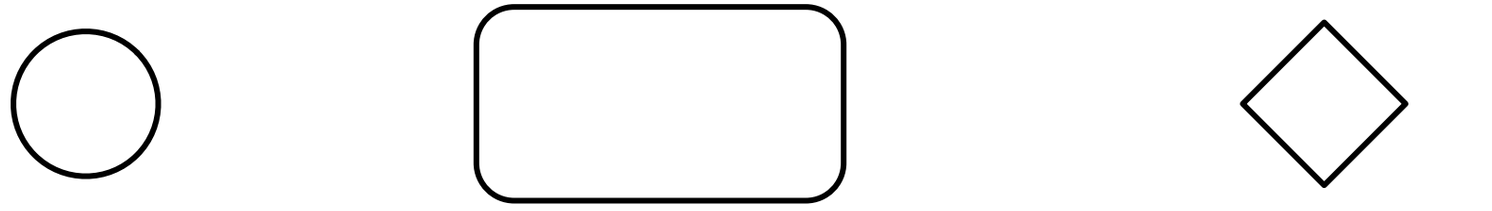
(events, activities, gateways)

Flow objects

Rationale:

fix a small set of core elements
so that modellers must learn a small number of shapes:

events **activities** **gateways**

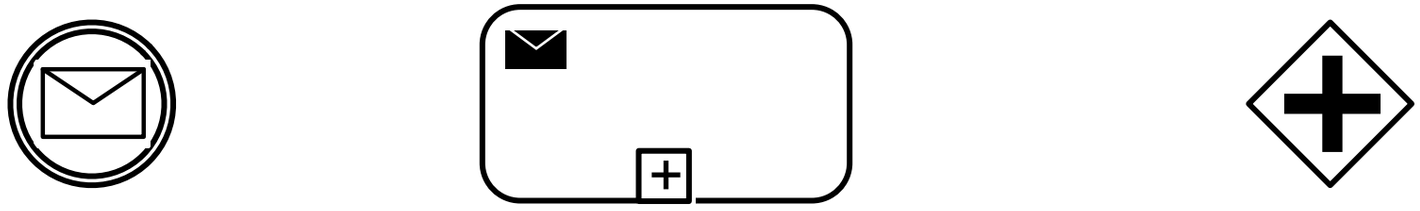


Flow objects

Rationale:

fix a small set of core elements
so that modellers must learn a small number of shapes:

events **activities** **gateways**



use different border styles and internal markers
to add many more information
(this way the notation is **extensible**)

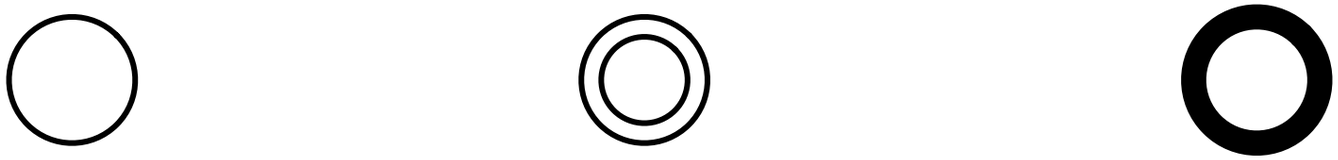
Flow objects: Events

Events

An **event** is something that “happens” during the course of a business process

An event is represented as a circle
different borders define the **type** of the event

start **intermediate** **end**



Naming conventions

Events:
the label should begin with a noun and
end with a verb in past participle form
to indicate something that just happened
(e.g., Invoice emitted)

the noun can be preceded by an adjective
(e.g., Urgent order sent)

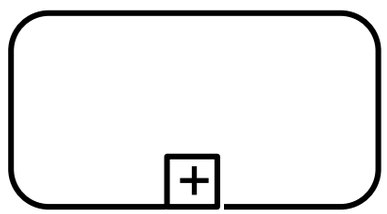
Avoid long labels
Articles are often omitted

Flow objects: Activities

Activities

An **activity** is some “unit of work” (job) to be done during the course of a business process

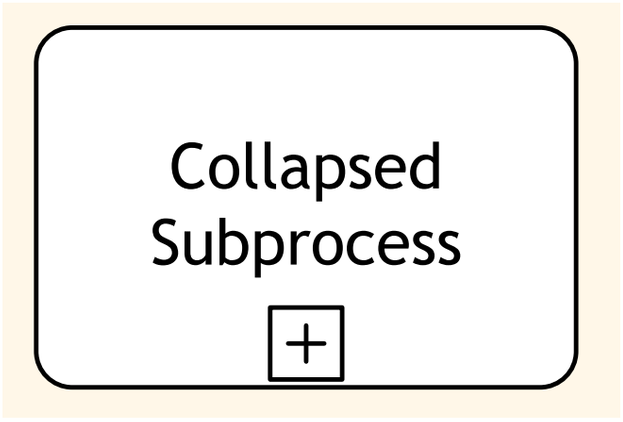
An activity is represented as a rounded box
BPMN has two main types of activities atomic (**task**) or compound (**sub-process**)

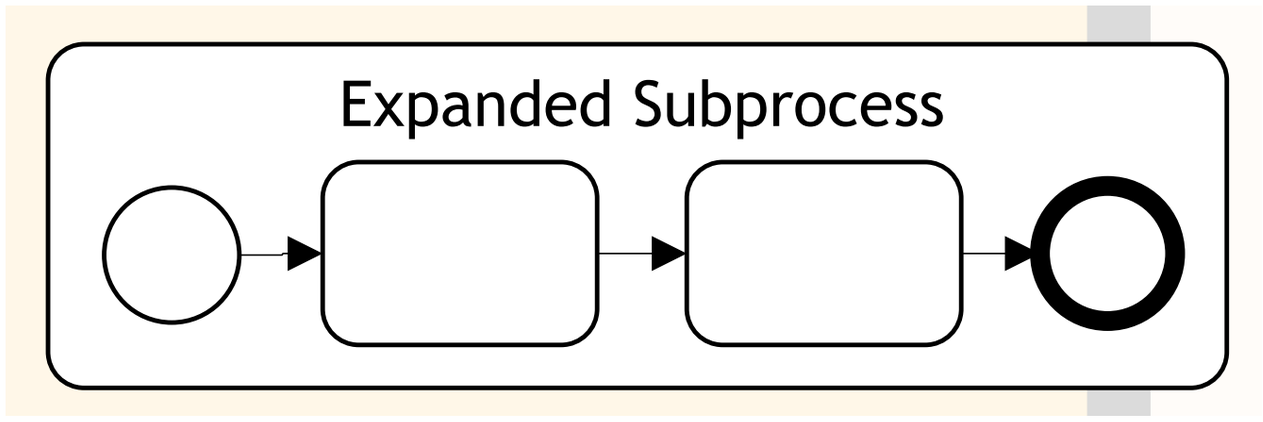


Sub-processes

Large process models are hard to parse:
we improve readability
by hiding certain parts within sub-processes

A **sub-process** is a self-contained, composite activity that can be broken into smaller units of work

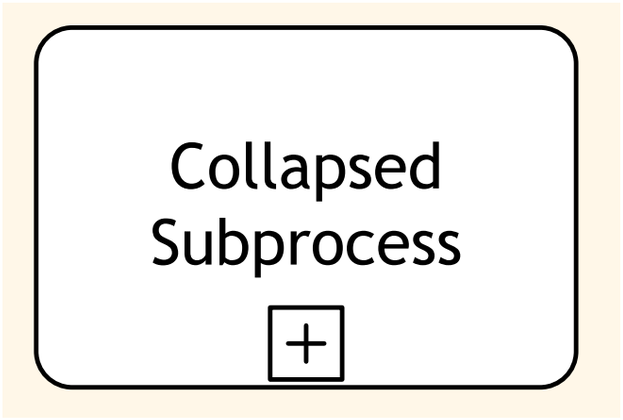


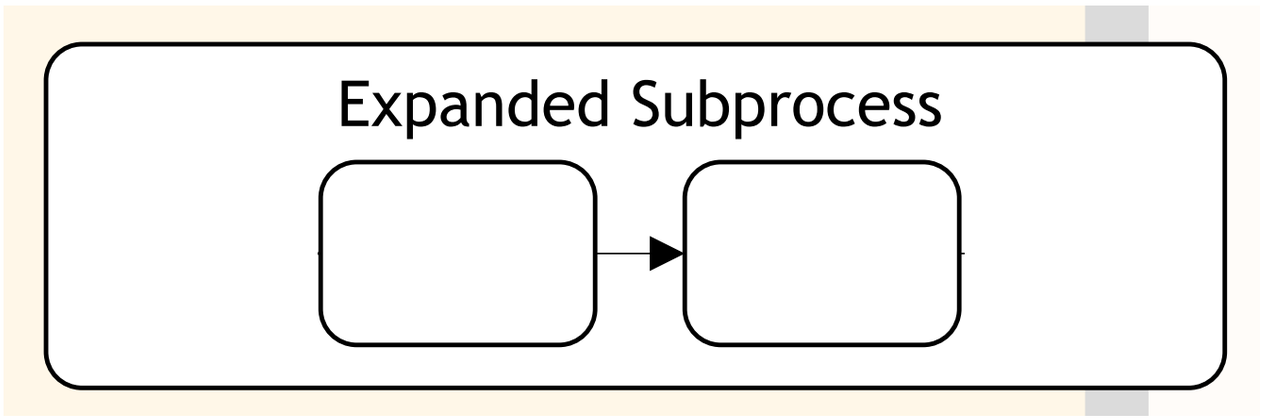


Sub-processes

Large process models are hard to parse:
we improve readability
by hiding certain parts within sub-processes

A **sub-process** is a self-contained, composite activity that can be broken into smaller units of work





implicit start / end

Naming conventions

Activities:
verb in the imperative form followed by a noun
(e.g., Approve order)

the noun can be preceded by an adjective
(e.g., Issue driver license)

the verb may be followed by a complement
(e.g., Renew driver license via offline agencies)

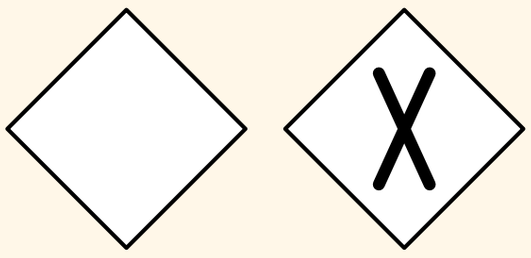
Avoid long labels
Articles are often omitted

Flow objects: Gateways

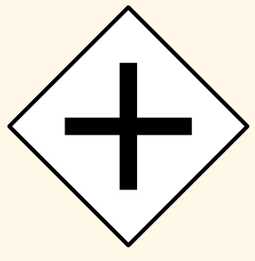
Gateways

A **gateway** is used to split/join the sequence flow (conditional, fork, wait)

A gateway is represented as a diamond shape internal markers indicate the nature of behaviour control



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



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.

Connecting objects (sequence flow, message flow, association)

Connecting objects

The Flow objects are connected together in a diagram to create the basic skeletal structure of a business process

Three connecting objects can be used:

Sequence flow

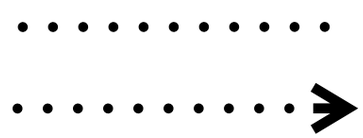
connected objects must **reside in the same pool** (but they can be in different lanes)

Message flow

connected objects must **reside in different pools**

to be discussed later

Association



connects flow objects with artefacts

to be discussed later

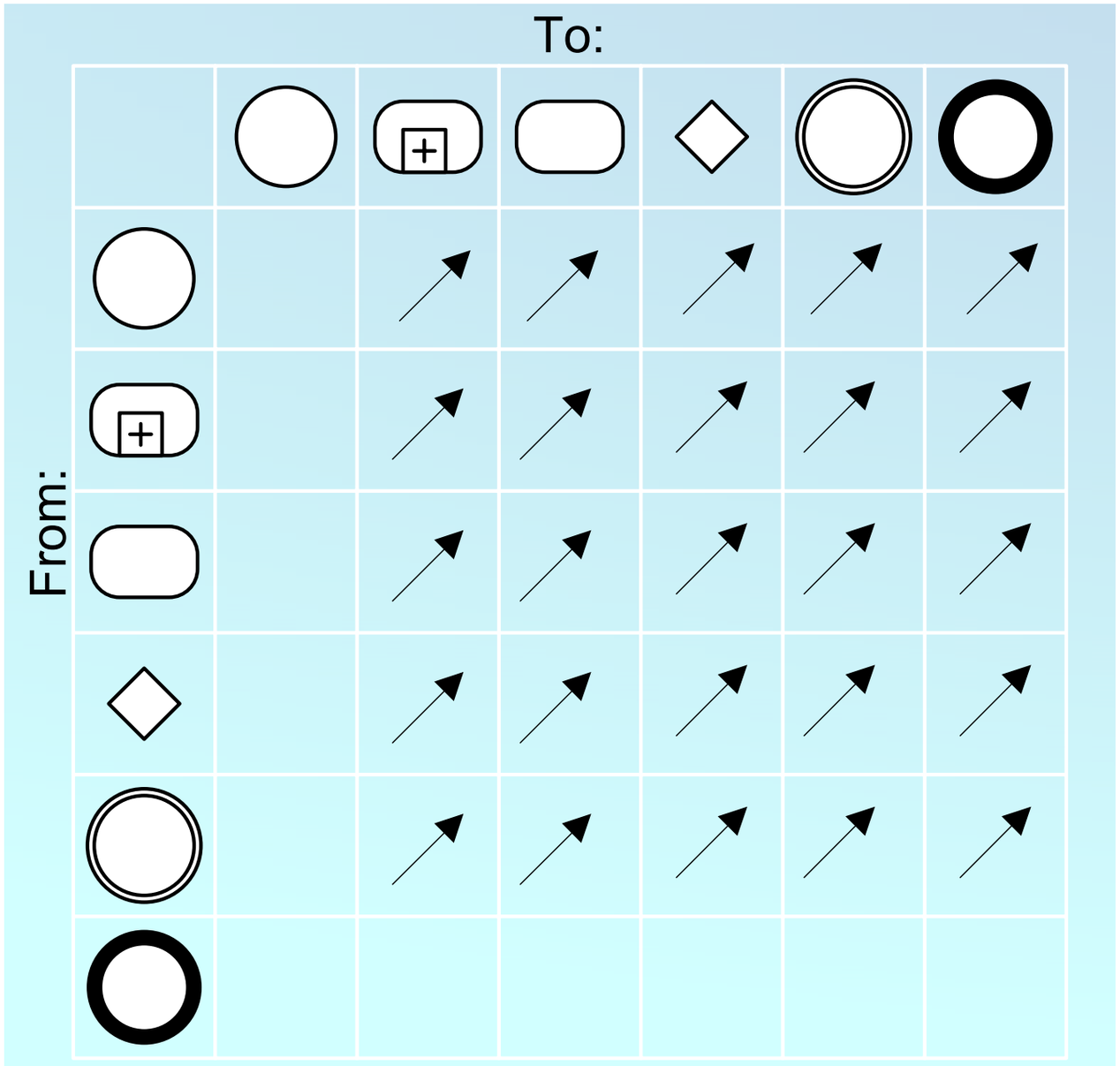
Sequence flow

A **sequence flow** is used to show the order
in which activities are to be performed

the term “control flow” is generally avoided in BPMN

A sequence flow is represented by
a solid line with a solid arrowhead

Constraints



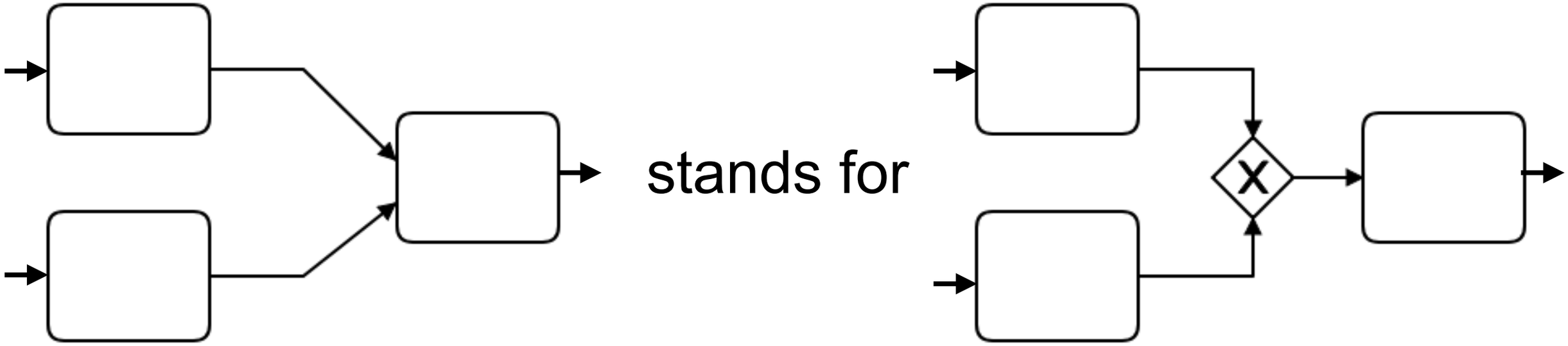
each event:
at most one incoming and
at most one outgoing
sequence flow

each activity:
exactly one incoming and
exactly one outgoing
sequence flow

each gateway:
one-to-many,
many-to-one,
many-to-many

Multiple flows and implicit gateways

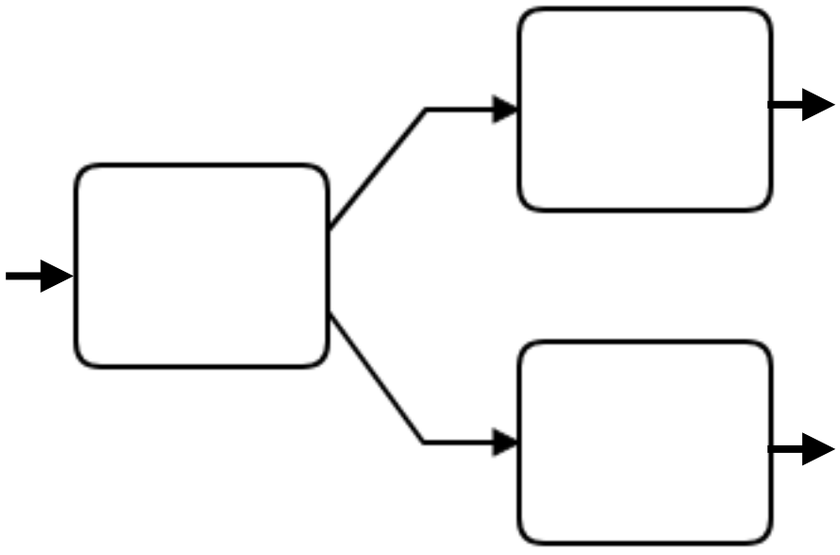
In principle each activity should have exactly:
one incoming arc, one outgoing arc
Be careful if this is not the case!



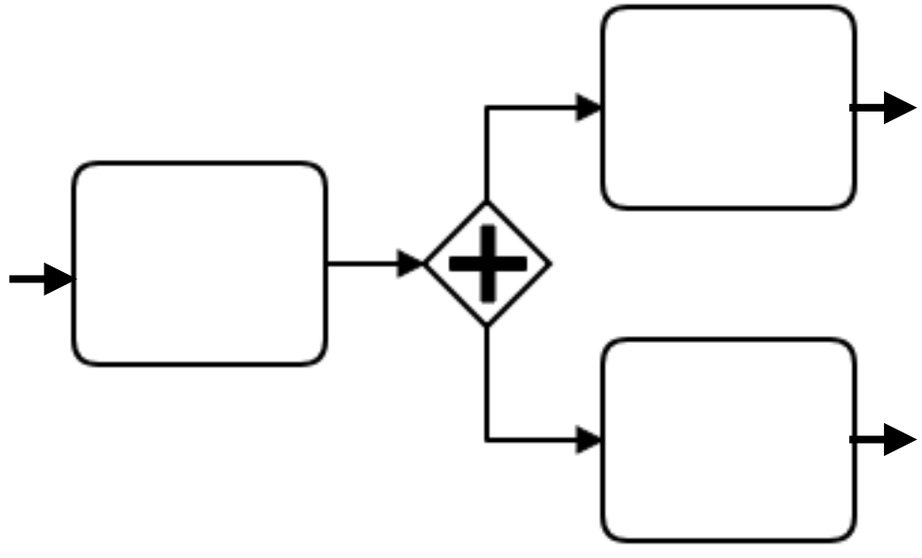
Multiple incoming flows are mutually exclusive

Multiple flows and implicit gateways

In principle each activity should have exactly:
one incoming arc, one outgoing arc
Be careful if this is not the case!



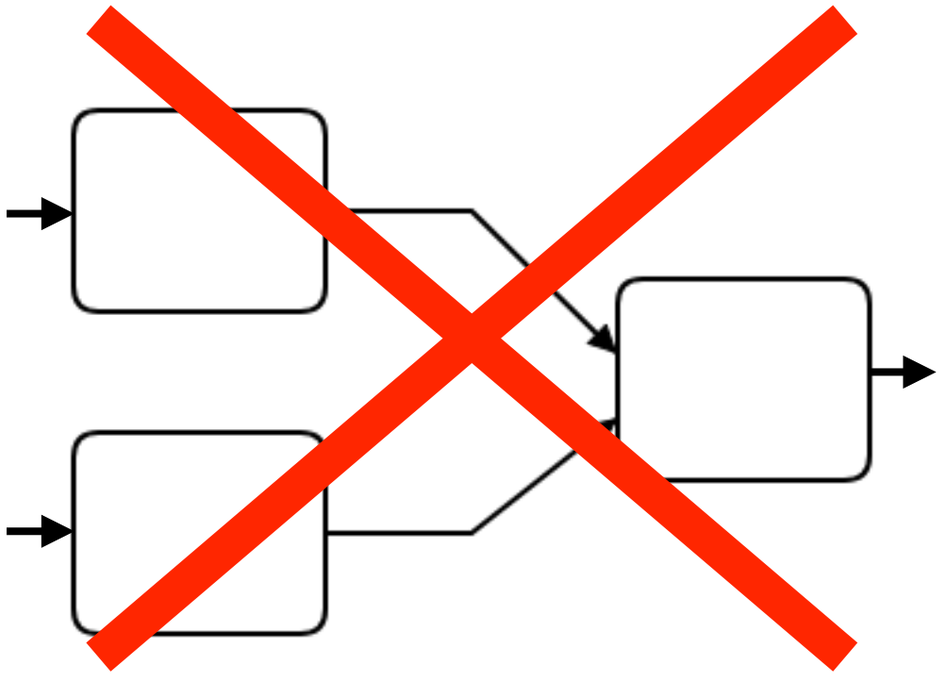
stands for

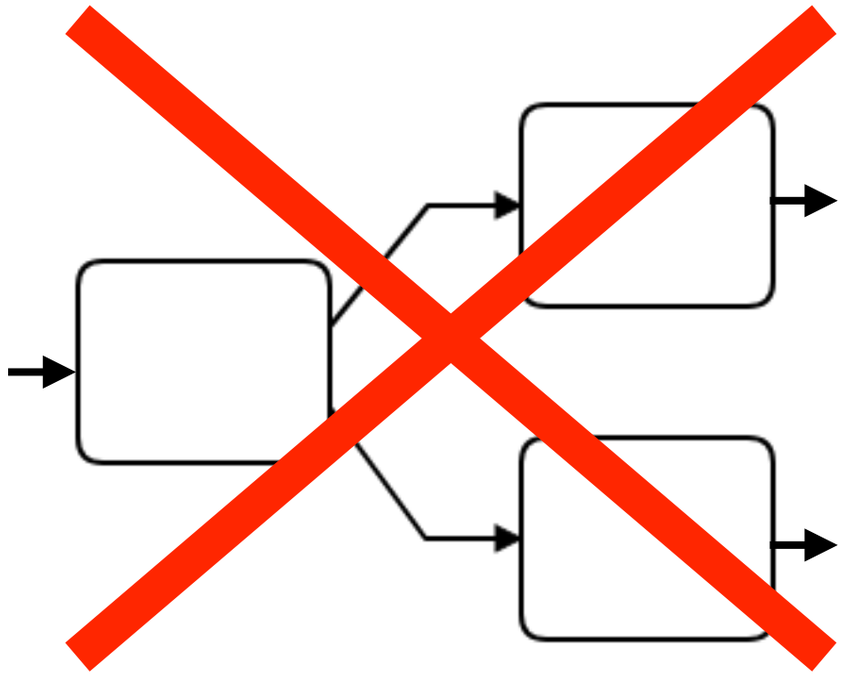


Multiple outgoing flows are activated in parallel
(unless conditions are attached to them)

In your final projects

Please avoid





Artefacts:

(data-objects, groups, text annotations)

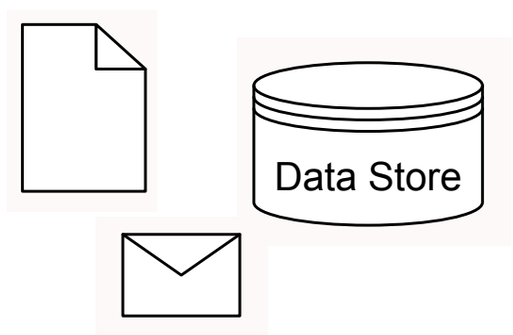
Artefacts

BPMN is designed to allow modellers and modelling tools some flexibility in extending the basic notation

Any kind of artefacts can be added to a diagram as appropriate for the specific modelling domain

BPMN includes three pre-defined types of artefacts:

data objects **groups** **text annotation**



to be discussed later

to be discussed later

Association

An **association** is used to associate data, text, and other artefacts with flow objects

An association is represented by
a dotted line (with an optional line-arrowhead)

used especially
for text annotation

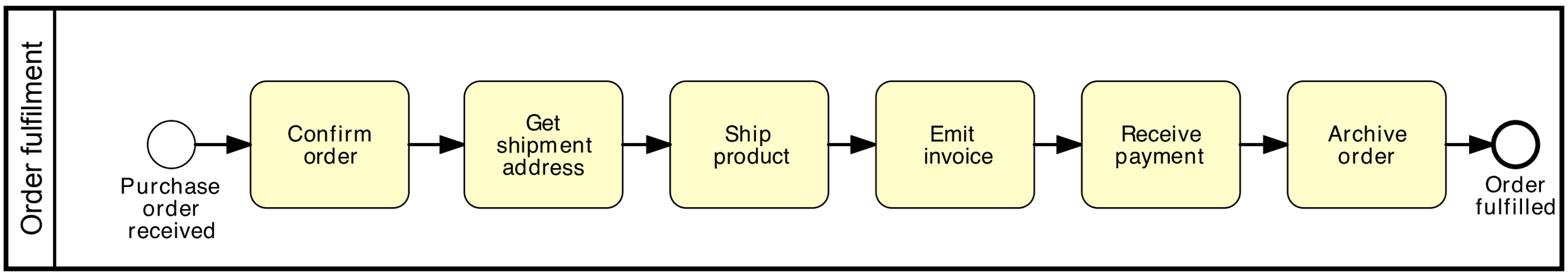
Text annotation

Any object can be associated with a **text annotation**
to provide any additional information and documentation
that can be needed

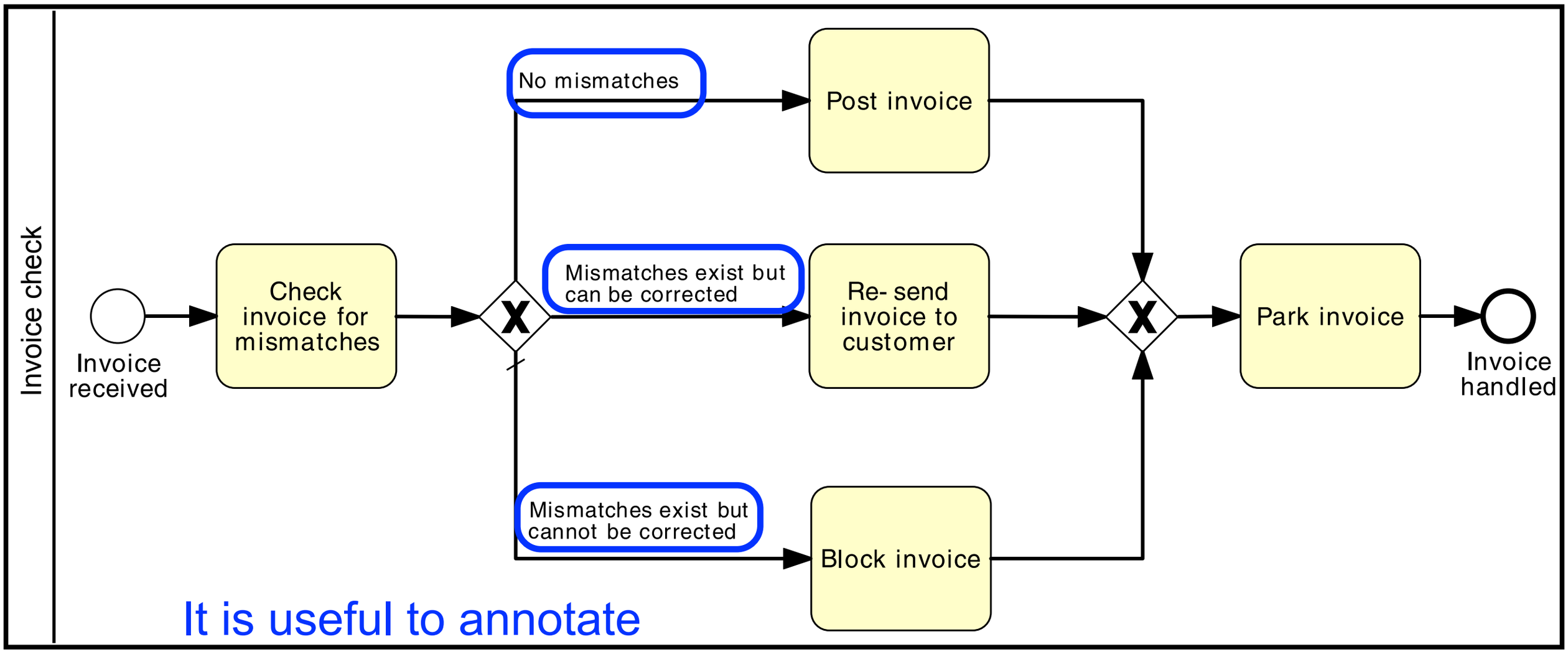
A text annotation is represented as a dotted-line call-out

Typical patterns

Sequence:
order fulfilment

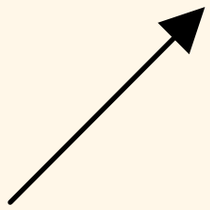


Exclusive decisions: invoice checking process

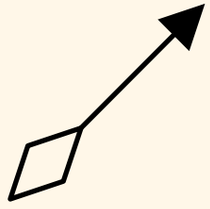


It is useful to annotate branches with the conditions under which they are taken

Annotated sequence flow

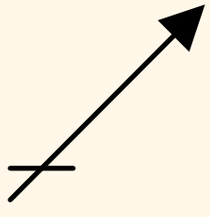


Sequence Flow defines the execution order of activities.



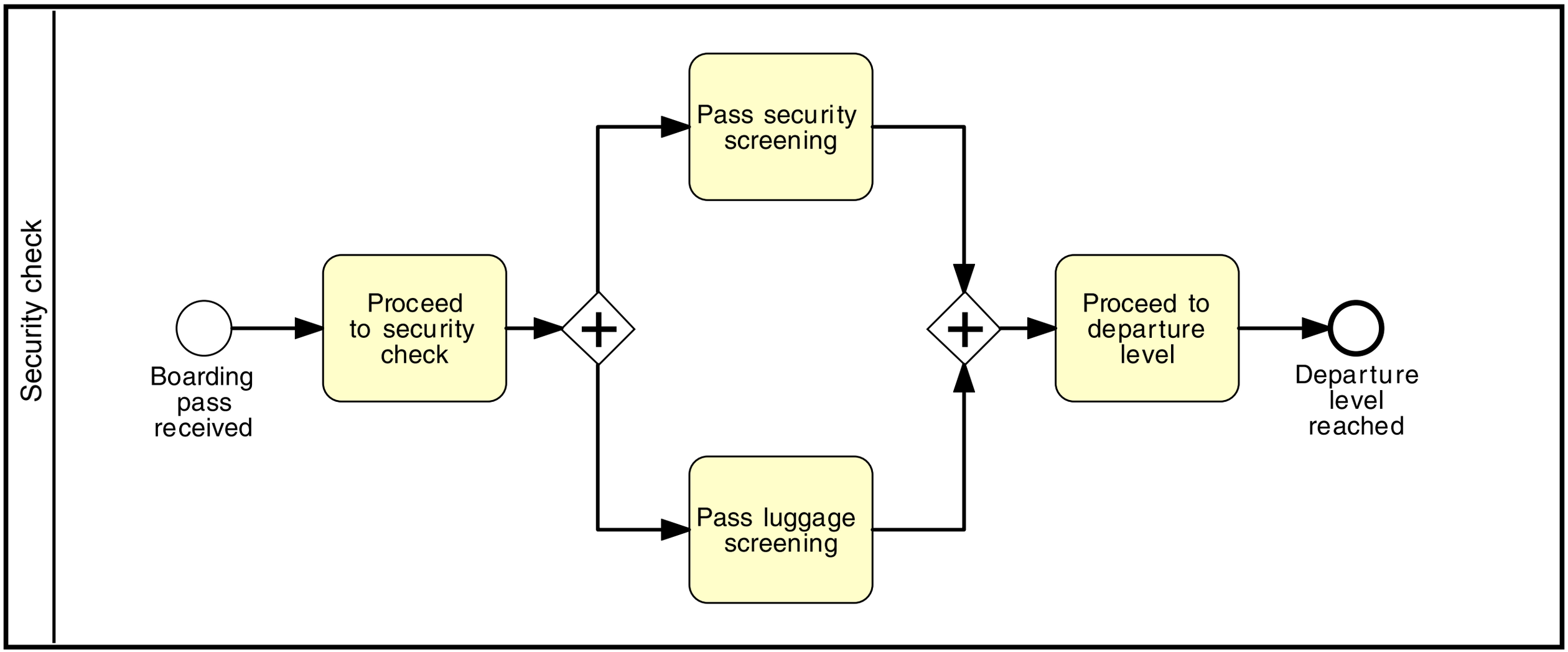
Conditional Flow has a condition assigned that defines whether or not the flow is used.

read as ``otherwise''

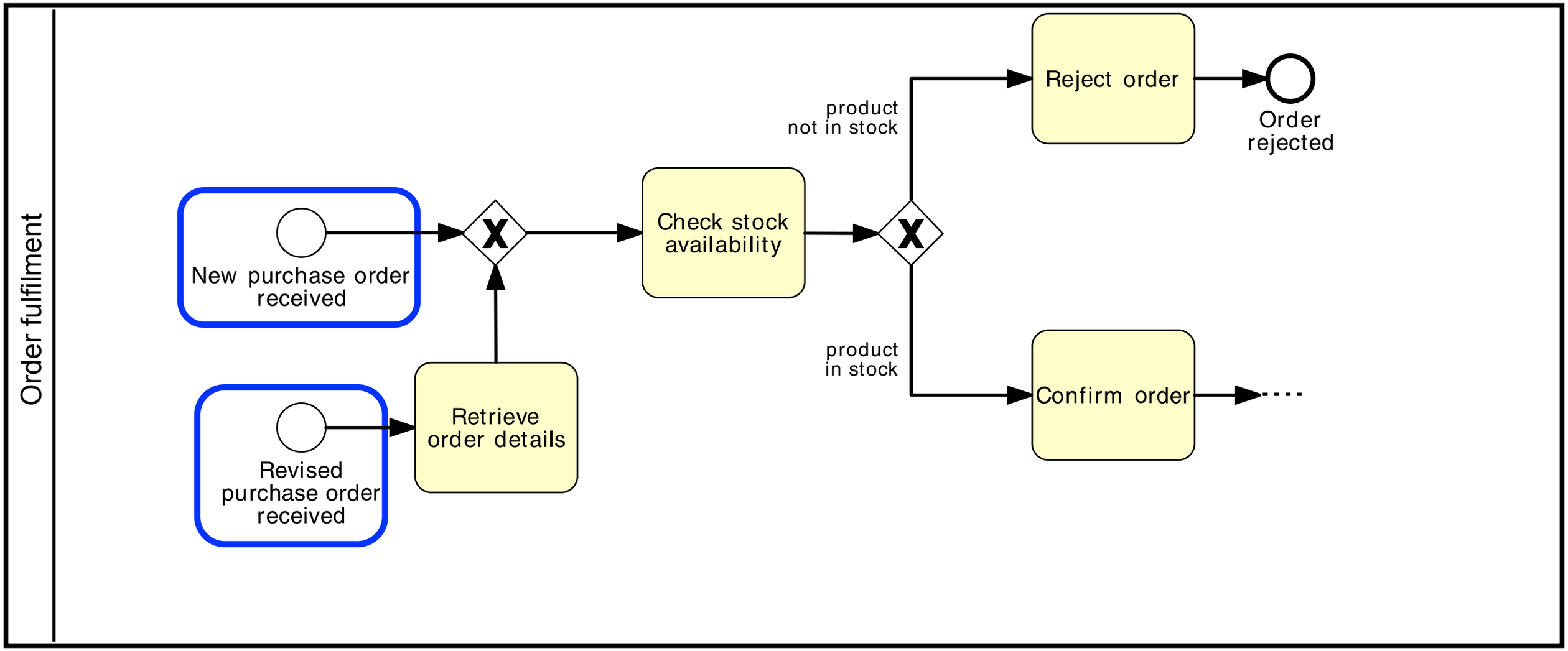


Default Flow is the default branch to be chosen if all other conditions evaluate to false.

Parallel activities: airport security check

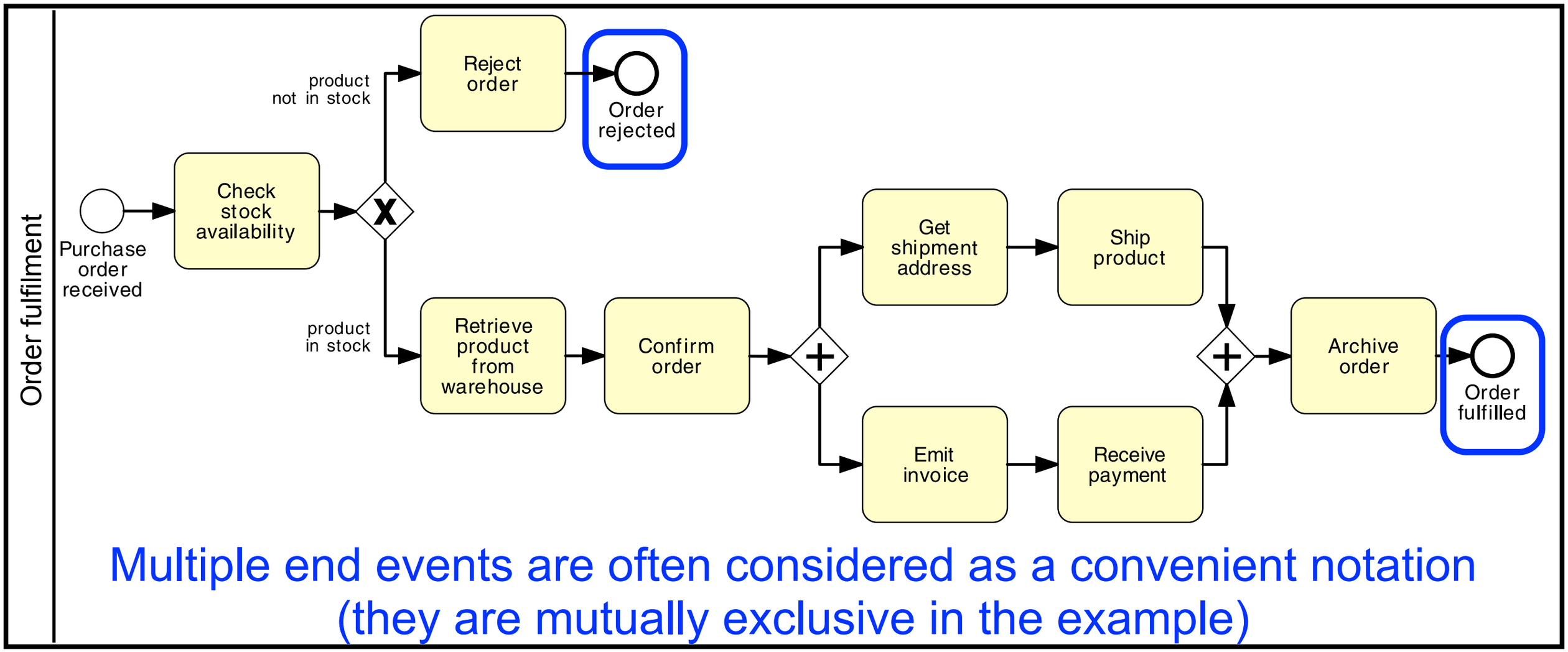


Multiple start events: order fulfilment



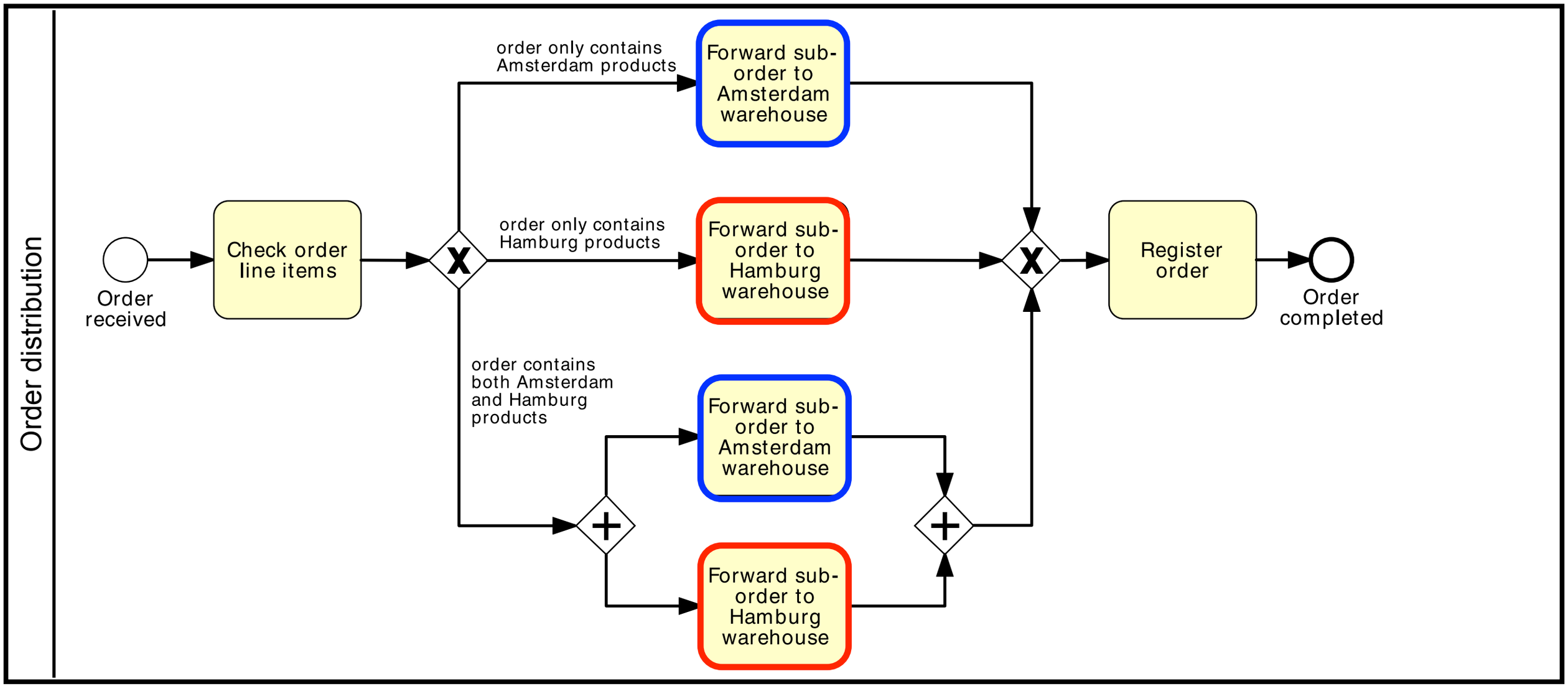
Multiple start events are often considered as a convenient notation (they capture mutually exclusive triggers to start a process instance)

Multiple end events: order fulfilment



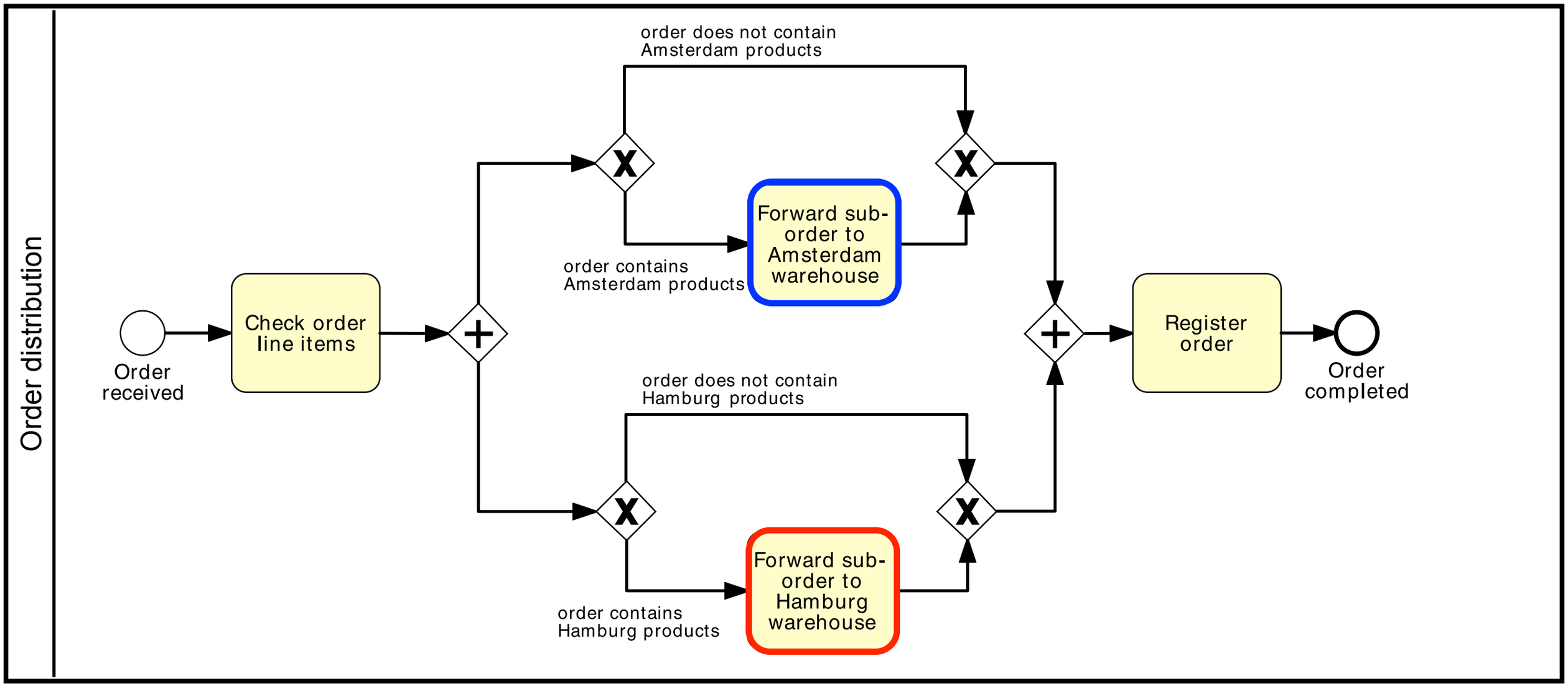
BPMN adopts **implicit termination** semantics:
a case ends only when each ``token'' reaches the end

Inclusive decisions: order distribution



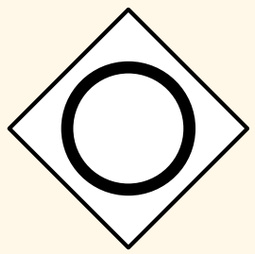
Only XOR / AND gateways, but the diagram is convoluted!
What if we had three or more warehouses? (does not scale)

Inclusive decisions: order distribution



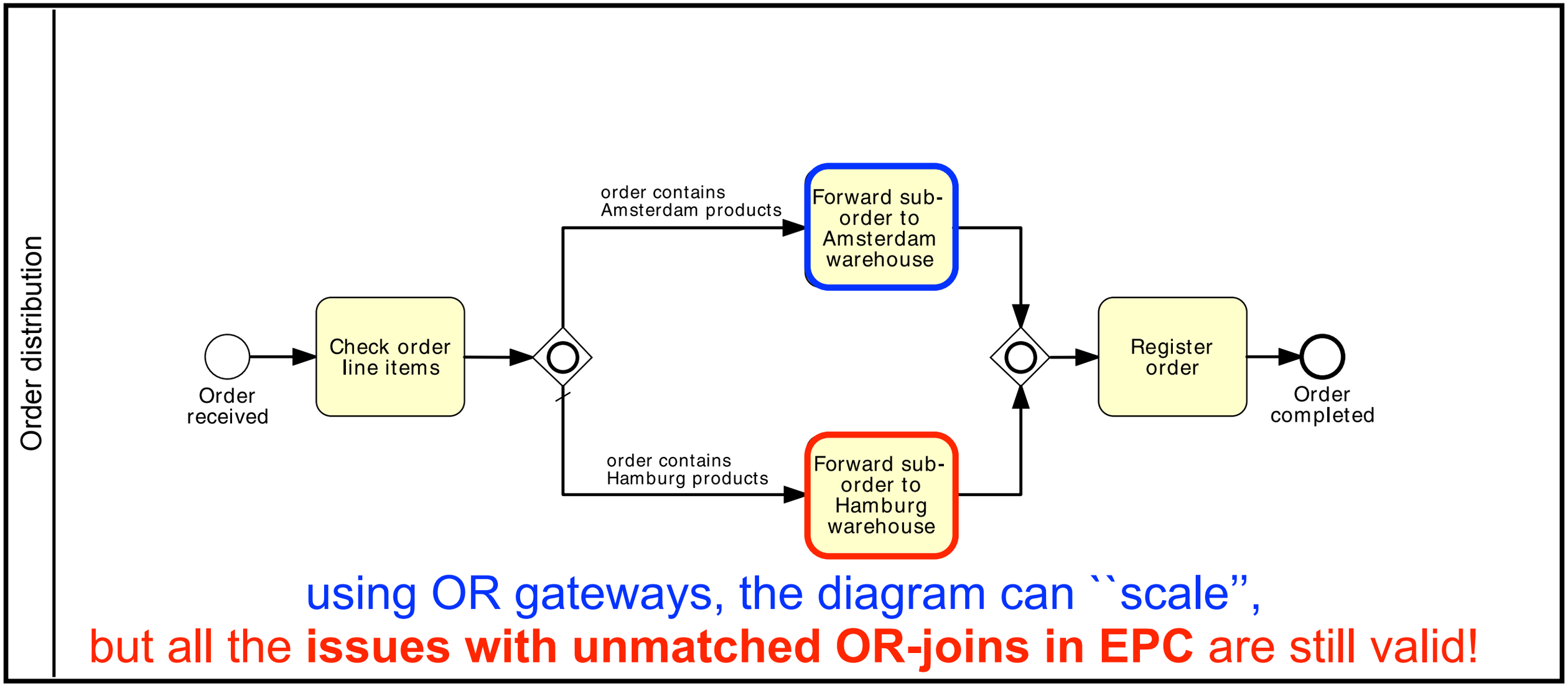
Only XOR / AND gateways, the diagram can ``scale'',
but is it correct? (also the case no-warehouse is now possible)

Inclusive decisions (one, many)



Inclusive Gateway
When splitting, one or more branches are activated based on branching conditions. When merging, it awaits all active incoming branches to complete.

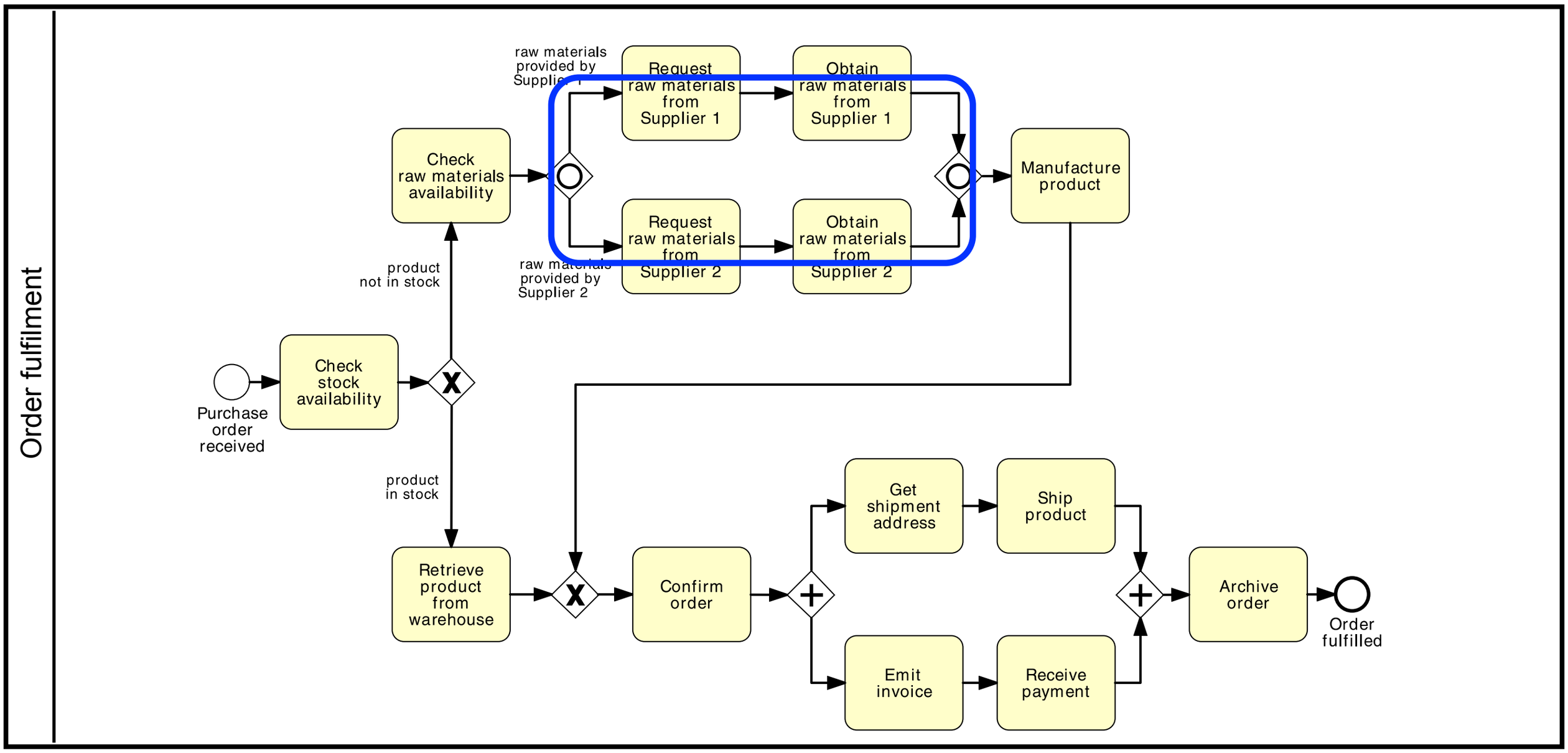
Inclusive decisions: order distribution



using OR gateways, the diagram can ``scale'',
but all the **issues with unmatched OR-joins in EPC** are still valid!

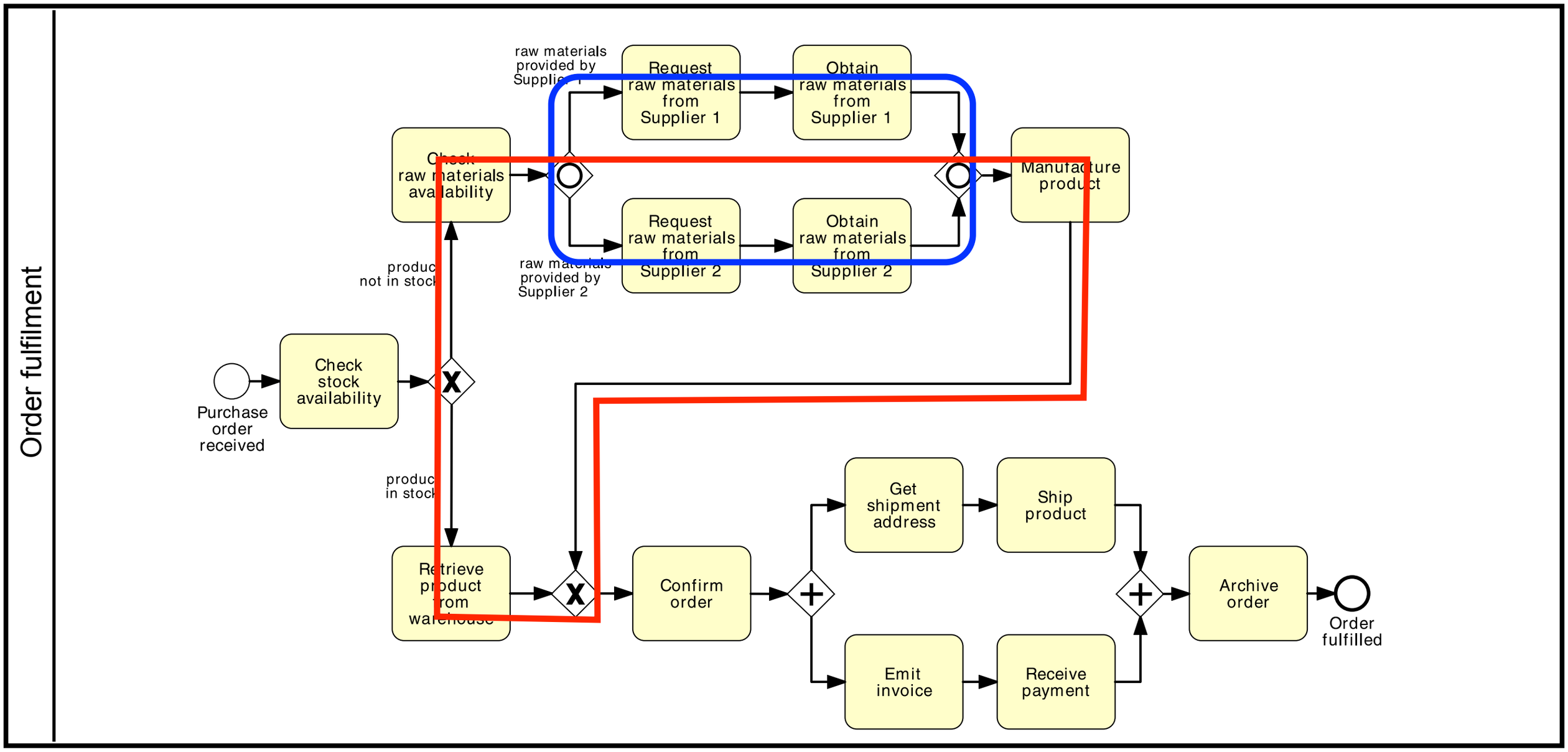
Use OR-gateways only when strictly necessary

XOR + AND + OR: order fulfilment



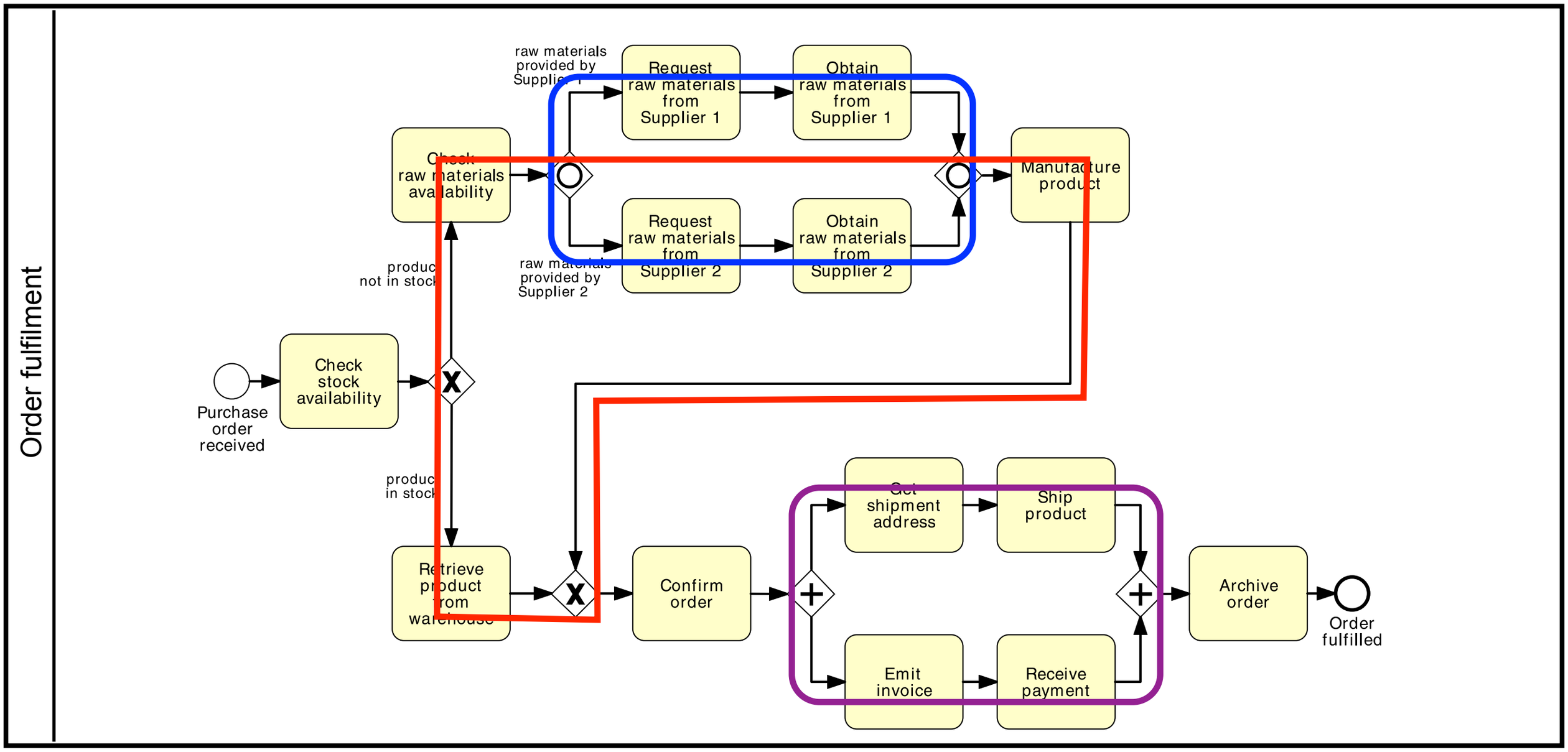
Better if gateways are balanced

XOR + AND + OR: order fulfillment



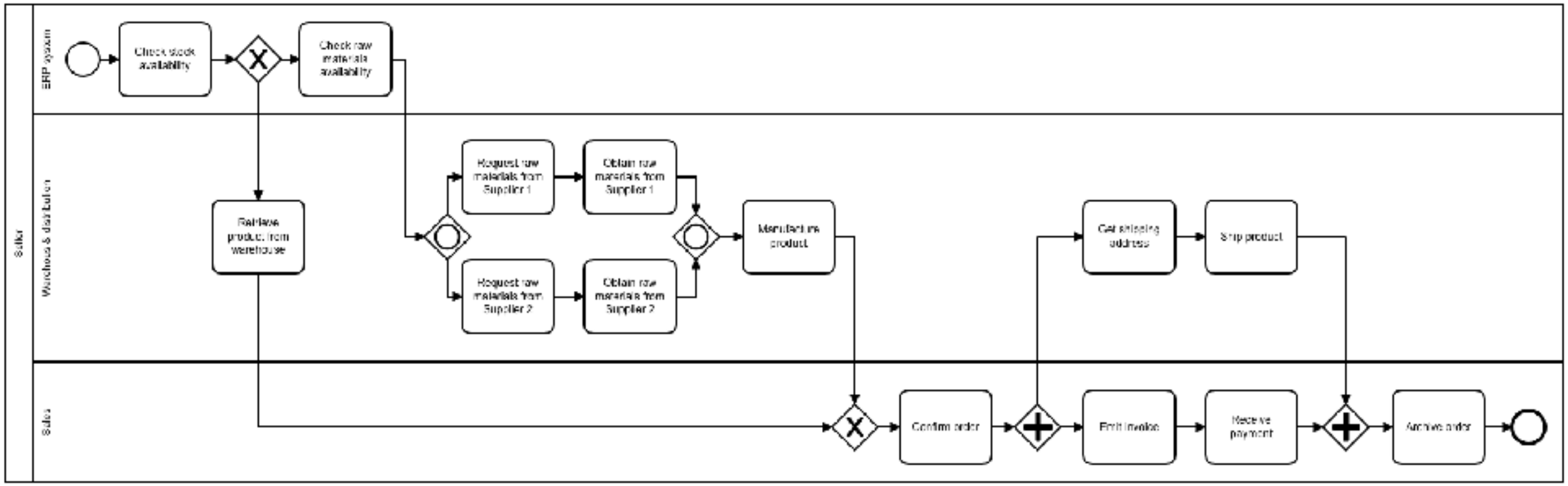
Better if gateways are balanced

XOR + AND + OR: order fulfillment

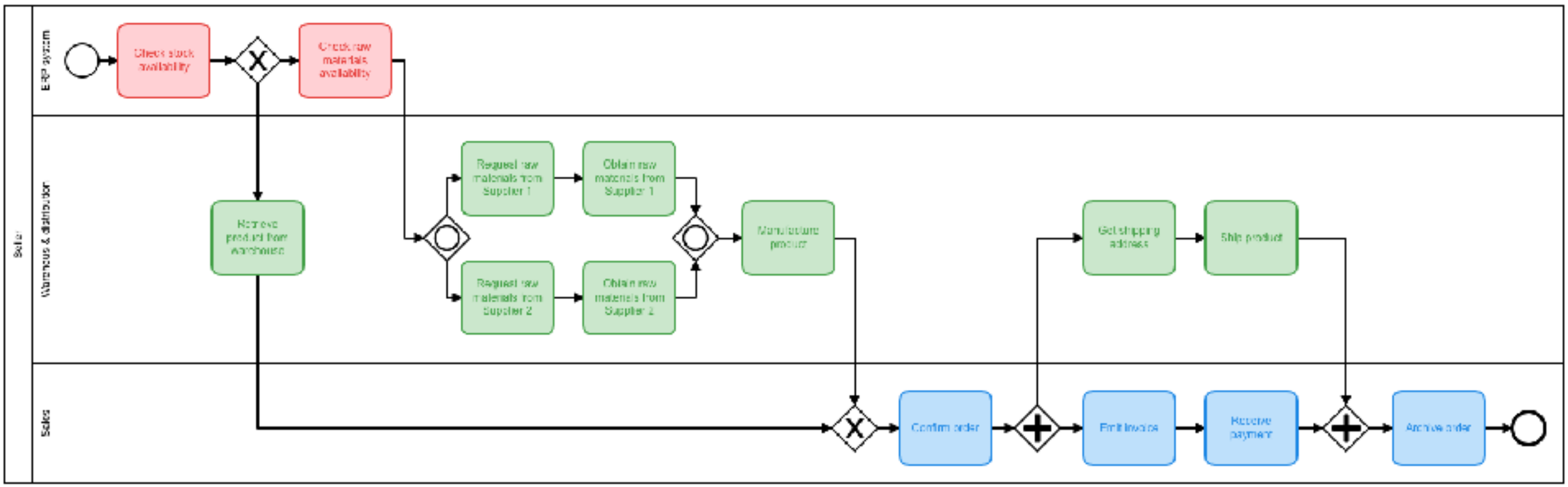


Better if gateways are balanced

Resources as lanes: order fulfillment



Resources as lanes: order fulfillment



Placing items in lanes

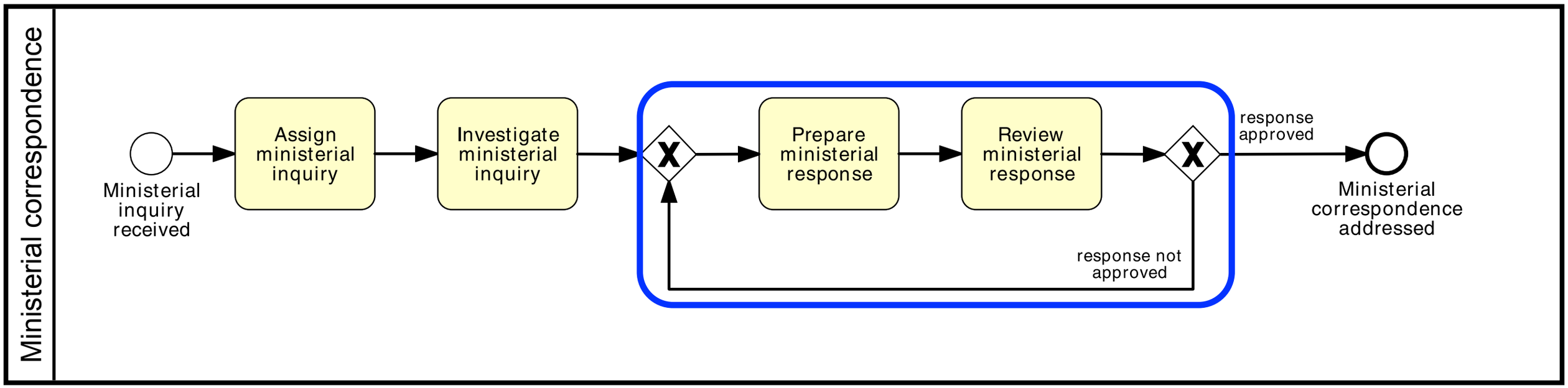
events: must be placed in the proper lane

activities: must be placed in the proper lane

gateways:
(X)OR-splits: same lane as preceding decision activity
AND-split: placement is irrelevant
(any kind of) **join**: placement is irrelevant

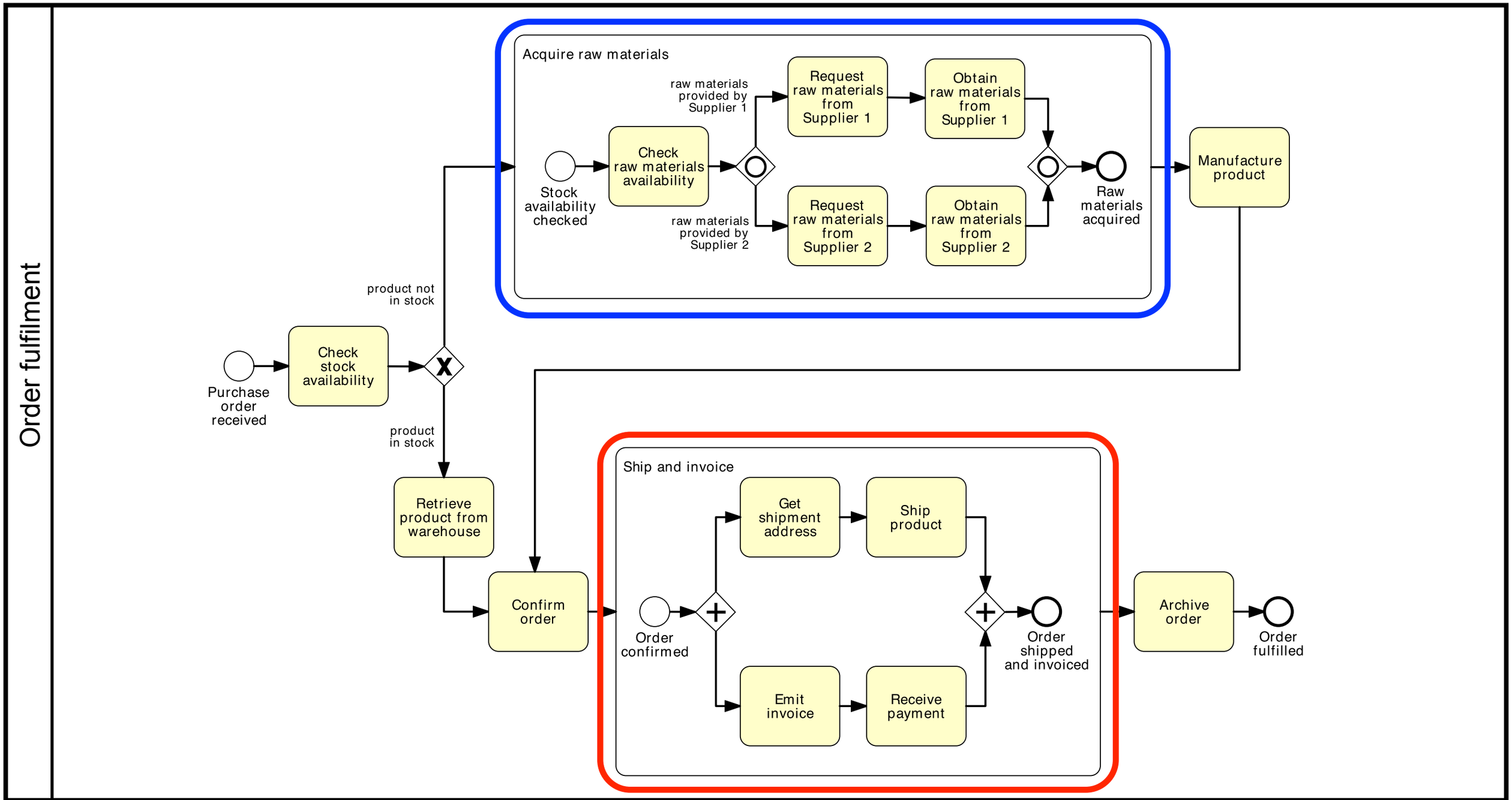
data-objects: placement is irrelevant

Rework and repetition: ministerial correspondence

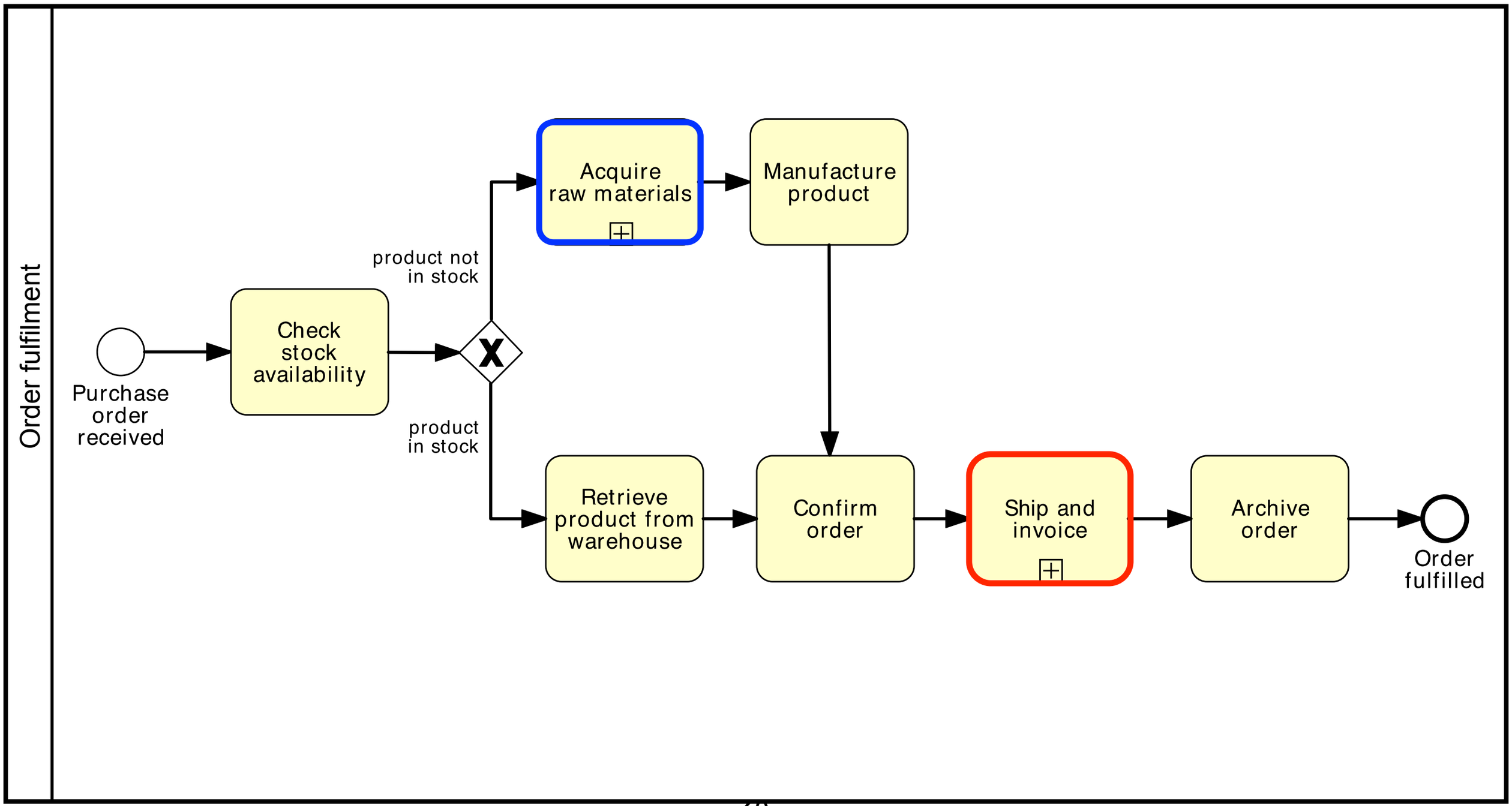


A repetition block starts with a XOR-join
and ends with a decision gateway (XOR-split)

Identify sub-processes



Collapsed sub-processes



2 - BPMN key features (with some examples)

Markers

(events, activities, gateways)

Activity types and markers

Internal markers indicate: the activity nature (**task type**) and the way it is executed (**activity marker**)

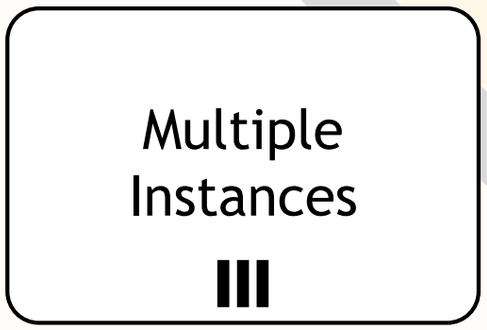
- Send Task
- Receive Task
- User Task
- Manual Task
- Business Rule Task
- Service Task
- Script Task

some types

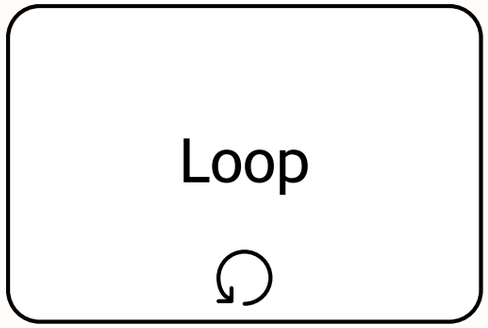
- Sub-Process Marker
- Loop Marker
- Parallel MI Marker
- Sequential MI Marker
- Ad Hoc Marker
- Compensation Marker

some markers

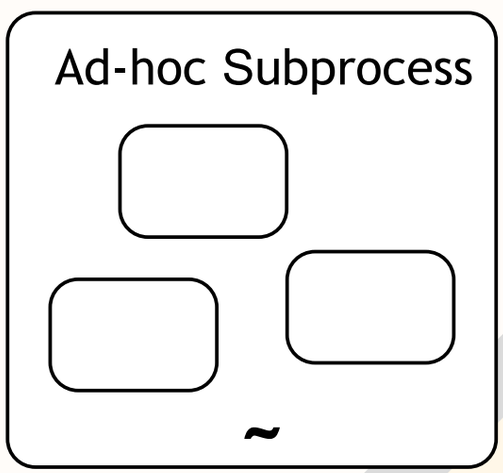
Some activity markers



Multiple Instances of the same activity are started in parallel or sequentially, e.g. for each line item in an order.

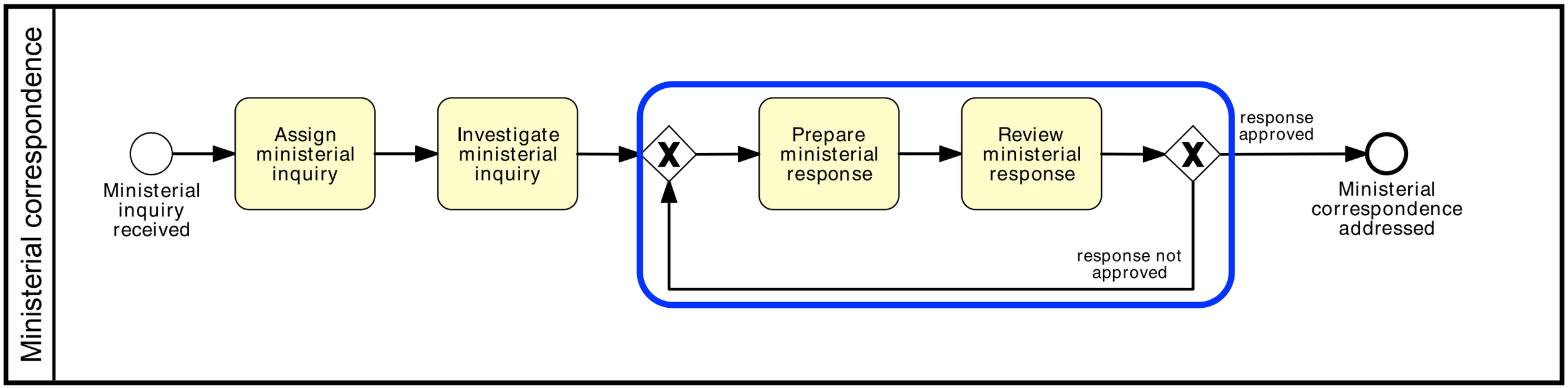


Loop Activity is iterated if a loop condition is true. The condition is either tested before or after the activity execution.

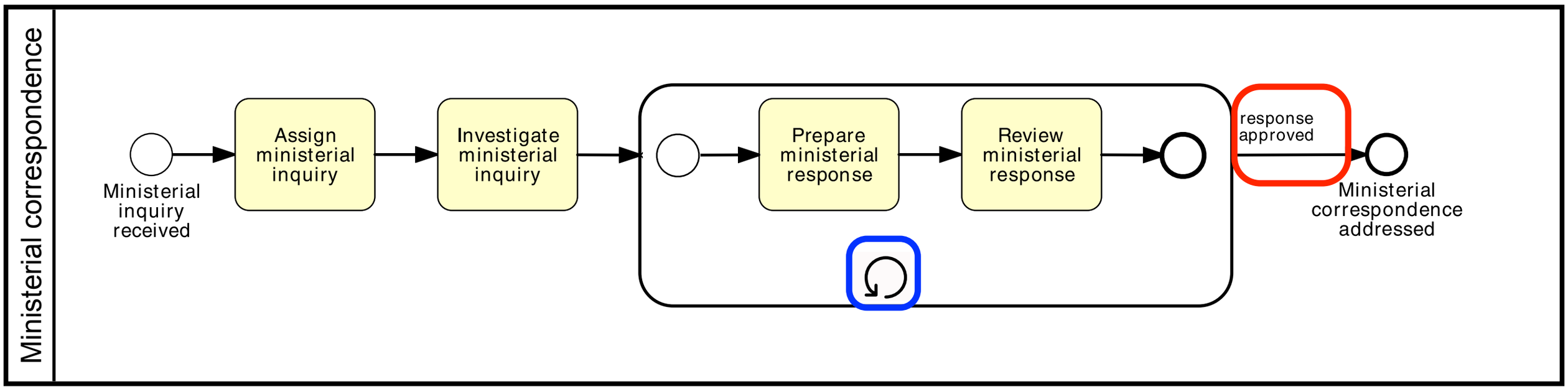


Ad-hoc Subprocesses contain tasks only. Each task can be executed arbitrarily often until a completion condition is fulfilled.

Loop marker: ministerial correspondence



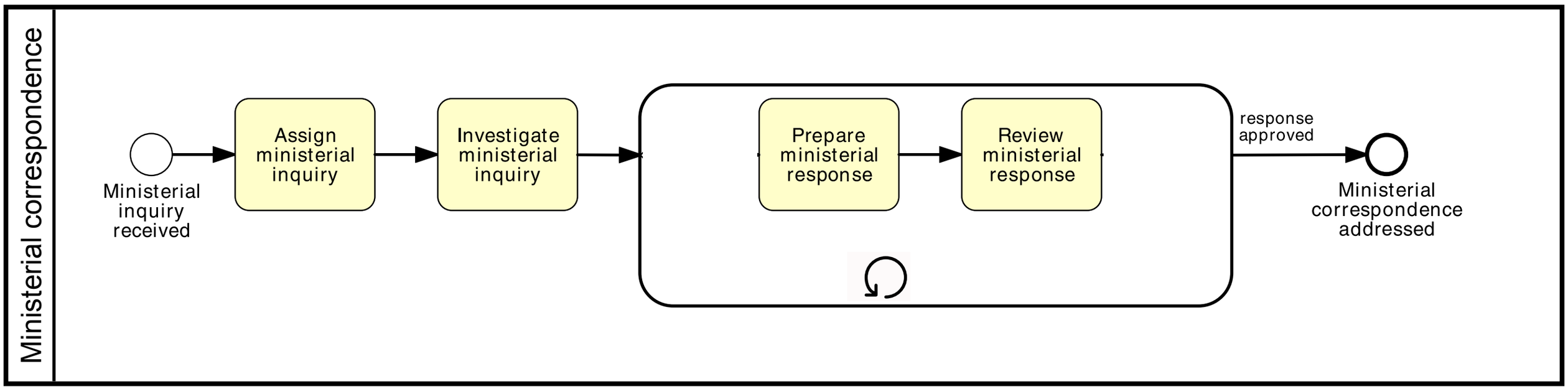
Loop marker: ministerial correspondence



the loop-symbol decoration marks the possible repetition of the sub-process activity

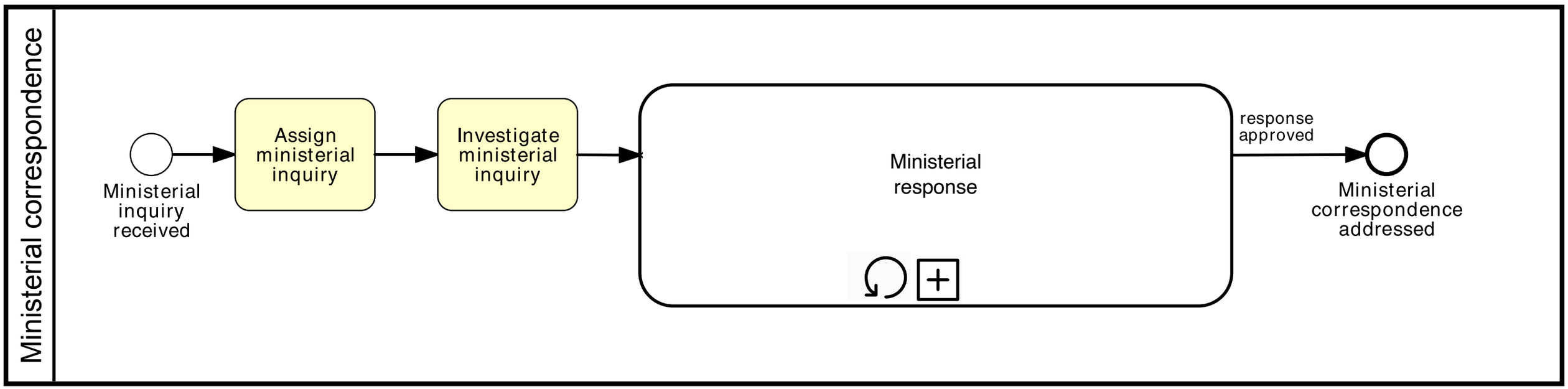
it is important to define exit conditions from loops!

Loop marker: ministerial correspondence



we can further simplify the inner process
(implicit start / end)

Loop marker: ministerial correspondence

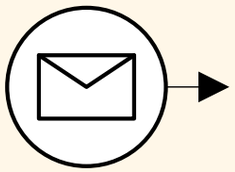


we can hide internal details

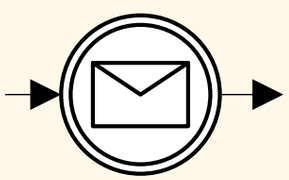
Catching and throwing

An event can catch a **trigger** or throw a **result**
Internal markers denote the trigger or result

Catching

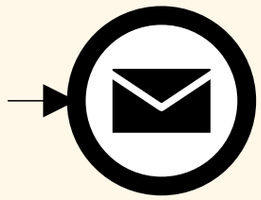


Start Event: Catching an event starts a new process instance.

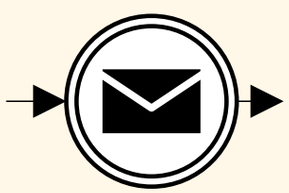


Intermediate Event (catching): The process can only continue once an event has been caught.

Throwing

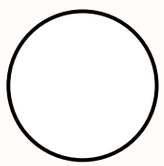
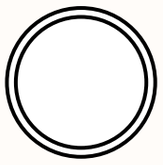
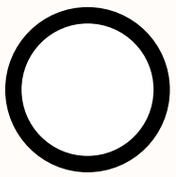


End Event: An event is thrown when the end of the process is reached.

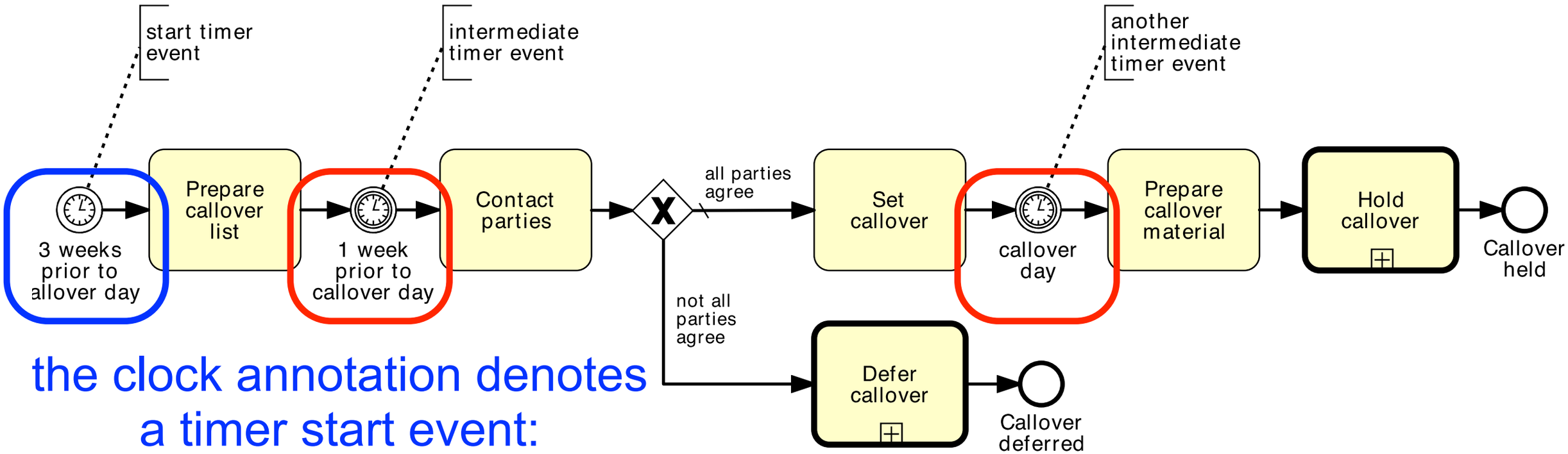


Intermediate Event (throwing): An event is thrown and the process continues.

Some internal markers

	Start	Intermediate		End	
	Catching		*Throwing*		
Plain					Untyped events, typically showing where the process starts or ends.
Message					Receiving and sending messages.
Timer					Cyclic timer events, points in time, time spans or timeouts.
Error					Catching or throwing named errors.
Terminate					Triggering the immediate termination of a process.

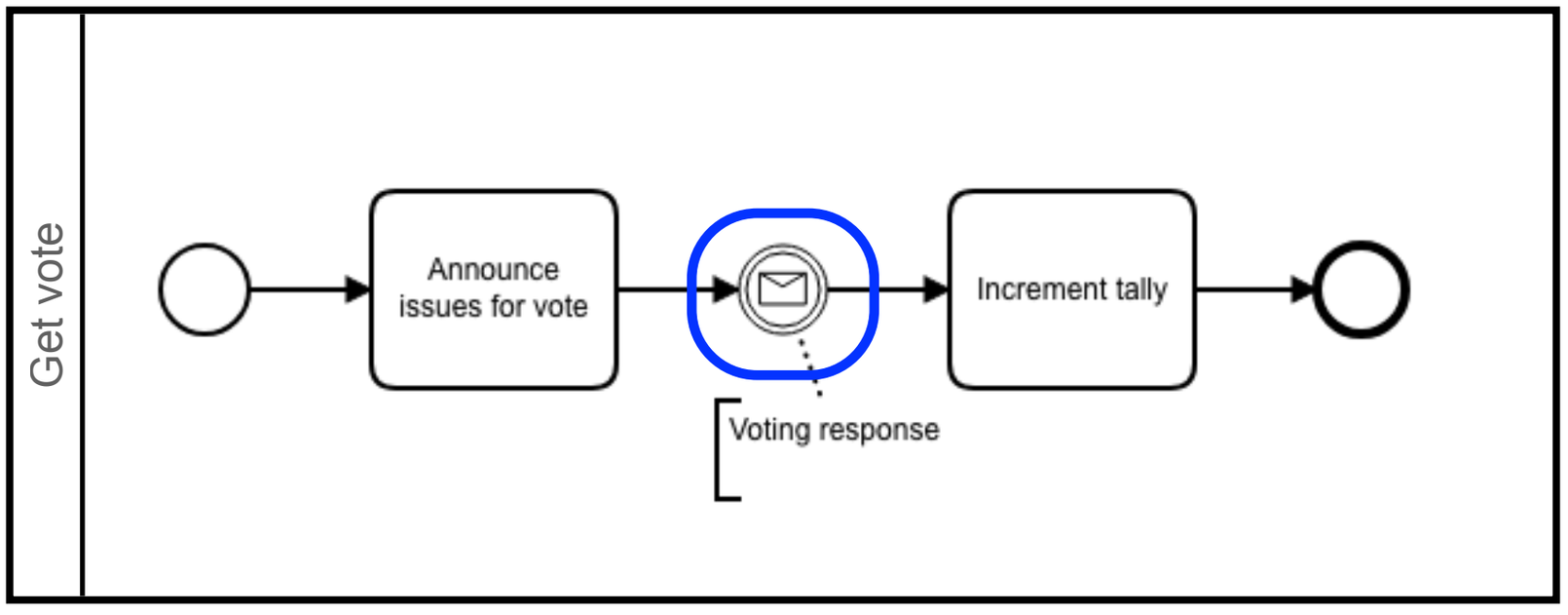
Timer events: small claims tribunal



the clock annotation denotes a timer start event: an instance of the process is created when some temporal event happens

the clock annotation denotes a timer intermediate event: the process is blocked until a time-out expires

Process break (event waiting)

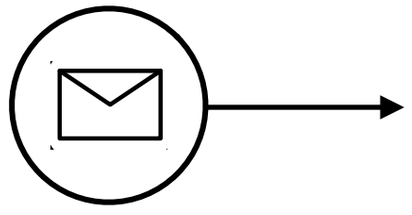


the envelope annotation denotes an intermediate message event: it signals the receipt of a message

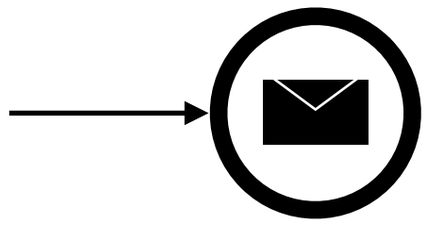
Collaboration diagrams (and message passing)

Message annotated events and activities

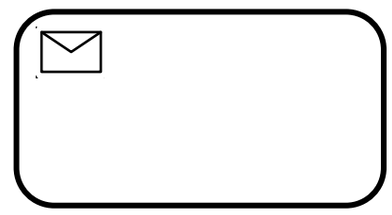
A start event can be annotated with a white-envelope:
a process instance is created
when a certain message is received



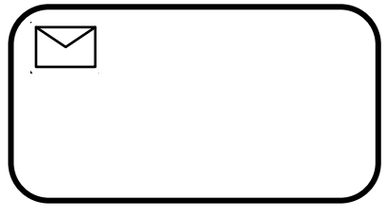
An end event can be annotated with a black-filled envelope:
when the process ends
a message is sent



Intermediate events and activities can be annotated
with both kinds of envelope
white = receipt of a message,
black = the sending of a message



Events vs Activities

Should we use  or  ?

No clear distinction is made, but typically

events are instantaneous

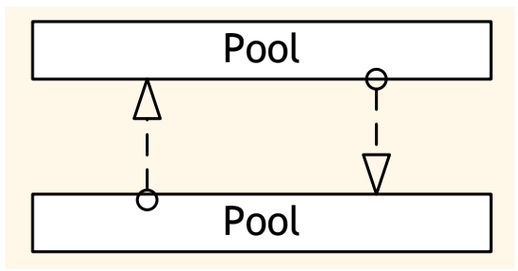
activities take time (have a duration)

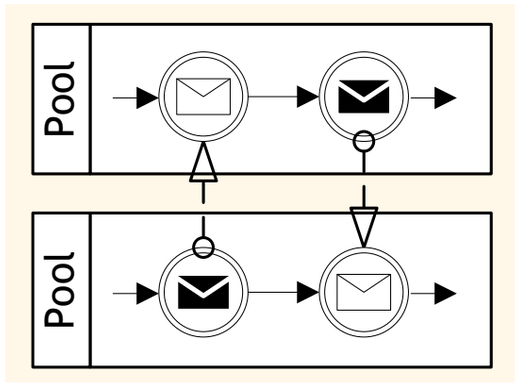
Collaboration

A **collaboration** contains two or more pools,
each representing a **participant** in the collaboration

A pool may be collapsed or exhibit the process within

Each possible communication corresponds to
a **message flow** between pools
(or between objects from different pools)

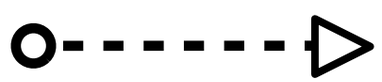




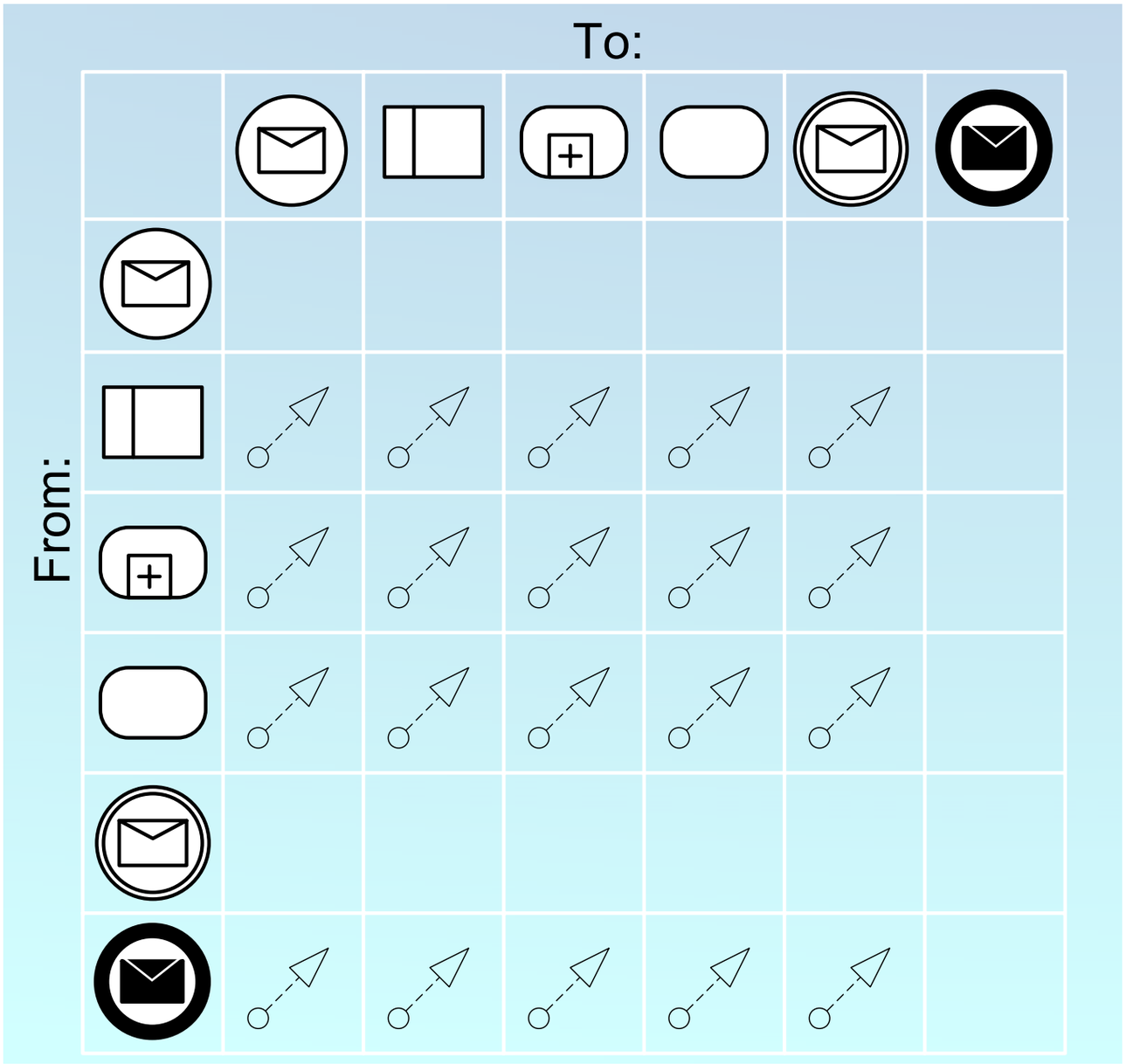
Message flow

A **message flow** represents communications (send/receive) between two separate participants (business entities or business roles)

A message flow is represented by a dashed line with a open arrowheads



Message flow constraints



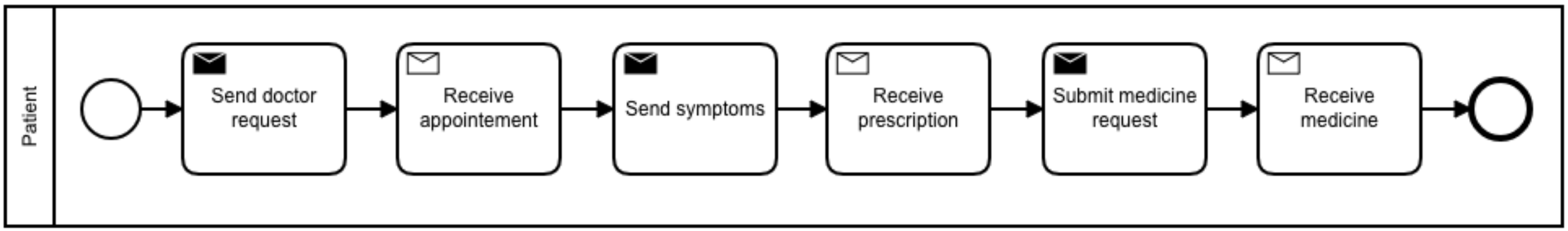
each event:
at most one message flow

each activity:
at most one message flow

each gateway:
no message flow!

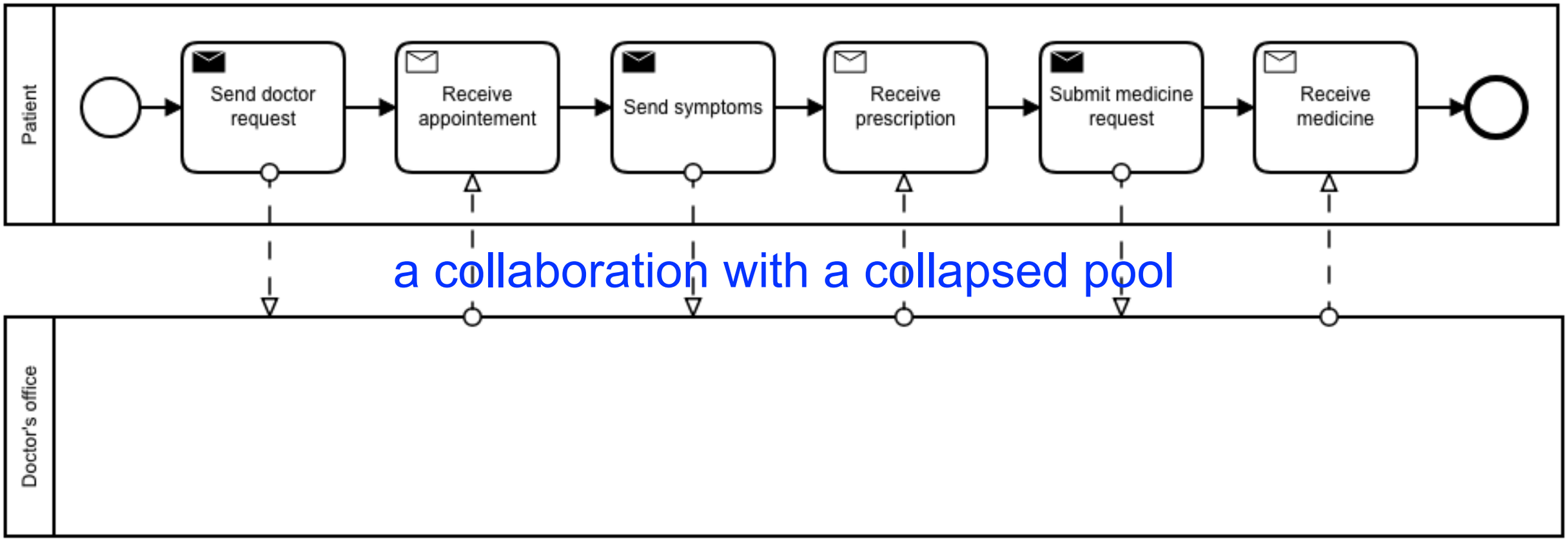
each pool:
any number of message flows

From processes to collaborations



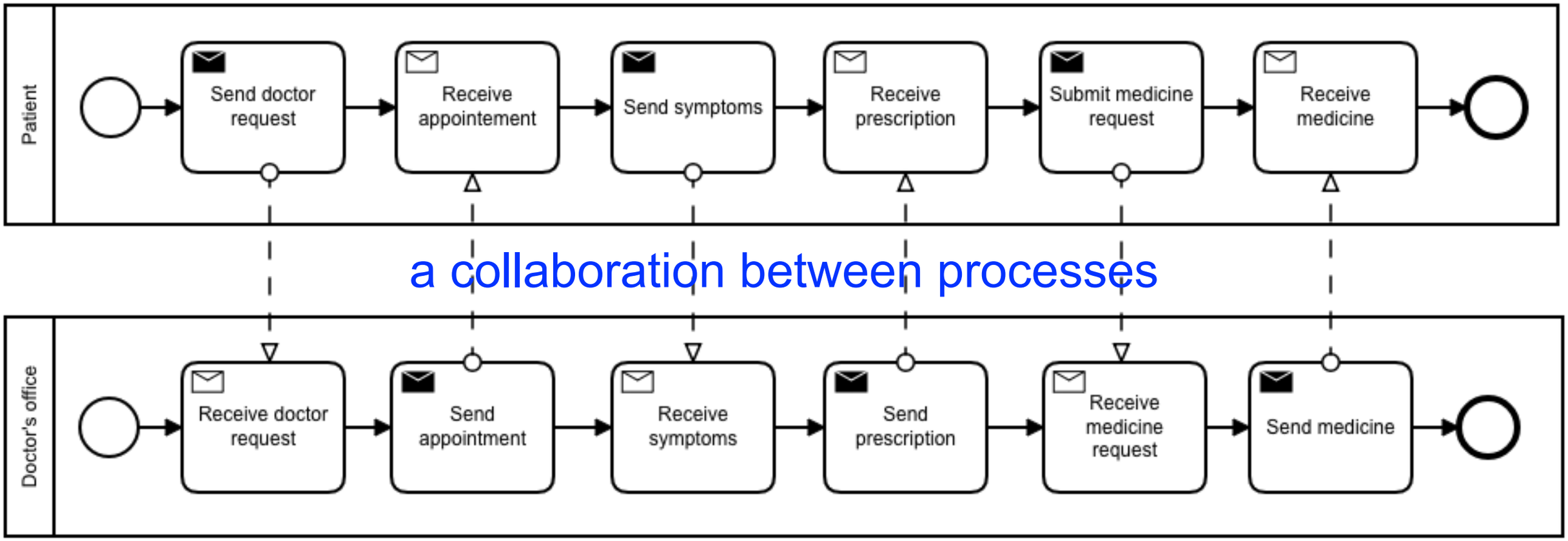
a stand-alone process

From processes to collaborations



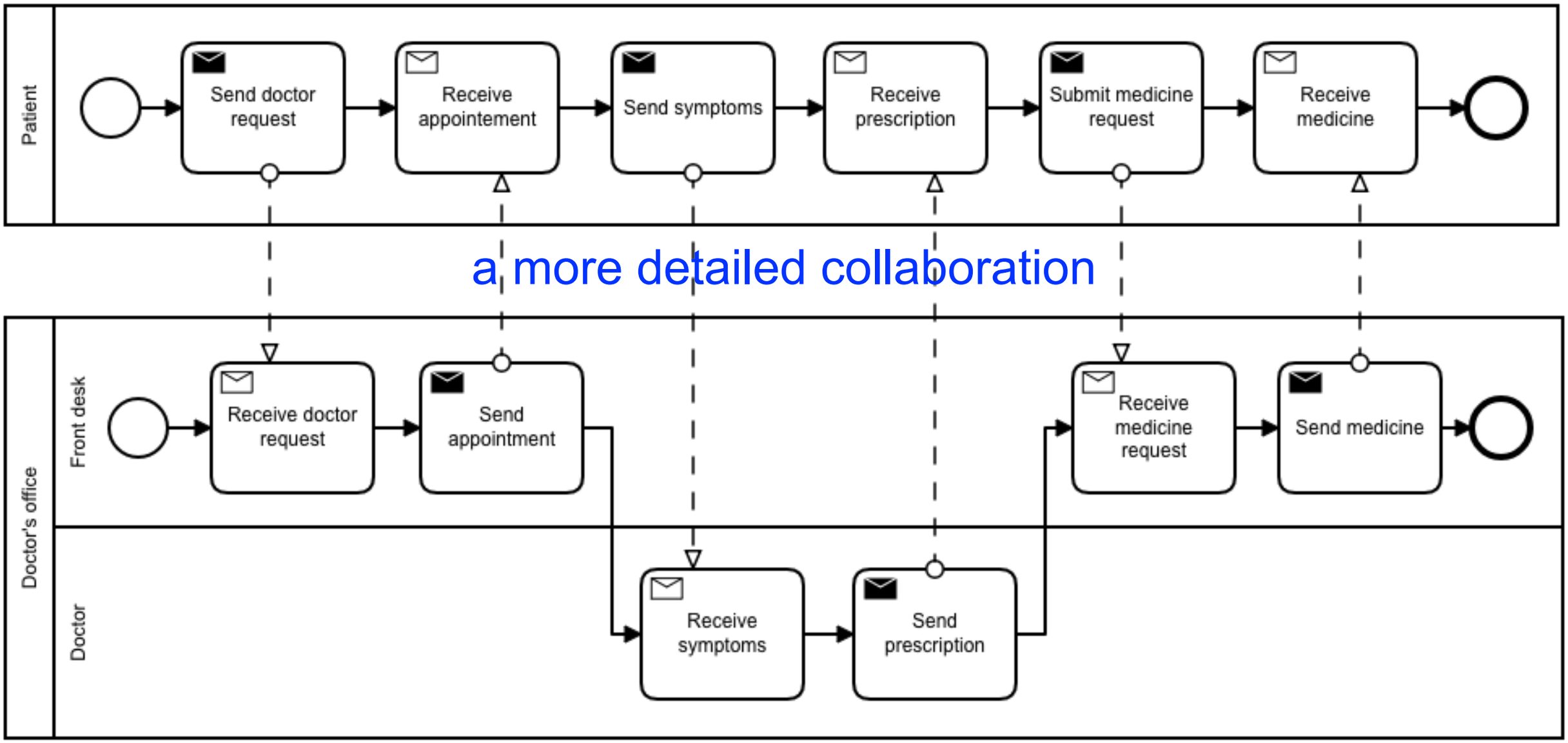
a collaboration with a collapsed pool

From processes to collaborations



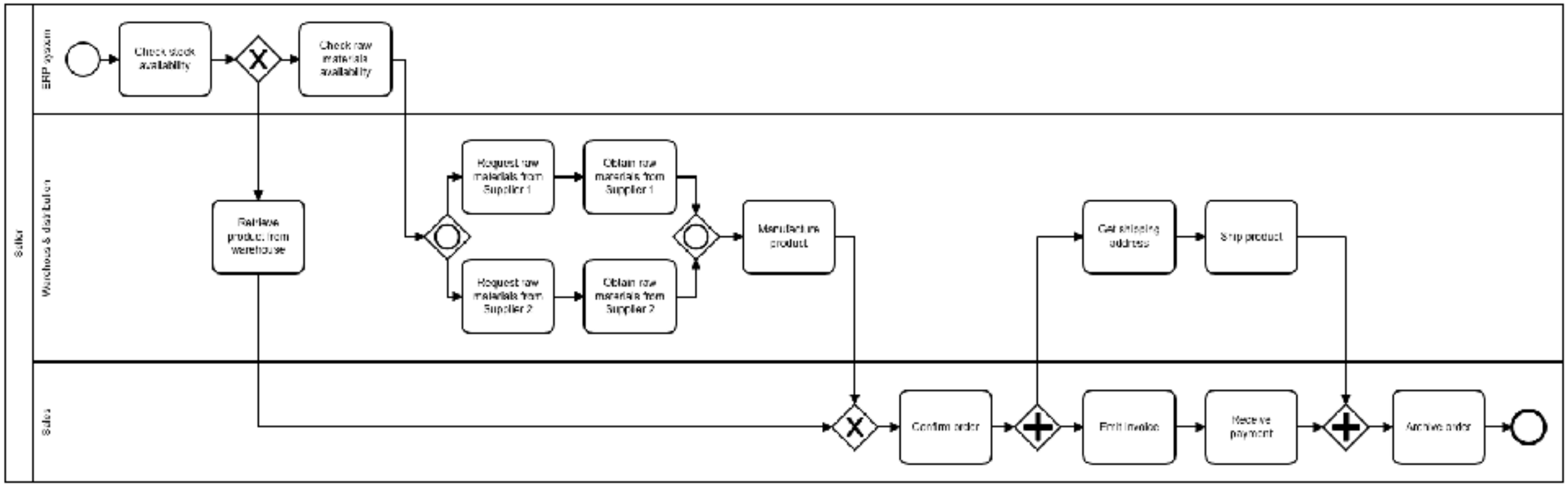
a collaboration between processes

From processes to collaborations

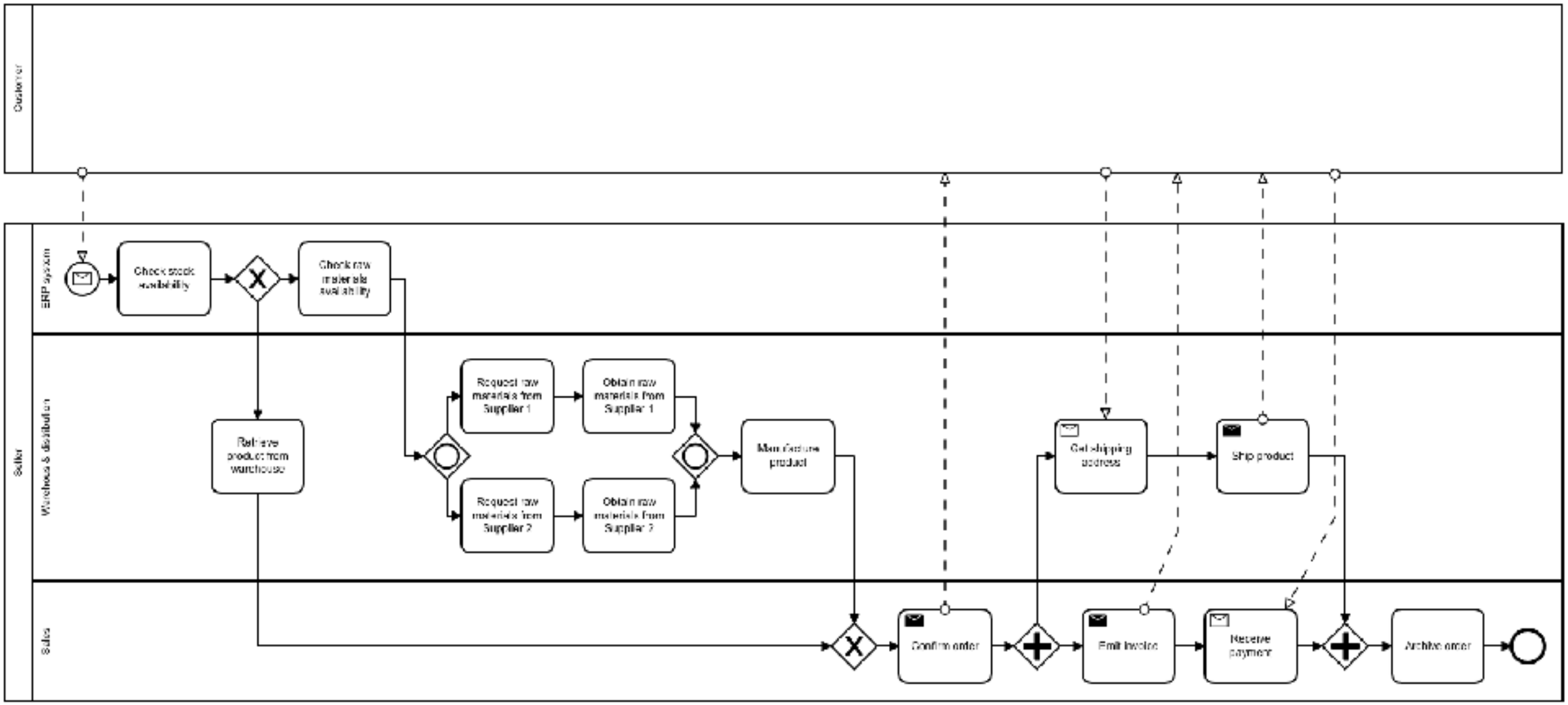


a more detailed collaboration

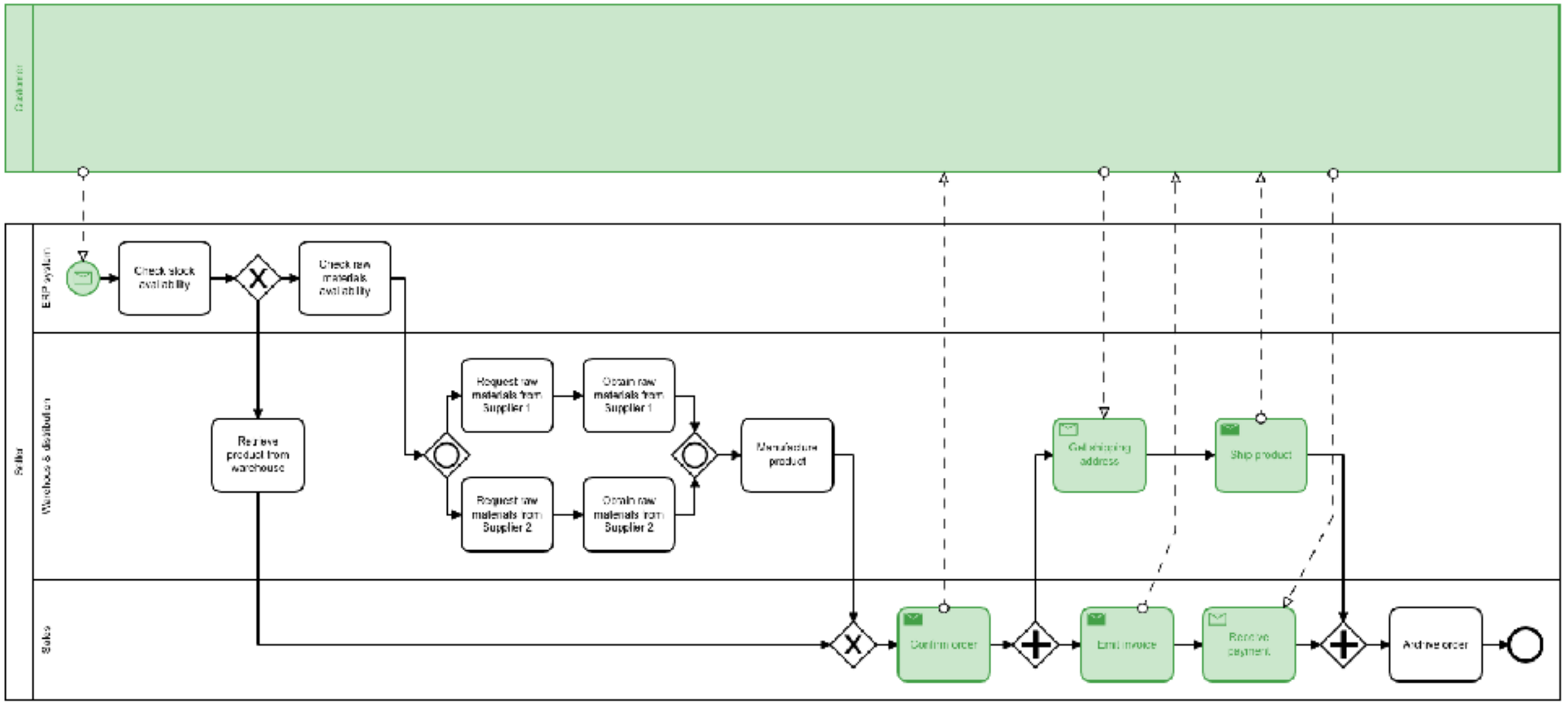
Example: Seller



Example: Seller & Customer



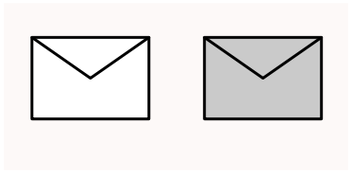
Example: Seller & Customer



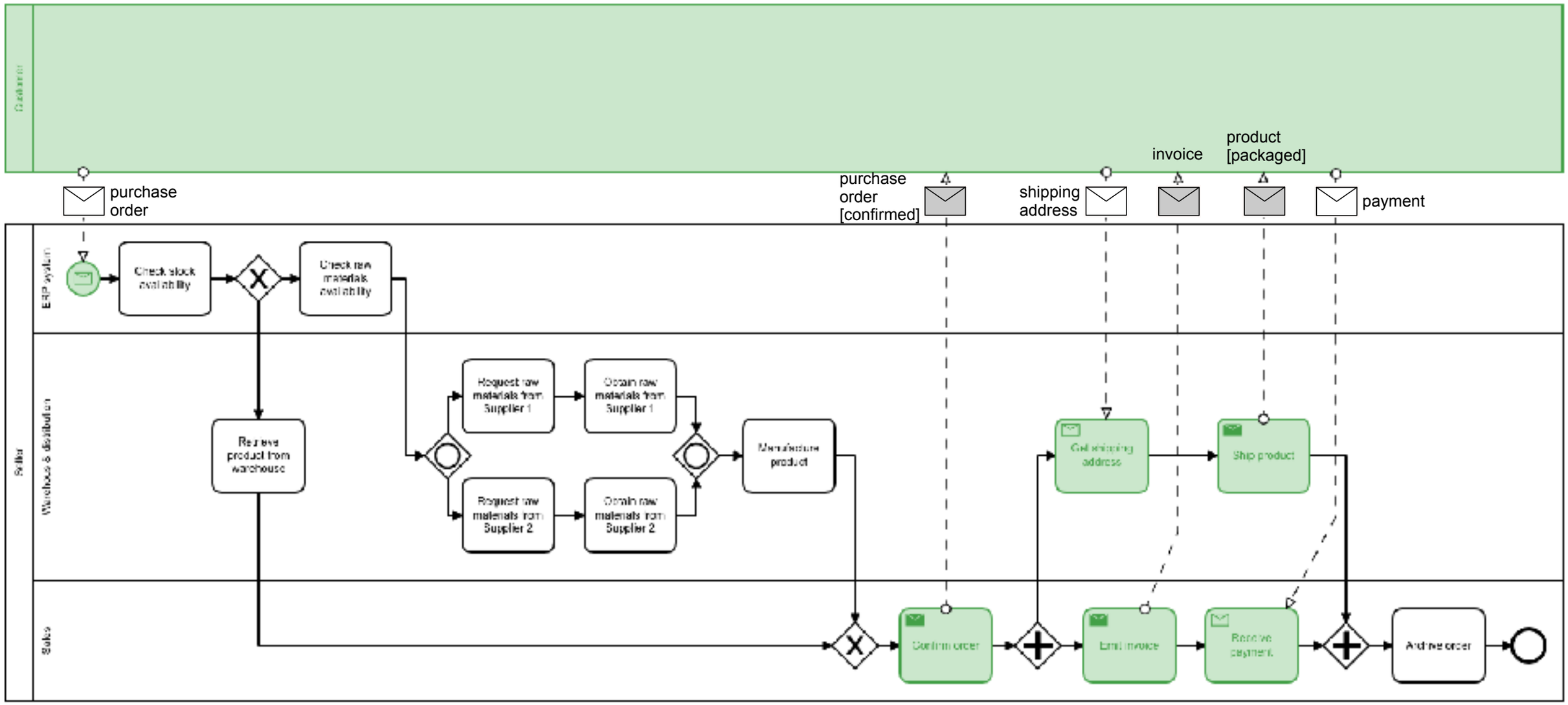
Artefacts: message data objects

A **message data object** depicts the data that are communicated between two participants

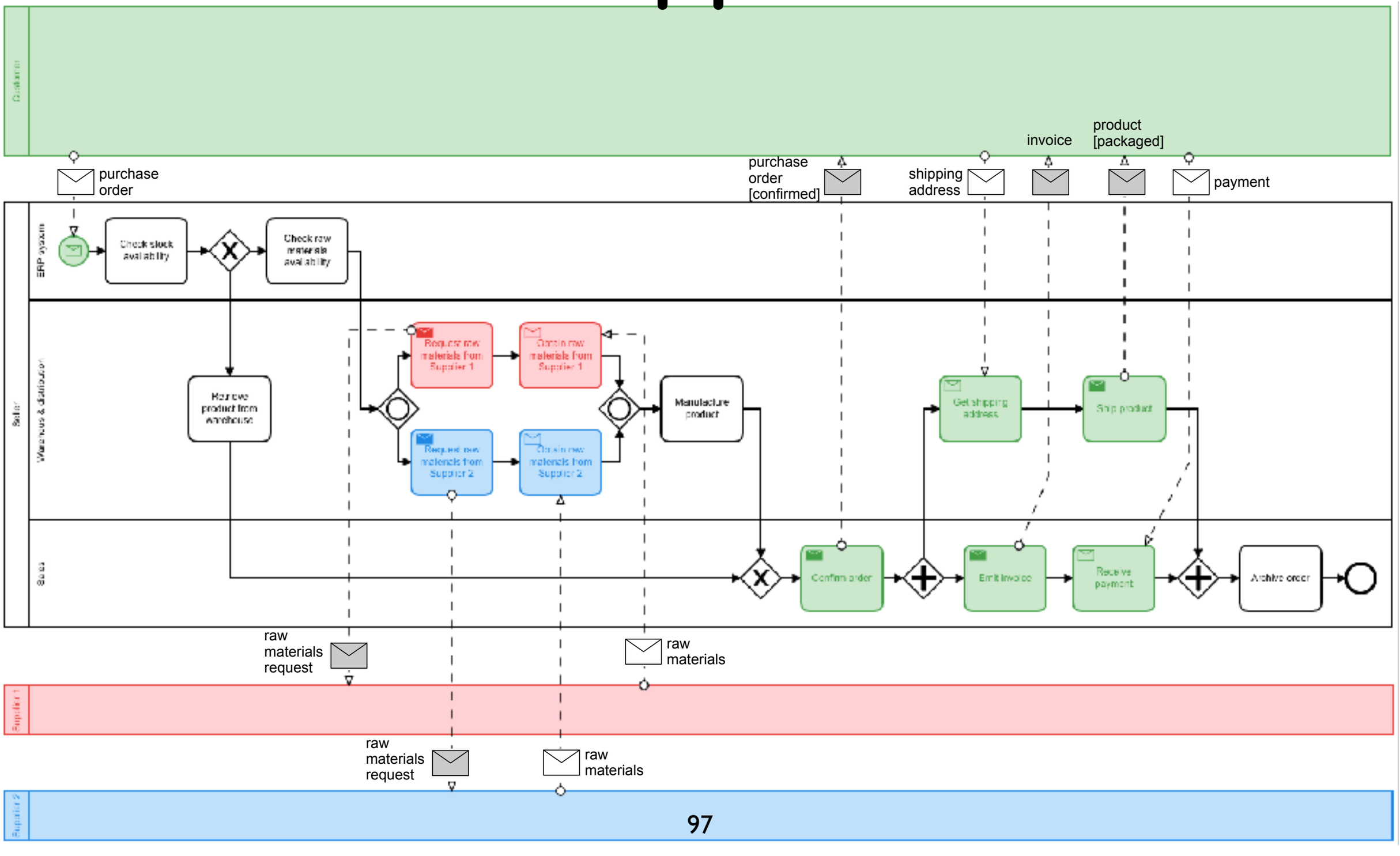
A message data object is represented as an envelope



Example: Seller & Customer

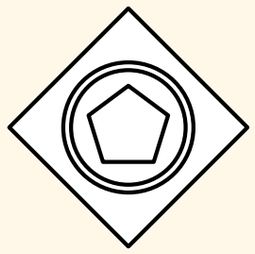


Example: Seller, Customer & Suppliers

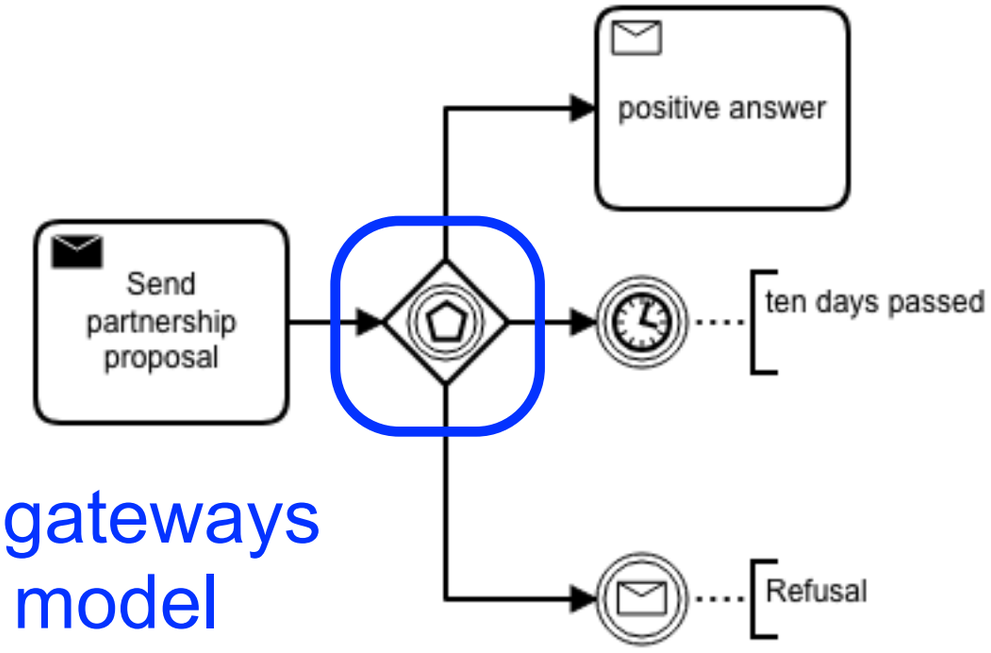


Deferred choice (event based decisions)

Event-based decisions

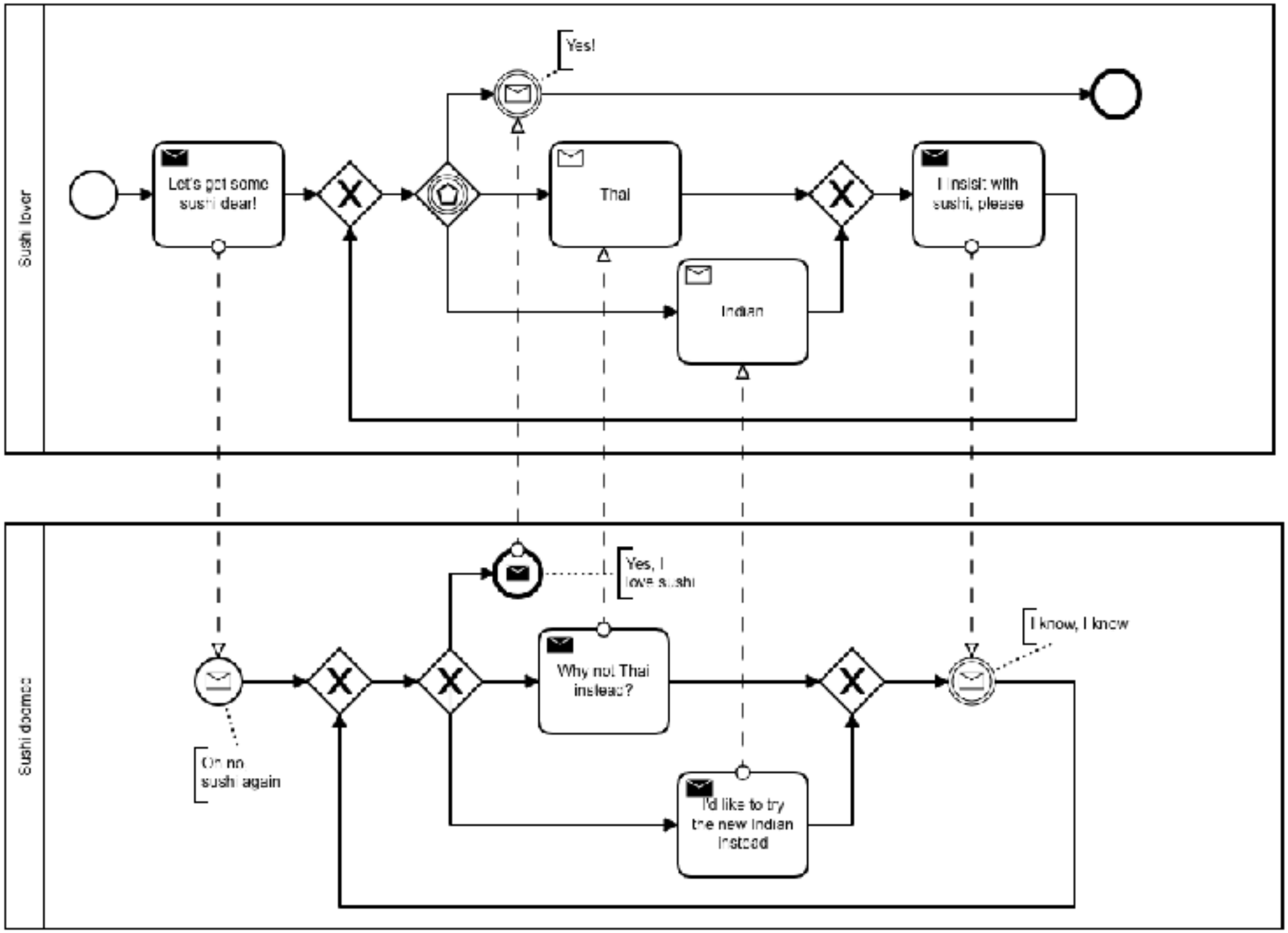


Event-based Exclusive Gateway
Is always followed by catching events or receive tasks. Sequence flow is routed to the subsequent event/task which happens first.



Event-based (split) gateways must be used to model decisions that depends on some external event

A negotiation without choice



Some remarks

Lanes are often used to separate activities associated with a specific company function or role

Sequence flow cannot cross the boundaries of a pool (it can cross lanes in the pool)

Message flow cannot connect flow objects in the same pool

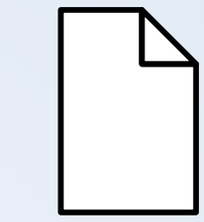
3 - more on BPMN (with some examples)

More artefacts (data-objects, groups)

Data object

A **data object** represents information flowing through the process, such as documents, emails and letters

A data object is often represented by the usual file icon



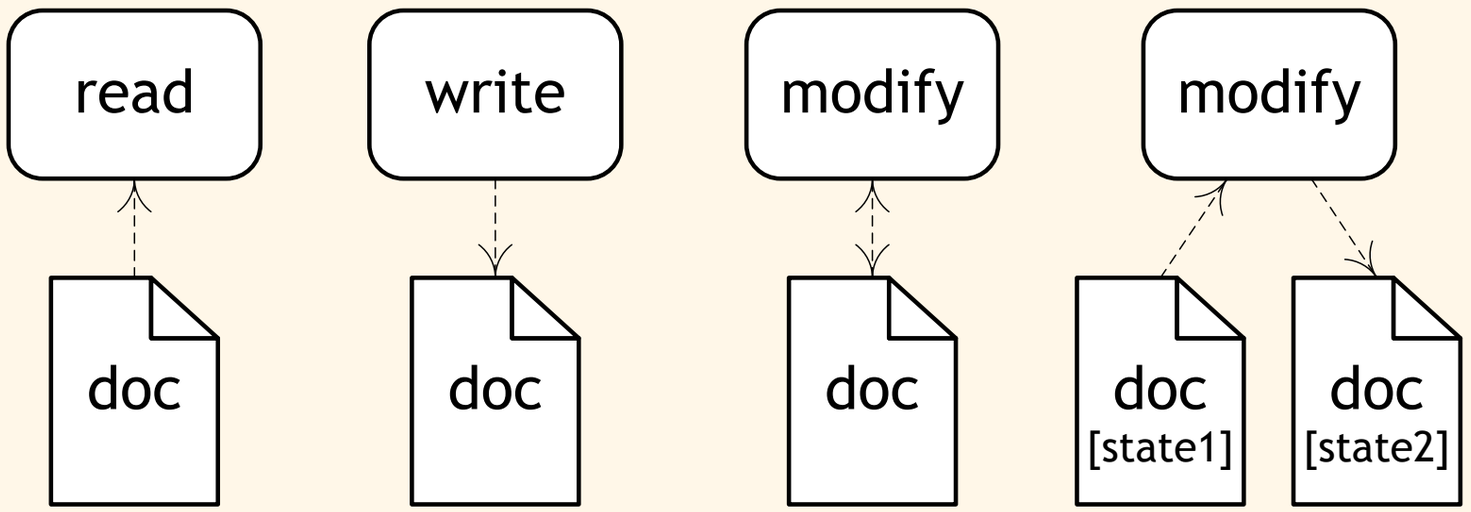
Data objects provide information about what activities are required to be triggered and/or what they produce. They are considered as Artefacts because they do not have any direct effect on the Sequence Flow or Message Flow of the Process. The state of the data object should also be set.

Association, again

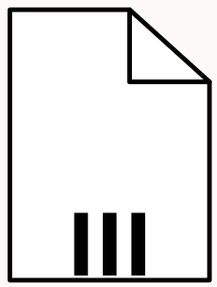
Attaching a data object with an **Undirected Association** to a sequence flow indicates hand-over of information between the activities involved.

A **Directed Association** indicates information flow. A data object can be read at the start of an activity or written upon completion.

A **Bidirected Association** indicates that the data object is modified, i.e. read and written during the execution of an actvity.



More data objects



A **Collection Data Object r**epresents a collection of information, e.g., a list of order items.

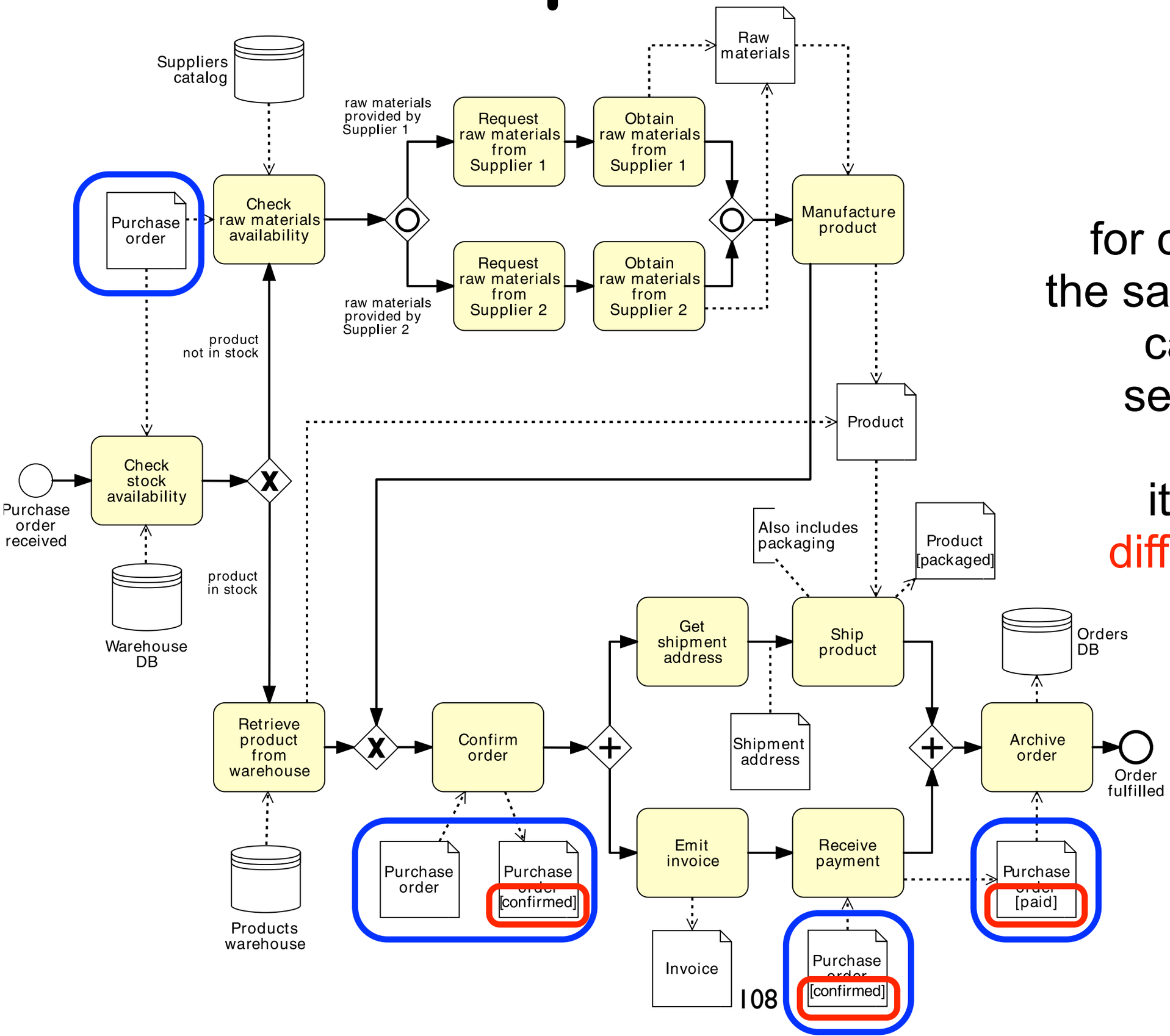
A **Data Store** is a place where the process can read or write data, e.g., a database or a filing cabinet. It persists beyond the lifetime of the process instance.

Group

An arbitrary set of objects can form a **group**
(if they logically belong together)
it has non behavioural effect (only documentation)

A group is represented by
rounded corner rectangles with dashed lines

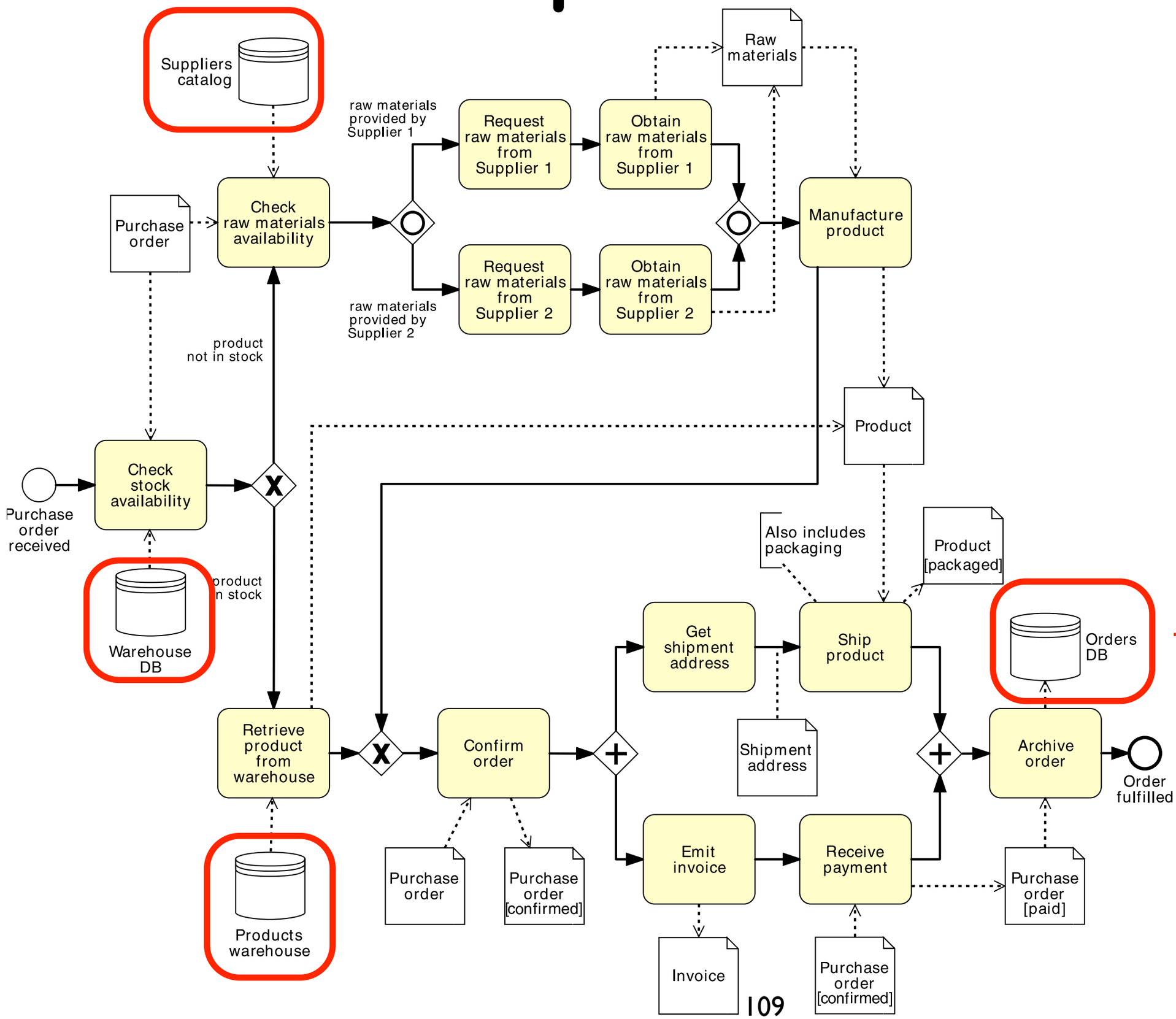
Example: artefacts



for convenience,
the same data object
can appear
several times

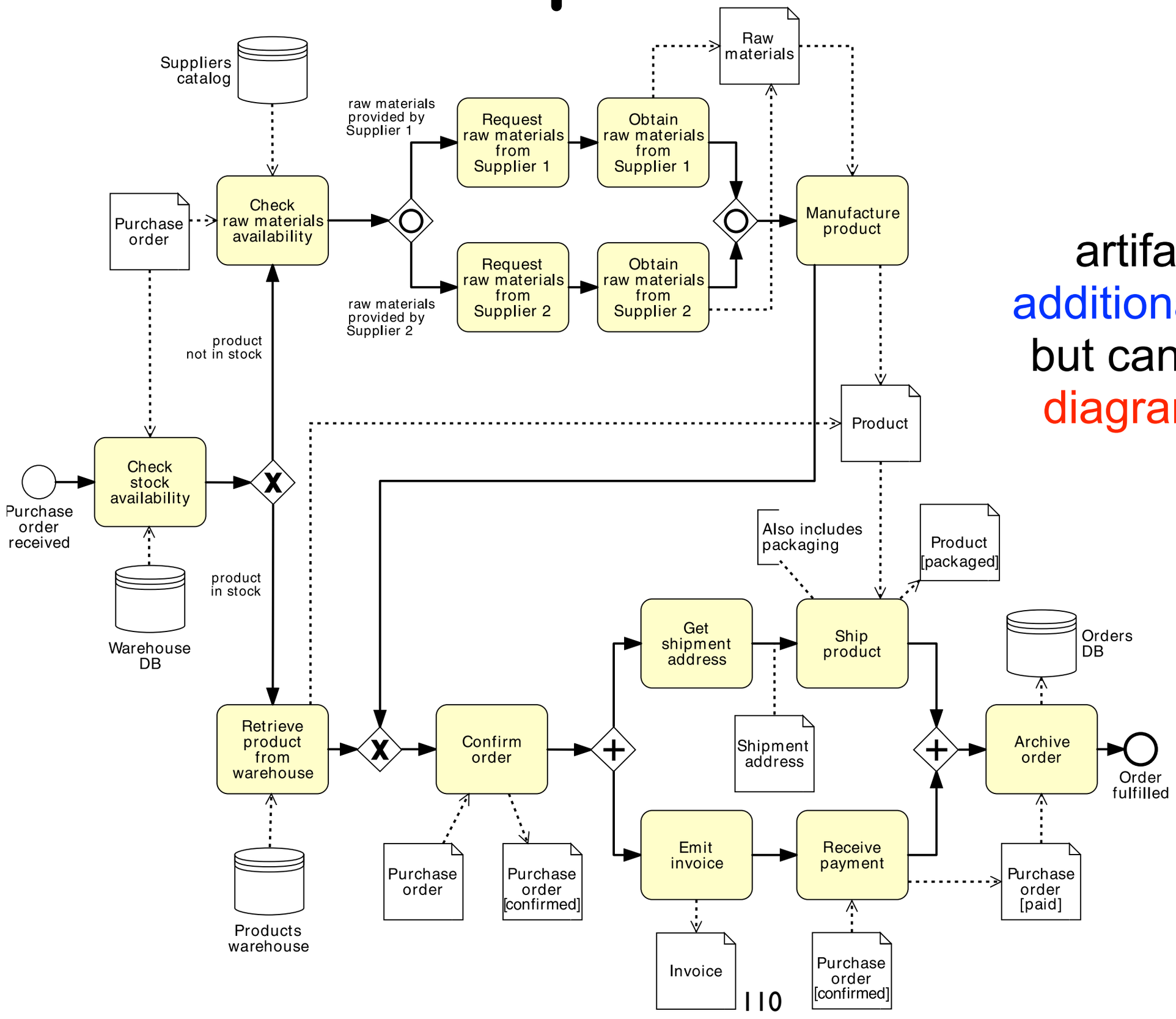
it can have
different states

Example: artefacts



data stores for persistent data objects

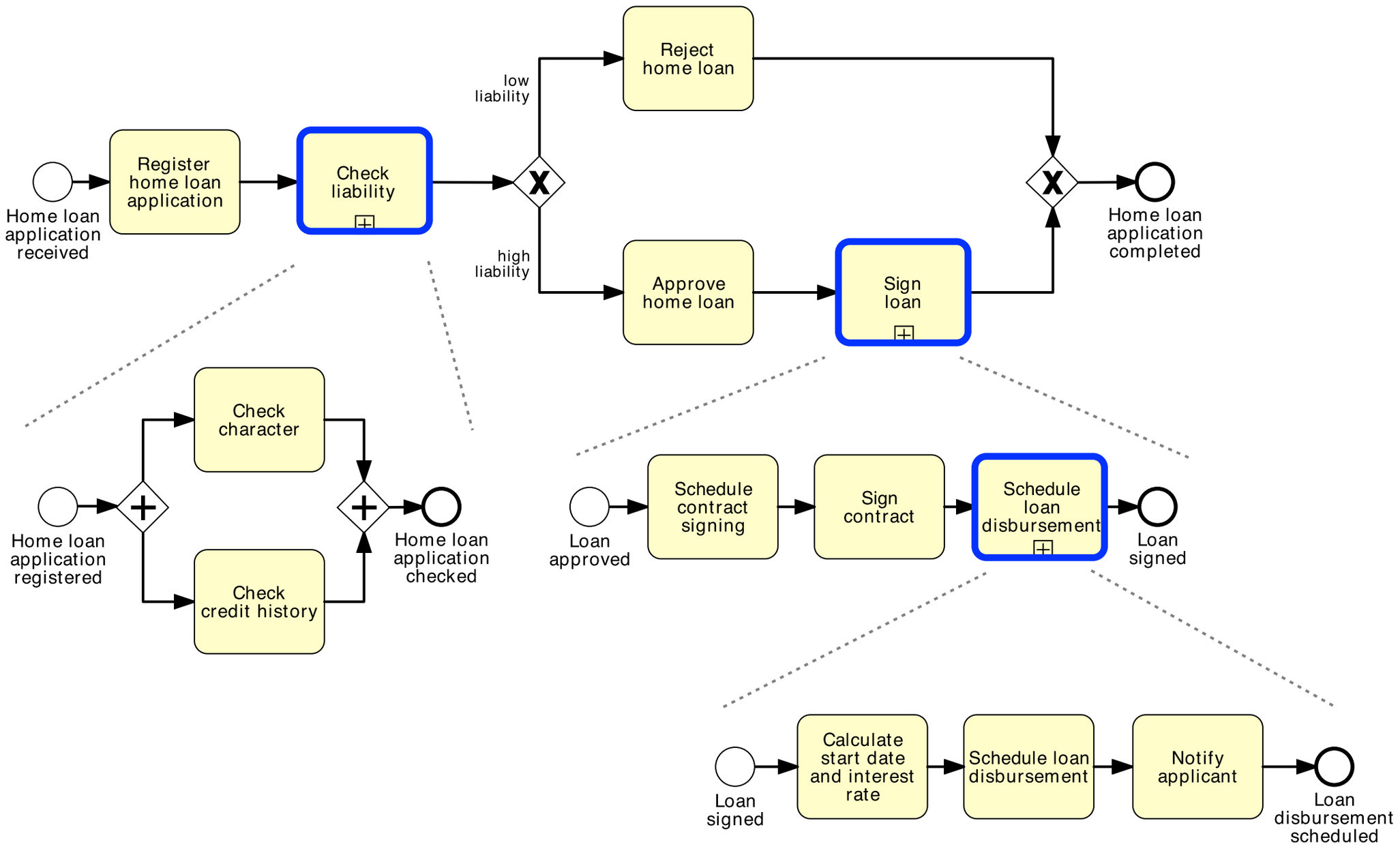
Example: artefacts



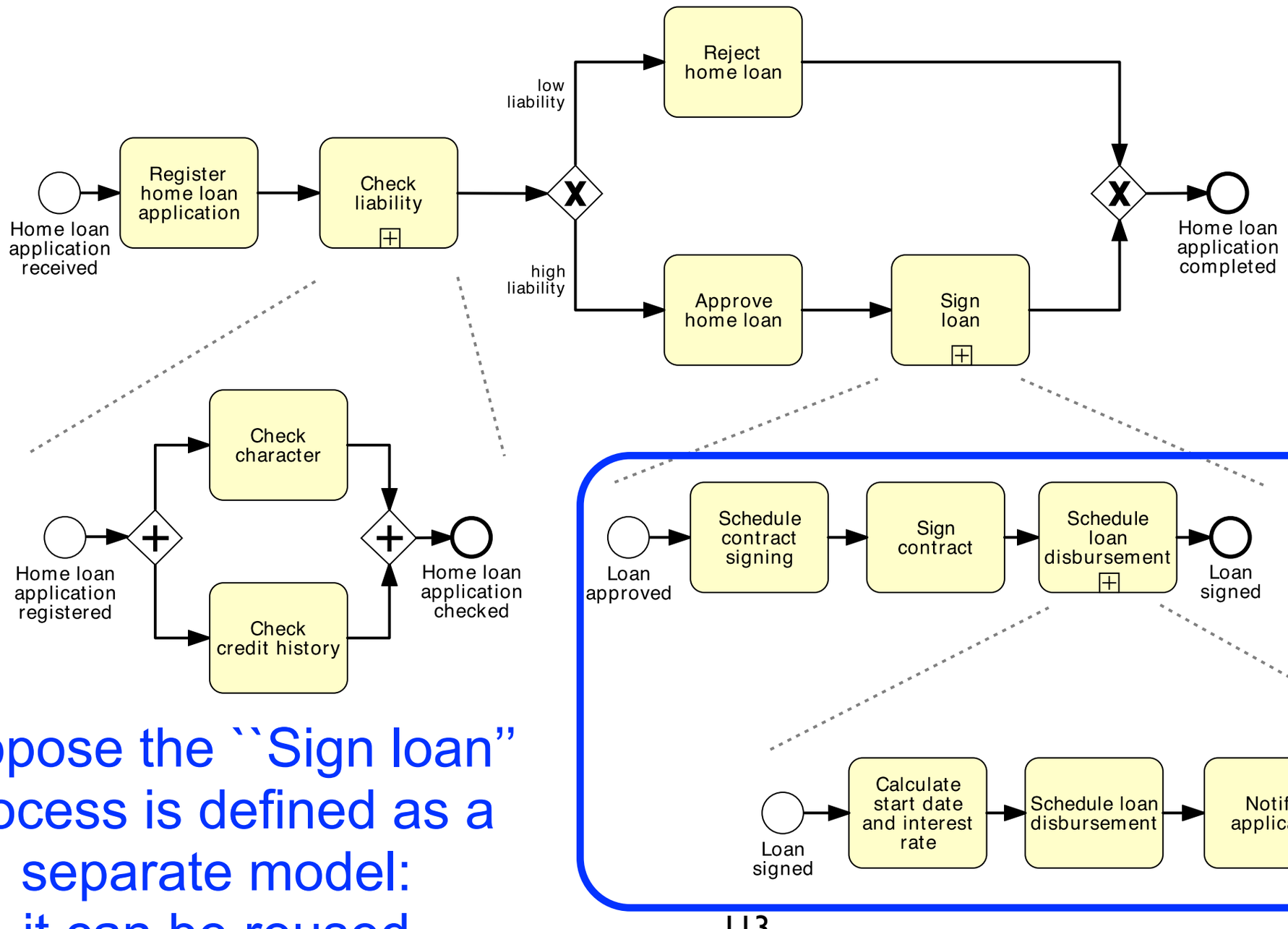
artifacts provide additional information, but can compromise diagram readability

Call activities

Nesting sub-processes: home loans



Global sub-processes: home / student loans

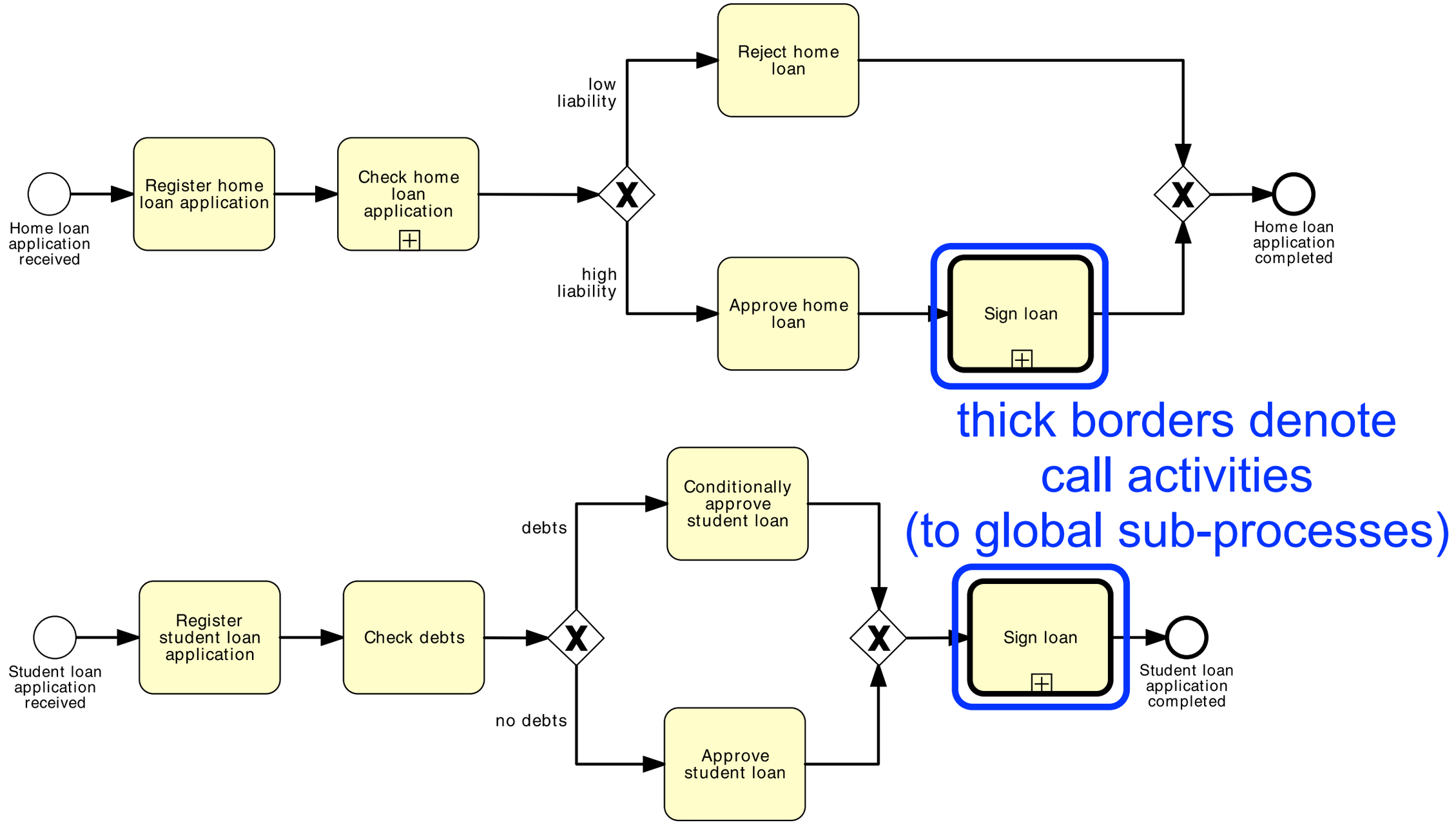


suppose the ``Sign loan'' process is defined as a separate model: it can be reused

Call activities

A **Call Activity** is a wrapper for a globally defined Sub-Process or Task that is reused in the current process.

Call activities: home / student loans



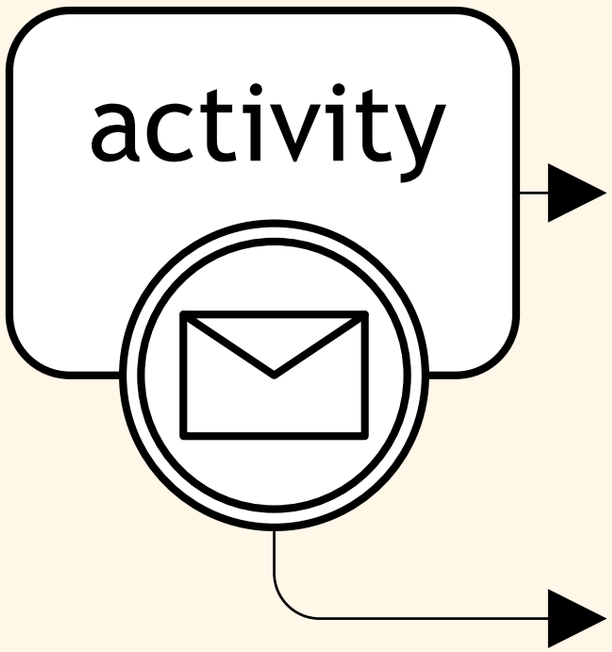
Global processes: advantages

Readability: processes tend to be smaller

Reusability: define once, use many time

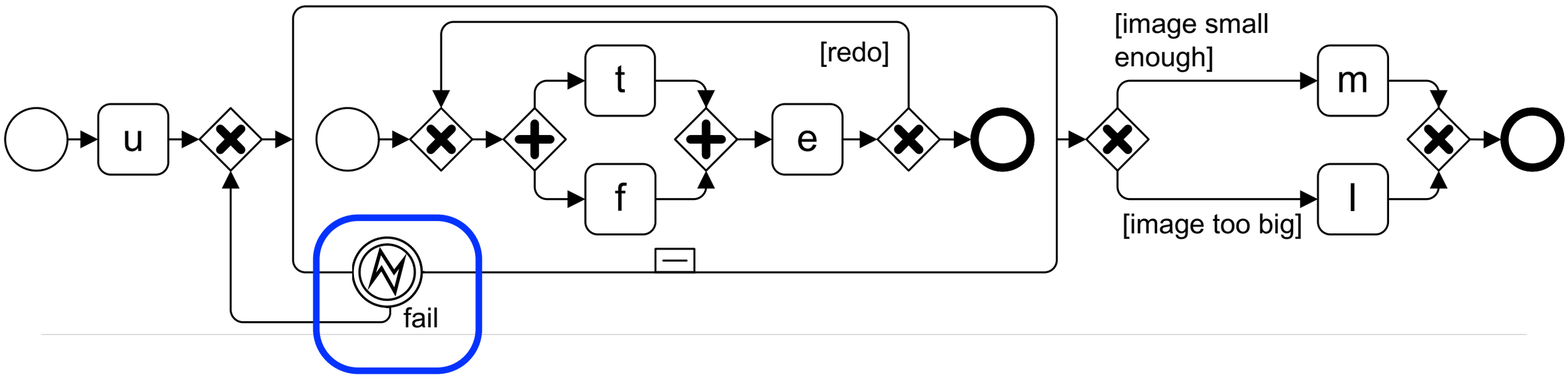
Sharing: any change made to a global process
is automatically propagated to all models that invoke it

Attached events



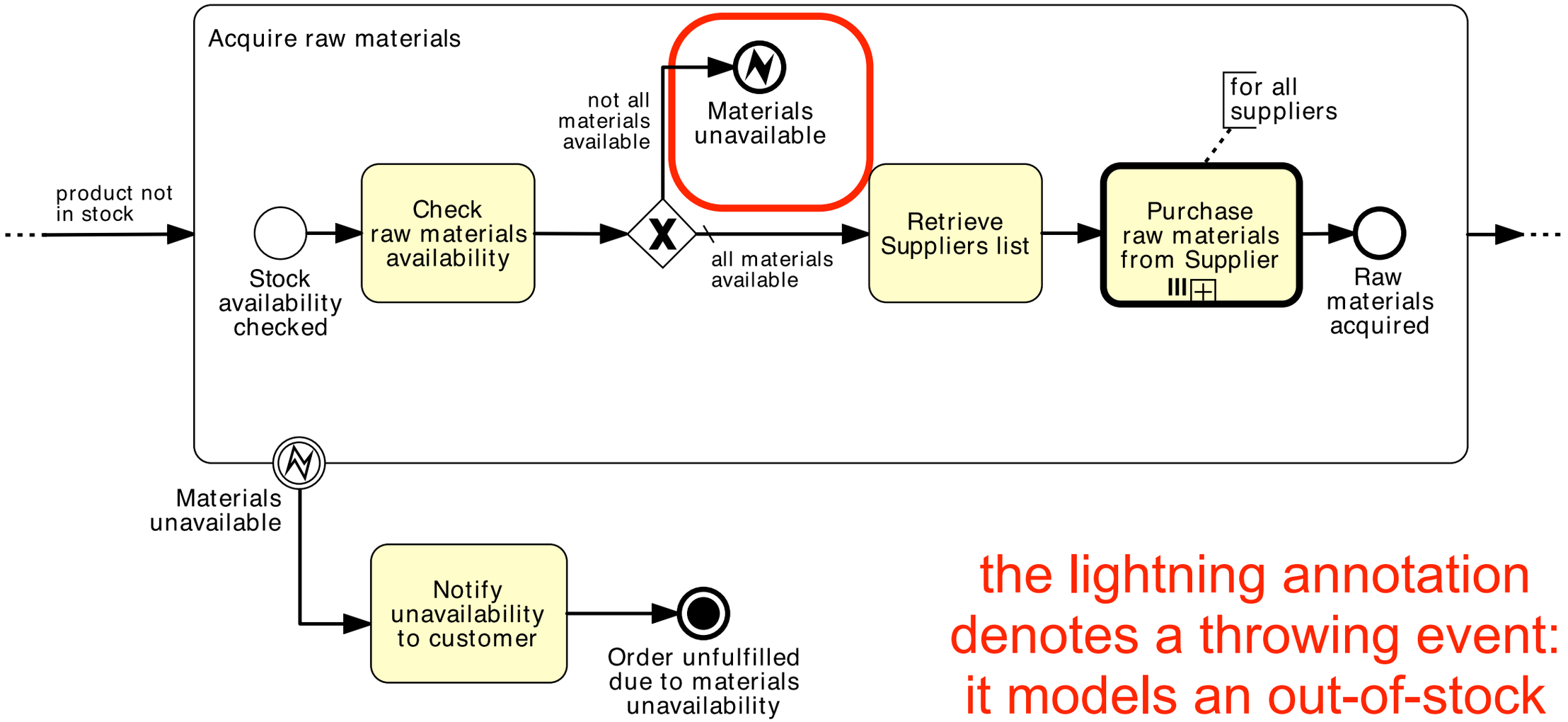
Attached Intermediate Event: The activity is aborted once an event is caught.

Recovery from faults: image manipulation



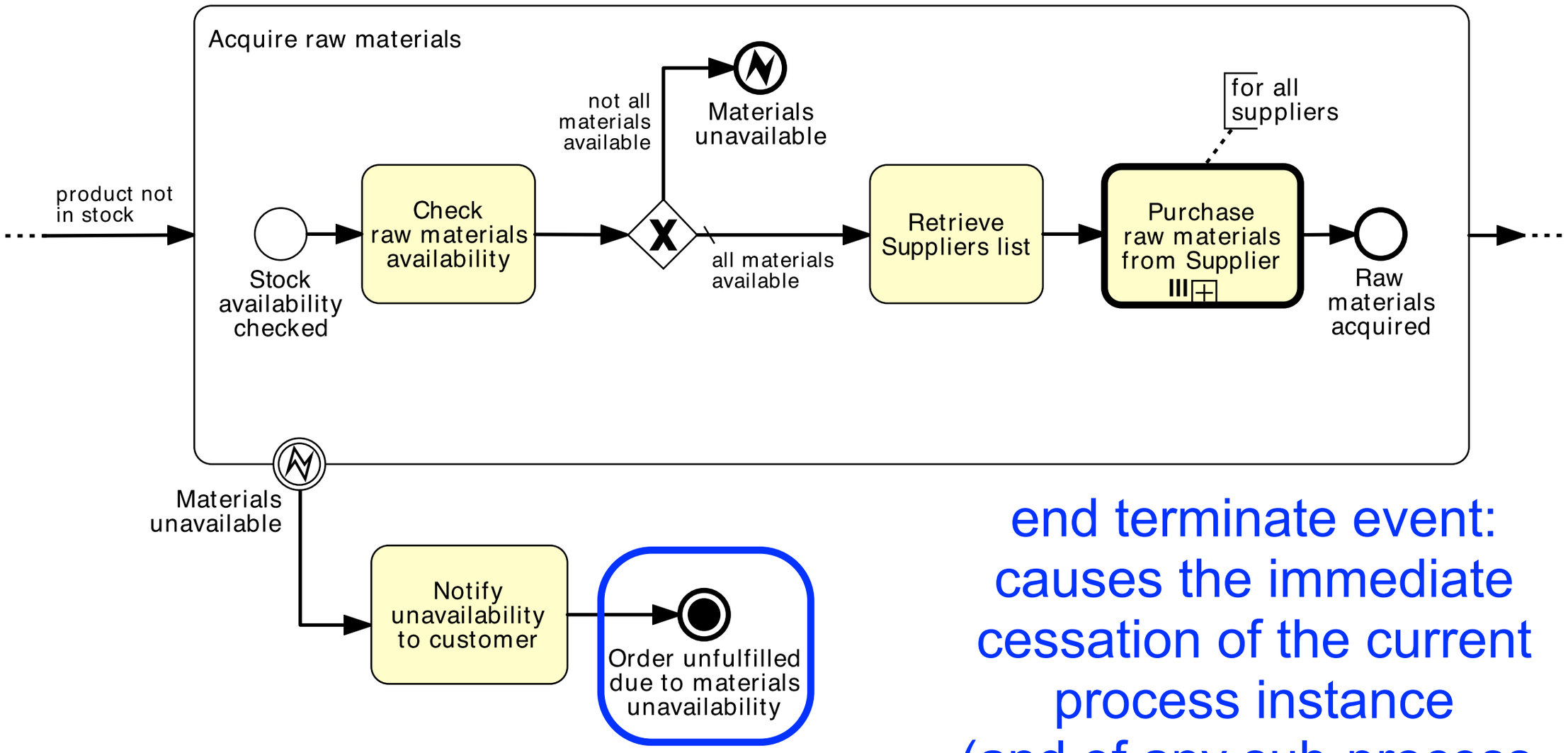
the lightning annotation
denotes an error-catching event

Throwing and catching: order fulfillment



the lightning annotation denotes a throwing event: it models an out-of-stock exception

Throwing and catching: order fulfillment



end terminate event:
causes the immediate cessation of the current process instance
(and of any sub-process, but not of the parent process if any)

Choreographies

Choreography

A **choreography** defines the sequence of interaction between participants

A choreography does not exists in a pool and it is not executable

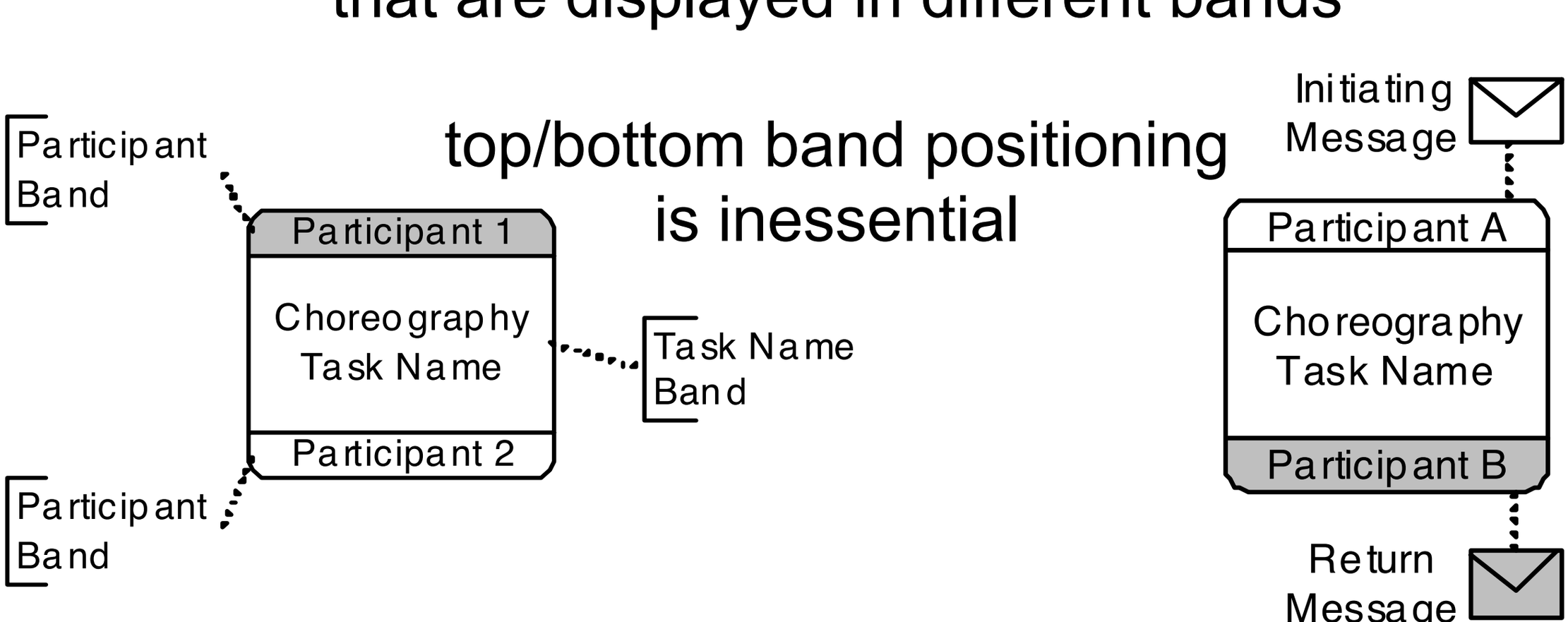
It describes how the participants are supposed to behave

a choreography can also use message data objects

Choreography task

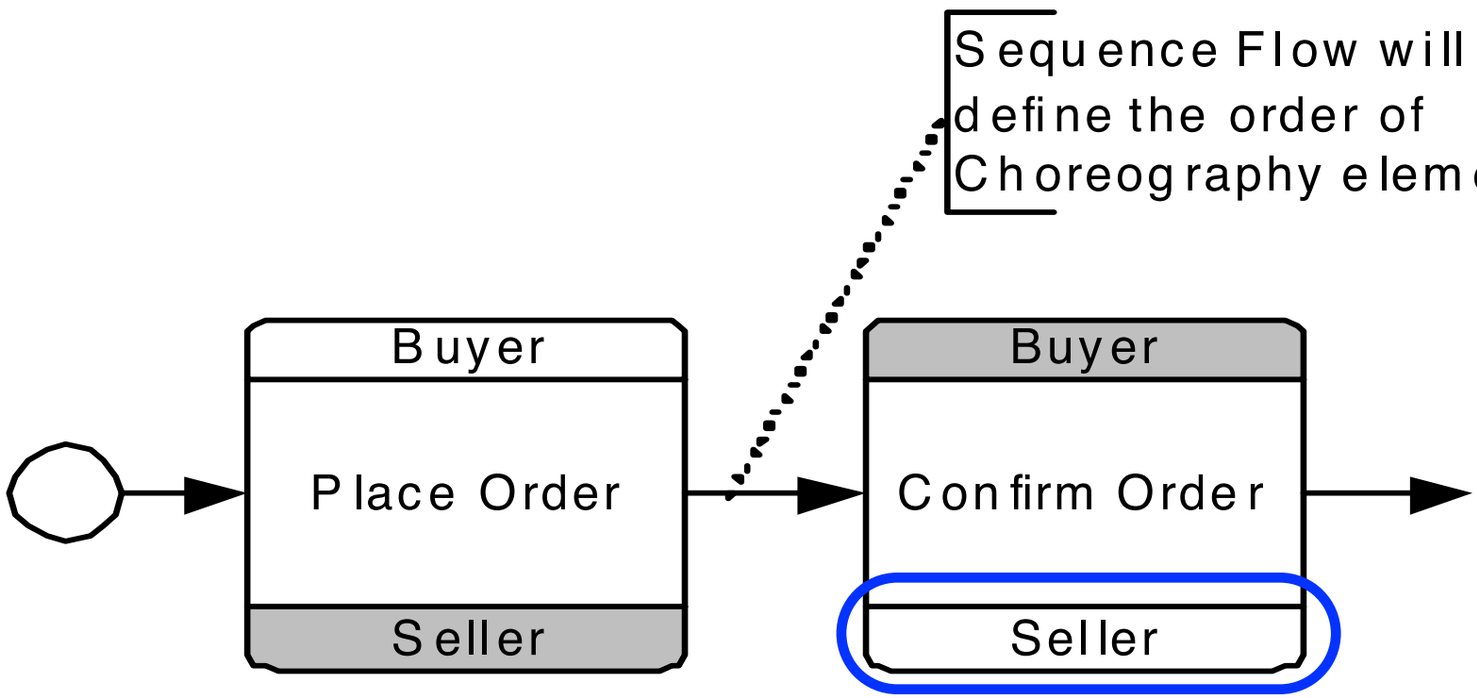
A **choreography task** is an activity in a choreography that consists of a set (one or more) communications

A choreography task involves two or more participants that are displayed in different bands



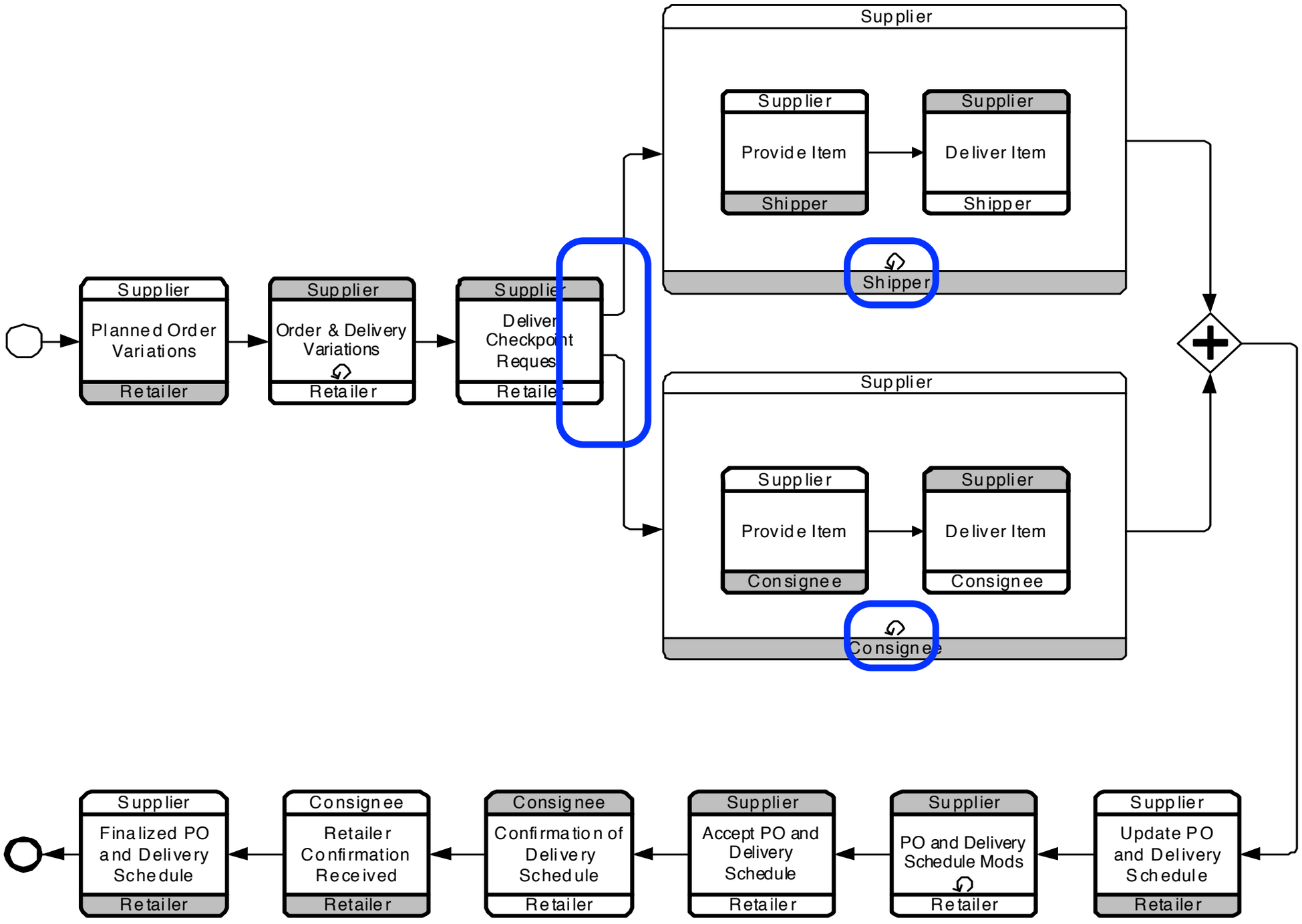
Choreography flow

Ordinary sequence flow and gateways
are used within choreographies
to show the sequence of tasks involved



the initiator of the second interaction
must be involved in the previous one

A choreography



BPMN Semantics

BPMN execution semantics

OMG
WE SET THE STANDARD™

Date: January 2011

BPMN Business Process Model and Notation (BPMN)

Version 2.0

Standard document URL: http://www.omg.org/spec/BPMN/2.0

Some sample paragraphs

The execution semantics are described informally (textually), and this is based on prior research involving the formalization of execution semantics using mathematical formalisms.

A **Process** is instantiated when one of its **Start Events** occurs.

A **Process** can also be started via an **Event-Based Gateway** or a **Receive Task** that has no incoming **Sequence Flows**

Each **Start Event** that occurs creates a *token* on its outgoing **Sequence Flows**, which is followed as described by the semantics of the other **Process** elements.

A **Process** *instance* is completed, if and only if the following three conditions hold:

- If the *instance* was created through an instantiating **Parallel Gateway**, then all subsequent **Events** (of that **Gateway**) MUST have occurred.
- There is no *token* remaining within the **Process** *instance*.
- No **Activity** of the **Process** is still active.

For a **Process** *instance* to become completed, all *tokens* in that instance MUST reach an end node.
A *token* reaching an **End Event** triggers the behavior associated with the **Event** type.
If a *token* reaches a **Terminate End Event**, the entire **Process** is abnormally terminated.

BPMN formal semantics?

Many attempts:
Abstract State Machines (ASM)
Term Rewriting Systems
Graph Rewrite Systems
Process Algebras
Temporal Logic
…
Petri nets

(Usual difficulties with OR-join semantics)

Sound BPMN diagrams

We can exploit the formal semantics of nets
to give unambiguous semantics to
BPMN process diagrams
BPMN collaboration diagrams

We transform
BPMN process diagrams to wf nets
BPMN collaboration diagrams to wf systems

A BPMN diagram is sound if its net is so

We can reuse the verification tools
to check if the net is sound

Translation of BPMN to Petri nets

From BPMN to Petri nets

Available online at www.sciencedirect.com
ScienceDirect
Information and Software Technology 50 (2008) 1281–1294
INFORMATION AND SOFTWARE TECHNOLOGY
www.elsevier.com/locate/infsof

Semantics and analysis of business process models in BPMN

Remco M. Dijkman [a], Marlon Dumas [b,c], Chun Ouyang [c,*]

[a] *Department of Technology Management, Eindhoven University of Technology, P.O. Box 513, 5600 MB, The Netherlands*
[b] *Institute of Computer Science, University of Tartu, J Liivi 2, Tartu 50409, Estonia*
[c] *Faculty of Information Technology, Queensland University of Technology, G.P.O. Box 2434, Brisbane, Qld 4001, Australia*

Simplified BPMN

a start / exception event has just one outgoing flow
and no incoming flow

an end event has just one incoming flow
and no outgoing flow

all activities and intermediate events have exactly
one incoming flow and one outgoing flow

all gateways have either
one incoming flow (and multiple outgoing)
or one outgoing flow (and multiple incoming)

Simplified BPMN

The previous constraints are no real limitation:

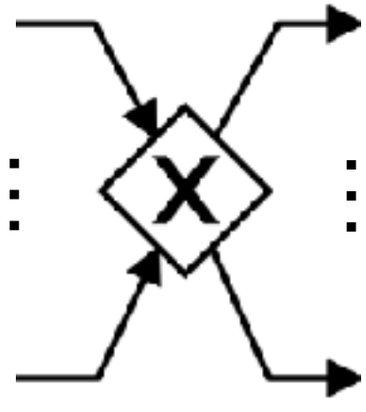
events or activities with multiple incoming flows:
insert a preceding XOR-join gateway

events or activities with multiple outgoing flows:
insert a following AND-split gateway

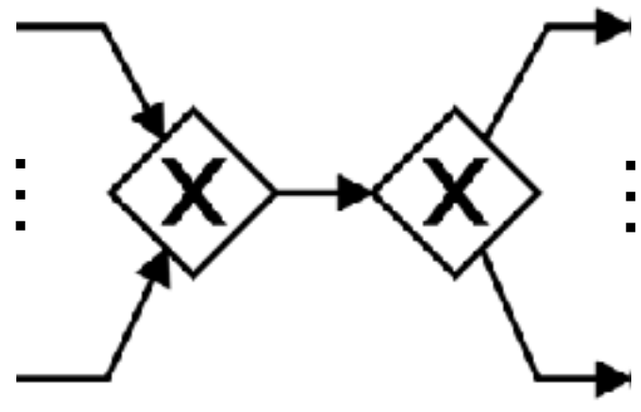
gateways with multiple incoming and outgoing flows:
decompose in two gateways

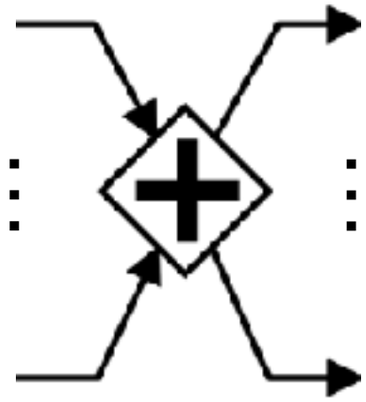
insert start / end events if needed

Pay attention to gateways

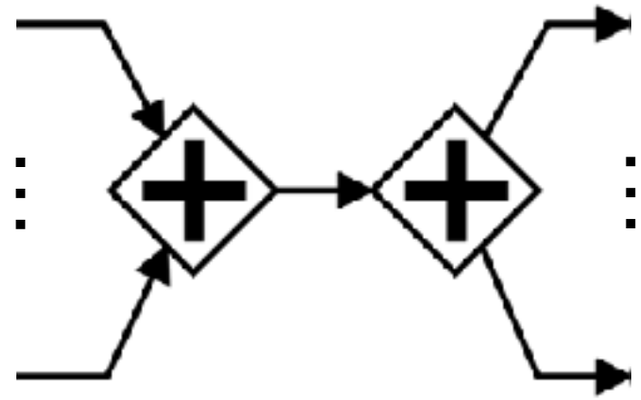


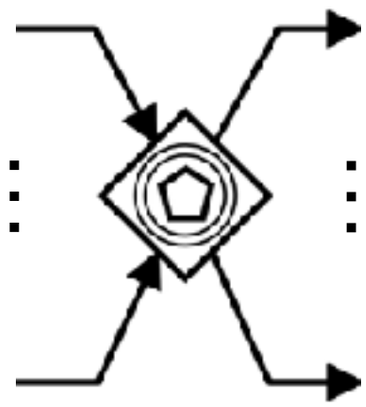
stands for



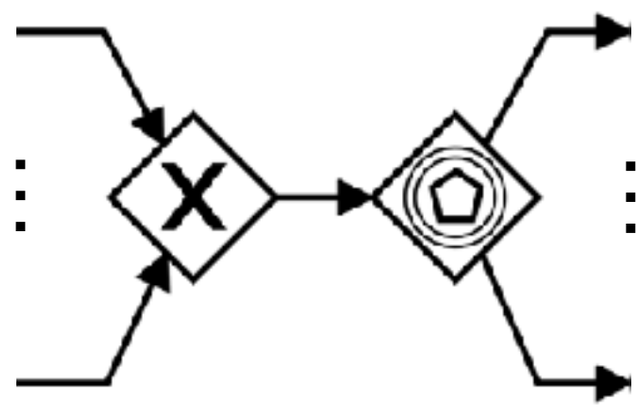


stands for

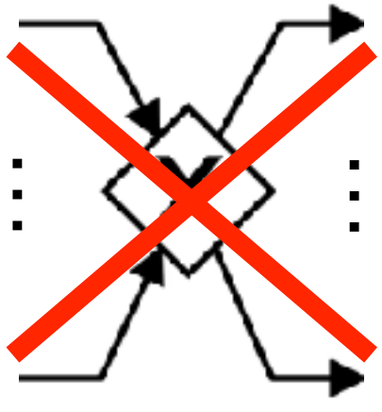




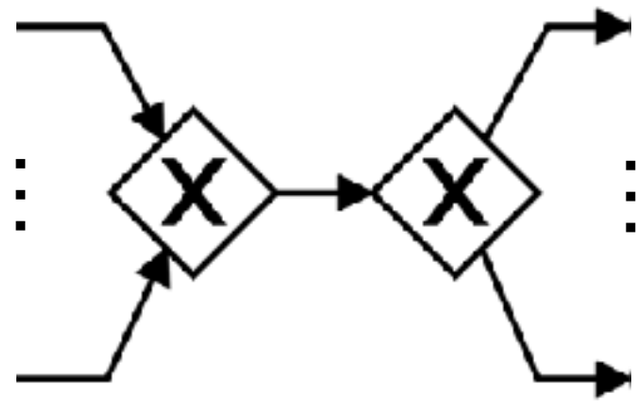
stands for

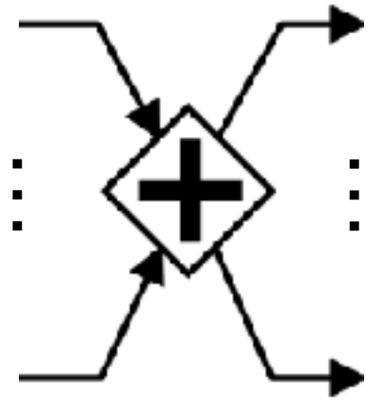


My suggestions

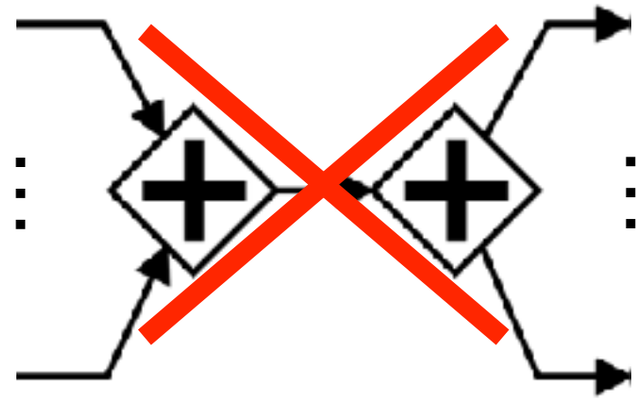


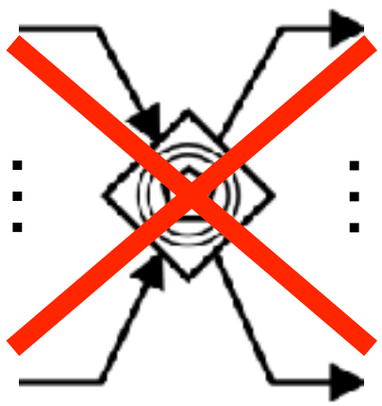
stands for



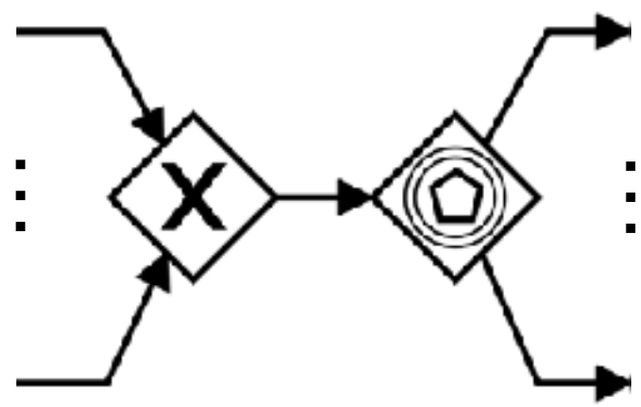


stands for



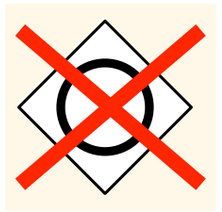


stands for



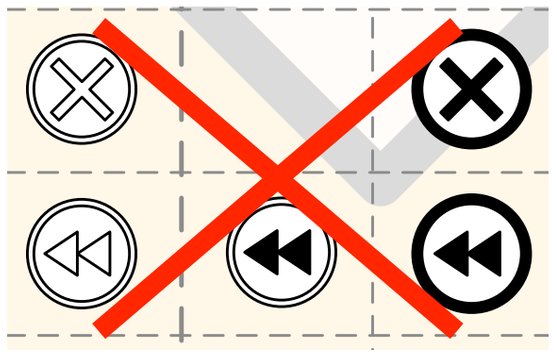
Simplified BPMN

Avoid OR-gateways
(all problems seen with EPC apply to BPMN as well)



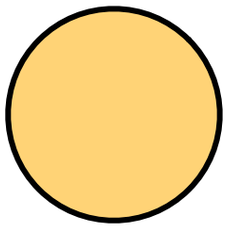
Limited form of sub-processing

No transactions and compensations



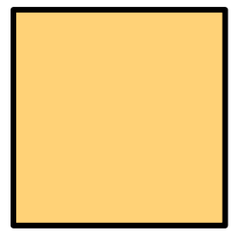
The twist!

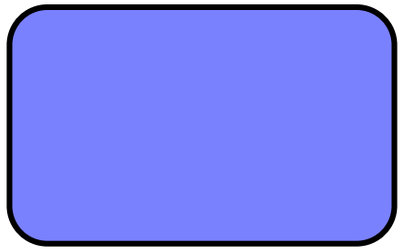
BPMN object | **net fragment**



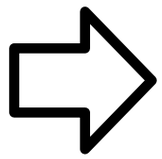
event

transition

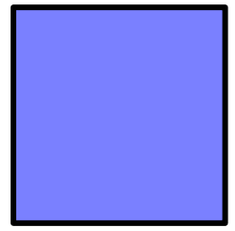


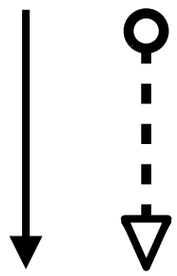


activity



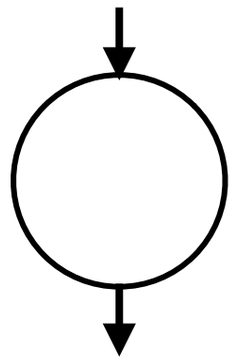
transition





sequence flow
message flow

place



Roughly

A place for each arc

one transitions for each event

one transition for each activity

one or two transitions for each gateway

…

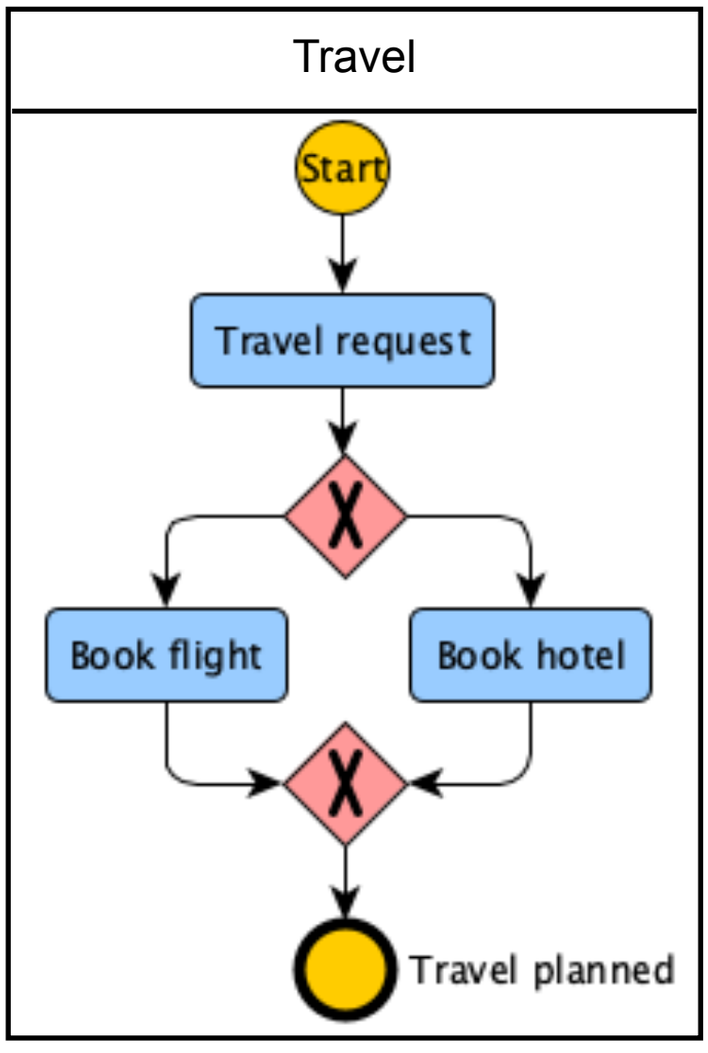
with some exceptions!

(start event, end event, event-based gateways, loops, …)

no dummy objects!

The strategy

From BPMN process diagrams to wf nets in three steps



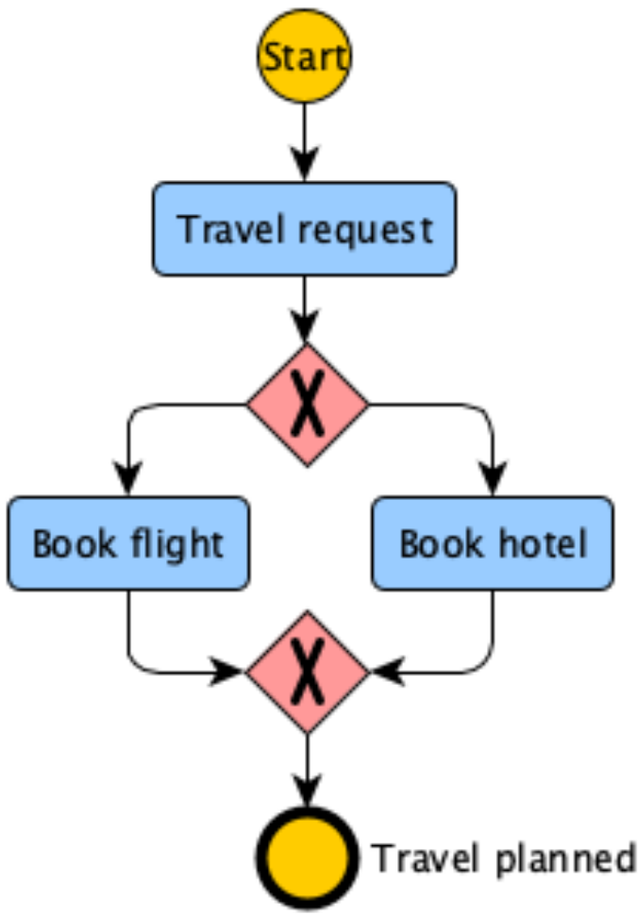
Step 1
convert
sequence flow
message flow

Step 2
convert
flow objects

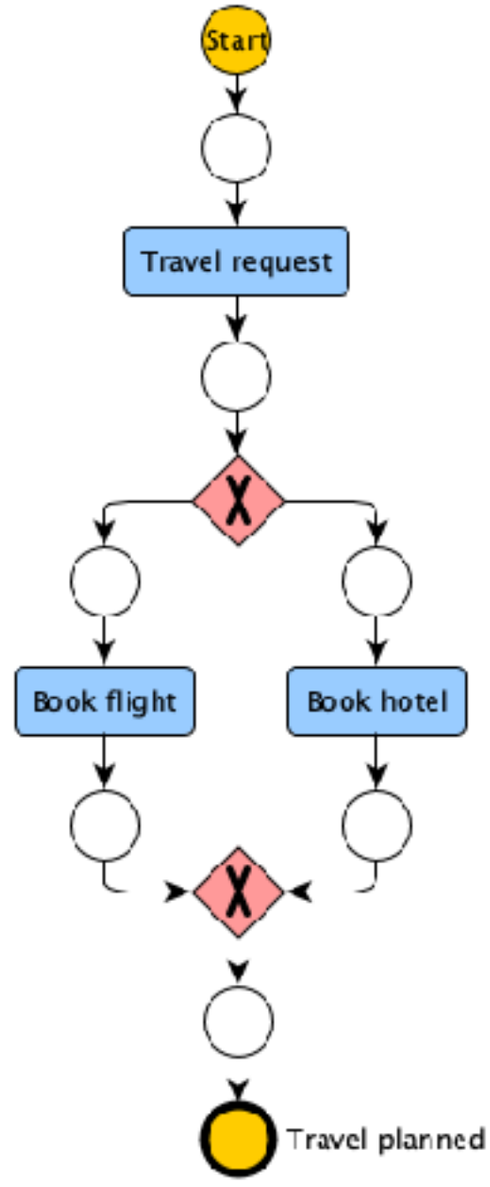
Step 3
enforce
initial place
final place

Step 1: convert flows

We insert a place for each sequence flow and message flow

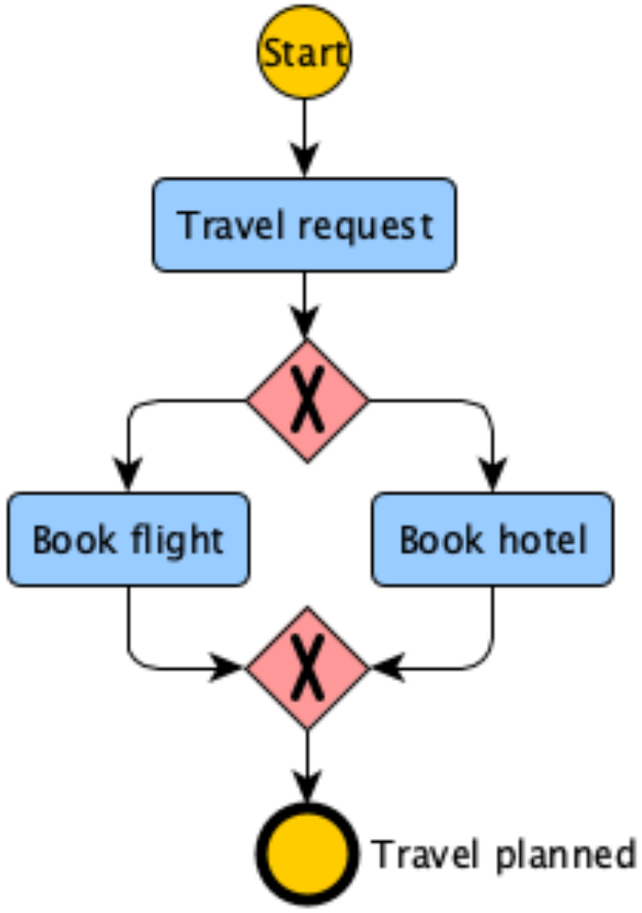


Step 1
sequence flow
message flow

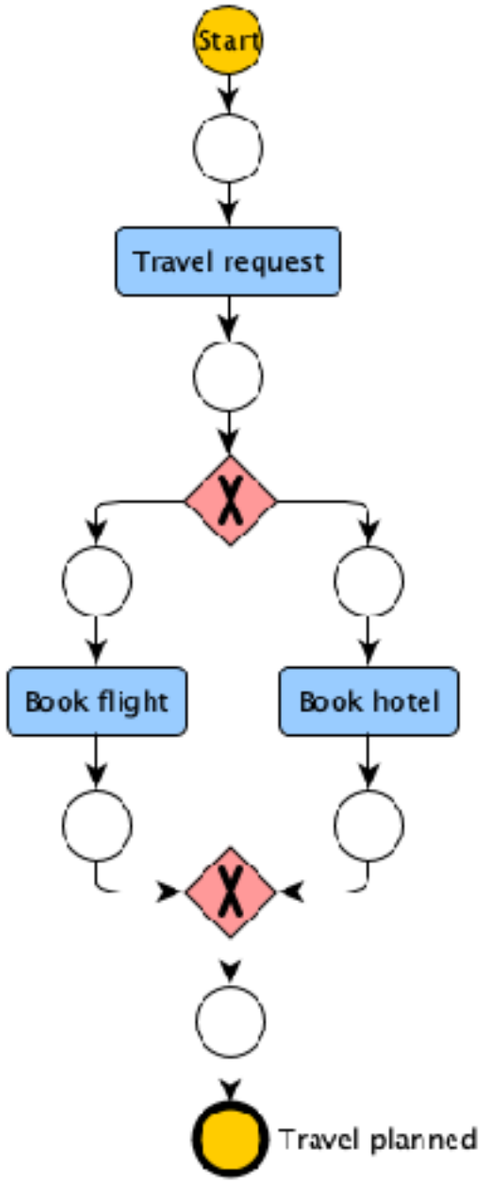


Step 2: convert flow objects

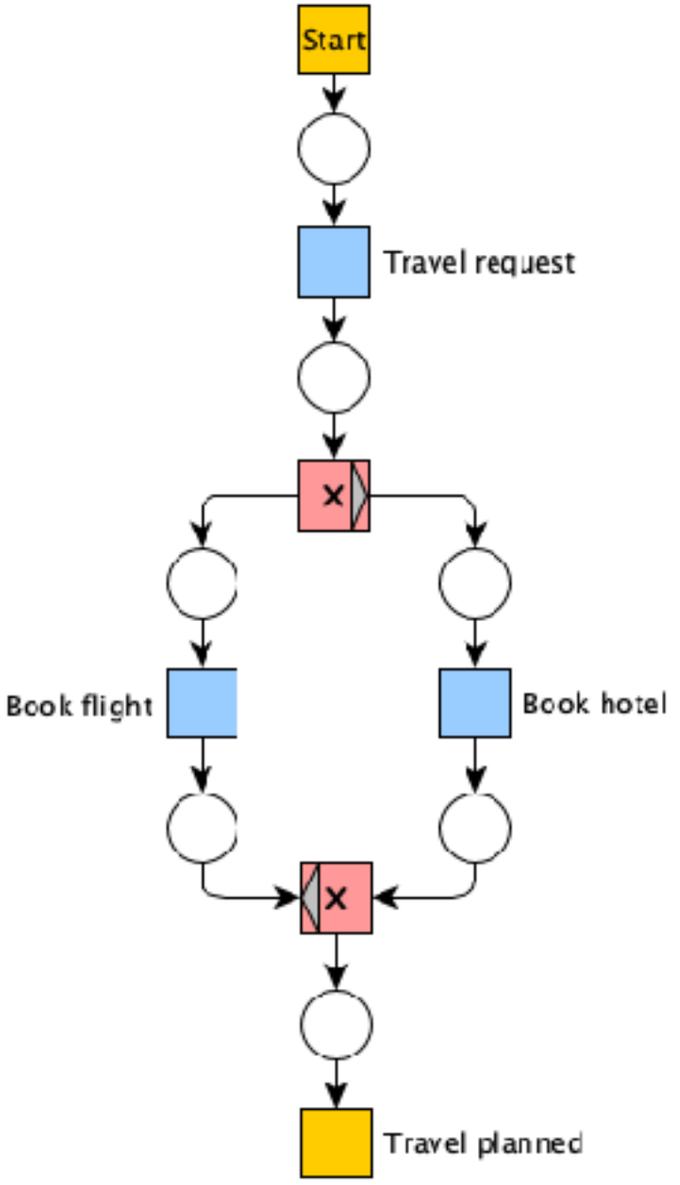
Then insert transitions



Step 1
sequence flow
message flow

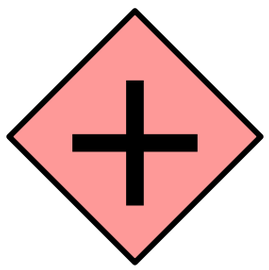


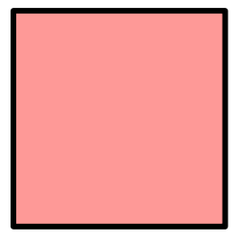
Step 2
flow objects

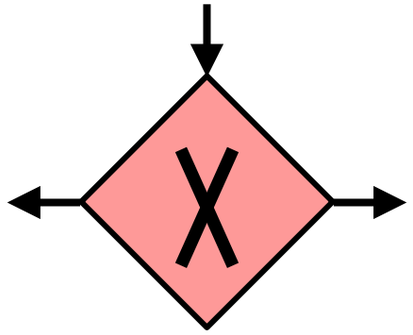


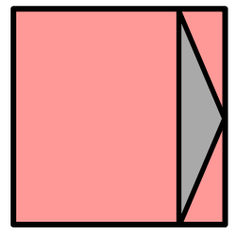
Step 2: gateways

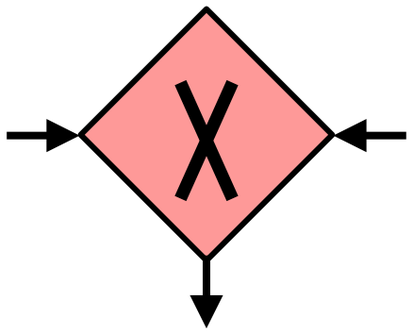
BPMN object **net fragment**

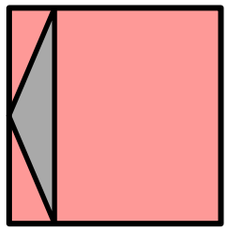


AND split / join transition 



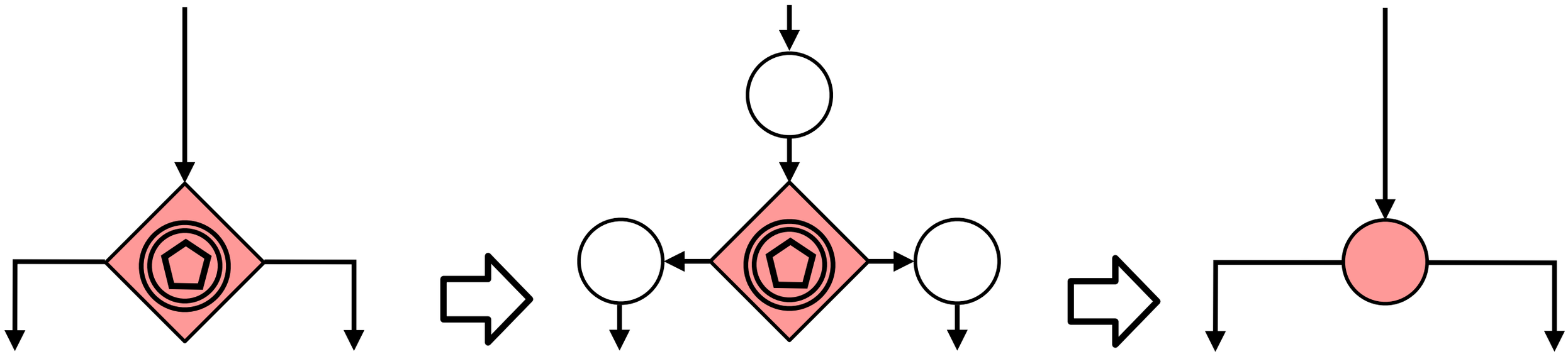
XOR split ⇨ transition 



XOR join transition 

Step 2: event-based

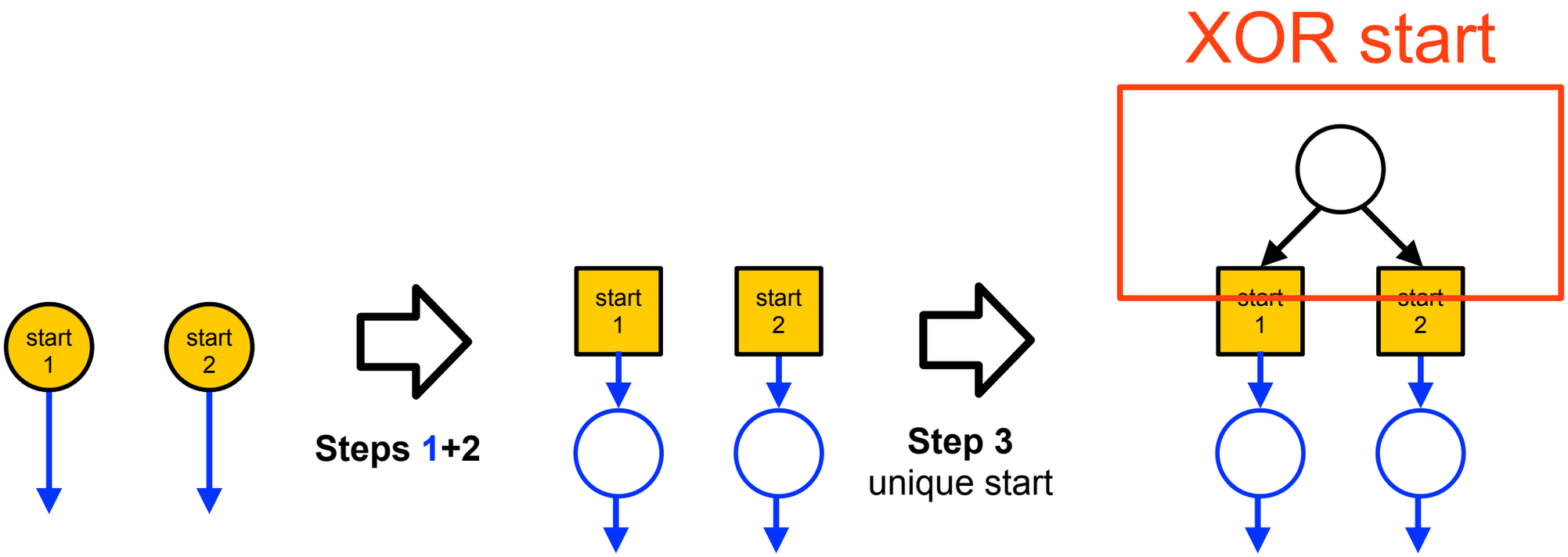
BPMN object **net fragment**



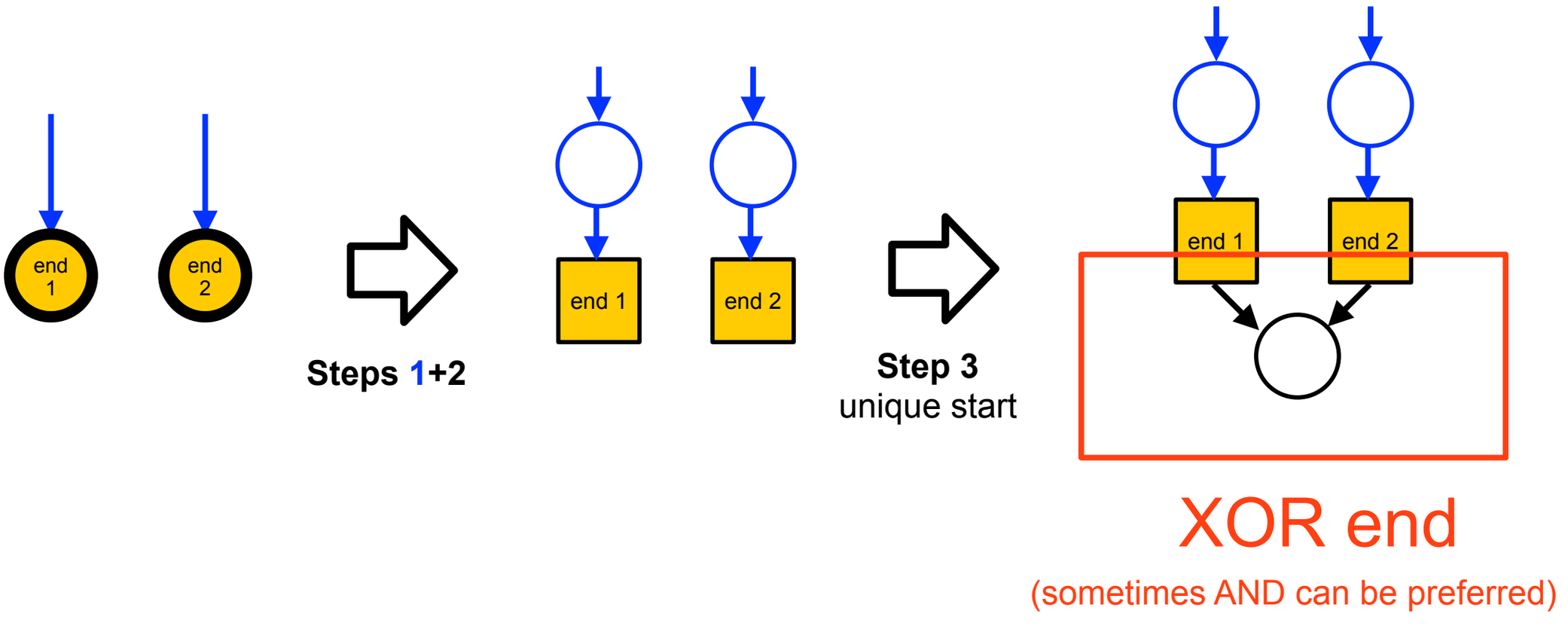
Step 1
sequence flow
message flow

Step 2
place fusion

Step 3: add unique start



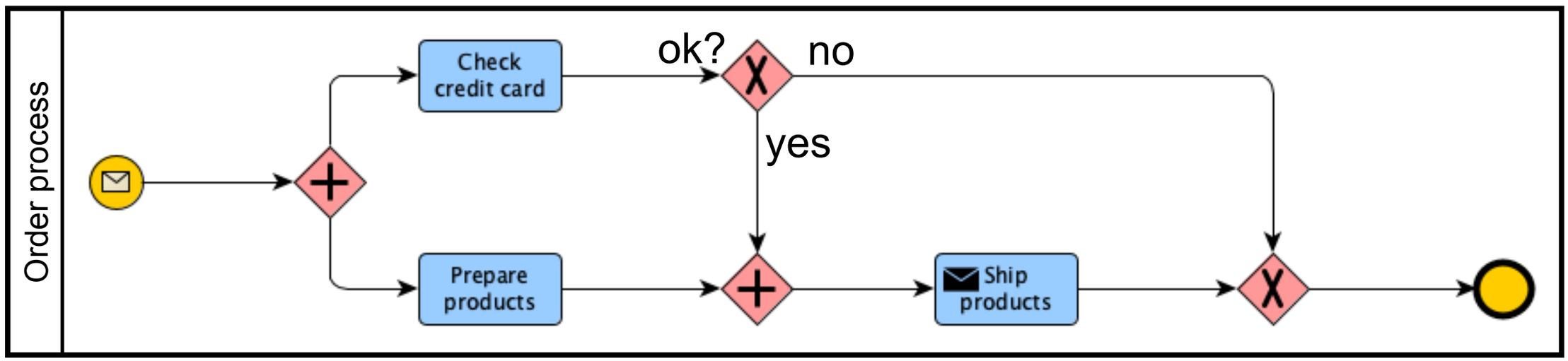
Step 3: add unique end



Example:

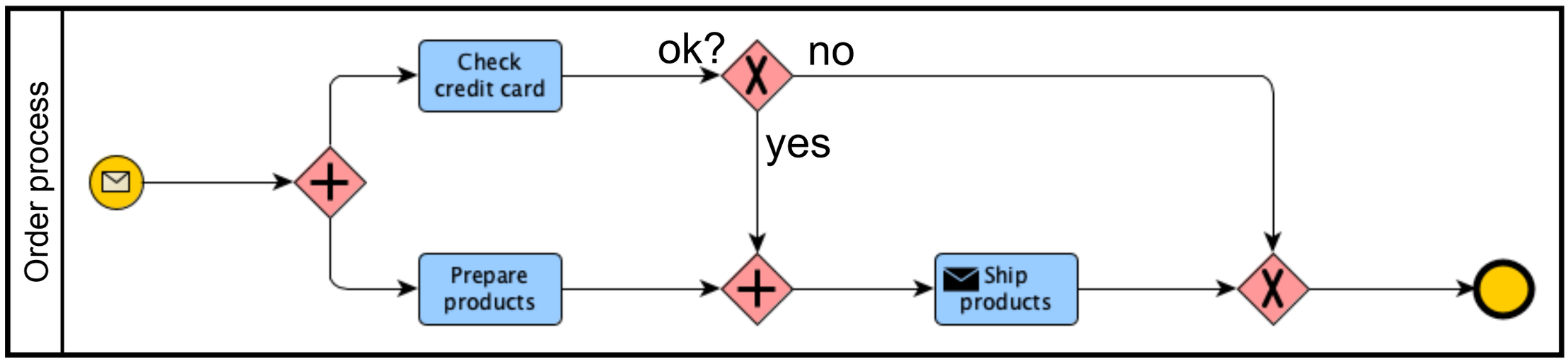
Order process

Order process

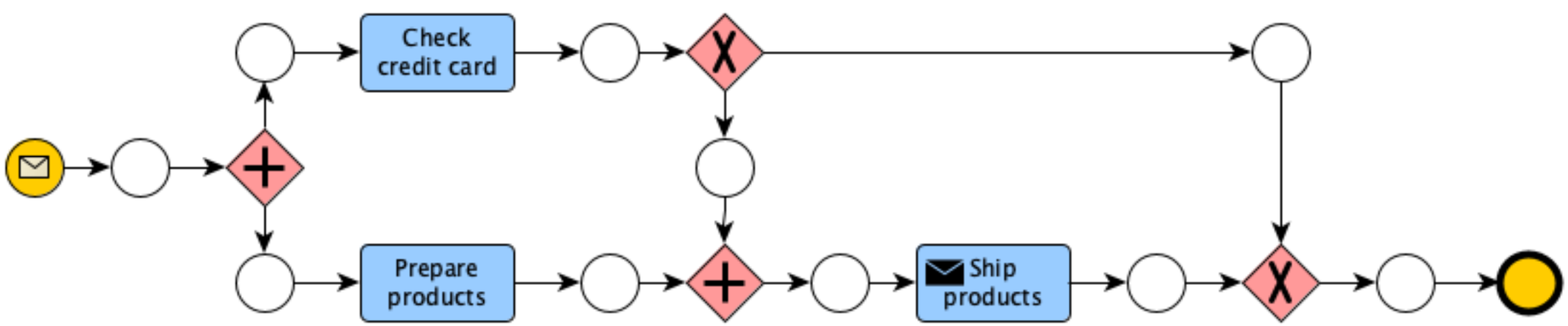


Sound?

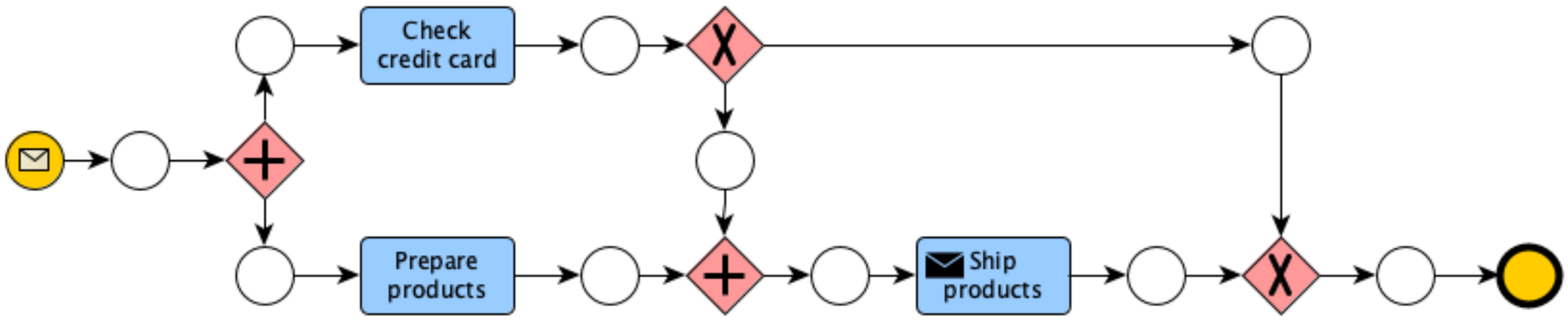
Order process: step 1

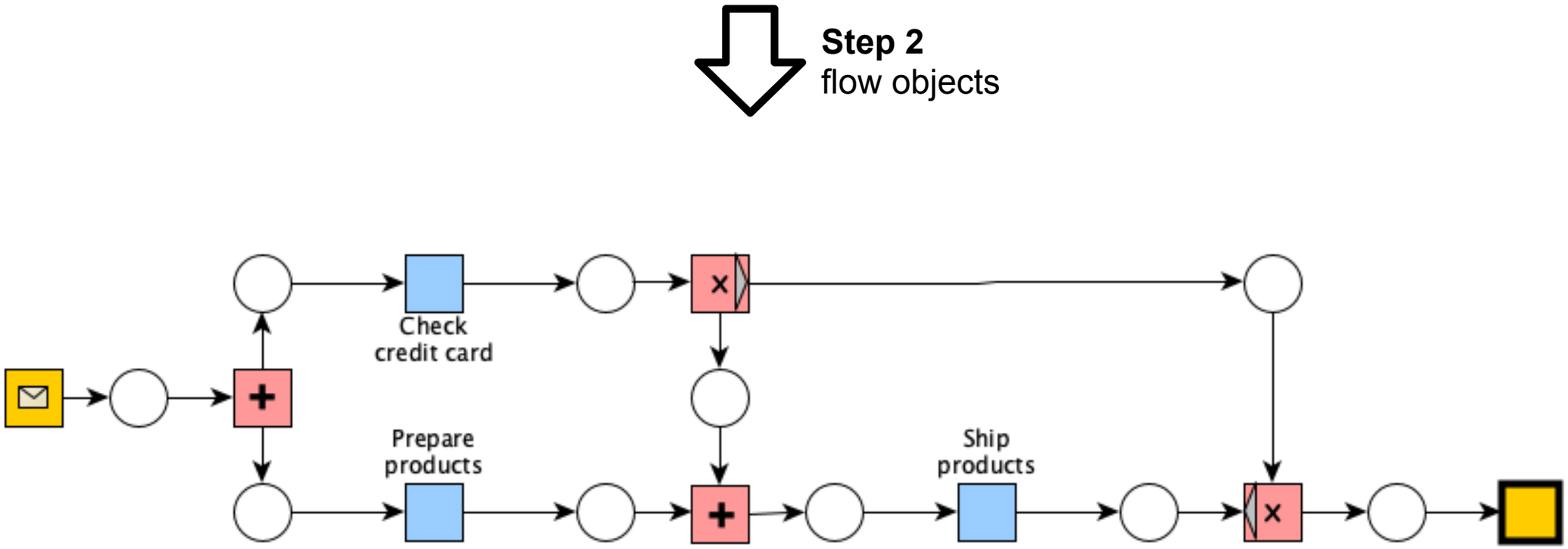


Step 1
sequence flow
message flow

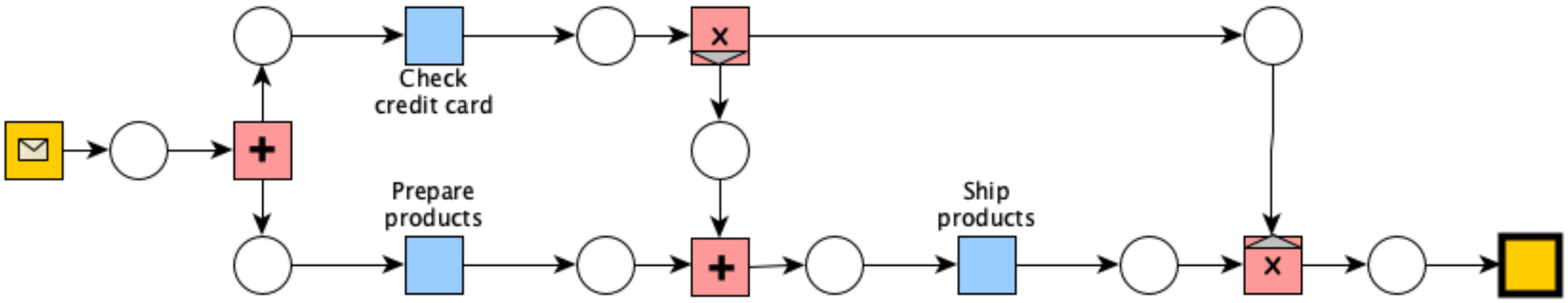


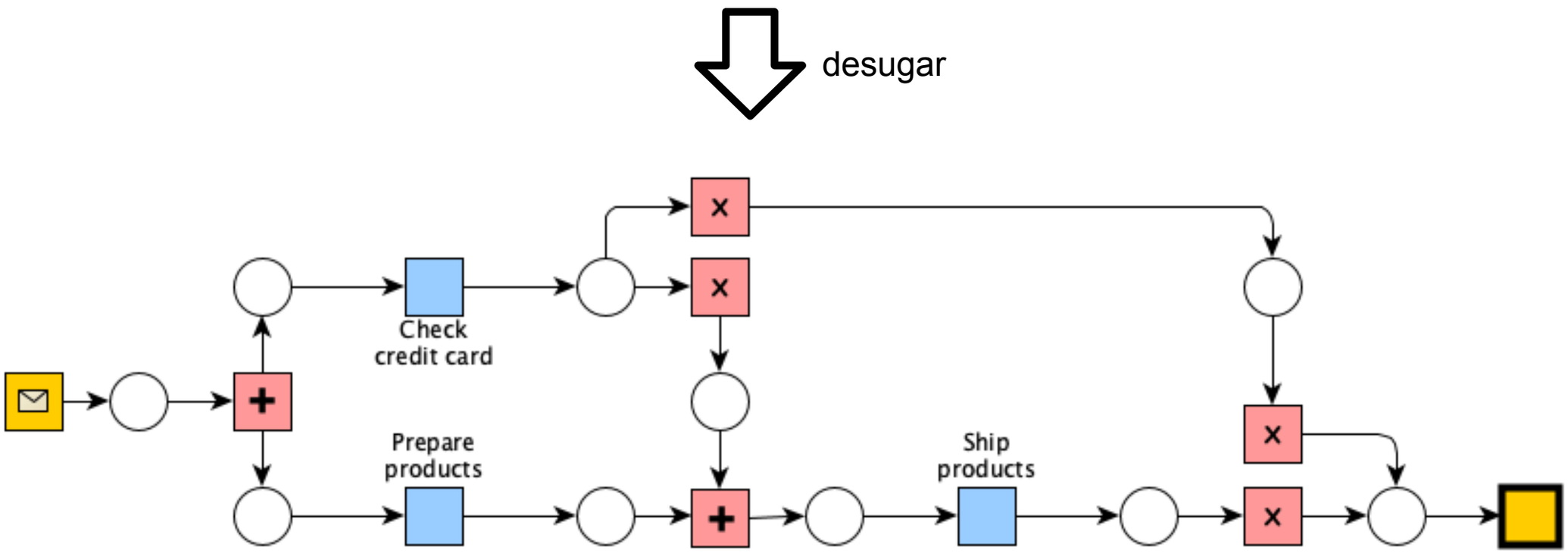
Order process: step 2



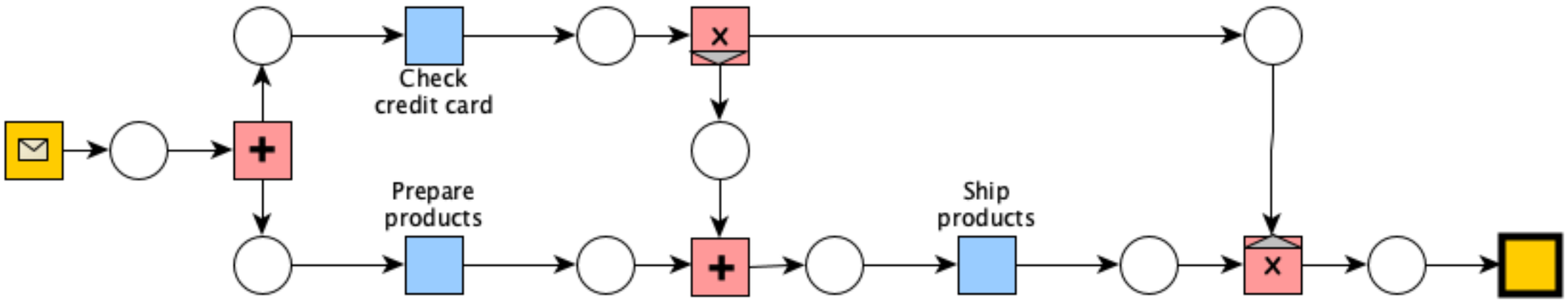


Order process: (desugar)

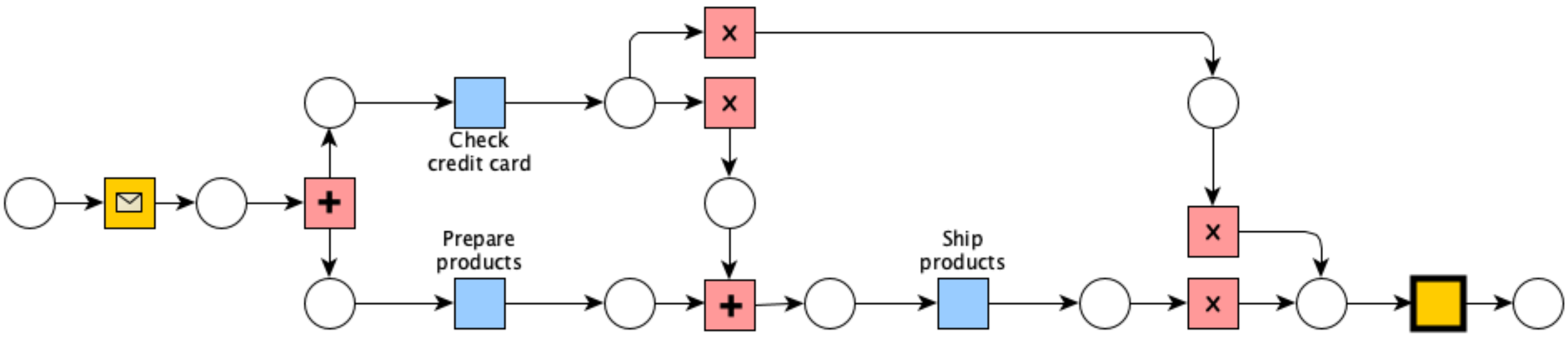




Order process: step 3

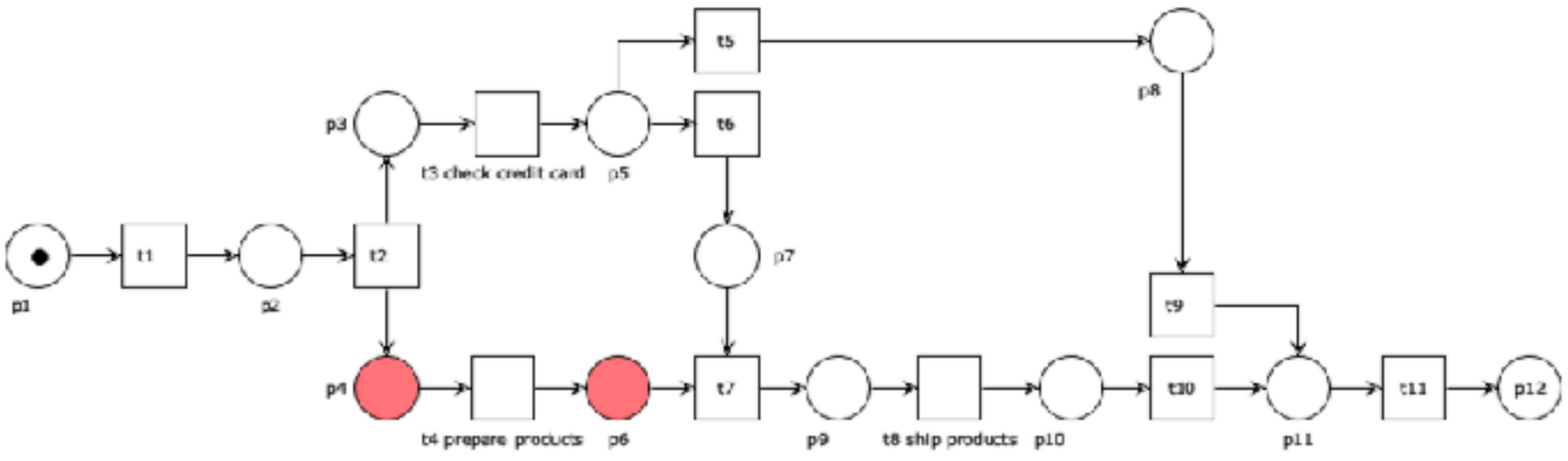


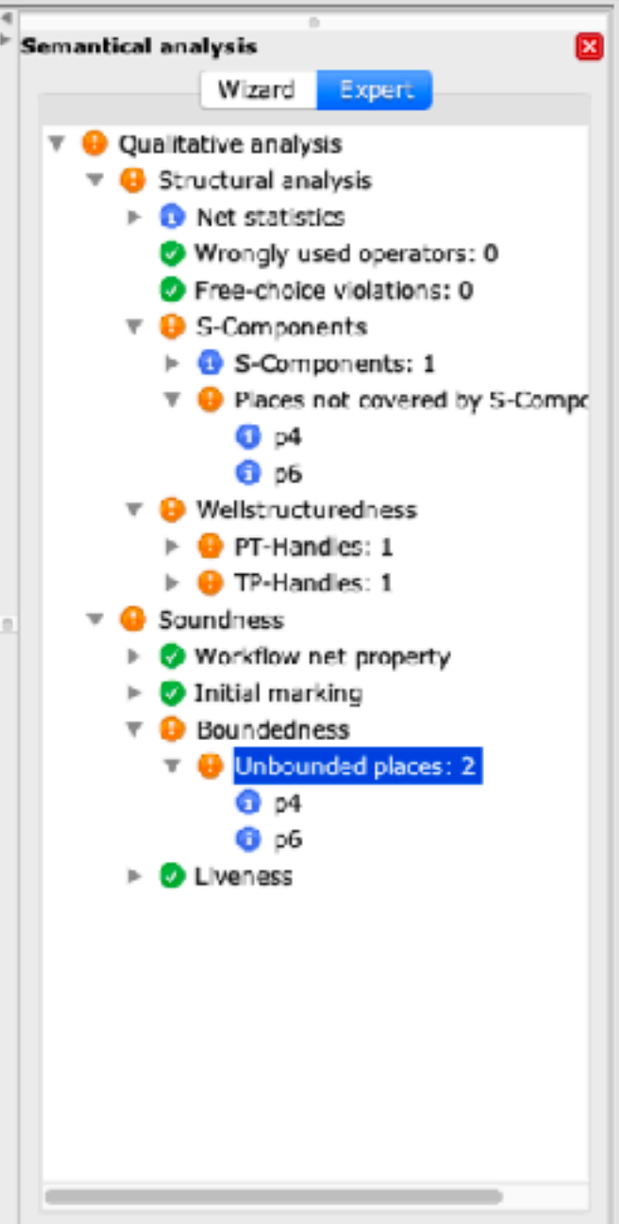
Step 3
enforce
initial place
final place



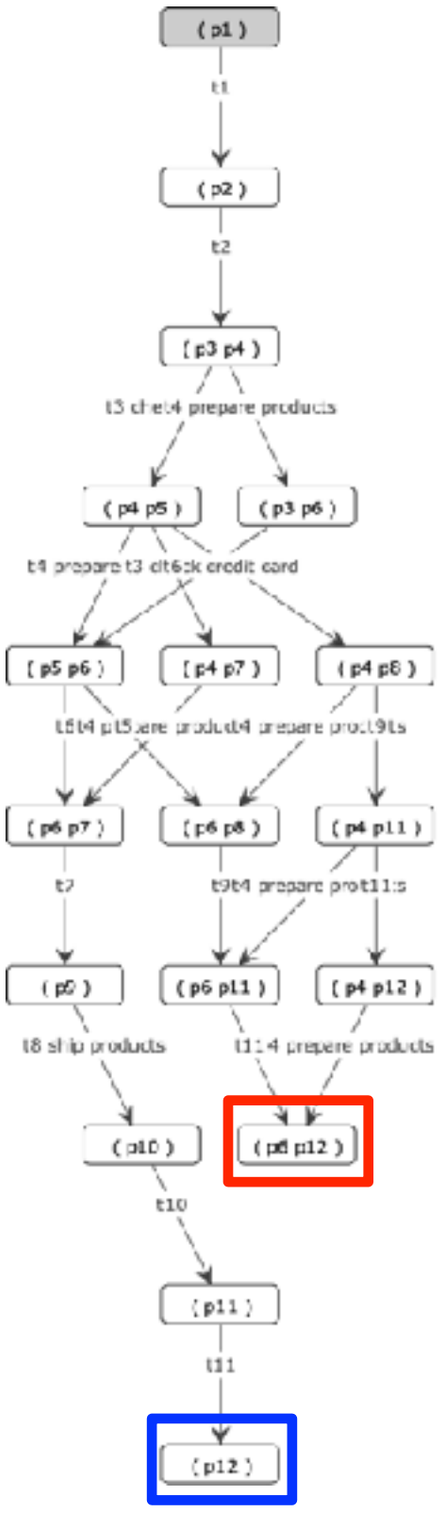
Soundness analysis

Not sound!

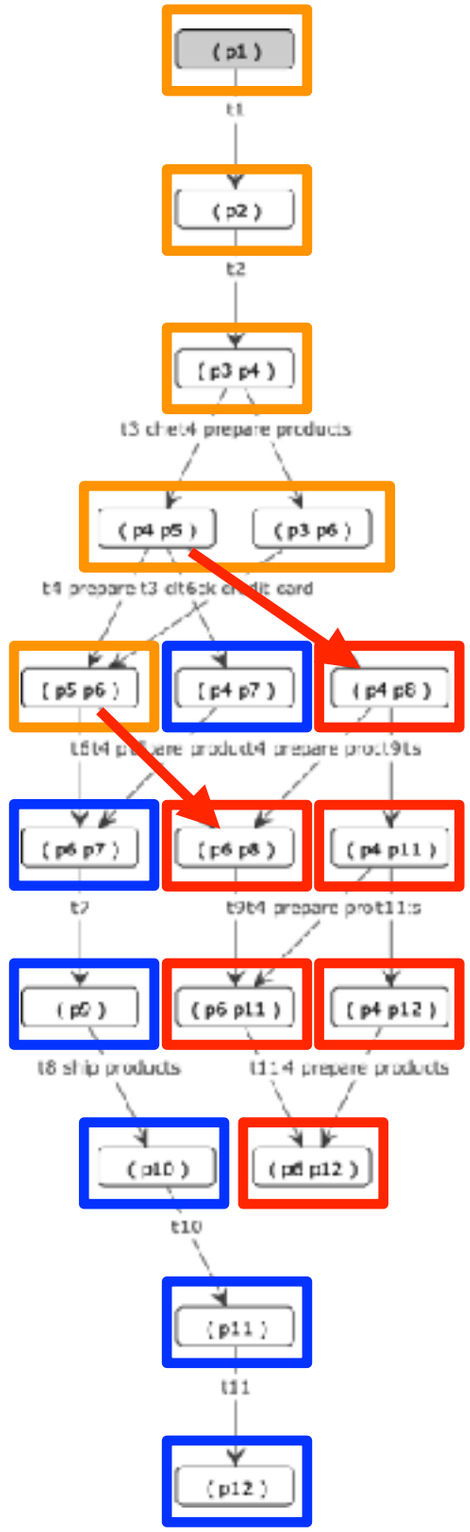




Soundness analysis



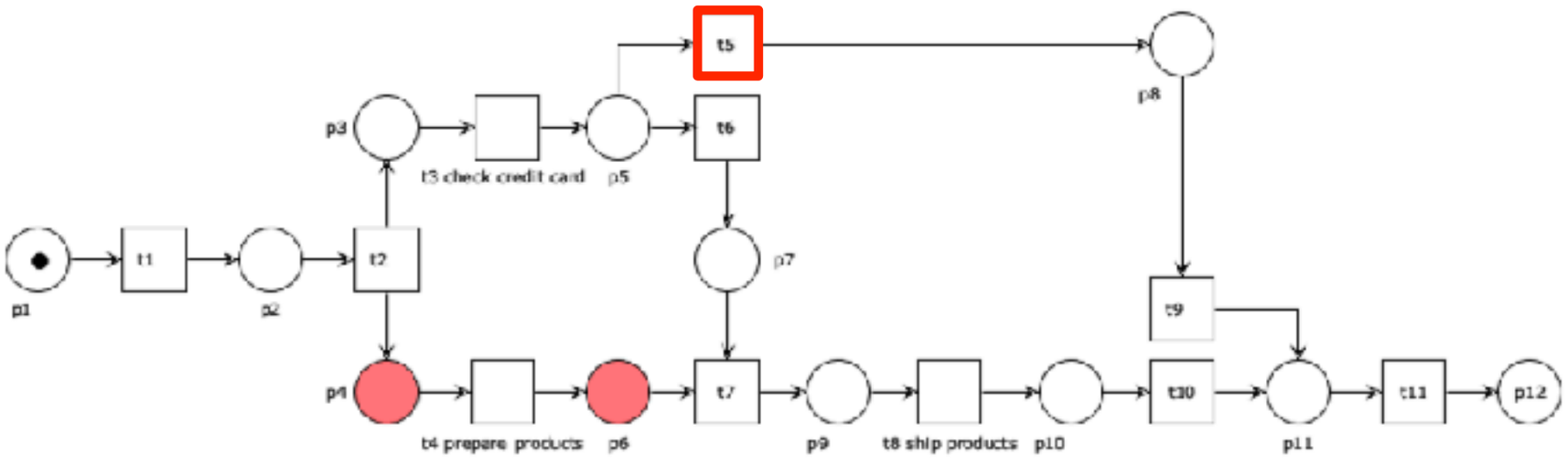
Soundness analysis

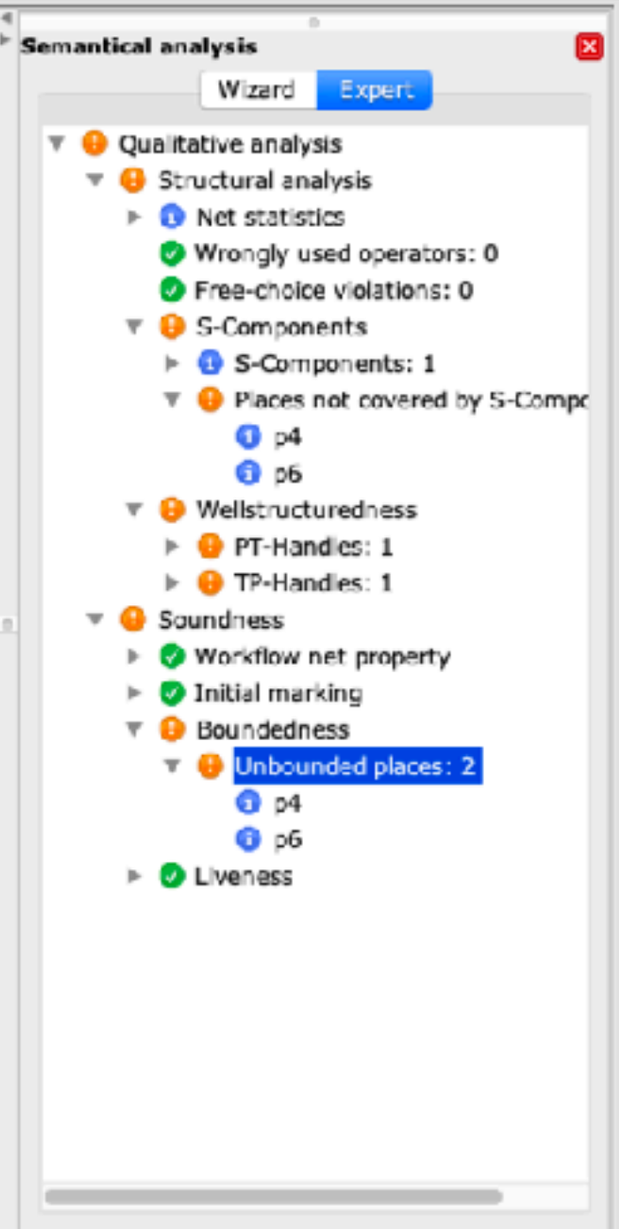


t5

Soundness analysis

Not sound!

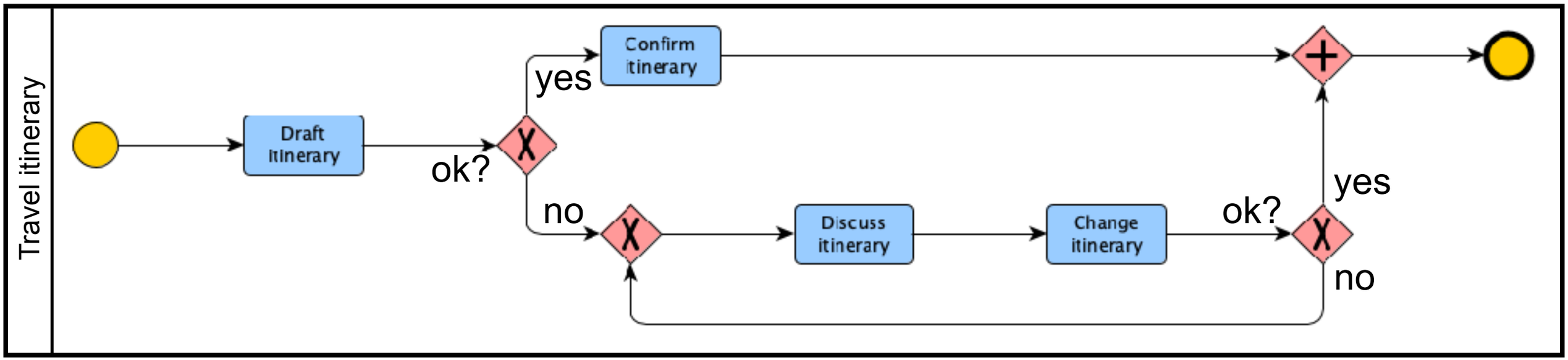




Example:

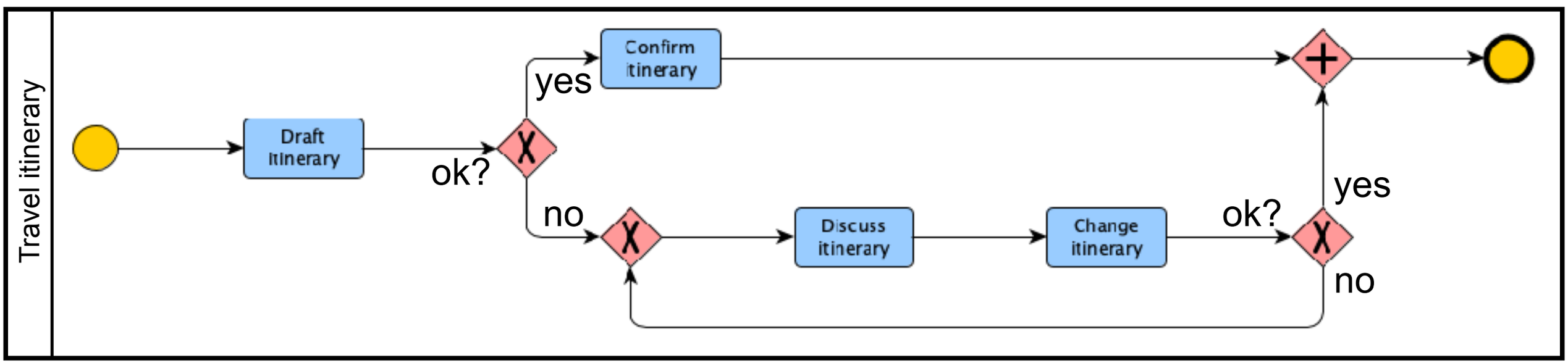
Travel itinerary

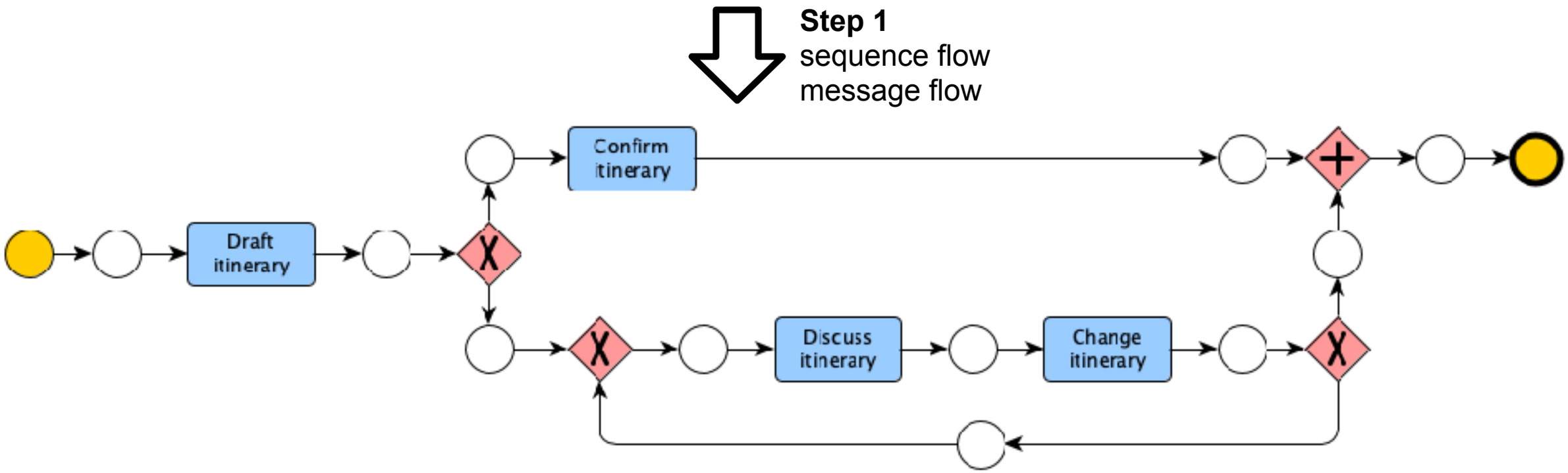
Travel itinerary



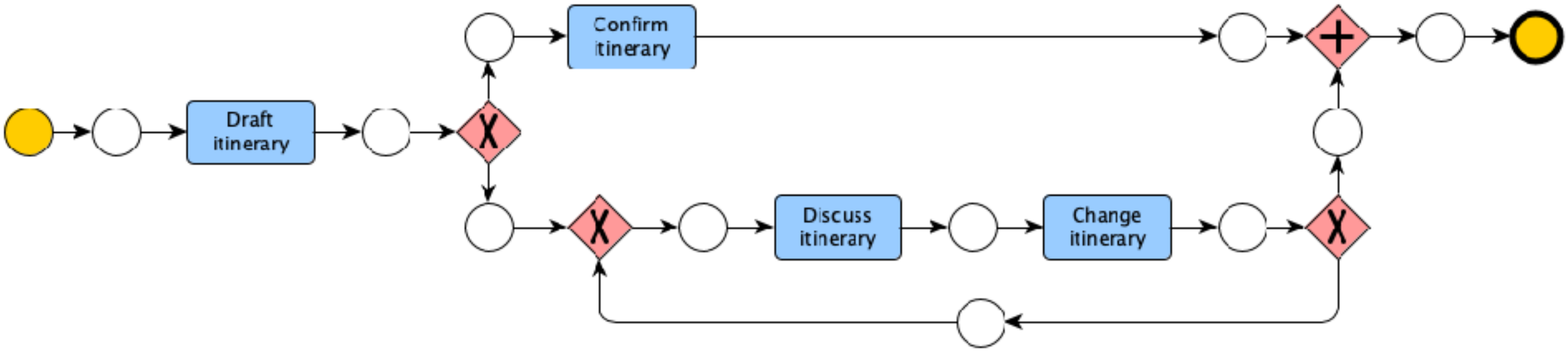
Sound?

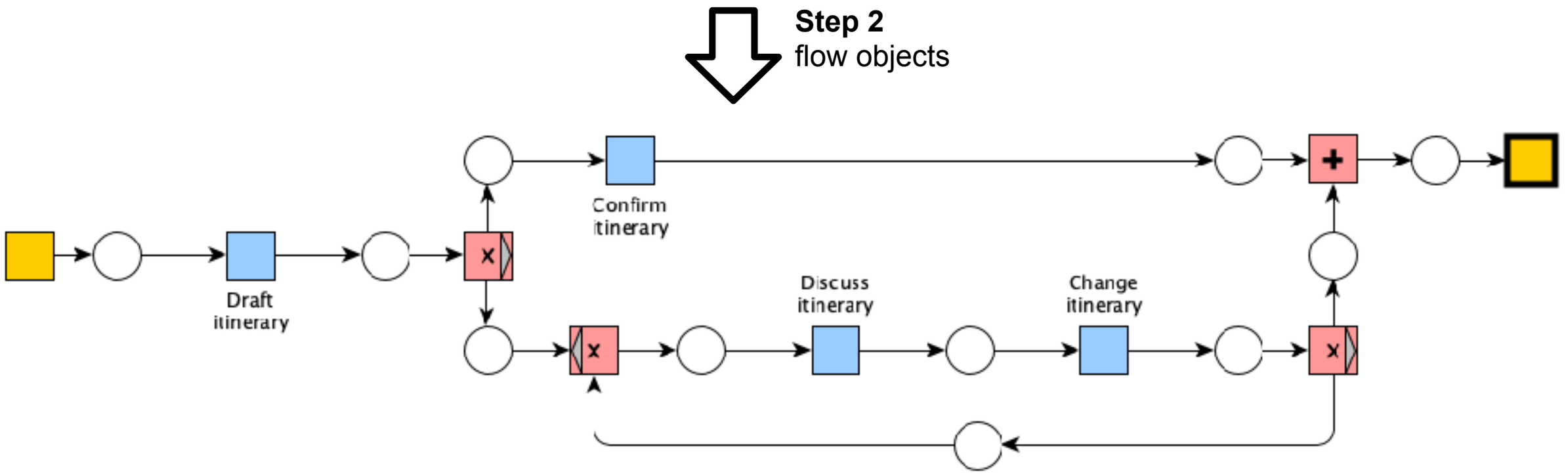
Travel itinerary: step 1



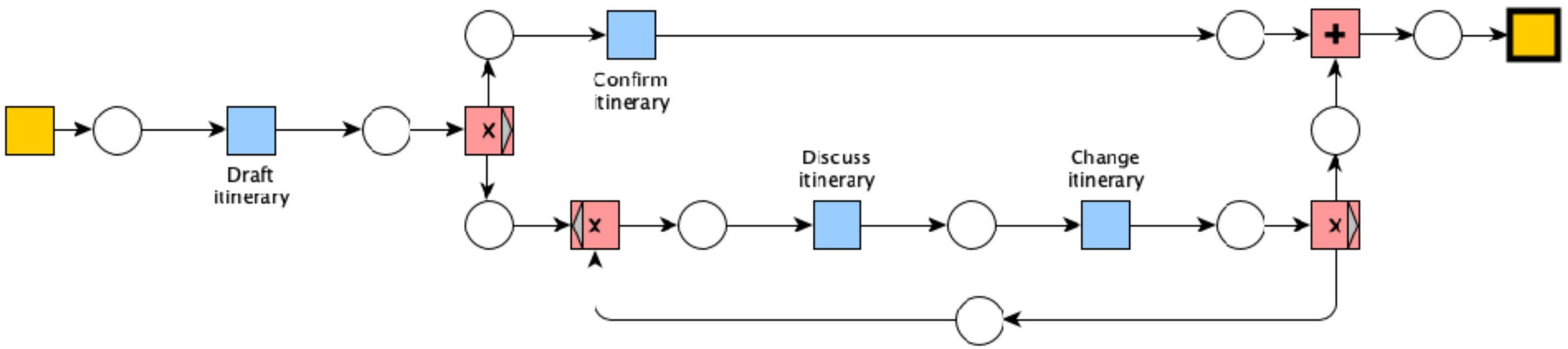


Travel itinerary: step 2

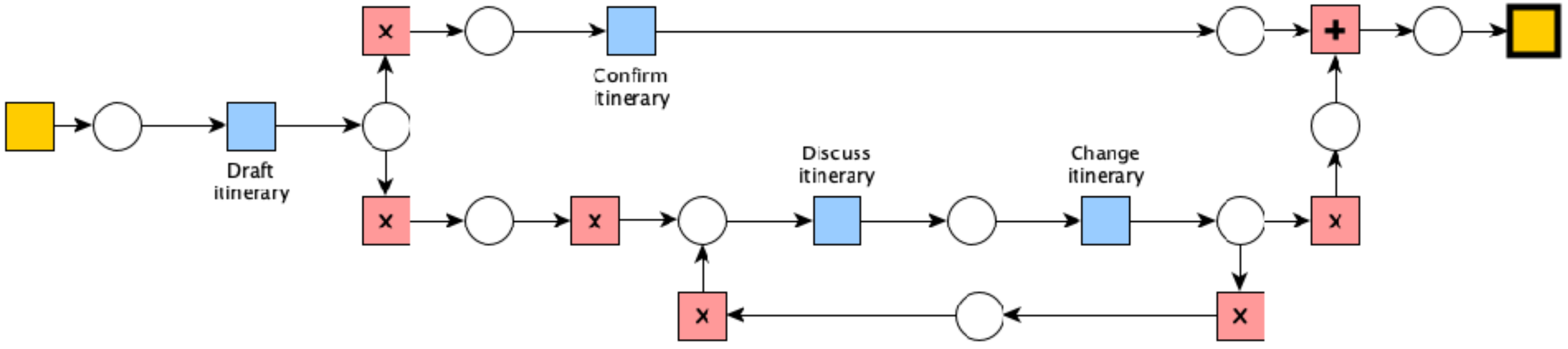




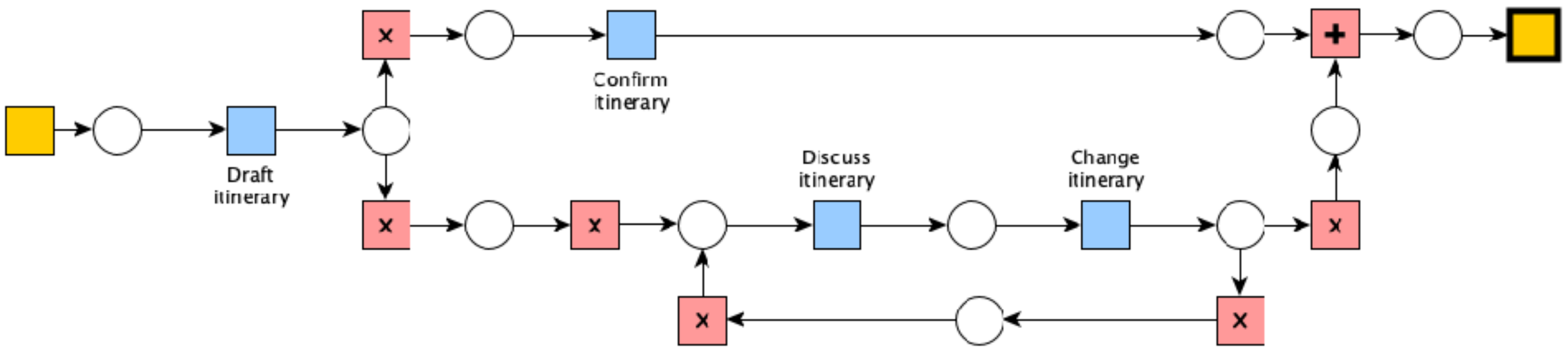
Travel itinerary: (desugar)



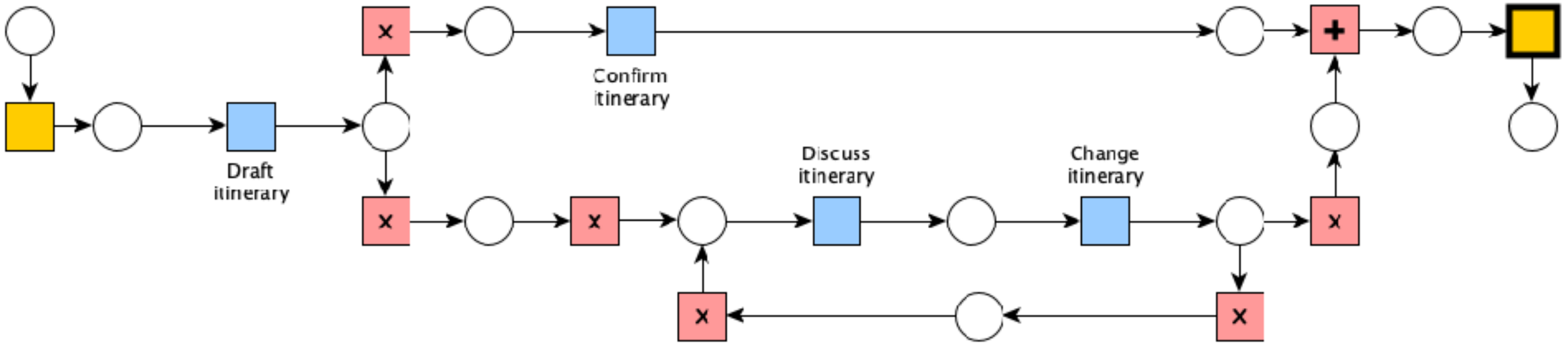
desugar



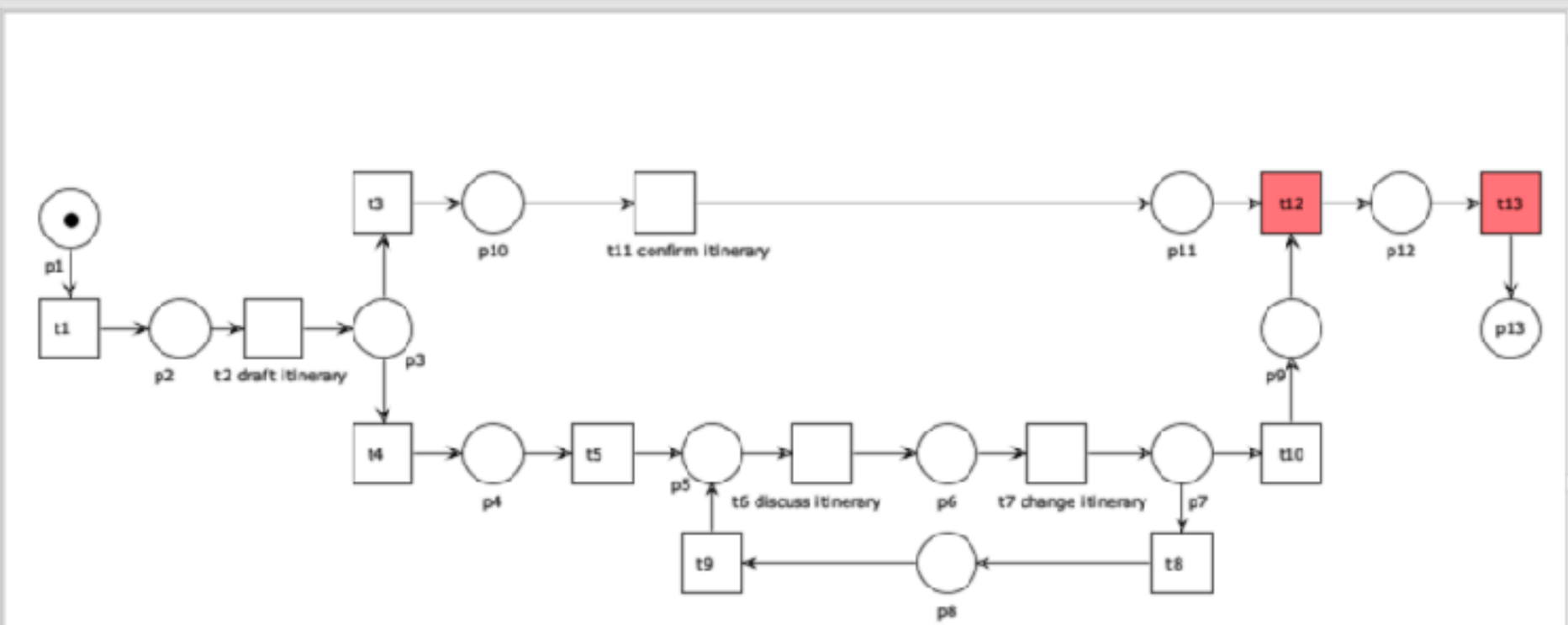
Travel itinerary: step 3



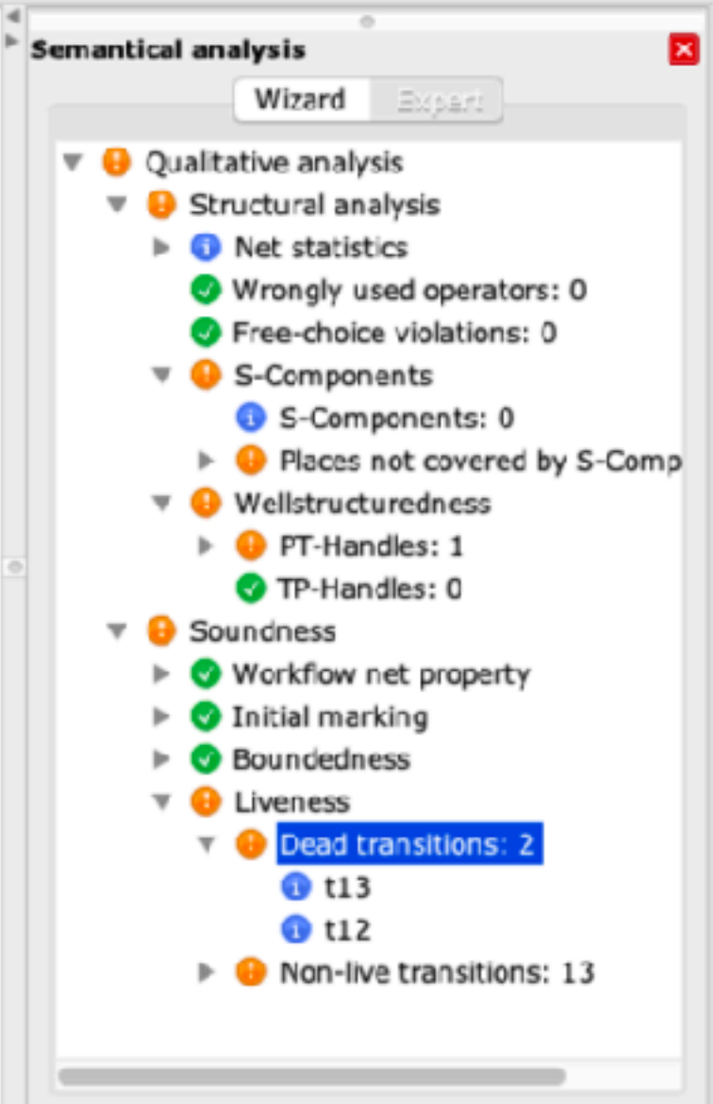
Step 3
enforce
initial place
final place



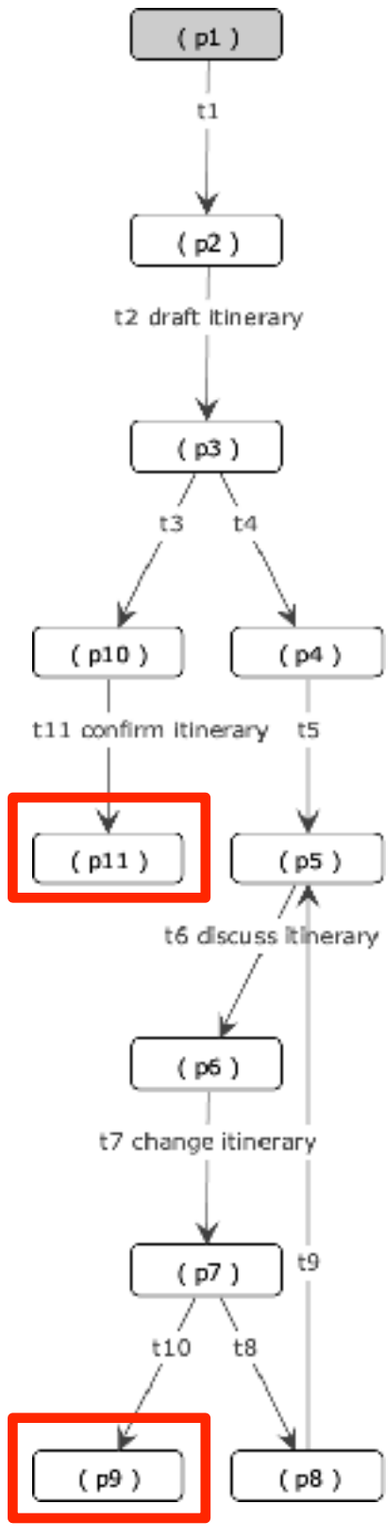
Soundness analysis



Not sound!



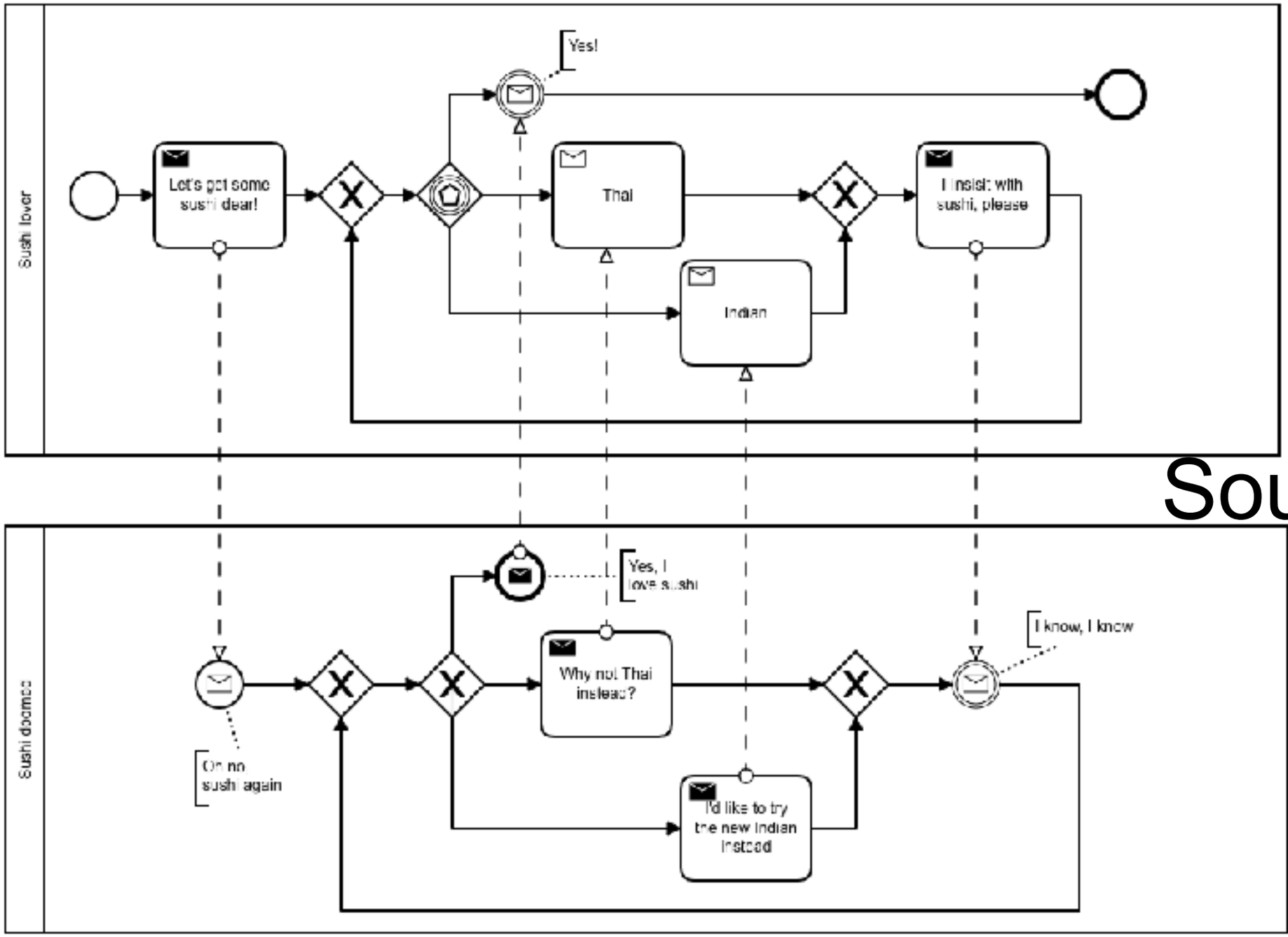
Soundness analysis



Example:

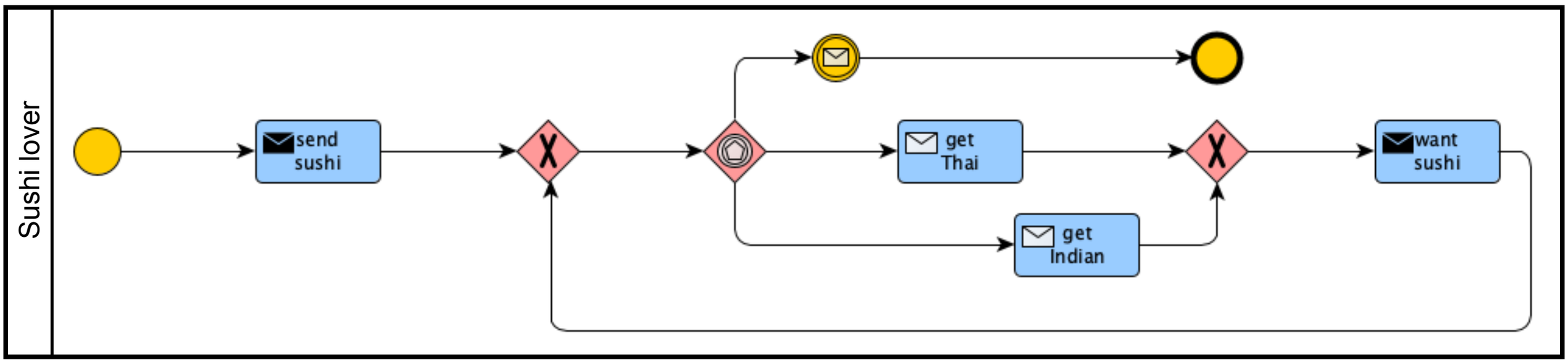
Always sushi

Always sushi



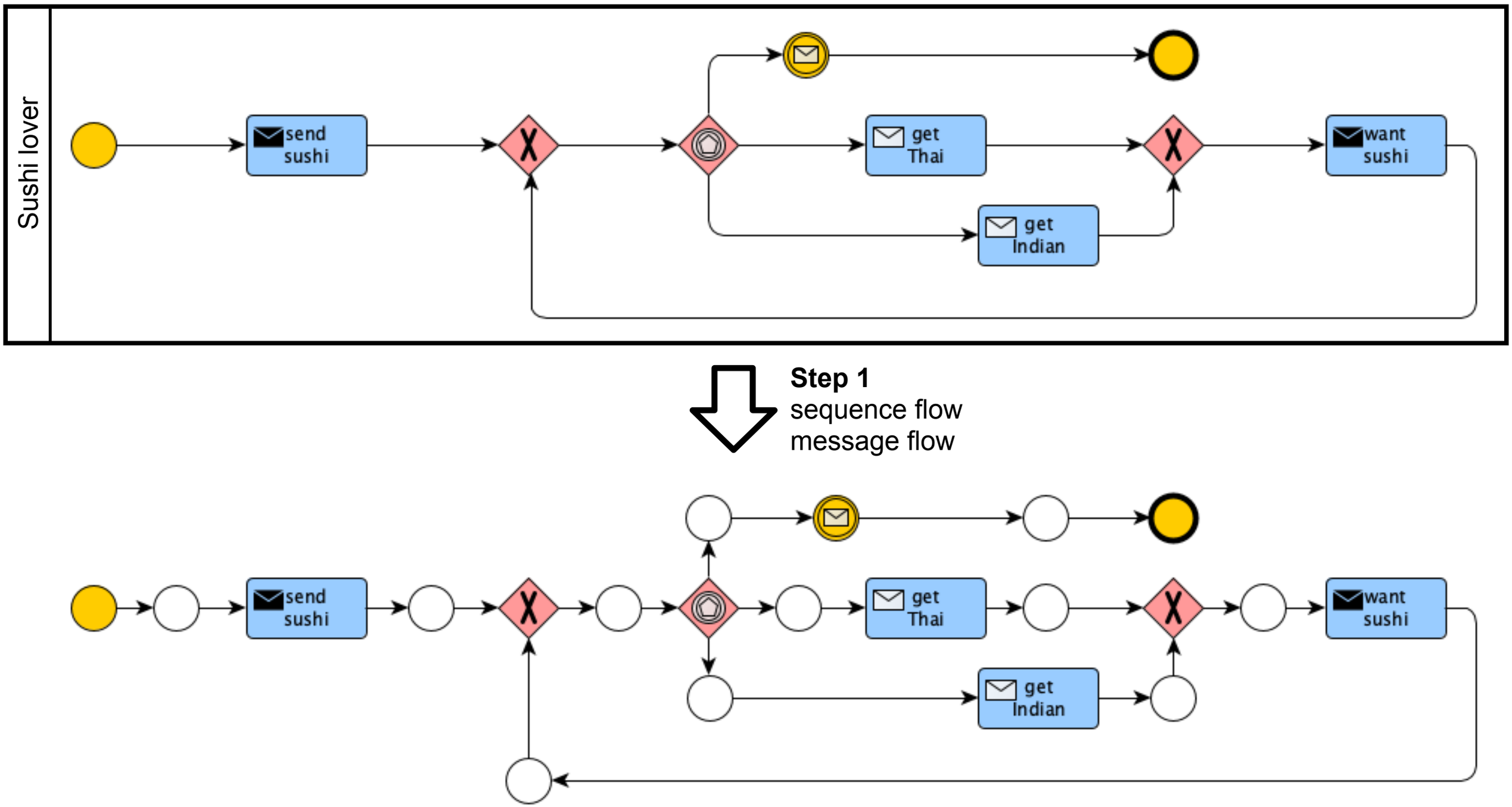
Sound?

Sushi lover

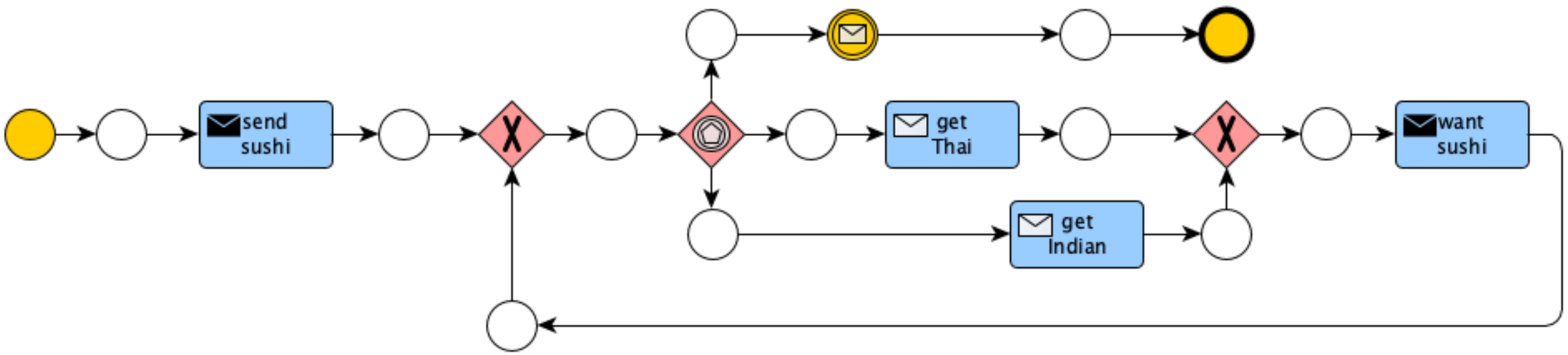


Sound?

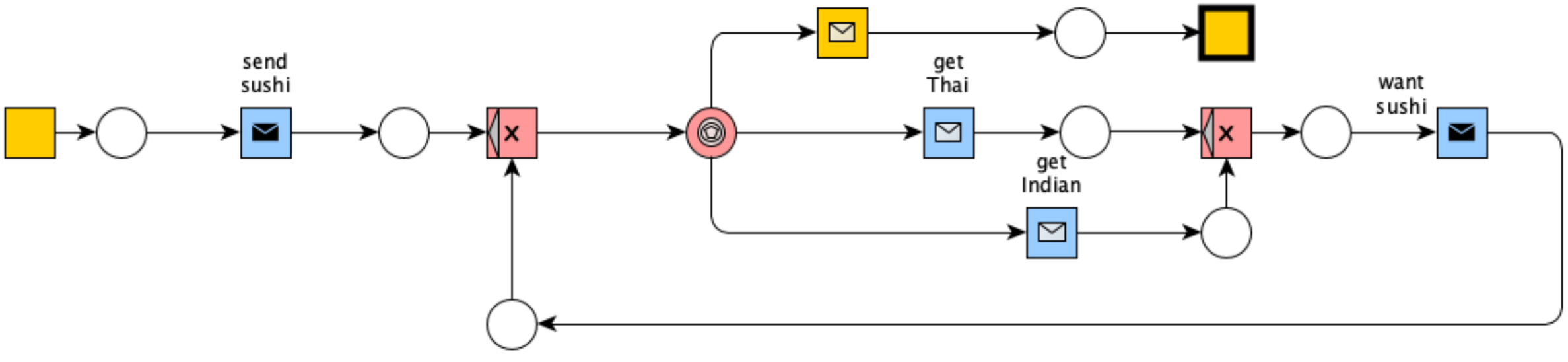
Sushi lover: step 1



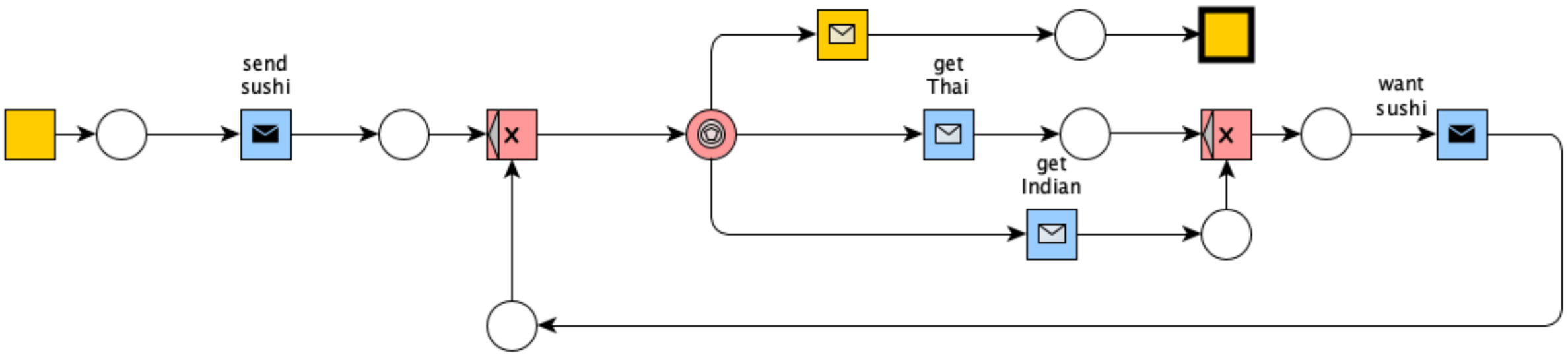
Sushi lover: step 2



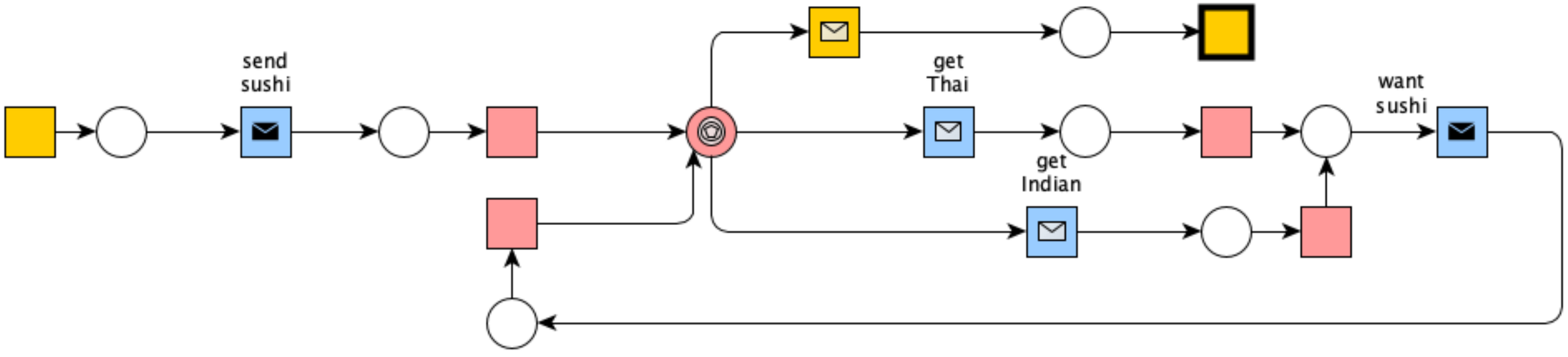
Step 2
flow objects



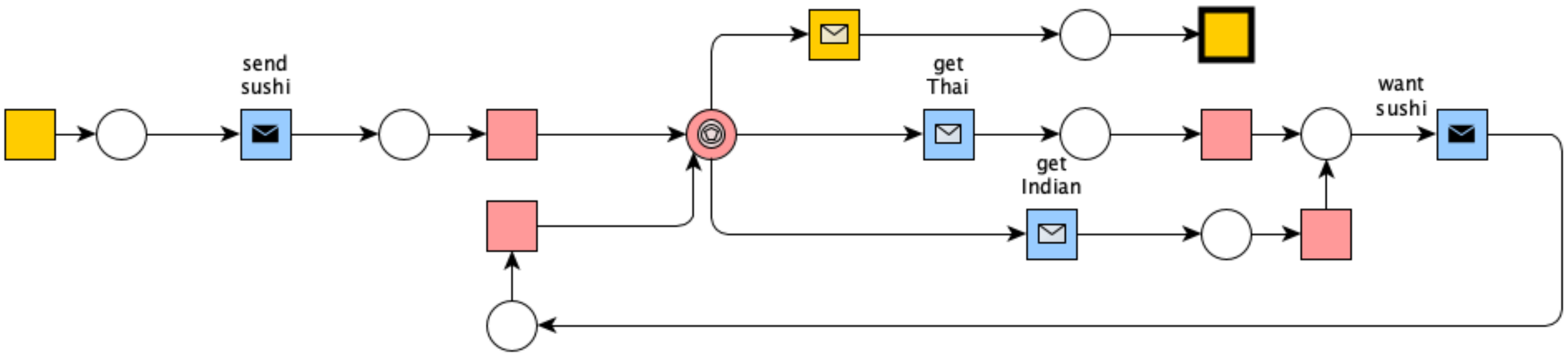
Sushi lover: (desugar)



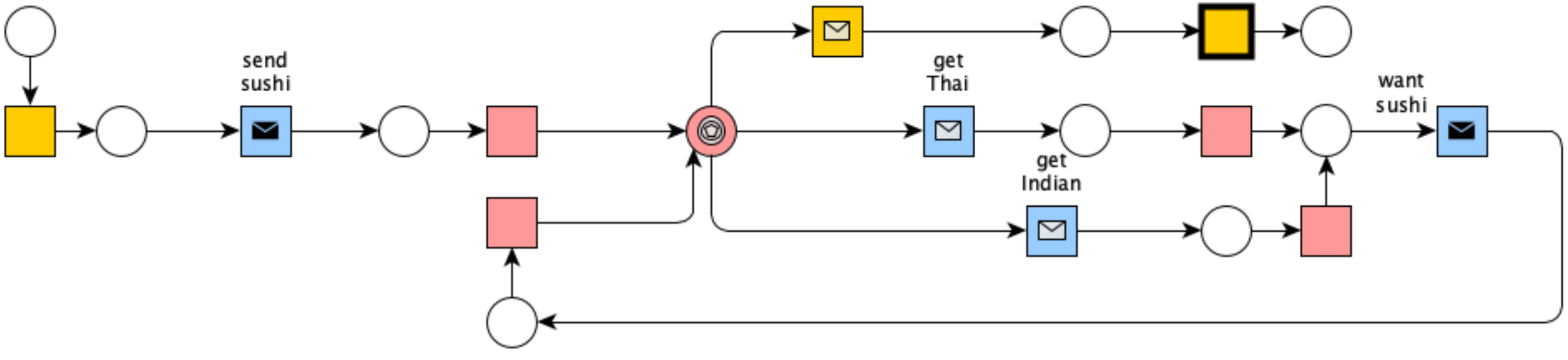
desugar



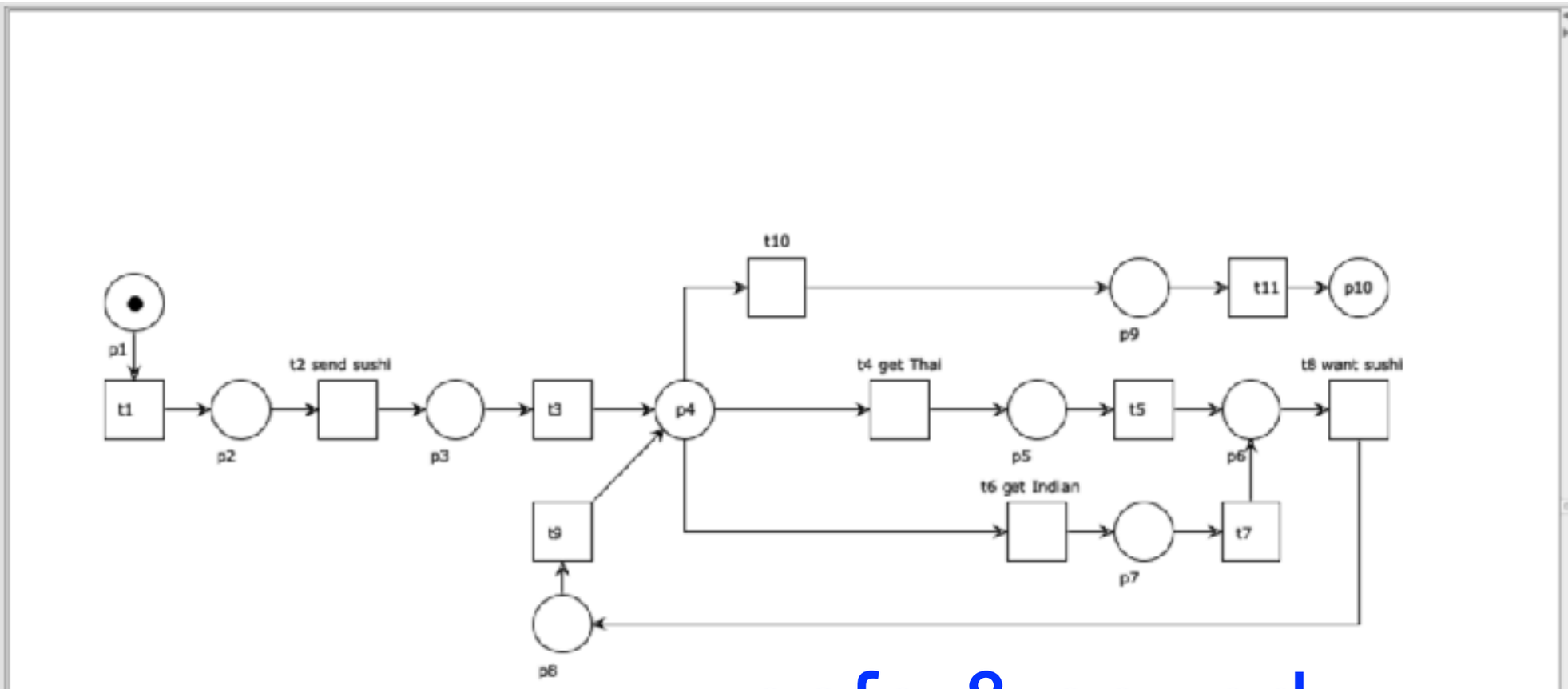
Sushi lover: step 3

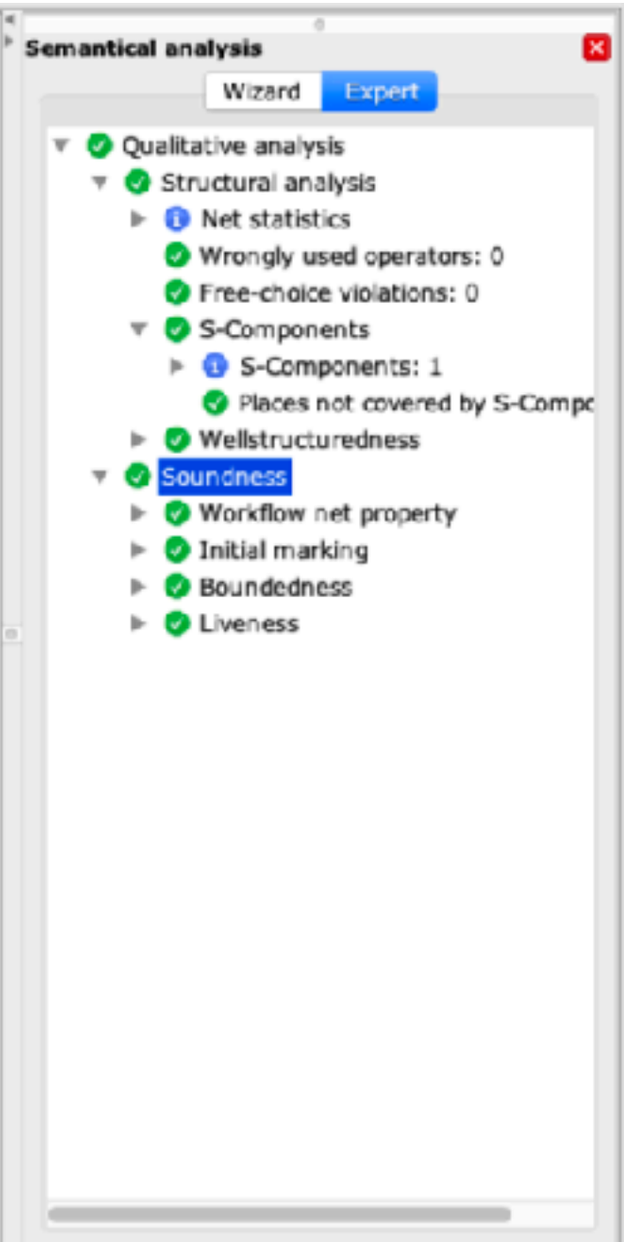


Step 3
enforce
initial place
final place



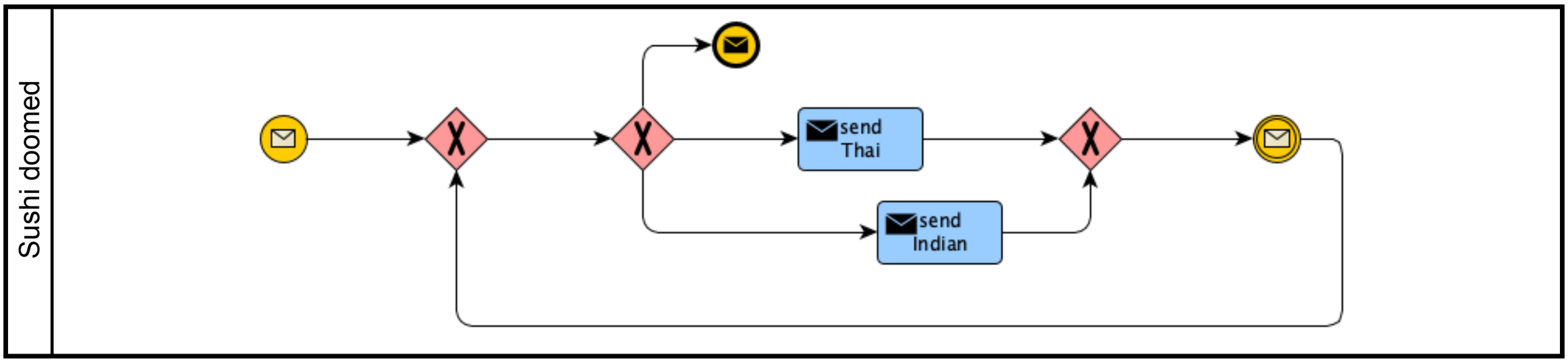
Soundness analysis





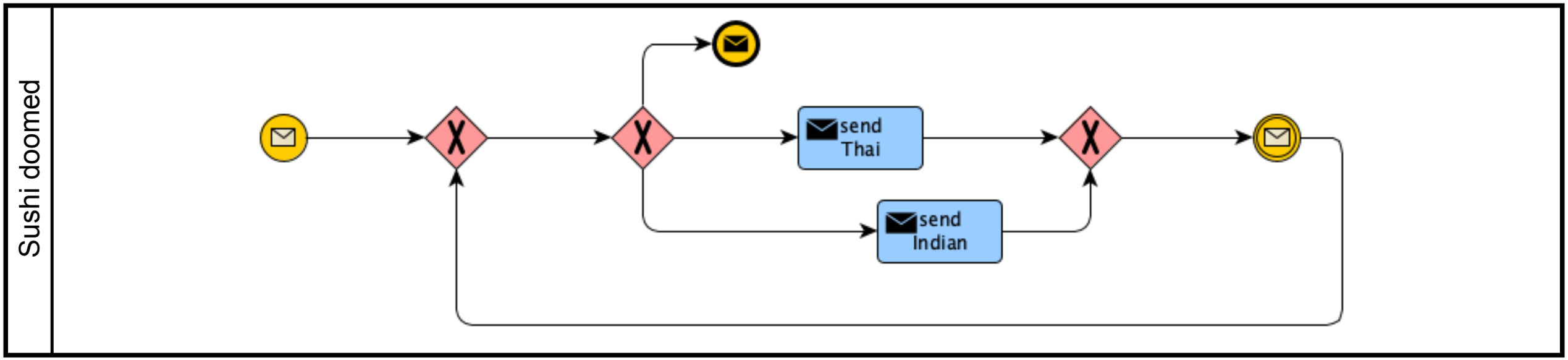
safe & sound
(s-net)

Sushi doomed

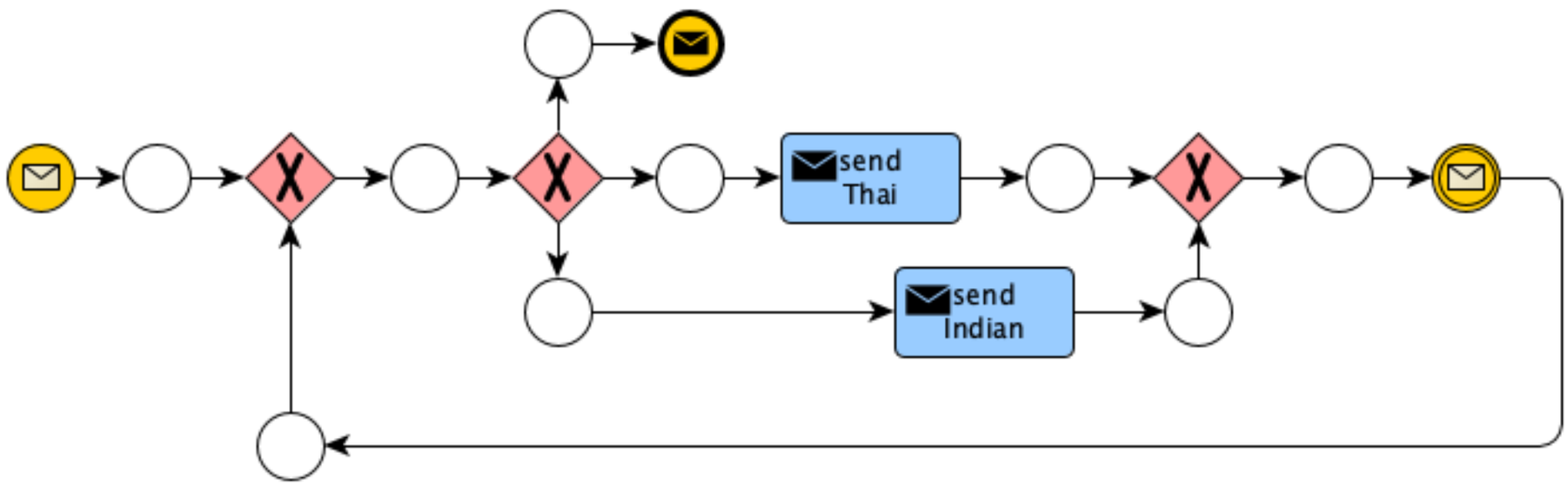


Sound?

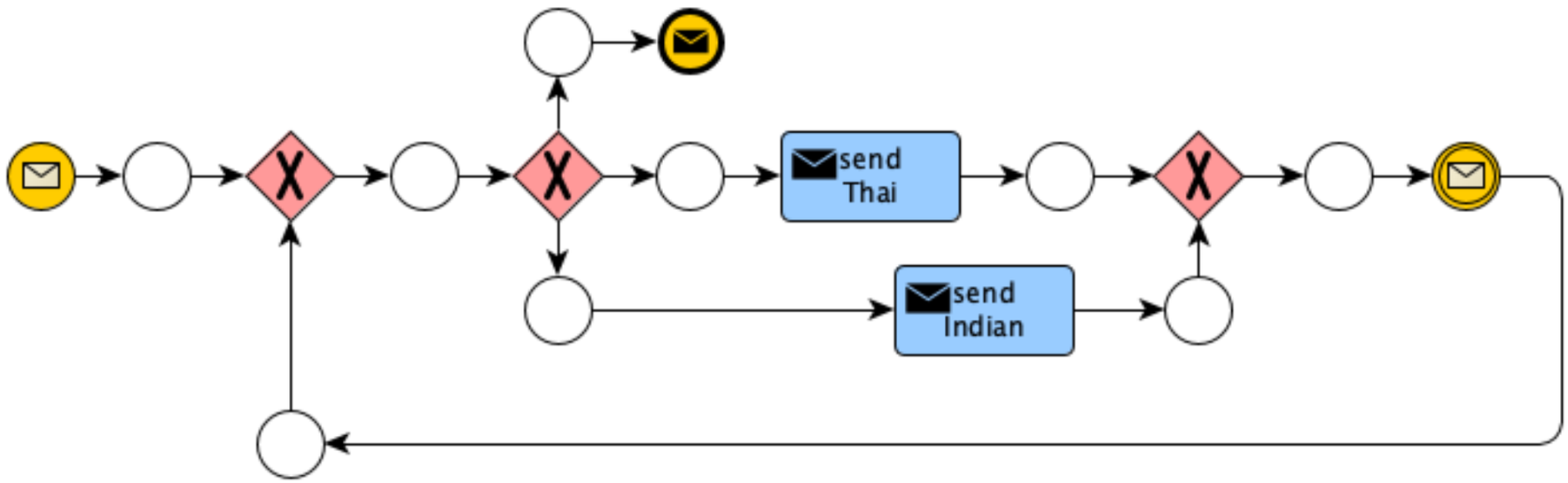
Sushi doomed: step 1



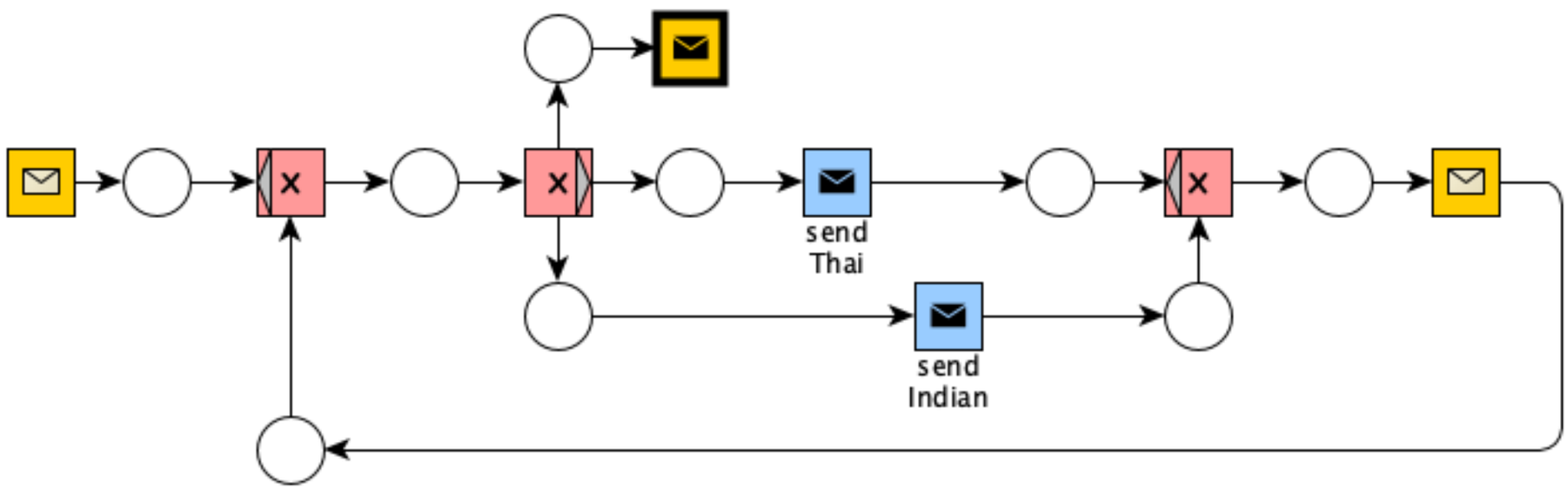
Step 1
sequence flow
message flow



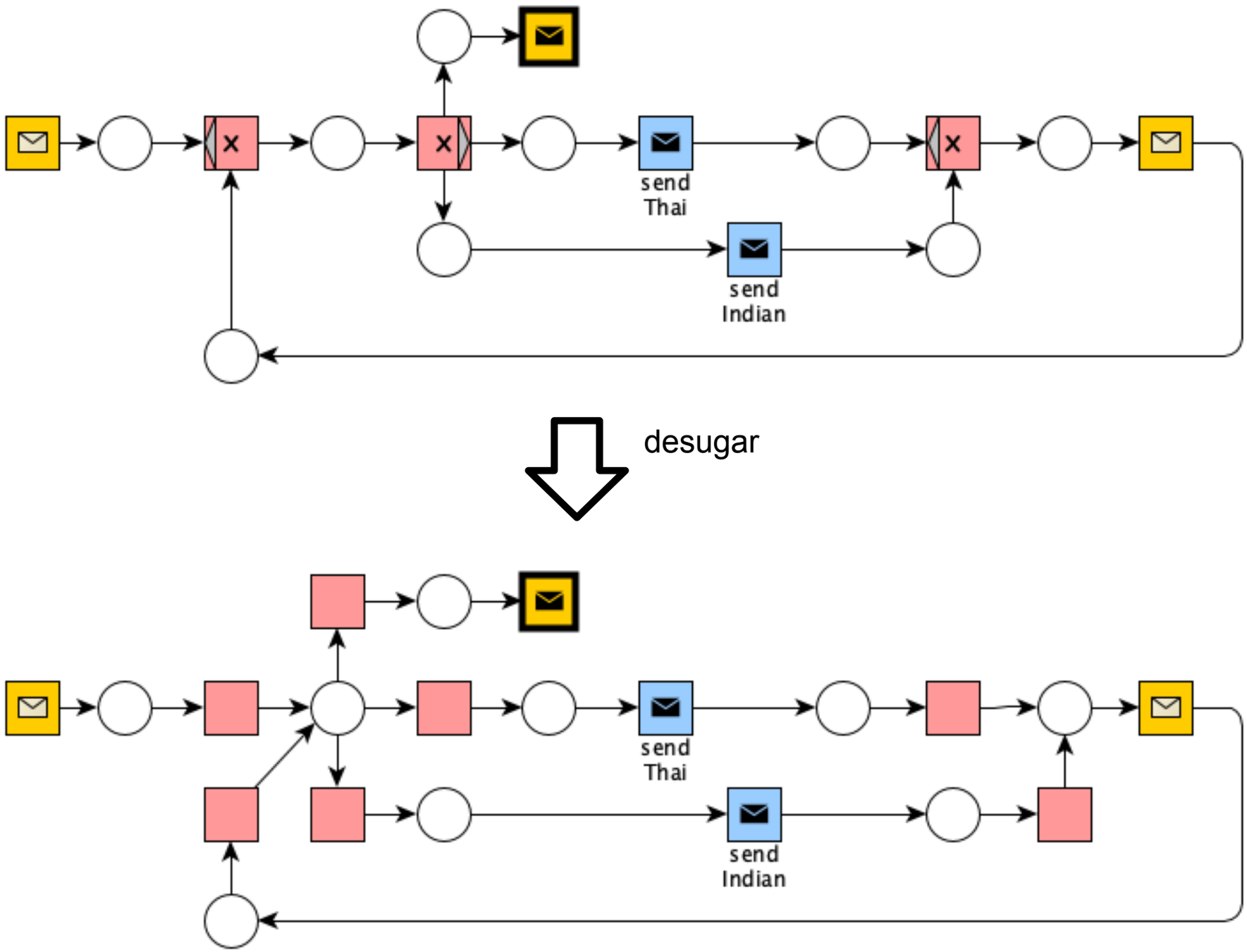
Sushi doomed: step 2



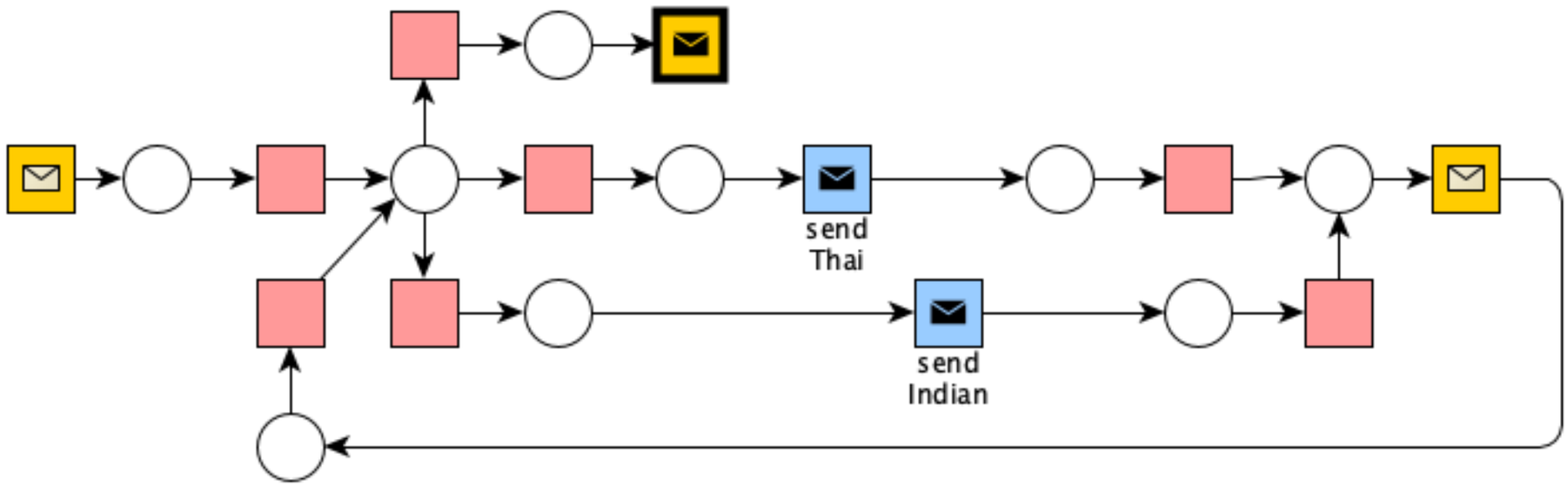
Step 2
flow objects



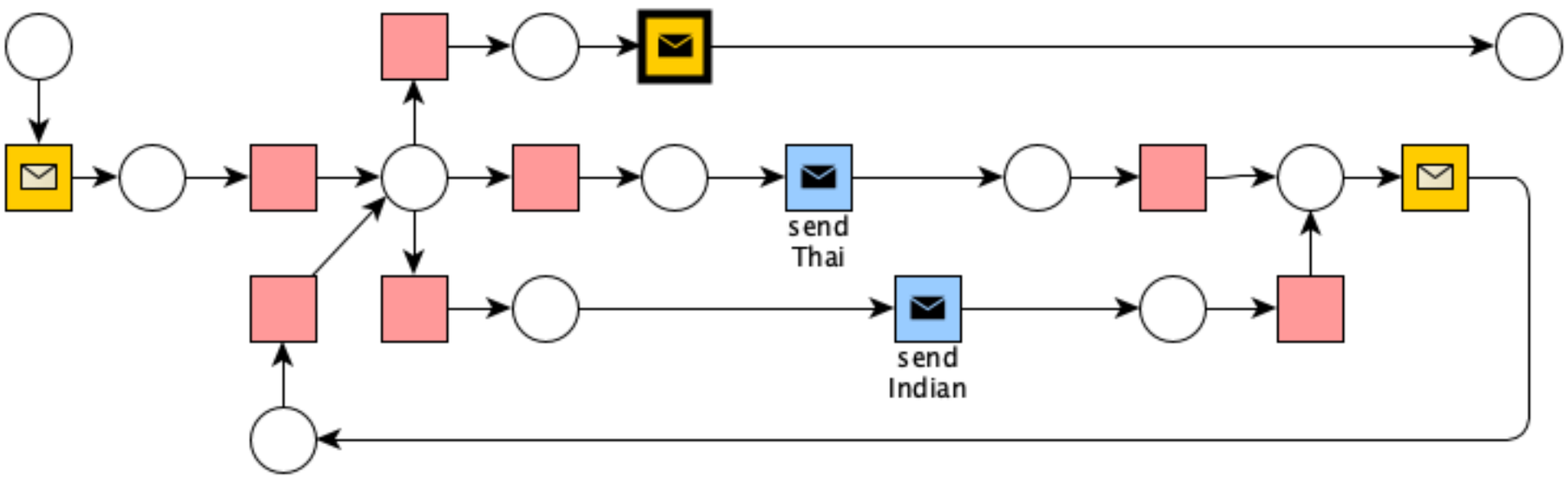
Sushi doomed: (desugar)



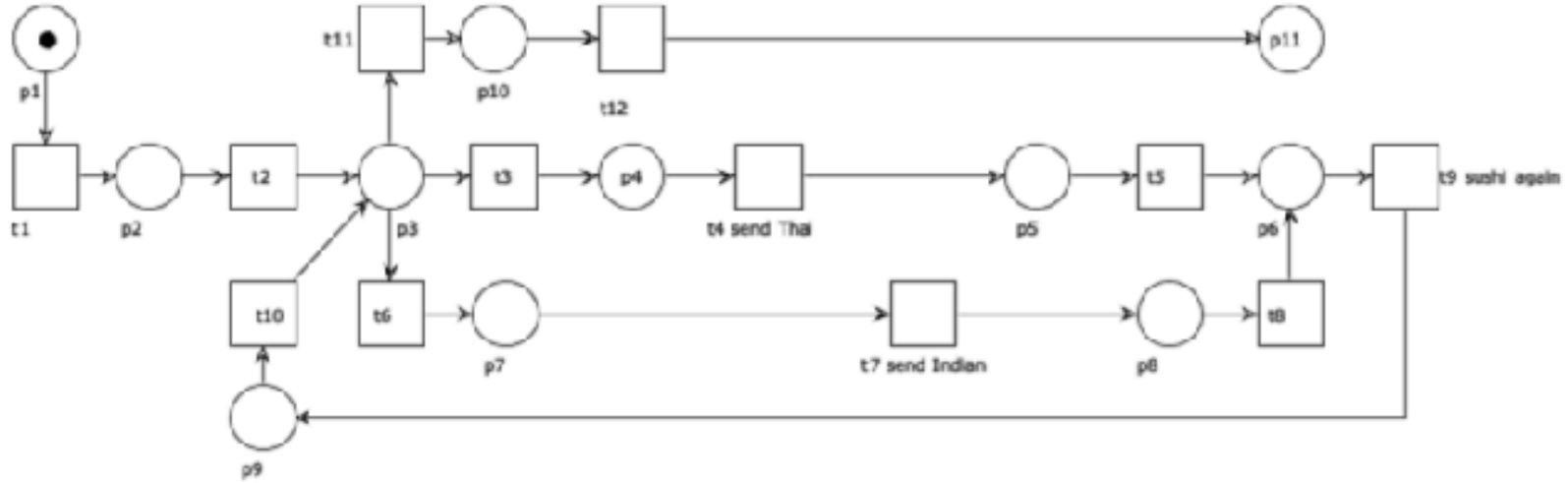
Sushi doomed: step 3

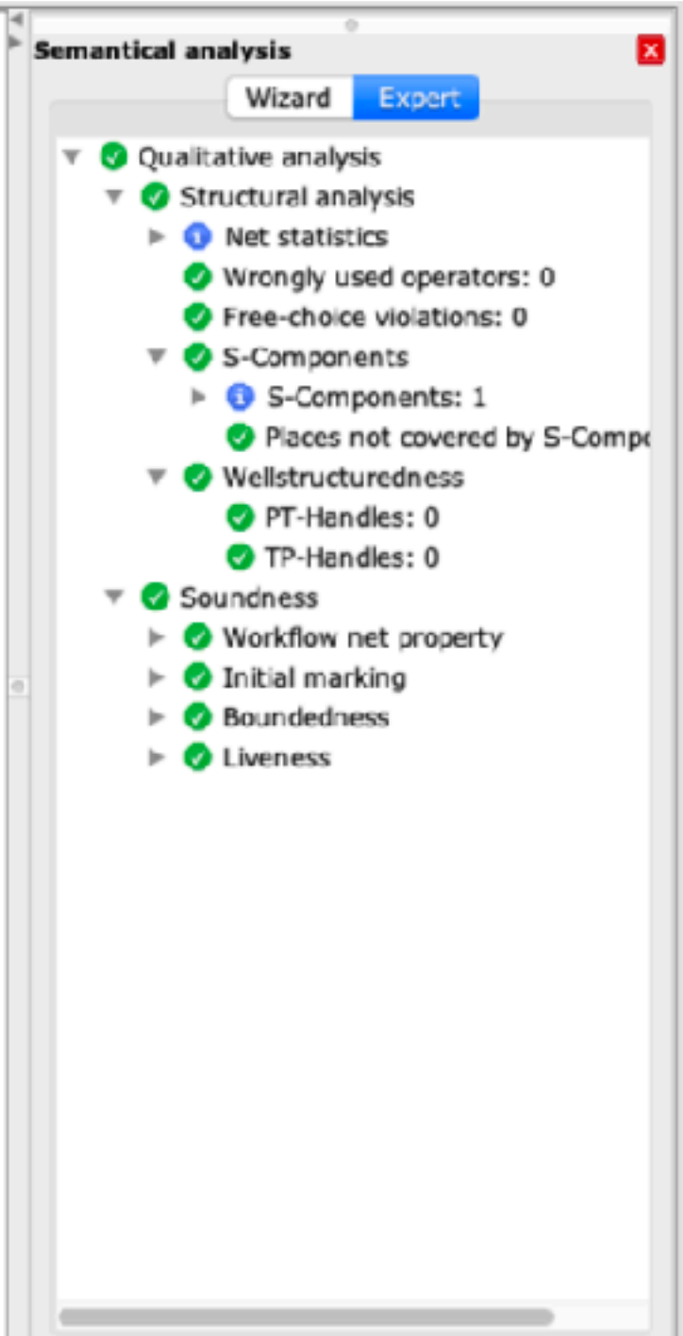


Step 3
enforce
initial place
final place



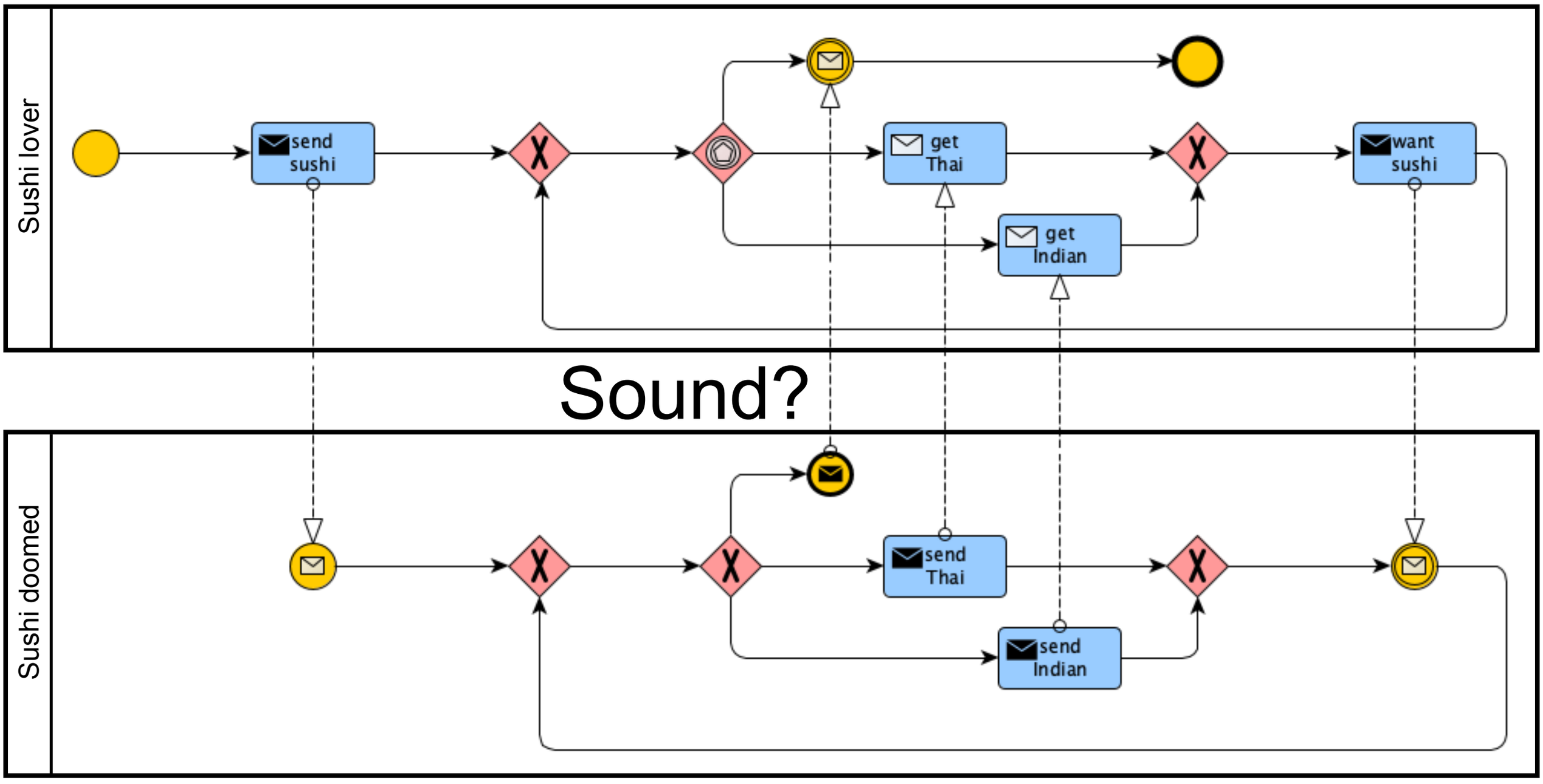
Soundness analysis





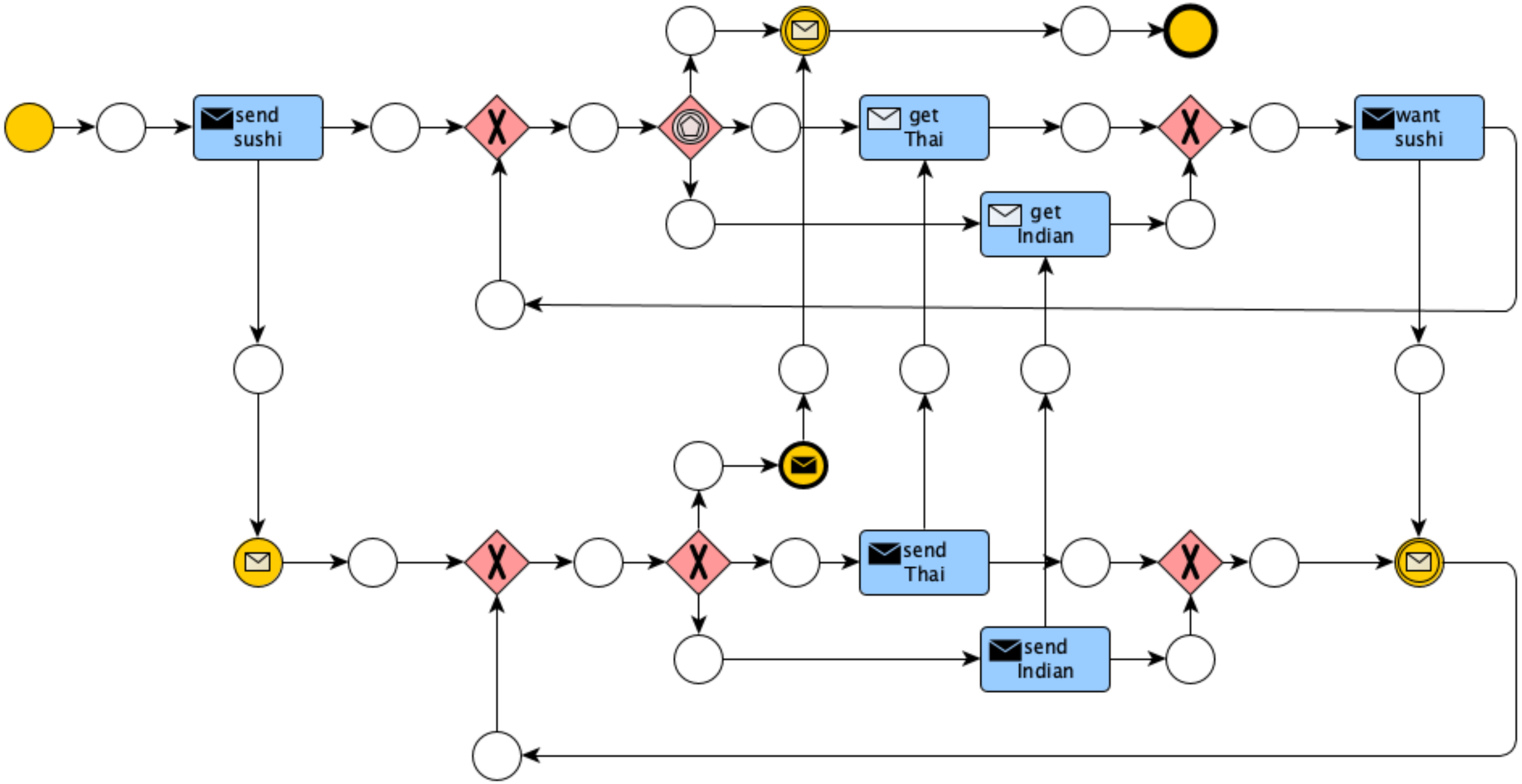
safe & sound
(s-net)

Sushi system

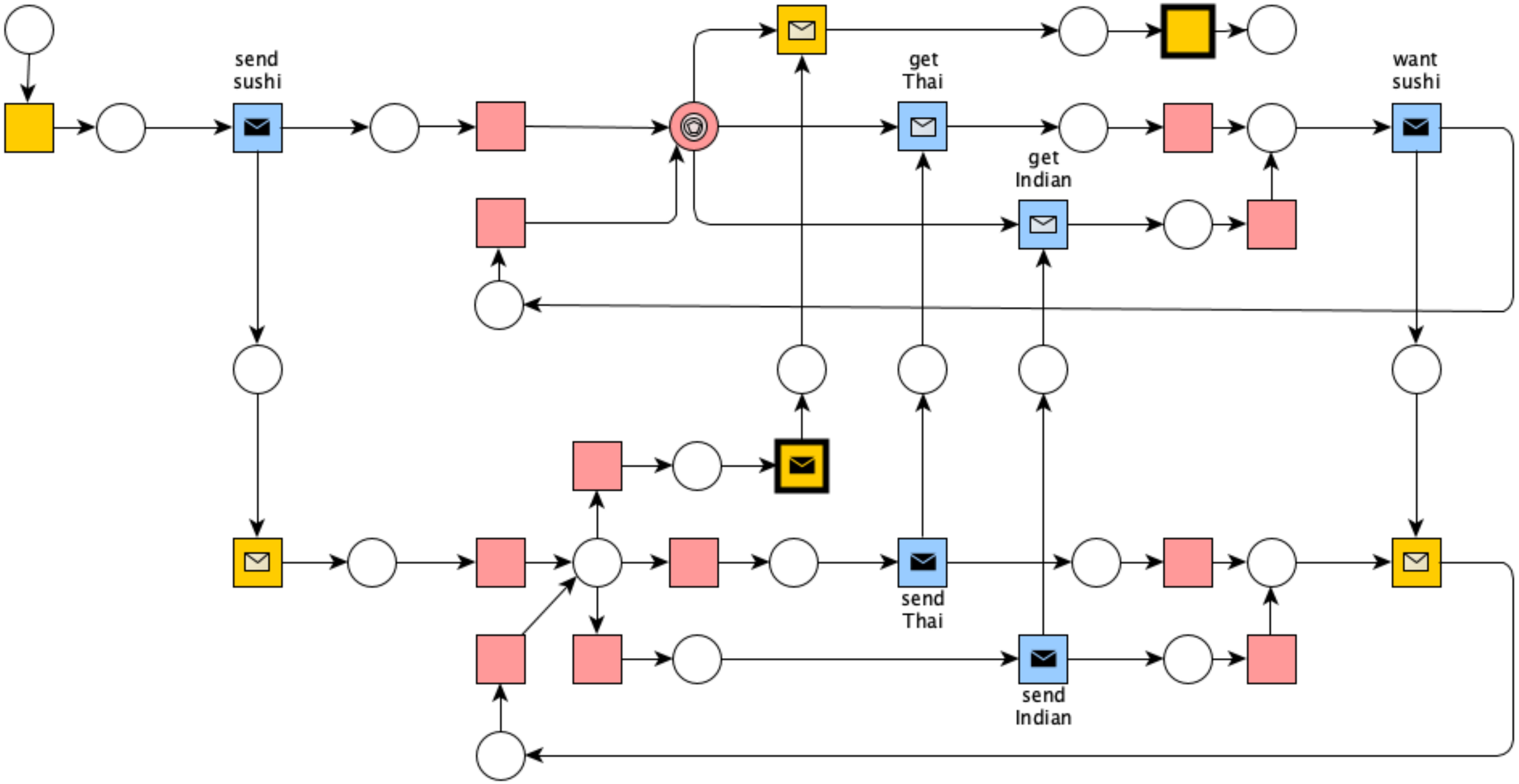


Sound?

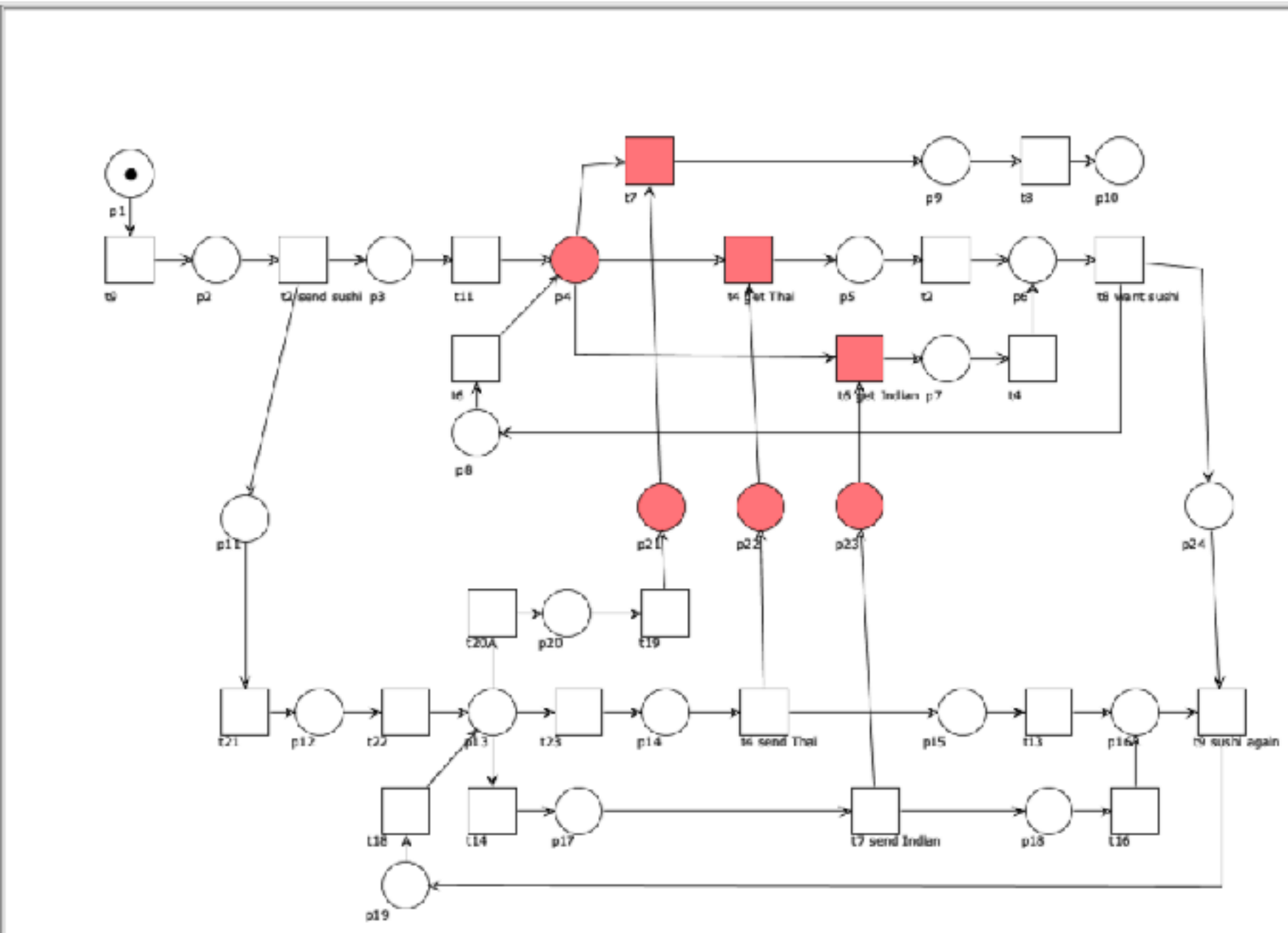
Sushi system: step 1

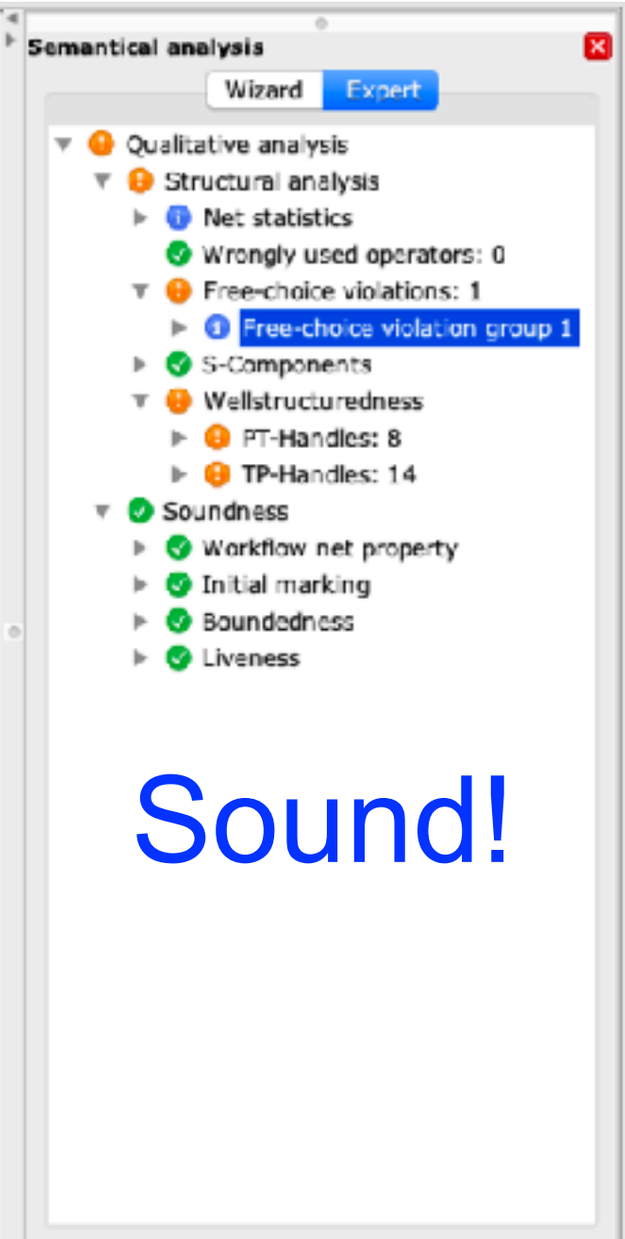


Sushi system: step 1+2+3



Soundness analysis





Sound!

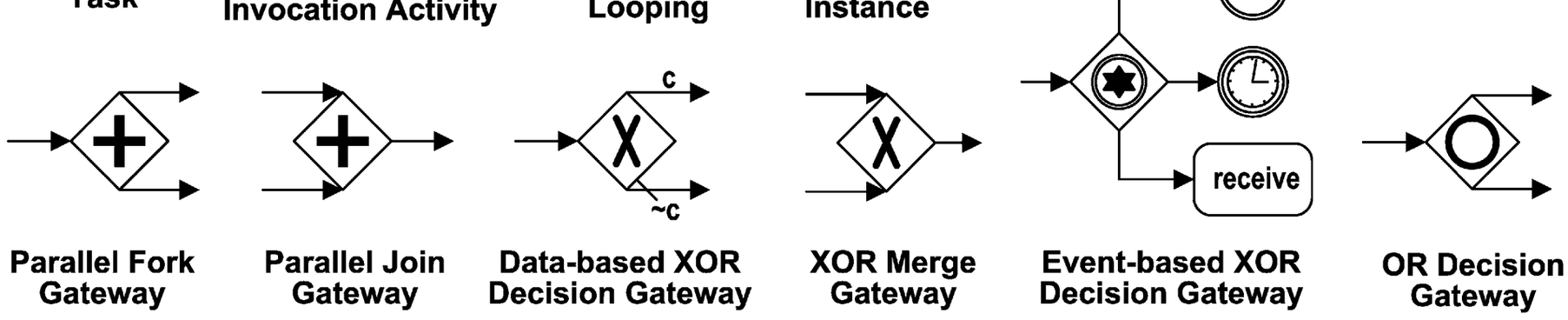
Step 0: preprocessing BPMN diagrams

Overview

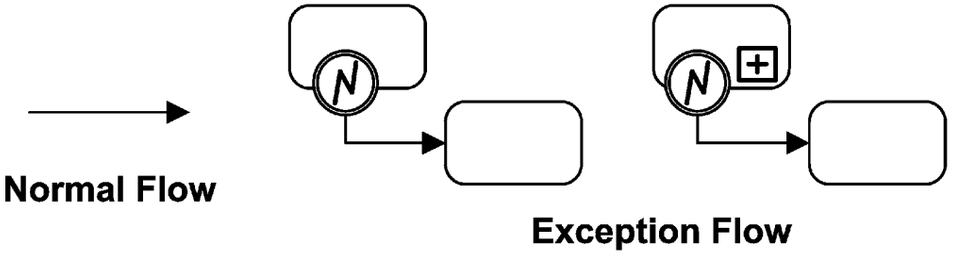
EVENT

ACTIVITY

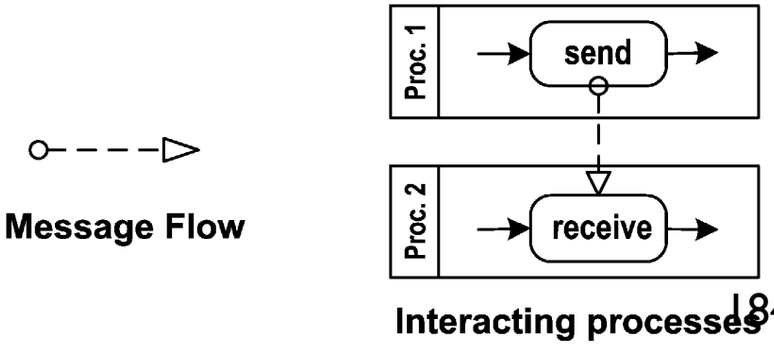
GATEWAY



SEQUENCE FLOW



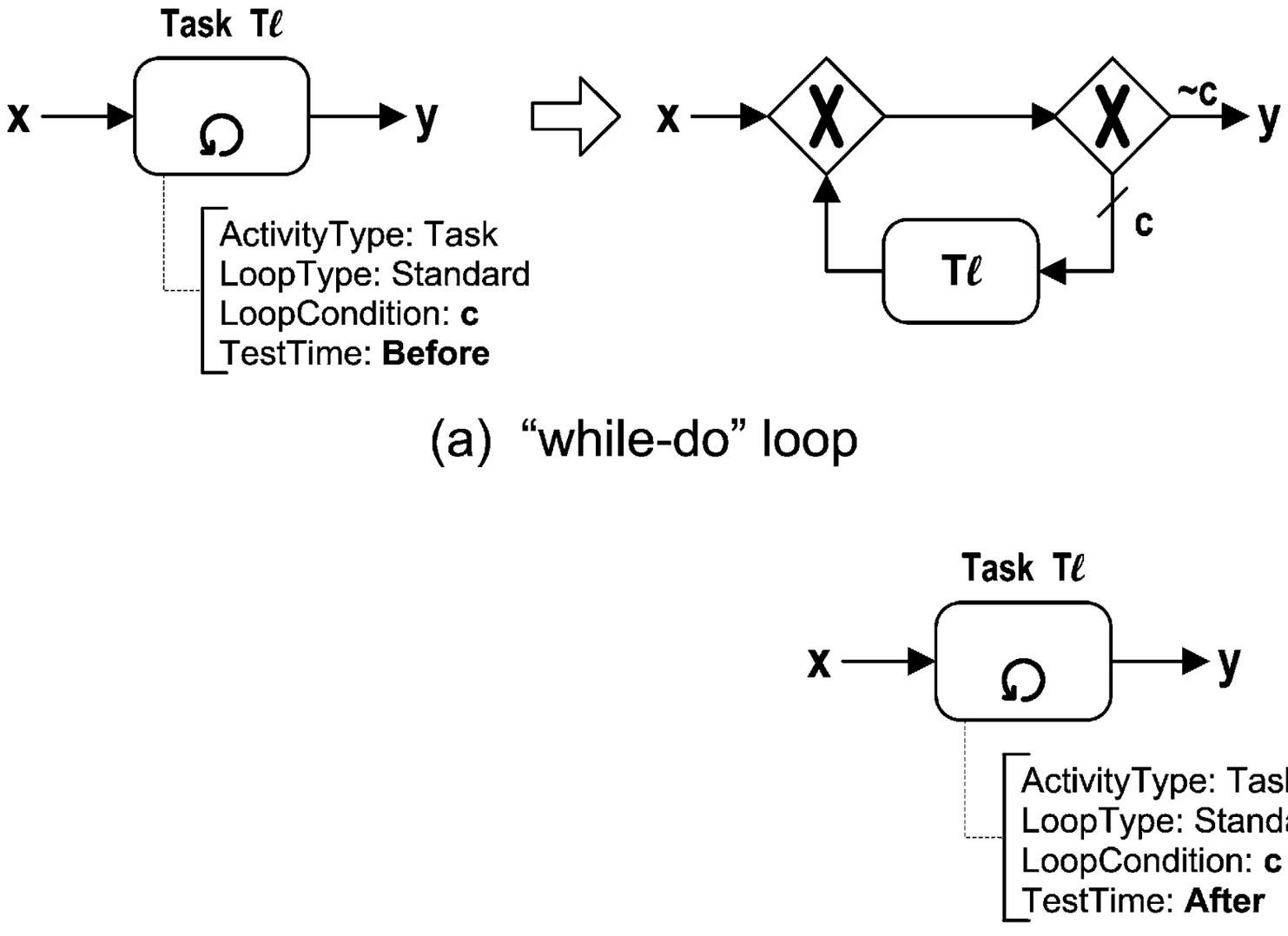
MESSAGE FLOW



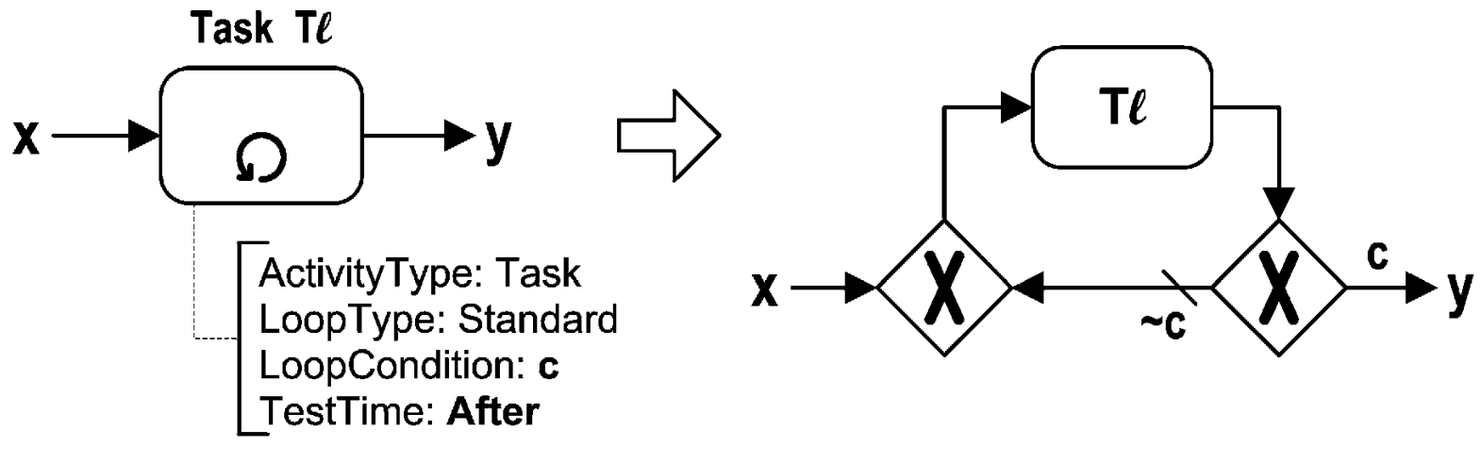
[Note]:

1. **Apart from intermediate error events, intermediate message or timer events may also be the source of exception flows.**
2. **A message flow may link task to task, end event to task, task to start event, and end event to start event.**

Activity looping

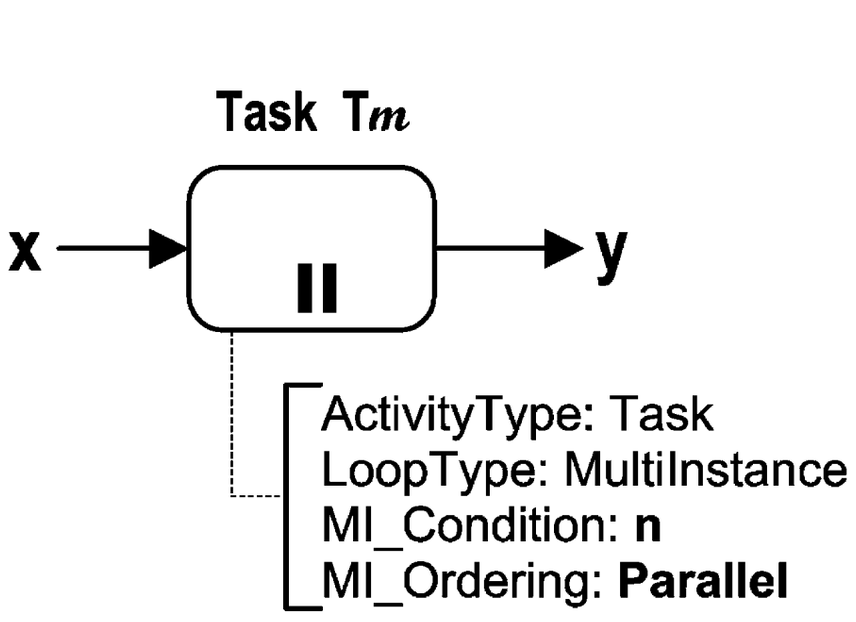


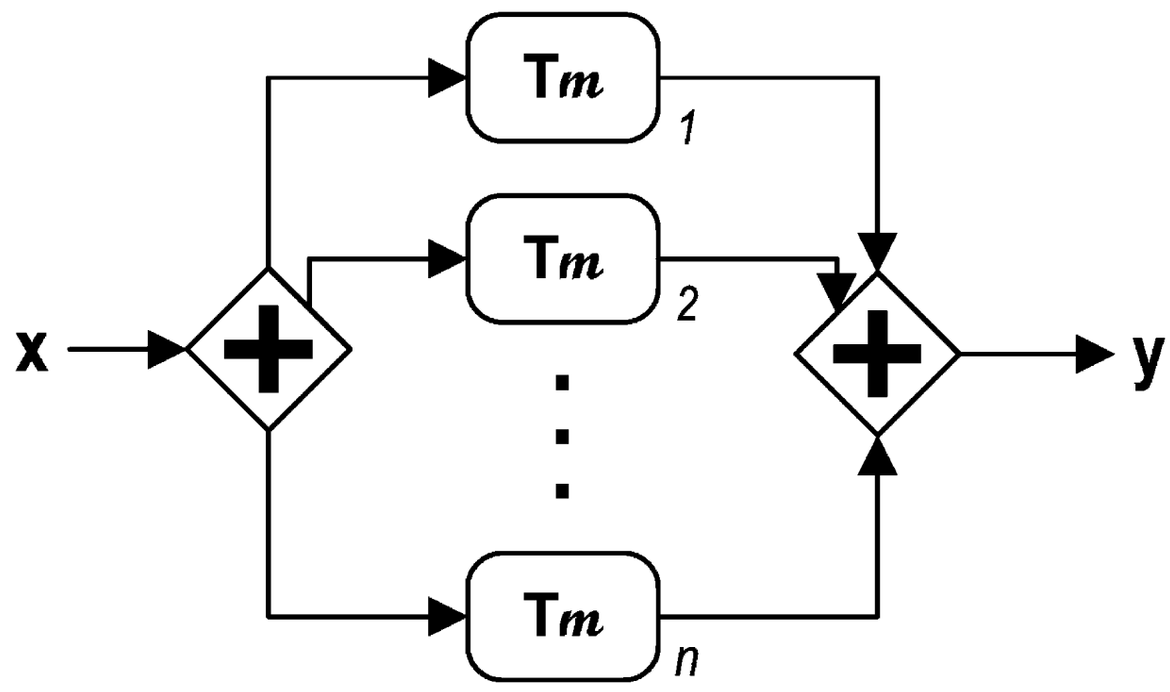
(a) “while-do” loop



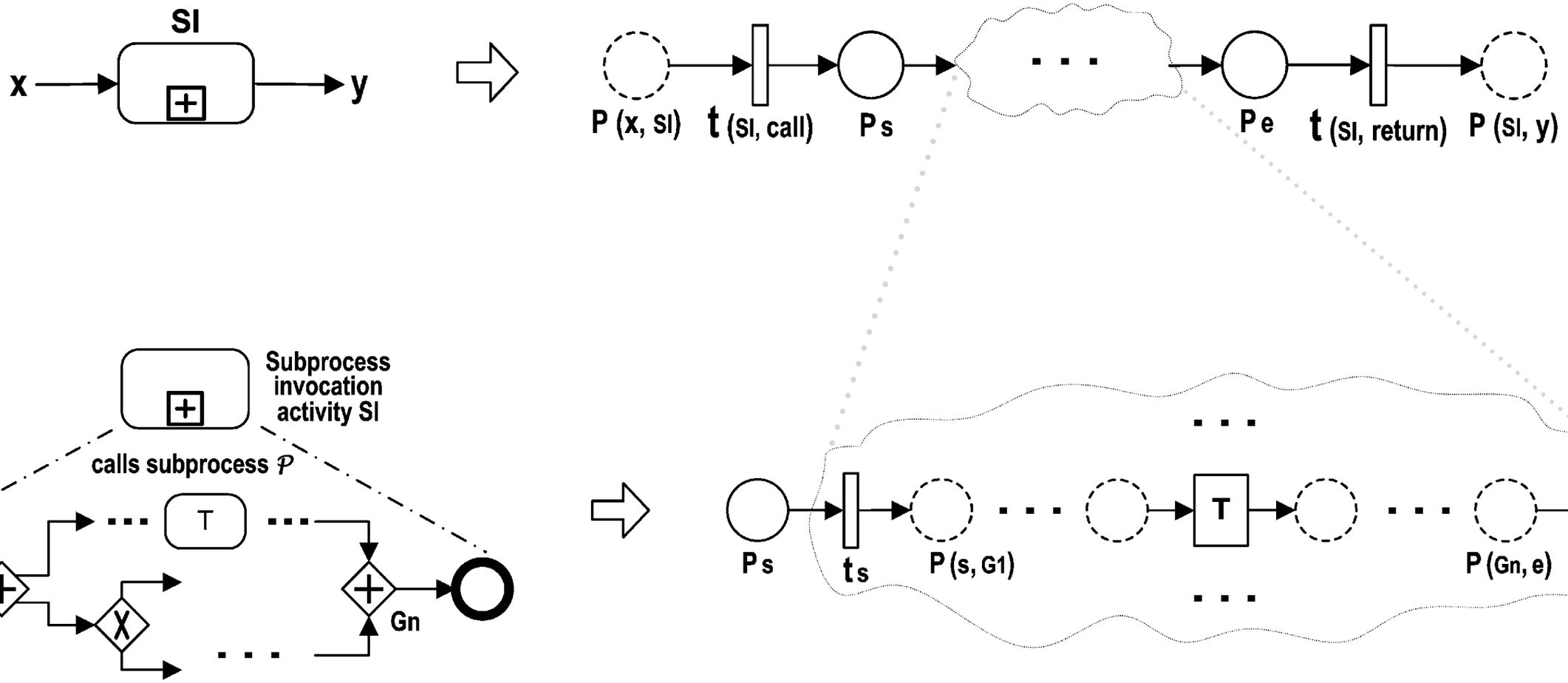
(b) “do-until” loop

Multiple instances (design-time bounded)

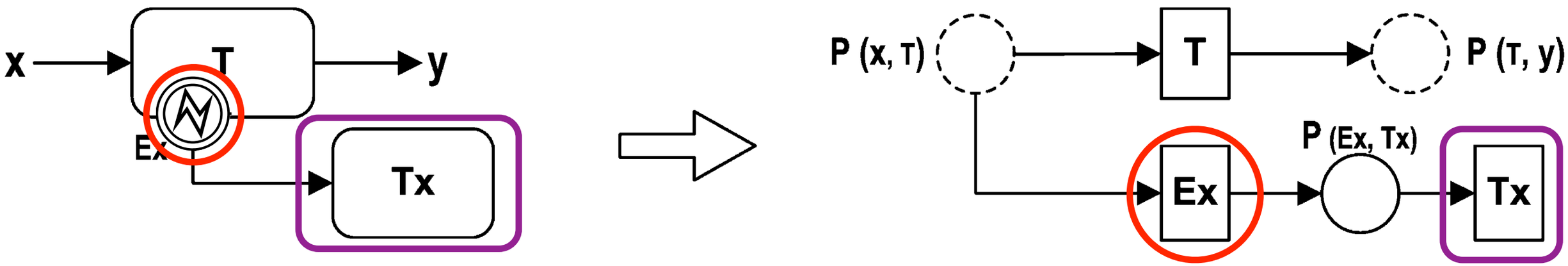




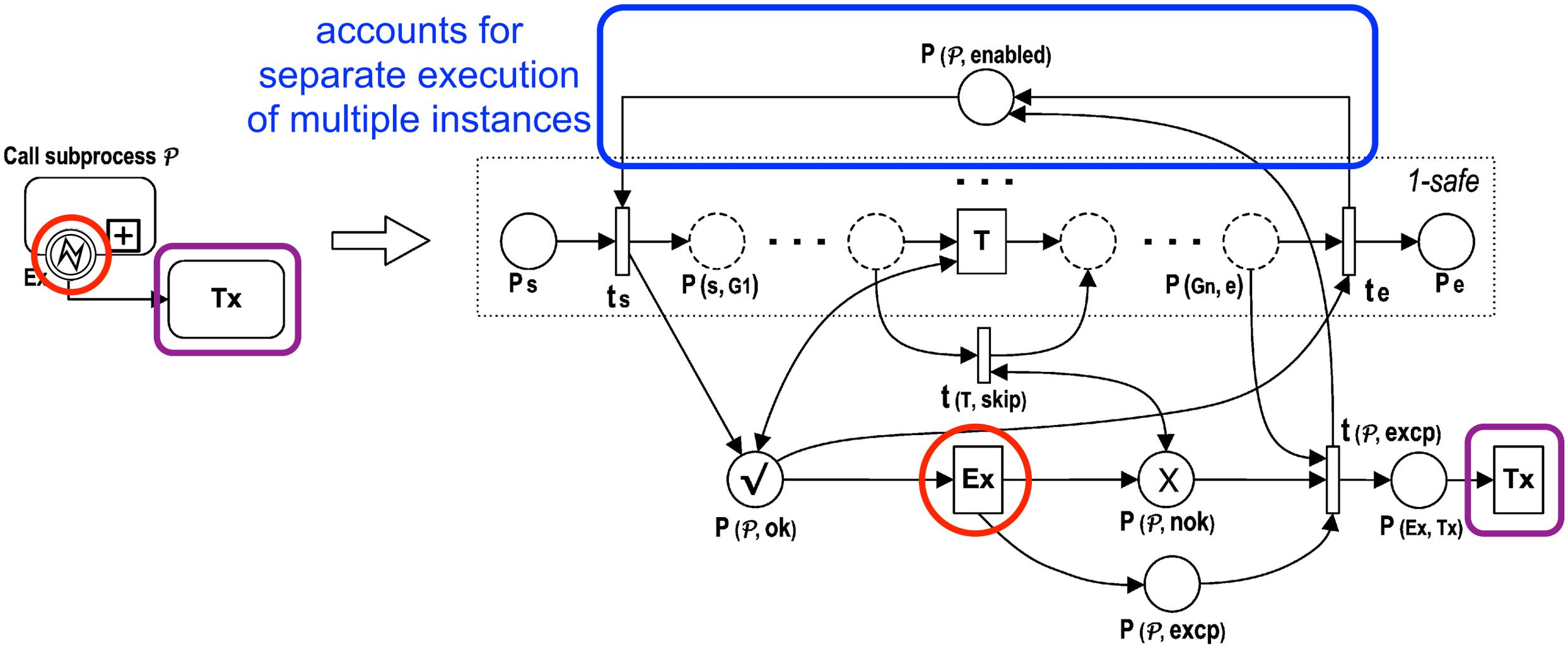
Sub-processes



Exception handling: single task



Exception handling: sub-processes



Exercises

Model the following fragments of business processes for assessing loan applications:

Example: loan application 1

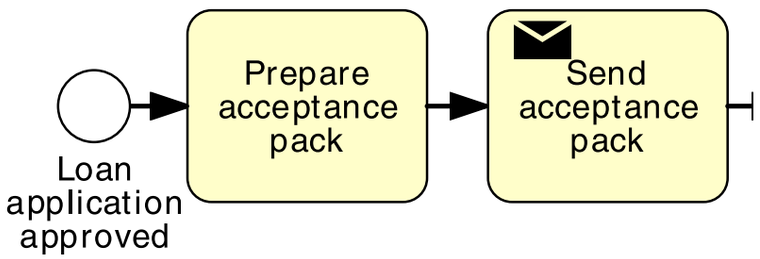
Once a loan application has been **approved** by the loan provider, an acceptance pack is **prepared** and **sent** to the customer.

The acceptance pack includes a repayment schedule which the customer needs to agree upon by **sending the signed documents** back to the loan provider.

The latter then verifies the repayment agreement:
if the applicant disagreed with the repayment schedule, the loan provider **cancels** the application;
if the applicant agreed, the loan provider **approves** the application.
In either case, the process completes with the loan provider **notifying** the applicant of the application status.

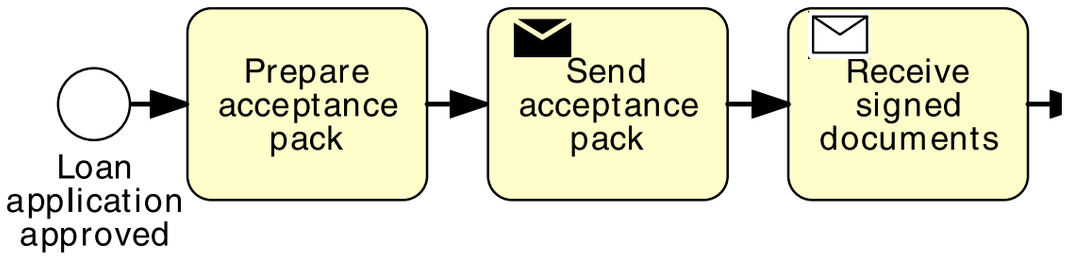
Example

Once a loan application has been **approved** by the loan provider, an acceptance pack is **prepared** and **sent** to the customer.

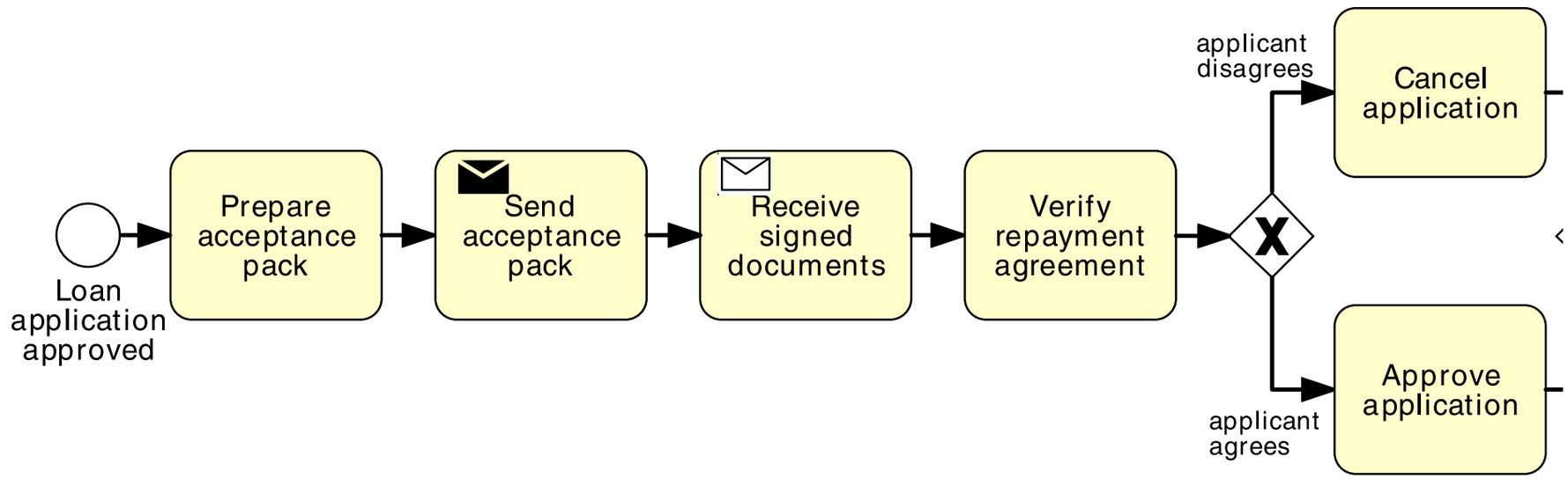


Example

The acceptance pack includes a repayment schedule which the customer needs to agree upon by **sending the signed documents** back to the loan provider.

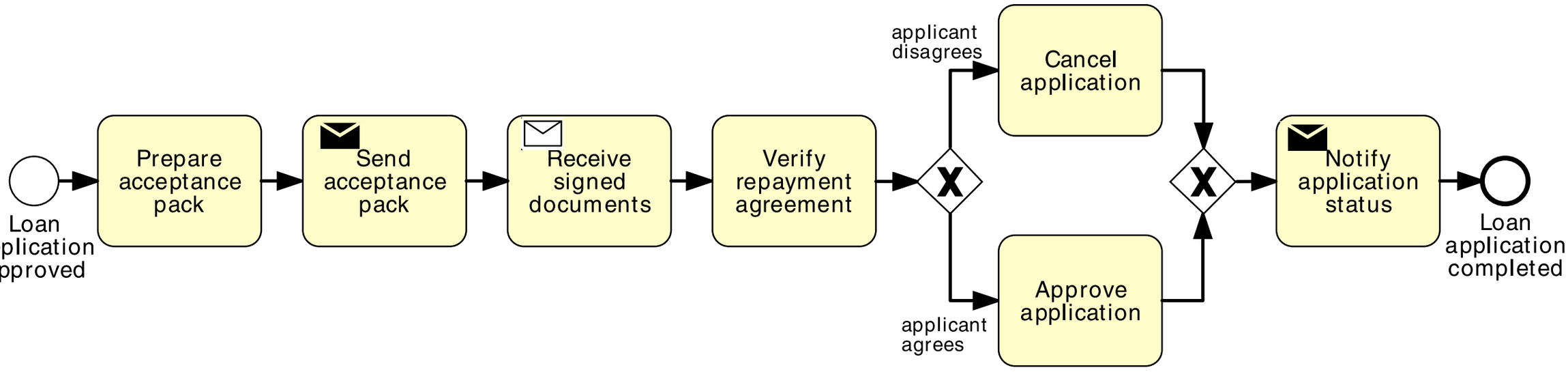


Example



The latter then verifies the repayment agreement:
if the applicant disagreed with the repayment schedule, the loan provider **cancels** the application;
if the applicant agreed, the loan provider **approves** the application.

Example



In either case, the process completes with the loan provider **notifying** the applicant of the application status.

Exercise: loan application 2

A loan application is **approved** if it passes **two checks**:
(i) the applicant's loan **risk assessment**, which is done automatically by a system, and
(ii) the **appraisal** of the property for which the loan has been asked, carried out by a property appraiser.

The risk assessment requires a **credit history check** on the applicant, which is performed by a financial officer.

Once both the loan risk assessment and the property appraisal have been performed, a loan officer can **assess** the applicant's eligibility.

If the applicant is not eligible, the application is **rejected**,
otherwise the acceptance pack is **prepared and sent** to the applicant.

Exercise: loan application 3

A loan application may be coupled with a home insurance which is offered at discounted prices.
The applicant may express their interest in a home insurance plan at the time of submitting their loan application to the loan provider.

Based on this information, **if** the loan application is **approved**, the loan provider may **either only** send an **acceptance pack** to the applicant, **or also** send a **home insurance quote**.

The process then continues with the **verification** of the repayment agreement.

Exercise: loan application 4

Once a loan application is **received** by the loan provider, and before proceeding with its assessment, the application itself needs to be **checked** for completeness.

If the application is incomplete, it is **returned** to the applicant, so that they can **fill out** the missing information and **send it back** to the loan provider.

This process is **repeated** until the application is complete.

Exercise: loan application 5

Put together the four fragments of the loan assessment process that you created in previous Exercises.

Then extend the resulting model by adding all the required artifacts.

Moreover, attach annotations to specify the business rules behind:
(i) checking an application completeness,
(ii) assessing an application eligibility, and
(iii) verifying a repayment agreement.

Exercise: loan application 6

Extend the business process for assessing loan applications that you created in previous exercises by considering the following resource aspects.

The process for assessing loan applications is executed by four roles within the **loan provider**:
a **financial officer** takes care of checking the applicant's credit history;
a **property appraiser** is responsible for appraising the property;
an **insurance sales representative** sends the home insurance quote to the applicant if this is required.
All other activities are performed by the **loan officer** who is the main point of contact with the applicant.

Exercises: loan application 7

Extend the loan application model by representing the interactions between the loan provider and the applicant.