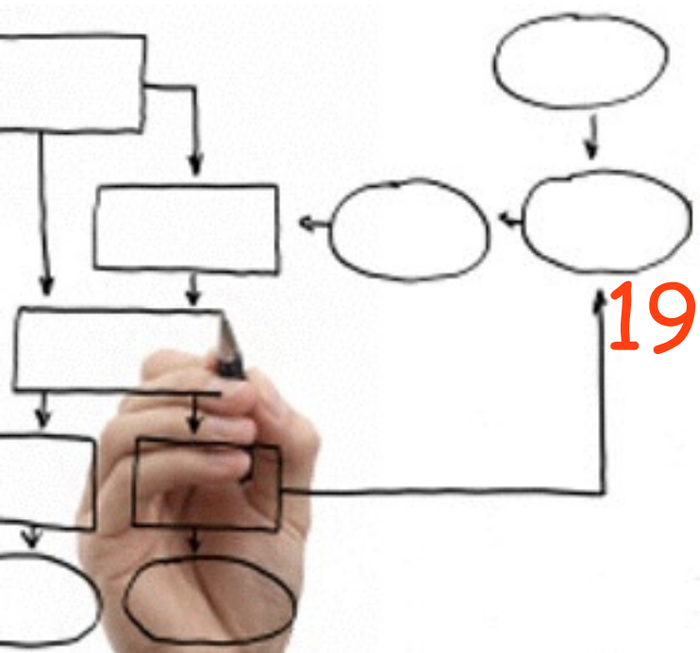
Methods for the specification and verification of business processes

MPB (6 cfu, 295AA)

Roberto Bruni

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



19 - Event-driven process chains

Object

We overview EPC and the main challenges that arise when analysing them with Petri nets

Ch.4.3, 6 of Business Process Management: Concepts, Languages, Architectures

Event-driven Process Chain

An **Event-driven Process Chain** (EPC) is a particular type of flow-chart that can be used for configuring an Enterprise Resource Planning (ERP) implementation

Supported by many tools (e.g. SAP R/3)

EPC Markup Language available (EPML) as interchange format

EPC overview

Important notation to model the domain aspects of business processes

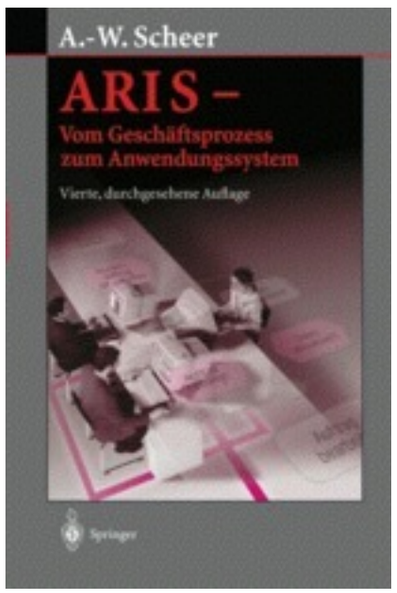
Rather informal notation

EPC focus is on representing domain concepts and processes (not their formal aspects and technical realization)

It can be used to drive the modelling, analysis and redesign of business process

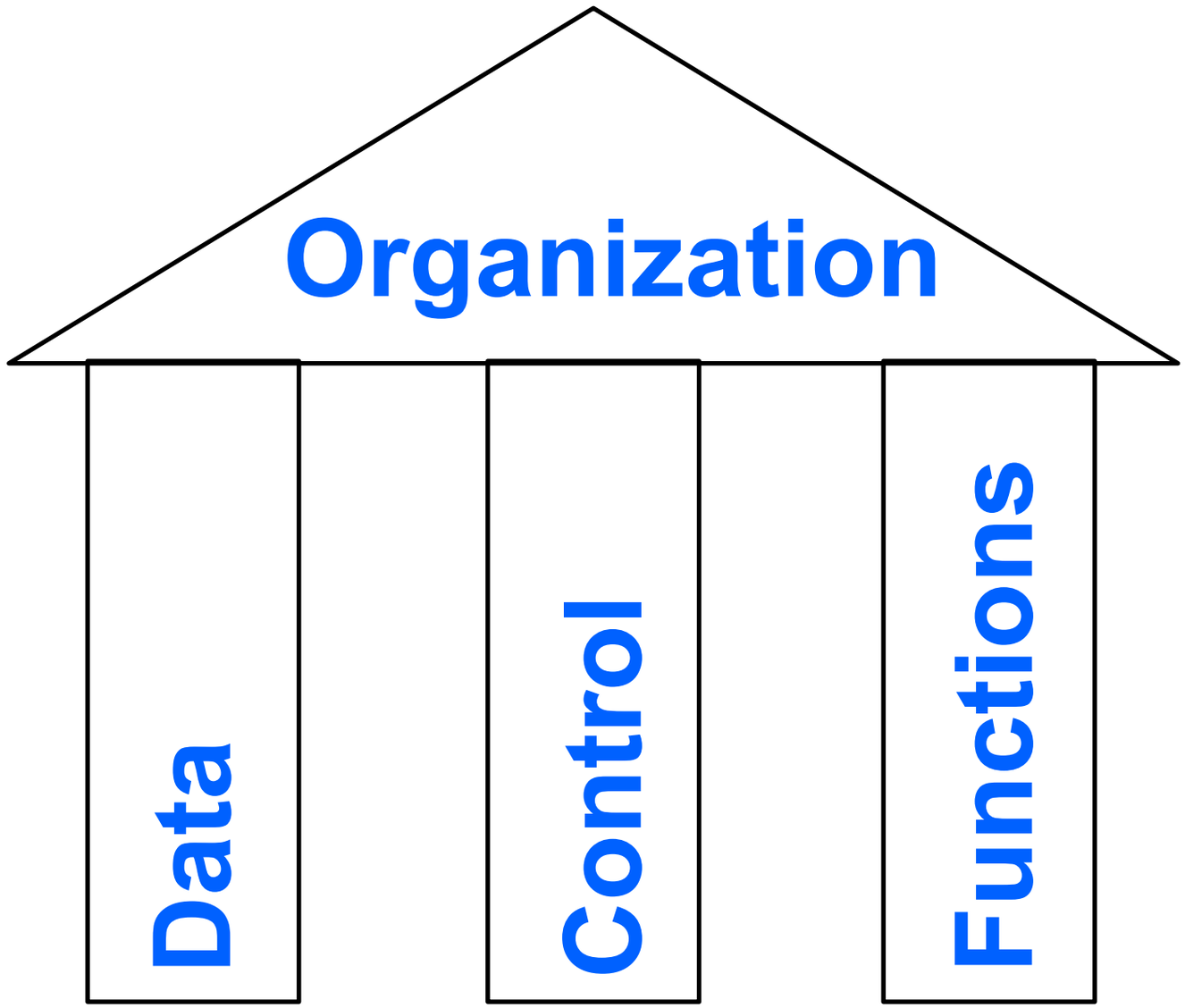
EPC origin

EPC method was originally developed by Wilhelm-August Scheer (early 1990's)

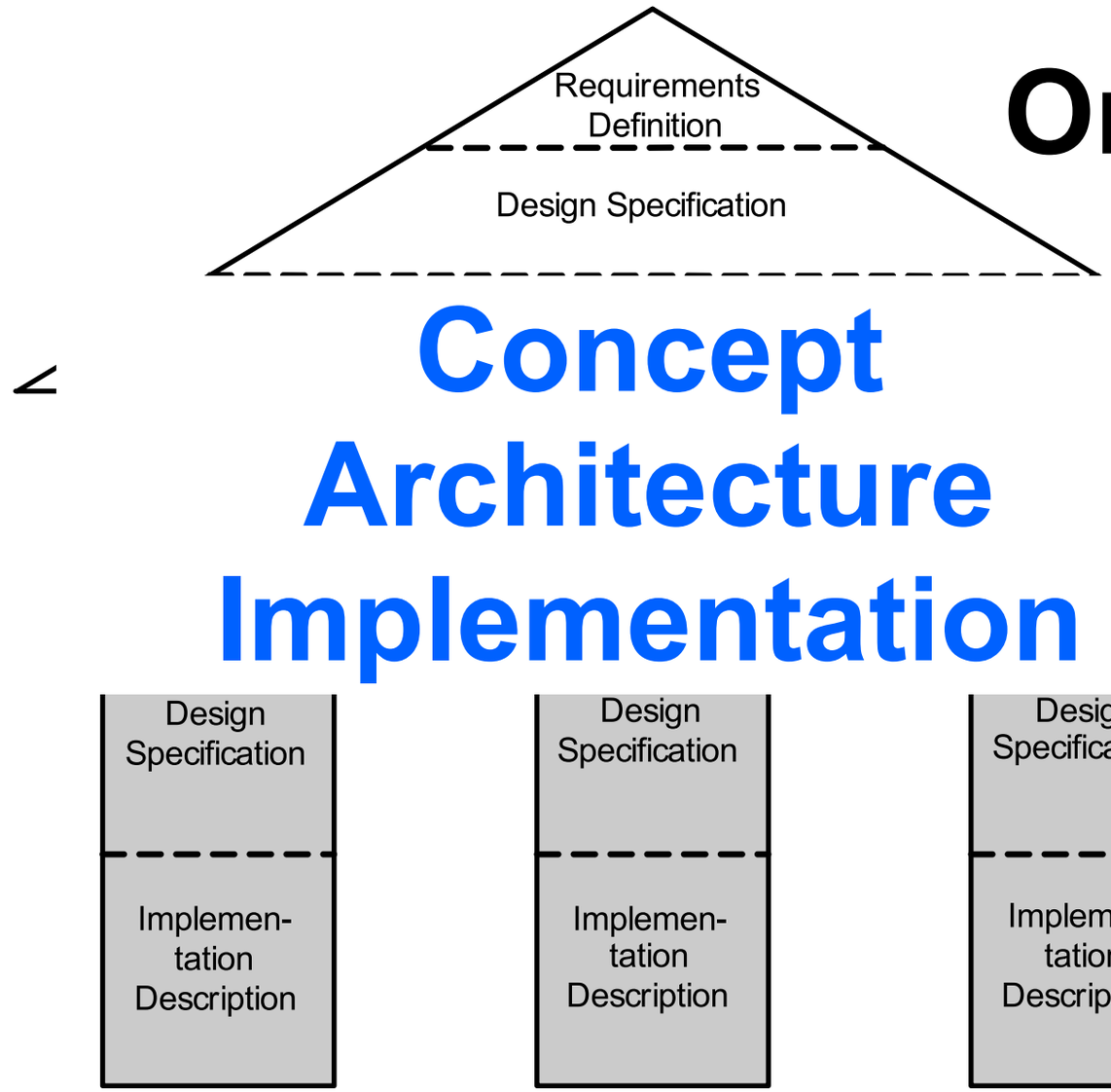


Part of a holistic modelling approach called ARIS framework (Architecture of Integrated Information Systems)

ARIS house (1999): three pillars and a roof...



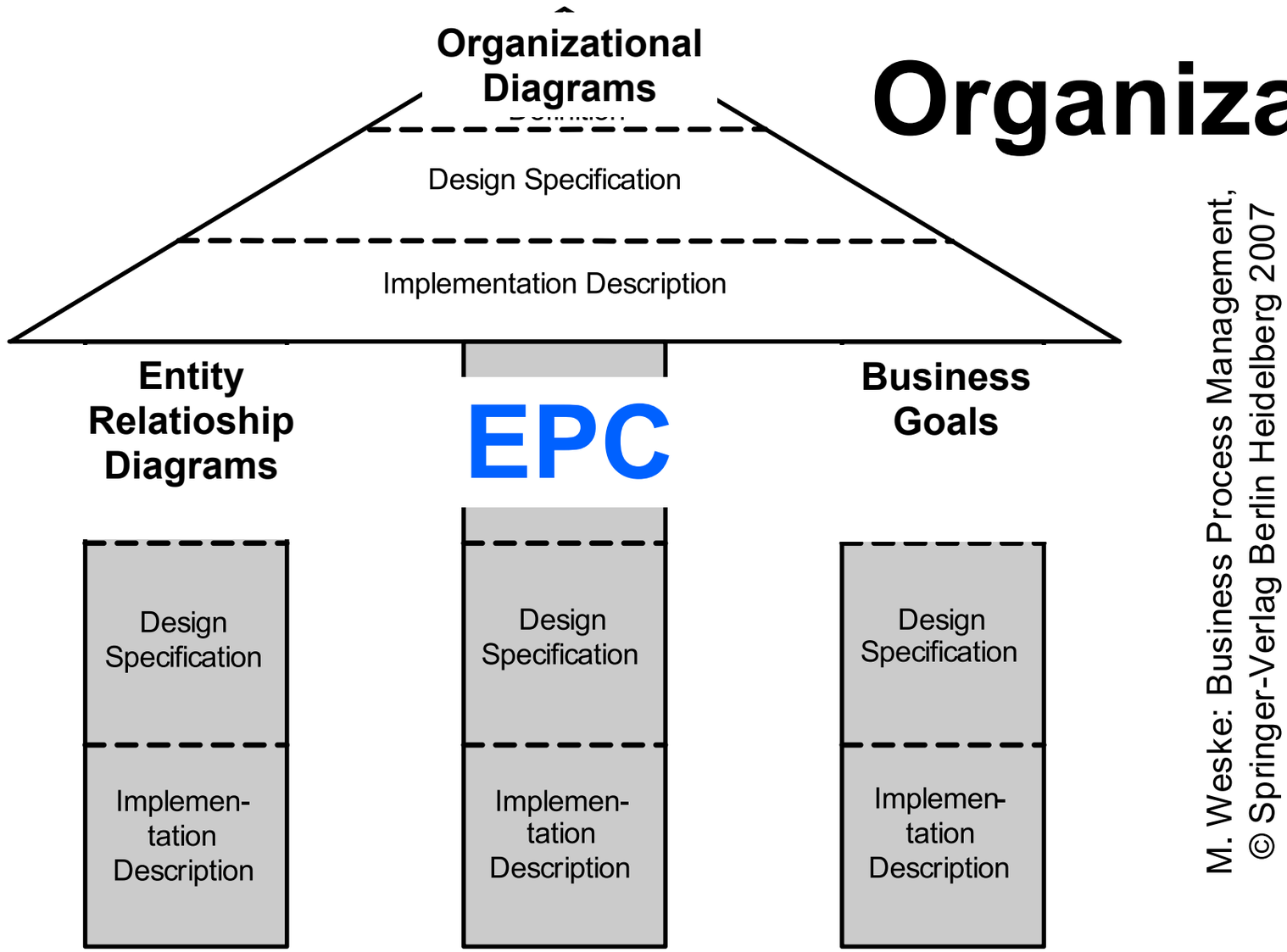
...and three levels of abstraction each



Organization

Data **Control** **Functions**

...and three levels of abstraction each



EPC informally

An EPC is an “ordered” graph
of events and functions

It provides various connectors that allow
alternative and parallel execution of processes

The flow is specified by logical operators
AND, XOR, OR

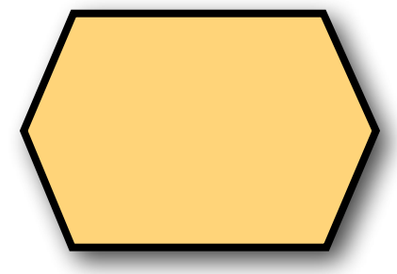
Simple, easy-to-understand notation

EPC ingredients: Event

Any EPC diagram must
start with **event(s)**
and end with **event(s)**

Passive elements used to describe
under which circumstances a process (or a function) works
or which state a process (or a function) results in
(like pre- / post-conditions)

Graphical representation: hexagons



EPC ingredients: Function

Any EPC diagram may involve
several **functions**

Active elements used to describe
the tasks or activities of a business process

Functions can be refined to other EPC

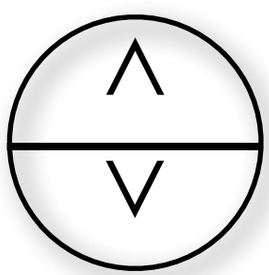
Graphical representation:
rounded rectangles

EPC ingredients: Logical connectors

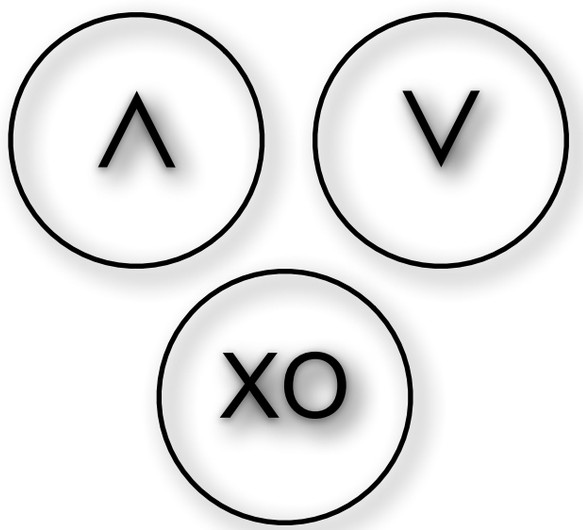
Any EPC diagram may involve
several **connectors**

Elements used to describe
the logical relationships between elements in the diagram

Branch, merge, fork, join



Graphical representation:
circles (or also octagons)



EPC ingredients: Control flow

Any EPC diagram may involve several **control flow connections**

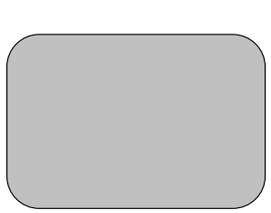
Control flow is used to connect events with functions and connectors by expressing causal dependencies

Graphical representation: dashed arrows

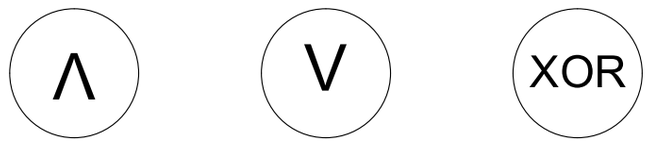
EPC ingredients at glance

Event

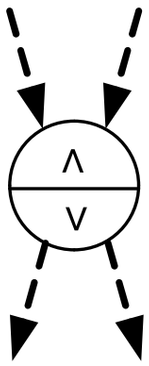
Function



Connectors



Combined Connectors (sample)



Control Flow

EPC ingredients: Diagrams

EPC elements can be combined in a fairly free manner (possibly including cycles)

There must be at least one start event and one end event
Events have at most one incoming and one outgoing arc
Events have at least one incident arc

Functions have exactly one incoming and one outgoing arc

The graph is weakly connected (no isolated nodes)

Connectors have either one incoming arc and multiple outgoing arcs or viceversa (multiple incoming arcs and one outgoing arc)

EPC ingredients: Diagrams

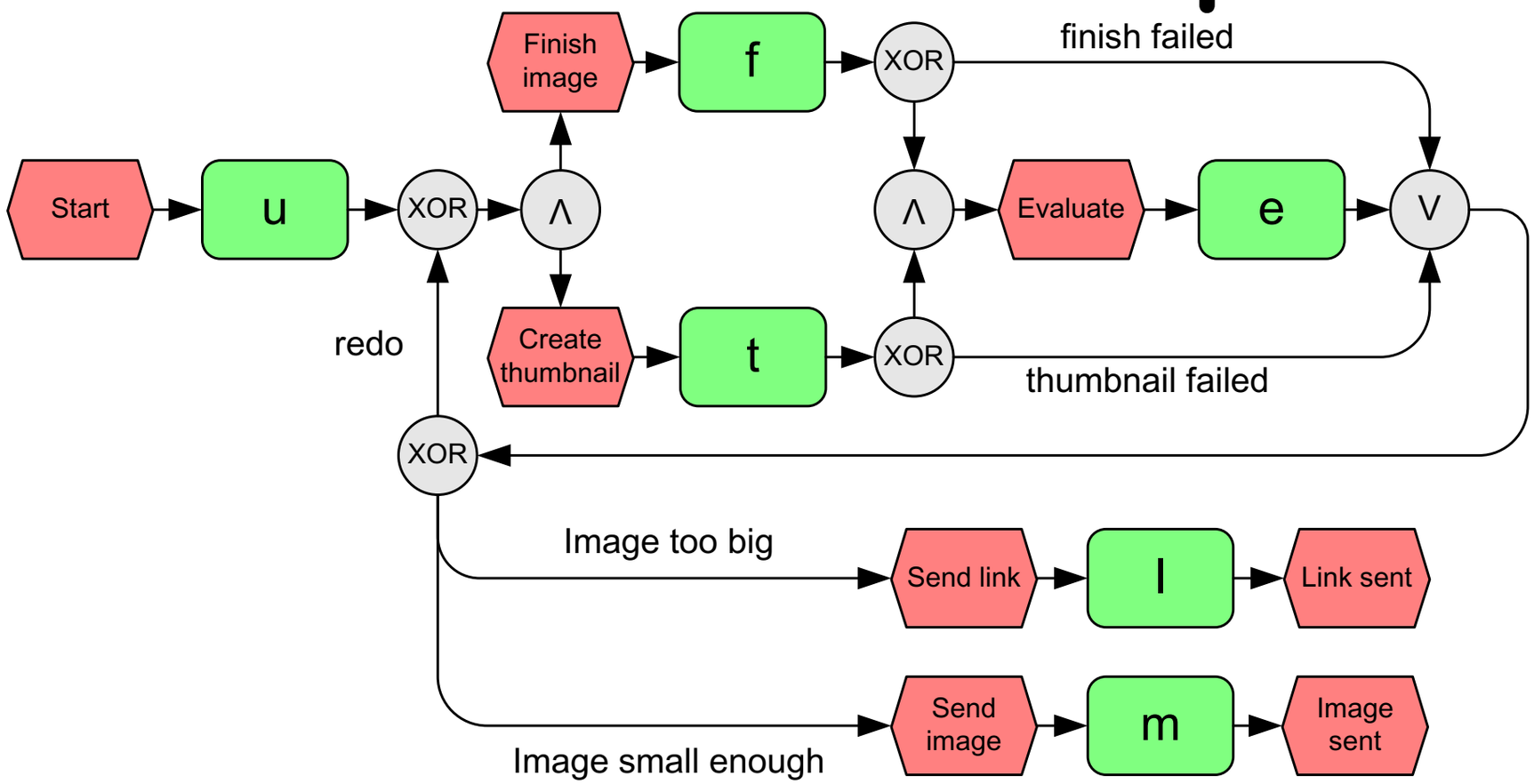
Other constraints are sometimes imposed

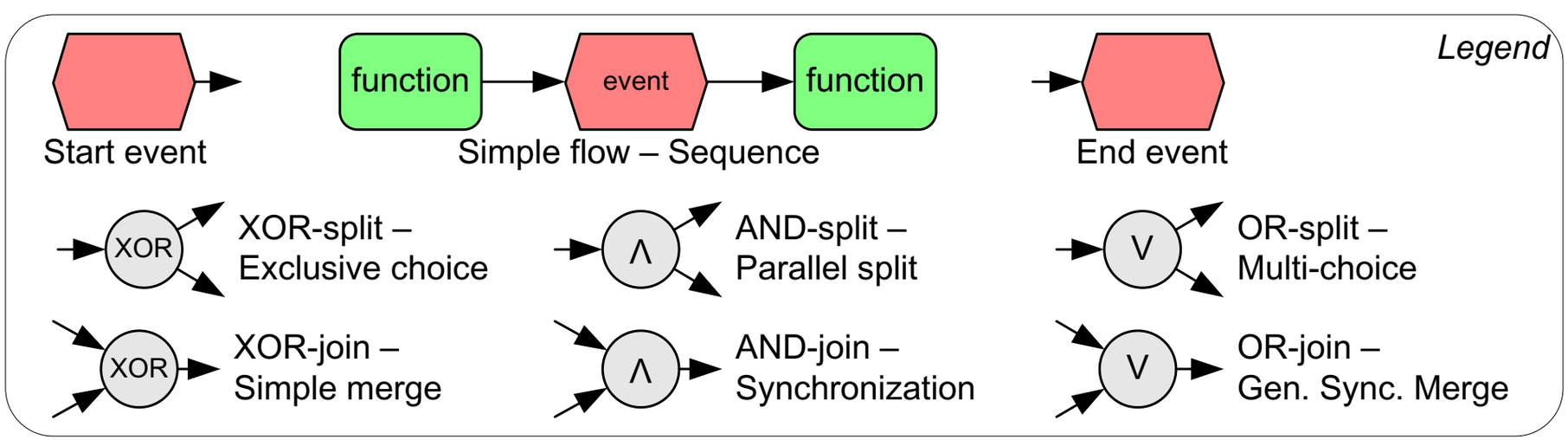
There is no arc between two events
There is no arc between two functions

Unique start / end event

No event is followed by a decision node
(i.e. (X)OR-split)

EPC an example





Other stuff

Other decorations / annotations for functions:

Information, material, resource object: represents objects in the real world that can be input data or output data for a function
(rectangles linked to function boxes)

Organization unit: determines the person or organization responsible for a specific function
(ellipses with a vertical line)

Supporting system: technical support
(rectangles with vertical lines on its sides)

EPC intuitive semantics

A process starts when some initial event(s) occurs

The activities are executed according to the constraints in the diagram

When the process is finished, only final events have not been dealt with

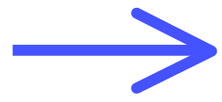
If this is always the case, then the EPC is “correct”

Folder-passing semantics

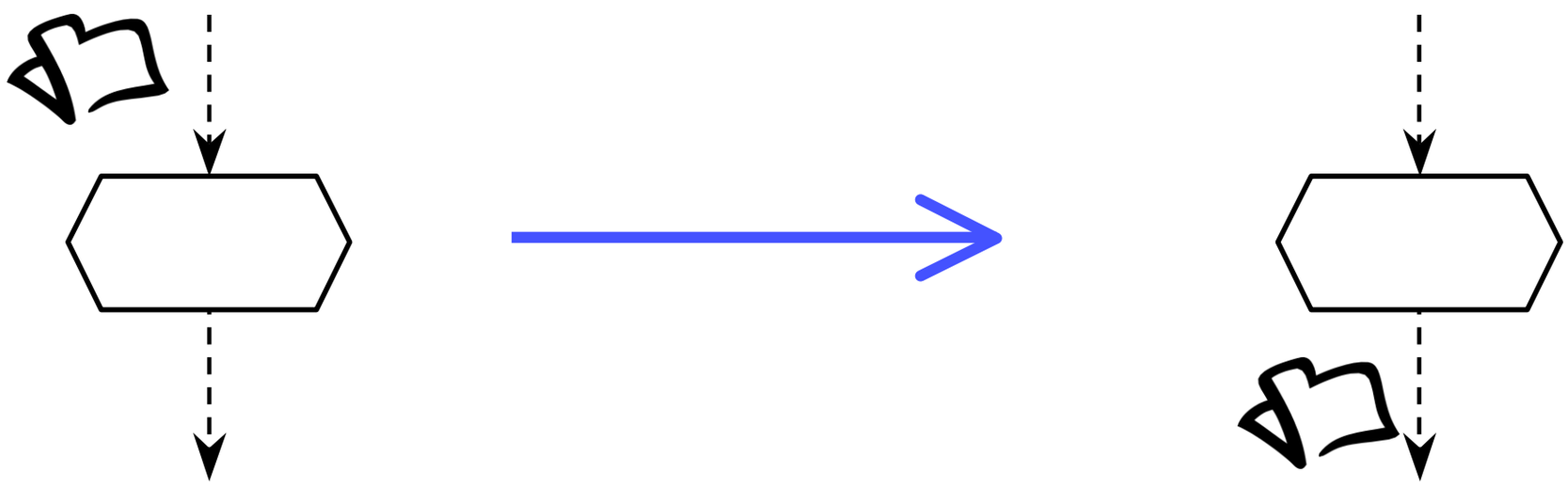
Semantics

- State: Process folders

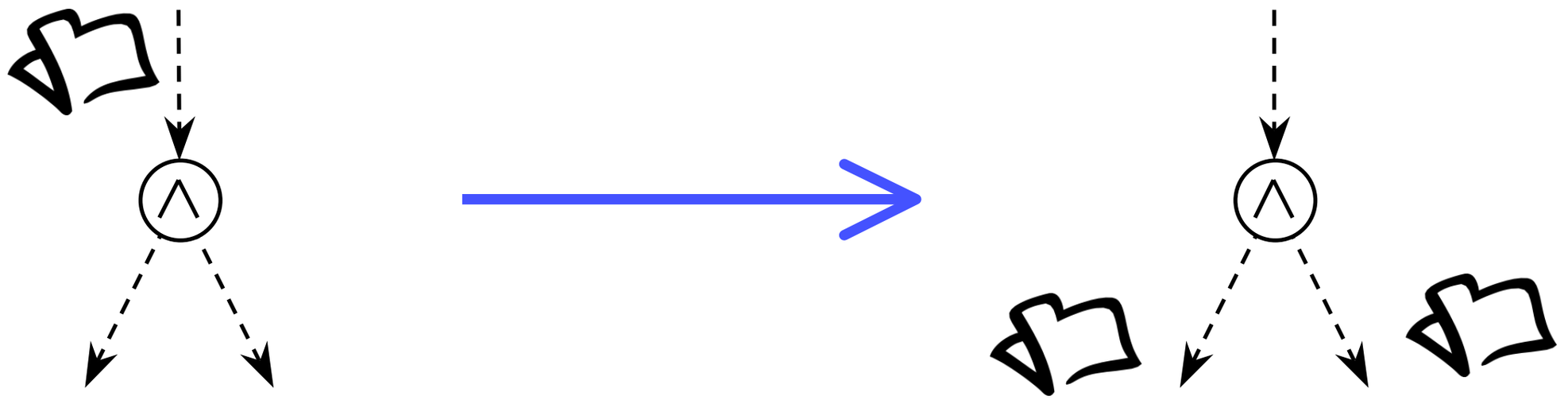
- Transition relation:
  Propagation of process folders



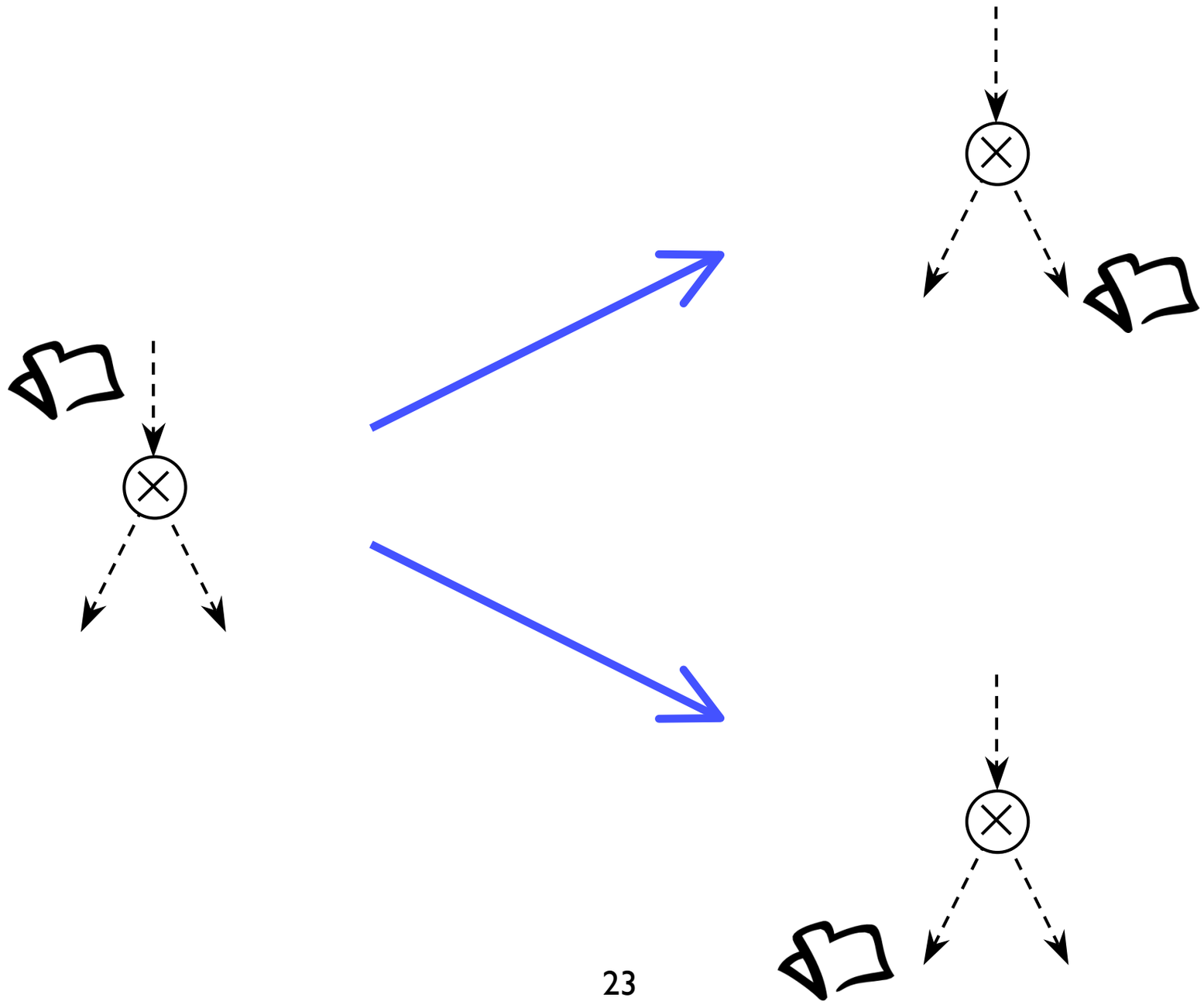
Folder-passing semantics: events



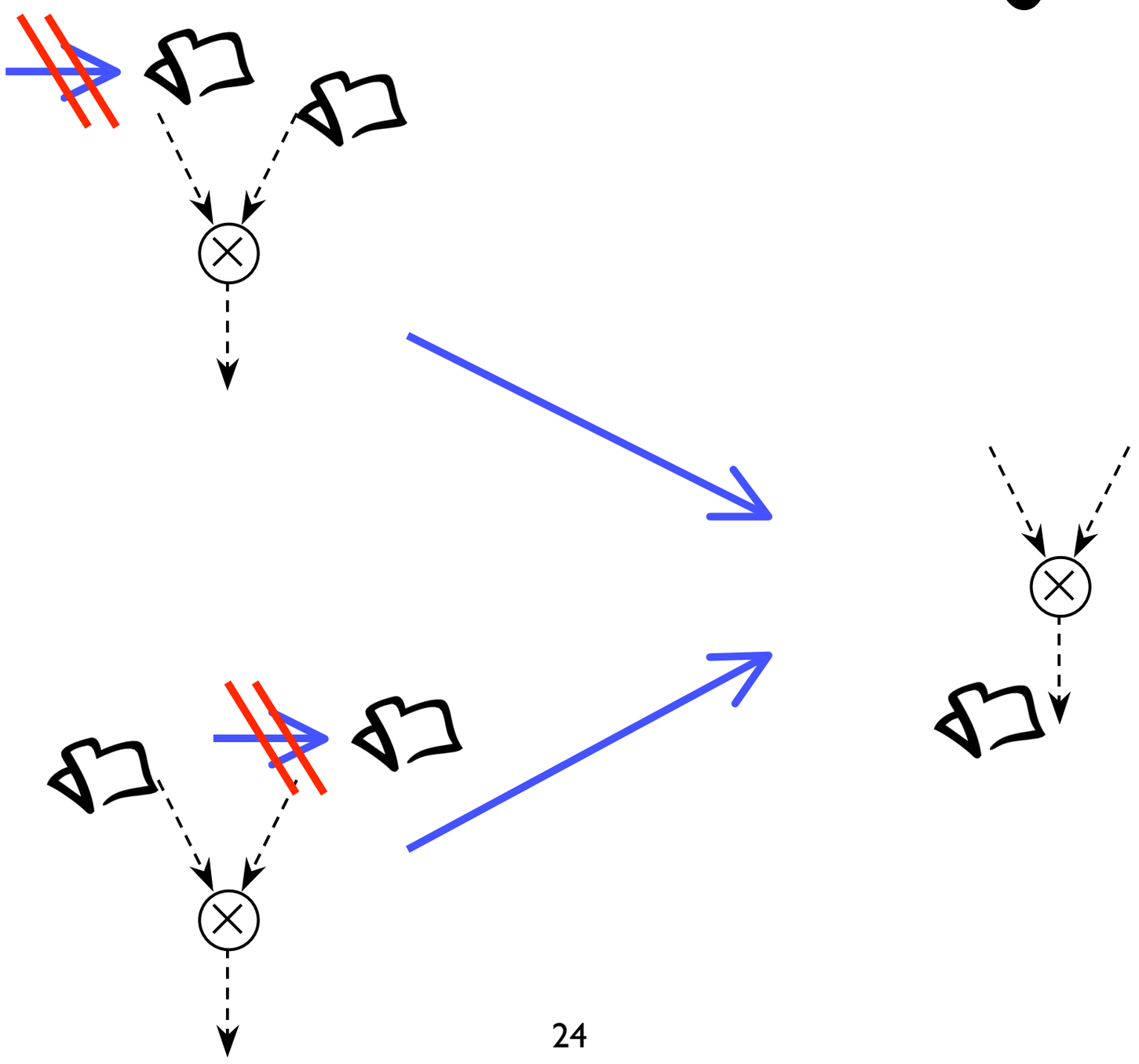
Folder-passing semantics: AND-split



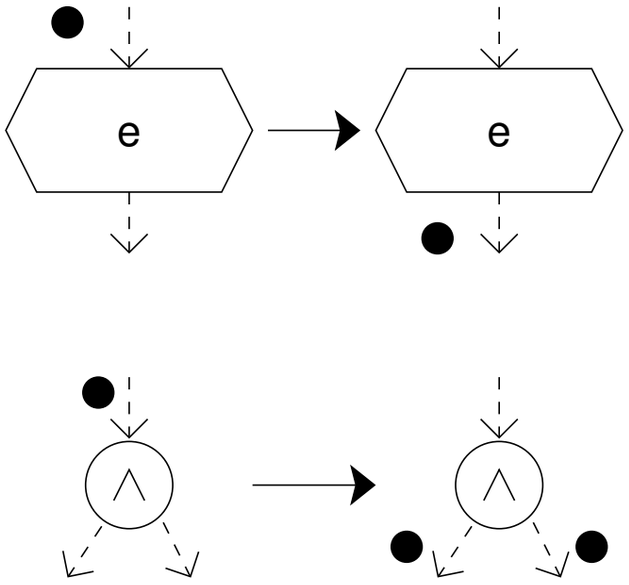
Folder-passing semantics: XOR-split

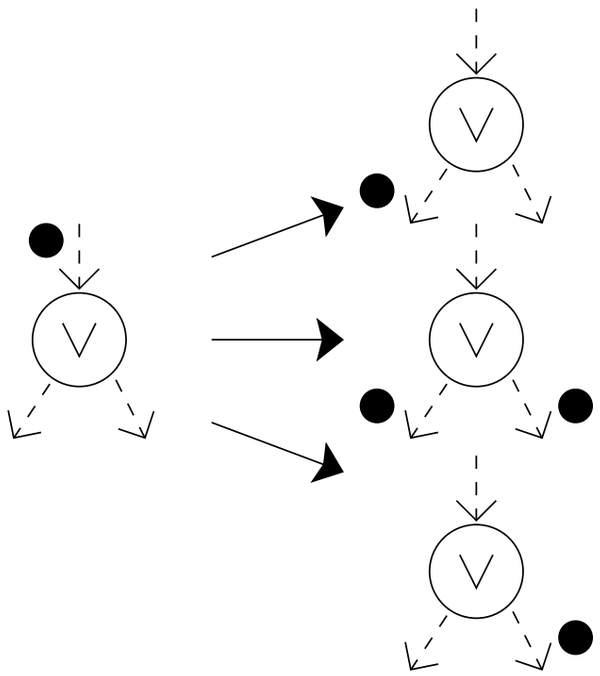


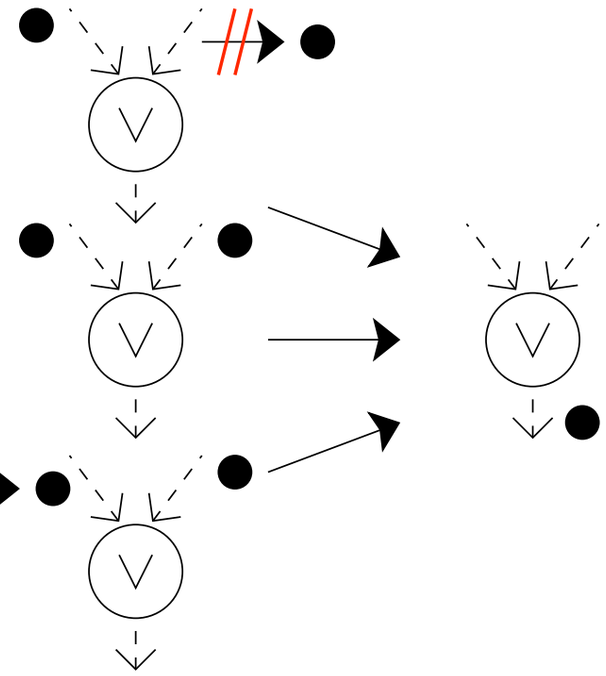
Folder-passing semantics: XOR-join

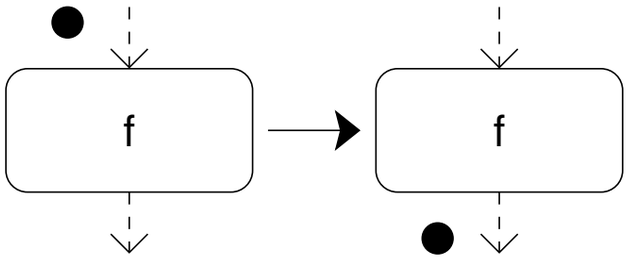


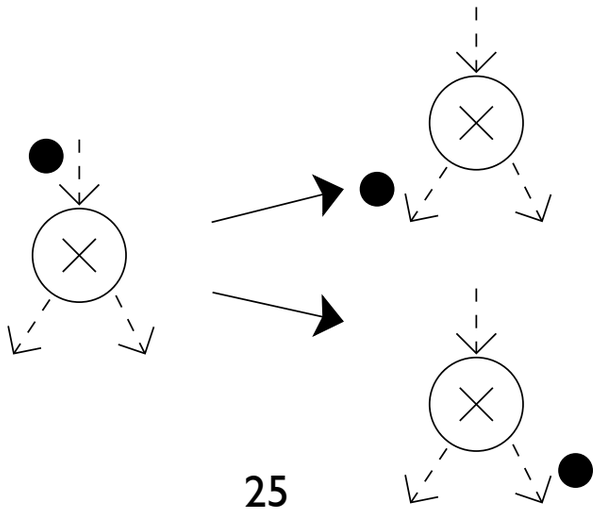
Folder-semantics in one slide

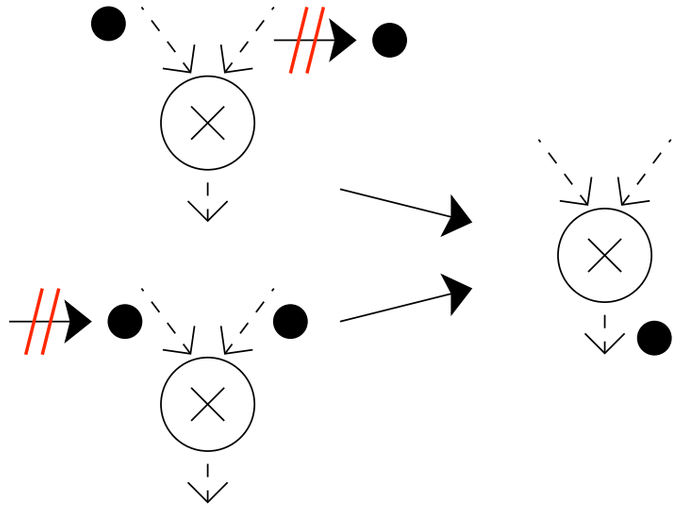




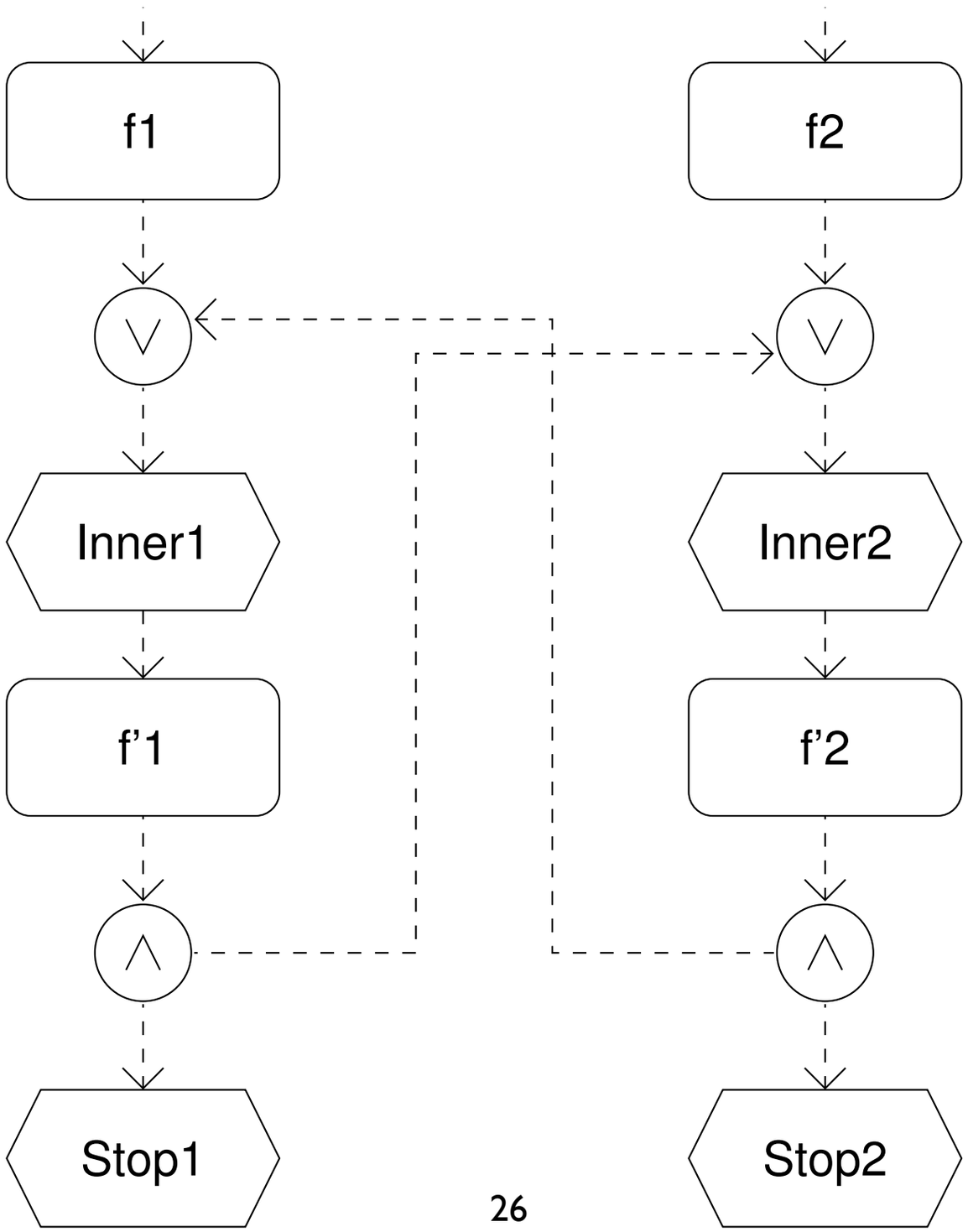




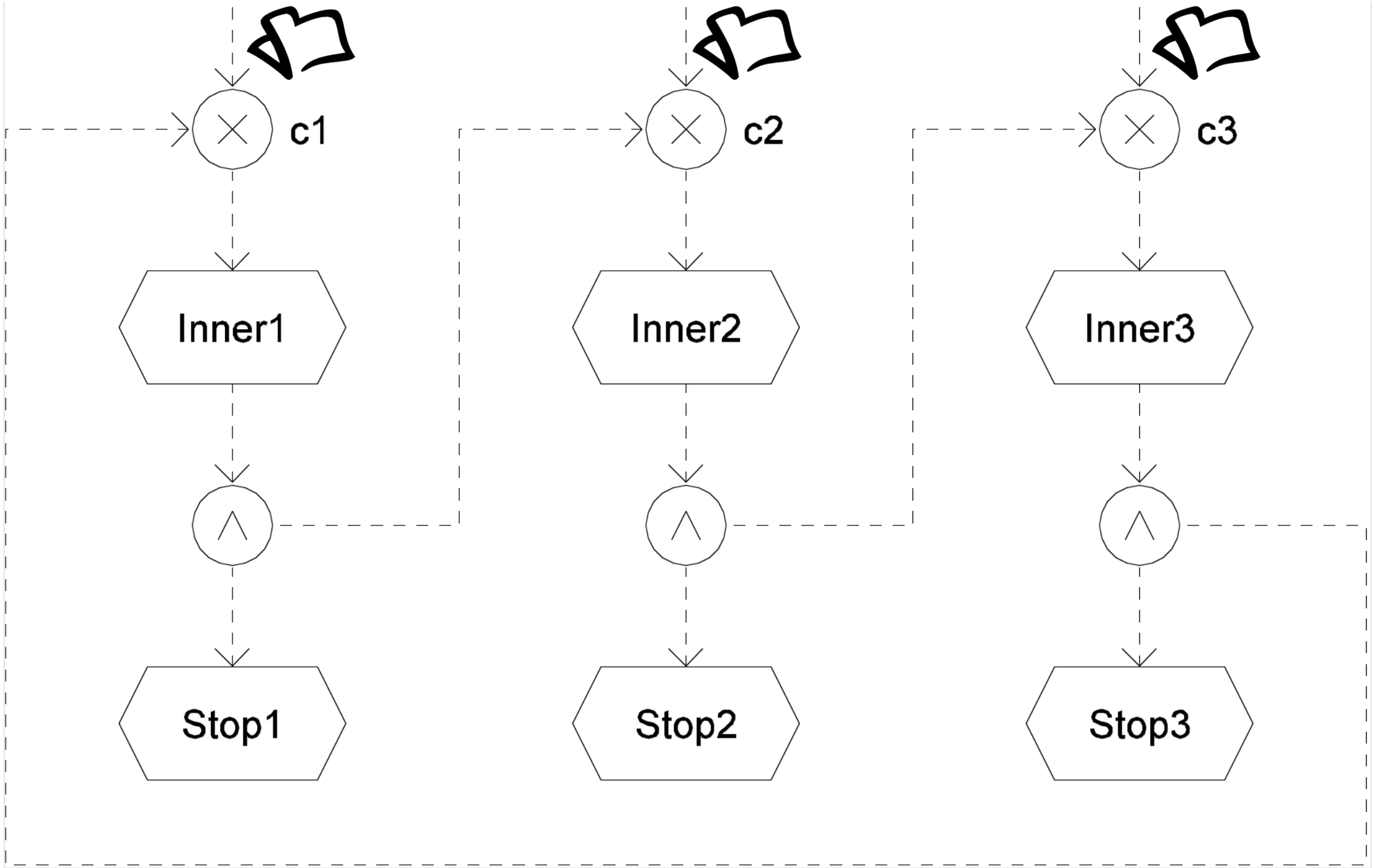




A vicious circle



A vicious circle?



EPC semantics?

Little unanimity around the EPC semantics

Roughly described (verbal form) in the original publication by Scheer (1992)

Later, several attempts to define formal semantics (in many cases they end up attributing different meanings to the same EPC)

Discrepancies typically stem from the interpretation of (X)OR connectors (in particular, join case)
Other issues: unclear start, join/split balancing, alternation of events and functions

Problem with start events

A start event is an event with no incoming arc

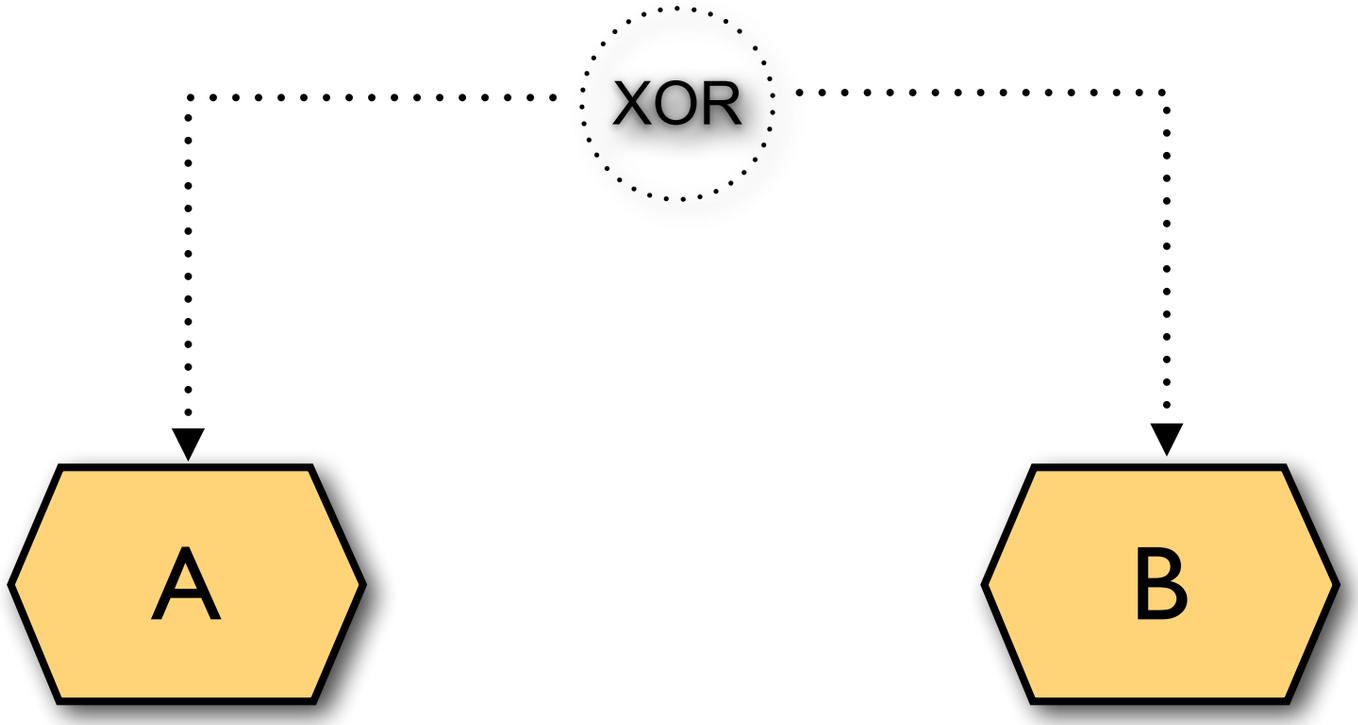
A start event invokes a new execution of the process template

What if multiple start events occur?

Start events are mutually exclusive
(as if they were preceded by an implicit XOR split)

Problem with start events

hypothetical / implicit split



Problem with corresponding splits

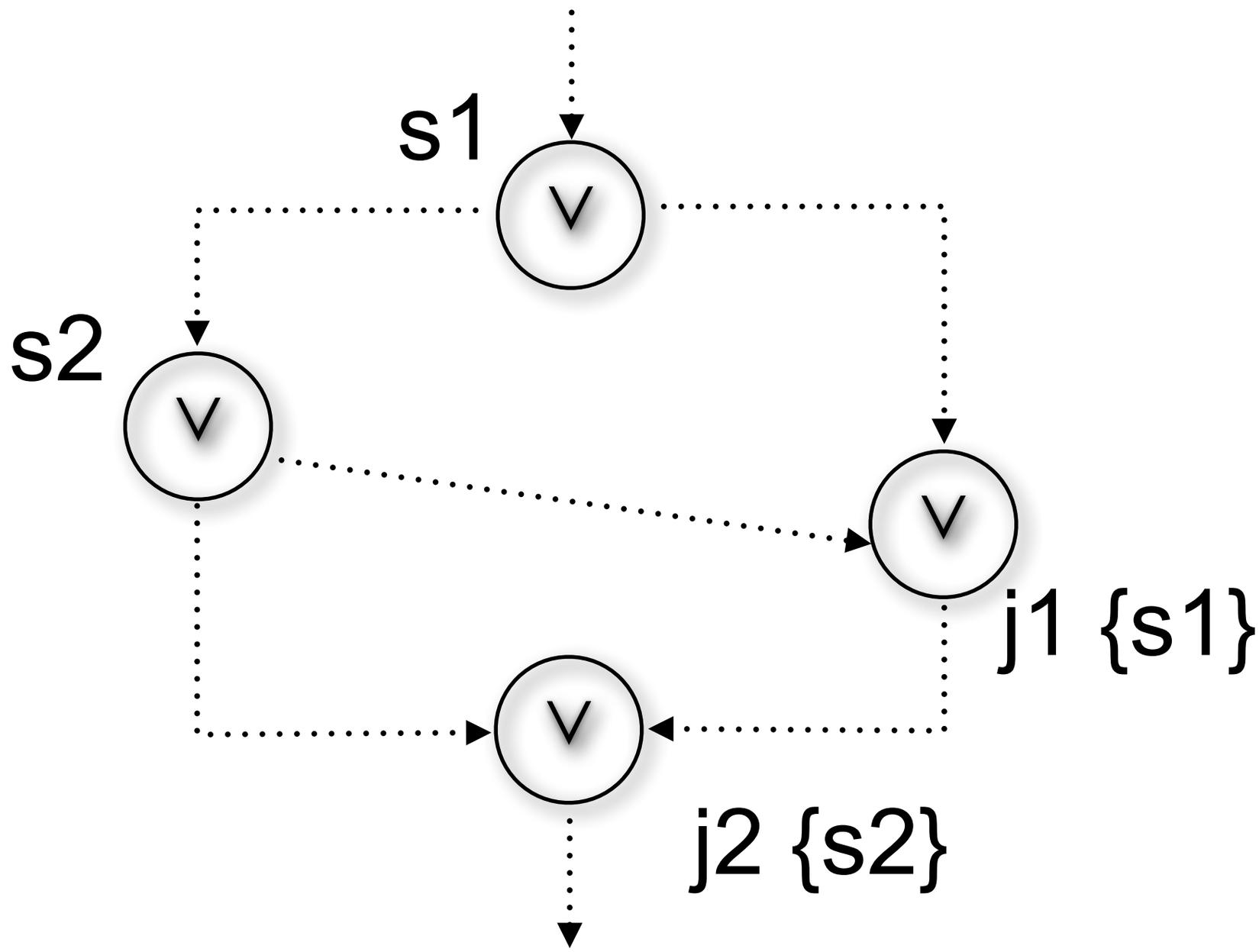
The semantics of a join often depends on whether or not it has a **corresponding** split

In theory, every join has at least one candidate split
(i.e. a split for which there is a path
from either output to the input of the join)
proof: we trace backward the paths leading to the join from start events; if the start events coincide there must be a split node in the path; if start events differ, the candidate split is the implicit XOR

But there can be more candidates to corresponding split
and they can have different type than the join
(candidates of the same type of the join are called **matching** split)

Some suggested to have a flag that denotes the corresponding split

Problem with corresponding splits



Problem with OR join

If an OR join has a matching split, the semantics is usually:
“wait for the completion of all paths activated by the matching split”

If there is no matching split, some ambiguities can arise:

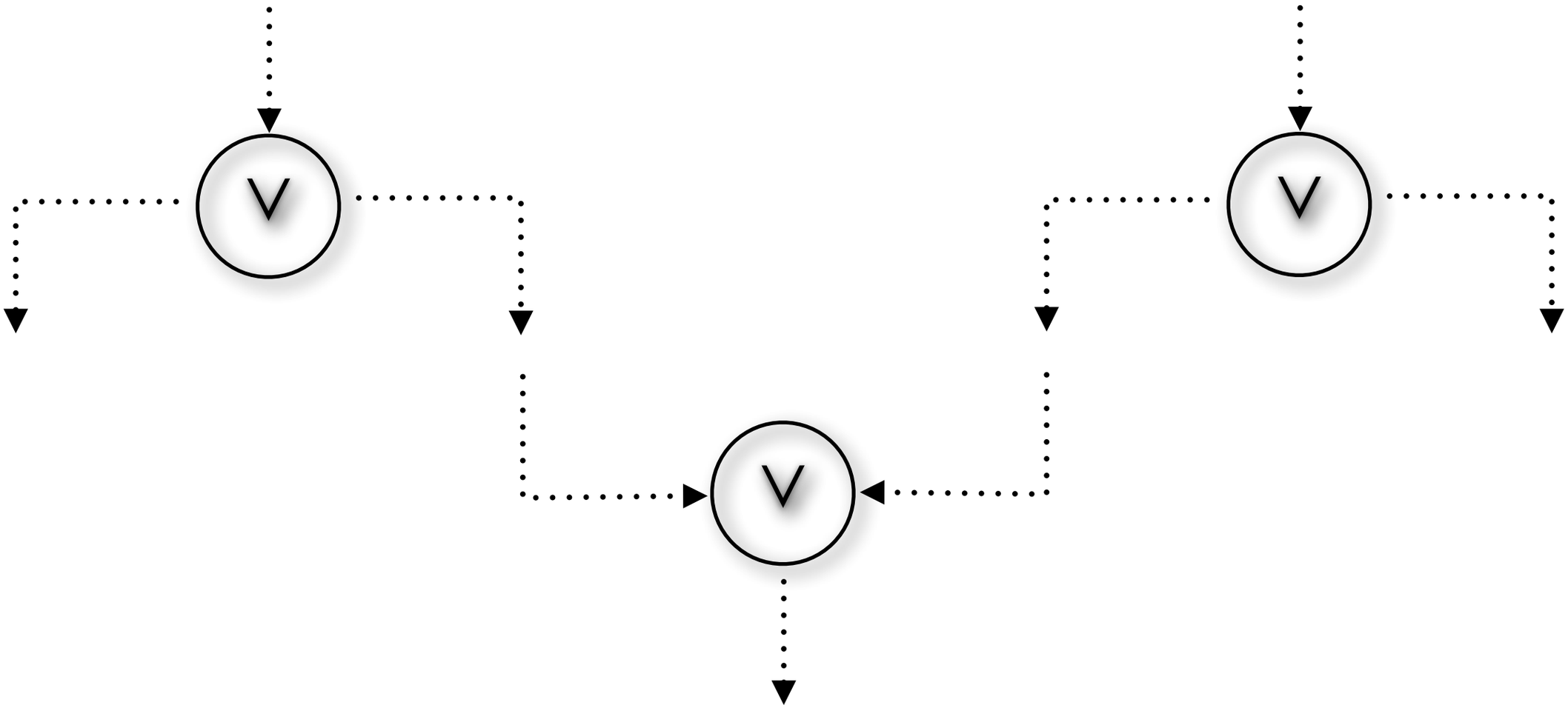
wait-for-all: wait for the completion of all activated paths
(default semantics, because it coincides with that of a matched OR)

first-come: wait only for the path that is completed first
and ignore the second

every-time: trigger the outgoing path on each completion
(the outgoing path can be activated multiple times)

Some suggested to have different (trapezoid) symbols or
suitable flags to distinguish the above cases and allow them all

Problem with OR join



Problem with XOR join

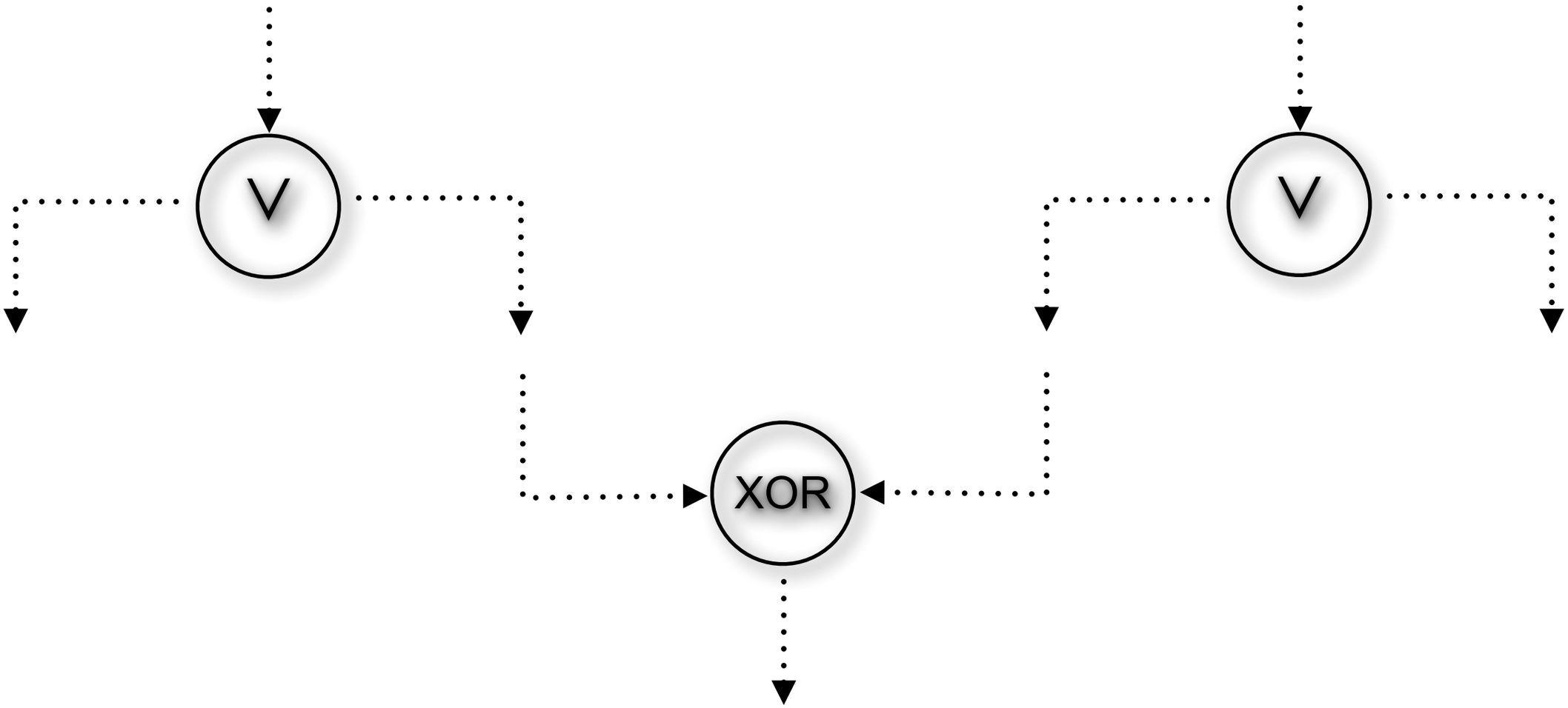
Similar considerations hold for the XOR join

If a XOR join has a matching split, the semantics is intuitive:
“it blocks if both paths are activated and it is triggered by the completion of a singe activated path”

If there is no matching split:
all feasible interpretations that do not involve blocking are already covered by the OR (wait-for-all, first-come, every-time)
and **contradict the exclusivity** of the XOR
(a token from one path can be accepted only if we make sure that no second token will arrive via the other path)

Some suggest to just forbid the use of XOR in the unmatched case
(the implicit start split is allowed as a valid match)

Problem with XOR join



Problem with alternation

Empirical studies have shown that middle and upper management people consider strict alternation between events and functions as too restrictive:
they find it hard to identify the necessary events at the abstract level of process description they are working at

It is safe to drop this requirement,
as dummy events might always be added later, if needed

Translation of EPC to Petri nets

Idea

We transform EPC diagrams to Workflow nets

We exploit the formal semantics of nets
to give unambiguous semantics to EPC diagrams

We apply the verification tools we have seen
to check if the net is sound:
the EPC diagram is sound if its net is so

A note about the transformation

We first transform each event, function and connector separately in small nets

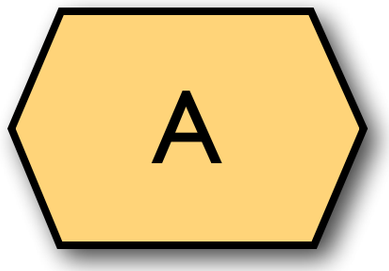
When translating the control flow arcs we may then introduce other places / transitions to preserve the bipartite structure in the net
(no arc allowed between two places,
no arc allowed between two transitions)

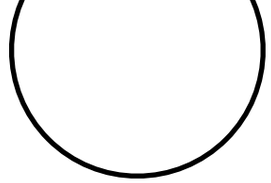
We show two translations, depending on whether the join are decorated or not

First attempt (decorated EPC)

EPC

Petri net





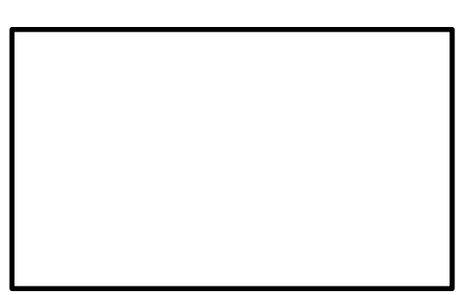
event

place

EPC

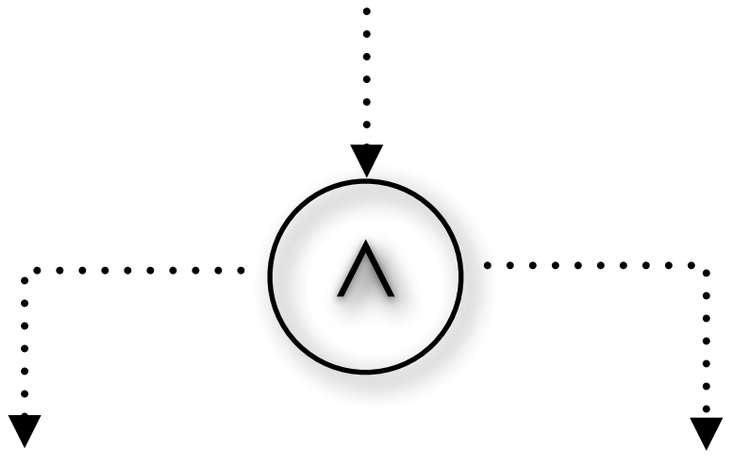
function

Petri net



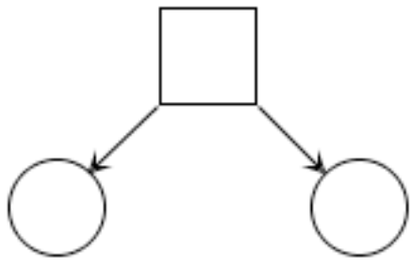
transition

EPC



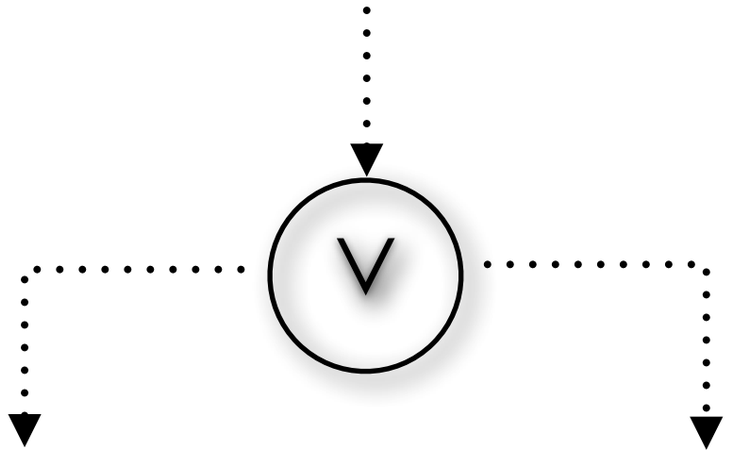
AND split

Petri net



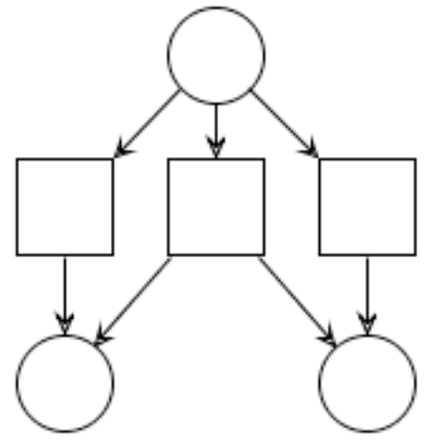
net

EPC



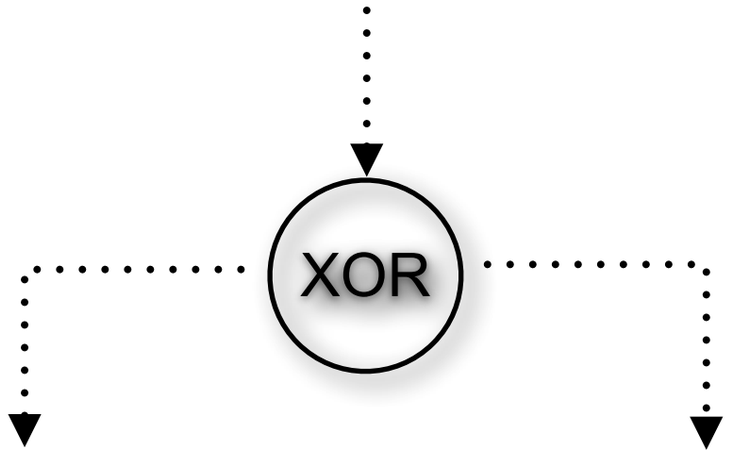
OR split

Petri net



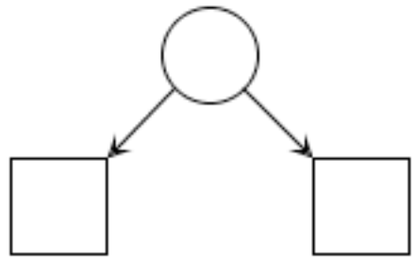
net

EPC



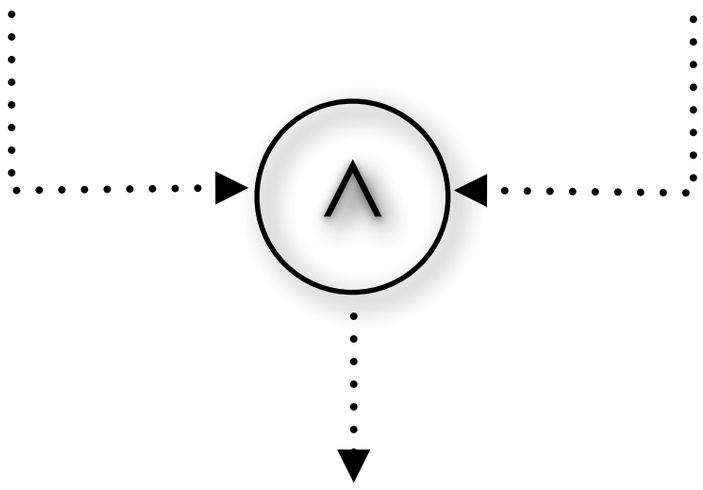
XOR split

Petri net



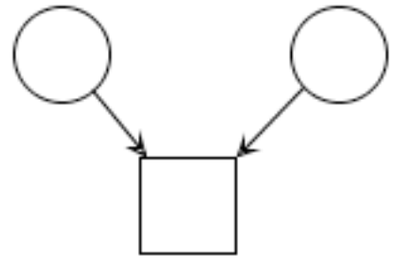
net

EPC



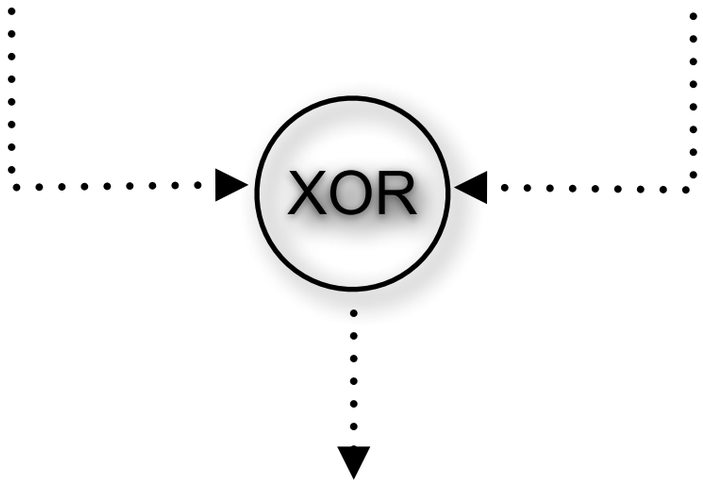
AND join

Petri net



net

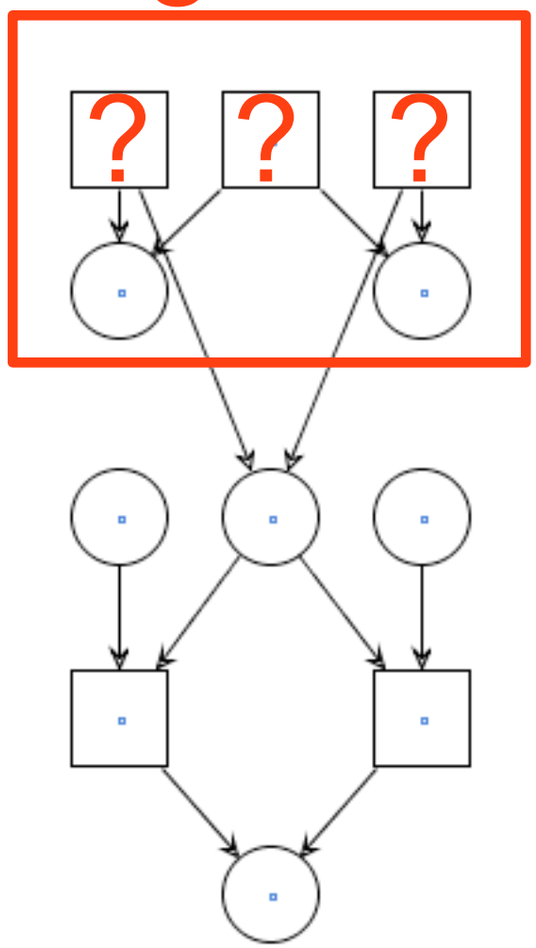
EPC



XOR join

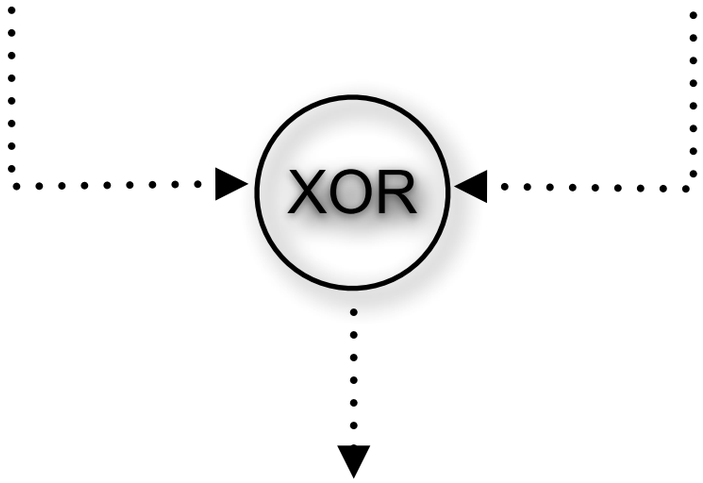
Petri net

corresponding split



net

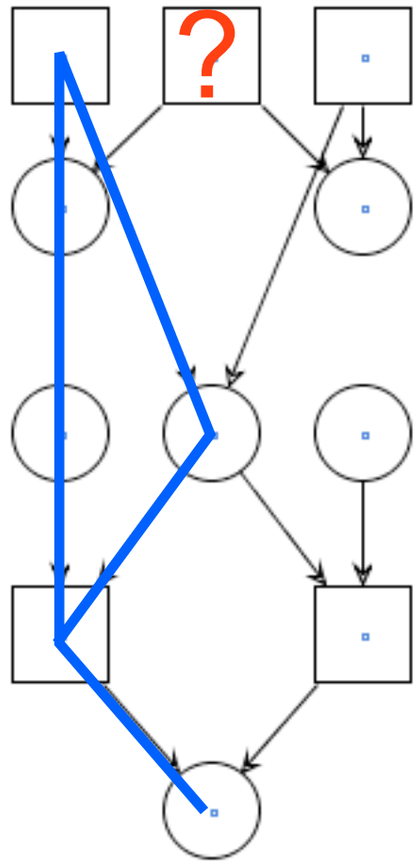
EPC



XOR join

Petri net

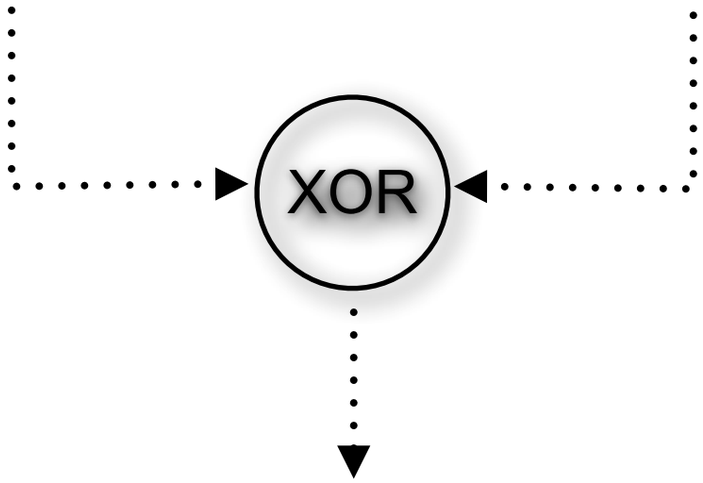
corresponding XOR/OR split



ok

net

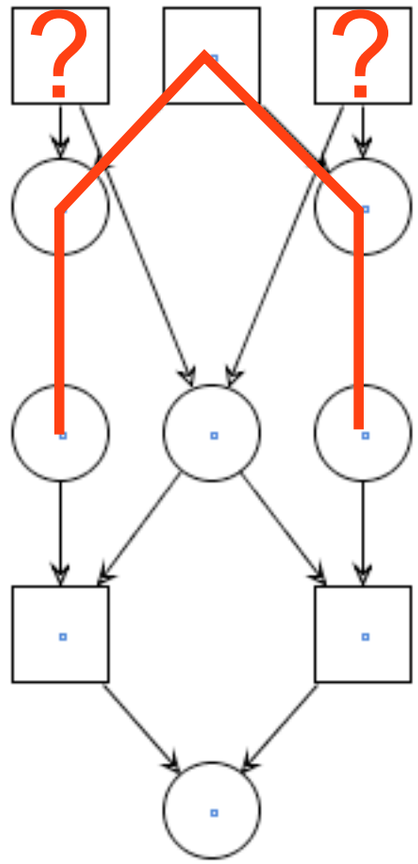
EPC



XOR join

Petri net

corresponding AND/OR split

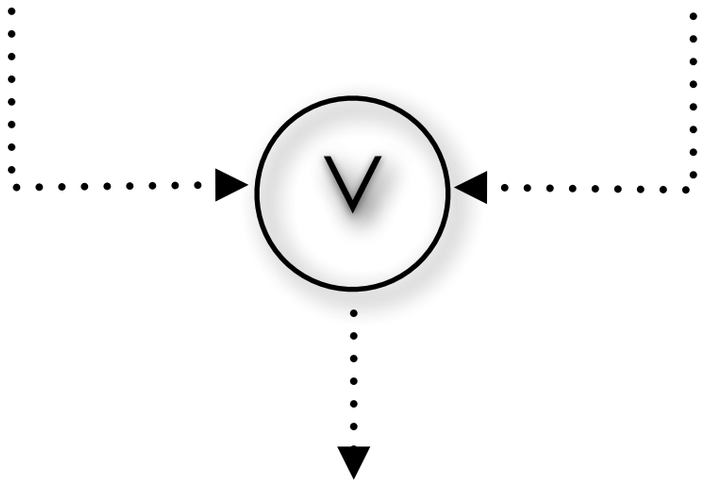


deadlock!

dangling tokens!

net

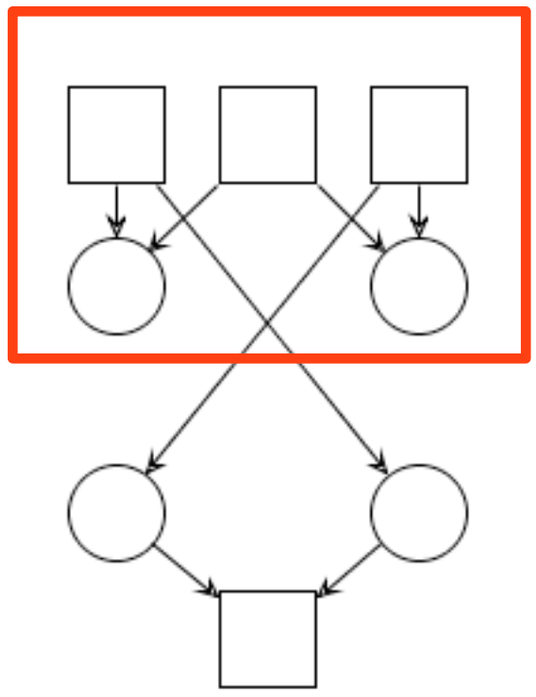
EPC



OR join
with
matched OR split

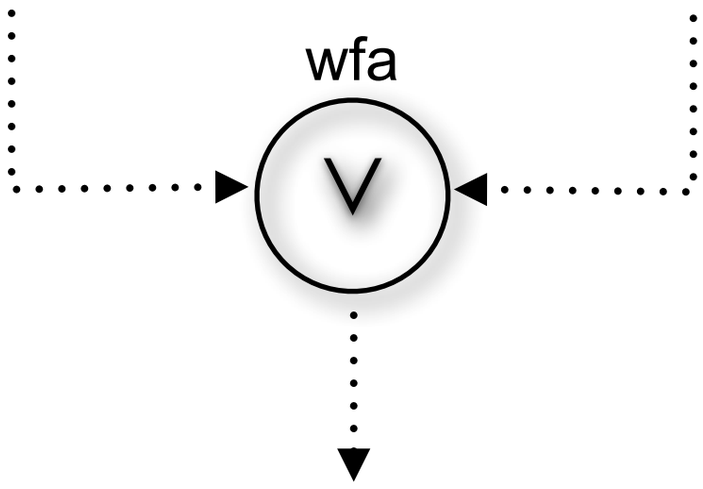
Petri net

matched
part



net

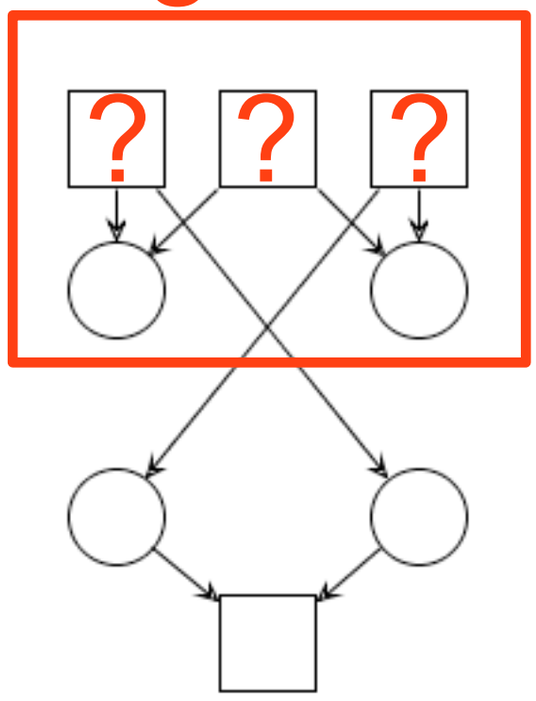
EPC



OR join
wait-for-all
(unmatched)

Petri net

corresponding
split

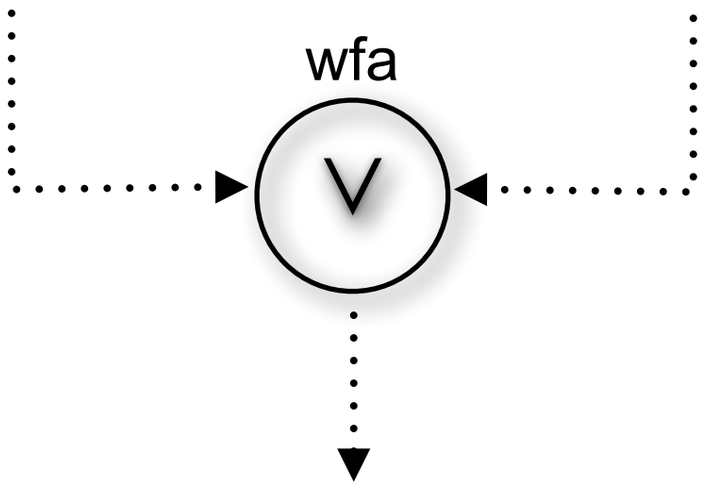


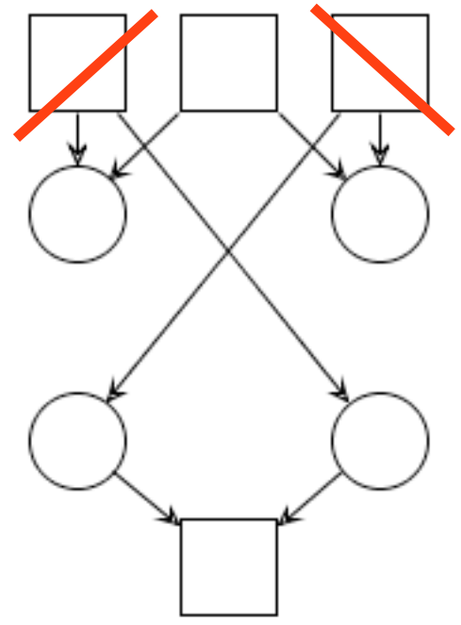
net

EPC

Petri net

corresponding
AND split

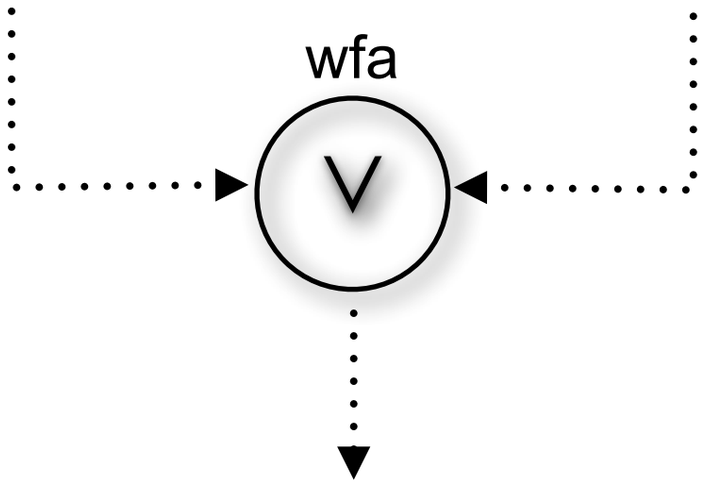




OR join
wait-for-all
(unmatched)

net

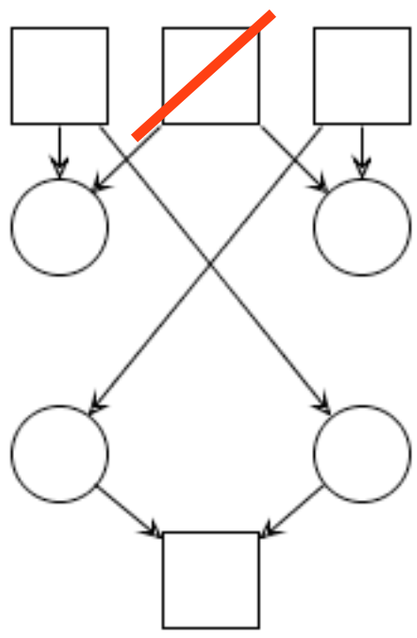
EPC



OR join
wait-for-all
(unmatched)

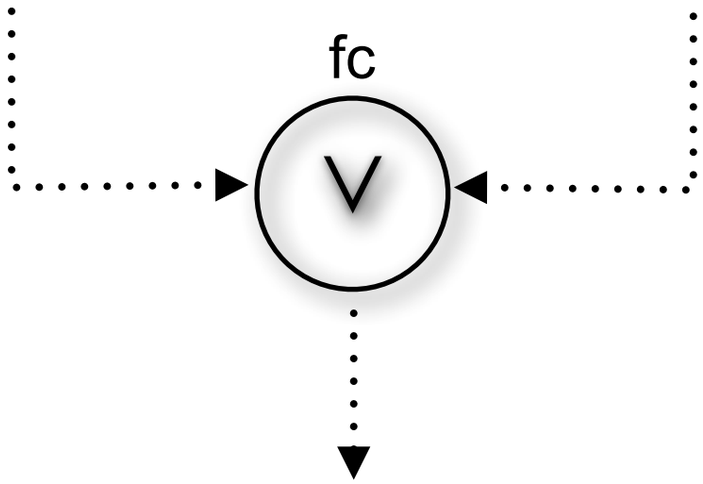
Petri net

corresponding
XOR split



net

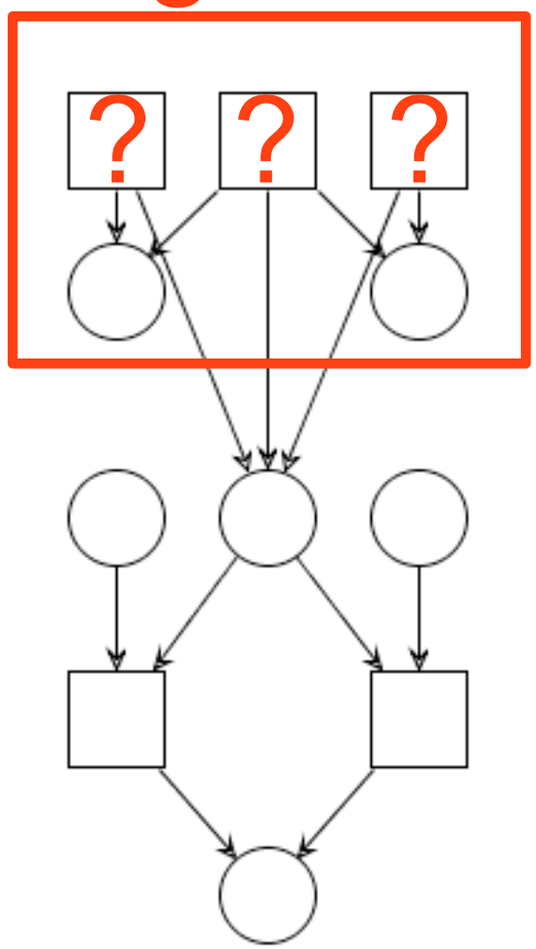
EPC



OR join
first-come
(unmatched)

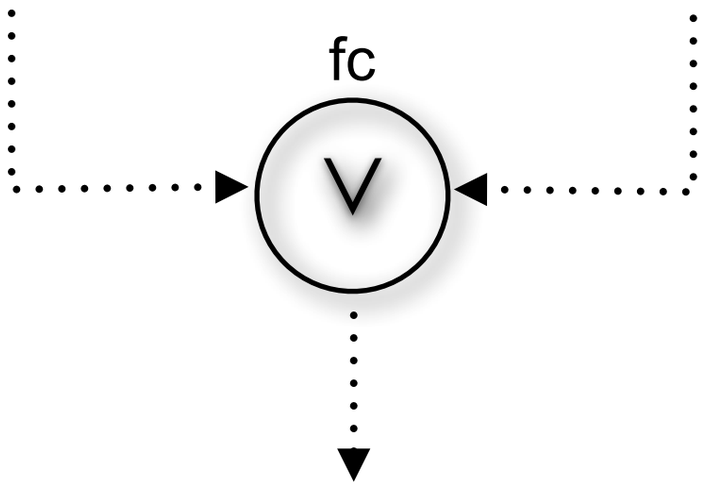
Petri net

corresponding
split



net

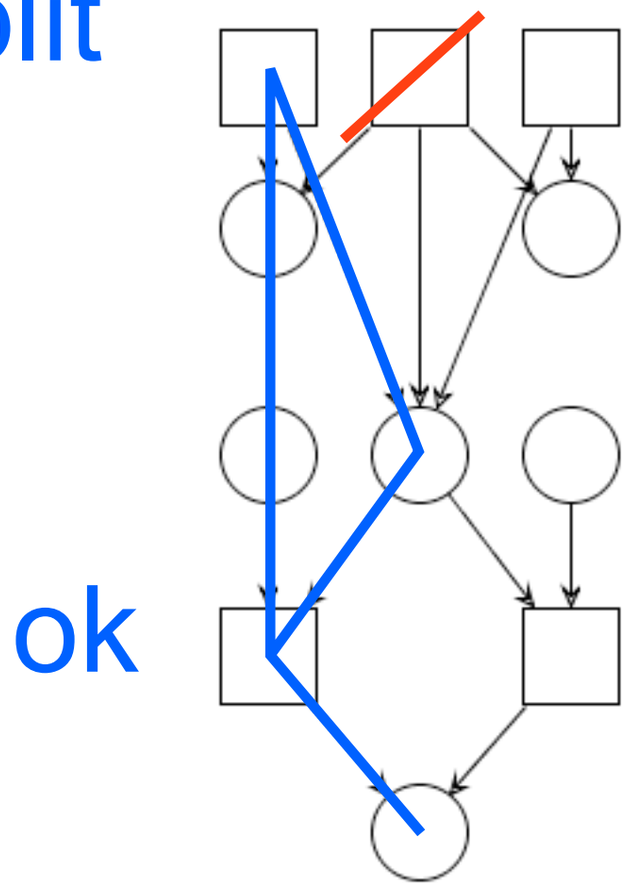
EPC



OR join
first-come
(unmatched)

Petri net

corresponding
XOR split

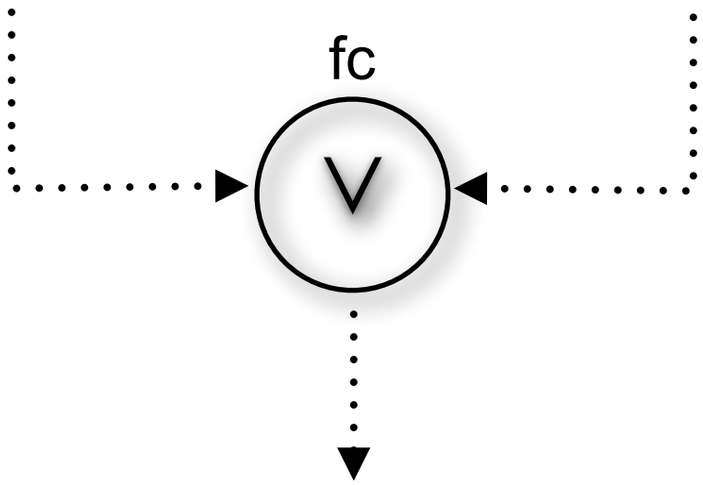


ok

net

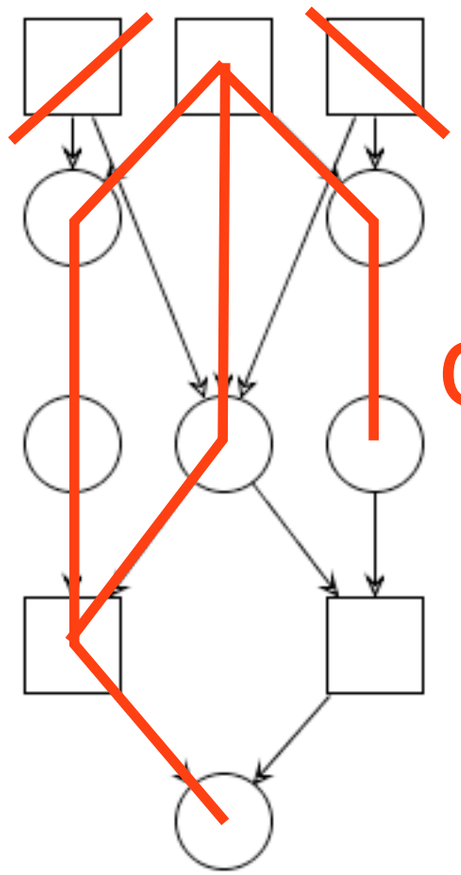
EPC

Petri net



OR join
first-come
(unmatched)

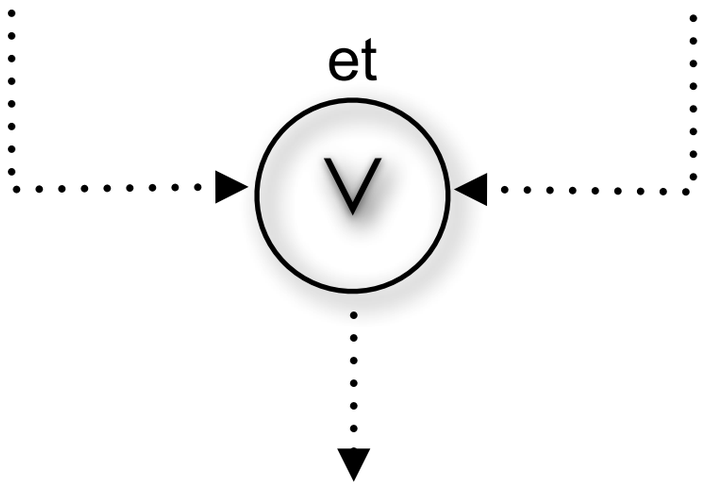
corresponding
AND split



dangling
token!

net

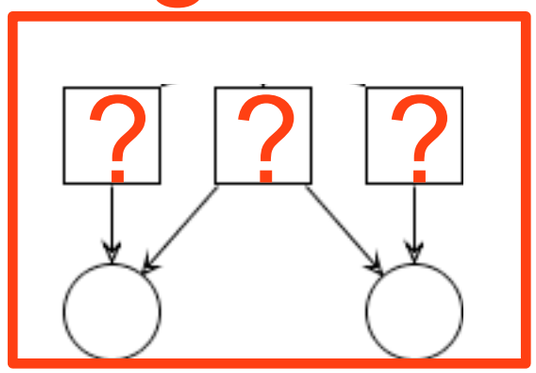
EPC

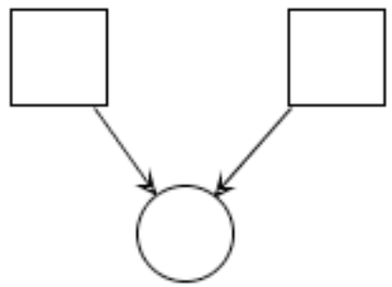


OR join
every-time
(unmatched)

Petri net

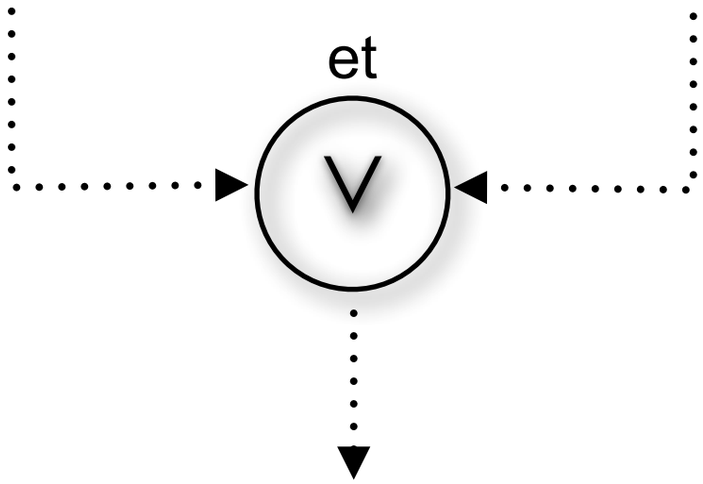
corresponding
split





net

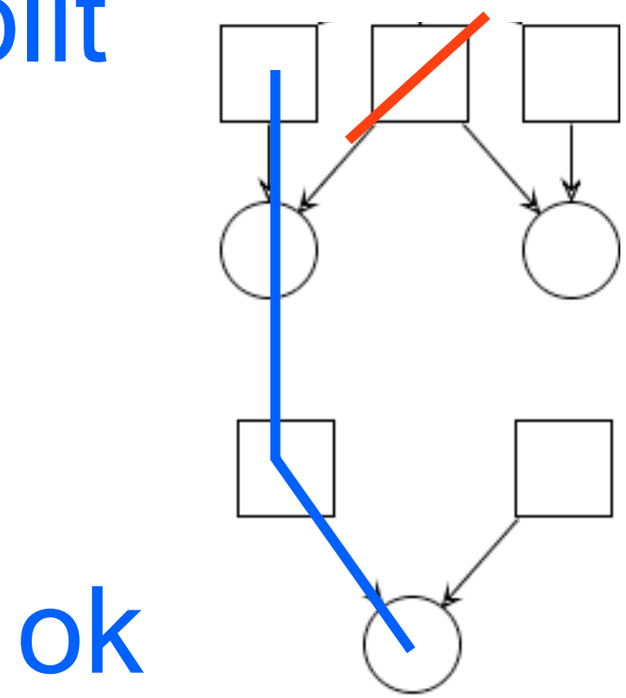
EPC



OR join
every-time
(unmatched)

Petri net

corresponding
XOR split

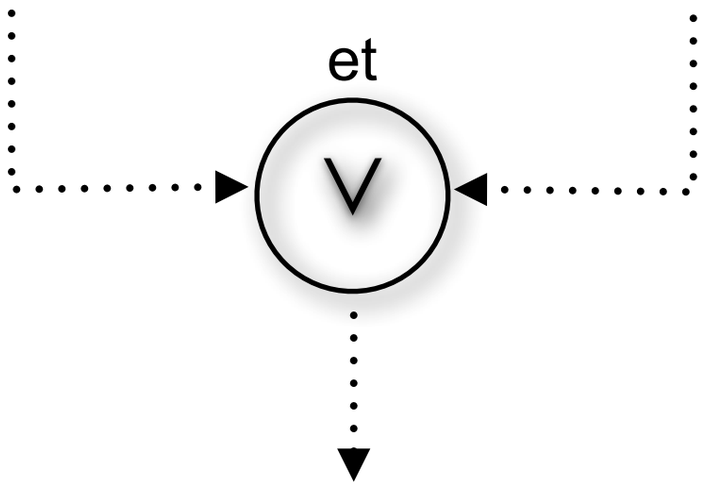


ok

net

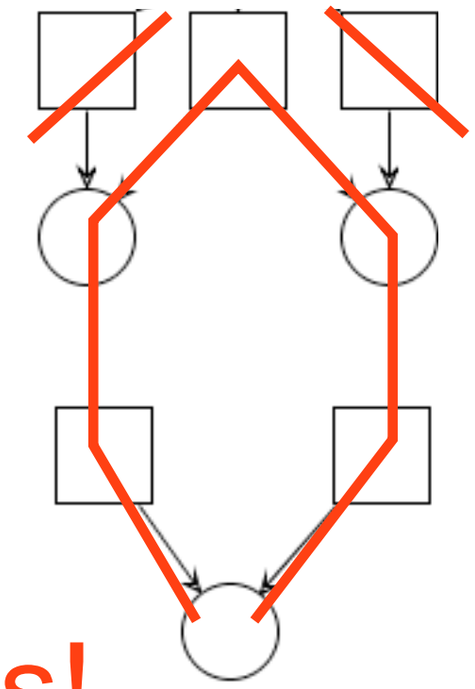
EPC

Petri net



OR join
every-time
(unmatched)

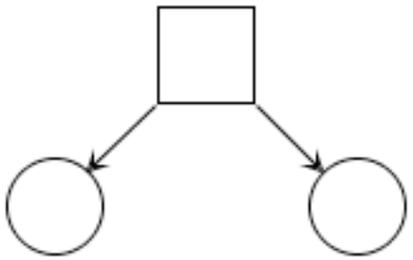
corresponding
AND split

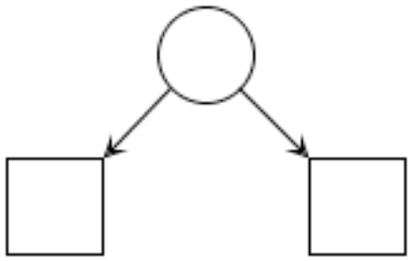


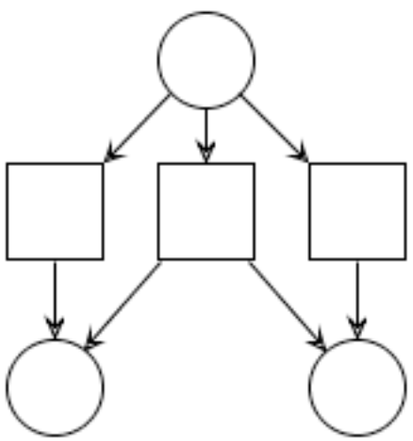
two
tokens!

net

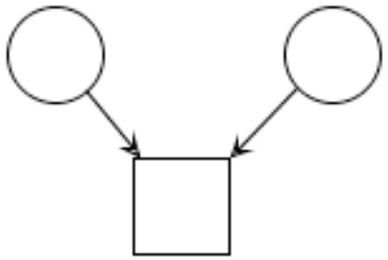
split



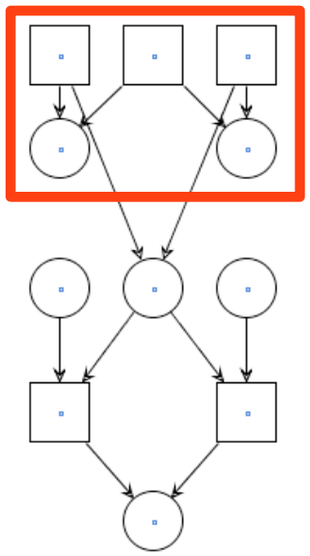




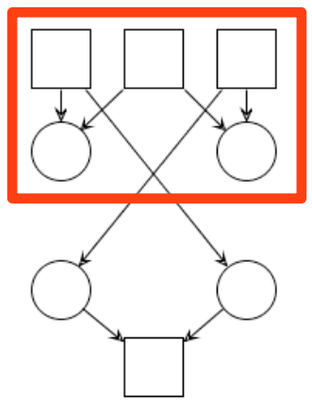
join



corresp. split

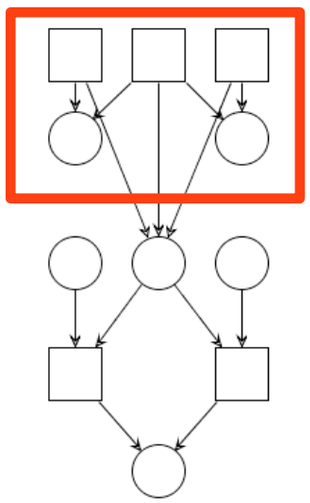


matched OR split

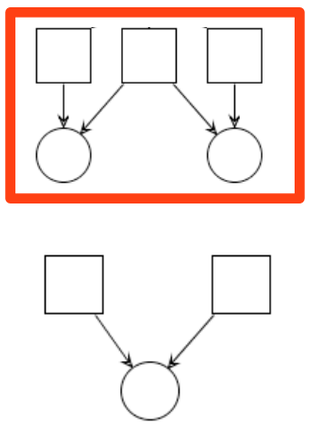


corresp. split: wfa

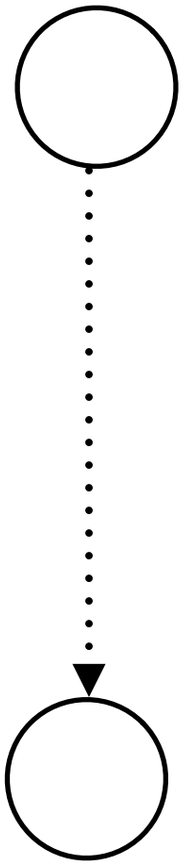
corresp. split: fc



corresp. split: et

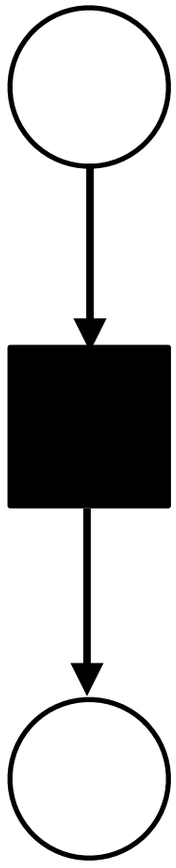


Ill-formed net

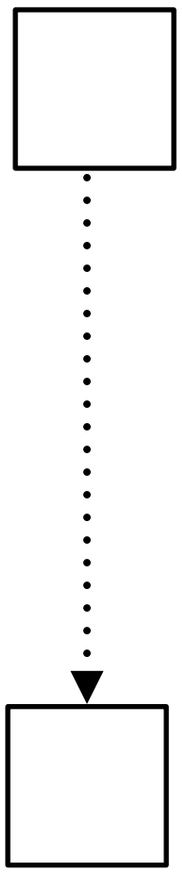


Petri net

dummy transition

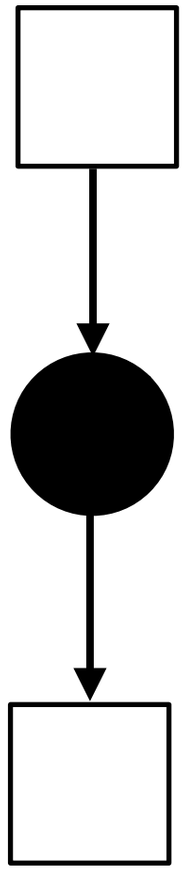


Ill-formed net

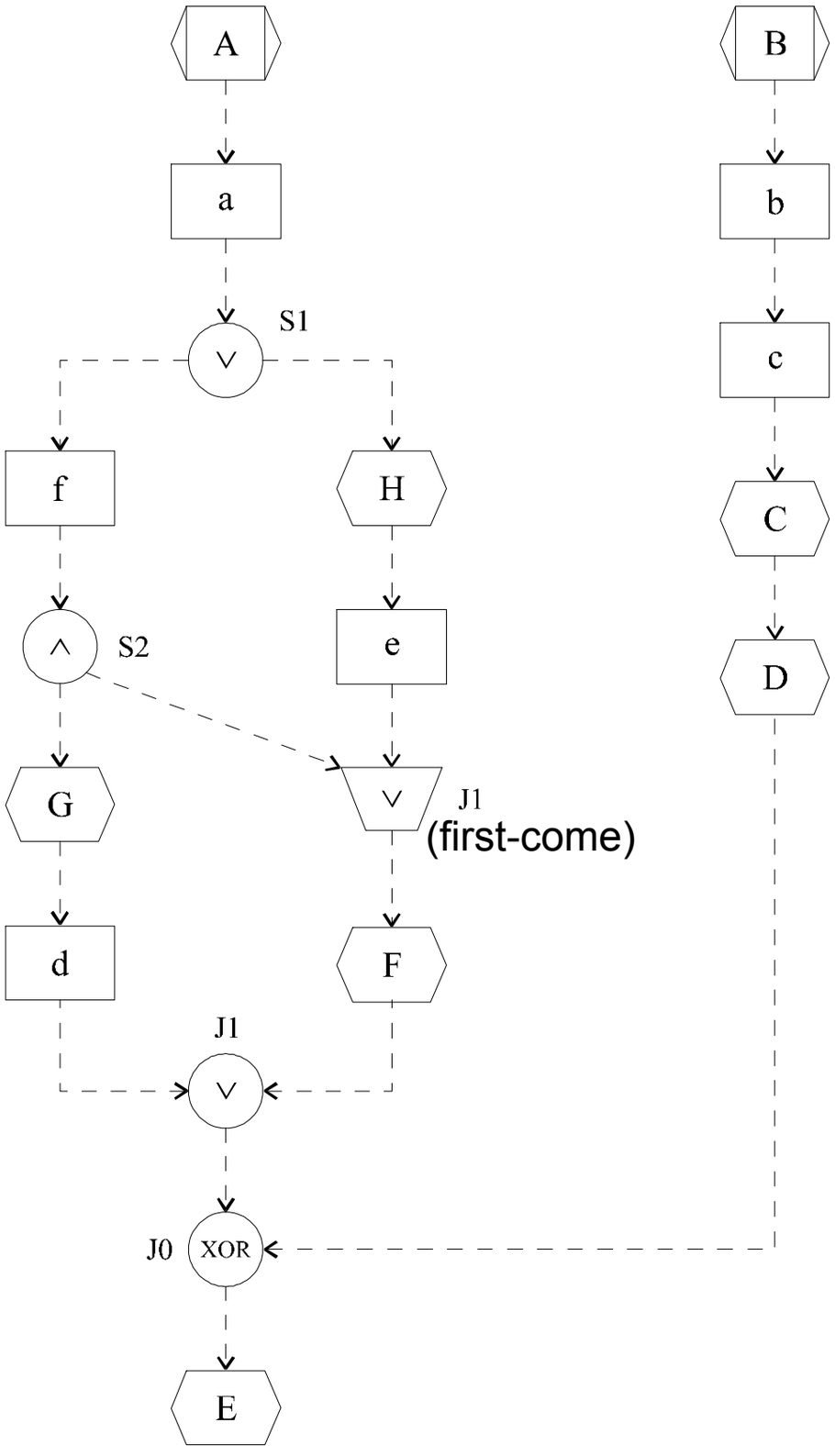


Petri net

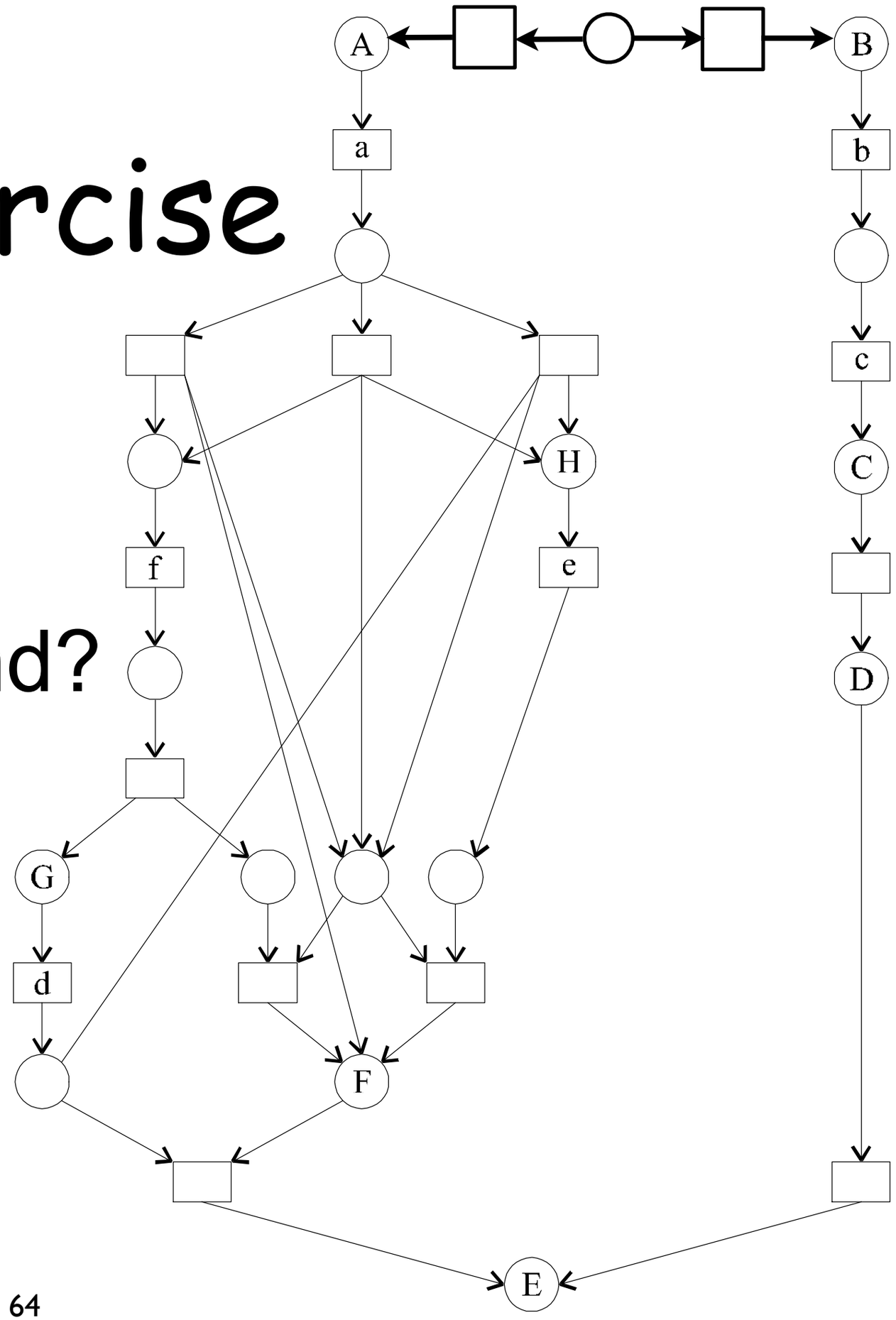
dummy place



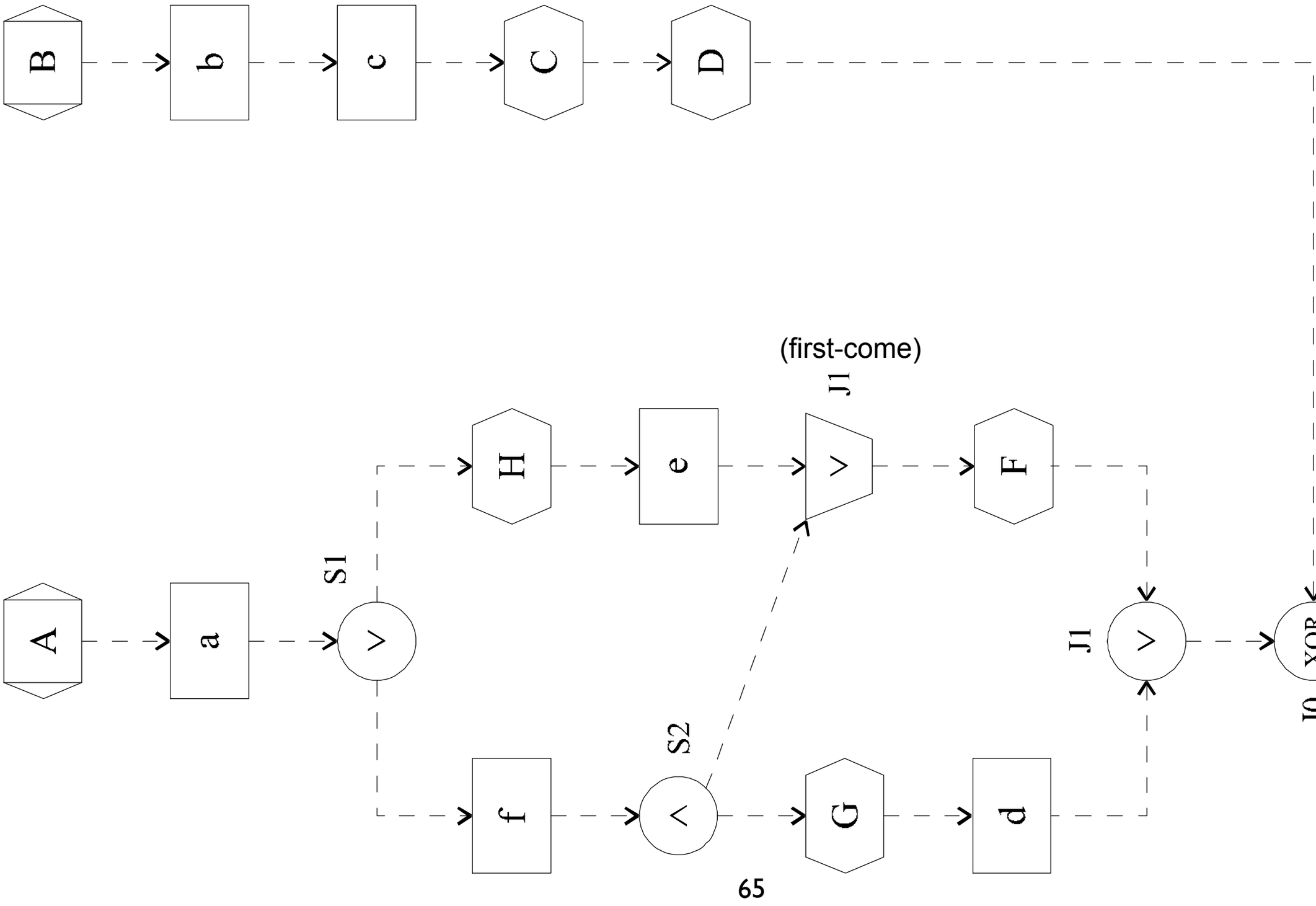
Exercise



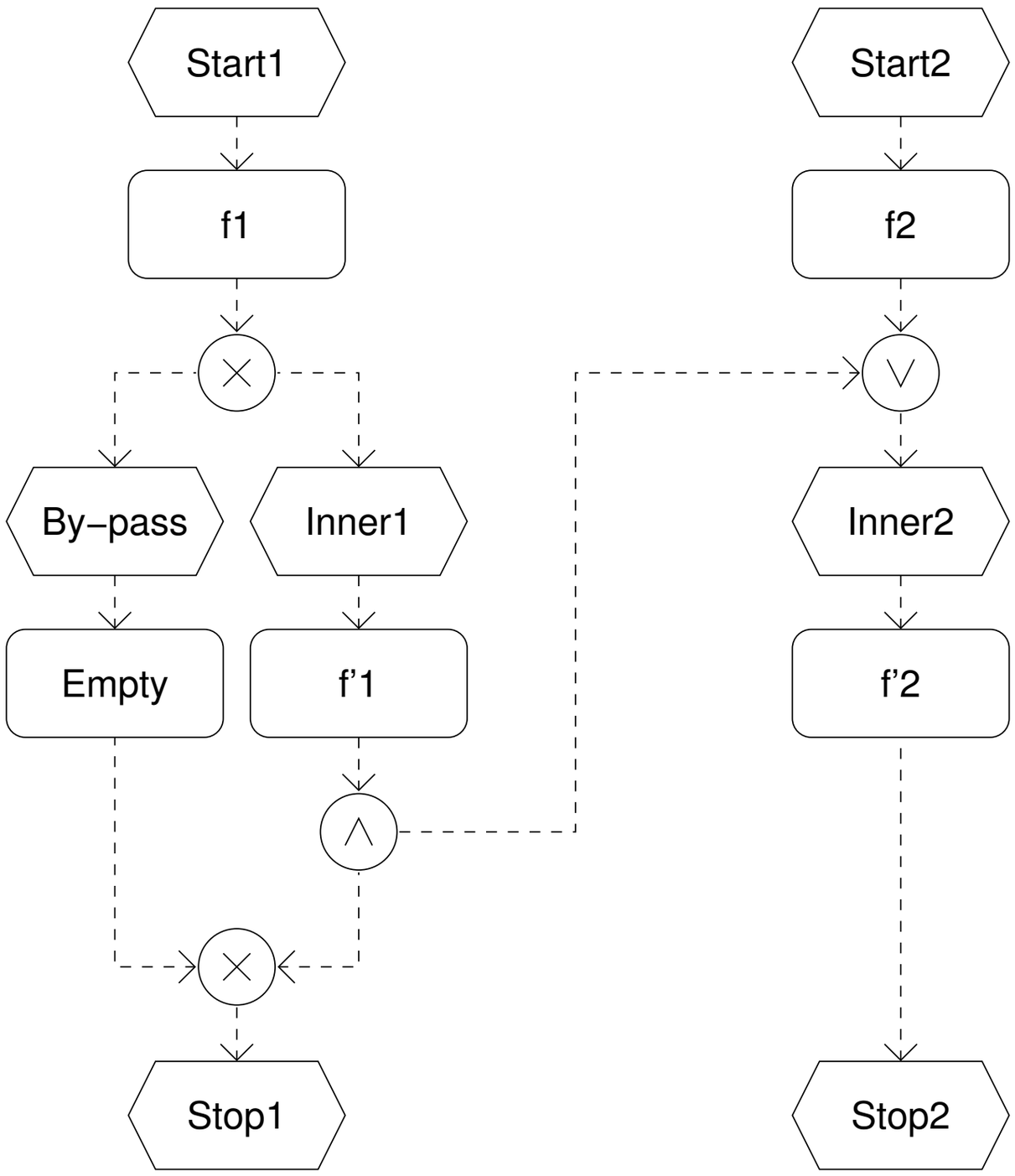
Sound?



ZOOM IN



Exercise



Sound?

Summary of possible problems

We need to decorate the EPC diagram
OR-join decorated with corresponding splits
OR-join decorated with policies (unmatched split)

Split / join mismatch may induce unexpected behaviour

Possible introduction of dummy places and transitions

Second attempt

(no decoration available)

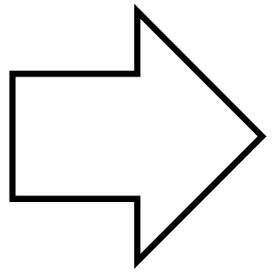
Simplified EPC

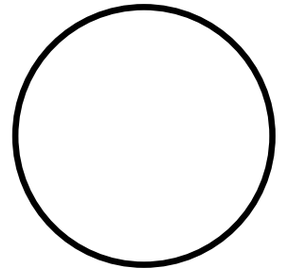
We rely on event / function alternation
along paths in the diagram
and also **along paths between two connectors**

OR-connectors are not considered

EPC 2 Petri nets

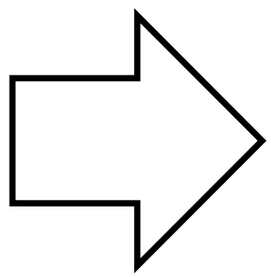
event

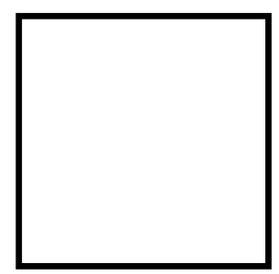




place

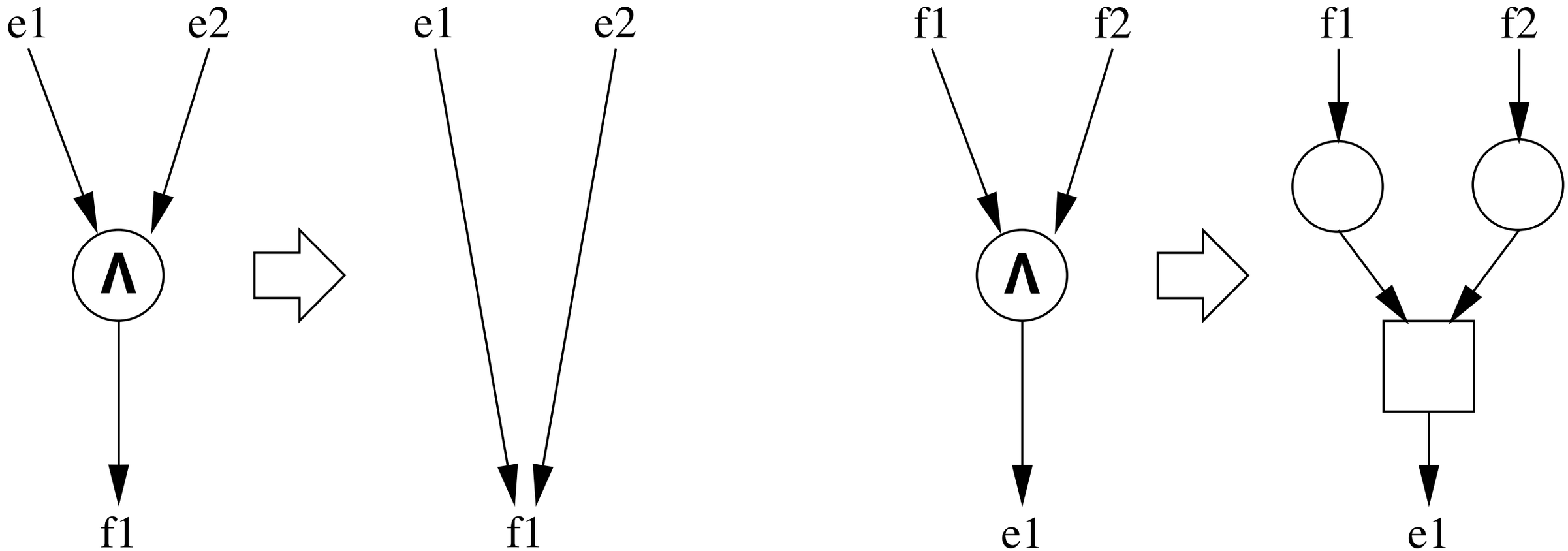
function



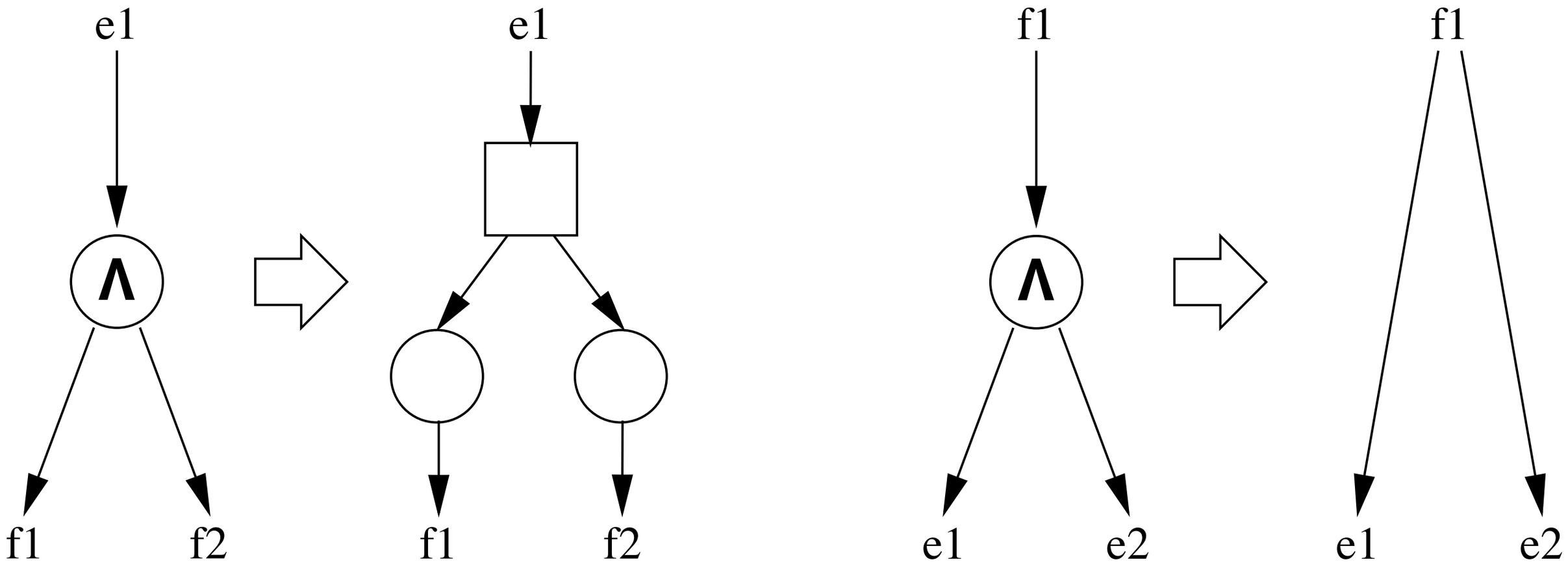


transition

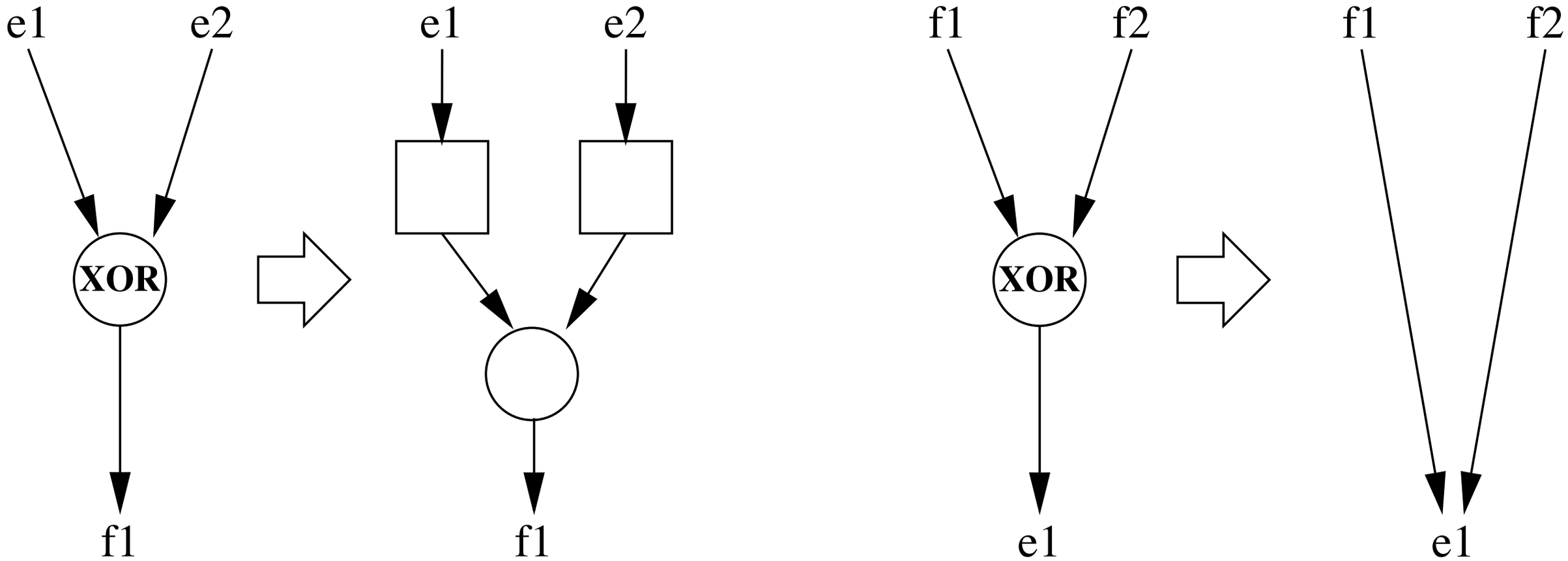
EPC 2 Petri nets



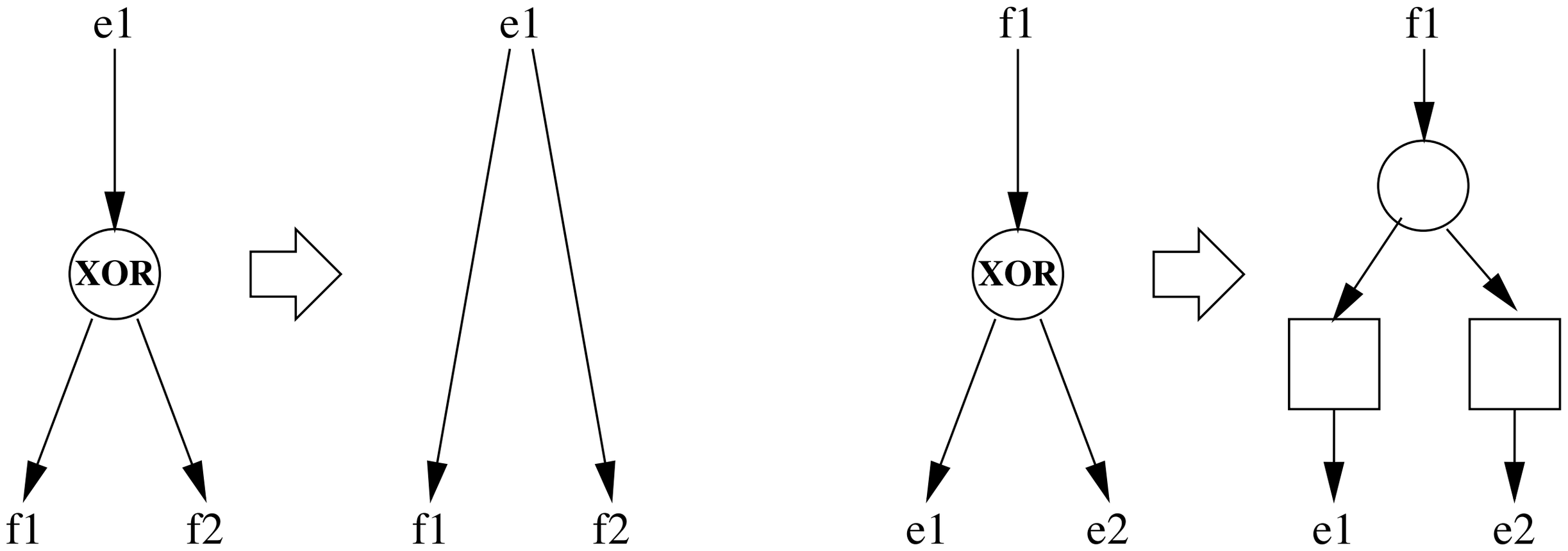
EPC 2 Petri nets



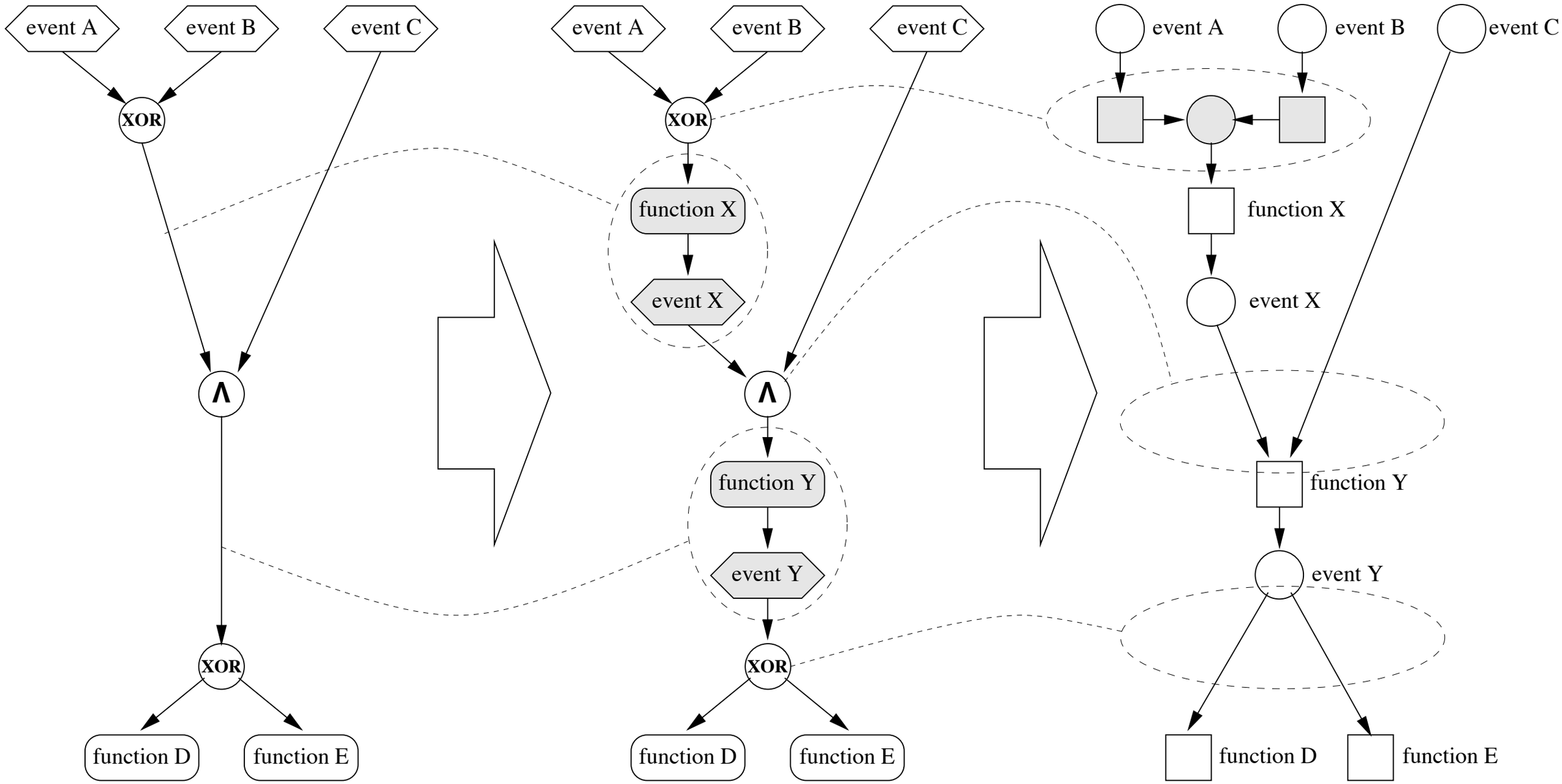
EPC 2 Petri nets



EPC 2 Petri nets



EPC 2 nets: Example

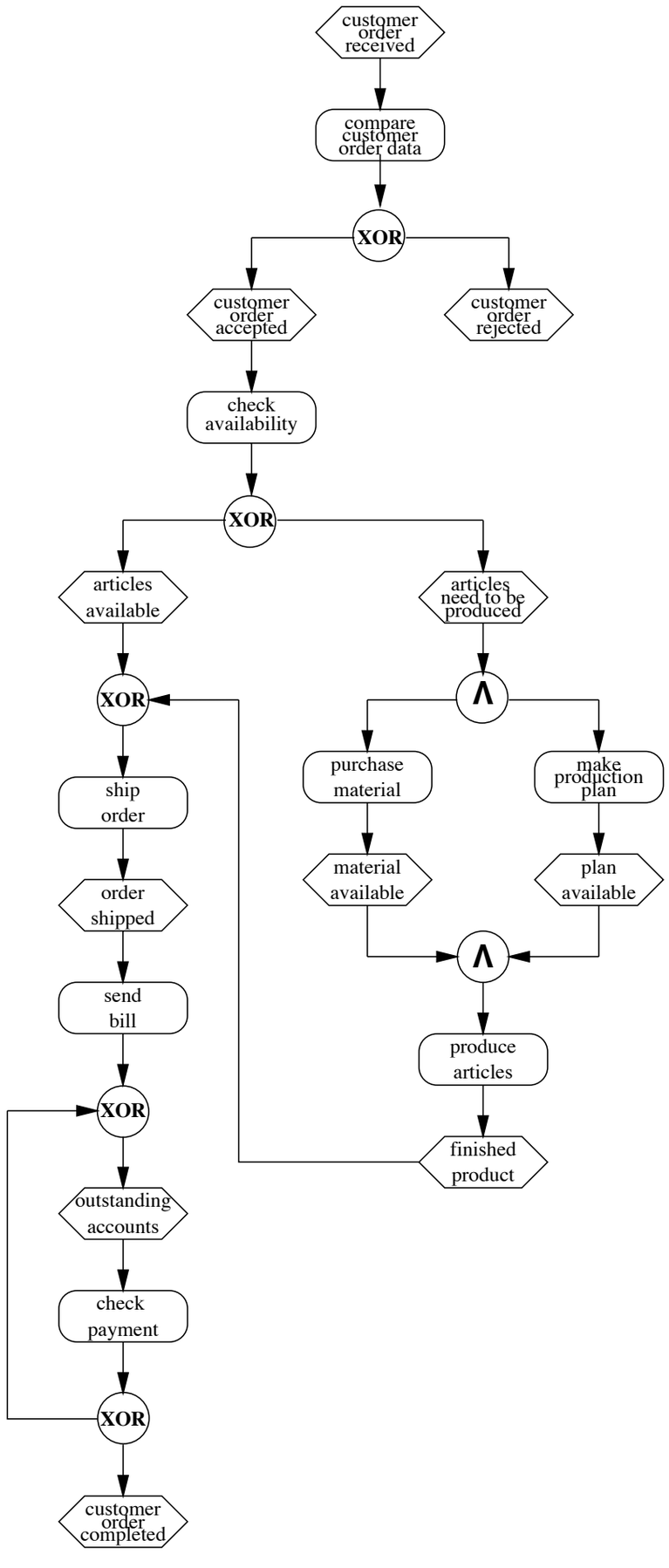


Outcome

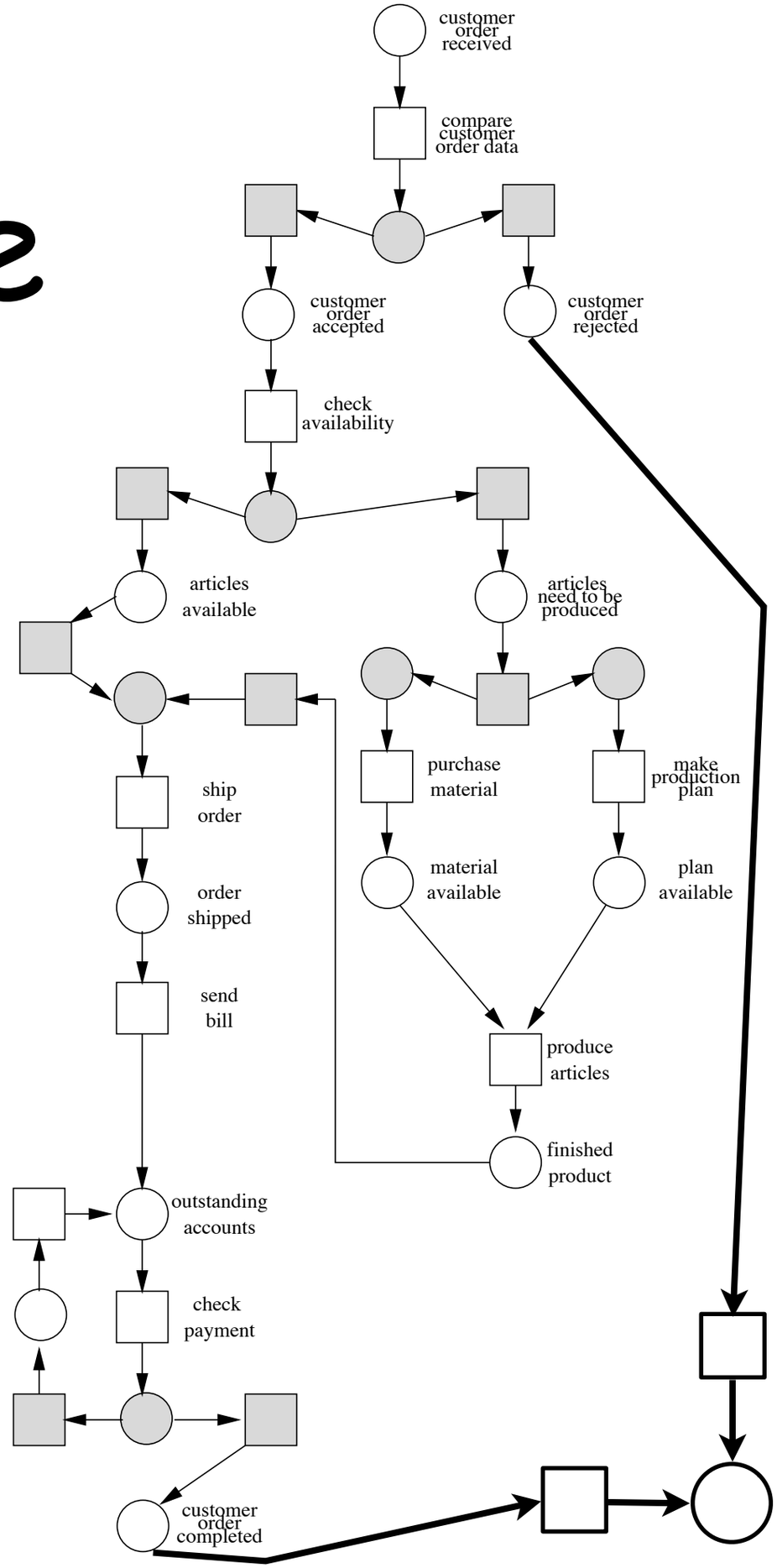
From any EPC we derive a free-choice net

Moreover, if we add unique start / end events
(and suitable transitions attached to them)
the net is a workflow net

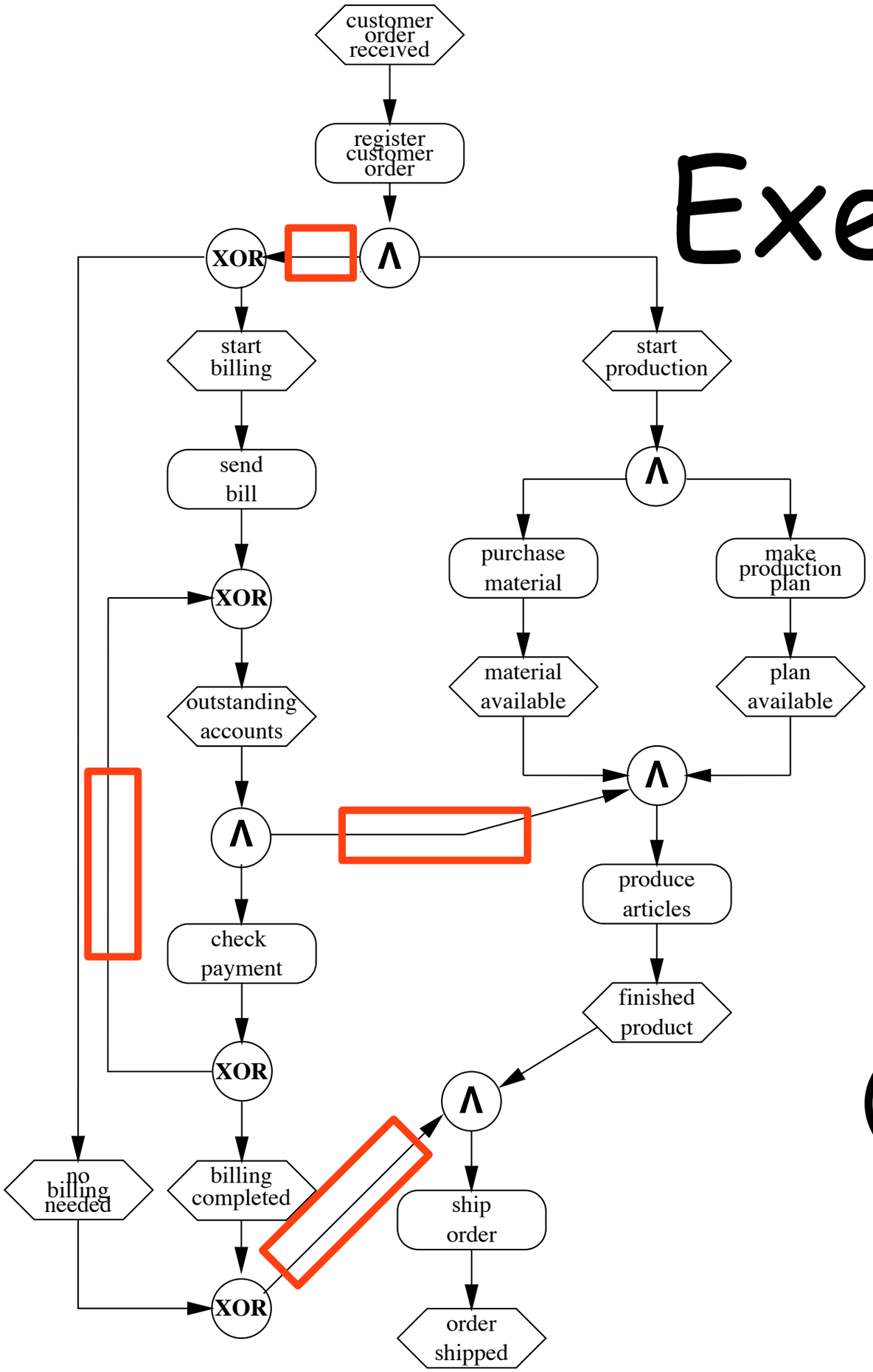
Exercise



Sound!

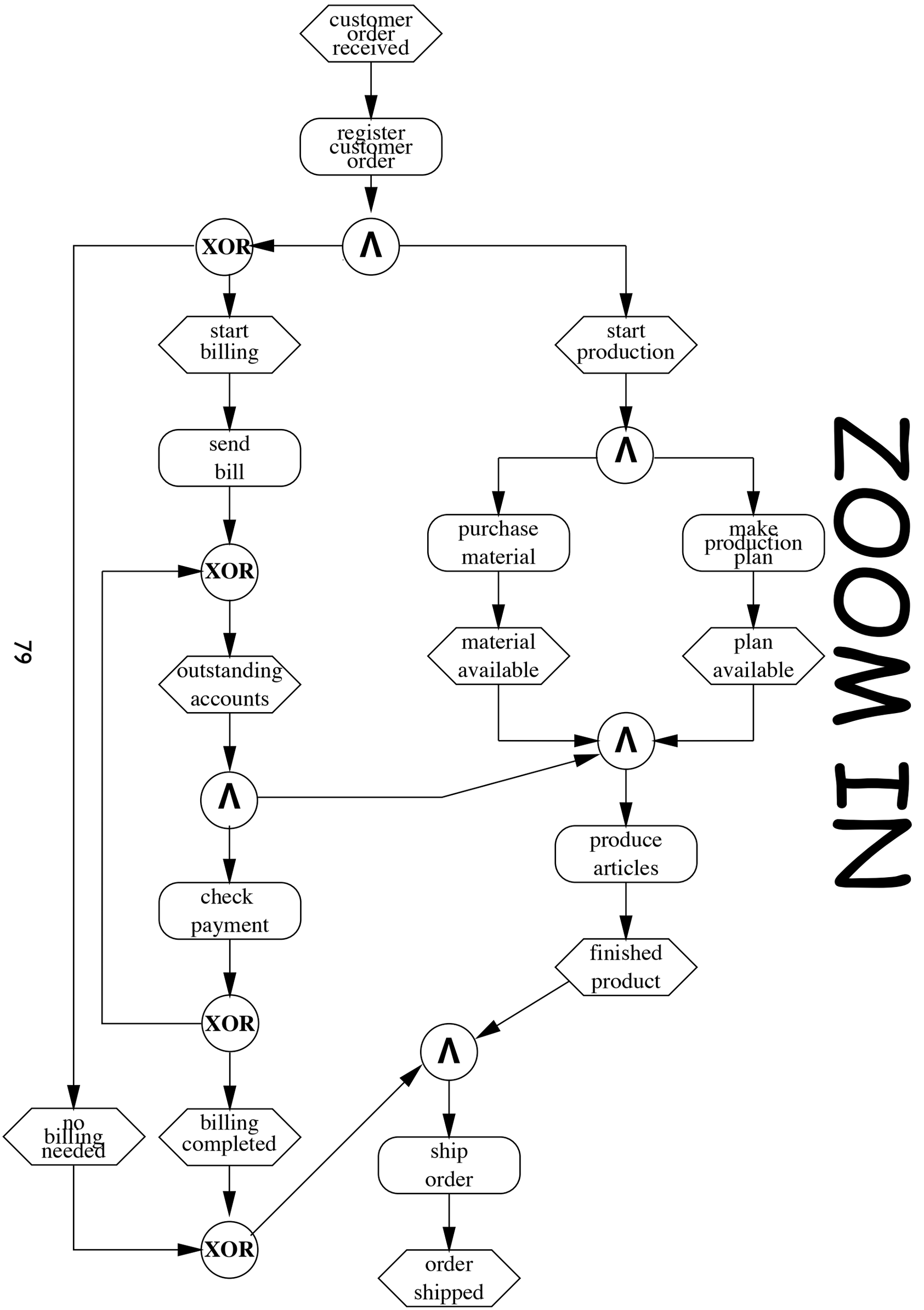


Exercise



Sound?

(remind to add dummy events and functions)


customer order received
register customer order
XOR
∧
start billing
start production
send bill
∧
purchase material
make production plan
XOR
material available
plan available
outstanding accounts
∧
∧
produce articles
check payment
finished product
XOR
∧
no billing needed
billing completed
ship order
XOR
order shipped

Relaxed soundness (a third attempt)

Popularity vs superiority

EPC are a quite successful, semiformal notation

They lack a comprehensive and consistent syntax

They lack even more a corresponding semantics

You may enrich the notation, but people will dislike or misinterpret implementation policies

You may restrict the notation, but people will prefer the more liberal (flexible) syntax and ignore the guidelines

Remember some good old friends

Chief Process Officer

EPC

Process participants

System architect

Business engineer

WFnet

Knowledge worker

Process responsible

Process designer

System developer

What are ultimately business process?

Graphical language to **communicate** concepts

Careful selection of symbols
shapes, colors, arrows
(the alphabet is necessary for communication)

Greatest common denominator of the people involved

Intuitive meaning
(verbal description, no math involved)

A secret not to tell

The more ways are to interpret a certain construct the more likely an agreement will be reached

A pragmatic consideration

Moreover

in the **analysis phase**
the participants may not be ready
to **finalise** the specification
and decide for the **correct interpretation**

However

ambiguous process description constitutes
a **major problem** in the **design phase**

Consequences

Yet

it is important to find out **flaws** as **soon** as possible

Therefore

we need to fix a **formal representation**
that **preserves all ambiguities**

Problem

EPC is fine (widely adopted)

WF net offer a useful tool

but

Soundness is too demanding at early stages

Relaxed soundness

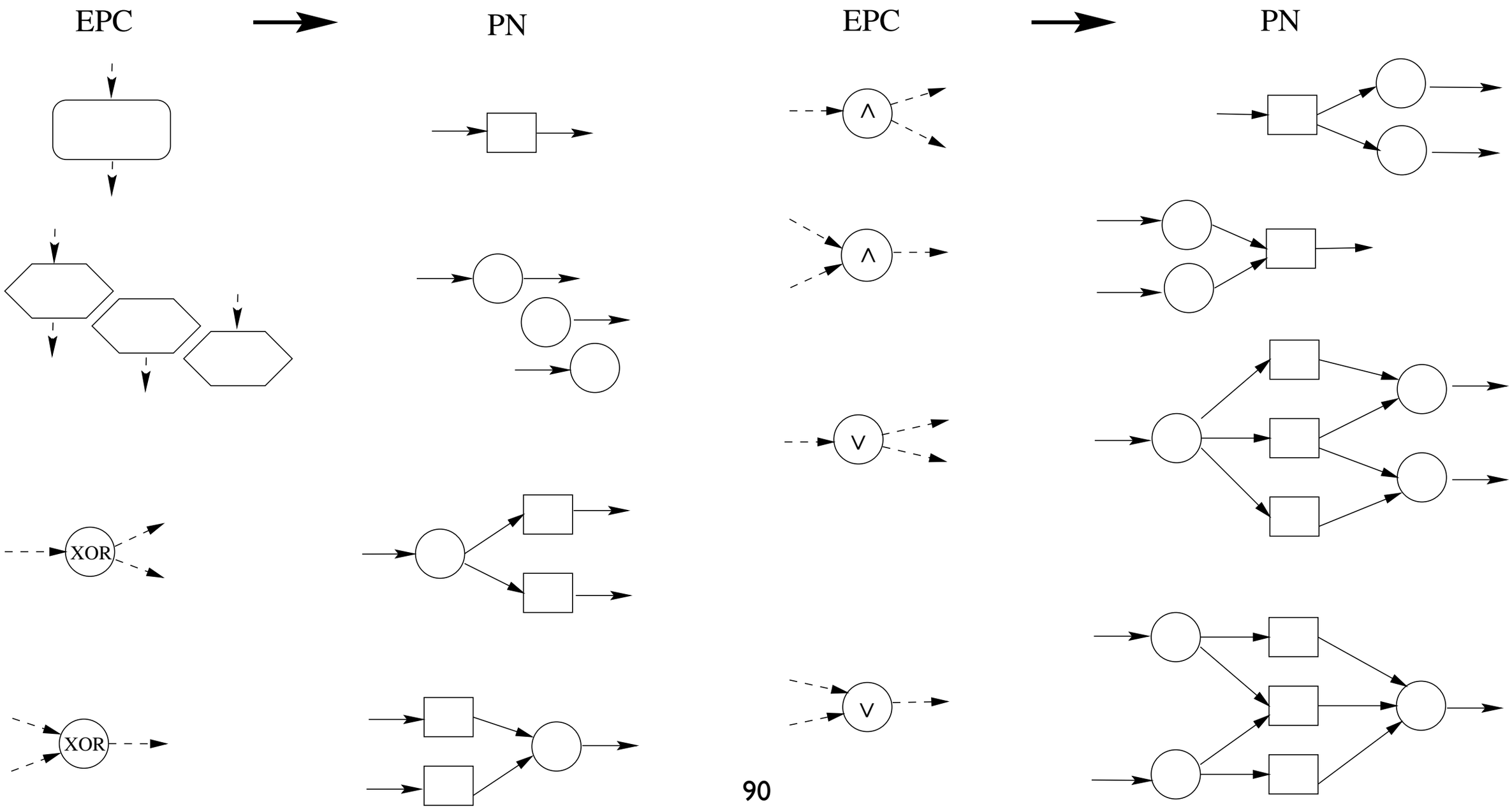
A **sound** behaviour:
we move from a start event to an end event
so that nothing blocks or remains undone

Execution paths leading to **unsound** behaviour
can be used to infer potential mistakes in the EPC

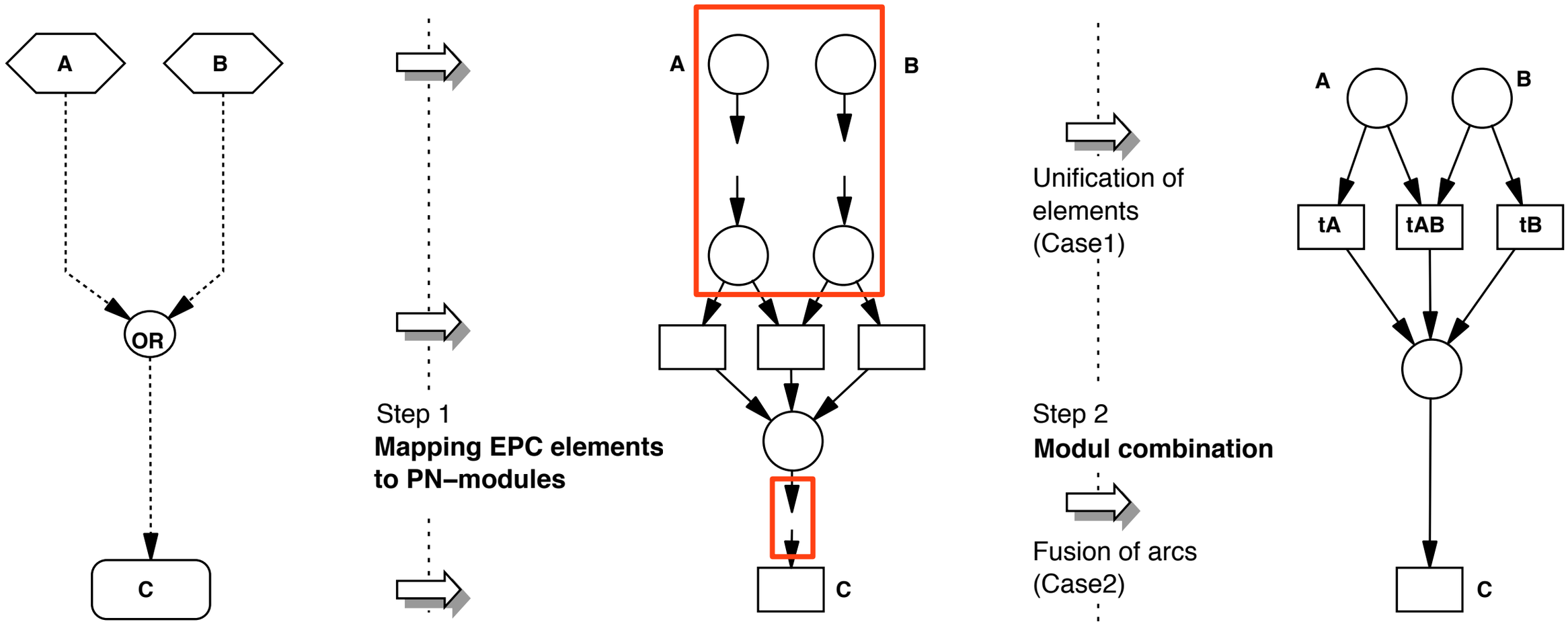
If some unsound behaviour is possible
but **enough** sound paths exist
the process is called **relaxed sound**

A 3-steps approach (keep it simple!)

Step 1: straightforward element map

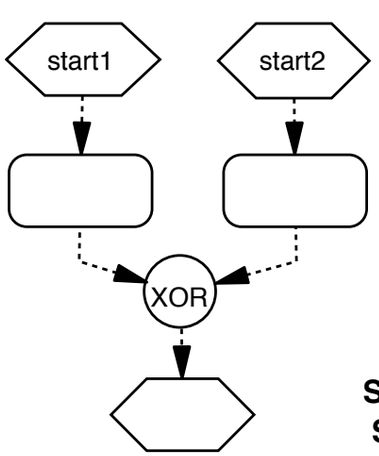


Step 2:
element fusion

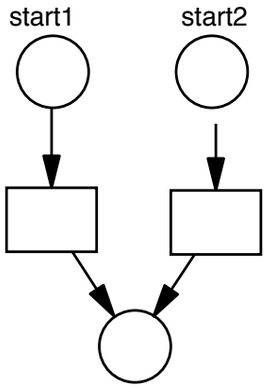


Step 3:
add unique start / end

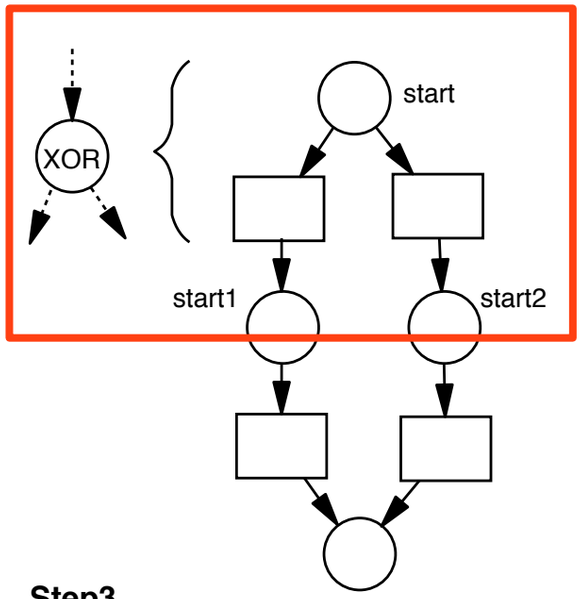
a)



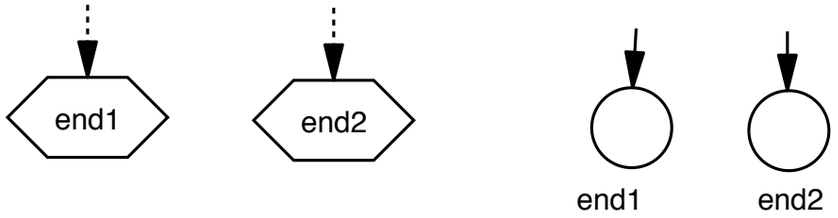
Step1 &
Step2



Step3

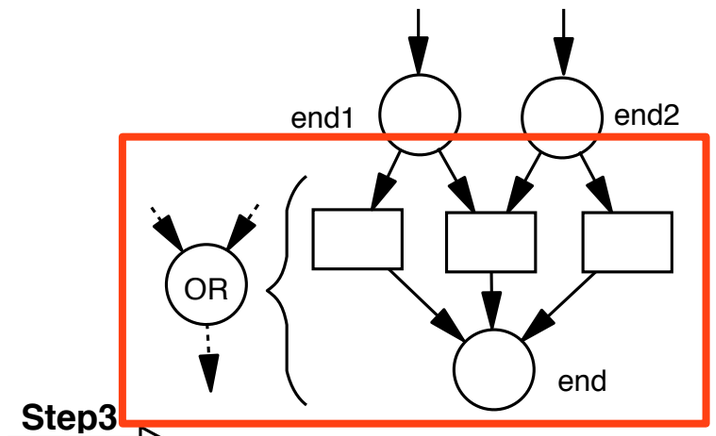


XOR start



Step1 &
Step2

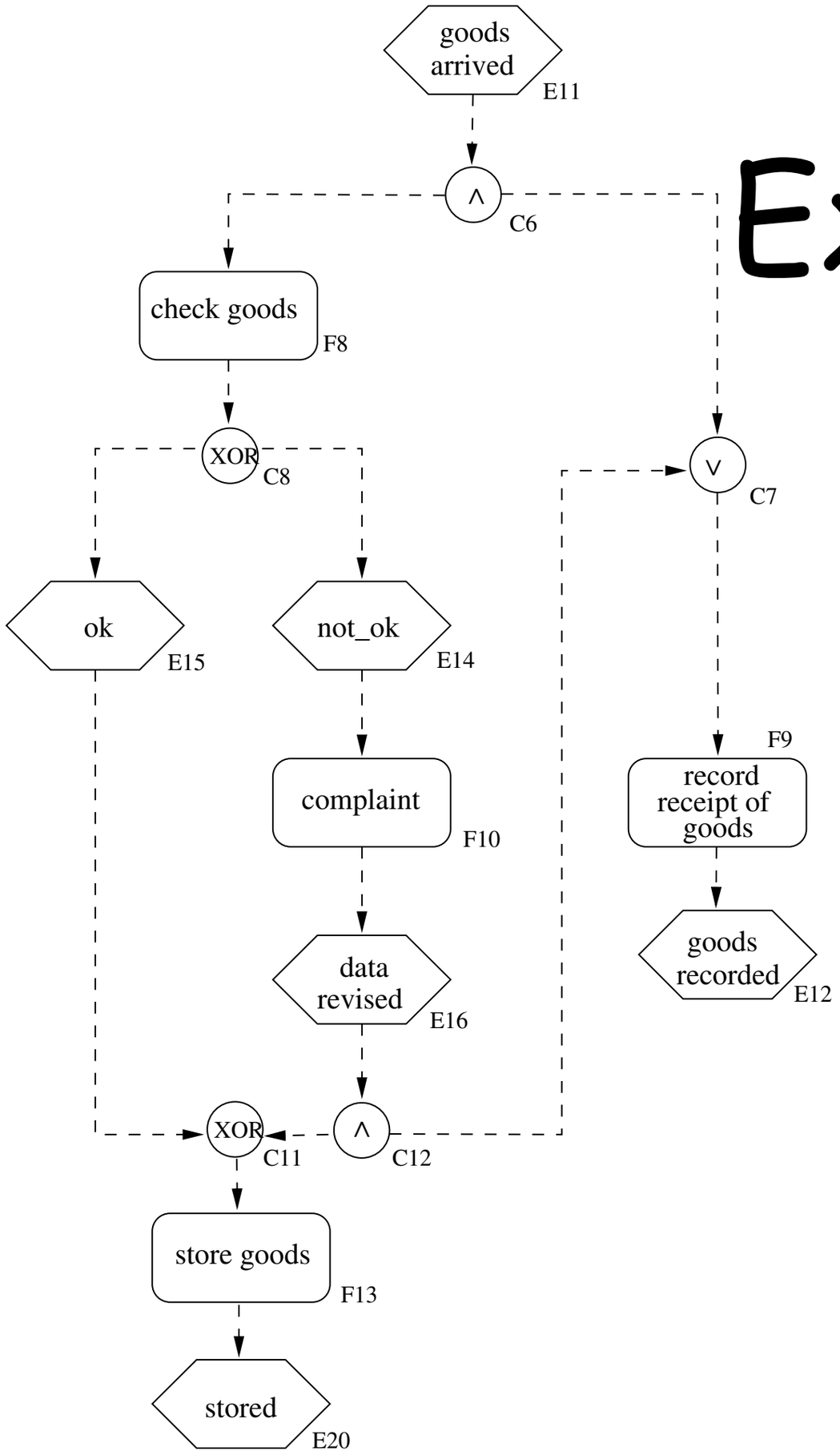
(sometimes AND can be preferred)



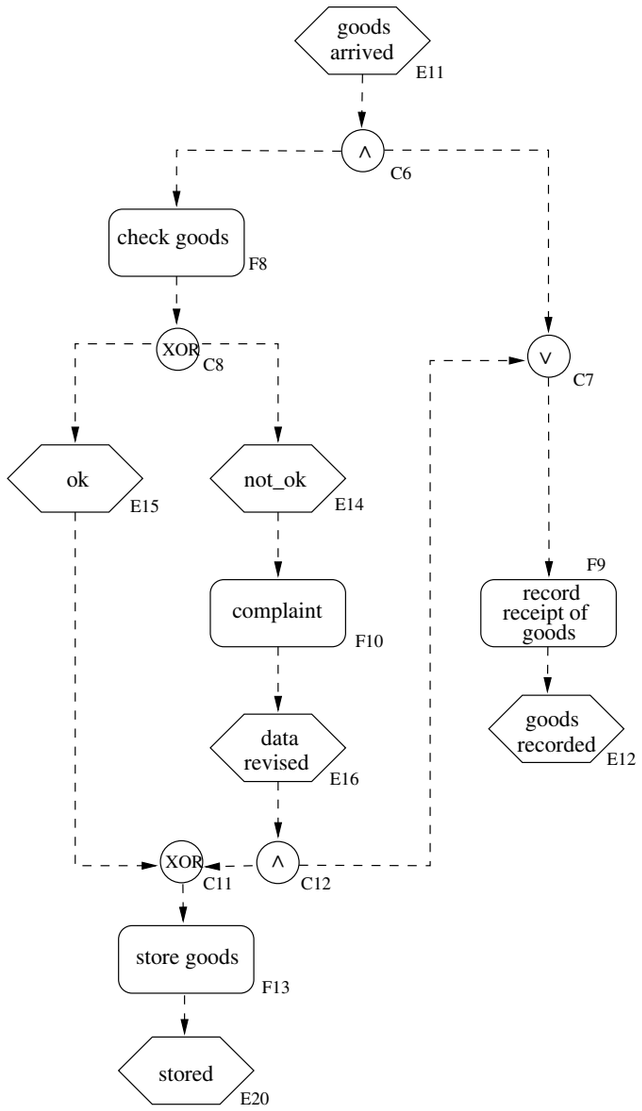
Step3

OR end

Example

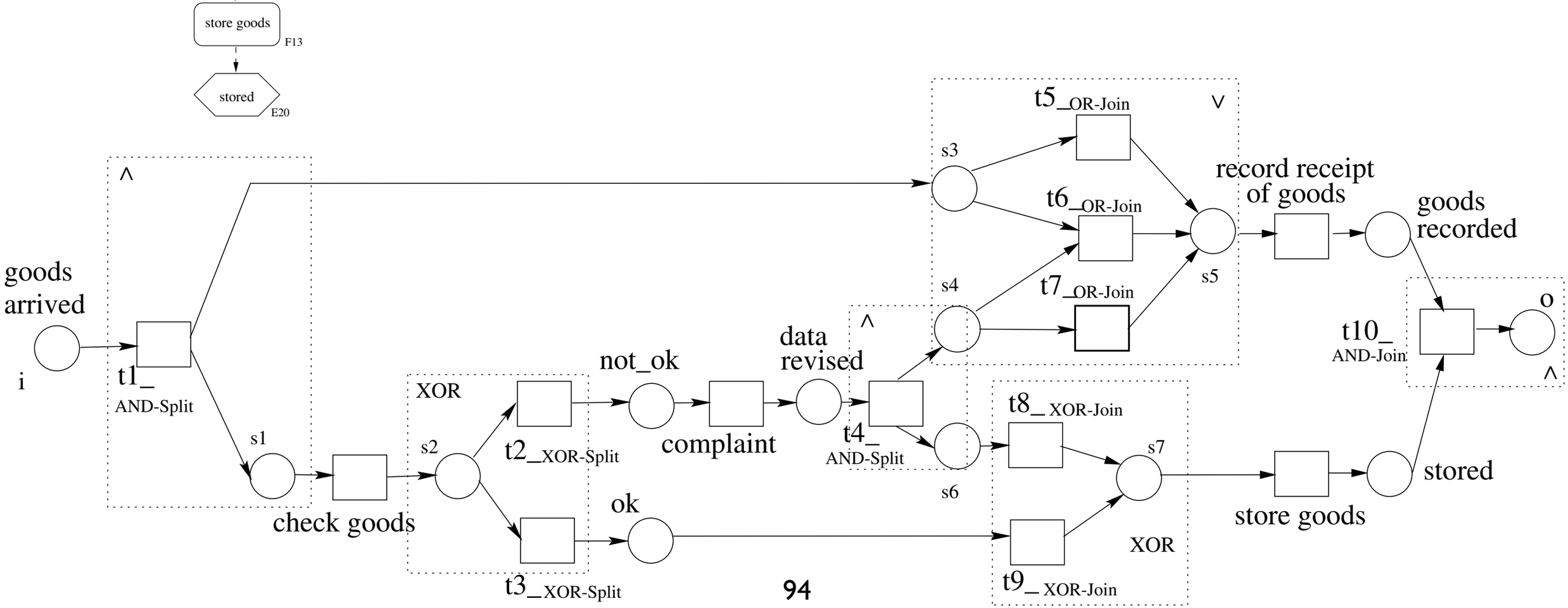


Sound?



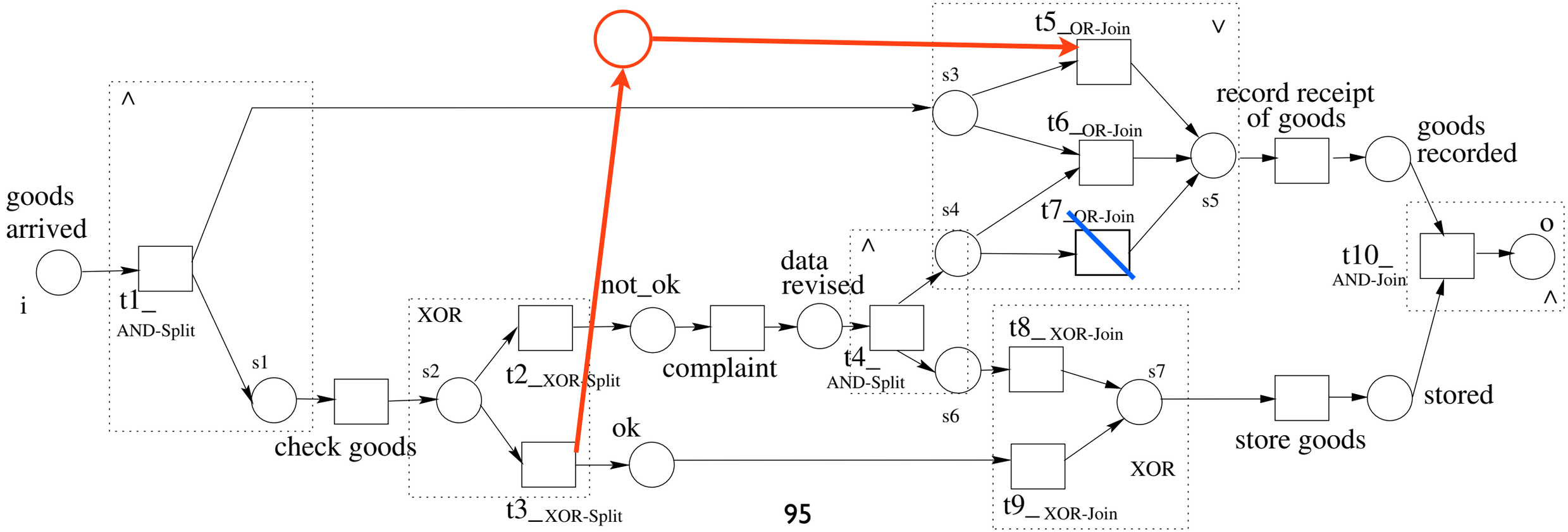
Example

Not sound!



Example

We can turn it to sound, but:
small changes in the net, turn big in EPC

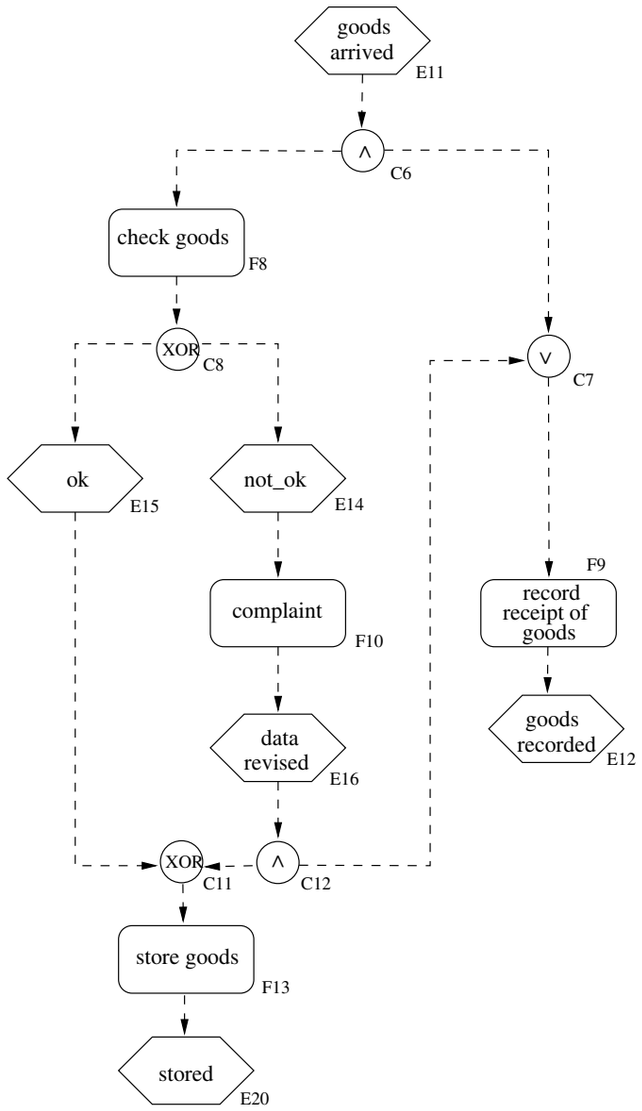


Relaxed soundness: formally

Definition: A WF net is **relaxed sound** if every transition belongs to a firing sequence that starts in state i and ends in state o

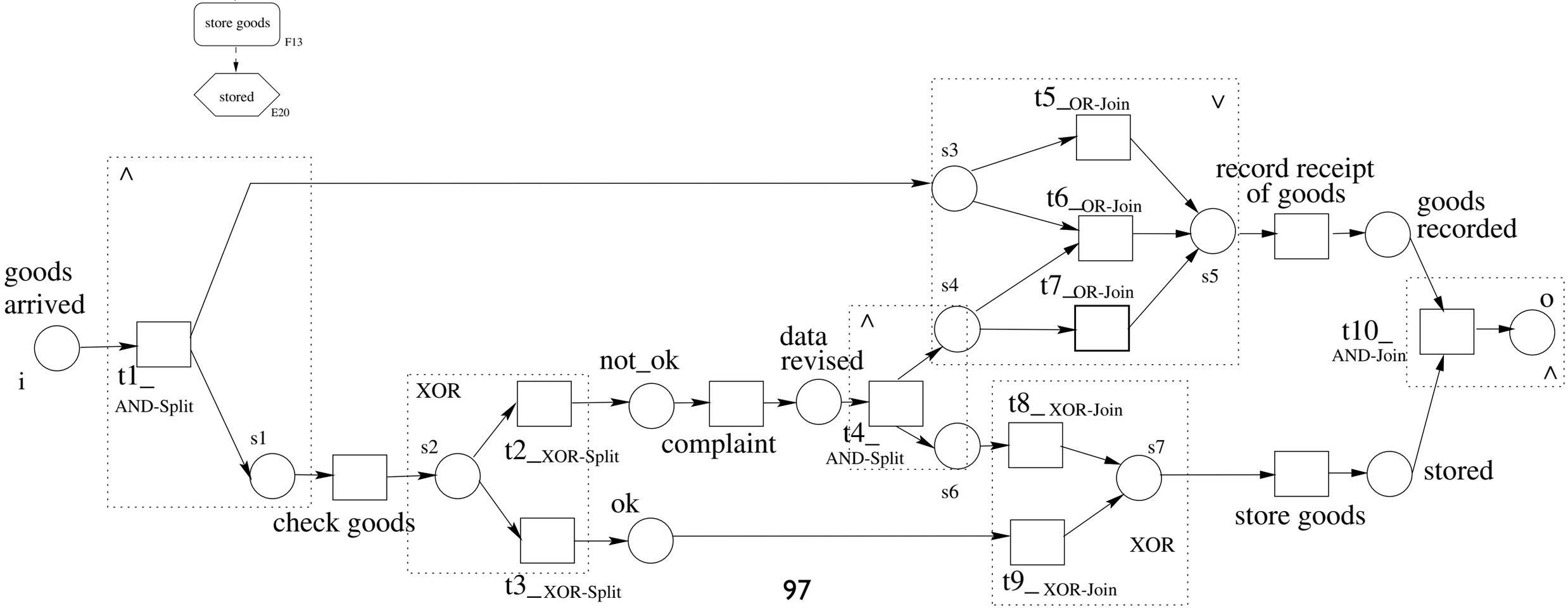
$$\forall t \in T. \exists M, M'. i \rightarrow^* M \xrightarrow{t} M' \rightarrow^* o$$

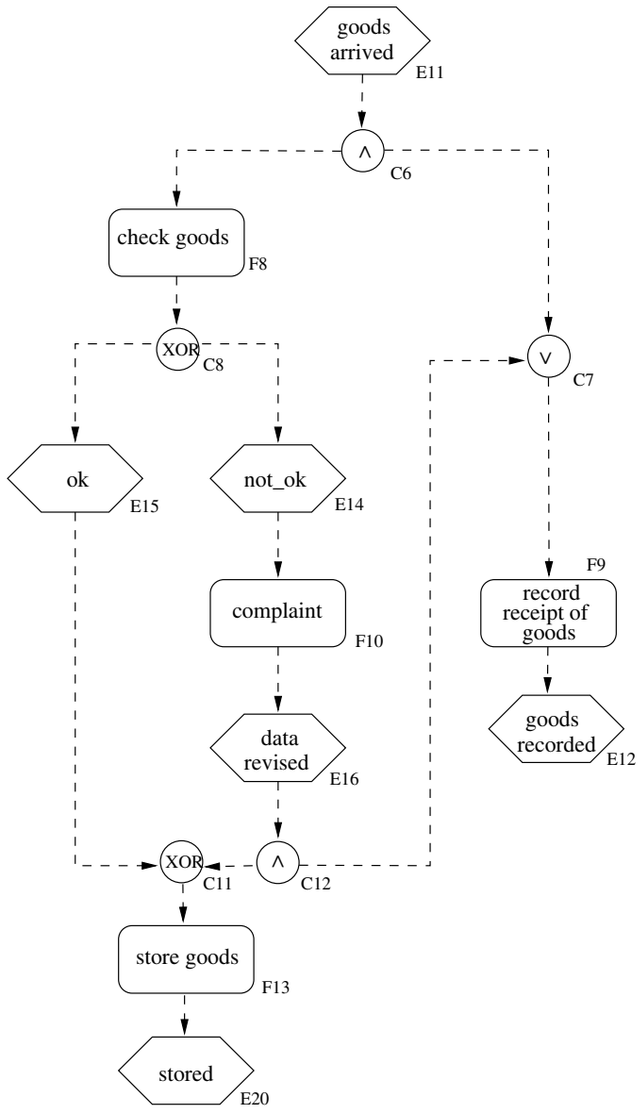
(it is sound “enough”, in the sense that all transitions are covered by at least one sound execution)



Example

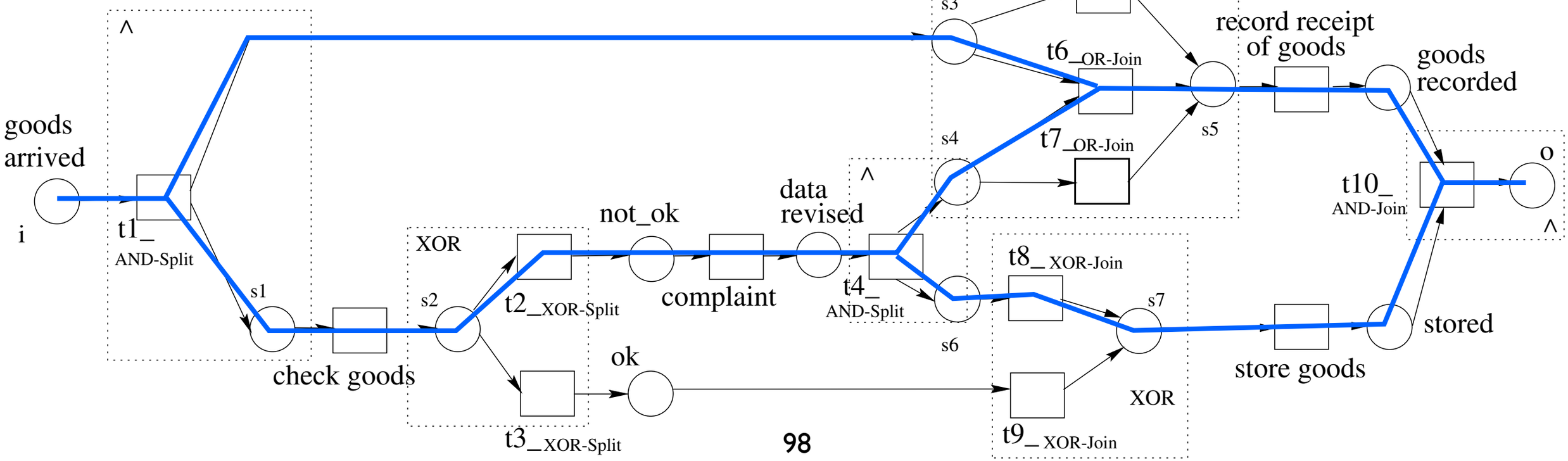
Relaxed sound?

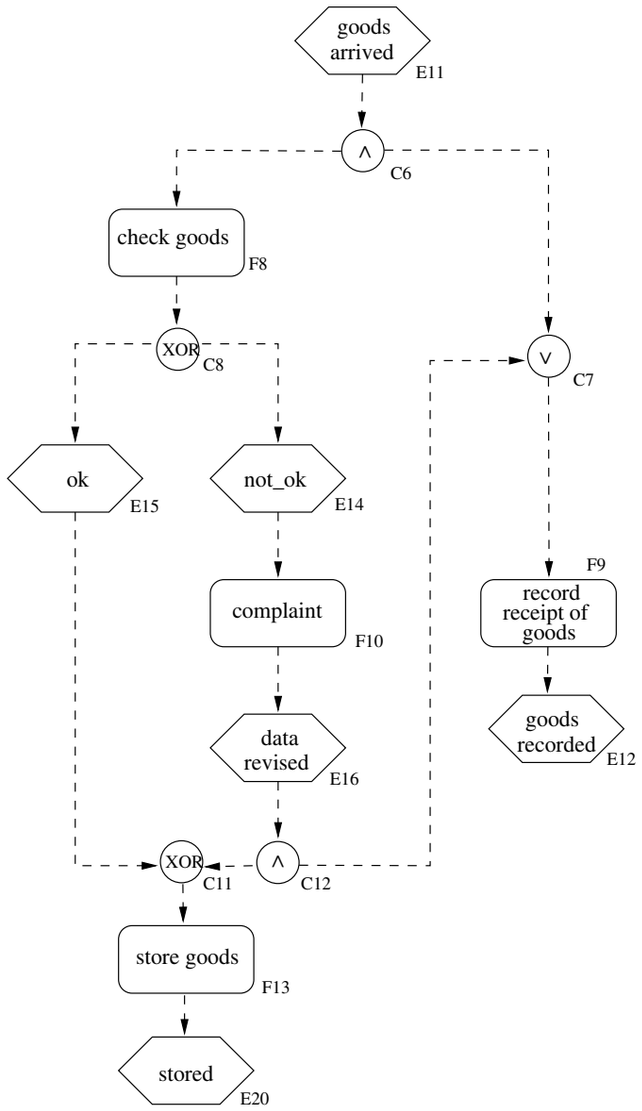




Example

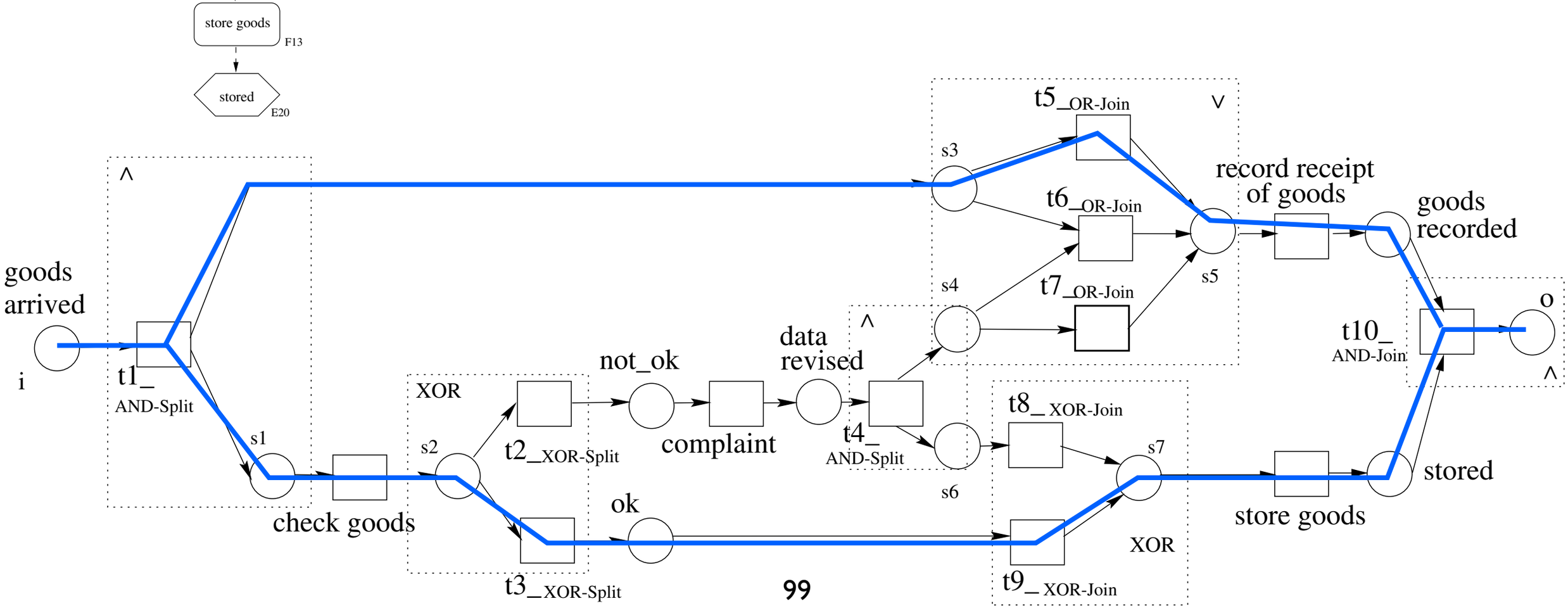
Relaxed sound?

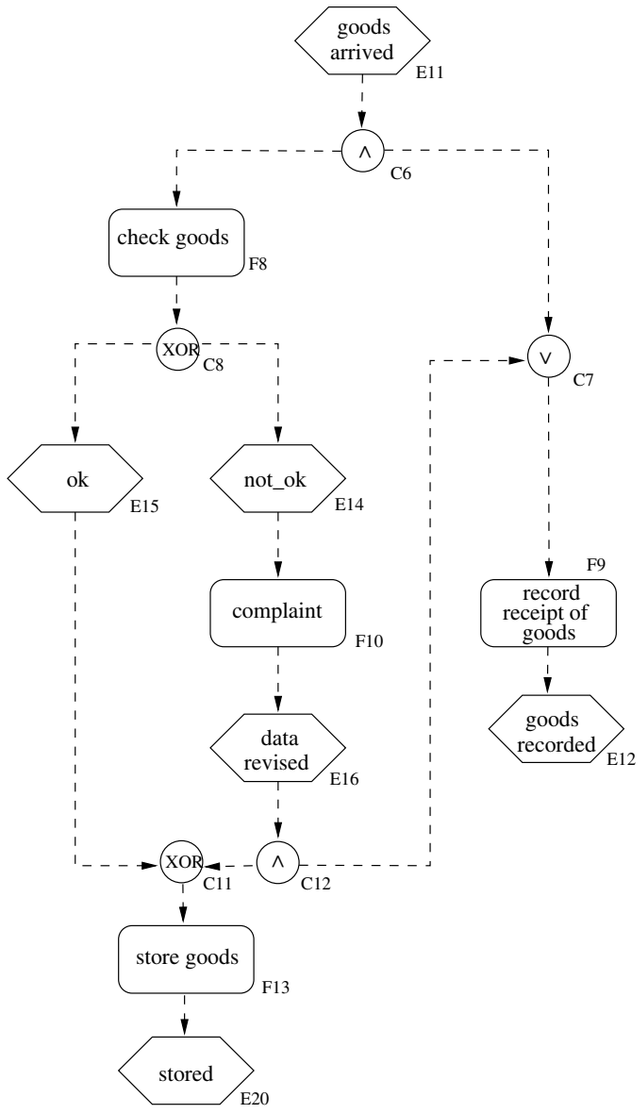




Example

Relaxed sound?

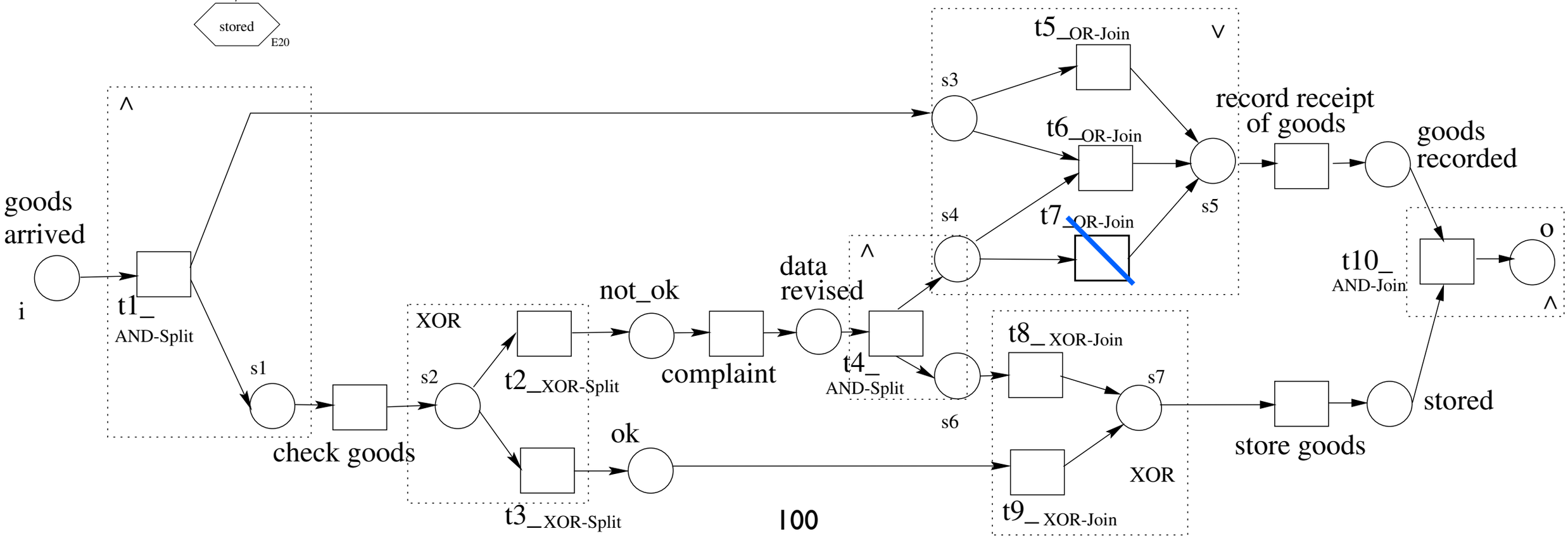




Example

Not relaxed sound!

But sound as EPC



Pros and Cons

If the WF net is **not sound**:
there are transitions that are not contained in any sound firing sequence

Hence their EPC counterparts need improvements

Relaxed soundness can be proven only by enumeration (of enough sound firing sequences)

No equivalent characterisation is known that is more convenient to check