

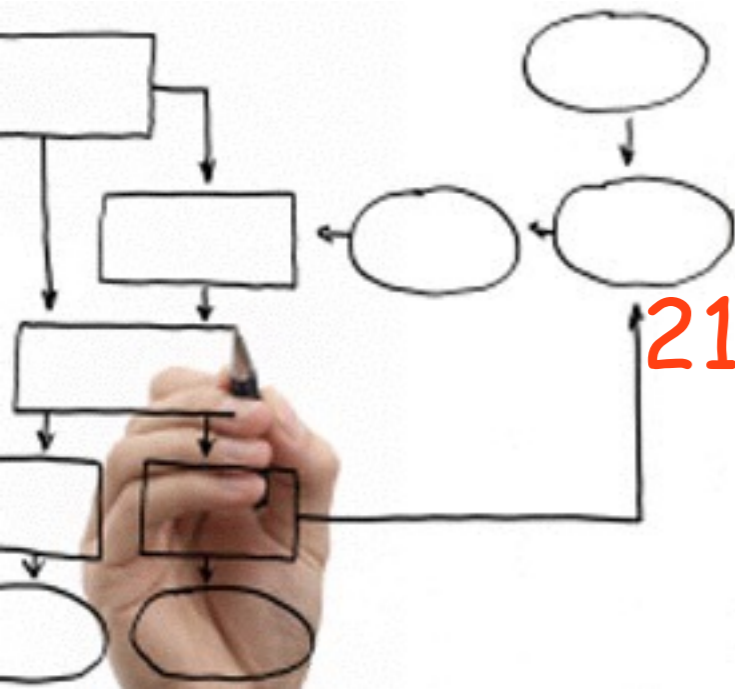
Methods for the specification and verification of business processes

MPB (6 cfu, 295AA)

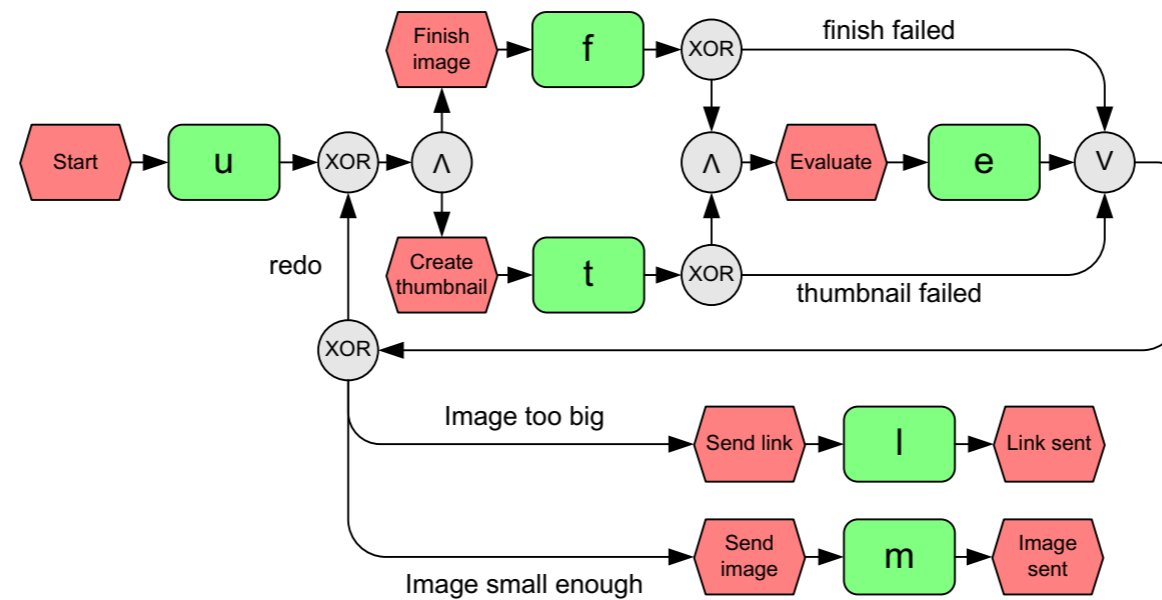
Roberto Bruni

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

21 - Event-driven process chains



Object



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

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

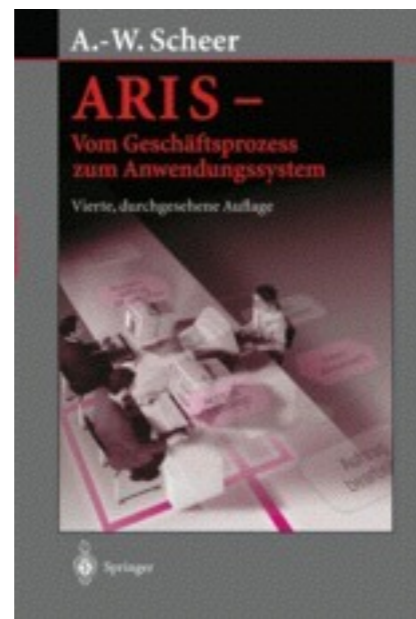
Rather informal notation
simple and easy-to-understand

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

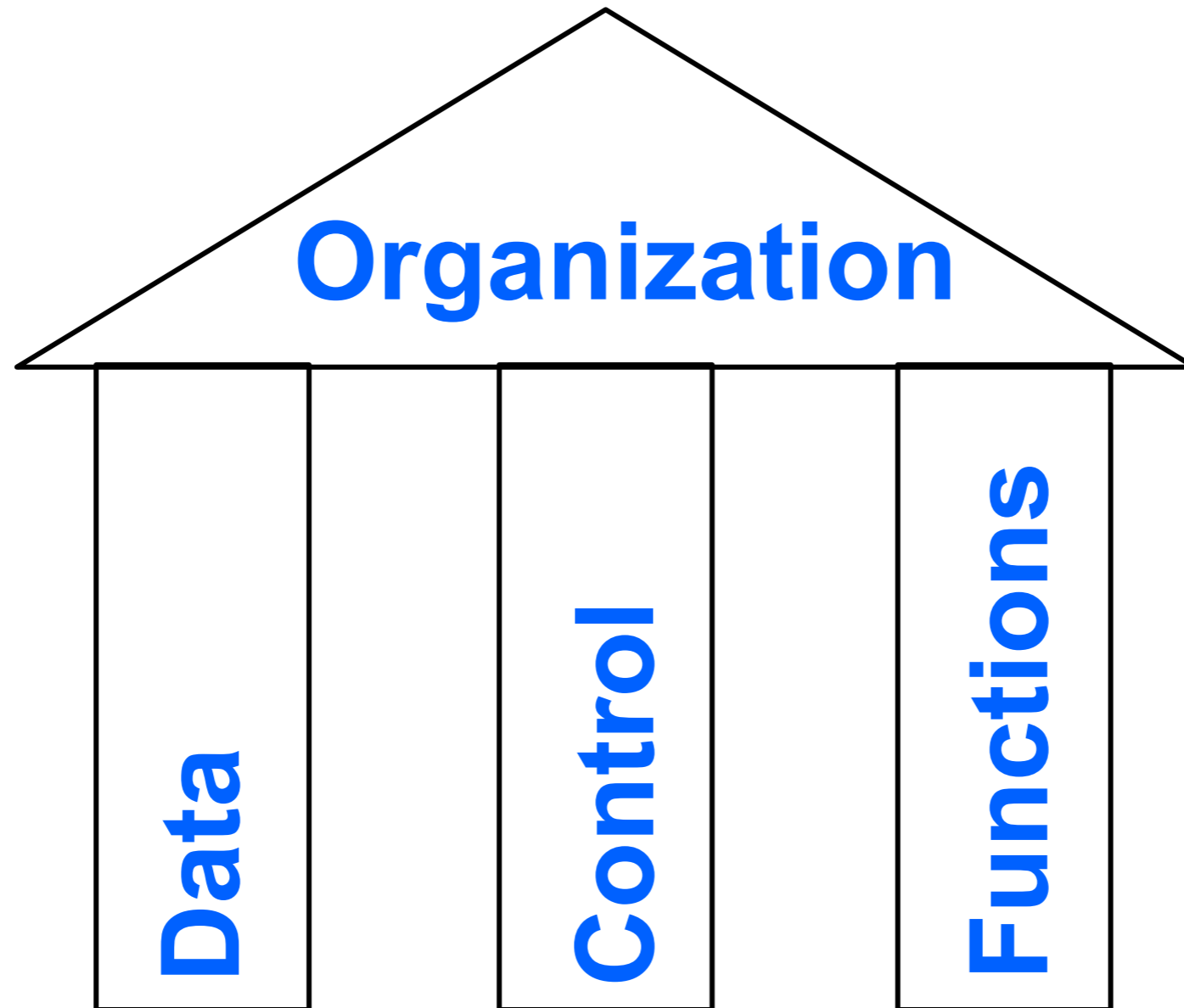
EPC origin

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

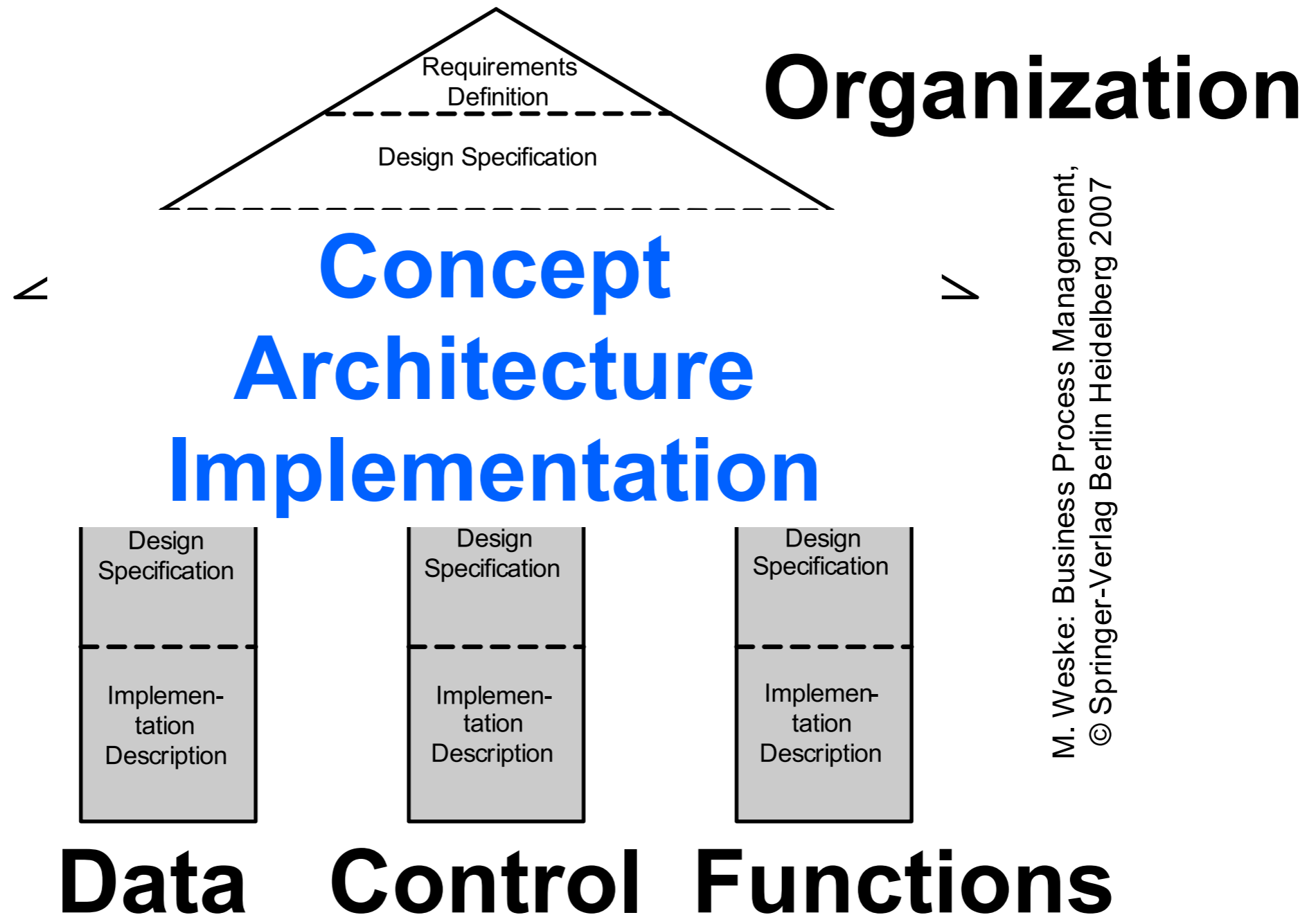


Part of a holistic modelling approach called
ARIS framework
(Architecture of Integrated Information Systems)

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

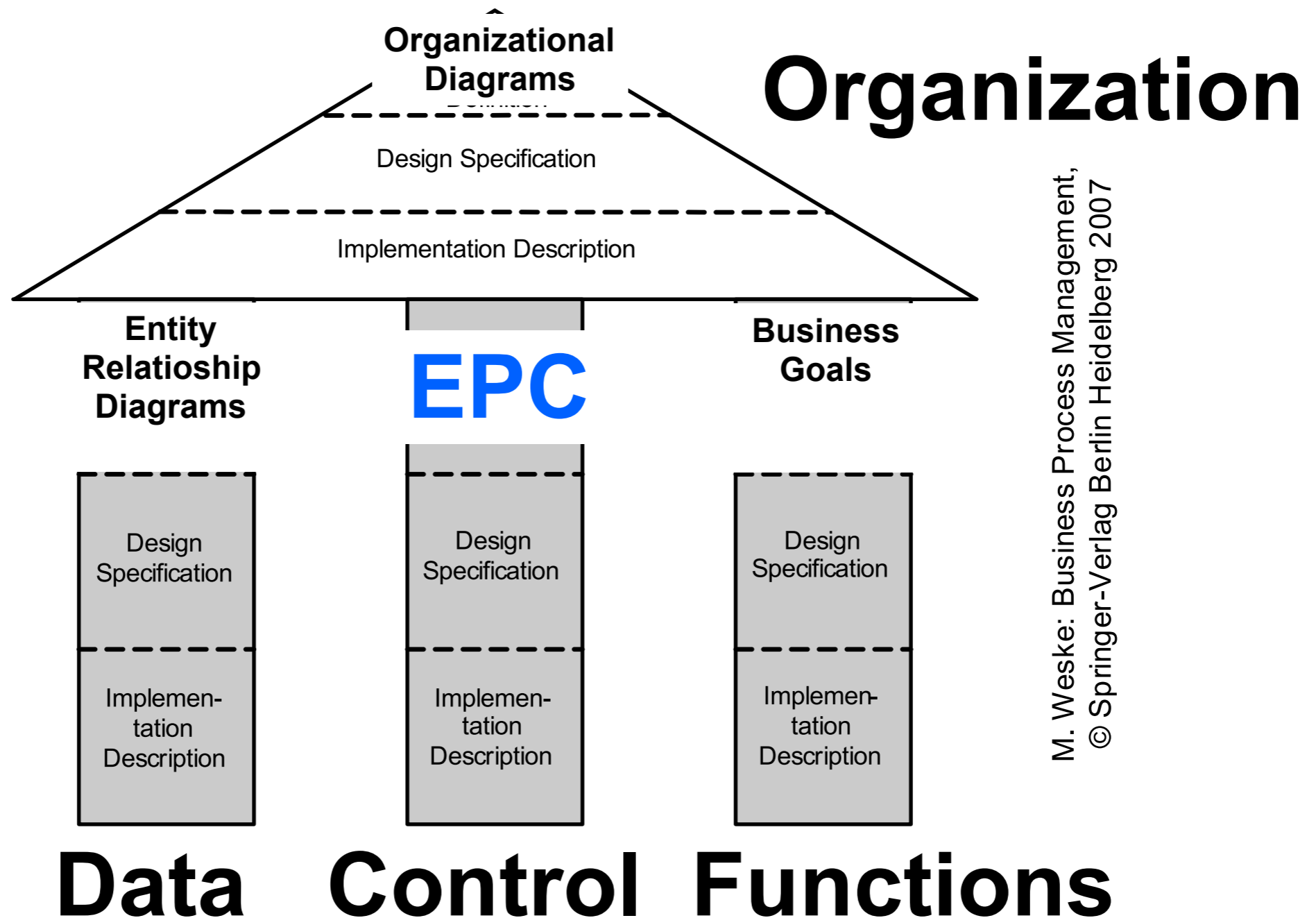


...and three levels of abstraction each



M. Weske: Business Process Management,
© Springer-Verlag Berlin Heidelberg 2007

...and three levels of abstraction each



M. Weske: Business Process Management,
© Springer-Verlag Berlin Heidelberg 2007

EPC informally

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

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

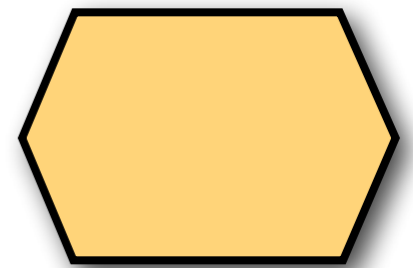
The flow is specified by logical operators
AND, XOR, OR

Events

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



Functions

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



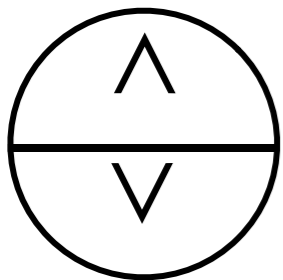
Logical connectors

Any EPC diagram may involve several **connectors**

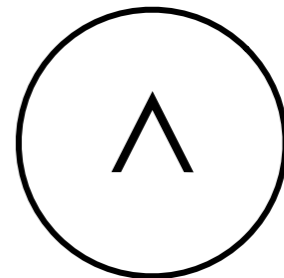
Elements used to describe the logical relationships between elements in the diagram

Branch, merge, fork, join

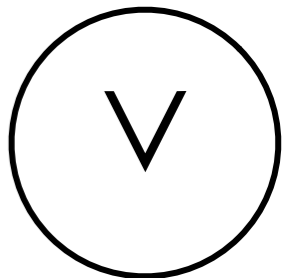
Graphical representation: circles (or also octagons)



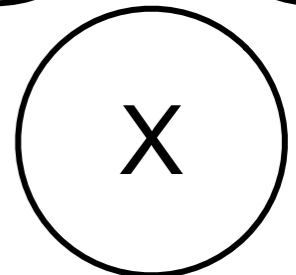
AND



OR



XOR



Control flow

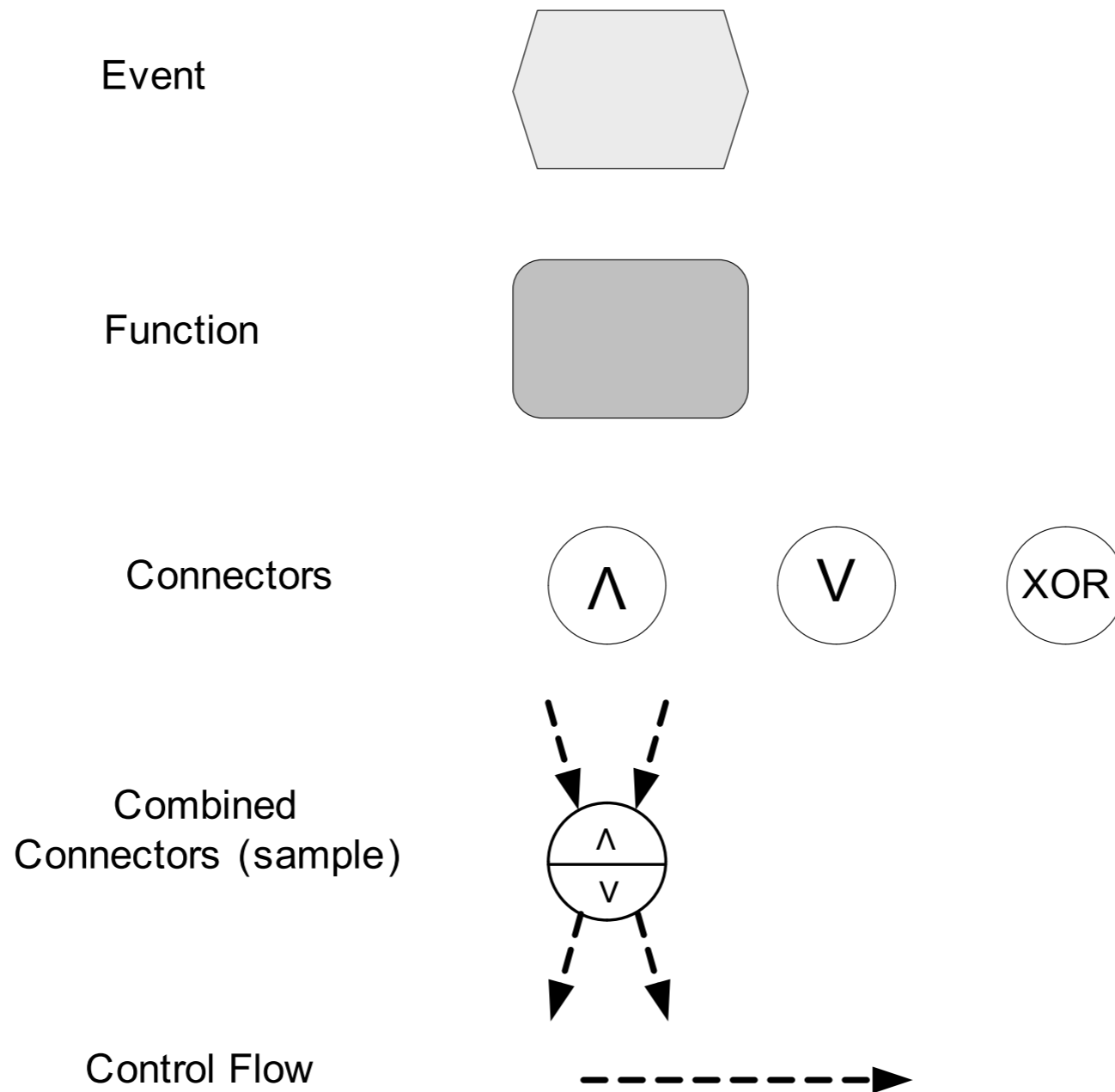
Any EPC diagram may involve several **control flow connections**

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

Graphical representation:
dashed arrows



EPC ingredients at a glance



M. Weske: Business Process Management,
© Springer-Verlag Berlin Heidelberg 2007

EPC 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

Unique start / end event

No arc between two events

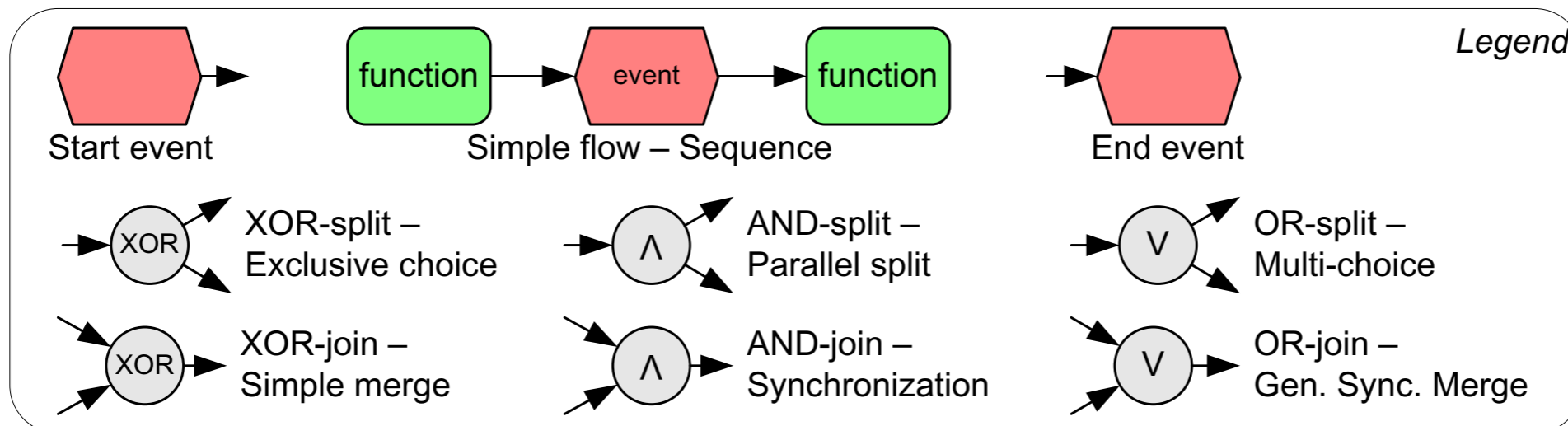
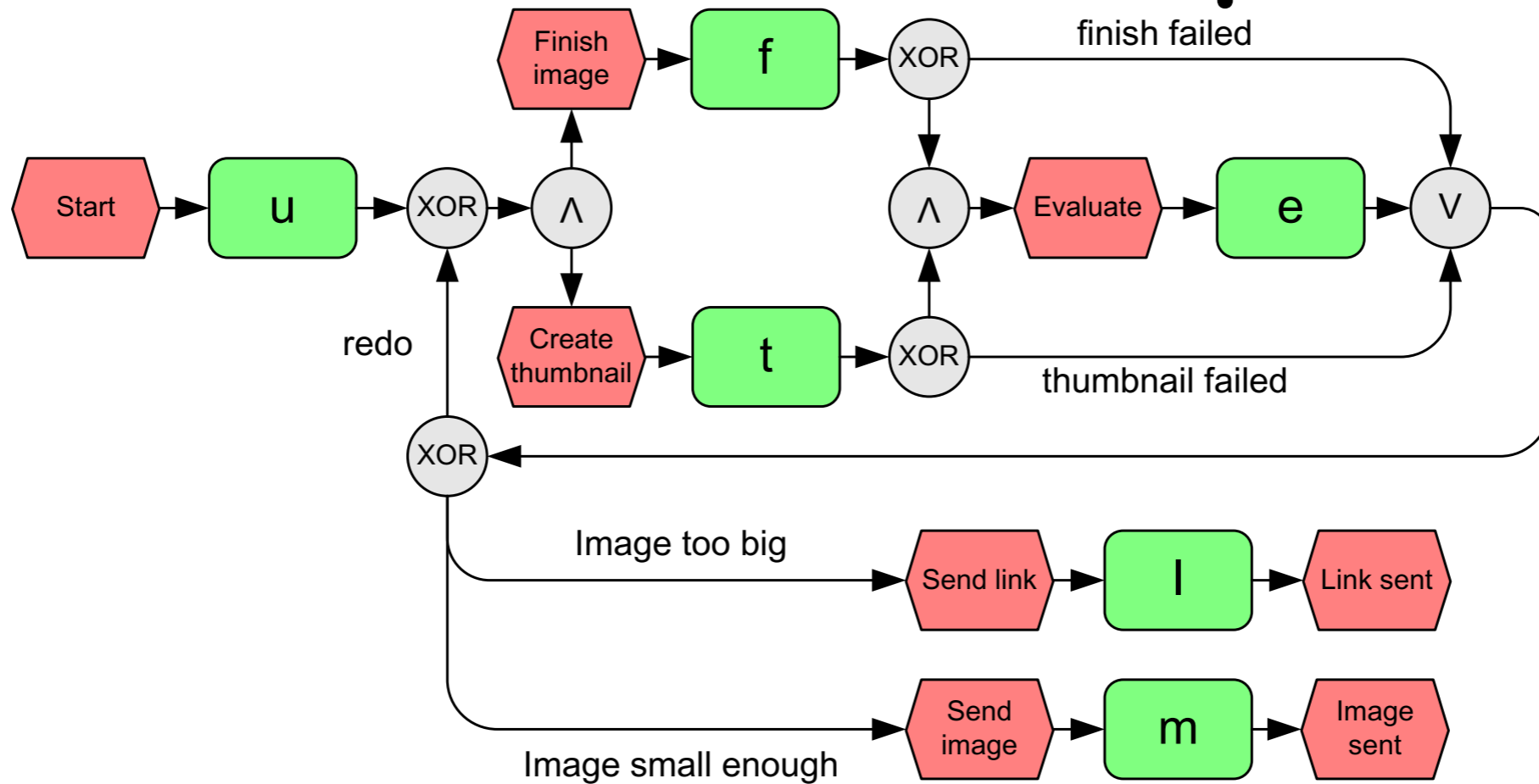
No arc between two functions

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

EPC allowed connections

	Event Connection		Activity Connection	
AND	<p>triggering Events</p>	<p>triggered Events</p>	<p>triggering Events</p>	<p>triggered Events</p>
OR			<p>not allowed</p>	
XOR			<p>not allowed</p>	

EPC an example



Other annotations for functions

Information, material, resource object:

represents objects in the real world

e.g. input data or output data for a function

(rectangles linked to function boxes)

Organization unit:

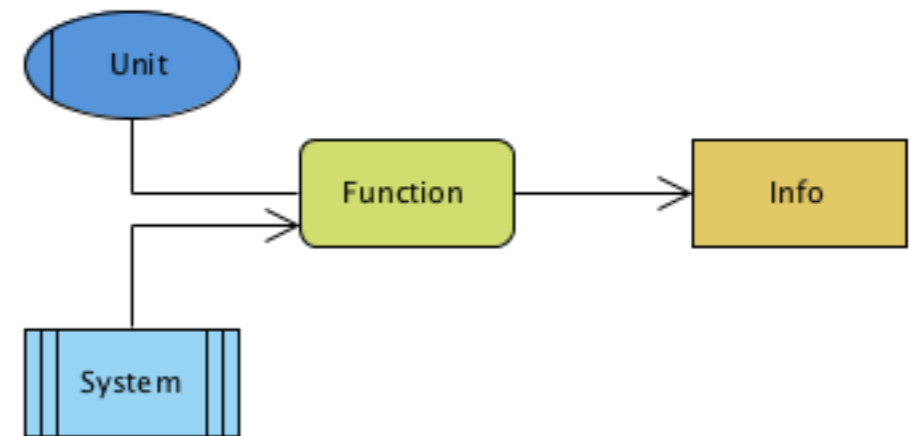
determines the person or organization

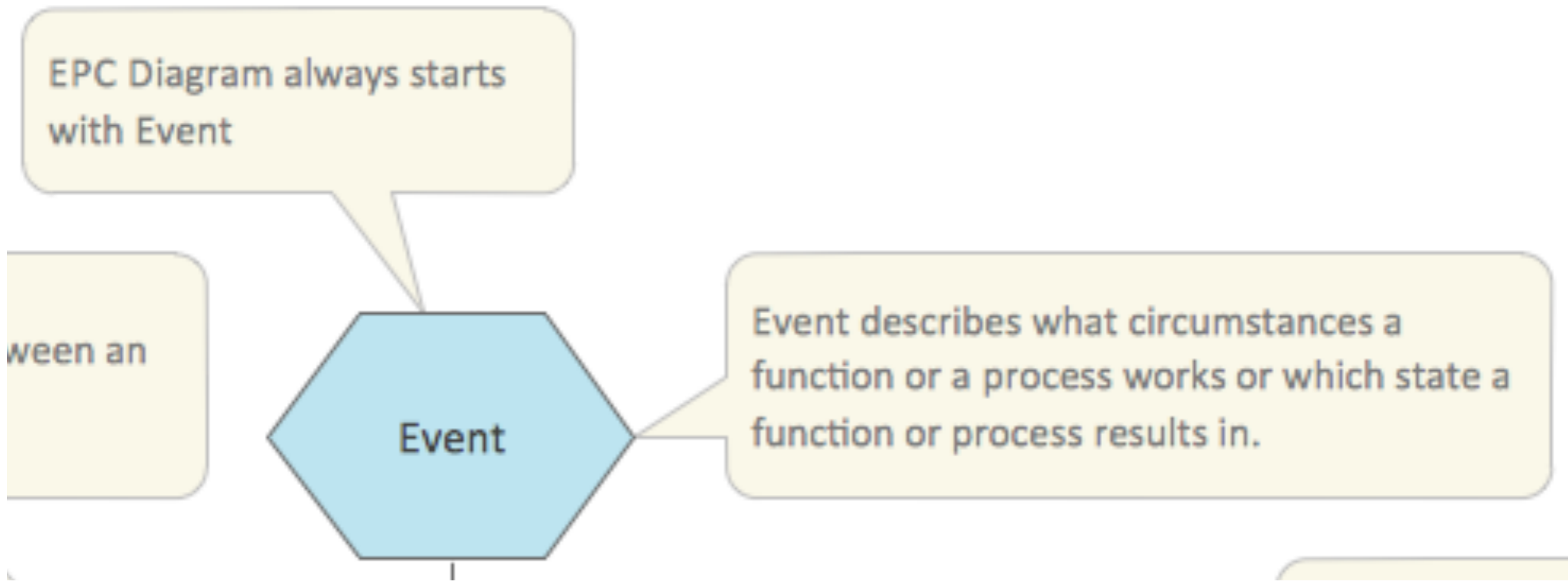
responsible for a specific function

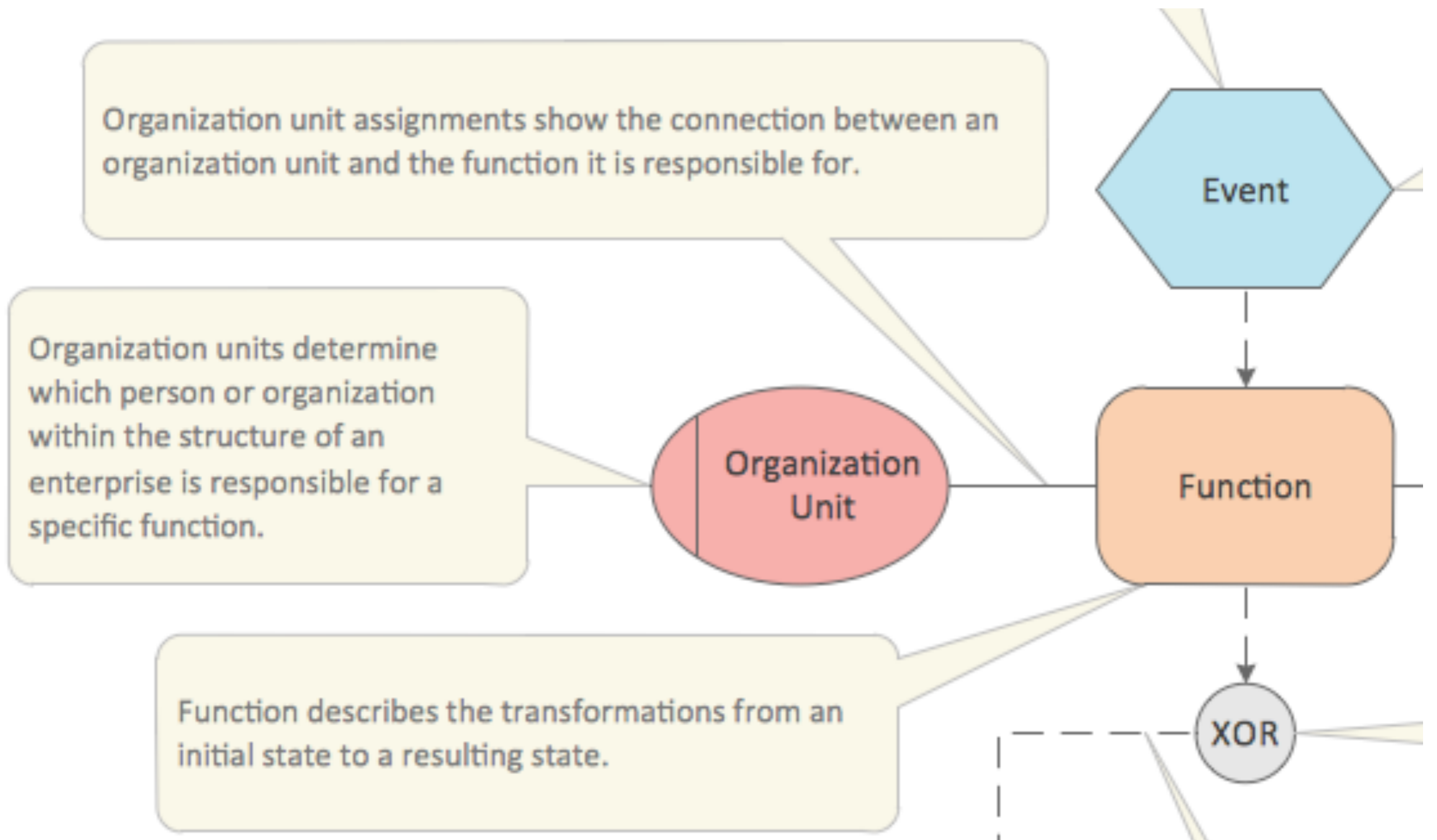
(ellipses with a vertical line)

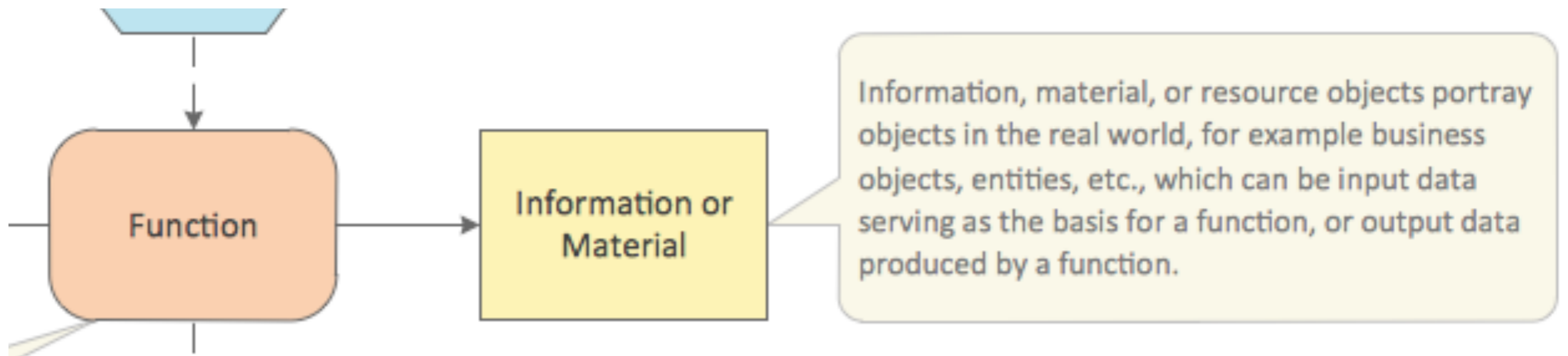
Supporting system: technical support

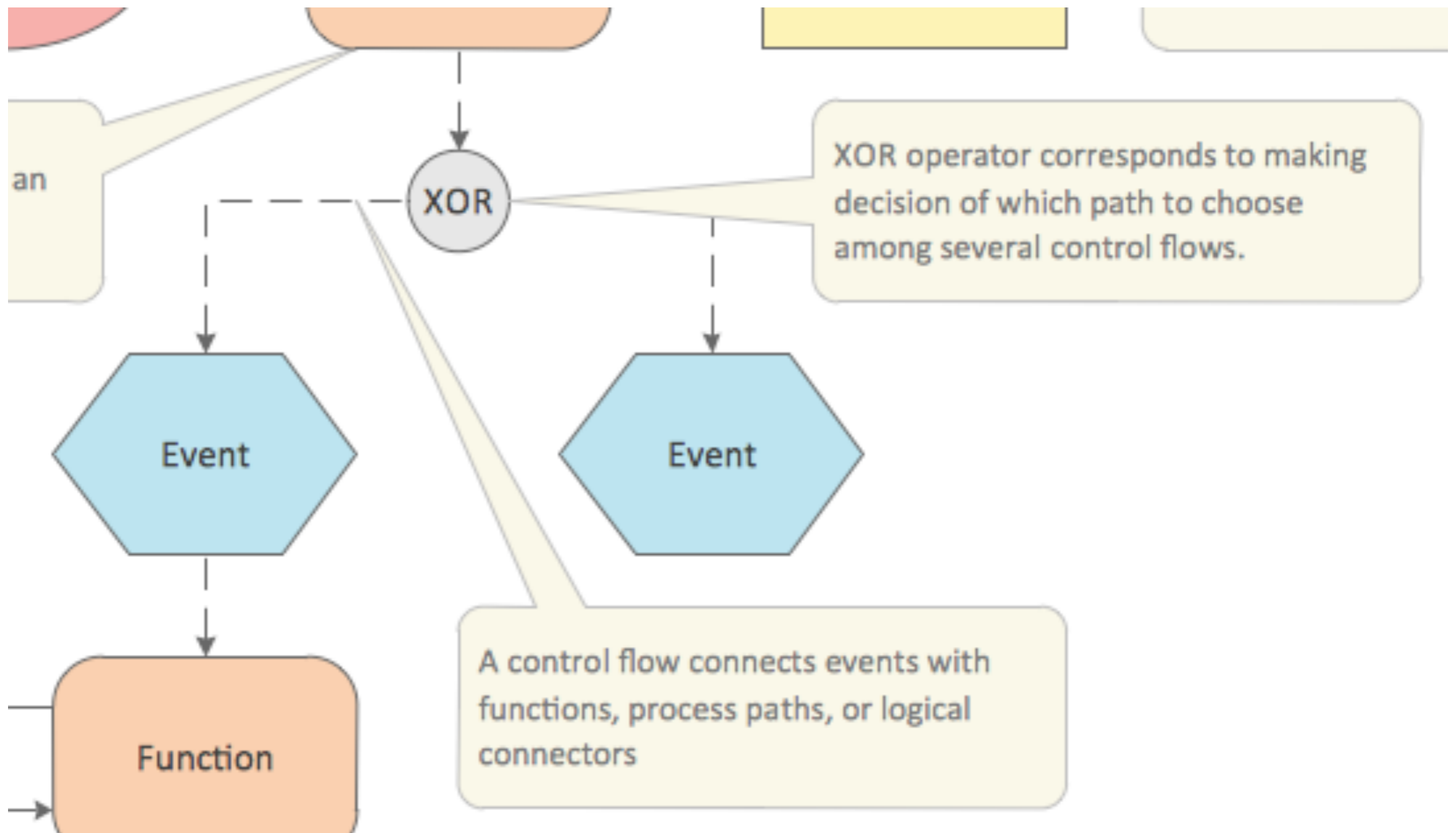
(rectangles with vertical lines on its sides)

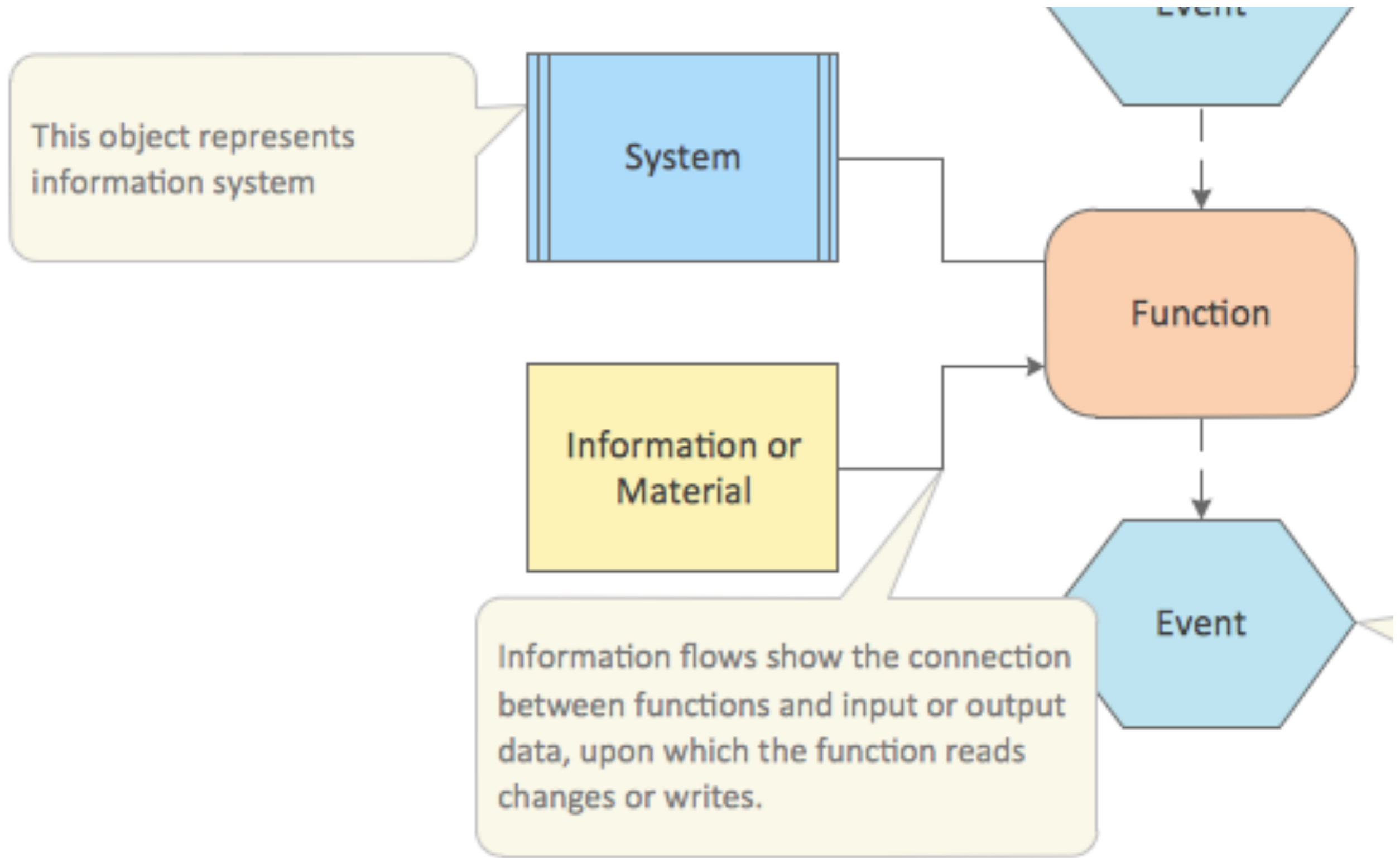


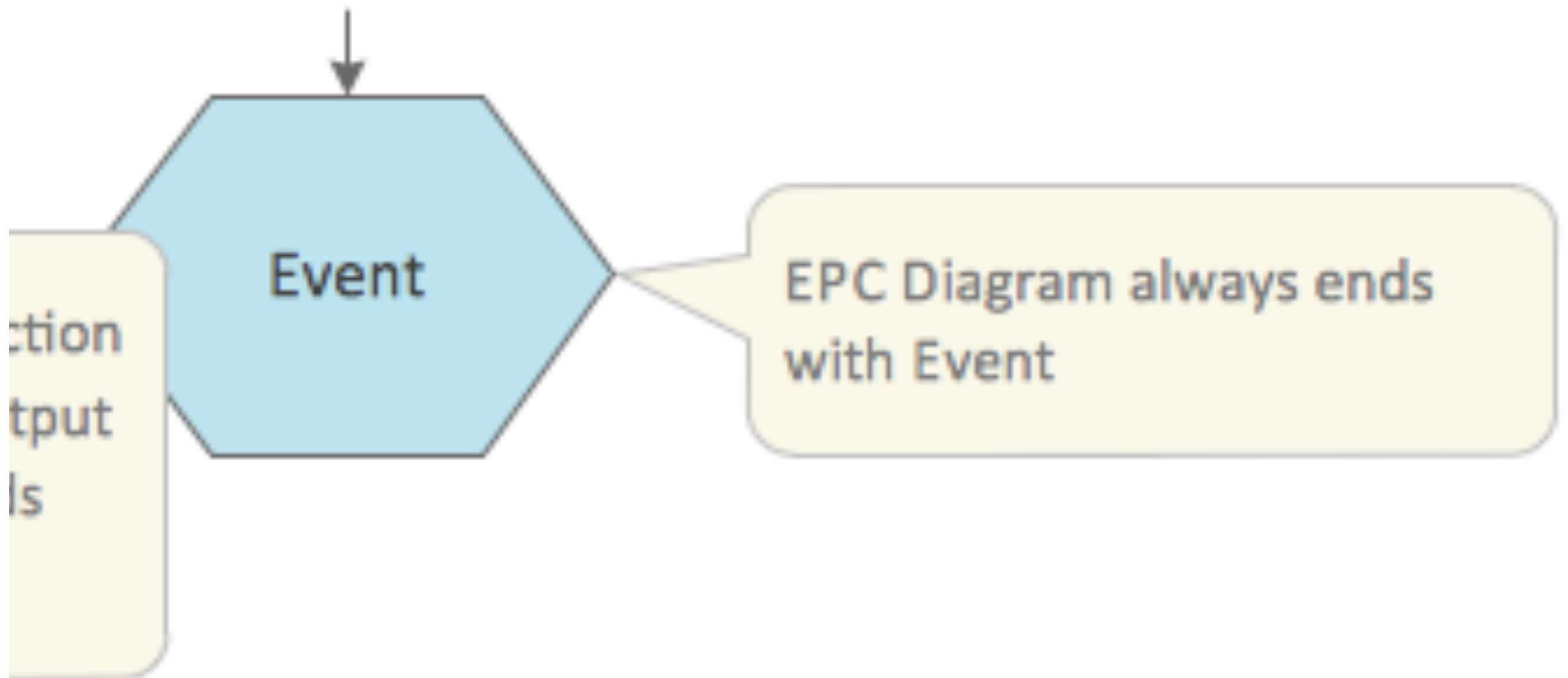


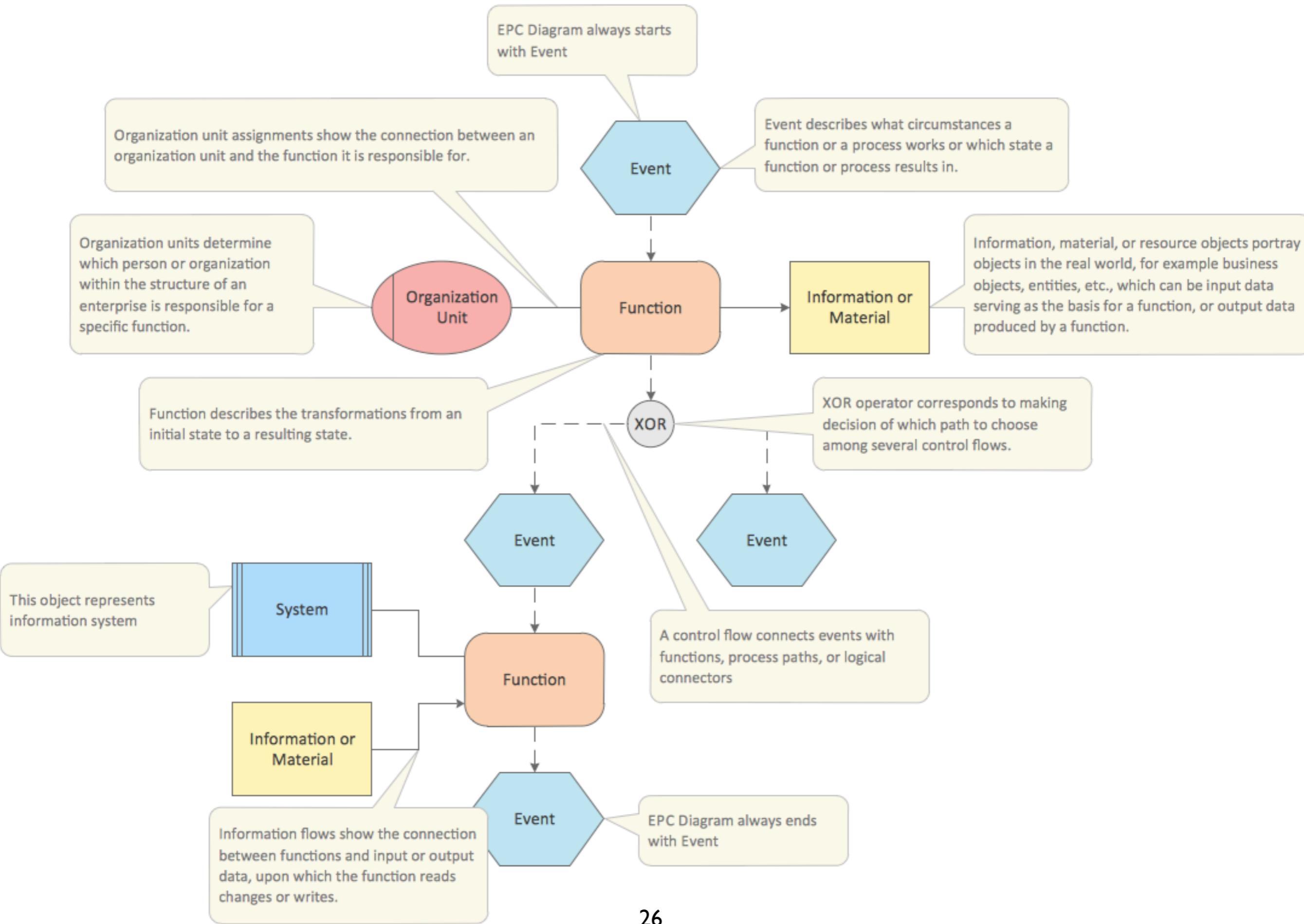












EPC intuitive semantics

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”

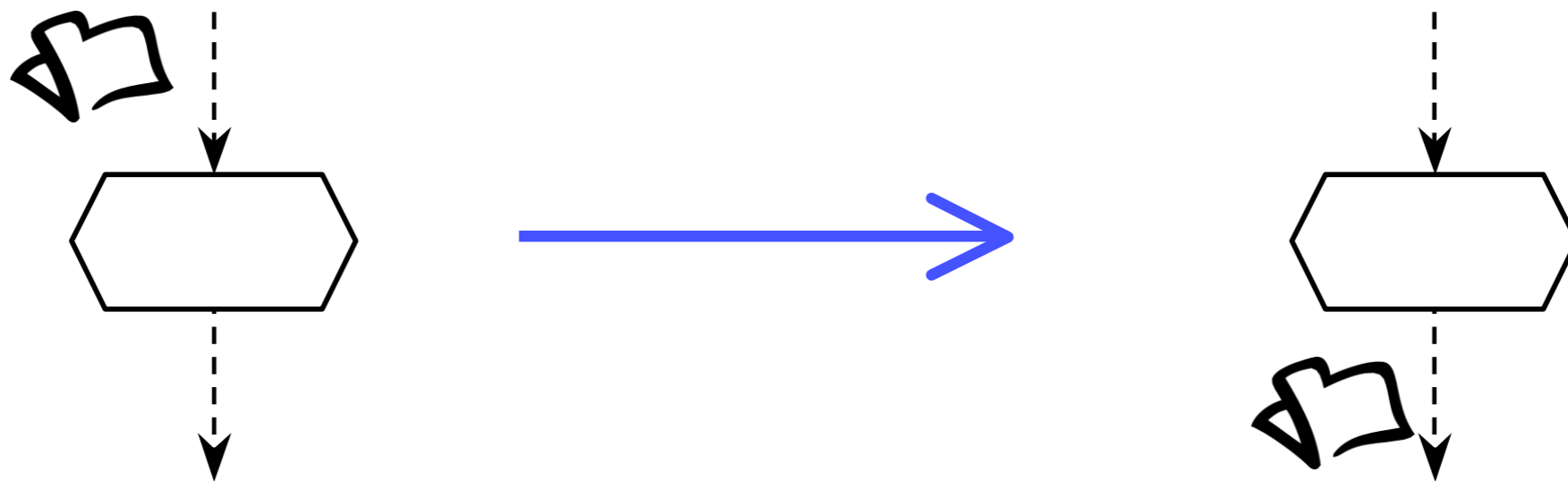
Folder-passing semantics

Semantics

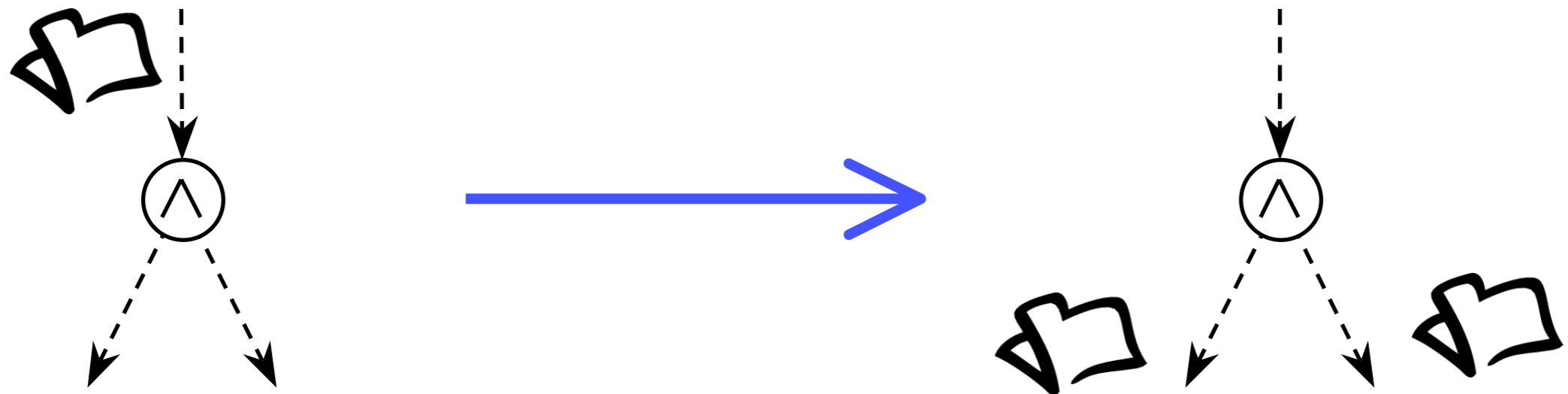
- State: Process folders
- Transition relation:
Propagation of process folders



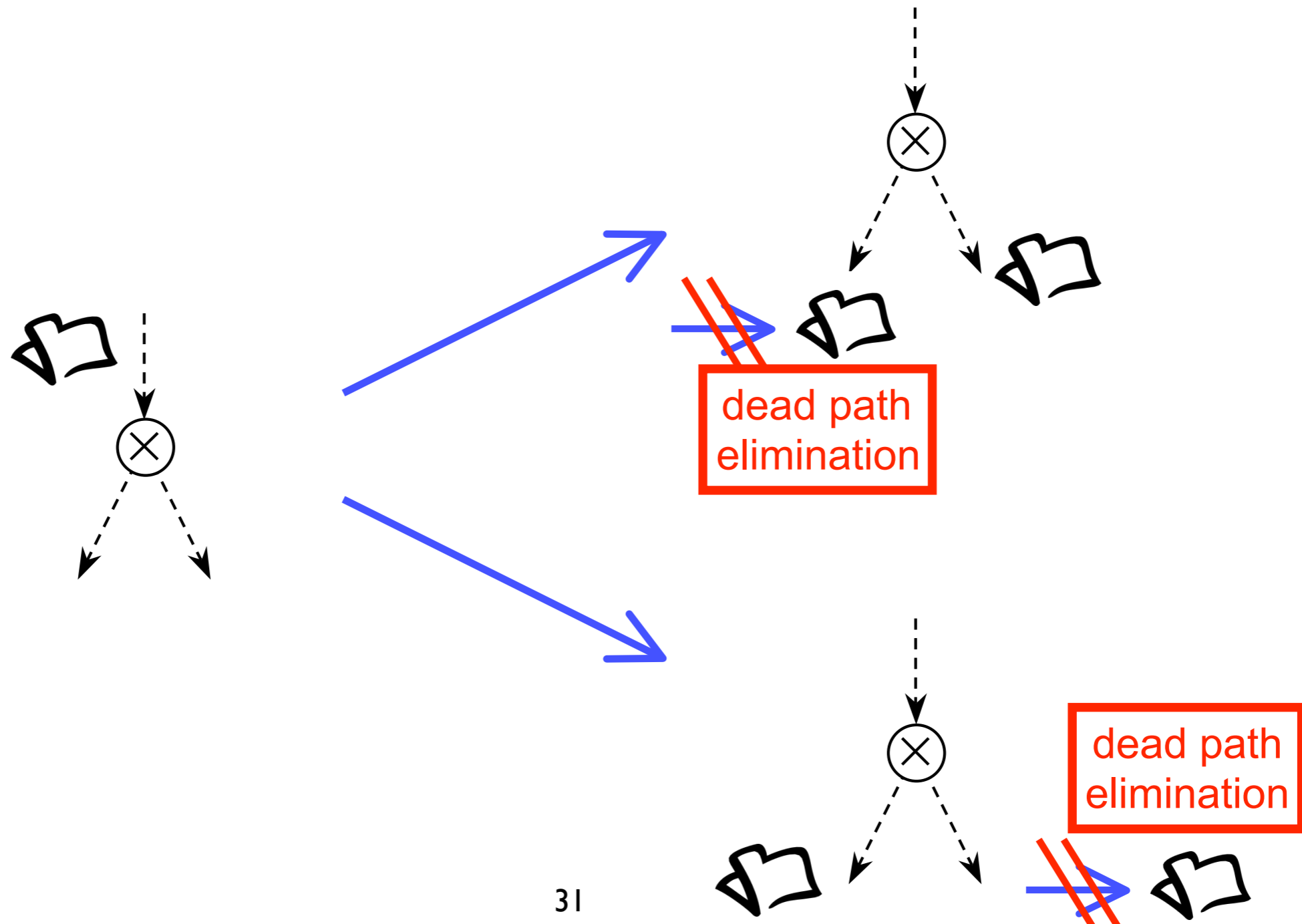
Folder-passing semantics: events



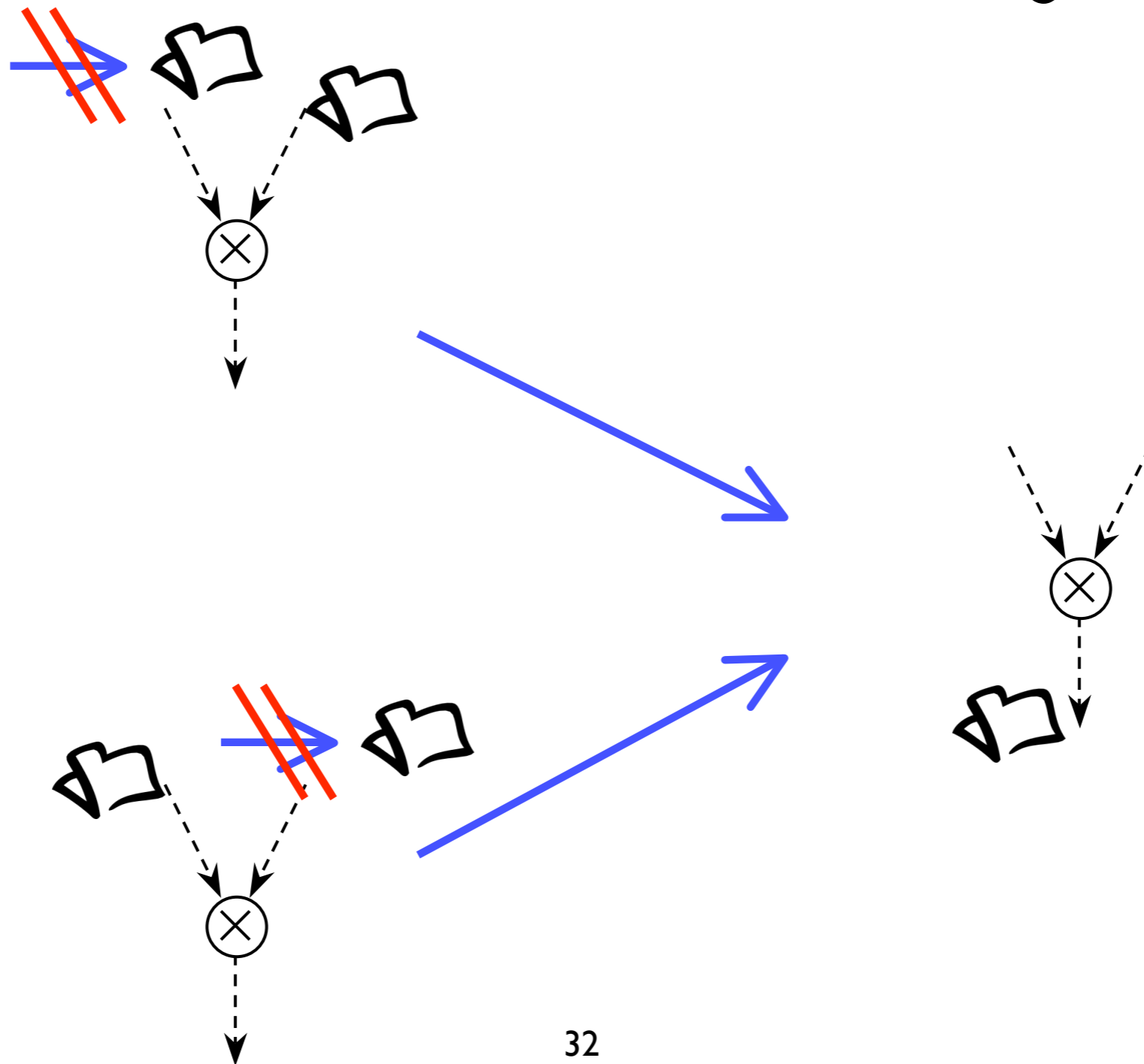
Folder-passing semantics: AND-split



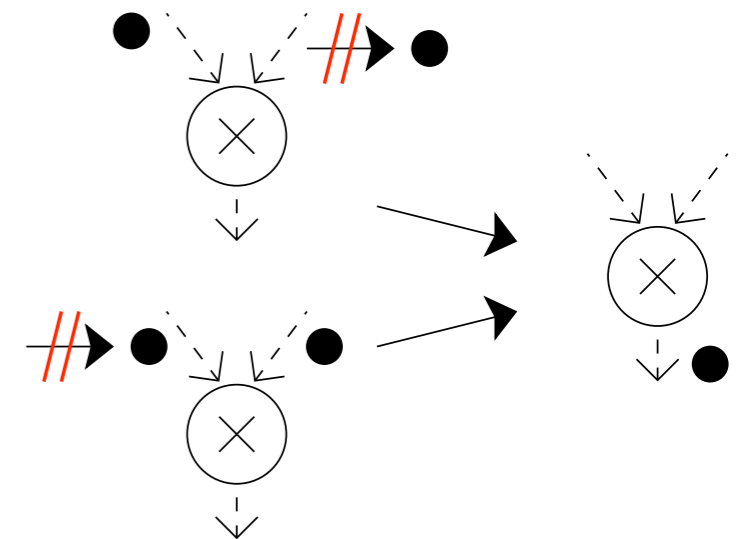
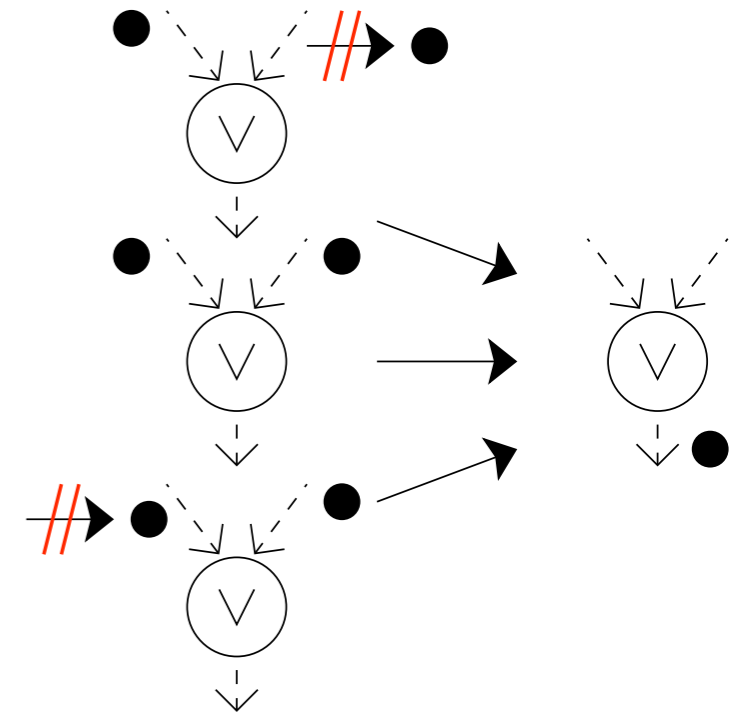
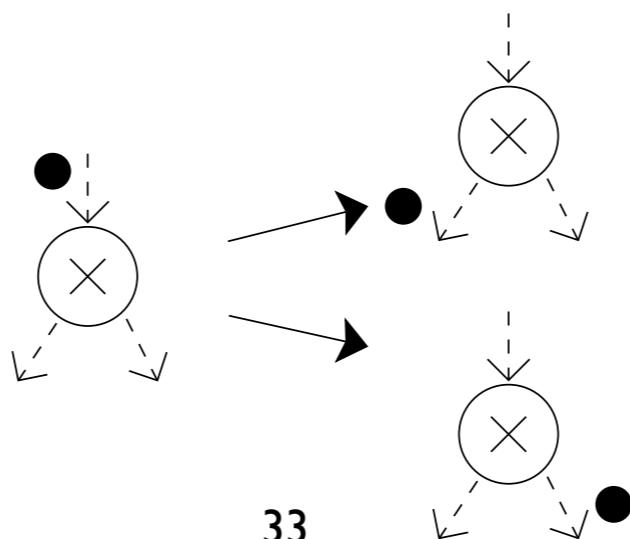
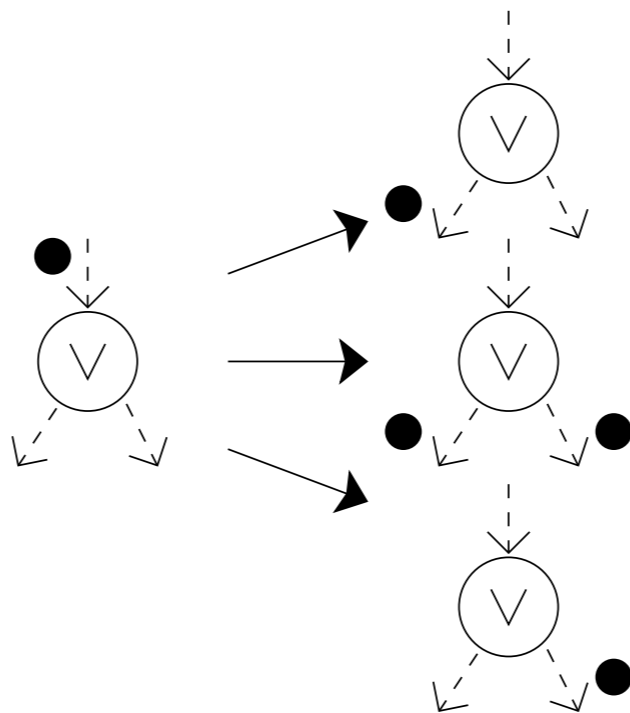
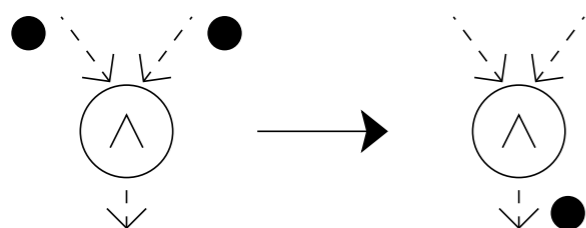
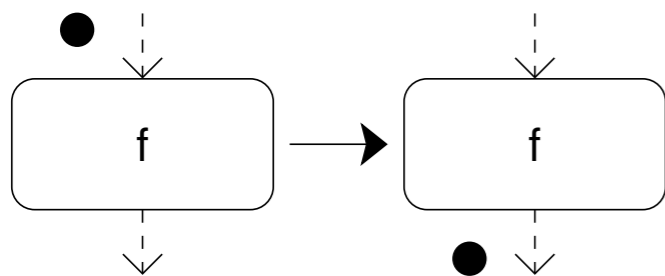
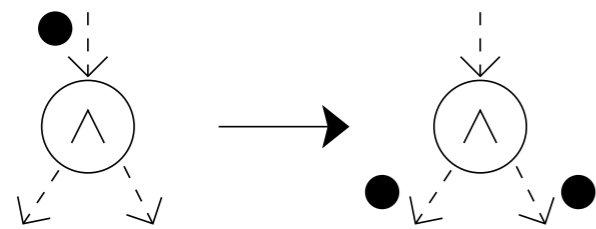
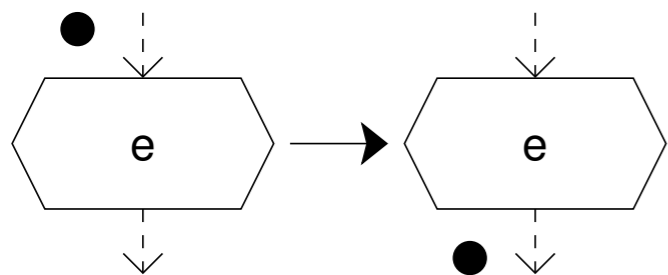
Folder-passing semantics: XOR-split



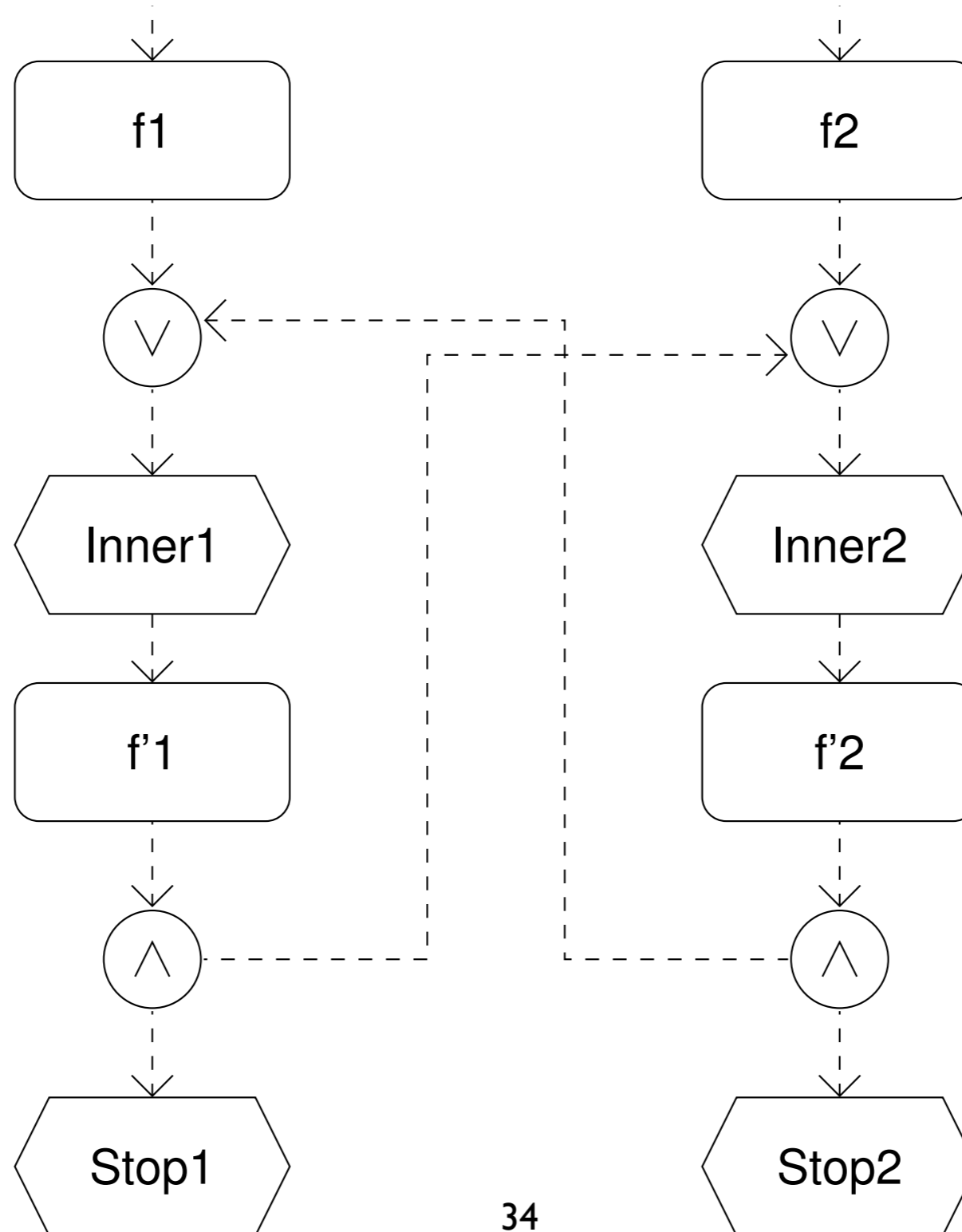
Folder-passing semantics: XOR-join



Folder-semantics in one slide



A vicious circle?



EPC semantics?

Little unanimity around the EPC semantics

Rough verbal description
in the original publication by Scheer (1992)

Later, several attempts to define formal semantics
(assigning 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 between 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

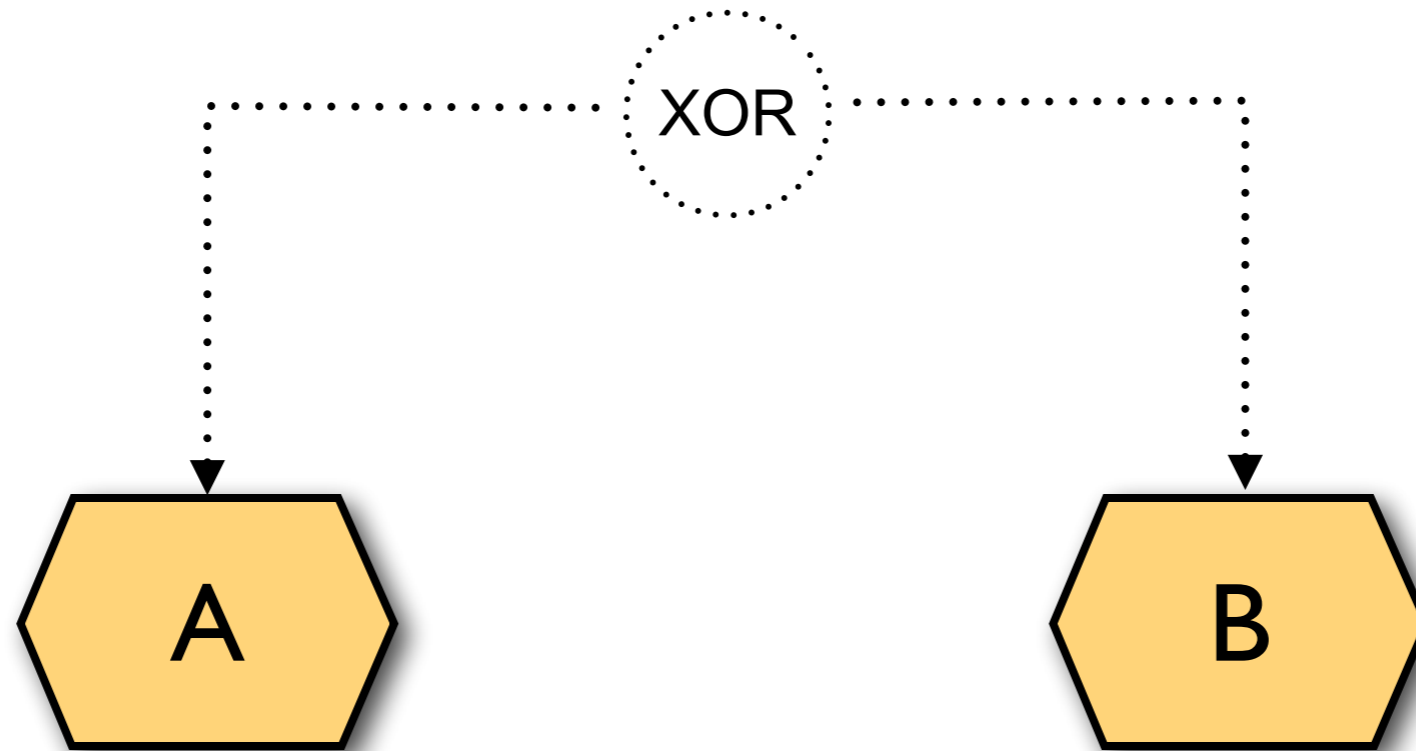
What if multiple start events occur?

Solution:

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

Problem with start events: solution

hypothetical / implicit split



Problem with alternation

From empirical studies:

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

Solution:

It is safe to drop the requirement about alternation
(dummy events might always be added later)

Every join has a split

observation:

Every join has at least one **corresponding** split
(i.e. a split for which there is a path
from either output to the input of the join)

proof sketch:

we trace backward the paths

leading to the join from start events;

if the start events coincide there is a split node in the path;

if start events differ, the candidate split is the implicit XOR

Problem with corresponding splits

The semantics of a join often depends on the nature of the corresponding split

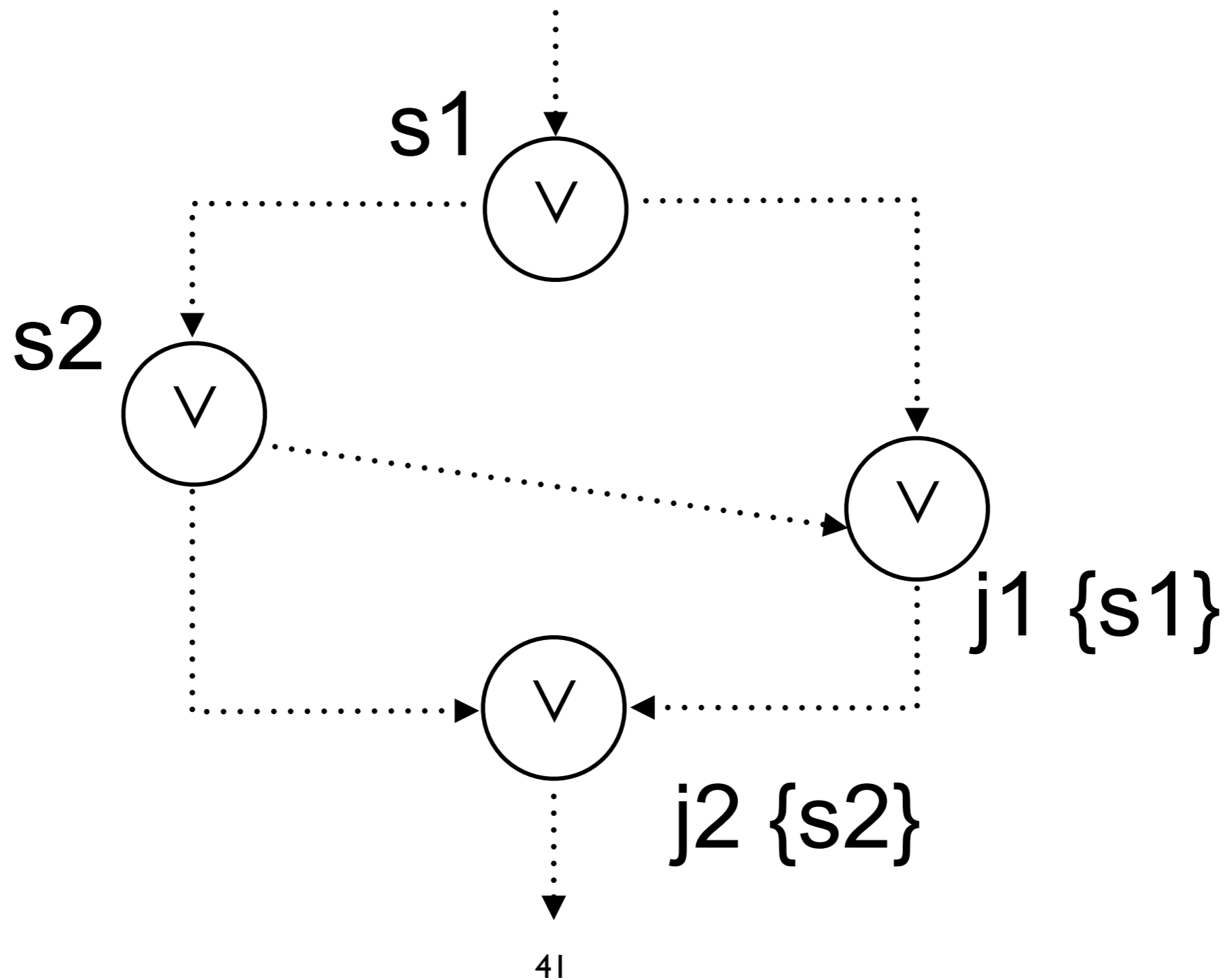
But:

- 1) there can be **more candidates** to corresponding split
- 2) 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

Tagging 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 policy must be applied:

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

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

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 single 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)

Sound EPC diagrams

We transform EPC diagrams to Workflow nets:
the EPC diagram is sound if its net is so

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

Translation of EPC to Petri nets

A note about the transformation

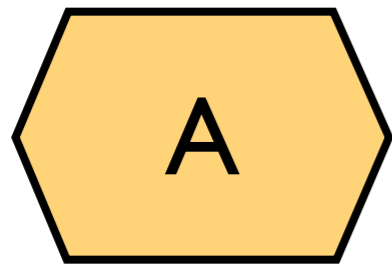
We first transform each event, function and connector separately in small net fragments

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 different translations, depending on whether joins are decorated or not

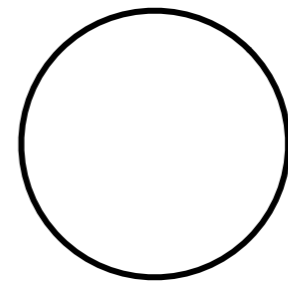
First attempt
(decorated EPC)

EPC



event

Petri net



place

EPC



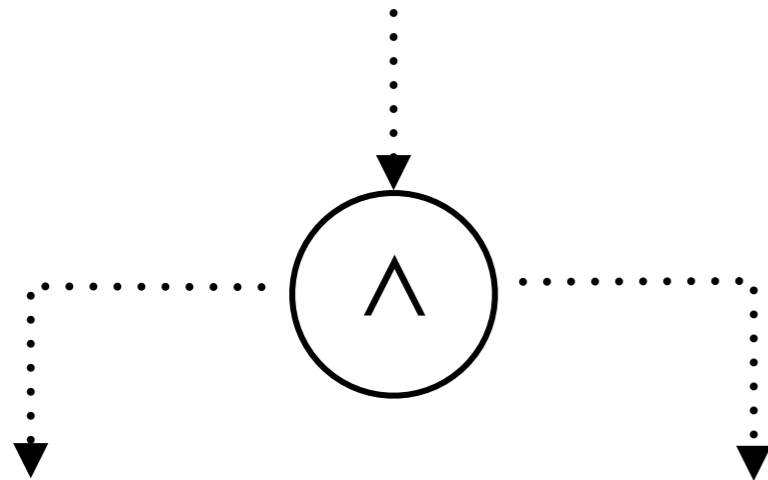
function

Petri net



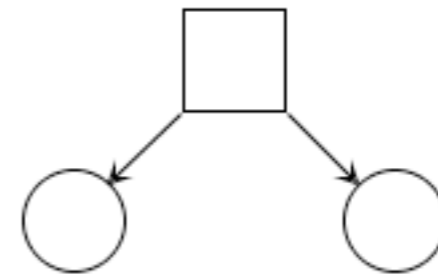
transition

EPC



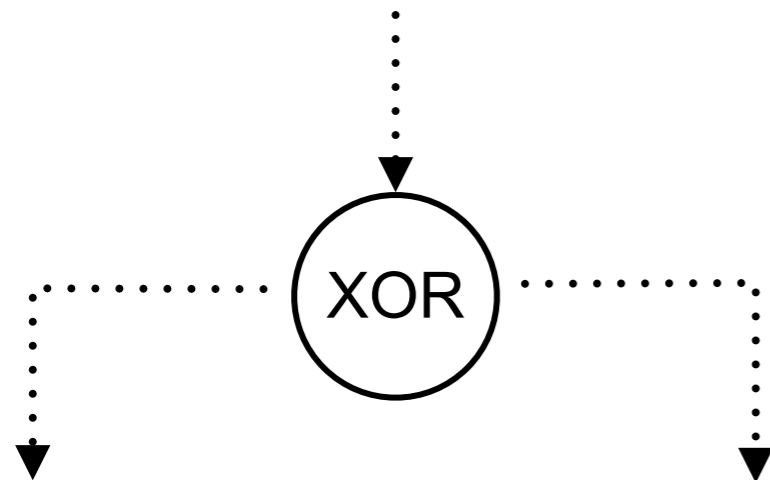
AND split

Petri net



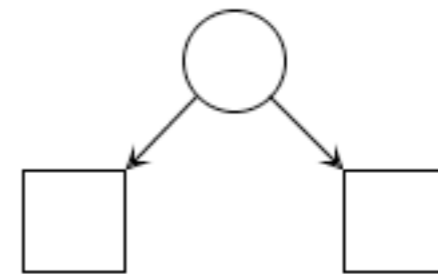
net

EPC



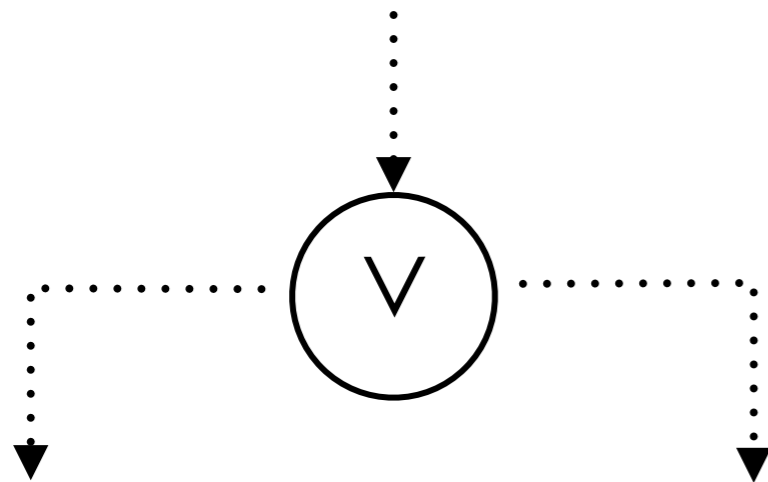
XOR split

Petri net



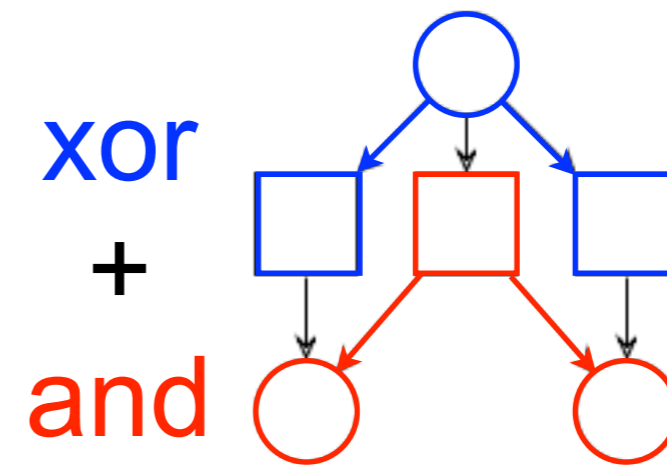
net

EPC



OR split

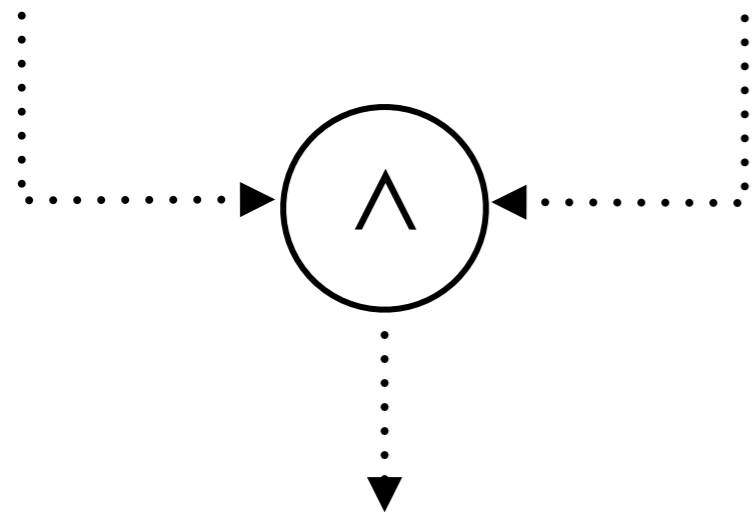
Petri net



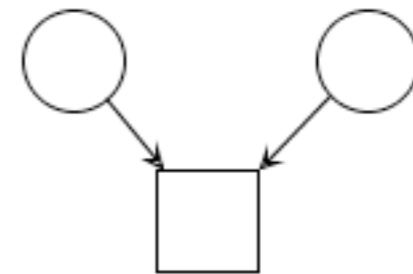
net

EPC

Petri net



AND join

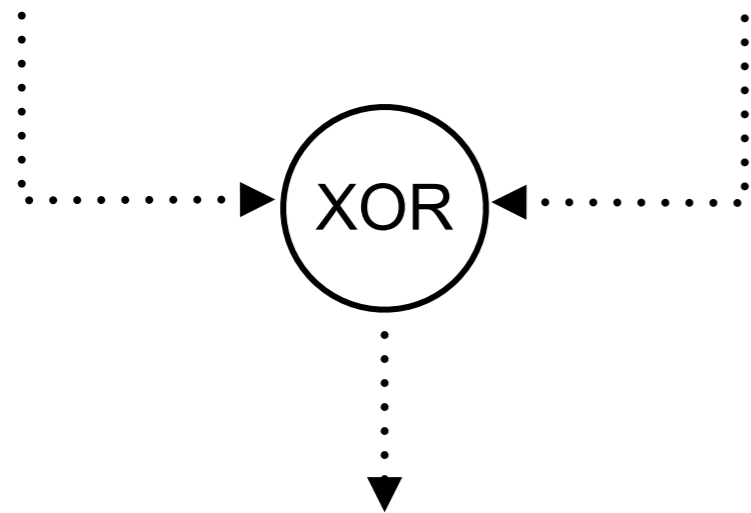


net

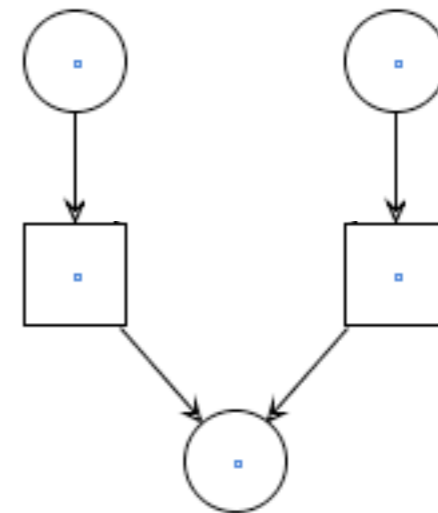
EPC

Petri net

corresponding
split



XOR join

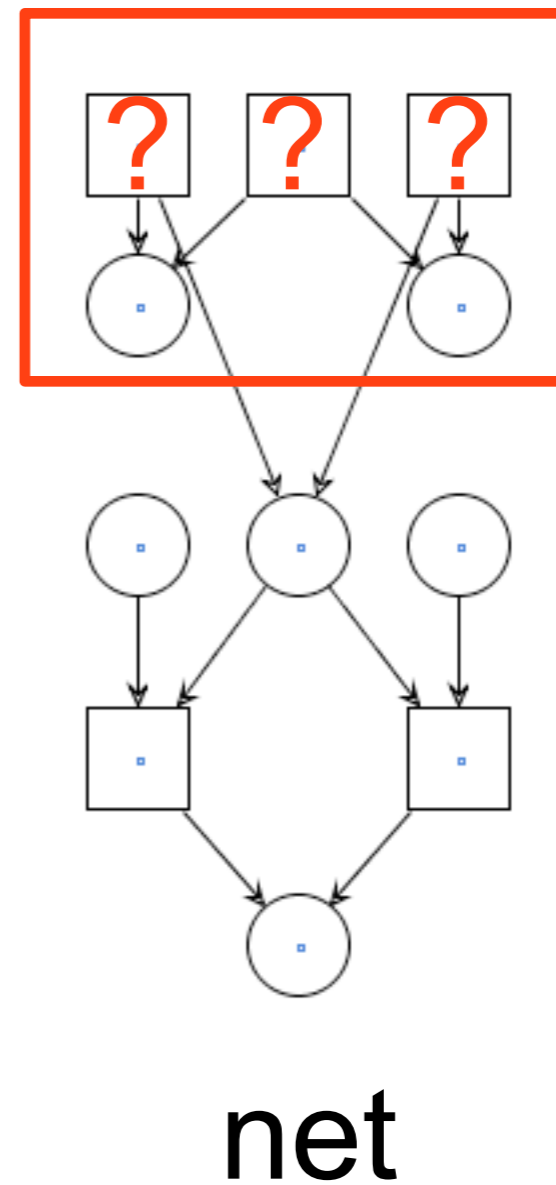
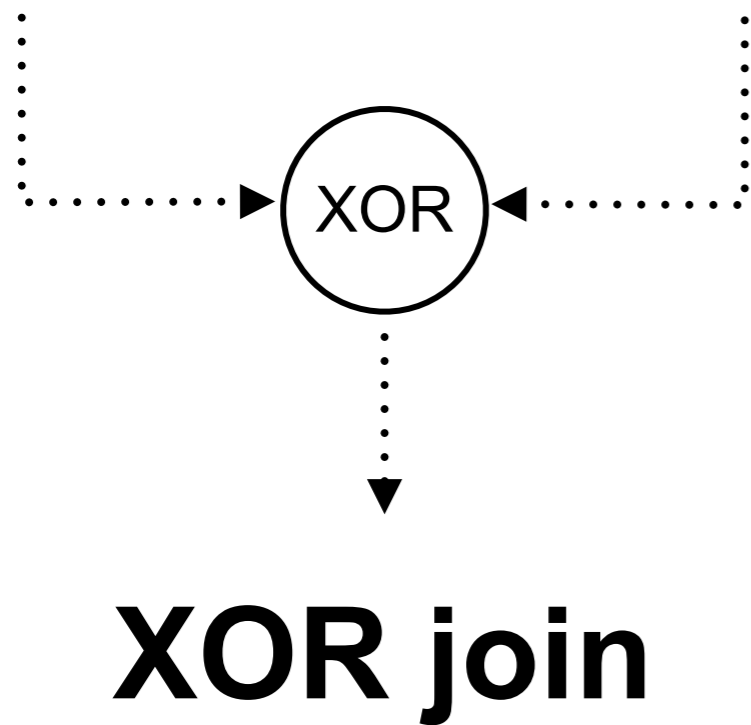


net

EPC

Petri net

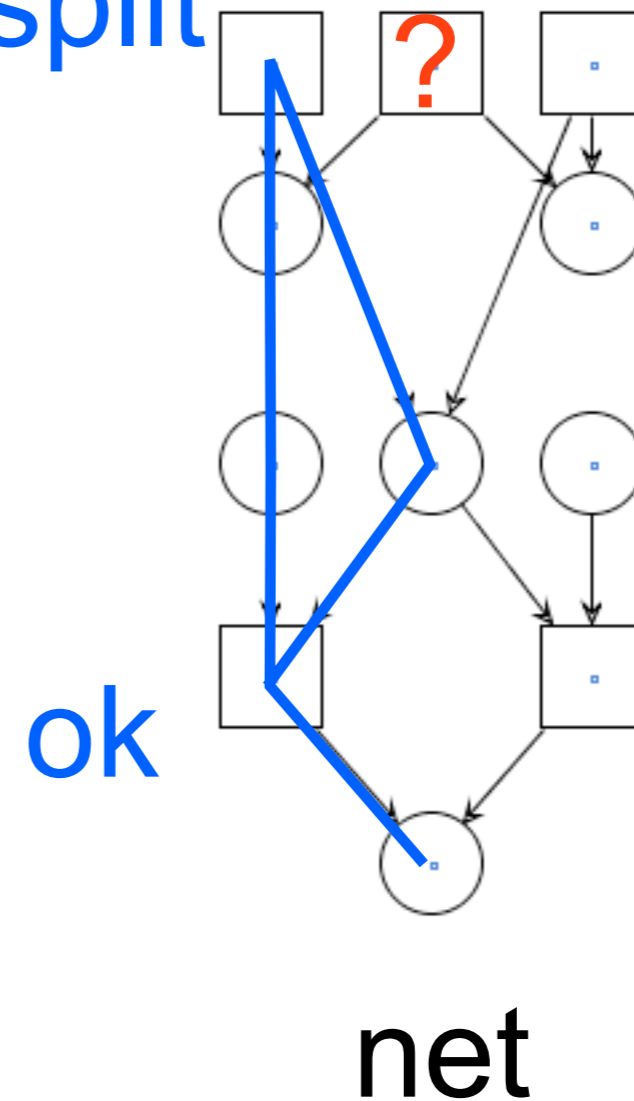
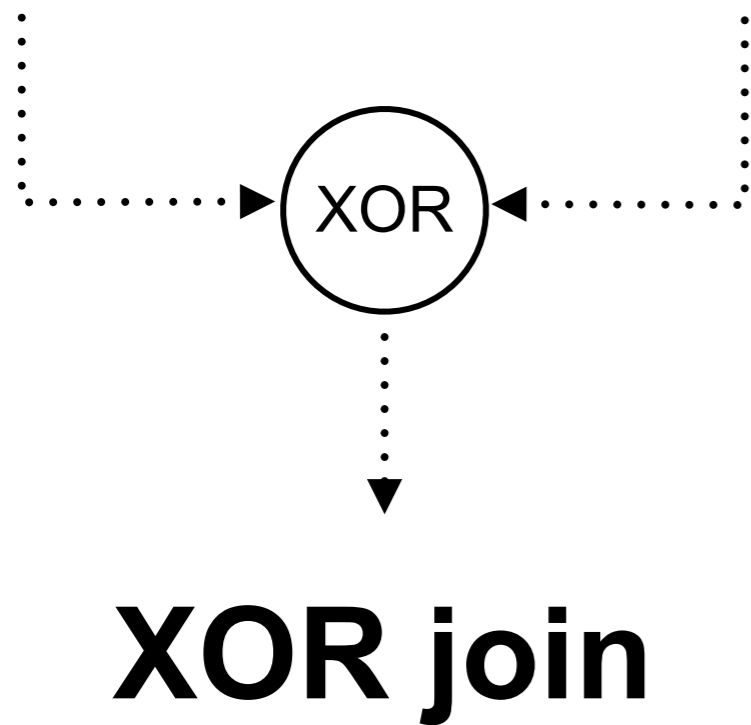
most general
case



EPC

Petri net

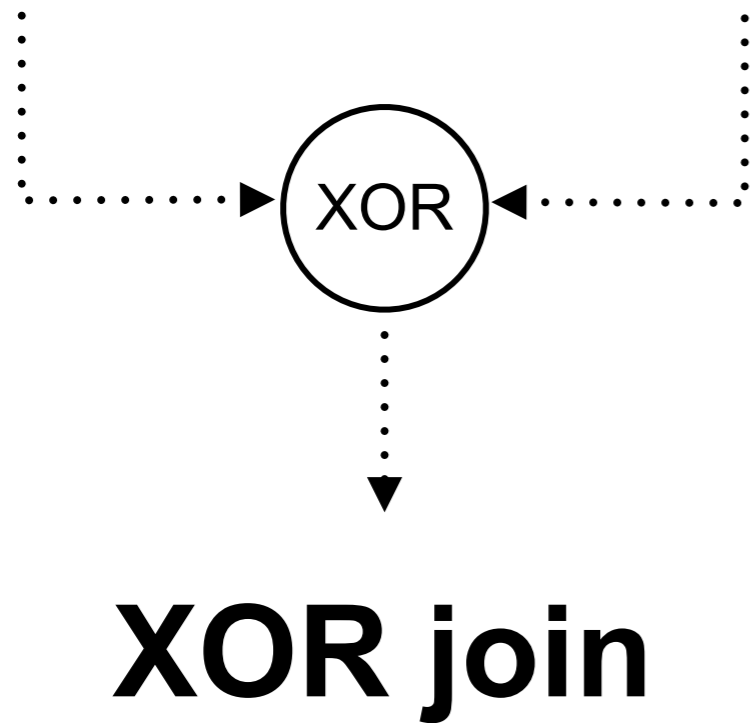
corresponding
XOR/OR split



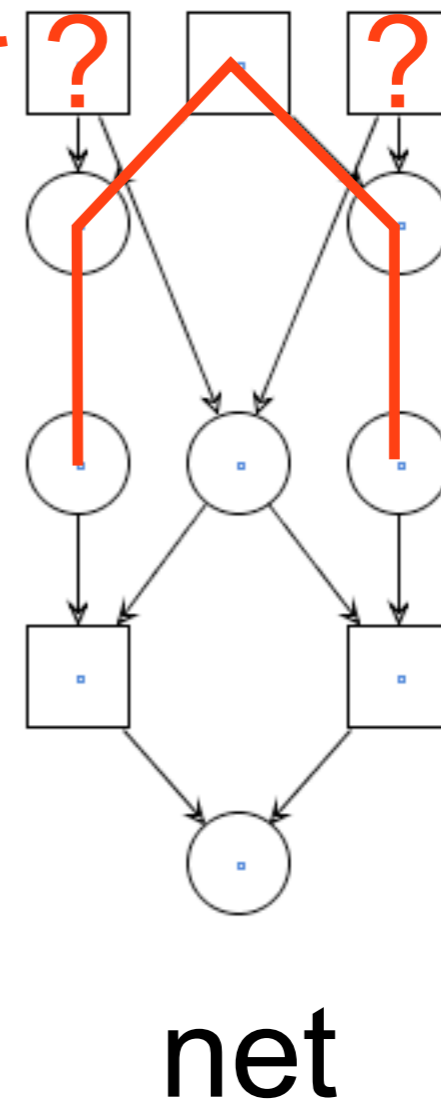
EPC

Petri net

corresponding
AND/OR split



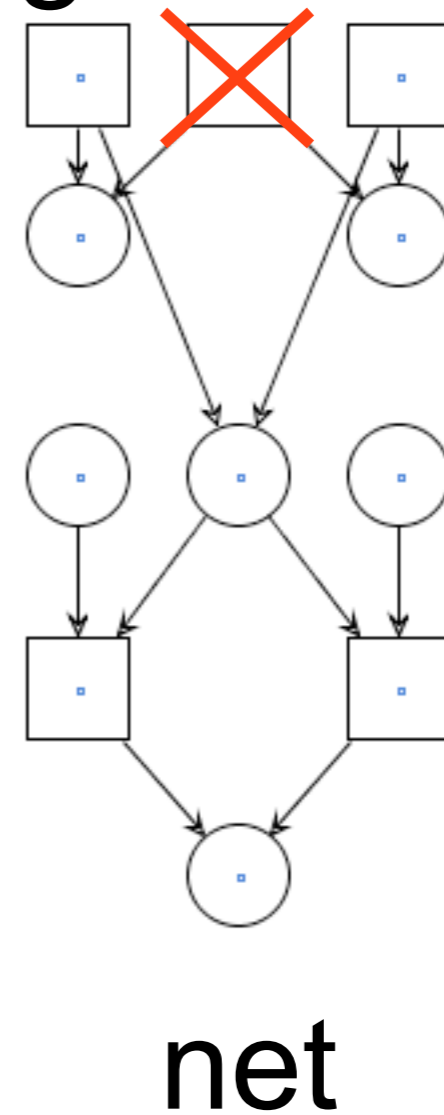
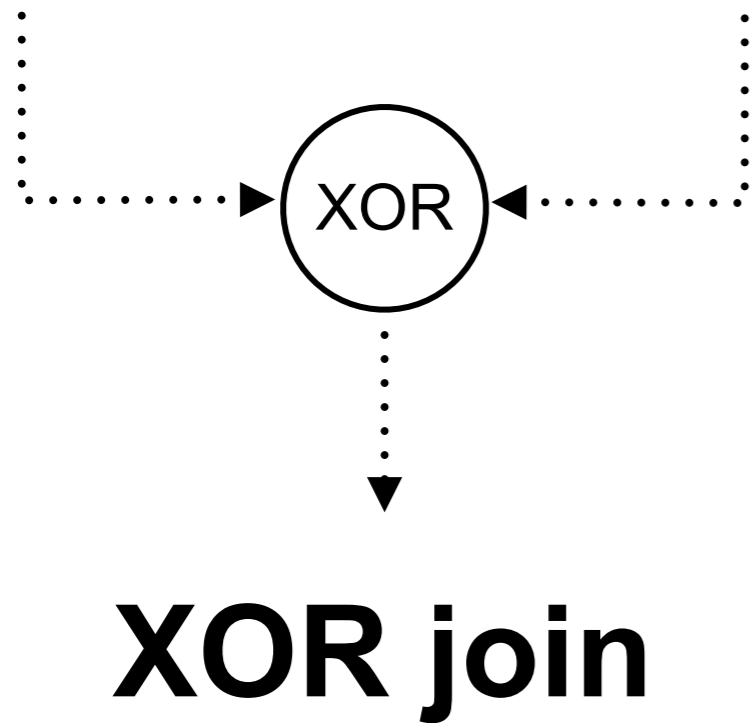
deadlock!



EPC

Petri net

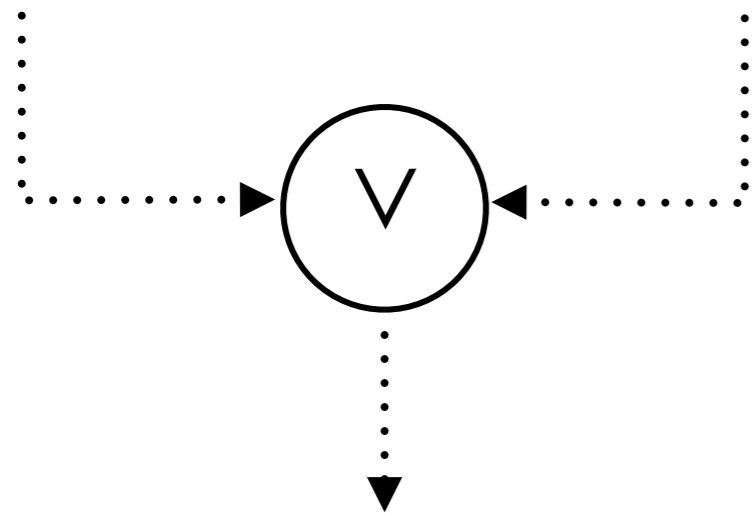
better to have
a corresponding
XOR split!



EPC

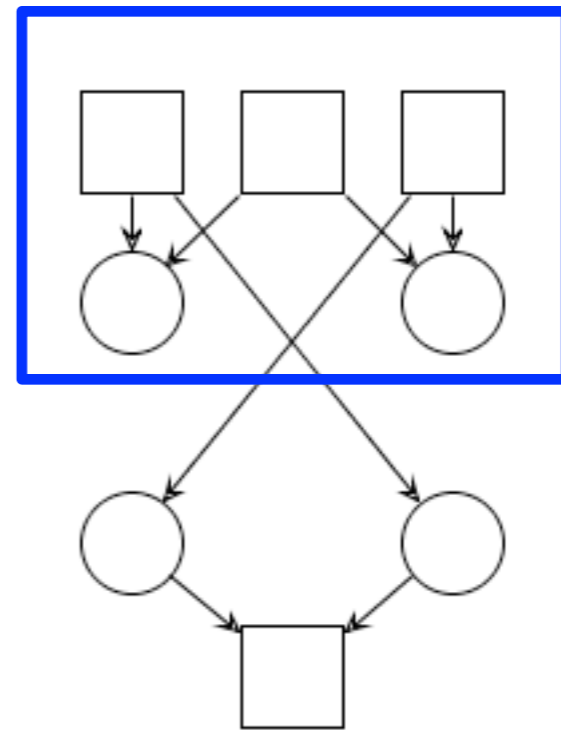
Petri net

matching
OR split



**OR join
with**

matched OR split

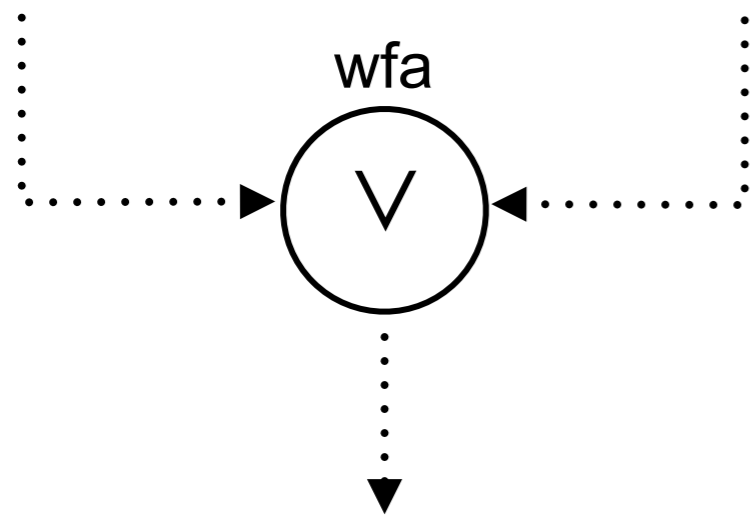


net

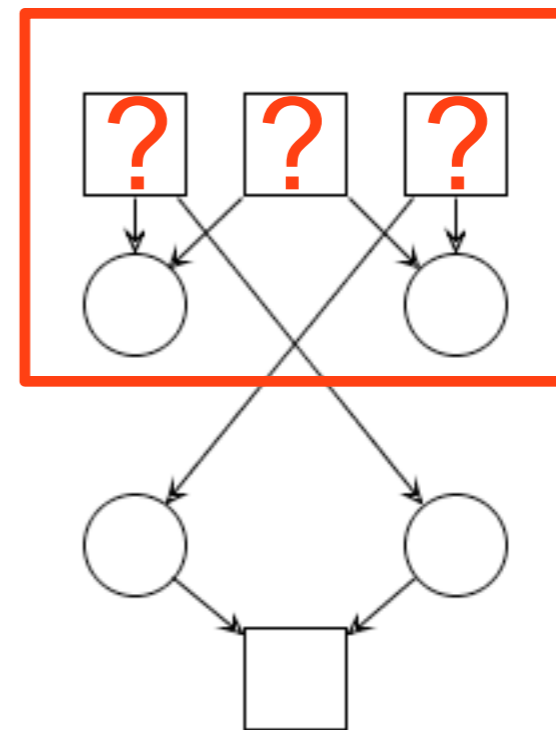
EPC

Petri net

mismatched corresponding split:
most general
case



OR join
wait-for-all
(mismatched)

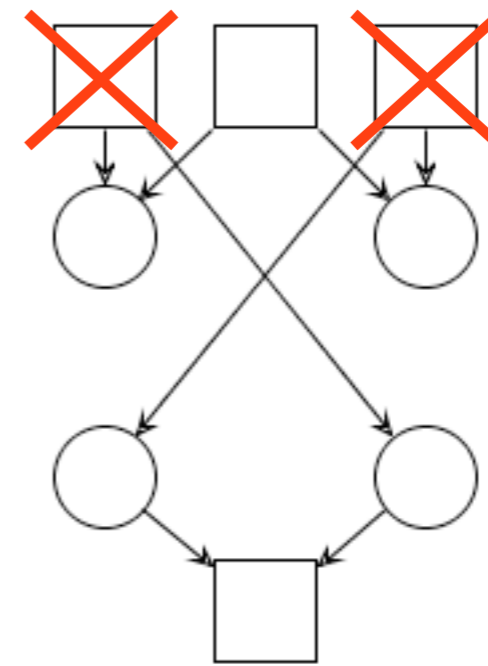
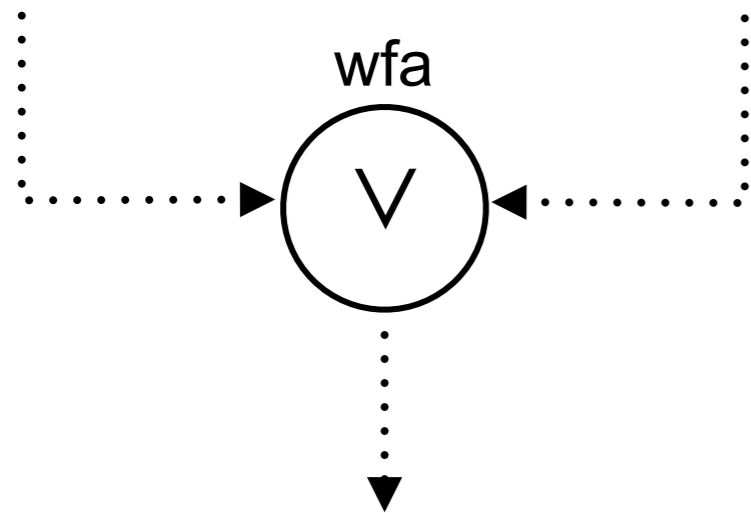


net

EPC

Petri net

corresponding
AND split



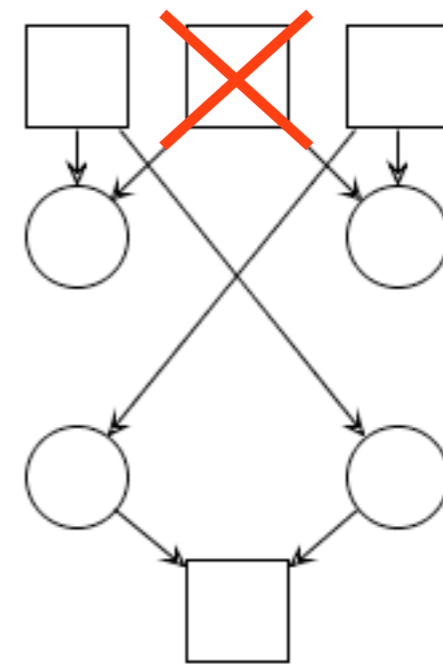
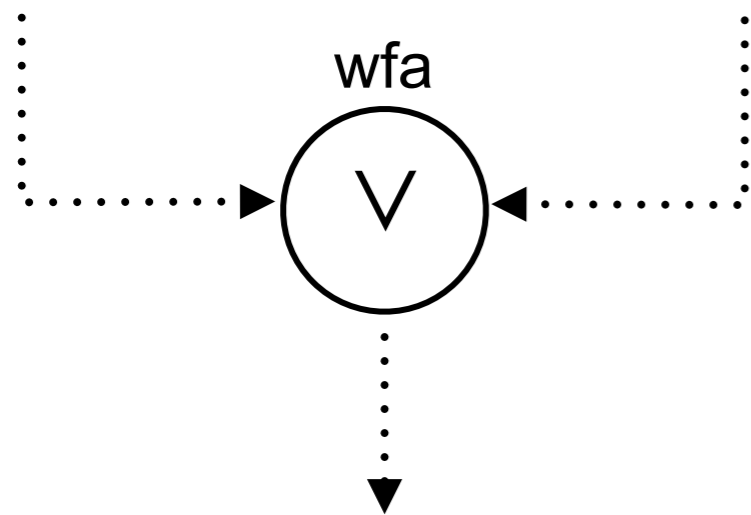
OR join
wait-for-all
(mismatched)

net

EPC

Petri net

corresponding
XOR split

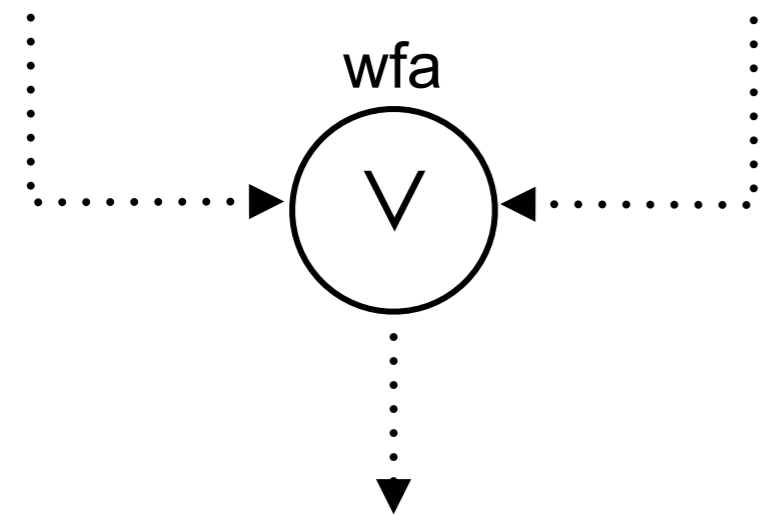


OR join
wait-for-all
(mismatched)

net

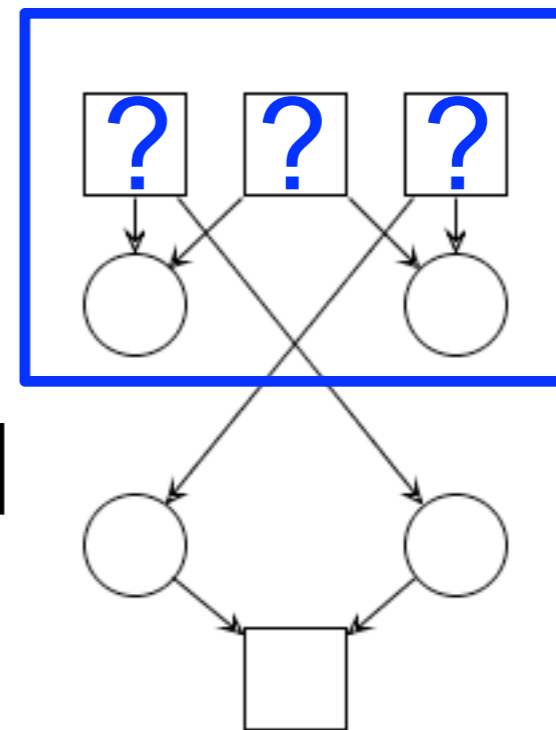
EPC

Petri net



OR join
wait-for-all
(mismatched)

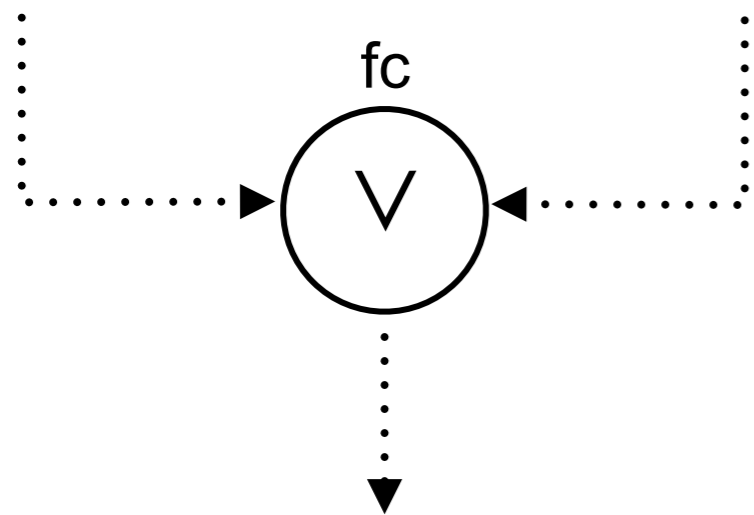
wfa
works well
with any
mismatched
split



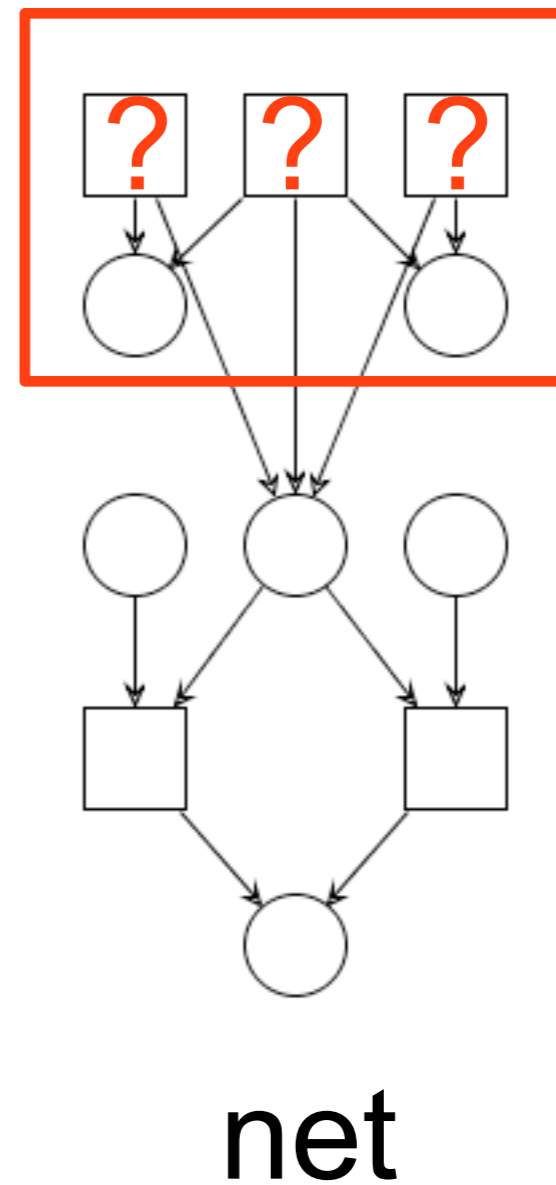
EPC

Petri net

mismatched corresponding split:
most general
case



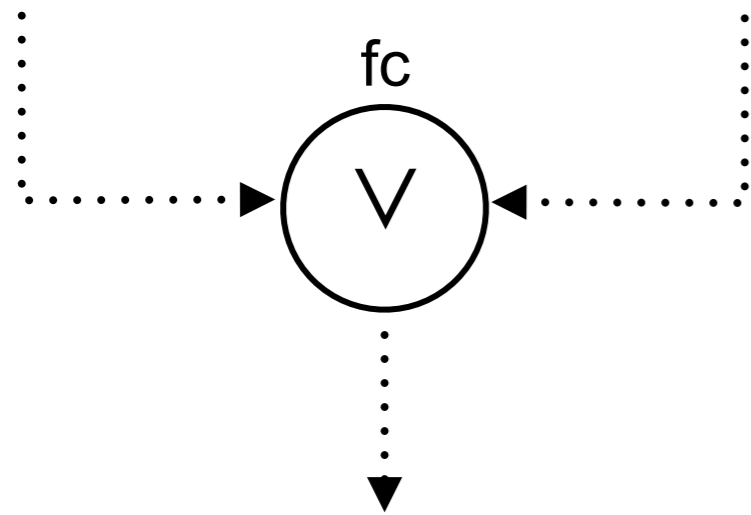
**OR join
first-come
(mismatched)**



EPC

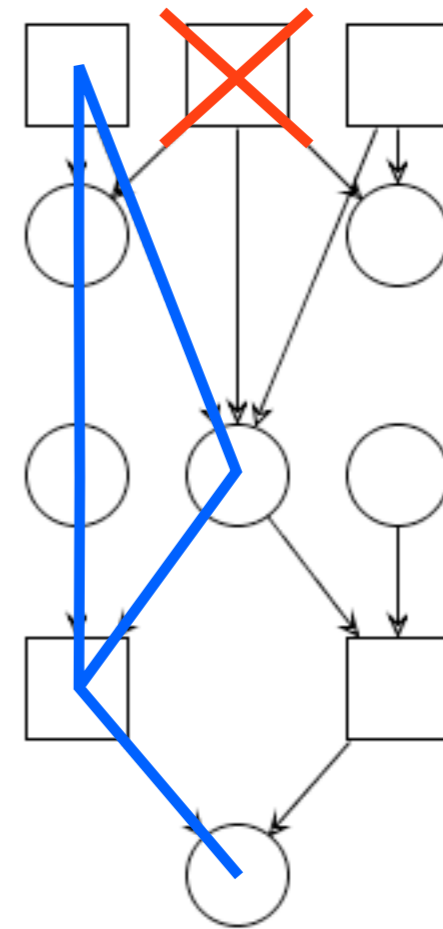
Petri net

corresponding
XOR split



**OR join
first-come
(unmatched)**

ok

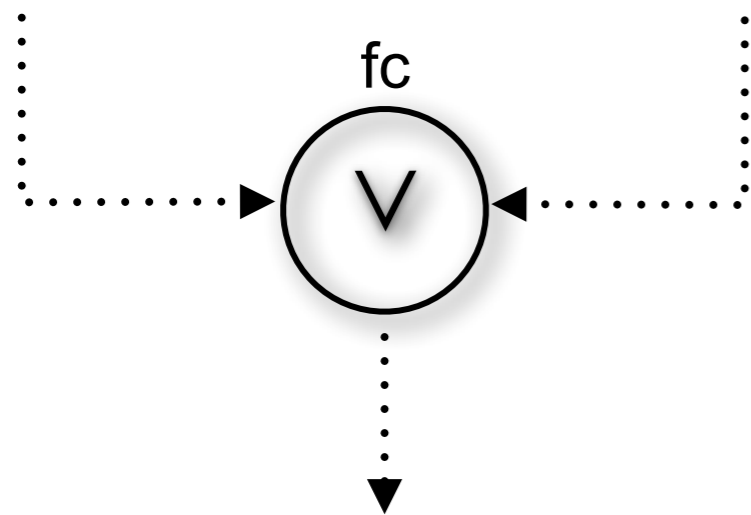


net

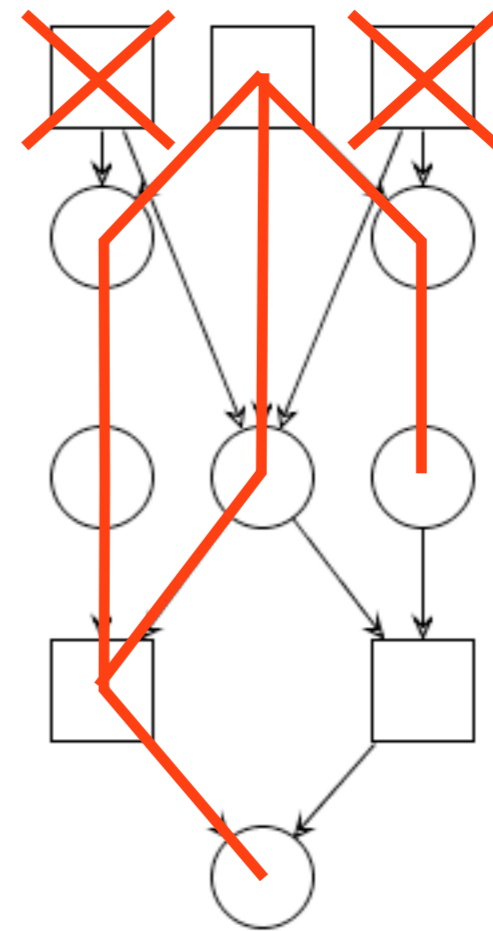
EPC

Petri net

corresponding
AND split



**OR join
first-come
(unmatched)**



pending
token!

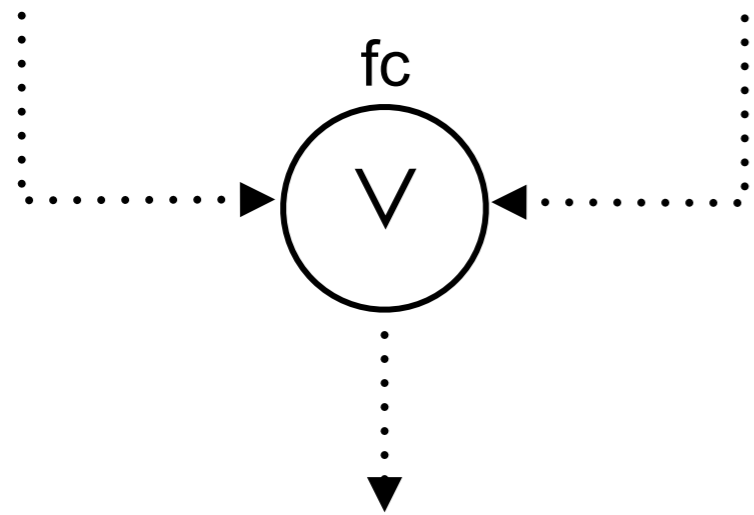
net

EPC

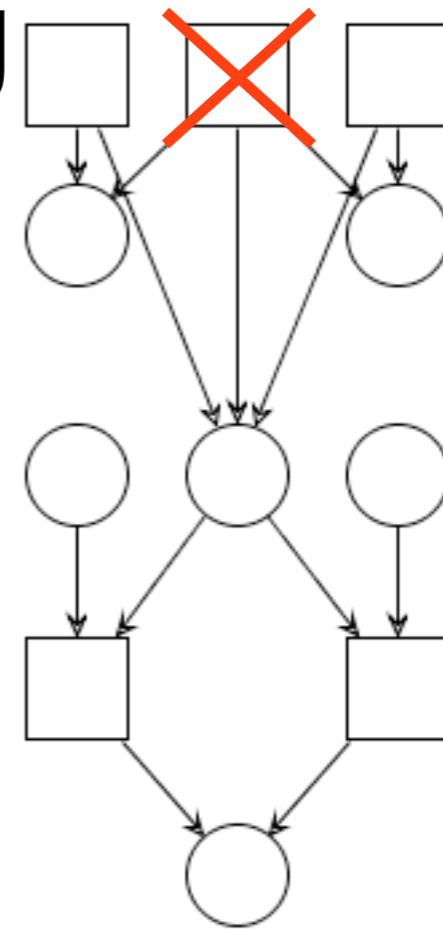
Petri net

fc:

better to have
a corresponding
XOR split!



**OR join
first-come
(mismatched)**

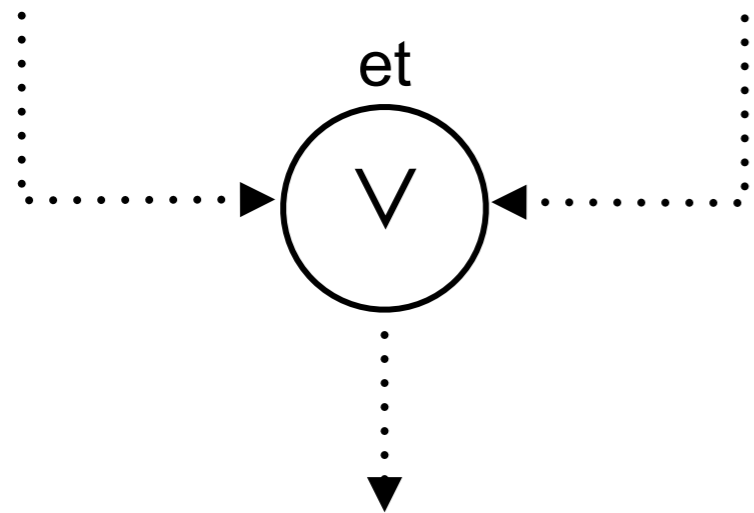


net

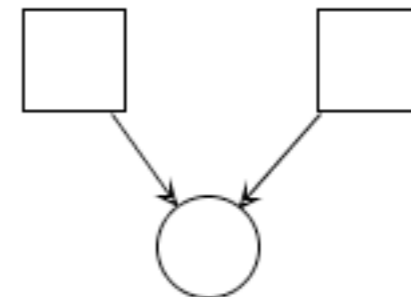
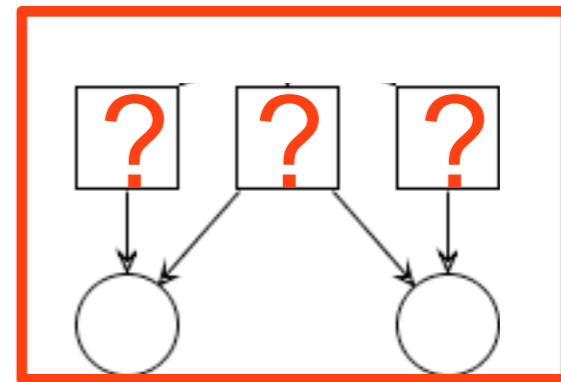
EPC

Petri net

mismatched corresponding split:
most general
case



**OR join
every-time
(mismatched)**

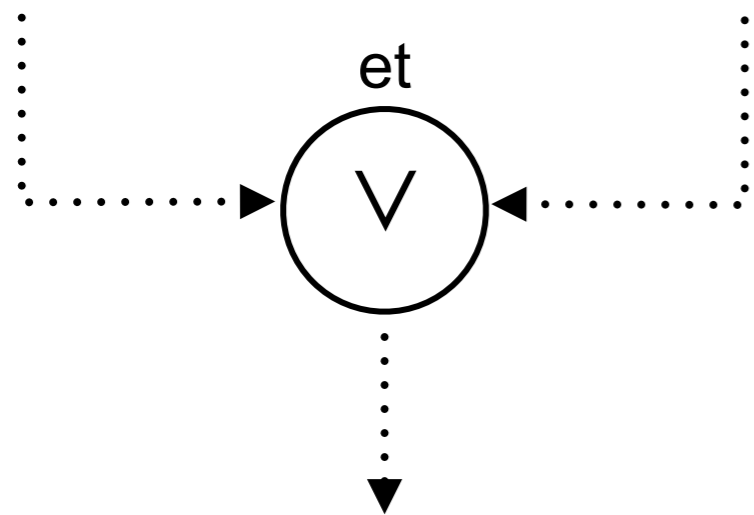


net

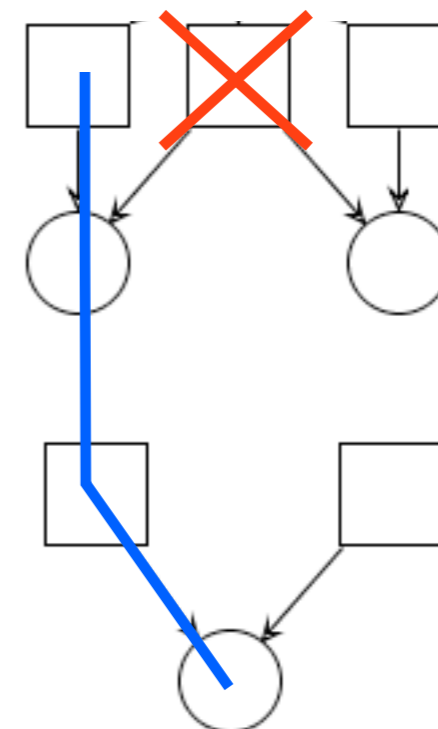
EPC

Petri net

corresponding
XOR split



**OR join
every-time
(mismatched)**



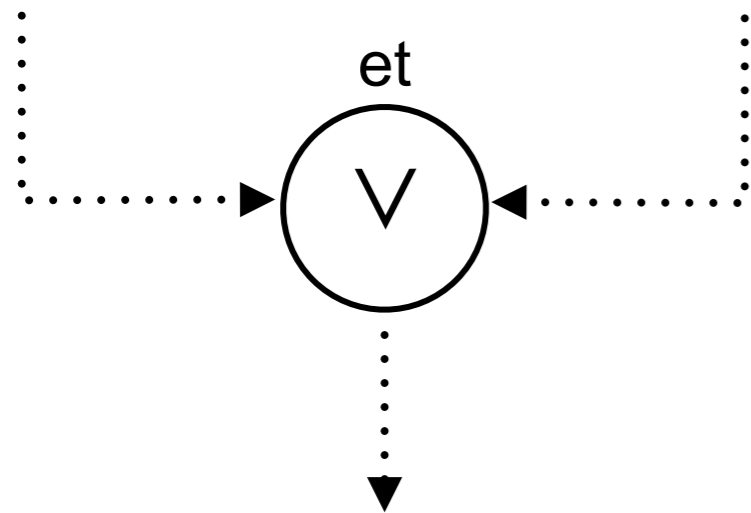
ok

net

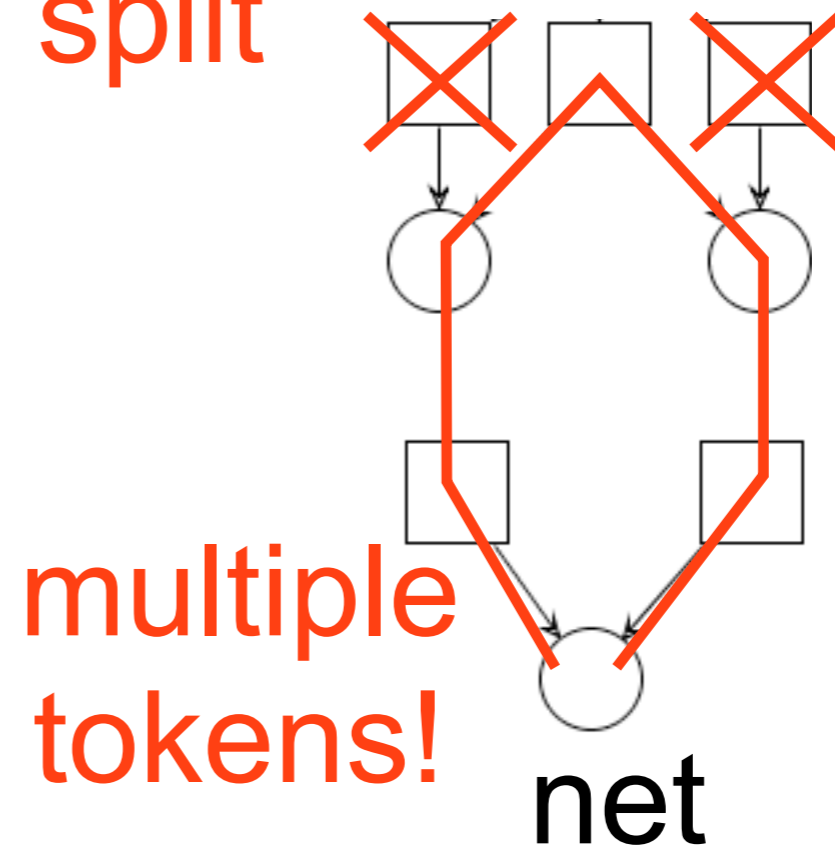
EPC

Petri net

corresponding
AND split



**OR join
every-time
(unmatched)**

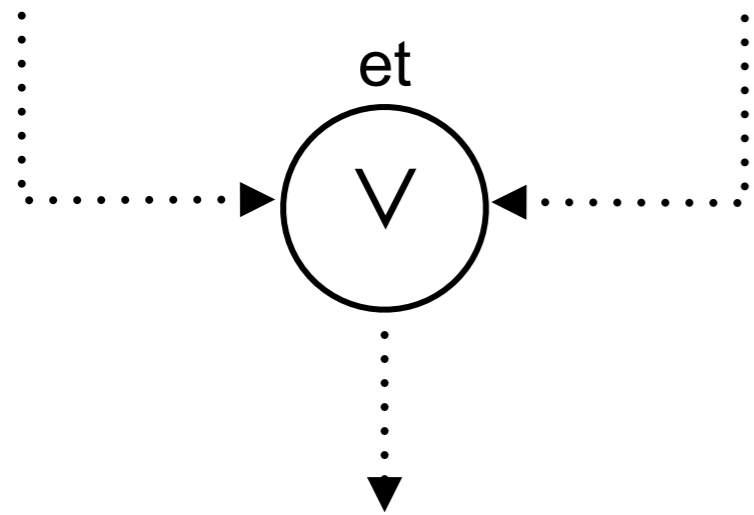


EPC

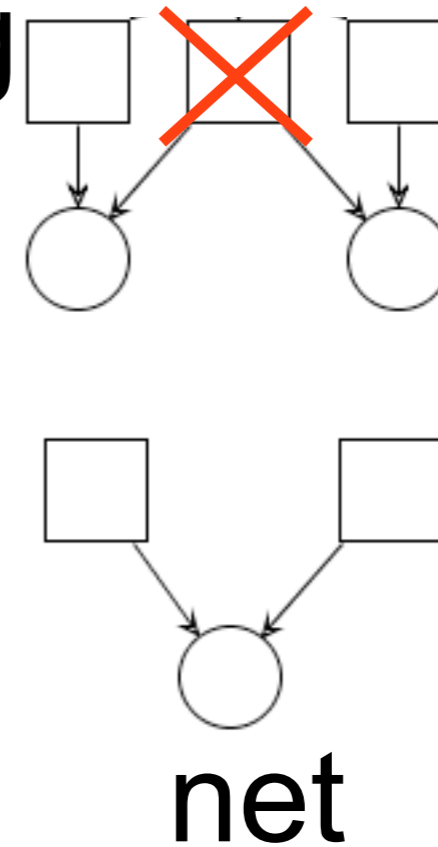
Petri net

et:

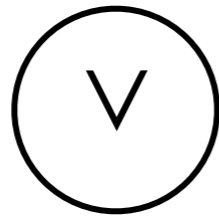
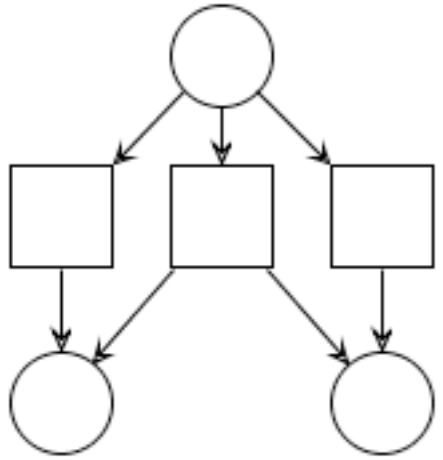
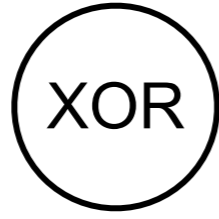
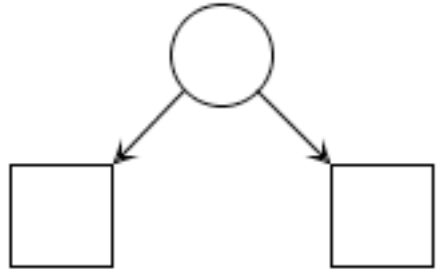
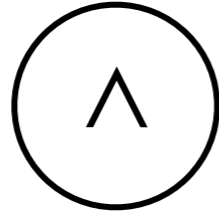
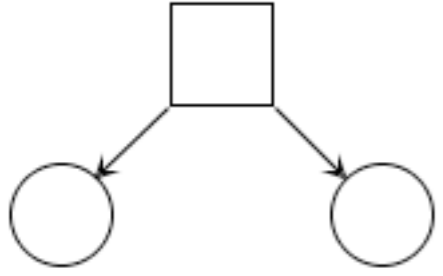
better to have
a corresponding
XOR split!



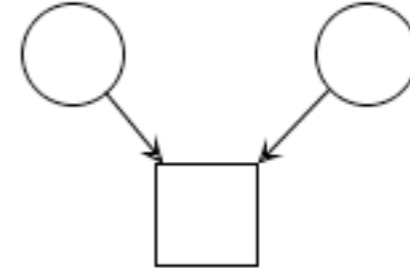
**OR join
every-time
(mismatched)**



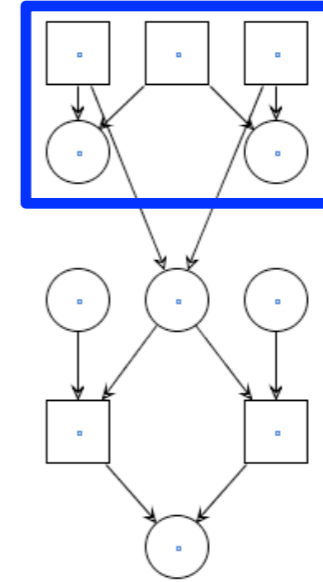
split



join

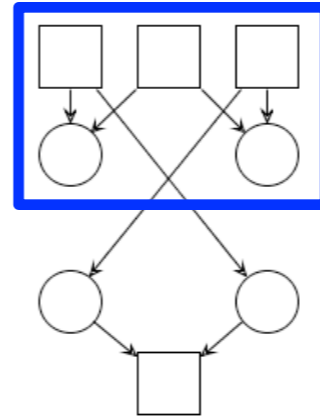


corresp.
split



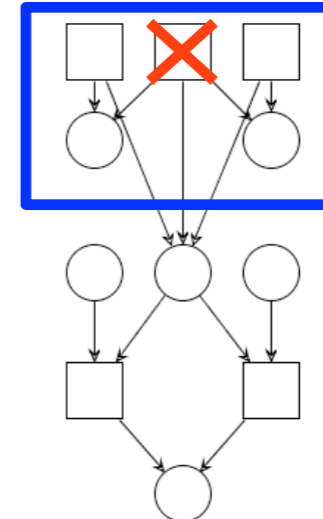
matched
OR split

corresp.
split: wfa

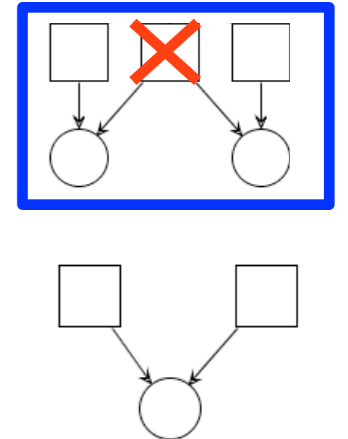


corresp.
split: fc

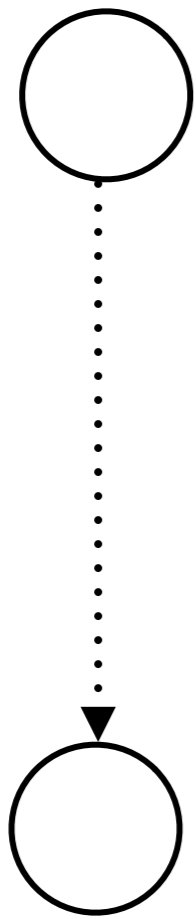
better to
have a
corresp.
XOR
split



corresp.
split: et



Ill-formed net

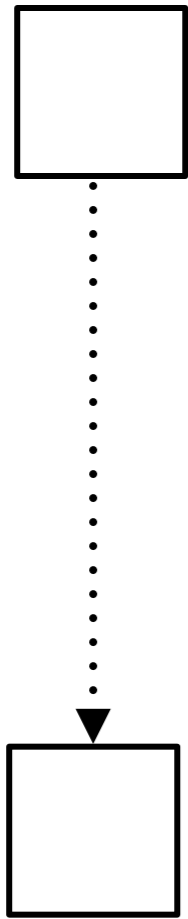


Petri net

dummy transition

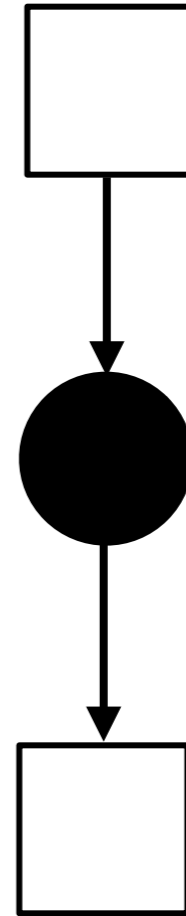


Ill-formed net



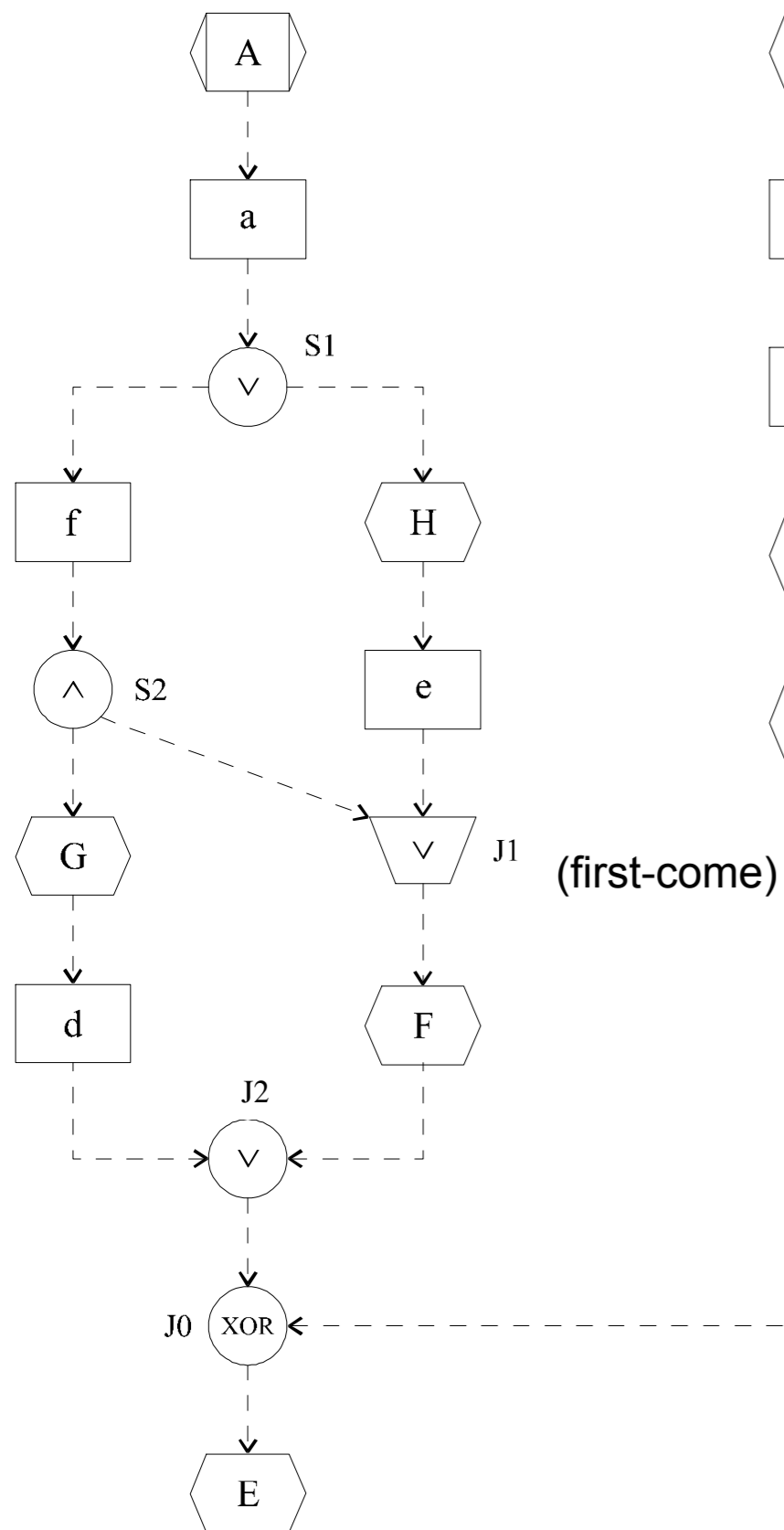
Petri net

dummy
place



Exercise

Sound?

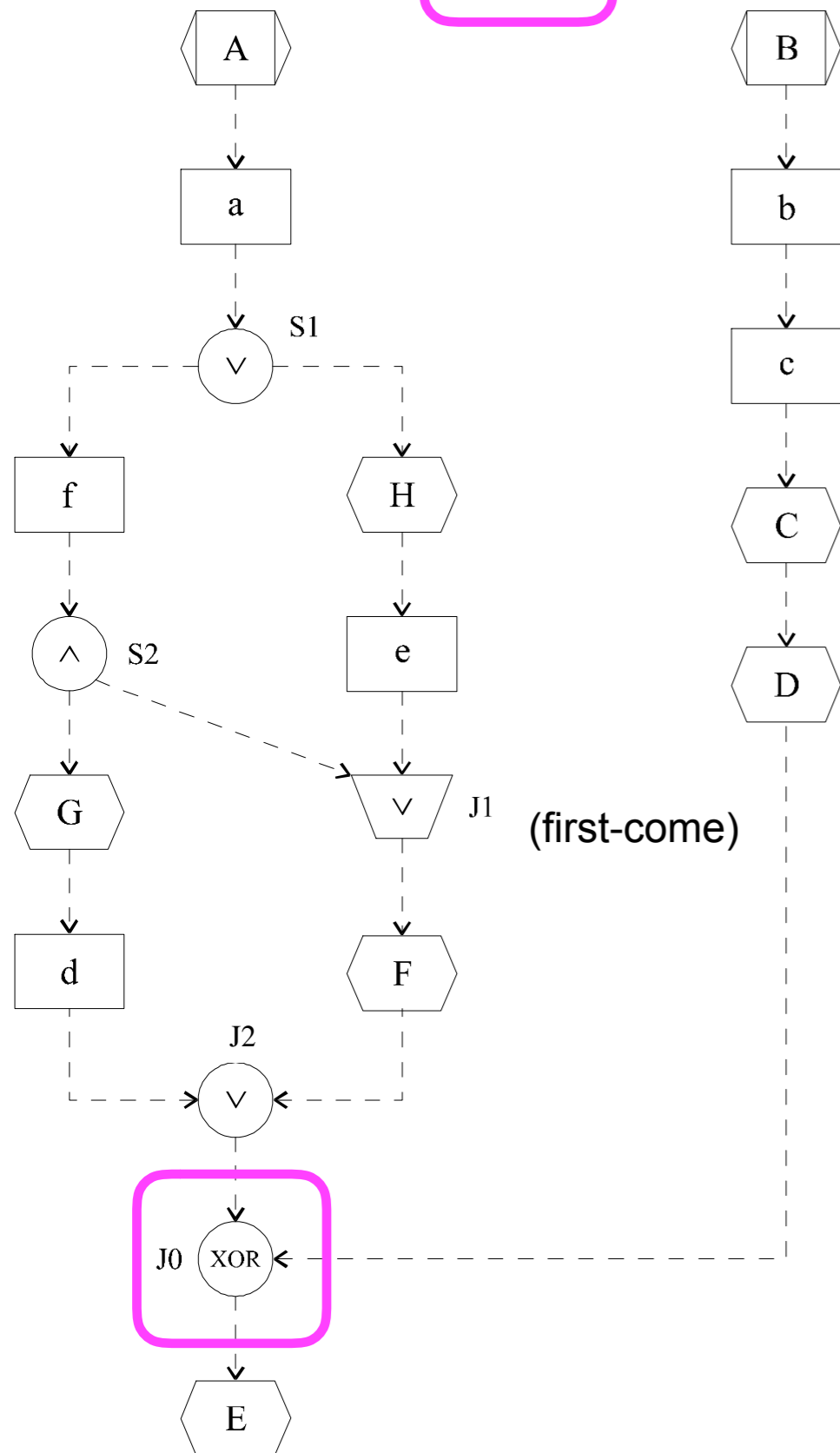


implicit
XOR

matching split

Exercise

Sound?

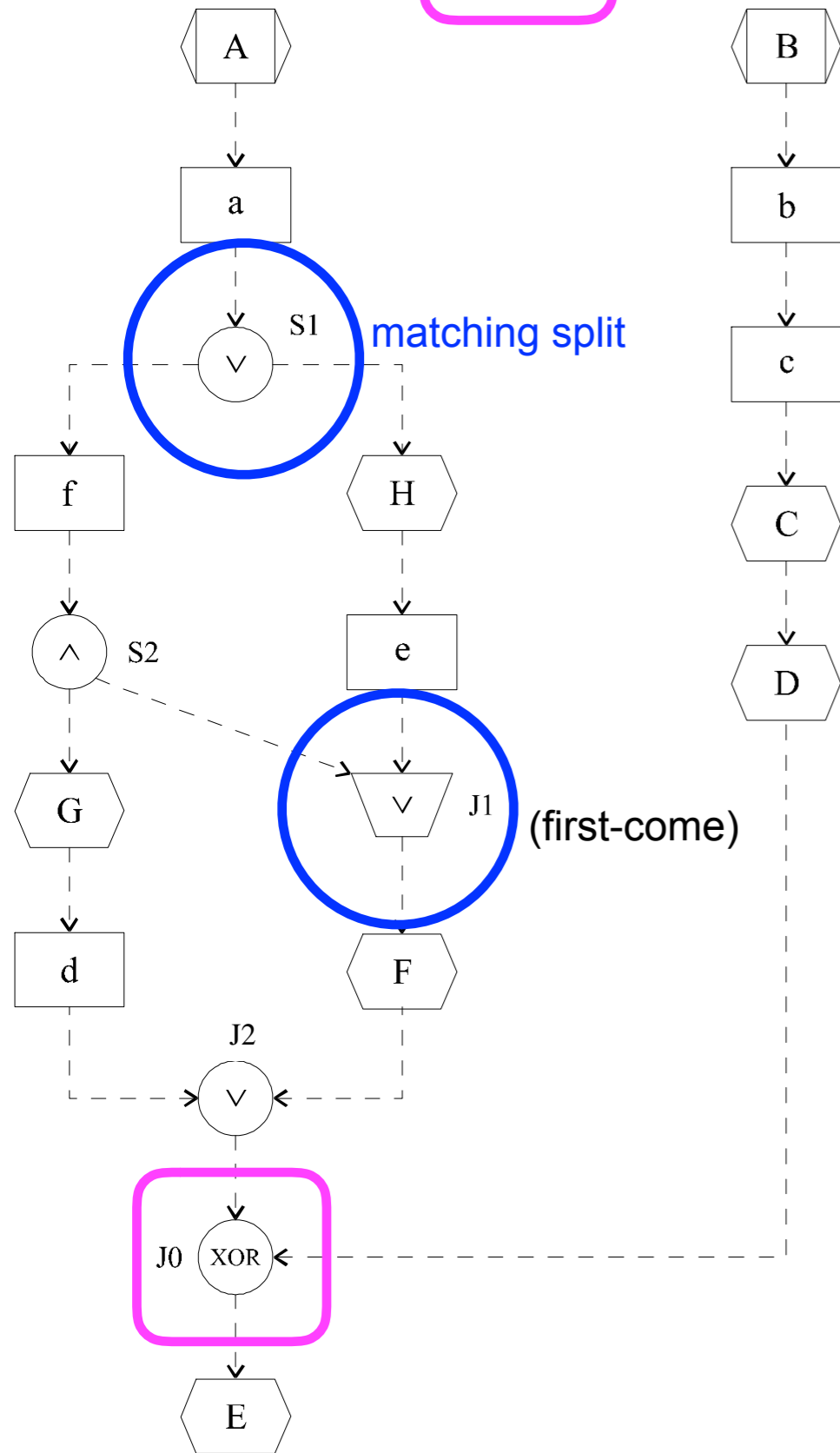


implicit XOR

matching split

Exercise

Sound?

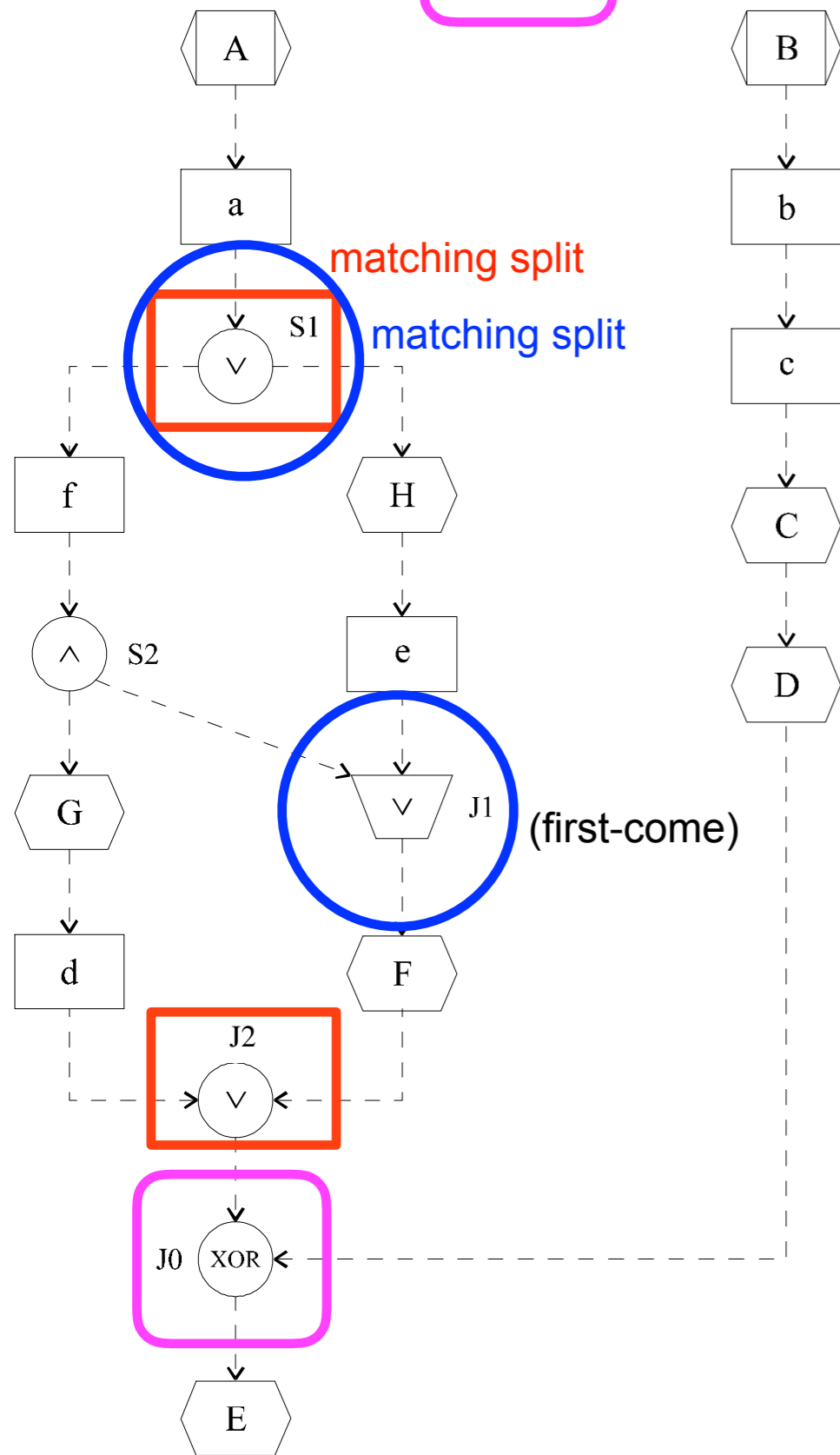


implicit XOR

matching split

Exercise

Sound?

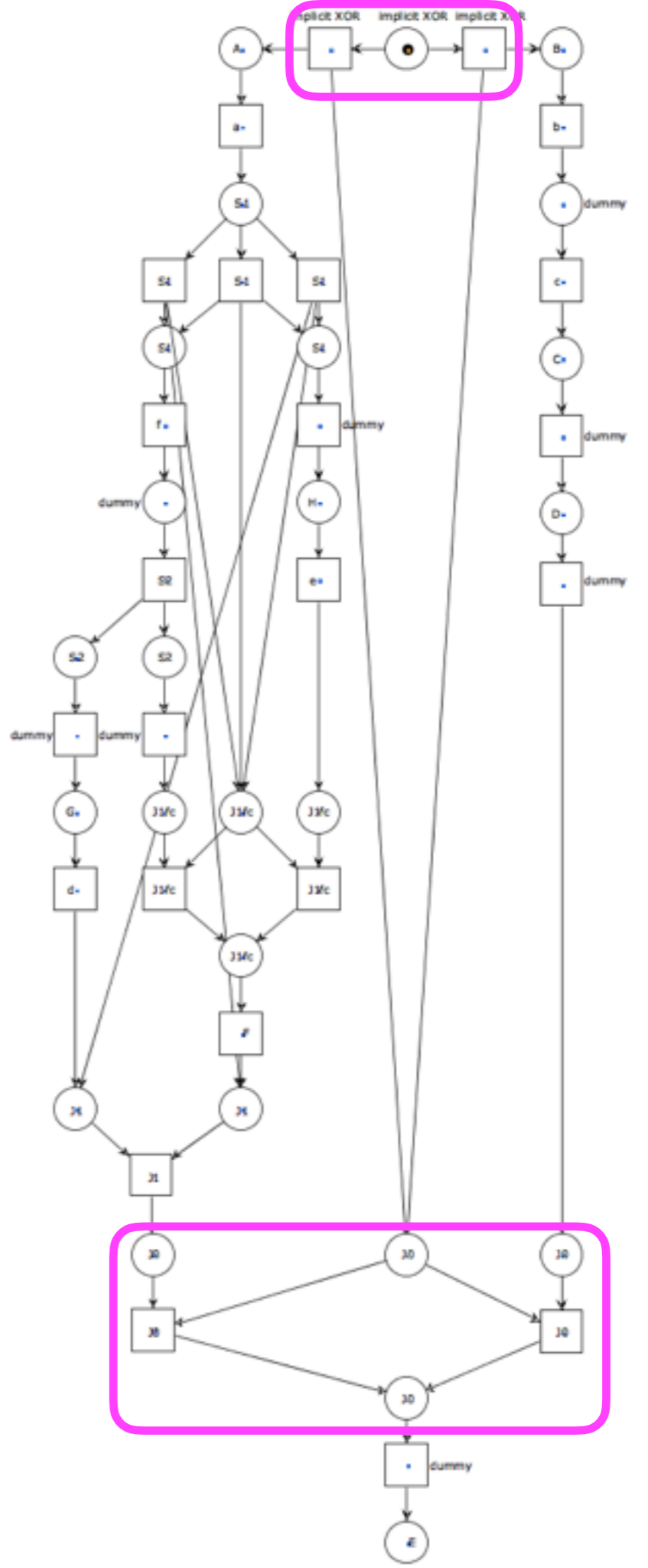
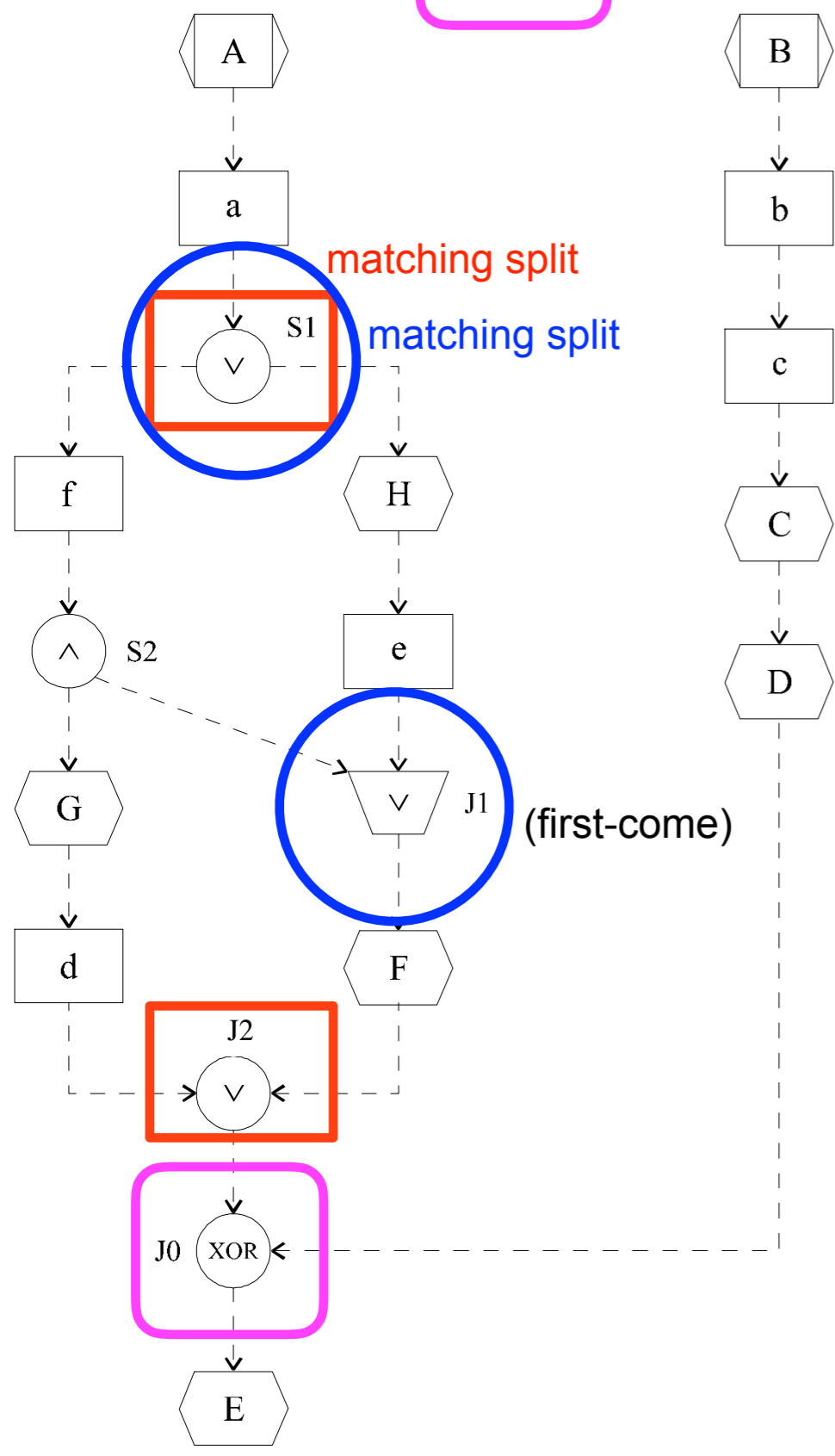


implicit XOR

matching split

Exercise

Sound?

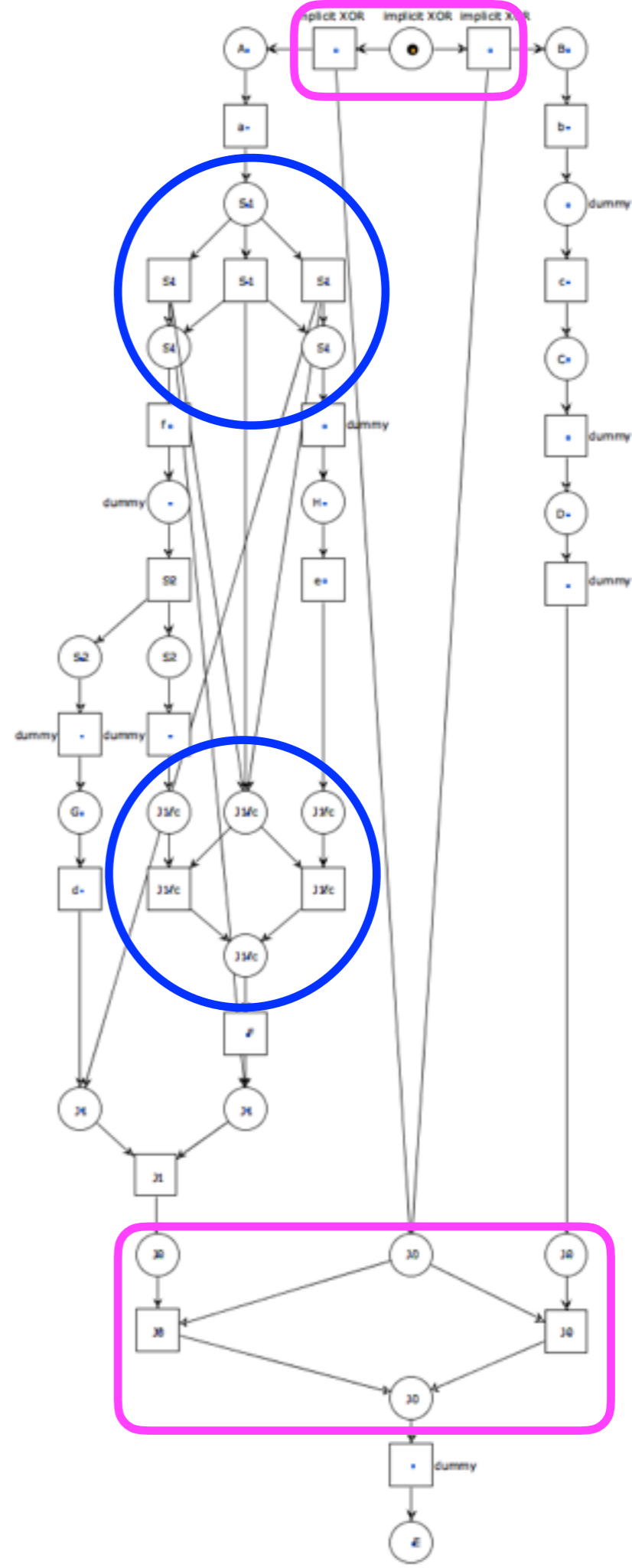
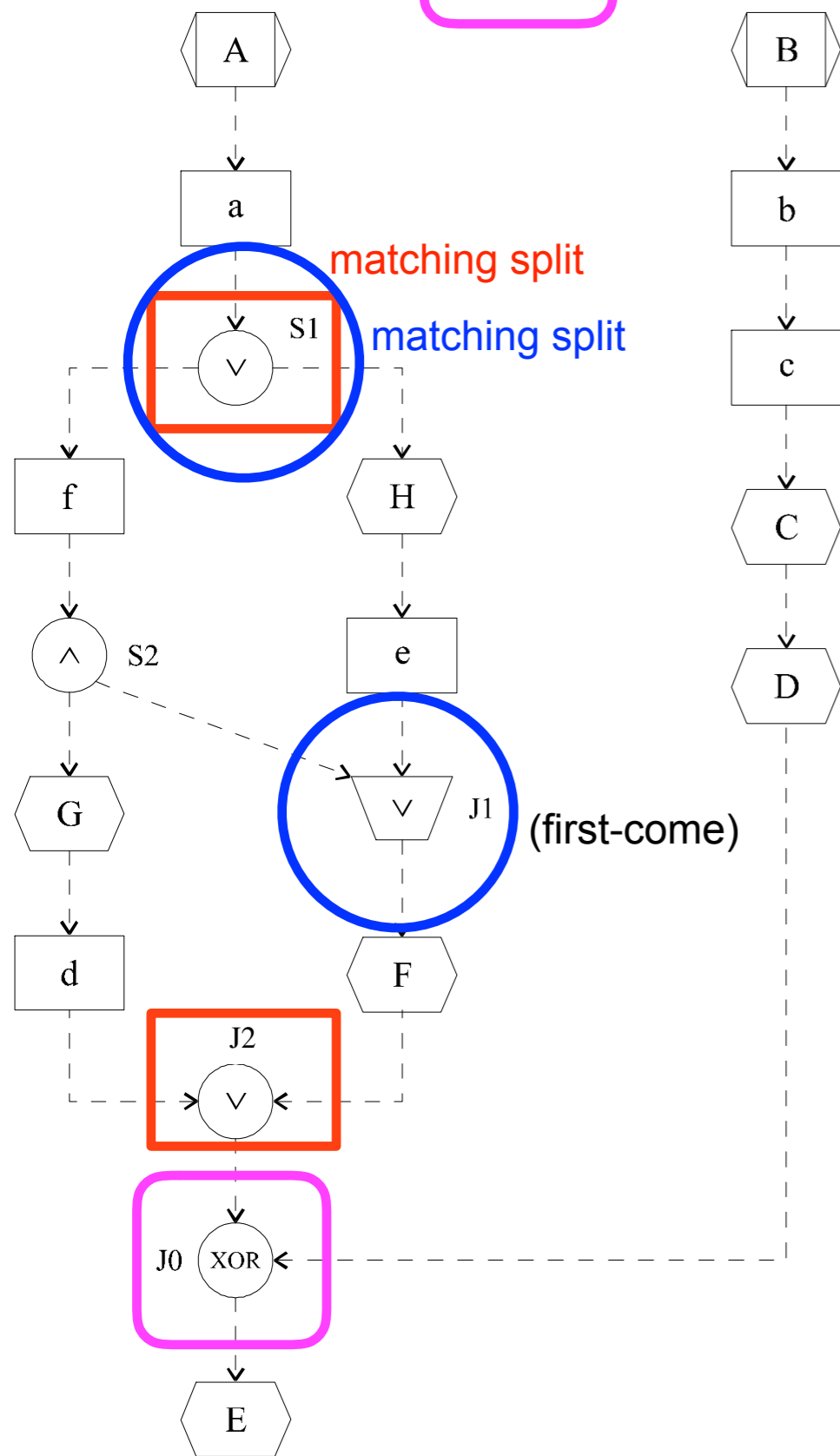


implicit XOR

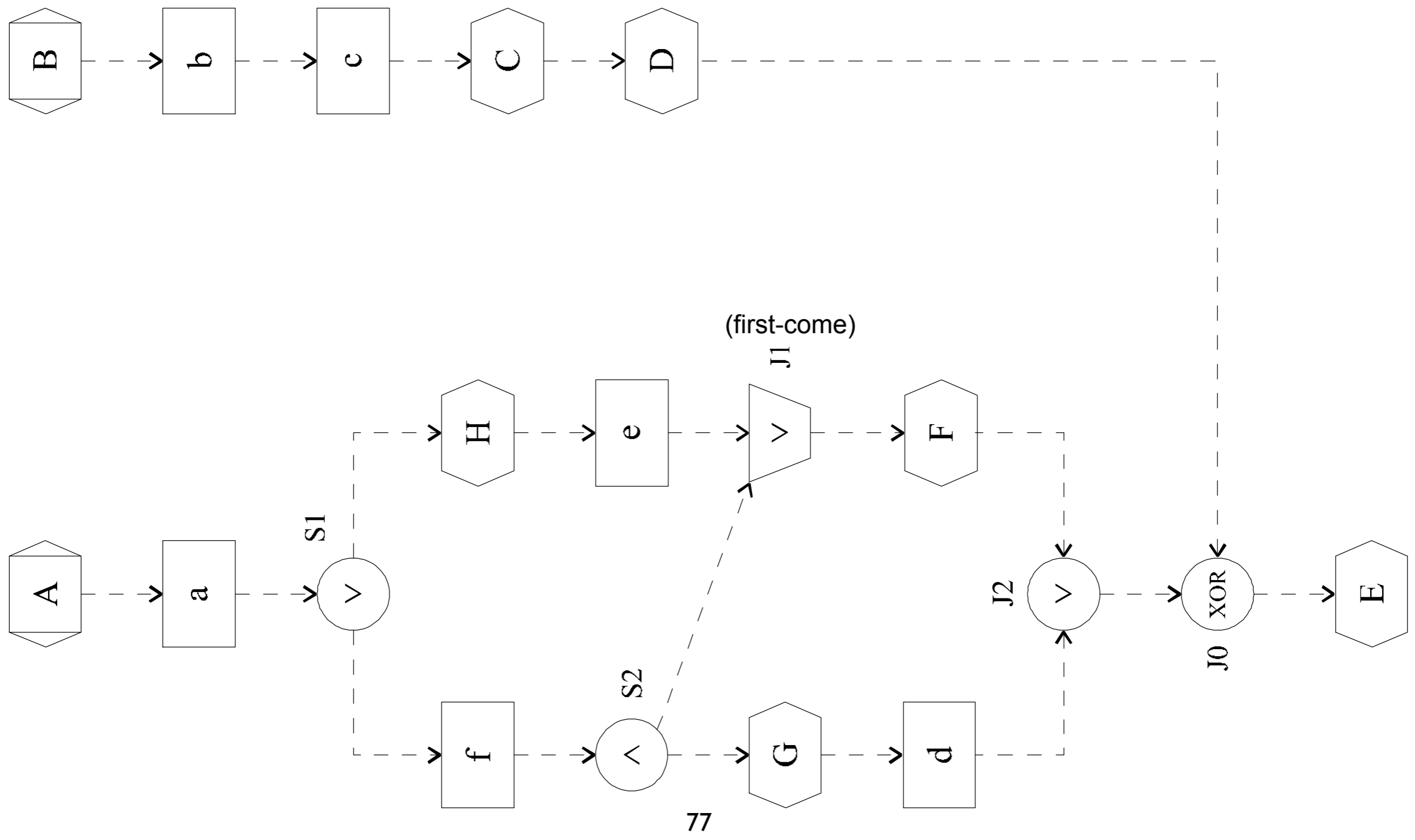
matching split

Exercise

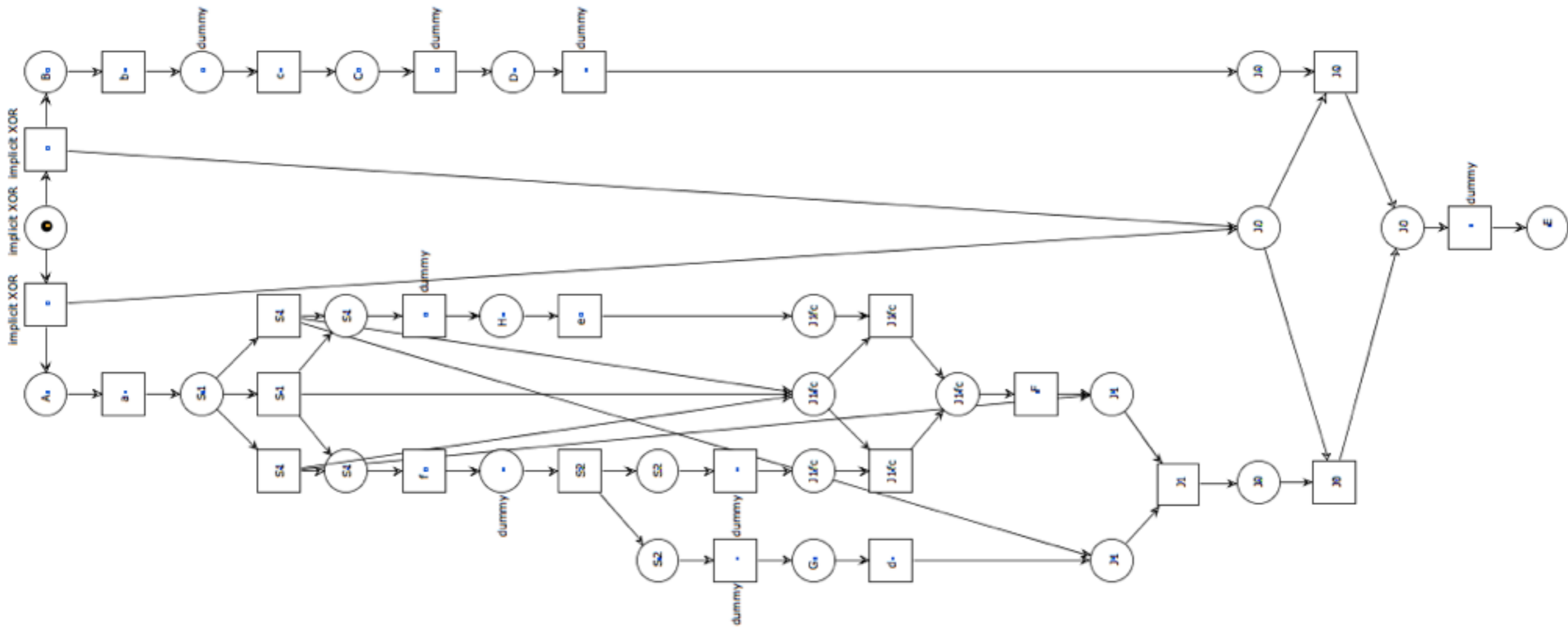
Sound?



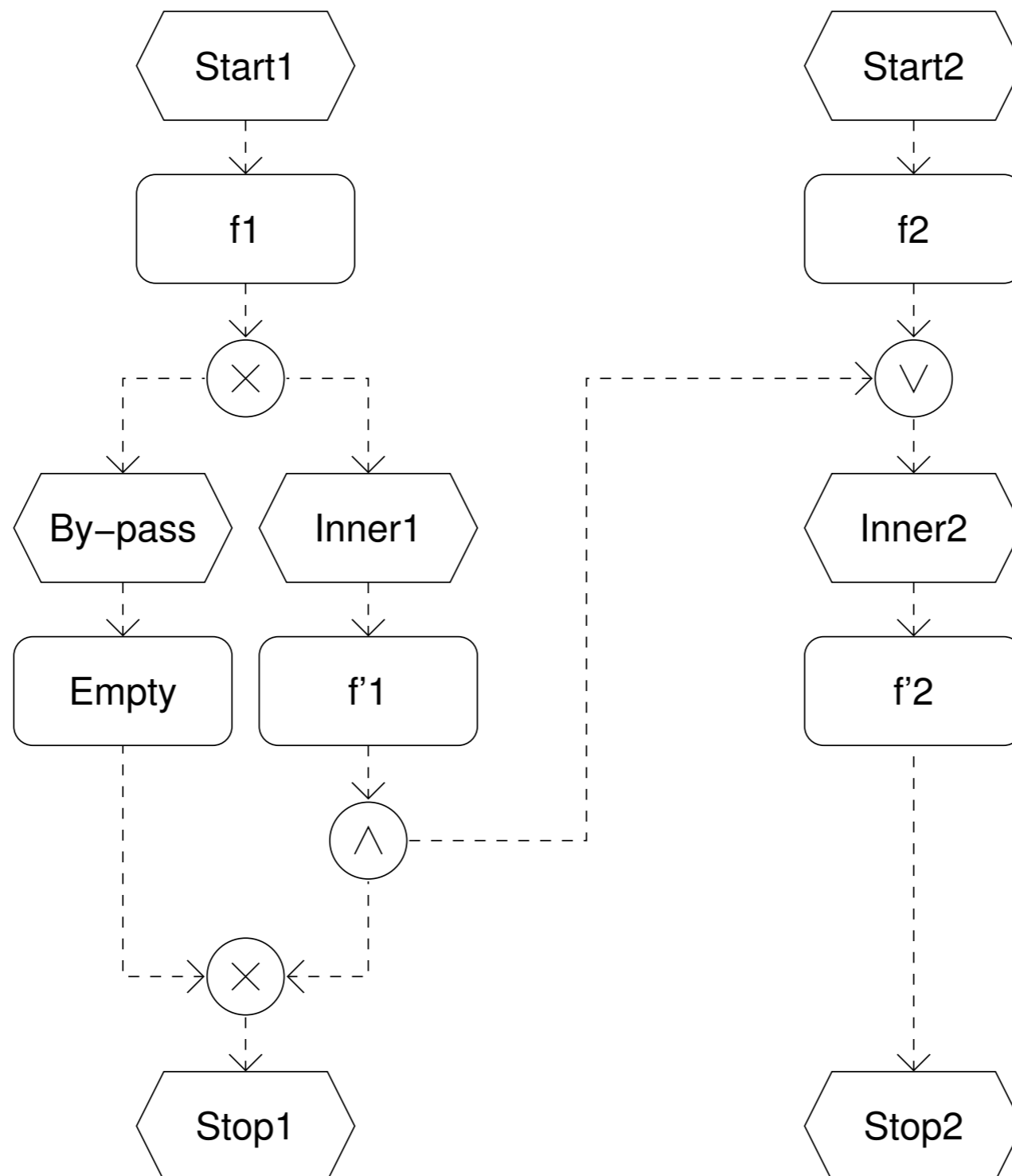
ZOOM IN



ZOOM IN



Exercise



Sound?

Summary of problems

We need to decorate the EPC diagram

joins must be decorated with matching/corresponding splits

mismatched OR-joins must be decorated with policies

Split / join mismatch may induce unexpected behaviour

Possible introduction of dummy places and transitions

Second attempt
(no decoration required)

Simplified EPC

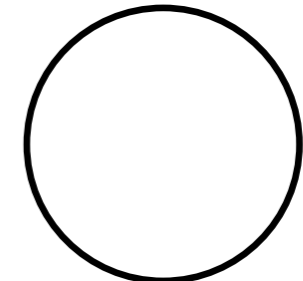
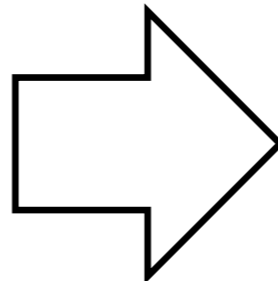
We rely on event / function alternation
along paths in the diagram
and also **along paths between two connectors**

OR-connectors are not considered

EPC 2 Petri nets: events and functions



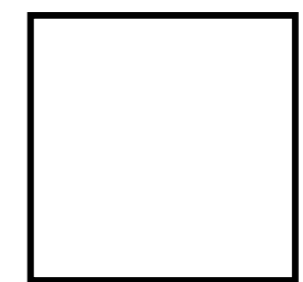
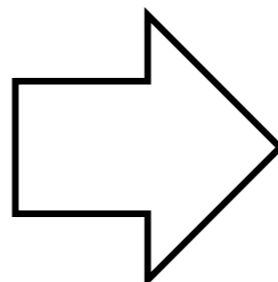
event



place



function



transition

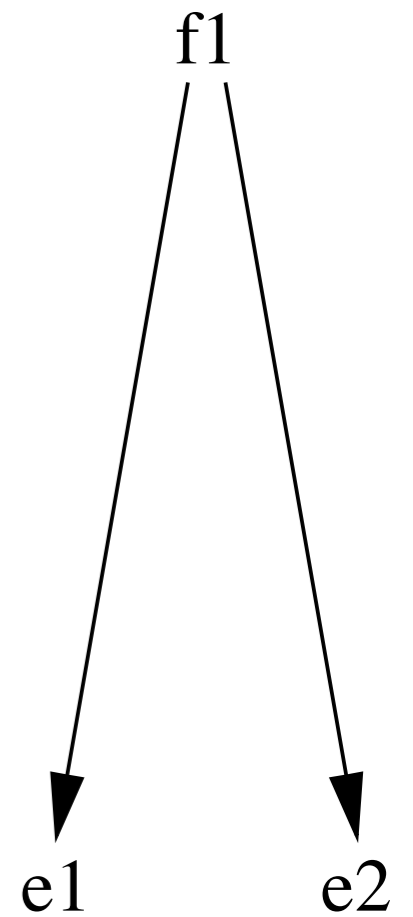
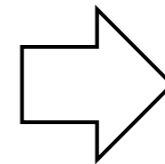
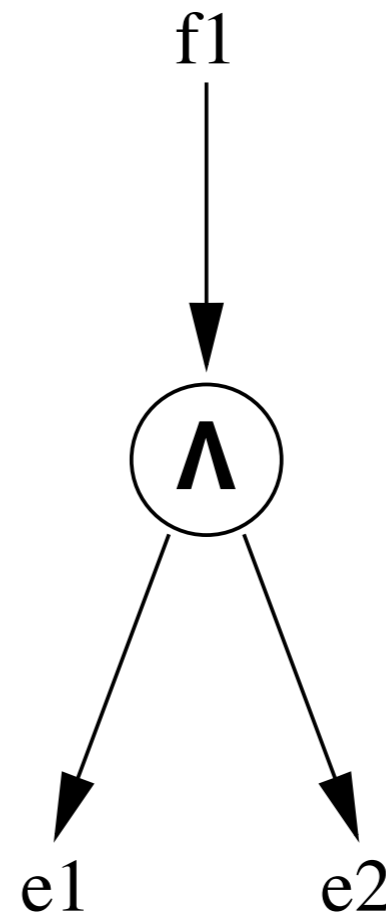
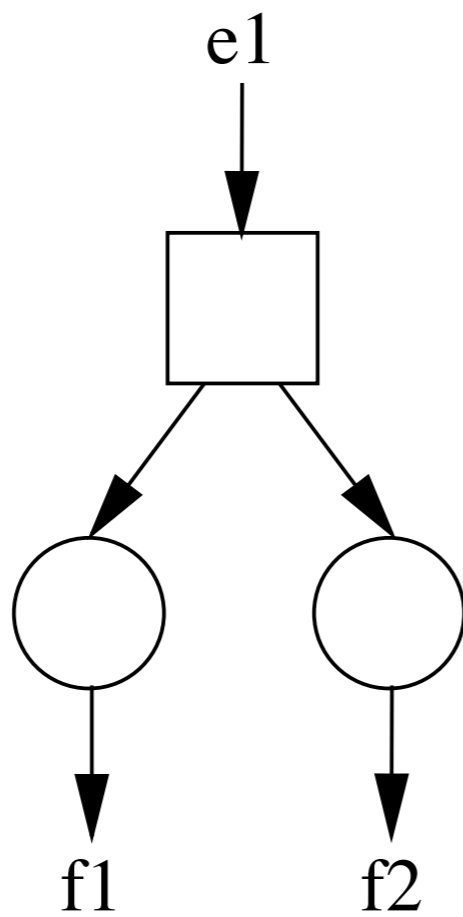
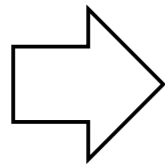
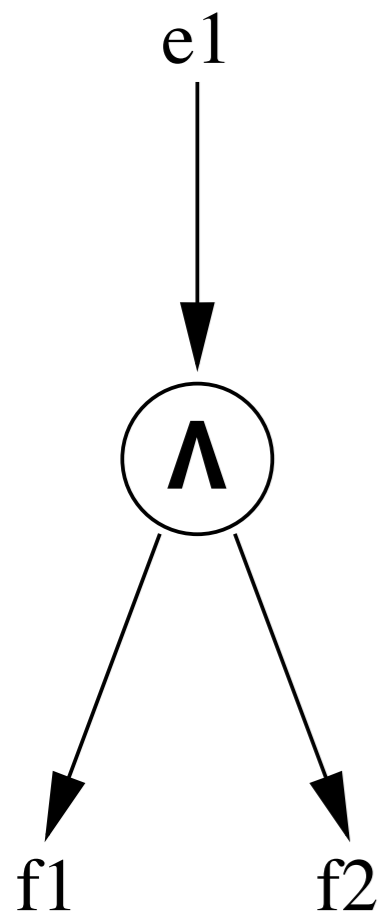
EPC 2 Petri nets: split/join connectors

The translation of logical connectors
depends on the context:

if a connector connects **functions to events**
we apply a certain translation

if it connects **events to functions**
we apply a different translation

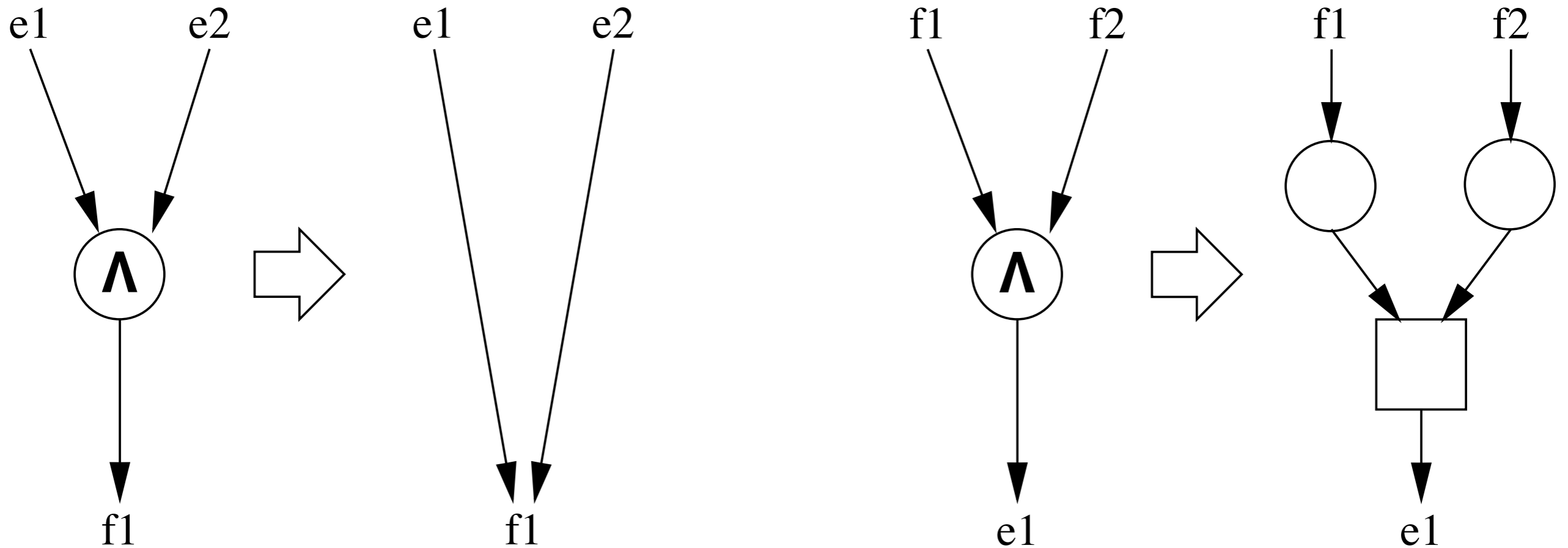
EPC 2 Petri nets: AND split



(event to functions)

(function to events)

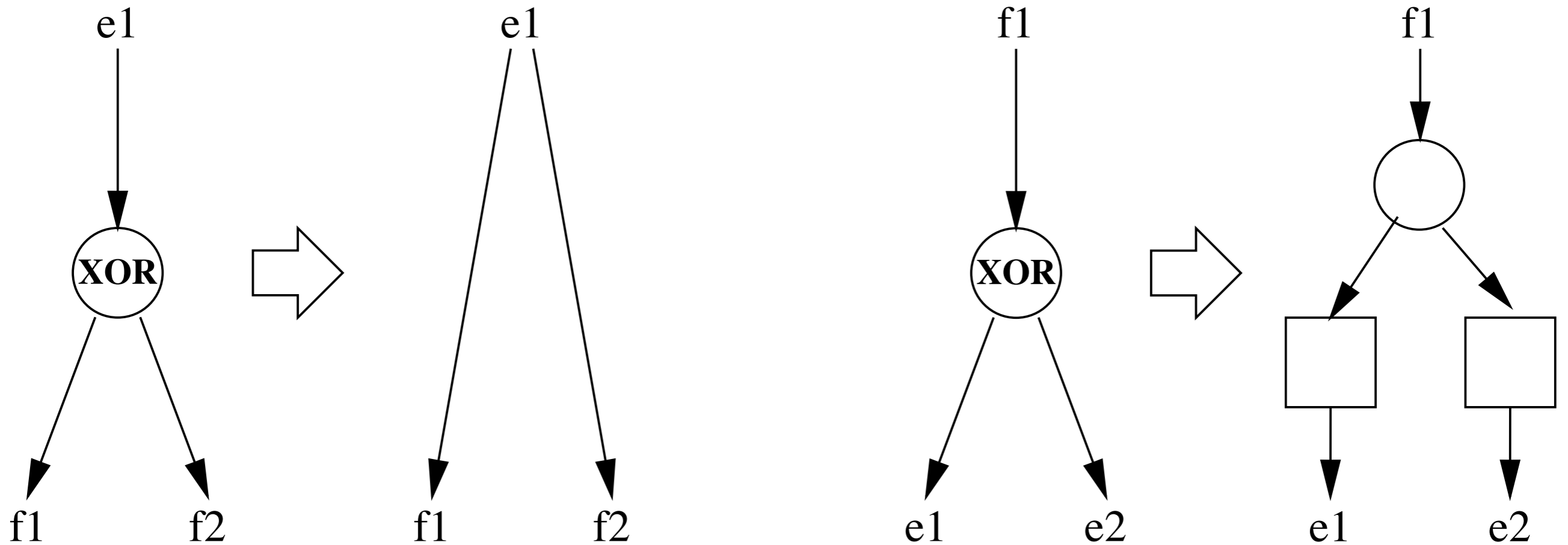
EPC 2 Petri nets: AND-join



(events to function)

(functions to event)

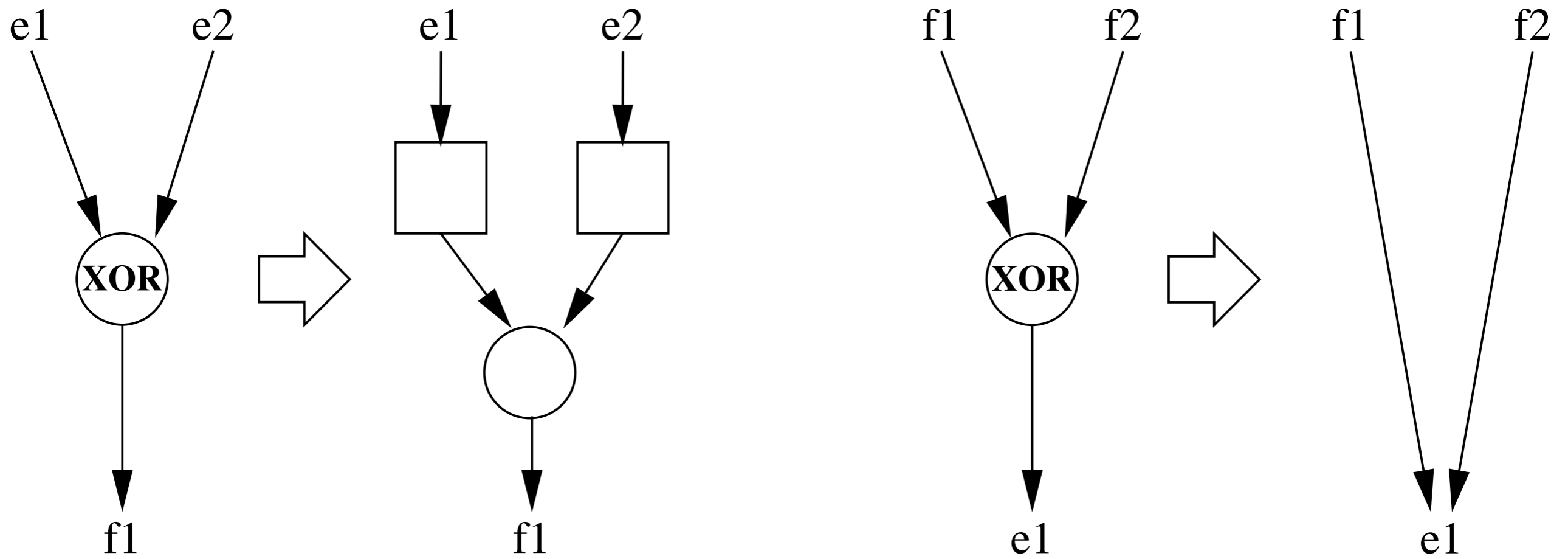
EPC 2 Petri nets: XOR split



(event to functions)

(function to events)

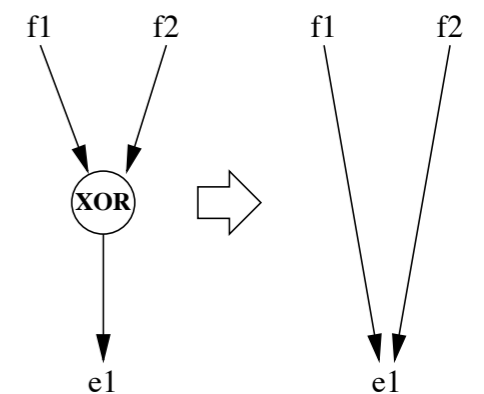
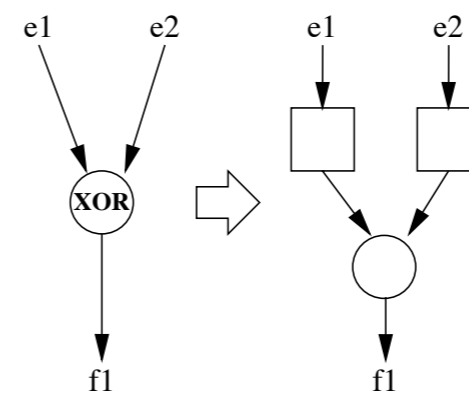
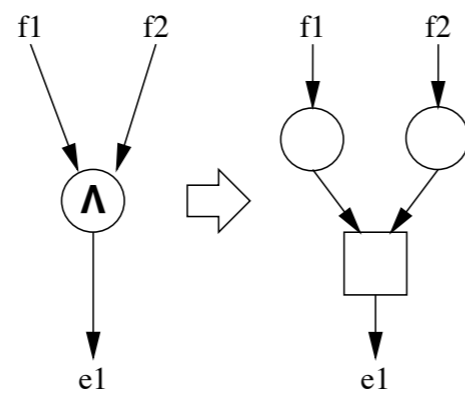
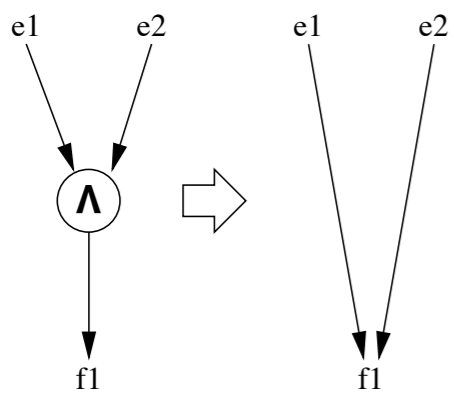
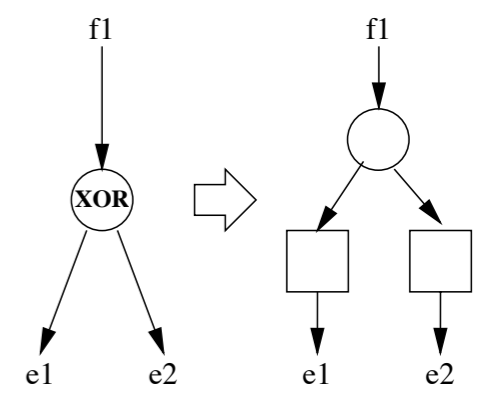
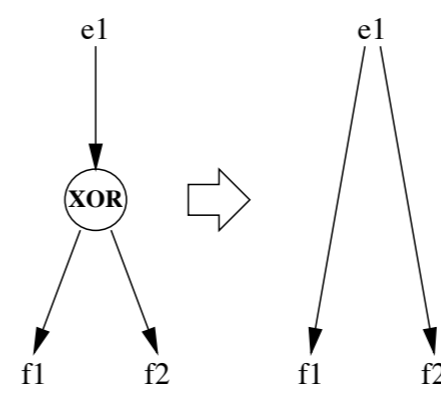
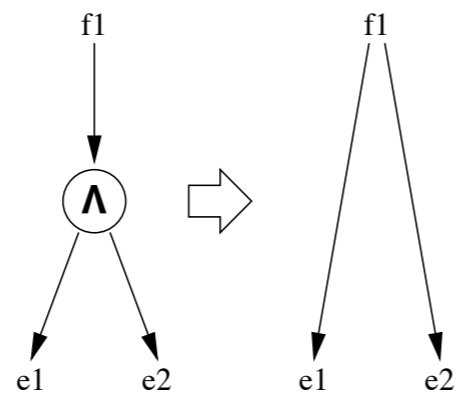
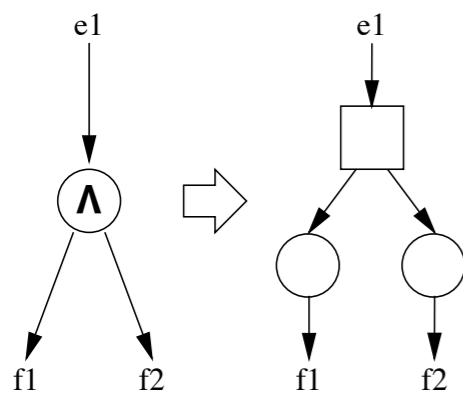
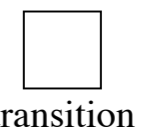
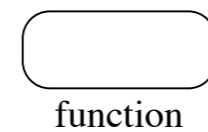
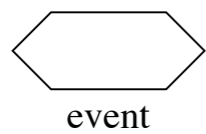
EPC 2 Petri nets: XOR join



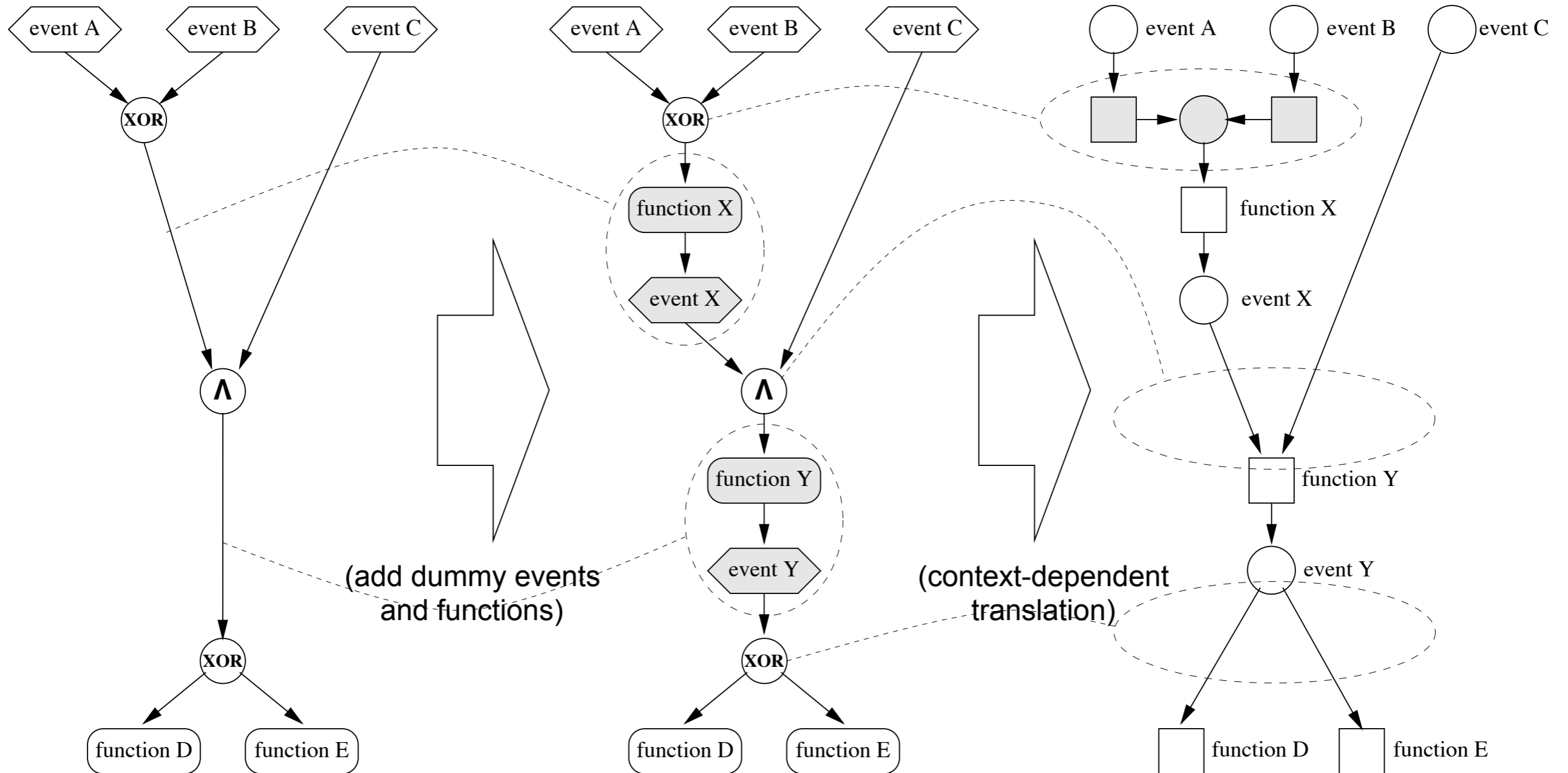
(events to function)

(functions to event)

EPC 2 Petri nets: at a glance



EPC 2 nets: Example



Outcome

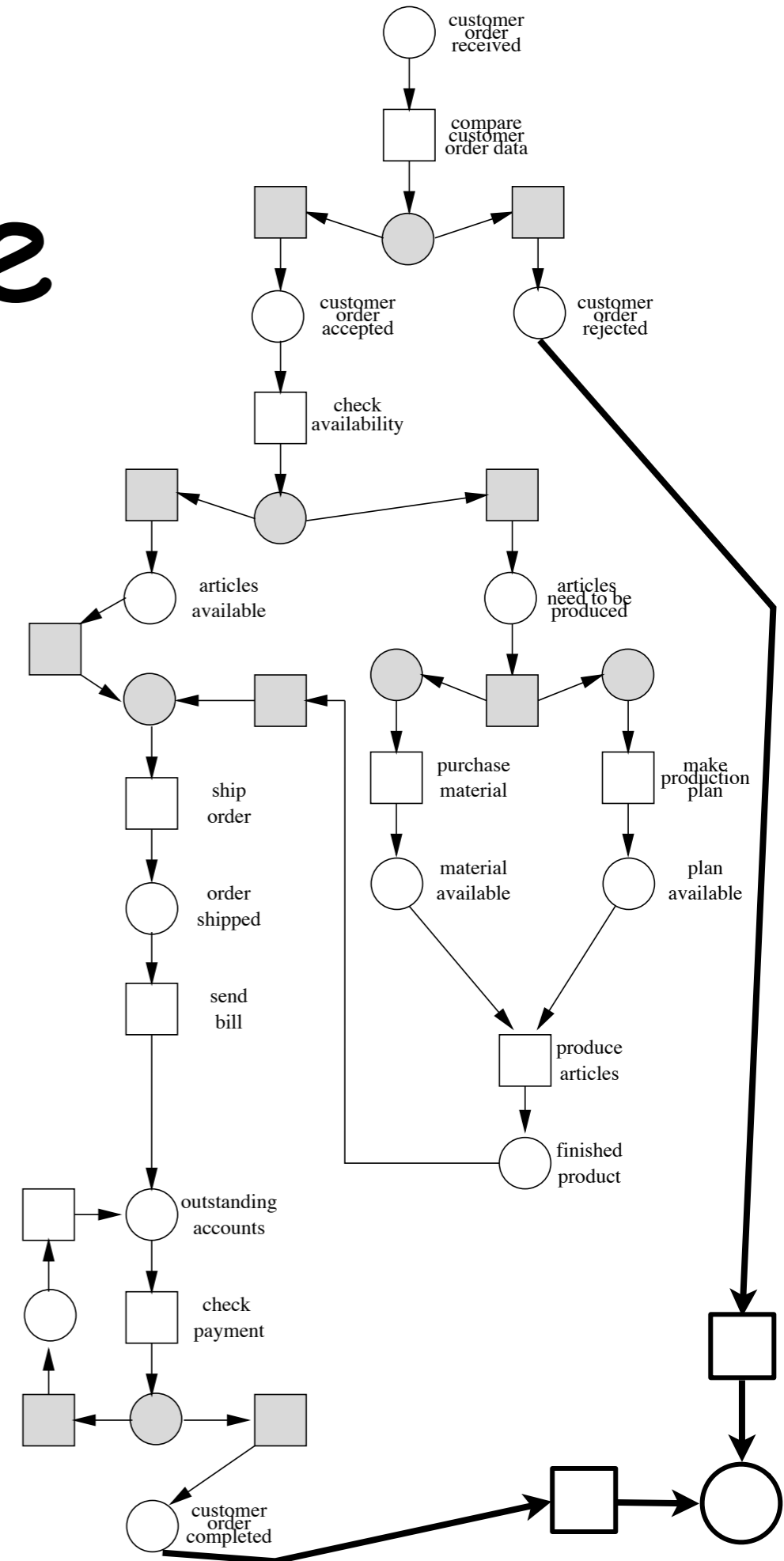
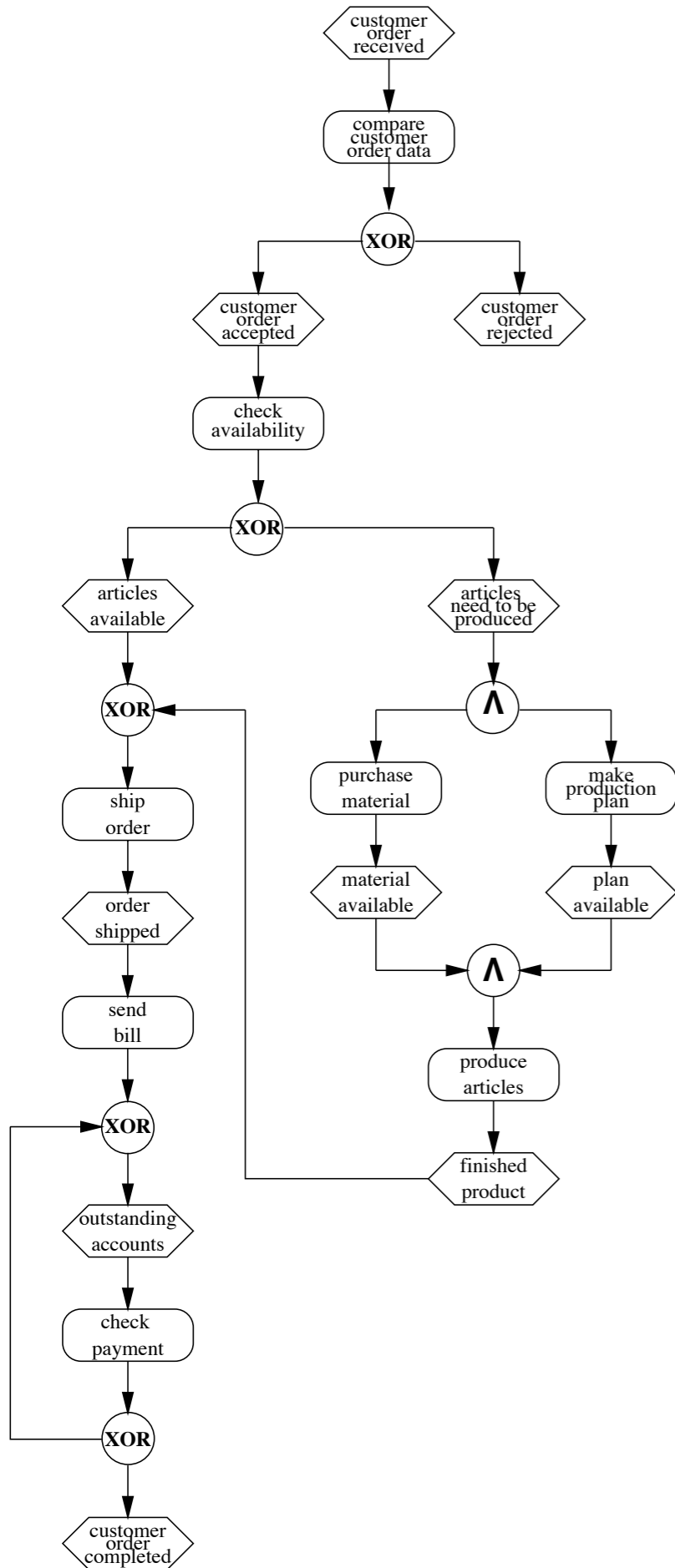
From any EPC we derive a free-choice net

Moreover, if we add unique start / end events
(and suitable transitions attached to them)

the net is a workflow net

Exercise

Check it sound!



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

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

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)

Remember some good old friends



Chief Process Officer



Process participants



System architect



Business engineer

EPC



Process designer

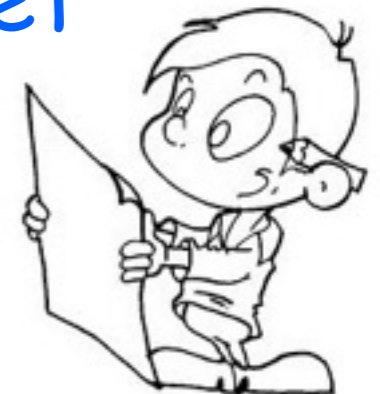


Knowledge worker



Process responsible

WFnet



System developer

A secret not to tell

Ambiguity is useful in practice!

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**

Yet

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

Consequences

Ambiguous process descriptions
arise in the **design phase**

therefore

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

Problem

EPC is fine (widely adopted)

WF nets offer a useful tool

but

Soundness is too demanding at early stages

Relaxed soundness

A **sound** behaviour:

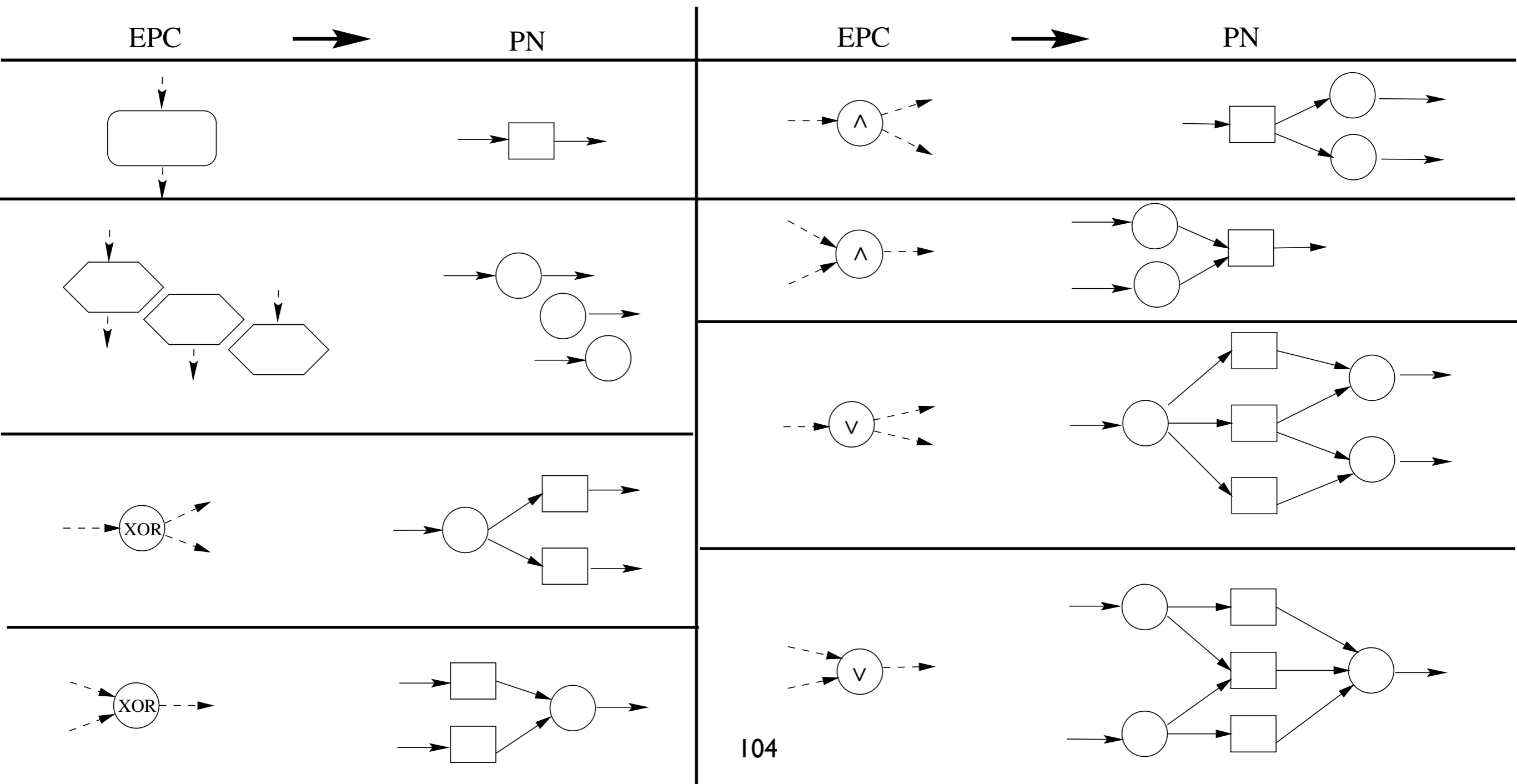
we move from a start event to an end event
so that nothing blocks or remains undone

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

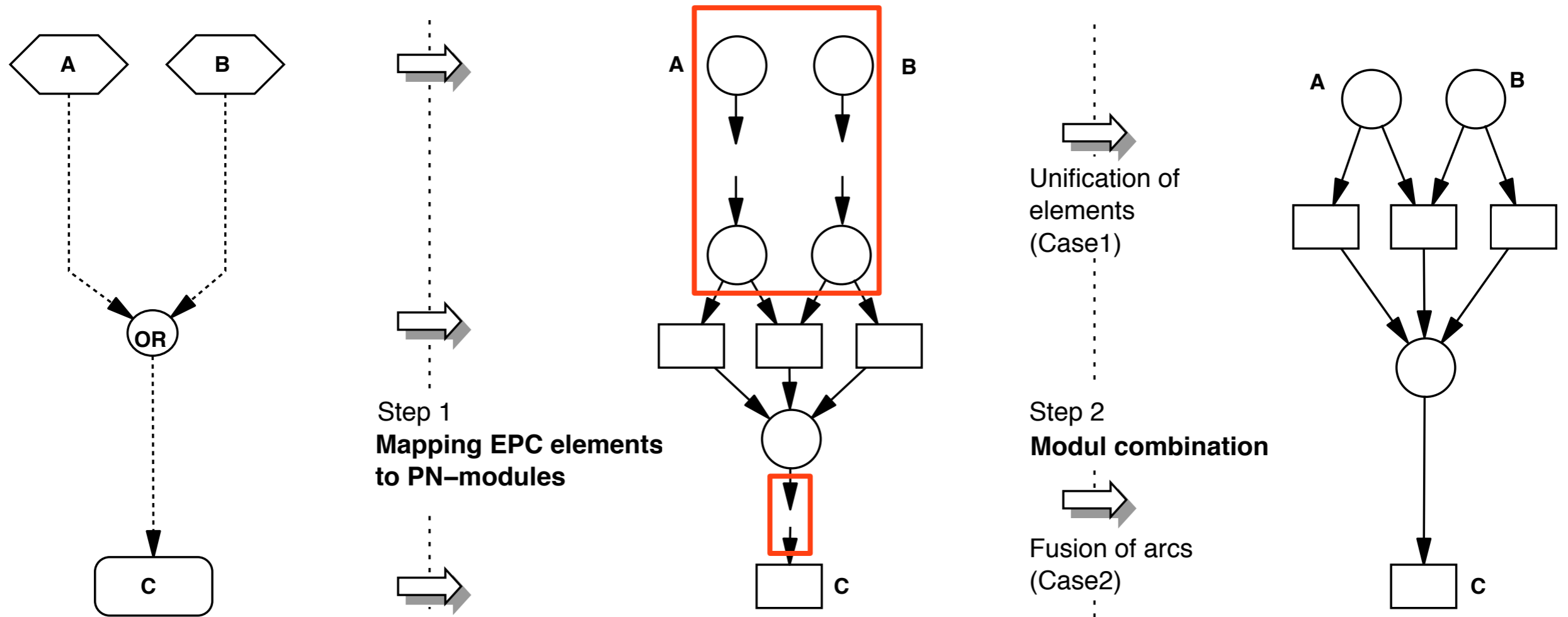
If some unsound behaviour is possible
but **enough** sound paths exist
the process is called **relaxed sound**

A 3-steps approach
(keep it simple!)

Step 1: straightforward element map

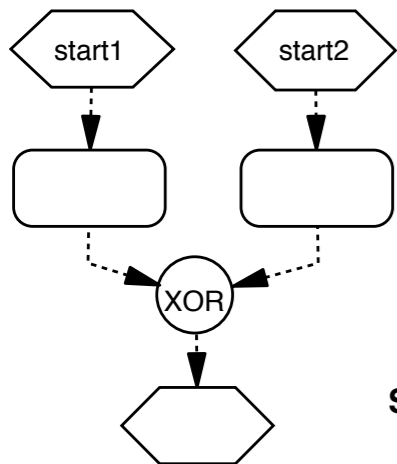


Step 2: element fusion

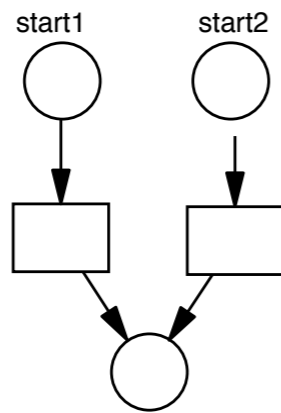
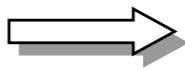


Step 3: add unique start / end

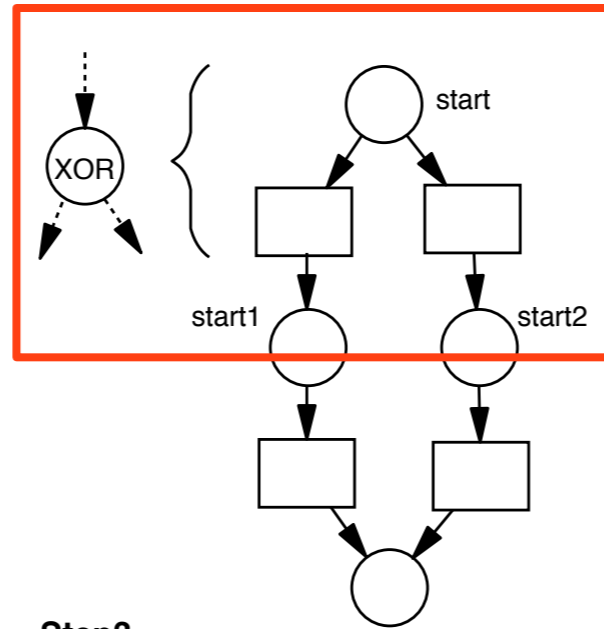
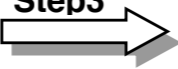
a)



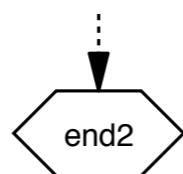
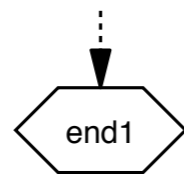
Step1 &
Step2



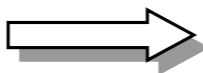
Step3



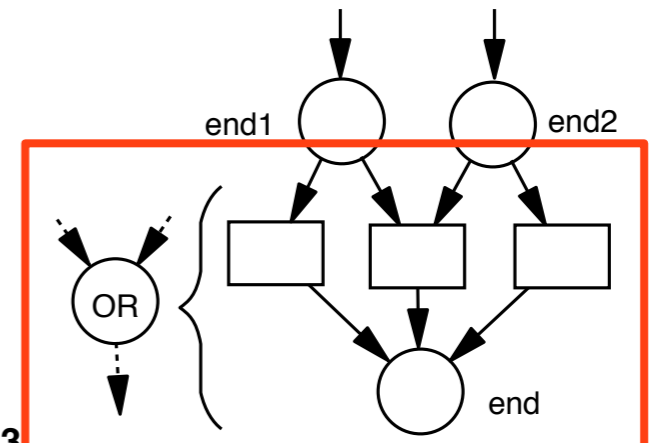
XOR start



Step1 &
Step2



(sometimes XOR/AND can be preferred)

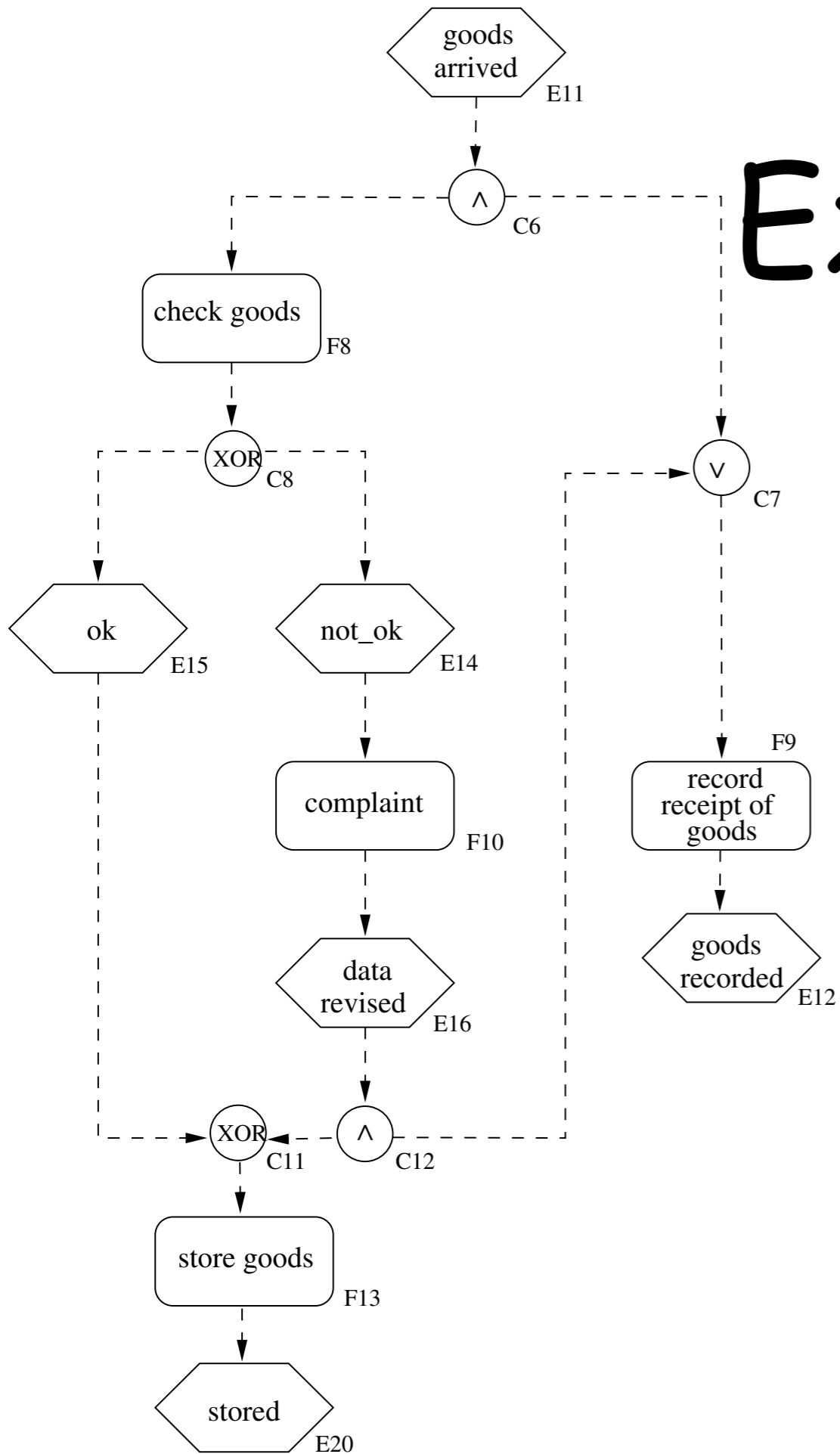


Step3



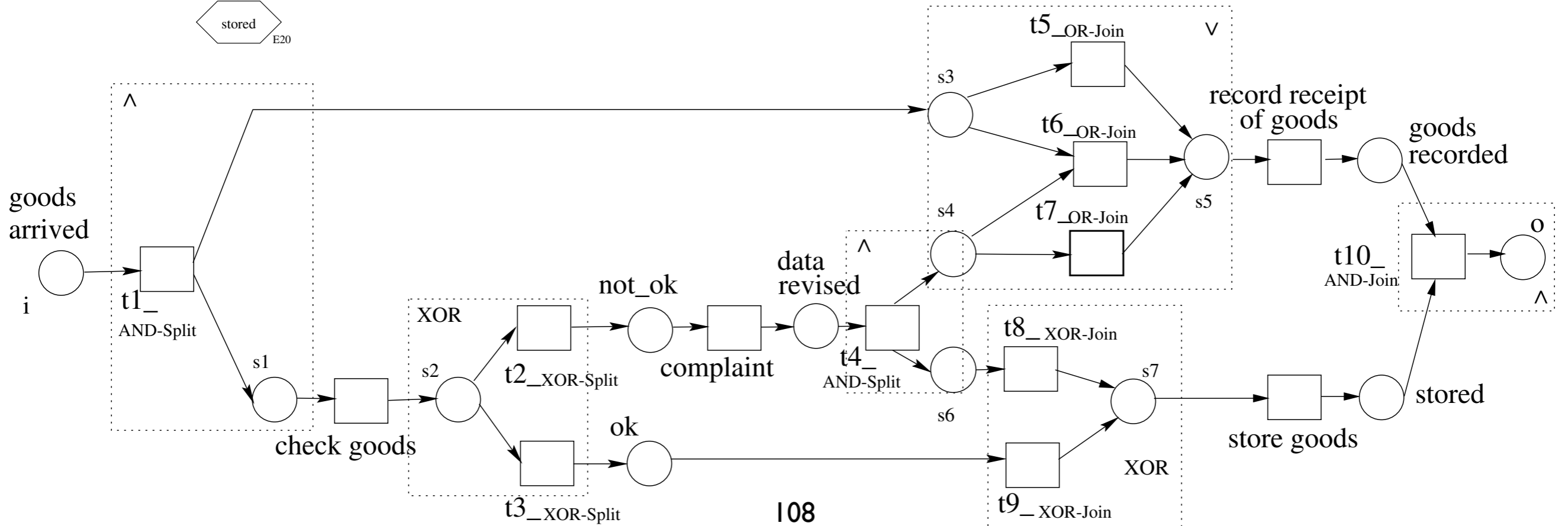
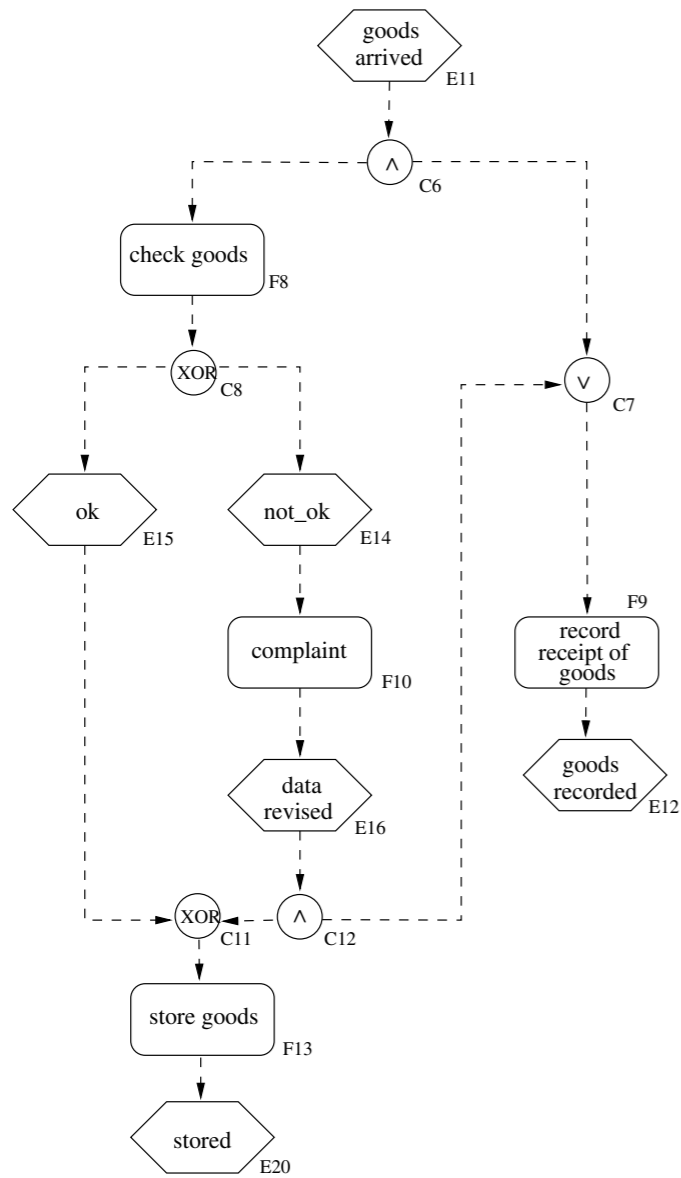
OR end

Example



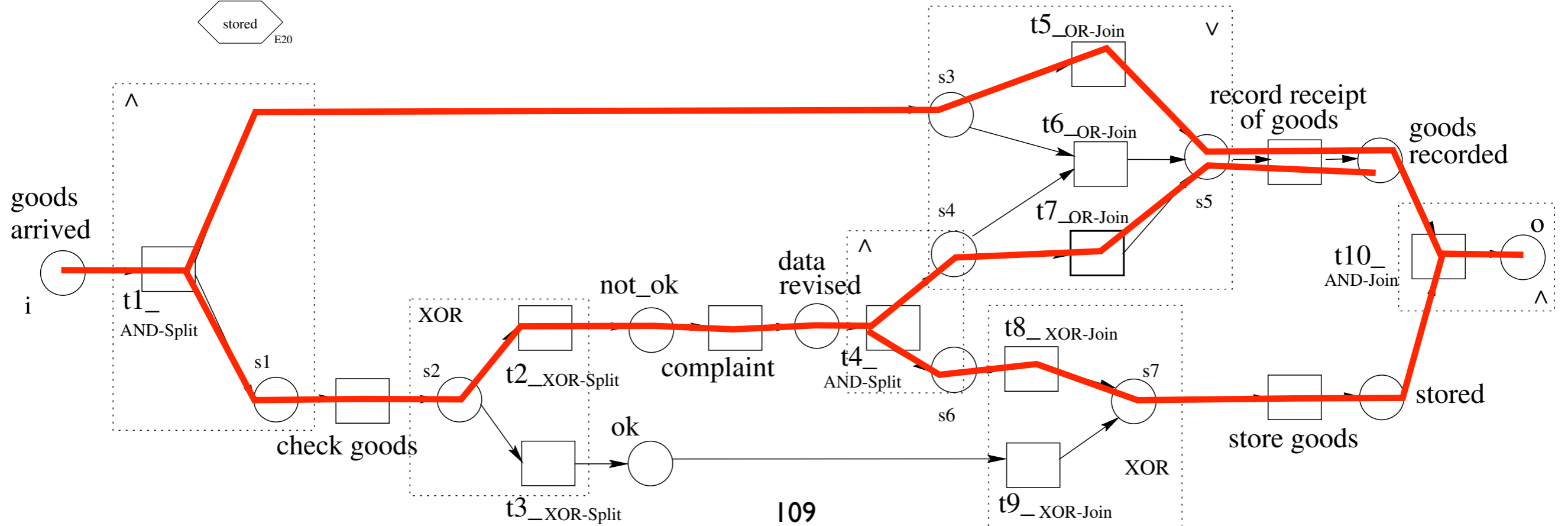
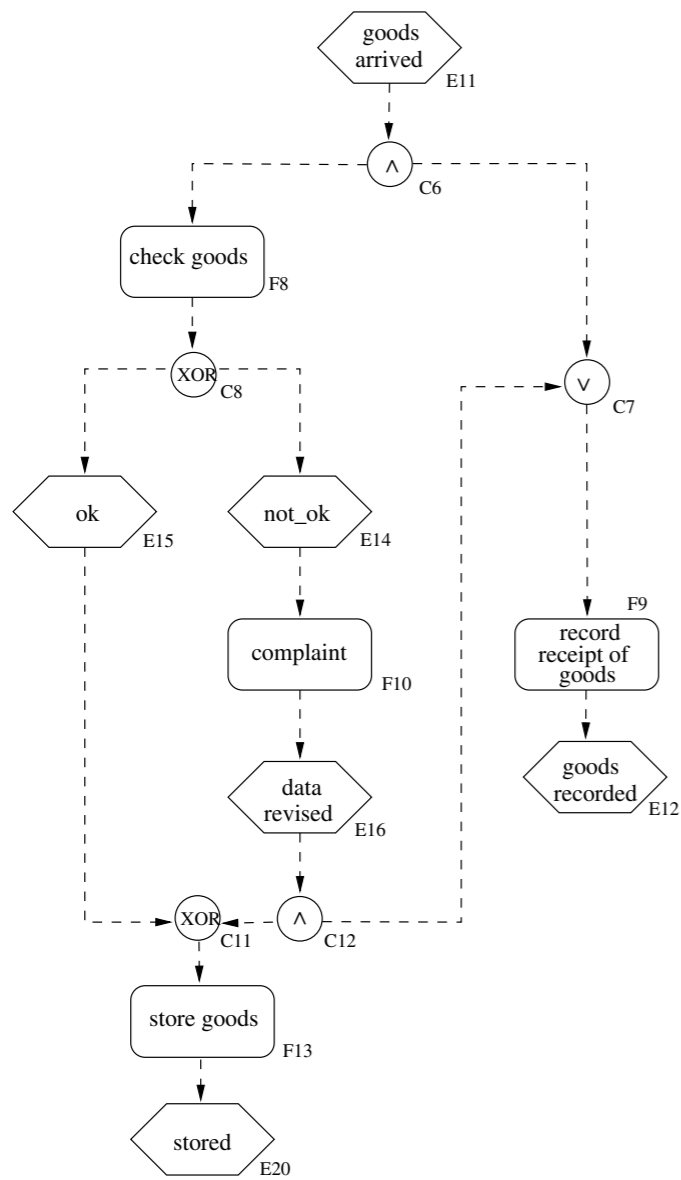
Sound?

Example



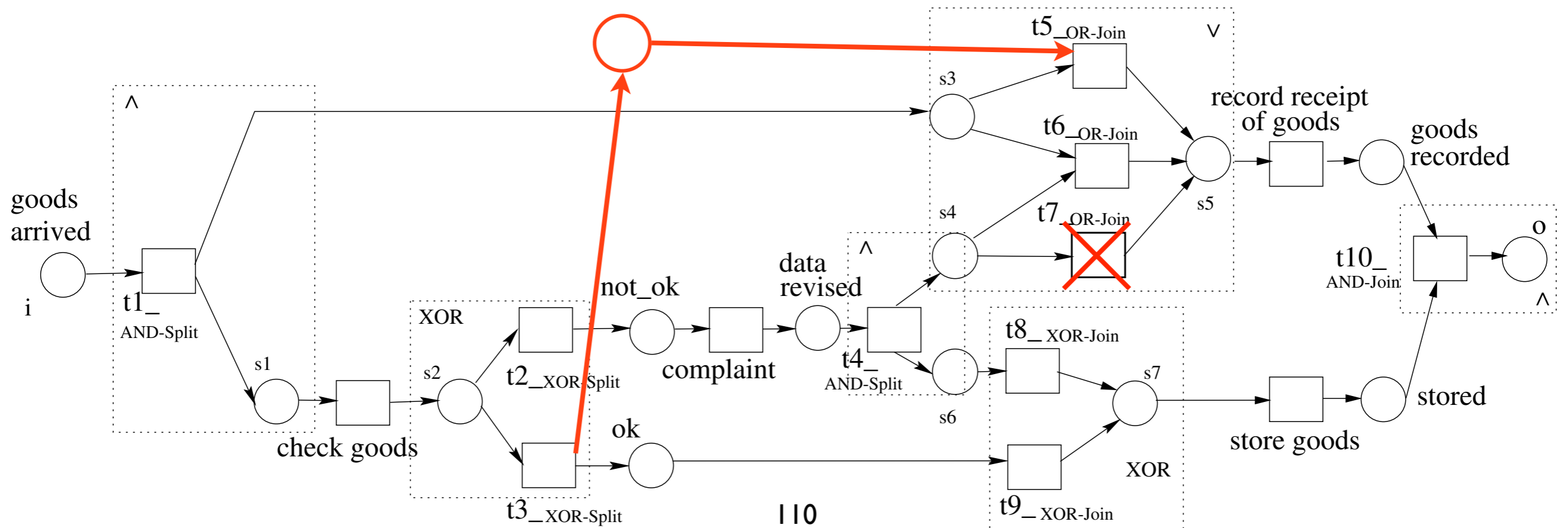
Example

Not sound!



Example

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



Relaxed soundness: formally

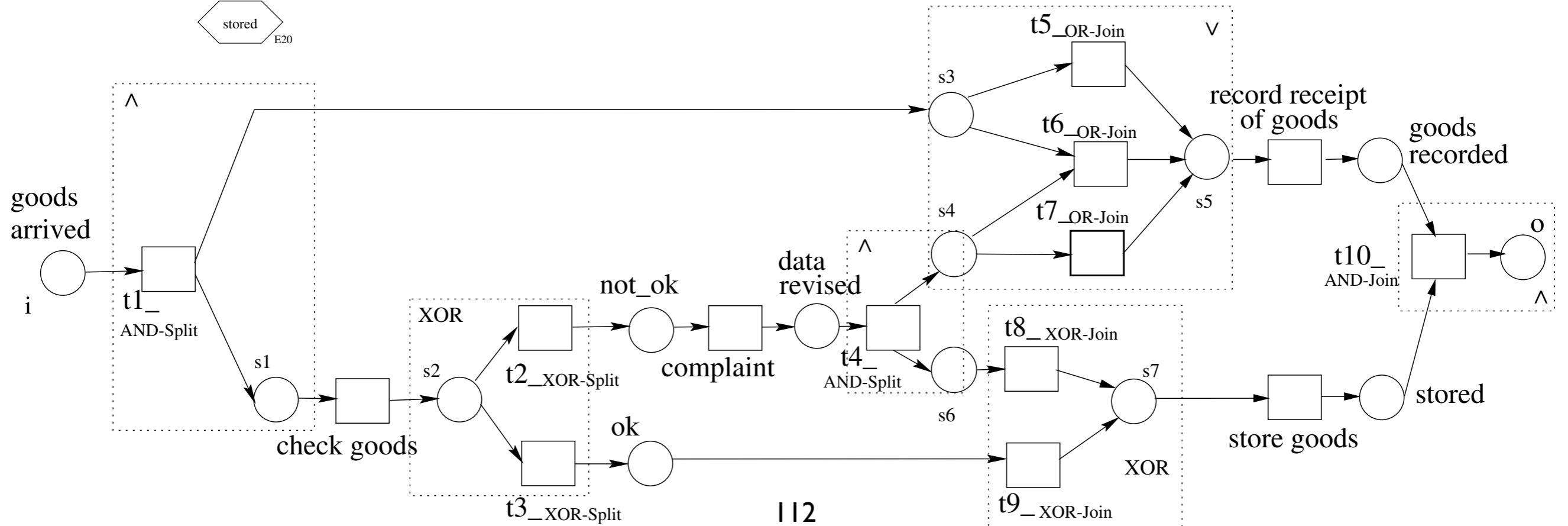
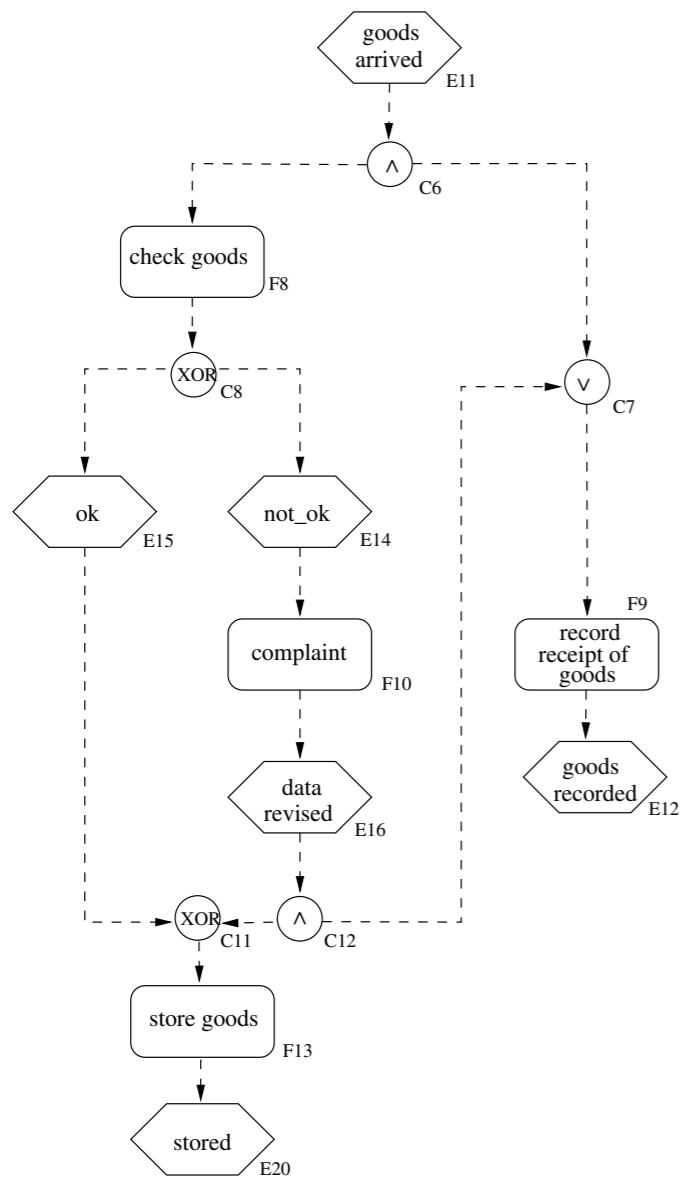
Definition: A WF net is **relaxed sound** if every transition belongs to a firing sequence that starts in state i and ends in state o

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

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

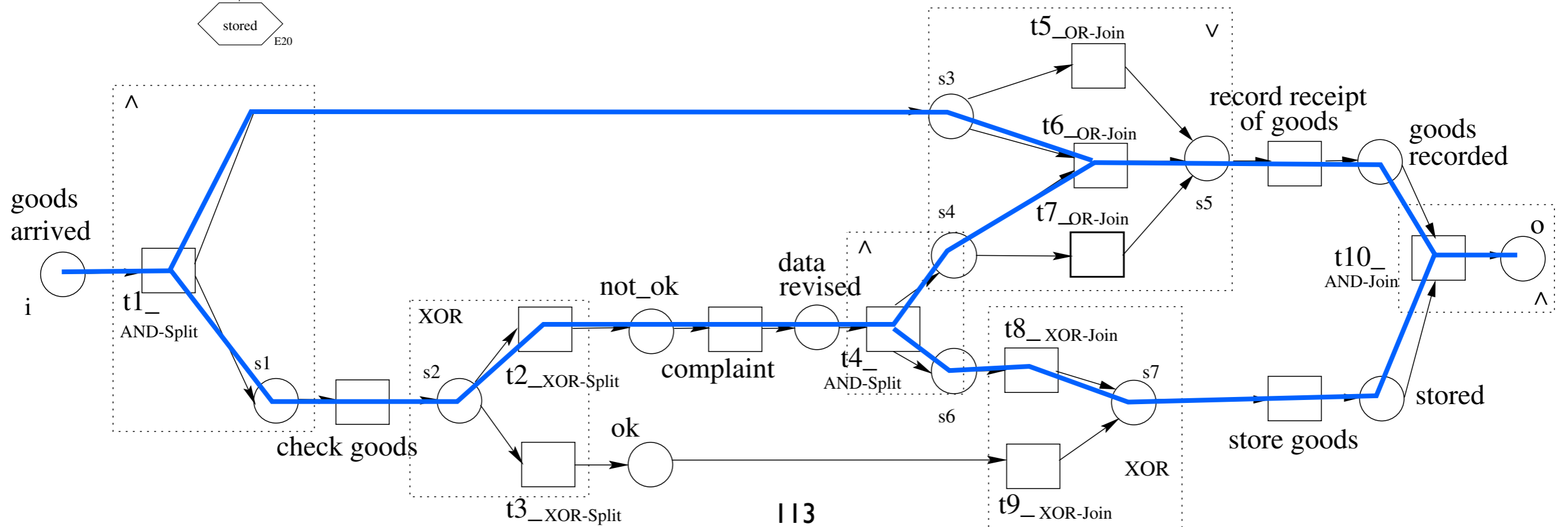
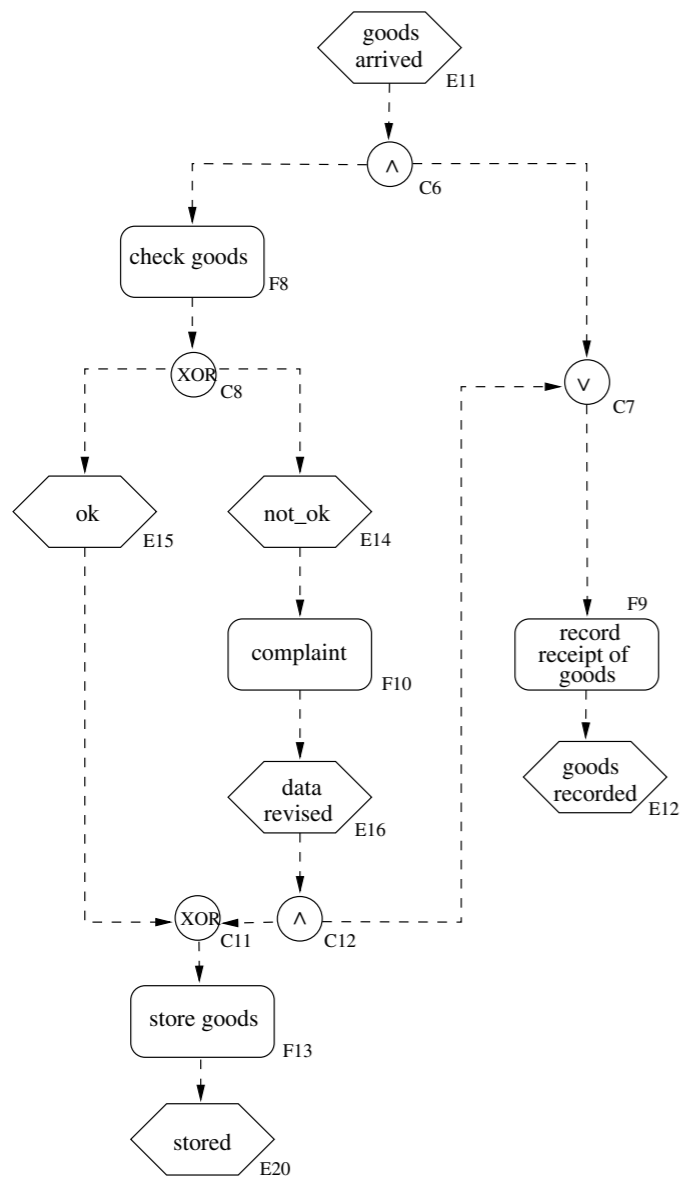
Example

Relaxed sound?



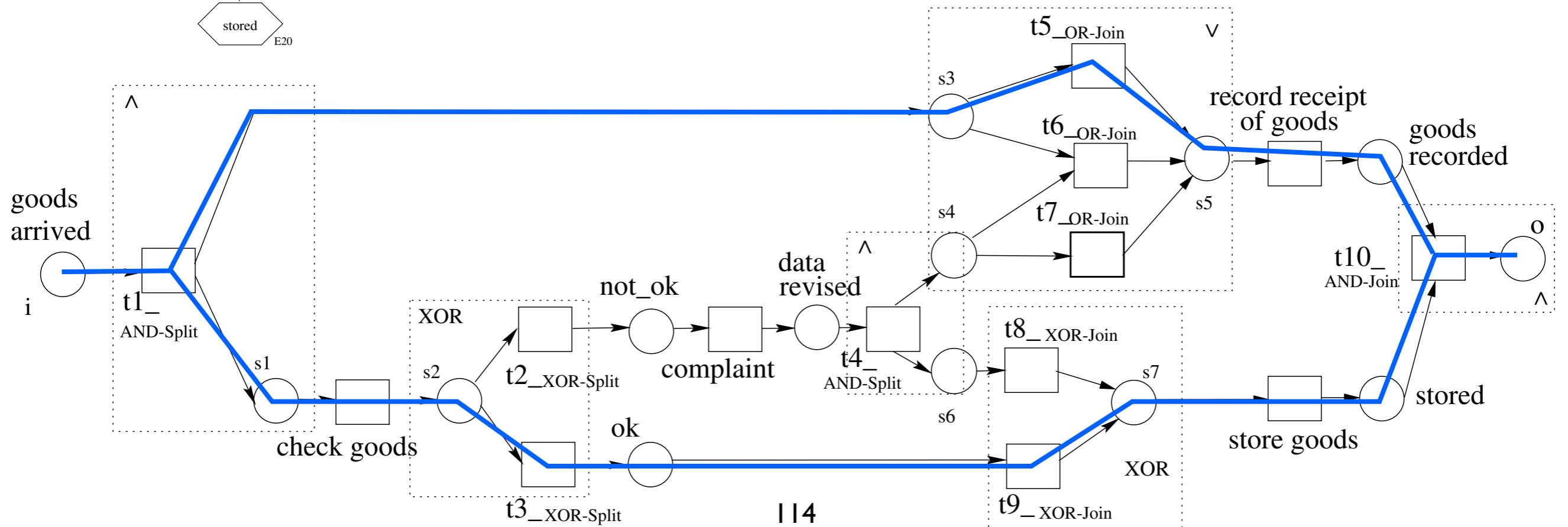
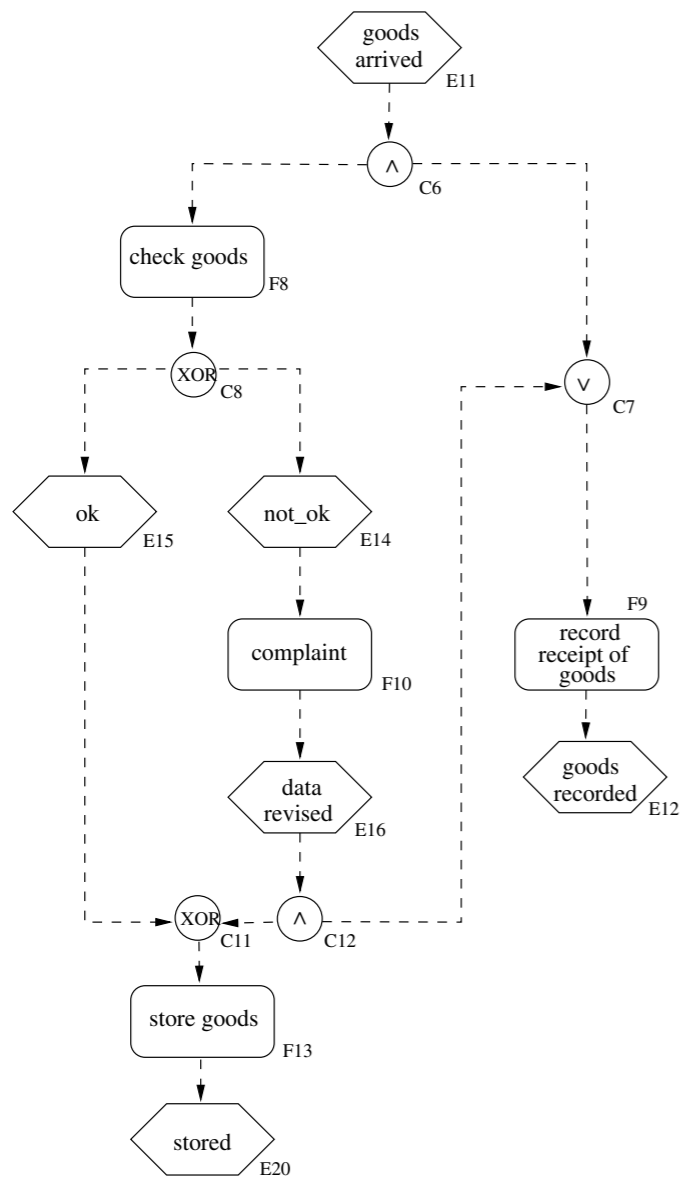
Example

Relaxed sound?



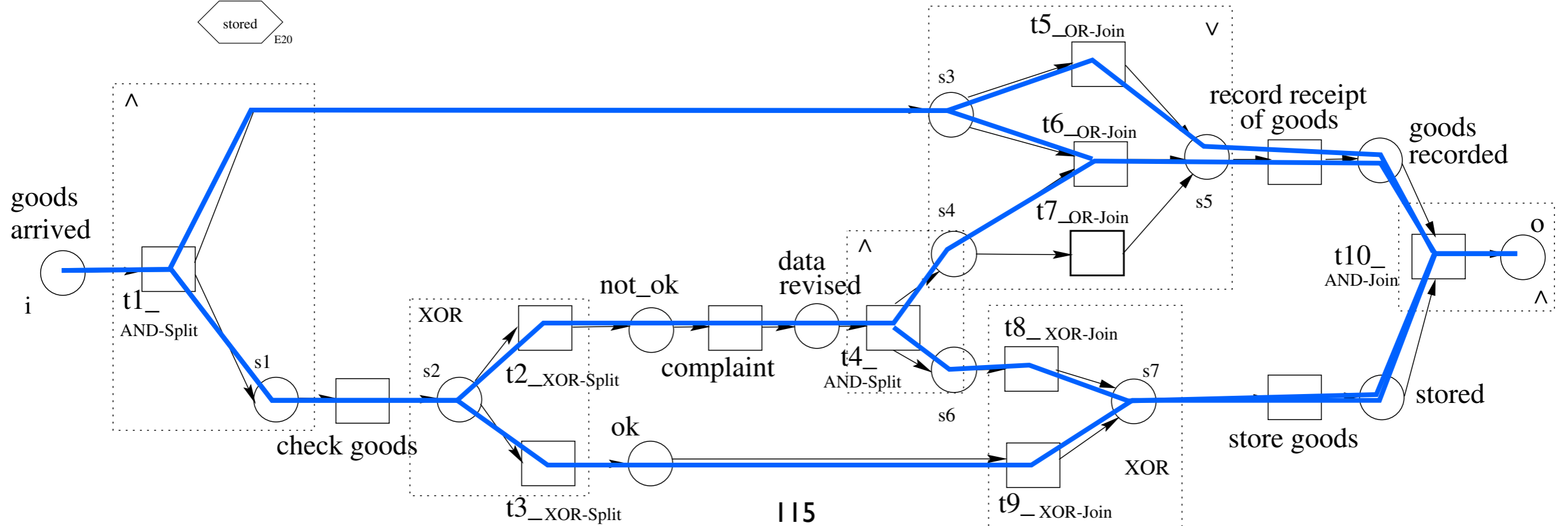
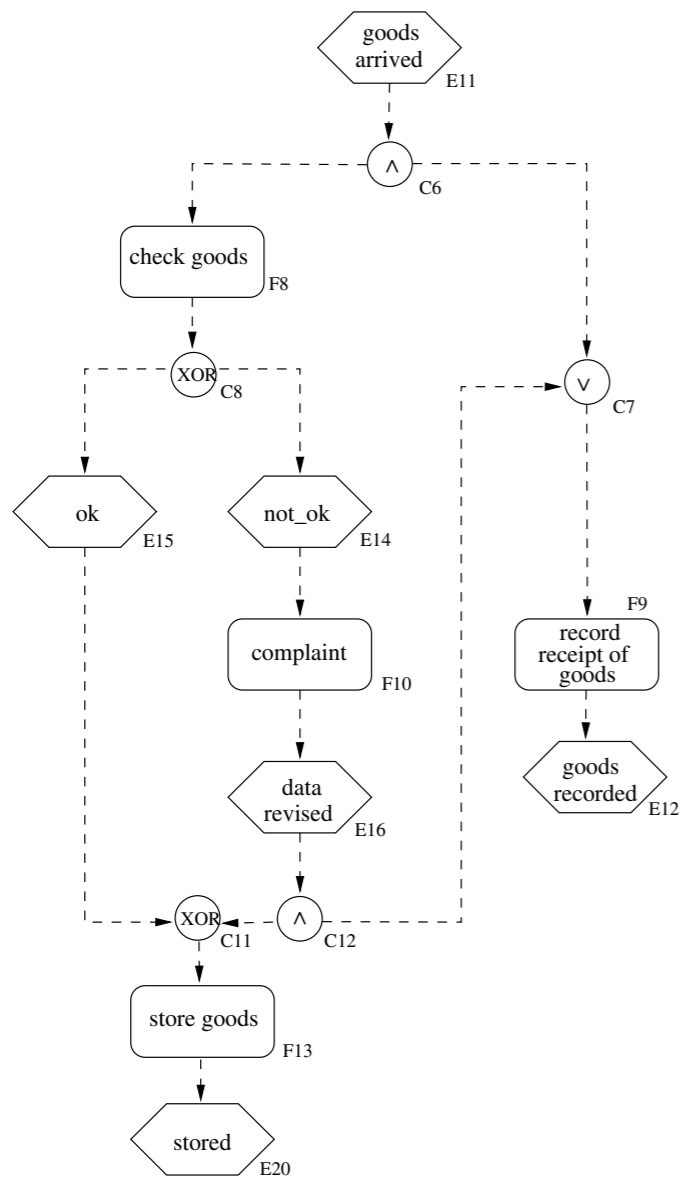
Example

Relaxed sound?



Example

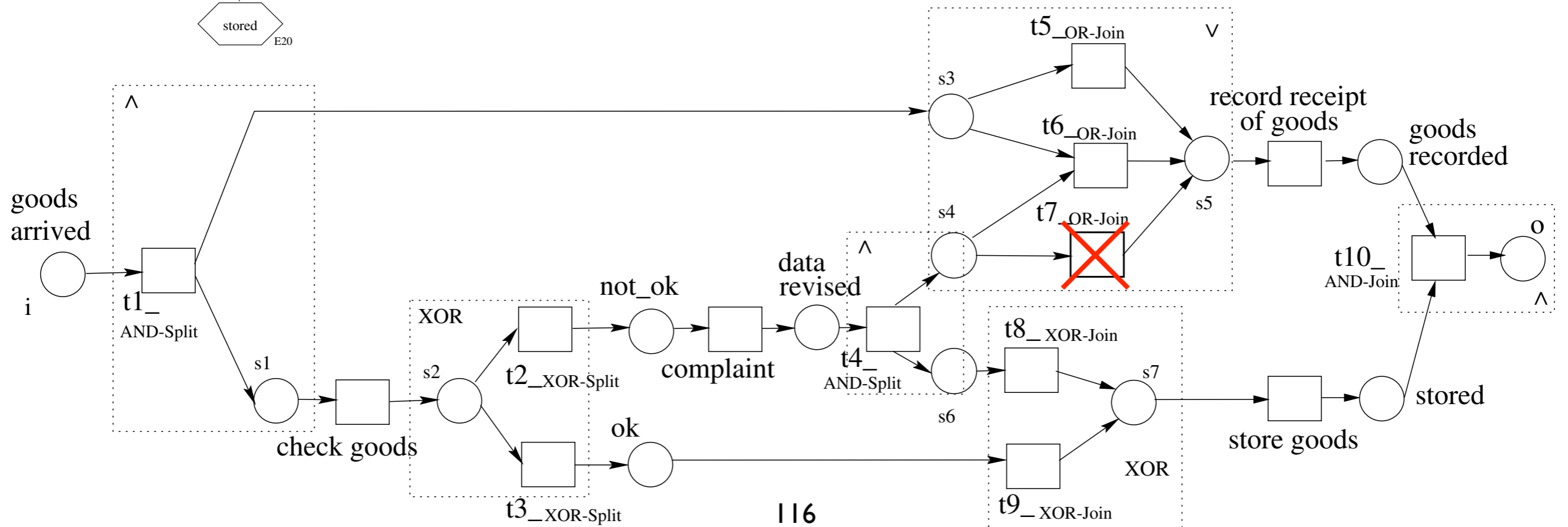
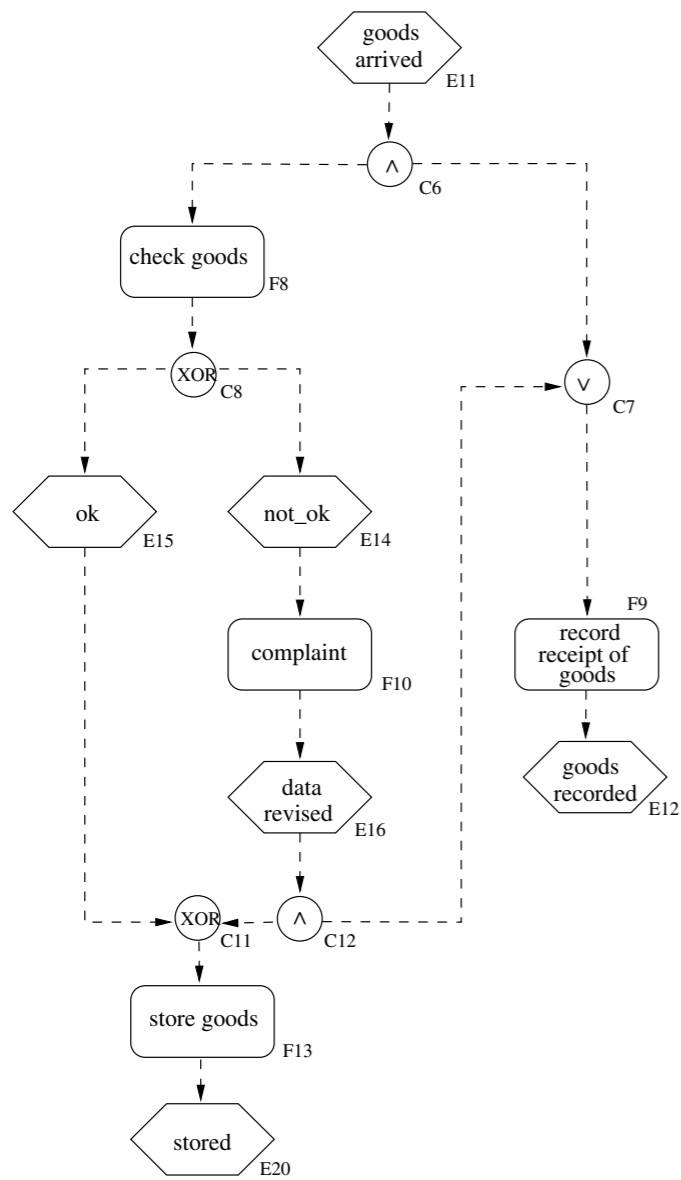
Relaxed sound?



Example

Not relaxed sound (as WF net)!

But relaxed sound as EPC
(all nodes are covered
by some sound execution)



Pros and Cons

If the WF net is **not relaxed sound**:
there are transitions that are not part of a
sound firing sequence

Hence their EPC counterparts need improvements

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

No equivalent characterization is known
that is more convenient to check

Open research problem...