

02291 System Integration

Goal-oriented Requirements Engineering

© Giovanni Meroni

Motivations

- Need to define the context in which an application works
 - Time
 - Place
 - Information
 - Operations
 - Stakeholders
- Need to define the motivations why an application works in a specific way
 - The same outcome could be reached in multiple ways
- Need to consider these aspects together, and not in isolation
- Goal-oriented requirements engineering help us do so

Goal-oriented Requirements Engineering

- Use of goals as a starting point to identify, elicit, document, specify and analyze requirements
- Goals are objectives that the system aims to achieve
- Goals are meant to specify why a requirement is needed, which tasks are required to reach it, and which ones may hamper its achievement

Zachman Framework

	DATA <i>If/for</i>	FUNCTION <i>How</i>	NETWORK <i>Where</i>	PEOPLE <i>Who</i>	TIME <i>When</i>	MOTIVATION <i>Why</i>	
SCOPE (CONTEXTUAL)	List of Things Important to the Business 	List of Processes the Business Performs 	List of Locations in which the Business Operates 	List of Organizations Important to the Business 	List of Events/Cycles Significant to the Business 	List of Business Goals/Strategies 	SCOPE (CONTEXTUAL)
Planner	ENTITY = Class of Business Thing	Process = Class of Business Process	Node = Major Business Location	People = Major Organization Unit	Time = Major Business Event/Cycle	Ends/Means = Major Business Goal/Strategy	Planner
BUSINESS MODEL (CONCEPTUAL)	e.g. Semantic Model 	e.g. Business Process Model 	e.g. Business Logistics System 	e.g. Work Flow Model 	e.g. Master Schedule 	e.g. Business Plan 	BUSINESS MODEL (CONCEPTUAL)
Owner	Ent = Business Entity Reln = Business Relationship	Proc. = Business Process I/O = Business Resources	Node = Business Location Link = Business Linkage	People = Organization Unit Work = Work Product	Time = Business Event Cycle = Business Cycle	End = Business Objective Means = Business Strategy	Owner
SYSTEM MODEL (LOGICAL)	e.g. Logical Data Model 	e.g. Application Architecture 	e.g. Distributed System Architecture 	e.g. Human Interface Architecture 	e.g. Processing Structure 	e.g., Business Rule Model 	SYSTEM MODEL (LOGICAL)
Designer	Ent = Data Entity Reln = Data Relationship	Proc. = Application Function I/O = User Views	Node = IS Function (Processor, Storage, etc.) Link = Line Characteristics	People = Role Work = Deliverable	Time = System Event Cycle = Processing Cycle	End = Structural Assertion Means = Action Assertion	Designer
TECHNOLOGY MODEL (PHYSICAL)	e.g. Physical Data Model 	e.g. System Design 	e.g. Technology Architecture 	e.g. Presentation Architecture 	e.g. Control Structure 	e.g. Rule Design 	TECHNOLOGY MODEL (PHYSICAL)
Builder	Ent = Segment/Table/etc. Reln = Pointer/Key/etc.	Proc. = Computer Function I/O = Data Elements/Sets	Node = Hardware/Systems Software Link = Line Specifications	People = User Work = Screen Format	Time = Execute Cycle = Component Cycle	End = Condition Means = Action	Builder
DETAILED REPRESENTATIONS (OUT-OF-CONTEXT)	e.g. Data Definition 	e.g. Program 	e.g. Network Architecture 	e.g. Security Architecture 	e.g. Timing Definition 	e.g. Rule Specification 	DETAILED REPRESENTATIONS (OUT-OF-CONTEXT)
Sub-Contractor	Ent = Field Reln = Address	Proc. = Language Statement I/O = Control Block	Node = Address Link = Protocol	People = Identity Work = Job	Time = Interrupt Cycle = Machine Cycle	End = Sub-condition Means = Stop	Sub-Contractor
FUNCTIONING ENTERPRISE	e.g. DATA	e.g. FUNCTION	e.g. NETWORK	e.g. ORGANIZATION	e.g. SCHEDULE	e.g. STRATEGY	FUNCTIONING ENTERPRISE

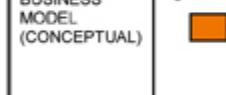
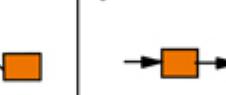
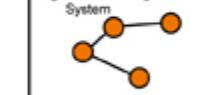
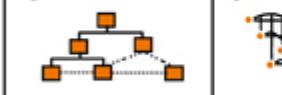
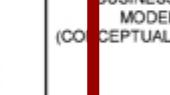
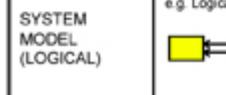
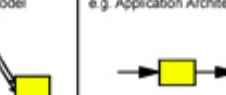
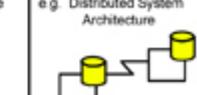
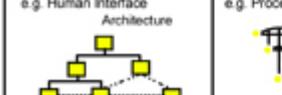
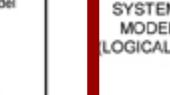
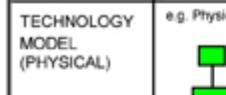
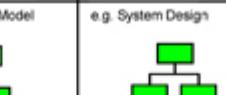
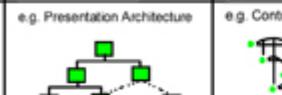
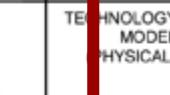
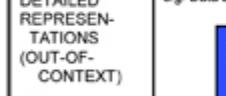
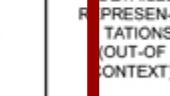
© John A. Zachman, Zachman International

Zachman Framework

	DATA What	FUNCTION How	NETWORK Where	PEOPLE Who	TIME When	MOTIVATION Why	
SCOPE (CONTEXTUAL)	List of Things Important to the Business 	List of Processes the Business Performs 	List of Locations in which the Business Operates 	List of Organizations Important to the Business 	List of Events/Cycles Significant to the Business 	List of Business Goals/Strategies 	SCOPE (CONTEXTUAL) Planner
BUSINESS MODEL (CONCEPTUAL)	e.g. Semantic Model 	e.g. Business Process Model 	e.g. Business Logistics System 	e.g. Work Flow Model 	e.g. Master Schedule 	e.g. Business Plan 	BUSINESS MODEL (CONCEPTUAL) Owner
SYSTEM MODEL (LOGICAL)	e.g. Logical Data Model 	e.g. Application Architecture 	e.g. Distributed System Architecture 	e.g. Human Interface Architecture 	e.g. Processing Structure 	e.g., Business Rule Model 	SYSTEM MODEL (LOGICAL) Designer
TECHNOLOGY MODEL (PHYSICAL)	e.g. Physical Data Model 	e.g. System Design 	e.g. Technology Architecture 	e.g. Presentation Architecture 	e.g. Control Structure 	e.g. Rule Design 	TECHNOLOGY MODEL (PHYSICAL) Builder
DETAILED REPRESENTATIONS (OUT-OF-CONTEXT)	e.g. Data Definition 	e.g. Program 	e.g. Network Architecture 	e.g. Security Architecture 	e.g. Timing Definition 	e.g. Rule Specification 	DETAILED REPRESENTATIONS (OUT-OF-CONTEXT) Sub-Contractor
FUNCTIONING ENTERPRISE	e.g. DATA	e.g. FUNCTION	e.g. NETWORK	e.g. ORGANIZATION	e.g. SCHEDULE	e.g. STRATEGY	FUNCTIONING ENTERPRISE

© John A. Zachman, Zachman International

Zachman Framework

	DATA What	FUNCTION How	NETWORK Where	PEOPLE Who	TIME When	INTERFACES With	SCOPE (CONTEXTUAL)
SCOPE (CONTEXTUAL)	List of Things Important to the Business 	List of Processes the Business Performs 	List of Locations in which the Business Operates 	List of Organizations Important to the Business 	List of Events/Cycles Significant to the Business 	List of Business Goals/Strategies 	SCOPE (CONTEXTUAL)
Planner	ENTITY = Class of Business Thing	Process = Class of Business Process	Node = Major Business Location	People = Major Organization Unit	Time = Major Business Event/Cycle	Ends/Means = Major Business Goal/Strategy	Planner
BUSINESS MODEL (CONCEPTUAL)	e.g. Semantic Model 	e.g. Business Process Model 	e.g. Business Logistics System 	e.g. Work Flow Model 	e.g. Master Schedule 	e.g. Business Plan 	BUSINESS MODEL (CONCEPTUAL)
Owner	Ent = Business Entity Reln = Business Relationship	Proc = Business Process I/O = Business Resources	Node = Business Location Link = Business Linkage	People = Organization Unit Work = Work Product	Time = Business Event Cycle = Business Cycle	End = Business Objective Means = Business Strategy	Owner
SYSTEM MODEL (LOGICAL)	e.g. Logical Data Model 	e.g. Application Architecture 	e.g. Distributed System Architecture 	e.g. Human Interface Architecture 	e.g. Processing Structure 	e.g., Business Rule Model 	SYSTEM MODEL (LOGICAL)
Designer	Ent = Data Entity Reln = Data Relationship	Proc = Application Function I/O = User Views	Node = IS Function (Processor, Storage, etc.) Link = Line Characteristics	People = Role Work = Deliverable	Time = System Event Cycle = Processing Cycle	End = Structural Assertion Means = Action Assertion	Designer
TECHNOLOGY MODEL (PHYSICAL)	e.g. Physical Data Model 	e.g. System Design 	e.g. Technology Architecture 	e.g. Presentation Architecture 	e.g. Control Structure 	e.g. Rule Design 	TECHNOLOGY MODEL (PHYSICAL)
Builder	Ent = Segment/Table/etc. Reln = Pointer/Key/etc.	Proc = Computer Function I/O = Data Elements/Sets	Node = Hardware/Systems Software Link = Line Specifications	People = User Work = Screen Format	Time = Execute Cycle = Component Cycle	End = Condition Means = Action	Builder
DETAILED REPRESENTATIONS (OUT-OF-CONTEXT)	e.g. Data Definition 	e.g. Program 	e.g. Network Architecture 	e.g. Security Architecture 	e.g. Timing Definition 	e.g. Rule Specification 	DETAILED REPRESENTATIONS (OUT-OF-CONTEXT)
Sub-Contractor	Ent = Field Reln = Address	Proc = Language Statement I/O = Control Block	Node = Address Link = Protocol	People = Identity Work = Job	Time = Interrupt Cycle = Machine Cycle	End = Sub-condition Means = Stop	Sub-Contractor
FUNCTIONING ENTERPRISE	e.g. DATA	e.g. FUNCTION	e.g. NETWORK	e.g. ORGANIZATION	e.g. SCHEDULE	e.g. STRATEGY	FUNCTIONING ENTERPRISE

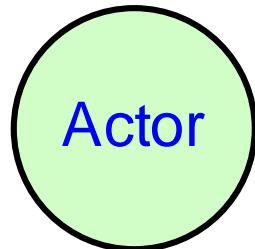
© John A. Zachman, Zachman International

Goal modelling

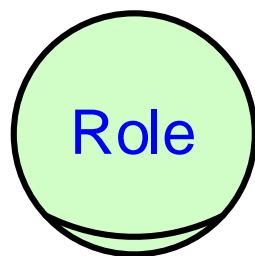
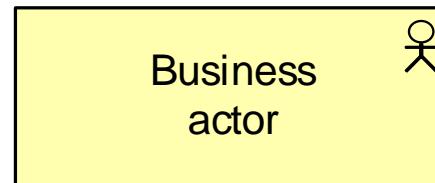
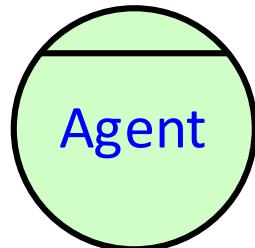
- Different languages exist to support goal-oriented requirements engineering:
 - KAOS
 - EEML
 - I-star
 - ArchiMate

Participants

I-star



ArchiMate



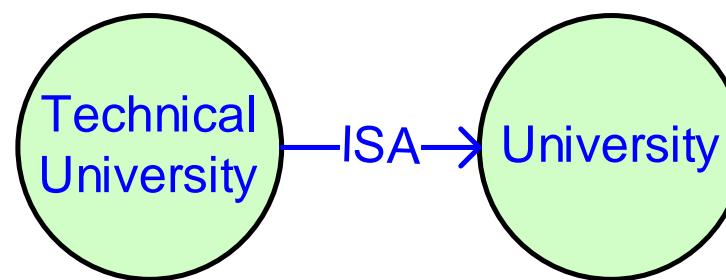
The role of an individual, team or organisation (or classes thereof) that represents their interest in the outcome of the application.

A business entity that is capable of performing behaviour.

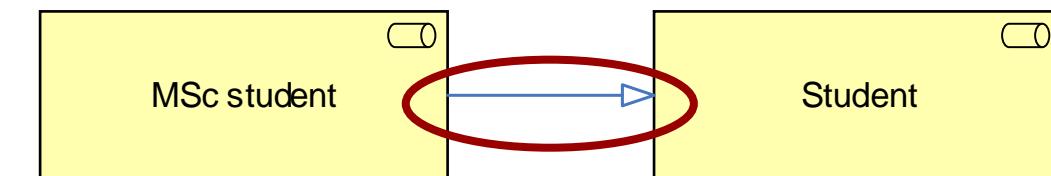
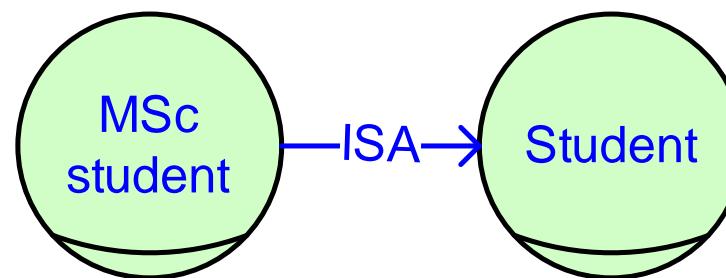
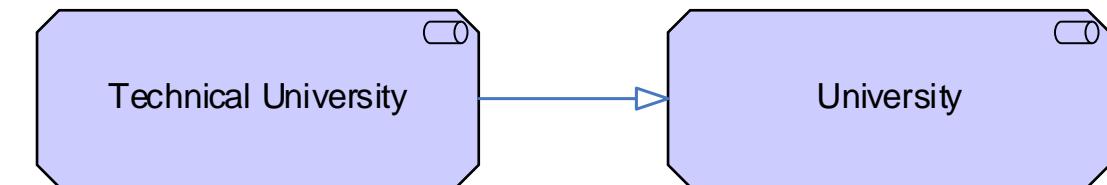
the responsibility for performing specific behaviour, or the part one plays in a particular action or event.

Specialization

I-star



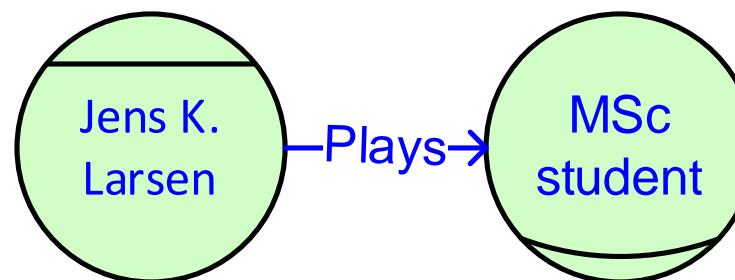
ArchiMate



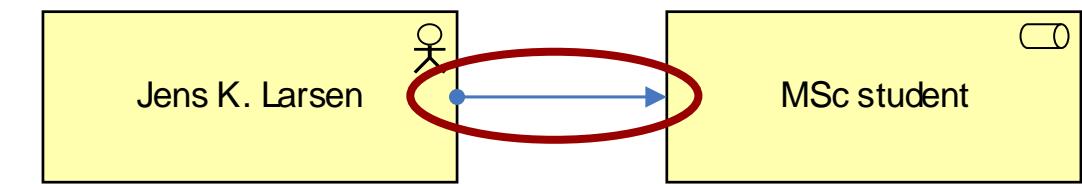
Specialization relation

Playing a role

I-star



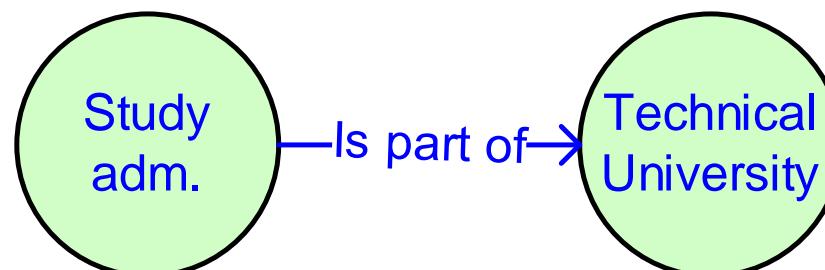
ArchiMate



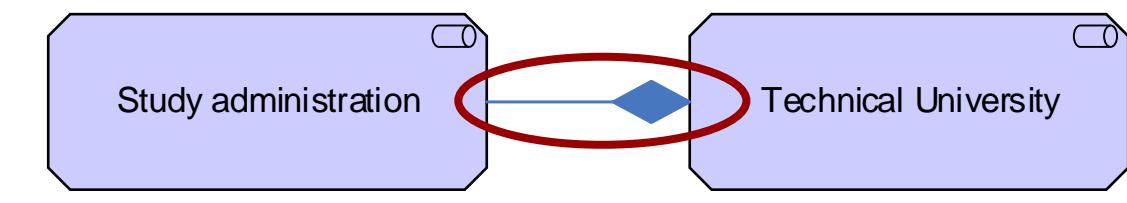
Assignment relation

Composition

I-star



ArchiMate



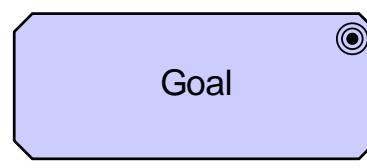
Composition relation

Intentional elements

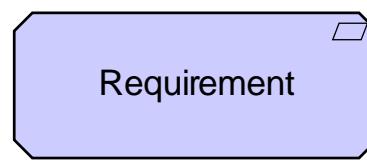
I-star



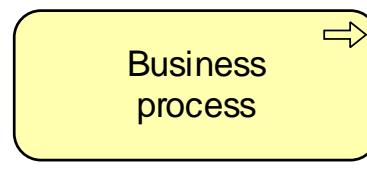
ArchiMate



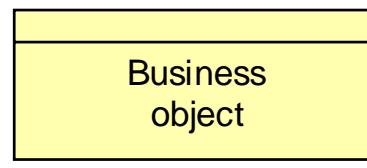
A high-level statement of intent, direction or desired end state for an organisation and its stakeholders.



a statement of need that must be met by the architecture.

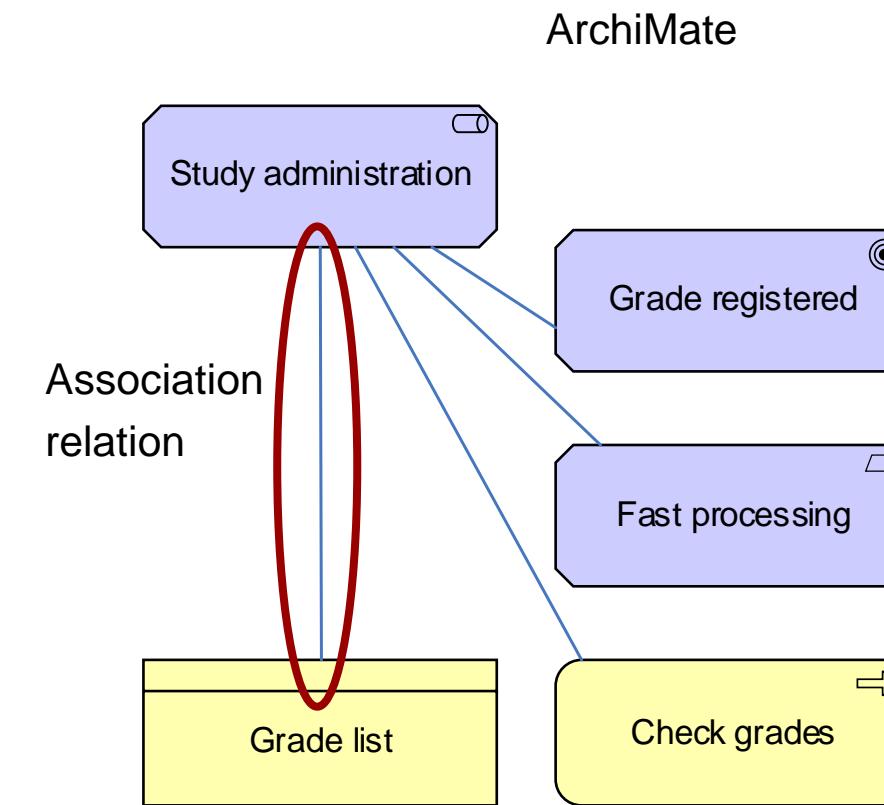
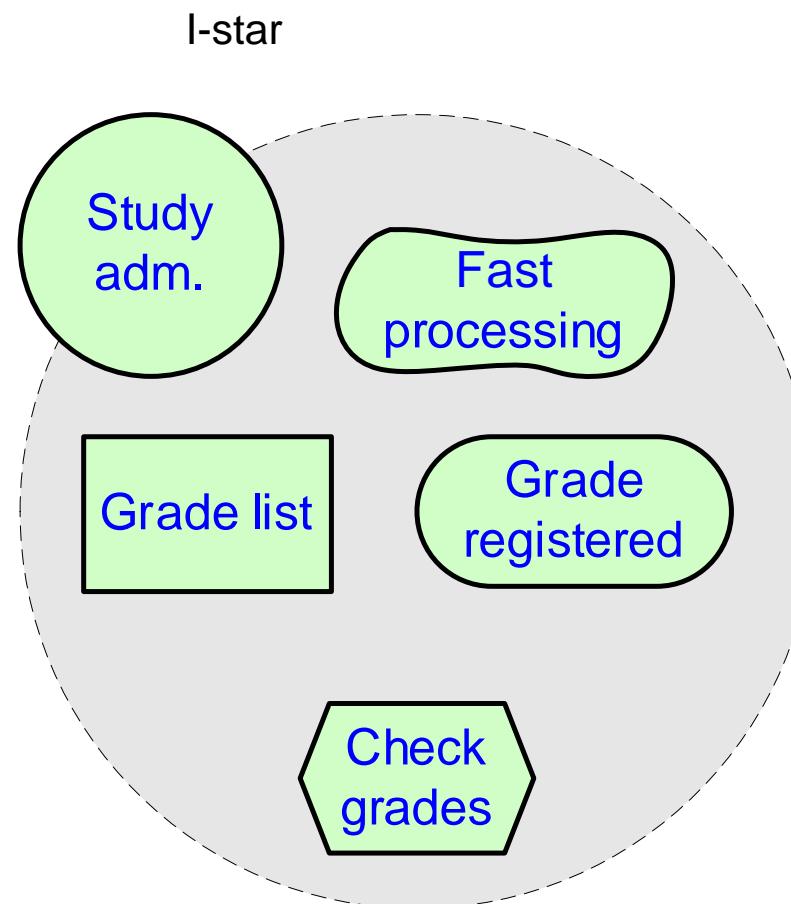


A set of actions that achieves a specific outcome.



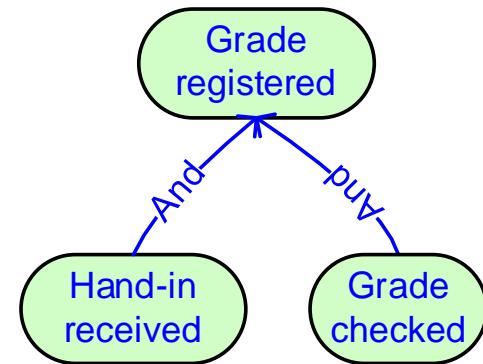
A concept used within a particular business domain

Associating intentional elements to participants

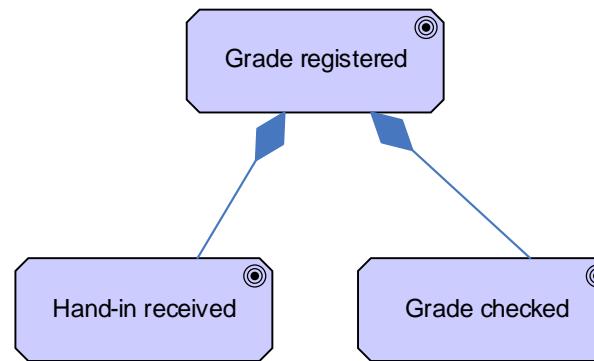


Refining goals

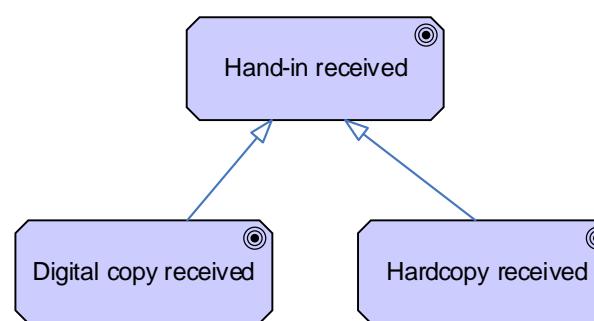
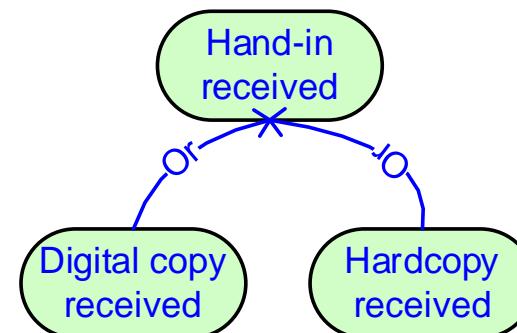
I-star



ArchiMate



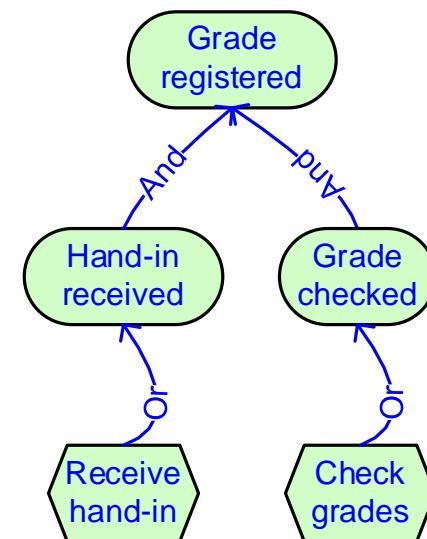
Goal decomposition



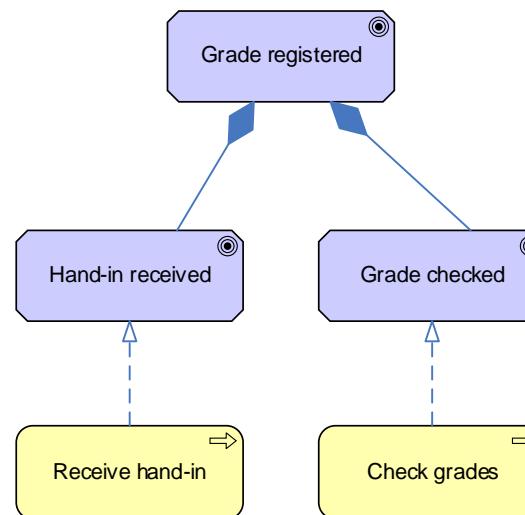
Goal specialization

Refining goals

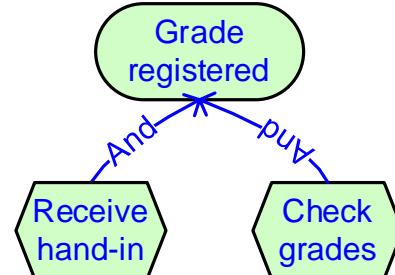
I-star



ArchiMate



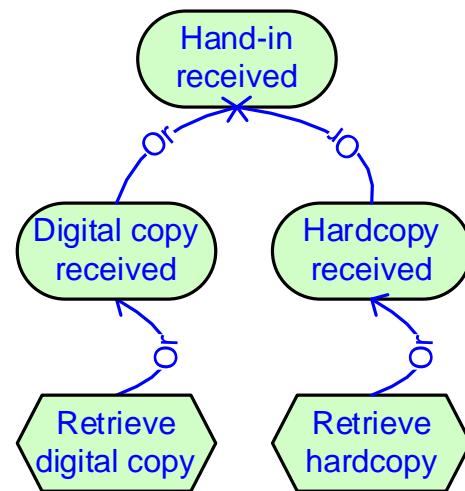
Cannot be modeled



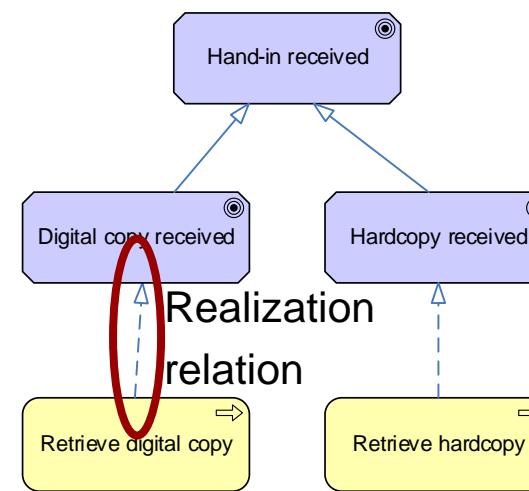
Goal realization (inclusive)

Refining goals

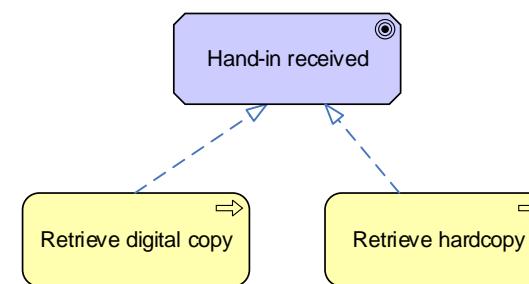
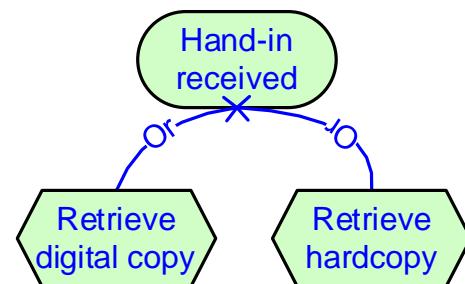
I-star



ArchiMate

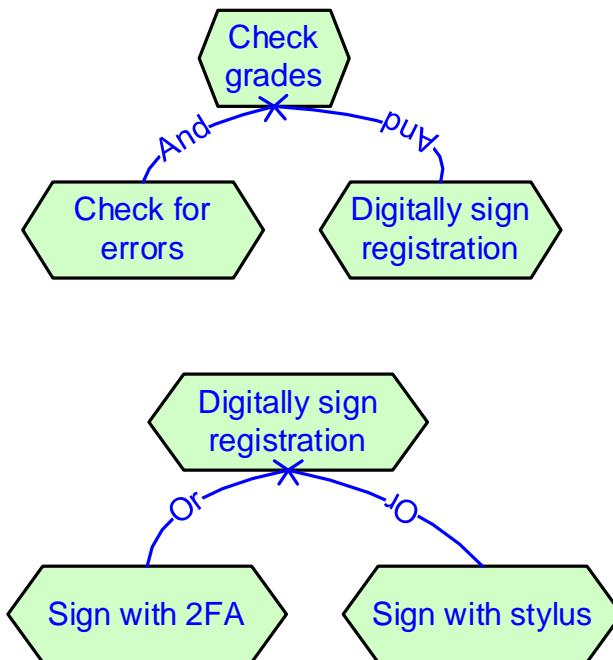


Goal realization (exclusive)

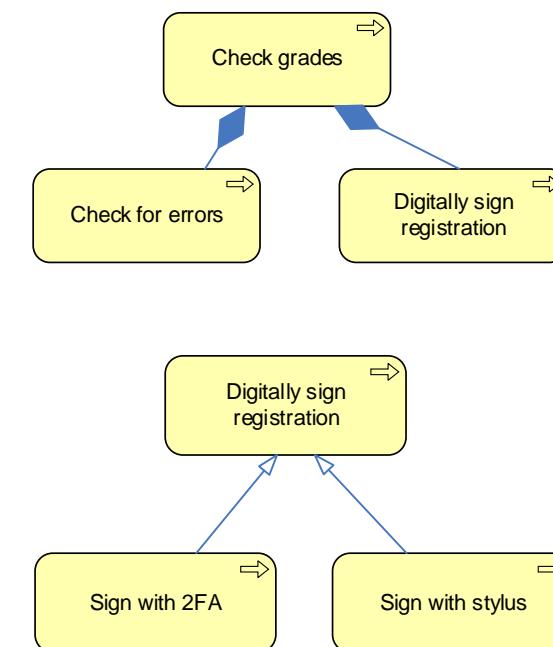


Refining tasks

I-star



ArchiMate



Task decomposition

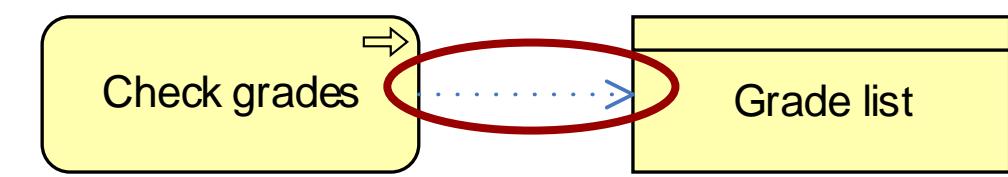
Task specialization

Resources for task execution

I-star



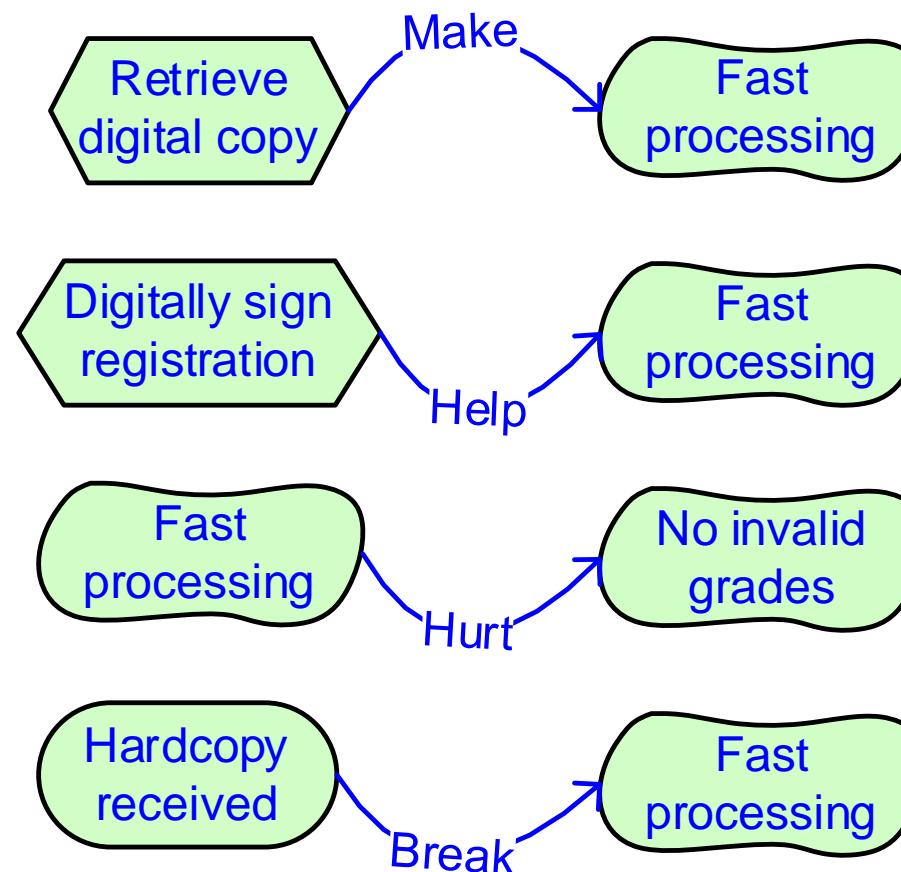
ArchiMate



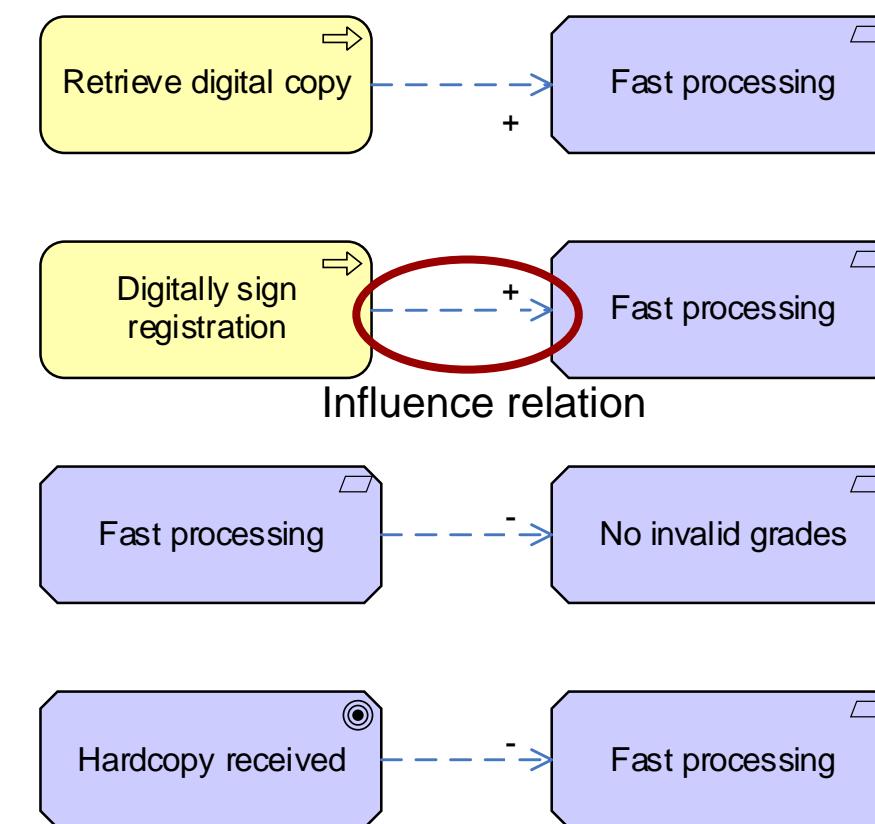
Access relation

Contribution links

I-star

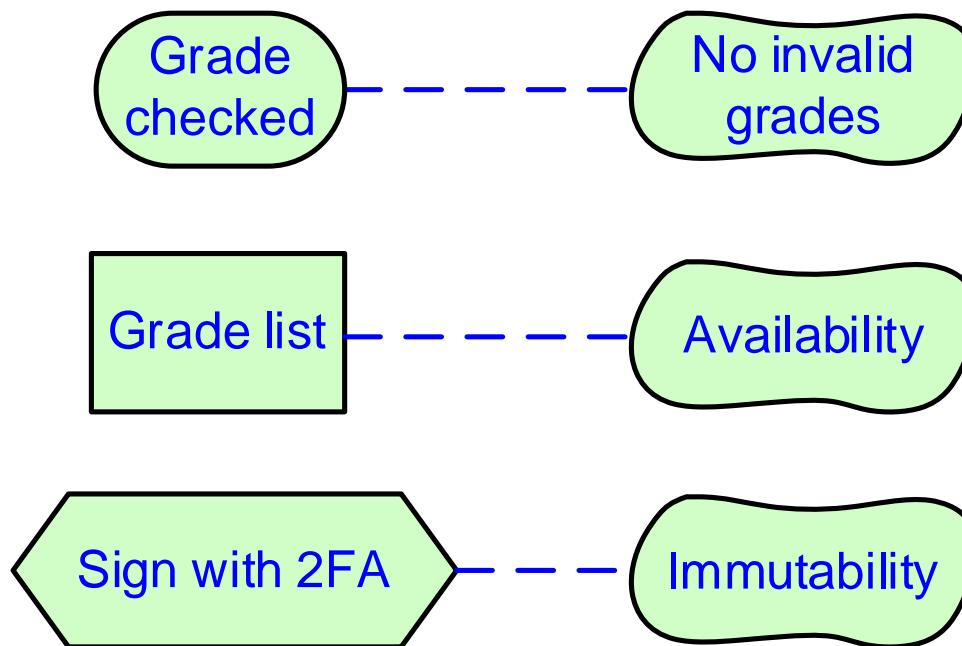


ArchiMate

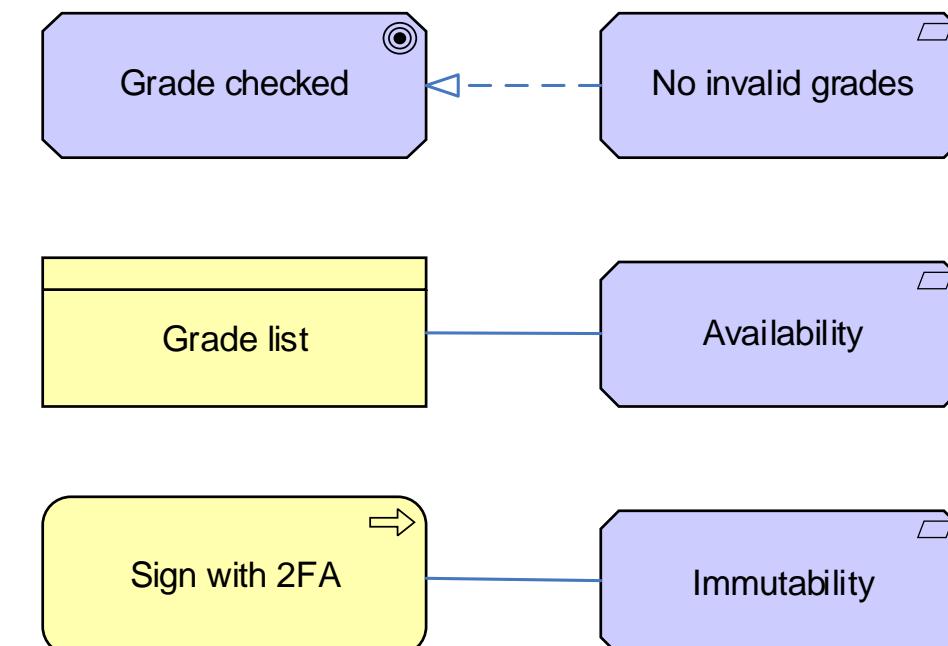


Qualification

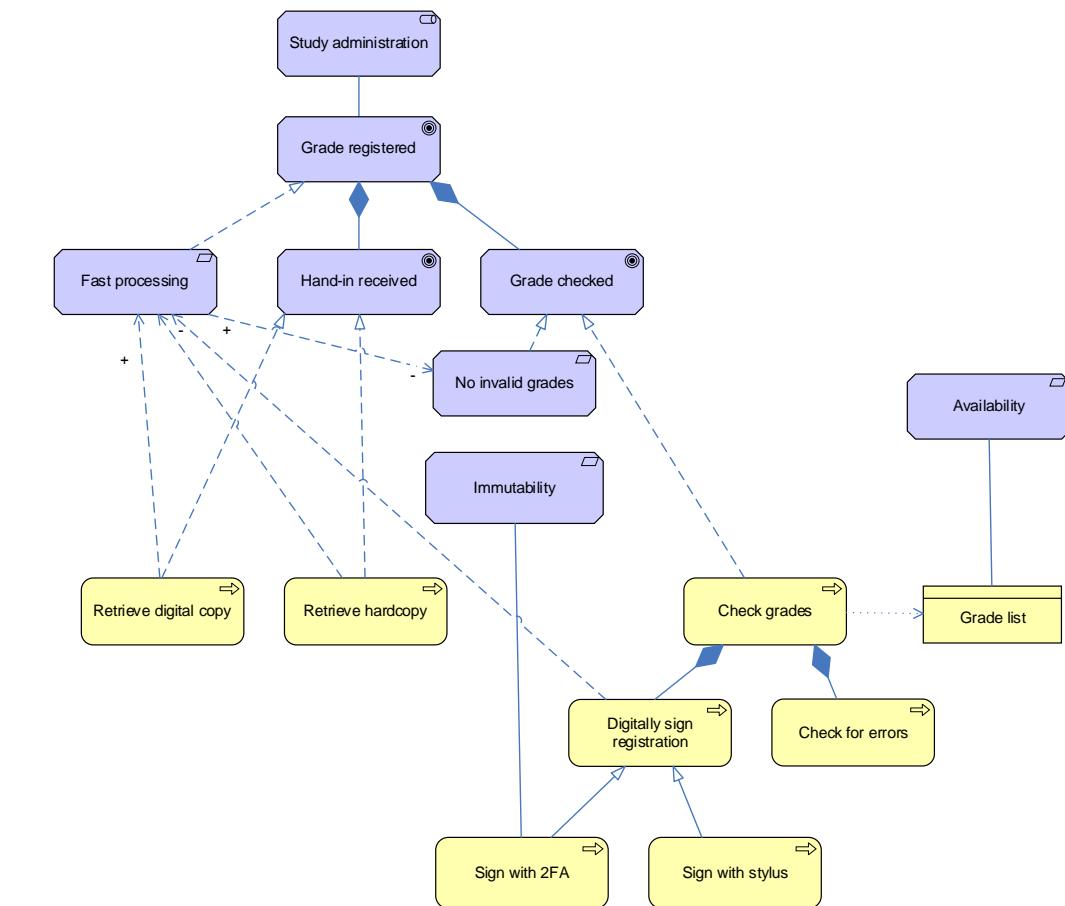
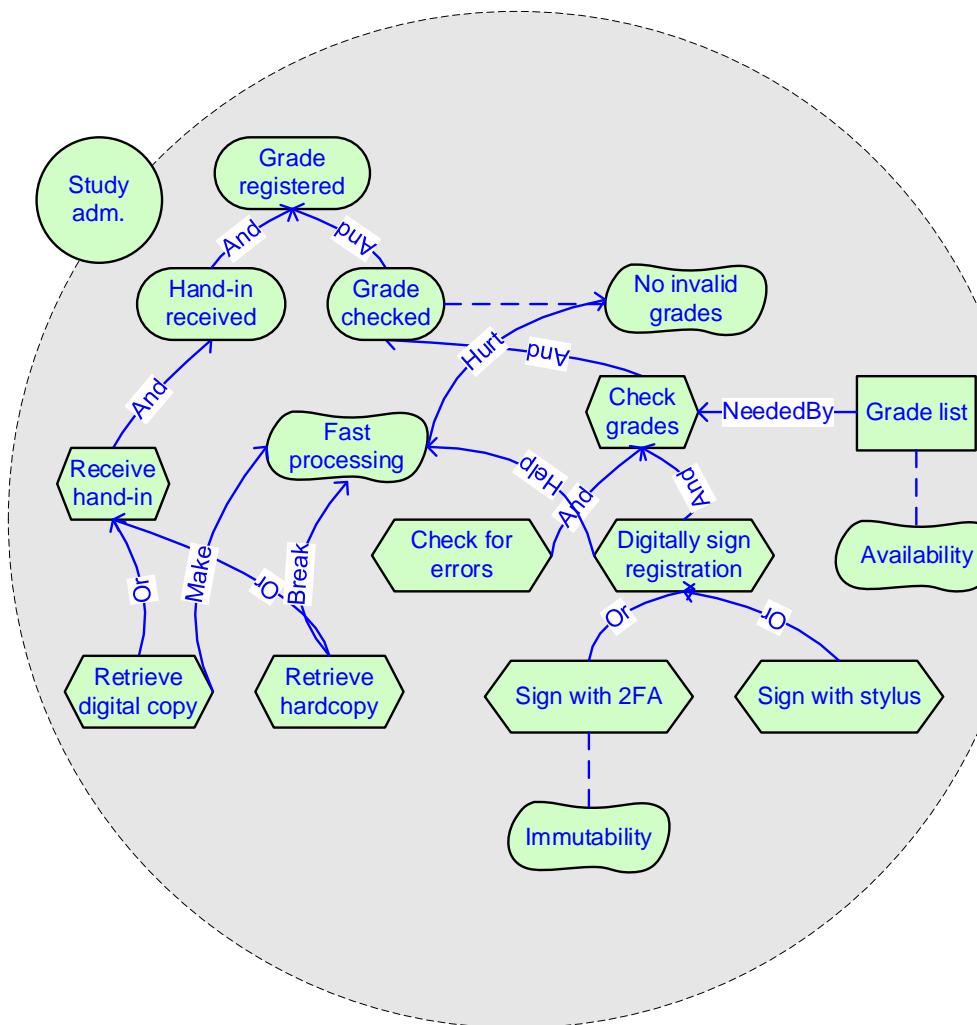
I-star



ArchiMate



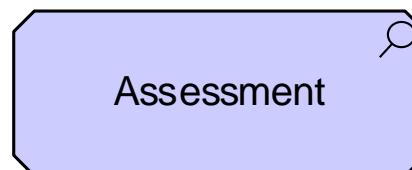
Example



ArchiMate-exclusive intentional elements



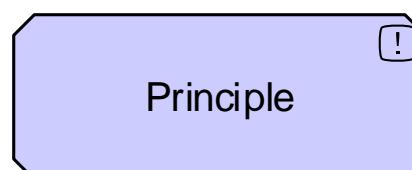
An external or internal condition that motivates an organisation to define its goals and implement the changes necessary to achieve them.



The result of an analysis of the state of affairs of the enterprise with respect to some driver.

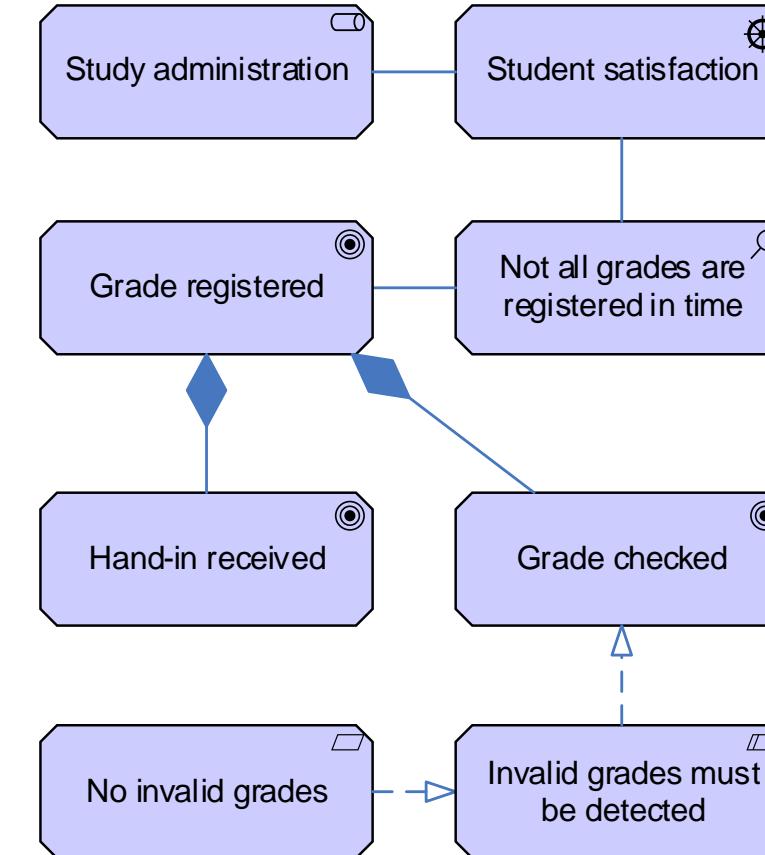
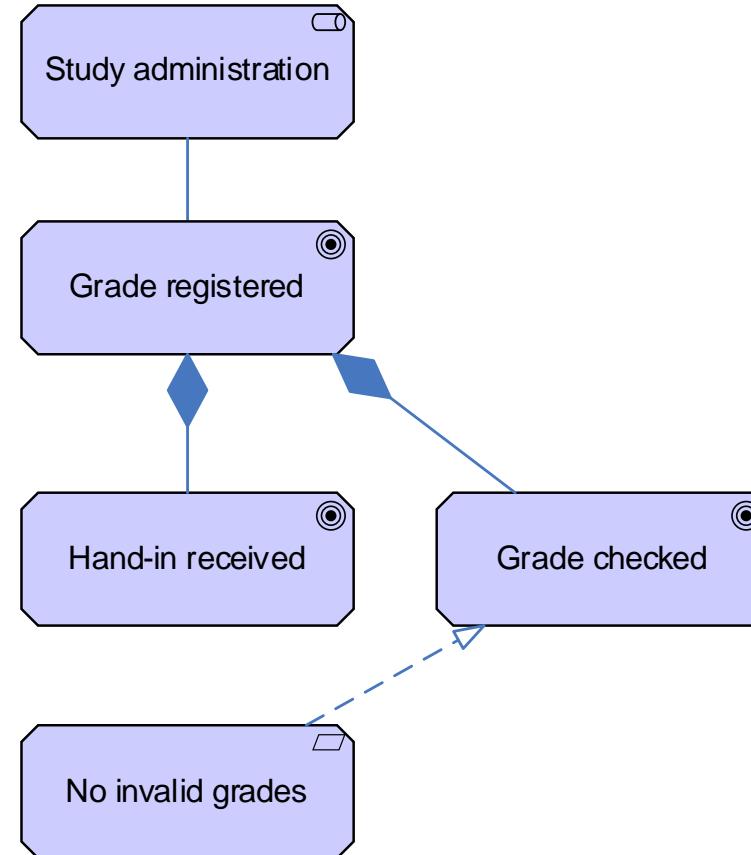


A factor that prevents or obstructs the realisation of goals.

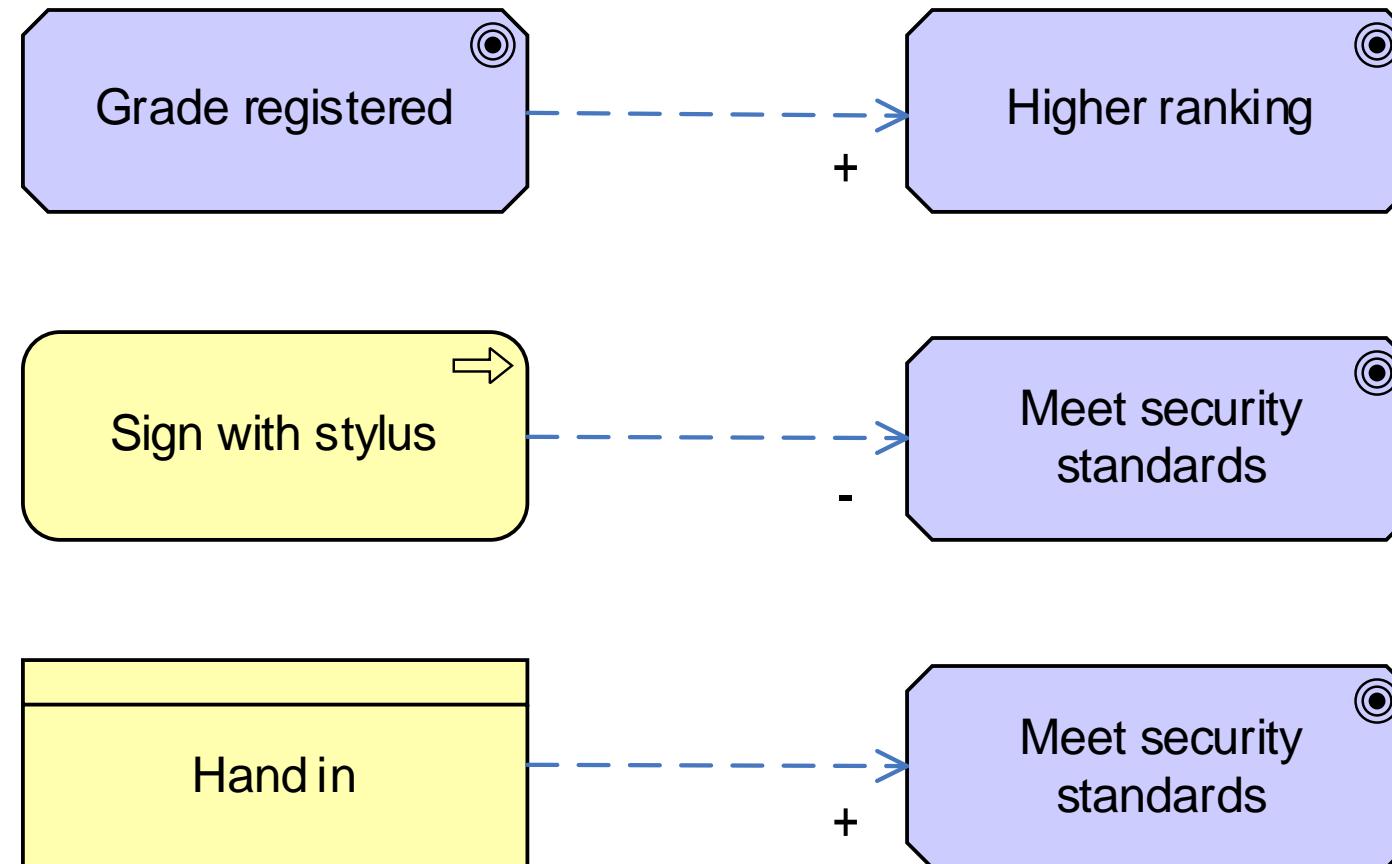


A qualitative statement of intent that should be met by the architecture

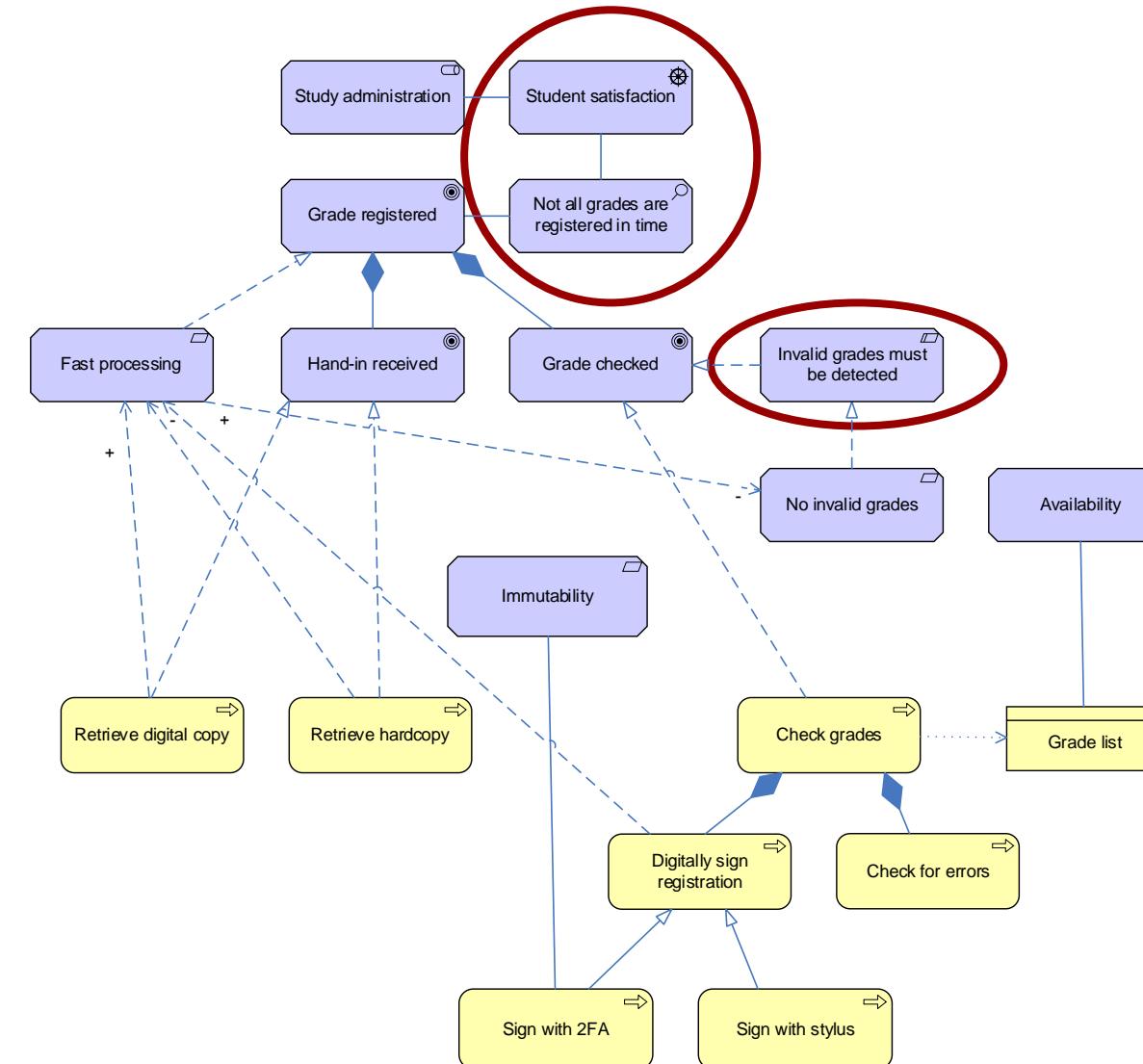
ArchiMate-exclusive intentional elements



ArchiMate-exclusive contribution links

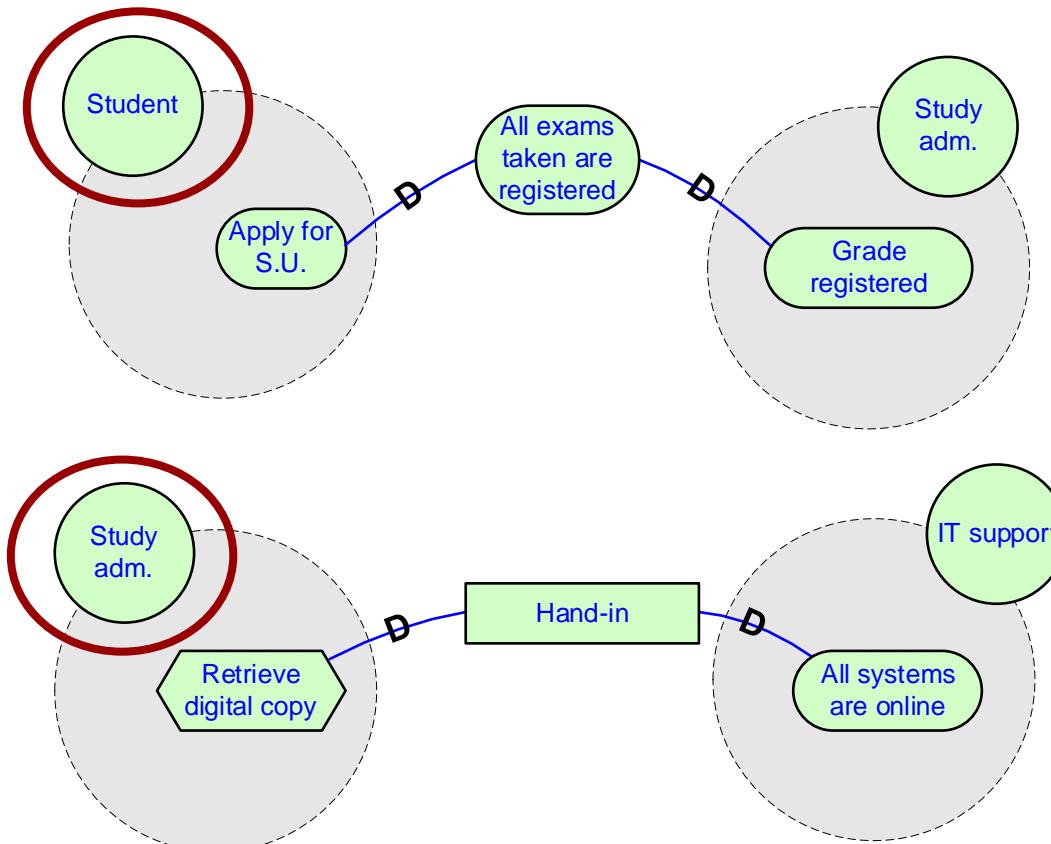


Extended example in ArchiMate

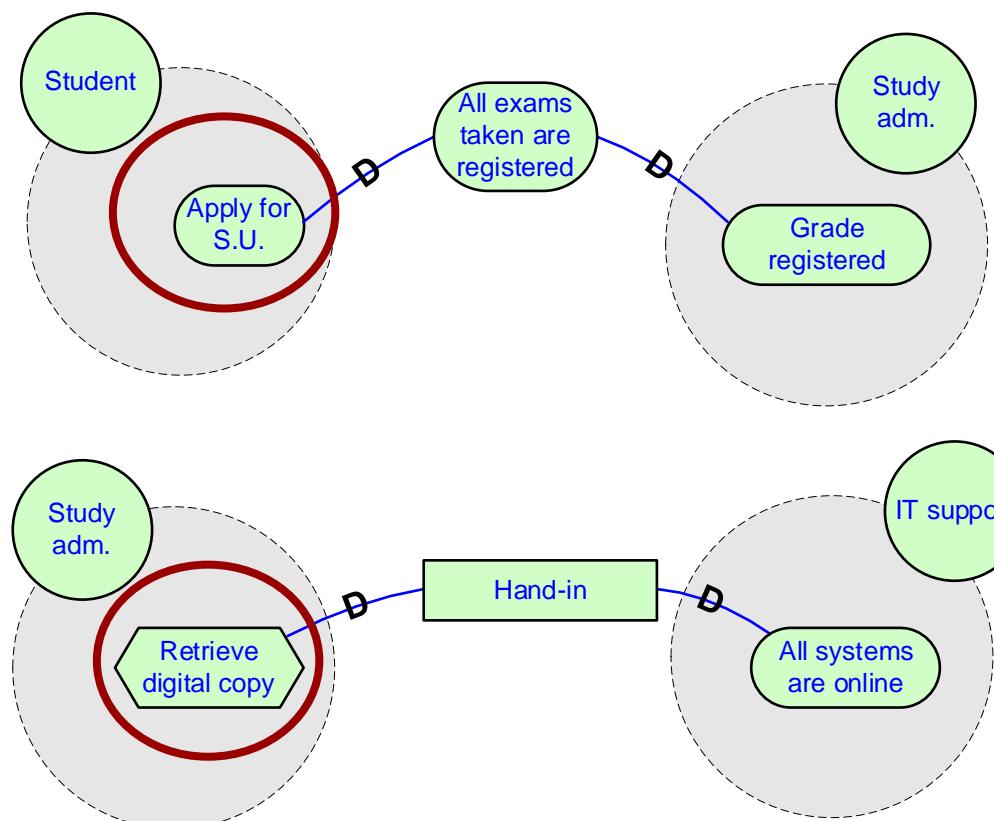


I-star-exclusive social dependencies

- Represent social relationships
 - **Depender**: an actor that depends for something (the dependum) to be provided

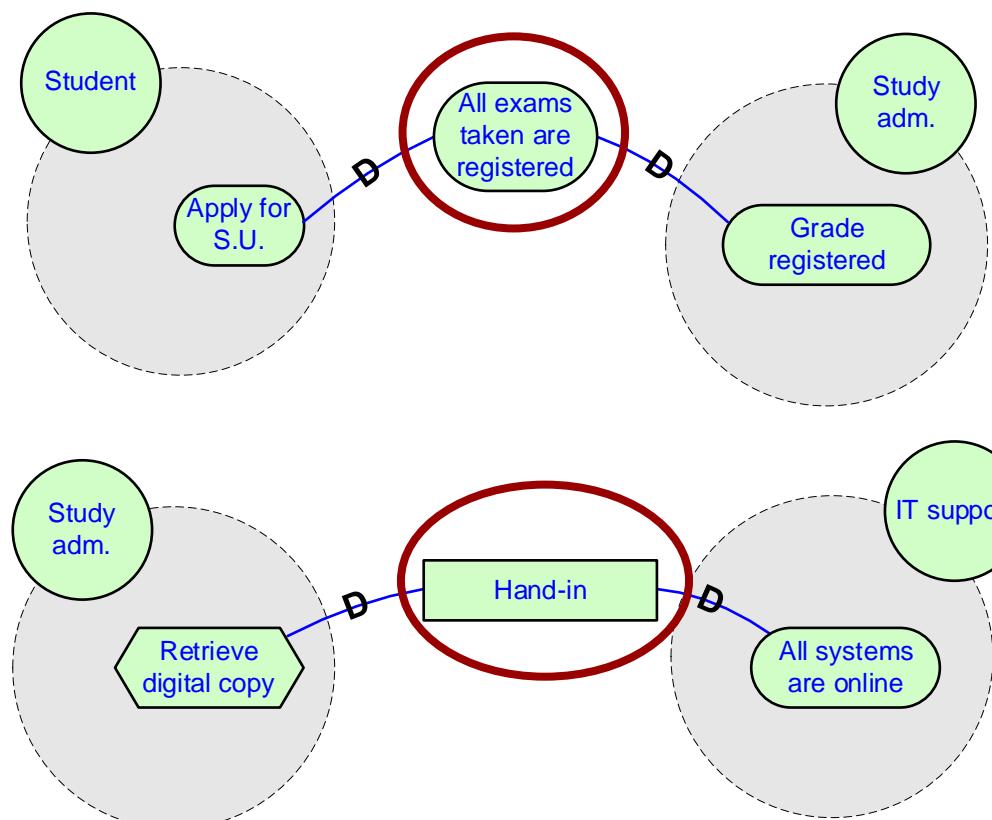


I-star-exclusive social dependencies



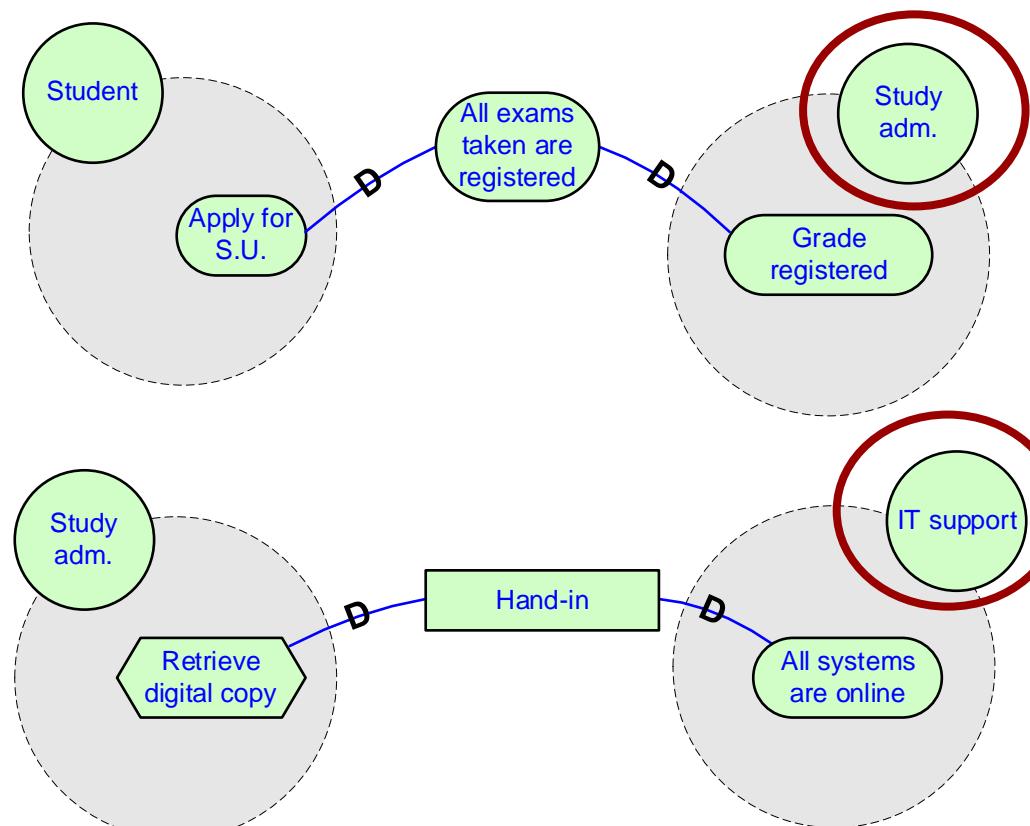
- Represent social relationships
 - **Depender:** an actor that depends for something (the dependum) to be provided
 - **DependerElement:** an intentional element within the depender's boundary where the dependency starts from, which explains why the dependency exists

I-star-exclusive social dependencies



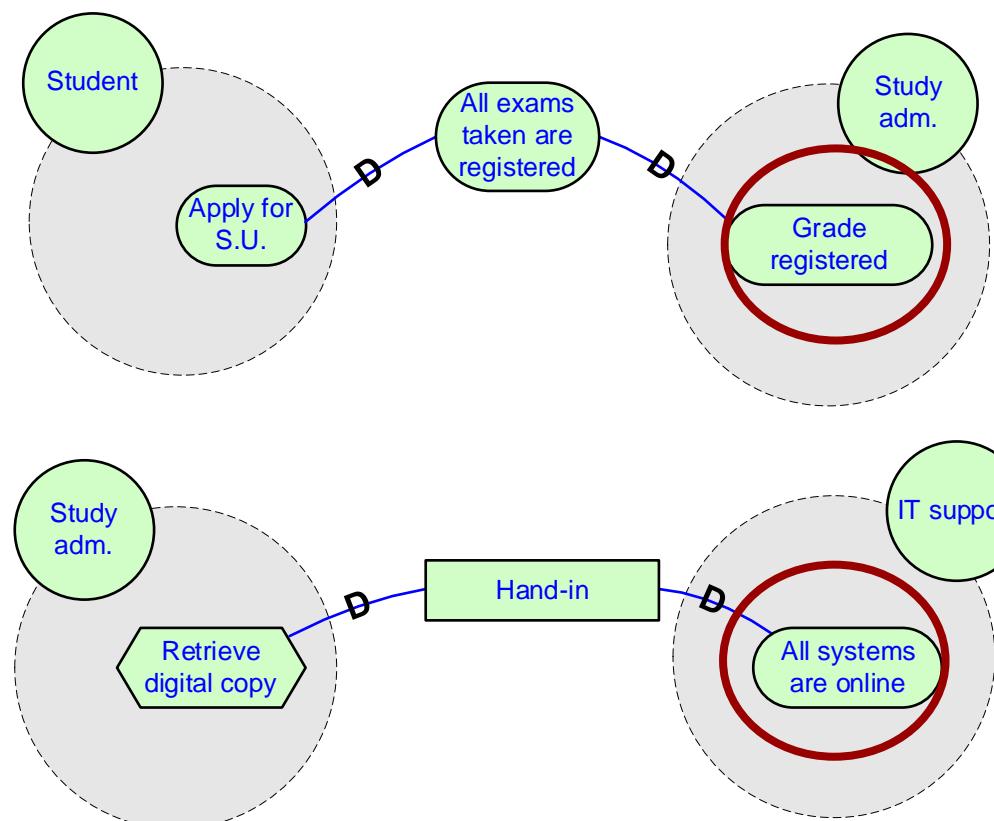
- Represent social relationships
 - **Depender:** an actor that depends for something (the dependum) to be provided
 - **DependerElement:** an intentional element within the depender's boundary where the dependency starts from, which explains why the dependency exists
 - **Dependum:** an intentional element that is the object of the dependency

I-star-exclusive social dependencies



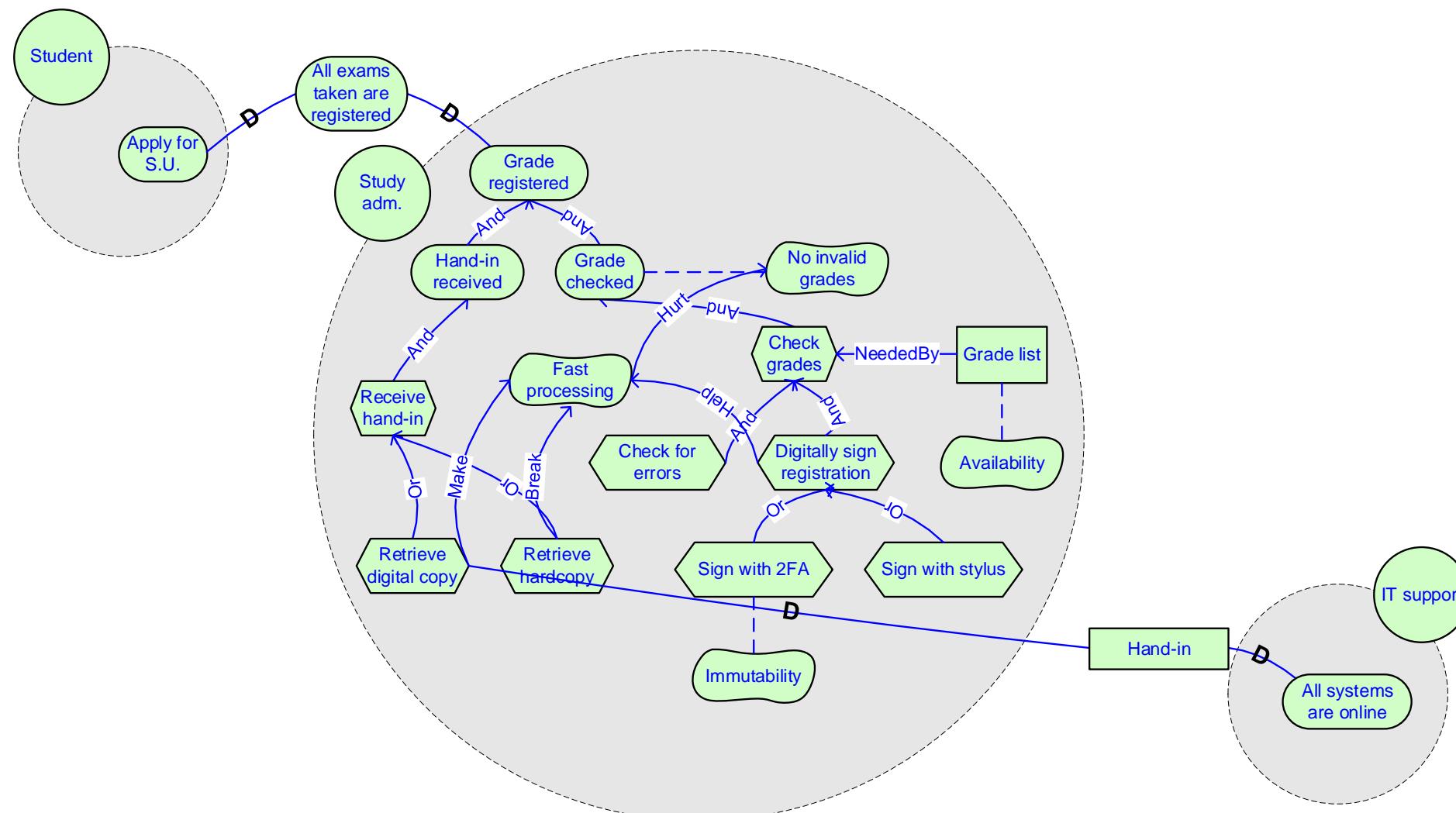
- Represent social relationships
 - **Depender:** an actor that depends for something (the dependum) to be provided
 - **DependerElement:** an intentional element within the depender's boundary where the dependency starts from, which explains why the dependency exists
 - **Dependum:** an intentional element that is the object of the dependency
 - **Dependee:** the actor that should provide the dependum

I-star-exclusive social dependencies

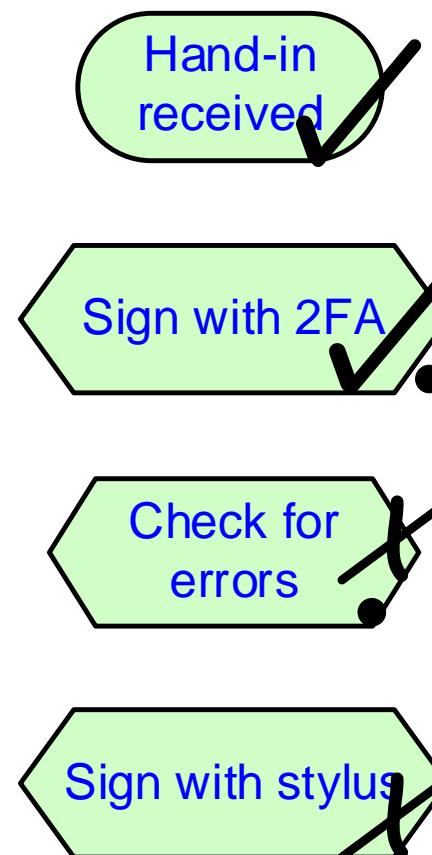


- Represent social relationships
 - Depender: an actor that depends for something (the dependum) to be provided
 - DependerElement: an intentional element within the depender's boundary where the dependency starts from, which explains why the dependency exists
 - Dependum: an intentional element that is the object of the dependency
 - Dependee: the actor that should provide the dependum
 - **DependeeElement:** the intentional element that explains how the dependee intends to provide the dependum

Extended example in I-star

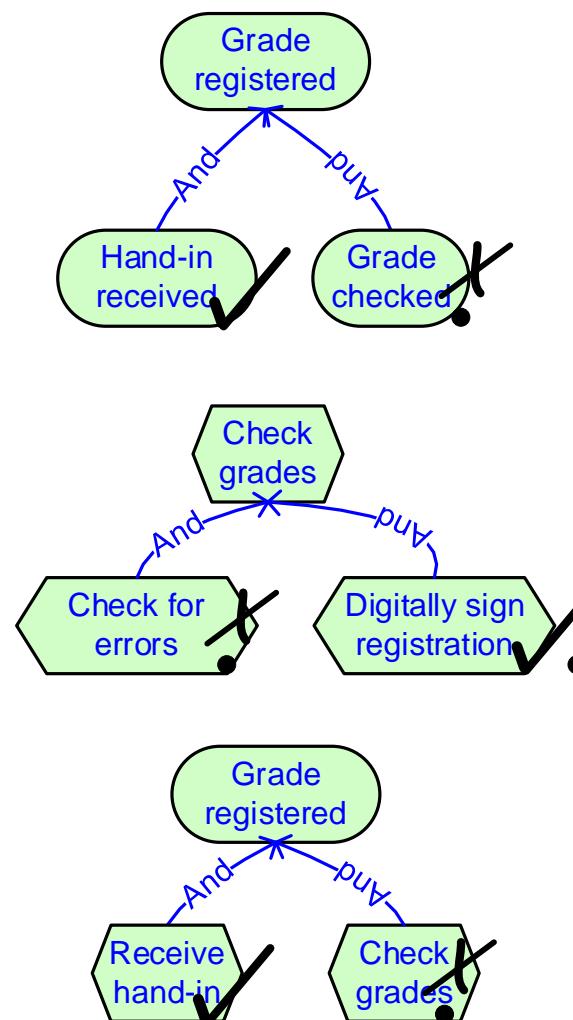


Reasoning



- Goals and Tasks can be characterized with label indicating to which extent they are achieved:
 - Denied
 - Partially denied
 - Partially satisfied
 - Satisfied
- By doing so, it is possible to identify their effect on the model

Forward analysis

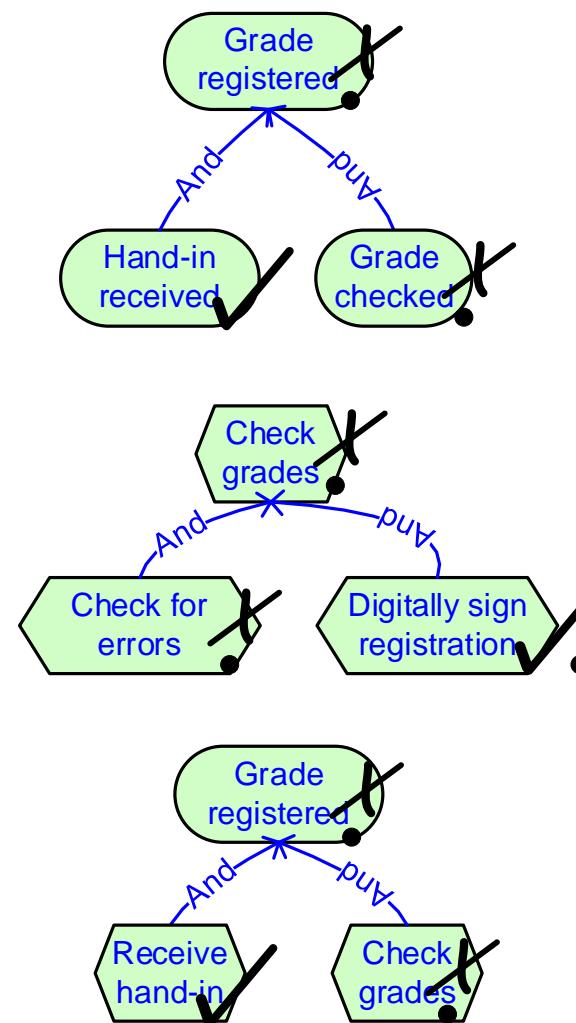


- For goal/task decomposition and inclusive goal realization, the minimum value of all the child nodes is taken.

Forward analysis

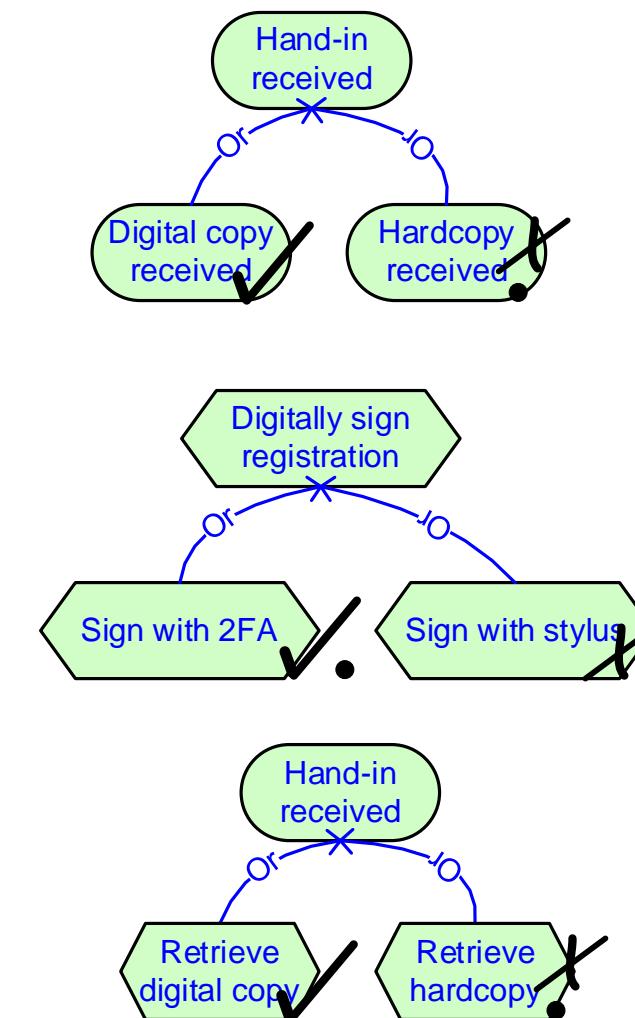
AND-Decomposition

Inputs		Result
Child 1	Child 2	Parent
✓	✓	✓
✓	✓	✓
✓	✗	✗
✓	✗	✗
✓	✓	✓
✓	✓	✓
✓	✗	✗
✓	✗	✗
✗	✓	✗
✗	✓	✗
✗	✗	✗
✗	✗	✗
✗	✓	✗
✗	✓	✗
✗	✗	✗



- For goal/task decomposition and inclusive goal realization, the minimum value of all the child nodes is taken.

Forward analysis

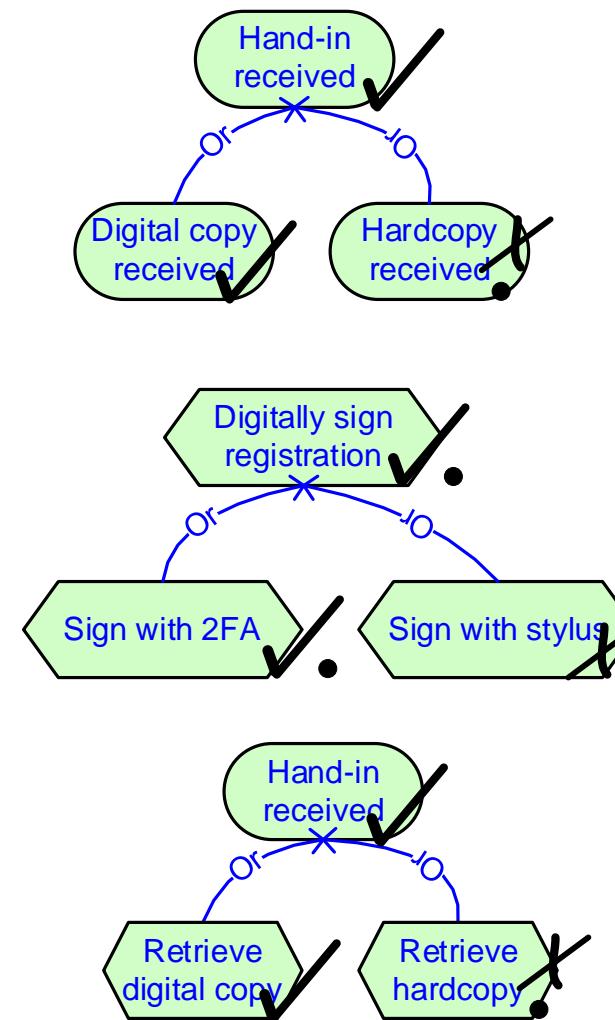


- For goal/task specialization and exclusive goal realization, the maximum value of all the child nodes is taken.

Forward analysis

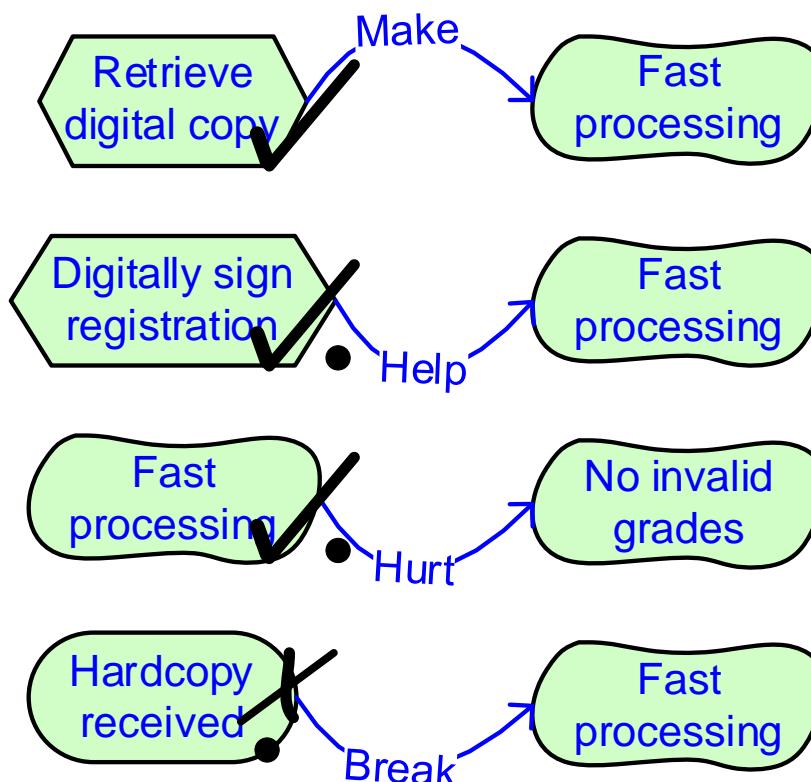
OR-Decomposition

Inputs		Result
Child 1	Child 2	Parent
✓	✓	✓
✓	✓	✓
✓	✗	✓
✓	✗	✓
✓	✓	✓
✓	✓	✓
✓	✗	✓
✓	✗	✓
✗	✓	✓
✗	✓	✓
✗	✗	✗
✗	✗	✗
✗	✓	✓
✗	✓	✓
✗	✗	✗
✗	✗	✗



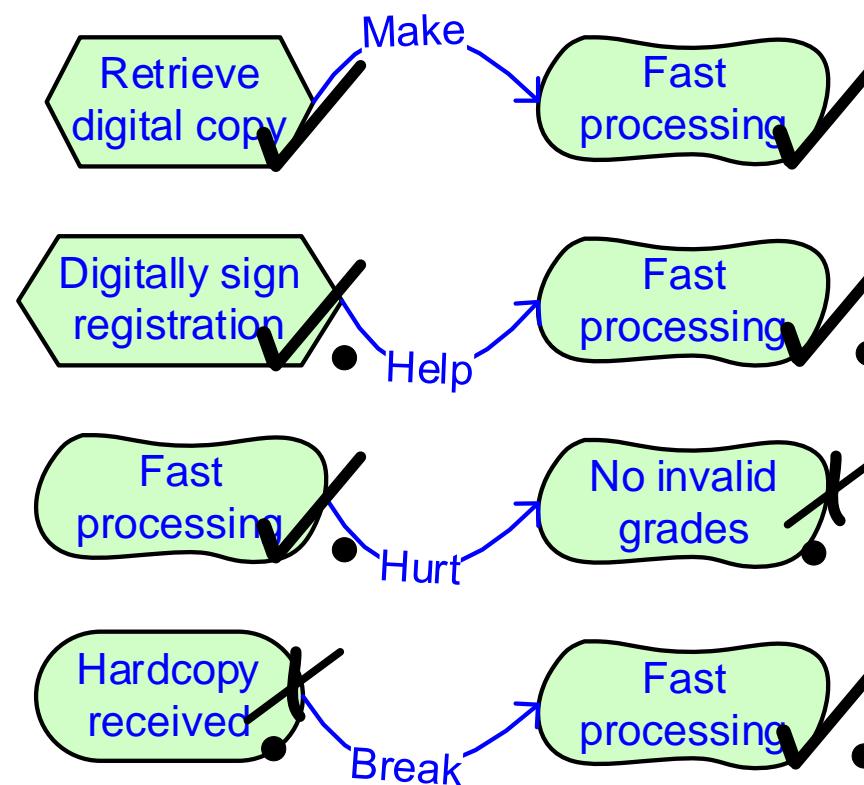
- For goal/task specialization and exclusive goal realization, the maximum value of all the child nodes is taken.

Forward analysis



- For contribution links, the value of the goal is propagated to the quality depending on the link type:
 - Make links propagate the value as-is
 - Help links propagate the value as *partial*
 - Hurt links invert and propagate the value as *partial*
 - Break links invert the value

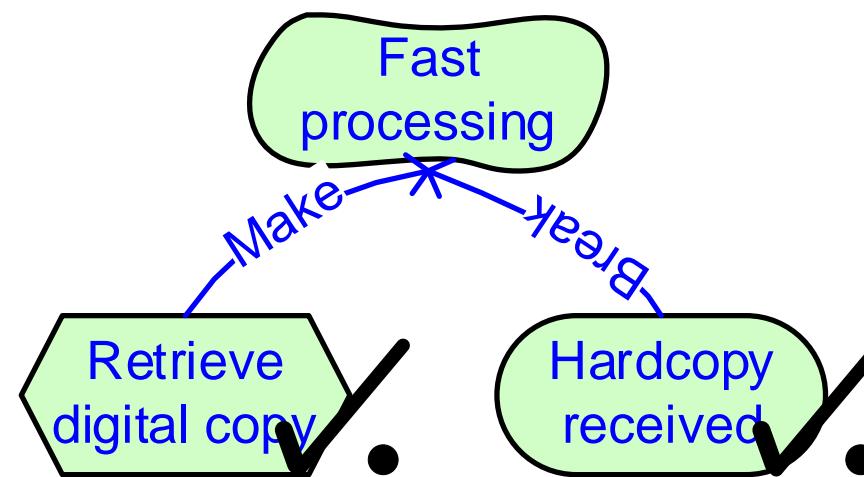
Forward analysis



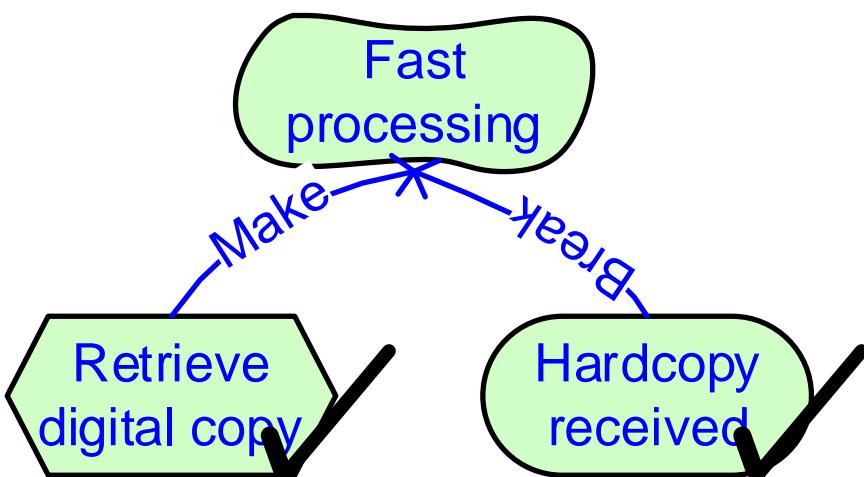
- For contribution links, the value of the goal is propagated to the quality depending on the link type:
 - Make links propagate the value as-is
 - Help links propagate the value as *partial*
 - Hurt links invert and propagate the value as *partial*
 - Break links invert the value

Source	Destination for each Link Type				
	Depends	Makes	Helps	Hurts	Breaks
✓	✓	✓	✓	✗	✗
✓	✓	✓	✓	✗	✗
✗	✗	✗	✗	✗	✓
✗	✗	✗	✗	✓	✓

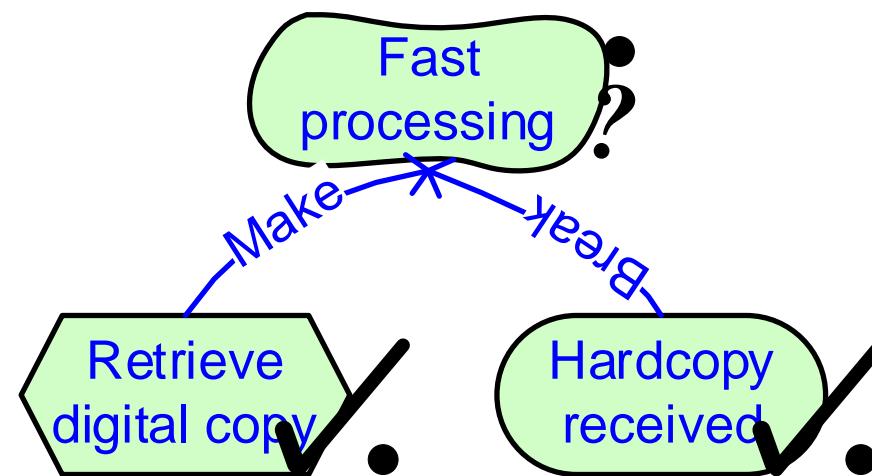
Forward analysis



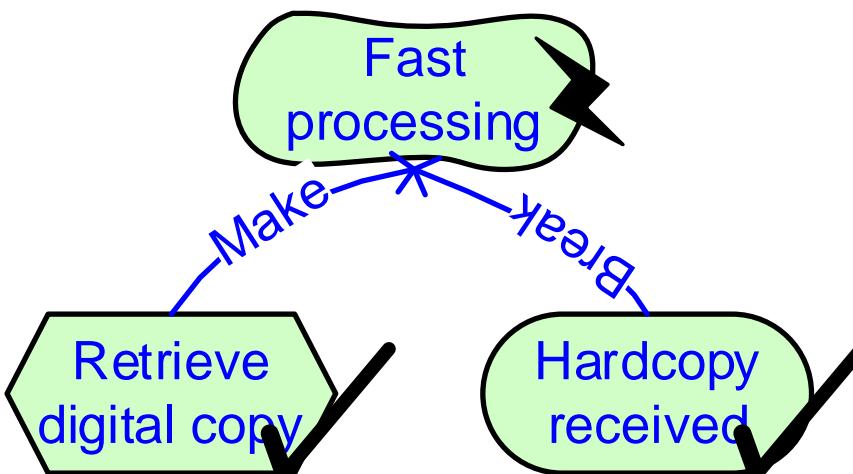
- When different contribution links affect the same quality, we may also get:
 - Unknown: it is unclear if the quality will be satisfied or not
 - Conflict: contributions contradict each other



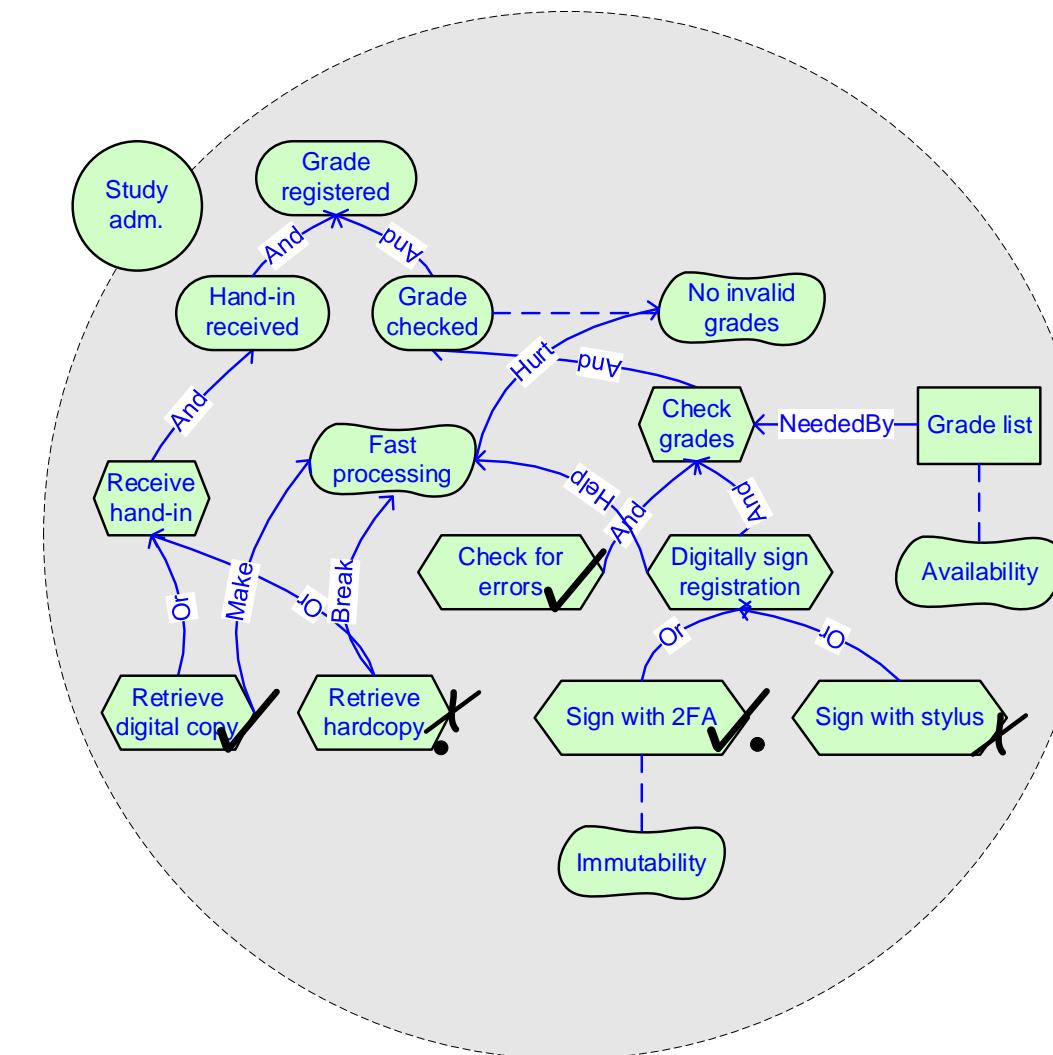
Forward analysis



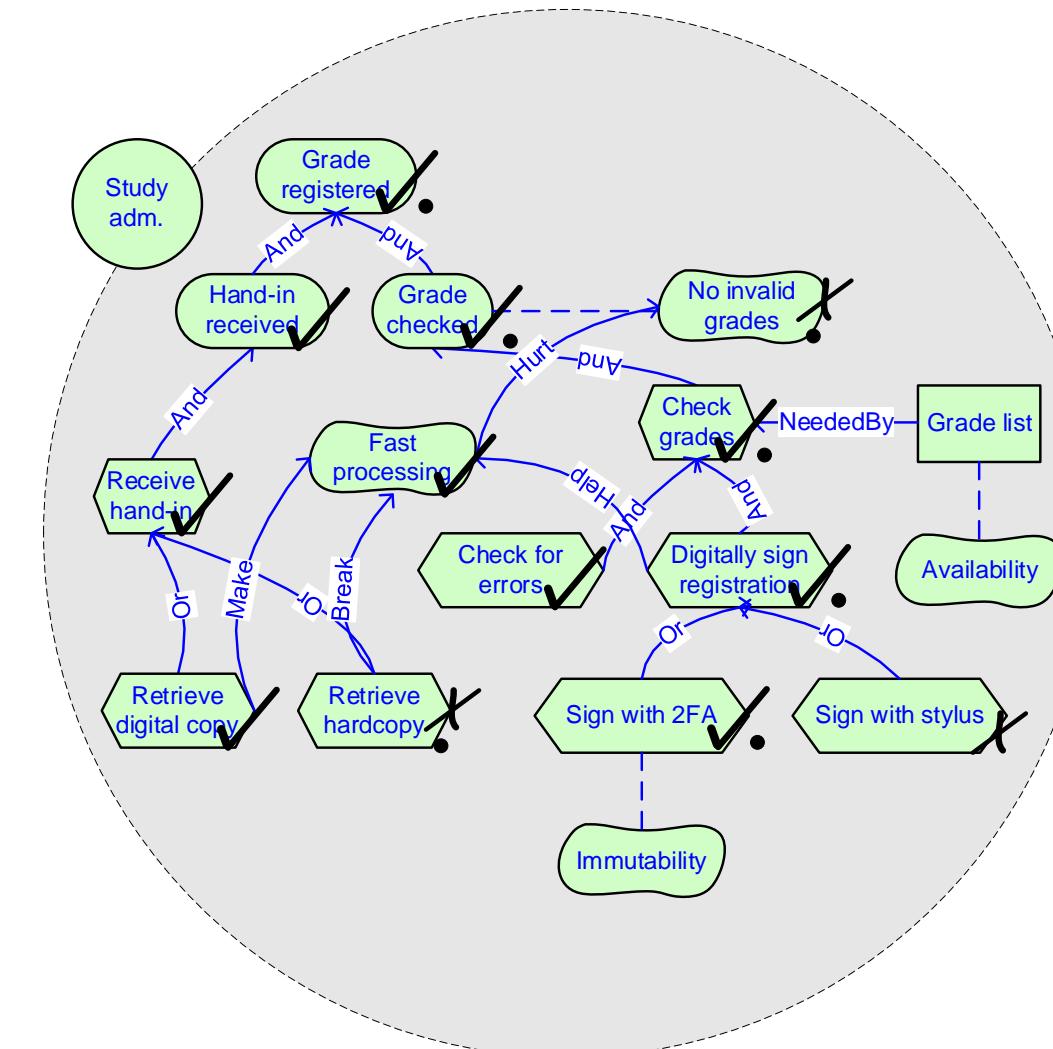
- When different contribution links affect the same quality, we may also get:
 - Unknown: it is unclear if the quality will be satisfied or not
 - Conflict: contributions contradict each other



Forward analysis – Example



Forward analysis – Results



Study material

- Books and articles:
 - Dalpiaz et al. - iStar 2.0 Language Guide
 - Available at: <https://sites.google.com/site/istarlanguage/home>
 - Lankhorst et al. - Enterprise Architectures at Work (4th Edition)
 - Available at: <https://link.springer.com/book/10.1007/978-3-662-53933-0>
 - Chapter 5.6
- Modeling tools:
 - piStar: <http://www.cin.ufpe.br/~jhcp/pistar/>
 - (alternatively) Leaf: <https://www.cs.toronto.edu/~amgrubb/leaf-2.0/Tool.html>
 - Archi: <https://www.archimatetool.com/>
 - (alternatively) SAP Signavio: <https://academic.signavio.com/p/login>

Exercises

Please answer all exercises to demonstrate your skills.

Solutions will be available at 11:45

Exercise 1 – Speedy

Speedy is an international delivery company that needs to refocus on which markets it should operate. Indeed, Speedy lacks a clear understanding on which markets are the most profitable. Thus, to address this issue, Speedy's top management aims at improving their governance. To achieve this, the top management requires sales reports containing timely information. Thus, the top management decided to build a new reporting system to automatically produce such reports. After inspecting the sales reports, the top management may also need to query the existing ERP system to get detailed sales and HR information.

Prepare an ArchiMate and an I-star model that captures the formalizes Speedy's new architecture.

- Can you model all the elements and relations described in this exercise with both languages? If not, which elements can be captured only in ArchiMate? Which ones in I-star?

Exercise 2 – IC

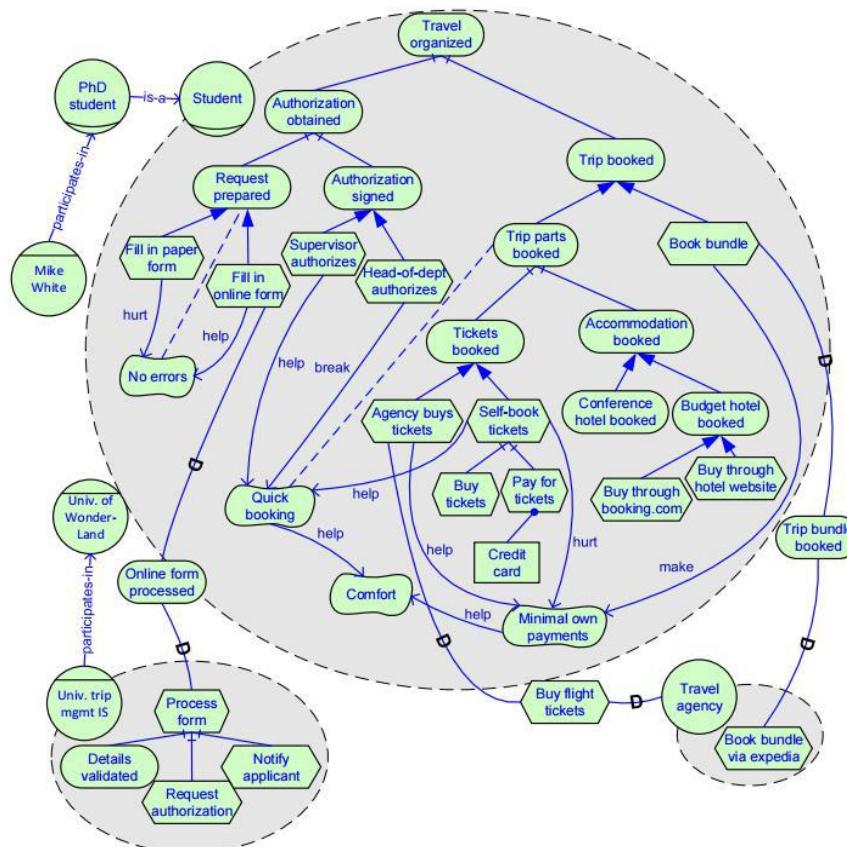
IC is an insurance company which wants to offer a new insurance service for small assets (<2000\$) managed completely online for reliable customers.

To achieve this, a customer who wants his assets to be insured has to provide its credentials and photo of the asset and its details (serial number, purchase date) to IC. To ensure that the customer is reliable and the asset inexpensive, IC will then check the customers credentials and past history and estimate the asset's price.

Prepare an ArchiMate and an I-star model that formalizes IC's requirements.

- Can you model all the elements and relations described in this exercise with both languages? If not, which elements can be captured only in ArchiMate? Which ones in I-star?

Exercise 3 – University travel reimbursement



- This I-star model represents a university travel reimbursement system
- Prepare an ArchiMate model representing the same system
 - Can you model all the elements and relations in this I-star model?
 - If not, which elements and relations cannot be modeled?

Model from: F. Dalpiaz, X. Franch, and J. Horko – iStar 2.0 Language Guide
<https://arxiv.org/abs/1605.07767>

System Integration - Refreshment to Formal Languages

Hugo A. López

hulo@dtu.dk



Hugo Andrés López



Title: Associate Professor
Software Systems Engineering
DTU Compute

EDUCATION



Ph.D. Computer Science - ITU
Ms.C. Computer Science - ITU
Ing. Sistemas & Computación. U. Javeriana - Cali

CURRENT PROJECTS



Global Digital Human Rights Network - GHRNet
Sustainable Consumption Habits - A cost benefit perspective
Danish Center for Digital Compliance
AI Denmark - Financial Compliance



RESEARCH

Programming Languages
Analysis and Verification of Software
Software Engineering (Processes, Empirical Methods)
Artificial Intelligence (Rule-based systems, NLP)



CV

22 - Now: Associate Professor - DTU
20 - 22: Assistant Professor - U. Copenhagen
17 - 21: Product Owner - DCR
17 - 20: Industrial Postdoc - ITU
15 - 17: Postdoc - DTU
14 - 15: Postdoc - U. Lisbon
12 - 14: Software Consultant - Configit

Why do we want to have models?

Advantages of Models

- Abstraction: decrease the accidental complexity of a system
 - The *essential complexity* is the same, but the models help isolating concerns
 - Are we shipping the right components in a deployment?
 - Have we captured all the stakeholders in a requirements elicitation phase?
 - Does my communication protocol allow deadlocks?
- Enables *reasoning about software behavior*
 - Testing does not guarantee the absence of bugs
 - Possibility of what-if scenario modelling
- Enable low-code executions

Abstraction

- Models become a way to talk about your software
- The pragmatics of the model might differ a lot



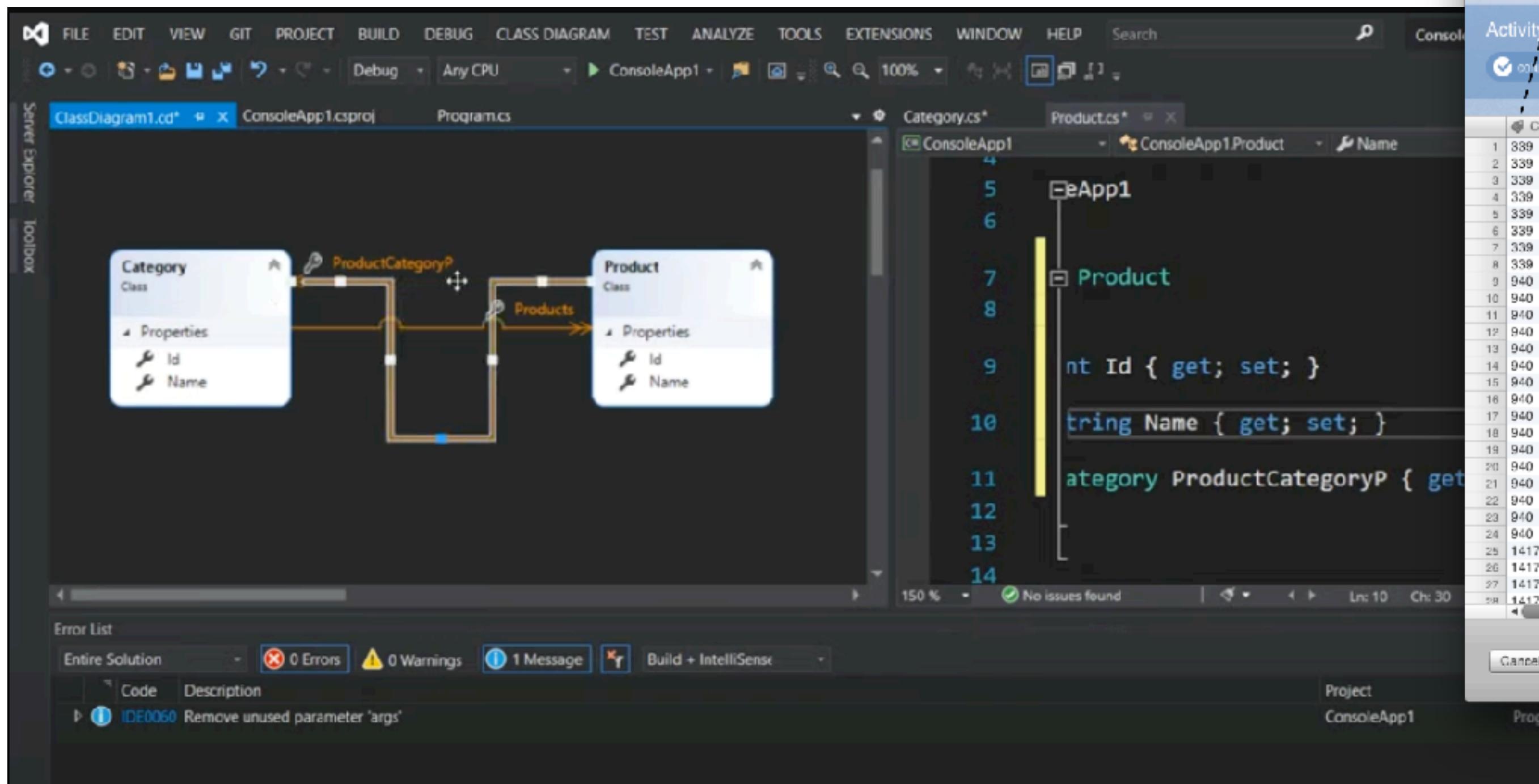
Sketch an existing process from users' perspective (process mapping)

Abstraction

- Models become a way to talk about your software
- The pragmatics of the model might differ a lot

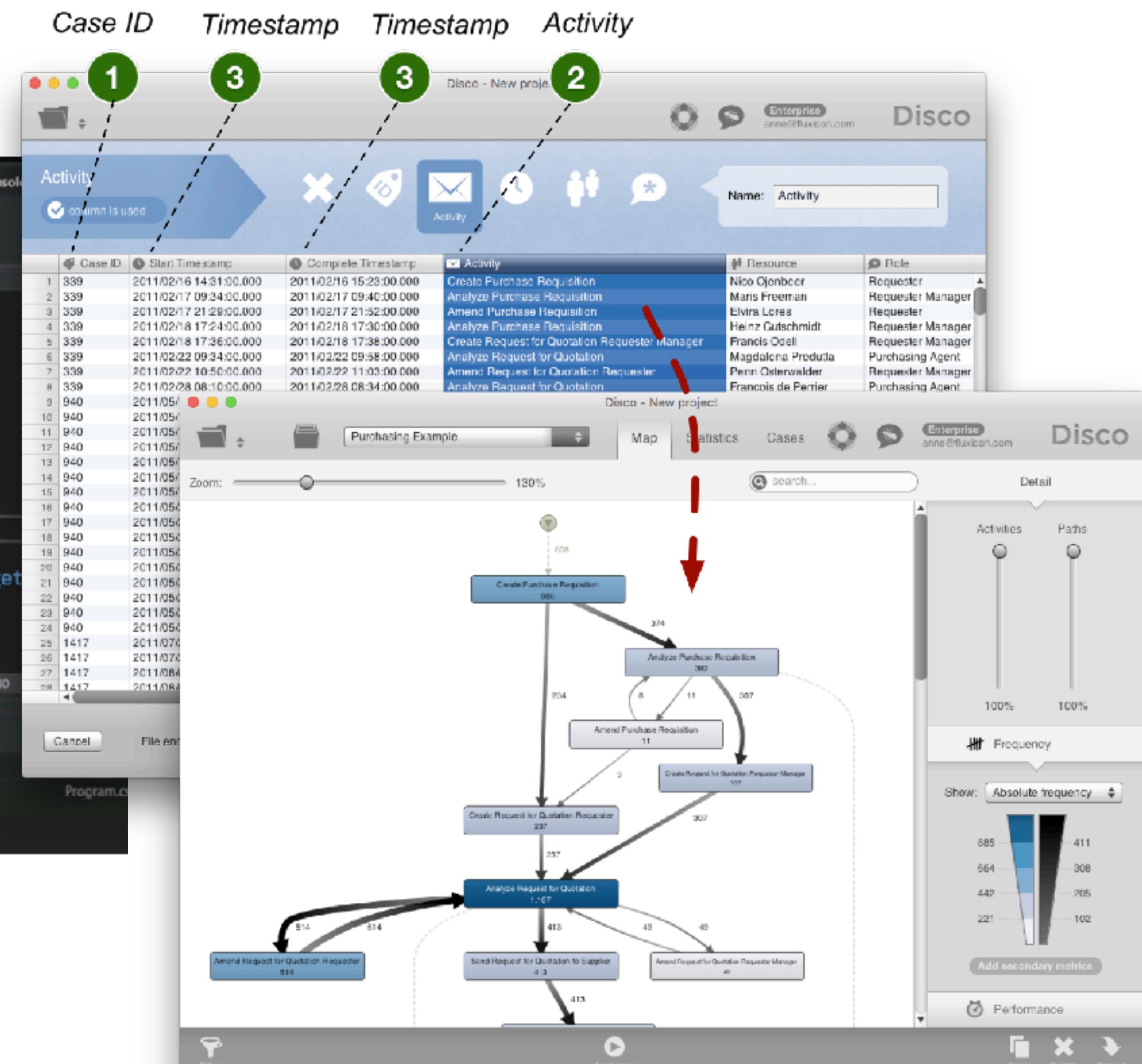
Abstraction

- Models become a way to talk about your software
- The pragmatics of the model might differ a lot



A screenshot of Microsoft Visual Studio. On the left, a Class Diagram shows a Category class with an association to a Product class named ProductCategoryP. On the right, the Product.cs code is shown:

```
namespace ConsoleApp1
{
    public class Product
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public Category ProductCategoryP { get; set; }
    }
}
```



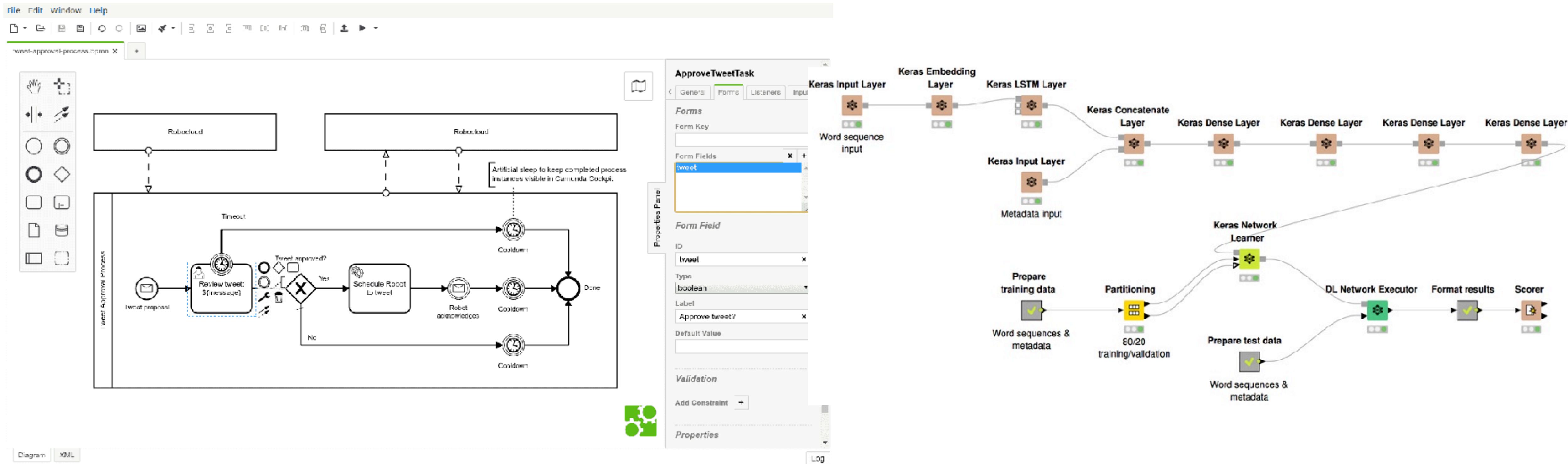
Representing behaviour from an existing system

Abstraction

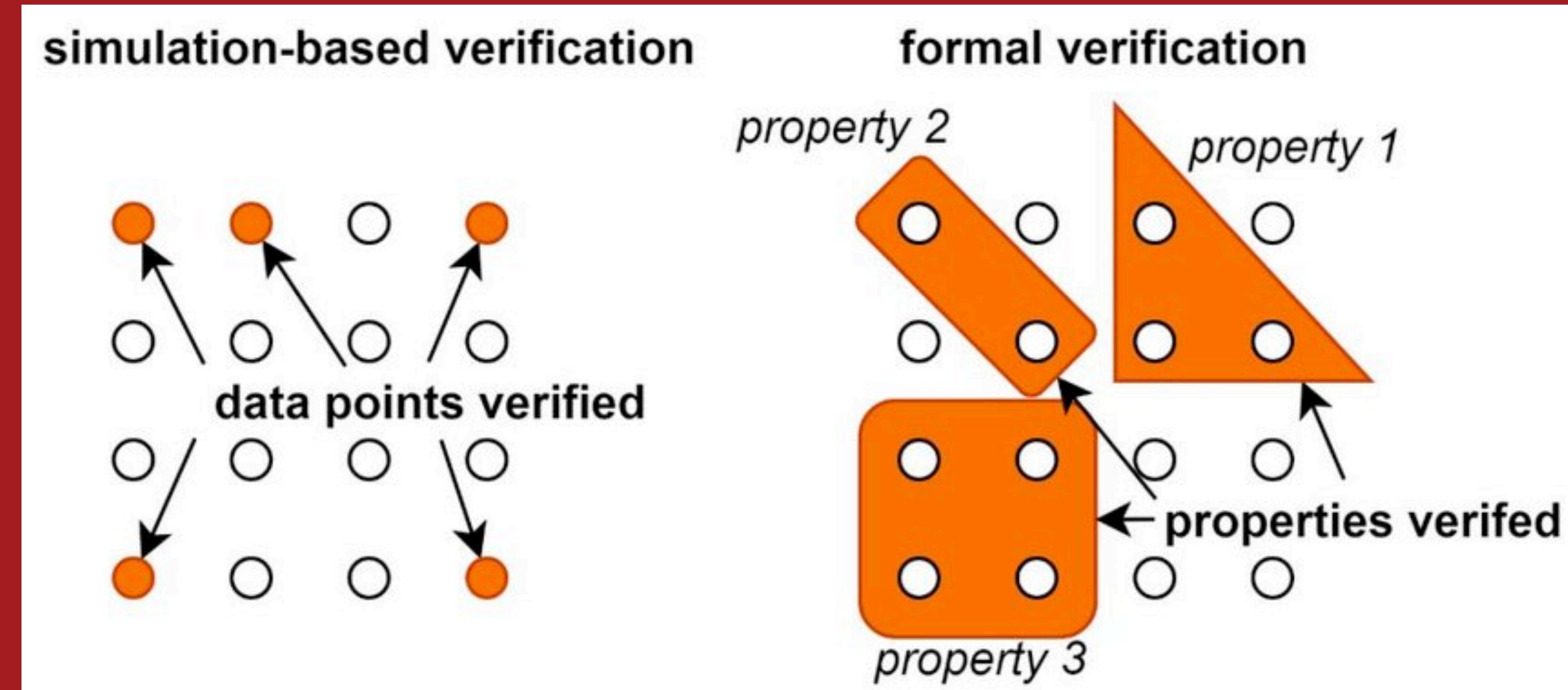
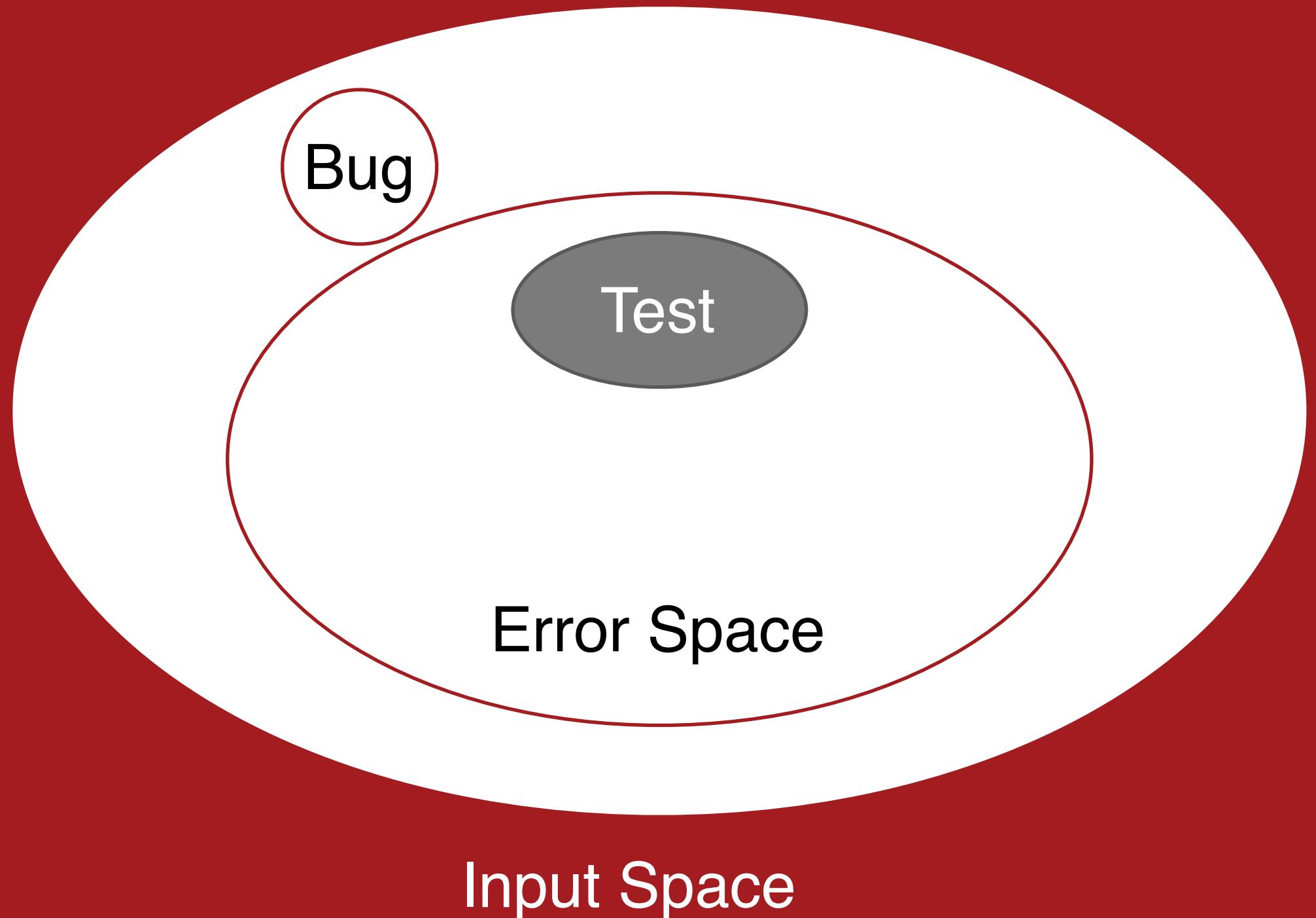
- Models become a way to talk about your software
- The pragmatics of the model might differ a lot

Abstraction

- Models become a way to talk about your software
- The pragmatics of the model might differ a lot



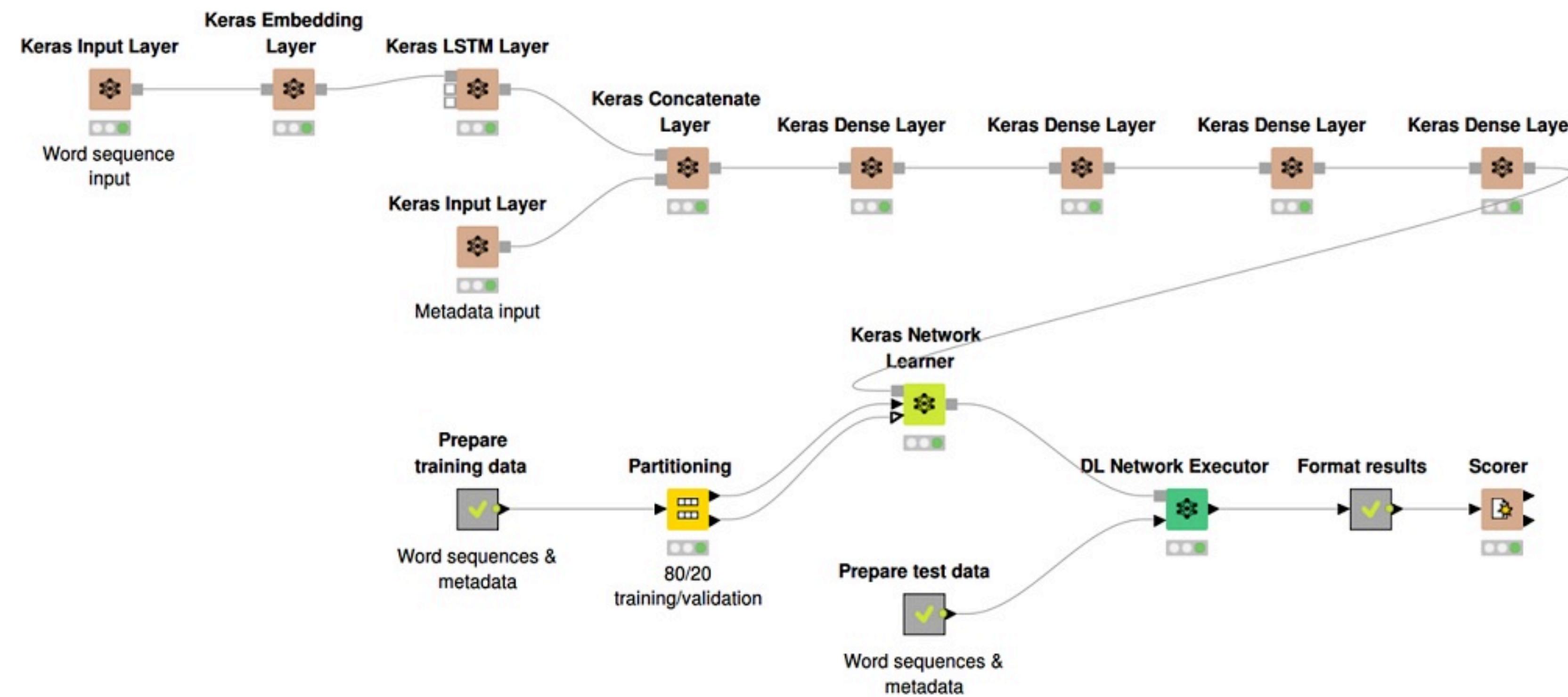
Automatic Generation of implementations (low-code)



Testing and Verification

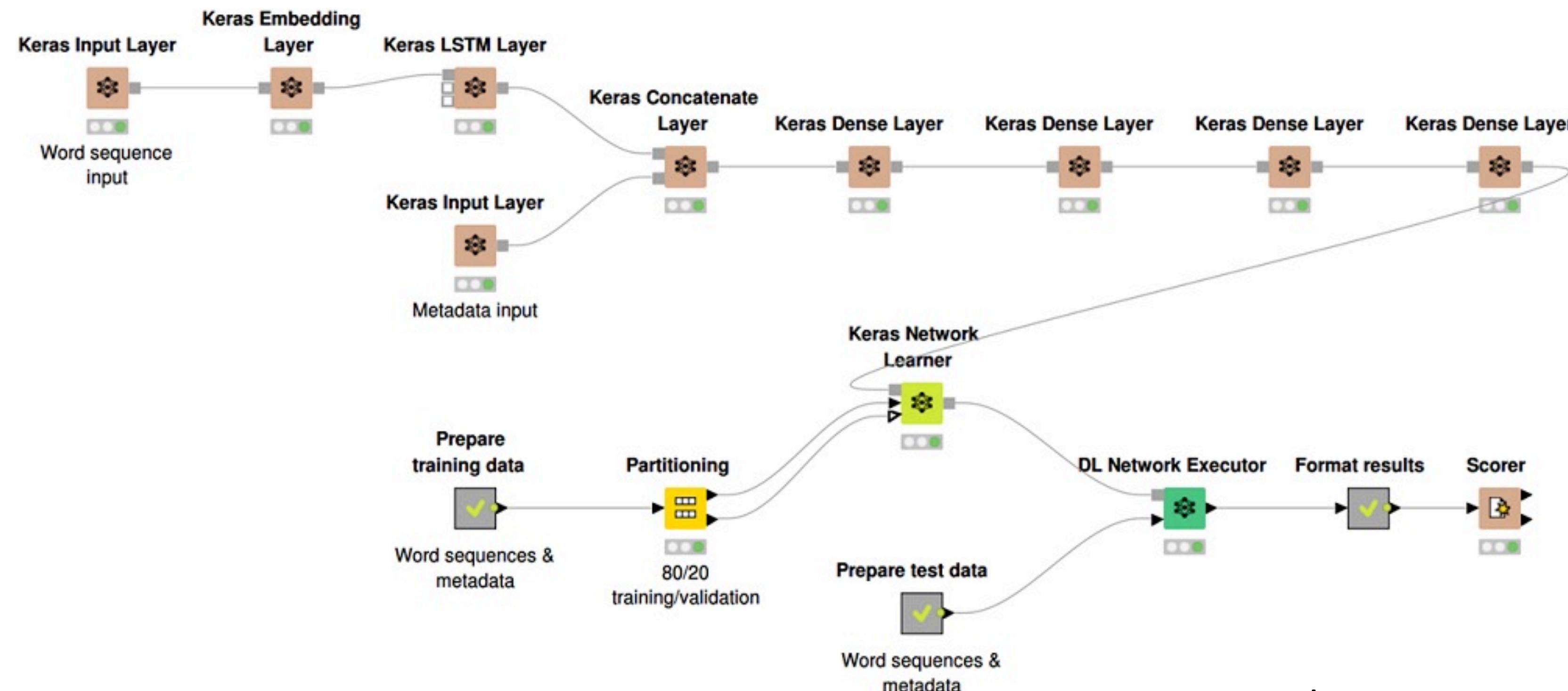
A testing approach does not guarantee a fail-free implementation. It guarantees that the software passes the tests you have provided.
If you need full-assurance, you need higher guarantees, for instance, formal methods

Execution Pipelines



- Programming and application directly on top of an infrastructure (OS, programming language, DB, etc) makes the application fragile and dependent the specific infrastructure
- By raising the abstraction level at what you define your needs, you can first focus on modeling the behaviour and later on refining it to integrate platform-specific details

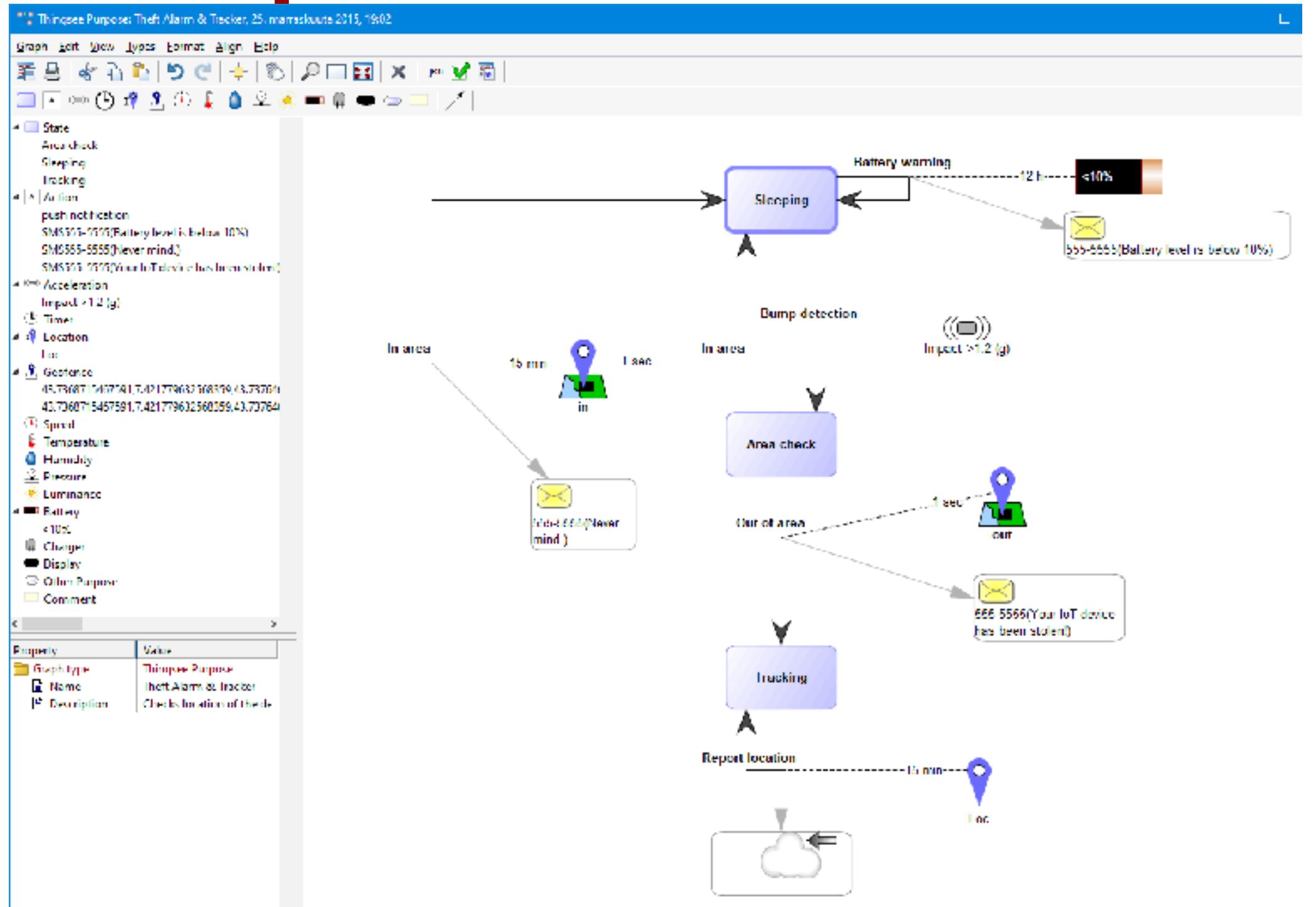
Execution Pipelines



Read more on [MBSE for AI](#)

- Programming and application directly on top of an infrastructure (OS, programming language, DB, etc) makes the application fragile and dependent the specific infrastructure
- By raising the abstraction level at what you define your needs, you can first focus on modeling the behaviour and later on refining it to integrate platform-specific details

Domain-specific Models



Modelling Language for IoT applications

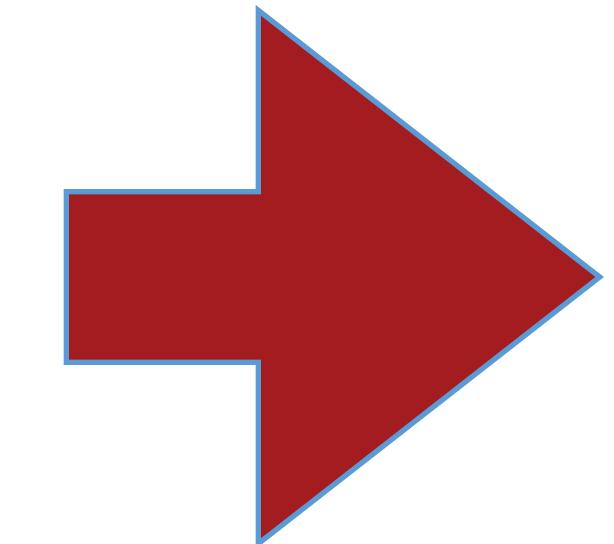


Patient-care process model
(MSc thesis@DTU)

- Reduce accidental complexity by talking in stakeholder's terms instead of computer science terms
- Possible to reuse a common modelling framework (e.g. [Eclipse](#)) and specialize for each domain

Formal Semantics for Models

- Goal:

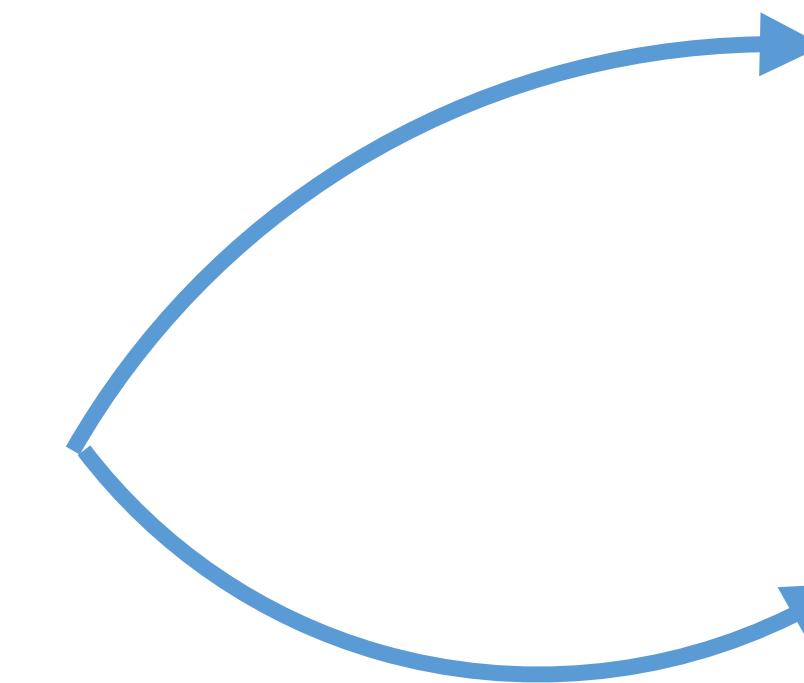


From Informal Specifications

To models as “digital twins”

Do I make myself clear?

- Consider the following requirement
 - "*Attach wheels and engine to frame*"

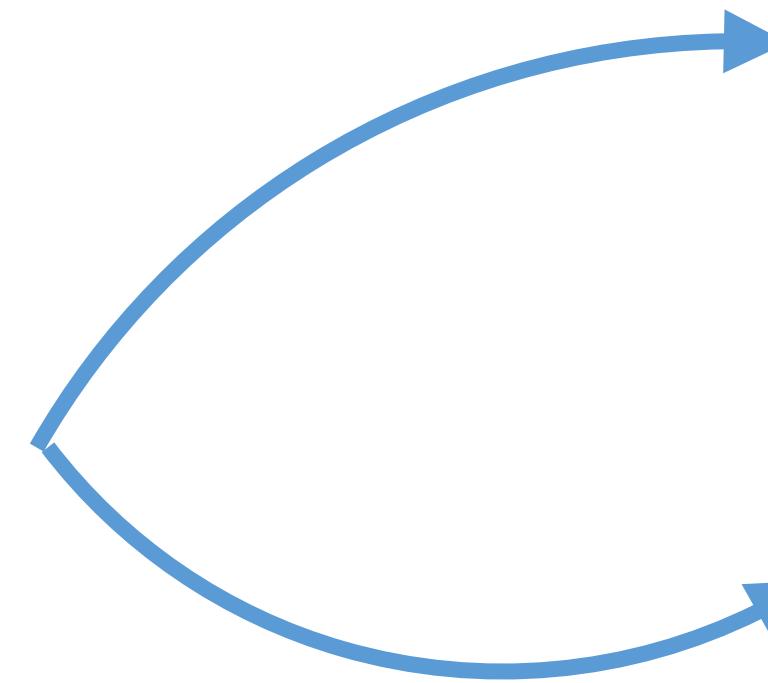


First, I attach the wheels, and **then** I attach the engine to the frame

I need to attach the wheels and the engine **at the same time** to the frame

Do I make myself clear?

- Consider the following requirement
 - "*Attach wheels and engine to frame*"



First, I attach the wheels, and then I attach the engine to the frame

I need to attach the wheels and the engine **at the same time** to the frame

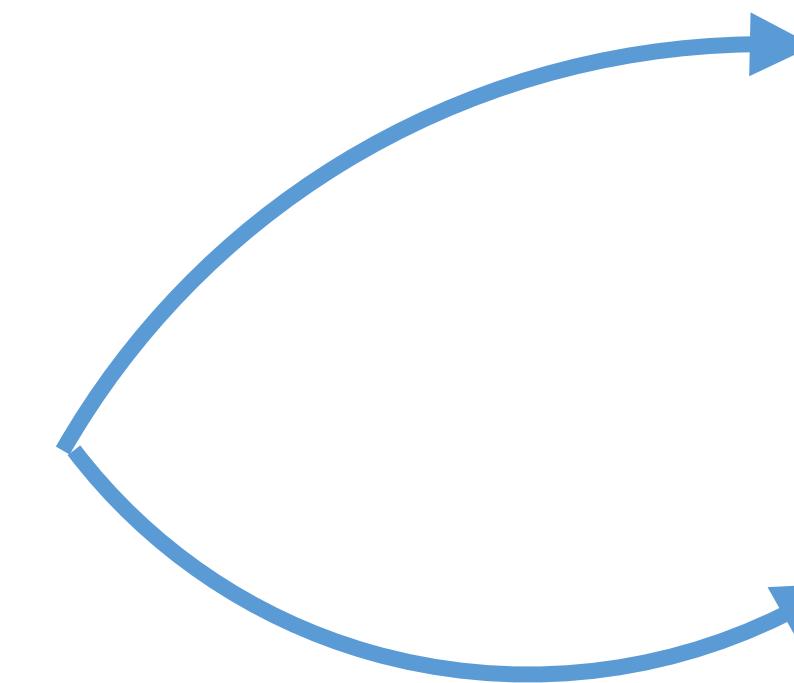
Ambiguity:

a situation in which something has more than one possible meaning and may therefore cause confusion

[Cambridge Dictionary](#)

Do I make myself clear?

- Consider the following requirement
 - "*Attach wheels and engine to frame*"



First, I attach the wheels, and **then** I attach the engine to the frame

I need to attach the wheels and the engine **at the same time** to the frame

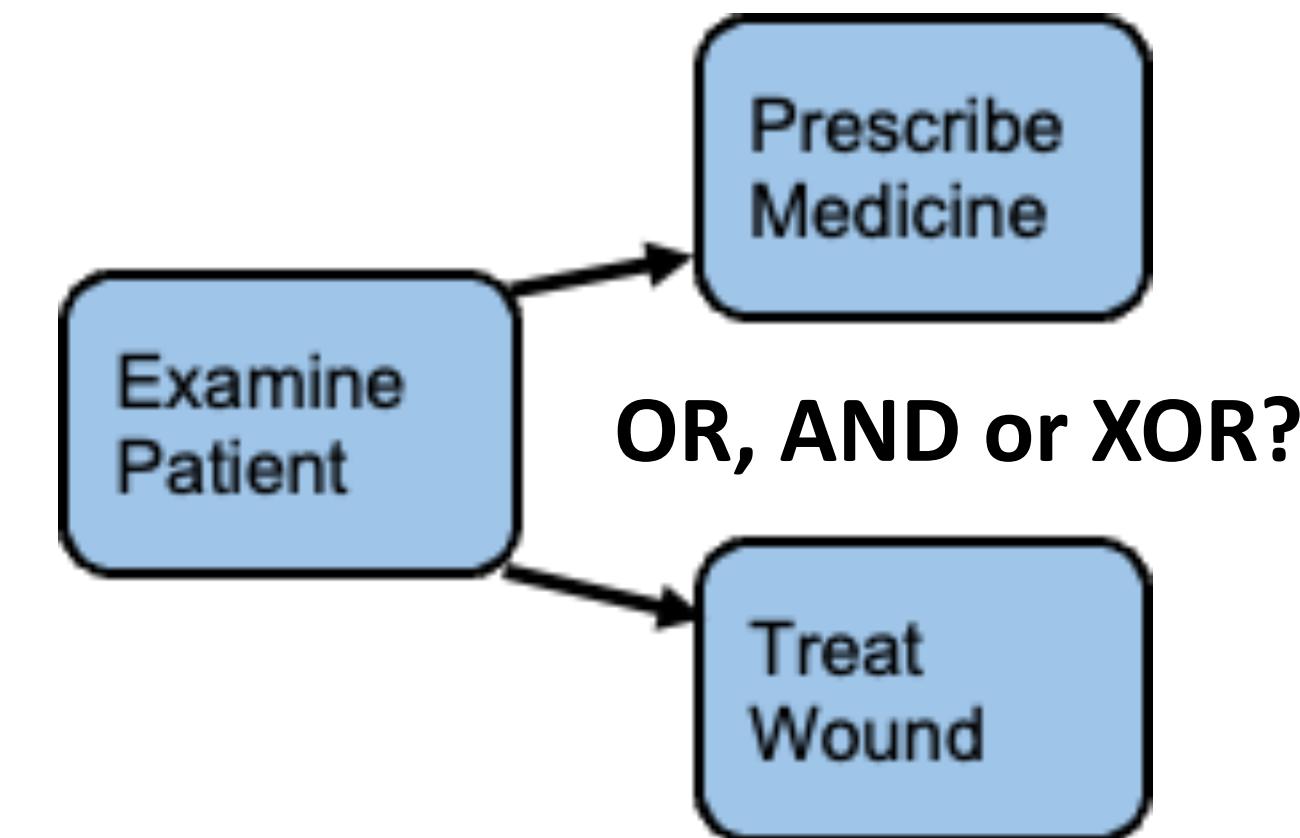
Ambiguity:

a situation in which something has more than one possible meaning and may therefore cause confusion

[Cambridge Dictionary](#)

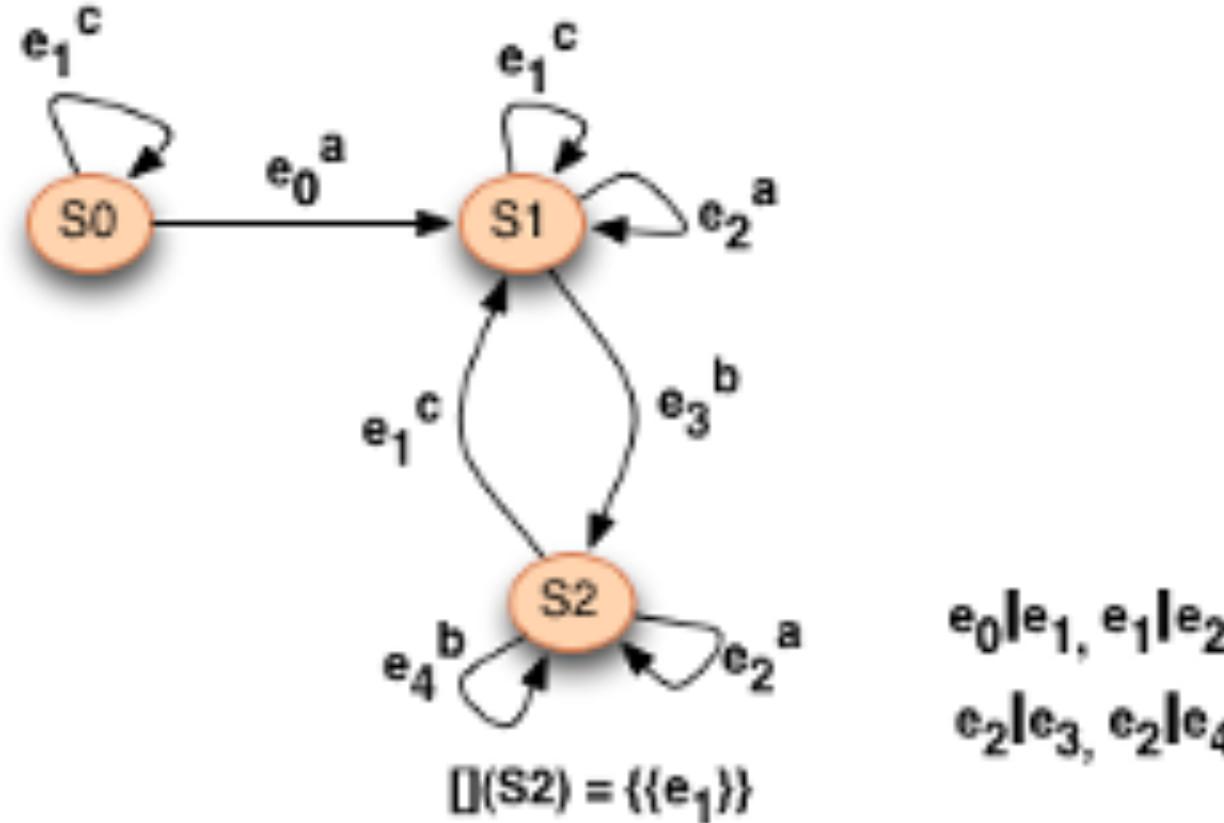
Exercise

In how many ways can you interpret the diagram?

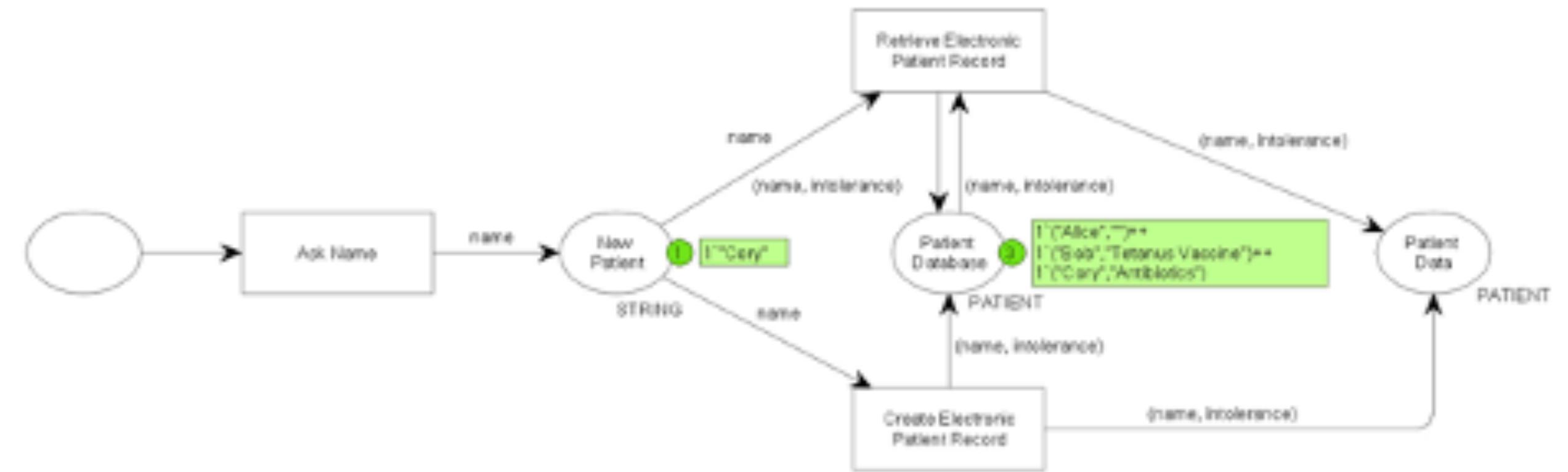


Formal Methods to the Rescue

- We want an unambiguous language to represent valid interpretations of our models



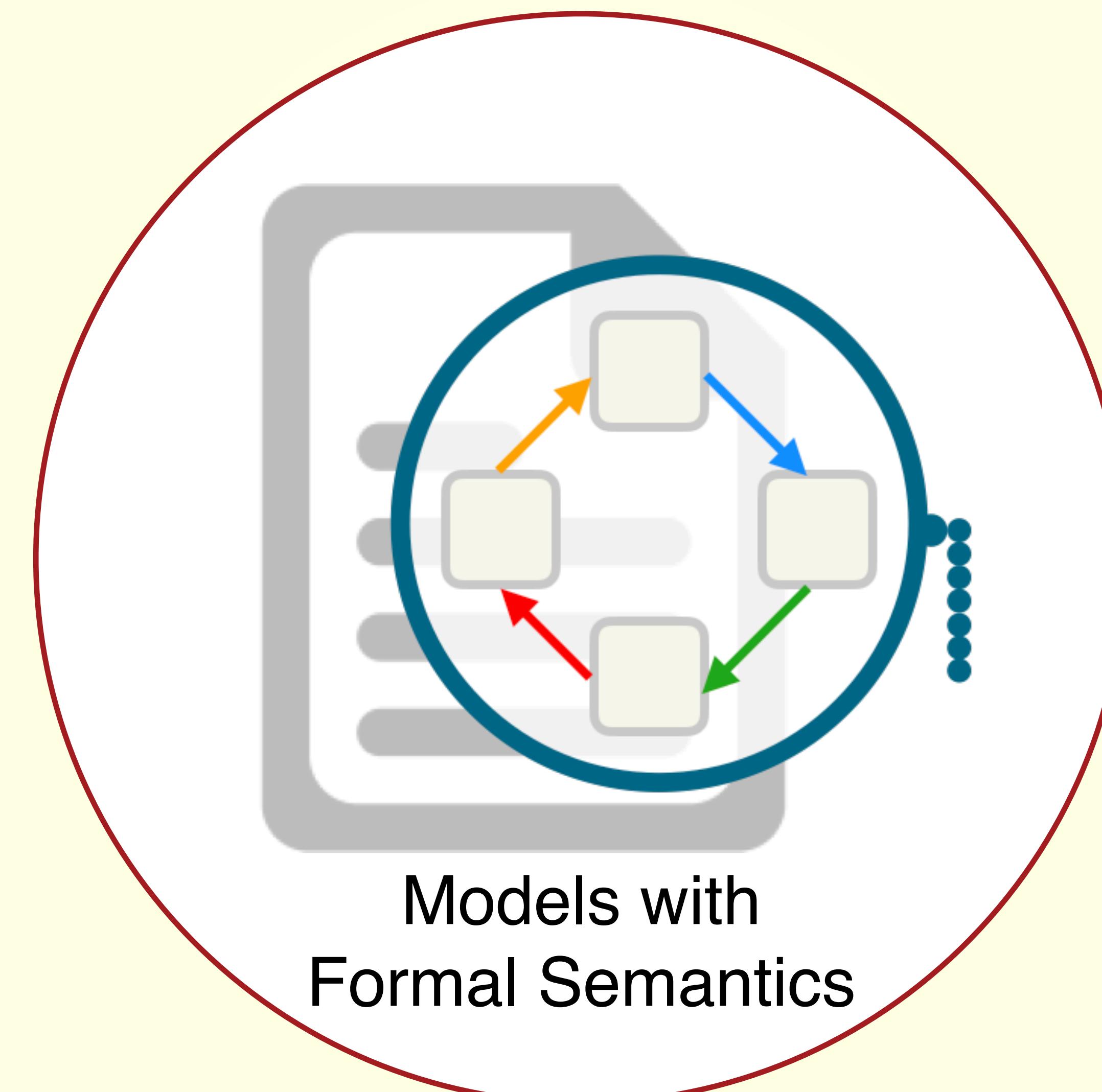
Deterministic Finite Automata or
Non-Deterministic Finite Automata



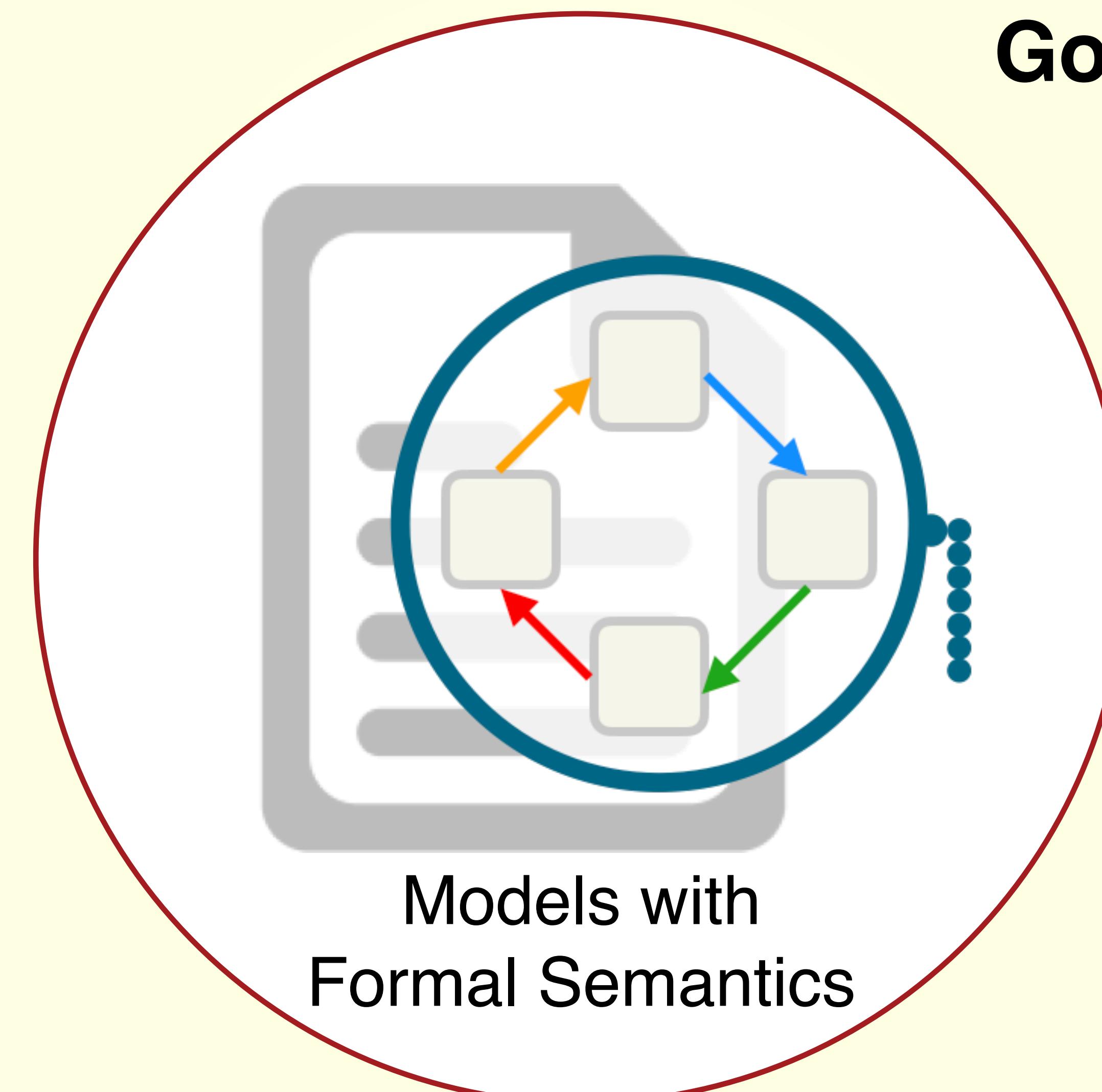
- (Receive Claim $\Rightarrow \diamond$ Evaluate Claim)
- (Approve Claim $\Rightarrow \diamond$ Payout Claim)
- (\neg Payout Claim \wedge Approve Claim)

Modal (Temporal) logics

Benefits of a Formal Semantics



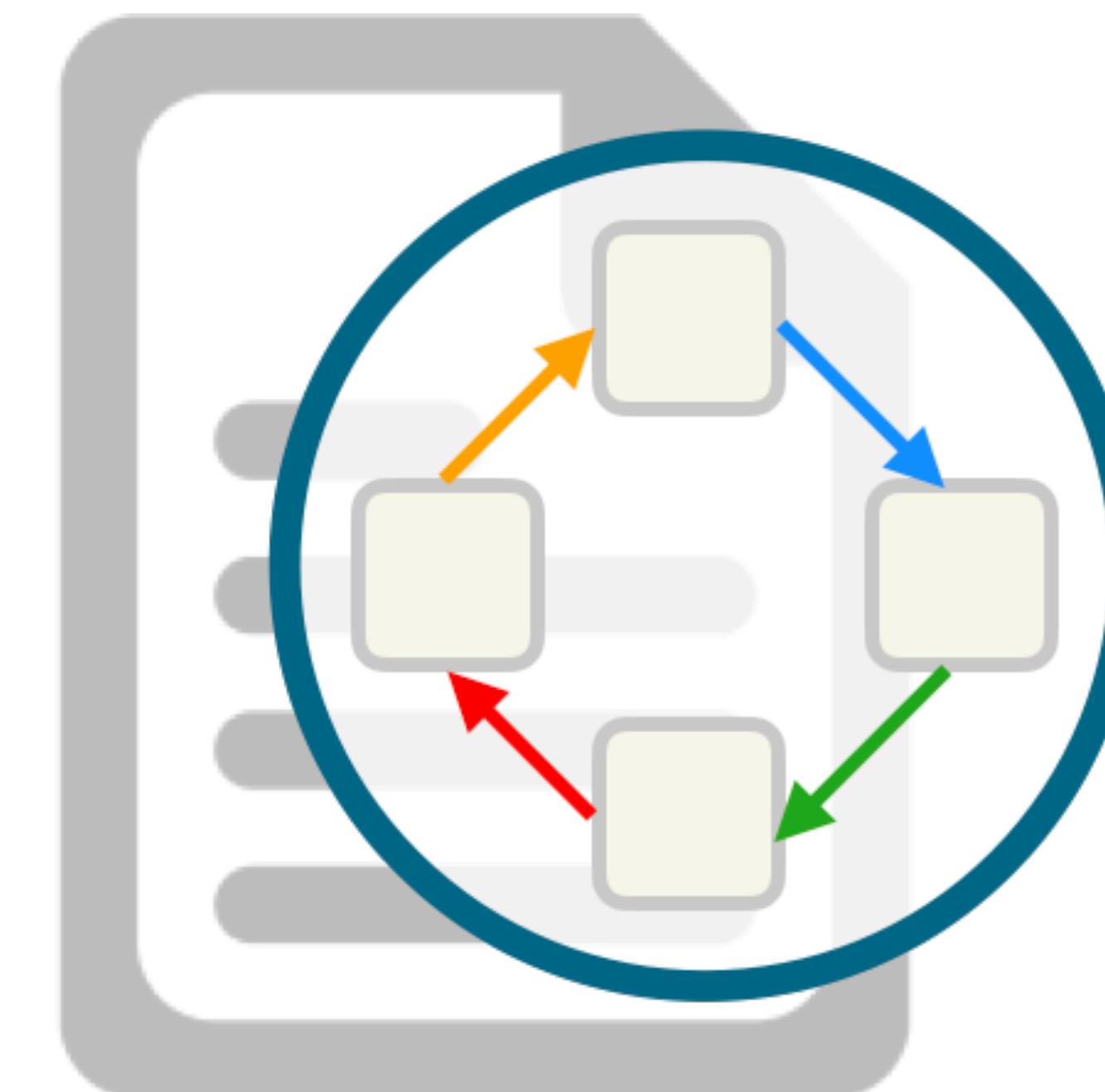
Benefits of a Formal Semantics



Benefits of a Formal Semantics

Semantics-Preserving

Goal-driven



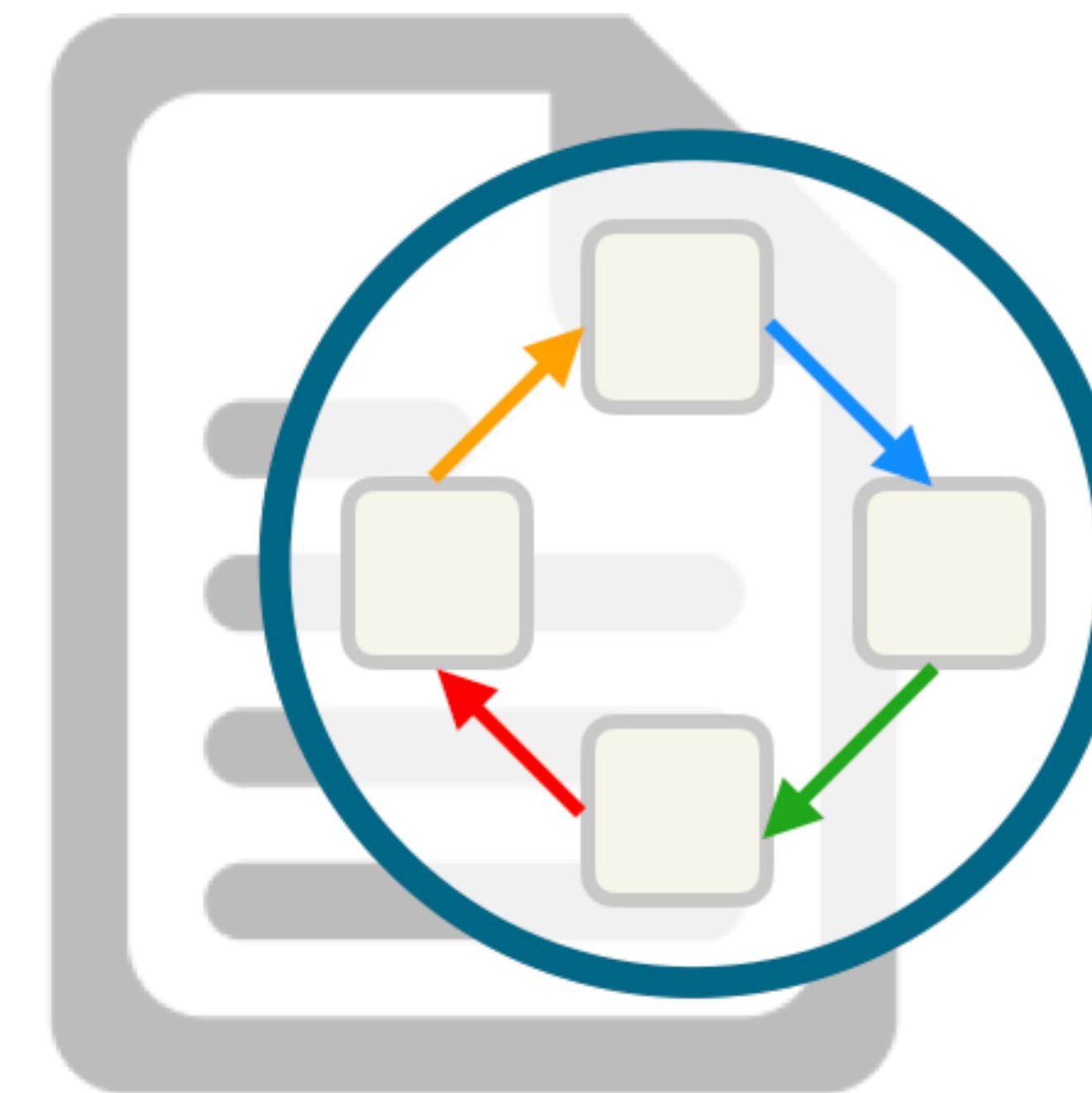
Models with
Formal Semantics

Benefits of a Formal Semantics

Semantics-Preserving

Automated

Goal-driven



Models with
Formal Semantics

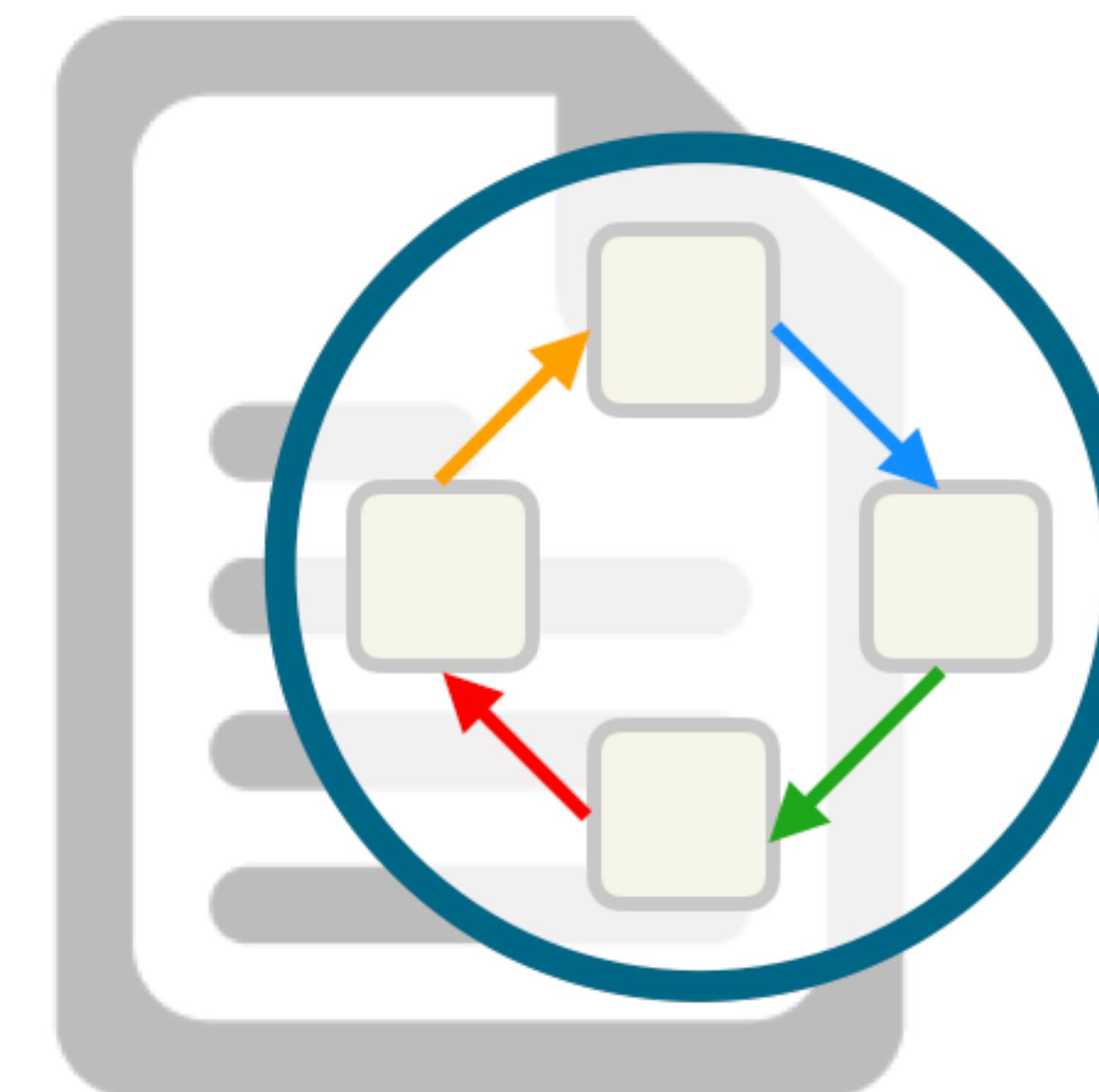
Benefits of a Formal Semantics

Semantics-Preserving

Automated

Goal-driven

Understandable



Models with
Formal Semantics

Benefits of a Formal Semantics

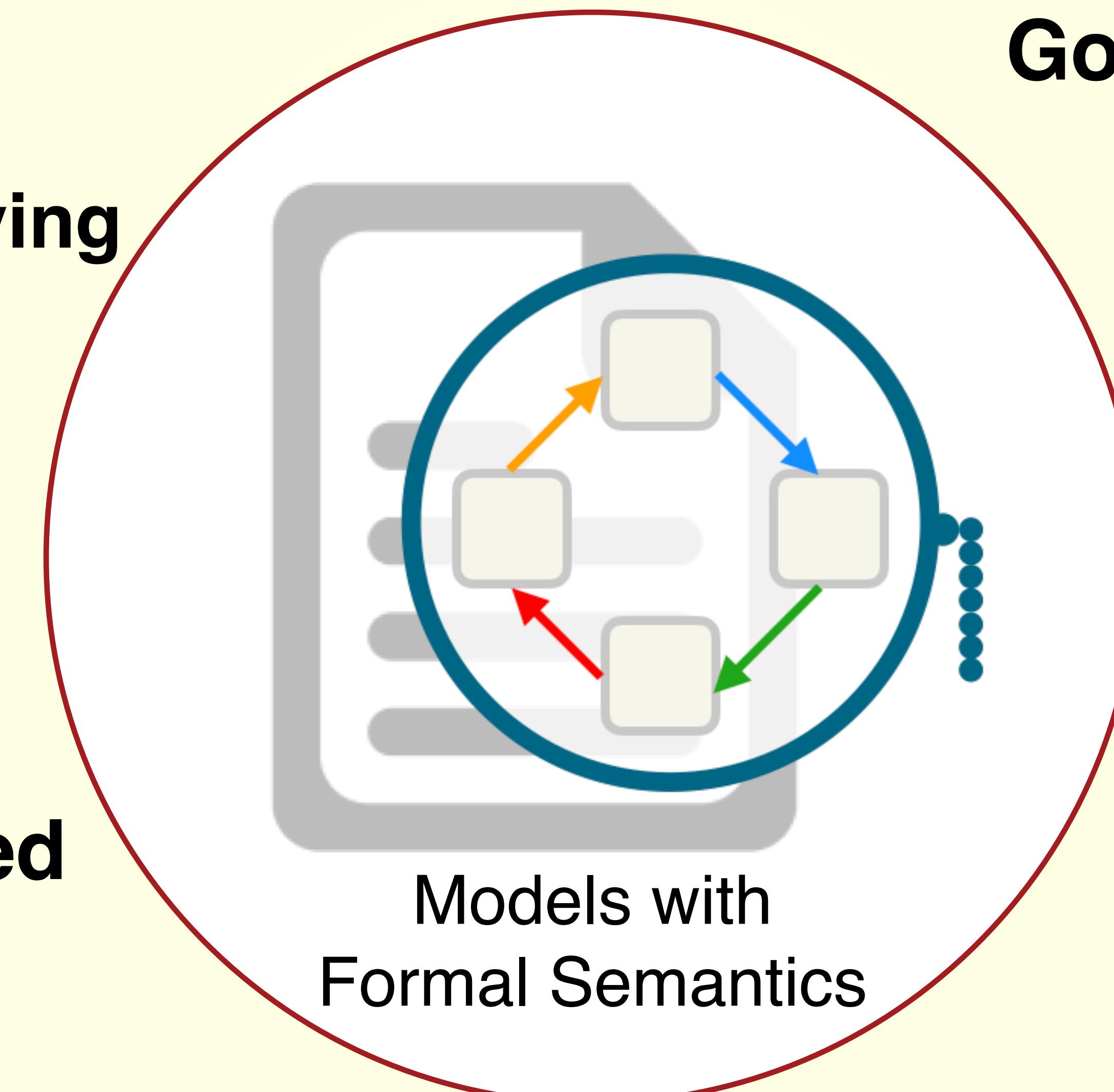
Semantics-Preserving

Automated

Goal-driven

Understandable

Analyzable

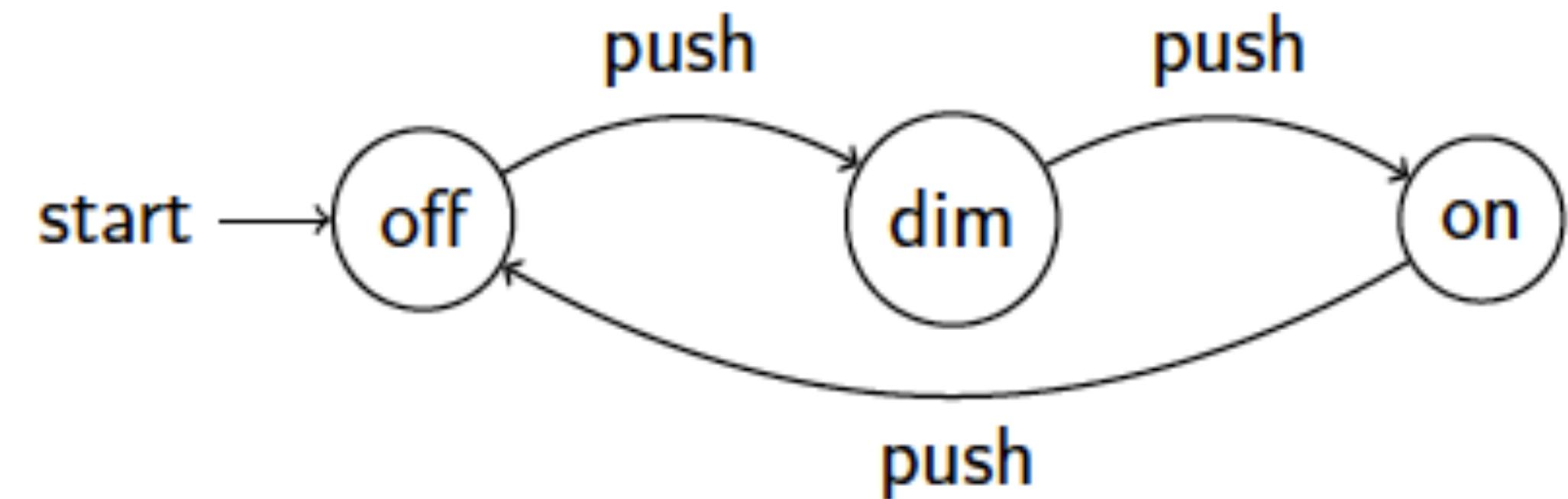
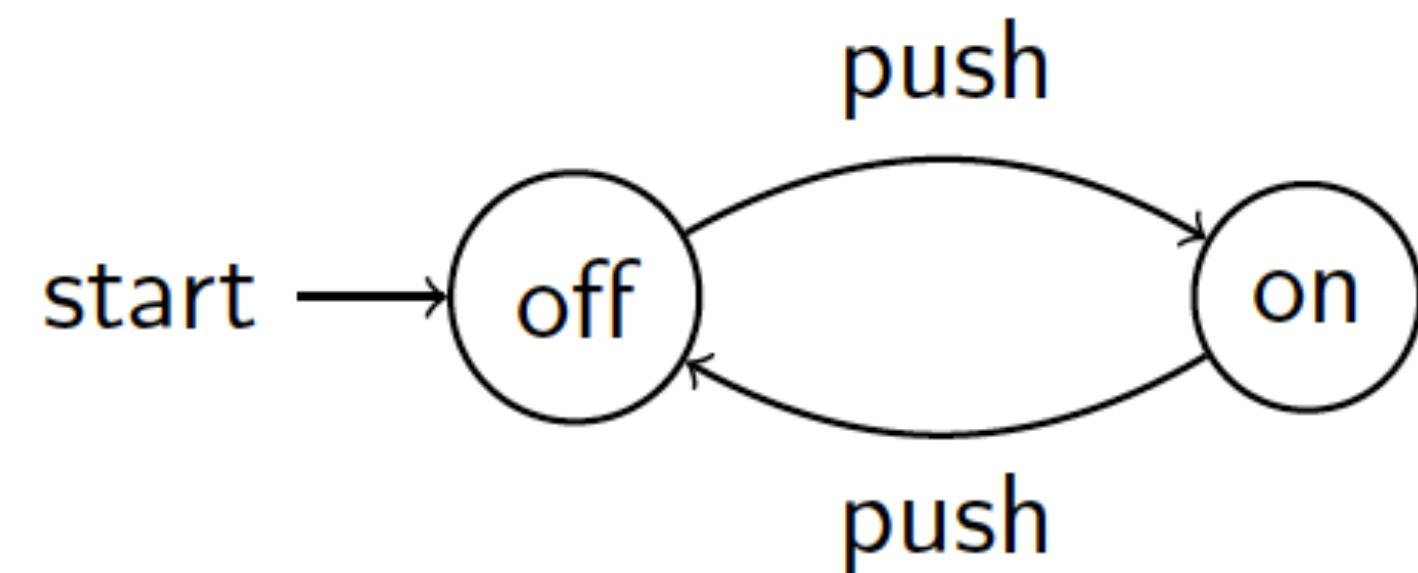


Recap of Formal Languages

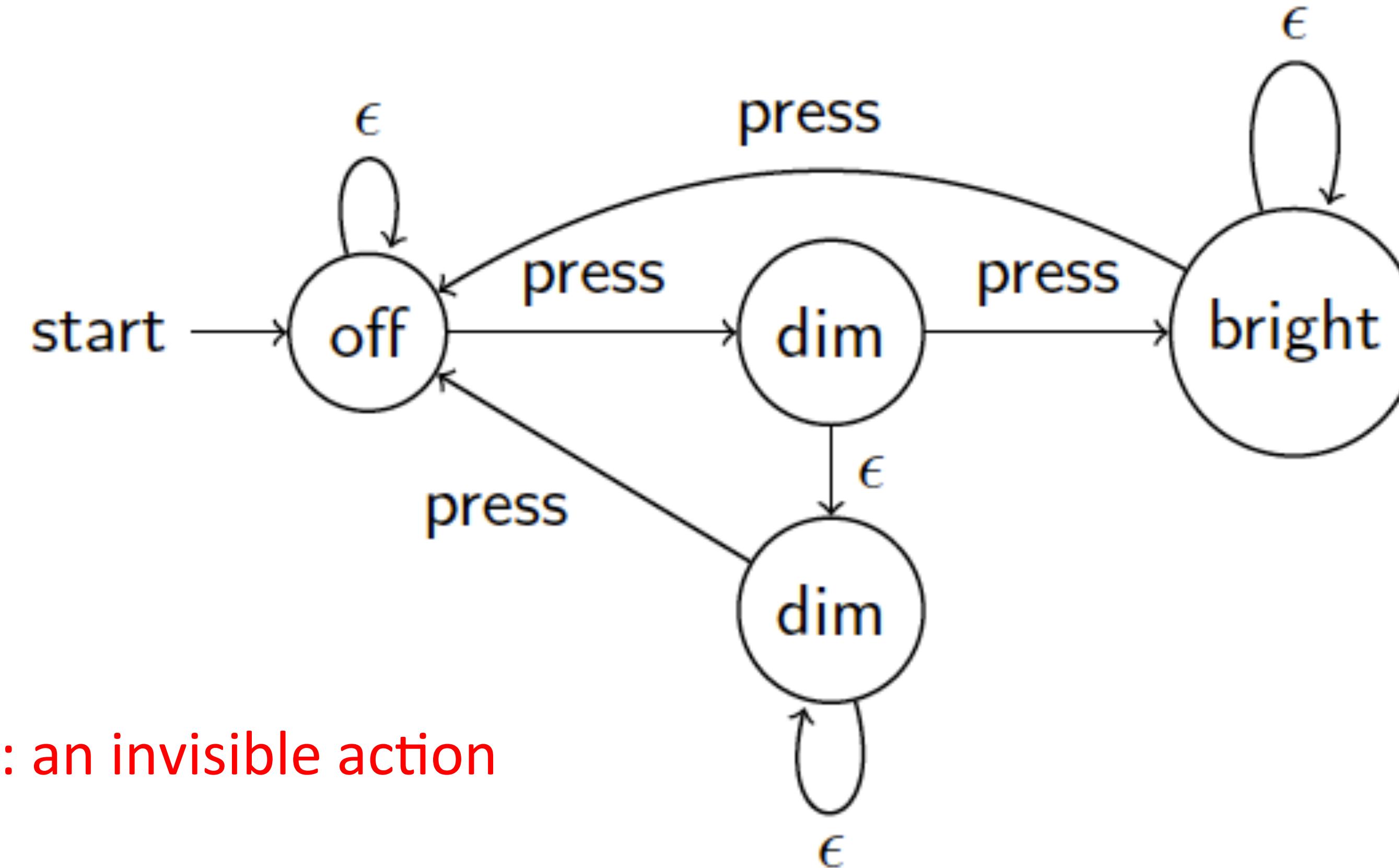
Hugo A. López

(Some material borrowed from Tijs Slaats and Alberto Lluch Lafuente with authorization)

Example: A lamp



Invisible Actions



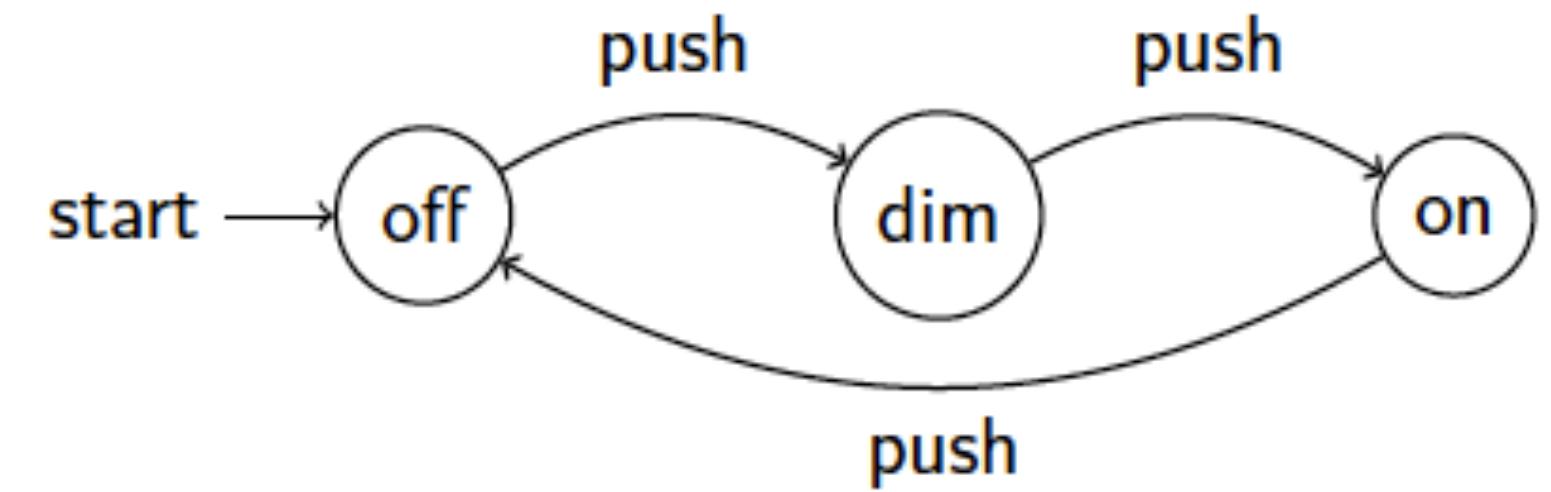
- Epsilon considered non-action, and its symbol might differ depending on the language (we will use tau for the same purpose in the CCS/π calculus lecture)

Alphabets, Strings and Languages

- An *alphabet* Σ is a set of symbols (or letters)
- A *string* $w=a_1, a_2, \dots, a_k$ is a sequence of symbols from Σ
- The empty string is written ϵ
- A language L is a set of strings over Σ , that is, a subset of Σ^*
 - It includes the empty language: \emptyset
 - It includes the language of the empty string: $\{\epsilon\}$

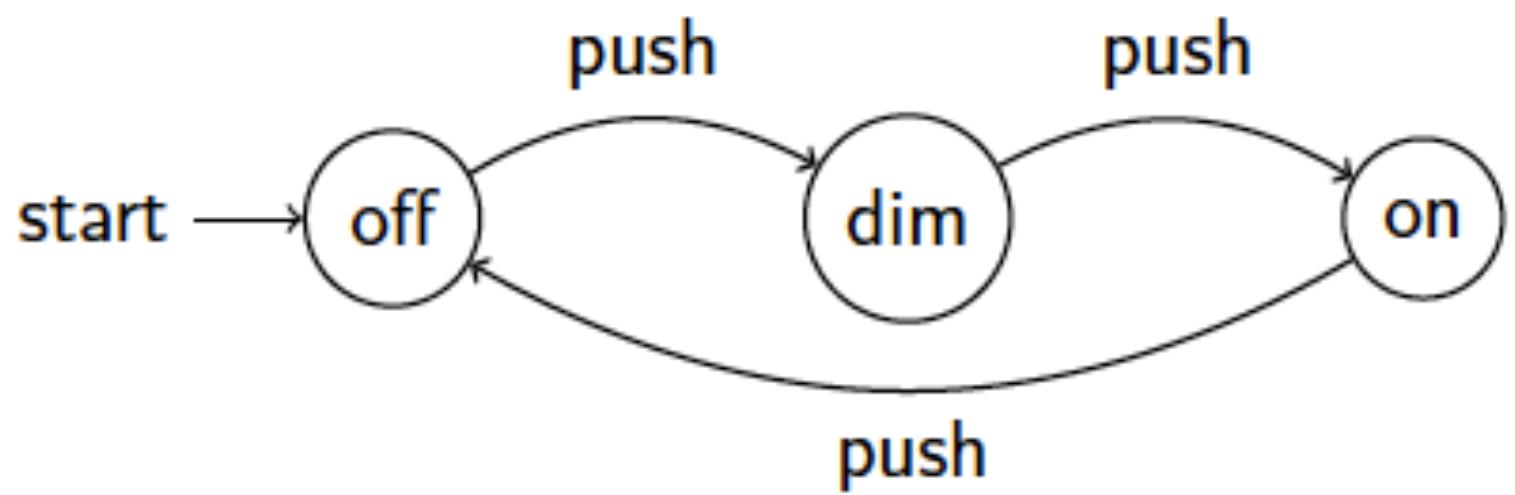
Alphabets, Strings and Languages

- An *alphabet* Σ is a set of symbols (or letters)
- A *string* $w=a_1, a_2, \dots, a_k$ is a sequence of symbols from Σ
- The empty string is written ϵ
- A language L is a set of strings over Σ , that is, a subset of Σ^*
 - It includes the empty language: \emptyset
 - It includes the language of the empty string: $\{\epsilon\}$



Alphabets, Strings and Languages

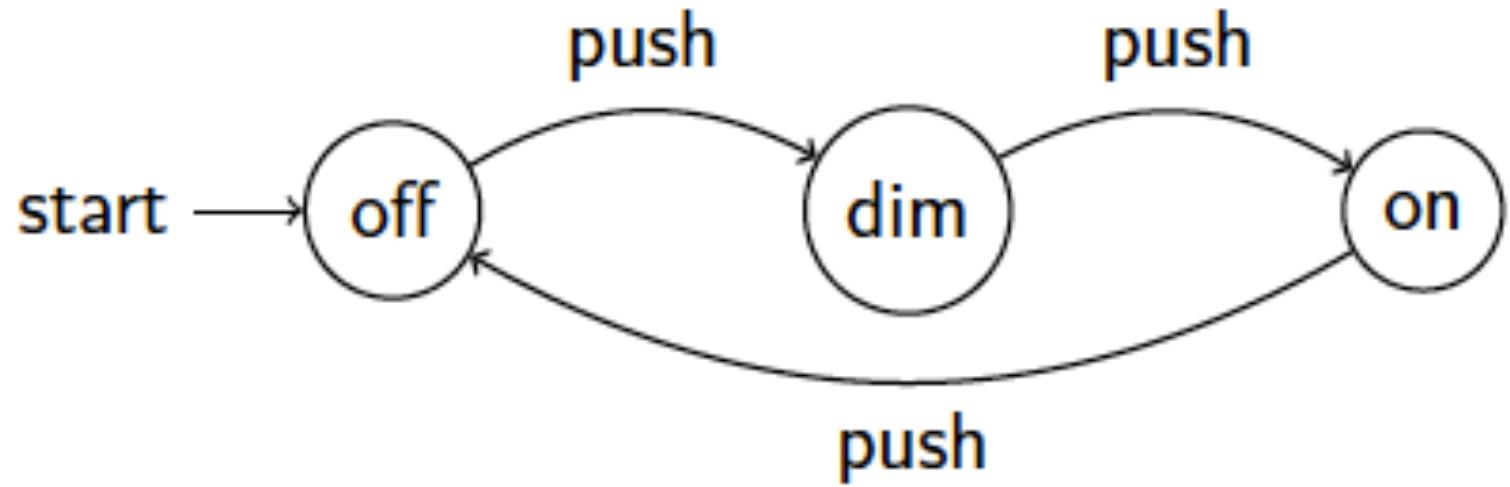
- An *alphabet* Σ is a set of symbols (or letters)
- A *string* $w=a_1, a_2, \dots, a_k$ is a sequence of symbols from Σ
- The empty string is written ϵ
- A language L is a set of strings over Σ , that is, a subset of Σ^*
 - It includes the empty language: \emptyset
 - It includes the language of the empty string: $\{\epsilon\}$



$\Sigma=\{push\}$

Alphabets, Strings and Languages

- An *alphabet* Σ is a set of symbols (or letters)
- A *string* $w=a_1, a_2, \dots, a_k$ is a sequence of symbols from Σ
- The empty string is written ϵ
- A language L is a set of strings over Σ , that is, a subset of Σ^*
 - It includes the empty language: \emptyset
 - It includes the language of the empty string: $\{\epsilon\}$



$\Sigma = \{\text{push}\}$

$w_1 = \text{push}, \text{push};$
 $w_2 = \text{push}, \text{push}, \text{push};$
 $w_3 =$
 $\text{push}, \text{push}, \text{push}, \text{push}, \text{push}, \text{push};$
...

Alphabets, Strings and Languages

- An *alphabet* Σ is a set of symbols (or letters)

$\Sigma = \{\text{push}\}$

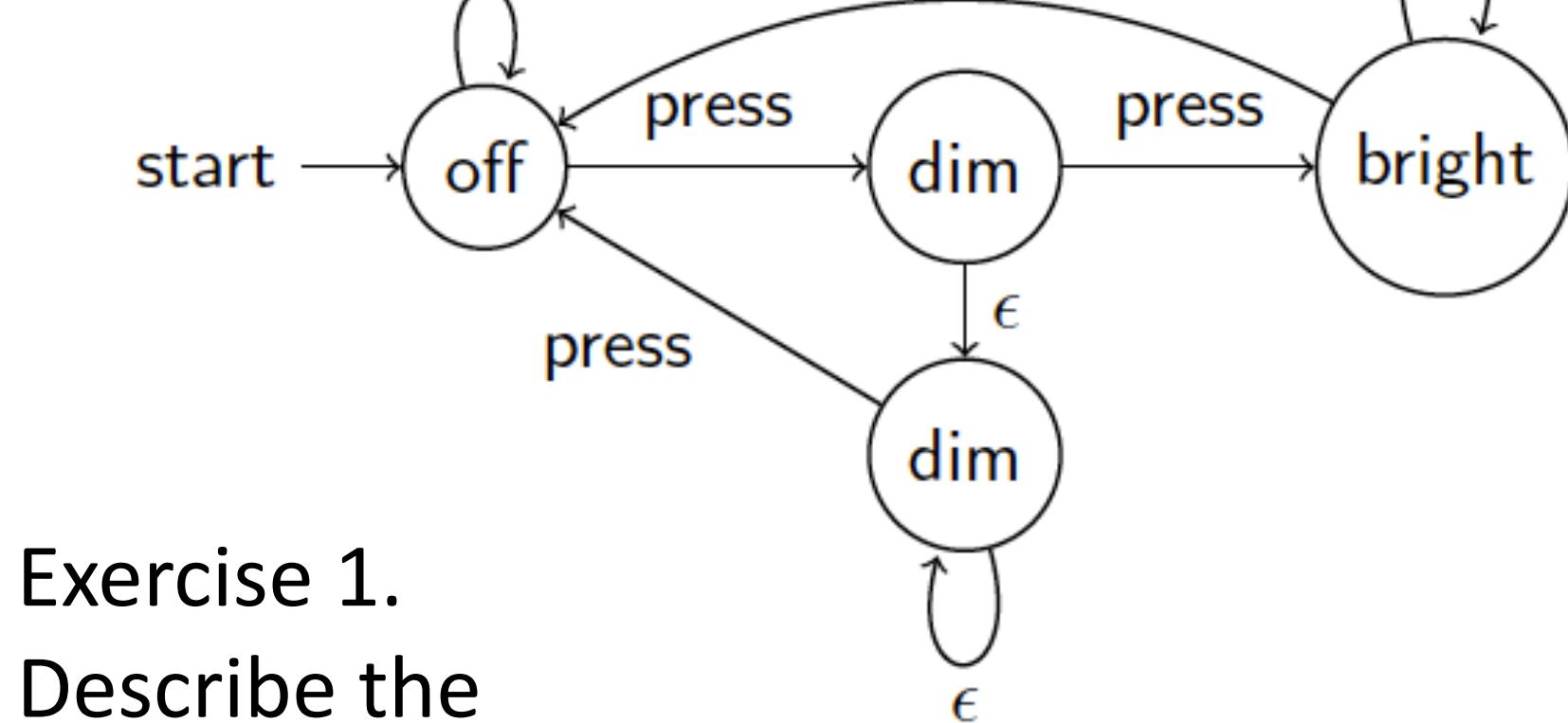
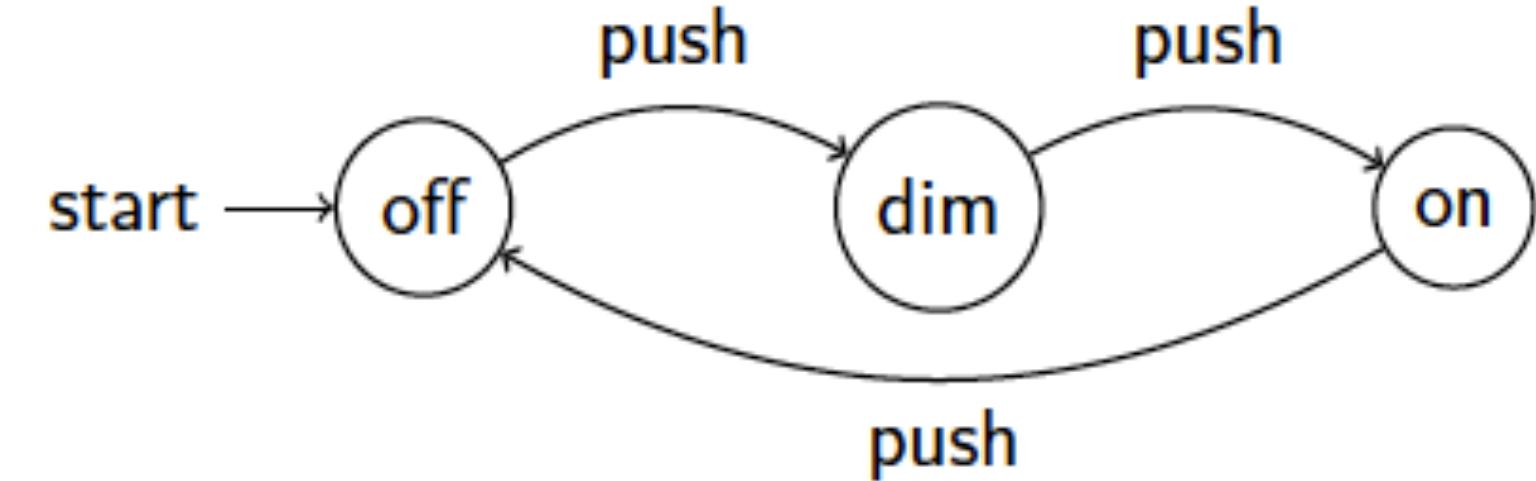
- A *string* $w = a_1, a_2, \dots, a_k$ is a sequence of symbols from Σ

- The empty string is written ϵ

$w_1 = \text{push}, \text{push};$
 $w_2 = \text{push}, \text{push}, \text{push};$
 $w_3 =$
 $\text{push}, \text{push}, \text{push}, \text{push}, \text{push}, \text{push};$
 \dots

- A language L is a set of strings over Σ , that is, a subset of Σ^*

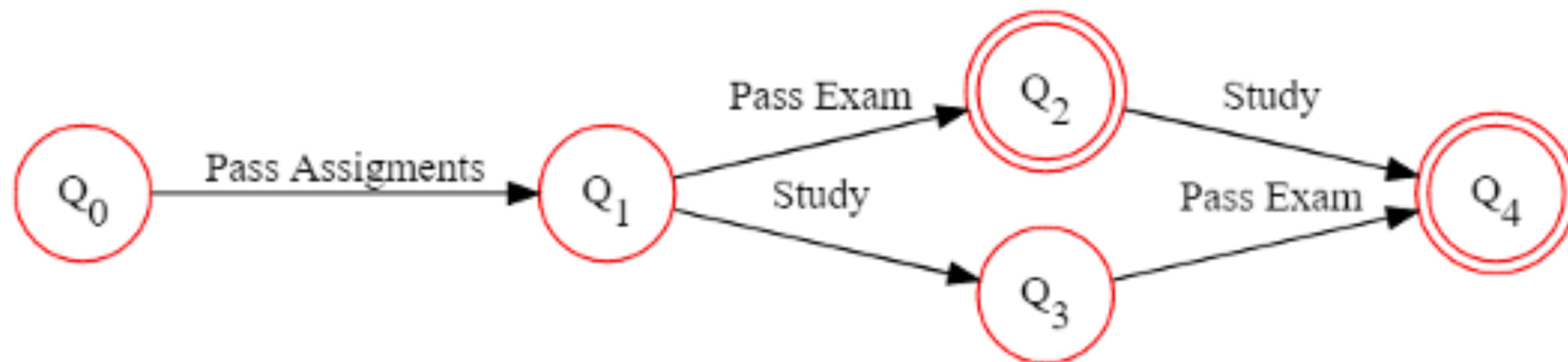
- It includes the empty language: \emptyset
- It includes the language of the empty string: $\{\epsilon\}$



Exercise 1.
Describe the
language of this
automaton

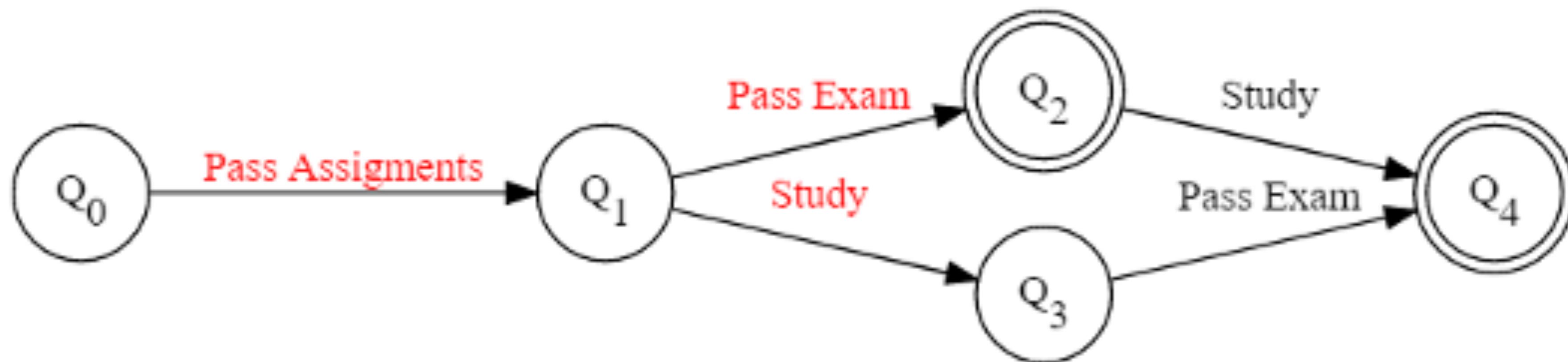
Deterministic Finite Automaton (DFA)

- A deterministic Finite Automaton (DFA) $A=(Q,\Sigma,\delta,q_0,F)$ consists of:
 - A finite set of states \mathbf{Q}
 - An alphabet Σ
 - A transition function $\delta: Q \times \Sigma \rightarrow Q$ for each state in q in Q and each symbol a in Σ , that determines a new state $\delta(q, a)$ in Q
 - A starting state $q_0 \in Q$
 - A set of accepting states $F \subseteq Q$



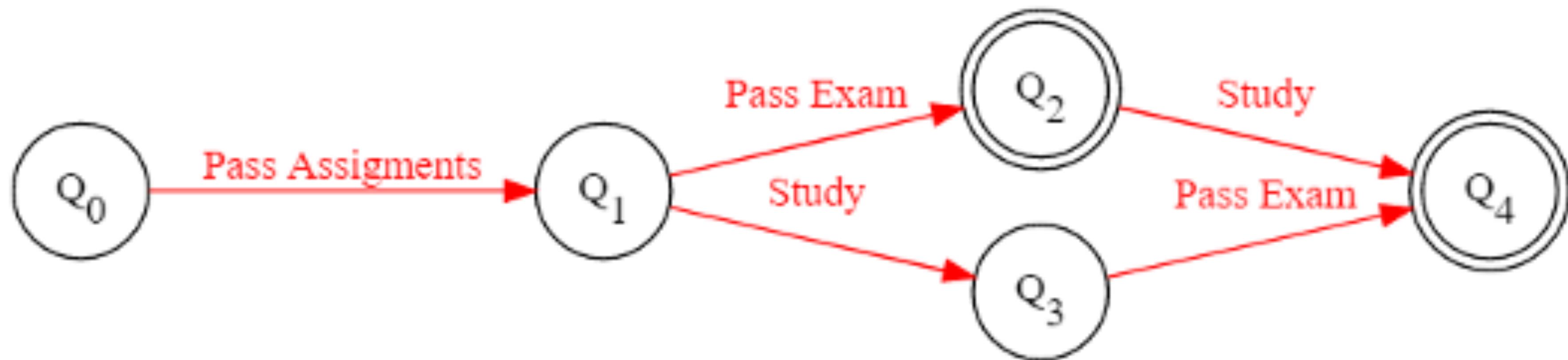
Deterministic Finite Automaton (DFA)

- A deterministic Finite Automaton (DFA) $A=(Q,\Sigma,\delta,q_0,F)$ consists of:
 - A finite set of states **Q**
 - An alphabet **Σ**
 - A transition function $\delta: Q \times \Sigma \rightarrow Q$ for each state in q in Q and each symbol a in Σ , that determines a new state $\delta(q, a)$ in Q
 - A starting state $q_0 \in Q$
 - A set of accepting states **F** $\subseteq Q$



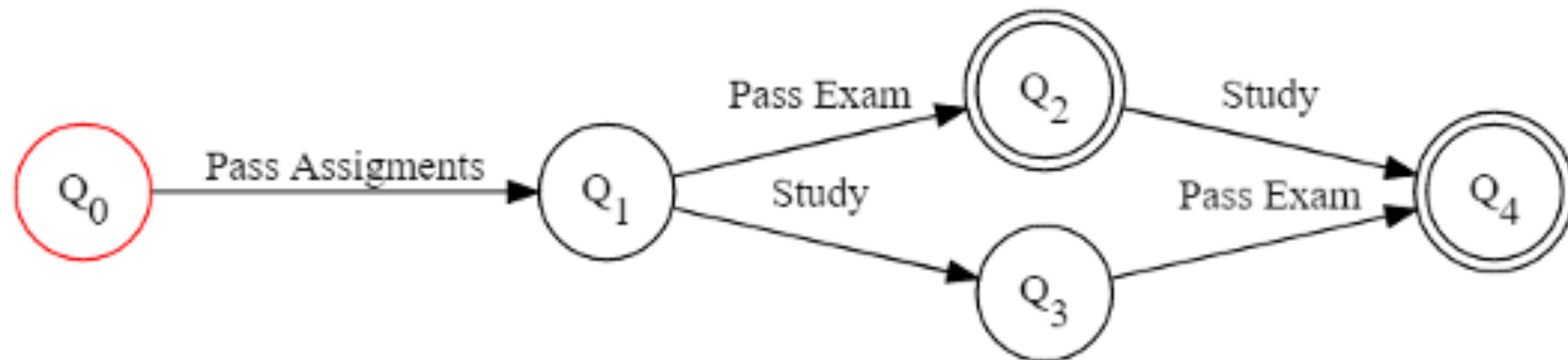
Deterministic Finite Automaton (DFA)

- A deterministic Finite Automaton (DFA) $A=(Q,\Sigma,\delta,q_0,F)$ consists of:
 - A finite set of states \mathbf{Q}
 - An alphabet Σ
 - A transition function $\delta: Q \times \Sigma \rightarrow Q$ for each state in q in Q and each symbol a in Σ , determines a new state $\delta(q, a)$ in Q
 - A starting state $q_0 \in Q$
 - A set of accepting states $F \subseteq Q$



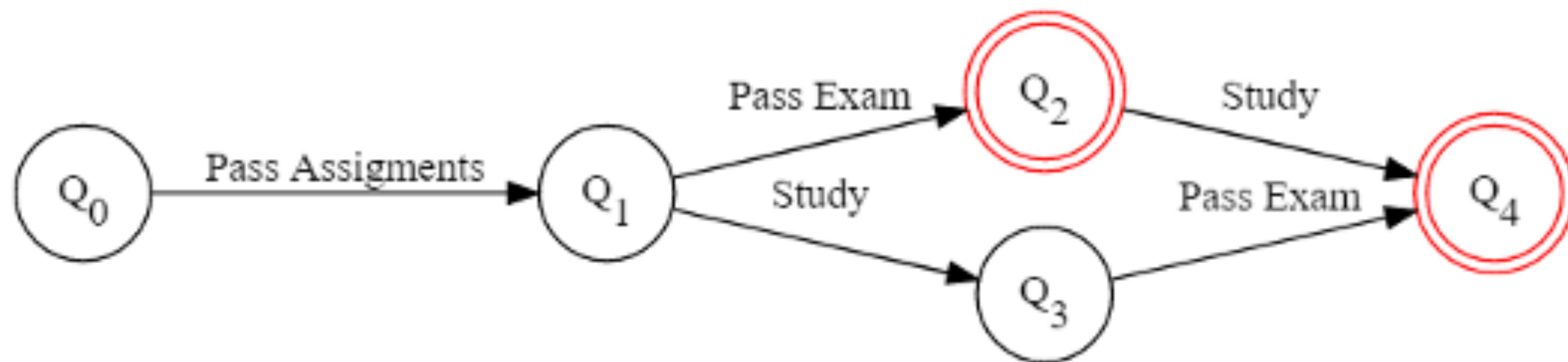
Deterministic Finite Automaton (DFA)

- A deterministic Finite Automaton (DFA) $A=(Q,\Sigma,\delta,q_0,F)$ consists of:
 - A finite set of states $\textcolor{red}{Q}$
 - An alphabet Σ
 - A transition function $\delta: Q \times \Sigma \rightarrow Q$ for each state in q in Q and each symbol a in Σ , determines a new state $\delta(q, a)$ in Q
 - A starting state $q_0 \in Q$
 - A set of accepting states $F \subseteq Q$



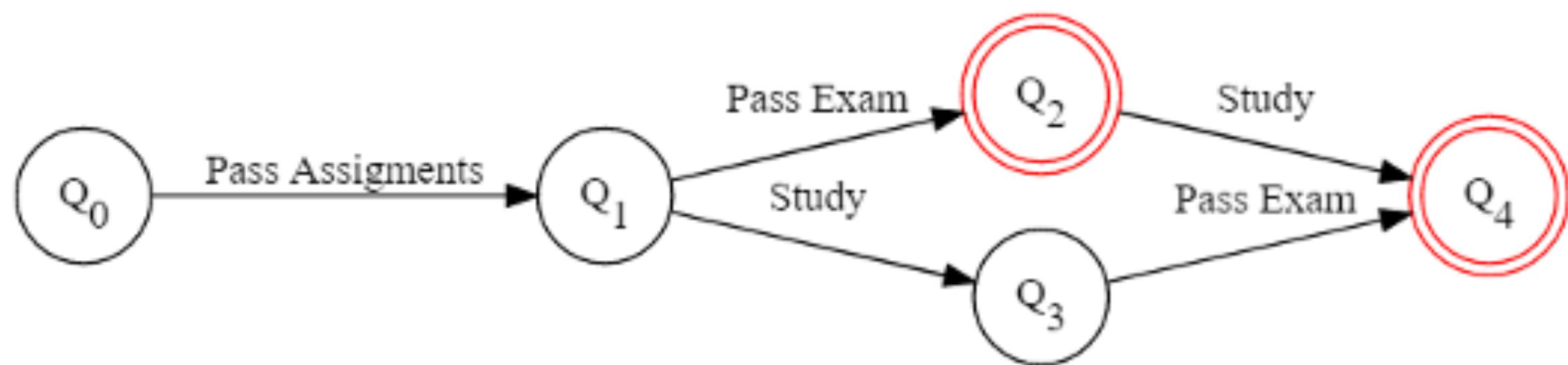
Deterministic Finite Automaton (DFA)

- A deterministic Finite Automaton (DFA) $A=(Q,\Sigma,\delta,q_0,F)$ consists of:
 - A finite set of states Q
 - An alphabet Σ
 - A transition function $\delta: Q \times \Sigma \rightarrow Q$ for each state in q in Q and each symbol a in Σ , determines a new state $\delta(q, a)$ in Q
 - A starting state $q_0 \in Q$
 - A set of accepting states $F \subseteq Q$
 - For each symbolic representation of the alphabet, there is only one state transition in A



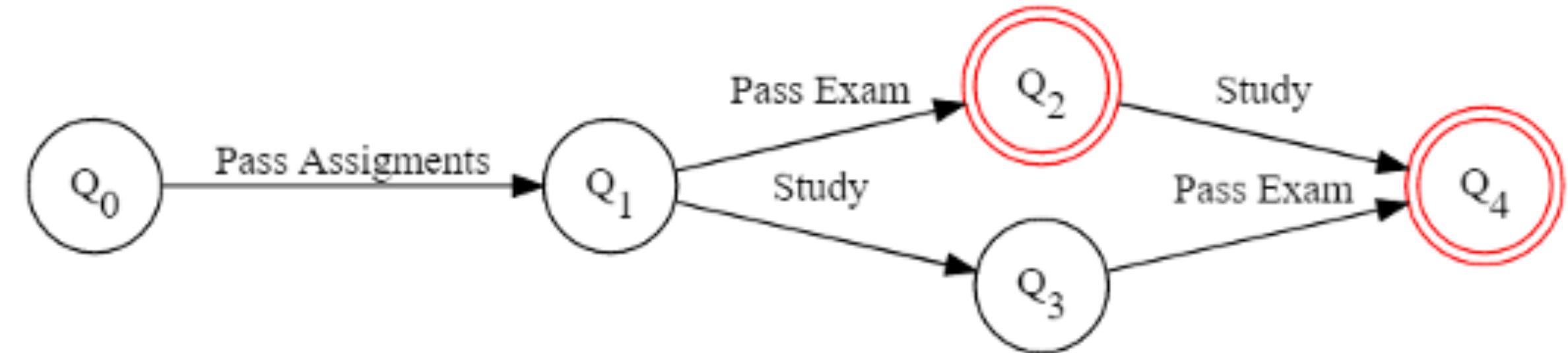
Transition Function

$\delta: Q \times \Sigma$	Pass Assignments	Pass Exam	Study
Q_0	Q_1	Q_0	Q_0
Q_1	Q_1	Q_2	Q_3
Q_2	Q_2	Q_2	Q_4
Q_3	Q_3	Q_4	Q_3
Q_4	Q_4	Q_4	Q_4



How does a DFA define a language?

- Consider a string w
- Start in the initial state Q_0
- Read the first symbol, say *Pass Assignments*, of w
- Determine the new state $Q_1 = \delta(Q_0, \text{Pass Assignments})$
- Read the second symbol, say *Pass Exam*, of w
- Determine the new state $Q_2 = \delta(Q_1, \text{Pass Exam})$
- ...
- Let Q_k be the state obtained after having read the last symbol of w
- If Q_k is in F then accept w ; otherwise reject w



Exercise 2:

How many words exist in the language above?

On Non-Determinism

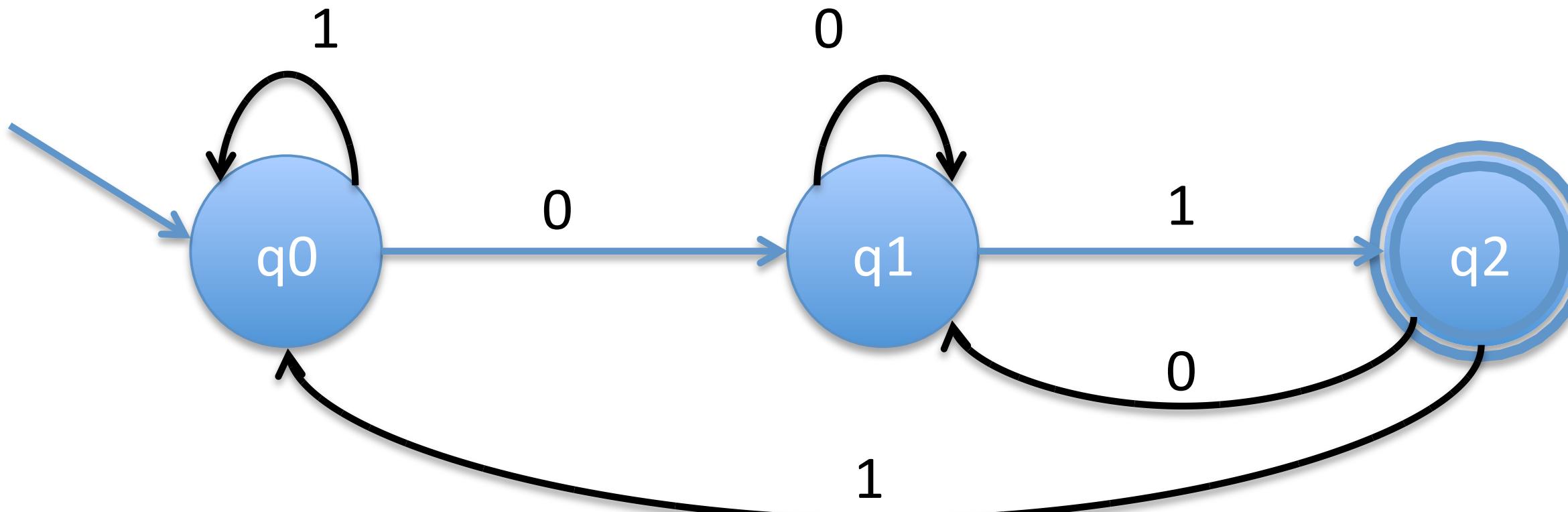
THE FREE WILL MACHINE



by 'Trick Slattery'

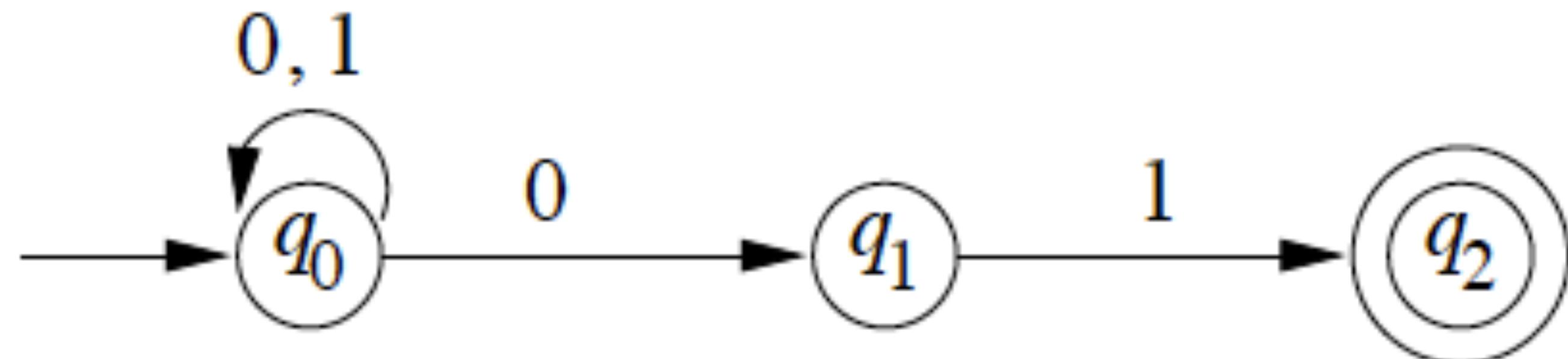
Non-determinism

DFA



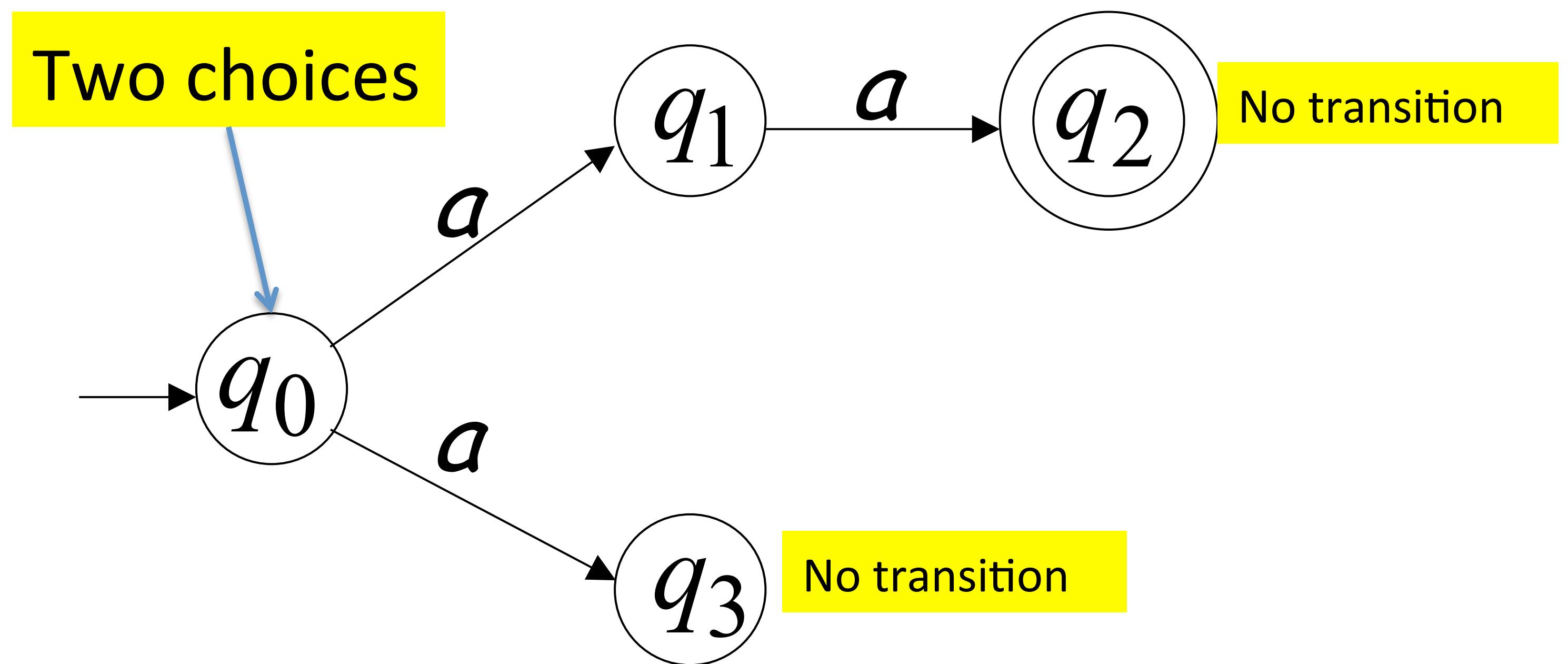
- We use non-deterministic transitions to capture how multiple symbols (including the empty string) may lead to the same state

NFA



Non-Deterministic Finite State Automata

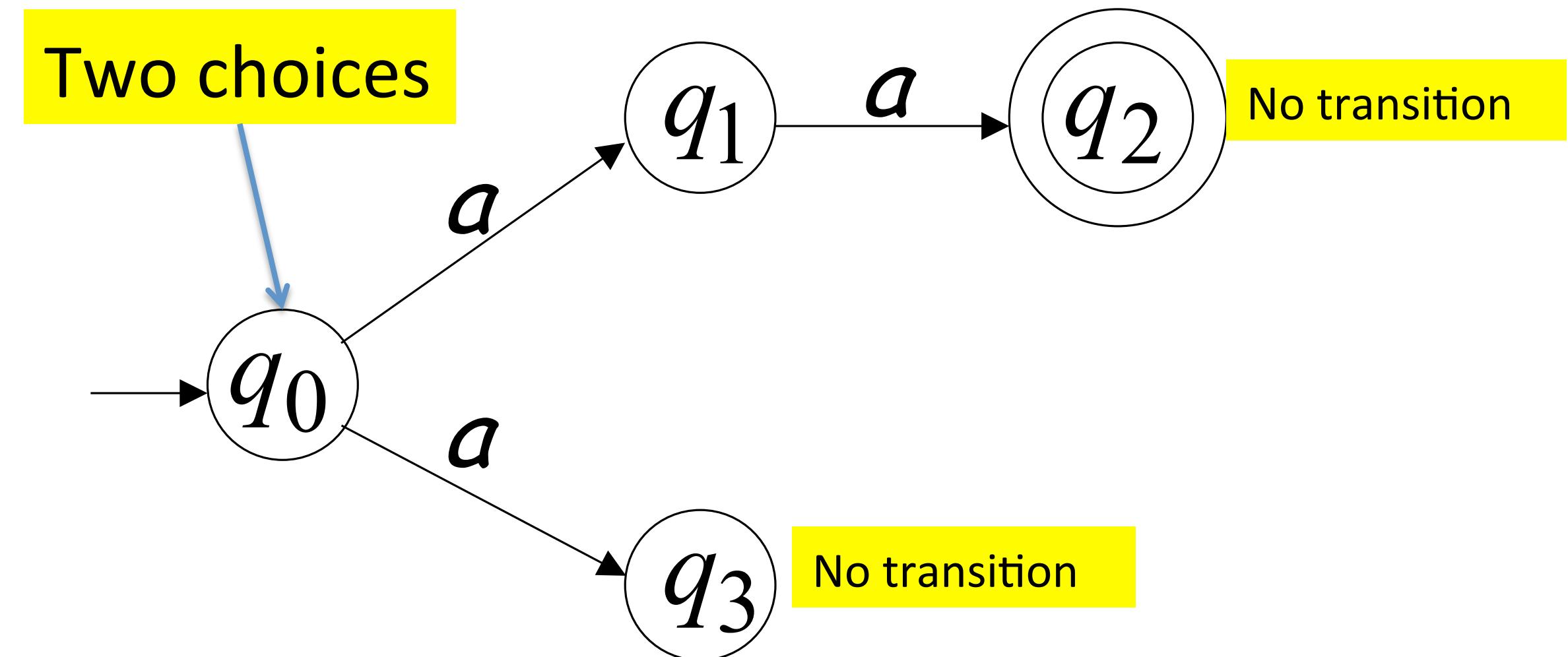
- What are the accepted strings?



Non-Deterministic Finite State Automata

- What are the accepted strings?

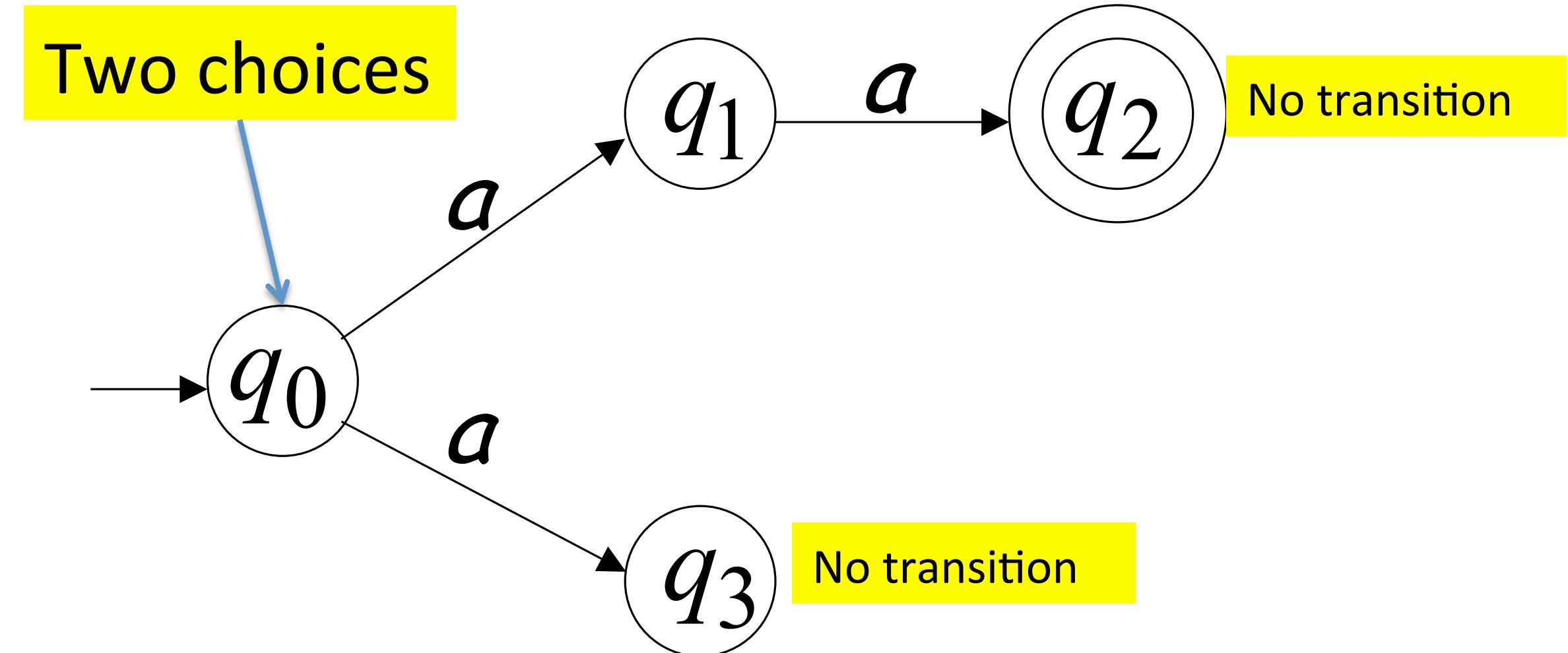
	Input	State sequence	Input consumed?	Stops in final state?	Accepted?
aa	$q_0 q_1 q_2$	✓	✓	✓	✓
aa	$q_0 q_3$	X	X	X	✓
a					
a					
aaa					
aaa					



Non-Deterministic Finite State Automata

- What are the accepted strings?

	Input	State sequence	Input consumed?	Stops in final state?	Accepted?
aa	$q_0 q_1 q_2$	✓	✓	✓	✓
aa	$q_0 q_3$	✗	✗	✗	✓
a					
a					
aaa					
aaa					



An NFA accepts a string w if and only if there is **at least one** computation of the NFA that accepts w i.e. a computation that:

- consumes the entire input, and
- it stops in a final state

NFA, Formally

- A non-deterministic finite automaton $A=(Q,\Sigma,\delta,q_0,F)$ consists of
- Q : a finite set of states
- Σ : an alphabet
- A transition function $\delta: Q \times \Sigma \rightarrow P(Q)$ that for each state q in Q and each symbol a in Σ , **it determines a set of new states** $\delta(q,a)$ in Q
- $q_0 \in Q$: the initial state
- $F \subseteq Q$: the final states

NFA, Formally

- A non-deterministic finite automaton $A=(Q,\Sigma,\delta,q_0,F)$ consists of
- Q : a finite set of states
- Σ : an alphabet
- A transition function $\delta: Q \times \Sigma \rightarrow P(Q)$ that for each state q in Q and each symbol a in Σ , **it determines a set of new states** $\delta(q,a)$ in Q
- $q_0 \in Q$: the initial state
- $F \subseteq Q$: the final states

(All the non-blue text is equal to the definition of a DFA)

NFA, Formally

- A non-deterministic finite automaton $A=(Q,\Sigma,\delta,q_0,F)$ consists of
- Q : a finite set of states
- Σ : an alphabet
- A transition function $\delta: Q \times \Sigma \rightarrow P(Q)$ that for each state q in Q and each symbol a in Σ , **it determines a set of new states** $\delta(q,a)$ in Q
- $q_0 \in Q$: the initial state
- $F \subseteq Q$: the final states

(All the non-blue text is equal to the definition of a DFA)

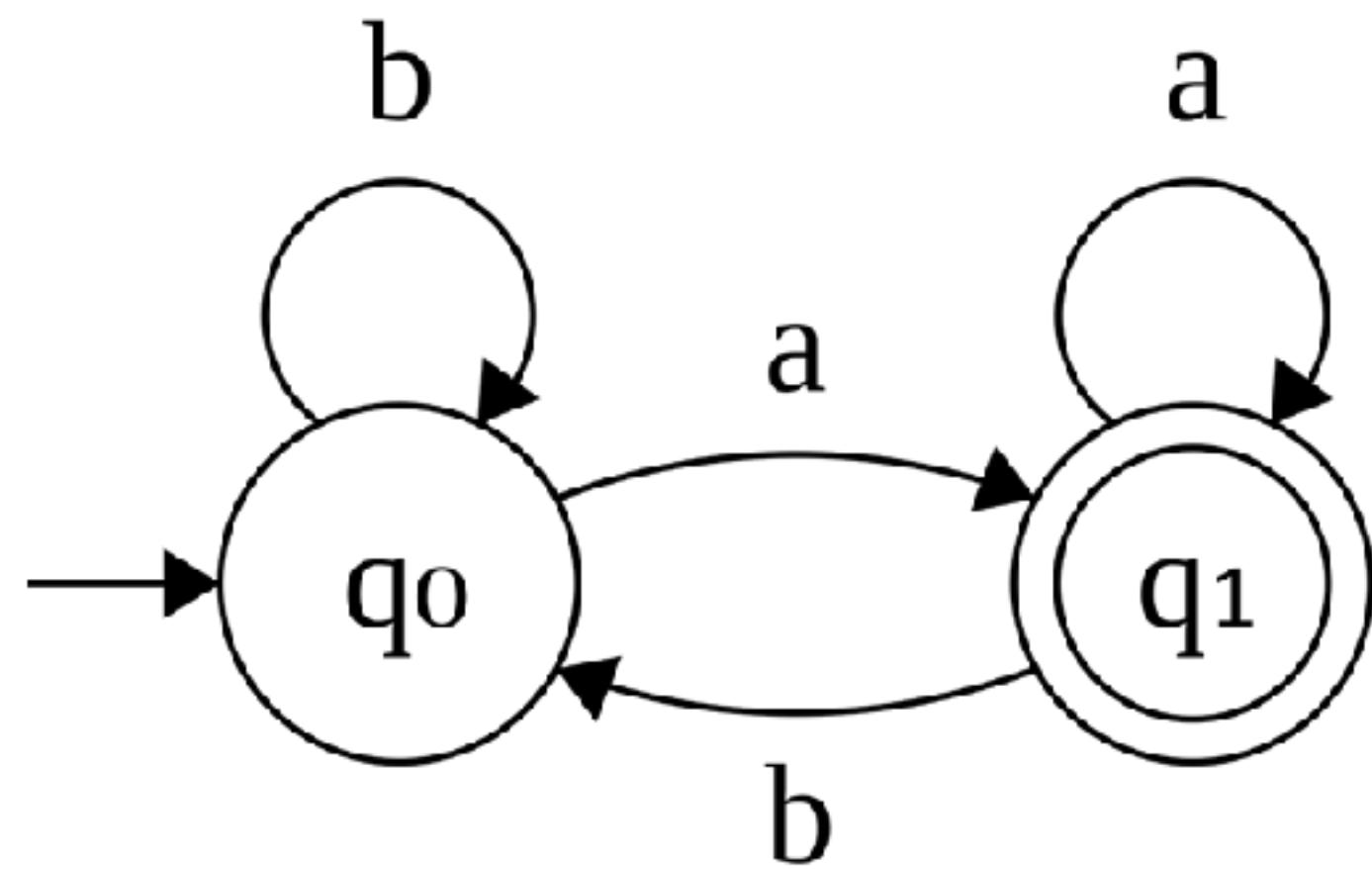
The **language of $A=(Q,\Sigma,\delta,q_0,F)$** is $L(A) = \{w \mid \delta^*(q_0,w) \text{ contains a state from } F\}$

Reasoning about Infinite Systems



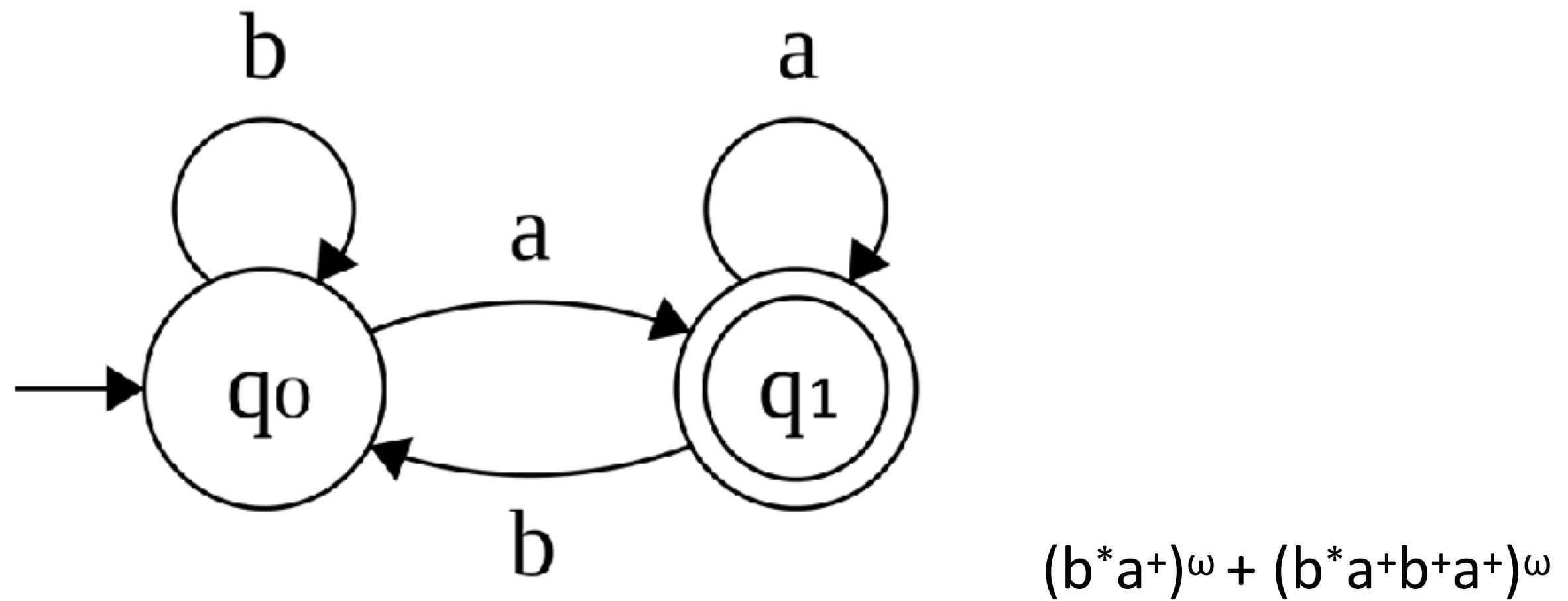
Büchi Automata

Can you enumerate the strings
in the language of this
automata?



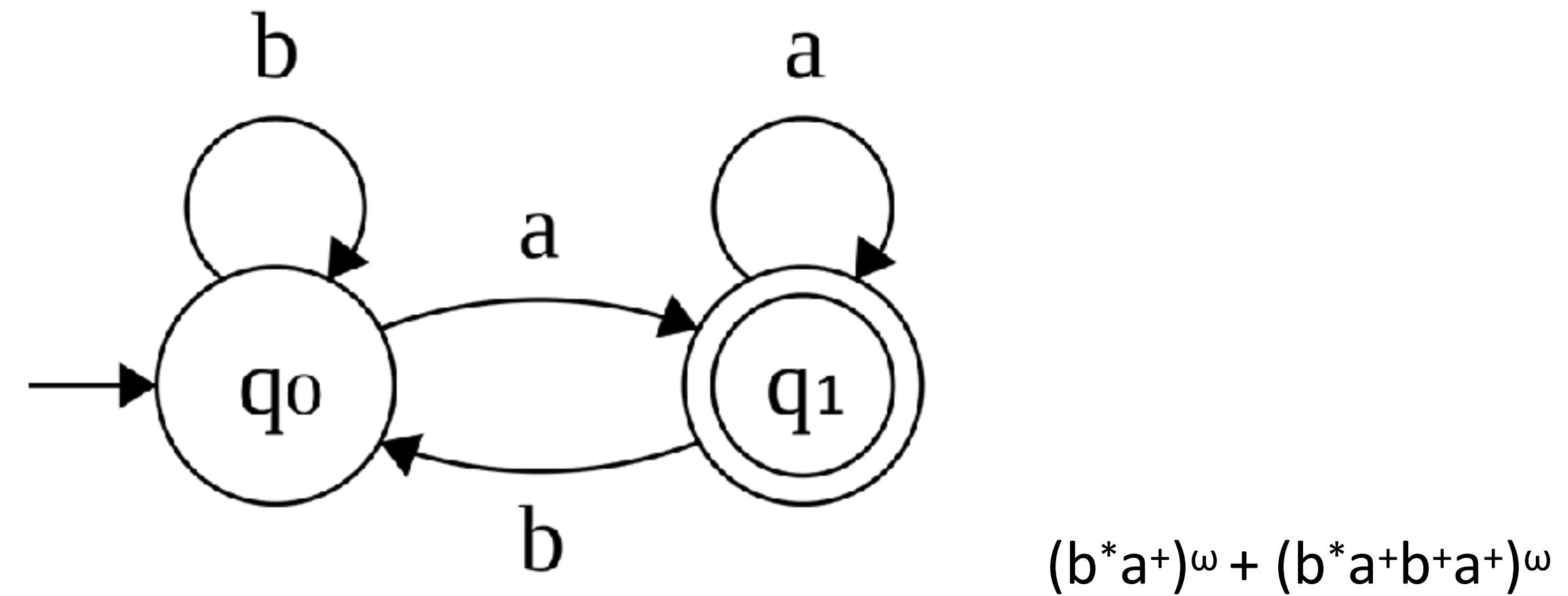
Büchi Automata

Can you enumerate the strings
in the language of this
automata?



Büchi Automata

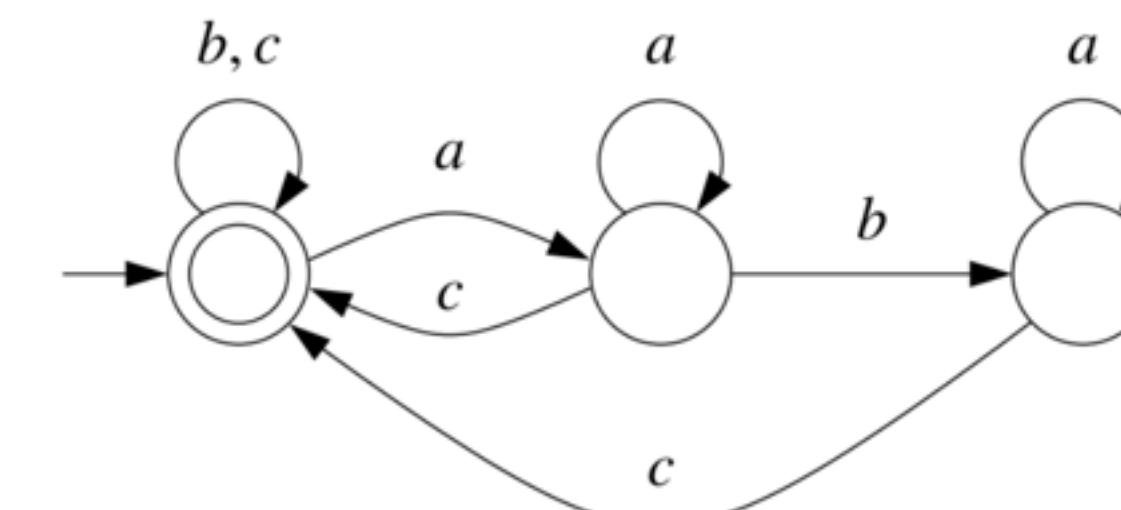
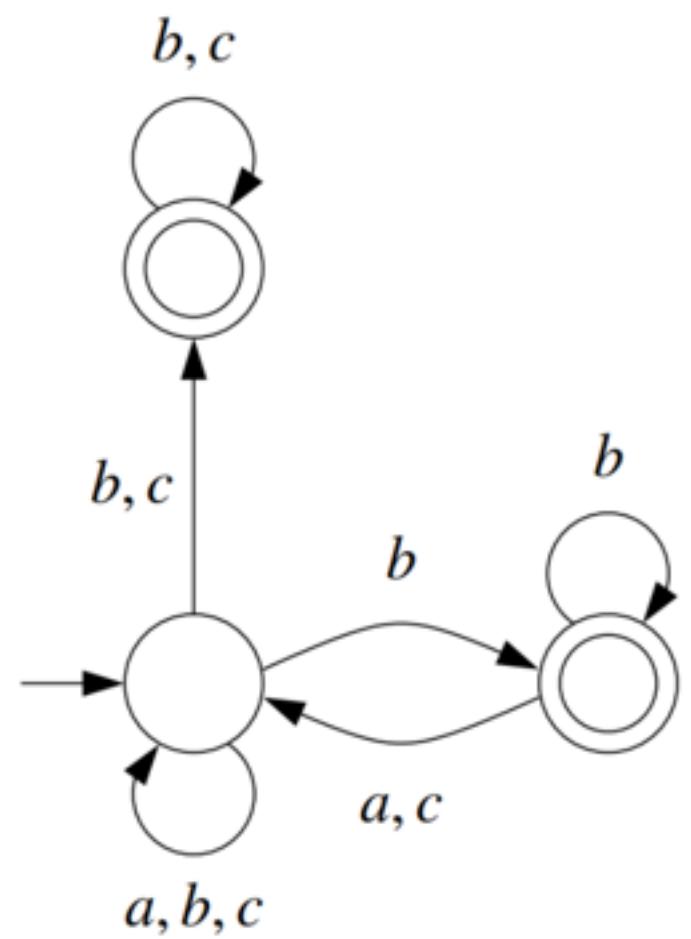
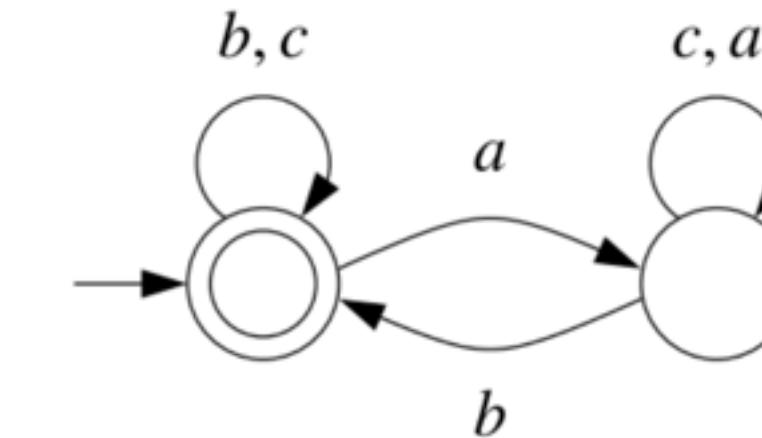
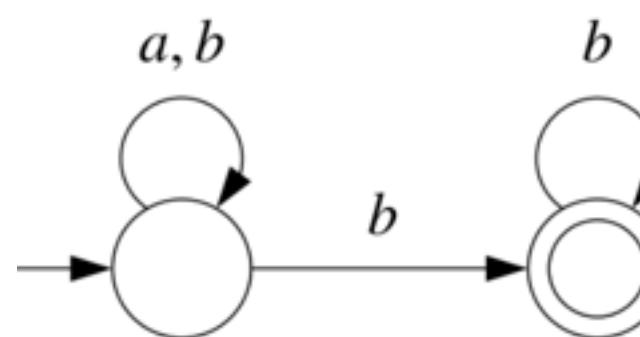
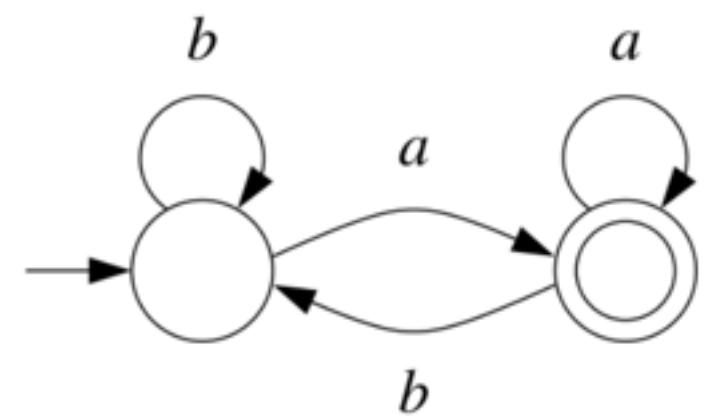
Can you enumerate the strings
in the language of this
automata?



- Büchi automata (BA) differs from traditional automata in its accepting condition.
It accepts strings of *infinite length* rather than finite length.

Exercise 3

- Describe in natural language the following automata



Büchi Automata, Formally

- A deterministic Büchi automaton is a tuple $A = (Q, \Sigma, \delta, q_0, F)$ that consists of the following components:
 - Q is the finite set of states of A
 - Σ is a finite set called the alphabet of A ,
 - $\delta: Q \times \Sigma \rightarrow Q$ is the transition function of A ,
 - $q_0 \in Q$ is the initial state of A .
 - $F \subseteq Q$ is the acceptance condition. A accepts exactly those strings in which at least one of the infinitely often occurring states is in F .

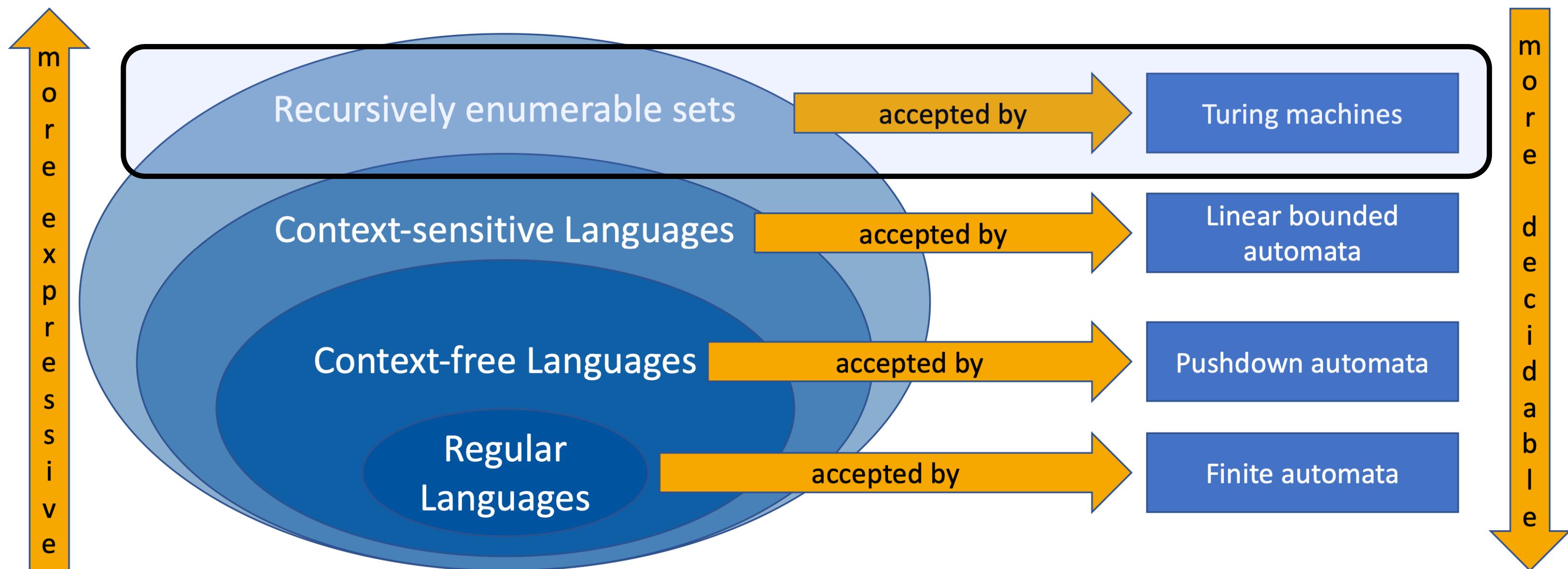
Compared to DFAs, Büchi automata recognizes ω -regular languages, that is, the generalization of regular languages to infinite words

Why do we care?

- **Safety** properties: Something bad does not happen
- **Liveness** properties: Something good eventually happens
- Regular languages commonly used to describe safety properties
 - **Exercise:** Consider a traffic light with three possible colors: red, yellow and green. The property “a red phase must be preceded immediately by a yellow phase”. Build an automaton that complies with the property
- Omega-regular languages used to describe liveness properties
 - **Exercise:** Consider a mutual exclusion algorithm (e.g. semaphores). Build a Buchi automaton that ensures that a process P visits its critical section infinitely often (you only need the alphabet {wait, crit})

Expressiveness

- The (absolute) **expressive power** describes the wealth of expressions that a model is capable of expressing
- Much work based on Chomski's Language Hierarchy
- Relative expressive power describe how one language can be encoded in another without losing its meaning (operational correspondence)



We won't cover the last point in the course but you can read more [here](#)

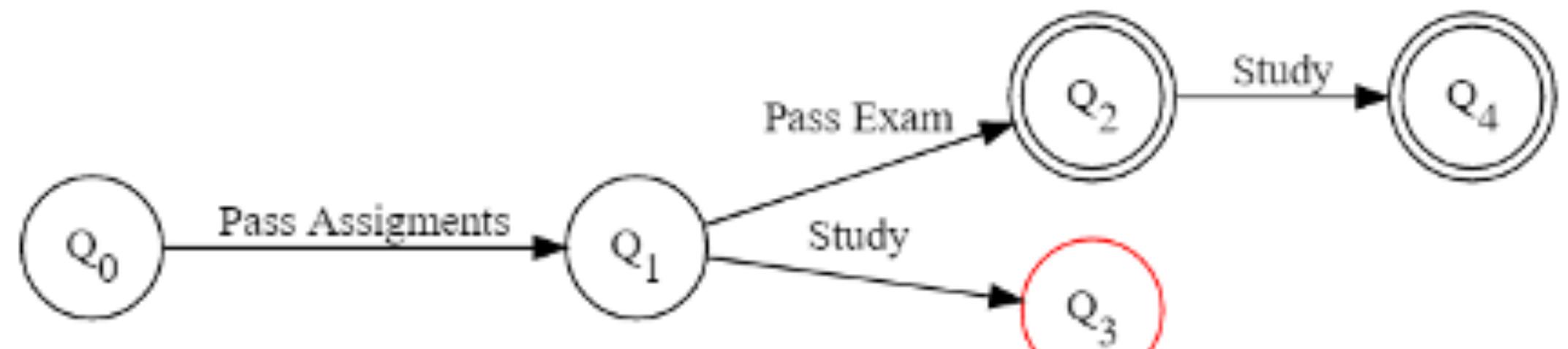
How good is your model?

- Two answers:
 - Is your model mapping the requirements? Adequacy
 - Does your model behave properly? Soundness
- Adequacy of the model depends on the stakeholder
 - Trace replay: given a word w and a DFA A , we say that w is played by A if $w \in L(A)$
- Deadlock and Livelock: Can you even execute this model?

Deadlocks and Livelocks

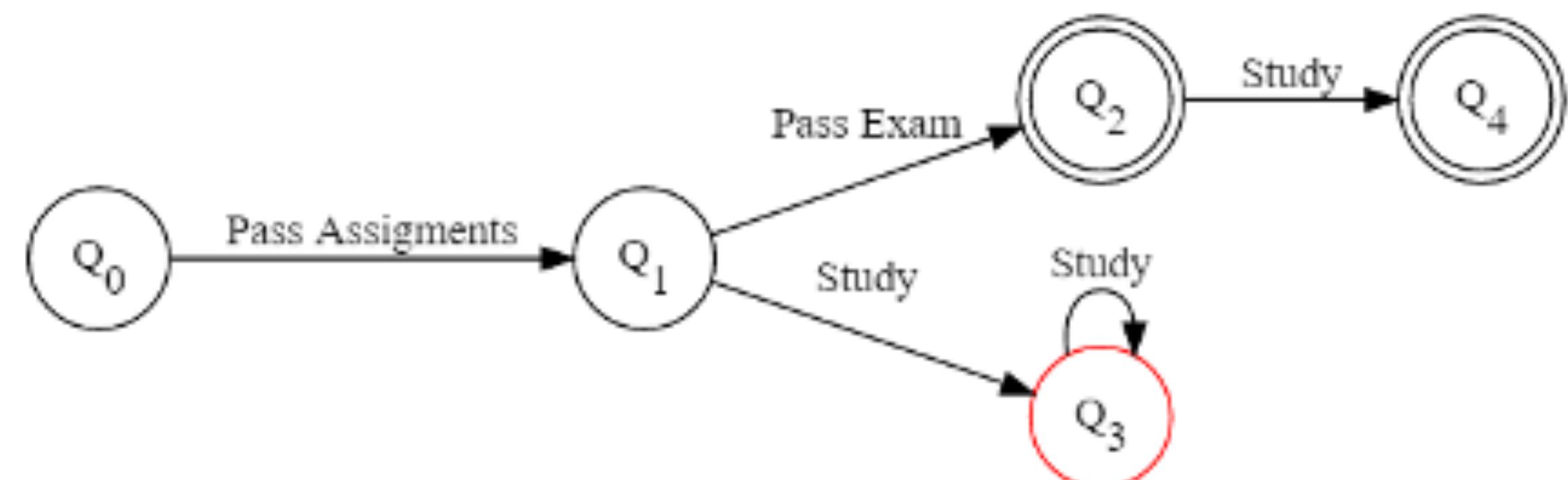
- **Deadlock:**

- A state in a system or process from which we can do no further actions, but which is not accepting.



- **Livelock:**

- A state or set of states in a system or process, in which we can still do actions, but from which no accepting run is possible.



To Summarize

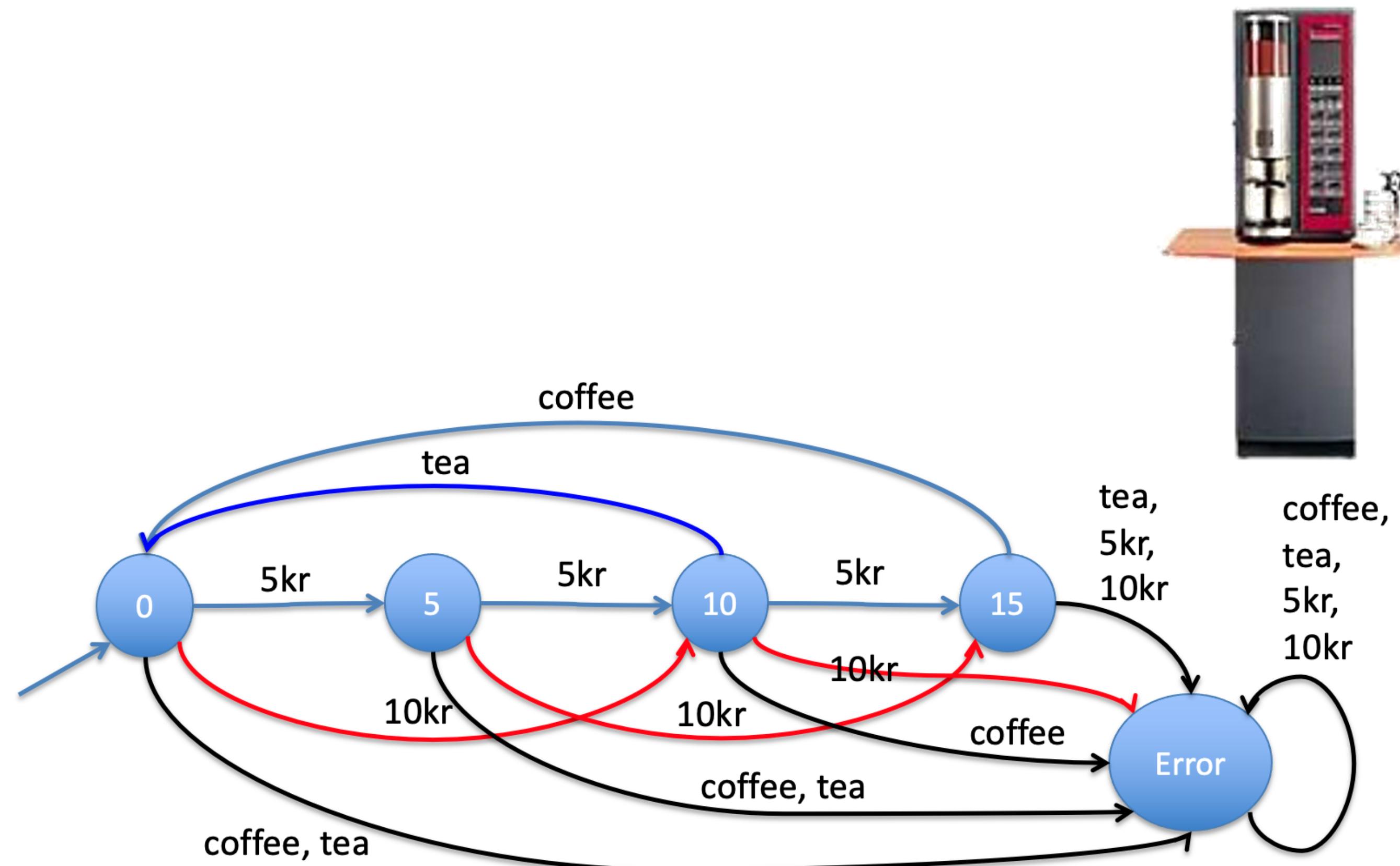
- Models as an abstraction mechanism used for communication, discovery or generation of behavior of a system
- Might have different granularity, or even different domain
- To use models for execution we need a semantic foundation
 - DFAs and NFAs map the behavior of finite systems
 - Buchi automata map the behavior of infinite (reactive) systems
- Correctness includes domain-specific properties (e.g. a trace that requires replay) or soundness guarantees
- We will use these elements when requiring our languages to have an *operational semantics* in the next few classes

Reading Material

- Hopcroft & Motwani & Ullman: Introduction to Automata Theory, Languages, and Computation. Third edition. Addison-Wesley, 2007.

Exercise 4

- In the programming language of your choice, implement the following vending machine



Exercise 5

- Extend the vending machine:
 - To offer chocolate for 20kr.
 - To accept coins of 20kr.

Exercise 6 2x2 puzzle

- Model the 2×2 puzzle, the simplest form of the $N \times N$ puzzle (https://en.wikipedia.org/wiki/15_puzzle) with a DFA. Which sequences of transitions help you solve the puzzle?

02291 System Integration

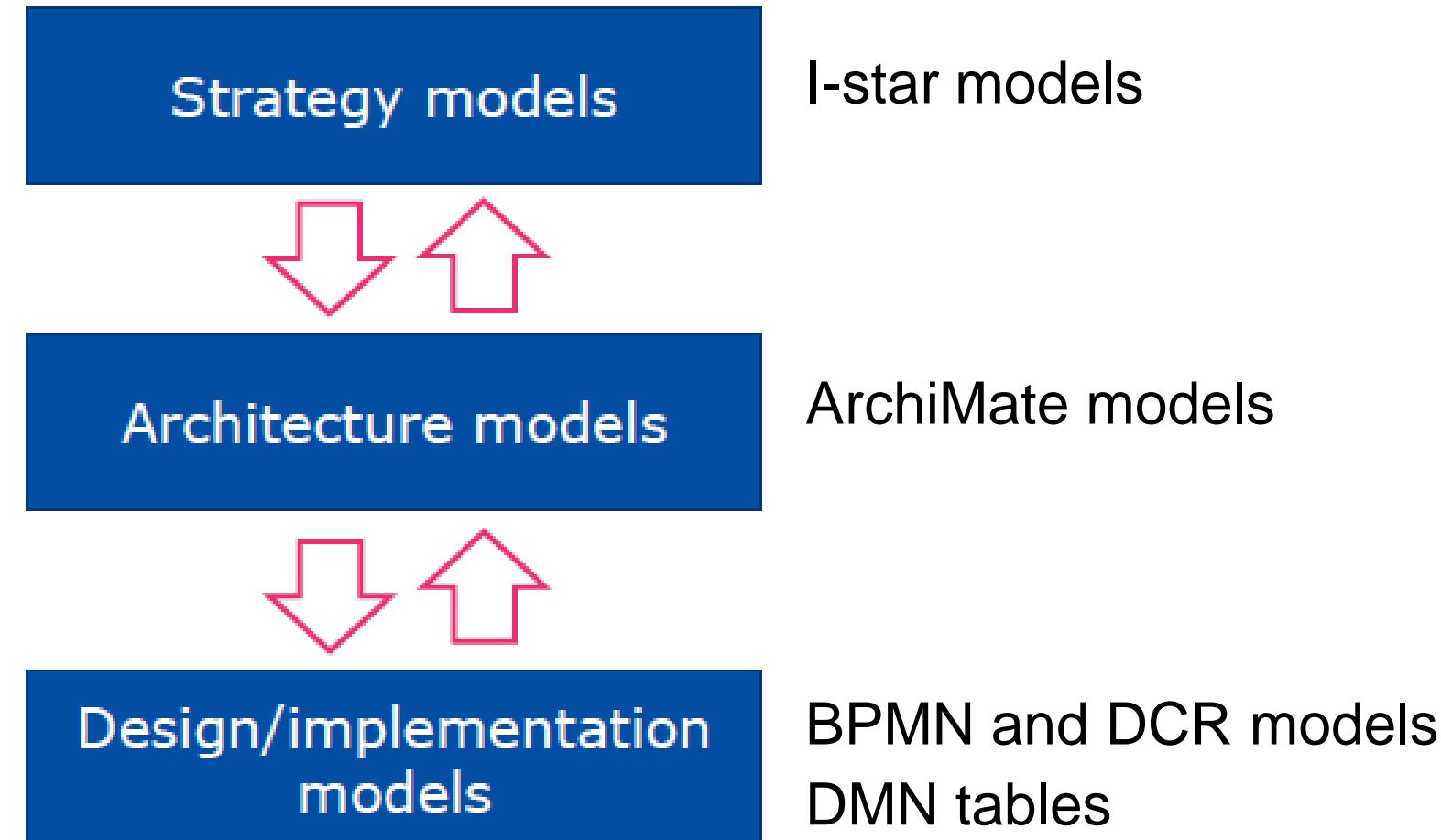
Introduction to ArchiMate

© Giovanni Meroni

What it is

- Archimate is a language for enterprise architecture modeling
- Mainly used to describe building blocks
- It is extensible

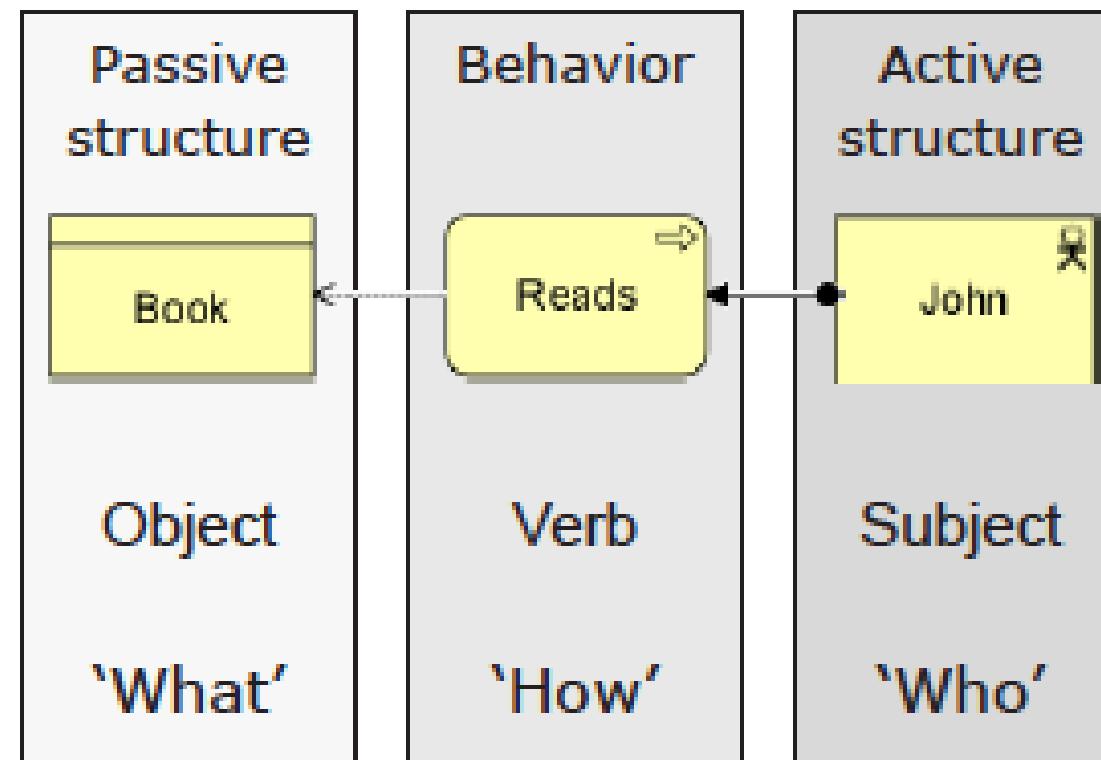
Positioning ArchiMate



Layers

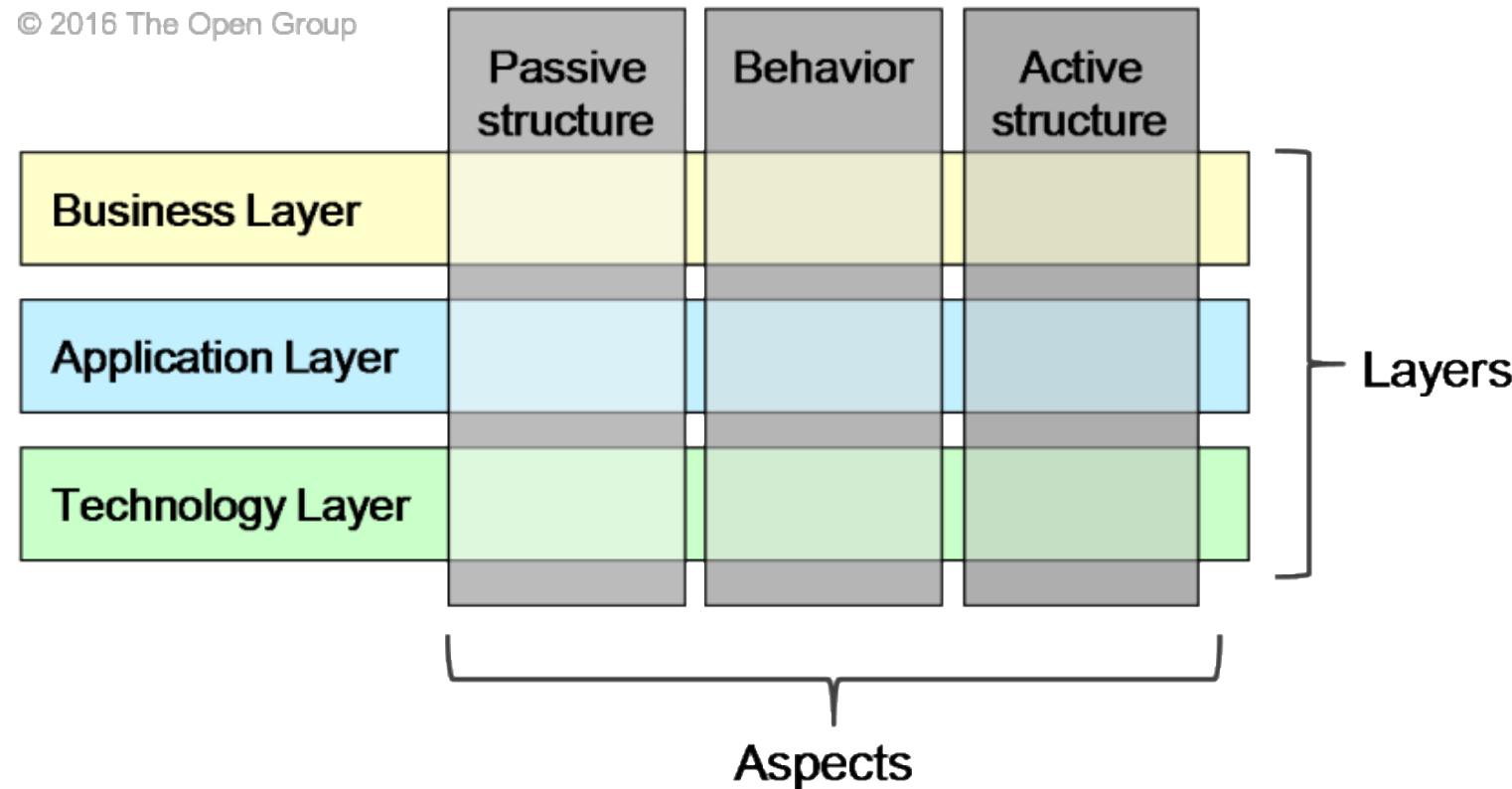
- Three main layers of modeling
 - Business layer: offer products and services to external customers
 - Application layer: support the business layer with application services – realized by software applications
 - Technology layer: offers infrastructure services needed to run applications, realized by computer and communication

Aspects



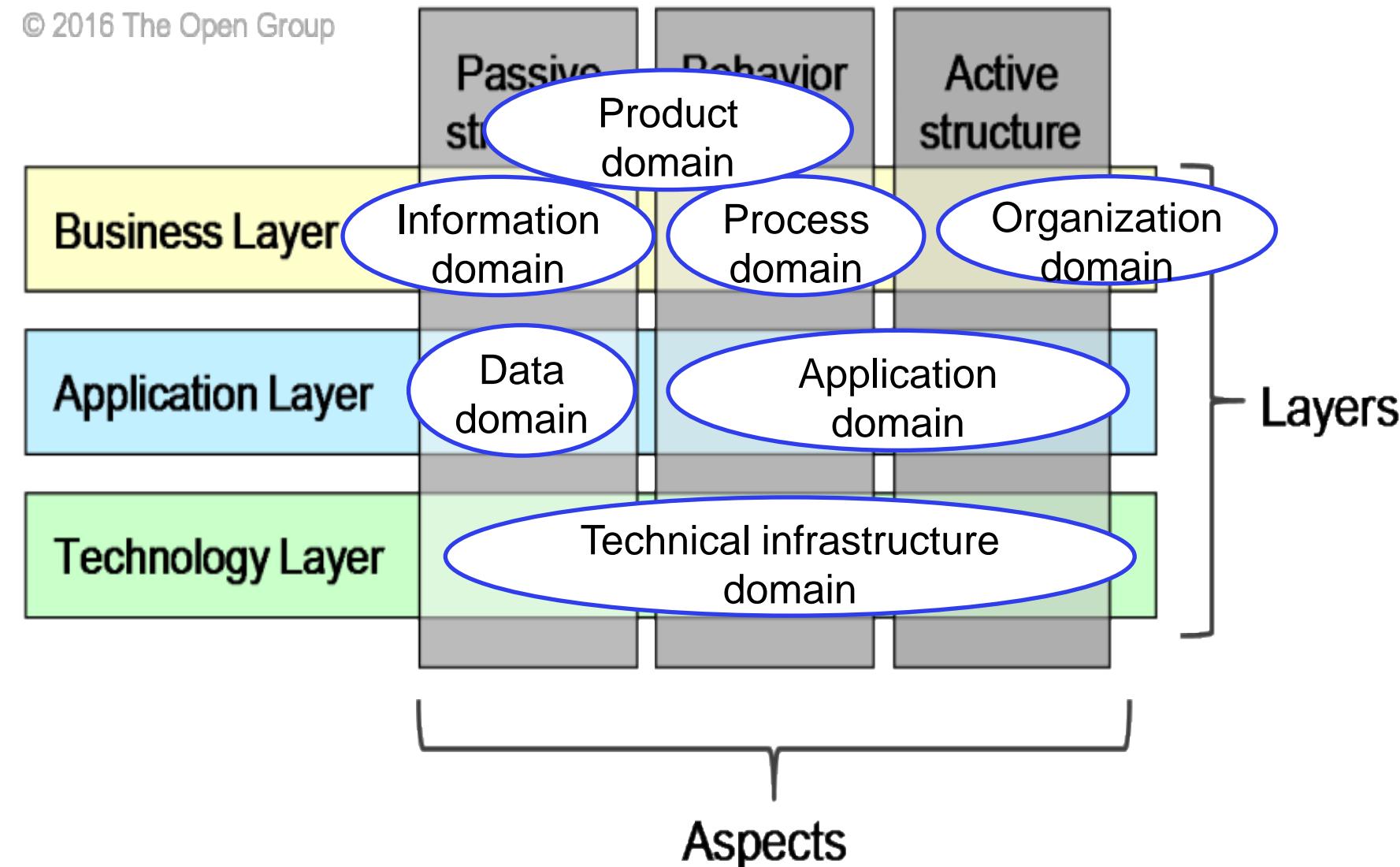
Aspects-layers matrix

© 2016 The Open Group



Aspects-layers matrix

© 2016 The Open Group



	DATA What	FUNCTION How	NETWORK Where	PEOPLE Who	TIME When	MOTIVATION Why	
SCOPE (CONTEXTUAL)	List of Things Important to the Business 	List of Processes the Business Performs 	List of Locations in which the Business Operates 	List of Organizations Important to the Business 	List of Events/Cycles Significant to the Business 	List of Business Goals/Strategies 	SCOPE (CONTEXTUAL)
Planner	ENTITY = Class of Business Thing	Process = Class of Business Process	Node = Major Business Location	People = Major Organization Unit	Time = Major Business Event/Cycle	Ends/Means = Major Business Goal/Strategy	Planner
BUSINESS MODEL (CONCEPTUAL)	e.g. Semantic Model 	e.g. Business Process Model 	e.g. Business Logistics System 	e.g. Work Flow Model 	e.g. Master Schedule 	e.g. Business Plan 	BUSINESS MODEL (CONCEPTUAL)
Owner	Ent = Business Entity Rein = Business Relationship	Proc. = Business Process I/O = Business Resources	Node = Business Location Link = Business Linkage	People = Organization Unit Work = Work Product	Time = Business Event Cycle = Business Cycle	End = Business Objective Means = Business Strategy	Owner
SYSTEM MODEL (LOGICAL)	e.g. Logical Data Model 	e.g. Application Architecture 	e.g. Distributed System Architecture 	e.g. Human Interface Architecture 	e.g. Processing Structure 	e.g. Business Rule Model 	SYSTEM MODEL (LOGICAL)
Designer	Ent = Data Entity Rein = Data Relationship	Proc. = Application Function I/O = User Views	Node = IS Function (Processor, Storage, etc) Link = Line Characteristics	People = Role Work = Deliverable	Time = System Event Cycle = Processing Cycle	End = Structural Assertion Means = Action Assertion	Designer
TECHNOLOGY MODEL (PHYSICAL)	e.g. Physical Data Model 	e.g. System Design 	e.g. Technology Architecture 	e.g. Presentation Architecture 	e.g. Control Structure 	e.g. Rule Design 	TECHNOLOGY MODEL (PHYSICAL)
Builder	Ent = Segment/Table/etc. Rein = Pointer/Key/etc.	Proc. = Computer Function I/O = Data Elements/Sets	Node = Hardware/Systems Software Link = Line Specifications	People = User Work = Screen Format	Time = Execute Cycle = Component Cycle	End = Condition Means = Action	Builder
DETAILED REPRESENTATIONS (OUT-OF-CONTEXT)	e.g. Data Definition 	e.g. Program 	e.g. Network Architecture 	e.g. Security Architecture 	e.g. Timing Definition 	e.g. Rule Specification 	DETAILED REPRESENTATIONS (OUT-OF-CONTEXT)
Sub-Contractor	Ent = Field Rein = Address	Proc. = Language Statement I/O = Control Block	Node = Address Link = Protocol	People = Identity Work = Job	Time = Interrupt Cycle = Machine Cycle	End = Sub-condition Means = Stop	Sub-Contractor
FUNCTIONING ENTERPRISE	e.g. DATA	e.g. FUNCTION	e.g. NETWORK	e.g. ORGANIZATION	e.g. SCHEDULE	e.g. STRATEGY	FUNCTIONING ENTERPRISE

© John A. Zachman, Zachman International

	DATA What	FUNCTION How	NETWORK Where	PEOPLE Who	TIME When	MOTIVATION Why	
SCOPE (CONTEXTUAL)					List of Events/Cycles Significant to the Business 		SCOPE (CONTEXTUAL)
Planner					Time = Major Business Event/Cycle		Planner
BUSINESS MODEL (CONCEPTUAL)	e.g. Semantic Model 	e.g. Business Process Model 	e.g. Business Logistics System 	e.g. Work Flow Model 	e.g. Master Schedule 		BUSINESS MODEL (CONCEPTUAL)
Owner	Ent = Business Entity Rein = Business Relationship	Proc. = Business Process I/O = Business Resources	Node = Business Location Link = Business Linkage	People = Organization Unit Work = Work Product	Time = Business Event Cycle = Business Cycle		Owner
SYSTEM MODEL (LOGICAL)	e.g. Logical Data Model 	e.g. Application Architecture 	e.g. Distributed System Architecture 	e.g. Human Interface Architecture 	e.g. Processing Structure 		SYSTEM MODEL (LOGICAL)
Designer	Ent = Data Entity Rein = Data Relationship	Proc. = Application Function I/O = User Views	Node = IS Function (Processor, Storage, etc) Link = Line Characteristics	People = Role Work = Deliverable	Time = System Event Cycle = Processing Cycle		Designer
TECHNOLOGY MODEL (PHYSICAL)	e.g. Physical Data Model 	e.g. System Design 	e.g. Technology Architecture 	e.g. Presentation Architecture 	e.g. Control Structure 		TECHNOLOGY MODEL (PHYSICAL)
Builder	Ent = Segment/Table/etc. Rein = Pointer/Key/etc.	Proc. = Computer Function I/O = Data Elements/Sets	Node = Hardware/Systems Software Link = Line Specifications	People = User Work = Screen Format	Time = Execute Cycle = Component Cycle		Builder
DETAILED REPRESENTATIONS (OUT-OF-CONTEXT)	e.g. Data Definition 	e.g. Program 	e.g. Network Architecture 	e.g. Security Architecture 	e.g. Timing Definition 		DETAILED REPRESENTATIONS (OUT-OF-CONTEXT)
Sub-Contractor	Ent = Field Rein = Address	Proc. = Language Statement I/O = Control Block	Node = Address Link = Protocol	People = Identity Work = Job	Time = Interrupt Cycle = Machine Cycle		Sub-Contractor
FUNCTIONING ENTERPRISE	e.g. DATA	e.g. FUNCTION	e.g. NETWORK	e.g. ORGANIZATION	e.g. SCHEDULE	e.g. STRATEGY	FUNCTIONING ENTERPRISE

Motivation

	DATA What	FUNCTION How	NETWORK Where	PEOPLE Who	TIME When	MOTIVATION Why	
SCOPE (CONTEXTUAL)					List of Events/Cycles Significant to the Business 		SCOPE (CONTEXTUAL)
Planner					Time = Major Business Event/Cycle e.g. Master Schedule 		Planner
BUSINESS MODEL (CONCEPTUAL)					Time = Business Event Cycle = Business Cycle 		BUSINESS MODEL CONCEPTUAL)
Owner							Owner
SYSTEM MODEL (LOGICAL)	e.g. Logical Data Model Ent = Data Entity Reln = Data Relationship	e.g. Application Architecture Proc = Application Function I/O = User Views	e.g. Distributed System Architecture Node = IS Function (Processor, Storage, etc) Link = Line Characteristics	e.g. Human Interface Architecture People = Role Work = Deliverable	e.g. Processing Structure Time = System Event Cycle = Processing Cycle		SYSTEM MODEL (LOGICAL)
Designer							Designer
TECHNOLOGY MODEL (PHYSICAL)	e.g. Physical Data Model 	e.g. System Design Proc = Computer Function I/O = Data Elements/Sets	e.g. Technology Architecture Node = Hardware/Systems Software Link = Line Specifications	e.g. Presentation Architecture People = User Work = Screen Format	e.g. Control Structure Time = Execute Cycle = Component Cycle		TECHNOLOGY MODEL (PHYSICAL)
Builder	Ent = Segment/Table/etc. Reln = Pointer/Key/etc.	Proc = Computer Function I/O = Data Elements/Sets	Node = Hardware/Systems Software Link = Line Specifications	People = User Work = Screen Format	Time = Execute Cycle = Component Cycle		Builder
DETAILED REPRESENTATIONS (OUT-OF-CONTEXT)	e.g. Data Definition Ent = Field Reln = Address	e.g. Program Proc = Language Statement I/O = Control Block	e.g. Network Architecture Node = Address Link = Protocol	e.g. Security Architecture People = Identity Work = Job	e.g. Timing Definition Time = Interrupt Cycle = Machine Cycle		DETAILED REPRESENTATIONS (OUT-OF-CONTEXT)
Sub-Contractor							Sub-Contractor
FUNCTIONING ENTERPRISE	e.g. DATA	e.g. FUNCTION	e.g. NETWORK	e.g. ORGANIZATION	e.g. SCHEDULE	e.g. STRATEGY	FUNCTIONING ENTERPRISE

Business

Motivation

© John A. Zachman, Zachman International

	DATA	What	FUNCTION	How	NETWORK	Where	PEOPLE	Who	TIME	When	MOTIVATION	Why	
SCOPE (CONTEXTUAL)									List of Events/Cycles Significant to the Business				SCOPE (CONTEXTUAL)
<i>Planner</i>													<i>Planner</i>
BUSINESS MODEL (CONCEPTUAL)									Time = Major Business Event/Cycle				BUSINESS MODEL (CONCEPTUAL)
<i>Owner</i>									e.g. Master Schedule				<i>Owner</i>
SYSTEM MODEL (LOGICAL)													SYSTEM MODEL (LOGICAL)
<i>Designer</i>									Time = Business Event Cycle = Business Cycle				<i>Designer</i>
TECHNOLOGY MODEL (PHYSICAL)	e.g. Physical Data Model 	e.g. System Design 	e.g. Technology Architecture 	e.g. Presentation Architecture 	e.g. Control Structure 				e.g. Processing Structure 				TECHNOLOGY MODEL (PHYSICAL)
<i>Builder</i>	Ent = Segment/Table/etc. Rein = Pointer/Key/etc.	Proc = Computer Function I/O = Data Elements/Sets	Node = Hardware/Systems Software Link = Line Specifications	People = User Work = Screen Format	Time = Execute Cycle = Component Cycle								<i>Builder</i>
DETAILED REPRESEN- TATIONS (OUT-OF- CONTEXT)	e.g. Data Definition 	e.g. Program 	e.g. Network Architecture 	e.g. Security Architecture 	e.g. Timing Definition 								DETAILED REPRESEN- TATIONS (OUT-OF- CONTEXT)
<i>Sub- Contractor</i>	Ent = Field Rein = Address	Proc = Language Statement I/O = Control Block	Node = Address Link = Protocol	People = Identity Work = Job	Time = Interrupt Cycle = Machine Cycle								<i>Sub- Contractor</i>
FUNCTIONING ENTERPRISE	e.g. DATA	e.g. FUNCTION	e.g. NETWORK	e.g. ORGANIZATION	e.g. SCHEDULE	e.g. STRATEGY							FUNCTIONING ENTERPRISE

Business

Application

Motivation

© John A. Zachman, Zachman International

	DATA	What	FUNCTION	How	NETWORK	Where	PEOPLE	Who	TIME	When	MOTIVATION	Why	
SCOPE (CONTEXTUAL)									List of Events/Cycles Significant to the Business 				SCOPE (CONTEXTUAL)
<i>Planner</i>									Time = Major Business Event/Cycle e.g. Master Schedule 			<i>Planner</i>	
BUSINESS MODEL (CONCEPTUAL)									Time = Business Event Cycle = Business Cycle e.g. Processing Structure 			BUSINESS MODEL (CONCEPTUAL)	
<i>Owner</i>									Time = System Event Cycle = Processing Cycle e.g. Control Structure 			<i>Owner</i>	
SYSTEM MODEL (LOGICAL)									Time = Execute Cycle = Component Cycle e.g. Timing Definition 			SYSTEM MODEL (LOGICAL)	
<i>Designer</i>												<i>Designer</i>	
TECHNOLOGY MODEL (PHYSICAL)												TECHNOLOGY MODEL (PHYSICAL)	
<i>Builder</i>												<i>Builder</i>	
DETAILED REPRESEN- TATIONS (OUT-OF- CONTEXT)	e.g. Data Definition 	e.g. Program 	e.g. Network Architecture 	e.g. Security Architecture 	e.g. Timing Definition 							DETAILED REPRESEN- TATIONS (OUT-OF- CONTEXT)	
<i>Sub- Contractor</i>	Ent = Field Reln = Address	Proc = Language Statement I/O = Control Block	Node = Address Link = Protocol	People = Identity Work = Job	Time = Interrupt Cycle = Machine Cycle							<i>Sub- Contractor</i>	
FUNCTIONING ENTERPRISE	e.g. DATA	e.g. FUNCTION	e.g. NETWORK	e.g. ORGANIZATION	e.g. SCHEDULE	e.g. STRATEGY						FUNCTIONING ENTERPRISE	

Business**Application****Technology****Motivation**

© John A. Zachman, Zachman International

	DATA What	FUNCTION How	NETWORK Where	PEOPLE Who	TIME When	MOTIVATION Why	
SCOPE (CONTEXTUAL)	Passive				List of Events/Cycles Significant to the Business 		SCOPE (CONTEXTUAL)
Planner					Time = Major Business Event/Cycle e.g. Master Schedule 		Planner
BUSINESS MODEL (CONCEPTUAL)					Time = Business Event Cycle = Business Cycle e.g. Processing Structure 		BUSINESS MODEL CONCEPTUAL)
Owner					Time = System Event Cycle = Processing Cycle e.g. Control Structure 		Owner
SYSTEM MODEL (LOGICAL)					Time = Execute Cycle = Component Cycle e.g. Timing Definition 		SYSTEM MODEL (LOGICAL)
Designer							Designer
TECHNOLOGY MODEL (PHYSICAL)							TECHNOLOGY MODEL (PHYSICAL)
Builder							Builder
DETAILED REPRESENTATIONS (OUT-OF-CONTEXT)	e.g. Data Definition 	e.g. Program 	e.g. Network Architecture 	e.g. Security Architecture 	e.g. Timing Definition 		DETAILED REPRESENTATIONS (OUT-OF-CONTEXT)
Sub-Contractor	Ent = Field Reln = Address	Proc = Language Statement I/O = Control Block	Node = Address Link = Protocol	People = Identity Work = Job	Time = Interrupt Cycle = Machine Cycle		Sub-Contractor
FUNCTIONING ENTERPRISE	e.g. DATA	e.g. FUNCTION	e.g. NETWORK	e.g. ORGANIZATION	e.g. SCHEDULE	e.g. STRATEGY	FUNCTIONING ENTERPRISE

© John A. Zachman, Zachman International

	DATA What	FUNCTION How	NETWORK Where	PEOPLE Who	TIME When	MOTIVATION Why	
SCOPE (CONTEXTUAL)	Passive	Behavior			List of Events/Cycles Significant to the Business 		SCOPE (CONTEXTUAL)
Planner					Time = Major Business Event/Cycle e.g. Master Schedule 		Planner
BUSINESS MODEL (CONCEPTUAL)					Time = Business Event Cycle = Business Cycle e.g. Processing Structure 		BUSINESS MODEL (CONCEPTUAL)
Owner					Time = System Event Cycle = Processing Cycle e.g. Control Structure 		Owner
SYSTEM MODEL (LOGICAL)					Time = Execute Cycle = Component Cycle e.g. Timing Definition 		SYSTEM MODEL (LOGICAL)
Designer							Designer
TECHNOLOGY MODEL (PHYSICAL)							TECHNOLOGY MODEL (PHYSICAL)
Builder							Builder
DETAILED REPRESENTATIONS (OUT-OF-CONTEXT)	e.g. Data Definition 	e.g. Program 	e.g. Network Architecture 	e.g. Security Architecture 	e.g. Timing Definition 		DETAILED REPRESENTATIONS (OUT-OF-CONTEXT)
Sub-Contractor	Ent = Field Reln = Address	Proc = Language Statement I/O = Control Block	Node = Address Link = Protocol	People = Identity Work = Job	Time = Interrupt Cycle = Machine Cycle		Sub-Contractor
FUNCTIONING ENTERPRISE	e.g. DATA	e.g. FUNCTION	e.g. NETWORK	e.g. ORGANIZATION	e.g. SCHEDULE	e.g. STRATEGY	FUNCTIONING ENTERPRISE

Business

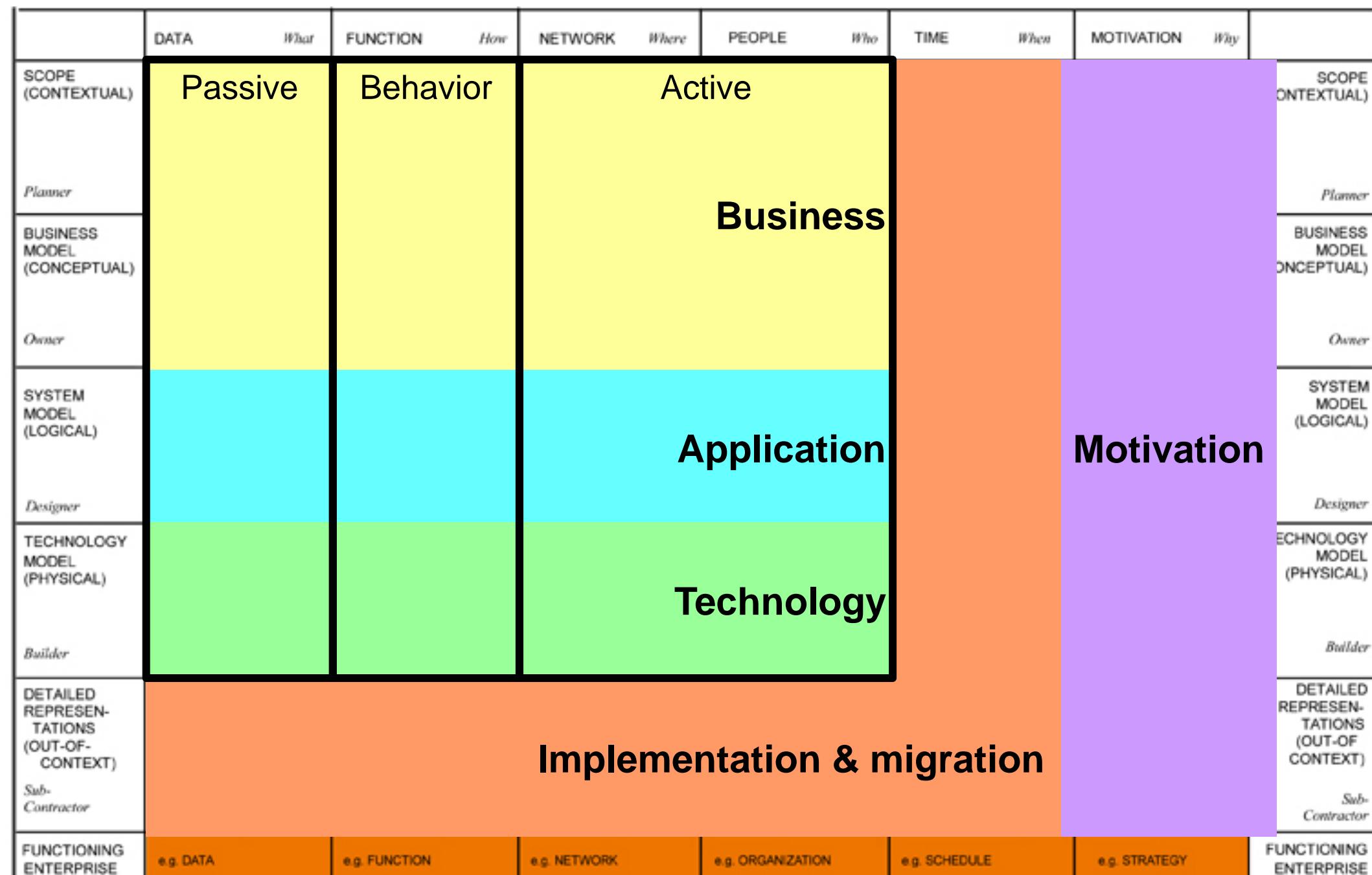
Application

Technology

Motivation

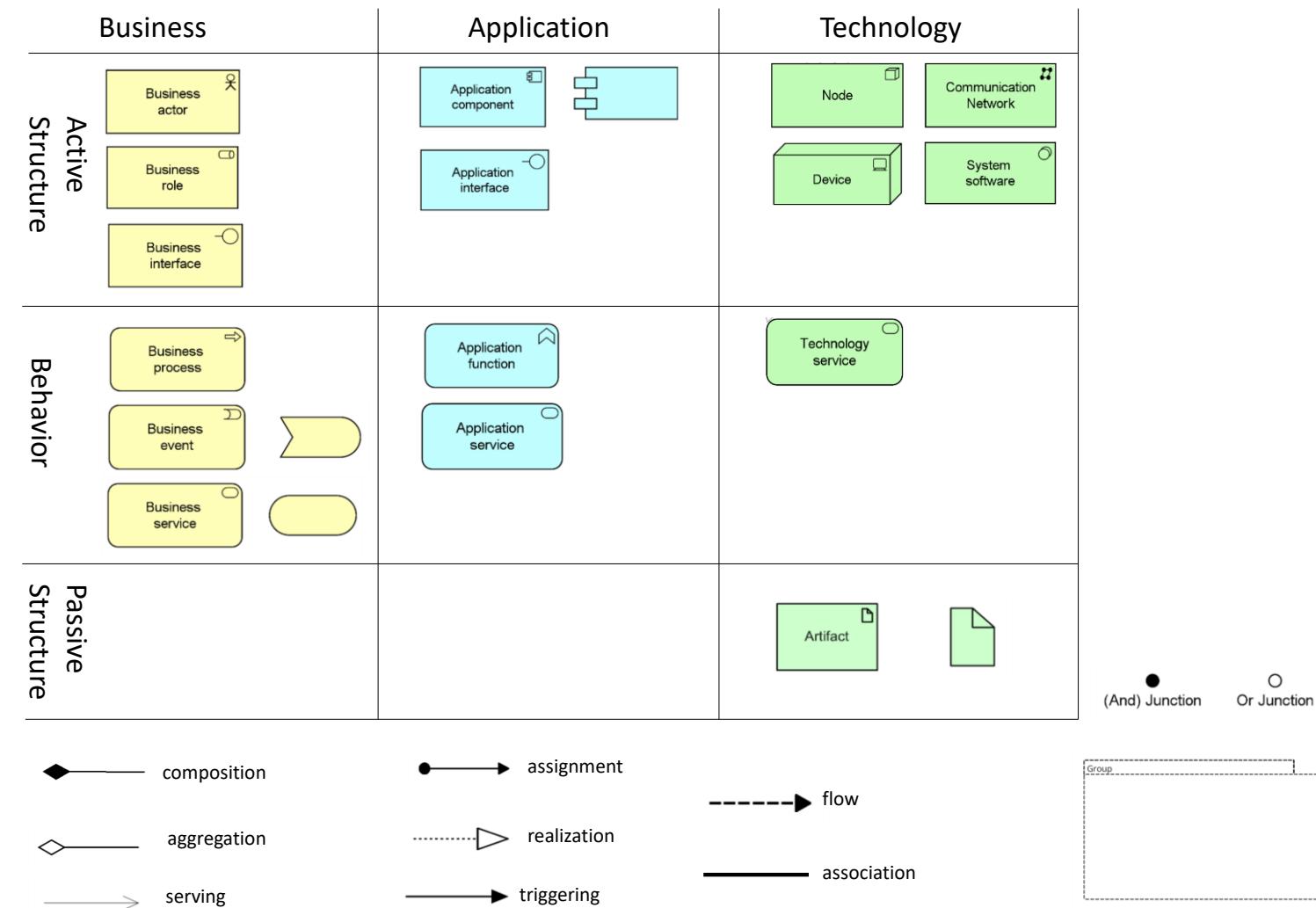
	DATA What	FUNCTION How	NETWORK Where	PEOPLE Who	TIME When	MOTIVATION Why	
SCOPE (CONTEXTUAL)	Passive	Behavior		Active			SCOPE (CONTEXTUAL)
<i>Planner</i>							<i>Planner</i>
BUSINESS MODEL (CONCEPTUAL)							BUSINESS MODEL (CONCEPTUAL)
<i>Owner</i>							<i>Owner</i>
SYSTEM MODEL (LOGICAL)							SYSTEM MODEL (LOGICAL)
<i>Designer</i>							<i>Designer</i>
TECHNOLOGY MODEL (PHYSICAL)							TECHNOLOGY MODEL (PHYSICAL)
<i>Builder</i>							<i>Builder</i>
DETAILED REPRESEN- TATIONS (OUT-OF- CONTEXT)	e.g. Data Definition 	e.g. Program 	e.g. Network Architecture 	e.g. Security Architecture 	e.g. Timing Definition 		DETAILED REPRESEN- TATIONS (OUT-OF- CONTEXT)
<i>Sub- Contractor</i>	Ent = Field Reln = Address	Proc = Language Statement I/O = Control Block	Node = Address Link = Protocol	People = Identity Work = Job	Time = Interrupt Cycle = Machine Cycle		<i>Sub- Contractor</i>
FUNCTIONING ENTERPRISE	e.g. DATA	e.g. FUNCTION	e.g. NETWORK	e.g. ORGANIZATION	e.g. SCHEDULE	e.g. STRATEGY	FUNCTIONING ENTERPRISE

© John A. Zachman, Zachman International



© John A. Zachman, Zachman International

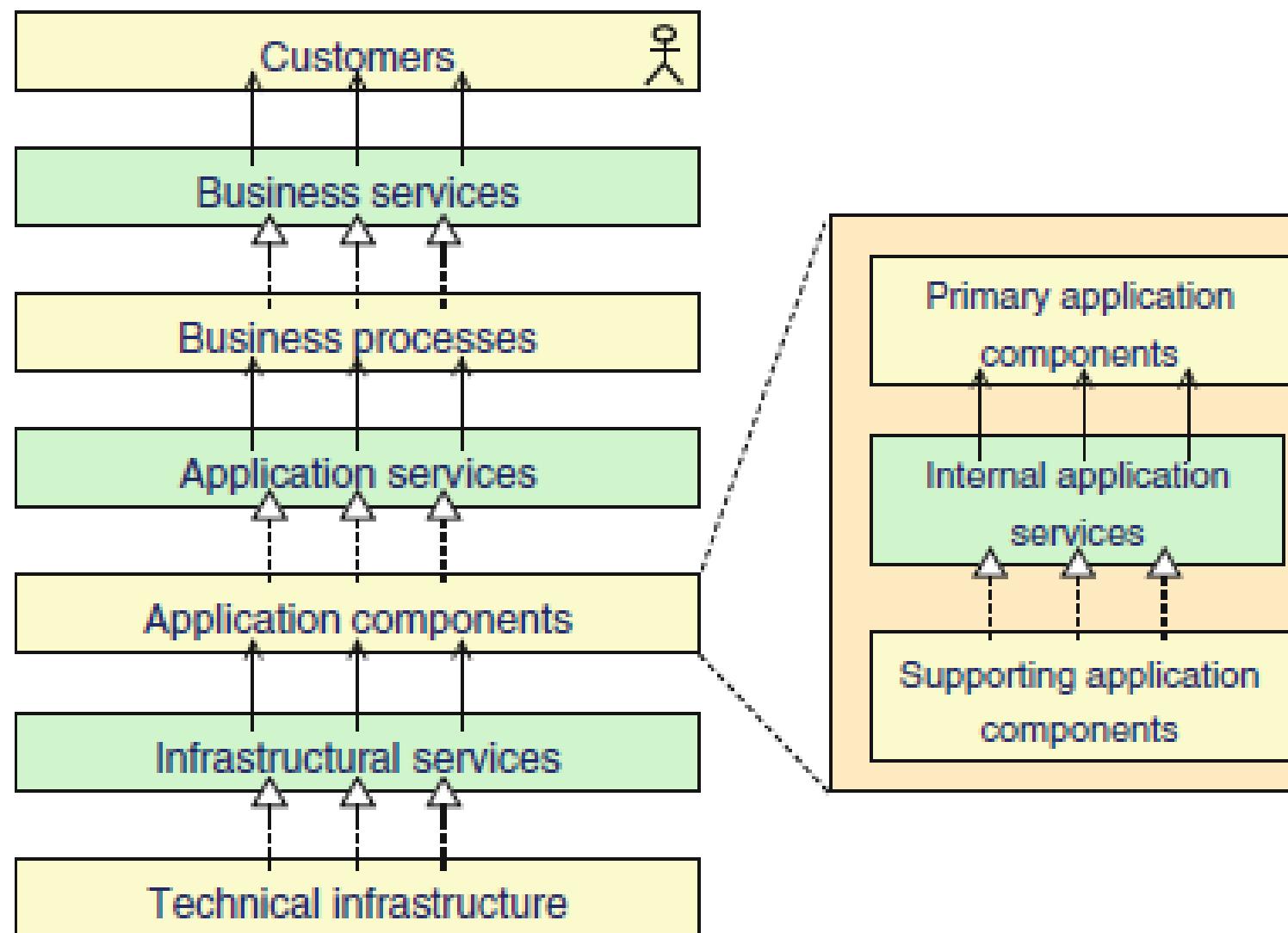
ArchiMate cheat sheet



Service Oriented approach

- Components (business, application, software, infrastructure) provide services to other components
- Services at all levels
- Examples:
 - Web services
 - In cloud computing: Software (SaaS), Platform (PaaS), Infrastructure (IaaS)
- Services are central to Archimate's architecture models

Service Oriented approach



Lankhorst et al
p. 78

Business Layer

Active elements

Business actor 

A business entity that is capable of performing behavior

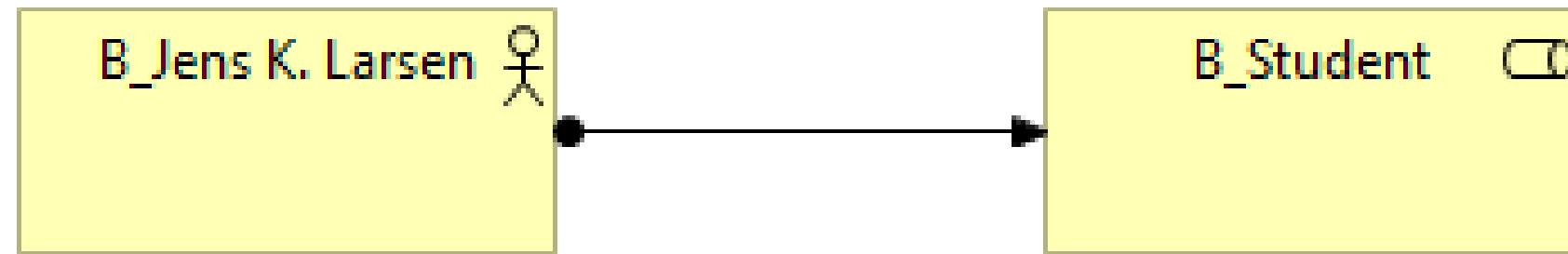
Business role 

The responsibility for performing specific behavior, to which an action can be assigned

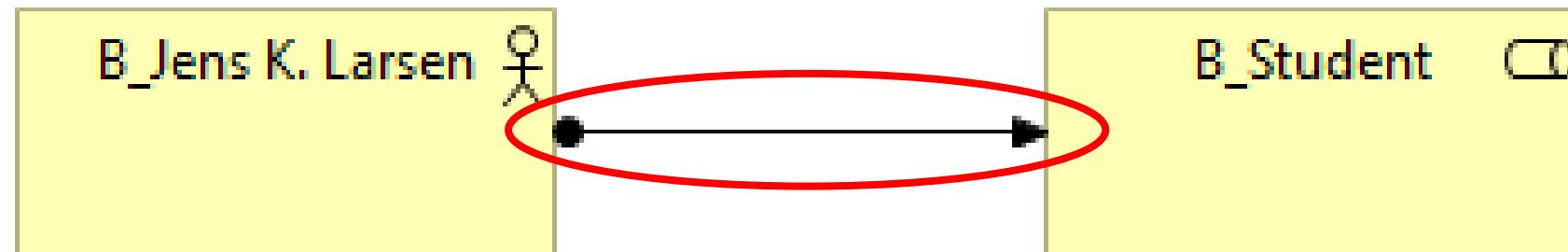
Business interface 

A point of access where a business service is made available to the environment

Active elements: example

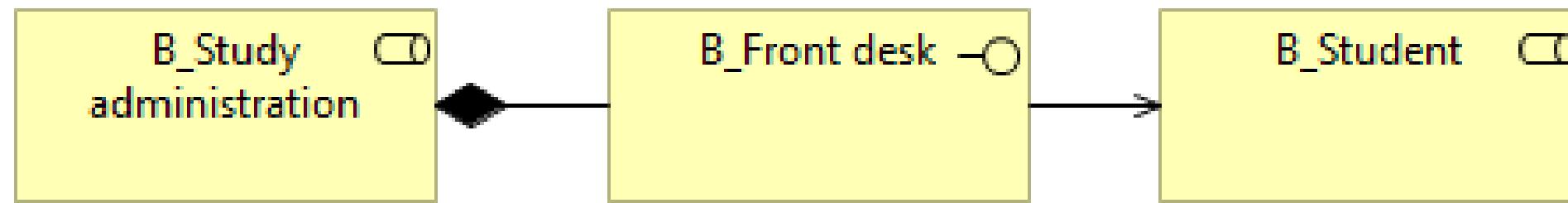


Active elements: example

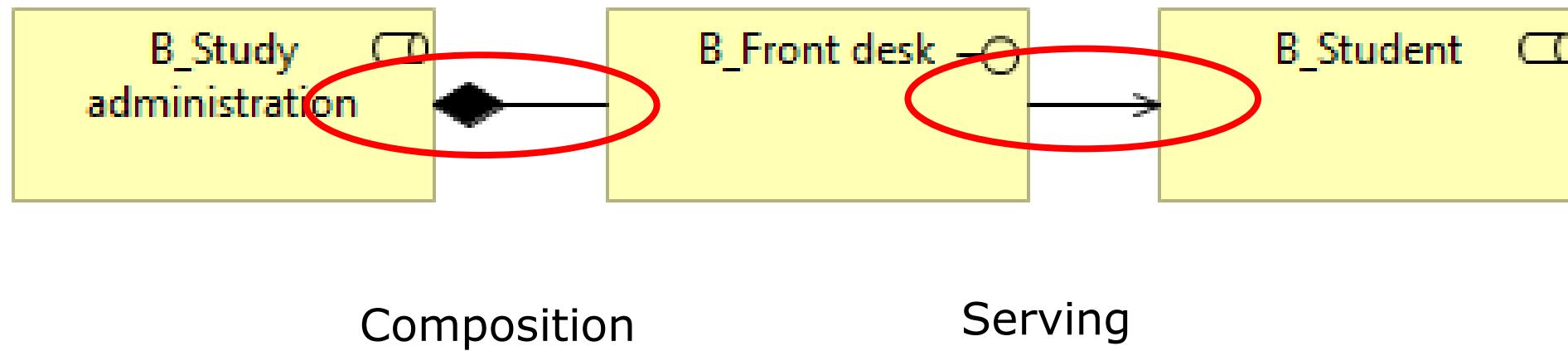


Assignment

Active elements: example



Active elements: example



Composition

Serving

Behavior elements

Business process



A sequence of business behaviors that achieves a specific outcome

Business function



A collection of business behaviors based on a chosen set of criteria, aligned to an organization

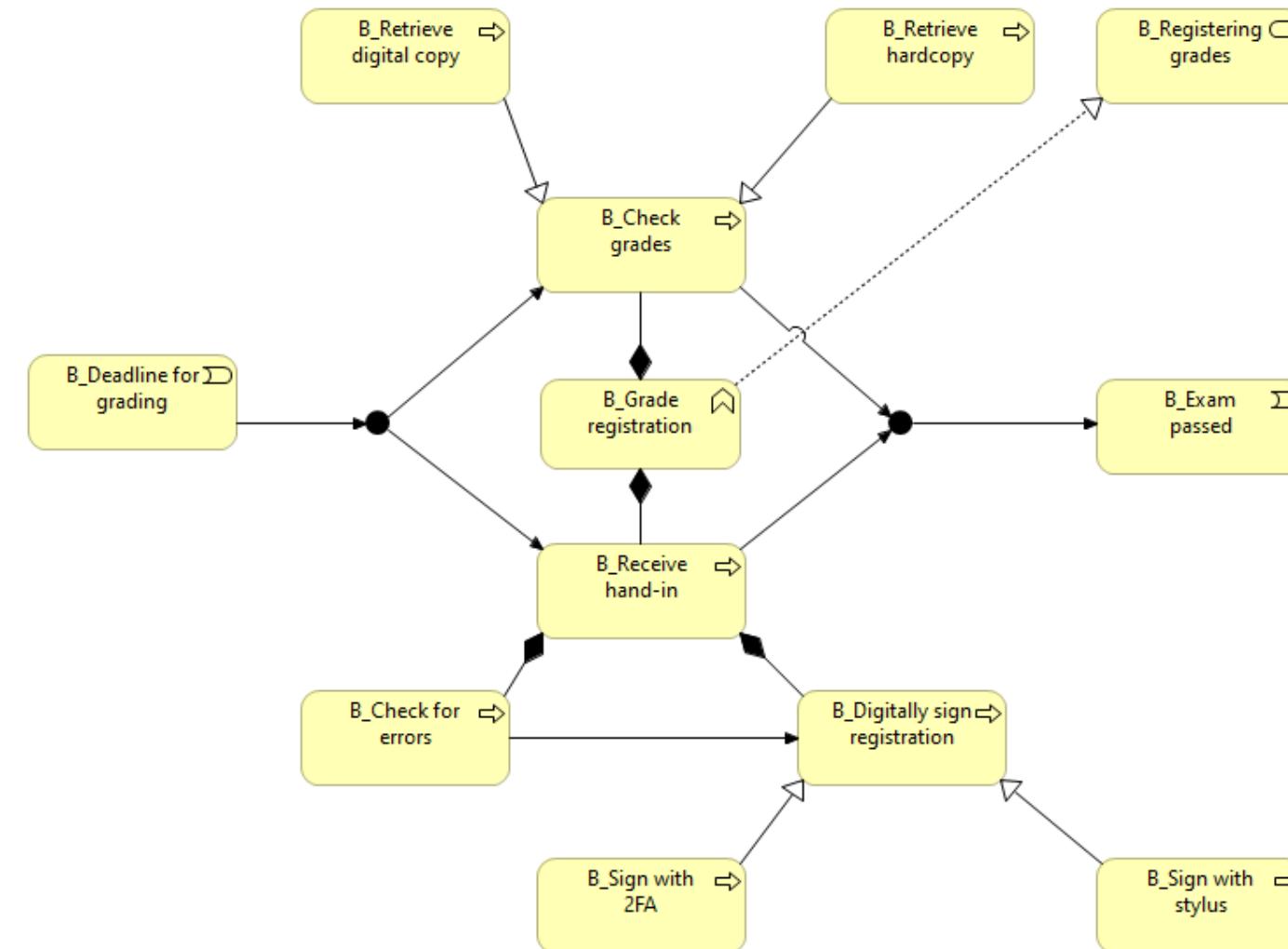
Business service

An explicitly defined exposed business behavior

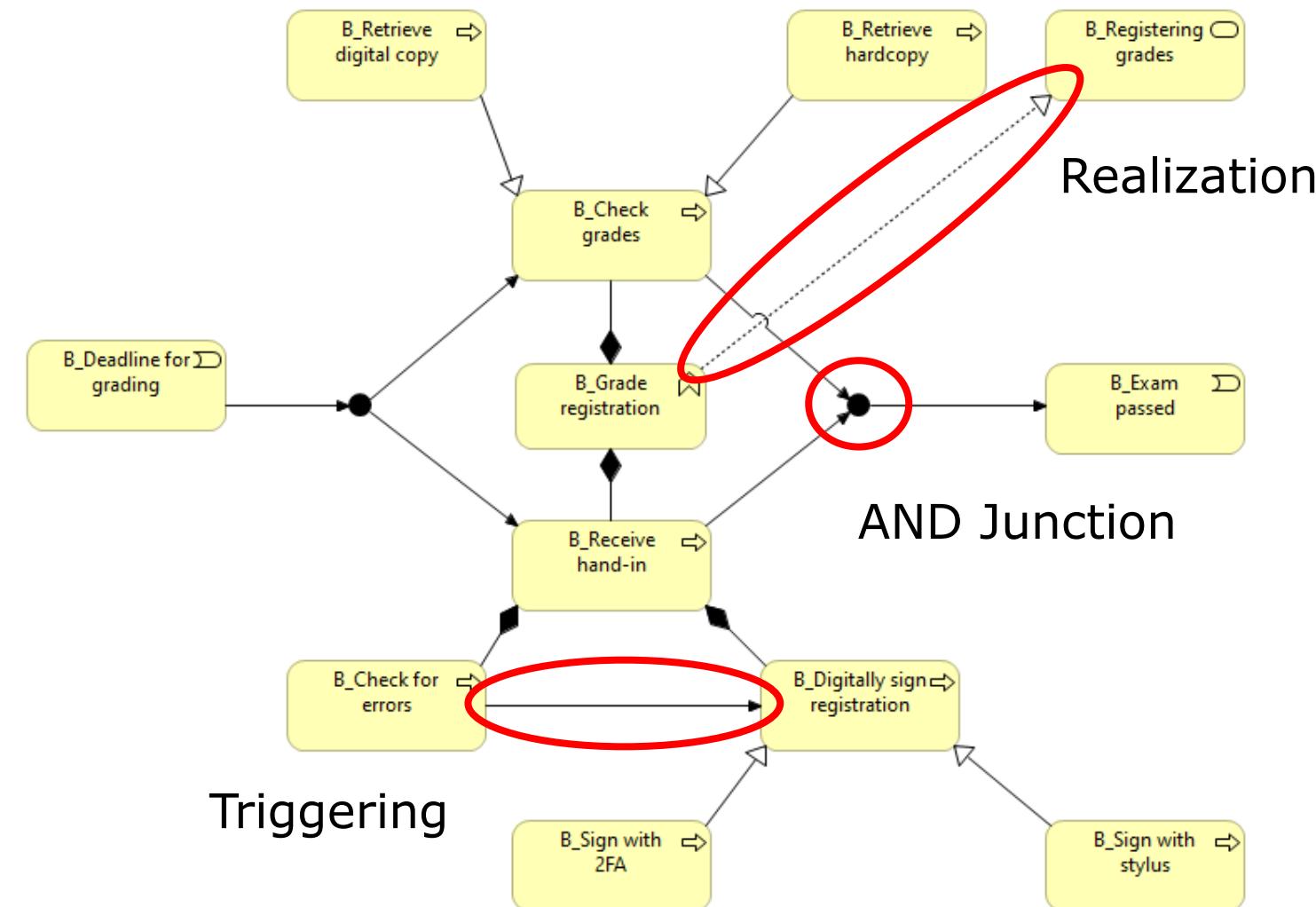
Business event

A business behavior element that denotes an organizational state change.

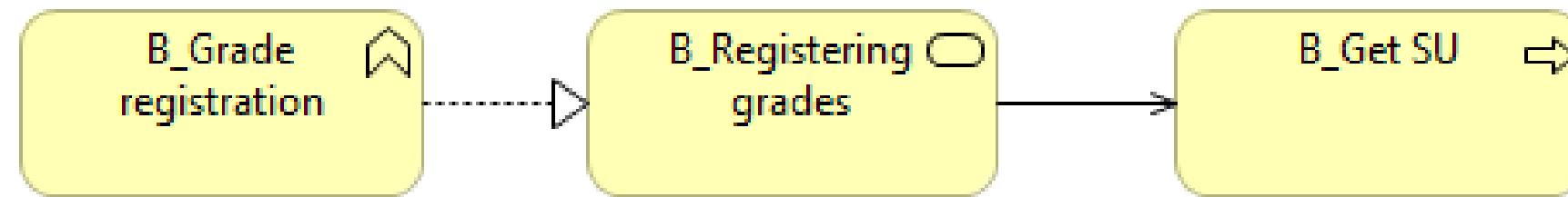
Behavior elements: example



Behavior elements: example



Behavior elements: example

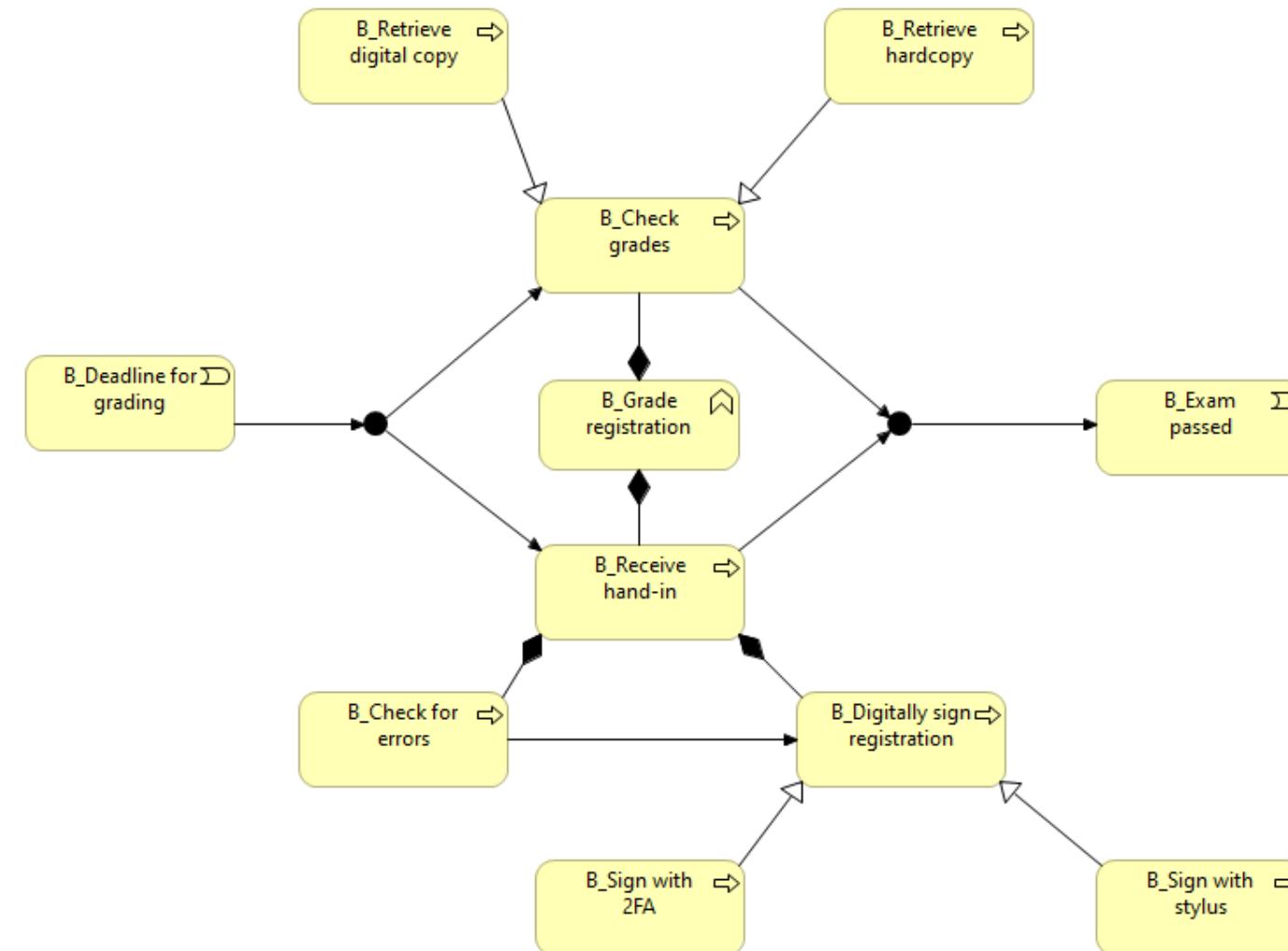


Passive elements

Business
object

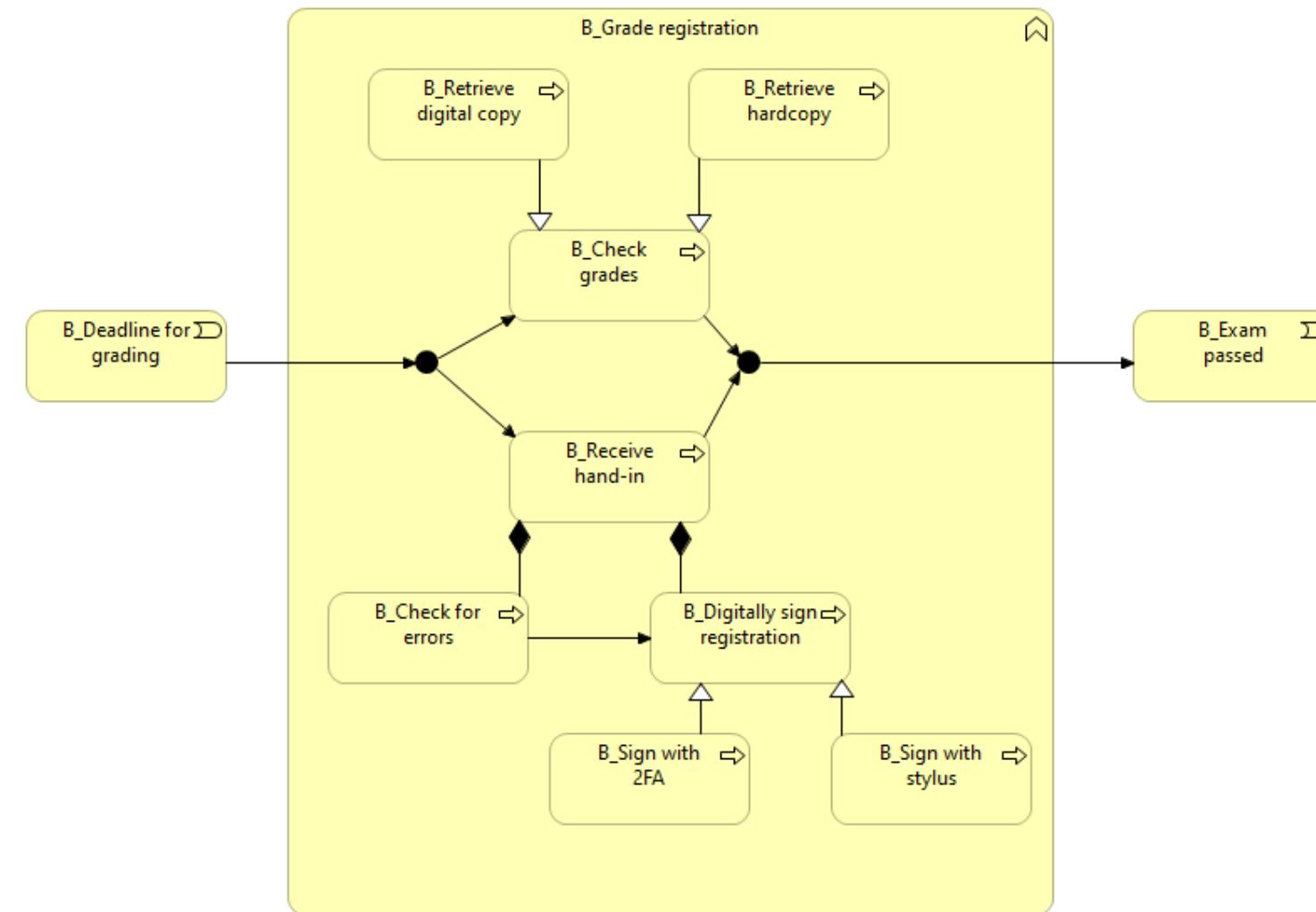
A concept used within a particular
business domain

Nesting example



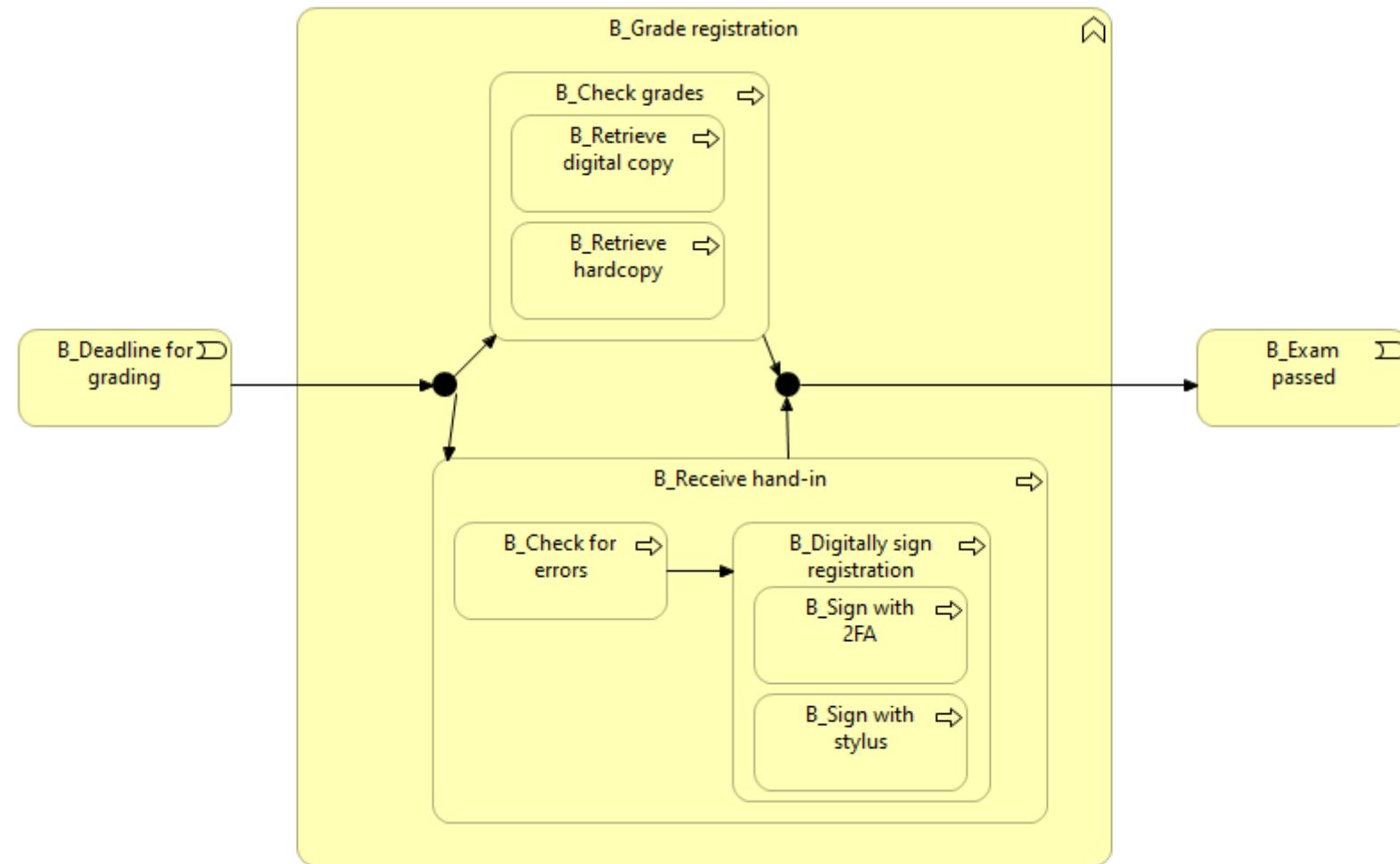
- Each element is shown with all its relations

Nesting example



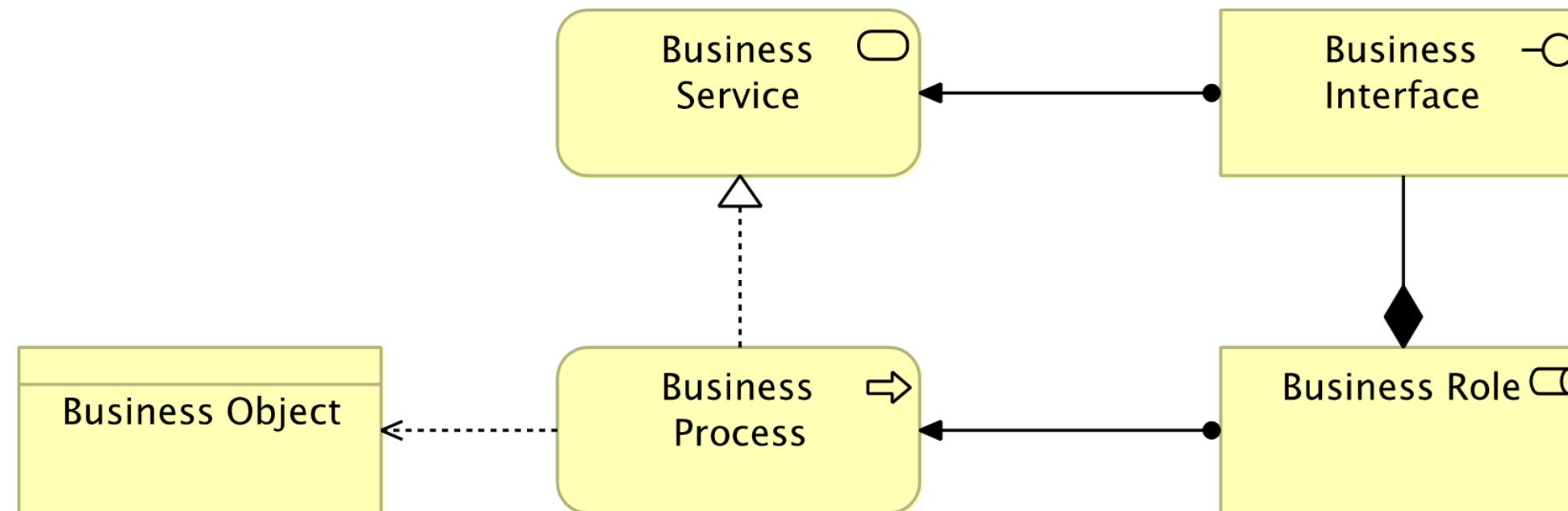
- Most relations can be expressed by nesting elements

Nesting example



- Doing so, we have a more compact visualization, but we lose details

Basic business pattern



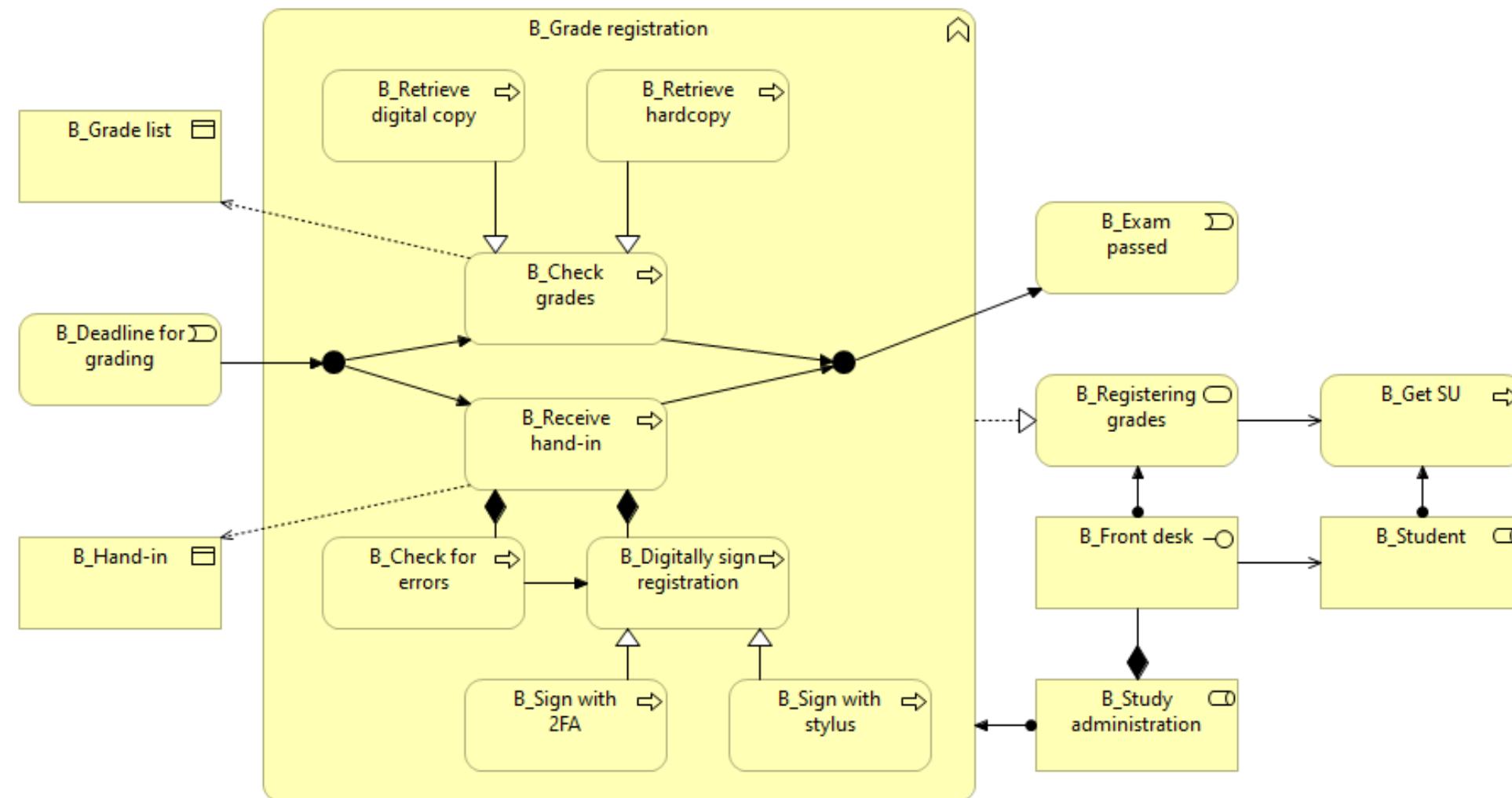
← This is the **Access** relation

→ This is the **Composition** relation

▷ This is the **Realization** relation

↔ This is the **Assignment** relation

Basic business pattern: example



Application layer

Active elements

Application component



An encapsulation of application functionality aligned to implementation structure, which is modular and replaceable

Application interface



A point of access where an application service is made available to a user, another application component, or a node

Behavior elements

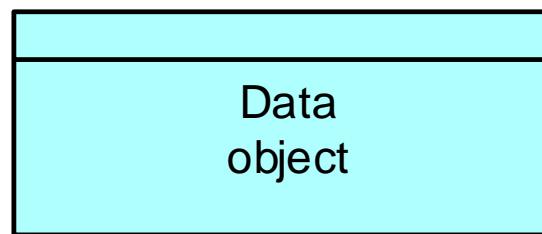
Application
function

Automated behavior that can be performed by an application component

Application
service

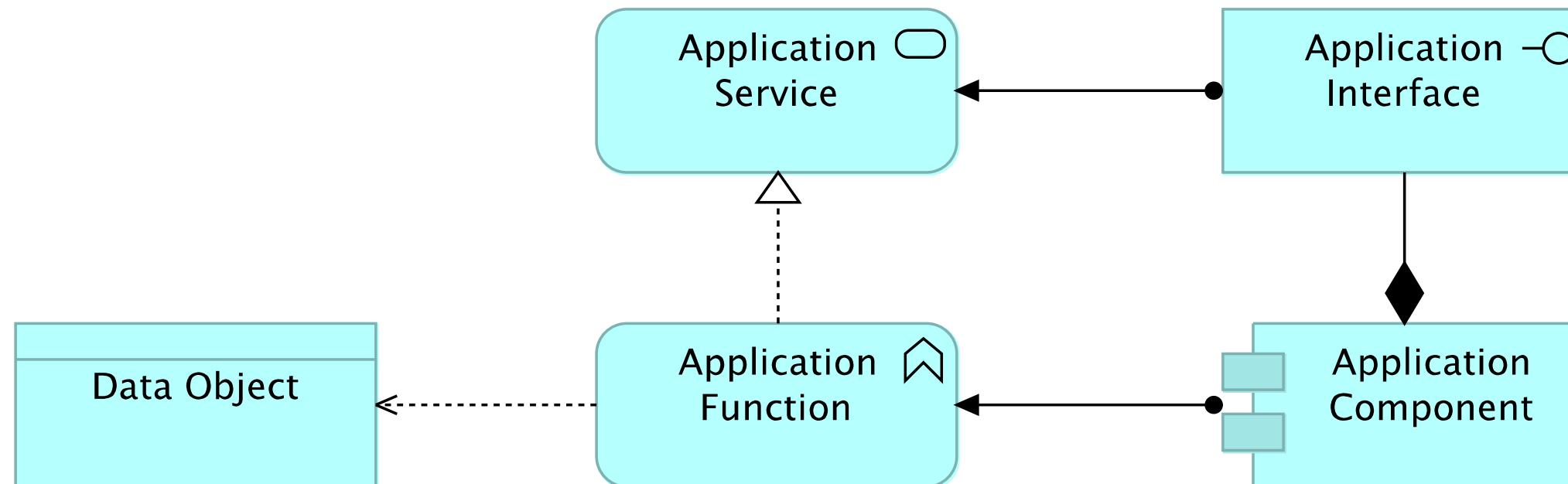
An explicitly defined exposed application behavior

Passive elements



Data structured for information processing

Basic application pattern



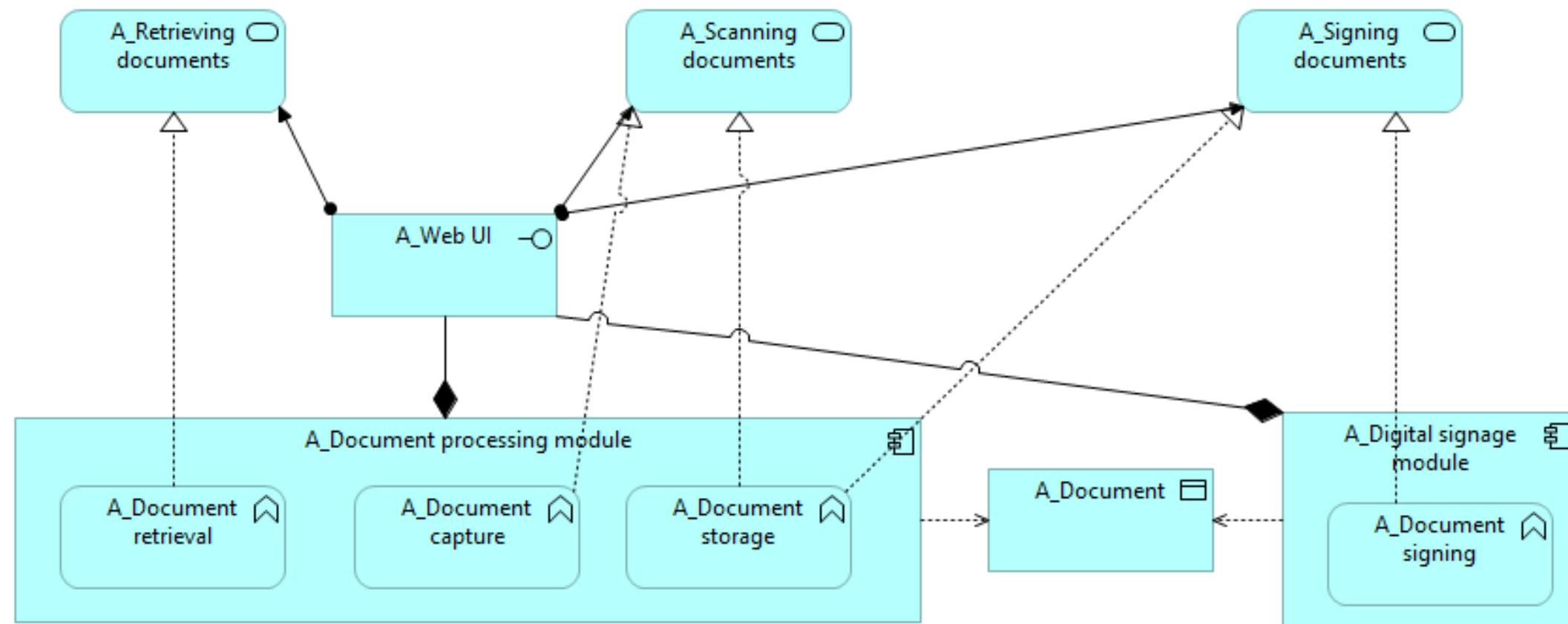
← This is the **Access** relation

→♦ This is the **Composition** relation

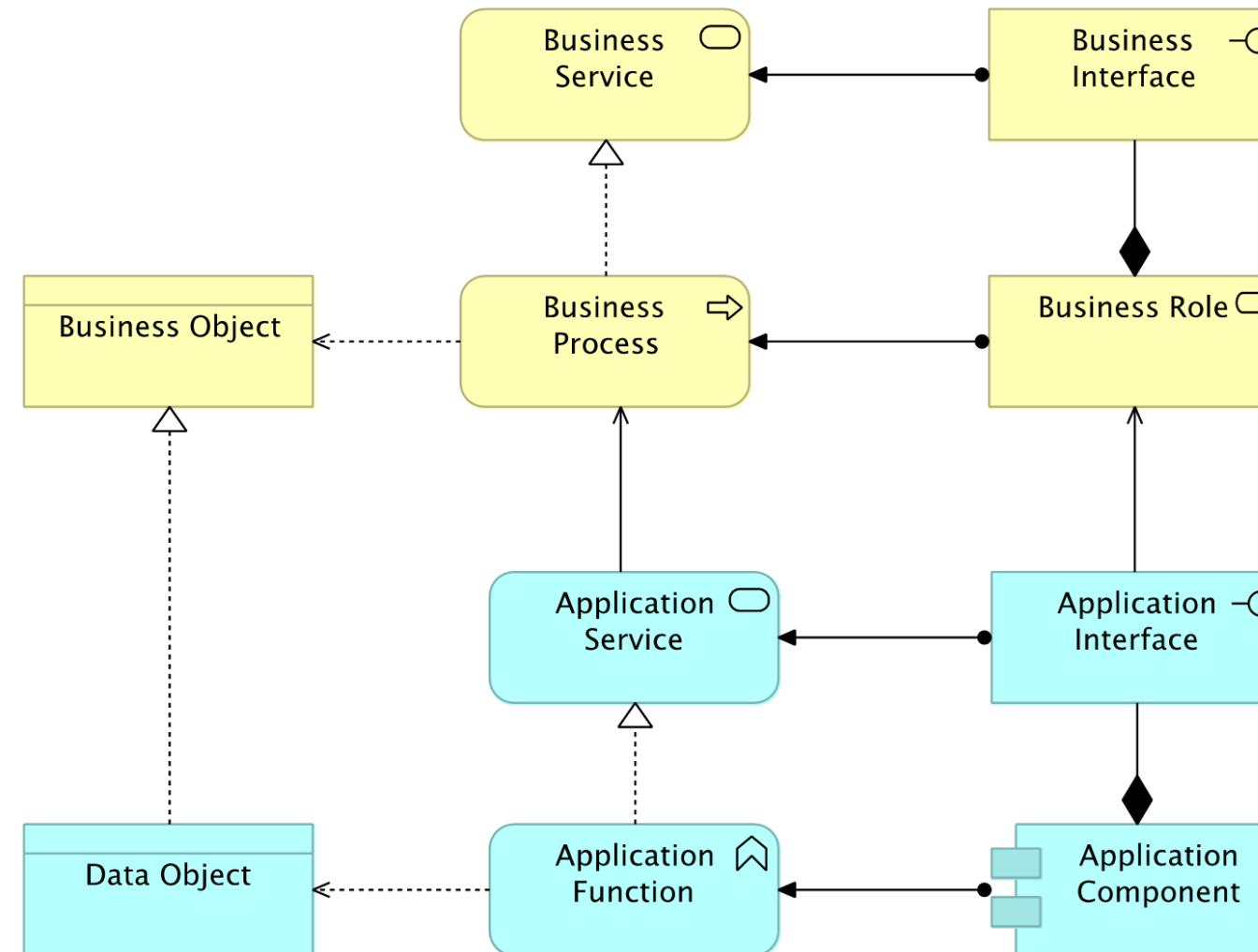
↔ This is the **Realization** relation

←● This is the **Assignment** relation

Basic application pattern: example



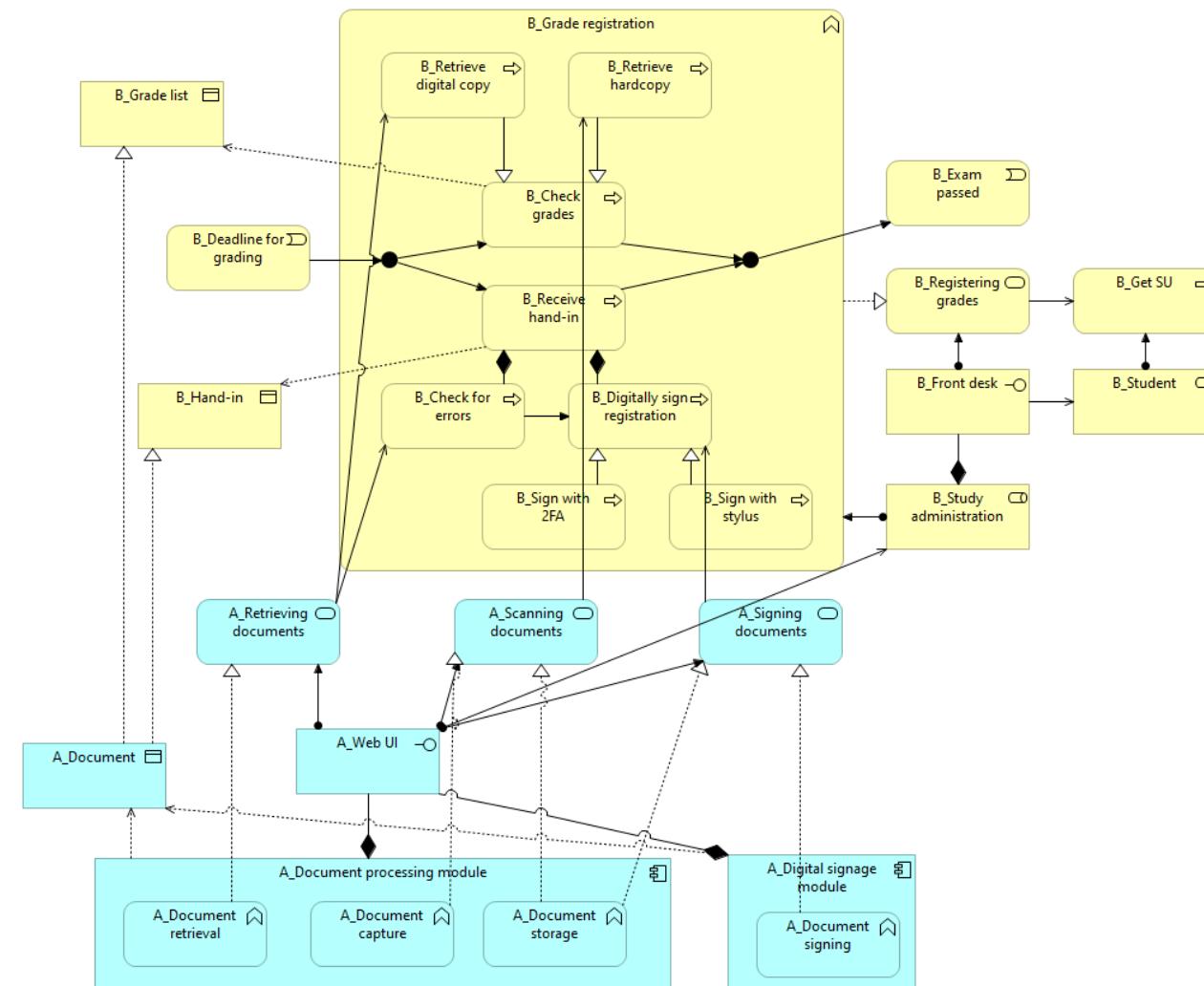
Business to application layer alignment



←

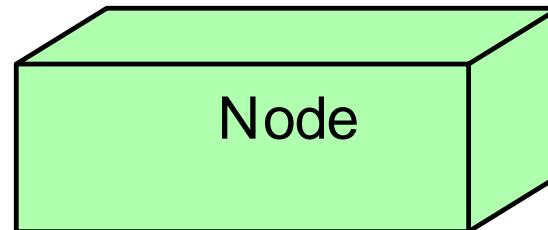
This is the *Serving* relation

Business to application layer alignment: example



Technology layer

Active elements

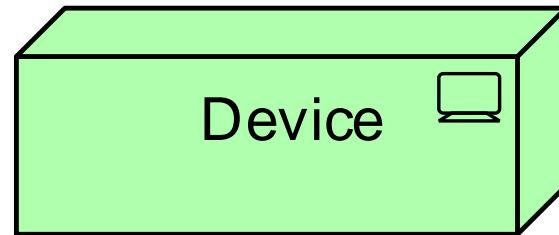


A computational or physical resource that hosts, manipulates or interacts with other computational or physical resources

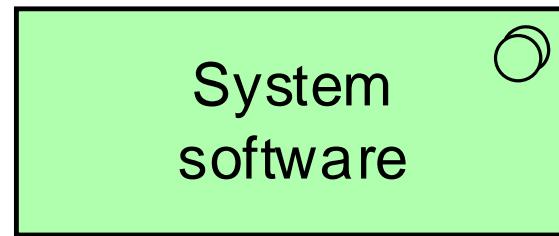


A point of access where a technology service offered by a node is made accessible

Active elements



A physical IT resource upon which system software and artifacts may be deployed

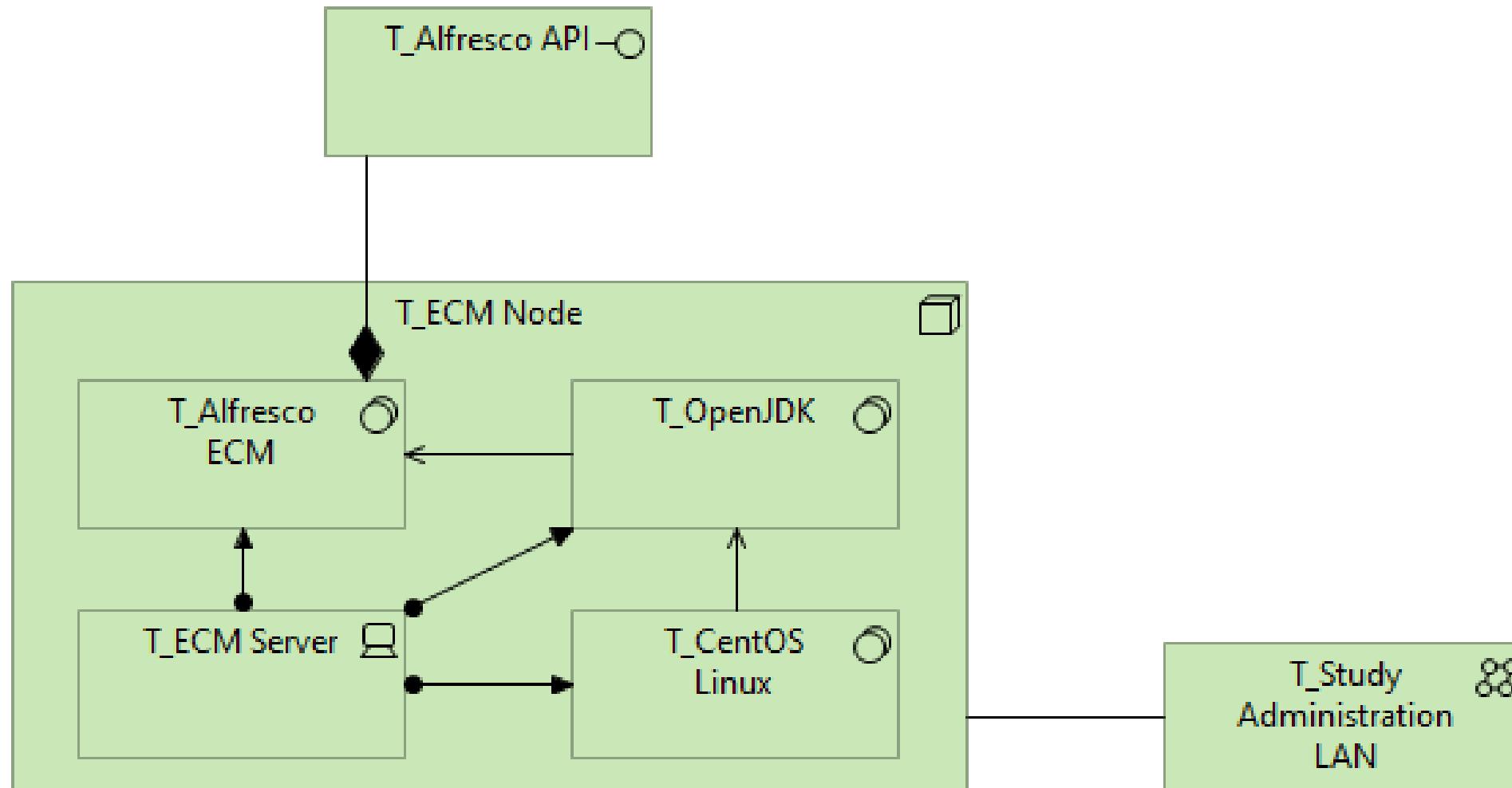


Software that provides or contributes to an environment for storing, executing and using software and data deployed within it



A set of structures that connects computer systems or other devices for transmission, routing and reception of data

Active elements: example



Behavior elements

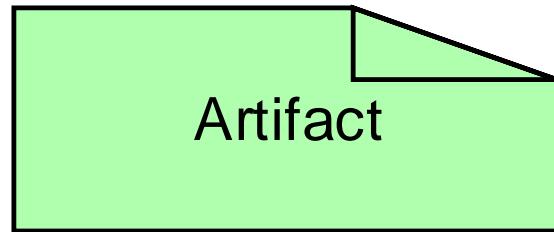
Technology
function

A behavior element that groups infrastructural behavior that can be performed by a node

Technology service

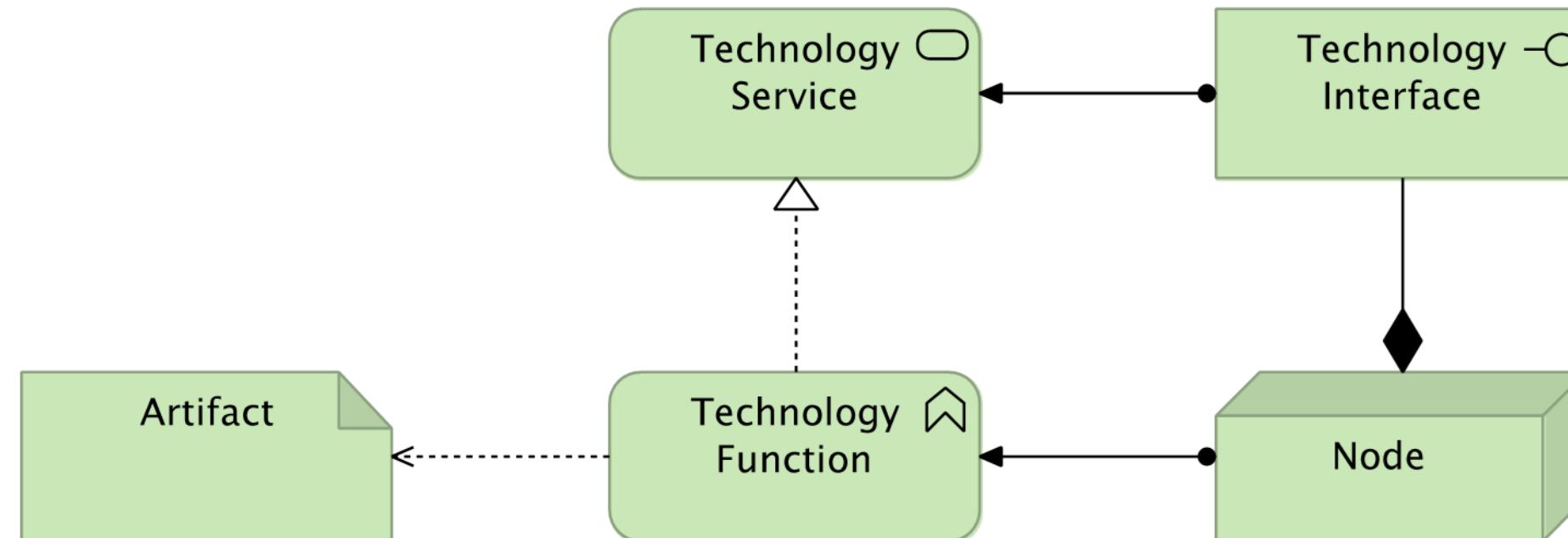
Externally visible unit of functionality, provided by one or more nodes, exposed through well-defined interfaces

Passive elements



A piece of data that is used or produced in a software development process or by deployment and operation of a system

Basic technology pattern



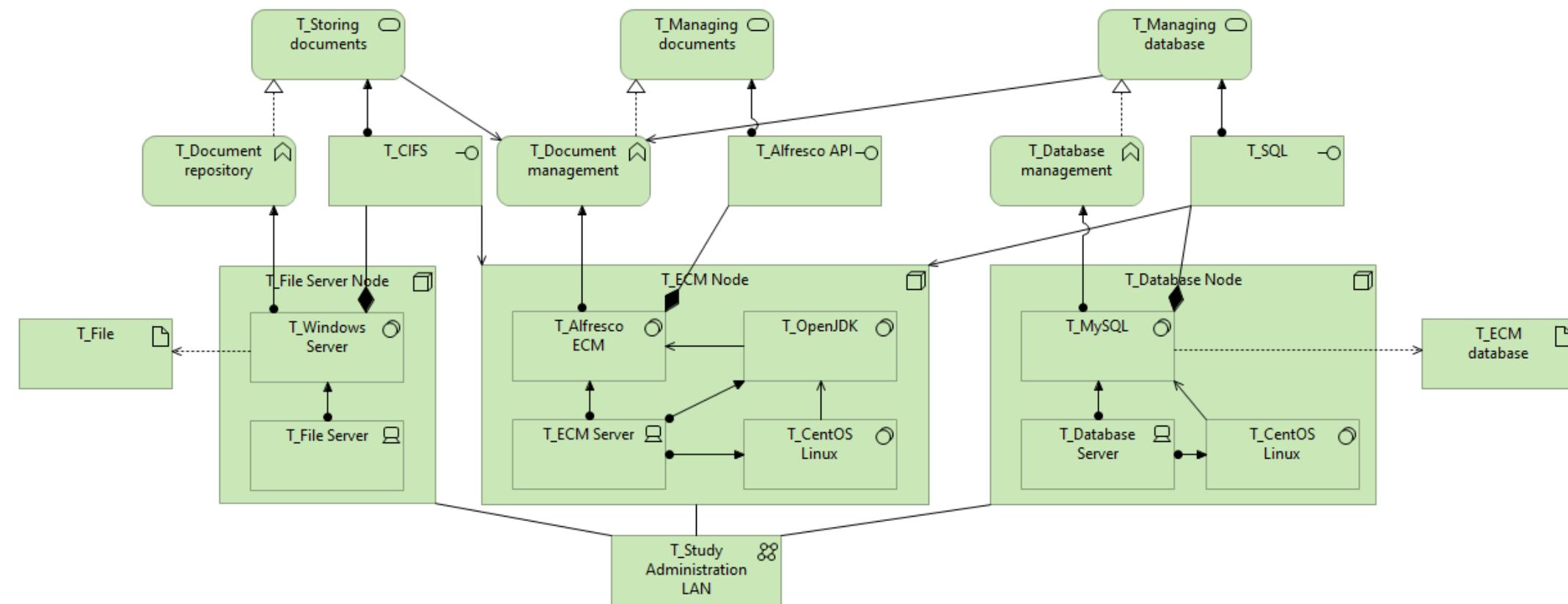
← This is the **Access** relation

→♦ This is the **Composition** relation

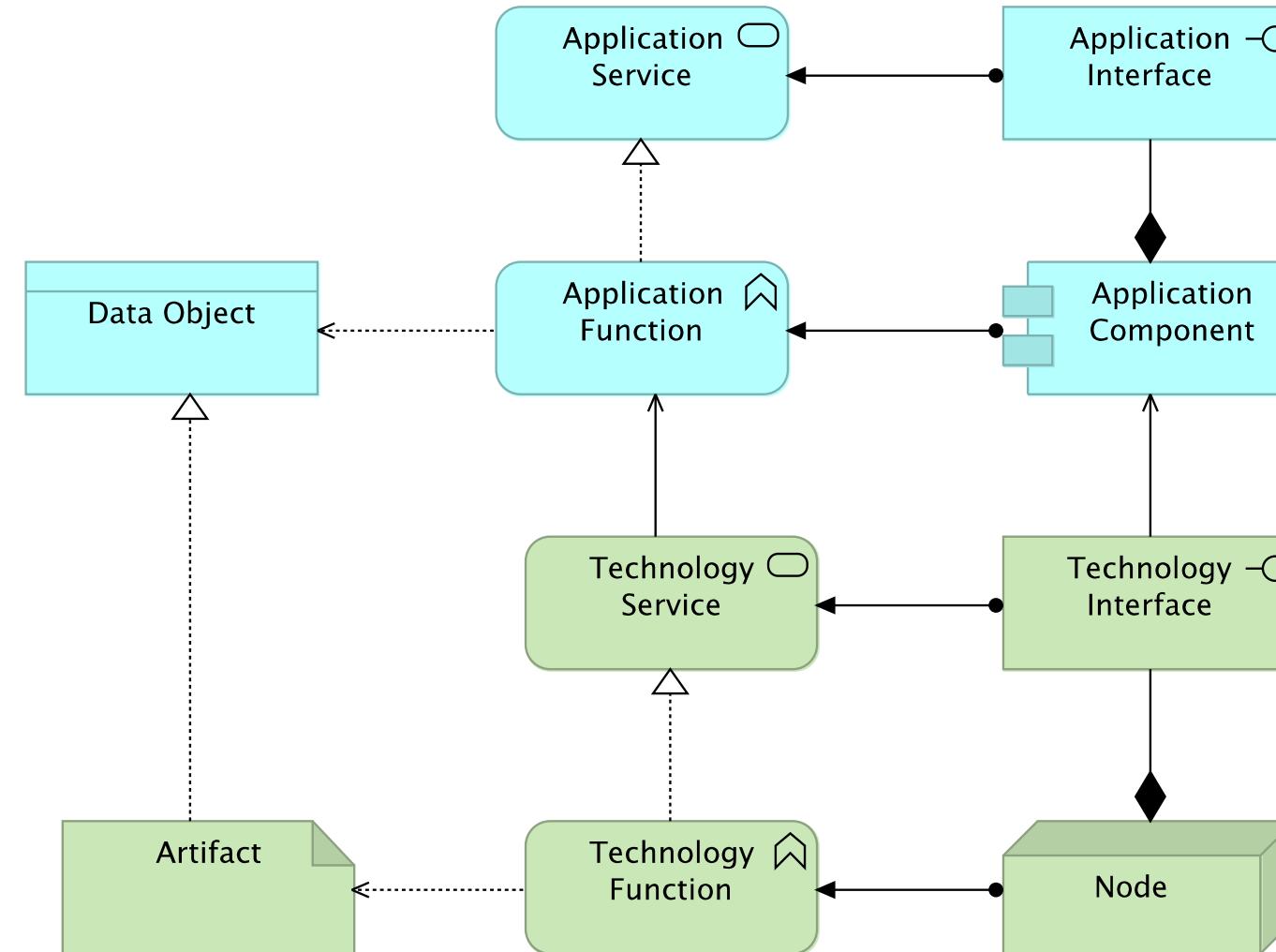
↔ This is the **Realization** relation

→— This is the **Assignment** relation

Basic technology pattern: example



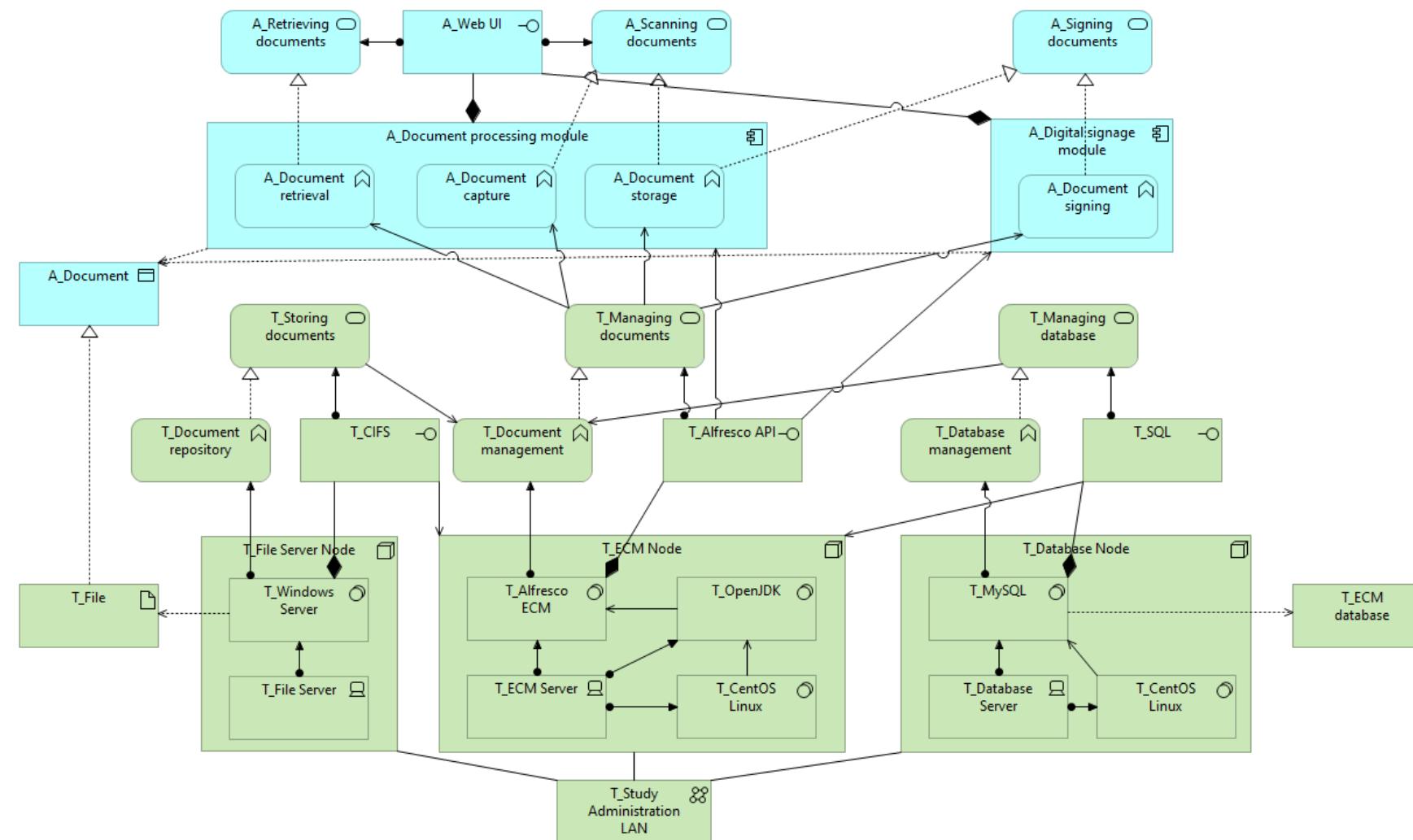
Application to technology alignment



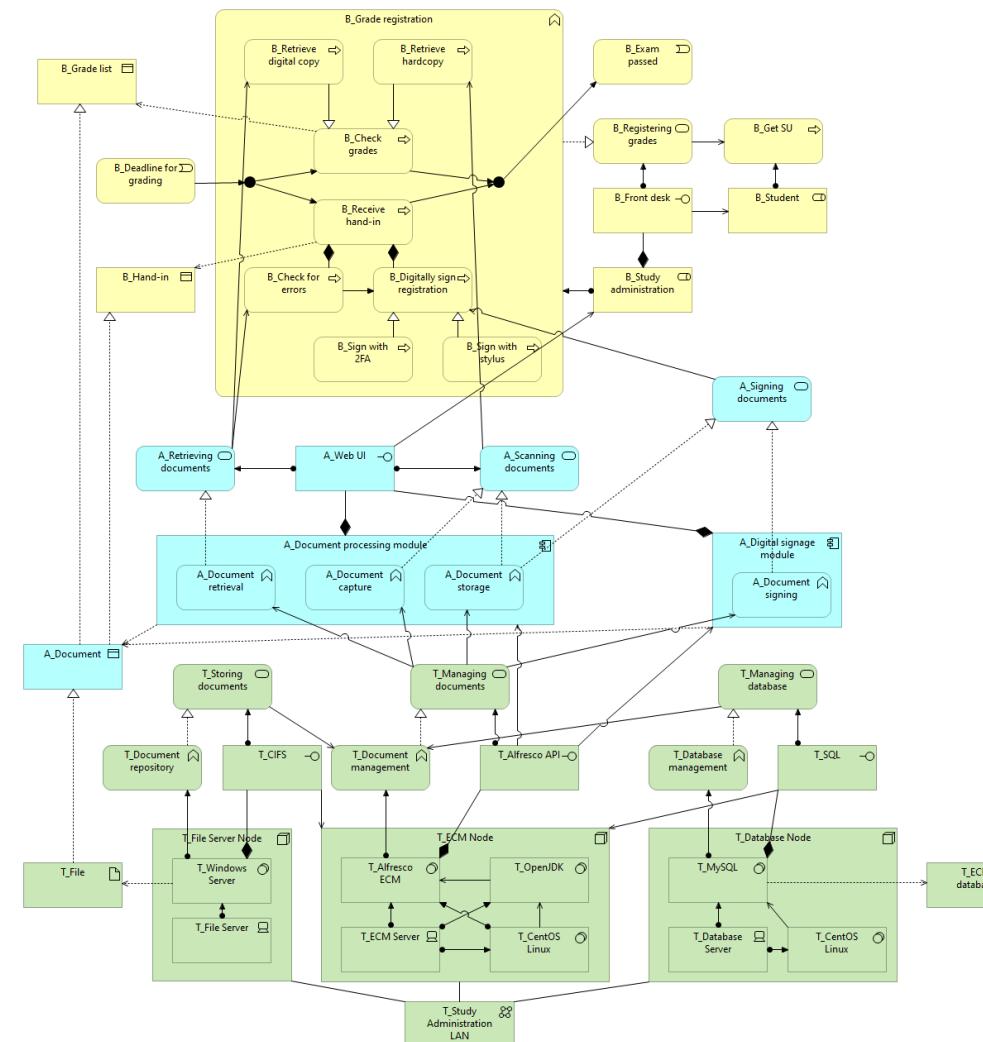
←

This is the **Serving** relation

Application to technology alignment: example



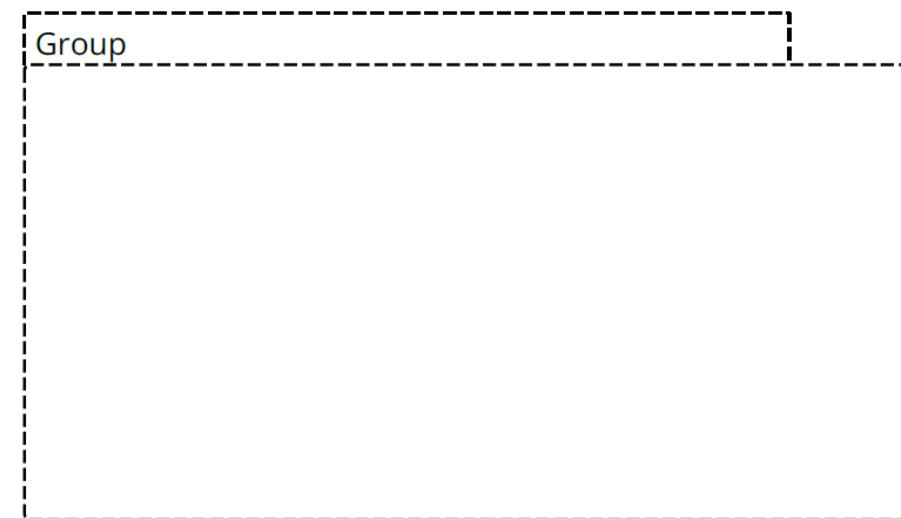
Complete example



Composite elements

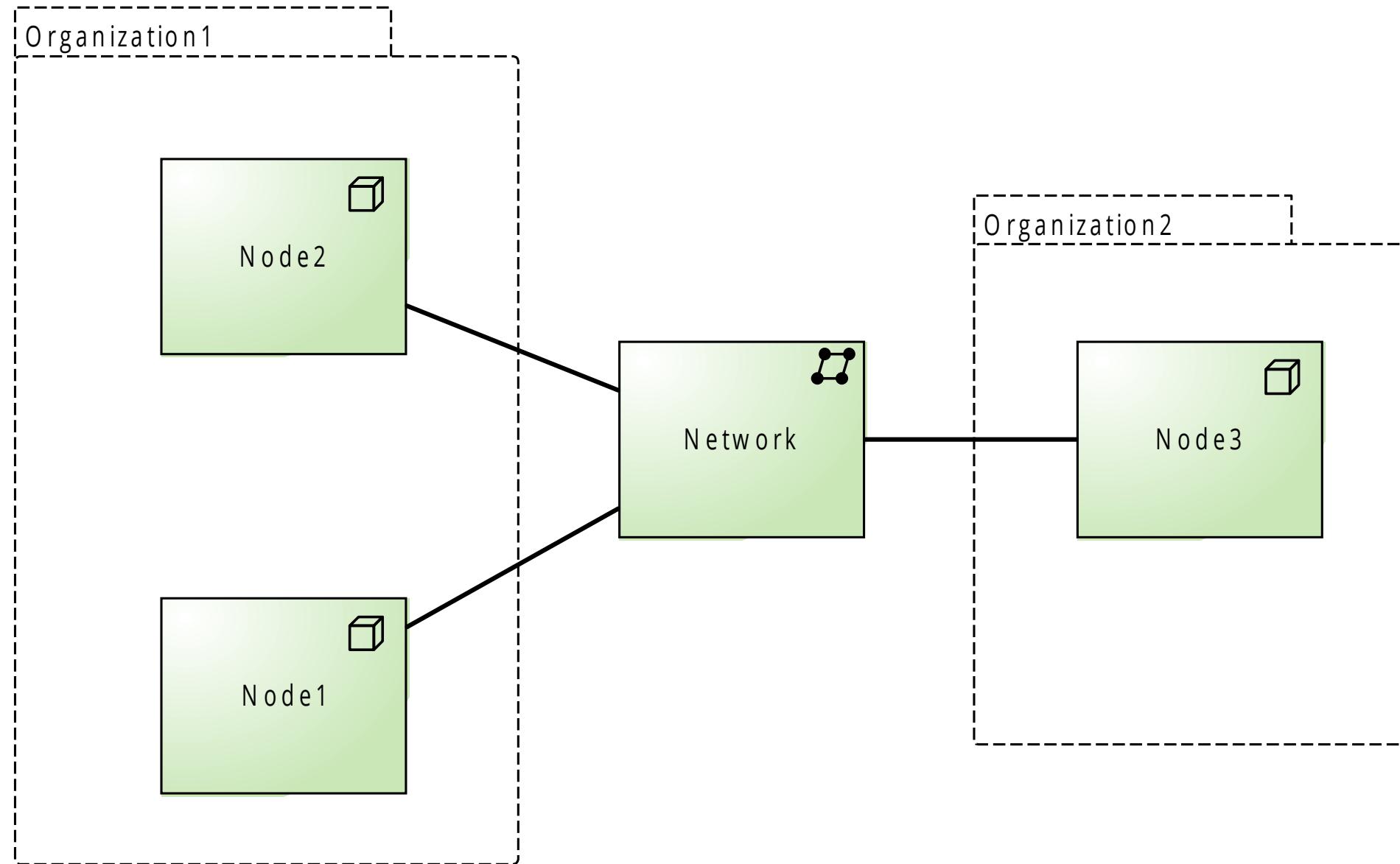
Grouping

- Aggregate elements together
- Aggregate elements of the same (external) organization

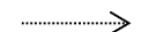
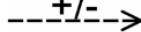


- We group external organizations, while we usually not group the target organization for clarity

Grouping example: two different organizations

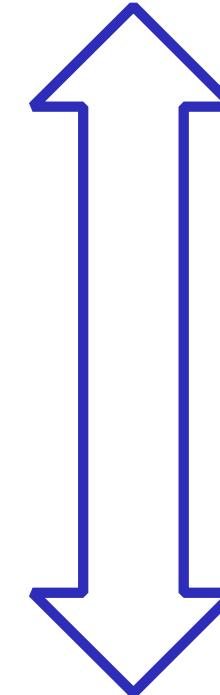


Relationships

Structural Relationships		Notation
Composition	Indicates that an element consists of one or more other concepts.	
Aggregation	Indicates that an element groups a number of other concepts.	
Assignment	Expresses the allocation of responsibility, performance of behavior, or execution.	
Realization	Indicates that an entity plays a critical role in the creation, achievement, sustenance, or operation of a more abstract entity.	
Dependency Relationships		Notation
Serving	Models that an element provides its functionality to another element.	
Access	Models the ability of behavior and active structure elements to observe or act upon passive structure elements.	 
Influence	Models that an element affects the implementation or achievement of some motivation element.	
Dynamic Relationships		Notation
Triggering	Describes a temporal or causal relationship between elements.	
Flow	Transfer from one element to another.	
Other Relationships		Notation
Specialization	Indicates that an element is a particular kind of another element.	
Association	Models an unspecified relationship, or one that is not represented by another ArchiMate relationship.	
Junction	Used to connect relationships of the same type.	(And) Junction  Or Junction 

Relationships strength

- Access
- Serving
- Realization
- Assignment
- Aggregation
- Composition

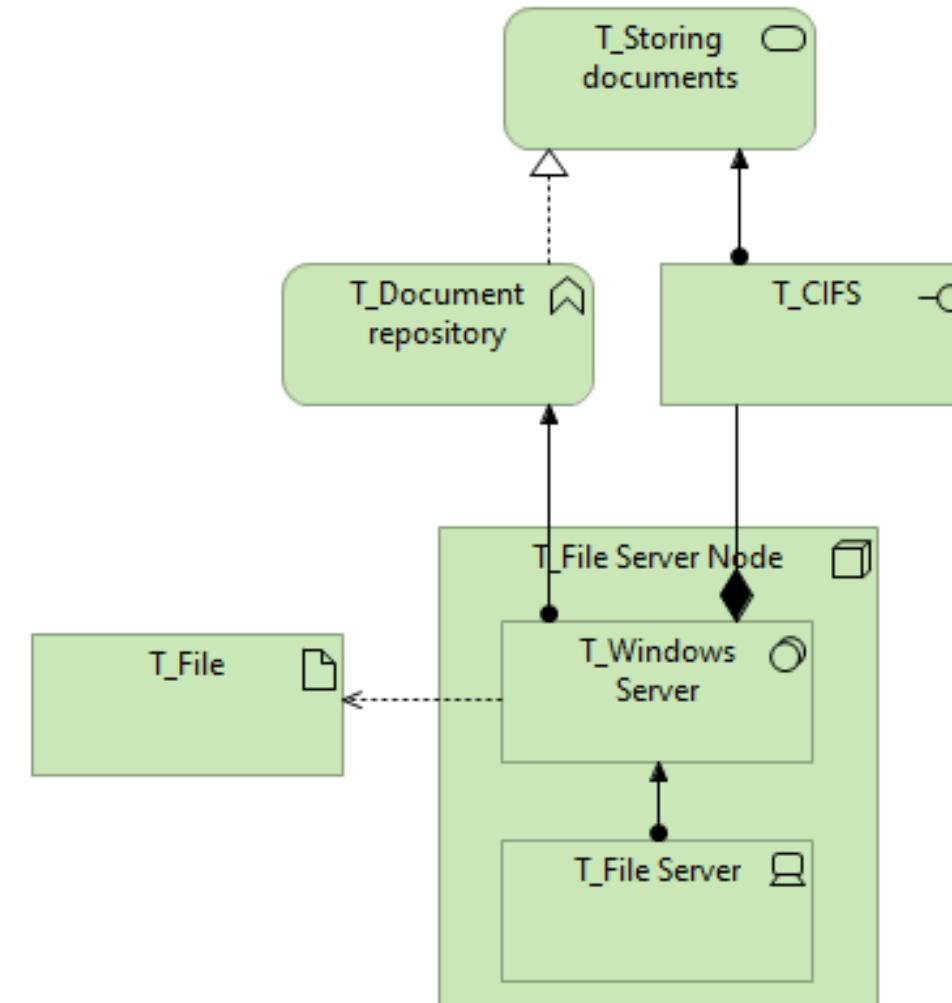


Weaker

Stronger

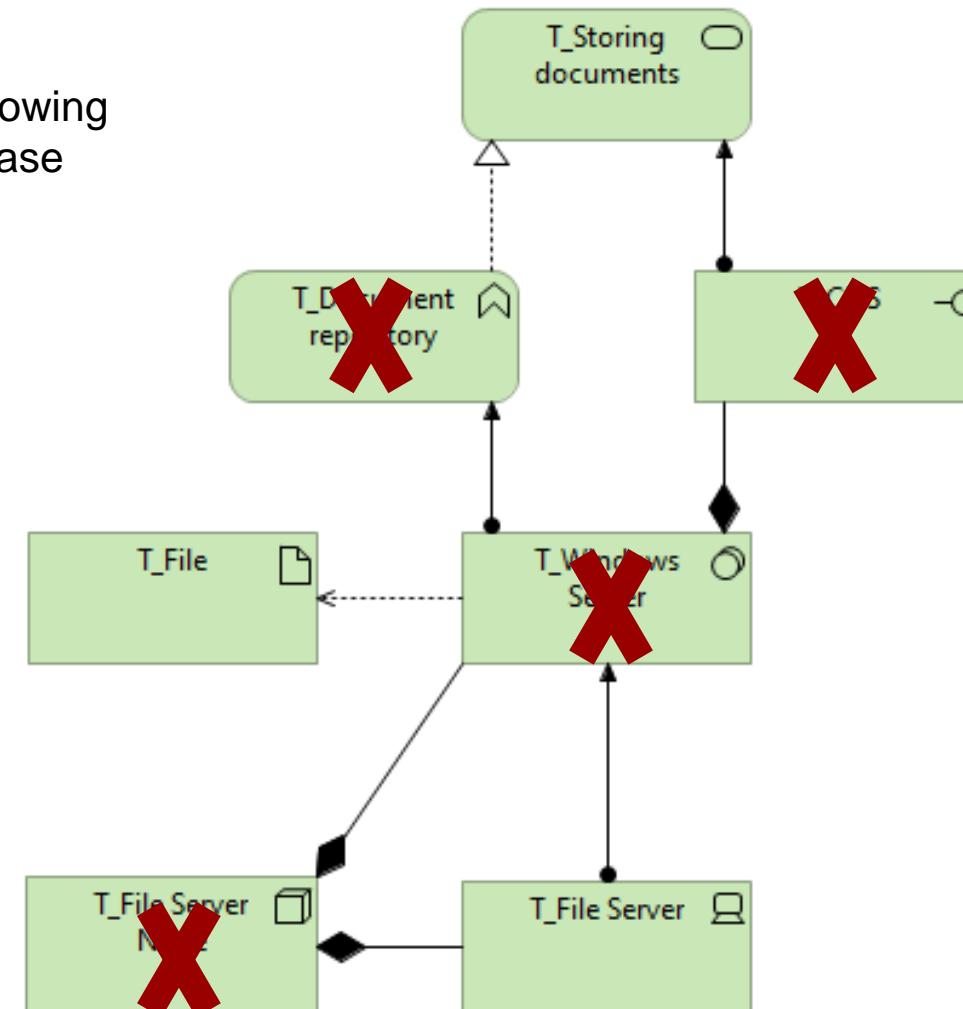
Association relation: generic relation

Derived Relationships: example



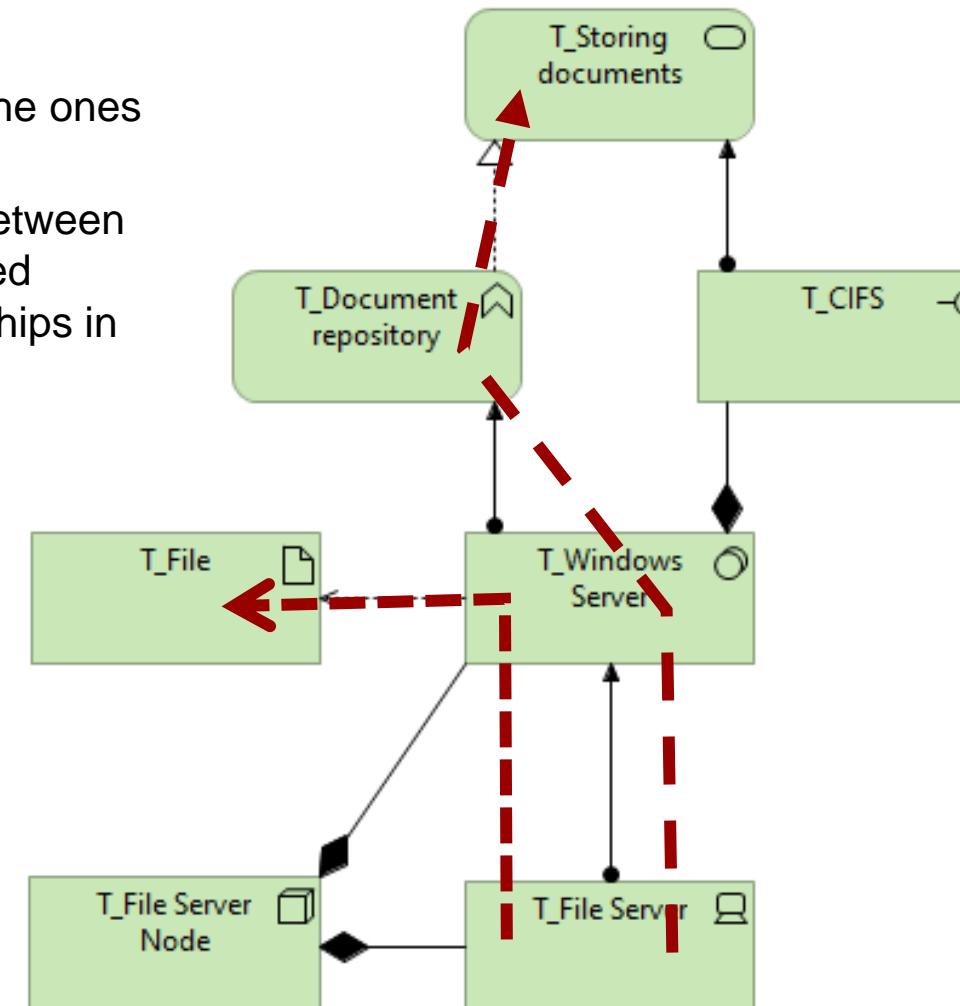
Derived Relationships: example

- We want to abstract from details, showing only elements relevant for our use case



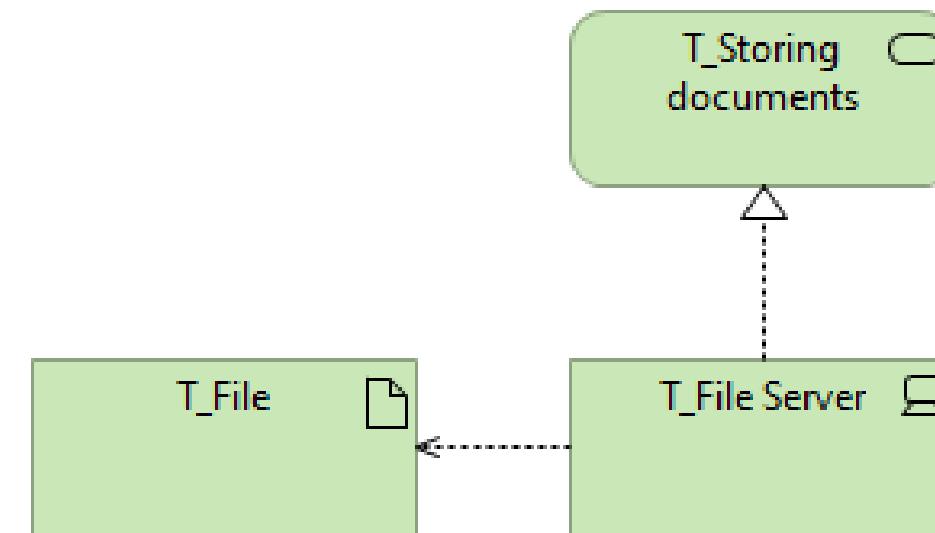
Derived Relationships: example

- We can derive relations, based on the ones in the detailed model.
- Intuitively, there must exist a path between two elements, and the type of derived relationship is the weakest relationships in the path



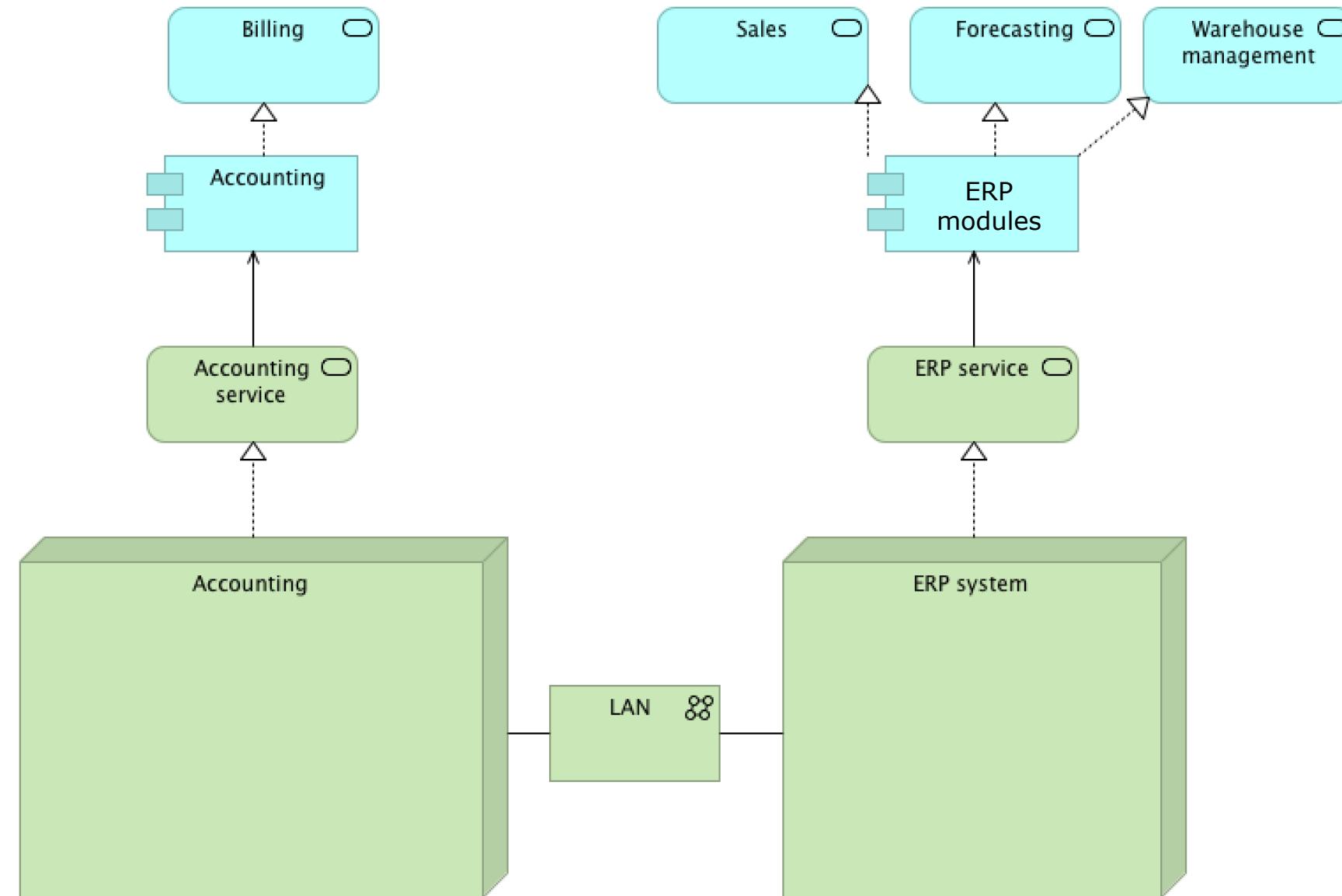
Derived Relationships: example

- The resulting simplified diagram still explains the relationships between elements

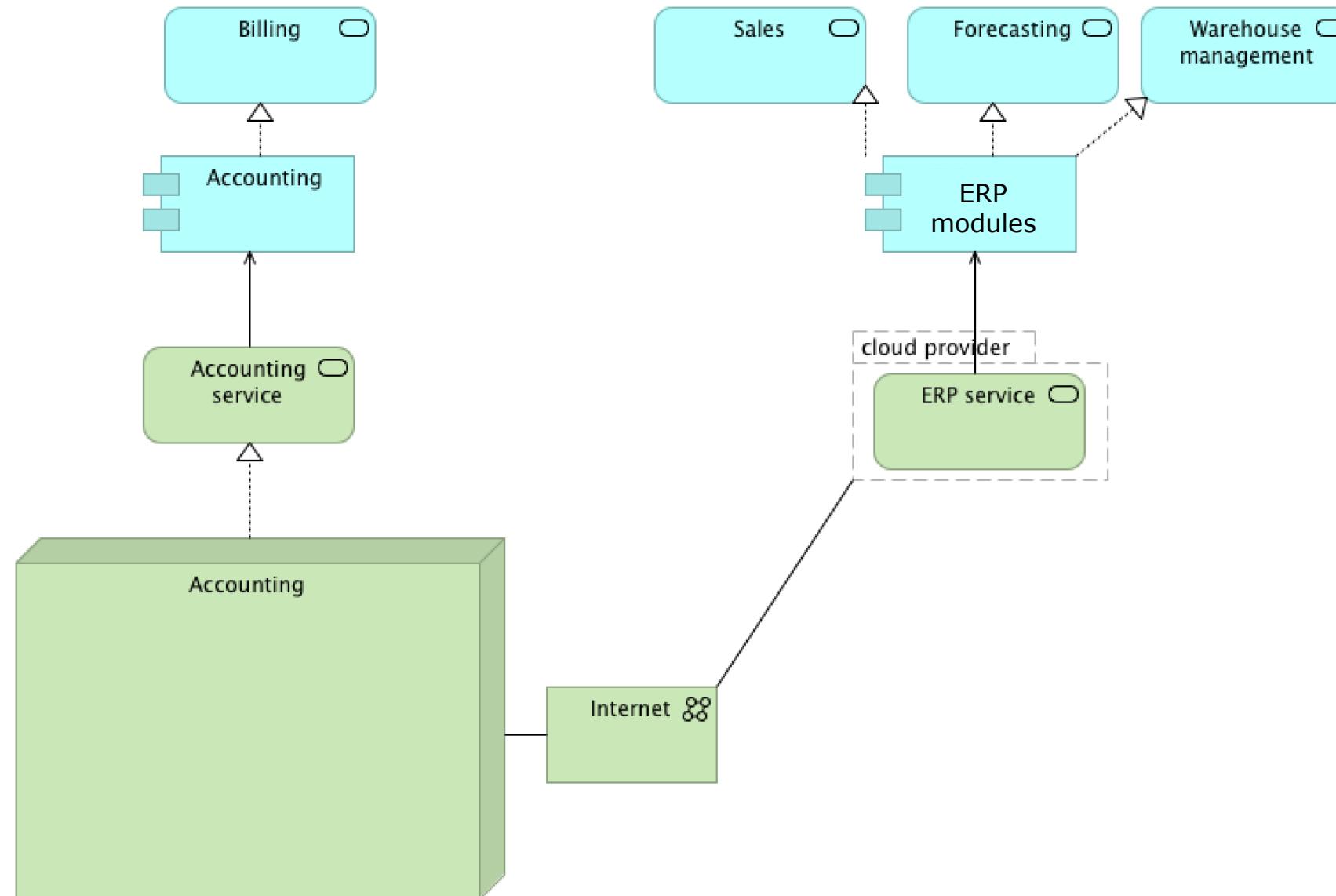


Useful patterns

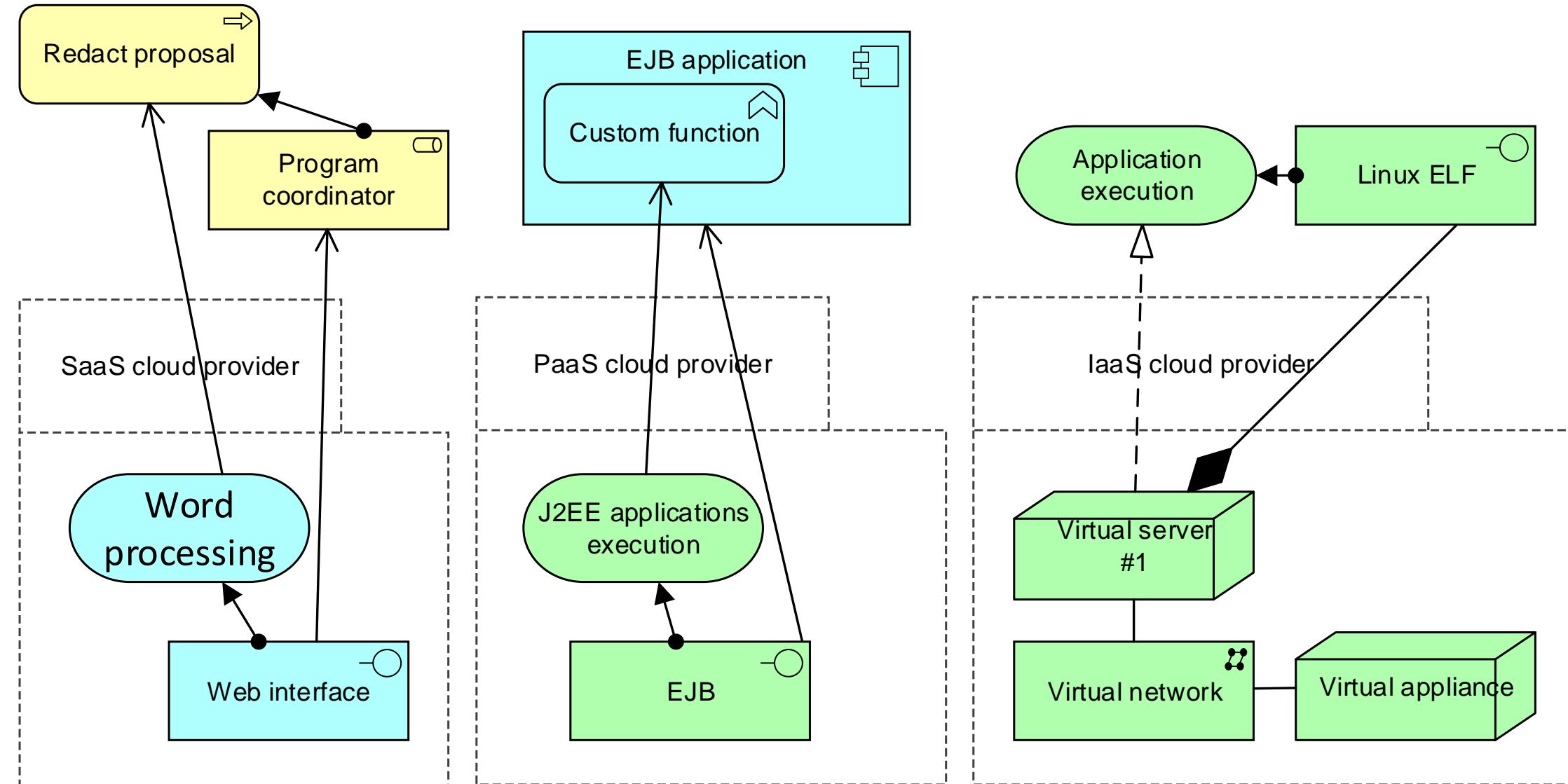
On-premise backend



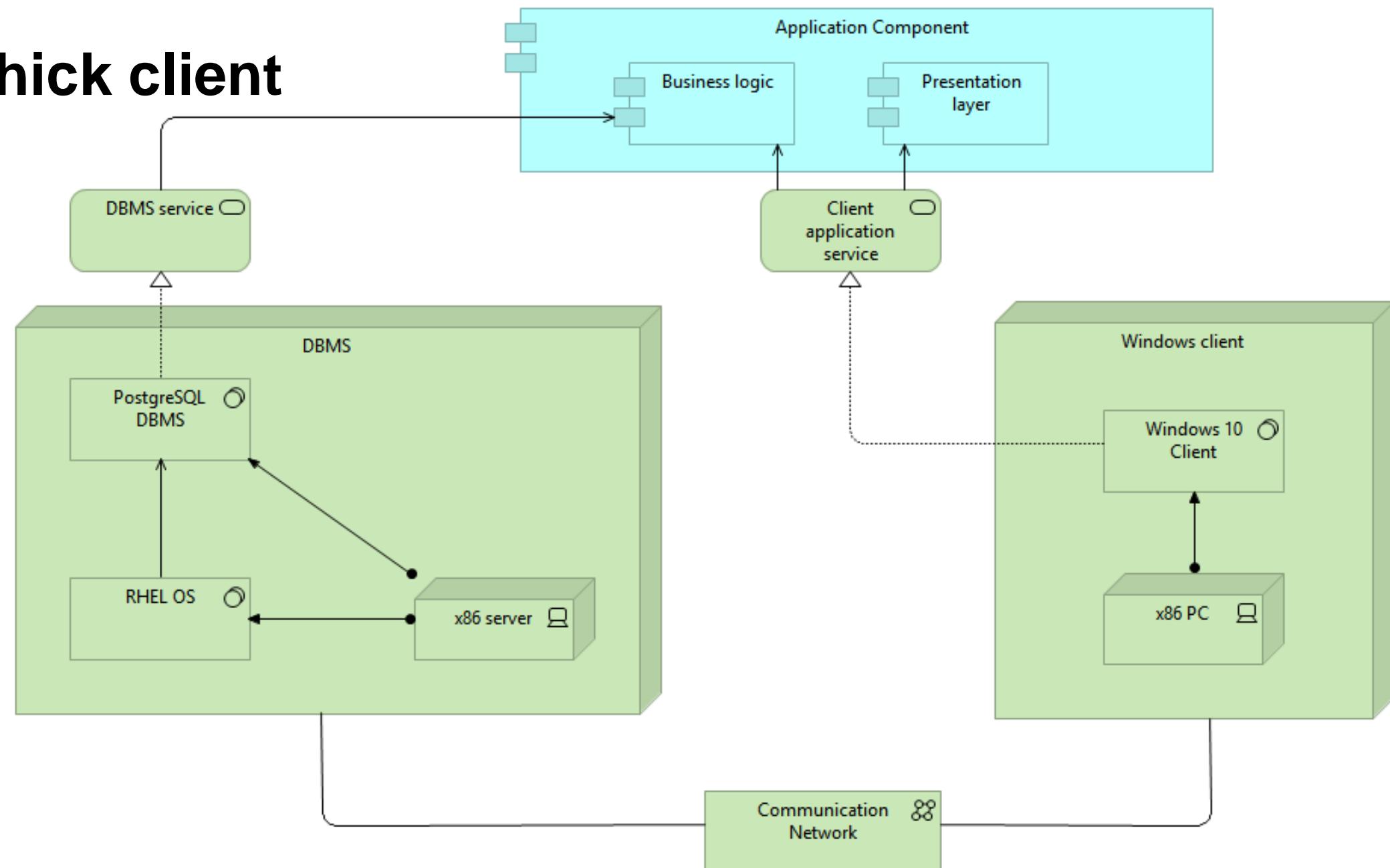
Switching to a cloud-based technology service



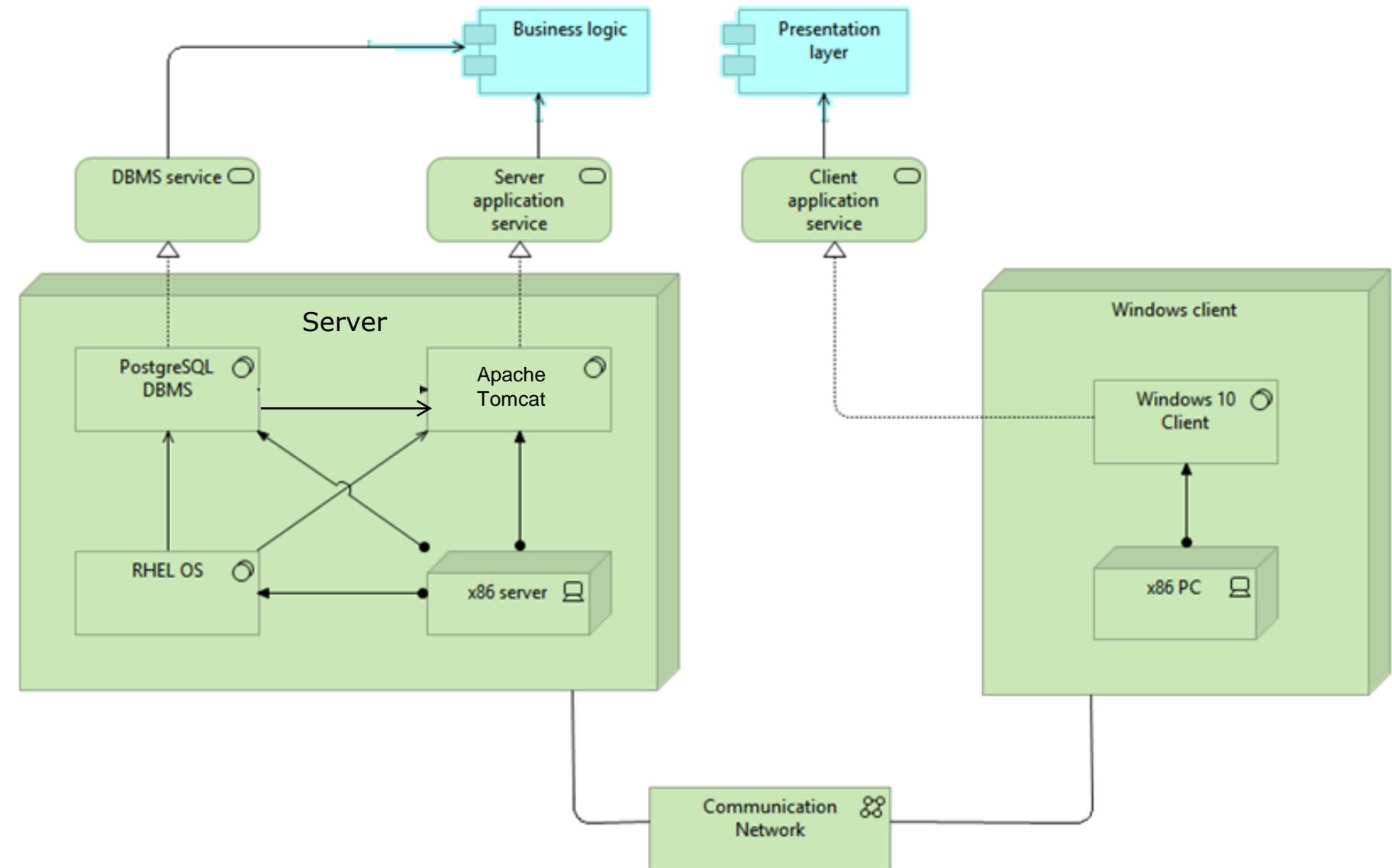
Cloud levels



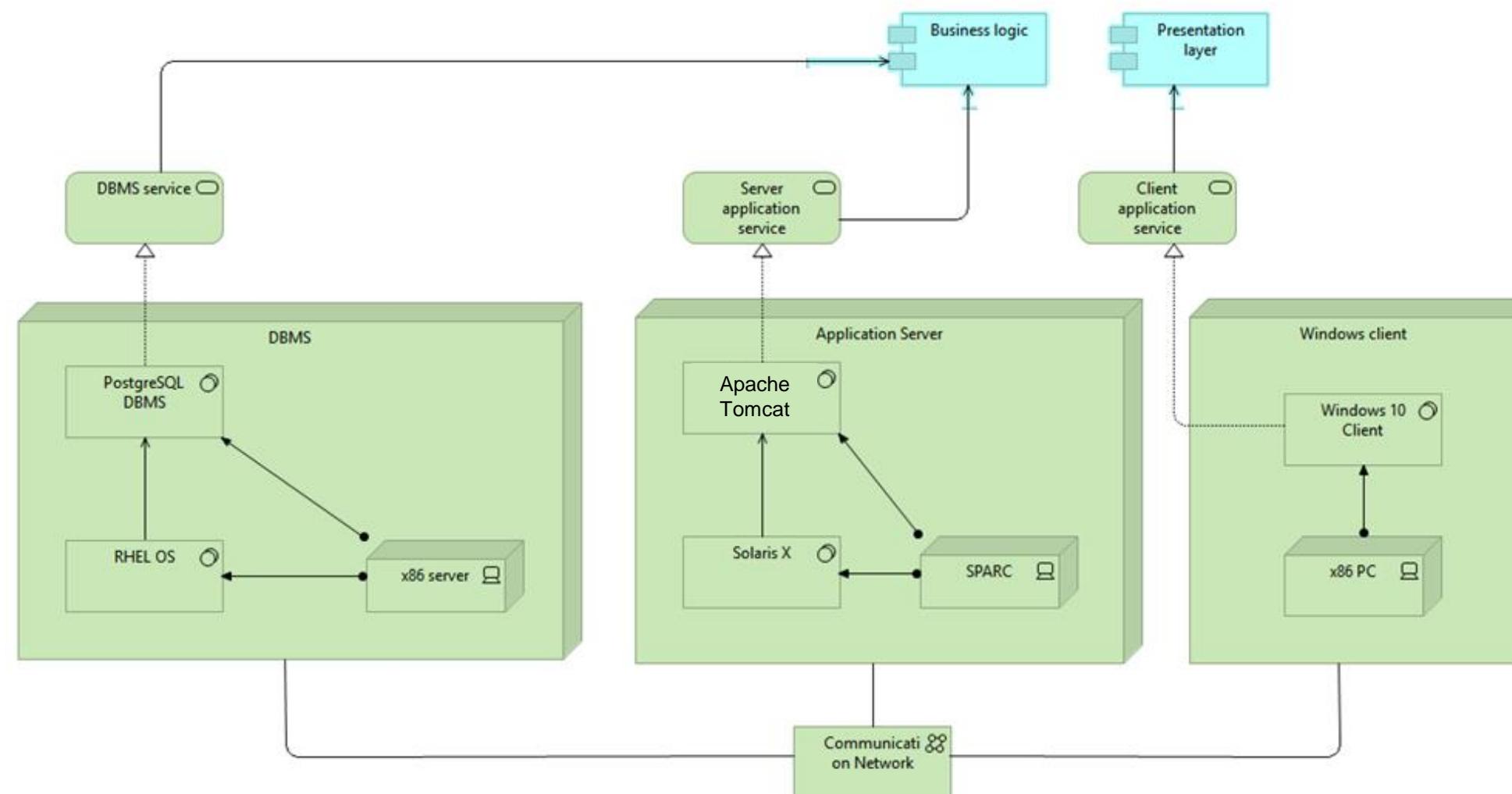
2-tier thick client



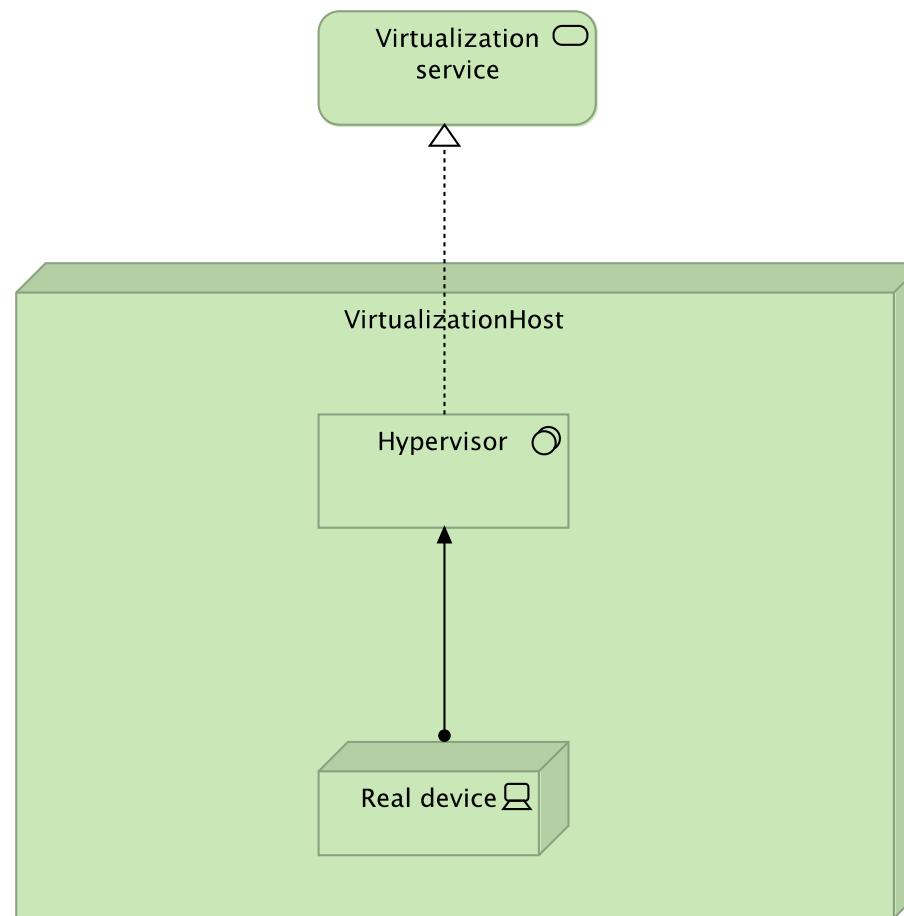
Switching to a 2-tier thin client



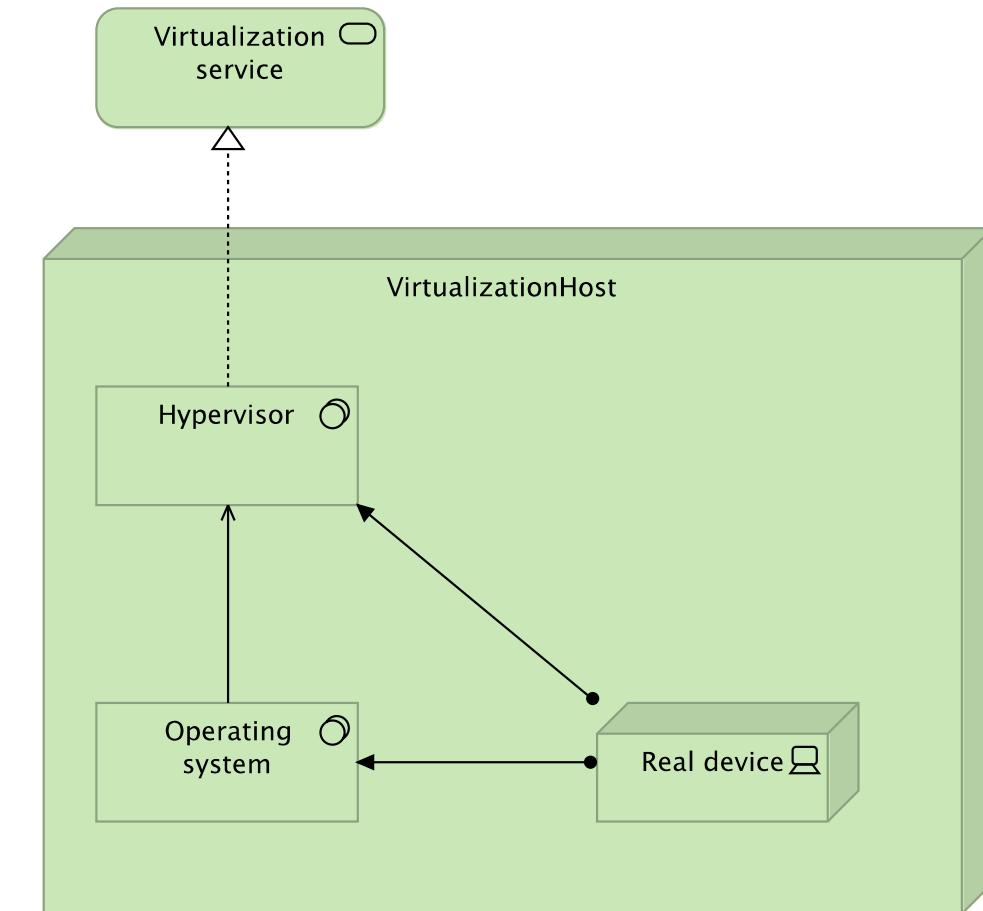
Switching to a 3-tier architecture



Bare metal vs hosted virtualization

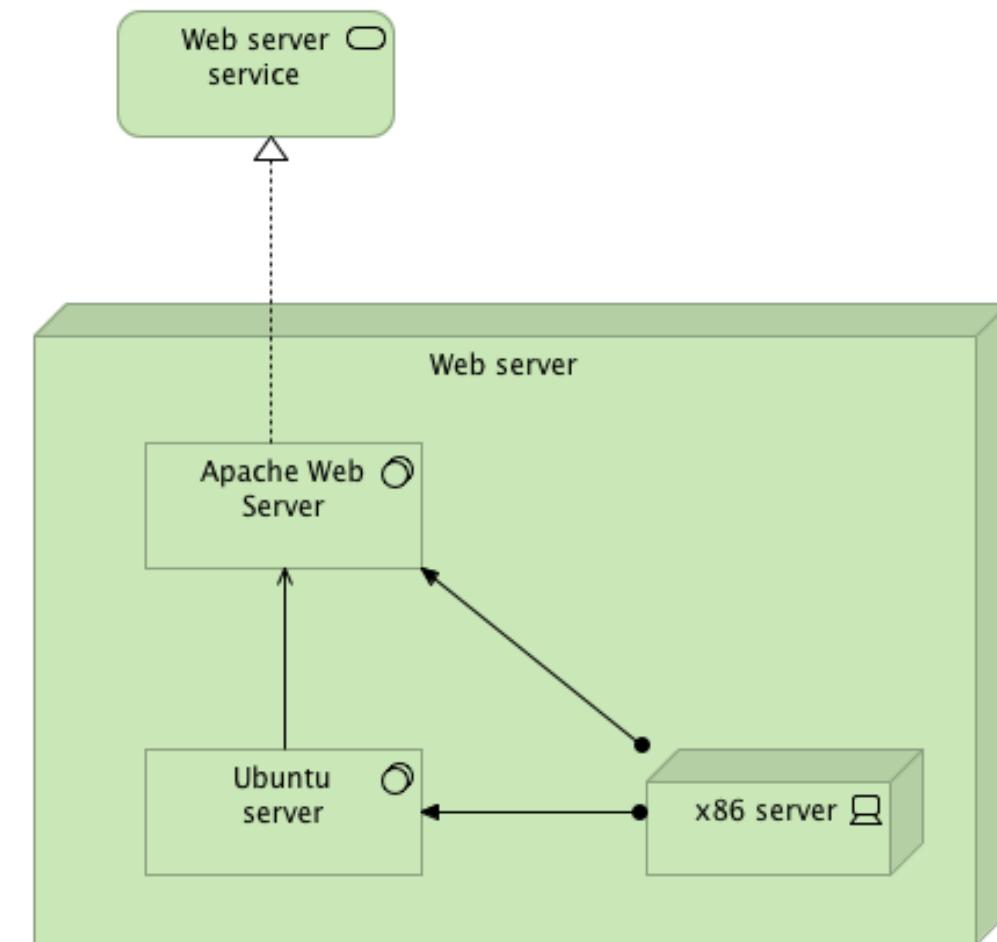
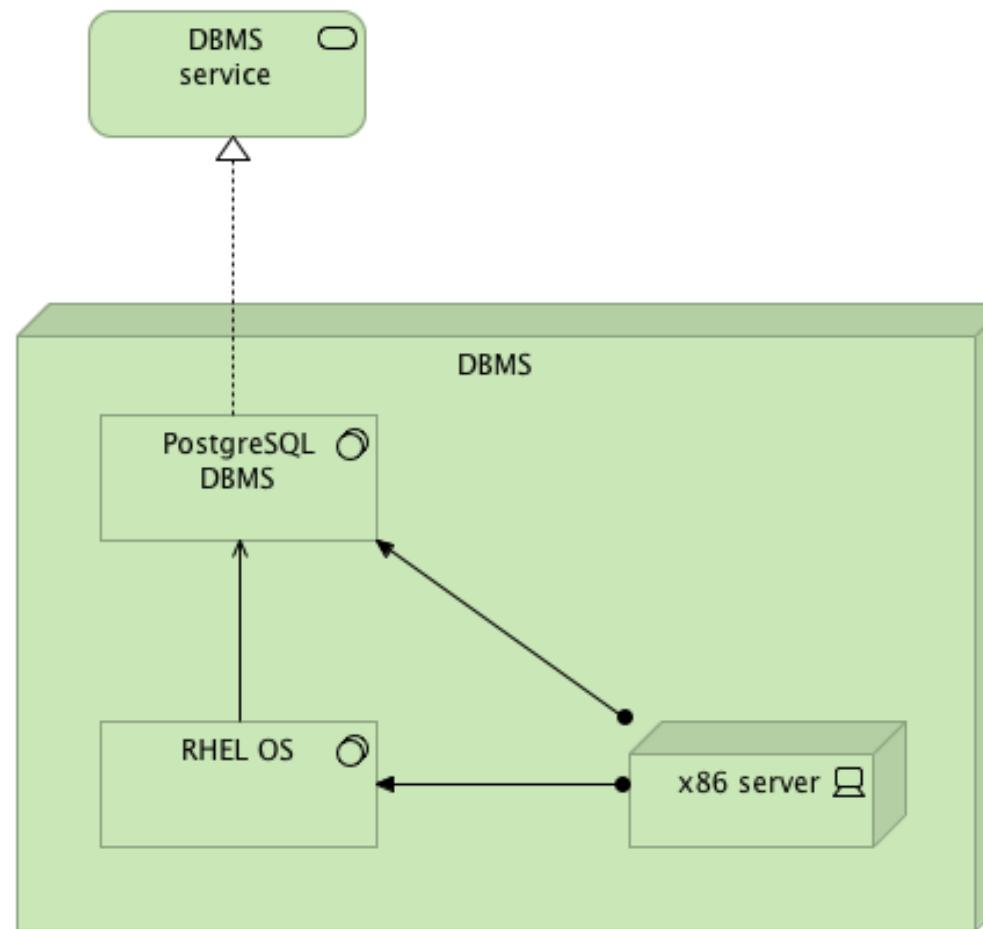


- Bare metal

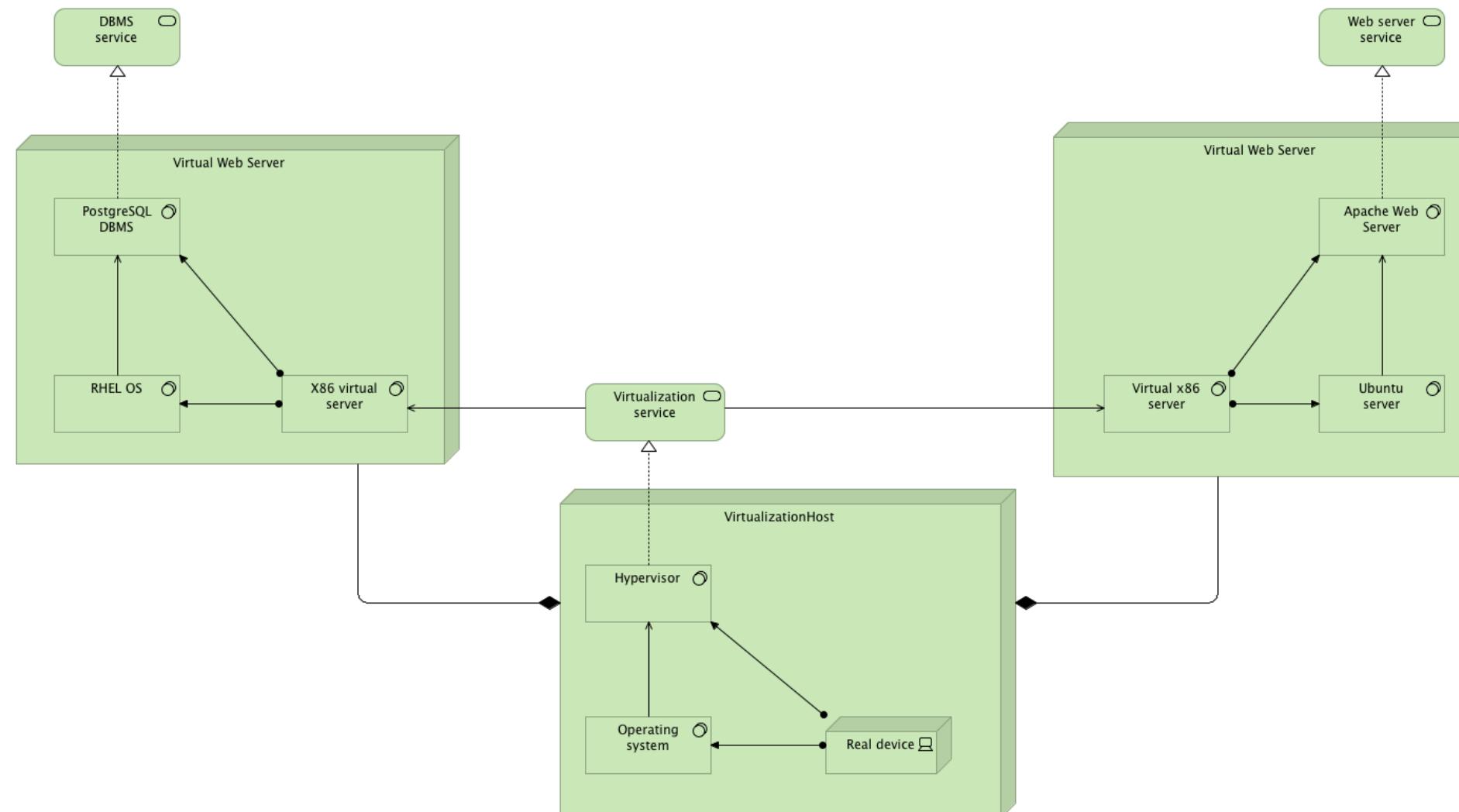


- Hosted

Physical servers



Virtualized servers



Study material

- Books and articles:
 - Lankhorst et al. - Enterprise Architectures at Work (4th Edition)
 - Available at: <https://link.springer.com/book/10.1007/978-3-662-53933-0>
 - Chapter 5.1 to 5.5, 5.8 to 5.10, 5.13
 - ArchiMate specification
 - Available at: <https://pubs.opengroup.org/architecture/archimate31-doc/>
- Modeling tools:
 - Archi: <https://www.archimatetool.com/>
 - (alternatively) SAP Signavio: <https://academic.signavio.com/p/login>

Exercises

Please answer all exercises to demonstrate your skills.

Solutions will be available at 11:45

Exercise 1 – Speedy

Speedy is a delivery company that wants to create a new reporting system for the top management. After inspecting the sales reports, the top management may also need to query the existing ERP system, based on Oracle Fusion, to get detailed sales and HR information.

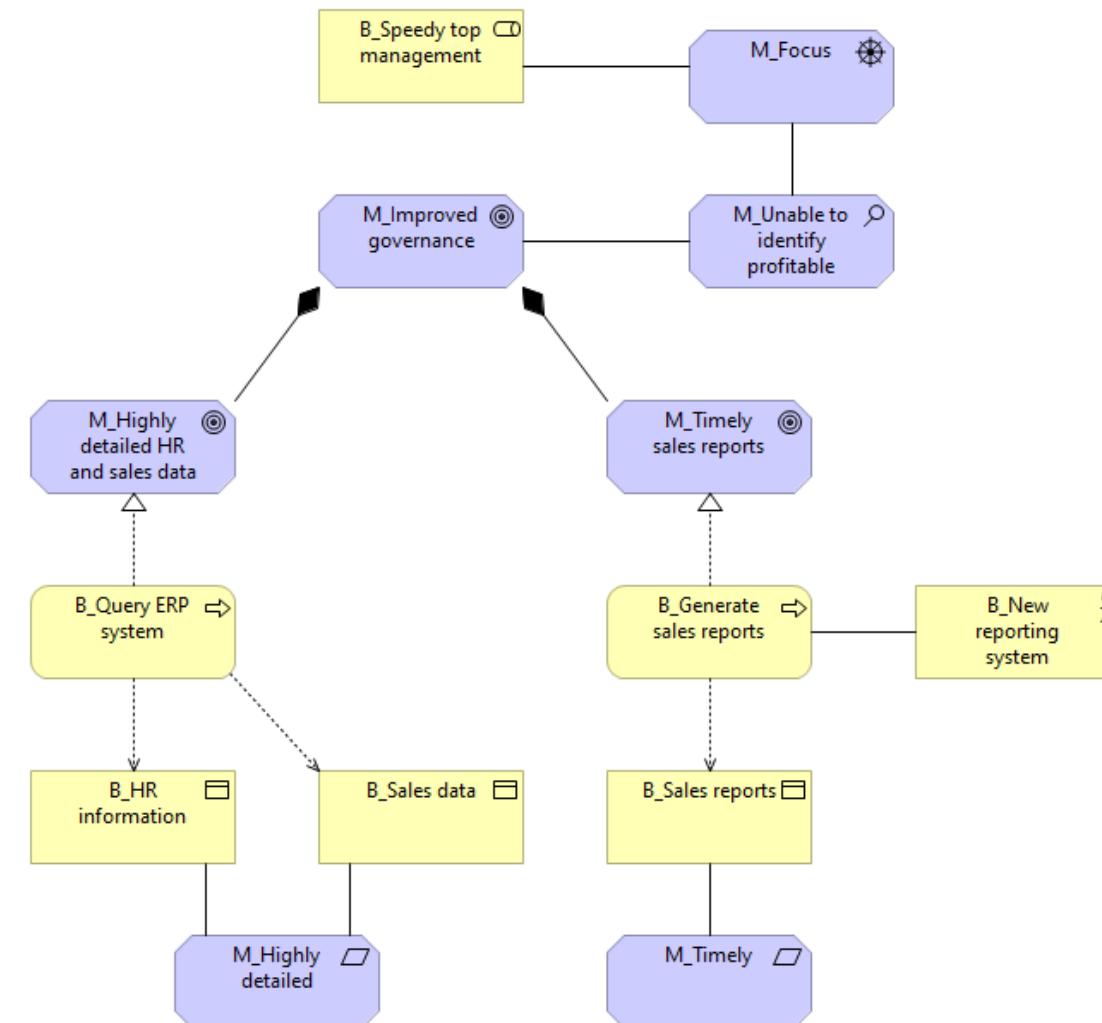
To implement the reporting system, a data warehouse (DW) based on Microsoft SQL Server 2022 will be used. The DW will run three components responsible for extracting sales data from the ERP database, managing analysis data, and generating sales reports.

The DW will run on-premise on a different node than the one running the ERP. Both nodes will share the same LAN.

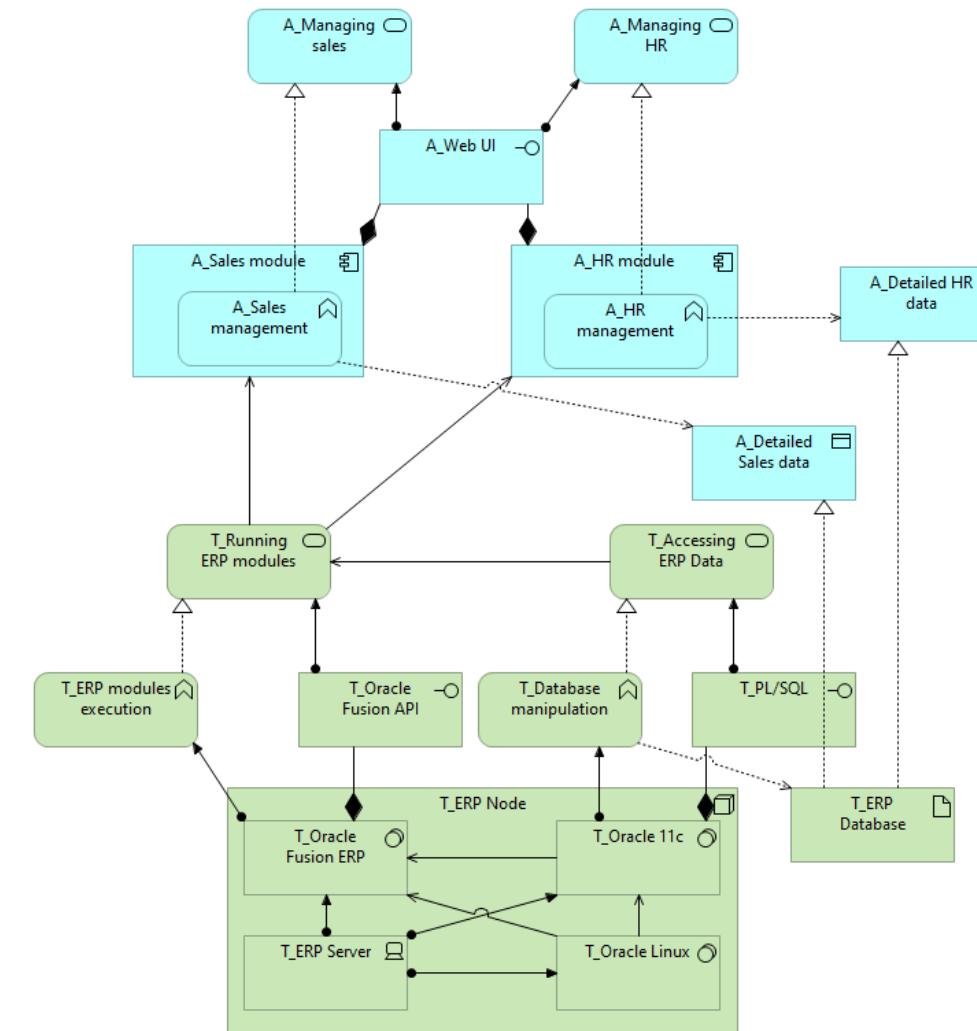
ArchiMate models representing the business goals and the existing ERP system are enclosed in the next slides.

Starting from these models, create a complete ArchiMate model that: 1) fully represents the business layer and links it to the application layer, 2) extends the application and technology layers by adding the elements required for the new reporting service.

Exercise 1 – Motivation



Exercise 1 – Existing ERP system



Exercise 2 – IC

IC is an insurance company which wants to offer a new insurance service for small objects (<2000\$) managed completely online for reliable customers.

To achieve this, a customer who wants his assets to be insured has to provide its credentials and photo of the asset and its details (serial number, purchase date) to IC. To ensure that the customer is reliable and the asset inexpensive, IC will then check the customers credentials and past history and estimate the asset's price. If the checks succeed, a proposal will be generated. Otherwise, the request will be rejected.

To support the new service, IC needs to develop a new application with the following functionalities: proposal generation, price estimation, customer reliability check.

The new application will be implemented as a Java EE component running on Apache Tomcat. It will also rely on an operational database deployed on Oracle MySQL. Both the database and the application will reside on a dedicated server, which is connected to the corporate LAN.

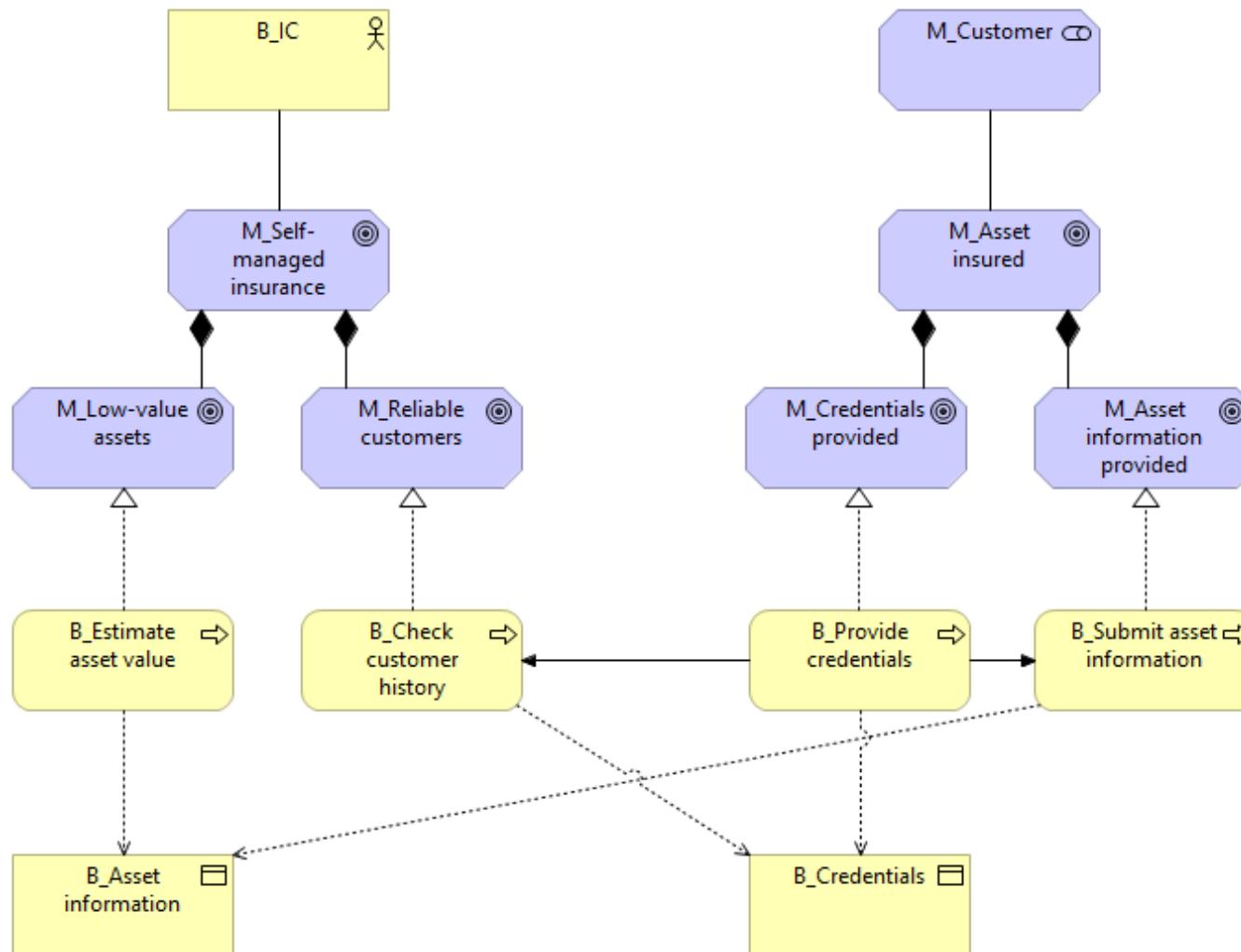
To estimate an item price, an external service will be used.

To check the customer reliability, the application needs data coming from the company's CRM, exposed by the CRM as a REST service and accessed through the corporate LAN.

The corporate LAN is separated from the public Internet by a firewall.

Model in ArchiMate the service provided by the company and its infrastructure.

Exercise 2 – Motivation



Exercise 3 – TEL

TEL is a telephony company interested in improving its customer experience. Currently, billing details are only accessible internally by TEL staff.

Data about telephony usage is stored on a Linux server running IBM DB2 UDB database, and it can only be accessed through SQL queries. Conversely, billing information is stored and handled by a legacy transactional CICS application running on an IBM mainframe.

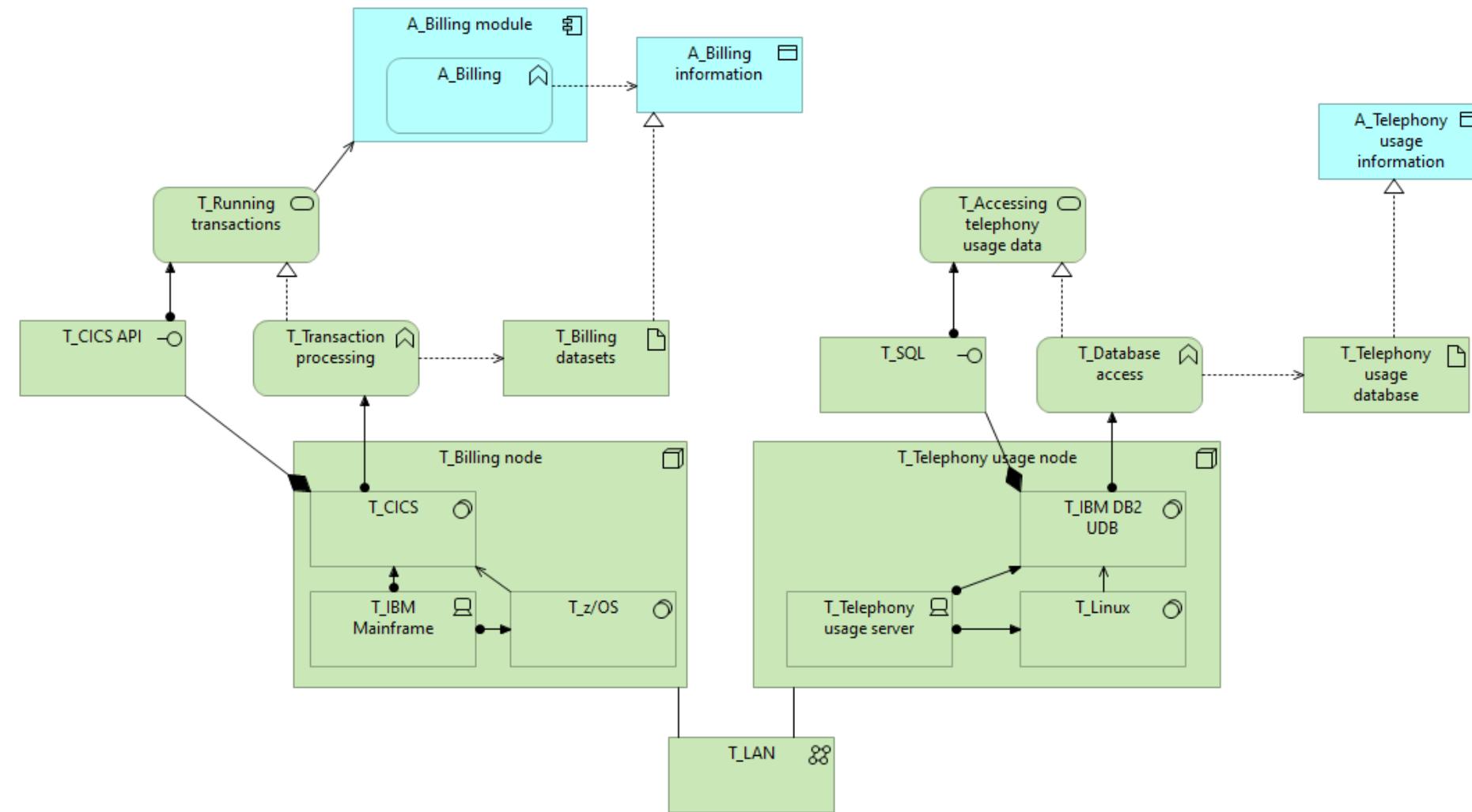
To improve its customer experience, TEL wants to develop a new web portal that can provide real-time billing details to users. In particular, the new portal will provide two main functionalities: inspect billing information and inspect usage information. The process enabled by the new portal will be organized as follows: the user logs in the portal, then he can select an item to inspect from the menu, and finally he can view the billing details in a dedicated page.

The new portal will be built for Microsoft SharePoint Online PaaS cloud service.

To access data from the existing infrastructure, a new node running Microsoft BizTalk Server 2020 middleware will also be introduced. BizTalk Server will offer a gateway service, making CICS applications and relational databases accessible through a standard REST interface.

An ArchiMate model of the existing system is enclosed in the next slide. Extend the model by adding the elements required for the new service.

Exercise 3 – Existing infrastructure

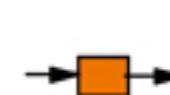
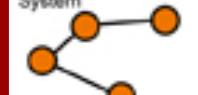
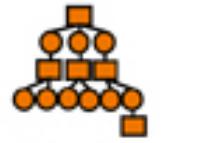
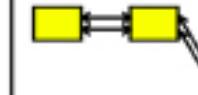
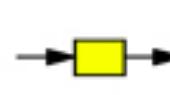
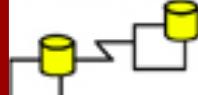
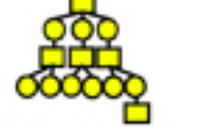
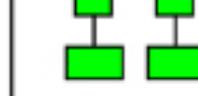
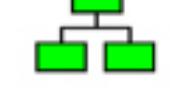
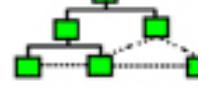
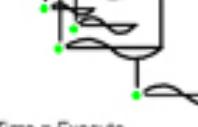
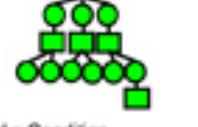


02291 System Integration

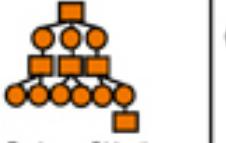
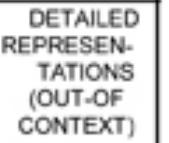
Behavioral Models with Petri Nets

© Giovanni Meroni

Slides based on previous versions by Prof. Pierluigi Plebani (Politecnico di Milano)

	DATA What	FUNCTION How	NETWORK Where	PEOPLE Who	TIME When	MOTIVATION Why	
SCOPE (CONTEXTUAL)	List of Things Important to the Business 	List of Processes the Business Performs 	List of Locations in which the Business Operates 	List of Organizations Important to the Business 	List of Events/Cycles Significant to the Business 	List of Business Goals/Strategies 	SCOPE (CONTEXTUAL)
Planner	ENTITY = Class of Business Thing	Process = Class of Business Process	Node = Major Business Location	People = Major Organization Unit	Time = Major Business Event/Cycle	Ends/Means = Major Business Goal/Strategy	Planner
BUSINESS MODEL (CONCEPTUAL)	e.g. Semantic Model 	e.g. Business Process Model 	e.g. Business Logistics System 	e.g. Work Flow Model 	e.g. Master Schedule 	e.g. Business Plan 	BUSINESS MODEL (CONCEPTUAL)
Owner	Ent = Business Entity Rein = Business Relationship	Proc. = Business Process I/O = Business Resources	Node = Business Location Link = Business Linkage	People = Organization Unit Work = Work Product	Time = Business Event Cycle = Business Cycle	End = Business Objective Means = Business Strategy	Owner
SYSTEM MODEL (LOGICAL)	e.g. Logical Data Model 	e.g. Application Architecture 	e.g. Distributed System Architecture 	e.g. Human Interface Architecture 	e.g. Processing Structure 	e.g. Business Rule Model 	SYSTEM MODEL (LOGICAL)
Designer	Ent = Data Entity Rein = Data Relationship	Proc. = Application Function I/O = User Views	Node = IS Function (Processor, Storage, etc) Link = Line Characteristics	People = Role Work = Deliverable	Time = System Event Cycle = Processing Cycle	End = Structural Assertion Means = Action Assertion	Designer
TECHNOLOGY MODEL (PHYSICAL)	e.g. Physical Data Model 	e.g. System Design 	e.g. Technology Architecture 	e.g. Presentation Architecture 	e.g. Control Structure 	e.g. Rule Design 	TECHNOLOGY MODEL (PHYSICAL)
Builder	Ent = Segment/Table/etc. Rein = Pointer/Key/etc.	Proc. = Computer Function I/O = Data Elements/Sets	Node = Hardware/Systems Software Link = Line Specifications	People = User Work = Screen Format	Time = Execute Cycle = Component Cycle	End = Condition Means = Action	Builder
DETAILED REPRESENTATIONS (OUT-OF-CONTEXT)	e.g. Data Definition 	e.g. Program 	e.g. Network Architecture 	e.g. Security Architecture 	e.g. Timing Definition 	e.g. Rule Specification 	DETAILED REPRESENTATIONS (OUT-OF-CONTEXT)
Sub-Contractor	Ent = Field Rein = Address	Proc. = Language Statement I/O = Control Block	Node = Address Link = Protocol	People = Identity Work = Job	Time = Interrupt Cycle = Machine Cycle	End = Sub-condition Means = Stop	Sub-Contractor
FUNCTIONING ENTERPRISE	e.g. DATA	e.g. FUNCTION	e.g. NETWORK	e.g. ORGANIZATION	e.g. SCHEDULE	e.g. STRATEGY	FUNCTIONING ENTERPRISE

© John A. Zachman, Zachman International

	DATA What	FUNCTION How	NETWORK Where	PEOPLE Who	TIME When	MOTIVATION Why	
SCOPE (CONTEXTUAL)		Behavior			List of Events/Cycles Significant to the Business 	List of Business Goals/Strategies 	SCOPE (CONTEXTUAL)
Planner					Time = Major Business Event/Cycle e.g. Master Schedule	Ends/Means = Major Business Goal/Strategy 	Planner
BUSINESS MODEL (CONCEPTUAL)					Time = Business Event Cycle = Business Cycle e.g. Processing Structure	End = Business Objective Means = Business Strategy 	BUSINESS MODEL (CONCEPTUAL)
Owner					Time = System Event Cycle = Processing Cycle e.g. Control Structure	End = Structural Assertion Means = Action Assertion 	Owner
SYSTEM MODEL (LOGICAL)					Time = Execute Cycle = Component Cycle e.g. Rule Specification	End = Condition Means = Action 	SYSTEM MODEL (LOGICAL)
Designer					Time = Interrupt Cycle = Machine Cycle e.g. Sub-condition	End = Sub-condition Means = Stop 	Designer
TECHNOLOGY MODEL (PHYSICAL)					Time = Sub-condition End = Sub-condition Means = Stop	End = Sub-condition Means = Stop 	TECHNOLOGY MODEL (PHYSICAL)
Builder							Builder
DETAILED REPRESENTATIONS (OUT-OF-CONTEXT)	e.g. Data Definition 	e.g. Program 	e.g. Network Architecture 	e.g. Security Architecture 	e.g. Timing Definition 	e.g. Rule Specification 	DETAILED REPRESENTATIONS (OUT-OF-CONTEXT)
Sub-Contractor	Ent = Field Reln = Address	Proc = Language Statement I/O = Control Block	Node = Address Link = Protocol	People = Identity Work = Job	Time = Interrupt Cycle = Machine Cycle	End = Sub-condition Means = Stop	Sub-Contractor
FUNCTIONING ENTERPRISE	e.g. DATA	e.g. FUNCTION	e.g. NETWORK	e.g. ORGANIZATION	e.g. SCHEDULE	e.g. STRATEGY	FUNCTIONING ENTERPRISE

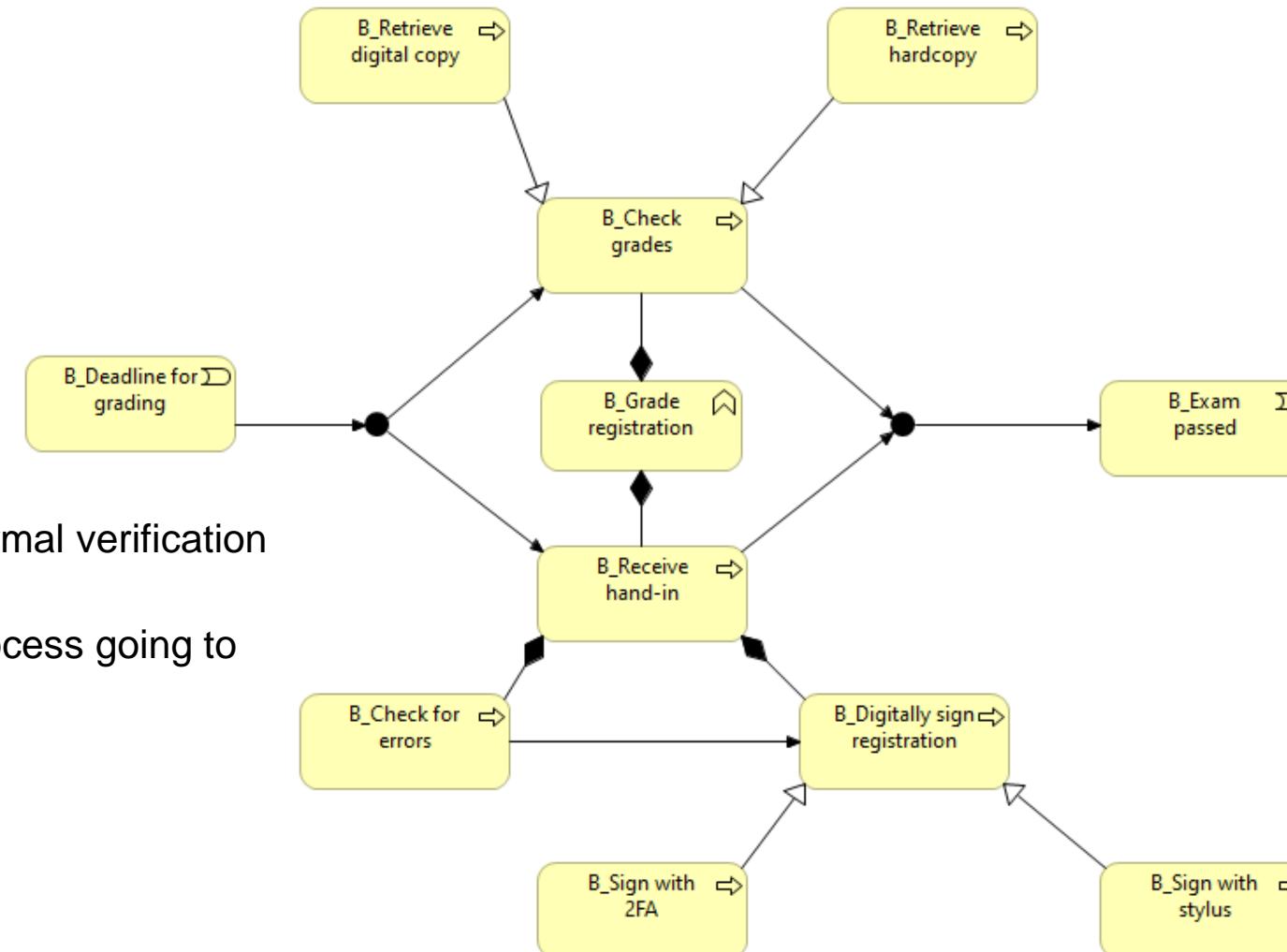
Business

Application

Technology

© John A. Zachman, Zachman International

Behavioral models in ArchiMate



ArchiMate does not provide formal verification techniques:

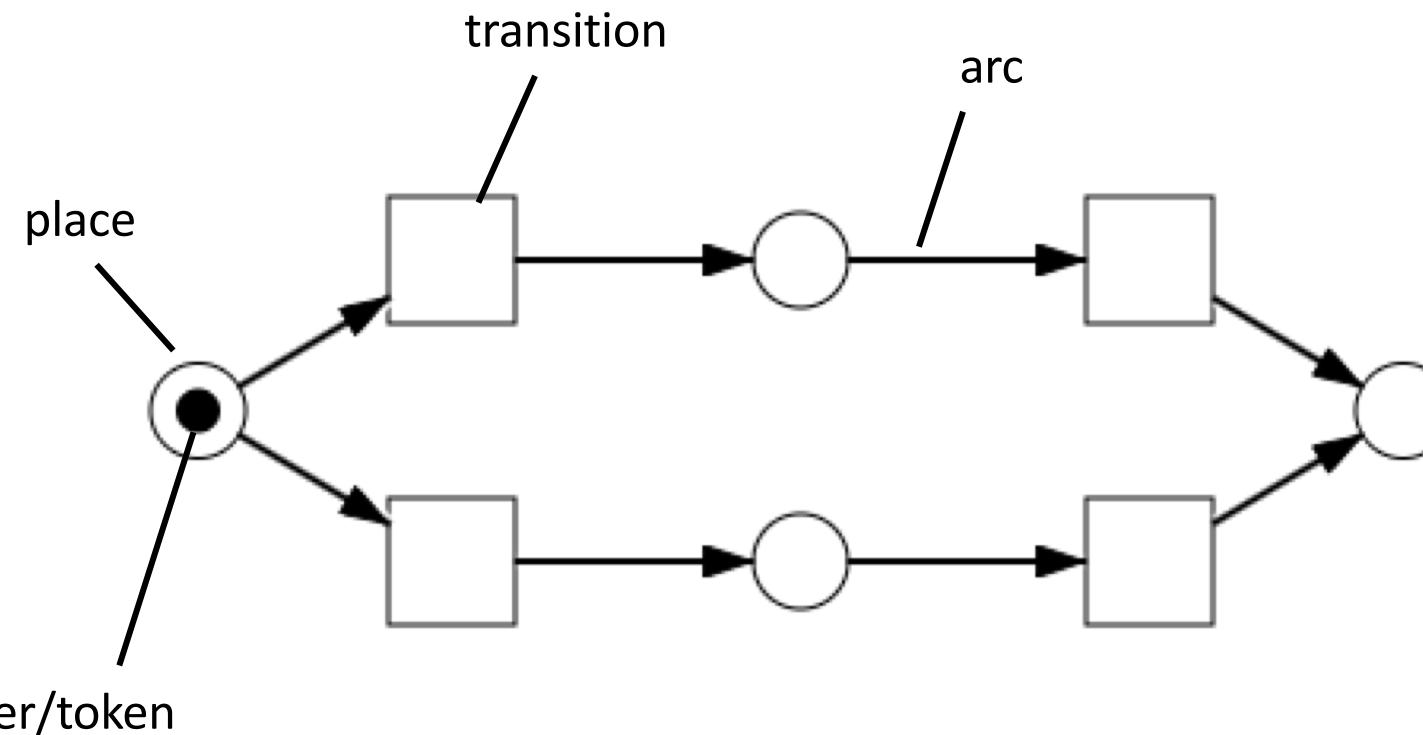
- Is the grade registration process going to terminate?
- Can all tasks be executed?

Petri Nets

Petri Nets (a very short introduction)

- Developed in the 60s by Carl Adam Petri
- Allows the modeling of concurrent, distributed, asynchronous systems
- Many variants exist to model additional aspects of a system
 - Temporal PN
 - Colored PN
 - Stochastic PN
 - ...
- Here we introduce, intuitively and formally, the basic elements
- For an exhaustive definition please refer to one of the thousands books available. We suggest:
 - Van der Aalst, Modeling Business Process: A Petri-Net Approach, MIT press, 2011

Petri Nets: an informal introduction

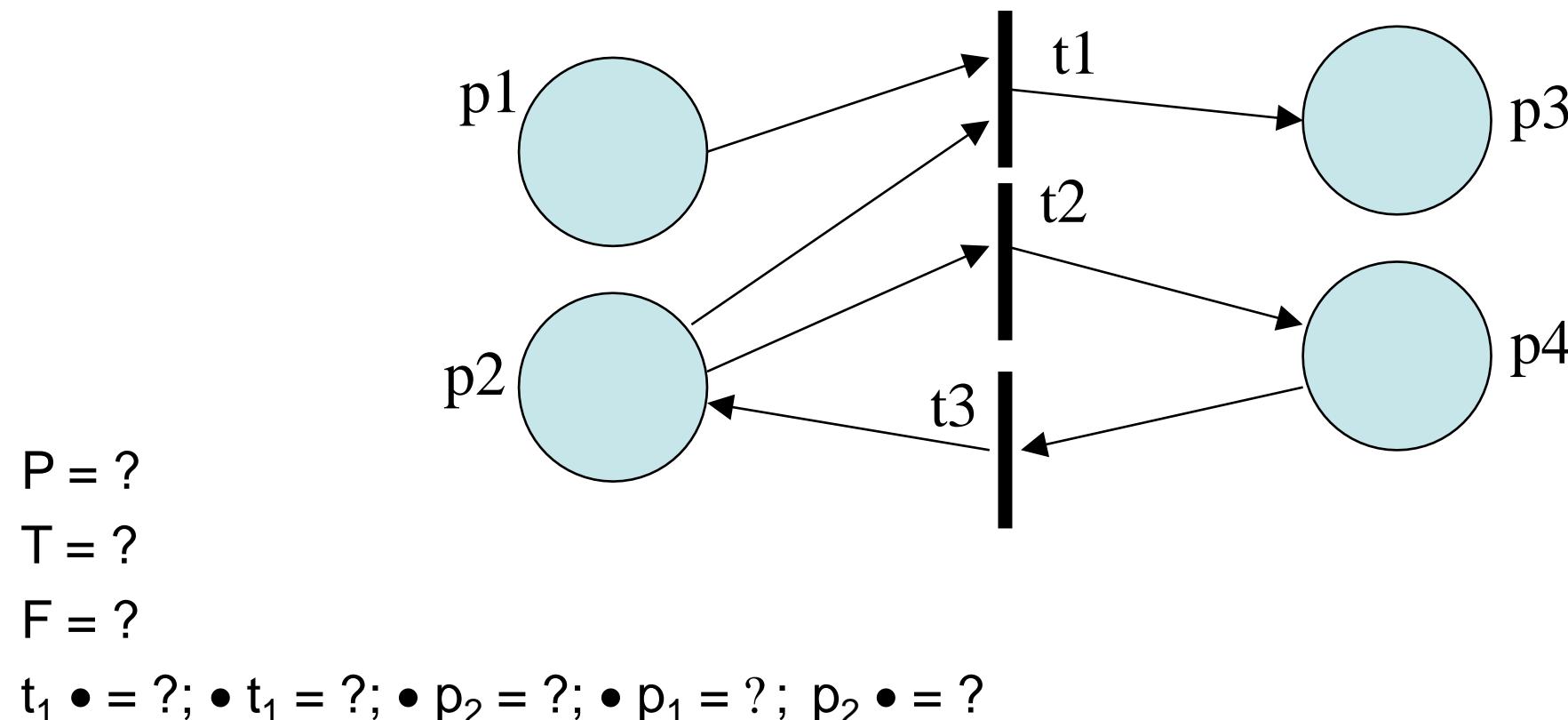


- Bipartite oriented graph connecting places and transitions
- Transitions and places define the structure of the net
- Tokens define the behaviour of the net

Petri Nets: a formal definition of the structure

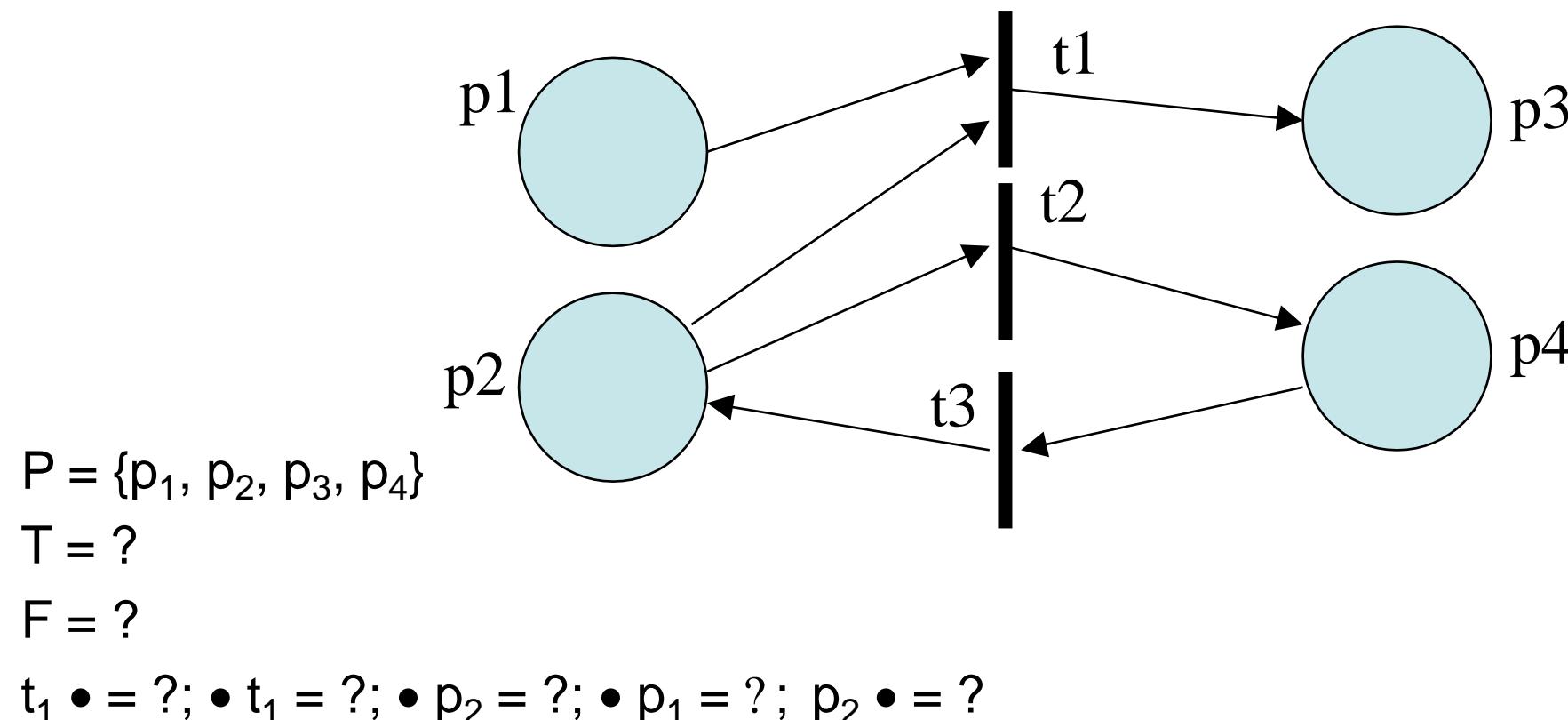
- A Petri Net \mathbf{N} is a triple (P, T, F) , where
 - P is a finite set of places
 - T is a finite set of transitions
 - $F \subset (P \times T \cup T \times P)$ is a flow relation
- Given a Petri Net $\mathbf{N} = (P, T, F)$
 - *Pre-sets* can be defined for places and transitions
 - Preset for a place p is defined as $\bullet p = \{t \in T \mid (t, p) \in F\}$
 - Preset for a transition t is defined as $\bullet t = \{p \in P \mid (p, t) \in F\}$
 - Similarly, places and transitions *post-sets* are defined
 - Post-set for a place p is defined as $p \bullet = \{t \in T \mid (p, t) \in F\}$
 - Post-set for a transition t is defined as $t \bullet = \{p \in P \mid (t, p) \in F\}$

Example



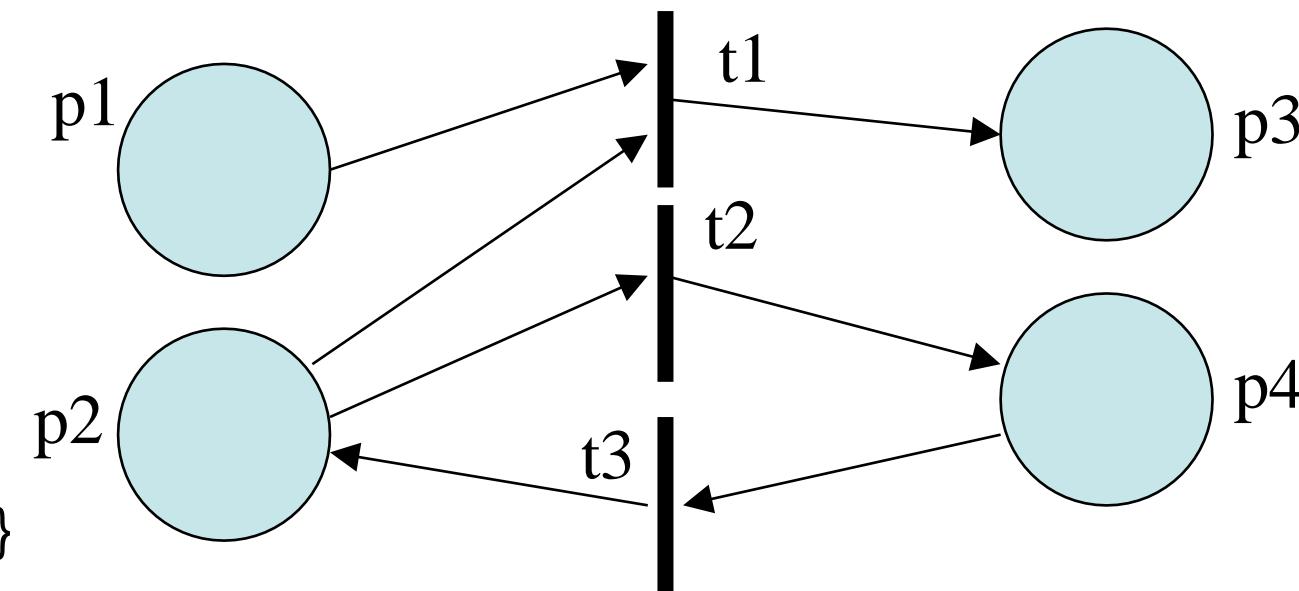
Source: A.H.M. ter Hofstede, W. van der Aalst, M. Adams, N. Russel, Modern Business Process Automation: YAWL and its support environment, Springer, 2009

Example



Source: A.H.M. ter Hofstede, W. van der Aalst, M. Adams, N. Russel, Modern Business Process Automation: YAWL and its support environment, Springer, 2009

Example



$$P = \{p_1, p_2, p_3, p_4\}$$

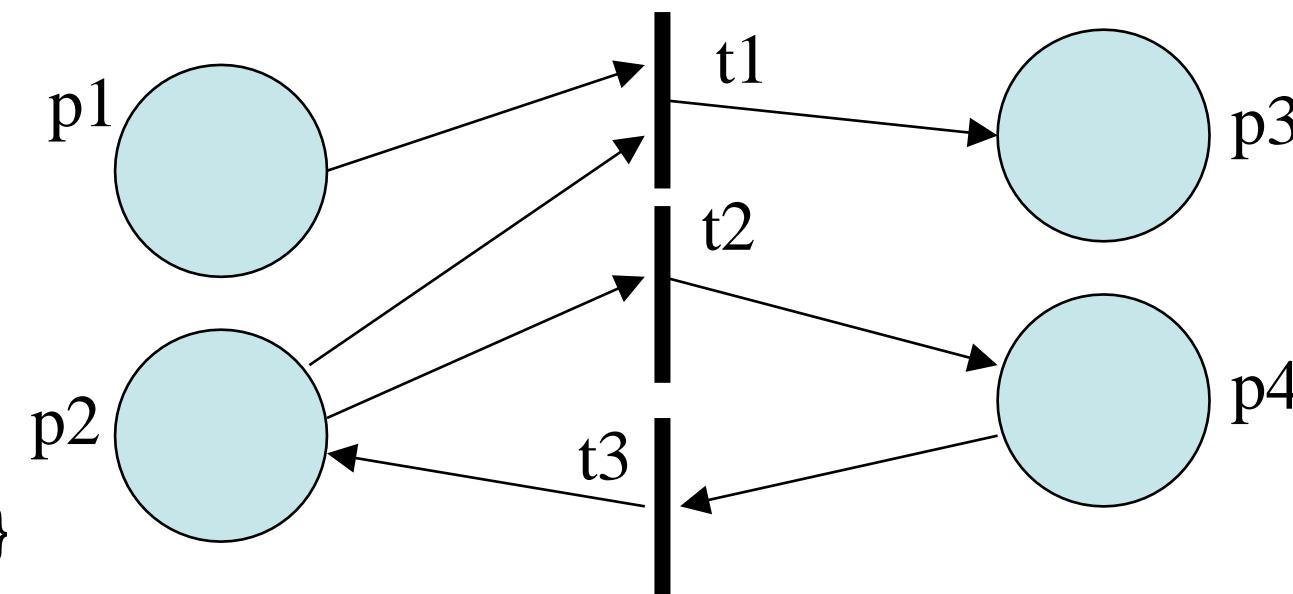
$$T = \{t_1, t_2, t_3\}$$

$$F = ?$$

$$t_1 \bullet = ?; \bullet t_1 = ?; \bullet p_2 = ?; \bullet p_1 = ?; p_2 \bullet = ?$$

Source: A.H.M. ter Hofstede, W. van der Aalst, M. Adams, N. Russel, Modern Business Process Automation: YAWL and its support environment, Springer, 2009

Example



$$P = \{p_1, p_2, p_3, p_4\}$$

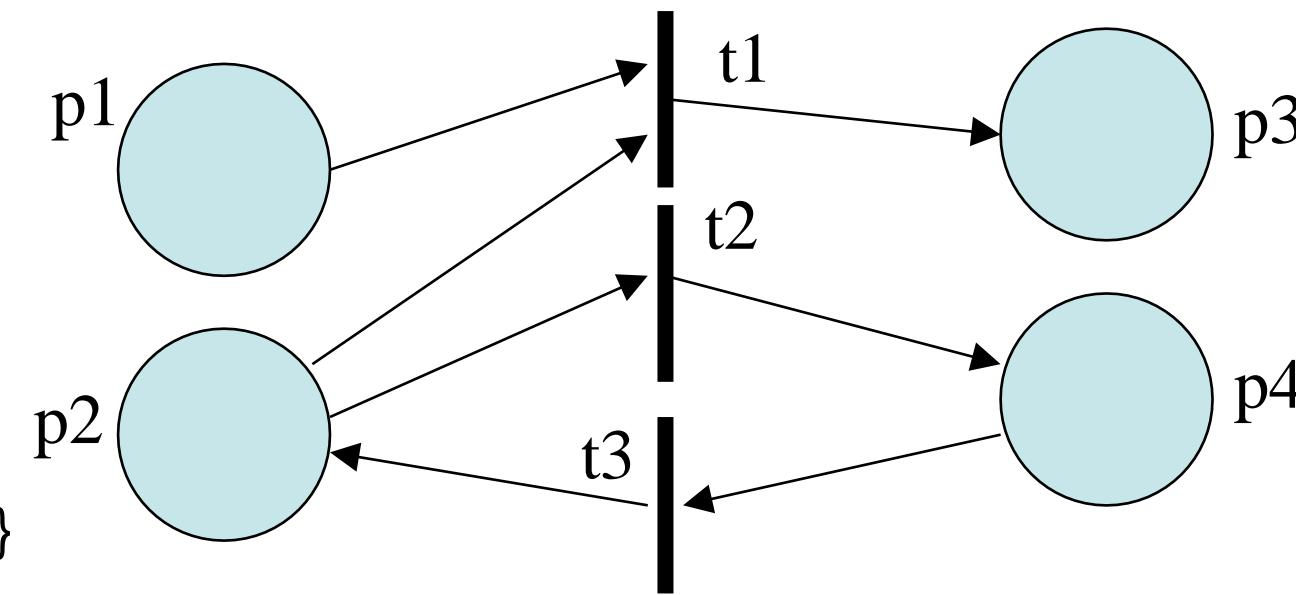
$$T = \{t_1, t_2, t_3\}$$

$$F = \{(p_1, t_1), (p_2, t_1), (t_1, p_3), (p_2, t_2), (t_2, p_4), (p_4, t_3), (t_3, p_2)\}$$

$$t_1 \bullet = ?; \bullet t_1 = ?; \bullet p_2 = ?; \bullet p_1 = ?; p_2 \bullet = ?$$

Source: A.H.M. ter Hofstede, W. van der Aalst, M. Adams, N. Russel, Modern Business Process Automation: YAWL and its support environment, Springer, 2009

Example



$$P = \{p_1, p_2, p_3, p_4\}$$

$$T = \{t_1, t_2, t_3\}$$

$$F = \{(p_1, t_1), (p_2, t_1), (t_1, p_3), (p_2, t_2), (t_2, p_4), (p_4, t_3), (t_3, p_2)\}$$

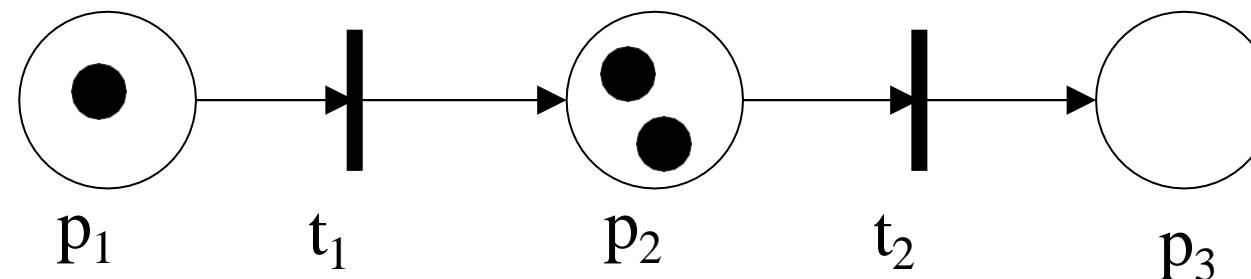
$$t_1 \bullet = \{p_3\}; \bullet t_1 = \{p_1, p_2\}; \bullet p_2 = \{t_3\}; \bullet p_1 = \emptyset; p_2 \bullet = \{t_1, t_2\}$$

Source: A.H.M. ter Hofstede, W. van der Aalst, M. Adams, N. Russel, Modern Business Process Automation: YAWL and its support environment, Springer, 2009

Petri Nets: a formal definition of the behaviour

- Marking is the key element describing the behaviour of a Petri Net
- Marking is represented by tokens and determines a state of the Petri Net
- Given a Petri Net $N = (P, T, F)$
 - $m: P \rightarrow N$
 - Marking assigns to each place $p \in P$ a number of tokens $m(p)$
 - The set $M = \{\forall p \in P \mid m(p)\}$
- For a Petri net an initial marking M_0 needs to be specified

Example

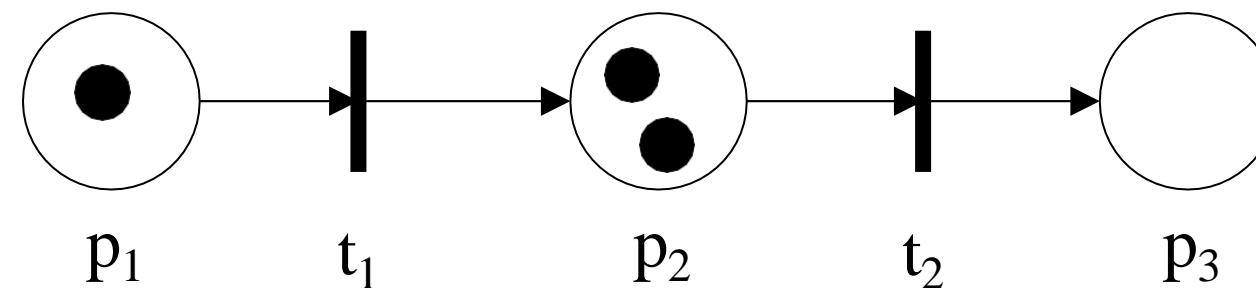


- The marking below is formally captured by:
 - $\{(p_1, 1), (p_2, 2), (p_3, 0)\}$
- Alternative notation
 - $p_1 + 2p_2$.

Source: A.H.M. ter Hofstede, W. van der Aalst, M. Adams, N. Russel, Modern Business Process Automation: YAWL and its support environment, Springer, 2009

Enabled transitions

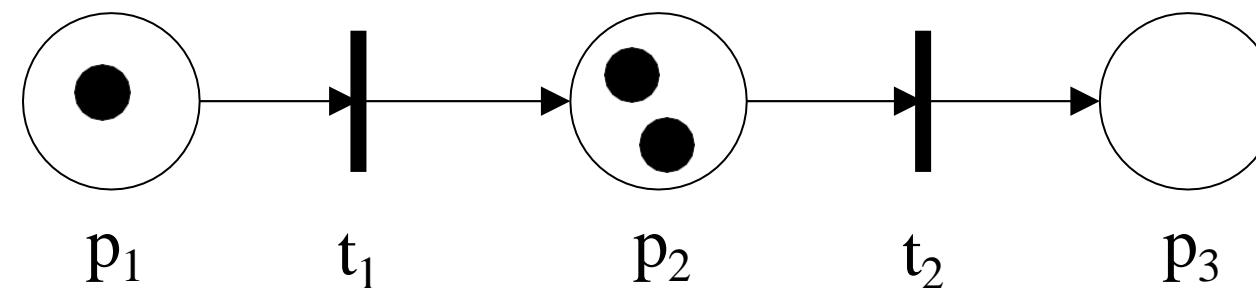
- Given a Petri Net $N = (P, T, F)$ a transition $t \in T$ is enabled in a marking M iff $\forall p \in \bullet t, m(p) > 0$



- Is t_1 enabled?
- Is t_2 enabled?

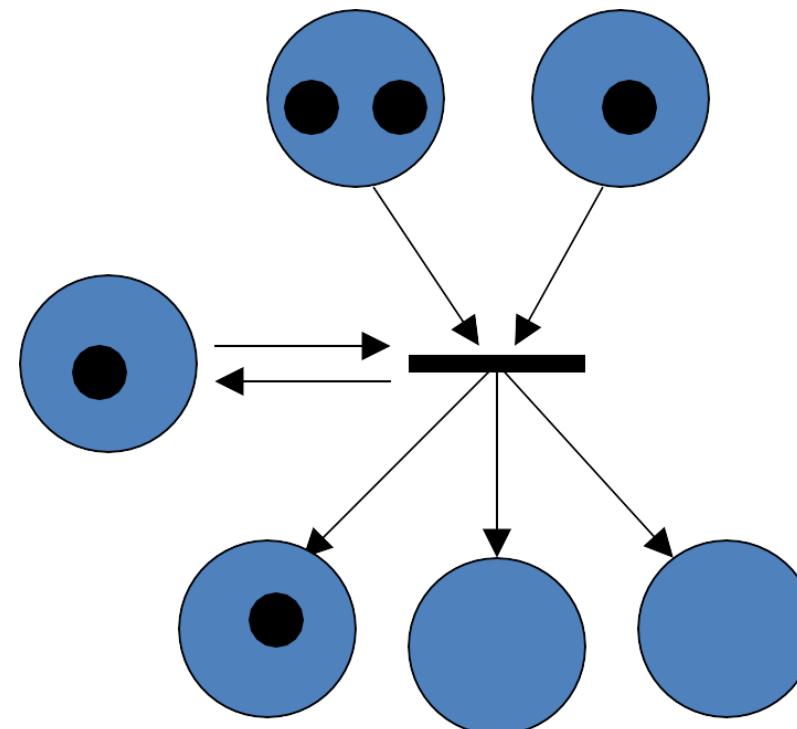
Enabled transitions

- Given a Petri Net $N = (P, T, F)$ a transition $t \in T$ is enabled in a marking M iff $\forall p \in \bullet t, m(p) > 0$



- Is t_1 enabled? Yes! $m(p_1) = 1 > 0$
- Is t_2 enabled? Yes! $m(p_2) = 2 > 0$

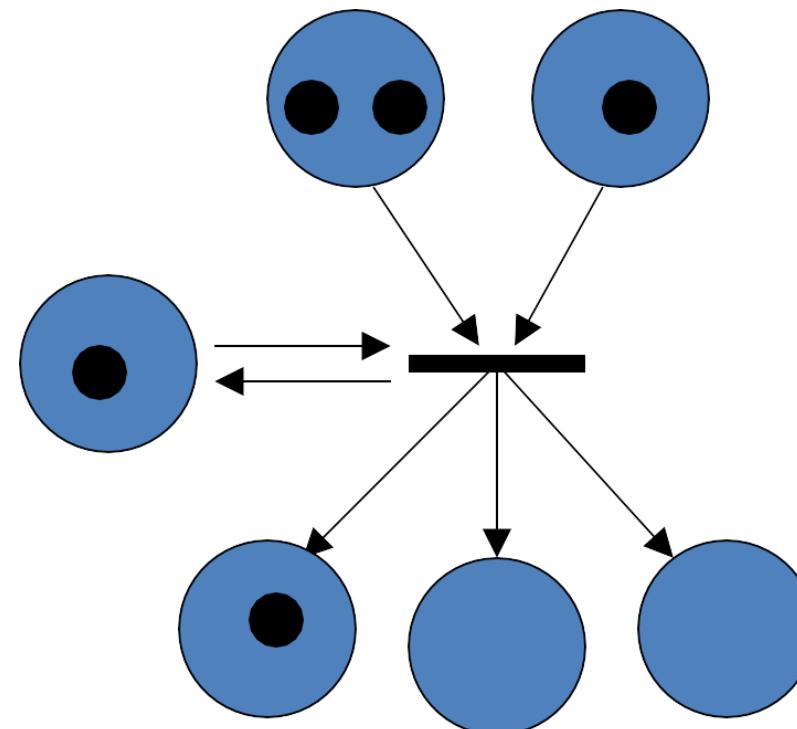
Example



Is the transition enabled?

Source: A.H.M. ter Hofstede, W. van der Aalst, M. Adams, N. Russel, Modern Business Process Automation: YAWL and its support environment, Springer, 2009

Example



Is the transition enabled?

Yes! We have a token in each place belonging to the preset

Source: A.H.M. ter Hofstede, W. van der Aalst, M. Adams, N. Russel, Modern Business Process Automation: YAWL and its support environment, Springer, 2009

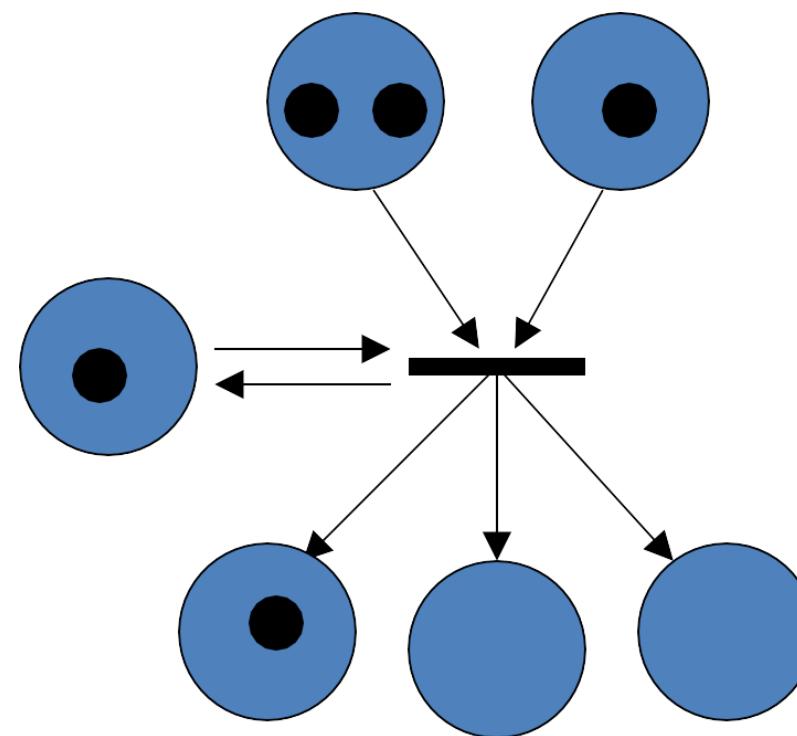
Firing transition

- Given a marking M of a Petri Net $N = (P, T, F)$, any enabled transition may fire
- When transition fires:
 - a token is removed from each of the input places
 - a token is produced for each of the output places
- To better define the transition firing we firstly introduce the weight function w
 - $w: (P \times T) \cup (T \times P) \rightarrow N$
 - $w(x,y) \neq 0$ if $(x,y) \in F$
 - $w(x,y) = 0$ if $(x,y) \notin F$
- In case range of w is $[0,1]$ then the Petri Net is named *ordinary Petri Net*

Firing transition

- We can formally define the firing transition as follows:
- Given a Petri Net $N = (P, T, F)$
 - the current marking M and the weight function w , a transition $t \in T$ can fire iif t is enabled in M
 - the firing of t yields to a new marking M' where
$$\forall p \in P, m'(p) = m(p) - w(p,t) + w(t,p)$$

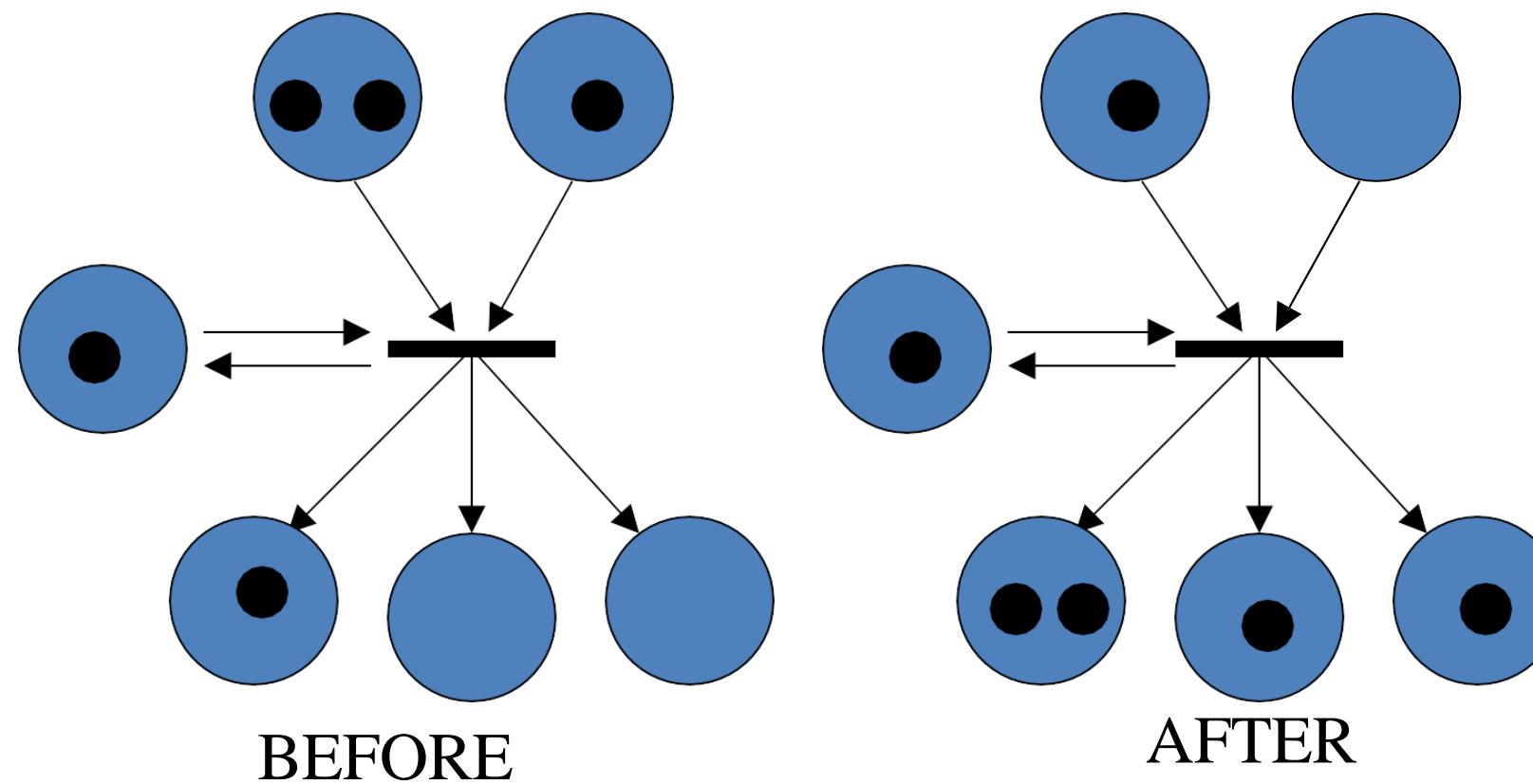
Example



BEFORE

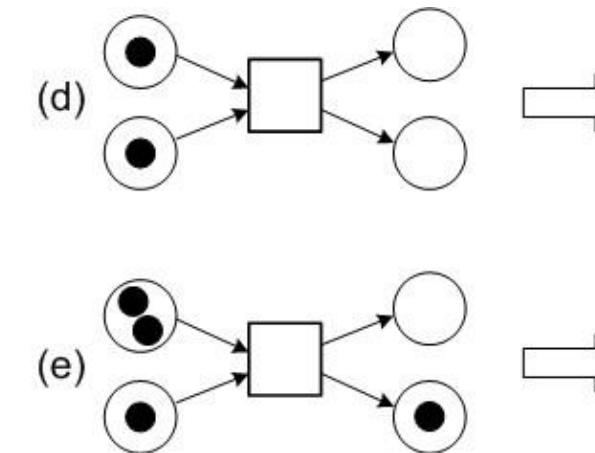
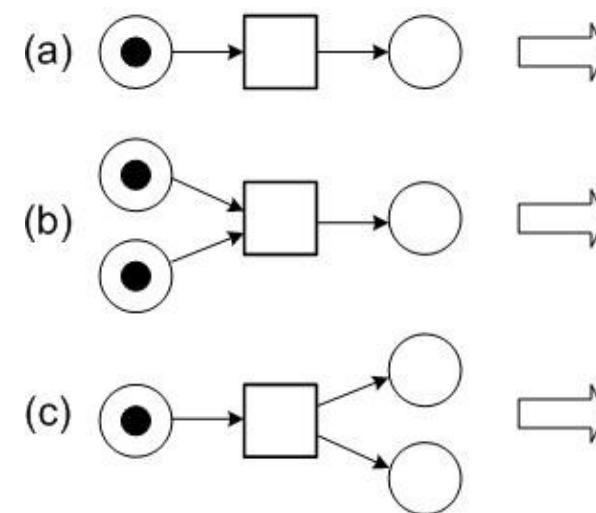
Source: A.H.M. ter Hofstede, W. van der Aalst, M. Adams, N. Russel, Modern Business Process Automation: YAWL and its support environment, Springer, 2009

Example

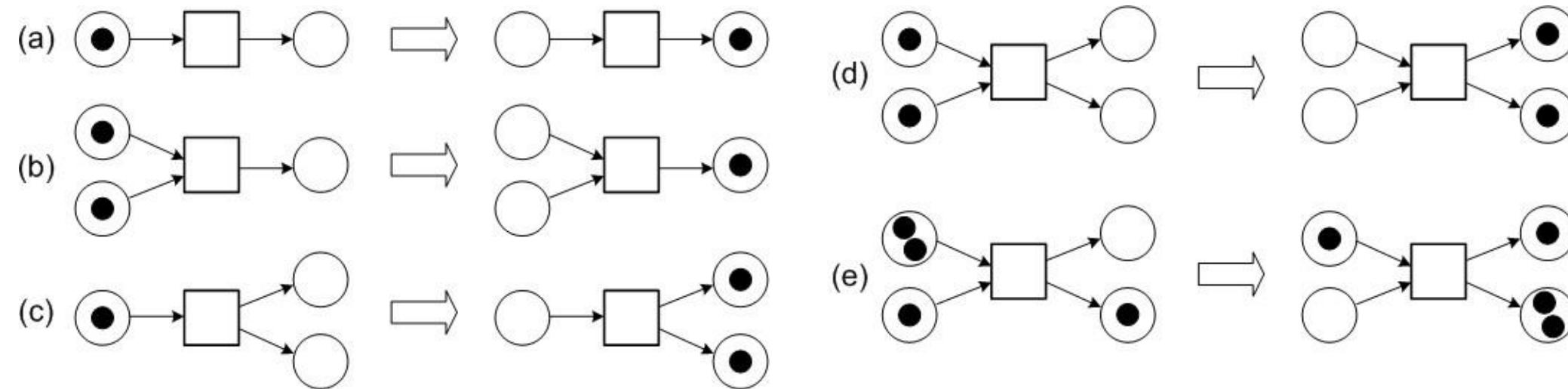


Source: A.H.M. ter Hofstede, W. van der Aalst, M. Adams, N. Russel, Modern Business Process Automation: YAWL and its support environment, Springer, 2009

Examples



Examples

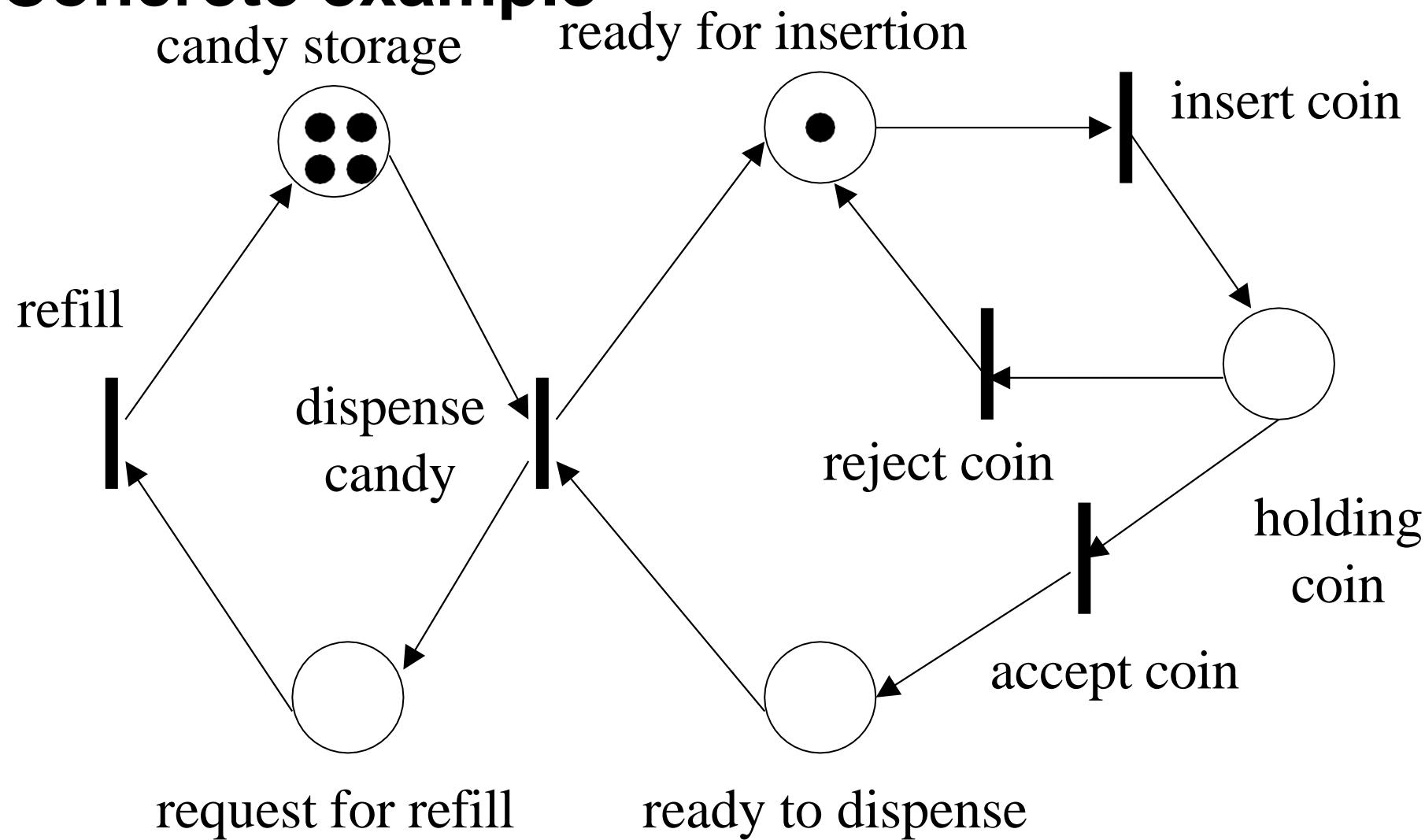


Concrete example

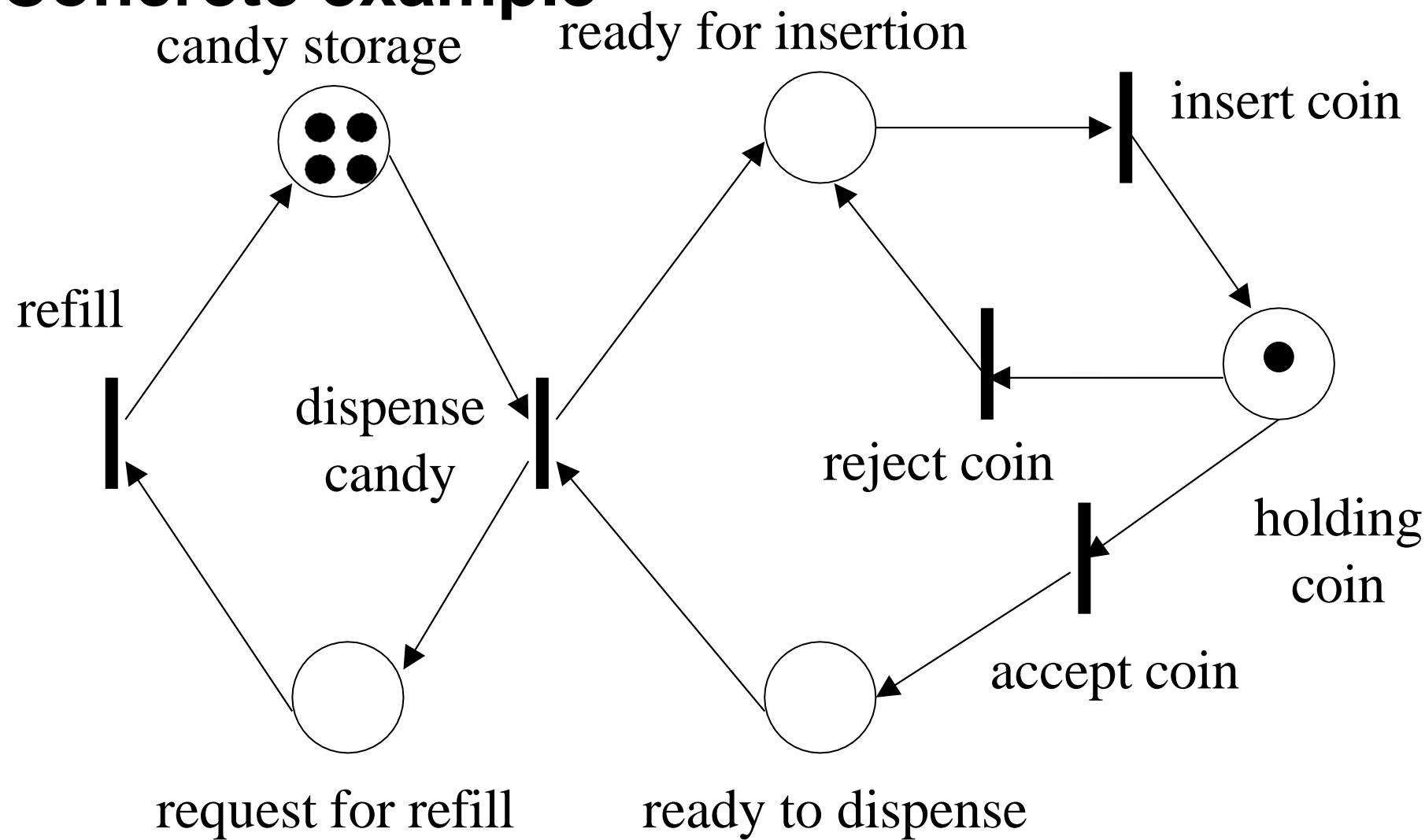


[This Photo](#) by Unknown Author is licensed under [CC BY-NC](#)

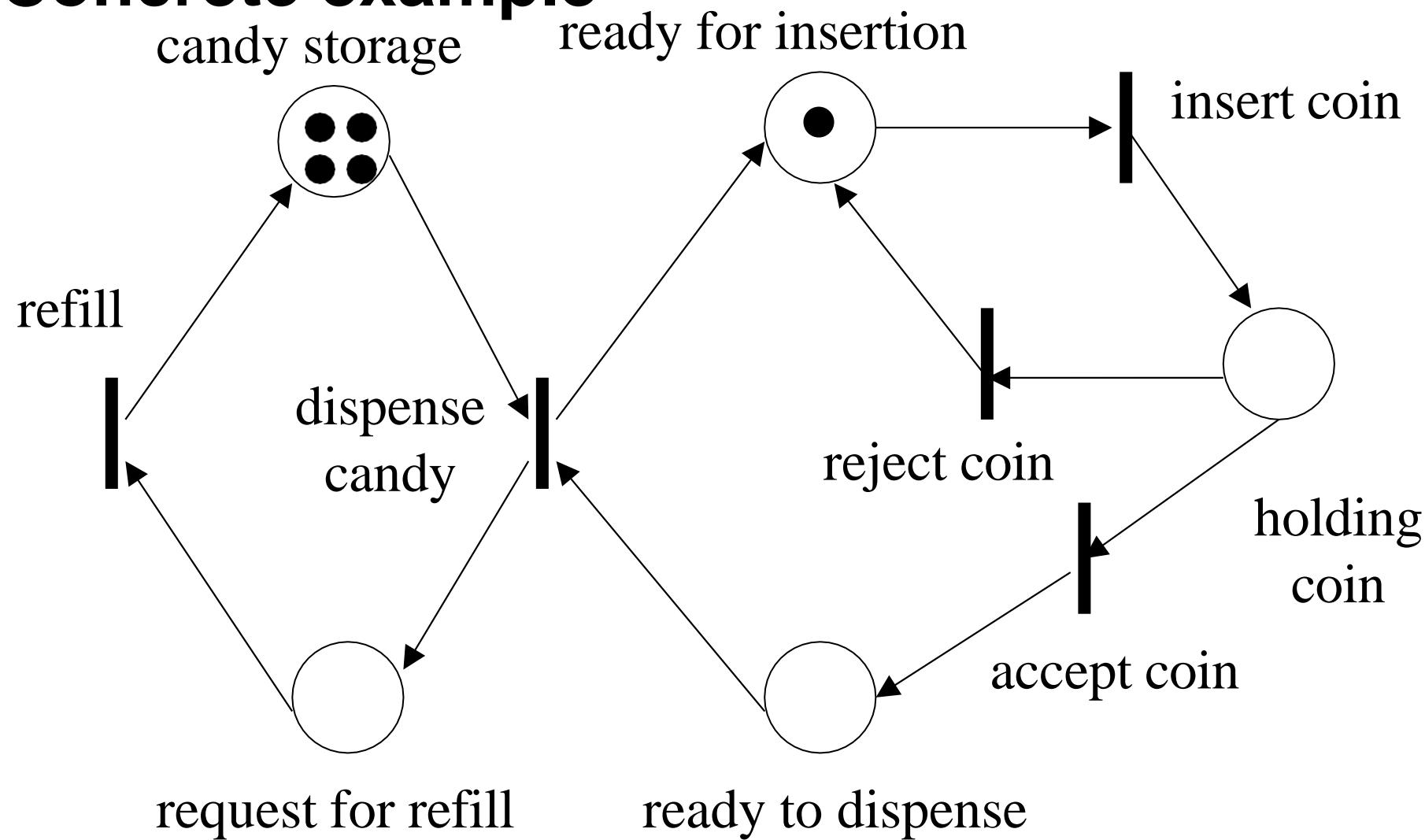
Concrete example



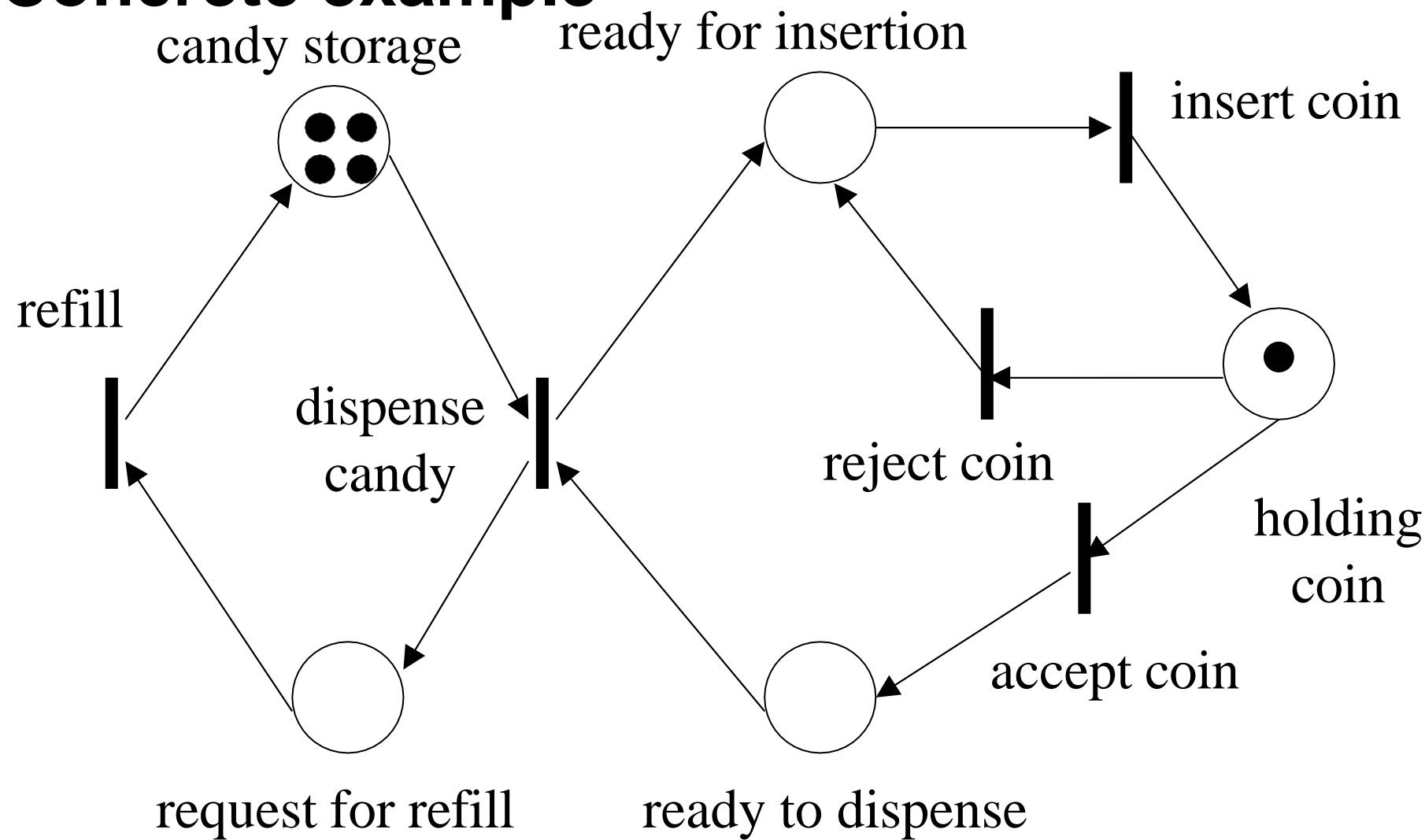
Concrete example



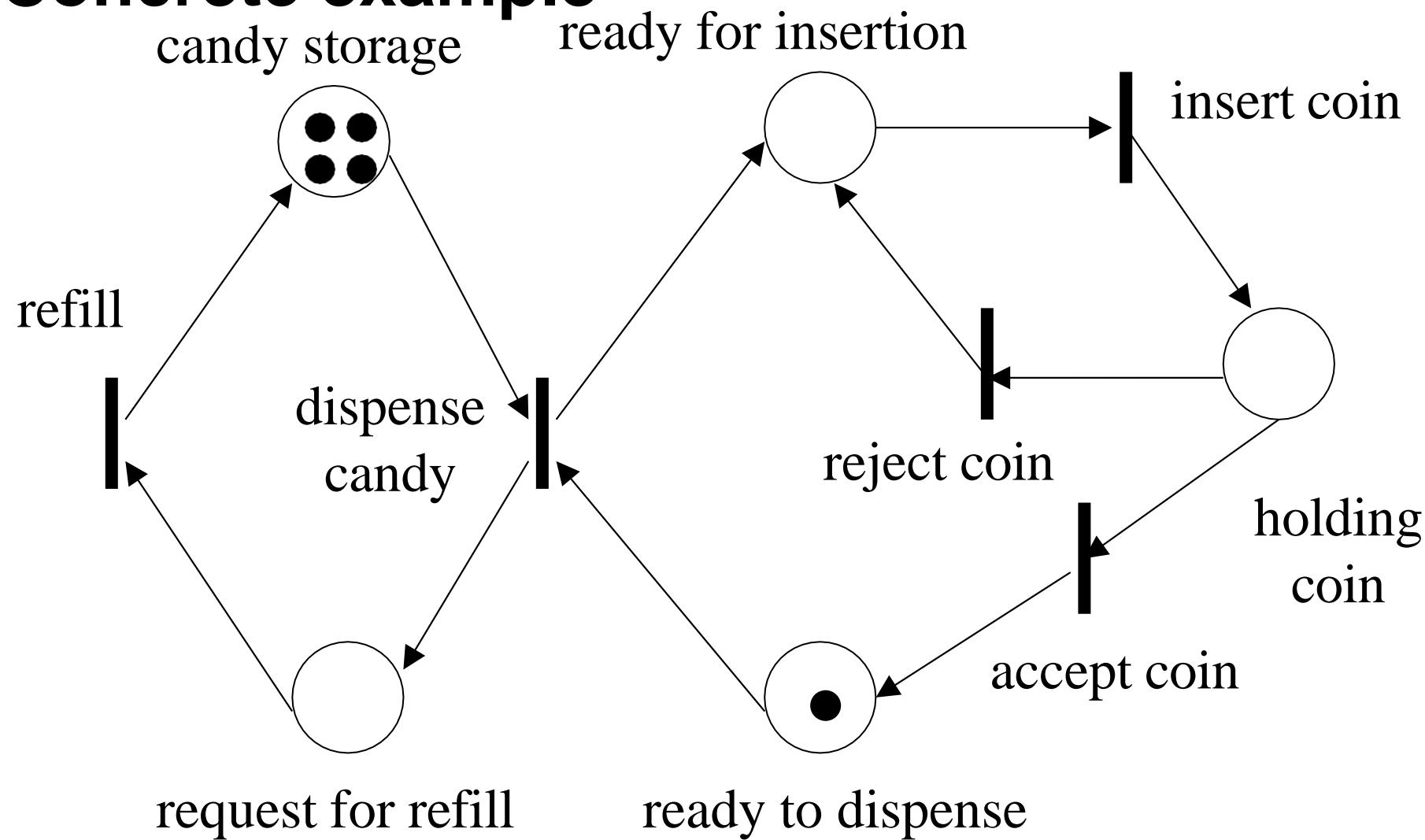
Concrete example



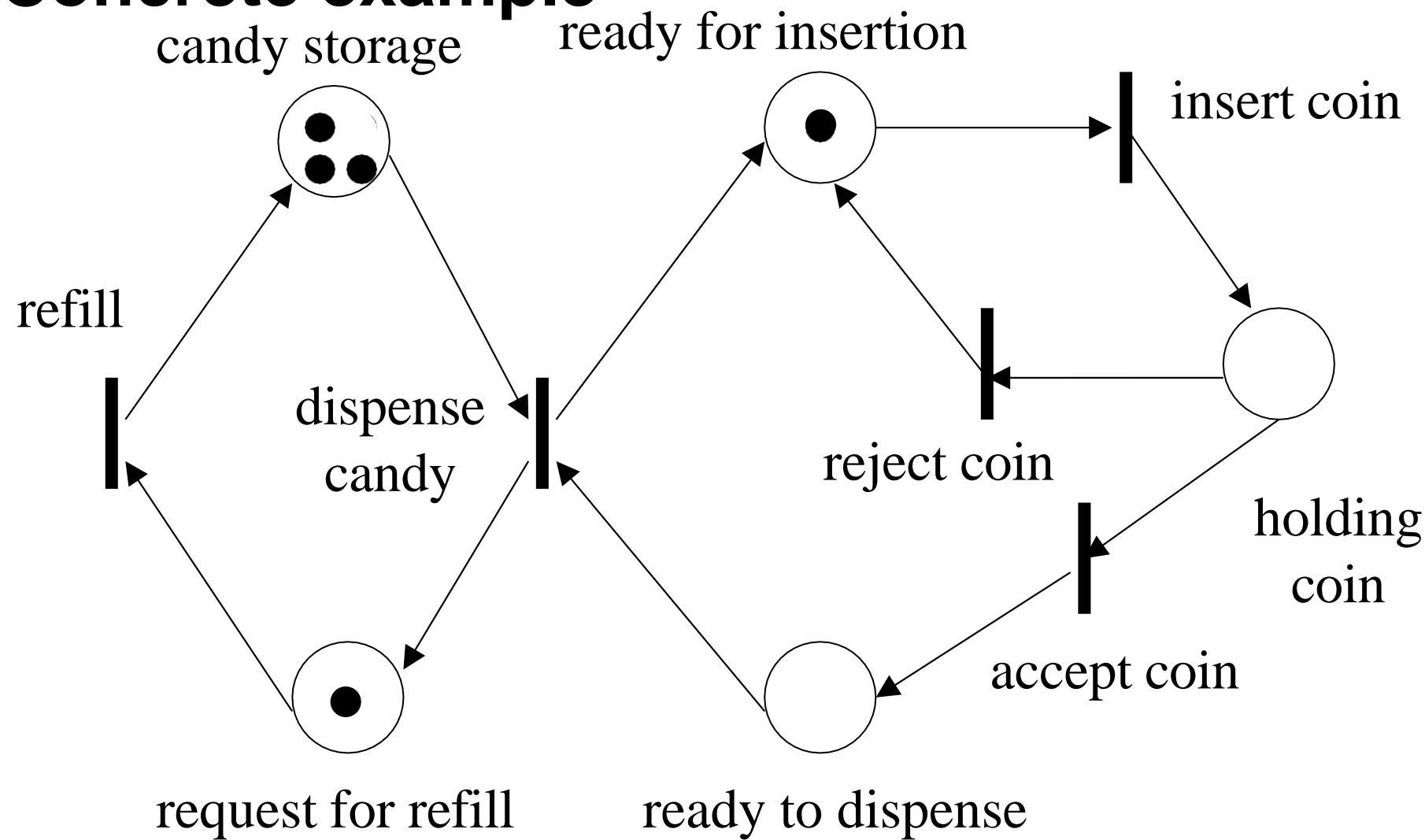
Concrete example



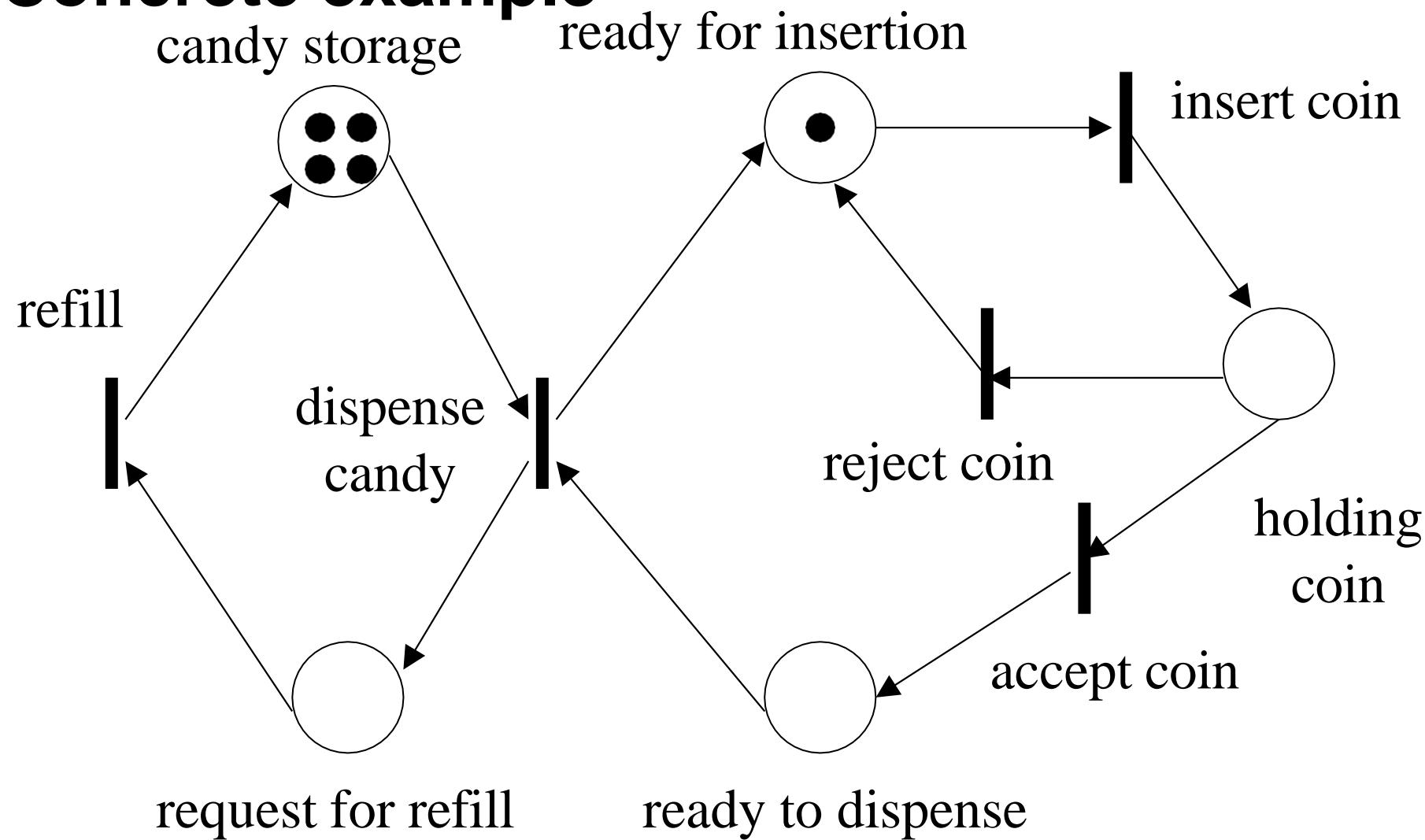
Concrete example



Concrete example



Concrete example



Nondeterminism

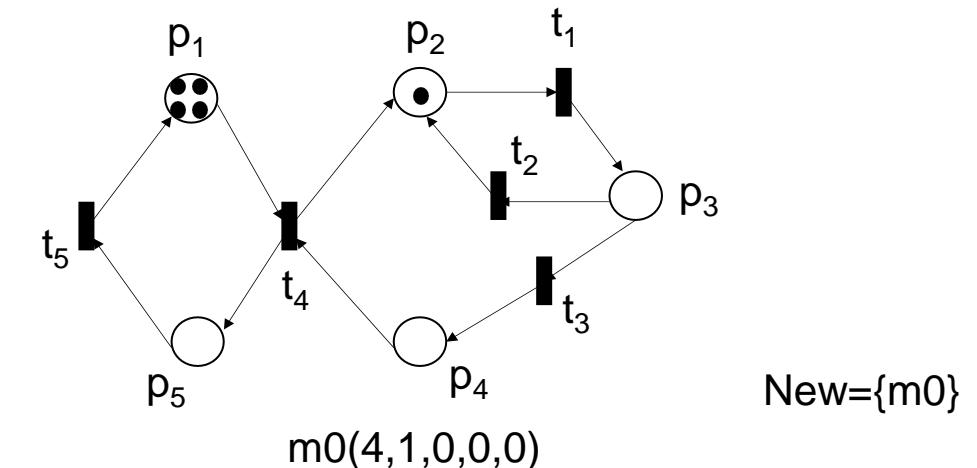
- For a given marking, more than one transitions could be enabled
- If there are multiple enabled transitions, any one of them may fire
 - for execution purposes, it is assumed that **they cannot fire simultaneously.**
- It is assumed that the firing of a transition is an atomic action that occurs instantaneously and cannot be interrupted

Reachable marking

- Given a Petri Net (P, T, F) and a state M_1 :
 - $M_1 [t] M_2$: a transition t is enables in state M_1 and firing t in M_1 results in state M_2
 - $M_1 \triangleright M_2$: there is a transition t such that $M_1 [t] M_2$
 - $M_1 [\sigma] M_n$: the firing sequence $\sigma=t_1, t_2, \dots, t_n$ leads from state M_1 to state M_n , i.e., $M_1 [t_1] M_2 [t_2] M_2 \dots M_{n-1} [t_n] M_n$
- A state M_n is called reachable from M_1 if and only if there is a firing sequence $\sigma=t_1, t_2, \dots, t_n$ such that $M_1 [\sigma] M_n$
- An empty firing sequence is also allowed $M_1 [\varepsilon] M_1$

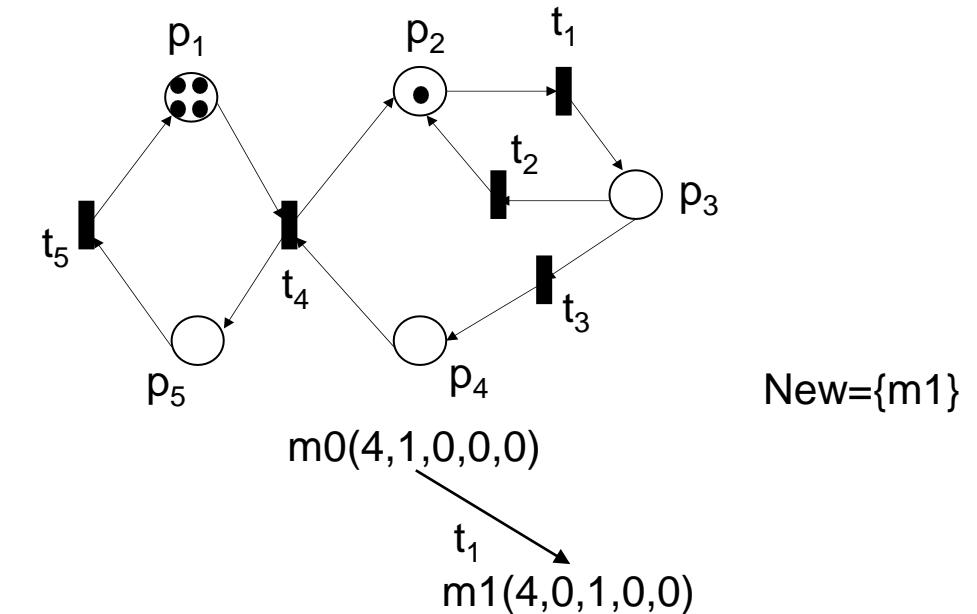
Reachability graph

1. Label the initial marking m_0 as the root and tag it "new".
2. While "new" markings exists, do the following:
 - a) Select a new marking m .
 - b) If no transitions are enabled at m , tag m "dead-end".
 - c) While there exist enabled transitions at m , do the following for each enabled transition t at m :
 - I. Obtain the marking m' that results from firing t at m .
 - II. If m' does not appear in the graph add m' and tag it "new".
 - III. Draw an arc with label t from m to m' (if not already present).
3. Output the graph



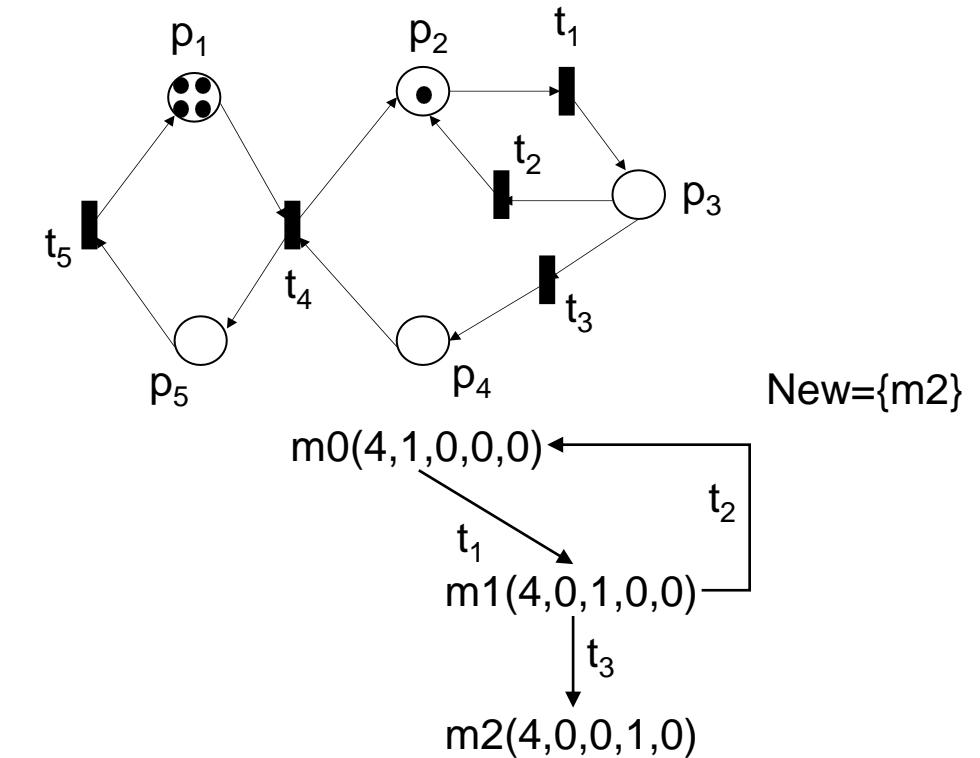
Reachability graph

1. Label the initial marking m_0 as the root and tag it "new".
2. **While "new" markings exists, do the following:**
 - a) Select a new marking m .
 - b) If no transitions are enabled at m , tag m "dead-end".
 - c) While there exist enabled transitions at m , do the following for each enabled transition t at m :
 - I. Obtain the marking m' that results from firing t at m .
 - II. If m' does not appear in the graph add m' and tag it "new".
 - III. Draw an arc with label t from m to m' (if not already present).
3. Output the graph



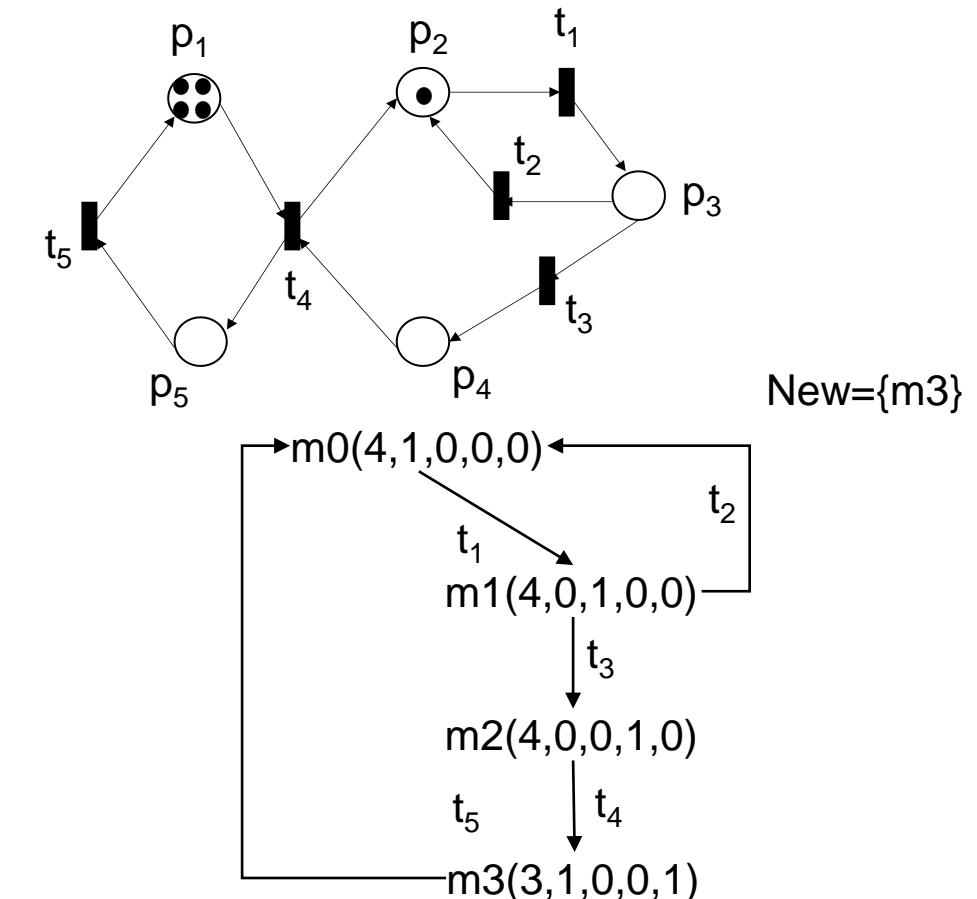
Reachability graph

1. Label the initial marking m_0 as the root and tag it "new".
2. **While "new" markings exists, do the following:**
 - a) Select a new marking m .
 - b) If no transitions are enabled at m , tag m "dead-end".
 - c) While there exist enabled transitions at m , do the following for each enabled transition t at m :
 - I. Obtain the marking m' that results from firing t at m .
 - II. If m' does not appear in the graph add m' and tag it "new".
 - III. Draw an arc with label t from m to m' (if not already present).
3. Output the graph



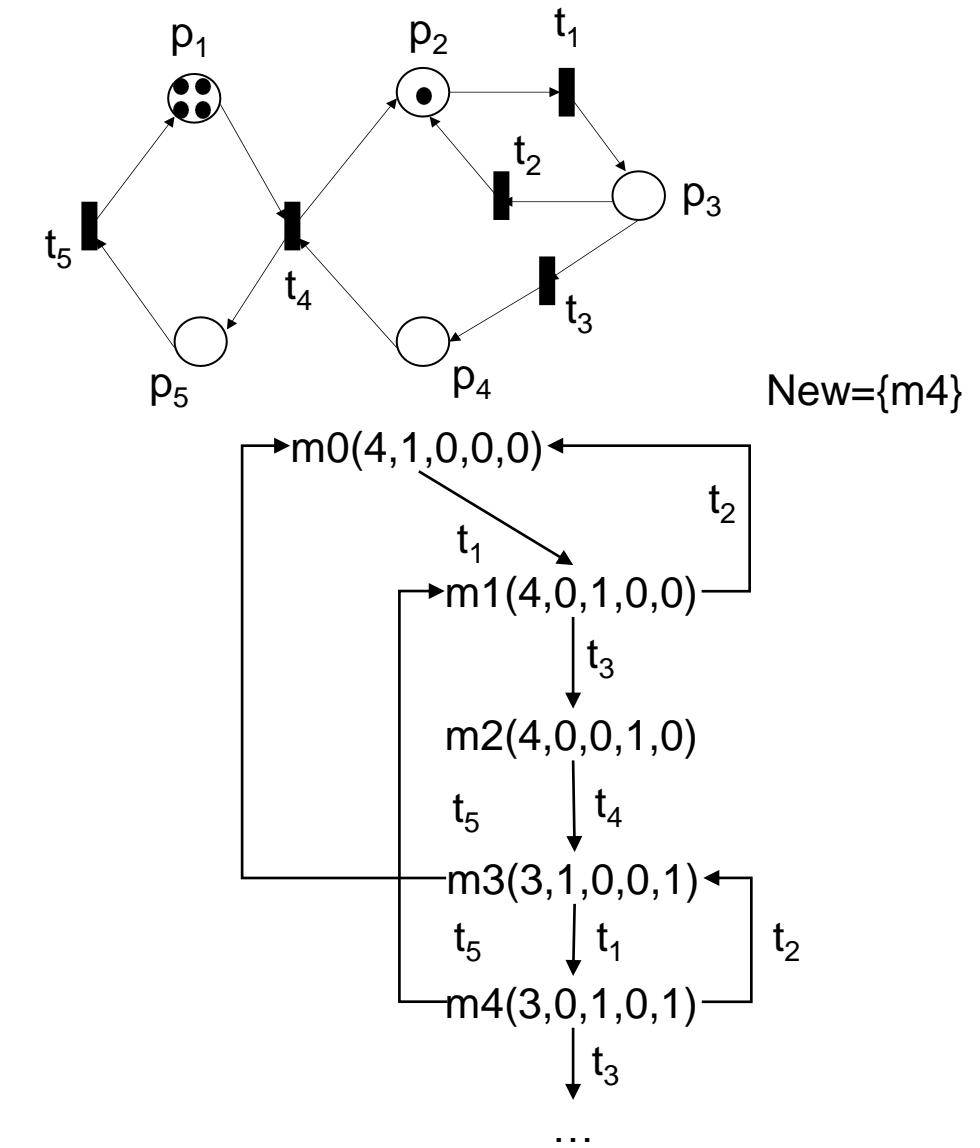
Reachability graph

1. Label the initial marking m_0 as the root and tag it "new".
2. **While "new" markings exists, do the following:**
 - a) Select a new marking m .
 - b) If no transitions are enabled at m , tag m "dead-end".
 - c) While there exist enabled transitions at m , do the following for each enabled transition t at m :
 - I. Obtain the marking m' that results from firing t at m .
 - II. If m' does not appear in the graph add m' and tag it "new".
 - III. Draw an arc with label t from m to m' (if not already present).
3. Output the graph

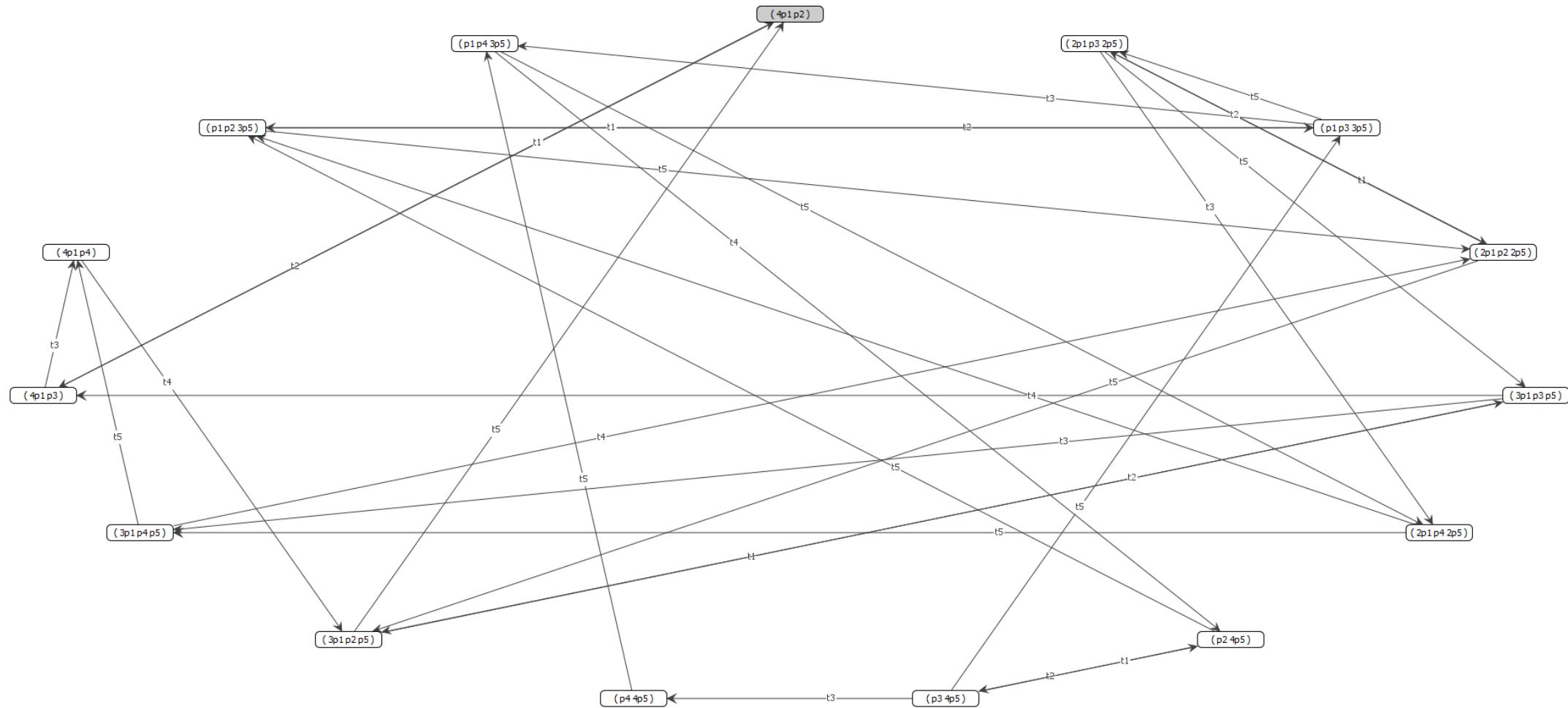


Reachability graph

1. Label the initial marking m_0 as the root and tag it "new".
2. **While "new" markings exists, do the following:**
 - a) Select a new marking m .
 - b) If no transitions are enabled at m , tag m "dead-end".
 - c) While there exist enabled transitions at m , do the following for each enabled transition t at m :
 - I. Obtain the marking m' that results from firing t at m .
 - II. If m' does not appear in the graph add m' and tag it "new".
 - III. Draw an arc with label t from m to m' (if not already present).
3. Output the graph



Reachability graph

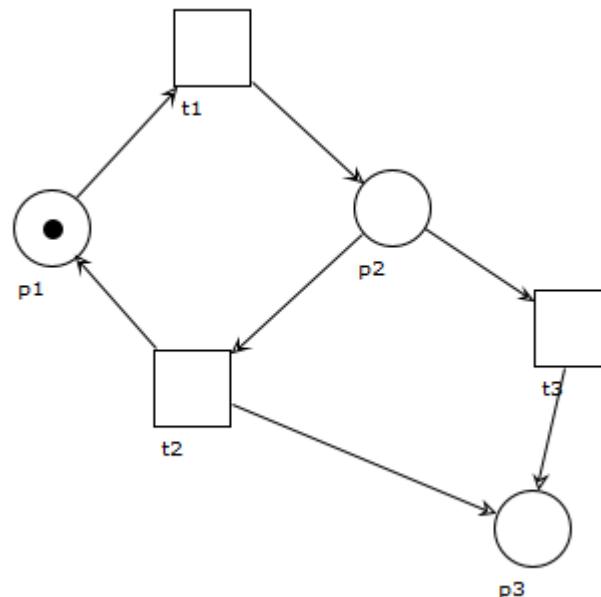


Some Petri Nets properties

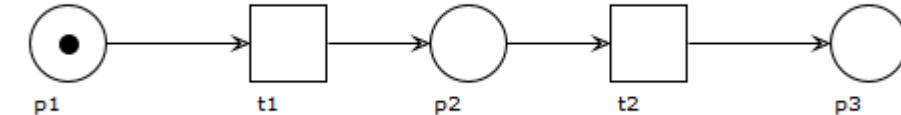
- Boundness
 - Does the net manage a limited number of tokens regardless of the reached marking?
 - Implies a finite state space
- Liveness
 - Does the net have a marking with none of the transitions are enabled?
 - Implies the absence of deadlocks

Boundness

- A Petri Net $N = (P, T, F)$ with initial marking M_0 is k -bounded iif for each reachable markings $M_0 \xrightarrow{\sigma} M_i$
 - $\forall p \in P, m_i(p) \leq k$
- If $k = 1$ then the Petri Net is safe



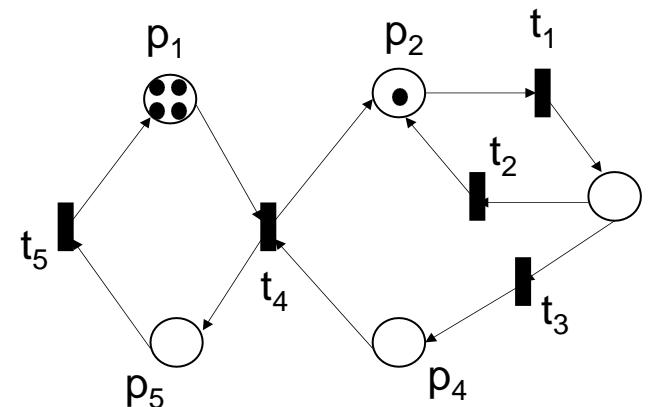
Unbounded: reachability graph grows infinitely



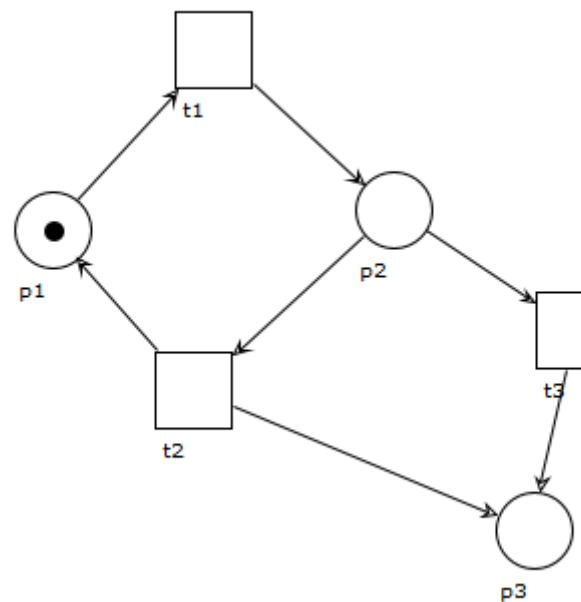
1-Bounded (safe)

Termination

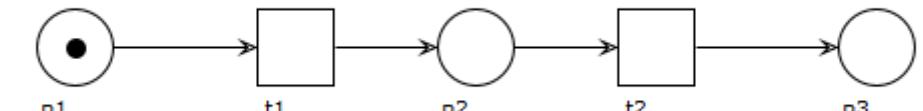
- Intuition: The Petri net always reaches a terminal marking
- A Petri net is terminating if every run is finite.
- Petri net with a finite and acyclic reachability graph is terminating
- A terminating Petri net has only finitely many runs



Non-terminating: reachability graph is cyclical



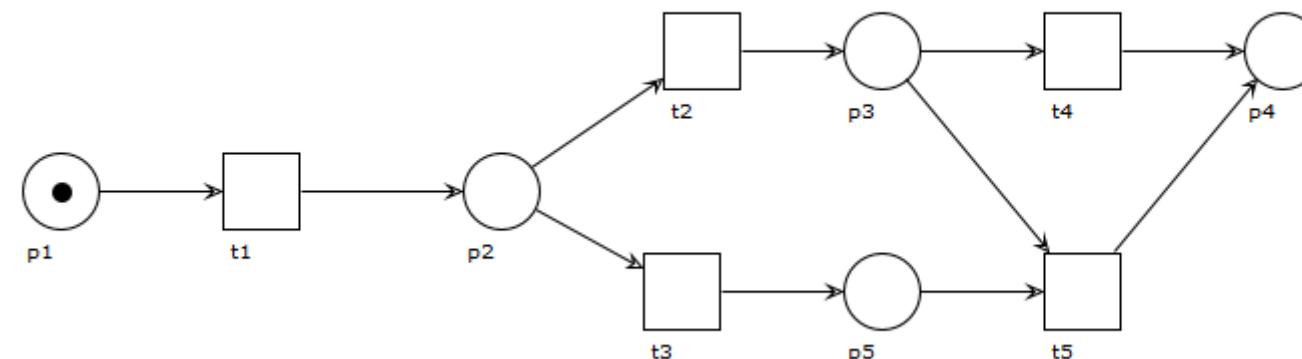
Non-terminating: infinite reachability graph



Terminating: marking $(0,0,3)$ is terminal

Deadlock freedom

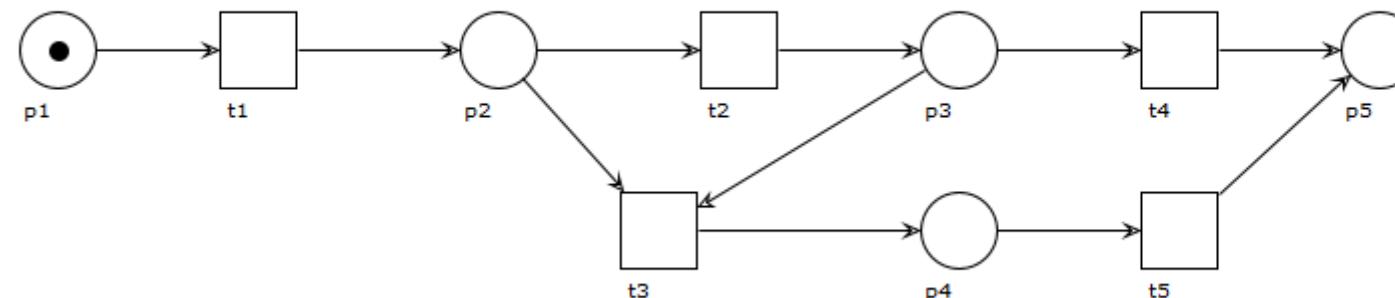
- Intuition: The Petri net can reach a terminal marking
- A Petri net is deadlock free if at least one transition is enabled at every reachable marking



Deadlock: no transition can fire from $(0,0,0,0,1)$

Dead transition

- Intuition: A transition can in principle occur (its implemented functionality can be used)
- A transition t of a Petri net is dead if t is not enabled at any reachable marking.

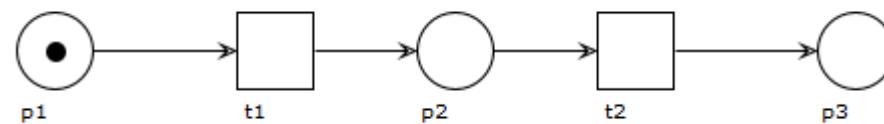


t_3 and t_5 are dead: they can never fire

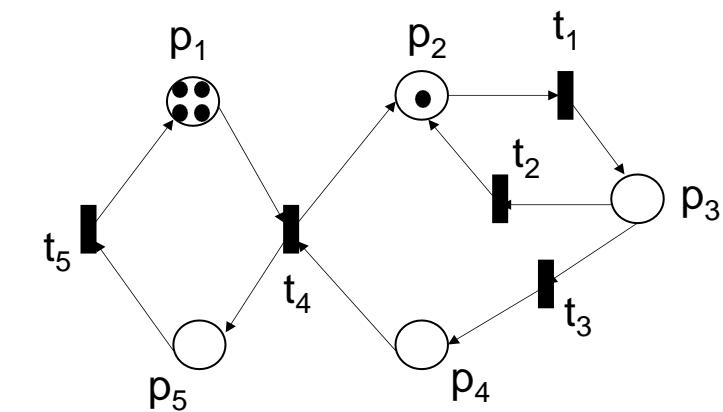
Liveness

- Live transition
 - A transition t is live iif from every reachable marking M there is a marking M' such that t is enabled
 - This property ensures that the transition can always become enabled again
- Live net
 - A Petri Net is live iif $\forall t \in T, t$ is live
- Liveness and termination exclude each other
 - Termination holds for Petri net that always reach a marking where no transition is enabled.
 - A Petri Net is terminating iif every run is finite: i.e., the reachability graph is finite and acyclic

Liveness



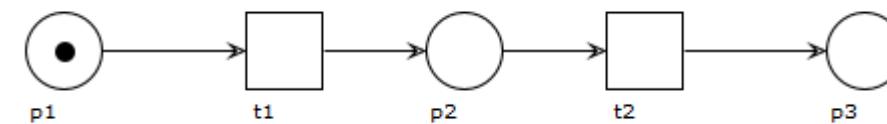
Not live: t_1 and t_2 can fire only once



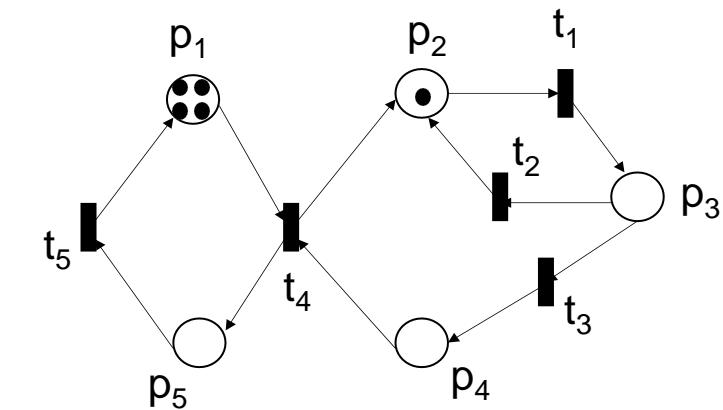
Live: all transitions can fire

Home-marking

- Intuition: A marking can always be reached again
- A marking m is a home-marking if from any reachable marking we can reach m . A Petri net is reversible if its initial marking is a home-marking.
- A Petri net is reversible if and only if its reachability graph is strongly connected.



Not reversible: $m_0(1,0,0)$ is not a home-marking



Reversible: we can always reach $m_0(4,1,0,0,0)$

Example

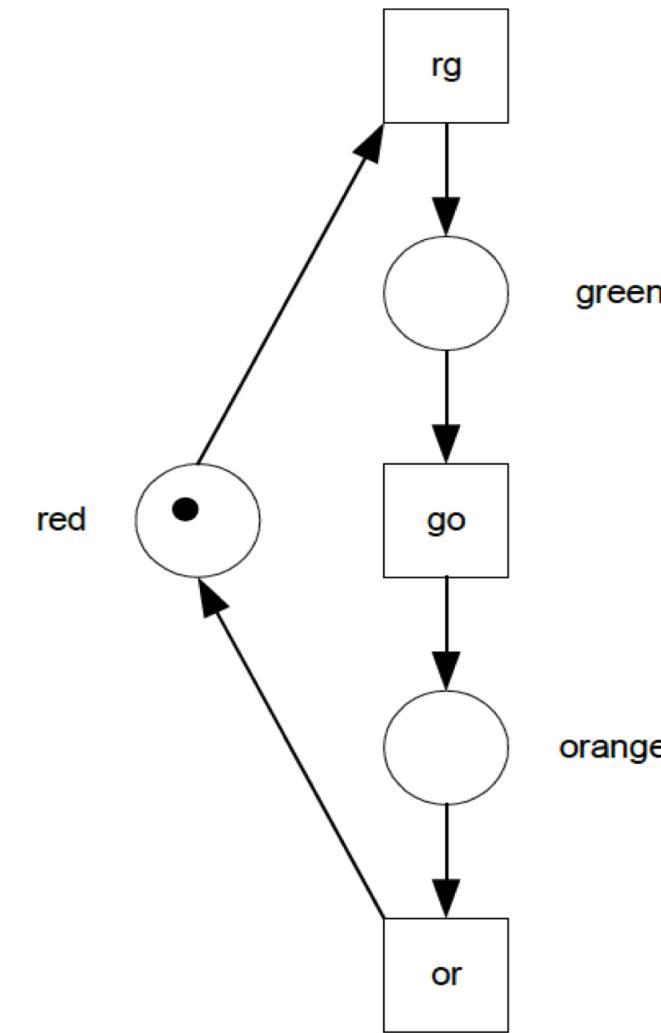
- We want to model the behaviour of a traffic light
 - Light can be green, yellow, or red



Source: W. van der Aalst, C. Stahl, Modeling Business Processes: A Petri net approach, MIT Press, 2011

Example

- We want to model the behaviour of a traffic light
 - Light can be green, yellow, or red



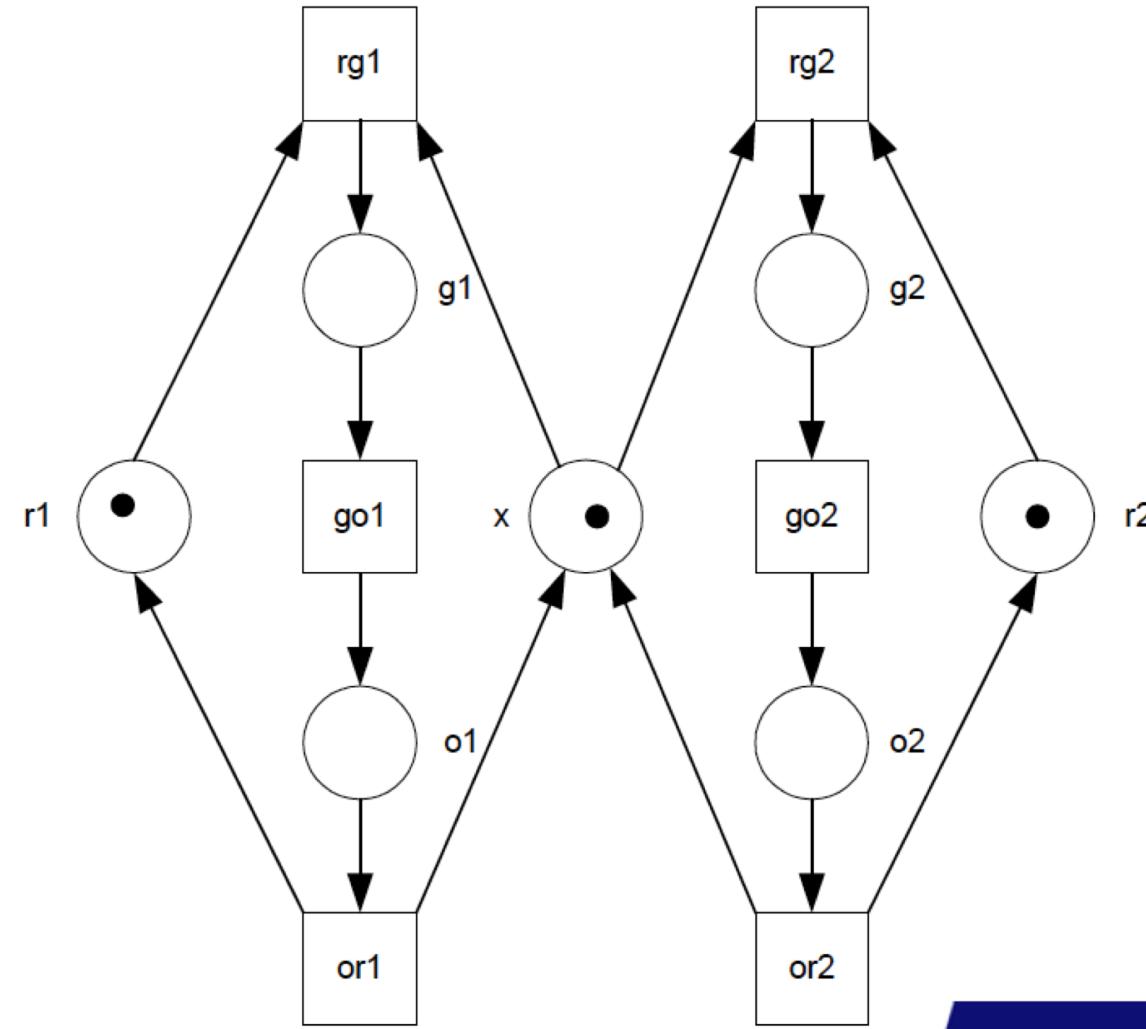
Source: W. van der Aalst, C. Stahl, Modeling Business Processes: A Petri net approach, MIT Press, 2011

Example

- We want to model with a Petri Net the behaviour of two traffic lights at an intersection, in a way that they cannot be green or yellow at the same time
- Conversely, they are allowed to signal red at the same time



Solution



Source: W. van der Aalst, C. Stahl, Modeling Business Processes: A Petri net approach, MIT Press, 2011

Workflow nets

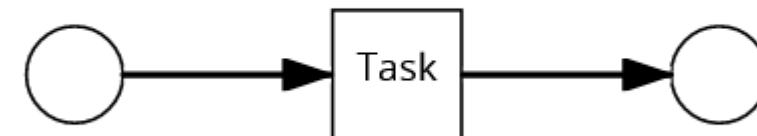
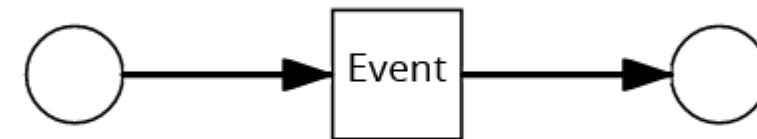
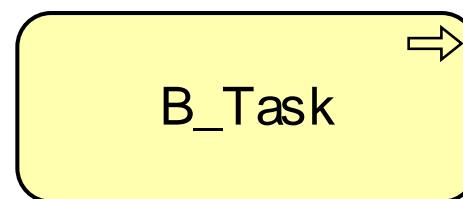
Workflow net

- Workflow net is defined as:
 - A Petri Net that models a workflow process definition (i.e., the life cycle of one case in isolation)
- A task
 - Generic piece of work (defined for a type of cases)
 - Corresponds to a transition
- A work item
 - Task enabled for a specific case
 - Corresponds to an enabled transition
- An activity
 - Actual execution of a work item
 - Correspond to a transition firing

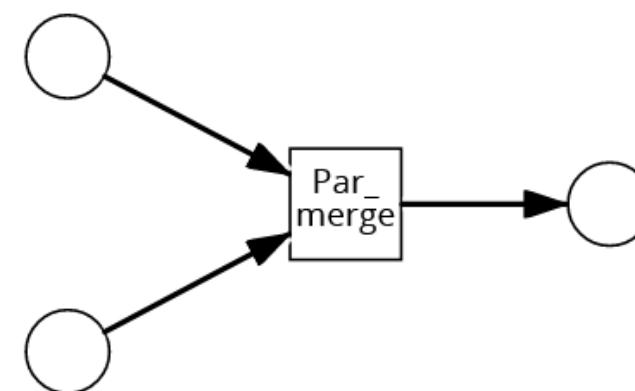
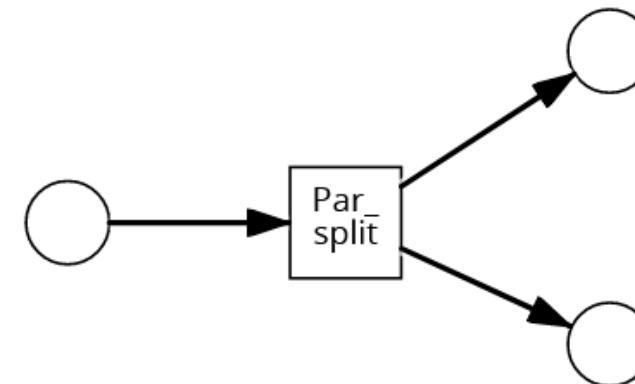
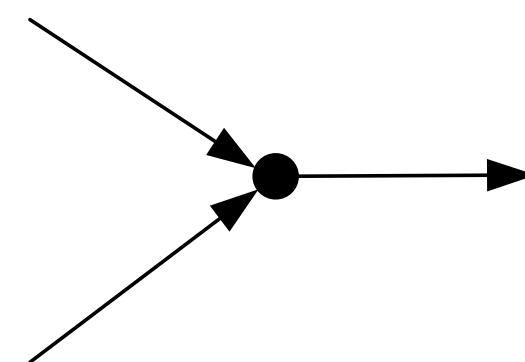
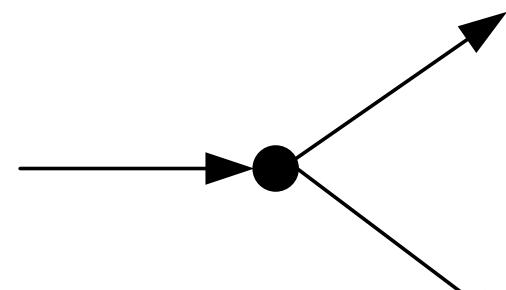
Workflow net

- A Petri net $N = (P, T, F)$ is a workflow net if
- Object creation: N contains an input place i (the source place) such that $\bullet i = \emptyset$.
- Object completion: N contains an output place o (the sink place) such that $o \bullet = \emptyset$.
- Connectedness: Every node in N is on a path from i to o .

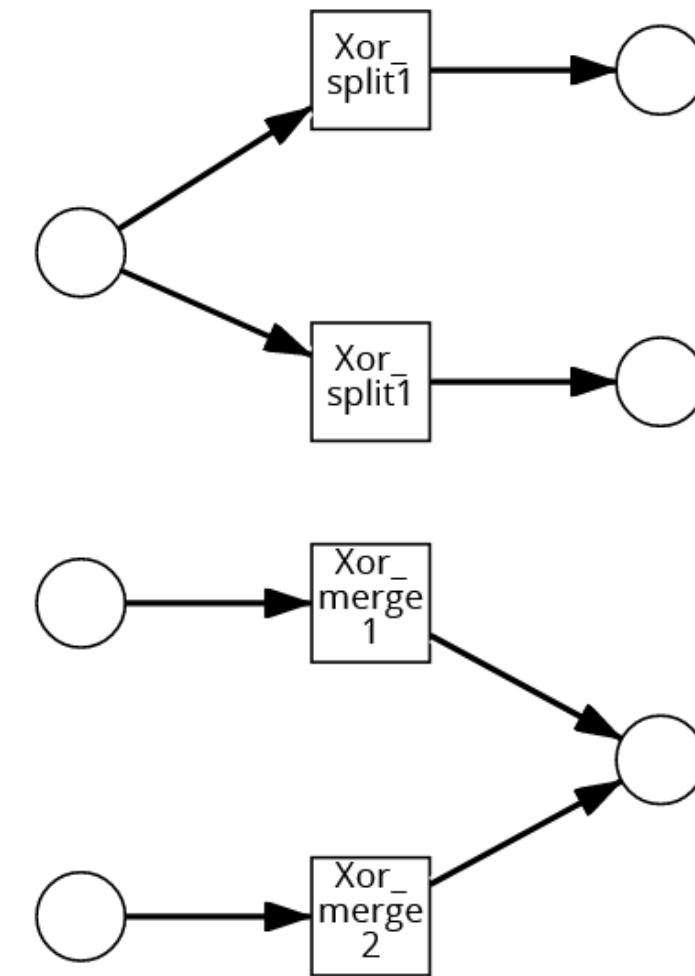
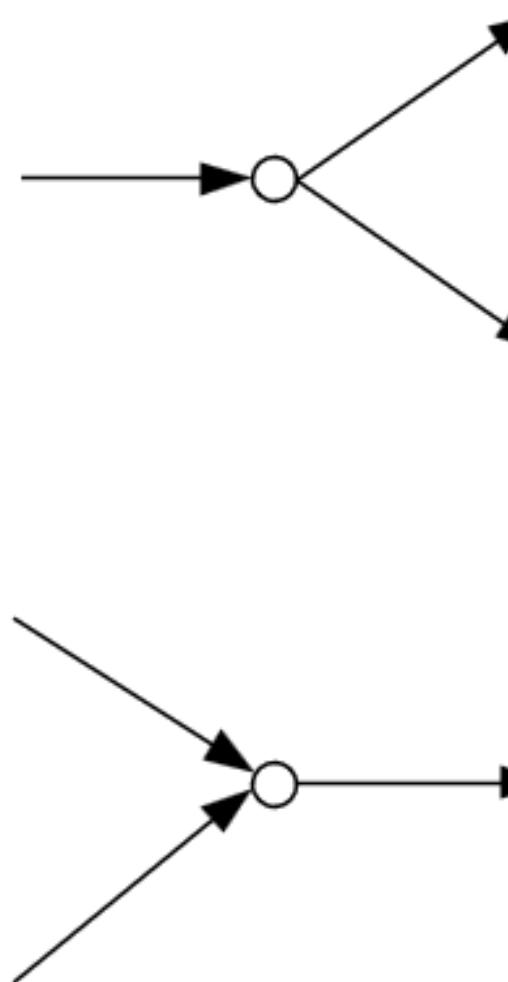
Workflow patterns with PN



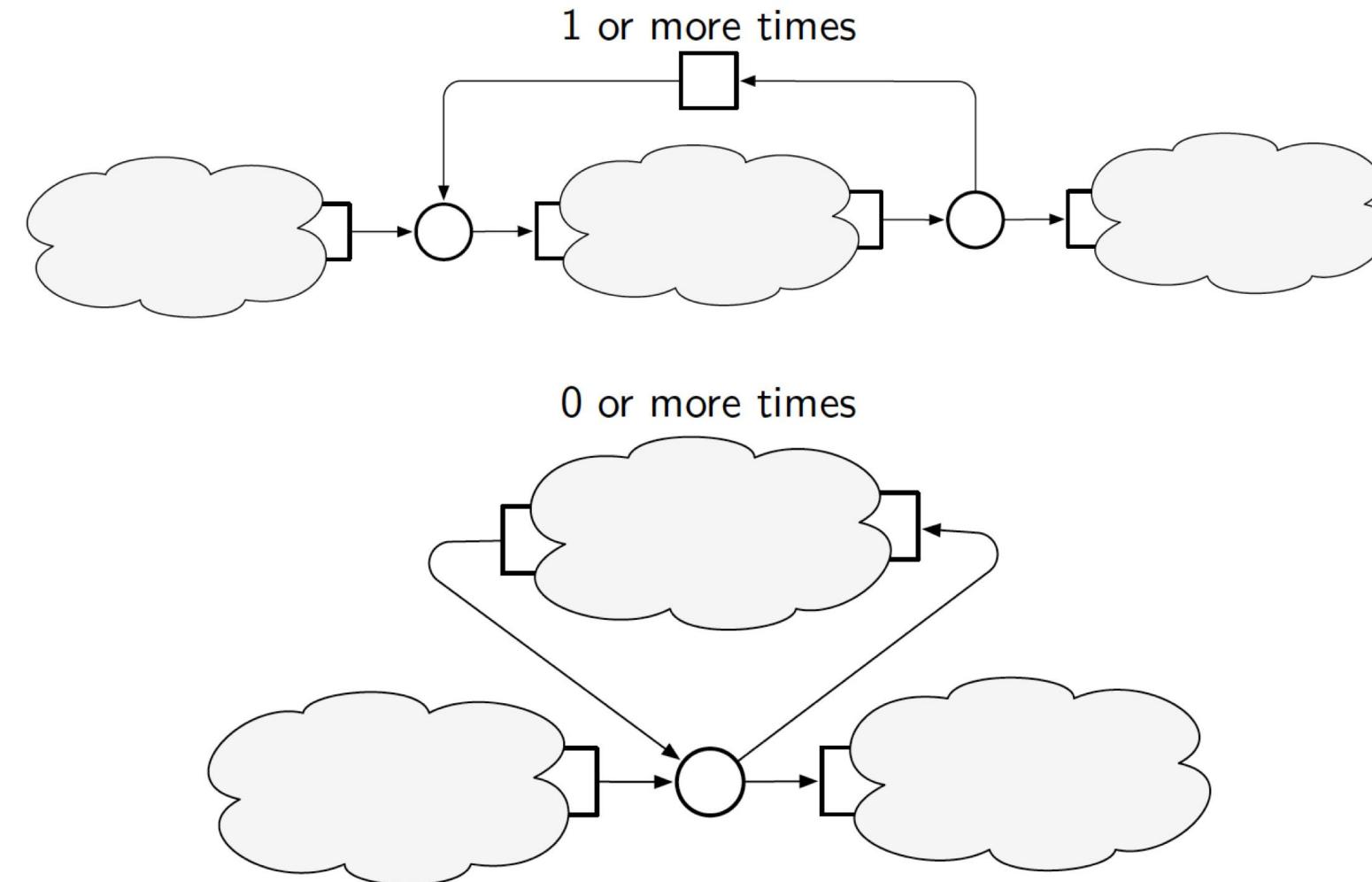
Workflow patterns with PN



Workflow patterns with PN



Workflow patterns with PN



Source: M. Montali, Conceptual Modeling form Information Systems, slides on Process Analysis – Petri Nets, Properties, A.Y. 2011/2012

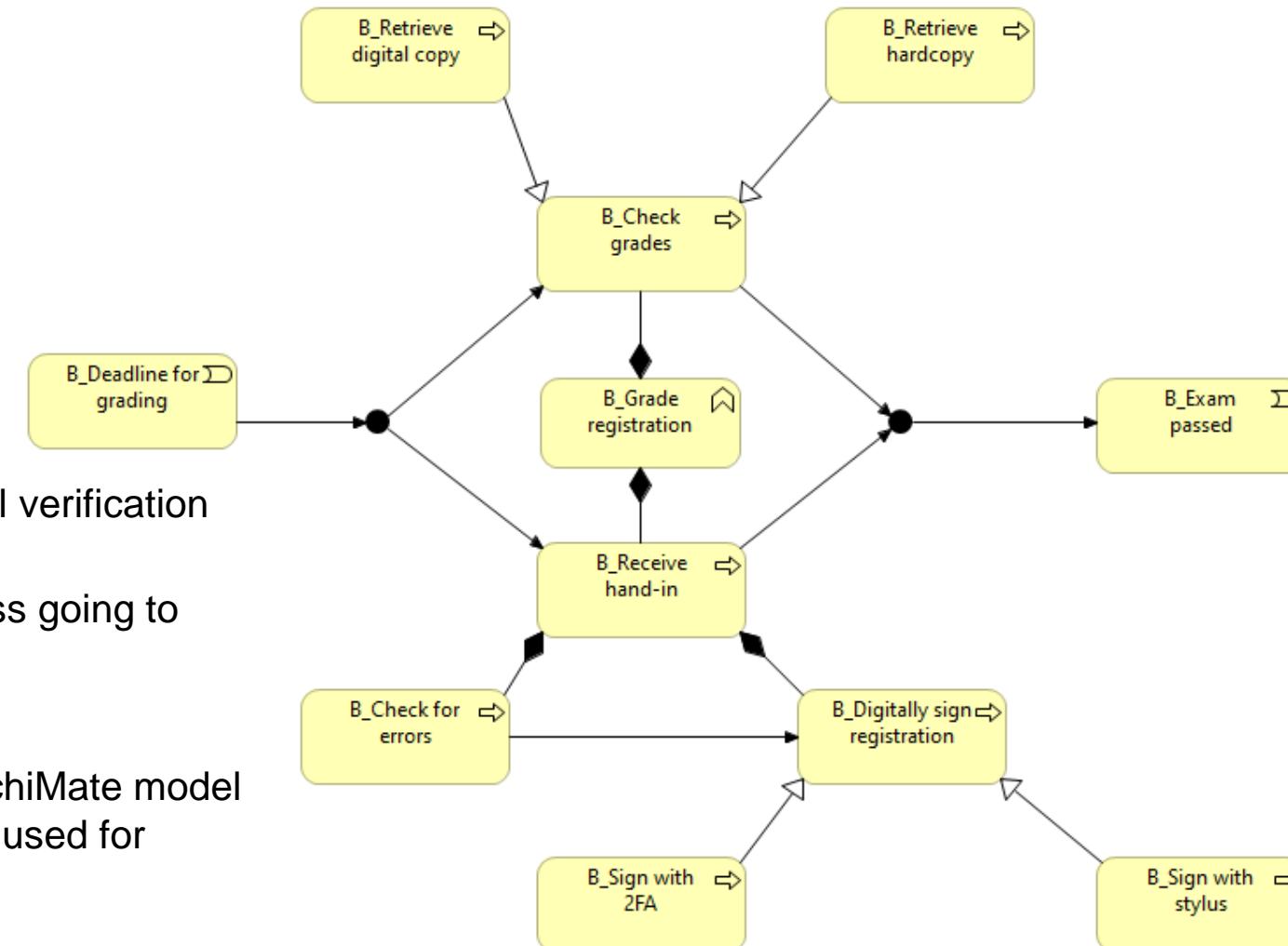
Soundness

- A WF-Net (corresponding to a BP) is sound iif
 - “for any case, the procedure will terminate eventually, and at the moment the procedure terminates there is a token in place ‘o’ and all the other places are empty”
[from W.M.P. van der Aalst. Verification of Workflow Nets. In P. Azéma and G. Balbo, editors, Application and Theory of Petri Nets 1997, LNCS1248, pages 407–426. Springer-Verlag, 1997.]
- A little bit more formally:
 - For every state M reachable from state i there exists a firing sequence σ so that $M[\sigma] o$
 - State o is the only state reachable from state i with at least one token in place o
 - There are no dead transitions in the workflow net in state i
- Reachability graph can be used to verify the soundness of a WF-net

Soundness

- Van der Aalst proves that a WF-net is sound iif for the extended WF-Net the following properties hold:
 - Liveness
 - Safeness (1-boundness)
- The extended WF-Net is obtained adding to the WF-net an additional transition which has o as its input place and i as its output place

Back to our example in ArchiMate

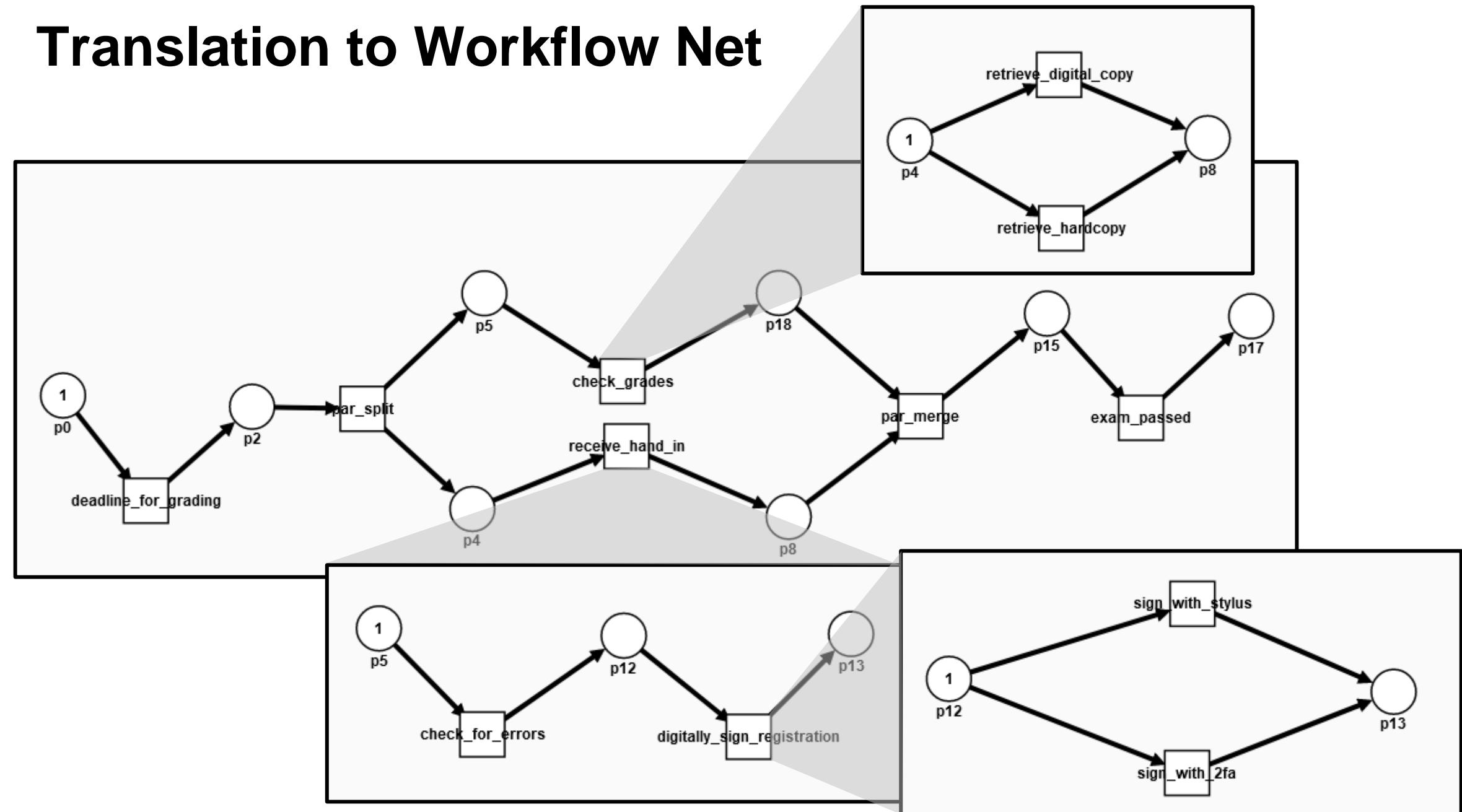


ArchiMate does not provide formal verification techniques:

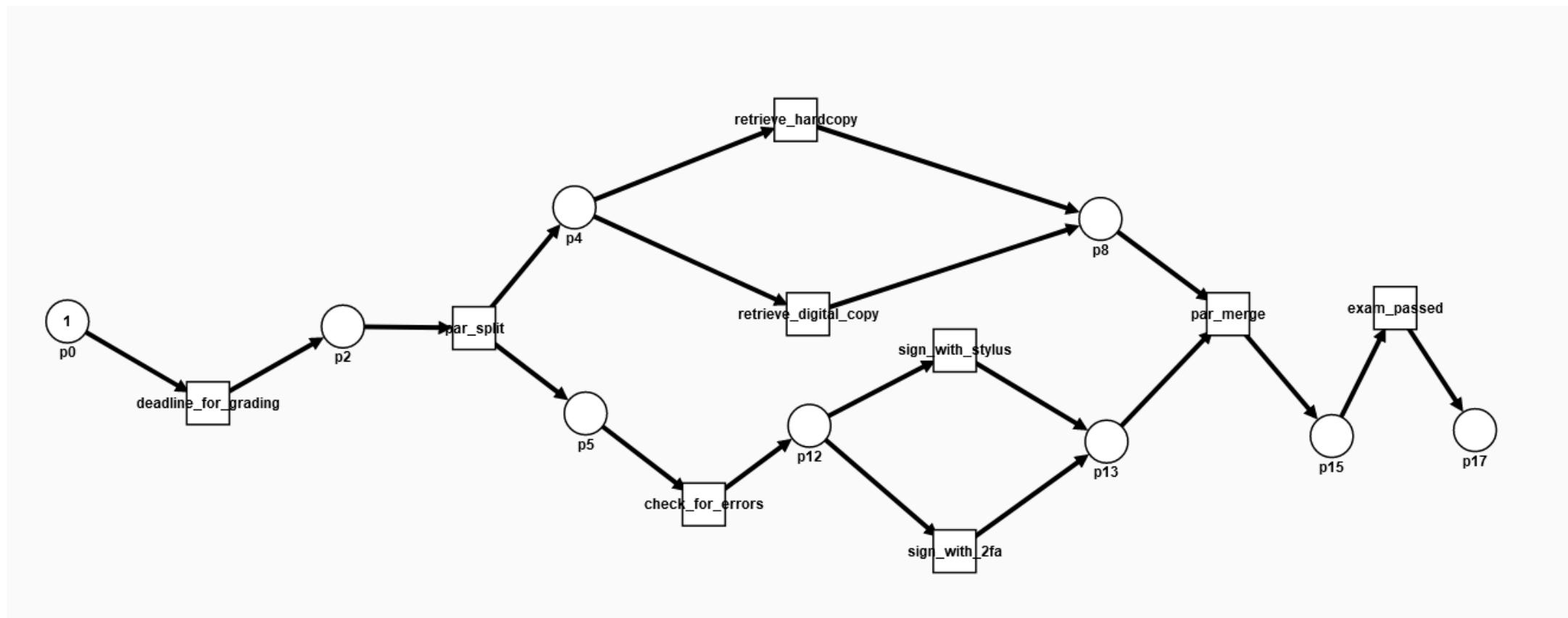
- Is the grade registration process going to terminate?
- Can all tasks be executed?

However, we can translate the ArchiMate model into a Workflow Net, which will be used for verification

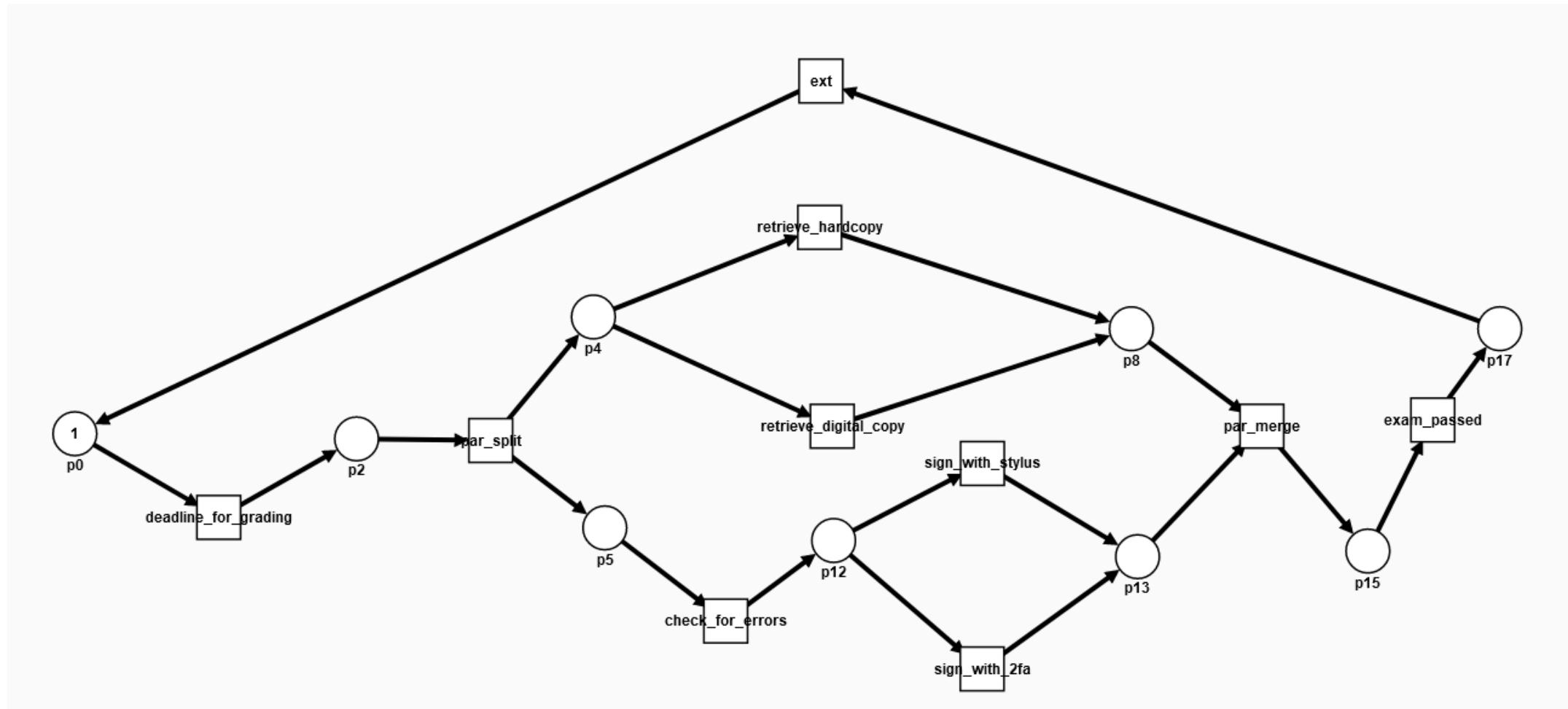
Translation to Workflow Net



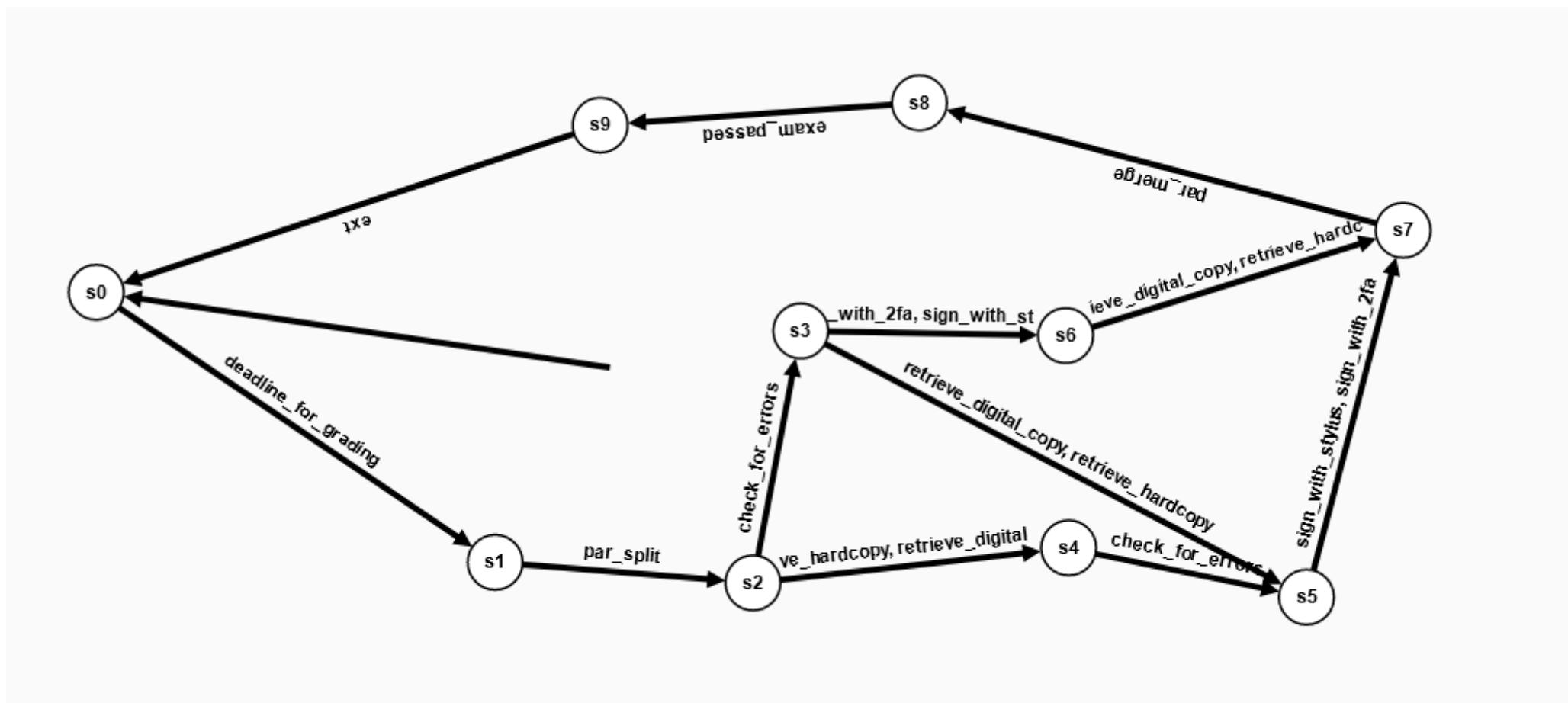
Equivalent Workflow Net



Extended Workflow Net



Reachability graph

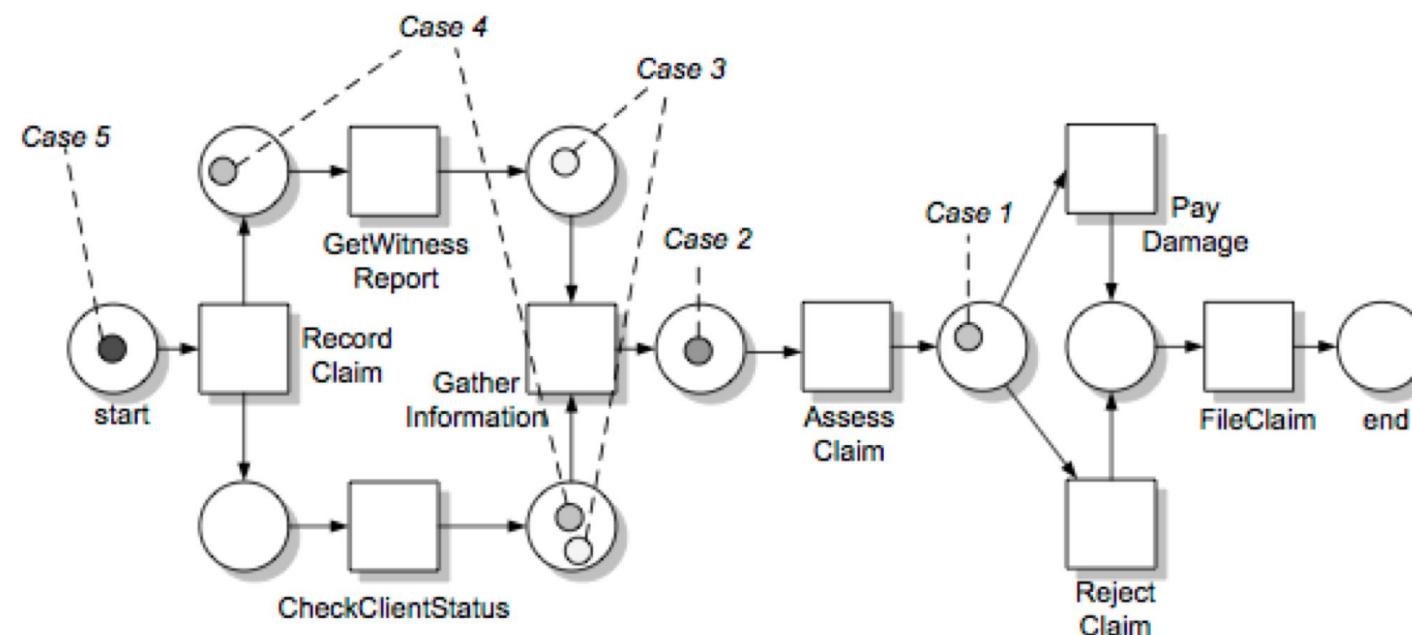


Analysis results

Petri Net Analysis				
Perform various tests on a petri net at once.				
bcf No	bicf No	output_nonbranching No	isolated_elements No	nonpure_only_simple_side_conditions No
pure Yes	strongly_connected Yes	simply_live Yes	backwards_persistent No	s_net No
reversible Yes	k-bounded 1	persistent No	t_net No	free_choice Yes
conflict_free No	k-marking 1	restricted_free_choice No	bounded Yes	homogeneous Yes
strongly_live Yes	safe Yes	asymmetric_choice Yes	weakly_live Yes	weakly_connected Yes

Managing multiple instances

- Classical Petri Nets are suitable to manage only one instance
- All the tokens have the same “meaning”
- Coloured Petri Nets are used to manage multi-instance processes
- Can also model other properties: resources, data, etc.



Weske, Business Process Management:
Concepts, Languages, and Architectures (2nd ed.), Springer, 2012

Study material

- Books and articles:
 - Van der Aalst et al. - Workflow Management: Models, methods and systems
 - Available at: <https://pure.tue.nl/ws/files/2456322/543561.pdf>
 - Chapter 2.2, 2.3 (not 2.3.3), 4.1 to 4.3, A.1 to A.3
- Modeling tools:
 - APO: <https://apo.adrian-jagusch.de/#!/Sample%20Net>
 - WoPeD: <https://github.com/woped/WoPeD/releases>

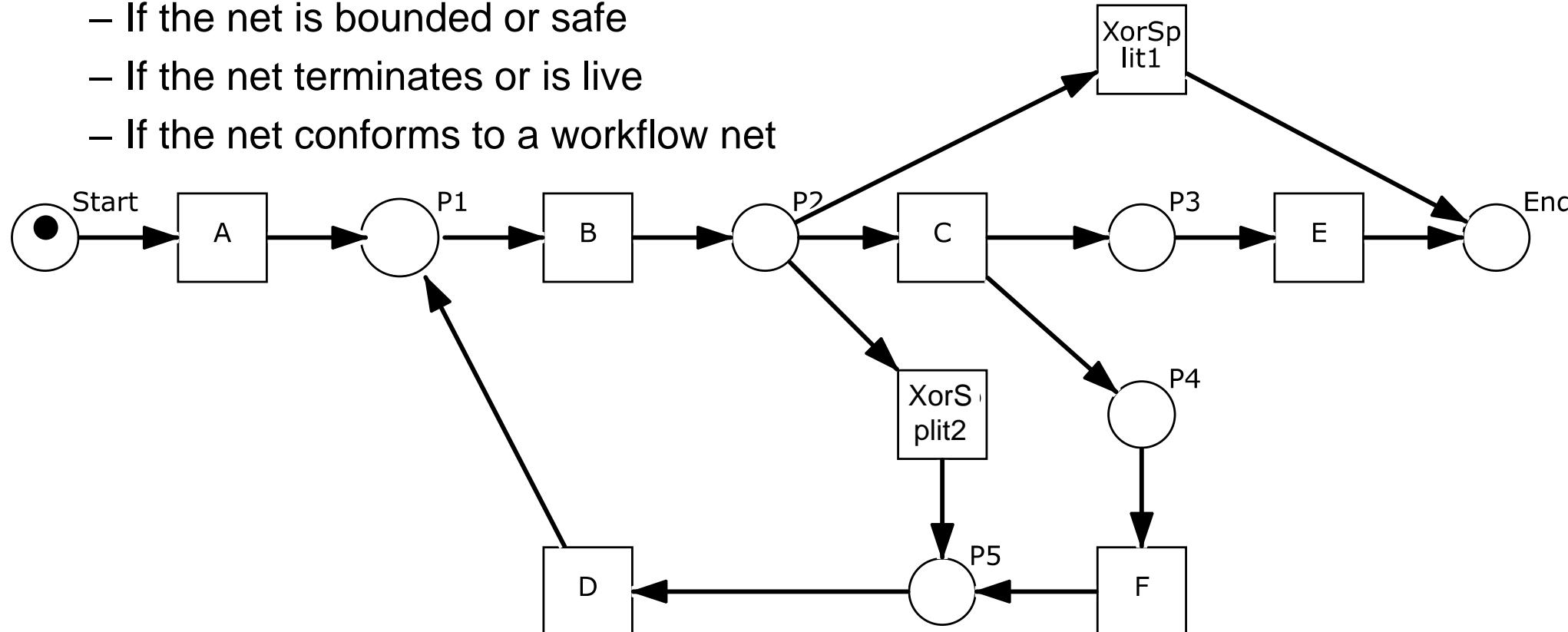
Exercises

Please answer all exercises to demonstrate your skills.

Solutions will be available at 11:45

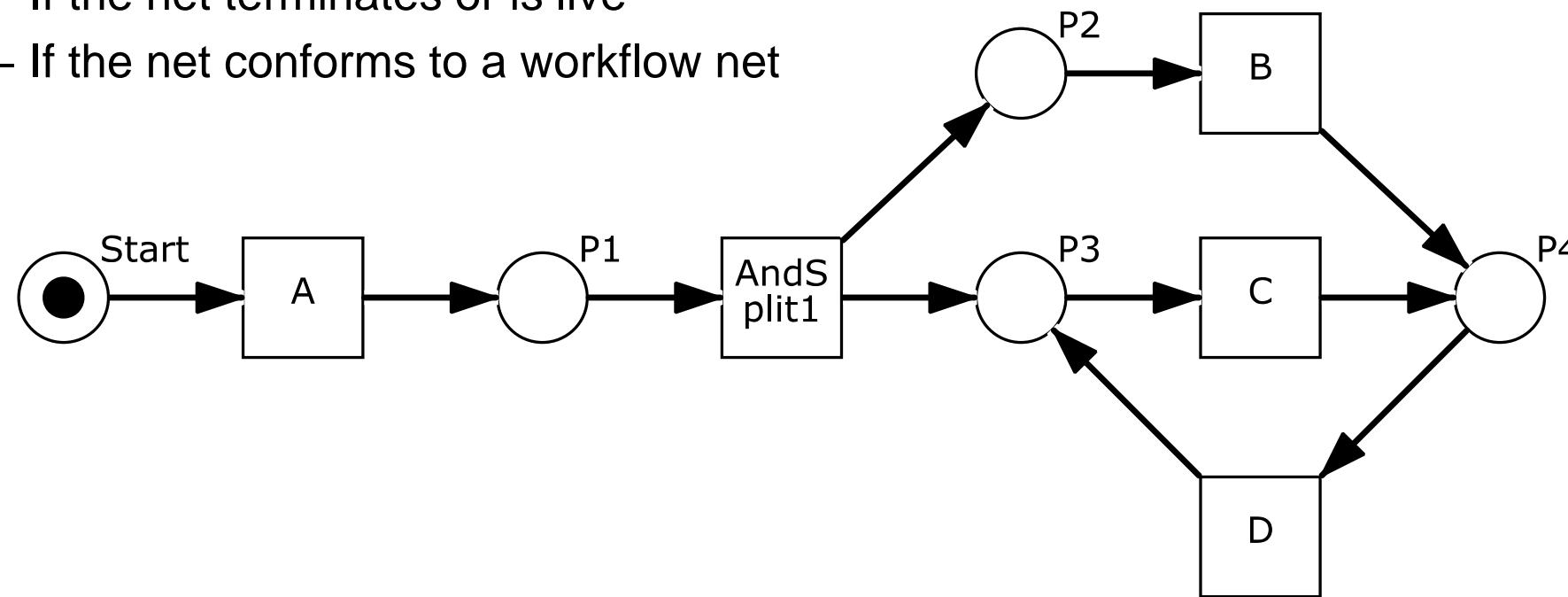
Exercise 1

- Given the following Petri Net, compute:
 - P , T , F sets
 - The reachability graph
 - If the net is bounded or safe
 - If the net terminates or is live
 - If the net conforms to a workflow net



Exercise 2

- Given the following Petri Net, compute:
 - P , T , F sets
 - The reachability graph
 - If the net is bounded or safe
 - If the net terminates or is live
 - If the net conforms to a workflow net



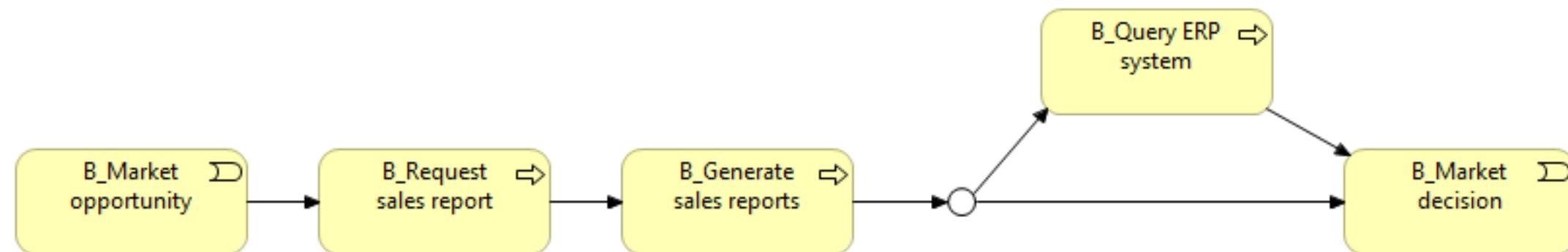
Exercise 3

Speedy is a delivery company that wants to create a new reporting system for the top management. After inspecting the sales reports, the top management may also need to query the existing ERP system, based on Oracle Fusion, to get detailed sales and HR information.

An ArchiMate model representing the process is enclosed below.

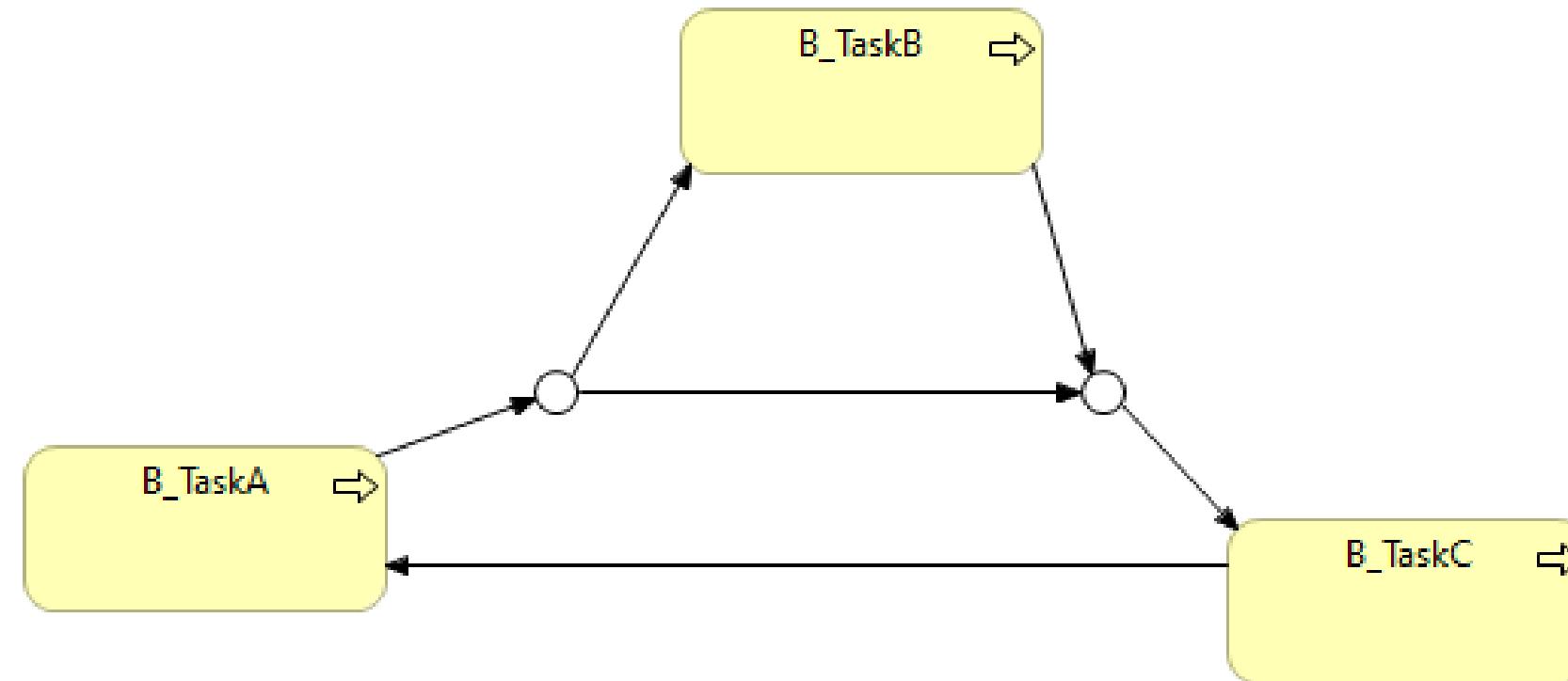
Starting from this model, create an equivalent Petri Net and try to answer the following questions:

- What is the reachability graph of the net?
- Is the net sound?



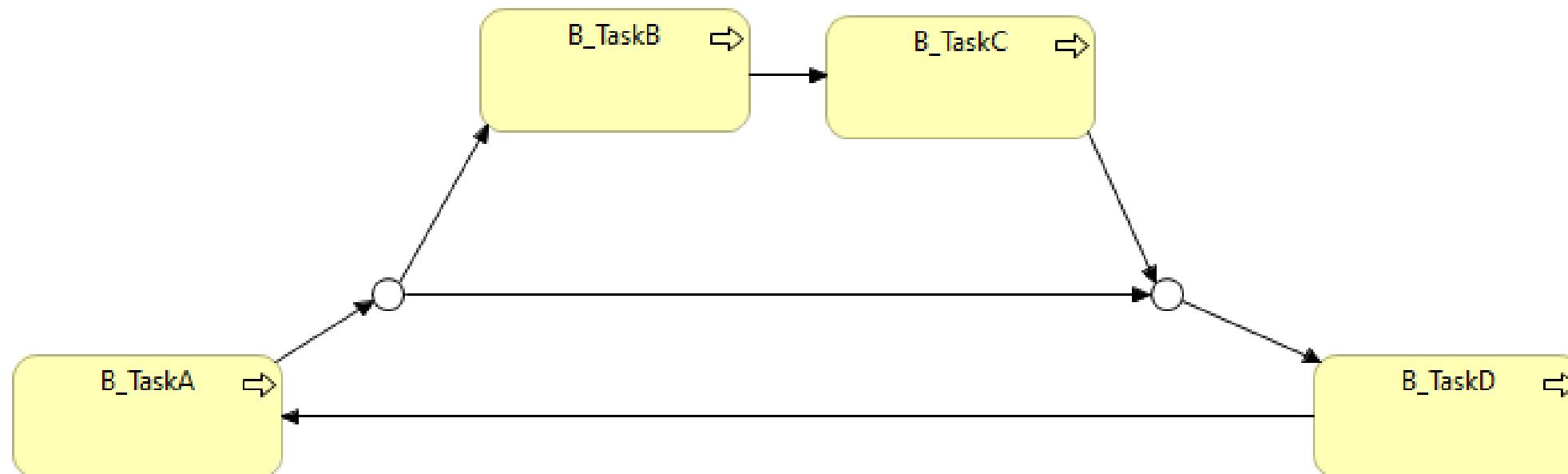
Exercise 4

- Translate the following process model into a Petri Net.
- Check if the resulting Petri Net is sound. If not, propose an action to repair the process, to make it sound.



Exercise 5

- Translate the following process model into a Petri Net.
- Check if the resulting Petri Net is sound. If not, propose an action to repair the process, to make it sound.



02291 Systems Integration

Models for Mobile and Distributed Systems

Hugo A. López (hulo@dtu.dk)



28 February 2023

Overview

Introduction

Labelled Transition Systems

CCS — the Calculus of Communicating Systems

Value-passing CCS

The π -calculus

Further reading

Survey

Concurrent applications are everywhere!



Cloud Computing, Internet of Things, Service-Oriented Architectures, Microservices, Blockchains...

Concurrent applications are everywhere!



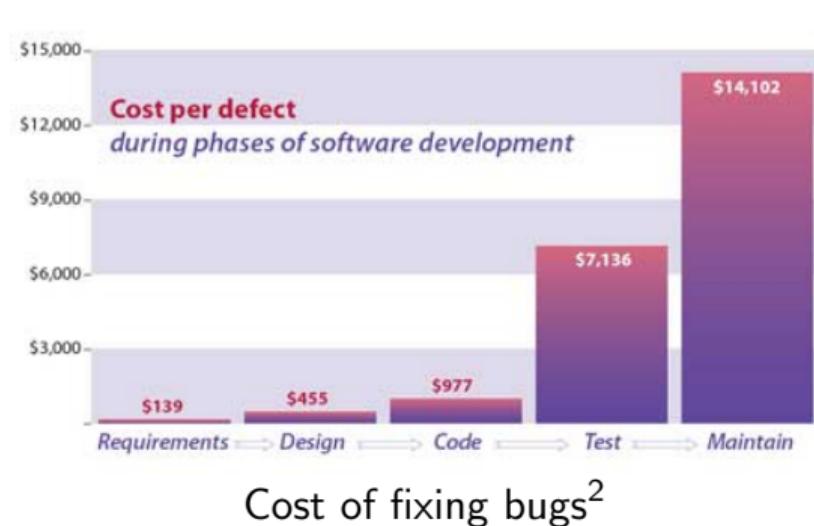
Cloud Computing, Internet of Things, Service-Oriented Architectures, Microservices,
Blockchains...



... personal computers, smartphones, cars...

Concurrent and distributed programming is difficult

Programming time spent debugging:¹ **49.9%**

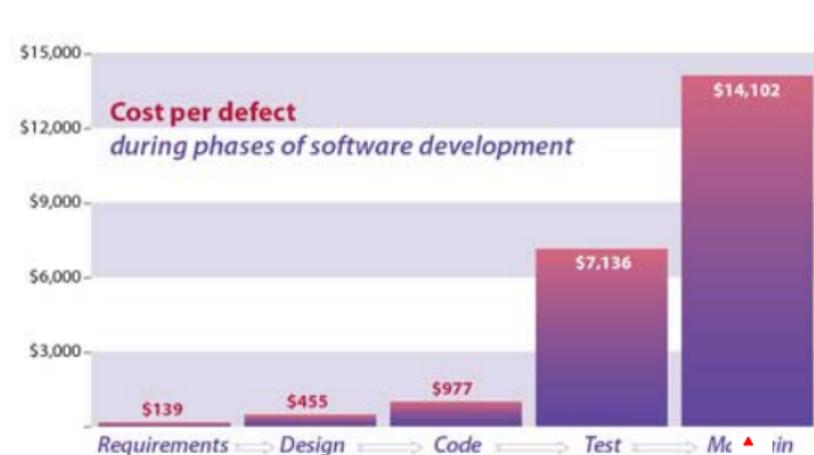


¹ Evans Data Corporation (2012), Payscale (2012), RTI (2002), CVP Surveys (2012)

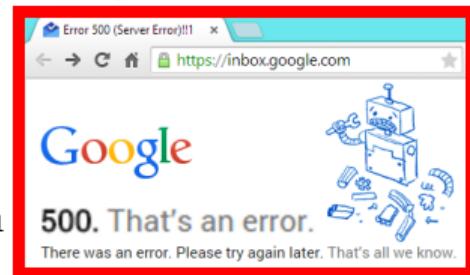
² B. Boehm and V. Basili. *Software Defect Reduction Top 10 List*. IEEE Computer, January 2001

Concurrent and distributed programming is difficult

Programming time spent debugging:¹ **49.9%**



Cost of fixing bugs²



¹ Evans Data Corporation (2012), Payscale (2012), RTI (2002), CVP Surveys (2012)

² B. Boehm and V. Basili. *Software Defect Reduction Top 10 List*. IEEE Computer, January 2001

Example: simple e-banking protocol

[START]

A client program can request *either*:

- ▶ the **Transactions**
for a certain date
 - ▶ then, the bank replies with a **Report**,
and the session goes back to [START]
- ▶ or **Quit**:
 - ▶ then, the session ends

Example: simple e-banking protocol

[START]

A client program can request *either*:

- ▶ the **Transactions** for a certain date
 - ▶ then, the bank replies with a **Report**, and the session goes back to [START]
- ▶ or **Quit**:
 - ▶ then, the session ends

```
def client(bank: Connection)
  bank.send("TRANS 07-10-2018") val report1 = bank.receive()
  bank.send("TRANS 08-10-2018") val report2 = bank.receive()
  bank.send("QUIT") bank.close()
  // ...
```

Example: simple e-banking protocol, version 2.0

[START]

A client program can request *either*:

- ▶ the **Transactions** for a certain date
 - ▶ then, the bank *can either*:
 - ▶ reply with a **Report**, and the session goes back to [START]
 - ▶ **or, reply Bye with the report,** and the session ends
- ▶ or **Quit**:
 - ▶ then, the session ends

```
def client(bank: Connection)
  bank.send("TRANS 07-10-2018") val report1 = bank.receive()
  bank.send("TRANS 08-10-2018") val report2 = bank.receive()
  bank.send("QUIT") bank.close()
// ...
```

Example: simple e-banking protocol, version 2.0

[START]

A client program can request *either*:

- ▶ the **Transactions** for a certain date
 - ▶ then, the bank *can either*:
 - ▶ reply with a **Report**, and the session goes back to [START]
 - ▶ **or, reply Bye with the report,** and the session ends
- ▶ or **Quit**:
 - ▶ then, the session ends

```
def client(bank: Connection)
  bank.send("TRANS 07-10-2018") val re-
  port1 = bank.receive()
  bank.send("TRANS 08-10-2018") val re-
  port2 = bank.receive()
  bank.send("QUIT") bank.close()
  // ...
```

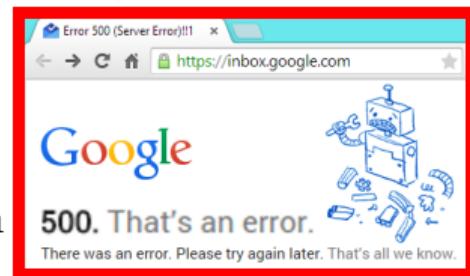
How can we spot the bug?

Goal: spot more bugs, earlier

Programming time spent debugging:³ **49.9%**



Cost of fixing bugs⁴

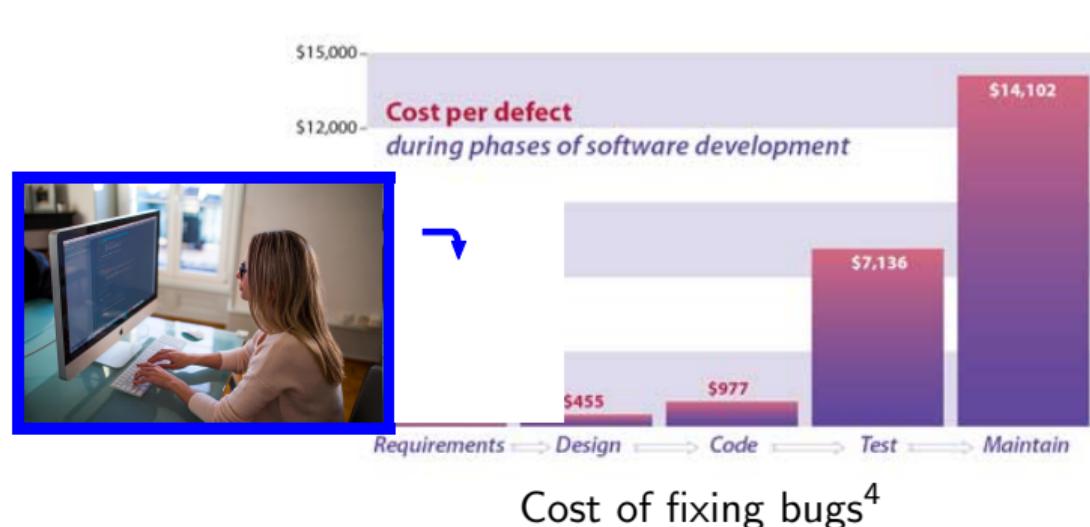


³ Evans Data Corporation (2012), Payscale (2012), RTI (2002), CVP Surveys (2012)

⁴ B. Boehm and V. Basili. *Software Defect Reduction Top 10 List*. IEEE Computer, January 2001

Goal: spot more bugs, earlier

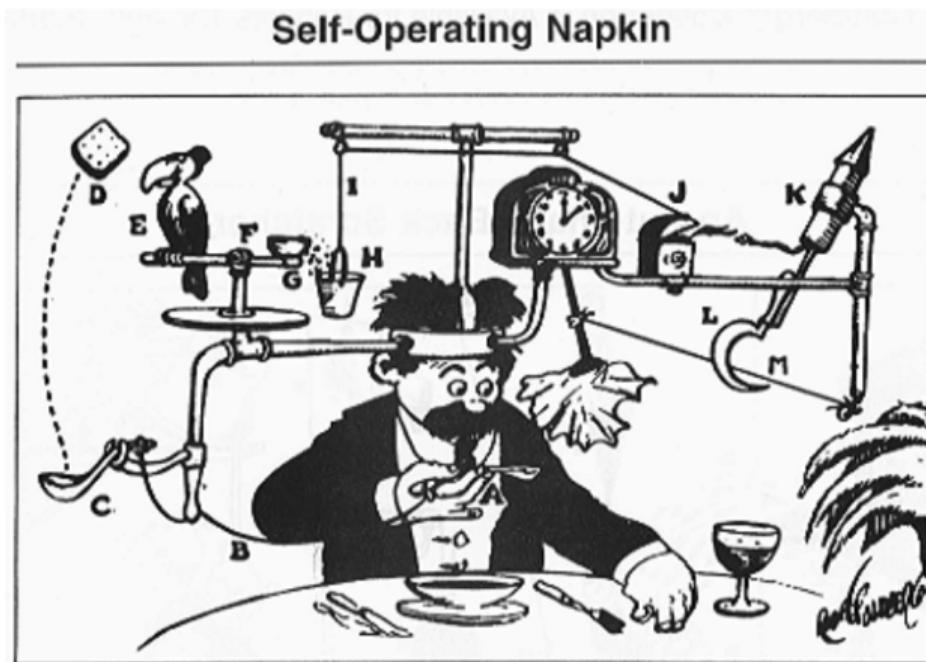
Programming time spent debugging:³ **49.9%**



³ Evans Data Corporation (2012), Payscale (2012), RTI (2002), CVP Surveys (2012)

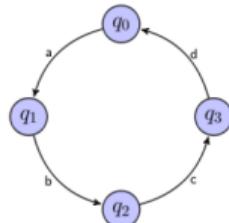
⁴ B. Boehm and V. Basili. *Software Defect Reduction Top 10 List*. IEEE Computer, January 2001

The verification problem



Program
(Turing-powerful)

The verification problem

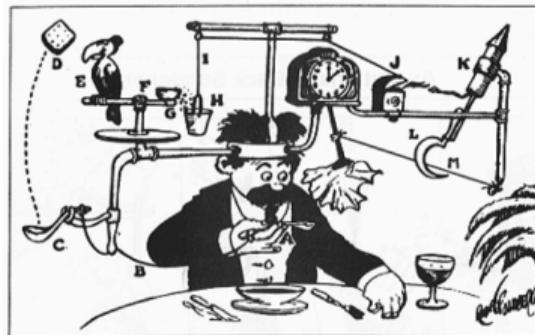


Model
(Much simpler!)

Abstraction

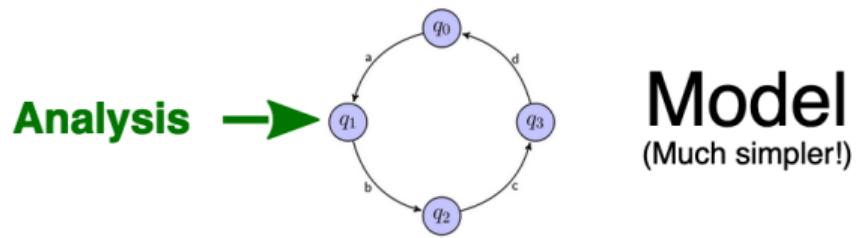


Self-Operating Napkin

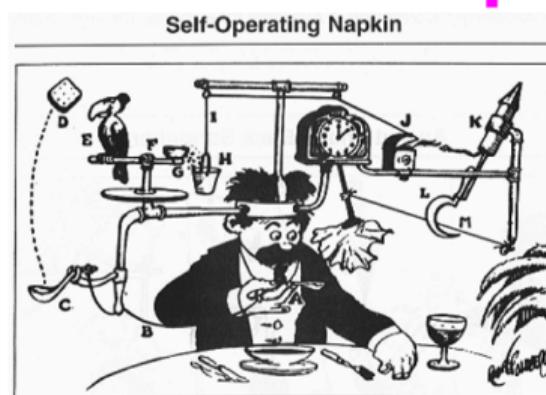


Program
(Turing-powerful)

The verification problem



Abstraction ↑



↓ Results

Program
(Turing-powerful)

Process calculi

A formalism for modelling **programs** communicating via **channels**

- ▶ **mathematical reasoning** for **proving correctness**

(Hoare '77; Milner '80; Bergstra and Klop, '82; Milner, Parrow and Walker '92; ...)



Process calculi

A formalism for modelling **programs** communicating via **channels**

- ▶ **mathematical reasoning** for **proving correctness**

(Hoare '77; Milner '80; Bergstra and Klop, '82; Milner, Parrow and Walker '92; ...)



```
c.send(42); x = c.recv()
```

Process calculi

A formalism for modelling **programs** communicating via **channels**

- ▶ **mathematical reasoning** for **proving correctness**

(Hoare '77; Milner '80; Bergstra and Klop, '82; Milner, Parrow and Walker '92; ...)



```
c.send<42>; x = c.recv()  
|| y = c.recv(); c.send<y+1>
```

Process calculi

A formalism for modelling **programs** communicating via **channels**

- ▶ **mathematical reasoning** for **proving correctness**

(Hoare '77; Milner '80; Bergstra and Klop, '82; Milner, Parrow and Walker '92; ...)



$$\frac{\text{c.send}(42); x = \text{c.recv}()}{\parallel y = \text{c.recv}(); \text{c.send}(y+1)} \rightarrow \frac{}{\parallel \text{c.send}(42+1)}$$

Process calculi

A formalism for modelling **programs** communicating via **channels**

- ▶ **mathematical reasoning** for **proving correctness**

(Hoare '77; Milner '80; Bergstra and Klop, '82; Milner, Parrow and Walker '92; ...)



$$\frac{\text{c.send}(42); x = \text{c.recv}()}{\parallel y = \text{c.recv}(); \text{c.send}(y+1)} \rightarrow \frac{x = \text{c.recv}()}{\parallel \text{c.send}(42+1)} \rightarrow \frac{0}{\parallel 0}$$

✓ **Correct interaction**

Process calculi

A formalism for modelling **programs** communicating via **channels**

- ▶ **mathematical reasoning** for **proving correctness**

(Hoare '77; Milner '80; Bergstra and Klop, '82; Milner, Parrow and Walker '92; ...)



$$\frac{\text{c.send}(42); x = \text{c.recv}()}{\parallel y = \text{c.recv}(); \text{c.send}(y+1)} \rightarrow \frac{x = \text{c.recv}()}{\parallel \text{c.send}(42+1)} \rightarrow \frac{0}{\parallel 0}$$

✓ **Correct interaction**

$$\frac{\text{c.send}("Hello"); x = \text{c.recv}()}{\parallel y = \text{c.recv}(); \text{c.send}(y+1)}$$

Process calculi

A formalism for modelling **programs** communicating via **channels**

- ▶ **mathematical reasoning** for **proving correctness**

(Hoare '77; Milner '80; Bergstra and Klop, '82; Milner, Parrow and Walker '92; ...)



$$\frac{}{\parallel c.\text{send}(42); x = c.\text{recv}() \quad \rightarrow \quad x = c.\text{recv}() \quad \rightarrow \quad 0}{\parallel y = c.\text{recv}(); c.\text{send}(y+1)} \quad \rightarrow \quad \parallel c.\text{send}(42+1) \quad \rightarrow \quad \parallel 0}$$

✓ Correct interaction

$$\frac{}{\parallel c.\text{send}("Hello"); x = c.\text{recv}() \quad \rightarrow \quad x = c.\text{recv}() \quad \rightarrow \quad 0}{\parallel y = c.\text{recv}(); c.\text{send}(y+1)} \quad \rightarrow \quad \parallel c.\text{send}("Hello"+1)}$$

✗ Incorrect interaction

Process calculi

A formalism for modelling **programs** communicating via **channels**

- ▶ **mathematical reasoning** for **proving correctness**

(Hoare '77; Milner '80; Bergstra and Klop, '82; Milner, Parrow and Walker '92; ...)



$$\frac{\text{c.send}(42); x = \text{c.recv}()}{\parallel y = \text{c.recv}(); \text{c.send}(y+1)} \rightarrow \frac{x = \text{c.recv}()}{\parallel \text{c.send}(42+1)} \rightarrow \frac{0}{\parallel 0}$$

✓ Correct interaction

$$\frac{\text{c.send}("Hello"); x = \text{c.recv}()}{\parallel y = \text{c.recv}(); \text{c.send}(y+1)} \rightarrow \frac{x = \text{c.recv}()}{\parallel \text{c.send}("Hello"+1)}$$

✗ Incorrect interaction

$$\frac{\text{c.send}(42); x = \text{c.recv}()}{\parallel y = \text{c.recv}()}$$

Process calculi

A formalism for modelling **programs** communicating via **channels**

- ▶ **mathematical reasoning** for **proving correctness**

(Hoare '77; Milner '80; Bergstra and Klop, '82; Milner, Parrow and Walker '92; ...)



$$\frac{\text{c.send}(42); x = \text{c.recv}()}{\parallel y = \text{c.recv}(); \text{c.send}(y+1)} \rightarrow \frac{x = \text{c.recv}()}{\parallel \text{c.send}(42+1)} \rightarrow \frac{0}{\parallel 0}$$

✓ Correct interaction

$$\frac{\text{c.send}("Hello"); x = \text{c.recv}()}{\parallel y = \text{c.recv}(); \text{c.send}(y+1)} \rightarrow \frac{x = \text{c.recv}()}{\parallel \text{c.send}("Hello"+1)}$$

✗ Incorrect interaction

$$\frac{\text{c.send}(42); x = \text{c.recv}()}{\parallel y = \text{c.recv}()} \rightarrow \frac{x = \text{c.recv}()}{\parallel 0}$$

✗ Incorrect interaction

Labelled Transition Systems: an intuition

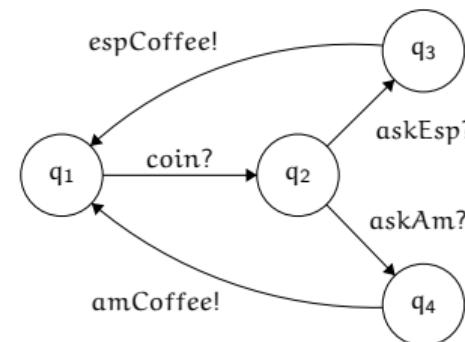
A **reactive system** can change its state by interacting with its environment



Labelled Transition Systems: an intuition

A **reactive system** can change its state by interacting with its environment

We can picture it as a **state machine** that changes state by responding to **inputs** and/or performing **outputs**



Labelled Transition Systems, formally

A **labelled transition system** is a triple $(Q, A, \dot{\rightarrow})$ where:

- ▶ Q is a non-empty set of **states**
- ▶ A is a non-empty set of **actions**, among which we distinguish:
 - ▶ visible **input actions**, written $a?$
 - ▶ visible **output actions**, written $a!$
 - ▶ a special **internal (invisible) action**, written τ
- ▶ $\dot{\rightarrow} \subseteq Q \times A \times Q$ is the **transition relation**
 - ▶ we write $q_1 \xrightarrow{\mu} q_2$ when $(q_1, \mu, q_2) \in \dot{\rightarrow}$ (read: in state q_1 , action μ leads to state q_2)

Labelled Transition Systems, formally

A **labelled transition system** is a triple (Q, A, \rightarrow) where:

- ▶ Q is a non-empty set of **states**
- ▶ A is a non-empty set of **actions**, among which we distinguish:
 - ▶ visible **input actions**, written $a?$
 - ▶ visible **output actions**, written $a!$
 - ▶ a special **internal (invisible) action**, written τ
- ▶ $\rightarrow \subseteq Q \times A \times Q$ is the **transition relation**
 - ▶ we write $q_1 \xrightarrow{\mu} q_2$ when $(q_1, \mu, q_2) \in \rightarrow$ (read: in state q_1 , action μ leads to state q_2)

Other notation:

- ▶ we may select one state in Q as the **initial state** (or **root**) of the LTS
- ▶ we use the symbol α to denote a **visible action** (input or output)
- ▶ we write $\bar{\alpha}$ for the **co-action** of α :
 - ▶ the co-action of the input $a?$ is the output $a!$, and vice versa

Labelled Transition Systems, formally

A **labelled transition system** is a triple (Q, A, \rightarrow) where:

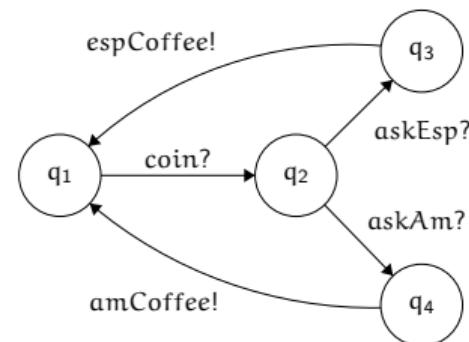
- ▶ Q is a non-empty set of **states**
- ▶ A is a non-empty set of **actions**, among which we distinguish:
 - ▶ visible **input actions**, written $a?$
 - ▶ visible **output actions**, written $a!$
 - ▶ a special **internal (invisible) action**, written τ
- ▶ $\rightarrow \subseteq Q \times A \times Q$ is the **transition relation**
 - ▶ we write $q_1 \xrightarrow{\mu} q_2$ when $(q_1, \mu, q_2) \in \rightarrow$ (read: in state q_1 , action μ leads to state q_2)

Other notation:

- ▶ we may select one state in Q as the **initial state** (or **root**) of the LTS
- ▶ we use the symbol α to denote a **visible action** (input or output)
- ▶ we write $\bar{\alpha}$ for the **co-action** of α :
 - ▶ the co-action of the input $a?$ is the output $a!$, and *vice versa*
- ▶ many books and papers use the notation: a for an input action, \bar{a} for an output

LTS examples: vending machine

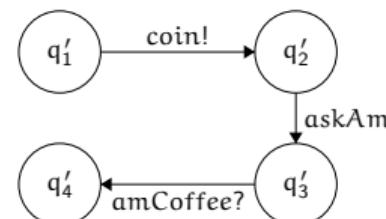
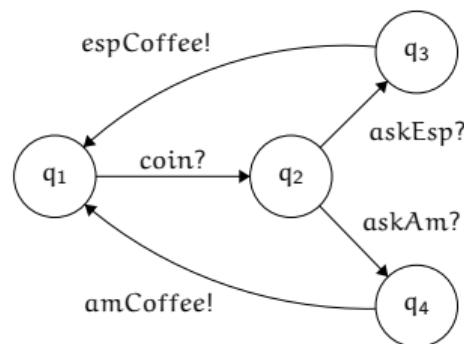
Vending machine LTS:
$$\left(\left\{ q_1, q_2, q_3, q_4 \right\}, \left\{ \begin{array}{l} \text{coin?}, \text{askEsp?}, \\ \text{askAm?}, \text{espCoffee!}, \\ \text{amCoffee!}, \tau \end{array} \right\}, \left\{ \begin{array}{l} (q_1, \text{coin?}, q_2), \\ (q_2, \text{askEsp?}, q_3), \\ (q_2, \text{askAm?}, q_4), \\ (q_3, \text{espCoffee!}, q_1), \\ (q_4, \text{amCoffee!}, q_1) \end{array} \right\} \right)$$



LTS examples: vending machine and user

Vending machine LTS:

$$\left(\left\{ q_1, q_2, \right. \begin{array}{l} \left. q_3, q_4 \right\}, \left\{ \begin{array}{l} \text{coin?}, \text{askEsp?}, \\ \text{askAm?}, \text{espCoffee!}, \end{array} \right. \begin{array}{l} \text{amCoffee!}, \tau \end{array} \right\}, \left\{ \begin{array}{l} (q_1, \text{coin?}, q_2), \\ (q_2, \text{askEsp?}, q_3), \\ (q_2, \text{askAm?}, q_4), \\ (q_3, \text{espCoffee!}, q_1), \\ (q_4, \text{amCoffee!}, q_1) \end{array} \right\} \right)$$



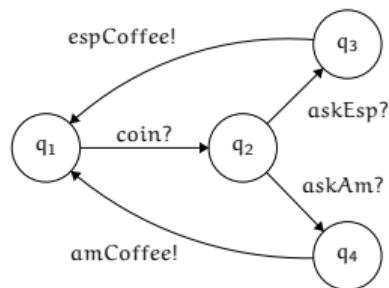
Vending machine user LTS:

$$\left(\left\{ q'_1, q'_2, \right. \begin{array}{l} \left. q'_3, q'_4 \right\}, \left\{ \begin{array}{l} \text{coin!}, \text{askAm!}, \\ \text{amCoffee?}, \tau \end{array} \right. \begin{array}{l} \end{array} \right\}, \left\{ \begin{array}{l} (q'_1, \text{coin!}, q'_2), \\ (q'_2, \text{askAm!}, q'_3), \\ (q'_3, \text{amCoffee?}, q'_1) \end{array} \right\} \right)$$

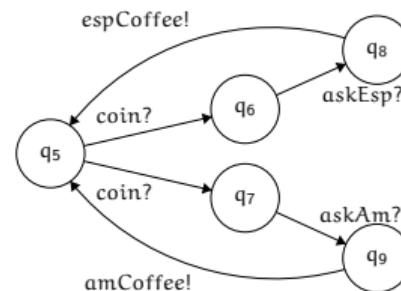
Equivalence between behaviours

Two LTS states may lead to “different” transitions, yet they may “behave the same.”
But what does it mean to have “the same” visible behaviour? It depends!

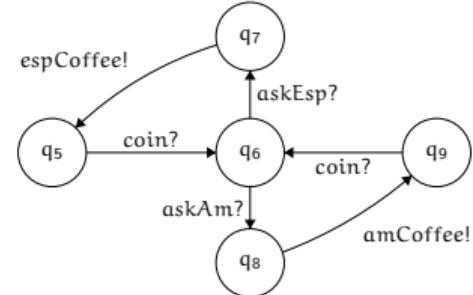
Vending machine



Alternative 1



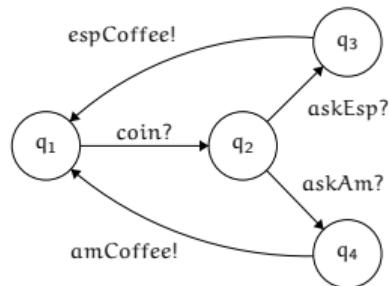
Alternative 2



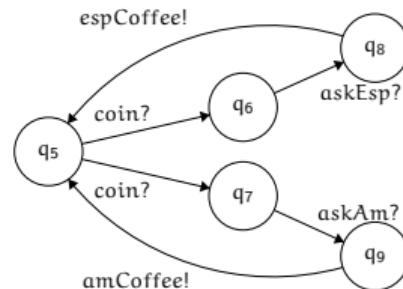
Equivalence between behaviours

Two LTS states may lead to “different” transitions, yet they may “behave the same.”
But what does it mean to have “the same” visible behaviour? It depends!

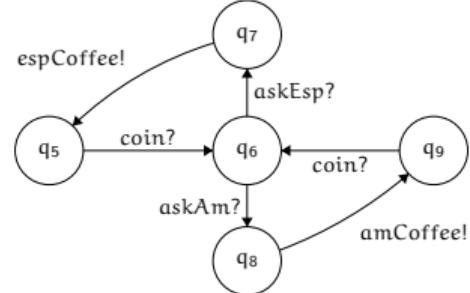
Vending machine



Alternative 1



Alternative 2



Trace equivalence

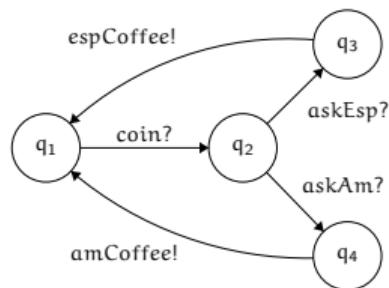
Produce the same sequences of input/output actions



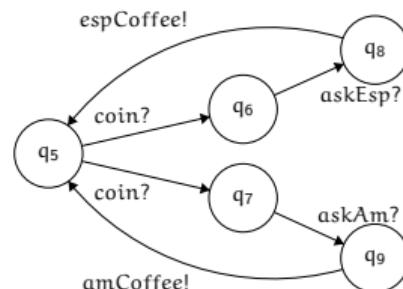
Equivalence between behaviours

Two LTS states may lead to “different” transitions, yet they may “behave the same.”
But what does it mean to have “the same” visible behaviour? It depends!

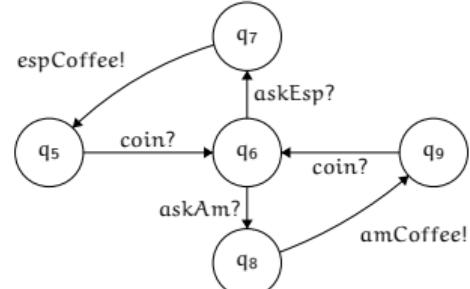
Vending machine



Alternative 1



Alternative 2



Trace equivalence

Produce the same sequences of input/output actions



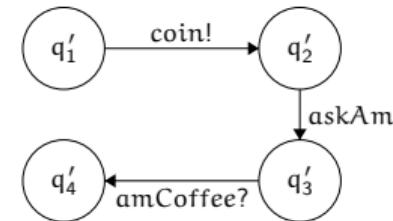
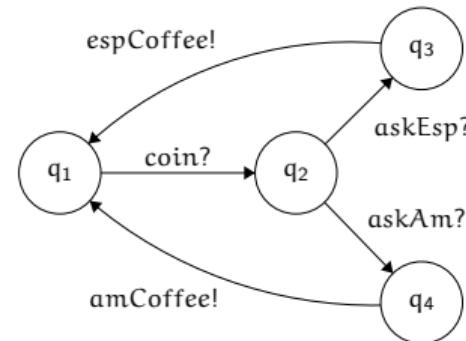
Bisimilarity

Simulate each other's input/output actions throughout each transition



Modelling interaction

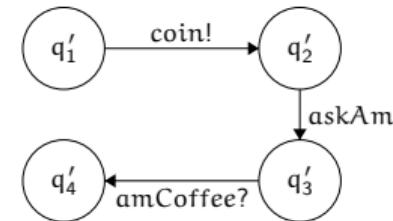
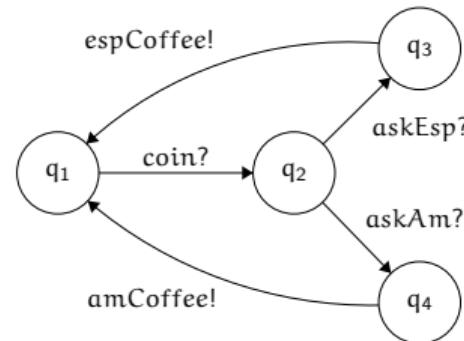
Can we use LTSs to model the **interaction** between concurrent components?



Intuition: we want the user and vending machine to **synchronise** on compatible inputs/outputs, and perform transitions together

Modelling interaction

Can we use LTSs to model the **interaction** between concurrent components?



Intuition: we want the user and vending machine to **synchronise** on compatible inputs/outputs, and perform transitions together

We could do it by suitably **pairing the LTS states**, but:

- ▶ the notation quickly becomes **verbose and cumbersome**
- ▶ we may need to model systems with **infinitely many states**: tricky!

It is better to use a **calculus** to describe concurrent components and their interaction

CCS — the Calculus of Communicating Systems (Robin Milner, 1980)

A calculus for modelling **communication and concurrency** (Robin Milner's book, 1989)

The syntax of CCS

Actions	μ	\coloneqq	a!	output action
			a?	input action
			τ	internal action

Sequential process	P	\coloneqq	0	inaction : a terminated process
			$\mu.Q$	prefix : perform action μ (see above), continue as Q
			$P + P'$	choice : continue as either P or P'

CCS — the Calculus of Communicating Systems (Robin Milner, 1980)

A calculus for modelling **communication and concurrency** (Robin Milner's book, 1989)

The syntax of CCS

Actions	μ	\coloneqq	a!	output action
			a?	input action
			τ	internal action

Sequential process	P	\coloneqq	0	inaction : a terminated process
			$\mu.Q$	prefix : perform action μ (see above), continue as Q
			$P + P'$	choice : continue as either P or P'

Process	Q	\coloneqq	P	a sequential process (see above)
			$Q Q'$	parallel composition of Q and Q'
			$(\text{va})Q$	restriction : action a is private in scope Q
			C	constant : run a process C defined elsewhere

CCS — the Calculus of Communicating Systems (Robin Milner, 1980)

A calculus for modelling **communication and concurrency** (Robin Milner's book, 1989)

The syntax of CCS on <https://pseuco.com/>

Actions	μ	\coloneqq	a!	output action
			a?	input action
			i	internal action

Sequential process	P	\coloneqq	0	inaction : a terminated process
			$\mu.Q$	prefix : perform action μ (see above), continue as Q
			$P + P'$	choice : continue as either P or P'

Process	Q	\coloneqq	P	a sequential process (see above)
			$Q Q'$	parallel composition of Q and Q'
			$Q \setminus \{a\}$	restriction : action a is private in scope Q
			C	constant : run a process C defined elsewhere

The Structural Operational Semantics of CCS

We can interpret CCS processes as **states in an LTS**

The **inference rules** on the right tell us when a **transition** $P \xrightarrow{\mu} Q$ is valid:

$$\frac{\text{premises}}{\text{conclusion}} \text{ [RULENAME]}$$

Meaning: if the *premises* of rule [RULENAME] are true, then its *conclusion* is also true

$$\frac{}{\mu.P \xrightarrow{\mu} P} \text{ [PREF]}$$

$$\frac{C := P \quad P \xrightarrow{\mu} P'}{C \xrightarrow{\mu} P'} \text{ [CONS]}$$

$$\frac{P \xrightarrow{\mu} P'}{P + Q \xrightarrow{\mu} P'} \text{ [SUM1]}$$

$$\frac{Q \xrightarrow{\mu} Q'}{P + Q \xrightarrow{\mu} Q'} \text{ [SUM2]}$$

$$\frac{P \xrightarrow{\mu} P'}{P | Q \xrightarrow{\mu} P' | Q} \text{ [PAR1]}$$

$$\frac{Q \xrightarrow{\mu} Q'}{P | Q \xrightarrow{\mu} P | Q'} \text{ [PAR2]}$$

$$\frac{P \xrightarrow{\alpha} P' \quad Q \xrightarrow{\bar{\alpha}} Q'}{P | Q \xrightarrow{\tau} P' | Q'} \text{ [COM]}$$

$$\frac{\mu \neq a?, a! \quad P \xrightarrow{\mu} P'}{(\nu a)P \xrightarrow{\mu} (\nu a)P'} \text{ [RES]}$$

Live coding with CCS

Experiments on <https://pseuco.com/>: the vending machine and its variants

Main hints:

- ▶ hover or **right-click** on a state or transition action to see its details
- ▶ **click** on a state to expand its transitions
- ▶ select “**Pin dragged nodes**” (bottom-right corner) to reduce LTS wobbling

Exercise (in groups): vending machine 2.0

Use CCS to model a vending machine selling apples or chocolate bars

The system has 3 processes: a **user**, a **selector**, and a **dispenser**

- ▶ The **selector** accepts 2 types of coins from **user**: **coin10** (10 kr) and **coin20** (20 kr)
 - ▶ when the **selector** collects 10 kr., it lets the **user** select **apple**, or insert 10 kr. more
 - ▶ if the **user** selects **apple**, the **selector** tells the **dispenser** to **giveApple**
 - ▶ when the **selector** collects 20 kr., it lets the **user** select **choc**
 - ▶ if the **user** selects **choc**, the **selector** tells the **dispenser** to **giveChoc**
- ▶ The **dispenser** waits for the **selector** to tell either:
 - ▶ **giveApple** — in this case, the **dispenser** outputs **takeApple**
 - ▶ **giveChoc** — in this case, the **dispenser** outputs **takeChoc**
- ▶ The **user** may either:
 - ▶ insert 10 kr. in the **selector**, select **apple**, and then **takeApple** from the **dispenser**
 - ▶ insert 20 kr. in the **selector**, select **choc**, and then **takeChoc** from the **dispenser**

Optional tasks: can you hide **giveApple** and **giveChoc** from the **user**? Can you extend the system to **give change**, and then to sell a **bagel** for 30 kr.?

Exercise 2

Model the system below in CCS. **email your solution to hulo@dtu.dk** if you have questions.

The system has 3 processes: an **online shop**, a **warehouse**, and a **customer**

- ▶ The **shop** allows the **customer** to choose between two items: **jeans** or **shoes**
- ▶ When the **customer** asks to buy an item, the **shop** asks the **warehouse** whether the item is available, by sending **jeansAvailable** or **shoesAvailable**
- ▶ The **warehouse** may answer **yes** or **no**:
 - ▶ if the **warehouse** says **yes**, the **shop** says **available** to the **customer**, then asks for a **shippingAddress**
 - ▶ if the **warehouse** says **no**, the **shop** says **unavailable** to the **customer**

Use Pseuco's CCS Doctor tab to generate the inference tree for the case the **customer** receives the **unavailable** message

Optional: extend the system to support payments with an additional **bank** process:

- ▶ after **shippingAddress**, the **shop** also asks a **creditCardNumber** to the **customer**
- ▶ then, the **shop** requests a **payment** to the **bank**, which can answer either **accepted**

A brief note on CCS semantics

We have seen the **LTS semantics** of CCS, where communication requires 2 rules:

- ▶ rule [REF] to perform an input or output
- ▶ rule [COM] to synchronise matching inputs/outputs

$$\frac{}{\mu.P \xrightarrow{\mu} P} [\text{REF}]$$

$$\frac{P \xrightarrow{\alpha} P' \quad Q \xrightarrow{\bar{\alpha}} Q'}{P | Q \xrightarrow{\tau} P' | Q'} [\text{COM}]$$

A brief note on CCS semantics

We have seen the **LTS semantics** of CCS, where communication requires 2 rules:

- ▶ rule [REF] to perform an input or output
- ▶ rule [COM] to synchronise matching inputs/outputs

$$\frac{}{\mu.P \xrightarrow{\mu} P} [\text{REF}]$$

$$\frac{P \xrightarrow{\alpha} P' \quad Q \xrightarrow{\bar{\alpha}} Q'}{P | Q \xrightarrow{\tau} P' | Q'} [\text{COM}]$$

There is an alternative formalisation called **reduction semantics**. It is **simpler** (one rule!) but it can be **restrictive** (no input/output transitions without communication)

$$\frac{}{a!.P \mid a?.Q \rightarrow P \mid Q} [\text{RED}]$$

Value-Passing CCS

We can extend CCS to transmit **values v** (e.g., numbers, strings, ...): they are sent by outputs, and received by inputs (with a **variable substitution $\{v/x\}$**)

$$\frac{v \text{ is a value}}{a!v.P \xrightarrow{a!v} P} \text{ [OUT]}$$

$$\frac{v \text{ is a value}}{a?x.P \xrightarrow{a?v} P\{v/x\}} \text{ [IN]}$$

$$\frac{P \xrightarrow{a!v} P' \quad Q \xrightarrow{a?v} Q'}{P|Q \xrightarrow{\tau} P'|Q'} \text{ [COM]}$$

With this variation, we are modelling **communication channels** to send/receive values

Value-Passing CCS

We can extend CCS to transmit **values v** (e.g., numbers, strings, ...): they are sent by outputs, and received by inputs (with a **variable substitution $\{v/x\}$**)

$$\frac{v \text{ is a value}}{a!v.P \xrightarrow{a!v} P} \text{ [OUT]}$$

$$\frac{v \text{ is a value}}{a?x.P \xrightarrow{a?v} P\{v/x\}} \text{ [IN]}$$

$$\frac{P \xrightarrow{a!v} P' \quad Q \xrightarrow{a?v} Q'}{P|Q \xrightarrow{\tau} P'|Q'} \text{ [COM]}$$

With this variation, we are modelling **communication channels** to send/receive values

Example: $c!42.0 \mid c?x.c!x.0$

Value-Passing CCS

We can extend CCS to transmit **values v** (e.g., numbers, strings, ...): they are sent by outputs, and received by inputs (with a **variable substitution $\{v/x\}$**)

$$\frac{v \text{ is a value}}{a!v.P \xrightarrow{a!v} P} \text{ [OUT]} \quad \frac{v \text{ is a value}}{a?v.P \xrightarrow{a?v} P\{v/x\}} \text{ [IN]} \quad \frac{P \xrightarrow{a!v} P' \quad Q \xrightarrow{a?v} Q'}{P|Q \xrightarrow{\tau} P'|Q'} \text{ [COM]}$$

With this variation, we are modelling **communication channels** to send/receive values

$$\text{Example: } c!42.\mathbf{0} \mid c?x.c!x.\mathbf{0} \xrightarrow{\tau} \mathbf{0} \mid (c!x.\mathbf{0})\{42/x\}$$

Value-Passing CCS

We can extend CCS to transmit **values v** (e.g., numbers, strings, ...): they are sent by outputs, and received by inputs (with a **variable substitution $\{v/x\}$**)

$$\frac{v \text{ is a value}}{a!v.P \xrightarrow{a!v} P} \text{ [OUT]} \quad \frac{v \text{ is a value}}{a?x.P \xrightarrow{a?v} P\{v/x\}} \text{ [IN]} \quad \frac{P \xrightarrow{a!v} P' \quad Q \xrightarrow{a?v} Q'}{P|Q \xrightarrow{\tau} P'|Q'} \text{ [COM]}$$

With this variation, we are modelling **communication channels** to send/receive values

$$\text{Example: } c!42.0 \mid c?x.c!x.0 \xrightarrow{\tau} 0 \mid c!42.0$$

Value-Passing CCS

We can extend CCS to transmit **values v** (e.g., numbers, strings, ...): they are sent by outputs, and received by inputs (with a **variable substitution $\{v/x\}$**)

$$\frac{v \text{ is a value}}{a!v.P \xrightarrow{a!v} P} \text{ [OUT]}$$

$$\frac{v \text{ is a value}}{a?v.P \xrightarrow{a?v} P\{v/x\}} \text{ [IN]}$$

$$\frac{P \xrightarrow{a!v} P' \quad Q \xrightarrow{a?v} Q'}{P|Q \xrightarrow{\tau} P'|Q'} \text{ [COM]}$$

With this variation, we are modelling **communication channels** to send/receive values

$$\text{Example: } c!42.0 \mid c?x.c!x.0 \xrightarrow{\tau} 0 \mid c!42.0$$

Let's write a **calculator server** on <https://pseuco.com>

Exercise 3

Model the system below in Value-passing CCS. [email to hulo@dtu.dk](mailto:hulo@dtu.dk) if you have questions.

IC is an insurance company which wants to offer a new insurance service for small objects ($\leq 2.000\text{Eur}$) managed completely online for reliable customers. To achieve this, a **customer** who wants his assets to be insured has to provide its **credentials** and the **cost** of the asset with its details (**serial number**, **purchase date**) to **IC**. To ensure that the customer is reliable and the asset inexpensive, IC will then check the customers credentials and past history and estimate the asset's price. If the checks **succeed**, a proposal will be sent back. Otherwise, the request will be **rejected**.

The π -calculus (Milner, Parrow and Walker, 1991)

The π -calculus is similar to value-passing CCS — plus, it can **send / receive channels over channels**



The π -calculus

 (Milner, Parrow and Walker, 1991)

The π -calculus is similar to value-passing CCS — plus, it can **send / receive channels over channels**

π -calculus processes can **change their communication topology**. Example: a **server** controls access to a **printer** (channel **a**), and is connected (via channel **b**) to a **client** who wants to print a file. The server can **send a over b**



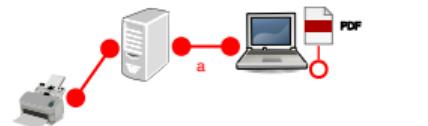
$a?y.P \mid b!a.Q \mid b?x.x!“PDF”.0$

The π -calculus

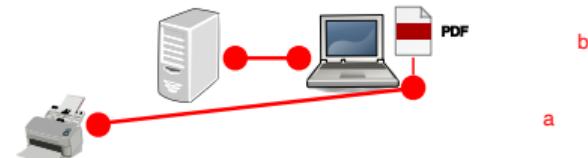
 (Milner, Parrow and Walker, 1991)

The π -calculus is similar to value-passing CCS — plus, it can **send / receive channels over channels**

π -calculus processes can **change their communication topology**. Example: a **server** controls access to a **printer** (channel **a**), and is connected (via channel **b**) to a **client** who wants to print a file. The server can **send a over b**



$a?y.P \mid b!a.Q \mid b?x.x!“PDF”.0$



$a?y.P \mid Q \mid a!“PDF”.0$

The π -calculus (continued)

The π -calculus is similar to value-passing CCS — plus, it can **send / receive channels over channels**



The π -calculus (continued)

The π -calculus is similar to value-passing CCS — plus, it can **send / receive channels over channels**

Unlike CCS, π -calculus processes can **connect on a public channel, exchange a new private channel, and continue interacting on the private channel**. This is obtained via **scope extrusion**



The π -calculus (continued)

The π -calculus is similar to value-passing CCS — plus, it can **send / receive channels over channels**

Unlike CCS, π -calculus processes can **connect on a public channel, exchange a new private channel, and continue interacting on the private channel**. This is obtained via **scope extrusion**

For example, consider:

$$\begin{aligned}\text{Calc} &:= \text{public?}z . z?x . z?y . z!(x * y) . \text{Calc} \\ \text{Client} &:= (\text{vs})(\text{public!}s . s!2 . s!3 . s?w . 0)\end{aligned}$$



The π -calculus (continued)

The π -calculus is similar to value-passing CCS — plus, it can **send / receive channels over channels**

Unlike CCS, π -calculus processes can **connect on a public channel, exchange a new private channel, and continue interacting on the private channel**. This is obtained via **scope extrusion**

For example, consider:

$$\begin{aligned}\text{Calc} &:= \text{public?}z . z?x . z?y . z!(x * y) . \text{Calc} \\ \text{Client} &:= (\text{vs})(\text{public!}s . s!2 . s!3 . s?w . 0)\end{aligned}$$

Calc | Client



The π -calculus (continued)

The π -calculus is similar to value-passing CCS — plus, it can **send / receive channels over channels**

Unlike CCS, π -calculus processes can **connect on a public channel, exchange a new private channel, and continue interacting on the private channel**. This is obtained via **scope extrusion**

For example, consider:

$$\begin{aligned} \text{Calc} &:= \text{public?}z . z?x . z?y . z!(x * y) . \text{Calc} \\ \text{Client} &:= (\text{vs})(\text{public!}s . s!2 . s!3 . s?w . 0) \end{aligned}$$

$$\text{Calc} \mid \text{Client} \xrightarrow{\tau} (\text{vs})(s?x . s?y . s!(x * y) . \text{Calc} \mid s!2 . s!3 . s?w . 0)$$



The π -calculus (continued)

The π -calculus is similar to value-passing CCS — plus, it can **send / receive channels over channels**

Unlike CCS, π -calculus processes can **connect on a public channel, exchange a new private channel, and continue interacting on the private channel**. This is obtained via **scope extrusion**

For example, consider:

$$\begin{aligned} \text{Calc} &:= \text{public?}z . z?x . z?y . z!(x * y) . \text{Calc} \\ \text{Client} &:= (\nu s)(\text{public!}s . s!2 . s!3 . s?w . 0) \end{aligned}$$

$$\begin{aligned} \text{Calc} \mid \text{Client} &\xrightarrow{\tau} (\nu s)(s?x . s?y . s!(x * y) . \text{Calc} \mid s!2 . s!3 . s?w . 0) \\ &\xrightarrow{\tau} (\nu s)(s?y . s!(2 * y) . \text{Calc} \mid s!3 . s?w . 0) \end{aligned}$$



The π -calculus (continued)

The π -calculus is similar to value-passing CCS — plus, it can **send / receive channels over channels**

Unlike CCS, π -calculus processes can **connect on a public channel, exchange a new private channel, and continue interacting on the private channel**. This is obtained via **scope extrusion**

For example, consider:

$$\begin{aligned}\text{Calc} &:= \text{public?}z . z?x . z?y . z!(x * y) . \text{Calc} \\ \text{Client} &:= (\text{vs})(\text{public!}s . s!2 . s!3 . s?w . \text{0})\end{aligned}$$

$$\begin{aligned}\text{Calc} | \text{Client} &\xrightarrow{\tau} (\text{vs})(s?x . s?y . s!(x * y) . \text{Calc} | s!2 . s!3 . s?w . \text{0}) \\ &\xrightarrow{\tau} (\text{vs})(s?y . s!(2 * y) . \text{Calc} | s!3 . s?w . \text{0}) \\ &\xrightarrow{\tau} (\text{vs})(s!(2 * 3) . \text{Calc} | s?w . \text{0})\end{aligned}$$



The π -calculus (continued)

The π -calculus is similar to value-passing CCS — plus, it can **send / receive channels over channels**

Unlike CCS, π -calculus processes can **connect on a public channel, exchange a new private channel, and continue interacting on the private channel**. This is obtained via **scope extrusion**

For example, consider:

$$\begin{aligned}\text{Calc} &:= \text{public?}z . z?x . z?y . z!(x * y) . \text{Calc} \\ \text{Client} &:= (\text{vs})(\text{public!}s . s!2 . s!3 . s?w . \text{0})\end{aligned}$$

$$\begin{aligned}\text{Calc} | \text{Client} &\xrightarrow{\tau} (\text{vs})(s?x . s?y . s!(x * y) . \text{Calc} | s!2 . s!3 . s?w . \text{0}) \\ &\xrightarrow{\tau} (\text{vs})(s?y . s!(2 * y) . \text{Calc} | s!3 . s?w . \text{0}) \\ &\xrightarrow{\tau} (\text{vs})(s!(2 * 3) . \text{Calc} | s?w . \text{0}) \\ &\xrightarrow{\tau} (\text{vs})(\text{Calc} | \text{0})\end{aligned}$$



Locks and Deadlocks

The notions of deadlock and livelock adapt easily to the π -calculus (being a calculus specially designed for distributed systems).

A canonical example of a deadlock is given by the following process:

$$a?x . b!x \mid b?y . a!y \not\rightarrow$$

Locks and Deadlocks

The notions of deadlock and livelock adapt easily to the π -calculus (being a calculus specially designed for distributed systems).

A canonical example of a deadlock is given by the following process:

$$a?x . b!x \mid b?y . a!y \not\rightarrow$$

An example of a locked process that is not a deadlock is

$$c!a \mid *c?x . c!x \mid a!1984$$

Where $*P$ represents the **replicated** process, defined as $*P := P \mid *P$.

Locks and Deadlocks

The notions of deadlock and livelock adapt easily to the π -calculus (being a calculus specially designed for distributed systems).

A canonical example of a deadlock is given by the following process:

$$a?x . b!x \mid b?y . a!y \not\rightarrow$$

An example of a locked process that is not a deadlock is

$$c!a \mid *c?x . c!x \mid a!1984$$

Where $*P$ represents the **replicated** process, defined as $*P := P \mid *P$.

The difference lies on the possible reductions. A deadlocked process contains cyclic dependencies and cannot reduce. A locked process might be able to perform reductions, but not all the ones required (a is never synchronized). In other words, the system does not **progress**

Further reading

Roberto Gorrieri and Cristian Versari. *Introduction to Concurrency Theory: Transition Systems and CCS*. Springer, 2015. <https://findit.dtu.dk/en/catalog/2688358590>

- ▶ Chapter 2: labelled transition systems, trace equivalence, bisimilarity
- ▶ Chapter 3: CCS and value-passing CCS

Joachim Parrow. *An Introduction to the π -Calculus*. Book chapter, Elsevier, 2001.

<https://findit.dtu.dk/en/catalog/2608173696>

- ▶ Section 1: introduction
- ▶ Section 2: basic definitions and examples

Your feedback is important!

Please take this **brief anonymous survey**
(if asked to log in, use your DTU email)



<https://forms.office.com/e/0zeRwpTw6m>

System Integration - Timed Automata

Hugo A. López

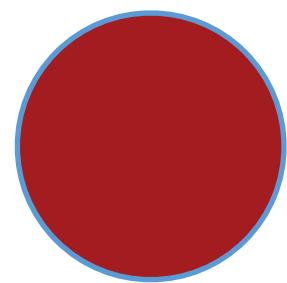
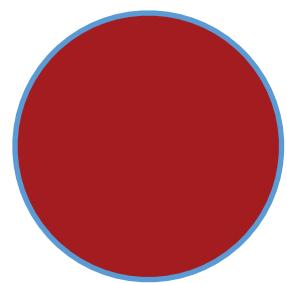
hulo@dtu.dk

A little recap

This course so far

+ Operational

Easier to verify



+ Abstract

Harder to verify

This course so far

+ Operational

Easier to verify

+ Abstract

Harder to verify



This course so far

+ Operational

Easier to verify

+ Abstract

Harder to verify



Automata

Computation

This course so far

+ Operational

Easier to verify

+ Abstract

Harder to verify



Automata

Workflow Models

Computation

Task distribution

This course so far

+ Operational
Easier to verify

+ Abstract
Harder to verify



Automata

Workflow Models

Interaction Models

Computation

Task distribution

Handovers

This course so far

+ Operational
Easier to verify

+ Abstract
Harder to verify



Automata

Workflow Models

Interaction Models

Requirement Models

Computation

Task distribution

Handovers

Strategy

This course so far

+ Operational
Easier to verify

+ Abstract
Harder to verify



Automata

Workflow Models

Interaction Models

Requirement Models

Enterprise
Architecture
Models

Computation

Task distribution

Handovers

Strategy

Technology
Alignment

This class

+ Operational
Easier to verify

+ Abstract
Harder to verify



Automata

Workflow Models

Interaction Models

Real-time models

Requirement Models

Enterprise
Architecture
Models

Computation

Task distribution

Handovers

Temporal Dependencies

Strategy

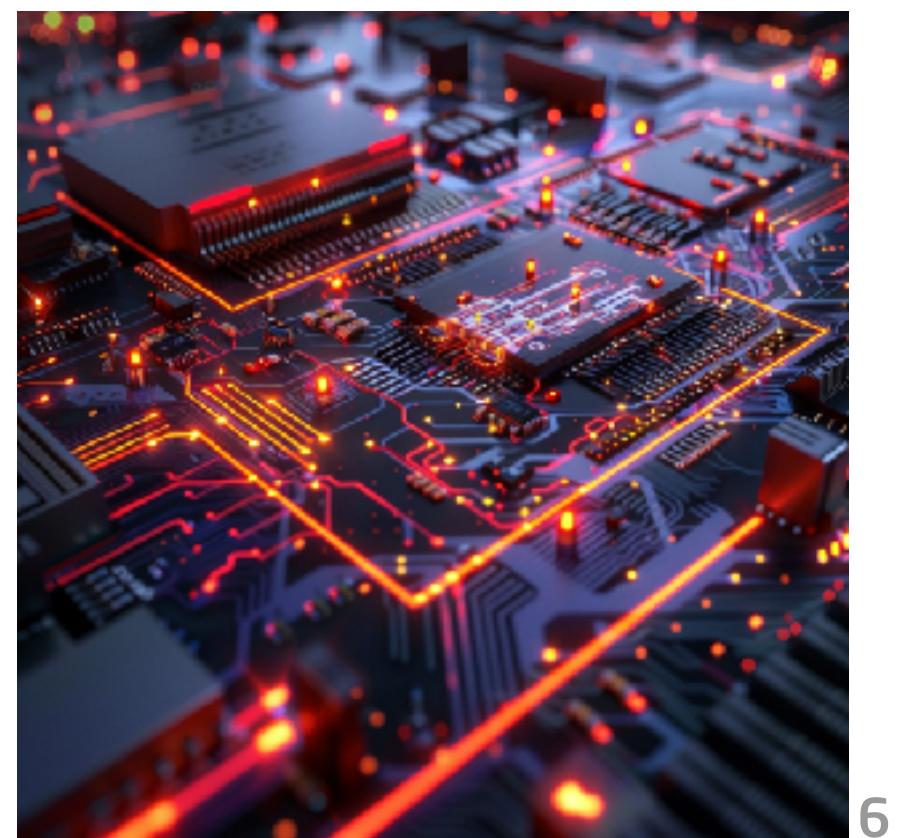
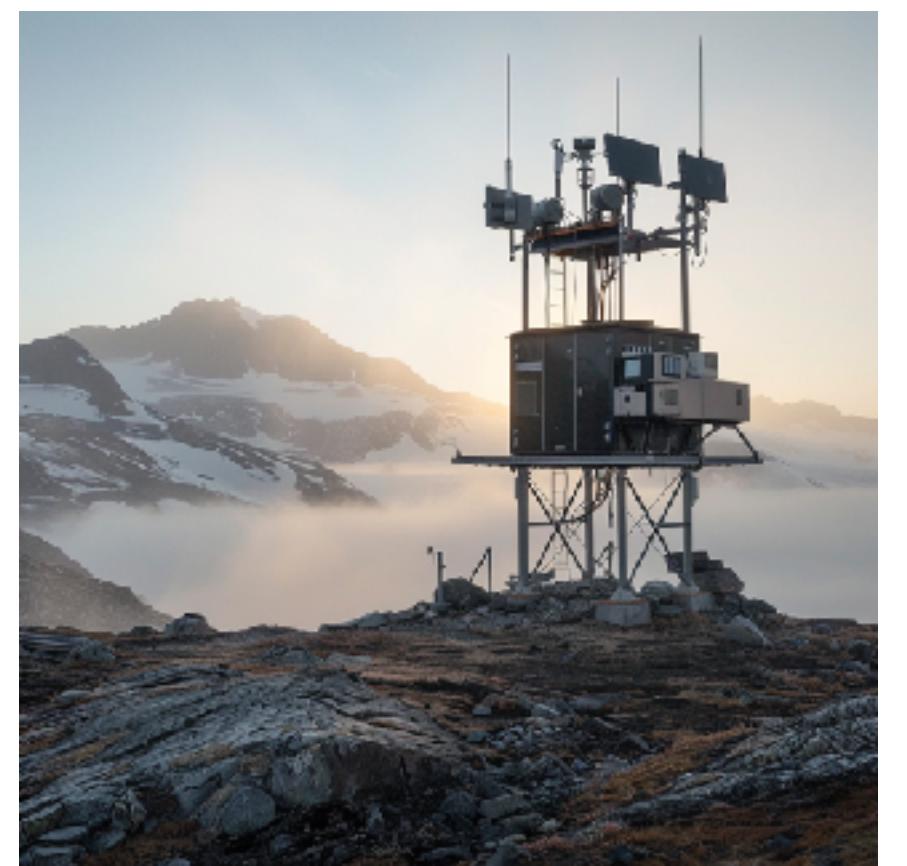
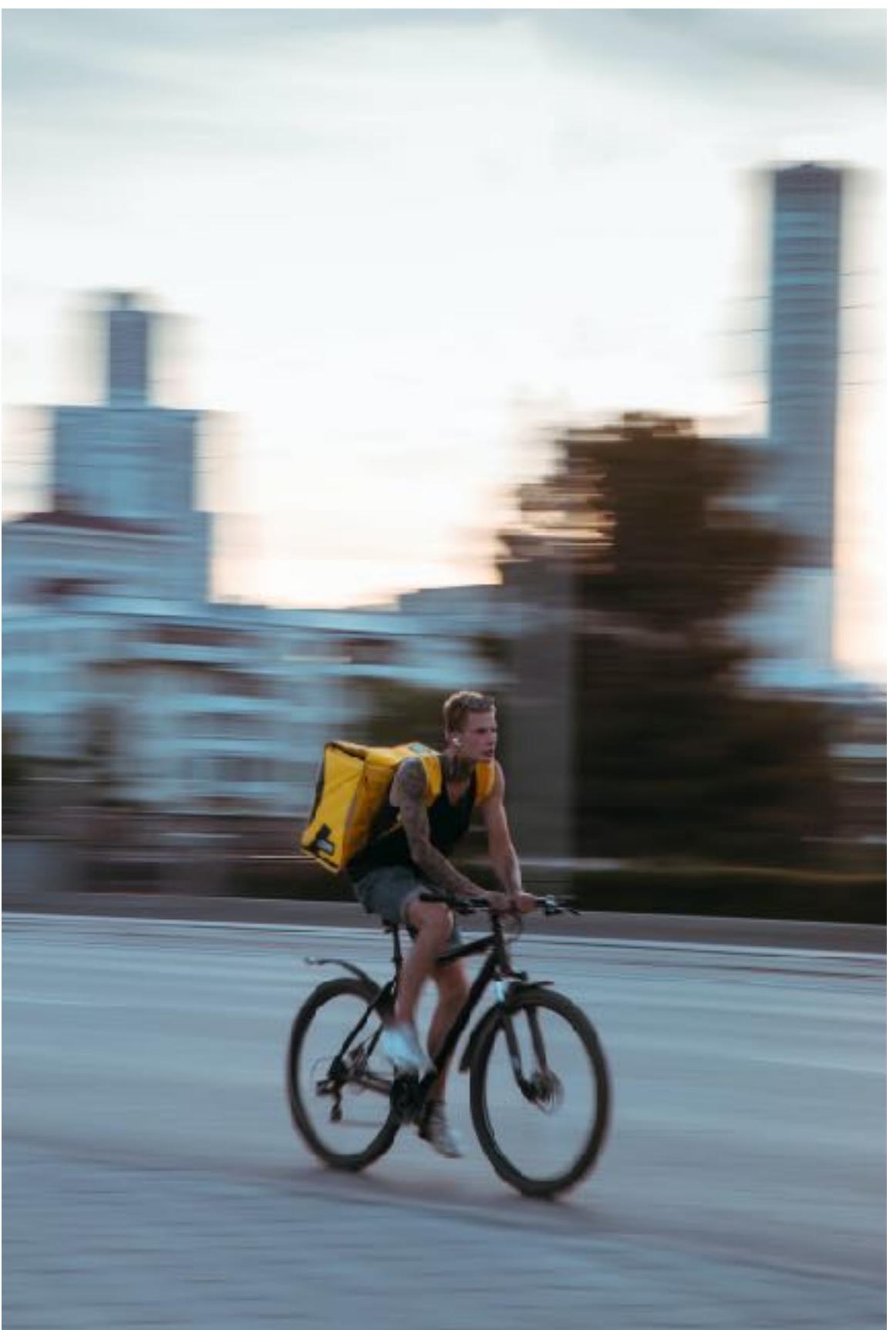
Technology
Alignment

Why real-time systems

Real-time systems

- Parts of the socio-technical system are expected to finish within reliable time bounds
- Communication with microservices does not run forever, but under standard service level agreements
- Fault-tolerant systems need to model availability considerations under absences of responses (failures)

A system where correctness does not only depend on the logical order of events, but also on their **timing**

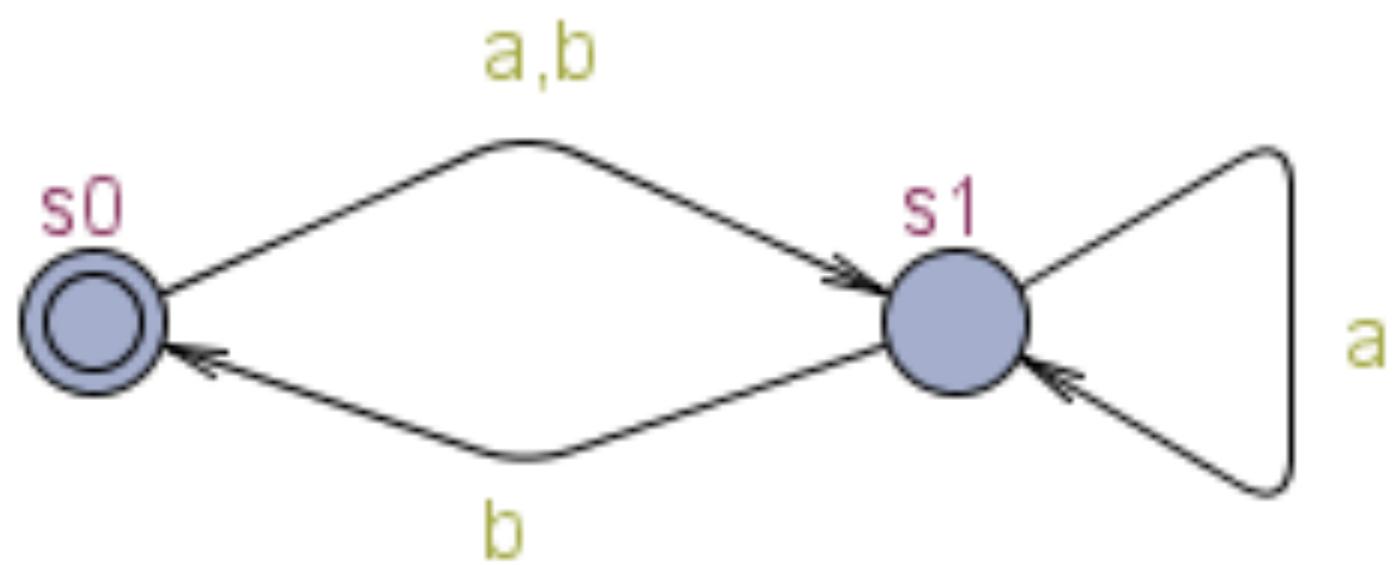


How does time influence our specification?

- Evolution

How does time influence our specification?

- Evolution

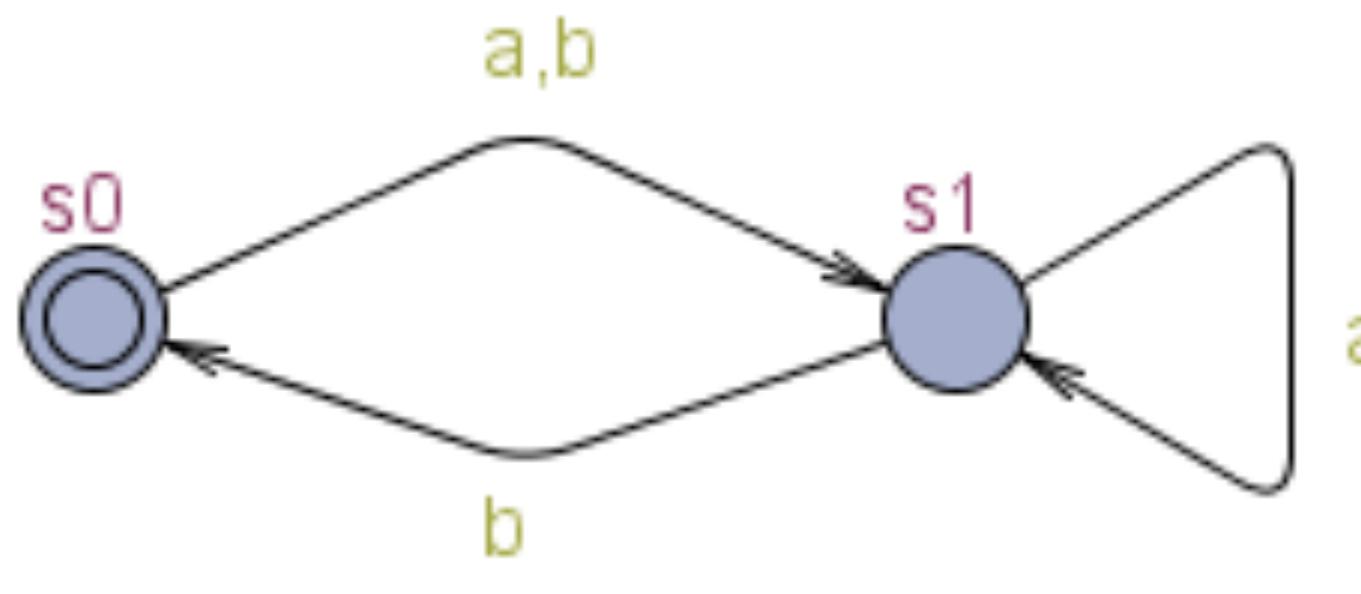


Büchi Automata

- Infinite alphabet
- Initial and accepting states
- Accept execution if pass through accepting state infinitely many times

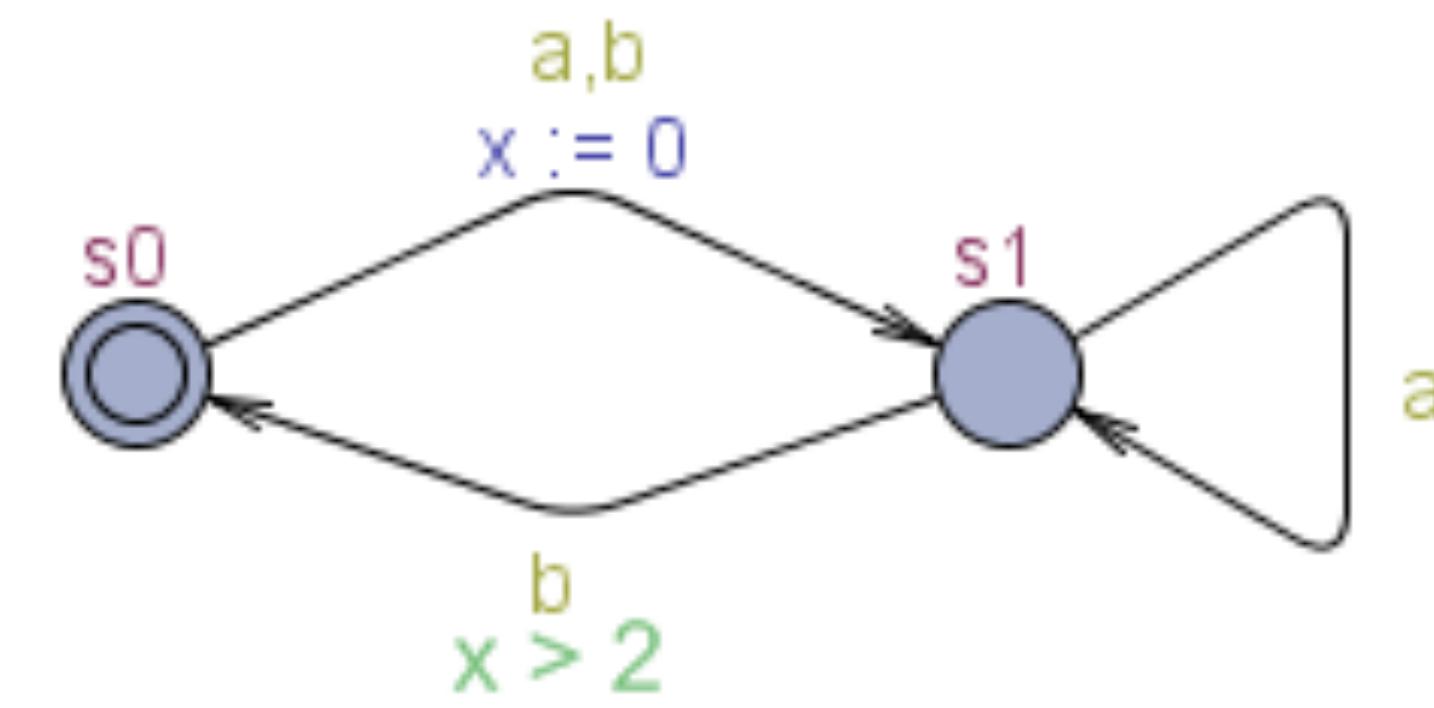
How does time influence our specification?

- Evolution



Büchi Automata

- Infinite alphabet
- Initial and accepting states
- Accept execution if pass through accepting state infinitely many times

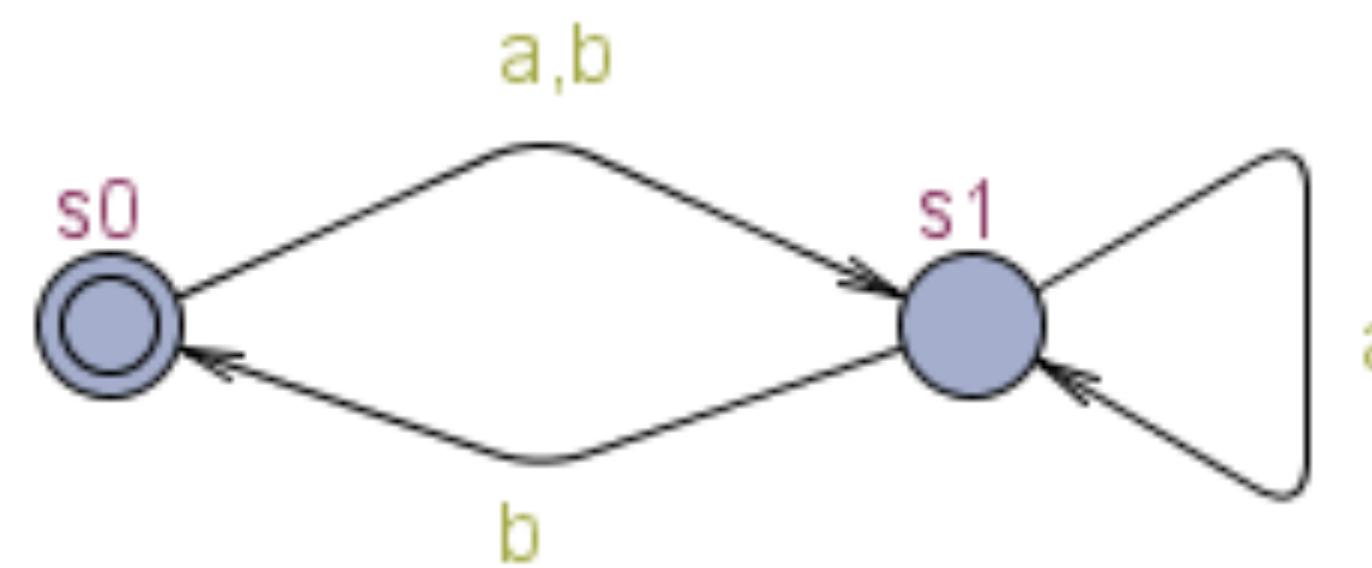


Büchi Timed Automata

- Büchi-accepting
- Real-valued variables: modelling clock
- Constraints on clock variables and resets

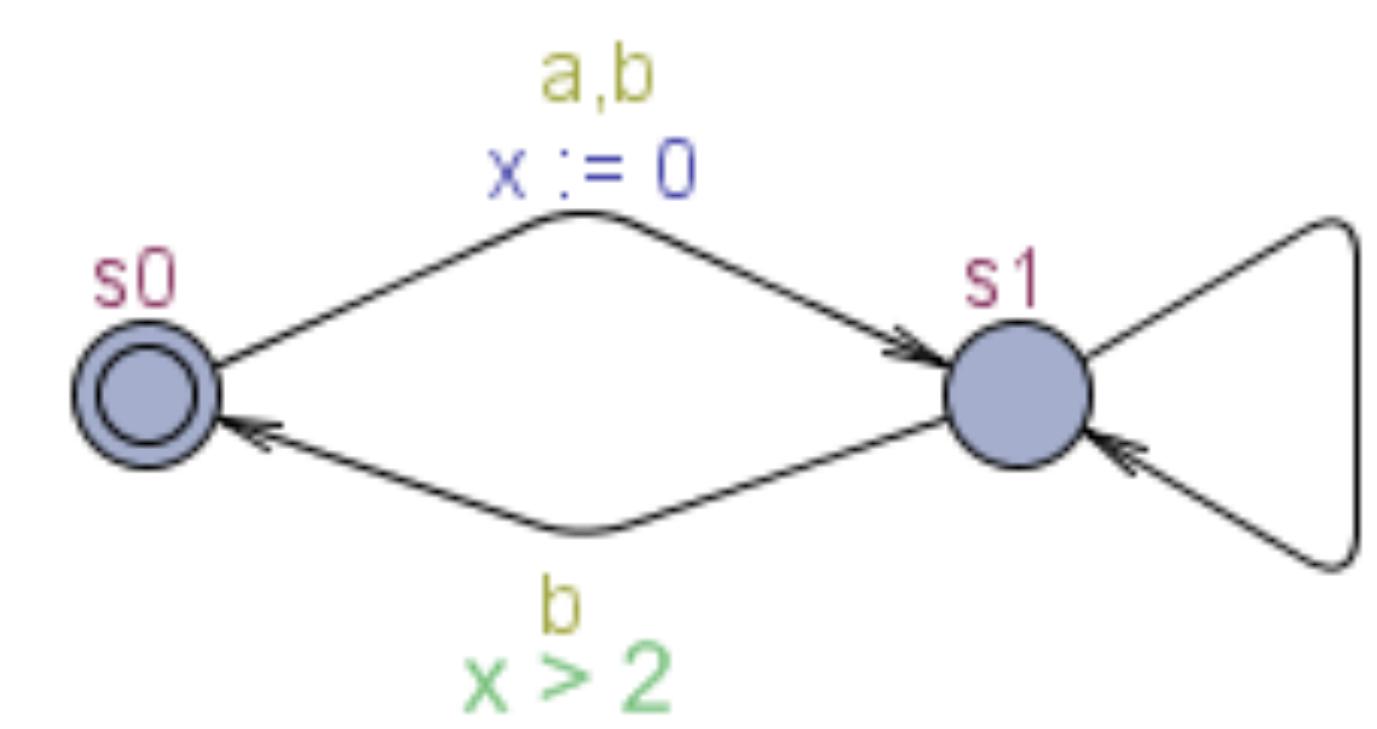
How does time influence our specification?

- Evolution



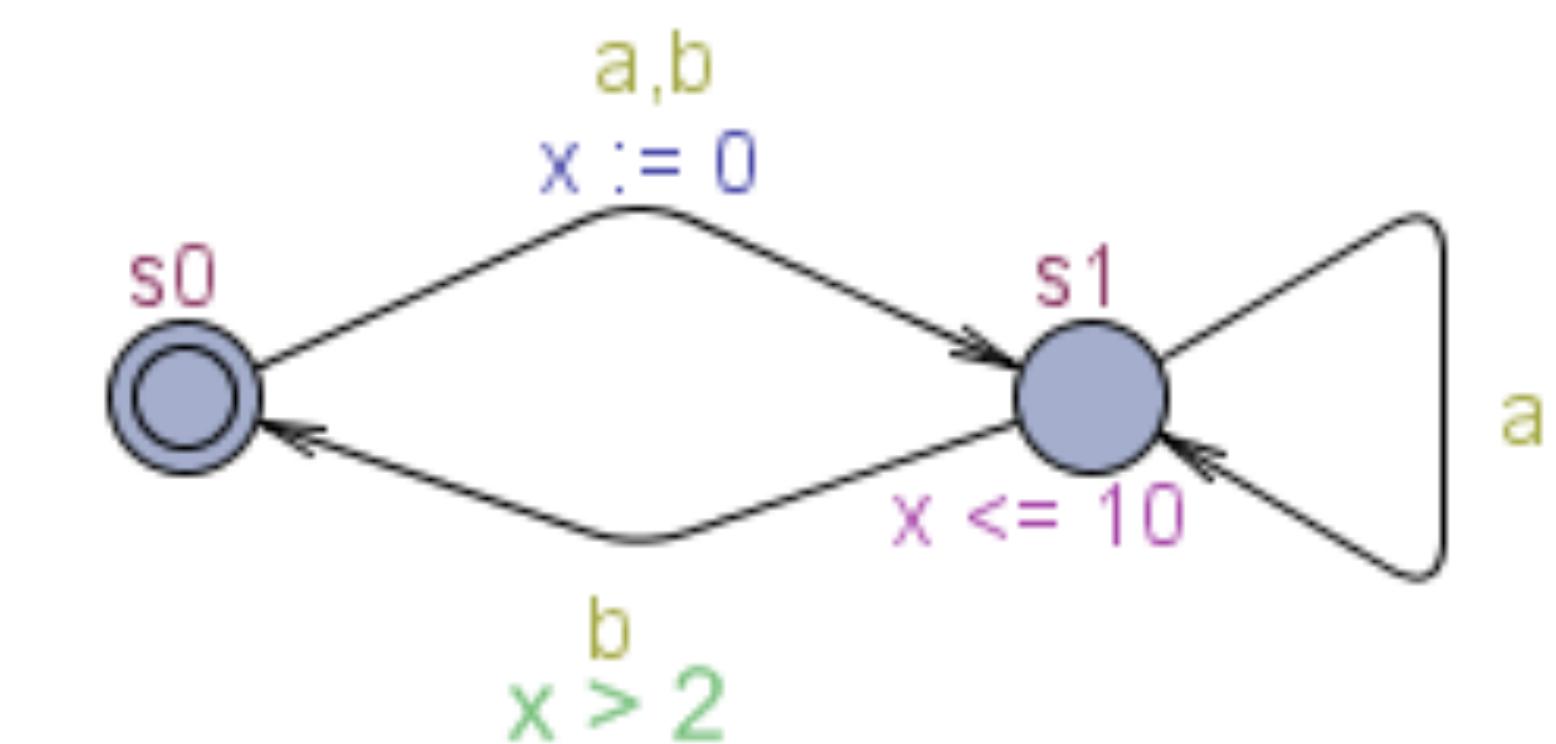
Büchi Automata

- Infinite alphabet
- Initial and accepting states
- Accept execution if pass through accepting state infinitely many times



Büchi Timed Automata

- Büchi-accepting
- Real-valued variables: modelling clock
- Constraints on clock variables and resets



Timed Safety Automata

- Clock variables
- Local invariant conditions
- Accept when invariant is satisfied

Timed Automata

- A timed automaton is a tuple

$$(\mathcal{L}, l_0, C, \mathcal{A}, \mathcal{E}, \mathcal{I})$$

- where \mathcal{L} is a set of locations,
- $l_0 \in \mathcal{L}$ is the initial location,
- C is the set of clocks,
- \mathcal{A} is a set of actions, co-actions and the internal τ -action,
- $\mathcal{E} \subseteq \mathcal{L} \times \mathcal{A} \times \mathcal{B}(C) \times 2^C \times \mathcal{L}$ is a set of edges between locations with an action, a guard and a set of clocks to be reset, and
- $\mathcal{I} : \mathcal{L} \rightarrow \mathcal{B}(C)$ assigns invariants to locations

Semantics

The operational Semantics of a timed automaton is:

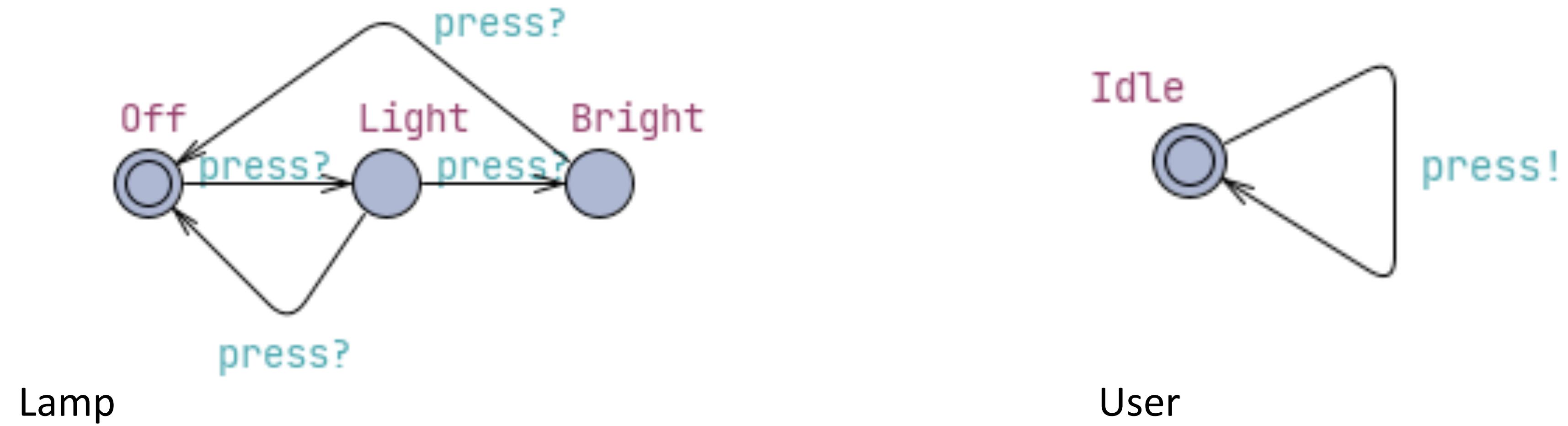
- If $u, u + d \in I(l)$ and $d \in \mathbb{R}^+$,
then $\langle l, u \rangle \xrightarrow{d} \langle l, u + d \rangle$ Timed action
 - If $l \xrightarrow{\tau, \alpha, r} l'$, $u \in g$, $u' = [r \mapsto 0]u$ and $u' \in I(l')$,
then $\langle l, u \rangle \xrightarrow{\alpha} \langle l', u' \rangle$ Labelled action
- Notation: $\langle l, u \rangle$ is a state
 $\langle l, u \rangle \xrightarrow{\alpha} \langle l', u' \rangle$ is a transition

Operational Semantics

- For a TA $(\mathcal{L}, \ell, C, \mathcal{A}, \mathcal{E}, \mathcal{I})$, its semantics is given in terms of an LTS $\langle S, s_o, \rightarrow \rangle$, where
 - $S \subseteq \mathcal{L} \times \mathcal{R}^C$ is the set of states,
 - $s_o = (\ell_o, u_o)$ is the initial state, and
 - $\rightarrow \subseteq S \times \{\mathcal{R}_{\geq 0} \cup \mathcal{A}\} \times S$ is the transition relation such that:
 - $(\ell, u) -d \rightarrow (\ell, u+d)$ if $\forall d' : o \leq d' \leq d \Rightarrow u+d' \in \mathcal{I}(\ell)$, and
 - $(\ell, u) -a \rightarrow (\ell', u')$ if there exists $e = (\ell, a, g, r, \ell') \in \mathcal{E}$ such that
 - $u \in g$,
 - $u' = [r| \rightarrow o]u$, and
 - $u' \in \mathcal{I}(\ell')$

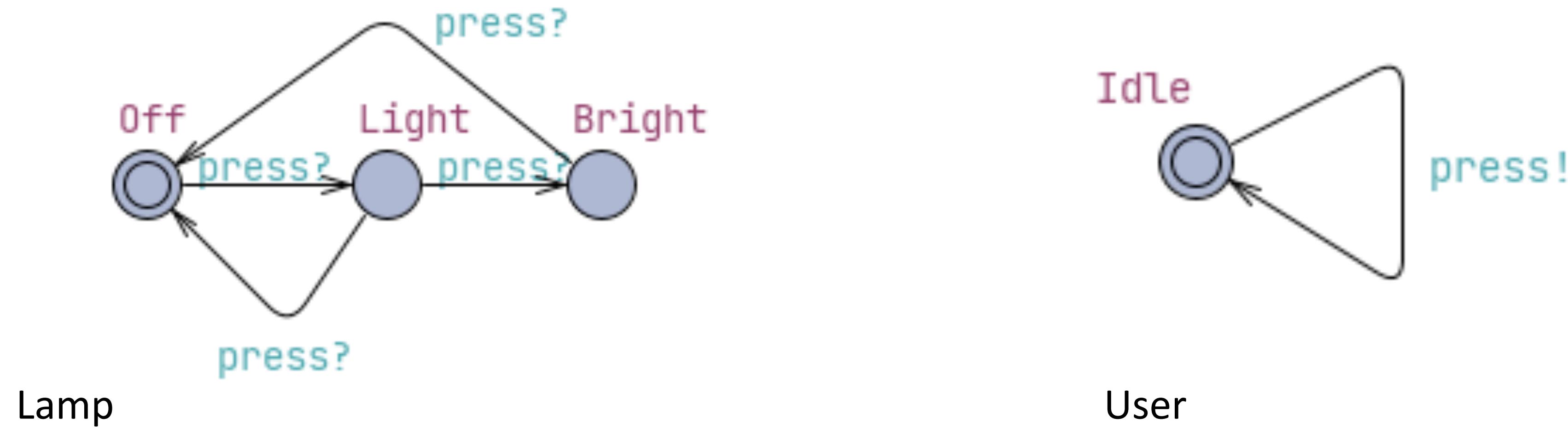
The transition respect timed invariants

Our first CPS



- System = parallel composition of lamp and user

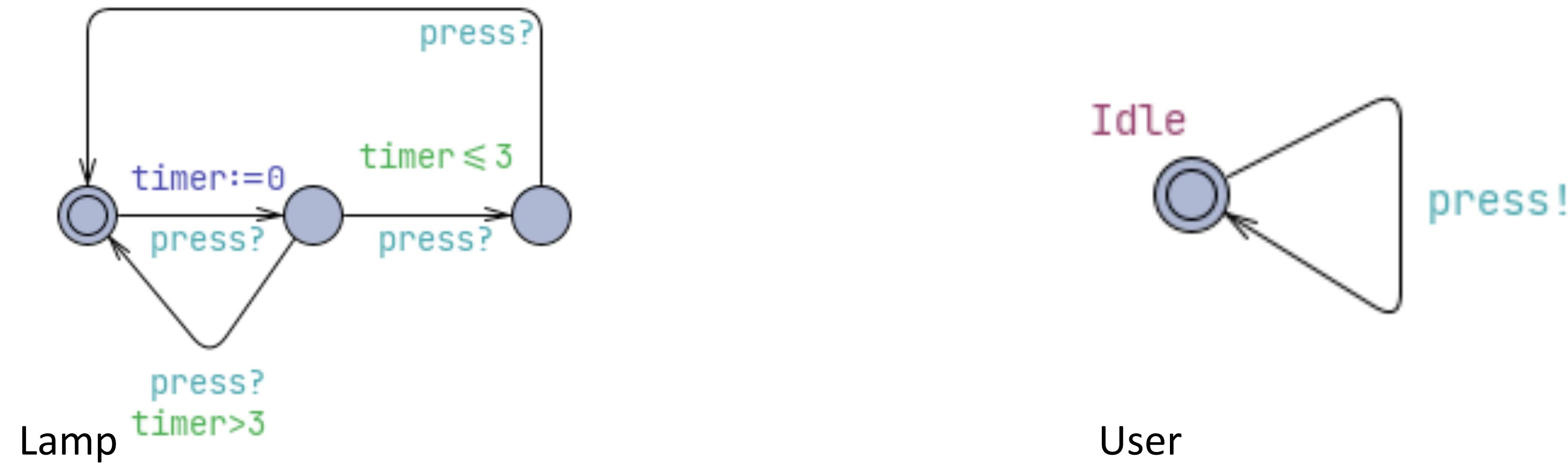
Our first CPS



- System = parallel composition of lamp and user

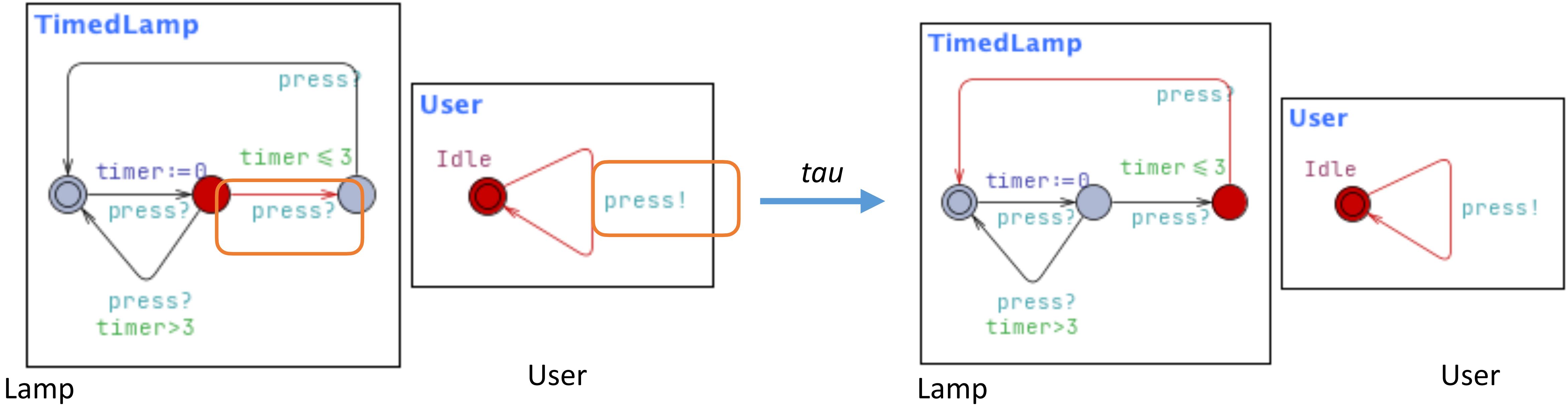
Question: how can we model the policy that the light should be dimmable if the user presses **quickly**?

Adding timers & guards



- A real-value timer measures the delay between the *press* events
- Clocks can be resetted, and queried
- Multiple clocks can be instantiated

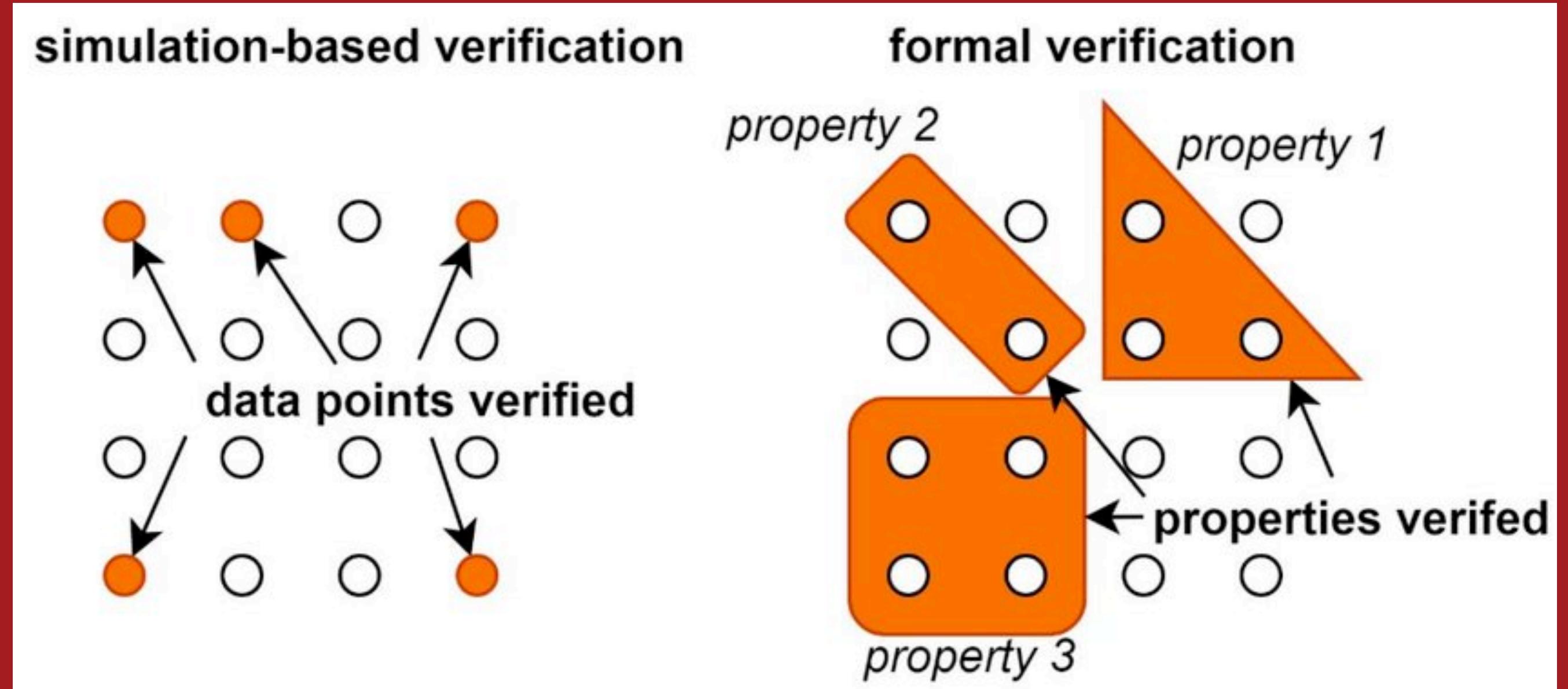
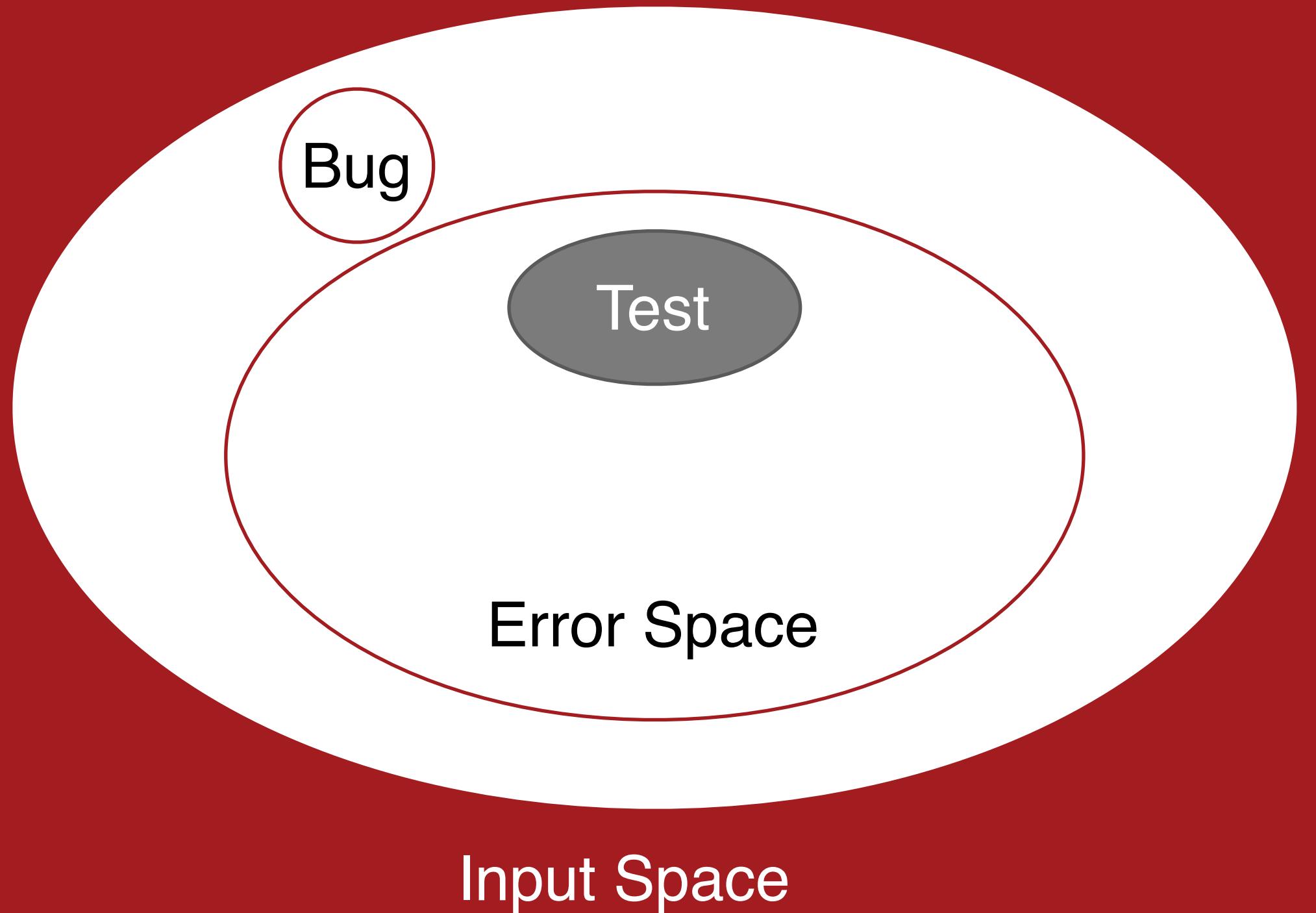
Networks of Timed Automata



- A network of timed automata provides a two-way synchronization between co-actions
 - Akin to CCS!

Extensions

- See tutorial for more details
 - Urgent channels: no delay if transition with urgent action can be taken.
 - Committed locations: reduce the number of clocks and disallow delays between committed locations
 - Broadcast channels: one-to-many communication
 - Parameterised templates: allow to span finite copies of a model



Testing and Verification

A testing approach does not guarantee a fail-free implementation. It guarantees that the software passes the tests you have provided.
If you need full-assurance, you need higher guarantees, for instance, formal methods

Verification of Timed Models

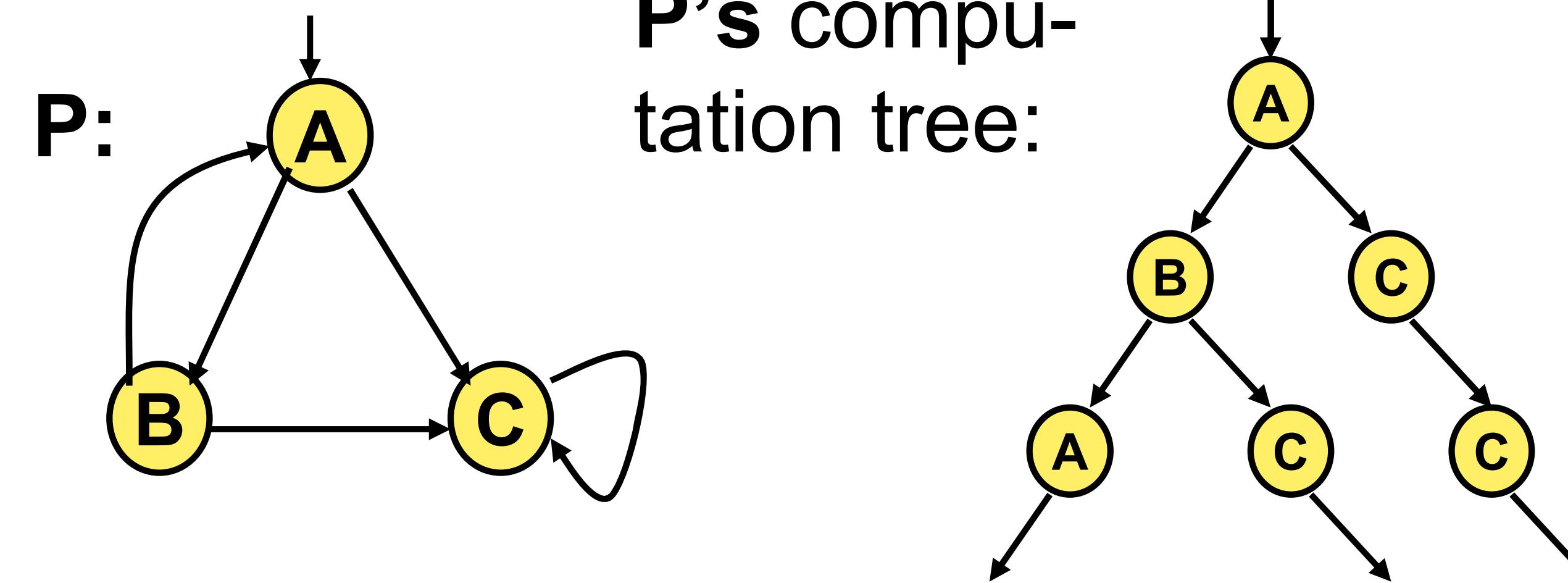
- Instead of running tests, we would like to apply **model-checking**
- Types of properties:
 - Safety: “**Something bad never happens**” [Lamport 1977]
 - Type 1: a broken invariant in the specification as state properties that fail
 - E.g. A mutual exclusion algorithm that allows more than 1 processors in the critical section
 - Every invariant is a safety property, but not the reverse
 - For e.g. "vending machine should get money **before** dispensing a product" is not a state property, but a **temporal property**

Liveness

- Doing nothing easily fulfils a safety property as this will never lead to a “bad” situation
 - For e.g. “vending machine should get money **before** dispensing a product” not buying a product satisfy the property
- Liveness properties complement safety properties and require progress
 - ‘`something good” will happen eventually [Lamport 1977]
- Liveness properties are violated in “infinite time”
 - whereas safety properties are violated in finite time
 - finite traces are of no use to decide whether P_{live} holds or not
 - any finite prefix can be extended such that the resulting infinite trace satisfies P_{live}
 - Examples: fairness

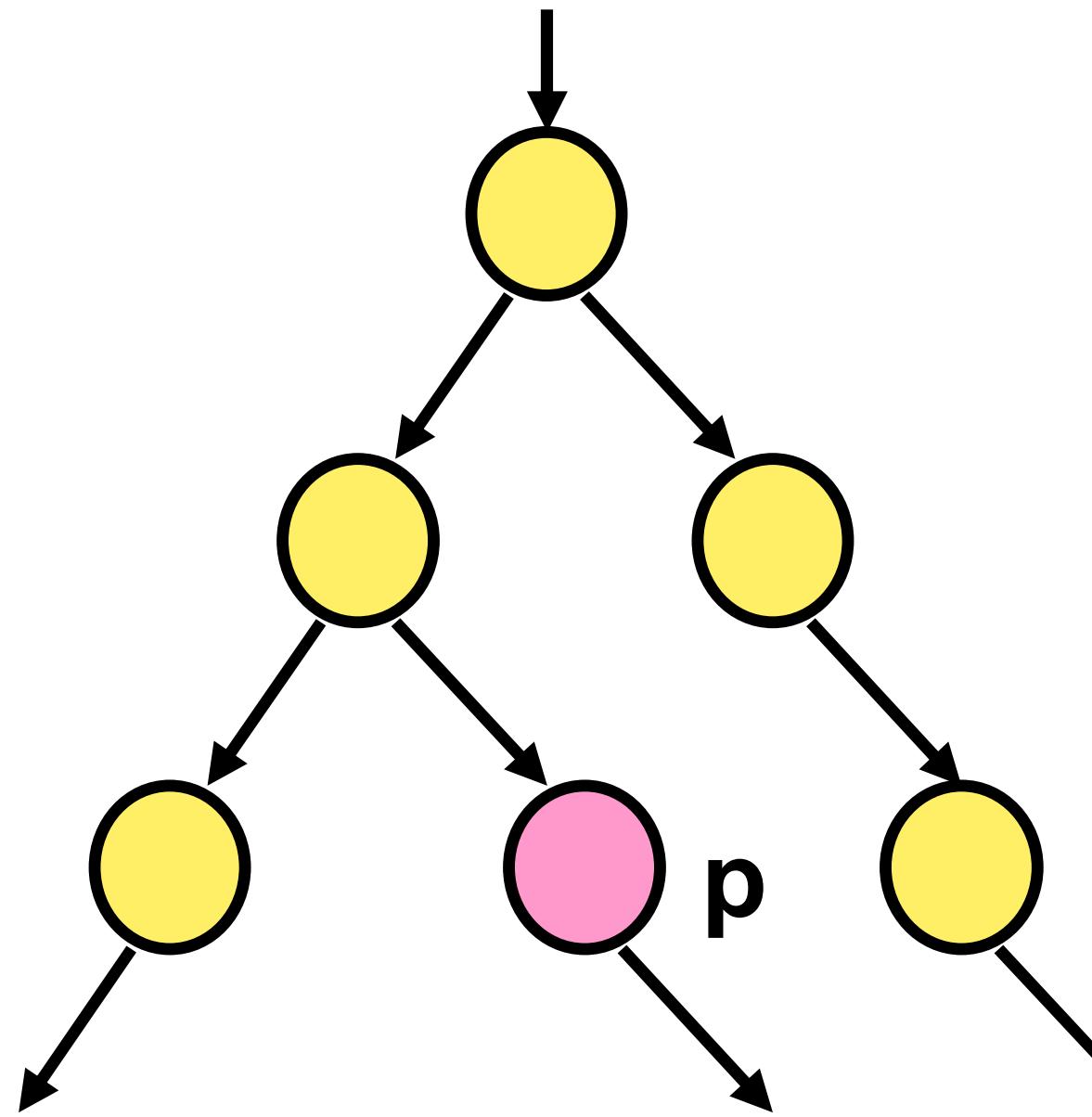
Verification in Uppaal

- A model checker verifies whether a model respects a requirement
- UPPAAL uses a simplified version of CTL [1] (temporal first-order logic)
- State formulae
- Path formulae: reachability, safety, liveness



Formulae in TCTL

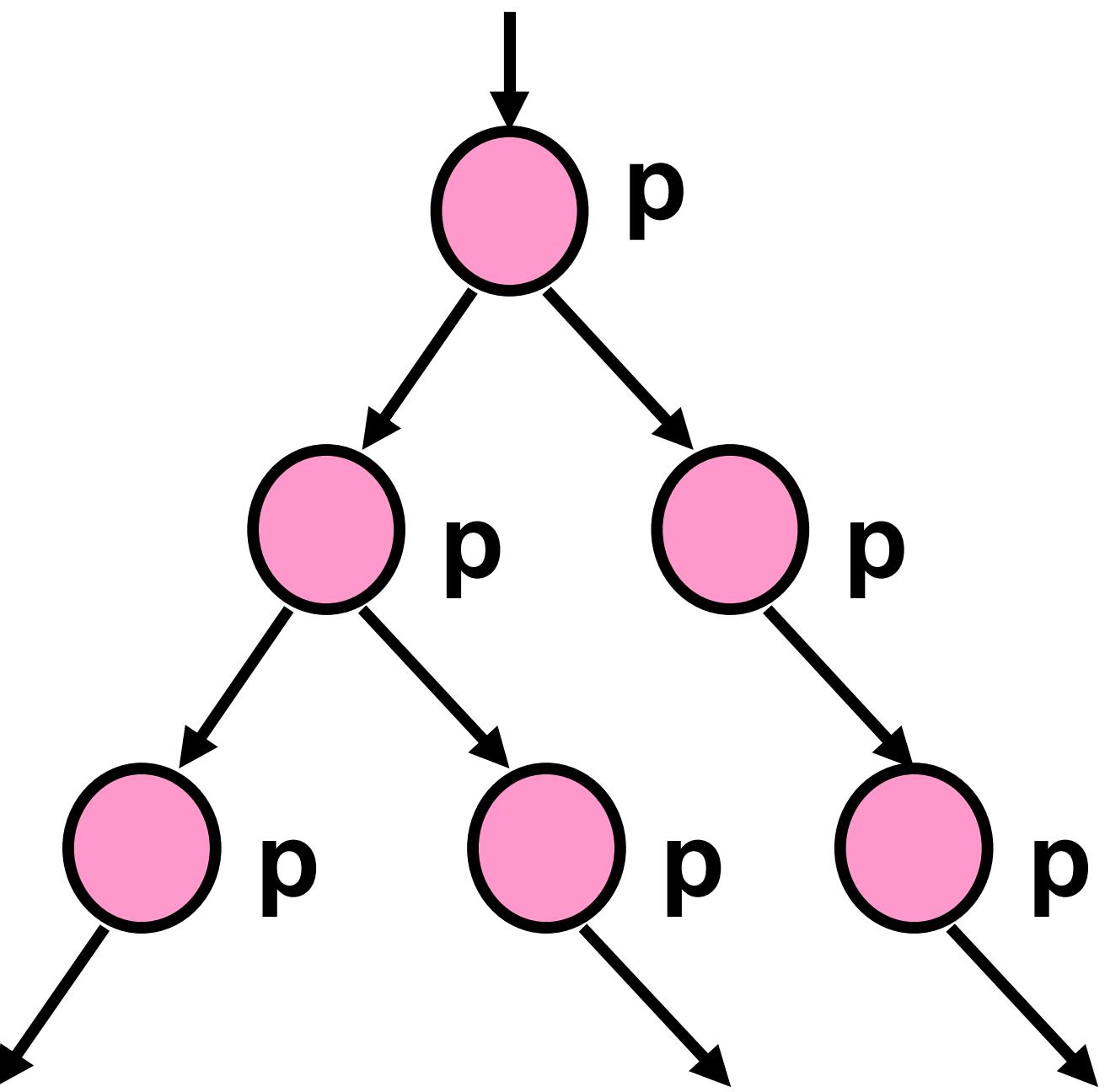
- Reachability $\rightarrow E<> P$
- Reads as “it is possible to reach a state in which P is satisfied



P is true in at least one reachable state

Formulae in TCTL

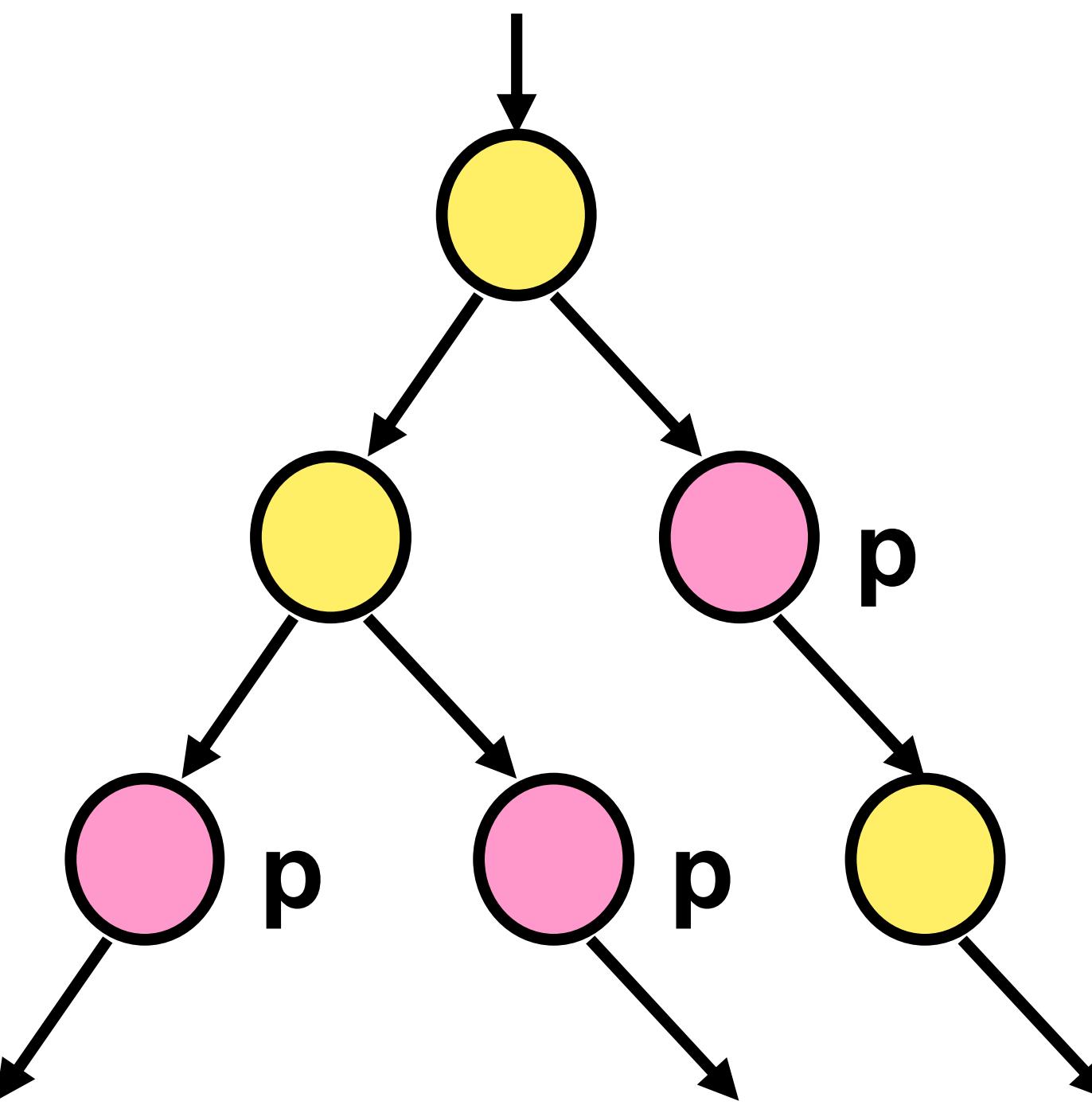
- Invariance: $A[] P$
- “Always P ”



P holds in each possible state

Formulae in TCTL

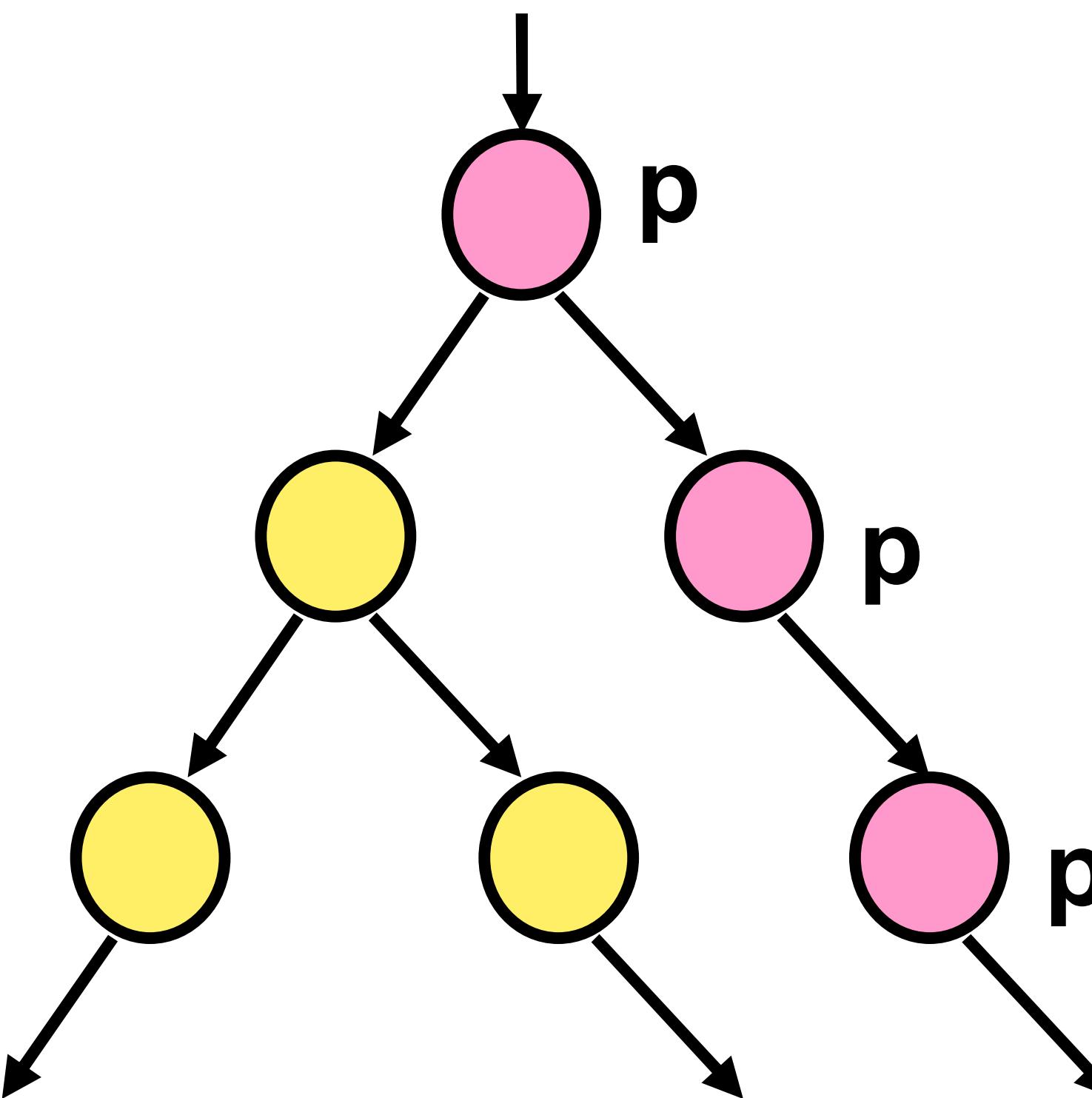
- Inevitable P , $\text{A}\lozenge P$
- P will always eventually become true



P is true in all paths, in some state

Formulae in TCTL

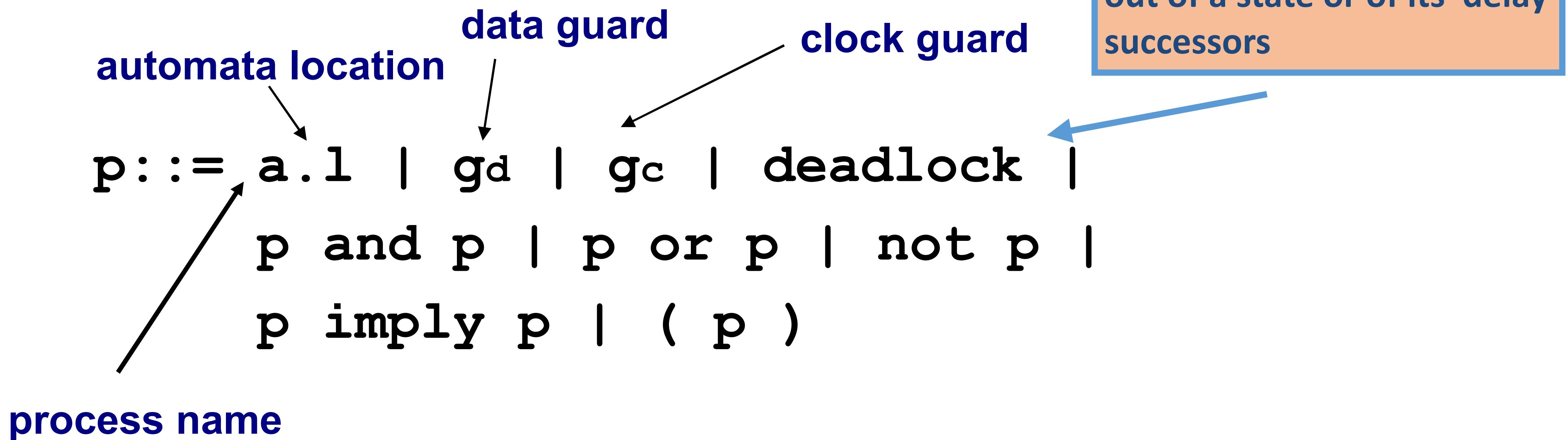
- Potentially Always: $E[] P$.
- There exists a path in which P is true for each transition



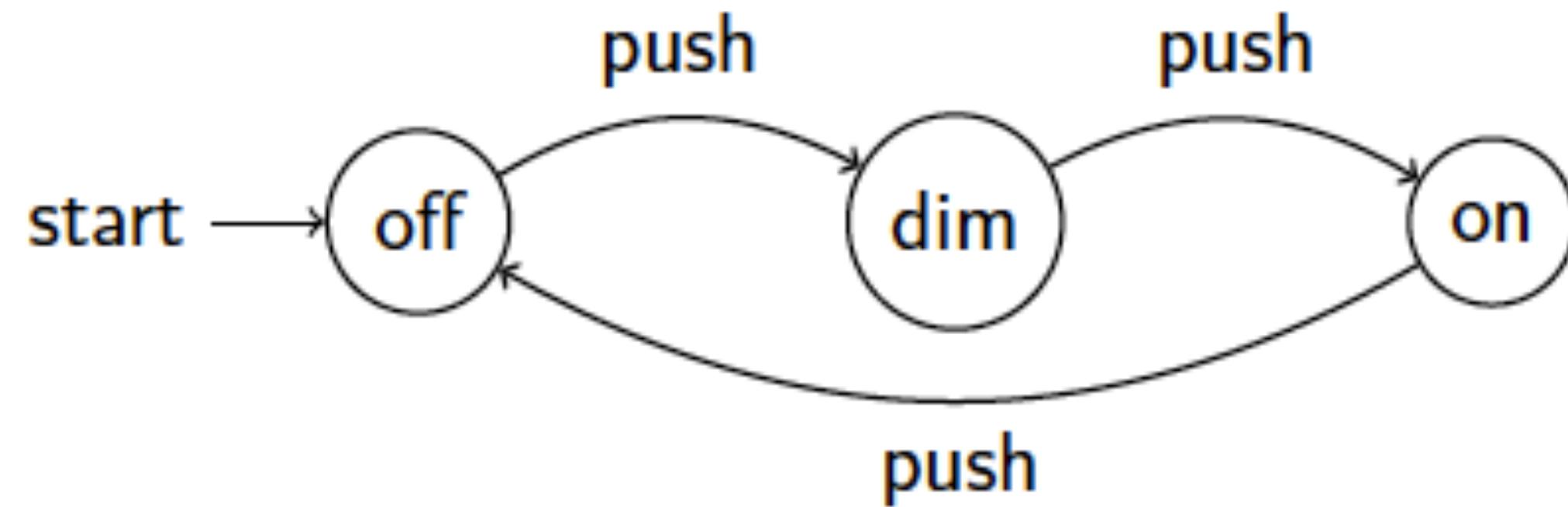
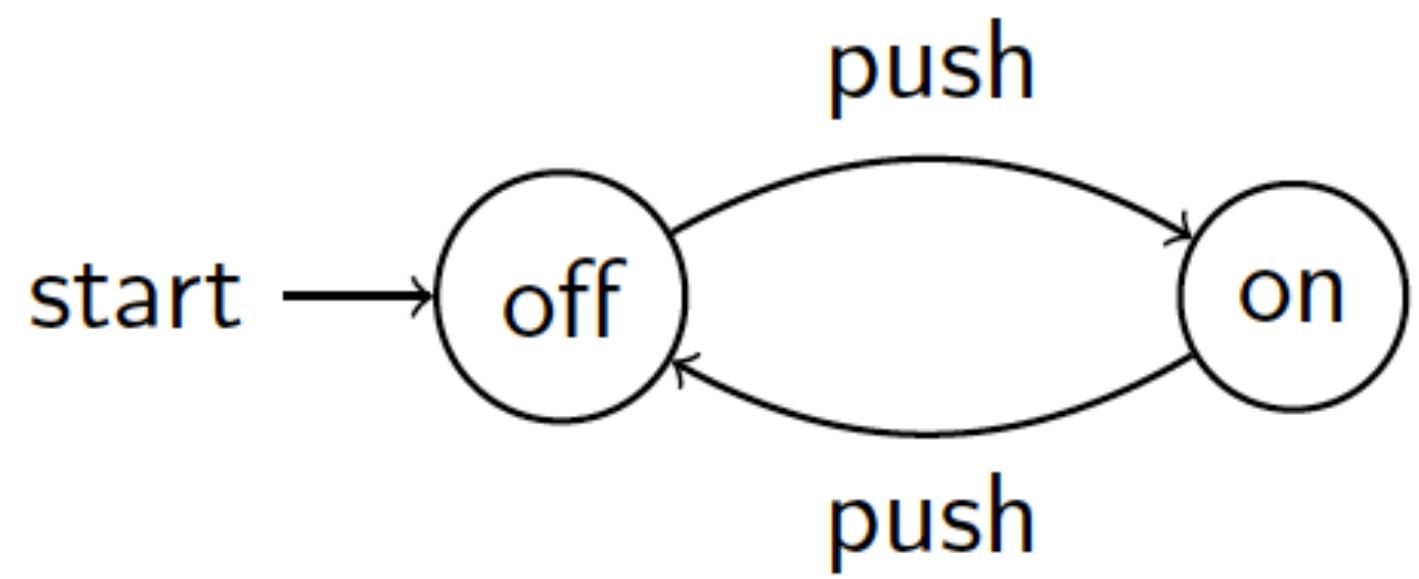
Formulae in TCTL

➤ $A[]p$, $A<>p$, $E<>p$, $E[]p$ - p is a local property

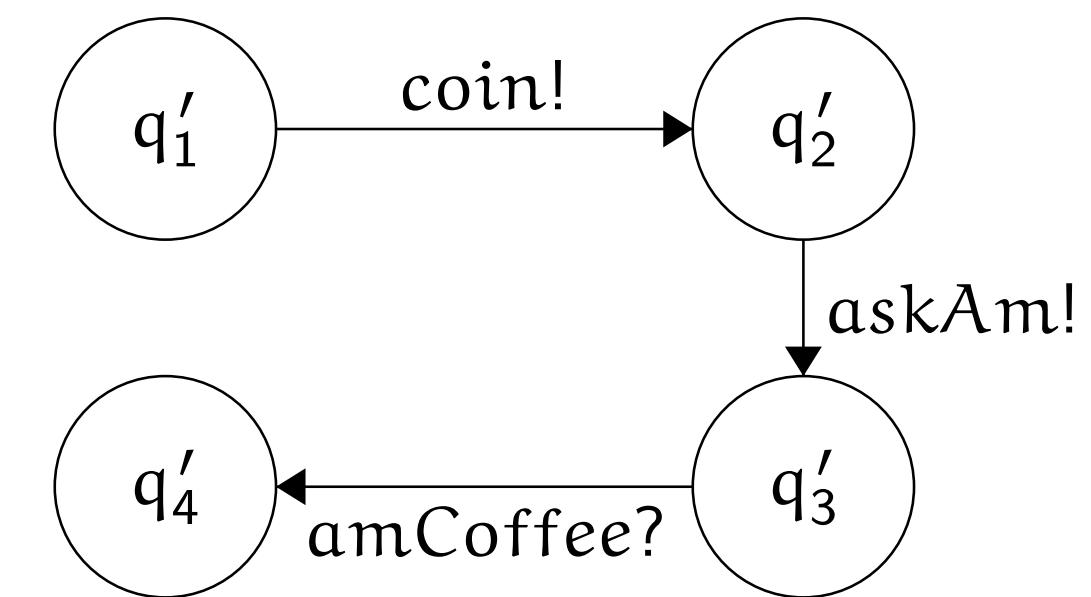
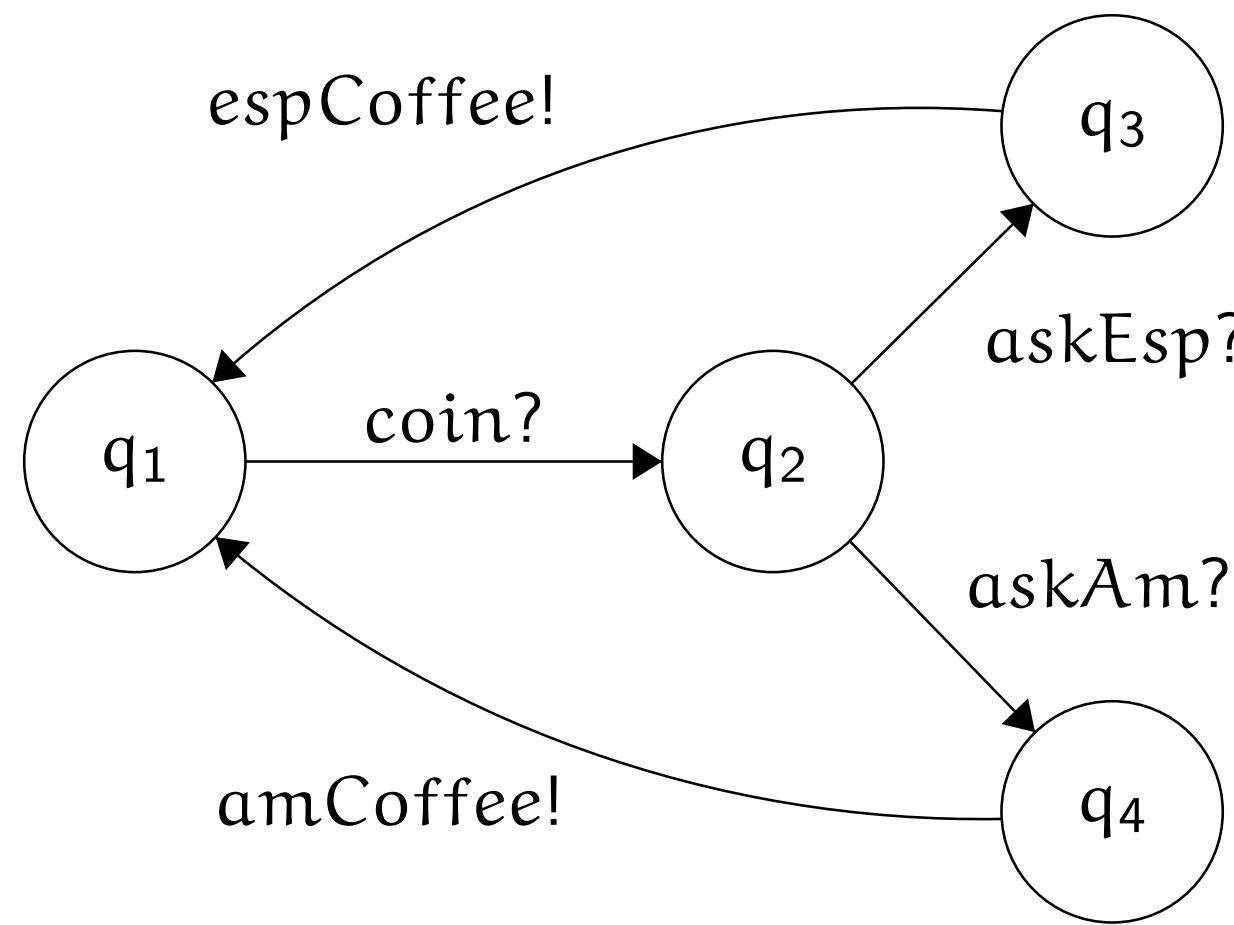
➤ **Syntax:**



Uppaal Demo: A lamp



Second try: the coffee vending machine



Take home message

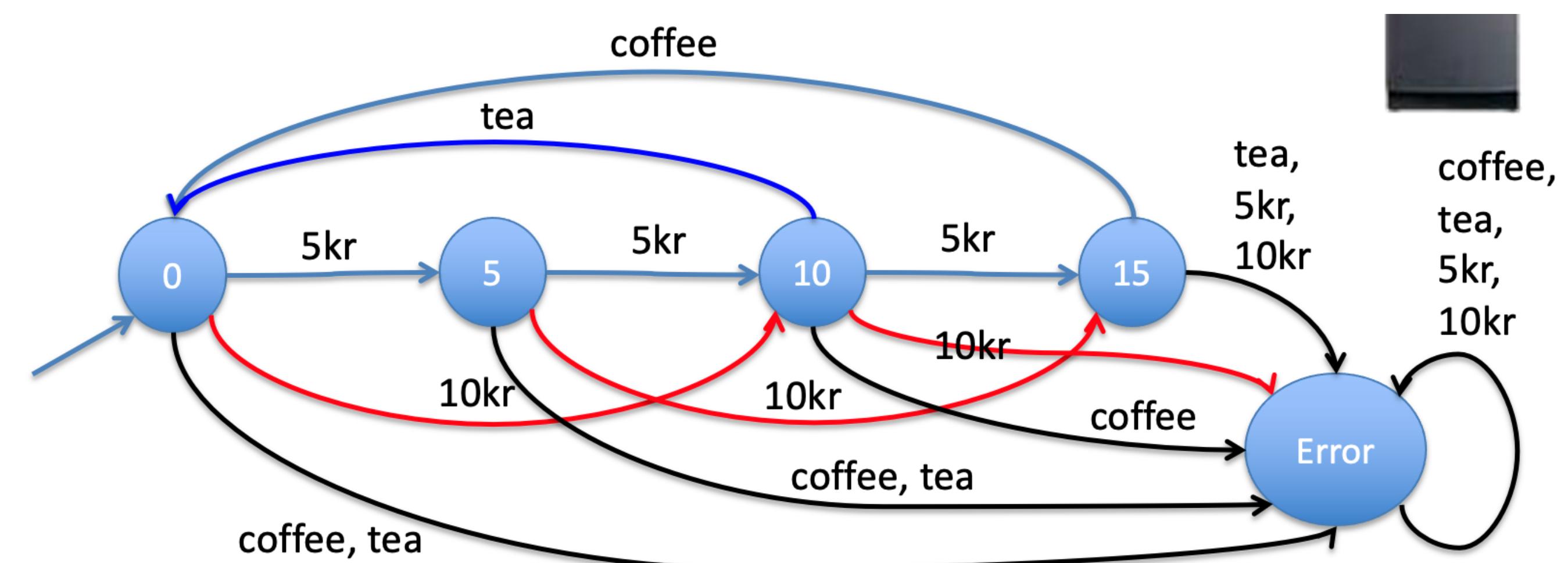
- Real time systems combine action-determinism, message passing, and time operations in to be considered correct
- While the state space becomes indeed humoungous, we can verify the correctness of the system via model checking techniques

Exercise

- Follow the tutorial "A Tutorial on Uppaal"
Gerd Behrmann, Alexandre David, and Kim G. Larsen, and implement the models in sections 4 and 5.

Exercise

- Implement the following state machine in uppaal, together with a corresponding set of users (that like tea or coffee)
- Make sure that the vending machine returns money after a timeout of 30 seconds
- Verify that for a multiple user can eventually be served by the vending machine (fairness)



Exercise

- Consider the interaction model you handed in
 - Discuss with your group the type of timing constraints you may have in your model
 - And if you can, implement the model in Uppaal
 - Discuss with your group the type of model-checking properties you may be interested
 - And If you manage to implement the models, try to verify the new properties

Peer review

- Carried out individually
- Step 1: each student submits a copy of the group project report and of the models they already submitted in the Assignment section.
- Submission through the Peer-review part 1 module in the Peer-review section
- All students working in the same group will submit the same files
- Make sure that the report and models are anonymized
- Deadline: March 14 at 11:59 (A.M.)

Peer review

- Step 2: each student reviews 1 report of another group.
- Review through the Peer-review part 1 module in the Peer-review section
- All students working in the same group will get different reports
- Assessment is guided: a rubric of several criteria is defined
- You need to assign a score for each criterion
- You need to motivate your score, pointing out where issues are present in the models
- Feedback is anonymous to students (but not to teachers)
- Deadline: March 19 at 23:59.
- If you get more than 1 report to review, please contact the teachers.

Declarative Process Modelling

Hugo A. López

hulo@dtu.dk

DTU Compute

Department of Applied Mathematics and Computer Science

$$f(x+\Delta x) = \sum_{i=0}^{\infty} \frac{(\Delta x)^i}{i!} f^{(i)}(x)$$
$$\int_a^b \Theta + \Omega \int_0^{\delta} e^{i\pi} =$$
$$\sqrt{17} \sum_{!} \Sigma \gg ,$$
$$\epsilon \in \{2.7182818284\}$$
$$\infty \rightarrow x^2$$

Agenda

DTU Compute

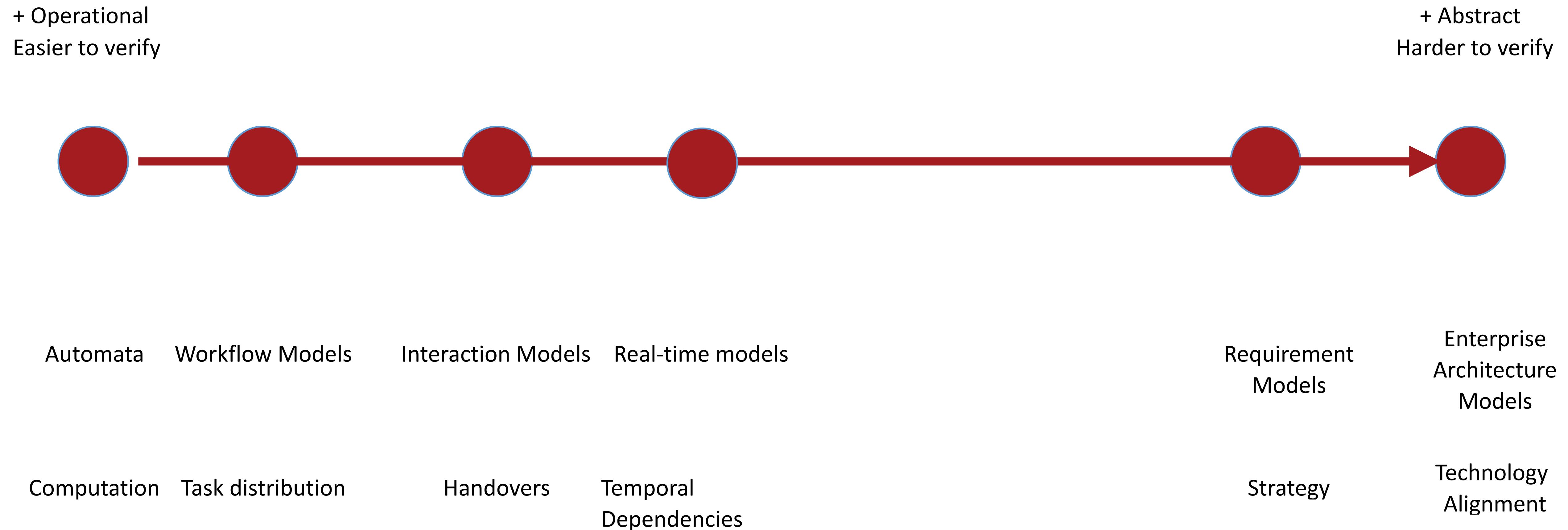
Department of Applied Mathematics and Computer Science

Hugo A. López

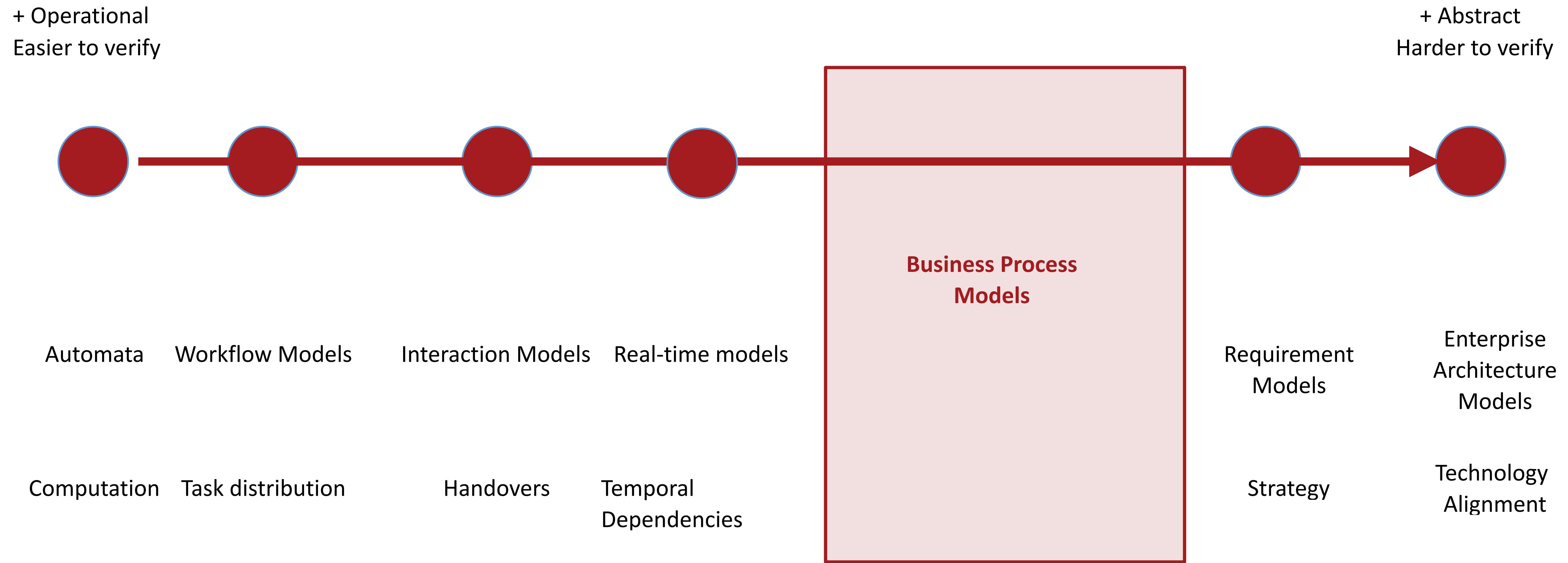


- Flexibility in Process Models
- Declarative Process Modelling
- Introduction to DCR graphs
- Introduction to part 2 of the assignment
- Exercises

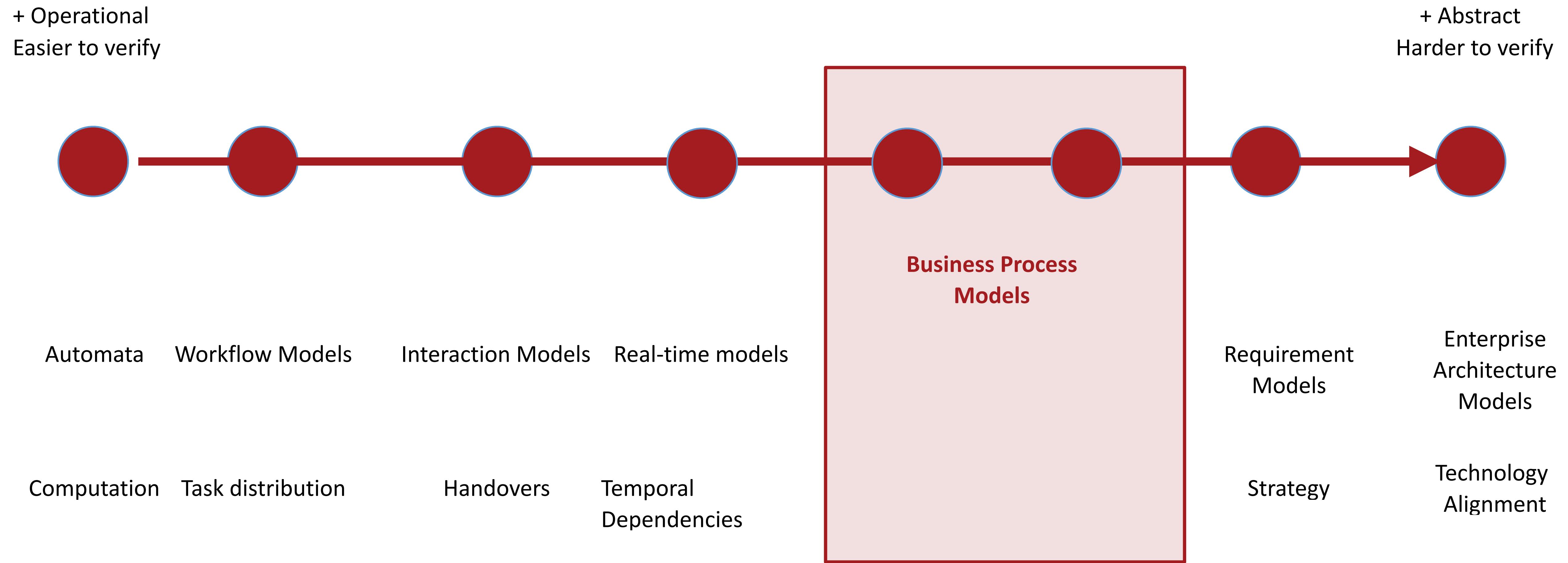
Our modelling dimensions



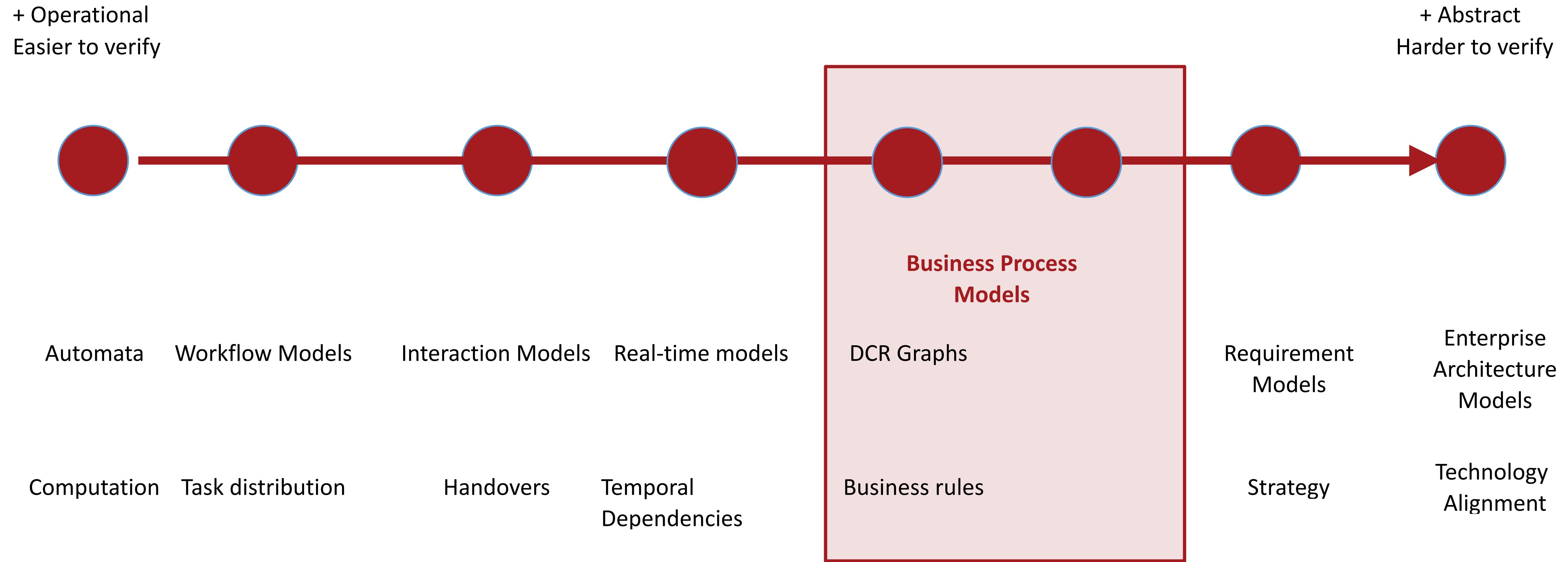
Our modelling dimensions



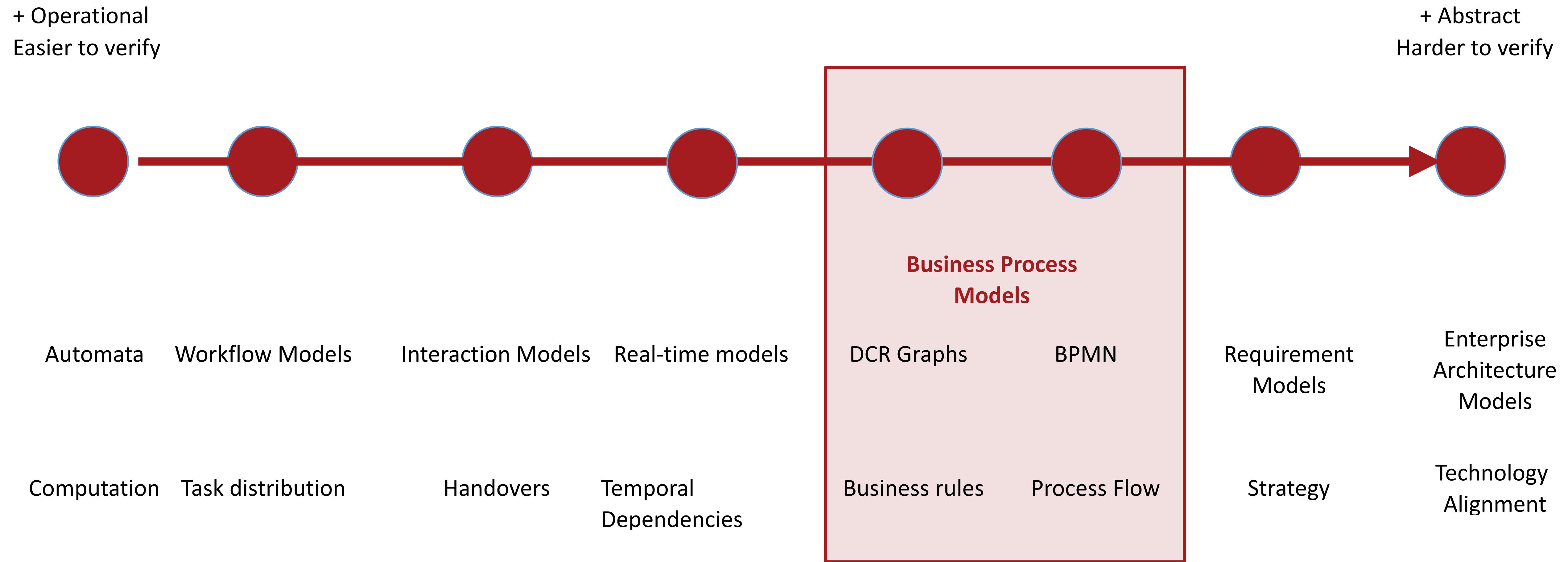
Our modelling dimensions



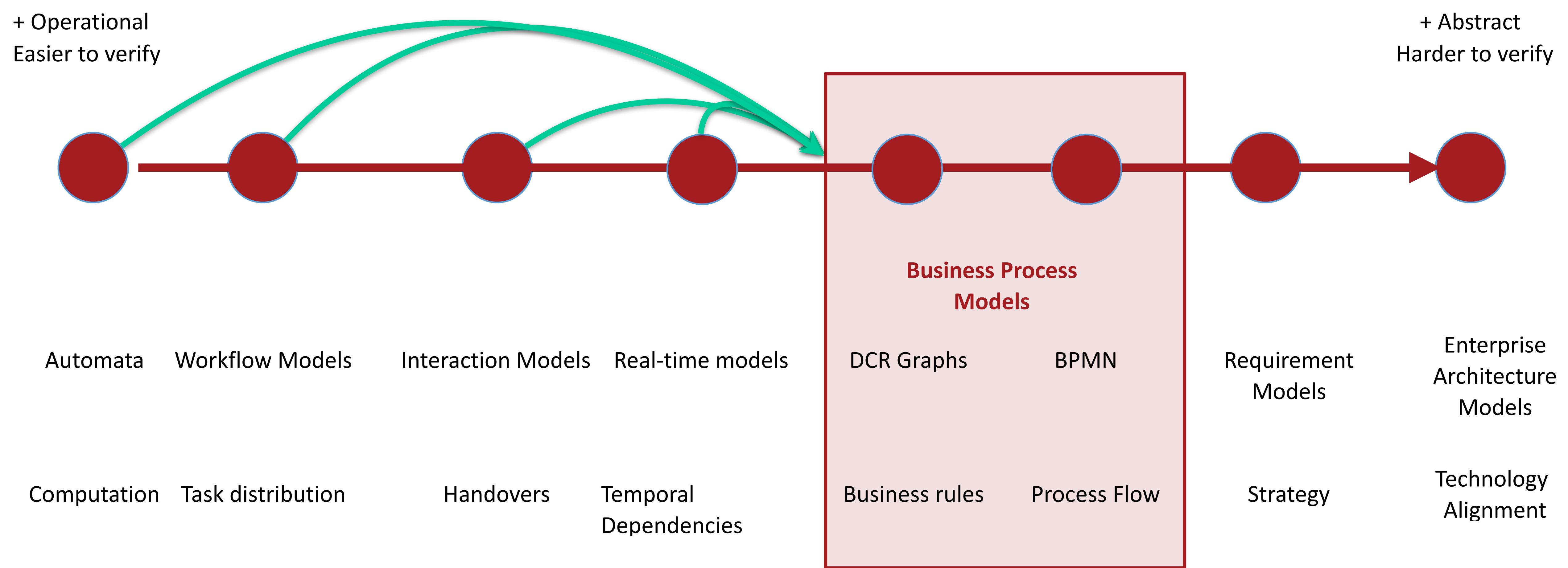
Our modelling dimensions



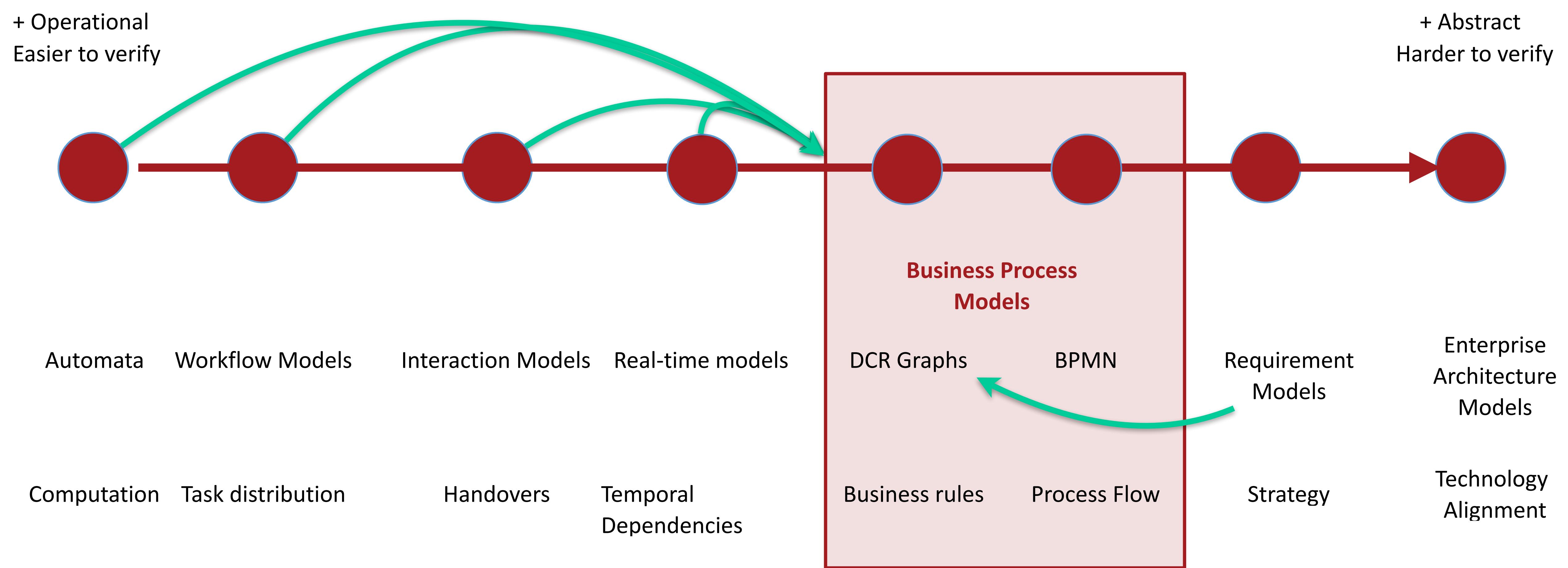
Our modelling dimensions



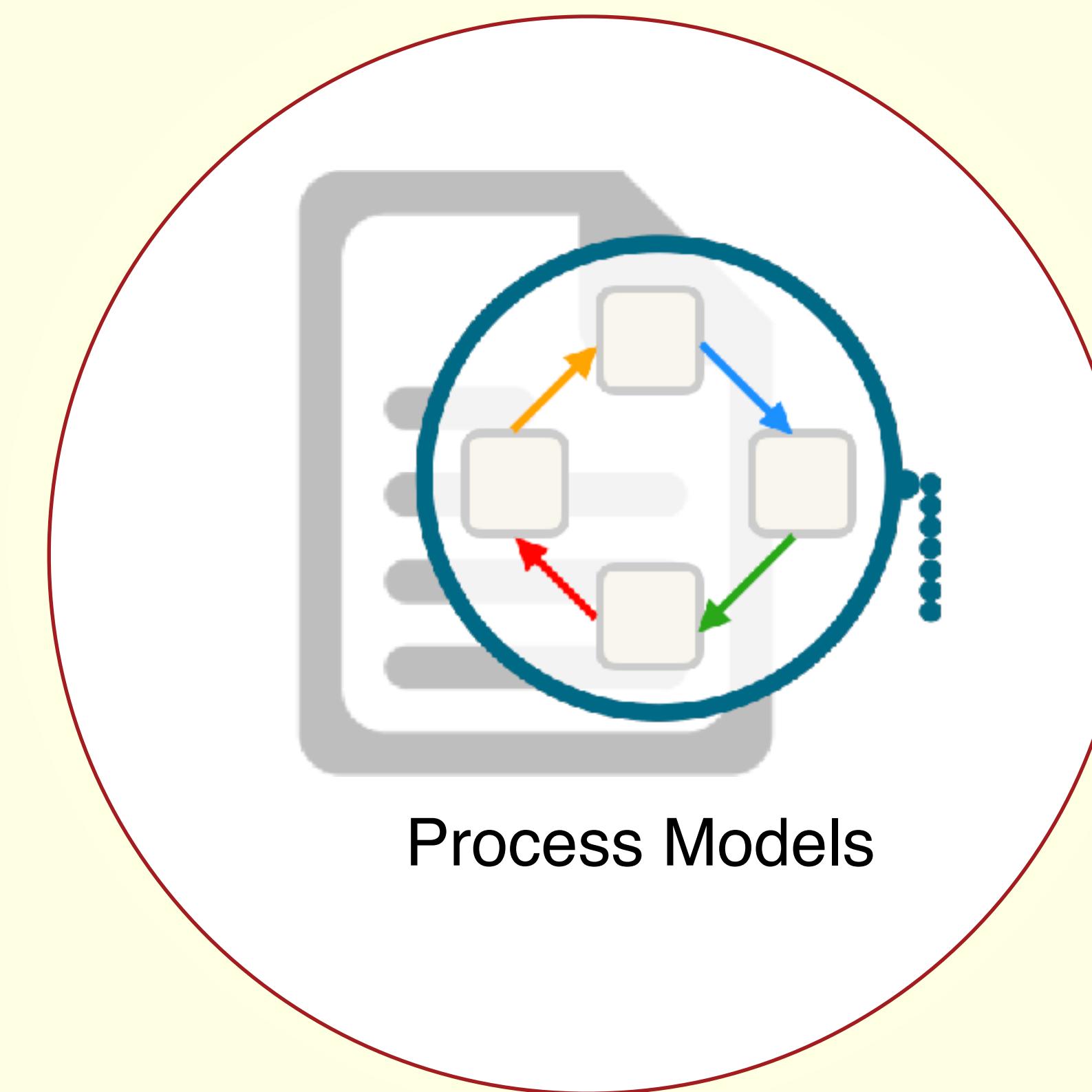
Our modelling dimensions



Our modelling dimensions

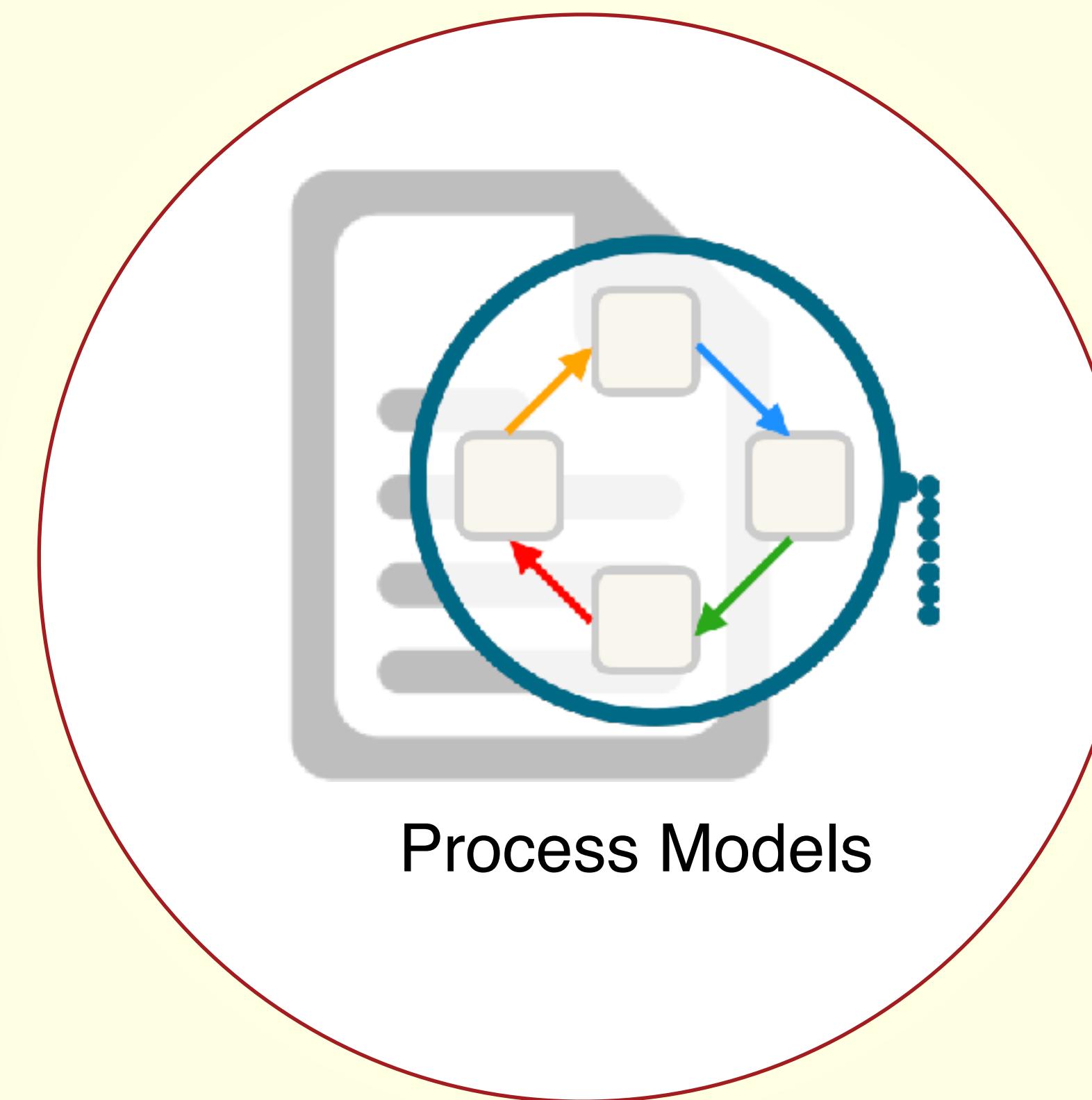


Process Models

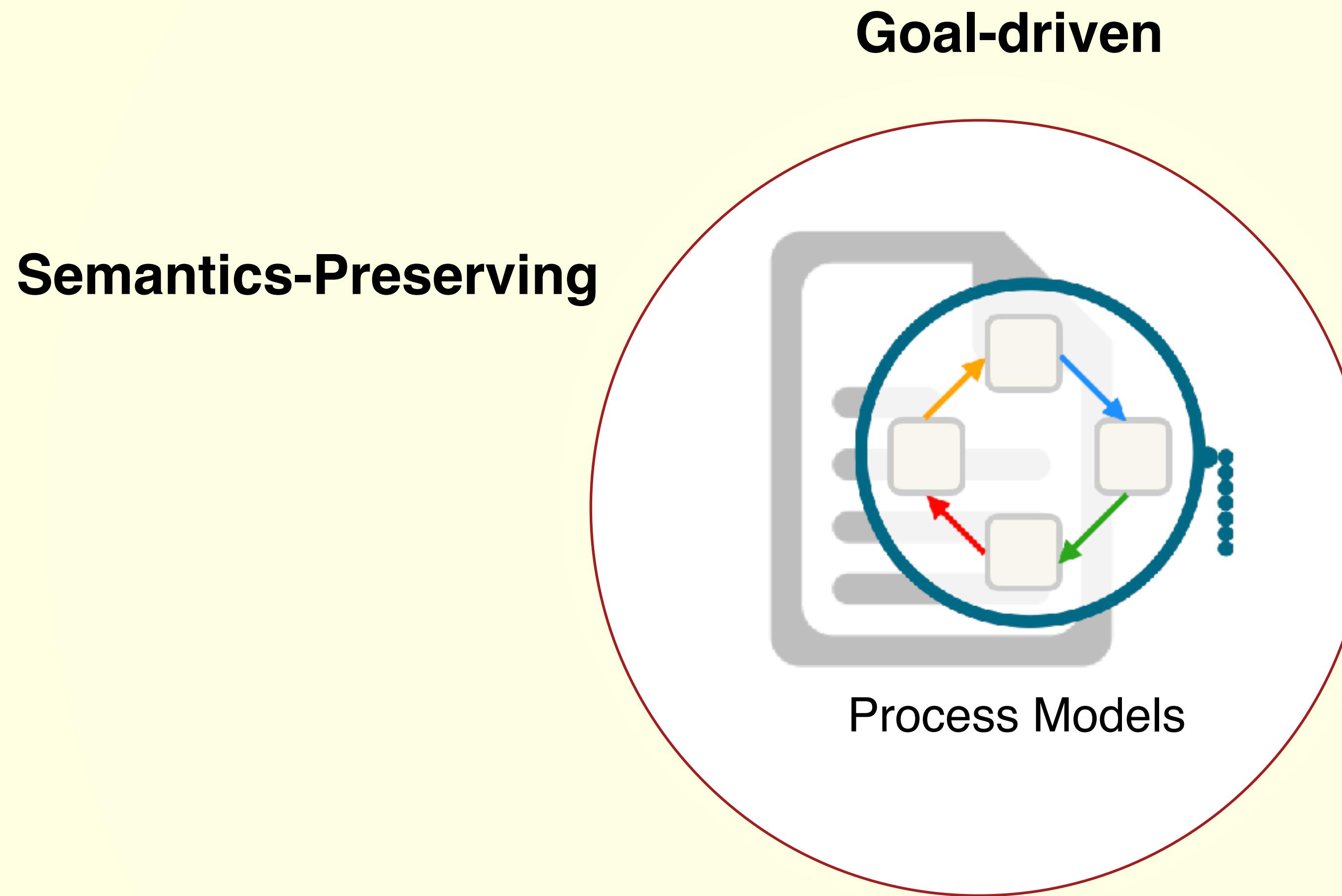


Process Models

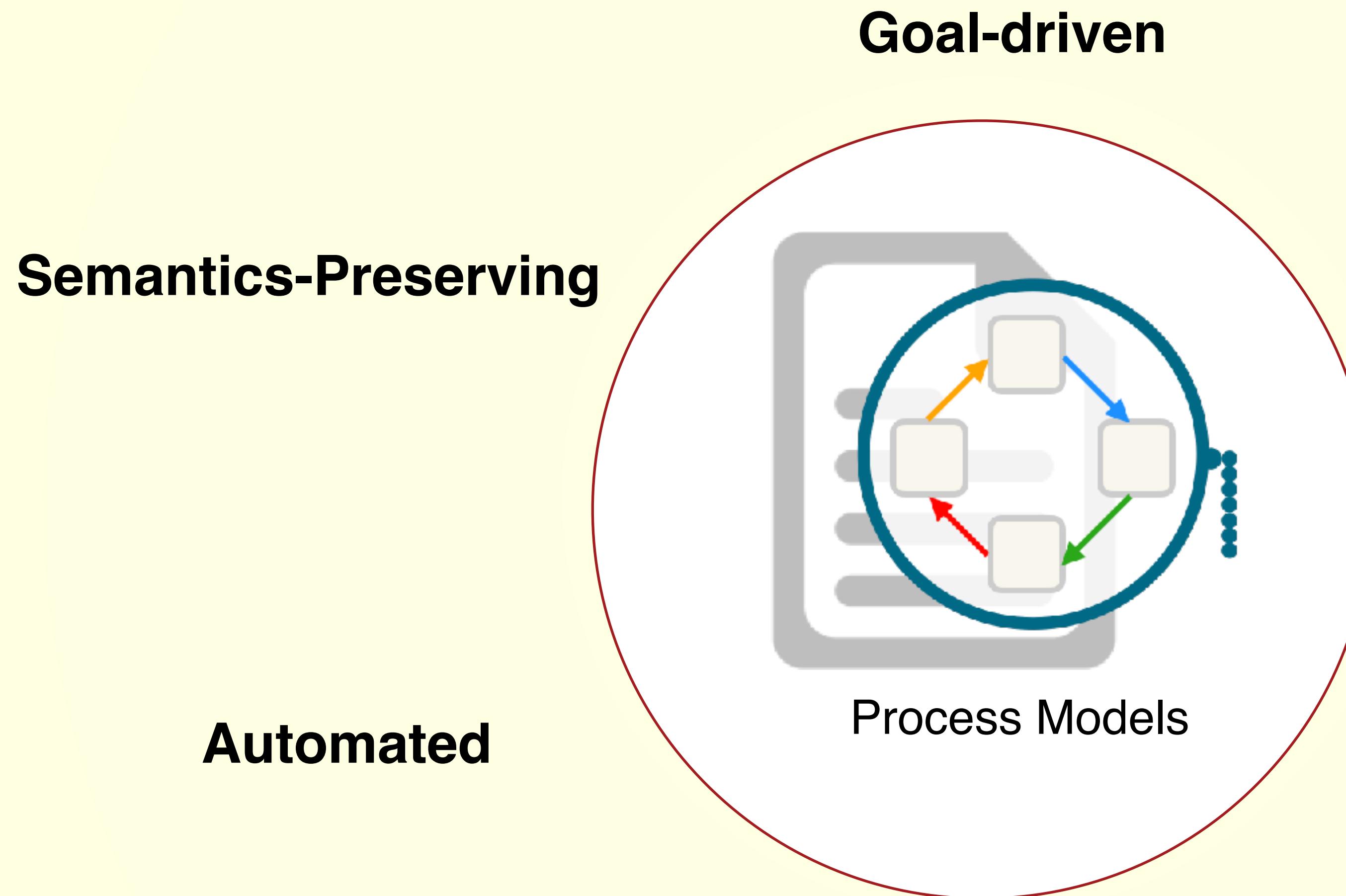
Goal-driven



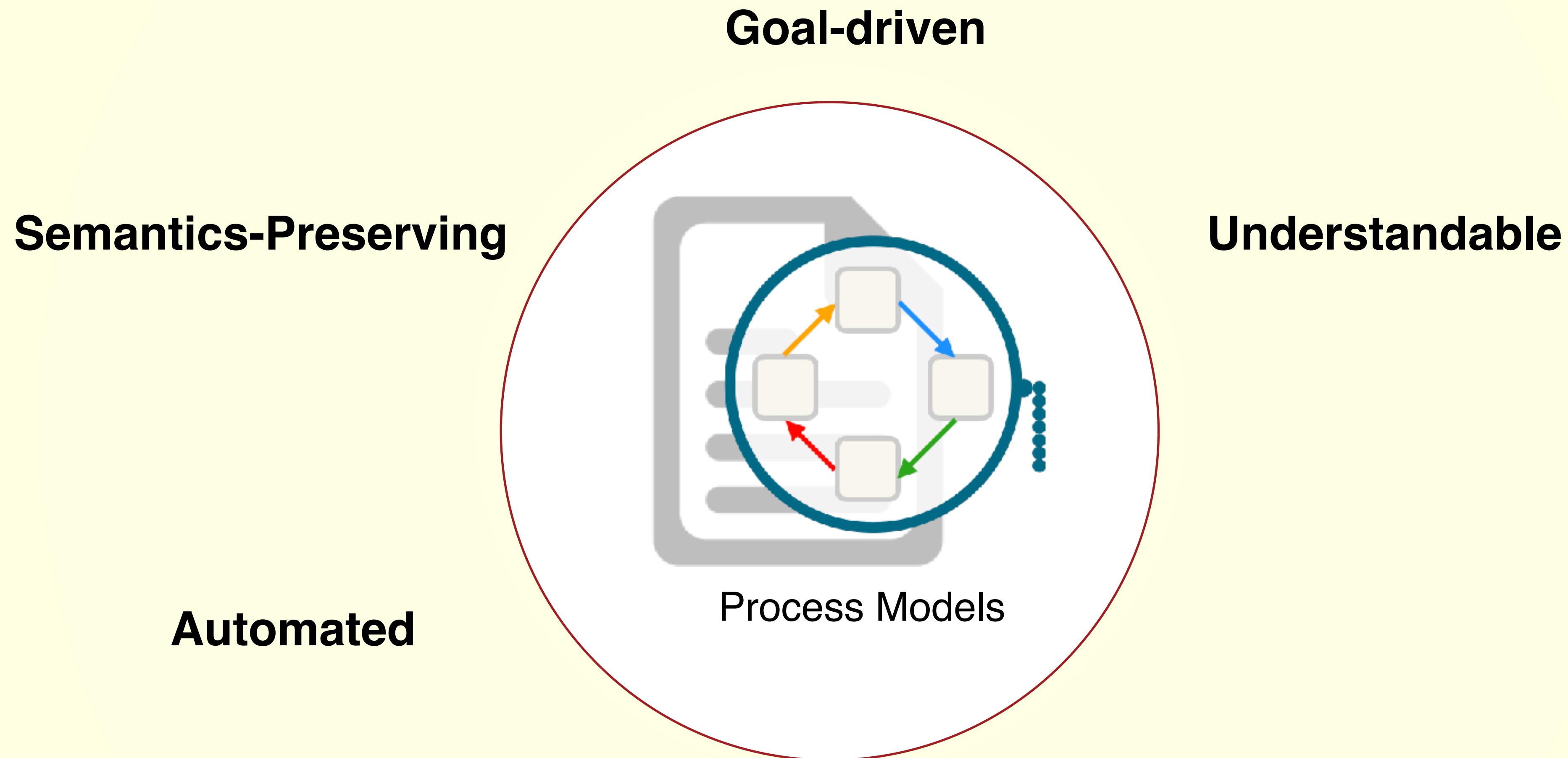
Process Models



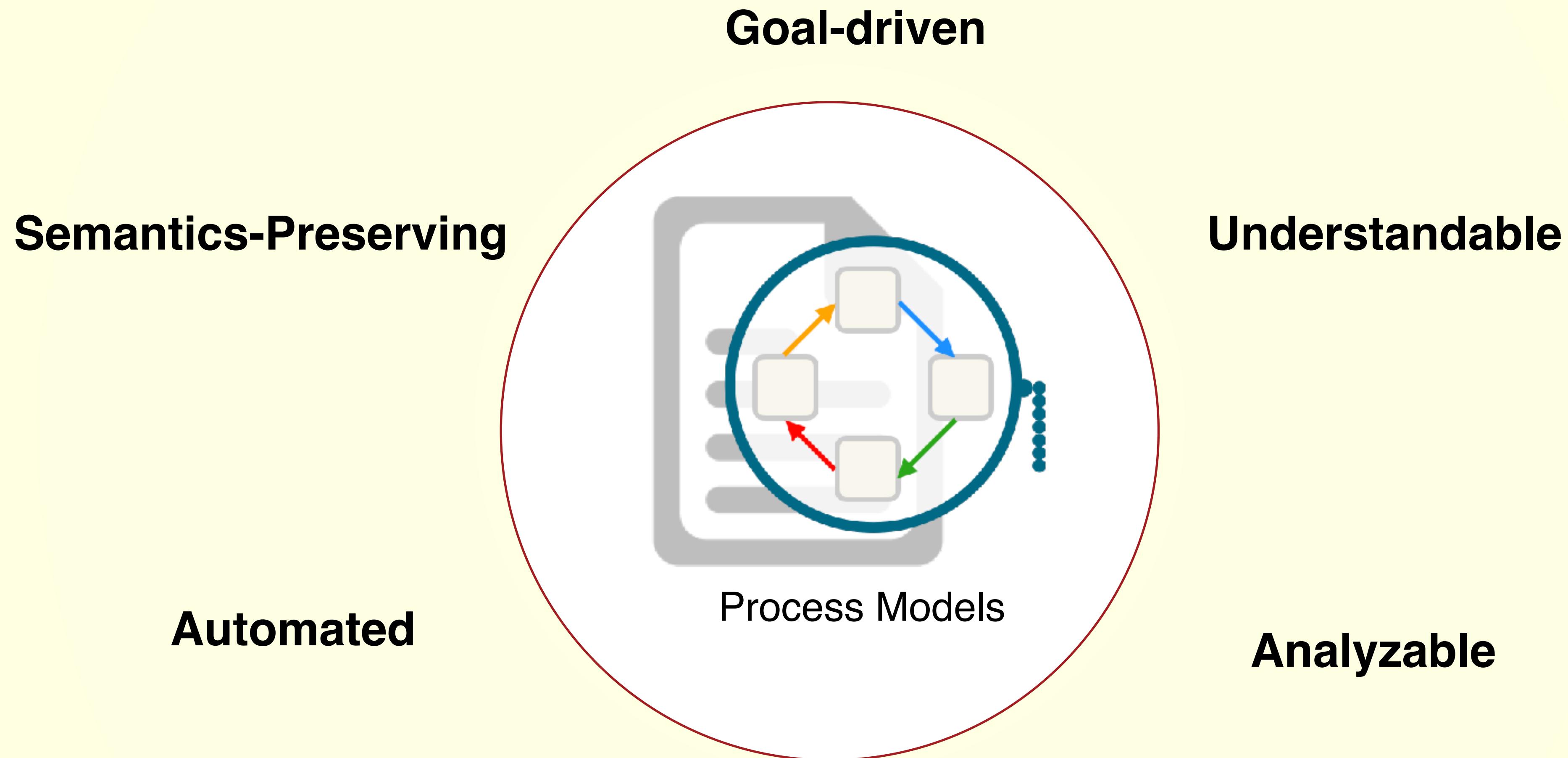
Process Models



Process Models



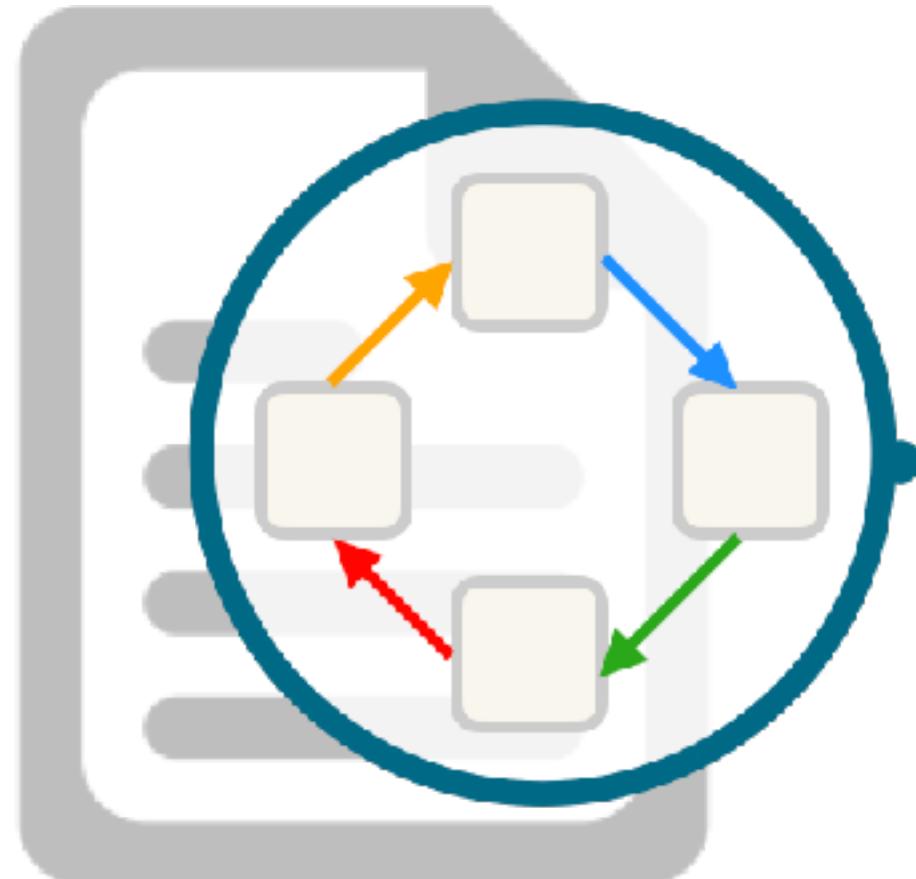
Process Models



Process Models

Semantics-Preserving

Goal-driven



Process Models

Automated

Understandable

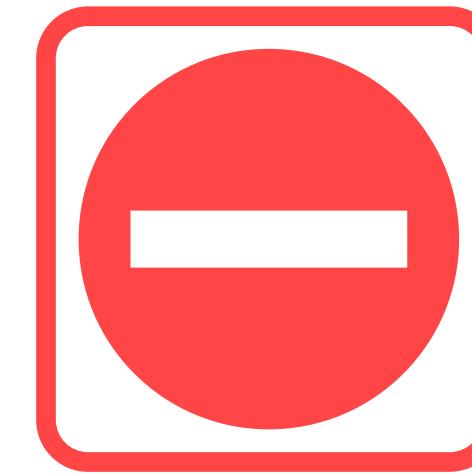
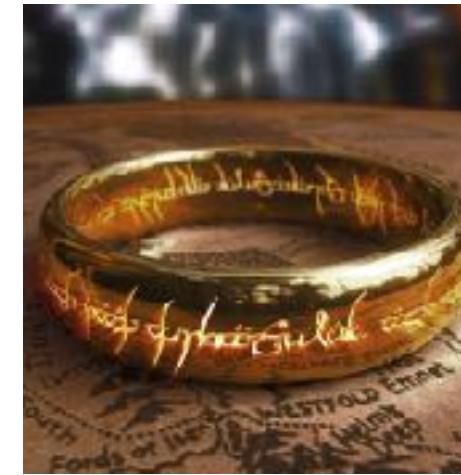


Analyzable

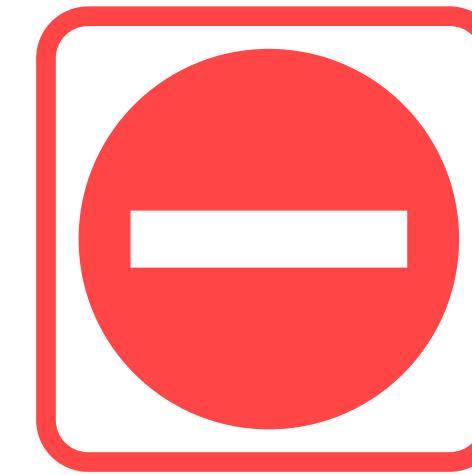
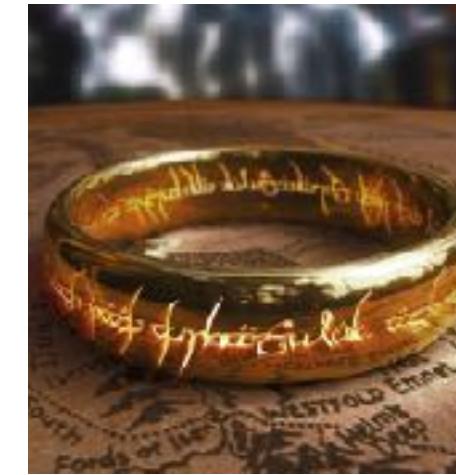


What is your process about?

Not all business processes are the same!

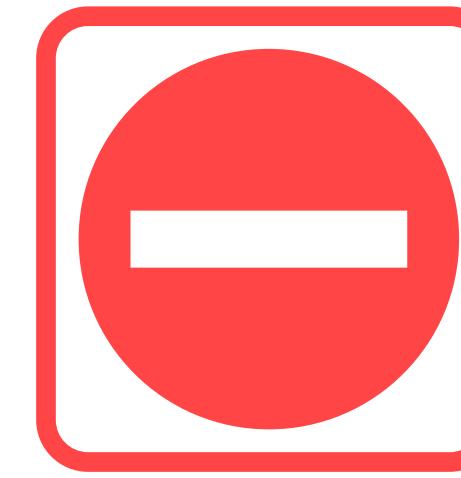


Not all business processes are the same!



- Even within the same organization, business processes can be very different in terms of their essential properties.
 - *Ex. Manufacturing: Automated production vs. Customer Service*

Not all business processes are the same!



- Even within the same organization, business processes can be very different in terms of their essential properties.
 - *Ex. Manufacturing: Automated production vs. Customer Service*
 - Today we will explore three dimensions:
 - Complexity
 - Predictability
 - Repetitiveness

Complexity

Complexity

- What is the degree of difficulty in collaboration, decision making and collaboration?
- **Low Complexity:** Exchanging a personal email message, handling a travel request.
- **High Complexity:** Payment of unemployment benefits, medical treatments.

Predictability

- How easy is to predict how the process will execute, and the possible outcomes:
 - **Low Predictability:** Exchanging personal email messages. Payment of unemployment benefits.
 - **High Predictability:** Handling travel requests, manufacturing product lines.

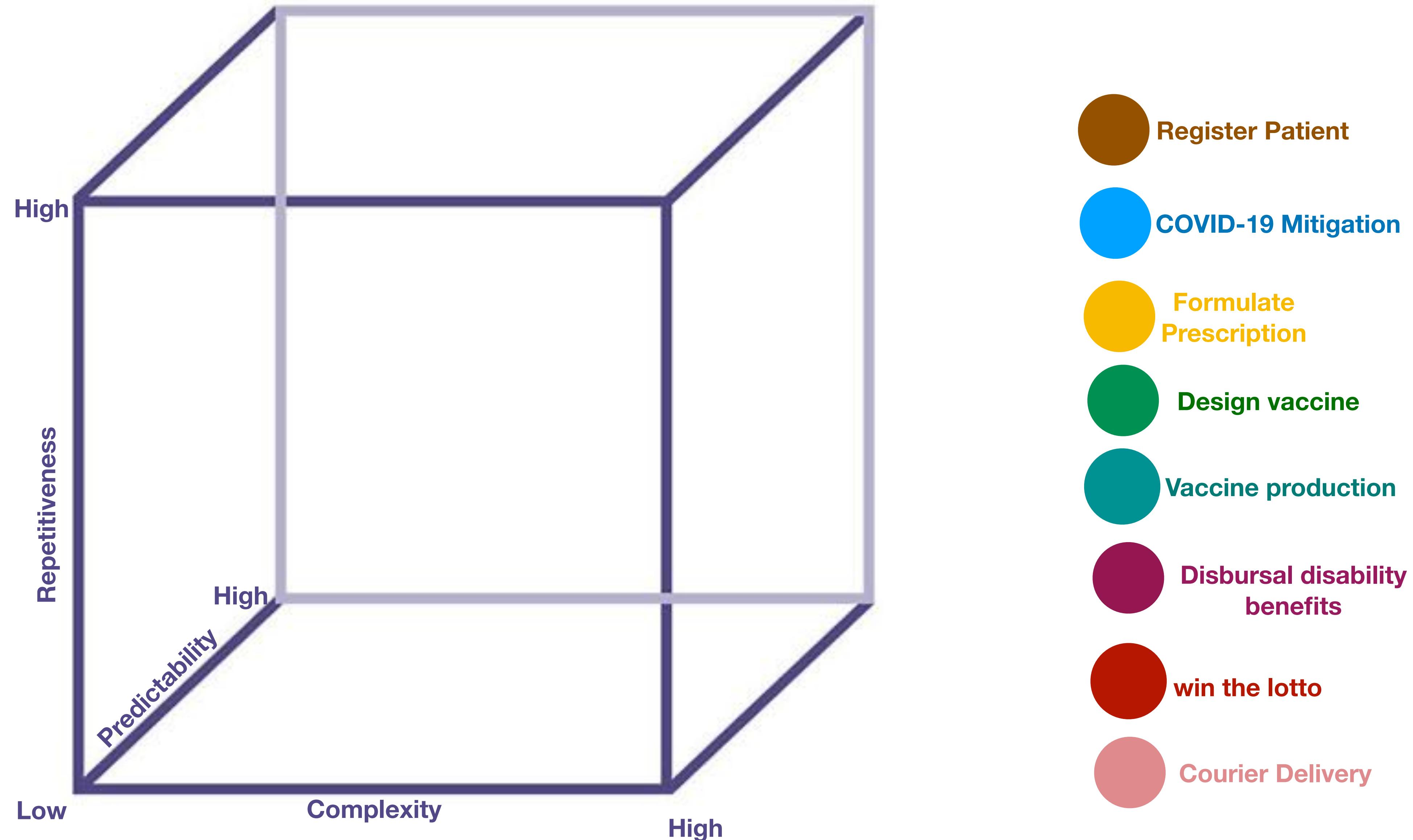
Repetitiveness

- How often do you run this process?
Once per minute, twice per month, once per year...
- **Low Frequency:** Disaster Mitigation.
- **High Frequency:** Manufacturing Product Lines

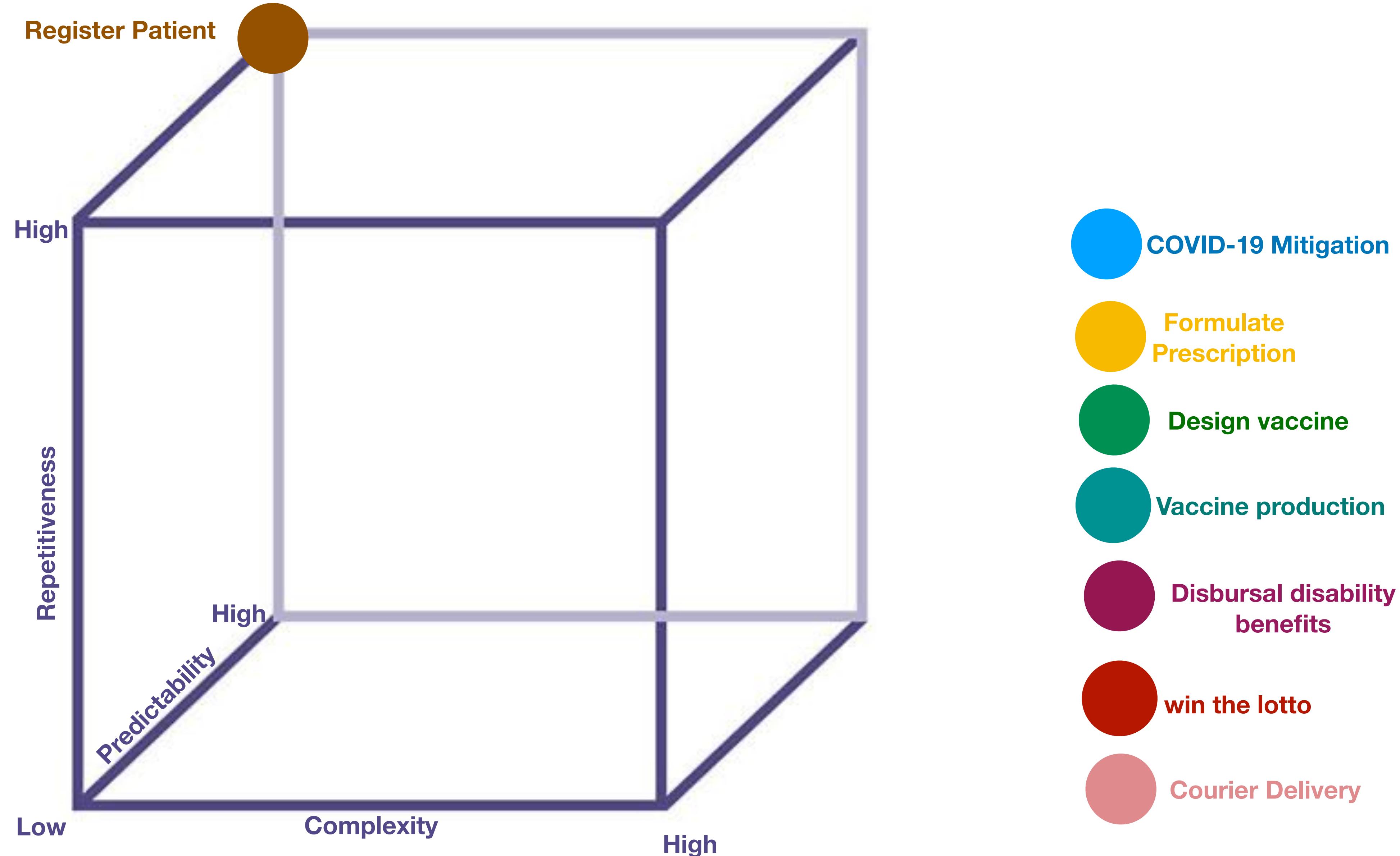
Exercise (5 min):

1. Classify the following processes according to complexity, predictability and repetitiveness:
 1. Register a new patient
 2. Pandemic (e.g. COVID-19) mitigation plan
 3. Formulate a drug prescription
 4. Design vaccine
 5. Vaccine production
 6. Disbursal of handicap benefits
 7. Win the lottery
 8. Courier delivery (aka Wolt)

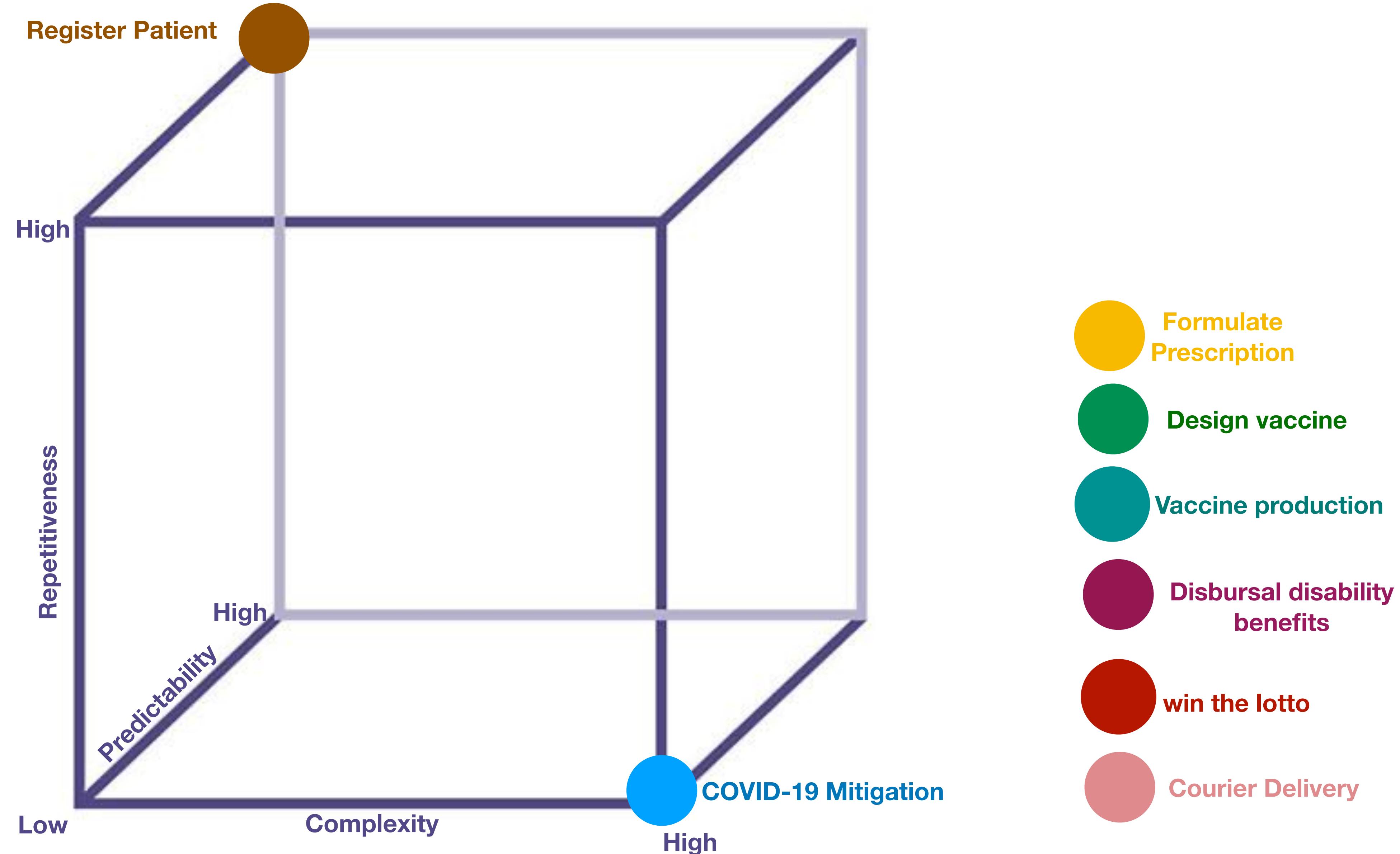
BP Dimensions



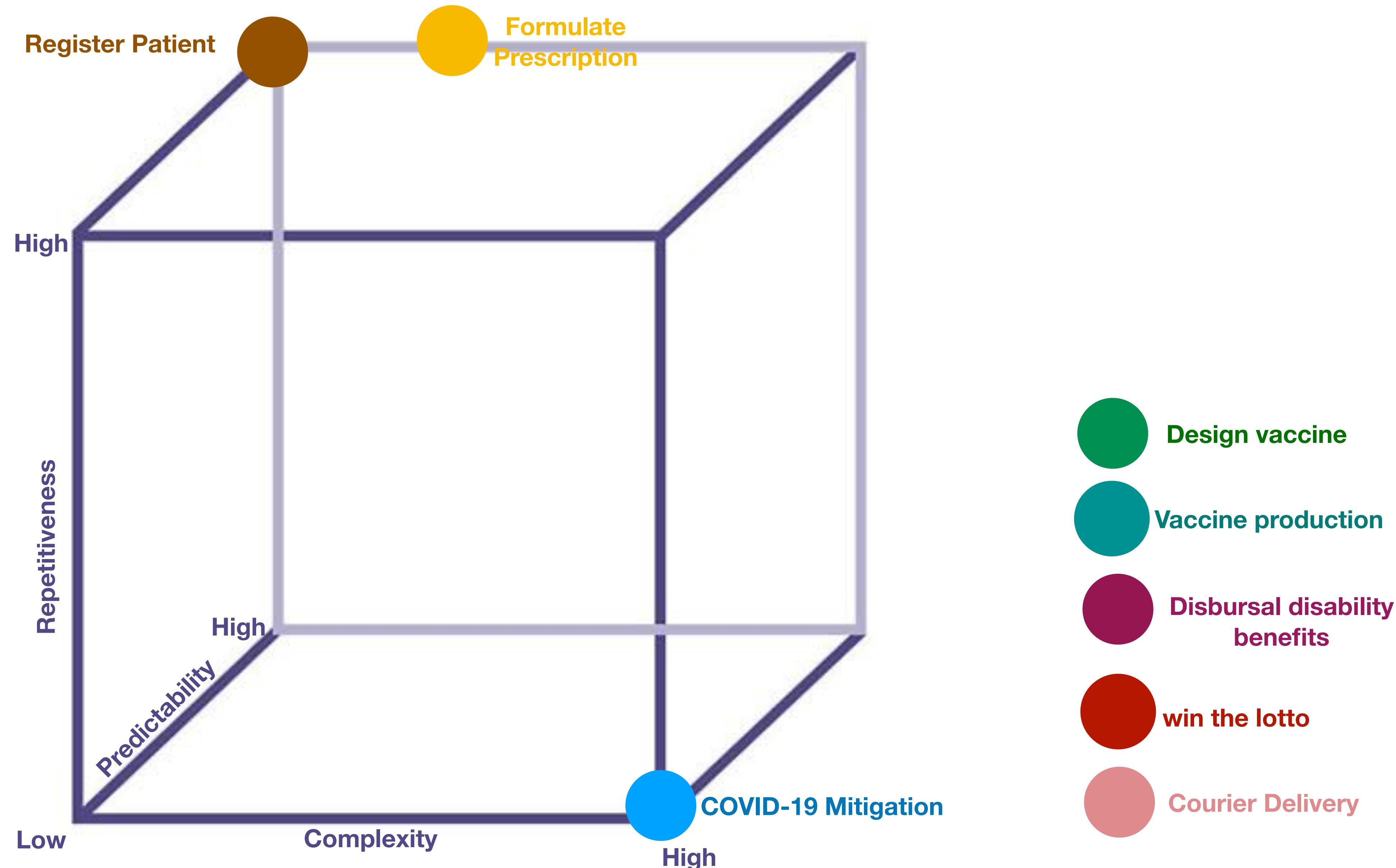
BP Dimensions



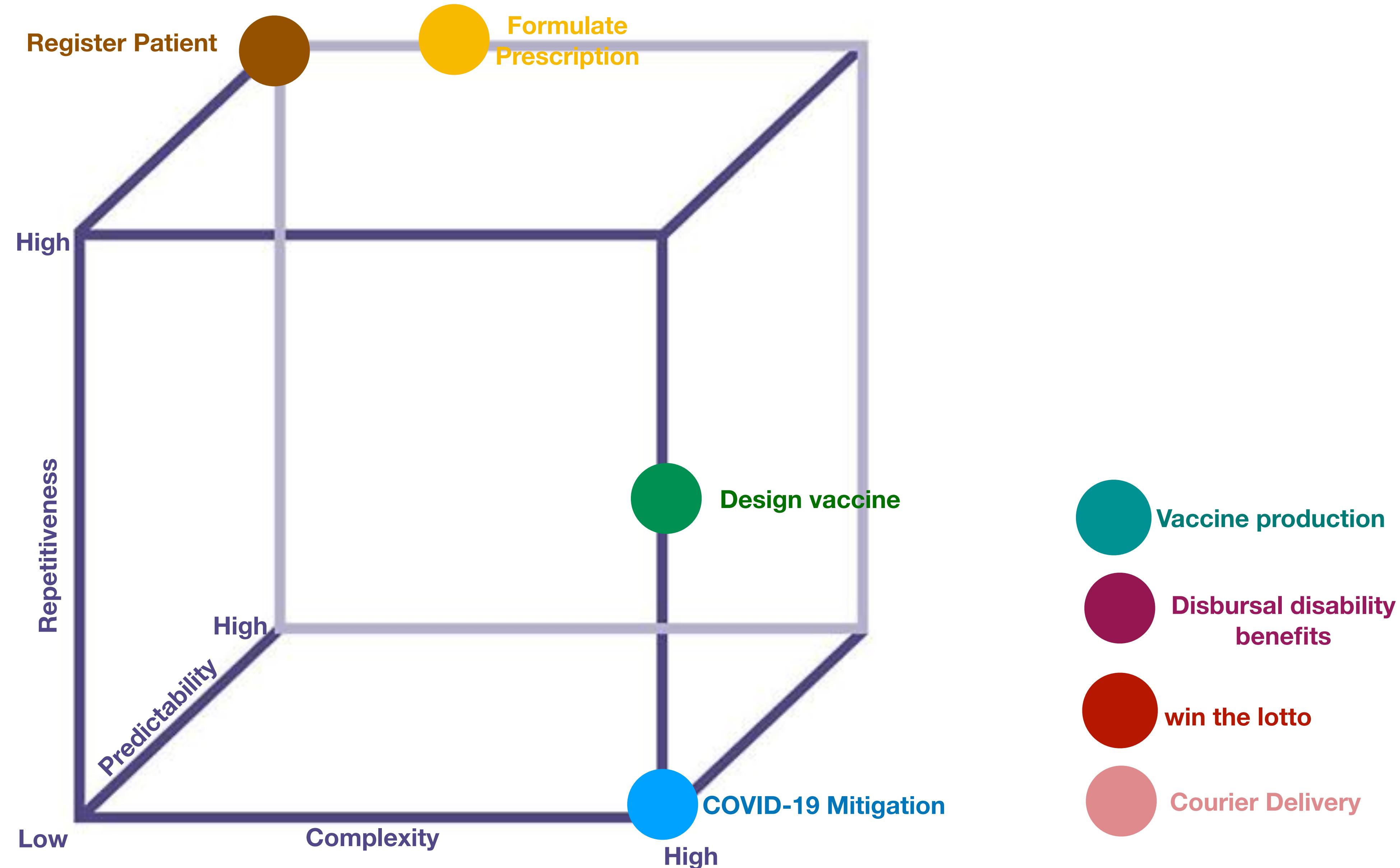
BP Dimensions



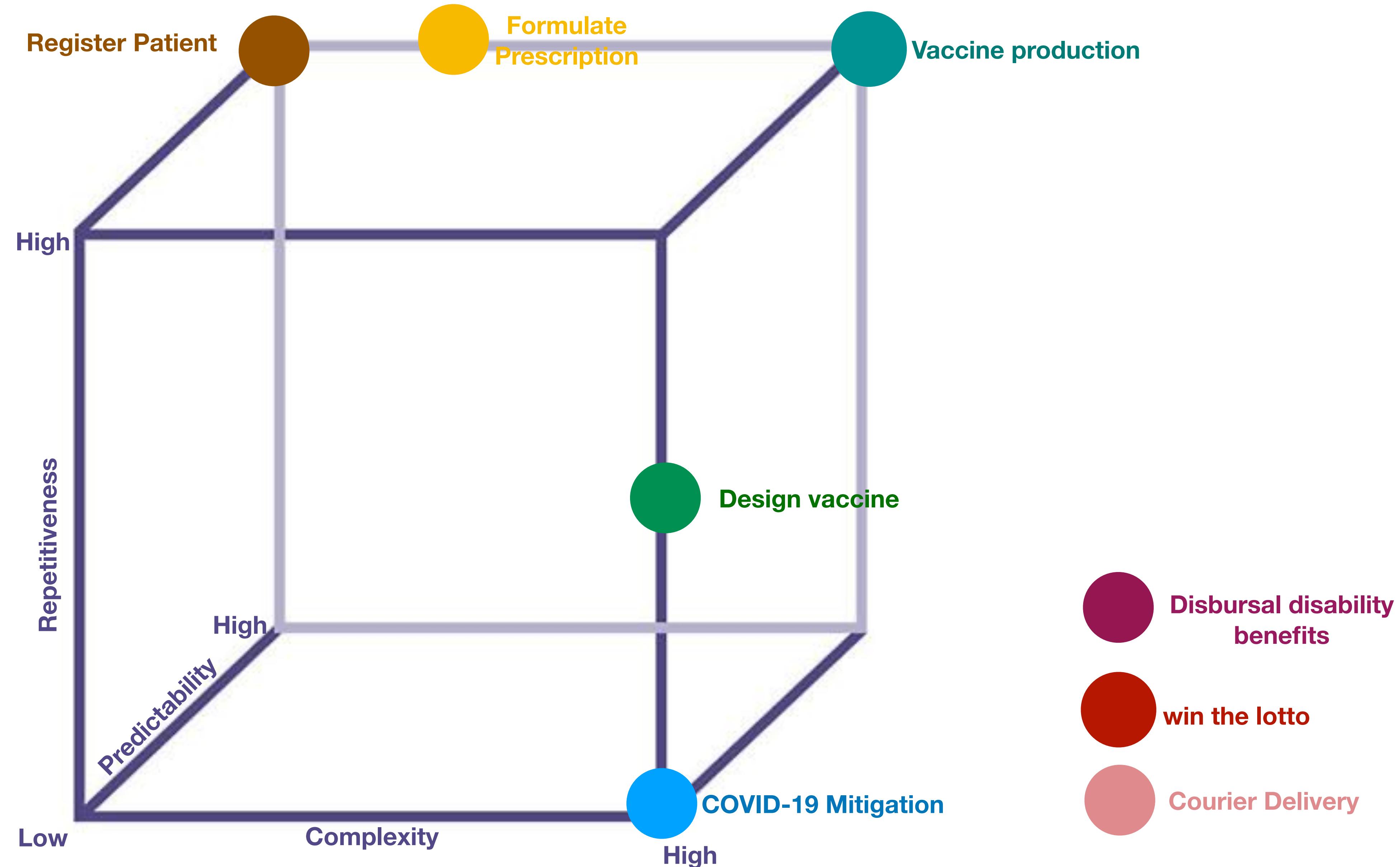
BP Dimensions



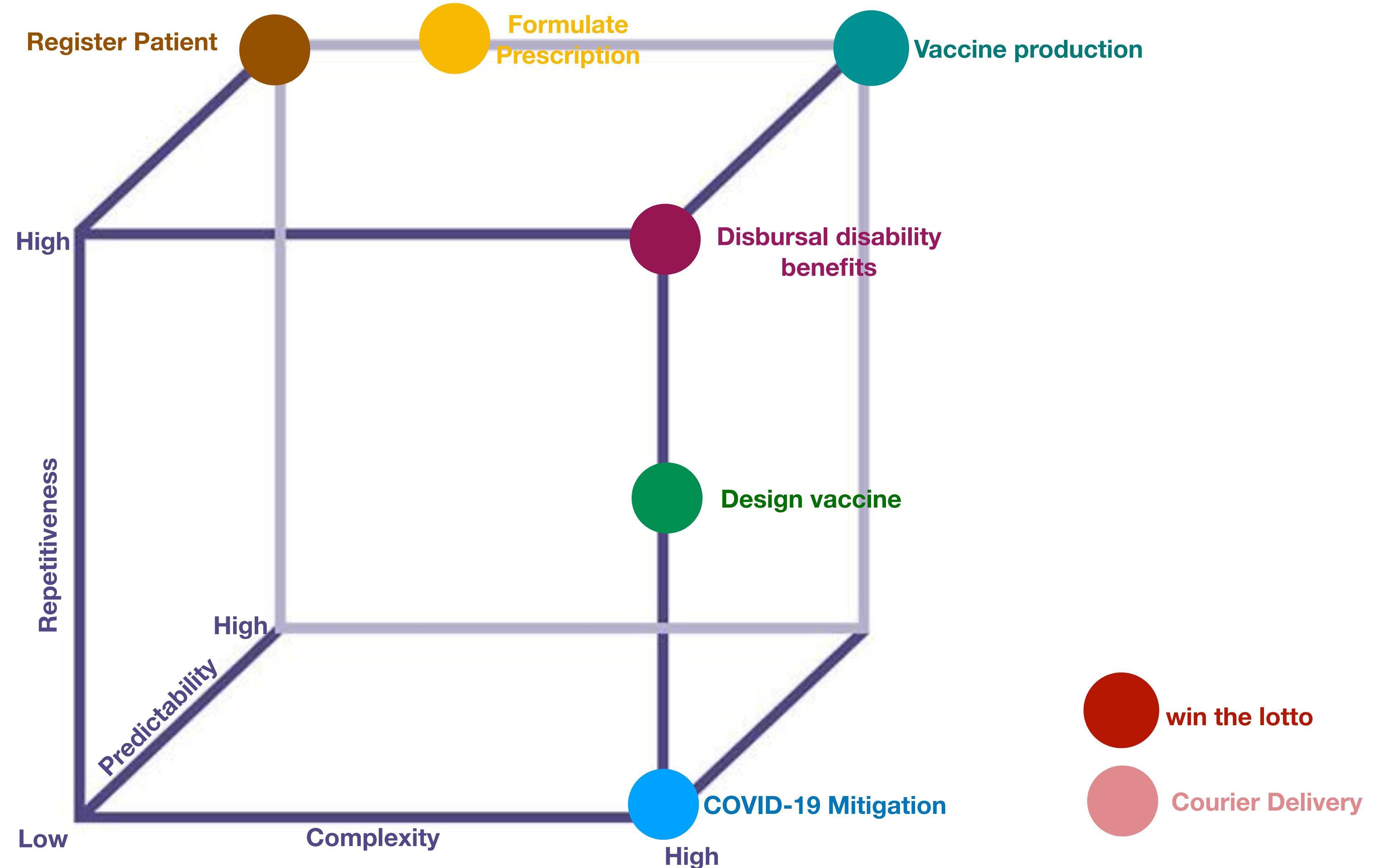
BP Dimensions



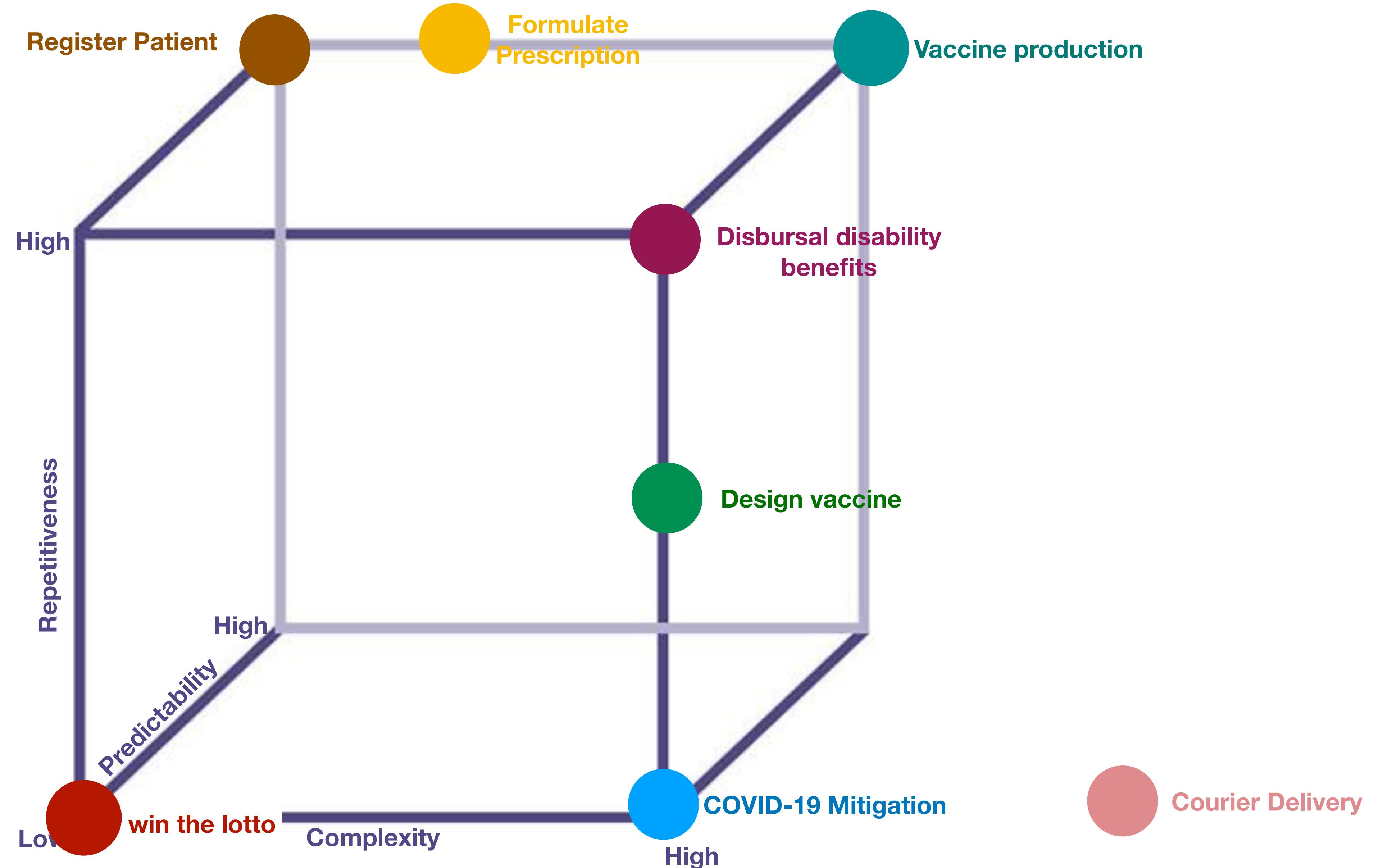
BP Dimensions



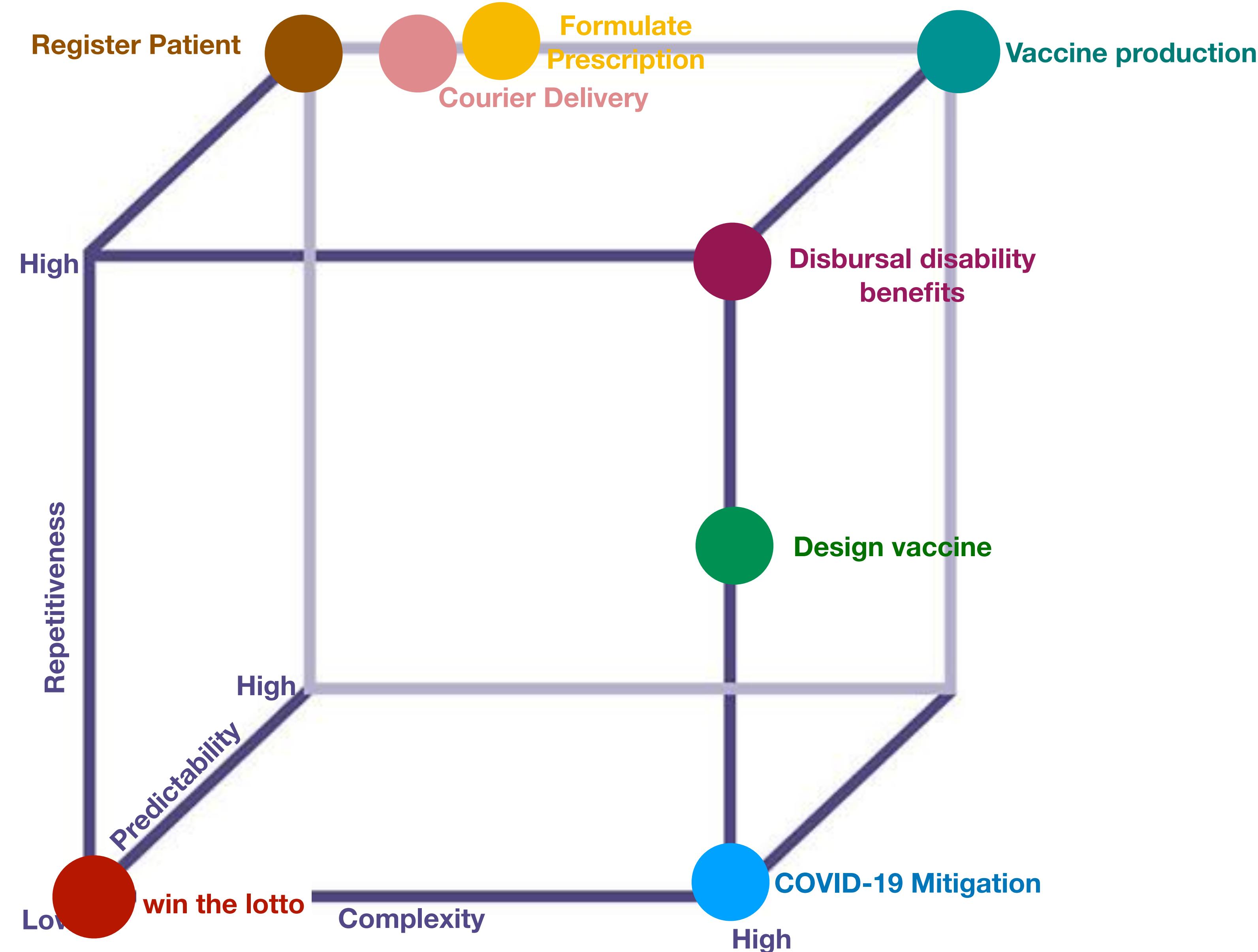
BP Dimensions



BP Dimensions



BP Dimensions



BPM Systems

BPM systems support collaboration, coordination and decision making in business processes.

- **Groupware systems** support human collaboration and co-decision.
- **Workflow management systems** explicitly control ordering, coordination and execution of activities with little human intervention.

Groupware

The screenshot shows a Trello board titled "Power-Ups For All Launch Timeline". The board has two main sections: "To Do" and "In Progress".

To Do:

- Fix alignment issue on /pricing
- Power-Up Demos for Sales
- Update assorted references to Power-Ups
- Script communications for support
- In-App final copy review

In Progress:

- Gather customer quotes/testimonials
- Illustrations for social, blog, pr
- Content audit for outdated language
- PR outreach

Power-Ups For All Launch Timeline

in list Helpful Launch Info

MEMBERS

ACTIONS

Description

Campaign name for all UTM params is "powerupsforall"

August 17

0% By 5pm EST - All items for translation sent in @brian @alexia

August 19

0% By 2pm EST - Blog post laid out - @lauren
 By 4pm EST - Marketing assets delivered to PR @brian
 By 5pm EST - blog post link and other assets sent to partners. @brian

August 22

0% By 2pm EST - Subject line tests & newsletter laid out @lauren
 By 3pm EST - Newsletter laid out in [customer.io](#) @samantha

BPMs and Decision Making



Groupware

**Adaptive Case
Management System**

**Workflow
Management
System**

BPMs and Decision Making



Groupware

**Adaptive Case
Management System**

**Workflow
Management
System**

- Activity Coordination and ordering not automated.
- Users control the ordering and coordination of activities “on the fly” while executing the business process.

BPMs and Decision Making



Groupware

- Activity Coordination and ordering not automated.
- Users control the ordering and coordination of activities “on the fly” while executing the business process.

Adaptive Case Management System

Workflow Management System

- Activity ordering highly automated.
- Users merely influence the outcome of the process by entering data requested.

BPMs and Decision Making



Groupware

- Activity Coordination and ordering not automated.
- Users control the ordering and coordination of activities “on the fly” while executing the business process.

Adaptive Case Management System

- Activity ordering “softly constrained”
- Users are free to choose activities as they deem necessary, but respecting constraints between activities.

Workflow Management System

- Activity ordering highly automated.
- Users merely influence the outcome of the process by entering data requested.

Exercise (5 min)

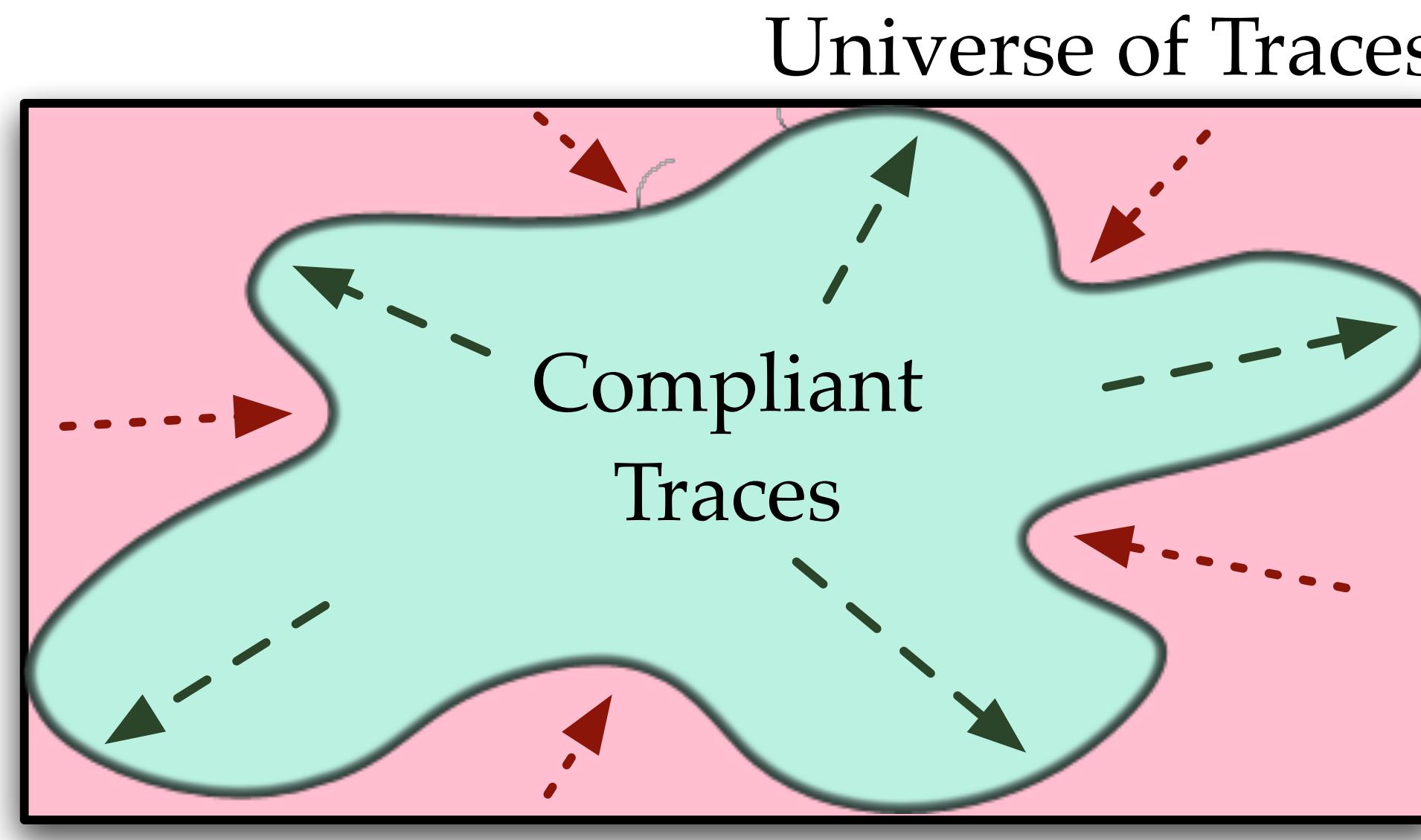
- Take the process you have selected for your project, and discuss its dimensions (predictability, complexity, repetitiveness).
- Which BPMS do you think will suit better the stakeholders in your process?
- (Take home) Investigate to which side of the decision making spectrum are the following products:
 - Trello
 - Podio
 - Salesforce

Exercise (5 min)

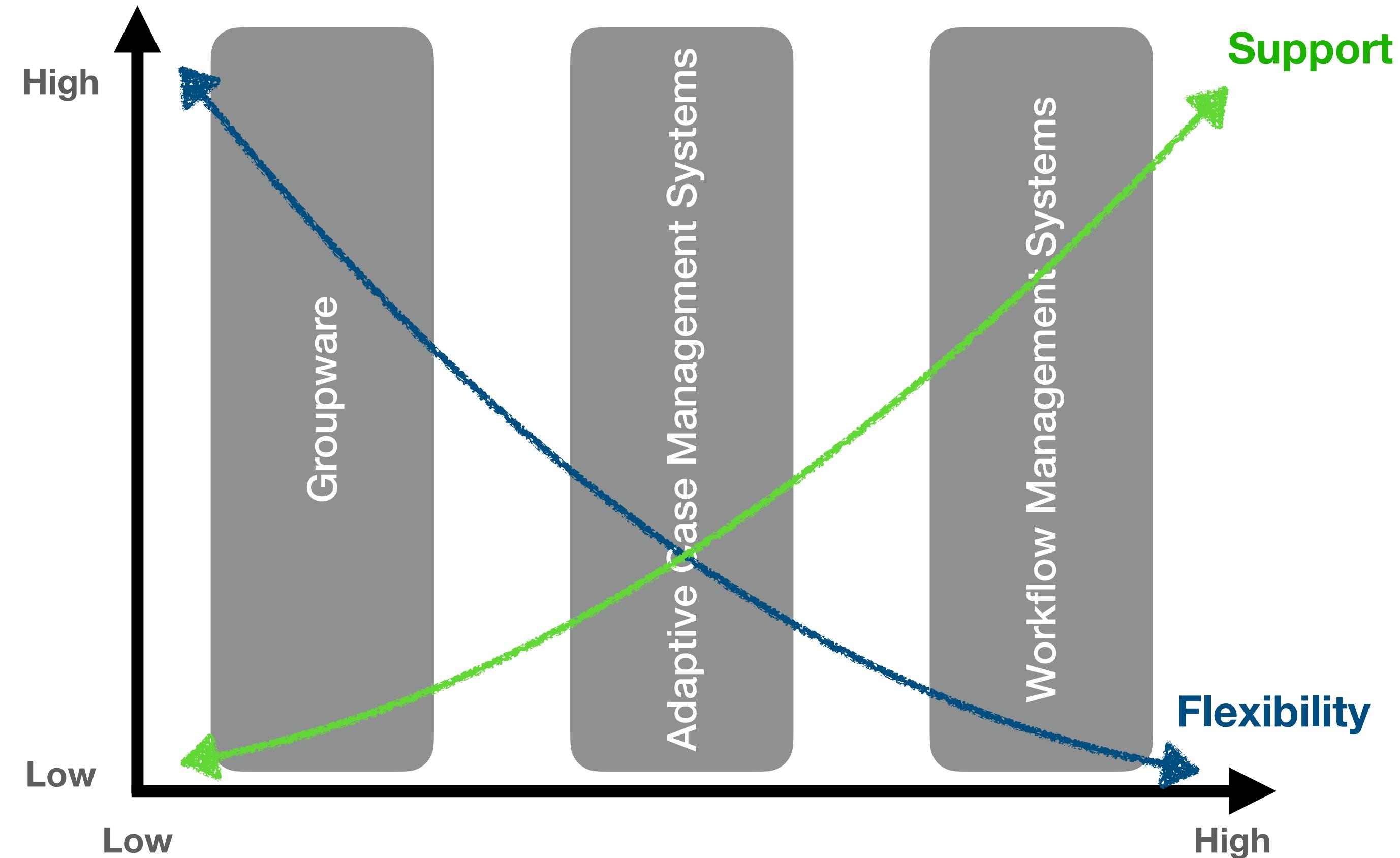
- Take the process you have selected for your project, and discuss its dimensions (predictability, complexity, repetitiveness).
- Which BPMS do you think will suit better the stakeholders in your process?
- (Take home) Investigate to which side of the decision making spectrum are the following products:
 - Trello
 - Podio
 - Salesforce

Flexibility & Support

- Flexibility: To which degree the users can take local decisions about how execute the process:
- Support: To which degree the system makes centralised decisions about how to execute the process:



Flexibility & Support



Decision Making



Trends in BPM



Imperative Modelling

Traditional, repetitive processes

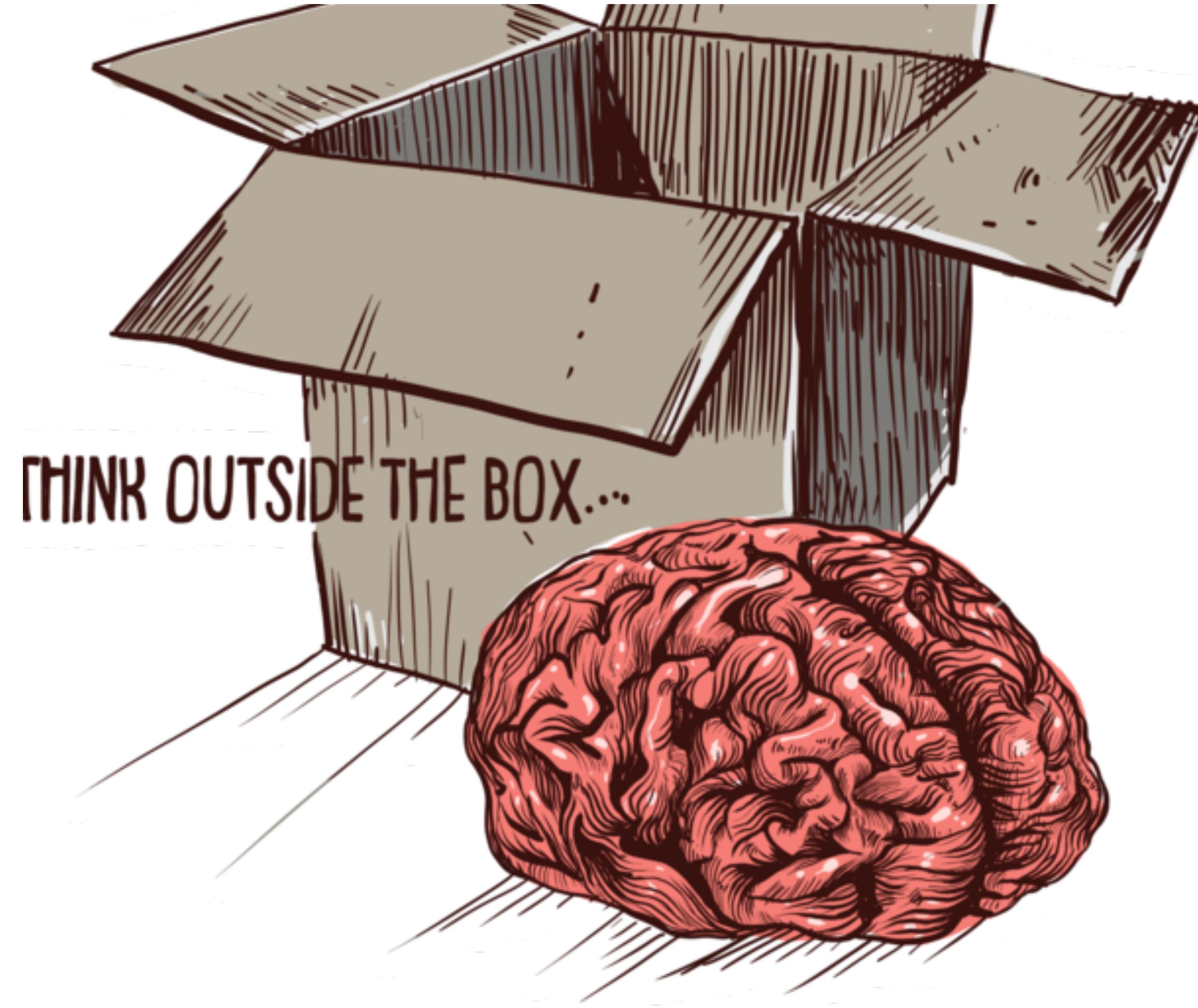
Ex. Flowcharts, Activity Diagrams, BPMN



Declarative Modelling

Knowledge-intensive, highly-variable processes

Ex. Constraints, LTL, DECLARE, DCR Graphs



Modelling Declarative Processes

What are the start and goal activities?

When a company receives an invoice, it must be registered in the accounting system and the employee responsible for the invoice must approve the expense before the deadline of the payment. When approved, the finance department must pay the invoice at the deadline of the payment.

What are resources are necessary for activities?

When a company receives an invoice, it must be registered in the accounting system and the employee responsible for the invoice must approve the expense before the deadline of the payment. When approved, the finance department must pay the invoice at the deadline of the payment.

What are resources are necessary for activities?

physical invoice

When a company receives an invoice, it must be registered in the accounting system and the employee responsible for the invoice must approve the expense before the deadline of the payment. When approved, the finance department must pay the invoice at the deadline of the payment.

What are resources are necessary for activities?

physical invoice

produce

When a company receives an invoice, it must be registered in the accounting system and the employee responsible for the invoice must approve the expense before the deadline of the payment. When approved, the finance department must pay the invoice at the deadline of the payment.

What are resources are necessary for activities?

physical invoice

produce

When a company receives an invoice, it must be registered in the accounting system and the employee responsible for the invoice must approve the expense before the deadline of the payment. When approved, the finance department must pay the invoice at the deadline of the payment.

What are resources are necessary for activities?

physical invoice

produce

When a company receives an invoice, it must be registered in the accounting system and the employee responsible for the invoice must approve the expense before the deadline of the payment. When approved, the finance department must pay the invoice at the deadline of the payment.

invoice data

What are resources are necessary for activities?

physical invoice

produce

When a company receives an invoice, it must be registered in the accounting system and the employee responsible for the invoice must approve the expense before the deadline of the payment. When approved, the finance department must pay the invoice at the deadline of the payment.

produce

invoice data

What are resources are necessary for activities?

physical invoice

produce

When a company receives an invoice, it must be registered in the accounting system and the employee responsible for the invoice must approve the expense before the deadline of the payment. When approved, the finance department must pay the invoice at the deadline of the payment.

invoice data

produce

use

What are resources are necessary for activities?

physical invoice

produce

When a company receives an invoice, it must be registered in the accounting system and the employee responsible for the invoice must approve the expense before the deadline of the payment. When approved, the finance department must pay the invoice at the deadline of the payment.

invoice data

produce

use

produce

approved invoice

What are resources are necessary for activities?

physical invoice

produce

When a company receives an invoice, it must be registered in the accounting system and the employee responsible for the invoice must approve the expense before the deadline of the payment. When approved, the finance department must pay the invoice at the deadline of the payment.

invoice data

produce

use

approved invoice

produce

use

How are activities ordered?

When a company receives an invoice, it **must** be registered in the accounting system and the employee responsible for the invoice **must** approve the expense before the deadline of the payment. When approved, the finance department **must** pay the invoice at the deadline of the payment.

How are activities ordered?

When a company receives an invoice, it **must** be registered in the accounting system and the employee responsible for the invoice **must** approve the expense before the deadline of the payment. When approved, the finance department **must** pay the invoice at the deadline of the payment.

Scenarios!

How are activities ordered?

When a company receives an invoice, it must be registered in the accounting system and the employee responsible for the invoice must approve the expense before the deadline of the payment. When approved, the finance department must pay the invoice at the deadline of the payment.

Scenarios

- Receive - Approve - Register - Pay
- Receive - Register - Approve - Pay

We should support this!

Positive

How are activities ordered?

When a company receives an invoice, it must be registered in the accounting system and the employee responsible for the invoice must approve the expense before the deadline of the payment. When approved, the finance department must pay the invoice at the deadline of the payment.

Scenarios

- Receive - Approve - Register - Pay
- Receive - Register - Approve - Pay

We should support this!

Positive

- Deadline - Pay
- Receive - Register - Approve - Pay - Pay

We want to avoid this!

Negative

How are activities ordered?

When a company receives an invoice, it must be registered in the accounting system and the employee responsible for the invoice must approve the expense before the deadline of the payment. When approved, the finance department must pay the invoice at the deadline of the payment.

Scenarios

- Receive - Approve - Register - Pay
- Receive - Register - Approve - Pay

We should support this!

Positive

- Receive - Register - Pay

**We might support this
(partial trace)**

Neutral

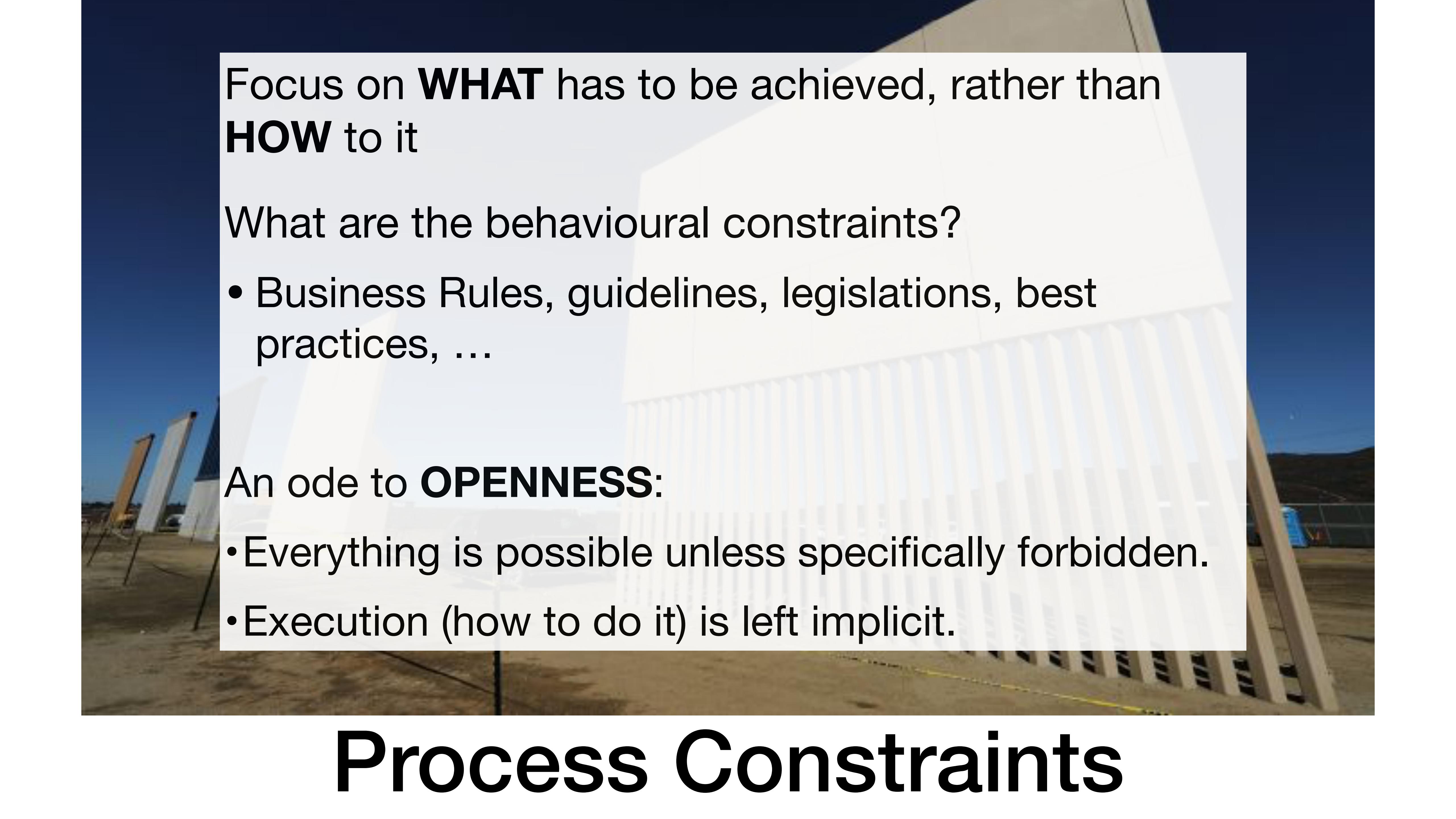
- Deadline - Pay
- Receive - Register - Approve - Pay - Pay

We want to avoid this!

Negative



Process Constraints



Focus on **WHAT** has to be achieved, rather than **HOW** to it

What are the behavioural constraints?

- Business Rules, guidelines, legislations, best practices, ...

An ode to **OPENNESS**:

- Everything is possible unless specifically forbidden.
- Execution (how to do it) is left implicit.

Process Constraints

Constraints in DCR graphs

Much richer than the simple “precedence sequence flow” connector in imperative languages.

Type of constraints

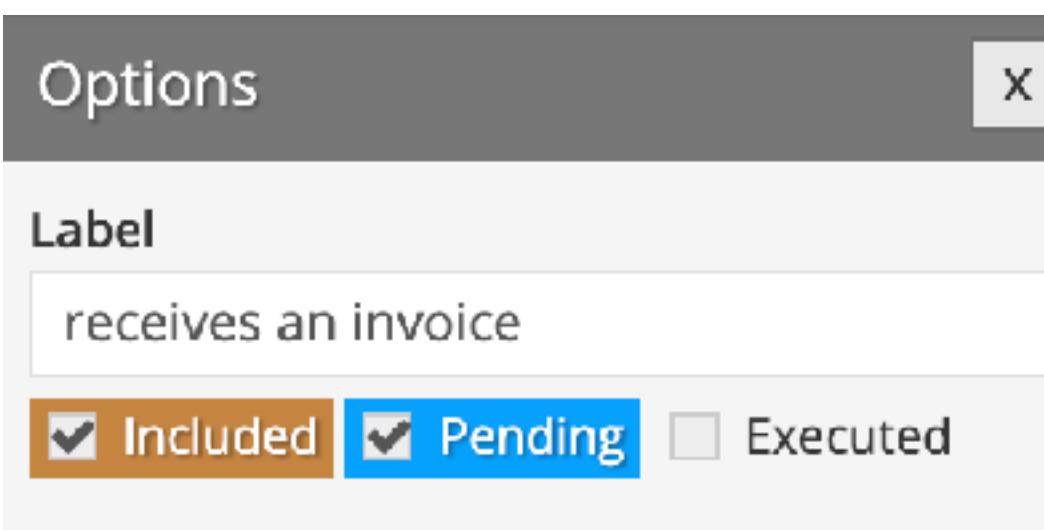
- Conditions
- Responses & Non-responses
- Dynamic Inclusion
- Dynamic Exclusion
- Milestones

Activities: Initial State

When a company receives an invoice, it must be registered in the accounting system and the employee responsible for the invoice must approve the expense before the deadline of the payment. When approved, the finance department must pay the invoice at the deadline of the payment.

Activities: Initial State

When a company receives an invoice, it must be registered in the accounting system and the employee responsible for the invoice must approve the expense before the deadline of the payment. When approved, the finance department must pay the invoice at the deadline of the payment.

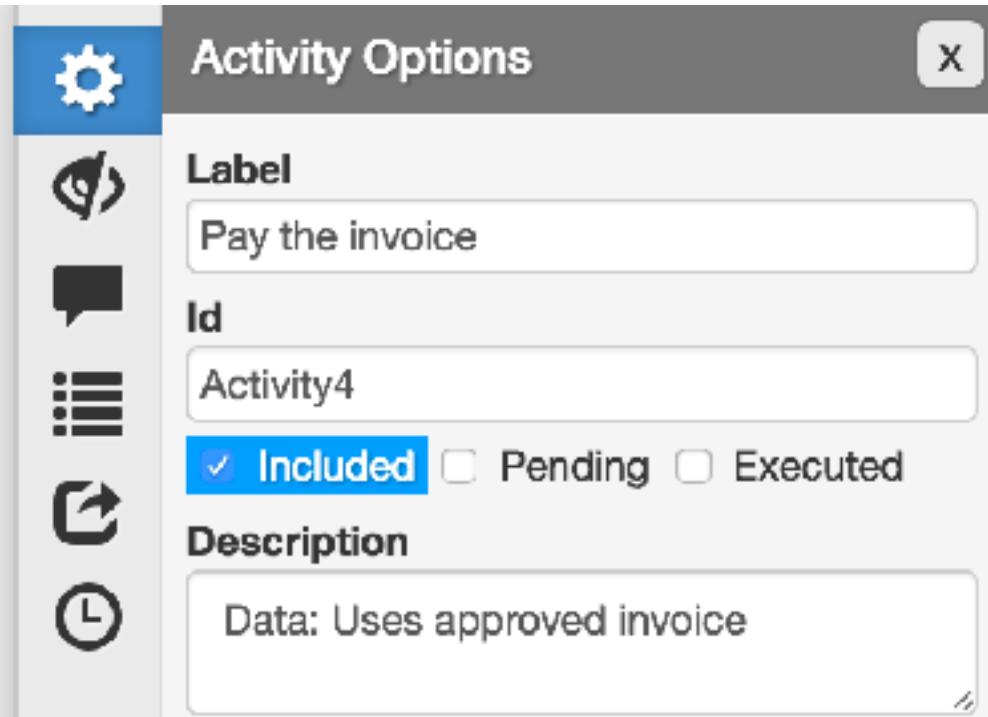


Activities: Final State(s)

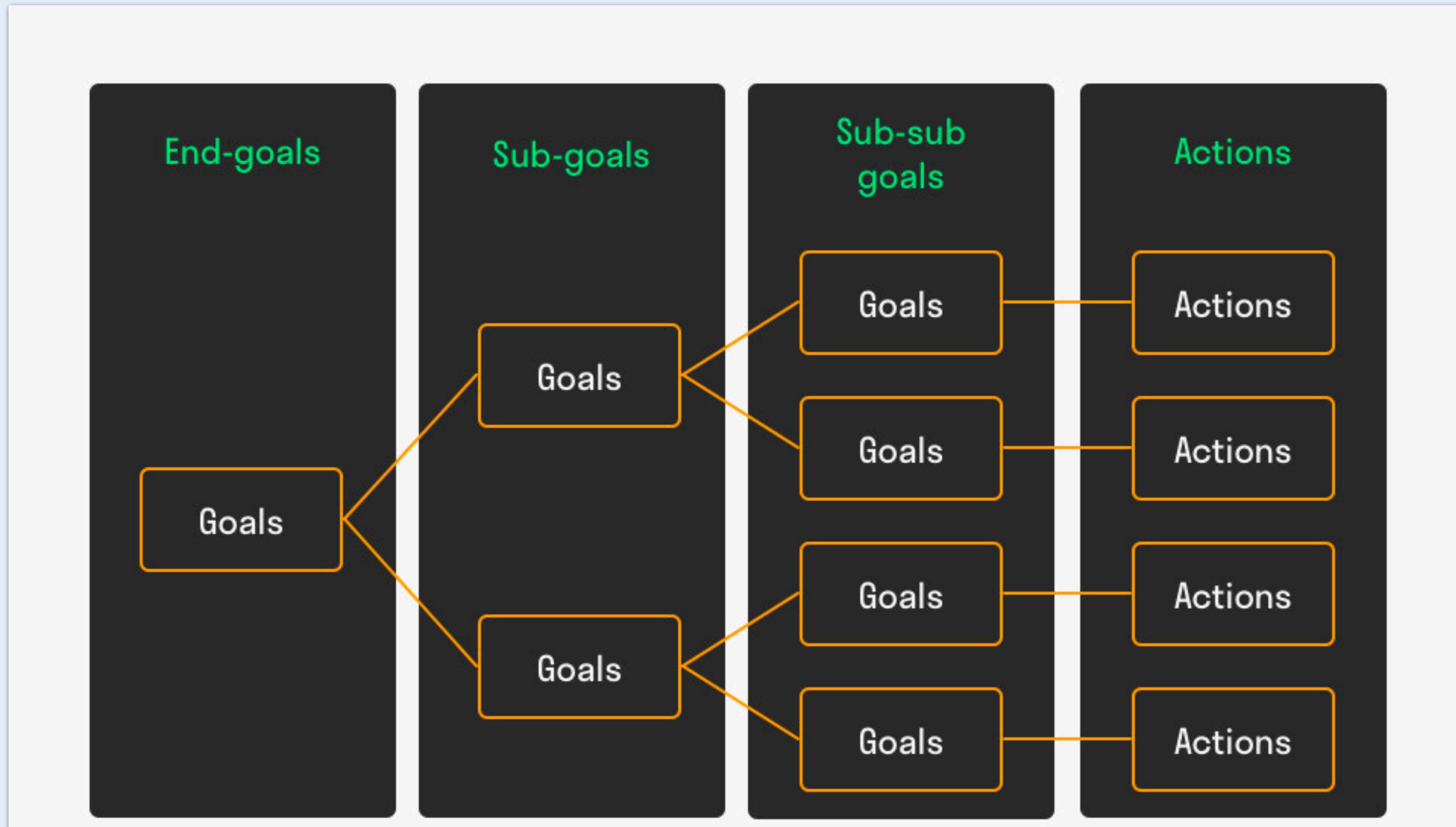
When a company receives an invoice, it must be registered in the accounting system and the employee responsible for the invoice must approve the expense before the deadline of the payment. When approved, the finance department must pay the invoice at the deadline of the payment.

Activities: Final State(s)

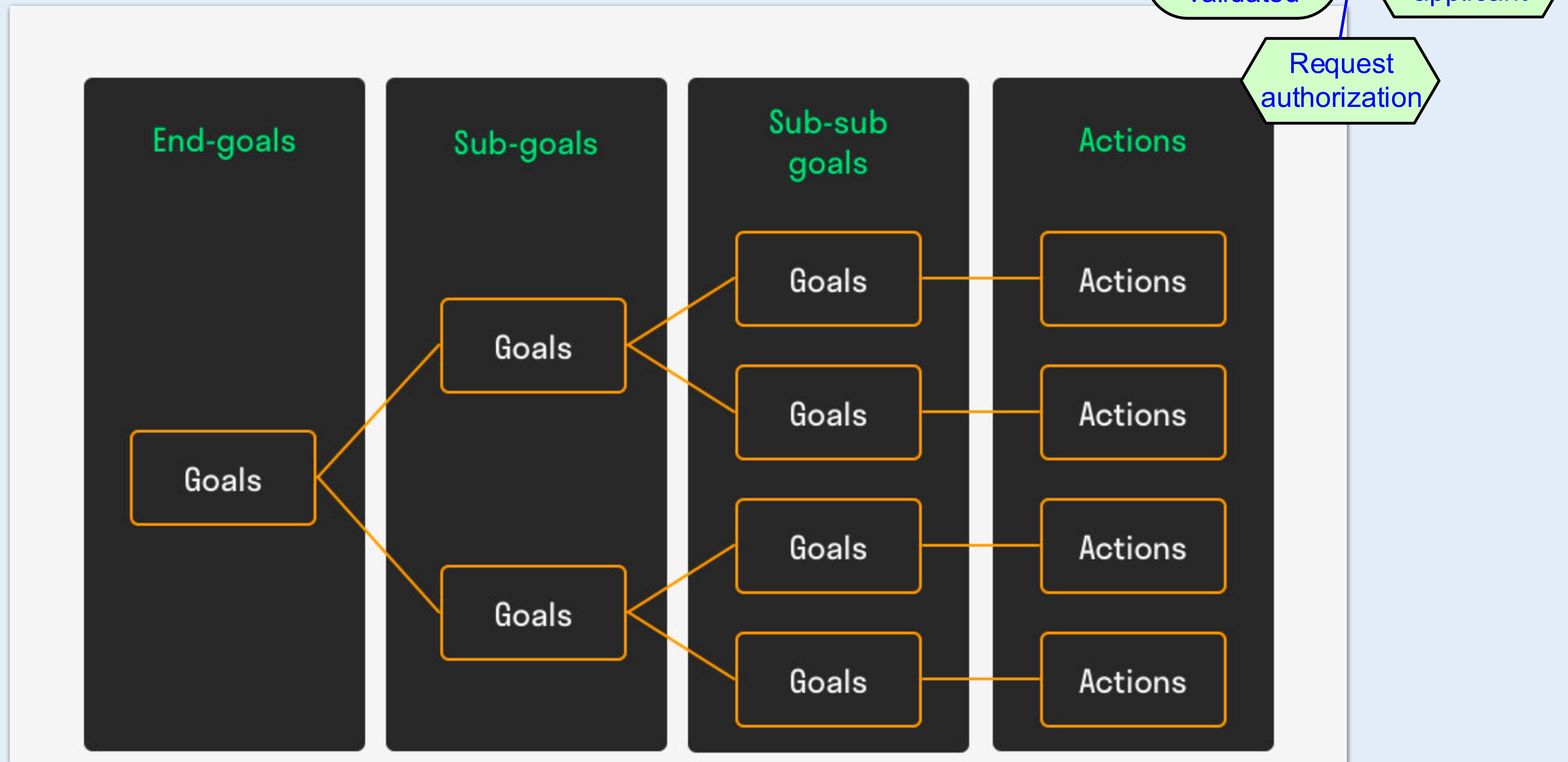
When a company receives an invoice, it must be registered in the accounting system and the employee responsible for the invoice must approve the expense before the deadline of the payment. When approved, the finance department must pay the invoice at the deadline of the payment.



Means-End Analysis



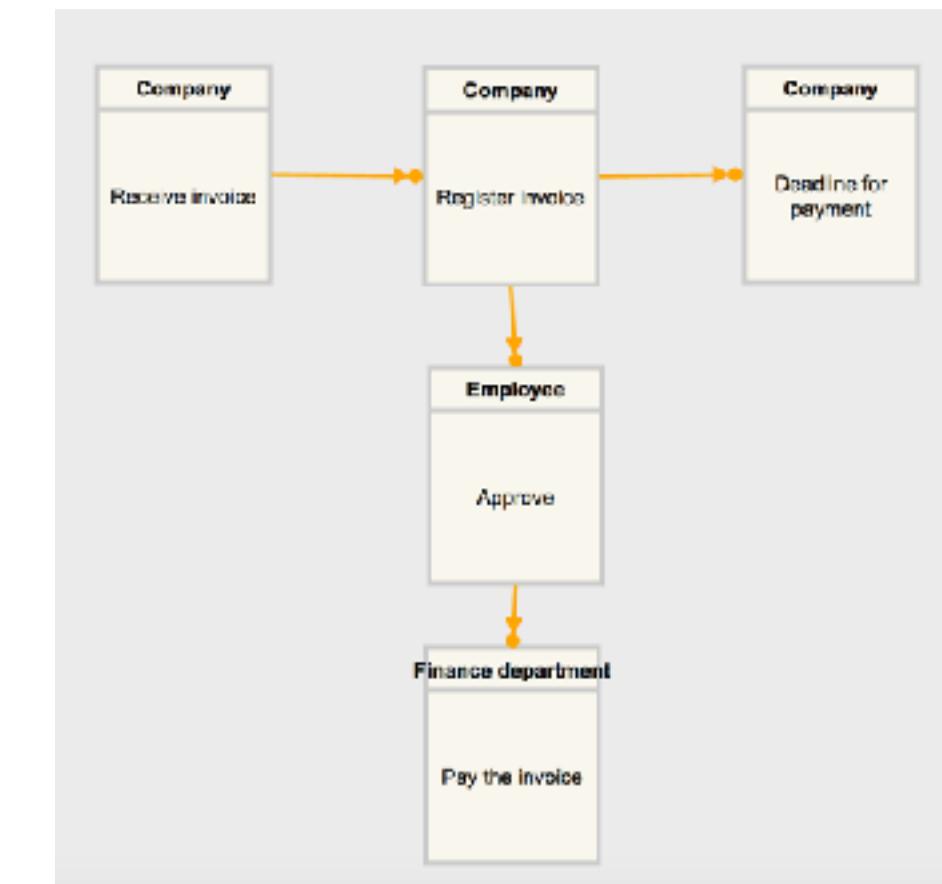
Means-End Analysis



Means End Analysis via Conditions

A condition denotes a simple precedence constraint. A $\rightarrow\bullet$ B expresses that B cannot be done before A is done.

When a company receives an invoice, it must be registered in the accounting system and the employee responsible for the invoice must approve the expense before the deadline of the payment. When approved, the finance department must pay the invoice at the deadline of the payment.

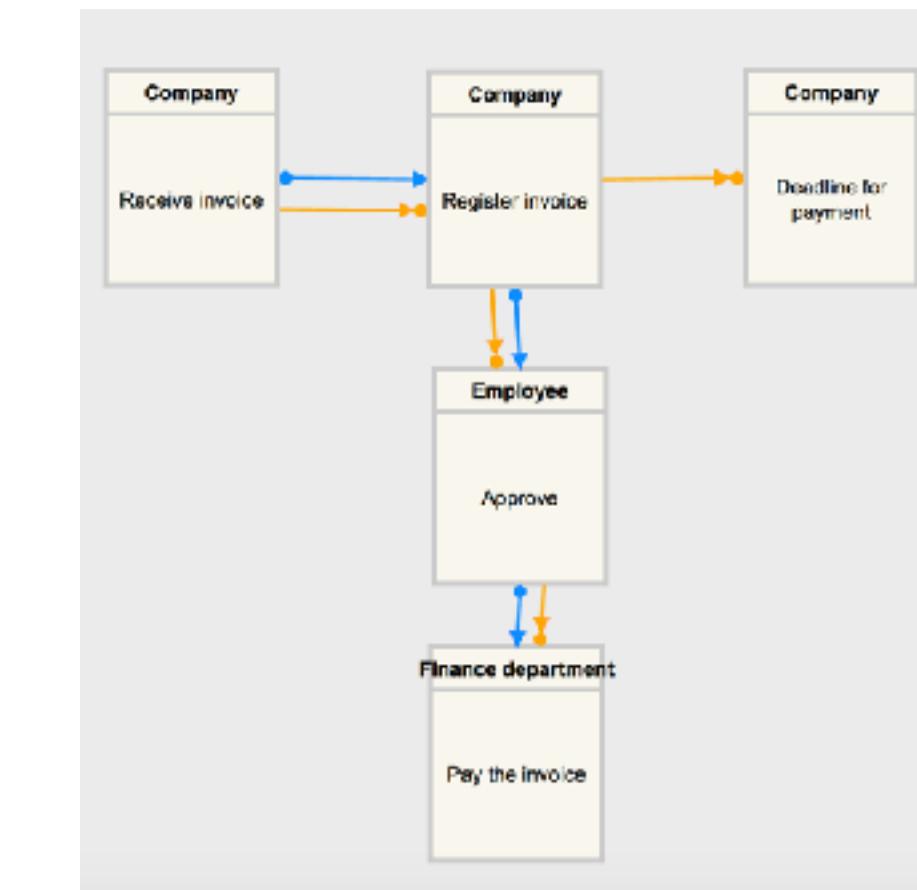
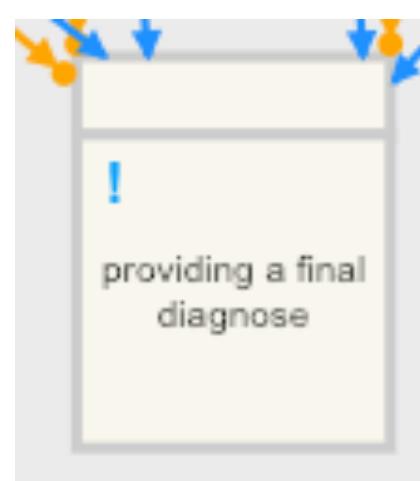


Responses

Denotes an obligation constraint. A $\bullet \rightarrow$ B if executing A deems the execution of B obligatory.

When a company receives an invoice, it **must** be registered in the accounting system and the employee responsible for the invoice **must approve** the expense before the deadline of the payment. When approved, the finance department **must pay the invoice** at the deadline of the payment.

An activity that must be executed has a **pending (!)** state



Dynamic Exclusions

Used to establish negation/inhibition. A →% B expresses that the execution of A inhibits the execution of B.

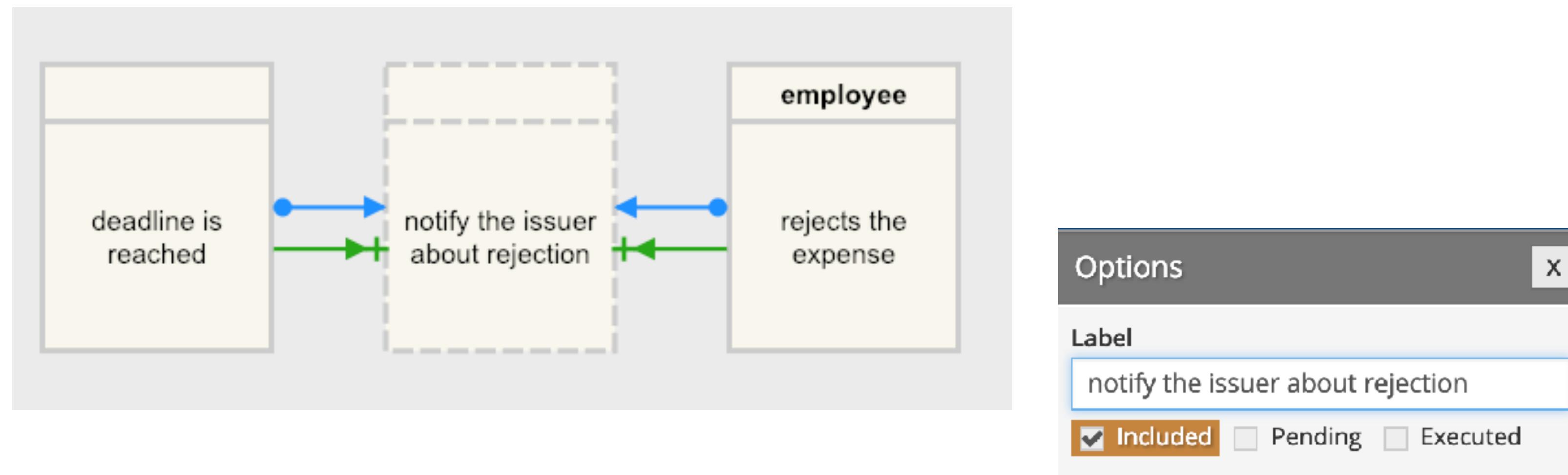
When a company receives an invoice, it must be registered in the accounting system and the employee responsible for the invoice must approve the expense before the deadline of the payment. When approved, the finance department must pay the invoice at the deadline of the payment.

How many times can you pay an invoice?

Dynamic Inclusions

Denotes a permission constraint. $A \rightarrow + B$ if executing A makes B available for execution.

When a company receives an invoice, it **must** be registered in the accounting system and the employee responsible for the invoice **must** approve the expense before the deadline of the payment. If the deadline is reached or the employee rejects the expense, a notification is given to the issuer



DCR graphs - briefly

- Assume:
 - A fixed universe of events \mathcal{E} , and labels \mathcal{L}
 - $e, f \in \mathcal{E}$
 - A mapping λ from events to labels
- The grammar for DCR processes is given as:

$P, Q ::= [M] \lambda T$

process

$$T, U ::= \begin{array}{l|l} e \xrightarrow{\cdot} f & e \xrightarrow{\bullet} f \\ | & | \\ e \rightarrow + f & e \rightarrow \% f \\ | & | \\ e \rightarrow \diamond f & T \parallel U \\ | & | \\ 0 & \end{array}$$

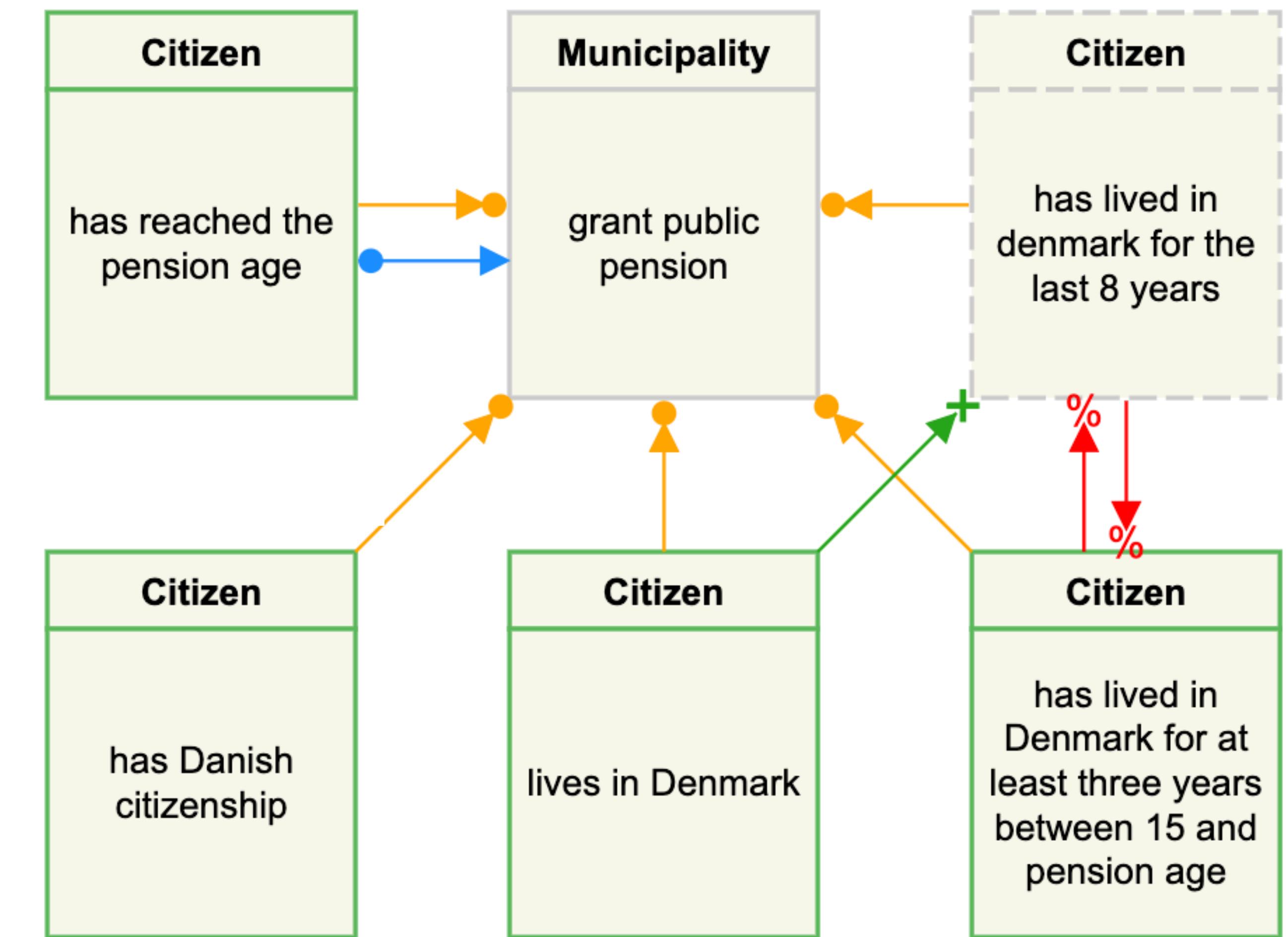
$M, N ::= M, e : \Phi \mid \epsilon$ marking
 $\lambda ::= \lambda, e : l \mid \epsilon$ labelling

$\Phi ::= (h, i, p)$

$h ::= t \mid f$

$i ::= t \mid f$

$p ::= t \mid f$



DCR graphs - briefly

- Assume:
 - A fixed universe of events \mathcal{E} , and labels \mathcal{L}
 - $e, f \in \mathcal{E}$
 - A mapping λ from events to labels
- The grammar for DCR processes is given as:

$P, Q ::= [M] \lambda T$ process

$T, U ::= e \xrightarrow{\cdot} f \quad | \quad e \xrightarrow{\bullet} f$
 $| \quad e \xrightarrow{+} f \quad | \quad e \xrightarrow{\%} f$
 $| \quad e \xrightarrow{\diamond} f \quad | \quad T \parallel U$
 $| \quad 0$

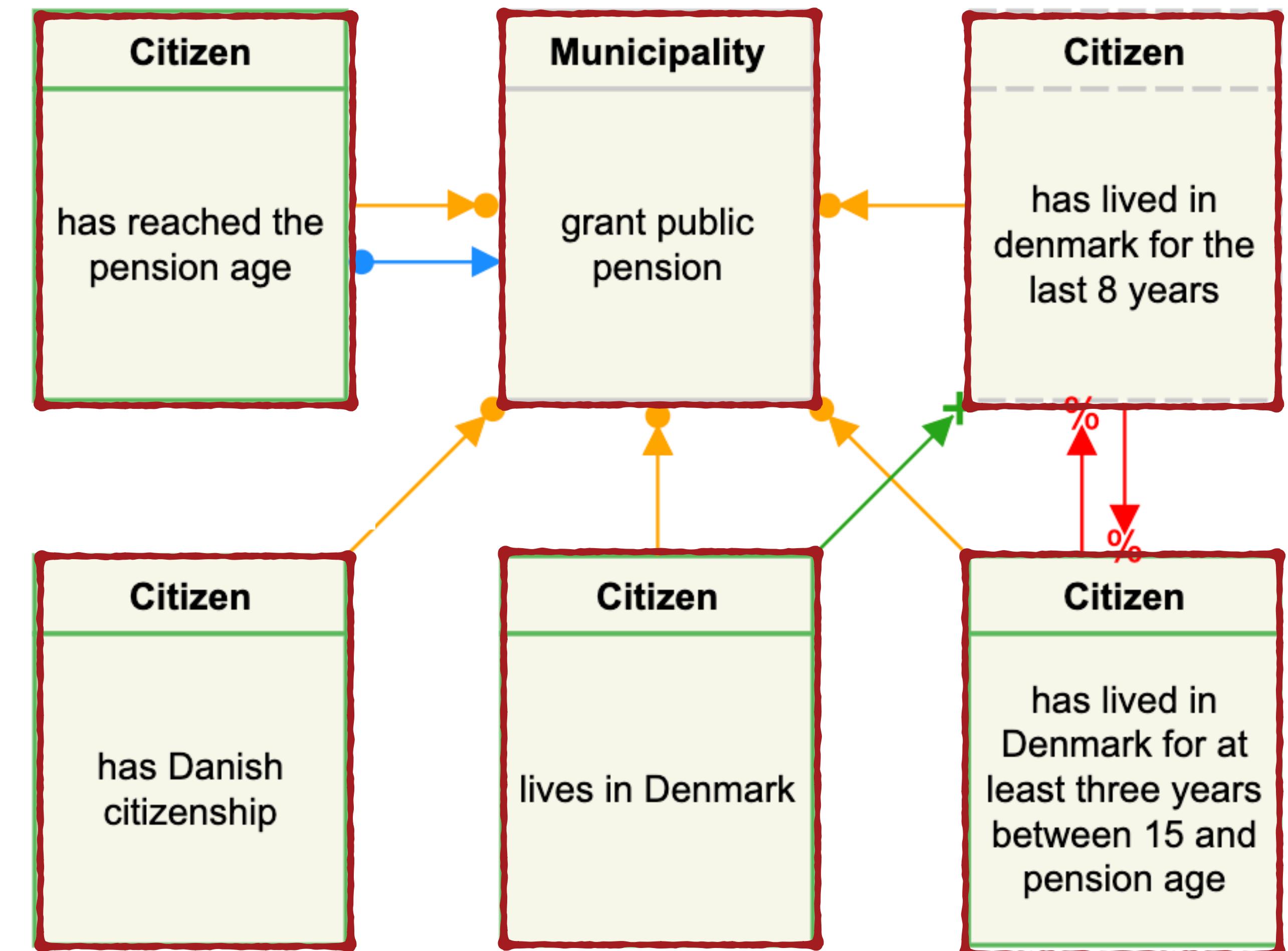
$M, N ::= M, e : \Phi \mid \epsilon$ marking
 $\lambda ::= \lambda, e : l \mid \epsilon$ labelling

$\Phi ::= (h, i, p)$

$h ::= t \mid f$

$i ::= t \mid f$

$p ::= t \mid f$



DCR graphs - briefly

- Assume:
 - A fixed universe of events \mathcal{E} , and labels \mathcal{L}
 - $e, f \in \mathcal{E}$
 - A mapping λ from events to labels
- The grammar for DCR processes is given as:

$P, Q ::= [M] \lambda T$

process

$$T, U ::= \begin{array}{l|l} e \xrightarrow{\cdot} f & e \xrightarrow{\bullet} f \\ | & | \\ e \rightarrow + f & e \rightarrow \% f \\ | & | \\ e \rightarrow \diamond f & T \parallel U \\ | & | \\ 0 & \end{array}$$

$M, N ::= M, e : \Phi \mid \epsilon$

marking

$\lambda ::= \lambda, e : l \mid \epsilon$

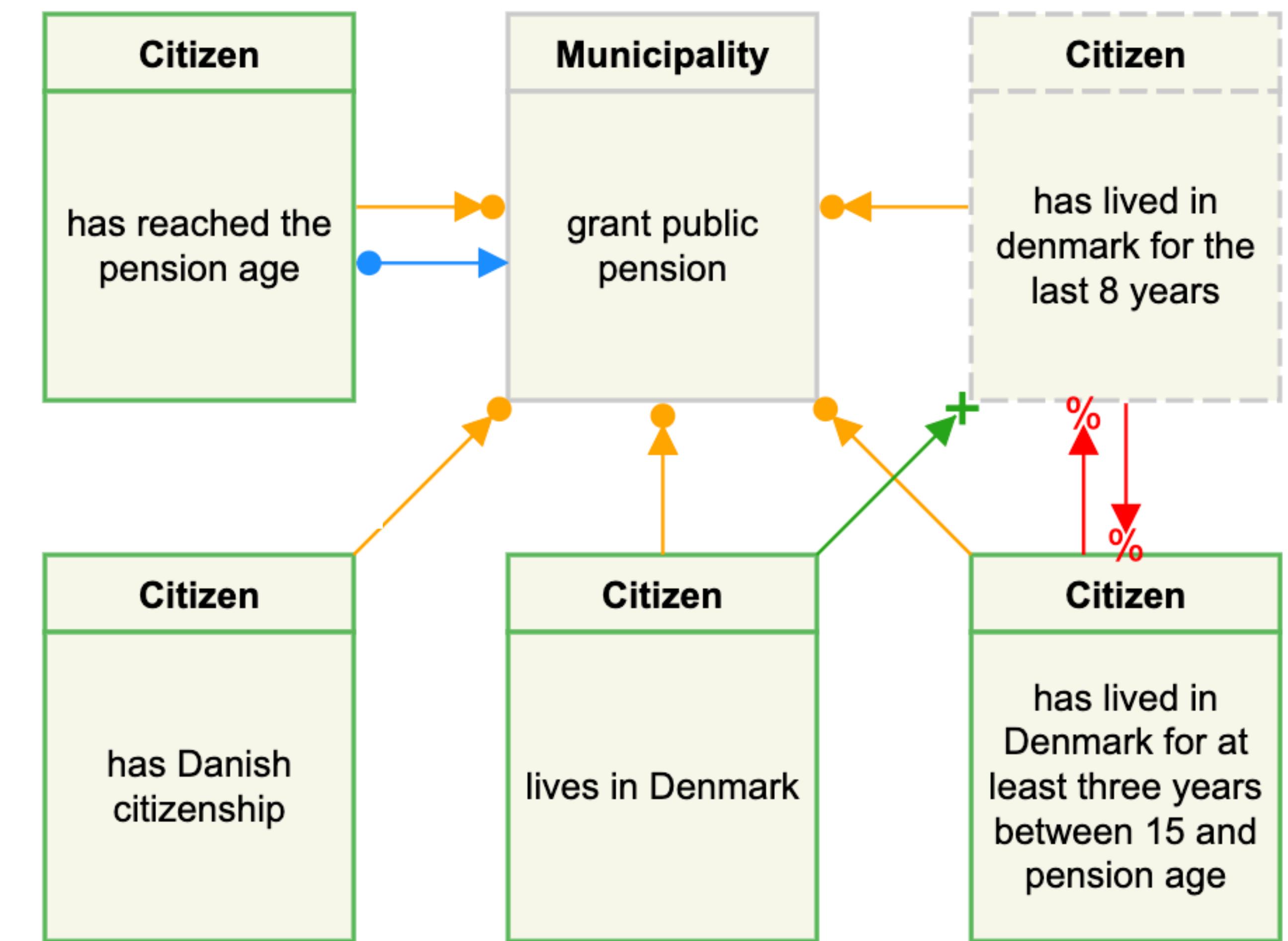
labelling

$\Phi ::= (h, i, p)$

$h ::= t \mid f$

$i ::= t \mid f$

$p ::= t \mid f$



DCR graphs - briefly

- Assume:
 - A fixed universe of events \mathcal{E} , and labels \mathcal{L}
 - $e, f \in \mathcal{E}$
 - A mapping λ from events to labels
- The grammar for DCR processes is given as:

$P, Q ::= [M] \lambda T$

process

$$T, U ::= \boxed{e \xrightarrow{\cdot} f} \mid e \xrightarrow{\bullet} f \mid e \xrightarrow{+} f \mid e \xrightarrow{\%} f \mid e \xrightarrow{\diamond} f \mid T \parallel U \mid 0$$

$M, N ::= M, e : \Phi \mid \epsilon$

marking

$\lambda ::= \lambda, e : l \mid \epsilon$

labelling

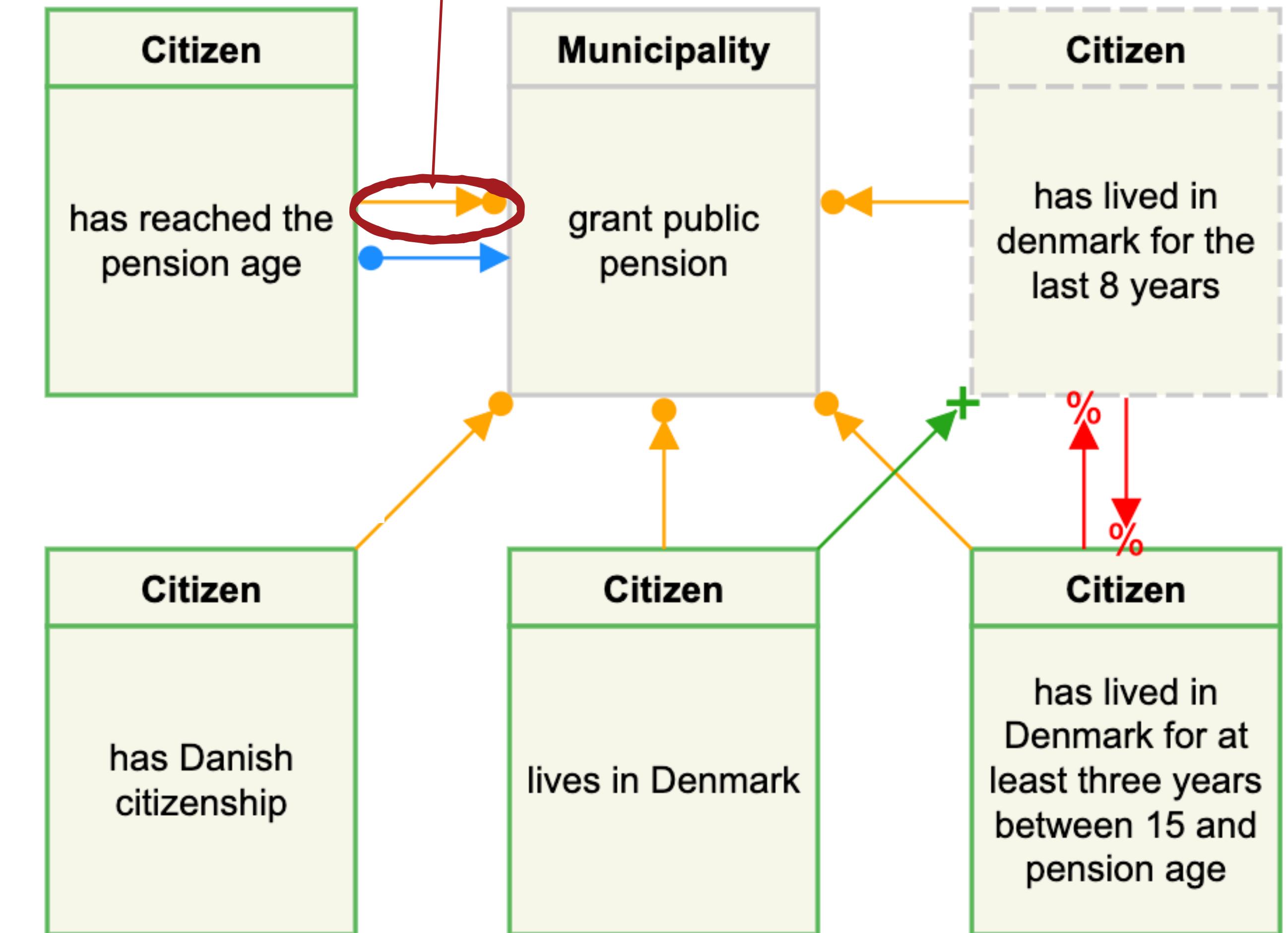
$\Phi ::= (h, i, p)$

$h ::= t \mid f$

$i ::= t \mid f$

$p ::= t \mid f$

Condition
relation



DCR graphs - briefly

- Assume:
 - A fixed universe of events \mathcal{E} , and labels \mathcal{L}
 - $e, f \in \mathcal{E}$
 - A mapping λ from events to labels
- The grammar for DCR processes is given as:

$$P, Q ::= [M] \lambda T$$

process

$$T, U ::= e \xrightarrow{\cdot} f \quad | \quad e \xrightarrow{+} f \quad | \quad e \xrightarrow{\diamond} f \quad | \quad T \parallel U$$

$| \quad 0$

$$M, N ::= M, e : \Phi \mid \epsilon$$

marking

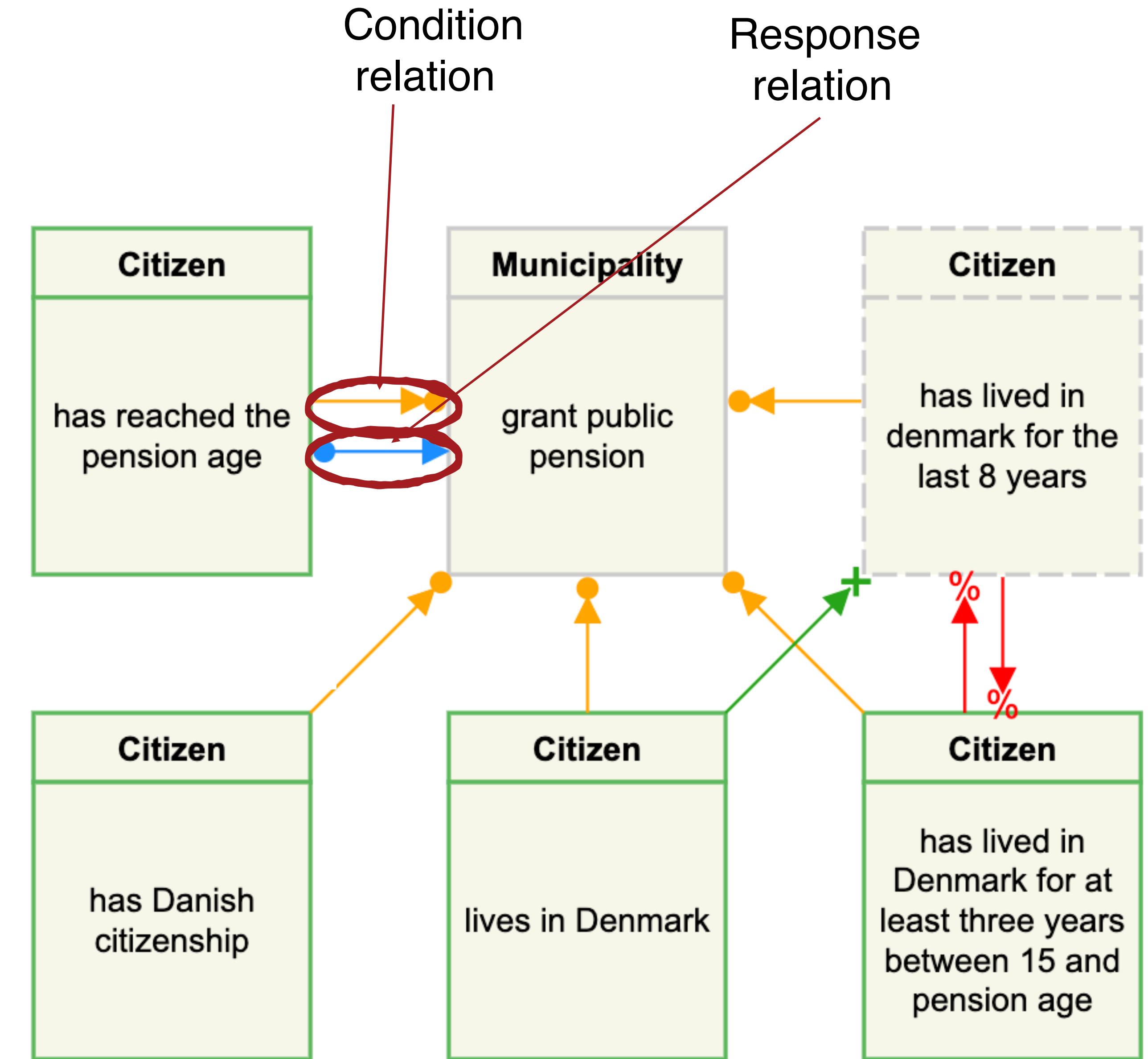
$$\lambda ::= \lambda, e : l \mid \epsilon$$

labelling

$$\Phi ::= (h, i, p)$$

$$h ::= t \mid f$$

$$i ::= t \mid f$$

$$p ::= t \mid f$$


DCR graphs - briefly

- Assume:
 - A fixed universe of events \mathcal{E} , and labels \mathcal{L}
 - $e, f \in \mathcal{E}$
 - A mapping λ from events to labels
- The grammar for DCR processes is given as:

$$P, Q ::= [M] \lambda T$$

process

$$T, U ::= e \xrightarrow{\cdot} f \quad | \quad e \xrightarrow{\bullet} f \\ | \quad e \xrightarrow{+} f \quad | \quad e \xrightarrow{\%} f \\ | \quad e \xrightarrow{\diamond} f \quad | \quad T \parallel U \\ | \quad 0$$

$$M, N ::= M, e : \Phi \quad | \quad \epsilon$$

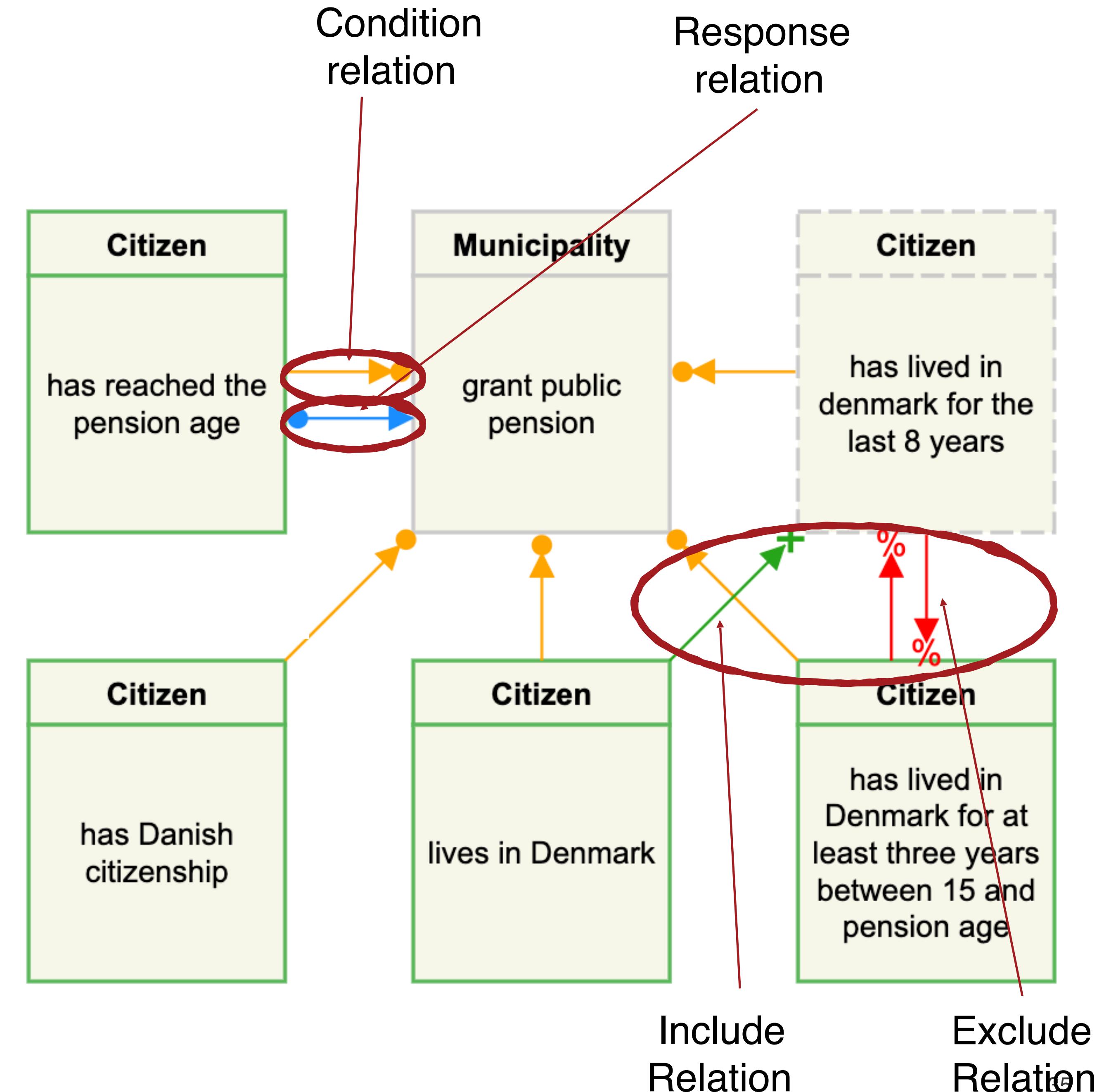
marking
labelling

$$\lambda ::= \lambda, e : l \quad | \quad \epsilon$$

$$\Phi ::= (h, i, p)$$

$$h ::= t \mid f$$

$$i ::= t \mid f$$

$$p ::= t \mid f$$


DCR graphs - briefly

- Assume:
 - A fixed universe of events \mathcal{E} , and labels \mathcal{L}
 - $e, f \in \mathcal{E}$
 - A mapping λ from events to labels
- The grammar for DCR processes is given as:

$P, Q ::= [M] \lambda T$

process

$$T, U ::= e \xrightarrow{\cdot} f \quad | \quad e \xrightarrow{\bullet} f \\ | \quad e \xrightarrow{+} f \quad | \quad e \xrightarrow{\circ} f \\ | \quad e \xrightarrow{\diamond} f \quad | \quad T \parallel U \\ | \quad 0$$

$M, N ::= M, e : \Phi \mid \epsilon$

marking

$\lambda ::= \lambda, e : l \mid \epsilon$

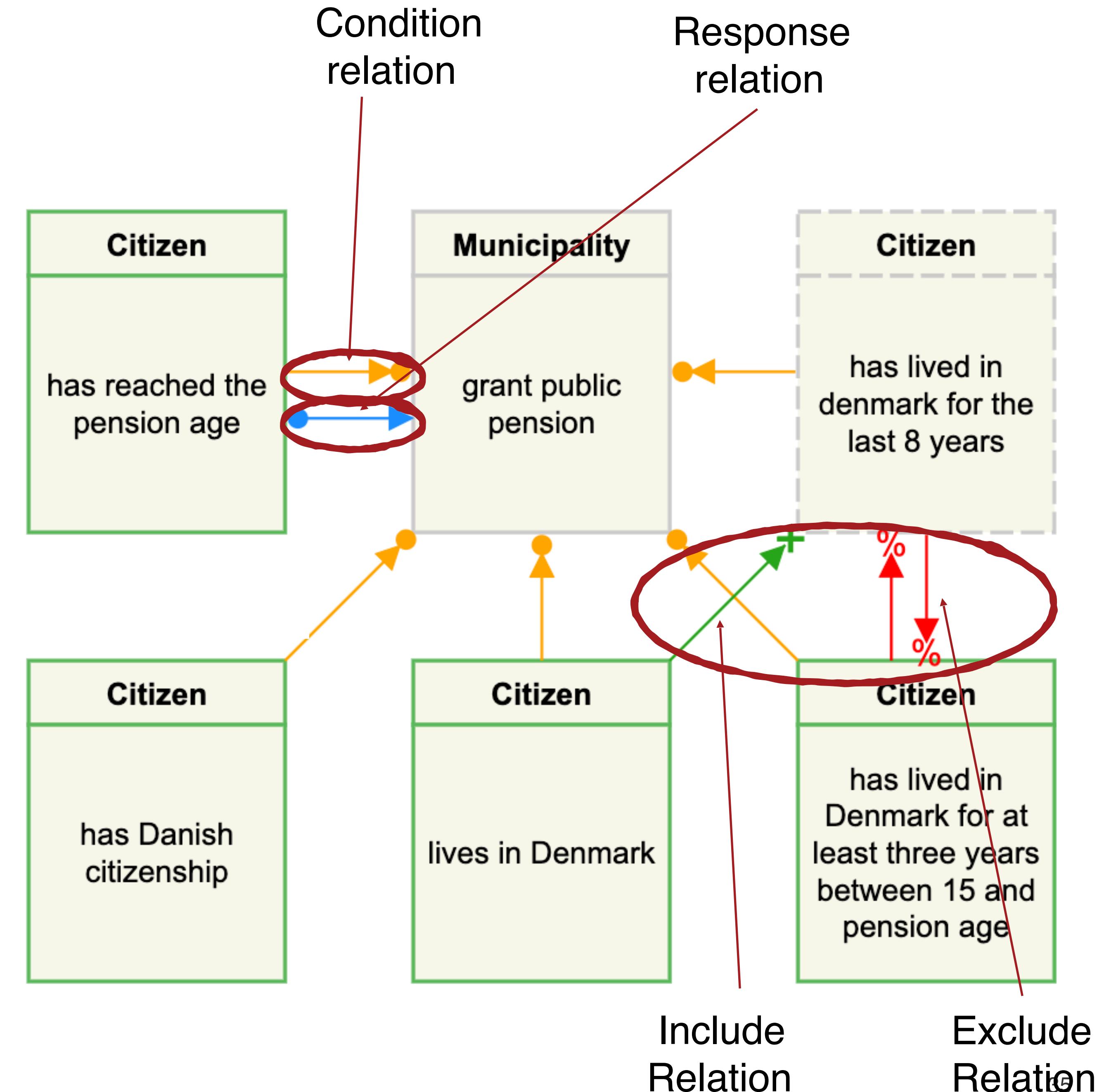
labelling

$\Phi ::= (h, i, p)$

$h ::= t \mid f$

$i ::= t \mid f$

$p ::= t \mid f$



DCR graphs - briefly

- Assume:
 - A fixed universe of events \mathcal{E} , and labels \mathcal{L}
 - $e, f \in \mathcal{E}$
 - A mapping λ from events to labels
- The grammar for DCR processes is given as:

$P, Q ::= [M] \lambda T$

process

$$T, U ::= e \xrightarrow{\cdot} f \quad | \quad e \xrightarrow{\bullet} f \\ | \quad e \xrightarrow{+} f \quad | \quad e \xrightarrow{\%} f \\ | \quad e \xrightarrow{\diamond} f \quad | \quad T \parallel U \\ | \quad 0$$

$M, N ::= M, e : \Phi \mid \epsilon$

marking

$\lambda ::= \lambda, e : l \mid \epsilon$

labelling

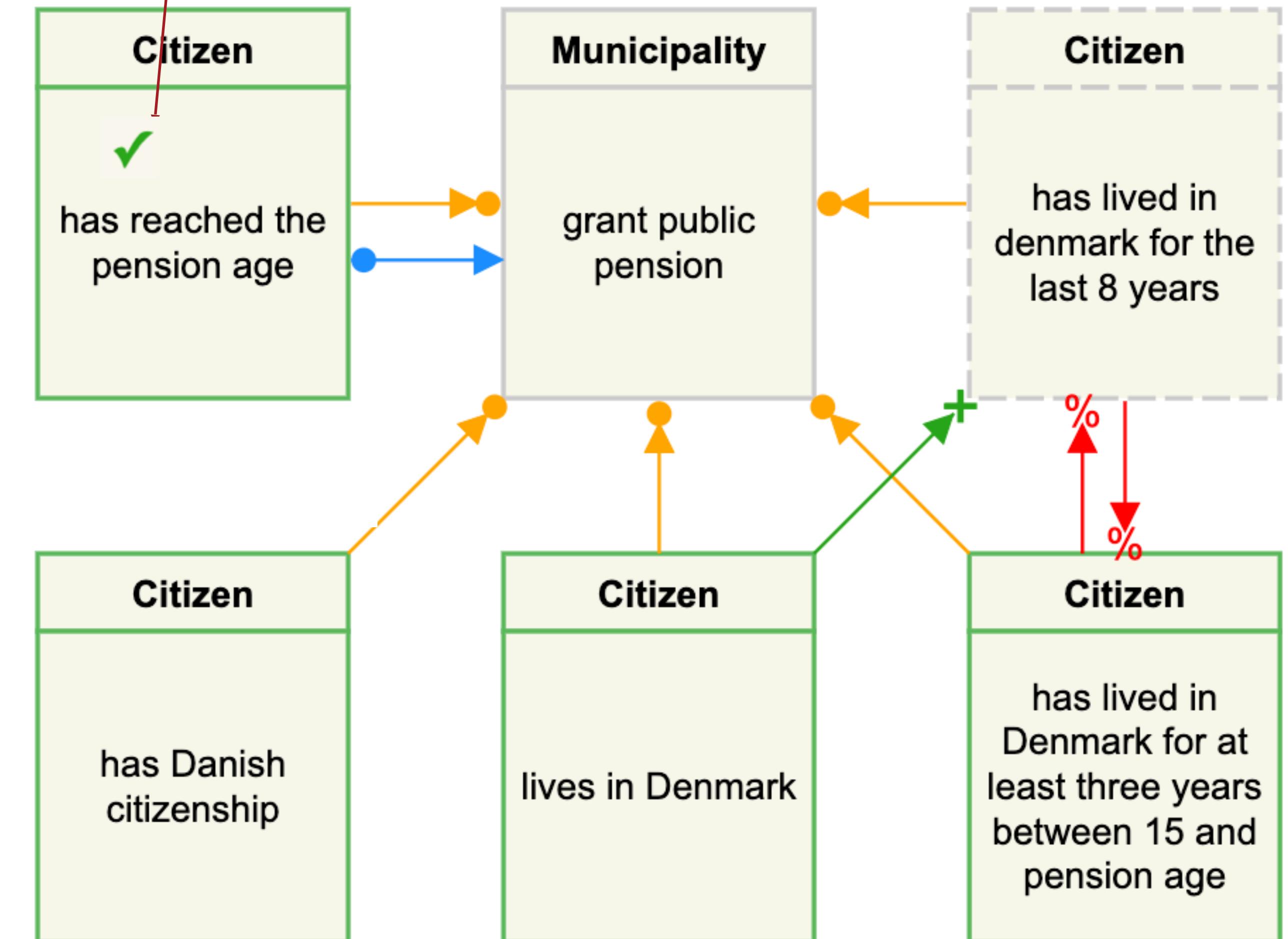
$\Phi ::= (h, i, p)$

$h ::= t \mid f$

$i ::= t \mid f$

$p ::= t \mid f$

Executed event



DCR graphs - briefly

- Assume:
 - A fixed universe of events \mathcal{E} , and labels \mathcal{L}
 - $e, f \in \mathcal{E}$
 - A mapping λ from events to labels
- The grammar for DCR processes is given as:

$P, Q ::= [M] \lambda T$

process

$$T, U ::= e \xrightarrow{\cdot} f \quad | \quad e \xrightarrow{\bullet} f \\ | \quad e \xrightarrow{+} f \quad | \quad e \xrightarrow{\%} f \\ | \quad e \xrightarrow{\diamond} f \quad | \quad T \parallel U \\ | \quad 0$$

$M, N ::= M, e : \Phi \mid \epsilon$

marking

$\lambda ::= \lambda, e : l \mid \epsilon$

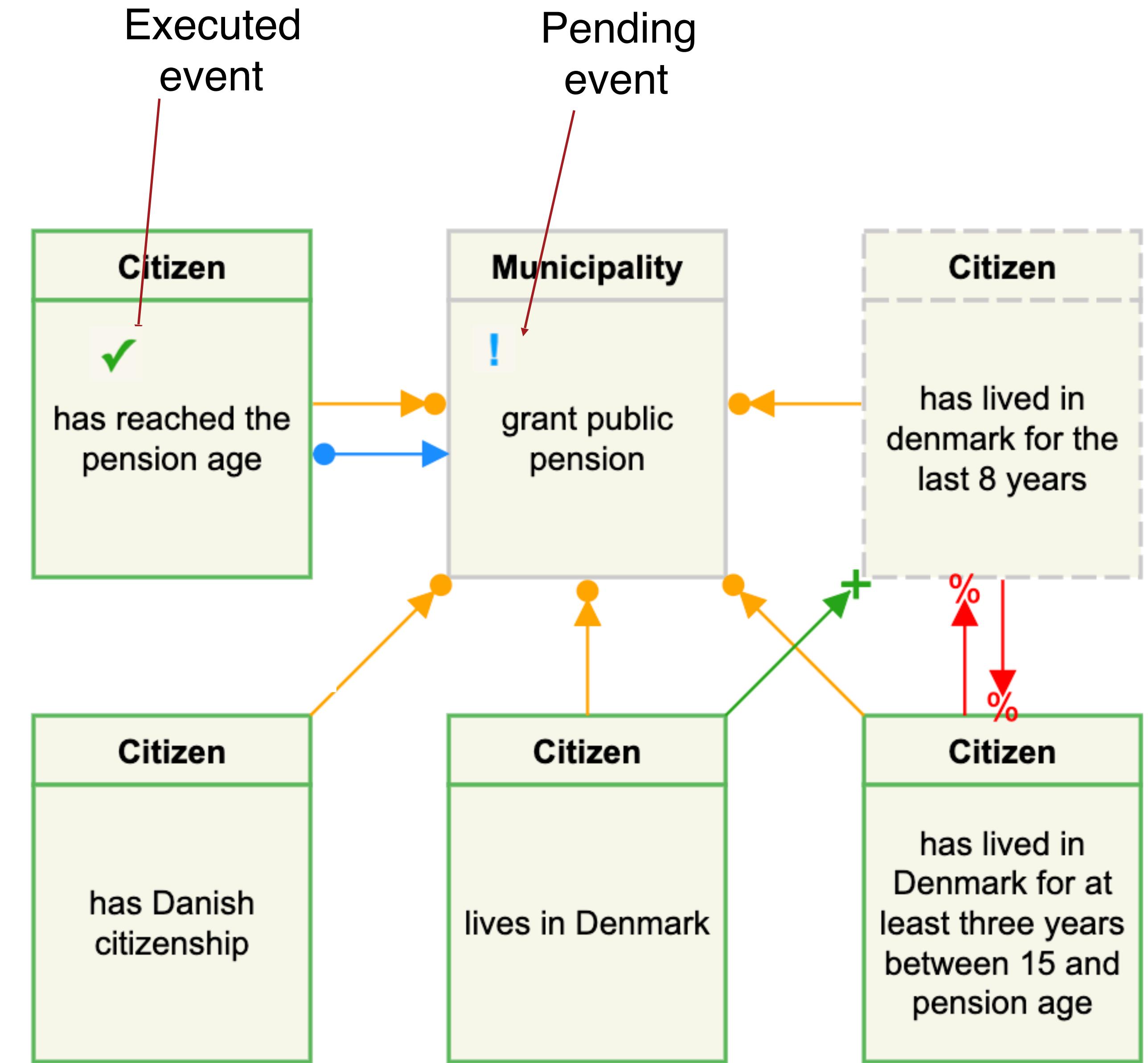
labelling

$\Phi ::= (h, i, p)$

$h ::= t \mid f$

$i ::= t \mid f$

$p ::= t \mid f$



DCR graphs - briefly

- Assume:
 - A fixed universe of events \mathcal{E} , and labels \mathcal{L}
 - $e, f \in \mathcal{E}$
 - A mapping λ from events to labels
- The grammar for DCR processes is given as:

$$P, Q ::= [M] \lambda T$$

process

$$T, U ::= e \xrightarrow{\cdot} f \quad | \quad e \xrightarrow{\bullet} f \\ | \quad e \xrightarrow{+} f \quad | \quad e \xrightarrow{\%} f \\ | \quad e \xrightarrow{\diamond} f \quad | \quad T \parallel U \\ | \quad 0$$

$$M, N ::= M, e : \Phi \mid \epsilon$$

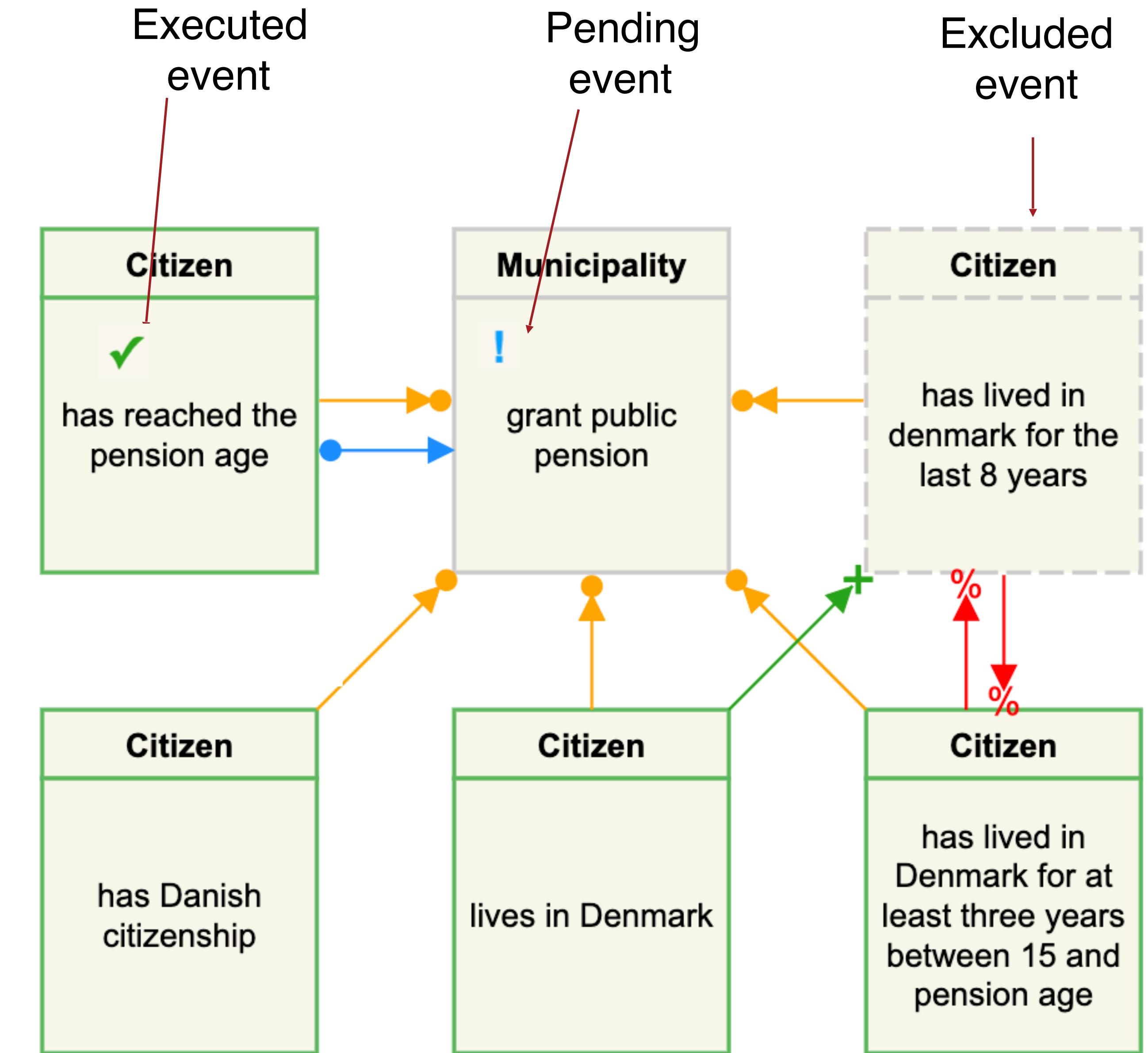
marking
labelling

$$\lambda ::= \lambda, e : l \mid \epsilon$$

$$\Phi ::= (h, i, p)$$

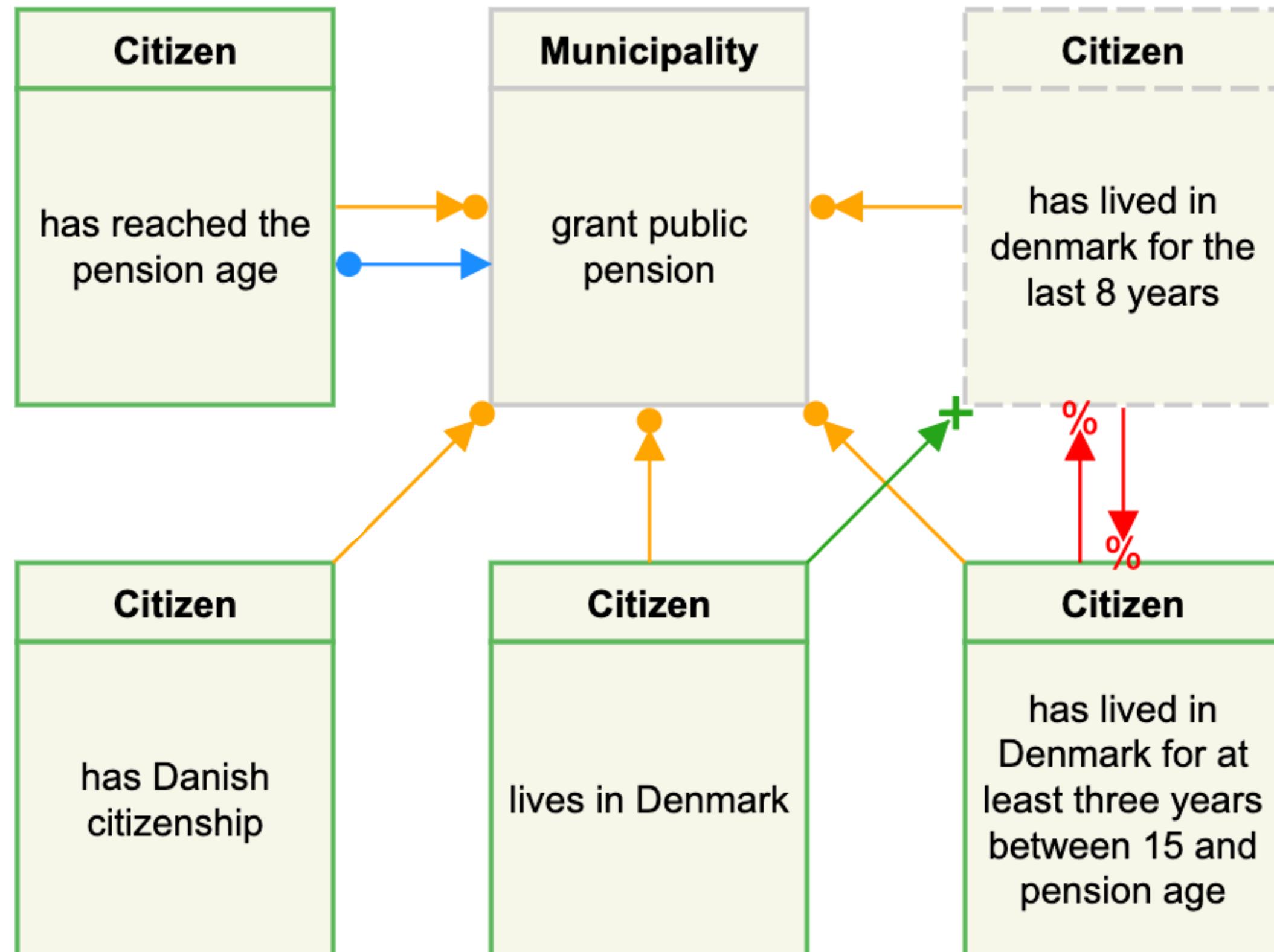
$$h ::= t \mid f$$

$$i ::= t \mid f$$

$$p ::= t \mid f$$


Operational Semantics: Enabledness and Effects

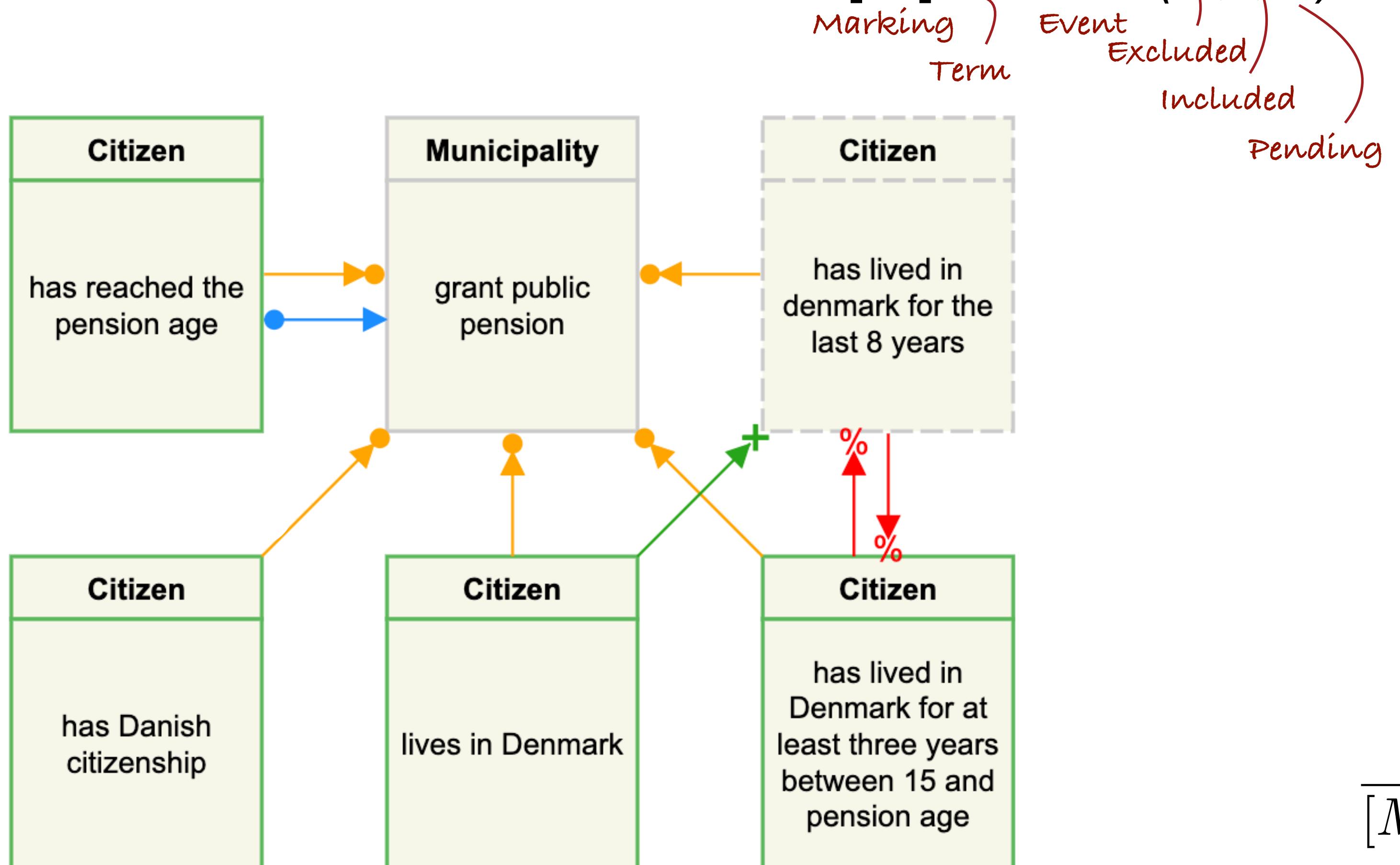
- Enabled events & effects $[M] T \vdash f : (E, I, P)$:



$$\frac{i \implies h}{[M, e: (h, i, -), f: (-, t, -)] e \rightarrow \bullet f \vdash f: \emptyset, \emptyset, \emptyset}$$

Operational Semantics: Enabledness and Effects

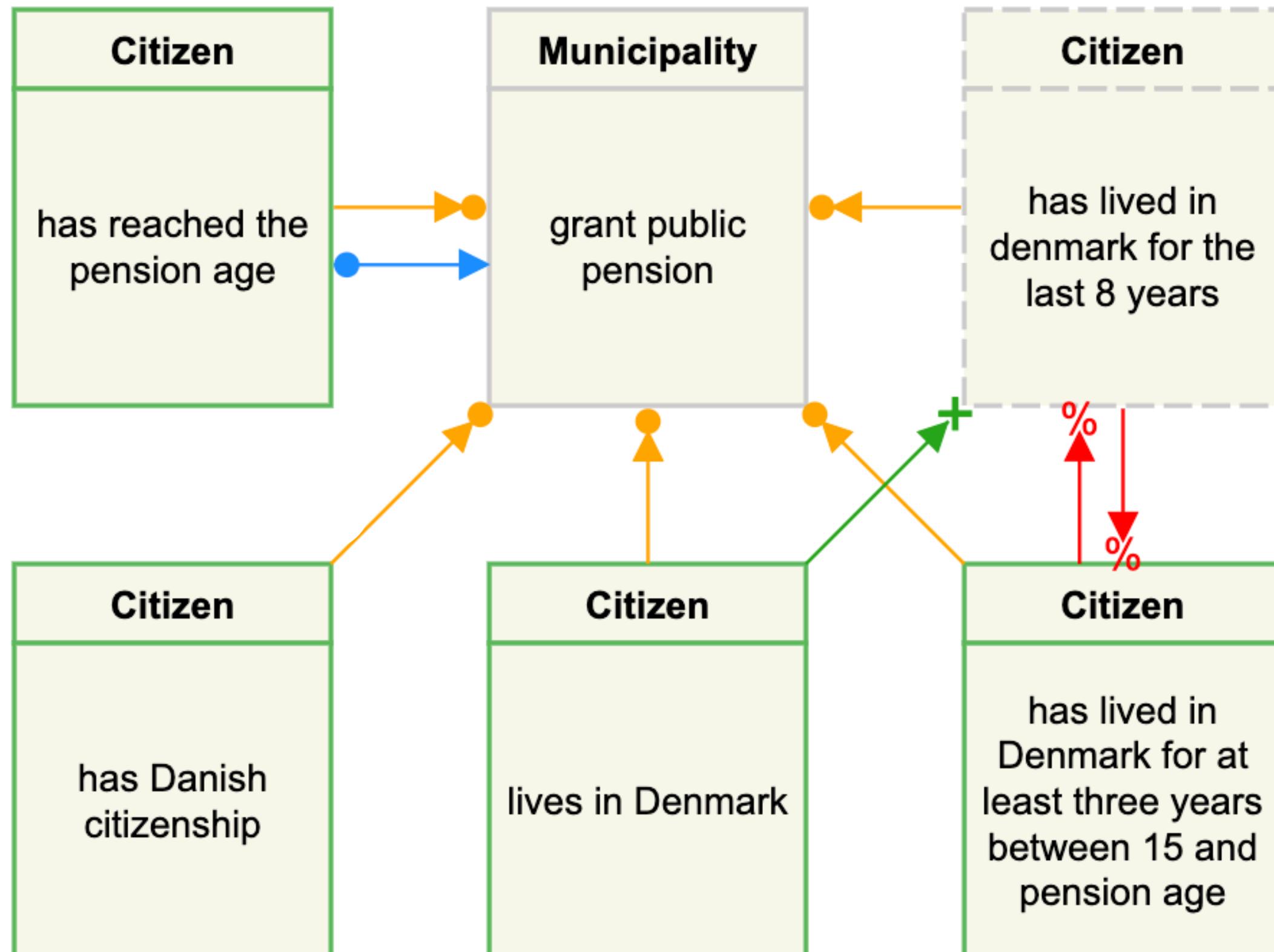
- Enabled events & effects $[M] T \vdash f : (E, I, P)$:



$$\frac{i \implies h}{[M, e: (h, i, -), f: (-, t, -)] e \rightarrow \bullet f \vdash f: \emptyset, \emptyset, \emptyset}$$

Operational Semantics: Enabledness and Effects

- Enabled events & effects $[M] T \vdash f : (E, I, P)$:

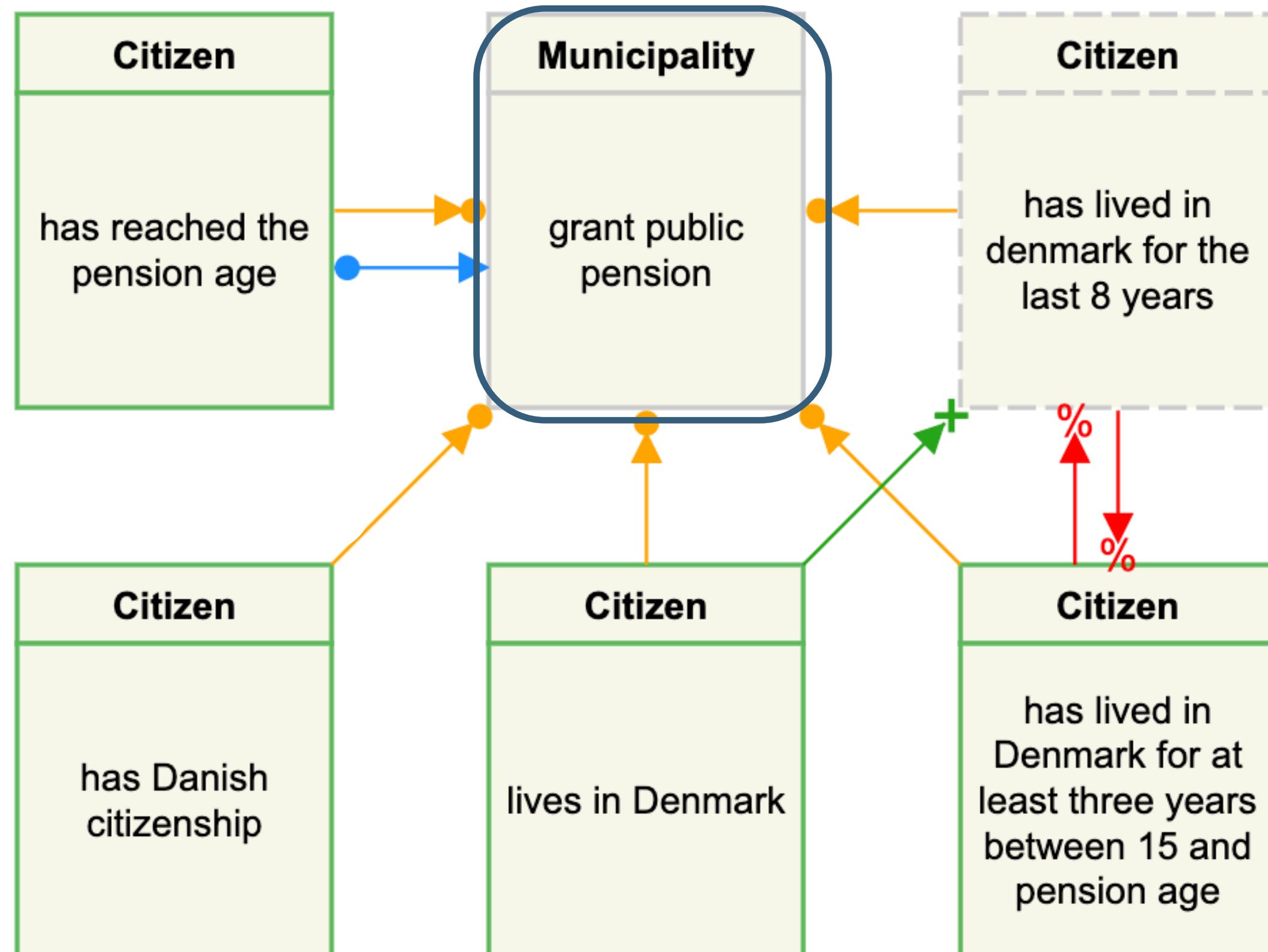


Event f is **enabled** iff

$$\frac{i \implies h}{[M, e: (h, i, -), f: (-, t, -)] e \rightarrow \bullet f \vdash f: \emptyset, \emptyset, \emptyset}$$

Operational Semantics: Enabledness and Effects

- Enabled events & effects $[M] T \vdash f : (E, I, P)$:

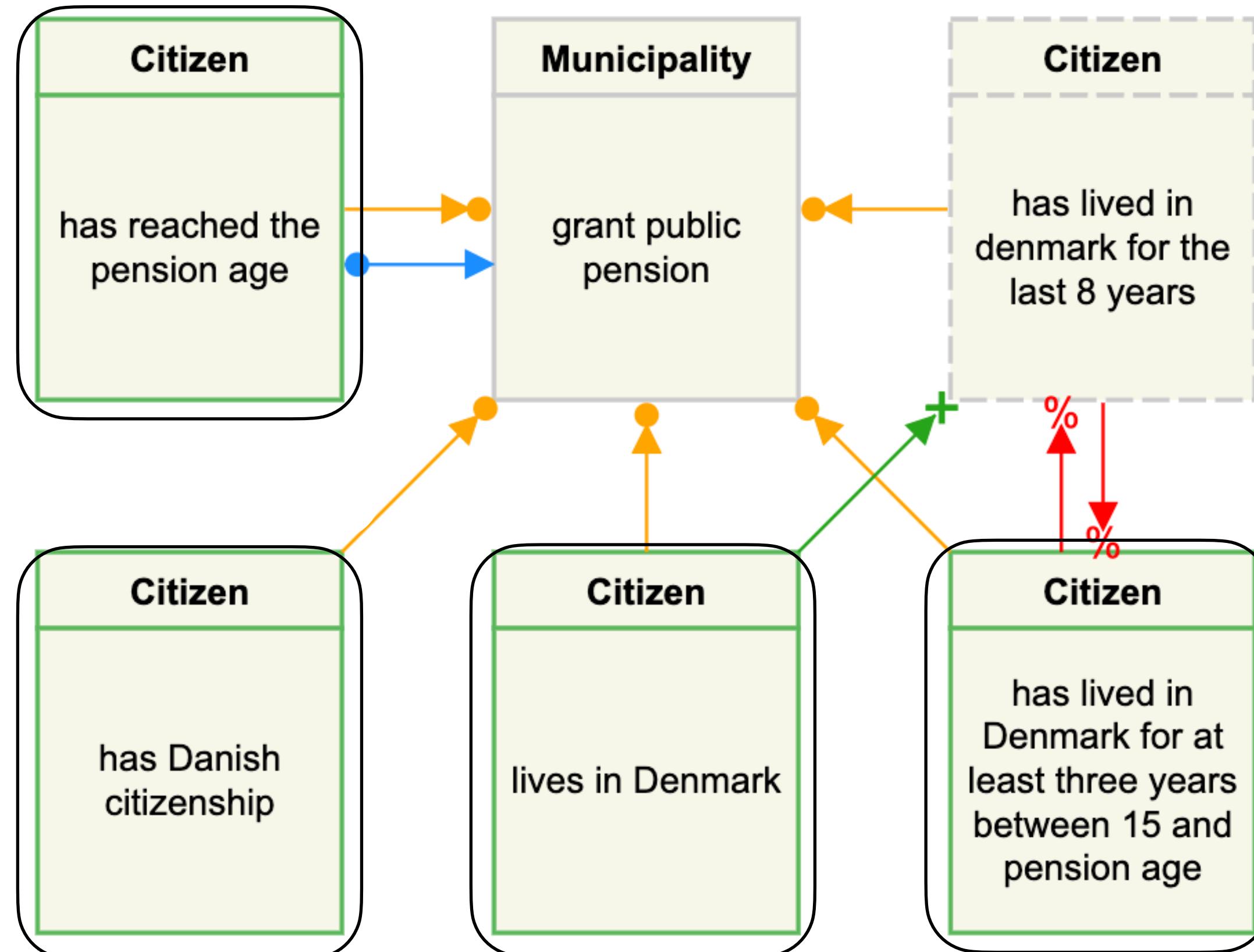


Event f is **enabled** iff
1. f is included

$$\frac{i \implies h}{[M, e: (h, i, -), f: (-, t, -)] e \rightarrow \bullet f \vdash f: \emptyset, \emptyset, \emptyset}$$

Operational Semantics: Enabledness and Effects

- Enabled events & effects $[M] T \vdash f : (E, I, P)$:



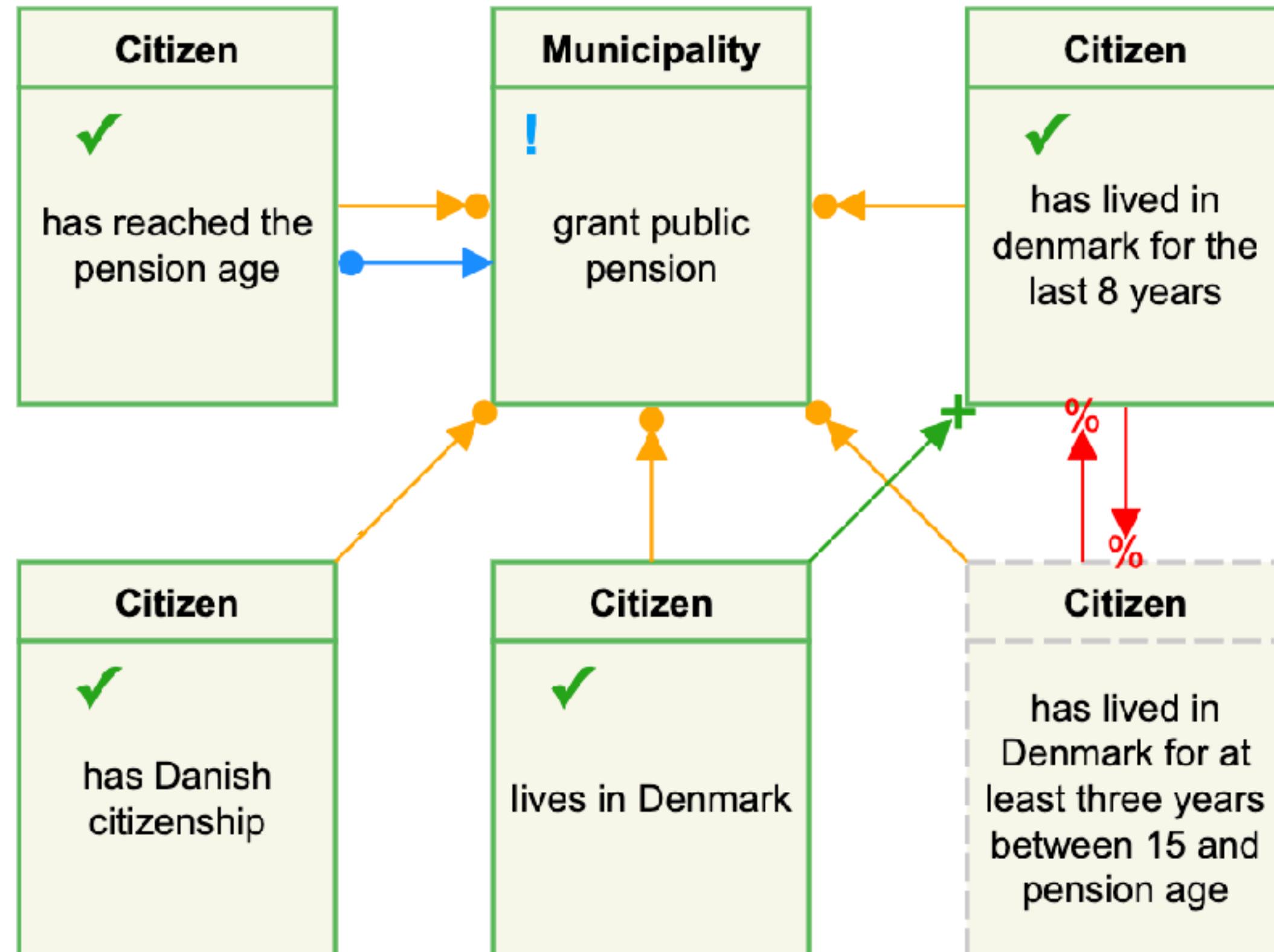
Event f is **enabled** iff

- f is included
- Its included preconditions have been executed or excluded

$$\frac{i \implies h}{[M, e: (h, i, -), f: (-, t, -)] e \rightarrow \bullet f \vdash f: \emptyset, \emptyset, \emptyset}$$

Operational Semantics: Enabledness and Effects

- Enabled events & effects $[M] T \vdash f : (E, I, P)$:



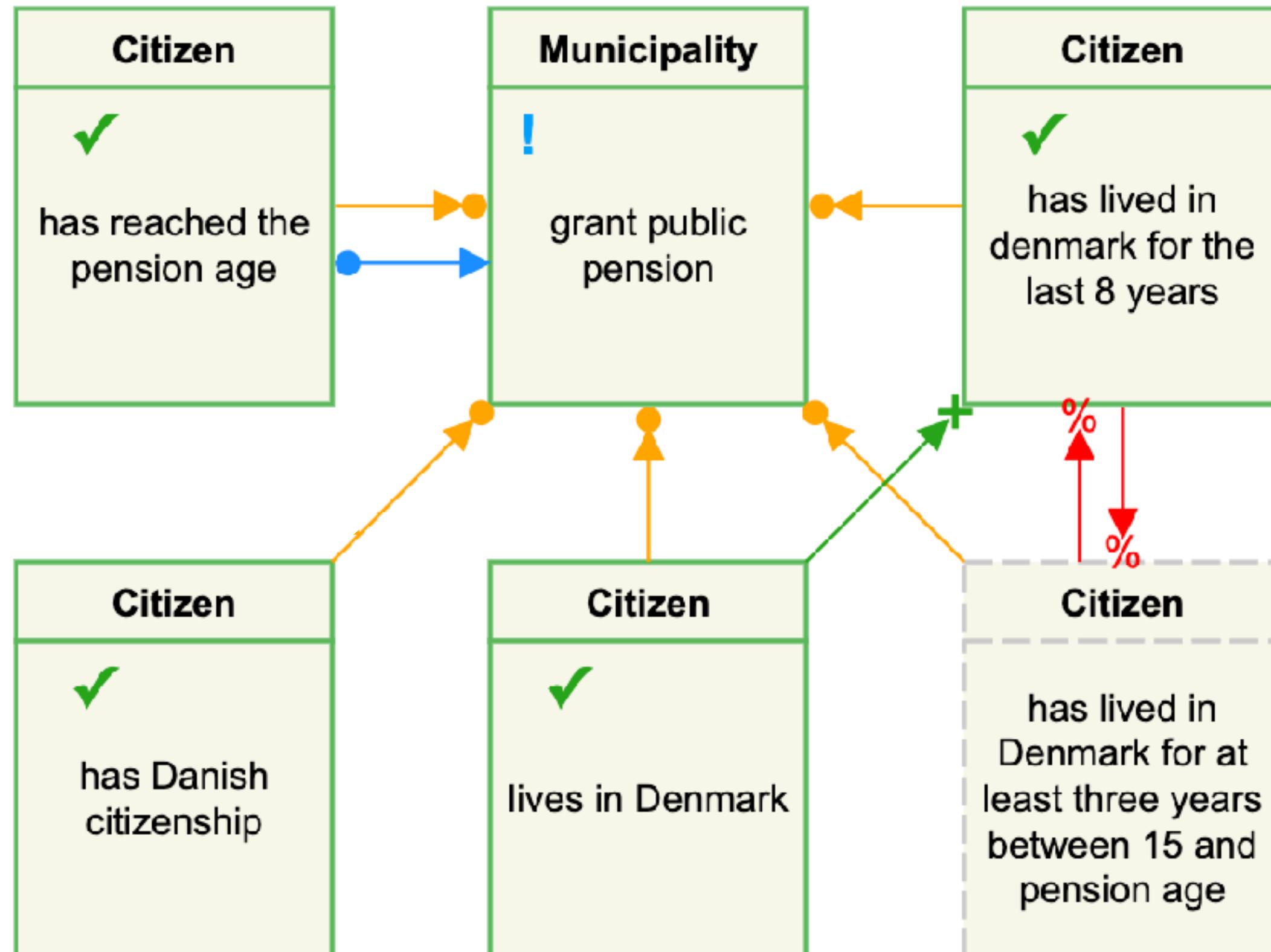
Event f is **enabled** iff

- f is included
- Its included preconditions have been executed or excluded

$$\frac{i \implies h}{[M, e: (h, i, -), f: (-, t, -)] e \rightarrow \bullet f \vdash f: \emptyset, \emptyset, \emptyset}$$

Operational Semantics: Enabledness and Effects

- Enabled events & effects $[M] T \vdash f : (E, I, P)$:



Event f is **enabled** iff

- f is included
- Its included preconditions have been executed or excluded

$$\frac{i \implies h}{[M, e : (h, i, -), f : (-, t, -)] e \xrightarrow{\bullet} f \vdash f : \emptyset, \emptyset, \emptyset}$$

$$[M, e : (-, t, -)] e \xrightarrow{\bullet} f \vdash e : (\emptyset, \emptyset, \{f\})$$

imposed obligations

Classic DCR: Operational Semantics

$$\frac{[M] T \vdash e : \delta}{[M] T \xrightarrow{e:\delta} T} \quad [\text{INTRO}]$$

- LTS keeps track of the sequence of fired events
- $e : \delta = (Ex, In, Pe)$: effects of executing e
- $\delta_1 \oplus \delta_2$: merge of those effects that agree on their overlap
- $e : \delta.M$: effect of executing e in marking M

Classic DCR: Operational Semantics

$$\frac{[M] T \vdash e : \delta}{[M] T \xrightarrow{e:\delta} T} \quad [\text{INTRO}]$$

$$\frac{[M] T_1 \xrightarrow{e:\delta_1} T'_1 \quad [M] T_2 \xrightarrow{e:\delta_2} T'_2}{[M] T_1 \parallel T_2 \xrightarrow{e:\delta_1 \oplus \delta_2} T'_1 \parallel T'_2} \quad [\text{PAR}]$$

- LTS keeps track of the sequence of fired events
- $e : \delta = (Ex, In, Pe)$: effects of executing e
- $\delta_1 \oplus \delta_2$: merge of those effects that agree on their overlap
- $e : \delta.M$: effect of executing e in marking M

Classic DCR: Operational Semantics

$$\frac{[M] T \vdash e : \delta}{[M] T \xrightarrow{e:\delta} T} \quad [\text{INTRO}]$$

$$\frac{[M] T_1 \xrightarrow{e:\delta_1} T'_1 \quad [M] T_2 \xrightarrow{e:\delta_2} T'_2}{[M] T_1 \parallel T_2 \xrightarrow{e:\delta_1 \oplus \delta_2} T'_1 \parallel T'_2} \quad [\text{PAR}]$$

$$\frac{[M] T \xrightarrow{e:\delta} T'}{[M] T \xrightarrow{e} [e : \delta \cdot M] T'} \quad [\text{EFFECT}]$$

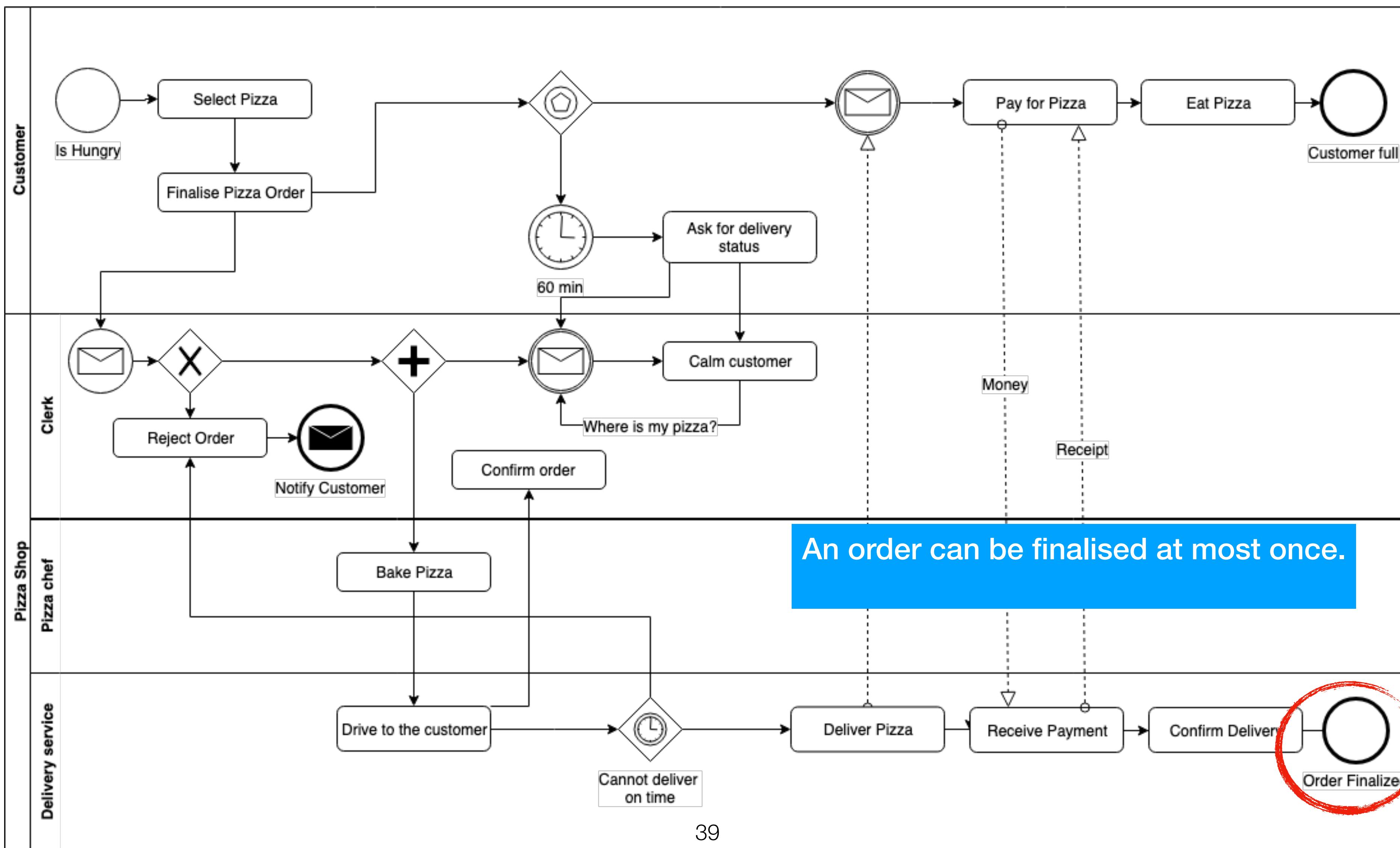
- LTS keeps track of the sequence of fired events
- $e : \delta = (Ex, In, Pe)$: effects of executing e
- $\delta_1 \oplus \delta_2$: merge of those effects that agree on their overlap
- $e : \delta.M$: effect of executing e in marking M

(Recall) the order placement process

- An order can be finalised at most once.
- Once an order is finalised, it must be confirmed or rejected.
- A confirmed order may be rejected later on (due to “late” problems), but not vice-versa: a reject cannot be reverted.
- An order can be confirmed only if it shipped before.
- An order may be rejected autonomously by the seller. However, if the warehouse notifies the seller of a shipment issue, then the seller has to reject the order.
- The warehouse decision is firm: “Ship” and “Notify shipment issue” are mutually exclusive.

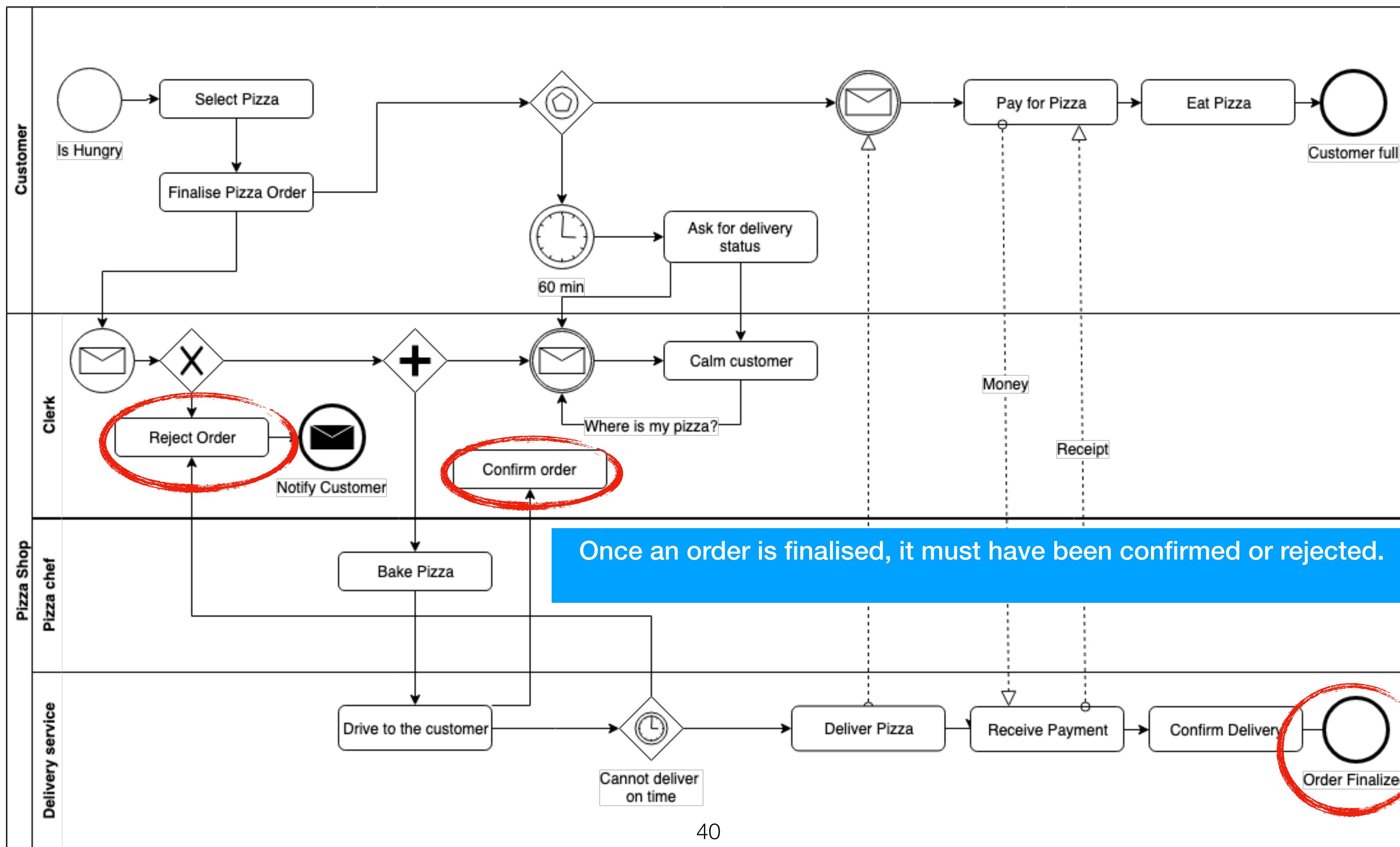
The pizza delivery process

In BPMN



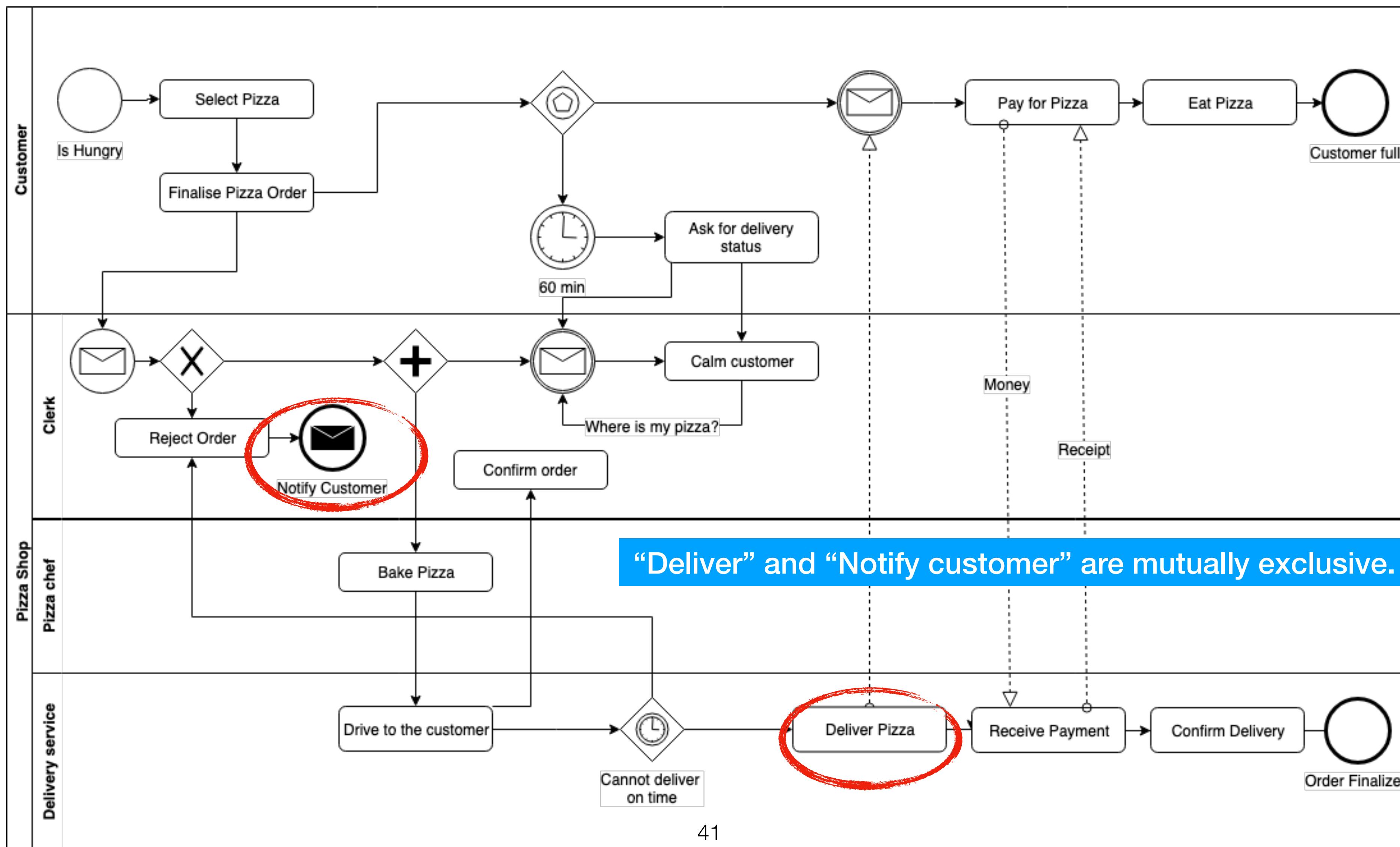
The pizza delivery process

In BPMN



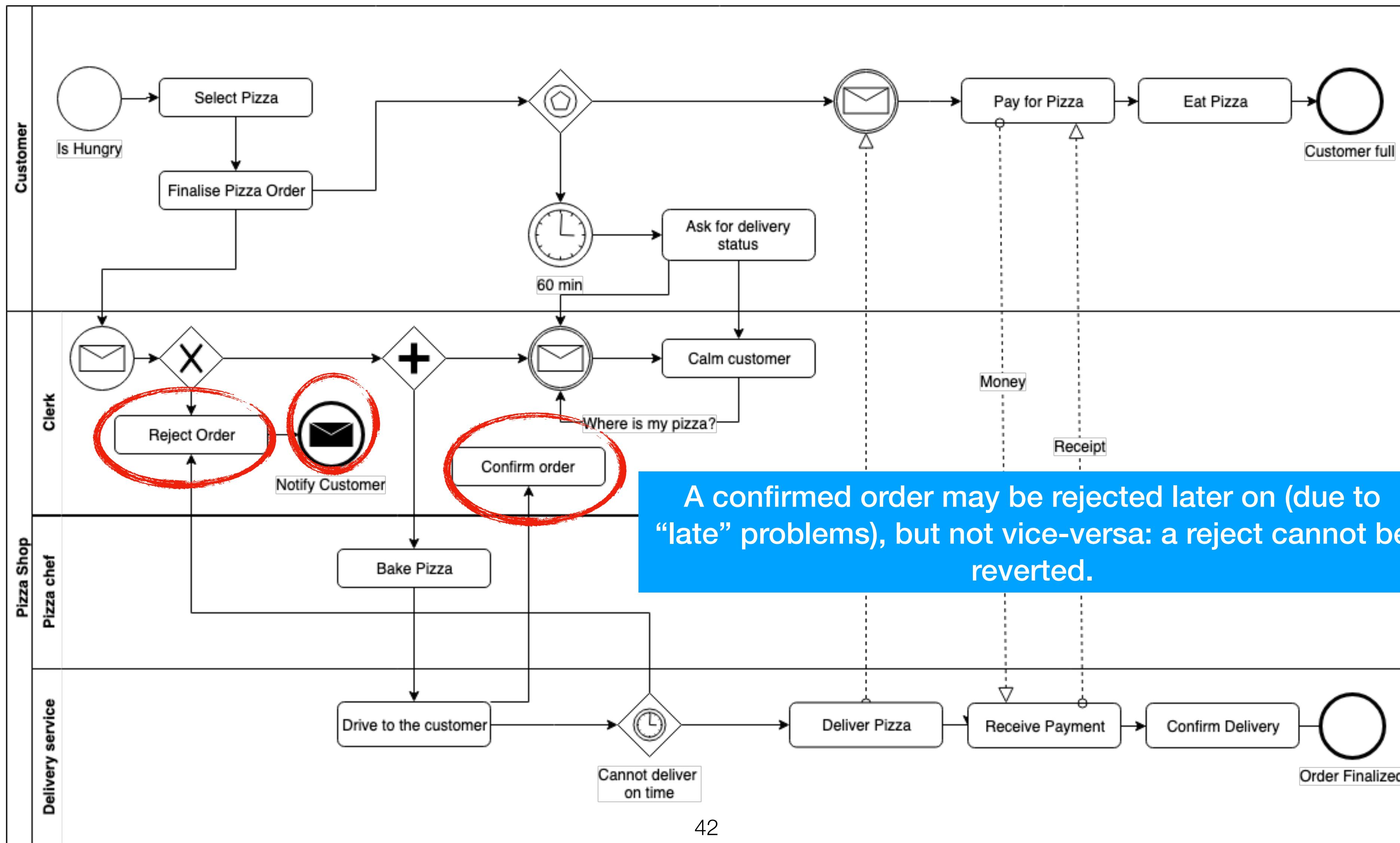
The pizza delivery process

In BPMN



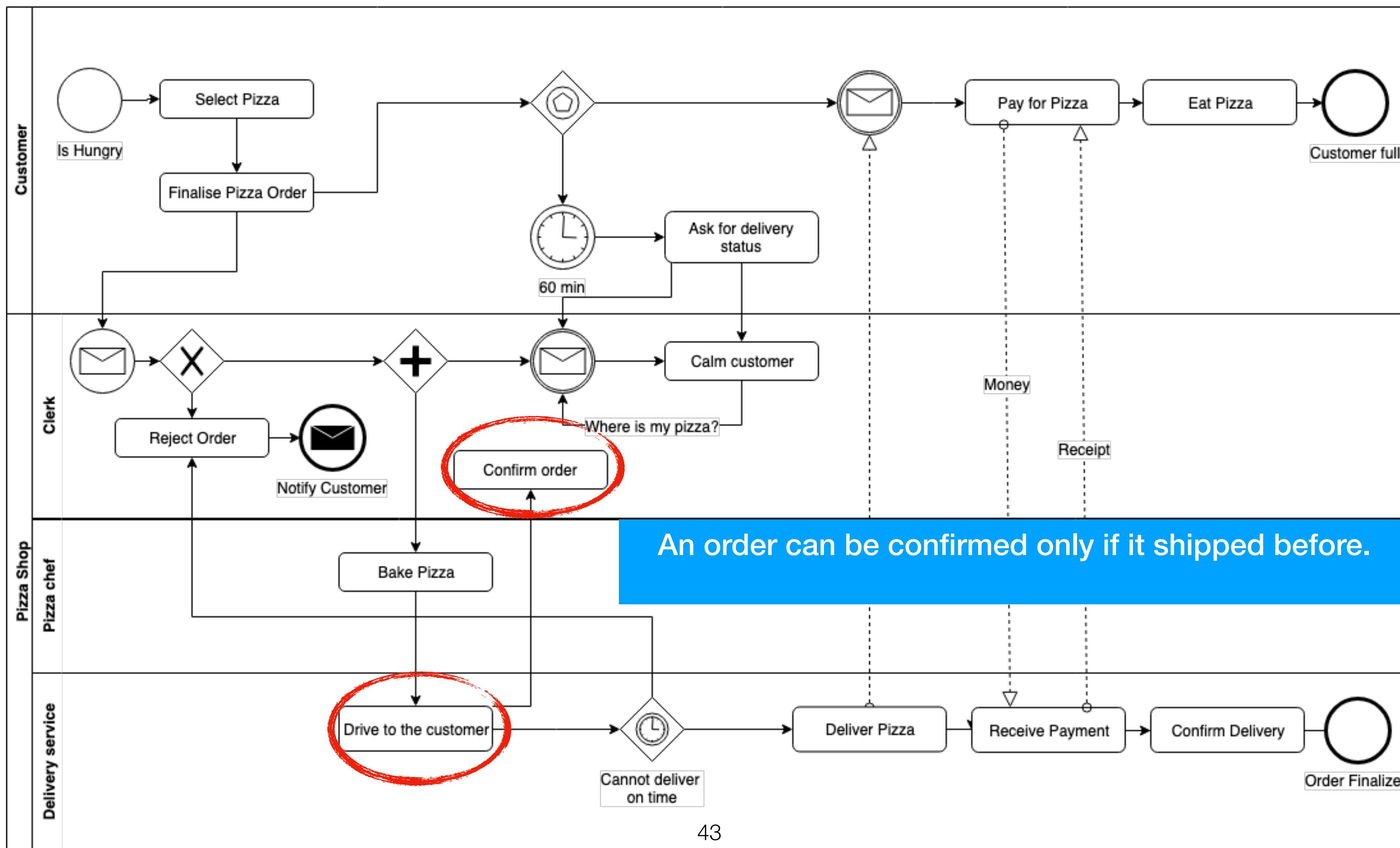
The pizza delivery process

In BPMN



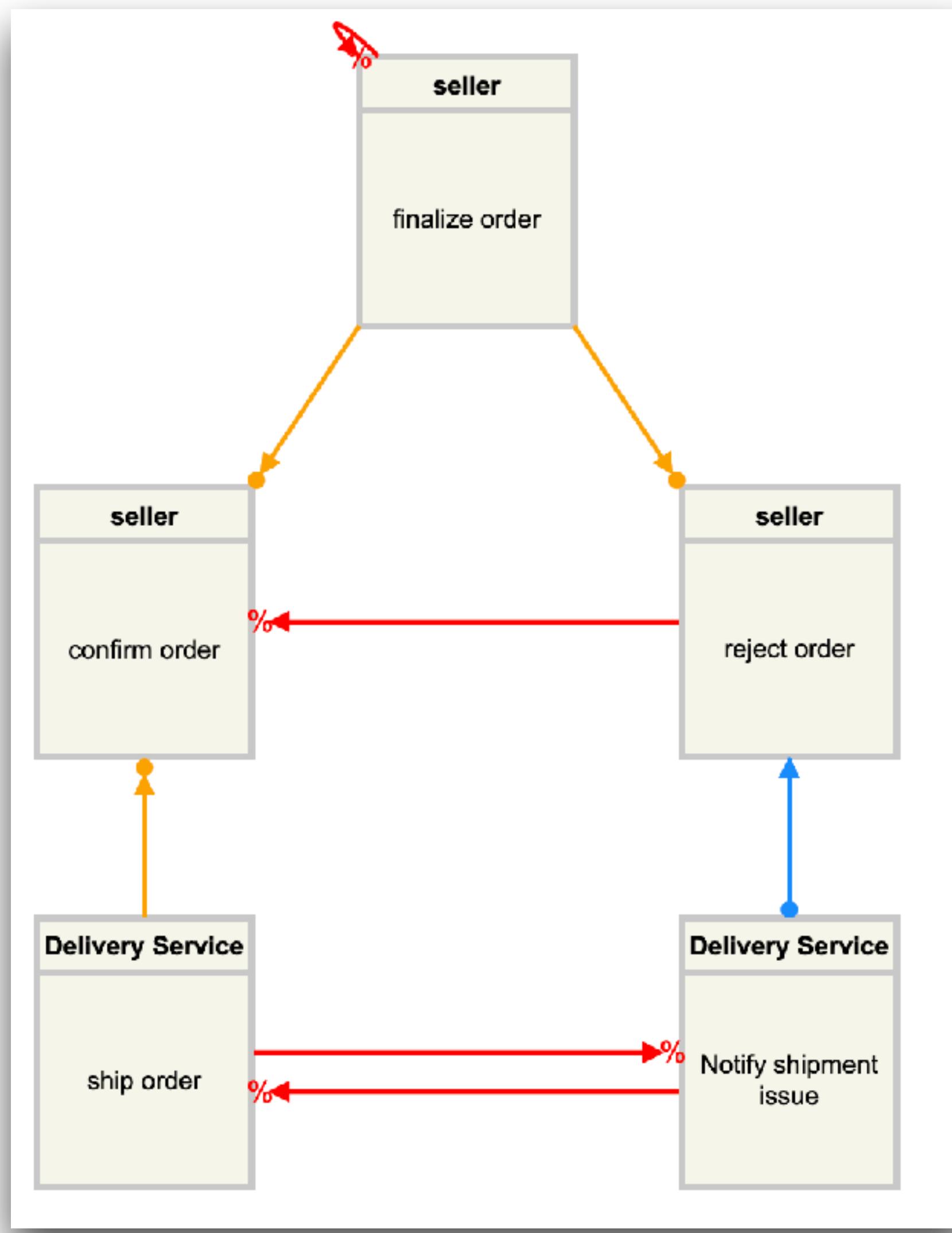
The pizza delivery process

In BPMN



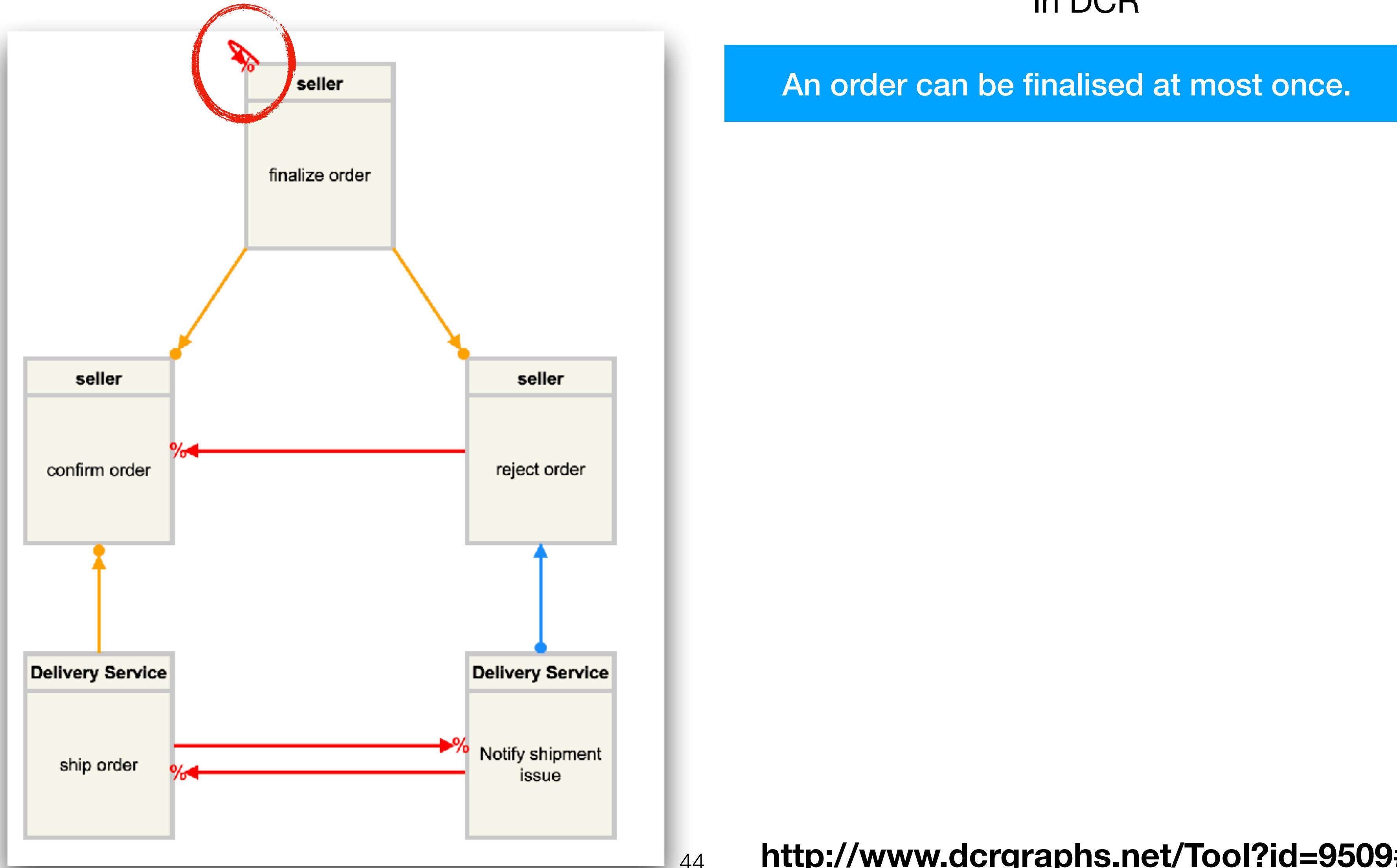
The delivery process

In DCR



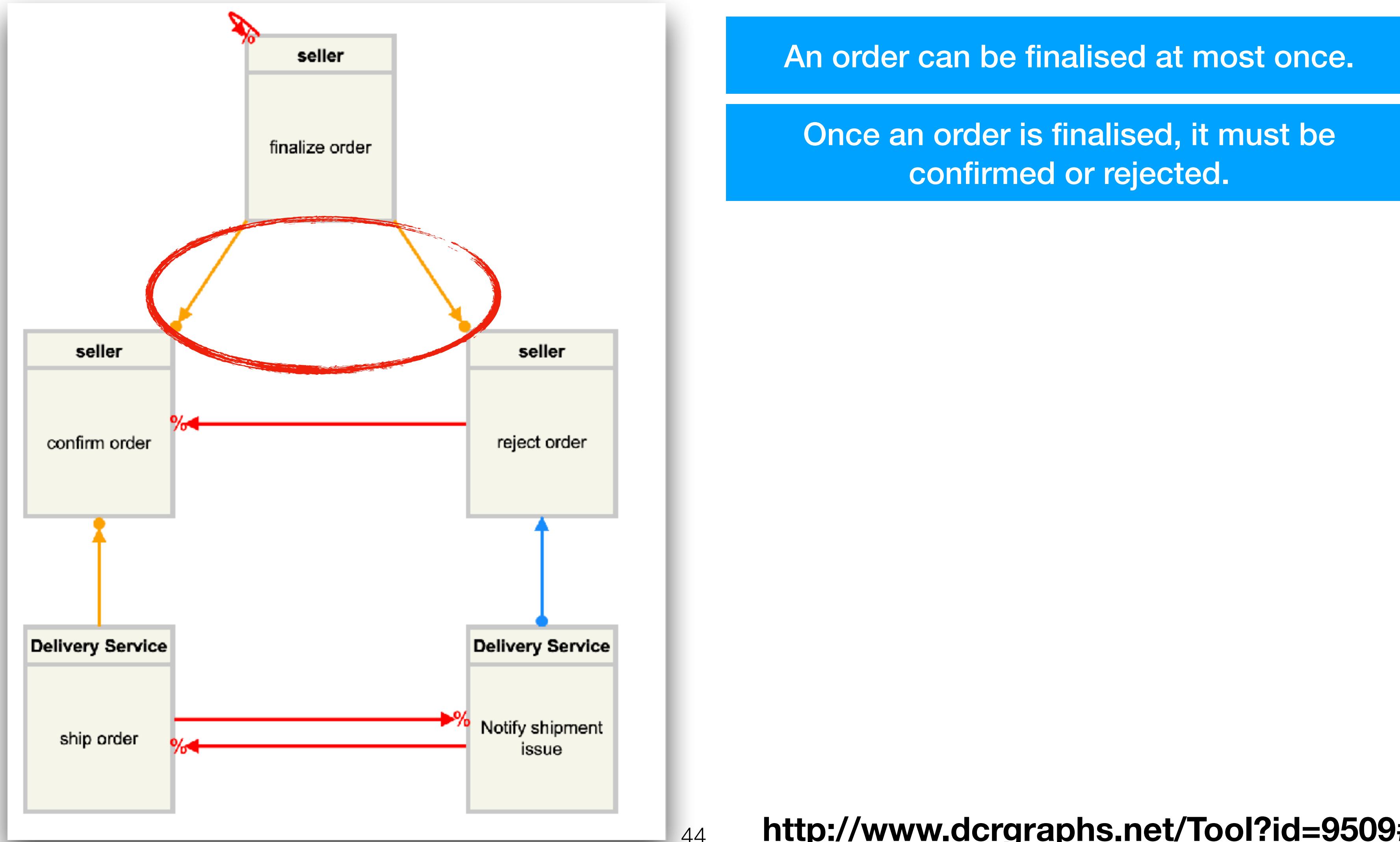
The delivery process

In DCR



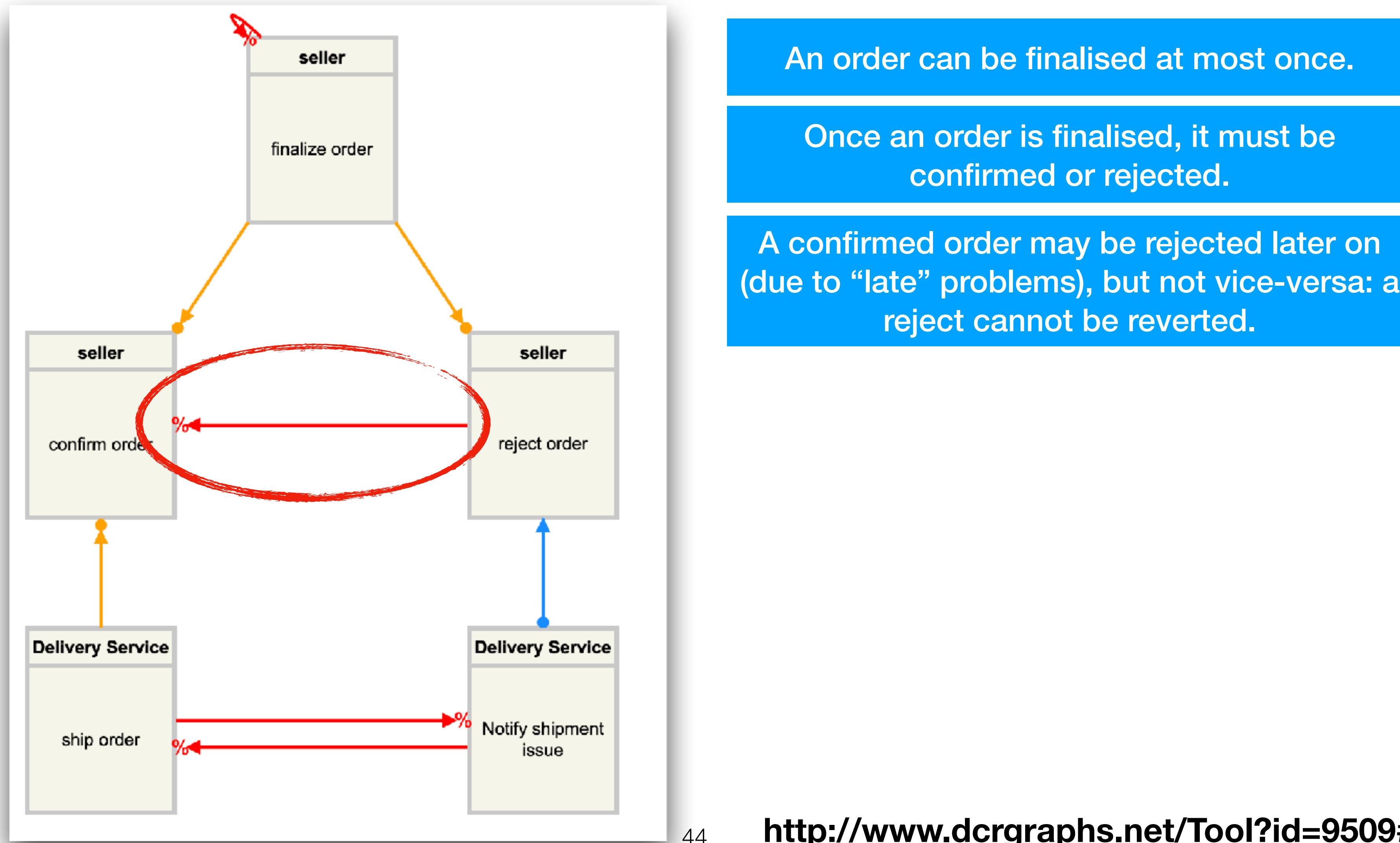
The delivery process

In DCR



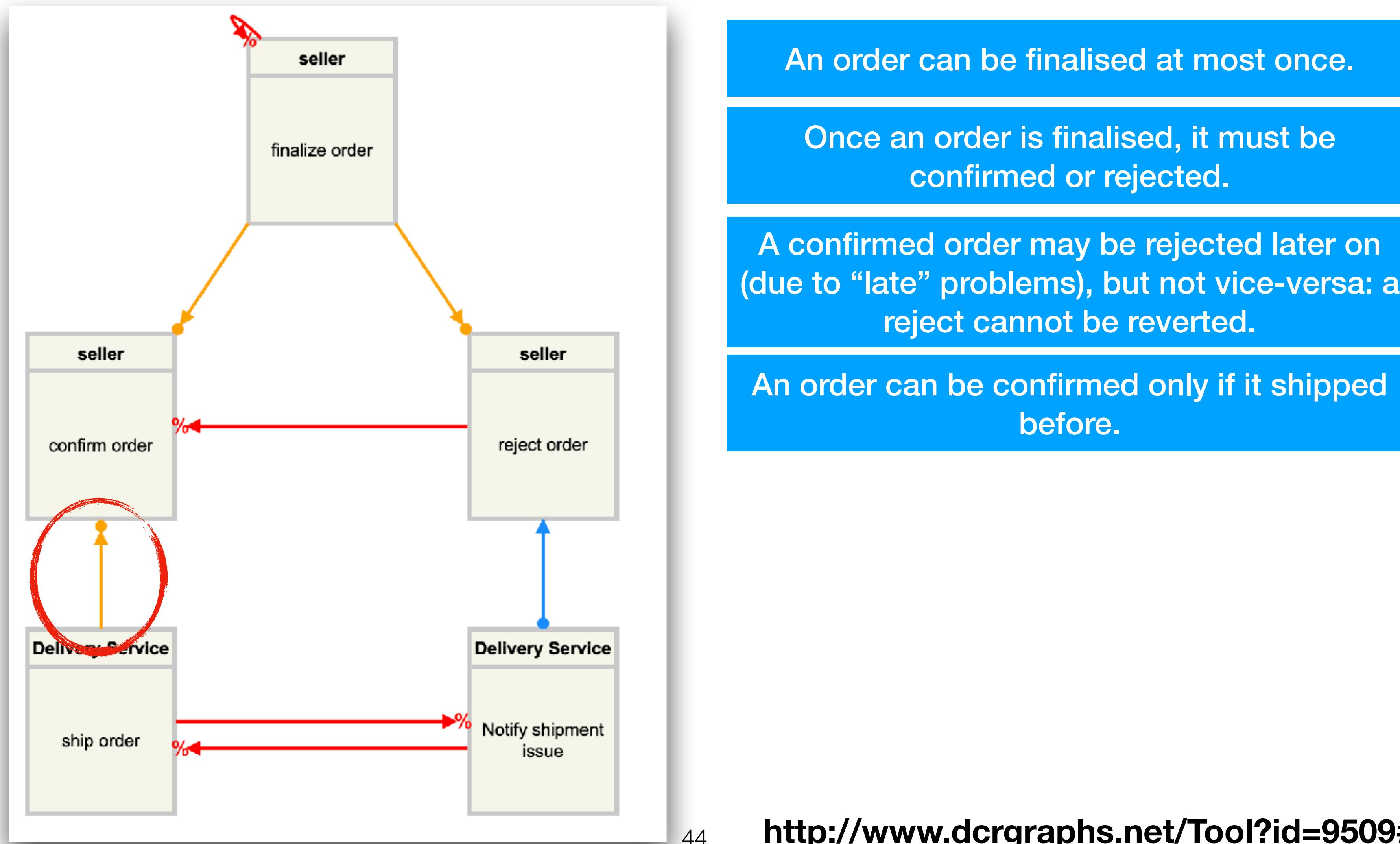
The delivery process

In DCR



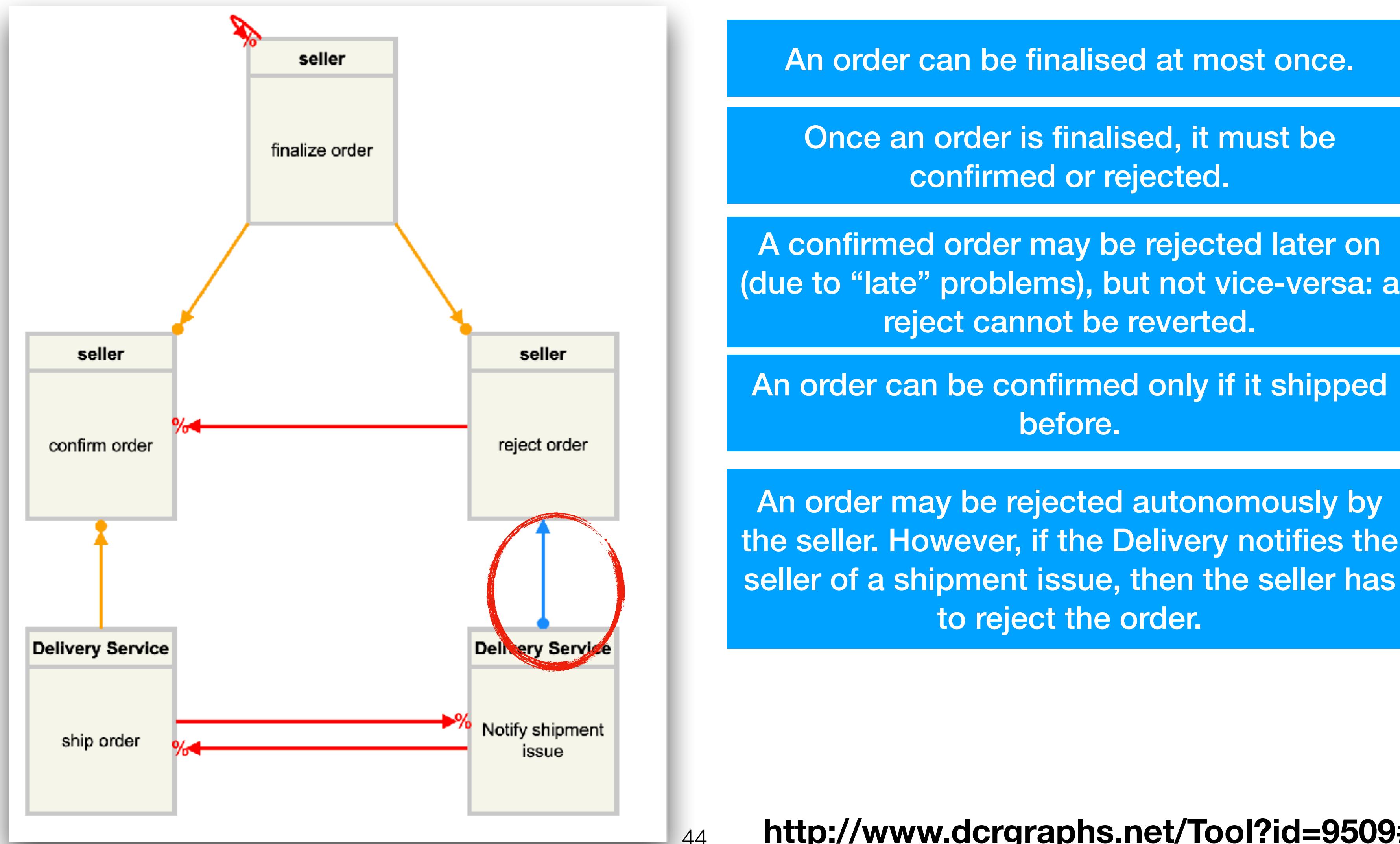
The delivery process

In DCR



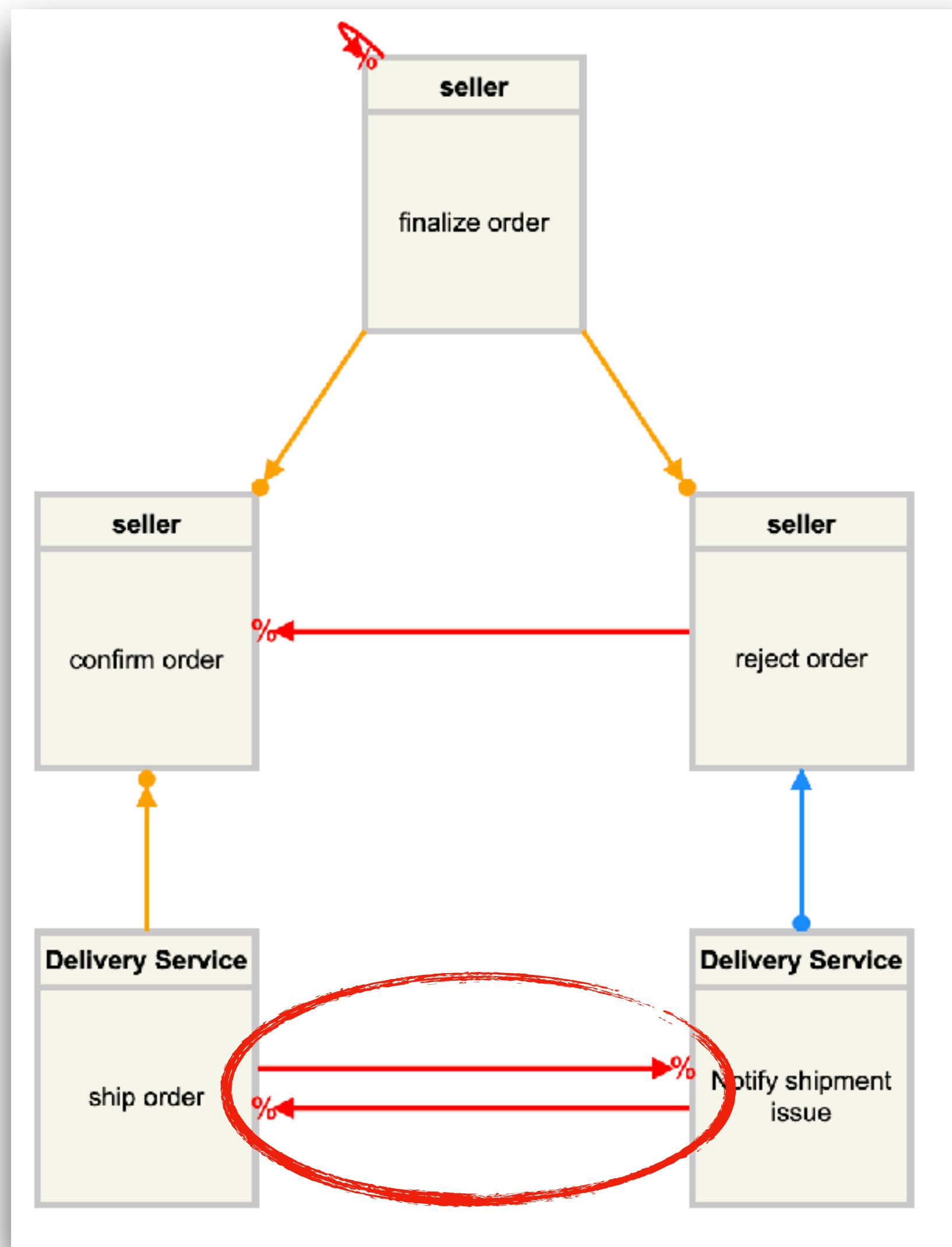
The delivery process

In DCR



The delivery process

In DCR



An order can be finalised at most once.

Once an order is finalised, it must be confirmed or rejected.

A confirmed order may be rejected later on (due to “late” problems), but not vice-versa: a reject cannot be reverted.

An order can be confirmed only if it shipped before.

An order may be rejected autonomously by the seller. However, if the Delivery notifies the seller of a shipment issue, then the seller has to reject the order.

The Delivery decision is firm: “Ship” and “Notify shipment issue” are mutually exclusive.

Soundness of declarative models

Hugo A. López

hulo@dtu.dk

DTU Compute

Department of Applied Mathematics and Computer Science

$$f(x+\Delta x) = \sum_{i=0}^{\infty} \frac{(\Delta x)^i}{i!} f^{(i)}(x)$$
$$\Theta^{\sqrt{17}} + \Omega \int_a^b \delta / e^{i\pi} =$$
$$\epsilon \in \{2.7182818284, \dots\}$$
$$\infty \rightarrow \Sigma \gg ,$$
$$\chi^2 !$$

Why would we make a model of a process?

Why would we make a model of a process?

- To understand the process

Why would we make a model of a process?

- To understand the process
- For communication (say, between managers)

Why would we make a model of a process?

- To understand the process
- For communication (say, between managers)
- For process optimisation

Why would we make a model of a process?

- To understand the process
- For communication (say, between managers)
- For process optimisation
- For education

Why would we make a model of a process?

- To understand the process
- For communication (say, between managers)
- For process optimisation
- For education
- For requirements specification

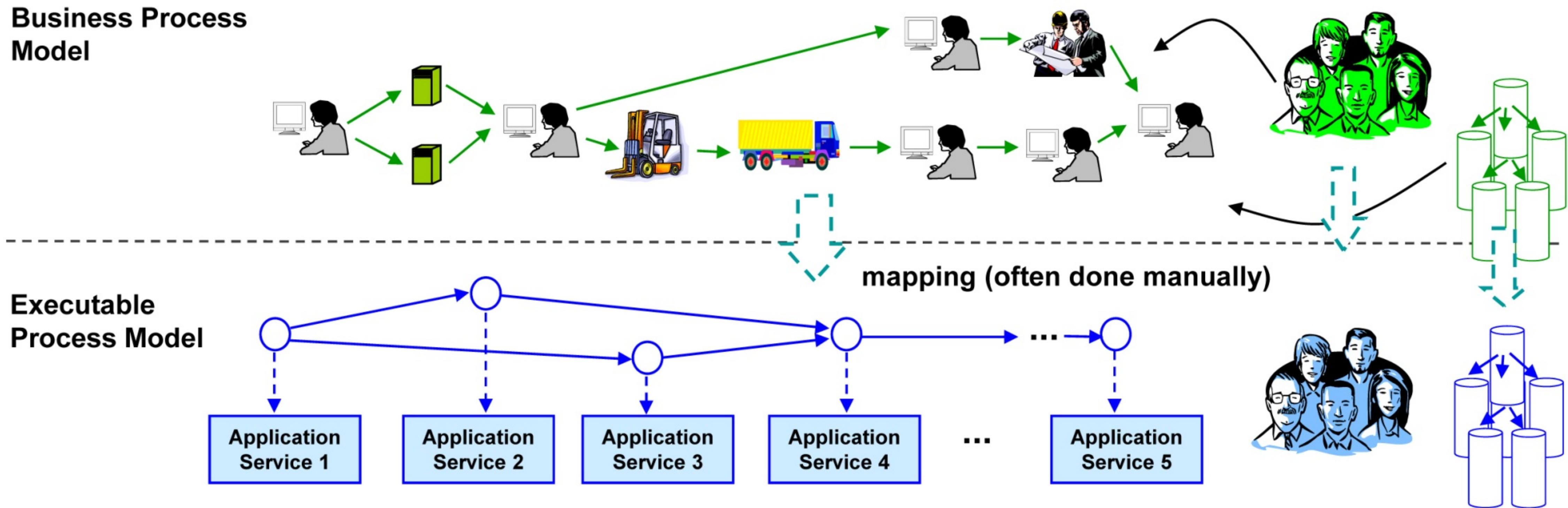
Why would we make a model of a process?

- To understand the process
- For communication (say, between managers)
- For process optimisation
- For education
- For requirements specification
- For executing outright

When is your business
process (model) correct?

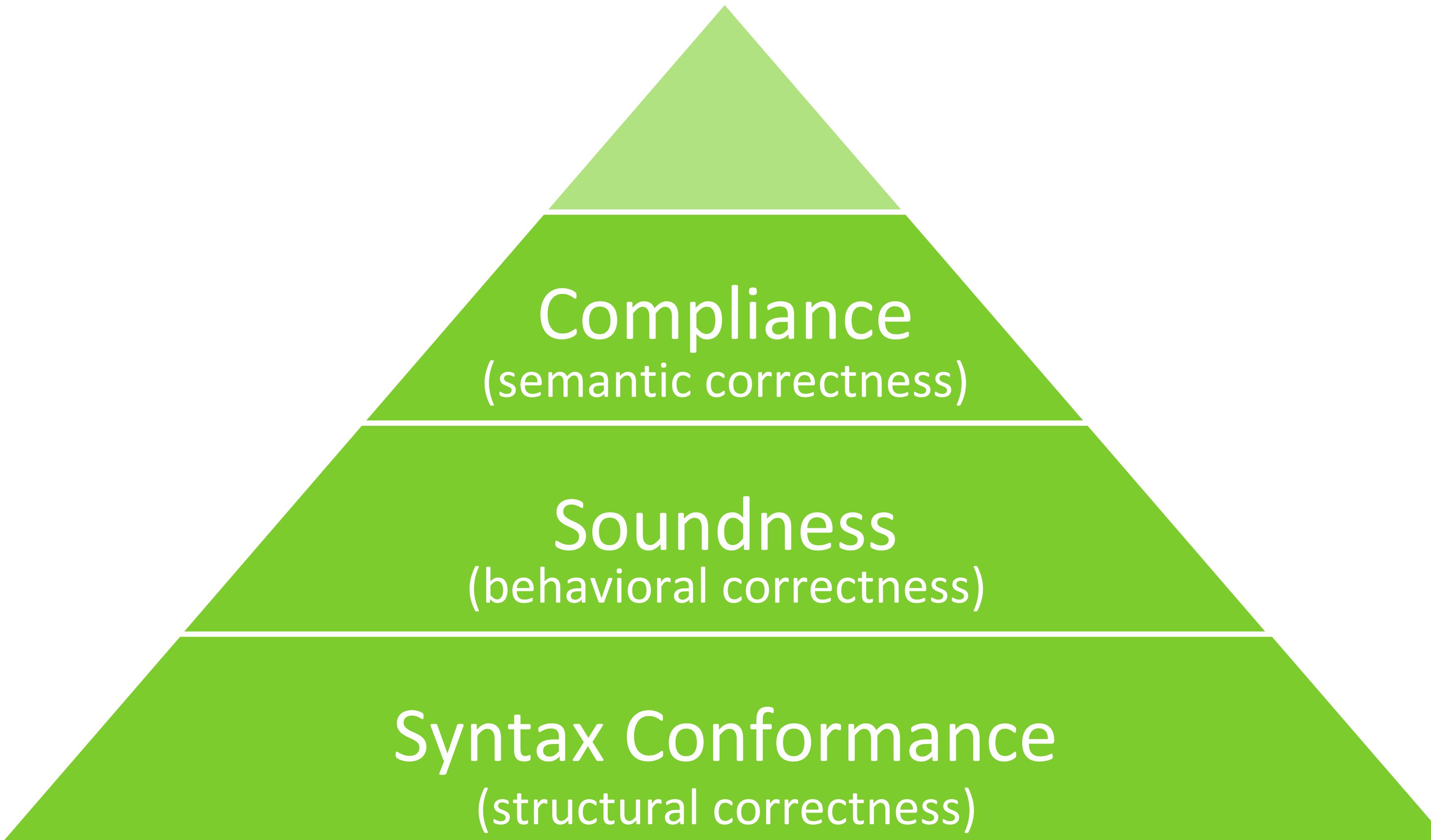
In mentimeter!

“For executing it outright”



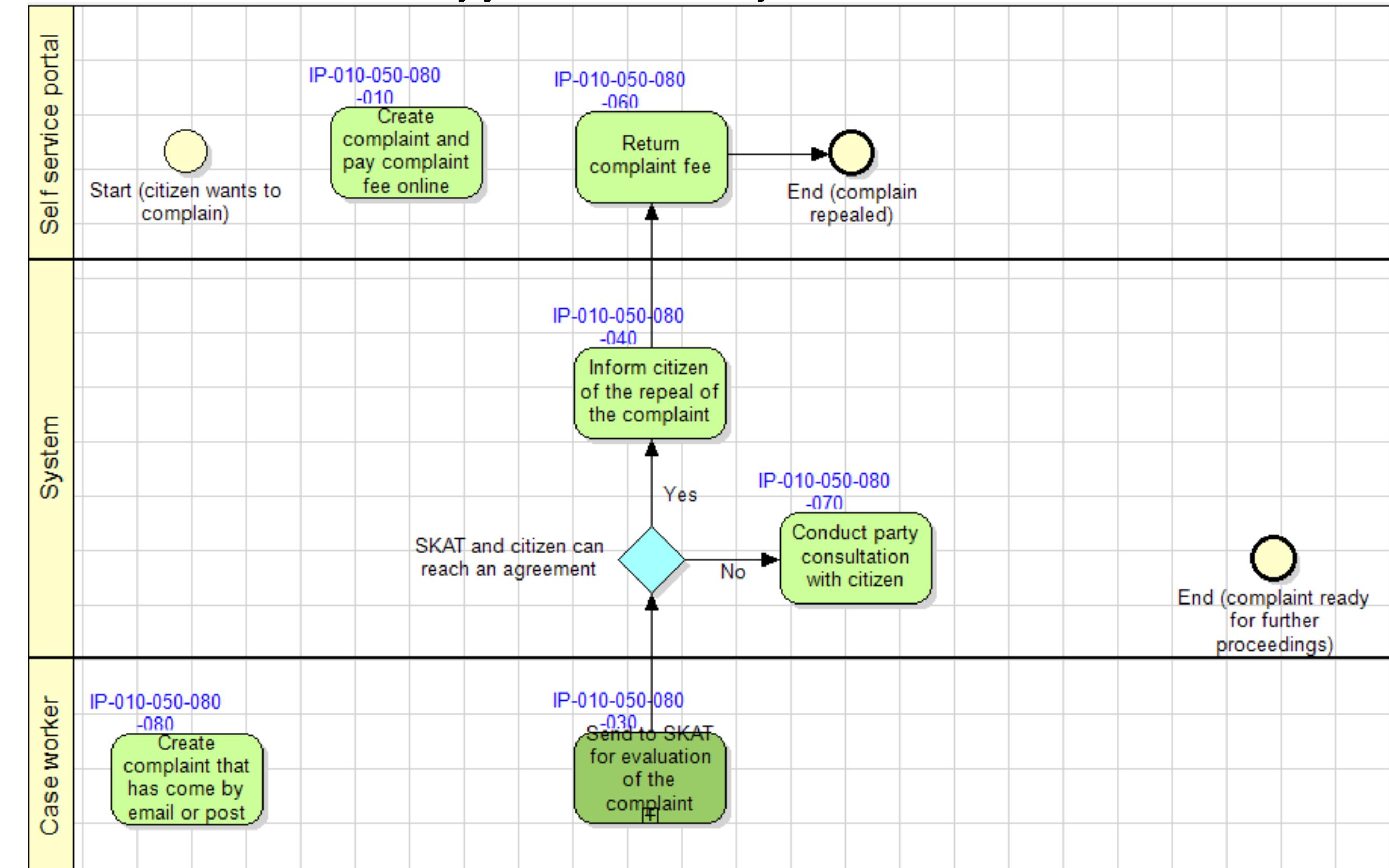
- should be preceded by process re-engineering and optimization efforts
[Hammer & Champy, 1993 & Reijers 2003]

CORRECTNESS IN BUSINESS PROCESSES



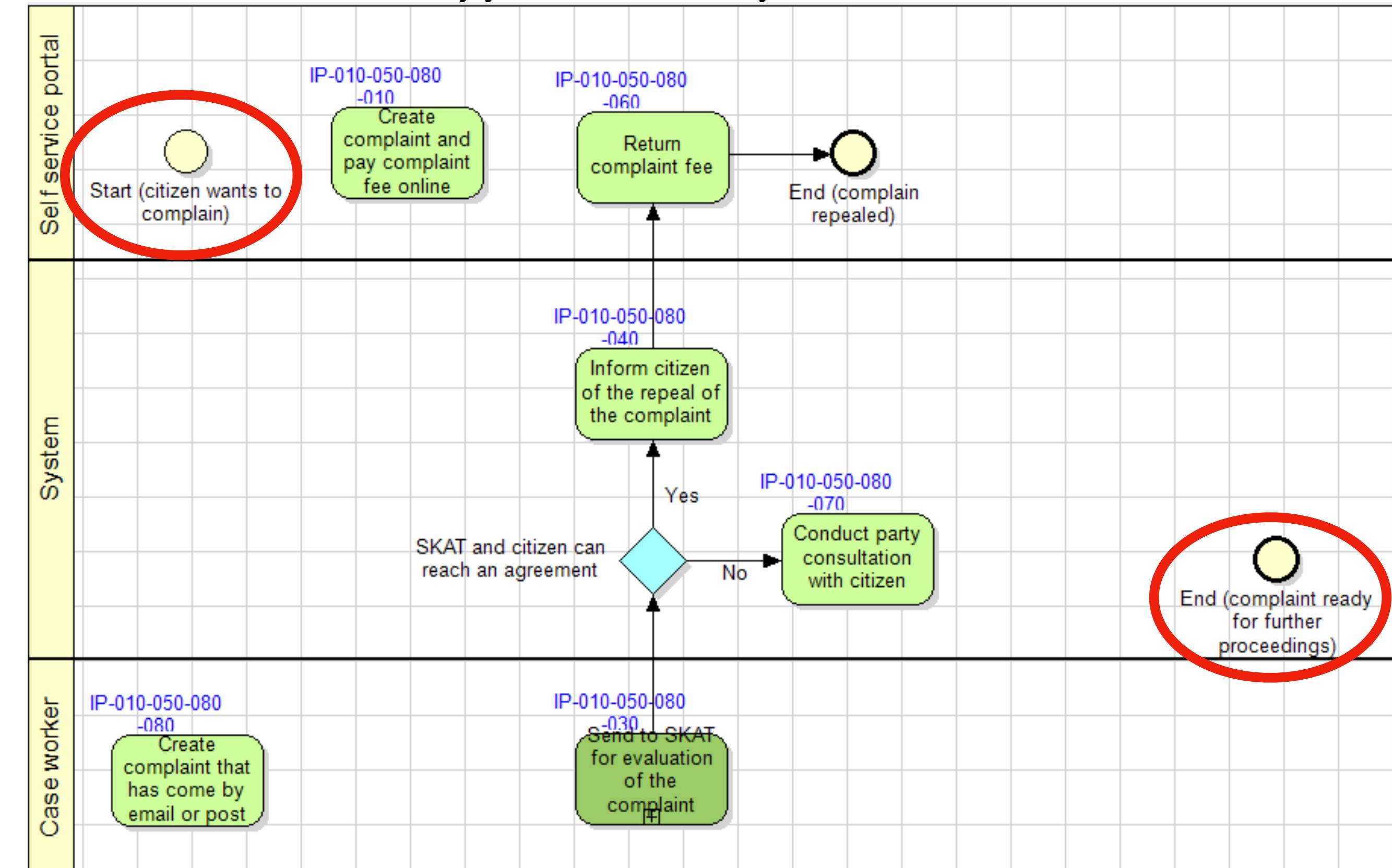
Syntax conformance

- Focus on the proper use of language abstractions
- Key: does it “reads well”? Syntax
- Typically controlled via modelling tool/compiler
- Common errors:
 - Unconnected relations
 - Missing variables
 - Wrong timestamps, etc.



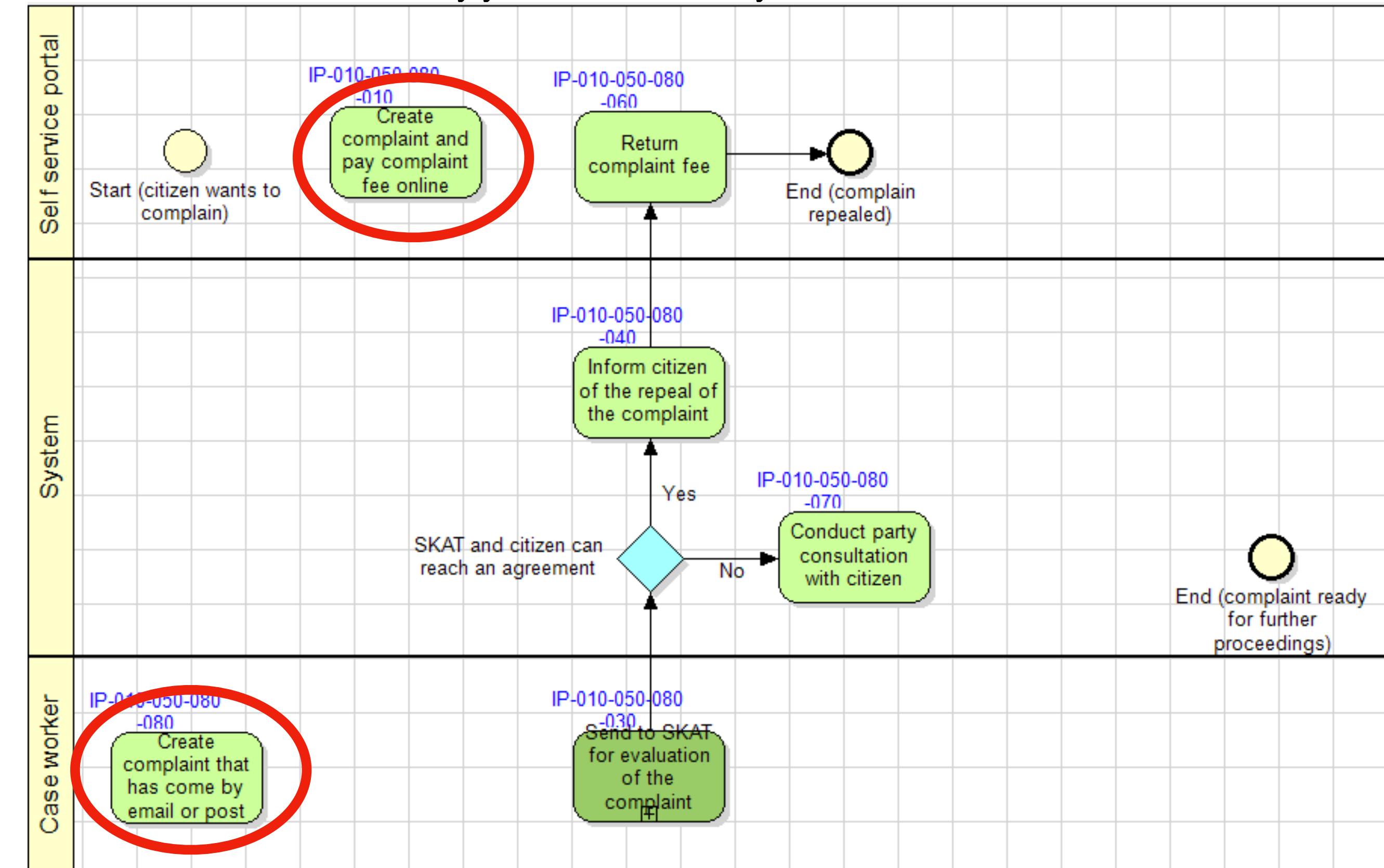
Syntax conformance

- Focus on the proper use of language abstractions
- Key: does it “reads well”? Syntax
- Typically controlled via modelling tool/compiler
- Common errors:
 - Unconnected relations
 - Missing variables
 - Wrong timestamps, etc.



Syntax conformance

- Focus on the proper use of language abstractions
- Key: does it “reads well”? Syntax
- Typically controlled via modelling tool/compiler
- Common errors:
 - Unconnected relations
 - Missing variables
 - Wrong timestamps, etc.

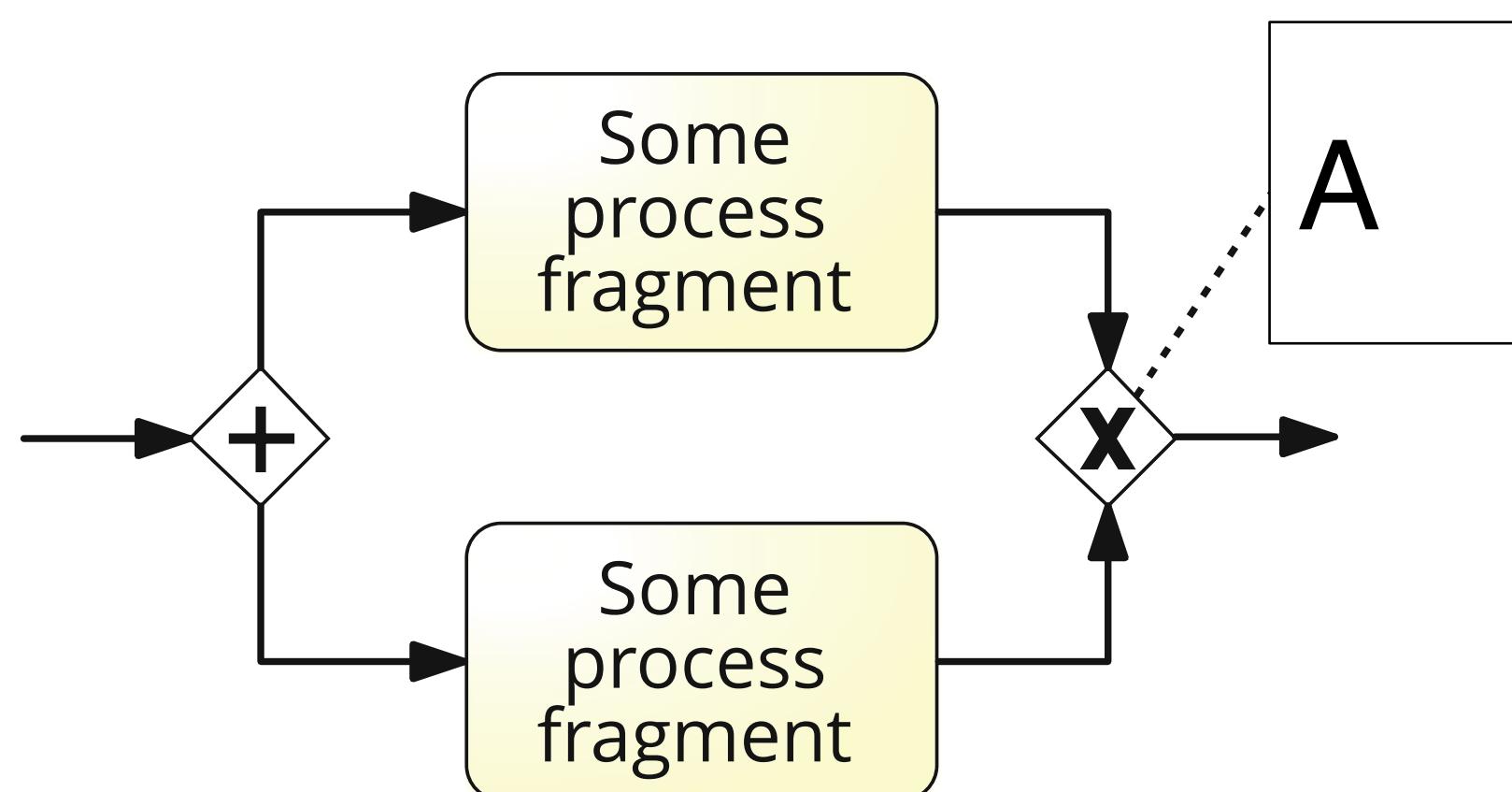


Soundness

- BPMN adopts the so-called *implicit termination semantics*: a process instance completes only when each token flowing in the model reaches an end event

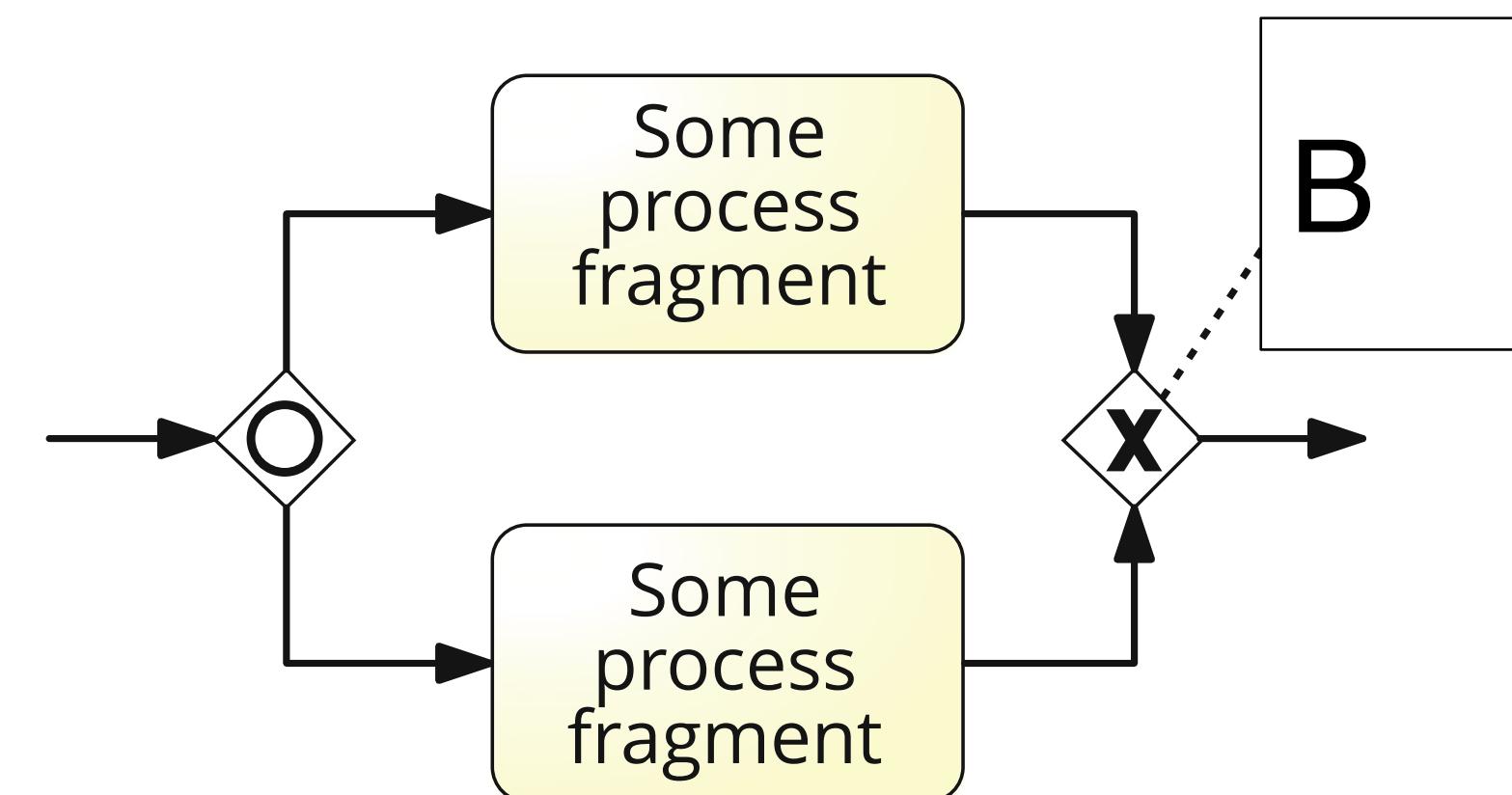
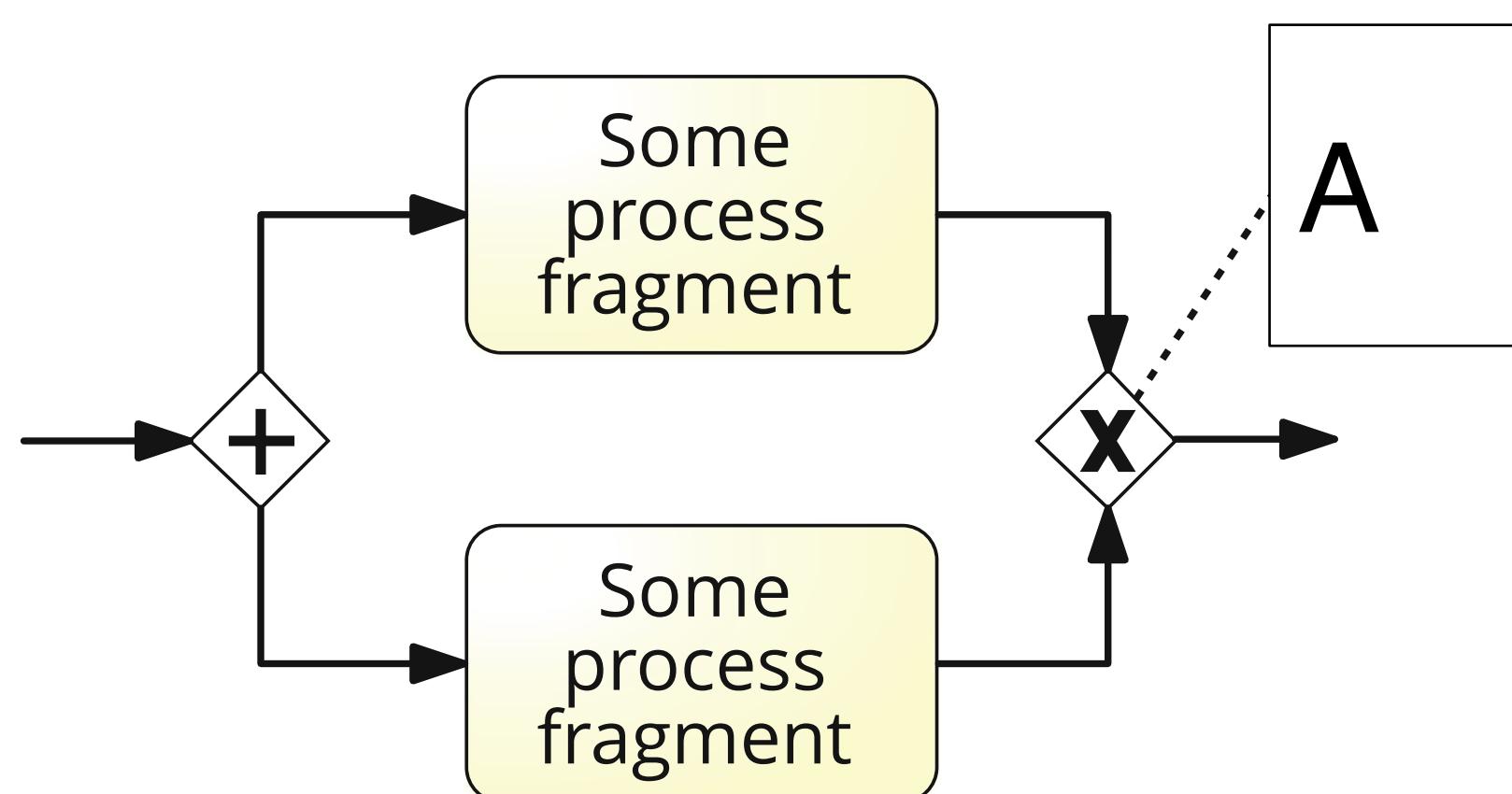
Soundness

- BPMN adopts the so-called *implicit termination semantics*: a process instance completes only when each token flowing in the model reaches an end event
 - Exercise: What is wrong with these models?



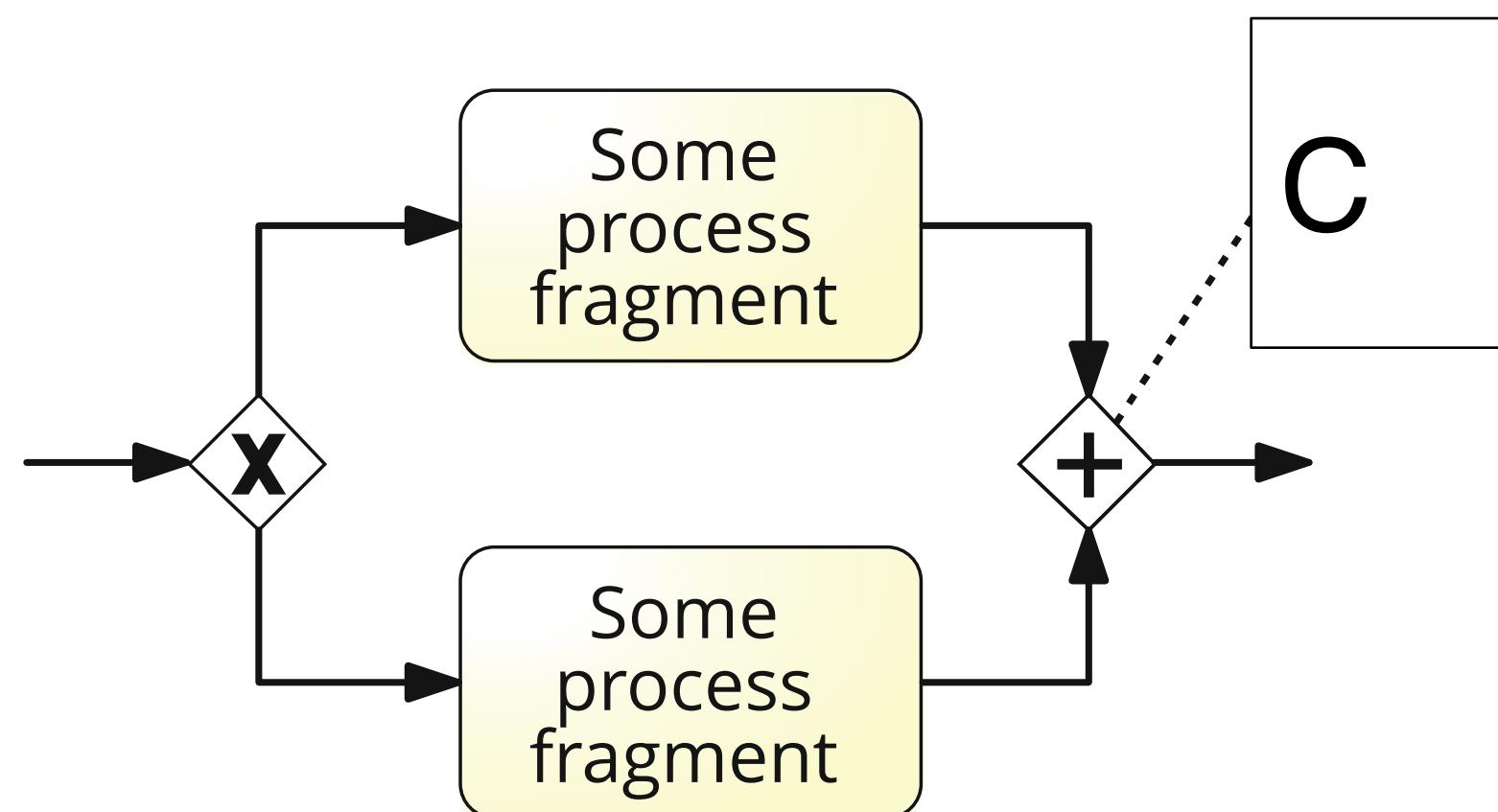
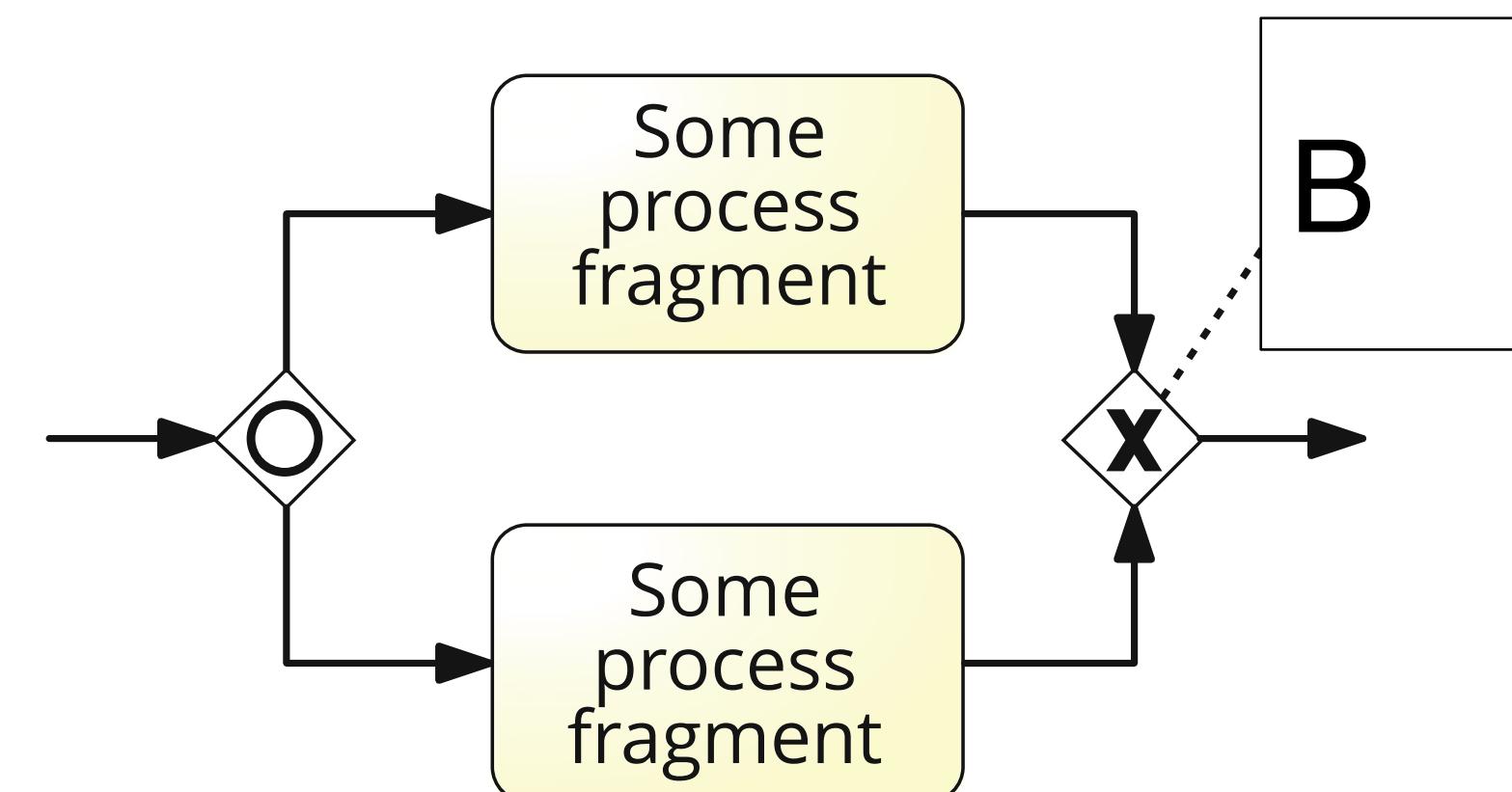
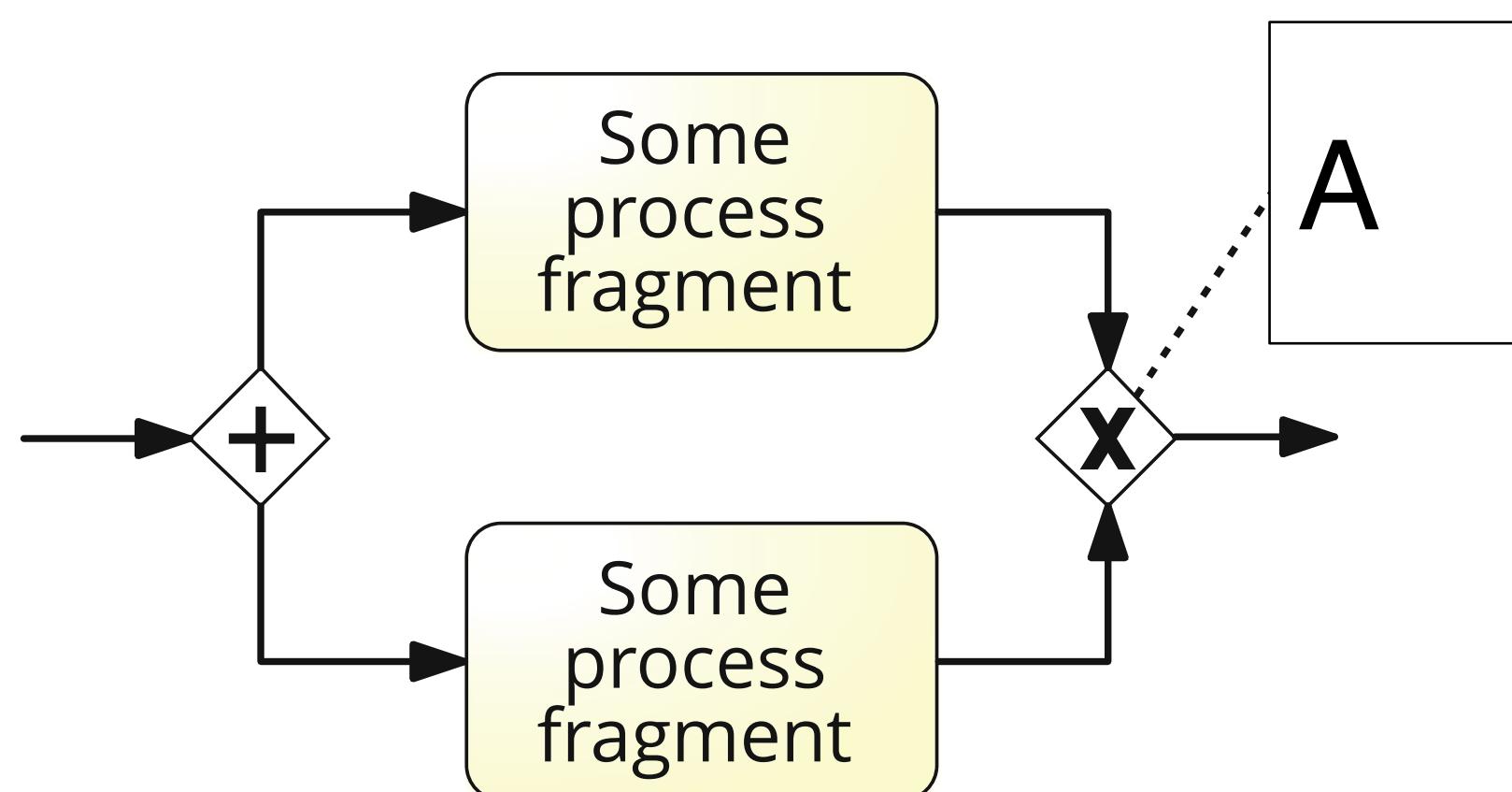
Soundness

- BPMN adopts the so-called *implicit termination semantics*: a process instance completes only when each token flowing in the model reaches an end event
 - Exercise: What is wrong with these models?



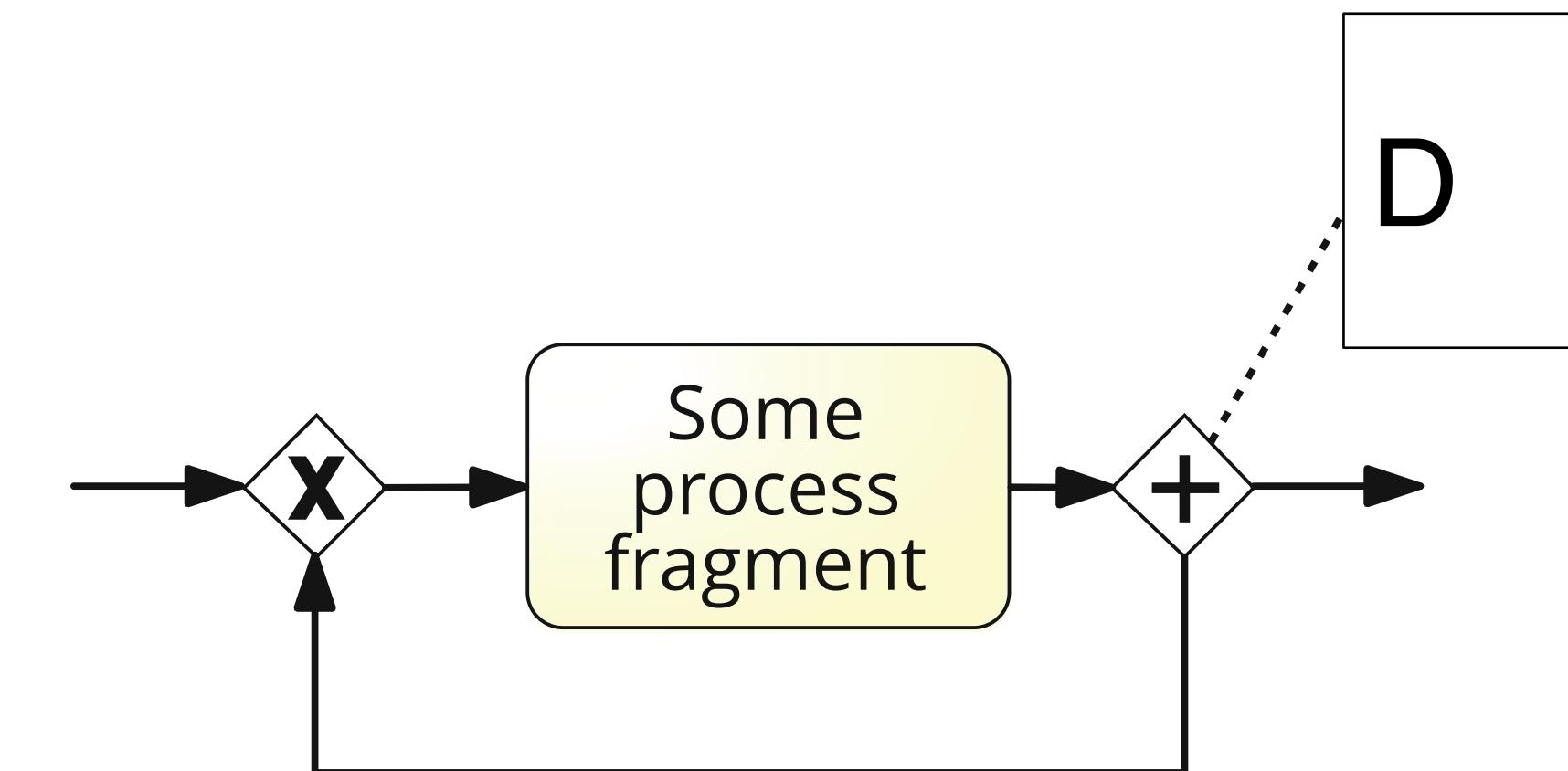
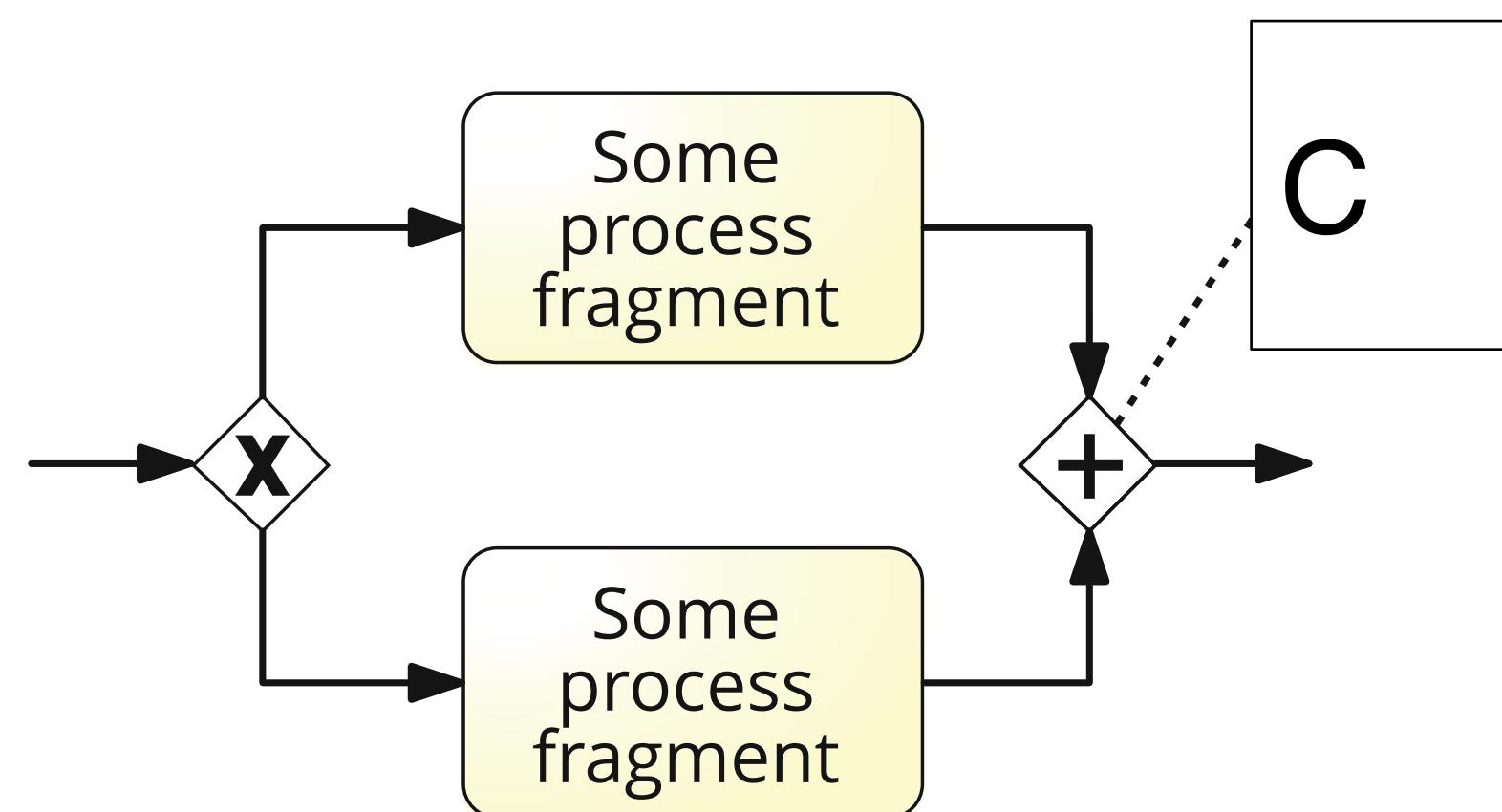
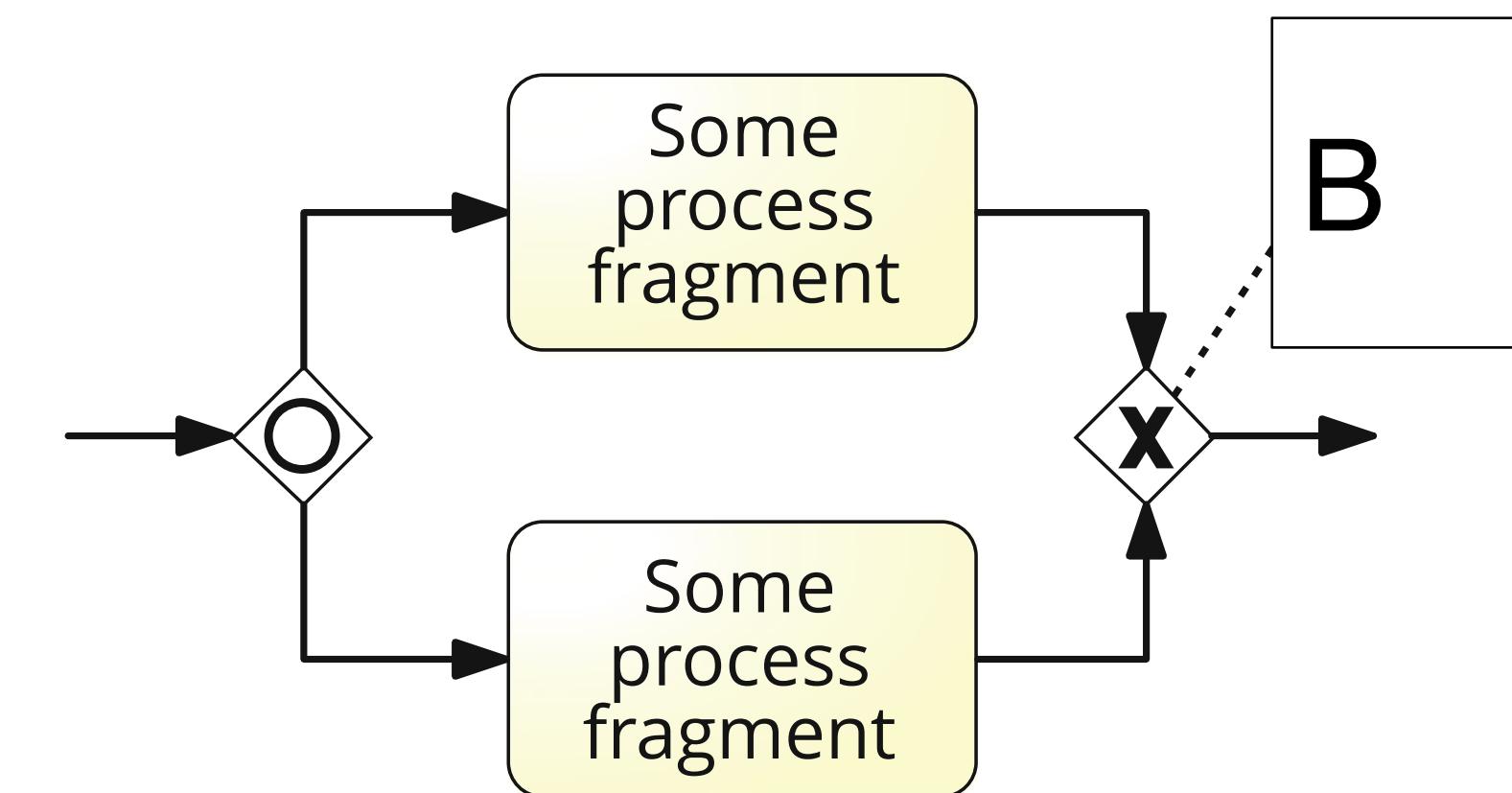
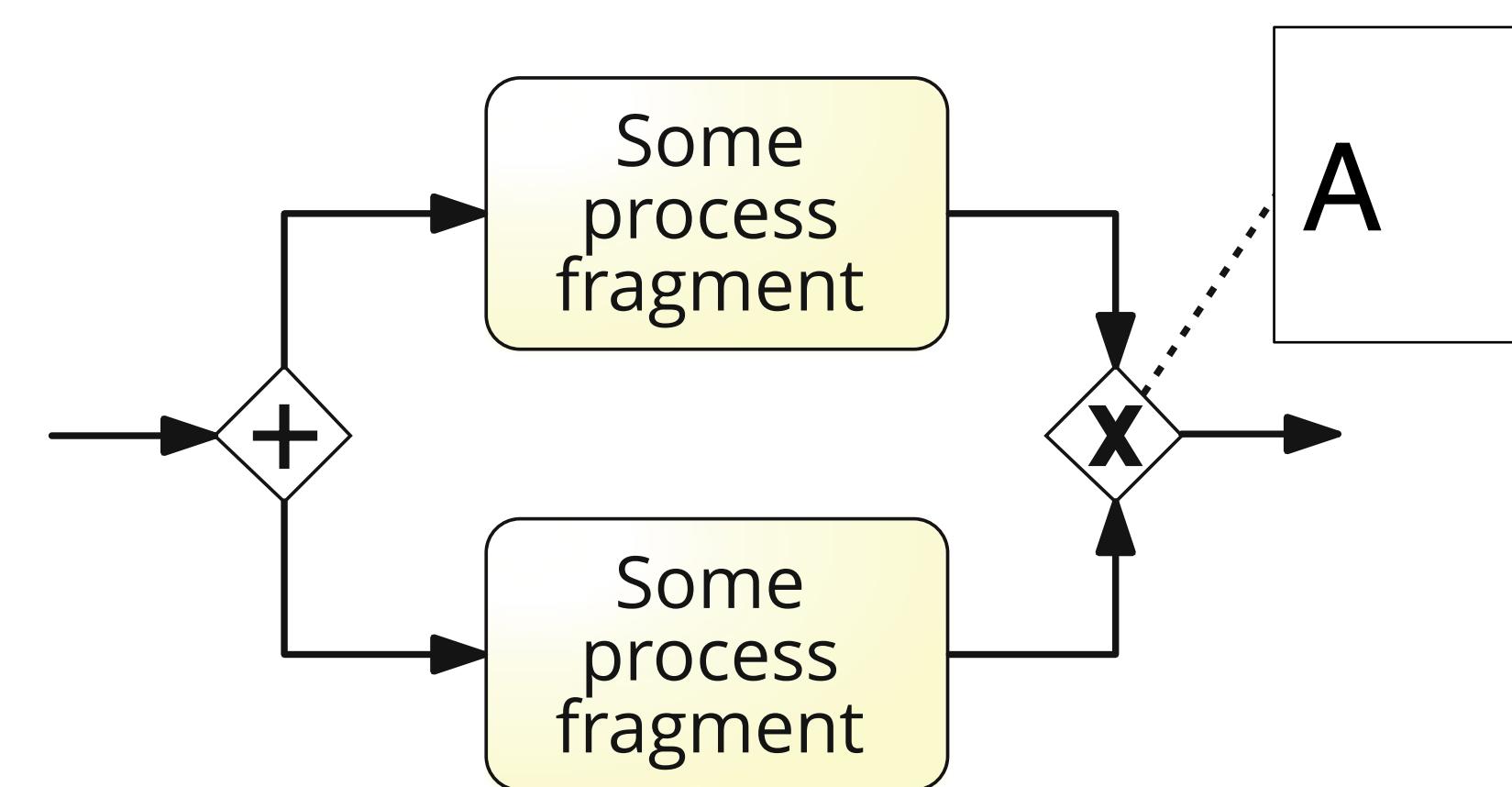
Soundness

- BPMN adopts the so-called *implicit termination semantics*: a process instance completes only when each token flowing in the model reaches an end event
 - Exercise: What is wrong with these models?



Soundness

- BPMN adopts the so-called *implicit termination semantics*: a process instance completes only when each token flowing in the model reaches an end event
 - Exercise: What is wrong with these models?



Soundness

Dumas et. al. Fundamentals of BPM. Chapter 5.

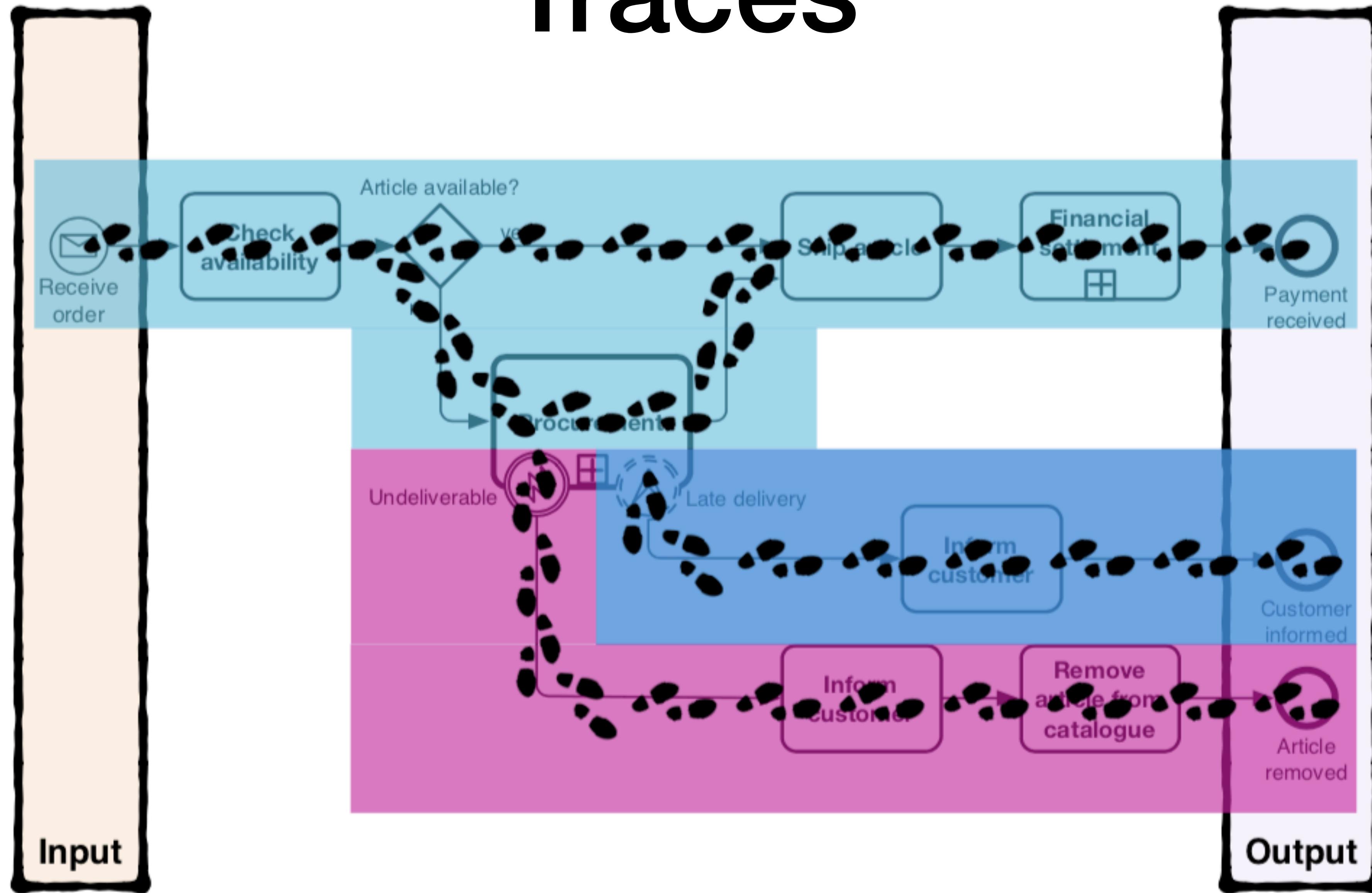
A process model is **behaviorally correct**, or sound, if and only if it satisfies the following behavioral rules:

1. **Option to complete:** any running process instance must eventually complete,
2. **Proper completion:** at the moment of completion, each token of the process instance must be in a different end event,
3. **No dead activities:** any activity can be executed in at least one process instance.

Soundness

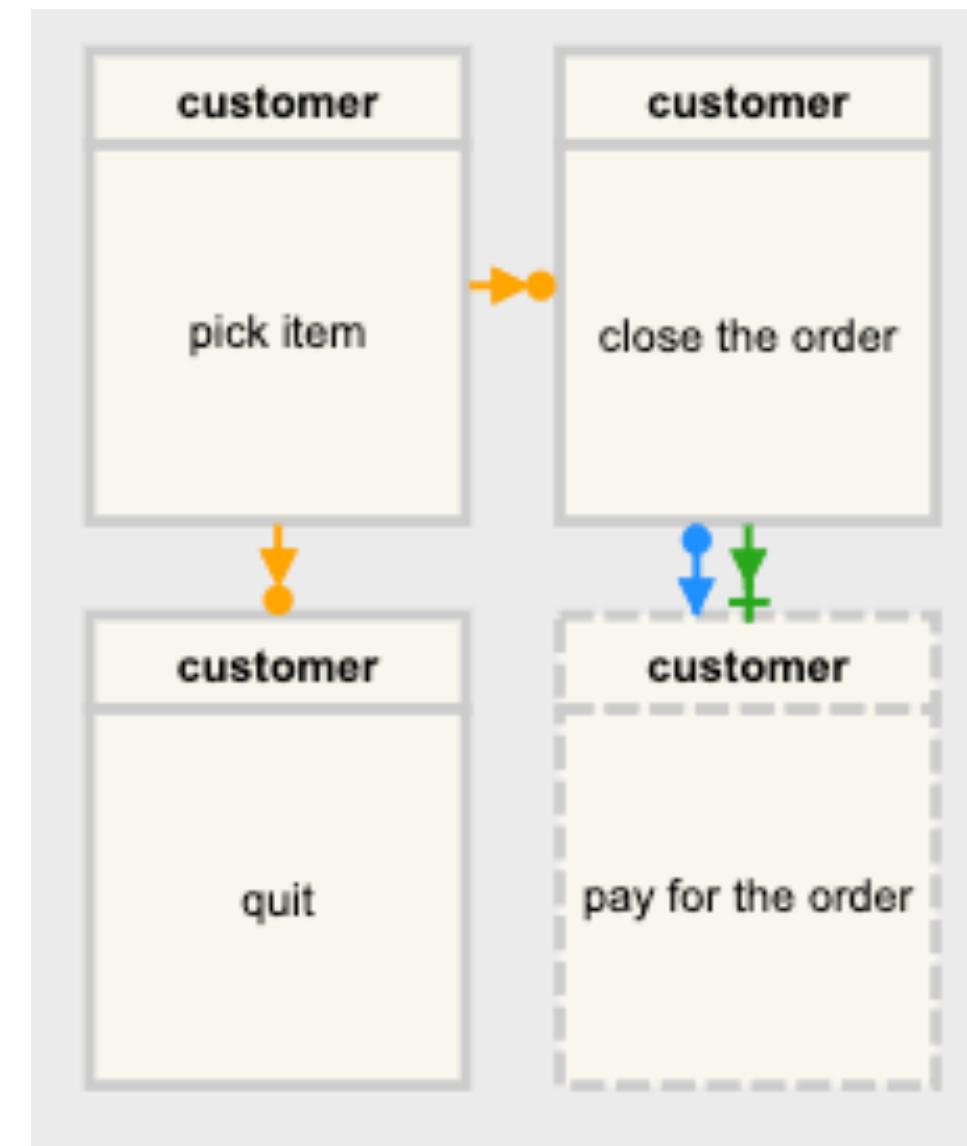
- Does my model have structural errors?
 - Does it ever terminate?
 - Does it ever get an accepting state?
 - Does it generate circular dependencies (deadlocks)?

Traces



Traces

- Trace model for DCR graphs:
 - Any enabled activity can be executed.
 - An activity can be executed any number of times, until disabled.
 - A trace is accepting, when there are no pending activities to be executed.



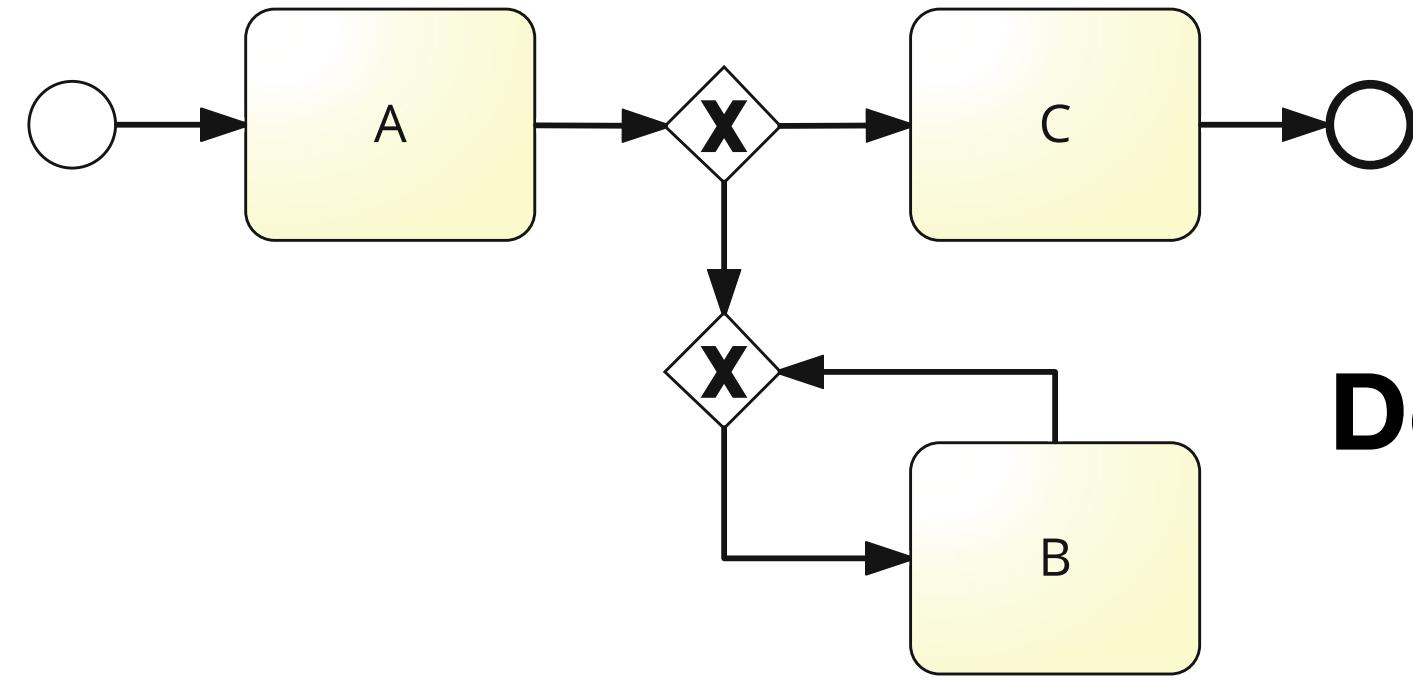
Simulation Log •

Trace	Accepting
<>	Yes
<Pick Item>	Yes
<Pick Item, Quit>	Yes
<Pick Item, Close Order, PayOrder>	Yes
<Pick Item, Pick Item, Close Order, Pay Order>	Yes

Soundness (I): Termination

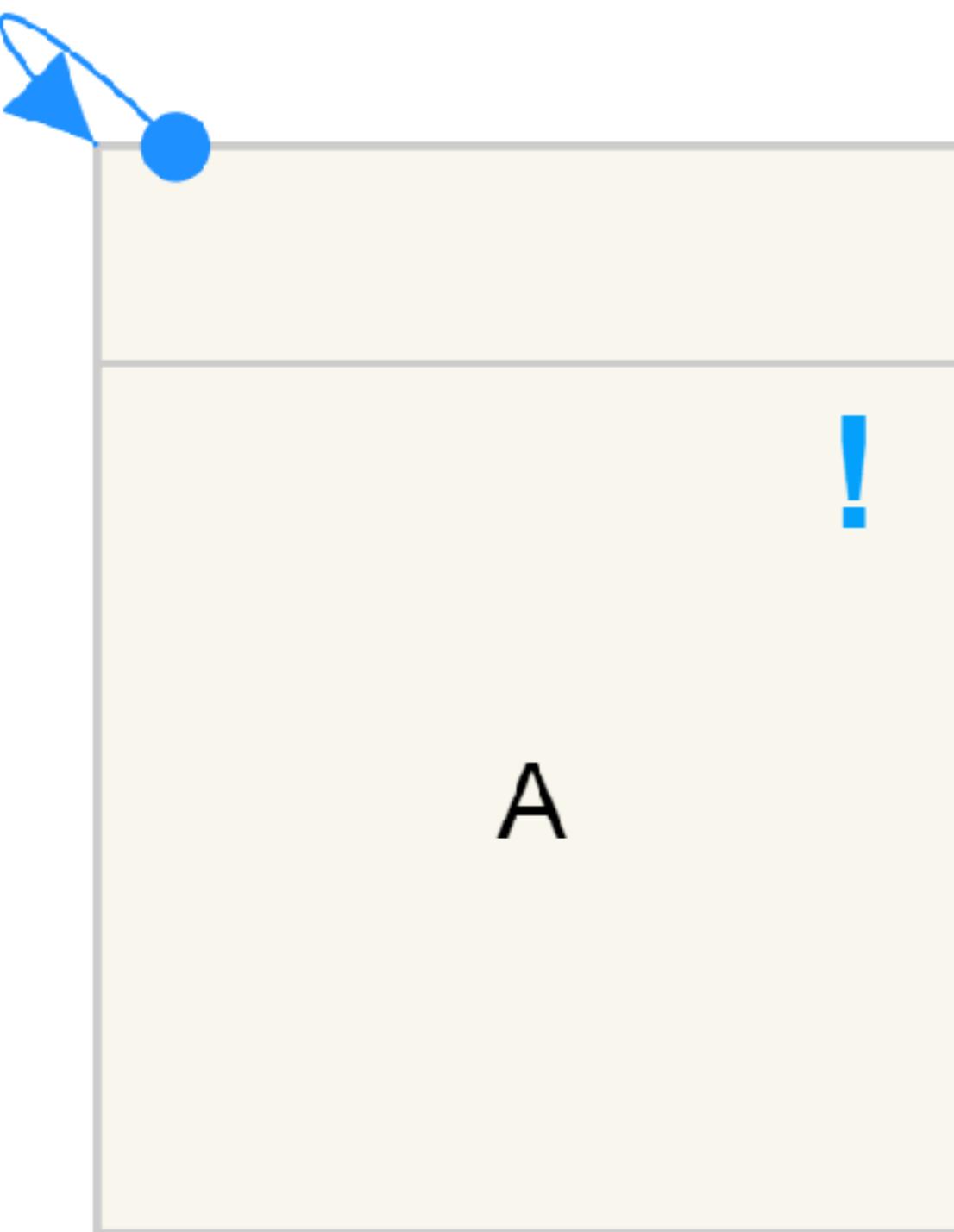
- Recall the trace model for DCR graphs:
 - Any enabled activity can be executed
 - An activity can be executed any number of times, unless it is disabled
- A trace is **accepting**, when there are no pending activities to be executed
- Trace **length**: the number of activities executed so far

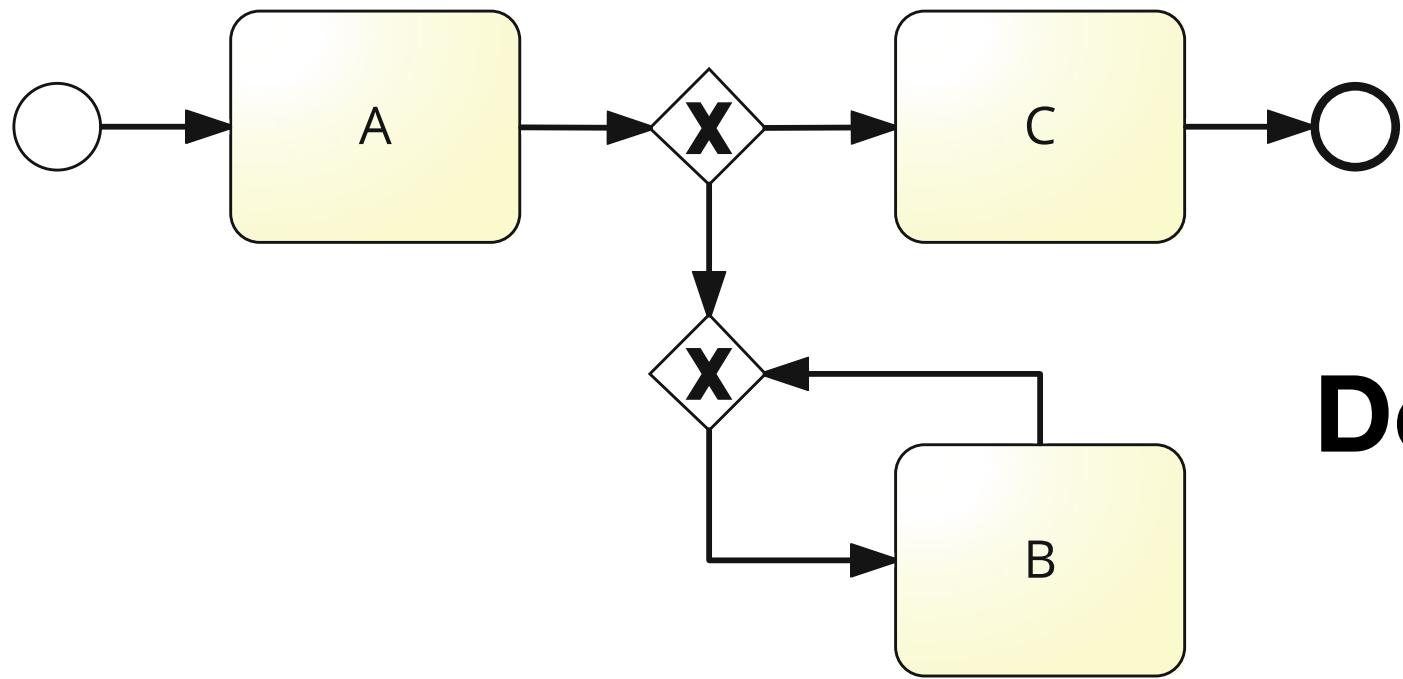
Trace	Length
$\langle \rangle$	0
$\langle \text{Pick Item} \rangle$	1
$\langle \text{Pick Item}, \text{Quit} \rangle$	2
$\langle \text{Pick Item}, \text{Close Order}, \text{PayOrder} \rangle$	3
$\langle \text{Pick Item}, \text{Pick Item}, \text{Close Order}, \text{Pay Order} \rangle$	4



Termination

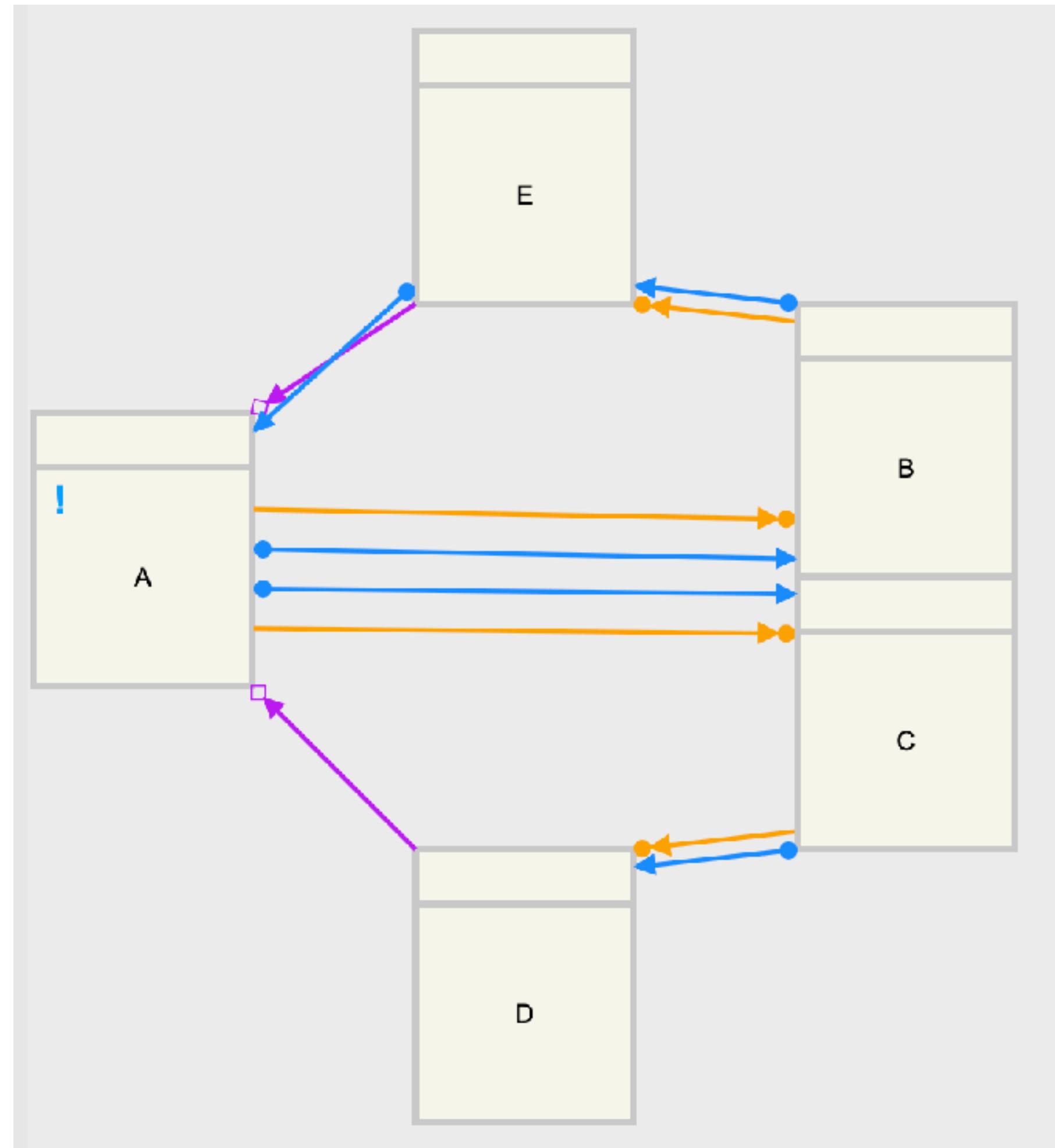
Do these models generate accepting traces?

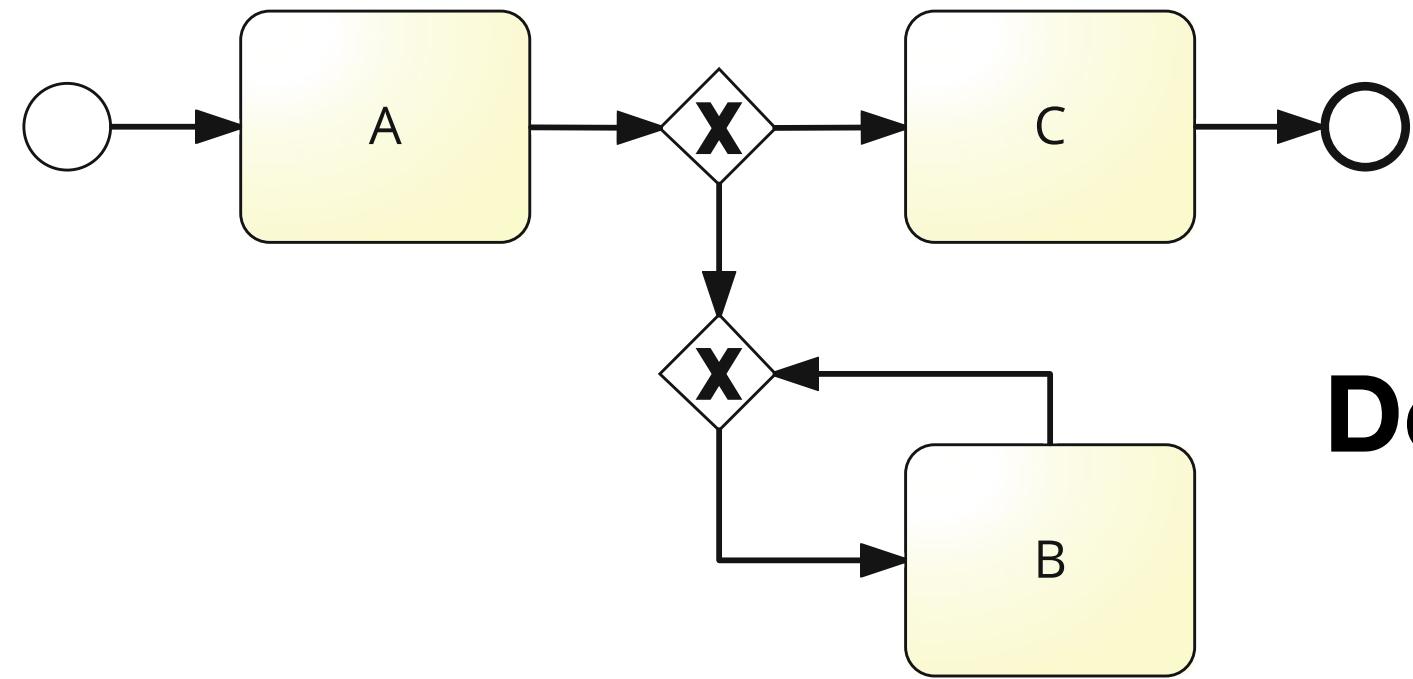




Termination

Do these models generate accepting traces?



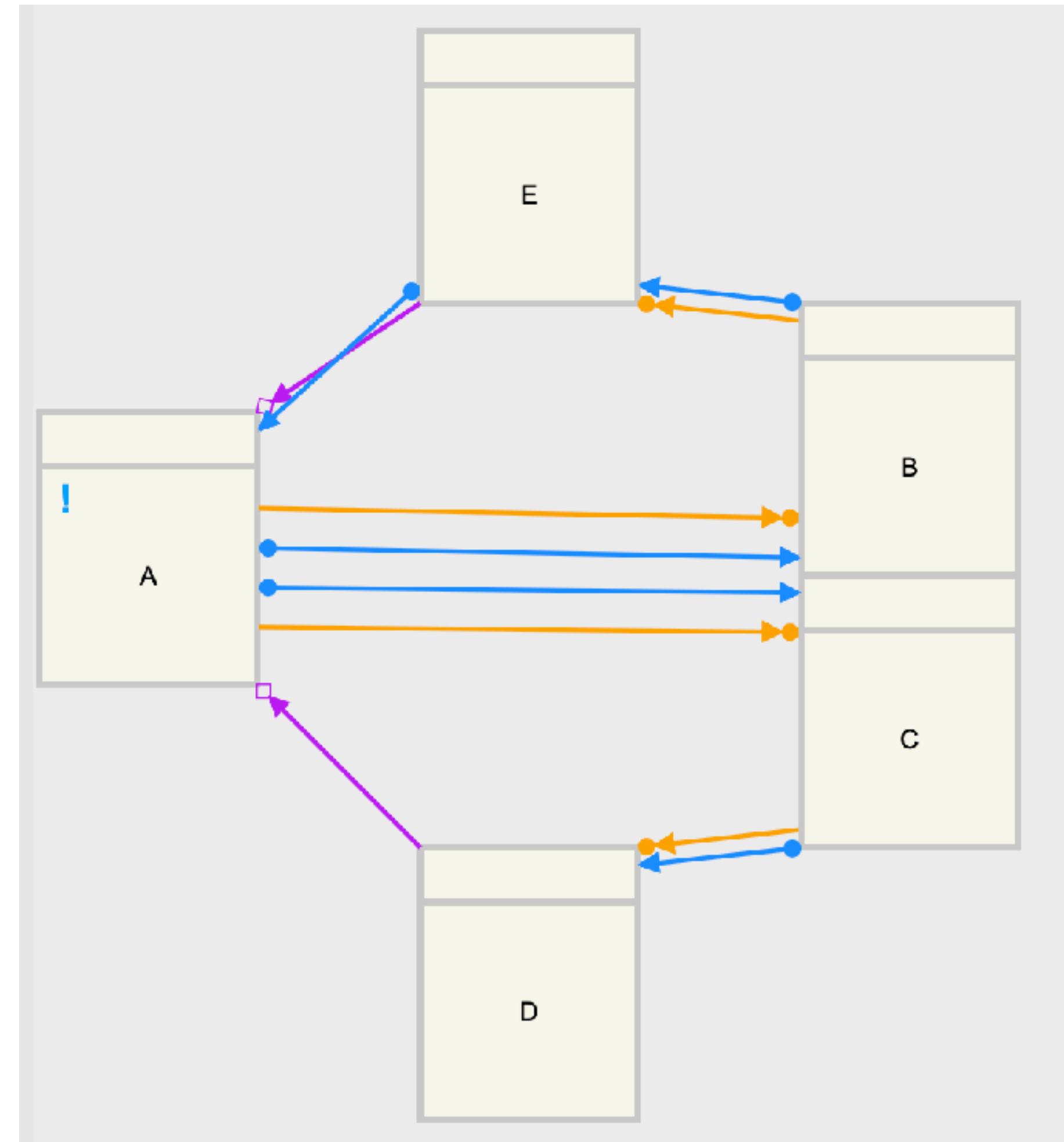


Termination

Do these models generate accepting traces?

A process is **non-terminating** if for any trace of length n , the execution of an activity generates a non-accepting trace, for any n

Hint: check for circular response/milestones dependencies in the graph

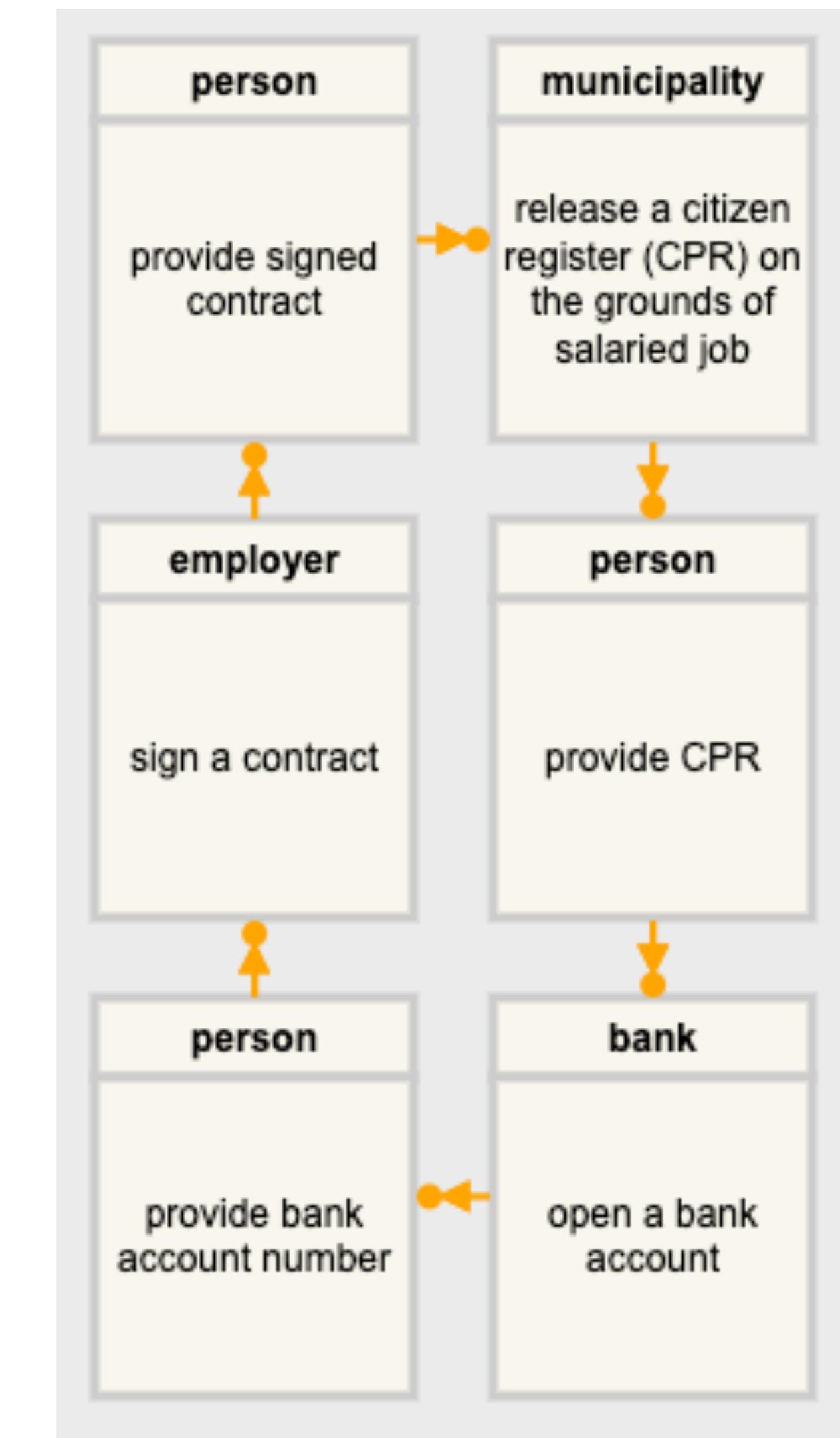


Soundness (II): The null process!

- In order to release a citizen register (CPR) on the grounds of salaried job, the municipality needs the signed contract of the person
- In order to sign a contract, the employer needs to have the bank account of the person
- In order to open a bank account, the bank needs to get the CPR of the person

Soundness (II): The null process!

- In order to release a citizen register (CPR) on the grounds of salaried job, the municipality needs the signed contract of the person
- In order to sign a contract, the employer needs to have the bank account of the person
- In order to open a bank account, the bank needs to get the CPR of the person



<http://www.dcrgraphs.net/Tool?id=9671#>

Soundness (II): The null process!

- In order to release a citizen register (CPR) on the grounds of salaried job, the municipality needs the signed contract of the person
- In order to sign a contract, the employer needs to have the bank account of the person
- In order to open a bank account, the bank needs to get the CPR of the person



When would you execute this graph?

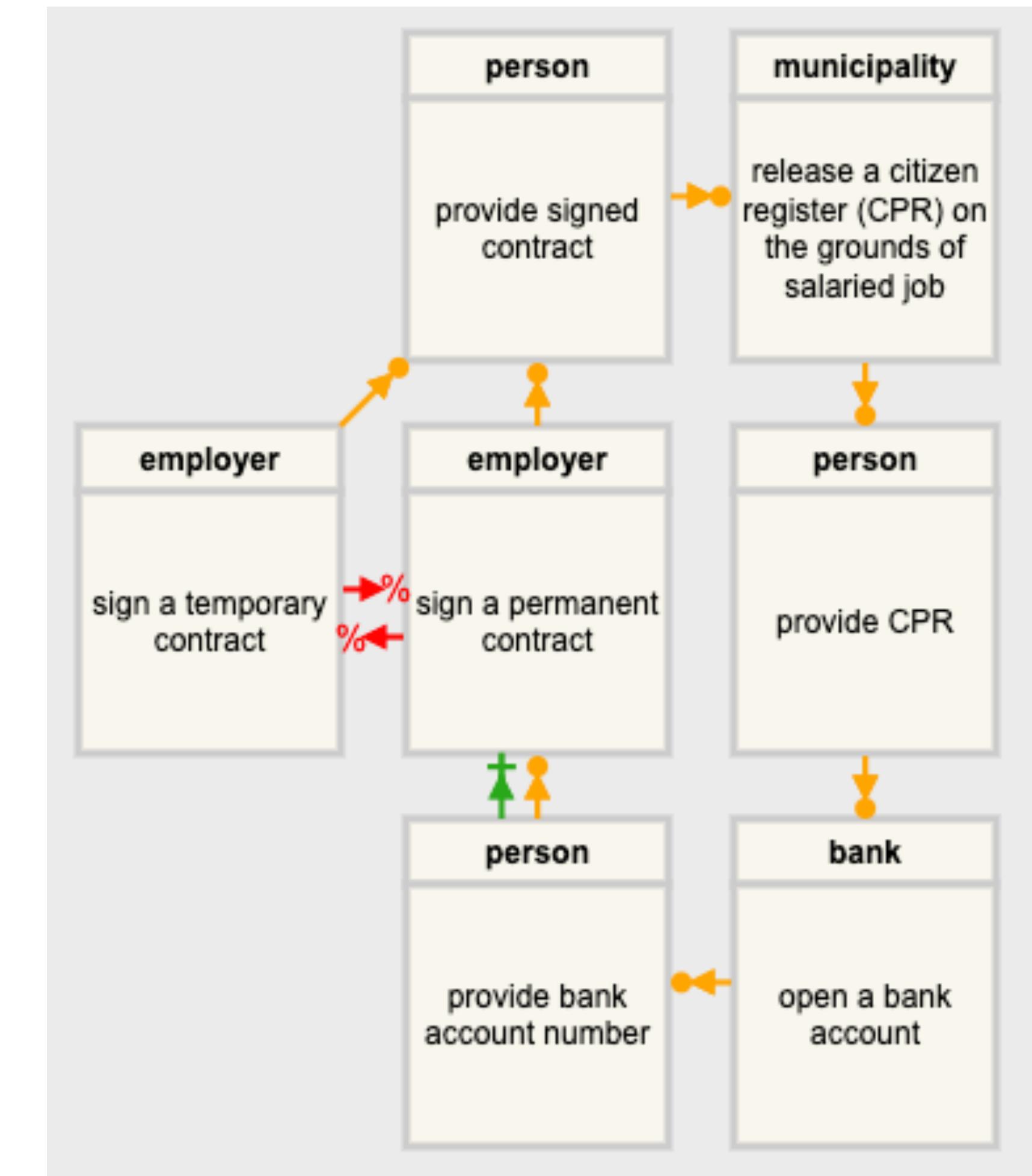
<http://www.dcrgraphs.net/Tool?id=9671#>

Null Process: Avoiding circular dependencies

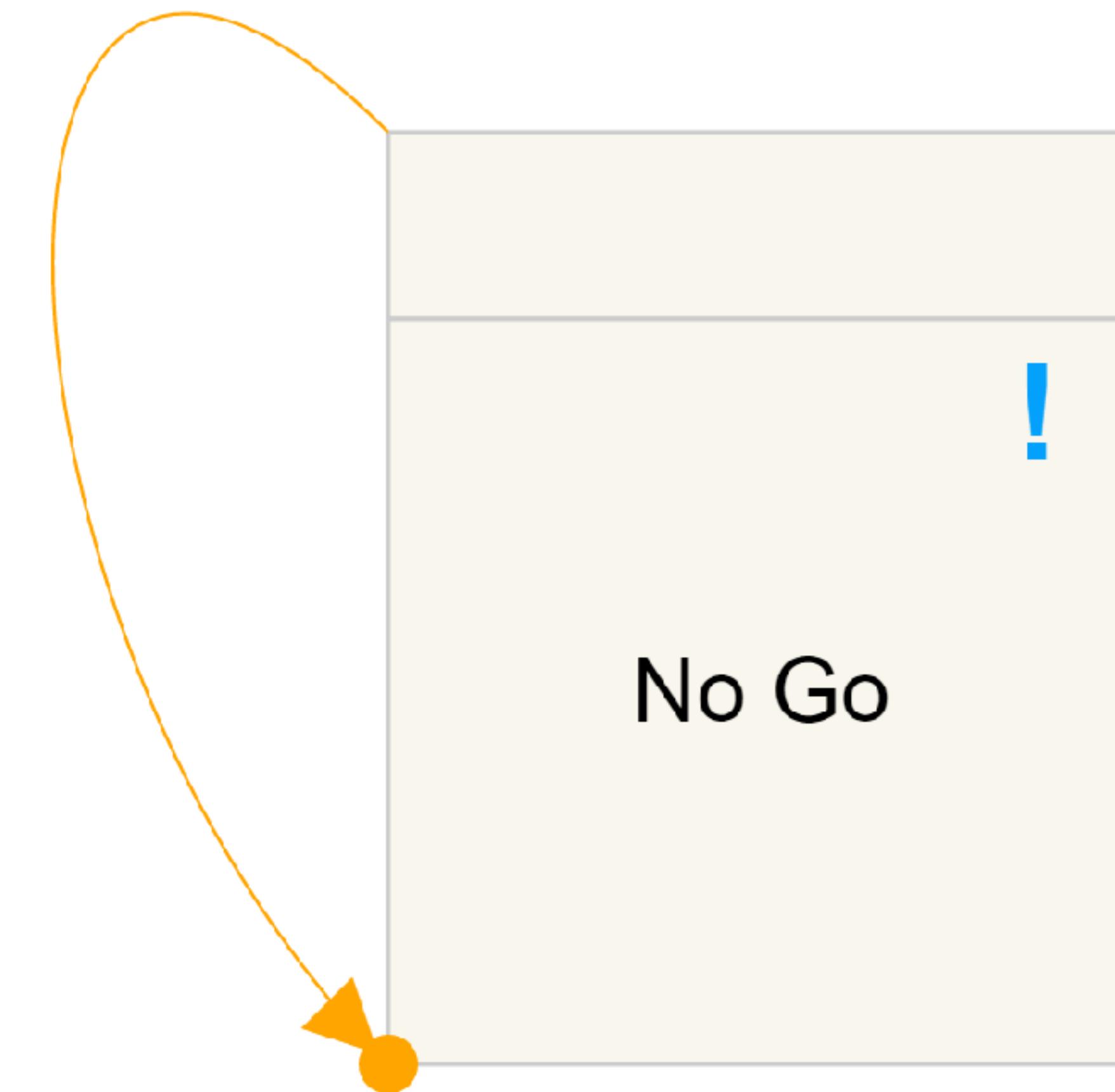
- Break circular dependencies by adding additional activities

Null Process: Avoiding circular dependencies

- Break circular dependencies by adding additional activities

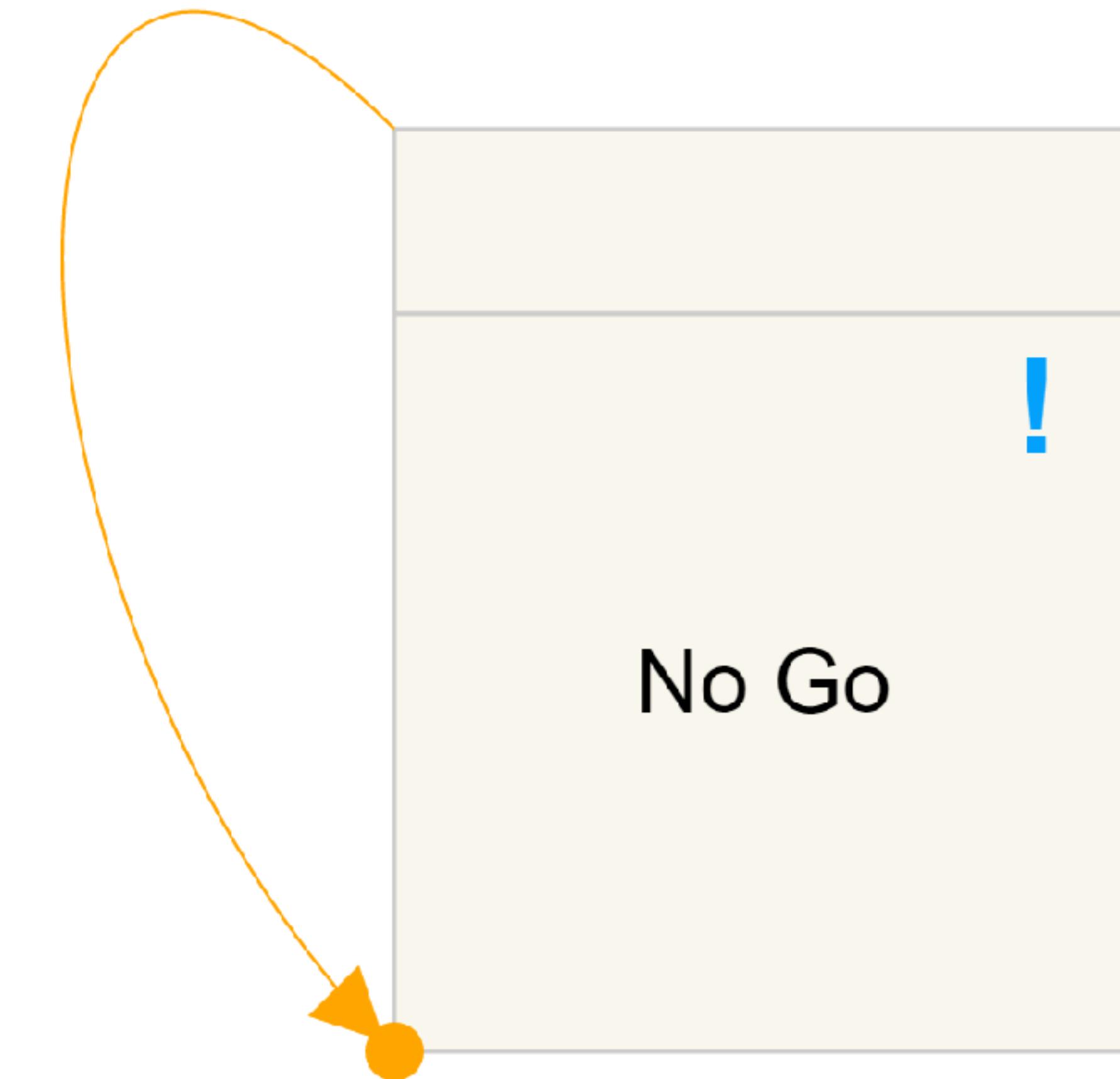


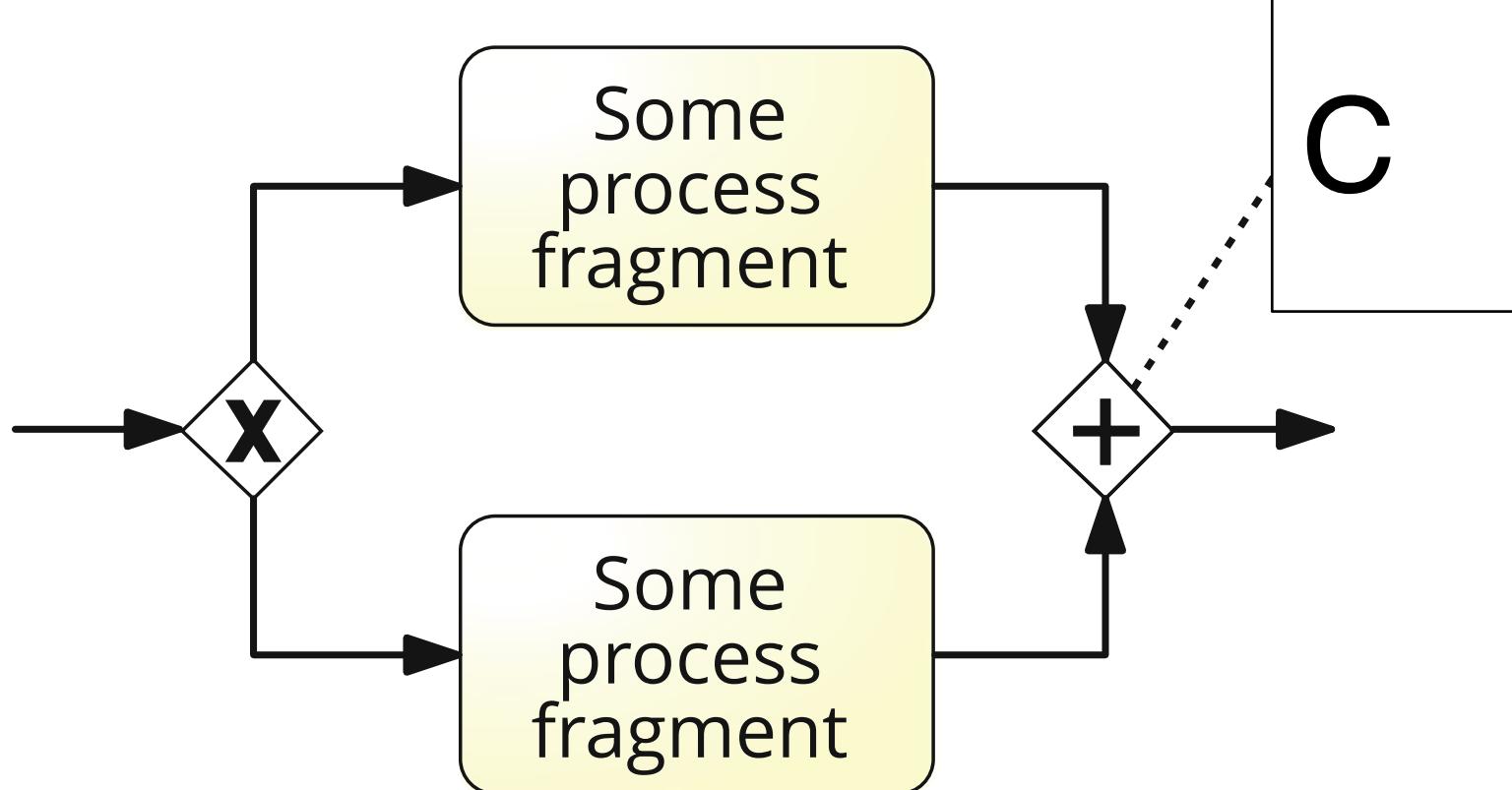
Deadlocks!



Deadlocks!

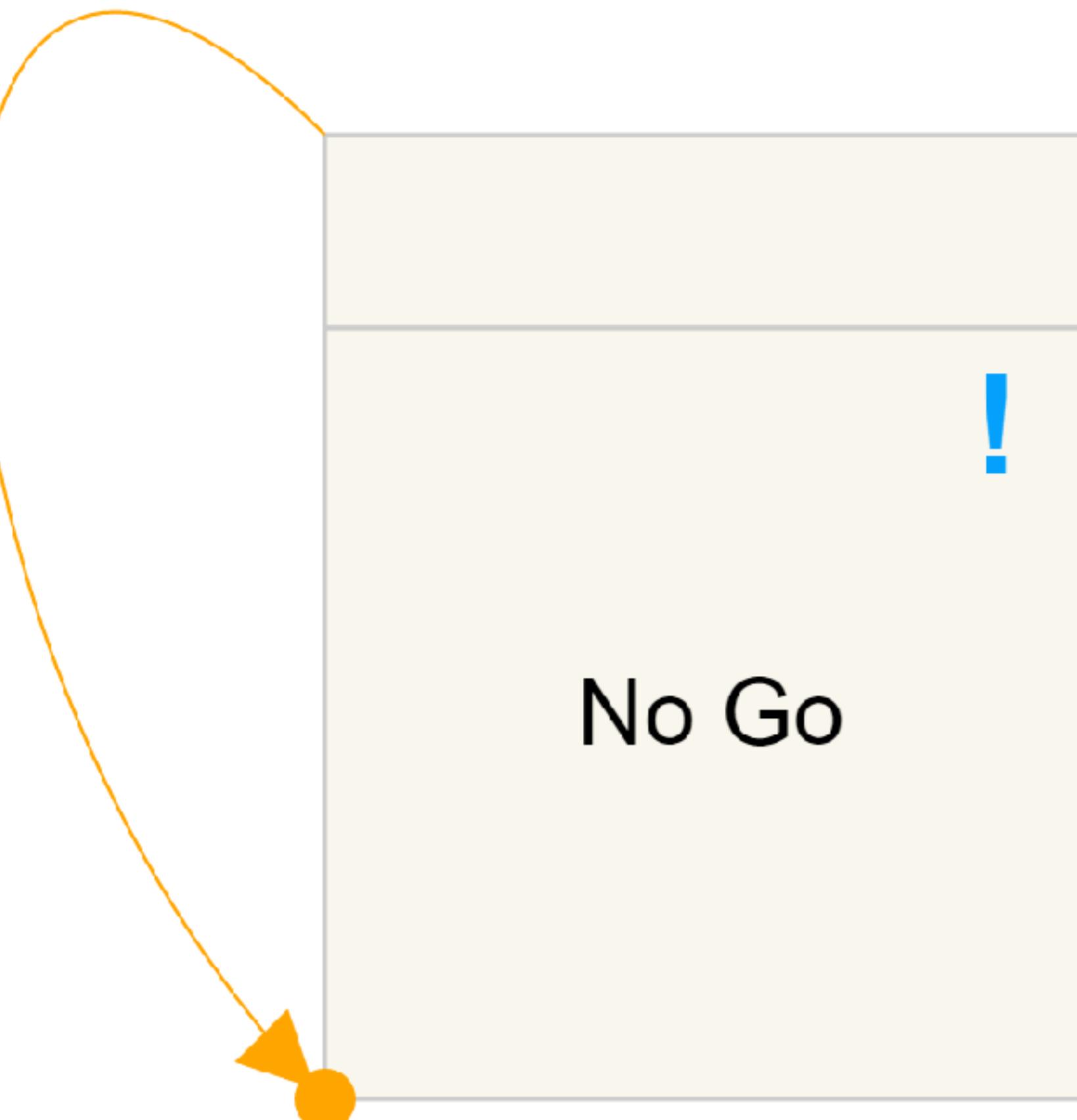
- There is a reachable state with no enabled activities, and there is at least one pending activity
- A DCR Graph is **deadlock free** if and only if for any execution, there is either an enabled event or no included required responses.

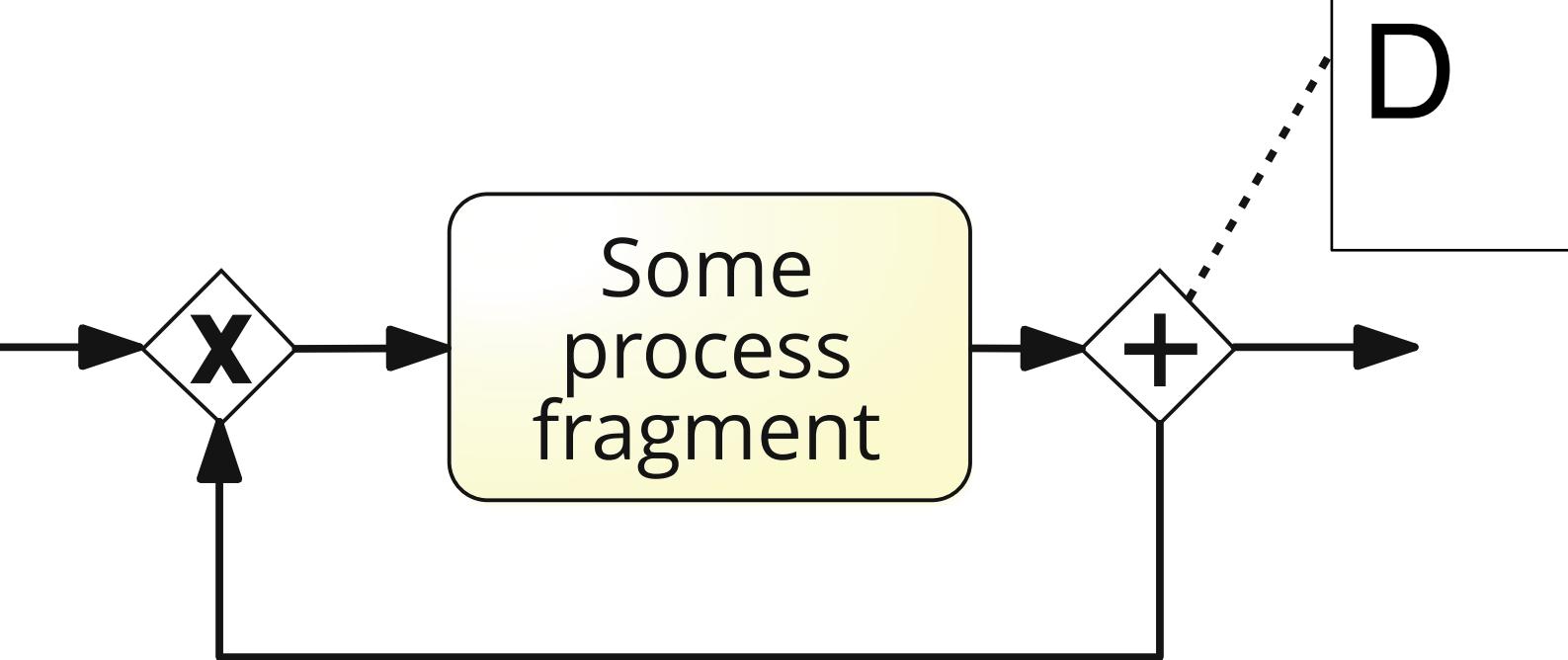




Deadlocks!

- There is a reachable state with no enabled activities, and there is at least one pending activity
- A DCR Graph is **deadlock free** if and only if for any execution, there is either an enabled event or no included required responses.

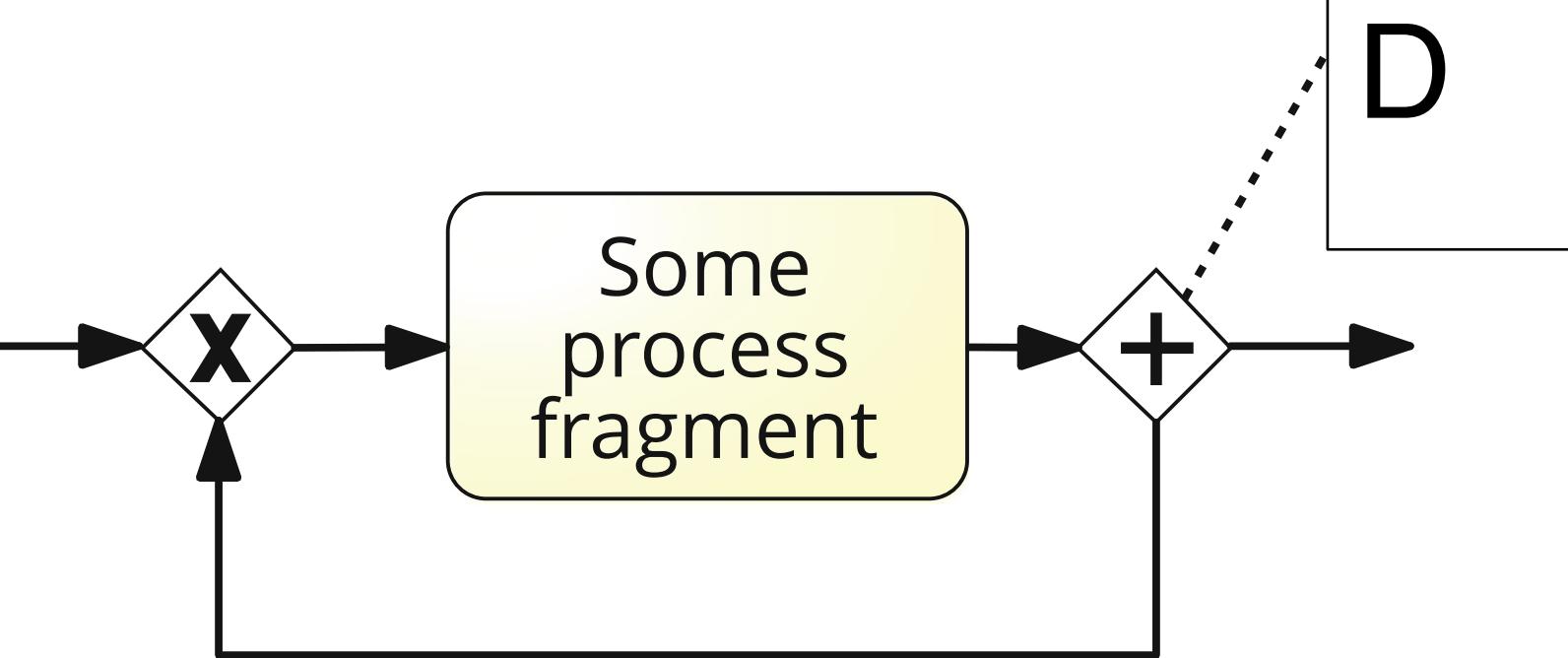




Liveness

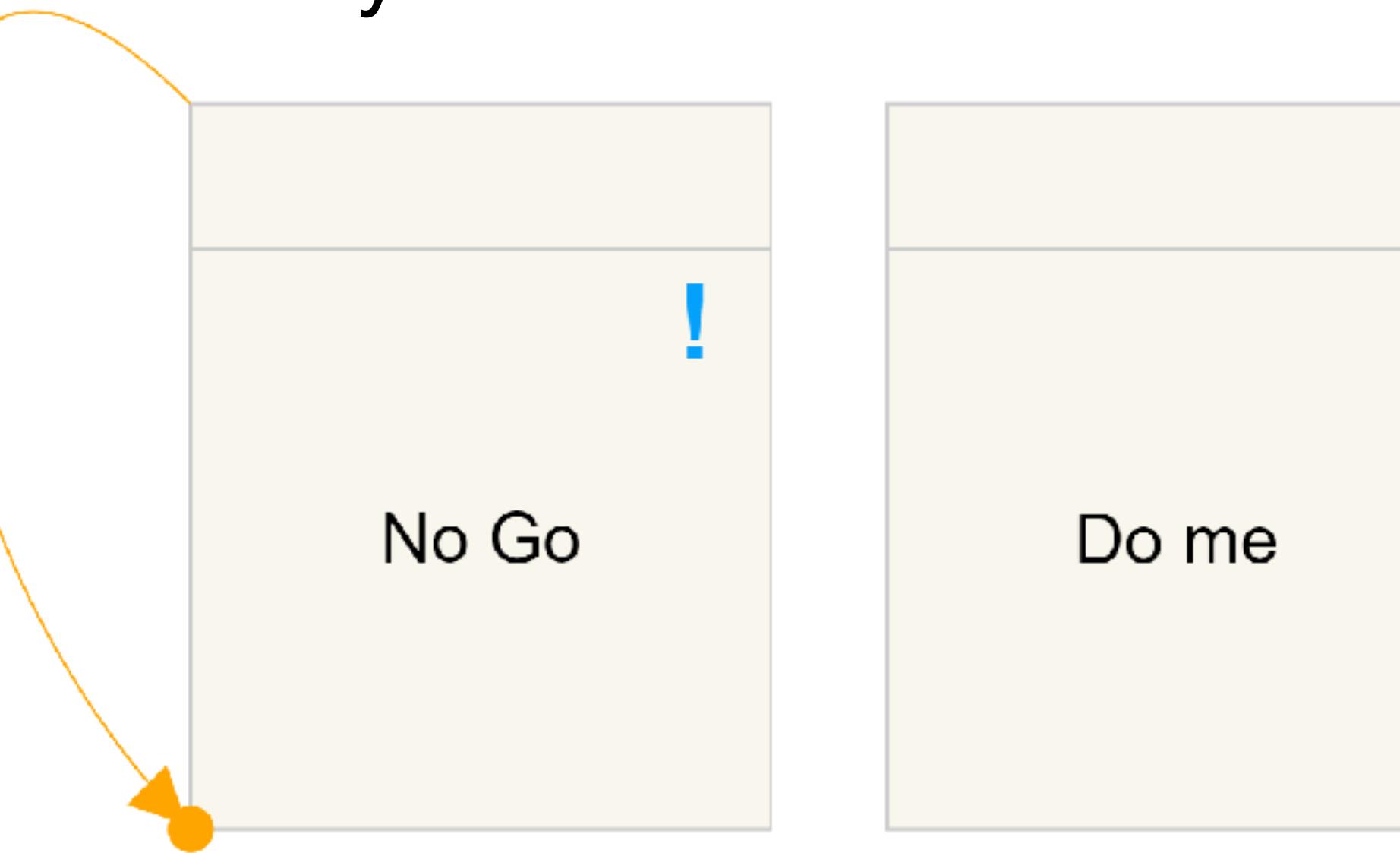
- Deadlock-freedom guarantees that we will not get stuck, but it does not say that we will do what we require:





Liveness

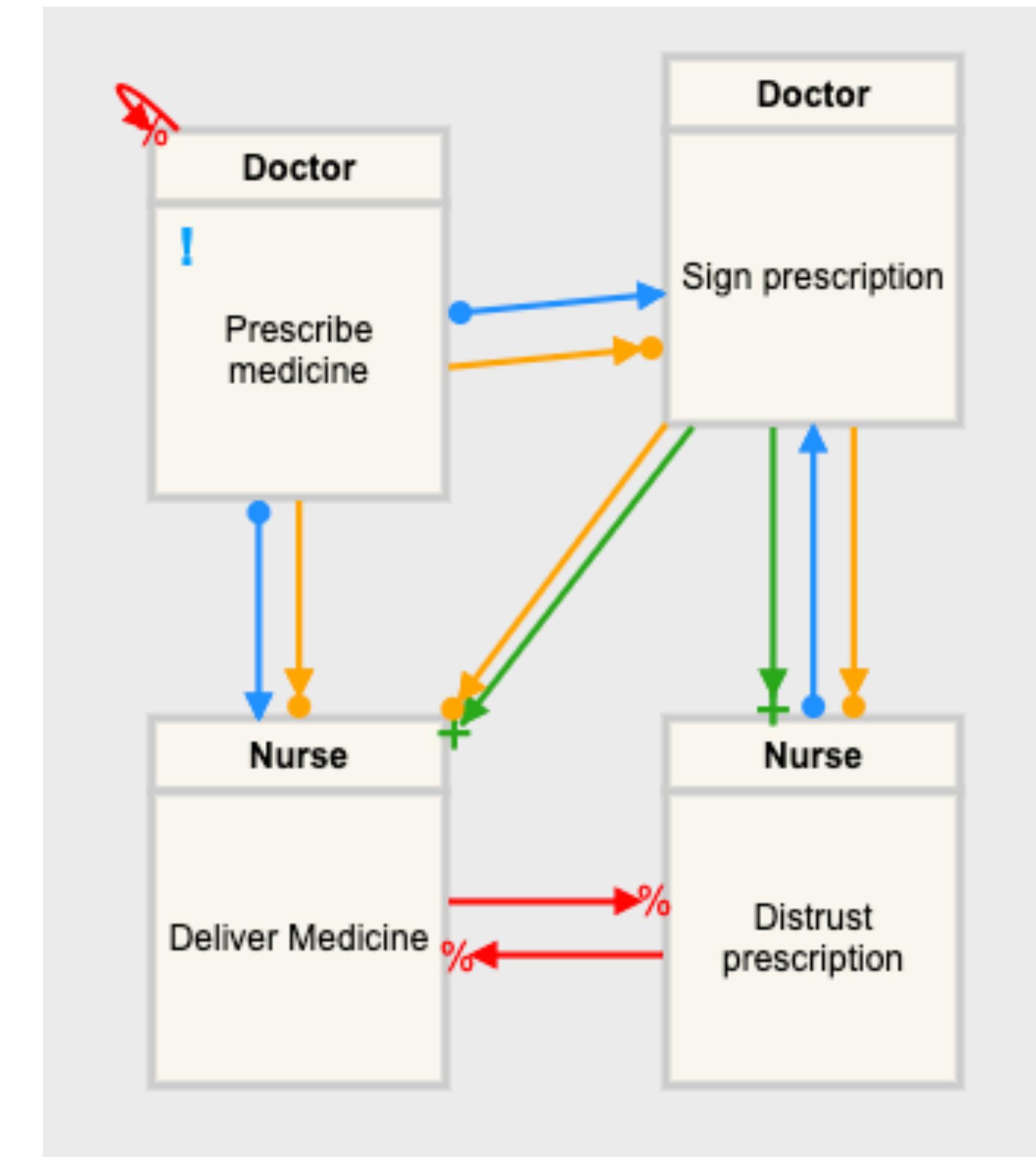
- Deadlock-freedom guarantees that we will not get stuck, but it does not say that we will do what we require:



- **Livelock:** For all states reachable from the current state, there exists an enabled event and at least one included pending event that never gets excluded or enabled

Liveness

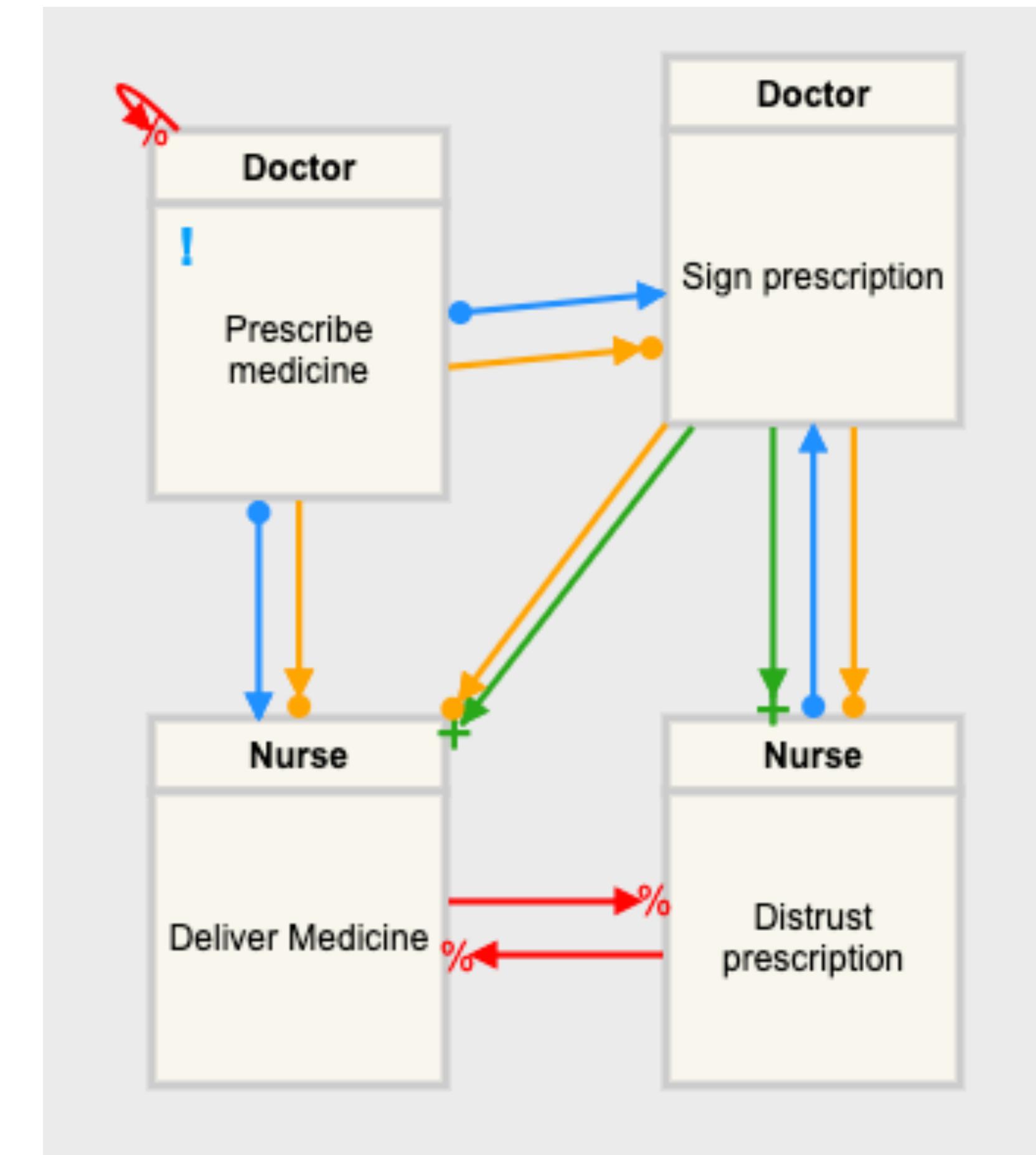
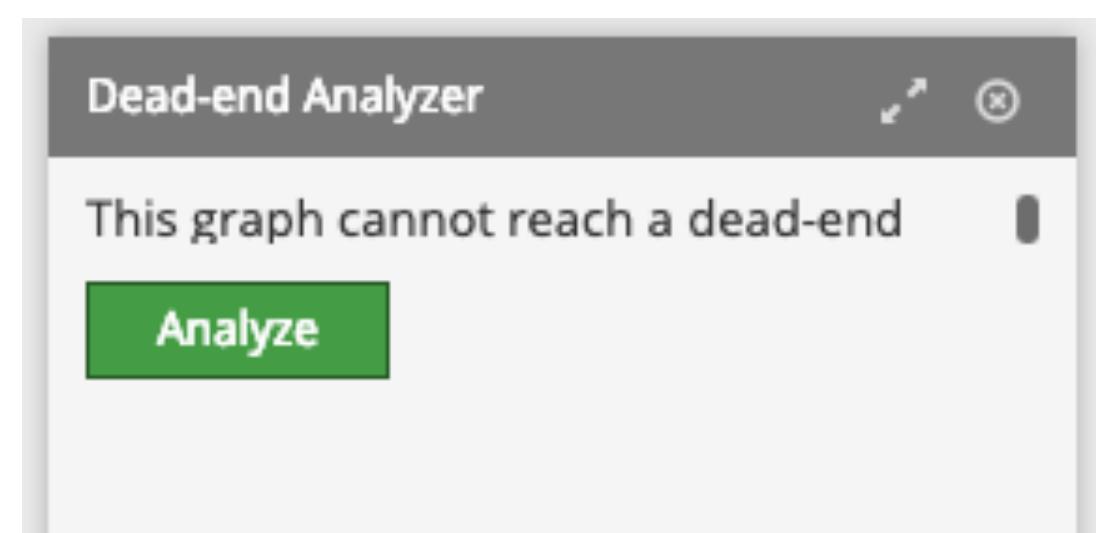
- A DCR Graph is **livelock free** if and only if, in every execution, it is always possible to continue along an accepting run (i.e. eventually execute or exclude any of the pending responses)



<http://www.dcrgraphs.net/Tool?id=9681>

Liveness

- A DCR Graph is **livelock free** if and only if, in every execution, it is always possible to continue along an accepting run (i.e. eventually execute or exclude any of the pending responses)



<http://www.dcrgraphs.net/Tool?id=9681>

- Before defining the best formalism to map a process, we need to consider its nature
- Complexity, predictability and repetitiveness help to understand process dimensions
- Knowledge-intensive processes are characterised for being complex and unpredictable
- Declarative process models as flexible process orchestrations
 - Less focus on how to do it, more on what are the rules of the game
- DCR graphs provides a way to simplify complex BPMN models into a simpler form

Assignment - part 2

- In this assignment, you will focus on the analysis and verification of process specification with timed constraints
 1. Extend the process description from your model, to identify at least 3 possible instances for real-timed behaviour as described in the timed systems lecture. Model resulting system behaviour in UPPAAL
 2. Identify a set (min. 5) of possible safety/liveness properties, and verify that your prototype exhibits them.
 3. Rewrite your process description so, instead of a process flow, it describes the game rules
 4. Identify 3 positive scenarios, 3 neutral and 3 forbidden scenarios
 5. Develop a DCR process model that represents the new process description
 6. Ensure that your process model correspond to the specification and complies to the scenarios.

Assignment - Part 2

- From the second lecture of DCR graphs
7. Incorporate the real timed behaviour from timed automata into DCR
 8. Identify a set (min. 5) of possible safety/liveness properties, and describe them as DCR graphs
 9. Implement a small engine that can execute DCR graphs based on its operational semantics
 10. Implement a compliance checker that given a Timed DCR process model and a Timed DCR graph describing temporal properties, it says whether the program is compliant or not.

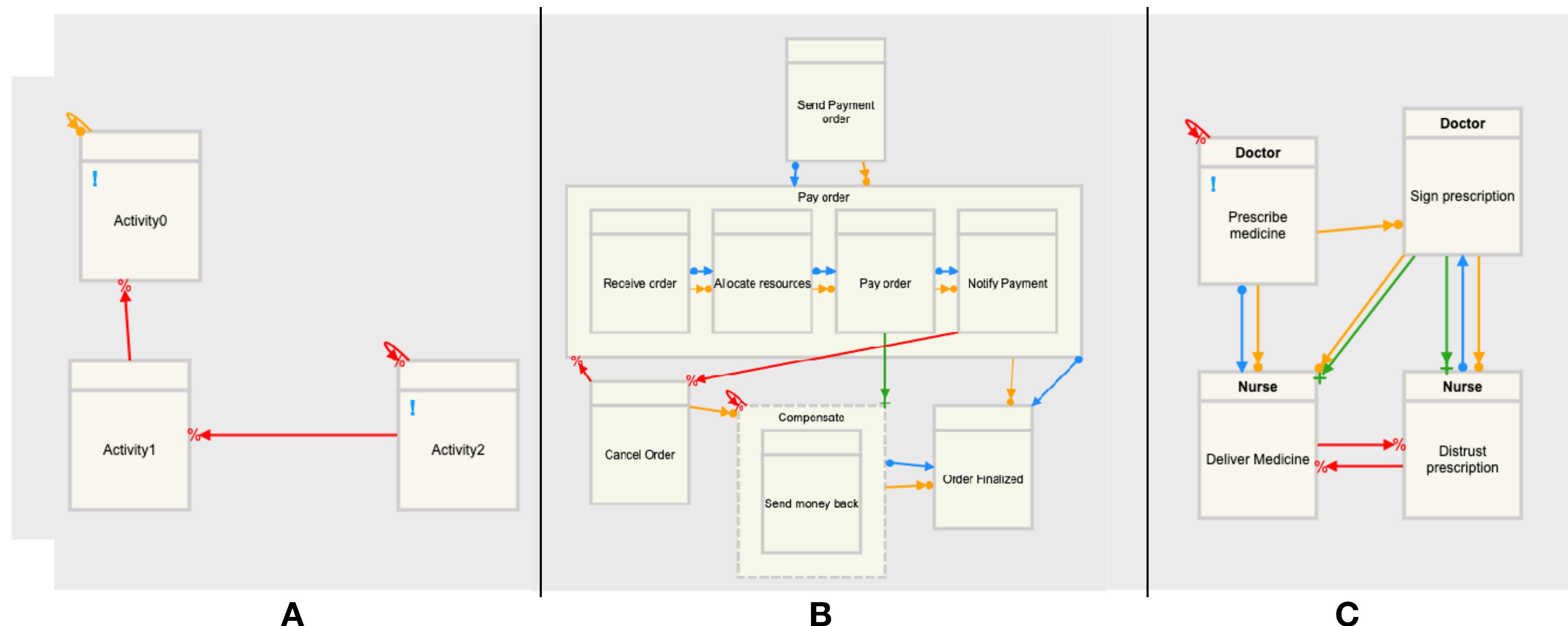
Exercises

Model the following patterns in a DCR-graph (use the simulator to verify if your model is right):

1. **Direct Precedence of a Task.** “Every time B occurs, it should be directly preceded by A.” If B occurs without a directly preceding A, the rule is violated. For instance, traces [`ACCAAC`](#) and [`ABCAAB`](#) comply to the rule, whereas [`ABACB`](#) violates the rule.
2. **Direct Precedence or Simultaneous Occurrences of Tasks.** “Task A must always be executed simultaneously or directly before task B.” (hint: consider A/B as non-atomic tasks)
3. **Bounded Existence of a Task:** Task A should be executed exactly k times.” If A occurs less than or more than k times, the rule is violated. For instance, for k = 2, the trace [`BCADBCAD`](#) complies to this rule and [`BCADBCAAB`](#) violates the rule.
4. **Execution in Between.** “Task B should be performed not before task A has been executed, and not later than C.”

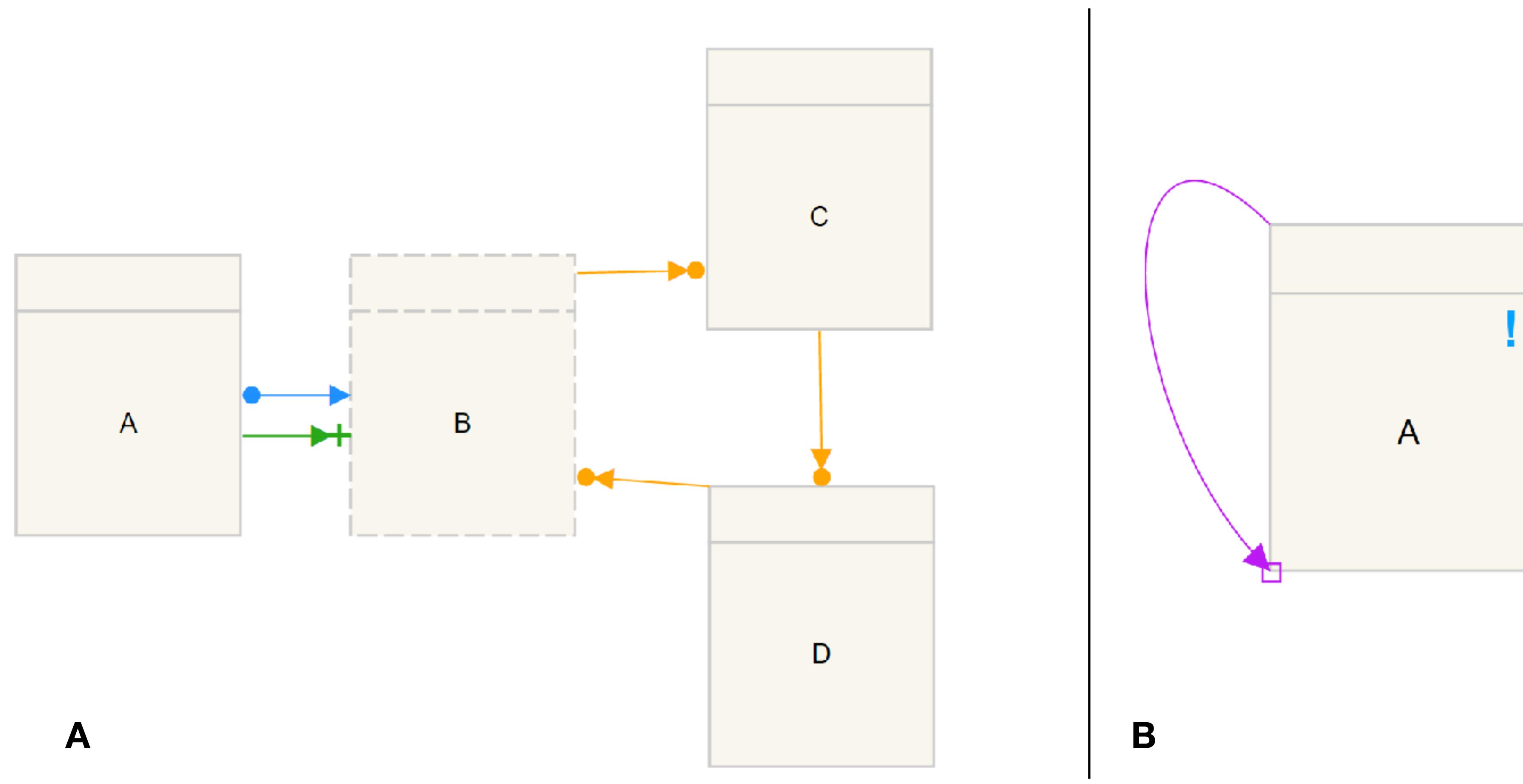
Exercises (Soundness)

1. Discuss in pairs whether the following processes are sound:



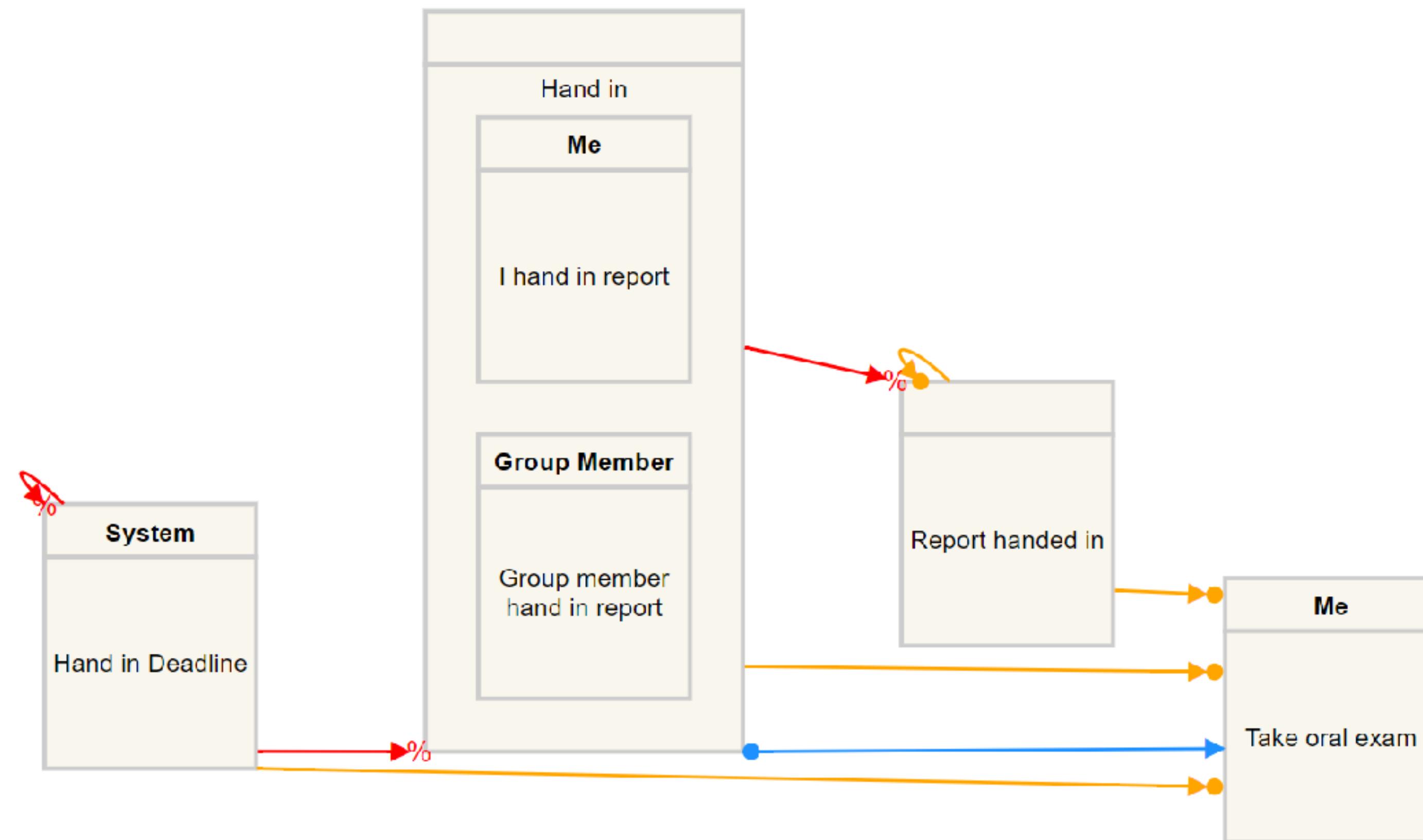
Exercise:

2. Argument whether the following processes are sound:



Exercise

3. Argument whether this process is sound:



Declarative Process Modelling: Advanced Operators and Compliance

Hugo A. López

hulo@dtu.dk

DTU Compute

Department of Applied Mathematics and Computer Science

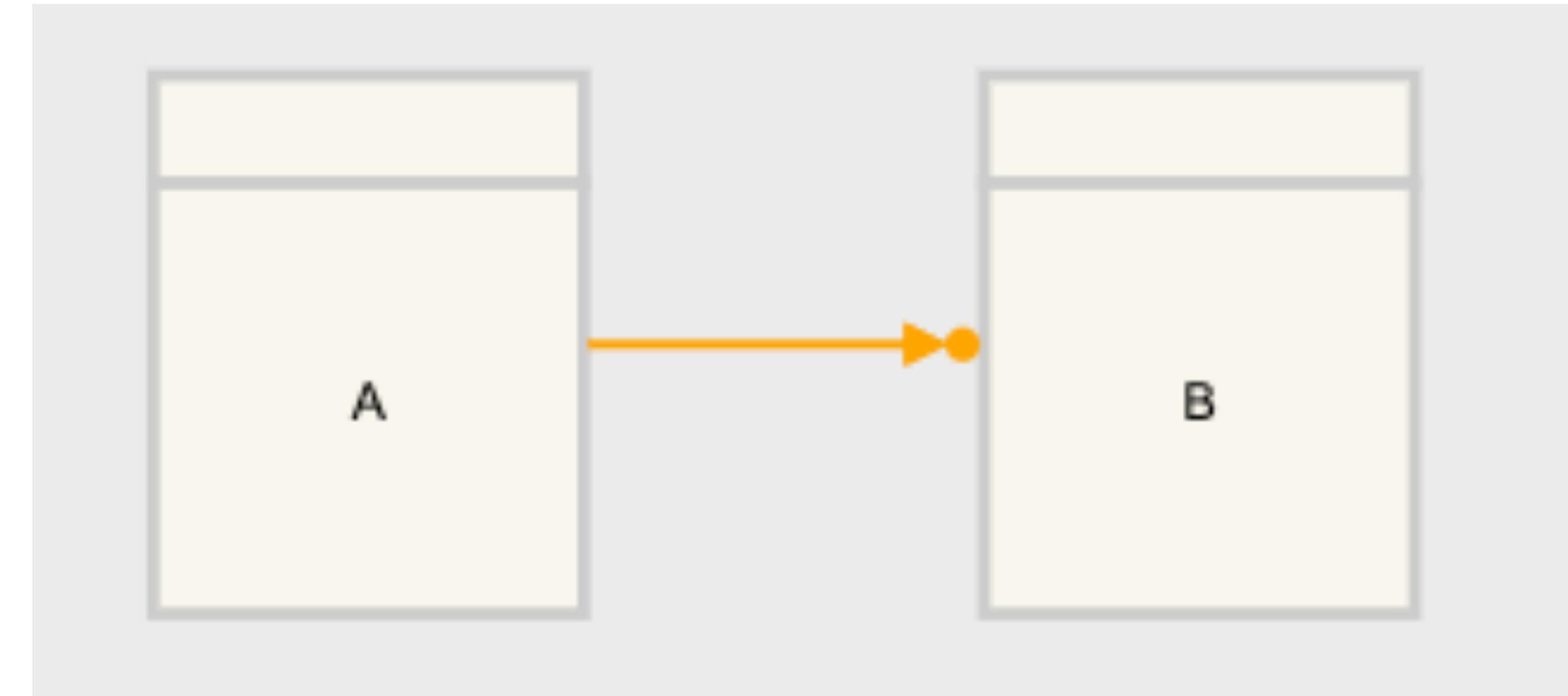
$$f(x+\Delta x) = \sum_{i=0}^{\infty} \frac{(\Delta x)^i}{i!} f^{(i)}(x)$$
$$\int_a^b \Theta + \Omega \int_0^{\infty} \delta / e^{i\pi} =$$
$$\sqrt{17} \sum_{!} \Sigma \gg \epsilon$$
$$\infty \rightarrow \{2.7182818284\}$$
$$\chi^2$$

Agenda

- Advanced DCR Constructs (time, data, subprocesses)
- Implementing DCR graphs
- Compliance Checking

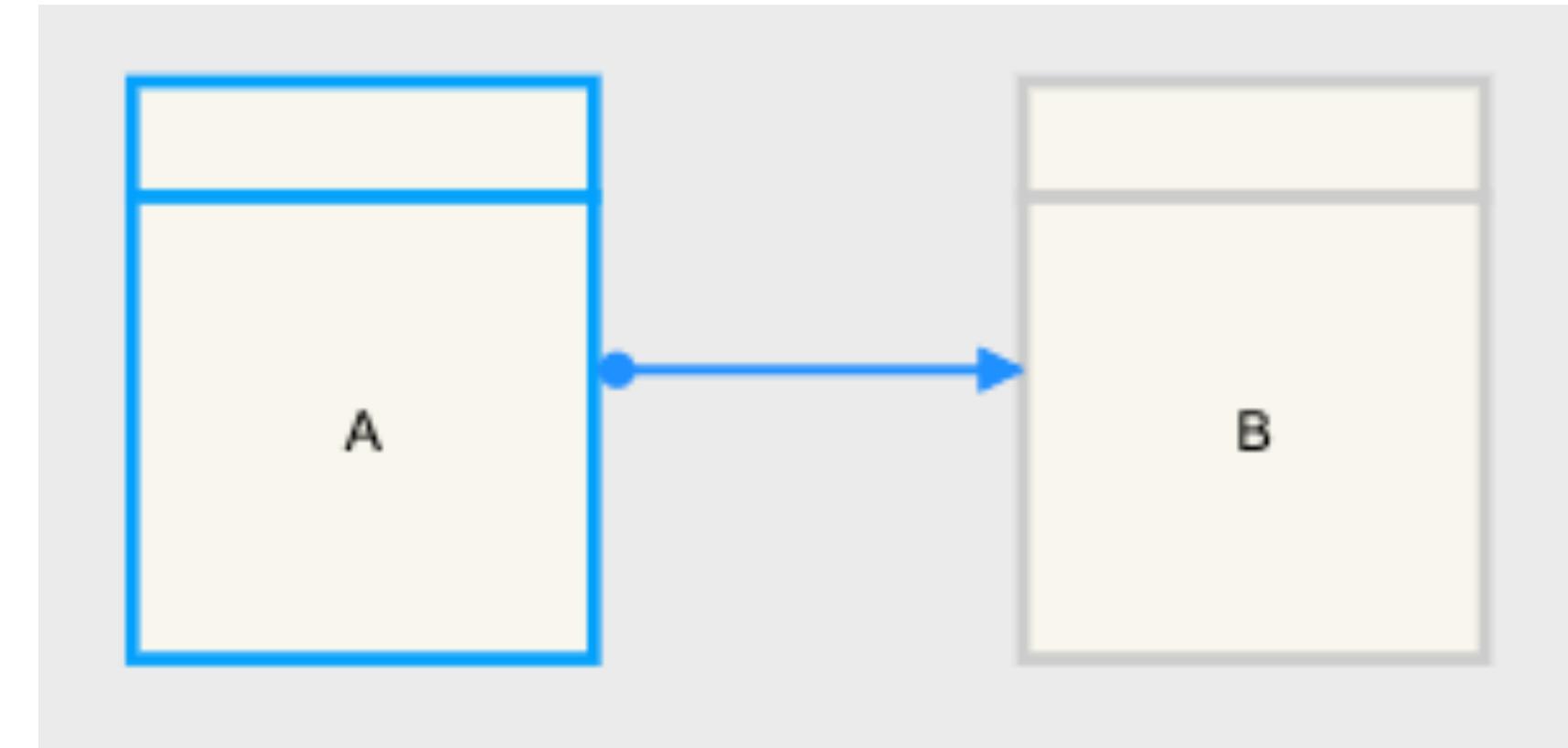
Basic Constructs

REFRESH



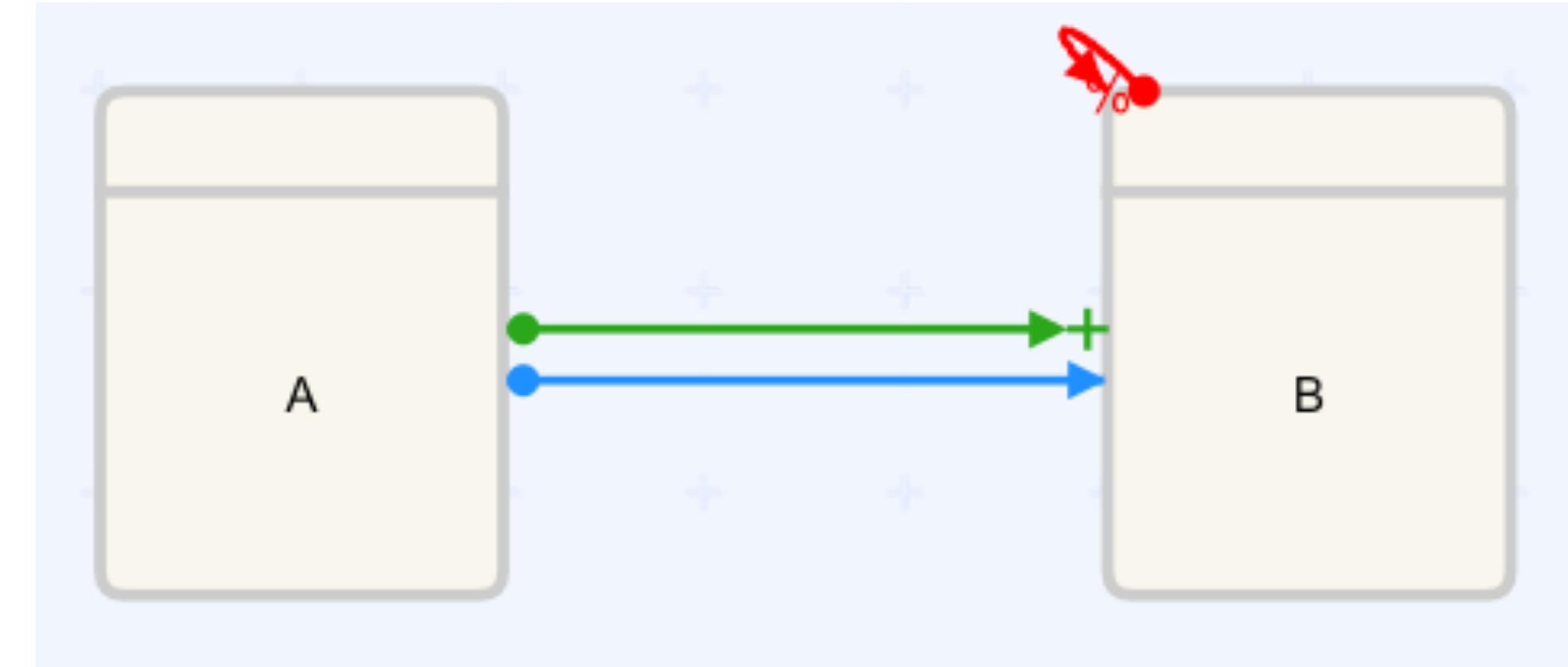
- I use a condition to define precedence between events
 - B can't be done before A
- I use a condition to provide rights
 - B may be executed if A is executed

Responses



- You impose obligations that need to be held before the trace ends
 - B must be done every time A happens
 - B can also be executed before A

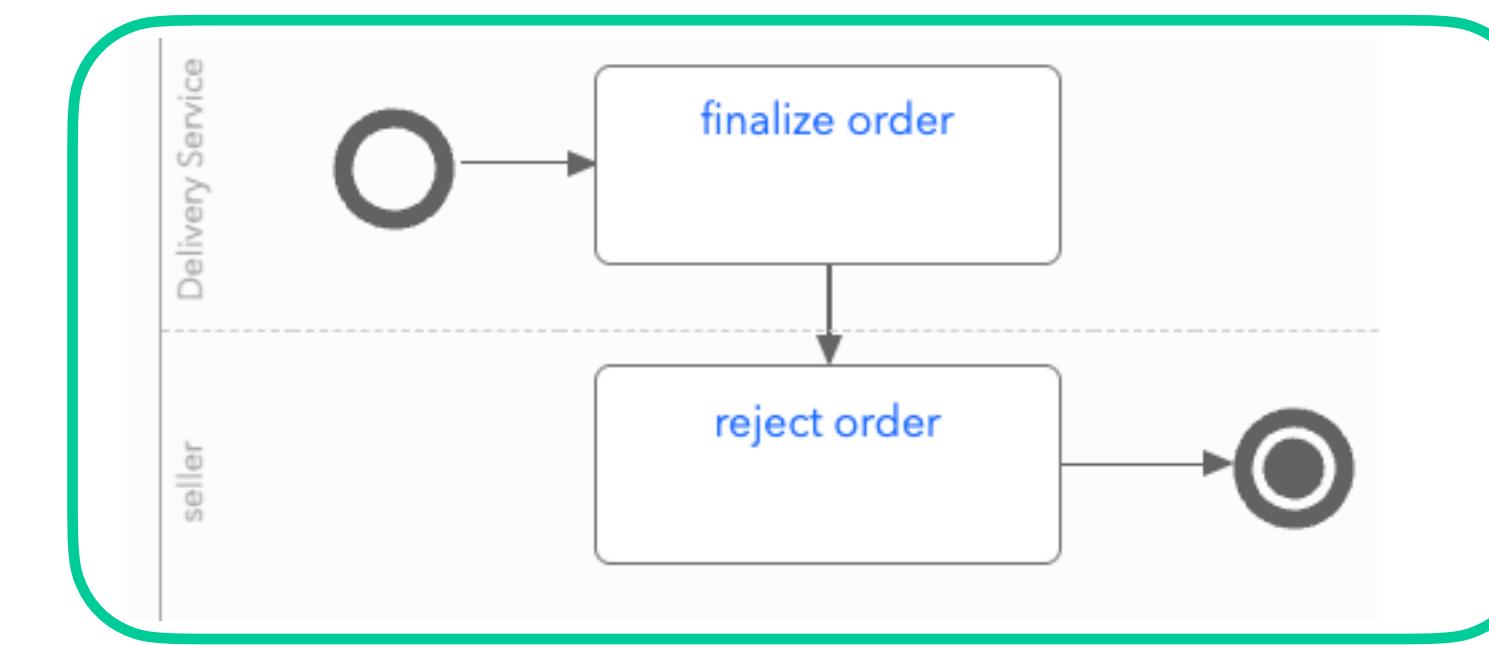
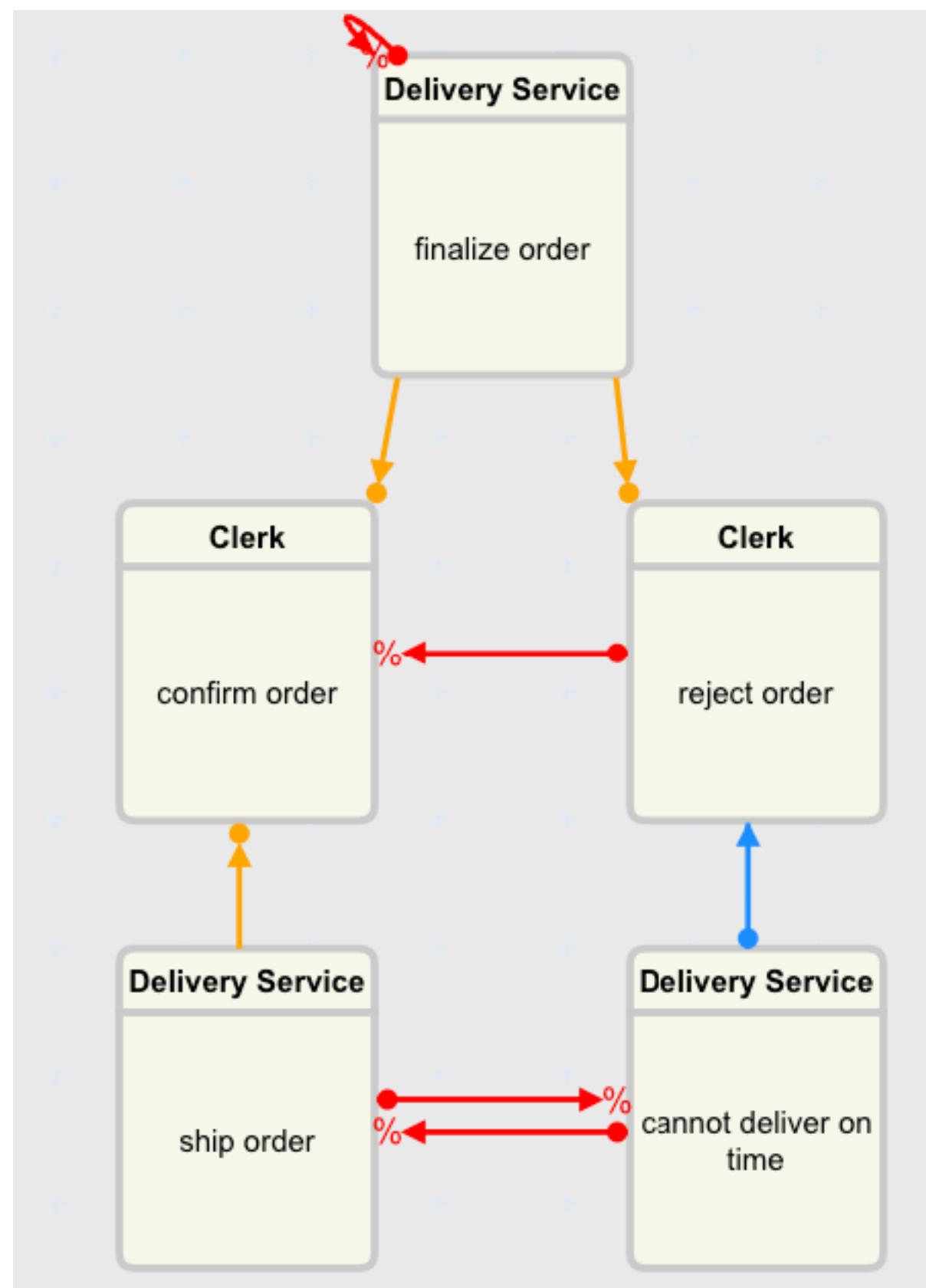
Exclusions / Inclusions



- Inclusions/exclusions modify the included marking of an activity
 - The pending/executed marking does not count when an activity is excluded
 - They can be reincluded upon the execution of other activities

Testing DCR graphs

- We can ensure our model lives up to the specification by verifying models recognize a set of tests



Required trace = must be replayed

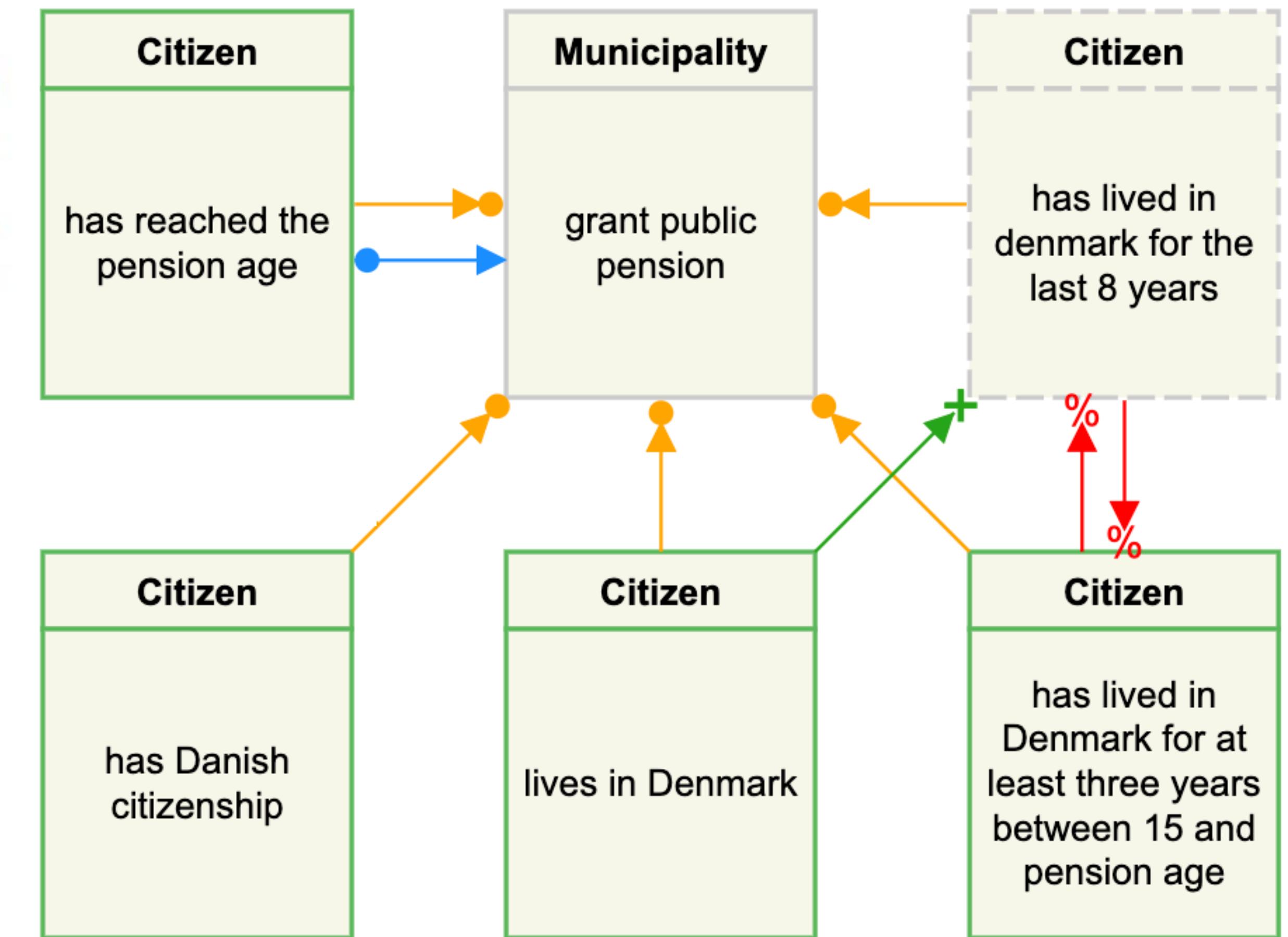


Forbidden trace = should not be accepted

DCR graphs

Definition 4.1 (DCR Graph [8]). A DCR graph is a tuple (E, R, M) where

- E is a finite set of (labelled) events, the nodes of the graph.
- R is the edges of the graph. Edges are partitioned into five kinds, named and drawn as follows: The conditions ($\rightarrow\bullet$), responses ($\bullet\rightarrow$), milestones ($\rightarrow\diamond$), inclusions ($\rightarrow+$), and exclusions ($\rightarrow\%$).
- M is the marking of the graph. This is a triple (Ex, Re, In) of sets of events, respectively the previously executed (Ex), the currently pending (Re), and the currently included (In) events.



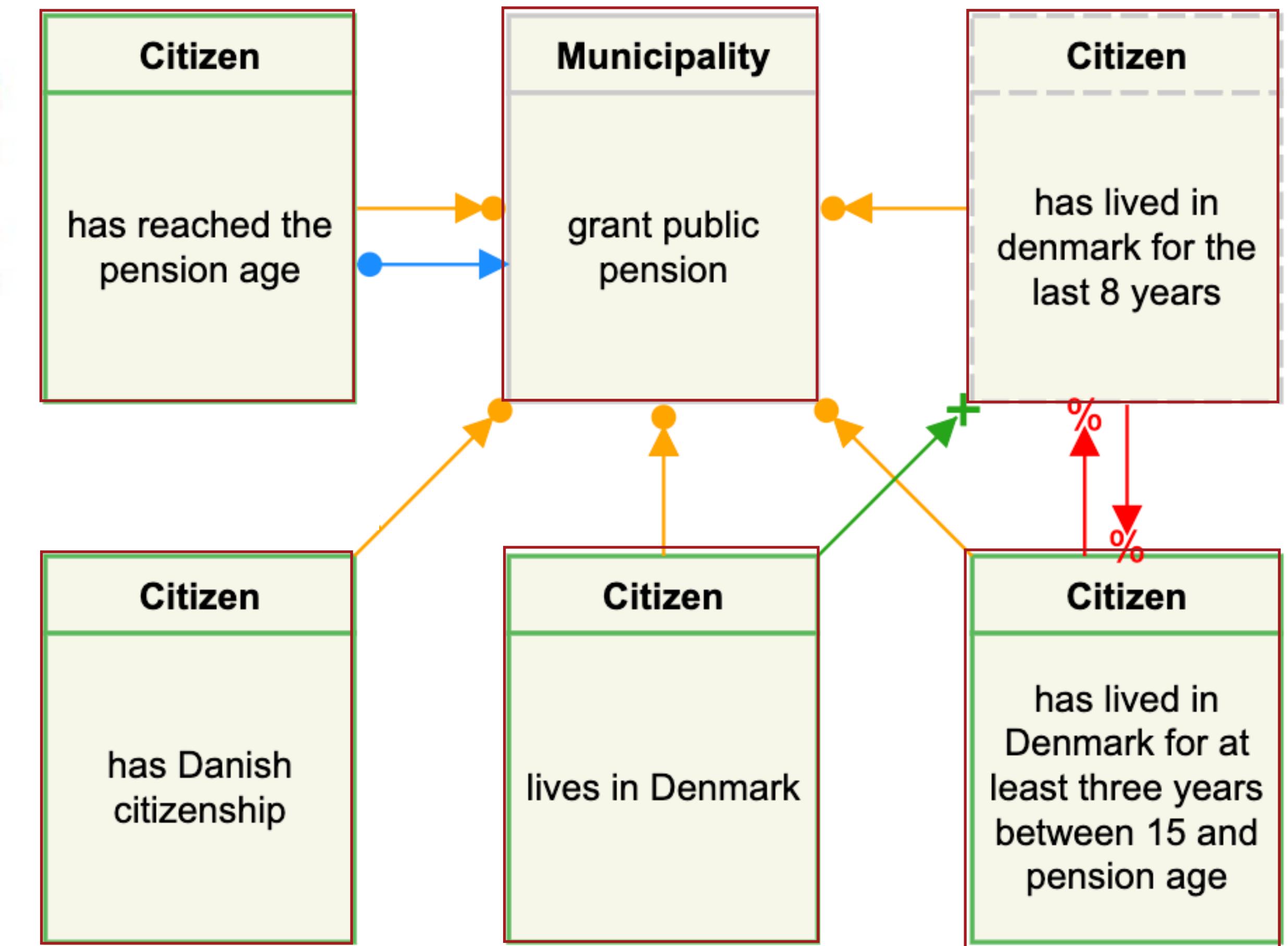
Definitions from [8] Søren Debois, Thomas T. Hildebrandt, Tijs Slaats:

Hierarchical Declarative Modelling with Refinement and Sub-processes. BPM 2014: 18-33

DCR graphs

Definition 4.1 (DCR Graph [8]). A DCR graph is a tuple (E, R, M) where

- E is a finite set of (labelled) events, the nodes of the graph.
- R is the edges of the graph. Edges are partitioned into five kinds, named and drawn as follows: The conditions ($\rightarrow\bullet$), responses ($\bullet\rightarrow$), milestones ($\rightarrow\diamond$), inclusions ($\rightarrow+$), and exclusions ($\rightarrow\%$).
- M is the marking of the graph. This is a triple (Ex, Re, In) of sets of events, respectively the previously executed (Ex), the currently pending (Re), and the currently included (In) events.



Definitions from [8] Søren Debois, Thomas T. Hildebrandt, Tijs Slaats:

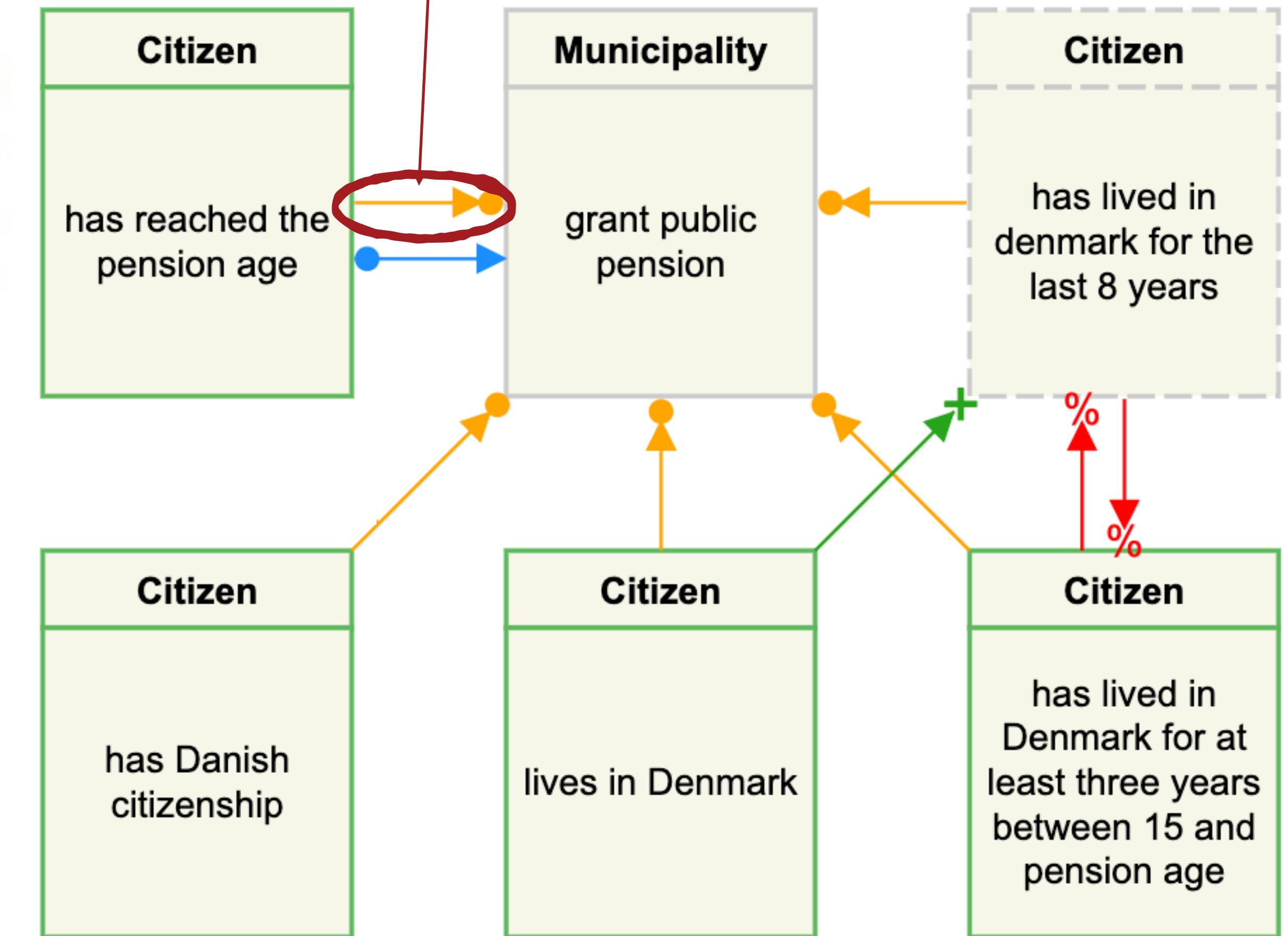
Hierarchical Declarative Modelling with Refinement and Sub-processes. BPM 2014: 18-33

DCR graphs

Condition
relation

Definition 4.1 (DCR Graph [8]). A DCR graph is a tuple (E, R, M) where

- E is a finite set of (labelled) events, the nodes of the graph.
- R is the edges of the graph. Edges are partitioned into five kinds, named and drawn as follows: The conditions ($\rightarrow\bullet$), responses ($\bullet\rightarrow$), milestones ($\rightarrow\diamond$), inclusions ($\rightarrow+$), and exclusions ($\rightarrow\%$).
- M is the marking of the graph. This is a triple (Ex, Re, In) of sets of events, respectively the previously executed (Ex), the currently pending (Re), and the currently included (In) events.



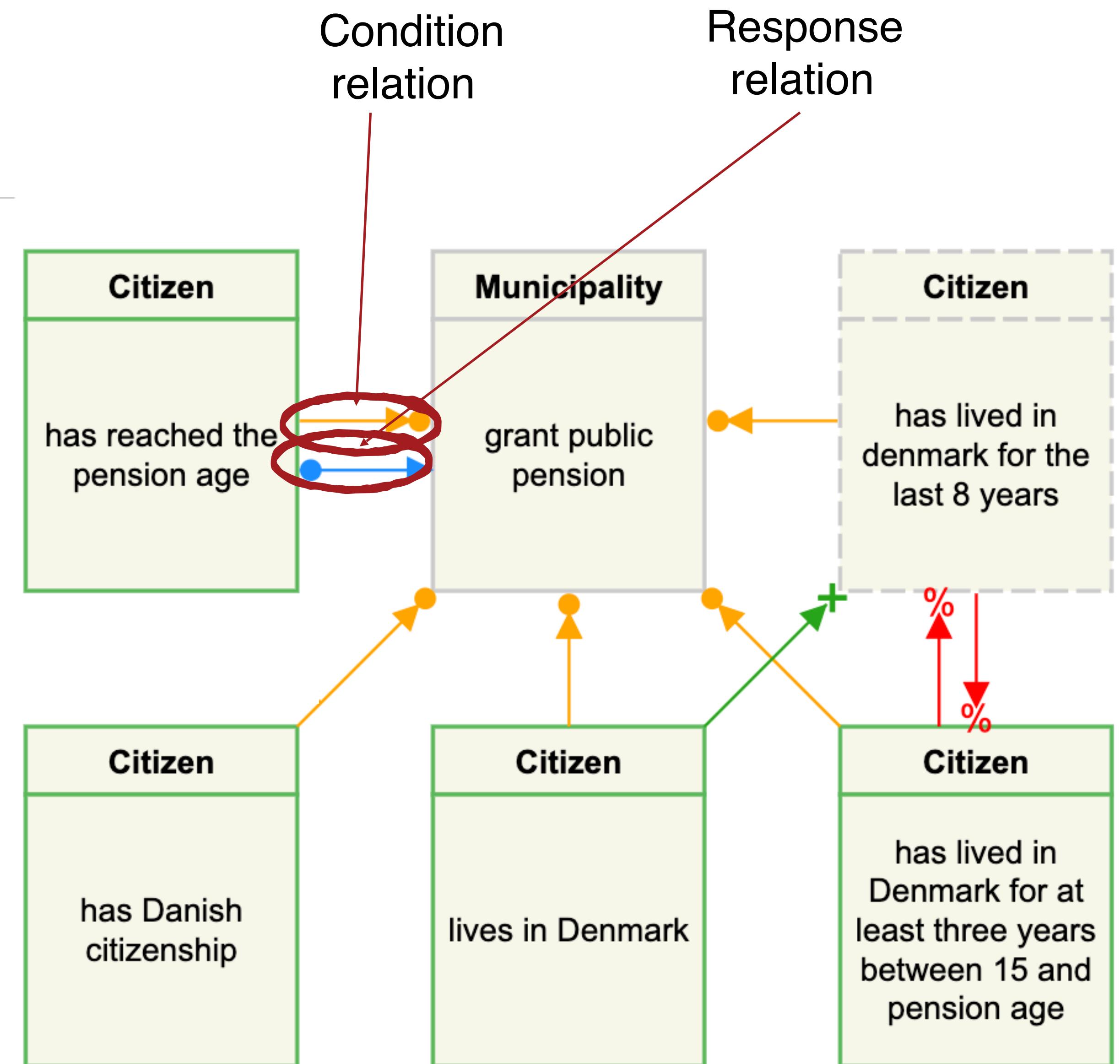
Definitions from [8] Søren Debois, Thomas T. Hildebrandt, Tijs Slaats:

Hierarchical Declarative Modelling with Refinement and Sub-processes. BPM 2014: 18-33

DCR graphs

Definition 4.1 (DCR Graph [8]). A DCR graph is a tuple (E, R, M) where

- E is a finite set of (labelled) events, the nodes of the graph.
- R is the edges of the graph. Edges are partitioned into five kinds, named and drawn as follows: The conditions ($\rightarrow\bullet$), responses ($\bullet\rightarrow$), milestones ($\rightarrow\diamond$), inclusions ($\rightarrow+$), and exclusions ($\rightarrow\%$).
- M is the marking of the graph. This is a triple (Ex, Re, In) of sets of events, respectively the previously executed (Ex), the currently pending (Re), and the currently included (In) events.



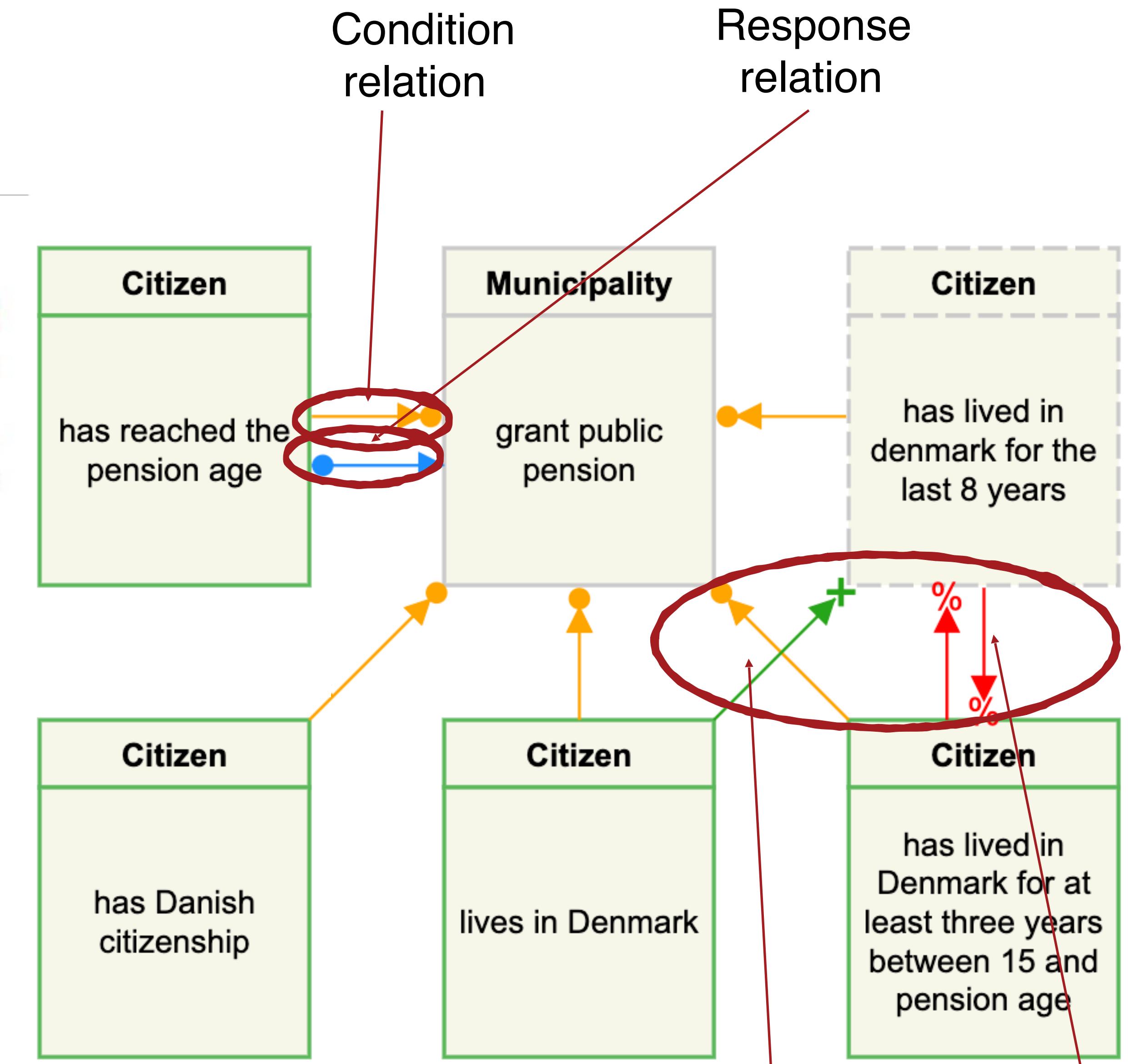
Definitions from [8] Søren Debois, Thomas T. Hildebrandt, Tijs Slaats:

Hierarchical Declarative Modelling with Refinement and Sub-processes. BPM 2014: 18-33

DCR graphs

Definition 4.1 (DCR Graph [8]). A DCR graph is a tuple (E, R, M) where

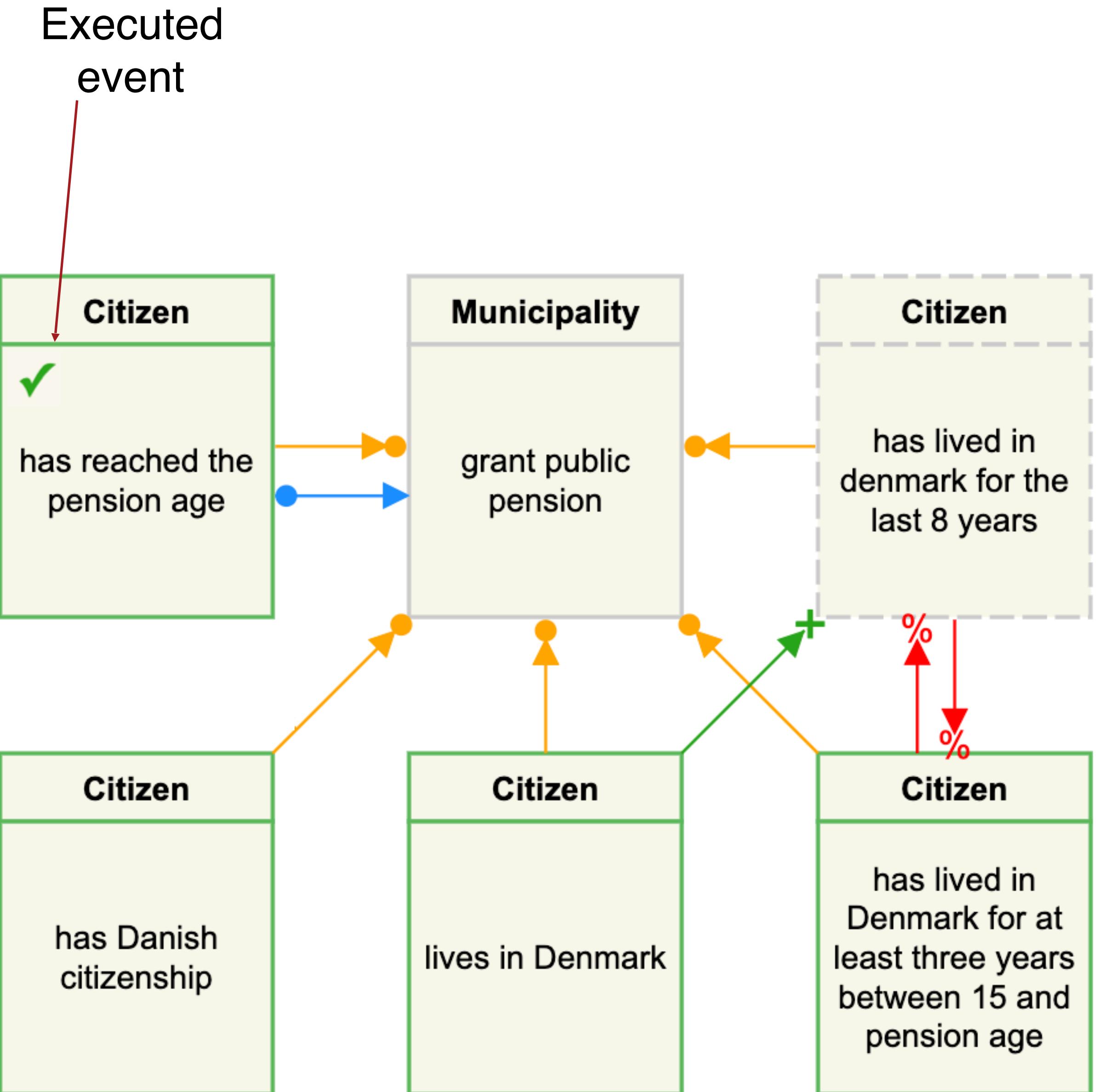
- E is a finite set of (labelled) events, the nodes of the graph.
- R is the edges of the graph. Edges are partitioned into five kinds, named and drawn as follows: The conditions ($\rightarrow\bullet$), responses ($\bullet\rightarrow$), milestones ($\rightarrow\diamond$), inclusions ($\rightarrow+$), and exclusions ($\rightarrow\%$).
- M is the marking of the graph. This is a triple (Ex, Re, In) of sets of events, respectively the previously executed (Ex), the currently pending (Re), and the currently included (In) events.



DCR graphs

Definition 4.1 (DCR Graph [8]). A DCR graph is a tuple (E, R, M) where

- E is a finite set of (labelled) events, the nodes of the graph.
- R is the edges of the graph. Edges are partitioned into five kinds, named and drawn as follows: The conditions ($\rightarrow\bullet$), responses ($\bullet\rightarrow$), milestones ($\rightarrow\diamond$), inclusions ($\rightarrow+$), and exclusions ($\rightarrow\%$).
- M is the marking of the graph. This is a triple (Ex, Re, In) of sets of events, respectively the previously executed (Ex), the currently pending (Re), and the currently included (In) events.



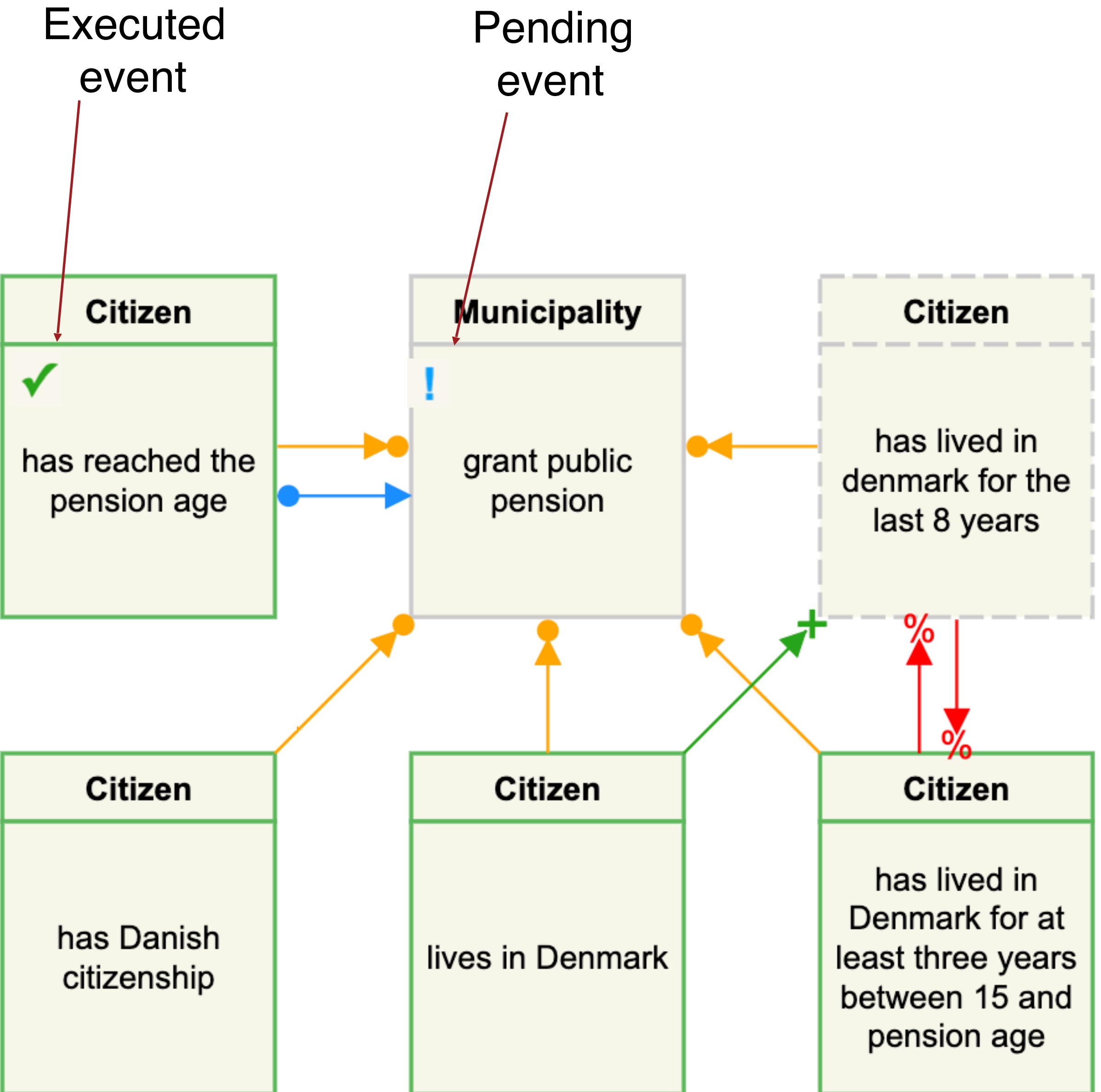
Definitions from [8] Søren Debois, Thomas T. Hildebrandt, Tijs Slaats:

Hierarchical Declarative Modelling with Refinement and Sub-processes. BPM 2014: 18-33

DCR graphs

Definition 4.1 (DCR Graph [8]). A DCR graph is a tuple (E, R, M) where

- E is a finite set of (labelled) events, the nodes of the graph.
- R is the edges of the graph. Edges are partitioned into five kinds, named and drawn as follows: The conditions ($\rightarrow\bullet$), responses ($\bullet\rightarrow$), milestones ($\rightarrow\diamond$), inclusions ($\rightarrow+$), and exclusions ($\rightarrow\%$).
- M is the marking of the graph. This is a triple (Ex, Re, In) of sets of events, respectively the previously executed (Ex), the currently pending (Re), and the currently included (In) events.



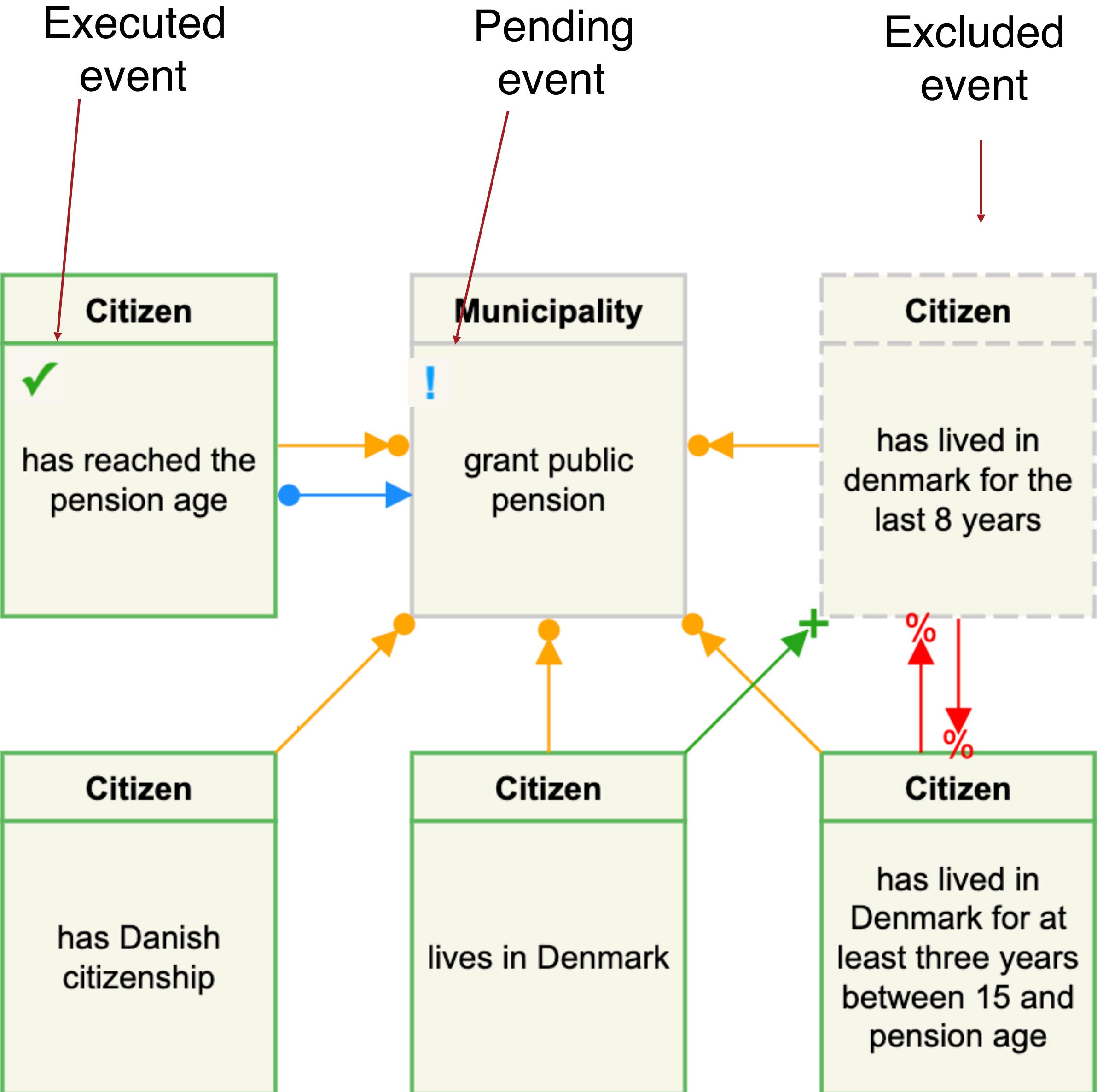
Definitions from [8] Søren Debois, Thomas T. Hildebrandt, Tijs Slaats:

Hierarchical Declarative Modelling with Refinement and Sub-processes. BPM 2014: 18-33

DCR graphs

Definition 4.1 (DCR Graph [8]). A DCR graph is a tuple (E, R, M) where

- E is a finite set of (labelled) events, the nodes of the graph.
- R is the edges of the graph. Edges are partitioned into five kinds, named and drawn as follows: The conditions ($\rightarrow\bullet$), responses ($\bullet\rightarrow$), milestones ($\rightarrow\diamond$), inclusions ($\rightarrow+$), and exclusions ($\rightarrow\%$).
- M is the marking of the graph. This is a triple (Ex, Re, In) of sets of events, respectively the previously executed (Ex), the currently pending (Re), and the currently included (In) events.



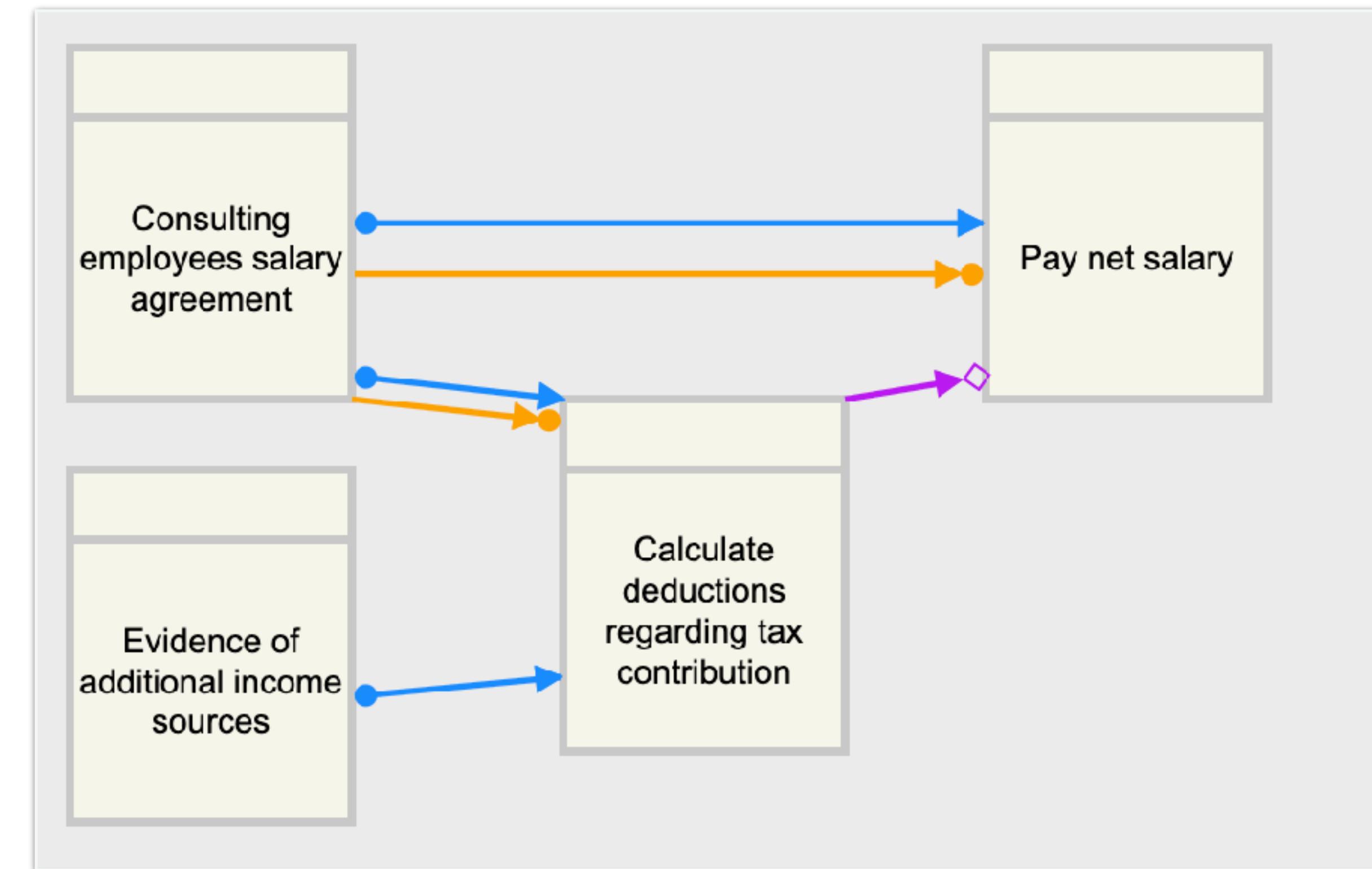
Definitions from [8] Søren Debois, Thomas T. Hildebrandt, Tijs Slaats:

Hierarchical Declarative Modelling with Refinement and Sub-processes. BPM 2014: 18-33

Milestone

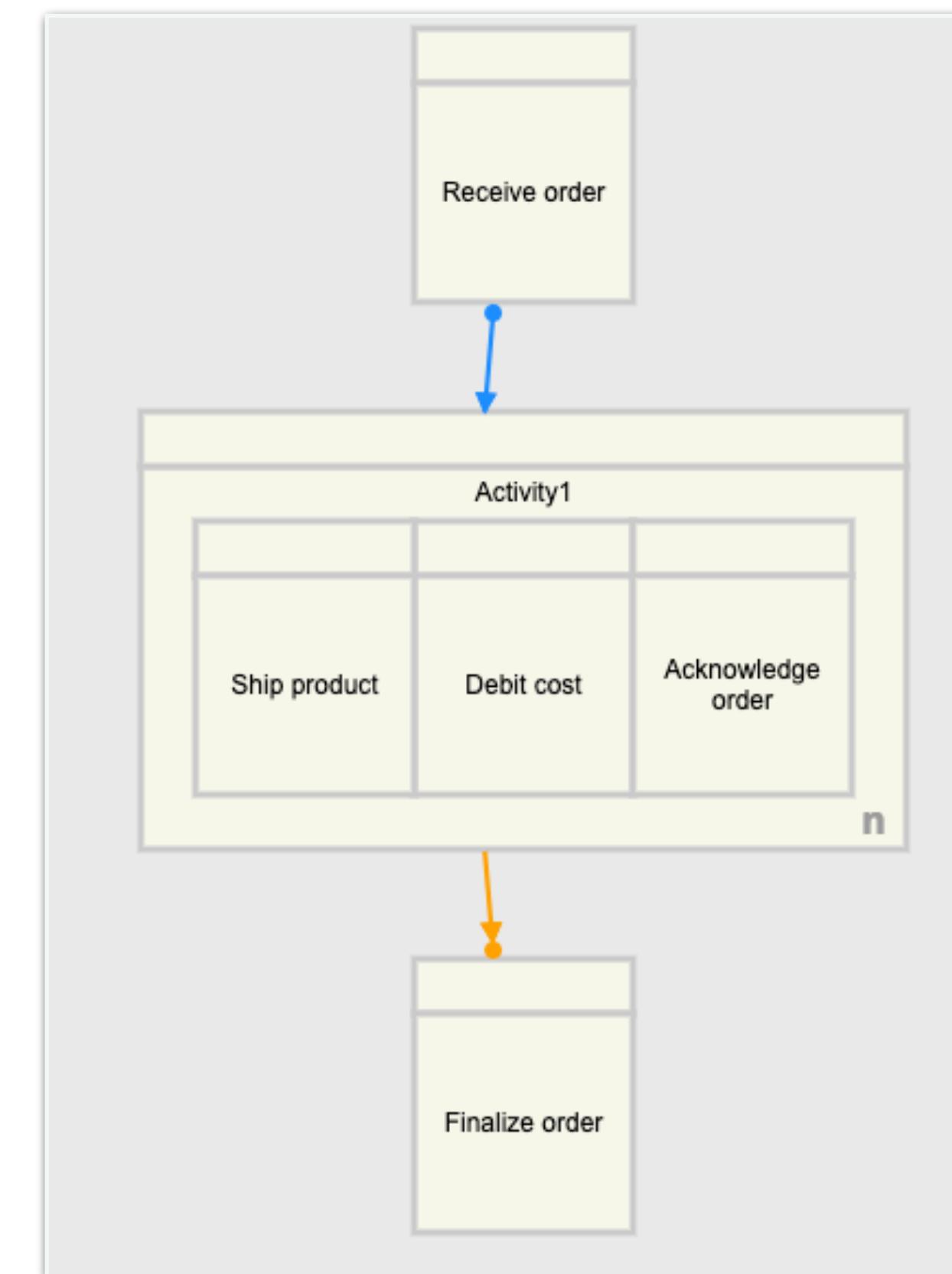
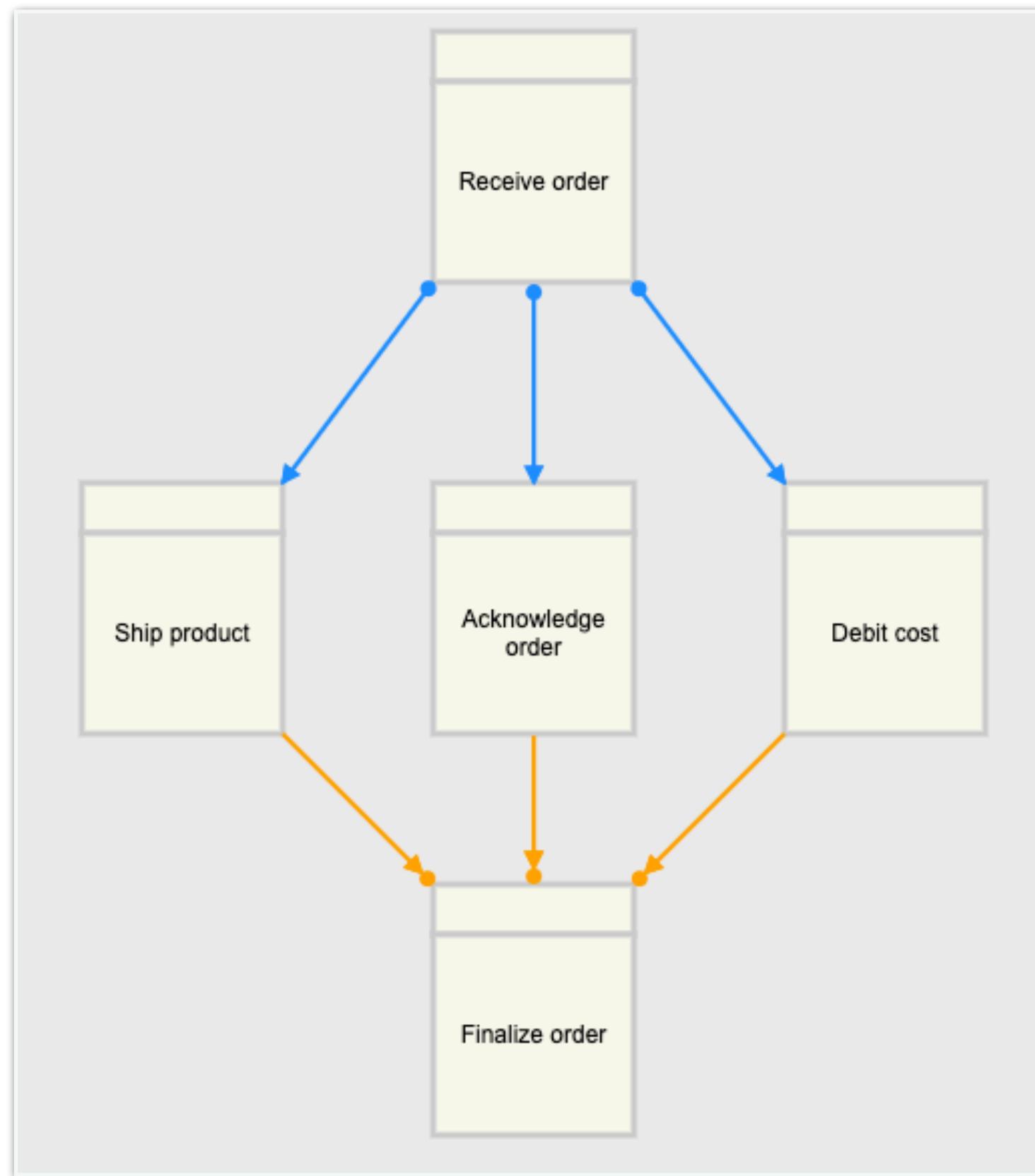
- Ensure that follow-up actions are performed.
- If an event A is a milestone for an event B, written A →◊ B, then B can not happen if A is included and required to be executed again (i.e. as a response).

The payment of the net salary can be performed only after consulting employees salary agreement with human resources. The payment will be done only after having performed the deductions regarding tax contributions for the employee, and it will be based current employees' salary agreement. If there has been additional income sources (i.e.: bonuses), this should be considered in the calculation of tax contribution.



Nesting

- More like a syntactic sugar, it decreases the number of connections (arrows) without affecting its semantics.
- Whatever relation is applied to a nesting, it is applied to all nested activities.



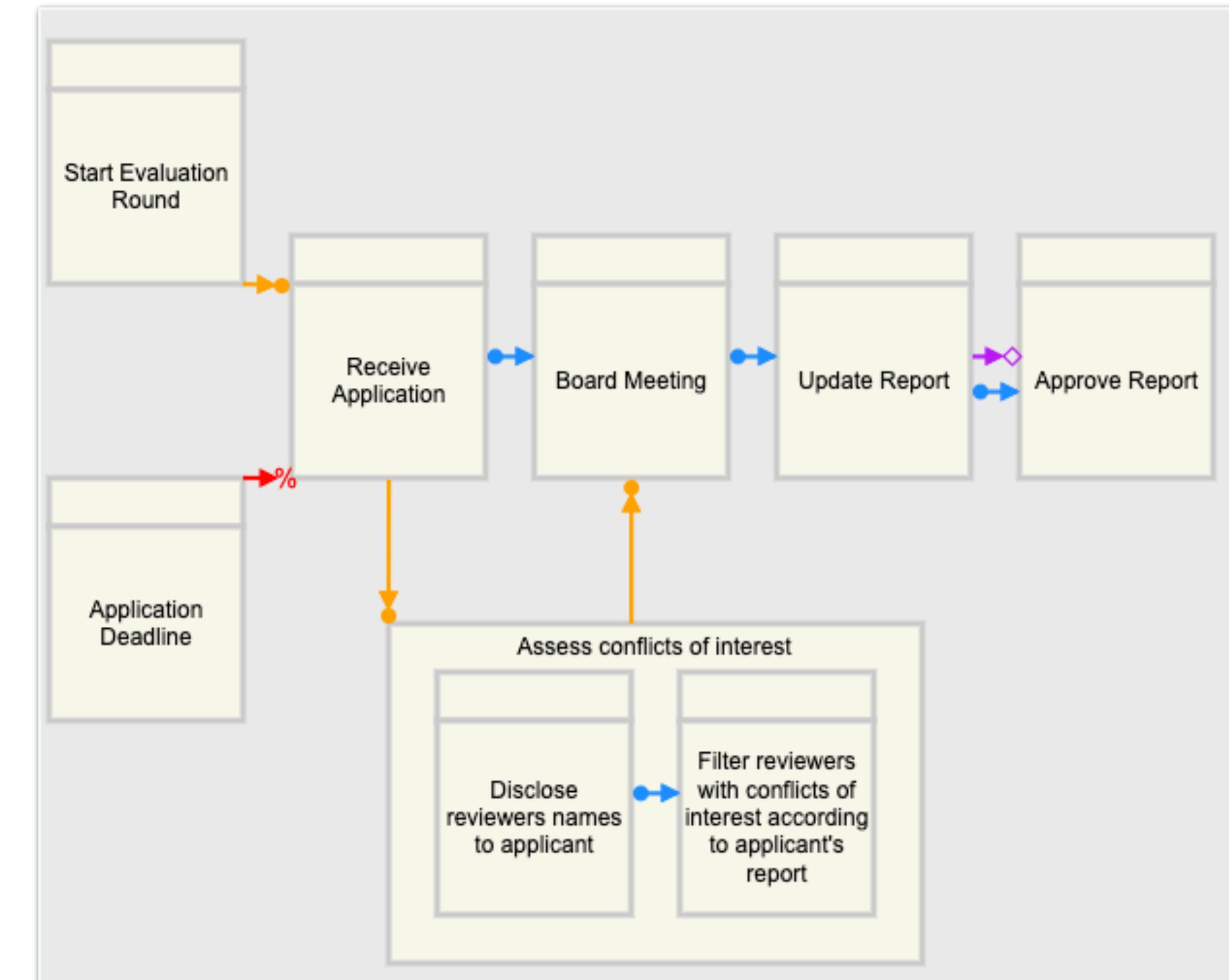
Hildebrandt, T., Mukkamala, R. R., & Slaats, T. (2011, April). Nested dynamic condition response graphs. In *International conference on fundamentals of software engineering* (pp. 343-350). Berlin, Heidelberg: Springer Berlin Heidelberg.

Subprocesses

- A graph is a process!
- A sub-process is a complex activity that has underlying behavior that is instantiated when the sub-process is started and closed when the sub-process ends.
- Sub-processes can be:
 - **single-instance:** only one instance of the sub-process will be active at any time, or
 - **multi-instance:** multiple instances of the sub-processes can execute concurrently.
- Subprocesses have their own marking, while nestings do not
- Formal definitions in Debois, S., Hildebrandt, T. T., & Slaats, T. (2018). Replication, refinement & reachability: complexity in dynamic condition-response graphs. *Acta Informatica*, 55(6), 489-520

Single Instance Subprocess

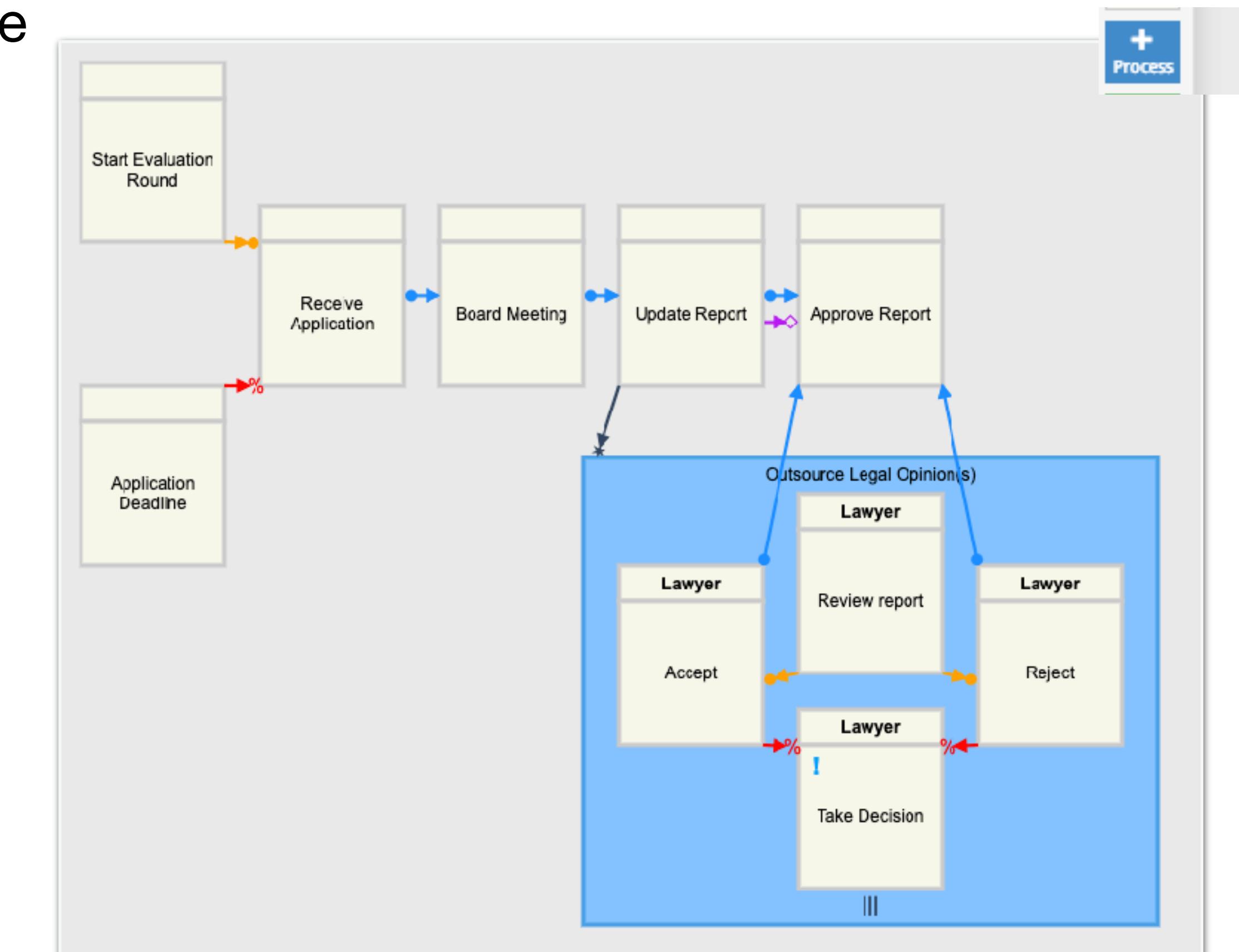
- The board needs to consider conflict of interest in its applications at least once. Further assessments are not necessary.



<http://www.dcrgraphs.net/Tool?id=9370#>

Multiple Instance Subprocesses

- The review might include one or more legal advisors, and their decision will influence the approval of the report.

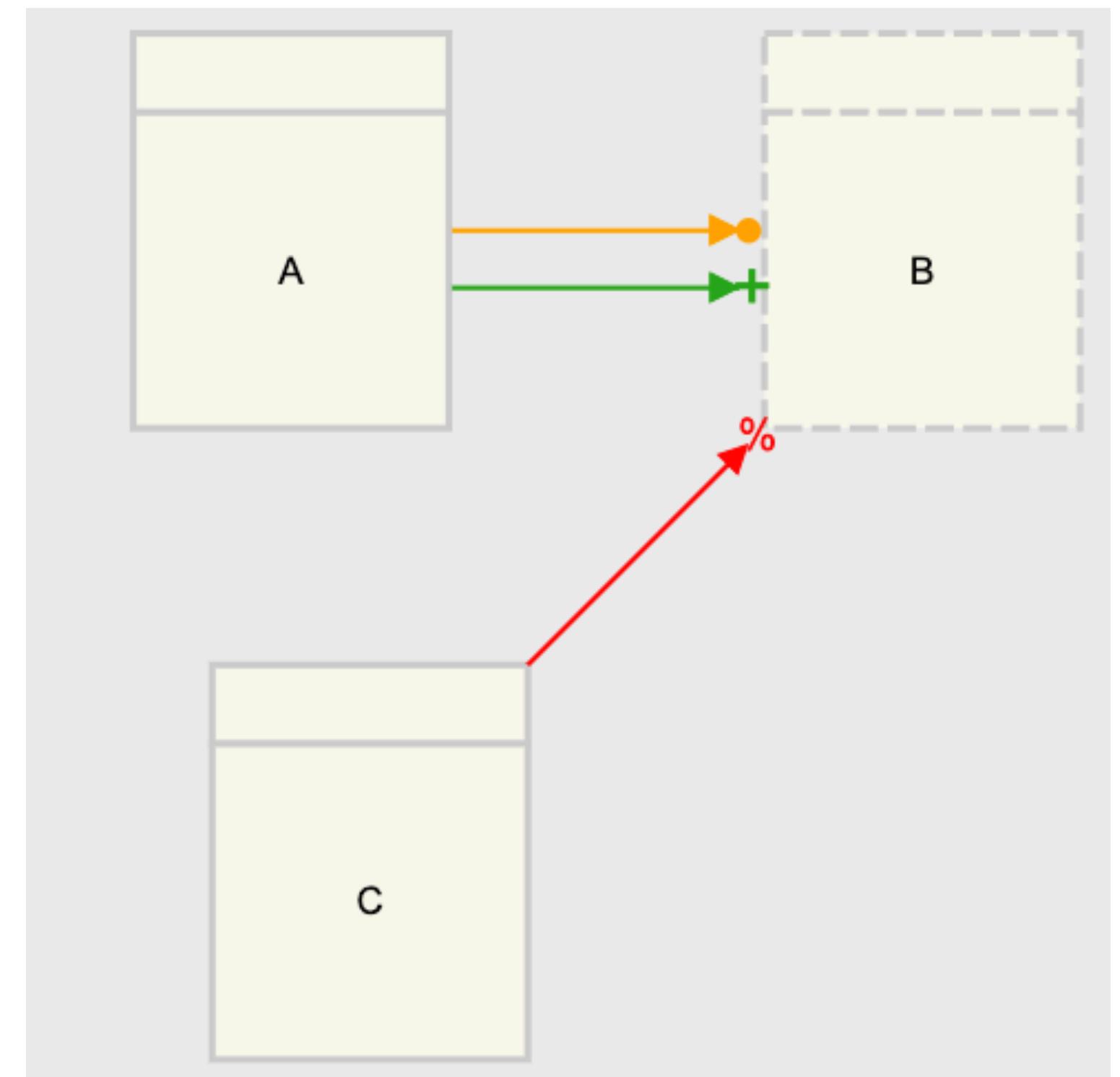


<http://www.dcrgraphs.net/Tool?id=9375#>

Last week's exercises

Model the following patterns in a DCR-graph

- 1. Direct Precedence of a Task.** “Every time B occurs, it should be directly preceded by A.” If B occurs without a directly preceding A, the rule is violated. For instance, traces [\$\langle \text{ACCAAC} \rangle\$](#) and [\$\langle \text{ABCAAB} \rangle\$](#) comply to the rule, whereas [\$\langle \text{ABACB} \rangle\$](#) violates the rule.

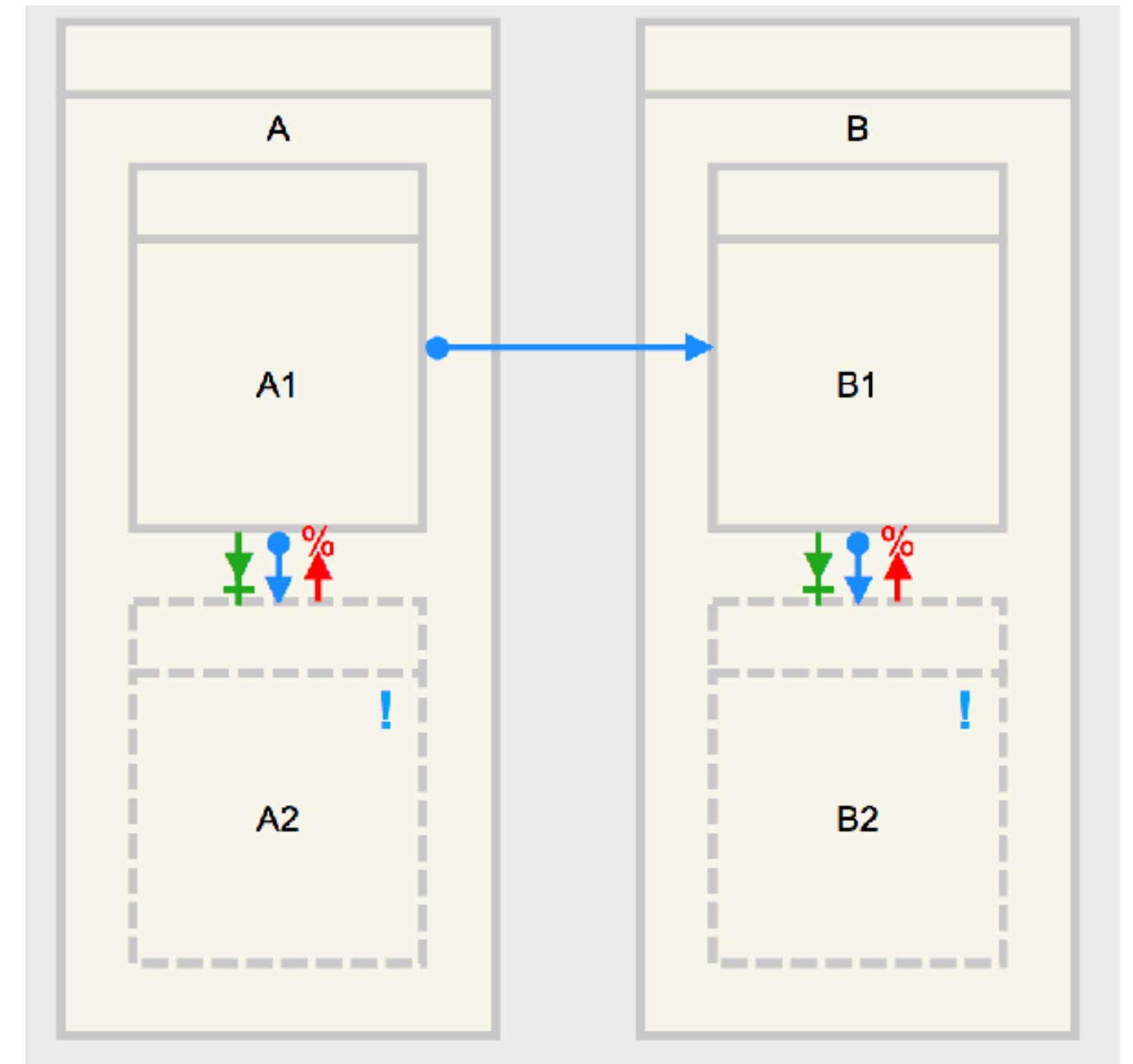


Last week's exercises

Model the following patterns in a DCR-graph:

2. Direct Precedence or Simultaneous

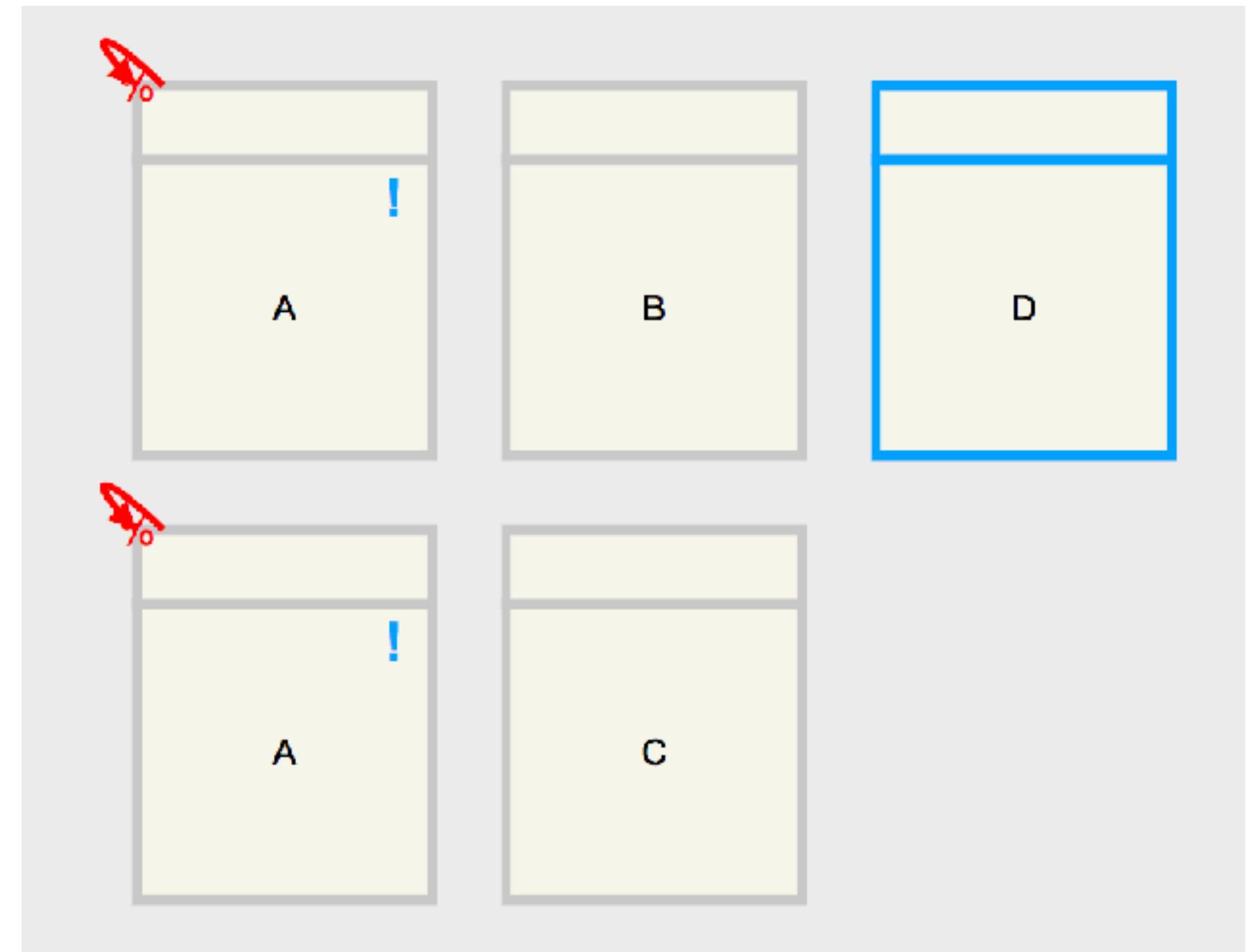
Occurrences of Tasks. “Task A must always be executed simultaneously or directly before task B.” (hint: consider A/B as non-atomic tasks)



Last week's exercises

Model the following patterns in a DCR-graph:

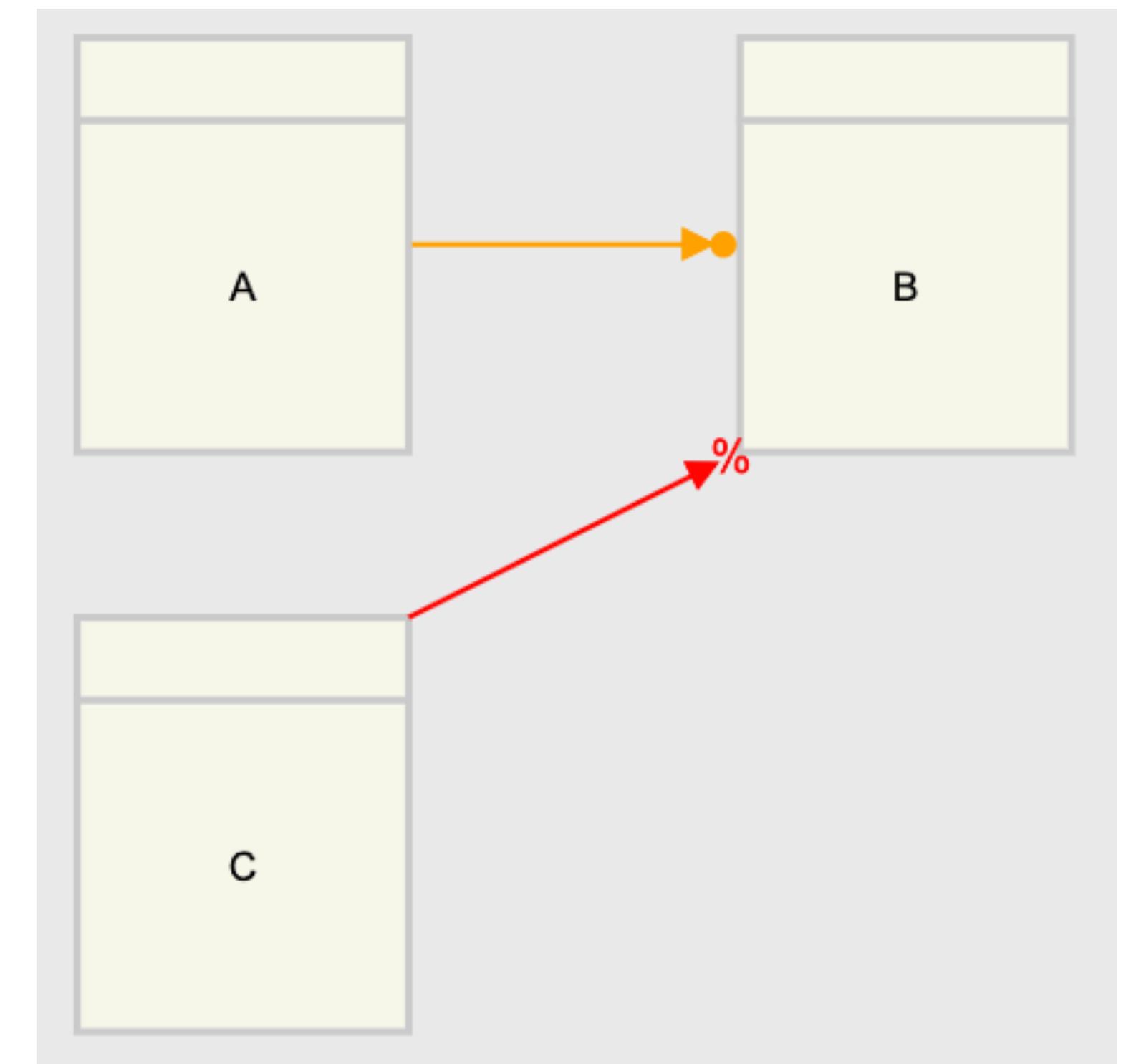
3. **Bounded Existence of a Task:** Task A should be executed exactly k times." If A occurs less than or more than k times, the rule is violated. For instance, for k = 2, the trace **<BCADBCAD>** complies to this rule and **<BCADBCAADD>** violates the rule.



Last week's exercises

Model the following patterns in a DCR-graph:

4. **Execution in Between.** “Task B should be performed not before task A has been executed, and not later than C.”

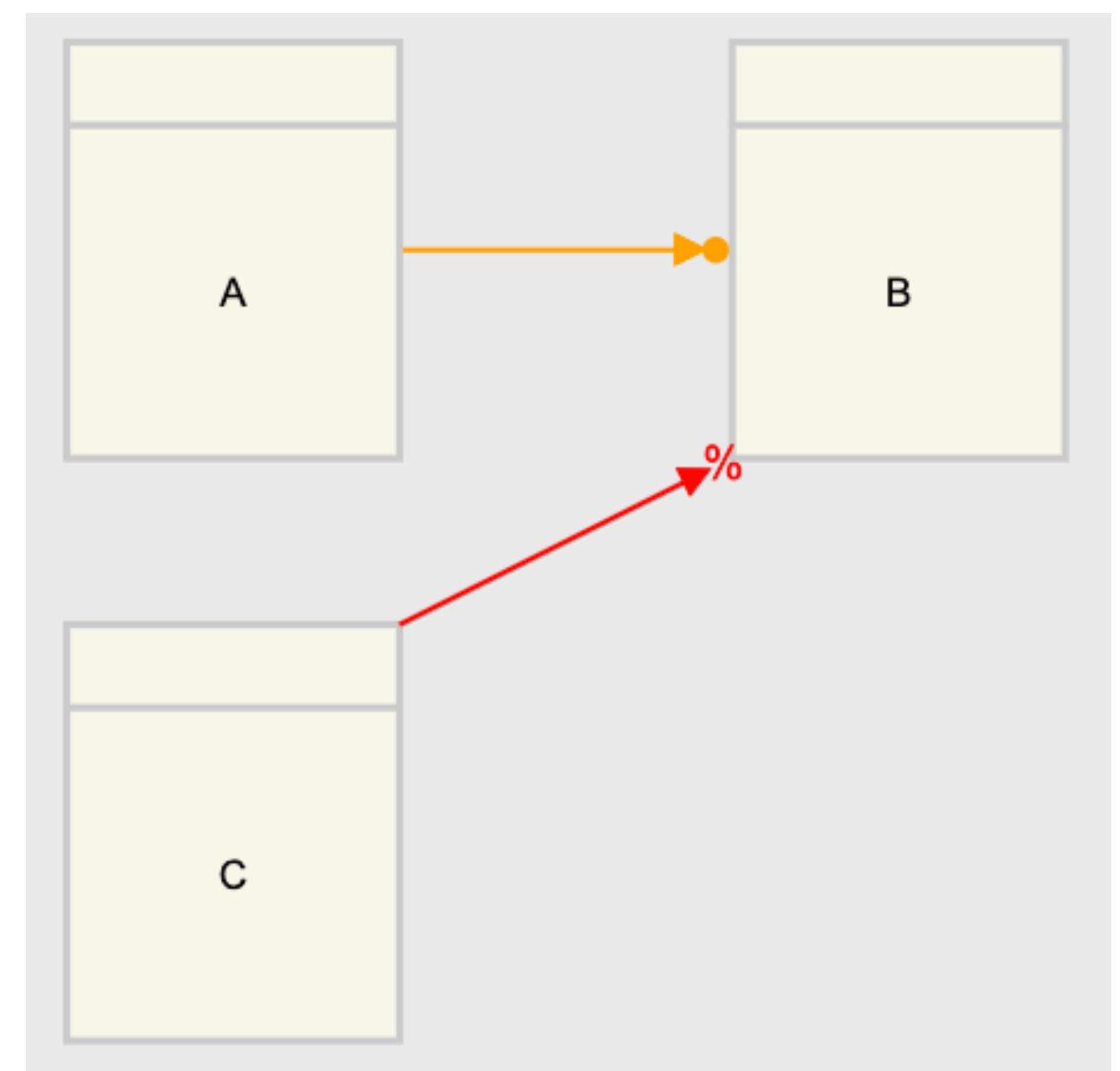


Last week's exercises

Model the following patterns in a DCR-graph:

4. **Execution in Between.** “Task B should be performed not before task A has been executed, and not later than C.”

Can you modify the graph so it allows multiple executions of B if A occurs?



IMPLEMENTING DCR GRAPHS

Implementing DCR graphs (I)

Definition 4.1 (DCR Graph [8]). A DCR graph is a tuple (E, R, M) where

- E is a finite set of (labelled) events, the nodes of the graph.
- R is the edges of the graph. Edges are partitioned into five kinds, named and drawn as follows: The conditions ($\rightarrow\bullet$), responses ($\bullet\rightarrow$), milestones ($\rightarrow\diamond$), inclusions ($\rightarrow+$), and exclusions ($\rightarrow\%$).
- M is the marking of the graph. This is a triple (Ex, Re, In) of sets of events, respectively the previously executed (Ex), the currently pending (Re), and the currently included (In) events.

```
public class DCRGraph {  
  
    // Events  
  
    protected HashSet<String> events = new HashSet<String>();  
  
    // Relations  
  
    private HashMap<String, HashSet<String>> conditionsFor = new HashMap<String, HashSet<String>>();  
    private HashMap<String, HashSet<String>> milestonesFor = new HashMap<String, HashSet<String>>();  
    private HashMap<String, HashSet<String>> responsesTo = new HashMap<String, HashSet<String>>();  
    private HashMap<String, HashSet<String>> excludesTo = new HashMap<String, HashSet<String>>();  
    private HashMap<String, HashSet<String>> includesTo = new HashMap<String, HashSet<String>>();  
  
    // Marking  
  
    public DCRMarking marking;  
}  
  
public class DCRMarking {  
  
    public HashSet<String> executed = new HashSet<String>();  
    public HashSet<String> included = new HashSet<String>();  
    public HashSet<String> pending = new HashSet<String>();  
}
```

Implementing DCR graphs (I)

Definition 4.1 (DCR Graph [8]). A DCR graph is a tuple (E, R, M) where

- E is a finite set of (labelled) events, the nodes of the graph.
- R is the edges of the graph. Edges are partitioned into five kinds, named and drawn as follows: The conditions ($\rightarrow\bullet$), responses ($\bullet\rightarrow$), milestones ($\rightarrow\circlearrowright$), inclusions ($\rightarrow+$), and exclusions ($\rightarrow\%$).
- M is the marking of the graph. This is a triple (Ex, Re, In) of sets of events, respectively the previously executed (Ex), the currently pending (Re), and the currently included (In) events.

```
public class DCRGraph {  
    // Events  
    protected HashSet<String> events = new HashSet<String>();  
  
    // Relations  
    private HashMap<String, HashSet<String>> conditionsFor = new HashMap<String, HashSet<String>>();  
    private HashMap<String, HashSet<String>> milestonesFor = new HashMap<String, HashSet<String>>();  
    private HashMap<String, HashSet<String>> responsesTo = new HashMap<String, HashSet<String>>();  
    private HashMap<String, HashSet<String>> excludesTo = new HashMap<String, HashSet<String>>();  
    private HashMap<String, HashSet<String>> includesTo = new HashMap<String, HashSet<String>>();  
    // Marking  
    public DCRMarking marking;  
}  
  
public class DCRMarking {  
    public HashSet<String> executed = new HashSet<String>();  
    public HashSet<String> included = new HashSet<String>();  
    public HashSet<String> pending = new HashSet<String>();  
}
```

Implementing DCR graphs (I)

Definition 4.1 (DCR Graph [8]). A DCR graph is a tuple (E, R, M) where

- E is a finite set of (labelled) events, the nodes of the graph.
- R is the edges of the graph. Edges are partitioned into five kinds, named and drawn as follows: The conditions ($\rightarrow\bullet$), responses ($\bullet\rightarrow$), milestones ($\rightarrow\circlearrowright$), inclusions ($\rightarrow+$), and exclusions ($\rightarrow\%$).
- M is the marking of the graph. This is a triple (Ex, Re, In) of sets of events, respectively the previously executed (Ex), the currently pending (Re), and the currently included (In) events.

```
public class DCRGraph {  
    // Events  
    protected HashSet<String> events = new HashSet<String>();  
  
    // Relations  
    private HashMap<String, HashSet<String>> conditionsFor = new HashMap<String, HashSet<String>>();  
    private HashMap<String, HashSet<String>> milestonesFor = new HashMap<String, HashSet<String>>();  
    private HashMap<String, HashSet<String>> responsesTo = new HashMap<String, HashSet<String>>();  
    private HashMap<String, HashSet<String>> excludesTo = new HashMap<String, HashSet<String>>();  
    private HashMap<String, HashSet<String>> includesTo = new HashMap<String, HashSet<String>>();  
    // Marking  
    public DCRMarking marking;  
}  
  
public class DCRMarking {  
    public HashSet<String> executed = new HashSet<String>();  
    public HashSet<String> included = new HashSet<String>();  
    public HashSet<String> pending = new HashSet<String>();  
}
```

Implementing DCR graphs (I)

Definition 4.1 (DCR Graph [8]). A DCR graph is a tuple (E, R, M) where

- E is a finite set of (labelled) events, the nodes of the graph.
- R is the edges of the graph. Edges are partitioned into five kinds, named and drawn as follows: The conditions ($\rightarrow\bullet$), responses ($\bullet\rightarrow$), milestones ($\rightarrow\circlearrowright$), inclusions ($\rightarrow+$), and exclusions ($\rightarrow\%$).
- M is the marking of the graph. This is a triple (Ex, Re, In) of sets of events, respectively the previously executed (Ex), the currently pending (Re), and the currently included (In) events.

```
public class DCRGraph {  
    // Events  
    protected HashSet<String> events = new HashSet<String>();  
  
    // Relations  
    private HashMap<String, HashSet<String>> conditionsFor = new HashMap<String, HashSet<String>>();  
    private HashMap<String, HashSet<String>> milestonesFor = new HashMap<String, HashSet<String>>();  
    private HashMap<String, HashSet<String>> responsesTo = new HashMap<String, HashSet<String>>();  
    private HashMap<String, HashSet<String>> excludesTo = new HashMap<String, HashSet<String>>();  
    private HashMap<String, HashSet<String>> includesTo = new HashMap<String, HashSet<String>>();  
  
    // Marking  
    public DCRMarking marking;  
}  
  
public class DCRMarking {  
    public HashSet<String> executed = new HashSet<String>();  
    public HashSet<String> included = new HashSet<String>();  
    public HashSet<String> pending = new HashSet<String>();  
}
```

Definition 4.2 (Enabled events). Let $G = (E, R, M)$ be a DCR graph, with marking $M = (Ex, Re, In)$. We say that an event $e \in E$ is enabled and write $e \in \text{enabled}(G)$ iff (a) $e \in In$, (b) $In \cap (\rightarrow\bullet e) \subseteq Ex$, and (c) $In \cap (\rightarrow\bowtie e) \subseteq E \setminus Re$.

Definition 4.2 (Enabled events). Let $G = (E, R, M)$ be a DCR graph, with marking $M = (Ex, Re, In)$. We say that an event $e \in E$ is enabled and write $e \in \text{enabled}(G)$ iff (a) $e \in In$, (b) $In \cap (\rightarrow\bullet e) \subseteq Ex$, and (c) $In \cap (\rightarrow\bowtie e) \subseteq E \setminus Re$.

e is included

Definition 4.2 (Enabled events). Let $G = (E, R, M)$ be a DCR graph, with marking $M = (Ex, Re, In)$. We say that an event $e \in E$ is enabled and write $e \in \text{enabled}(G)$ iff (a) $e \in In$, (b) $In \cap (\rightarrow\bullet e) \subseteq Ex$, and (c) $In \cap (\rightarrow\bowtie e) \subseteq E \setminus Re$.

e is included

All **conditions** for e have been executed

Definition 4.2 (Enabled events). Let $G = (E, R, M)$ be a DCR graph, with marking $M = (Ex, Re, In)$. We say that an event $e \in E$ is enabled and write $e \in \text{enabled}(G)$ iff (a) $e \in In$, (b) $In \cap (\rightarrow\bullet e) \subseteq Ex$, and (c) $In \cap (\rightarrow\bowtie e) \subseteq E \setminus Re$.

e is included

All **conditions** for e have been executed

There are no pending **milestones** for e

Definition 4.2 (Enabled events). Let $G = (E, R, M)$ be a DCR graph, with marking $M = (Ex, Re, In)$. We say that an event $e \in E$ is enabled and write $e \in \text{enabled}(G)$ iff (a) $e \in In$, (b) $In \cap (\rightarrow\bullet e) \subseteq Ex$, and (c) $In \cap (\rightarrow\bowtie e) \subseteq E \setminus Re$.

e is included

All **conditions** for e have been executed

There are no pending **milestones** for e

Only conditions and milestones are blocking!

Definition 4.2 (Enabled events). Let $G = (E, R, M)$ be a DCR graph, with marking $M = (Ex, Re, In)$. We say that an event $e \in E$ is enabled and write $e \in \text{enabled}(G)$ iff (a) $e \in In$, (b) $In \cap (\rightarrow\bullet e) \subseteq Ex$, and (c) $In \cap (\rightarrow\diamond e) \subseteq E \setminus Re$.

```
public Boolean enabled(final DCRMarking marking, final String event) {

    // Open world assumption: if an event doesn't exist in the graph it must be enabled.
    if (!events.contains(event)) { return true; }

    // check included
    if (!marking.included.contains(event)) { return false; }

    // Select only the included conditions
    final Set<String> inccon = new HashSet<String>(conditionsFor.get(event));
    inccon.retainAll(marking.included);
    // Check if all included conditions have been executed
    if (!marking.executed.containsAll(inccon)) { return false; }

    // Select only the included milestones
    final Set<String> incmil = new HashSet<String>(milestonesFor.get(event));
    incmil.retainAll(marking.included);
    // Check if any included milestone has a pending response
    for (final String p : marking.pending) {
        if (incmil.contains(p)) { return false; }
    }

    return true;
}
```

Definition 4.2 (Enabled events). Let $G = (E, R, M)$ be a DCR graph, with marking $M = (Ex, Re, In)$. We say that an event $e \in E$ is enabled and write $e \in \text{enabled}(G)$ iff (a) $e \in In$, (b) $In \cap (\rightarrow\bullet e) \subseteq Ex$, and (c) $In \cap (\rightarrow\diamond e) \subseteq E \setminus Re$.

```
public Boolean enabled(final DCRMarking marking, final String event) {  
  
    // Open world assumption: if an event doesn't exist in the graph it must be enabled.  
    if (!events.contains(event)) { return true; }  
  
    // check included  
    if (!marking.included.contains(event)) { return false; }  
  
    // Select only the included conditions  
    final Set<String> inccon = new HashSet<String>(conditionsFor.get(event));  
    inccon.retainAll(marking.included);  
    // Check if all included conditions have been executed  
    if (!marking.executed.containsAll(inccon)) { return false; }  
  
    // Select only the included milestones  
    final Set<String> incmil = new HashSet<String>(milestonesFor.get(event));  
    incmil.retainAll(marking.included);  
    // Check if any included milestone has a pending response  
    for (final String p : marking.pending) {  
        if (incmil.contains(p)) { return false; }  
    }  
  
    return true;  
}
```

Definition 4.2 (Enabled events). Let $G = (E, R, M)$ be a DCR graph, with marking $M = (Ex, Re, In)$. We say that an event $e \in E$ is enabled and write $e \in \text{enabled}(G)$ iff (a) $e \in In$, (b) $In \cap (\rightarrow\bullet e) \subseteq Ex$, and (c) $In \cap (\rightarrow\diamond e) \subseteq E \setminus Re$.

```

public Boolean enabled(final DCRMarking marking, final String event) {

    // Open world assumption: if an event doesn't exist in the graph it must be enabled.
    if (!events.contains(event)) { return true; }

    // check included
    if (!marking.included.contains(event)) { return false; }

    // Select only the included conditions
    final Set<String> inccon = new HashSet<String>(conditionsFor.get(event));
    inccon.retainAll(marking.included);
    // Check if all included conditions have been executed
    if (!marking.executed.containsAll(inccon)) { return false; }

    // Select only the included milestones
    final Set<String> incmil = new HashSet<String>(milestonesFor.get(event));
    incmil.retainAll(marking.included);
    // Check if any included milestone has a pending response
    for (final String p : marking.pending) {
        if (incmil.contains(p)) { return false; }
    }

    return true;
}

```

Definition 4.2 (Enabled events). Let $G = (E, R, M)$ be a DCR graph, with marking $M = (Ex, Re, In)$. We say that an event $e \in E$ is enabled and write $e \in \text{enabled}(G)$ iff (a) $e \in In$, (b) $In \cap (\rightarrow \bullet e) \subseteq Ex$, and (c) $In \cap (\rightarrow \diamond e) \subseteq E \setminus Re$.

```

public Boolean enabled(final DCRMarking marking, final String event) {

    // Open world assumption: if an event doesn't exist in the graph it must be enabled.
    if (!events.contains(event)) { return true; }

    // check included
    if (!marking.included.contains(event)) { return false; }

    // Select only the included conditions
    final Set<String> inccon = new HashSet<String>(conditionsFor.get(event));
    inccon.retainAll(marking.included);
    // Check if all included conditions have been executed
    if (!marking.executed.containsAll(inccon)) { return false; }

    // Select only the included milestones
    final Set<String> incmil = new HashSet<String>(milestonesFor.get(event));
    incmil.retainAll(marking.included);
    // Check if any included milestone has a pending response
    for (final String p : marking.pending) {
        if (incmil.contains(p)) { return false; }
    }

    return true;
}

```

Definition 4.3 (Execution). Let $G = (E, R, M)$ be a DCR graph with marking $M = (Ex, Re, In)$. Suppose $e \in \text{enabled}(G)$. We may execute e obtaining the resulting DCR graph (E, R, M') with $M' = (Ex', Re', In')$ defined as follows.

1. $Ex' = Ex \cup e$
2. $Re' = (Re \setminus e) \cup (e \bullet \rightarrow)$
3. $In' = (In \setminus (e \rightarrow \%)) \cup (e \rightarrow +)$

Definition 4.3 (Execution). Let $G = (E, R, M)$ be a DCR graph with marking $M = (Ex, Re, In)$. Suppose $e \in \text{enabled}(G)$. We may execute e obtaining the resulting DCR graph (E, R, M') with $M' = (Ex', Re', In')$ defined as follows.

1. $Ex' = Ex \cup e$
2. $Re' = (Re \setminus e) \cup (e \bullet \rightarrow)$
3. $In' = (In \setminus (e \rightarrow \%)) \cup (e \rightarrow +)$

1. add e to the set Ex of executed events.

Definition 4.3 (Execution). Let $G = (E, R, M)$ be a DCR graph with marking $M = (Ex, Re, In)$. Suppose $e \in \text{enabled}(G)$. We may execute e obtaining the resulting DCR graph (E, R, M') with $M' = (Ex', Re', In')$ defined as follows.

1. $Ex' = Ex \cup e$
2. $Re' = (Re \setminus e) \cup (e \bullet \rightarrow)$
3. $In' = (In \setminus (e \rightarrow \%)) \cup (e \rightarrow +)$

1. add e to the set Ex of executed events.
2. Update the currently required responses Re by first removing e , then adding any responses required by e

Definition 4.3 (Execution). Let $G = (E, R, M)$ be a DCR graph with marking $M = (Ex, Re, In)$. Suppose $e \in \text{enabled}(G)$. We may execute e obtaining the resulting DCR graph (E, R, M') with $M' = (Ex', Re', In')$ defined as follows.

1. $Ex' = Ex \cup e$
2. $Re' = (Re \setminus e) \cup (e \bullet \rightarrow)$
3. $In' = (In \setminus (e \rightarrow \%)) \cup (e \rightarrow +)$

1. add e to the set Ex of executed events.
2. Update the currently required responses Re by first removing e , then adding any responses required by e
3. Update the currently included events by first removing all those excluded by e , then adding all those included by e

DCR Execution Semantics

Definition 4.3 (Execution). Let $G = (E, R, M)$ be a DCR graph with marking $M = (Ex, Re, In)$. Suppose $e \in \text{enabled}(G)$. We may execute e obtaining the resulting DCR graph (E, R, M') with $M' = (Ex', Re', In')$ defined as follows.

1. $Ex' = Ex \cup e$
2. $Re' = (Re \setminus e) \cup (e \bullet \rightarrow)$
3. $In' = (In \setminus (e \rightarrow \%)) \cup (e \rightarrow +)$

```
public DCRMarking execute(final DCRMarking marking, final String event) {  
    // Check if the event exists  
    if (!events.contains(event)) { return marking; }  
  
    // Check if the event is enabled  
    if (!this.enabled(marking, event)) { return marking; }  
  
    // Create a new marking  
    DCRMarking result = marking.clone();  
  
    // Add the event to the set of executed events.  
    result.executed.add(event);  
  
    // Remove the event from the set of pending events.  
    result.pending.remove(event);  
  
    // Add all new responses.  
    result.pending.addAll(responsesTo.get(event));  
  
    // Remove all excluded events  
    result.included.removeAll(excludesTo.get(event));  
  
    // Add all included events  
    result.included.addAll(includesTo.get(event));  
  
    return result;  
}
```

DCR Execution Semantics

Definition 4.3 (Execution). Let $G = (E, R, M)$ be a DCR graph with marking $M = (Ex, Re, In)$. Suppose $e \in \text{enabled}(G)$. We may execute e obtaining the resulting DCR graph (E, R, M') with $M' = (Ex', Re', In')$ defined as follows.

1. $Ex' = Ex \cup e$
2. $Re' = (Re \setminus e) \cup (e \bullet \rightarrow)$
3. $In' = (In \setminus (e \rightarrow \%)) \cup (e \rightarrow +)$

```
public DCRMarking execute(final DCRMarking marking, final String event) {  
    // Check if the event exists  
    if (!events.contains(event)) { return marking; }  
  
    // Check if the event is enabled  
    if (!this.enabled(marking, event)) { return marking; }  
  
    // Create a new marking  
    DCRMarking result = marking.clone();  
  
    // Add the event to the set of executed events.  
    result.executed.add(event);  
  
    // Remove the event from the set of pending events.  
    result.pending.remove(event);  
  
    // Add all new responses.  
    result.pending.addAll(responsesTo.get(event));  
  
    // Remove all excluded events  
    result.included.removeAll(excludesTo.get(event));  
  
    // Add all included events  
    result.included.addAll(includesTo.get(event));  
  
    return result;  
}
```

DCR Execution Semantics

Definition 4.3 (Execution). Let $G = (E, R, M)$ be a DCR graph with marking $M = (Ex, Re, In)$. Suppose $e \in \text{enabled}(G)$. We may execute e obtaining the resulting DCR graph (E, R, M') with $M' = (Ex', Re', In')$ defined as follows.

1. $Ex' = Ex \cup e$
2. $Re' = (Re \setminus e) \cup (e \bullet \rightarrow)$
3. $In' = (In \setminus (e \rightarrow \%)) \cup (e \rightarrow +)$

```
public DCRMarking execute(final DCRMarking marking, final String event) {  
    // Check if the event exists  
    if (!events.contains(event)) { return marking; }  
  
    // Check if the event is enabled  
    if (!this.enabled(marking, event)) { return marking; }  
  
    // Create a new marking  
    DCRMarking result = marking.clone();  
  
    // Add the event to the set of executed events.  
    result.executed.add(event);  
  
    // Remove the event from the set of pending events.  
    result.pending.remove(event);  
  
    // Add all new responses.  
    result.pending.addAll(responsesTo.get(event));  
  
    // Remove all excluded events  
    result.included.removeAll(excludesTo.get(event));  
  
    // Add all included events  
    result.included.addAll(includesTo.get(event));  
  
    return result;  
}
```

DCR Execution Semantics

Definition 4.3 (Execution). Let $G = (E, R, M)$ be a DCR graph with marking $M = (Ex, Re, In)$. Suppose $e \in \text{enabled}(G)$. We may execute e obtaining the resulting DCR graph (E, R, M') with $M' = (Ex', Re', In')$ defined as follows.

1. $Ex' = Ex \cup e$
2. $Re' = (Re \setminus e) \cup (e \bullet \rightarrow)$
3. $In' = (In \setminus (e \rightarrow \%)) \cup (e \rightarrow +)$

```
public DCRMarking execute(final DCRMarking marking, final String event) {  
    // Check if the event exists  
    if (!events.contains(event)) { return marking; }  
  
    // Check if the event is enabled  
    if (!this.enabled(marking, event)) { return marking; }  
  
    // Create a new marking  
    DCRMarking result = marking.clone();  
  
    // Add the event to the set of executed events.  
    result.executed.add(event);  
  
    // Remove the event from the set of pending events.  
    result.pending.remove(event);  
  
    // Add all new responses.  
    result.pending.addAll(responsesTo.get(event));  
  
    // Remove all excluded events  
    result.included.removeAll(excludesTo.get(event));  
  
    // Add all included events  
    result.included.addAll(includesTo.get(event));  
  
    return result;  
}
```

Definition 4.4 (Transitions, runs, traces). Let G be a DCR graph. If $e \in \text{enabled}(G)$ and executing e in G yields H , we say that G has transition on e to H and write $G \rightarrow_e H$. A run of G is a (finite or infinite) sequence of DCR graphs G_i and events e_i such that: $G = G_0 \rightarrow_{e_0} G_1 \rightarrow_{e_1} \dots$

A trace of G is a sequence of labels of events e_i associated with a run of G . We write $\text{runs}(G)$ and $\text{traces}(G)$ for the set of runs and traces of G , respectively

Definition 4.4 (Transitions, runs, traces). Let G be a DCR graph. If $e \in \text{enabled}(G)$ and executing e in G yields H , we say that G has transition on e to H and write $G \rightarrow_e H$. A run of G is a (finite or infinite) sequence of DCR graphs G_i and events e_i such that: $G = G_0 \rightarrow_{e_0} G_1 \rightarrow_{e_1} \dots$. A trace of G is a sequence of labels of events e_i associated with a run of G . We write $\text{runs}(G)$ and $\text{traces}(G)$ for the set of runs and traces of G , respectively.

- When is a trace accepting?

Definition 4.4 (Transitions, runs, traces). Let G be a DCR graph. If $e \in \text{enabled}(G)$ and executing e in G yields H , we say that G has transition on e to H and write $G \rightarrow_e H$. A run of G is a (finite or infinite) sequence of DCR graphs G_i and events e_i such that: $G = G_0 \rightarrow_{e_0} G_1 \rightarrow_{e_1} \dots$. A trace of G is a sequence of labels of events e_i associated with a run of G . We write $\text{runs}(G)$ and $\text{traces}(G)$ for the set of runs and traces of G , respectively.

- When is a trace accepting?

Definition 4.5 (Acceptance). A run $G_0 \rightarrow_{e_0} G_1 \rightarrow_{e_1} \dots$ is accepting iff for all n with $e \in \text{In}(G_n) \cap \text{Re}(G_n)$ there exists $m \geq n$ s.t. either $e_m = e$, or $e \notin \text{In}(G_m)$. A trace is accepting iff it has an underlying run which is.

Definition 4.4 (Transitions, runs, traces). Let G be a DCR graph. If $e \in \text{enabled}(G)$ and executing e in G yields H , we say that G has transition on e to H and write $G \rightarrow_e H$. A run of G is a (finite or infinite) sequence of DCR graphs G_i and events e_i such that: $G = G_0 \rightarrow_{e_0} G_1 \rightarrow_{e_1} \dots$. A trace of G is a sequence of labels of events e_i associated with a run of G . We write $\text{runs}(G)$ and $\text{traces}(G)$ for the set of runs and traces of G , respectively.

- When is a trace accepting?

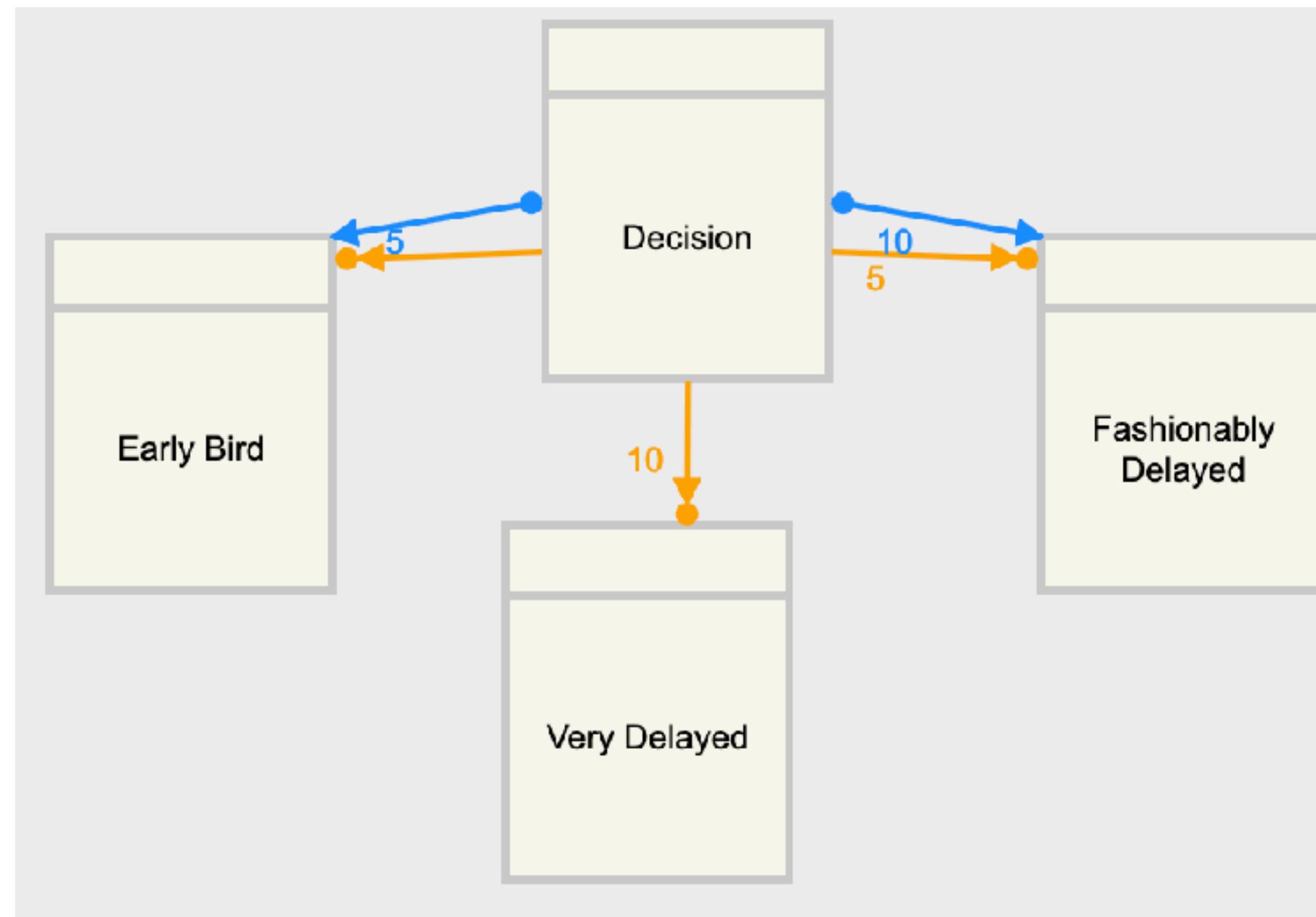
Definition 4.5 (Acceptance). A run $G_0 \rightarrow_{e_0} G_1 \rightarrow_{e_1} \dots$ is accepting iff for all n with $e \in \text{In}(G_n) \cap \text{Re}(G_n)$ there exists $m \geq n$ s.t. either $e_m = e$, or $e \notin \text{In}(G_m)$. A trace is accepting iff it has an underlying run which is.

If an event e becomes included and pending, then at some point afterwards, it is either or excluded

```
public Set<String> getIncludedPending()
{
    // Select those included events that are also pending
    HashSet<String> result = new HashSet<String>(marking.included);
    result.retainAll(marking.pending);
    return result;
}

public boolean isAccepting()
{
    // Check if there are any included pending responses
    return getIncludedPending().isEmpty();
}
```

Temporal Dependencies

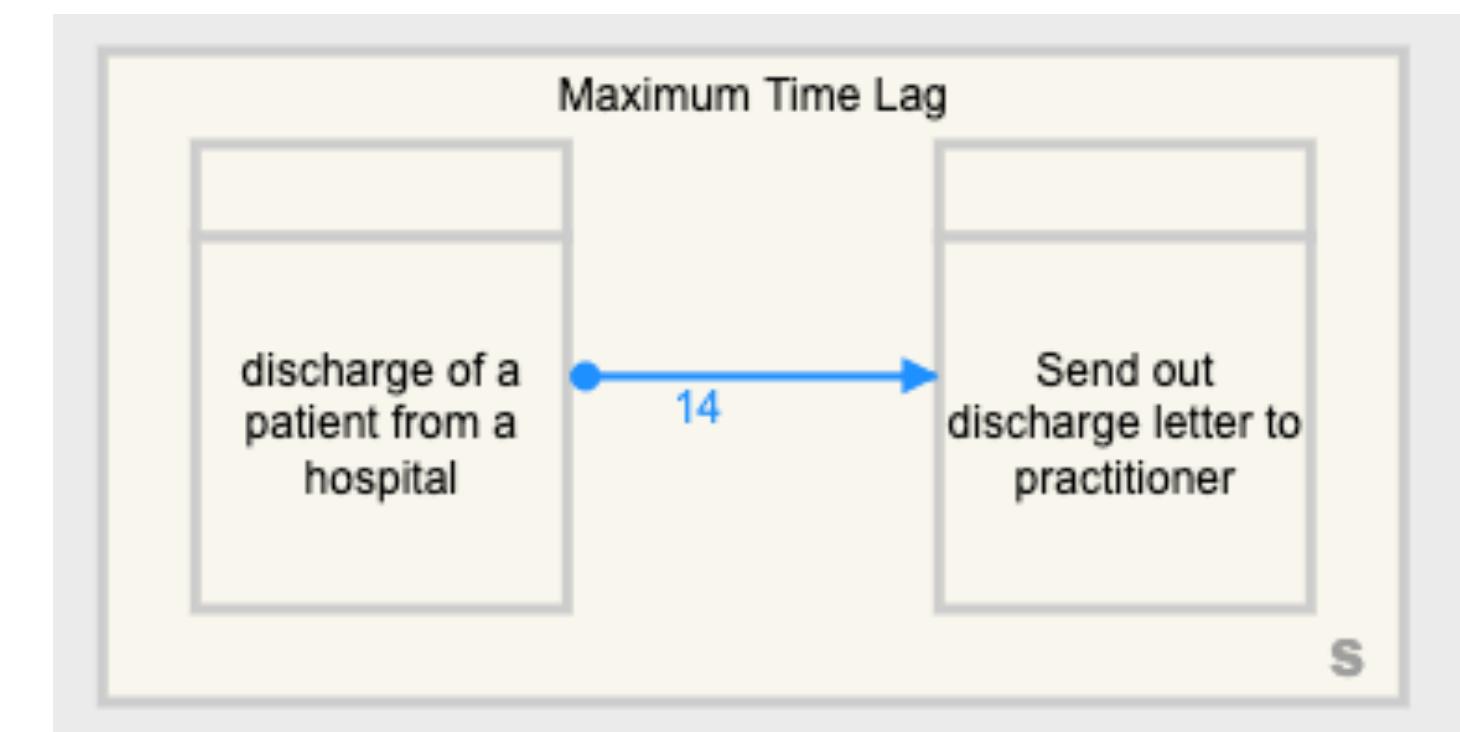


Activity	Enabled timespan	Pending timespan
EB	$[0, \infty)$	$[0, \infty)$
FD	$(5, \infty)$	$[0, \infty)$
VD	$(10, \infty)$	—

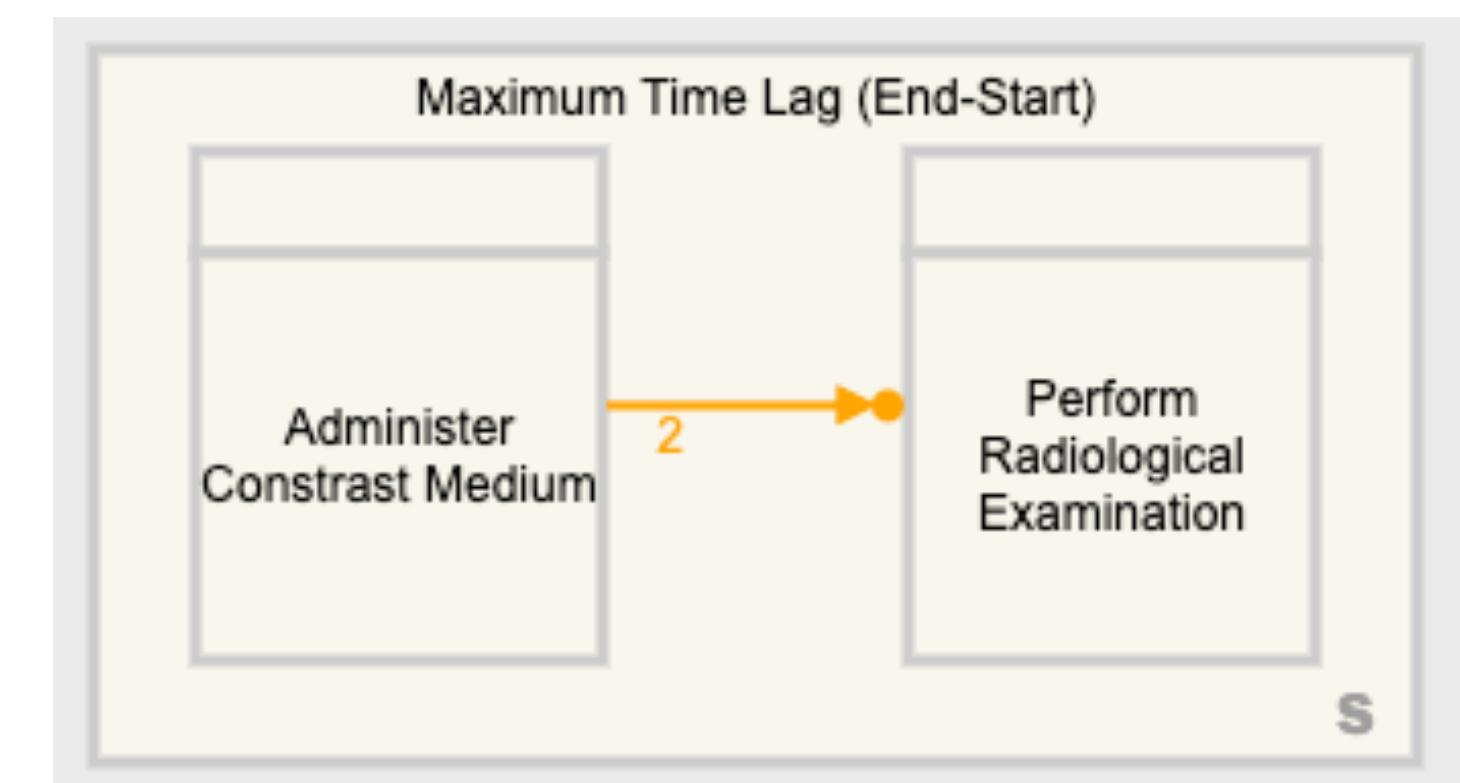
- Conditions and Responses might define temporal dependencies between events.
- An x -timed response: “The execution of this activity has to be done within x time units”
- An x -time condition: “The execution of this activity is possible after x time units”
- Performing activities on time determines whether the graph is accepting or not

Examples of time patterns

- **Maximum time lag (end-end)** between discharge of a patient from a hospital and sending out the discharge letter to the general practitioner of the patient should be 2 weeks

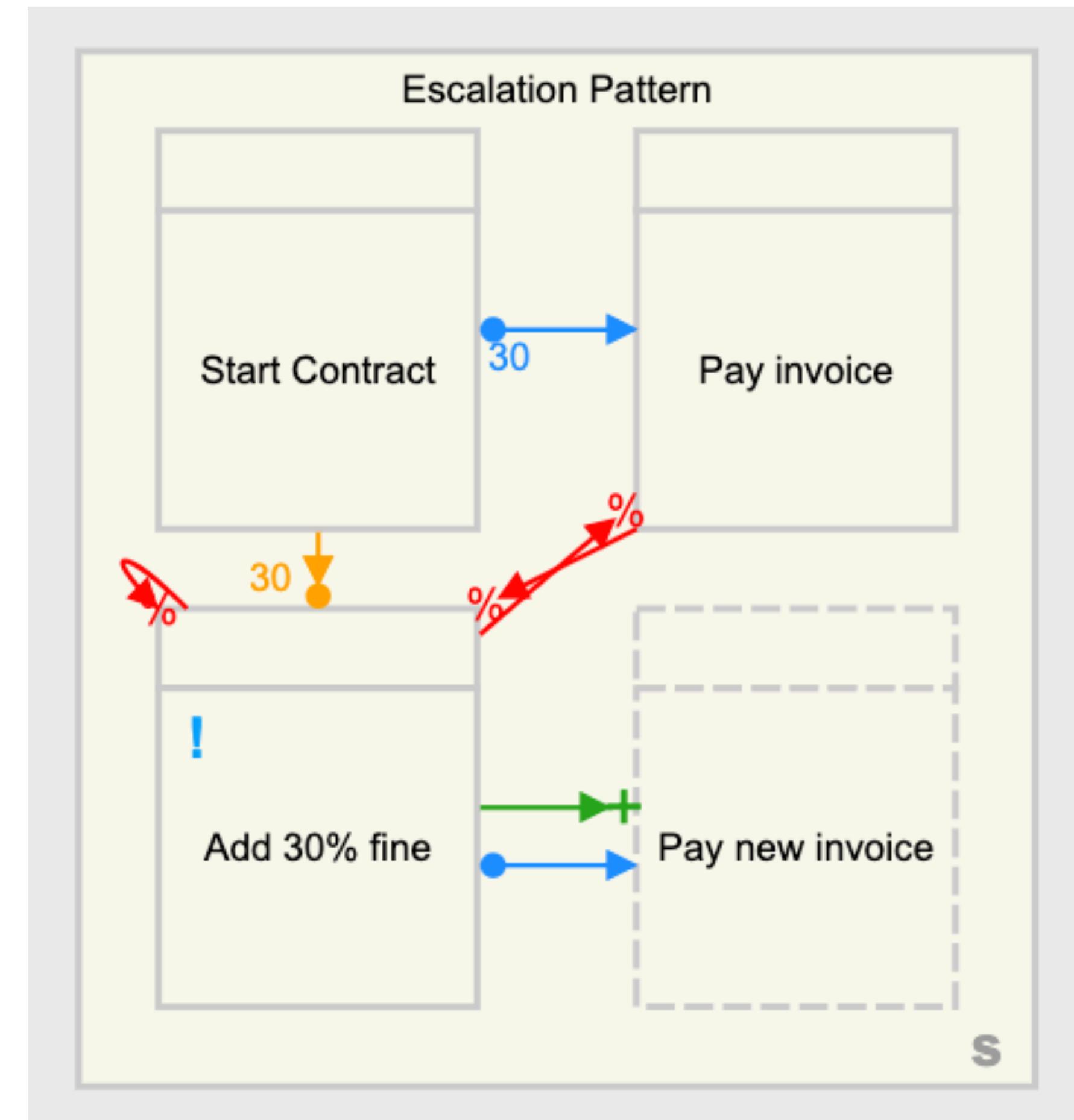


- **Maximum time lag (end-start)** A contrast medium has to be administered 2 days before a radiological examination



Escalation Pattern

- Defines what to do in the exceptional case that a default activity cannot be achieved within the expected time
- The invoice should be paid within 30 days, and in case this condition cannot be achieved, the payer should incur a 30% fine



Timed DCR graphs - briefly

- Assume:
 - A fixed universe of events \mathcal{E} and labels \mathcal{L}
 - $e, f \in \mathcal{E}$
 - A mapping λ from events to labels
 - tick $\notin \mathcal{E}$
 - The grammar for DCR process is given as:

$P, Q ::= [M] \lambda T$ process

$T, U ::=$	$e \xrightarrow{t_0} \bullet f$	$ $	$e \bullet \xrightarrow{t_\omega} f$
	$ e \xrightarrow{+} f$	$ $	$e \xrightarrow{\%} f$
	$ e \xrightarrow{\diamond} f$	$ $	$T \parallel U$
	$ 0$		

$M, N ::= M, e : \Phi \mid \epsilon$	marking
$\lambda ::= \lambda, e : l \mid \epsilon$	labelling

$\Phi ::= (h, i, p)$

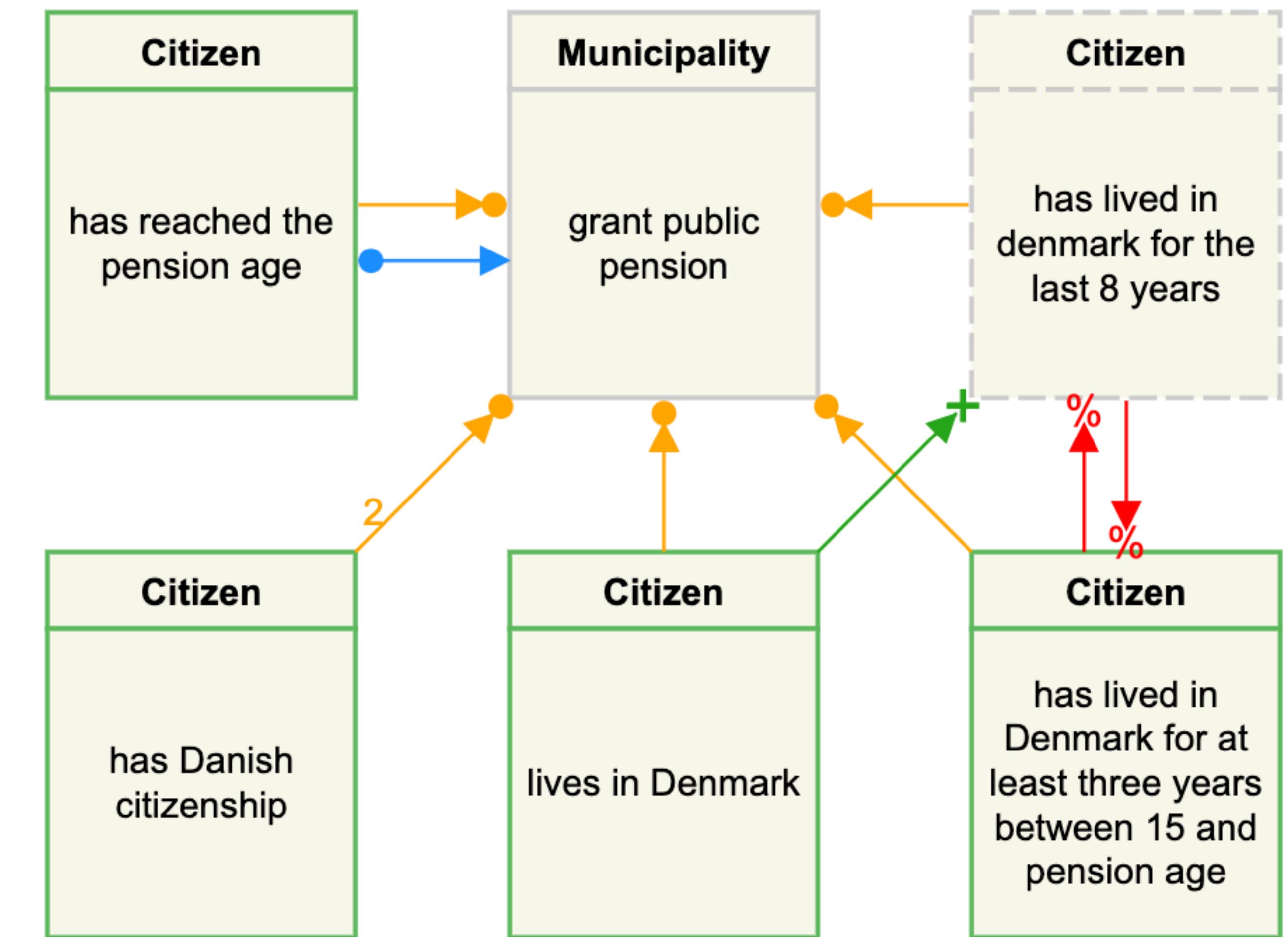
$h ::= f \mid t_0$

i ::= f | t

$t_0 \in \mathbb{N} \cup \{0\}$ 0-time

$t_{\text{,,}} \in \mathbb{N} \cup \{\omega\}$ ω -tim

$p ::= \text{f} \mid 0 \mid t_\omega$



Timed DCR graphs - briefly

- Assume:
 - A fixed universe of events \mathcal{E} and labels \mathcal{L}
 - $e, f \in \mathcal{E}$
 - A mapping λ from events to labels
 - tick $\notin \mathcal{E}$
 - The grammar for DCR process is given as:

$P, Q ::= [M] \lambda T$ process

$T, U ::=$	$e \xrightarrow{t_0} \bullet f$	$ $	$e \bullet \xrightarrow{t_\omega} f$
	$ e \xrightarrow{+} f$	$ $	$e \xrightarrow{\%} f$
	$ e \xrightarrow{\diamond} f$	$ $	$T \parallel U$
	$ 0$		

$M, N ::= M, e : \Phi \mid \epsilon$ marking

$\lambda ::= \lambda, e : l \mid \epsilon$

$\Phi ::= (h, i, v)$

$h ::= f \mid t_0$

i ::= f | t

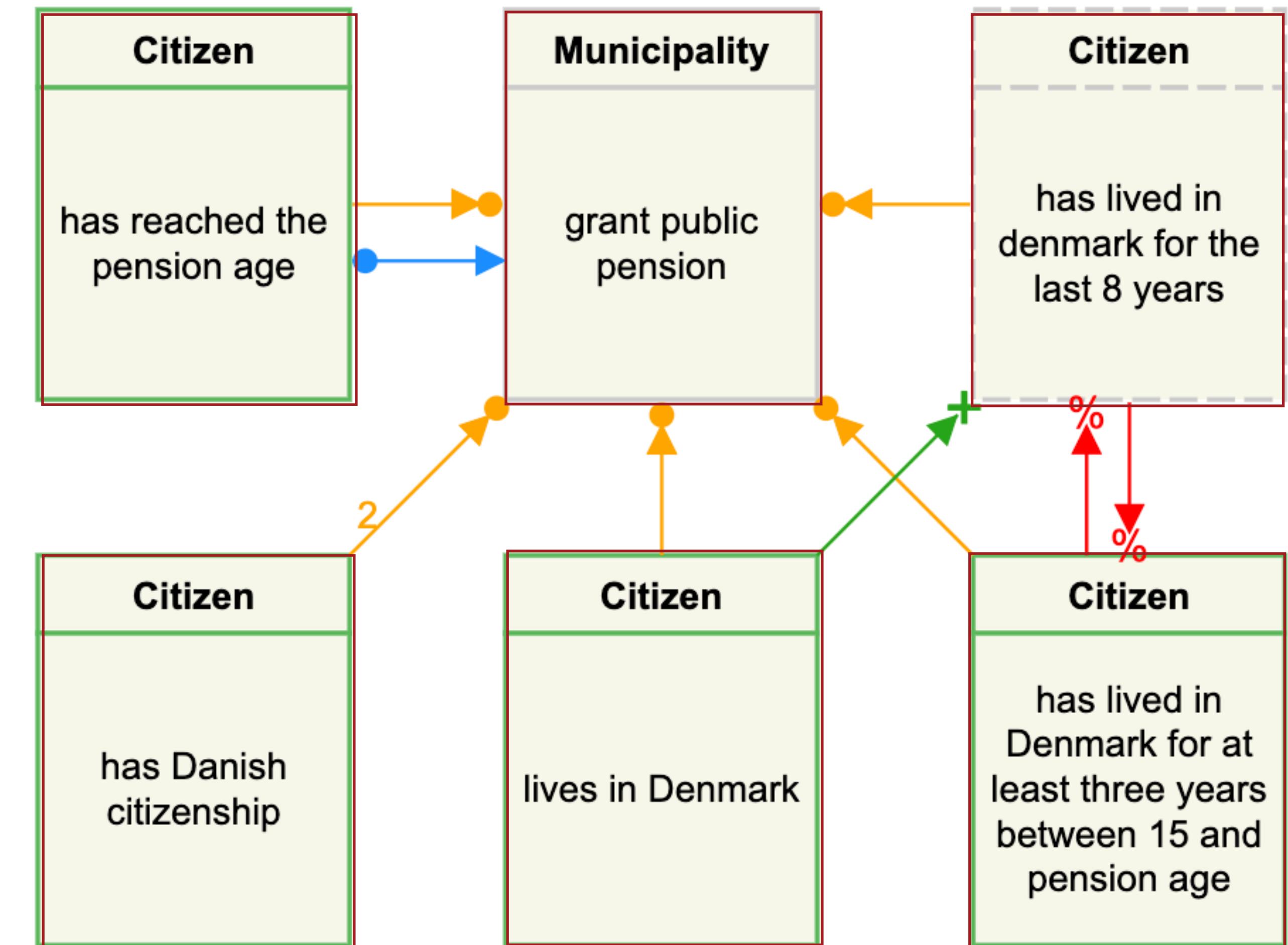
$p ::= f \mid 0 \mid t_\omega$

$t_0 \in \mathbb{N} \cup \{0\}$ 0-time

0-time

$$t_{\omega} \in \mathbb{N} \cup \{\omega\}$$

ω-time



Timed DCR graphs - briefly

- Assume:
 - A fixed universe of events \mathcal{E} and labels \mathcal{L}
 - $e, f \in \mathcal{E}$
 - A mapping λ from events to labels
 - tick $\notin \mathcal{E}$
 - The grammar for DCR process is given as:

$P, Q ::= [M] \lambda T$ process

$T, U ::=$	$e \xrightarrow{t_0} \bullet f$	$ $	$e \bullet \xrightarrow{t_\omega} f$
	$ e \xrightarrow{+} f$	$ $	$e \xrightarrow{\%} f$
	$ e \xrightarrow{\diamond} f$	$ $	$T \parallel U$
	$ 0$		

$M, N ::= M, e : \Phi \mid \epsilon$ marking

$\lambda ::= \lambda, e : l \mid \epsilon$

$\Phi ::= (h, i, p)$

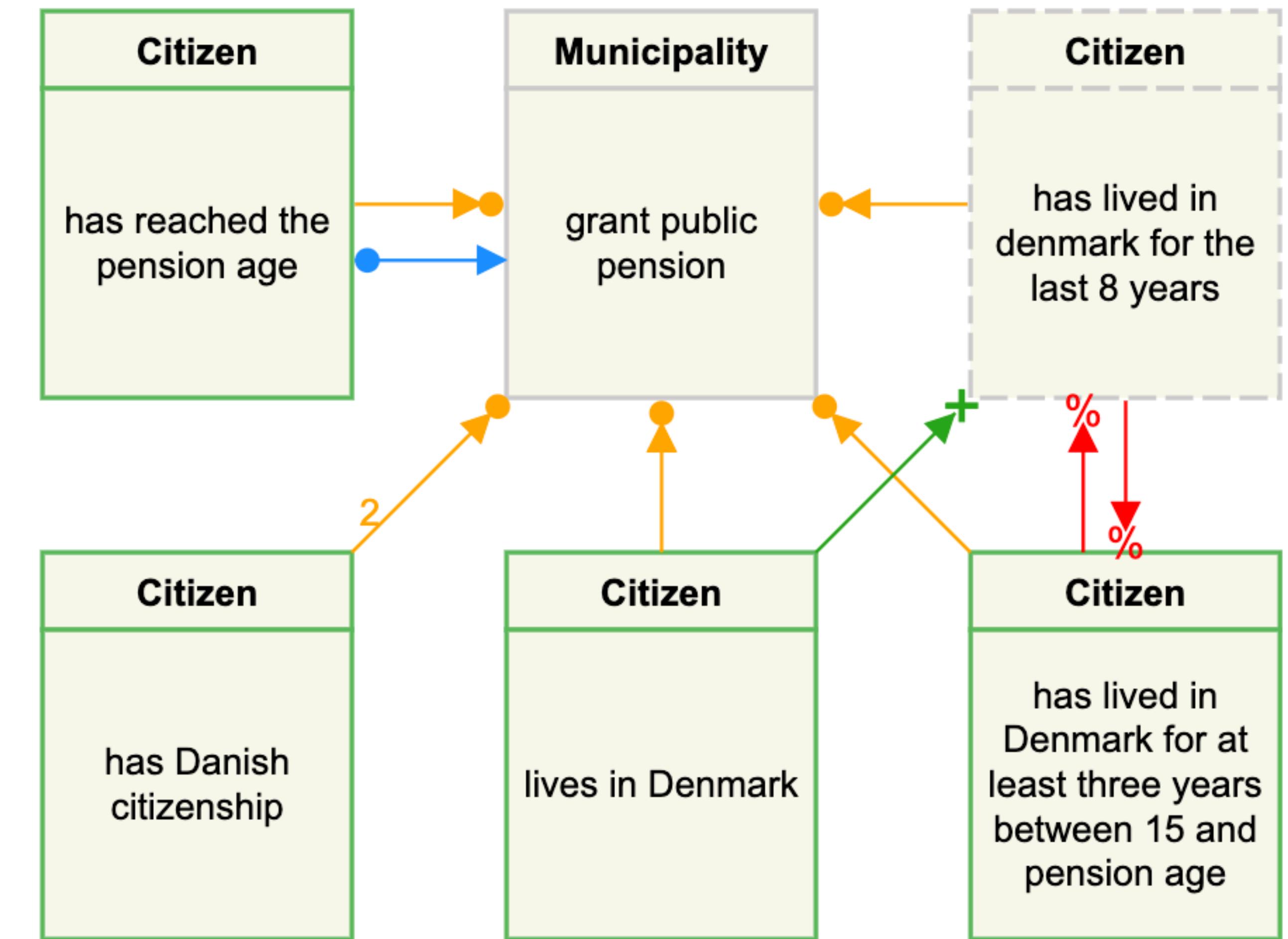
$h ::= f \mid t_0$

i ::= f | t

$t_0 \in \mathbb{N} \cup \{0\}$ 0-time

$t_{\text{,,}} \in \mathbb{N} \cup \{\omega\}$ ω -tim

$p ::= \text{f} \mid 0 \mid t_\omega$



Timed DCR graphs - briefly

- Assume:
 - A fixed universe of events \mathcal{E} , and labels \mathcal{L}
 - $e, f \in \mathcal{E}$
 - A mapping λ from events to labels
 - tick $\notin \mathcal{E}$
- The grammar for DCR process is given as:

$P, Q ::= [M] \lambda T$ process

$T, U ::= e \xrightarrow{t_0} f$ | $e \xrightarrow{t_\omega} f$
 | $e \rightarrow + f$ | $e \rightarrow \% f$
 | $e \rightarrow \diamond f$ | $T \parallel U$
 | 0

$M, N ::= M, e : \Phi$ | ϵ marking
 $\lambda ::= \lambda, e : l$ | ϵ labelling

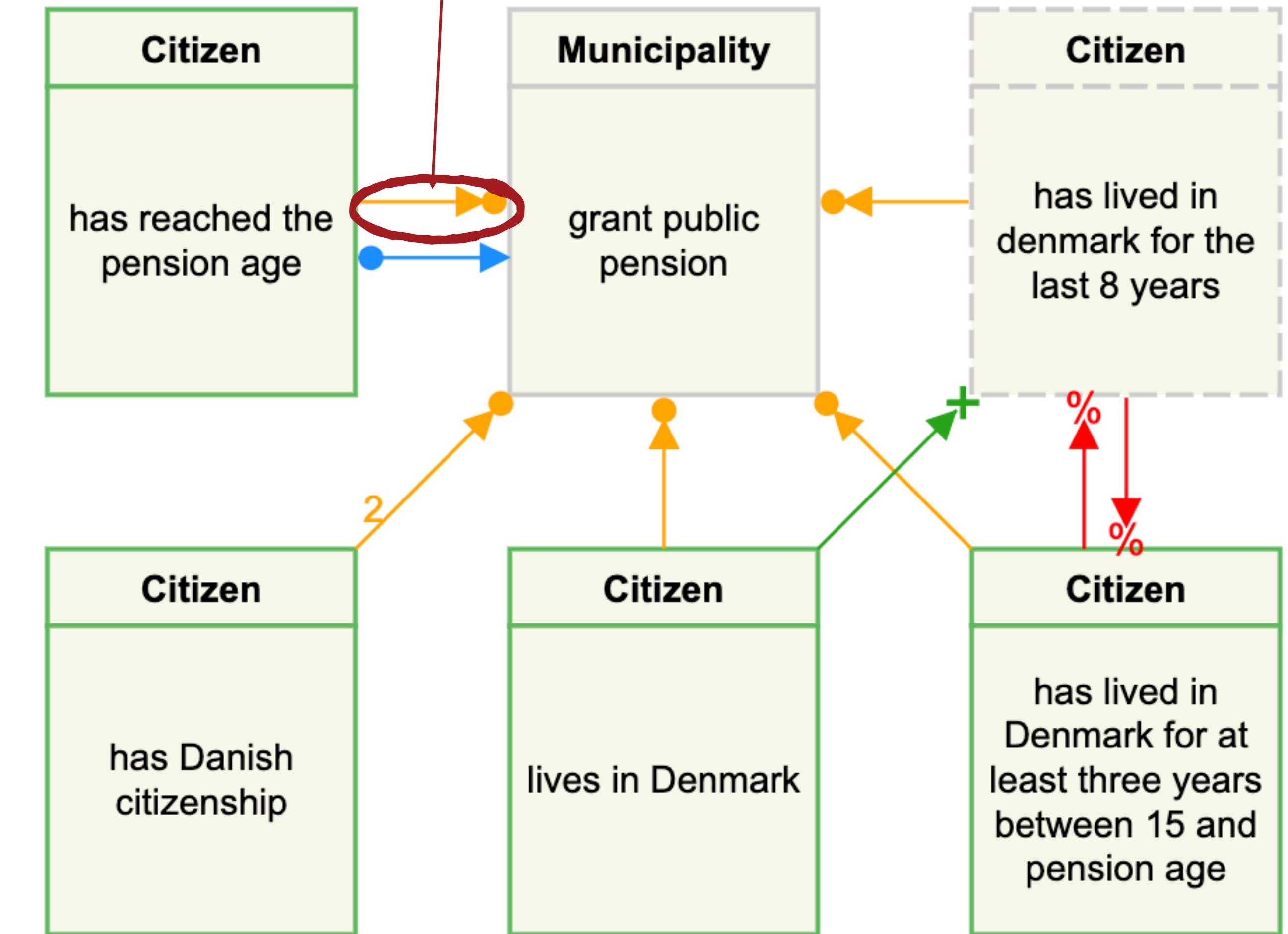
$\Phi ::= (h, i, p)$

$h ::= f$ | t_0 $t_0 \in \mathbb{N} \cup \{0\}$ 0-time

$i ::= f$ | t $t_\omega \in \mathbb{N} \cup \{\omega\}$ ω -time

$p ::= f$ | 0 | t_ω

Condition
relation



Timed DCR graphs - briefly

- Assume:
 - A fixed universe of events \mathcal{E} , and labels \mathcal{L}
 - $e, f \in \mathcal{E}$
 - A mapping λ from events to labels
 - tick $\notin \mathcal{E}$
- The grammar for DCR process is given as:

$P, Q ::= [M] \lambda T$ process

$T, U ::= e \xrightarrow{t_0} f$ $e \xrightarrow{t_\omega} f$
 $| e \rightarrow + f$ $e \rightarrow \% f$
 $| e \rightarrow \diamond f$ $T \parallel U$
 $| 0$

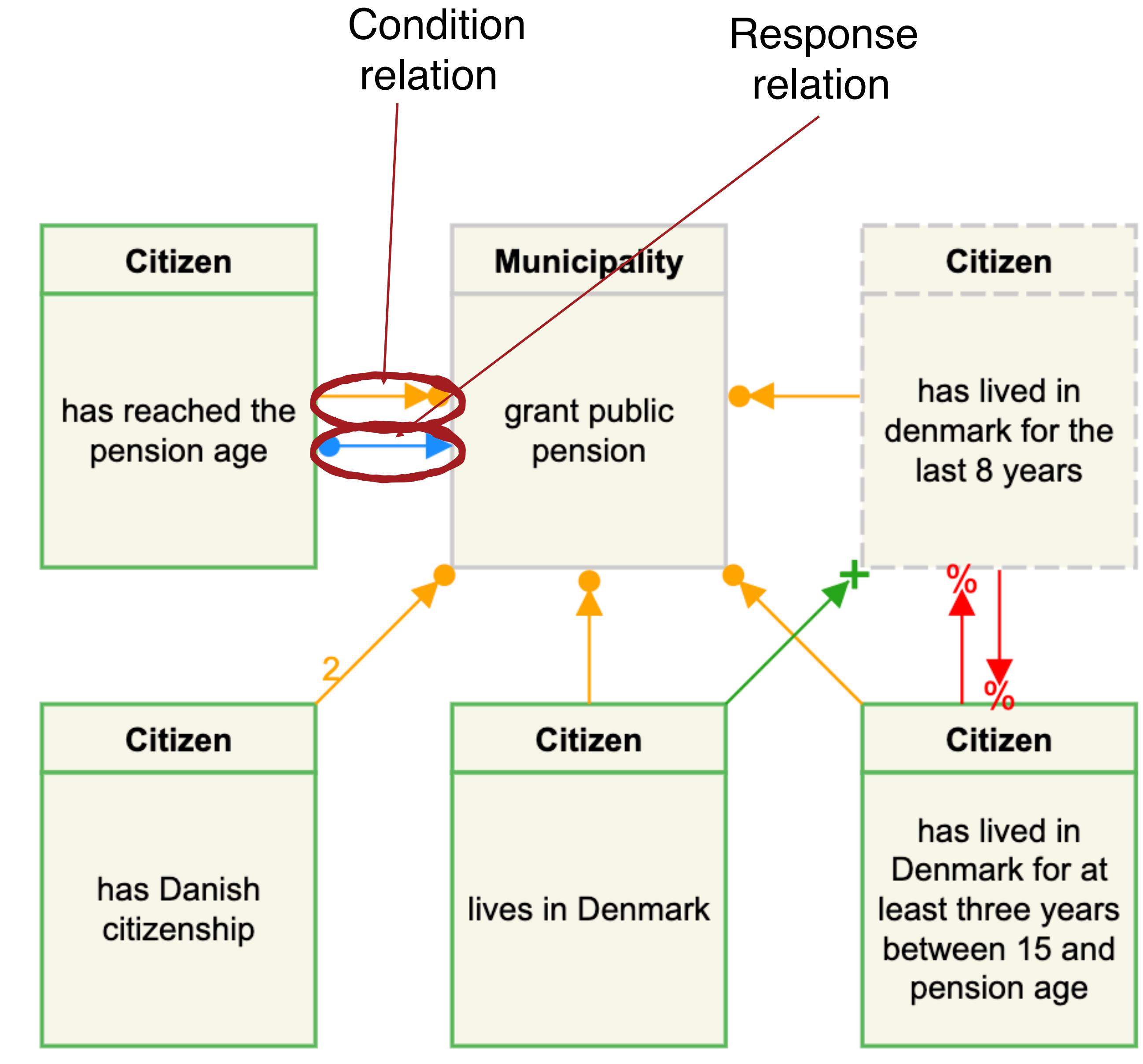
$M, N ::= M, e : \Phi \mid \epsilon$ marking
 $\lambda ::= \lambda, e : l \mid \epsilon$ labelling

$\Phi ::= (h, i, p)$

$h ::= f \mid t_0$ $t_0 \in \mathbb{N} \cup \{0\}$ 0-time

$i ::= f \mid t$ $t_\omega \in \mathbb{N} \cup \{\omega\}$ ω -time

$p ::= f \mid 0 \mid t_\omega$



Timed DCR graphs - briefly

- Assume:
 - A fixed universe of events \mathcal{E} , and labels \mathcal{L}
 - $e, f \in \mathcal{E}$
 - A mapping λ from events to labels
 - tick $\notin \mathcal{E}$
- The grammar for DCR process is given as:

$$P, Q ::= [M] \lambda T \quad \text{process}$$

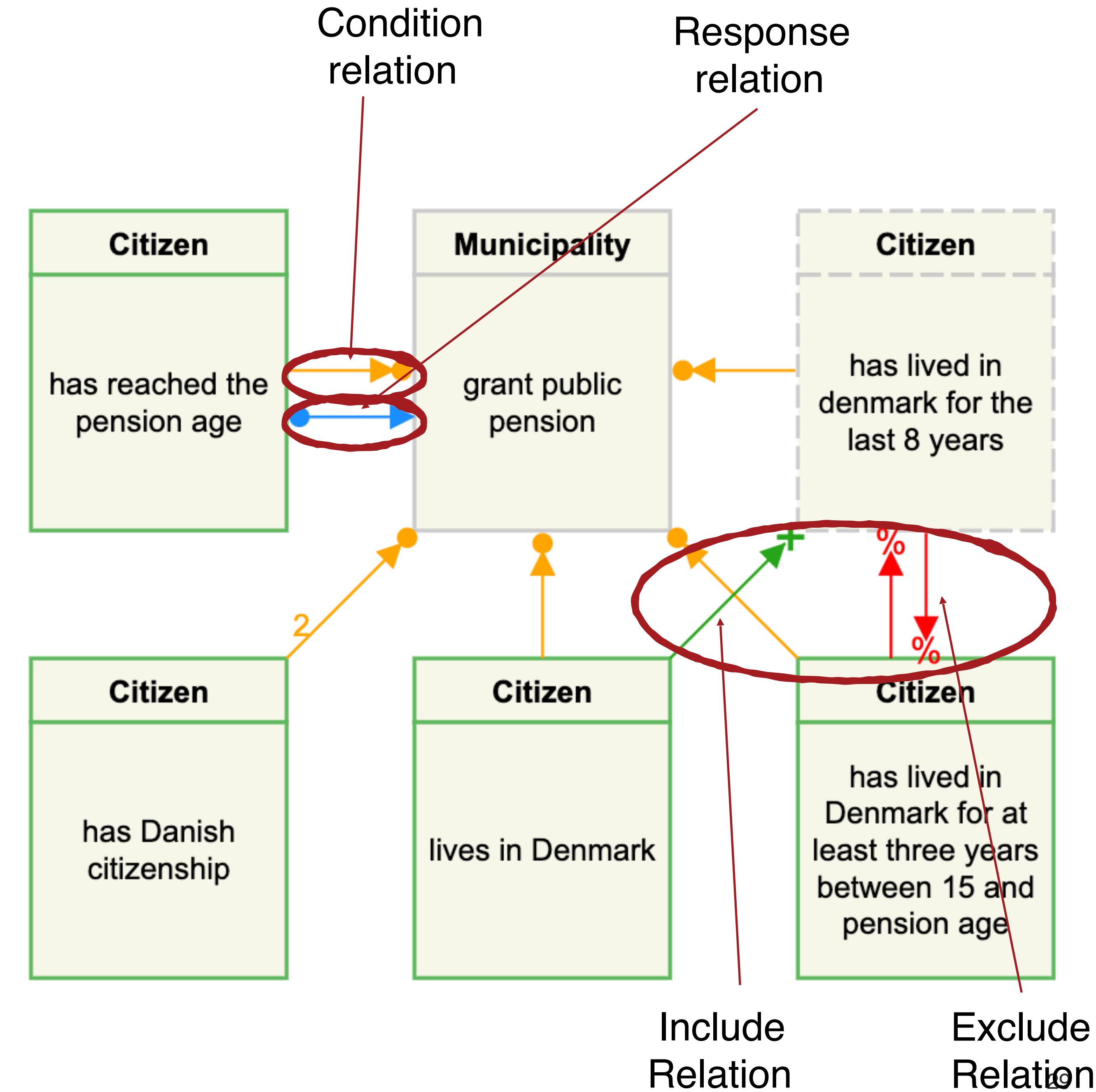
$$T, U ::= e \xrightarrow{t_0} f \quad | \quad e \xrightarrow{t_\omega} f \\ | \quad e \xrightarrow{+} f \quad | \quad e \xrightarrow{\%} f \\ | \quad e \xrightarrow{\diamond} f \quad | \quad T \parallel U \\ | \quad 0$$

$$M, N ::= M, e : \Phi \mid \epsilon \quad \text{marking} \\ \lambda ::= \lambda, e : l \mid \epsilon \quad \text{labelling}$$

$$\Phi ::= (h, i, p)$$

$$h ::= f \mid t_0 \quad t_0 \in \mathbb{N} \cup \{0\} \quad \text{0-time}$$

$$i ::= f \mid t \quad t_\omega \in \mathbb{N} \cup \{\omega\} \quad \omega\text{-time}$$

$$p ::= f \mid 0 \mid t_\omega$$


Timed DCR graphs - briefly

- Assume:
 - A fixed universe of events \mathcal{E} , and labels \mathcal{L}
 - $e, f \in \mathcal{E}$
 - A mapping λ from events to labels
 - tick $\notin \mathcal{E}$
- The grammar for DCR process is given as:

$$P, Q ::= [M] \lambda T \quad \text{process}$$

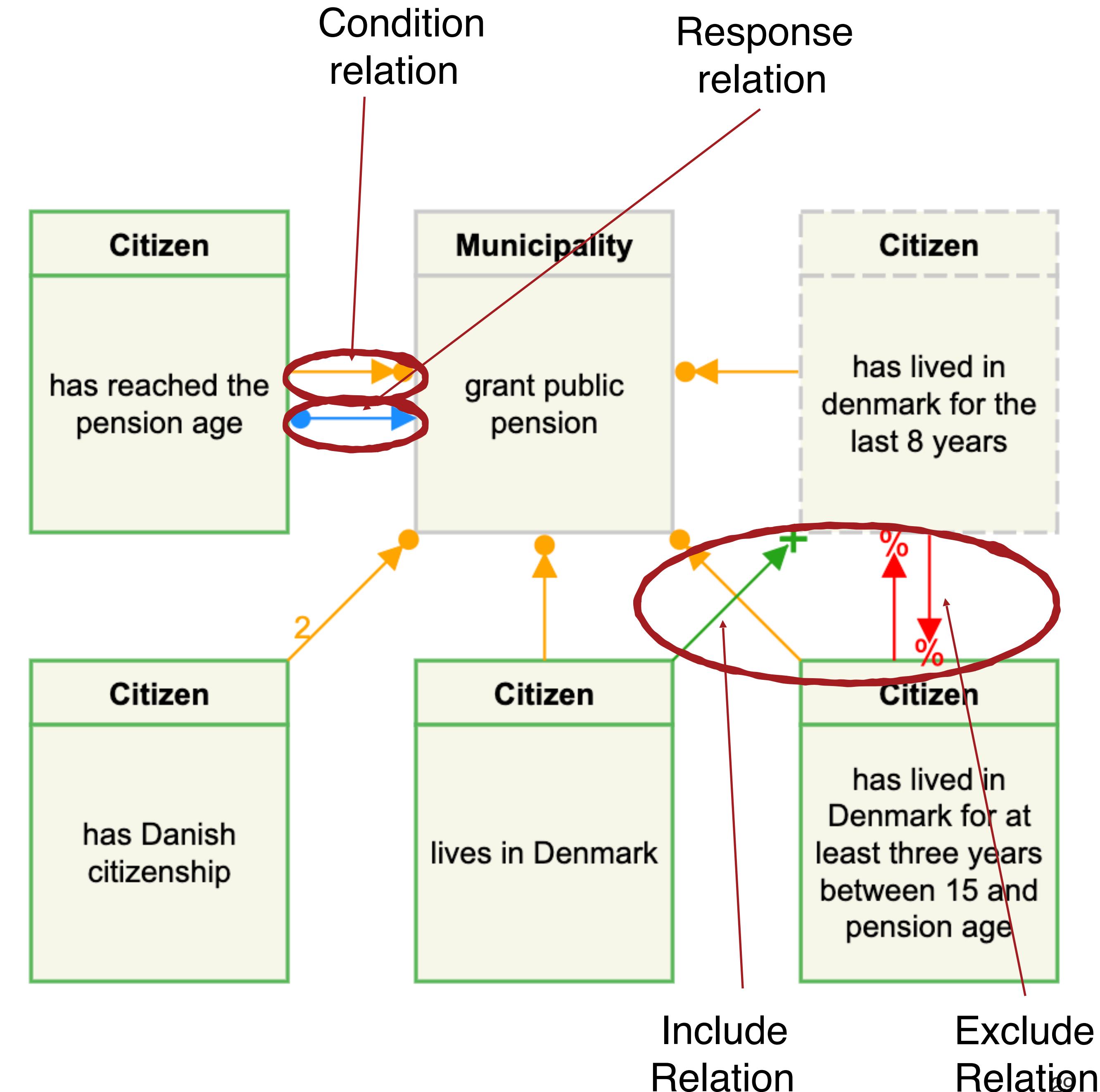
$$T, U ::= e \xrightarrow{t_0} f \quad | \quad e \xrightarrow{t_\omega} f \\ | \quad e \rightarrow + f \quad | \quad e \rightarrow \% f \\ | \quad e \rightarrow \diamond f \quad | \quad T \parallel U \\ | \quad 0$$

$$M, N ::= M, e : \Phi \mid \epsilon \quad \text{marking} \\ \lambda ::= \lambda, e : l \mid \epsilon \quad \text{labelling}$$

$$\Phi ::= (h, i, p)$$

$$h ::= f \mid t_0 \quad t_0 \in \mathbb{N} \cup \{0\} \quad \text{0-time}$$

$$i ::= f \mid t \quad t_\omega \in \mathbb{N} \cup \{\omega\} \quad \omega\text{-time}$$

$$p ::= f \mid 0 \mid t_\omega$$


Timed DCR graphs - briefly

- Assume:
 - A fixed universe of events \mathcal{E} , and labels \mathcal{L}
 - $e, f \in \mathcal{E}$
 - A mapping λ from events to labels
 - tick $\notin \mathcal{E}$
- The grammar for DCR process is given as:

$P, Q ::= [M] \lambda T$ process

$T, U ::= e \xrightarrow{t_0} f \quad | \quad e \xrightarrow{t_\omega} f$
 $| \quad e \rightarrow + f \quad | \quad e \rightarrow \% f$
 $| \quad e \rightarrow \diamond f \quad | \quad T \parallel U$
 $| \quad 0$

$M, N ::= M, e : \Phi \mid \epsilon$ marking
 $\lambda ::= \lambda, e : l \mid \epsilon$ labelling

$\Phi ::= (h, i, p)$

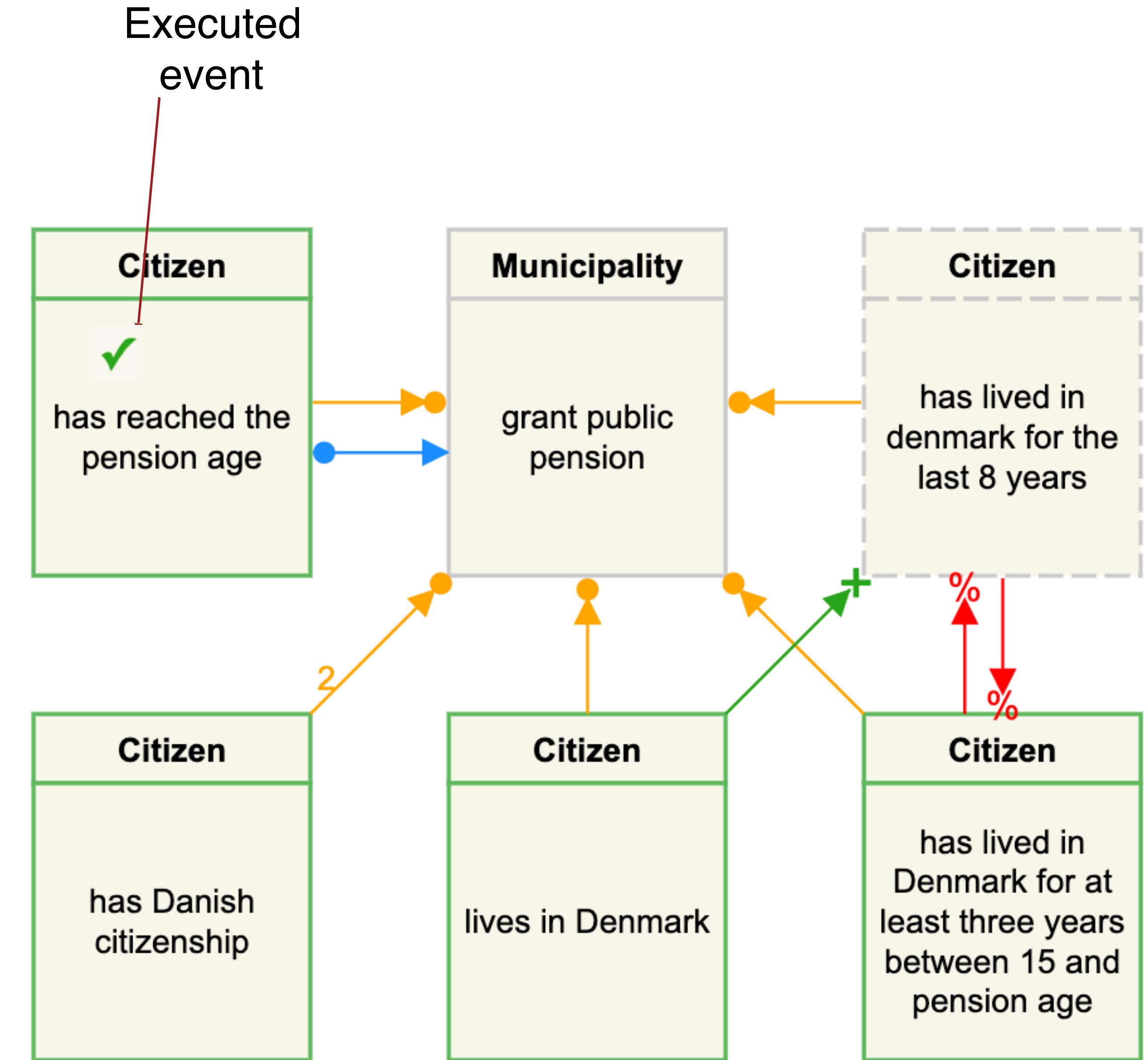
$h ::= f \mid t_0$

$i ::= f \mid t$

$p ::= f \mid 0 \mid t_\omega$

$t_0 \in \mathbb{N} \cup \{0\}$ 0-time

$t_\omega \in \mathbb{N} \cup \{\omega\}$ ω -time



Timed DCR graphs - briefly

- Assume:
 - A fixed universe of events \mathcal{E} , and labels \mathcal{L}
 - $e, f \in \mathcal{E}$
 - A mapping λ from events to labels
 - tick $\notin \mathcal{E}$
- The grammar for DCR process is given as:

$$P, Q ::= [M] \lambda T \quad \text{process}$$

$$T, U ::= e \xrightarrow{t_0} f \quad | \quad e \xrightarrow{t_\omega} f \\ | \quad e \xrightarrow{+} f \quad | \quad e \xrightarrow{\%} f \\ | \quad e \xrightarrow{\diamond} f \quad | \quad T \parallel U \\ | \quad 0$$

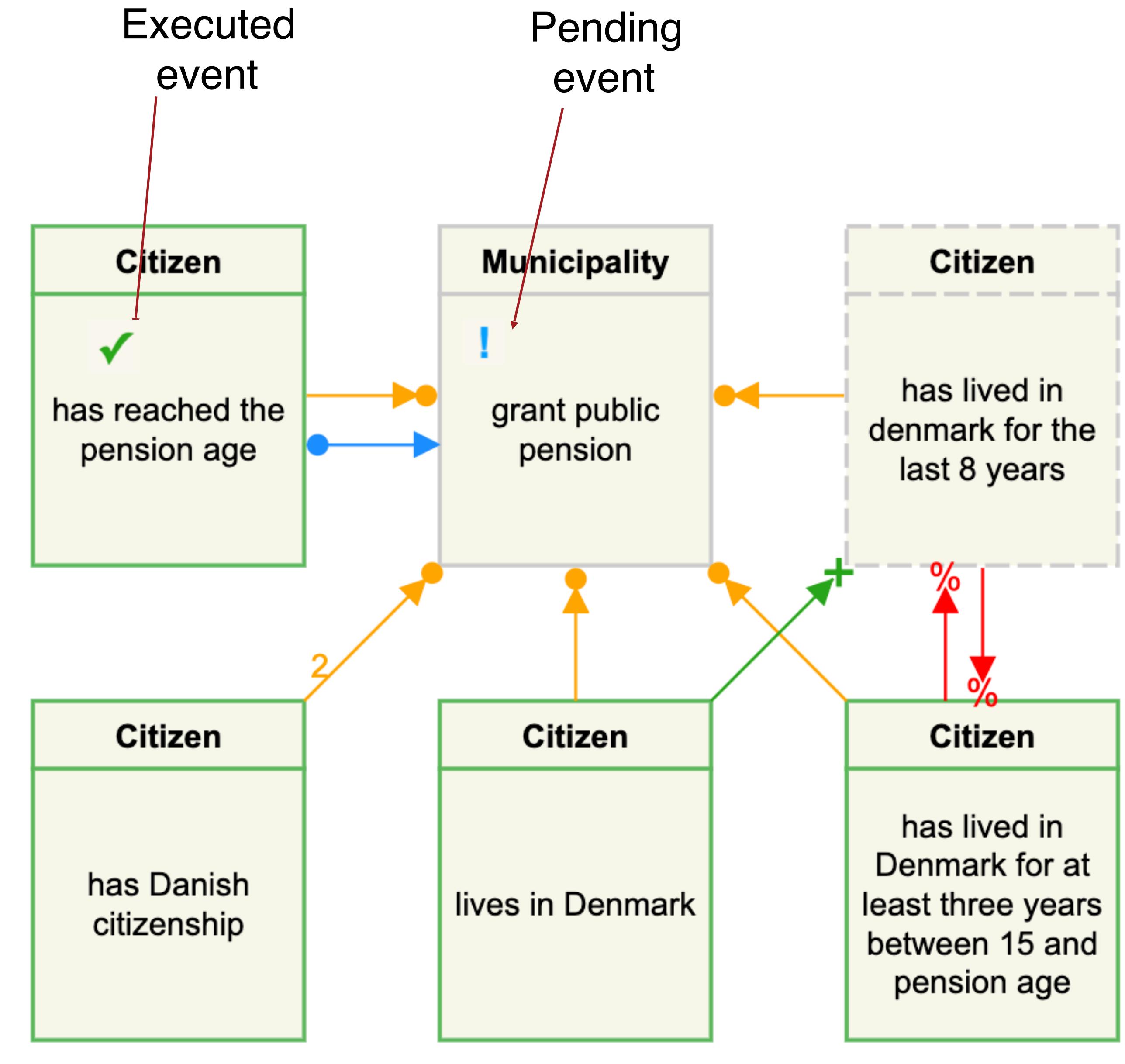
$$M, N ::= M, e : \Phi \mid \epsilon \quad \text{marking} \\ \lambda ::= \lambda, e : l \mid \epsilon \quad \text{labelling}$$

$$\Phi ::= (h, i, p)$$

$$h ::= f \mid t_0$$

$$i ::= f \mid t$$

$$p ::= f \mid 0 \mid t_\omega$$

$$t_0 \in \mathbb{N} \cup \{0\} \quad \text{0-time} \\ t_\omega \in \mathbb{N} \cup \{\omega\} \quad \omega\text{-time}$$


Timed DCR graphs - briefly

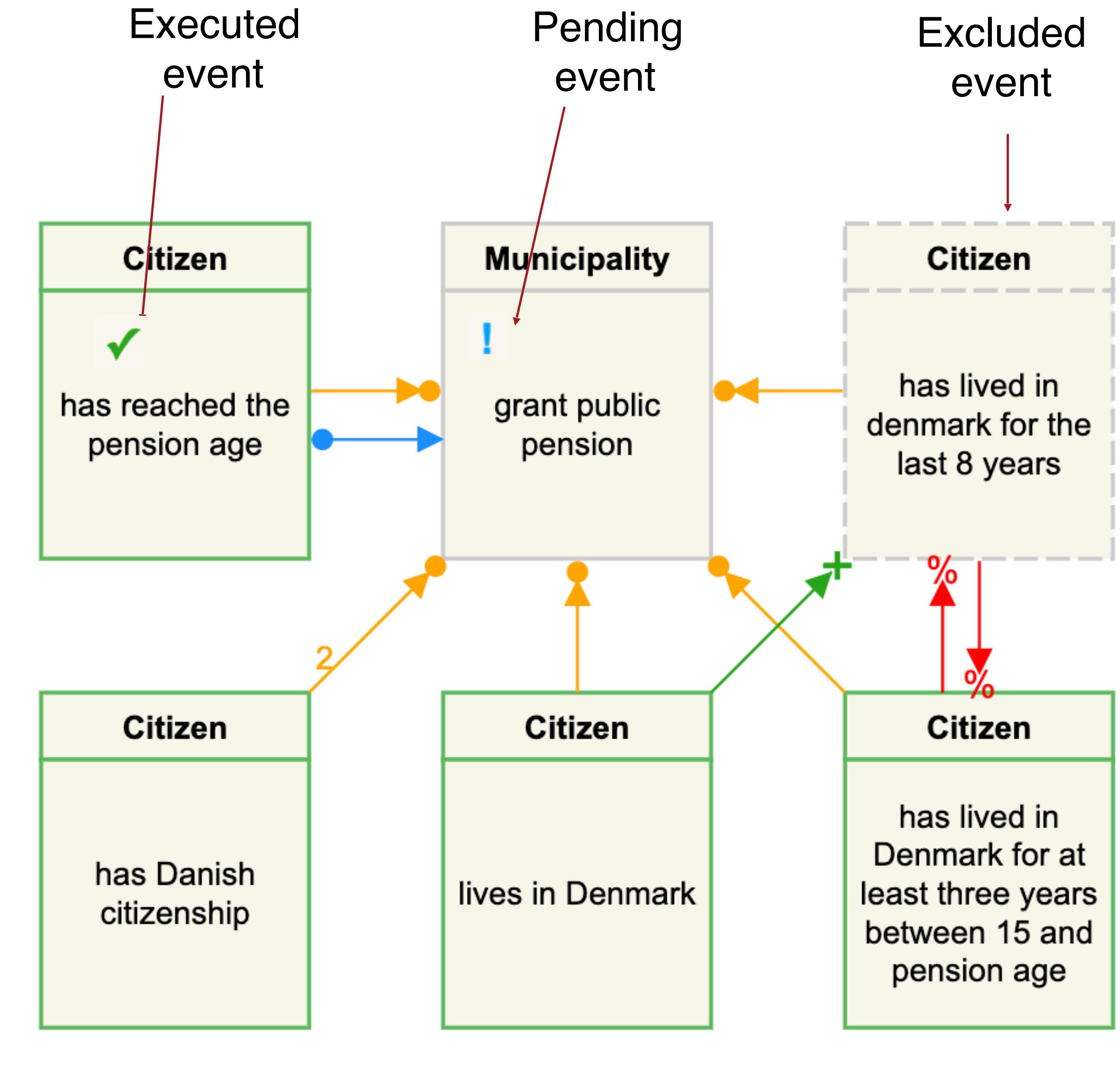
- Assume:
 - A fixed universe of events \mathcal{E} , and labels \mathcal{L}
 - $e, f \in \mathcal{E}$
 - A mapping λ from events to labels
 - tick $\notin \mathcal{E}$
- The grammar for DCR process is given as:

$$P, Q ::= [M] \lambda T \quad \text{process}$$

$$T, U ::= e \xrightarrow{t_0} f \quad | \quad e \xrightarrow{t_\omega} f \\ | \quad e \xrightarrow{+} f \quad | \quad e \xrightarrow{\%} f \\ | \quad e \xrightarrow{\diamond} f \quad | \quad T \parallel U \\ | \quad 0$$

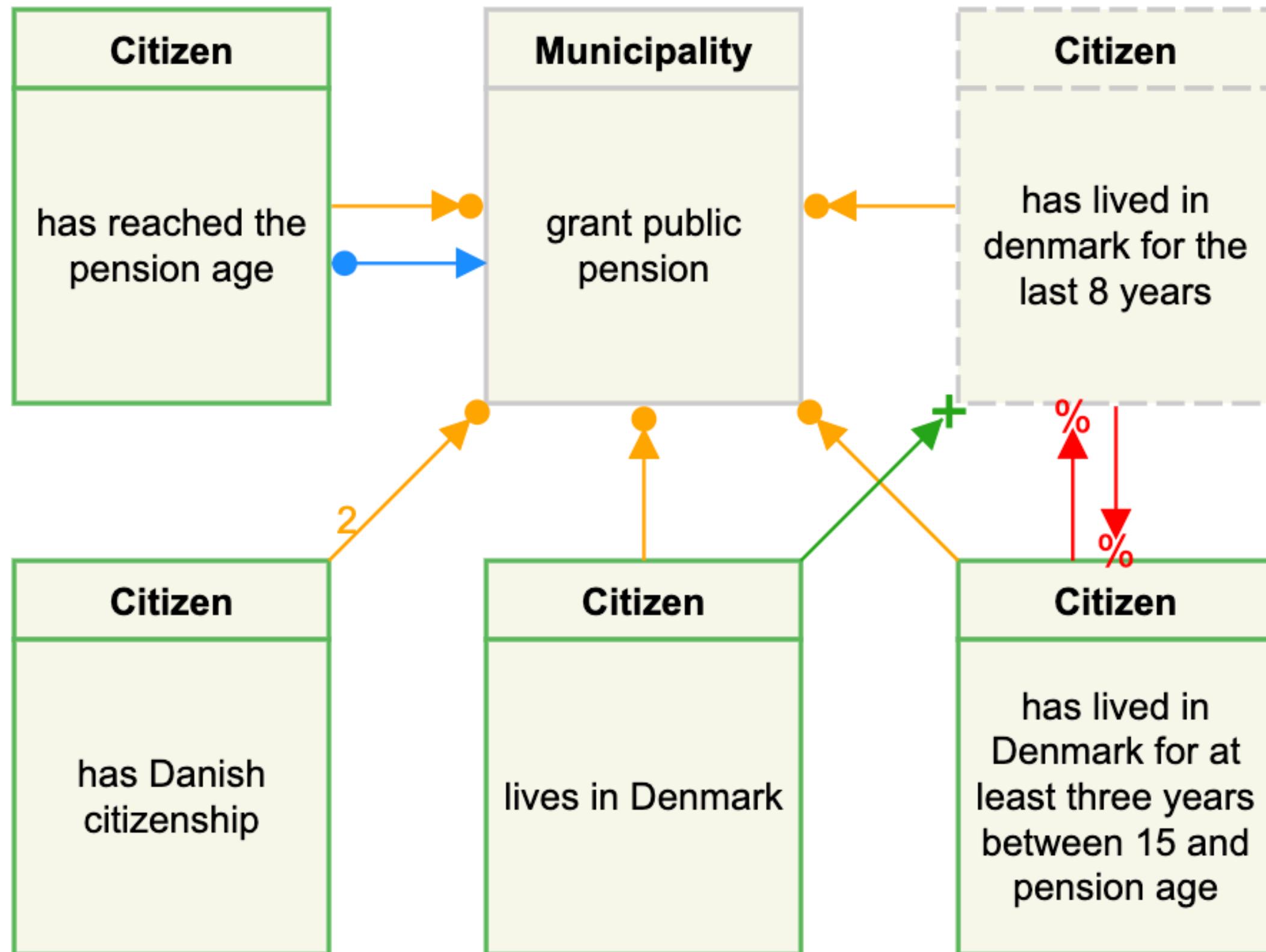
$$M, N ::= M, e : \Phi \mid \epsilon \quad \text{marking} \\ \lambda ::= \lambda, e : l \mid \epsilon \quad \text{labelling}$$

$$\Phi ::= (h, i, p)$$

$$h ::= f \mid t_0 \quad t_0 \in \mathbb{N} \cup \{0\} \quad \text{0-time} \\ i ::= f \mid t \quad t_\omega \in \mathbb{N} \cup \{\omega\} \quad \omega\text{-time} \\ p ::= f \mid 0 \mid t_\omega$$


Operational Semantics: Enabledness and Effects

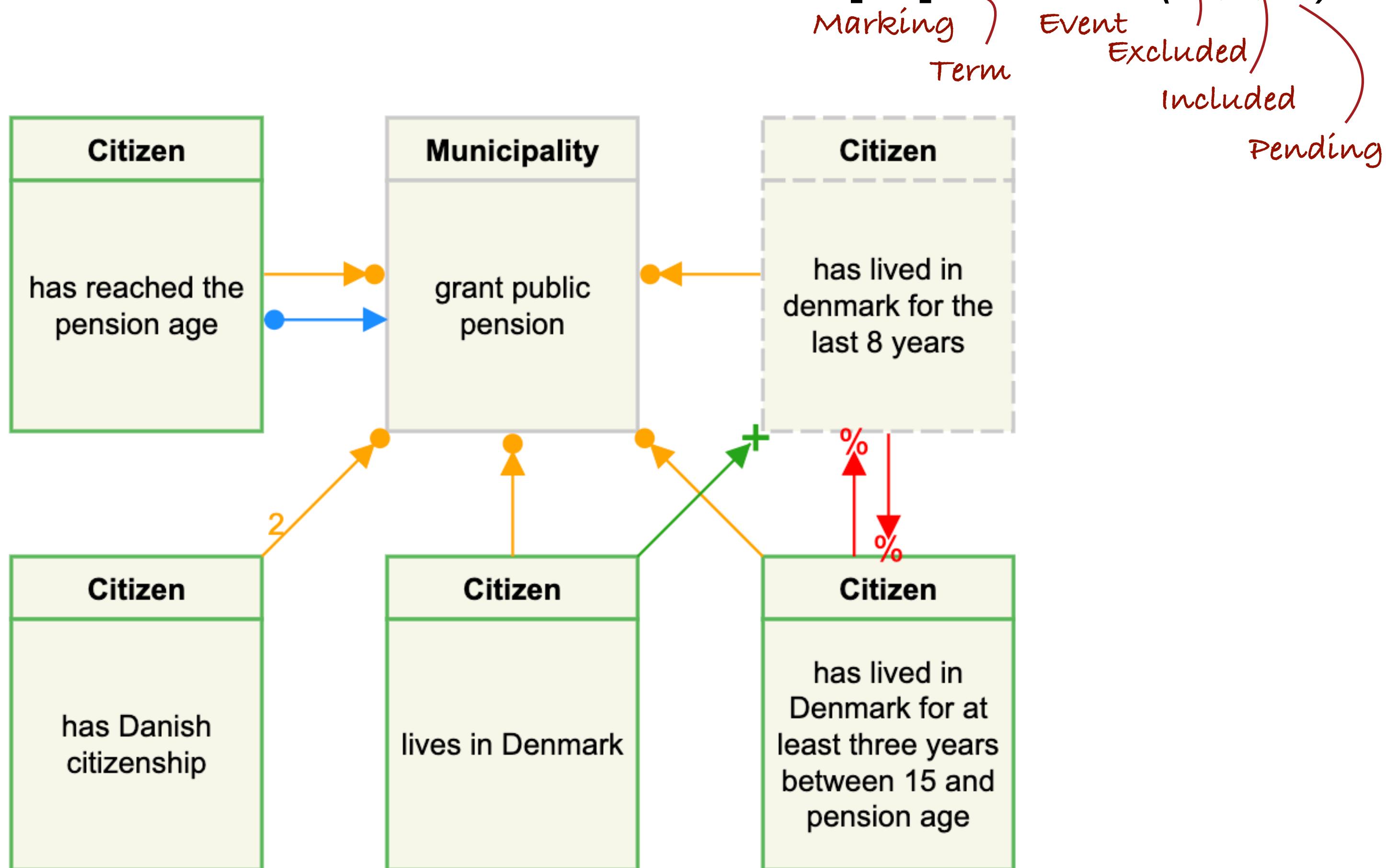
- Enabled events & effects $[M] \ T \vdash f : (E, I, P)$:



$k=0$

Operational Semantics: Enabledness and Effects

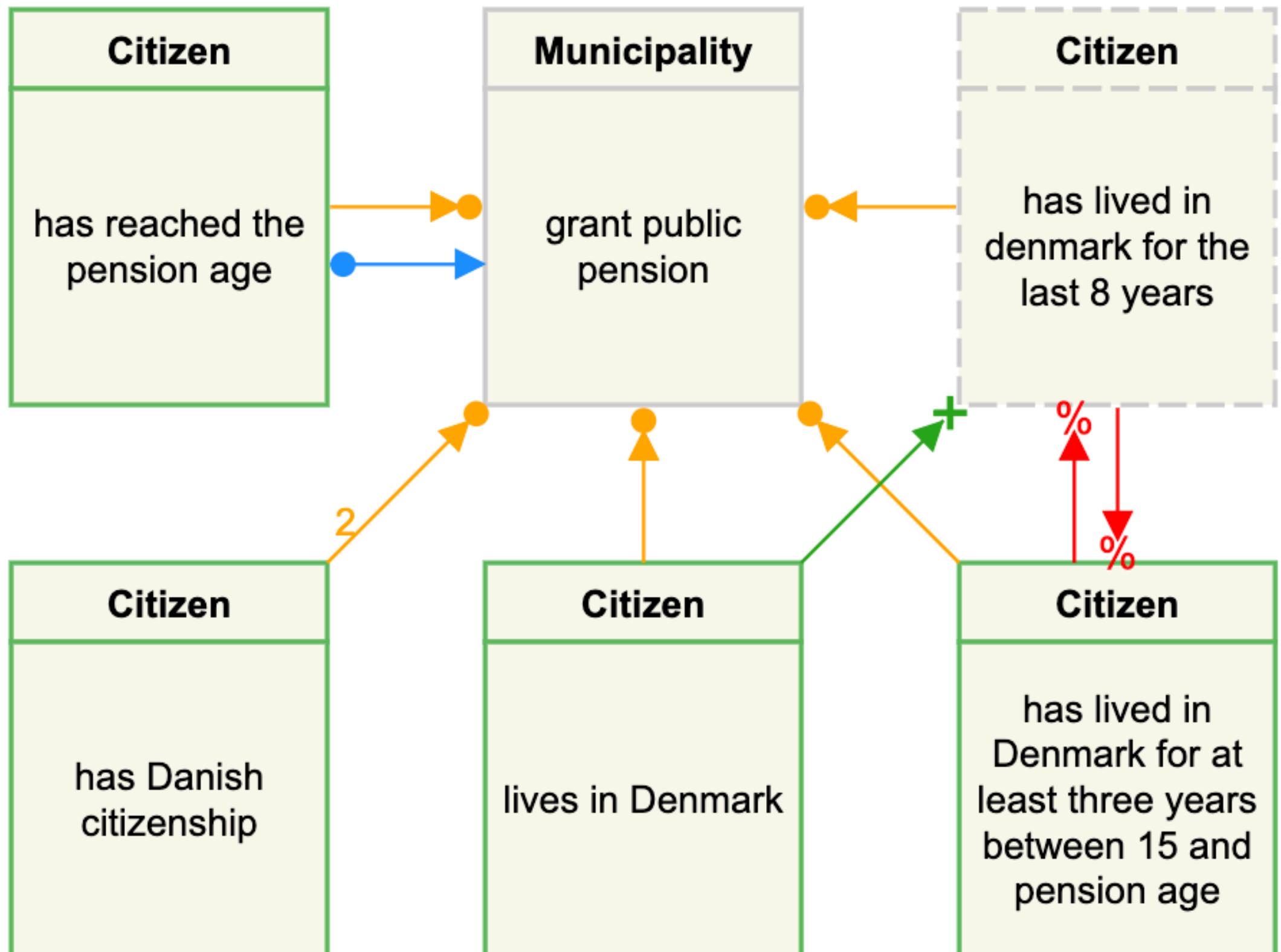
- Enabled events & effects $[M] \vdash f : (E, I, P)$:



$k=0$

Operational Semantics: Enabledness and Effects

- Enabled events & effects $[M] T \vdash f : (E, I, P)$:



Event f is **enabled** iff

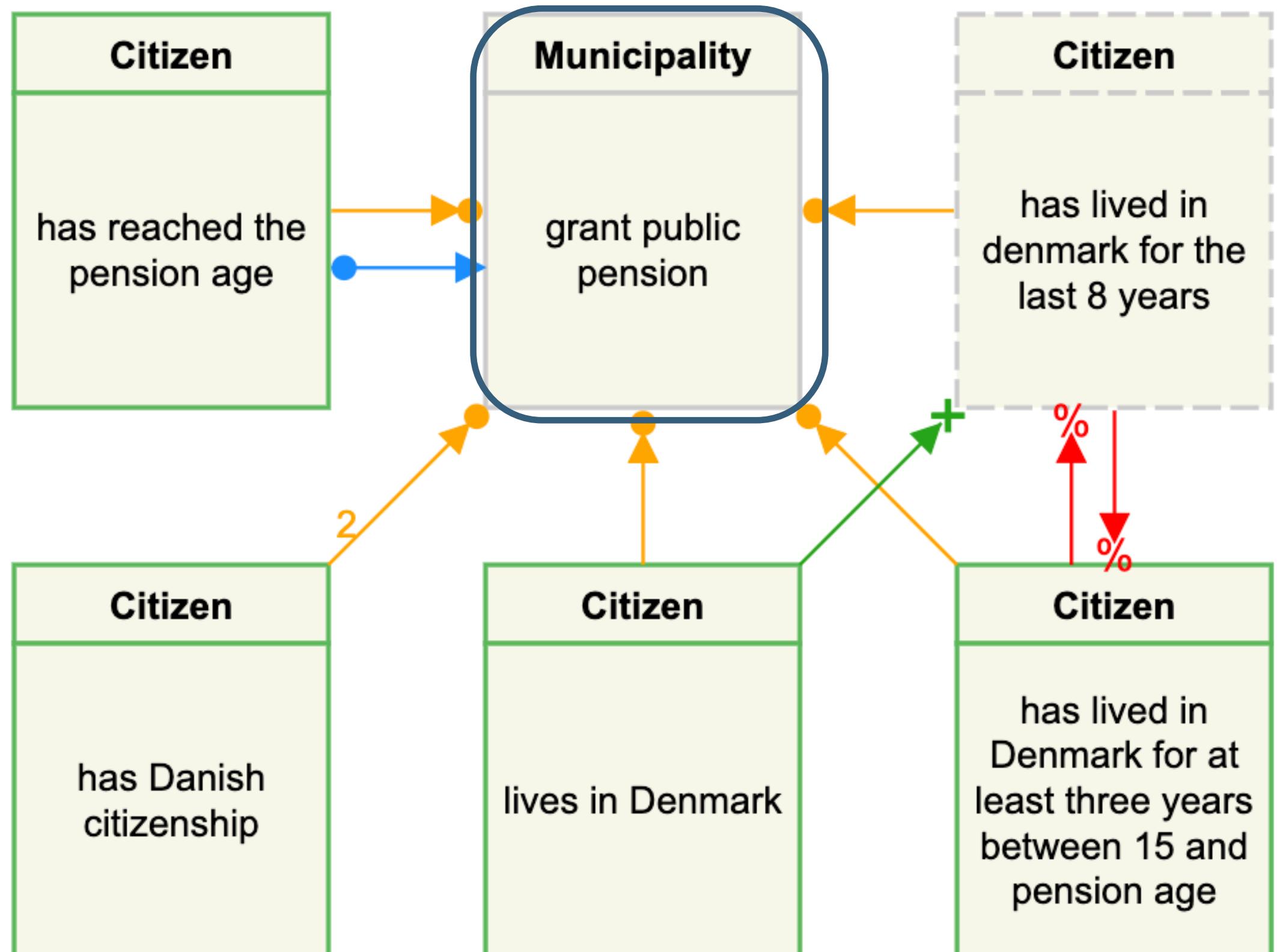
$$i \Rightarrow h \geq k$$

$$[M, e : (h, i, -), f : (-, t, -)] e \xrightarrow{k} f \vdash \boxed{f} : (\emptyset, \emptyset, \emptyset)$$

$k=0$

Operational Semantics: Enabledness and Effects

- Enabled events & effects $[M] T \vdash f : (E, I, P)$:



Event f is **enabled** iff

- f is included

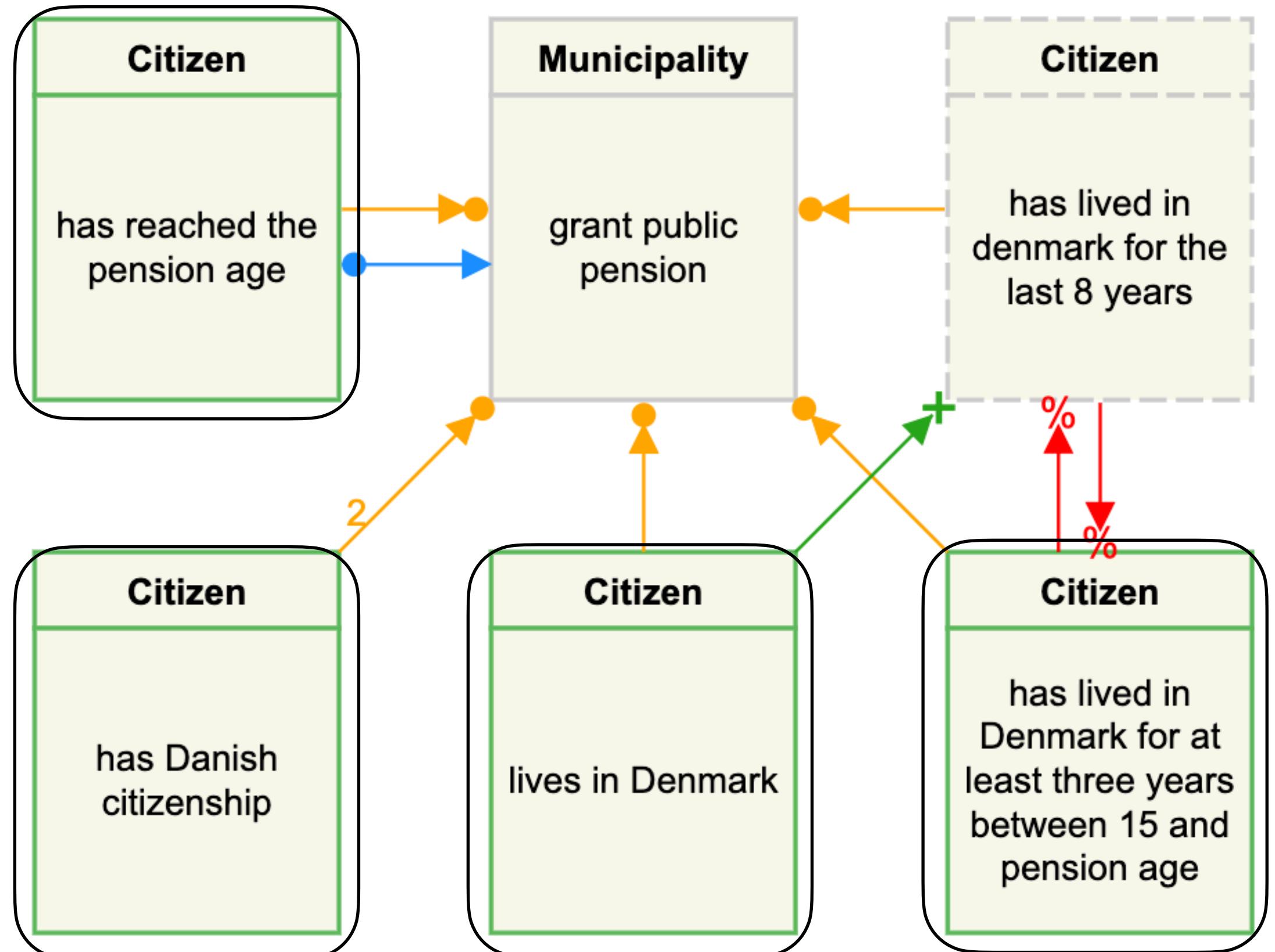
$$i \Rightarrow h \geq k$$

$$[M, e : (h, i, -), f : (-, t, -)] e \xrightarrow{k} f \vdash f : (\emptyset, \emptyset, \emptyset)$$

$k=0$

Operational Semantics: Enabledness and Effects

- Enabled events & effects $[M] T \vdash f : (E, I, P)$:



Event f is **enabled** iff

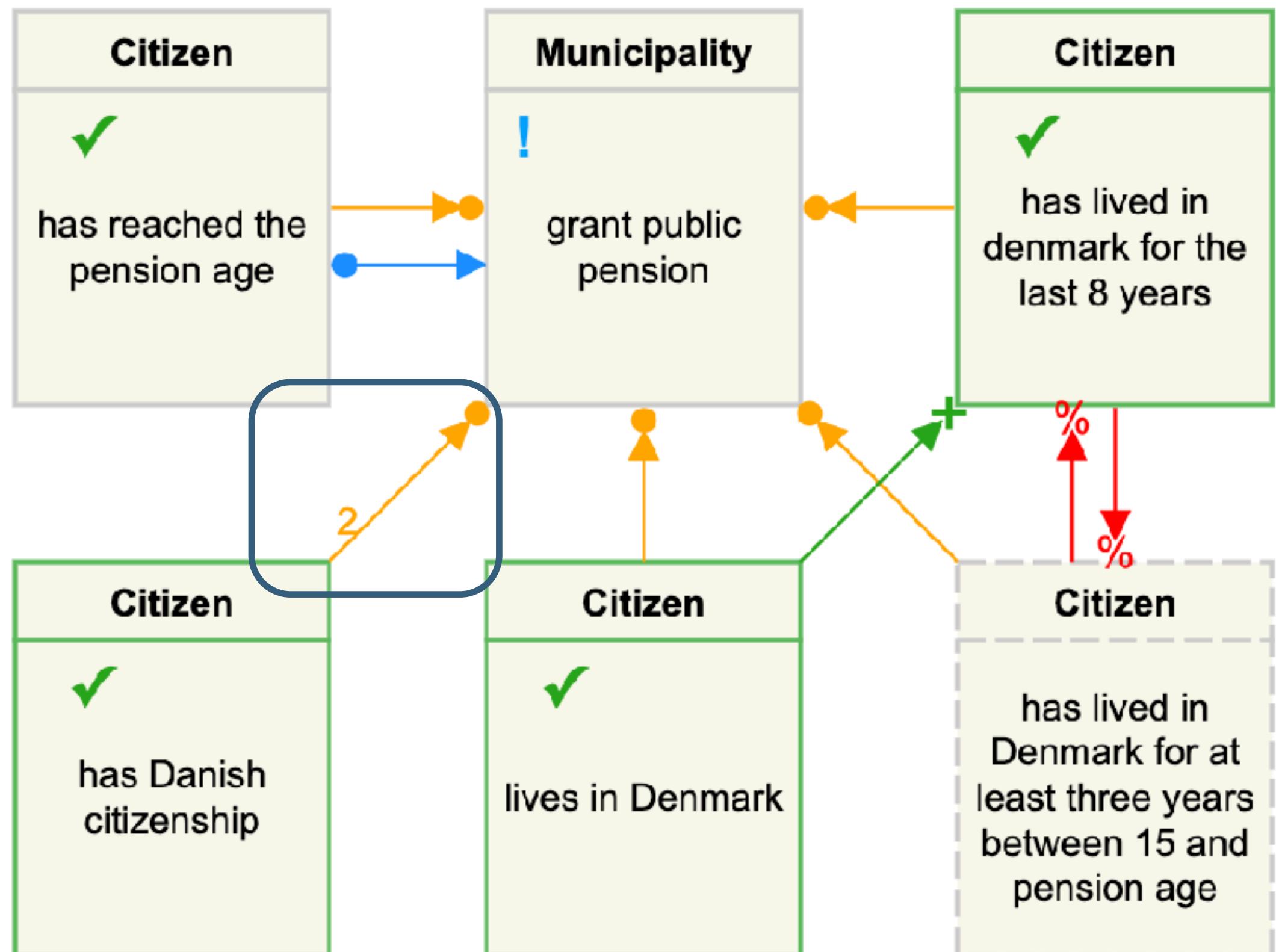
- f is included
- Its included preconditions have been executed or excluded

$$i \Rightarrow h \geq k$$

$$[M, e : (h, i, -), f : (-, t, -)] e \xrightarrow{k} f : (\emptyset, \emptyset, \emptyset)$$

Operational Semantics: Enabledness and Effects

- Enabled events & effects $[M] T \vdash f : (E, I, P)$:



Event f is **enabled** iff

- f is included
- Its included preconditions have been executed or excluded
- If it depends on a time condition, this has been executed at least k steps ago

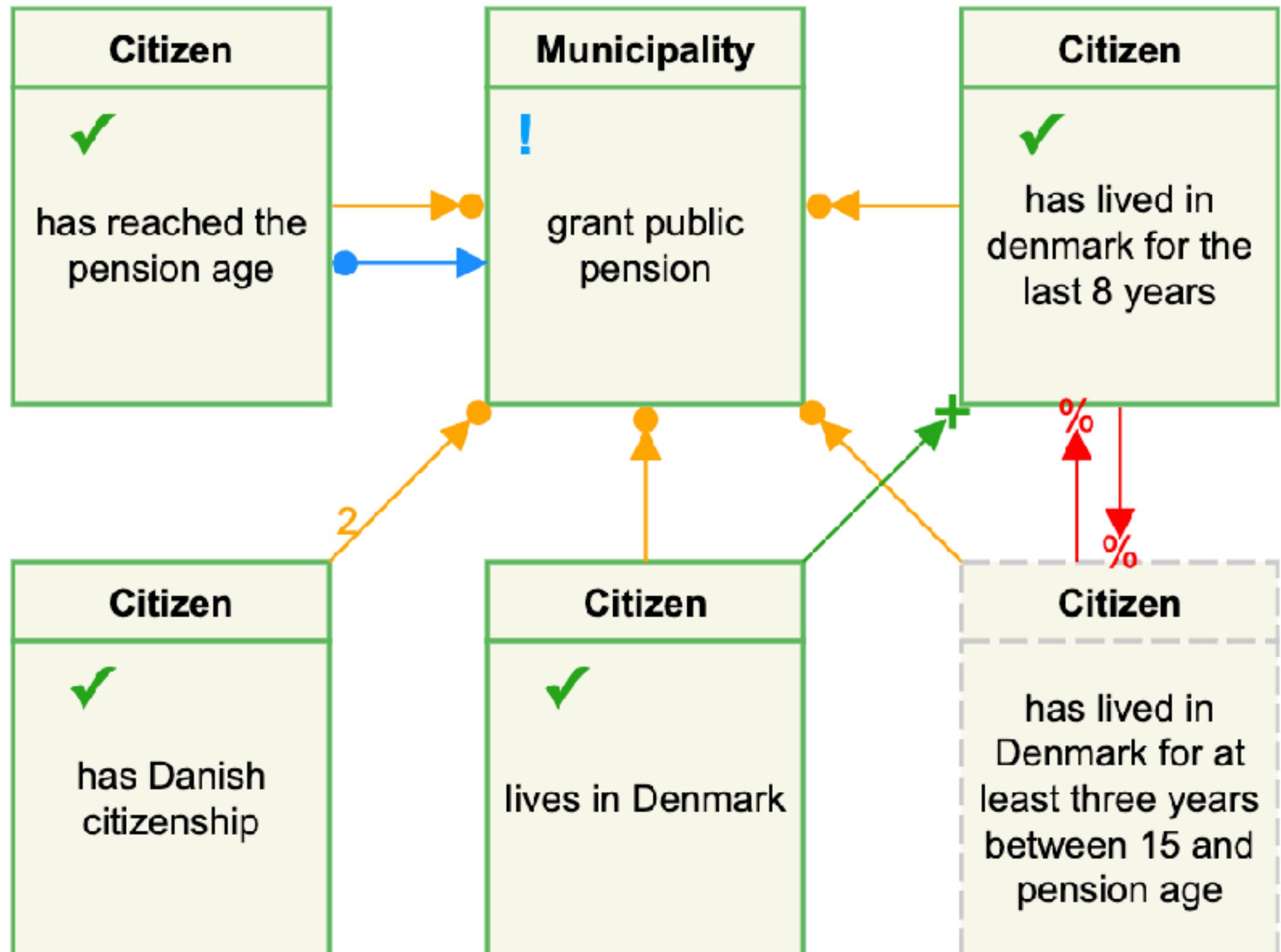
$$i \Rightarrow h \geq k$$

$$[M, e : (h, i, -), f : (-, t, -)] e \xrightarrow{k} f : (\emptyset, \emptyset, \emptyset)$$

$k=0$

Operational Semantics: Enabledness and Effects

- Enabled events & effects $[M] T \vdash f : (E, I, P)$:



- Event f is **enabled** iff
- f is included
 - Its included preconditions have been executed or excluded
 - If it depends on a time condition, this has been executed at least k steps ago

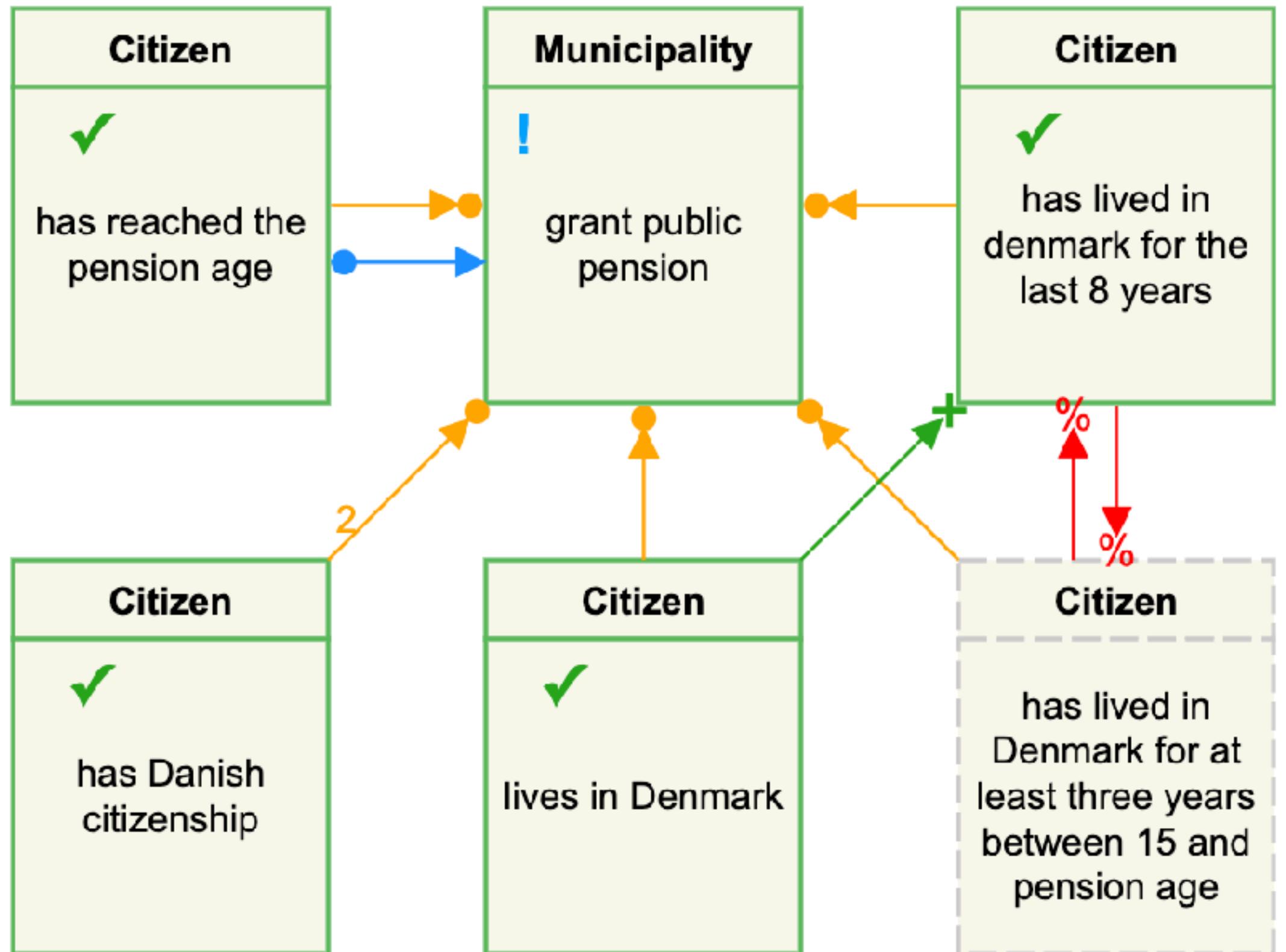
$$i \Rightarrow h \geq k$$

$$[M, e : (h, i, -), f : (-, t, -)] e \xrightarrow{k} f : (\emptyset, \emptyset, \emptyset)$$

$k=2$

Operational Semantics: Enabledness and Effects

- Enabled events & effects $[M] T \vdash f : (E, I, P)$:



Event f is **enabled** iff

- f is included
- Its included preconditions have been executed or excluded
- If it depends on a time condition, this has been executed at least k steps ago

$k=2$

$$i \Rightarrow h \geq k$$

$$[M, e : (h, i, -), f : (-, t, -)] e \xrightarrow{k} f \vdash f : (\emptyset, \emptyset, \emptyset)$$

$$[M, e : (-, t, -)] e \xrightarrow{k} f \vdash e : (\emptyset, \emptyset, \{f : k\})$$

)
imposed obligations

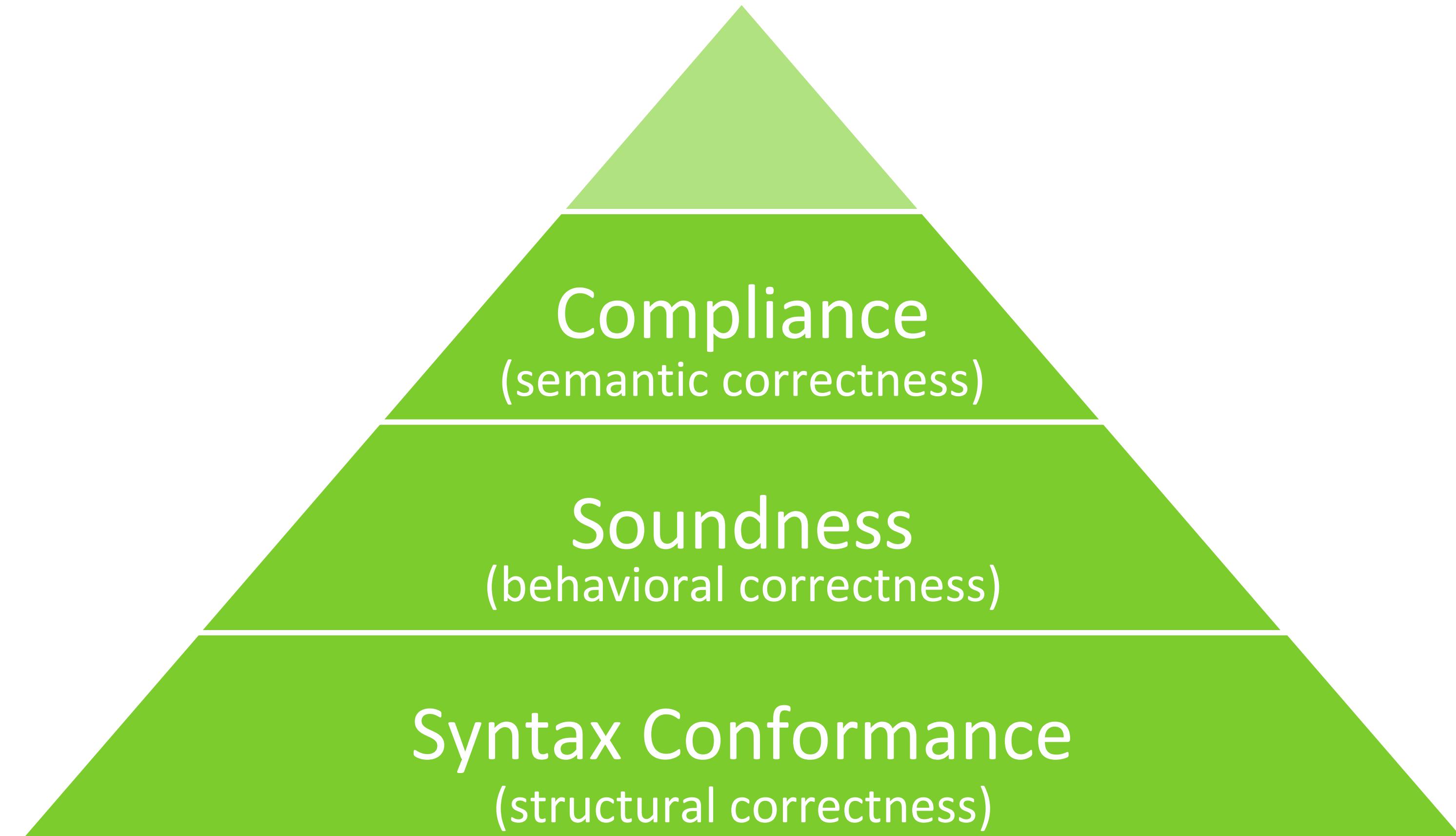
Operational Semantics: Timed Transitions

$$\frac{[M] T \vdash e : \delta}{T \vdash M \xrightarrow{e} \delta\langle e\langle M \rangle \rangle} \quad [\text{EVENT}]$$

$$\frac{\text{deadline}\langle M \rangle > 0}{T \vdash M \xrightarrow{\text{tick}} \text{tick}\langle M \rangle} \quad [\text{TIME}]$$

- LTS keeps track of the sequence of fired events & time changes
- $e\langle M \rangle$: effect of executing e in marking M
- $e : \delta = (Ex, In, Pe)$: effects of executing e
- $\text{deadline}\langle M \rangle$: modify pending/executed markings.

Correctness in business processes



Regulations and processes: Understanding the balance



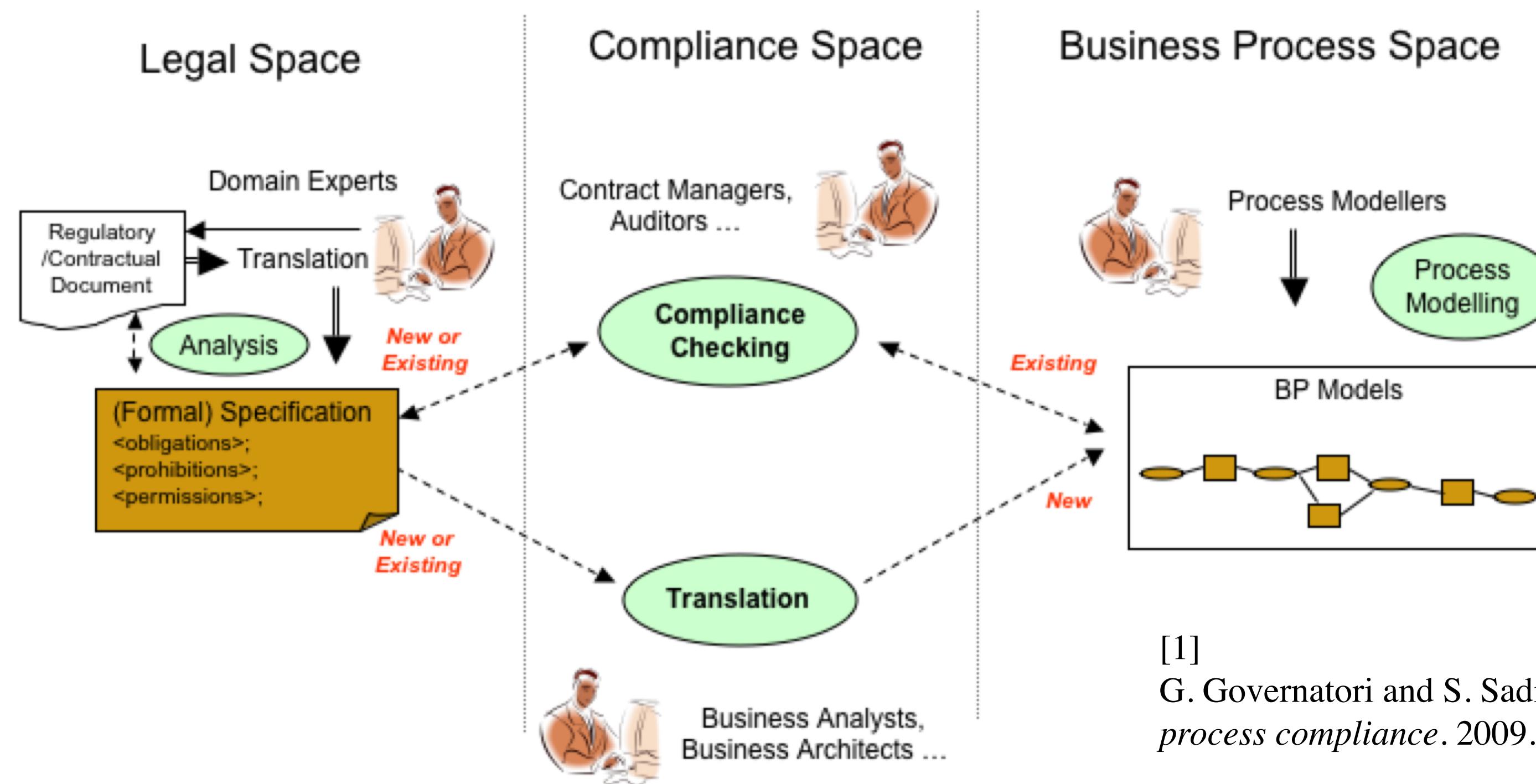
Governmental regulations	Business processes
Issued by legal authorities	Designed and implemented by organizations
National perspective	Organizational perspective
Focus on the effects of actions	Focus on the process of actions
Increase social welfare; economics, safety, environmental concern	Increase the efficiency of organizations to achieve higher organizational benefit
Legal authority	Secondary authority

[1]

J. Jiang, H. Aldewereld, V. Dignum, S. Wang, and Z. Baida, "Regulatory compliance of business processes," *AI & Society*, vol. 30, no. 3, pp. 393–402, Aug. 2015.

Compliance Checking

- Matches the specification of compliance rules and the executions of business processes.
- If specifications are satisfied by the runs in a BP, the process is **compliant**, otherwise, it is **in violation** with the compliance rule.



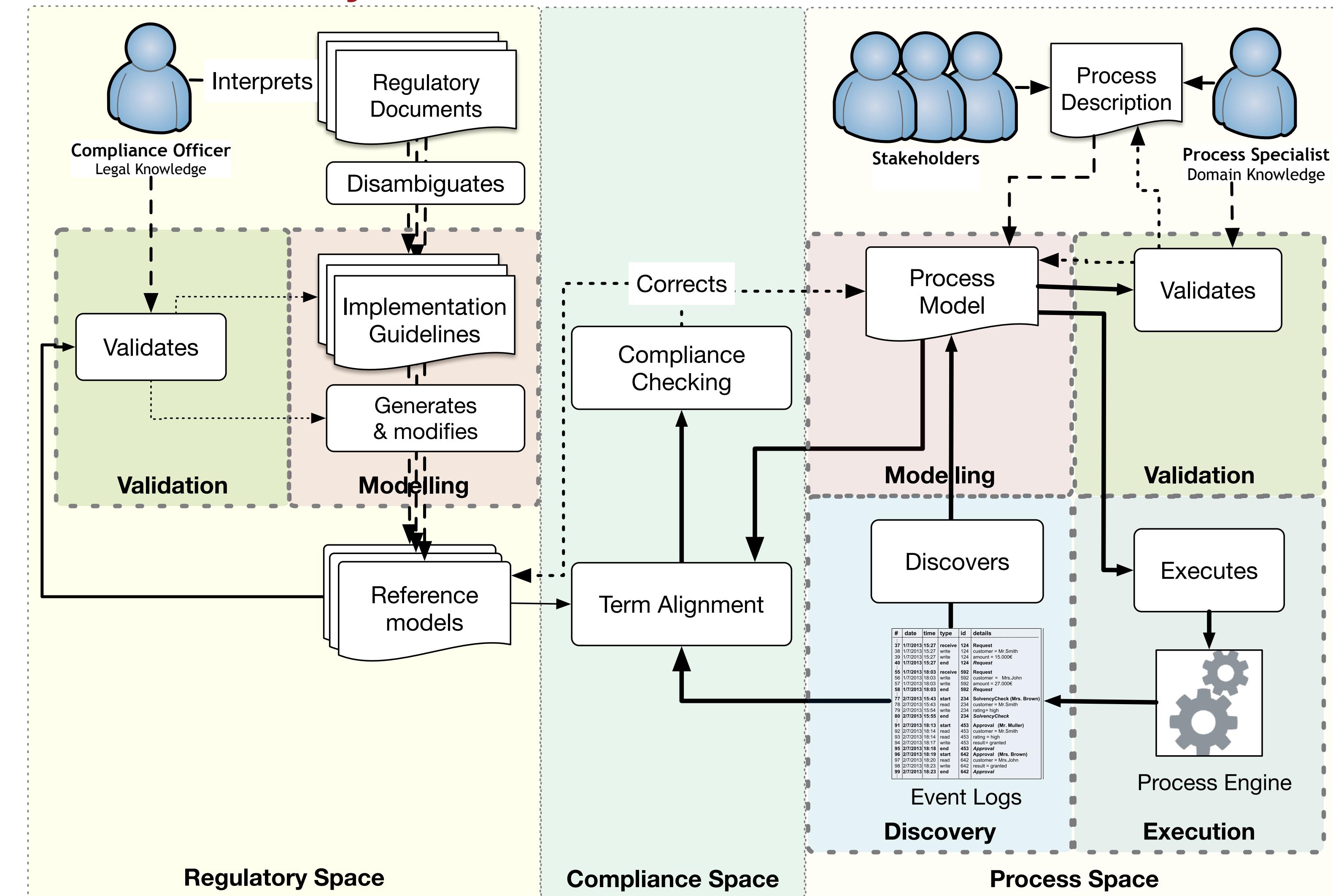
[1]

G. Governatori and S. Sadiq, *The journey to business process compliance*. 2009.

Verifying Compliance

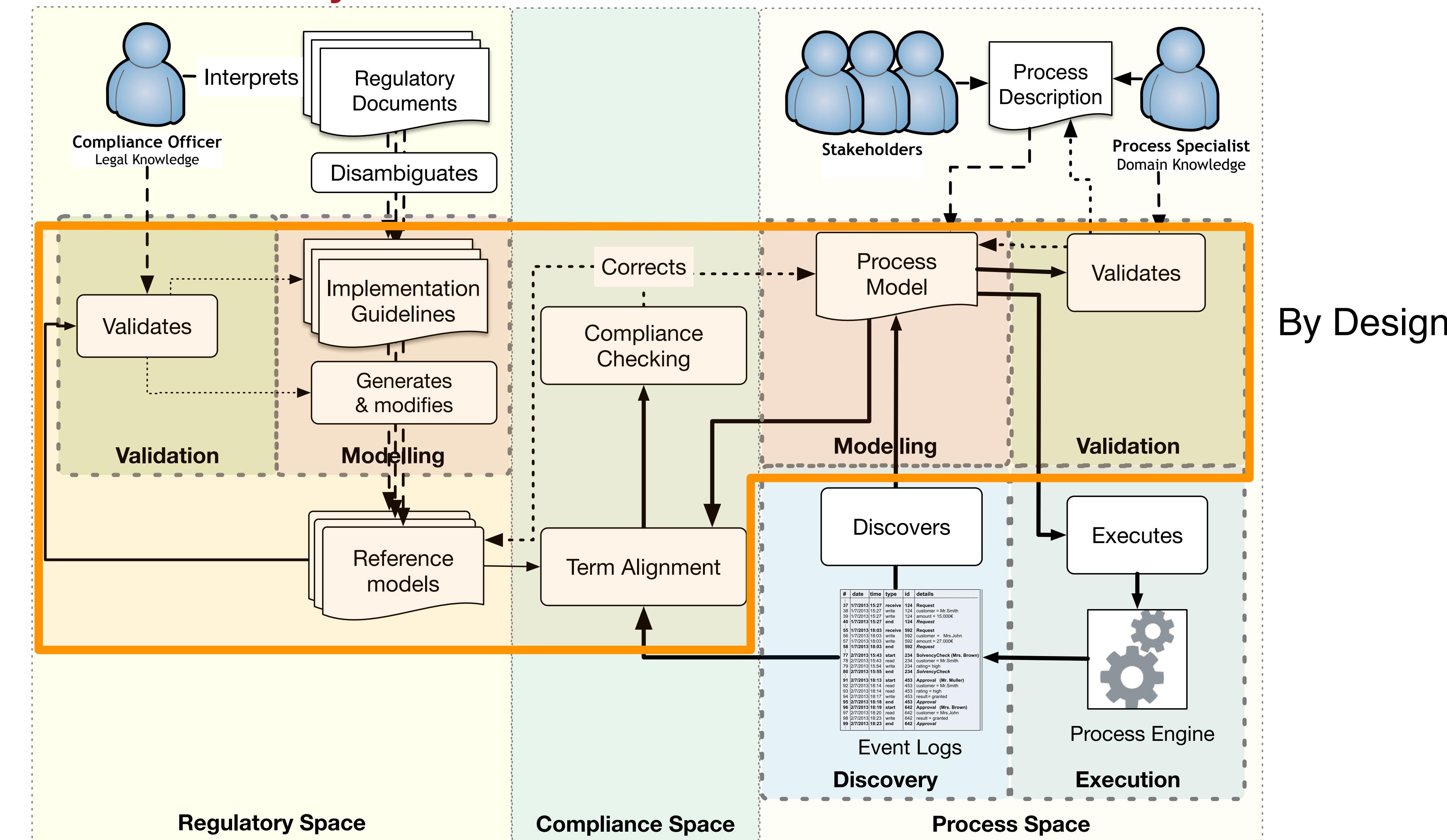
- **Policies against traces:** Conformance Checking
- **Policies against models:** Refinement

The Compliance Lifecycle



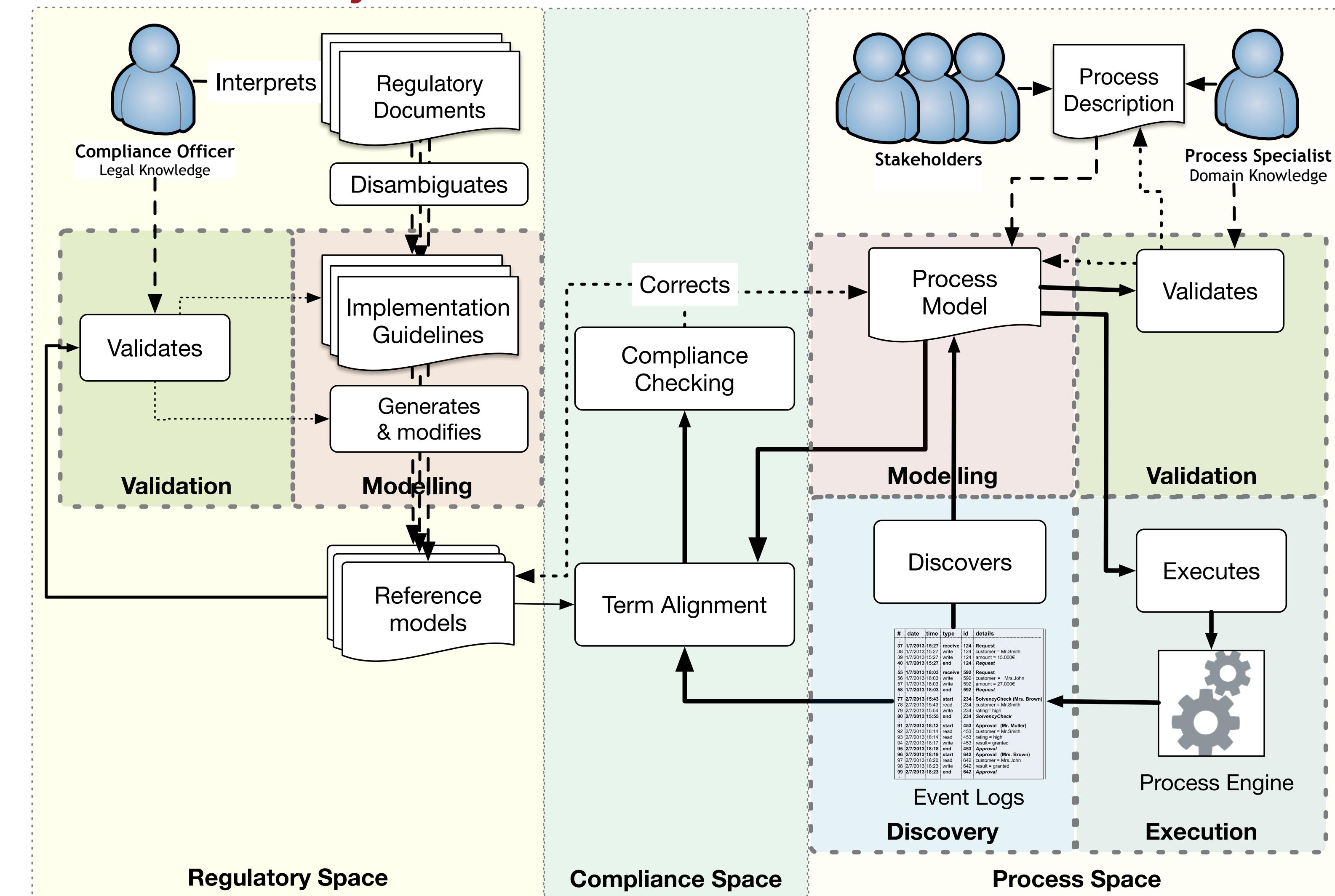
[1] H.A. López, S. Debois, T. Slaats and T. Hildebrandt, Business Process Compliance using Reference Models of Law. In *Fundamental Aspects of Software Engineering*, 2020.

The Compliance Lifecycle



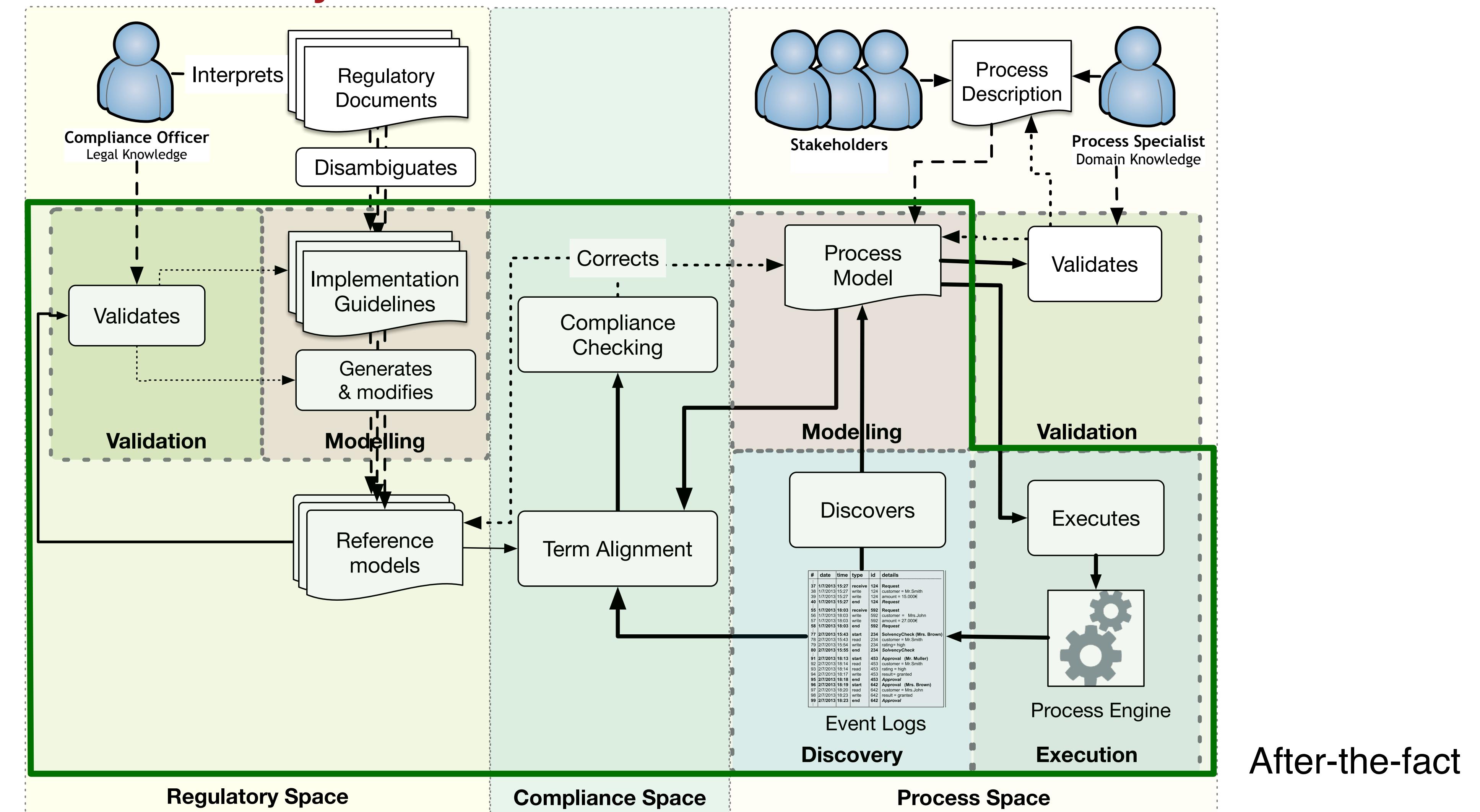
[1] H.A. López, S. Debois, T. Slaats and T. Hildebrandt, Business Process Compliance using Reference Models of Law. In *Fundamental Aspects of Software Engineering*, 2020.

The Compliance Lifecycle



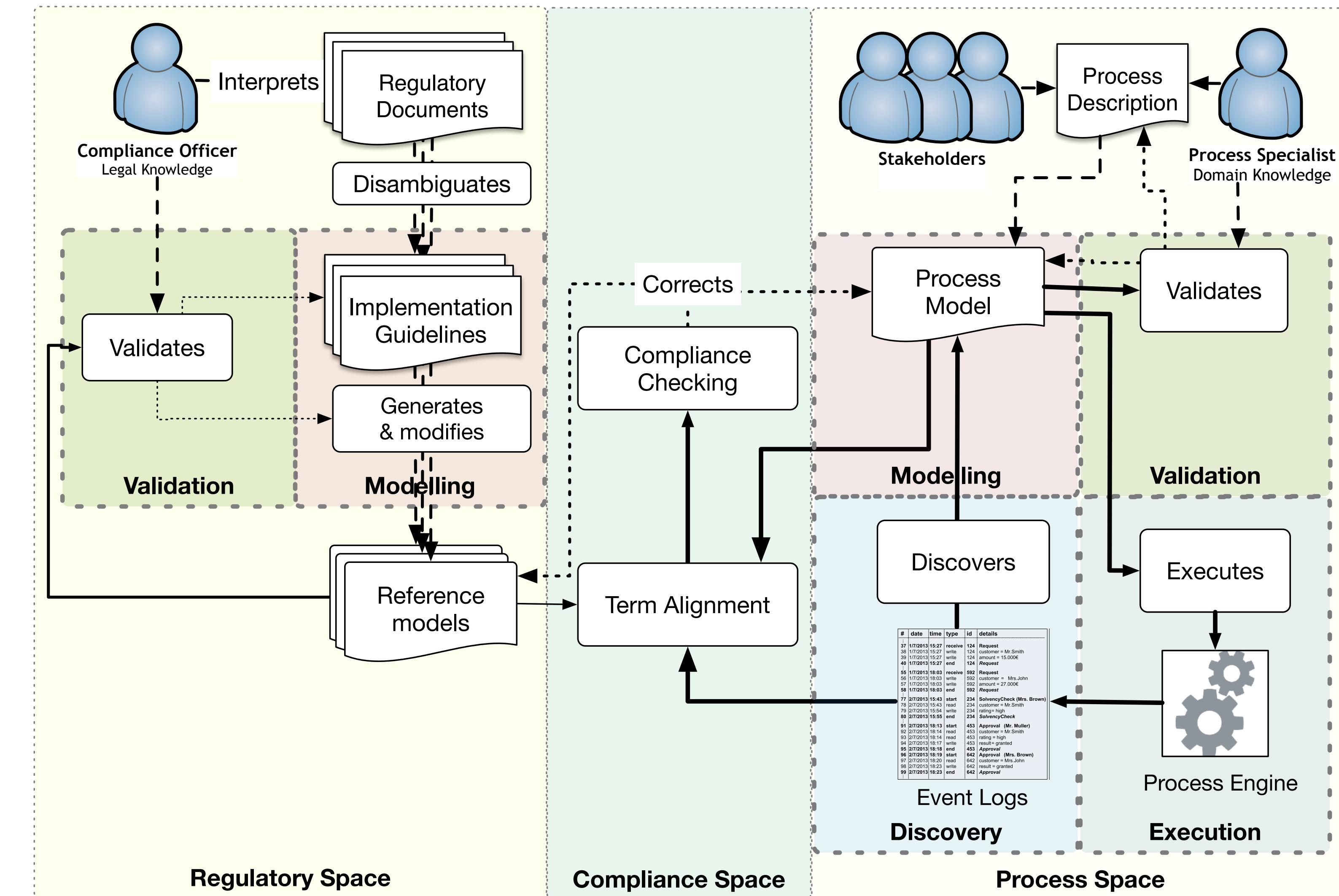
[1] H.A. López, S. Debois, T. Slaats and T. Hildebrandt, Business Process Compliance using Reference Models of Law. In *Fundamental Aspects of Software Engineering*, 2020.

The Compliance Lifecycle

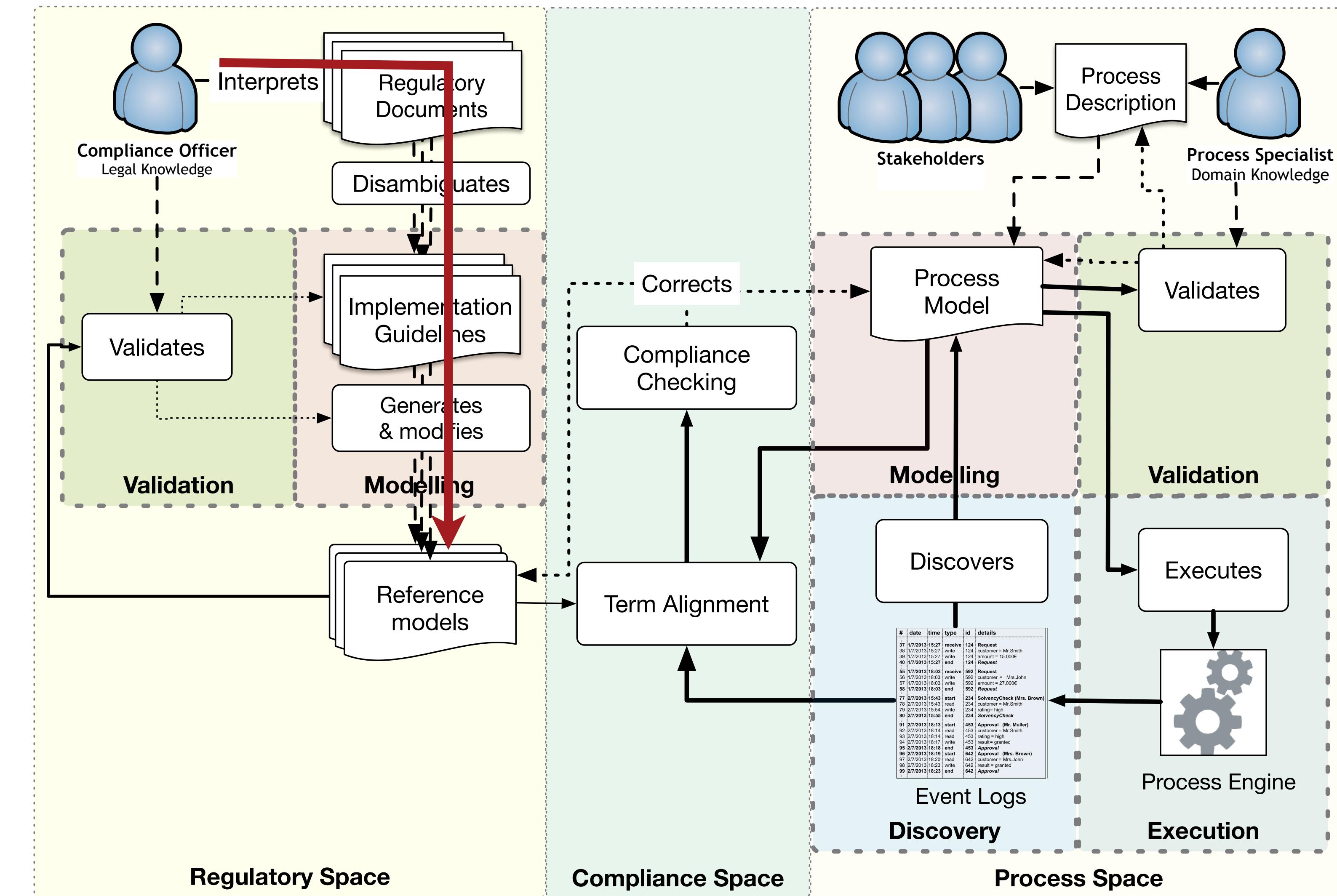


[1] H.A. López, S. Debois, T. Slaats and T. Hildebrandt, Business Process Compliance using Reference Models of Law. In *Fundamental Aspects of Software Engineering*, 2020.

Expressing Compliance Policies



Expressing Compliance Policies

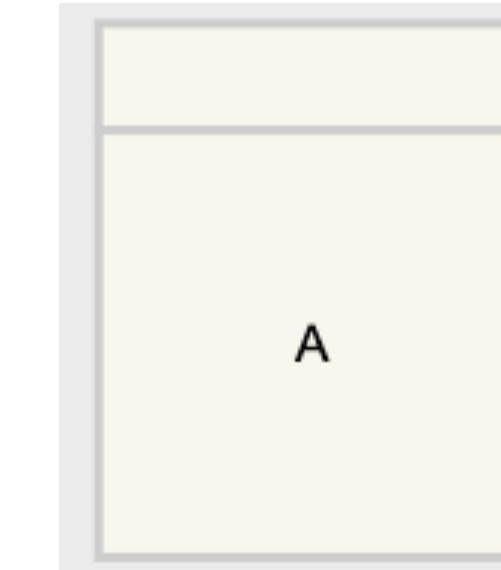


1. Expressing Policies as DCR graphs

Occurrence: The occurrence of task **A** is within the scope of the process.

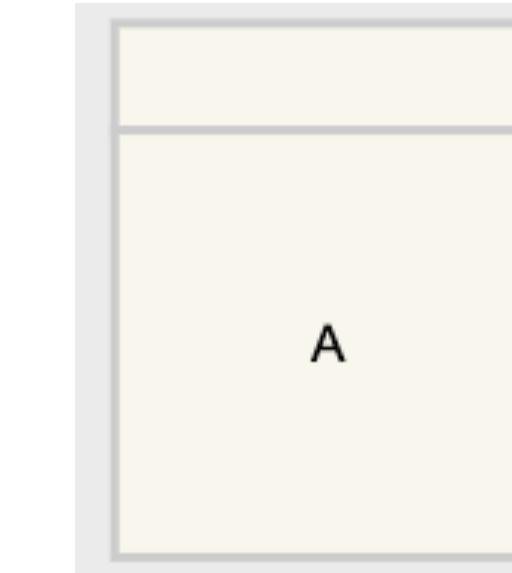
1. Expressing Policies as DCR graphs

Occurrence: The occurrence of task **A** is within the scope of the process.



1. Expressing Policies as DCR graphs

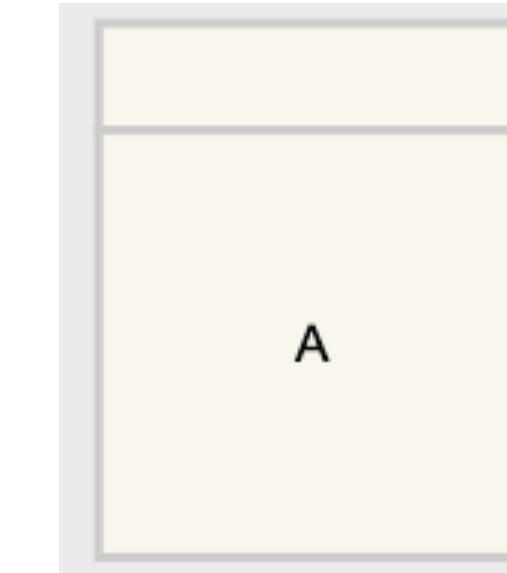
Occurrence: The occurrence of task **A** is within the scope of the process.



Inclusion: Presence of a given event **A** mandates **B** to be also present

1. Expressing Policies as DCR graphs

Occurrence: The occurrence of task **A** is within the scope of the process.

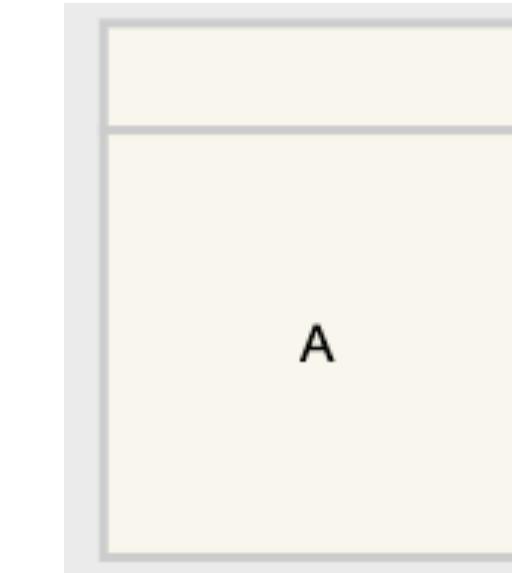


Inclusion: Presence of a given event **A** mandates **B** to be also present

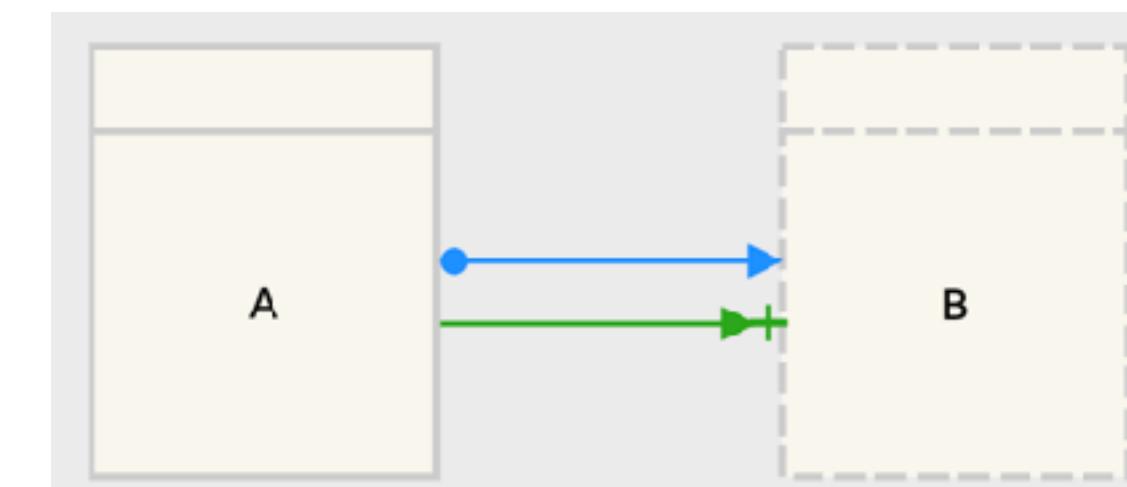


1. Expressing Policies as DCR graphs

Occurrence: The occurrence of task **A** is within the scope of the process.



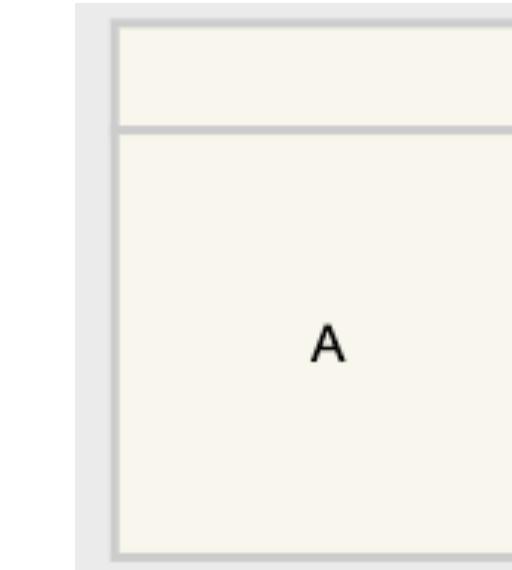
Inclusion: Presence of a given event **A** mandates **B** to be also present



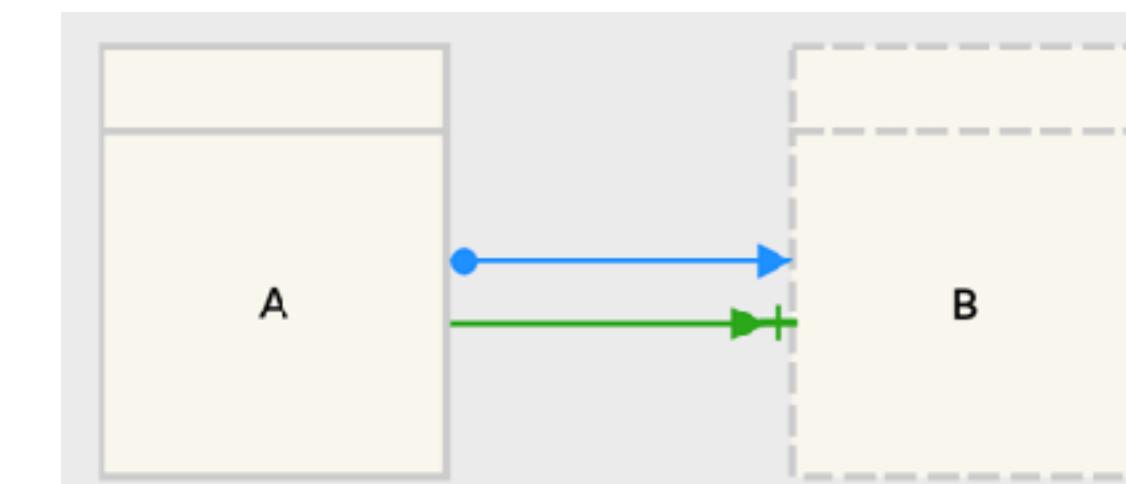
Response: Limits the occurrence of a given event **B** in response to a given event **A**

1. Expressing Policies as DCR graphs

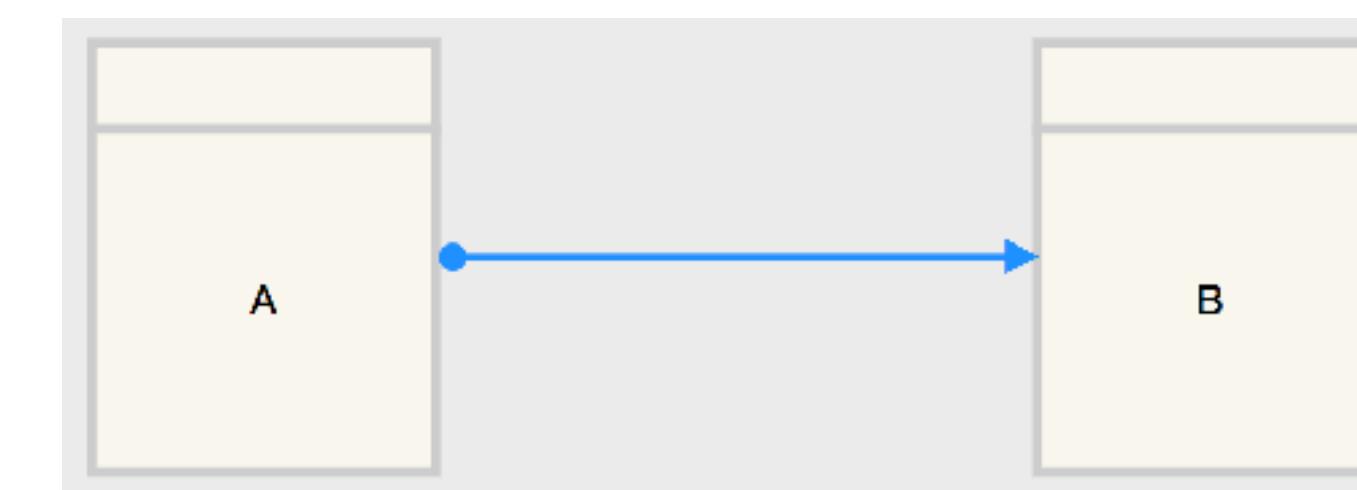
Occurrence: The occurrence of task **A** is within the scope of the process.



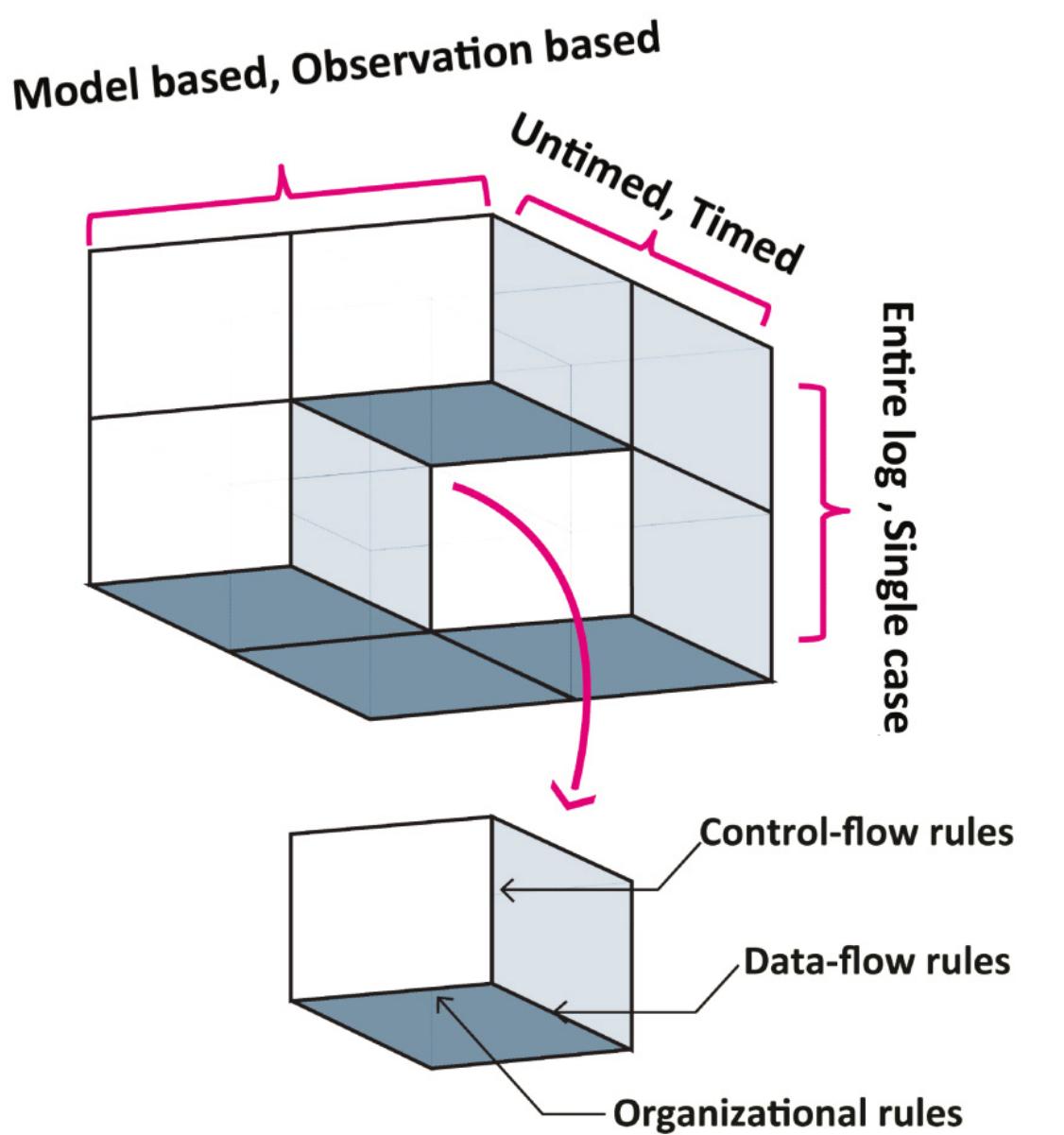
Inclusion: Presence of a given event **A** mandates **B** to be also present



Response: Limits the occurrence of a given event **B** in response to a given event **A**



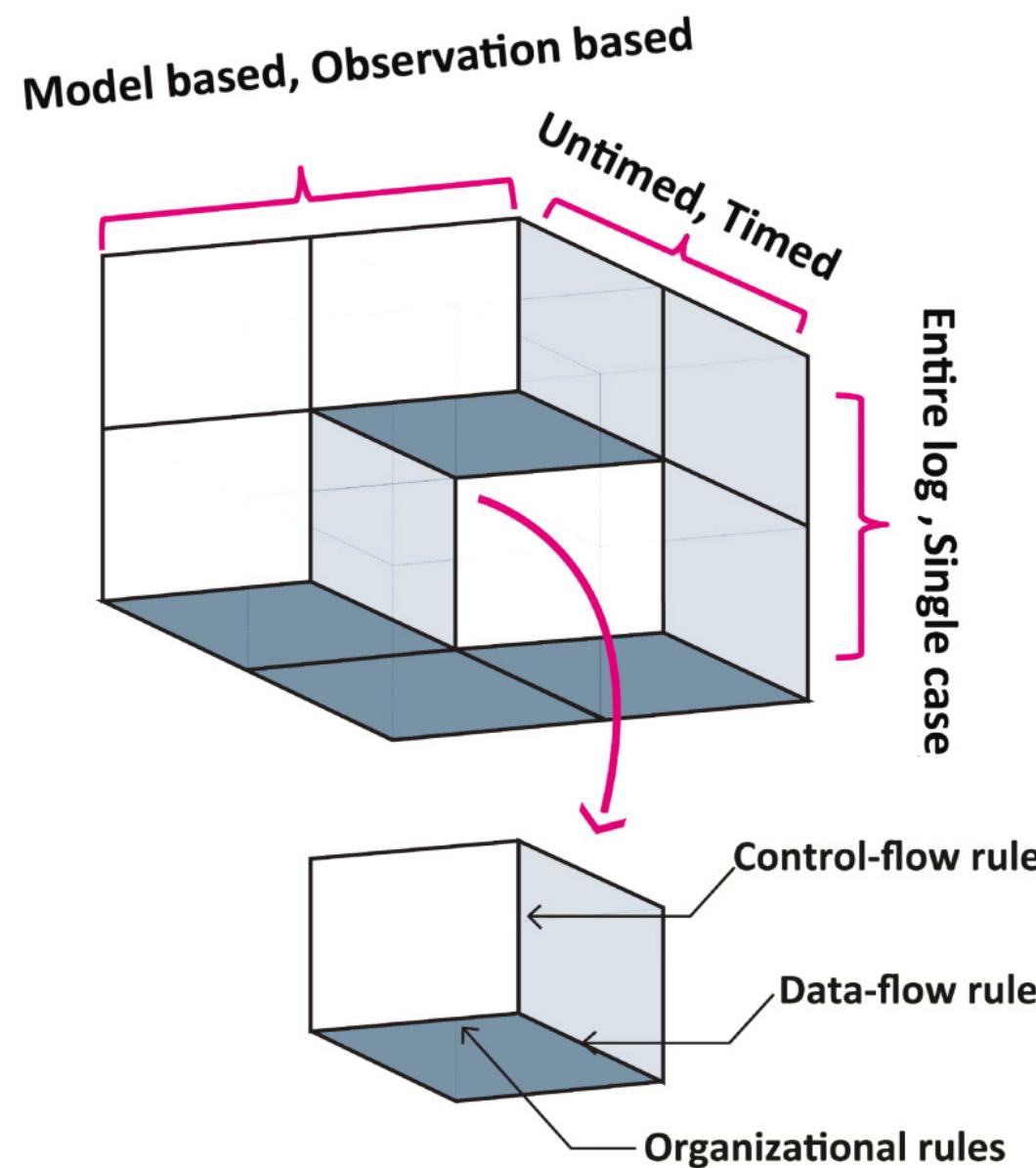
Anatomy of a Compliance Rule



[1]

E. Ramezani, D. Fahland, and W. M. P. van der Aalst,
“Where Did I Misbehave? Diagnostic Information in
Compliance Checking,” in *Business Process
Management*, 2012, pp. 262–278.

Anatomy of a Compliance Rule

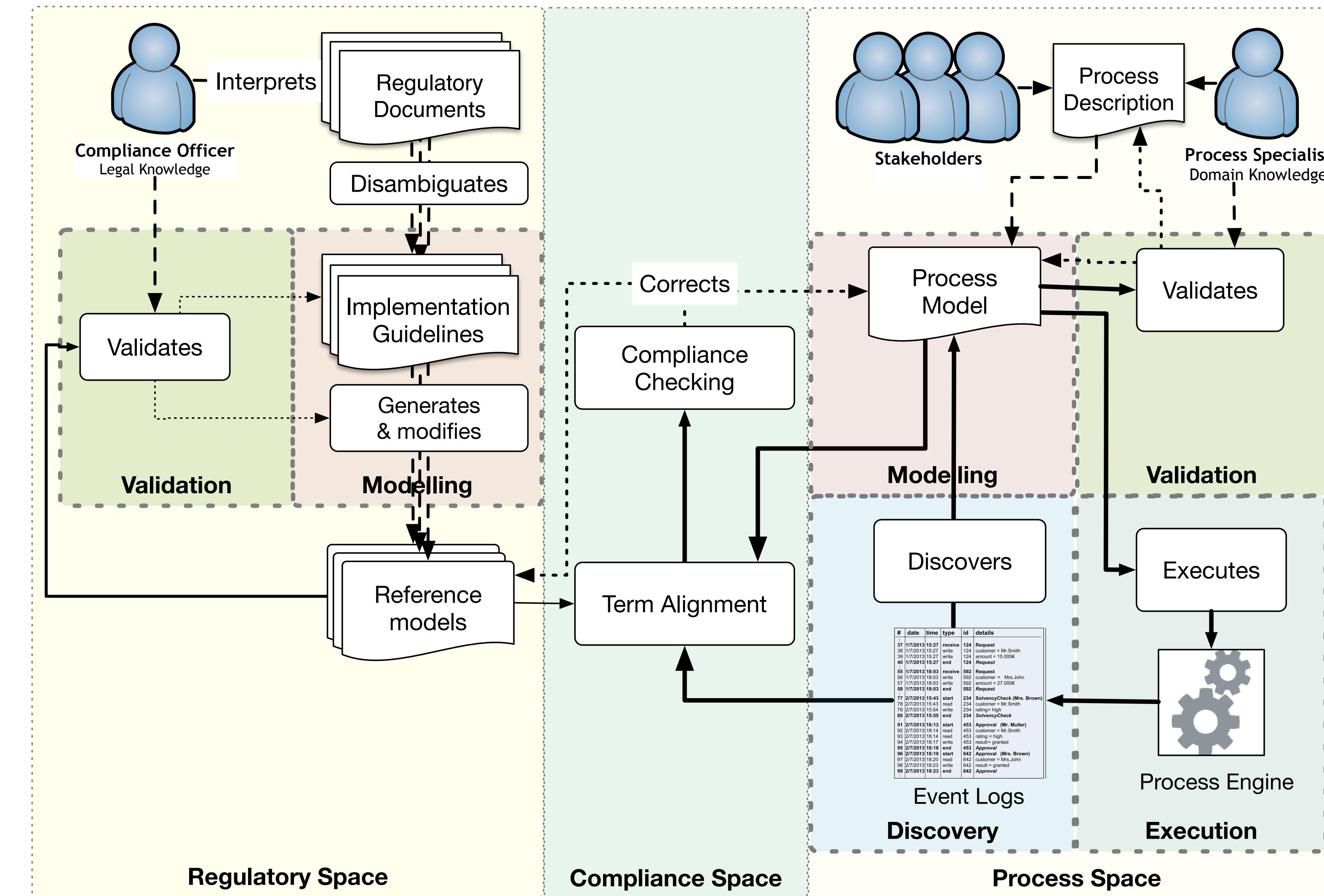


Category (Rules)	Description
Existence (2)	Limits the occurrence or absence of a given event A within a scope. [4],[15],[9], [14],[21],[36],[33]
Bounded Existence (6)	Limits the number of times a given event A must or must not occur within a scope. [15],[14]
Bounded Sequence (5)	Limits the number of times a given sequence of events must or must not occur within a scope. [15],[14]
Parallel (1)	A specific set of events should occur in parallel within a scope. [33]
Precedence (10)	Limits the occurrence of a given event A in precedence over a given event B . [15],[33],[14],[36],[9],[19],[21],[4],[33]
Chain Precedence (4)	Limits the occurrence of a sequence of events A_1, \dots, A_n over a sequence of events B_1, \dots, B_n . [15],[14],[21]
Response (10)	Limits the occurrence of a given event B in response to a given event A . [33],[14],[21],[15],[37],[9],[19]
Chain Response (4)	Limits the occurrence of a sequence of events B_1, \dots, B_n in response to a sequence of events A_1, \dots, A_n . [15]
Between (7)	Limits the occurrence of a given event B between a sequence of events A and C . [14]
Exclusive (1)	Presence of a given event A mandates the absence of an event B . [15]
Mutual Exclusive (1)	Either a given event A or event B must exist but not none of them or both. [15],[34]
Inclusive (1)	Presence of a given event A mandates that event B is also present. [15]
Prerequisite (1)	Absence of a given event A mandates that event B is also absent. [15]
Substitute (1)	A given event B substitutes the absence of event A . [15]
Corequisite (1)	Either given events A and B should exist together or to be absent together. [15]

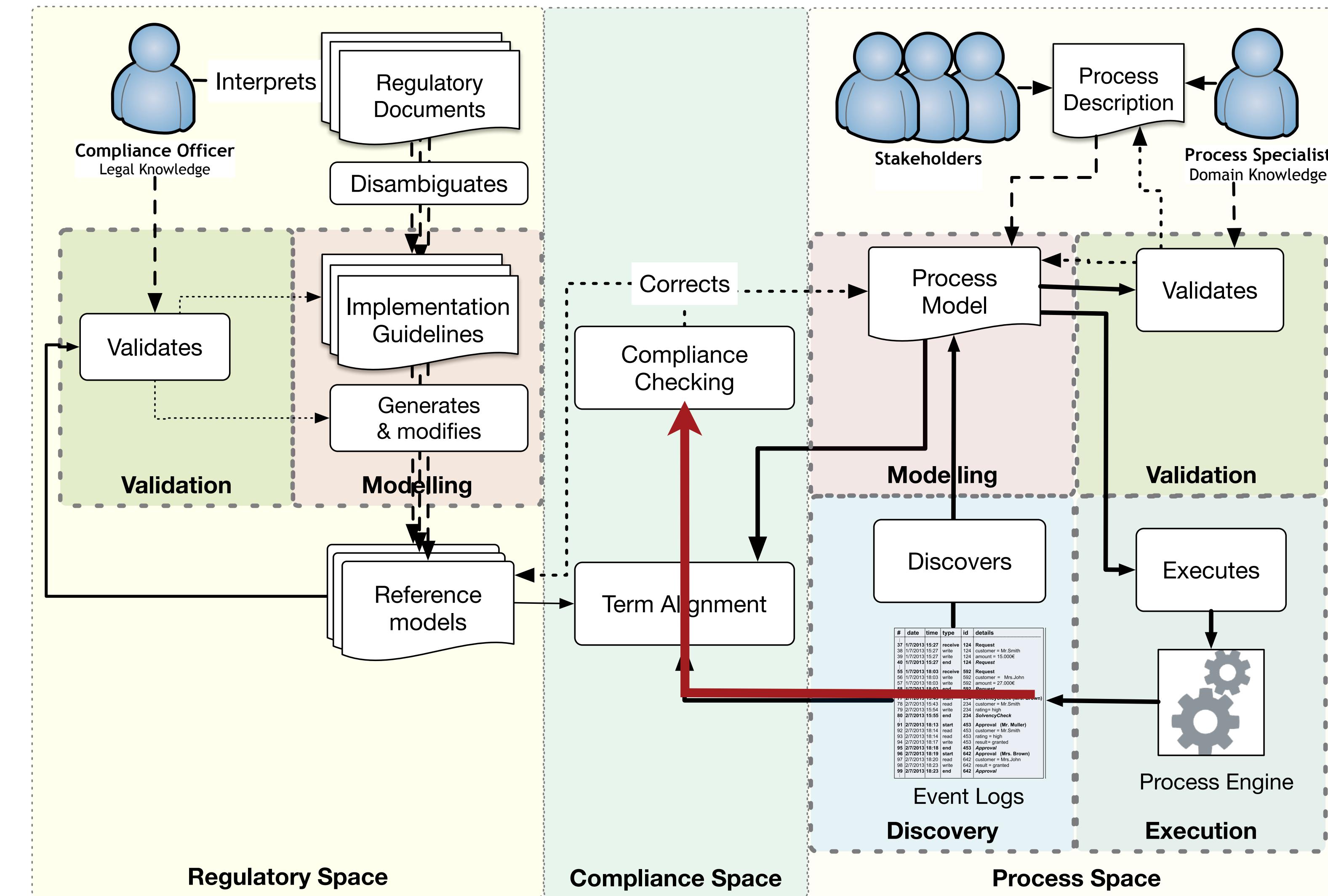
[1]

E. Ramezani, D. Fahland, and W. M. P. van der Aalst, “Where Did I Misbehave? Diagnostic Information in Compliance Checking,” in *Business Process Management*, 2012, pp. 262–278.

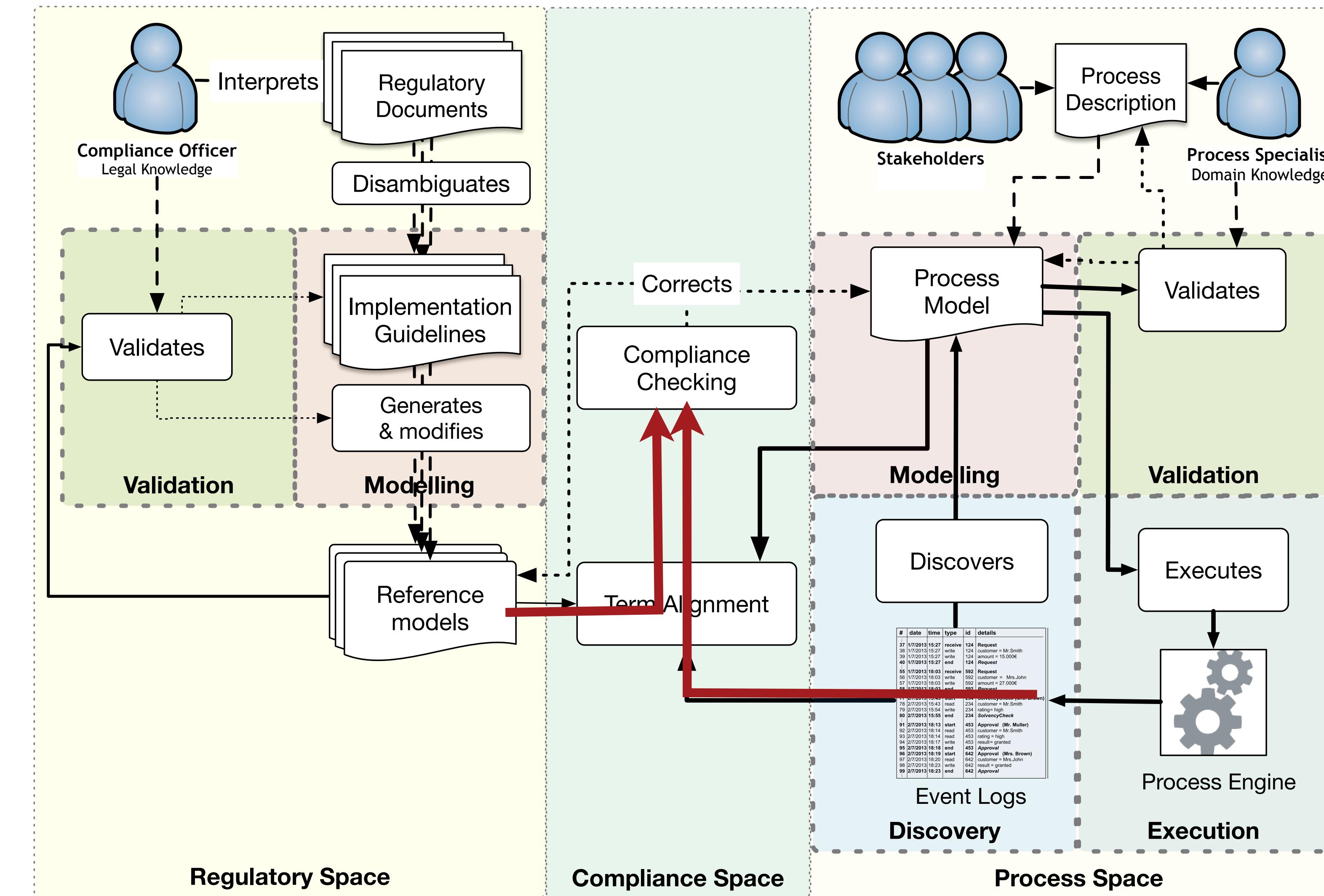
2. Compliance as Conformance Checking



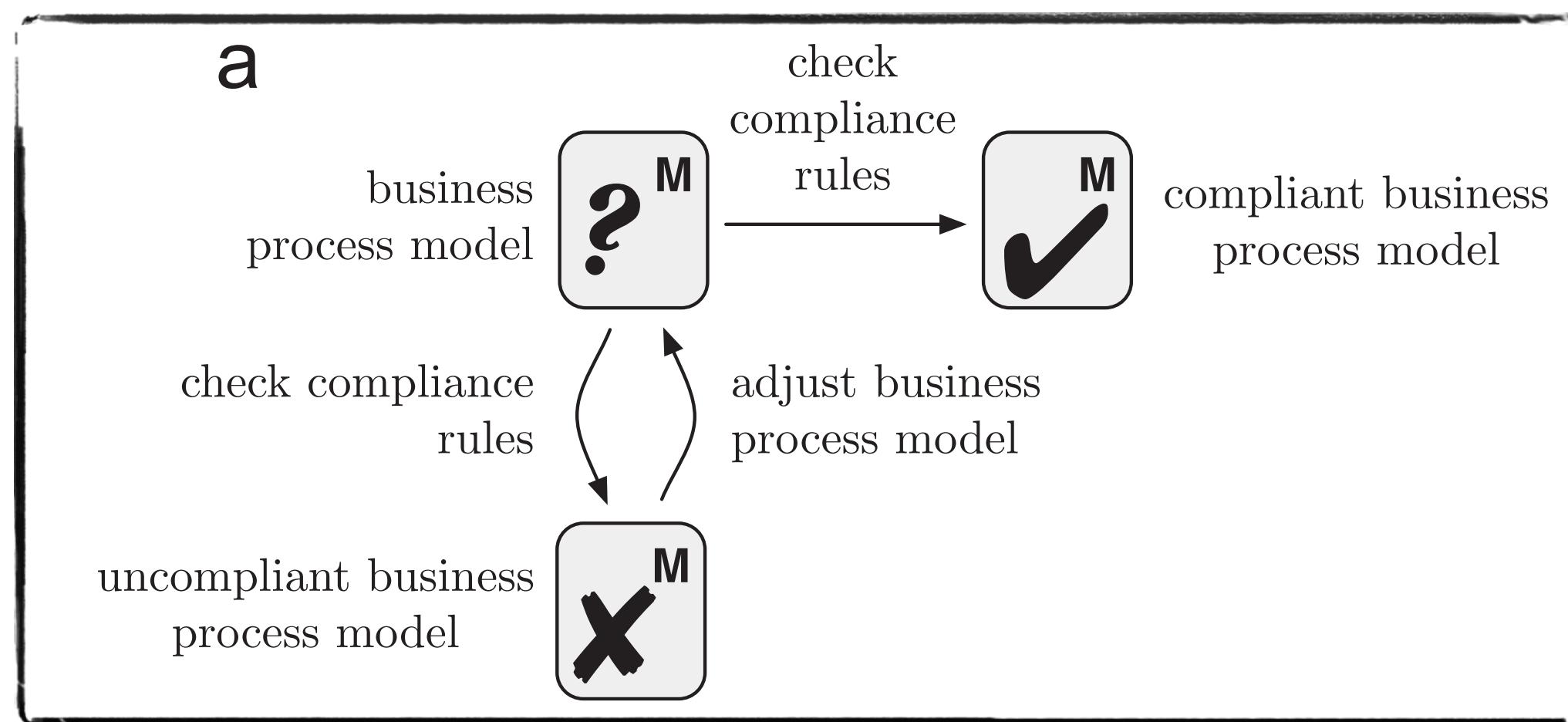
2. Compliance as Conformance Checking



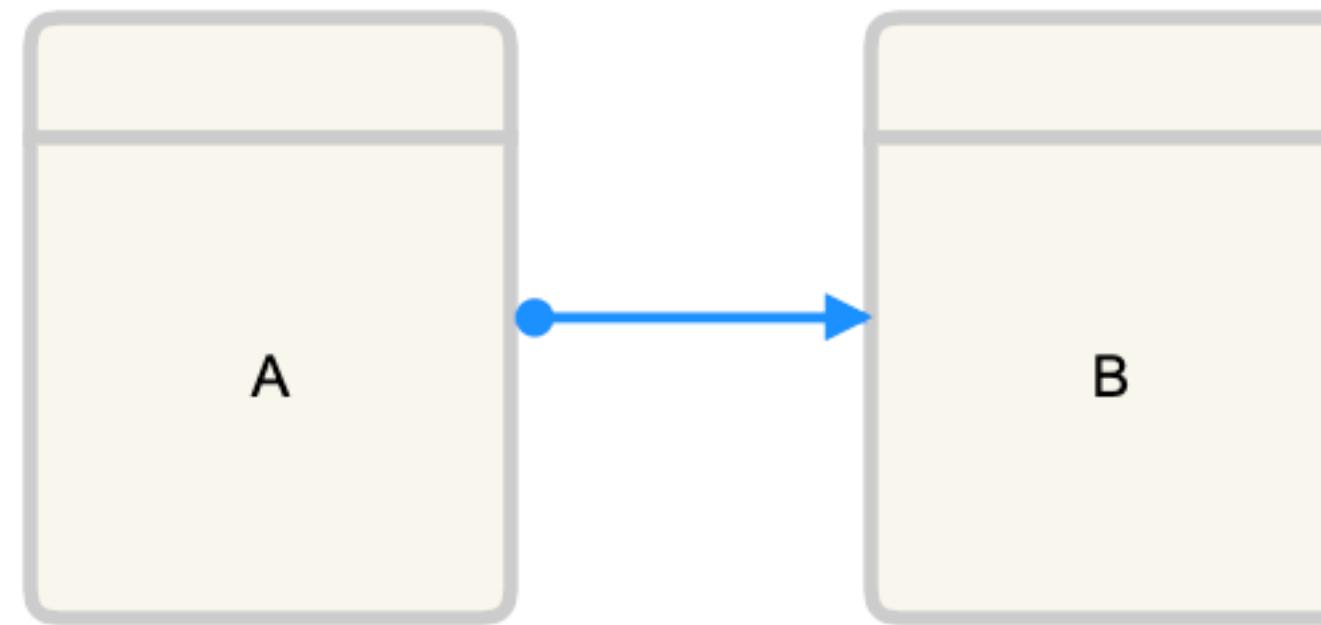
2. Compliance as Conformance Checking



2.Compliance via Conformance Checking



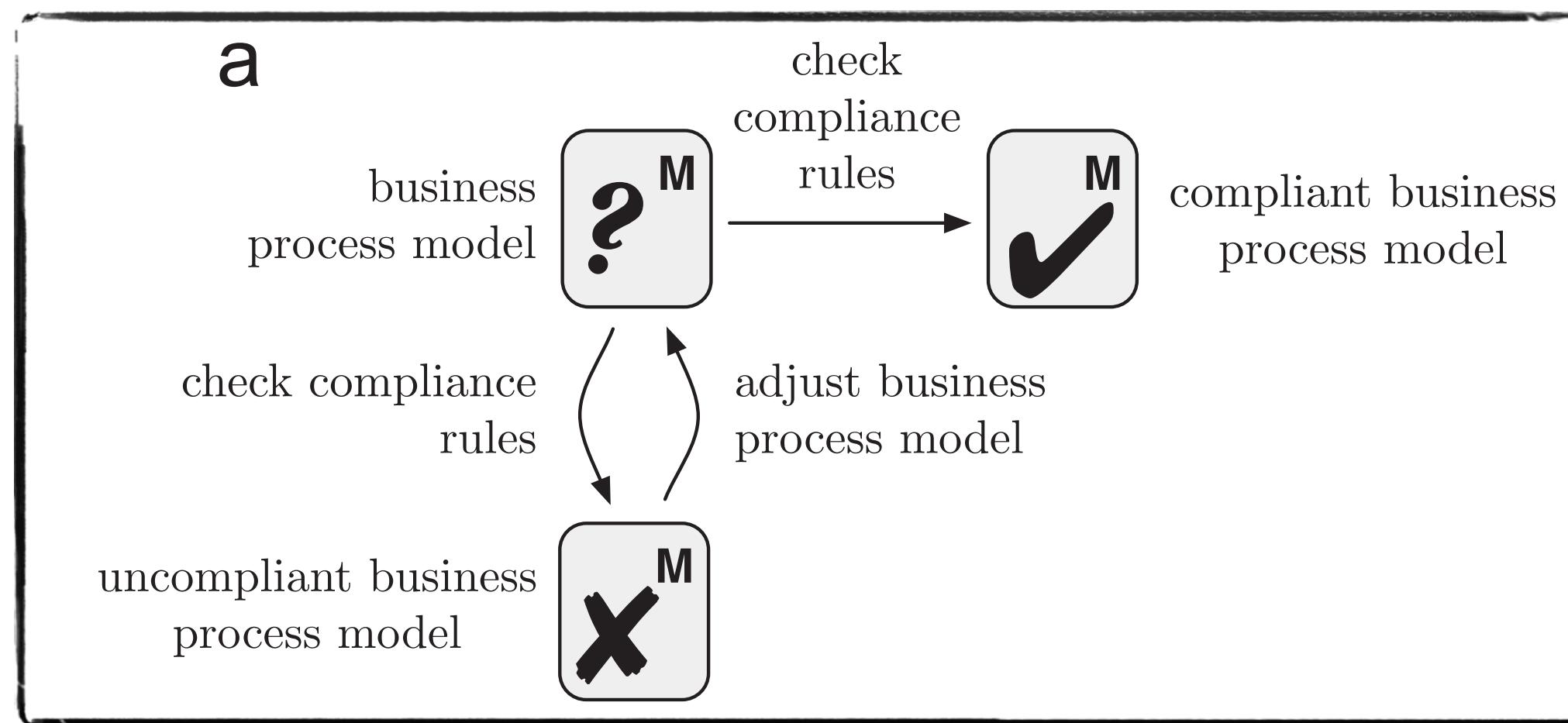
Compliance Rule
(Response of a Task)



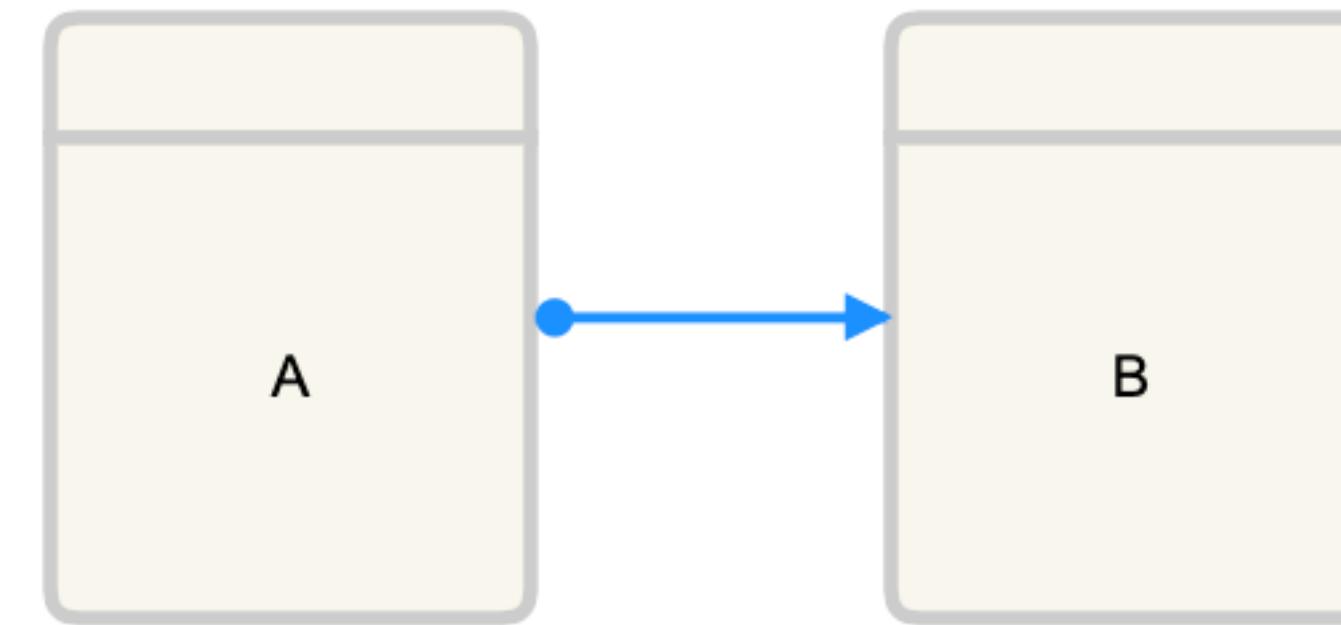
Event log
(Might involve events outside the scope of the rule)

- < E, D, F, G, B >
- < G, C, F, D, G >
- < C, A, B, D, B >
- < G, C, B, A, D >
- < A, F, A, D, B >
- < A, D, B, G, A >

2.Compliance via Conformance Checking



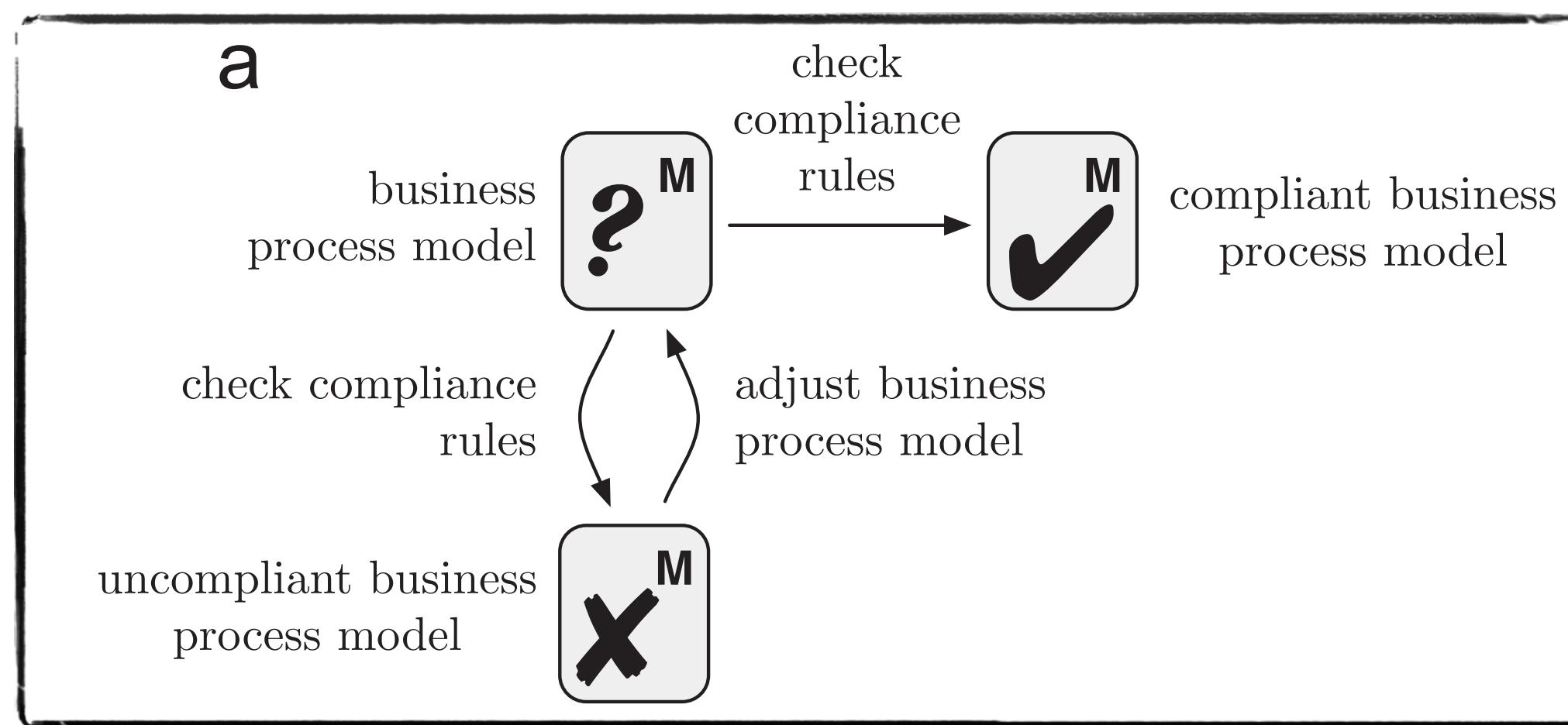
Compliance Rule (Response of a Task)



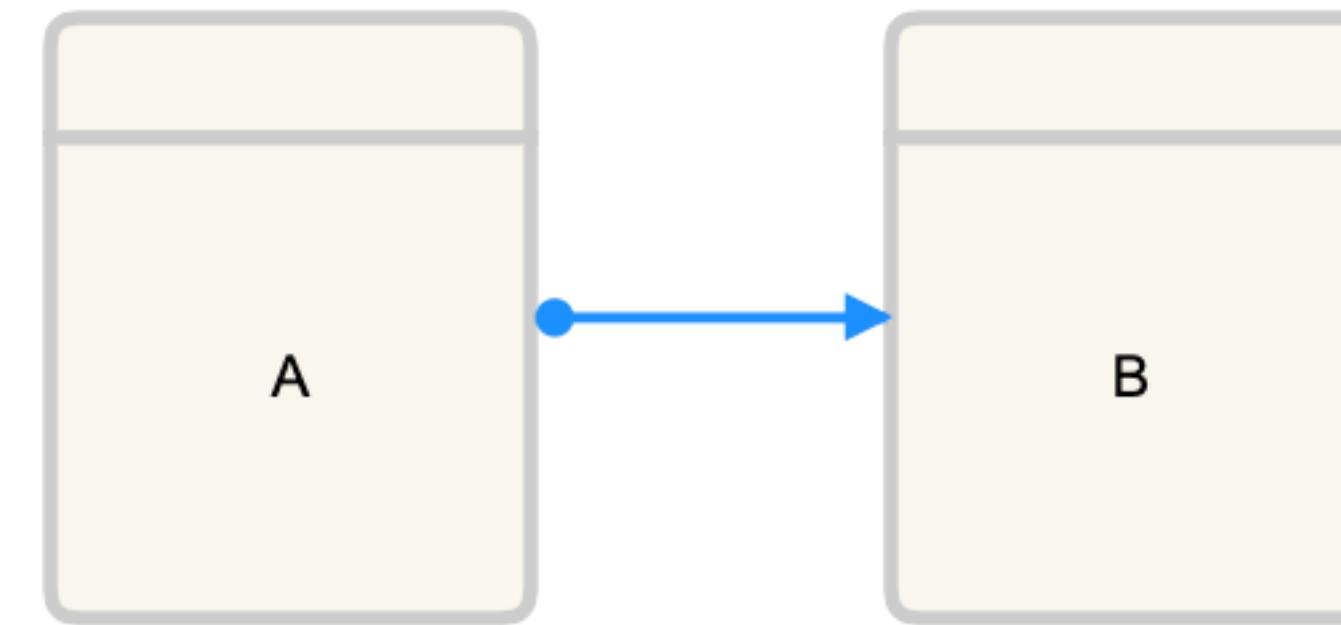
Event log
(Might involve events outside the scope of the rule)

- < E, D, F, G, B > ✓
- < G, C, F, D, G >
- < C, A, B, D, B >
- < G, C, B, A, D >
- < A, F, A, D, B >
- < A, D, B, G, A >

2.Compliance via Conformance Checking



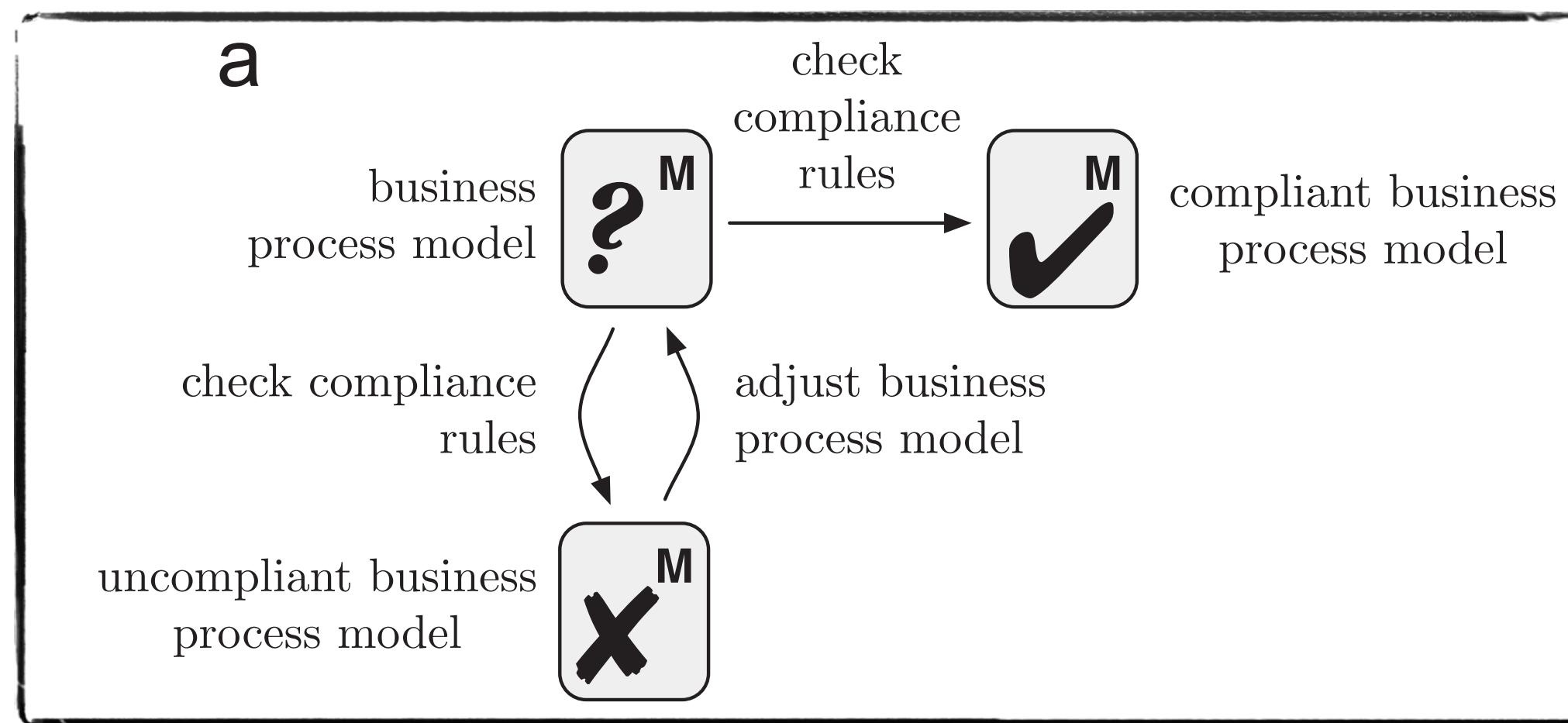
Compliance Rule (Response of a Task)



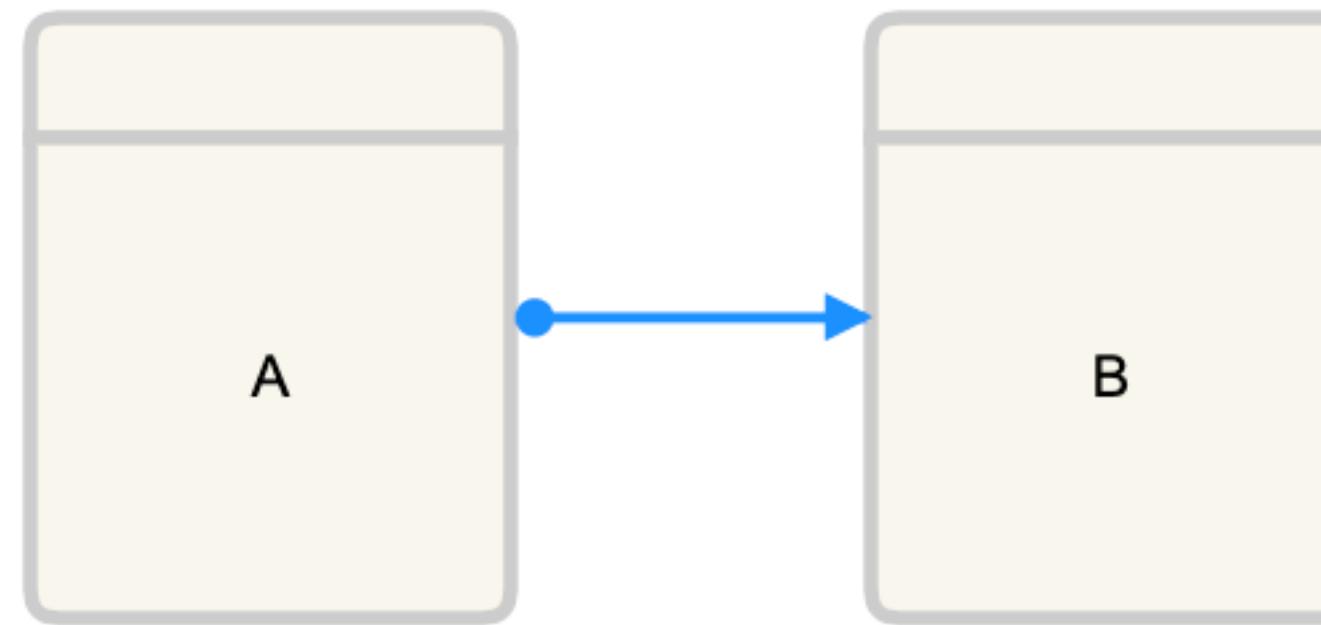
Event log
(Might involve events outside the scope of the rule)

- < E, D, F, G, B > ✓
- < G, C, F, D, G > ✓
- < C, A, B, D, B >
- < G, C, B, A, D >
- < A, F, A, D, B >
- < A, D, B, G, A >

2.Compliance via Conformance Checking



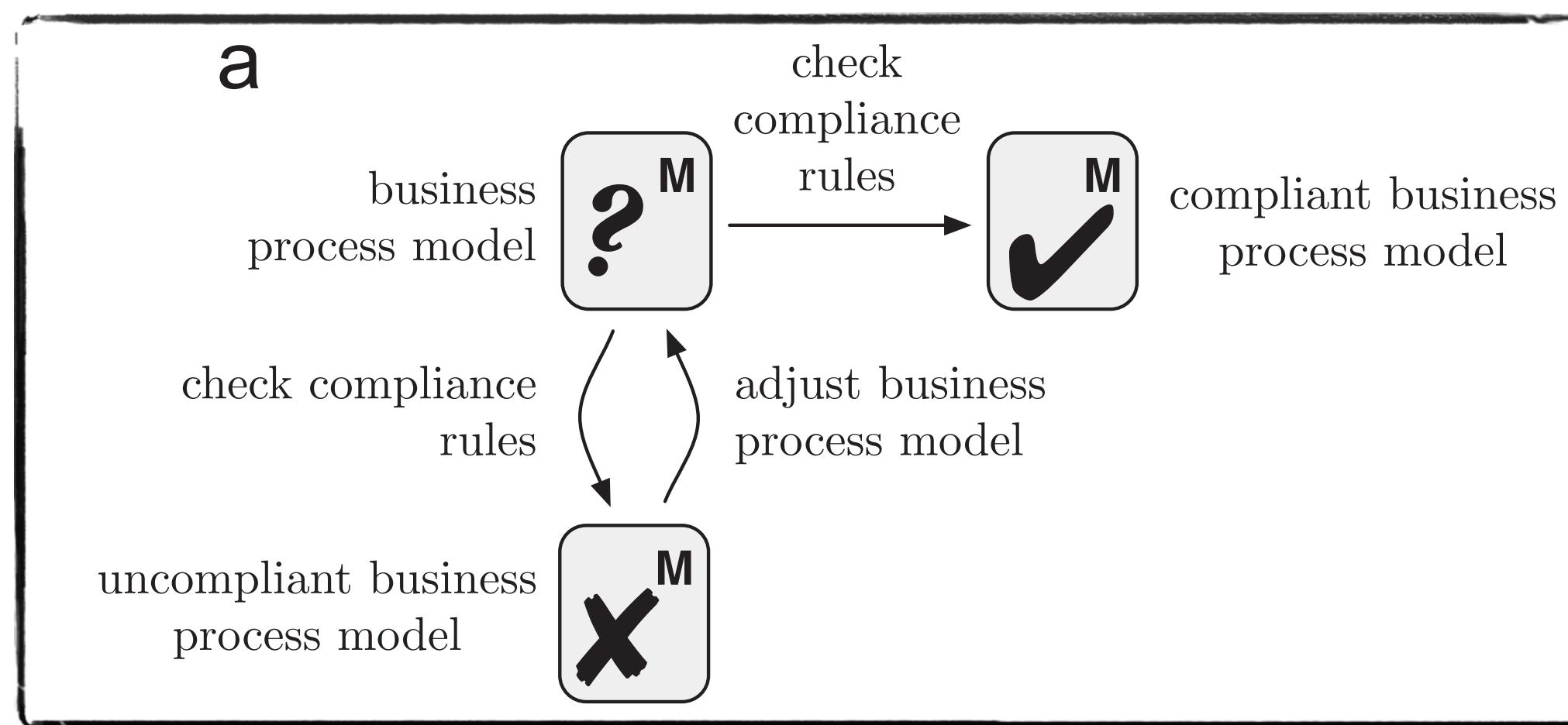
Compliance Rule (Response of a Task)



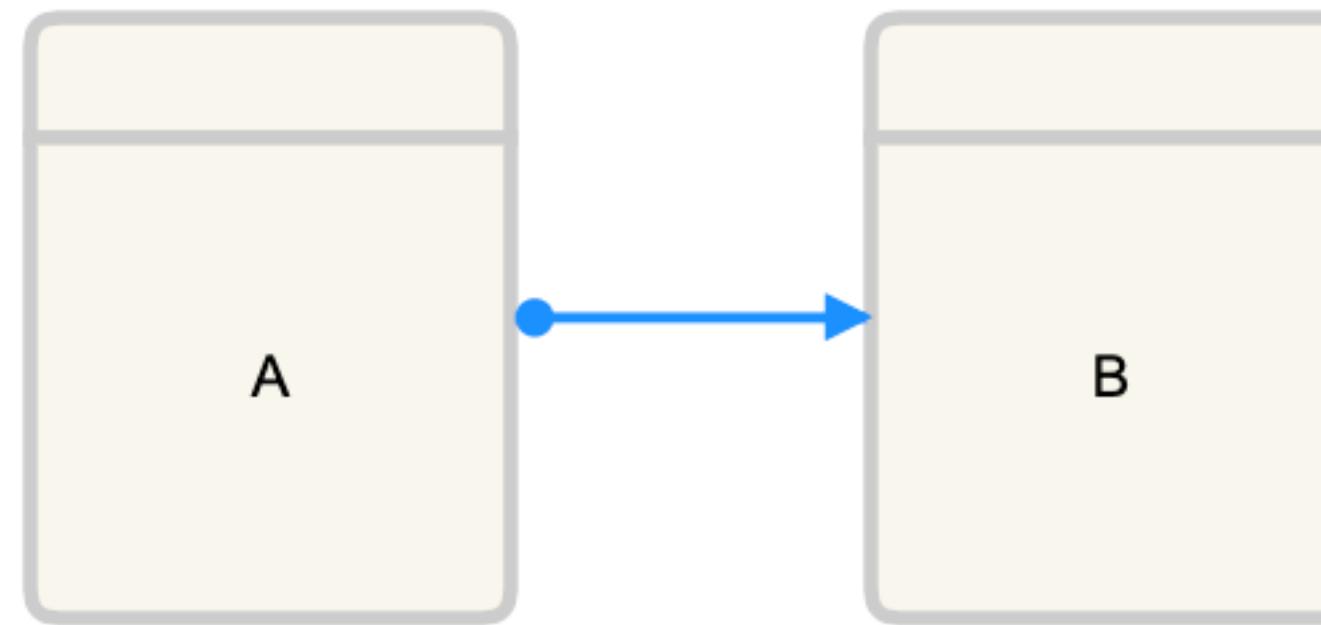
Event log
(Might involve events outside the scope of the rule)

- < E, D, F, G, B > ✓
- < G, C, F, D, G > ✓
- < C, A, B, D, B > ✓
- < G, C, B, A, D >
- < A, F, A, D, B >
- < A, D, B, G, A >

2.Compliance via Conformance Checking



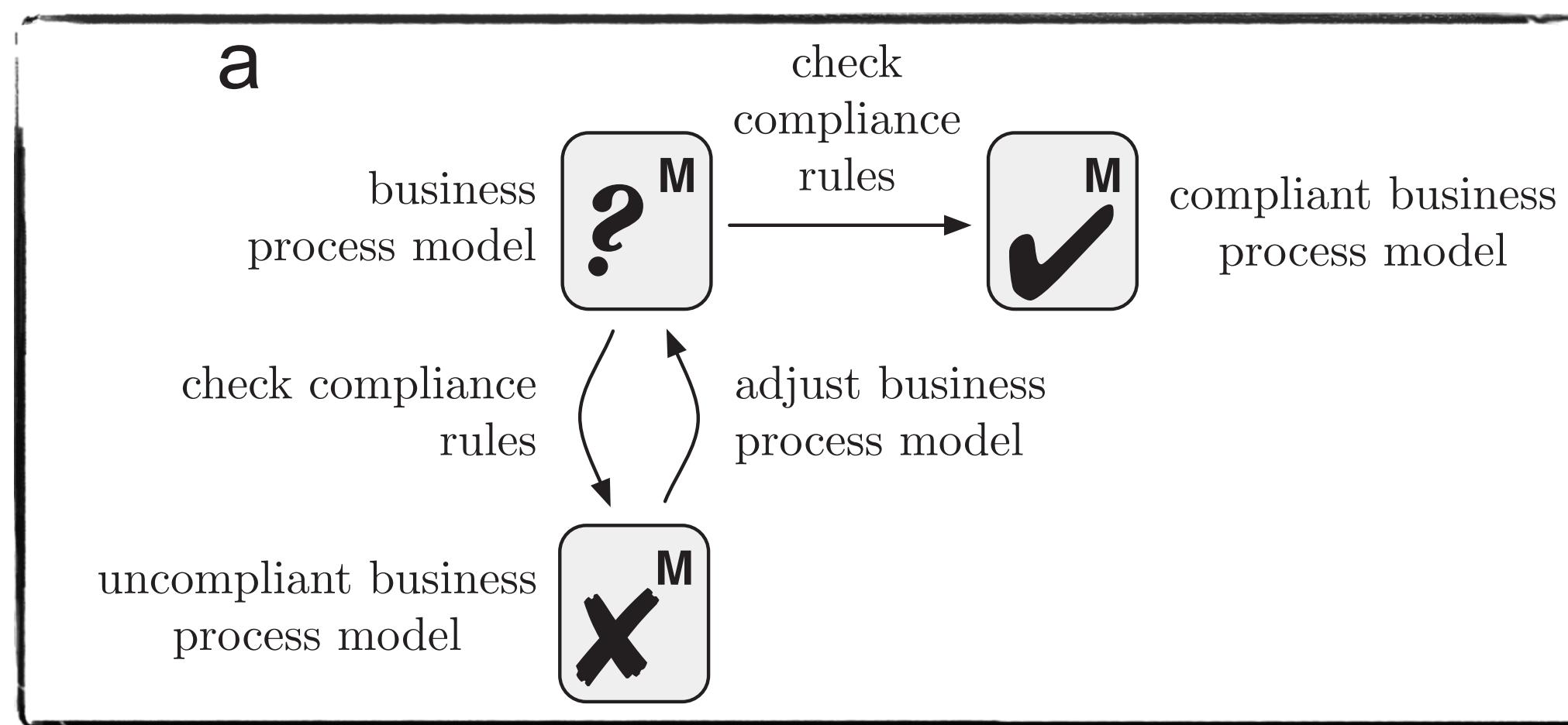
Compliance Rule (Response of a Task)



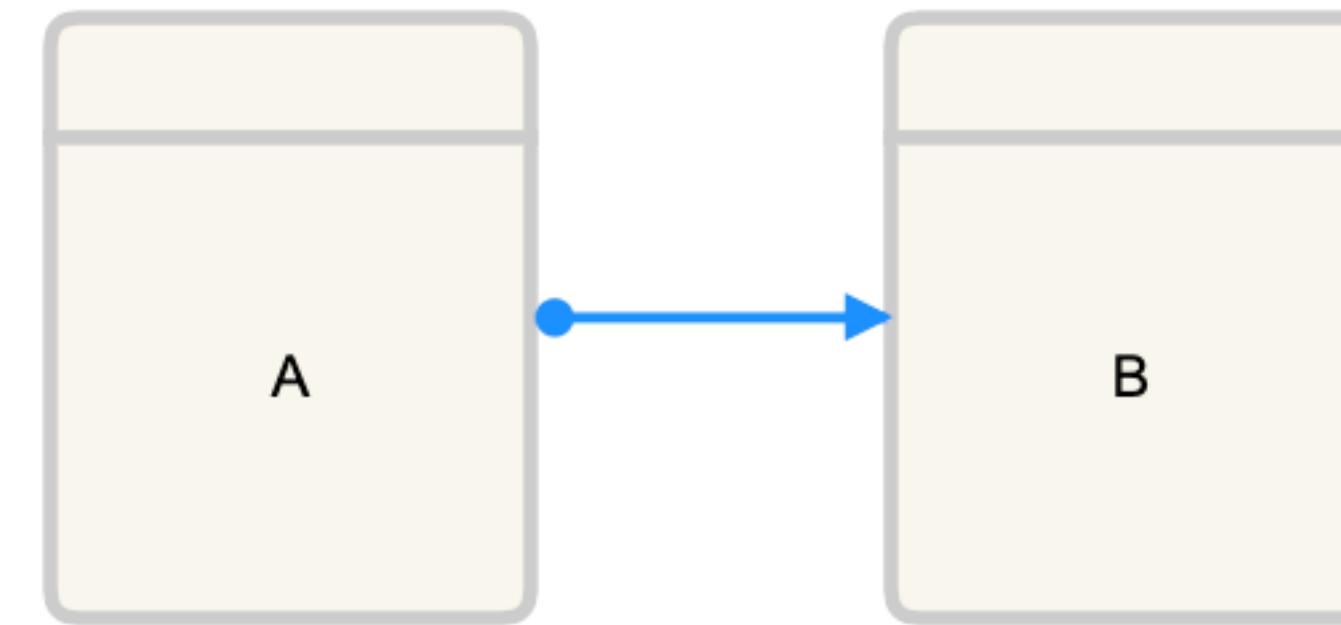
Event log
(Might involve events outside the scope of the rule)

< E, D, F, G, B >	✓
< G, C, F, D, G >	✓
< C, A, B, D, B >	✓
< G, C, B, A, D >	⌚
< A, F, A, D, B >	
< A, D, B, G, A >	

2.Compliance via Conformance Checking



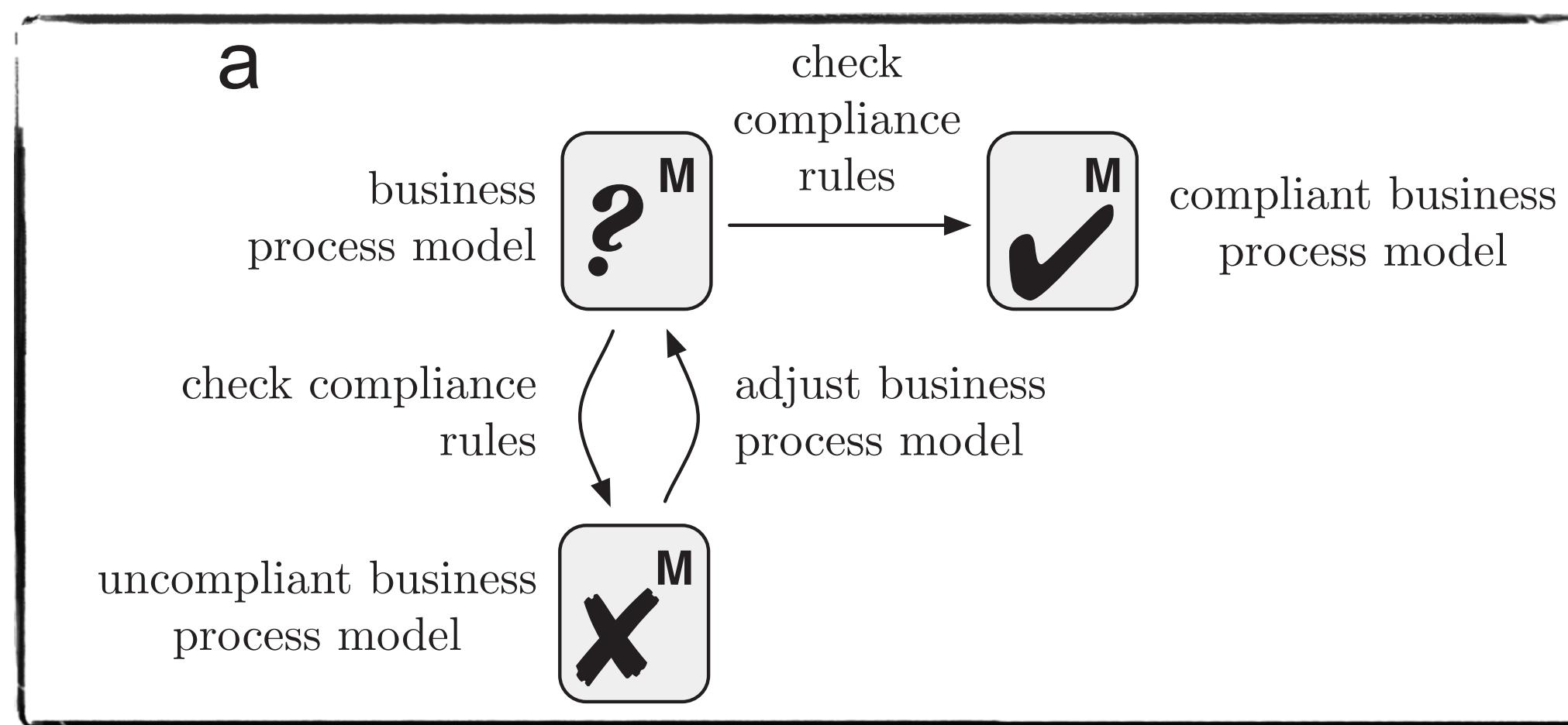
Compliance Rule (Response of a Task)



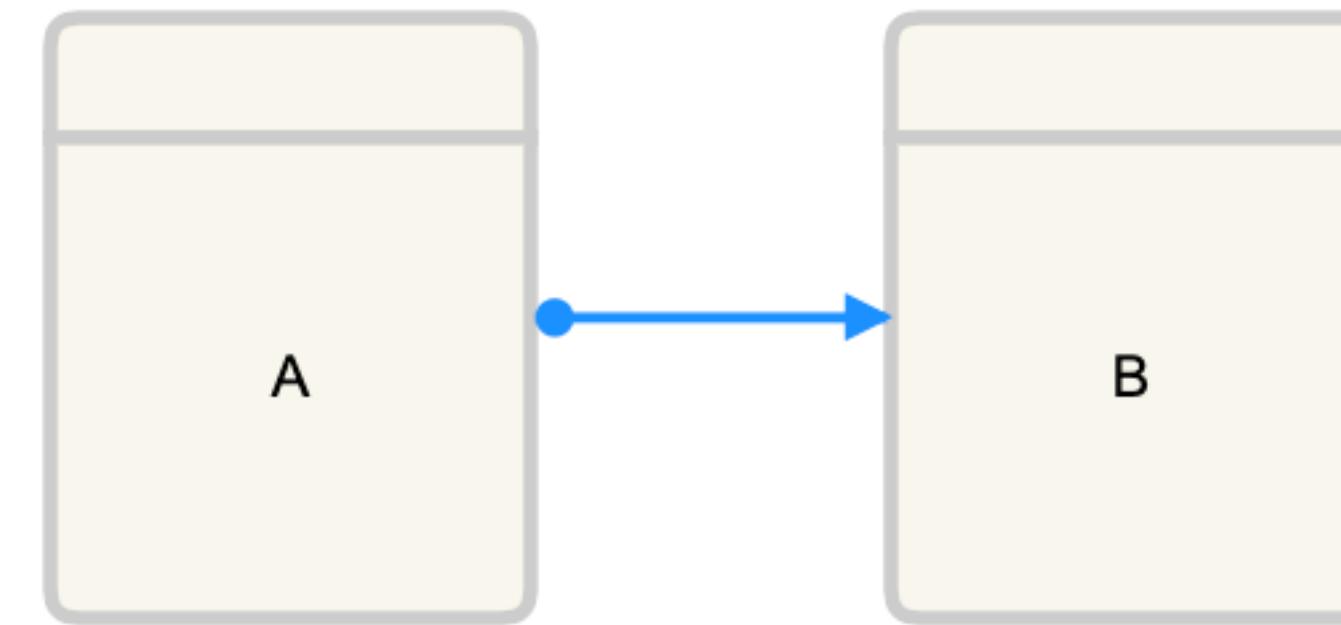
Event log
(Might involve events outside the scope of the rule)

< E, D, F, G, B >	✓
< G, C, F, D, G >	✓
< C, A, B, D, B >	✓
< G, C, B, A, D >	⌚
< A, F, A, D, B >	✓
< A, D, B, G, A >	

2.Compliance via Conformance Checking



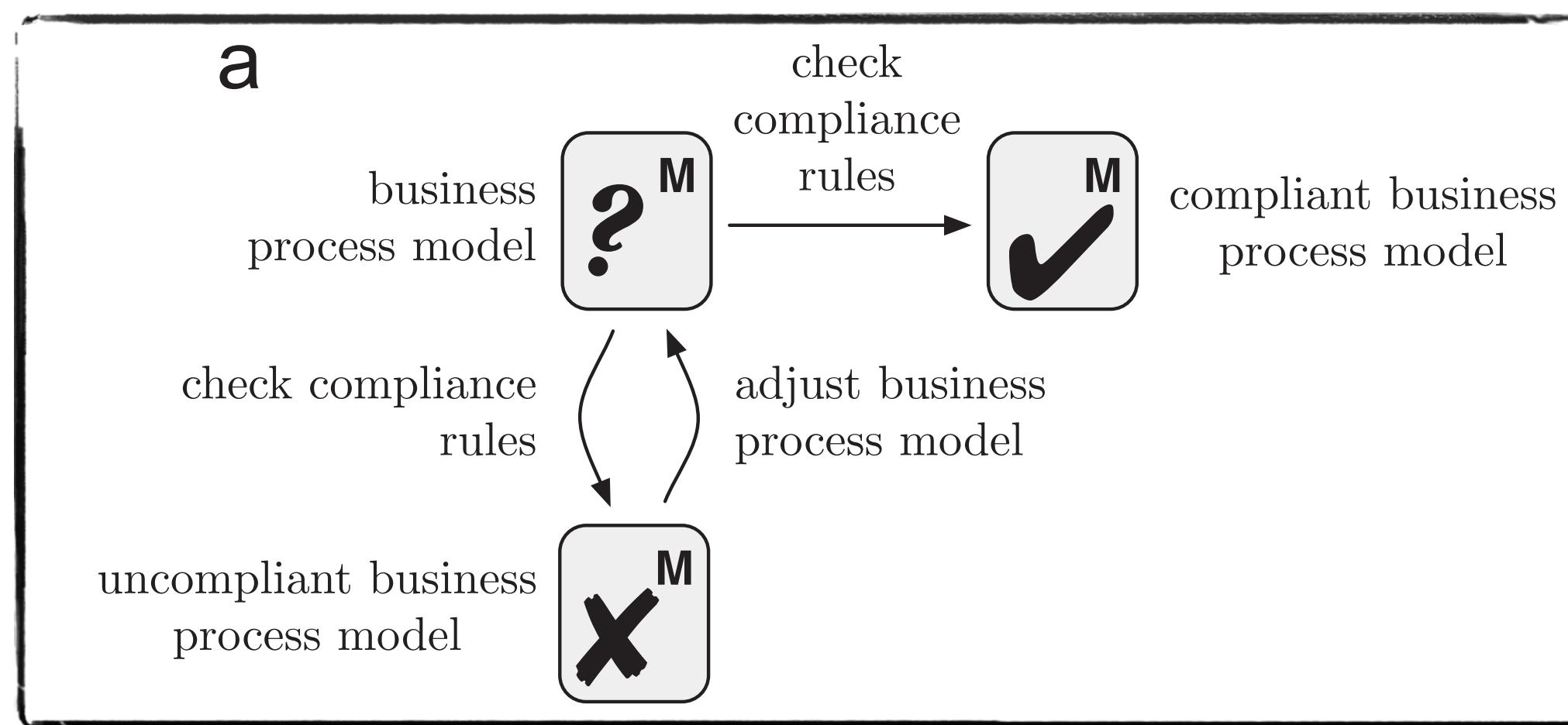
Compliance Rule (Response of a Task)



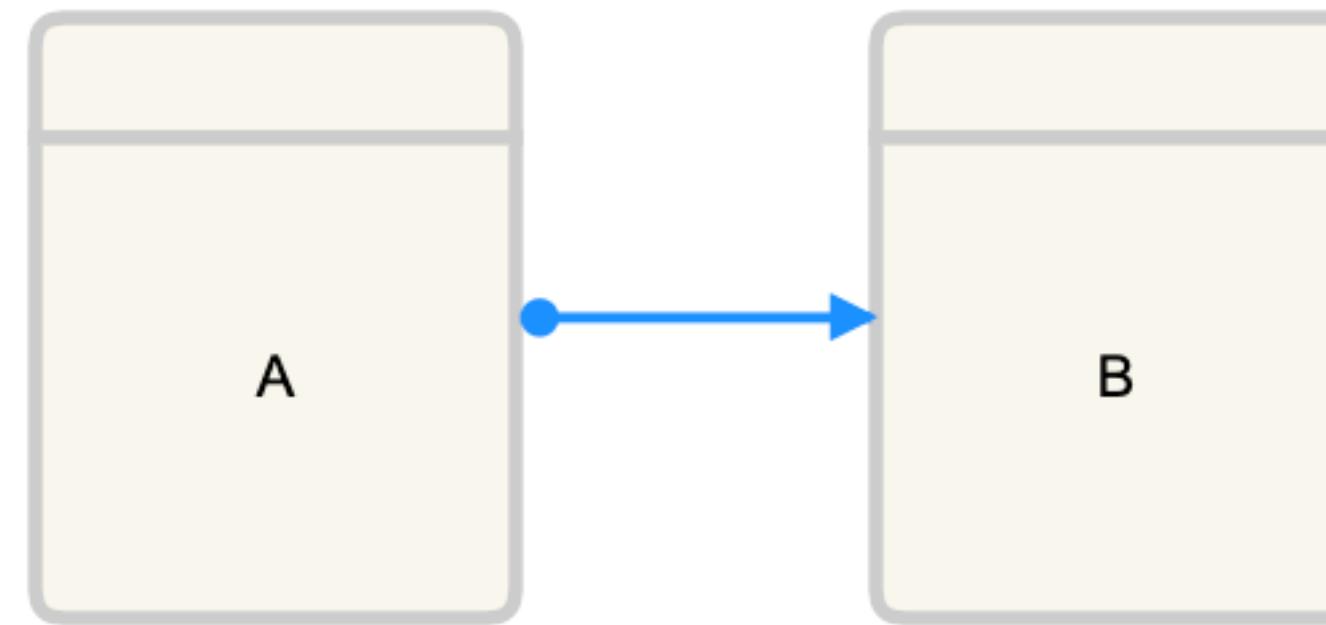
Event log
(Might involve events outside the scope of the rule)

< E, D, F, G, B >	✓
< G, C, F, D, G >	✓
< C, A, B, D, B >	✓
< G, C, B, A, D >	⌚
< A, F, A, D, B >	✓
< A, D, B, G, A >	⌚

2.Compliance via Conformance Checking



Compliance Rule (Response of a Task)



Full Compliant

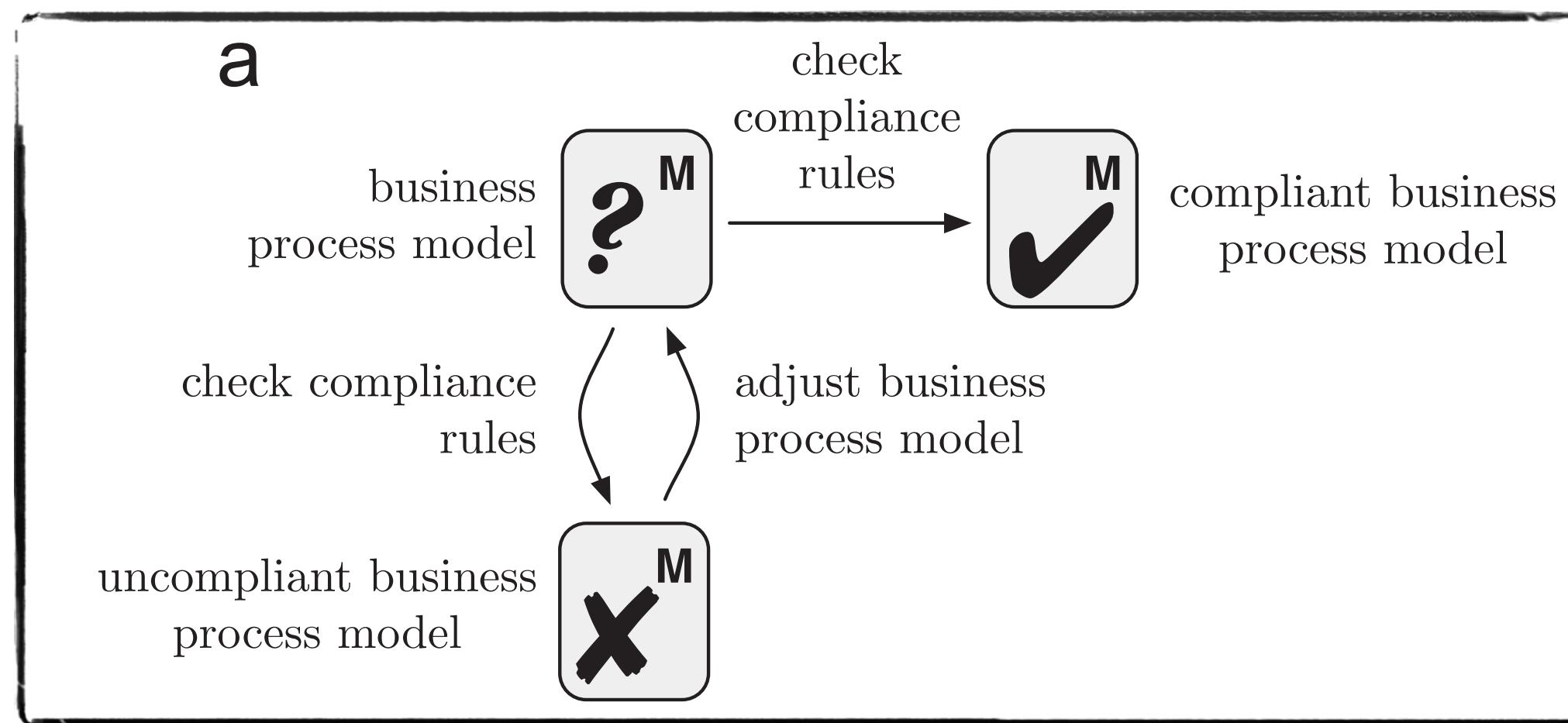
All traces satisfy the rule

Event log

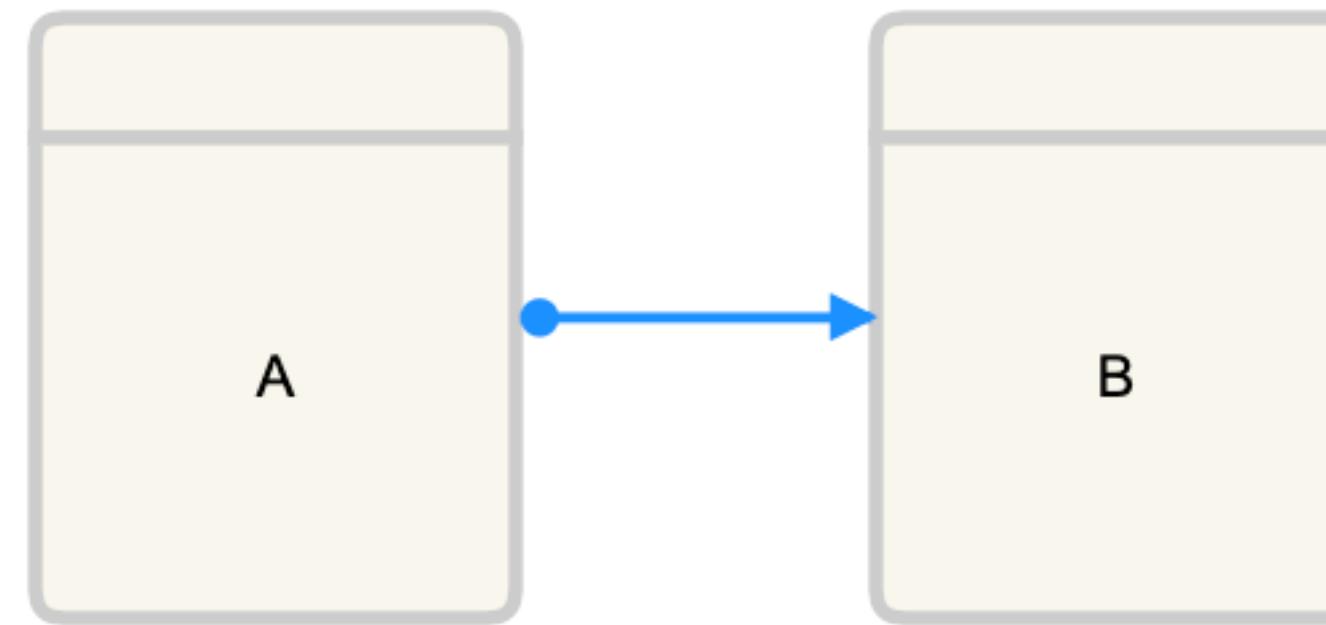
(Might involve events outside the scope of the rule)

$\langle E, D, F, G, \textcolor{blue}{B} \rangle$	✓
$\langle G, C, F, D, G \rangle$	✓
$\langle C, \textcolor{green}{A}, \textcolor{blue}{B}, D, \textcolor{blue}{B} \rangle$	✓
$\langle G, C, \textcolor{blue}{B}, \textcolor{green}{A}, D \rangle$	⌚
$\langle \textcolor{green}{A}, F, \textcolor{green}{A}, D, \textcolor{blue}{B} \rangle$	✓
$\langle \textcolor{green}{A}, D, \textcolor{blue}{B}, G, \textcolor{green}{A} \rangle$	⌚

2.Compliance via Conformance Checking



Compliance Rule (Response of a Task)



Full Compliant

All traces satisfy the rule

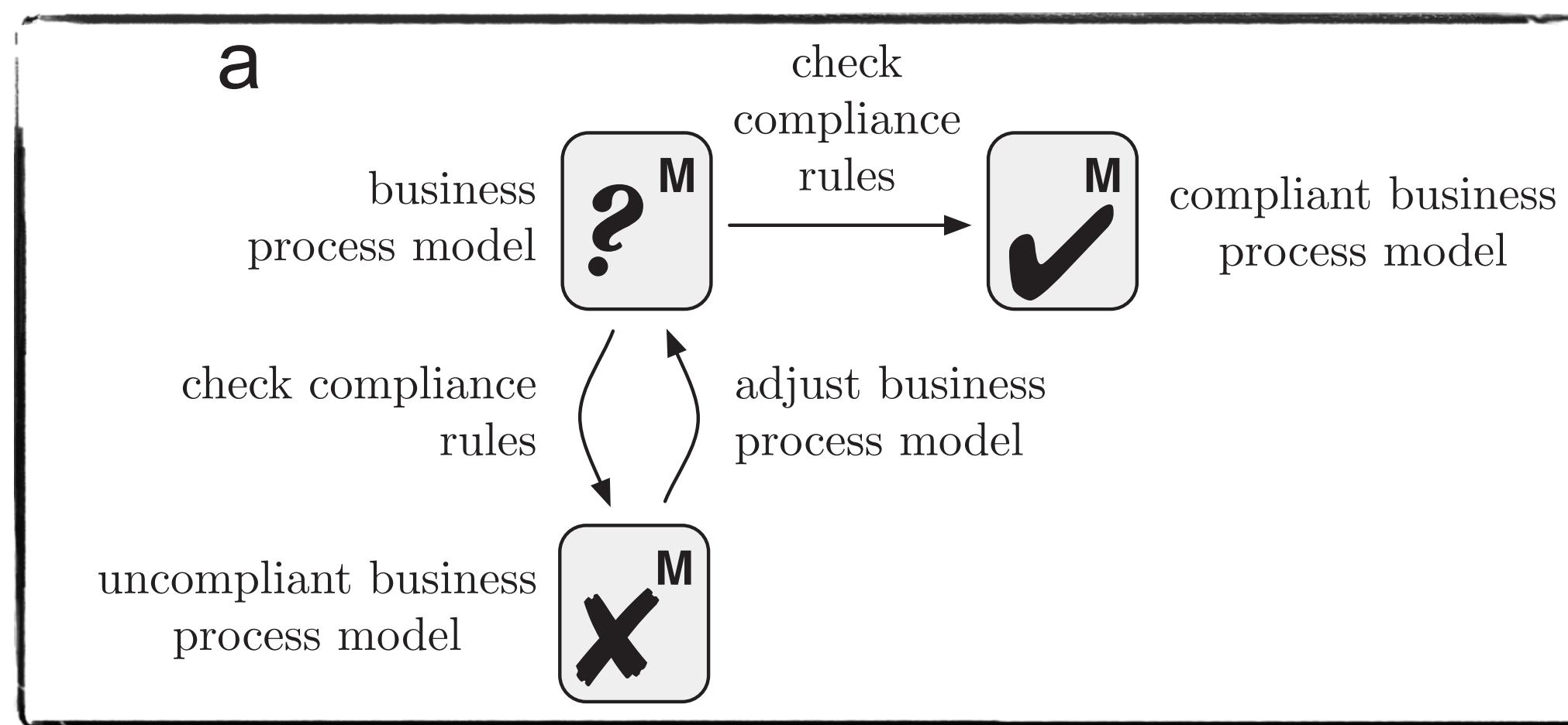


Event log

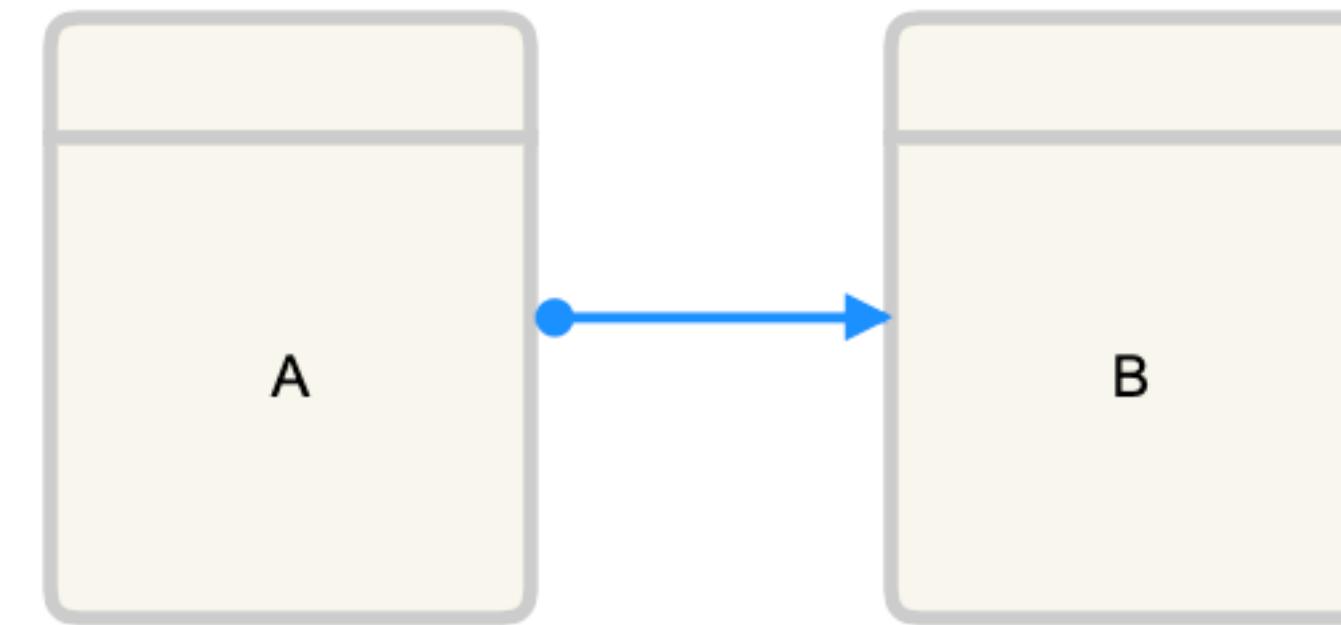
(Might involve events outside the scope of the rule)

$\langle E, D, F, G, \textcolor{blue}{B} \rangle$	✓
$\langle G, C, F, D, G \rangle$	✓
$\langle C, \textcolor{green}{A}, \textcolor{blue}{B}, D, \textcolor{blue}{B} \rangle$	✓
$\langle G, C, \textcolor{blue}{B}, \textcolor{green}{A}, D \rangle$	⊖
$\langle \textcolor{green}{A}, F, \textcolor{green}{A}, D, \textcolor{blue}{B} \rangle$	✓
$\langle \textcolor{green}{A}, D, \textcolor{blue}{B}, G, \textcolor{green}{A} \rangle$	⊖

2.Compliance via Conformance Checking



Compliance Rule (Response of a Task)



Full Compliant

All traces satisfy the rule



Partially (Weakly) Compliant

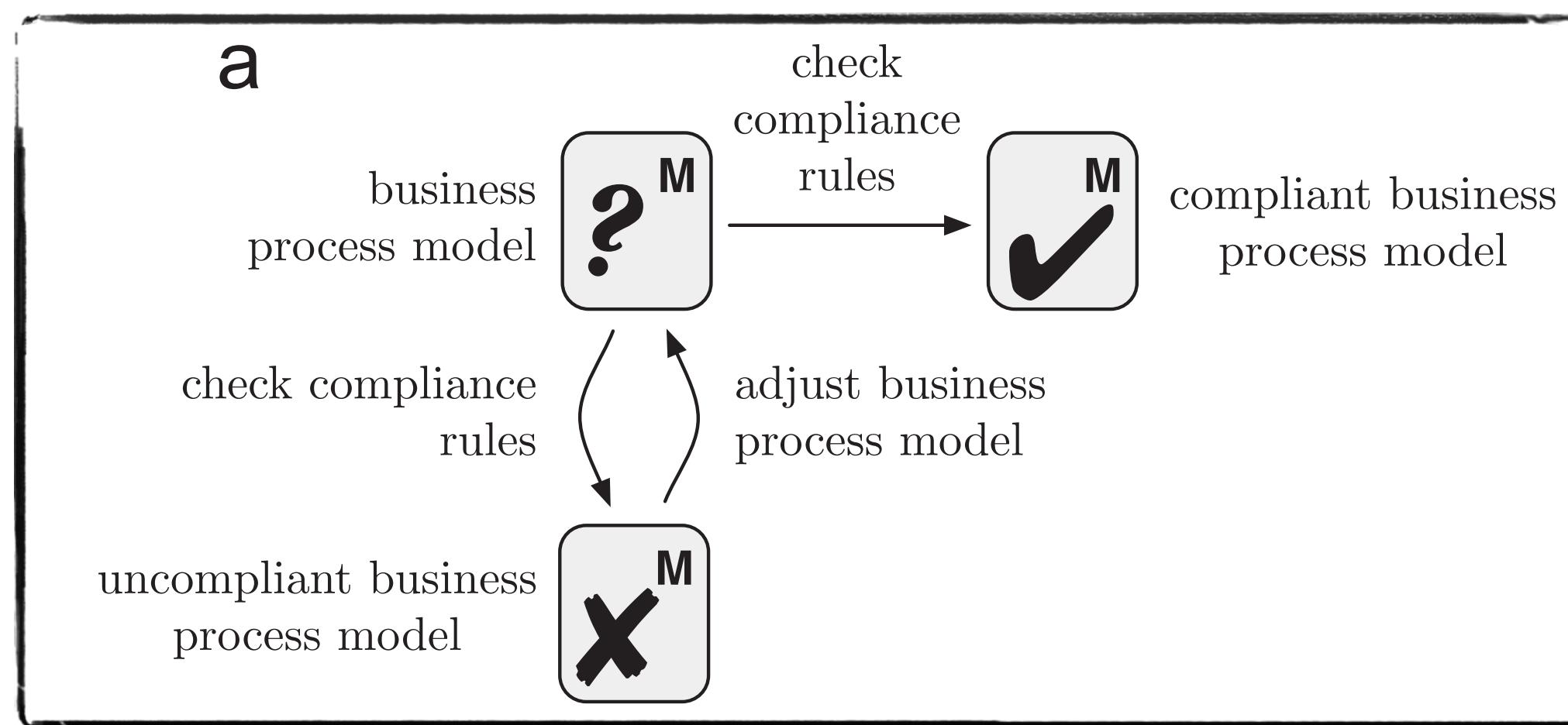
At least a trace satisfy the rule

Event log

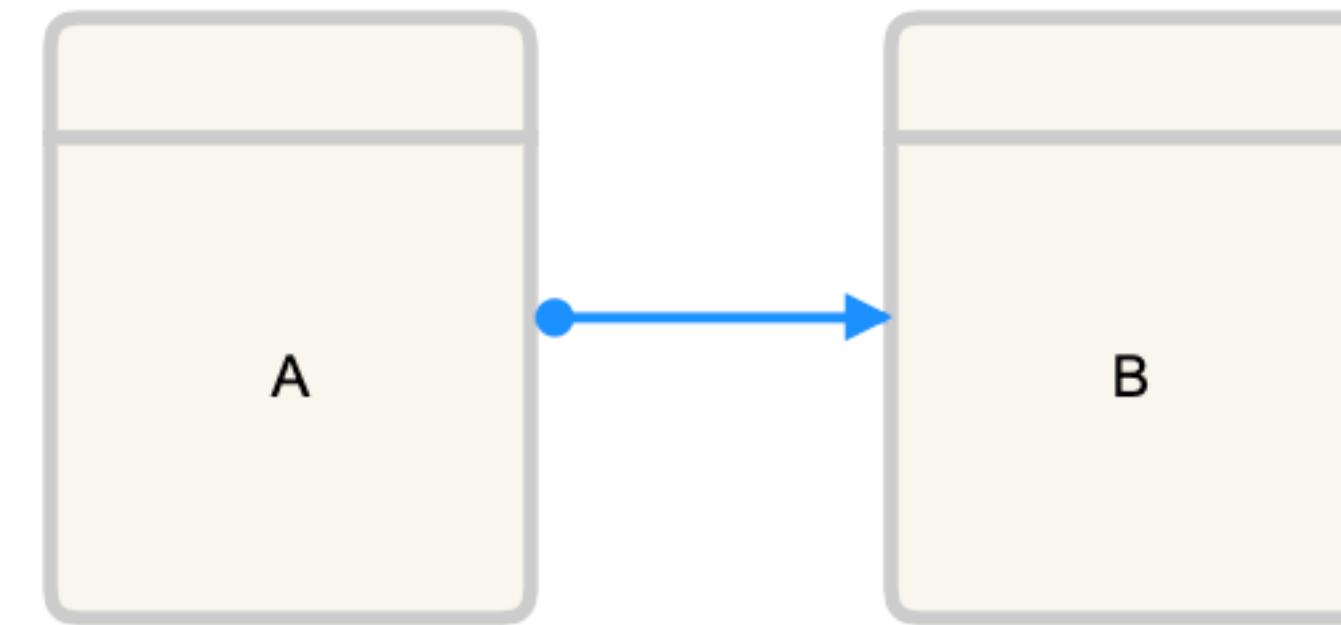
(Might involve events outside the scope of the rule)

$\langle E, D, F, G, \text{B} \rangle$	✓
$\langle G, C, F, D, G \rangle$	✓
$\langle C, \text{A}, \text{B}, D, \text{B} \rangle$	✓
$\langle G, C, \text{B}, \text{A}, D \rangle$	⊖
$\langle \text{A}, F, \text{A}, D, \text{B} \rangle$	✓
$\langle \text{A}, D, \text{B}, G, \text{A} \rangle$	⊖

2.Compliance via Conformance Checking



Compliance Rule (Response of a Task)



Full Compliant

All traces satisfy the rule



Partially (Weakly) Compliant

At least a trace satisfy the rule

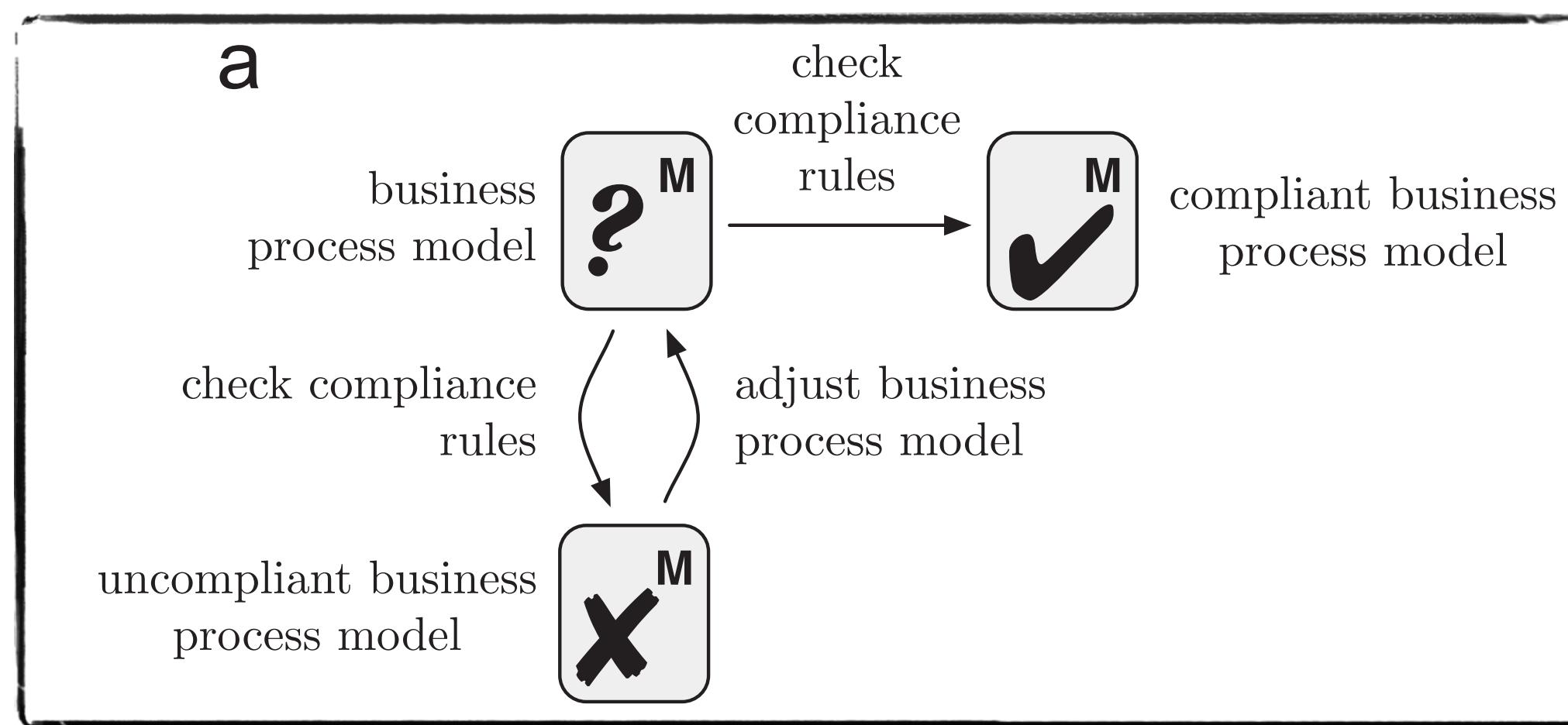


Event log

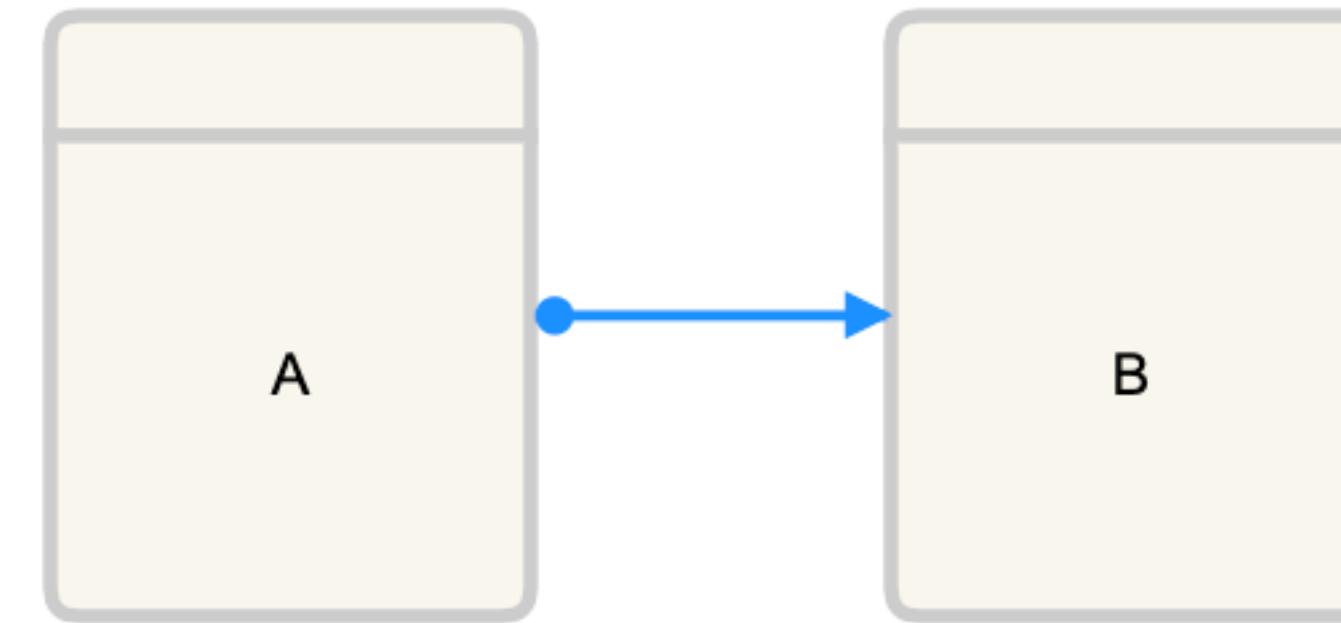
(Might involve events outside the scope of the rule)

$\langle E, D, F, G, \text{B} \rangle$	✓
$\langle G, C, F, D, G \rangle$	✓
$\langle C, \text{A}, \text{B}, D, \text{B} \rangle$	✓
$\langle G, C, \text{B}, \text{A}, D \rangle$	✗
$\langle \text{A}, F, \text{A}, D, \text{B} \rangle$	✓
$\langle \text{A}, D, \text{B}, G, \text{A} \rangle$	✗

2.Compliance via Conformance Checking



Compliance Rule (Response of a Task)



Full Compliant

All traces satisfy the rule



Partially (Weakly) Compliant

At least a trace satisfy the rule



Violated

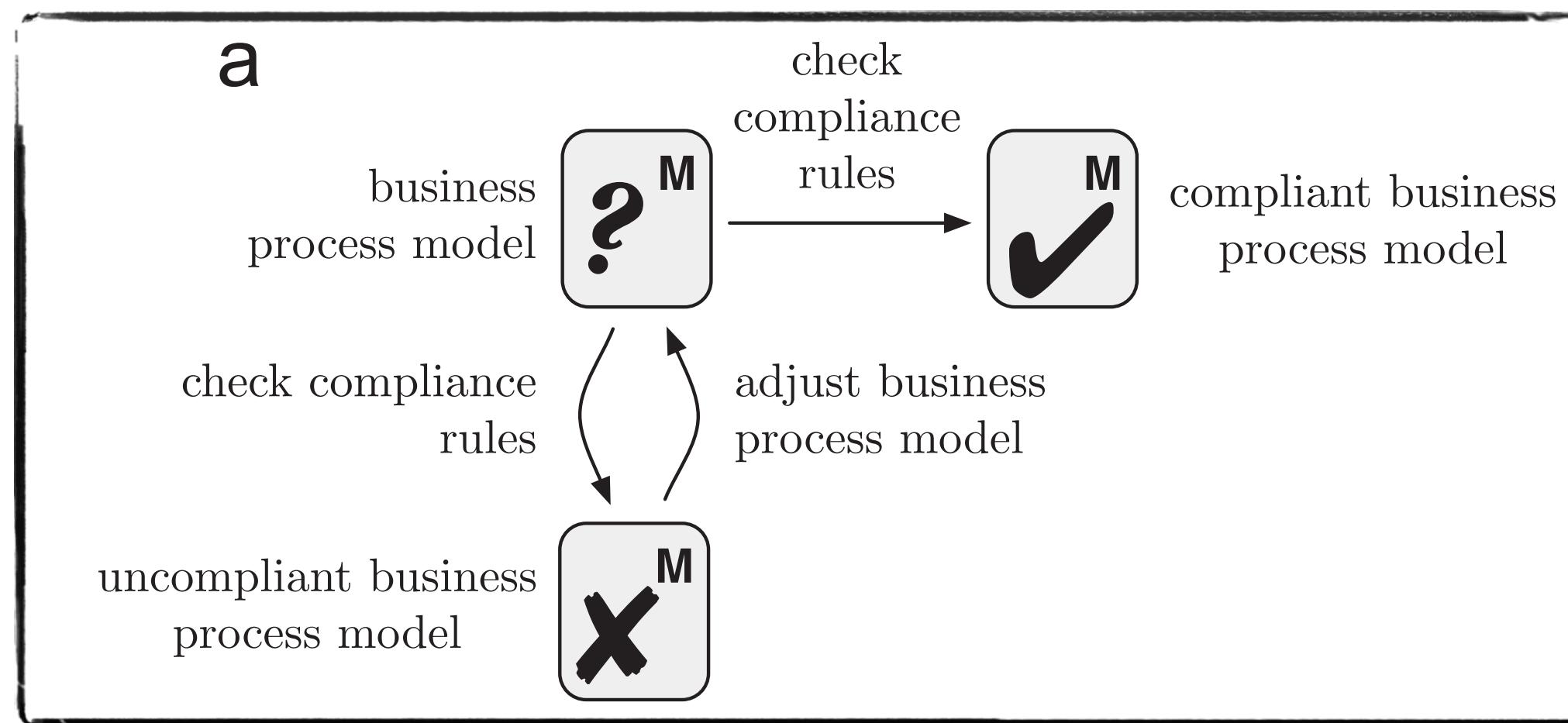
All traces violate the rule

Event log

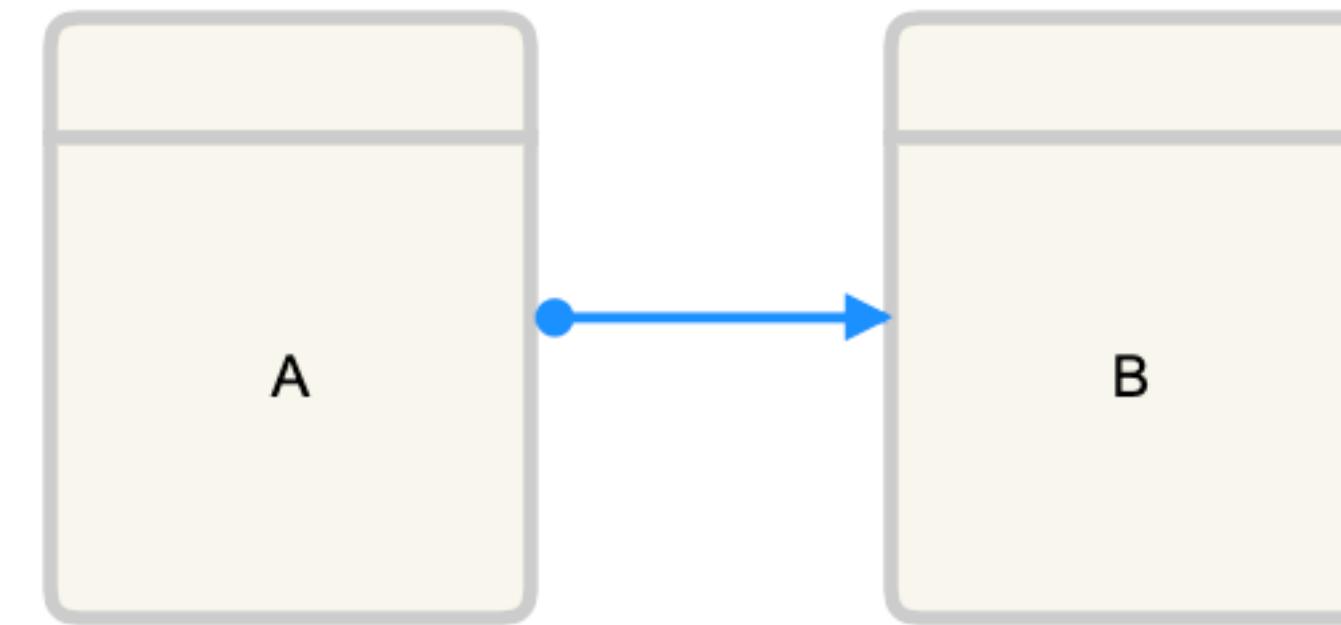
(Might involve events outside the scope of the rule)

$\langle E, D, F, G, \textcolor{blue}{B} \rangle$	✓
$\langle G, C, F, D, G \rangle$	✓
$\langle C, \textcolor{green}{A}, \textcolor{blue}{B}, D, \textcolor{blue}{B} \rangle$	✓
$\langle G, C, \textcolor{blue}{B}, \textcolor{green}{A}, D \rangle$?
$\langle \textcolor{green}{A}, F, \textcolor{green}{A}, D, \textcolor{blue}{B} \rangle$	✓
$\langle \textcolor{green}{A}, D, \textcolor{blue}{B}, G, \textcolor{green}{A} \rangle$?

2.Compliance via Conformance Checking



Compliance Rule (Response of a Task)



Full Compliant

All traces satisfy the rule



Partially (Weakly) Compliant

At least a trace satisfy the rule



Violated

All traces violate the rule

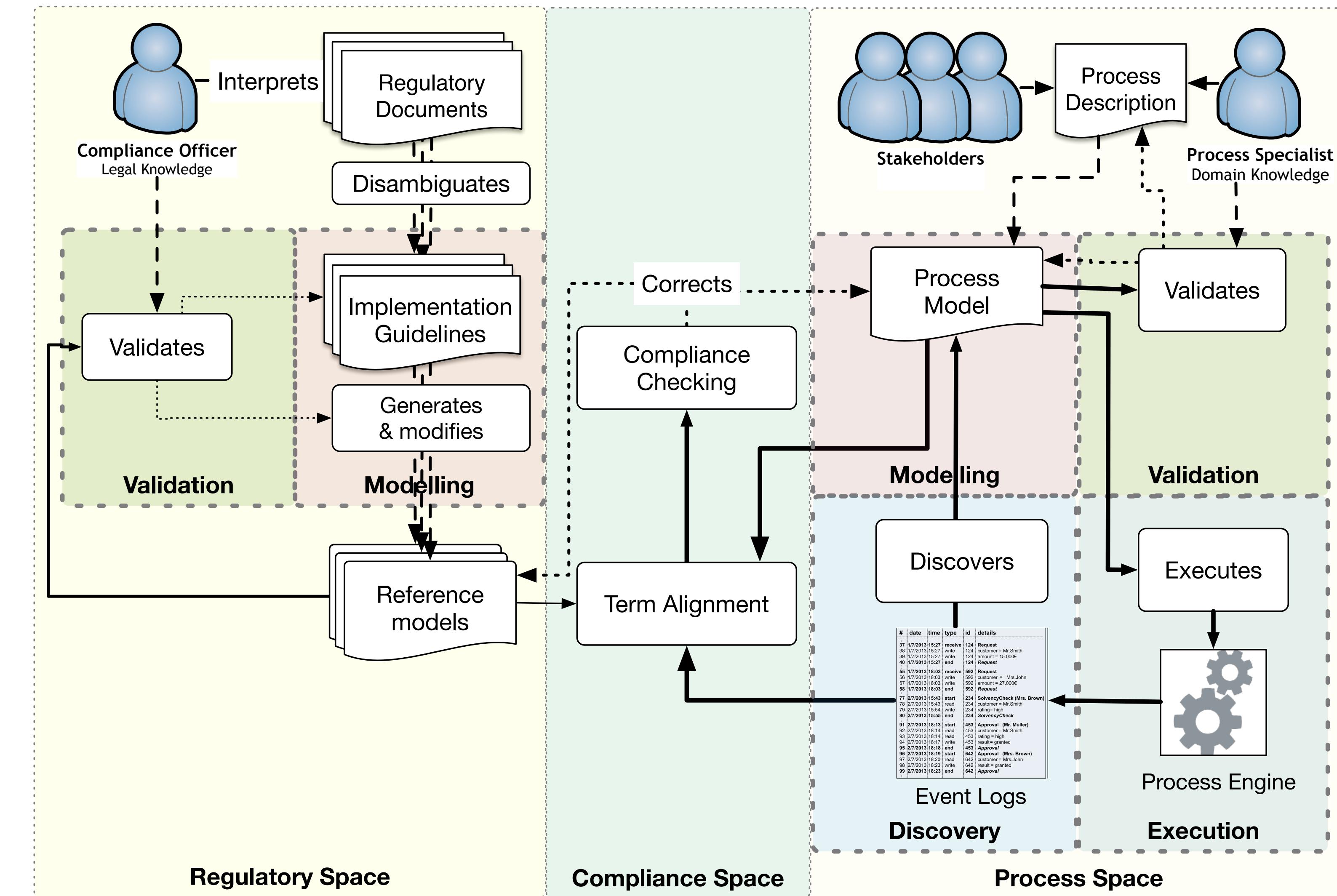


Event log

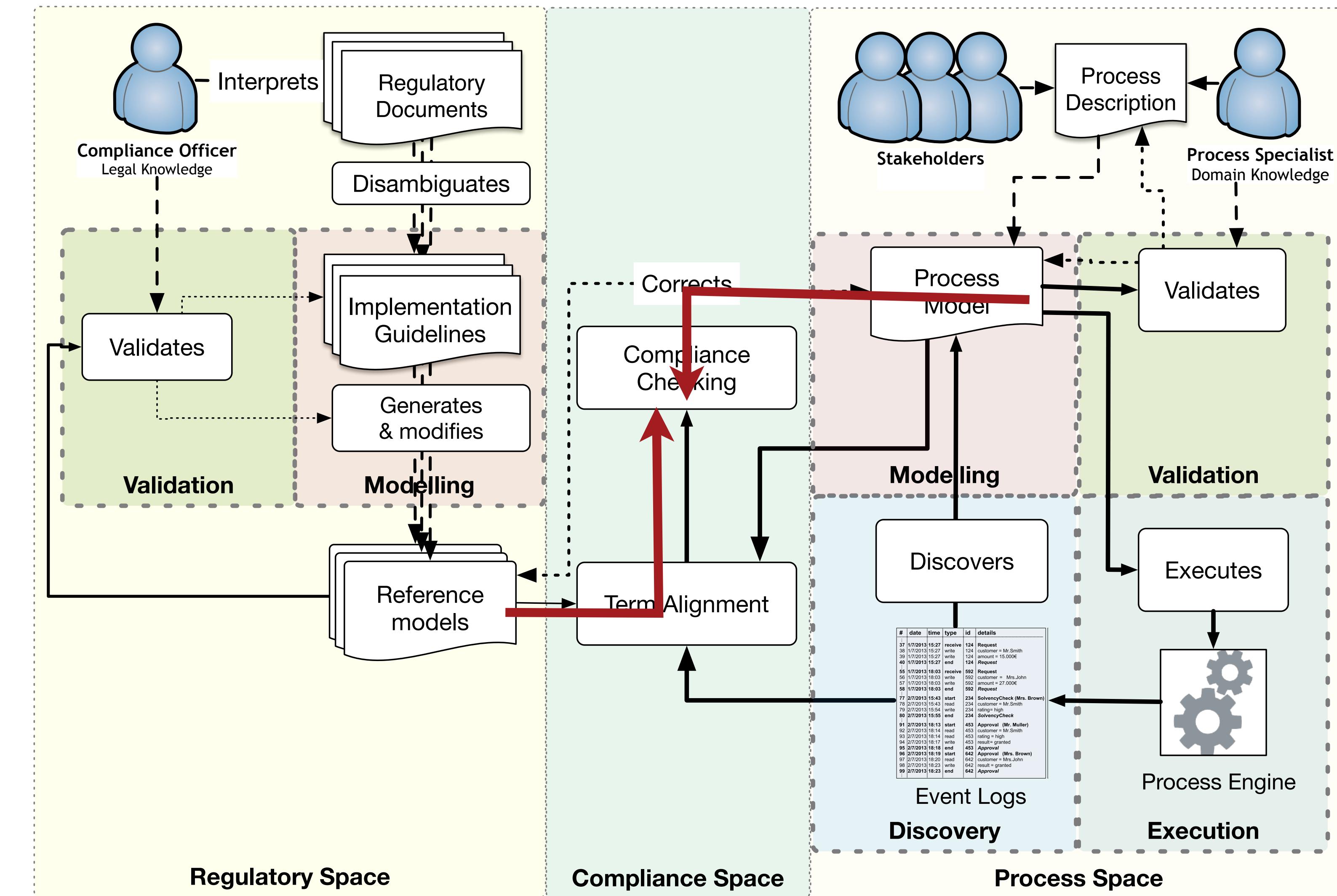
(Might involve events outside the scope of the rule)

$\langle E, D, F, G, \text{B} \rangle$	✓
$\langle G, C, F, D, G \rangle$	✓
$\langle C, \text{A}, \text{B}, D, \text{B} \rangle$	✓
$\langle G, C, \text{B}, \text{A}, D \rangle$?
$\langle \text{A}, F, \text{A}, D, \text{B} \rangle$	✓
$\langle \text{A}, D, \text{B}, G, \text{A} \rangle$?

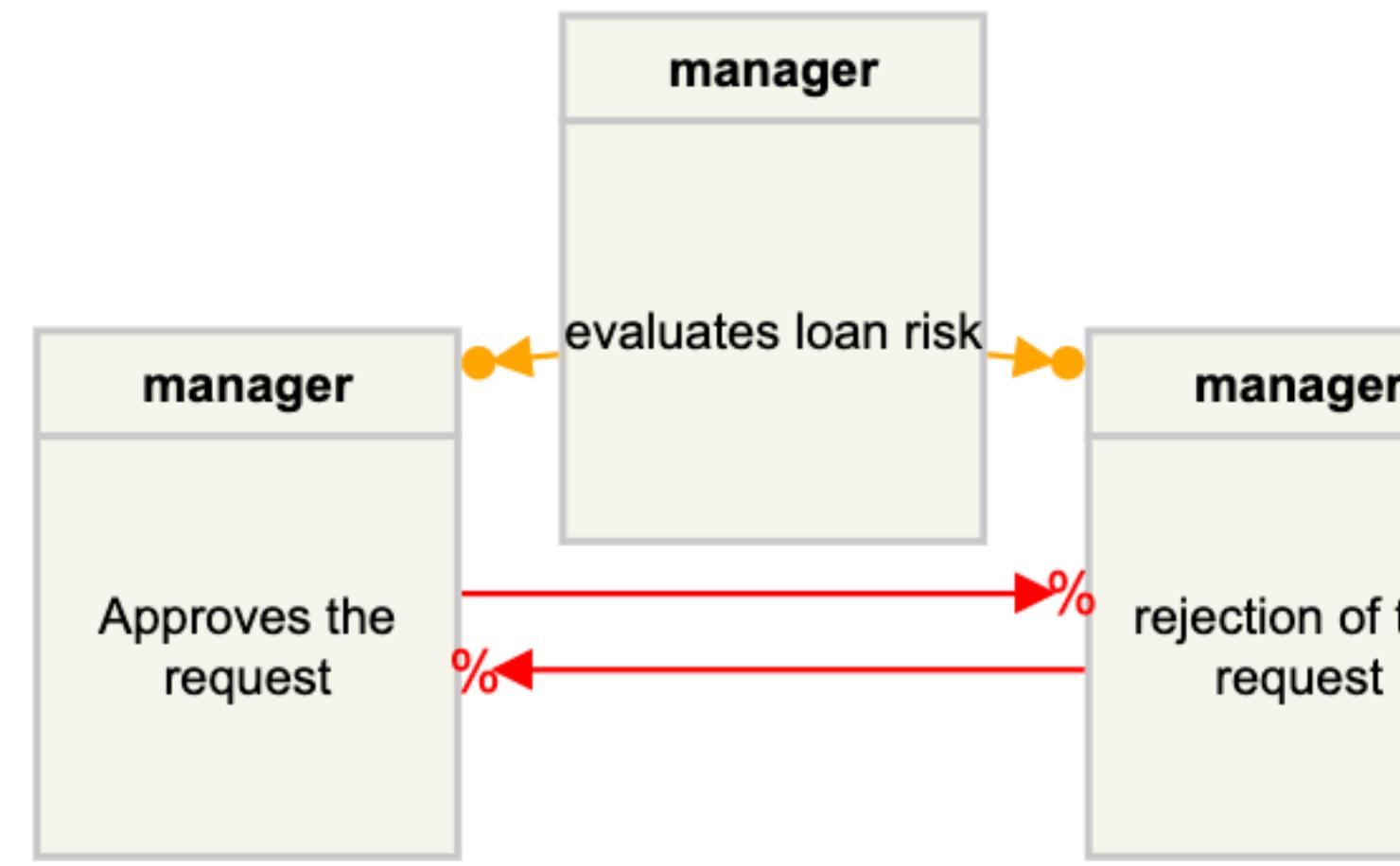
3. Compliance as Process Refinement



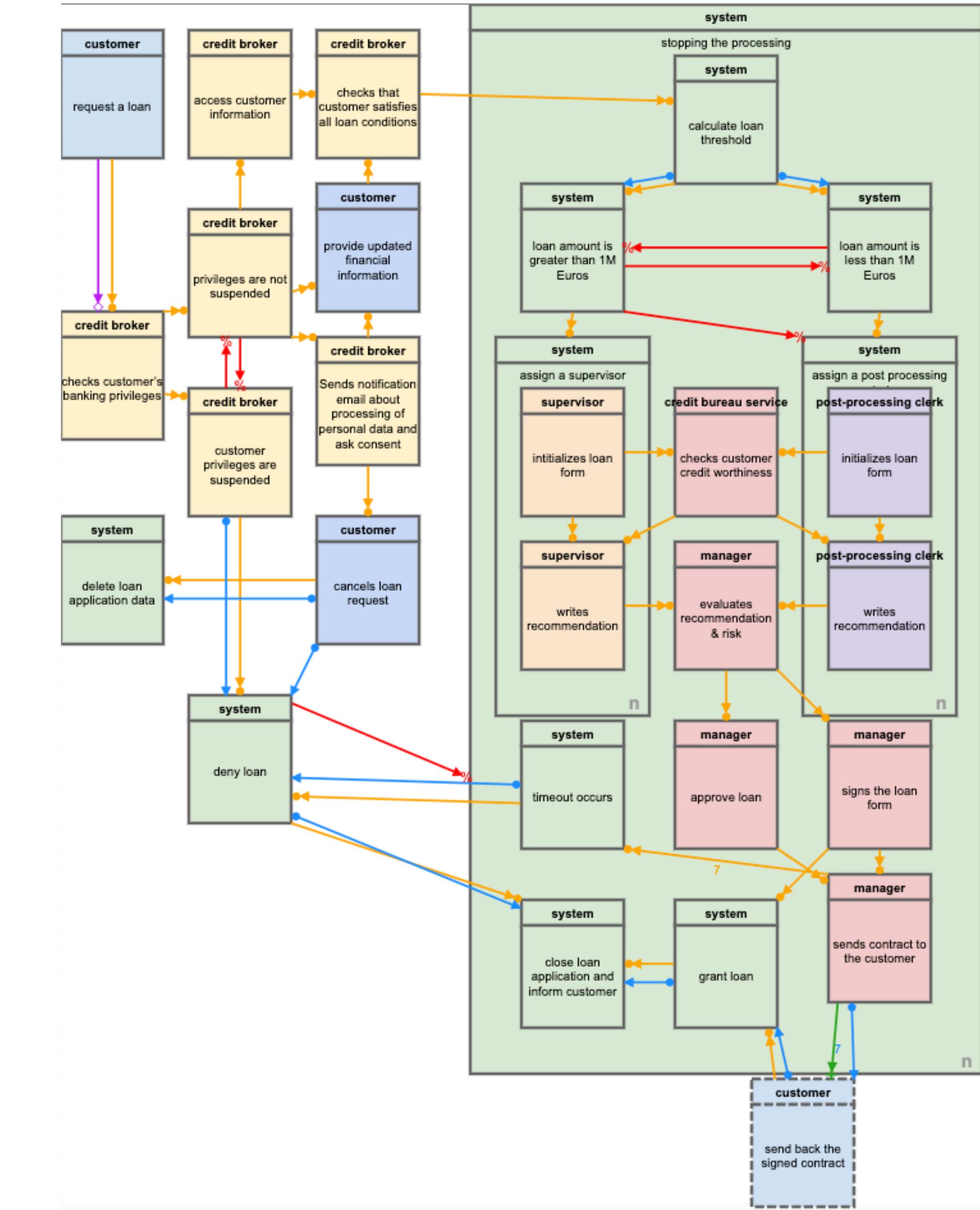
3. Compliance as Process Refinement



3. Compliance as Process Refinement

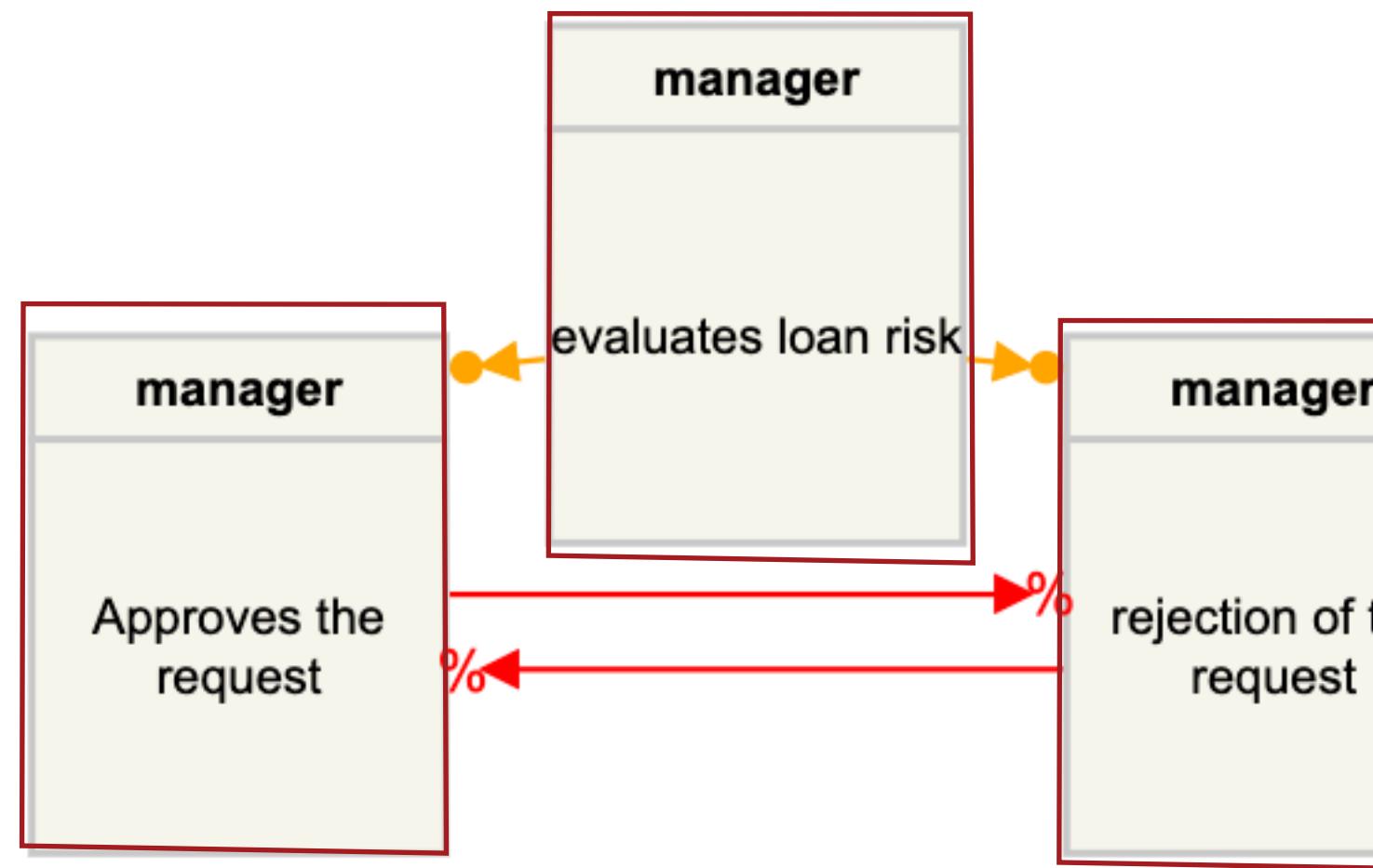


Normative Model



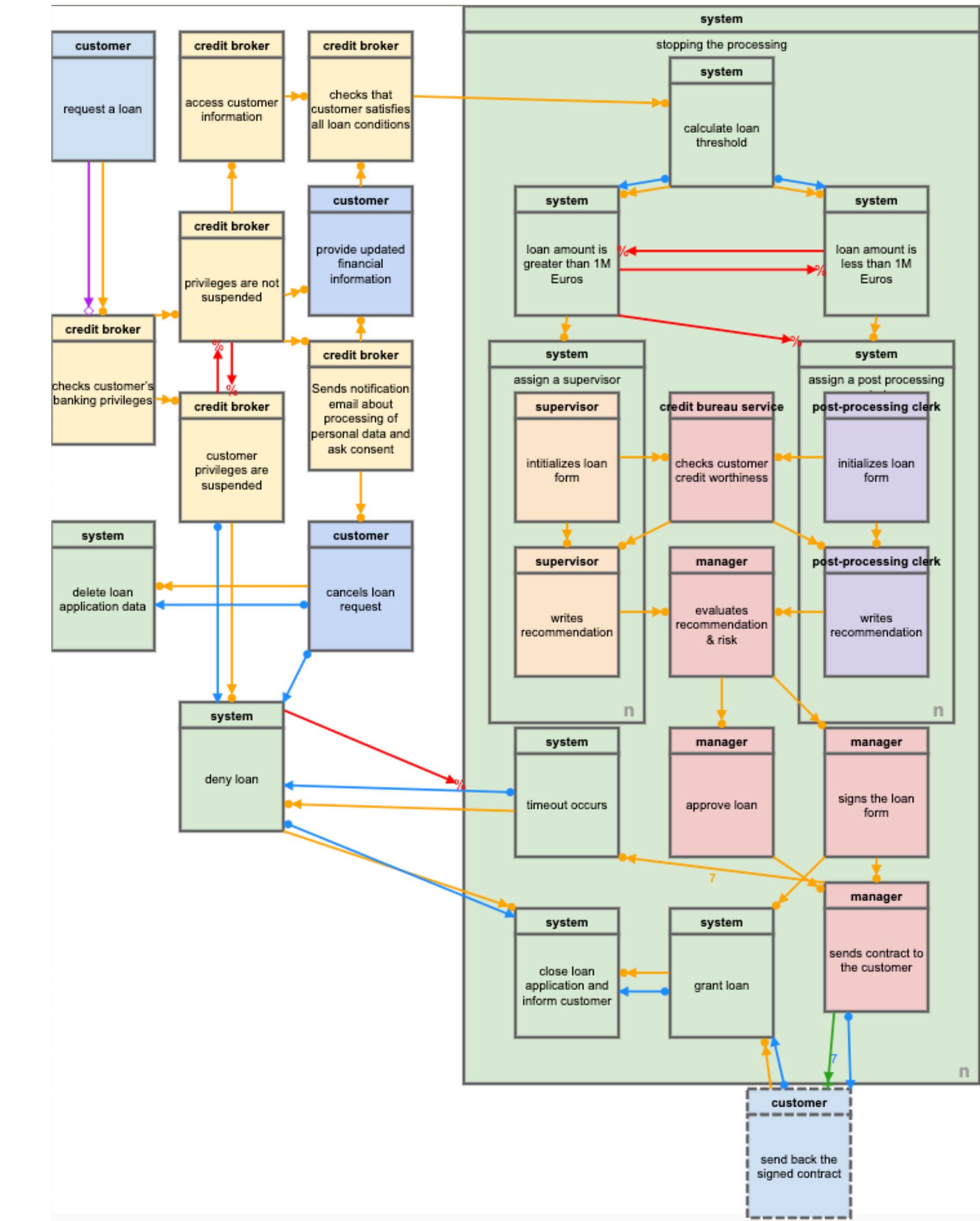
Process Model

3. Compliance as Process Refinement



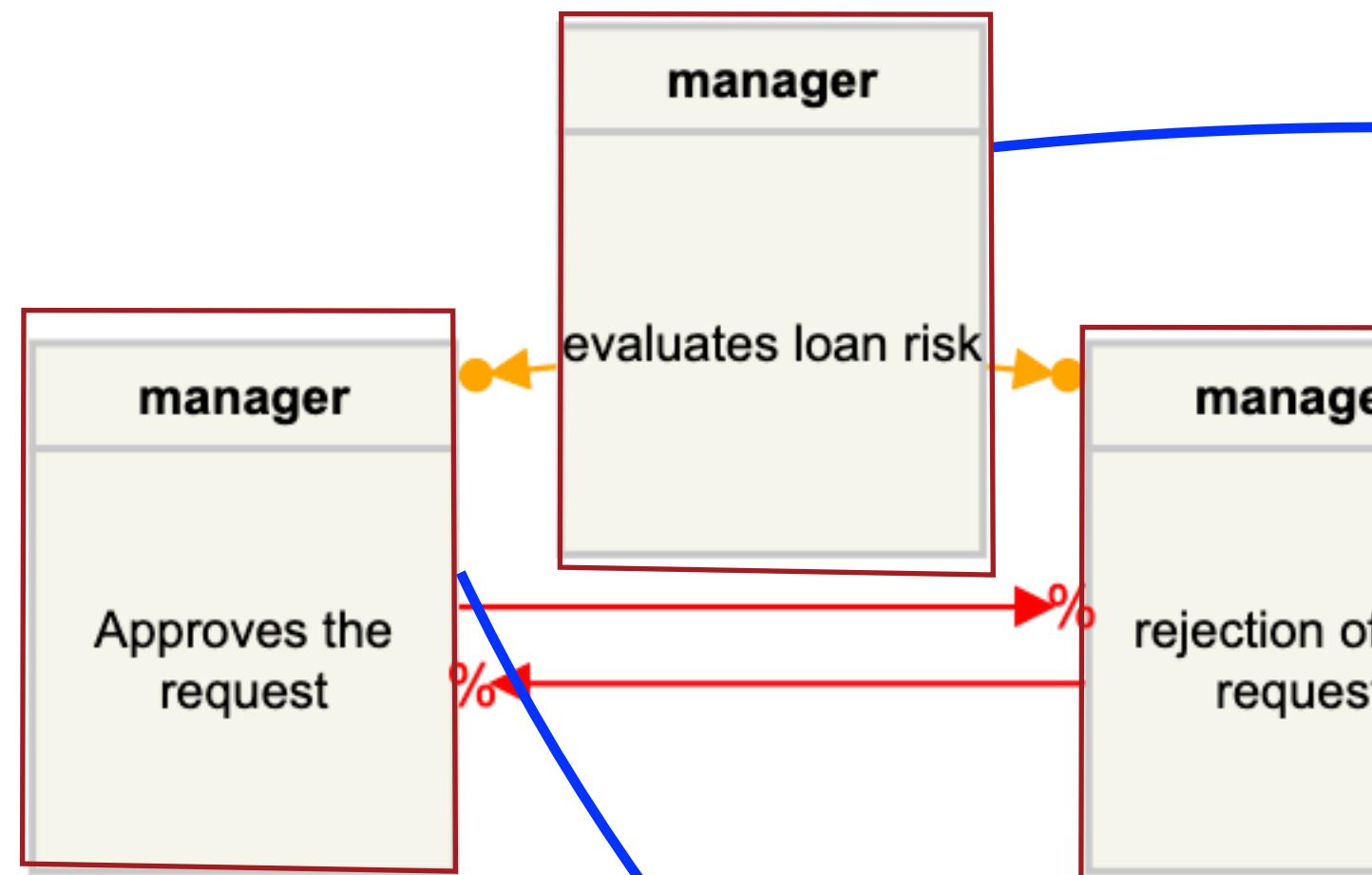
The branch office manager checks whether the risks calculated by the supervisor or the clerk, are acceptable, after which he makes either the final approval or the rejection of the request

Normative Model



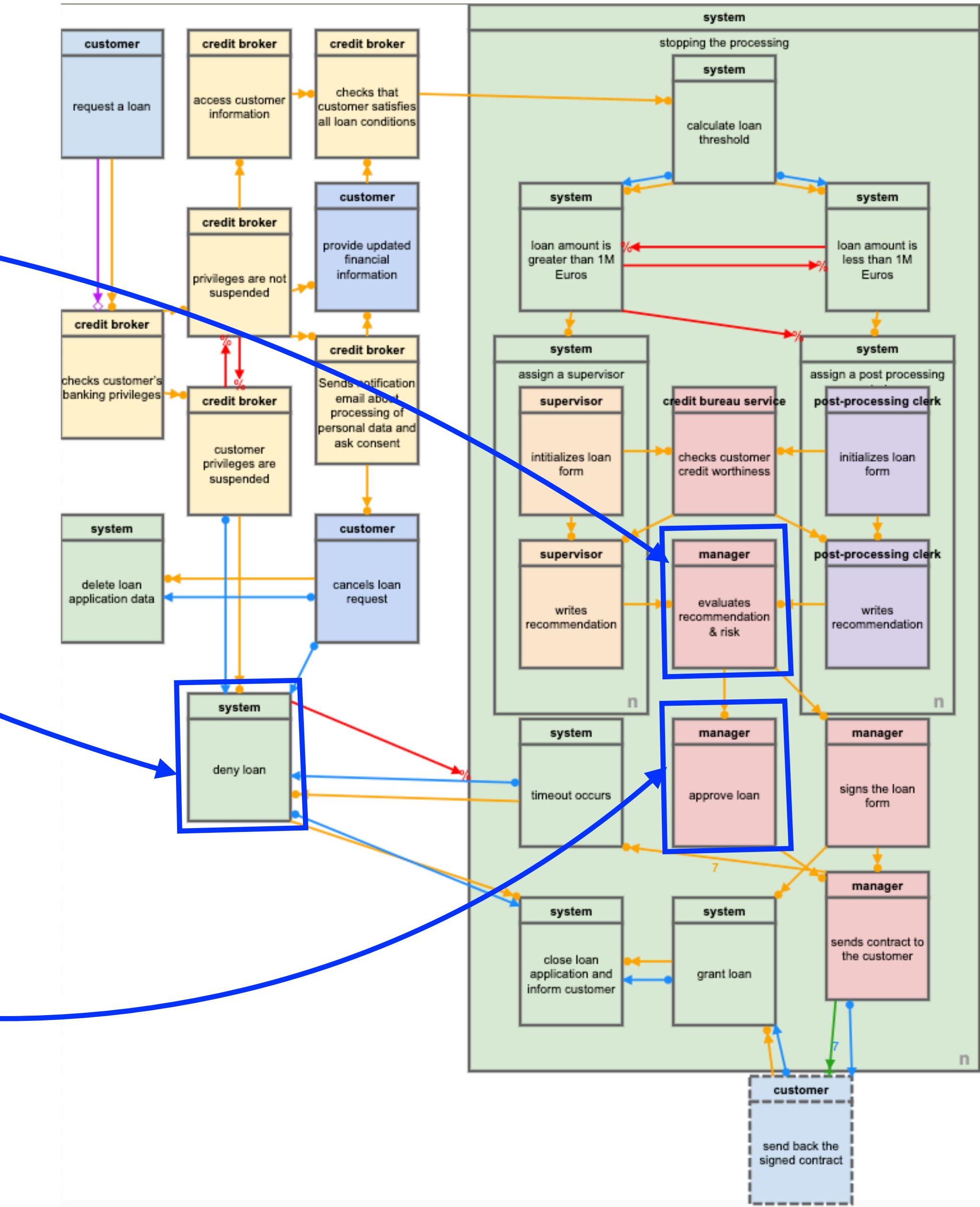
Process Model

3. Compliance as Process Refinement



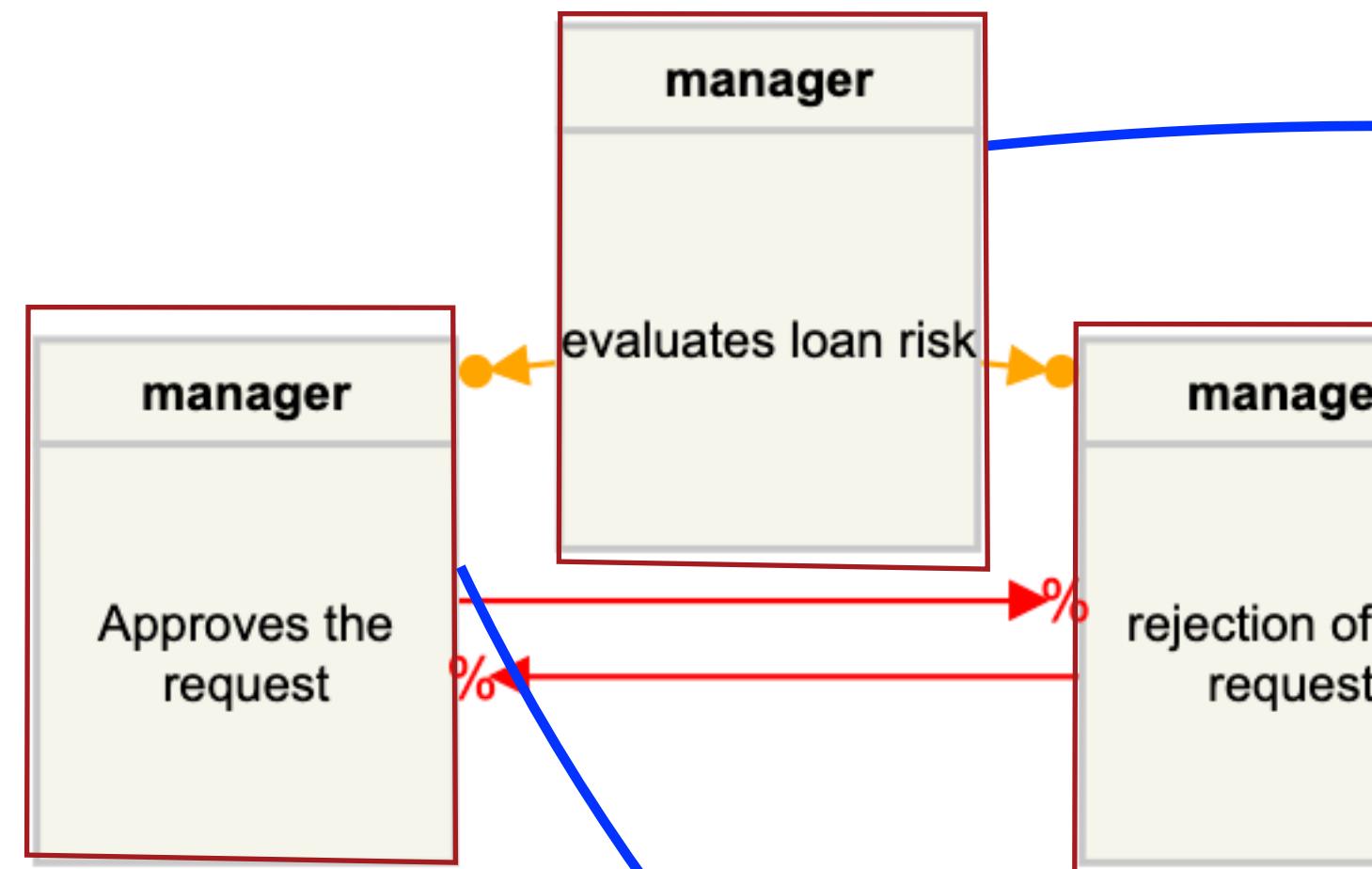
The branch office manager checks whether the risks calculated by the supervisor or the clerk, are acceptable, after which he makes either the final approval or the rejection of the request

Normative Model



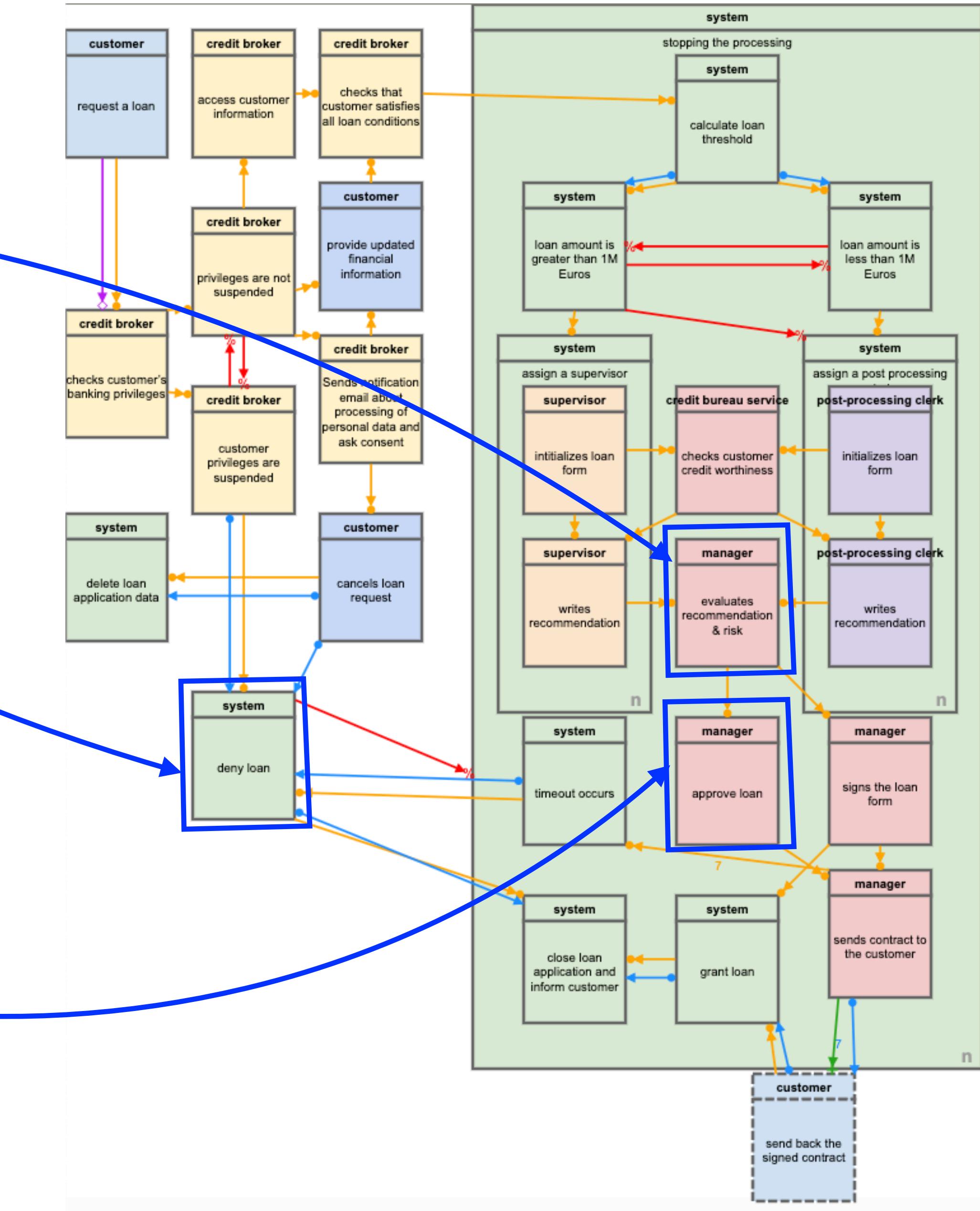
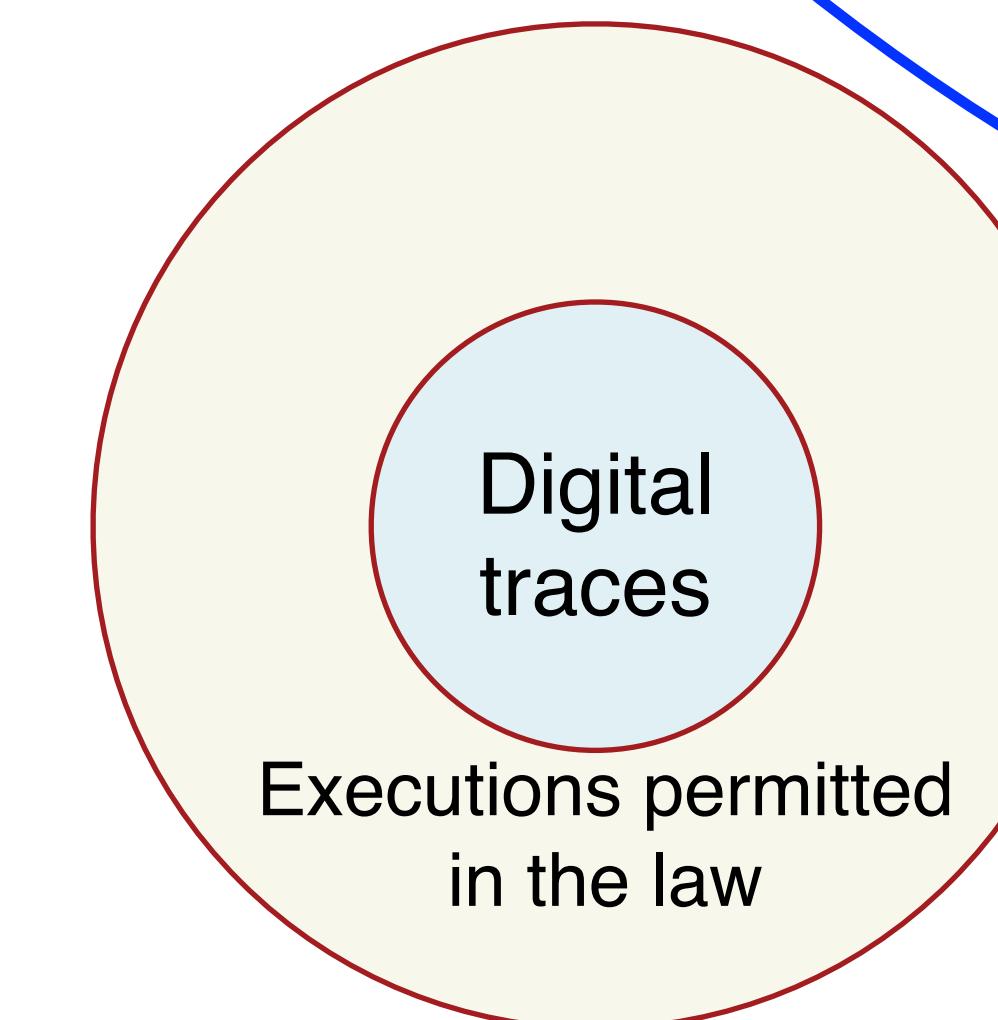
Process Model

3. Compliance as Process Refinement



The branch office manager checks whether the risks calculated by the supervisor or the clerk, are acceptable, after which he makes either the final approval or the rejection of the request

Normative Model



Process Model

ALIGNING LAWS AND PROCESSES

Definition 3 (Term Alignment & Target events). Let $L, L' \subseteq \mathcal{L}$. A term alignment is the function $g : L \rightarrow \mathcal{P}(L')$, such that $\forall l \in L. g(l) \neq \emptyset$. If P, Q are DCR processes with labels L, L' respectively, we say that g is a term alignment from P to Q if g is a term alignment from L to L' . Moreover, we define the target event of g for e in P as $tg(g, e, P) = \{\lambda^{-1}(g(l)) \mid \lambda(l) \in \text{dom}(g) \wedge e \in \text{fe}(P)\}$.

Event in Legislation	Activity/event in Process Model
B_1 : Process personal data	A2: Submit change request
B_2 : Right to object	A3: Cancel request
B_3 : Stop processing	A17: Delete request

ALIGNING LAWS AND PROCESSES

Definition 3 (Term Alignment & Target events). Let $L, L' \subseteq \mathcal{L}$. A term alignment is the function $g : L \rightarrow \mathcal{P}(L')$, such that $\forall l \in L. g(l) \neq \emptyset$. If P, Q are DCR processes with labels L, L' respectively, we say that g is a term alignment from P to Q if g is a term alignment from L to L' . Moreover, we define the target event of g for e in P as $tg(g, e, P) = \{\lambda^{-1}(g(l)) \mid \lambda(l) \in \text{dom}(g) \wedge e \in \text{fe}(P)\}$.

Event in Legislation	Activity/event in Process Model
B_1 : Process personal data	A2: Submit change request
B_2 : Right to object	A3: Cancel request
B_3 : Stop processing	A17: Delete request

Definition 4 (Instances of a Compliance Rule). Let $G = \{g_1, \dots, g_n\}$ be a set of term alignments from P to Q , an instance of P under g in Q , written $P \downarrow_g Q$ for $g \in G$, is $P\sigma$ with labelling $\lambda'(e) = g(\lambda(e))$ when defined, and $\lambda(e)$ otherwise, such that $\sigma = \{f_1, \dots, f_n/e_1, \dots, e_n\}$ and $f_i = tg(g, e_i, P), \forall 1 \leq i \leq n$. The set $Inst(P, G, Q)$ denotes the set of all the instances of P under G in Q , that is $\{R \mid R = P \downarrow_g Q \wedge g \in G\}$.

ALIGNING LAWS AND PROCESSES

Definition 3 (Term Alignment & Target events). Let $L, L' \subseteq \mathcal{L}$. A term alignment is the function $g : L \rightarrow \mathcal{P}(L')$, such that $\forall l \in L. g(l) \neq \emptyset$. If P, Q are DCR processes with labels L, L' respectively, we say that g is a term alignment from P to Q if g is a term alignment from L to L' . Moreover, we define the target event of g for e in P as $tg(g, e, P) = \{\lambda^{-1}(g(l)) \mid \lambda(l) \in \text{dom}(g) \wedge e \in \text{fe}(P)\}$.

Event in Legislation	Activity/event in Process Model
B_1 : Process personal data	A2: Submit change request
B_2 : Right to object	A3: Cancel request
B_3 : Stop processing	A17: Delete request

Definition 4 (Instances of a Compliance Rule). Let $G = \{g_1, \dots, g_n\}$ be a set of term alignments from P to Q , an instance of P under g in Q , written $P \downarrow_g Q$ for $g \in G$, is $P\sigma$ with labelling $\lambda'(e) = g(\lambda(e))$ when defined, and $\lambda(e)$ otherwise, such that $\sigma = \{f_1, \dots, f_n/e_1, \dots, e_n\}$ and $f_i = tg(g, e_i, P), \forall 1 \leq i \leq n$. The set $Inst(P, G, Q)$ denotes the set of all the instances of P under G in Q , that is $\{R \mid R = P \downarrow_g Q \wedge g \in G\}$.

Term Alignment	Label Reference Model	Event P_{spec}	Label Process Model
g_3	A2: submit a change request Finish Processing request Cancel Processing	f_1 f_2 f_3	A2: submit a change request A15: Amend initial contract with approved change A3: Delete request
g_4	A2: submit a change request Finish Processing request Cancel Processing	f_1 f_4 f_3	A2: submit a change request A16: Receive reason for change rejection A15: Delete request

Compliance as process refinement

- **Process Refinement.** Let P_{law} , P_{process} be processes. We say that P_{process} is a refinement of P_{law} (written $P_{\text{process}} \sqsubseteq P_{\text{law}}$) iff $\text{lang}(P_{\text{process}})I_{\text{alph}}(P_{\text{law}}) \subseteq \text{lang}(P_{\text{law}})$.

Compliance as process refinement

- **Process Refinement.** Let P_{law} , P_{process} be processes. We say that P_{process} is a refinement of P_{law} (written $P_{\text{process}} \sqsubseteq P_{\text{law}}$) iff $\text{lang}(P_{\text{process}}) \cap \text{alph}(P_{\text{law}}) \subseteq \text{lang}(P_{\text{law}})$.

Obs: For an arbitrary P , $\text{lang}(P)$ can be regular or ω -regular, thus not very practical

Compliance as process refinement

- **Process Refinement.** Let P_{law} , P_{process} be processes. We say that P_{process} is a refinement of P_{law} (written $P_{\text{process}} \sqsubseteq P_{\text{law}}$) iff $\text{lang}(P_{\text{process}})I_{\text{alph}}(P_{\text{law}}) \subseteq \text{lang}(P_{\text{law}})$.
Obs: For an arbitrary P , $\text{lang}(P)$ can be regular or ω -regular, thus not very practical
- **Alternative: Refinement as composition (Debois et al 2017).** Idea: R , P are in refinement, iff we can merge the events of R and P and still behave as R .
- Merge (\oplus) is defined as the partial relation between markings agreeing on their overlap:

$$(M_1, e : m) \oplus (M_2, e : m) = (M_1 \oplus M_2), e : m$$

$$(M_1, e : m) \oplus M_2 = (M_1 \oplus M_2), e : m \quad \text{when } e \notin \text{dom}(M_2)$$

$$M_1 \oplus (M_2, e : m) = (M_1 \oplus M_2), e : m \quad \text{when } e \notin \text{dom}(M_1).$$
- This extends to processes in the standard way. For $P=[M]\lambda_1 T$ and $Q=[N]\lambda_2 U$, then $P \oplus Q = [M \oplus N](\lambda_1 \cup \lambda_2)(T \parallel U)$
- **Refines:** Q refines P iff $P \oplus Q \sqsubseteq P$

Compliance as refinement

- **Intuition:** when checking compliance between rule R , a process P and a term alignment mapping labels in R to P . Compliance requires us
 1. Generate all instances of R in P and
 2. Check whether the merge of each instance with P is compatible (i.e. refines) the instance.

(Compliance). Let P, R be DCR processes, and G be a set of term alignments from R to P . We say that P is compliant with R under G , written $P \leq_s^G R$ if $\forall R_i \in Inst(R, G, P), P$ refines R_i .

Algorithmic Compliance Checking

- Language inclusion is NP-hard for DCR Graphs (Debois et al 2017).
- We give efficient static guarantees for process refinement.
- For DCR graphs P, R , an implementation P is **non-invasive** to its specification R iff their markings are compatible, and P does not induce inclusions, exclusions or responses to the free events in R .

Definition 5.7 (Non-invasiveness). *Let $P = [M_P] \lambda_P T_P$ and R be marking compatible processes. We say that P is non-invasive for R iff for every context $C[-]$, such that $T_P = C[e \rightarrow \% f]$, $T_P = C[e \rightarrow + f]$ or $T_P = C[e \bullet \rightarrow f]$, $f \notin \text{fe}(R)$.*

- Complexity: an algorithm only needs to check for each inclusion, response and exclusion relation in P if the target event exists in R . In its worst case, we will have n relations in T_P . Assuming a standard set membership operation of **O(1)**, then the worst complexity is **O(n)**.

Implementing Compliance Checking

Fig. 8. Non-invasiveness checking, $R \models T$

$$\frac{}{R \models 0} \text{ [I-NIL]} \quad \frac{f \notin \text{fe}(R)}{R \models e \xrightarrow{\%} f} \text{ [I-EXCL]} \quad \frac{f \notin \text{fe}(R)}{R \models e \xrightarrow{+} f} \text{ [I-INCL]} \quad \frac{f \notin \text{fe}(R)}{R \models e \bullet \xrightarrow{} f} \text{ [I-RESP]}$$

$$\frac{R \models T \quad R \models U}{R \models T \parallel U} \text{ [I-COMP]}$$

Implementing Compliance Checking

Fig. 8. Non-invasiveness checking, $R \models T$

$$\frac{}{R \models 0} \text{ [I-NIL]} \quad \frac{f \notin \text{fe}(R)}{R \models e \xrightarrow{\%} f} \text{ [I-EXCL]} \quad \frac{f \notin \text{fe}(R)}{R \models e \xrightarrow{+} f} \text{ [I-INCL]} \quad \frac{f \notin \text{fe}(R)}{R \models e \bullet \xrightarrow{} f} \text{ [I-RESP]}$$

$$\frac{R \models T \quad R \models U}{R \models T \parallel U} \text{ [I-COMP]}$$

Algorithm 1 refines relation

Require: DCR graphs $P = [M_p] T_p, Q = [M_q] T_q$

```

1: function REFINES( $P, Q$ )
2:   try
3:     DCR graph  $merge \leftarrow P \oplus Q$ 
4:     return  $merge \models T_p$ 
5:   catch  $P \oplus Q$  undefined
6:     return false
7:   end try

```

Implementing Compliance Checking

Fig. 8. Non-invasiveness checking, $R \models T$

$$\frac{}{R \models 0} \text{ [I-NIL]} \quad \frac{f \notin \text{fe}(R)}{R \models e \xrightarrow{\%} f} \text{ [I-EXCL]} \quad \frac{f \notin \text{fe}(R)}{R \models e \xrightarrow{+} f} \text{ [I-INCL]} \quad \frac{f \notin \text{fe}(R)}{R \models e \bullet \xrightarrow{} f} \text{ [I-RESP]}$$

$$\frac{R \models T \quad R \models U}{R \models T \parallel U} \text{ [I-COMP]}$$

Algorithm 1 refines relation

Require: DCR graphs $P = [M_p] T_p, Q = [M_q] T_q$

```

1: function REFINES( $P, Q$ )
2:   try
3:     DCR graph  $merge \leftarrow P \oplus Q$ 
4:     return  $merge \models T_p$ 
5:   catch  $P \oplus Q$  undefined
6:     return false
7:   end try

```

Algorithm 2 Compliance Checking algorithm

Require: DCR graphs P, R , term alignment $G = \{g_i \mid 1 \leq i \leq n\}$

Ensure: $n \in \mathbb{N}$

```

1: function COMPLIANCE( $P, R, G$ )
2:    $violations \leftarrow 0$ 
3:   for each  $g \in G$  do
4:     DCR graph  $Q \leftarrow R \downarrow_g P$ 
5:     if not REFINES( $P, Q$ ) then
6:        $violations \leftarrow violations + 1$ 
7:   return  $\frac{|G| - violations}{|G|} * 100\%$ 

```

To summarise

To summarise

- Declarative process models involve multiple dimensions including causal flows, time, internal processes, and data

To summarise

- Declarative process models involve multiple dimensions including causal flows, time, internal processes, and data
- Each dimension extends the operational semantics, and enriches the execution model

To summarise

- Declarative process models involve multiple dimensions including causal flows, time, internal processes, and data
- Each dimension extends the operational semantics, and enriches the execution model
- They are necessary in order to make executable models that are compliant with legislation.

To summarise

- Declarative process models involve multiple dimensions including causal flows, time, internal processes, and data
- Each dimension extends the operational semantics, and enriches the execution model
- They are necessary in order to make executable models that are compliant with legislation.
- Compliance is a semantic correctness property to be ensured at all stages in the business process lifecycle:
 - Design (compliance checking)
 - Execution (compliance monitoring)
 - Auditing (compliance audit)

Reading Material

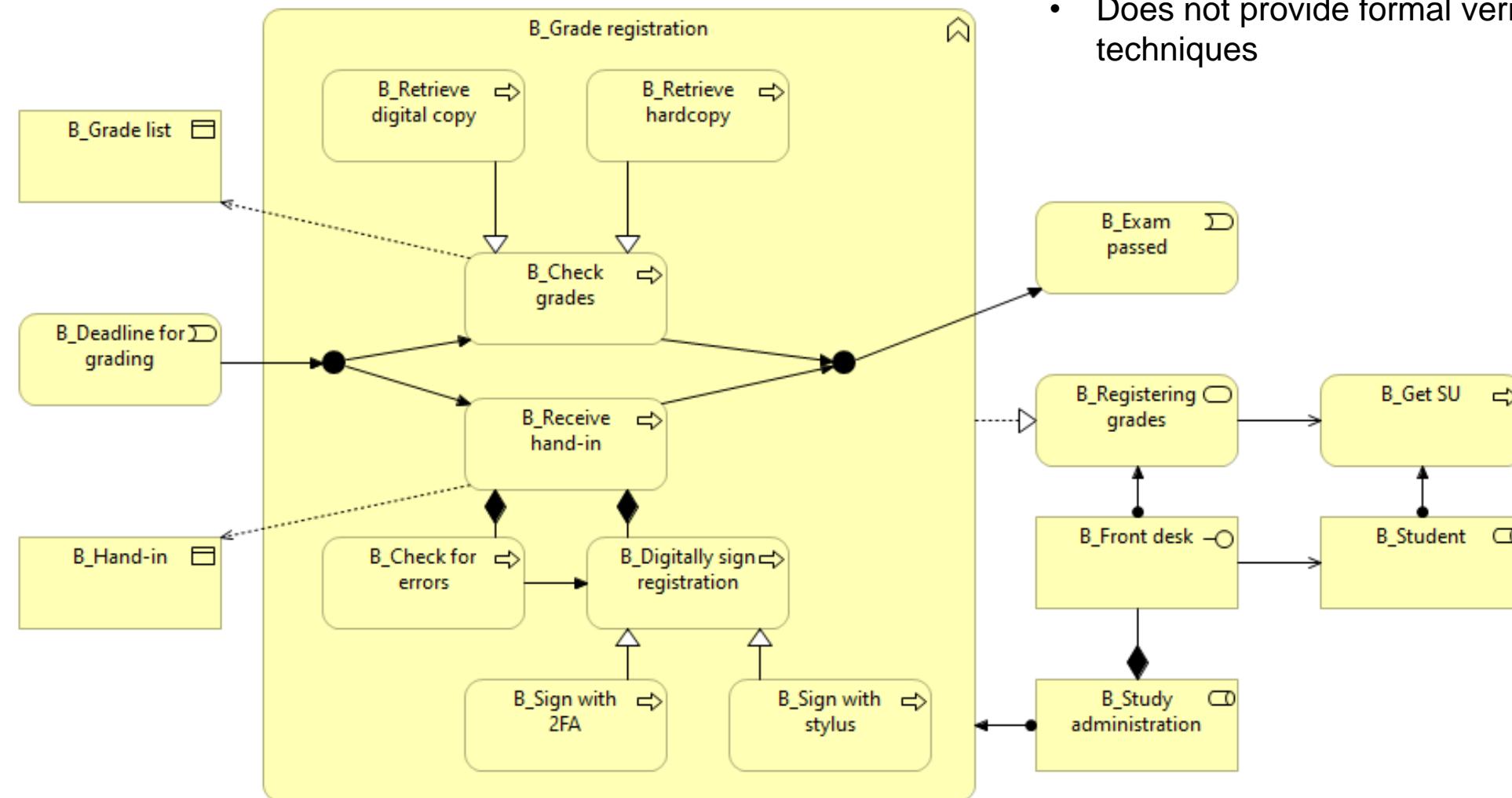
- Debois, S., Hildebrandt, T. T., & Slaats, T. (2018). Replication, refinement & reachability: complexity in dynamic condition-response graphs. *Acta Informatica*, 55(6), 489-520.
- Debois S., Hildebrandt T., Slaats T. (2014) Hierarchical Declarative Modelling with Refinement and Sub-processes. *BPM 2014*: 18-33
- Hildebrandt, T., Mukkamala, R. R., & Slaats, T. (2011, April). Nested dynamic condition response graphs. In International conference on fundamentals of software engineering (pp. 343-350). Berlin, Heidelberg: Springer Berlin Heidelberg.
- López, H. A., Debois, S., Slaats, T., & Hildebrandt, T. T. (2020, April). Business Process Compliance Using Reference Models of Law. In International Conference on Fundamental Approaches to Software Engineering (pp. 378-399).

02291 System Integration

Introduction to BPMN

© Giovanni Meroni

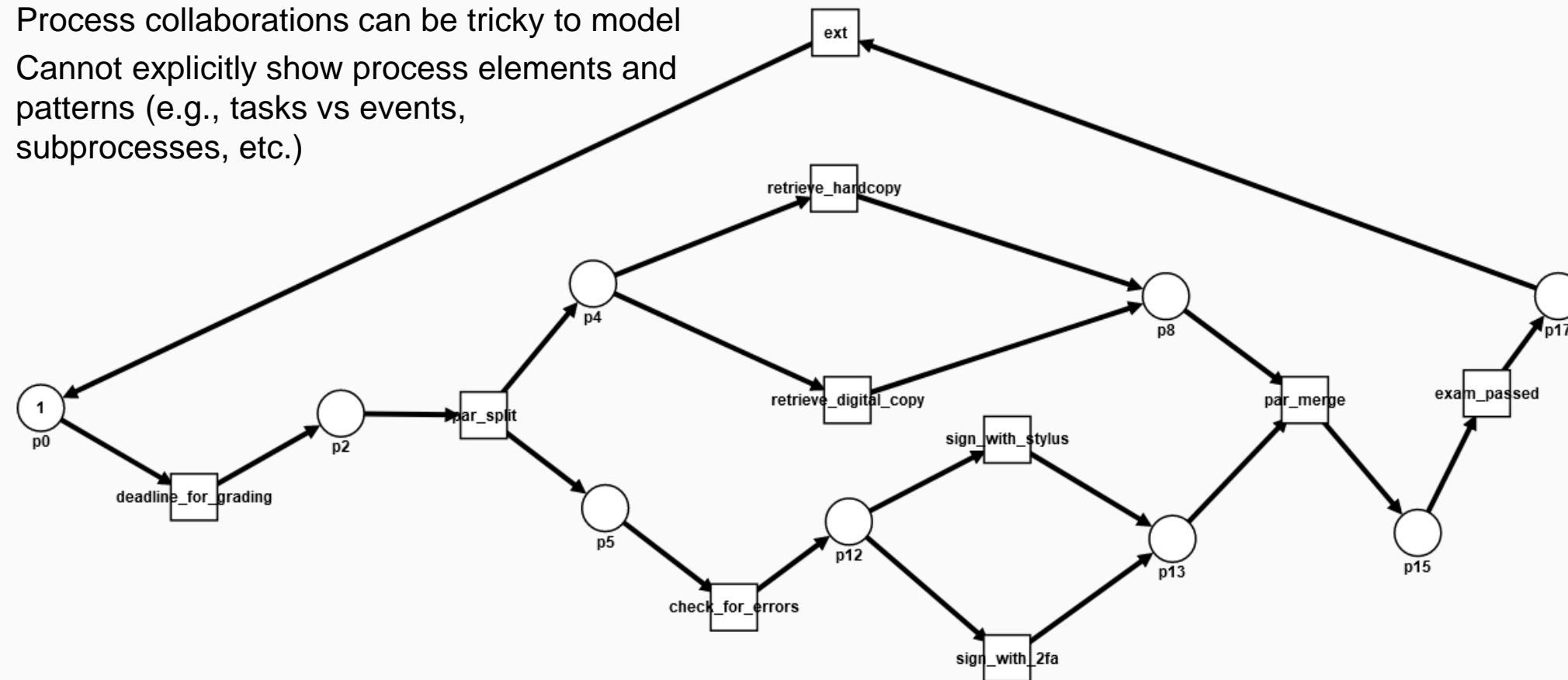
Behavioral models in ArchiMate



- Does not provide formal verification techniques

Behavioral models in Petri Nets

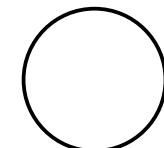
- Difficult to understand by non-experts
- Process collaborations can be tricky to model
- Cannot explicitly show process elements and patterns (e.g., tasks vs events, subprocesses, etc.)



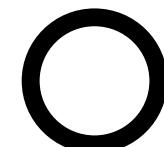
Introducing BPMN

- Business Process Modeling Notation
- A visual language to model business processes
- Can be understood by both computer scientists and non-computer scientists (Business Analyst, Process Designer, ...)
- Supports both the analysis/definition of processes, and the execution through a BPMS (Business Process Management System)
- Supports Orchestration: Workflows, internal processes of an organization, private processes
- Supports Collaboration: Multi-party processes, B2B processes

Main elements



- Start Event
 - There must always be at least one
 - Indicates the start of the process



- End Event
 - There must always be at least one
 - Indicates the end of the process



- Activity
 - It represents a unit of work to be done
 - It can be atomic (task) or indicate a subprocess (more on this later)
- Control flow (also named execution flow)

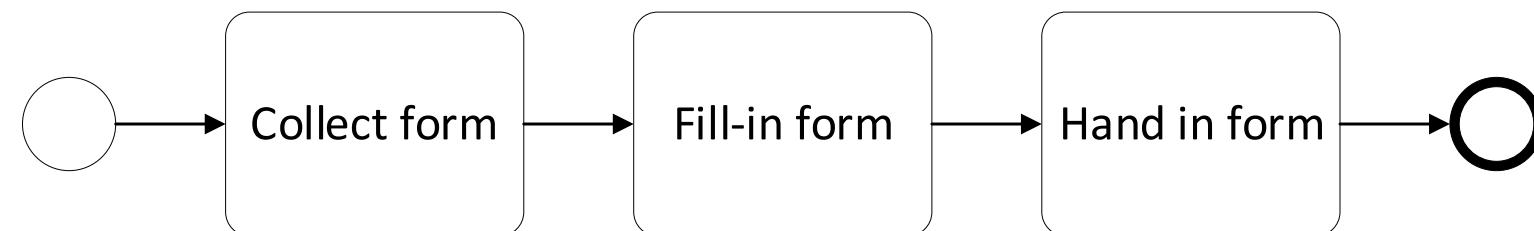


Example 1

To get a refund, each user has to collect a specific form from the secretariat on the ground floor, fill it in, and finally hand it in to the payroll office on the third floor.

Example 1

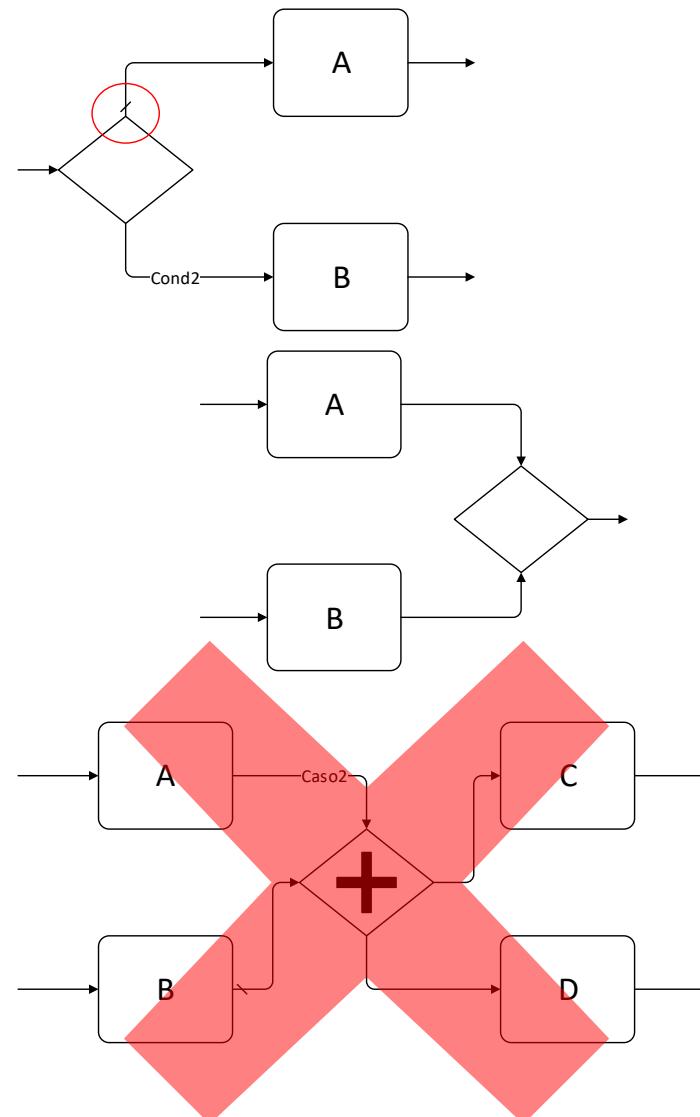
To get a refund, each user has to **collect** a specific **form** from the secretariat on the ground floor, **fill it in**, and finally **hand it in** to the payroll office on the third floor.



Gateways

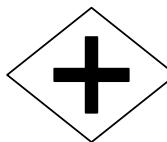
- An activity doesn't always need to be done
- Some activities may be mutually exclusive
- To improve efficiency, it would be desirable to carry out some activities in parallel
- Some activities may need to be repeated until a condition is met
- Gateways allow you to model these behaviors

Gateways

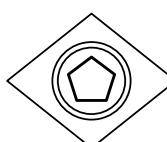
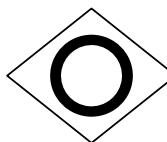
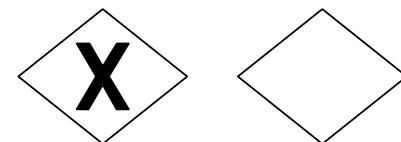


- **Gateway in Split Mode**
 - Splits the execution flow into multiple branches
 - The condition for a branch to be taken is specified on the flow
 - The default flow (if any) is marked with a dash
- **Gateway in Merge Mode**
 - Merges multiple execution flows
- **Gateway in Hybrid Mode (deprecated)**
 - Merges multiple execution flows and immediately splits them again
 - Ambiguous, use a gateway in merge mode followed by a gateway in split mode instead

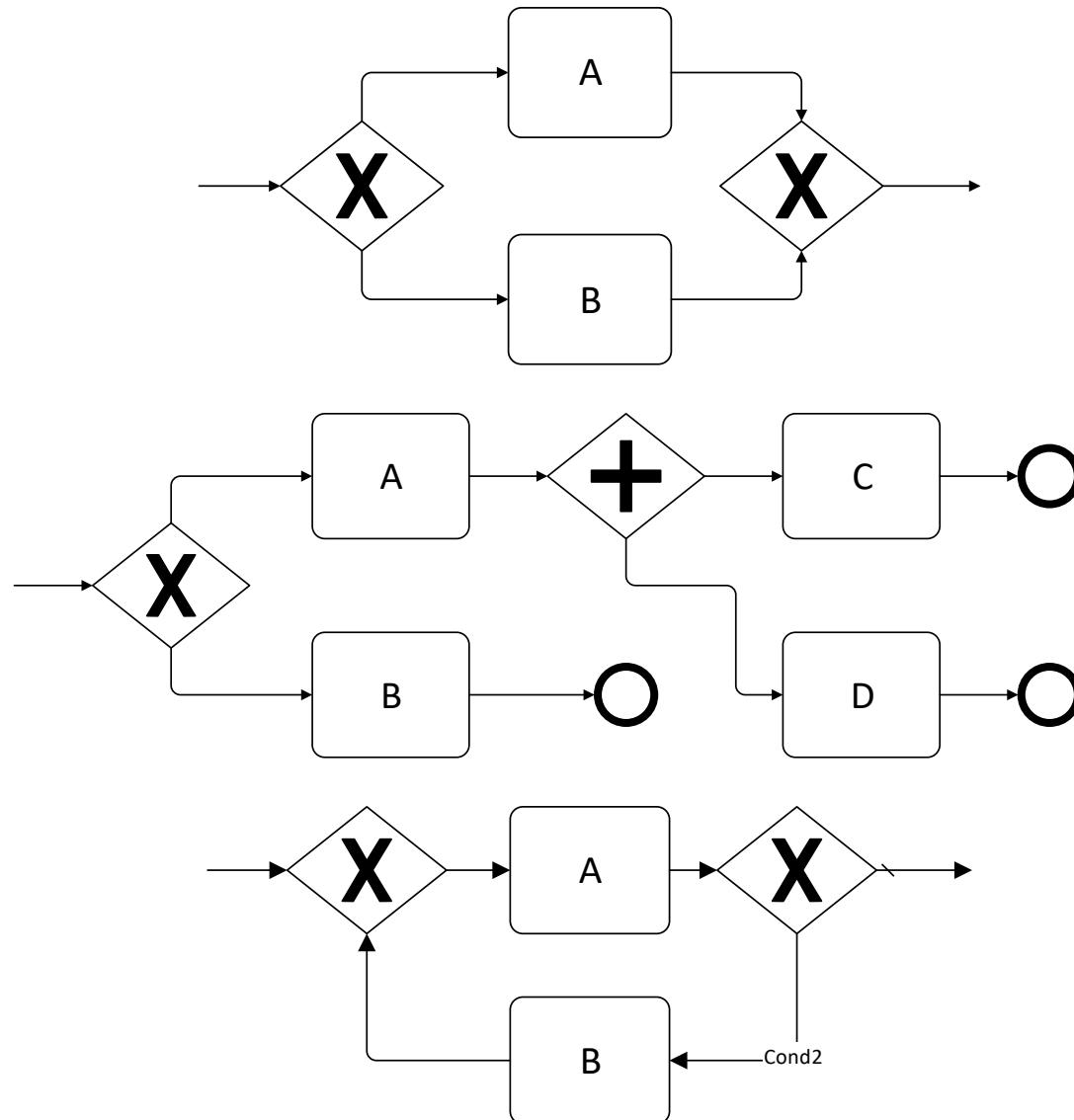
Gateways



- Parallel gateway (AND)
 - In split mode, all branches will be taken
 - In merge mode, all branches being merged must complete
- Exclusive gateway (XOR)
 - In split mode, only one branch will be taken
 - In merge mode, only one branch being merged must complete
- Inclusive gateway (OR)
 - In split mode, at least one branch will be taken
 - In merge mode, all **active** branches being merged must complete
- Event-based Exclusive gateway (XOR)
 - We'll discuss this later when we talk about the events



Gateways

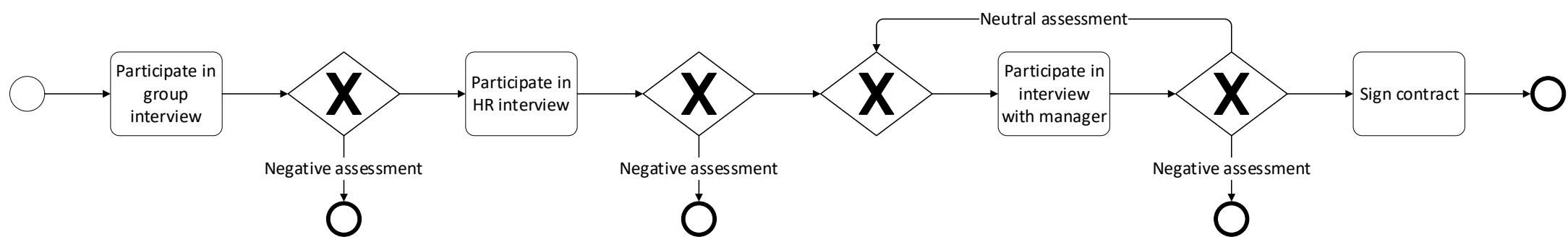


- Branches created by a gateway should be merged with a gateway of the same type
 - This prevents deadlocks or activities executed multiple times
 - When using an inclusive gateway, you must do so
- All branches must be merged or eventually end with an end event
- Loops are modeled using Exclusive gateways

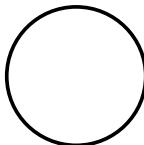
Example 2

The recruitment process of a company is structured as follows: first, a candidate participates in a group interview, which is meant to assess his/her problem-solving skills. If the assessment is negative, the process ends. Otherwise, the candidate participates in an individual interview with the HR staff, which is meant to assess his/her soft skills. If the assessment is negative, the process ends. Otherwise, the candidate participates in an interview with a manager to assess his/her technical skills. If the assessment is positive, the candidate signs the employment contract, and the process ends. If the outcome is negative, the process ends. If the outcome is neutral, the candidate participates in a new interview with another manager.

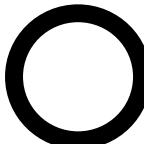
Example 2



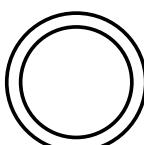
Events



- Start Event
 - There must always be at least one
 - Indicates the start of the process
 - Cannot have incoming control flows

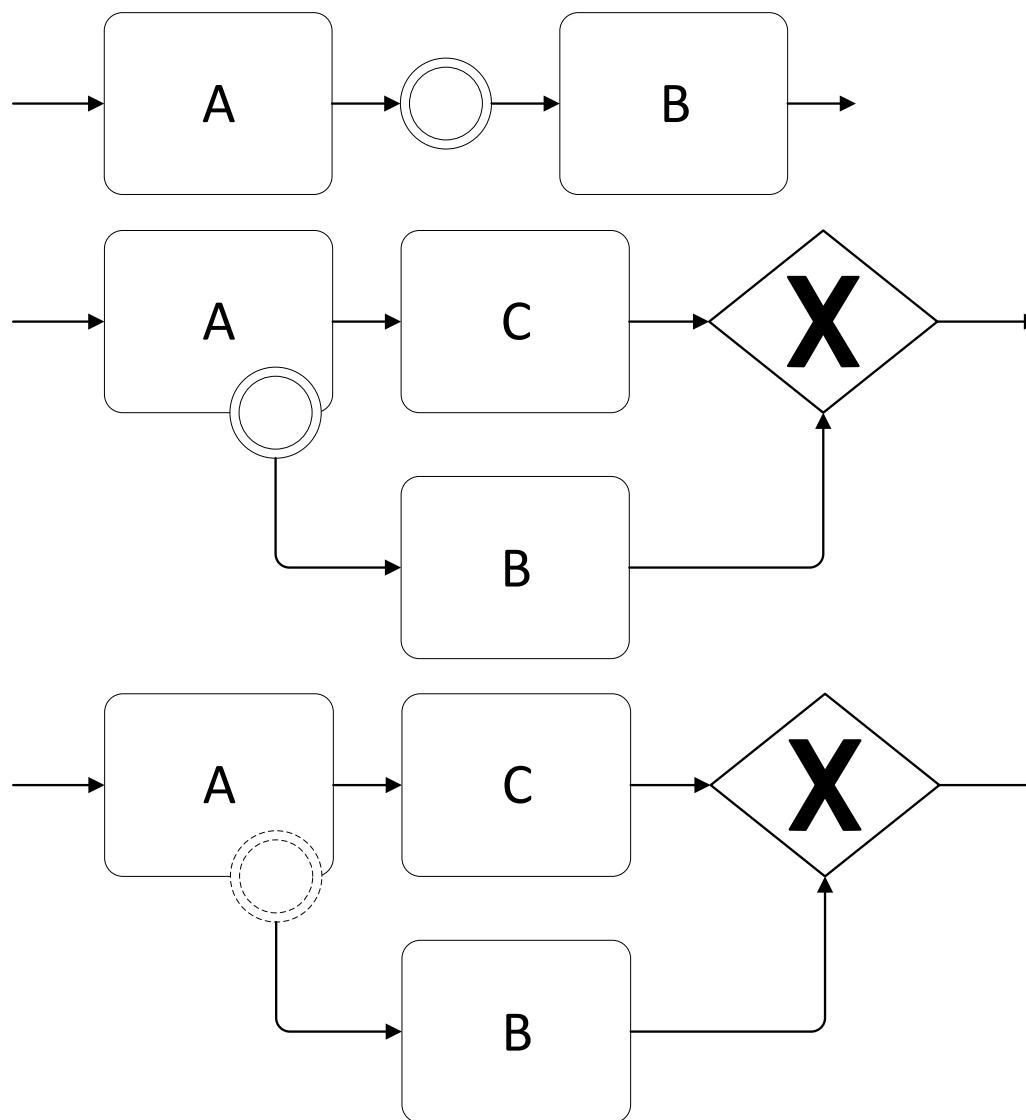


- End Event
 - There must always be at least one
 - Indicates the end of the process
 - Cannot have outgoing control flows



- Intermediate event
 - Indicates the occurrence of an event during the execution of the process
 - Can be applied to an activity or to a control flow

Events



- Applied to a control flow
 - Catch event: stops the flow until the event occurs
 - Throw event: causes the event to occur
- Applied to an activity (interrupting)
 - If the event occurs while the activity is running, interrupts the activity
- Applied to an activity (non-interrupting)
 - If the event occurs while the activity is running, triggers the outgoing flow from the event
 - The activity continues to run in parallel with the flow triggered by the event

Events



- Generic
 - Doesn't specify the event



- Message
 - Sender and recipient are known



- Signal
 - Recipient is unknown



- Timer
 - A date/time or interval

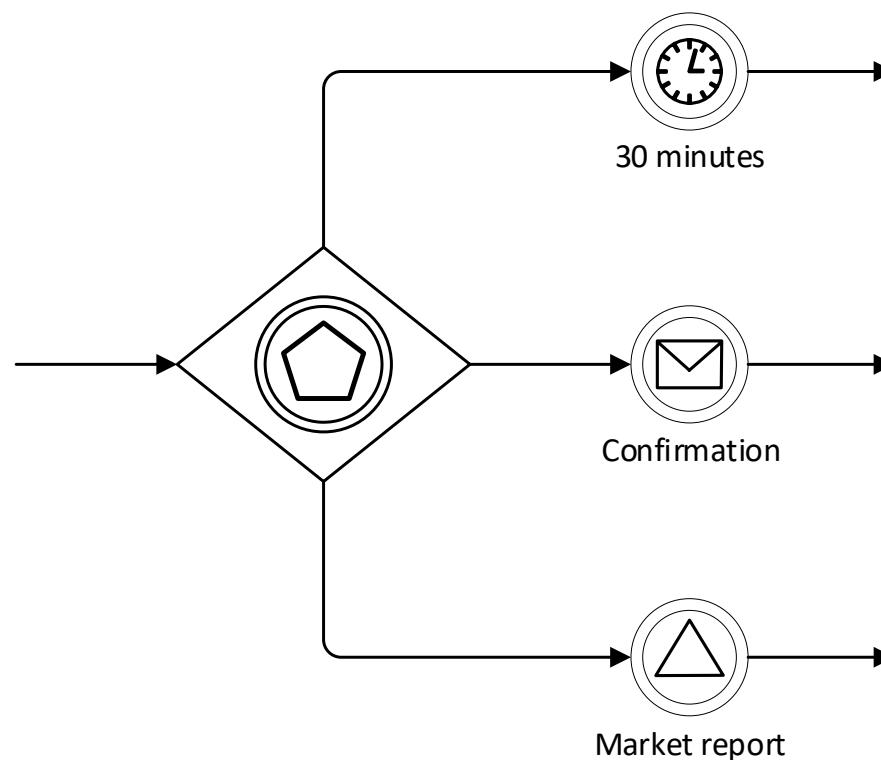


- Condition
 - One condition is valid



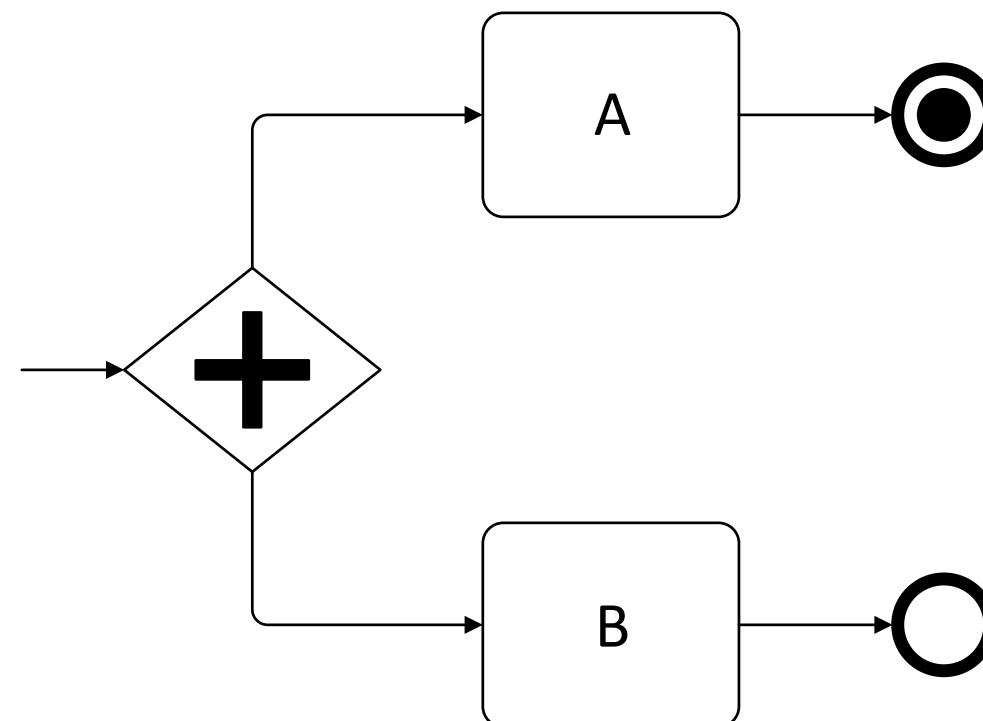
- Termination
 - Terminates the whole process

Event-based Exclusive Gateway



- Similar behavior to the Exclusive gateway
- The active branch is determined by the event
- If more than one event occurs, only the one that happened first is considered

Events

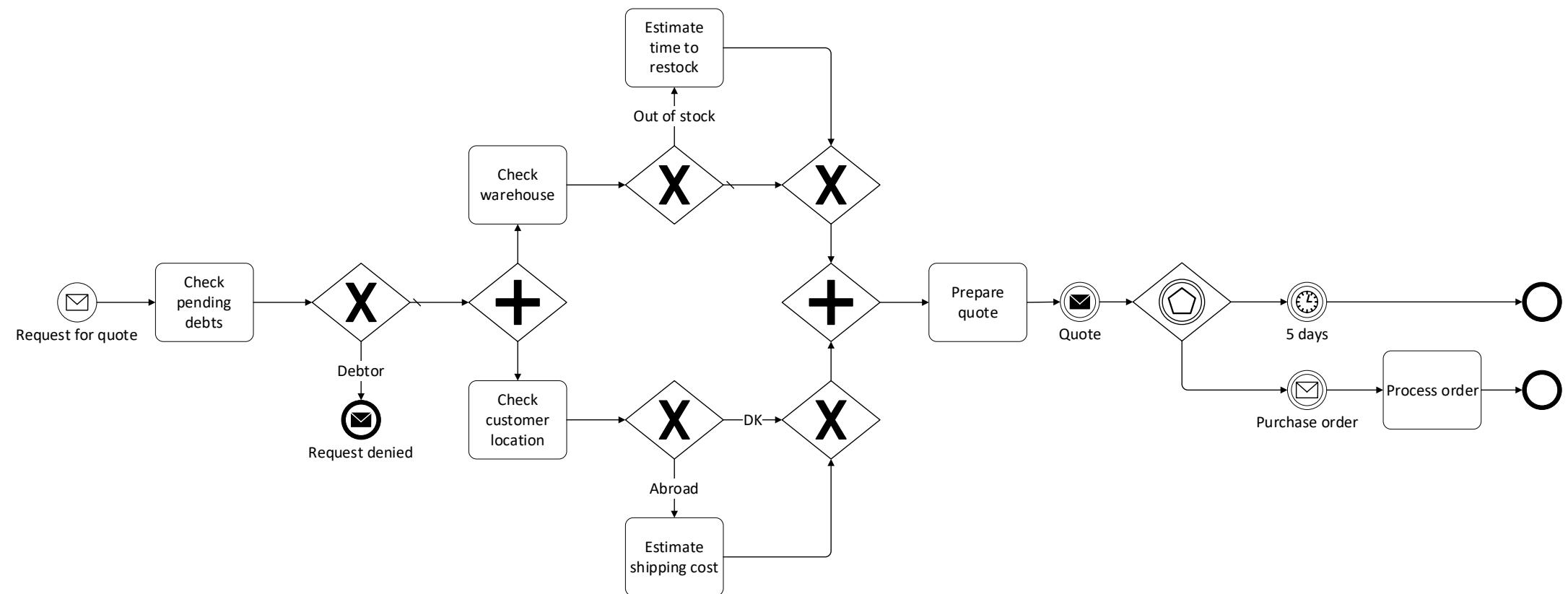


- End events generate an event
- The Terminate event terminates all active execution flows
- All running activities are terminated

Example 3

A supplier of electronic equipment receives a request for a quote from a customer. First, it checks if the customer has no pending debts. If this is the case, the request is rejected, and the customer is notified. Otherwise, the supplier checks the availability of the equipment being requested. If it is not in stock, it estimates the time to restock it. In the meantime, if the customer is located outside Denmark, the supplier also estimates the cost to ship the order. When all these steps are completed, the supplier sends the proposal to the customer and waits for a purchase order. When the order arrives, the supplier processes the order, and the process ends. If no response is received within 5 days since the quote was sent, the process ends.

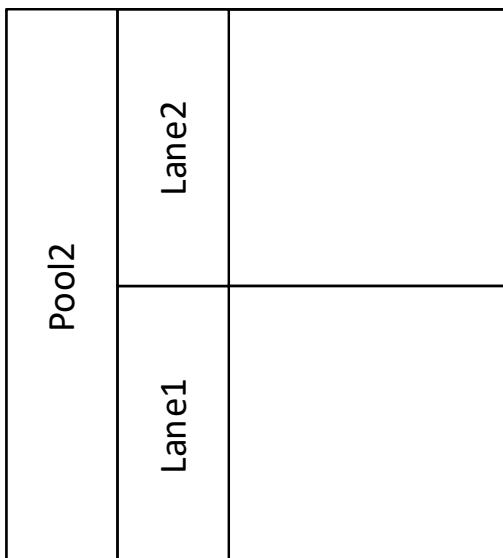
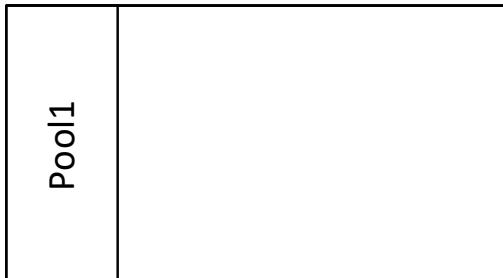
Example 3



Process collaboration

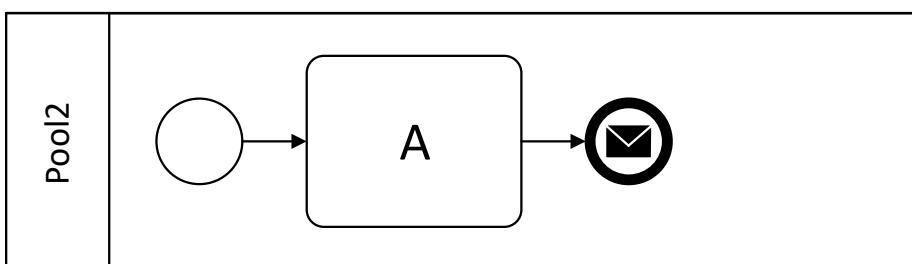
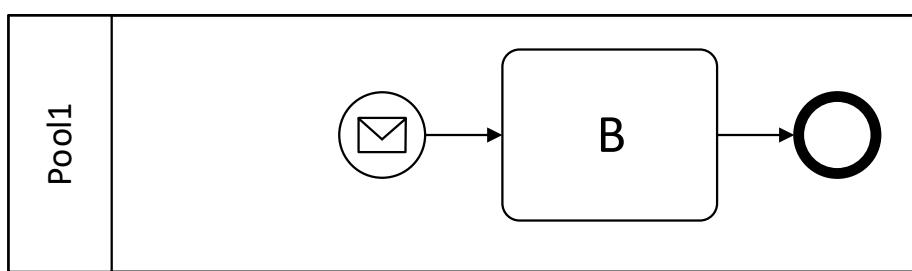
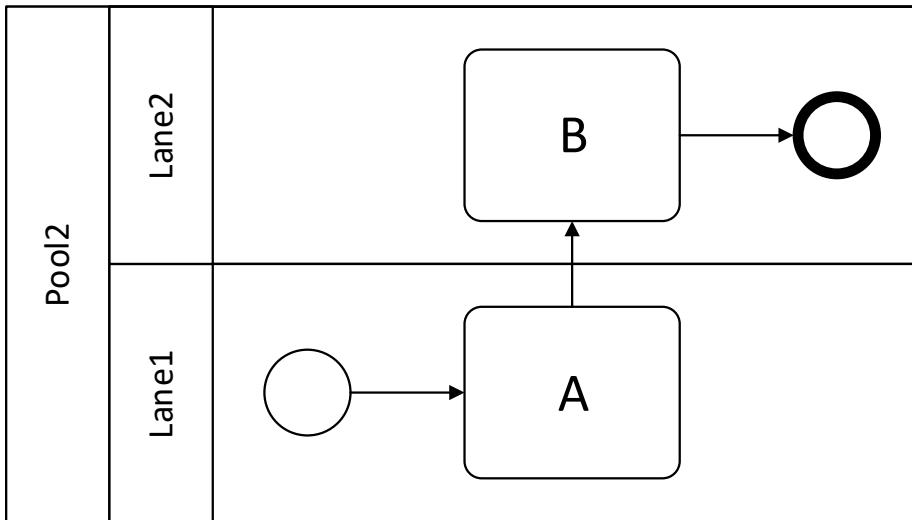
- So far, we have been working on modeling processes belonging to a single organization
 - Process Orchestration
- BPMN also allows you to model interactions between processes belonging to distinct business entities
 - Cross-process collaboration

Pools and lanes



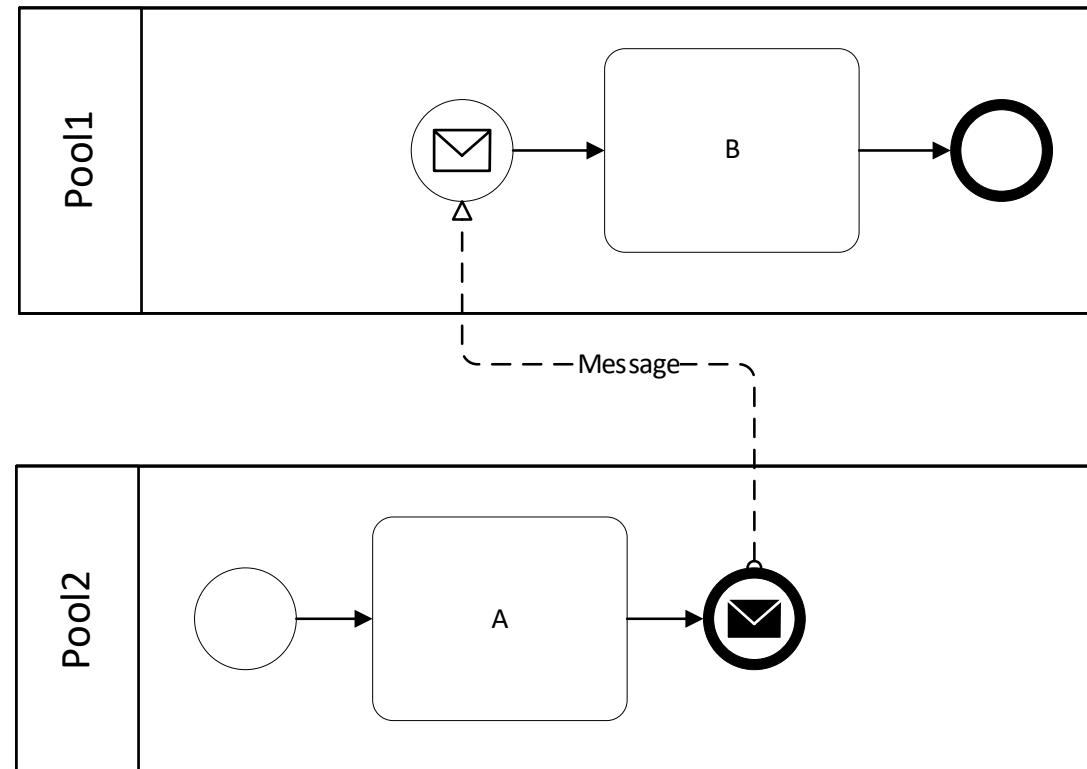
- Pool: Represents a separate business entity
 - A pool can be treated as a black-box or its processes made explicit (only if known)
- Lane: Represents a partition of a pool
 - Typically, a role or division (e.g. administration)
 - It only makes sense inside a Pool

Pools and lanes



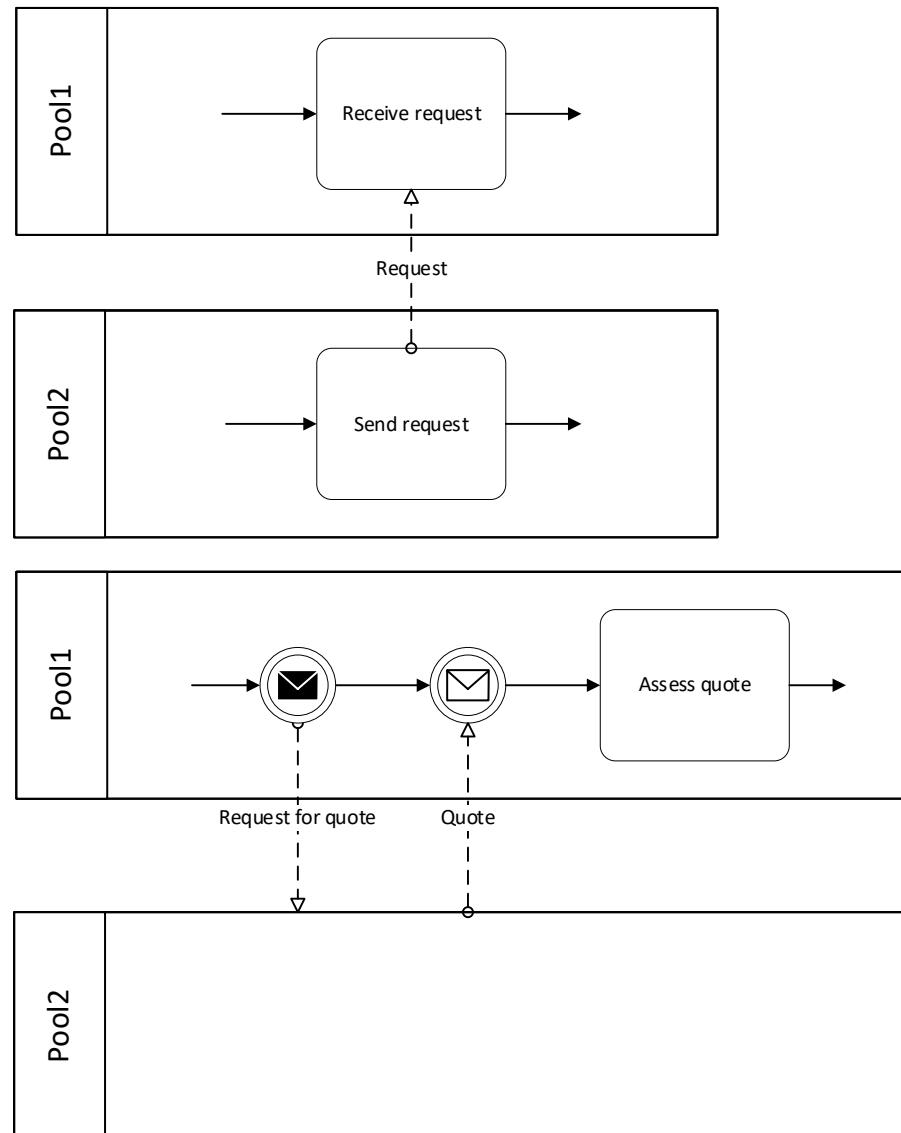
- Within a pool, coordination takes place through the control flow
- Between separate pools, communication takes place only through events
- It is **FORBIDDEN** to use the control flow outside of a pool

Pools and lanes



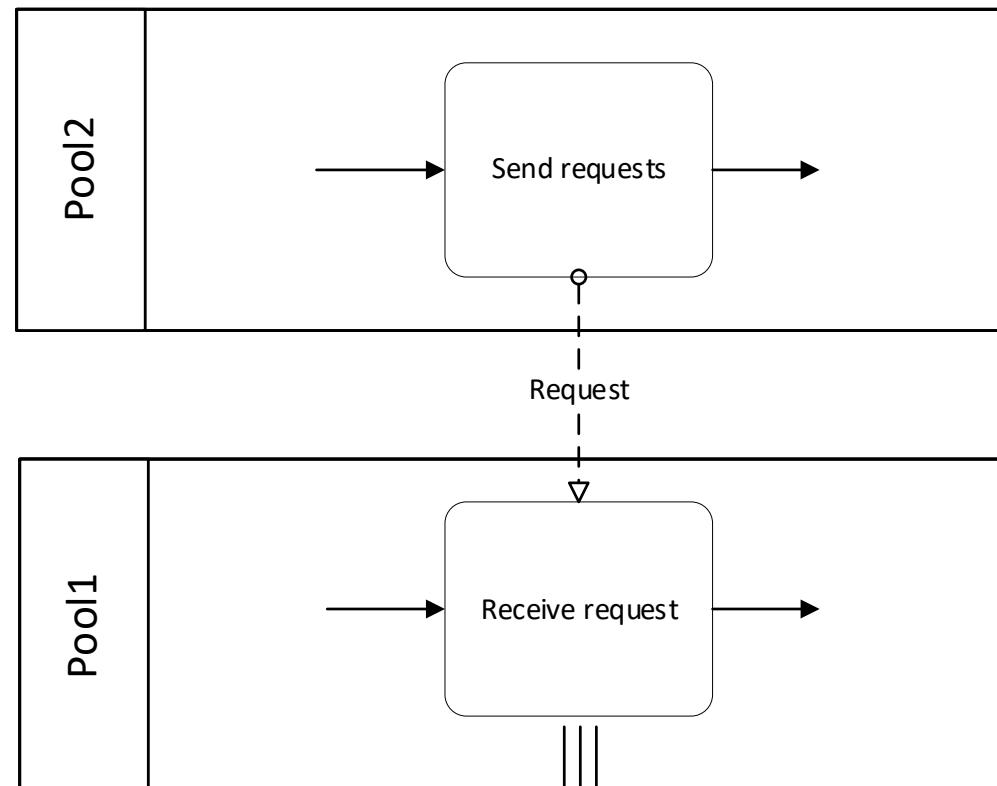
- It is possible to make explicit the coordination between processes by representing the message flow
- You can't use message flows within a pool
 - Coordination within a pool is done exclusively through the control flow

Message flow



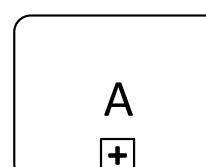
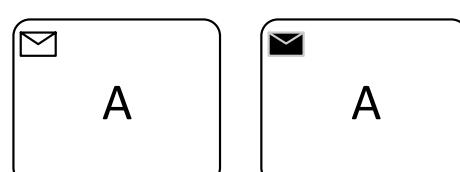
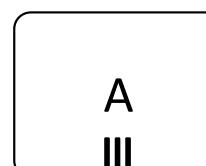
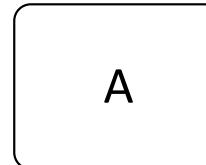
- Message flow can be used only with message events
- If the message flow is explicit, you can replace intermediate events with activities
- If the process inside a pool is not known, the message flow is connected to the pool

Multi-instance pools



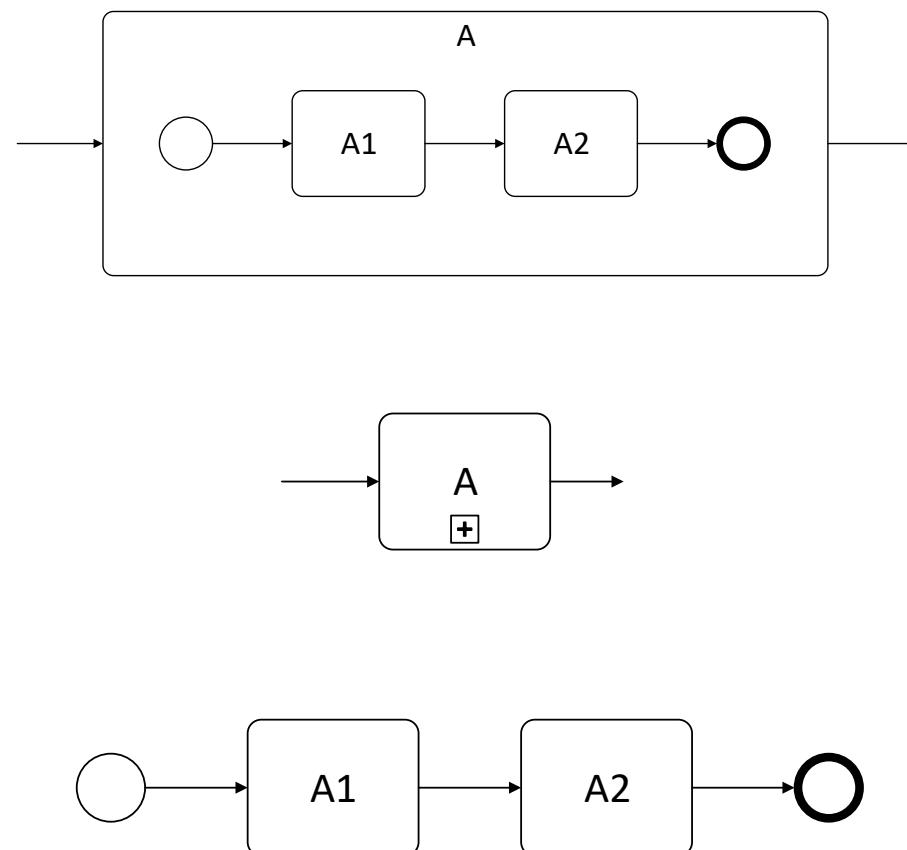
- Multiple business entities, all following the same process

Activities



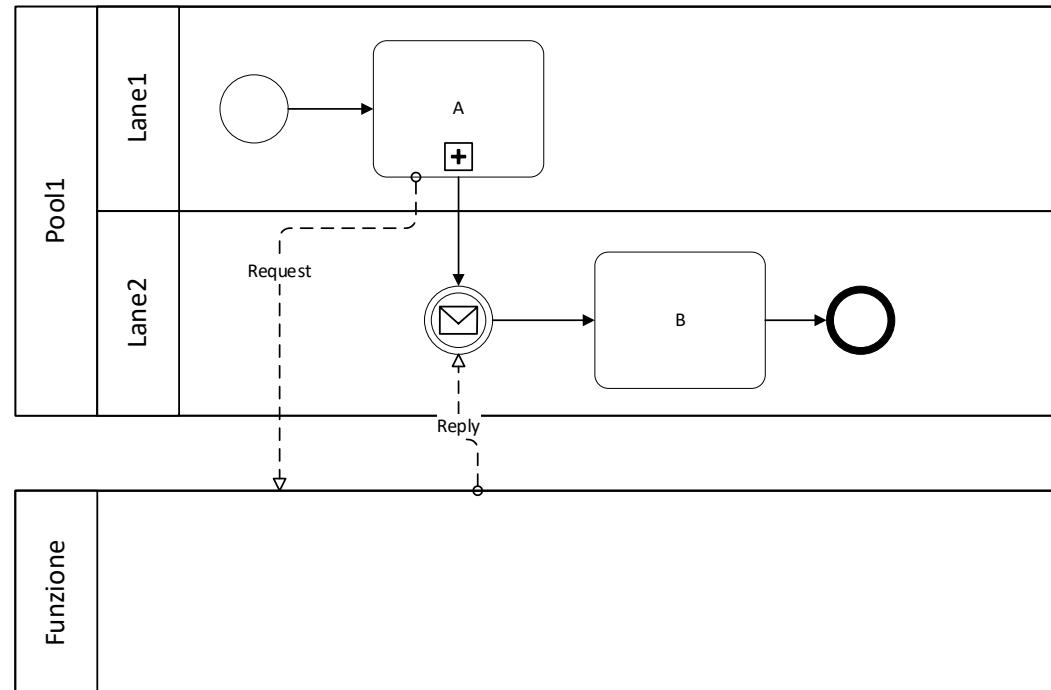
- **Task**
 - Atomic Operation Performed Only Once
- **Cyclical activity**
 - Atomic operation performed repeatedly until a condition is true
- **Parallel multi-instance activity**
 - Multiple operations of the same type carried out in parallel
- **Receive/Send Tasks**
 - Allows sending/receiving a message (similar to a Message event)
- **Sub-process**
 - A non-atomic operation that can also be represented as a process

Subprocesses

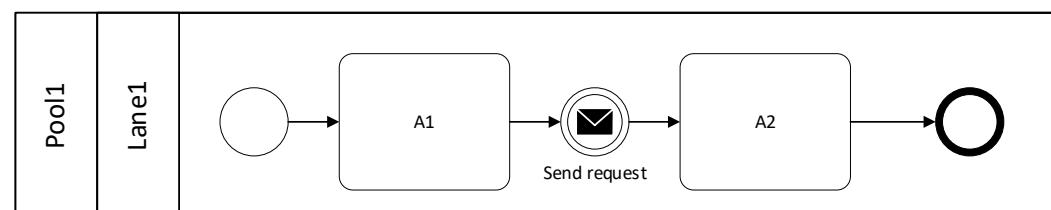


- It is possible to make a subprocess explicit by representing it directly inside the activity of the parent process where it is executed
- You can also represent it separately in another diagram

Subprocesses



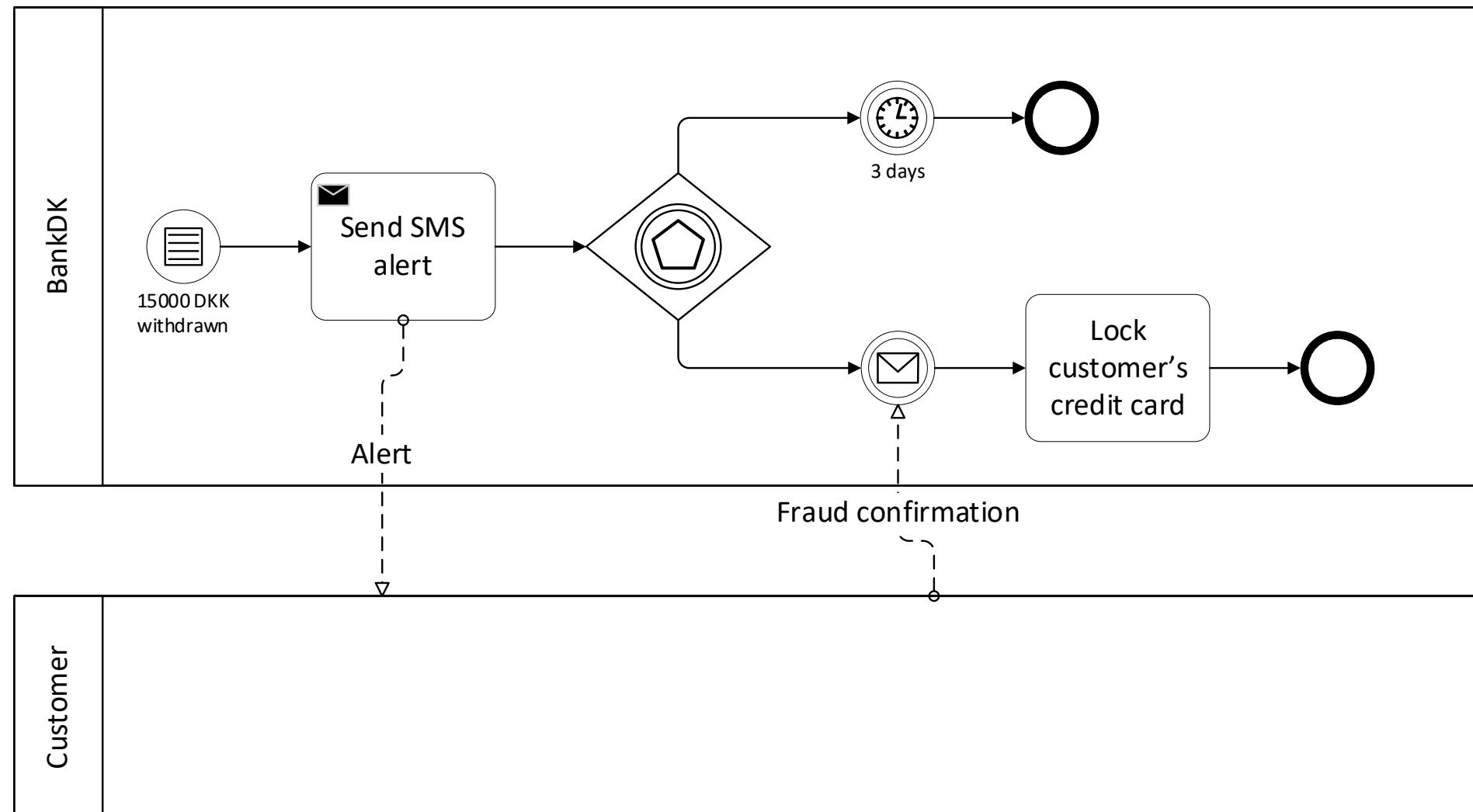
- A subprocess can contain only the pool and lane of the parent process
- Collaboration between actors cannot be represented within a sub-process



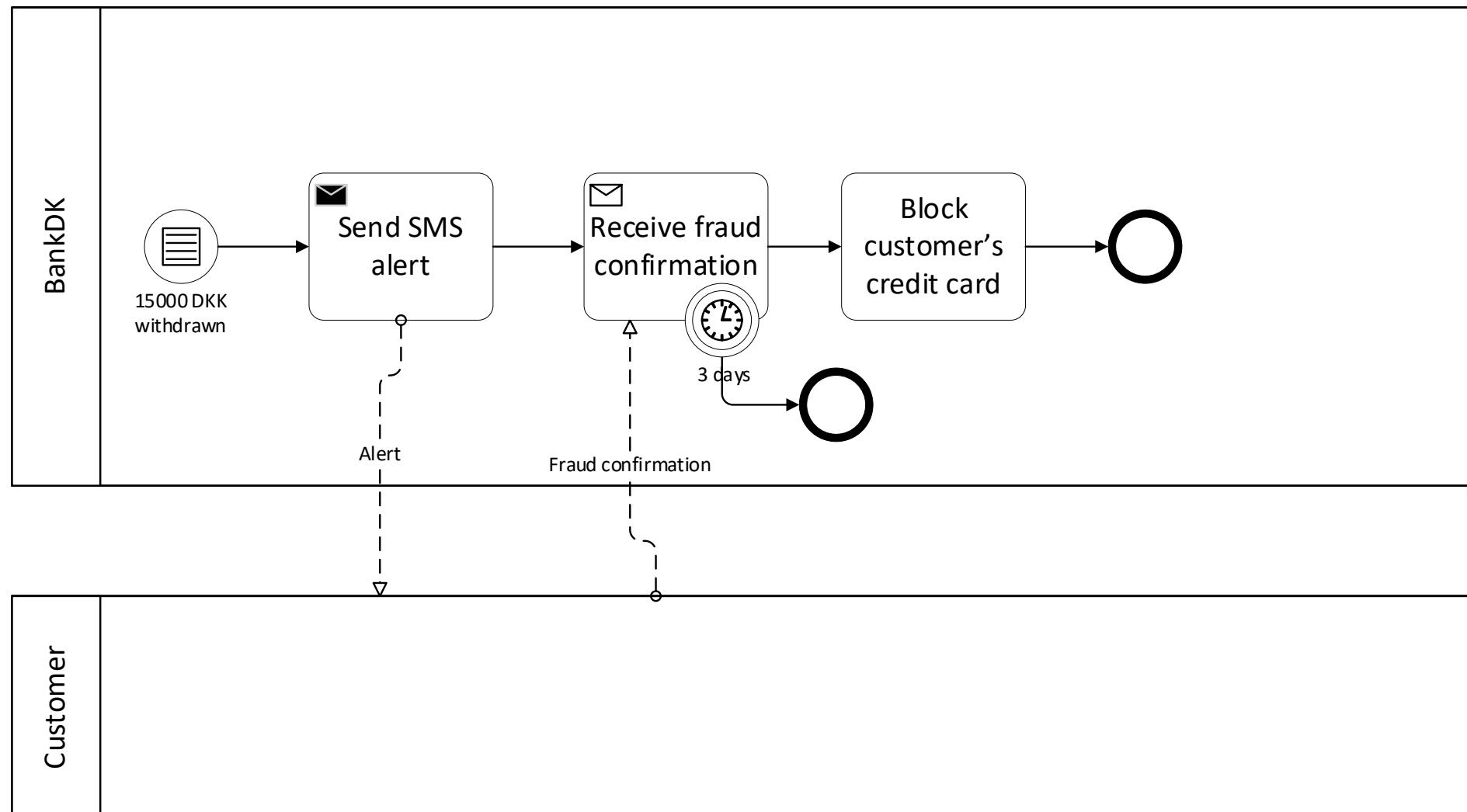
Example 4

The fraud detection process of BankDK is structured as follows. If withdrawals made with a customer's credit card at ATMs exceed 15000 DKK, BankDK sends an SMS alert to the customer's mobile phone. If the customer replies to that alert with a message containing the text FRAUD, BankDK blocks the customer's credit card. If no message is received after 3 days since the SMS was sent, the process ends.

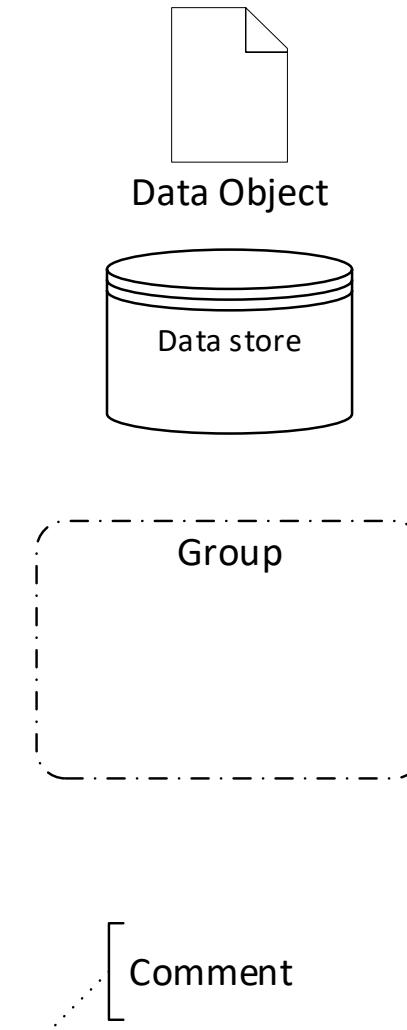
Example 4



Example 4 (alternative solution)

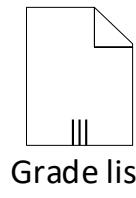


Artifacts

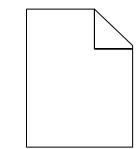


- **Data object**
 - Shows the physical or virtual objects used by the process
- **Data store**
 - Shows where the data reside
- **Group**
 - Groups related process elements
 - Has no effect on the execution of the process
- **Annotation**
 - Adds a comment to a linked element

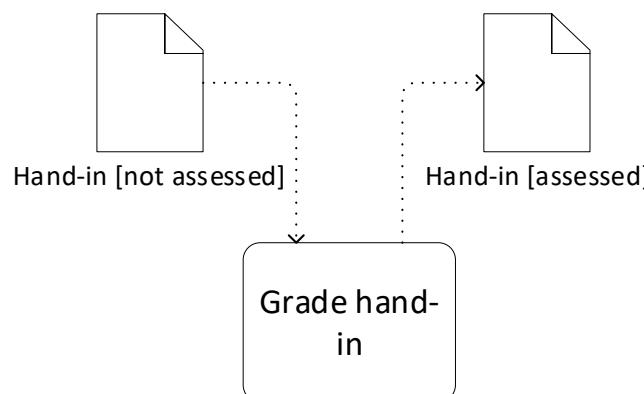
Data objects



Grade list

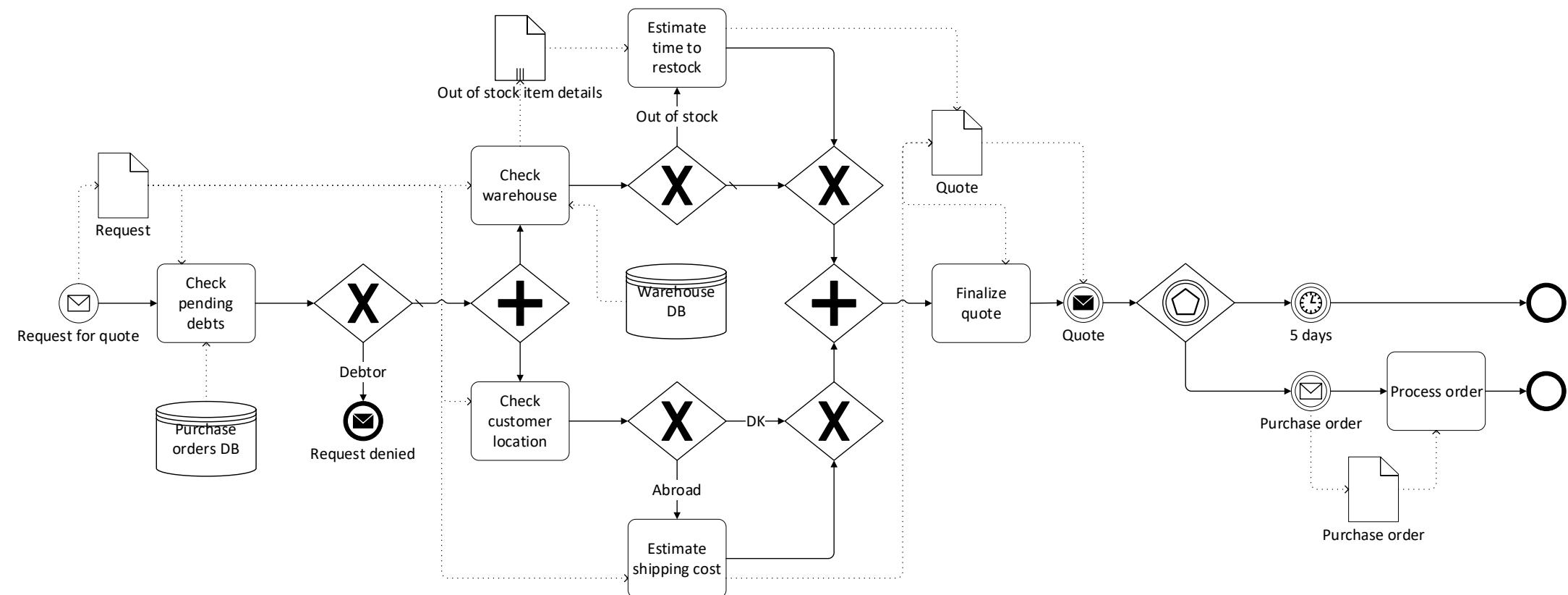


Hand-in [assessed]



- Data collection
 - Aggregates multiple data objects of the same type
- Data state
 - Specifies the state of a data object
- Data association
 - Links a data object to an activity
 - Can be input, output, or both

Example 3 enriched



Study material

- Books and articles:
 - Dumas et al. - Fundamentals of Business Process Management (2nd Edition)
 - Available at: <https://link.springer.com/book/10.1007/978-3-662-56509-4>
 - Chapter 3 (not 3.6) and 4
 - (alternatively) Weske - Business Process Management
 - Available at: <https://link.springer.com/book/10.1007/978-3-662-59432-2>
 - Chapter 4.7
- Modeling tools:
 - bpmn.io: <https://demo.bpmn.io/>
 - (alternatively) Camunda Modeler: <https://camunda.com/download/modeler/>
 - (alternatively) SAP Signavio: <https://academic.signavio.com/p/login>

Exercises

Please answer all exercises to demonstrate your skills.

Solutions will be available at 11:45

Exercise 1 – MagiMug

MagiMug is a manufacturer of promotional mugs, which are sold to companies, universities, hotels, etc. In particular, MagiMug buys white ceramic mugs from a wholesaler and screen prints on them a picture provided by the customer.

When the number of cups in stock is less than 10000 units, MagiMug's administration prepares a purchase order and sends it to its wholesaler. Once the order has been received, the wholesaler contacts a shipper. Then, while waiting for the shipper to reach its premises, the wholesaler prepares a container to be shipped to MagiMug. If, while preparing the container, part of the load gets damaged, the activity is stopped, and the extent of the damage is estimated. At the same time, the damaged load is removed from the container. Once these two activities are complete, the container preparation is repeated. Once the container is ready and the shipper has reached the wholesaler's premises, the container is given to the shipper, who attaches it to his/her truck, and delivers it to MagiMug. Once the container is received, MagiMug's warehouse workers check the integrity of the load. If everything is in order, this is reported to the shipper, who in turn notifies the wholesaler of the success of the activity, and the process ends. If, on the other hand, part of the load is damaged, the shipper takes the container back to the wholesaler and the process ends."

Exercise 2 – AIText

AIText is a publisher of printed academic textbooks and scientific journals.

When the paper rolls that AIText has in stock fall below 500 rolls, a request for a quote for 2000 rolls is prepared and sent to one of its supplier. The supplier, upon reception of the request, verifies if it has enough paper rolls in its warehouses. If the supplier does not have enough rolls, it terminates its process by informing AIText. AIText then sends the request for a quote again to another supplier. If a supplier has enough rolls available, it prepares a quote, which is sent to AIText. If, after assessing the quote, AIText is satisfied with it, AIText notifies the supplier that the quote has been accepted, the supplier registers the order, and the process ends on both sides. If AIText is not satisfied with the quote, it sends the request for a quote again to another supplier. If a supplier does not receive the notification within 10 days since the quote was sent, it ends the process.

Exercise 3 - CompGears

CompGears, a consumer electronics company, wants to model its helpdesk's technical support process.

The process begins when a customer contacts the company's helpdesk with a technical support request, communicating his/her identity. First, the helpdesk operator checks whether the customer is already present in the system and, if not, (s)he registers the customer. Then, the helpdesk operator asks the customer to provide the serial number of the product. Once this information is received, the helpdesk operator checks if that product is still under warranty. If not, the helpdesk operator ends the process by notifying the customer that the product is out of warranty. Otherwise, (s)he notifies the user that the request has been accepted and forwards it to an operator from the company's in-house technical department.

First, the technical operator asks the customer to describe the problem. Once this information is received, (s)he examines the symptoms and then looks for a possible resolution action, which is provided to the customer. The customer then verifies whether that resolution action solves his problem and communicates the outcome to the technical operator. If the resolution action is successful, the process ends on both sides. If not, the technical operator searches again for a resolution action. This is repeated until the problem is resolved. If the technical operator is busy for more than 30 minutes on a specific support request, the process ends by inviting the user to repeat the entire procedure again.

02291 System Integration

Introduction to DMN

© Giovanni Meroni

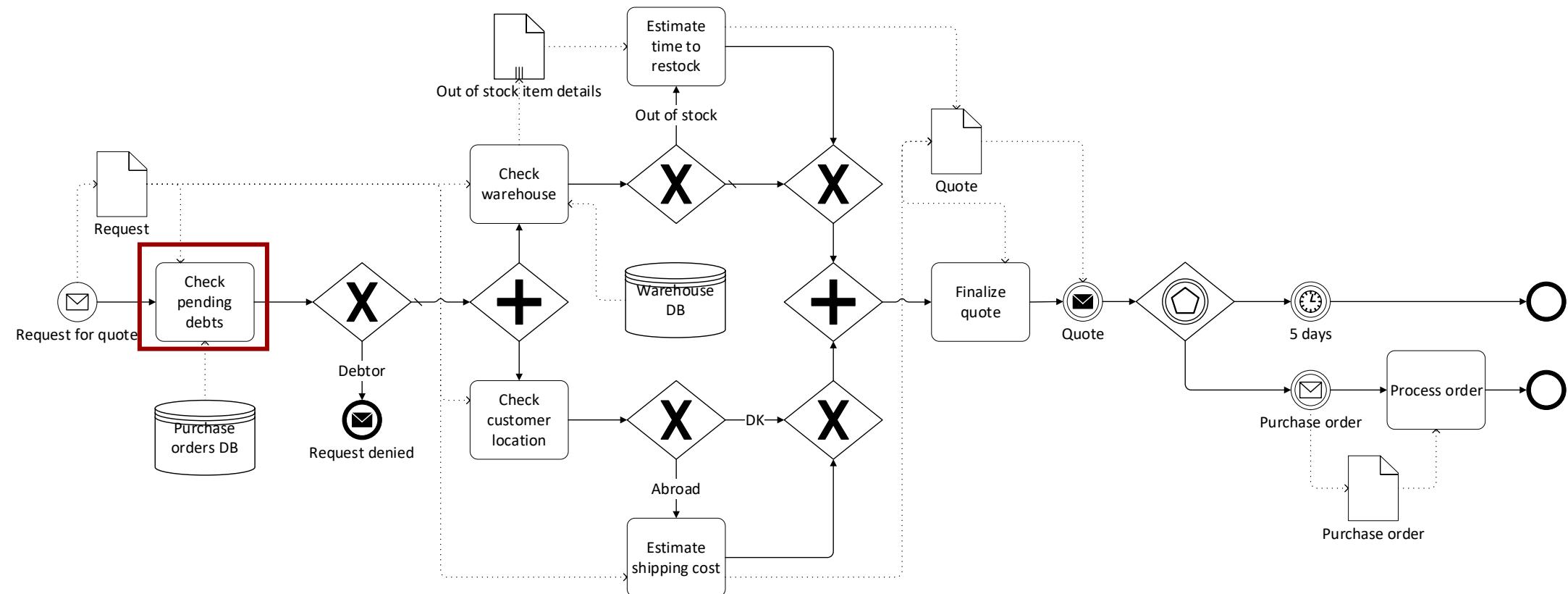
Decisions in BPMN

- In BPMN, decisions are modelled as tasks followed by gateways
- However, it is not explicitly shown:
 - What policies are used to make a decision
 - Which information is used to make a decision
 - Who is responsible for making a decision

Example 3 (from lecture 11)

A supplier of electronic equipment receives a request for a quote from a customer. First, it **checks if the customer has no pending debts**. If this is the case, the request is rejected, and the customer is notified. Otherwise, the supplier checks the availability of the equipment being requested. If it is not in stock, it estimates the time to restock it. In the meantime, if the customer is located outside Denmark, the supplier also estimates the cost to ship the order. When all these steps are completed, the supplier sends the proposal to the customer and waits for a purchase order. When the order arrives, the supplier processes the order, and the process ends. If no response is received within 5 days since the quote was sent, the process ends.

Example 3 (from lecture 11)



Example 3 (extended)

A customer is considered as a debtor based on its fidelity level (gold, silver, bronze), on the payments due, and on the number of unpaid invoices:

- Gold customers are debtors if the total payments due exceed 100000 DKK, regardless of the number of unpaid invoices.
- Silver customers are debtors if the total payments due exceed 10000 DKK or they have more than 5 unpaid invoices.
- Bronze customers are debtors if the total payments due exceed 5000 DKK or they have more than 2 unpaid invoices.

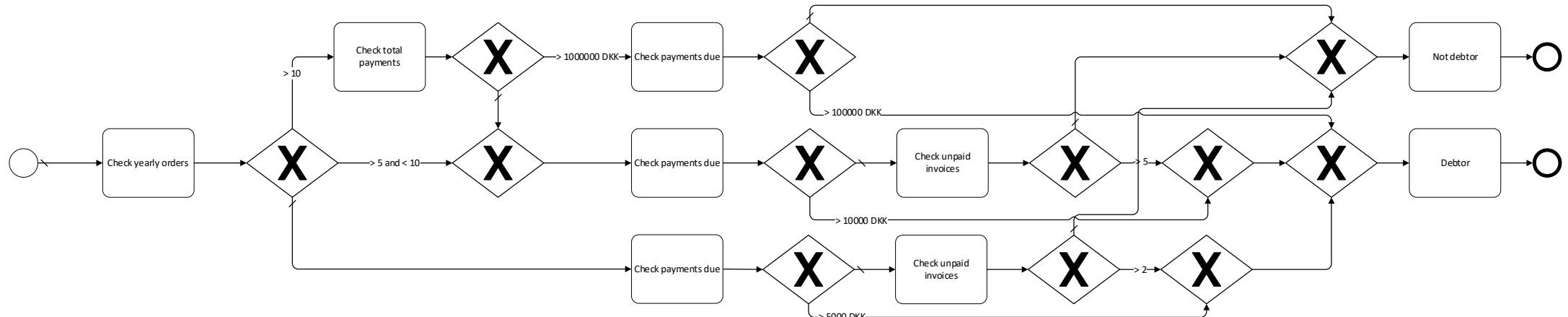
A customer earns gold fidelity level if (s)he places at least 10 orders per year amounting to 1000000 DKK or more in total.

(S)he earns silver fidelity level if (s)he places at least 5 orders per year, independently of the amount.

Otherwise, (s)he earns bronze fidelity level.

Example 3 (extended)

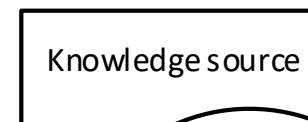
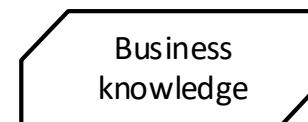
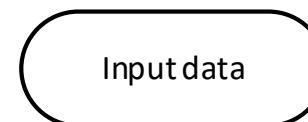
- Difficult to understand
 - Has repeated and improper activities
 - Embeds the decision logic into the process logic



Introducing DMN

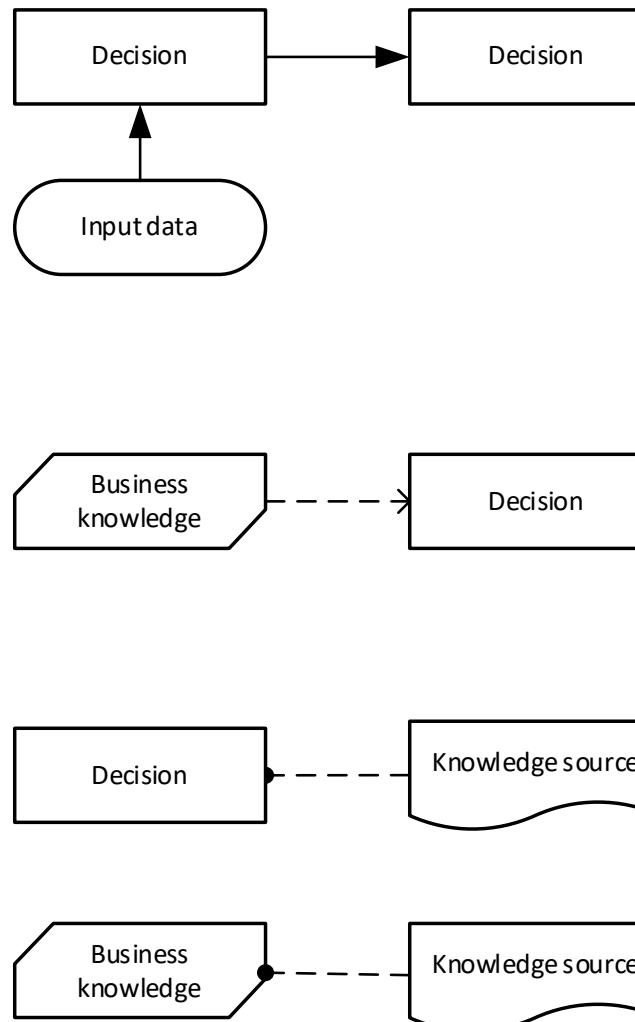
- Decision Model and Notation
- A language to model operational decisions
- Can be understood by both computer scientists and non-computer scientists (Business Analyst, Process Designer, ...)
- Complementary to BPMN

Elements in DMN



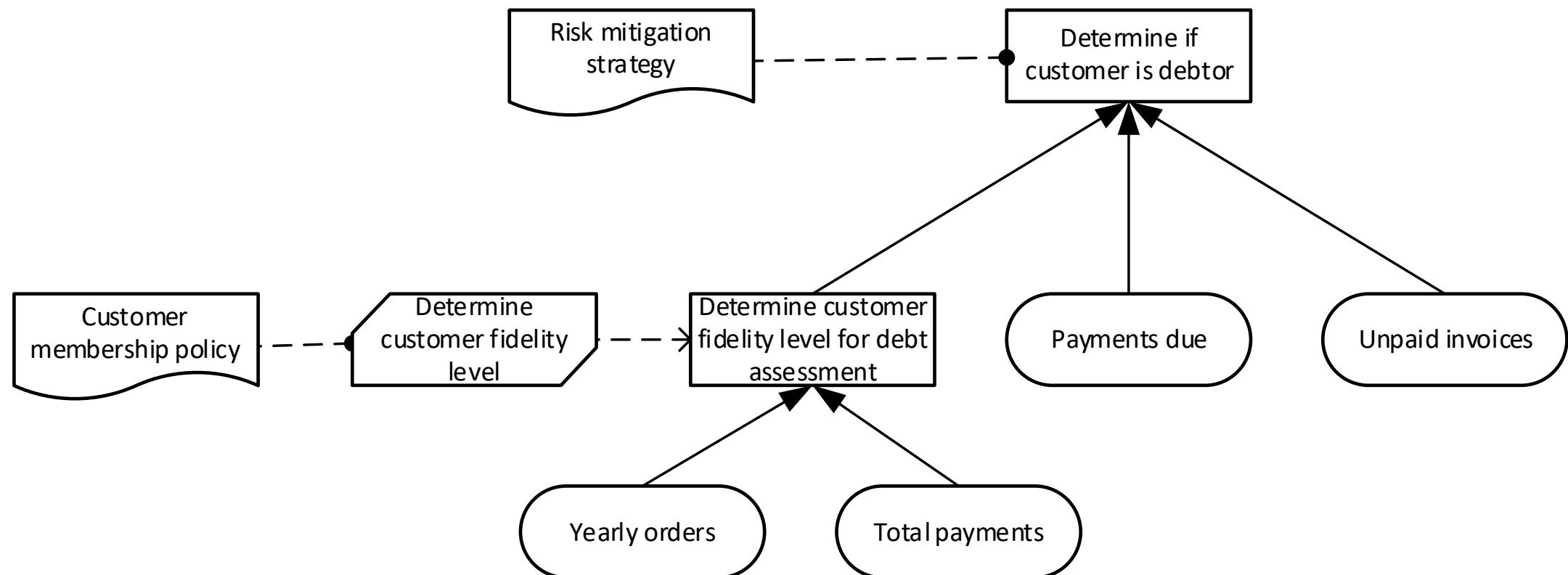
- Decision
 - Logic to make a decision
- Input data
 - Information needed to make a decision
- Business knowledge
 - Logic that can be reused in multiple decisions
- Knowledge source
 - Motivation for a specific decision logic

Elements in DMN



- **Information requirements**
 - Specify which input data is used for a decision
 - Specify which decision is used in another decision
- **Knowledge requirements**
 - Specify which business knowledge is used for a decision
- **Authority requirements**
 - Specify which knowledge source is used for a decision

Example 3 (DMN)



Decisions

- Decisions can be determined by:
 - Expressions: informal (natural language), or executable (FEEL language)
 - Invocations: external software components
 - **Decision tables**: a tabular representation of the decision rules

Decision tables in DMN

Determine if customer is debtor	Fidelity level {Gold, Silver, Bronze}	Payments due [0, 10000000]	Unpaid invoices [0,10]	Debtor {Yes, No} No > Yes
1	-	<= 5000	<= 2	No
2	Gold	<= 100000	-	No
3	Silver	<= 10000	<= 5	No
4	-	-	-	Yes

Decision tables in DMN

Determine if customer is debtor	Fidelity level {Gold, Silver, Bronze}	Payments due [0, 10000000]	Unpaid invoices [0,10]	Debtor {Yes, No} No > Yes
1	-	<= 5000	<= 2	No
2	Gold	<= 100000	-	No
3	Silver	<= 10000	<= 5	No
4	-	-	-	Yes

The current table is ambiguous!

- A silver customer with 4000 DKK due and 2 unpaid invoices would match entry 1, 3 and 4
- We need a mechanism to disambiguate such cases

Hit policies

- Determine which and how many rules are selected:
 - Single hit policies: only one rule can be selected:
 - Multi-hit policies: more than one rule can be selected, and the outcome depends on the output values of the selected rules.

Single hit policies

- Unique (U): only one rule can match each possible input combination
- Any (A): multiple rules can match an input combination, but the outcome must be the same
- Priority (P): multiple rules can match an input combination, and the outcome is determined by the ordering of output values in the output domain
- First (F): multiple rules can match an input combination, and the outcome is determined by the rule that appears first in the table

Decision tables in DMN: First hit policy

Determine if customer is debtor (F)	Fidelity level {Gold, Silver, Bronze}	Payments due [0, 10000000]	Unpaid invoices [0,10]	Debtor {Yes, No} No > Yes
1	-	<= 5000	<= 2	No
2	Gold	<= 100000	-	No
3	Silver	<= 10000	<= 5	No
4	-	-	-	Yes

Ambiguities are resolved

- A silver customer with 4000 DKK due and 2 unpaid invoices would match entry 1 before 3 and 4
 - “No” is selected

Decision tables in DMN: Priority hit policy

Determine if customer is debtor (P)	Fidelity level {Gold, Silver, Bronze}	Payments due [0, 10000000]	Unpaid invoices [0,10]	Debtor {Yes, No} No > Yes
1	-	<= 5000	<= 2	No
2	Gold	<= 100000	-	No
3	Silver	<= 10000	<= 5	No
4	-	-	-	Yes

Ambiguities are resolved

- A silver customer with 4000 DKK due and 2 unpaid invoices would match entry 1, 3 and 4
- “No” has higher priority than “Yes”
 - “No” is selected

Decision tables in DMN: Any hit policy

Determine if customer is debtor (A)	Fidelity level {Gold, Silver, Bronze}	Payments due [0, 10000000]	Unpaid invoices [0,10]	Debtor {Yes, No} No > Yes
1	-	<= 5000	<= 2	No
2	Gold	<= 100000	-	No
3	Silver	<= 10000	<= 5	No
4	-	-	-	Yes

The table violates the hit policy!

- A silver customer with 4000 DKK due and 2 unpaid invoices would match entry 1, 3 and 4
- “Yes” and “No” are different output values
 - We either have to change hit policy, or “fix” the table

Decision tables in DMN: Any hit policy (fixed)

Determine if customer is debtor (A)	Fidelity level {Gold, Silver, Bronze}	Payments due [0, 10000000]	Unpaid invoices [0,10]	Debtor {Yes, No} No > Yes
1	-	> 100000	-	Yes
2	Silver	> 10000	-	Yes
3	Silver	-	> 5	Yes
4	Bronze	> 5000	-	Yes
5	Bronze	-	> 2	Yes
6	-	<= 5000	<= 2	No
7	Gold	<= 100000	-	No
8	Silver	<= 10000	<= 5	No

Violations are resolved

- A silver customer with 4000 DKK due and 2 unpaid invoices would match entry 6 and 8, but both of them give “No” as output

Decision tables in DMN: Unique hit policy

Determine if customer is debtor (U)	Fidelity level {Gold, Silver, Bronze}	Payments due [0, 10000000]	Unpaid invoices [0,10]	Debtor {Yes, No} No > Yes
1	-	> 100000	-	Yes
2	Silver	> 10000	-	Yes
3	Silver	-	> 5	Yes
4	Bronze	> 5000	-	Yes
5	Bronze	-	> 2	Yes
6	-	<= 5000	<= 2	No
7	Gold	<= 100000	-	No
8	Silver	<= 10000	<= 5	No

The table violates the hit policy!

- A silver customer with 4000 DKK due and 2 unpaid invoices would match entry 6 and 8

Decision tables in DMN: Unique hit policy (fixed)

Determine if customer is debtor (U)	Fidelity level {Gold, Silver, Bronze}	Payments due [0, 10000000]	Unpaid invoices [0,10]	Debtor {Yes, No} No > Yes
1	Gold	> 100000	-	Yes
2	Silver	> 10000	> 5	Yes
3	Silver	<= 10000	> 5	Yes
4	Silver	> 10000	<= 5	Yes
5	Bronze	> 5000	> 2	Yes
6	Bronze	<= 5000	> 2	Yes
7	Bronze	> 5000	<= 2	Yes
8	Gold	<= 100000	-	No
9	Silver	<= 10000	<= 5	No
10	Bronze	<= 5000	<= 2	No

Multi-hit policies

- Output order (O): the outcome is a list ordered by the output value
- Rule order (R): the outcome is a list ordered by the rule number
- Collect (C): the outcome is an unordered set
 - (C+) sums the values
 - (C#) counts the values
 - (C<) retrieves the minimum value
 - (C>) retrieves the maximum value

Verification in DMN

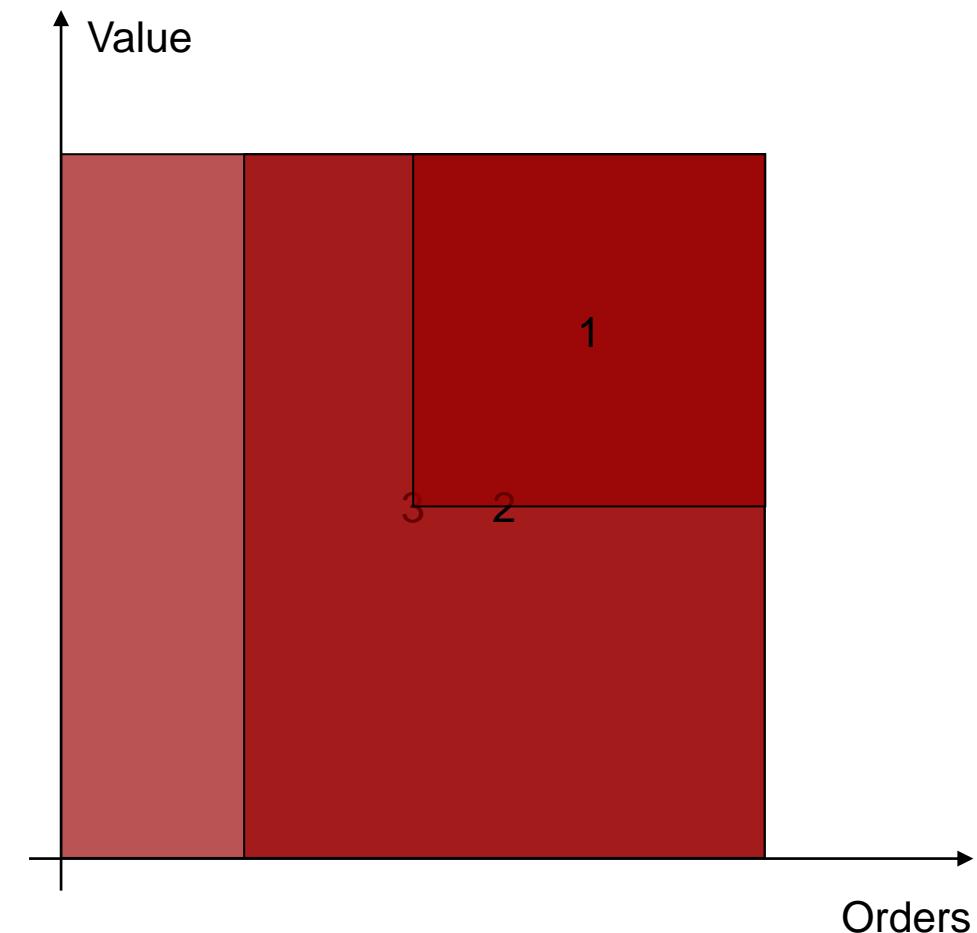
- We can see every rule as an iso-oriented hyper-rectangle in an N-dimensional space
 - N is the number of columns
- We can then look for overlapping uncovered areas
 - Overlapping areas indicate potentially conflicting rules
 - Whether a conflict occurs depends on the output value and hit policy
 - Uncovered areas indicate undefined behaviours

Calvanese, Diego, et al. "Semantics and analysis of DMN decision tables." *Business Process Management: 14th International Conference, BPM 2016, Rio de Janeiro, Brazil, September 18-22, 2016. Proceedings 14*. Springer International Publishing, 2016.

Verification in DMN

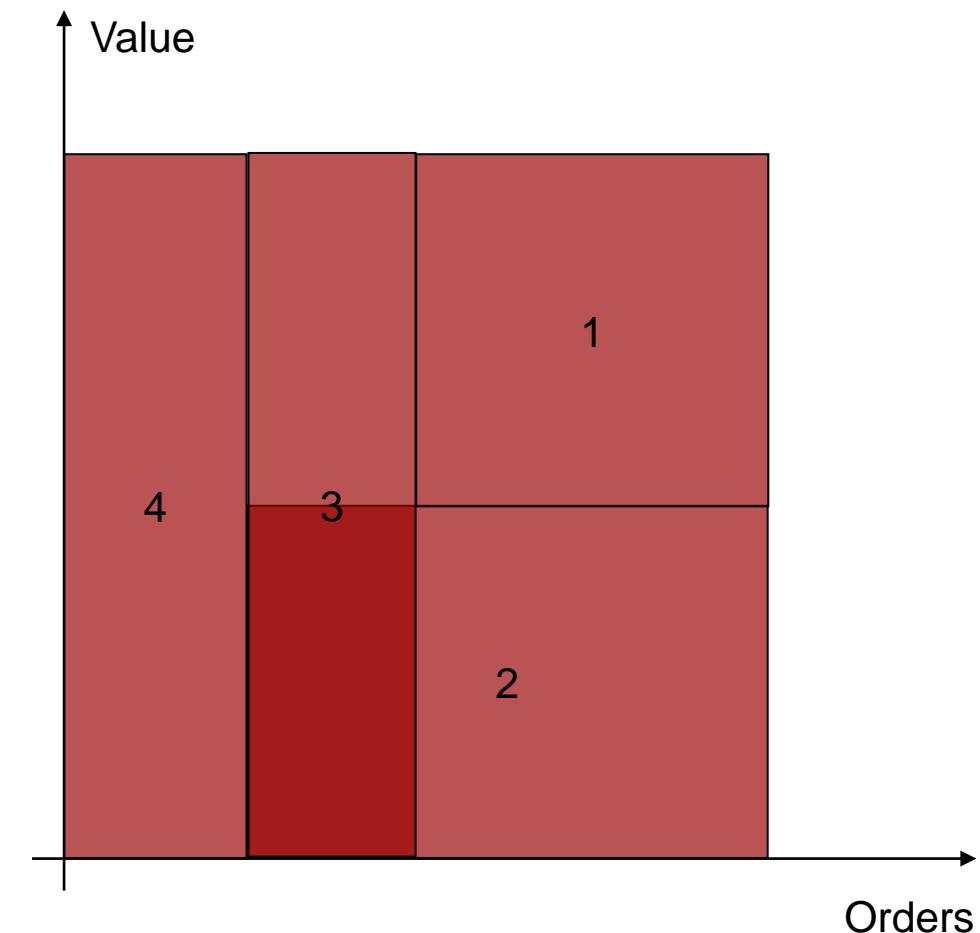
Determine customer fidelity (U)	Total orders [0,20]	Total orders value [1,2000000]	Fidelity Level {gold, silver, bronze} Gold>Silver>Bronze
1	≥ 10	≥ 1000000	Gold
2	≥ 5	-	Silver
3	-	-	Bronze

X



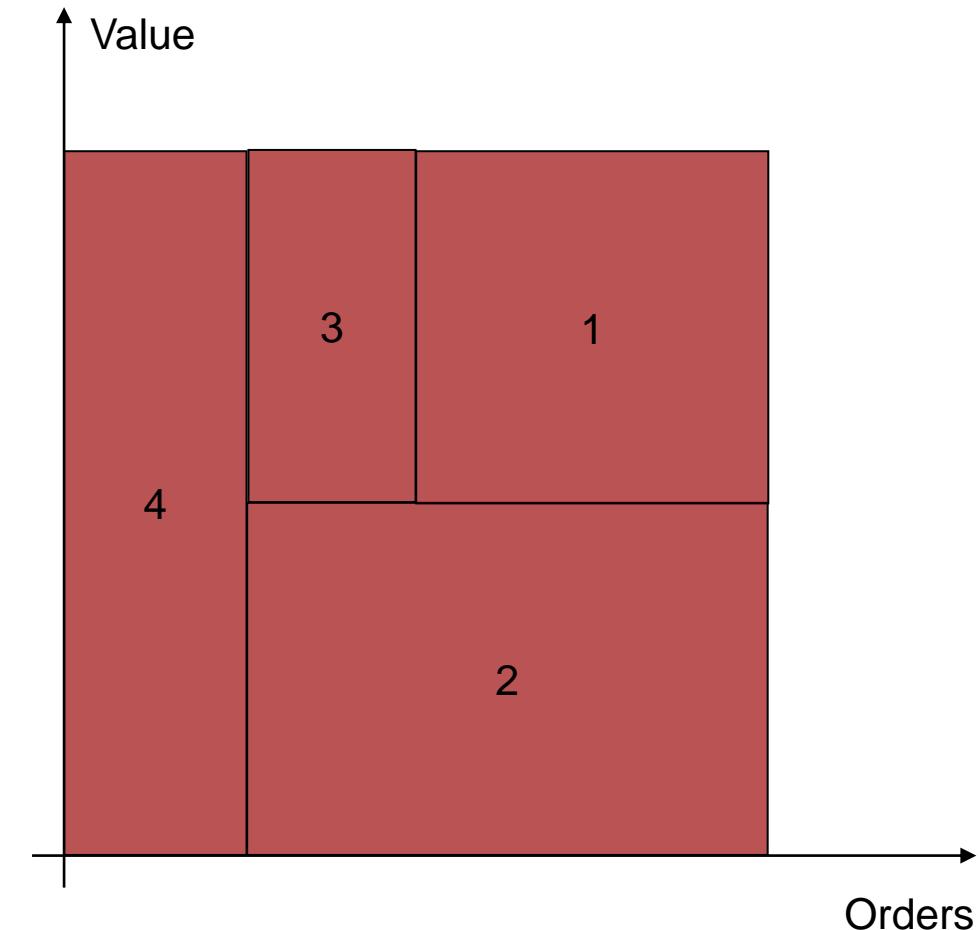
Verification in DMN

Determine customer fidelity (A)	Total orders [0,20]	Total orders value [1,2000000]	Fidelity Level {gold, silver, bronze} Gold>Silver>Bronze
1	≥ 10	≥ 1000000	Gold
2	≥ 5	< 1000000	Silver
3	[5,9]	-	Silver
4	< 5	-	Bronze



Verification in DMN

Determine customer fidelity (U)	Total orders [0,20]	Total orders value [1,2000000]	Fidelity Level {gold, silver, bronze} Gold>Silver>Bronze
1	≥ 10	≥ 1000000	Gold
2	≥ 5	< 1000000	Silver
3	[5,9]	> 1000000	Silver
4	< 5	-	Bronze



Study material

- Books and articles:
 - Weske - Business Process Management
 - Available at: <https://link.springer.com/book/10.1007/978-3-662-59432-2>
 - Chapter 5
 - Calvanese et al.: Semantics and Analysis of DMN Decision Tables, BPM 2016 Conference Proceedings, 217-233 (2016)
 - Available at: https://doi.org/10.1007/978-3-319-45348-4_13
- Modeling tools:
 - Camunda DMN-Simulator: <https://consulting.camunda.com/dmn-simulator/>
 - bpmn.io: <https://demo.bpmn.io/>
 - (alternatively) SAP Signavio: <https://academic.signavio.com/p/login>

Exercises

Please answer all exercises to demonstrate your skills.

Solutions will be available at 11:45

Exercise 1

The decision task “Estimate shipping cost” of Example 3 is organized as it follows:

- If the customer is in EMEA, the weight of the package is lower than 10 Kg, and the volume of the package is smaller than 10 cm³, then the cost amounts to 100 DKK. Otherwise, the cost is 300 DKK.
- If the customer is in America or APAC, the weight of the package is lower than 5 Kg, and the volume of the package is smaller than 7 cm³, then the cost amounts to 200 DKK. If the weight of the package is greater than 20 Kg, and the volume of the package is bigger than 30 cm³, then the cost amounts to 2000 DKK. Otherwise, the cost amounts to 1000 DKK.

Implement this task with DMN:

- Draw the DMN diagram of this task
- Implement the decision with a decision table that uses the First (F) hit policy.
- Modify that decision table to use the Any (A) hit policy instead.
- Modify that decision table to use the Unique (U) hit policy instead.

Exercise 2

- Given the following decision table:

Determine discount (?)	Fidelity level {gold, silver, bronze}	Items bought [0,100]	Total value [0,100000]	Discount [0,30] 30>0
1	Gold	> 2	-	30
2	Gold	-	-	10
3	Silver	-	> 10000	20
4	Silver	> 2	-	10
5	Bronze	> 2	> 1000	10

- Which single hit policies would not cause any violation?
- Would all the previously identified hit policies lead to the same outcome?
- Is the table complete?

Formalising and animating multiple instances in BPMN collaborations

Lorenzo Rossi

University of Camerino, Italy



Guest Lecture of System Integration
Technical University of Denmark, April 24th, 2024

GOAL OF THE LECTURE

In this lecture you will see:

- a formal **operational semantics** of BPMN collaboration models including multiple instances, data and message exchange
- an **animator tool** implementing the semantics and providing debugging features

Introduction

MOTIVATION

» BPMN

- OMG **BPMN** notation is widely applied in **industry** and academia
- **BPMN collaboration diagrams** are an effective way to model **collaborative scenarios**
- In **collaborative systems**, multiple participants cooperate and share information via message exchange to reach a shared goal

Interactions are first-class citizens in BPMN collaborations

MOTIVATION

» CAKE DELIVERY SCENARIO

Example scenario:

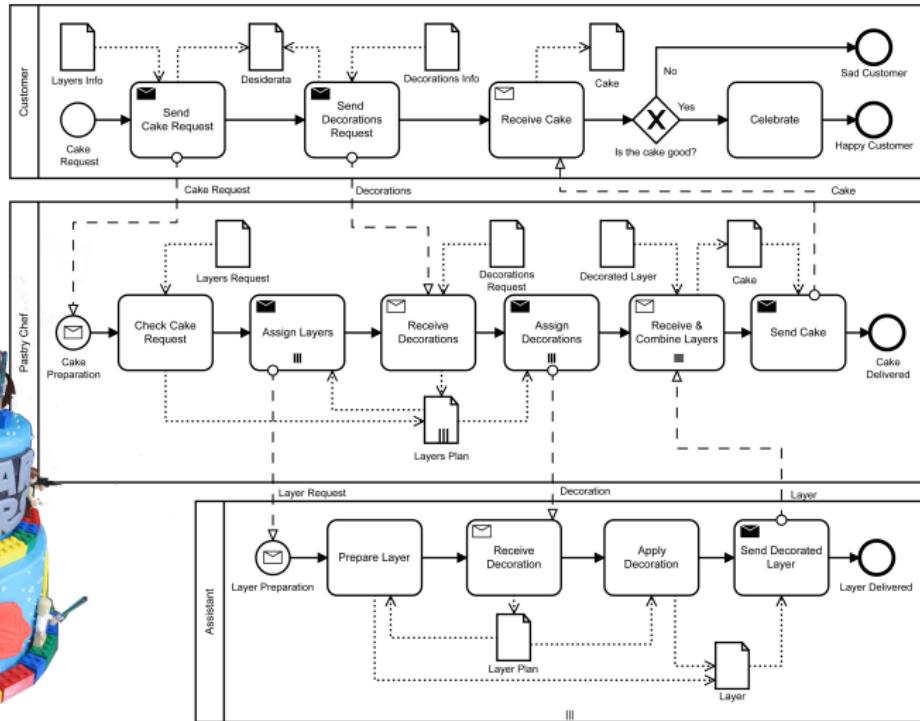
A **customer** requests a layered cake, choosing the layers' color and decorations, to a **pastry chef** that assigns a layer with the corresponding characteristic to one of his **assistants**. Once each layer is ready, the pastry chef assembles the cake and delivers it to the customer.

Details:

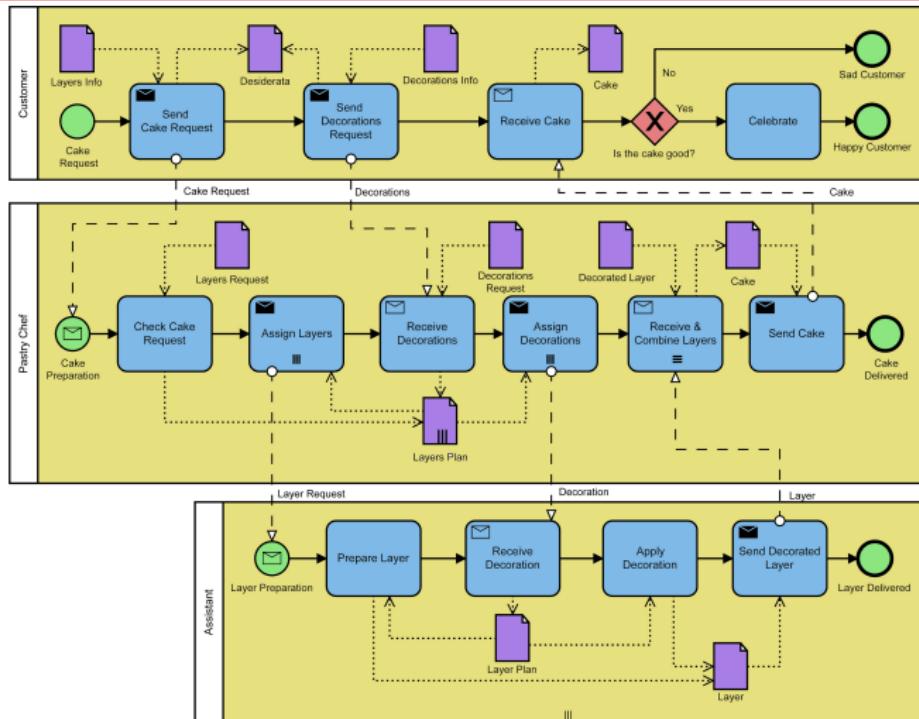
- **Cake layers:** bottom = *brown*, middle = *pink*, top = *blue*
- **Cake decorations:** bottom = *brown*, middle = *pink*, top = *blue*
- **Assistants:** 3

<https://bitbucket.org/proslabteam/mida/src/master/assets/examples/Cake/Cake.bpmn>

MOTIVATION » CAKE DELIVERY SCENARIO



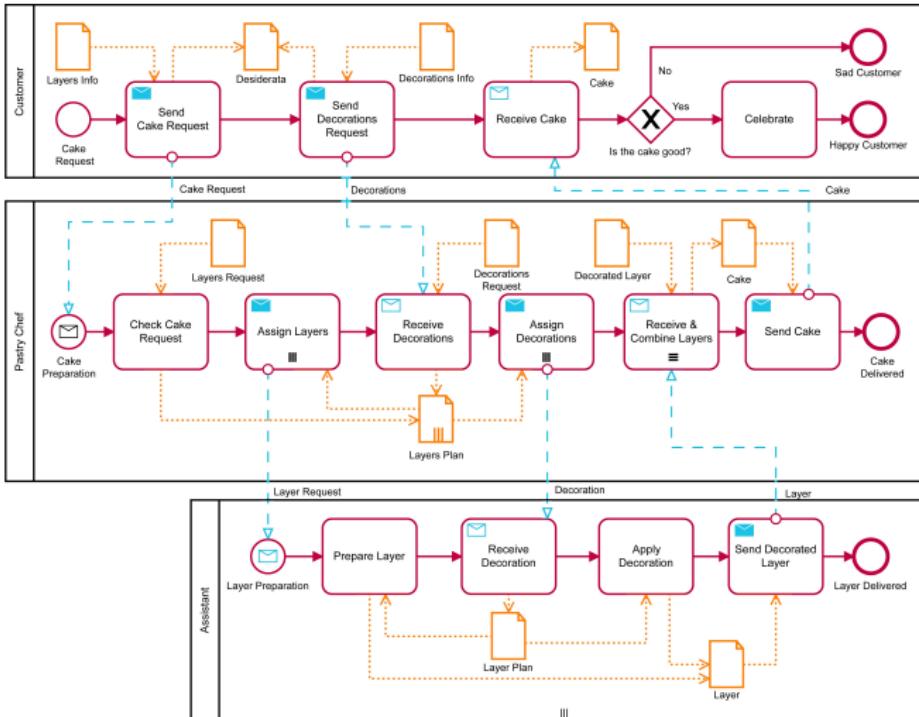
MOTIVATION » CAKE DELIVERY SCENARIO



BPMN elements can represent participants, activities, events, gateways, and data elements

MOTIVATION

» CAKE DELIVERY SCENARIO



Modeling multi-instance collaborations and **understanding** the interplay among **control**, **message** and **data** flow can be error-prone tasks

MOTIVATION

There can be issues concerning

- bad data handling
- malformed, missing or unexpected messages
- message correlation

and **possible misinterpretations of the execution semantics**



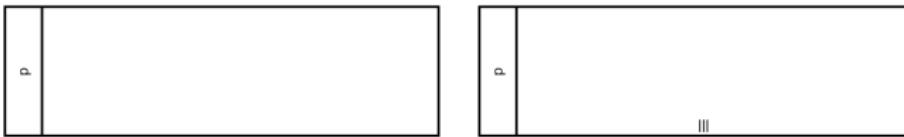
OMG does not provide a formal semantics for BPMN notation

INFORMAL SEMANTICS

» MULTIPLE INSTANCES

BPMN semantics is defined in a semi-formal in the standard

<https://www.omg.org/spec/BPMN/2.0/>



Pools and MI pools wrap processes (and all the related information) belonging to different organizations/entities

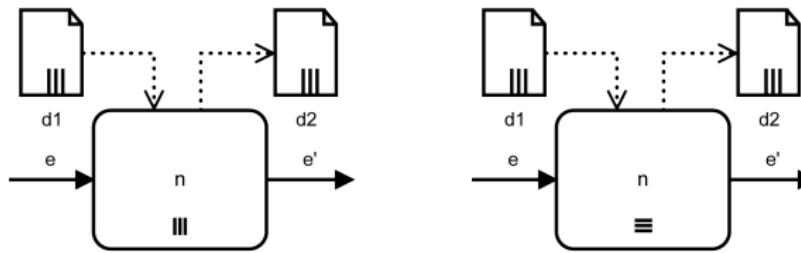
- Pools allow the execution of **one process instance at a time**
- MI pools specify a *min* and a *max* number of **instances that can be executed concurrently**

INFORMAL SEMANTICS

» MULTIPLE INSTANCES

BPMN semantics is defined in a semi-formal in the standard

<https://www.omg.org/spec/BPMN/2.0/>



MI tasks (parallel and sequential) execute the same activity a defined number of times (*loop cardinality*) unless the *completion condition* becomes true

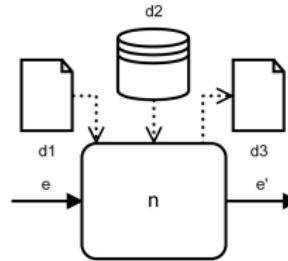
MI tasks can read/write **data object collections** i.e, collection of the same kind of data object

INFORMAL SEMANTICS

» DATA HANDLING

BPMN semantics is defined in a semi-formal in the standard

<https://www.omg.org/spec/BPMN/2.0/>



Tasks can manipulate (read/write) data stored in data objects and data stores

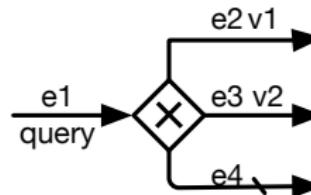
- Data objects are accessible by the process **instance** to which they belong
- Data stores are accessible by any process and any instance

INFORMAL SEMANTICS

» DATA HANDLING

BPMN semantics is defined in a semi-formal in the standard

<https://www.omg.org/spec/BPMN/2.0/>



XOR split gateways drive the control flow according to data conditions

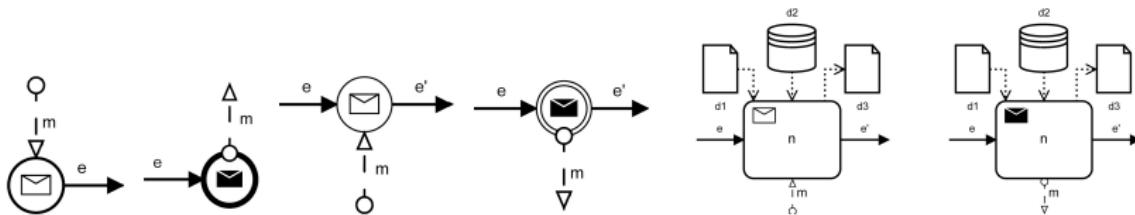
- One out of the sequence flows with a condition evaluated as *true* is non-deterministically chosen
- Default sequence flow is taken if no condition is true

INFORMAL SEMANTICS

»MESSAGE EXCHANGE

BPMN semantics is defined in a semi-formal in the standard

<https://www.omg.org/spec/BPMN/2.0/>



Send events/tasks, once triggered, deliver data via message flows. Receive events/tasks consume the message

- sender and receiver instances are correlated via **pattern-matching**
- Receive events/tasks **blocks the control flow** until there is a message to consume

LITERATURE

Several formalisations, given via **mapping** or **direct semantics**, have been proposed, mainly focusing on the **control flow perspective**

A formal semantics that considers all together

- multiple instances
- message exchanges and their correlation
- data objects
- control features, e.g. data-based decision gateways

is missing

Formal Semantics

FORMAL SEMANTICS

We provide an **operational semantics** of BPMN collaboration models including

- **multiple instances participants**, while taking into account the
- **data perspective**, considering both data objects and data-based decision gateways

The operational semantics is based on three pillars:

- a **textual representation** of the collaboration model
- a **stateful description** of the execution state of the collaboration
- **operational rules** allowing to construct the corresponding LTS

FORMAL SEMANTICS

To simplify the formal treatment we resort to a **textual representation**

$C ::= \text{pool}(p, P) \mid \text{miPool}(p, P, max) \mid C_1 \parallel C_2$

$P ::= \text{start}(e, e') \mid \text{startRcv}(m : \tilde{t}, e) \mid \text{end}(e) \mid \text{endSnd}(e, m : \tilde{exp}) \mid \text{terminate}(e)$

$\quad \mid \text{interRcv}(e, m : \tilde{t}, e) \mid \text{interSnd}(e, m : \tilde{exp}, e)$

$\quad \mid \text{andSplit}(e, E) \mid \text{xorSplit}(e, G) \mid \text{andJoin}(E, e) \mid \text{xorJoin}(E, e)$

$\quad \mid \text{eventBased}(e, (m_1 : \tilde{t}_1, e_1), \dots, (m_h : \tilde{t}_h, e_h))$

$\quad \mid \text{mipTask}(e, exp, T, c, exp', e') \mid \text{misTask}(e, exp, T, c, exp', e') \mid T \mid P \parallel P$

$T ::= \text{task}(e, t, M, exp, A, e') \mid \text{taskRcv}(e, t, M, exp, A, m : \tilde{t}, e')$

$\quad \mid \text{taskSnd}(e, t, M, exp, A, m : \tilde{exp}, e')$

$N ::= \text{na_c} \mid \text{na_nc}$

$A ::= \epsilon \mid \text{do}.f := \text{exp} \mid \text{ds}.f := \text{exp} \mid \text{get}(\text{do}) \mid \text{push}(\text{do}) \mid A, A$

We support models with **arbitrary topology**

FORMAL SEMANTICS

To simplify the formal treatment we resort to a **textual representation**

$C ::= \text{pool}(p, P) \mid \text{miPool}(p, P, max) \mid C_1 \parallel C_2$

$P ::= \text{start}(e, e') \mid \text{startRcv}(m : \tilde{t}, e) \mid \text{end}(e) \mid \text{endSnd}(e, m : \tilde{exp}) \mid \text{terminate}(e)$

$\quad \mid \text{interRcv}(e, m : \tilde{t}, e) \mid \text{interSnd}(e, m : \tilde{exp}, e)$

$\quad \mid \text{andSplit}(e, E) \mid \text{xorSplit}(e, G) \mid \text{andJoin}(E, e) \mid \text{xorJoin}(E, e)$

$\quad \mid \text{eventBased}(e, (m_1 : \tilde{t}_1, e_1), \dots, (m_h : \tilde{t}_h, e_h))$

$\quad \mid \text{mipTask}(e, exp, T, c, exp', e') \mid \text{misTask}(e, exp, T, c, exp', e') \mid T \mid P \parallel P$

$T ::= \text{task}(e, t, M, exp, A, e') \mid \text{taskRcv}(e, t, M, exp, A, m : \tilde{t}, e')$

$\quad \mid \text{taskSnd}(e, t, M, exp, A, m : \tilde{exp}, e')$

$N ::= \text{na_c} \mid \text{na_nc}$

$A ::= \epsilon \mid \text{do}.f := \text{exp} \mid \text{ds}.f := \text{exp} \mid \text{get(do)} \mid \text{push(do)} \mid A, A$



FORMAL SEMANTICS

To simplify the formal treatment we resort to a **textual representation**

$$C ::= \text{pool}(p, P) \mid \text{miPool}(p, P, max) \mid C_1 \parallel C_2$$

$$P ::= \text{start}(e, e') \mid \text{startRcv}(m : \tilde{t}, e) \mid \text{end}(e) \mid \text{endSnd}(e, m : \tilde{e}\tilde{x}p) \mid \text{terminate}(e)$$

$$\quad \mid \text{interRcv}(e, m : \tilde{t}, e) \mid \text{interSnd}(e, m : \tilde{e}\tilde{x}p, e)$$

$$\quad \mid \text{andSplit}(e, E) \mid \text{xorSplit}(e, G) \mid \text{andJoin}(E, e) \mid \text{xorJoin}(E, e)$$

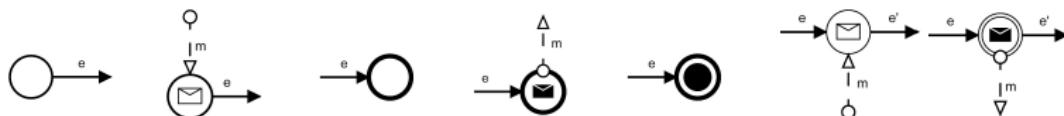
$$\quad \mid \text{eventBased}(e, (m_1 : \tilde{t}_1, e_1), \dots, (m_h : \tilde{t}_h, e_h))$$

$$\quad \mid \text{mipTask}(e, exp, T, c, exp', e') \mid \text{misTask}(e, exp, T, c, exp', e') \mid T \mid P \parallel P$$

$$T ::= \text{task}(e, t, M, exp, A, e') \mid \text{taskRcv}(e, t, M, exp, A, m : \tilde{t}, e')$$

$$\quad \mid \text{taskSnd}(e, t, M, exp, A, m : \tilde{e}\tilde{x}p, e')$$

$$N ::= \text{na_c} \mid \text{na_nc}$$

$$A ::= \epsilon \mid \text{do}.f := \text{exp} \mid \text{ds}.f := \text{exp} \mid \text{get(do)} \mid \text{push(do)} \mid A, A$$


FORMAL SEMANTICS

To simplify the formal treatment we resort to a **textual representation**

$C ::= \text{pool}(p, P) \mid \text{miPool}(p, P, max) \mid C_1 \parallel C_2$

$P ::= \text{start}(e, e') \mid \text{startRcv}(m : \tilde{t}, e) \mid \text{end}(e) \mid \text{endSnd}(e, m : \tilde{e}xp) \mid \text{terminate}(e)$

| $\text{interRcv}(e, m : \tilde{t}, e) \mid \text{interSnd}(e, m : \tilde{e}xp, e)$

| $\text{andSplit}(e, E) \mid \text{xorSplit}(e, G) \mid \text{andJoin}(E, e) \mid \text{xorJoin}(E, e)$

| $\text{eventBased}(e, (m_1 : \tilde{t}_1, e_1), \dots, (m_h : \tilde{t}_h, e_h))$

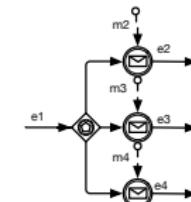
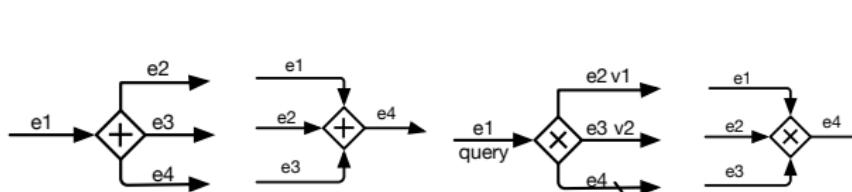
| $\text{mipTask}(e, exp, T, c, exp', e') \mid \text{misTask}(e, exp, T, c, exp', e') \mid T \mid P \parallel P$

$T ::= \text{task}(e, t, M, exp, A, e') \mid \text{taskRcv}(e, t, M, exp, A, m : \tilde{t}, e')$

| $\text{taskSnd}(e, t, M, exp, A, m : \tilde{e}xp, e')$

$N ::= \text{na_c} \mid \text{na_nc}$

$A ::= \epsilon \mid \text{do}.f := \text{exp} \mid \text{ds}.f := \text{exp} \mid \text{get(do)} \mid \text{push(do)} \mid A, A$



FORMAL SEMANTICS

To simplify the formal treatment we resort to a **textual representation**

$$C ::= \text{pool}(p, P) \mid \text{miPool}(p, P, max) \mid C_1 \parallel C_2$$

$$P ::= \text{start}(e, e') \mid \text{startRcv}(m : \tilde{t}, e) \mid \text{end}(e) \mid \text{endSnd}(e, m : \tilde{e}xp) \mid \text{terminate}(e)$$

$$\quad \mid \text{interRcv}(e, m : \tilde{t}, e) \mid \text{interSnd}(e, m : \tilde{e}xp, e)$$

$$\quad \mid \text{andSplit}(e, E) \mid \text{xorSplit}(e, G) \mid \text{andJoin}(E, e) \mid \text{xorJoin}(E, e)$$

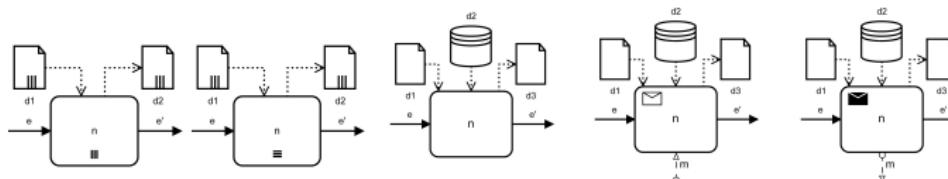
$$\quad \mid \text{eventBased}(e, (m_1 : \tilde{t}_1, e_1), \dots, (m_h : \tilde{t}_h, e_h))$$

$$\quad \mid \text{mipTask}(e, exp, T, c, exp', e') \mid \text{misTask}(e, exp, T, c, exp', e') \mid T \mid P \parallel P$$

$$T ::= \text{task}(e, t, M, exp, A, e') \mid \text{taskRcv}(e, t, M, exp, A, m : \tilde{t}, e')$$

$$\quad \mid \text{taskSnd}(e, t, M, exp, A, m : \tilde{e}xp, e')$$

$$N ::= \text{na_c} \mid \text{na_nc}$$

$$A ::= \epsilon \mid \text{do}.f := \text{exp} \mid \text{ds}.f := \text{exp} \mid \text{get(do)} \mid \text{push(do)} \mid A, A$$


FORMAL SEMANTICS

The textual representation is the mere structure of processes and collaborations, the semantics is given in terms of **stateful** descriptions:

- **collaboration configurations:** $\langle C, \sigma_i, \sigma_m, \sigma_{ds} \rangle$
 - C is a collaboration structure;
 - $\sigma_i : \mathbb{P} \rightarrow 2^{\mathbb{S}_{\sigma_e} \times \mathbb{S}_{\sigma_{do}} \times \mathbb{S}_{\sigma_{dc}} \times \mathbb{S}_{\sigma_t} \times \mathbb{S}_{\sigma_c}}$ is an *instance state function*
 - $\sigma_m : \mathbb{M} \rightarrow 2^{\mathbb{V}^n}$ is a *message state function*
 - σ_{ds} is a *data store state function*
- **process configurations:** $\langle P, \sigma_e, \sigma_{do}, \sigma_{dc}, \sigma_{ds}, \sigma_t, \sigma_c \rangle$
 - P is a process structure;
 - $\sigma_e : \mathbb{E} \rightarrow \mathbb{N}$ is a *sequence edge state function*
 - $\sigma_{do} : \mathbb{F} \rightarrow \mathbb{V}$ is a *data object state function*
 - $\sigma_{dc} : \mathbb{D} \rightarrow (\mathbb{F} \rightarrow \mathbb{V})^n$ is a *data collection state function*
 - $\sigma_{ds} : \mathbb{F} \rightarrow \mathbb{V}$ is a *data store state function*
 - $\sigma_t : \mathbb{T} \times \{a, s, r\} \rightarrow \mathbb{N}$ is a *task state function*
 - $\sigma_c : \mathbb{C} \rightarrow \mathbb{N}$ is a *counter state function*

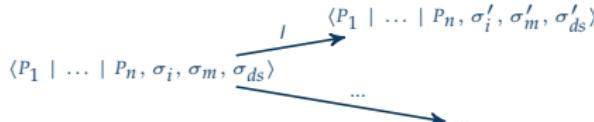
FORMAL SEMANTICS: OPERATIONAL RULES

- **Operational Semantics:** labelled transitions defined by rules of the form

$$\frac{\text{premise}_1 \dots \text{premise}_n}{\text{conclusion}} \begin{matrix} \text{conditions} \\ \text{conclusion} \end{matrix}$$

- **Compositional approach**

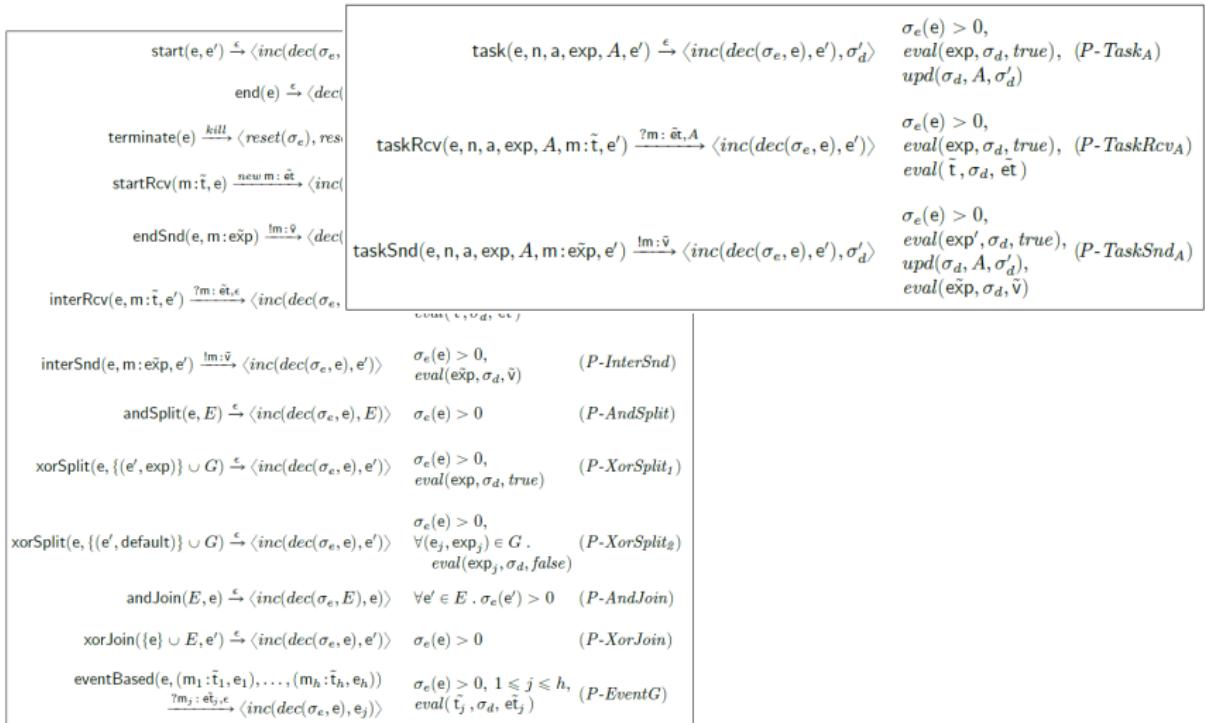
- Collaboration semantics is given in terms of the semantics of its pools
- Pool semantics is given in terms of the semantics of its process
- Process semantics is given in terms of the semantics of its elements
- Each process element has its semantic rules



FORMAL SEMANTICS: OPERATIONAL RULES

$\text{start}(e, e') \xrightarrow{\epsilon} \langle \text{inc}(\text{dec}(\sigma_e, e), e') \rangle$	$\sigma_e(e) > 0$	(P-Start)
$\text{end}(e) \xrightarrow{\epsilon} \langle \text{dec}(\sigma_e, e) \rangle$	$\sigma_e(e) > 0$	(P-End)
$\text{terminate}(e) \xrightarrow{\text{kill}} \langle \text{reset}(\sigma_e), \text{reset}(\sigma_t) \rangle$	$\sigma_e(e) > 0$	(P-Terminate)
$\text{startRcv}(m : \tilde{t}, e) \xrightarrow{\text{new } m : \tilde{e} t} \langle \text{inc}(\sigma_e, e) \rangle$	$\text{eval}(\tilde{t}, \sigma_d, \tilde{e} t)$	(P-StartRcv)
$\text{endSnd}(e, m : \tilde{e} \tilde{x} p) \xrightarrow{!m : \tilde{v}} \langle \text{dec}(\sigma_e, e) \rangle$	$\sigma_e(e) > 0,$ $\text{eval}(\tilde{e} \tilde{x} p, \sigma_d, \tilde{v})$	(P-EndSnd)
$\text{interRcv}(e, m : \tilde{t}, e') \xrightarrow{?m : \tilde{e} t, e} \langle \text{inc}(\text{dec}(\sigma_e, e), e') \rangle$	$\sigma_e(e) > 0,$ $\text{eval}(\tilde{t}, \sigma_d, \tilde{e} t)$	(P-InterRcv)
$\text{interSnd}(e, m : \tilde{e} \tilde{x} p, e') \xrightarrow{!m : \tilde{v}} \langle \text{inc}(\text{dec}(\sigma_e, e), e') \rangle$	$\sigma_e(e) > 0,$ $\text{eval}(\tilde{e} \tilde{x} p, \sigma_d, \tilde{v})$	(P-InterSnd)
$\text{andSplit}(e, E) \xrightarrow{\epsilon} \langle \text{inc}(\text{dec}(\sigma_e, e), E) \rangle$	$\sigma_e(e) > 0$	(P-AndSplit)
$\text{xorSplit}(e, \{(e', \text{exp})\} \cup G) \xrightarrow{\epsilon} \langle \text{inc}(\text{dec}(\sigma_e, e), e') \rangle$	$\sigma_e(e) > 0,$ $\text{eval}(\text{exp}, \sigma_d, \text{true})$	(P-XorSplit ₁)
$\text{xorSplit}(e, \{(e', \text{default})\} \cup G) \xrightarrow{\epsilon} \langle \text{inc}(\text{dec}(\sigma_e, e), e') \rangle$	$\sigma_e(e) > 0,$ $\forall (e_j, \text{exp}_j) \in G .$ $\text{eval}(\text{exp}_j, \sigma_d, \text{false})$	(P-XorSplit ₂)
$\text{andJoin}(E, e) \xrightarrow{\epsilon} \langle \text{inc}(\text{dec}(\sigma_e, E), e) \rangle$	$\forall e' \in E . \sigma_e(e') > 0$	(P-AndJoin)
$\text{xorJoin}(\{e\} \cup E, e') \xrightarrow{\epsilon} \langle \text{inc}(\text{dec}(\sigma_e, e), e') \rangle$	$\sigma_e(e) > 0$	(P-XorJoin)
$\text{eventBased}(e, (m_1 : \tilde{t}_1, e_1), \dots, (m_h : \tilde{t}_h, e_h))$ $\xrightarrow{?m_j : \tilde{e} t_j, e} \langle \text{inc}(\text{dec}(\sigma_e, e), e_j) \rangle$	$\sigma_e(e) > 0, 1 \leq j \leq h,$ $\text{eval}(\tilde{t}_j, \sigma_d, \tilde{e} t_j)$	(P-EventG)

FORMAL SEMANTICS: OPERATIONAL RULES



FORMAL SEMANTICS: OPERATIONAL RULES

	$\text{start}(e, e') \xrightarrow{\epsilon} \langle \text{inc}(\text{dec}(\sigma_e), e') \rangle$ $\text{end}(e) \xrightarrow{\epsilon} \langle \text{dec} \rangle$	$\text{task}(e, n, a, \text{exp}, A, e') \xrightarrow{\epsilon} \langle \text{inc}(\text{dec}(\sigma_e, e), e'), \sigma'_d \rangle$ $\sigma_e(e) > 0, \text{eval}(\text{exp}, \sigma_d, \text{true}), (\text{P-Task}_A)$ $\text{upd}(\sigma_d, A, \sigma'_d)$
	$\text{task}(e, n, N, \text{exp}, A, e') \xrightarrow{\epsilon} \langle \text{dec}(\sigma_e, e), \text{inc}(\sigma_t, n, a) \rangle$ $N = \text{na_nc} \Rightarrow \text{isInactive}(\sigma_t, n)$	$\sigma_e(e) > 0, \text{eval}(\text{exp}, \sigma_d, \text{true}), (\text{P-Task}_{N1})$ $\text{l}(\text{exp}, \sigma_d, \text{true}), (\text{P-TaskRcv}_A)$ $\text{l}(\tilde{t}, \sigma_d, \text{et})$
	$\text{task}(e, n, N, \text{exp}, A, e') \xrightarrow{\epsilon} \langle \text{inc}(\sigma_e, e'), \sigma'_d, \text{dec}(\sigma_t, n, a) \rangle$ $\sigma_t(n, a) > 0, \text{upd}(\sigma_d, A, \sigma'_d)$	(P-Task_{N2}) $\sigma_e(e) > 0,$ $\text{l}(\text{exp}', \sigma_d, \text{true}), (\text{P-TaskSnd}_A)$ $\text{l}(\sigma_d, A, \sigma'_d),$ $\text{l}(\tilde{e}, \sigma_d, \tilde{v})$
int	$\text{taskRcv}(e, n, N, \text{exp}, A, m:\tilde{t}, e') \xrightarrow{\epsilon} \langle \text{dec}(\sigma_e, e), \text{inc}(\sigma_t, n, a) \rangle$ $N = \text{na_nc} \Rightarrow \text{isInactive}(\sigma_t, n)$	$\sigma_e(e) > 0, \text{eval}(\text{exp}, \sigma_d, \text{true}), (\text{P-TaskRcv}_{N1})$ $\text{l}(\tilde{t}, \sigma_d, \text{et})$
inte	$\text{taskRcv}(e, n, N, \text{exp}, A, m:\tilde{t}, e') \xrightarrow{?m:\tilde{e}\tilde{t}, e} \langle \text{inc}(\text{dec}(\sigma_t, n, a), n, r) \rangle$ $\sigma_t(n, a) > 0, \text{eval}(\tilde{t}, \sigma_d, \tilde{e}t)$	(P-TaskRcv_{N2})
	$\text{taskRcv}(e, n, N, \text{exp}, A, m:\tilde{t}, e') \xrightarrow{\epsilon} \langle \text{inc}(\sigma_e, e'), \sigma'_d, \text{dec}(\sigma_t, n, r) \rangle$ $\sigma_t(n, r) > 0, \text{upd}(\sigma_d, A, \sigma'_d)$	(P-TaskRcv_{N3})
xorS	$\text{taskSnd}(e, n, N, \text{exp}, A, m:\tilde{e}\tilde{x}, e') \xrightarrow{\epsilon} \langle \text{dec}(\sigma_e, e), \text{inc}(\sigma_t, n, a) \rangle$ $N = \text{na_nc} \Rightarrow \text{isInactive}(\sigma_t, n)$	$\sigma_e(e) > 0, \text{eval}(\text{exp}, \sigma_d, \text{true}), (\text{P-TaskSnd}_{N1})$ $\text{l}(\tilde{e}\tilde{x}, \sigma_d, \text{et})$
	$\text{taskSnd}(e, n, N, \text{exp}, A, m:\tilde{e}\tilde{x}, e') \xrightarrow{\epsilon} \langle \sigma'_d, \text{inc}(\text{dec}(\sigma_t, n, a), n, s) \rangle$ $\sigma_t(n, a) > 0, \text{upd}(\sigma_d, A, \sigma'_d)$	(P-TaskSnd_{N2})
xorSplit	$\text{taskSnd}(e, n, N, \text{exp}, A, m:\tilde{e}\tilde{x}, e') \xrightarrow{!m:\tilde{q}} \langle \text{inc}(\sigma_e, e'), \text{dec}(\sigma_t, n, s) \rangle$ $\sigma_t(n, s) > 0, \text{eval}(\tilde{e}\tilde{x}, \sigma_d, \tilde{v})$	(P-TaskSnd_{N3})
	$\text{xorJoin}(\{e\} \cup E, e') \xrightarrow{\epsilon} \langle \text{inc}(\text{dec}(\sigma_e, e), e') \rangle$ $\sigma_e(e) > 0$	(P-XorJoin)
	$\text{eventBased}(e, (m_1:\tilde{t}_1, e_1), \dots, (m_h:\tilde{t}_h, e_h))$ $\xrightarrow{?m_j:\tilde{e}\tilde{t}_j, e} \langle \text{inc}(\text{dec}(\sigma_e, e), e_j) \rangle$ $\sigma_e(e) > 0, 1 \leq j \leq h,$ $\text{eval}(\tilde{t}_j, \sigma_d, \tilde{e}\tilde{t}_j)$	(P-EventG)

FORMAL SEMANTICS: OPERATIONAL RULES

		$\sigma_e(e) > 0,$ $\sigma_c(c) = 0,$ $eval(exp, \sigma_d, h)$ with $h > 0$	$\vdash\text{-Task}_A$
	$start(e, e') \xrightarrow{e} \langle inc(e), \dots \rangle$		
	$end(e)$		
int	$task(e, n, N, e)$	$mipTask(e, exp, T, c, exp', e') \xrightarrow{e} \langle set(dec(\sigma_e, e), in(T), h), set(\sigma_c, c, h) \rangle$	$(P\text{-MipTask}_1)$
inte	$task(e, n, N, exp, _)$	$mipTask(e, exp, T, c, exp', e') \xrightarrow{e} \langle inc(dec(\sigma_e, e), e') \rangle$	$(P\text{-MipTask}_2)$
xorS	$taskRcv(e, n, N, exp, A)$	$\frac{\langle T, \sigma_e, \sigma_d, \sigma_t, \sigma_c \rangle \xrightarrow{e} \langle \sigma'_e, \sigma'_d, \sigma'_t, \sigma'_c \rangle}{mipTask(e, exp, T, c, exp', e') \xrightarrow{e} \langle \sigma'_e, \sigma'_d, \sigma'_t, \sigma'_c \rangle}$	$(P\text{-MipTask}_3)$
xorSplit	$taskRcv(e, n, N, exp, A, m:)$	$mipTask(e, exp, T, c, exp', e') \xrightarrow{e} \langle inc(reset(\sigma_e, edges(T)), e'), reset(\sigma_c, c) \rangle$	$\sigma_e(out(T)) = \sigma_c(c) \vee eval(exp', \sigma_d, true)$ $(P\text{-MipTask}_4)$
	$taskRcv(e, n, N, exp, A, m:)$		
	$taskSnd(e, n, N, exp, A, _)$	$misTask(e, exp, T, c, exp', e') \xrightarrow{e} \langle inc(dec(\sigma_e, e), in(T)), set(\sigma_c, c, h) \rangle$	$(P\text{-MisTask}_1)$
	$taskSnd(e, n, N, exp, A, m:)$	$misTask(e, exp, T, c, exp', e') \xrightarrow{e} \langle inc(dec(\sigma_e, e), e') \rangle$	$(P\text{-MisTask}_2)$
	$taskSnd(e, n, N, exp, A, m: e)$	$\frac{\langle T, \sigma_e, \sigma_d, \sigma_t, \sigma_c \rangle \xrightarrow{e} \langle \sigma'_e, \sigma'_d, \sigma'_t, \sigma'_c \rangle}{misTask(e, exp, T, c, exp', e') \xrightarrow{e} \langle \sigma'_e, \sigma'_d, \sigma'_t, \sigma'_c \rangle}$	$(P\text{-MisTask}_3)$
	$xorJoin(\{e\} \cup E, e') \xrightarrow{e} \langle inc(e), \dots \rangle$		
	$eventBased(e, (m_1 : \tilde{t}_1, e_1), \dots)$	$misTask(e, exp, T, c, exp', e') \xrightarrow{e} \langle inc(dec(\sigma_e, out(T)), in(T)), dec(\sigma_c, c) \rangle$	$\sigma_c(c) > 1,$ $\sigma_e(out(T)) = 1,$ $eval(exp', \sigma_d, false)$ $(P\text{-MisTask}_4)$
	$\xrightarrow{?m_2 : \tilde{e}_2, e} \langle inc(e), \dots \rangle$	$misTask(e, exp, T, c, exp', e') \xrightarrow{e} \langle inc(dec(\sigma_e, out(T)), e'), reset(\sigma_c, c) \rangle$	$\sigma_e(out(T)) = 1,$ $(\sigma_c(c) = 1 \vee eval(exp', \sigma_d, true))$ $(P\text{-MisTask}_5)$

FORMAL SEMANTICS: OPERATIONAL RULES

		$\sigma_e(e) > 0,$ $\sigma_c(c) = 0,$ $eval(exp, \sigma_d, h)$ with $h > 0$	$\vdash\text{-Task}_A$
	$start(e, e') \xrightarrow{\ell} \langle inc \rangle$		
	$end(e)$		
	$task(e, n, N, e)$		
	$task(e, n, N, exp, _)$	$mipTask(e, exp, T, c, exp', e') \xrightarrow{\ell} \langle set(dec(\sigma_e, e), in(T), h), set(\sigma_c, c, h) \rangle$	$(P\text{-MipTask}_1)$
	$taskRcv(e, n, N, _, _, A)$	$mipTask(e, exp, T, c, exp', e') \xrightarrow{\ell} \langle inc(dec(\sigma_e, e), e') \rangle$	$(P\text{-MipTask}_2)$
int	$taskRcv(e, n, N, exp, _, _)$	$\frac{\langle T, \sigma_e, \sigma_d, \sigma_t, \sigma_c \rangle \xrightarrow{\ell} \langle \sigma'_e, \sigma'_d, \sigma'_t, \sigma'_c \rangle}{P_1 \parallel P_2 \xrightarrow{\ell} \langle \sigma'_e, \sigma'_d, \sigma'_t, \sigma'_c \rangle}$	$(P\text{-MipTask}_3)$
inte	$taskRcv(e, n, N, exp, A, m:)$	$\frac{\langle P_1, \sigma_e, \sigma_d, \sigma_t, \sigma_c \rangle \xrightarrow{\ell} \langle \sigma'_e, \sigma'_d, \sigma'_t, \sigma'_c \rangle \quad \langle P_2, \sigma_e, \sigma_d, \sigma_t, \sigma_c \rangle \xrightarrow{\ell} \langle \sigma'_e, \sigma'_d, \sigma'_t, \sigma'_c \rangle}{P_1 \parallel P_2 \xrightarrow{\ell} \langle \sigma'_e, \sigma'_d, \sigma'_t, \sigma'_c \rangle}$	$(P\text{-Int}_1) \quad (P\text{-Int}_2)$
xorS	$taskRcv(e, n, N, exp, A, m:)$	$misTask(e, exp, T, c, exp', e') \xrightarrow{\ell} \langle inc(dec(\sigma_e, e), in(T)), set(\sigma_c, c, h) \rangle$	$(P\text{-MisTask}_1)$
xorS	$taskSnd(e, n, N, exp, A, m:)$	$misTask(e, exp, T, c, exp', e'), \xrightarrow{\ell} \langle inc(dec(\sigma_e, e), e') \rangle$	$(P\text{-MisTask}_2)$
xorSplit	$taskSnd(e, n, N, exp, A, m: e)$	$\frac{\langle T, \sigma_e, \sigma_d, \sigma_t, \sigma_c \rangle \xrightarrow{\ell} \langle \sigma'_e, \sigma'_d, \sigma'_t, \sigma'_c \rangle}{misTask(e, exp, T, c, exp', e') \xrightarrow{\ell} \langle \sigma'_e, \sigma'_d, \sigma'_t, \sigma'_c \rangle}$	$(P\text{-MisTask}_3)$
	$xorJoin(\{e\} \cup E, e') \xrightarrow{\ell} \langle inc \rangle$		
	$eventBased(e, (m_1 : \tilde{t}_1, e_1), \dots)$	$misTask(e, exp, T, c, exp', e') \xrightarrow{\ell} \langle inc(dec(\sigma_e, out(T)), in(T)), dec(\sigma_c, c) \rangle$	$\sigma_c(c) > 1,$ $\sigma_e(out(T)) = 1,$ $eval(exp', \sigma_d, false)$
	$\frac{?m_2 : \tilde{e}_2, e}{\langle inc(e) \rangle}$	$misTask(e, exp, T, c, exp', e') \xrightarrow{\ell} \langle inc(dec(\sigma_e, out(T)), e'), reset(\sigma_c, c) \rangle$	$\sigma_e(out(T)) = 1,$ $(\sigma_c(c) = 1 \vee eval(exp', \sigma_d, true))$

FORMAL SEMANTICS: OPERATIONAL RULES

		$\sigma_e(e) > 0,$	
	start($e^{\wedge \ell_1 \wedge \dots \wedge \ell_n}$)	$\vdash \text{-Task}_A$	
	ta	k_I	
	task(ϵ)	k_2	
	taskRcv(e)	k_3	
int	taskRcv(e, n, Λ)	t_2	
inte	taskRcv(e, n, λ)	k_I	
xorS	taskSnd(e, n, \neg)	k_I	
xorSplit	taskSnd(e, n, N)	k_2	
	xorJoin($\{e\} \cup E$)	k_3	
	eventBased($e, (m)$)	k_4	
		$\sigma_e(e) = 1,$ $\sigma_e(\text{out}(T)) = 1,$ $\sigma_e(c) = 1 \vee$ $\text{eval}(\text{exp}', \sigma_d, \text{true}))$	
		$(P\text{-MisTask}_5)$	
		$\vdash \text{-TaskRcv}_A$	
		$\vdash \text{-TaskSnd}_A$	
		$\vdash \text{-Task}_B$	
		$\vdash \text{-Task}_C$	
		$\vdash \text{-Task}_D$	
		$\vdash \text{-Task}_E$	
		$\vdash \text{-Task}_F$	
		$\vdash \text{-Task}_G$	
		$\vdash \text{-Task}_H$	
		$\vdash \text{-Task}_I$	
		$\vdash \text{-Task}_J$	
		$\vdash \text{-Task}_K$	
		$\vdash \text{-Task}_L$	
		$\vdash \text{-Task}_M$	
		$\vdash \text{-Task}_N$	
		$\vdash \text{-Task}_O$	
		$\vdash \text{-Task}_P$	
		$\vdash \text{-Task}_Q$	
		$\vdash \text{-Task}_R$	
		$\vdash \text{-Task}_S$	
		$\vdash \text{-Task}_T$	
		$\vdash \text{-Task}_U$	
		$\vdash \text{-Task}_V$	
		$\vdash \text{-Task}_W$	
		$\vdash \text{-Task}_X$	
		$\vdash \text{-Task}_Y$	
		$\vdash \text{-Task}_Z$	
		$\vdash \text{-Task}_A'$	
		$\vdash \text{-Task}_B'$	
		$\vdash \text{-Task}_C'$	
		$\vdash \text{-Task}_D'$	
		$\vdash \text{-Task}_E'$	
		$\vdash \text{-Task}_F'$	
		$\vdash \text{-Task}_G'$	
		$\vdash \text{-Task}_H'$	
		$\vdash \text{-Task}_I'$	
		$\vdash \text{-Task}_J'$	
		$\vdash \text{-Task}_K'$	
		$\vdash \text{-Task}_L'$	
		$\vdash \text{-Task}_M'$	
		$\vdash \text{-Task}_N'$	
		$\vdash \text{-Task}_O'$	
		$\vdash \text{-Task}_P'$	
		$\vdash \text{-Task}_Q'$	
		$\vdash \text{-Task}_R'$	
		$\vdash \text{-Task}_S'$	
		$\vdash \text{-Task}_T'$	
		$\vdash \text{-Task}_U'$	
		$\vdash \text{-Task}_V'$	
		$\vdash \text{-Task}_W'$	
		$\vdash \text{-Task}_X'$	
		$\vdash \text{-Task}_Y'$	
		$\vdash \text{-Task}_Z'$	
		$\vdash \text{-Task}_A''$	
		$\vdash \text{-Task}_B''$	
		$\vdash \text{-Task}_C''$	
		$\vdash \text{-Task}_D''$	
		$\vdash \text{-Task}_E''$	
		$\vdash \text{-Task}_F''$	
		$\vdash \text{-Task}_G''$	
		$\vdash \text{-Task}_H''$	
		$\vdash \text{-Task}_I''$	
		$\vdash \text{-Task}_J''$	
		$\vdash \text{-Task}_K''$	
		$\vdash \text{-Task}_L''$	
		$\vdash \text{-Task}_M''$	
		$\vdash \text{-Task}_N''$	
		$\vdash \text{-Task}_O''$	
		$\vdash \text{-Task}_P''$	
		$\vdash \text{-Task}_Q''$	
		$\vdash \text{-Task}_R''$	
		$\vdash \text{-Task}_S''$	
		$\vdash \text{-Task}_T''$	
		$\vdash \text{-Task}_U''$	
		$\vdash \text{-Task}_V''$	
		$\vdash \text{-Task}_W''$	
		$\vdash \text{-Task}_X''$	
		$\vdash \text{-Task}_Y''$	
		$\vdash \text{-Task}_Z''$	
		$\vdash \text{-Task}_A'''$	
		$\vdash \text{-Task}_B'''$	
		$\vdash \text{-Task}_C'''$	
		$\vdash \text{-Task}_D'''$	
		$\vdash \text{-Task}_E'''$	
		$\vdash \text{-Task}_F'''$	
		$\vdash \text{-Task}_G'''$	
		$\vdash \text{-Task}_H'''$	
		$\vdash \text{-Task}_I'''$	
		$\vdash \text{-Task}_J'''$	
		$\vdash \text{-Task}_K'''$	
		$\vdash \text{-Task}_L'''$	
		$\vdash \text{-Task}_M'''$	
		$\vdash \text{-Task}_N'''$	
		$\vdash \text{-Task}_O'''$	
		$\vdash \text{-Task}_P'''$	
		$\vdash \text{-Task}_Q'''$	
		$\vdash \text{-Task}_R'''$	
		$\vdash \text{-Task}_S'''$	
		$\vdash \text{-Task}_T'''$	
		$\vdash \text{-Task}_U'''$	
		$\vdash \text{-Task}_V'''$	
		$\vdash \text{-Task}_W'''$	
		$\vdash \text{-Task}_X'''$	
		$\vdash \text{-Task}_Y'''$	
		$\vdash \text{-Task}_Z'''$	
		$\vdash \text{-Task}_A''''$	
		$\vdash \text{-Task}_B''''$	
		$\vdash \text{-Task}_C''''$	
		$\vdash \text{-Task}_D''''$	
		$\vdash \text{-Task}_E''''$	
		$\vdash \text{-Task}_F''''$	
		$\vdash \text{-Task}_G''''$	
		$\vdash \text{-Task}_H''''$	
		$\vdash \text{-Task}_I''''$	
		$\vdash \text{-Task}_J''''$	
		$\vdash \text{-Task}_K''''$	
		$\vdash \text{-Task}_L''''$	
		$\vdash \text{-Task}_M''''$	
		$\vdash \text{-Task}_N''''$	
		$\vdash \text{-Task}_O''''$	
		$\vdash \text{-Task}_P''''$	
		$\vdash \text{-Task}_Q''''$	
		$\vdash \text{-Task}_R''''$	
		$\vdash \text{-Task}_S''''$	
		$\vdash \text{-Task}_T''''$	
		$\vdash \text{-Task}_U''''$	
		$\vdash \text{-Task}_V''''$	
		$\vdash \text{-Task}_W''''$	
		$\vdash \text{-Task}_X''''$	
		$\vdash \text{-Task}_Y''''$	
		$\vdash \text{-Task}_Z''''$	

CAKE DELIVERY SCENARIO

» FORMALISATION

Overall cake preparation collaboration scenario:

$$\text{pool}(\text{p}_c, P_c) \parallel \text{pool}(\text{p}_p, P_p) \parallel \text{miPool}(\text{p}_a, P_a, 3)$$

CAKE DELIVERY SCENARIO

»CUSTOMER

Customer process:

$$P_c = \text{start}(e_1, e_2) \parallel \text{taskSnd}(e_2, \text{SendCakeRequest}, a, \exp_1, A_1, \text{CakeRequest} : \tilde{\exp}_2, e_3) \parallel \\ \text{taskSnd}(e_3, \text{SendDecorationsRequest}, a, \exp_3, A_2, \text{Decorations} : \tilde{\exp}_4, e_4) \parallel \\ \text{taskRcv}(e_4, \text{ReceiveCake}, a, \text{true}, \epsilon, \text{Cake} : t_1, e_5) \parallel \\ \text{xorSplit}(e_5, \{(e_6, \text{Cake.cake} \neq \text{Desiderata.cake}), (e_7, \text{Cake.cake} = \text{Desiderata.cake})\}) \parallel \\ \text{end}(e_6) \parallel \text{task}(e_7, \text{Celebrate}, a, \text{true}, \epsilon, e_8) \parallel \text{end}(e_8)$$

Templates, expressions, assignments

\exp_1	=	LayersInfo.top \neq null and LayersInfo.middle \neq null and LayersInfo.bottom \neq null
A_1	=	Desiderata.top := LayersInfo.top, Desiderata.middle := LayersInfo.middle, Desiderata.bottom := LayersInfo.bottom
$\tilde{\exp}_2$	=	(LayersInfo.top, LayersInfo.middle, LayersInfo.bottom)
A_2	=	Desiderata.cake := Desiderata.top + '&' + DecorationsInfo.top + 'on' + Desiderata.middle + '&' + DecorationsInfo.middle + 'on' + Desiderata.bottom + '&' + DecorationsInfo.bottom
$\tilde{\exp}_4$	=	(DecorationsInfo.top, DecorationsInfo.middle, DecorationsInfo.bottom)
t_1	=	(?Cake.cake)

CAKE DELIVERY SCENARIO

» PASTRY CHEF

Pastry Chef process

```

 $P_p = \text{startRcv(CakeRequest : } t_{\bar{21}}, e_{21}) \parallel \text{task}(e_{21}, \text{CheckCakeRequest}, a, \text{exp}_{21}, A_{21}, e_{22}) \parallel$ 
 $\text{mipTask}(e_{22}, 3, \text{taskSnd}(e'_{22}, \text{AssignLayers}, a, \text{exp}_{22}, A_{22}, \text{LayerRequest : exp}_{23}, e'_{23}), c_1, \text{false}, e_{23}) \parallel$ 
 $\text{taskRcv}(e_{23}, \text{ReceiveDecorations}, a, \text{exp}_{24}, A_{23}, \text{Decorations : } t_{\bar{22}}, e_{24}) \parallel$ 
 $\text{mipTask}(e_{24}, 3, \text{taskSnd}(e'_{24}, \text{AssignDecorations}, a, \text{exp}_{25}, A_{24}, \text{Decoration : exp}_{26}, e'_{25}), c_2, \text{false}, e_{25}) \parallel$ 
 $\text{misTask}(e_{25}, 3, \text{taskRcv}(e'_{25}, \text{Receive\&CombineLayers}, a, \text{exp}_{27}, A_{25}, \text{Layer : } t_{\bar{23}}, e'_{26}), c_3, \text{false}, e_{26}) \parallel$ 
 $\text{taskSnd}(e_{26}, \text{SendCake}, a, \text{exp}_{28}, \epsilon, m : \text{exp}_{29}, e_{27}) \parallel \text{end}(e_{27})$ 

```

Templates, expressions, assignments

$t_{\bar{21}}$	=	$\langle ?\text{LayersRequest.top}, ?\text{LayersRequest.middle}, ?\text{LayersRequest.bottom} \rangle$
A_{21}	=	$\text{LayersPlan.position := 'bottom'}, \text{LayersPlan.layerColor := LayersRequest.bottom, push(LayersPlan)},$ $\text{LayersPlan.position := 'middle'}, \text{LayersPlan.layerColor := LayersRequest.middle, push(LayersPlan)},$ $\text{LayersPlan.position := 'top'}, \text{LayersPlan.layerColor := LayersRequest.top, push(LayersPlan)}$
A_{22}	=	get(LayersPlan)
exp_{23}	=	$\langle \text{LayersPlan.layerColor}, \text{LayersPlan.position} \rangle$
A_{23}	=	$\text{LayersPlan.position := 'bottom'}, \text{LayersPlan.decorationColor := DecorationsRequest.bottom, push(LayersPlan)},$ $\text{LayersPlan.position := 'middle'}, \text{LayersPlan.decorationColor := DecorationsRequest.middle, push(LayersPlan)},$ $\text{LayersPlan.position := 'top'}, \text{LayersPlan.decorationColor := DecorationsRequest.top, push(LayersPlan)}$
$t_{\bar{22}}$	=	$\langle ?\text{DecorationsRequest.top}, ?\text{DecorationsRequest.middle}, ?\text{DecorationsRequest.bottom} \rangle$
A_{24}	=	get(LayersPlan)
exp_{26}	=	$\langle \text{LayersPlan.position}, \text{LayersPlan.decorationColor} \rangle$
A_{25}	=	$\text{Cake.cake := addLayer(DecoratedLayer.layer, DecoratedLayer.position)}, \text{Cake.numLayers := Cake.numLayers + 1}$
$t_{\bar{23}}$	=	$\langle ?\text{DecoratedLayer.layer}, ?\text{DecoratedLayer.position} \rangle$
exp_{28}	=	$\text{Cake.numLayers = 3}$
exp_{29}	=	$\langle \text{Cake.cake} \rangle$

CAKE DELIVERY SCENARIO

»ASSISTANT

Assistant process

$$P_a = \text{startRcv}(\text{LayerRequest} : t_{31}, e_{31}) \parallel \text{task}(e_{31}, \text{PrepareLayer}, a, \text{exp}_{31}, A_{31}, e_{32}) \parallel \\ \text{taskRcv}(e_{32}, \text{ReceiveDecoration}, a, \text{true}, \epsilon, \text{Decoration} : t_{32}, e_{33}) \parallel \\ \text{task}(e_{33}, \text{ApplyDecoration}, a, \text{exp}_{32}, A_{32}, e_{34}) \parallel \\ \text{taskSnd}(e_{34}, \text{SendDecoratedLayer}, a, \text{exp}_{33}, \epsilon, \text{Layer} : \tilde{\text{exp}}_{34}, e_{35}) \parallel \text{end}(e_{35})$$

Templates, expressions, assignments

t_{31}	=	$\langle ?\text{LayerPlan.layerColor}, ?\text{LayerPlan.position} \rangle$
A_{31}	=	$\text{Layer.status} := \text{'prepared'}$
t_{32}	=	$\langle \text{LayerPlan.position}, ?\text{LayerPlan.decorationColor} \rangle$
exp_{32}	=	$\text{Layer.status} = \text{'prepared'}$
A_{32}	=	$\text{Layer.status} := \text{'decorated'}, \text{Layer.position} := \text{LayerPlan.position},$ $\text{Layer.layer} := \text{LayerPlan.layerColor} + \& + \text{LayerPlan.decorationColor}$
exp_{33}	=	$\text{Layer.status} = \text{'decorated'}$
$\tilde{\text{exp}}_{34}$	=	$\langle \text{Layer.layer}, \text{Layer.position} \rangle$

I think it is

crystal clear

Animation

MIDA

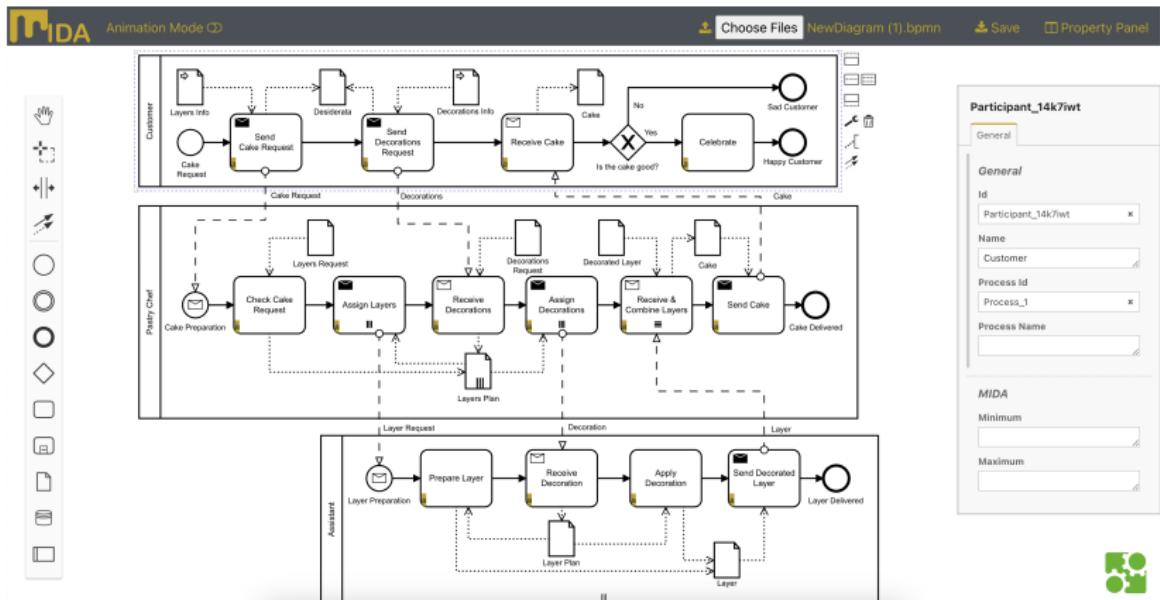
MIDA is a free web application for animating BPMN collaboration with multiple instances, data and messages



The MIDA tool, its source code, examples, user guide, and screencast freely available at <https://pros.unicam.it/mida/>

MODELING

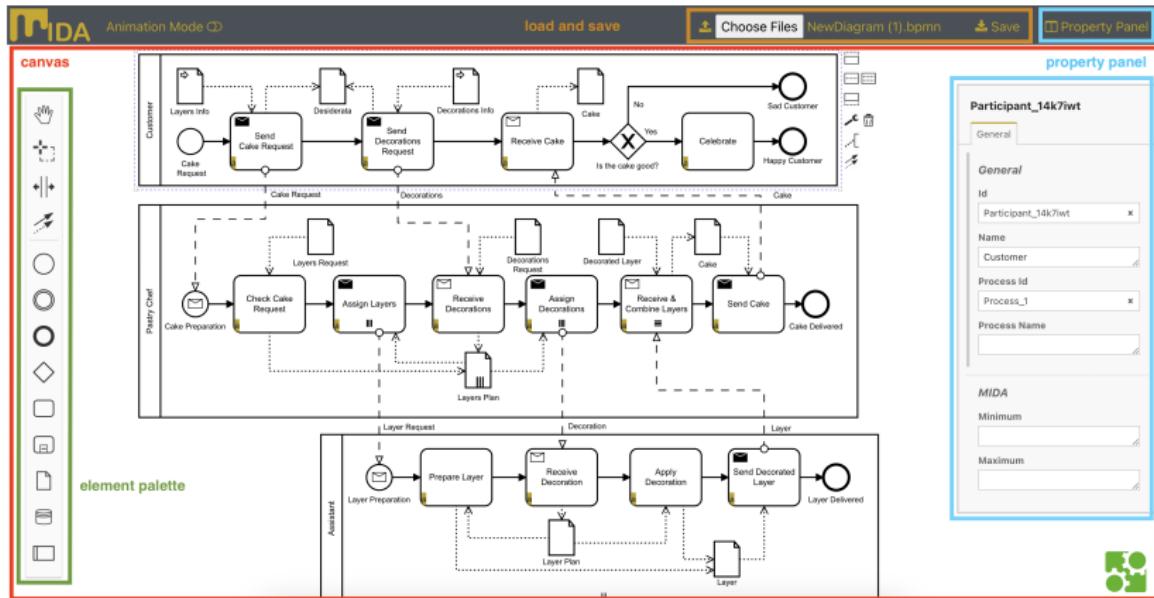
Mida exploits the *bpmn.io* web modeler



For each graphical element, the modeler can specify additional information related to our semantics

MODELING

Mida exploits the *bpmn.io* web modeler



For each graphical element, the modeler can specify additional information related to our semantics

MODELING

»MULTIPLE INSTANCES

Mida allows to model MI pools and tasks

Selected element



Property panel

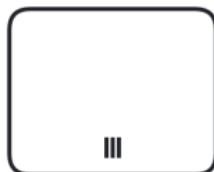
Minimum	<input type="text" value="1"/>
Maximum	<input type="text" value="10"/>

Loop Cardinality

Collection

Element Variable

Completion Condition



MODELING

»DATA

Mida allows to model data fields used by guard conditions, assignments, and XOR conditions

Selected element



Property panel

Data Fields

x	x
y = 1	x
z = "hello"	x

Guard

x == 10	x
---------	---

Add Assignment

Assignments

x = x + 1	x
y = 4	x

Expression

x > 0	x
-------	---

MODELING

»MESSAGES

Mida allows to model messages carrying data

Selected element



Property panel

Fields

"hello"	x
"id_01"	x



Fields

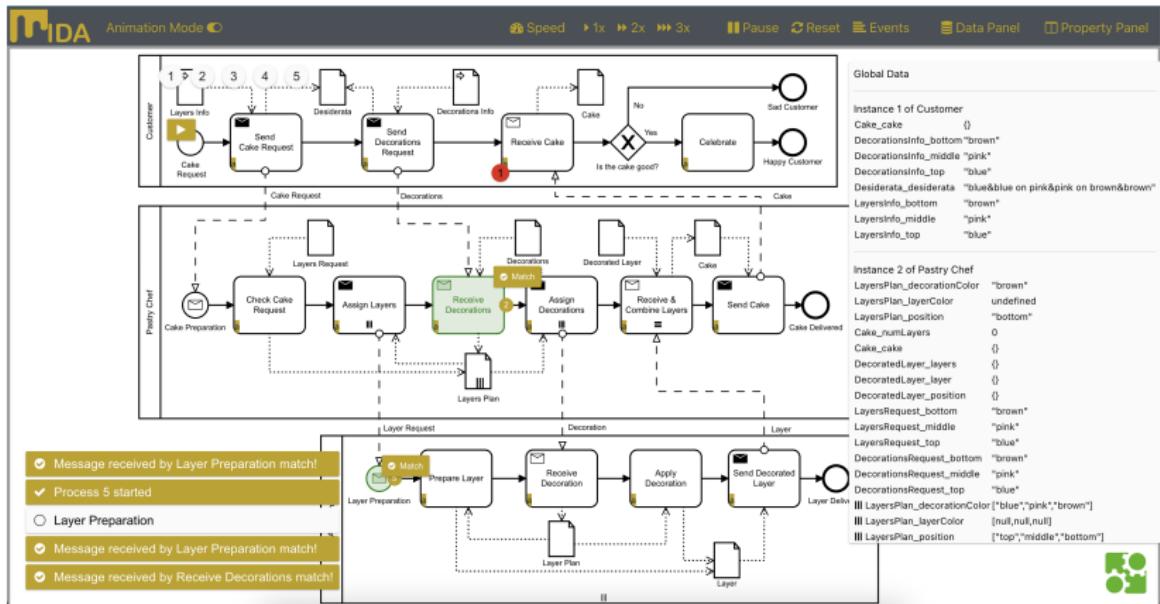
message
identifier

Correlation

<input type="checkbox"/>	x
<input checked="" type="checkbox"/>	x

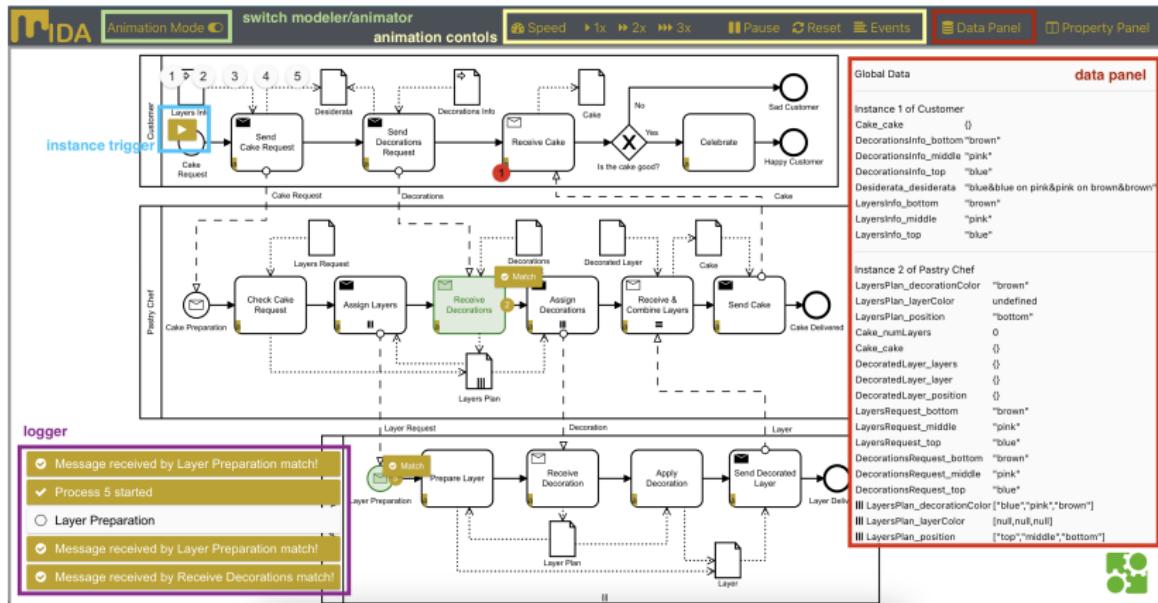
ANIMATION

The tool shows the model execution via token game



ANIMATION

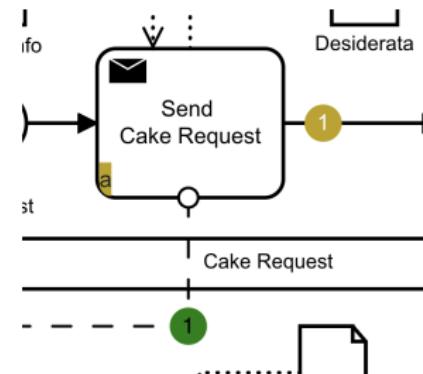
The tool shows the model execution via token game



ANIMATION

By pausing the animation, the modeler can look at the **flow of tokens**

- **Control-flow tokens** show the marking of each process instance
- **Message tokens** show the delivery of messages



- Each token reports an instance identifier (1, 2, 3, ...)
- The animation ends once no more moves are allowed

ANIMATION

By pausing the animation, the modeler can look at the **evolution of data**

The data panel reports changes in the status of data fields

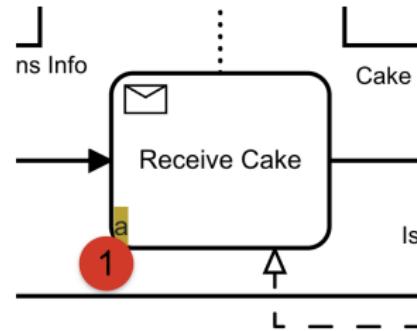
Demo

Global Data	
Instance 1 of Customer	
Cake_cake	[]
DecorationsInfo_bottom	"brown"
DecorationsInfo_middle	"pink"
DecorationsInfo_top	"blue"
Desiderata_desiderata	"blue&blue on pink&pink on brown&brown"
LayersInfo_bottom	"brown"
LayersInfo_middle	"pink"
LayersInfo_top	"blue"
Instance 2 of Pastry Chef	
Cake_numLayers	0
Cake_cake	[]
DecoratedLayer_layers	0
DecoratedLayer_layer	0
DecoratedLayer_position	0
LayersRequest_bottom	"brown"
LayersRequest_middle	"pink"
LayersRequest_top	"blue"
DecorationsRequest_bottom	0
DecorationsRequest_middle	null
DecorationsRequest_top	null
III LayersPlan_decorationColor	[]
III LayersPlan_layerColor	[]
III LayersPlan_position	[]

DEBUGGING

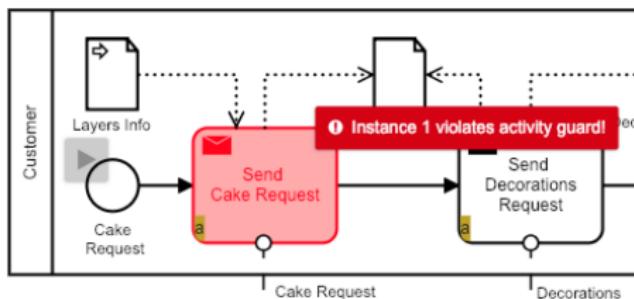
Mida highlights in red **pending tokens or violations** (e.g., guard conditions, XOR conditions, constraints about maximum number of instances)

Pending tokens are reported in red to show execution flows waiting for messages or synchronization



DEBUGGING

Mida highlights in red **pending tokens or violations** (e.g., guard conditions, XOR conditions, constraints about maximum number of instances)

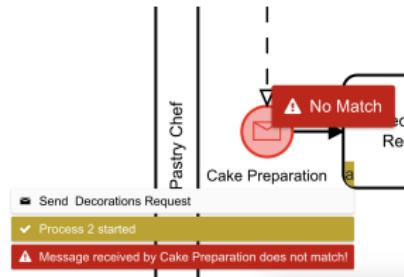


Violations of **guard conditions** block the token and trigger an exception

DEBUGGING

Mida highlights in red **pending tokens or violations** (e.g., guard conditions, XOR conditions, constraints about maximum number of instances)

Violations of **message template** block the receiving of message

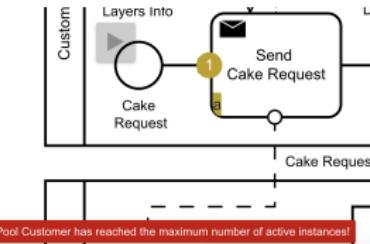


DEBUGGING

Mida highlights in red **pending tokens or violations** (e.g., guard conditions, XOR conditions, constraints about maximum number of instances)

Reaching the limit of **process instances**
stops the production of tokens

Demo



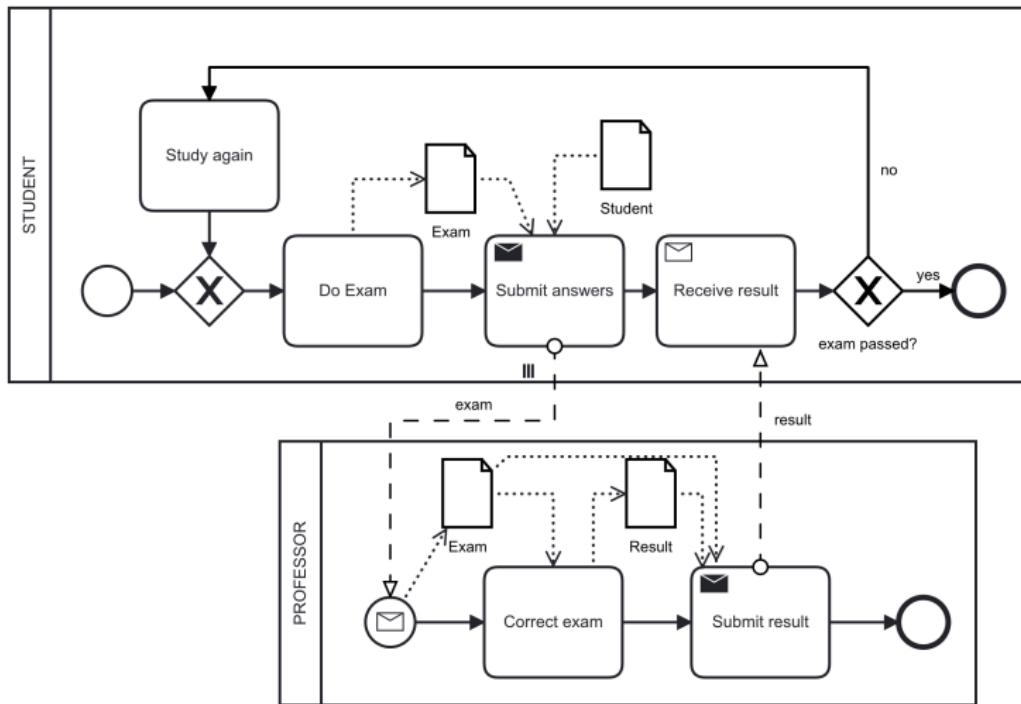
EXERCISE

Model and animate: Design a collaboration between **students** and a **professor** for a course exam.

- The student performs the exam valorising 3 answers with a *true* or *false* value, and sends it to the professor.
- Therefore, the professor receives the answers and a student identifier, give 10 points to each correct answer and submit the result with the student identifier.
- Finally, the student receives its own result and, if passed ($\text{mark} > 17$), terminates the process, otherwise she/he study again and re-do the exam.

ANSWER

A possible process implementation



ANSWER

A possible process implementation

- **Exam:**

Declarations: Exam_answer1, Exam_answer2, Exam_answer3

- **Student:**

Declarations: Student_id = 101010

- **Do Exam:**

Assignments: Exam_answer1 = true, Exam_answer2 = false,
Exam_answer3 = true

- **Submit answers:**

Guard: (Exam_answer1 != null) & (Exam_answer2 != null) &
(Exam_answer3 != null)

Message: Student_id, Exam_answer1, Exam_answer2, Exam_answer3

- **Receive result:**

Template: Student_id, Exam_answer1, Exam_answer2, Exam_answer3

ANSWER

A possible process implementation

- **Exam:**

Declarations: Exam_id, Exam_ans1, Exam_ans2, Exam_ans3

- **Result:**

Declarations: Result_res = 0

- **Correct Exam:**

Assignments: Result_res += 10*(Exam_ans1) + 10*(Exam_ans2) + 10*(Exam_ans3)

- **Submit result:**

Message: Exam_id, Result_res

Conclusion

CONCLUSION

We provide

- **a formal semantics**

- Corradini F., Muzi C., Re B., Rossi L., Tiezzi F.. *Formalising and animating multiple instances in BPMN collaborations*. In: Information Systems. 103, 101459. Springer, 2022
- Corradini F., Muzi C., Re B., Rossi L., Tiezzi F.. *Animating Multiple Instances in BPMN Collaborations: From Formal Semantics to Tool Support*. In: BPM Conference. Vol. 11080 of LNCS, pp. 83-101. Springer, 2018

and

- **an animator tool**

- Corradini F., Muzi C., Re B., Rossi L., Tiezzi F.. *MIDA: Multiple Instances and Data Animator*. In: BPM Demo. Vol. 2196 of CEUR-WS, pp. 86-90. CEUR, 2018

for BPMN collaborations with multiple instances, data, and message exchange



Research Group



RESEARCH GROUP

PROcess and Services Lab <https://pros.unicam.it>



Mario Corradi
FULL PROFESSOR



Andrea Poli
FULL PROFESSOR



Barbara Br
ASSOCIATE PROFESSOR



Lorenzo Rossi
RESEARCH FELLOW



Massimo Collino De Donato
RESEARCH FELLOW



Marco Pisapicci
RESEARCH FELLOW



Sara Pettinari
PHD STUDENT



Francesco Testi
ASSOCIATE PROFESSOR



Andrea Marchetti
RESEARCH FELLOW



Federico Forneri
RESEARCH FELLOW



Alessandro Marchetti
POSTDOCTORAL RESEARCHER



Caterina Locati
PHD STUDENT



Ivan Compagnò
PHD STUDENT



Vincenzo Nard
PHD STUDENT



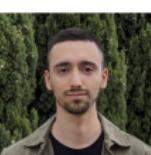
Arianna Teddi
PHD STUDENT



Monica Barfond
PHD STUDENT



Moustapha Senepute
PHD STUDENT



Francesco Cianci
PHD STUDENT



Melania Puletti
PHD STUDENT



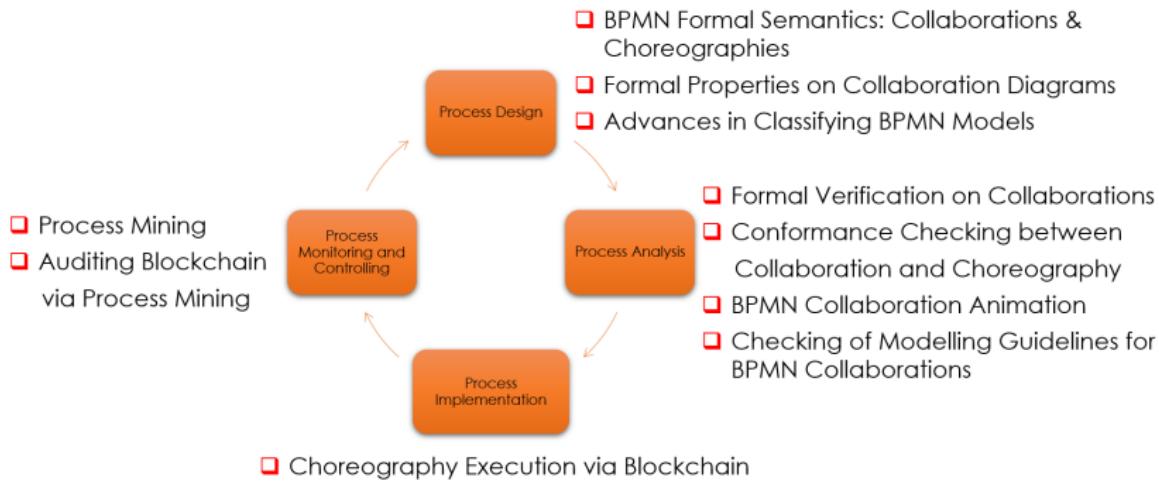
Umar Qureshi
PHD STUDENT



Luca Rischbieter
PHD STUDENT

RESEARCH GROUP

- development of languages and techniques for the modelling and analysis of collaborative systems
- development of process aware information systems and services oriented applications





Francesco Tiezzi and Lorenzo Rossi

francesco.tiezzi@unifi.it - lorenzo.rossi@unicam.it

A formal approach to verifying BPMN collaborations

The BProVe way

Francesco Tiezzi

Università degli Studi di Firenze, Italy



UNIVERSITÀ
DEGLI STUDI
FIRENZE

Guest Lecture of System Integration
Technical University of Denmark, April 24th, 2024

Goal and Motivations

GOAL OF THE LECTURE

In previous lectures, you have seen:

GOAL OF THE LECTURE

In previous lectures, you have seen:

- **Formal models**, to specify systems behaviour

GOAL OF THE LECTURE

In previous lectures, you have seen:

- **Formal models**, to specify systems behaviour
 - Techniques and tools, to **verify properties** over models

GOAL OF THE LECTURE

In previous lectures, you have seen:

- **Formal models**, to specify systems behaviour
 - Techniques and tools, to **verify properties** over models
 - **BPMN models**, to specify Business Processes (BP)

GOAL OF THE LECTURE

In previous lectures, you have seen:

- **Formal models**, to specify systems behaviour
 - Techniques and tools, to **verify properties** over models
 - **BPMN models**, to specify Business Processes (BP)

In this lectures, you will see:

Verification of BP-relevant properties over BPMN collaboration models

MOTIVATIONS

Why do we need to verify Business Processes?

- Nowadays, organizations recognized the importance of having tools to model the different and interrelated aspects governing their behavior
 - BPMN collaboration models, in particular, have acquired increasing relevance for this task
 - also in software development, since they shorten the communication gap between domain experts and IT specialists and permit clarifying the characteristics of software systems needed to provide automatic support for the organization activities
 - There is the need of precisely checking the satisfaction of behavioral properties, to enhance the quality of the BP and the related software
 - Hence, effective formal verification capabilities can foster the full adoption of the BPMN standard

THE INGREDIENTS

The “recipe” for formal verification requires the following ingredients:

- **BPMN notation**, for modelling the system to be analysed
 - **Formal semantics**, associating to each BPMN model a corresponding formal model (LTS in our case)
 - **BP-relevant properties**, defined on top of BPMN models and mapped to logical formulae (LTL in our case)
 - **Verification tool** (MC in our case), automating the verification of the formal property over the formal model



The BPMN notation

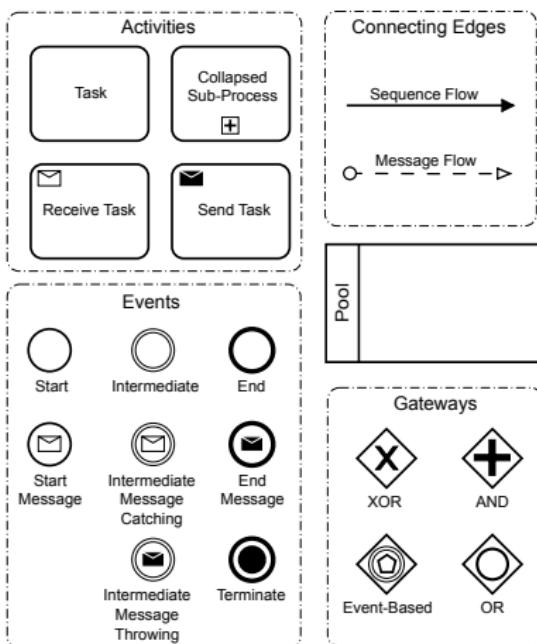
BPMN

- BPMN is an OMG standard (version 2.0 is released on 2011)
- BPMN is currently acquiring a clear predominance among the proposed notations to model BP thanks to:
 - its intuitive and graphical notation that is widely accepted by the industry and the academia
 - the support provided by a broad spectrum of modeling and enacting tools
- You already know the BPMN notation, here we just clarify the **elements supported by the BProVe approach**:
 - we selected a subset of BPMN elements following the pragmatic approach of retaining those features most used in practice¹

¹ Muehlen, M.Z., Recker, J., 2008. How much language is enough? Theoretical and practical use of the business process modeling notation. In: Advanced Information Systems Engineering. In: LNCS, vol. 5074, Springer, pp. 465–479

BPMN COLLABORATION DIAGRAMS

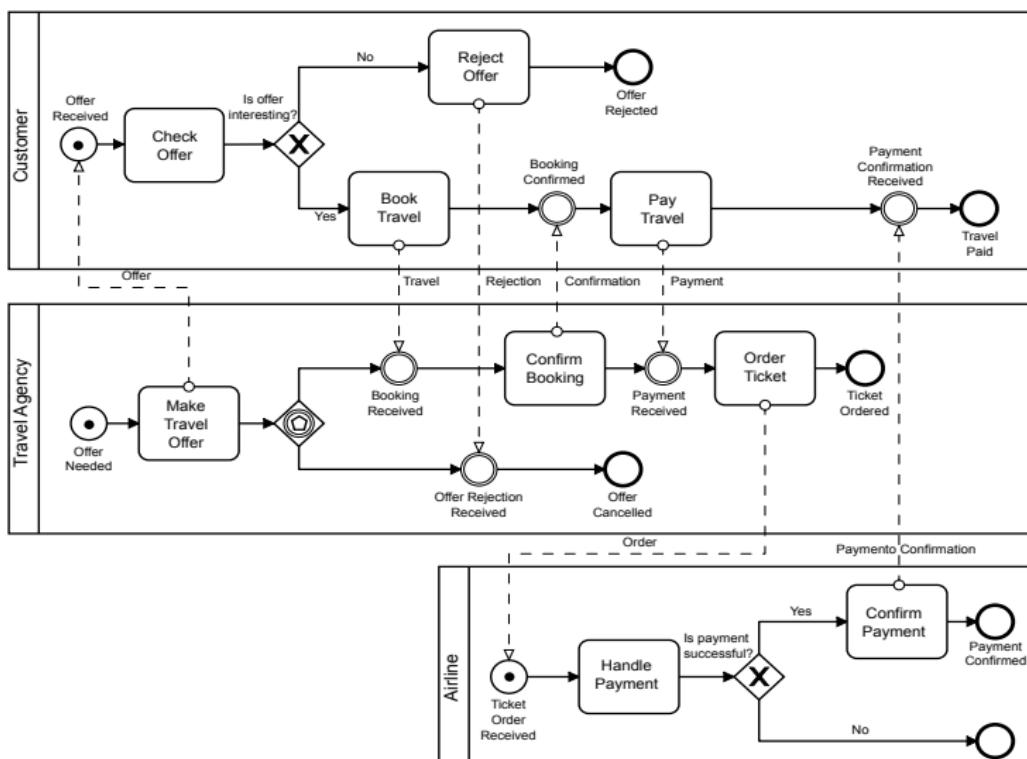
» CONSIDERED ELEMENTS



We support models with an **arbitrary topology**, i.e. we do not impose any syntactical restriction on the usage of the modeling notation (such as well-structuredness)

BPMN COLLABORATION DIAGRAMS

» TRAVEL AGENCY EXAMPLE



BPMN COLLABORATION DIAGRAMS

» INFORMAL SEMANTICS

- The execution of BPMN models is based on the notion of **tokens**, graphically denoted as black dots labeling BPMN elements
 - In the example, the presence of such tokens over the start events enables the execution of the three processes (representing the collaboration's initial status)
 - Tokens traverse the sequence edges of processes and pass through their elements enabling their execution
 - The notation element's specific characteristics define the rules to follow to move, consume and generate tokens

BPMN formalization

FORMALIZATION

- **Goal:** providing BPMN with a **structural operational semantics (SOS)**
 - *SOS style:* we define the behavior of a model in terms of the behavior of its elements, in a syntax-oriented and inductive way
 - The behavior is formalized as a *Labeled Transition System* (LTS)
- **Issue:** The syntax of BPMN 2.0 is given in the standard document by a metamodel in classical UML-style
 - Such syntax is not suitable for providing an operational semantics
 - Thus, we provide an alternative **syntax in BNF-style**

BNF syntax is defined by grammar productions $N ::= A_1 \mid \dots \mid A_n$ where N is a non-terminal symbol and alternatives A_1, \dots, A_n are compositions of terminal and non-terminal symbols

BNF SYNTAX

The BPMN formal syntax is defined by a grammar whose

- *non-terminal symbols*, C and P, represent *Collaborations* and *Processes*
 - *terminal symbols* are the typical graphical elements of a BPMN model,
i.e. pools, events, tasks, gateways, and edges

To obtain a **compositional** definition, each (message/sequence) edge is divided in two parts:

- the part outgoing from the source element
 - the part incoming into the target element

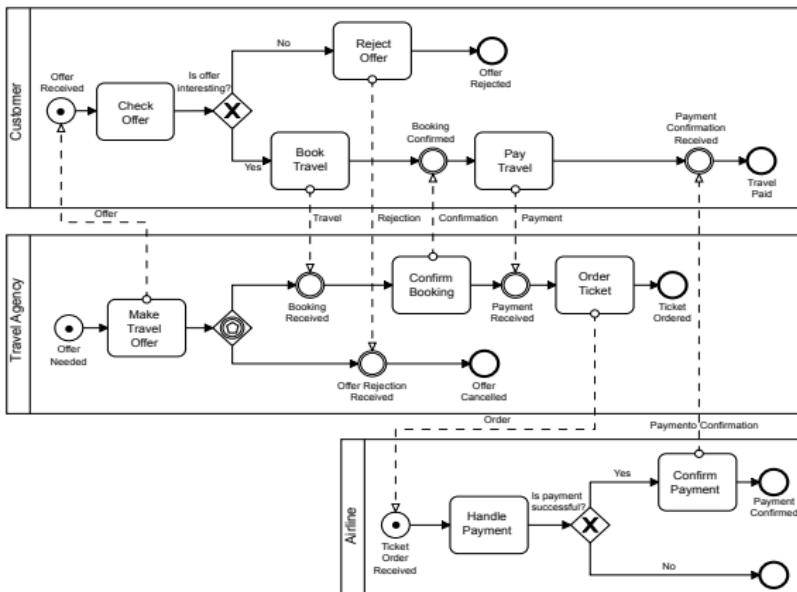
A term of the syntax can be straightforwardly obtained from a BPMN model by decomposing:

- the collaboration in a collection of pools
 - processes in collection of nodes
 - edges in two parts

BNF SYNTAX

» MODEL DECOMPOSITION

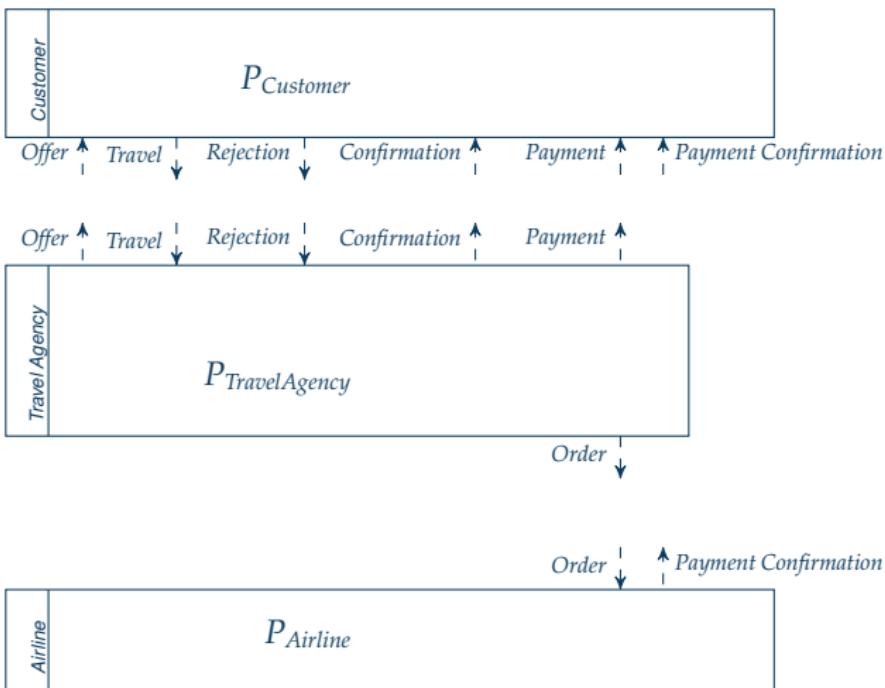
Our travel agency BPMN collaboration model...



BNF SYNTAX

» MODEL DECOMPOSITION

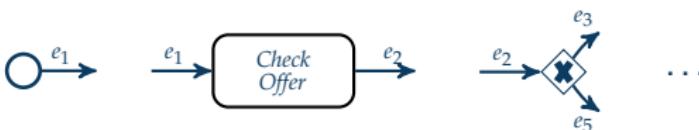
...is decomposed as



BNF SYNTAX

➤ MODEL DECOMPOSITION

...where (an excerpt of) process $P_{Customer}$ is defined as follows:

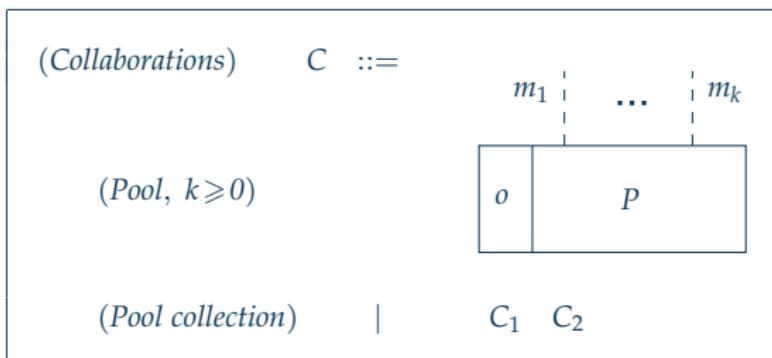


and processes $P_{TravelAgency}$ and $P_{Airline}$ are defined in a similar way

BNF SYNTAX

»GRAMMAR

A BPMN collaboration is rendered as a collection of pools where message edges can connect different pools and each pool contains a process

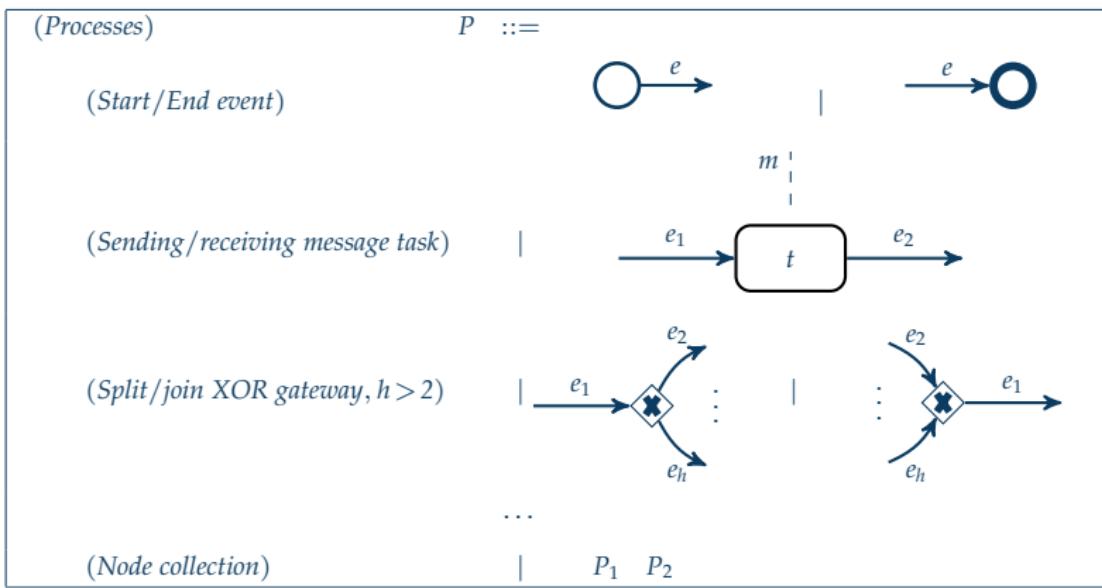


Names: organizations (o), messages (m), sequence edges (e), and tasks (t). We use edges of the form $_ \xrightarrow{m} _$ to denote message edges of the form $_ \xrightarrow{m} _ \rightarrow _$ either incoming into or outgoing from pools/nodes. We identify *collaborations* (resp. *processes*) up to commutativity and associativity of *pool* (resp. *node*) *collection*

BNF SYNTAX

»GRAMMAR (AN EXCERPT)

A BPMN process is rendered as a collection of nodes (events, tasks and gateways), each one with incoming and/or outgoing sequence edges



OPERATIONAL SEMANTICS

The semantics of BPMN collaborations requires at the same time:

- dealing with **workflow aspects** requires synchronising the process elements in order to model the internal execution flow of an organisation
- dealing with **collaborative aspects** requires managing asynchronous message exchanges between different organisations
- dealing with **killing effects** of the terminate end event, which must be confined within a single process (not propagated to other organisations)

To overcome these issues we defined the operational semantics by separating process and collaboration layers

- **process layer** deals with internal interactions among process elements, including the effects of the terminate end event
- **collaboration layer** only focusses on inter-communication between organisations

OPERATIONAL SEMANTICS

We give the semantics of BPMN in terms of **marked collaborations**, i.e. collaboration enriched with message edges, sequence edges, events and tasks marked by (possibly multiple) tokens

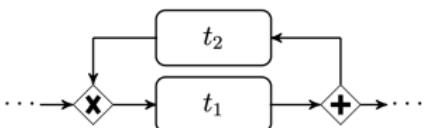
- A **marking** is a distribution of **tokens** over pool message edges and process elements
- This resembles the notions of token and marking in Petri Nets
- Tokens move along the syntax constructs, acting as program counters
- A single token is denoted by \bullet , while multiple tokens labelling a message edge m (resp. sequence edge e) are denoted by $m.n$ (resp. $e.n$), where $n \in \mathbb{N}$ is the token multiplicity

OPERATIONAL SEMANTICS

We give the semantics of BPMN in terms of **marked collaborations**, i.e. collaboration enriched with message edges, sequence edges, events and tasks marked by (possibly multiple) tokens

- A **marking** is a distribution of **tokens** over pool message edges and process elements
- This resembles the notions of token and marking in Petri Nets
- Tokens move along the syntax constructs, acting as program counters
- A single token is denoted by \bullet , while multiple tokens labelling a message edge m (resp. sequence edge e) are denoted by $m.n$ (resp. $e.n$), where $n \in \mathbb{N}$ is the **token multiplicity**

The use of tokens with multiplicity is due to the arbitrary topology of processes



OPERATIONAL SEMANTICS

»PROCESS LAYER

The labeled transition relation of the LTS defining the **semantics of marked processes** is induced by a set of inference rules

- Transitions have the form $P \xrightarrow{\alpha} P'$
- The labels are generated by:

(Actions) $\alpha ::= \tau \quad | \quad !m \quad | \quad ?m$

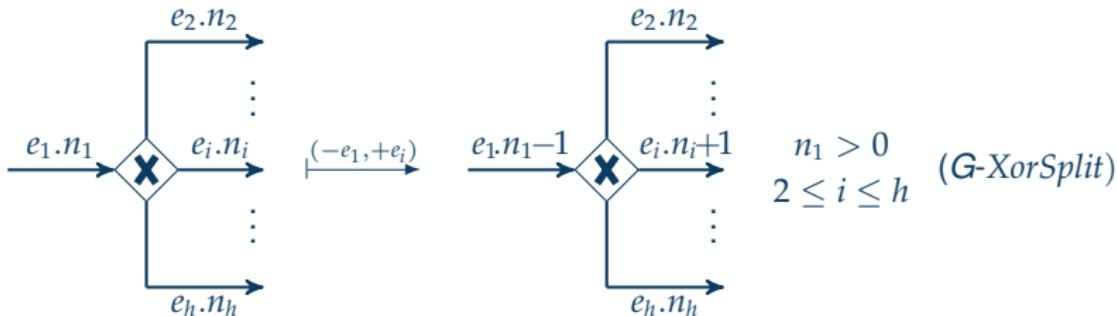
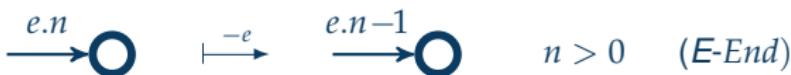
(Internal actions) $\tau ::= \text{running } t \mid \text{completed } t \mid (-\tilde{e}_1, +\tilde{e}_2) \mid \text{kill}$

where $(-\tilde{e}_1, +\tilde{e}_2)$ denotes movement of workflow tokens in the process

OPERATIONAL SEMANTICS

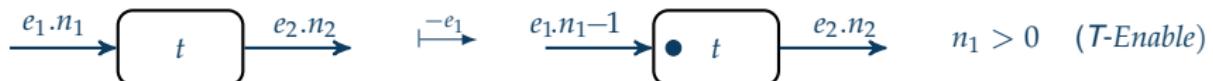
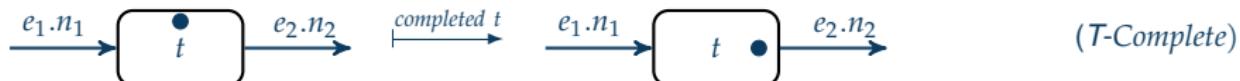
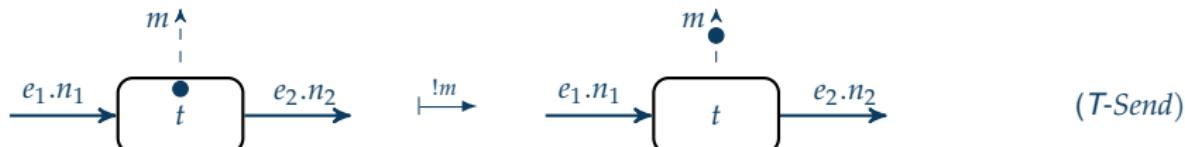
»PROCESS LAYER

Some operational rules:



OPERATIONAL SEMANTICS

»PROCESS LAYER

 \vdash^{-e_1} $e_1.n_1 - 1$ \vdash $e_1.n_1 - 1$ \vdash $e_1.n_1 - 1$ \vdash $n_1 > 0$ $(T\text{-Enable})$  $\vdash^{running \ t}$ $e_1.n_1$ \vdash $e_1.n_1$ \vdash $(T\text{-Running})$  $\vdash^{completed \ t}$ $e_1.n_1$ \vdash $e_1.n_1$ \vdash $(T\text{-Complete})$  \vdash^{+e_2} $e_1.n_1$ \vdash $e_1.n_1$ \vdash $(T\text{-Proceed})$  $\vdash^{!m}$ $e_1.n_1$ \vdash $e_1.n_1$ \vdash $(T\text{-Send})$

OPERATIONAL SEMANTICS

» PROCESS LAYER

$$\frac{P_1 \xrightarrow{(-\tilde{e}_1, +\tilde{e}_2)} P'_1}{P_1 \quad P_2 \xrightarrow{(-\tilde{e}_1, +\tilde{e}_2)} P'_1 \quad P_2 \cdot (-\tilde{e}_1, +\tilde{e}_2)} \quad (N\text{-MarkingUpd})$$

$$\frac{P_1 \xrightarrow{\text{kill}} P'_1}{P_1 \quad P_2 \xrightarrow{\text{kill}} P'_1 \quad P_2\dagger} \quad (N\text{-Kill})$$

$$\frac{P_1 \xrightarrow{\alpha} P'_1 \quad \alpha \notin \{(-\tilde{e}_1, +\tilde{e}_2), \text{kill}\}}{P_1 \quad P_2 \xrightarrow{\alpha} P'_1 \quad P_2} \quad (N\text{-Interleaving})$$

The *marking updating function* $P \cdot (-\tilde{e}_1, +\tilde{e}_2)$ returns a process obtained from P by unmarking (resp. marking) edges in \tilde{e}_1 (resp. \tilde{e}_2). The *killing function* $P \dagger$ returns a process obtained from P by completely unmarking it.

OPERATIONAL SEMANTICS

»COLLABORATION LAYER

The labeled transition relation of the LTS defining the **semantics of marked collaborations** is induced by a set of inference rules

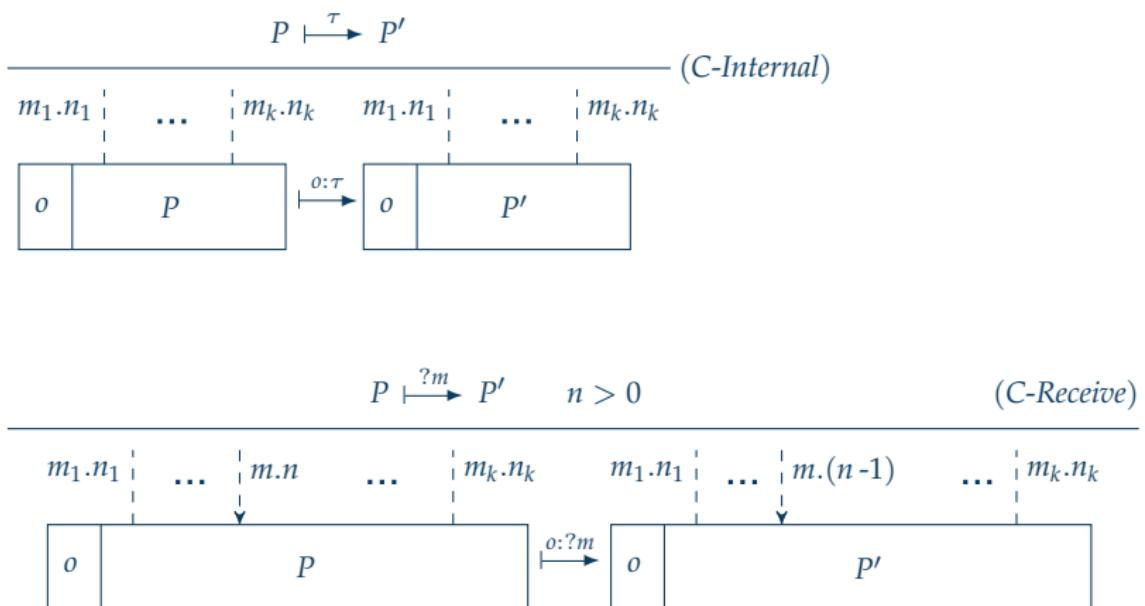
- Transitions have the form $C \xrightarrow{l} C'$
- The labels are generated by:

$$(Labels) \quad l ::= o : \tau \quad | \quad o : ?m \quad | \quad o_1 \rightarrow o_2 : m$$

where internal/receiving/sending actions take place within/between organizations

OPERATIONAL SEMANTICS

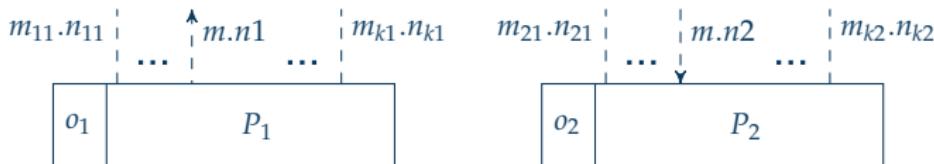
»COLLABORATION LAYER



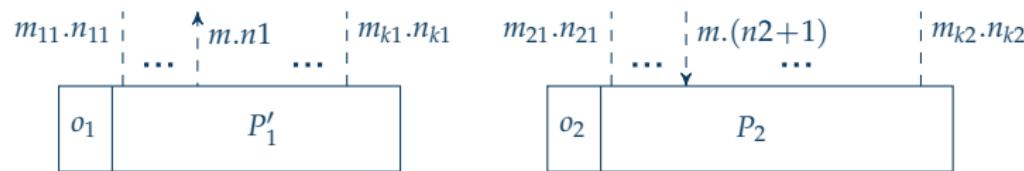
OPERATIONAL SEMANTICS

» COLLABORATION LAYER

$$P_1 \xrightarrow{!m} P'_1 \quad (C-Deliver)$$



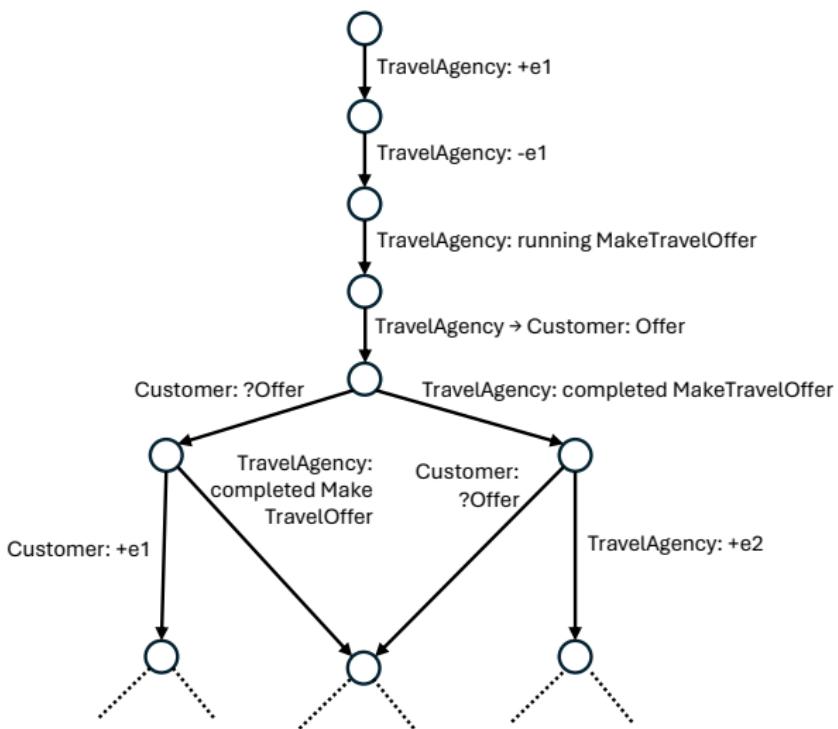
$o_1 \rightarrow o_2 : m$



$$\frac{C_1 \xrightarrow{l} C'_1}{C_1 \quad C_2 \quad \xrightarrow{l} \quad C'_1 \quad C_2} \quad (C\text{-Interleaving})$$

OPERATIONAL SEMANTICS

» TRAVEL AGENCY EXAMPLE: LTS



BP-relevant properties

PROPERTIES

- There is a rich literature about properties of process models
- We consider both the internal characteristics of a single process in a collaboration and the whole collaboration
- We refer to well-established in the business process domain:
safeness and **soundness**
- We also support **ad-hoc properties** specifically defined for given application scenarios

PROPERTIES

»SAFENESS

A BPMN process model is **safe** if during its execution no more than one token occurs along the same sequence edge

- The satisfaction of this property is generally considered a minimum guarantee to avoid unexpected behaviours
- This definition is inspired by the Petri Net formalism, where safeness means that a Petri Net does not have more than one token at each place in all reachable markings

Safeness of processes scales to collaborations, saying that no more than one token occurs on the same sequence edge during a collaboration execution

PROPERTIES

»SOUNDNESS

Soundness can be described as the combination of three basic properties concerning the behaviour of a BPMN process model:

- (i) *Option to Complete*: the running process must eventually complete
- (ii) *Proper Completion*: at the moment of completion, each token of the process must be in a different end event
- (iii) *No dead activities*: any activity can be executed in at least one process execution

Soundness is extended to process collaborations, involving the whole collaboration execution and requiring that all sent messages are properly received

PROPERTIES

»AD-HOC PROPERTIES

Besides system-independent properties, we also consider **application-dependent properties** specifically defined for the given application scenario

Examples concerning the travel agency scenario.

- (P1) Does the start of the *Confirm Booking* task in the *TravelAgency* pool implies that the same task will sooner or later complete?
- (P2) Does the completion of a specified task in the *Airline* pool, say *HandlePayment*, implies the completion of another task in the same pool, say *ConfirmPayment*?
- (P3) If the *Customer* has sent the payment to the *TravelAgency*, may it happen that the corresponding confirmation, by the *Airline*, is never received?

PROPERTIES

» FORMALISATION

We formalize properties in terms of **Linear Temporal Logic (LTL)** formulae

Soundness and safeness properties in LTL:

Property	LTL formula
Soundness (i): Option to Complete	[] processStart(orgName) —> <>processCompletion(orgName)
Soundness (ii): Proper Completion	[] ~ noProperCompletion(orgName)
Soundness (iii): No Dead Activities	[] ~ aTaskRunning(taskName)
Safeness	[] safeState(orgName)

LTL operators:

- [] ϕ holds if ϕ *globally* holds;
- $<> \phi$ holds if ϕ *eventually* holds;
- $\phi \rightarrow \varphi$ corresponds to the *implication*
- $\sim \phi$ corresponds to the *negation*

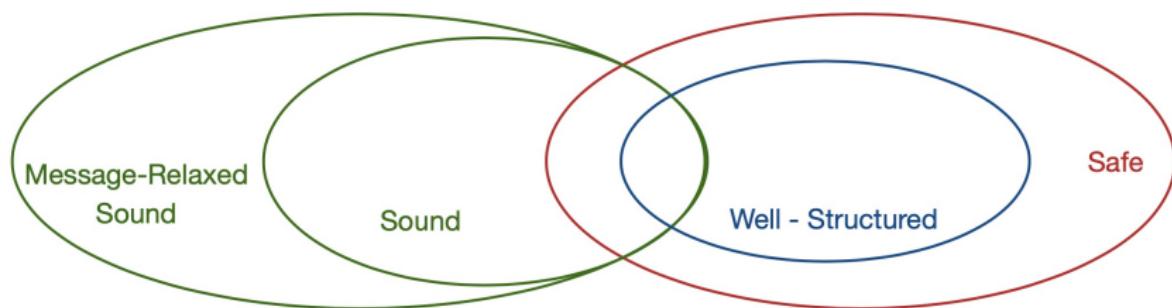
To ease the definition of properties we have defined some recurrent predicates representing high-level BPMN-related concepts

Using operators and predicates we can “cook” ad-hoc properties

PROPERTIES

»CLASSIFICATION OF BPMN COLLABORATIONS

System-independent properties permitted to define this **classification of BPMN collaborations**²



²Corradini, F., Morichetta, A., Muzi, C., Re, B., Tiezzi, F. Well-structuredness, safeness and soundness: A formal classification of BPMN collaborations. Journal of Logical and Algebraic Methods in Programming 119, 2021

From theory to practice

MODEL CHECKING

Property Verification enables to check the properties of interest detecting behavioural issues of the BPMN collaboration model

- This check, named **Model Checking** (MC) is based on:
the model behaviour → its semantics → its LTS
 - MC consists of systematically exploring the LTS to establish if a logical formula formally holds
 - This exploration is error-prone and unfeasible with a manual approach
an automated approach is needed → a **model checker tool**

MODEL CHECKING

Property Verification enables to check the properties property of interest detecting behavioural issues of the BPMN collaboration model

- This check, named **Model Checking** (MC) is based on:
the model behaviour → its semantics → its LTS
- MC consists of systematically exploring the LTS to establish if a logical formula formally holds
- This exploration is error-prone and unfeasible with a manual approach
an automated approach is needed → a **model checker tool**

To support property verification of BPMN collaborations we developed



TOWARDS AUTOMATIC VERIFICATION

To enable automatic verification of BPMN collaborations, as a first step we have **implemented the BPMN formal semantics** in the form of an executable interpreter in **MAUDE**

Why MAUDE as language for implementing our BPMN semantics?

- MAUDE permits to code rules of the operational semantics simply as rewriting rules with a one-to-one correspondence
- This ensures the faithfulness of the MAUDE implementation of the semantics with respect to its formal definition
- MAUDE comes equipped with a LTL model checker

TOWARDS AUTOMATIC VERIFICATION » BPMN SYNTAX IN MAUDE

Each BPMN element is declared as a MAUDE operation

- For example, the task element is rendered as:

```
op task( , , , ) : Status Edge Edge TaskName -> ProcElement
```

A sketch of the Travekl Agency collaboration is:

```

collaboration(
  pool( "Travel Agency" ,
    proc( start( enabled, "e1".0 ) |
      taskSnd ( disabled, "e1".0, "e2".0, "Offer".msg 0, "Make Travel Offer" ) |
      ...
    ) , in: ... , out: "Offer" .msg 0 ... ) |

  pool( "Customer" ,
    proc( startRcv ( enabled , " e2 " . 0 , "Offer".msg 0) |
      ...
    ) , in: "Offer" .msg 0 andmsg IMsgSet , out: OMsgSet ) |

  pool( "Airline" , proc( ... ) , in: ... , out: ... )
).

```

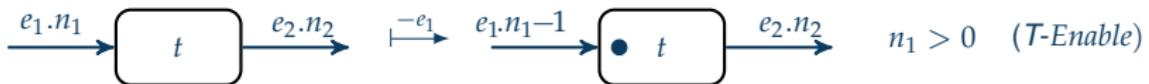
TOWARDS AUTOMATIC VERIFICATION » BPMN SEMANTICS IN MAUDE

Each rule of the BPMN semantics is rendered as a MAUDE rewriting rule.

- Rewriting rules are exhaustively applied by pattern matching on each generated state, starting from the initial one, until no new states can be generated
 - A rewriting rule has the following form:

crl [Label] :: Term-1 => Term-2 **if** Condition(s) .

- Example: rule for enabling a task



```

crl [T-Enable] :
task( disabled , IENName . IEToken , OENName . OEToken , TName )
=>
{tUpd(IENName . IEToken , emptyEdgeSet)}
task( enabled , IENName . decreaseToken( IEToken ) , OENName . OEToken , TName )
if IEToken > 0 .

```

VERIFICATION

The verification featured by BProVe is based on the state-space exploration capabilities offered by our BPMN interpreter implemented in the MAUDE language

- Given a state of the collaboration under analysis, the interpreter permits to compute its set of one-step next states, i.e. all states reachable from the given state in one execution step, by applying the rewriting rules defining the BPMN semantics
 - This allows the MAUDE interpreter to generate the LTS of the collaboration model
 - The MAUDE LTL model checker derives and explores on-the-fly the state-space of the LTS

VERIFICATION

» STATISTICAL MODEL CHECKING

The MAUDE LTL model checker allows for an *exhaustive exploration* of the model reachable states and provides a reliable response in the case of systems with a finite state space

VERIFICATION

» STATISTICAL MODEL CHECKING

The MAUDE LTL model checker allows for an *exhaustive exploration* of the model reachable states and provides a reliable response in the case of systems with a finite state space

However, model checking techniques are known to be affected by the state-space explosion problem

- if a model generates many states, model checking cannot complete in a reasonable amount of time, or may fail due to high memory requirements
 - the design of models with large state spaces is not uncommon in the context of business process modelling

VERIFICATION

» STATISTICAL MODEL CHECKING

The MAUDE LTL model checker allows for an *exhaustive exploration* of the model reachable states and provides a reliable response in the case of systems with a finite state space

However, model checking techniques are known to be affected by the state-space explosion problem

- if a model generates many states, model checking cannot complete in a reasonable amount of time, or may fail due to high memory requirements
 - the design of models with large state spaces is not uncommon in the context of business process modelling

Therefore, to be able to provide a valid response in those cases, we resort to a simulation-based technique known as **statistical model checking** (SMC)

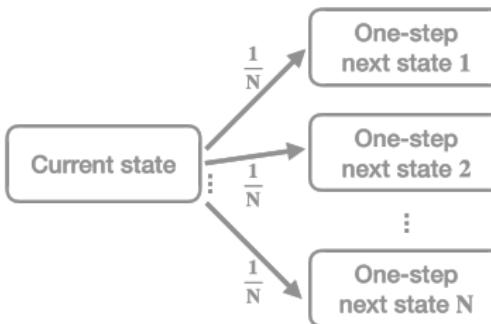
- SMC runs independent executions as long as a required level of statistical accuracy has not been reached, so to provide statistical evidence on the satisfaction or violation of a property

VERIFICATION

»STATISTICAL MODEL CHECKING

The SMC technique we use is supported by MultiVeStA

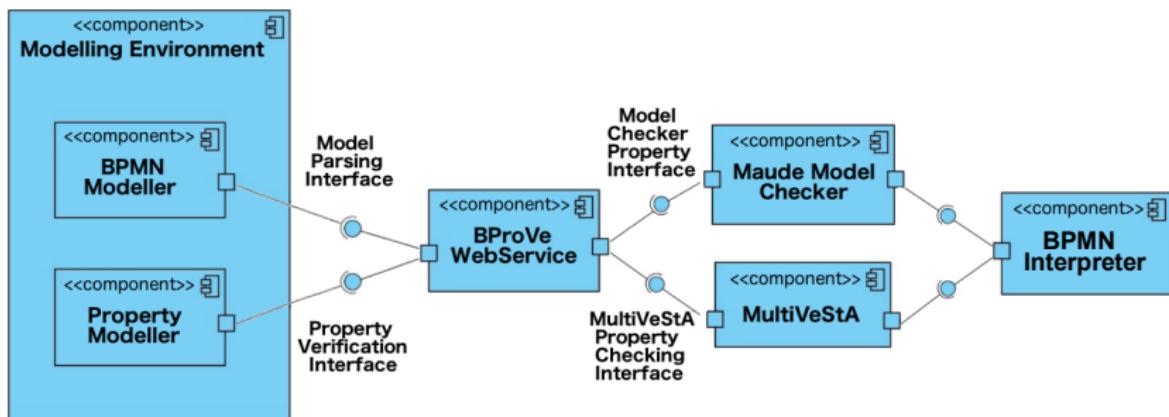
- It associates a discrete uniform probability distribution to the states that can be reached in one step from a specific state, allowing for probabilistic simulations



- This approach to path selection allows to introduce a form of fairness that is not considered when using the MAUDE LTL model checker
- The MultiVeStA's query specification language is MultiQuaTEx

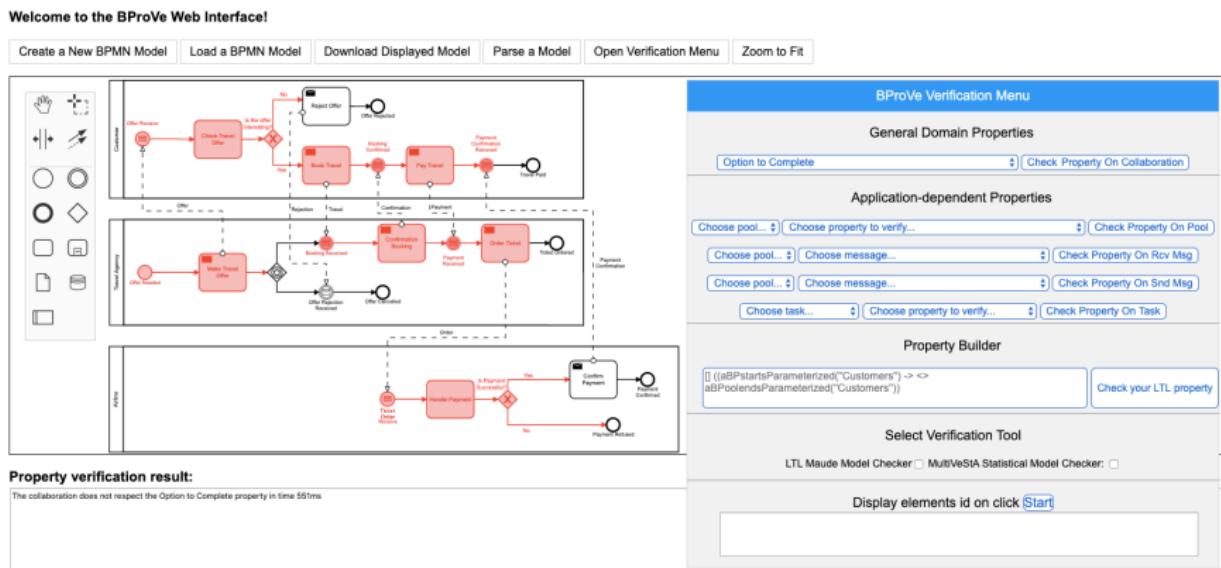
VERIFICATION ➤ THE BProVE TOOL

Component diagram of the BProVe tool-chain:



VERIFICATION »THE BProVE TOOL

BProVe web interface:



Demo

DEMO

- BProVe web app available at:

<https://pros.unicam.it/bprove/bprove-web-interface>

- Local installation of BProVe via Docker:

- Install and run Docker in your system:

<https://www.docker.com/>

- Download the BProVe docker image:

```
docker pull proslab/bprove
```

- Create a container:

```
docker run -itd -p 8080:8080 --name bprove proslab/bprove
```

- Run and stop a container:

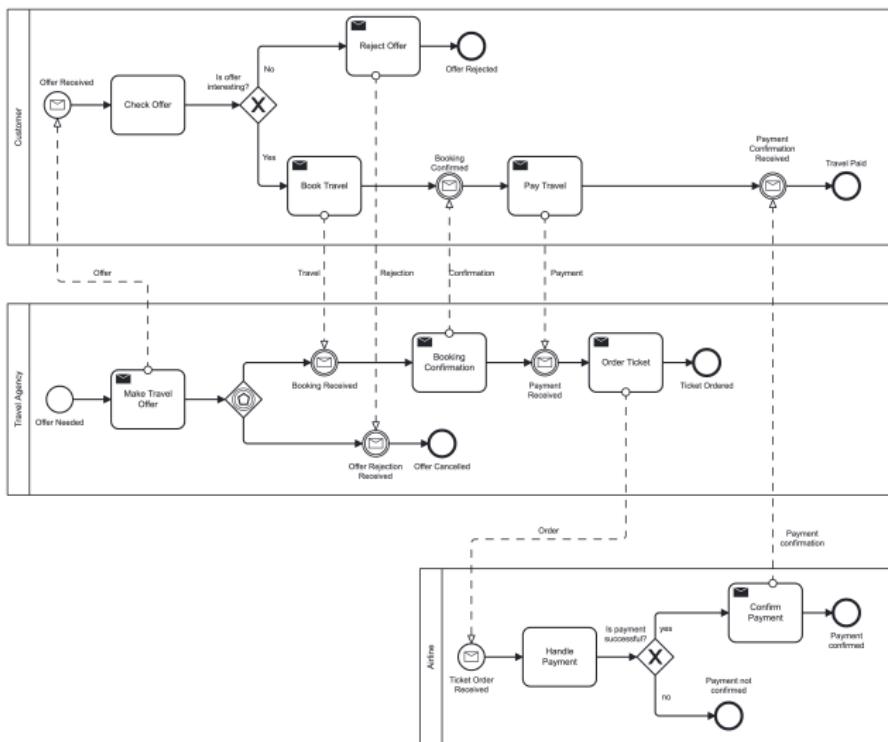
```
docker start bprove / docker stop bprove
```

- Access the app via a browser:

<http://localhost:8080/BProVe>

DEMO

BPMN travel agency model:



DEMO

Analysis of the Travel Agency scenario:

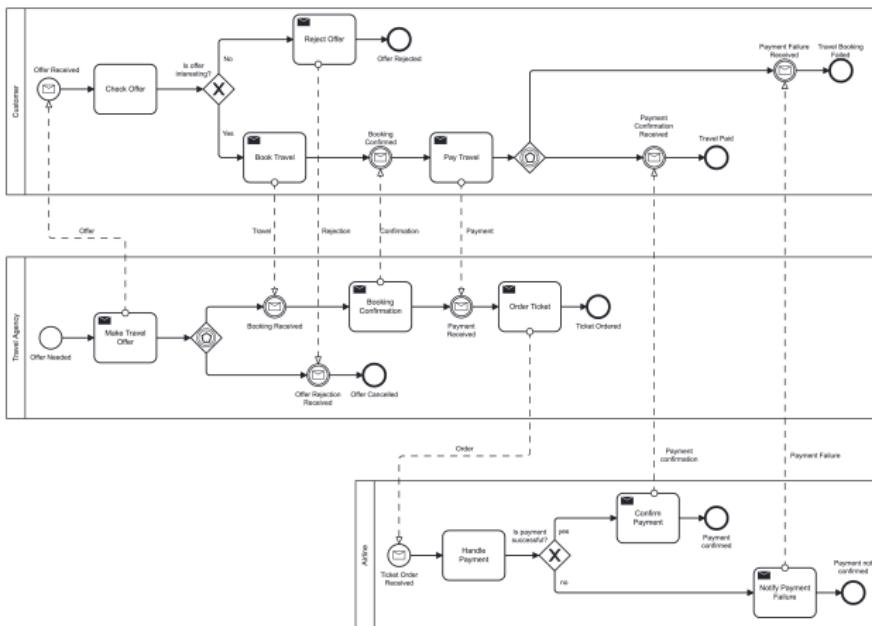
- **Safeness:** TRUE
 - **Soundness:** FALSE
 - Option to Complete: FALSE
 - Proper Completion: TRUE
 - No Dead Activities: TRUE

EXERCISE 1

- Fix the Travel Agency model to make it sound

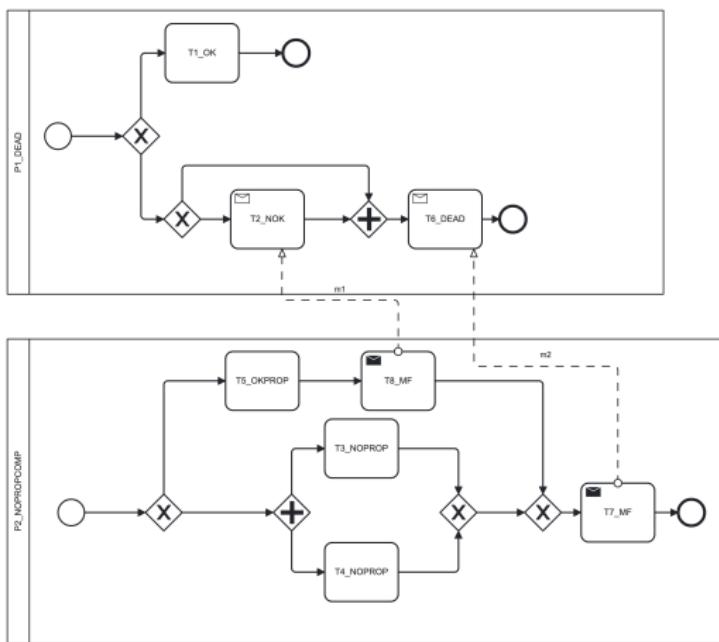
EXERCISE 1

- Fix the Travel Agency model to make it sound
 - Solution:



EXERCISE 2

Consider the BPMN collaboration loaded at the start of BProVe:

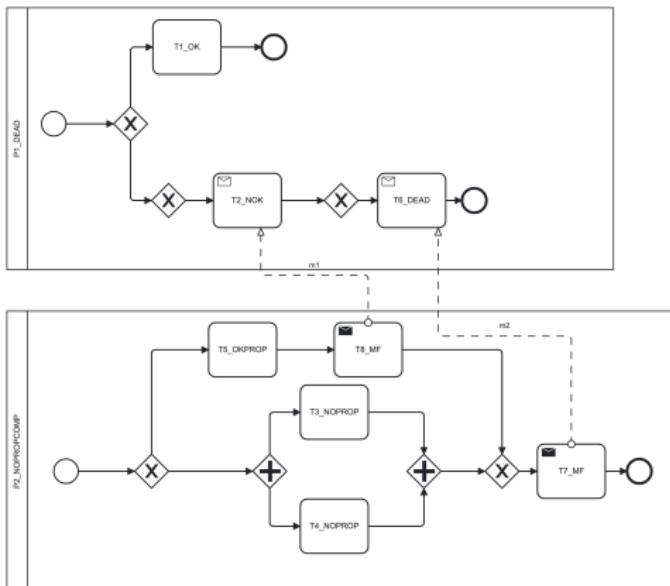


EXERCISE 2

- Check Safety and Soundness
 - Fix the model to make it safe (hint: the designer used a wrong gateway)
 - Fix the model to make it safe and without dead activities

EXERCISE 2

- Check Safety and Soundness
 - Fix the model to make it safe (hint: the designer used a wrong gateway)
 - Fix the model to make it safe and without dead activities
 - Solution:



Thank you!

For further details about BProVe, visit
<https://pros.unicam.it/bprove/>