System Design and Engineering (2IMP30)

WEEK6 - SYSML – STATE MACHINE DIAGRAM

- Ion Barosan – i.barosan@tue.nl
- Mathematics and Computer Science – SET

TU/e EINDHOVEN UNIVERSITY OF TECHNOLOGY

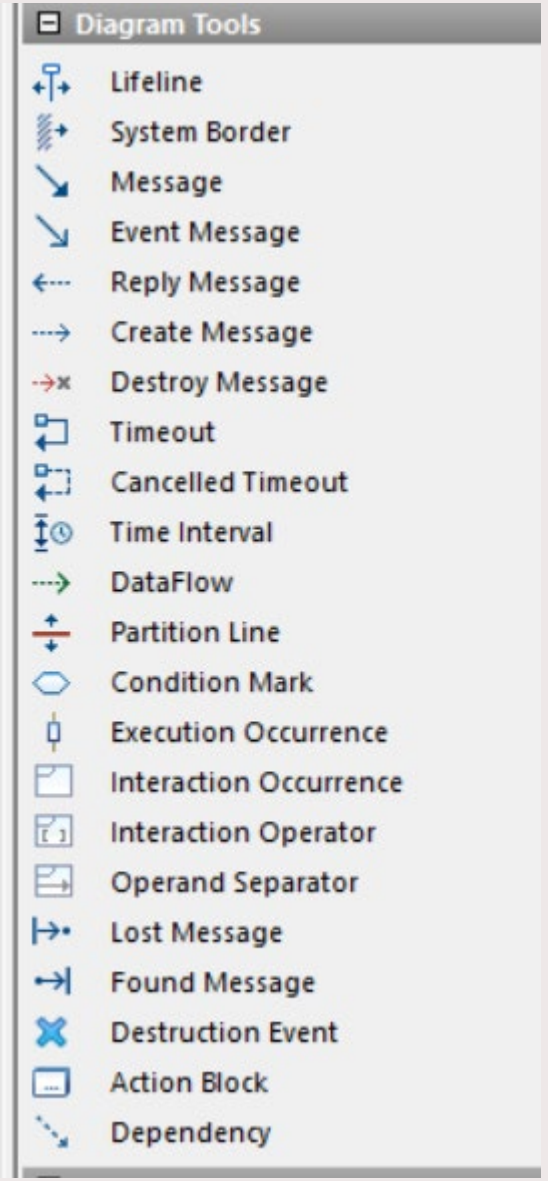
Content

1. Questions – Sequence Diagram
2. SysML State Machine Diagrams
3. Hands-On Light System

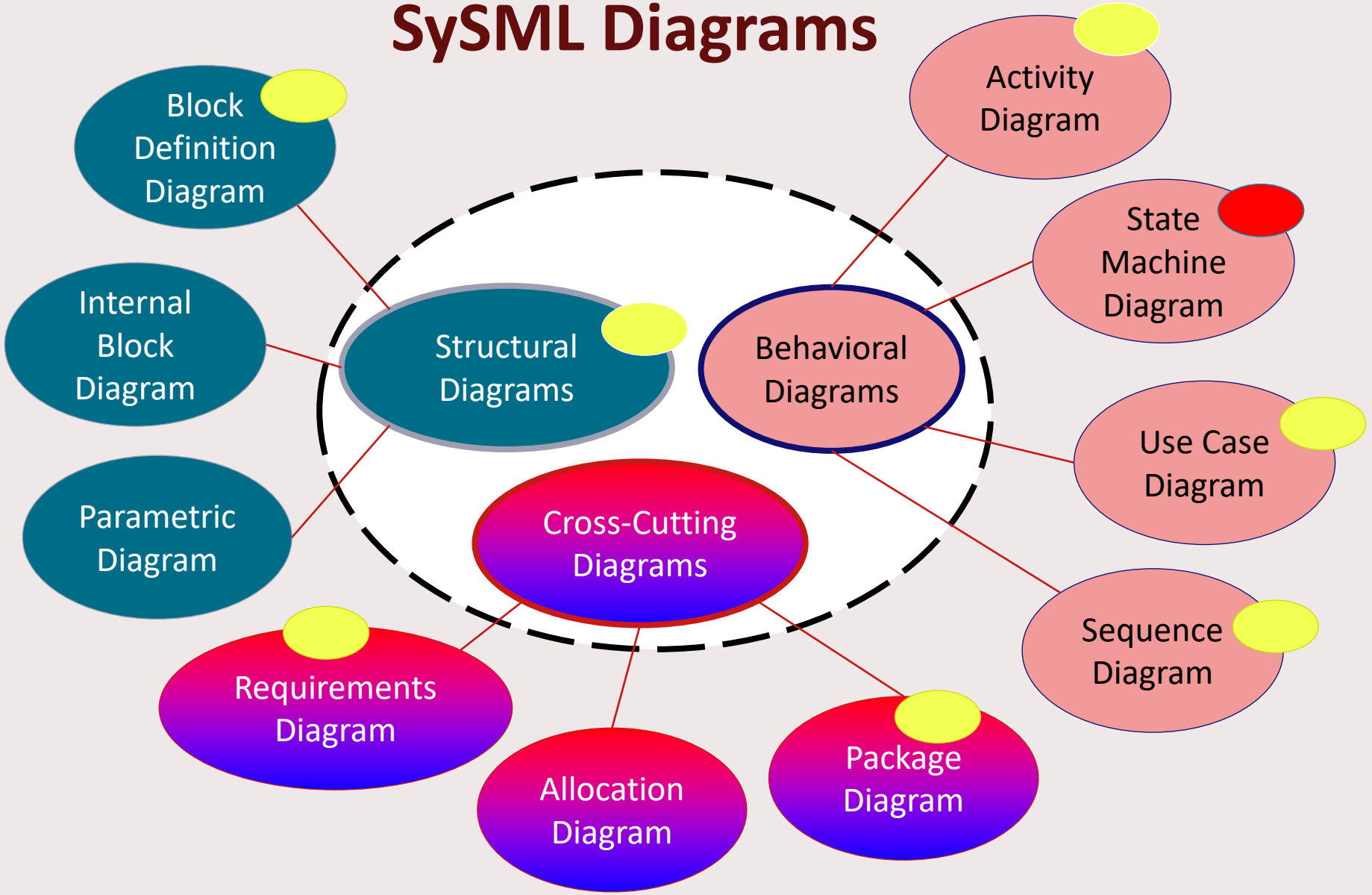
QUESTIONS [1]

- What can we model with a Sequence Diagram?

- Time?
- Parallel Data / Control Flow?
- Sequential Data / Control Flow?
- Objects behavior?
- How can we allocate data/control flow to structure?
- Which are the message types used in SEQ Diagram?
- System Border?



SySML Diagrams



Contens

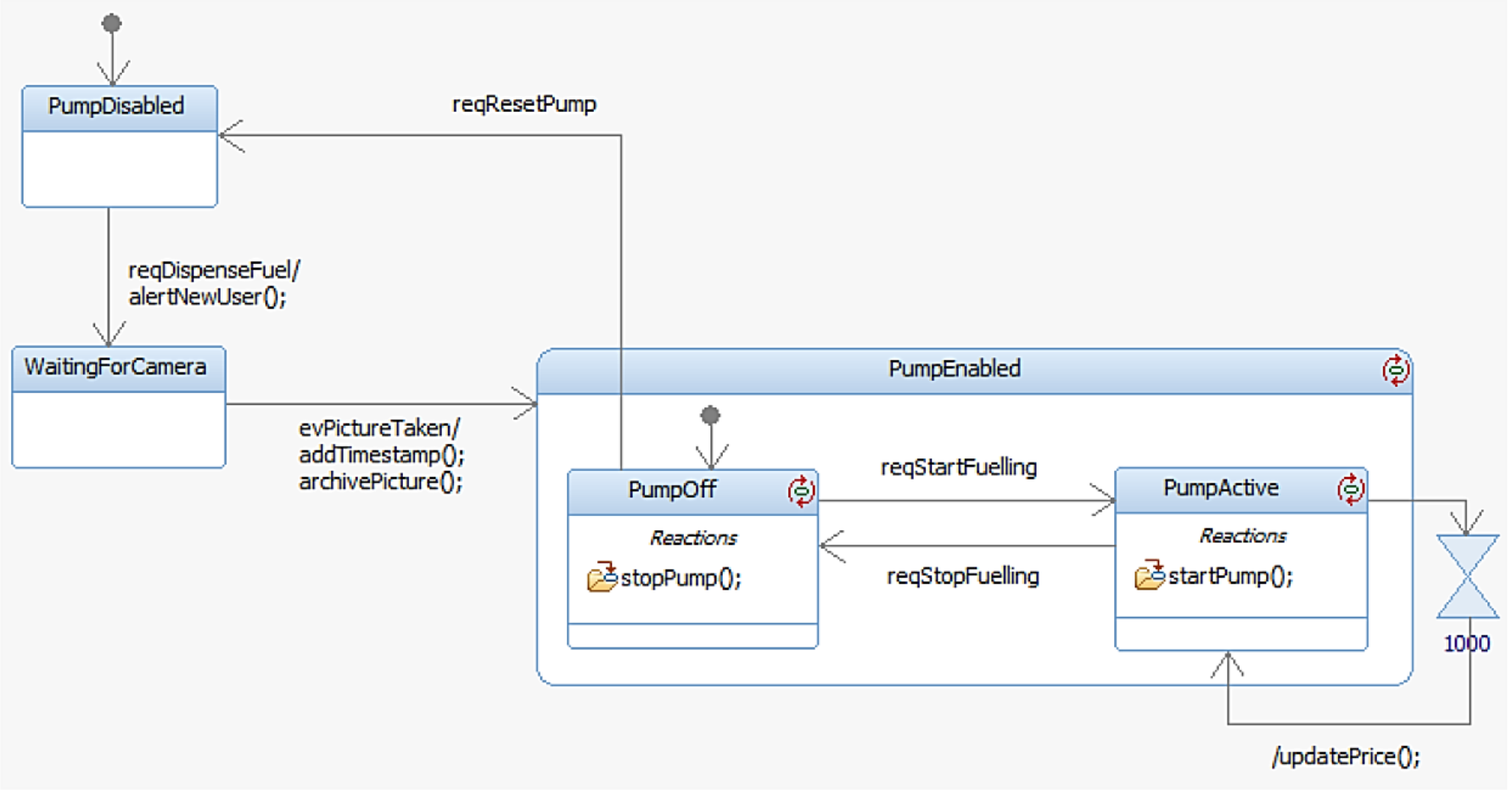
- State Machine Diagram
- Components
- Overview
- Purpose
- Relationships
- Use case textual description

STATE MACHINE DIAGRAM

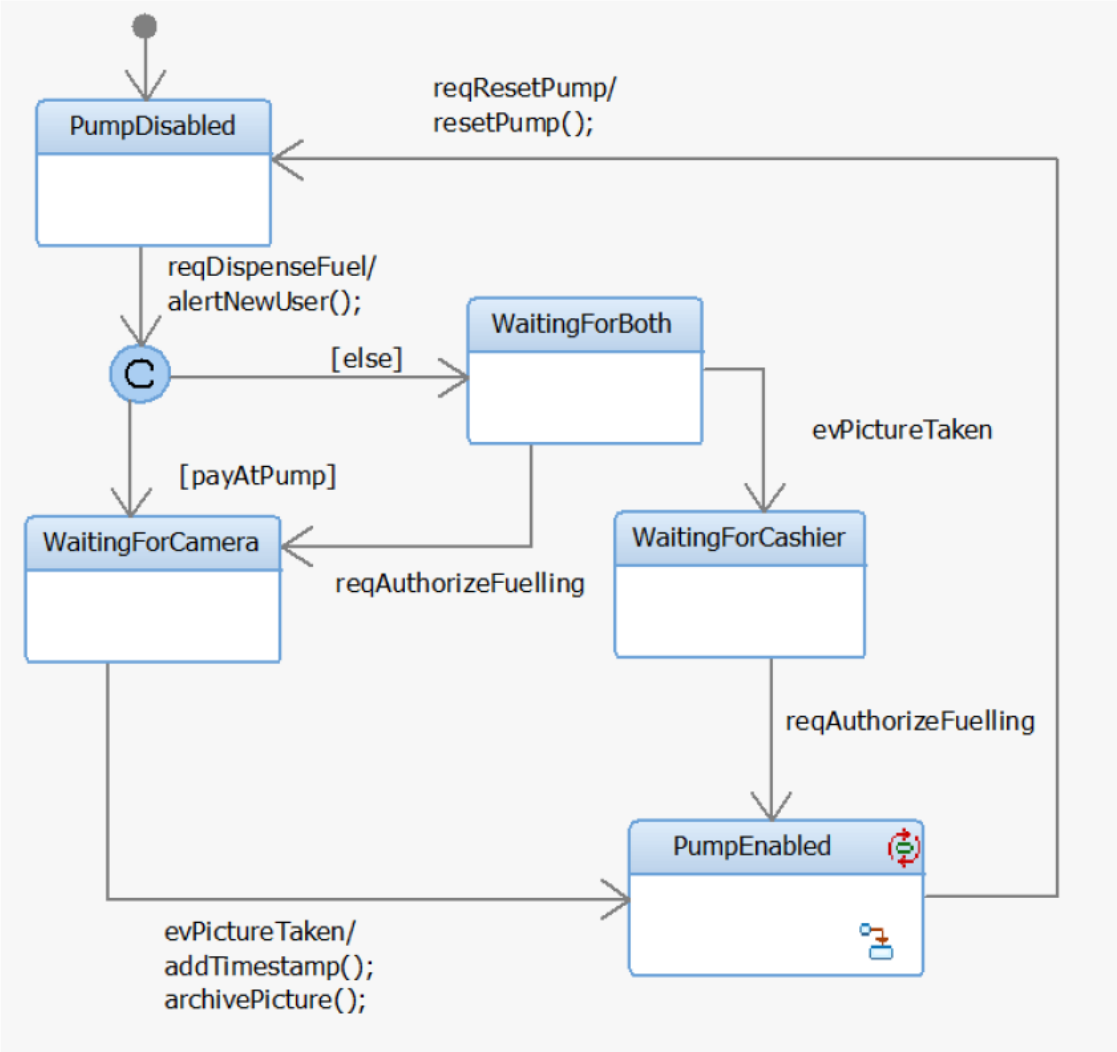
Contents

- **State Machine Diagram**
  - Components
  - Overview
  - Purpose
  - Relationships
  - Use case textual description

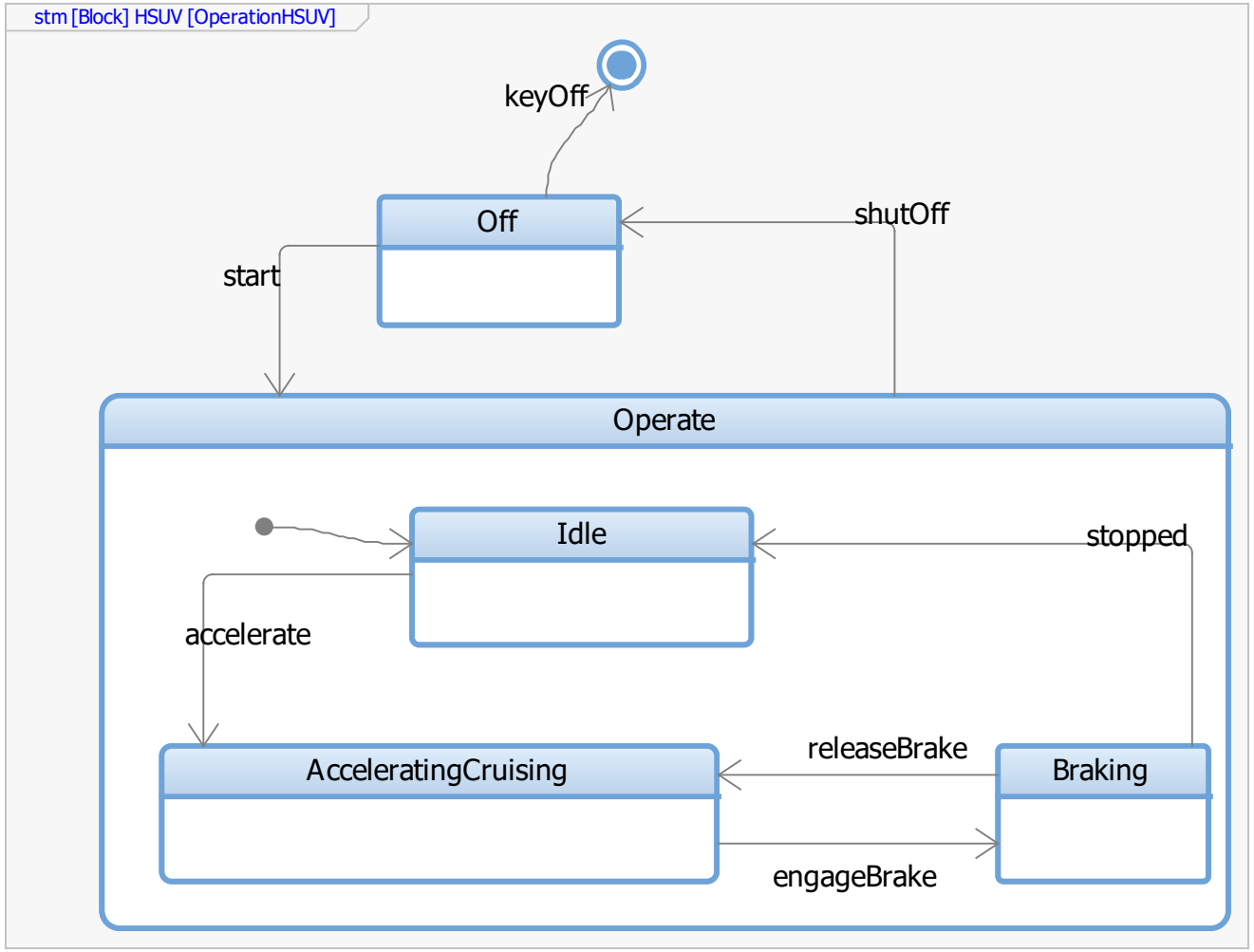
The system behavior when the motorist pays at the pump



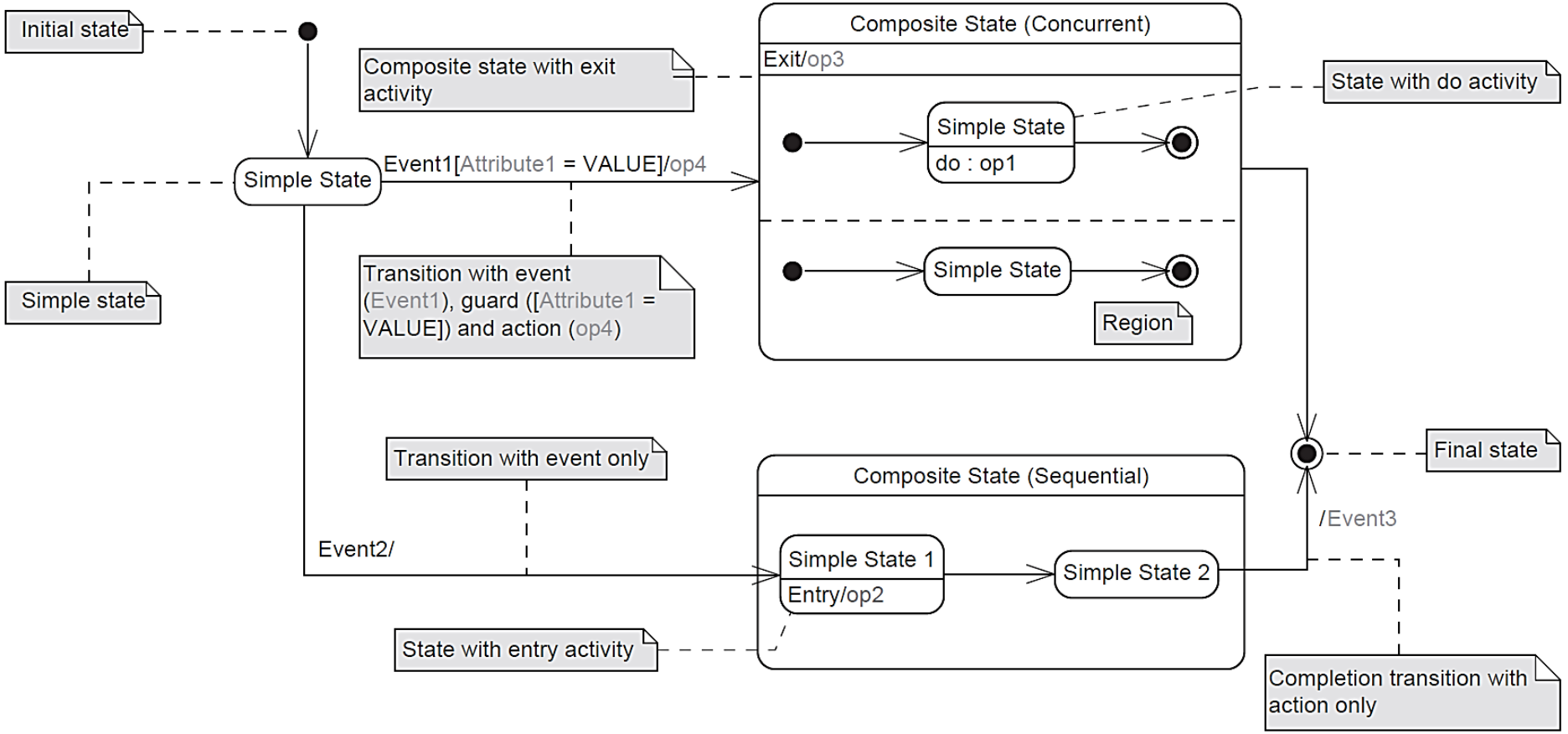
The system behavior when the motorist pays in the booth.



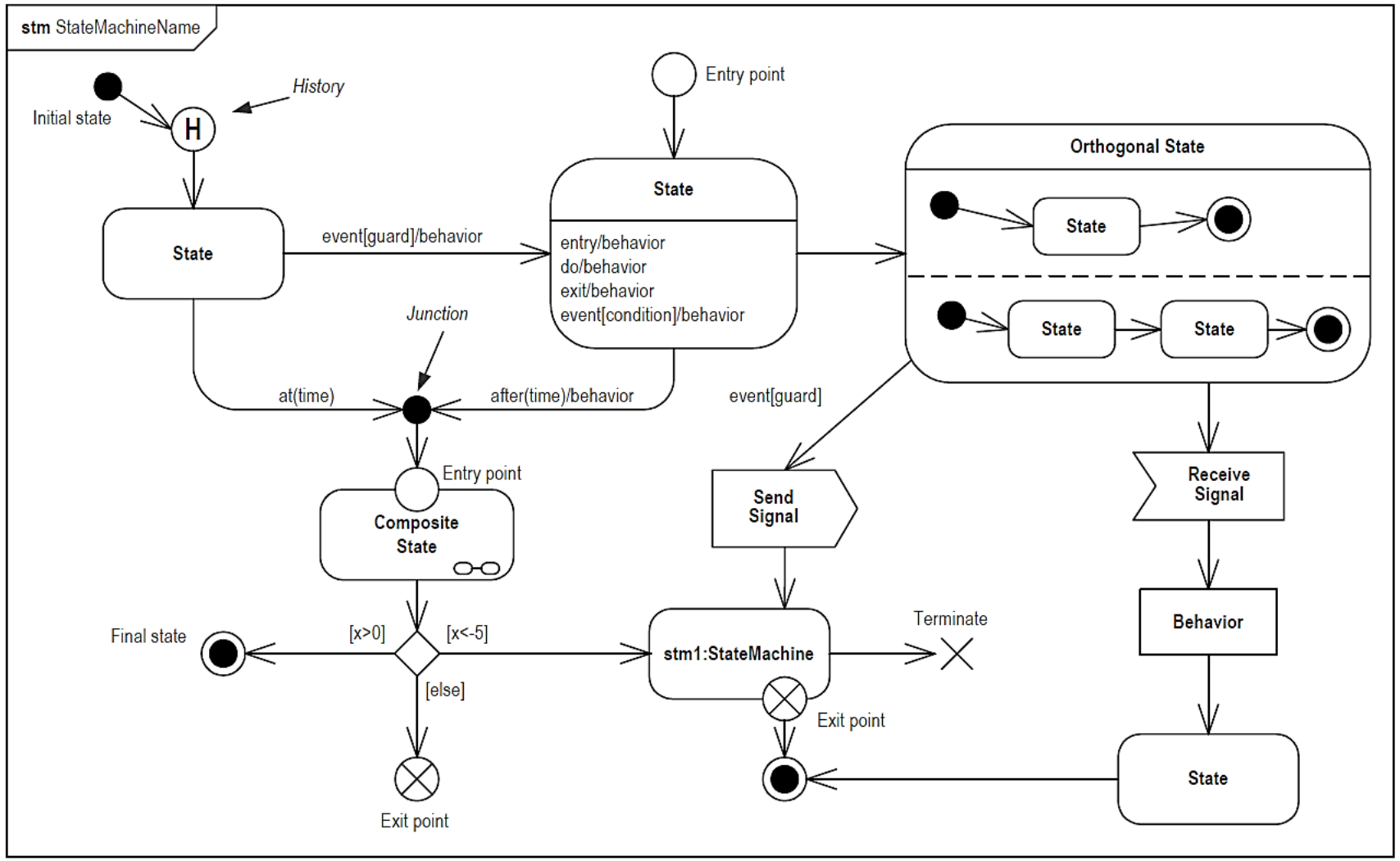
HSUV Operations



State Machine Diagram Notation[1/2]



State Machine Diagram Notation[2/2]

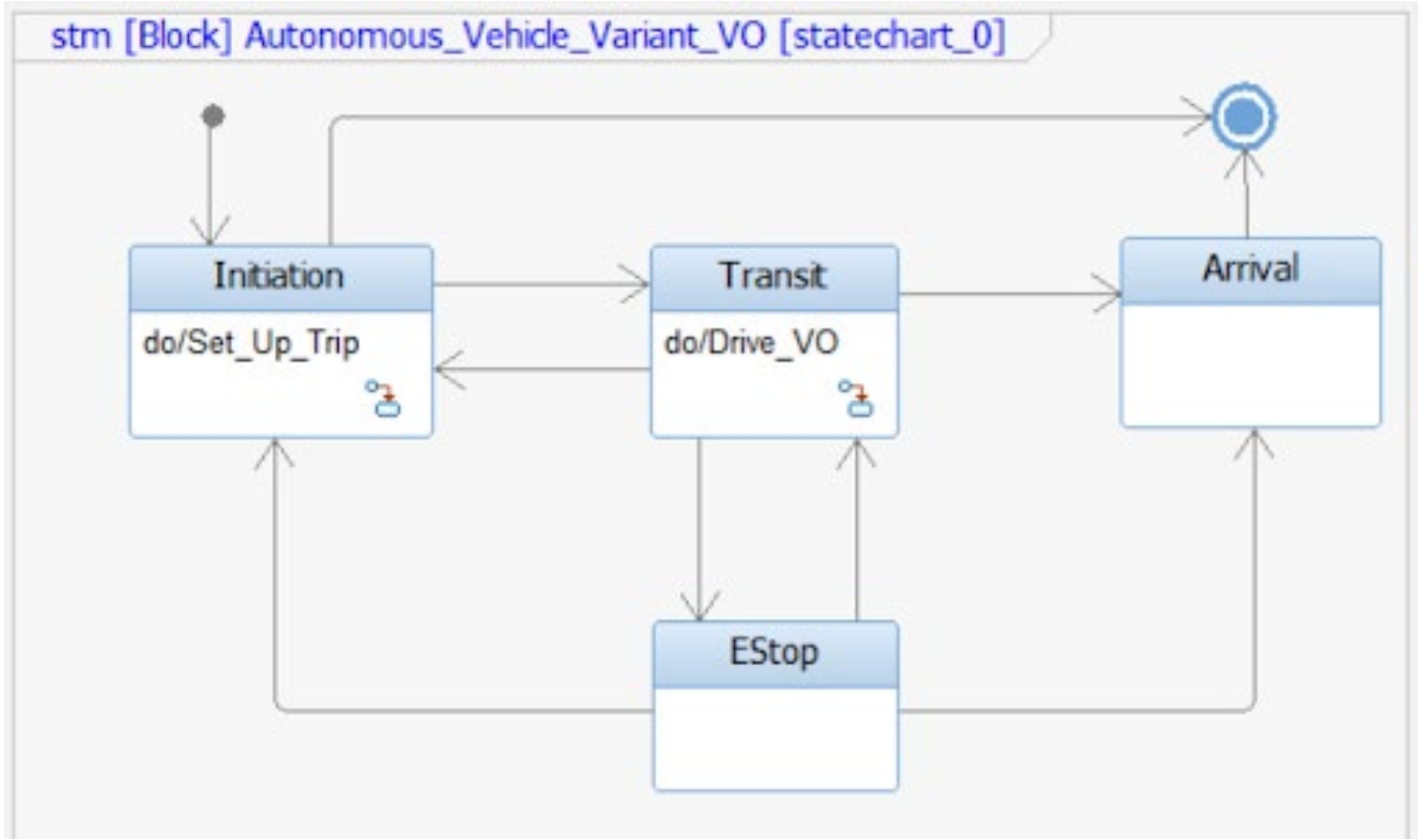


State machines

- **A State machine** shows the state-based behavior - typically of a Block
- **Blocks** have **Operations** that describe their ***behavioral*** possibilities in a primitive way.
- **State machines** can be EXECUTED: their dynamic behavior is shown graphically.

Purpose

- **The behavior displayed** on a state machine diagram most often ***represents a block's classifier behavior***
- **A state machine diagram** is well suited to serve as a *detailed design artifact* (that is, an input into development).
- **A state machine diagram** is *a precise and unambiguous specification of behavior*.
- **Code Generation!**



When Should You Create a State Machine Diagram?

Create a state machine diagram *to describe the behavior of a block at any level in the system hierarchy* such as:

- *the system of interest itself*
- *a subsystem*
- *a single component*

You can *potentially create a state machine diagram at any point in the system life cycle*.

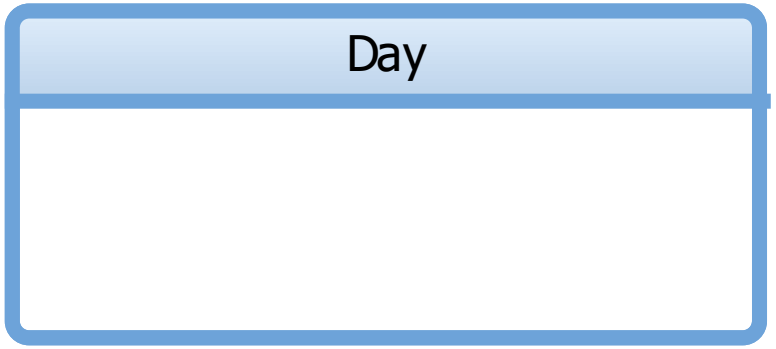
The State Machine Diagram

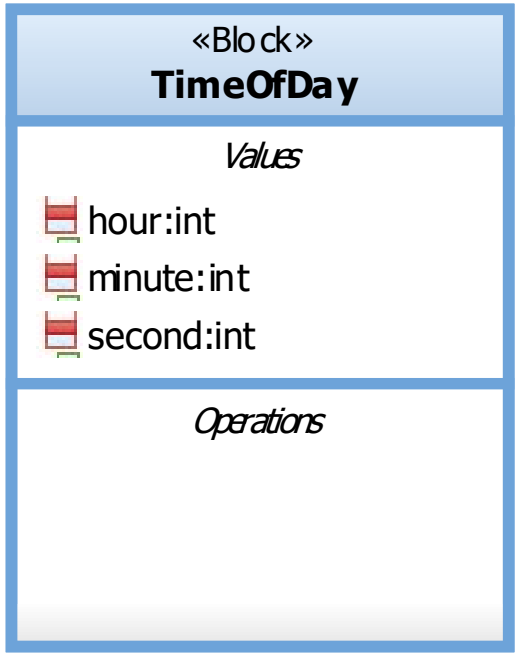
- **A system** is always in a *state that abstracts a combination of values given* in the system
- *Events arriving* at the system lead to reactions—depending on the state—that *change values* and *results in new states*.
  - **A state machine** diagram *contains state machines* with *states and state transitions that are triggered by events*.
- **The semantics** of the state model is defined in such *great detail in UML that it can be executed*.

WHAT is a State Machine ?

State [1/4]

- **A state** represents ***a set of value combinations*** for the underlying element:
  - It has a ***name***, and may have an ***internal behavior*** that is ***executed based on defined events***.
- Ex: The state *Day* stands for all value combinations of the attributes *hour* , *minute* , and *second* , between the two events, *sunrise* and *sunset*

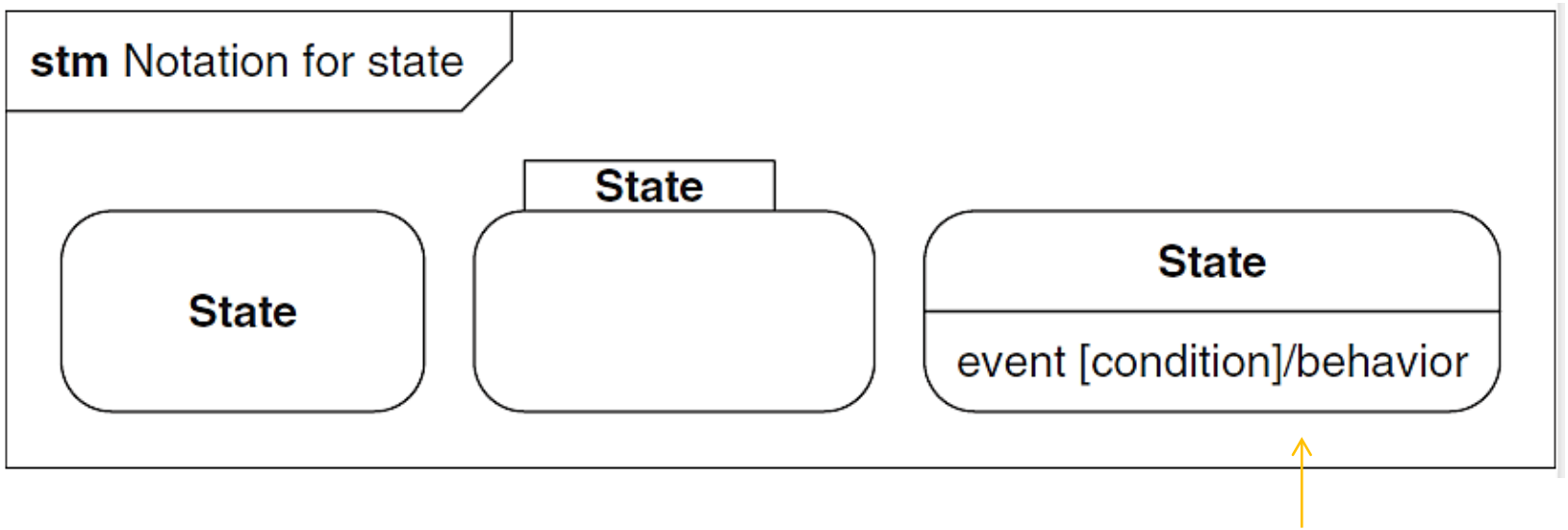




States [2/4]

- **A system** (or a part within a system) sometimes has a defined *set of states* in which it can exist during system operation.
- **The concept of state** is difficult to define formally but easy to infer from real-world examples:
  - A file, for example, can exist in the following states: *Open, Closed, Modified, Unmodified, Encrypted, Unencrypted, and others*

State [3/5]



Another compartment shows the ***internal behavior***, which is ***triggered by events***, similar to transitions, but which ***does not cause the state to be exited.***

State [4/4]

- In addition to the internal behavior, a state can also have *three special behaviors* that are triggered on the basis of predefined events:
  - The *entry behavior* is executed immediately upon entering the state - NOT INTERRUPTABLE
  - The *exit behavior* is executed immediately before exiting the state – NOT INTERRUPTABLE
  - The *do behavior* is executed while the state is active - INTERRUPTABLE

State, Activity, Action[1/2]

A *state* may describe a situation in which the *System is doing something*.

States are assumed to take *a finite amount of time*

Transitions are assumed *to take no time*

There are two things that can be happening *during a state*: an *activity* and/or one or more *actions*.

State, Activity, Action[2/2]

An *activity* is a unit of behaviour that is *non-atomic* and, as such, can be interrupted

- ***Activities*** can only appear inside a *state*
- ***Activities*** can be differentiated from *actions* inside *states* by the presence of the keyword *do*, whereas *actions* will have other keywords, including ***Entry*** and ***Exit***.

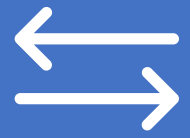
Actions are units of behaviour that are *atomic* and cannot be interrupted

- ***An action*** can exist either within a *state* or on a *transition*.

What is a transition ?

```
trigger1, trigger2, ... [condition] / behavior
```

Transition [1/4]



A transition *specifies a state transition*.

It is a *directed relationship* between two states

Defines:

a trigger and a condition that both lead to the state transition

a behavior that is *executed during the transition*.

What is an event ?

Trigger and Event

A trigger references exactly one event and establishes a relationship to a behavior.

An event specifies some occurrence that can be measured with regard to location and time.

A trigger *is the connecting link* between event and behavior in the model.

Denoted at a transition **means that** ***the transition is activated as soon as the event occurs***

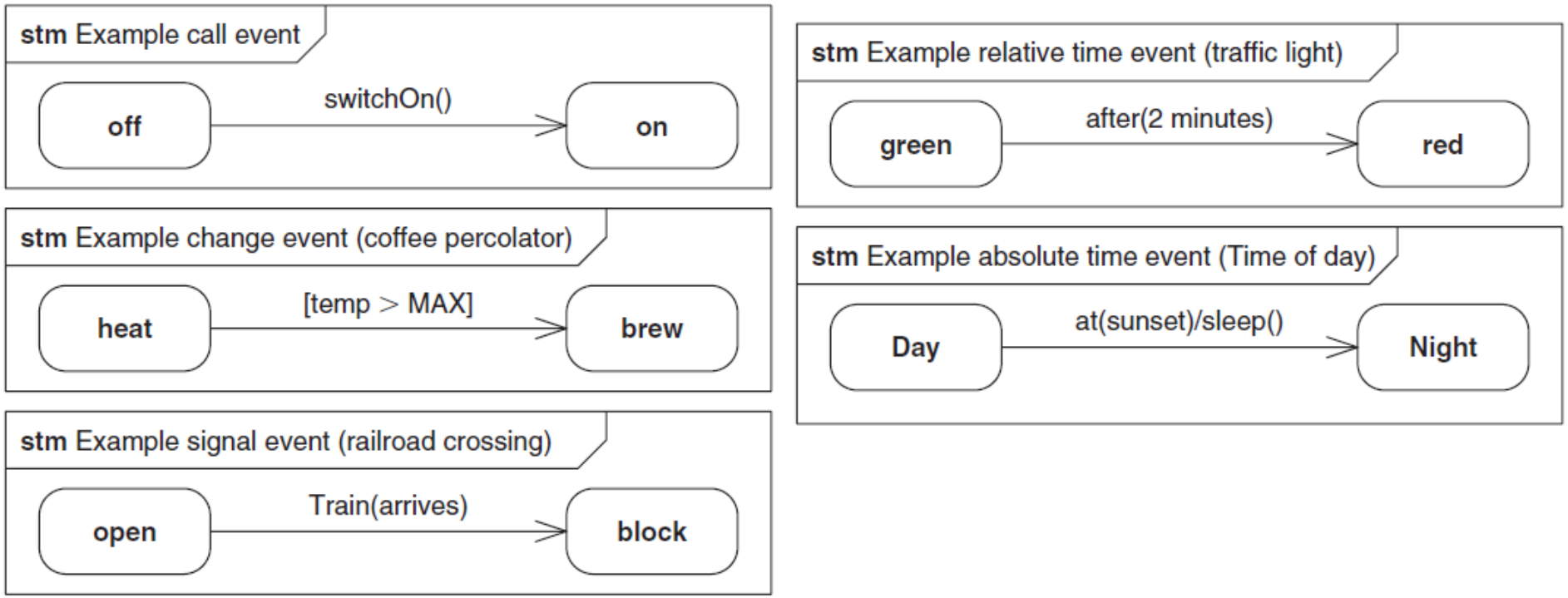
Events [1/3]

- **UML knows four types of events** that can trigger a transition:
  - **1.** A *call event* occurs when an operation is invoked.
  - **2.** *A change event* occurs when *a Boolean expression tests to true*:
    - This is the case when a value has changed accordingly in the system.

Events [2/3]

- UML knows four types of events that can trigger a transition:
  - **3**. A ***signal event*** occurs when a signal has been received.
  - **4**. **A *time event:***
    - A *relative* time event occurs when the state from which the corresponding transition originates has been active over a certain period of time (keyword *after(<time>)*)
    - An *absolute* time event occurs when the defi ned absolute time occurs (keyword *at(<time>)*).

Events [3/3]



Examples for the types of events

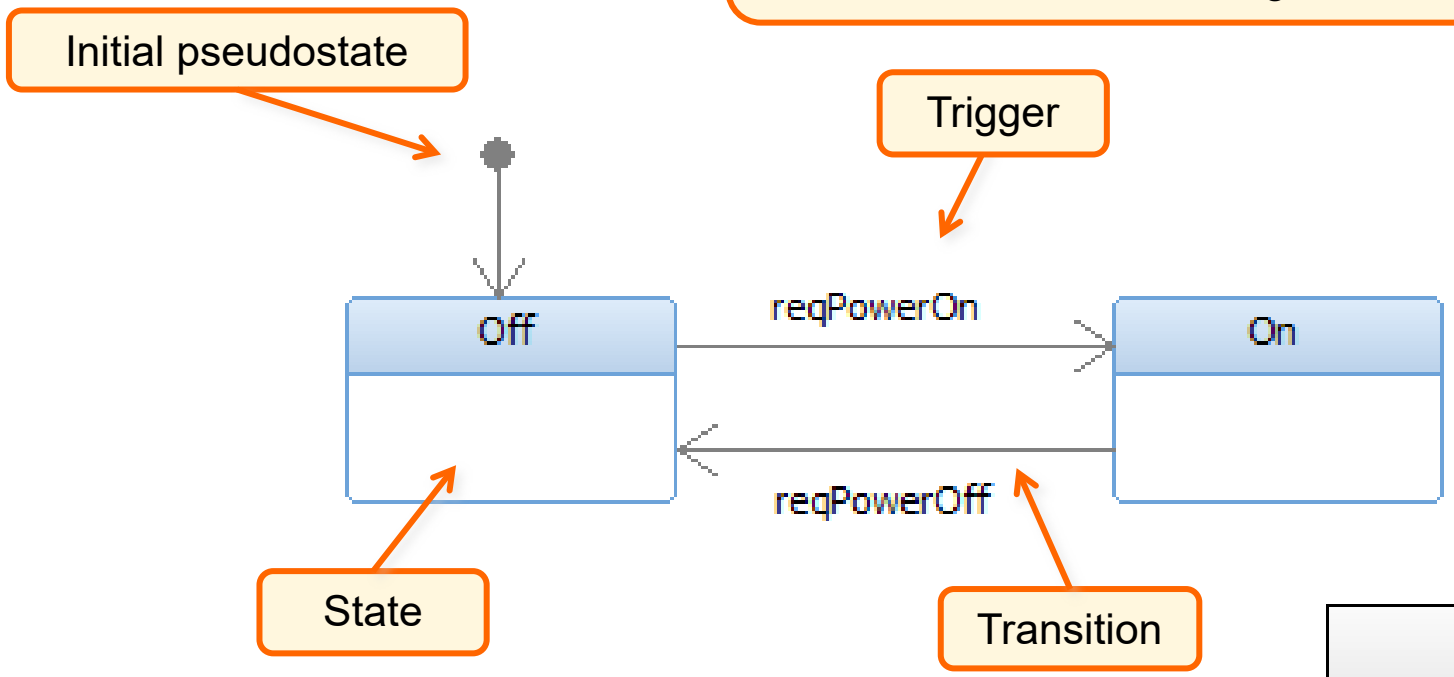
EXAMPLE: *Smart Power UnIT FOR THE CAR*

Using State machines to model requirements

Name	Specification
REQ38	The user shall be able to switch the system on and off
REQ39	The system shall indicate its power state to the user
REQ40	The system shall perform a system check before powering up
REQ41	The system shall only power up if a system check is successful
REQ42	The system shall automatically power down after a specified time in full power mode
REQ43	The system shall be able to operate in a low power mode
REQ45	Once full power mode has been activated the system shall power down after 8 seconds
REQ46	The system shall indicate the result of an unsuccessful system check to the user

States, Transitions and Triggers

In a State – time is being consumed
Triggers = Events typically defined on the Sequence diagrams



«Requirement»
REQ38

The user shall be able to switch the system on and off

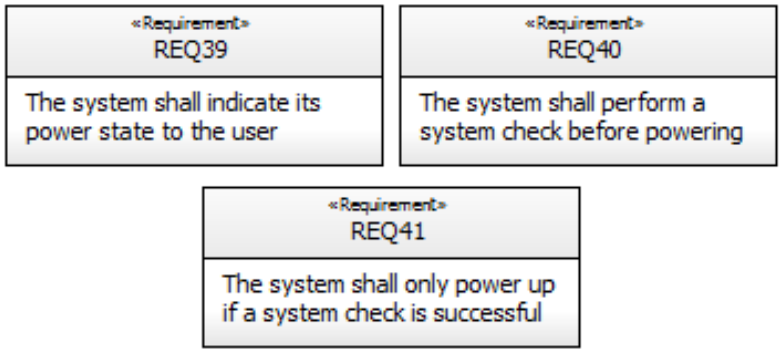
Events

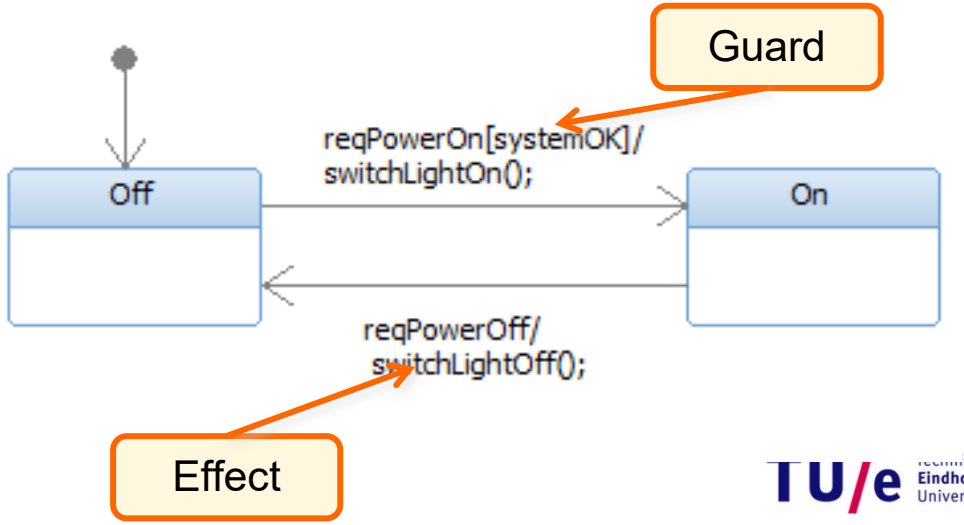
- SysML defines 4 kinds of events:
  - Signal event
    - Asynchronous signal received
  - Call event
    - Operation call received
  - Change event
    - Change in value occurred
  - Time event
    - Absolute time arrived
    - Relative time elapsed

Transition syntax

Transition syntax
trigger [Guard] / effect

- Transitions are made up of:
  - A trigger – the event that causes the transition to be considered
  - A guard – A condition that must be met before the transition takes place
  - An effect – The list of behaviors that occur when the transition takes place

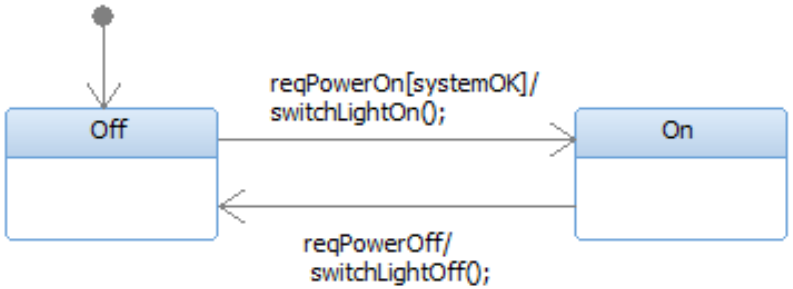




Guards

A Guard is a condition that must be met for the transition to be taken.

- Guards are evaluated *prior* to t



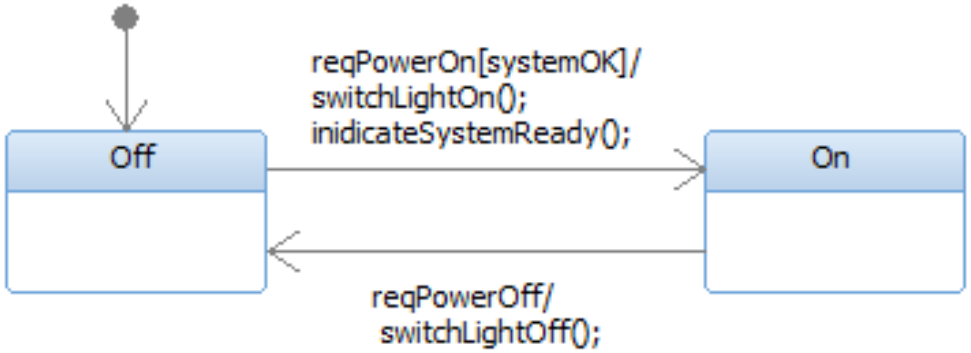
Guards can check:

- Values (that is attributes of the Block): `[cost<50]`
- Any operation that returns a `True` or `False` result

Behaviors

Behaviors on State machines occur when:

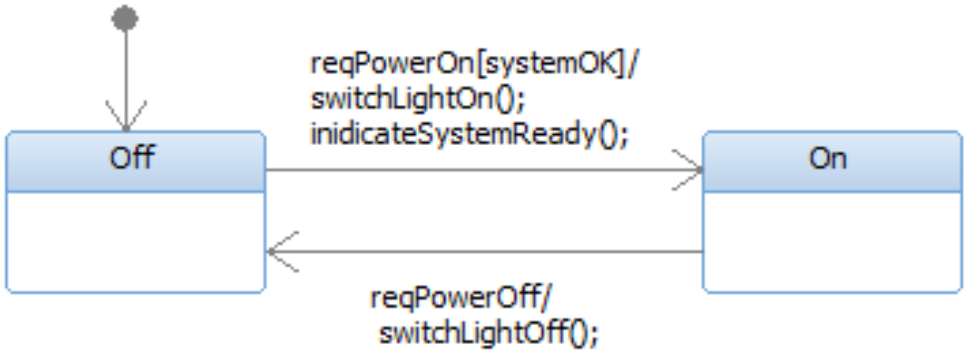
- *A transition is taken* (effects)
  - Could be external or internal to a state
- *A state is entered* (*entry* behavior)
- *A state is exited* (*exit* behavior)
- *Continually while in a state* (*do* behavior)

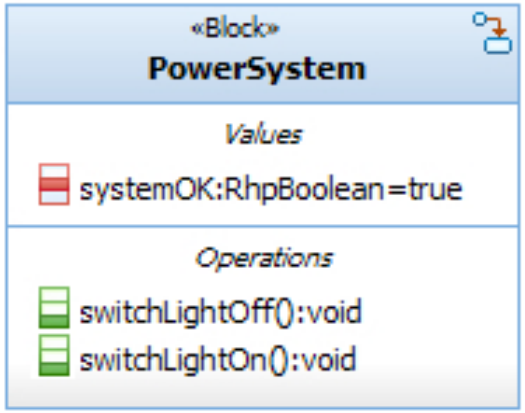


Describing behaviors

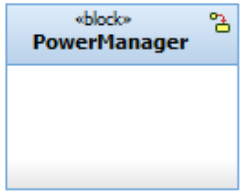
Behaviors may be described using:

- *Opaque actions directly on the diagram*
- *Activities*
- *Operations*



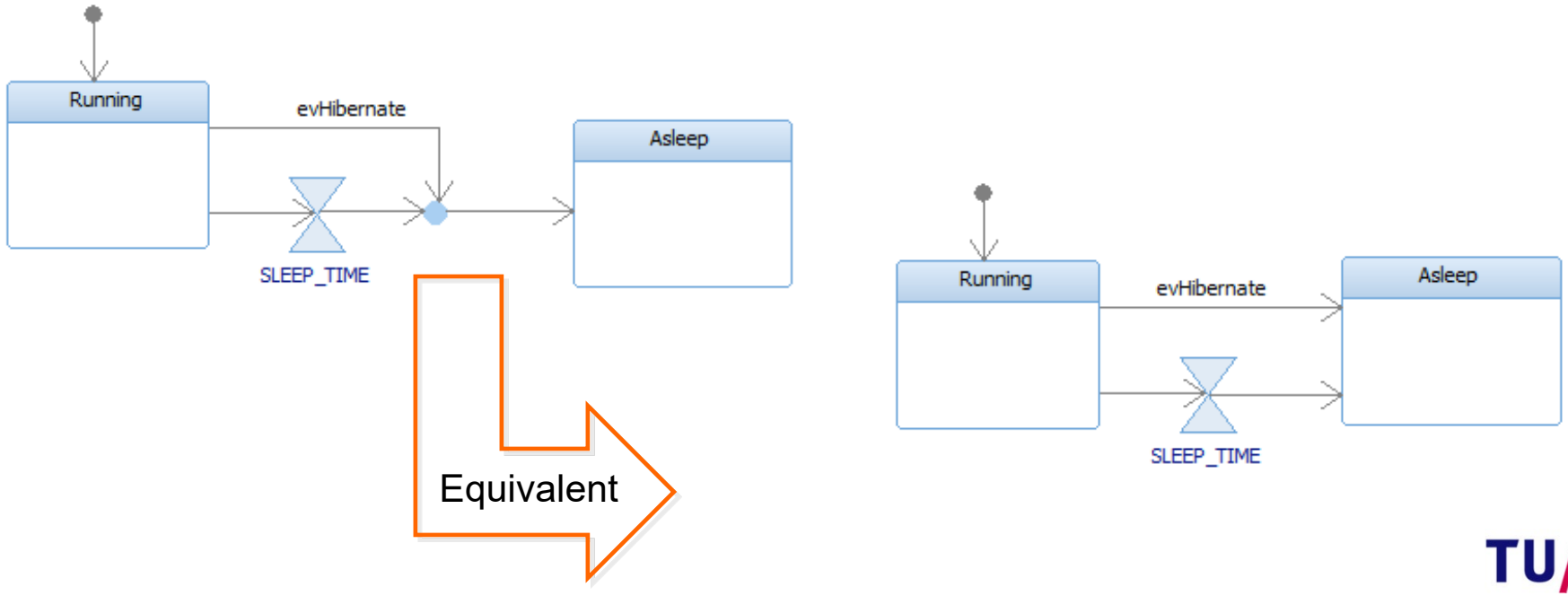


Junctions (1)



Junctions can be used to:

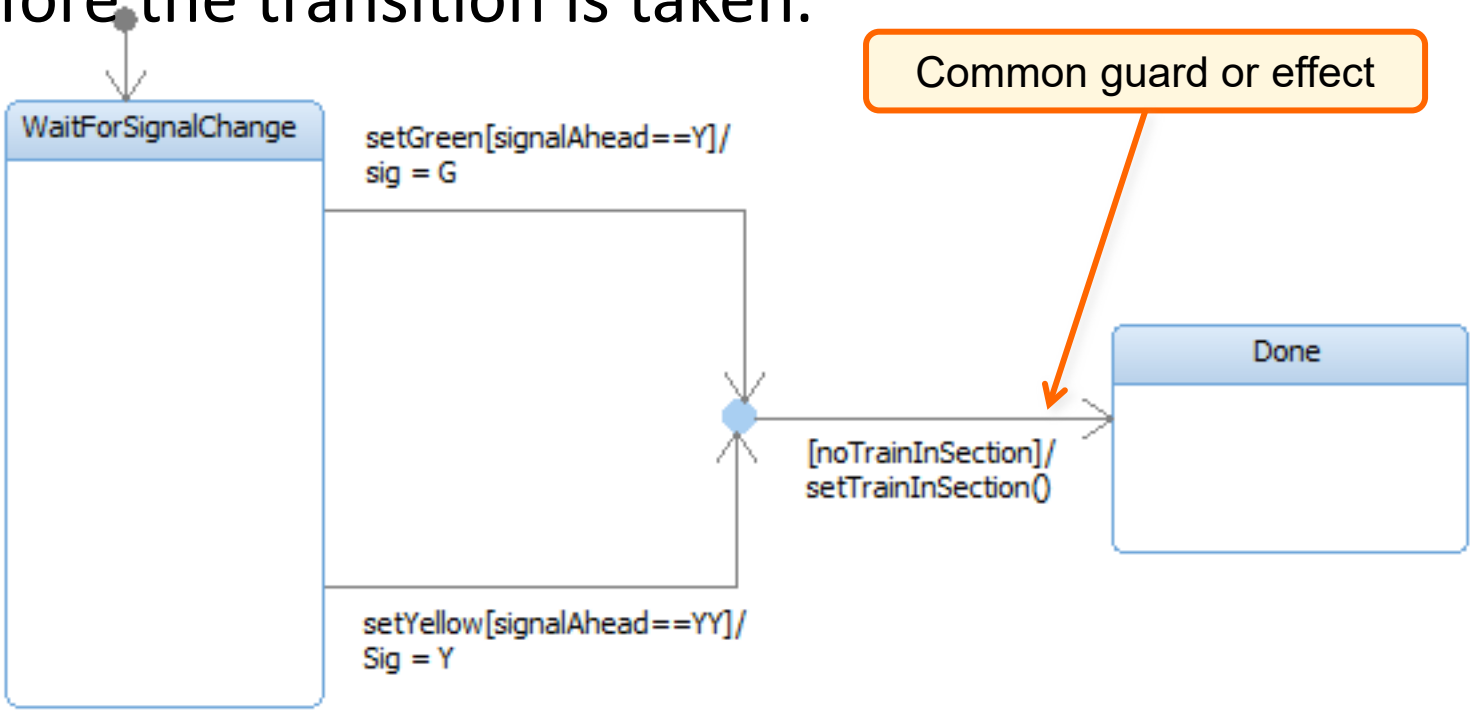
- *Join multiple transitions* into one
- *Split a single transition* into multiple possible destinations



Junctions (2)

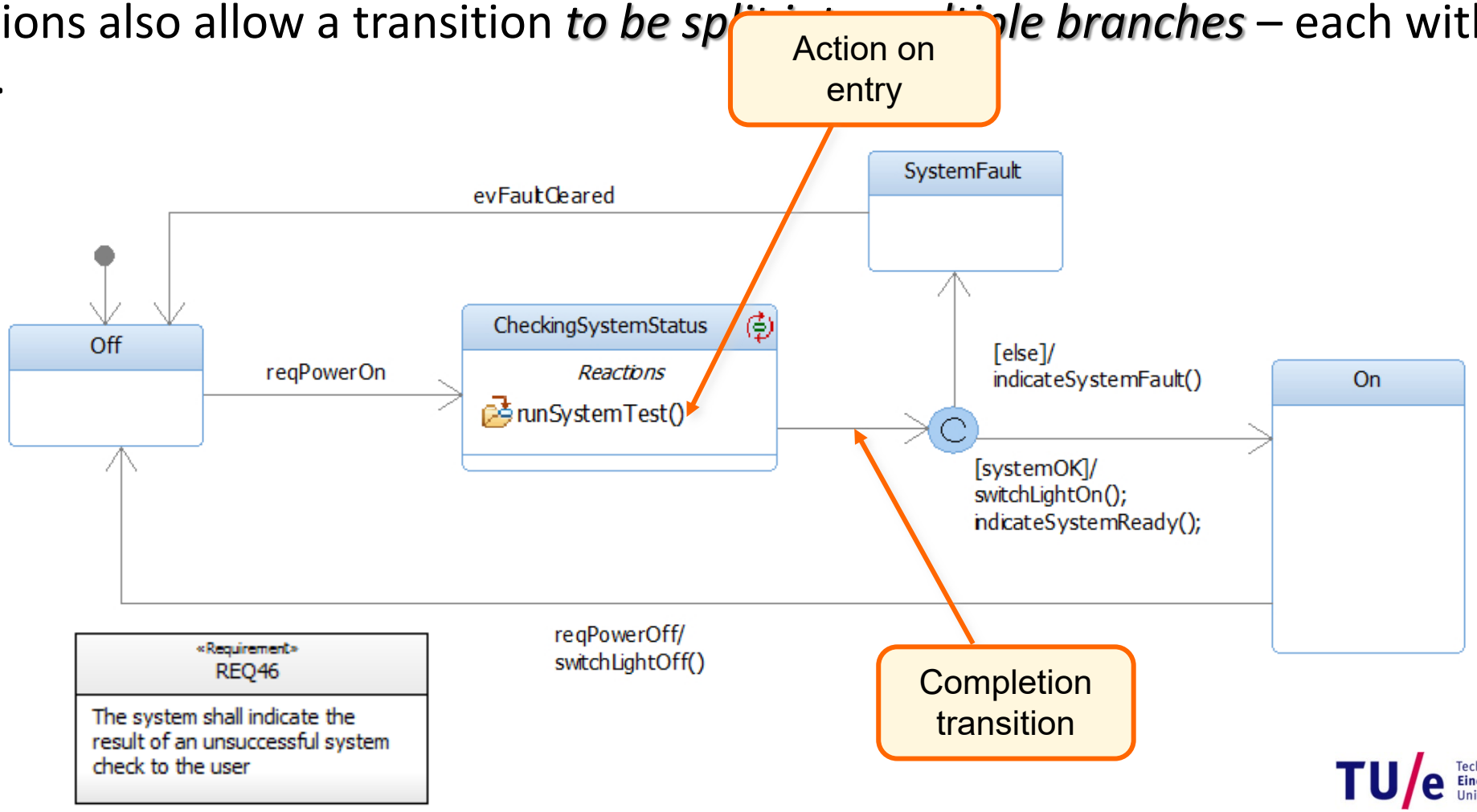
A Junction is a ***compound*** transition.

- If guards are used, then the entire compound transition is treated as a single transition: for example, all guards must evaluate to true before the transition is taken.



Junctions (3)

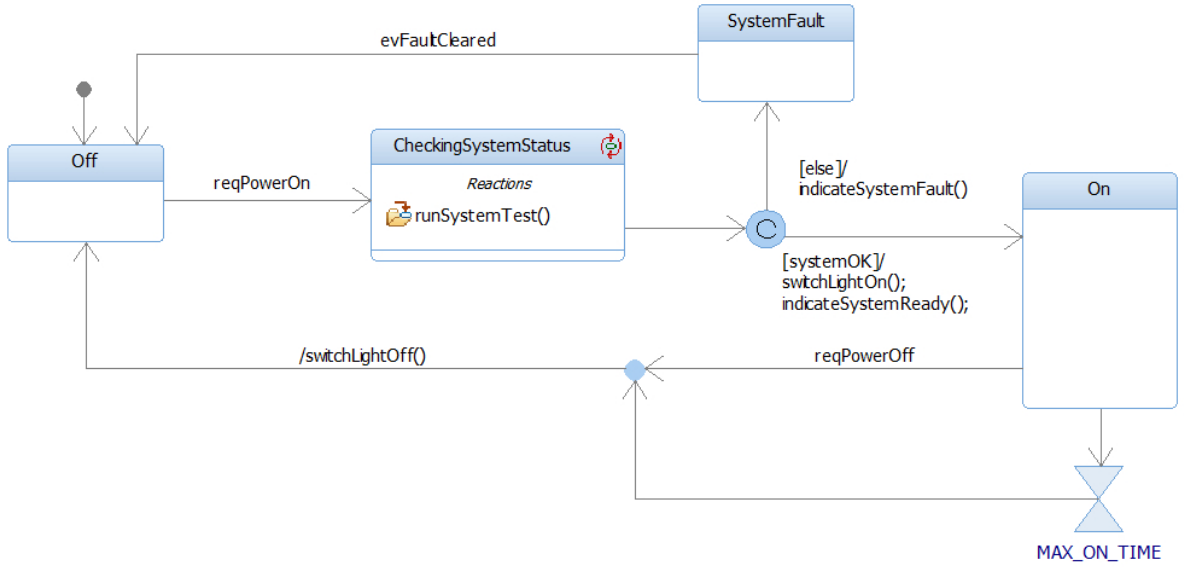
•Junctions also allow a transition *to be split into multiple branches* – each with a guard.

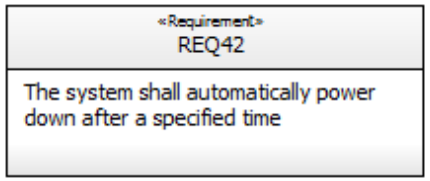


Time events

When an instance of a Block enters a state, a timer is started for any time event associated with that state.

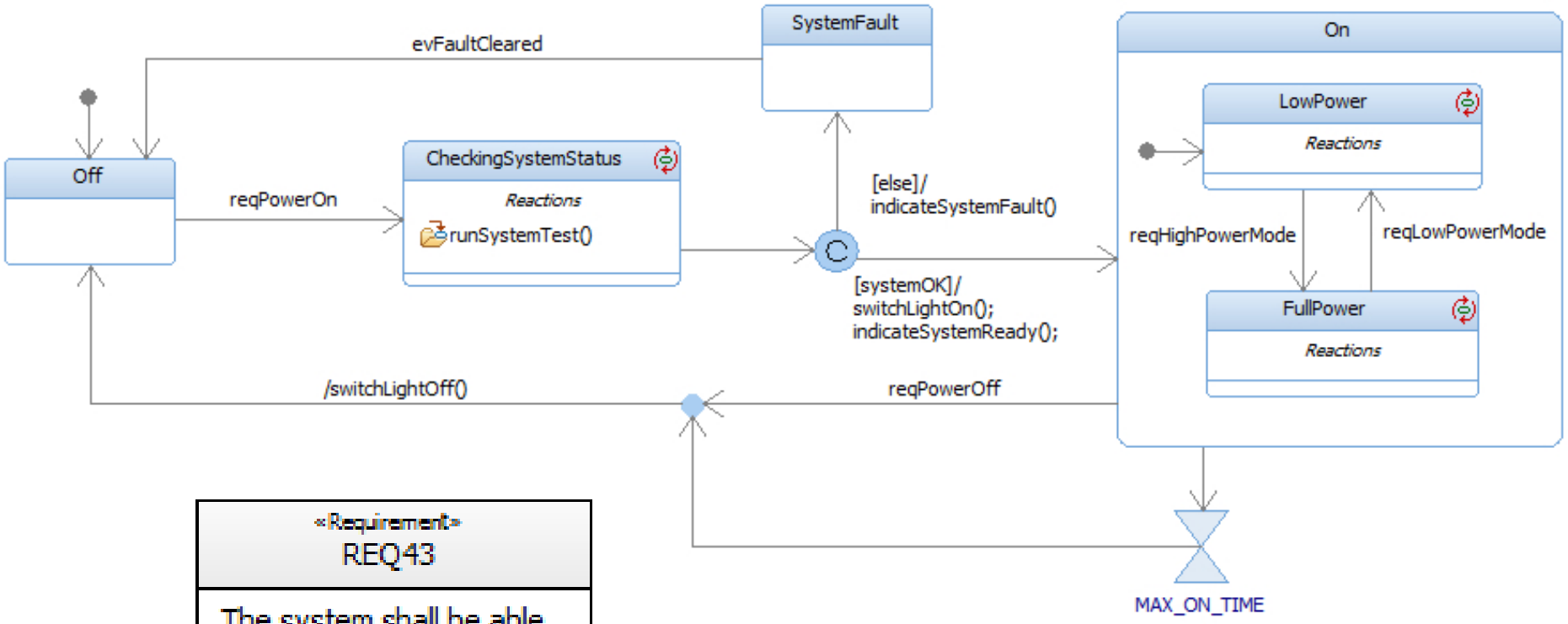
- When ***the timer expires***, the State machine receives the expiration as an event.
- When ***the state is exited***, any timer that was started on entry to that State ***is cancelled***.





Time event

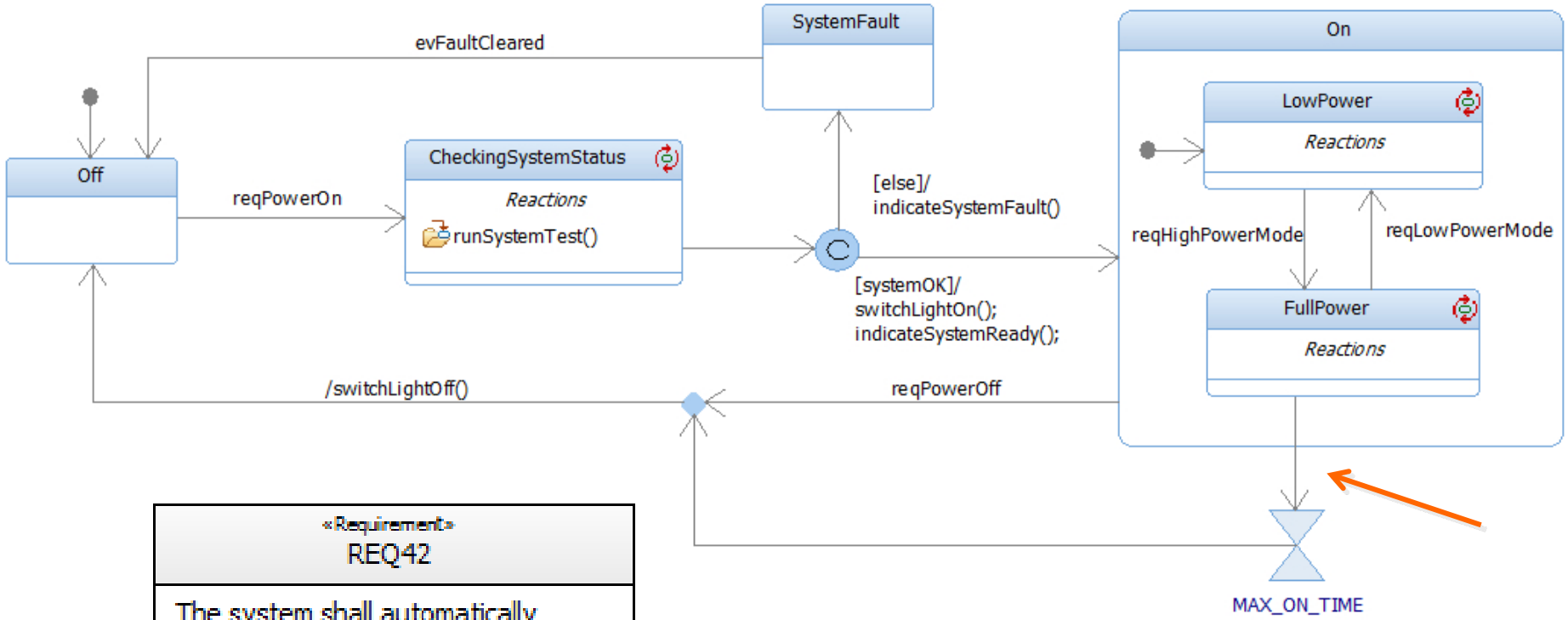
Nested states



«Requirement»
REQ43

The system shall be able to operate in a low power mode

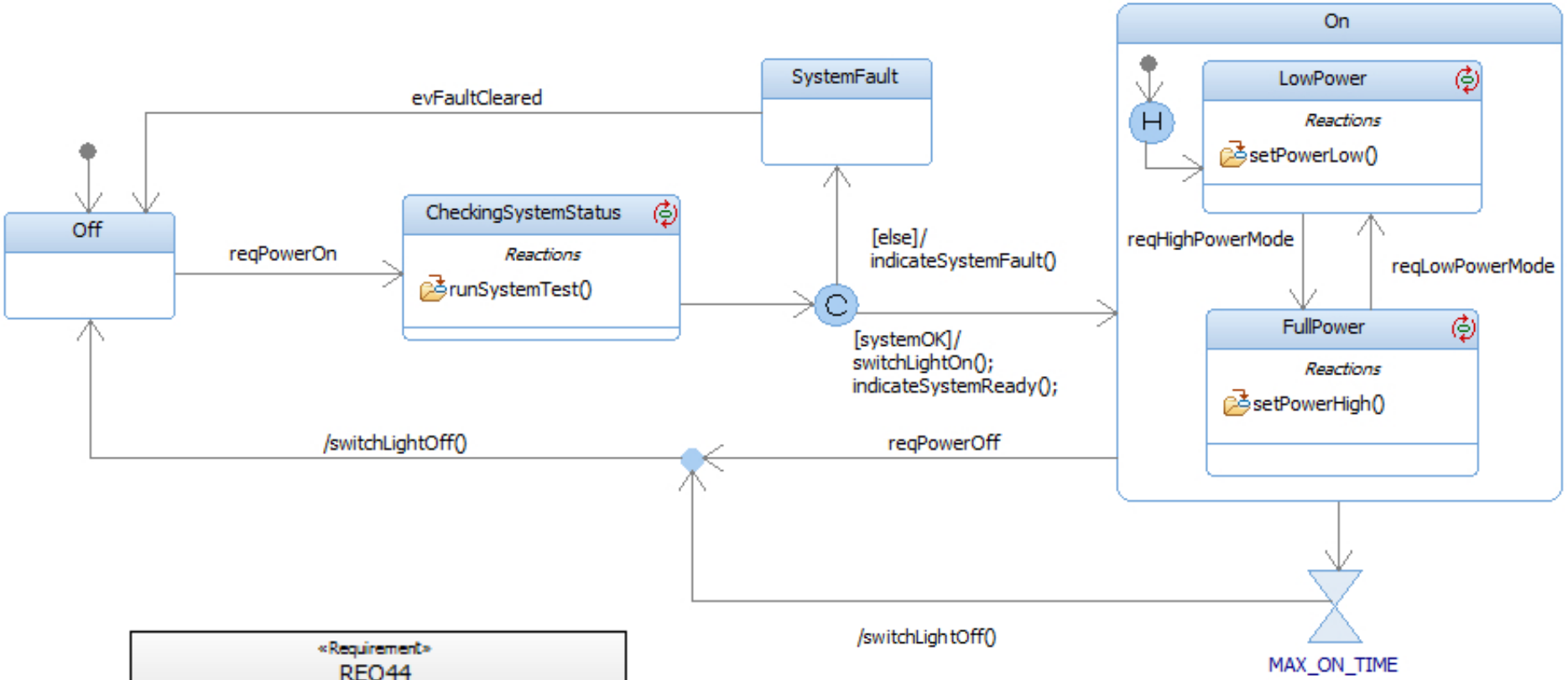
Nested states: responding to events



«Requirement»
REQ42

The system shall automatically power down after a specified time in full power mode

Nested states: history



«Requirement»
REQ44

When powered up the system shall return to the power mode it was in when it was last powered down

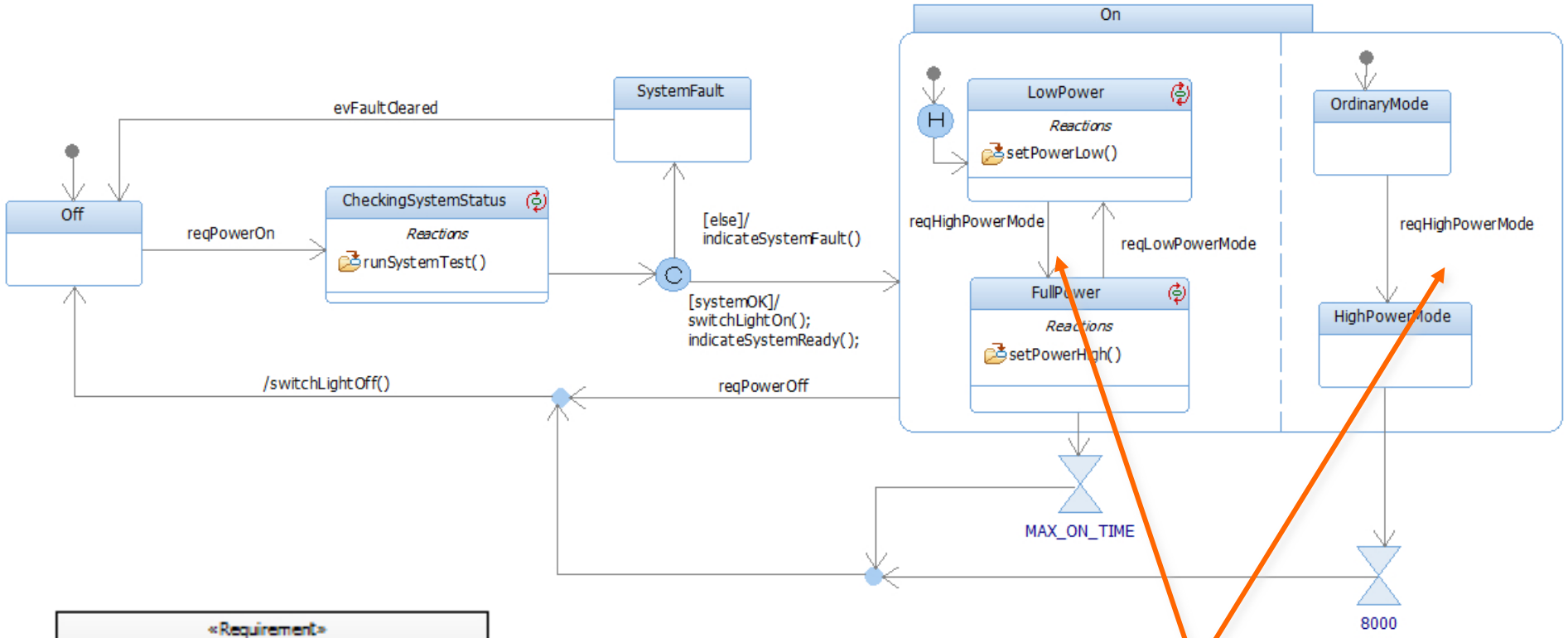
Deep or shallow history

- There are two types of history connector:

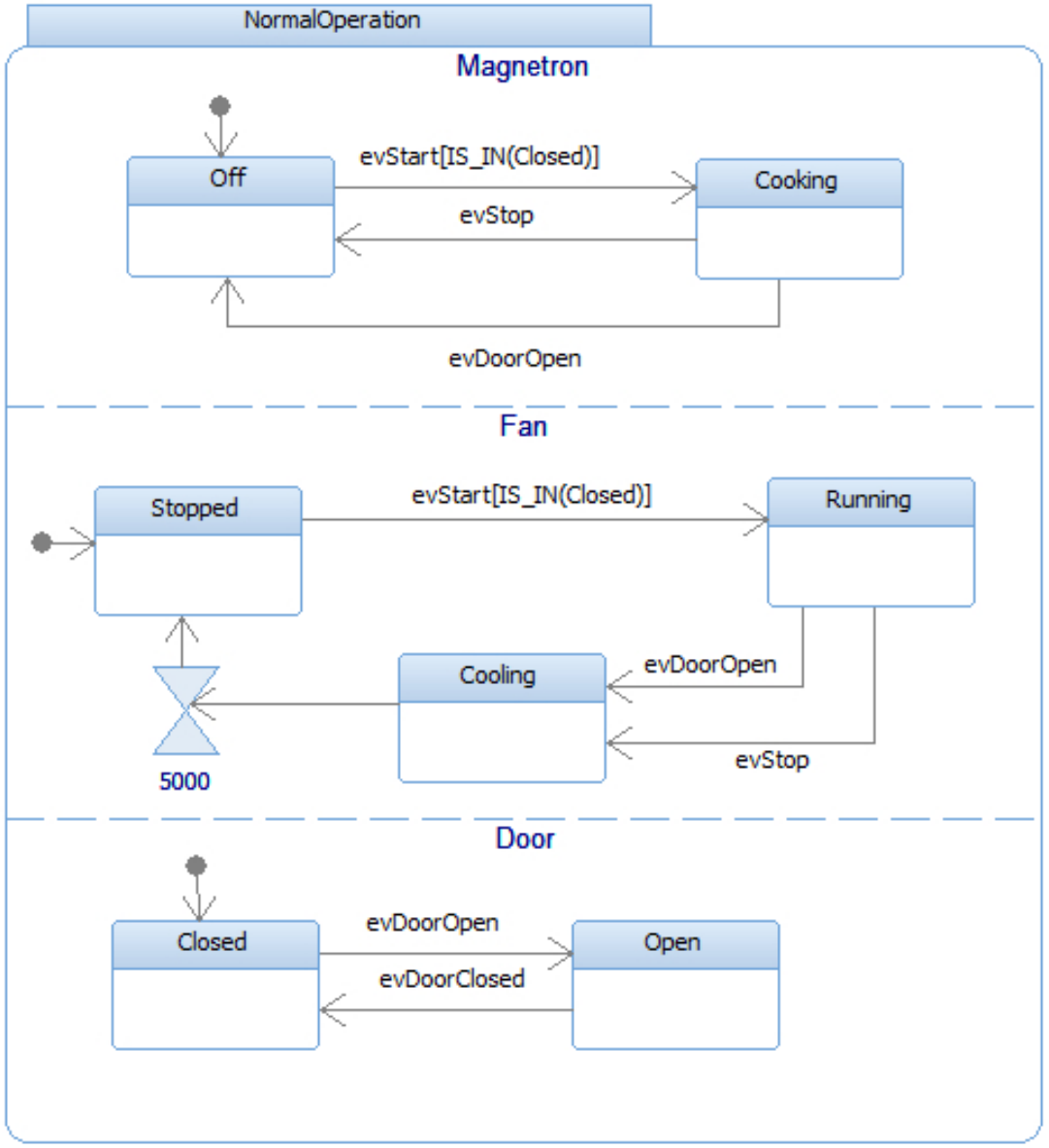
- Deep history
- Remembers which nested state you were in to any level of nesting

- Shallow history
- Only remembers which state you were in at this level – nested states take their defaults

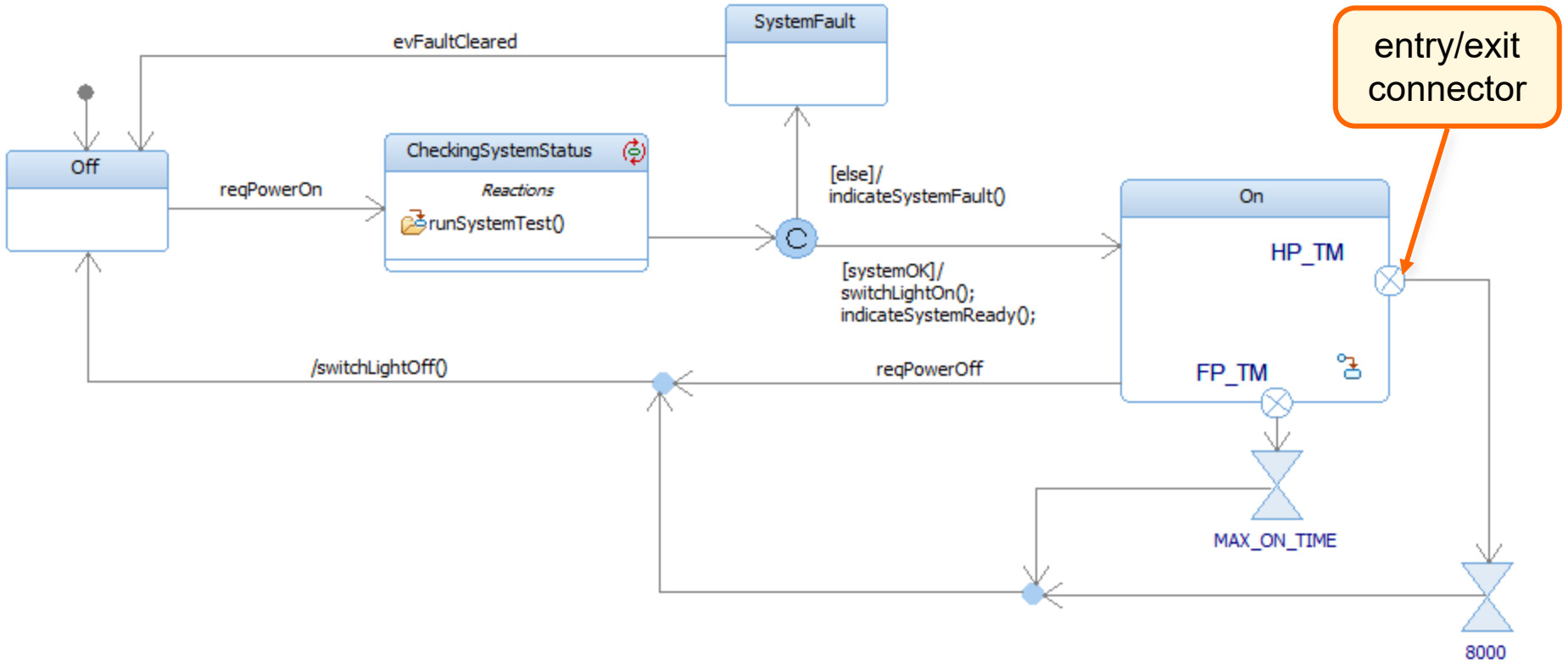
Orthogonal regions



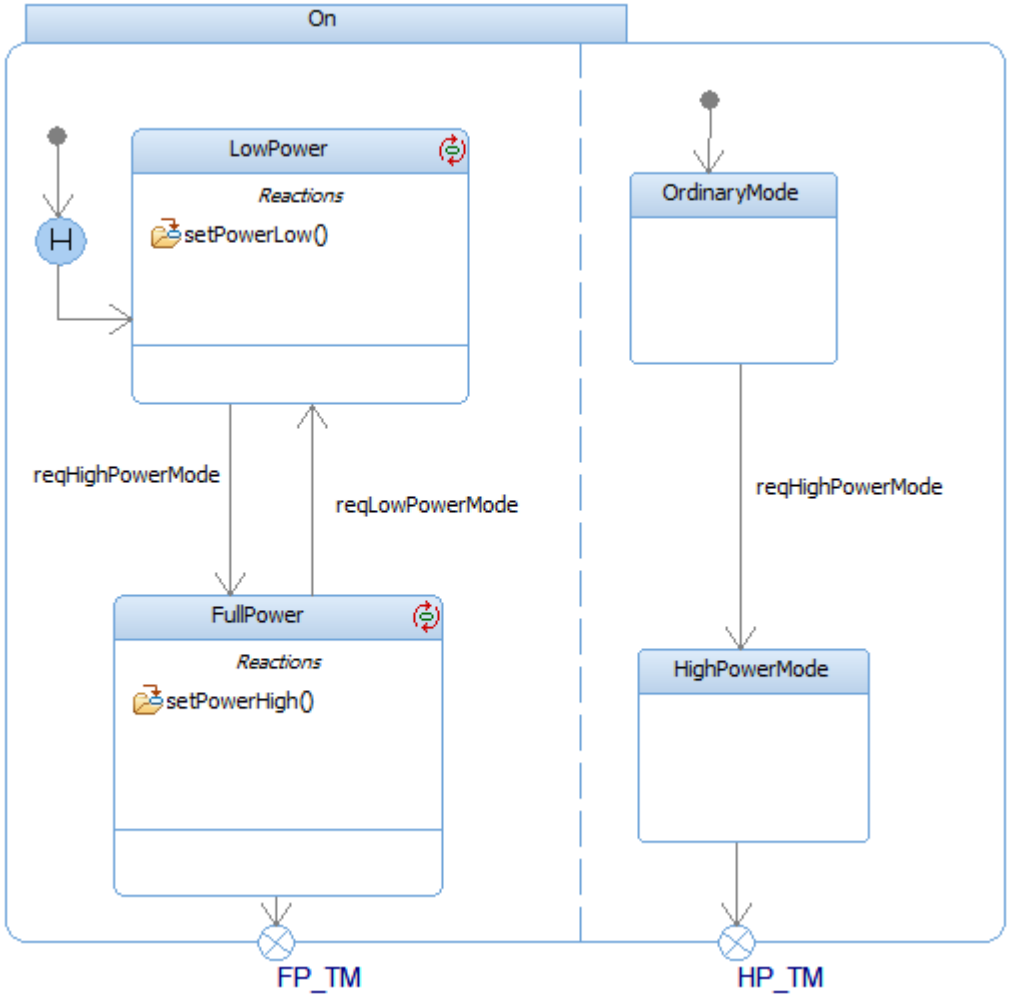
Orthogonal region example – Microwave oven



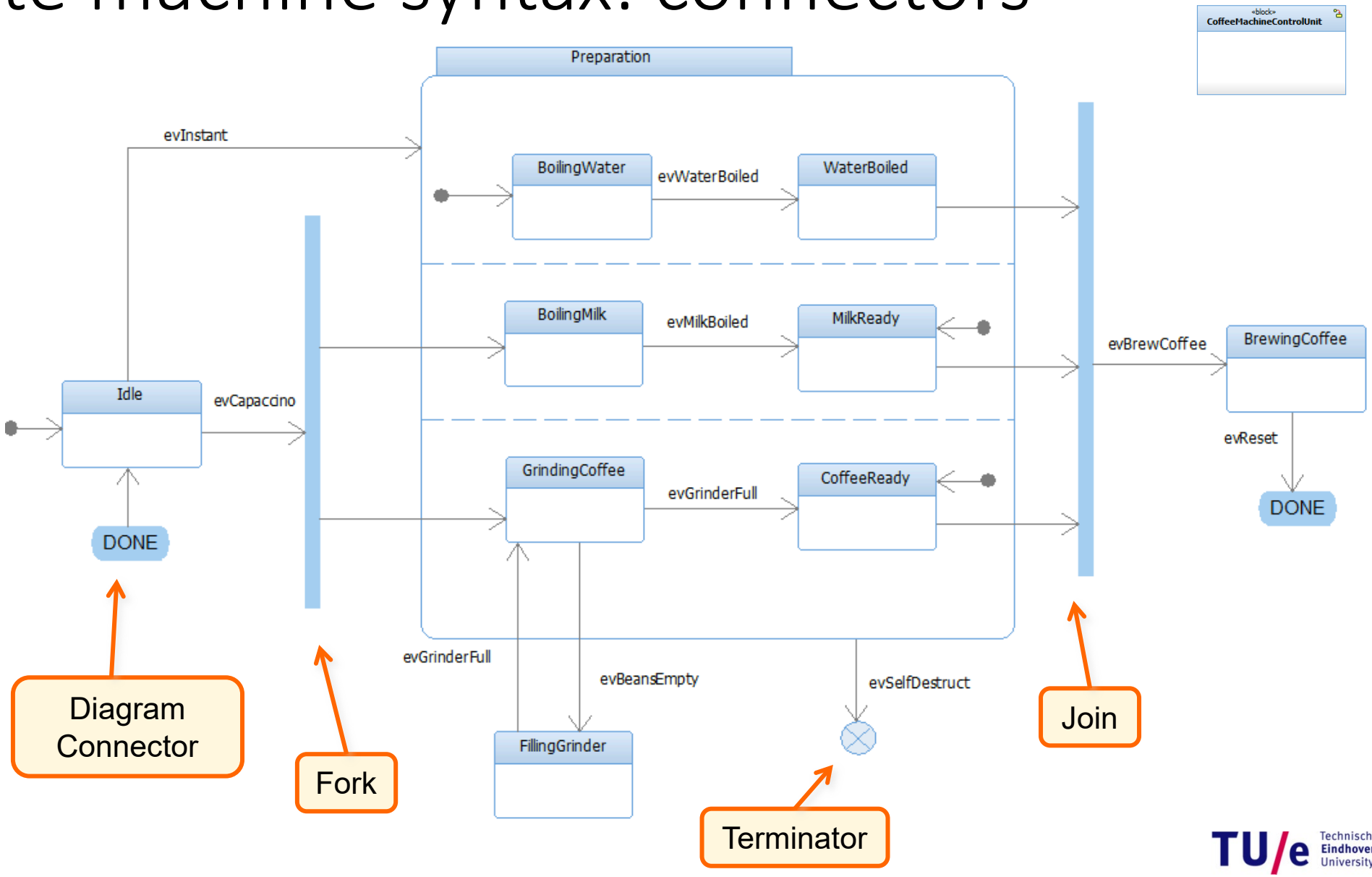
Sub-machine states - parent



Sub-machine states - child

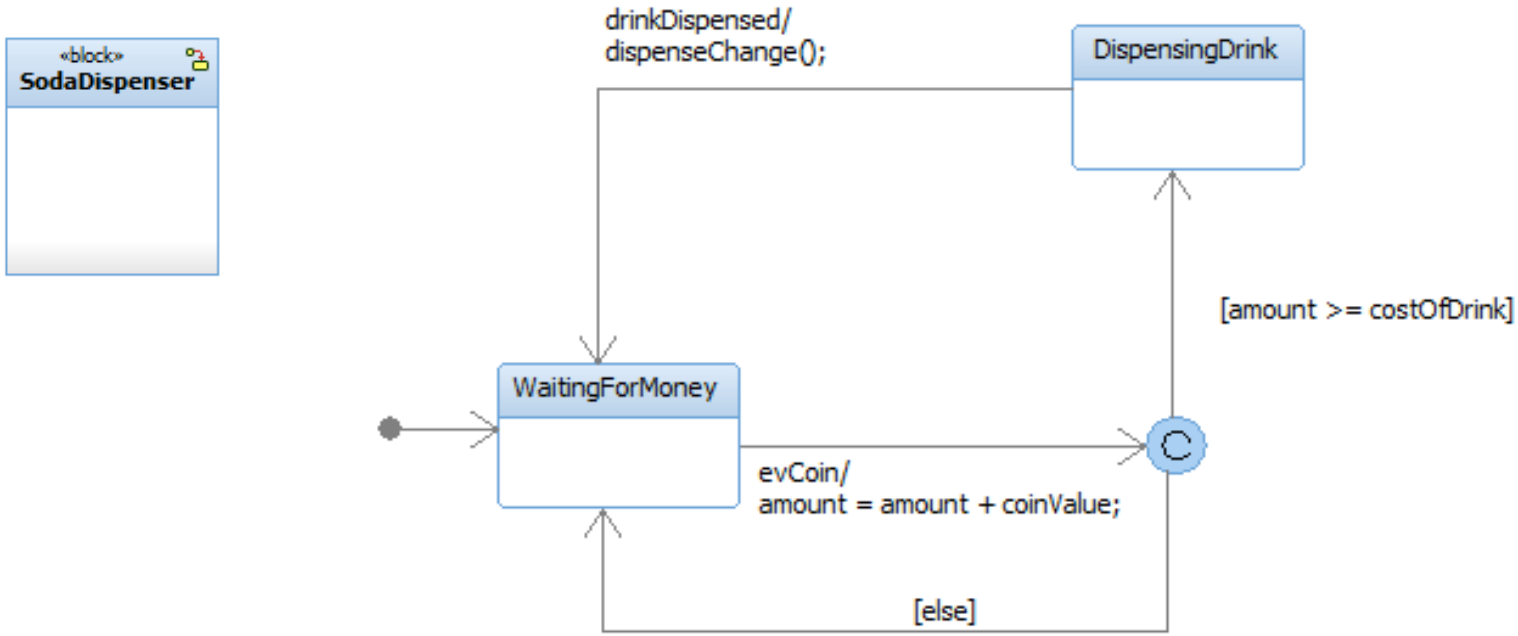


State machine syntax: connectors

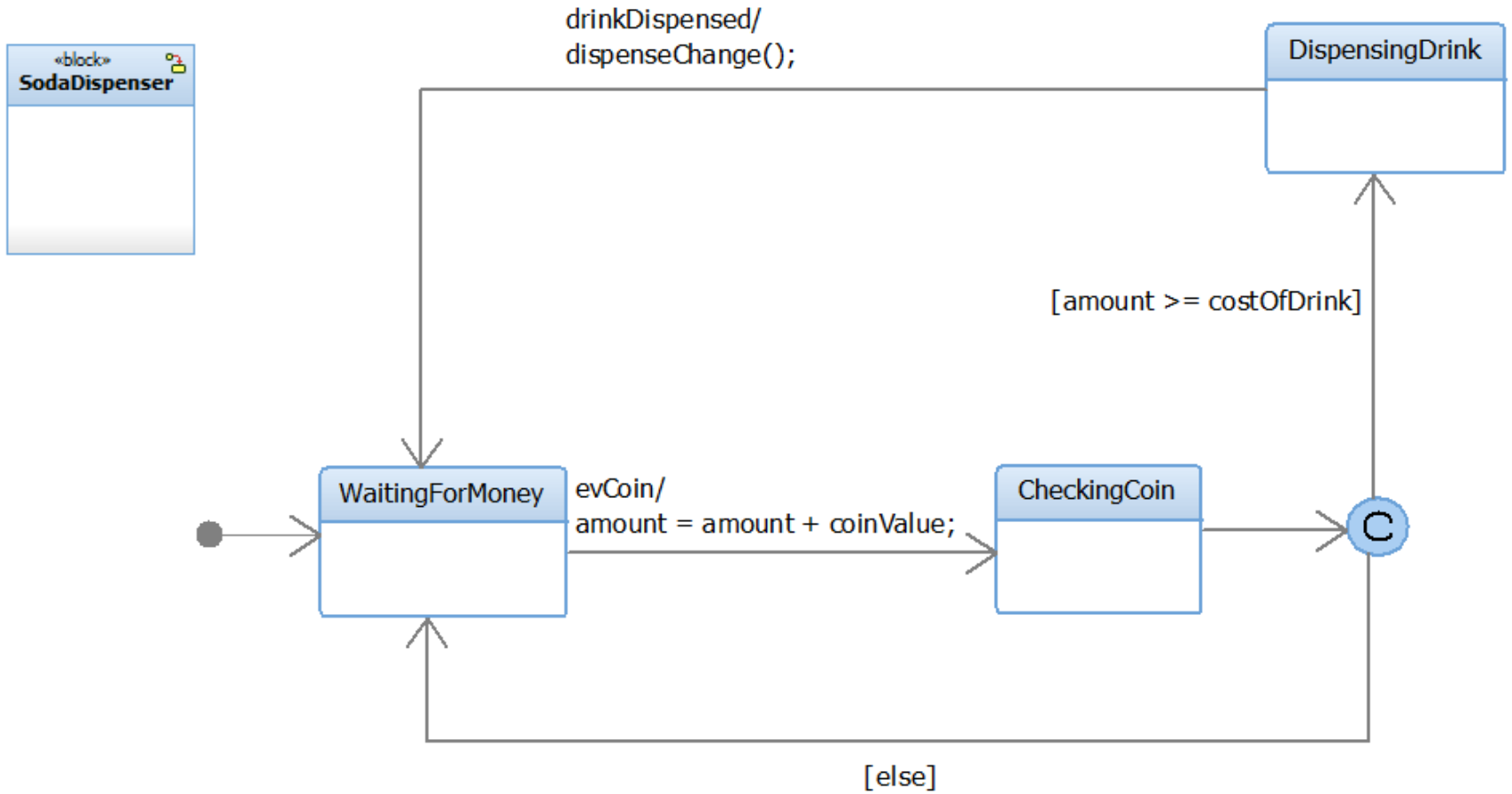


State machine problems (1)

What is wrong with the following Statemachine ?



State machine problems (1) - fixed



TU/e EINDHOVEN UNIVERSITY OF TECHNOLOGY