State Based Modelling

Budapesti Műszaki és Gazdaságtudományi Egyetem
Hibatűrő Rendszerek Kutatócsoport

Table of Contents

BEHAVIOURAL MODELLING

Structural and Behavioural Modelling

- Structural
  - static
  - whole and part, components
  - connections
- Behavioural
  - dynamic
  - timeliness
  - states, processes
  - reactions to the environment (context)

The main components of the robot vacuum cleaner are the control unit, the roller gear and the vacuum cleaner.

For the command „to right” changes the roller gear its operational mode to „turn”.

Viewing Points of Behavioural Modelling

- What the system „does”?

  → Event based models
  Process based models
  …

- What are the properties of the system now, and how do they „change”?

  → State based models

Motivating Example: Virtal Keyboard

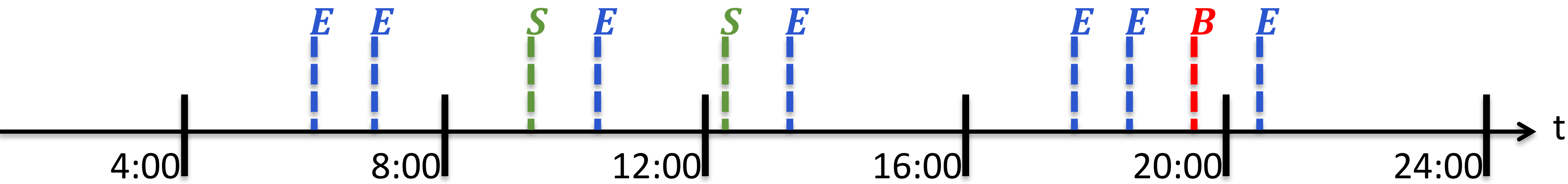
- What happens, if the top left corner is touched?
  - Q, q, 1 or =
  - Is it fully determined by „the past” only?

Basic Terms: Discrete Events

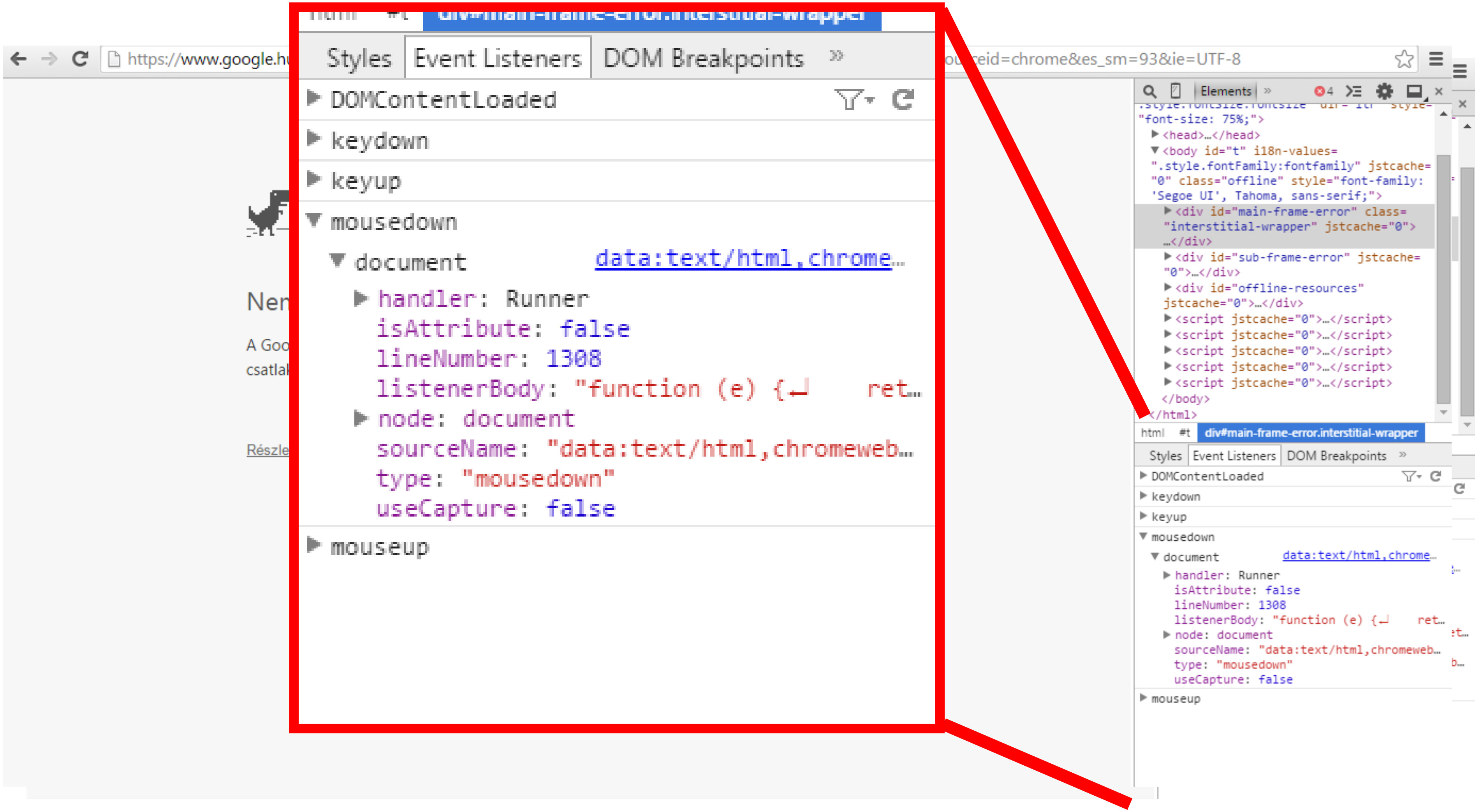
- **Event**
  - instantaneous change (e.g. at the input/output of the system)
- **Event stream**
  - e.g. one per input/output – one per data source
- **Event space** – {allowed events}
  - Readable input values / emittable output values
- **Series of** (instantaneous) **Events**, ≤ 1 at a time

Event stream: smart phone status messages

Event space: {***Email***, ***SMS***, ***Battery low***}



Event Based Programming



STATE PARTITIONING

Key Concept: State Space

The **state space**

- is a set of distinct system states,
- from which always <u>exactly one</u> element (the **current state**) is characteristic for the system at a time.

- Examples: state spaces
  - days: {*Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday*}
  - States of the microwave oven: *{full power, defrost, off}*
- Examples: current state
  - Today is *Thursday*.
  - The microwave oven is *off*.

Properties of the State Space

„ always <u>exactly one</u> element is characteristic for the system”

- Not any set of states can be a state space!

- **Completeness**
  - Always at least one of the states is active
  - Counter example (not a state space!)
    - {*Monday, Tuesday, Thursday, Saturday*} not complete
- **Mutual exclusivity**
  - Only a single state can be current in a moment
  - Counter examples (no state spaces!)
    - {*Working day, Weekend, Afternoon*} comp. but not exclusive
    - microwave oven: {*door is open, switched off*}

Why Are These Properties That Important?

- On the 29th February at the airport of Düsseldorf

29. Februar 2016 | 13.46 Uhr

29. Februar

Schalttag legt Gepäckförderband am Flughafen lahm

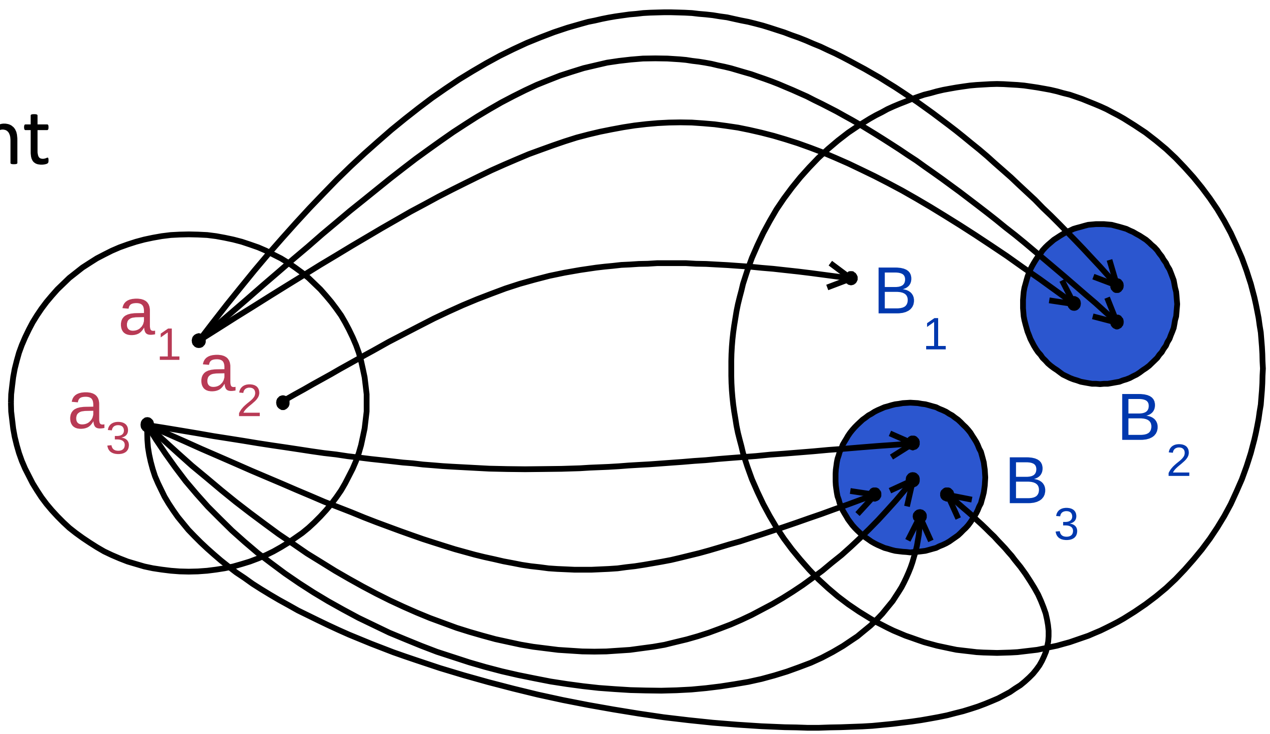
State Refinement, State Abstraction

State refinement and **state abstraction**, respectively, are **set refinement** and **set abstraction** on a state space that result in a new state space.

- (other kinds of abstraction will be discussed later …)

Repetition:

Set refinement



State Refinement, State Abstraction

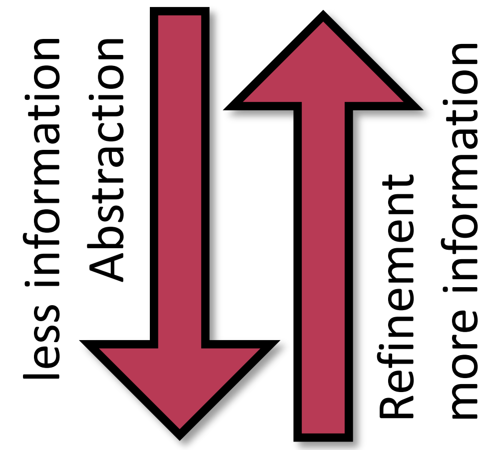
- State refinement/abstraction:
  Set refinement/abstraction of a state space
  - {*Mo*, *Tu*, *We*, *Th*, *Fr*, *Sa*, *So*}

    {*Working day*, *Weekend*}

    Only they are relevant for a schedule.

  - Abstraction of the microwave oven
    - {*full power*, *defrost*, *off*} → {*on*, *off*}
    - „*off*" state was not refined

IS IT TRUE?



Motivation for Refinement

- State refinement: Why?
  - Adding implementation detail during a design process
    - e.g. knowing the possible power levels of the oven is important for designing its power adaptor
  - Specialization / extension
    - e.g. a more advanced oven may contain timer
  - Joint behavior of several subsystems (see later)

- More information, more knowledge
  - What is the trade-off?

Motivation for Abstraction

- State abstraction: Why?
  - Useful if the abstract states are „uniform“
    - Merged sub-states are similar in certain aspects
  - Details may be irrelevant for some design phases
    - Easier to work with smaller, simpler state space
    - Less storage for the states, easier processing
    - Hidden details can be changed
    - Corner-case: **stateless** model ($|S| = 1$)
  - Sometimes only limited amount of information can be disclosed
  - Frequent form: **decomposition** (see it later)

Partitioning from Multiple Viewpoints

- Multiple correct state spaces of a single system
- E.g.: two disjoint state spaces of the microwave
  - according to power: {*full power*, *defrost*, *off*}
  - according to door position: {*open*, *closed*}
  - not completely independent:
    if the door is open, power must be switched off

- State space of the subsystem
  - It is decidable without knowledge of the system state, which state of the subsystem is the current one.

(Direct) Product of State Spaces

- Considering two state spaces together
  - $S_1 = \{$*full power*, *defrost*, *off*$\}$
  - $S_2 = \{$*open*, *closed*$\}$

$S_1 \times S_2$	*open*	*closed*
full power	*full power* and *open*	*full power* and *closed*
defrost	*defrost* and *open*	*defrost* and *closed*
off	*off* and *open*	*off* and *closed*

state abstraction

state abstraction (projection)

Remark: $|S_1 \times S_2| = |S_1| \cdot |S_2|$

(Direct) Product of State Spaces

Direct product of state spaces

- **Composition operation** over the component state spaces
- that results in a new state space (**product state space**),
  - which is formed as the Descartes product of the sets of the component state spaces.

In the product state spaces corresponds

- to each combination of the states of the component state spaces
- a combined state (**state vector**) .

$$\{\mathrm{AM},\mathrm{PM}\} \times \{1\mathrm{h}..12\mathrm{h}\} \times \{0\mathrm{m}..59\mathrm{m}\}$$
$$\Cup$$
$$\langle PM, 12h, 08m \rangle$$

Projection of the State Space to a Component

Projection to a component is

- a state abstraction operation
- that from the product state space
  - keeps one or more components,
  - and neglects the others.

$$\{\mathrm{AM},\mathrm{PM}\} \times \{1\mathrm{h}..12\mathrm{h}\} \times \{0\mathrm{m}..59\mathrm{m}\}$$

$$\rotatebox{90}{$\in$}$$

$$\langle PM, 12h, 08m \rangle$$

$$\langle PM, 12h \rangle$$

(like the projection of tables)

Refined Composition of the State Space

- Dependent state variables
  - Not all combinations can actually manifest
  - Composite state space is <u>more fine</u> than the product

$S_1 \times S_2$	*open*	*closed*
full power	*full power* and *open*	*full power* and *closed*
defrost	*defrost* and *open*	*defrost* and *closed*
off	*off* and *open*	*off* and *closed*

state abstraction

state abstraction (projection)

Remark: projection is still an abstraction relationship

Refined Composition of the State Variables

- „ Composite state space is <u>more fine</u> than the product”
  - ... because we excluded two composite states
  - ... the refined state space has *fewer* states!
    - until now state space refinement resulted in a higher number of states
    - after refinement the number of states can go up or down
    - the important thing:
      more is known about the system, more precise description is provided
    - therefore:
      less systems satisfy the model

$S_1 \times S_2$	*open*	*closed*
full power	*full power* and *open*	*full power* and *closed*
defrost	*defrost* and *open*	*defrost* and *closed*
off	*off* and *open*	*off* and *closed*

Decomposition of the State Variables

- Decomposition: reversing production / composition
  - off and open
  - off and closed
  - defrost and closed
  - *full power and closed*

⇨

S_1 = {*full power*, *defrost*, off}

S_2 = {open, closed}

- The projected state variables are abstractions
- Why do we decompose?
  - to process state variables separately
  - to store state variables separately

Example from Our Profession

- Where does state-based modelling play a role in IT?
- Social network – relation between Jack and Jill
  (certain functions depend on this, e.g. visibility of uploaded pictures)
  - State variables:
    - Did Jack mark Jill as a friend?
    - Vice versa

store separately:
- in RAM
- in a database (permanent)

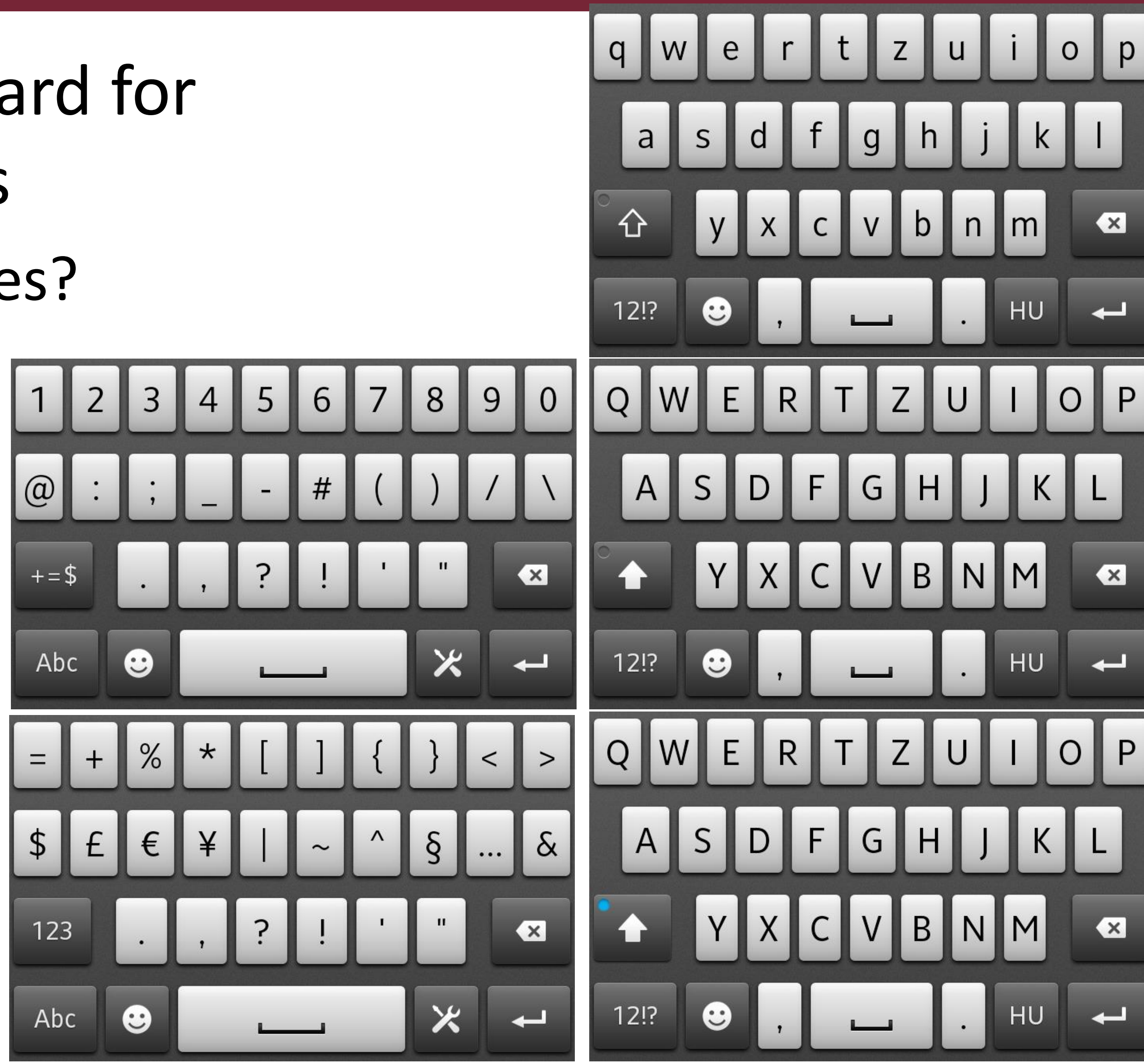
$S_1 \times S_2$		
	no relation	Jack knows Jill
	Jill knows Jack	friends

Example from Our Profession

- What is the motivation behind state abstraction?
- Social networks: only a part of the database can be visible for a single user
  - Privacy
    - We have no right to learn whether Jack knows Jill or not
  - Smaller data traffic
  - Decomposition of a software system
    - Simpler view component (HTML + CSS + JavaScript) if only the relevant information is available
    - More secure, more flexible
    - Single implementation of access control policies

Example from Our Profession

- Virtual keyboard for touchscreens
  - state variables?



Example from Our Profession

- Programming: how can we store the state?
  - Variable with appropriate value domain (object field, etc.)

```
enum VirtualKeyboardState {
    LOWER_CASE,
    UPPER_CASE_ONCE,
    UPPER_CASE_LOCK,
    NUMBERS_COMMON_SYMBOLS,
    RARE_SYMBOLS
}
// ...
VirtualKeyboardState keyboardState;
```

  - Extension: store the state of SHIFT key!
    - the alphanumeric mode „remembers” the state of the SHIFT

Example from Our Profession

- Programming: how can we store the state?
  - Extension: store the state of SHIFT key!

```
enum VirtualKeyboardStateWithMemory {
    LOWER_CASE,
    UPPER_CASE_ONCE,
    UPPER_CASE_LOCK,
    NUMBERS_COMMON_SYMBOLS_WITH_LOWER_CASE,
    NUMBERS_COMMON_SYMBOLS_WITH_UPPER_CASE_ONCE,
    NUMBERS_COMMON_SYMBOLS_WITH_UPPER_CASE_LOCK,
    RARE_SYMBOLS_WITH_LOWER_CASE,
    RARE_SYMBOLS_WITH_UPPER_CASE_ONCE,
    RARE_SYMBOLS_WITH_UPPER_CASE_LOCK
}
// ...
VirtualKeyboardStateWithMemory keyboardStateWithMemory;
```

  - Phenomenon is called **state space explosion**

Example from Our Profession

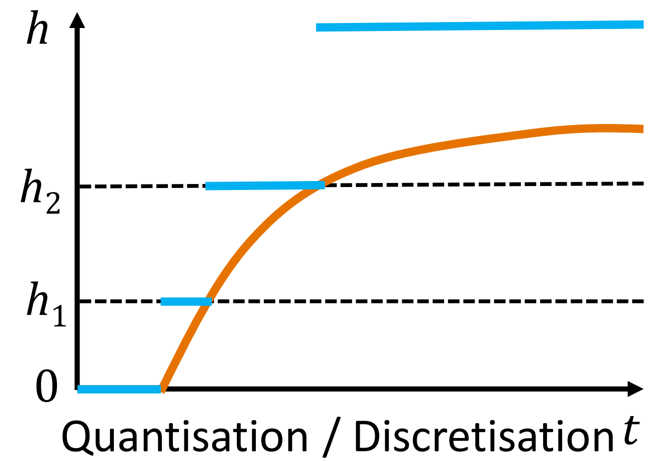
- Programming: how can we store the state?
  - compact solution: multiple state variables

```
enum VirtualKeyboardFacet {
    ALPHABETIC,
    NUMBERS_COMMON_SYMBOLS,
    RARE_SYMBOLS
}
enum CapsState {
    LOWER_CASE,
    UPPER_CASE_ONCE,
    UPPER_CASE_LOCK
}
// ...
VirtualKeyboardFacet keyboardFacet;
CapsState capsState;
```

  - decomposition of state variables

From Other Engineering Professions

- Potential infinite (or even continuum) state spaces
  - e.g. state variables of an airplane
    - $v \in \mathbb{R}$ speed
    - $h \in \mathbb{R}$ flight altitude
    - $\alpha \in [{}^{-\pi}/_{2}, {}^{\pi}/_{2}]$ rise angle
  - The state change can be continuous
    - e.g. rise of the airplane: $\partial h / \partial t = v \sin \alpha$



Quantisation / Discretisation

- But for typical IT system models
  - **discrete states** (no continuous change)
  - often **finite state space** (counter example: counter $\in \mathbb{N}$)
  - instantaneous **state transitions,** fixed states inbetween

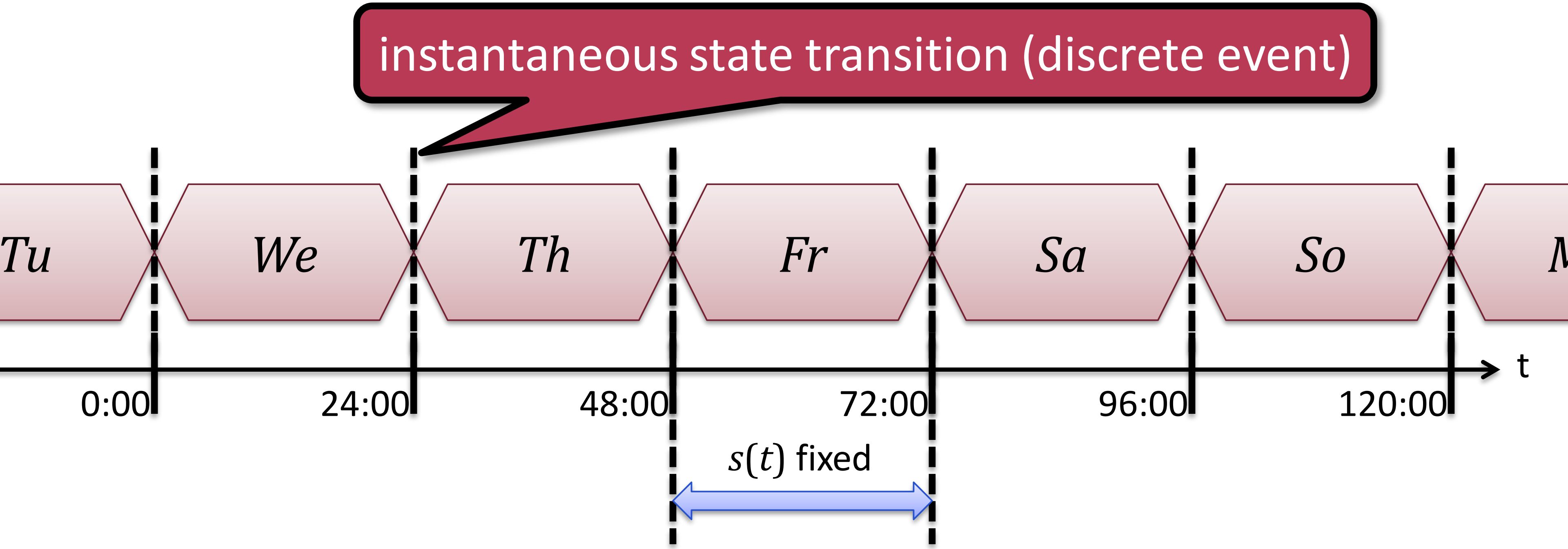
SIMPLE (MEALY) AUTOMATA

From Other Engineering Professions

- Potential infinite (or even continuum) state spaces
  - e.g. state variables of an airplane
    - $v \in \mathbb{R}$ speed
    - $h \in \mathbb{R}$ flight altitude
    - $\alpha \in [{}^{-\pi}/_2, {}^{\pi}/_2]$ rise angle
  - The state change can be continuous
    - e.g. rise of the airplane: $\partial h / \partial t = v \ \sin \alpha$
- But for typical IT system models
  - **discrete states** (no continuous change)
  - often **finite state space** (counter example: counter $\in \mathbb{N}$)
  - instantaneous **state transitions,** fixed states inbetween

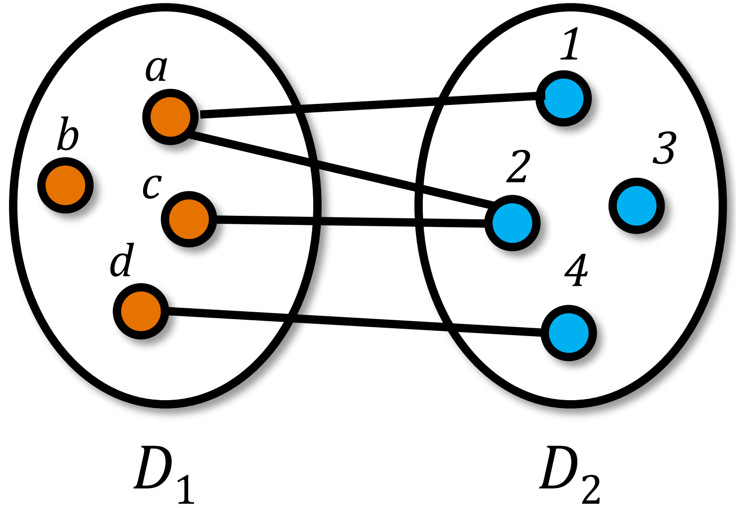
State Transitions

- State space: S
  - e.g. $S = \{Mo, Tu, We, Th, Fr, Sa, So\}$
- $s(t) \in S$
  - The current state as a function of time



Repetition: Binary Relation

- Binary Relation :
  - subset of the Descartes product of two sets
  - $R \subseteq D_1 \times D_2$



$$R = \{(a, 1), (a, 2), (c, 2), (d, 4)\}$$

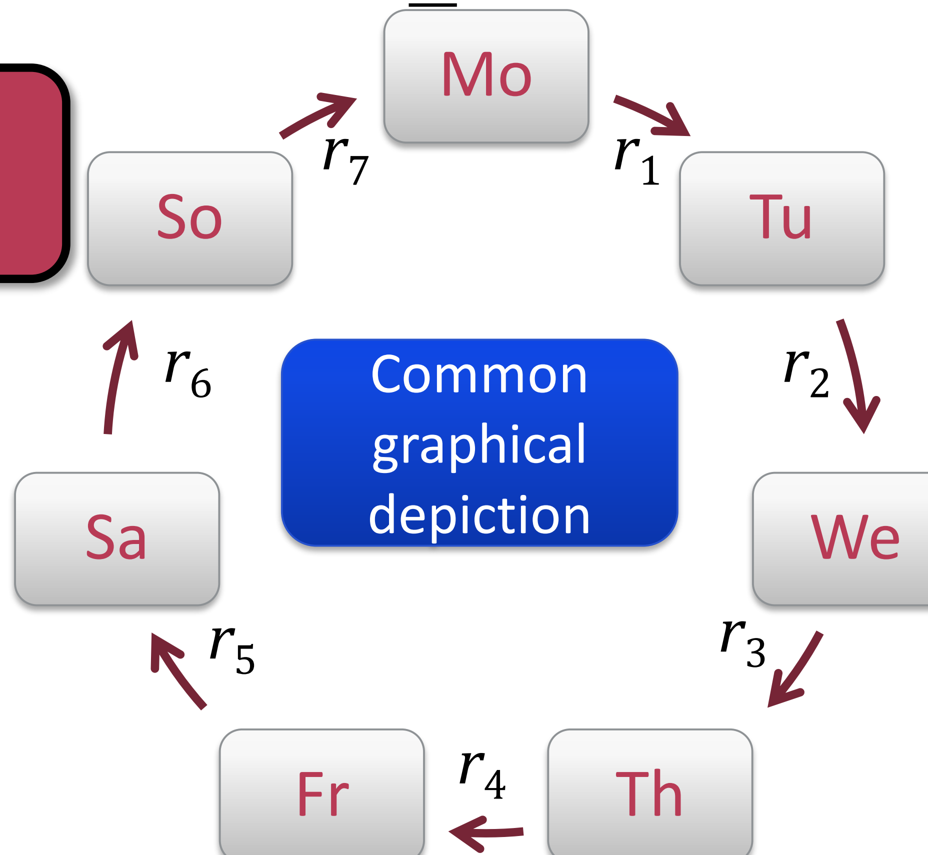
State Transitions

- What are possible sequences of states?
  - state space $S = \{Mo, Tu, We, Th, Fr, Sa, So\}$
  - event space: **state transition relation** $R \subseteq S \times S$

state transition **rules** $r \in R$

- $r_1=(Mo, Tu)$
- $r_2=(Tu, We)$
- $r_3=(We, Th)$
- $r_4=(Th, Fr)$
- $r_5=(Fr, Sa)$
- $r_6=(Sa, So)$
- $r_7=(So, Mo)$

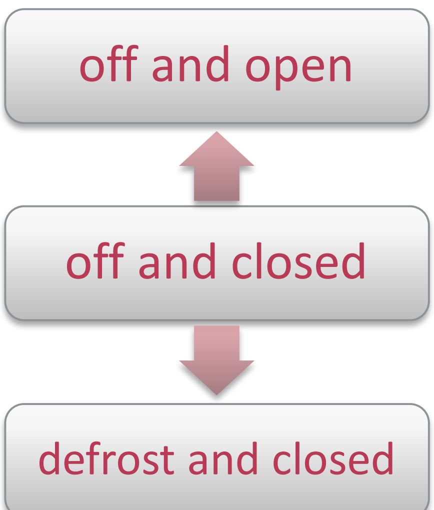
From Th can the system move to Fr



  - Also called a **state graph**

Remarks about the State Graph

- It is possible that …
  - … the graph is complete → all transitions are allowed
    - $S_{\text{kitten}} = \{\mathit{sleeping}, \mathit{playing}, \mathit{drinking}\}$
  - … not every state is reachable from each state
    - $S_{\text{glass}} = \{\mathit{empty}, \mathit{full}, \mathit{broken}\}$ → no path $\mathit{broken} \rightsquigarrow \mathit{empty}$
  - … some states have multiple successors



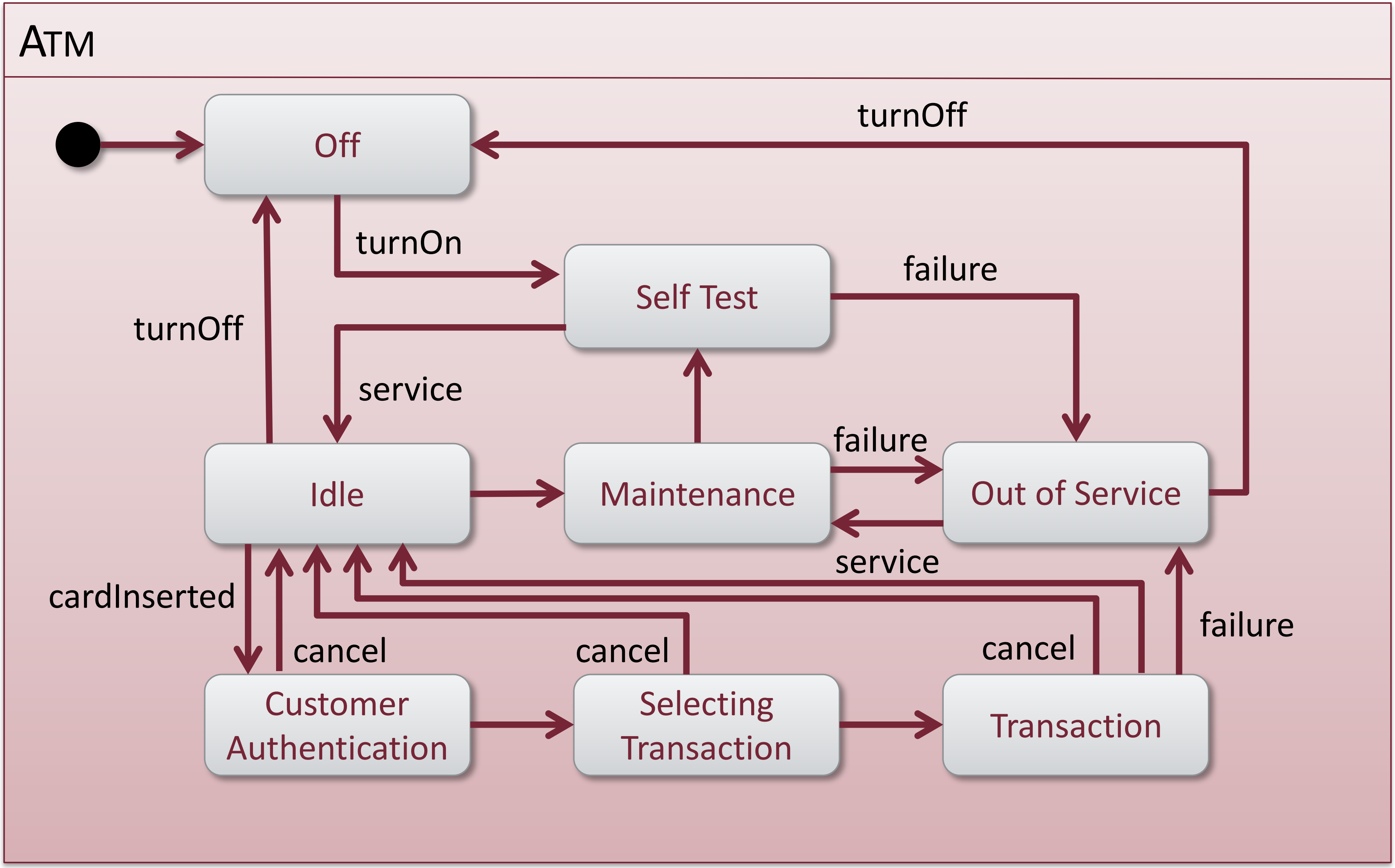
Non-determinism

- Possible sources:
  - Behaviour of the modelled system
  - Abstraction introduced during modelling

e.g. ignored internal variables, input from environment, …

SIMPLE (MEALY) AUTOMATA

State Graph Example: ATM



Labelling Transition with Events

- Label of the transition
  - Instantaneous event
  - Transition can be associated with an event

$s_1 \xrightarrow{\text{event}} s_2$

- Possible interpretations: the event can be...
  - ... the result of the transition (post condition)

$\text{Defrost} \xrightarrow{\text{bell chime}} \text{Switched off}$

  - ... the cause / trigger of the transition (pre condition)

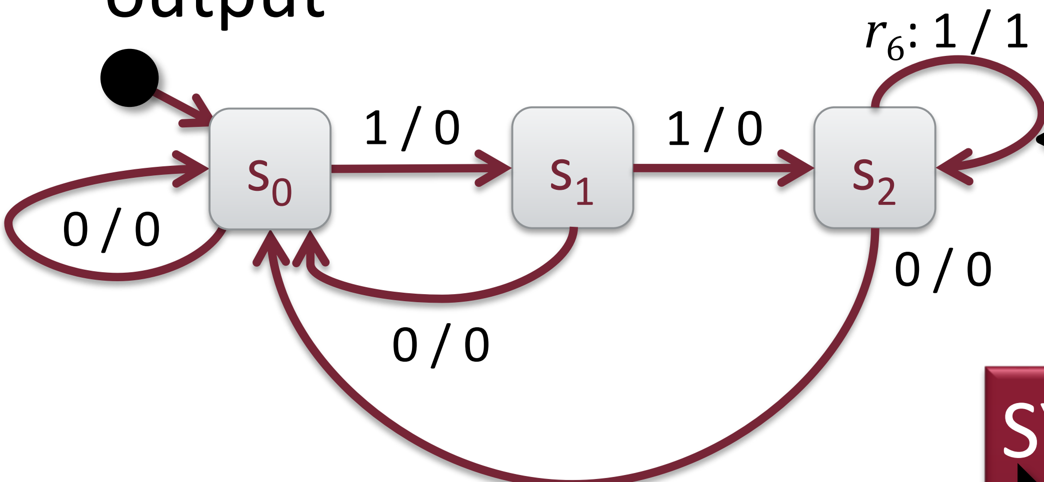
$\text{Switched Off} \xrightarrow{\text{press button}} \text{Defrost}$

- A Transition can have multiple labels
  - Reading input / Writing output

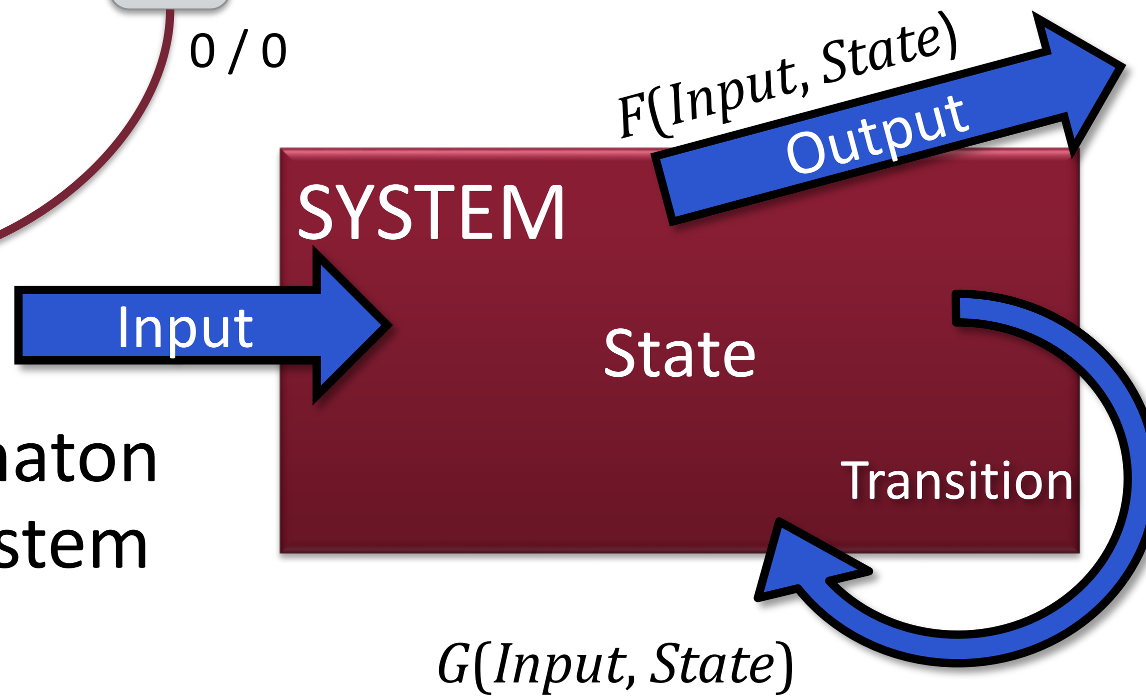
$\xrightarrow{\text{Timer signal / bell chime}}$

Memory Hook: Mealy Finite State Machine

- Initial state → $s_0 = s(t=0)$
- All transitions deterministic, reading input and writing output



Question:
- What does this machine do?
- If rule r_6 transfer to s_0?

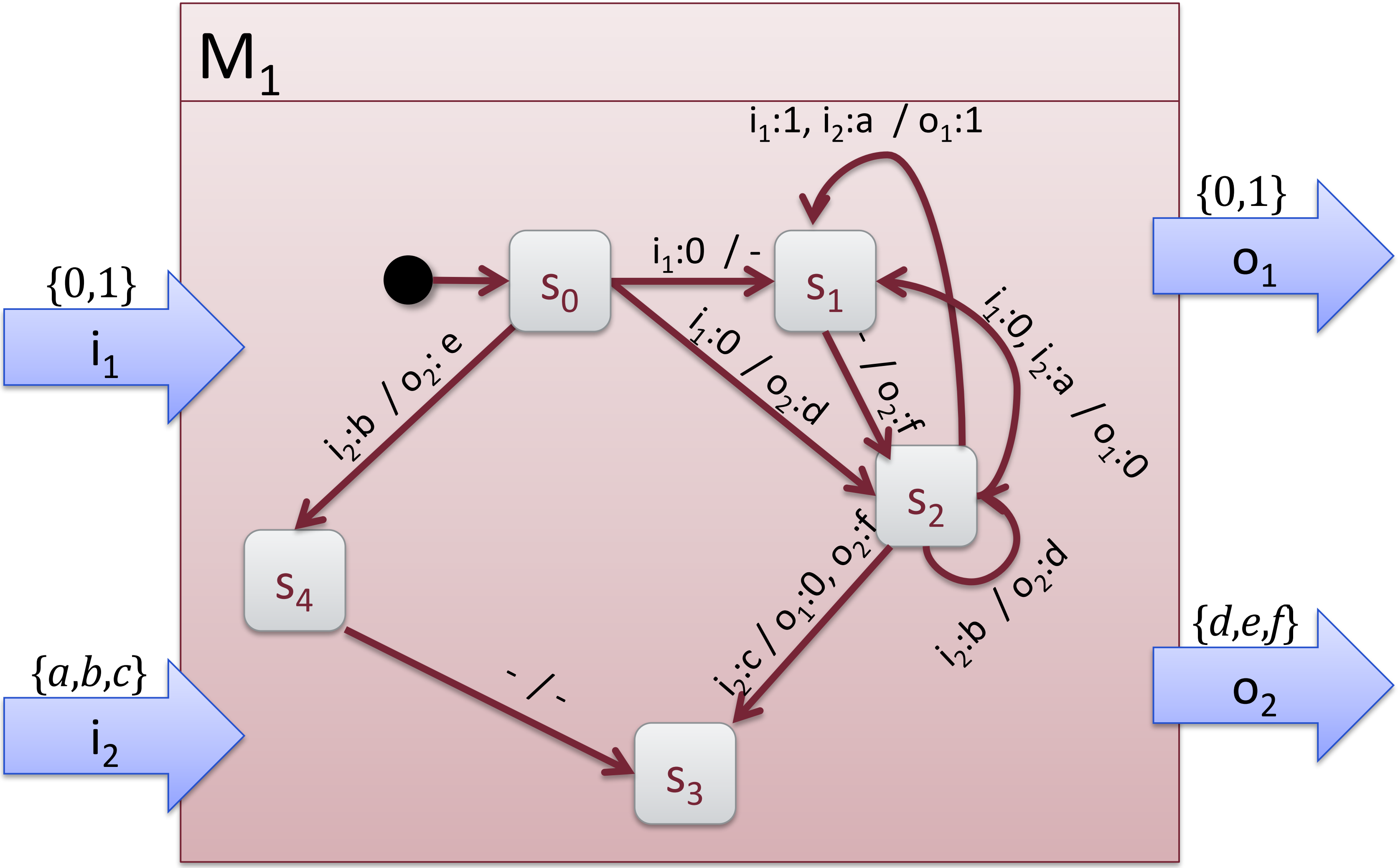


- Compare to automaton model in classic system theory

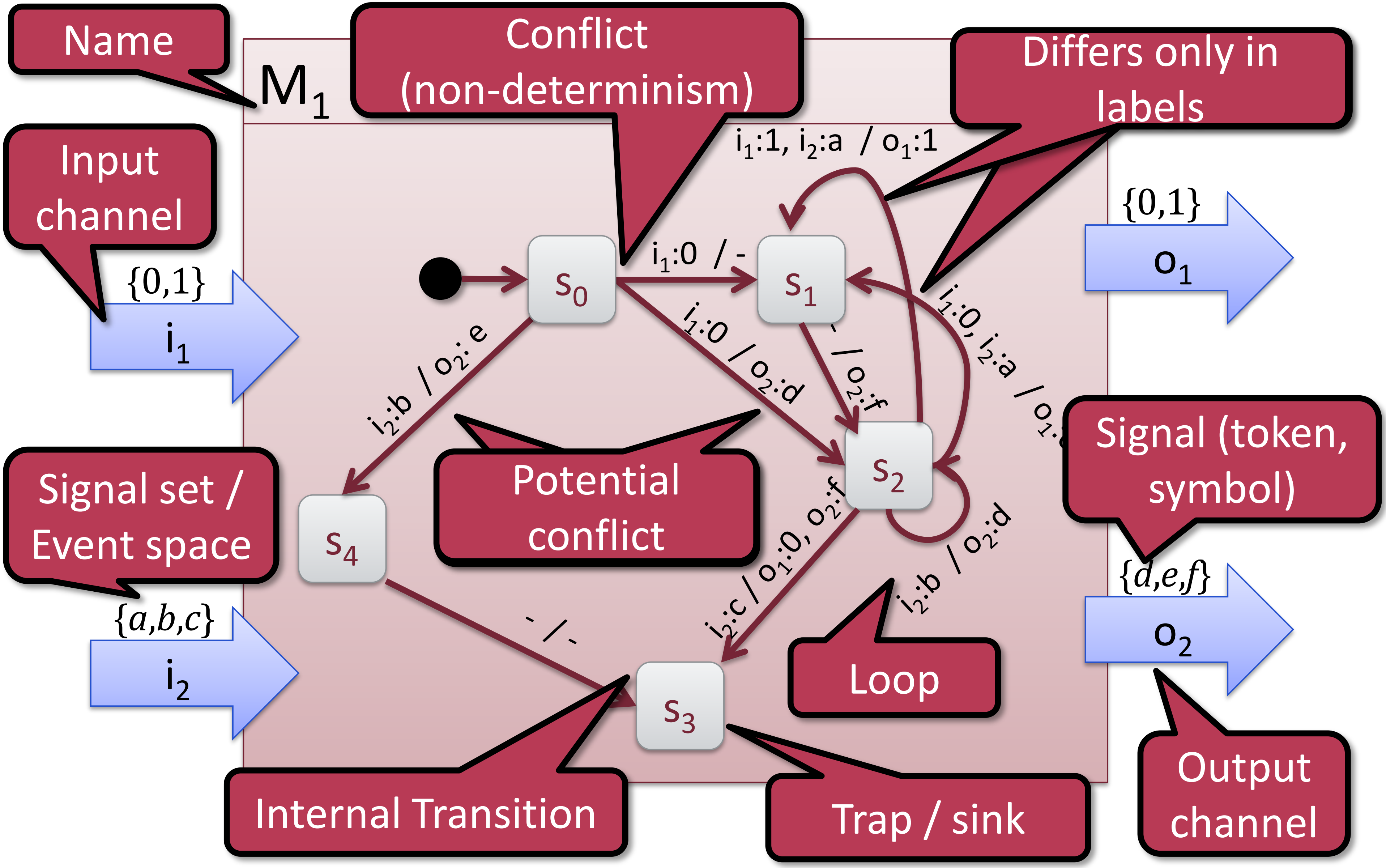
Extensions of Mealy Machine

- Nondeterministic model (wrt. input)
- „Spontaneous” transition without reading input
  - Internal, effect of non-modelled events
  - heating finished, oven switches off (timer not modeled)
- Multiple output channels (separate event streams!)
  - Disjoint signal sets
  - The rule emits signals to a specific subset of channels
- Multiple input channels (separate event streams!)
  - The rule reads input signal from a subset of channels
  - This can cause non-determinism

Extended State Machine



Extended State Machine

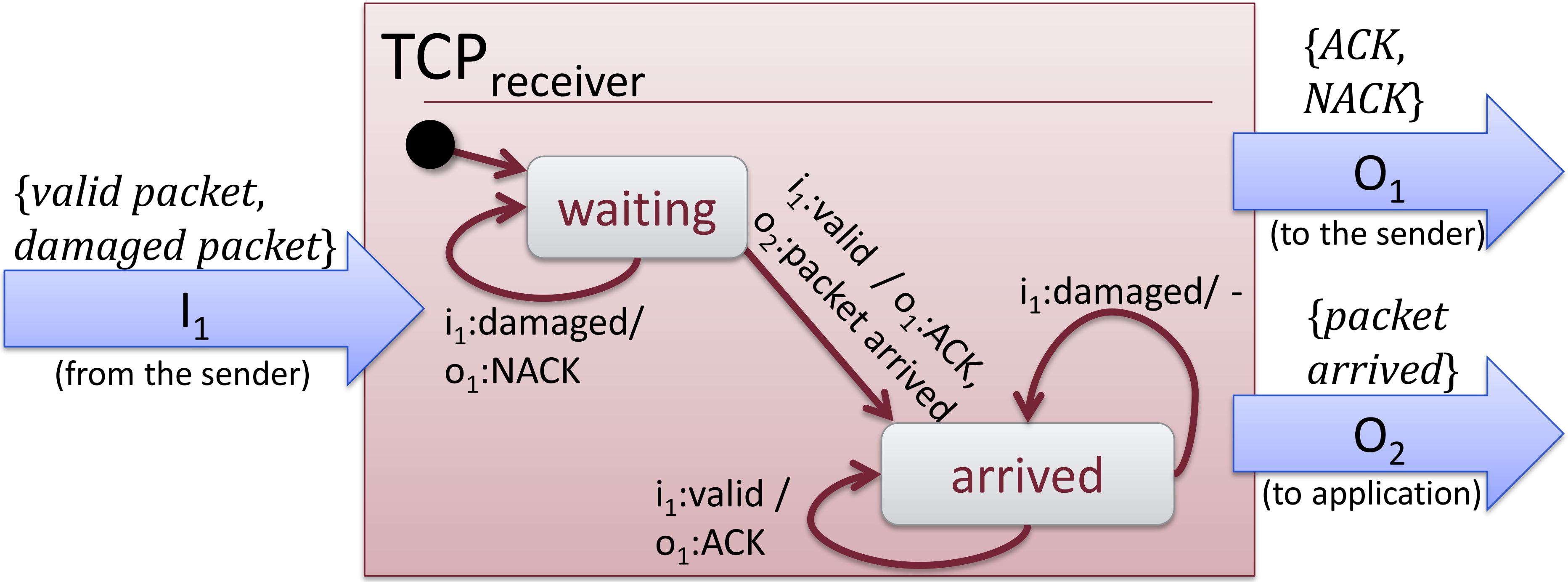


Unspecified Input

- Do we need a rule for every possible input in each state?
  - If no rule → transition is not allowed
    - Theoretically it is so, but for real systems it is not realistic
    - What happens if such a situation occurs anyway?
  - If no rule → invisible loop transition
    - All unspecified input tokens are consumed but ignored
  - If no rule → invalid model
    - E.g. critical embedded system
    - Multiple rules are also invalid (determinism is required)
- If there are rules for each case -> fully specified

Examples: Comm. Protocol

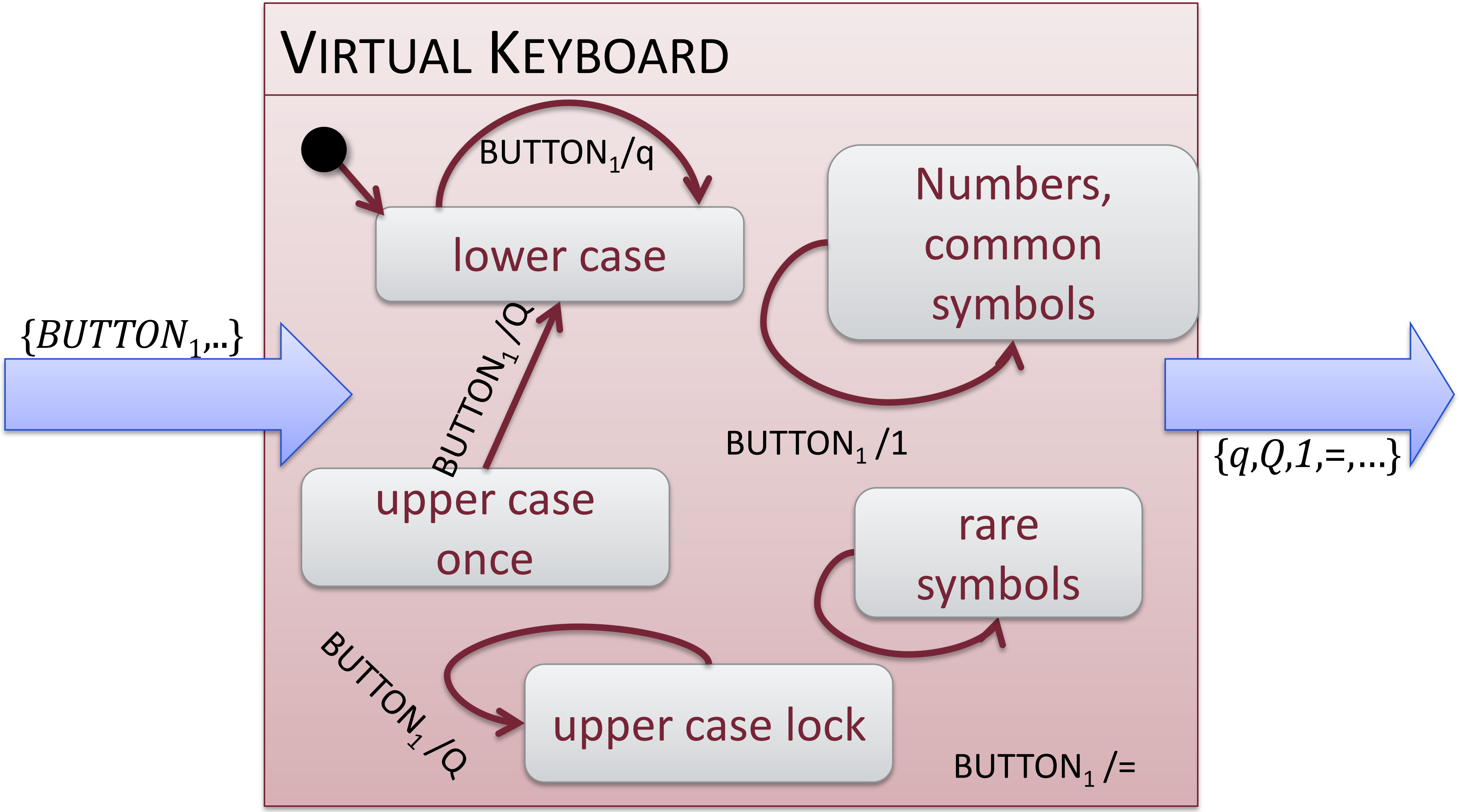
- Packet based communication protocol
  - Packets can be lost, can be damaged
  - Acknowledge, resend



- (For more details see course Computer Networks)

Examples: Virtual Keyboard

- Virtual keyboard



Examples

- Programming: implementing state machine
  - Branching condition:
    - State variables and
    - input
  - All branches:
    - output emission (if any)
    - State transition (if needed)

```
void handleKey(KeyCode input) {
    switch(input) {
    case BUTTON_1:
        switch(keyboardState) {
        case LOWER_CASE:
            emit('q');
            break;
        case UPPER_CASE_ONCE:
            keyboardState = LOWER_CASE;
        case UPPER_CASE_LOCK:
            emit('Q');
            break;
        case NUMBERS_COMMON_SYMBOLS:
            emit('1');
            break;
        case RARE_SYMBOLS:
            emit('=');
        }
        break;
    case SWITCH_1:
        //...
```

Remark

- If the (state machine) model is…
  - … detailed enough (deterministic), and
  - … formalized in a way, which can be processed
    - E.g. domain specific languages (protocol design)
    - E.g. standard modeling notation (UML)
- … can be translated automatically into source code
  - E.g. code generation for a communication protocol
  - E.g. development of an embedded controller
- … or can be executed with an interpreter
  - E.g. IT system management application

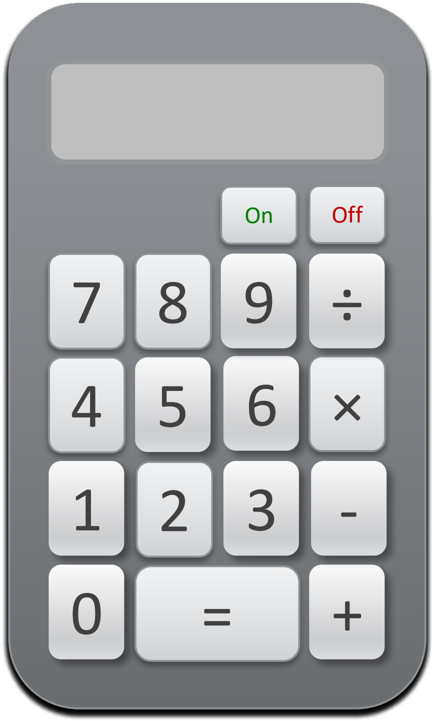
Behavioural Modelling → State Partitioning → Simple (Mealy) Automata → **State Machine Extensions** → Outlook

STATE MACHINE EXTENSIONS

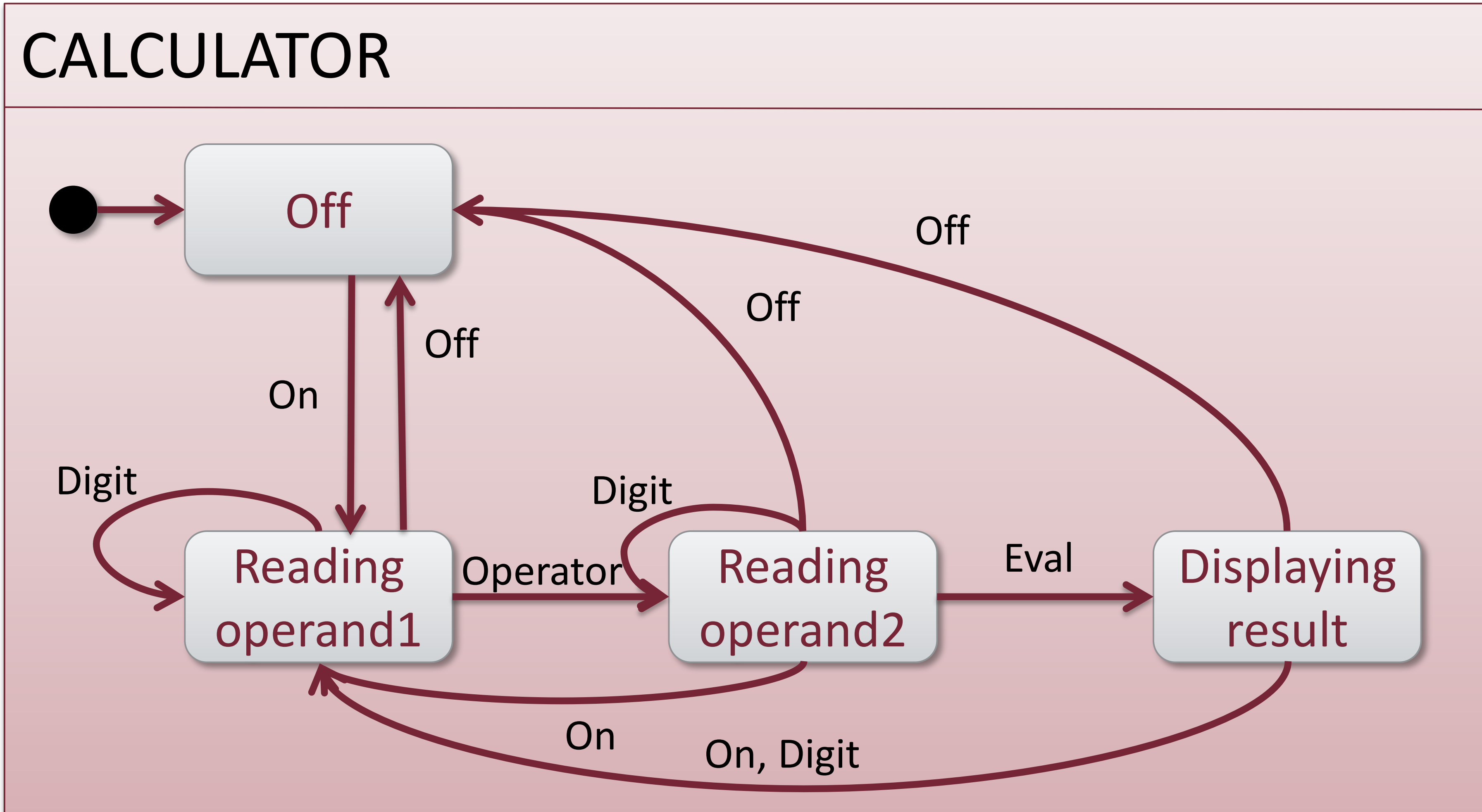
State Chart Languages

- (Harel) State Chart = state machine +
  - State hierarchy
  - Orthogonality
  - Variables
  - Pseudo states
  - …
- E.g.
  - Yakindu
  - UML

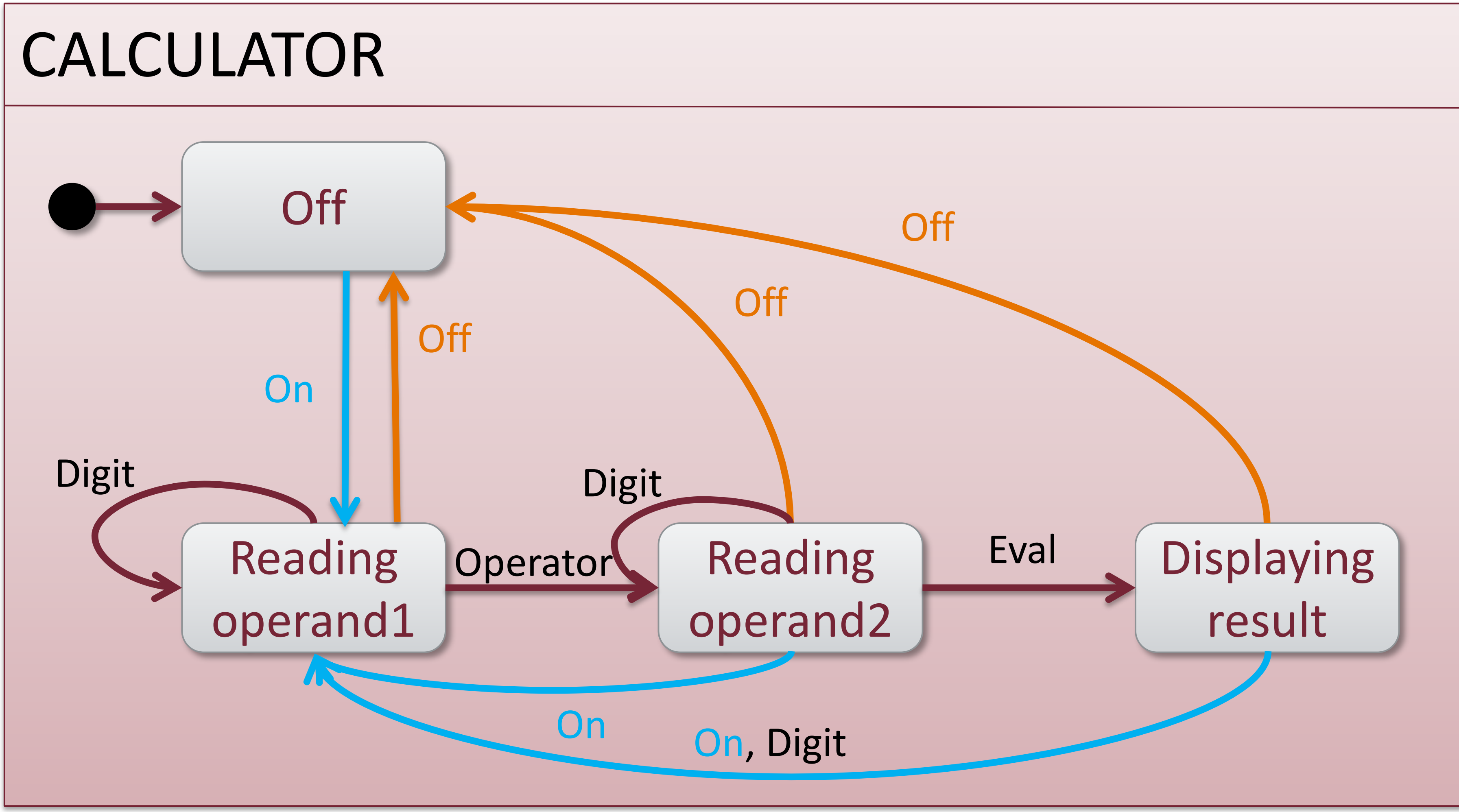
Calculator Example



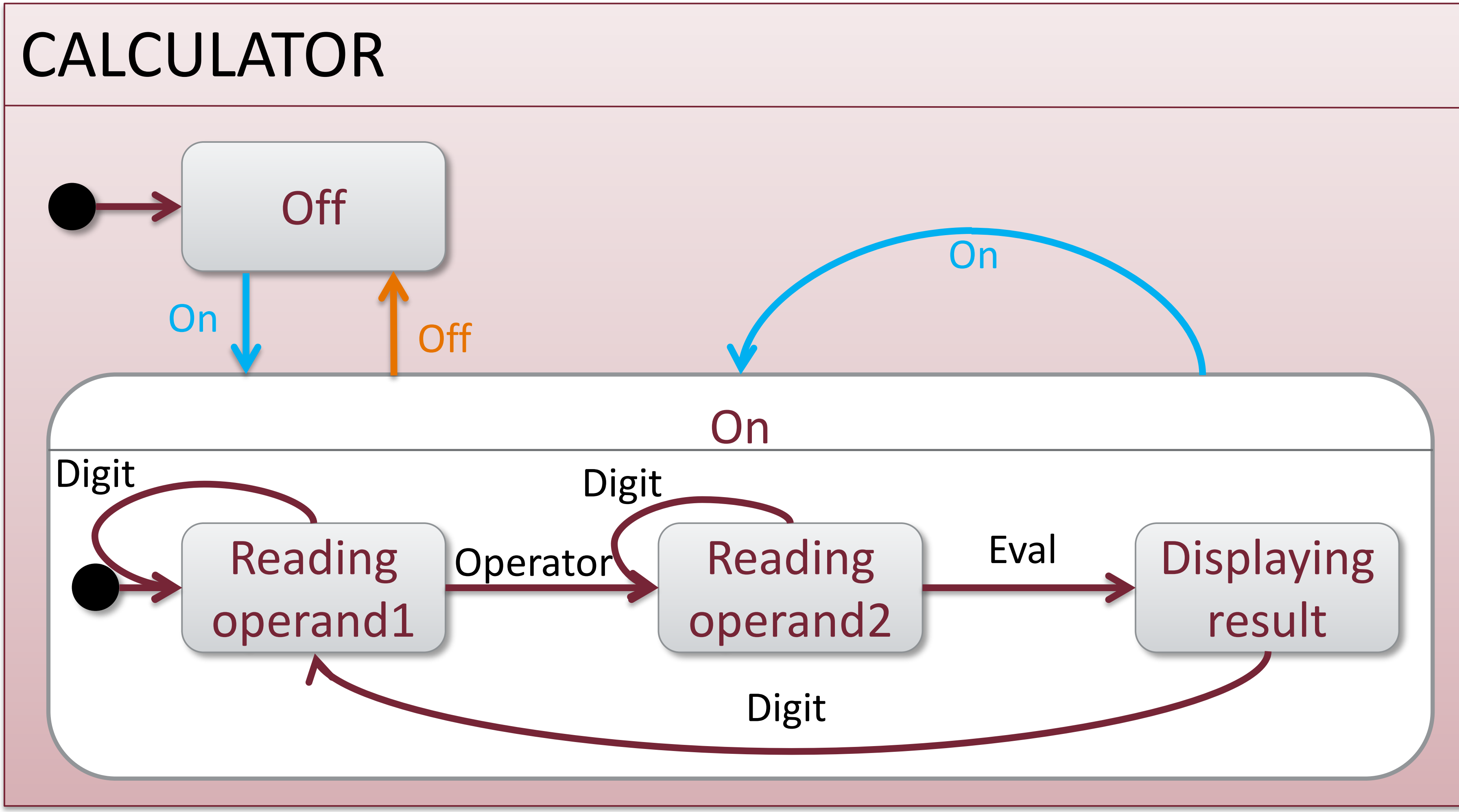
State Hierarchy



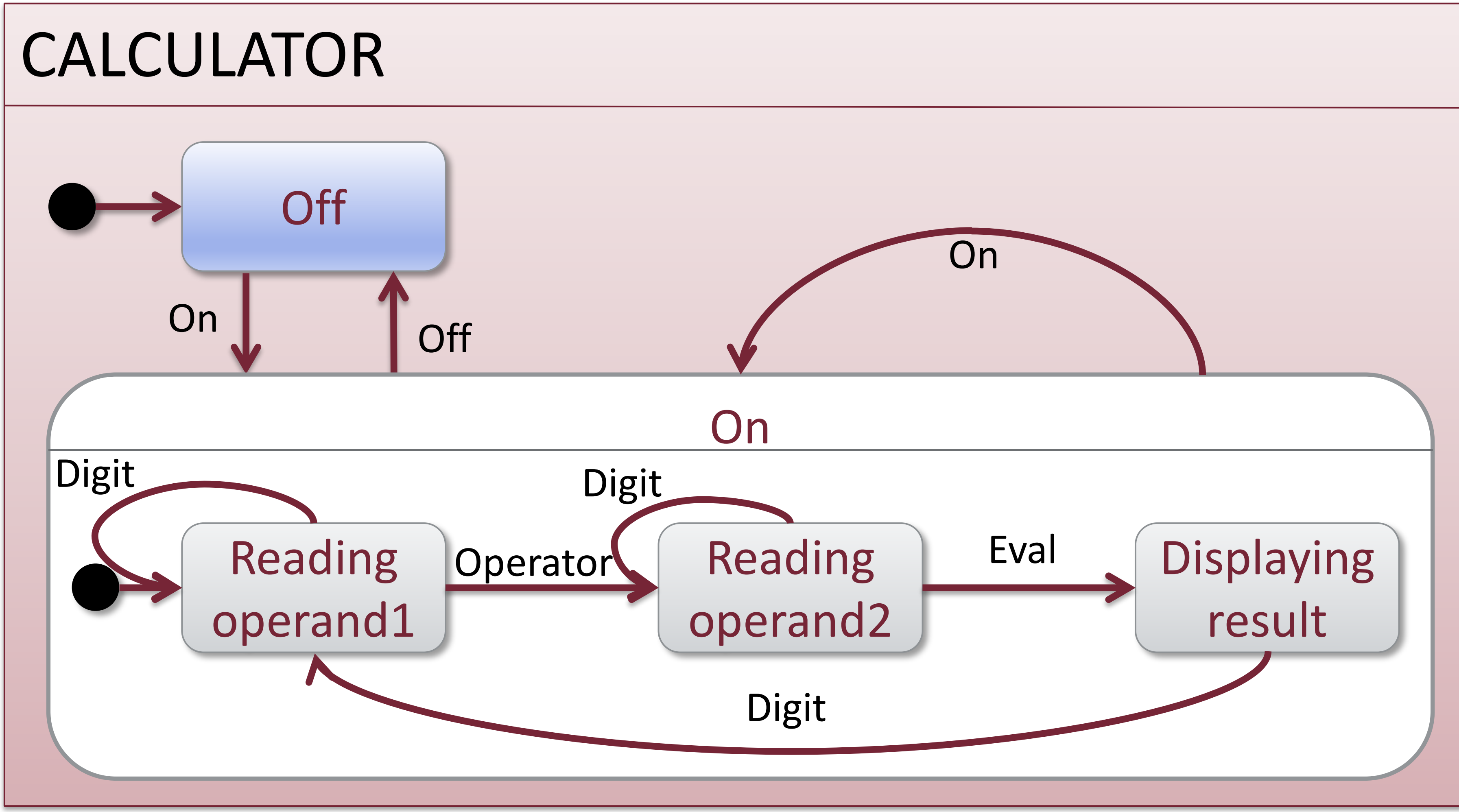
State Hierarchy



State Hierarchy

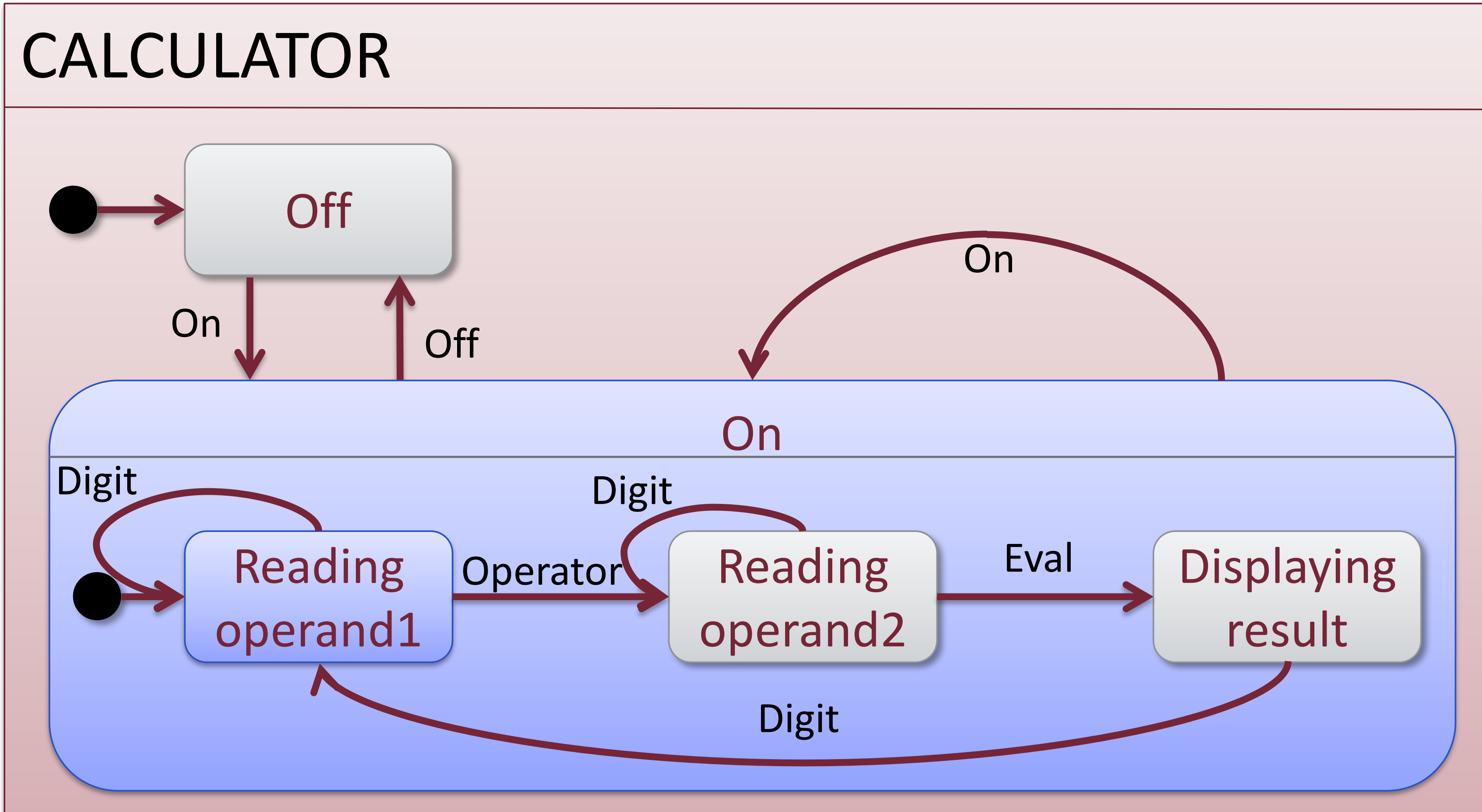


State Hierarchy



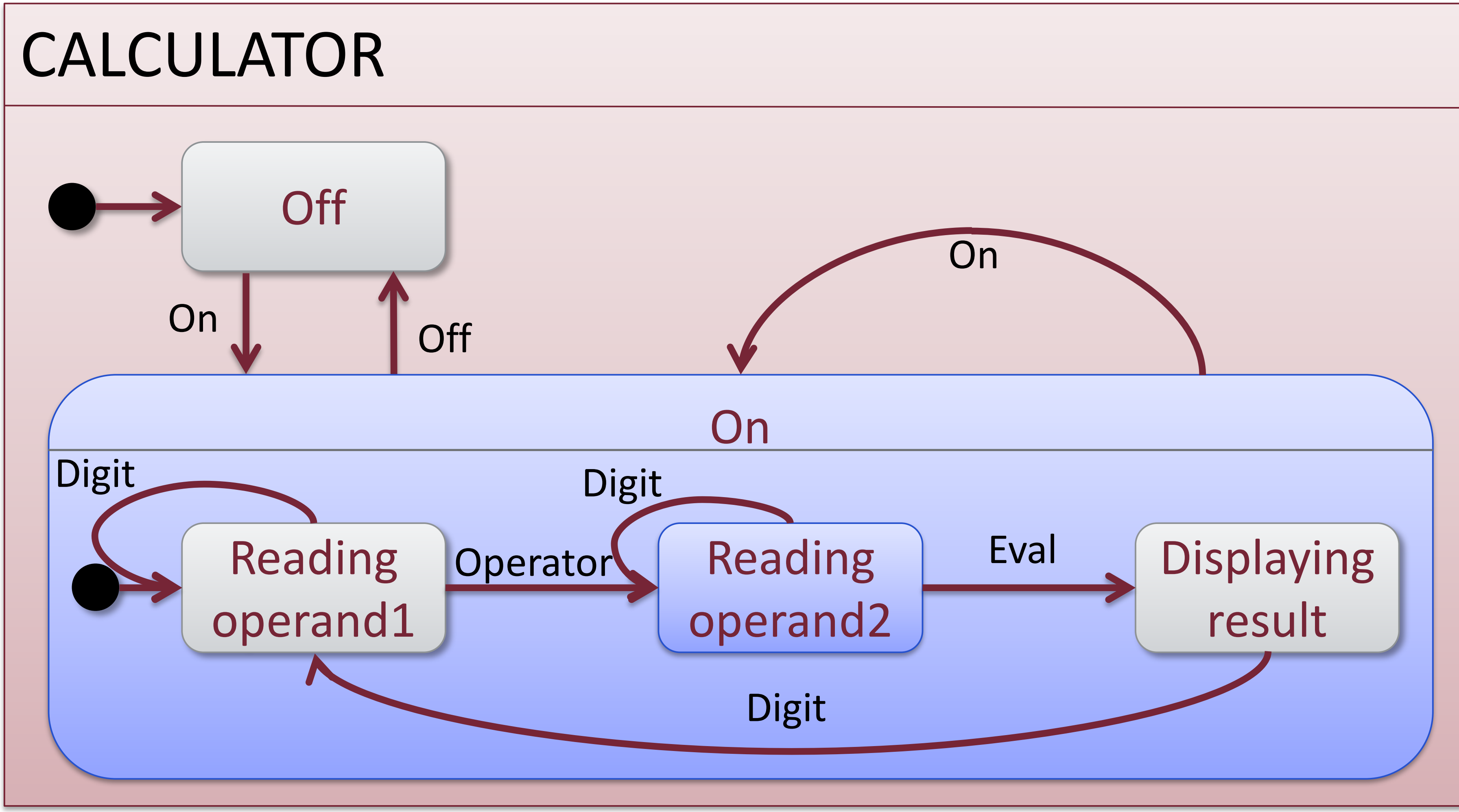
- State configuration: $\{Off\}$

State Hierarchy



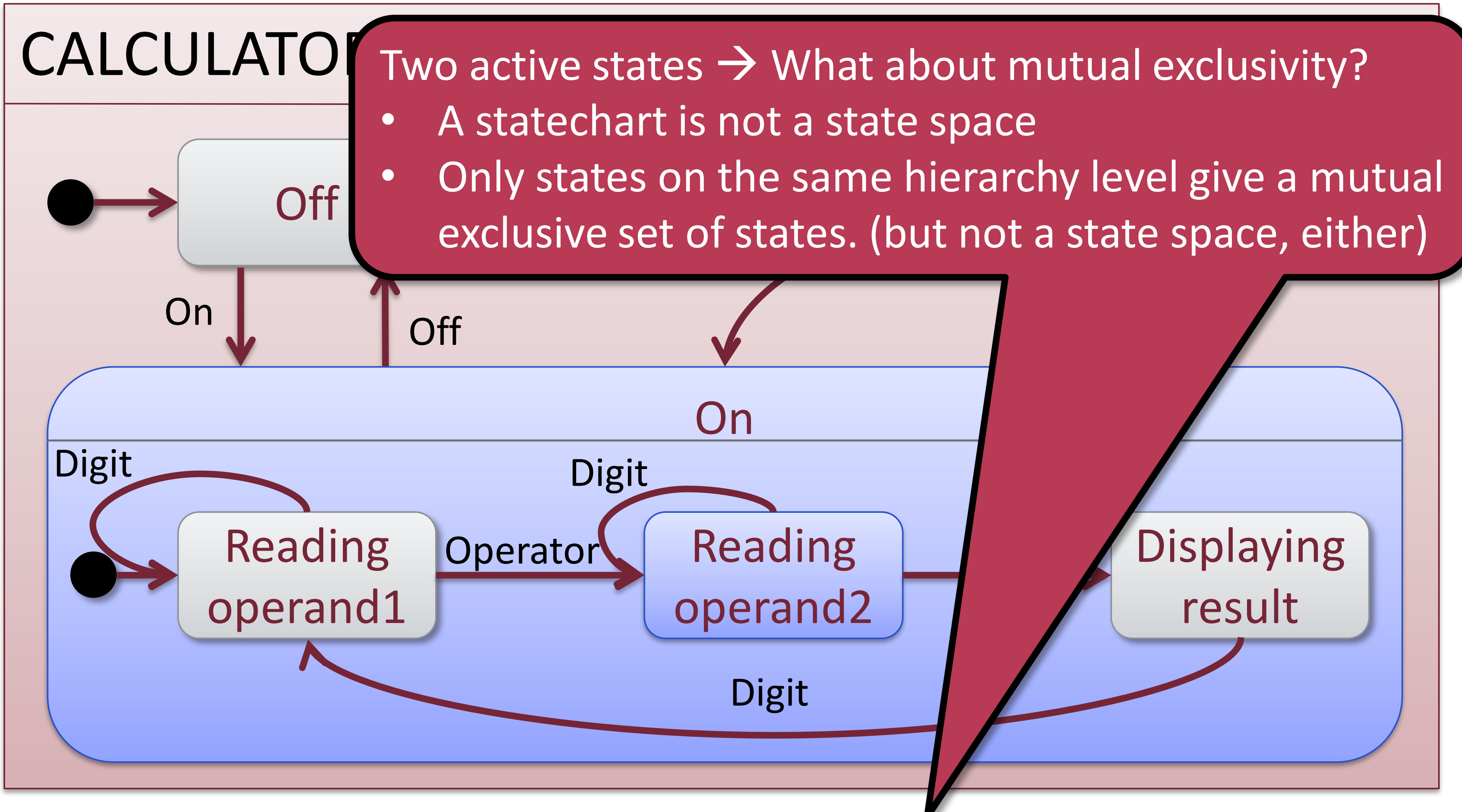
- State configuration : $\{On, Reading\ operand1\}$

State Hierarchy



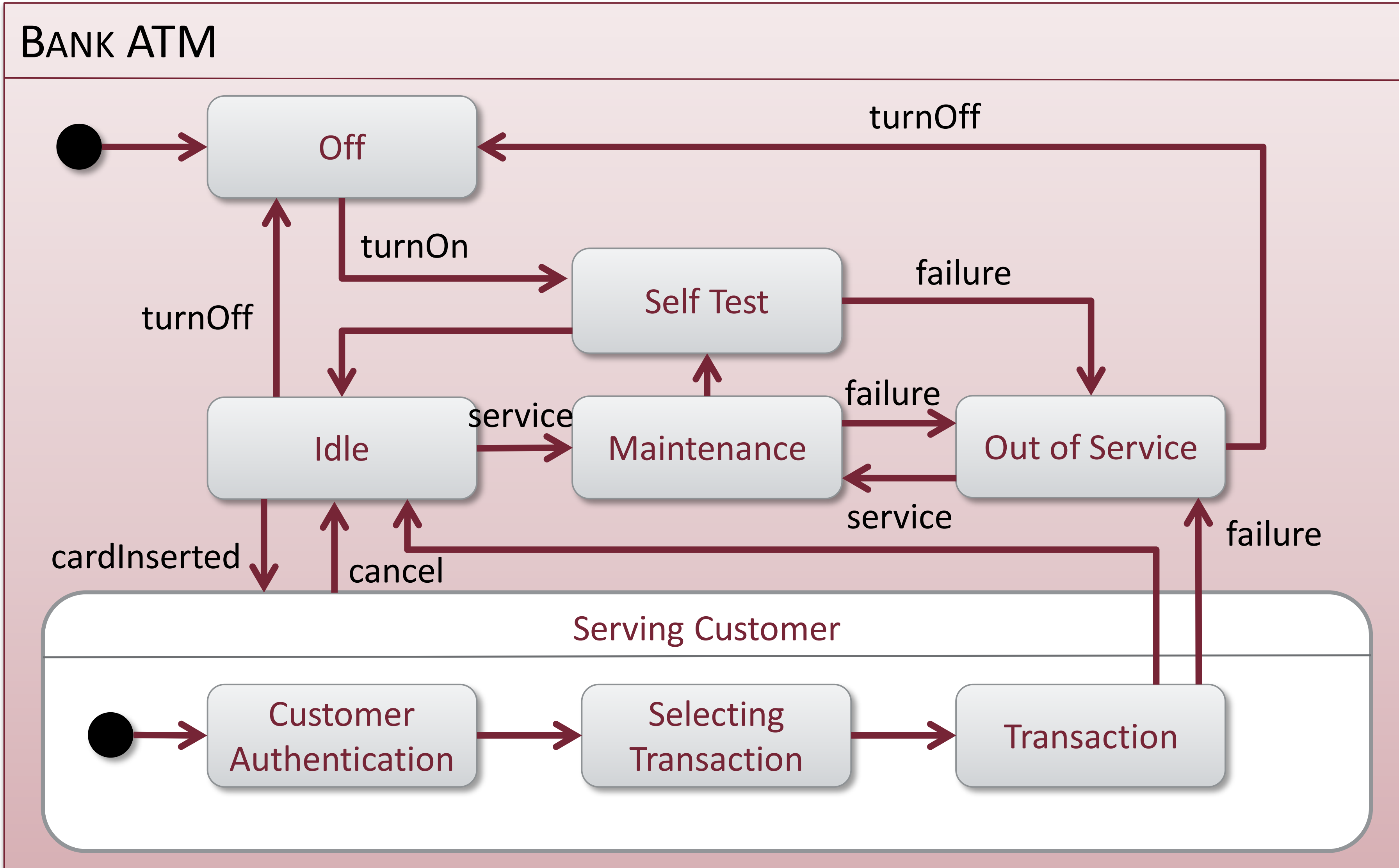
- State configuration : $\{On, Reading\ operand2\}$

State Hierarchy



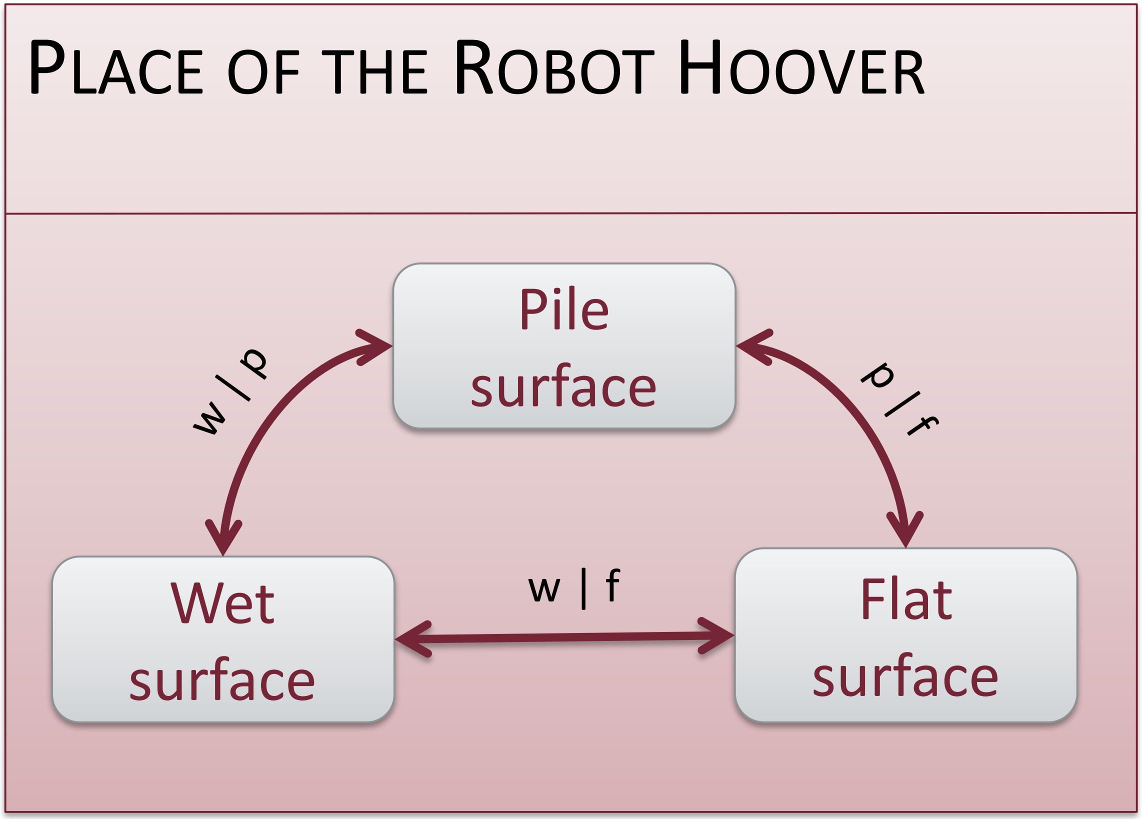
- State configuration : $\{On, Reading\ operand2\}$

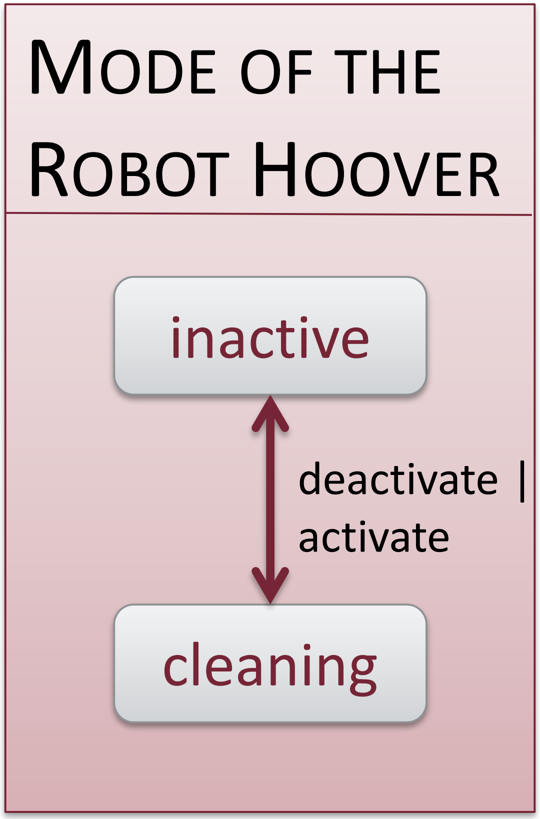
ATM Example


BANK ATM
Off
turnOff
turnOn
Self Test
failure
turnOff
Idle
service
Maintenance
failure
Out of Service
service
failure
cardInserted
cancel
Serving Customer
Customer Authentication
Selecting Transaction
Transaction

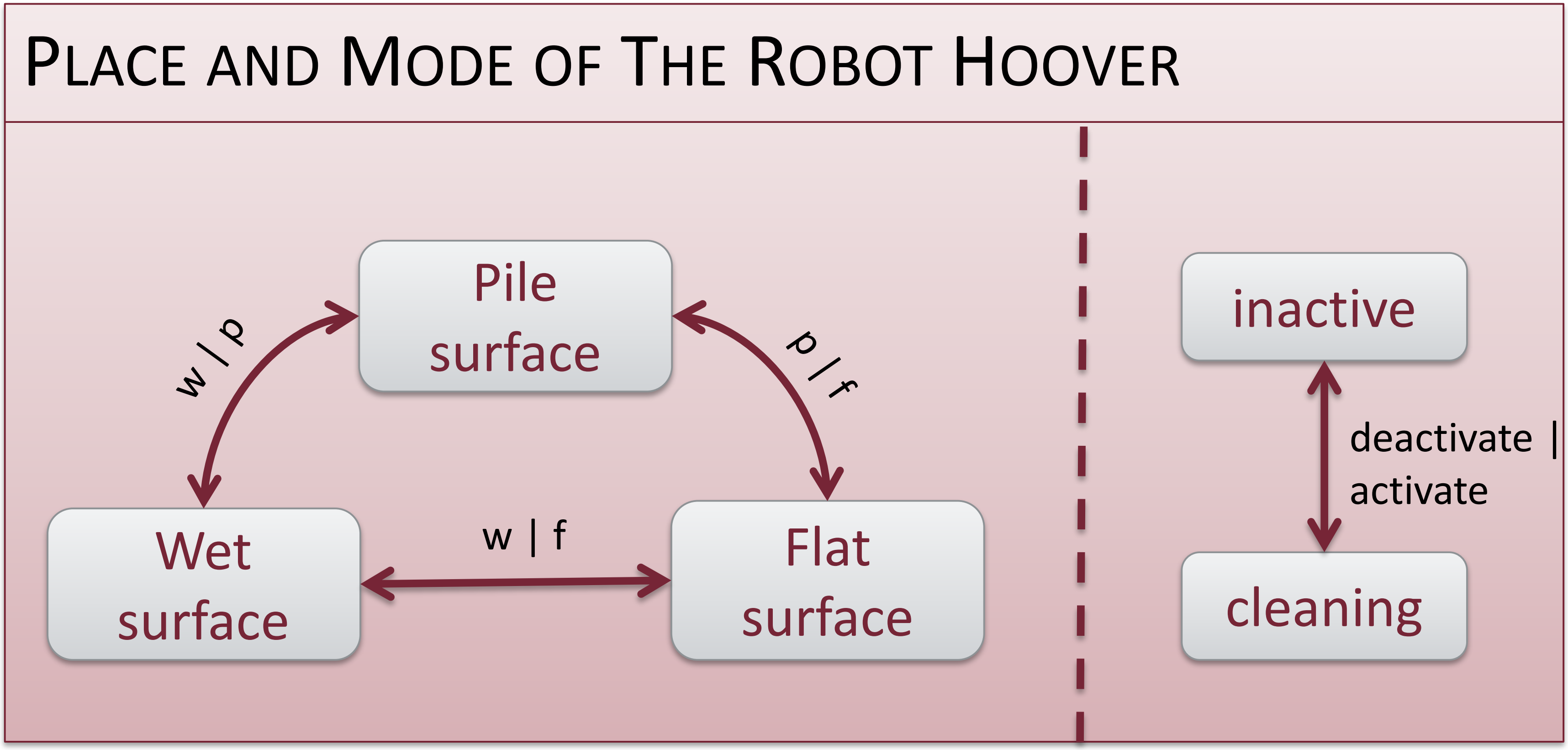
Example: robot hoover

Orthogonality

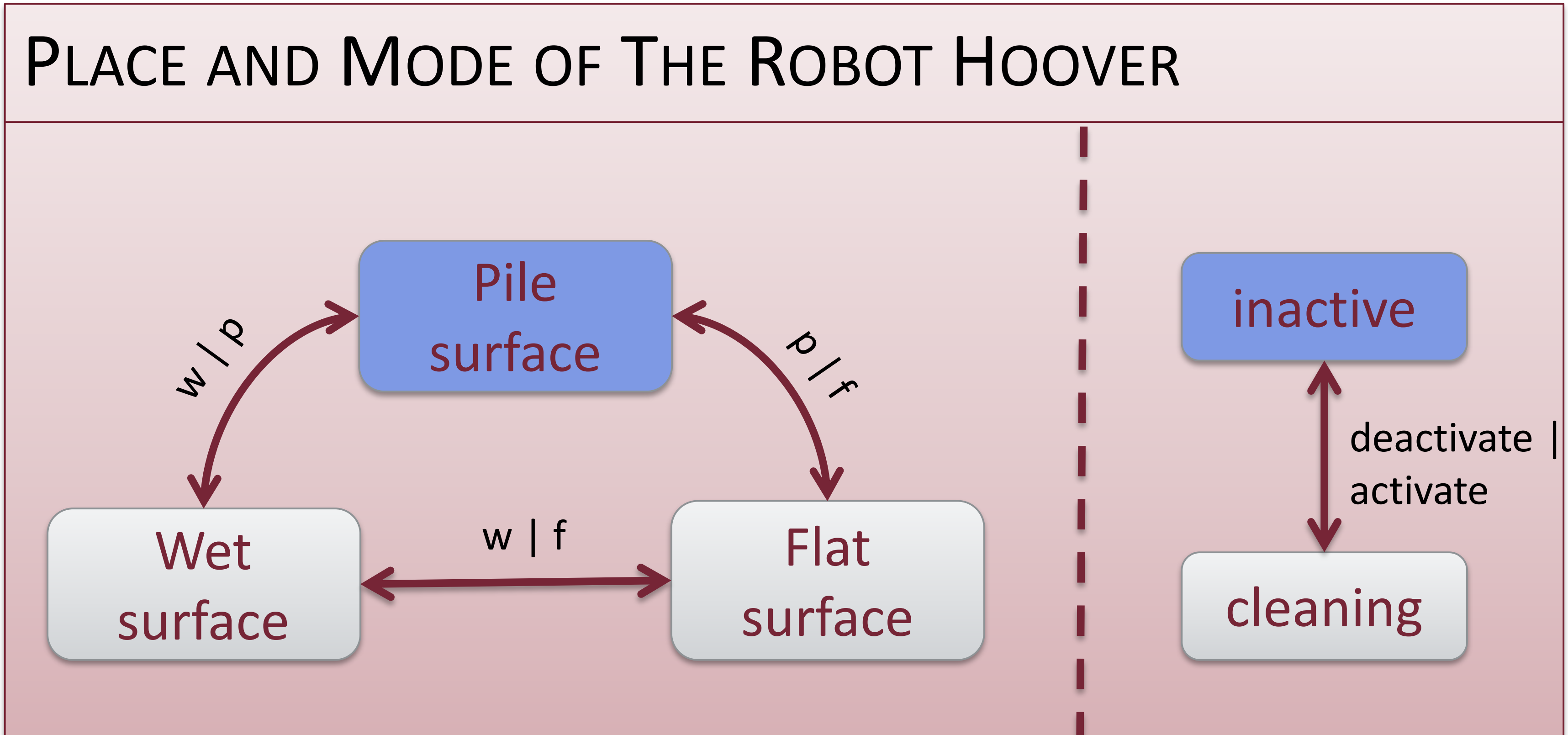




Orthogonality

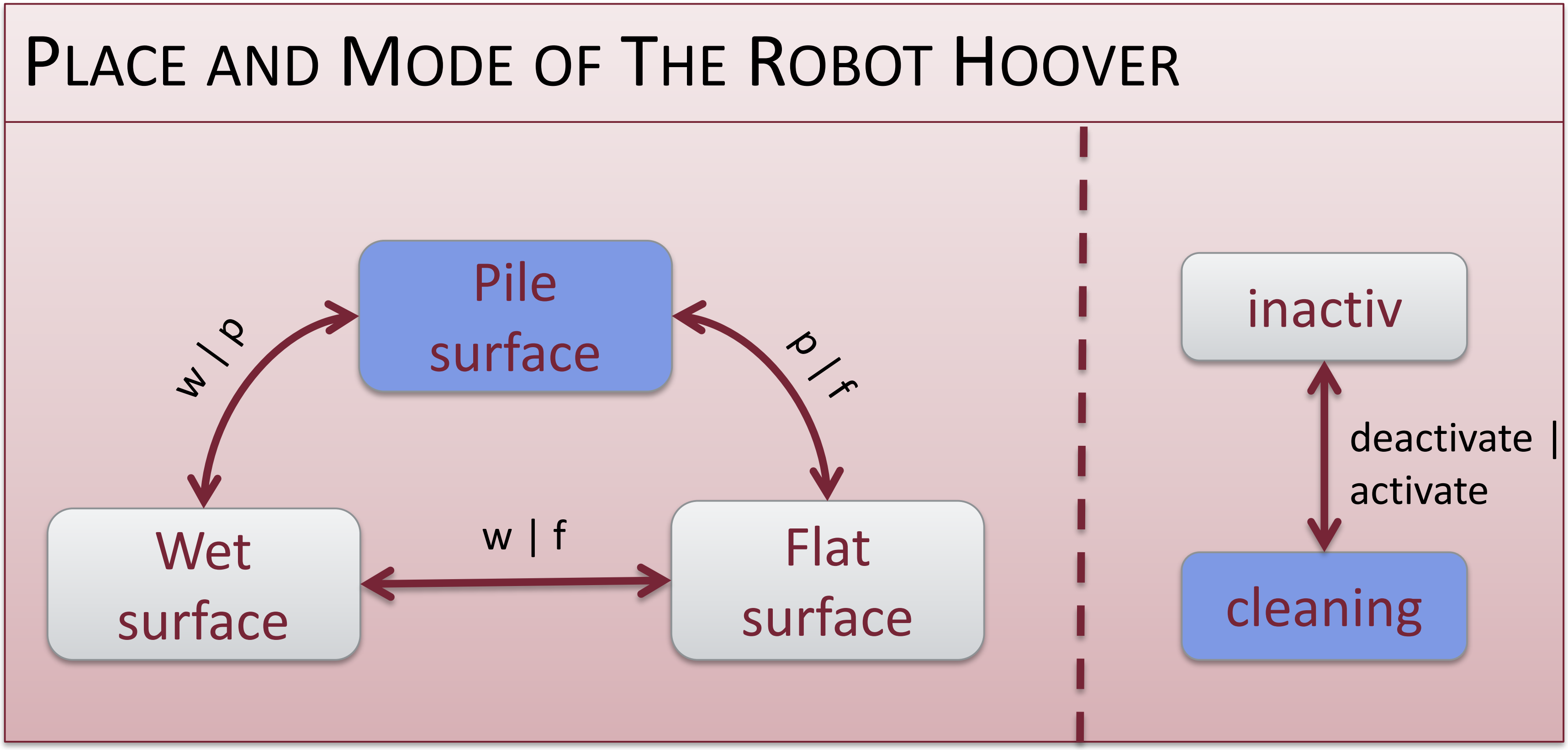


Orthogonality



- State configuration: {*pile surface, inactive*}

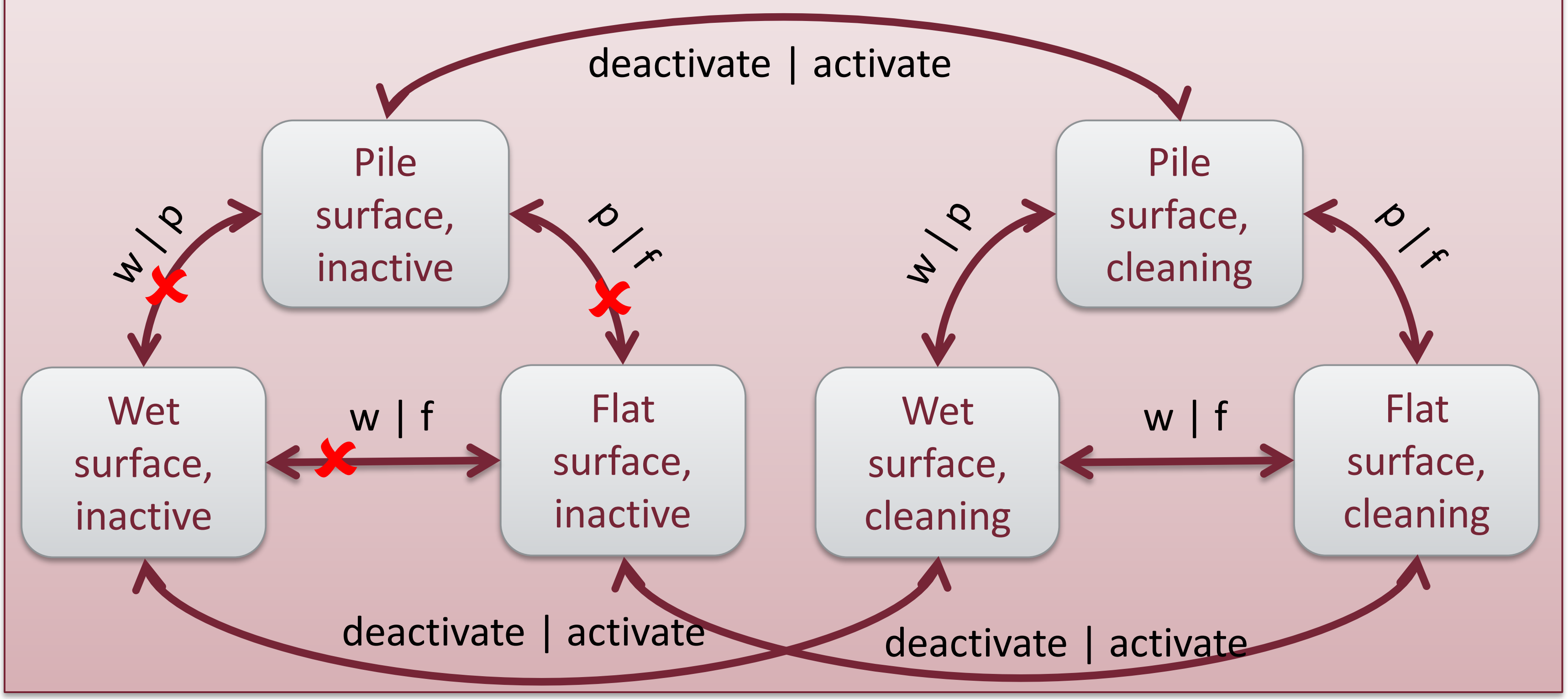
Orthogonality



- State configuration: {*pile surface, cleaning*}

Asynchronous Product

Place and Mode of The Robot Hoover

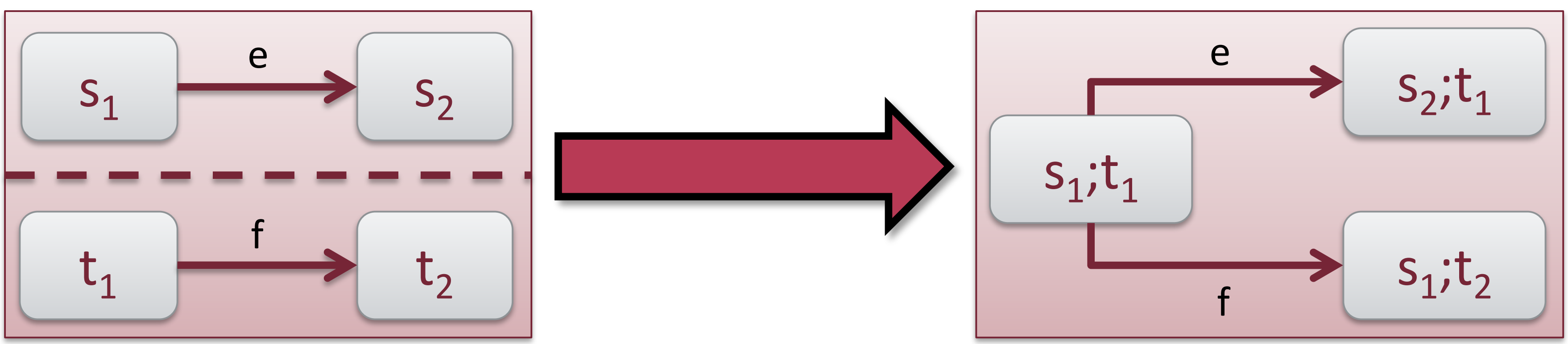


- Needs further refinements: Transitions are excluded → States may become unreachable

Definition: Asynchronous Product

The **asynchronous product** of (Mealy-) state machines is a **composition operation** over the component state machines (also called the regions**.** The **result** of the composition is a (Mealy-) state machine.

- State space: the direct product of the state spaces of the regions
- Initial state: every region in its initial state
- Transition rules: all transition rules in which
  - **exactly one region** makes a transition,
  - while all other regions keep their actual states.

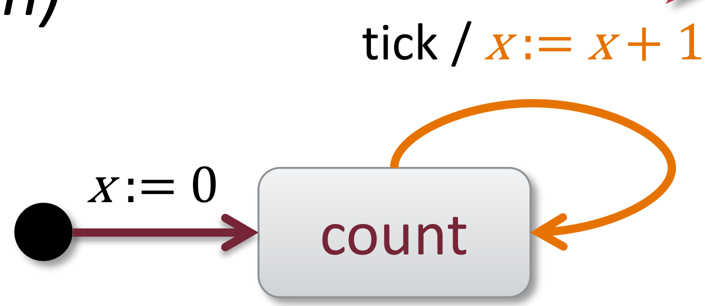


Variables

- Infinite counter
  - $S = \mathbb{N}$

Actually, *x* can be considered as being in an other region...

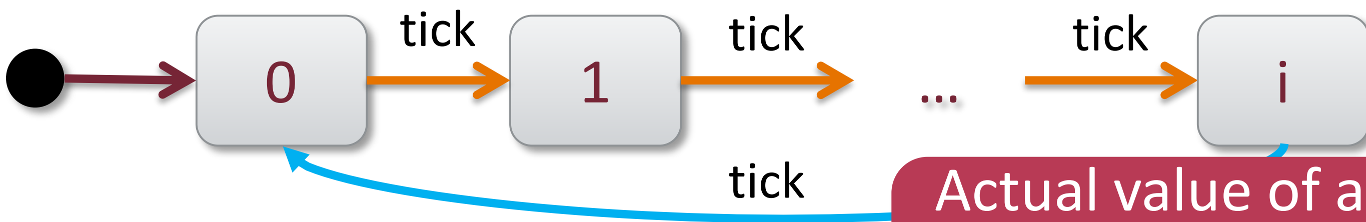
- Introduction of variables: x
  - *(like a separate region)*



$$(count, \{x \mapsto 0\}) \rightarrow (count, \{x \mapsto 1\}) \rightarrow \ \ldots$$

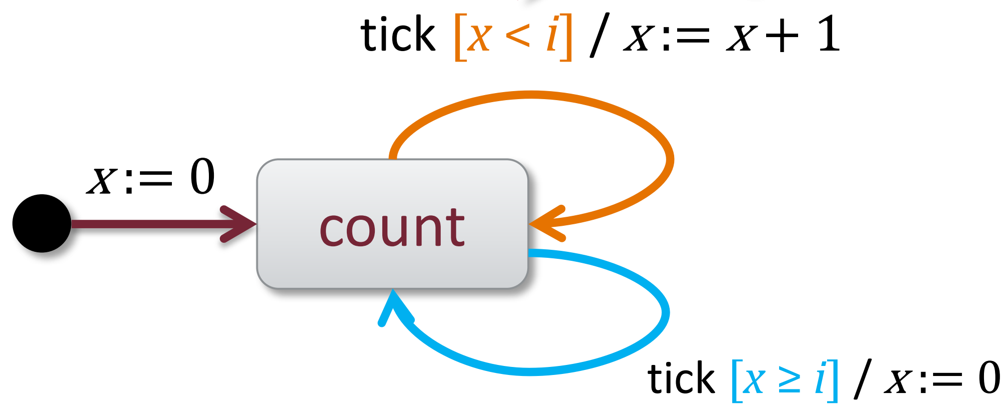
Variable + Guard Condition

- Cycle counter
  - $S = \{0, 1, \dots, i\}$



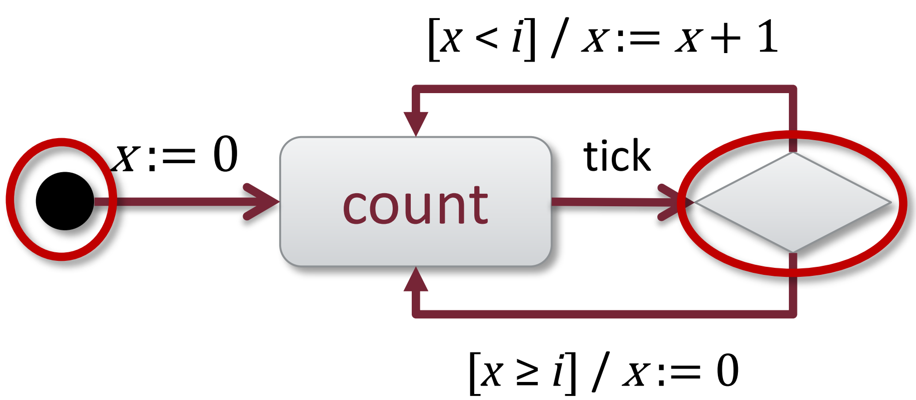
Actual value of a variable or actual state of an other region can be referred

- With Guard Conditions:

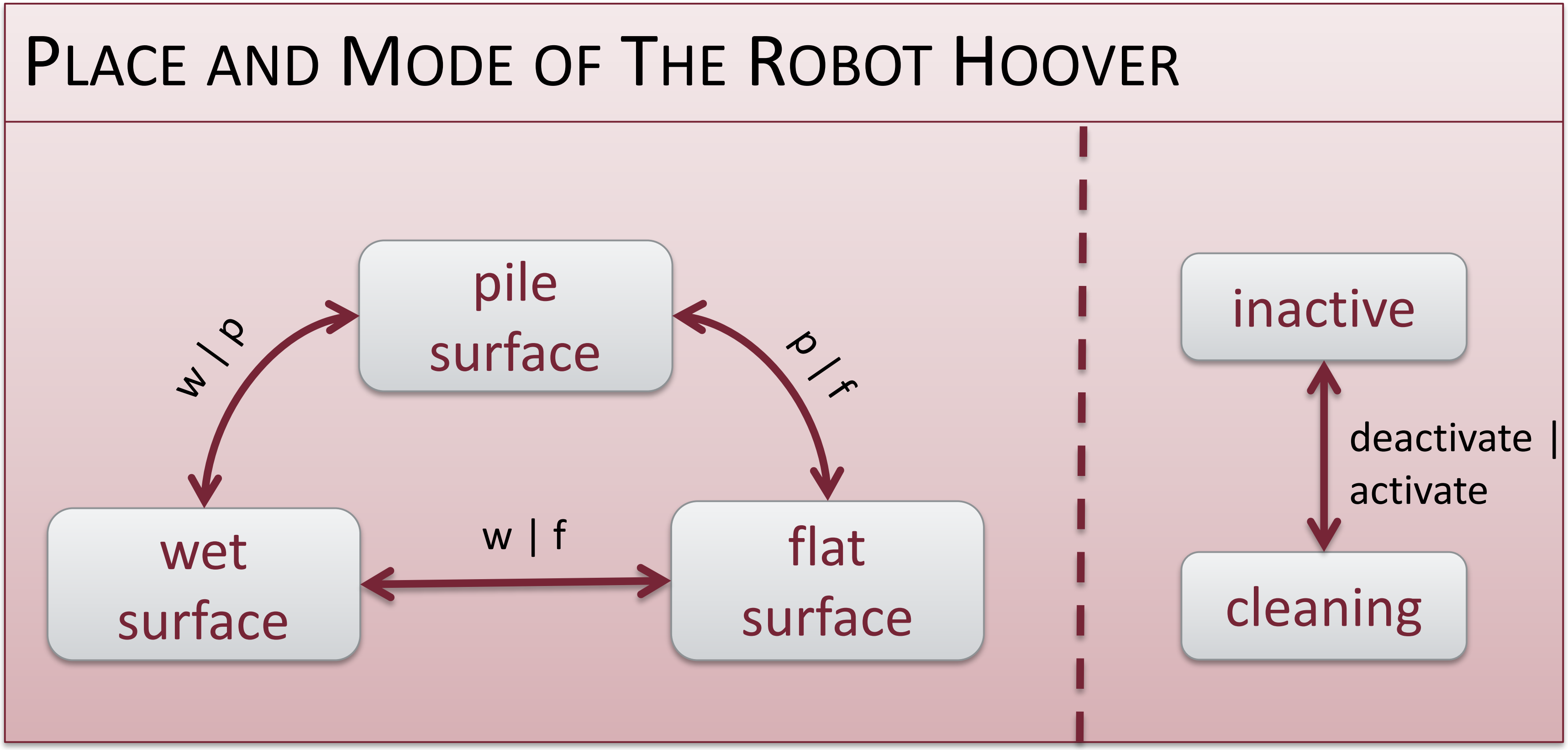


Pseudo States

- Pseudo state:
  - Semantically it is not a state:
    - There is no time instant when it represents the state of the system
  - Syntactically it is a state:
    - Can be the start or the end state of a transition

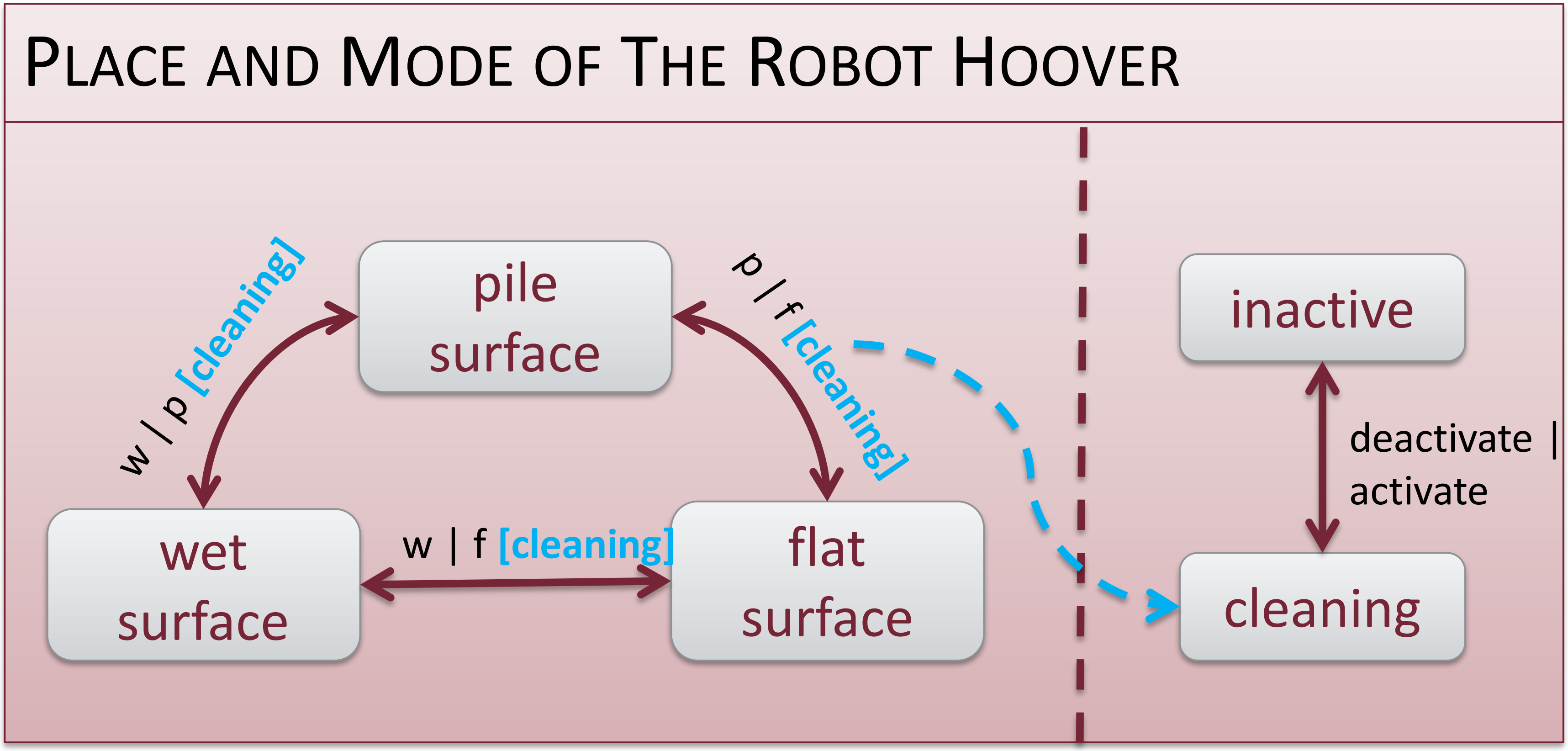


Asynchronous Product: Cooperation



- How can we model cooperation?
  - How exactly are the two regions **not independent**?

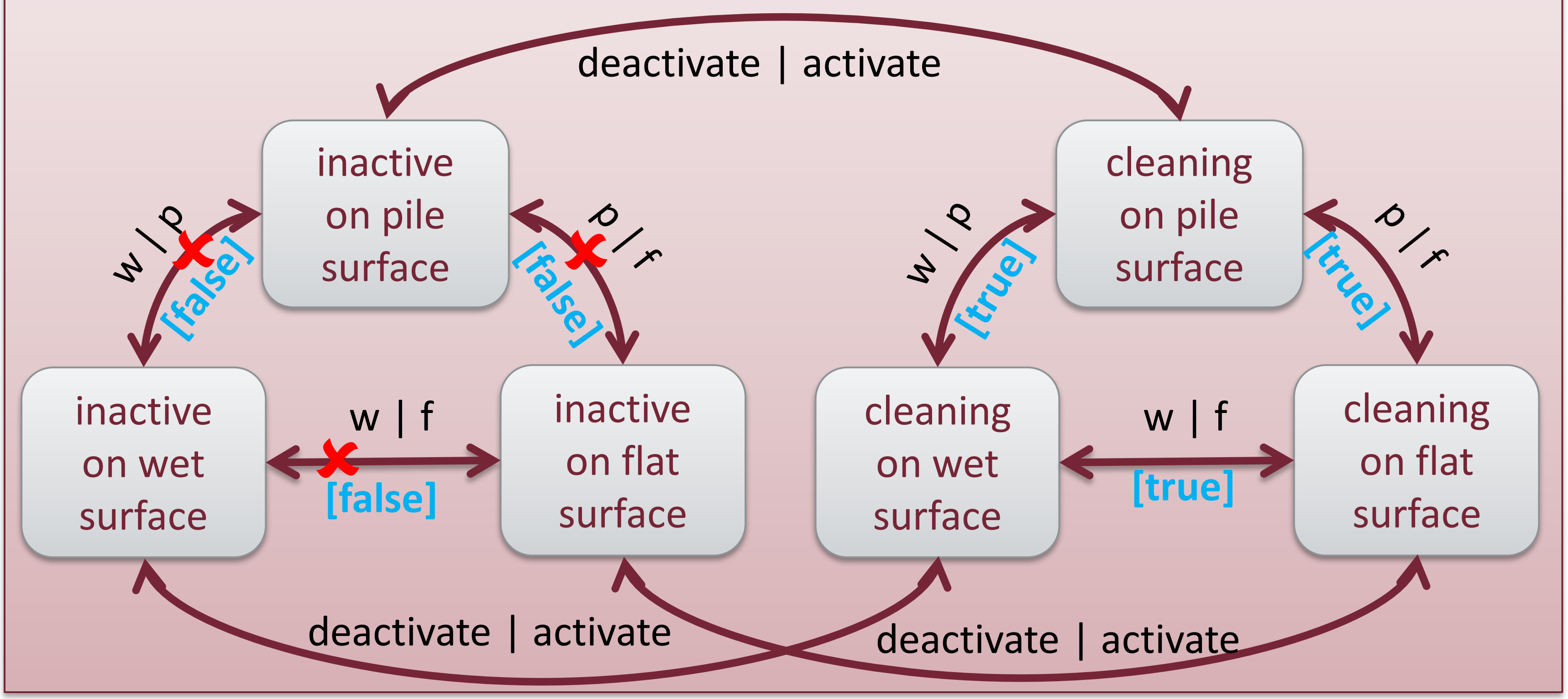
Asynchronous Product: Cooperation



- Cooperation by **guards**
  - The condition of a transition in one region is a state in the other region.

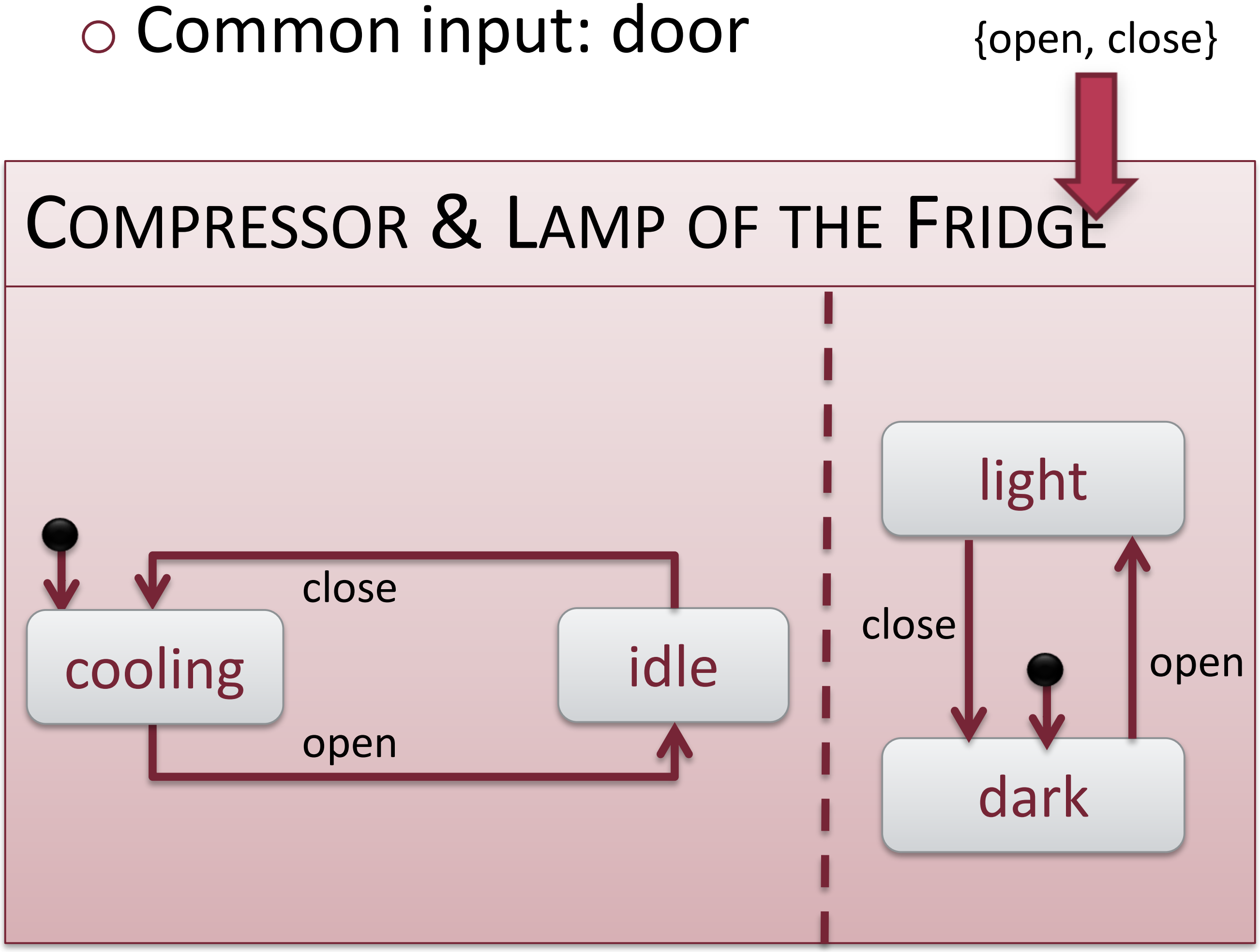
Asynchronous Product: Cooperation

PLACE AND MODE OF THE ROBOT HOOVER



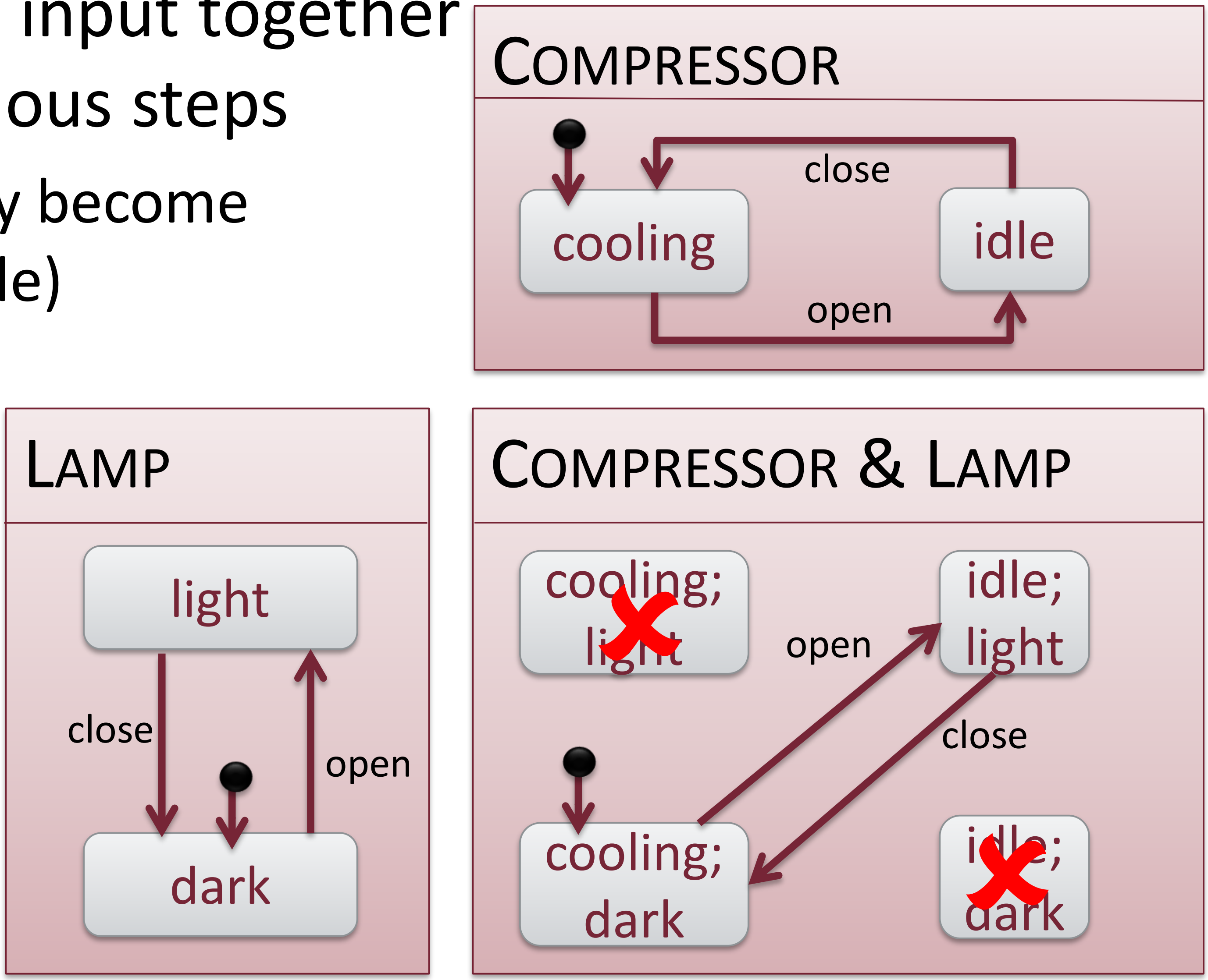
Example: Synchronous Product

- Fridge with compressor & lamp
  - Common input: door



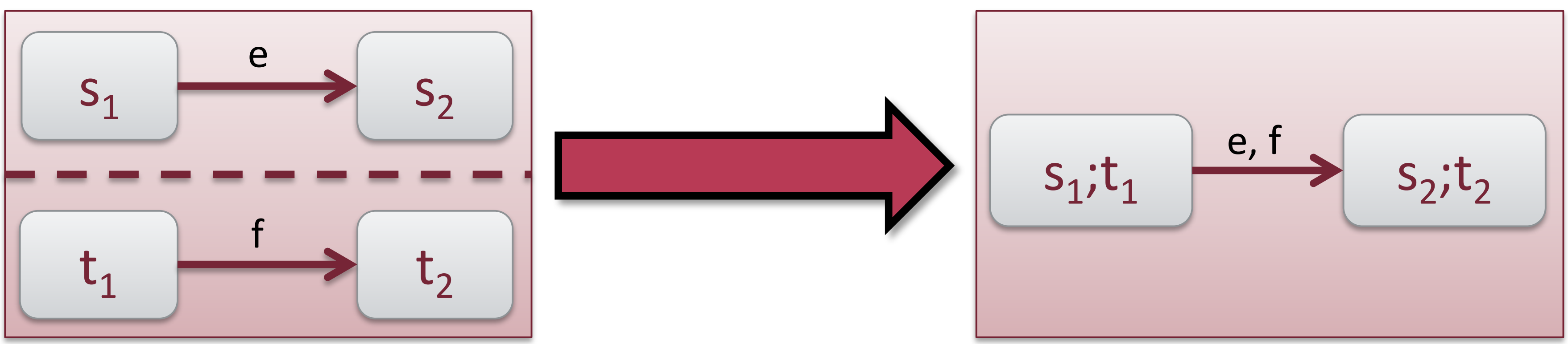
Example: Synchronous Product

- Reading the input together → synchronous steps
  - (States may become unreachable)



Synchronous Product

- Product of state machines
  - Composing the model from state regions (components)
  - State space: **direct product** of the region state spaces
  - Initial state: n-tuple of the initial states of every region
- Transition rules: all transition rules in which
  - each region makes a transition **at the same time**.
  - „gluing" transitions, union of the labels



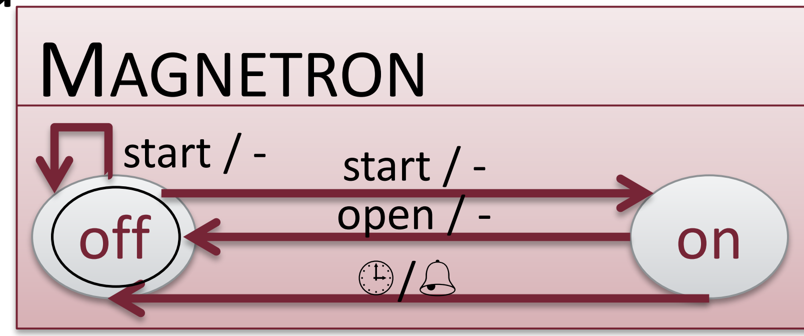
Mixed Product

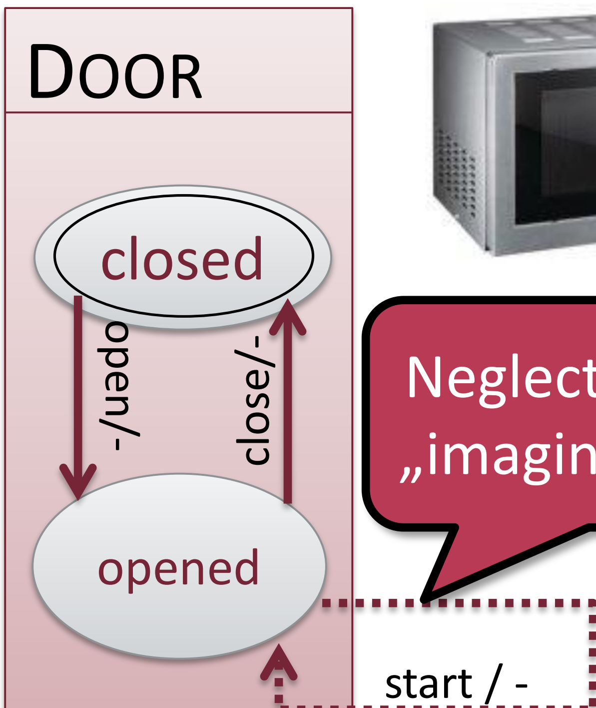
- Product of state machines
  - Composing the model from state regions (components)
  - State space: **direct product** of the region state spaces
  - Initial state: n-tuple of the initial states of every region
- Transition rules: sometimes synchronised
  - Basically an asynchronous composition …
  - … but in some cases the regions act together

A simple case of synchronisation: there are (also) common inputs

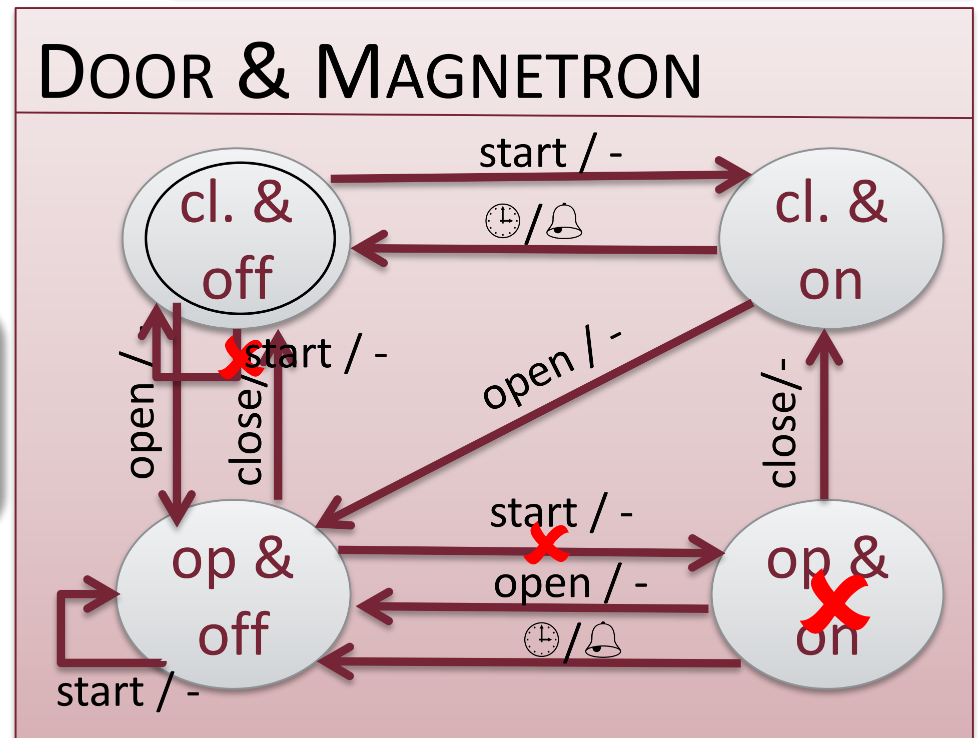
Example: Mixed Product

- Further refinement required
  - Transitions may vanish
  - (States may become unreachable)





Neglected inputs:
„imaginery" loops



Mixed Product

- Product of state machines
  - Composing the model from state regions (components)
  - State space: **direct product** of the region state spaces
  - Initial state: n-tuple of the initial states of every region
- Transition rules: sometimes synchronised
  - Basically an asynchronous composition …
  - … but in some cases the regions act together

A simple case of synchronisation: there are (also) common inputs

Advanced cooperation: rendezvous (internal synchronising event)

Overview of Products

- Product of state <u>spaces</u>
  - **Direct product**: $S_1 \times S_2 \times \ldots \times S_n$
    - Composed states: n-tuples of the states of the components
- Product of state <u>machines</u>
  - The state space is always (refinement of) the direct product
  - **Synchronous product**
    - The components/regions step always at the same time
  - **Asynchronous product**
    - The components/regions step always one at a time
  - **Mixed product**
    - Sometimes at the same time and sometimes one at a time

Overview of Products

- Product of state spaces
  - **Direct product**: $S_1 \times S_2 \times \ldots \times S_n$
    - Composed states: n-tuples of the states of the components
- Product of state machines
  - The state space is always (refinement of) the direct product
  - **Synchronous product**
    - The components/regions step always at the same time
  - **Asynchronous product**
    - The components/regions step always one at a time
  - **Mixed product**
    - Sometimes at the same time and sometimes one at a time

Modes of cooperation

Guards

Rendezvous