ÉCOLE
SUPÉRIEURE
D'ÉLECTRICITÉ

A Gentle Introduction to ModHel'X

Frédéric Boulanger Cécile Hardebolle Christophe Jacquet

Supélec E3S – Computer Science Department

July 2013

2013-06-30 17:36

Agenda

Who are we?

Supélec

- French engineering school ("grande école")
- 460 engineering degrees delivered each year
- Continuing education

Supélec Systems Science

Multi-disciplinary research team

- Power systems, Control Science, Electro-magnetism, Telecommunications, Signal processing
- ... and Computer science

Computer Science Department

- Personalization of hypermedia and web queries
- Optimization of high-performance networks
- Modeling and validation of heterogeneous systems

Our View of Heterogeneity

System Design

Our View of Heterogeneity

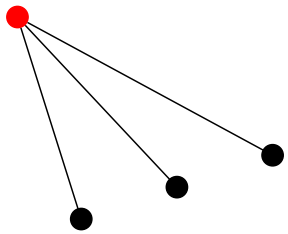
System Design

Starting from an overall specification

Our View of Heterogeneity

System Design

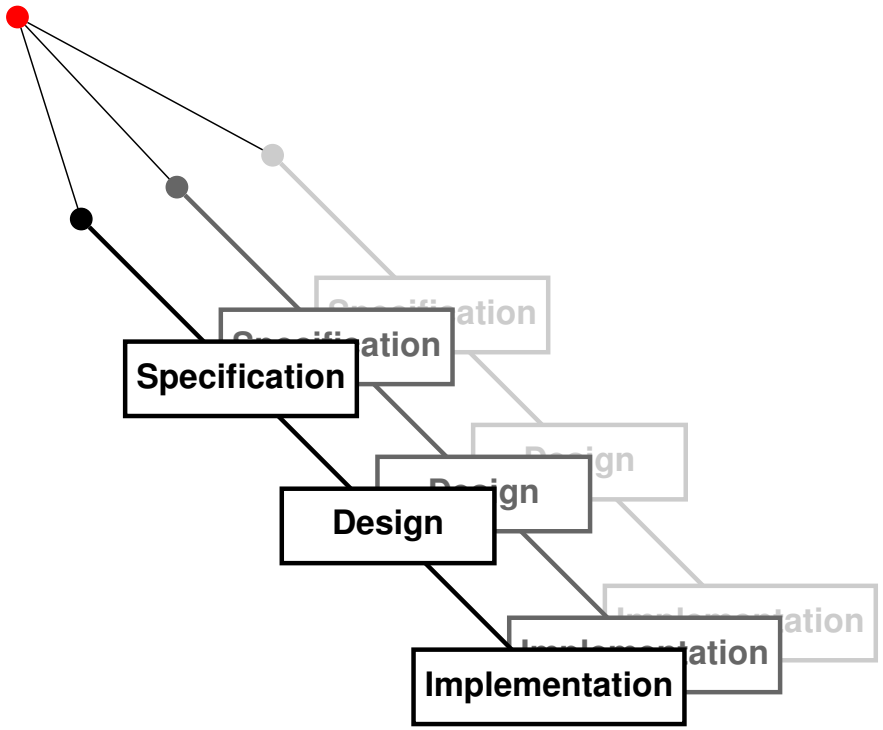
Decompose into simpler subsystems



Our View of Heterogeneity

System Design

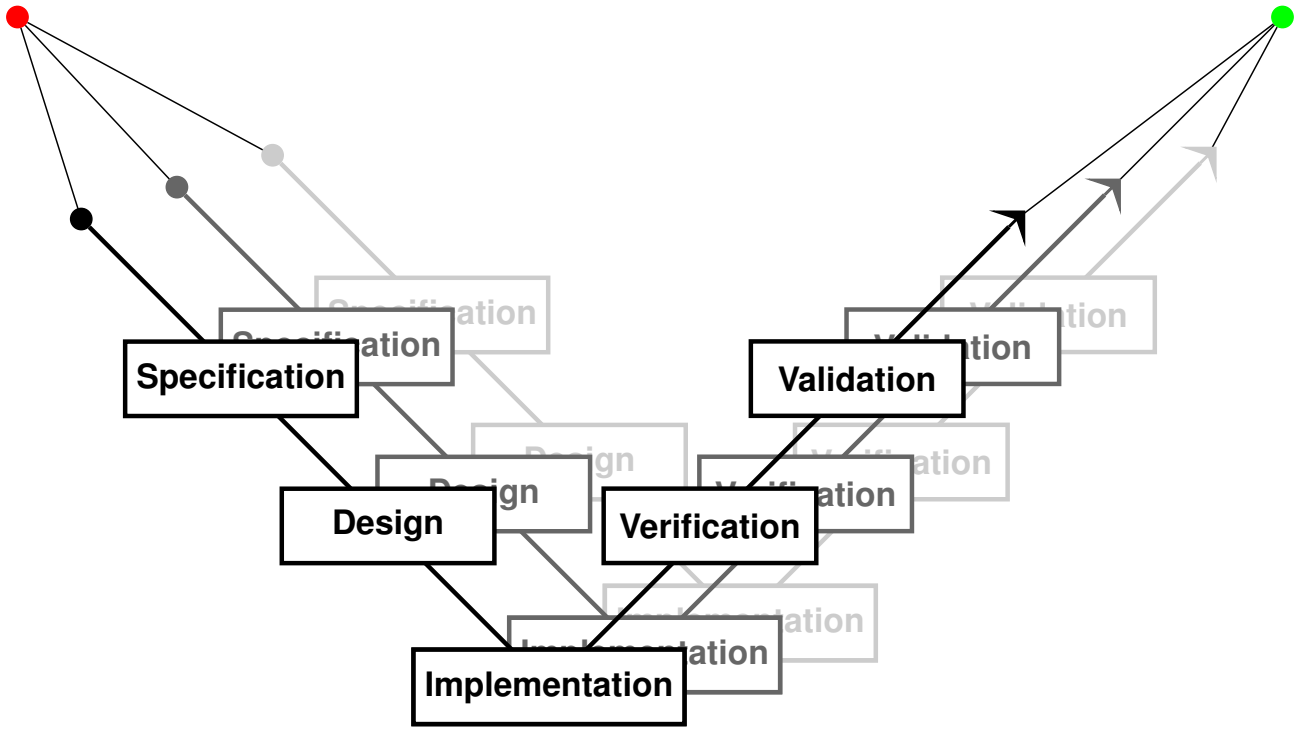
Refine down to implementation



Our View of Heterogeneity

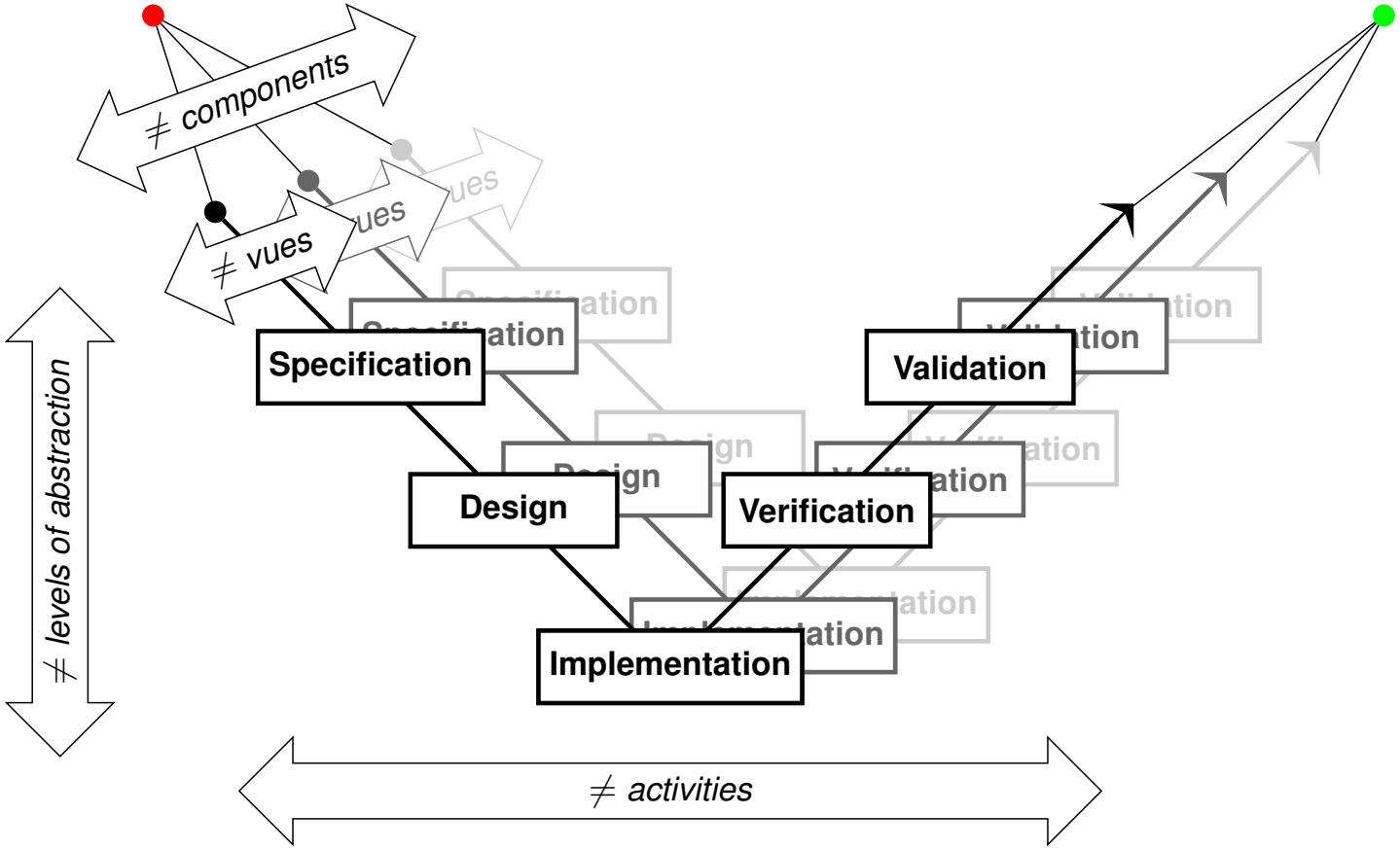
System Design

Recompose components to build the system (integration)



Our View of Heterogeneity

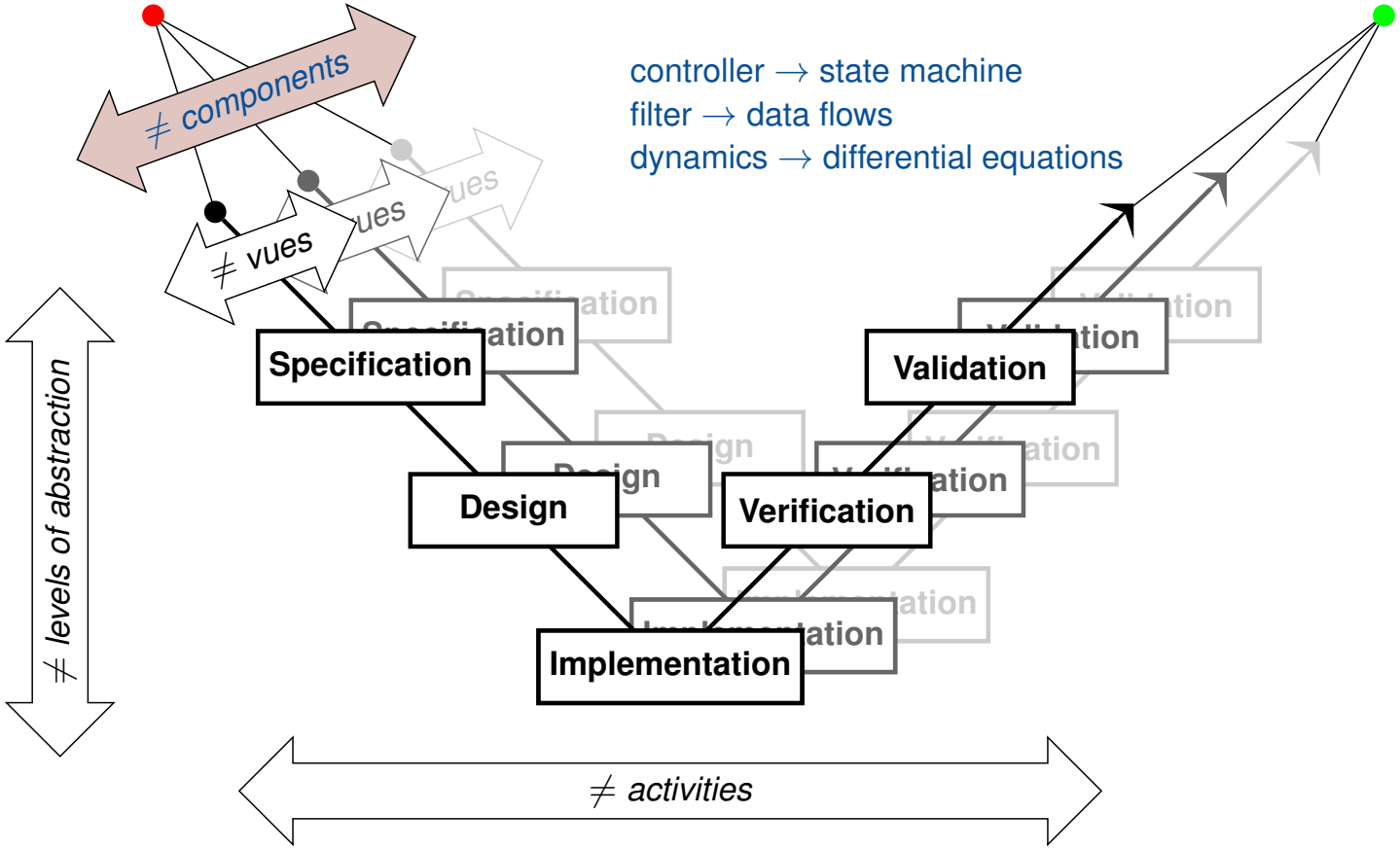
Four sources of heterogeneity



Our View of Heterogeneity

1 - Different components are modeled according to different paradigms

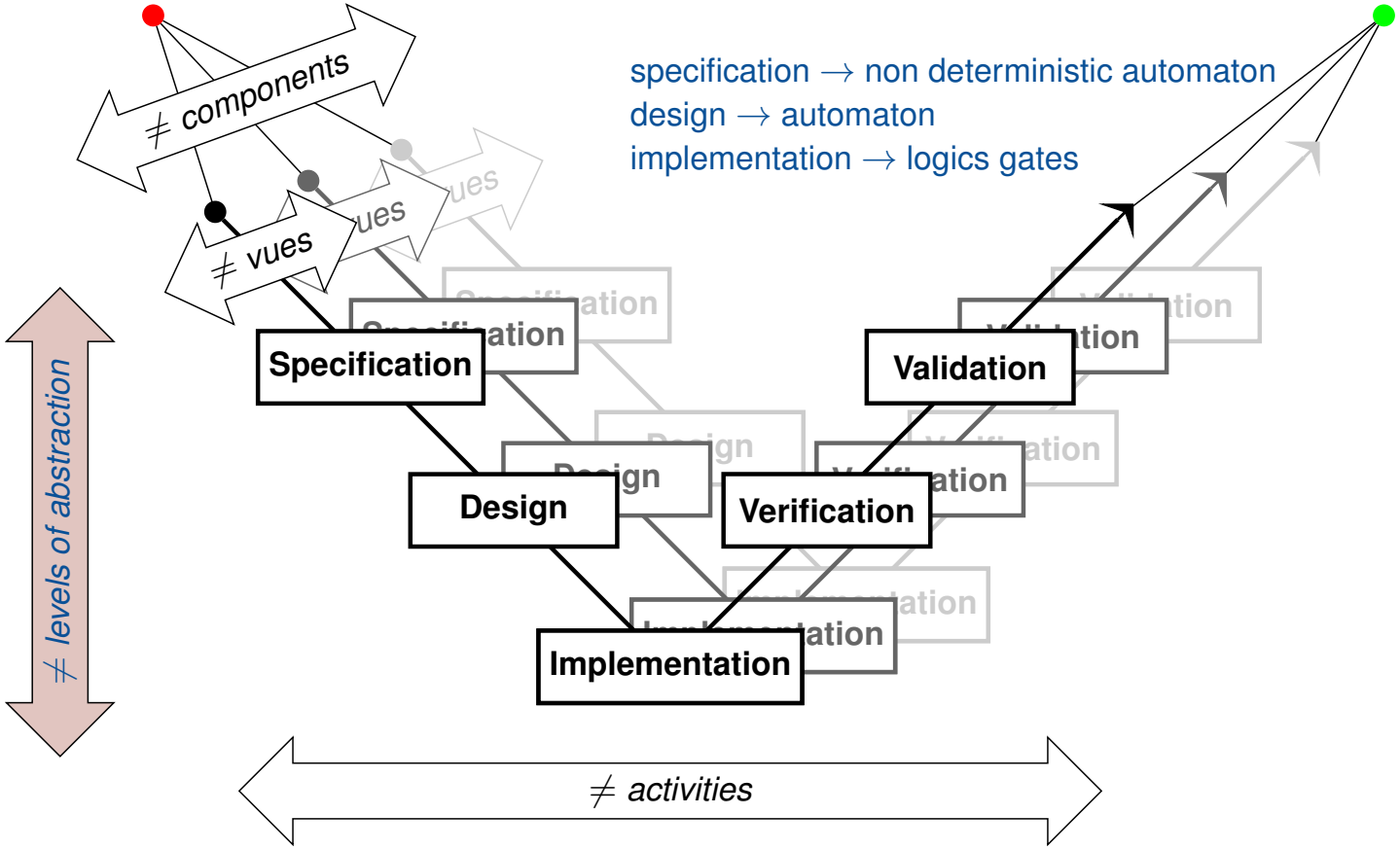
$\Rightarrow$ Issue: combine heterogeneous behaviors



Our View of Heterogeneity

2 - Different formalisms are used at different levels of abstraction

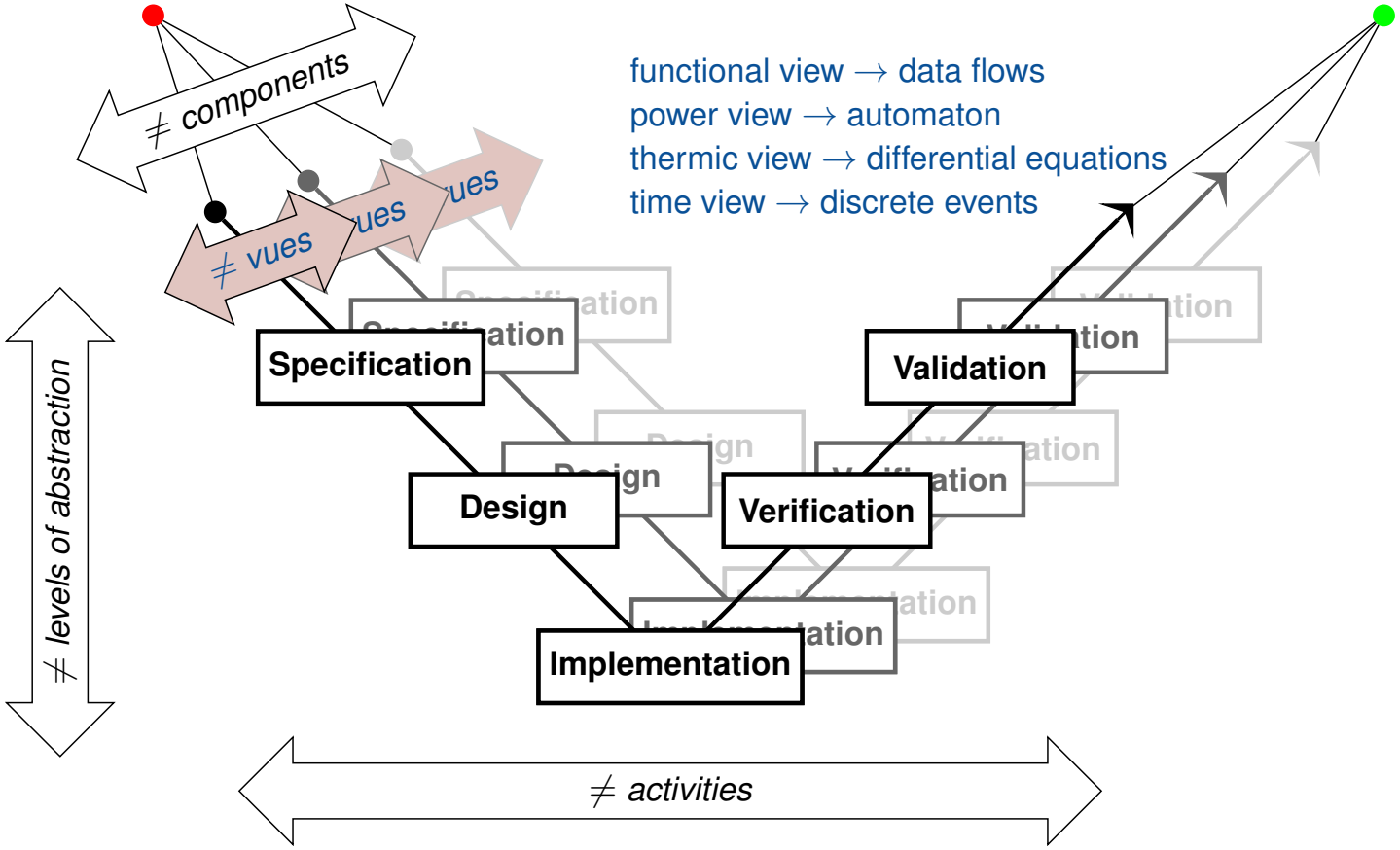
$\Rightarrow$ Issue: conformance of refinements



Our View of Heterogeneity

3 - Different views of a component use different paradigms

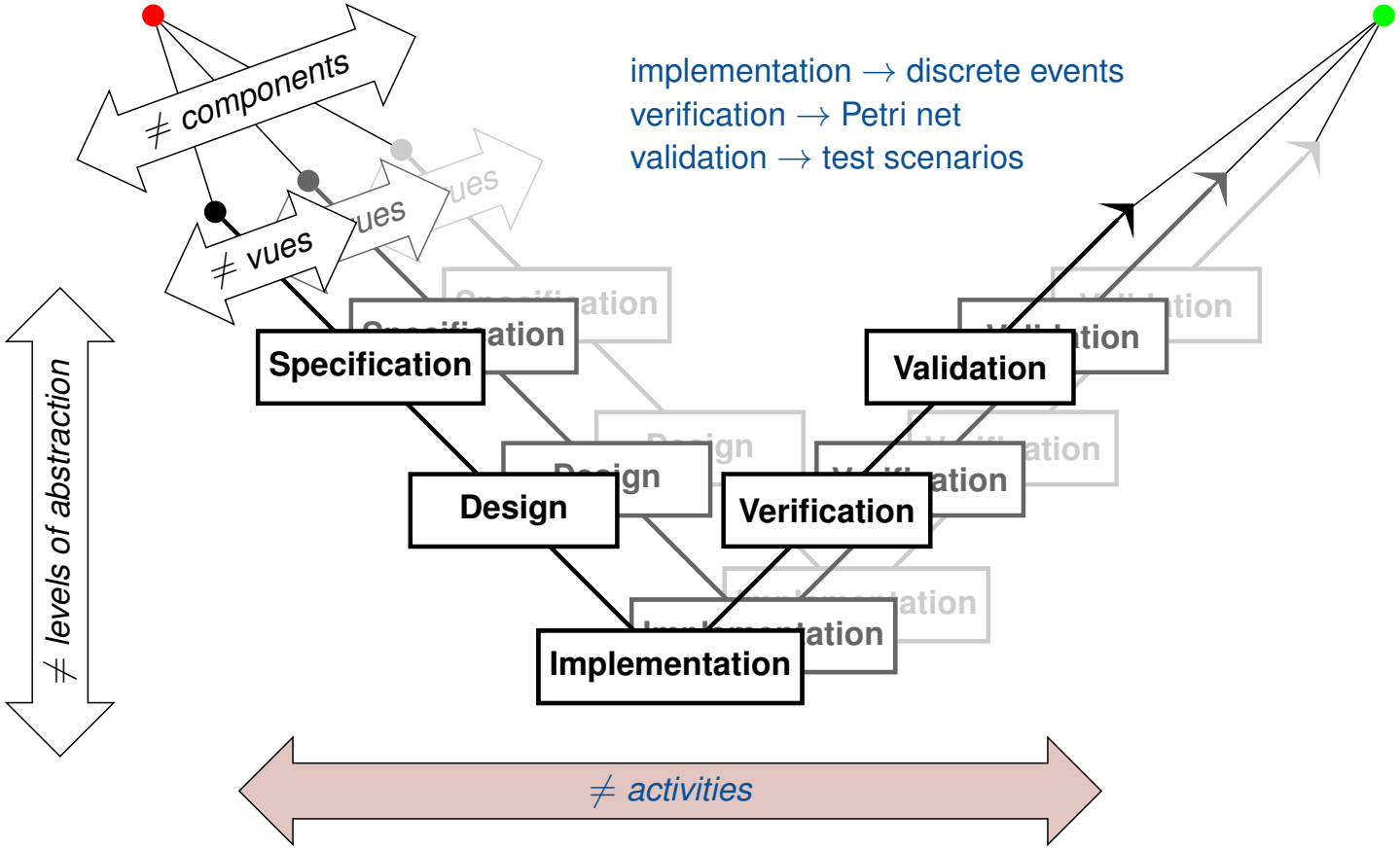
⇒ Issue: synchronize heterogeneous models



Our View of Heterogeneity

4 - Different activities require different formalisms

$\Rightarrow$ Issue: consistence of different models



Problems Raised by Heterogeneity

Four problems to solve

1. Composition of components ⇒ combine heterogeneous behaviors
   ModHel'X Interface blocks, Semantic adaptation
2. Different levels of abstraction ⇒ conformance of refinements
   Conformance testing, B method
3. Superposition of views ⇒ synchronize heterogeneous models
   VUML (OCL constraints)
4. Per activity models ⇒ consistence of different models
   Validation of model transformations

Transverse Issues

- Extra-functional views constrain the refinement of the functional views
- Decomposition is not necessarily the same in all views/activities
  (no one-to-one mapping between components of different views)

Problems Raised by Heterogeneity

Four problems to solve Our current work

1. Composition of components ⇒ combine heterogeneous behaviors
   ModHel'X Interface blocks, Semantic adaptation
2. Different levels of abstraction ⇒ conformance of refinements
   Conformance testing, B method
3. Superposition of views ⇒ synchronize heterogeneous models
   VUML (OCL constraints)
4. Per activity models ⇒ consistence of different models
   Validation of model transformations

Transverse Issues

- Extra-functional views constrain the refinement of the functional views
- Decomposition is not necessarily the same in all views/activities
  (no one-to-one mapping between components of different views)

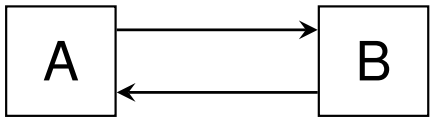
Languages and Models of Computation

Heterogeneous Model

- Joint use of several modeling languages
- Our approach: language $\rightarrow$ model of computation (common syntax)

Model of Computation

- Set of rules for composing the behavior of the components of a model. Semantics for the structure of the model
- Algorithm for solving these rules $\Rightarrow$ model of execution



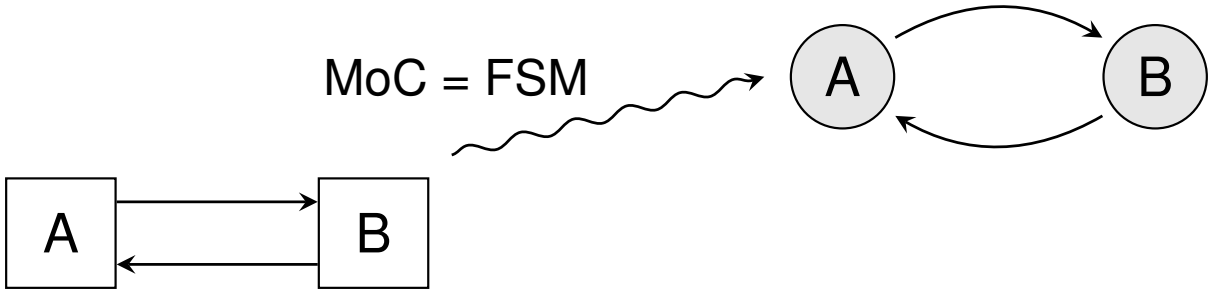
Languages and Models of Computation

Heterogeneous Model

- Joint use of several modeling languages
- Our approach: language $\rightarrow$ model of computation (common syntax)

Model of Computation

- Set of rules for composing the behavior of the components of a model. Semantics for the structure of the model
- Algorithm for solving these rules $\Rightarrow$ model of execution



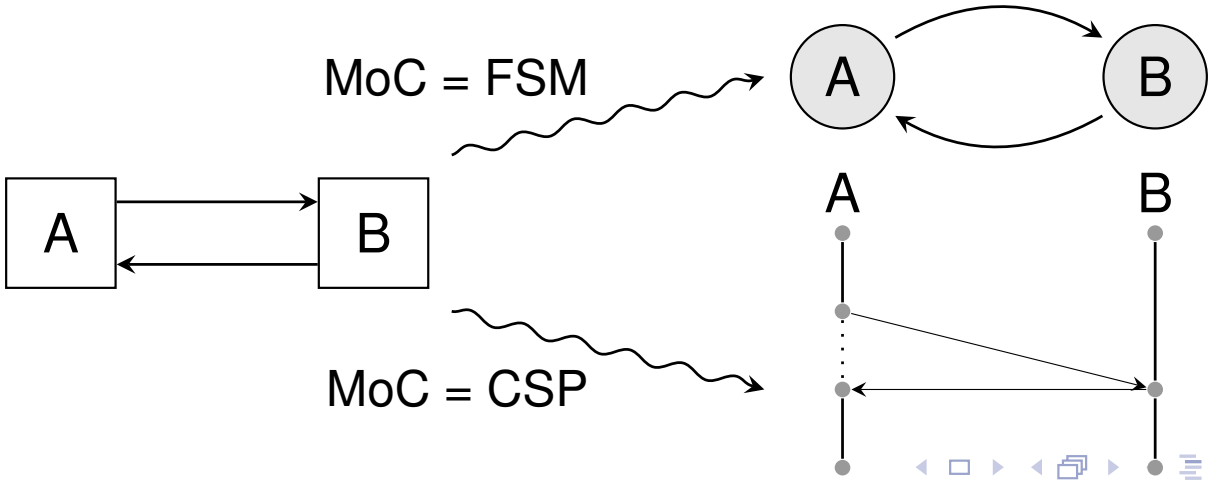
Languages and Models of Computation

Heterogeneous Model

- Joint use of several modeling languages
- Our approach: language $\rightarrow$ model of computation (common syntax)

Model of Computation

- Set of rules for composing the behavior of the components of a model. Semantics for the structure of the model
- Algorithm for solving these rules $\Rightarrow$ model of execution



Execution Machine

Execution Machine

- Environment needed for executing a model correctly
- Communications between the model and its environment

Roles of an Execution Machine

Preservation of the semantics

- Provide the components with an environment which respects the semantics of the modeling language
- Combine the behavior of the components of the model

Semantic Adaptation

- Manage the interface between the model and its environment, which may obey different rules

Models of Computation in ModHel'X

From Ptolemy (Berkeley) to ModHel'X (Supélec)

- Experimental platform for executing heterogeneous models
- Extends previous work, goes beyond some limitations of Ptolemy

ModHel'X

- Metamodel + generic execution engine:

  schedule select next component to observe
  update ask the component to update its interface
  propagate propagate information to the other components

- Behavior of a model = fixed point of *propagate* ∘ *update* ∘ *schedule*
- Does this fixed point exist? Is it unique?
  Can it be reached through iteration?

Generic Metamodel

Behavioral unit: Block, "black box" with an interface

Generic Metamodel

Behavioral unit: Block, "black box" with an interface

Interface unit: Pin, to send and receive information

Generic Metamodel

Behavioral unit: Block, "black box" with an interface

Interface unit: Pin, to send and receive information

Structure: Relations between pins, semantics is defined by the MoC

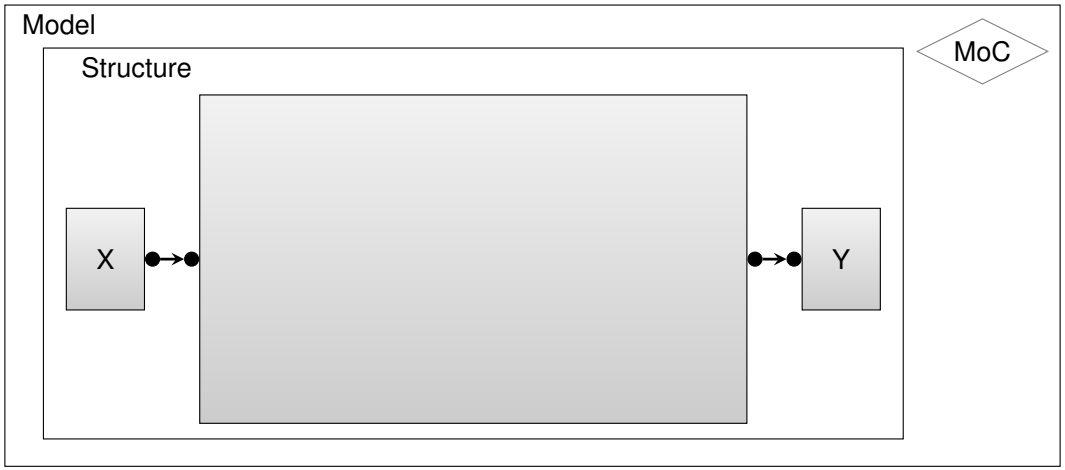
Generic Metamodel

Behavioral unit: Block, "black box" with an interface

Interface unit: Pin, to send and receive information

Structure: Relations between pins, semantics is defined by the MoC

Model: Model = structure + MoC



Generic Metamodel

Behavioral unit: Block, "black box" with an interface

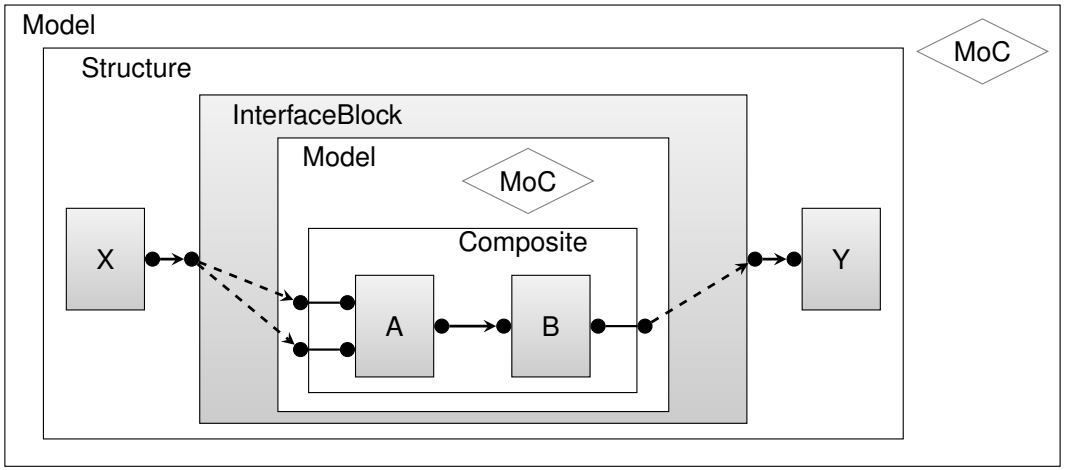
Interface unit: Pin, to send and receive information

Structure: Relations between pins, semantics is defined by the MoC

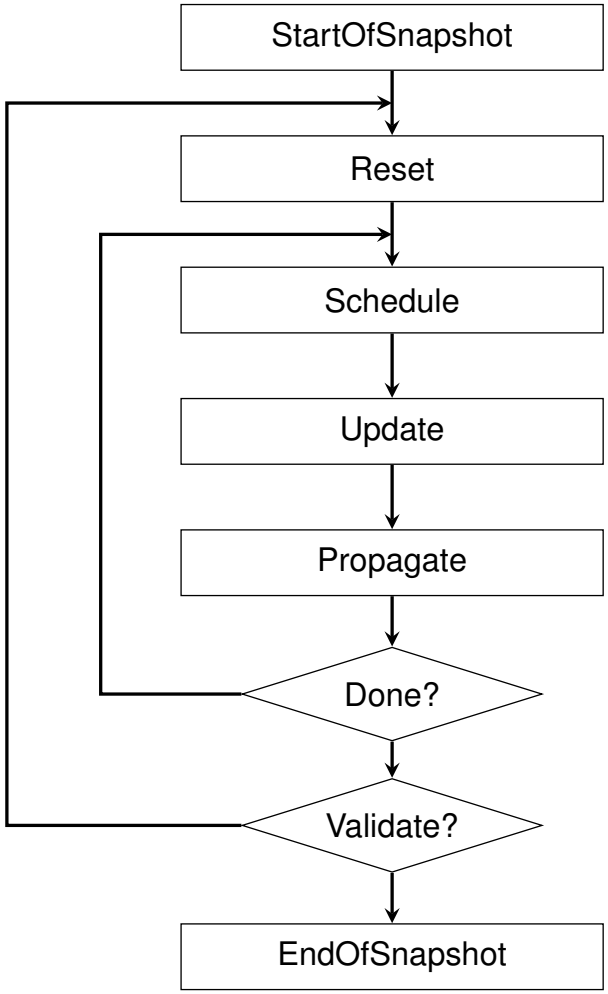
Model: Model = structure + MoC

Hierarchical Heterogeneity

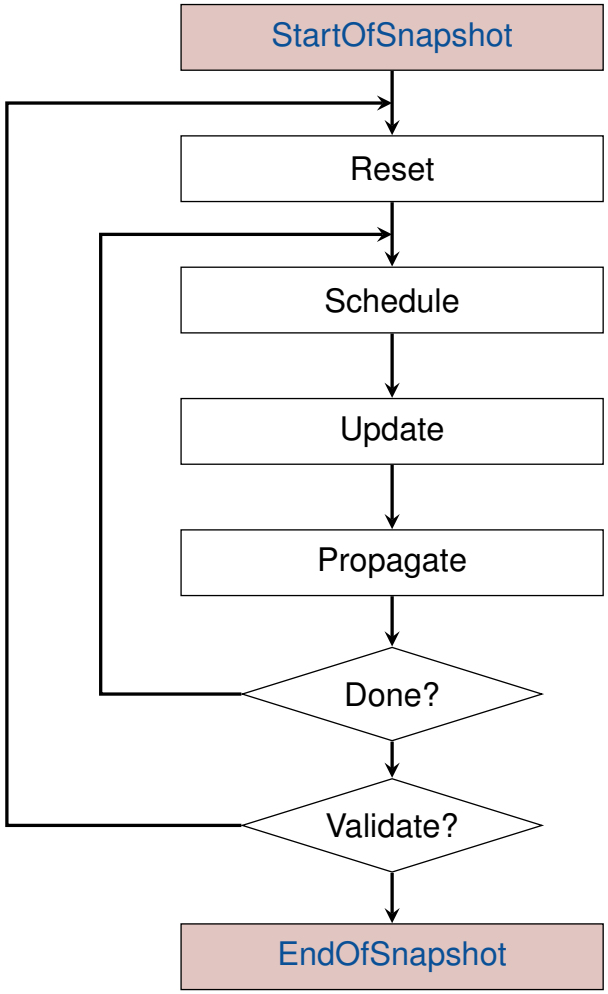
InterfaceBlock : behavior described by a Model



Generic Execution Engine



Generic Execution Engine

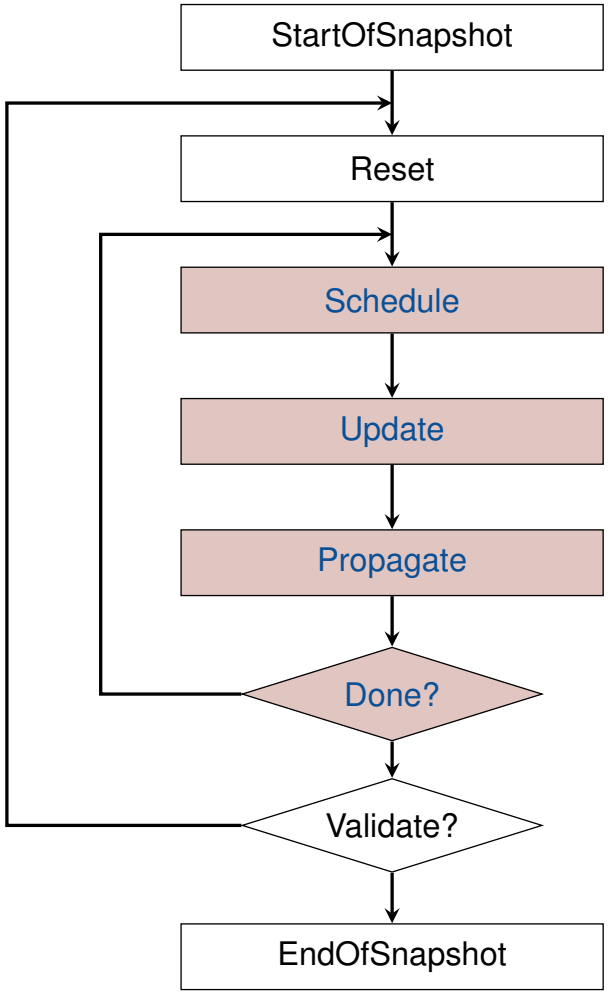


Behavior of a model = series of observations

Synchronous approach of the observation of models:

- no communication with the environment during the snapshot;
- no modification of the internal state of the blocks.

Generic Execution Engine



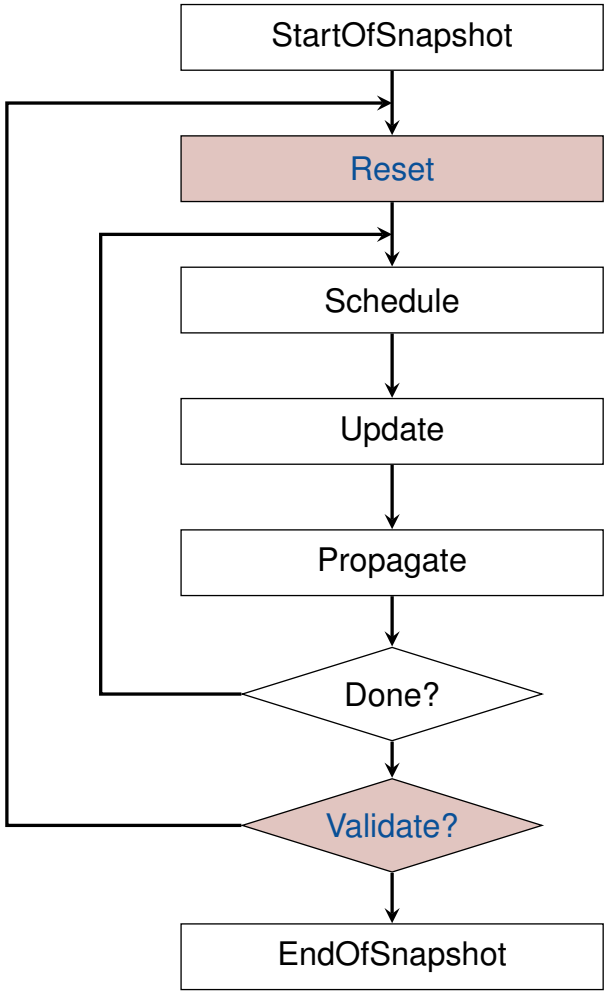
Computation of a fixed point by iteration:

- sequential observation of the blocks;
- update of their interface;
- propagation of the information according to the relations between pins.

Schedule, Propagate and Done are the 3 operations which define a MoC.

Update represents the observable behavior of the blocks.

Generic Execution Engine



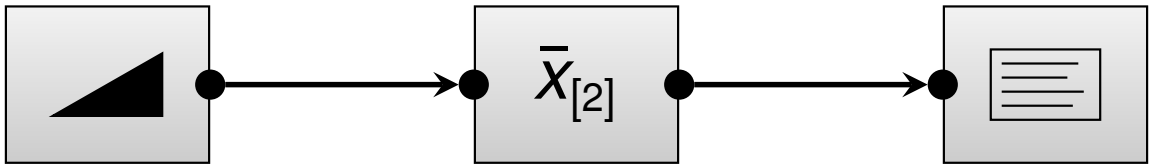
It is possible to reject the fixed point that has been reached.

Search another fixed point starting from different initial conditions.

Example of Computation of a Snapshot

MoC with Synchronous Data Flows

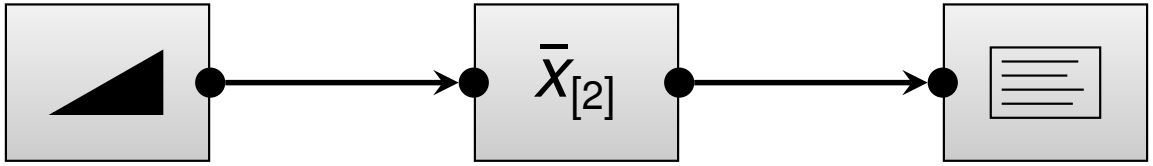
- Instantaneous propagation of data
- Consumption and production of a fixed number of data
- Snapshot = shortest production-consumption cycle



Example of Computation of a Snapshot

MoC with Synchronous Data Flows

- Instantaneous propagation of data
- Consumption and production of a fixed number of data
- Snapshot = shortest production-consumption cycle

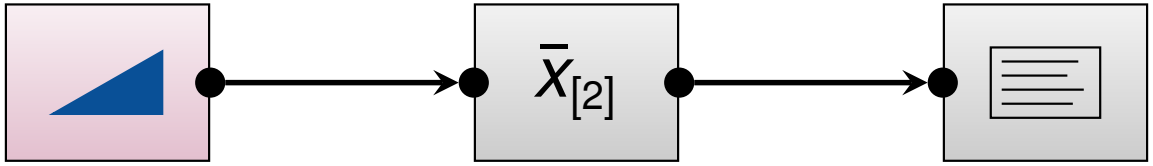


StartOfSnapshot

Example of Computation of a Snapshot

MoC with Synchronous Data Flows

- Instantaneous propagation of data
- Consumption and production of a fixed number of data
- Snapshot = shortest production-consumption cycle

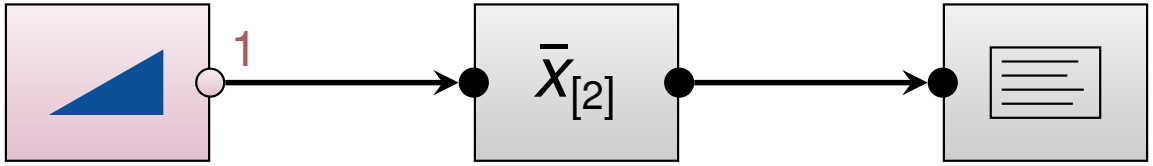


Schedule $\rightarrow$ ramp

Example of Computation of a Snapshot

MoC with Synchronous Data Flows

- Instantaneous propagation of data
- Consumption and production of a fixed number of data
- Snapshot = shortest production-consumption cycle

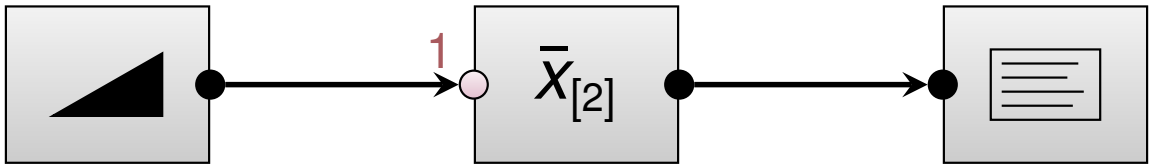


Update ramp

Example of Computation of a Snapshot

MoC with Synchronous Data Flows

- Instantaneous propagation of data
- Consumption and production of a fixed number of data
- Snapshot = shortest production-consumption cycle

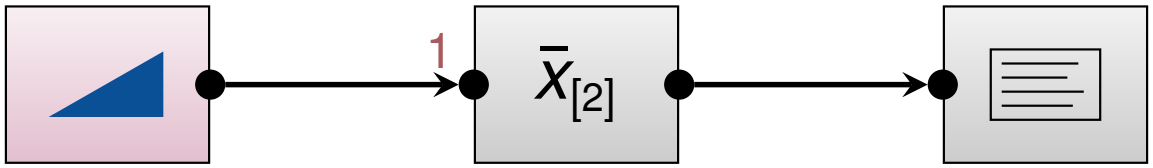


Propagate

Example of Computation of a Snapshot

MoC with Synchronous Data Flows

- Instantaneous propagation of data
- Consumption and production of a fixed number of data
- Snapshot = shortest production-consumption cycle

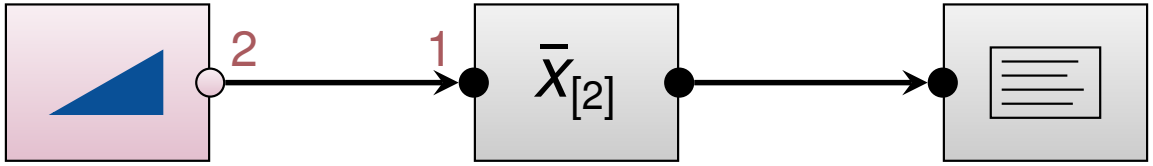


Schedule $\rightarrow$ ramp

Example of Computation of a Snapshot

MoC with Synchronous Data Flows

- Instantaneous propagation of data
- Consumption and production of a fixed number of data
- Snapshot = shortest production-consumption cycle

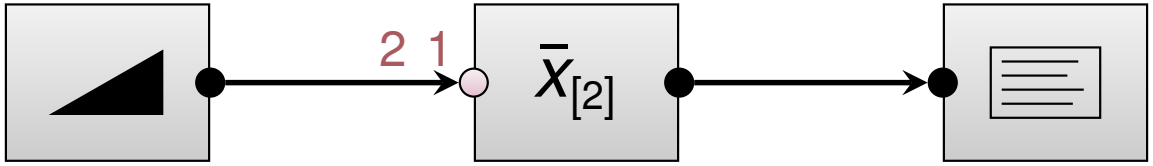


Update ramp

Example of Computation of a Snapshot

MoC with Synchronous Data Flows

- Instantaneous propagation of data
- Consumption and production of a fixed number of data
- Snapshot = shortest production-consumption cycle

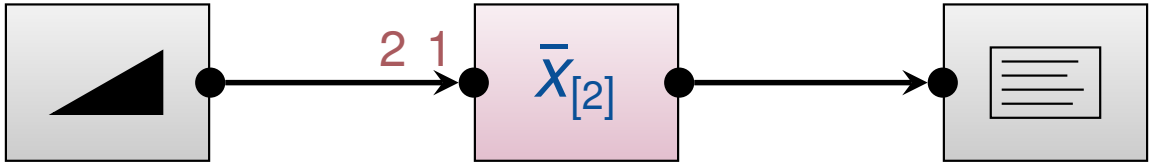


Propagate

Example of Computation of a Snapshot

MoC with Synchronous Data Flows

- Instantaneous propagation of data
- Consumption and production of a fixed number of data
- Snapshot = shortest production-consumption cycle

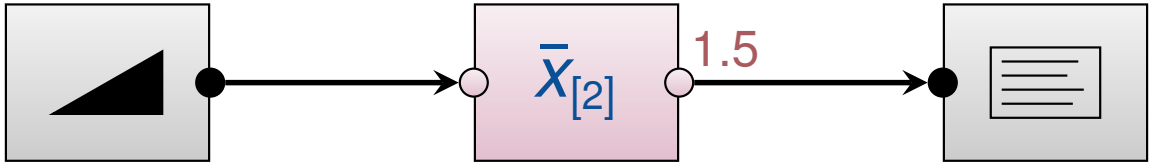


Schedule $\rightarrow$ average

Example of Computation of a Snapshot

MoC with Synchronous Data Flows

- Instantaneous propagation of data
- Consumption and production of a fixed number of data
- Snapshot = shortest production-consumption cycle

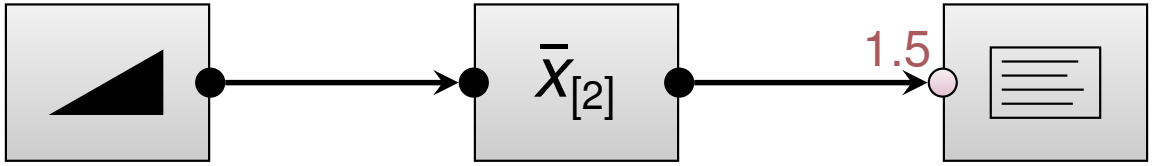


Update average

Example of Computation of a Snapshot

MoC with Synchronous Data Flows

- Instantaneous propagation of data
- Consumption and production of a fixed number of data
- Snapshot = shortest production-consumption cycle

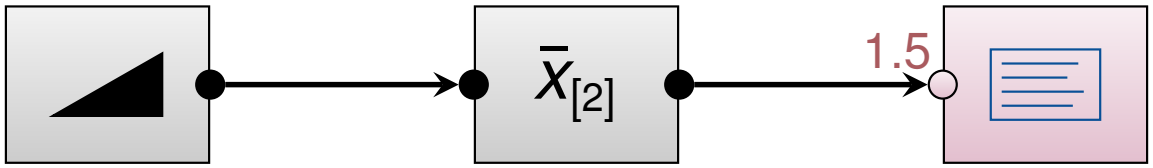


Propagate

Example of Computation of a Snapshot

MoC with Synchronous Data Flows

- Instantaneous propagation of data
- Consumption and production of a fixed number of data
- Snapshot = shortest production-consumption cycle

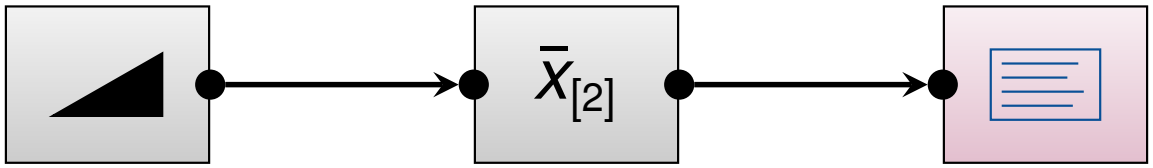


Schedule → display

Example of Computation of a Snapshot

MoC with Synchronous Data Flows

- Instantaneous propagation of data
- Consumption and production of a fixed number of data
- Snapshot = shortest production-consumption cycle

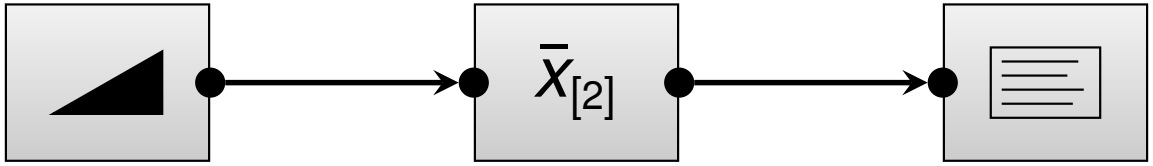


Update display

Example of Computation of a Snapshot

MoC with Synchronous Data Flows

- Instantaneous propagation of data
- Consumption and production of a fixed number of data
- Snapshot = shortest production-consumption cycle

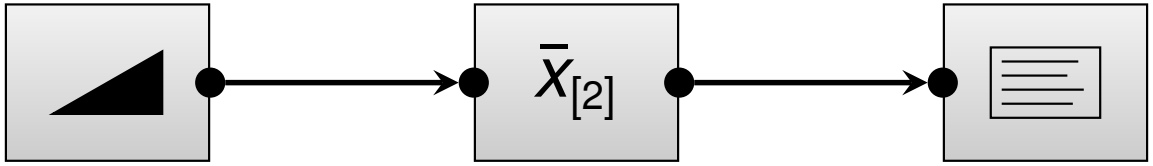


Done

Example of Computation of a Snapshot

MoC with Synchronous Data Flows

- Instantaneous propagation of data
- Consumption and production of a fixed number of data
- Snapshot = shortest production-consumption cycle

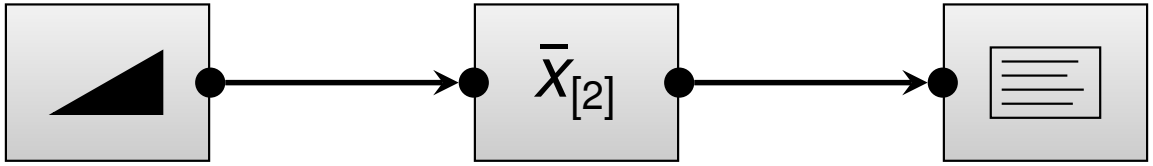


Validate

Example of Computation of a Snapshot

MoC with Synchronous Data Flows

- Instantaneous propagation of data
- Consumption and production of a fixed number of data
- Snapshot = shortest production-consumption cycle

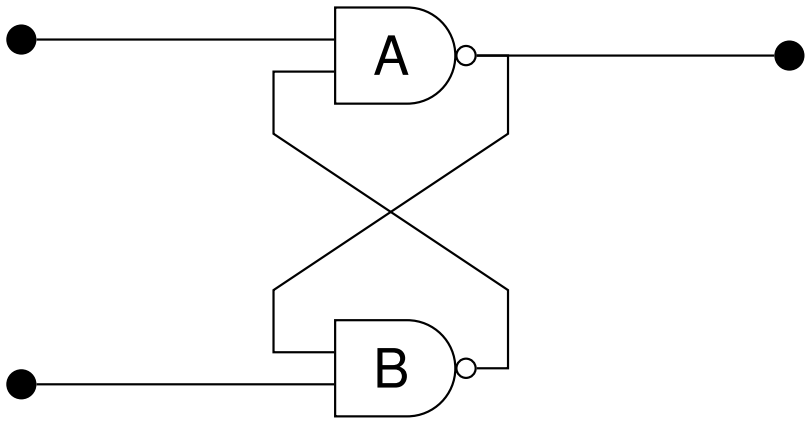


EndOfSnapshot

Example of Computation of a Snapshot

Synchronous reactive MoC

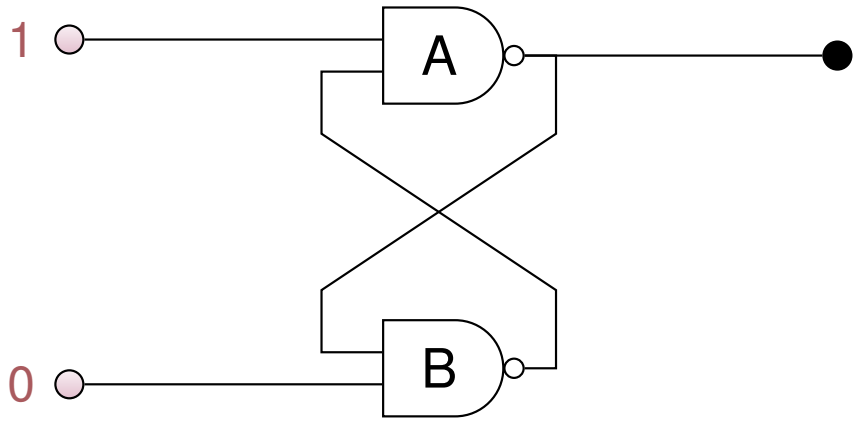
- Instantaneous propagation of data
- Non-strict blocks (may react to partial input)
- Snapshot = all signals are known



Example of Computation of a Snapshot

Synchronous reactive MoC

- Instantaneous propagation of data
- Non-strict blocks (may react to partial input)
- Snapshot = all signals are known

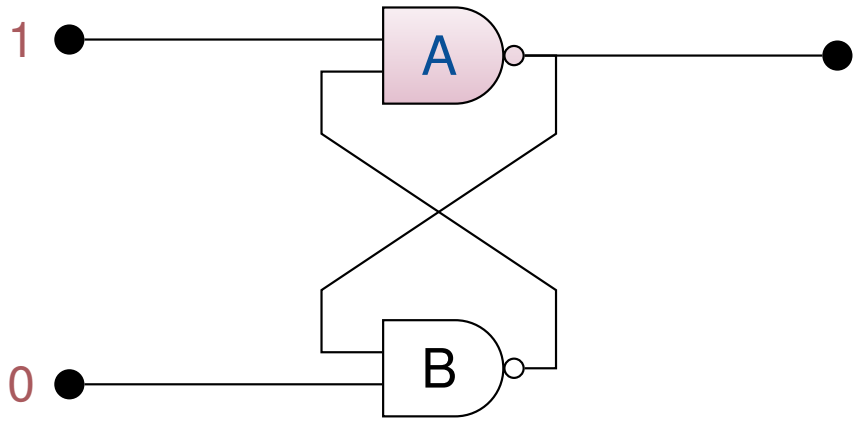


StartOfSnapshot

Example of Computation of a Snapshot

Synchronous reactive MoC

- Instantaneous propagation of data
- Non-strict blocks (may react to partial input)
- Snapshot = all signals are known

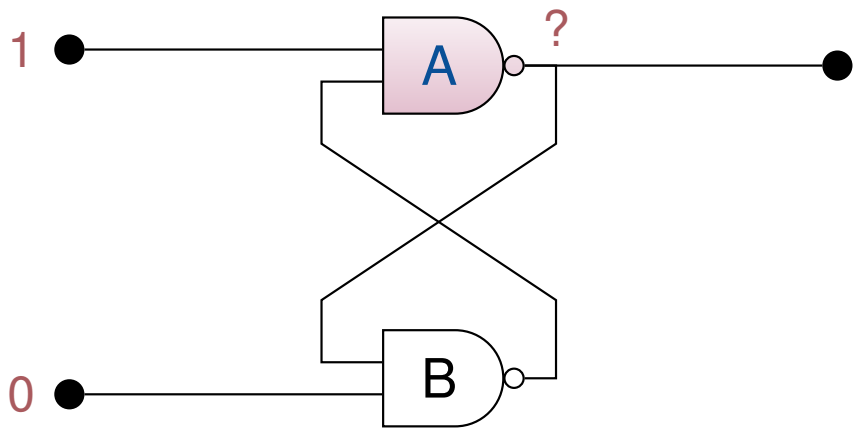


Schedule → A

Example of Computation of a Snapshot

Synchronous reactive MoC

- Instantaneous propagation of data
- Non-strict blocks (may react to partial input)
- Snapshot = all signals are known

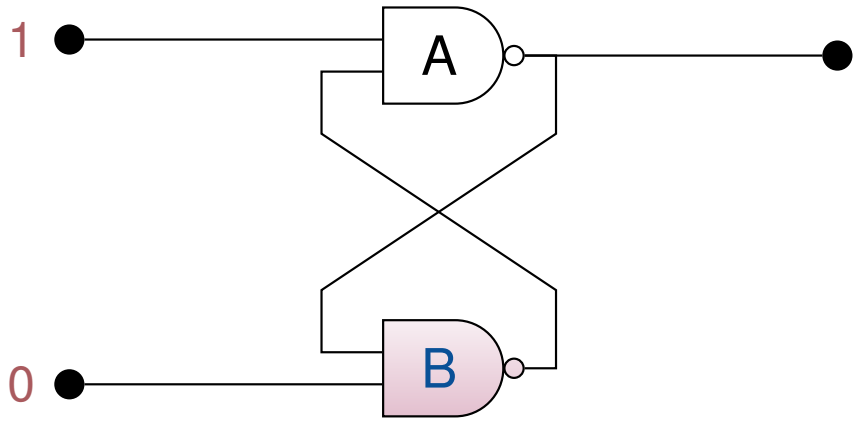


Update A

Example of Computation of a Snapshot

Synchronous reactive MoC

- Instantaneous propagation of data
- Non-strict blocks (may react to partial input)
- Snapshot = all signals are known

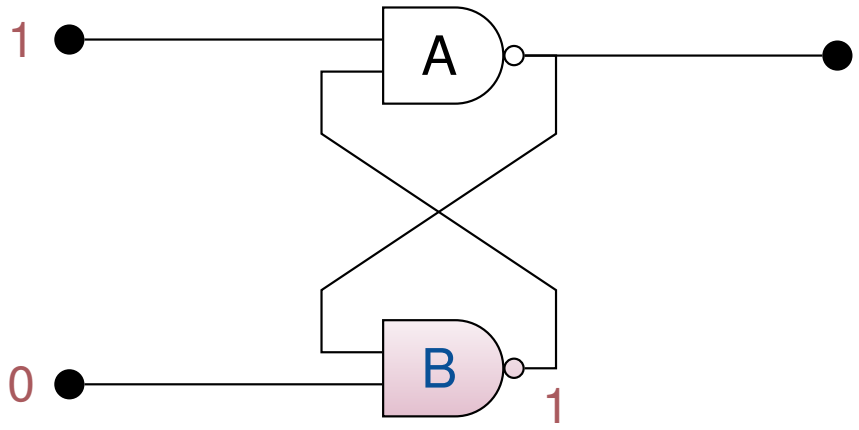


Schedule → B

Example of Computation of a Snapshot

Synchronous reactive MoC

- Instantaneous propagation of data
- Non-strict blocks (may react to partial input)
- Snapshot = all signals are known

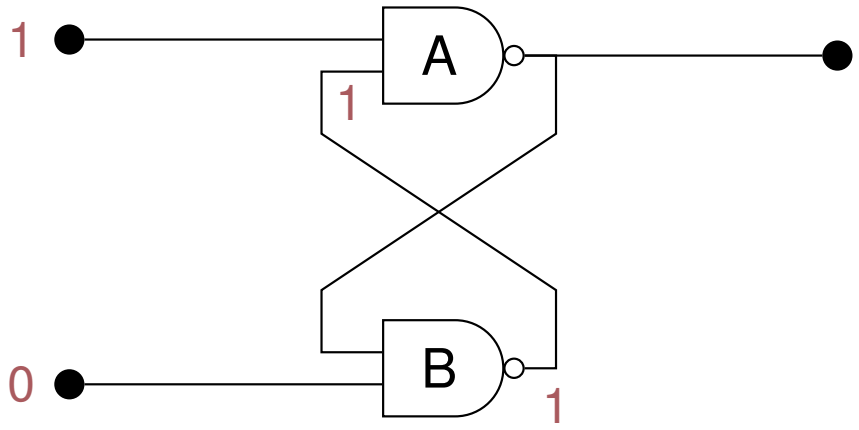


Update B

Example of Computation of a Snapshot

Synchronous reactive MoC

- Instantaneous propagation of data
- Non-strict blocks (may react to partial input)
- Snapshot = all signals are known

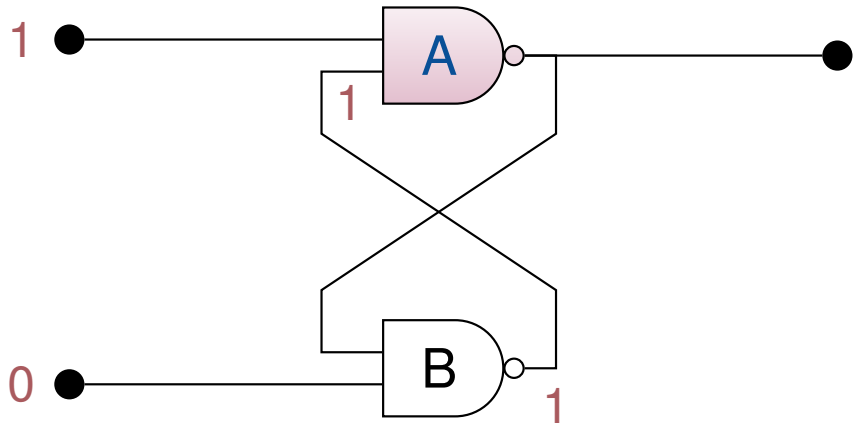


Propagate

Example of Computation of a Snapshot

Synchronous reactive MoC

- Instantaneous propagation of data
- Non-strict blocks (may react to partial input)
- Snapshot = all signals are known

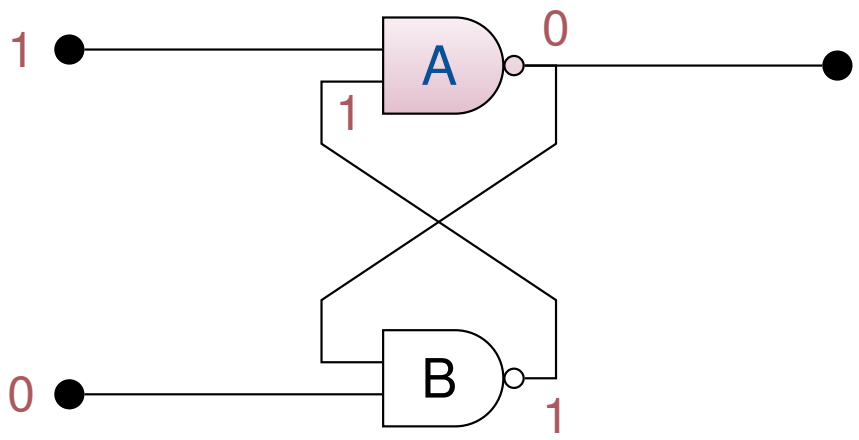


Schedule → A

Example of Computation of a Snapshot

Synchronous reactive MoC

- Instantaneous propagation of data
- Non-strict blocks (may react to partial input)
- Snapshot = all signals are known

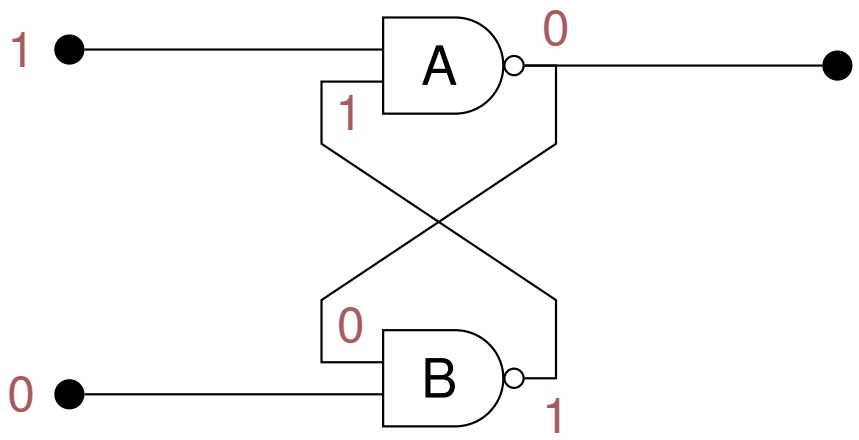


Update A

Example of Computation of a Snapshot

Synchronous reactive MoC

- Instantaneous propagation of data
- Non-strict blocks (may react to partial input)
- Snapshot = all signals are known

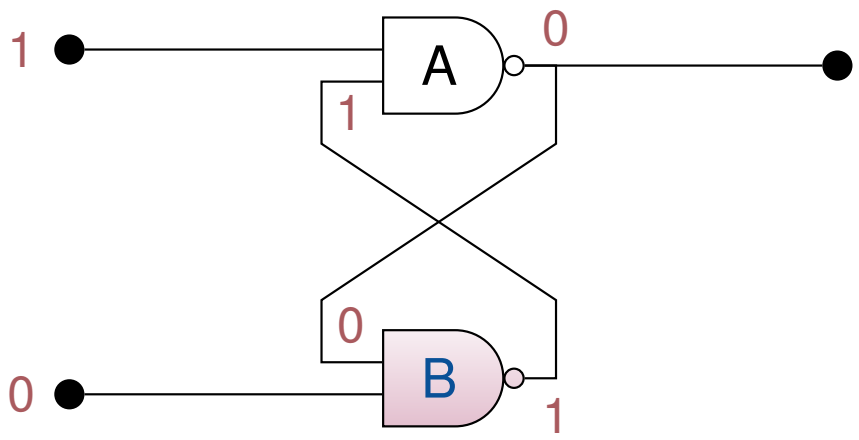


Propagate

Example of Computation of a Snapshot

Synchronous reactive MoC

- Instantaneous propagation of data
- Non-strict blocks (may react to partial input)
- Snapshot = all signals are known

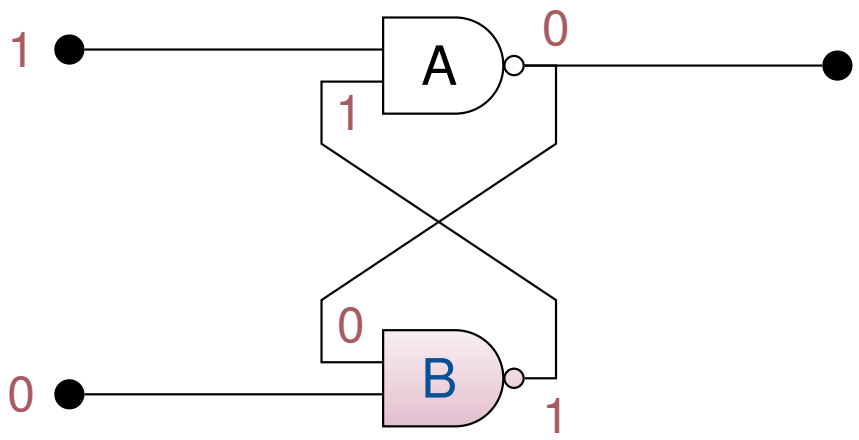


Schedule → B

Example of Computation of a Snapshot

Synchronous reactive MoC

- Instantaneous propagation of data
- Non-strict blocks (may react to partial input)
- Snapshot = all signals are known

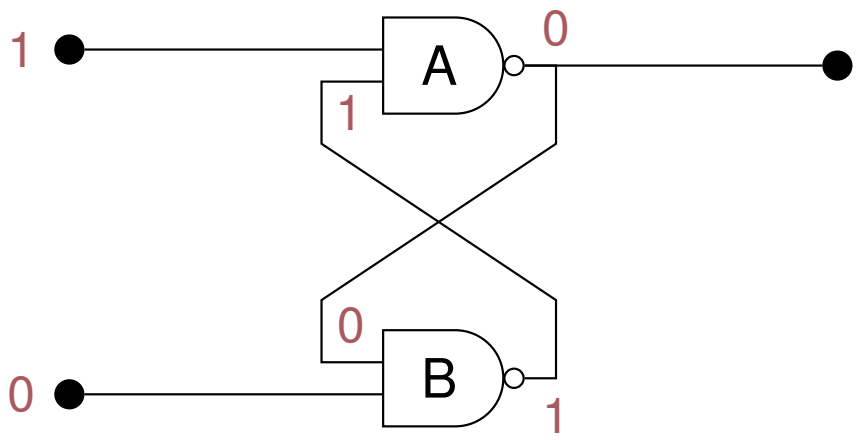


Update B

Example of Computation of a Snapshot

Synchronous reactive MoC

- Instantaneous propagation of data
- Non-strict blocks (may react to partial input)
- Snapshot = all signals are known

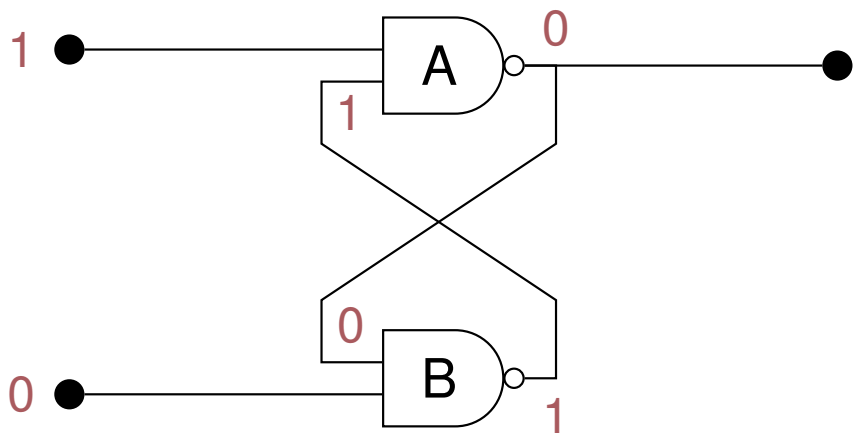


Done

Example of Computation of a Snapshot

Synchronous reactive MoC

- Instantaneous propagation of data
- Non-strict blocks (may react to partial input)
- Snapshot = all signals are known

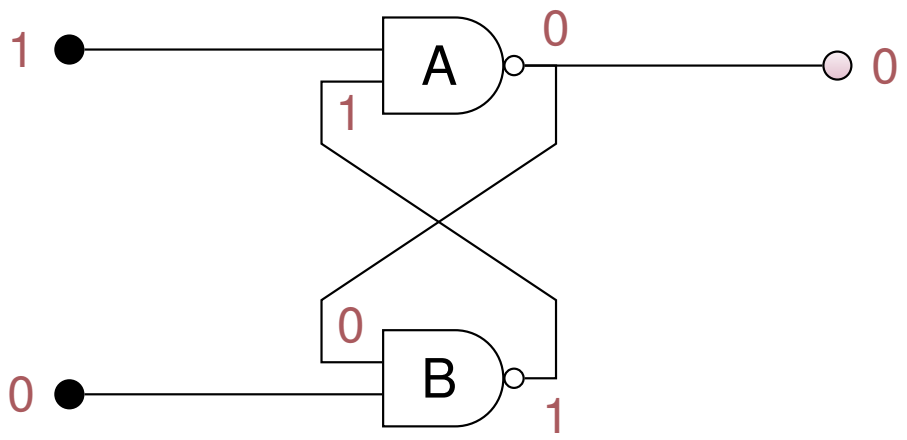


Validate

Example of Computation of a Snapshot

Synchronous reactive MoC

- Instantaneous propagation of data
- Non-strict blocks (may react to partial input)
- Snapshot = all signals are known



EndOfSnapshot

Non-validation

Plan

Managing Heterogeneity

Heterogeneous Modeling ⇒ description of different MoCs

Interactions between models that obey different models of computation?

How to combine

- State machines
- Block diagrams
- Process networks
- Discrete behaviors
- Continuous behaviors

Managing Heterogeneity

Heterogeneous Modeling ⇒ description of different MoCs

Interactions between models that obey different models of computation?

How to combine

- State machines
- Block diagrams
- Process networks
- Discrete behaviors
- Continuous behaviors

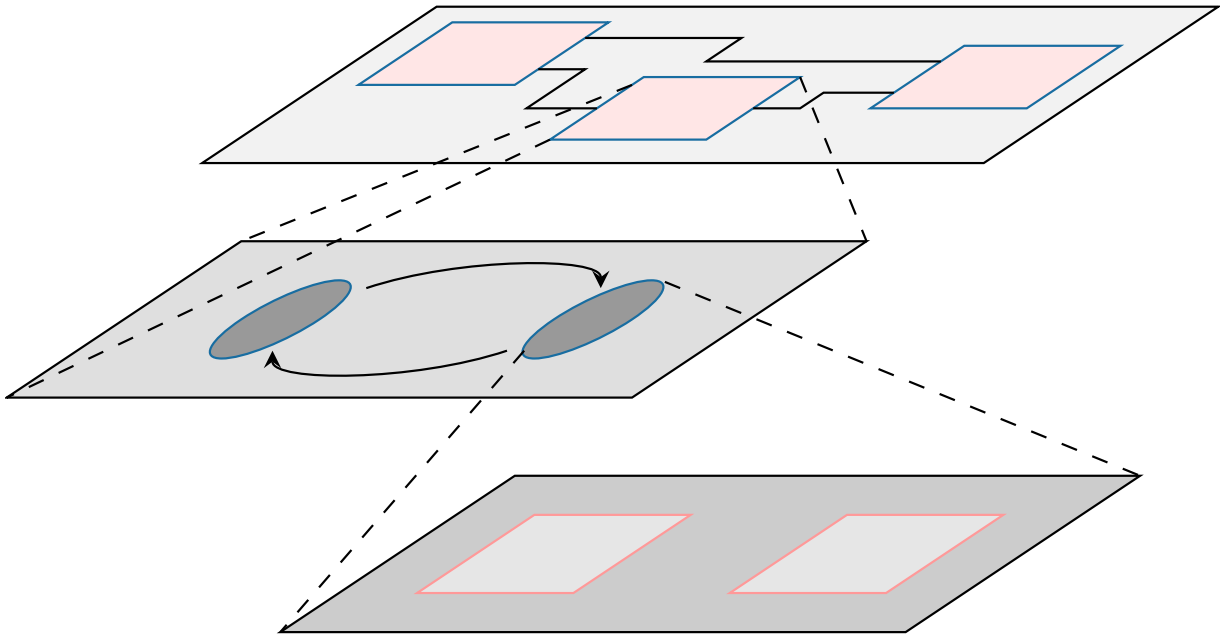
Two kinds of chimeras

- Flat composition of models of computation
- Hierarchical composition of models of computation

Heterogeneity

Hierarchical Composition

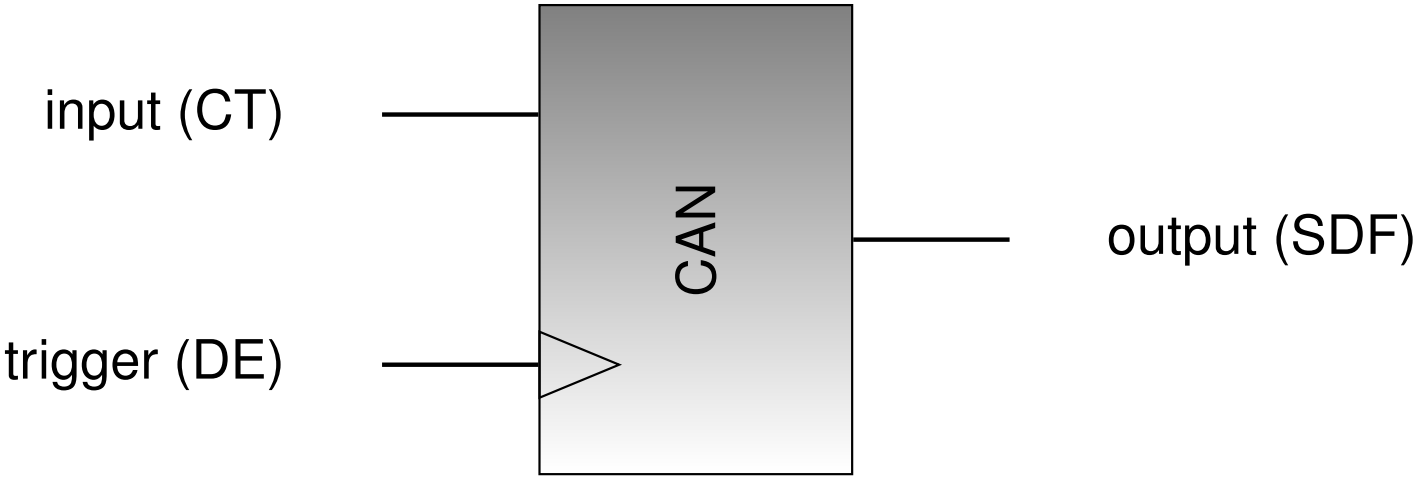
- Approach used by Ptolemy and ModHel'X
- Each hierarchical level uses only one MoC
- Allows us to consider adaptation between pairs of MoC only



Heterogeneity

Flat Composition

- Several paradigms can be used at a given hierarchical level
- Approaches with connectors (limited number of combinations)
- Heterogeneous interface components



It is sometimes possible to convert a flat model into a hierarchical model

Plan

Semantic Adaptation

ModHel'X: Semantic Adaptation along Three Axes

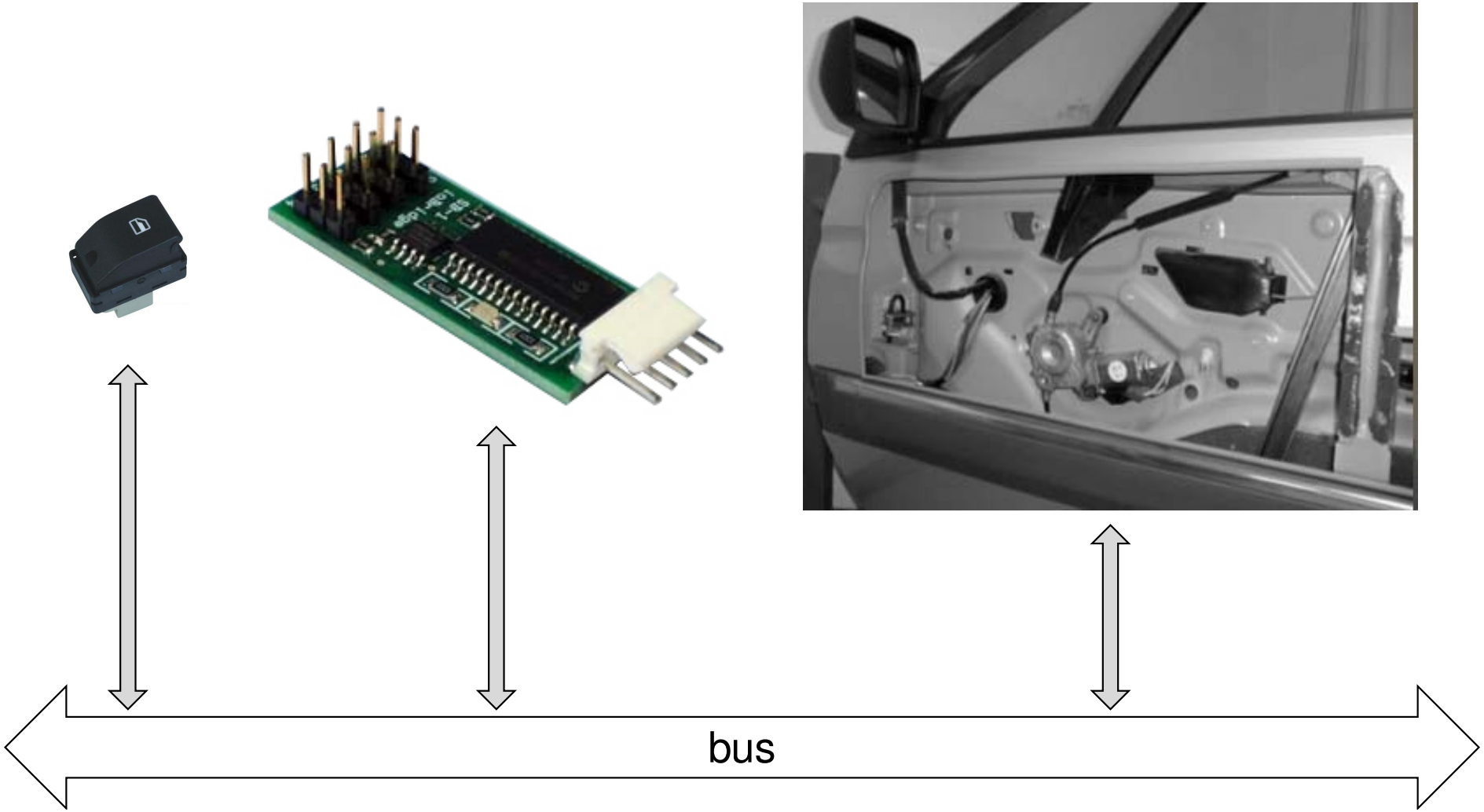
- Adaptation of data:
  different MoCs may use different kinds of data
- Adaptation of time :
  different notions of time are used in different MoCs
- Adaptation of control :
  the instants at which a model should be observed depend on the MoC

Current implementation

- MoCs: DE, SDF, TFSM
- Parameterized semantic adaptation patterns
- Example: the power window

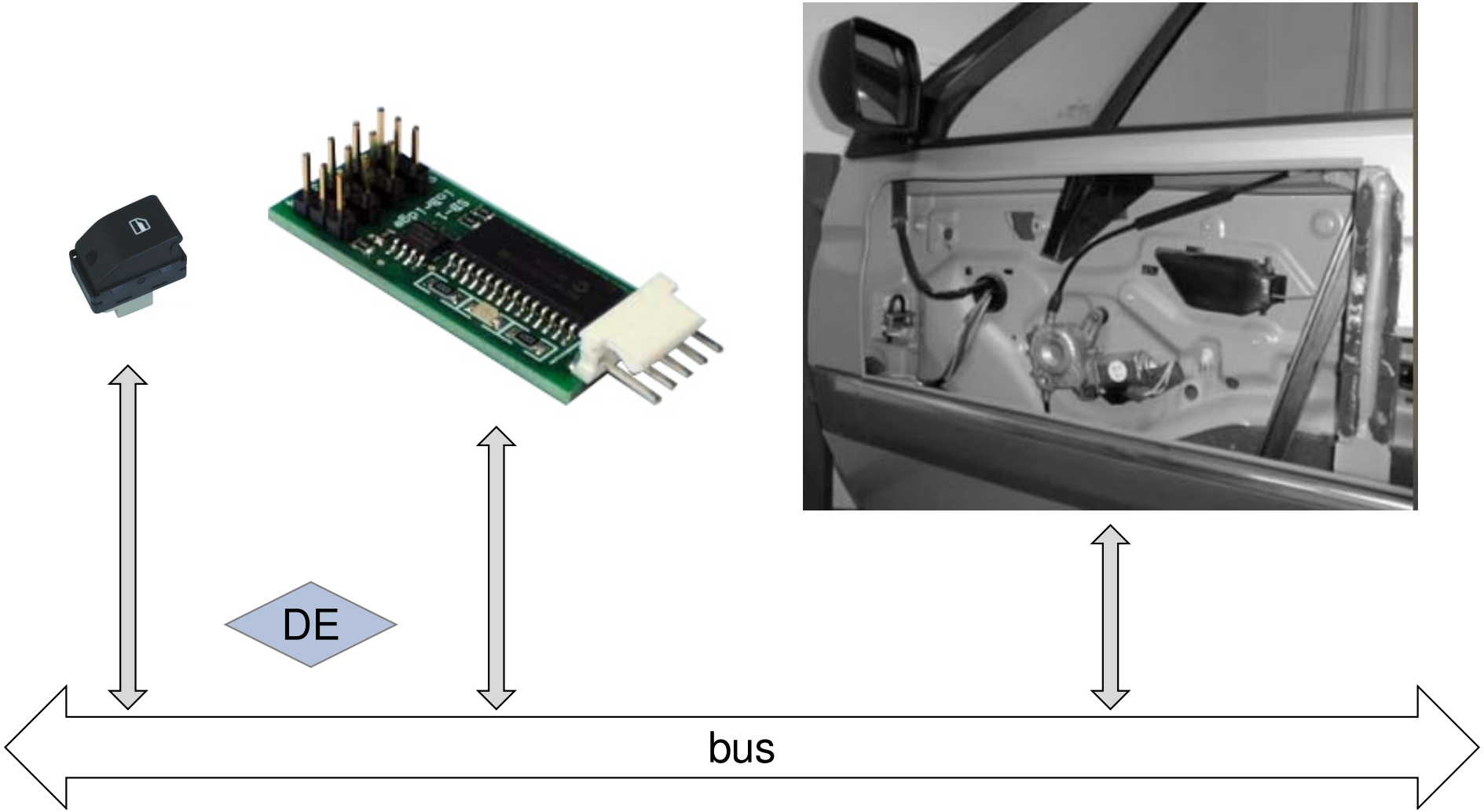
Example: power window

demo



Example: power window

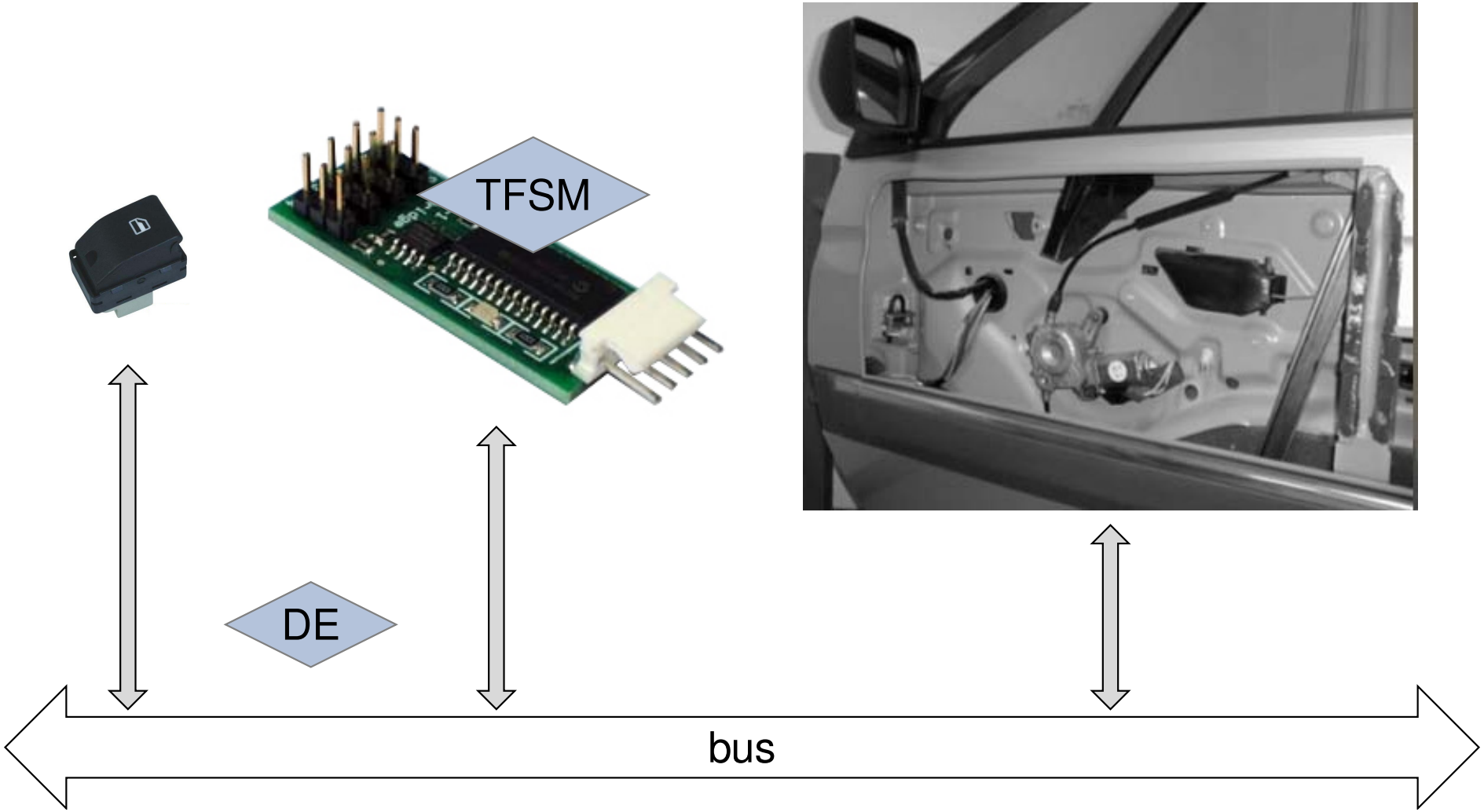
demo



Semantic Adaptation

Example: power window

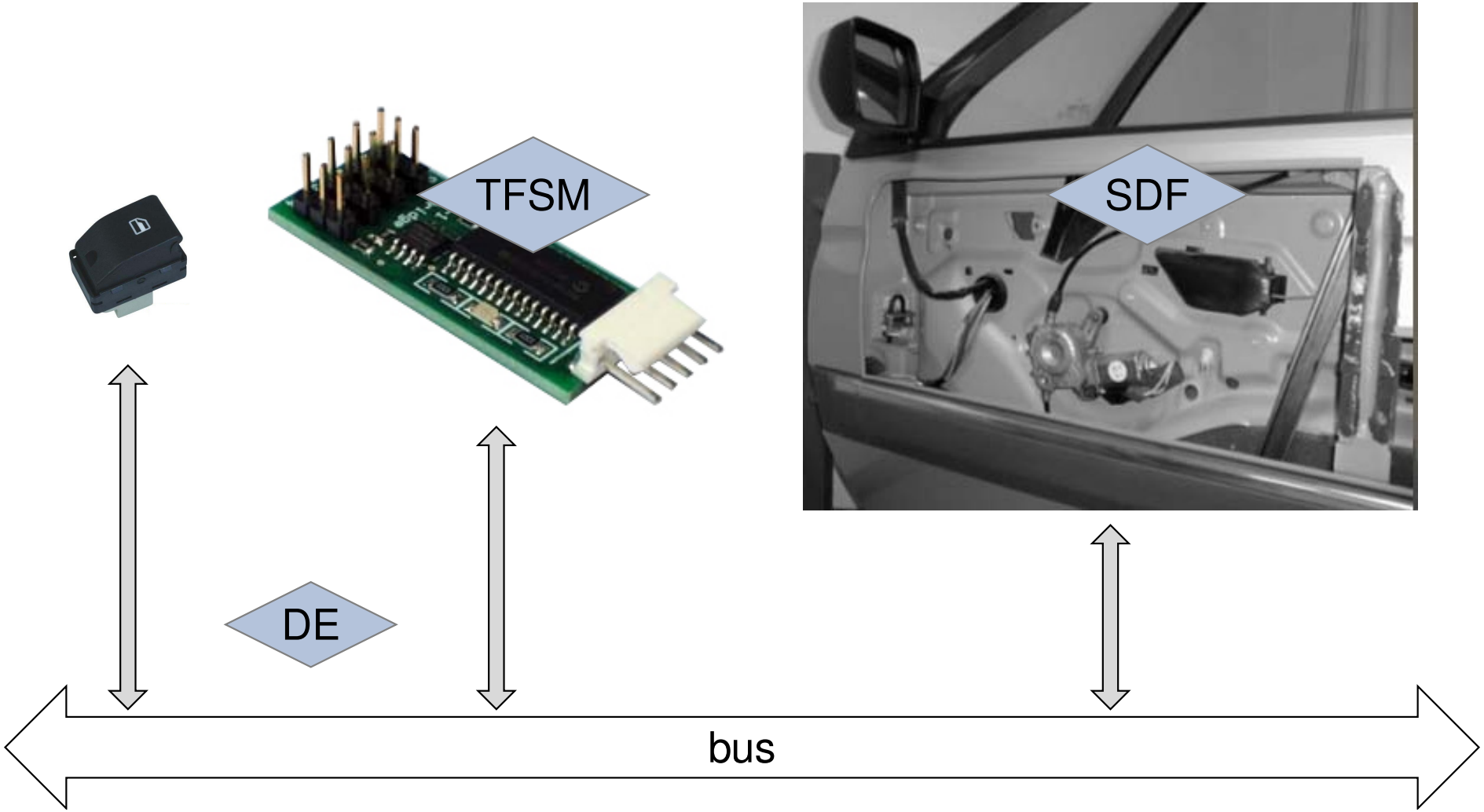
demo



Semantic Adaptation

Example: power window

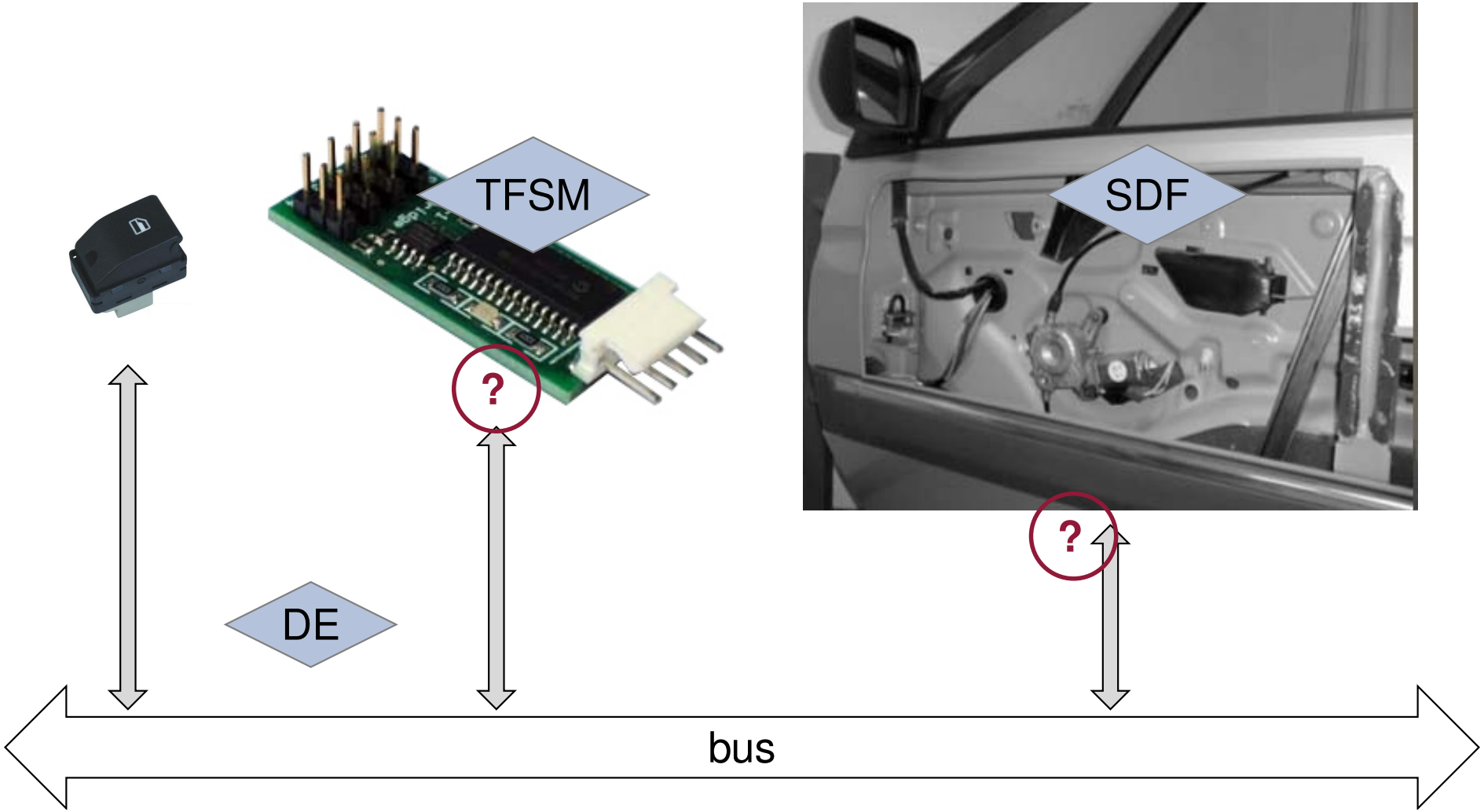
demo



Semantic Adaptation

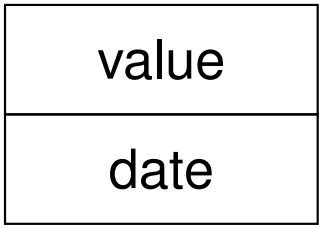
Example: power window

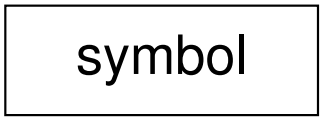
demo



Semantic Adaptation (data)

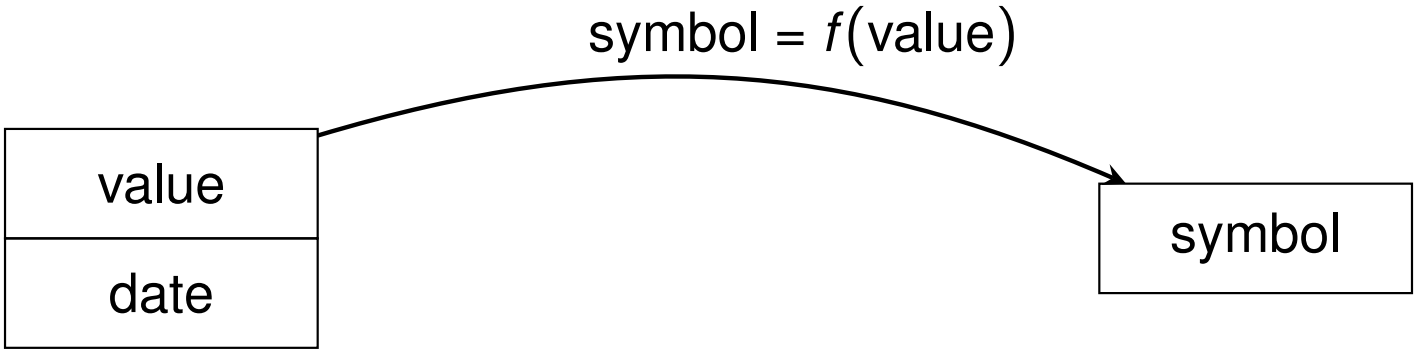
Adaptation of data between DE and TFSM





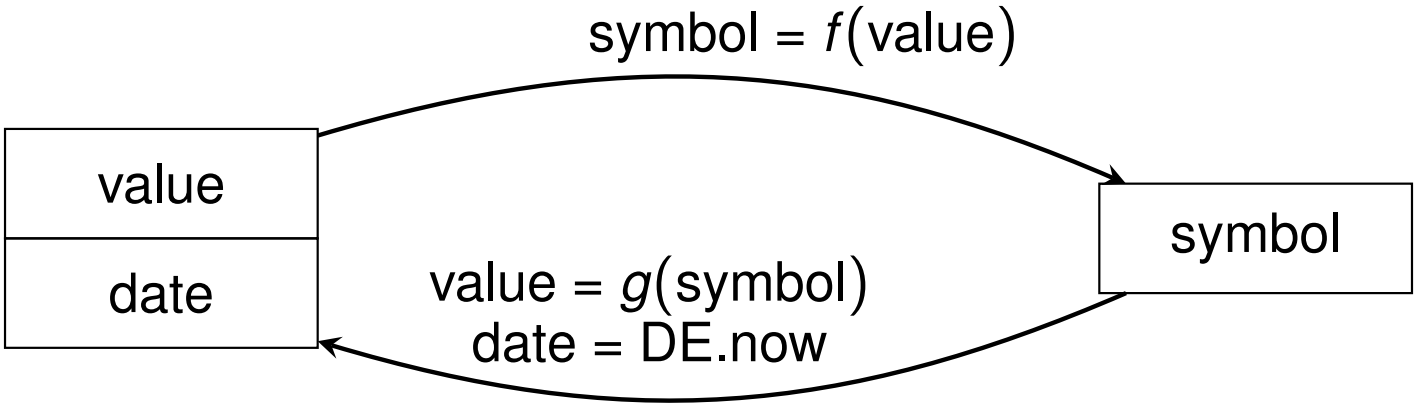
Semantic Adaptation (data)

Adaptation of data between DE and TFSM



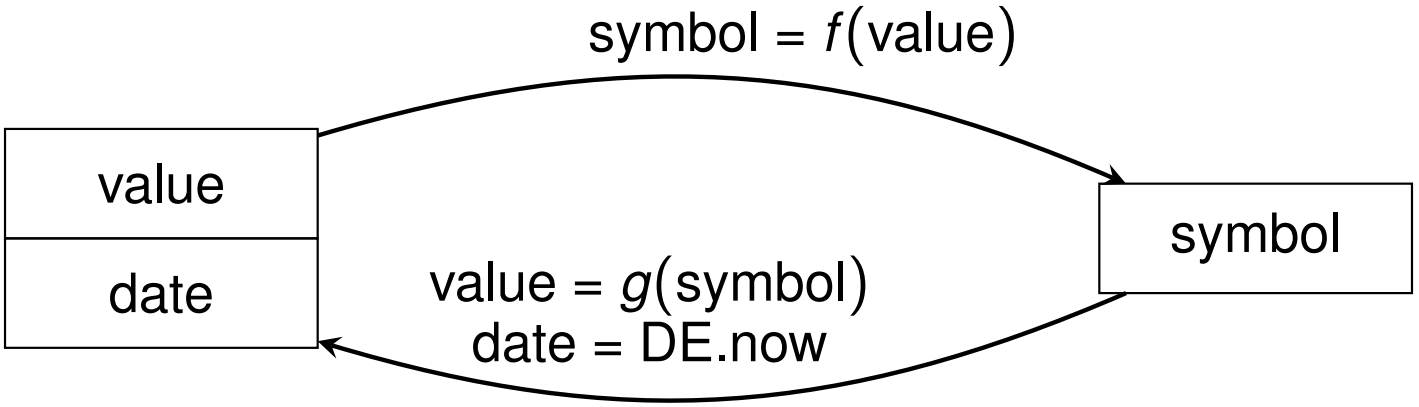
Semantic Adaptation (data)

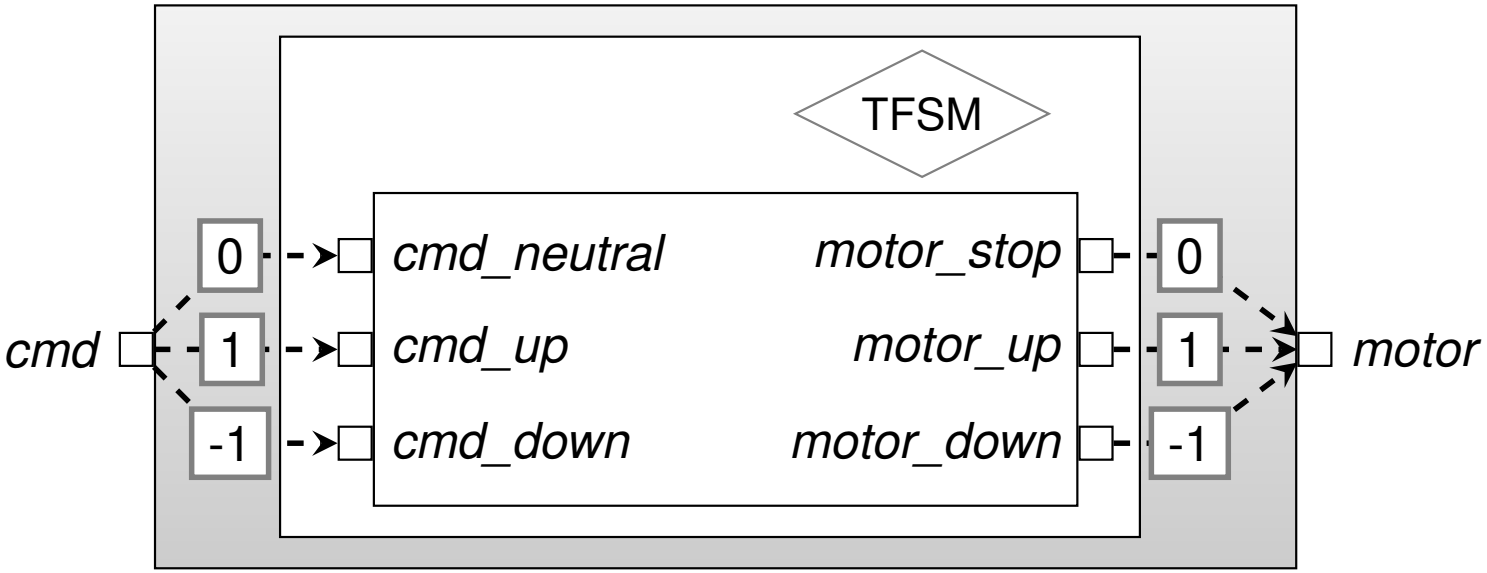
Adaptation of data between DE and TFSM



Semantic Adaptation (data)

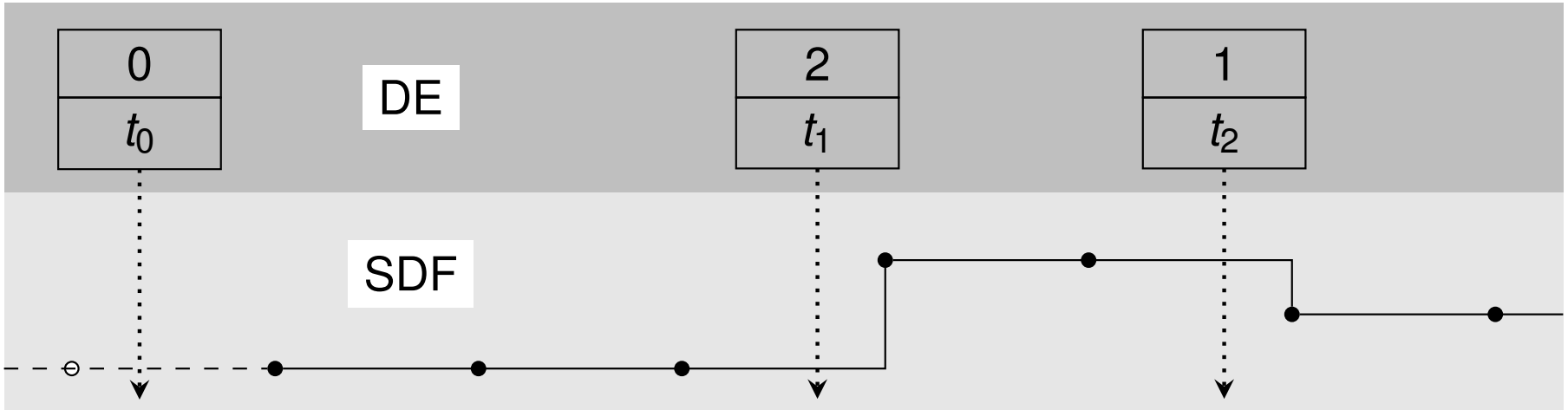
Adaptation of data between DE and TFSM





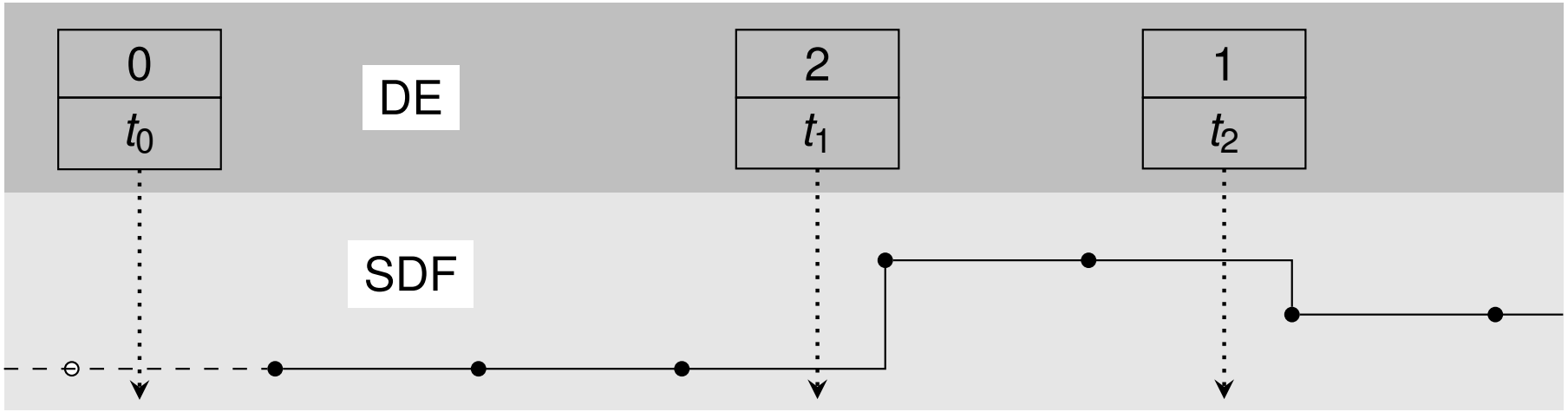
Semantic Adaptation (data)

Adaptation of data between DE and SDF



Semantic Adaptation (data)

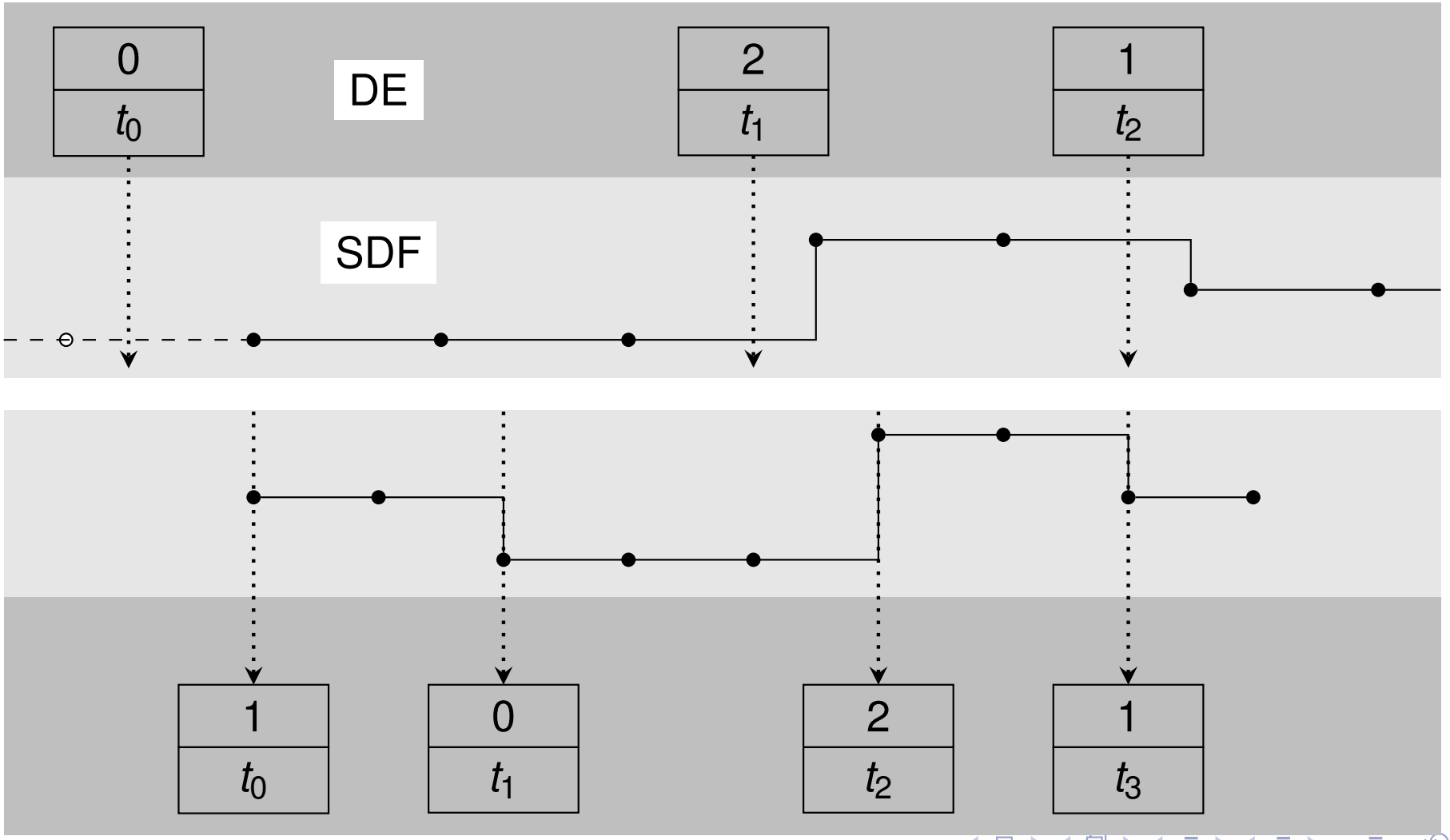
Adaptation of data between DE and SDF



- Relative positioning of events / samples ⇒ adaptation of time
- Occurrence of samples ⇒ adaptation of control

Semantic Adaptation (data)

Adaptation of data between DE and SDF



Semantic Adaptation (time)

Time in Models of Computation

Kinds of time

- Tags on data (dates)
- Durations (elapsed time)

Roles of time

- Partial order on data (causality)
- Durations, dead-lines (control)

Semantic Adaptation (time)

Adaptation of time

Time in DE

- Temporal tags in $\mathbb{R} \times \mathbb{N}$
- Synchronization, causality
- Control of event processing

Time in TFSM

- Elapsed time since a state was entered
- An absolute date has no meaning (*after*, not @)

Adaptation between DE and TFSM

- Match a TFSM duration with a difference between to DE dates
- Consequences on control

Semantic Adaptation (time)

Adaptation of time

Time in DE

- Temporal tags in $\mathbb{R} \times \mathbb{N}$
- Synchronization, causality
- Control of event processing

Time in SDF

- Series of samples, no real notion of date
- No notion of duration

Adaptation between DE and SDF

- Assign a DE date to SDF samples
- Sampling period, consequences on control

Semantic Adaptation (control)

Adaptation of control

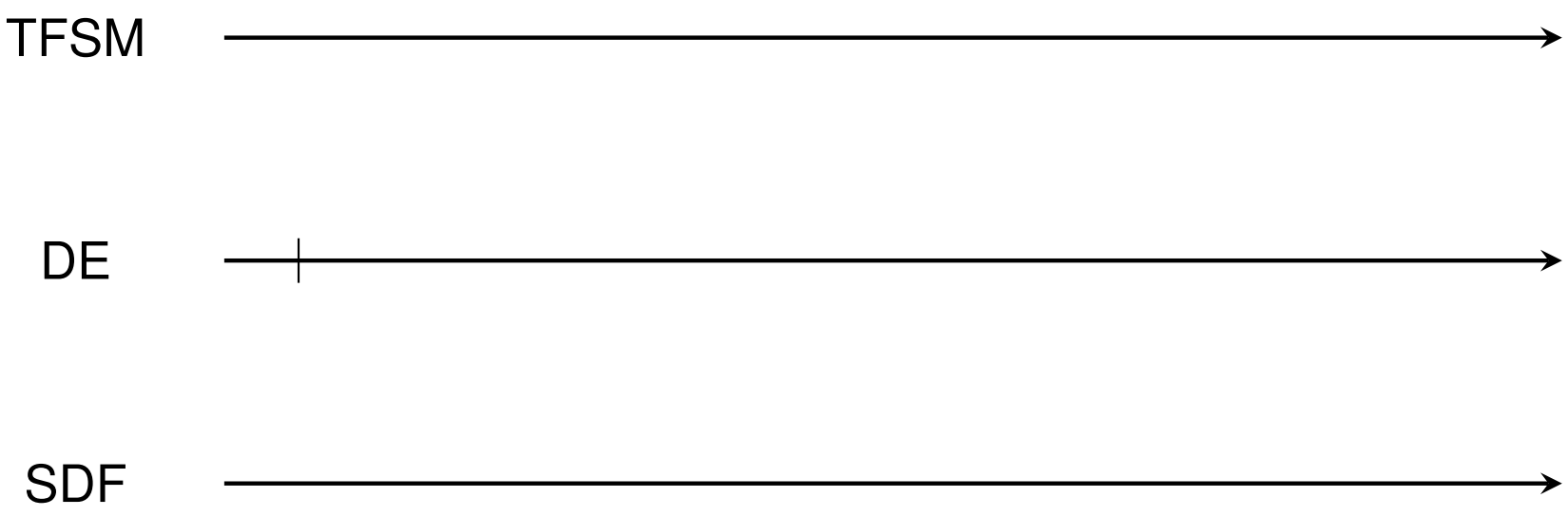
TFSM →

DE →

SDF →

Semantic Adaptation (control)

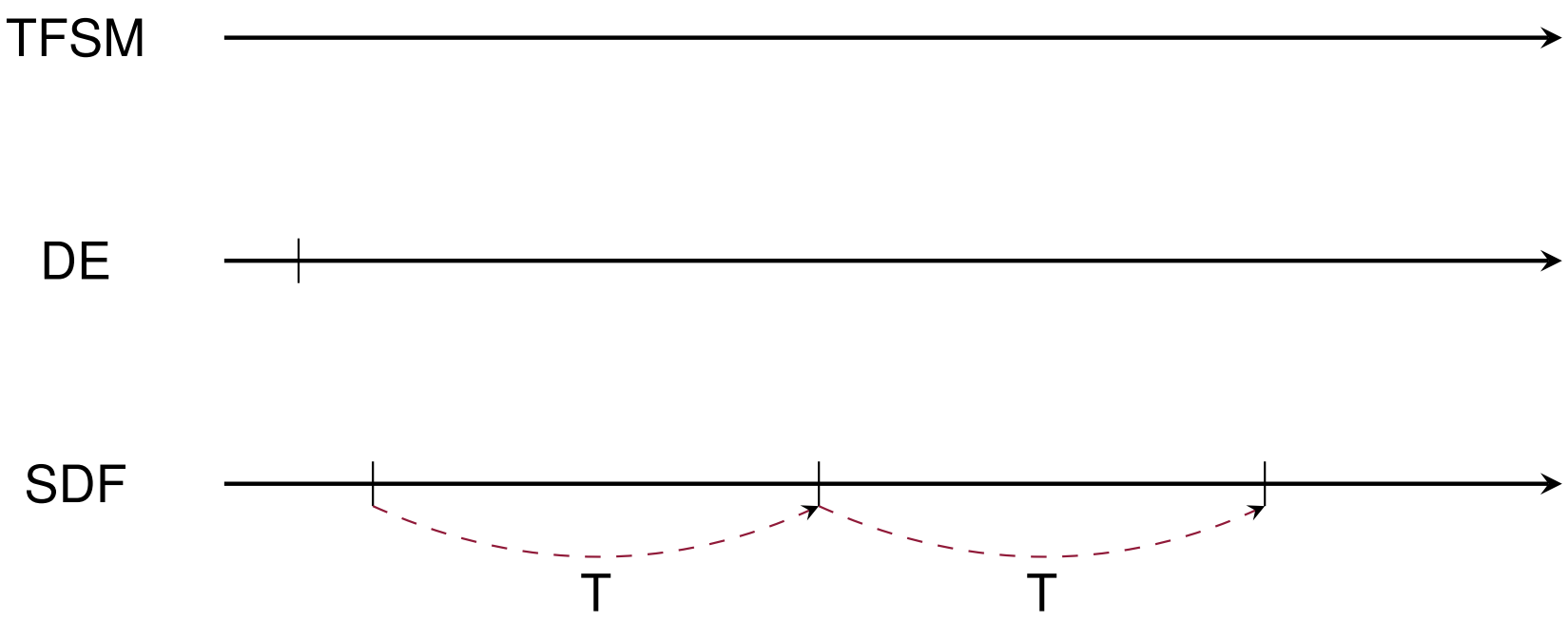
Adaptation of control



Event in DE, no input for the state machine

Semantic Adaptation (control)

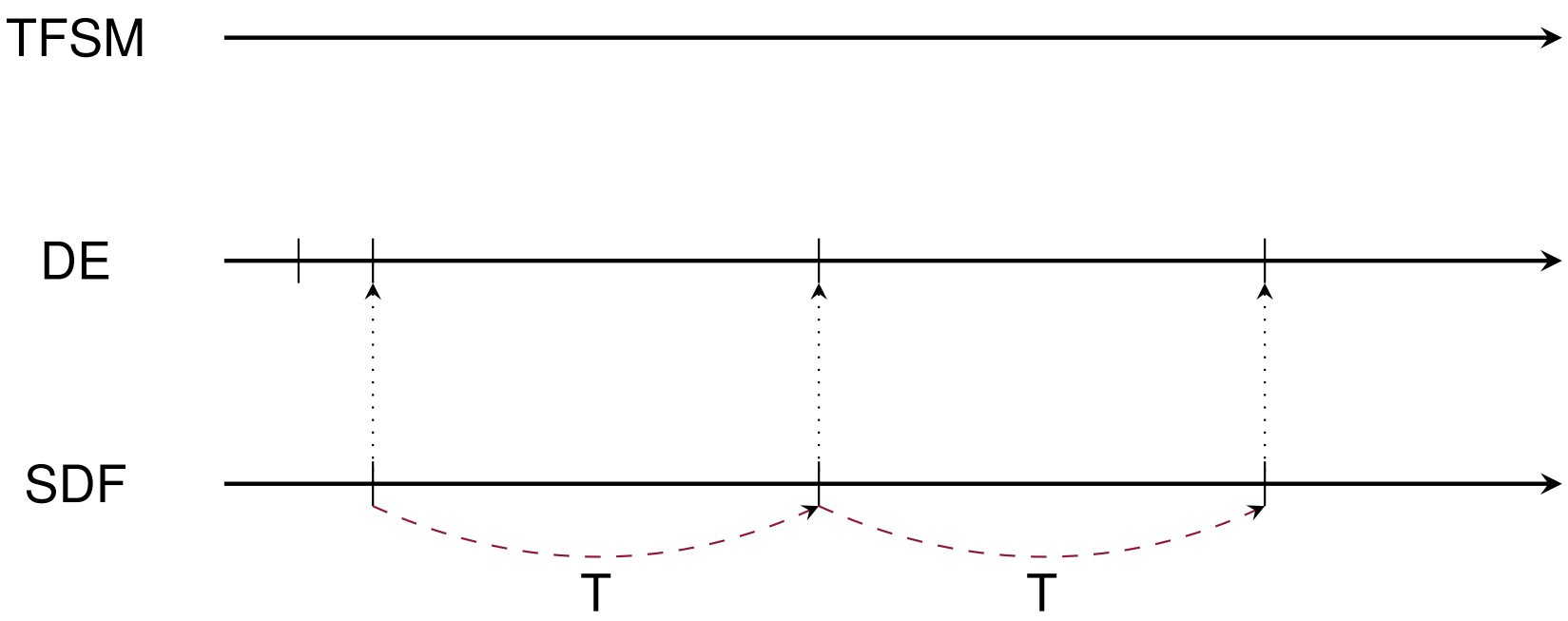
Adaptation of control



Periodic control for SDF $\Rightarrow$ control in DE

Semantic Adaptation (control)

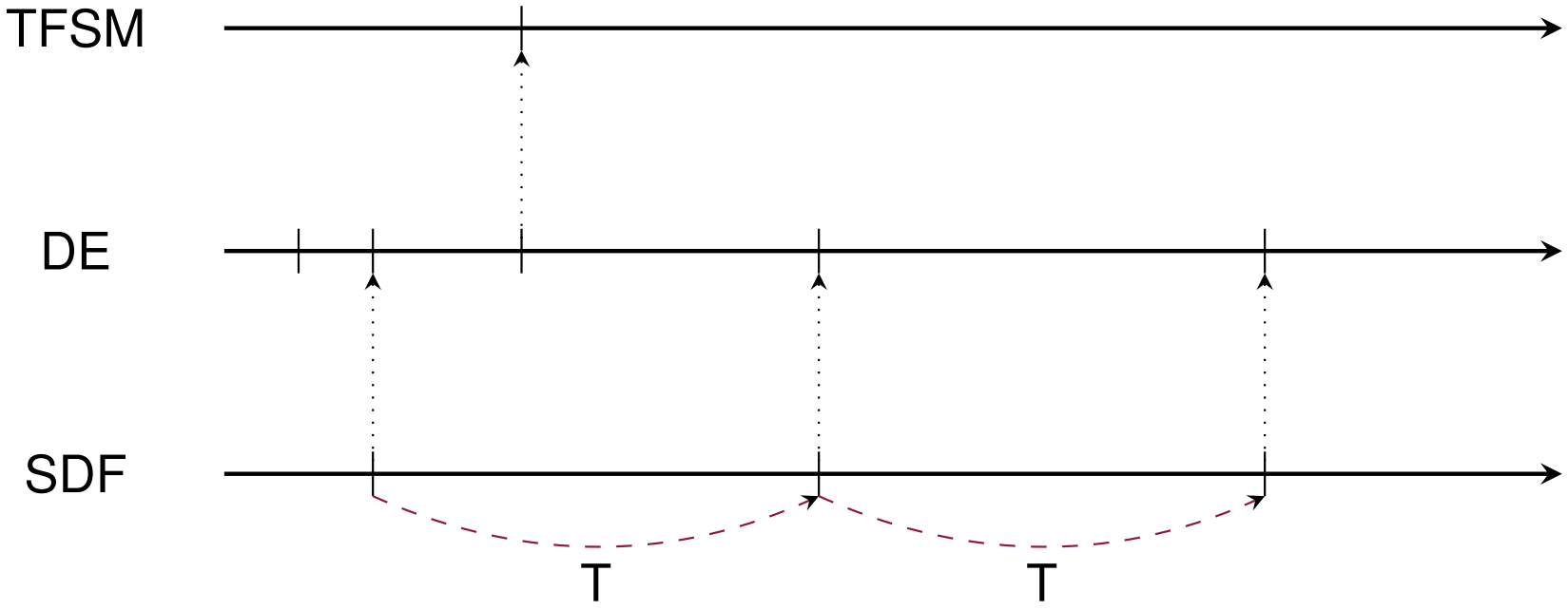
Adaptation of control



Periodic control for SDF $\Rightarrow$ control in DE

Semantic Adaptation (control)

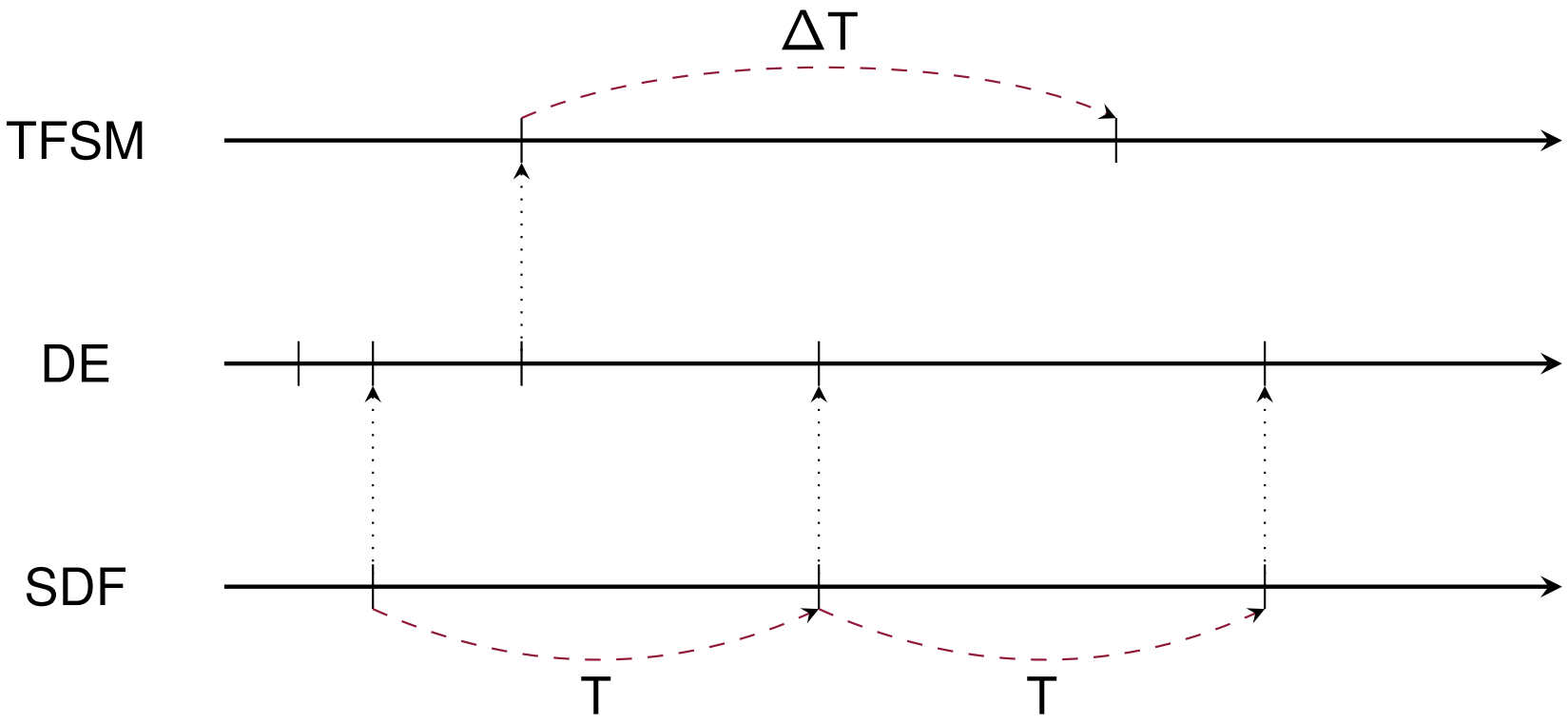
Adaptation of control



Event in DE, input for the state machine
$\Rightarrow$ control in TFSM

Semantic Adaptation (control)

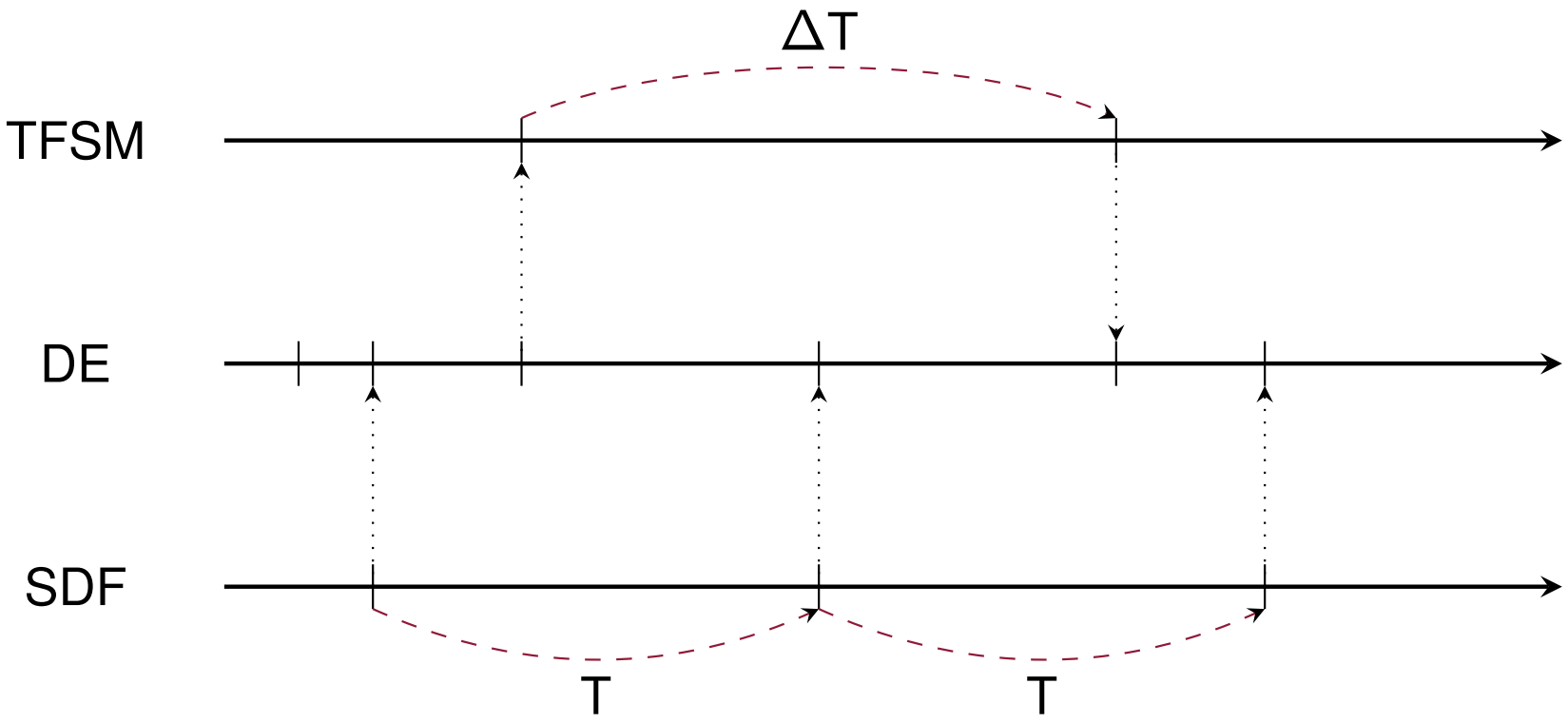
Adaptation of control



Firing of a temporized transition in the state machine
$\Rightarrow$ control in DE

Semantic Adaptation (control)

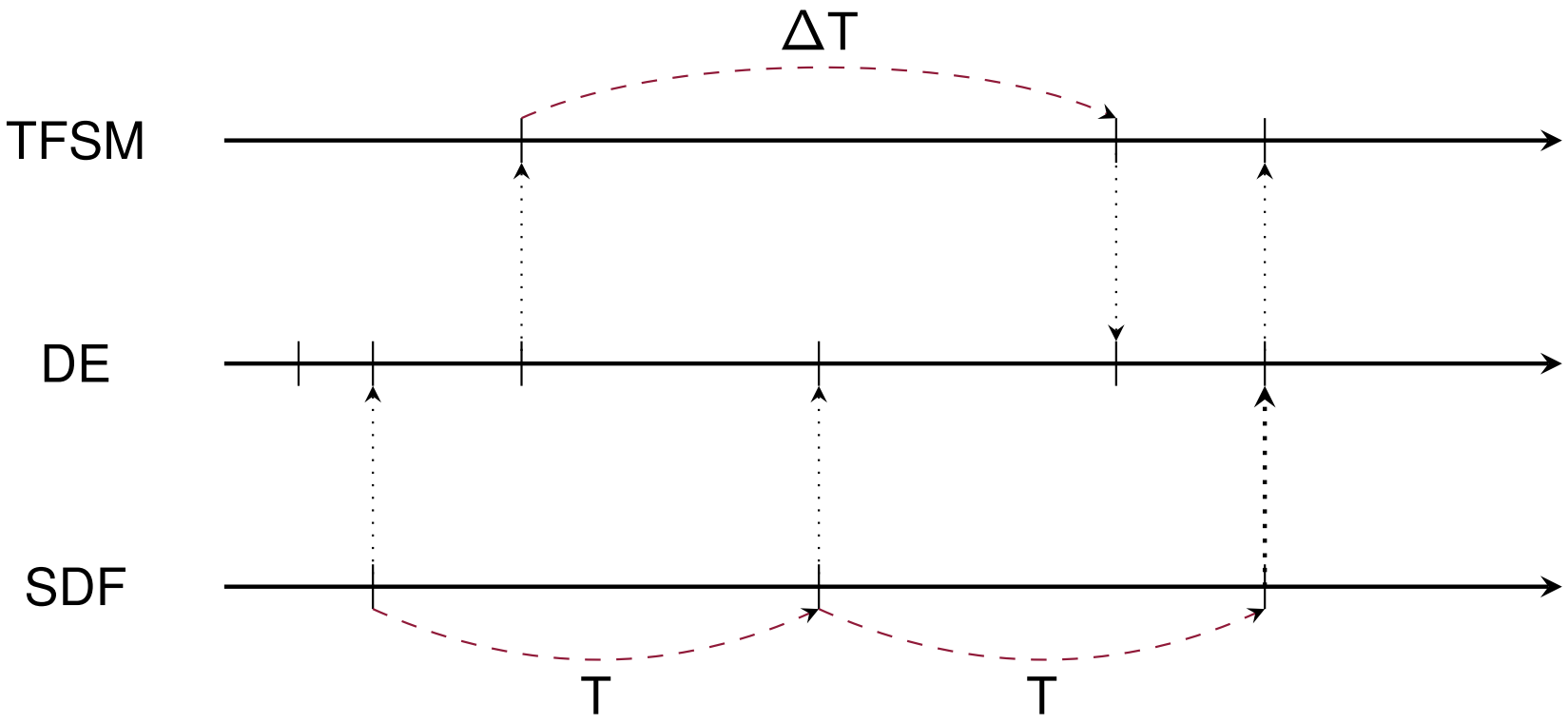
Adaptation of control



Firing of a temporized transition in the state machine
$\Rightarrow$ control in DE

Semantic Adaptation (control)

Adaptation of control



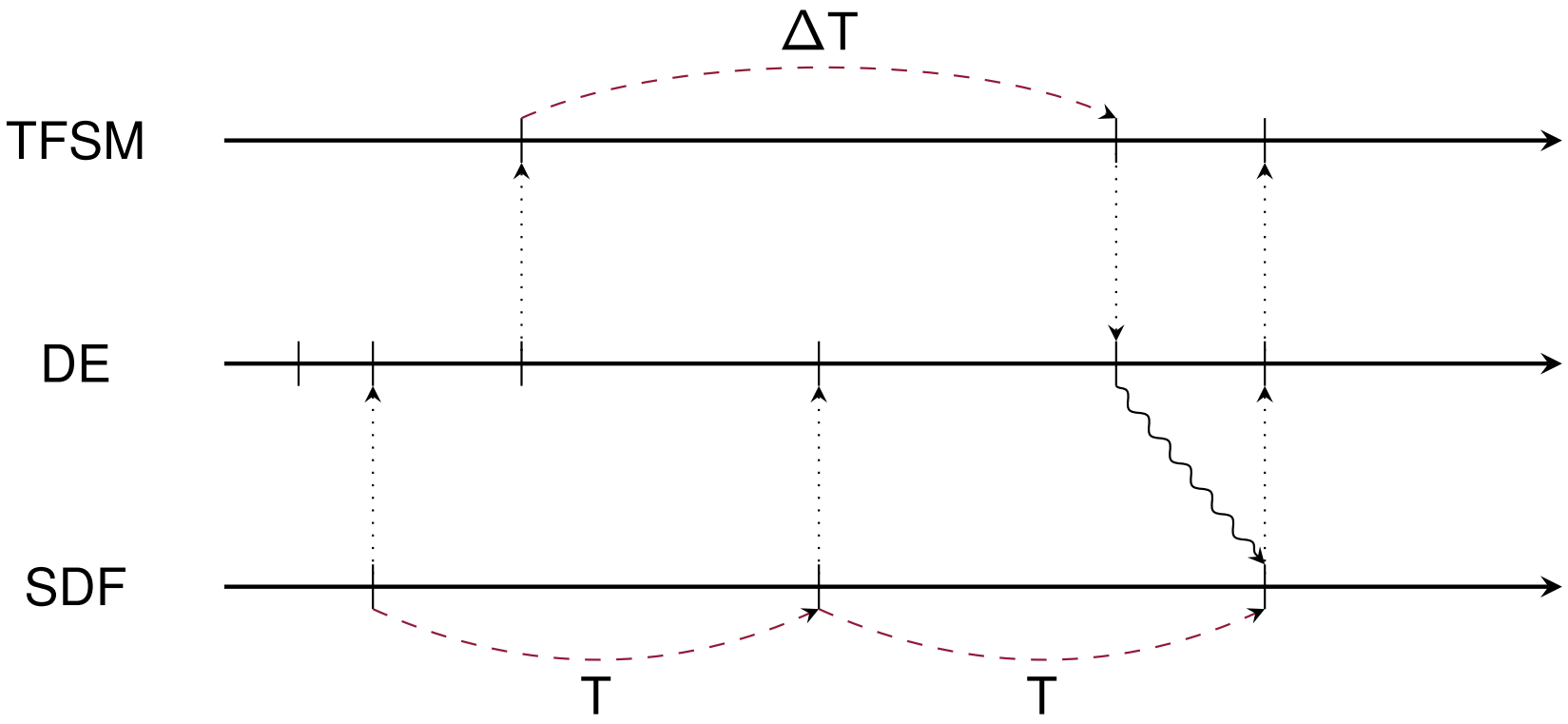
Periodic observation of the SDF model

⇒ control in DE + event for the state machine

⇒ control in TFSM

Semantic Adaptation (control)

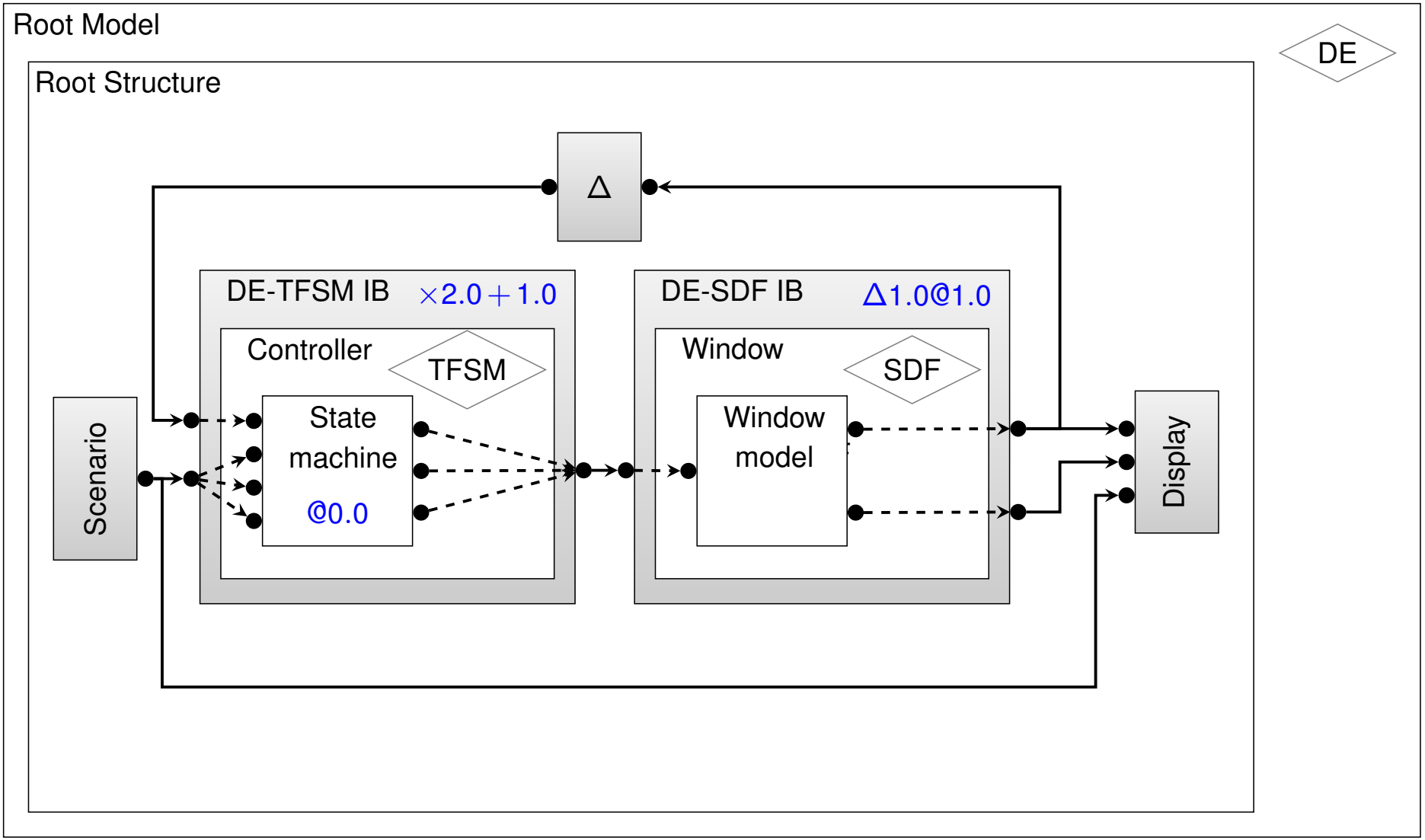
Adaptation of control



Data do not create control for SDF

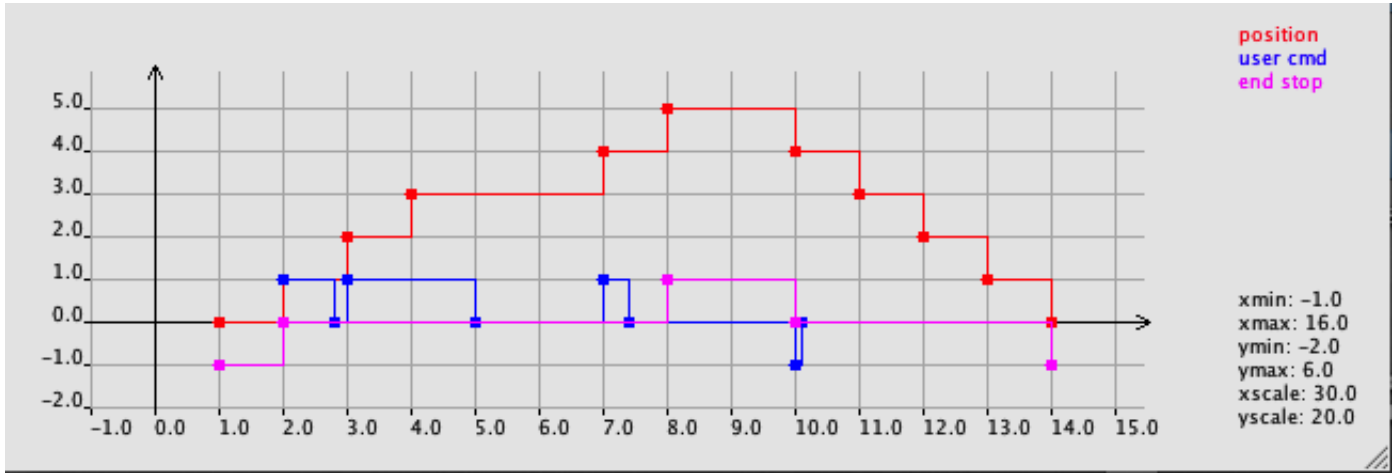
⇒ data is memorized at the interface

ModHel'X Model of the Power Window

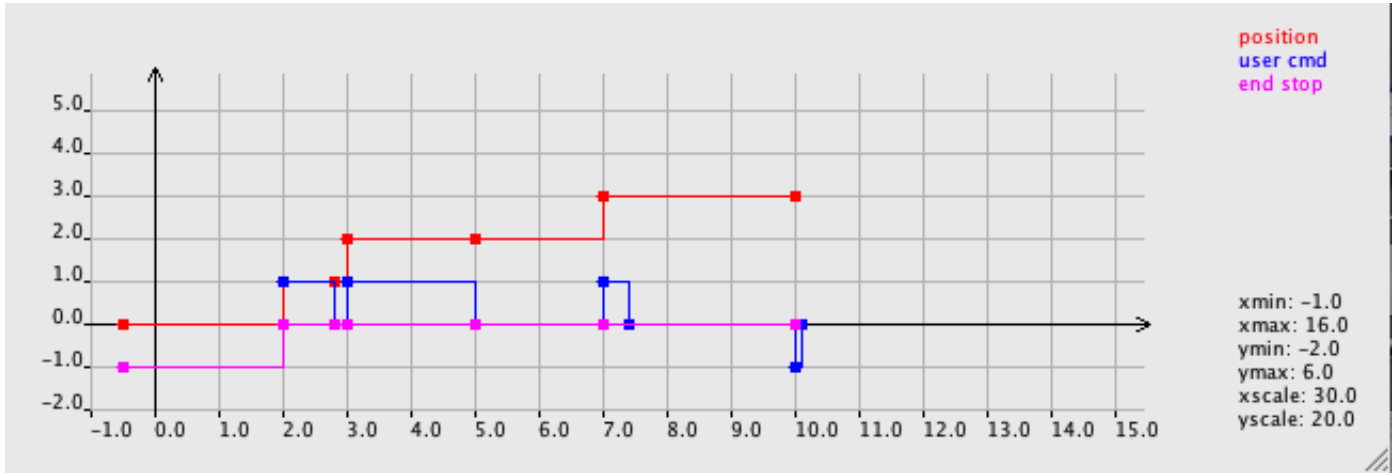


scenario: 1 @2.0; 0 @2.8; 1 @3.0; 0 @5.0; 1 @7.0; 0 @7.4; -1 @10.0; 0 @10.1;

Semantic Adaptation: The Power Window



Correct semantic adaptation with periodic SDF and edge detection



Default semantic adaptation with data driven SDF

Semantic Adaptation

Current implementation

- Adaptation of data: parameterized adaptation patterns
- Adaptation of time: relations between tags on clocks
- Adaptation of control: implication relations between clocks
  + creation of ticks with a given tag

Clocks, ticks and tags

- Tagged Events Specification Language
- Constructive solver used for:
  - Adaptation of time and control in ModHel'X
  - Driving ModHel'X simulations
    (adaptation of time and control with the real world)

Conclusion and Perspectives

ModHel'X provides:

- a generic metamodel for heterogeneous modeling
- an abstract semantics for model execution
- concrete semantics in the form of MoCs
- semantic adaptation between heterogeneous models
- a time model for:
  - defining the adaptation of time and control
  - synchronizing the execution engine with the simulation environment
  - defining hierarchical time structures

Perspectives

- define MoCs in terms of clocks and Domain Specific Actions
- use hierarchical time structures for executing models
- validate TESL implementations with respect to CCSL specifications

Backup

TESL: Tagged Events Specification Language

A language for:

- building instants on a set of clocks
- finding the *next* instant, called "now"
- using causality between event occurrences
- using synchronization between time scales (tag sets)

Differences with CCSL

- only constructive statements (no non-determinism)
- ticks on clocks have a tag
- tag relations between clocks
- a tick can be specified to occur at a given tag

TESL: Tagged Events Specification Language

Clocks

- time scale + arithmetic operators on tags $(+, -, \times, /)$
- set of ticks, each tick has a tag
- "now" tick, which belongs to the next instant
- special $\perp$ tag, meaning "as soon as possible"

Relations

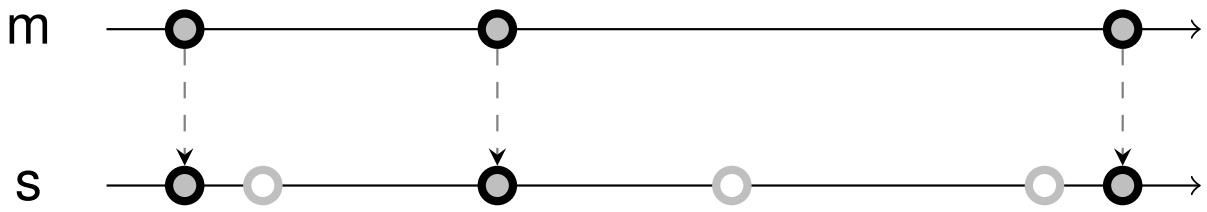
- implication relations between clocks
- filtering of implications (pattern, counting, delay)
- tag relations = non-decreasing functions

Solver

- use implication relations to propagate "now" ticks
- use tag relations to find "now" ticks and synchronize tags
- compute the next instant as a (smallest or largest) fixed point

Implication relations

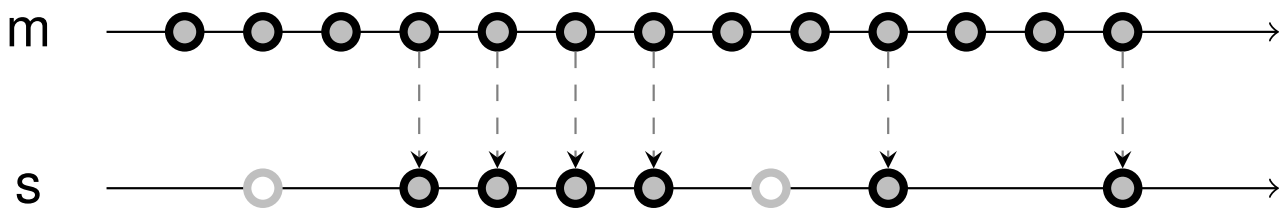
m implies s



Implication relations

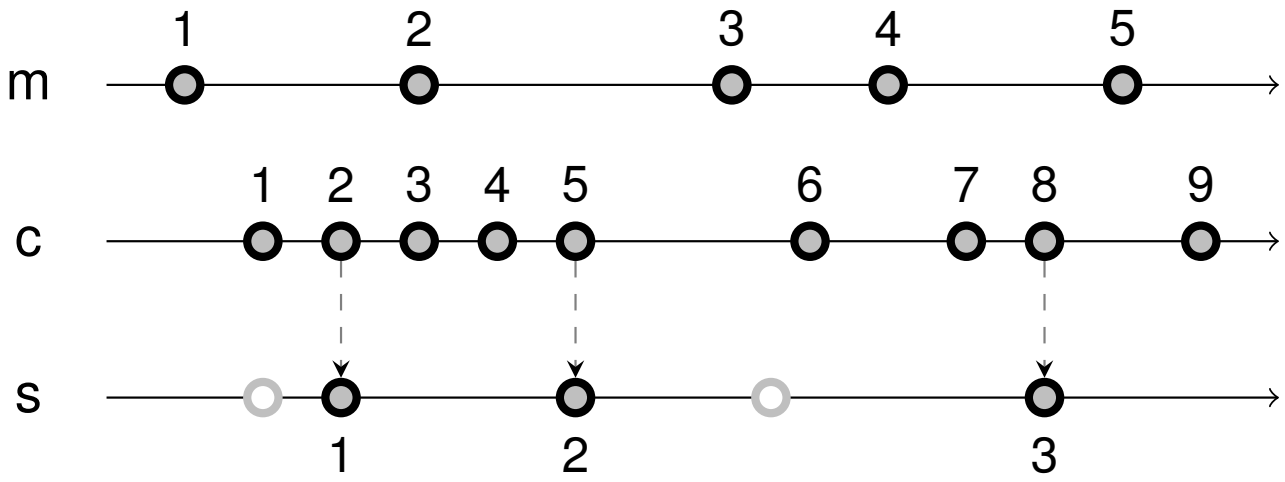
m filtered by [3,4,2,1] implies s

Pattern: [skip, keep, repeated skip, repeated keep]



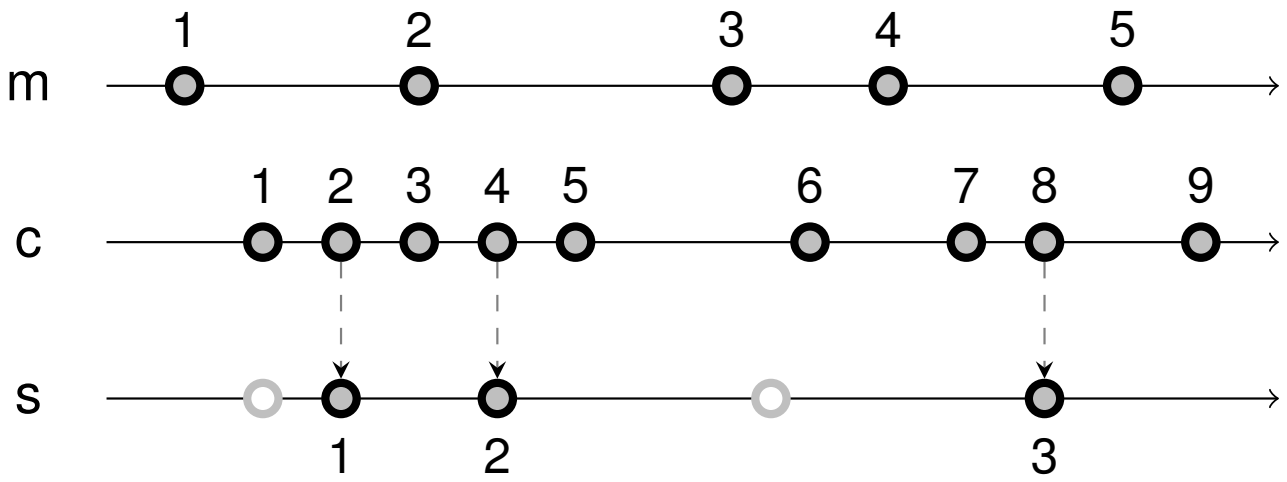
Implication relations

m delayed by 2 on c implies s



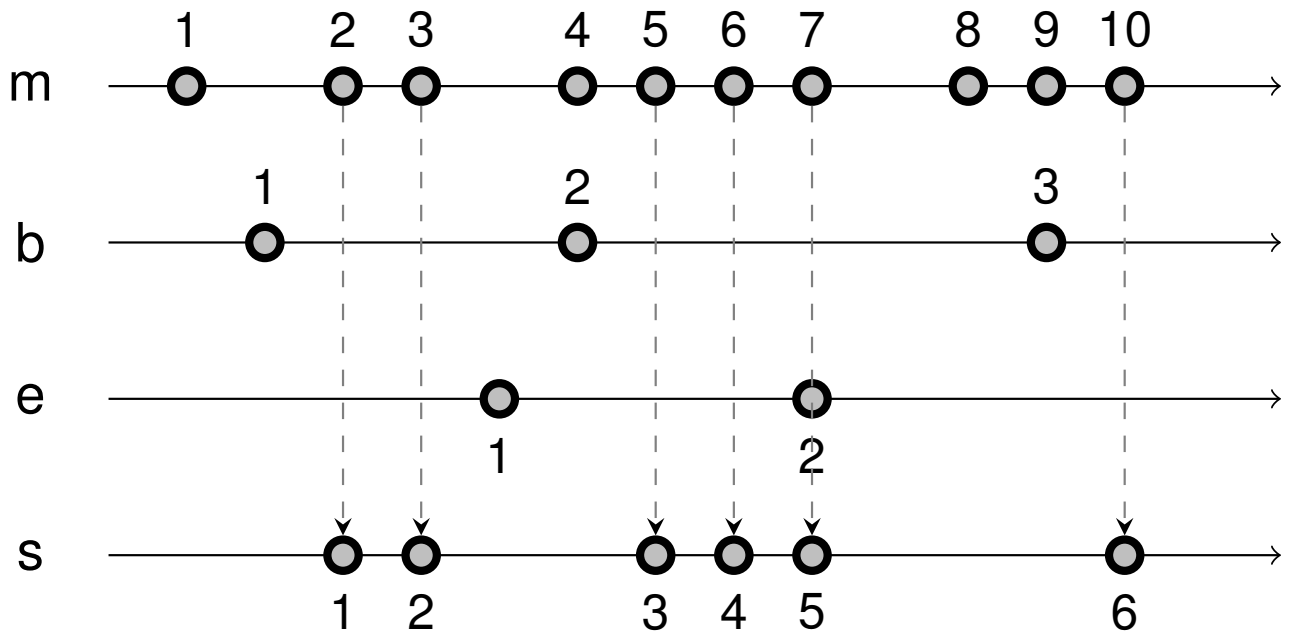
Implication relations

m **immediately** delayed by 2 on c implies s



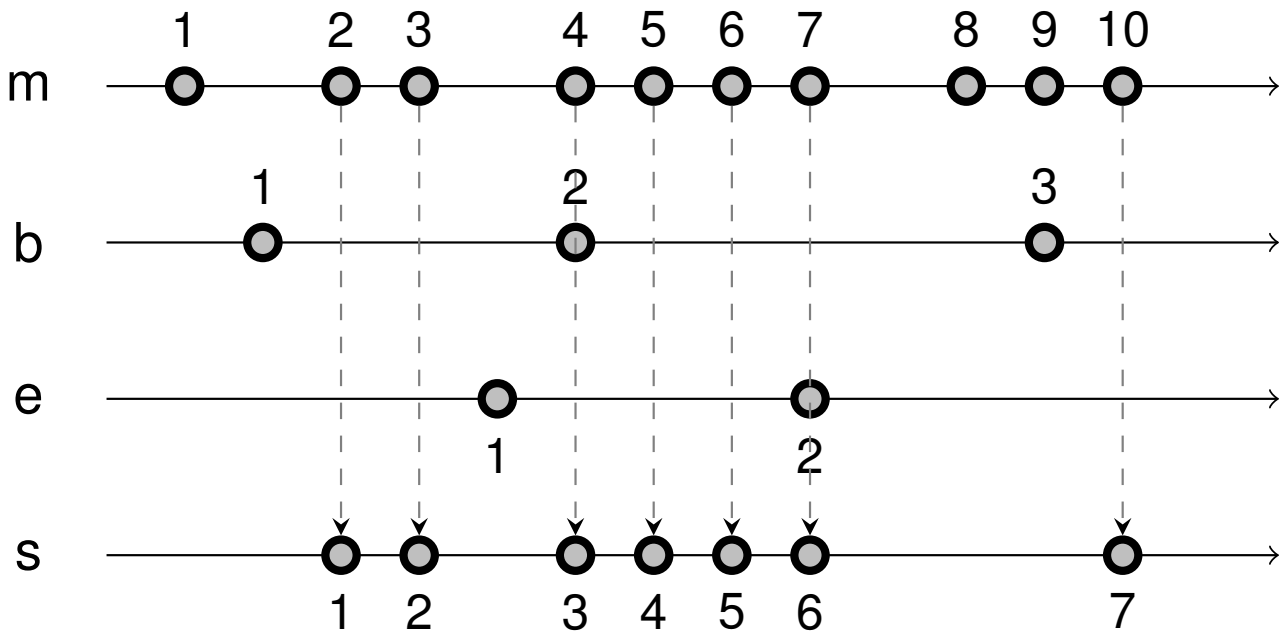
Implication relations

M sustained from B to E implies S



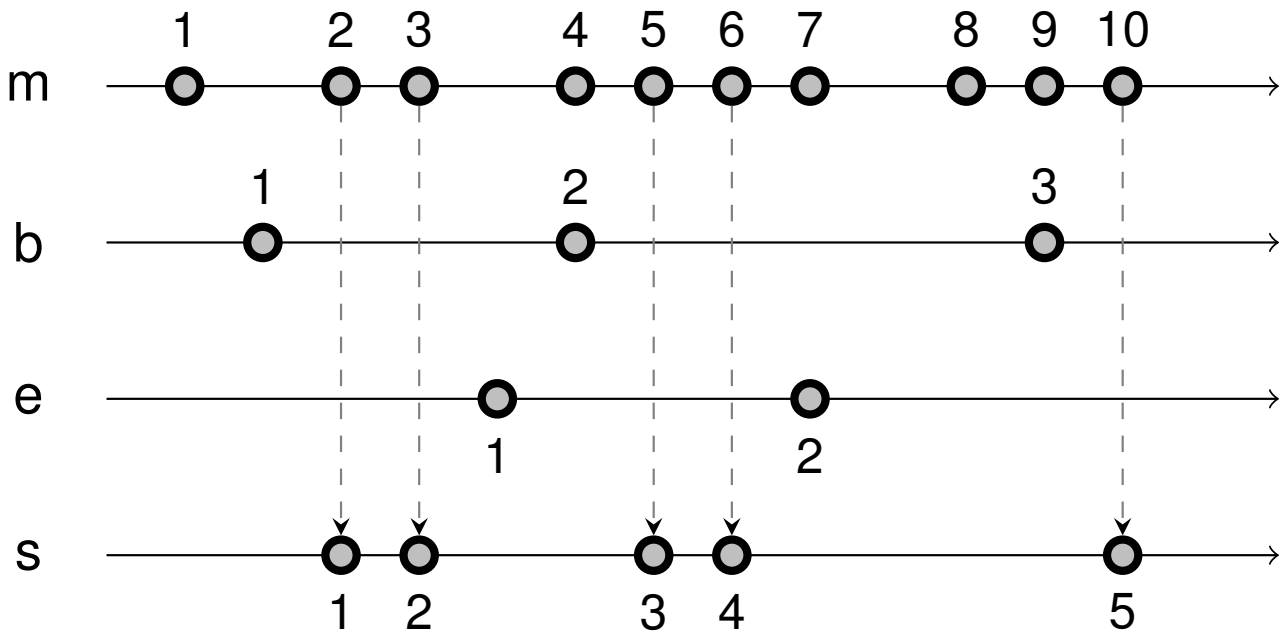
Implication relations

M sustained immediately from B to E implies S



Implication relations

M sustained weakly from B to E implies S



Tag relations

Tag relation between clocks =

relation between tags of simultaneous ticks.

Relation between two clocks a and b, $T = (d, r)$ with:

$d : dom(a) \rightarrow dom(b)$

$r : dom(b) \rightarrow dom(a)$

Preserving causality: both d and r must be non-decreasing.

d and r may not be the inverse function of one another.

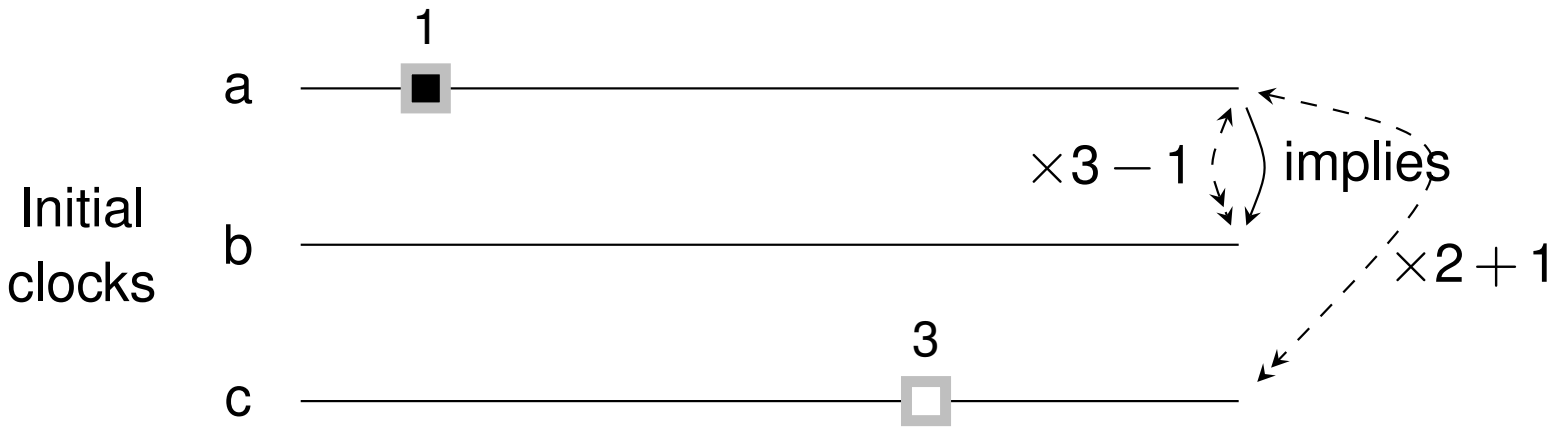
Currently: only affine tag relations $d(t) = \alpha t + b$, $r(t) = (t - b)/\alpha$

Tag relations are bidirectional:

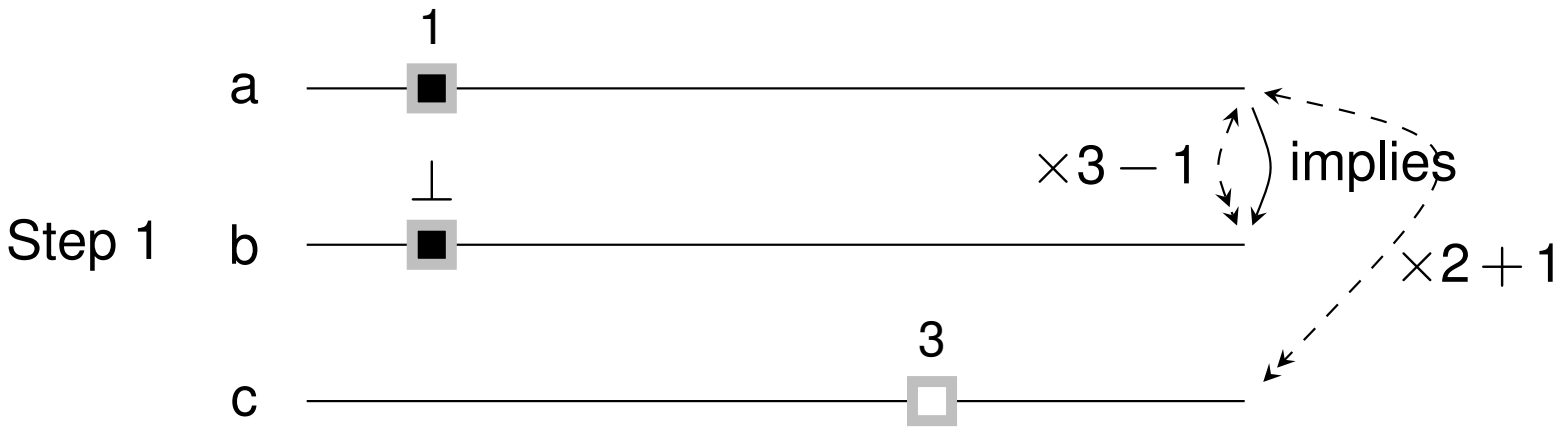
same tag $\Rightarrow$ simultaneous

simultaneous $\Rightarrow$ same tag.

Solving TESL specifications

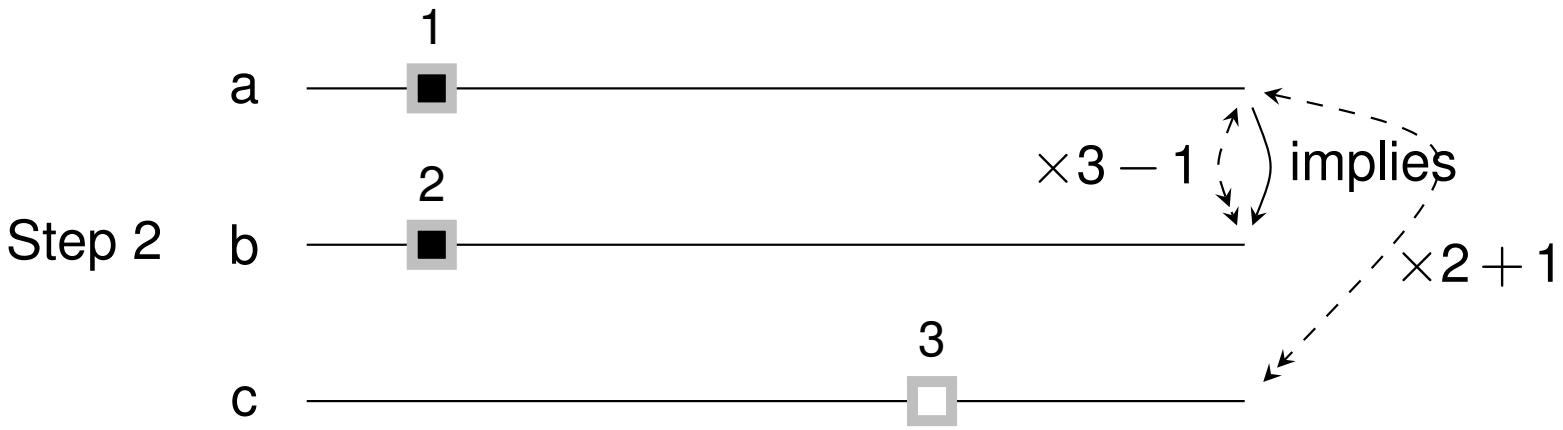


Solving TESL specifications



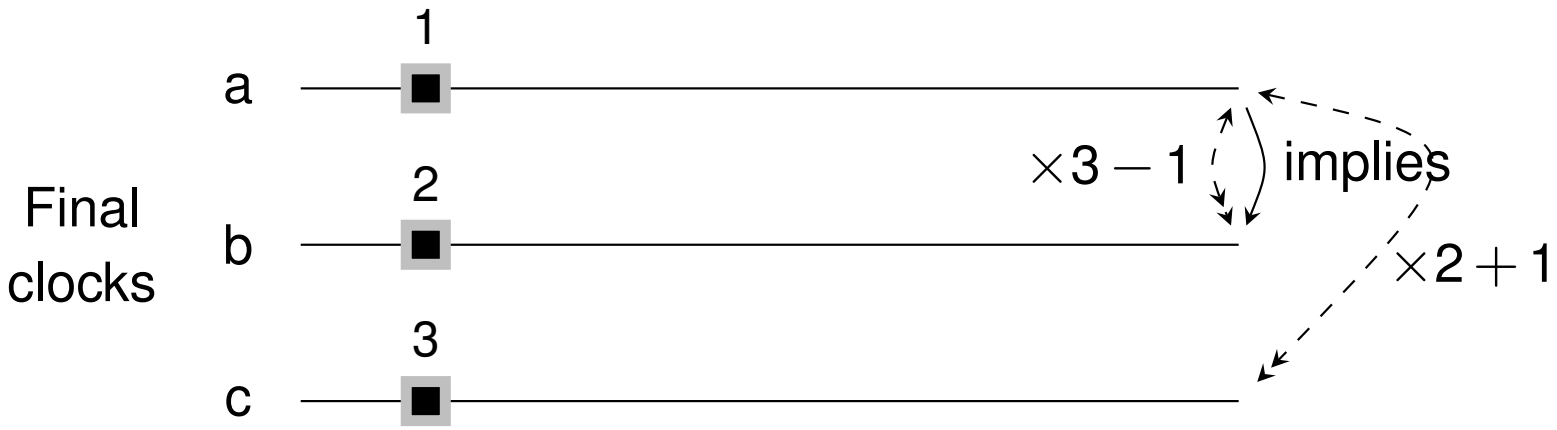
Applying the implication relation

Solving TESL specifications



Using the tag relation between a and b

Solving TESL specifications



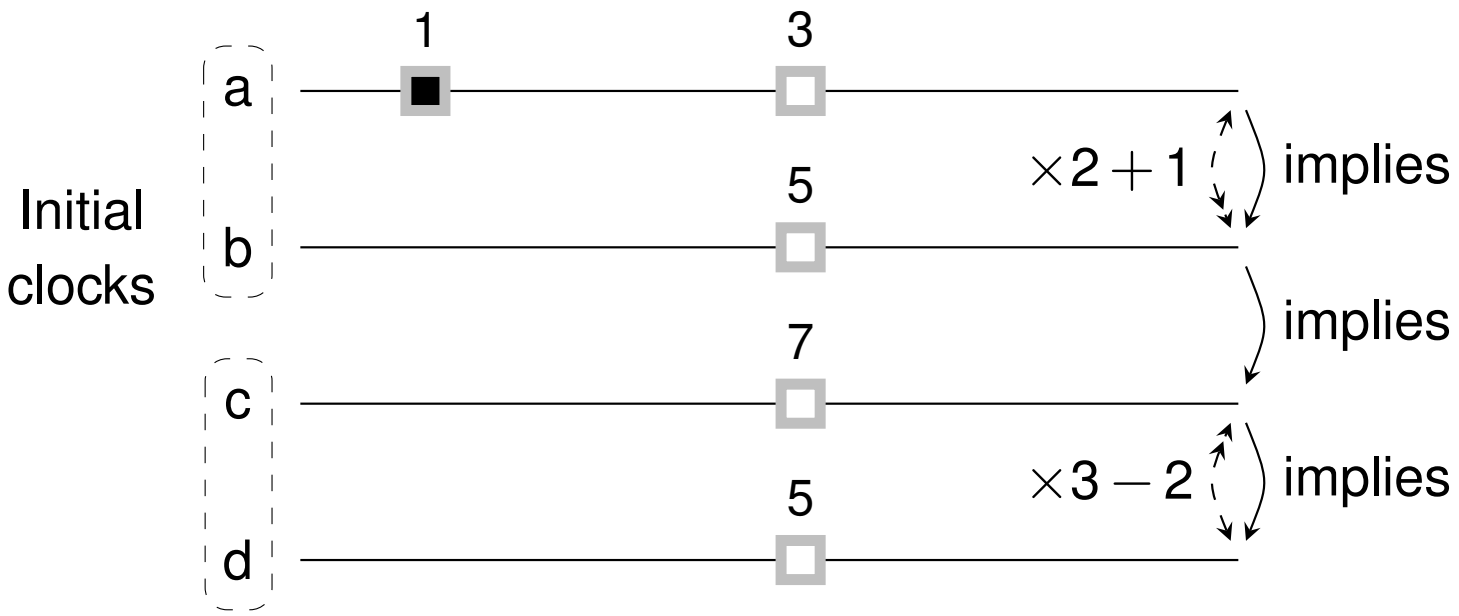
Using the tag relation between b and c

Dealing with time islands

A time island is a connected subgraph of the tag relation graph.

Merging a floating tick with a tagged tick

can be done independently in each time island.

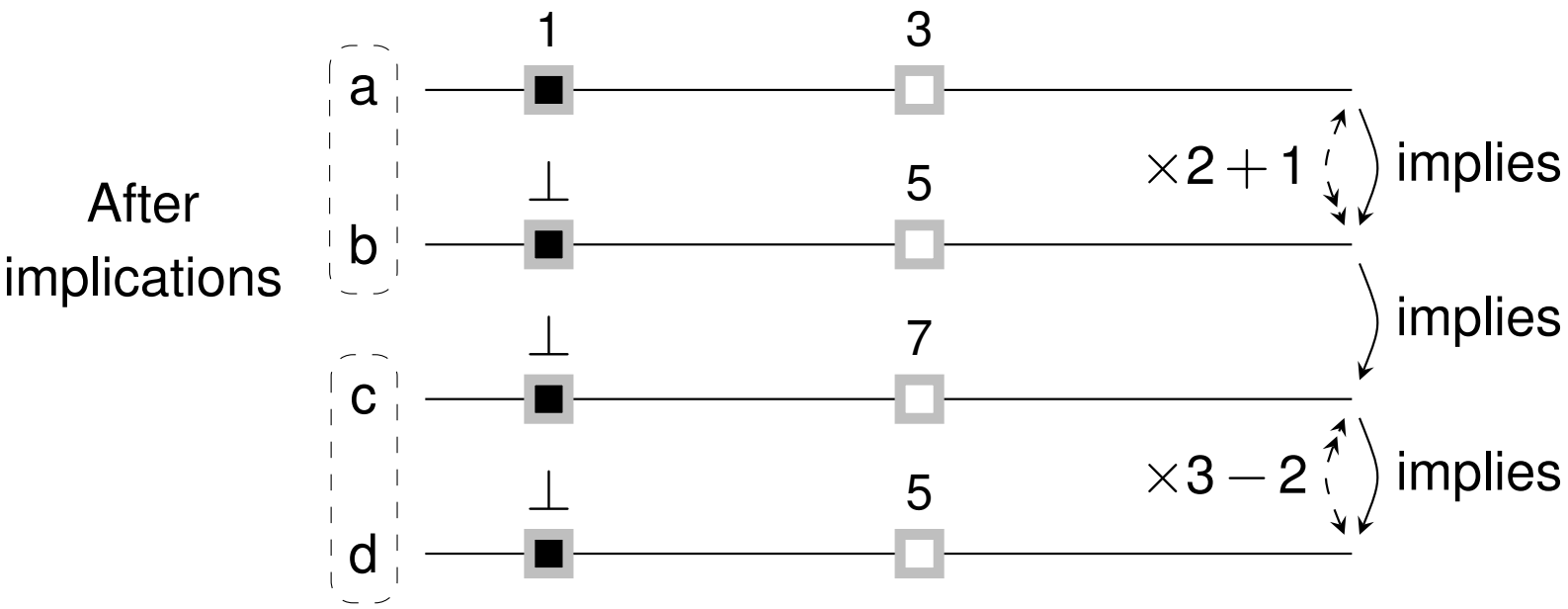


Dealing with time islands

A time island is a connected subgraph of the tag relation graph.

Merging a floating tick with a tagged tick

can be done independently in each time island.



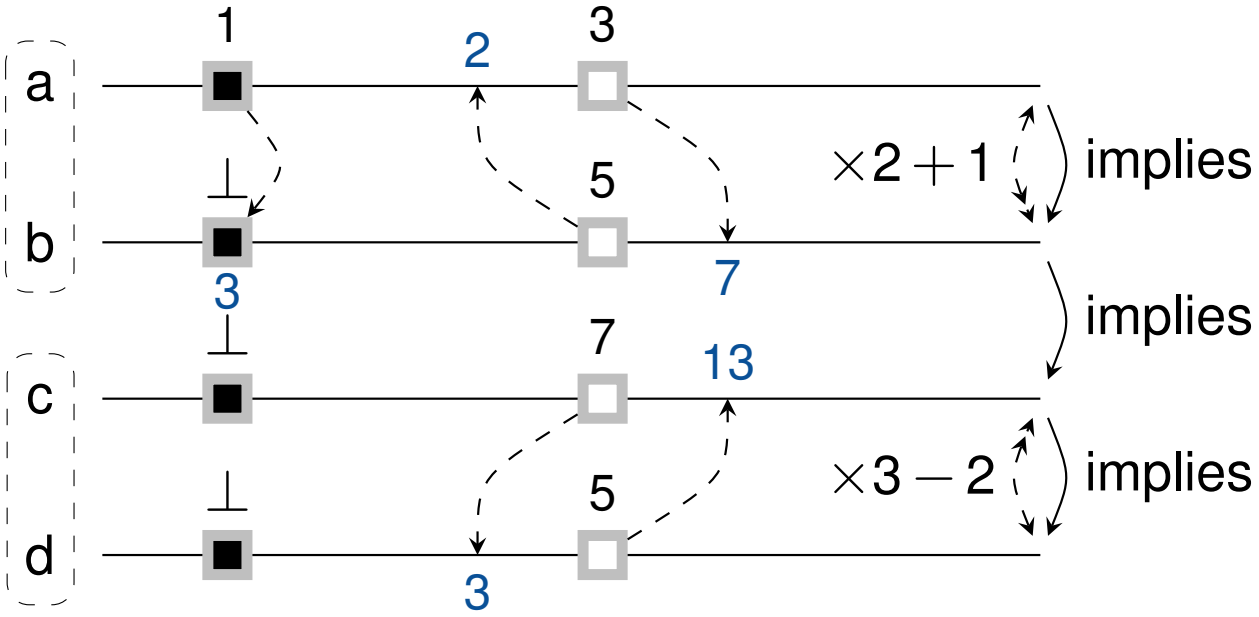
Dealing with time islands

A time island is a connected subgraph of the tag relation graph.

Merging a floating tick with a tagged tick

can be done independently in each time island.

Using tag relations

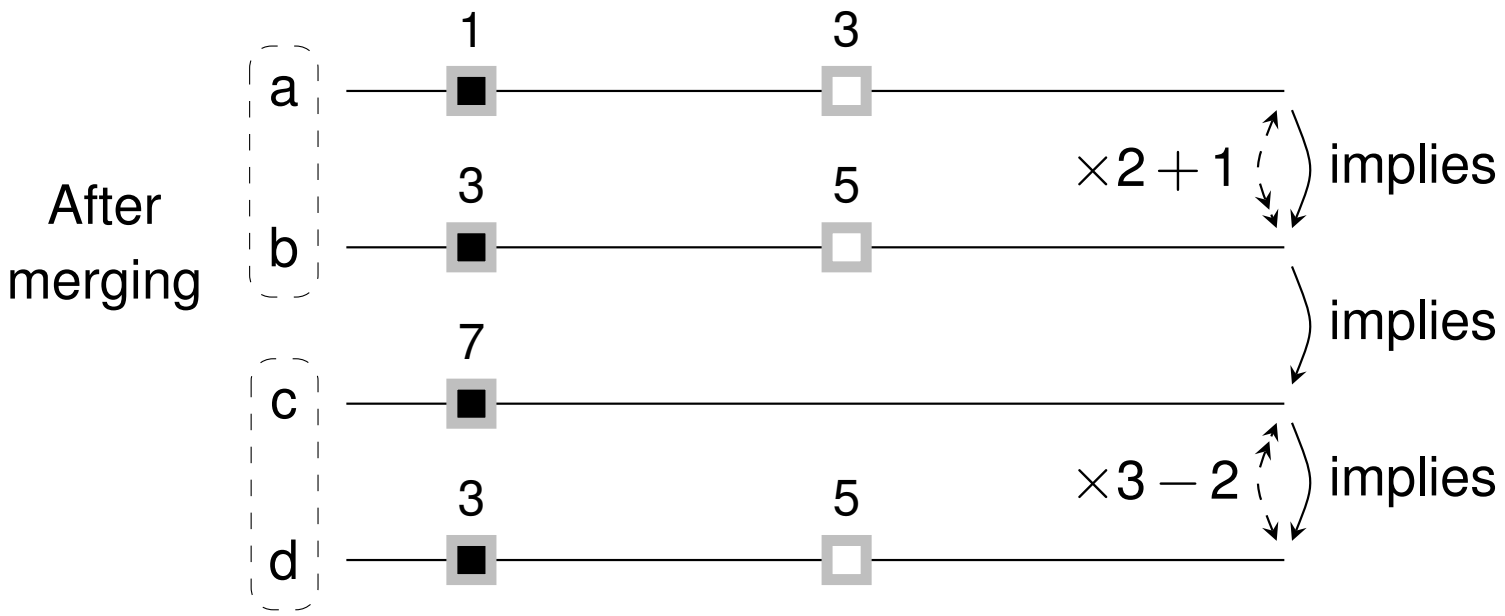


Dealing with time islands

A time island is a connected subgraph of the tag relation graph.

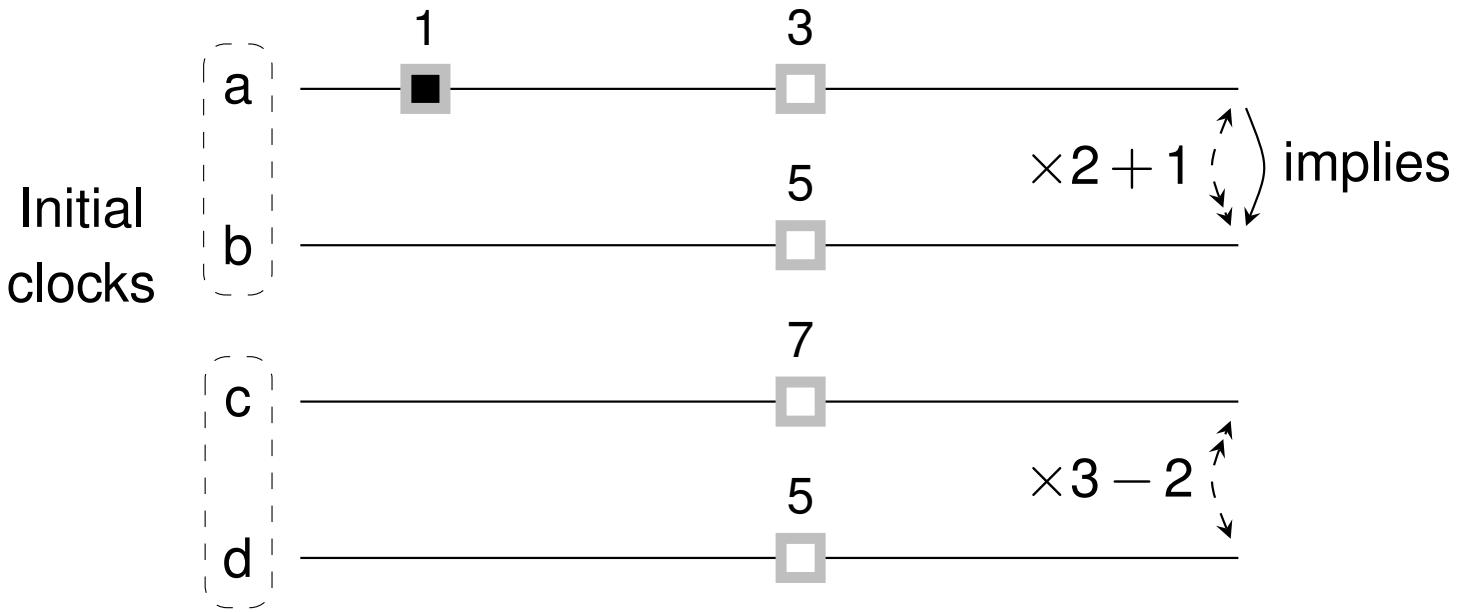
Merging a floating tick with a tagged tick

can be done independently in each time island.



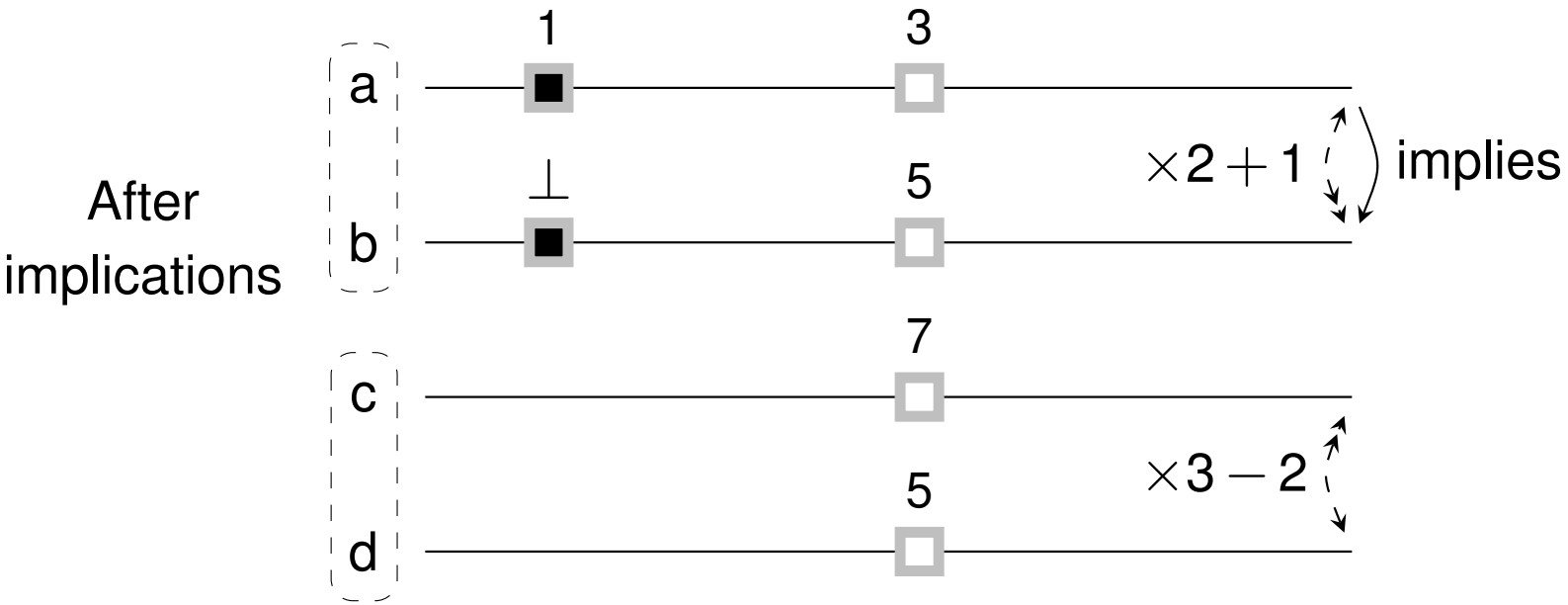
Greedy clocks

When a tick is set on a clock and nothing prevents to set it to now we set it to now if the clock is greedy



Greedy clocks

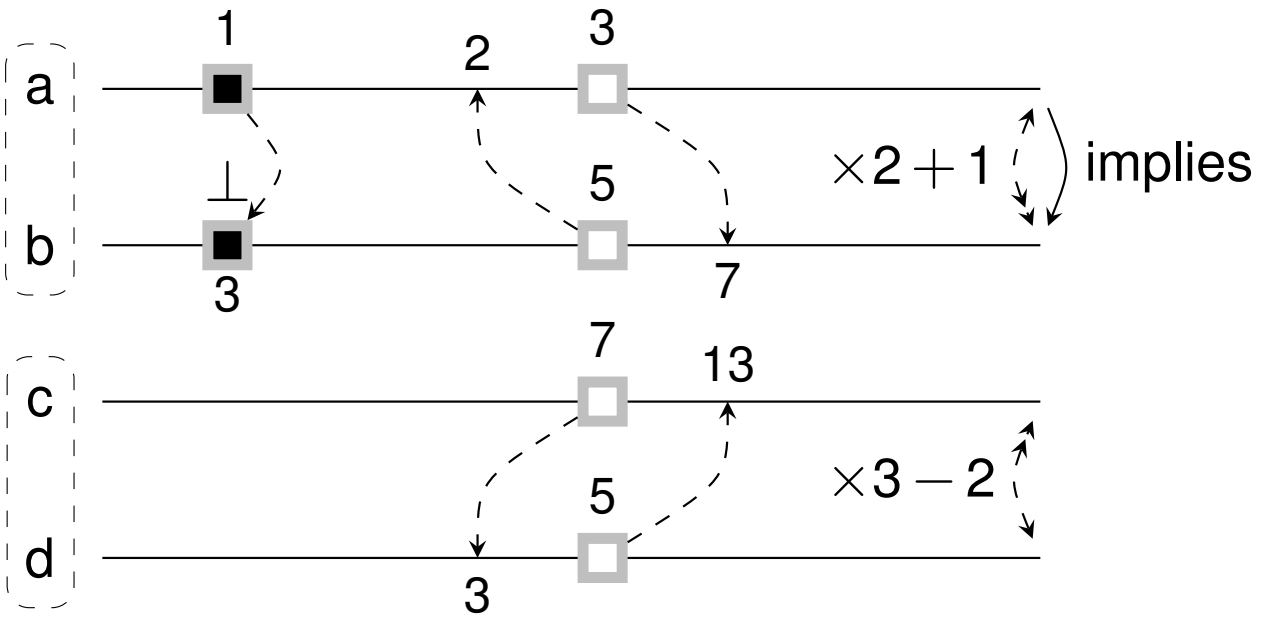
When a tick is set on a clock and nothing prevents to set it to now
we set it to now if the clock is greedy



Greedy clocks

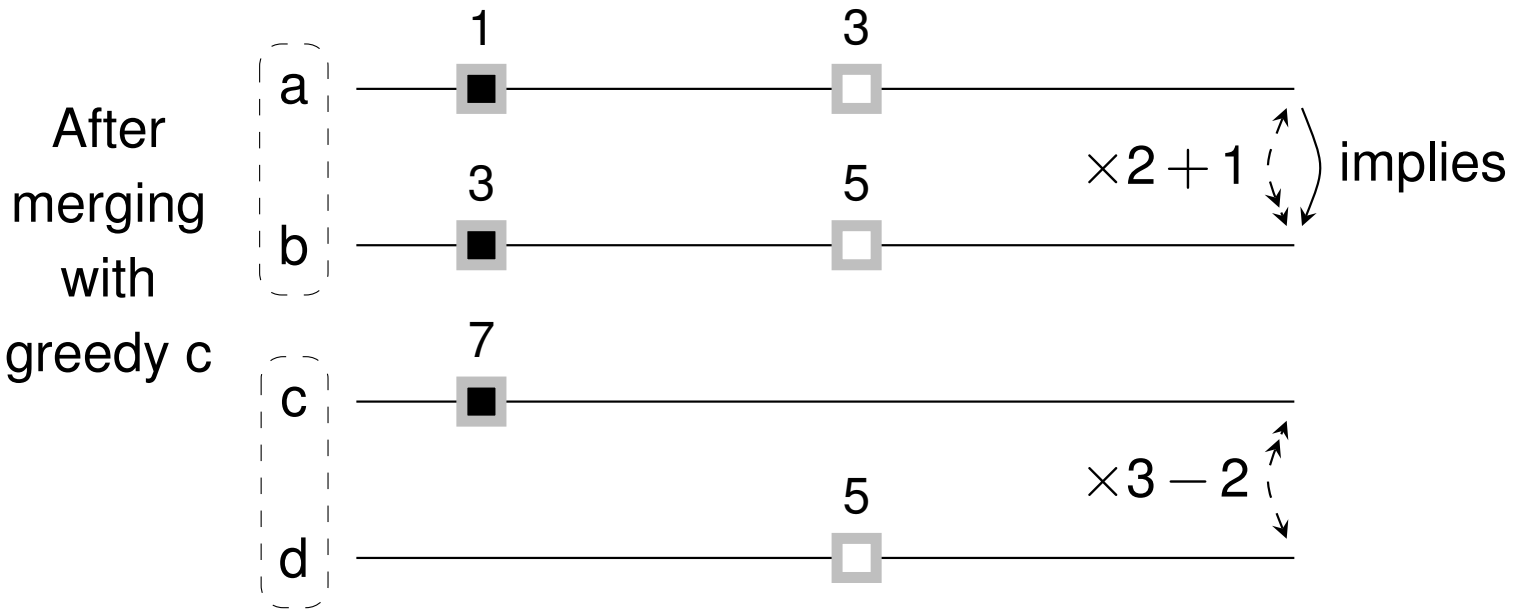
When a tick is set on a clock and nothing prevents to set it to now
we set it to now if the clock is greedy

Using tag relations



Greedy clocks

When a tick is set on a clock and nothing prevents to set it to now we set it to now if the clock is greedy



Use of TESL in ModHel'X

Adaptation of control

- each model has an observation clock
- the observation clock of a model implies the observation clock of the embedding model
- at the end of each snapshot, a block can create a tick to specify a dead-line for its next observation
- interface blocks can set implication relations

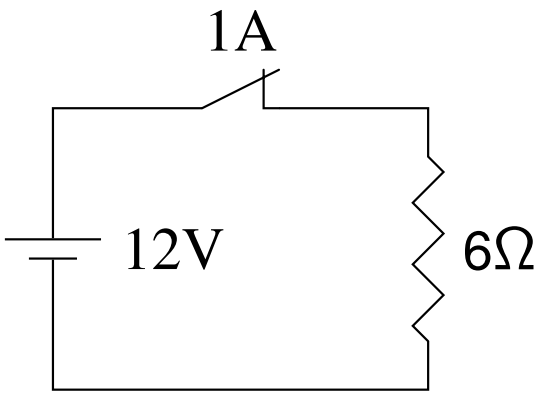
Adaptation of time

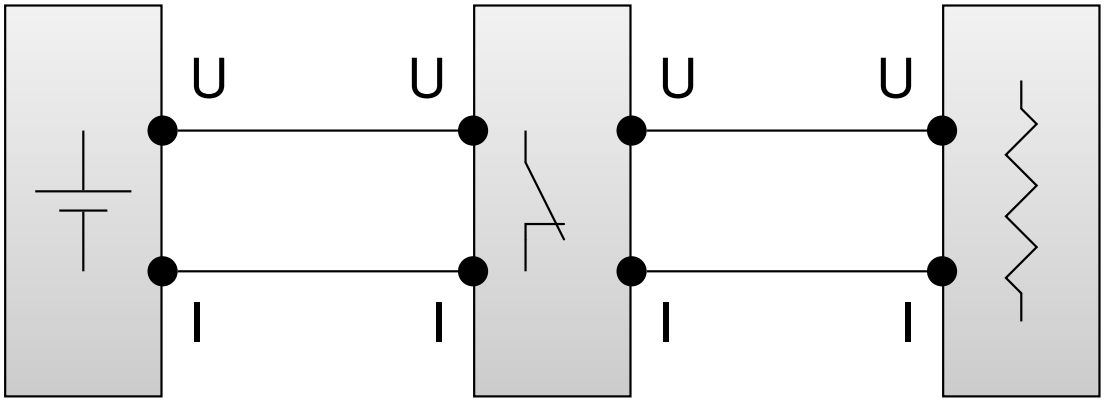
- interface blocks can set tag relations between clocks

Execution management

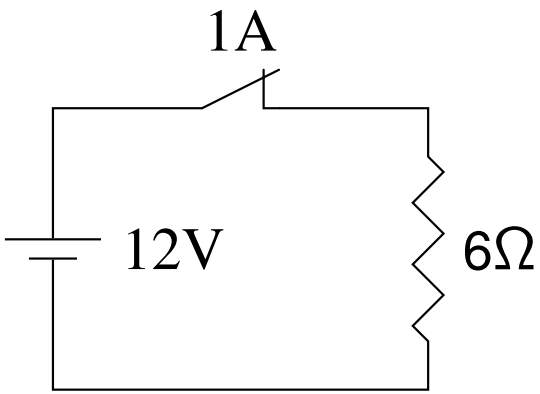
- the observation clock of the top level model implies the snapshot clock
- driving clocks have ticks generated by the execution platform
- tag relations are used to synchronize the simulation time with the time of the execution platform

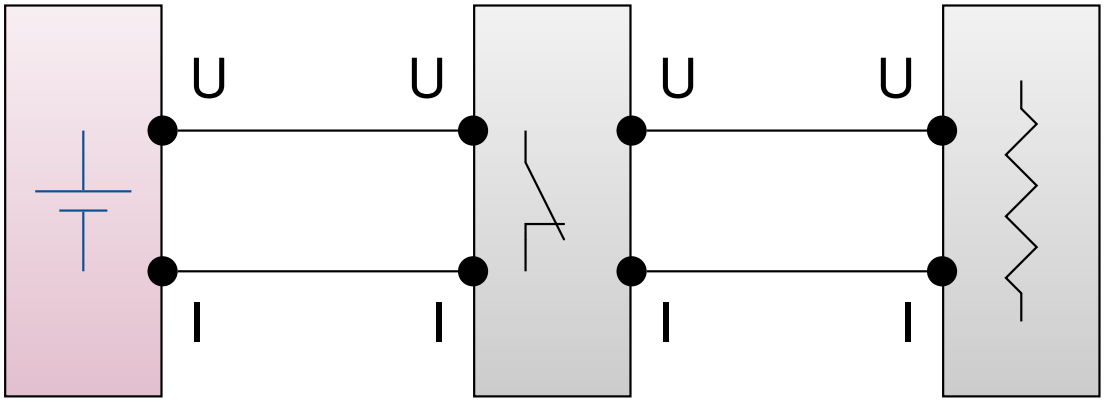
Snapshot with Non Validation





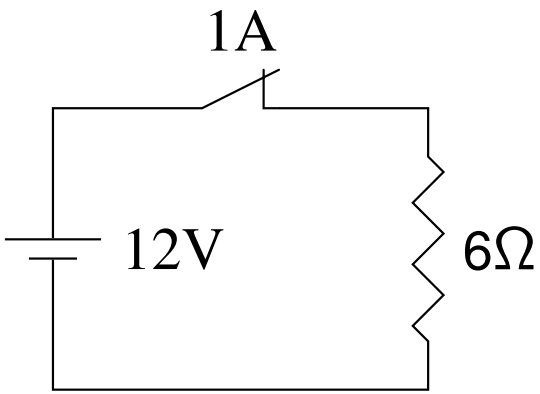
Snapshot with Non Validation

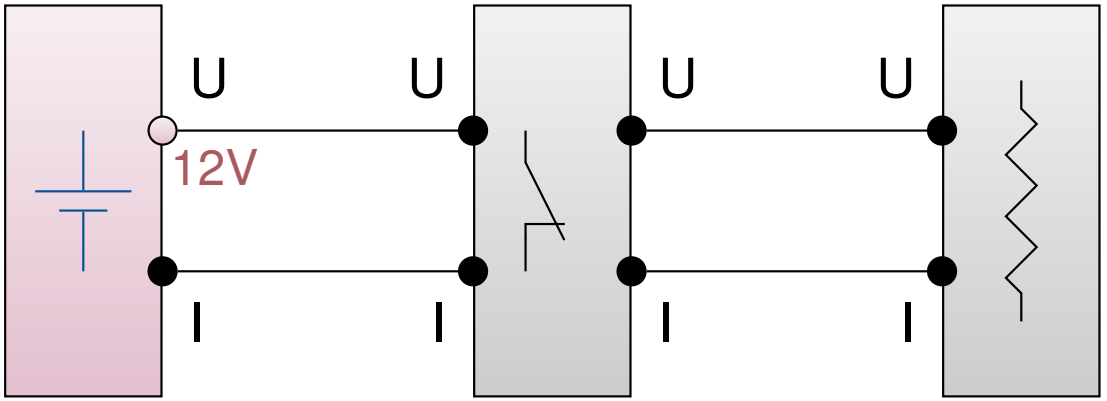




Schedule → battery

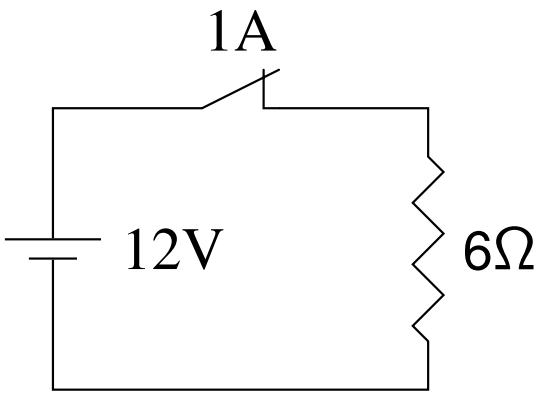
Snapshot with Non Validation

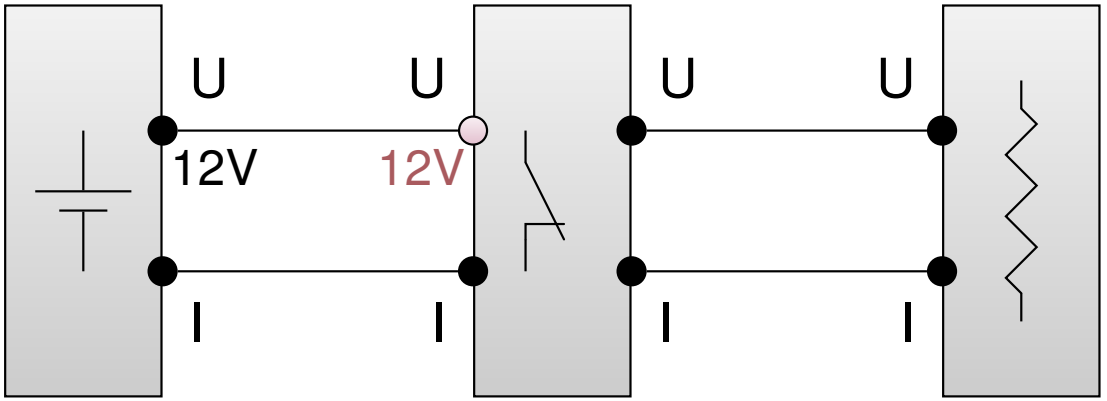




Update battery

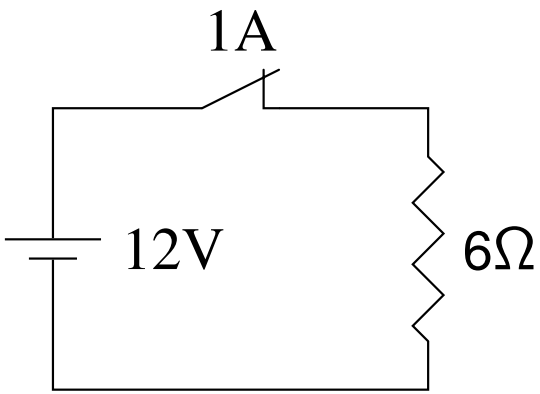
Snapshot with Non Validation

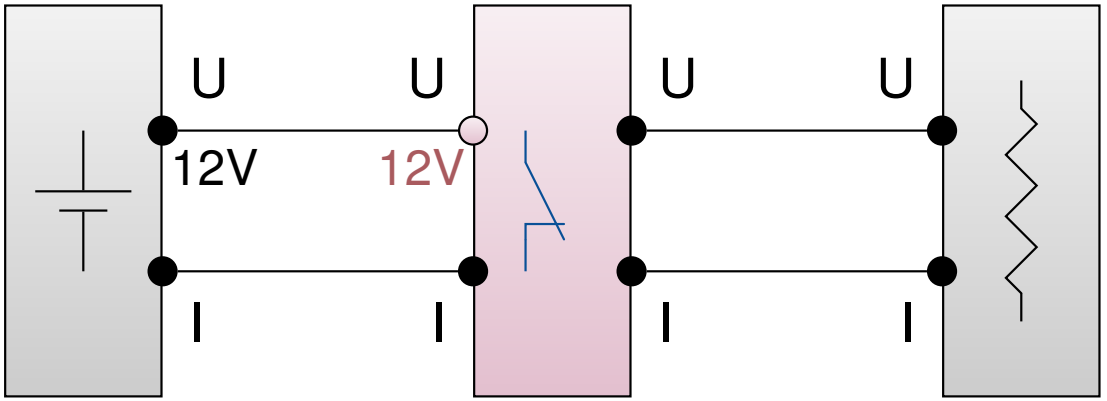




Propagate

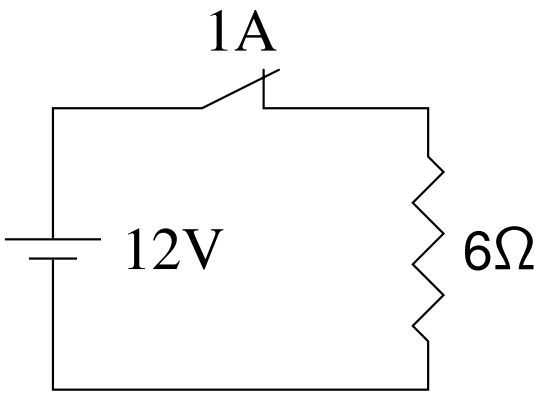
Snapshot with Non Validation

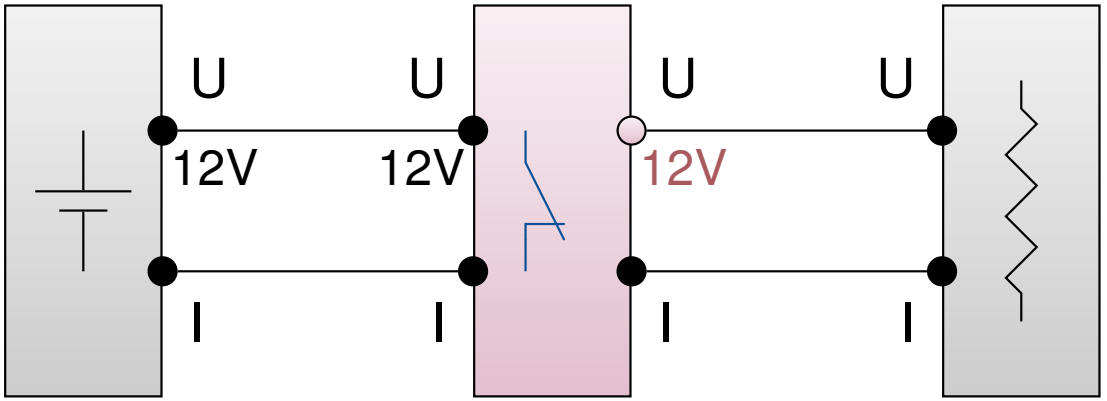




Schedule → fuse

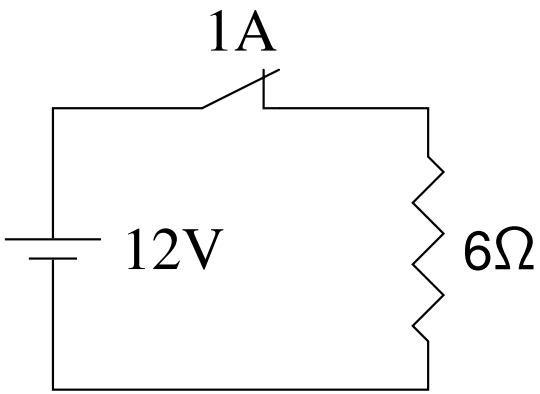
Snapshot with Non Validation

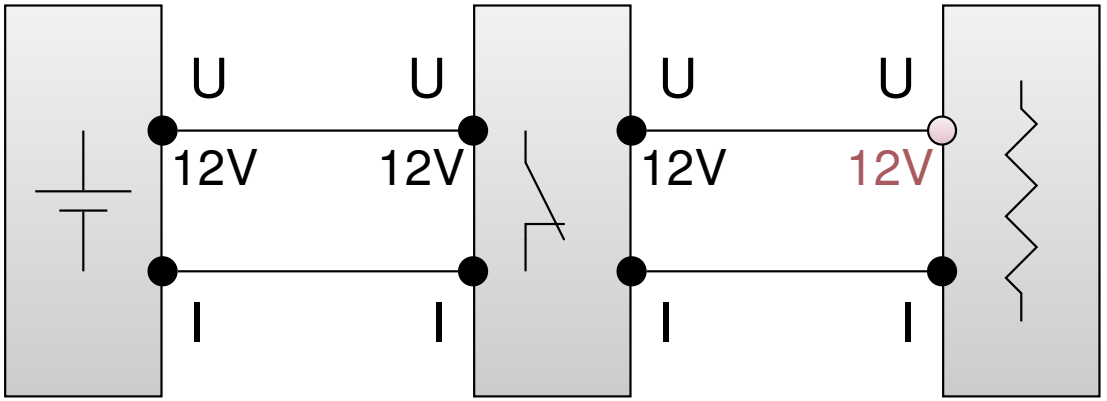




Update fuse

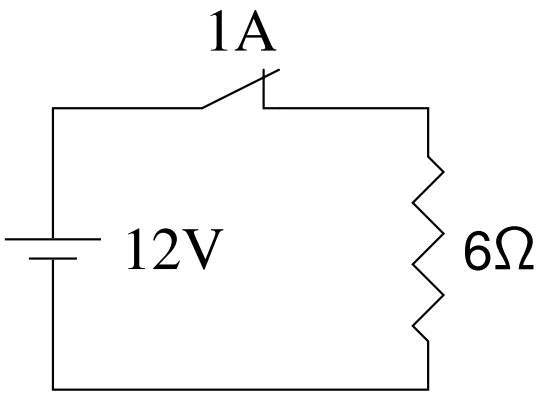
Snapshot with Non Validation

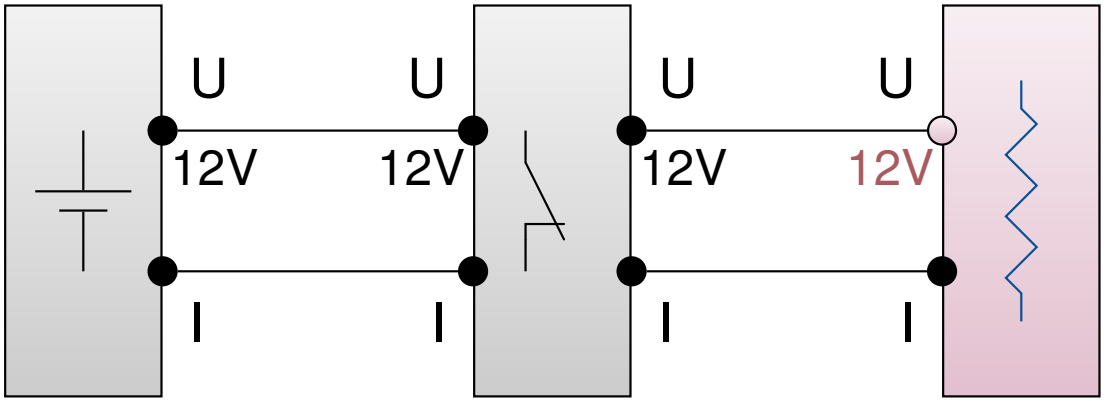




Propagate

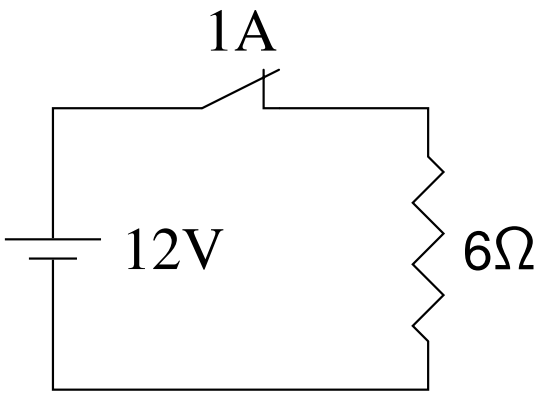
Snapshot with Non Validation

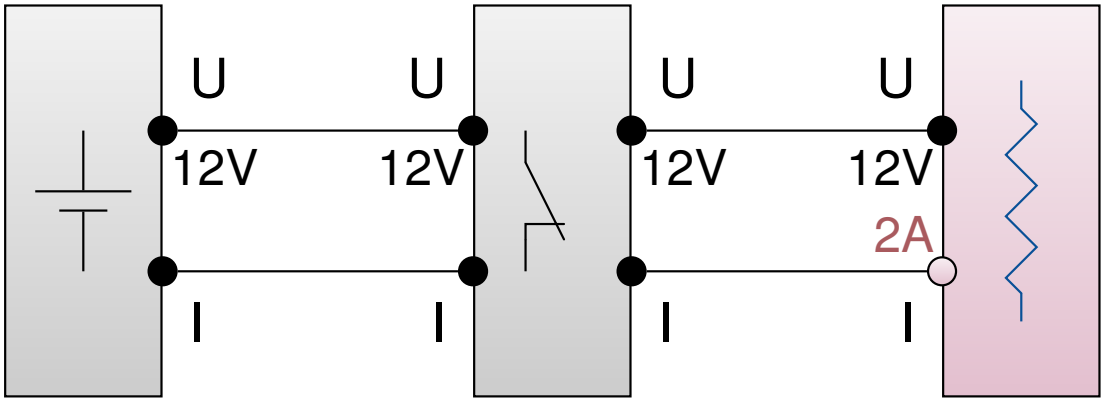




Schedule → resistor

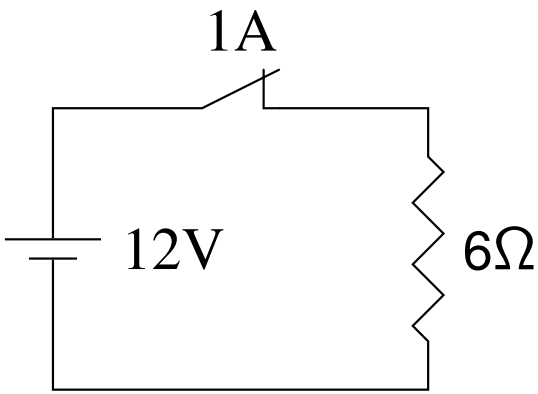
Snapshot with Non Validation

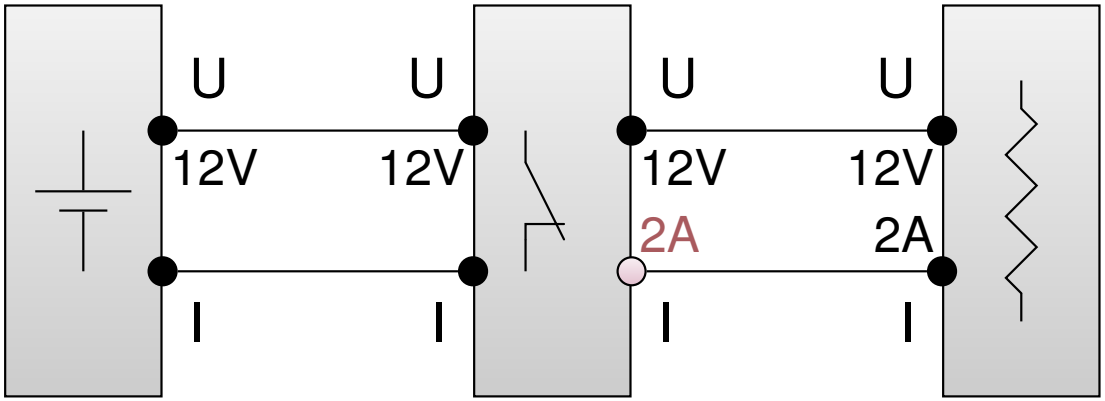




Update resistor

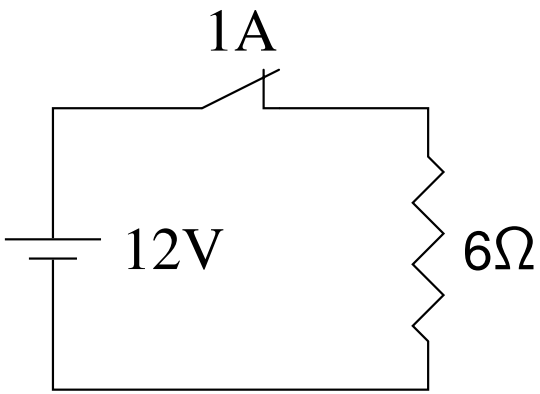
Snapshot with Non Validation

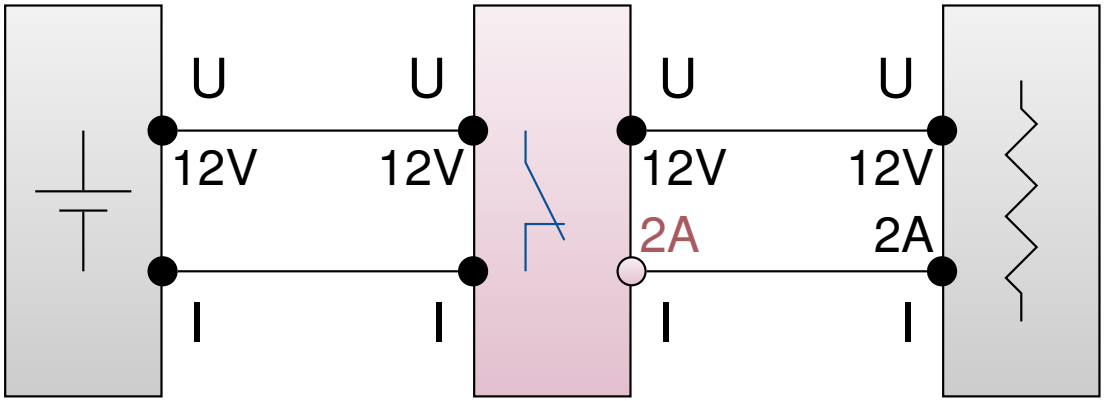




Propagate

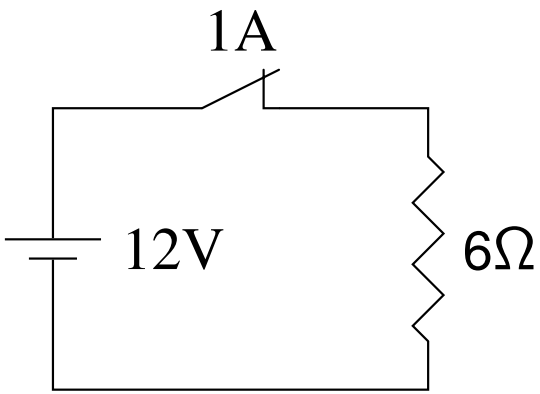
Snapshot with Non Validation

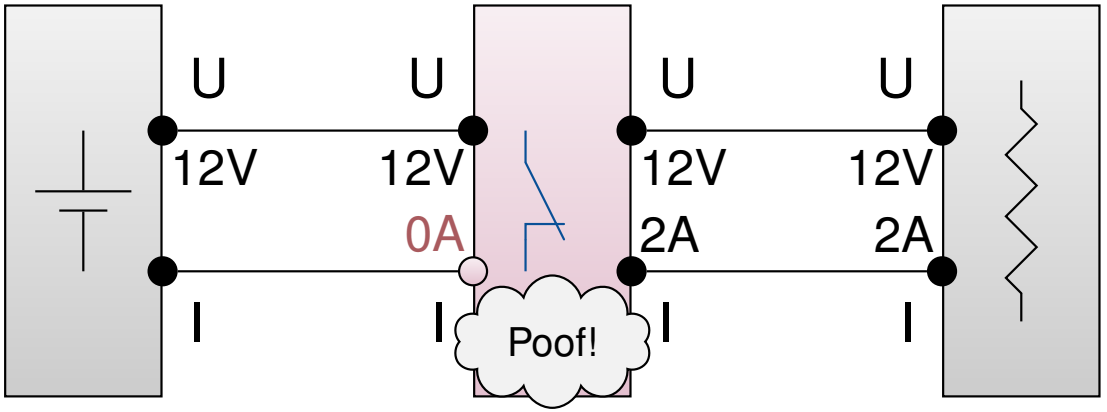




Schedule → fuse

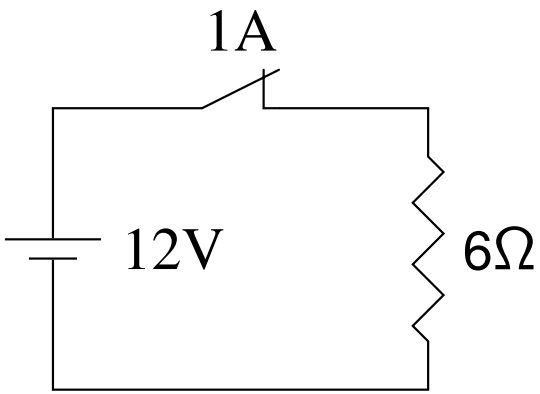
Snapshot with Non Validation

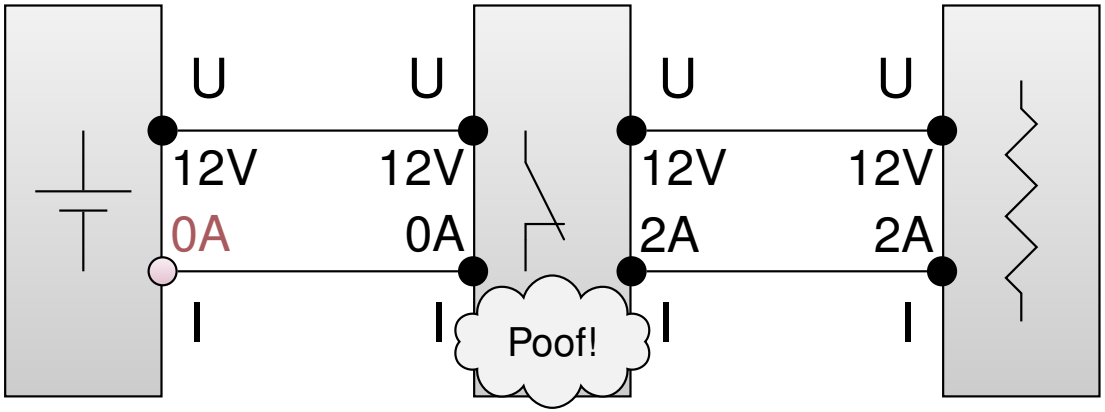




Update fuse

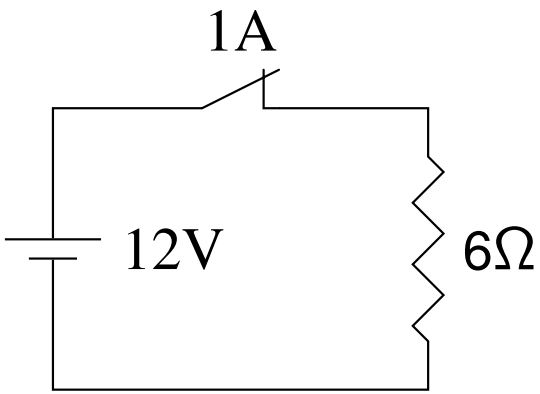
Snapshot with Non Validation

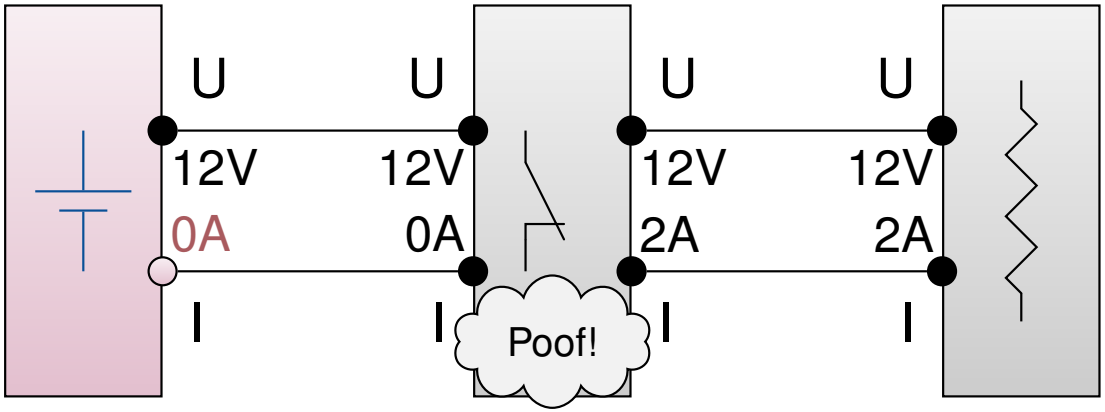




Propagate

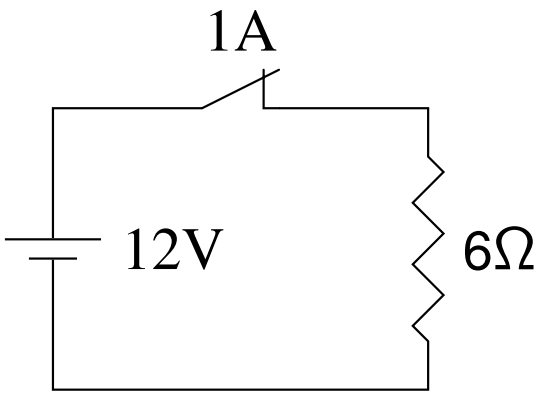
Snapshot with Non Validation

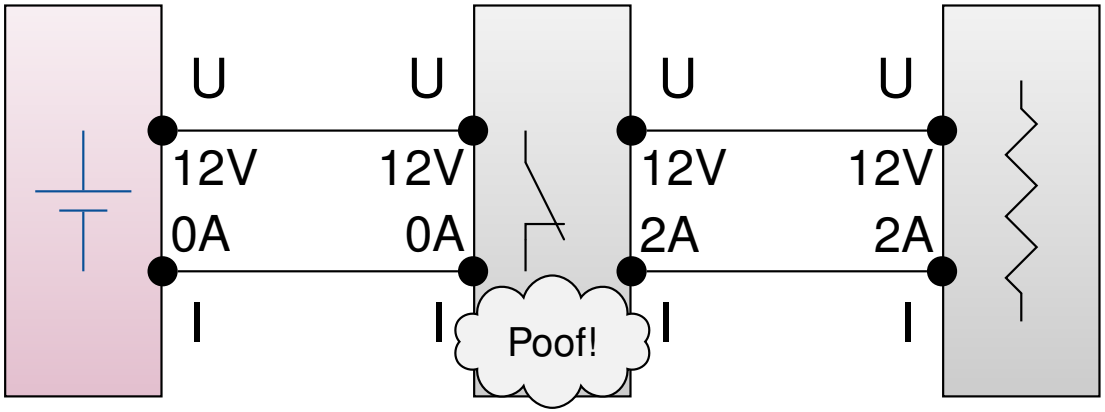




Schedule → battery

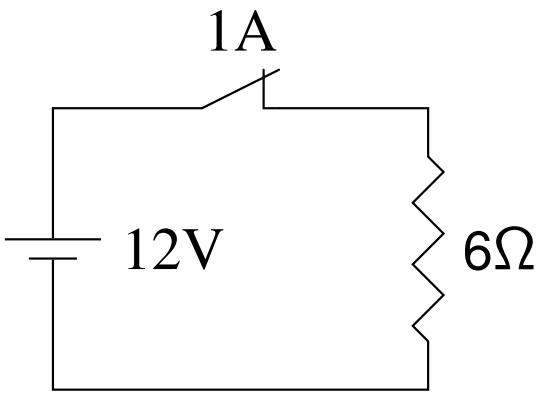
Snapshot with Non Validation

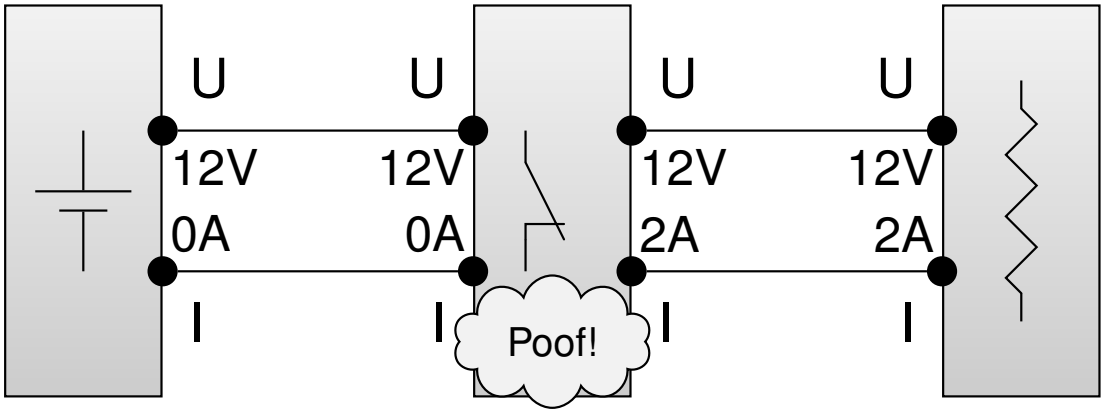




Update battery

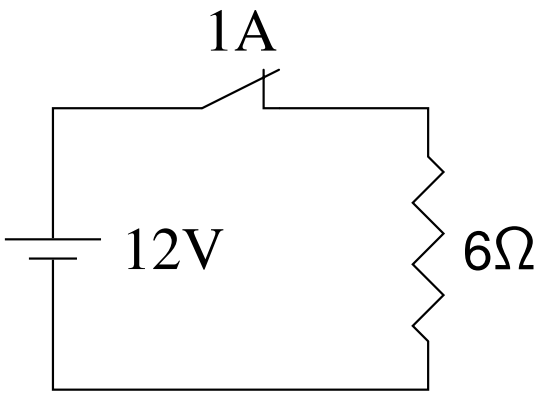
Snapshot with Non Validation

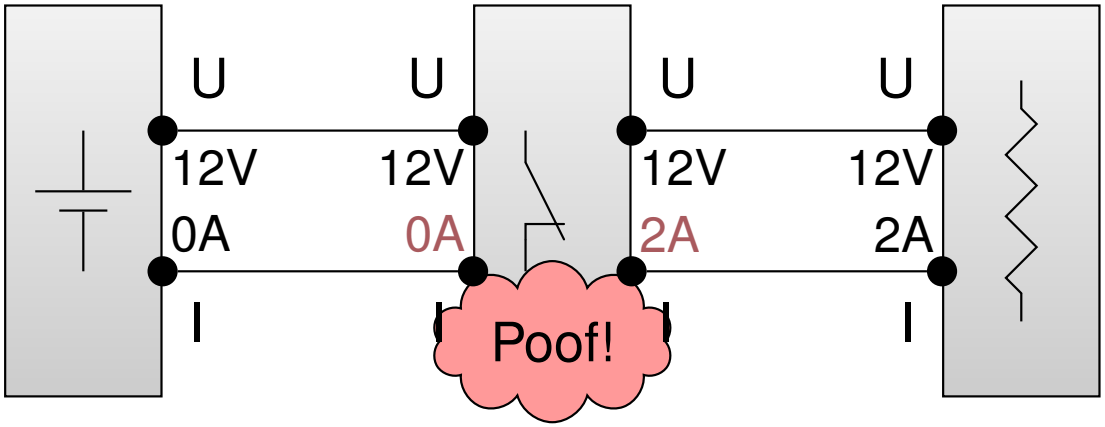




Done

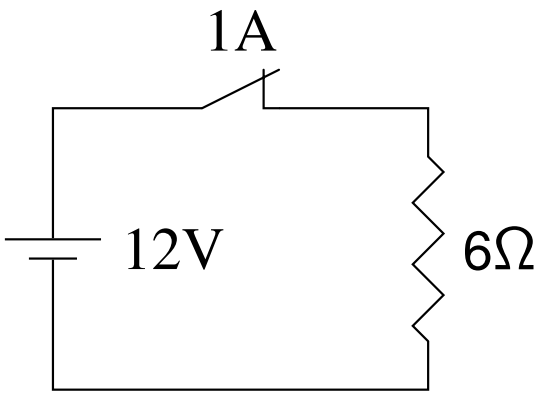
Snapshot with Non Validation

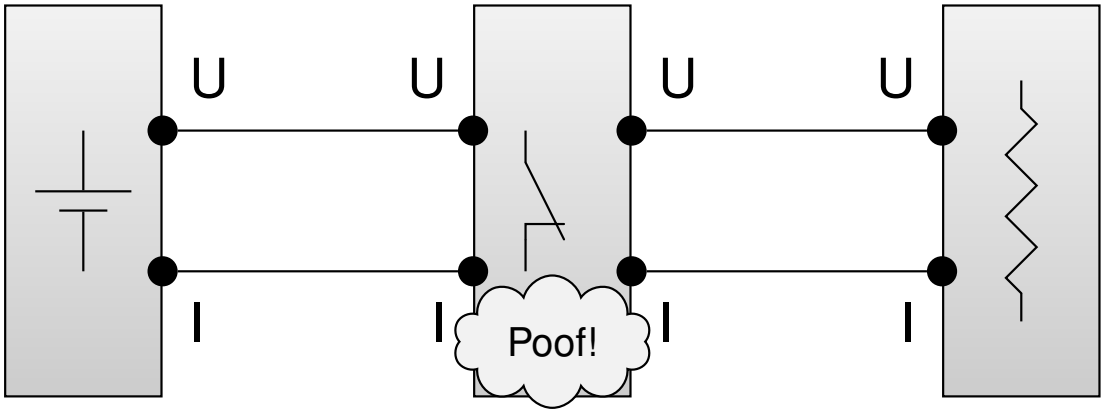




Validate

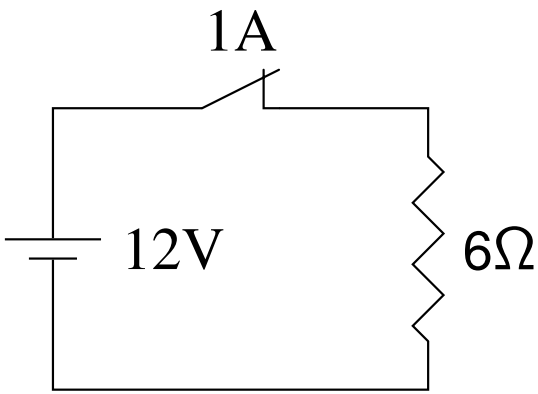
Snapshot with Non Validation

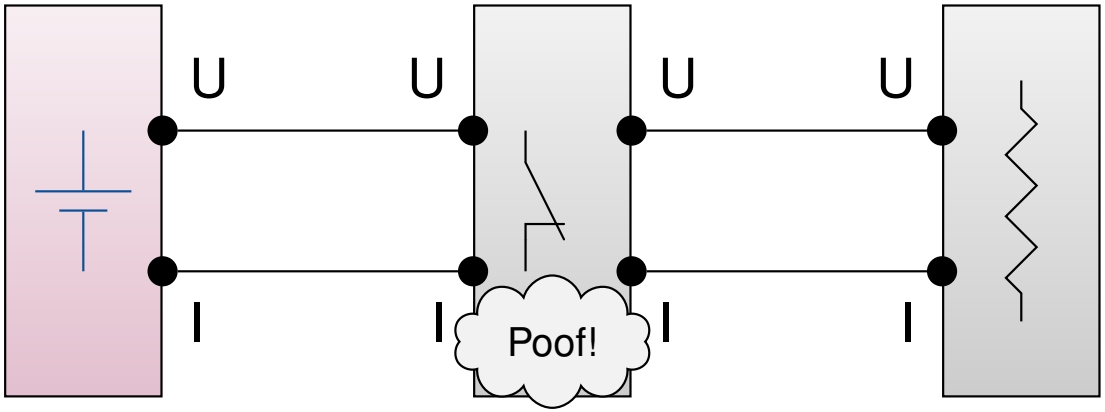




Reset

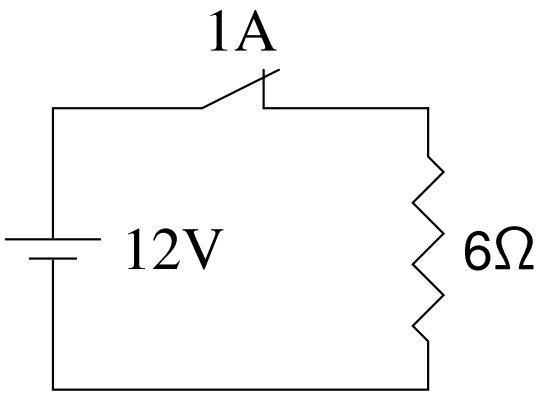
Snapshot with Non Validation

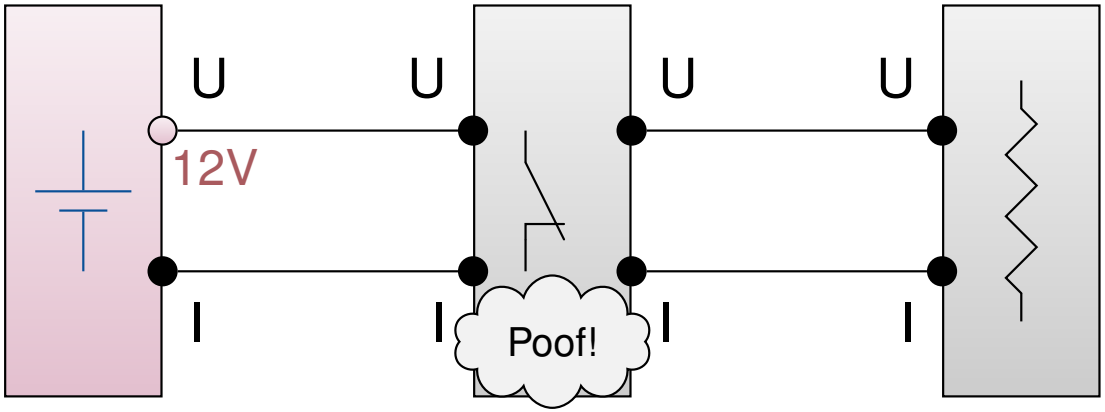




Schedule → battery

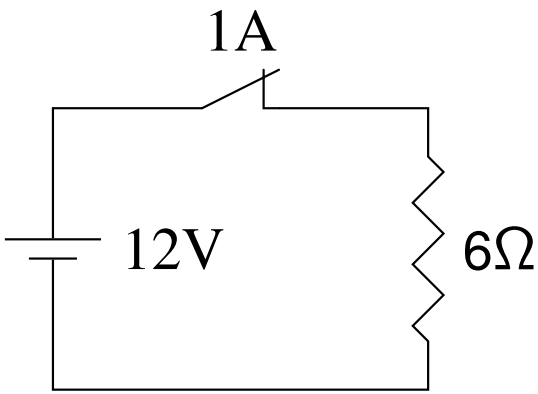
Snapshot with Non Validation

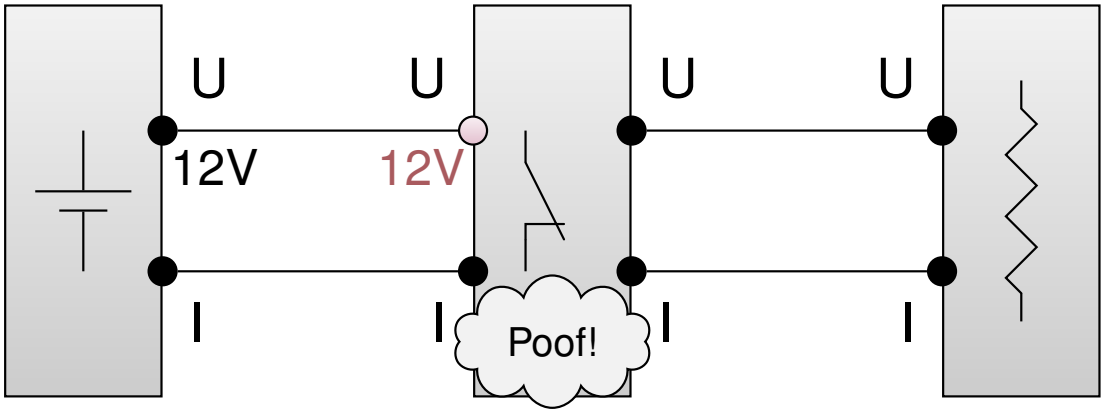




Update battery

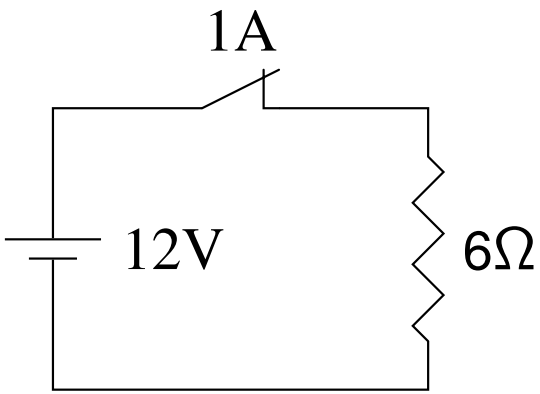
Snapshot with Non Validation

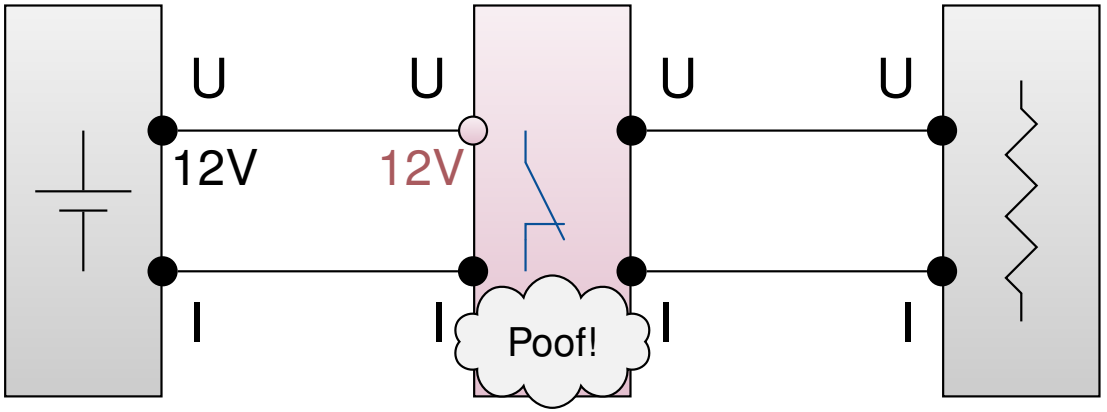




Propagate

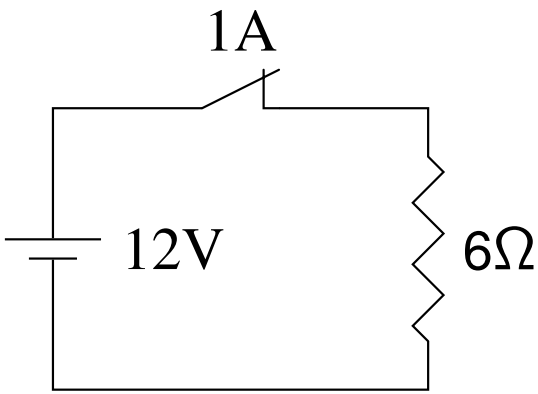
Snapshot with Non Validation

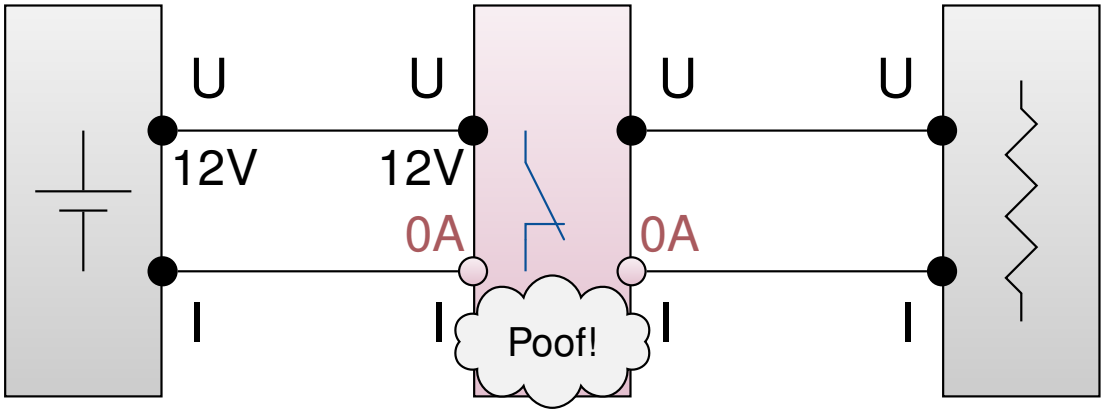




Schedule → fuse

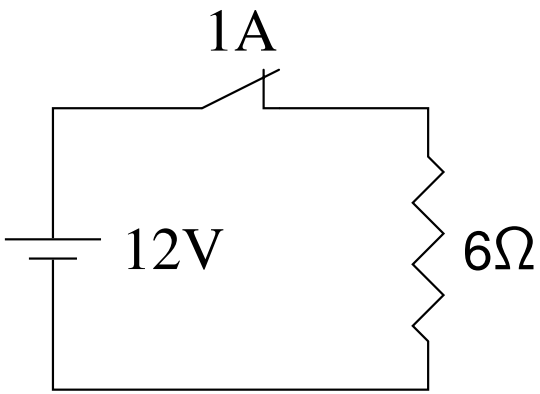
Snapshot with Non Validation

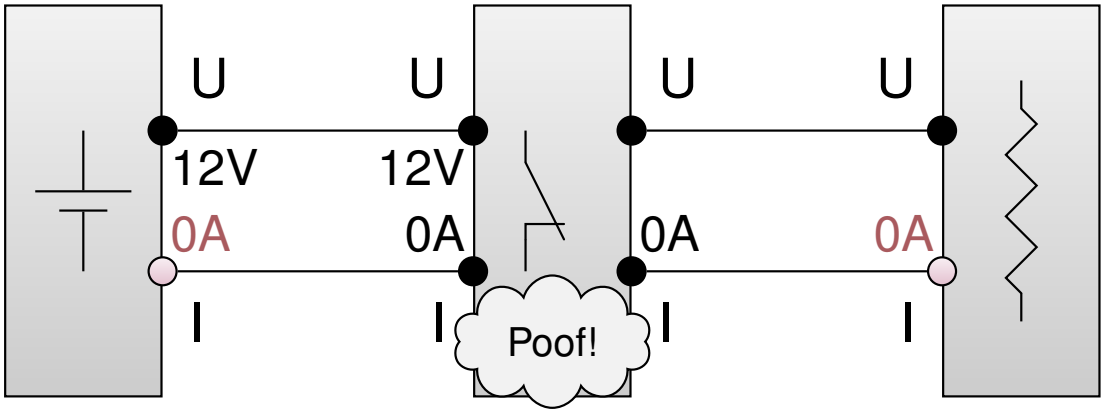




Update fuse

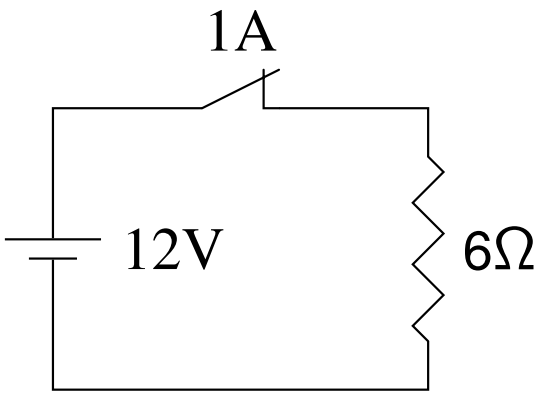
Snapshot with Non Validation

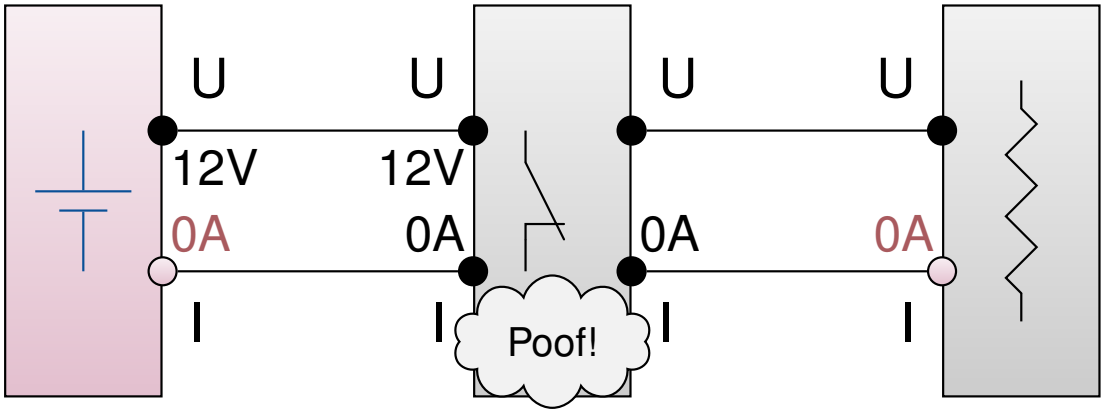




Propagate

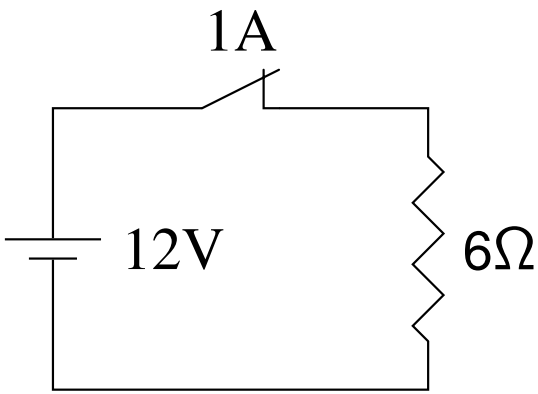
Snapshot with Non Validation

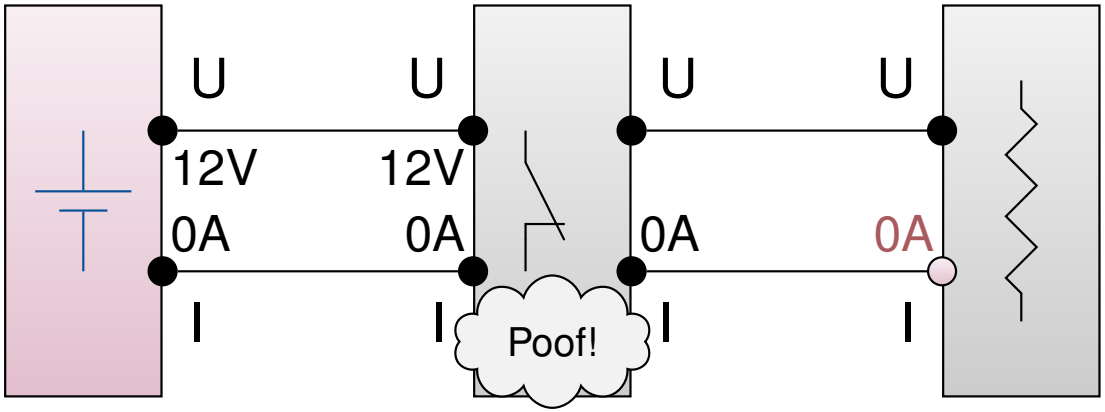




Schedule → battery

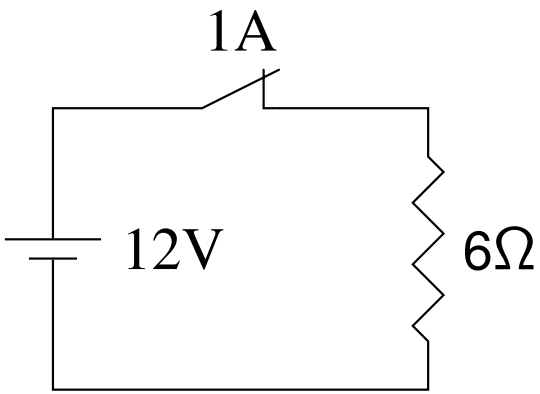
Snapshot with Non Validation

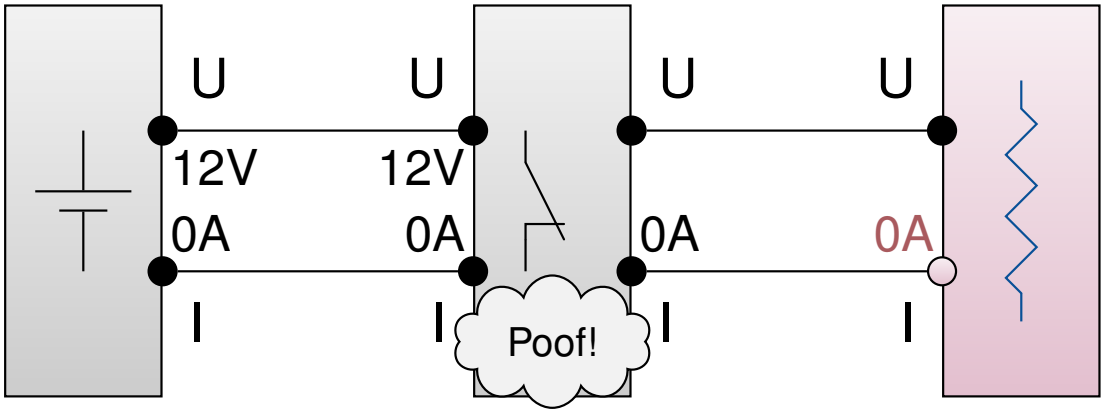




Update battery

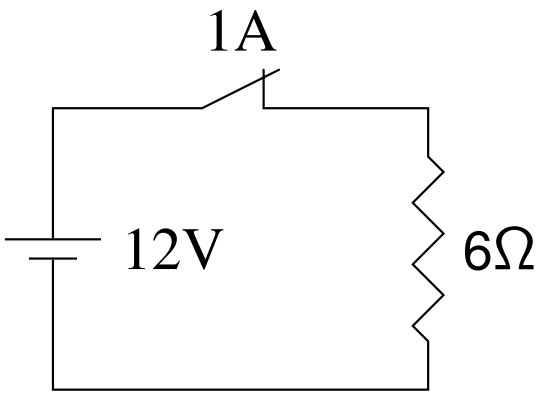
Snapshot with Non Validation

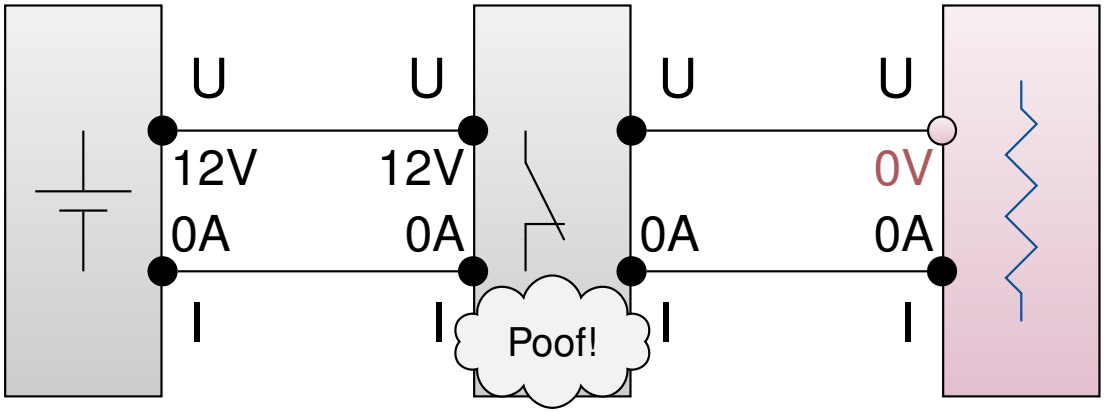




Schedule → resistor

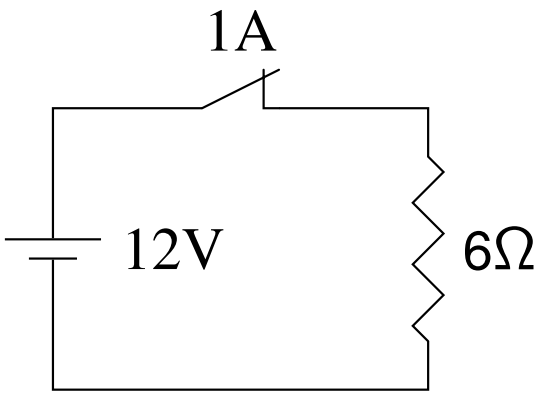
Snapshot with Non Validation

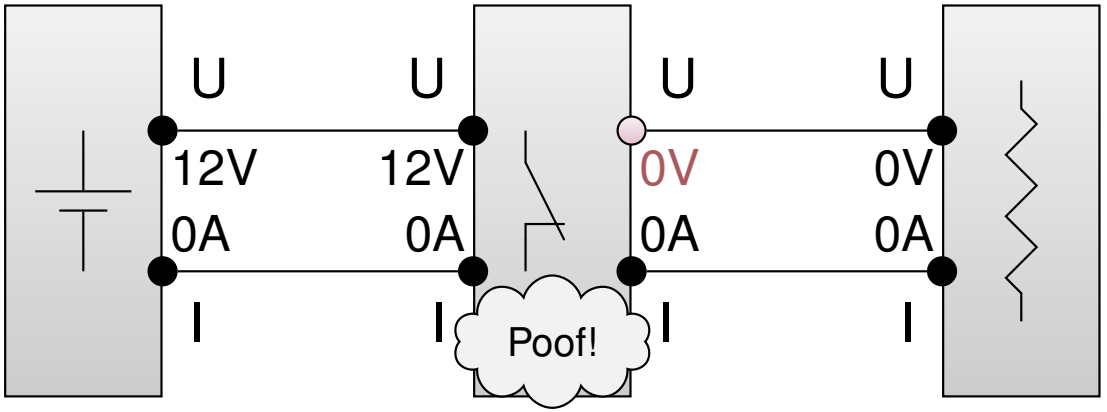




Update resistor

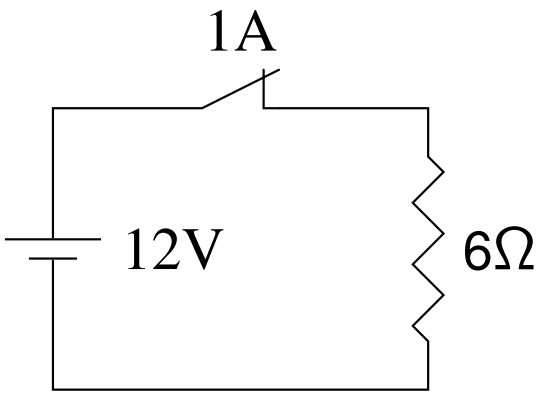
Snapshot with Non Validation

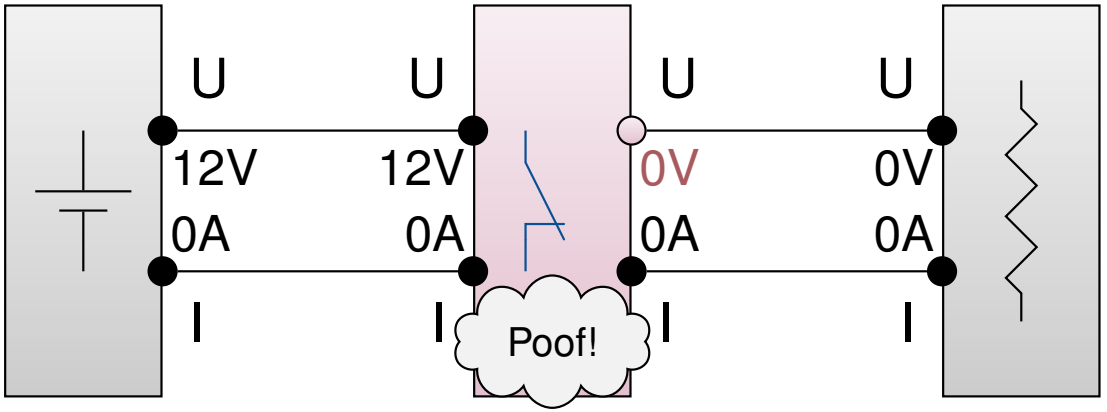




Propagate

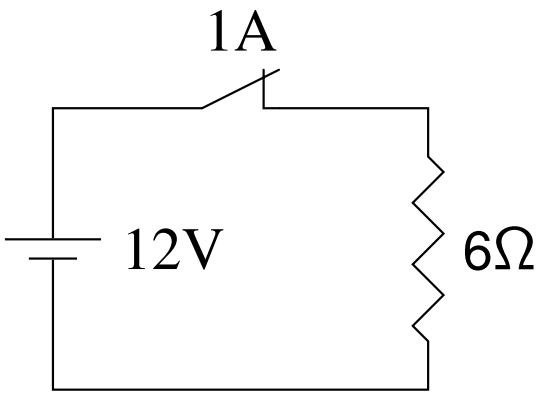
Snapshot with Non Validation

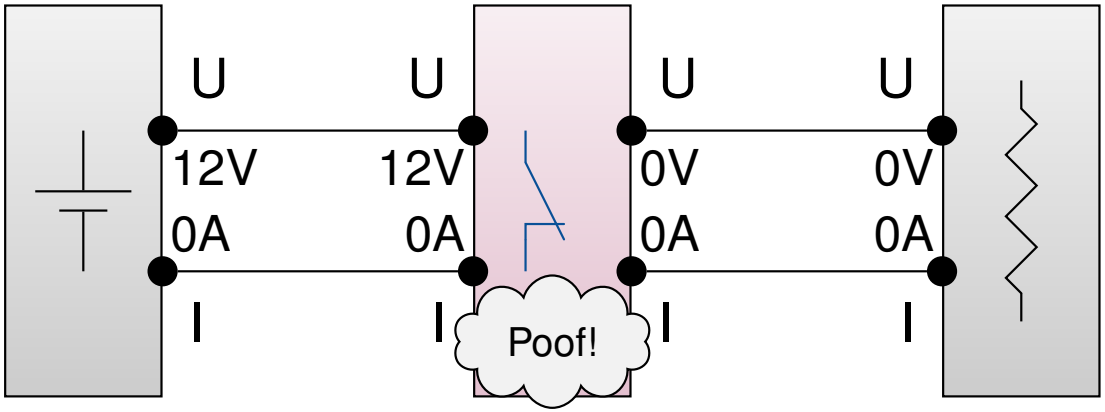




Schedule → fuse

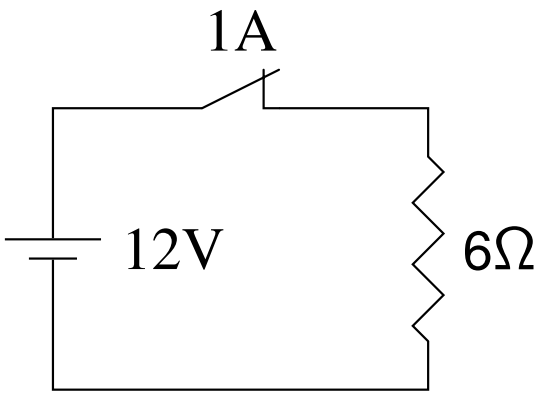
Snapshot with Non Validation

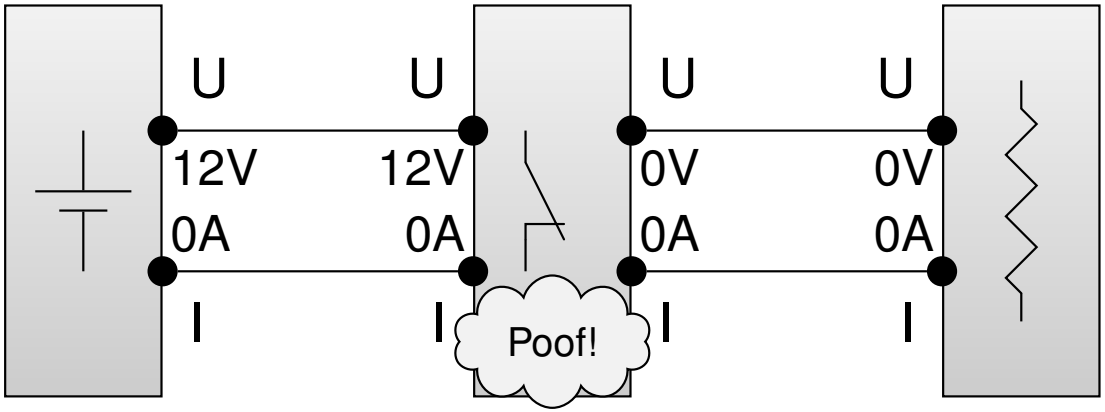




Update fuse

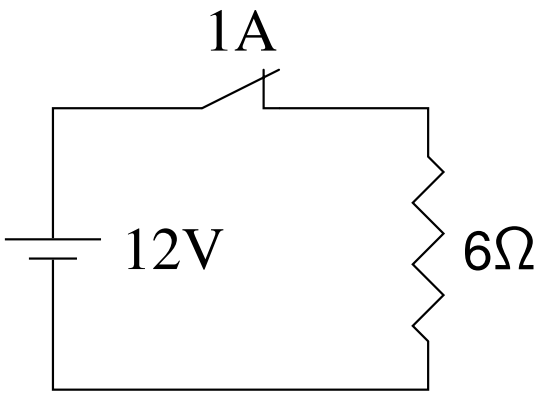
Snapshot with Non Validation

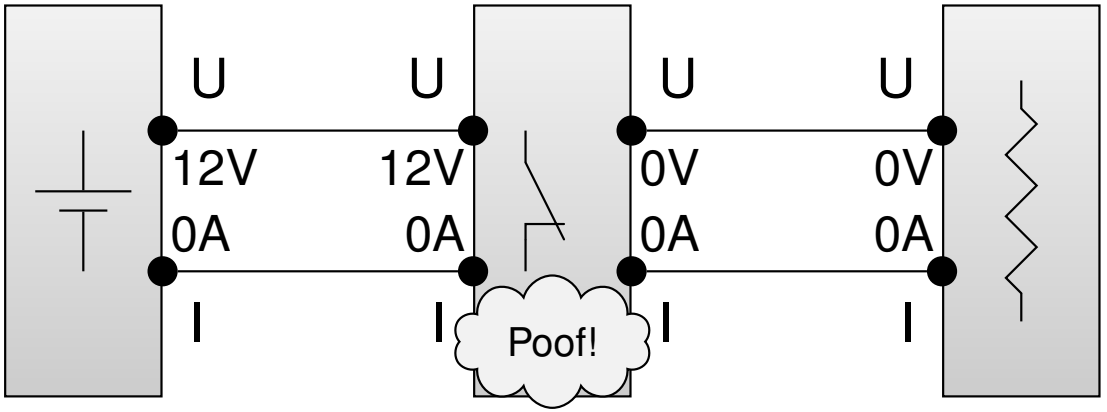




Done

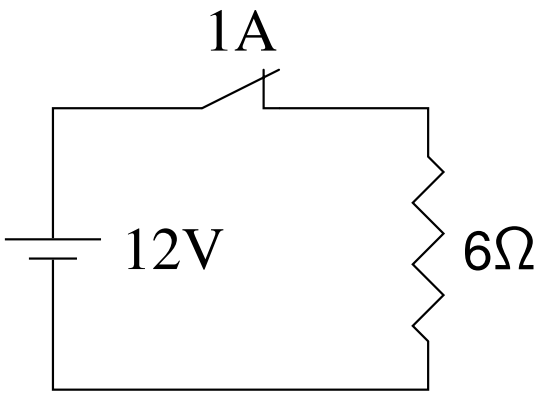
Snapshot with Non Validation

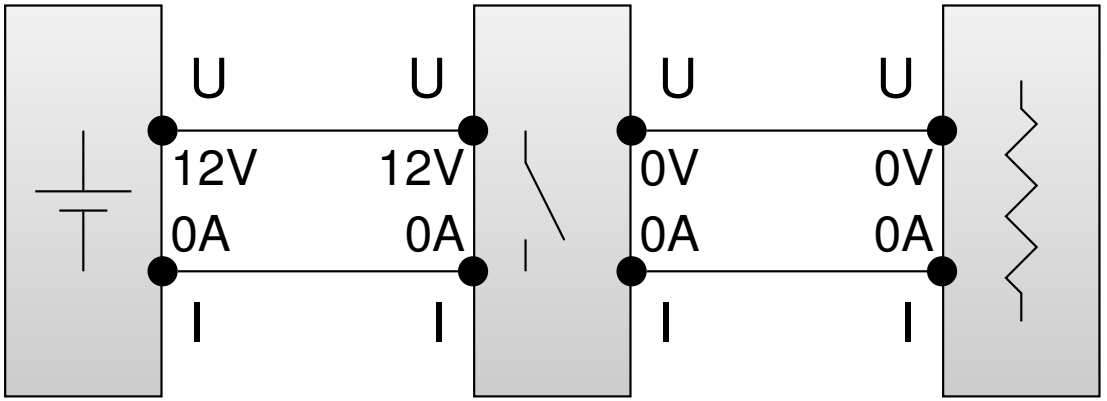




Validate

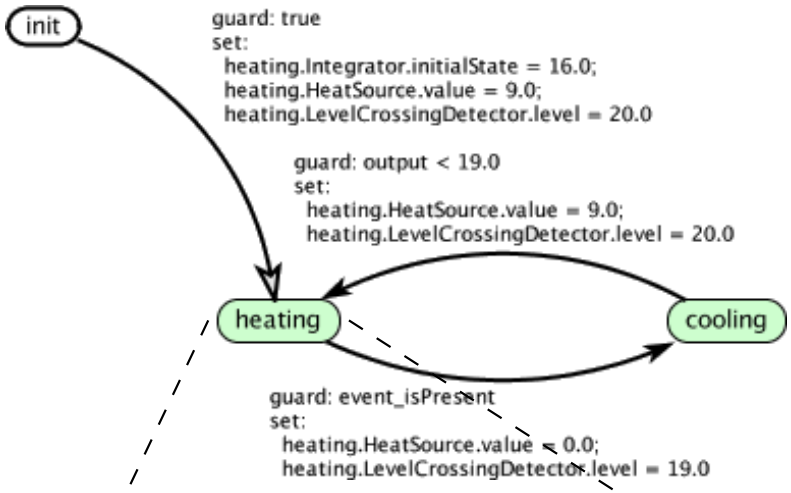
Snapshot with Non Validation

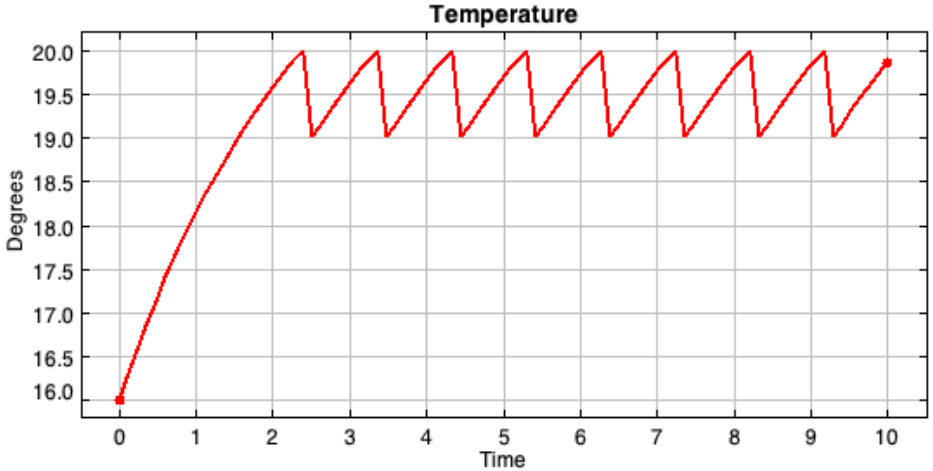


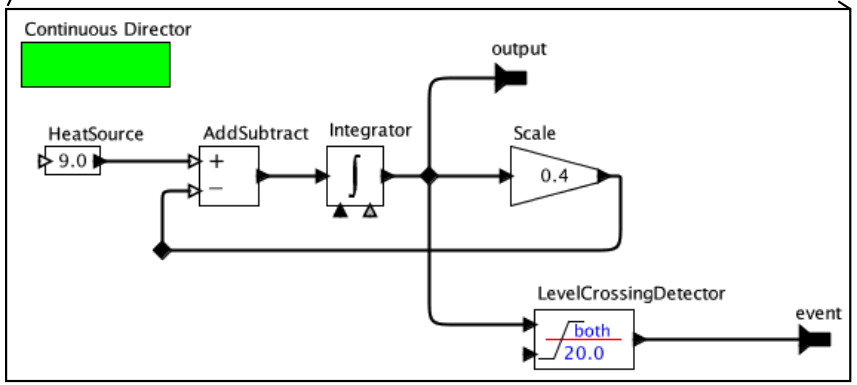


EndOfSnapshot

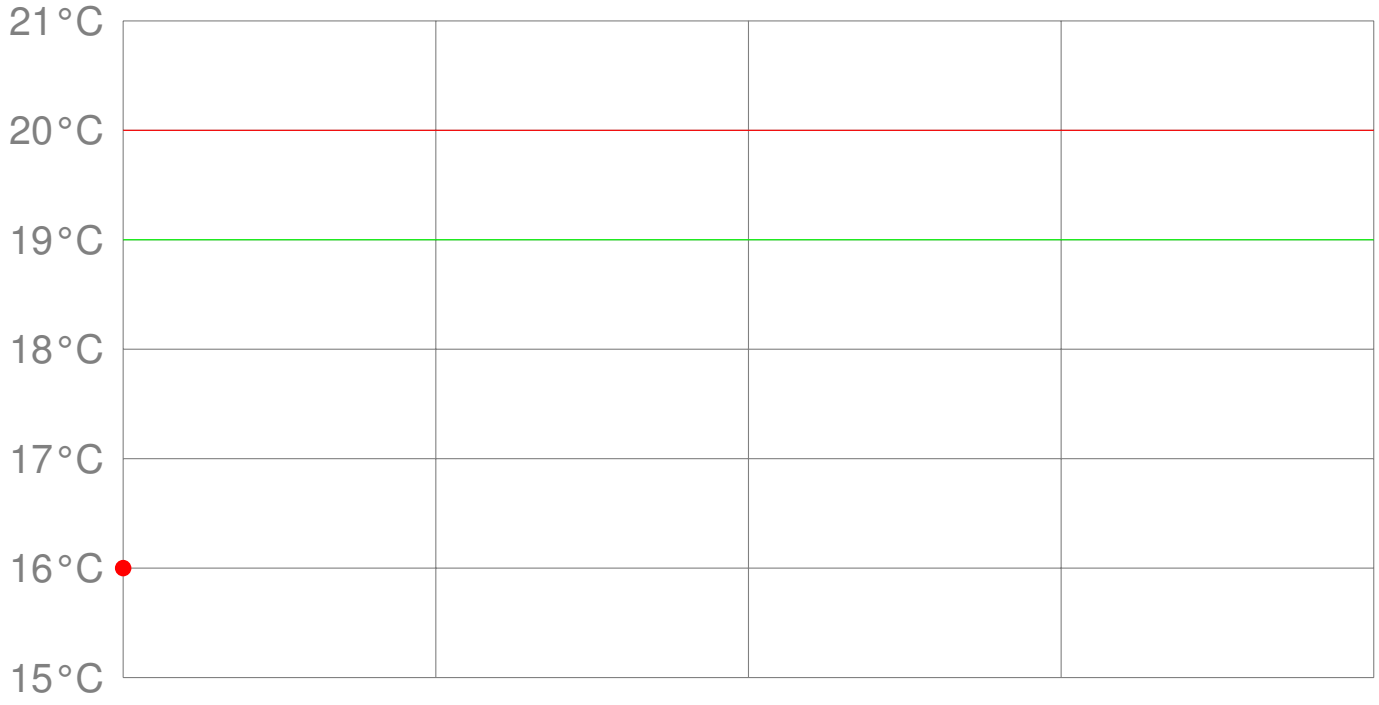
CT Example with Non-Validation





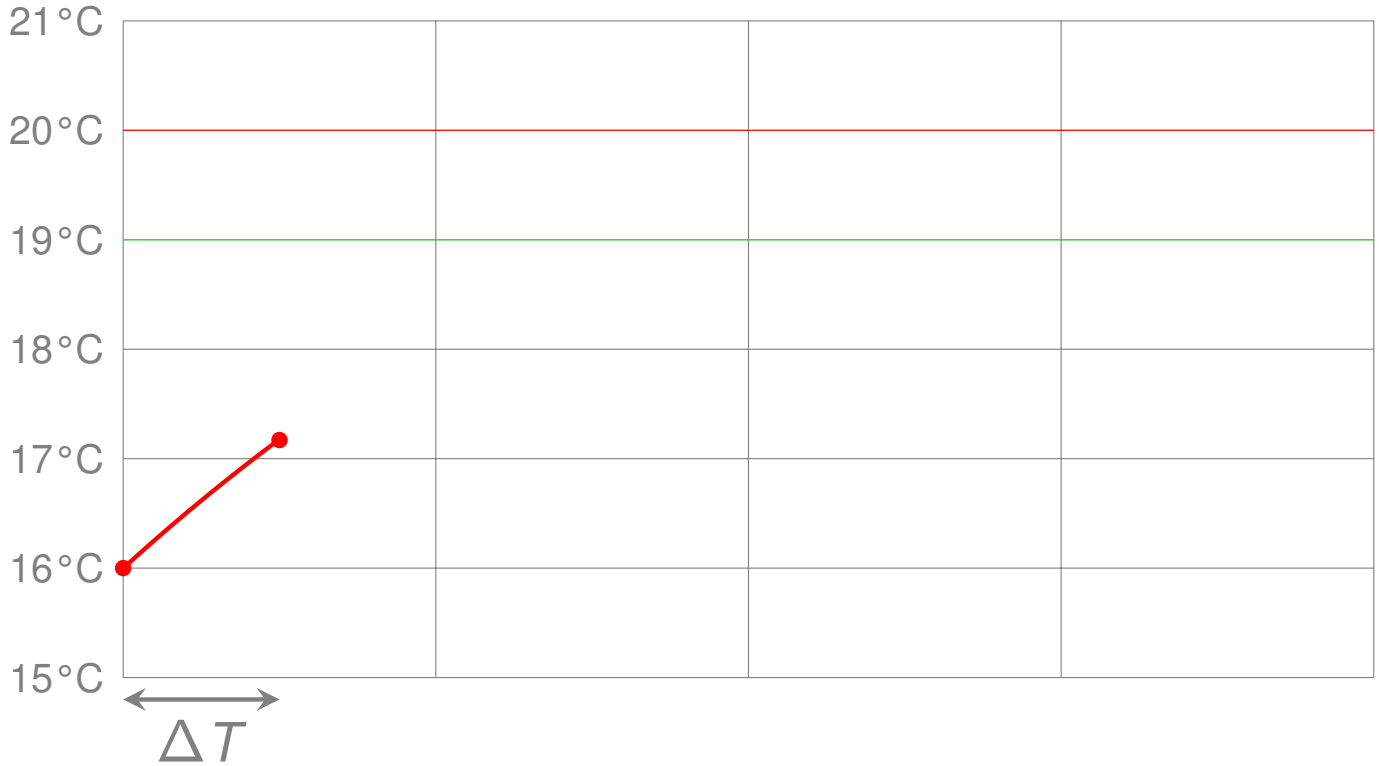


CT Example with Non-Validation



State = heating

CT Example with Non-Validation



State = heating

ΔT computed according to the precision of the CT solver

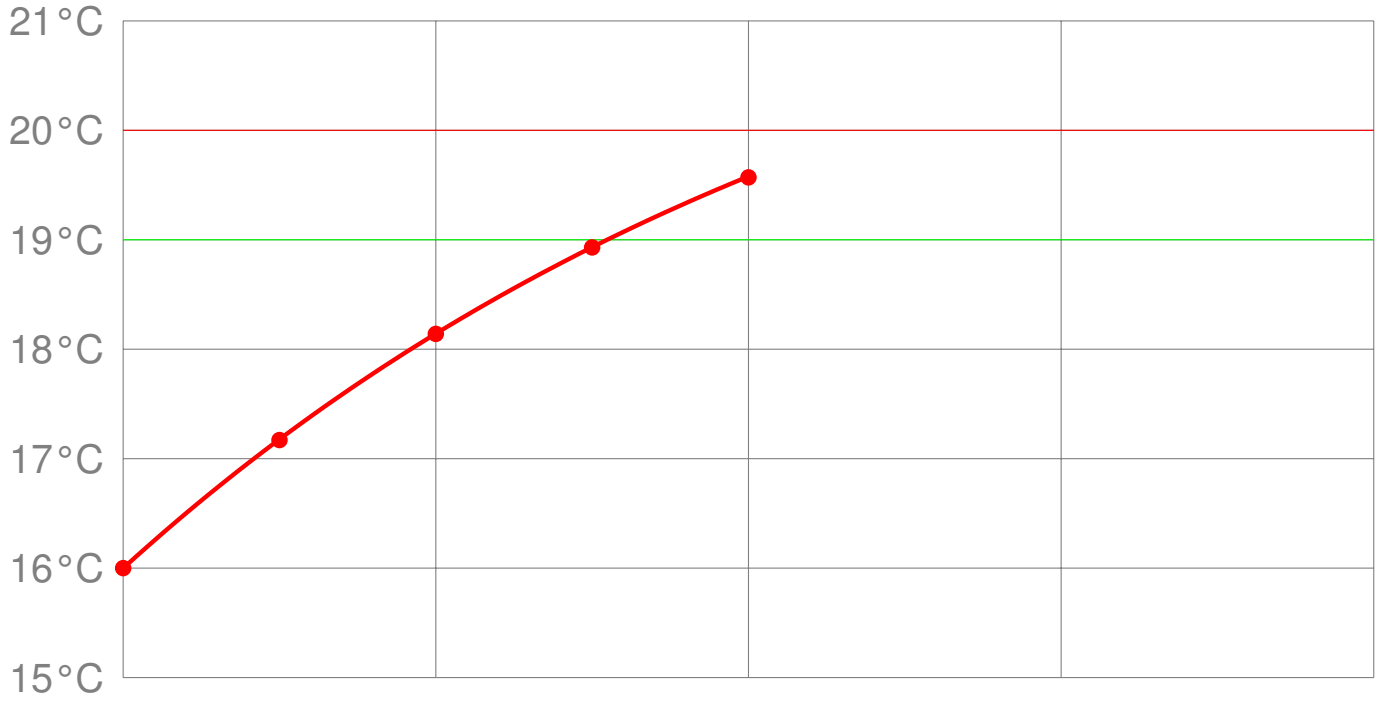
CT Example with Non-Validation

State = heating

CT Example with Non-Validation

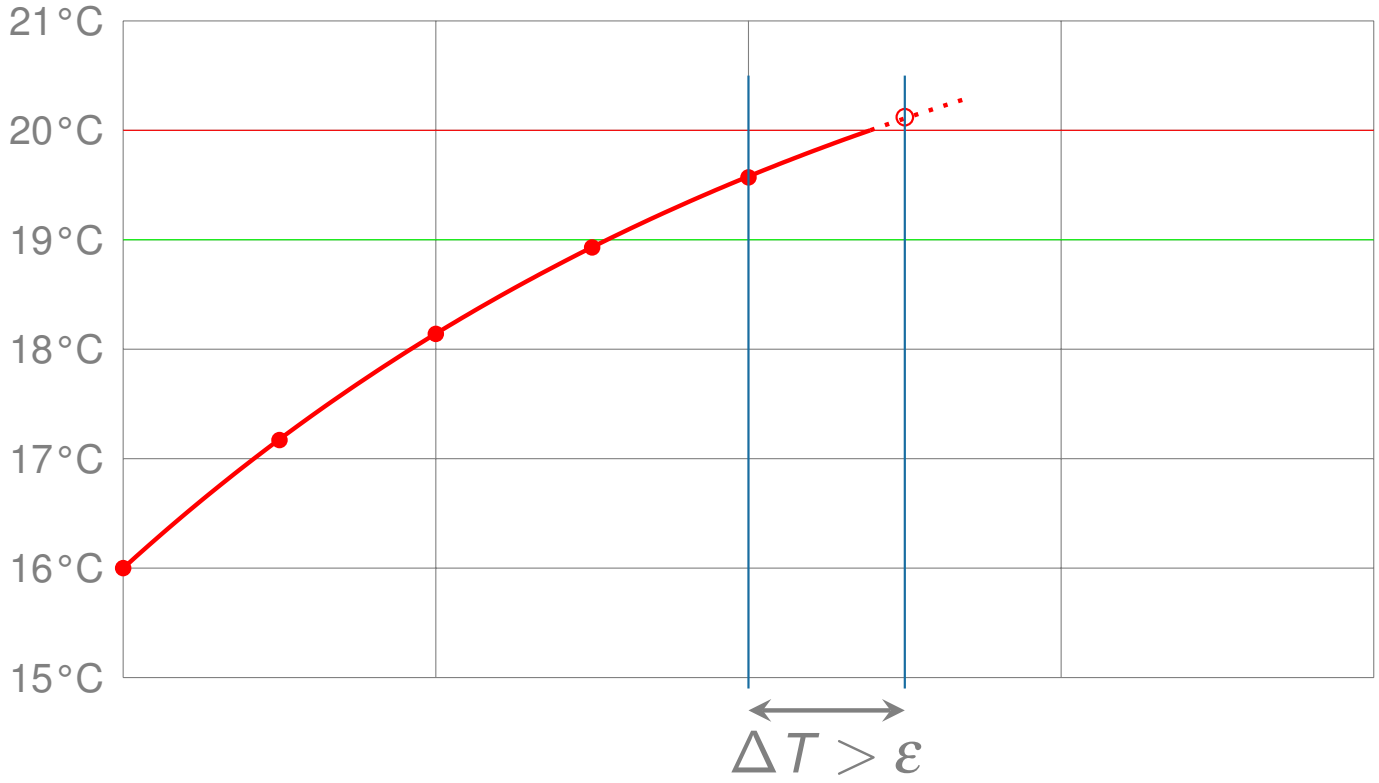
State = heating

CT Example with Non-Validation



State = heating

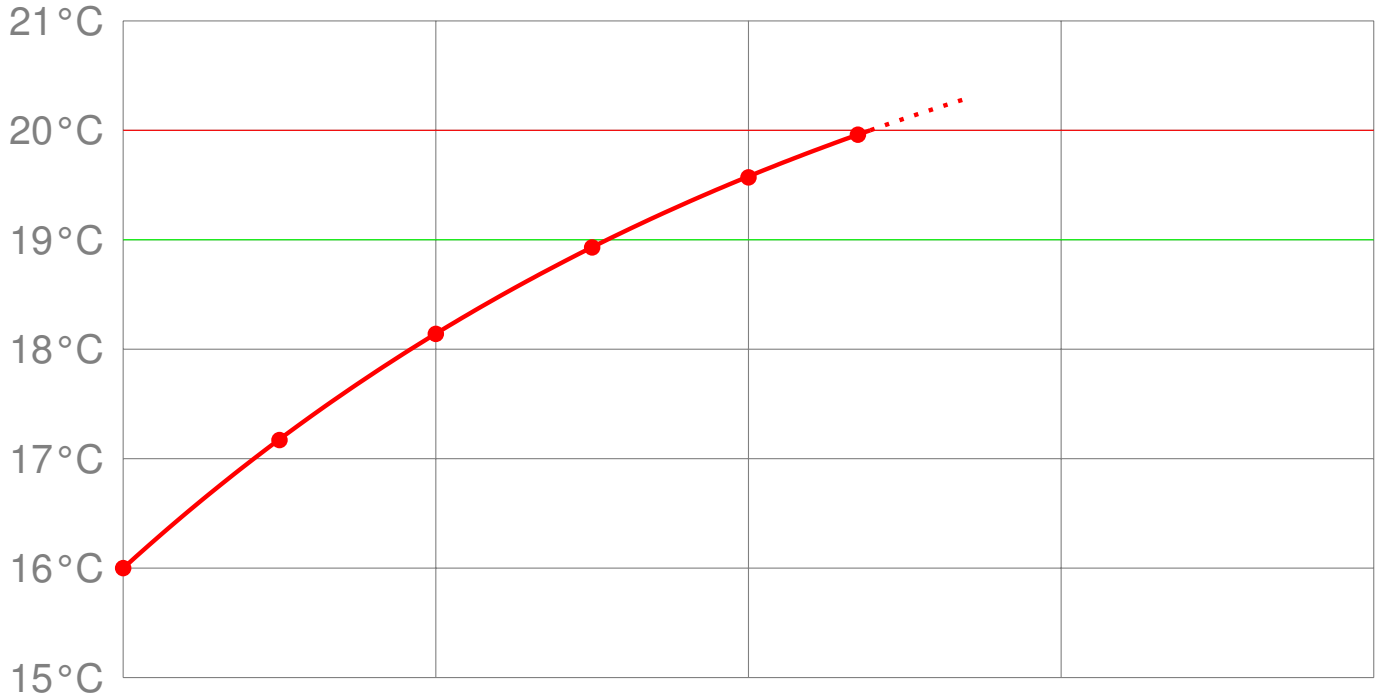
CT Example with Non-Validation



State = heating

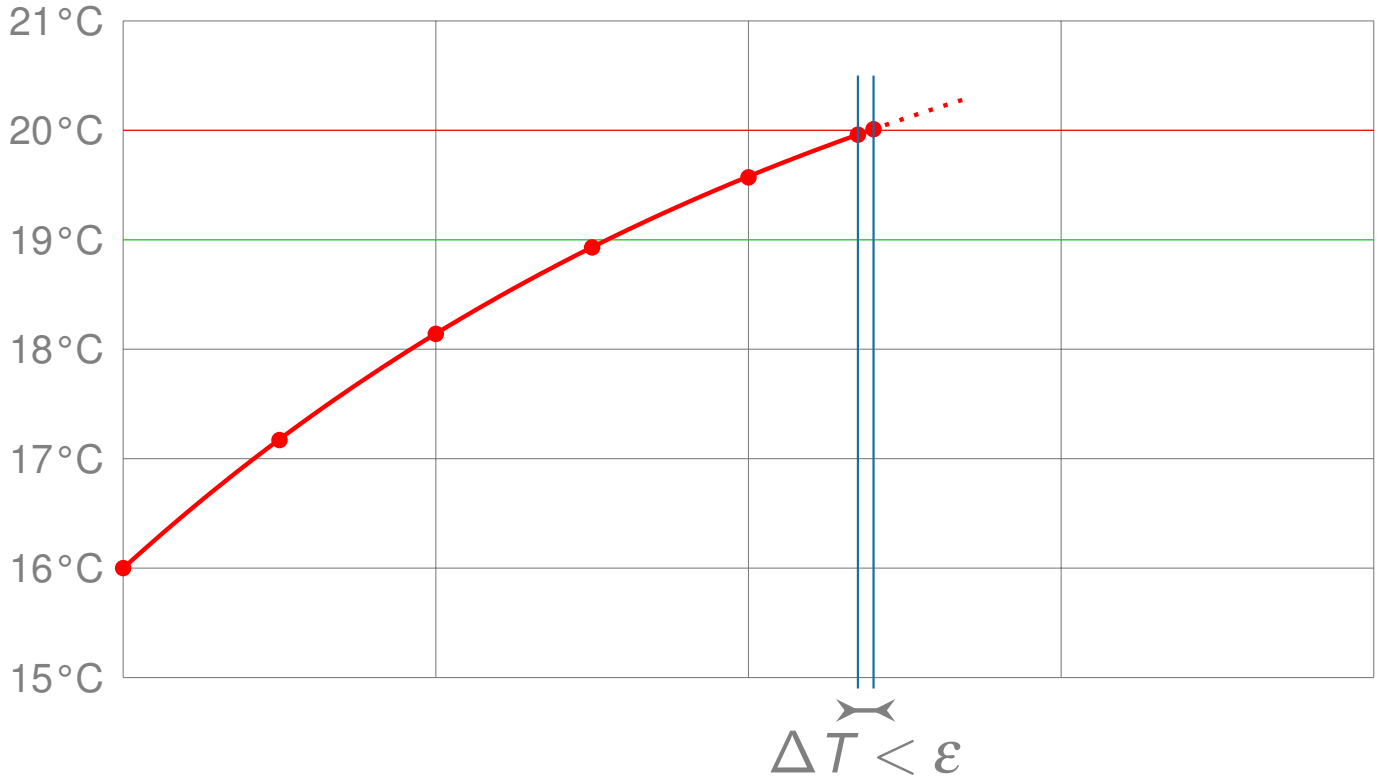
The high threshold is crossed, but we do not know precisely when.
The snapshot is not valid

CT Example with Non-Validation



State = heating

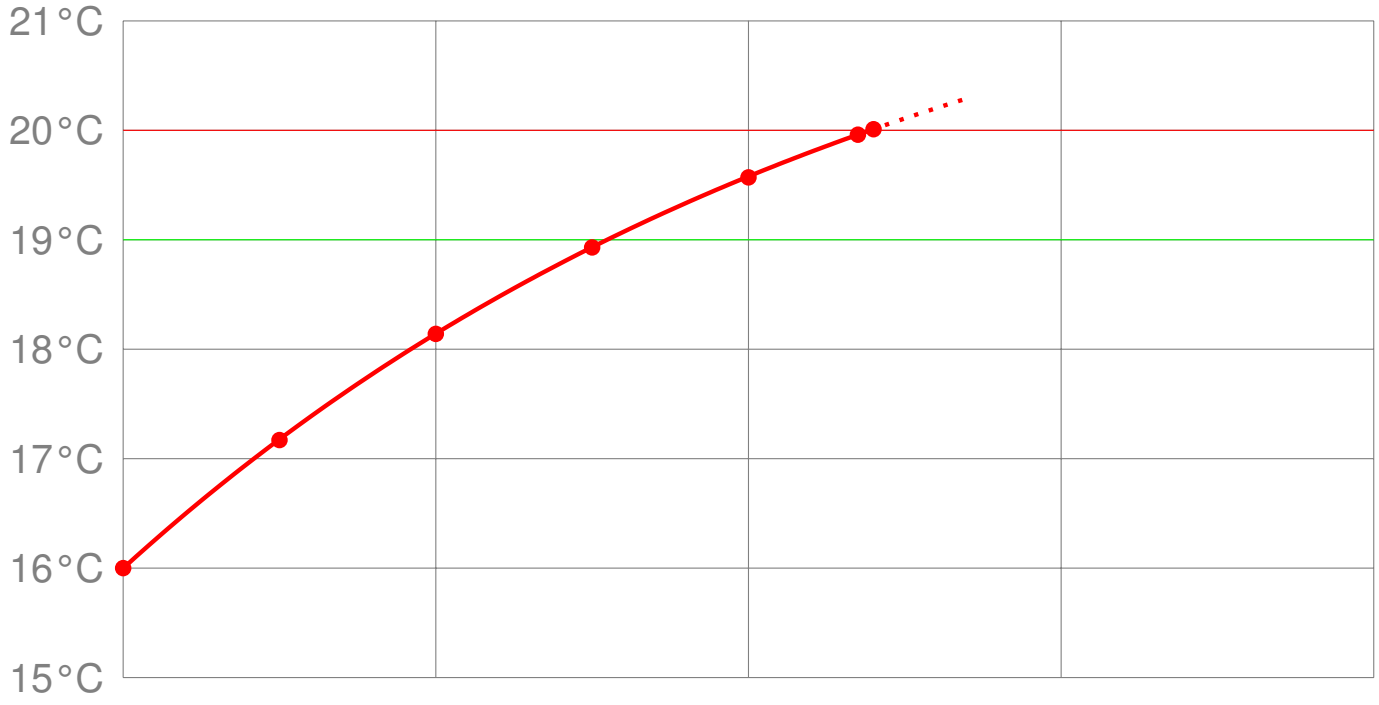
CT Example with Non-Validation



State = heating

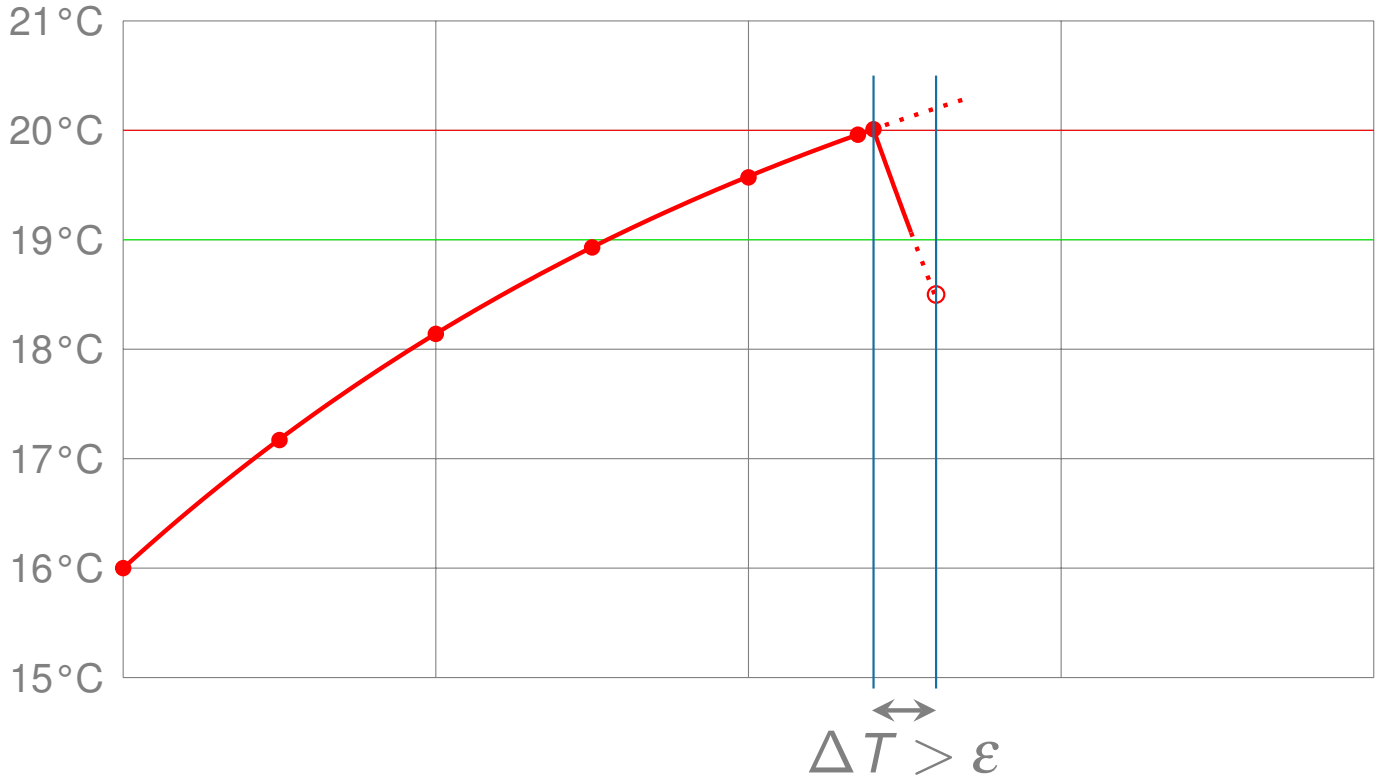
The high threshold is crossed, the date is precise enough.
The state is changed

CT Example with Non-Validation



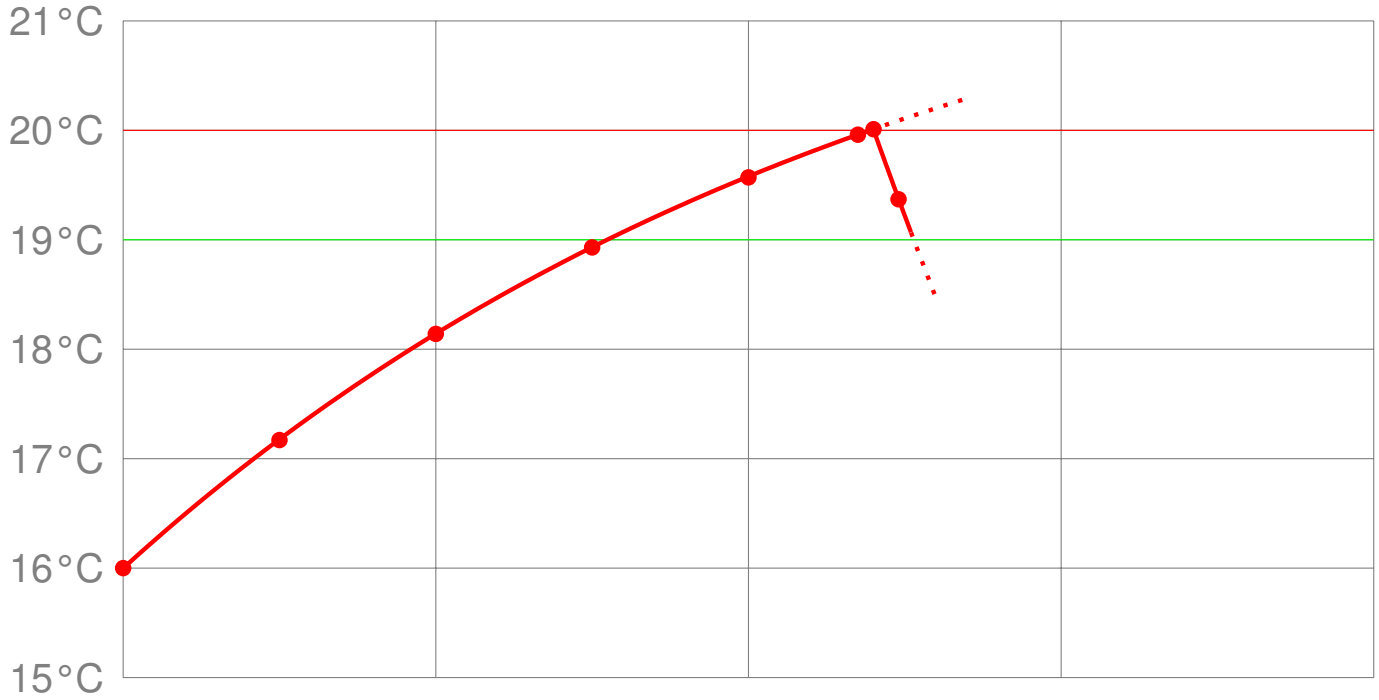
State = cooling

CT Example with Non-Validation



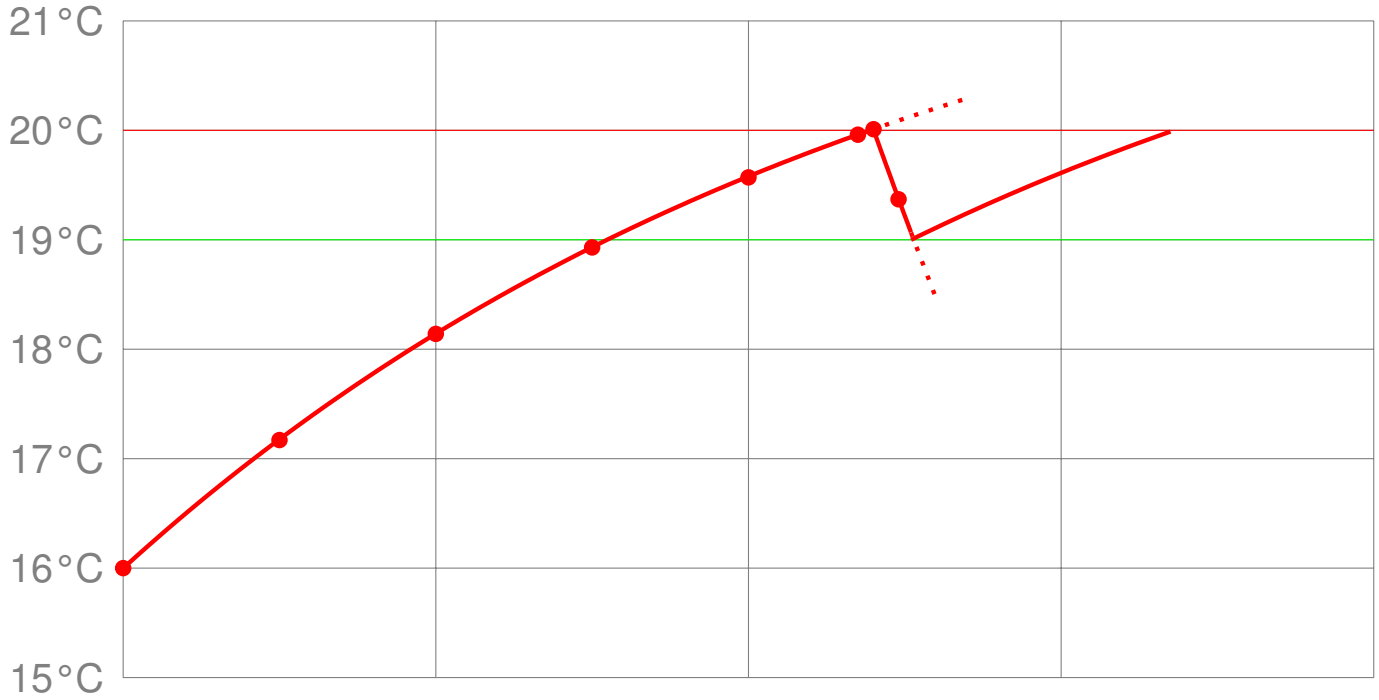
State = cooling

CT Example with Non-Validation

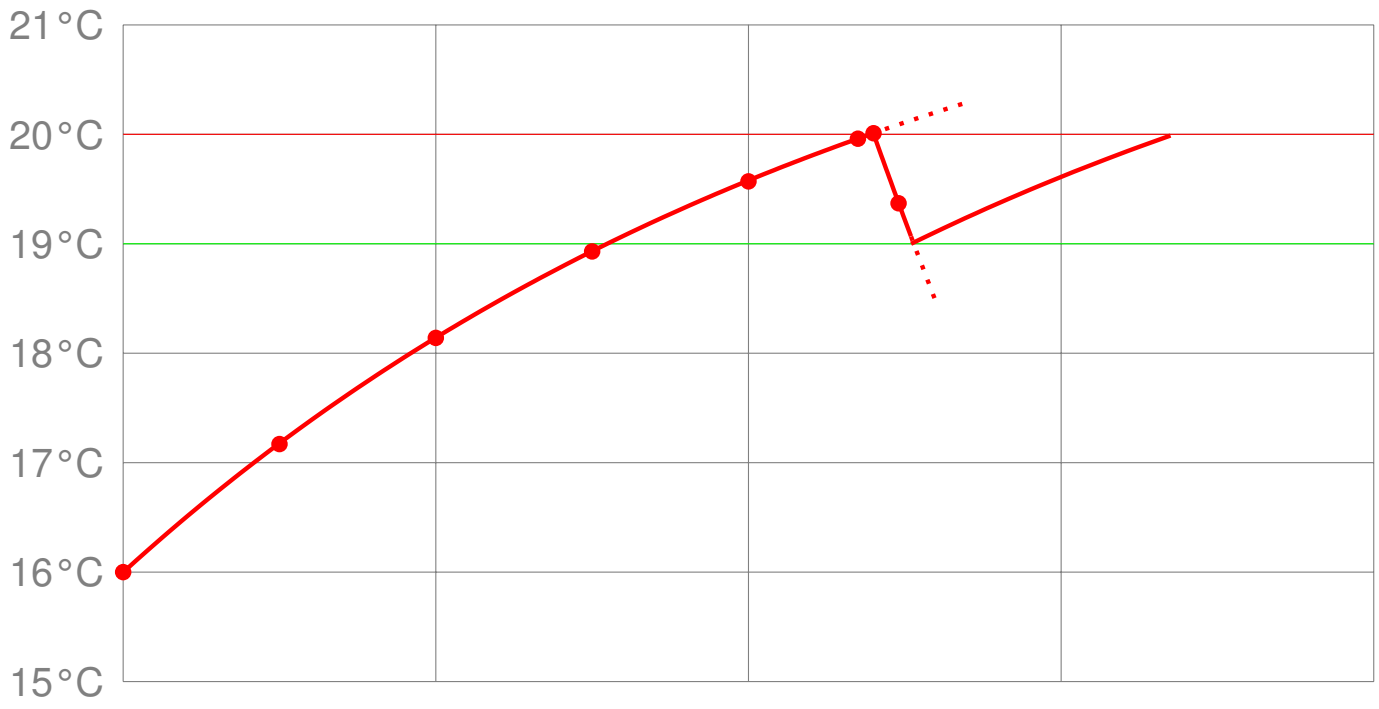


State = cooling

CT Example with Non-Validation



CT Example with Non-Validation



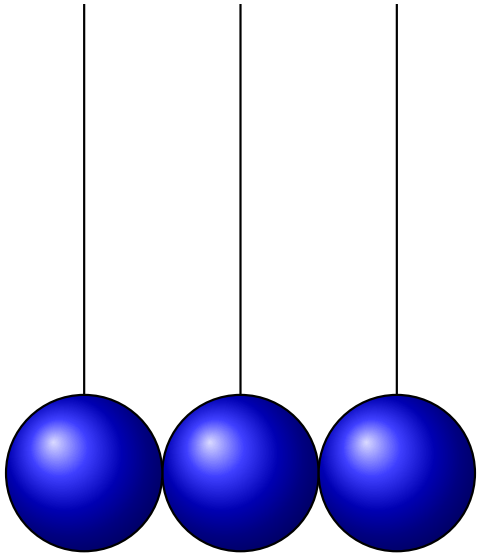
back

Super-Dense Time

Goal

Model causality relations between events that occur at the same date

Sequencing of instantaneous actions



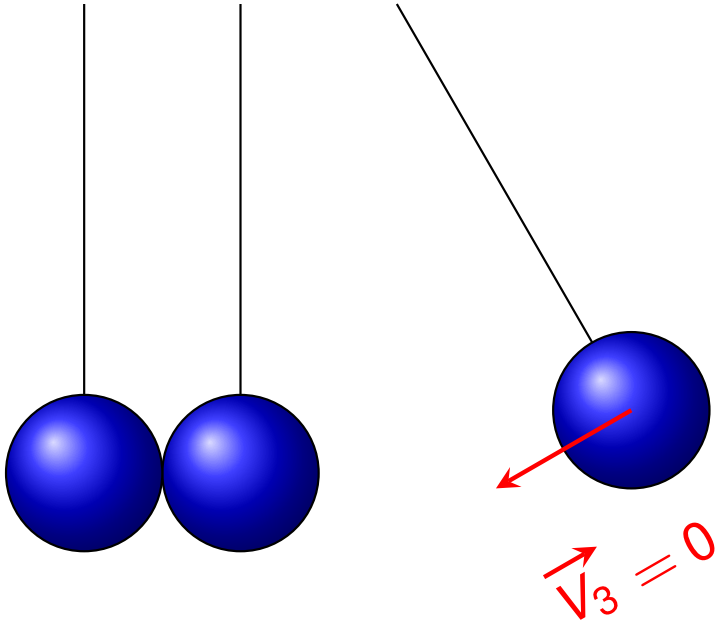
$$t \in \mathbb{R} \times \mathbb{N}$$

Super-Dense Time

Goal

Model causality relations between events that occur at the same date

Sequencing of instantaneous actions



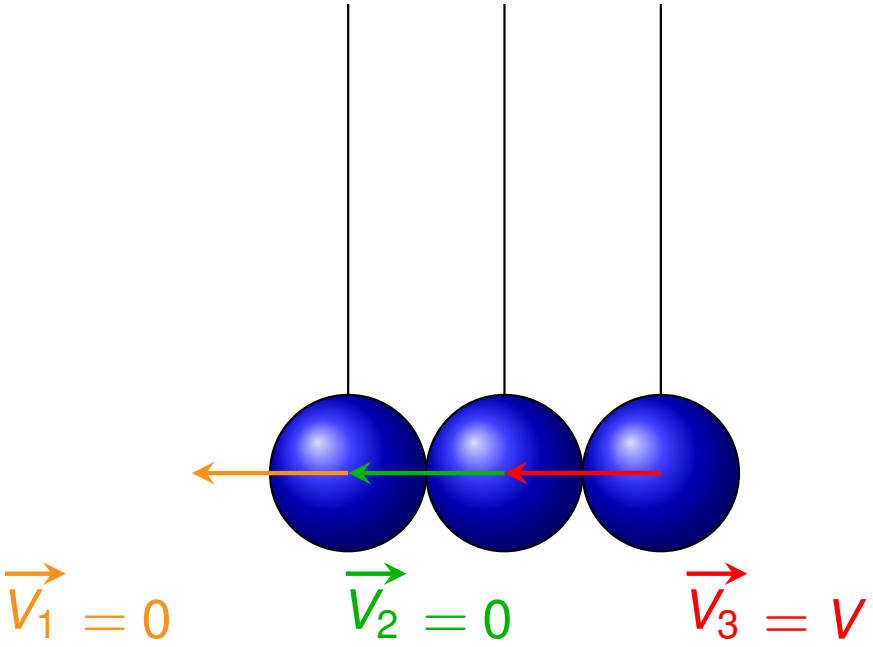
$$t = \langle t_0, 0 \rangle$$

Super-Dense Time

Goal

Model causality relations between events that occur at the same date

Sequencing of instantaneous actions



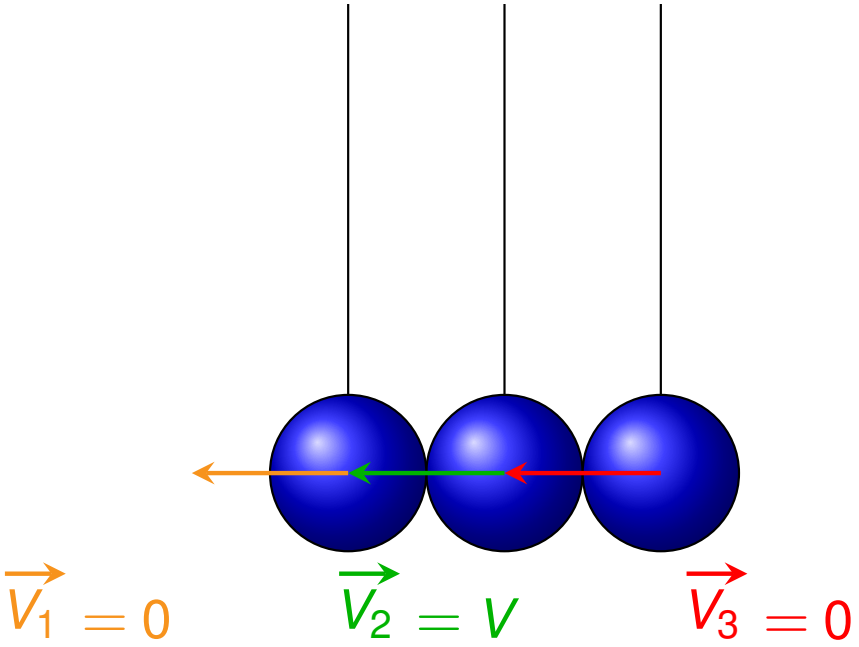
$$t = \langle t_1, 0 \rangle$$

Super-Dense Time

Goal

Model causality relations between events that occur at the same date

Sequencing of instantaneous actions



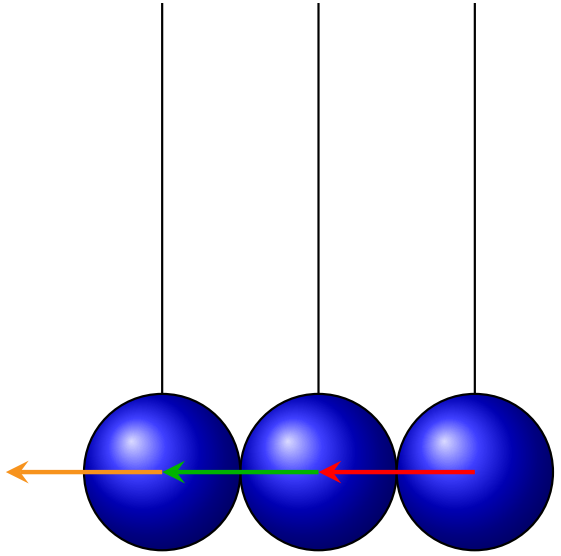
$$t = \langle t_1, 1 \rangle$$

Super-Dense Time

Goal

Model causality relations between events that occur at the same date

Sequencing of instantaneous actions



$\overrightarrow{V_1} = V$ $\overrightarrow{V_2} = 0$ $\overrightarrow{V_3} = 0$

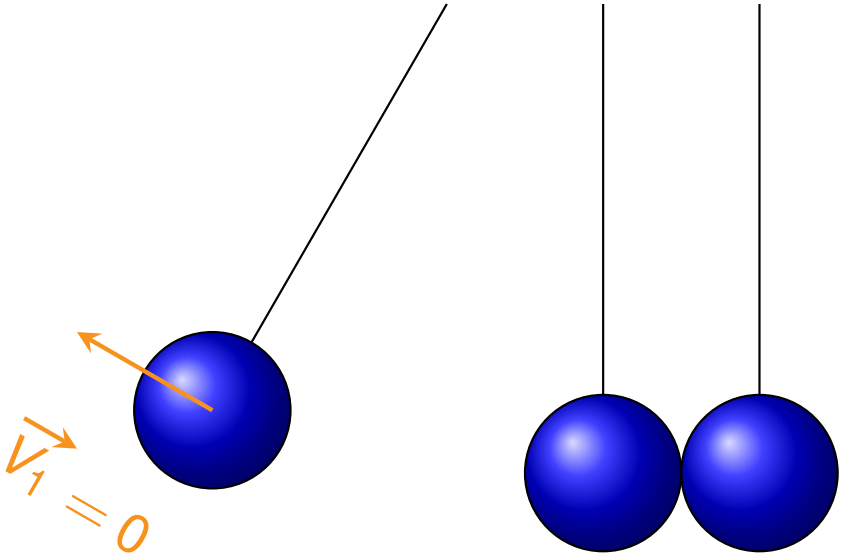
$$t = \langle t_1, 2 \rangle$$

Super-Dense Time

Goal

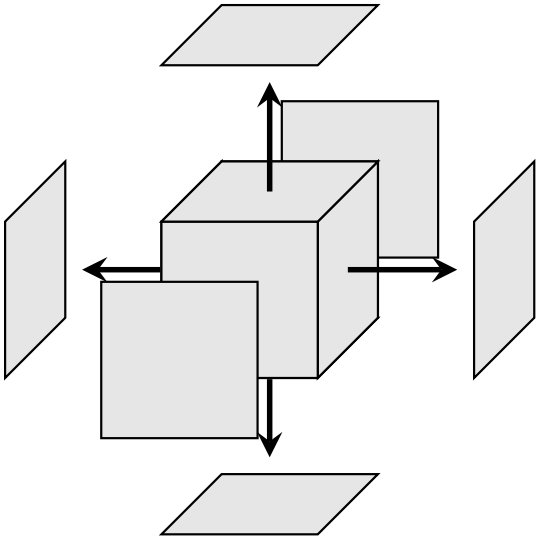
Model causality relations between events that occur at the same date

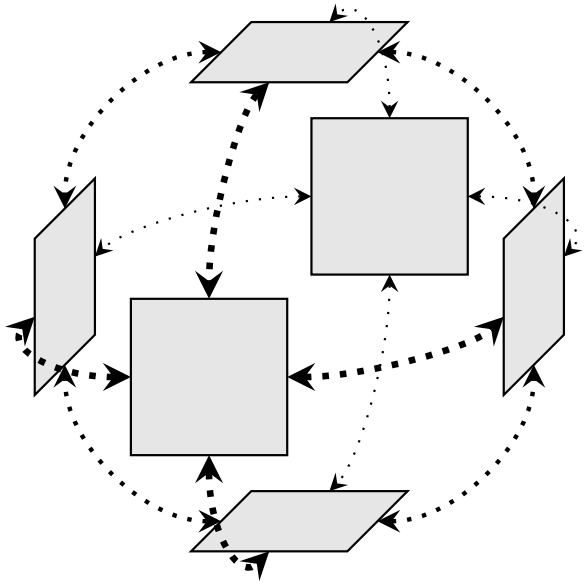
Sequencing of instantaneous actions



$$t = \langle t_2, 0 \rangle$$

Multi-View Modeling





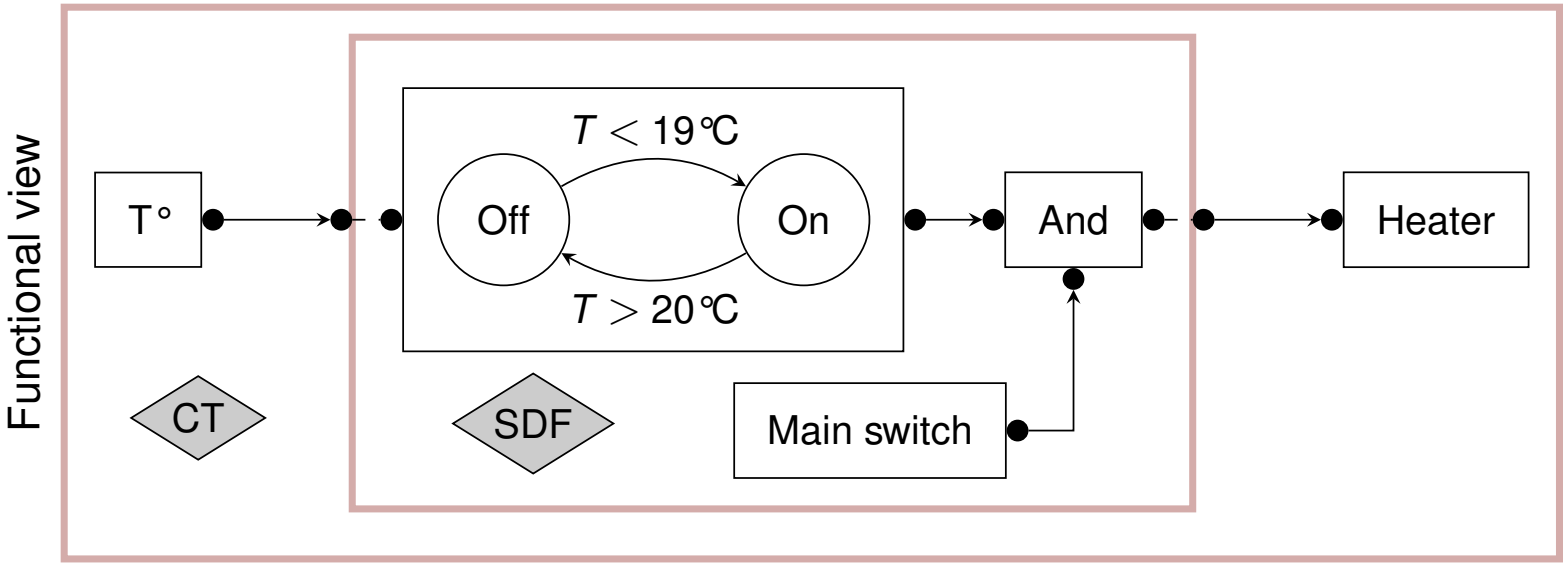
Explicit global model

- + consistency
- + integration
- – language
- – openness

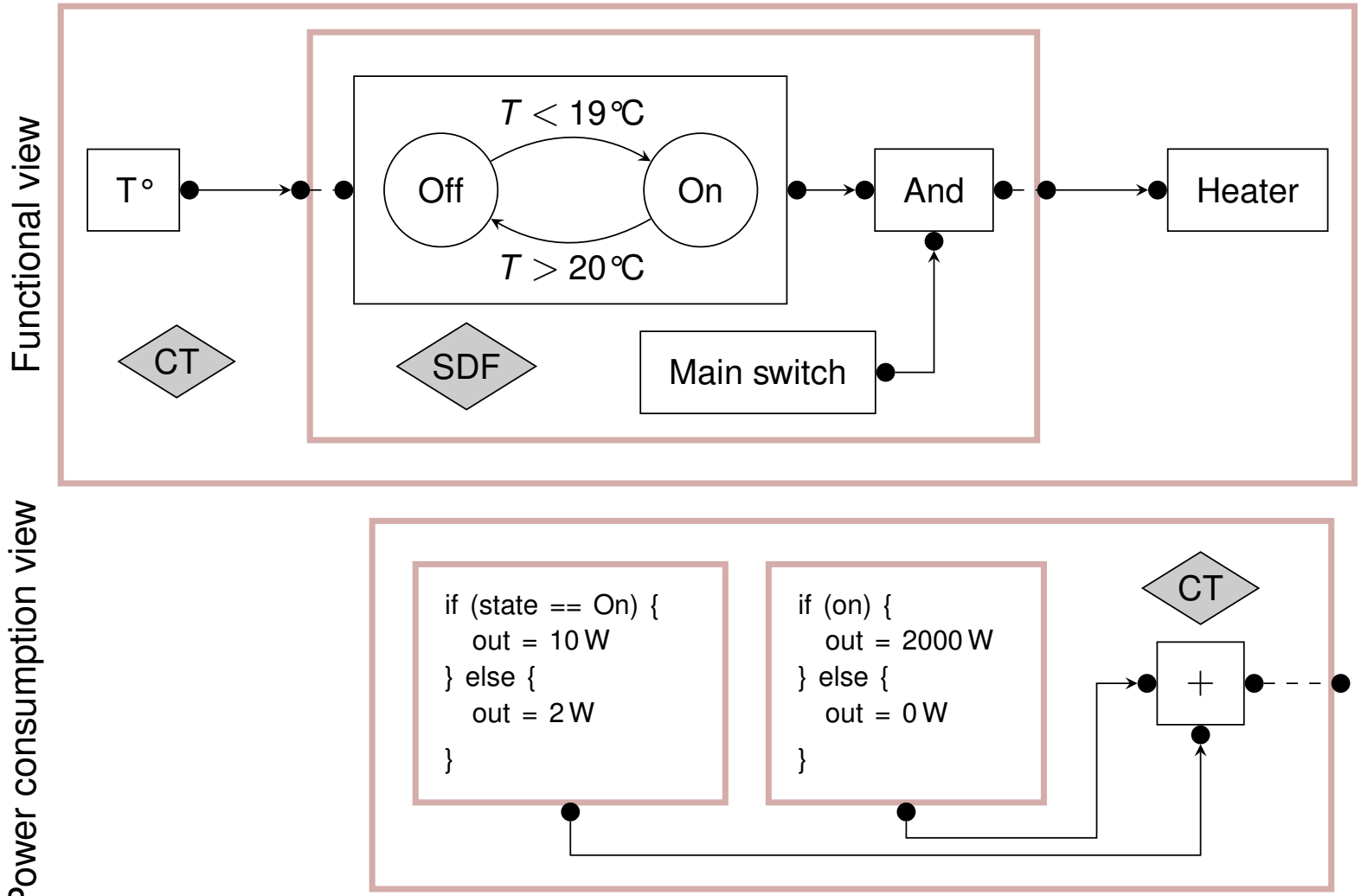
Virtual global model

- – consistency
- – integration/articulation
- + languages
- + openness

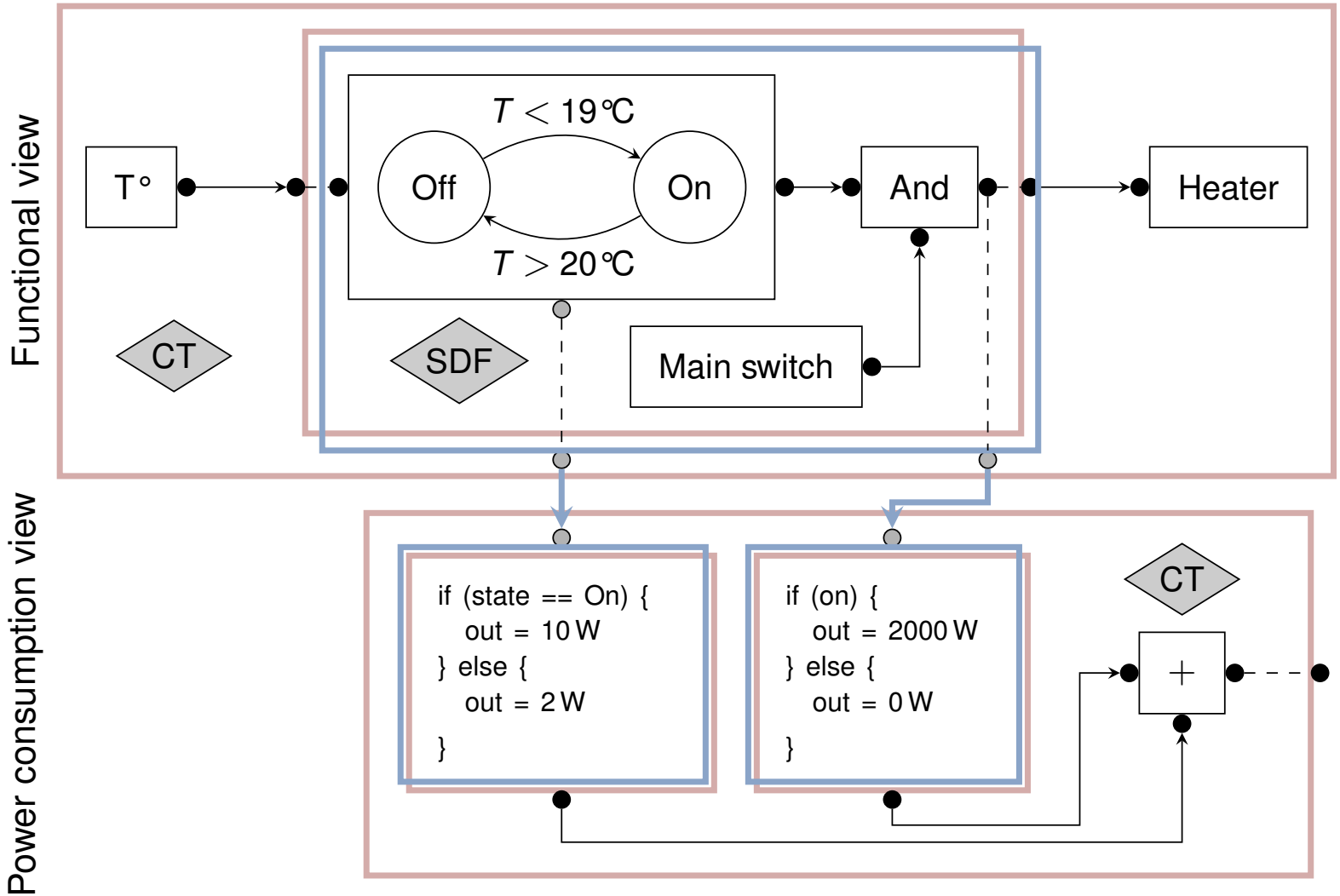
Multi-View Modeling in ModHel'X



Multi-View Modeling in ModHel'X



Multi-View Modeling in ModHel'X



Multi-View Modeling in ModHel'X