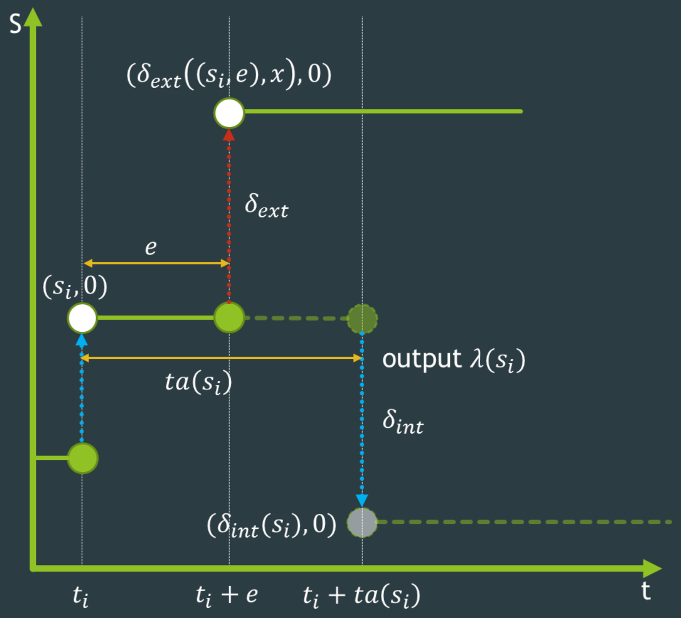


Discrete Event System Specification (DEVS) Modelling and Simulation

An Introduction to "Classic" DEVS Using PythonPDEVS

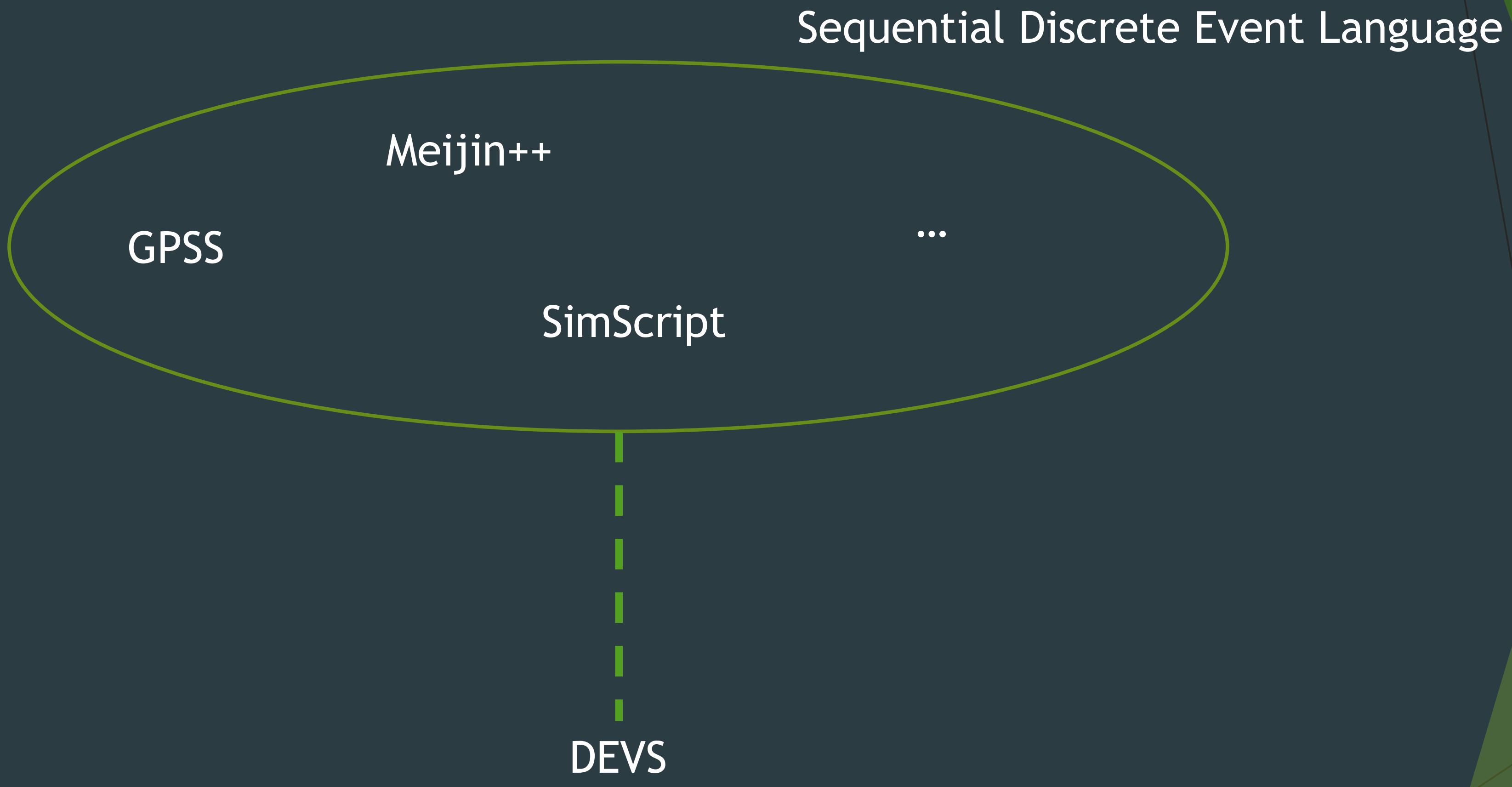
Yentl Van Tendeloo, Hans Vangheluwe, Romain Franceschini

Universiteit Antwerpen

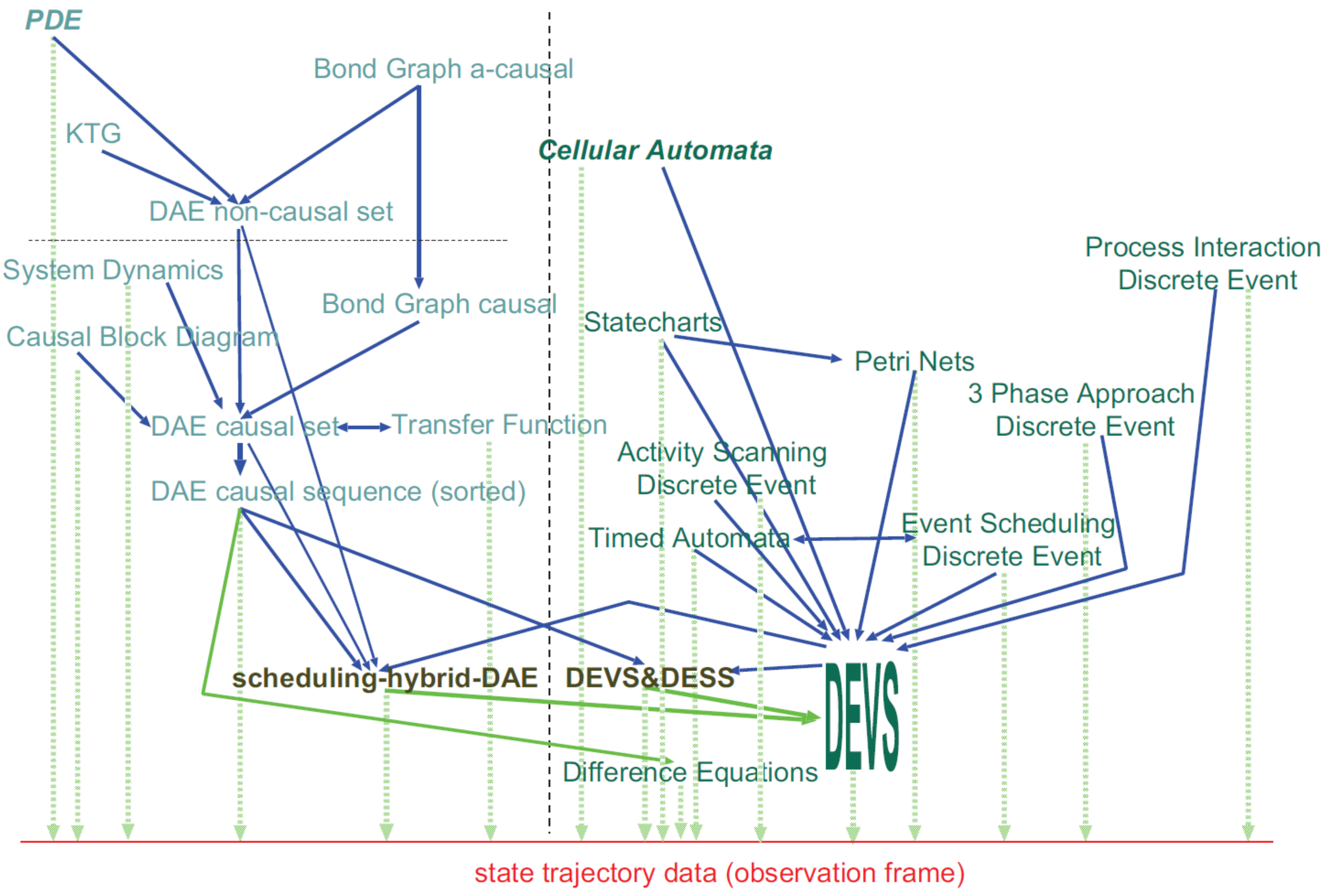
FLANDERS MAKE

McGill

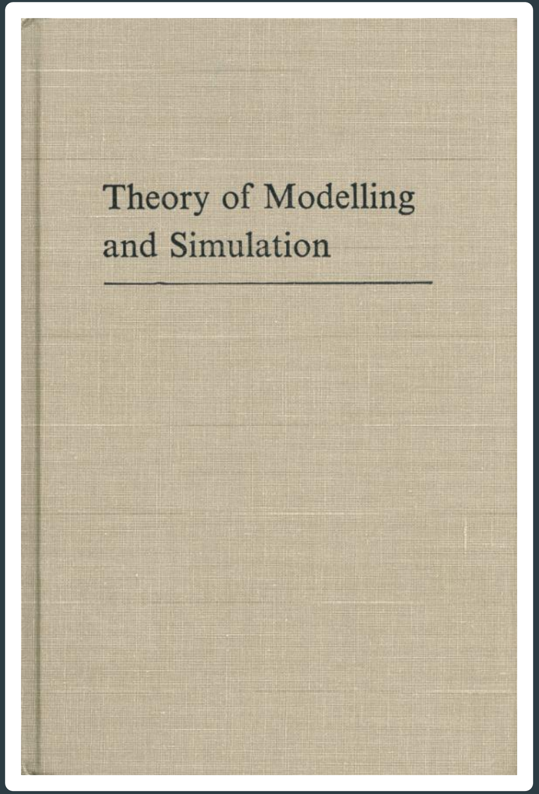
Introduction



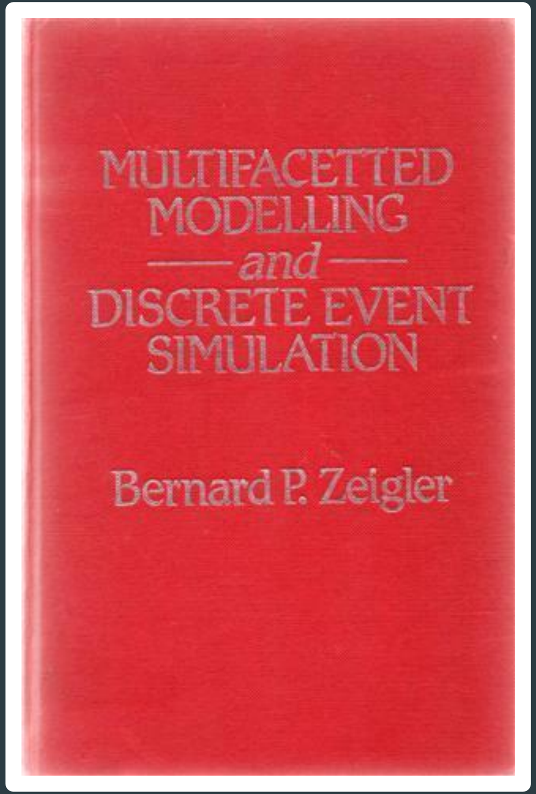
= modular simulation assembly language



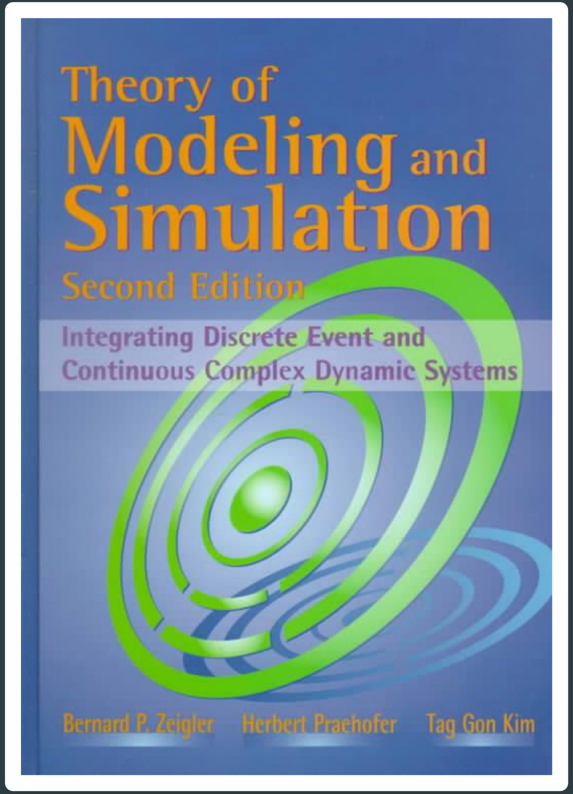
Vangheluwe, Hans. DEVS as a common denominator for multi-formalism hybrid systems modelling.
In proceedings of the International Symposium on Computer-Aided Control System Design, pp. 129-134. 2000.



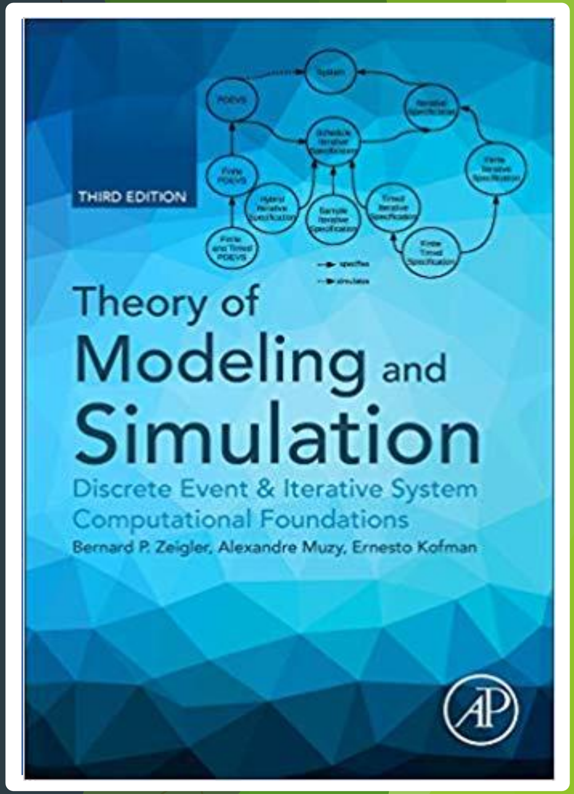
Bernard P. Zeigler.
Theory Of Modelling And Simulation.
1st ed. Wiley, 1976.



Bernard P. Zeigler.
Multifacetted Modelling and Discrete Event Simulation.
1st ed. Academic Press, 1984.



Bernard P. Zeigler, Herbert Praehofer, and Tag Gon Kim.
Theory Of Modelling And Simulation
.2nd ed. Academic Press, 2000.



Bernard P. Zeigler, Alexandre Muzy, and Ermesto Kofman.
Theory Of Modelling And Simulation.
3rd ed. Academic Press, 2018.

An overview of PythonPDEVS

Yentl Van Tendeloo[1] Hans Vangheluwe[1,2]
[1] University of Antwerp, Belgium
[2] McGill University, Canada

Yentl.VanTendeloo@uantwerpen.be
Hans.Vangheluwe@uantwerpen.be

Simulation

Methodology

Simulation: Transactions of the Society for Modeling and Simulation International
1–19

DOI: 10.1177/0037549716678330
sim.sagepub.com
SAGE SCS

An evaluation of DEVS simulation tools

Yentl Van Tendeloo[1*] and Hans Vangheluwe[1,2,3*]

Abstract
DEVS is a popular formalism for modeling complex dynamic systems using a discrete-event abstraction. Owing to its popularity, and the simplicity of the simulation kernel, a number of tools have been constructed by academia and industry. However, each of these tools has distinct design goals and a specific programming language implementation. Consequently, each supports a specific set of formalisms, combined with a specific set of features. Performance differs significantly between different tools. We provide an overview of the current state of eight different DEVS simulation tools: ADEVS, CD++, DEVS-Suite, MS4 Me, PowerDEVS, PythonPDEVS, VLE, and X-S-Y. We compare supported formalisms, compliance, features, and performance. This paper aims to help modelers in deciding which tools to use to solve their specific problems. It further aims to help tool builders, by showing the aspects of their tools that could be extended in future tool versions.

Yentl Van Tendeloo and Hans Vangheluwe.
An Overview of PythonPDEVS.
In Proceedings of Journées DEVS Francophones (JDF), pages 59-66, 2016.

Yentl Van Tendeloo and Hans Vangheluwe.
An Evaluation of DEVS Simulation Tools.
Simulation: Transactions of the Society for Modeling and Simulation International.
2017, 93(2): 103-121

Our presentation uses initialized DEVS models, which contain an initial total state. This was left implicit in the original DEVS specification.

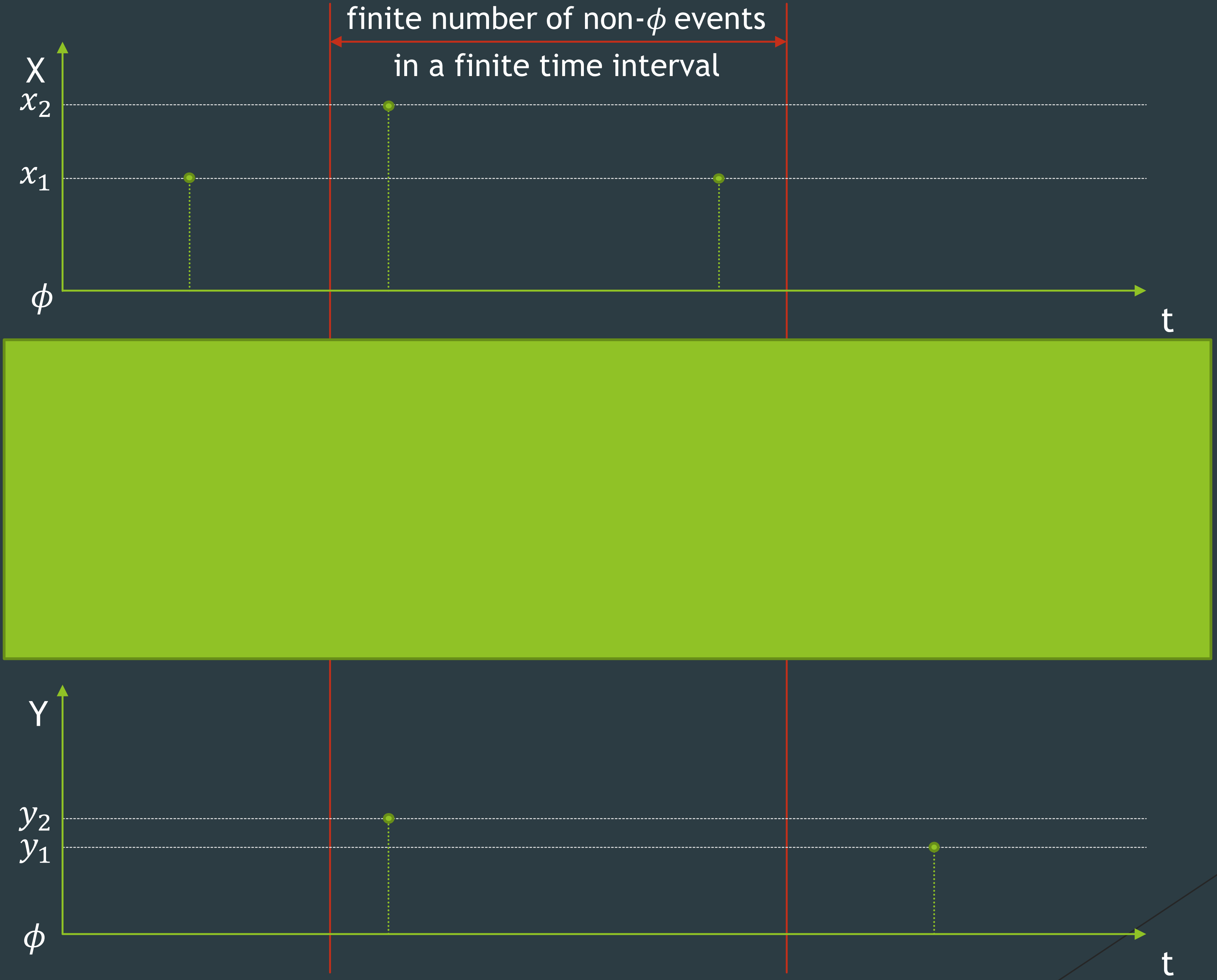
Extending the DEVS Formalism with Initialization Information

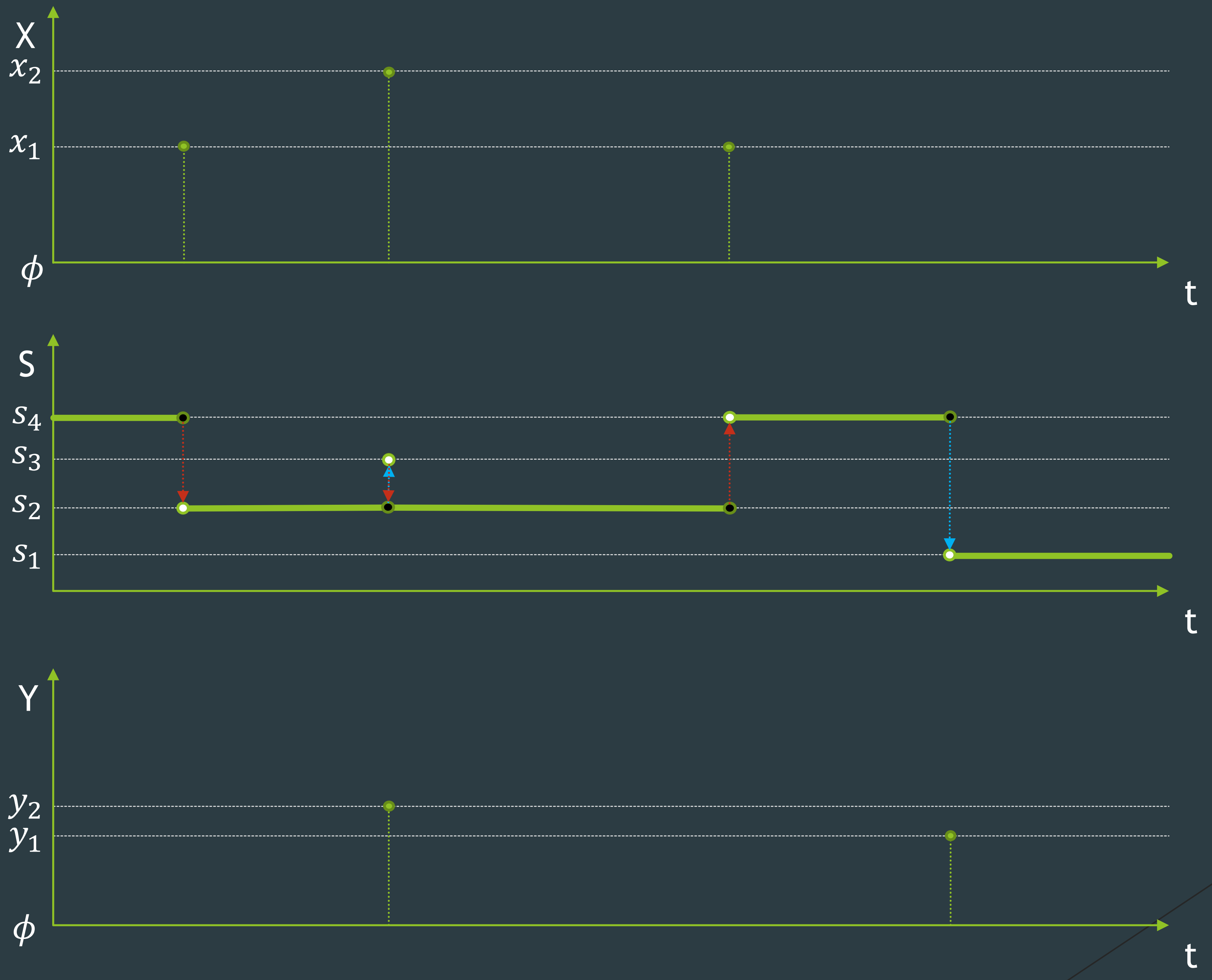
Yentl Van Tendeloo* Hans Vangheluwe*†‡
{Yentl.VanTendeloo,Hans.Vangheluwe}@uantwerpen.be

DEVS is a popular formalism to model system behaviour using a discrete-event abstraction. The main advantages of DEVS are its rigourous and precise specification, as well as its support for modular, hierarchical construction of models. DEVS frequently serves as a simulation "assembly language" to which models in other formalisms are translated, either giving meaning to new (domain-specific) languages, or reproducing semantics of existing languages. Despite this rigourous definition of its syntax and semantics, initialization of DEVS models is left unspecified in both the Classic and Parallel DEVS formalism definition. In this paper, we extend the DEVS formalism by including an initial total state. Extensions to syntax as well as denotational (closure under coupling) and operational semantics (abstract simulator) are presented. The extension is applicable to both main variants of the DEVS formalism. Our extension is such that it adds to, but does not alter the original specification. All changes are illustrated by means of a traffic light example.

Keywords: Classic DEVS, Parallel DEVS, Experimentation, Initialization

Yentl Van Tendeloo and Hans Vangheluwe. Extending the DEVS Formalism with Initialization Information, 2018. ArXiv e-prints.

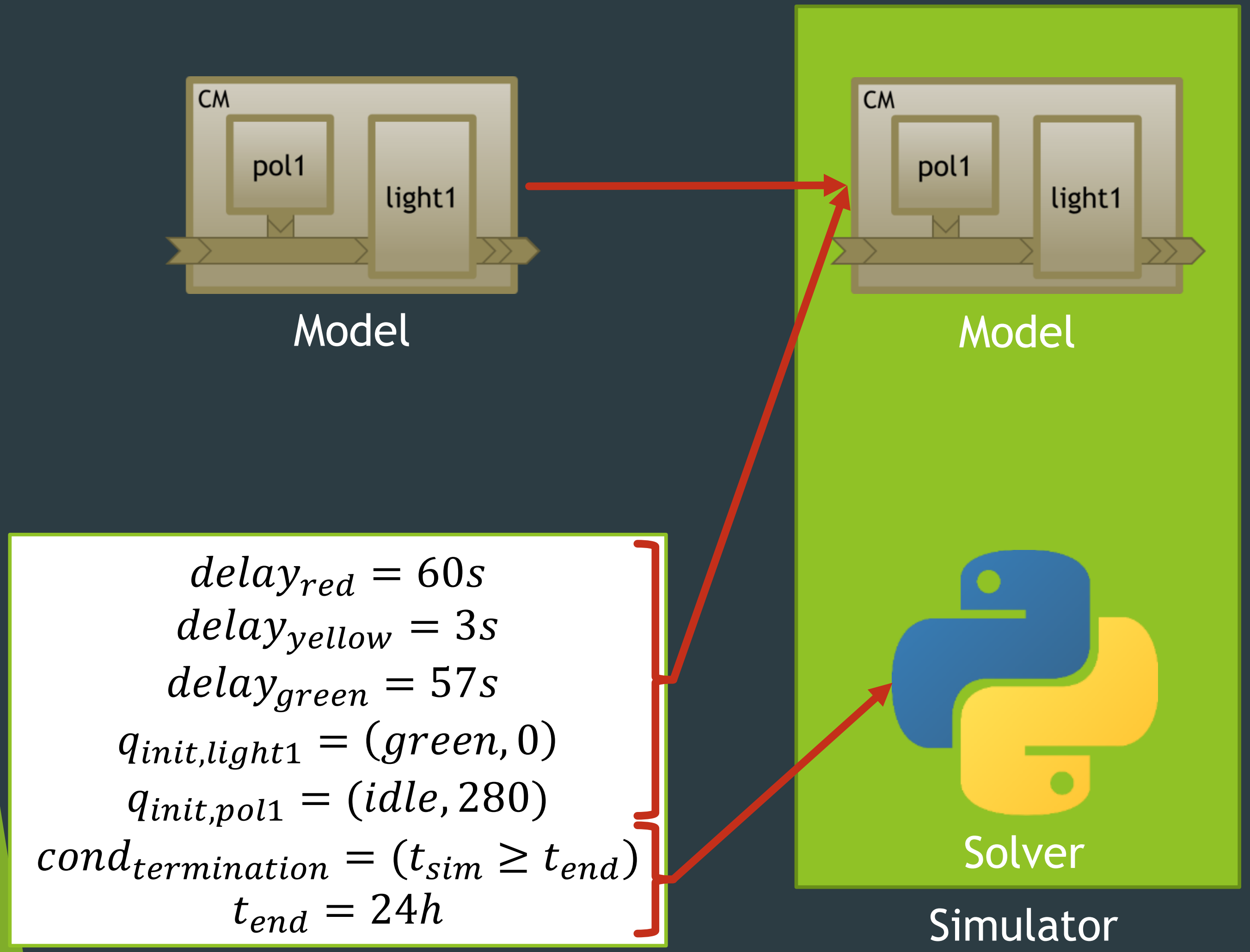

finite number of non-ϕ events
in a finite time interval
X
x_2
x_1
ϕ
t
Y
y_2
y_1
ϕ
t


X
x_2
x_1
ϕ
t
S
s_4
s_3
s_2
s_1
t
Y
y_2
y_1
ϕ
t


TRAFFIC

Experimentation

Simulation

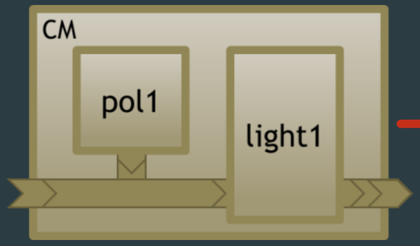

CM
pol1
light1
Model
CM
pol1
light1
Model
$delay_{red} = 60s$
$delay_{yellow} = 3s$
$delay_{green} = 57s$
$q_{init,light1} = (green, 0)$
$q_{init,pol1} = (idle, 280)$
$cond_{termination} = (t_{sim} \geq t_{end})$
$t_{end} = 24h$
Solver
Simulator

Concrete Syntax

simple_experiment.py

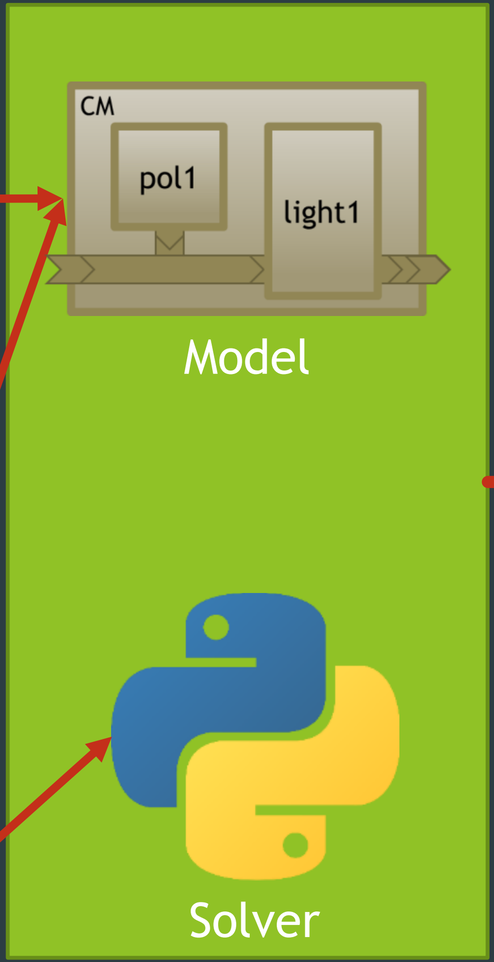
```
from pypdevs.simulator import Simulator

from mymodel import MyModel

model = MyModel(\
            q_init_pol1 = (“idle”, 280),
            q_init_light1 = (“green”, 0),
            delay_red = 60,
            delay_yellow = 3,
            delay_green = 57
)
simulator = Simulator(model)

simulator.setTerminationTime(24*60*60)
simulator.setClassicDEVS()
simulator.setVerbose()

simulator.simulate()
```

Simulation



Model

$$delay_{red} = 60s$$
$$delay_{yellow} = 3s$$
$$delay_{green} = 57s$$
$$q_{init,light1} = (green, 0)$$
$$q_{init,pol1} = (idle, 280)$$
$$cond_{termination} = (t_{sim} \geq t_{end})$$
$$t_{end} = 24h$$



Simulator

__ Current Time: 0.00 ____________

INITIAL CONDITIONS in model <system.light>
Initial State: green
Next scheduled internal transition at time 57.00

INITIAL CONDITIONS in model <system.policeman>
Initial State: idle
Next scheduled internal transition at time 20.00

__ Current Time: 20.00 ____________

EXTERNAL TRANSITION in model <system.light>
Input Port Configuration:
port <interrupt>:
toManual
New State: going_manual
Next scheduled internal transition at time 20.0

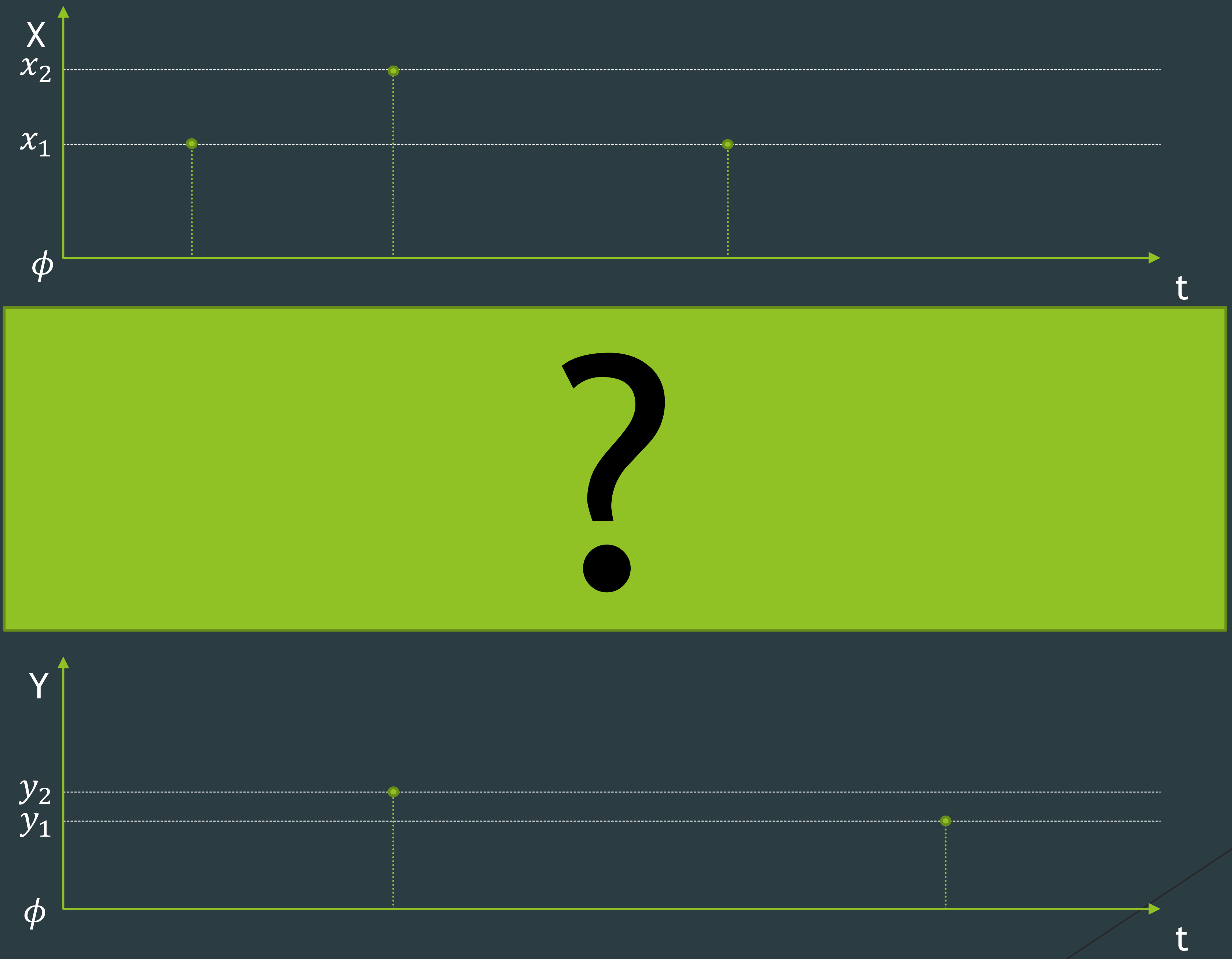
INTERNAL TRANSITION in model <system.policeman>
New State: working
Output Port Configuration:
port <output>:
go_to_work
Next scheduled internal transition at time 3620.00

__ Current Time: 20.00 ____________

INTERNAL TRANSITION in model <system.light>
Output Port Configuration:
port <observer>:
turn_off
New State: manual
Next scheduled internal transition at time inf

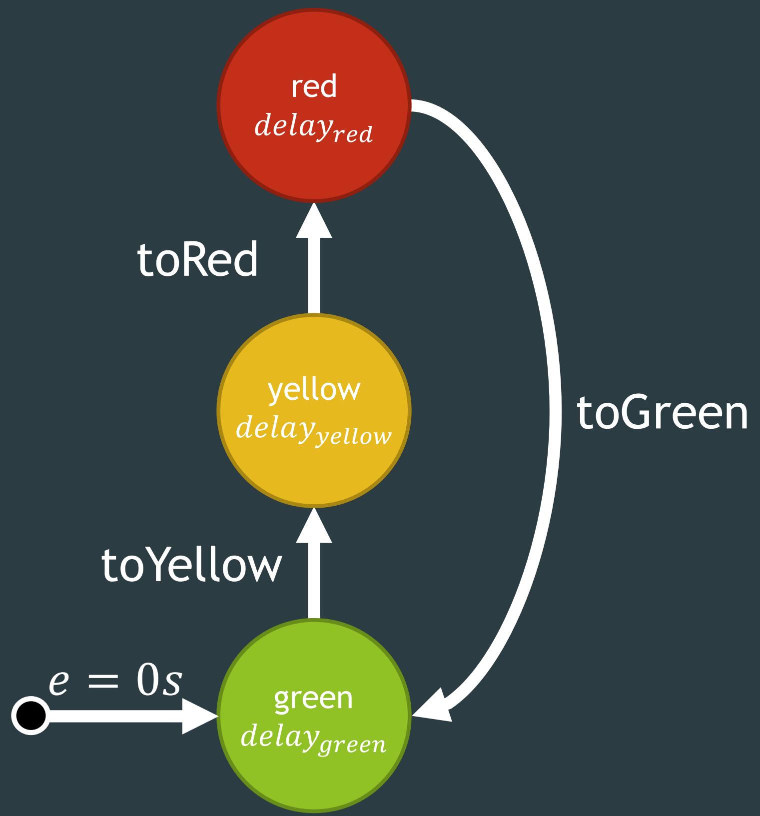
Simulated (Behaviour) Trace

Atomic Models

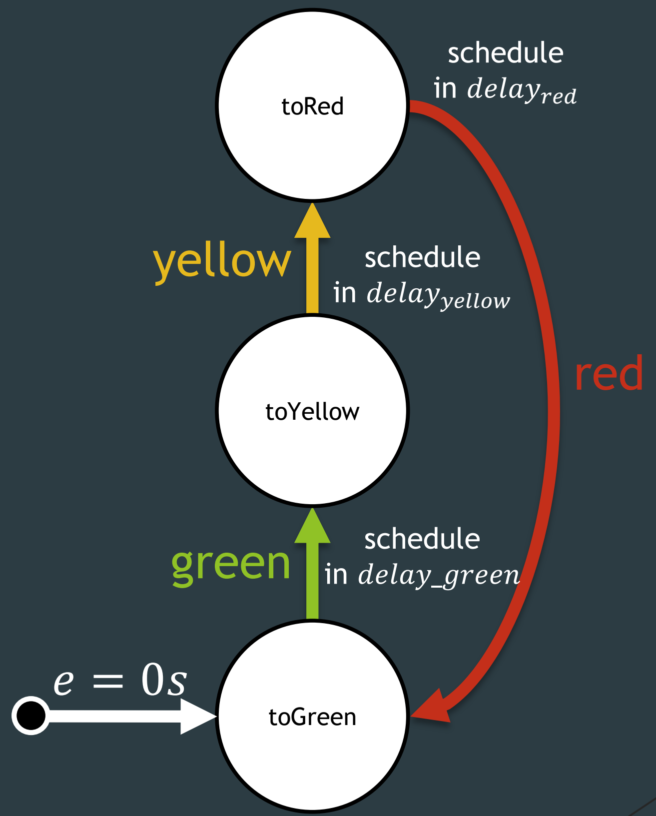

X
x_2
x_1
ϕ
t
?
Y
y_2
y_1
ϕ
t

Modelling Discrete Event Behaviour

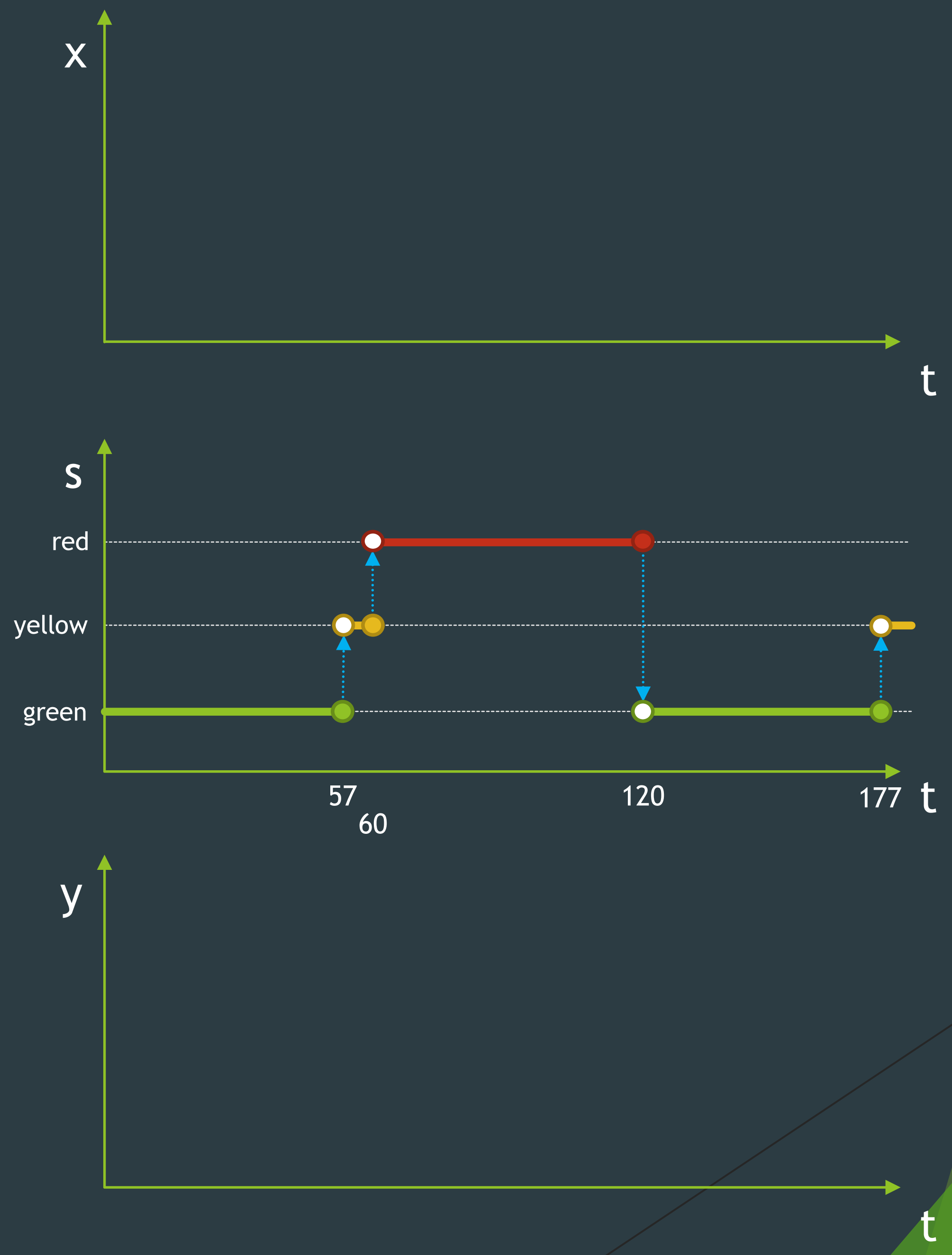
Finite State Automaton



Timed Event Scheduling Graph




red
$delay_{red}$
yellow
$delay_{yellow}$
green
$delay_{green}$


x
t
s
red
yellow
green
57
60
120
177
t
y
t

Autonomous (no input)

$M = \langle\, S\,,\qquad ,\, \delta_{int}\ \ , ta\, \rangle$

S : set of sequential states

S = {red, yellow, green}

$\delta_{int} : S \rightarrow S$

δ_{int} = {red → green,
green → yellow,
yellow → red}

$ta : S \rightarrow \mathbb{R}^{+}_{0,+\infty}$

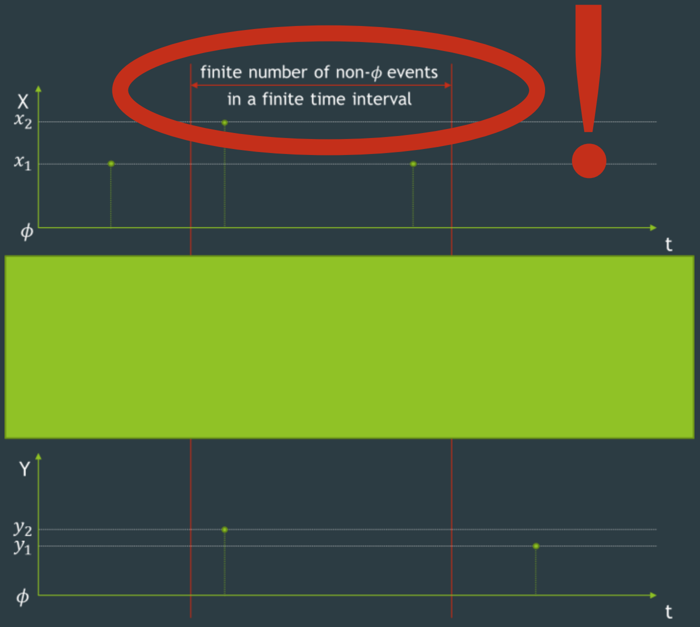
ta = {red → $delay_{red}$,
green → $delay_{green}$,
yellow → $delay_{yellow}$}

Time Advance: corner cases

$$ta : S \rightarrow \mathbb{R}^{+}_{0,+\infty}$$

$$ta(s) = 0$$

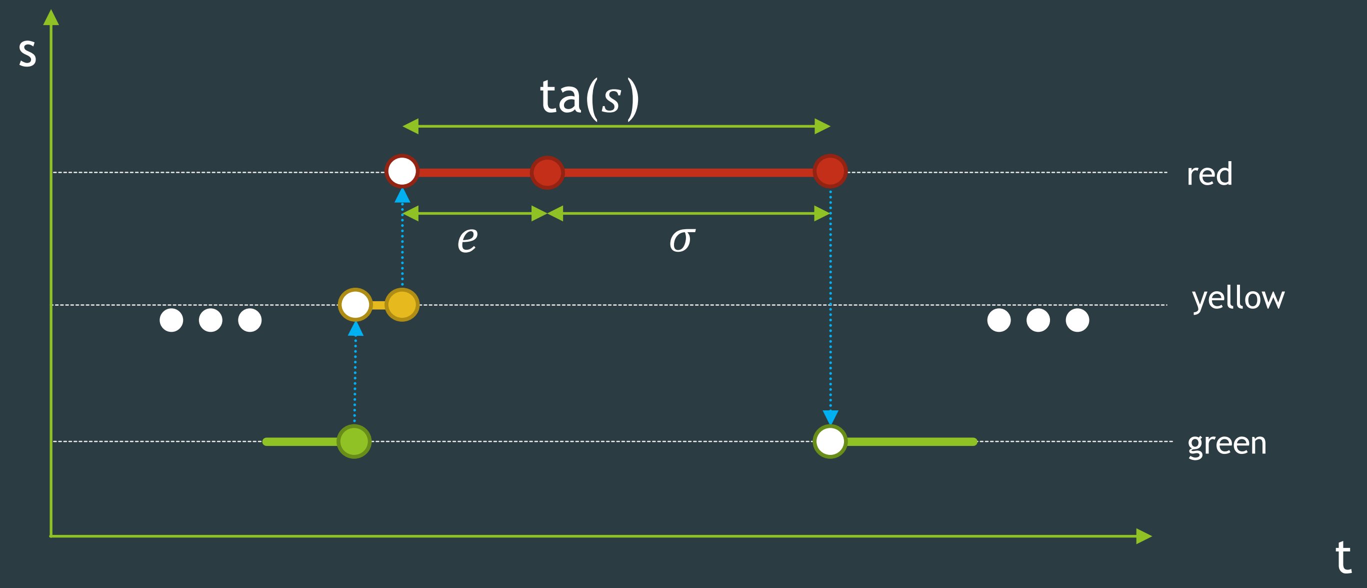
transient states



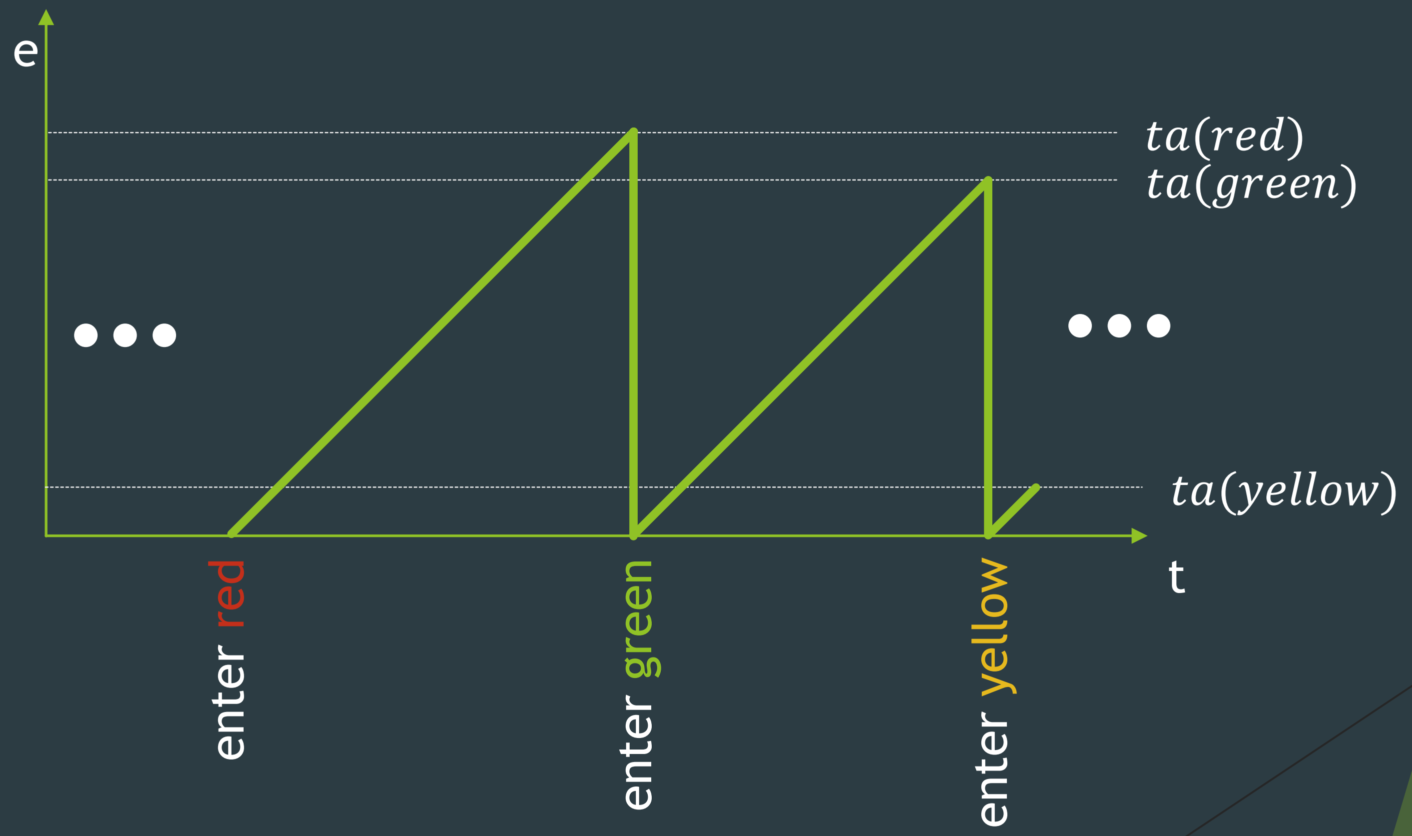
$$ta(s) = +\infty$$

passive states

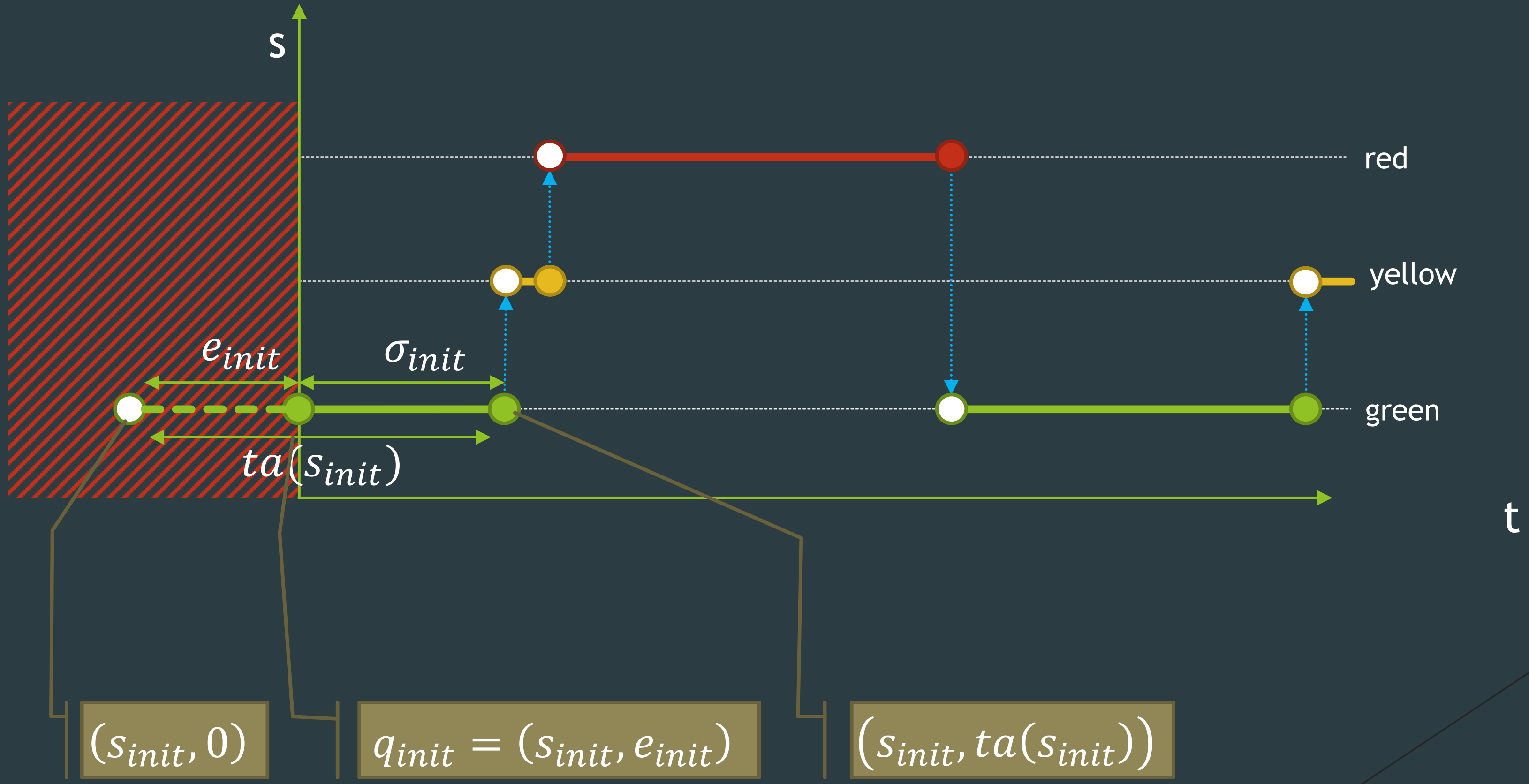
Elapsed time



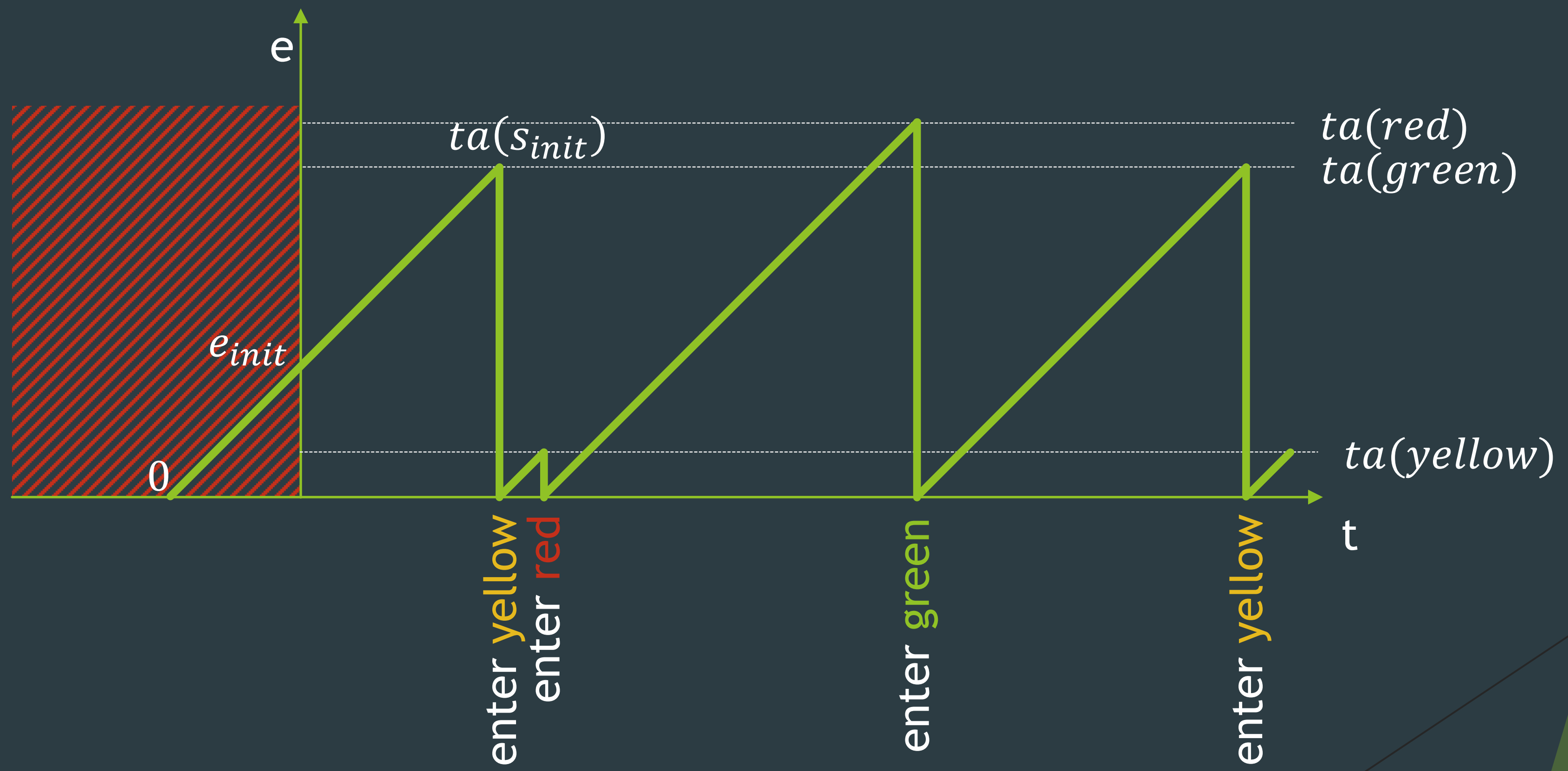
Elapsed time

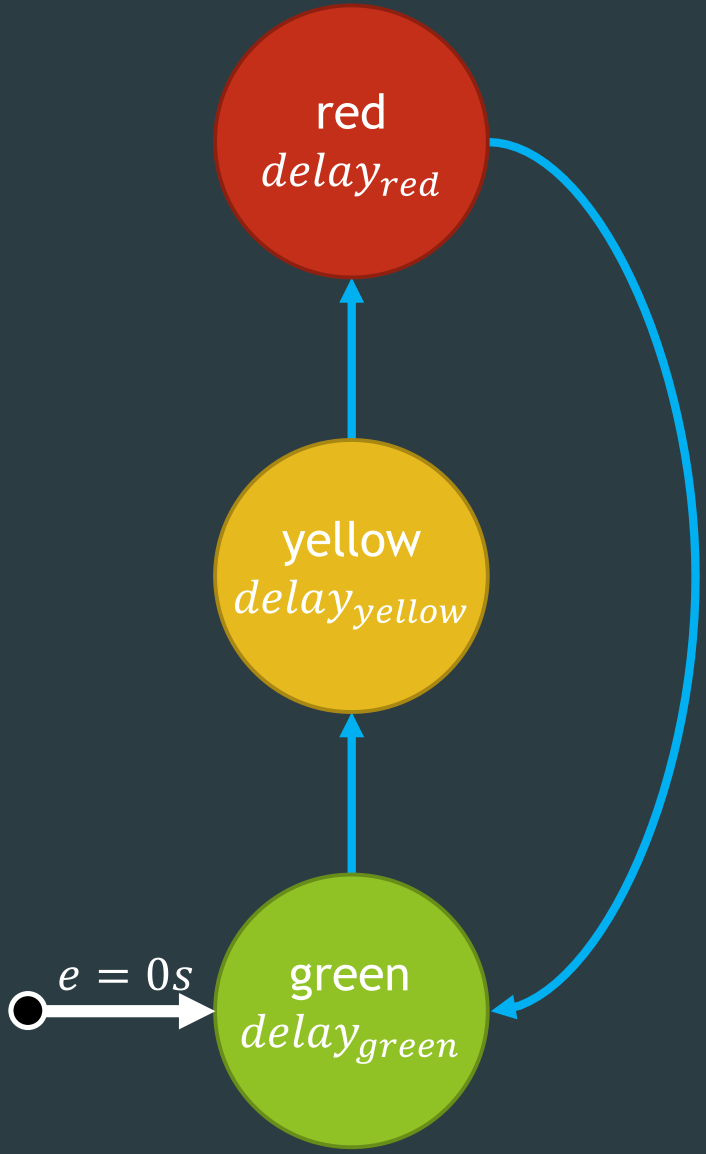


Initialization of State



Elapsed time





Autonomous, no output

$M = \langle S, q_{init}, \delta_{int}, ta \rangle$

S : set of sequential states
S = {red, yellow, green}

$\delta_{int} : S \rightarrow S$
δ_{int} = {red → green,
green → yellow,
yellow → red}

$ta : S \rightarrow \mathbb{R}^{+}_{0,+\infty}$
ta = {red → $delay_{red}$,
green → $delay_{green}$,
yellow → $delay_{yellow}$}

q_{init} : Q - set of total states
$Q = \{(s,e) | s \in S, 0 \leq e \leq ta(s)\}$
q_{init} = (green, 0)

Abstract Syntax

S = {red, yellow, green}
δ_{int} = { red → green,
green → yellow,
yellow → red}
ta = {red → $delay_{red}$,
green → $delay_{green}$,
yellow → $delay_{yellow}$}
q_{init} = (green, 0)

Operational Semantics

```
time = 0
current_state = initial_state
last_time = -initial_elapsed
while not termination_condition():
    time = last_time + ta(current_state)
    current_state = δ_int(current_state)
    last_time = time
```

total_state is a tuple

Concrete Syntax

atomic_int.py

```
from pypdevs.DEVS import *

class TrafficLightAutonomous(AtomicDEVS):
    def __init__(self, q_init, delay_green,
                        delay_yellow, delay_red):
        AtomicDEVS.__init__(self, "light")
        self.state, self.elapsed = q_init
        self.delay_green = delay_green
        self.delay_yellow = delay_yellow
        self.delay_red = delay_red

    def intTransition(self):
        state = self.state
        return {"red": "green",
                "yellow": "red",
                "green": "yellow"}[state]

    def timeAdvance(self):
        state = self.state
        return {"red": self.delay_red,
                "yellow": self.delay_yellow,
                "green": self.delay_green}[state]
```

__ Current Time: 0.00 ____

INITIAL CONDITIONS in model <light>
Initial State: green
Next scheduled internal transition at time 57.00

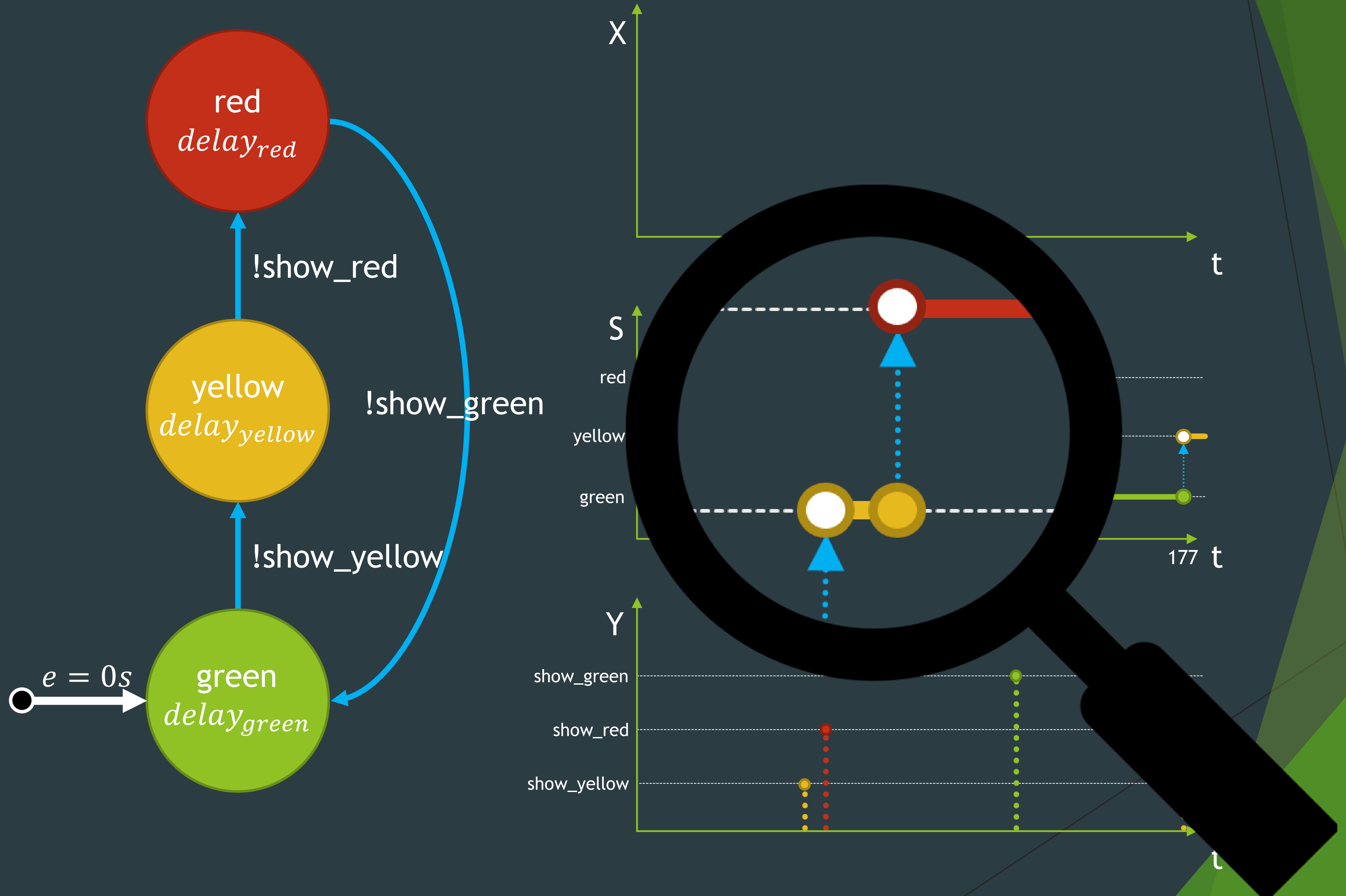
__ Current Time: 57.00 ____

INTERNAL TRANSITION in model <light>
New State: yellow
Output Port Configuration:
Next scheduled internal transition at time 60.00

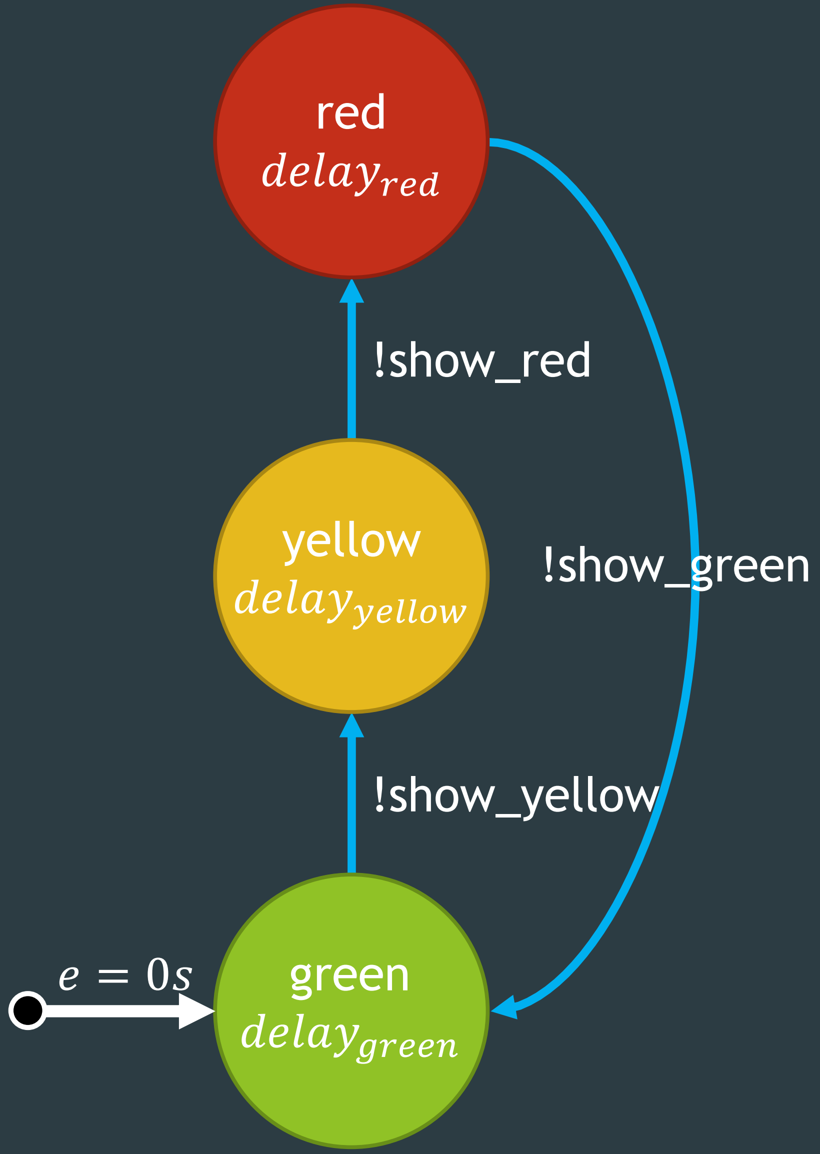
__ Current Time: 60.00 ____

INTERNAL TRANSITION in model <light>
New State: red
Output Port Configuration:
Next scheduled internal transition at time 120.00

red
$delay_{red}$
yellow
$delay_{yellow}$
green
$delay_{green}$
!show_red
!show_yellow
!show_green
$e = 0s$
X
t
S
red
yellow
green
57
60
120
177
t
Y
show_green
show_red
show_yellow
t



red
$delay_{red}$
!show_red
yellow
$delay_{yellow}$
!show_green
!show_yellow
$e = 0s$
green
$delay_{green}$
X
t
S
red
yellow
green
177
t
Y
show_green
show_red
show_yellow
t



Autonomous, with output

$M = \langle Y, S, q_{init}, \delta_{int}, \lambda, ta \rangle$

S = {red, yellow, green}
δ_{int} = { red → green, green → yellow, yellow → red}
q_{init} = (green, 0)
ta = {red → $delay_{red}$, green → $delay_{green}$, yellow → $delay_{yellow}$}

Y : set of output events
Y = {“show_red”, “show_green”, “show_yellow”}

$\lambda : S \rightarrow Y \cup \{\phi\}$
λ = { green → “show_yellow”,
yellow → “show_red”,
red → “show_green”}

Abstract Syntax

S = {red, yellow, green}
q_{init} = (green, 0)
δ_{int} = { red → green,
green → yellow,
yellow → red}
ta = {red → $delay_{red}$,
green → $delay_{green}$,
yellow → $delay_{yellow}$}
Y = {“show_red”,
“show_green”,
“show_yellow”}
λ = {green → “show_yellow”,
yellow → “show_red”,
red → “show_green”}

Operational Semantics

```
time = 0
current_state = initial_state
last_time = -initial_elapsed
while not termination_condition():
    time = last_time + ta(current_state)
    output(λ(current_state))
    current_state = δint(current_state)
    last_time = time
```

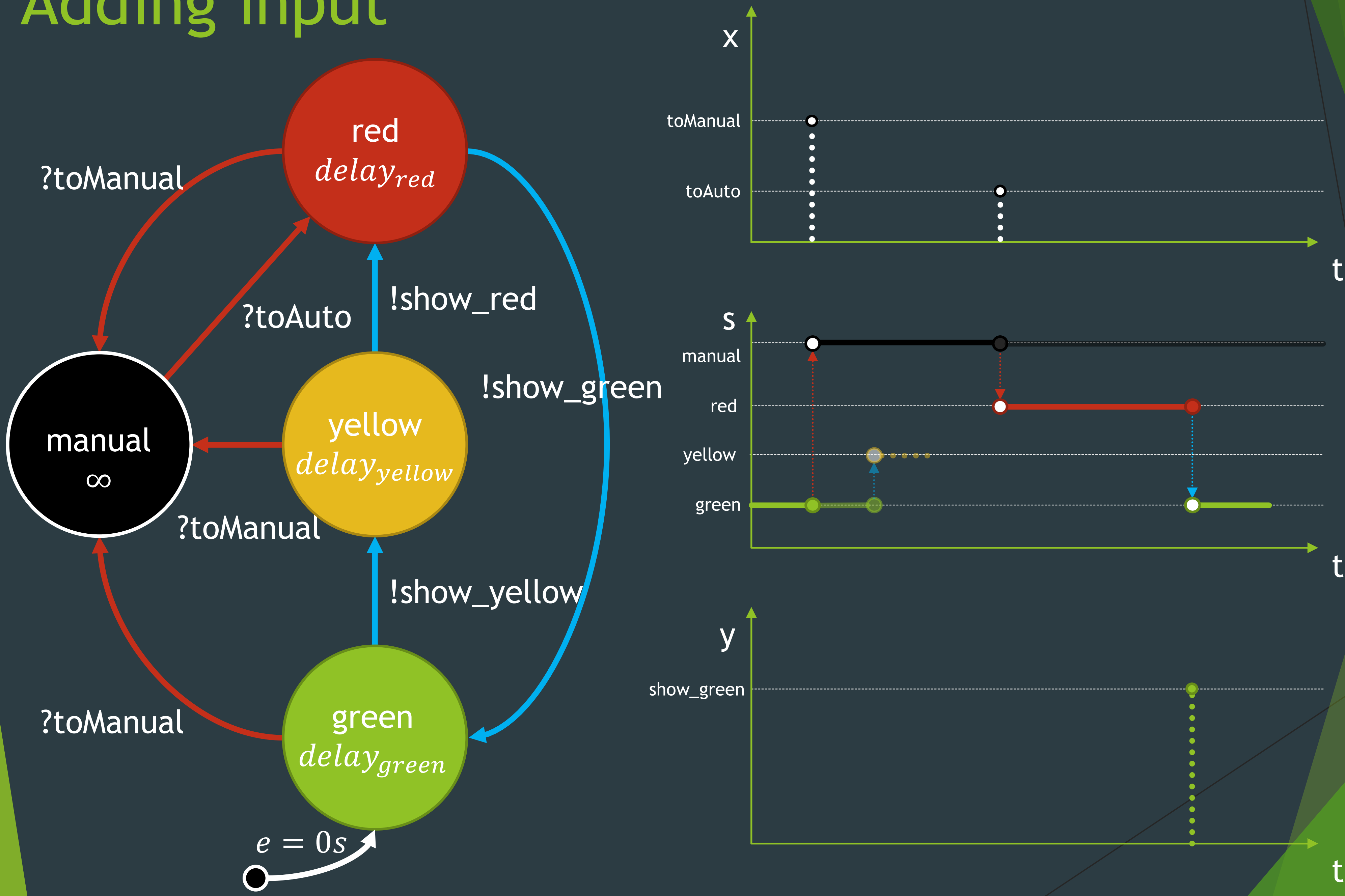
Concrete Syntax

atomic_out.py

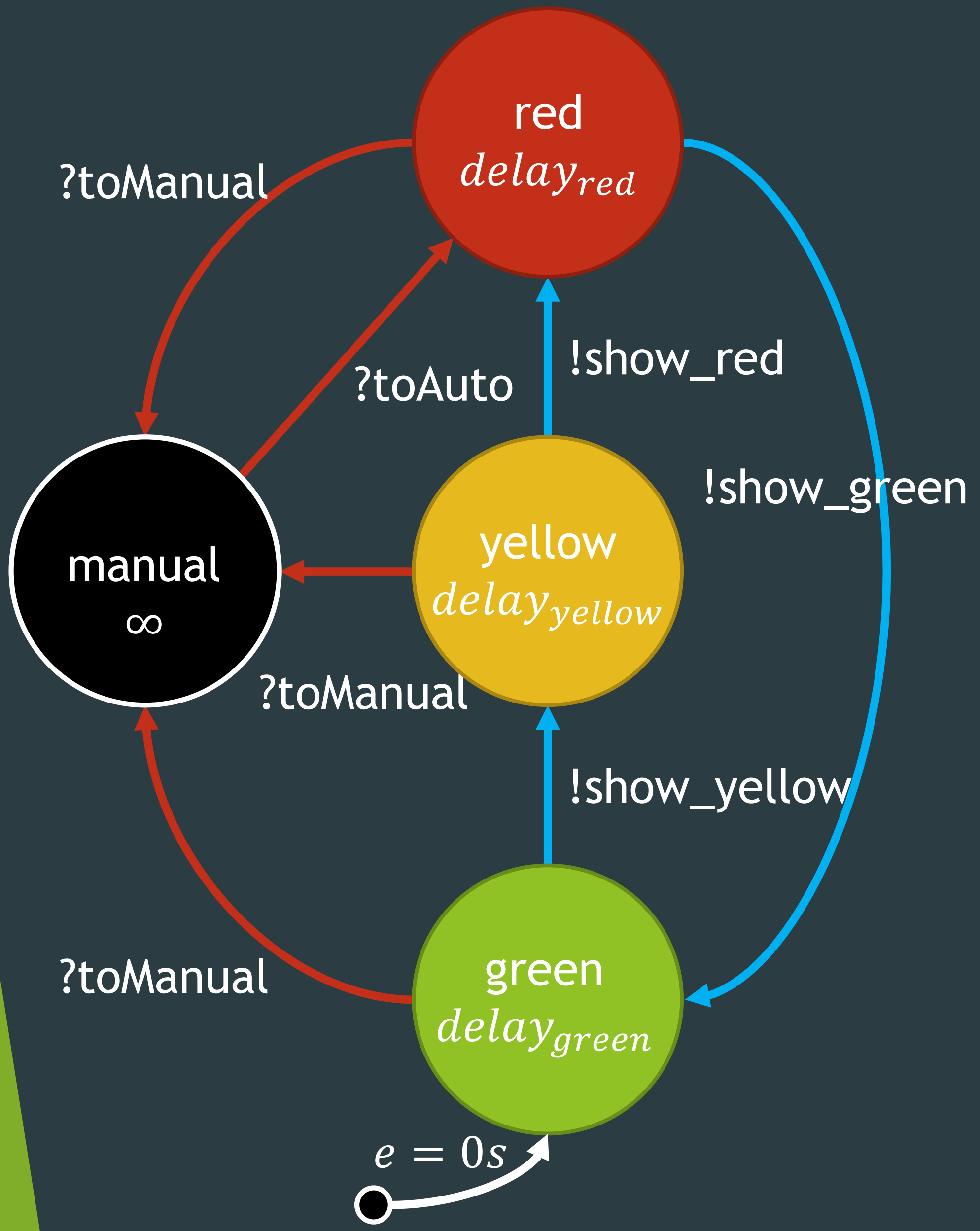
```
from pypdevs.DEVS import *

class TrafficLightWithOutput(AtomicDEVS):
    def __init__(self, …):
        AtomicDEVS.__init__(self, "light")
        self.observe = self.addOutPort("observer")
        …
    …

    def outputFnc(self):
        state = self.state
        if state == "red":
            return {self.observe: "show_green"}
        elif state == "yellow":
            return {self.observe: "show_red"}
        elif state == "green":
            return {self.observe: "show_yellow"}
```

```
__  Current Time:       0.00 __________________________________________

      INITIAL CONDITIONS in model <light>
            Initial State: green
            Next scheduled internal transition at time 57.00

__  Current Time:      57.00 __________________________________________

      INTERNAL TRANSITION in model <light>
            New State: yellow
            Output Port Configuration:
                  port <observer>:
                        show_yellow
            Next scheduled internal transition at time 60.00

__  Current Time:      60.00 __________________________________________

      INTERNAL TRANSITION in model <light>
            New State: red
            Output Port Configuration:
                  port <observer>:
                        show_red
            Next scheduled internal transition at time 120.00
```

Adding input



Adding input



Reactive

$M = \langle X, Y, S, q_{init}, \delta_{int}, \delta_{ext}, \lambda, ta \rangle$

Y = {"show_red", "show_green", "show_yellow"}
S = {red, yellow, green, manual}
q_{init} = (green, 0)
δ_{int} = {red → green,
green → yellow,
yellow → red}
λ = {green → "show_yellow",
yellow → "show_red",
red → "show_green"}
ta = {red → $delay_{red}$,
green → $delay_{green}$,
yellow → $delay_{yellow}$,
manual → +∞}

X : set of input events
X = {"toAuto", "toManual"}

$\delta_{ext} : Q \times X \rightarrow S$
$Q = \{(s, e) | s \in S, 0 \leq e \leq ta(s)\}$
δ_{ext} = {((*, *), "toManual") → "manual",
(("manual", *), "toAuto") → "red"}

Abstract Syntax

Y = {“show_red”, “show_green”, “show_yellow”}
S = {red, yellow, green, manual}
q_{init} = (green, 0)
δ_{int} = {red → green,
green → yellow,
yellow → red}
λ = {green → “show_yellow”,
yellow → “show_red”,
red → “show_green”}
ta = {red → $delay_{red}$,
green → $delay_{green}$,
yellow → $delay_{yellow}$,
manual → ∞}
X = {“toAuto”, “toManual”}
δ_{ext} = {((*, *), “toManual”) → manual,
((manual, *), “toAuto”) → red}

Operational Semantics

```
time = 0
current_state = initial_state
last_time = -initial_elapsed
while not termination_condition():
    next_time = last_time + ta(current_state)
    if time_next_ev <= next_time:
        e = time_next_ev - last_time
        time = time_next_ev
        current_state = δ_ext((current_state, e), next_ev)
    else:
        time = next_time
        output(λ(current_state))
        current_state = δ_int(current_state)
    last_time = time
```

Abstract Syntax

Y = {"show_red", "show_green", "show_yellow"}
S = {red, yellow, green, manual}
q_{init} = (green, 0)
δ_{int} = {red → green,
green → yellow,
yellow → red}
λ = {green → "show_yellow",
yellow → "show_red",
red → "show_green"}
ta = {red → $delay_{red}$,
green → $delay_{green}$,
yellow → $delay_{yellow}$,
manual → ∞}
X = {"toAuto", "toManual"}
δ_{ext} = {((*, *), "toManual") → manual,
((manual, *), "toAuto") → red}

Concrete Syntax

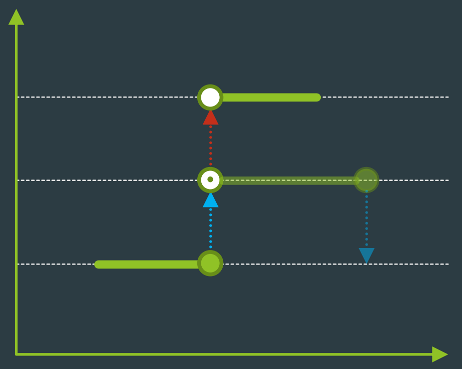
atomic_ext.py

```
from pypdevs.DEVS import *

class TrafficLight(AtomicDEVS):
 def __init__(self, …):
        AtomicDEVS.__init__(self, "light")
        self.interrupt = self.addInPort("interrupt")
        …
    …

    def extTransition(self, inputs):
        inp = inputs[self.interrupt]
        if inp == "toManual":
            return "manual"
        elif inp == "toAuto":
            if self.state == "manual":
                return "red"
```

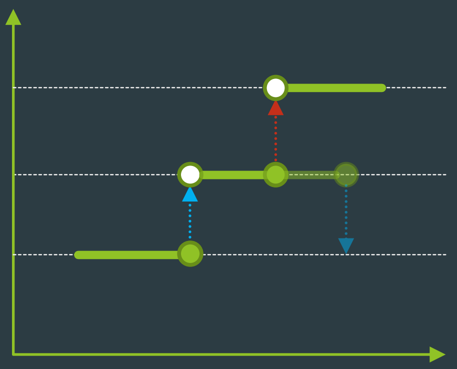
```
__  Current Time:       0.00 ________________________________________________

        INITIAL CONDITIONS in model <light>
                Initial State: green
                Next scheduled internal transition at time 57.00

__  Current Time:      57.00 ________________________________________________

        INTERNAL TRANSITION in model <light>
                New State: yellow
                Output Port Configuration:
                        port <observer>:
                                show_yellow
                Next scheduled internal transition at time 60.00

__  Current Time:      60.00 ________________________________________________

        INTERNAL TRANSITION in model <light>
                New State: red
                Output Port Configuration:
                        port <observer>:
                                show_red
                Next scheduled internal transition at time 120.00
```

$$Q = \{(s,e) | s \in S, 0 \leq e \leq ta(s)\}$$

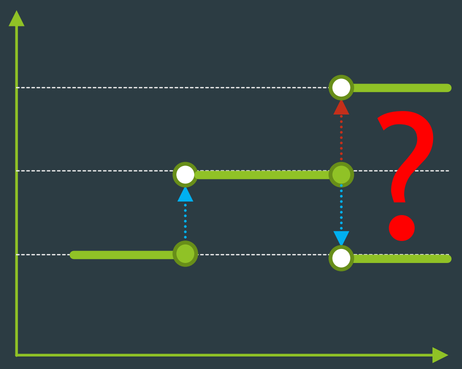
$e = 0$

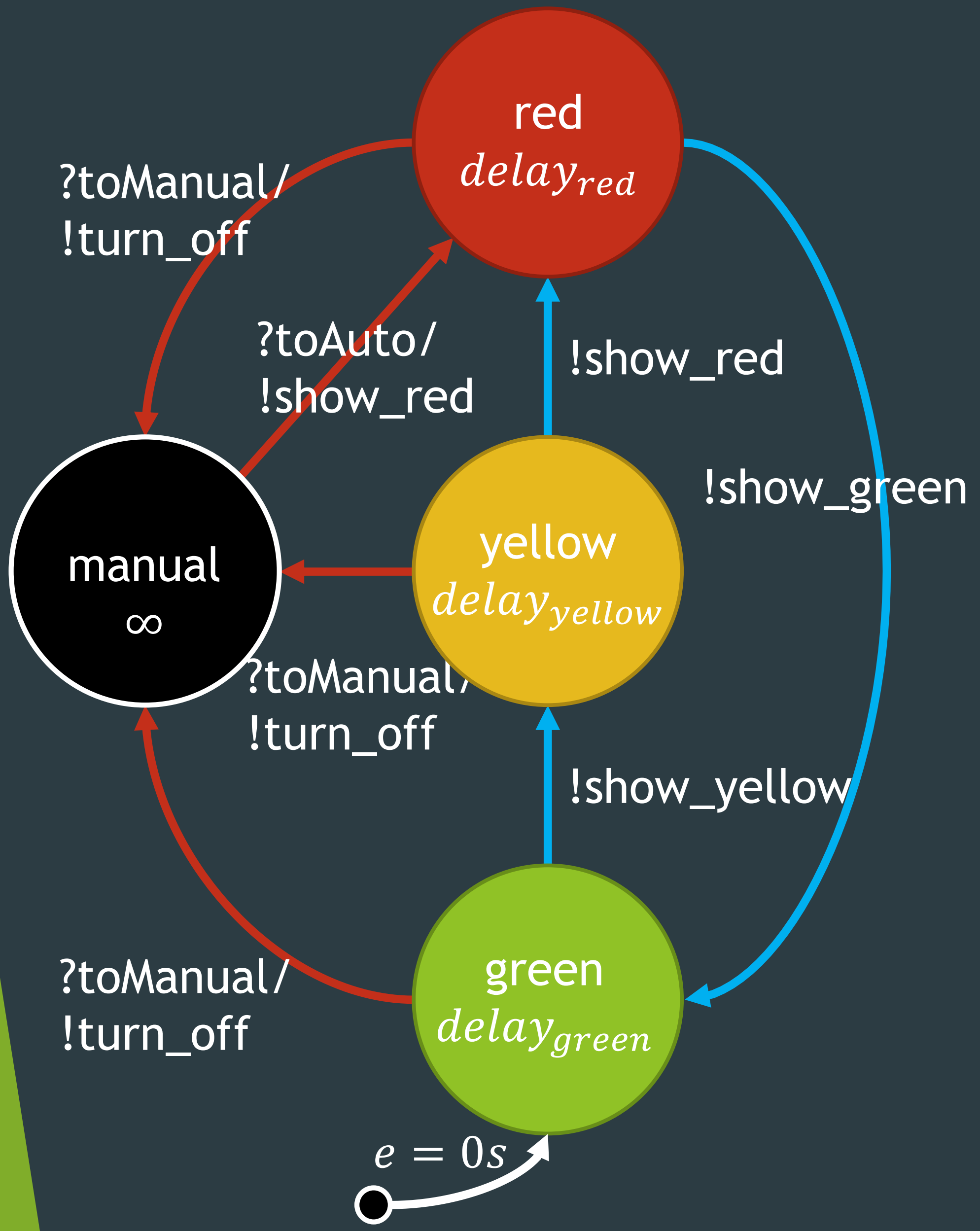


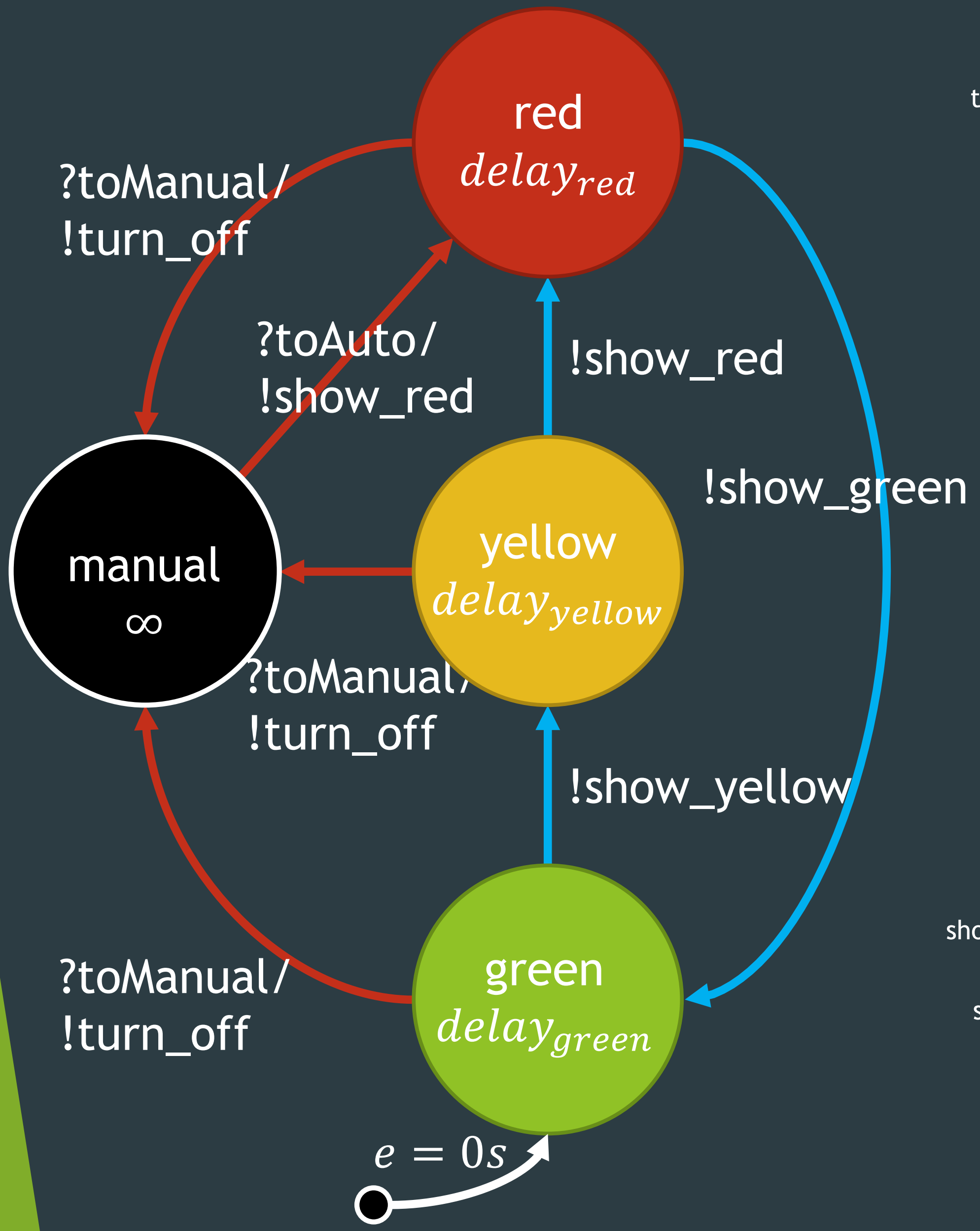
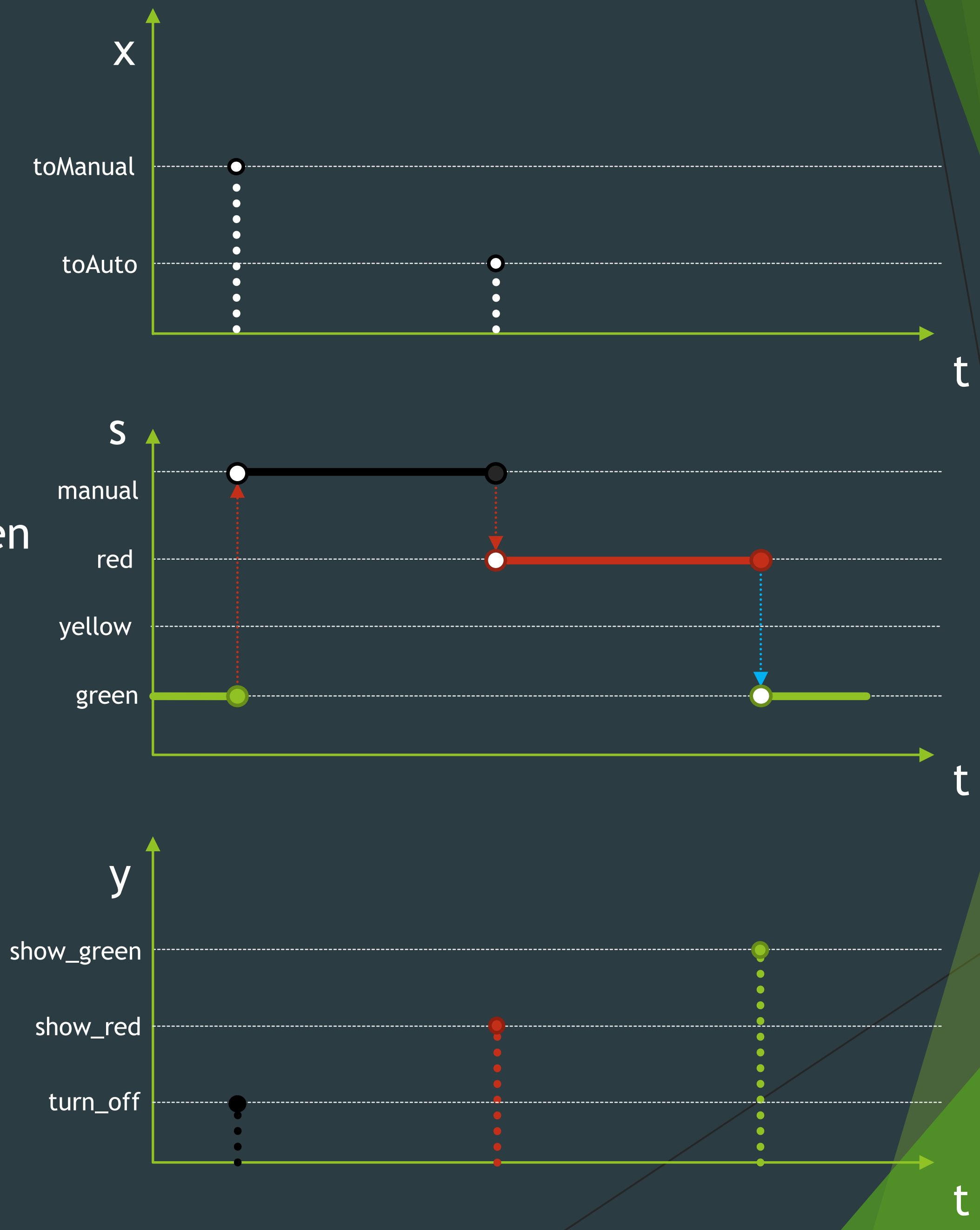
$0 < e < ta(s)$

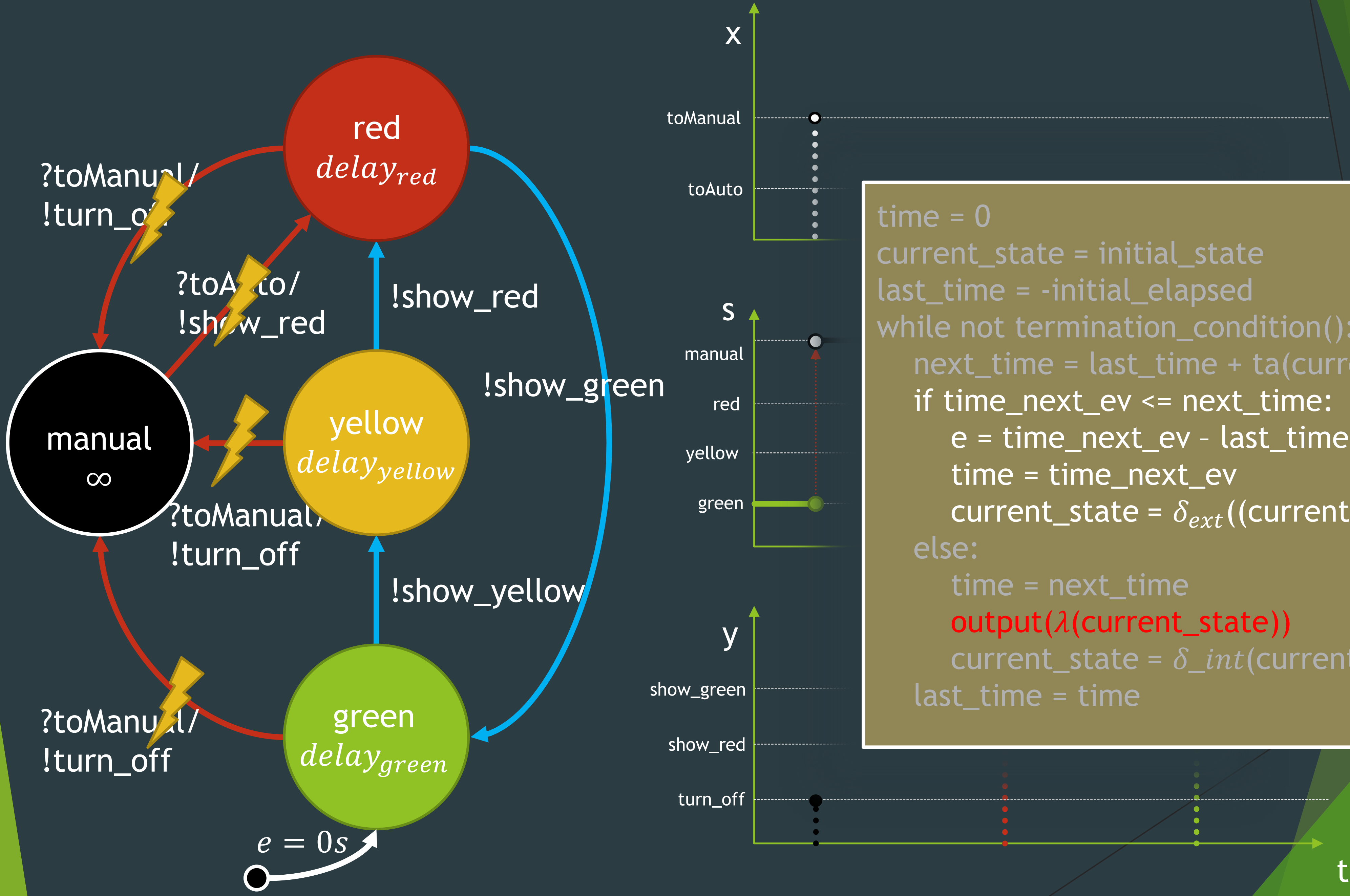


$e = ta(s)$

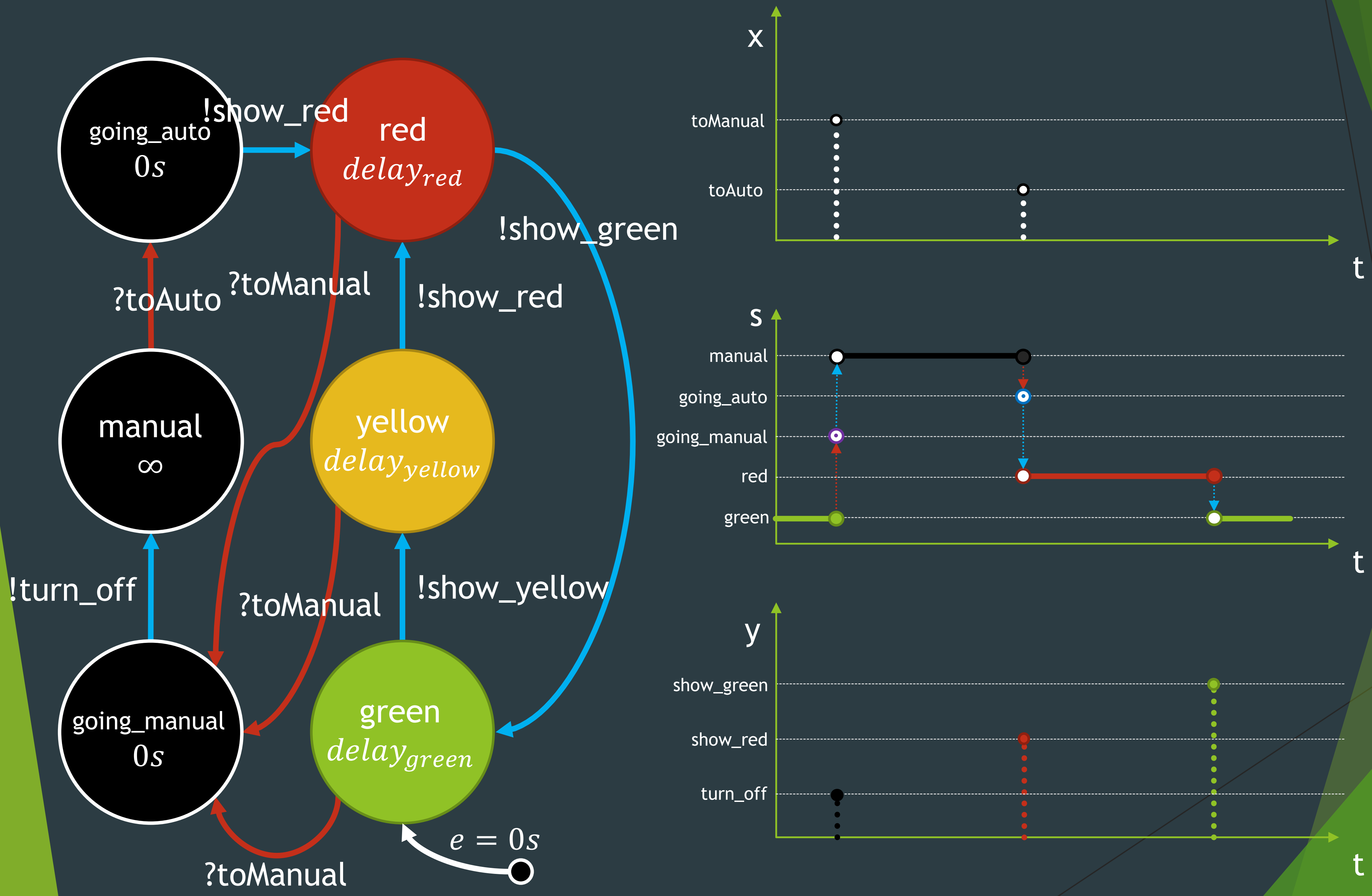



red
$delay_{red}$
?toManual/
!turn_off
?toAuto/
!show_red
!show_red
!show_green
manual
∞
yellow
$delay_{yellow}$
?toManual/
!turn_off
!show_yellow
?toManual/
!turn_off
green
$delay_{green}$
$e = 0s$


red
$delay_{red}$
?toManual/
!turn_off
?toAuto/
!show_red
!show_red
!show_green
manual
∞
yellow
$delay_{yellow}$
?toManual/
!turn_off
!show_yellow
?toManual/
!turn_off
green
$delay_{green}$
$e = 0s$

x
toManual
toAuto
t
s
manual
red
yellow
green
t
y
show_green
show_red
turn_off
t




```
time = 0
current_state = initial_state
last_time = -initial_elapsed
while not termination_condition():
    next_time = last_time + ta(current_state)
    if time_next_ev <= next_time:
        e = time_next_ev - last_time
        time = time_next_ev
        current_state = δ_ext((current_state, e), next_ev)
    else:
        time = next_time
        output(λ(current_state))
        current_state = δ_int(current_state)
    last_time = time
```



going_auto
0s
!show_red
red
$delay_{red}$
!show_green
?toAuto
?toManual
!show_red
manual
∞
yellow
$delay_{yellow}$
!turn_off
?toManual
!show_yellow
going_manual
0s
green
$delay_{green}$
$e = 0s$
?toManual
x
toManual
toAuto
t
s
manual
going_auto
going_manual
red
green
t
y
show_green
show_red
turn_off
t

Full Atomic DEVS Specification

$$M = \langle X, Y, S, q_{init}, \delta_{int}, \delta_{ext}, \lambda, ta \rangle$$

X : *set of input events*

Y : *set of output events*

S : *set of sequential states*

$q_{init} : Q$

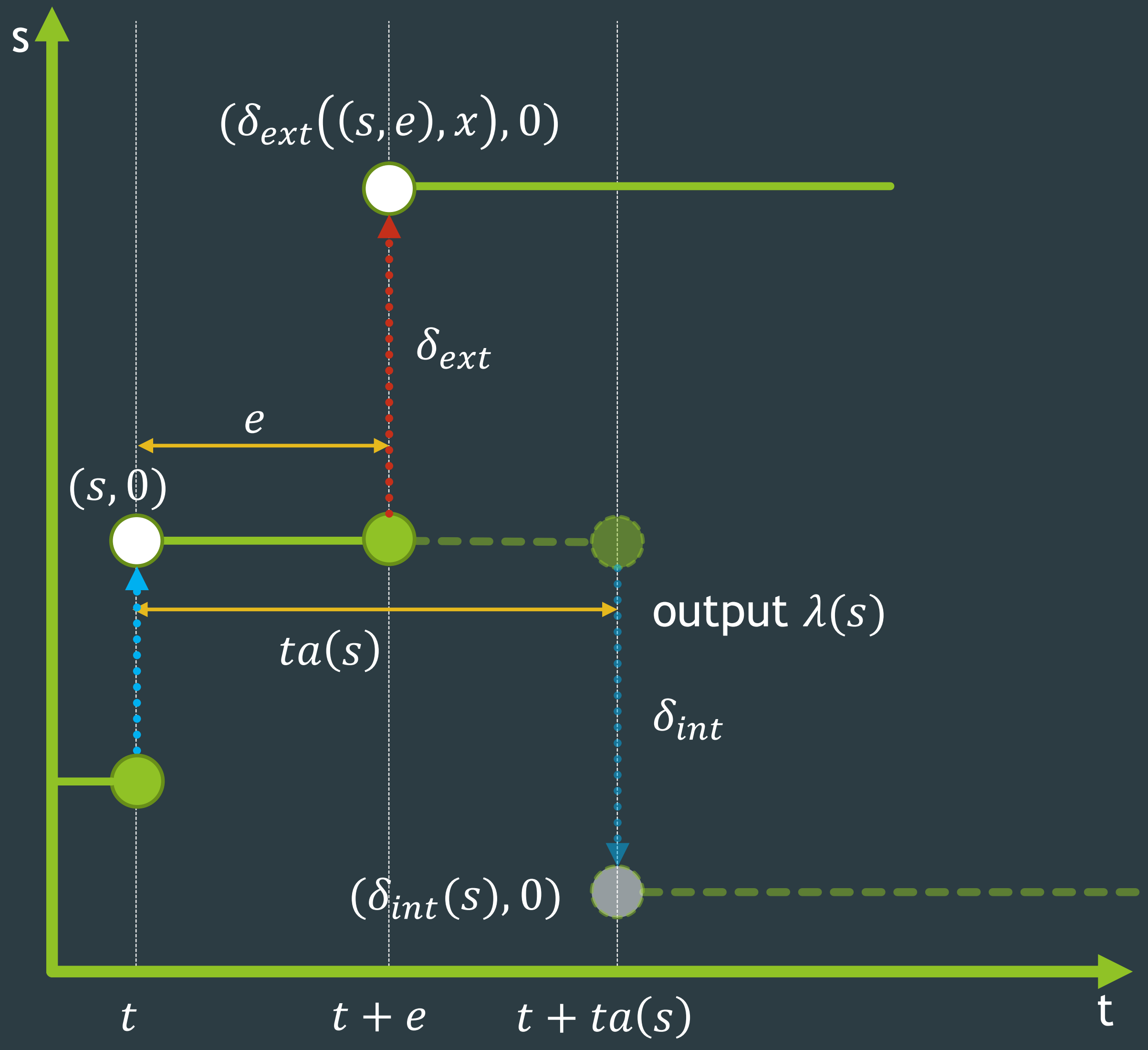
$$Q = \{(s,e) | s \in S, 0 \leq e \leq ta(s)\}$$

$\delta_{int} : S \rightarrow S$

$\delta_{ext} : Q \times X \rightarrow S$

$\lambda : S \rightarrow Y \cup \{\phi\}$

$ta : S \rightarrow \mathbb{R}^{+}_{0,+\infty}$



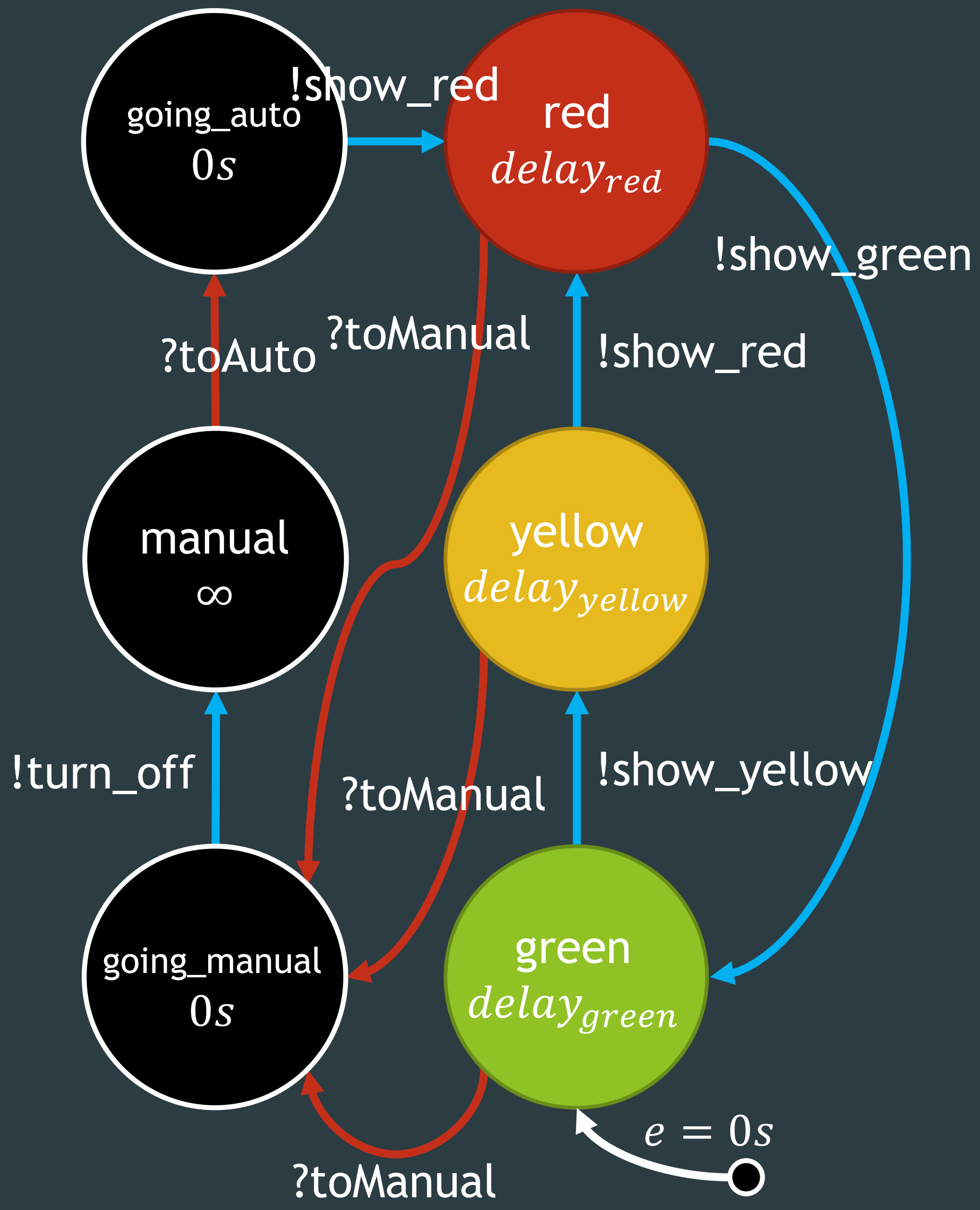
s
$(\delta_{ext}((s,e),x),0)$
δ_{ext}
e
$(s,0)$
output $\lambda(s)$
$ta(s)$
δ_{int}
$(\delta_{int}(s),0)$
t
$t+e$
$t+ta(s)$
t

DEVS Semantics

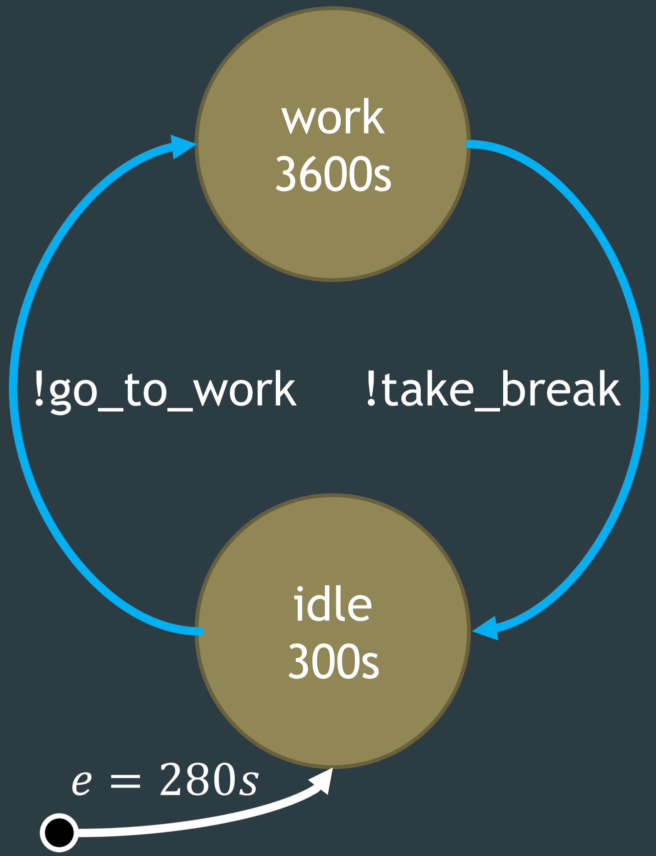
	Operational Semantics	Denotational Semantics
Atomic DEVS	Abstract Simulator	modal Discrete Event Logic L_{DE} [1]
Coupled DEVS	Hierarchical Simulator	Closure under Coupling

[1] Ashvin Radiya and Robert G. Sargent. A logic-based foundation of discrete event modeling and simulation. ACM Transactions on Modeling and Computer Simulation, 1(1):3-51, 1994.

Coupled Models



going_auto
$0s$
!show_red
red
$delay_{red}$
!show_green
?toAuto
?toManual
!show_red
manual
∞
yellow
$delay_{yellow}$
!turn_off
?toManual
!show_yellow
going_manual
$0s$
green
$delay_{green}$
$e = 0s$
?toManual

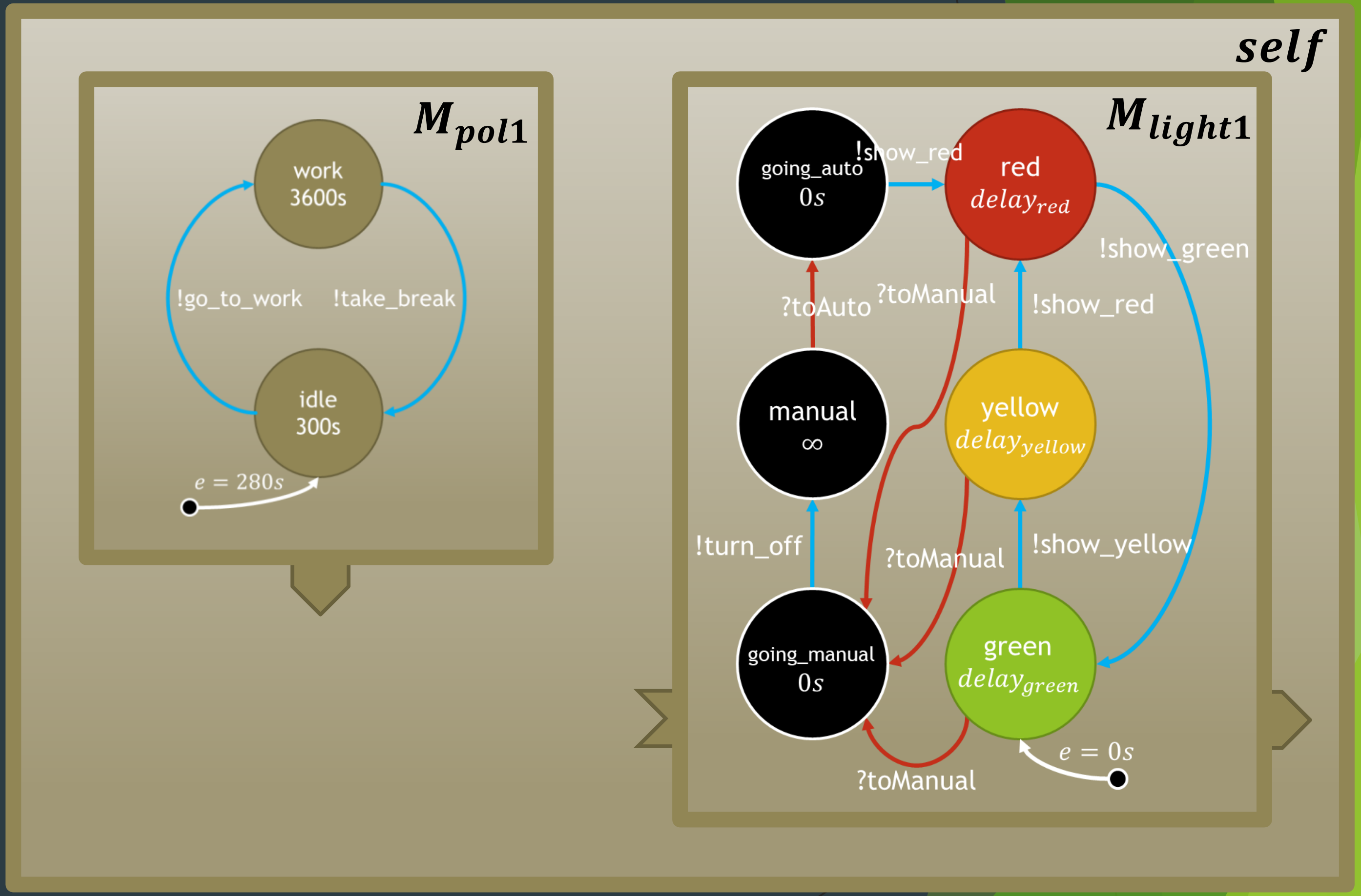

work
3600s
!go_to_work
!take_break
idle
300s
e = 280s


TRAFFIC

$$C = \langle D, MS \rangle$$

$$MS = \{M_i | i \in D\}$$

$$M_i = \langle X_i, Y_i, S_i, q_{init,i}, \delta_{int,i}, \delta_{ext,i}, \lambda_i, ta_i \rangle, \forall\, i \in D$$

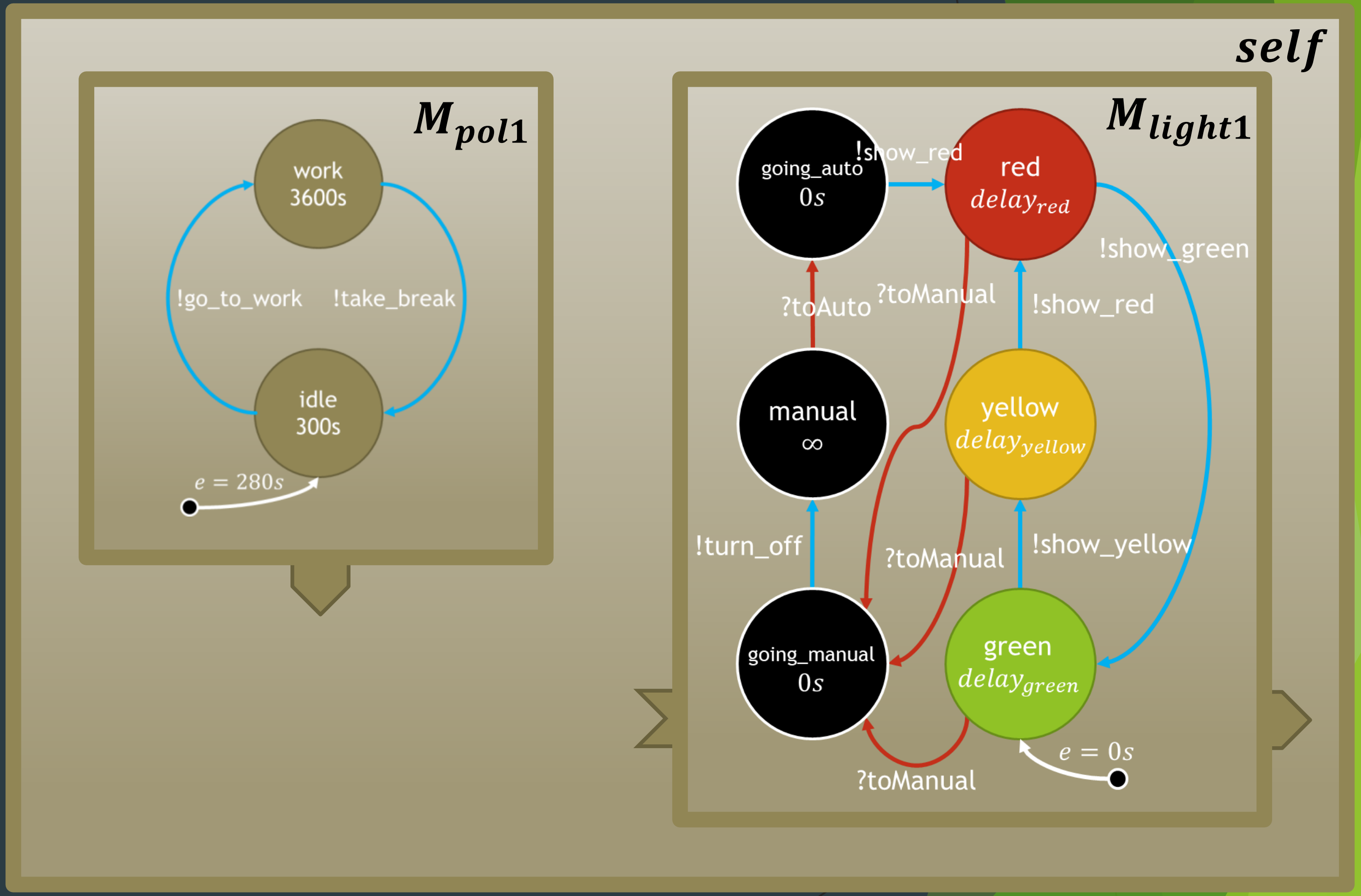


$D = \{light_1, pol_1\}$

$$C = \langle D, MS \rangle$$

$$MS = \{M_i | i \in D\}$$

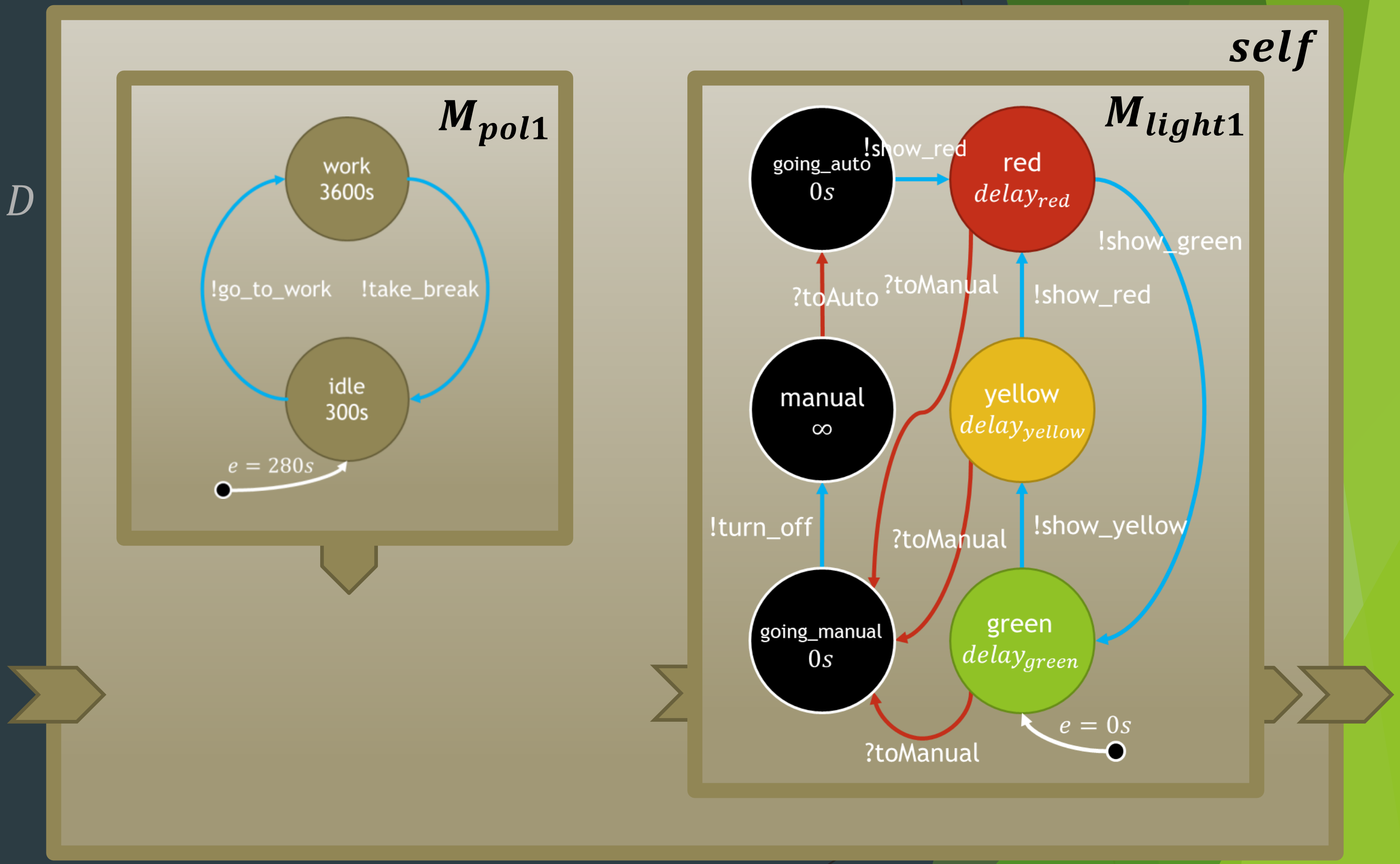
$$M_i = \langle X_i, Y_i, S_i, q_{init,i}, \delta_{int,i}, \delta_{ext,i}, \lambda_i, ta_i \rangle, \forall\, i \in D$$



$$C = \langle X_{self}, Y_{self}, D, MS \rangle$$

$$MS = \{M_i | i \in D\}$$

$$M_i = \langle X_i, Y_i, S_i, q_{init,i}, \delta_{int,i}, \delta_{ext,i}, \lambda_i, ta_i \rangle, \forall\ i \in D$$



$$C = \langle X_{self}, Y_{self}, D, MS, IS \rangle$$

$$MS = \{M_i | i \in D\}$$

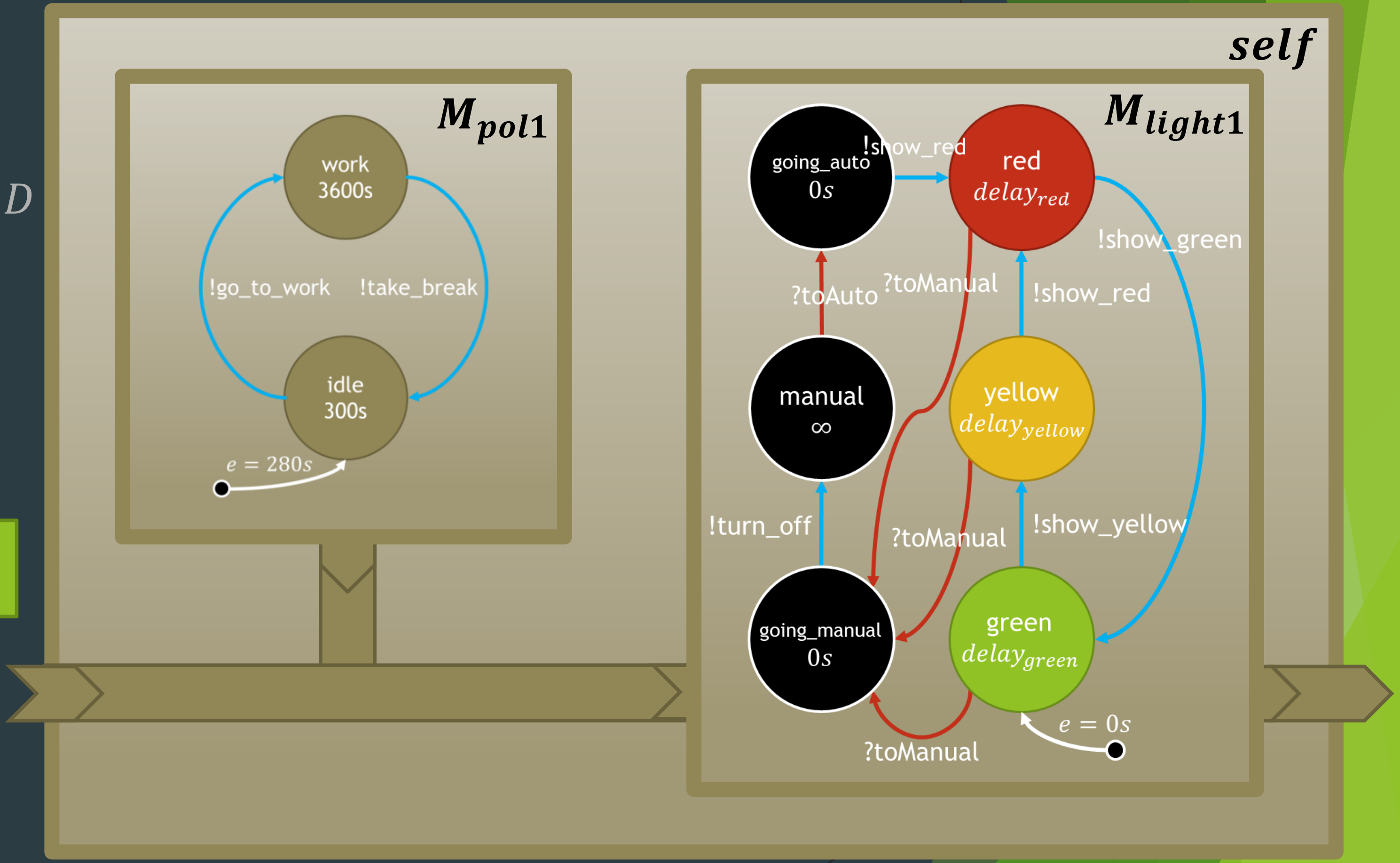
$$M_i = \langle X_i, Y_i, S_i, q_{init,i}, \delta_{int,i}, \delta_{ext,i}, \lambda_i, ta_i \rangle, \forall\, i \in D$$

$$IS = \{I_i | i \in D \cup \{self\}\}$$

$$\forall\, i \in D \cup \{self\} : I_i \subseteq D \cup \{self\}$$

$$\forall\, i \in D \cup \{self\} : i \notin I_i$$

I_i :Influencees of i



$$C = \langle X_{self}, Y_{self}, D, MS, IS \rangle$$

$$MS = \{M_i | i \in D\}$$

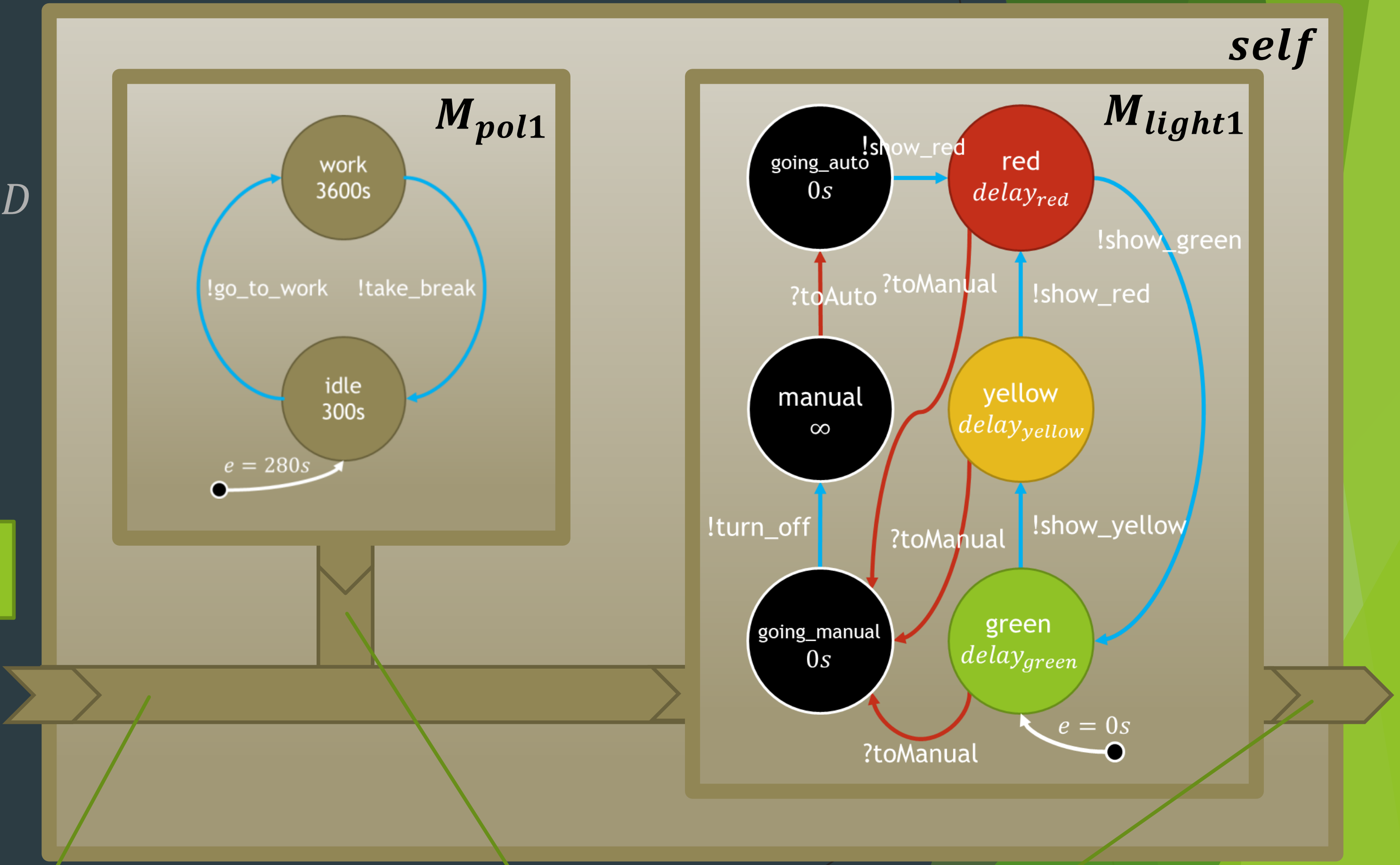
$$M_i = \langle X_i, Y_i, S_i, q_{init,i}, \delta_{int,i}, \delta_{ext,i}, \lambda_i, ta_i \rangle, \forall\, i \in D$$

$$IS = \{I_i | i \in D \cup \{self\}\}$$

$$\forall\, i \in D \cup \{self\} : I_i \subseteq D \cup \{self\}$$

$$\forall\, i \in D \cup \{self\} : i \notin I_i$$

I_i :Influencees of i



$I_{self} = \{light1\}$

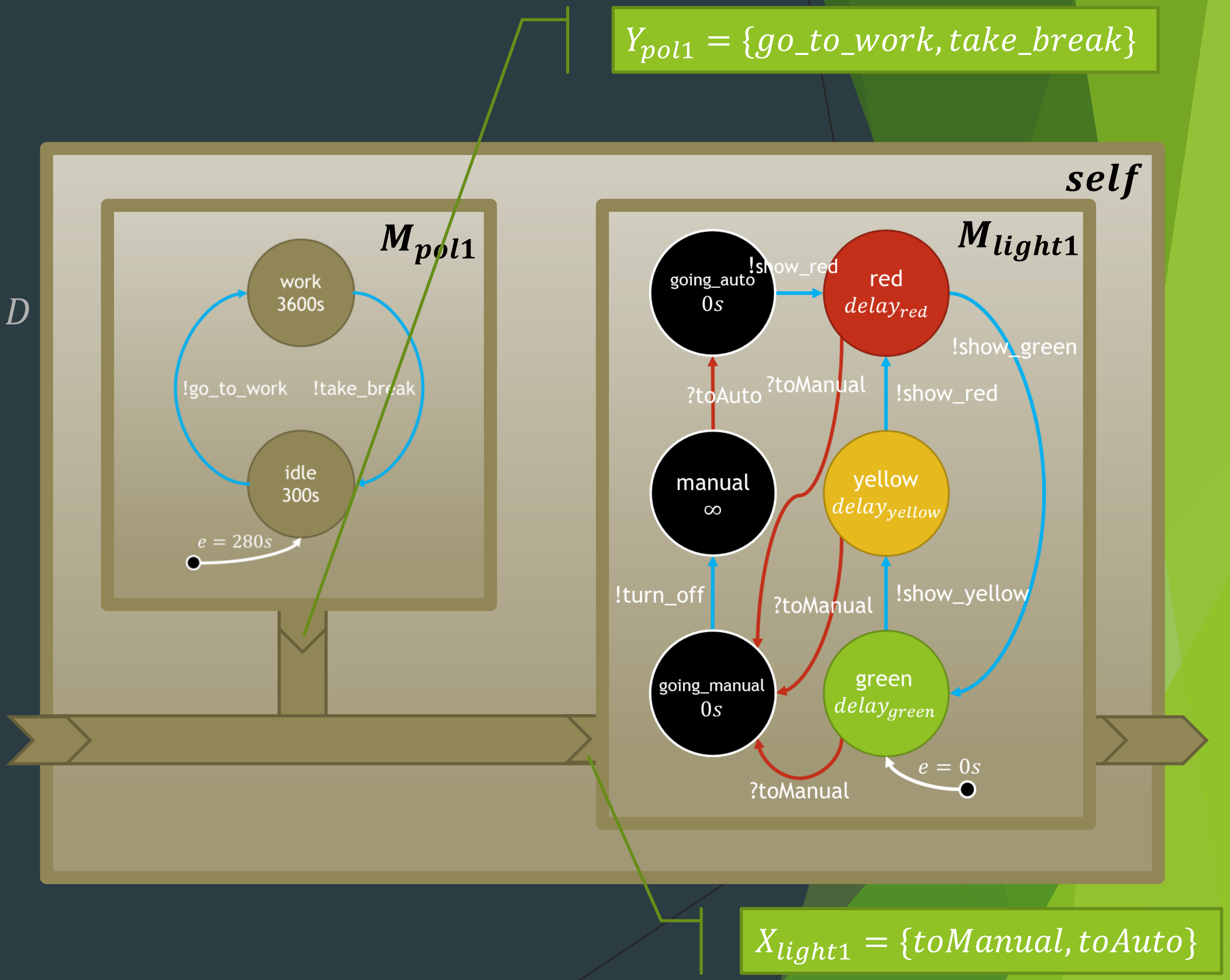
$I_{pol1} = \{light1\}$

$I_{light1} = \{self\}$

$$C = \langle X_{self}, Y_{self}, D, MS, IS, ZS \rangle$$
$$MS = \{M_i | i \in D\}$$
$$M_i = \langle X_i, Y_i, S_i, q_{init,i}, \delta_{int,i}, \delta_{ext,i}, \lambda_i, ta_i \rangle, \forall\, i \in D$$

$$IS = \{I_i | i \in D \cup \{self\}\}$$
$$\forall\, i \in D \cup \{self\} : I_i \subseteq D \cup \{self\}$$
$$\forall\, i \in D \cup \{self\} : i \notin I_i$$

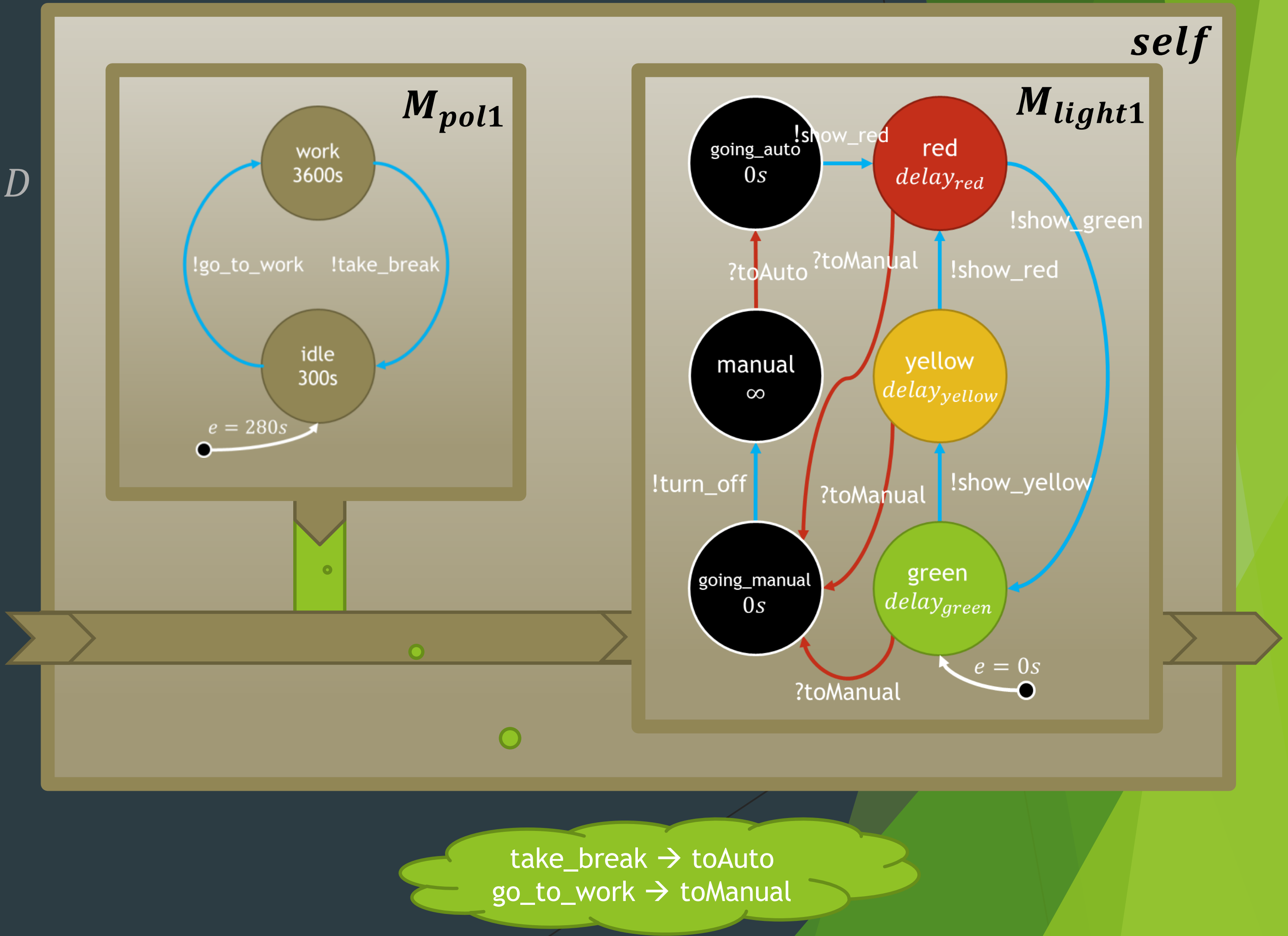
$$ZS = \{Z_{i,j} | i \in D \cup \{self\}, j \in I_i\}$$
$$Z_{self,j} : X_{self} \rightarrow X_j, \forall\, j \in D$$
$$Z_{i,self} : Y_i \rightarrow Y_{self}, \forall\, i \in D$$
$$Z_{i,j} : Y_i \rightarrow X_j, \forall\, i, j \in D$$



$$C = \langle X_{self}, Y_{self}, D, MS, IS, ZS \rangle$$
$$MS = \{M_i | i \in D\}$$
$$M_i = \langle X_i, Y_i, S_i, q_{init,i}, \delta_{int,i}, \delta_{ext,i}, \lambda_i, ta_i \rangle, \forall\, i \in D$$

$$IS = \{I_i | i \in D \cup \{self\}\}$$
$$\forall\, i \in D \cup \{self\} : I_i \subseteq D \cup \{self\}$$
$$\forall\, i \in D \cup \{self\} : i \notin I_i$$

$$ZS = \{Z_{i,j} | i \in D \cup \{self\}, j \in I_i\}$$
$$Z_{self,j} : X_{self} \rightarrow X_j, \forall\, j \in D$$
$$Z_{i,self} : Y_i \rightarrow Y_{self}, \forall\, i \in D$$
$$Z_{i,j} : Y_i \rightarrow X_j, \forall\, i,j \in D$$



$$C = \langle X_{self}, Y_{self}, D, MS, IS, ZS, select \rangle$$

$$MS = \{M_i | i \in D\}$$

$$M_i = \langle X_i, Y_i, S_i, q_{init,i}, \delta_{int,i}, \delta_{ext,i}, \lambda_i, ta_i \rangle, \forall\, i \in D$$

$$IS = \{I_i | i \in D \cup \{self\}\}$$

$$\forall\, i \in D \cup \{self\} : I_i \subseteq D \cup \{self\}$$

$$\forall\, i \in D \cup \{self\} : i \notin I_i$$

$$ZS = \{Z_{i,j} | i \in D \cup \{self\}, j \in I_i\}$$

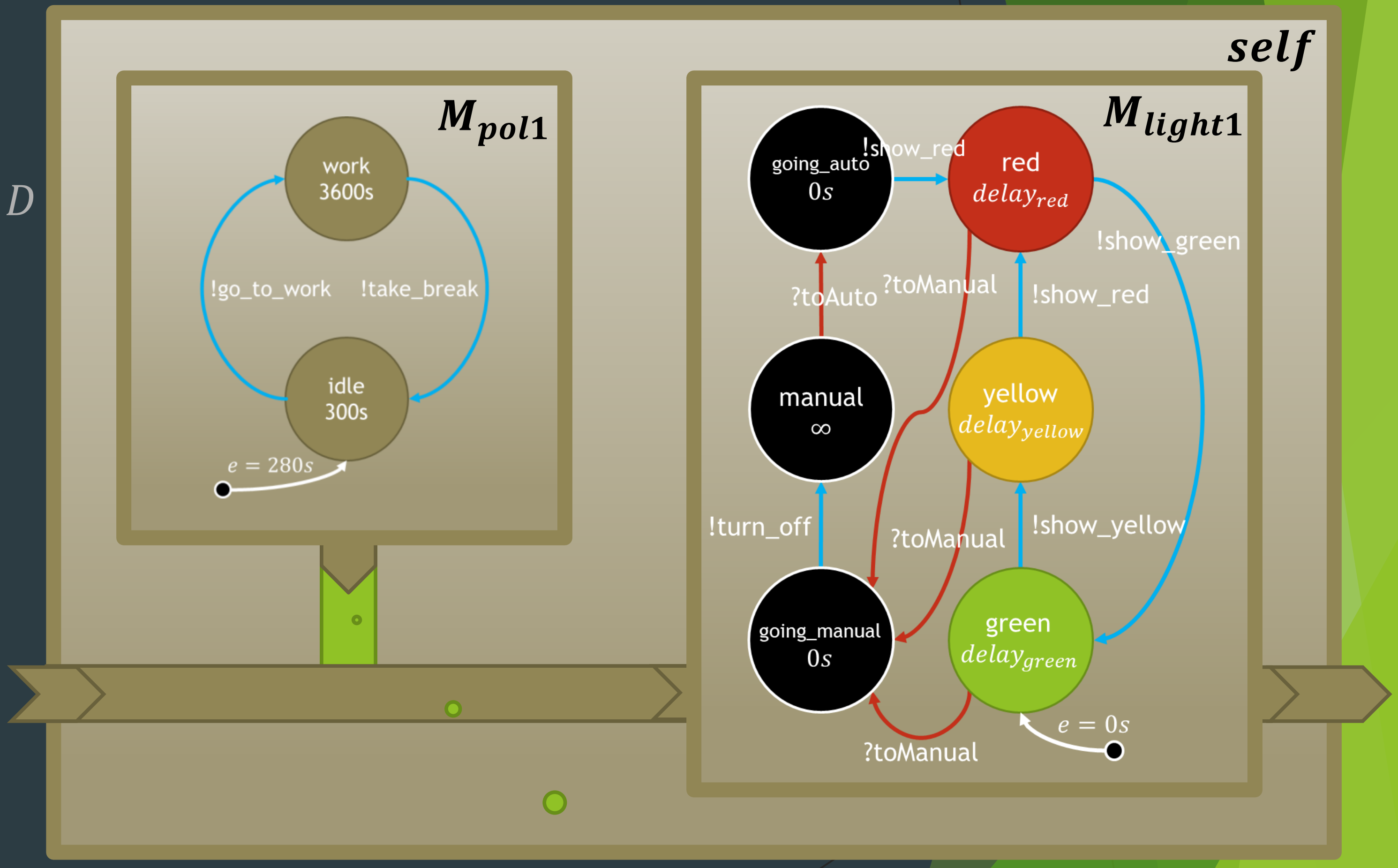
$$Z_{self,j} : X_{self} \rightarrow X_j, \forall\, j \in D$$

$$Z_{i,self} : Y_i \rightarrow Y_{self}, \forall\, i \in D$$

$$Z_{i,j} : Y_i \rightarrow X_j, \forall\, i, j \in D$$

$$select : 2^D \rightarrow D$$

$$\forall\, E \subseteq D, E \neq \emptyset : select(E) \in E$$



take_break → toAuto
go_to_work → toManual

Concrete Syntax

trafficlight_system.py

```
from pypdevs.DEVS import *

from trafficlight import TrafficLight
from policeman import Policeman

def translate(in_evt):
    mapping = {“take_break”: “toAuto”,
                     “go_to_work”: “toManual”}
    return mapping[in_evt]

class TrafficLightSystem(CoupledDEVS):
    def __init__(self):
        CoupledDEVS.__init__(self, “system”)
        self.light = self.addSubModel(TrafficLight(\
                q_init_pol1 = (“idle”, 280), q_init_light1 = (“green”, 0),
                delay_red = 60, delay_yellow = 3, delay_green = 57))
        self.police = self.addSubModel(Policeman())
        self.connectPorts(self.police.out, self.light.interrupt, translate)

    def select(self, immlist):
        if self.police in immlist:
            return self.police
        else:
            return self.light
```

__ Current Time: 0.00 ____

INITIAL CONDITIONS in model <system.light>

Initial State: green

Next scheduled internal transition at time 57.00

INITIAL CONDITIONS in model <system.policeman>

Initial State: idle

Next scheduled internal transition at time 20.00

```
__  Current Time:      20.00 ____________________________________________

    EXTERNAL TRANSITION in model <system.light>
        Input Port Configuration:
            port <interrupt>:
                toManual
        New State: going_manual
        Next scheduled internal transition at time 20.0

    INTERNAL TRANSITION in model <system.policeman>
        New State: working
        Output Port Configuration:
            port <output>:
                go_to_work
        Next scheduled internal transition at time 3620.00
```

__ Current Time: 20.00 ____

```
INTERNAL TRANSITION in model <system.light>
    Output Port Configuration:
        port <observer>:
            turn_off
    New State: manual
    Next scheduled internal transition at time inf
```

```
__  Current Time:     3620.00 _______________________________________

        EXTERNAL TRANSITION in model <system.light>
                Input Port Configuration:
                        port <interrupt>:
                                toAuto
                New State: going_auto
                Next scheduled internal transition at time 3620.00

        INTERNAL TRANSITION in model <system.policeman>
                New State: idle
                Output Port Configuration:
                        port <output>:
                                take_break
                Next scheduled internal transition at time 3920.00
```

__ Current Time: 3620.00 ____

INTERNAL TRANSITION in model <system.light>

Output Port Configuration:

port <observer>:

show_red

New State: red

Next scheduled internal transition at time 3680.00

```
__  Current Time:     3620.00 ____________________________________________

    EXTERNAL TRANSITION in model <system.light>
        Input Port Configuration:
            port <interrupt>:
                toAuto
        New State: going_auto
        Next scheduled internal transition at time 3620.00

    INTERNAL TRANSITION in model <system.policeman>
        New State: idle
        Output Port Configuration:
            port <output>:
                take_break
        Next scheduled internal transition at time 3920.00
```

```
__  Current Time:     3920.00 ________________________________________

      CONFLICT between models:
              <system.light>
            * <system.policeman>

      EXTERNAL TRANSITION in model <system.light>
            Input Port Configuration:
                  port <interrupt>:
                        toManual
            New State: going_manual
            Next scheduled internal transition at time 3920.00

      INTERNAL TRANSITION in model <system.policeman>
            New State: work
            Output Port Configuration:
                  port <output>:
                        go_to_work
            Next scheduled internal transition at time 7520.00
```

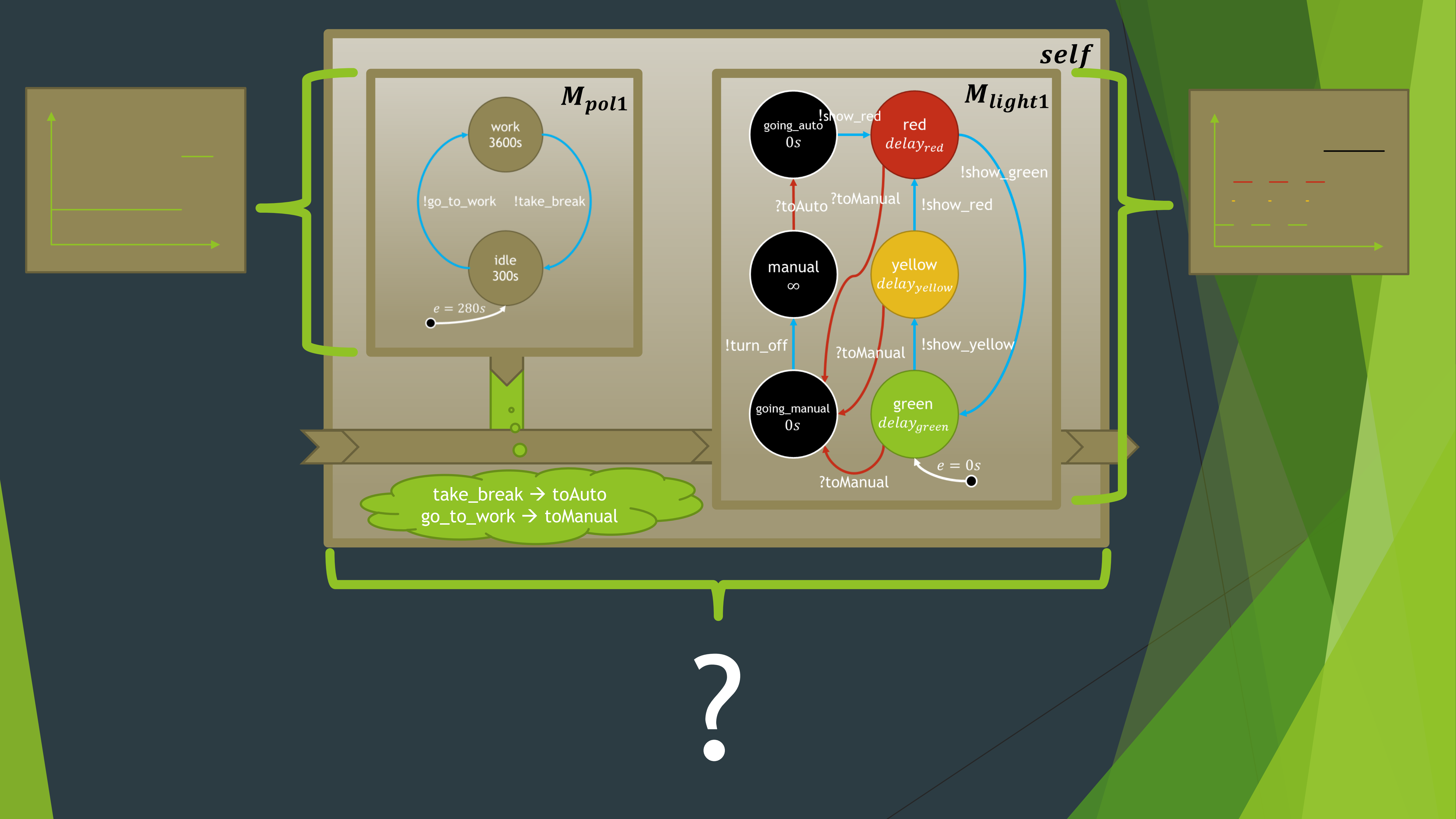
Closure Under Coupling

Denotational semantics for Coupled DEVS models

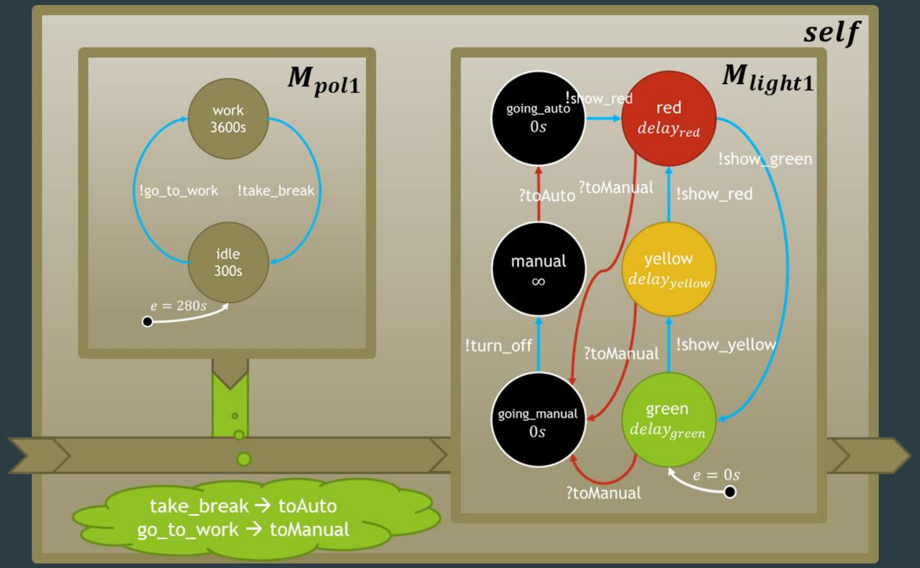
DEVS Semantics

	Operational Semantics	Denotational Semantics
Atomic DEVS	Abstract Simulator	modal Discrete Event Logic L_{DE} [1]
Coupled DEVS	Hierarchical Simulator	Closure under Coupling

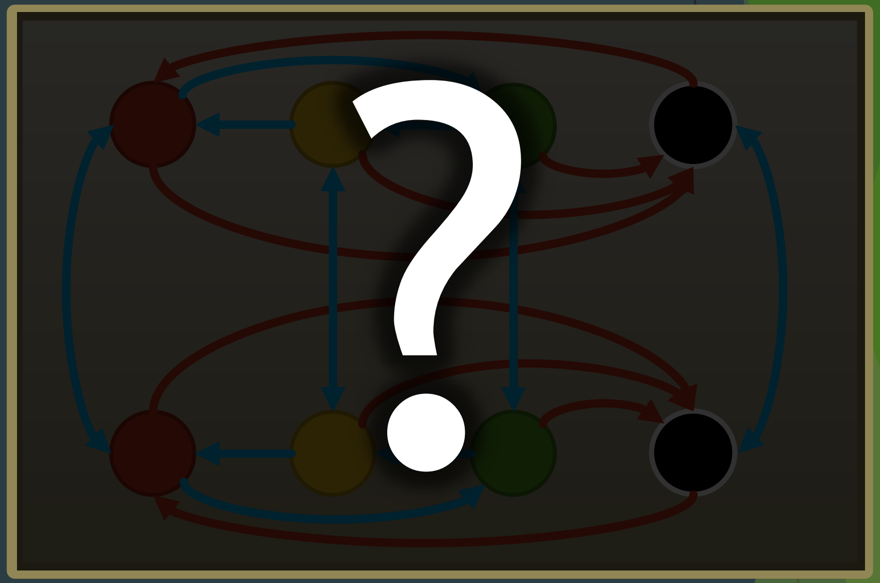
[1] Ashvin Radiya and Robert G. Sargent. A logic-based foundation of discrete event modeling and simulation. ACM Transactions on Modeling and Computer Simulation, 1(1):3-51, 1994.



self
M_{pol1}
work
3600s
idle
300s
!go_to_work
!take_break
$e = 280s$
M_{light1}
going_auto
$0s$
red
$delay_{red}$
manual
∞
yellow
$delay_{yellow}$
going_manual
$0s$
green
$delay_{green}$
!show_red
!show_green
?toAuto
?toManual
!show_red
!turn_off
?toManual
!show_yellow
?toManual
$e = 0s$
take_break → toAuto
go_to_work → toManual
?

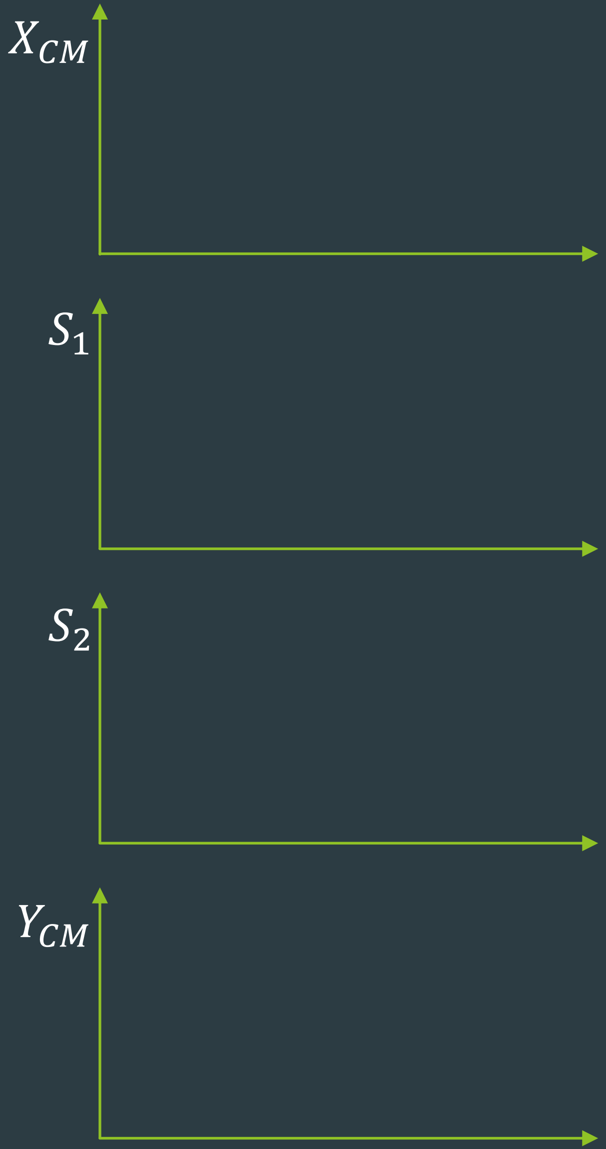


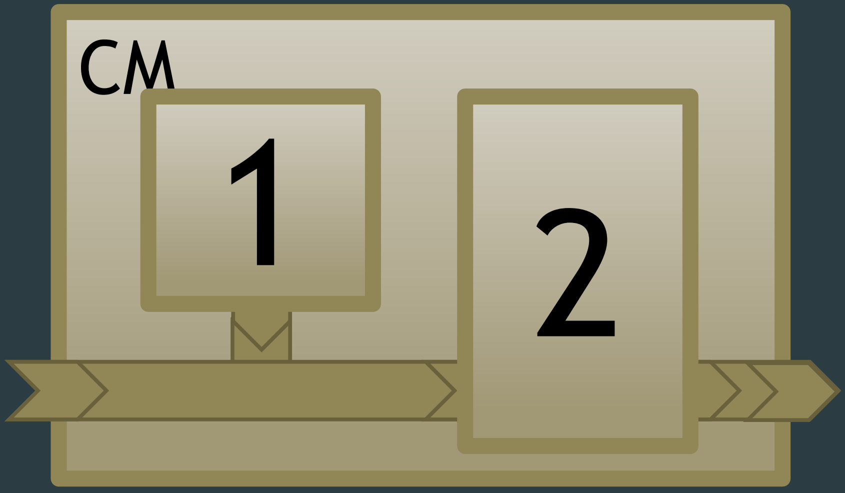
flatten



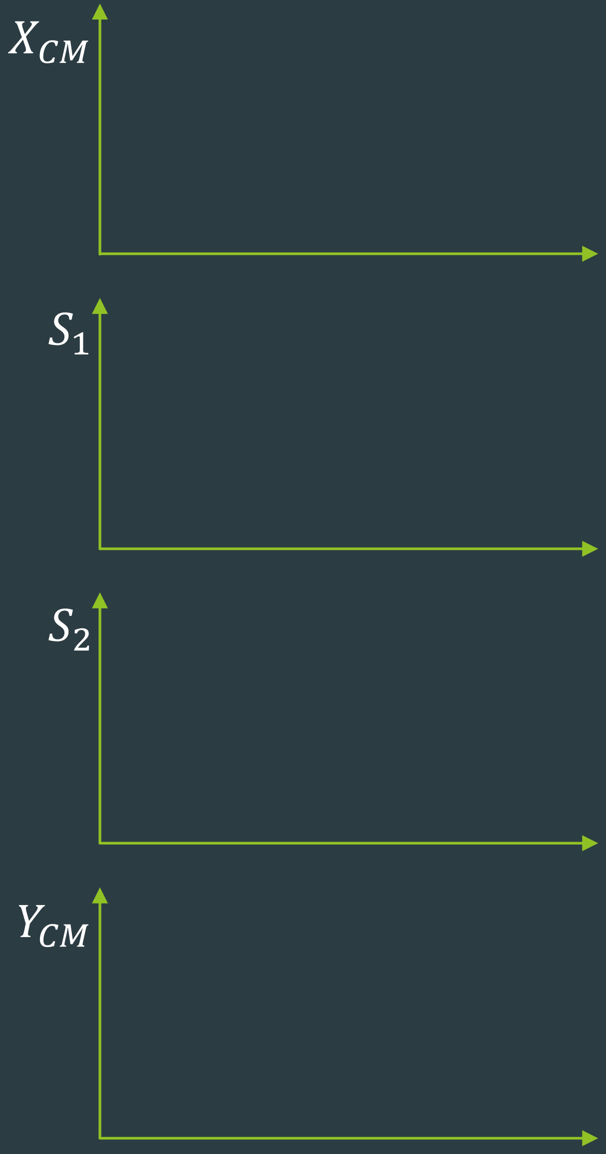
$$CM = \langle X_{self}, Y_{self}, D, MS, IS, ZS \rangle$$

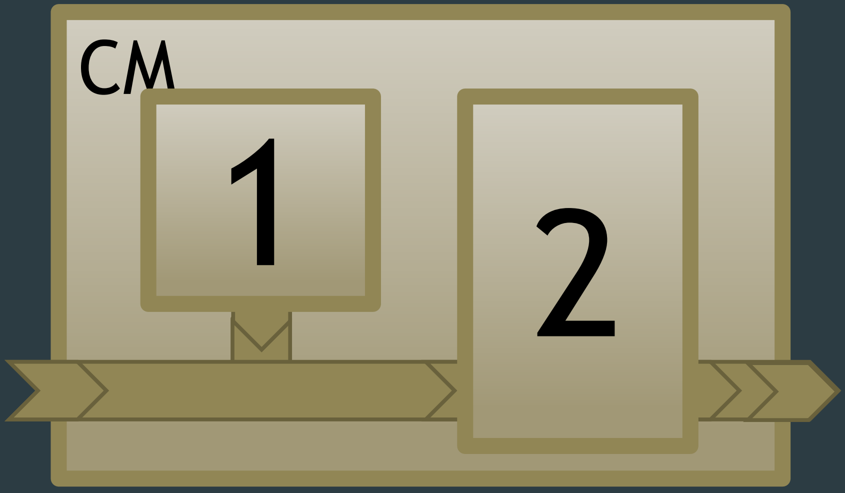
$$flatten(CM) = \langle X, Y, S, q_{init}, \delta_{int}, \delta_{ext}, \lambda, ta \rangle$$





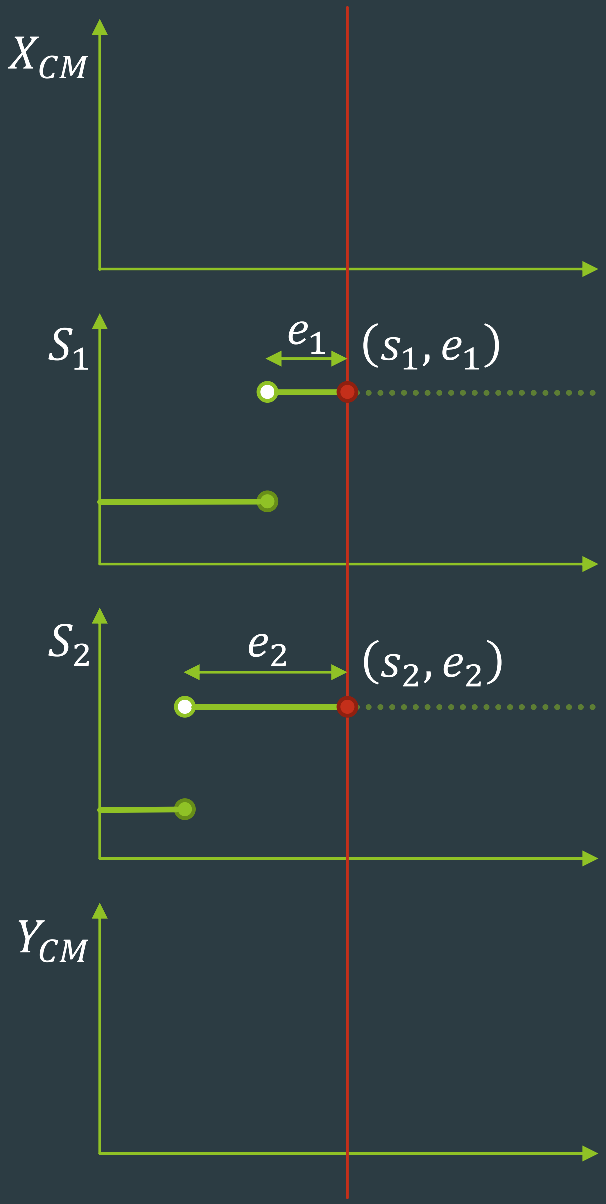
$$flatten(CM) = \langle X, Y, S, q_{init}, \delta_{int}, \delta_{ext}, \lambda, ta \rangle$$

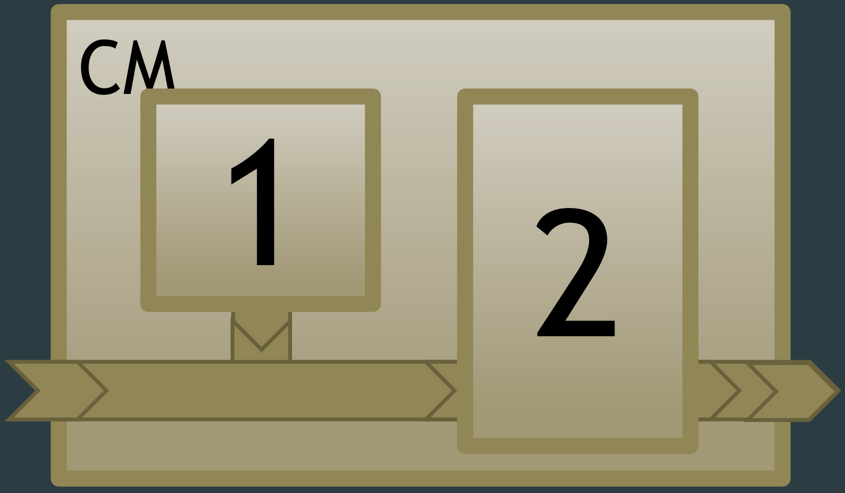




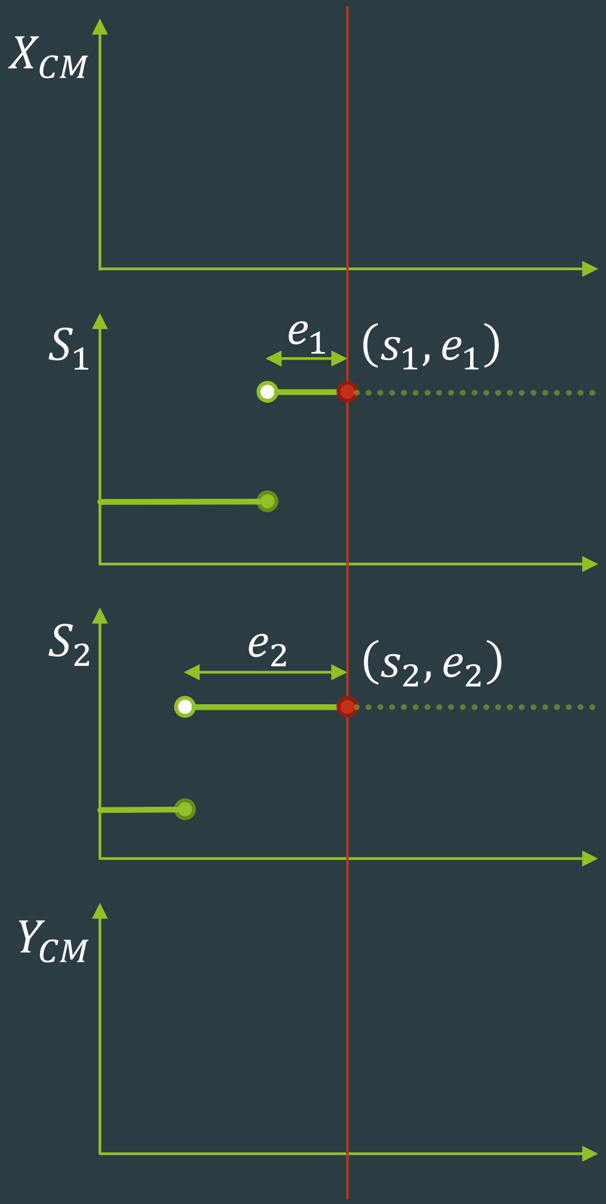
$$flatten(CM) = \langle X, Y, S, q_{init}, \delta_{int}, \delta_{ext}, \lambda, ta \rangle$$

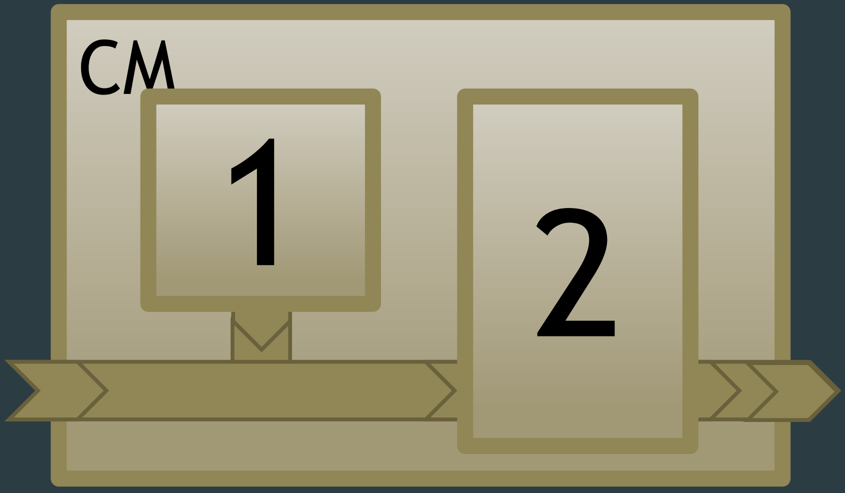
$$X = X_{CM}$$
$$Y = Y_{CM}$$





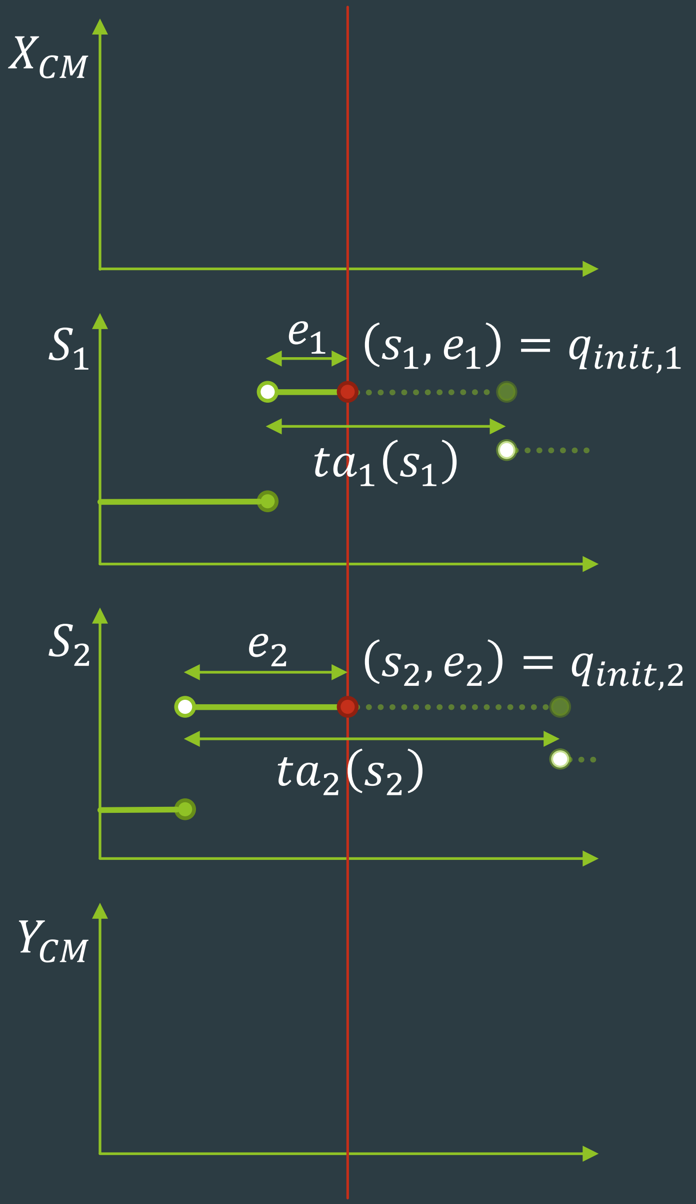
$$flatten(CM) = \langle X, Y, S, q_{init}, \delta_{int}, \delta_{ext}, \lambda, ta \rangle$$

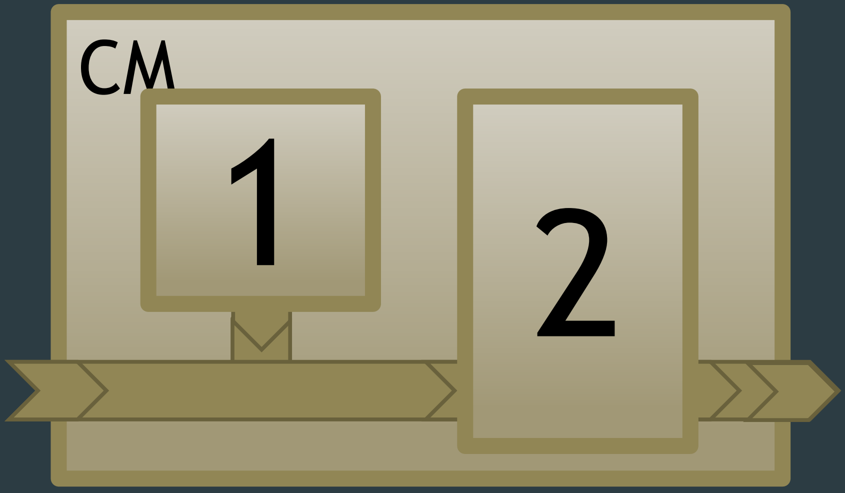




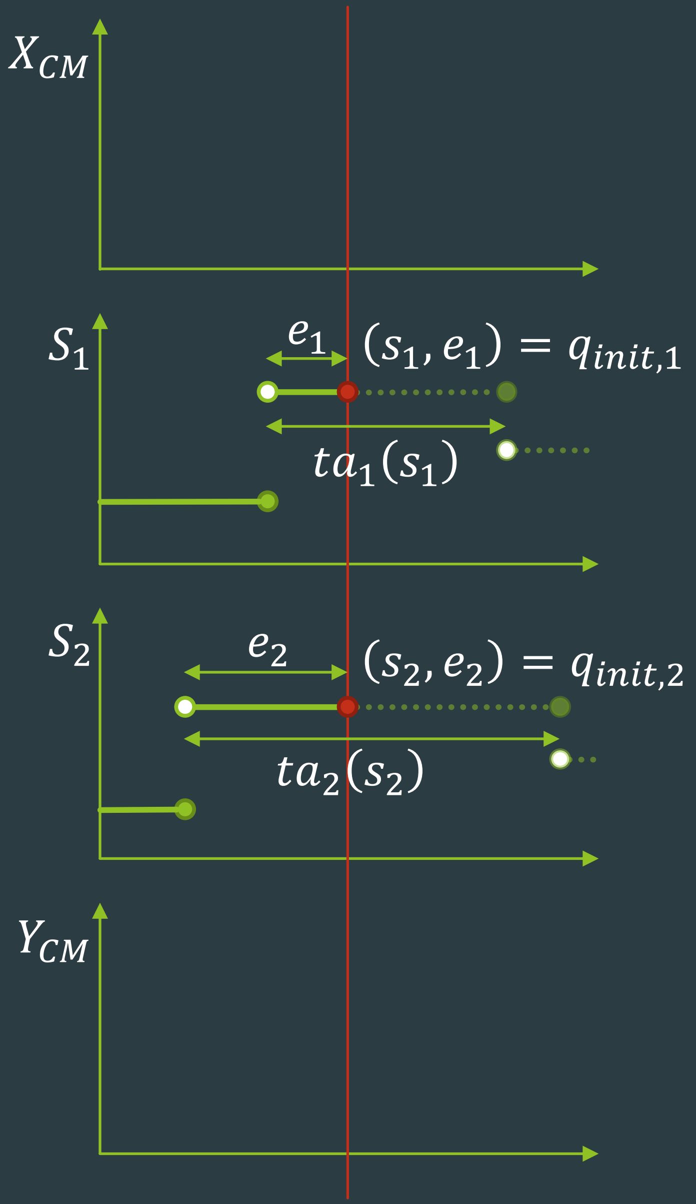
$$flatten(CM) = \langle X, Y, S, q_{init}, \delta_{int}, \delta_{ext}, \lambda, ta \rangle$$

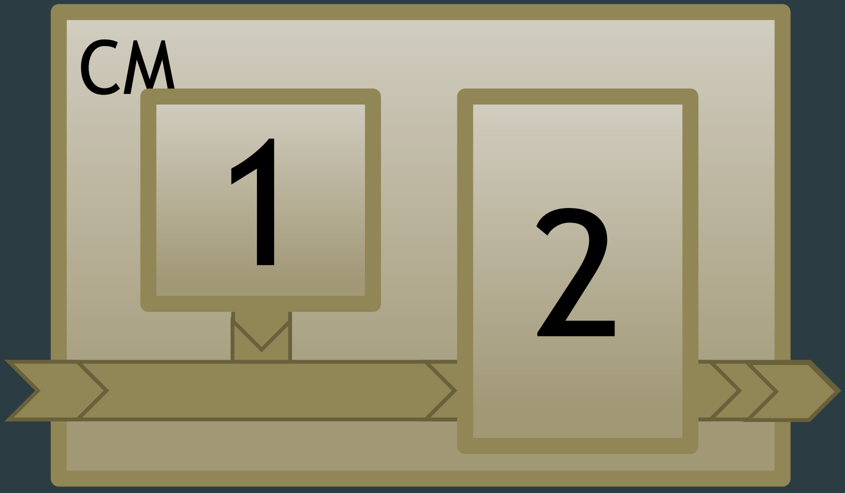
$$S = \times_{i \in D} Q_i$$





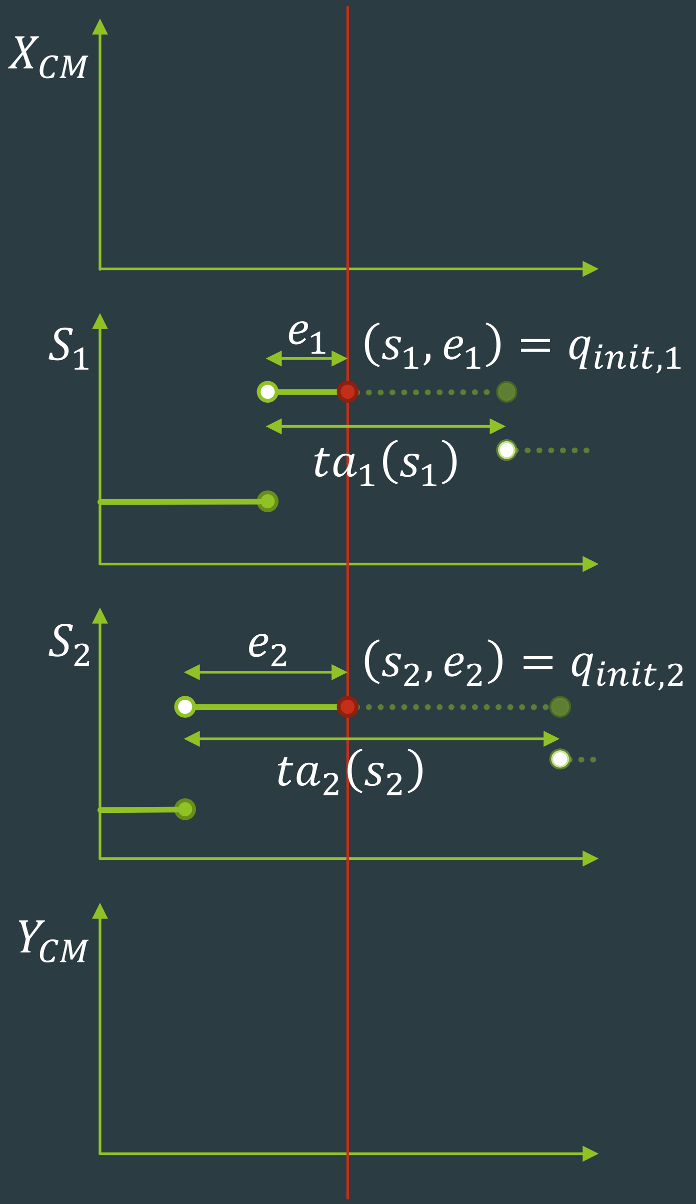
$$flatten(CM) = \langle X, Y, S, q_{init}, \delta_{int}, \delta_{ext}, \lambda, ta \rangle$$

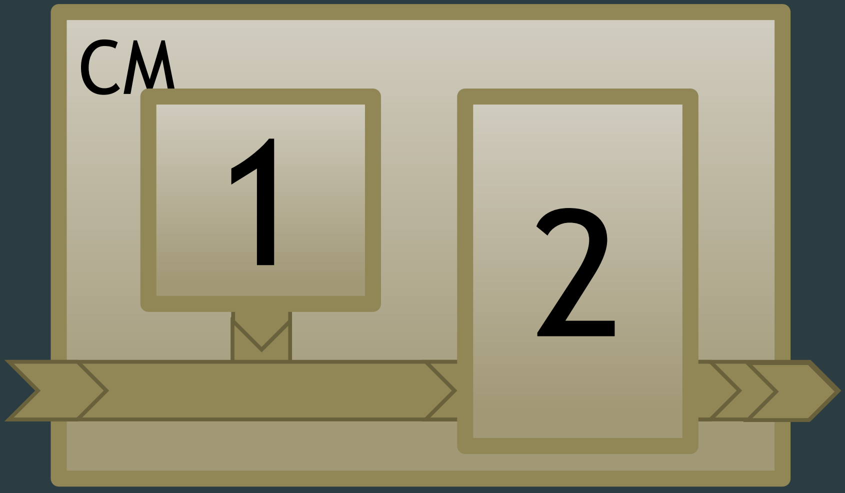




$$flatten(CM) = \langle X, Y, S, q_{init}, \delta_{int}, \delta_{ext}, \lambda, ta \rangle$$

$$q_{init} = (s_{init}, e_{init})$$

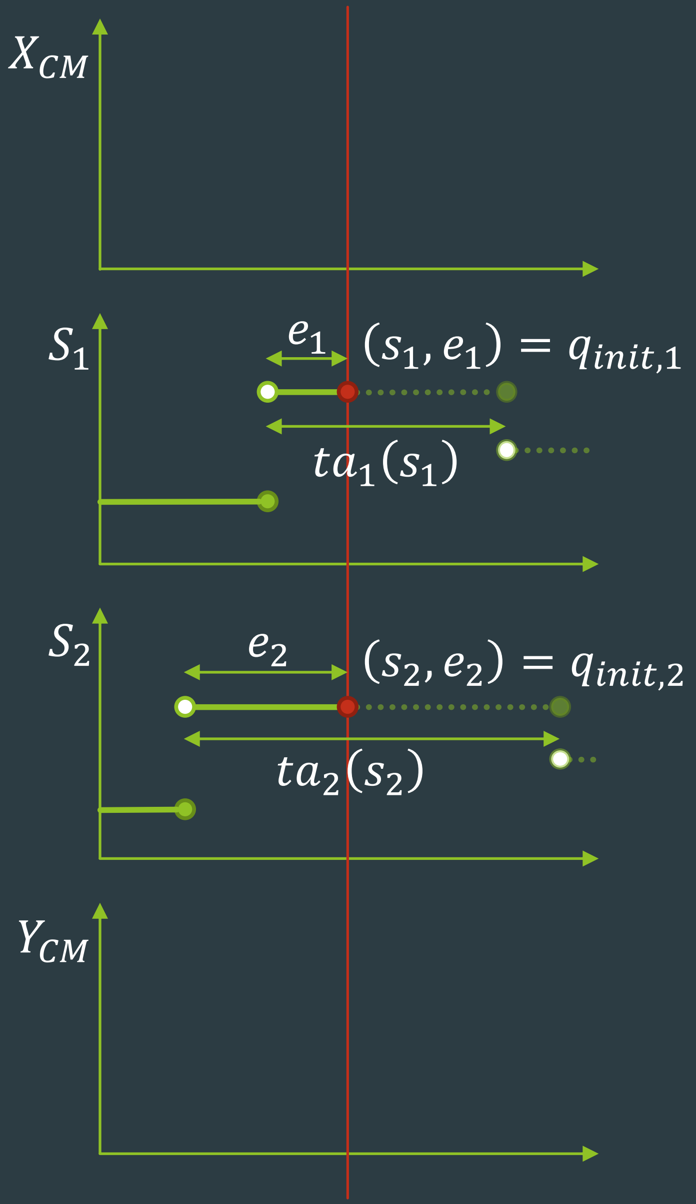


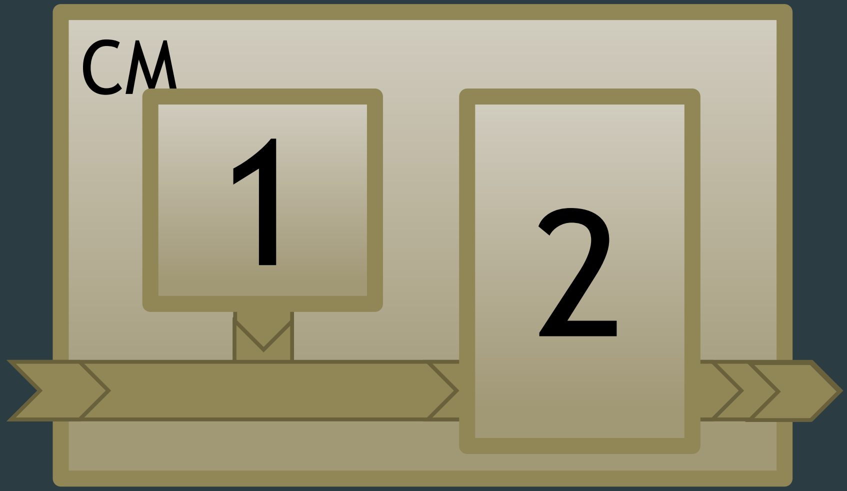


$$flatten(CM) = \langle X, Y, S, q_{init}, \delta_{int}, \delta_{ext}, \lambda, ta \rangle$$

$$q_{init} = (s_{init}, e_{init})$$

$$s_{init} = (\ldots, (s_{init,i}, e_{init,i} - e_{init}), \ldots)$$



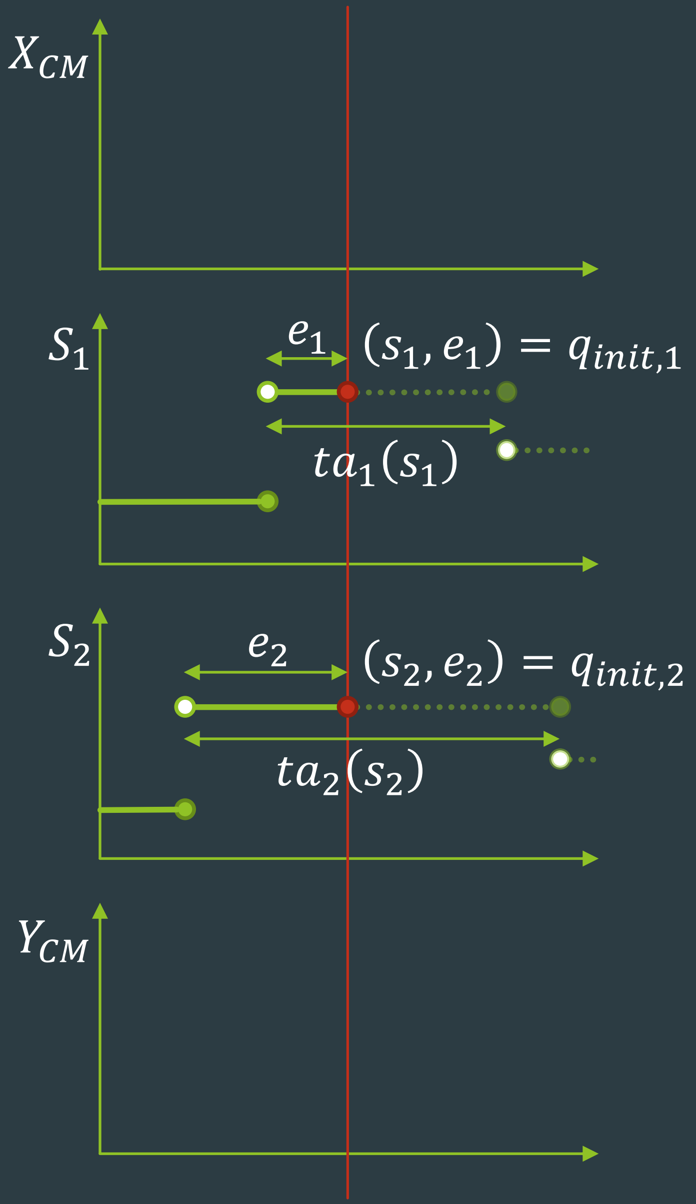


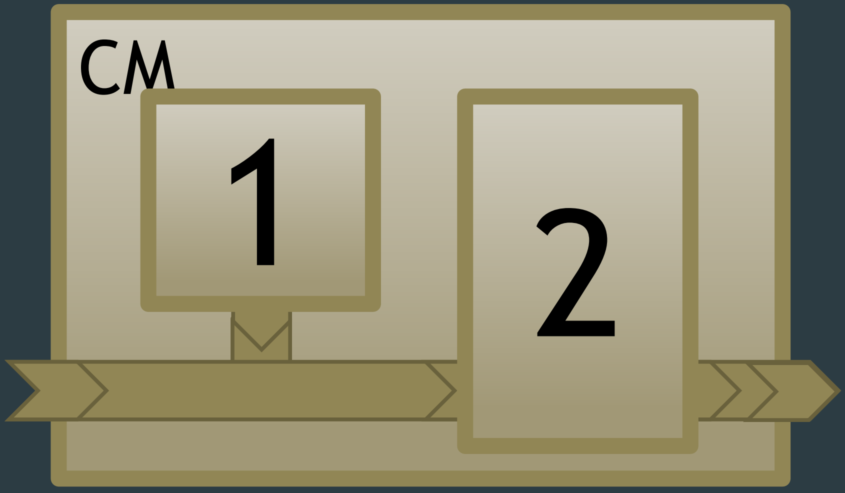
$$flatten(CM) = \langle X, Y, S, q_{init}, \delta_{int}, \delta_{ext}, \lambda, ta \rangle$$

$$q_{init} = (s_{init}, e_{init})$$

$$s_{init} = (\ldots, (s_{init,i}, e_{init,i} - e_{init}), \ldots)$$

$$e_{init} = \min_{i \in D}\{e_{init,i}\}$$





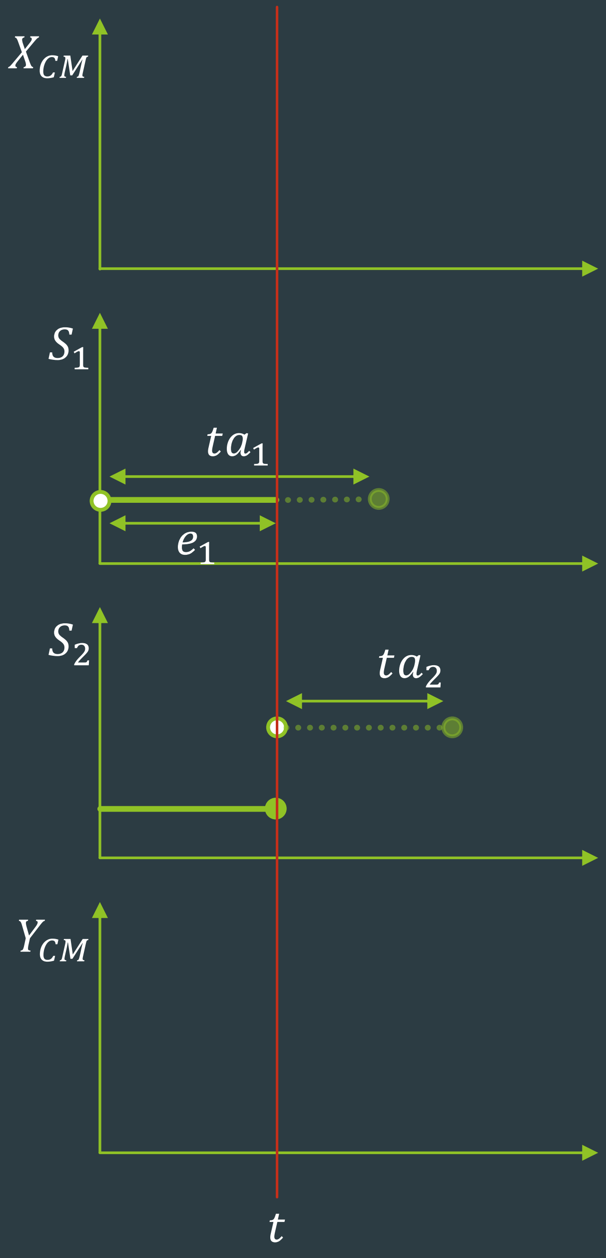
$$flatten(CM) = \langle X, Y, S, q_{init}, \delta_{int}, \delta_{ext}, \lambda, ta \rangle$$

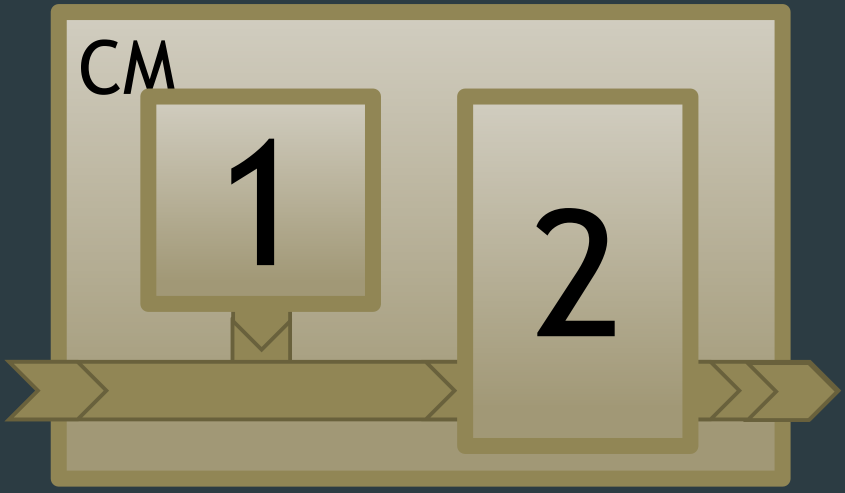
$$q_{init} = (s_{init}, e_{init})$$

$$s_{init} = \left(\ldots, \left(s_{init,i}, e_{init,i} - e_{init}\right), \ldots\right)$$

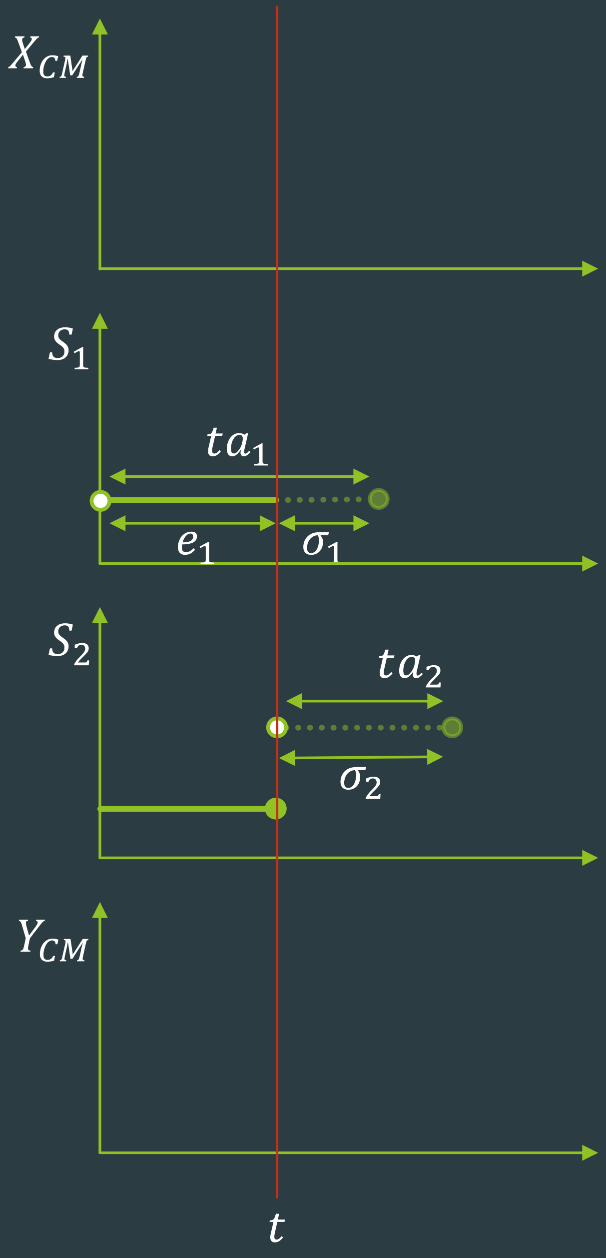
$$e_{init} = \min_{i \in D}\{e_{init,i}\}$$

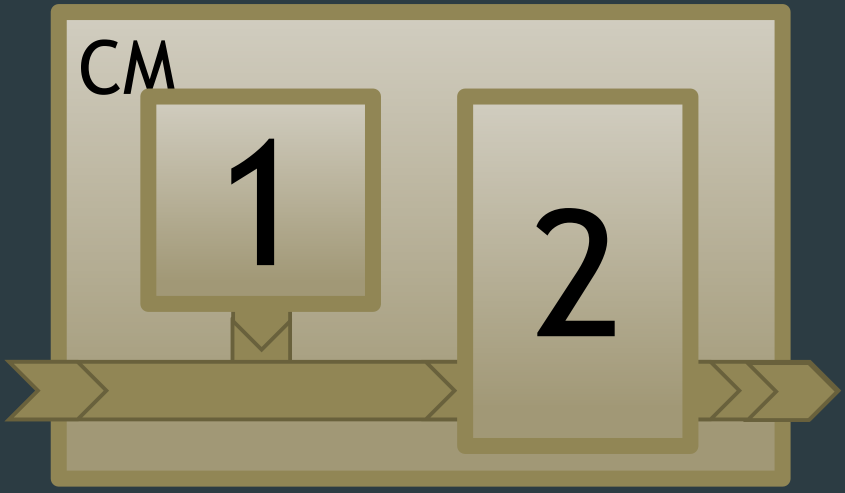
$$\left(s_{init,i}, e_{init,i}\right) = q_{init,i}$$





$$flatten(CM) = \langle X, Y, S, q_{init}, \delta_{int}, \delta_{ext}, \lambda, ta \rangle$$

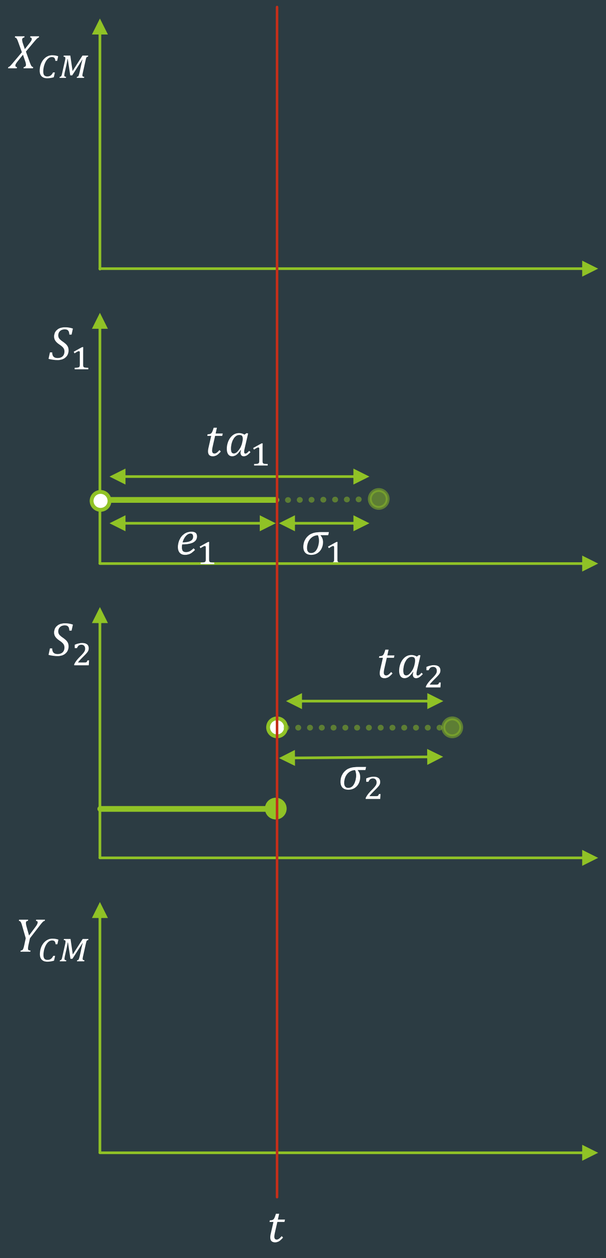


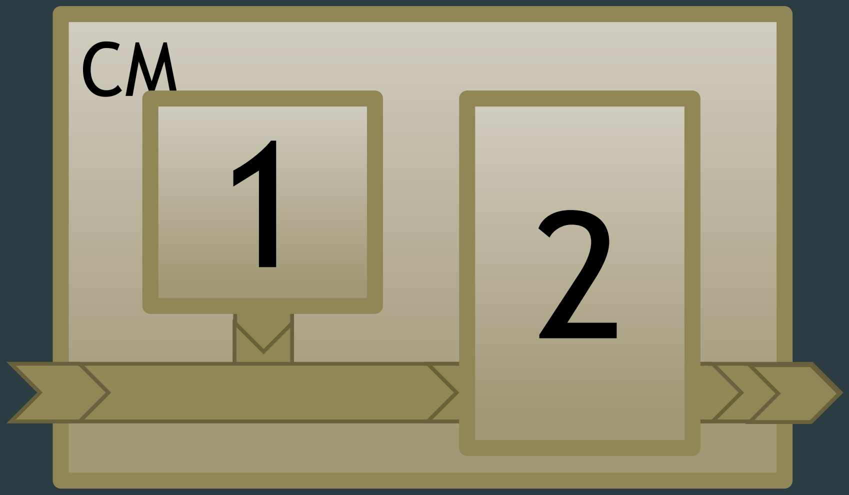


$$flatten(CM) = \langle X, Y, S, q_{init}, \delta_{int}, \delta_{ext}, \lambda, ta \rangle$$

$$ta : S \rightarrow \mathbb{R}^{+}_{0,+\infty}$$

$$ta(s) = \min_{i \in D}\{\sigma_i = ta_i(s_i) - e_i\}$$





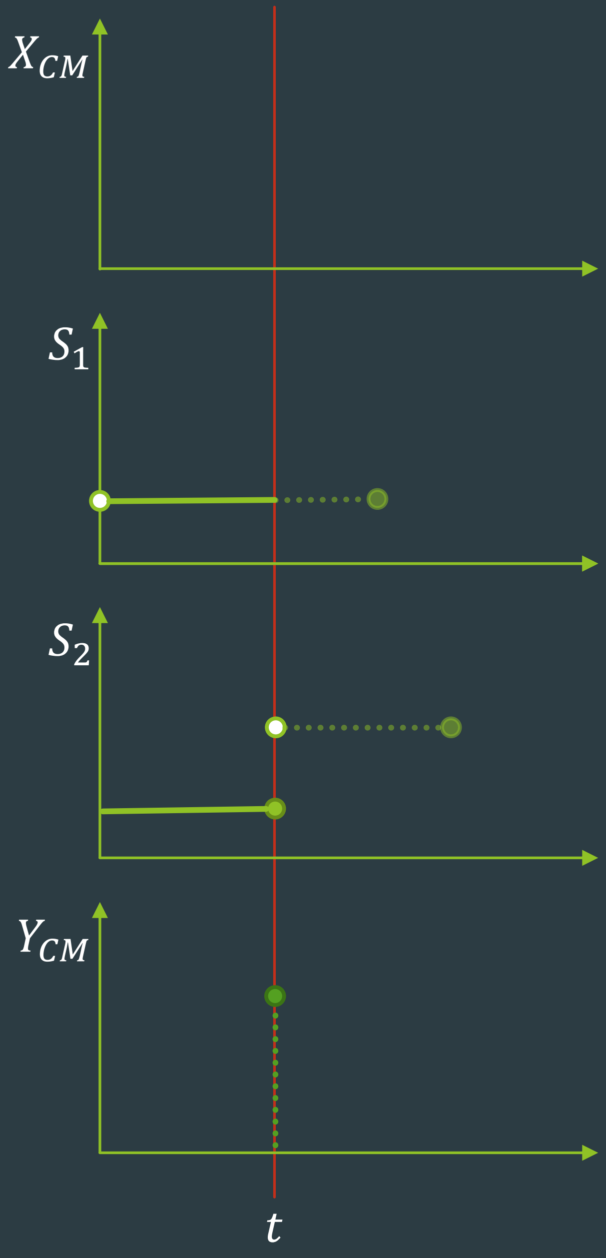
$$flatten(CM) = \langle X, Y, S, q_{init}, \delta_{int}, \delta_{ext}, \lambda, ta \rangle$$

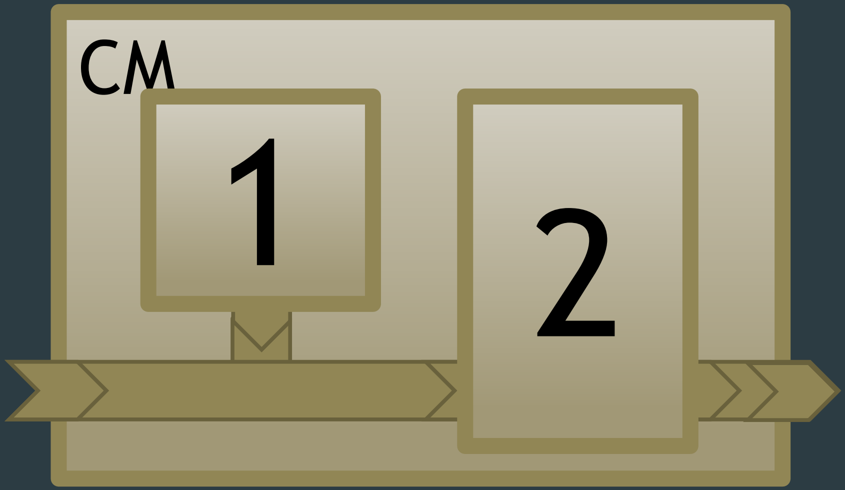
$$ta : S \rightarrow \mathbb{R}^{+}_{0,+\infty}$$

$$ta(s) = \min_{i \in D}\{\sigma_i = ta_i(s_i) - e_i\}$$

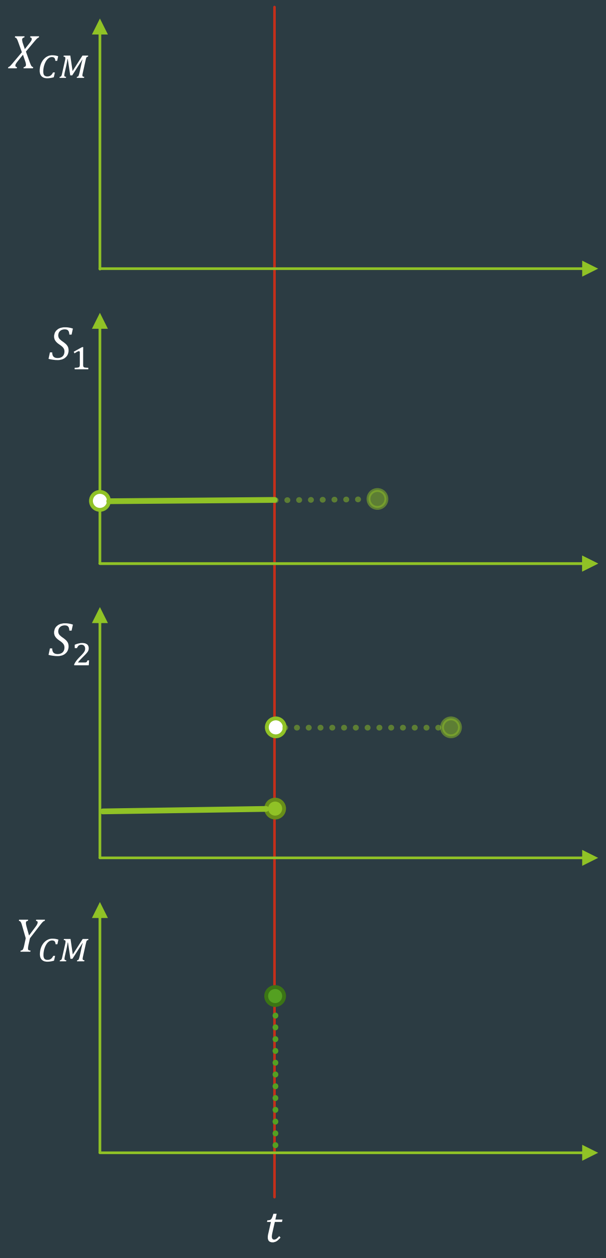
$$IMM(s) = \{i \in D | \sigma_i = ta(s)\}$$

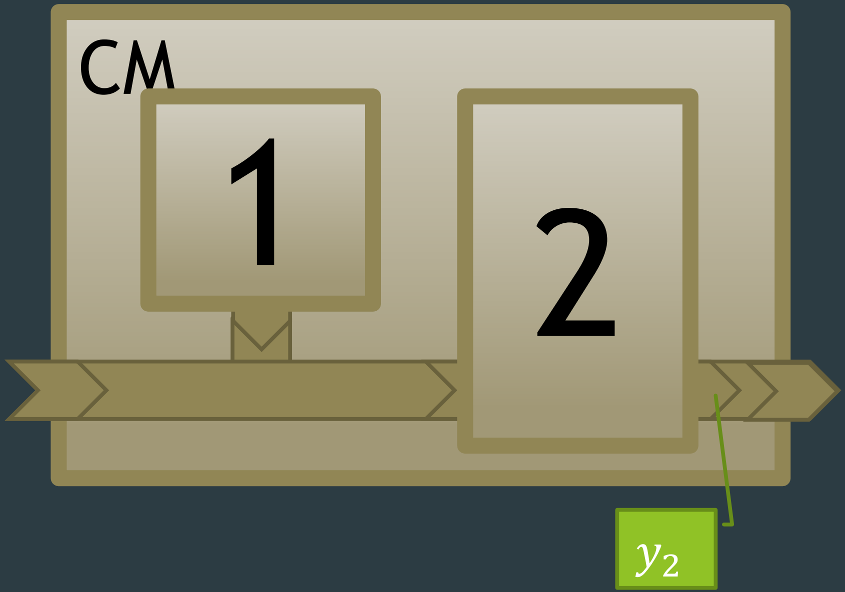
$$select(IMM(s)) = i^{*}$$





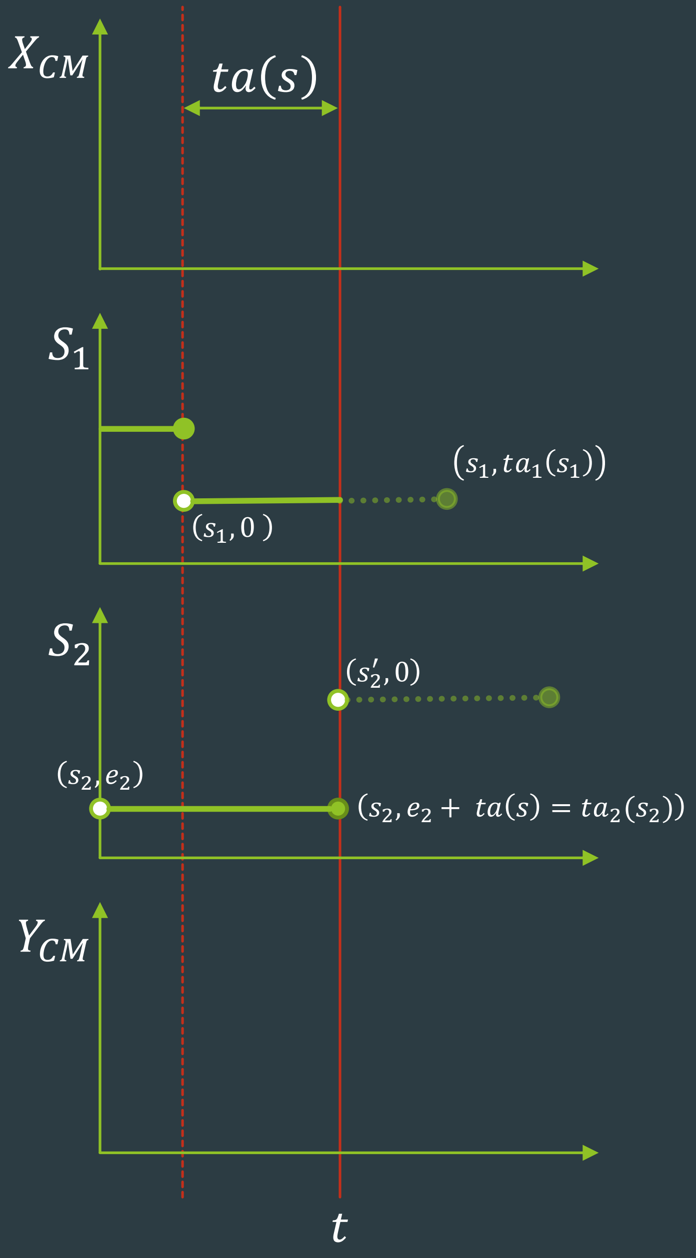
$$flatten(CM) = \langle X, Y, S, q_{init}, \delta_{int}, \delta_{ext}, \lambda, ta \rangle$$

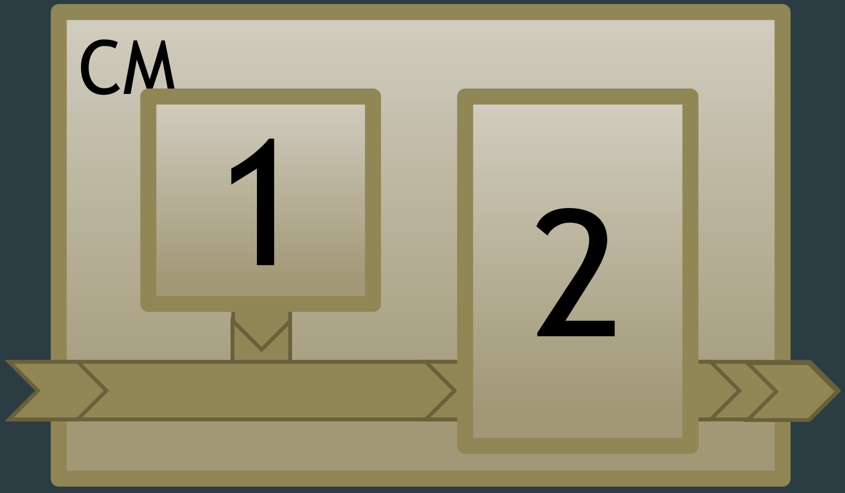




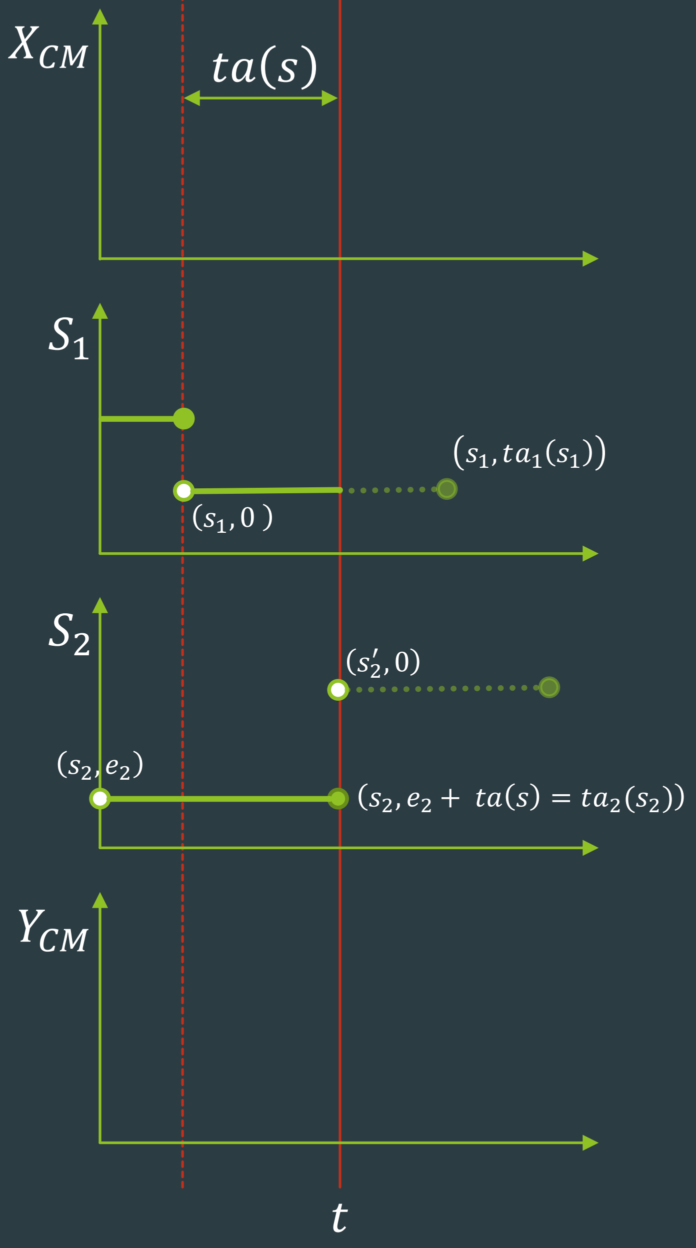
$$flatten(CM) = \langle X, Y, S, q_{init}, \delta_{int}, \delta_{ext}, \lambda, ta \rangle$$

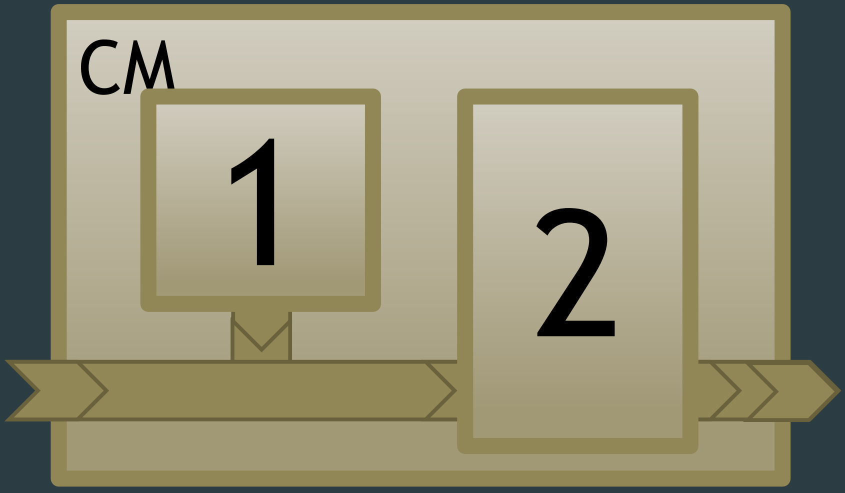
$$\lambda(s) = \begin{cases} Z_{i^*,self}\left(\lambda_{i^*}(s_{i^*})\right) & if\ self \in I_{i^*} \\ \emptyset & if\ self \notin I_{i^*} \end{cases}$$





$$flatten(CM) = \langle X, Y, S, q_{init}, \delta_{int}, \delta_{ext}, \lambda, ta \rangle$$

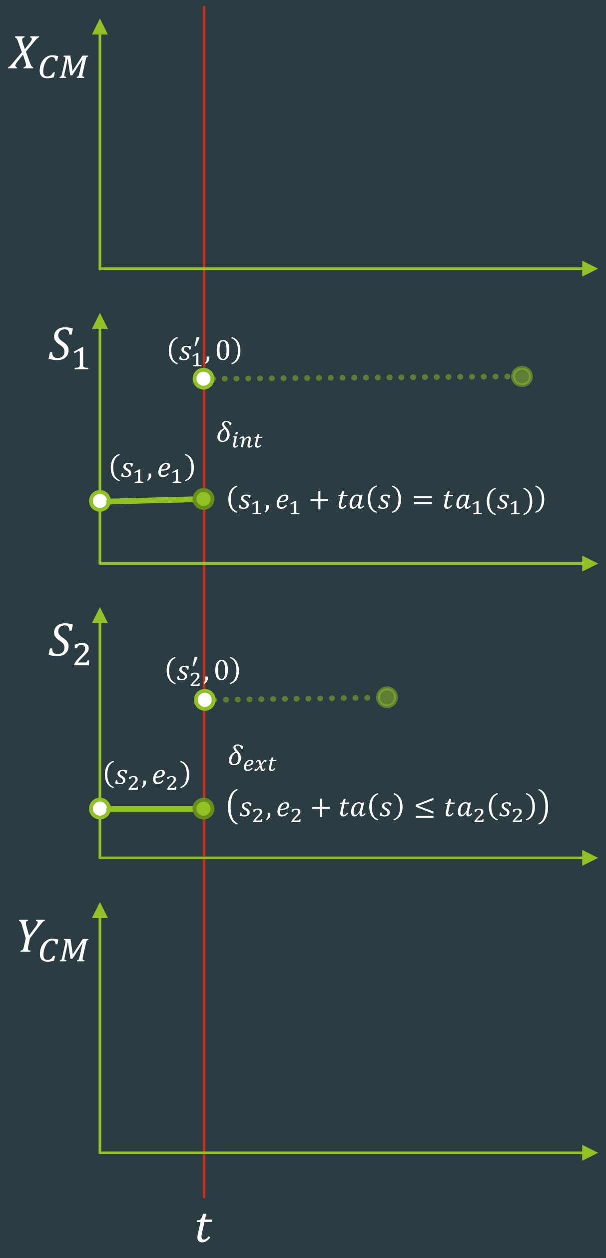


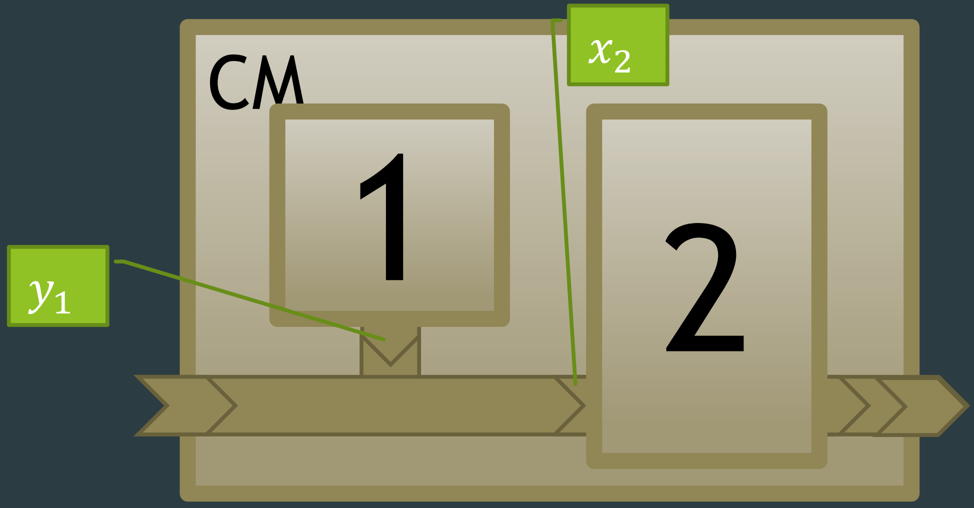


$$flatten(CM) = \langle X, Y, S, q_{init}, \delta_{int}, \delta_{ext}, \lambda, ta \rangle$$

$$\delta_{int}(s) = (\ldots, (s_j', e_j'), \ldots)$$

$$(s_j', e_j') = \begin{cases} (\delta_{int,j}(s_j), 0) & for\ j = i^* \\ ? & for\ j \in I_{i^*} \setminus \{self\} \\ \left(s_j, e_j + ta(s)\right) & else \end{cases}$$

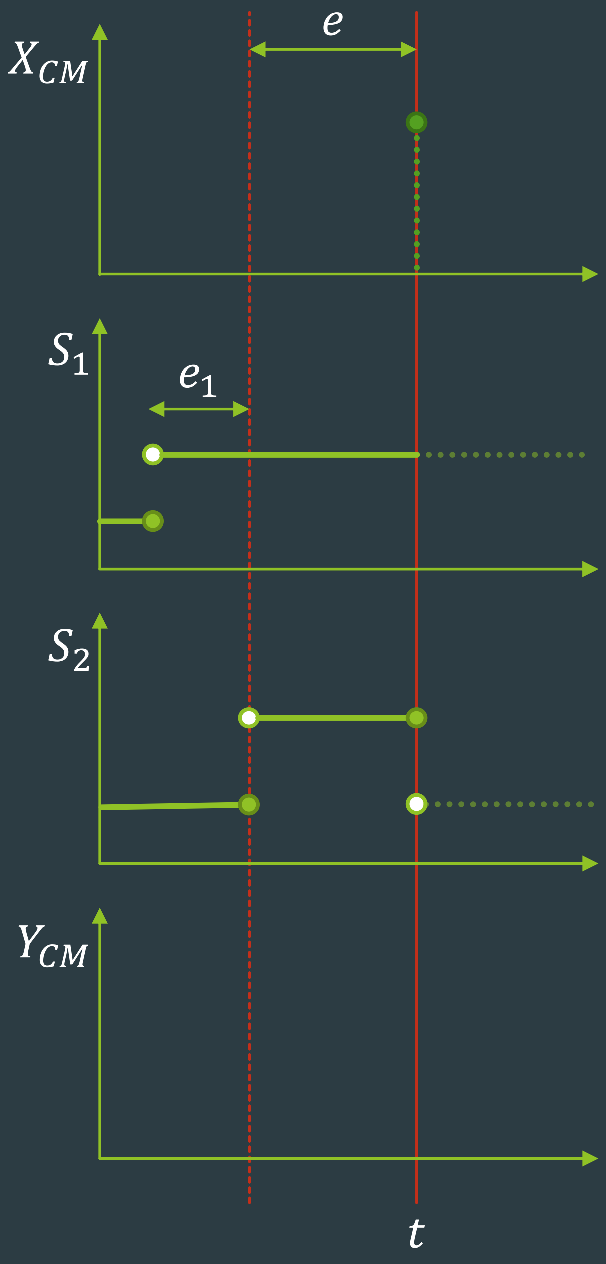


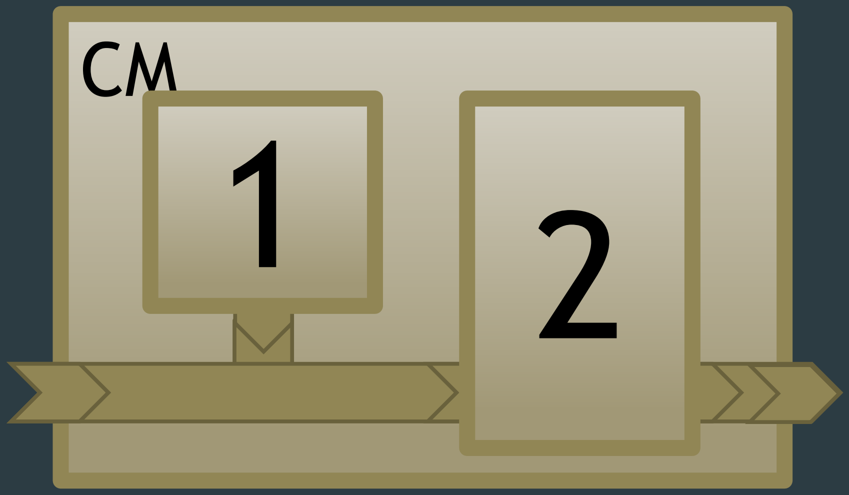


$$flatten(CM) = \langle X, Y, S, q_{init}, \delta_{int}, \delta_{ext}, \lambda, ta \rangle$$

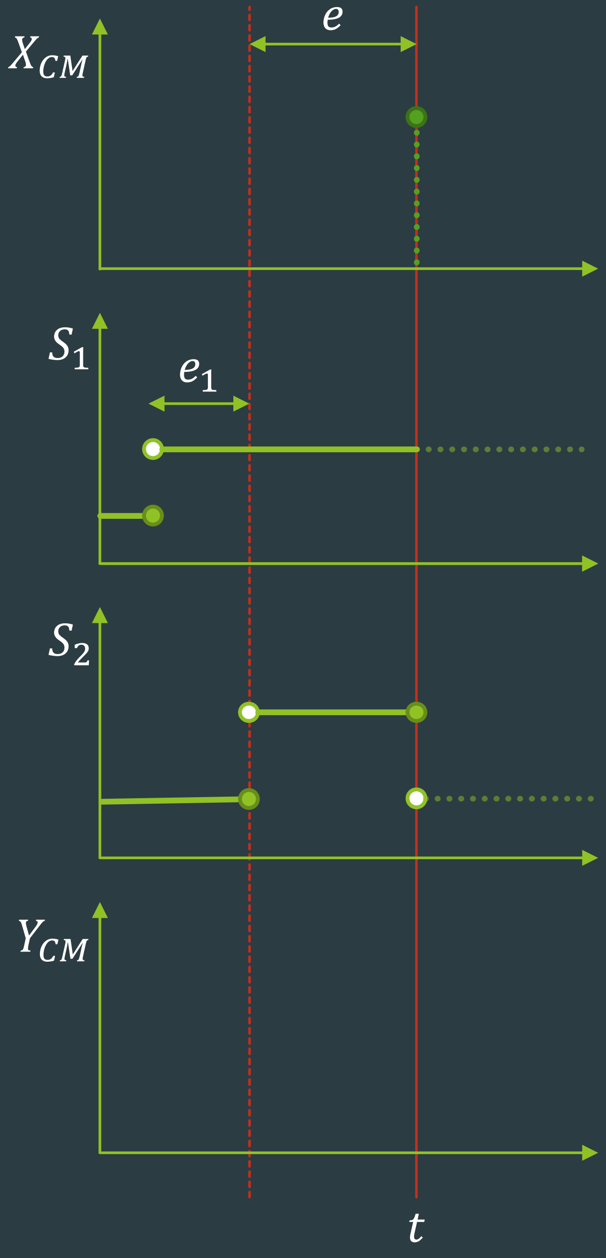
$$\delta_{int}(s) = (\dots, (s'_j, e'_j), \dots)$$

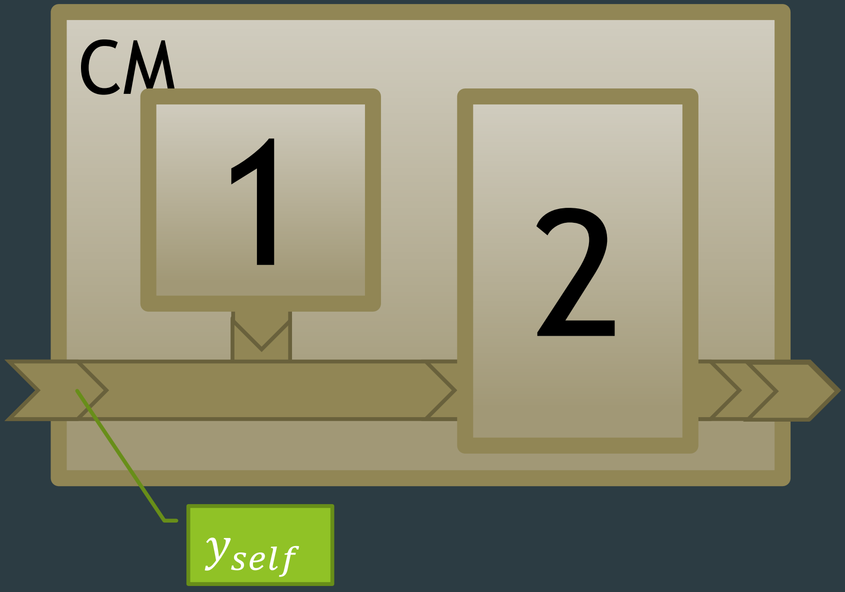
$$(s'_j, e'_j) = \begin{cases} (\delta_{int,j}(s_j), 0) & for\ j = i^* \\ \left(\delta_{ext,j}\left(\left(s_j, e_j + ta(s)\right), Z_{i^*,j}(\lambda_{i^*}(s_{i^*}))\right), 0\right) & for\ j \in I_{i^*} \setminus \{self\} \\ (s_j, e_j + ta(s)) & else \end{cases}$$





$$flatten(CM) = \langle X, Y, S, q_{init}, \delta_{int}, \delta_{ext}, \lambda, ta \rangle$$





$$flatten(CM) = \langle X, Y, S, q_{init}, \delta_{int}, \delta_{ext}, \lambda, ta \rangle$$

$$\delta_{ext}\big((s,e),x\big) = (\dots, (s_i', e_i'), \dots)$$

$$(s_i', e_i') = \begin{cases} \Big(\delta_{ext,i}\Big((s_i, e_i + e), Z_{self,i}(x)\Big), 0\Big) & for\ i \in I_{self} \\ (s_i, e_i + e) & else \end{cases}$$

$$flatten(CM) = \langle X, Y, S, q_{init}, \delta_{int}, \delta_{ext}, \lambda, ta \rangle$$

$$X = X_{CM}$$

$$Y = Y_{CM}$$

$$S = \times_{i \in D} Q_i$$

$$Q = \{(s, e) | s \in S, 0 \leq e \leq ta(s)\}$$

$$q_{init} = (s_{init}, e_{init}) \in Q$$

$$s_{init} = (\ldots, (s_{init,i}, e_{init,i} - e_{init}), \ldots)$$

$$e_{init} = \min_{i \in D}\{e_{init,i}\}$$

$$(s_{init,i}, e_{init,i}) = q_{init,i}$$

$$\delta_{int}(s) = (\ldots, (s'_j, e'_j), \ldots)$$

$$(s'_j, e'_j) = \begin{cases} (\delta_{int,j}(s_j), 0) & for\ j = i^* \\ \left(\delta_{ext,j}\left((s_j, e_j + ta(s)), Z_{i^*,j}(\lambda_{i^*}(s_{i^*}))\right), 0\right) & for\ j \in I_{i^*} \\ (s_j, e_j + ta(s)) & else \end{cases}$$

$$\delta_{ext}((s, e), x) = (\ldots, (s'_i, e'_i), \ldots)$$

$$(s'_i, e'_i) = \begin{cases} \left(\delta_{ext,i}\left((s_i, e_i + e), Z_{self,i}(x)\right), 0\right) & for\ i \in I_{self} \\ (s_i, e_i + e) & else \end{cases}$$

$$\lambda(s) = \begin{cases} Z_{i^*,self}(\lambda_{i^*}(s_{i^*})) & if\ self \in I_{i^*} \\ \phi & if\ self \notin I_{i^*} \end{cases}$$

$$i^* = select(IMM(s))$$

$$IMM(s) = \{i \in D | \sigma_i = ta(s)\}$$

$$ta(s) = \min_{i \in D}\{\sigma_i = ta_i(s_i) - e_i\}$$

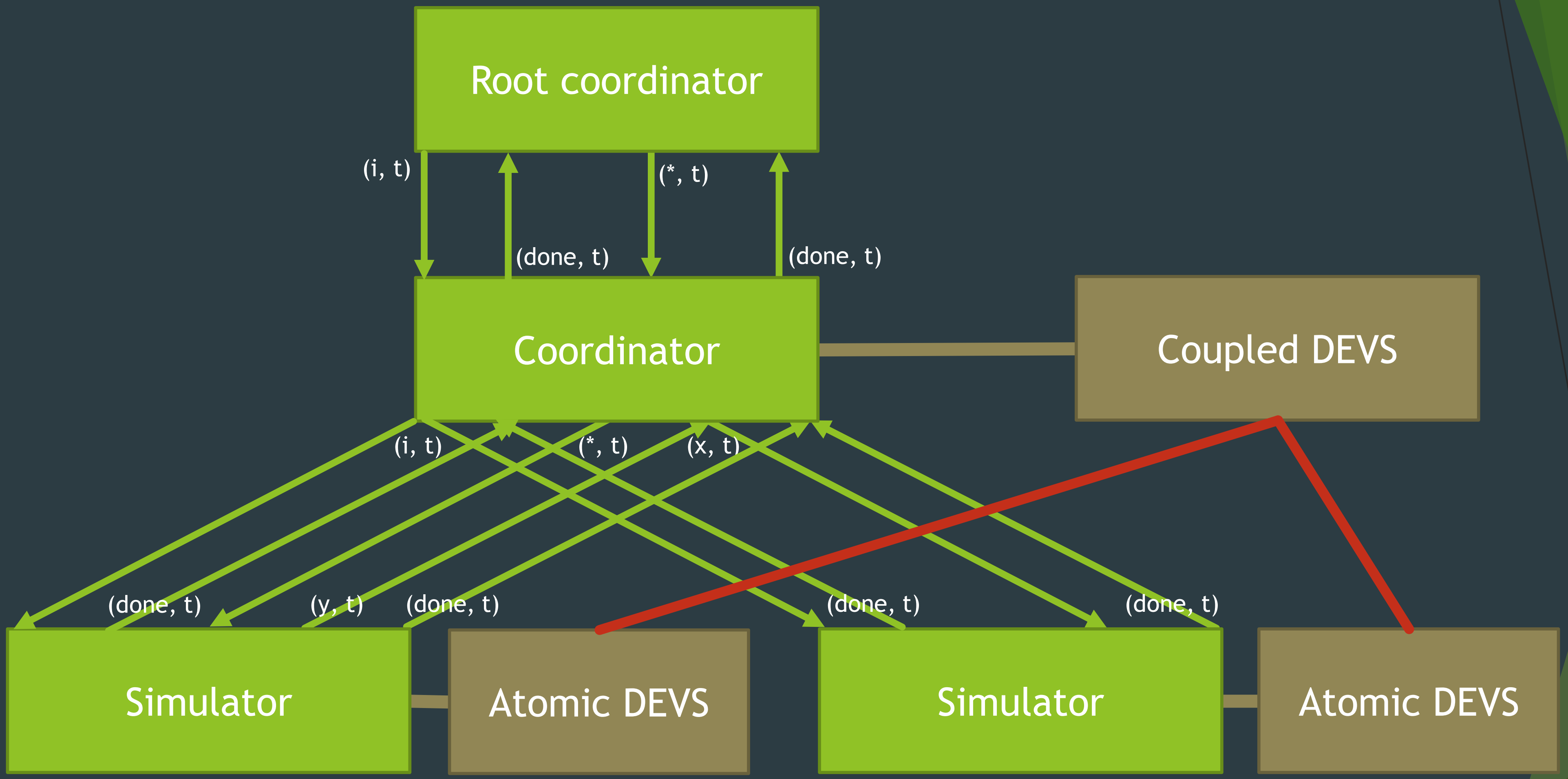
Hierarchical Simulator

Operational semantics for Coupled DEVS models

DEVS Semantics

	Operational Semantics	Denotational Semantics
Atomic DEVS	Abstract Simulator	modal Discrete Event Logic L_{DE} [1]
Coupled DEVS	Hierarchical Simulator	Closure under Coupling

[1] Ashvin Radiya and Robert G. Sargent. A logic-based foundation of discrete event modeling and simulation. ACM Transactions on Modeling and Computer Simulation, 1(1):3-51, 1994.


Root coordinator
(i, t)
(done, t)
(*, t)
(done, t)
Coordinator
Coupled DEVS
(i, t)
(*, t)
(x, t)
(done, t)
(y, t)
(done, t)
(done, t)
(done, t)
Simulator
Atomic DEVS
Simulator
Atomic DEVS

message m	**simulator**	**coordinator**
$(*, from, t)$	simulator correct only if $t = t_N$	
	$y \leftarrow \lambda(s)$	**send** $(*, self, t)$ to i^*, where
	if $y \neq \phi$:	$i^* = select(imm\_children)$
	send $(\lambda(s), self, t)$ to $parent$	$imm\_children = \{i \in D \mid M_i.t_N = t\}$
	$s \leftarrow \delta_{int}(s)$	$active\_children \leftarrow active\_children \cup \{i^*\}$
	$t_L \leftarrow t$	
	$t_N \leftarrow t_L + ta(s)$	
	send $(done, self, t_N)$ to $parent$	

message m	simulator	coordinator
$(x, from, t)$	simulator correct only if $t_L \leq t \leq t_N$ (ignore δ_{int} to resolve a $t = t_N$ *conflict*)	
	$e \leftarrow t - t_L$	$\forall i \in I_{self}$:
	$s \leftarrow \delta_{ext}(s, e, x)$	**send** $(Z_{self,i}(x), self, t)$ to i
	$t_L \leftarrow t$	$active\_children \leftarrow active\_children \cup \{i\}$
	$t_N \leftarrow t_L + ta(s)$	
	send $(done, self, t_N)$ to $parent$	

message m	simulator	coordinator
$(y, from, t)$		$\forall i \in I_{from} \setminus \{self\}$:
		send $(Z_{from,i}(y), from, t)$ to i
		$active\_children \leftarrow active\_children \cup \{i\}$
		if $self \in I_{from}$:
		send $(Z_{from,self}(y), self, t)$ to $parent$
$(done, from, t)$		$active\_children \leftarrow active\_children \setminus \{from\}$
		if $active\_children = \emptyset$:
		$t_L \leftarrow t$
		$t_N \leftarrow min\{M_i.t_N \mid i \in D\}$
		send $(done, self, t_N)$ to $parent$

$t \leftarrow t_N$ of topmost coordinator

repeat until $t \geq t_{end}$ (or some other termination condition)

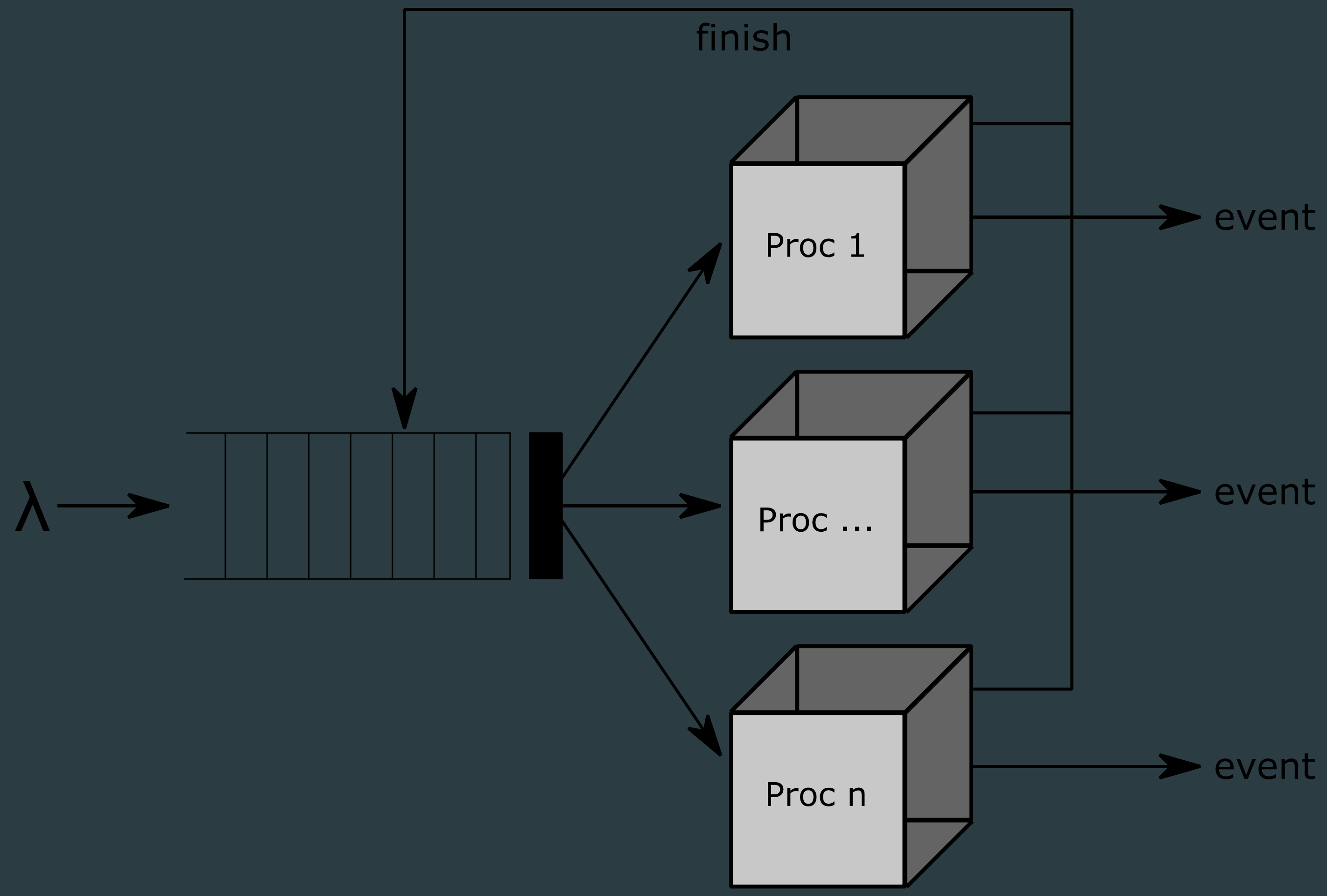
  send $(*, main, t)$ to topmost coupled model top

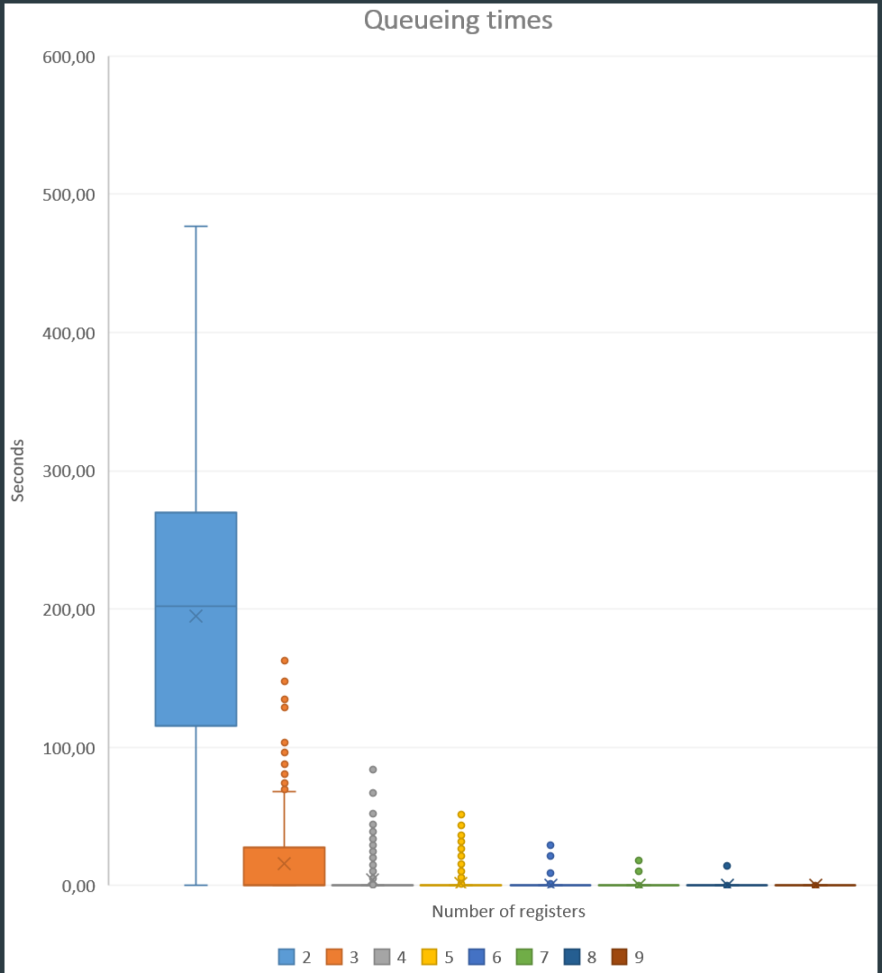
  wait for $(done, top, t_N)$

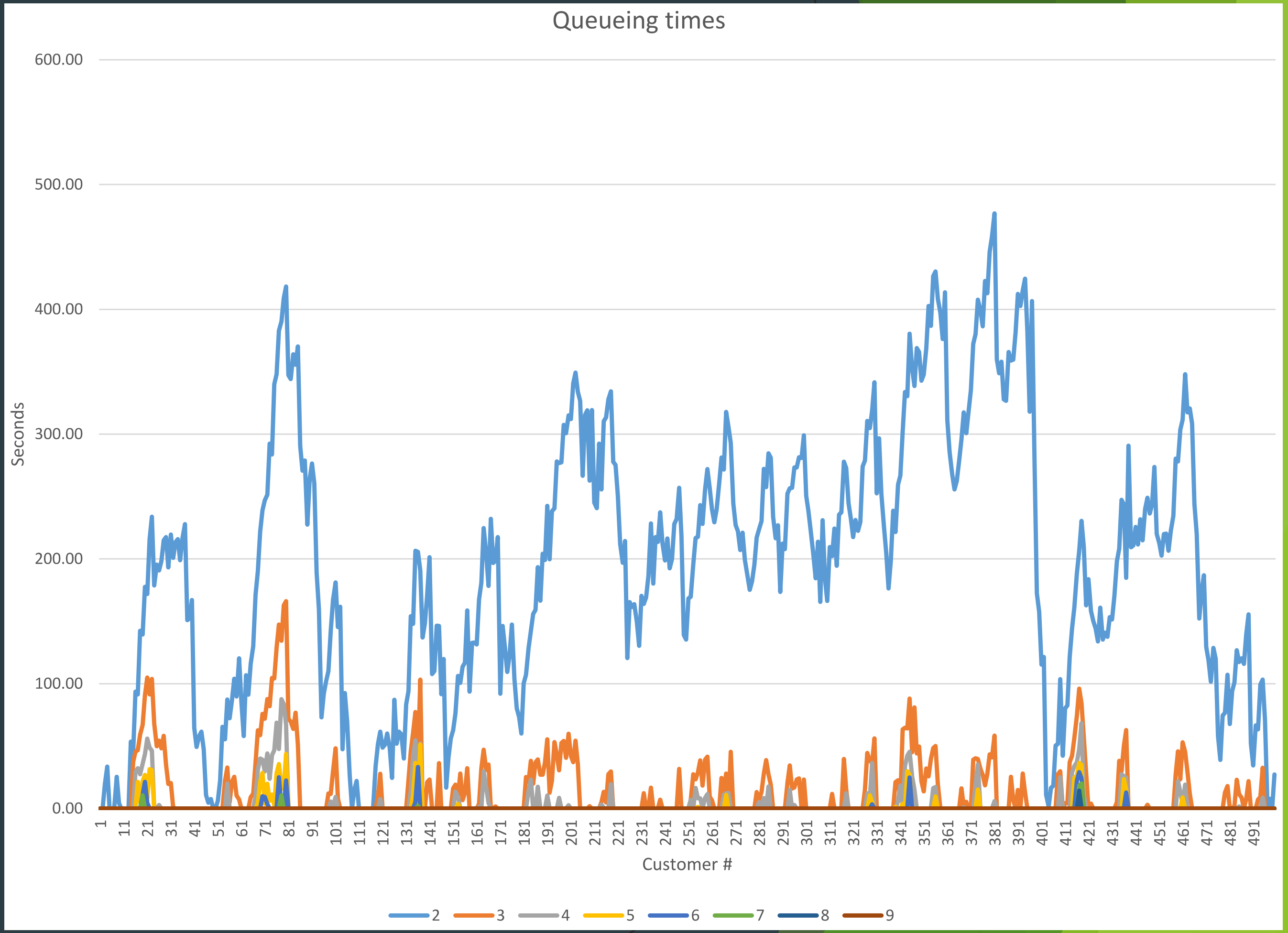
  $t \leftarrow t_N$

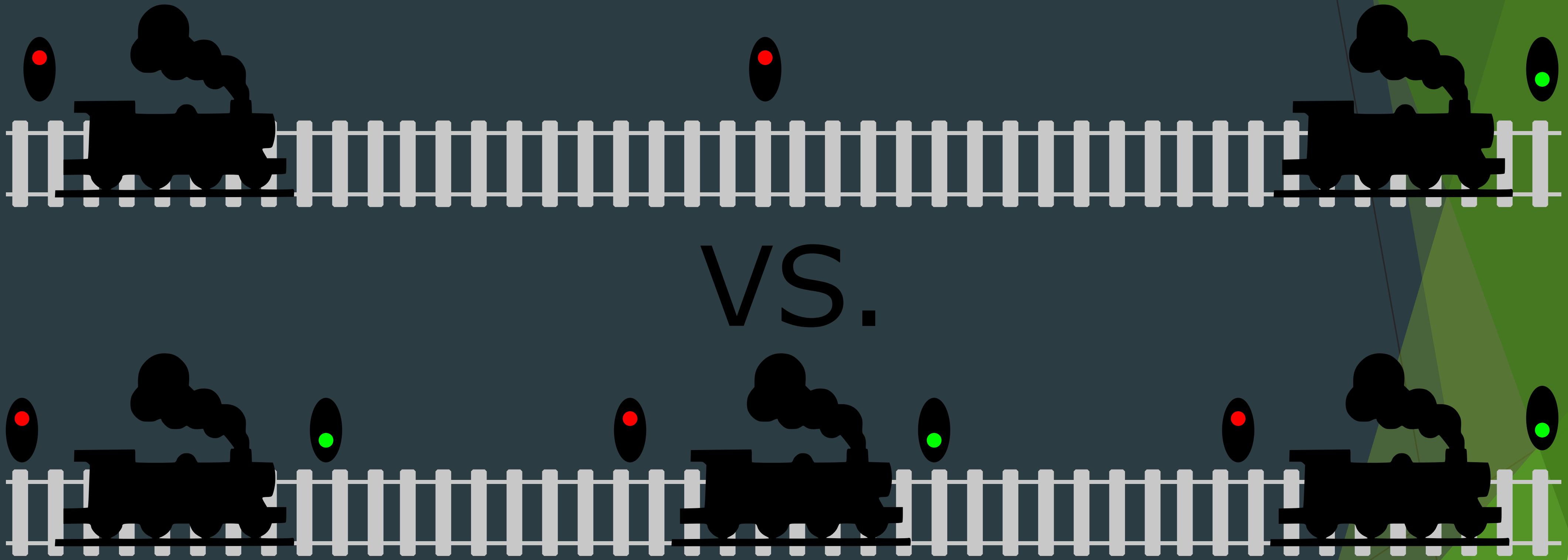
Applications of DEVS

DEVS in practice

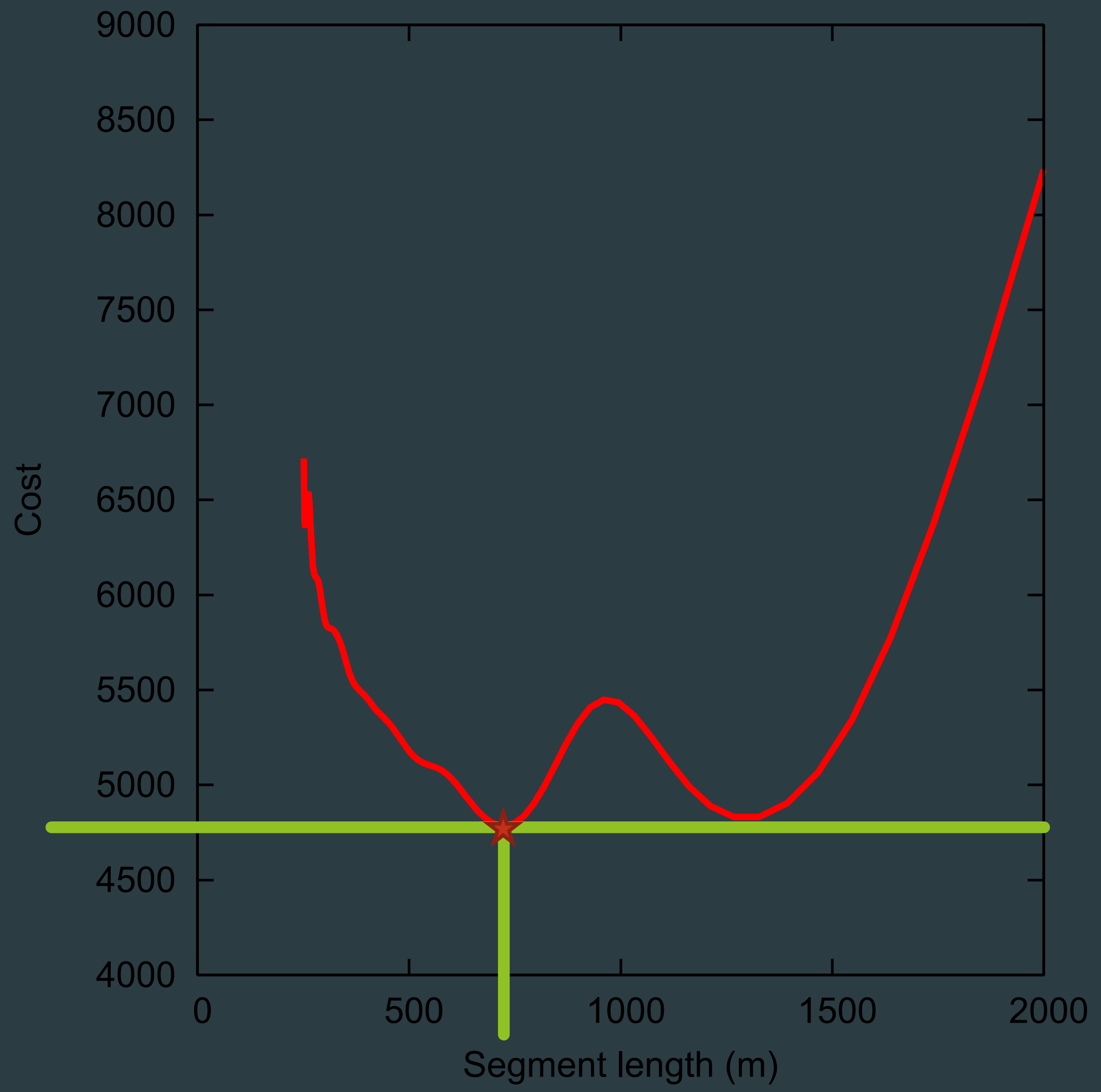

finish
λ
Proc 1
Proc ...
Proc n
event
event
event

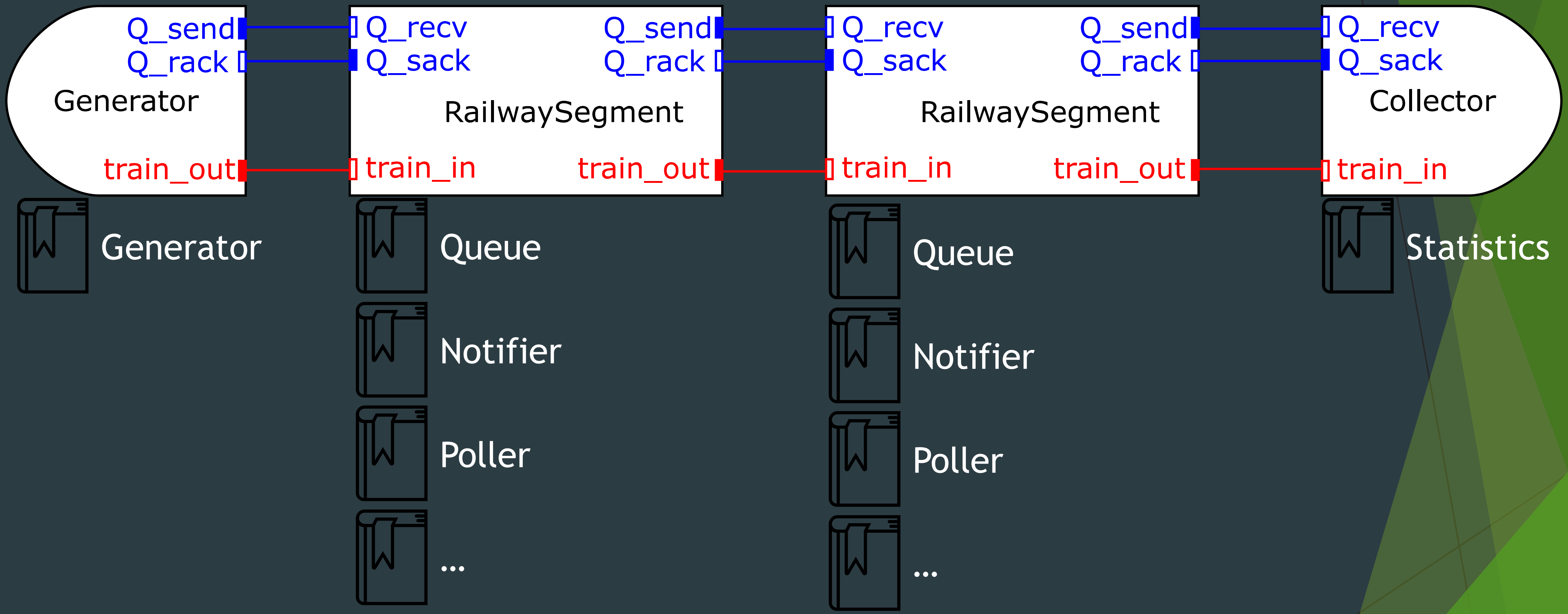

Queueing times
600,00
500,00
400,00
300,00
200,00
100,00
0,00
Seconds
Number of registers
2 3 4 5 6 7 8 9


Queueing times
600.00
500.00
400.00
300.00
200.00
100.00
0.00
Seconds
Customer #
2 3 4 5 6 7 8 9


VS.

$$cost = 10 \times \#lights + avg(t_{travel})$$





Q_send
Q_rack
Generator
train_out
Q_recv
Q_sack
Q_send
Q_rack
RailwaySegment
train_in
train_out
Q_recv
Q_sack
Q_send
Q_rack
RailwaySegment
train_in
train_out
Q_recv
Q_sack
Collector
train_in
Generator
Queue
Notifier
Poller
...
Queue
Notifier
Poller
...
Statistics

Custom Atomic DEVS Models

Extending a DEVS repository

Atomic DEVS Specification

$$M = \langle X, Y, S, q_{init}, \delta_{int}, \delta_{ext}, \lambda, ta \rangle$$

X : set of input events

Y : set of output events

S : set of sequential states

$q_{init} : Q$

$$Q = \{(s, e) | s \in S, 0 \le e \le ta(s)\}$$

$\delta_{int} : S \rightarrow S$

$\delta_{ext} : Q \times X \rightarrow S$

$\lambda : S \rightarrow Y \cup \{\phi\}$

$ta : S \rightarrow \mathbb{R}^{+}_{0,+\infty}$

Atomic DEVS Specification

$$M = \langle X, Y, S, q_{init}, \delta_{int}, \delta_{ext}, \lambda, ta \rangle$$

X : set of input events

Y : set of [redacted]

[redacted]

$= \{(s,e) | s \in S, 0 \leq e \leq ta(s)\}$

$\delta_{int} : S \rightarrow S$

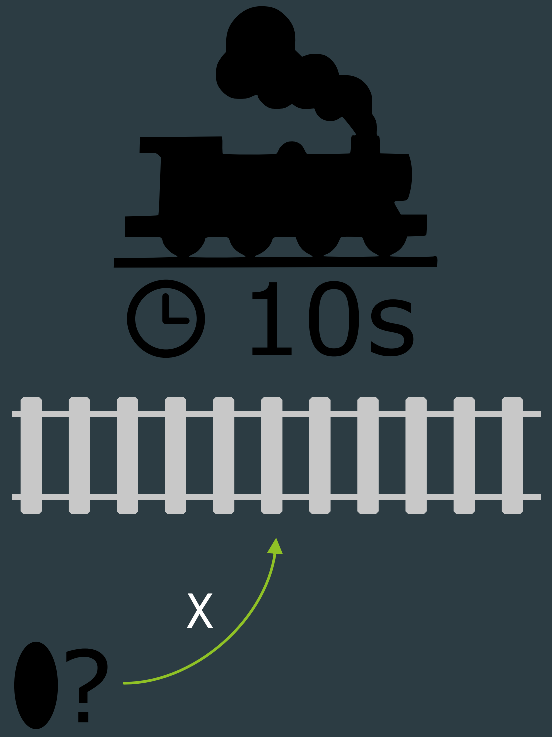
$\delta_{ext} : Q \times X \rightarrow S$

$\lambda : S \rightarrow Y \cup \{\phi\}$

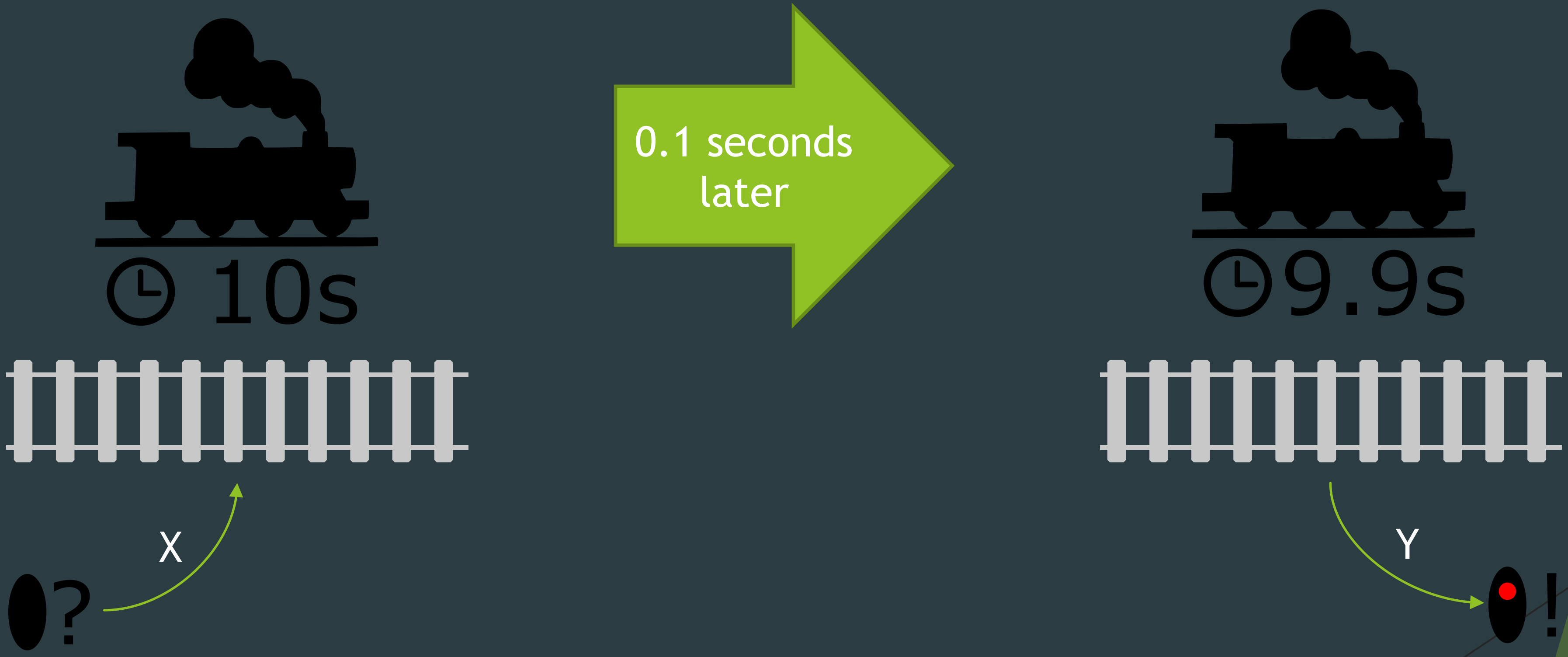
$ta : S \rightarrow \mathbb{R}^{+}_{0,+\infty}$

PATTERNS!

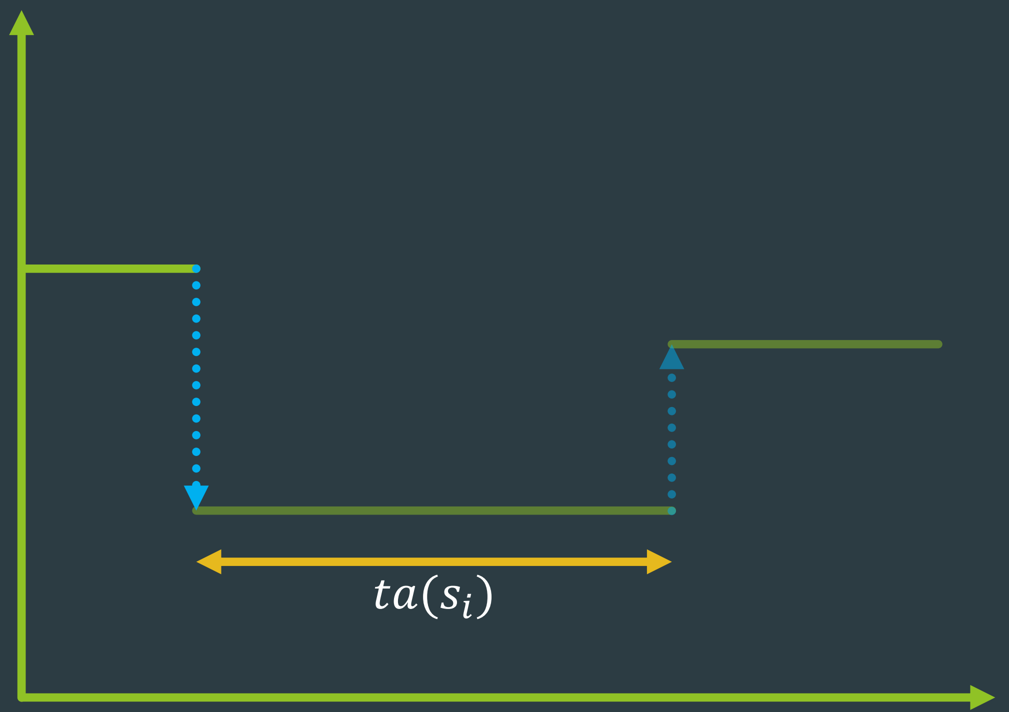
Pattern 1: Ignore an Event



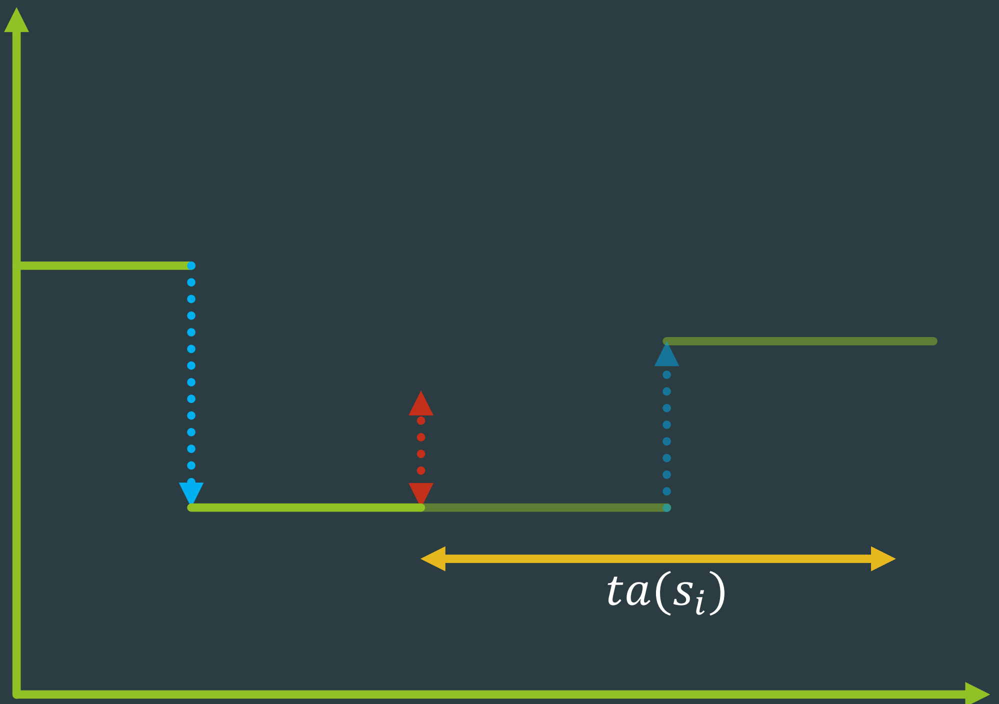
Pattern 1: Ignore an Event



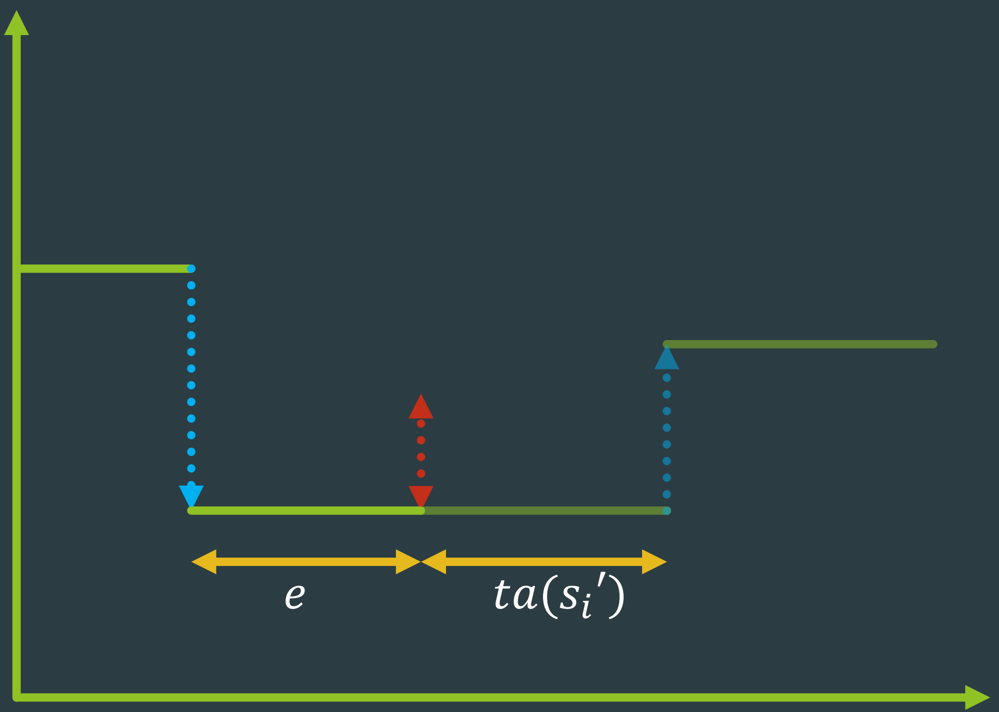
Pattern 1: Ignore an Event



Pattern 1: Ignore an Event



Pattern 1: Ignore an Event

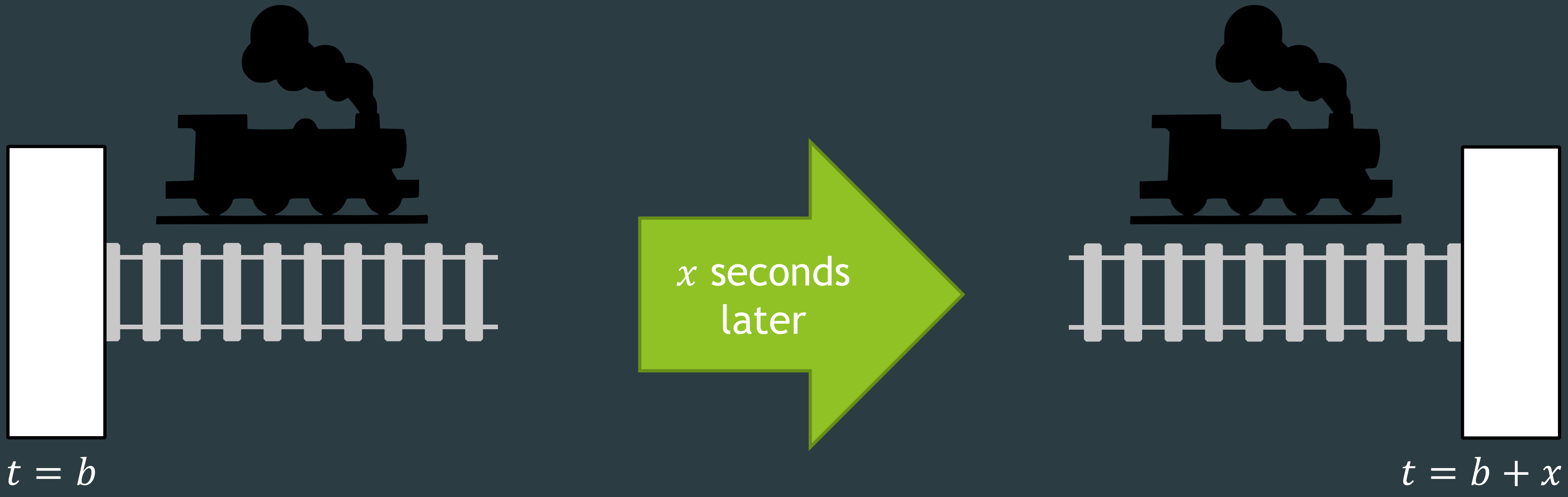


$$ta(s_i') = ta(s_i) - e$$

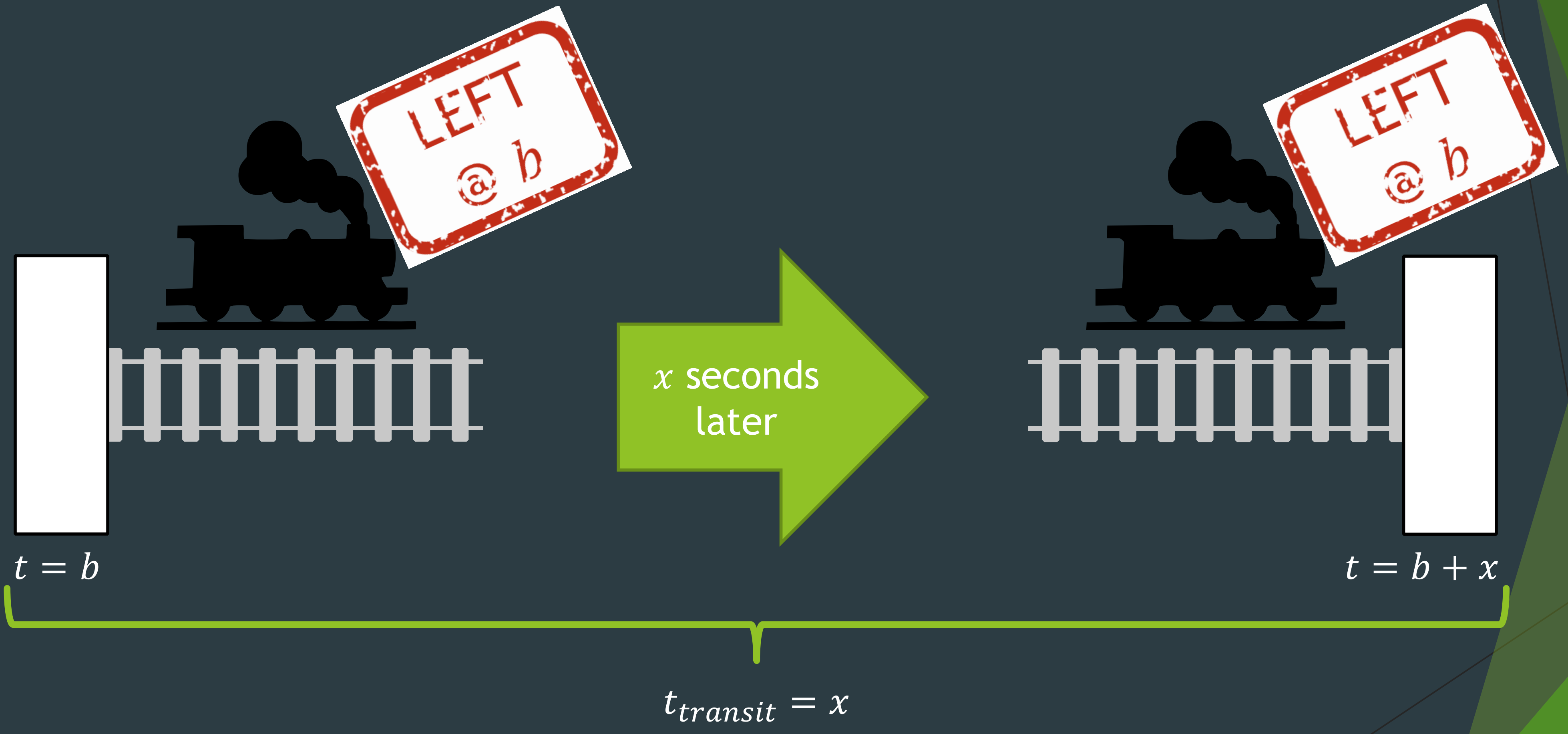
Pattern 1: Ignore an Event

```python
class RailwaySegmentState():
    def __init__(self):
        self.timer = INFINITY

class RailwaySegment(AtomicDEVS):
    def extTransition(self, inputs):
        self.state.timer -= self.elapsed
        …

    def timeAdvance(self):
        return self.state.timer
```

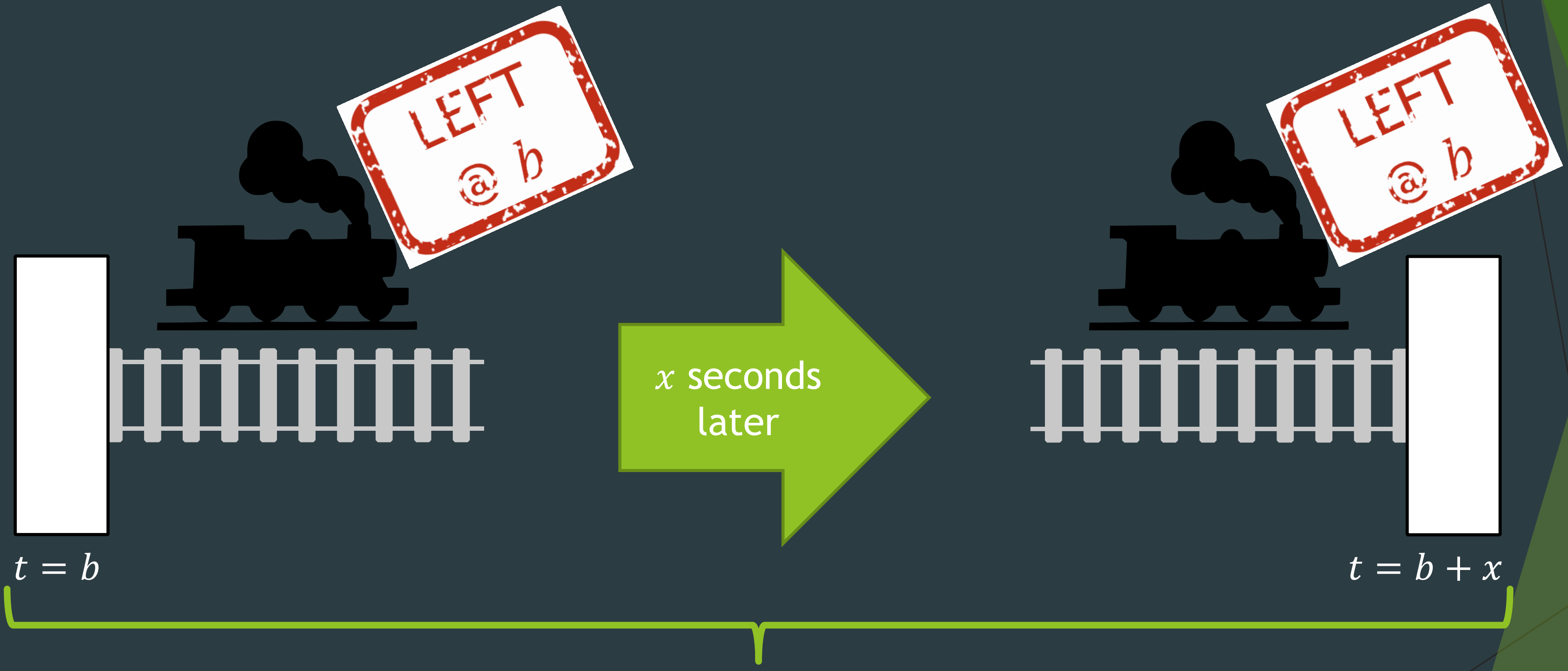
Pattern 2: Simulated Time



Pattern 2: Simulated Time



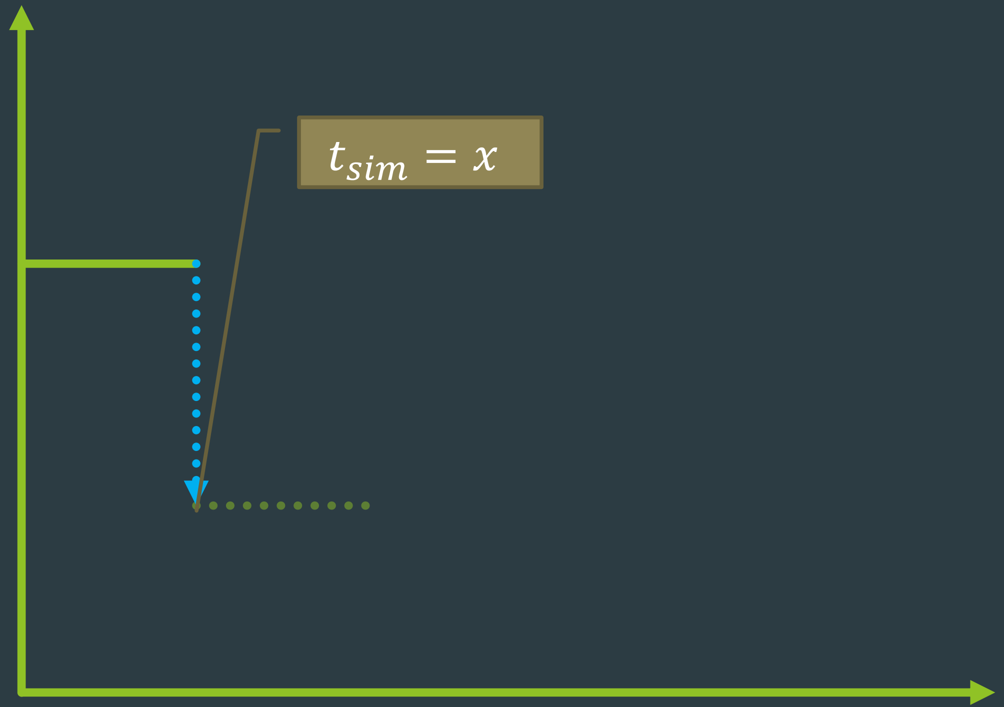
Pattern 2: Simulated Time



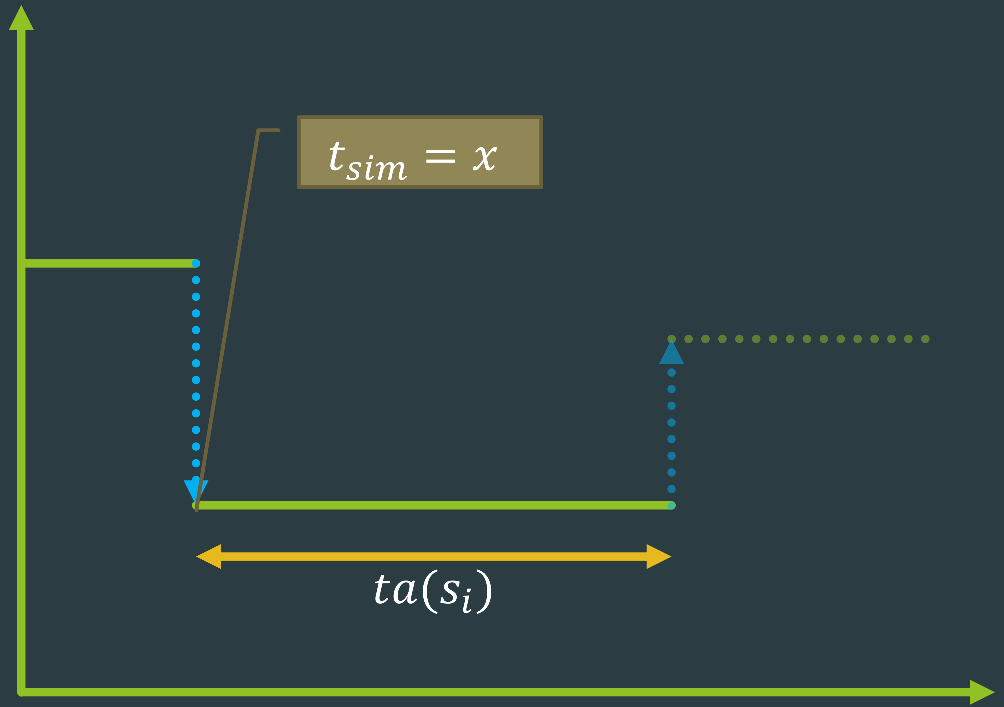
$$t_{transit} = x$$

$$t = ?$$

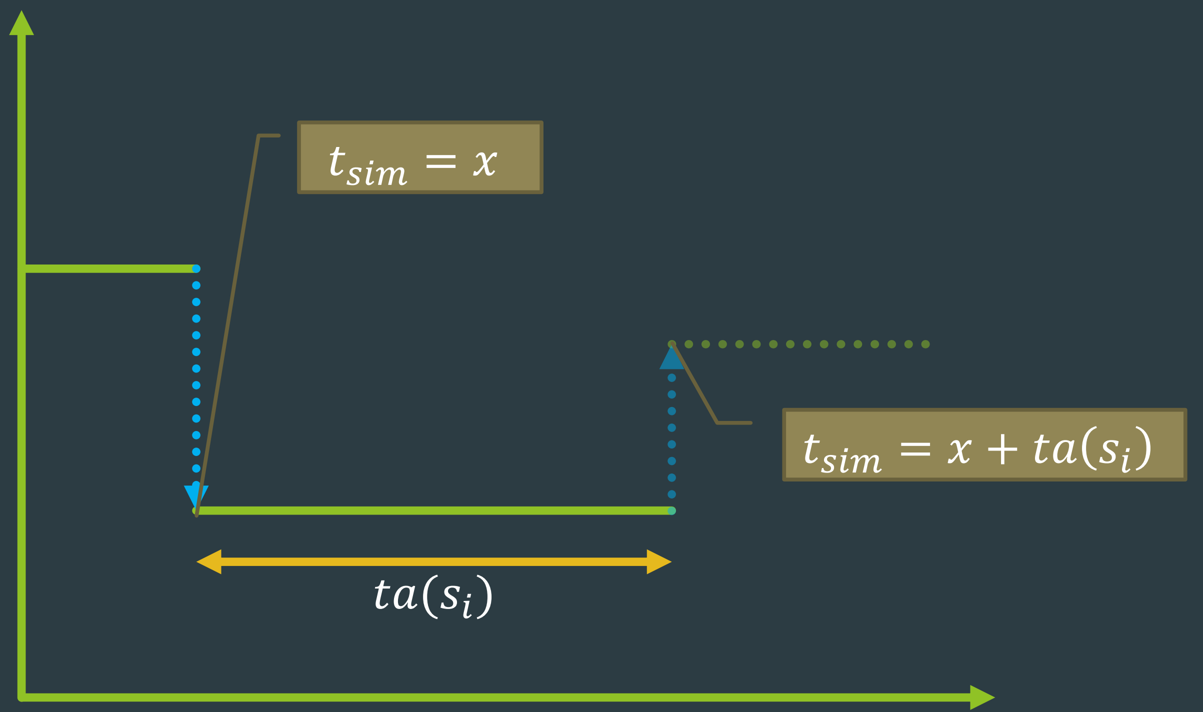
Pattern 2: Simulated Time



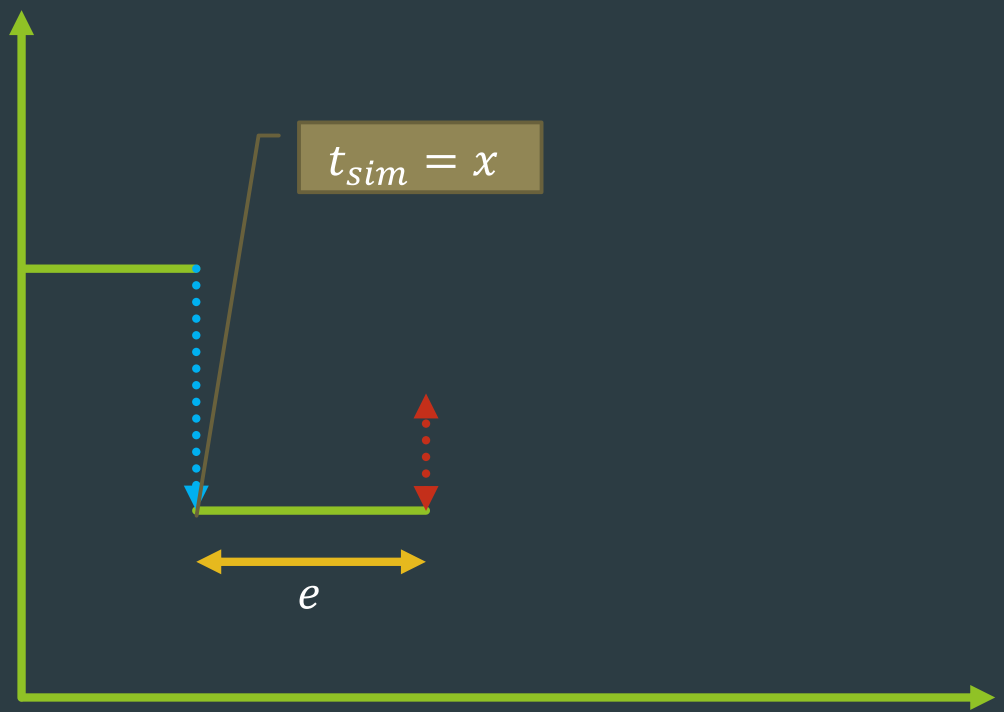
Pattern 2: Simulated Time



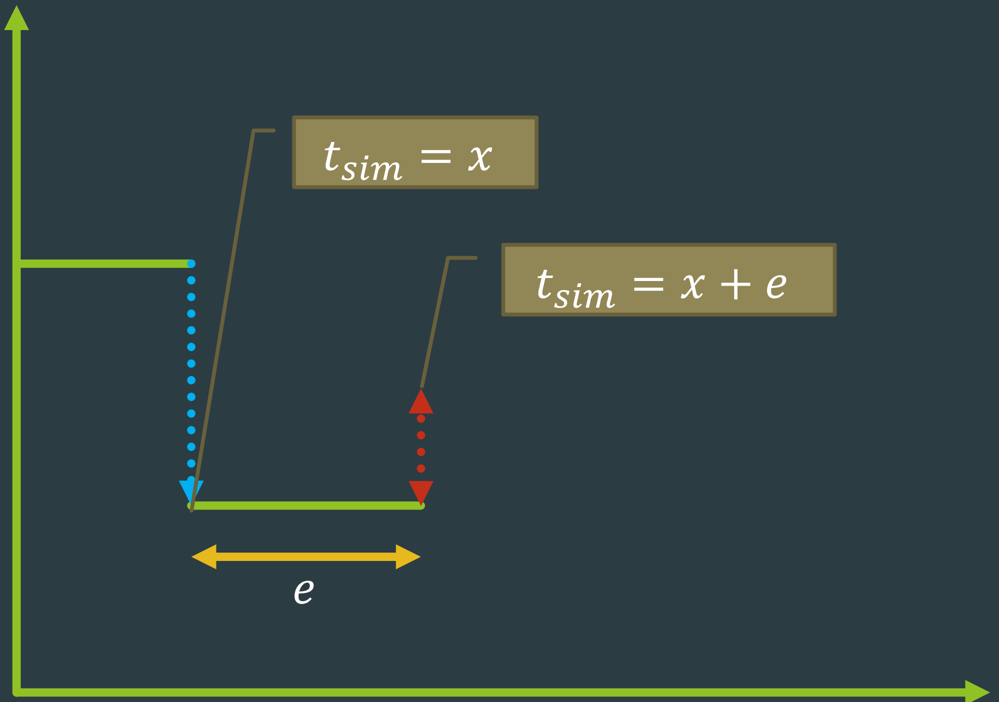
Pattern 2: Simulated Time



Pattern 2: Simulated Time



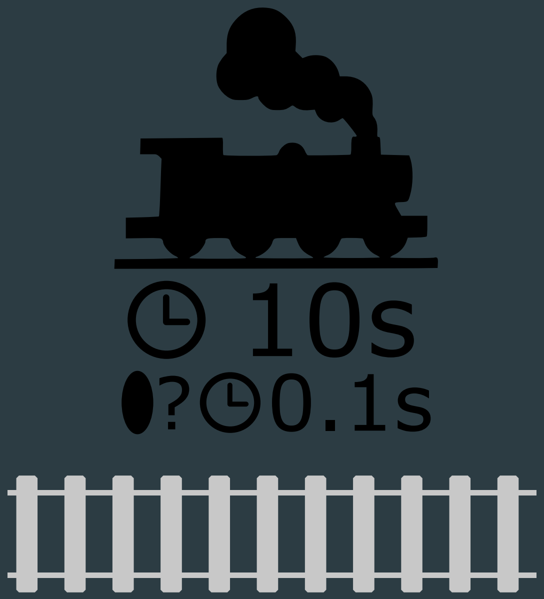
Pattern 2: Simulated Time



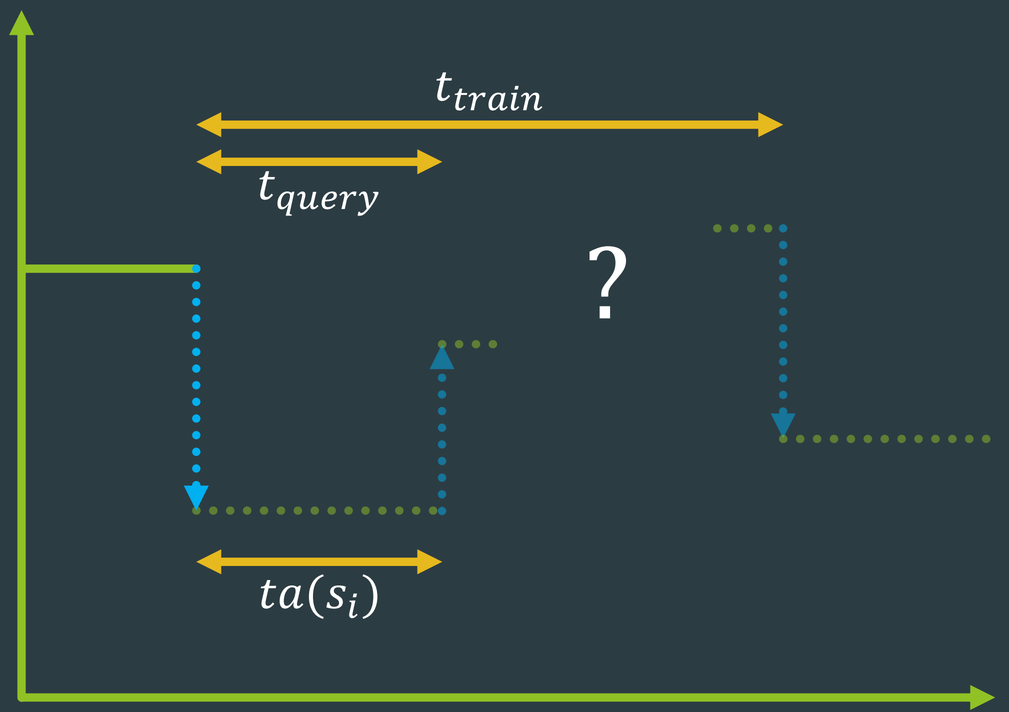
Pattern 2: Simulated Time

```
class GeneratorState:
    def __init__(self):
        self.t_sim = 0.0
        …

class Generator(AtomicDEVS):
    def __init__(self):
        self.state = GeneratorState()
        …

    def intTransition(self):
        self.state.t_sim += self.timeAdvance()
        …

    def extTransition(self, inputs):
        self.state.t_sim += self.elapsed
        …
```

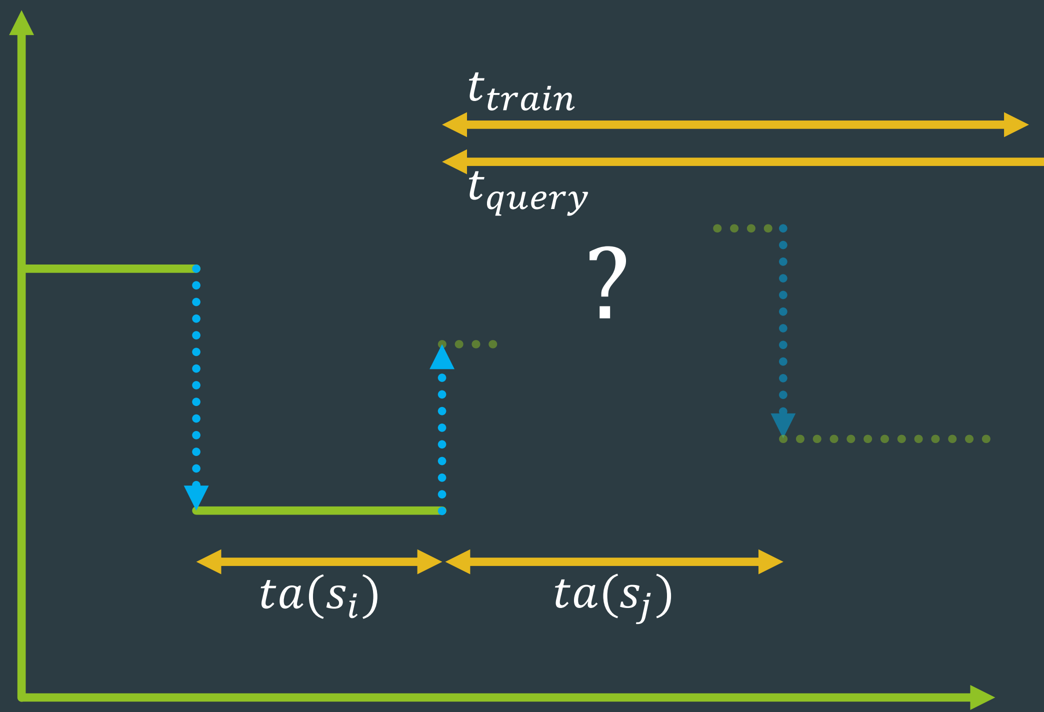
Pattern 3: Multiple Timers



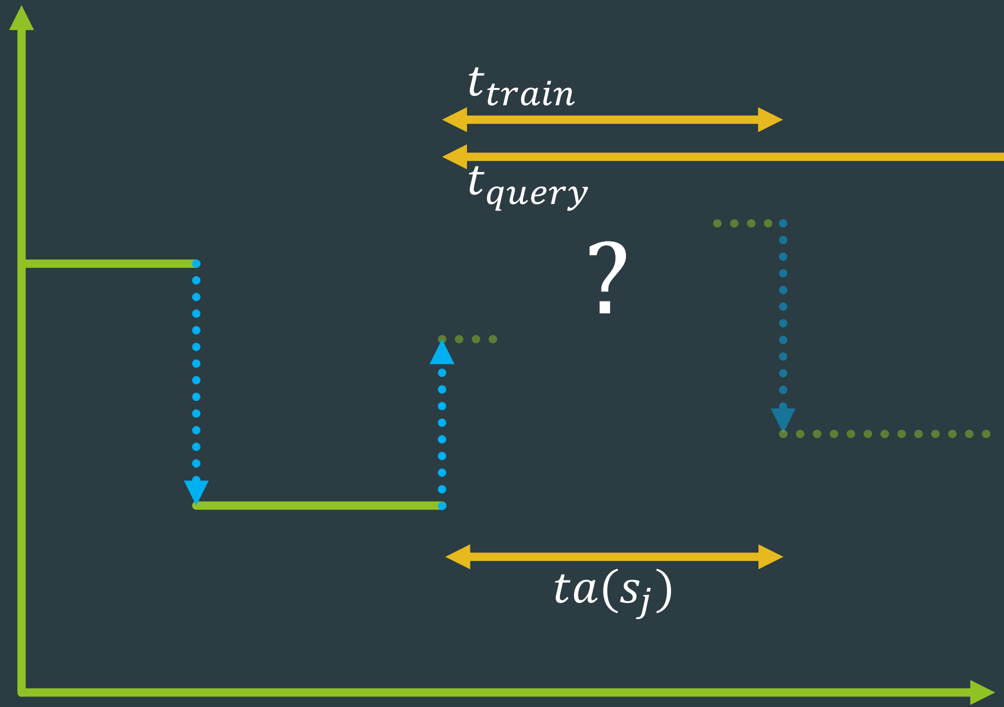
Pattern 3: Multiple Timers



Pattern 3: Multiple Timers



Pattern 3: Multiple Timers

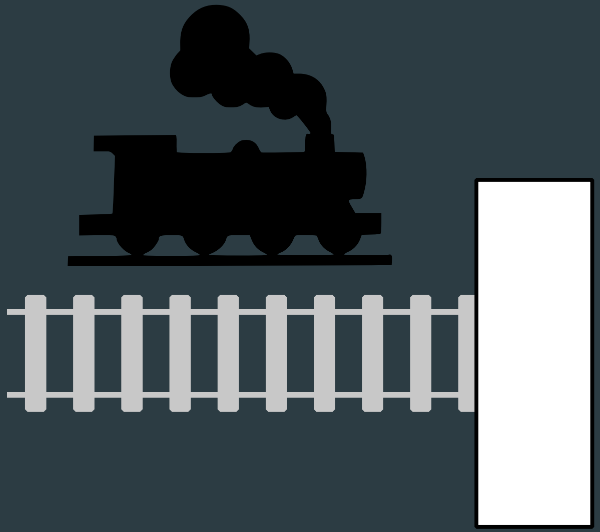


Pattern 3: Multiple Timers

```
class RailwaySegmentState:
    def __init__(self):
        self.t_query = INFINITY
        self.t_train = INFINITY

class RailwaySegment(AtomicDEVS):
    def timeAdvance(self):
        return min(self.state.t_query, \
                   self.state.t_train)

    def extTransition(self, inputs):
        self.state.t_query -= self.elapsed
        self.state.t_train -= self.elapsed
    …
```

```
    …

    def intTransition(self):
        self.state.t_query -= self.timeAdvance()
        self.state.t_train -= self.timeAdvance()
        if (self.state.t_query == 0):
            … # process query
        elif (self.state.t_train == 0):
            … # process train
```

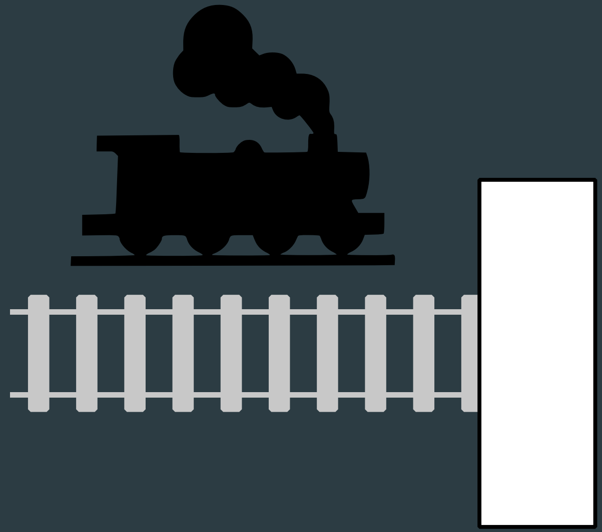
Pattern 4: Statistics Gathering



n trains = n doubles = $8n$ bytes

[1815.7; 1901.1; 1801.4; 1867.3; 1911.8; 1846.4; 1844.3; 1873.5; …]

Pattern 4: Statistics Gathering



n trains = n doubles = $8n$ bytes

[1815.7; 1901.1; 1801.4; 1867.3[illegible]1.8; 1846.4; 1844.3; 1873.5; …]

$$sum = \sum_i t_{travel} = 36{,}902{,}140.45$$

$$count = |arrivals| = 20{,}000$$

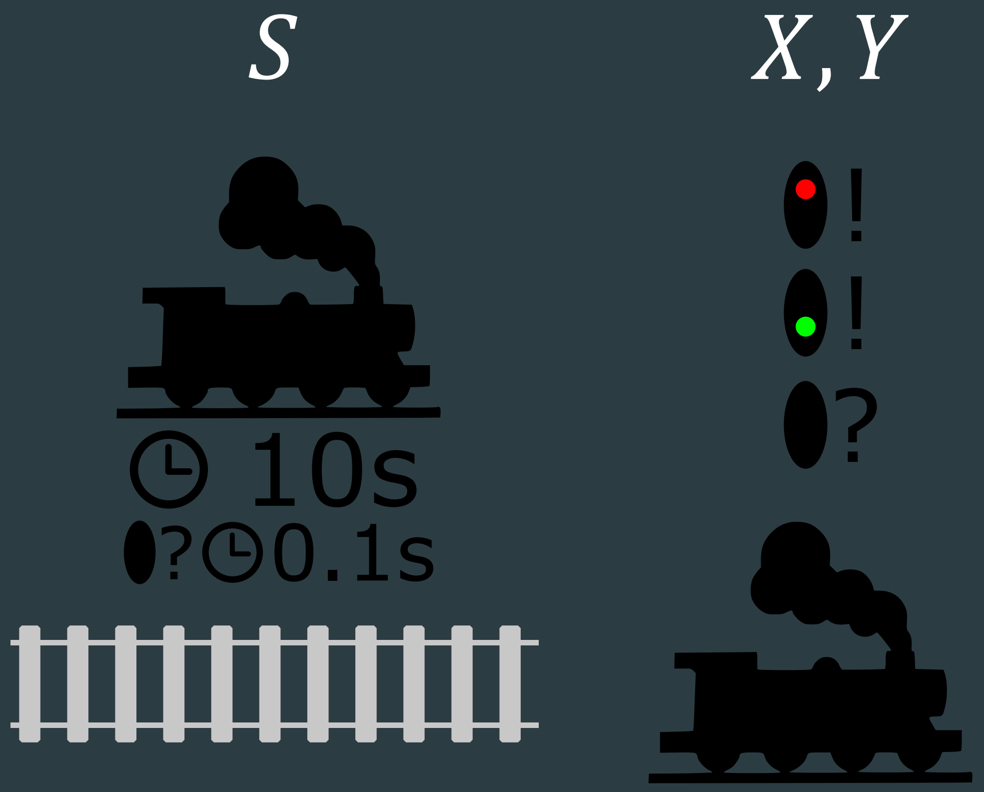
$$avg = \frac{sum}{count}$$

n trains = 1 double + 1 long long int = 16 bytes

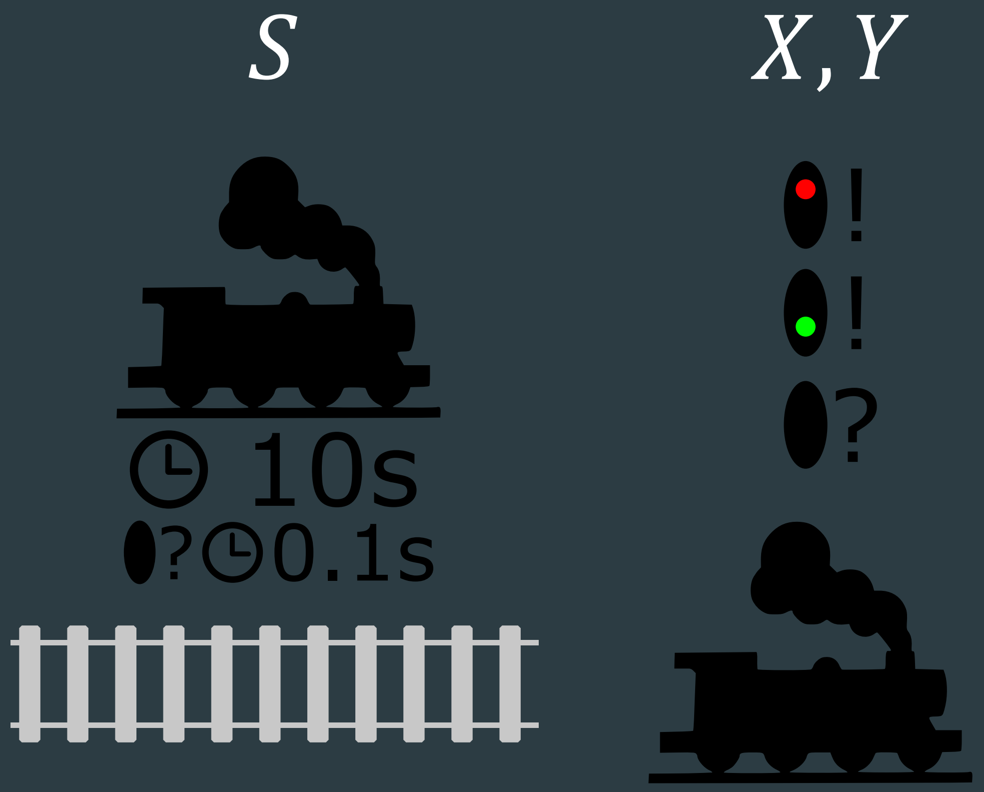
Pattern 4: Statistics Gathering

```python
class CollectorState():
    def __init__(self):
        self.counter = 0
        self.accum = 0.0
        self.t_sim = 0.0

class Collector(AtomicDEVS):
    def extTransition(self, inputs):
        left = inputs[self.train].left
        transit = self.state.t_sim - left
        self.accum += transit
        self.counter += 1
```

Pattern 5: Complex State/Event



Pattern 5: Complex State/Event



$$S = \{\langle timer_{train}, timer_{query}, v_{train}, t_{train}, a_{train}\rangle | timer_{train} \in \mathbb{R}^{+}, \dots\}$$

$$X, Y = \{\langle t, v, a\rangle | t \in \mathbb{R}^{+}, \dots\} \cup \{\text{query}\} \cup \{colour | colour \in \{red, green\}\}$$

Pattern 5: Complex State/Event

```python
class RailwaySegmentState:
    def __init__(self):
        self.train = None

class RailwaySegment(AtomicDEVS):
    def __init__(self):
        self.state = RailwaySegmentState()
        …

    def extTransition(self, inputs):
        …
        self.state.train = inputs[self.train_in]
        …
```

```python
class Train:
    def __init__(self, t, a):
        self.t = t
        self.a = a
        self.v = 0.0

class Query:
    def __init__(self):
        pass

class QueryAck:
    def __init__(self, colour):
        self.colour = colour
```

What’s left ?

DEVS Semantics

	Operational Semantics	Denotational Semantics
Atomic DEVS	Abstract Simulator	modal Discrete Event Logic L_{DE} [1]
Coupled DEVS	Hierarchical Simulator	Closure under Coupling

[1] Ashvin Radiya and Robert G. Sargent. A logic-based foundation of discrete event modeling and simulation. ACM Transactions on Modeling and Computer Simulation, 1(1):3-51, 1994.

Limitations of Classic DEVS

- Parallel implementation
  - Parallel DEVS [1]
- Select function is artificial
  - Parallel DEVS [1]
- Dynamic Structure systems
  - Dynamic Structure DEVS [2]

[1] A.C.-H. Chow. Parallel DEVS: A parallel, hierarchical, modular modeling formalism and its distributed simulator. Transactions of the Society for Computer Simulation International, 13(2):55-68, 1996.
[2] F. Barros. The dynamic structure discrete event system specification formalism. Transactions of the Society for Computer Simulation International, 13(1):35-46, 1996.

Formalisms
- Dynamic Structure
- Real-time
- Cell DEVS
- Verification

Standardization
- Tools
- Languages
- Interoperable

Performance
- Algorithms
- Activity
- Distribution
- Parallel

Model libraries
- Example
- Reusable

Applications

Quick search

Go

Enter search terms or a module, class or function name.

Examples

A small *trafficModel* and corresponding *trafficExperiment* file is included in the *examples* folder of the PyPDEVS distribution. This (completely working) example is slightly too big to use as a first introduction to PyPDEVS and therefore this page will start with a very simple example.

For this, we will first introduce a simplified queue model, which will be used as the basis of all our examples. The complete model can be downloaded: `queue_example_classic.py`.

This section should provide you with all necessary information to get you started with creating your very own PyPDEVS simulation. More advanced features are presented in the next section.

Generator

Somewhat simpler than a queue even, is a generator. It will simply create a message to send after a certain delay and then it will stop doing anything.

Informally, this would result in a DEVS specification as:

- Time advance function returns the waiting time to generate the message, infinity after the message was created
- Output function outputs the generated message
- External transition function will never happen (as there are no inputs)

http://msdl.cs.mcgill.ca/projects/PythonPDEVS

An overview of PythonPDEVS

Yentl Van Tendeloo[1] Hans Vangheluwe[1,2]
[1] University of Antwerp, Belgium
[2] McGill University, Canada

Yentl.VanTendeloo@uantwerpen.be
Hans.Vangheluwe@uantwerpen.be

Methodology

Simulation

Simulation: Transactions of the Society for Modeling and Simulation International
1–19

DOI: 10.1177/0037549716678330
sim.sagepub.com
SAGE SCS

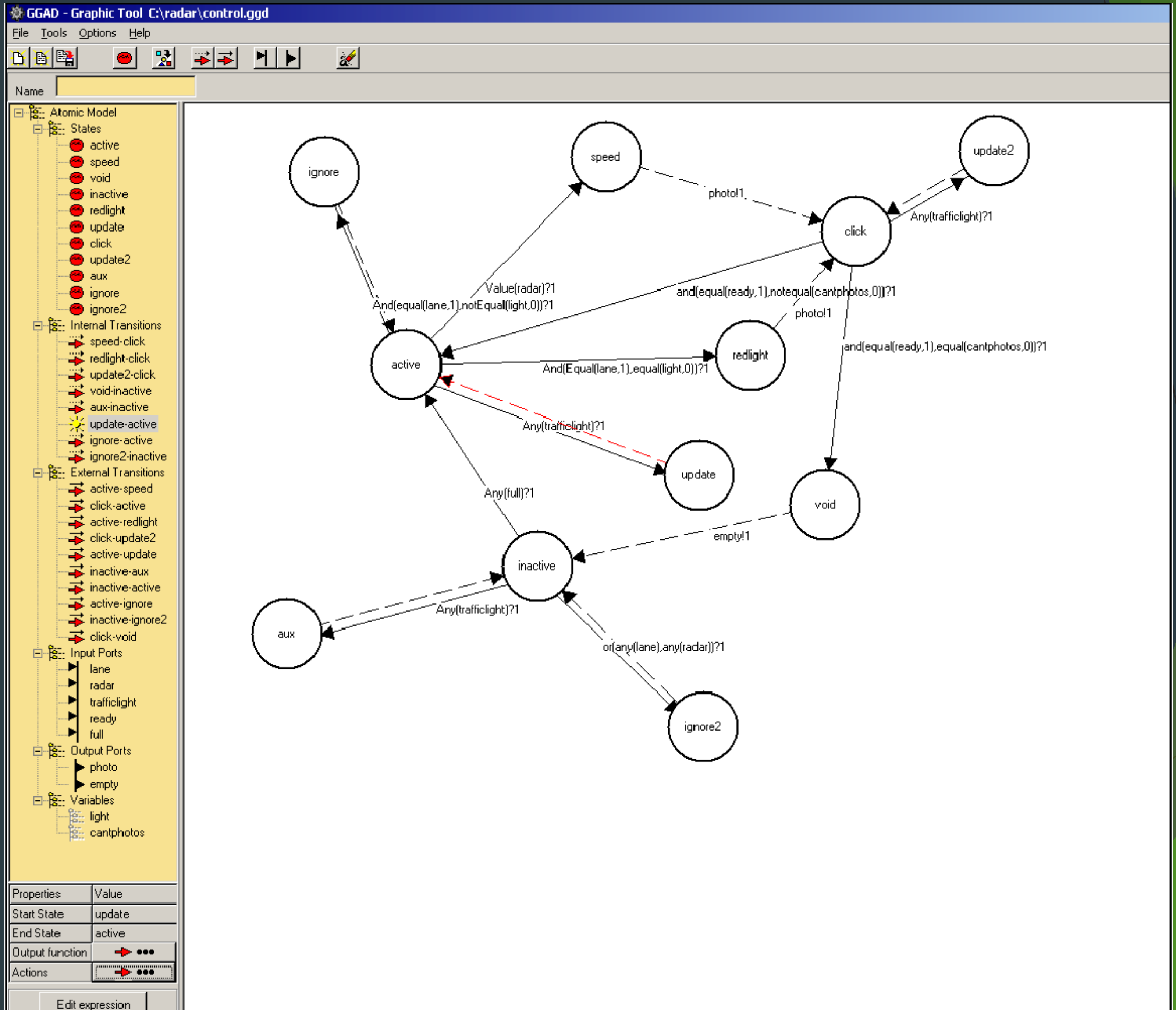
An evaluation of DEVS simulation tools

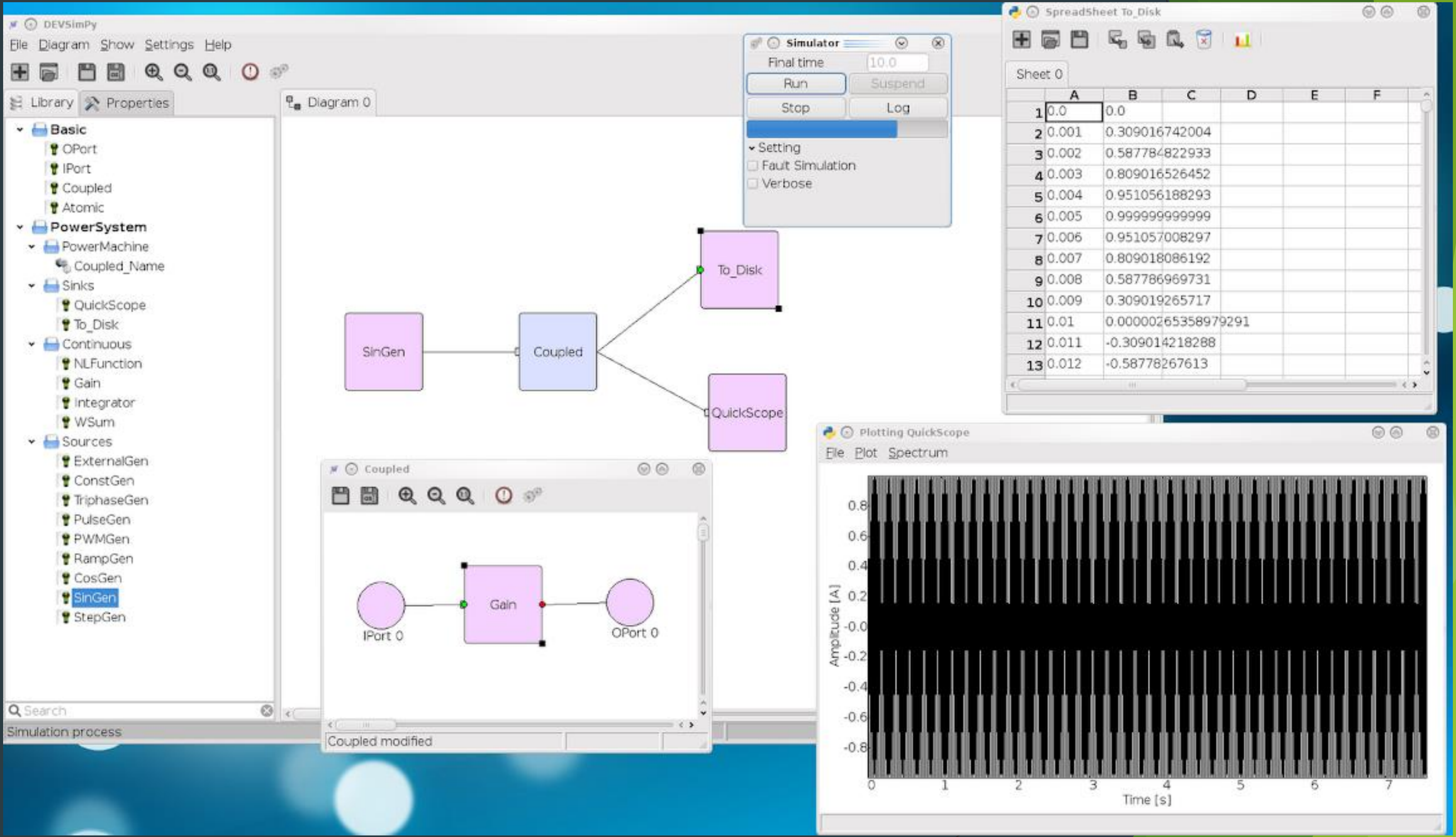
Yentl Van Tendeloo[1*] and Hans Vangheluwe[1,2,3*]

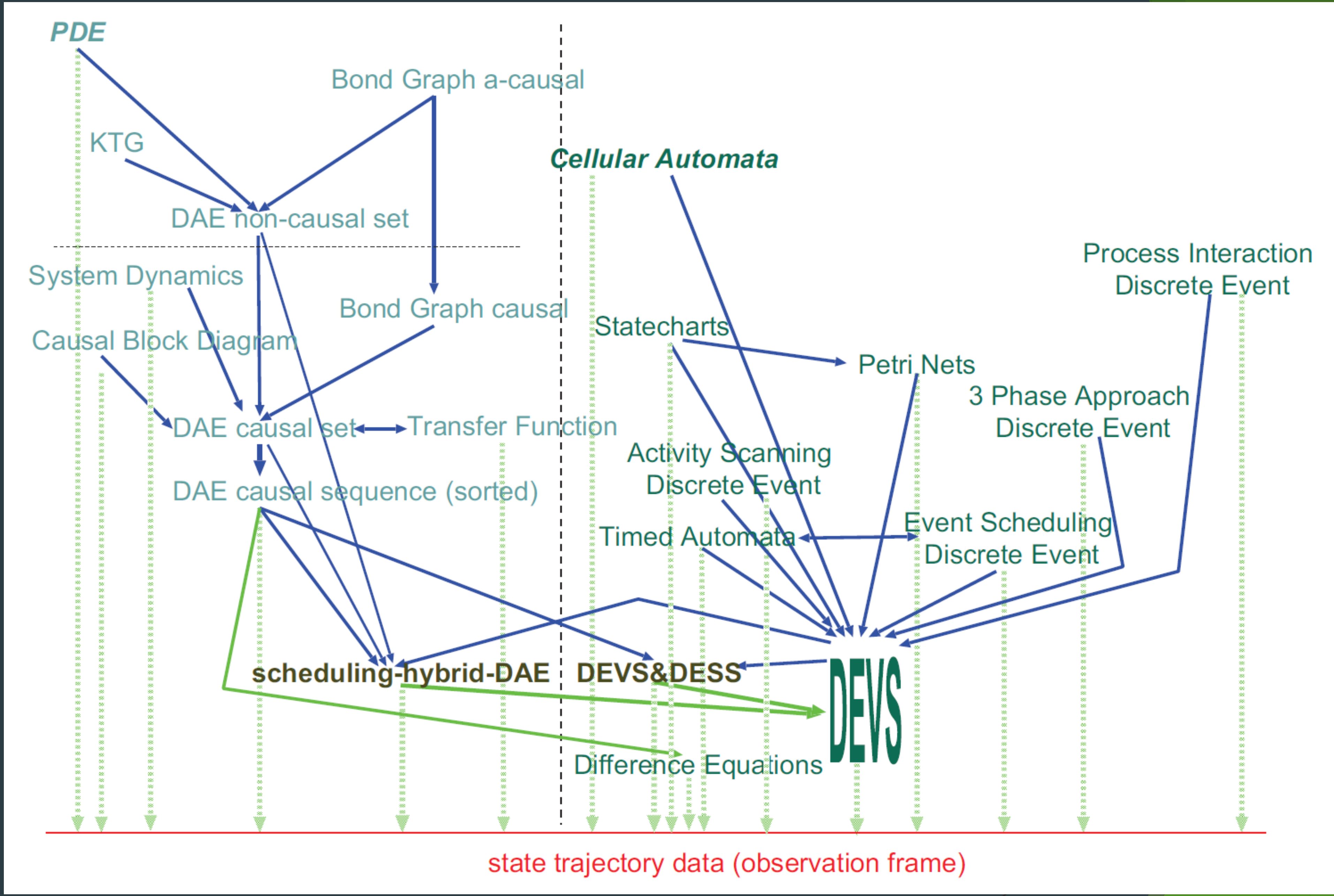
Abstract
DEVS is a popular formalism for modeling complex dynamic systems using a discrete-event abstraction. Owing to its popularity, and the simplicity of the simulation kernel, a number of tools have been constructed by academia and industry. However, each of these tools has distinct design goals and a specific programming language implementation. Consequently, each supports a specific set of formalisms, combined with a specific set of features. Performance differs significantly between different tools. We provide an overview of the current state of eight different DEVS simulation tools: ADEVS, CD++, DEVS-Suite, MS4 Me, PowerDEVS, PythonPDEVS, VLE, and X-S-Y. We compare supported formalisms, compliance, features, and performance. This paper aims to help modelers in deciding which tools to use to solve their specific problems. It further aims to help tool builders, by showing the aspects of their tools that could be extended in future tool versions.

Yentl Van Tendeloo and Hans Vangheluwe.
An Overview of PythonPDEVS.
In Proceedings of Journées DEVS Francophones (JDF), pages 59-66, 2016.

Yentl Van Tendeloo and Hans Vangheluwe.
An Evaluation of DEVS Simulation Tools.
Simulation: Transactions of the Society for Modeling and Simulation International.
2017, 93(2): 103-121


GGAD - Graphic Tool C:\radar\control.ggd
File Tools Options Help
Name
Atomic Model
States
active
speed
void
inactive
redlight
update
click
update2
aux
ignore
ignore2
Internal Transitions
speed-click
redlight-click
update2-click
void-inactive
aux-inactive
update-active
ignore-active
ignore2-inactive
External Transitions
active-speed
click-active
active-redlight
click-update2
active-update
inactive-aux
inactive-active
active-ignore
inactive-ignore2
click-void
Input Ports
lane
radar
trafficlight
ready
full
Output Ports
photo
empty
Variables
light
cantphotos
Properties
Value
Start State
update
End State
active
Output function
Actions
Edit expression
ignore
speed
update2
photo!1
click
Any(trafficlight)?1
Value(radar)?1
And(equal(lane,1),notEqual(light,0))?1
and(equal(ready,1),notequal(cantphotos,0))?1
photo!1
redlight
and(equal(ready,1),equal(cantphotos,0))?1
active
And(Equal(lane,1),equal(light,0))?1
Any(trafficlight)?1
update
void
Any(full)?1
empty!1
inactive
aux
Any(trafficlight)?1
or(any(lane),any(radar))?1
ignore2


DEVSimPy
File Diagram Show Settings Help
Library Properties
Basic
OPort
IPort
Coupled
Atomic
PowerSystem
PowerMachine
Coupled_Name
Sinks
QuickScope
To_Disk
Continuous
NLFunction
Gain
Integrator
WSum
Sources
ExternalGen
ConstGen
TriphaseGen
PulseGen
PWMGen
RampGen
CosGen
SinGen
StepGen
Search
Simulation process
Diagram 0
SinGen
Coupled
To_Disk
QuickScope
Simulator
Final time
10.0
Run
Suspend
Stop
Log
Setting
Fault Simulation
Verbose
Coupled
IPort 0
Gain
OPort 0
Coupled modified
SpreadSheet To_Disk
Sheet 0
A
B
C
D
E
F
1 0.0 0.0
2 0.001 0.309016742004
3 0.002 0.587784822933
4 0.003 0.809016526452
5 0.004 0.951056188293
6 0.005 0.999999999999
7 0.006 0.951057008297
8 0.007 0.809018086192
9 0.008 0.587786969731
10 0.009 0.309019265717
11 0.01 0.00000265358979291
12 0.011 -0.309014218288
13 0.012 -0.58778267613
Plotting QuickScope
File Plot Spectrum
Amplitude [A]
Time [s]



Hans Vangheluwe. DEVS as a common denominator for multi-formalism hybrid systems modelling.
In proceedings of the International Symposium on Computer-Aided Control System Design, pp. 129-134. 2000.

Hans.Vangheluwe@uantwerpen.be