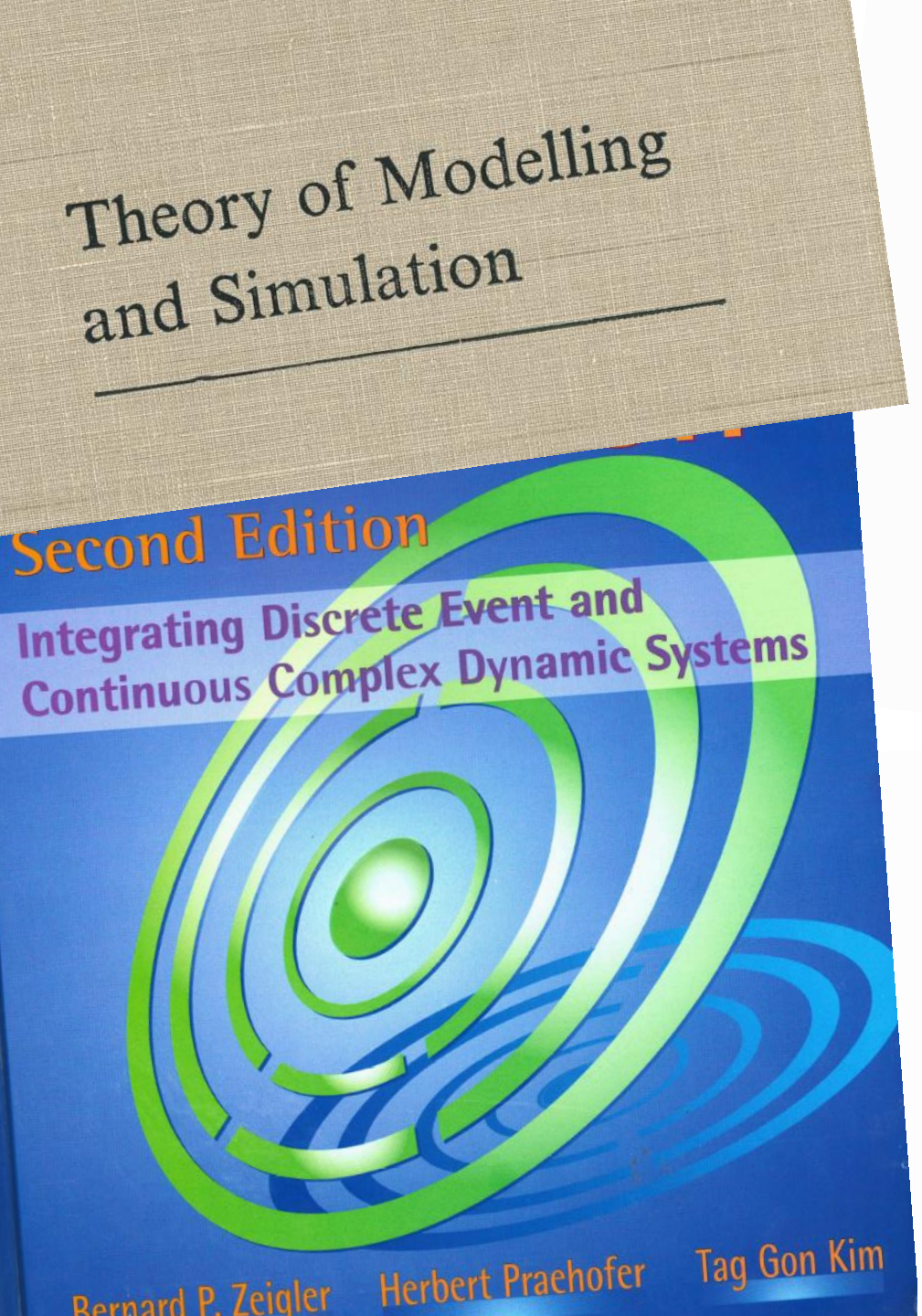
Parallel DEVS

An Introduction Using PythonPDEVS

Yentl Van Tendeloo, Hans Vangheluwe

Introduction



Parallel DEVS: A parallel, hierarchical, modular modeling formalism and its distributed simulator

A. C.-H. Chow

IBM Corporation, P.O. Box 180124, Austin, Texas 78758
Tel: 512-838-8069 FAX: 512-250-2346 E-mail: alexc@austin.ibm.com

We present a parallel, hierarchical, modular Discrete Event System Specification (P-DEVS) modeling formalism which provides a modeler with both conceptual and parallel execution benefits. The parallel formalism distinguishes between transition collisions and ordinary external events in the external transition function of DEVS models. Such separation enables us to extend the modeling capability of the collisions. The formalism also does away with the necessity for tie-breaking of simultaneously scheduled events, as embodied in the select function. We next present a design for the parallel simulation procedures needed to prove the formalism's soundness and to serve as a reference for implementation. We then discuss a prototype implementation that affords a high degree of flexibility by mechanizing the "closure under coupling" property of the Parallel DEVS formalism and the object-oriented characteristic

Keywords: System specification, discrete event simulat
simulation, distributed/parallel simulati

Abstract Simulator for the Parallel DEVS Formalism

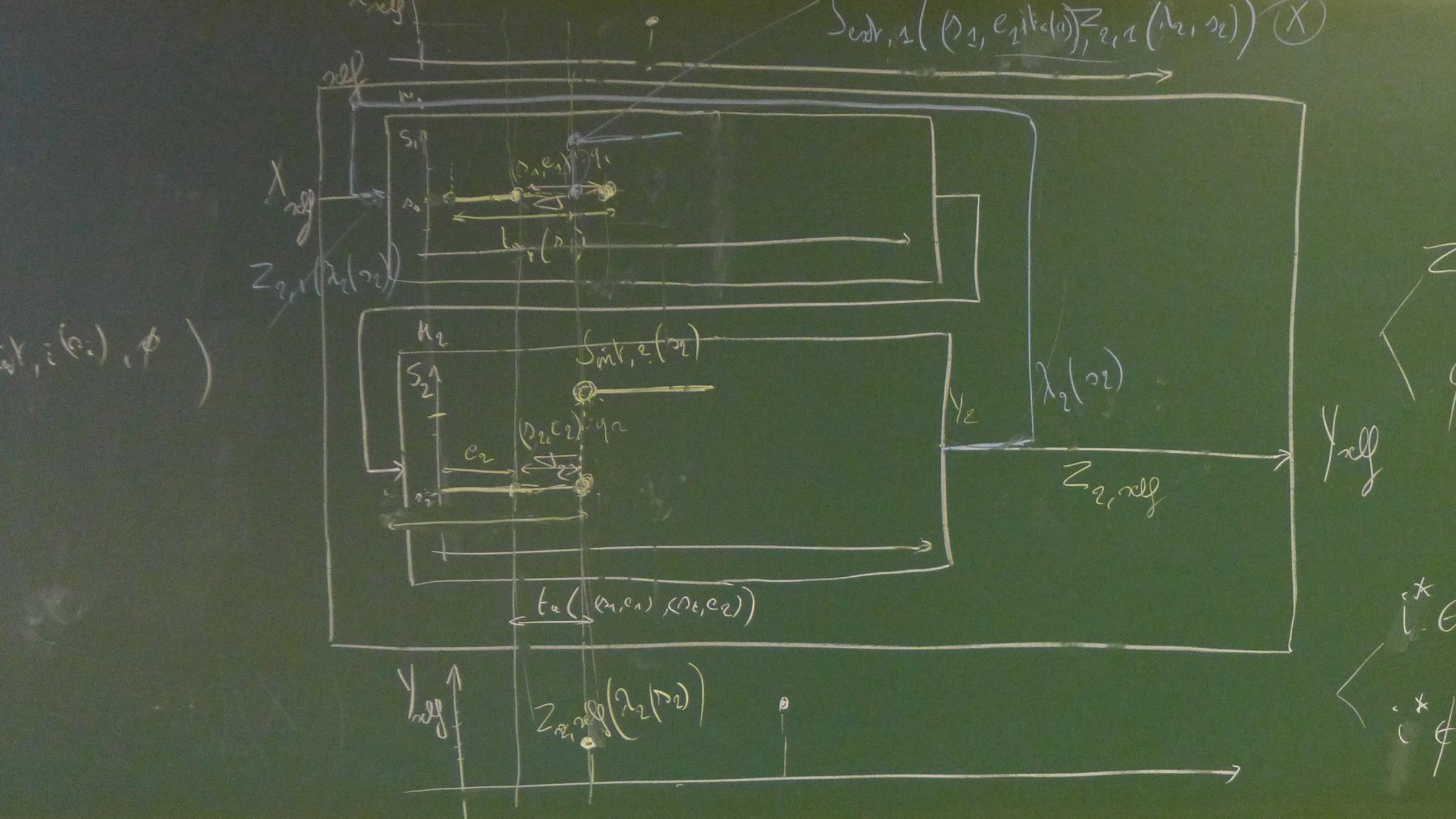
Alex ChungHen Chow

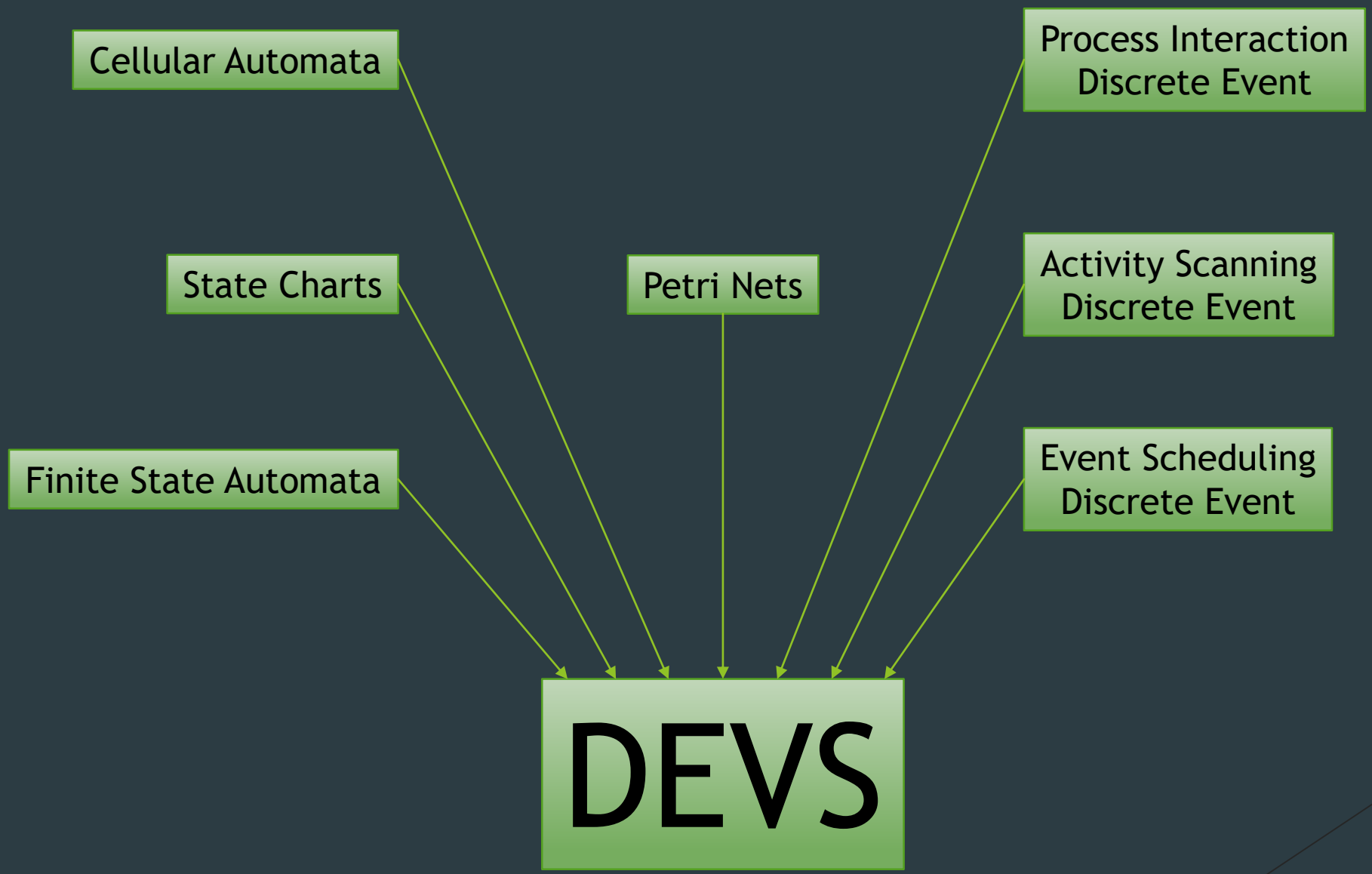
Object Technology Products

IBM Corp.
Austin, TX 78758
alexc@austin.ibm.com

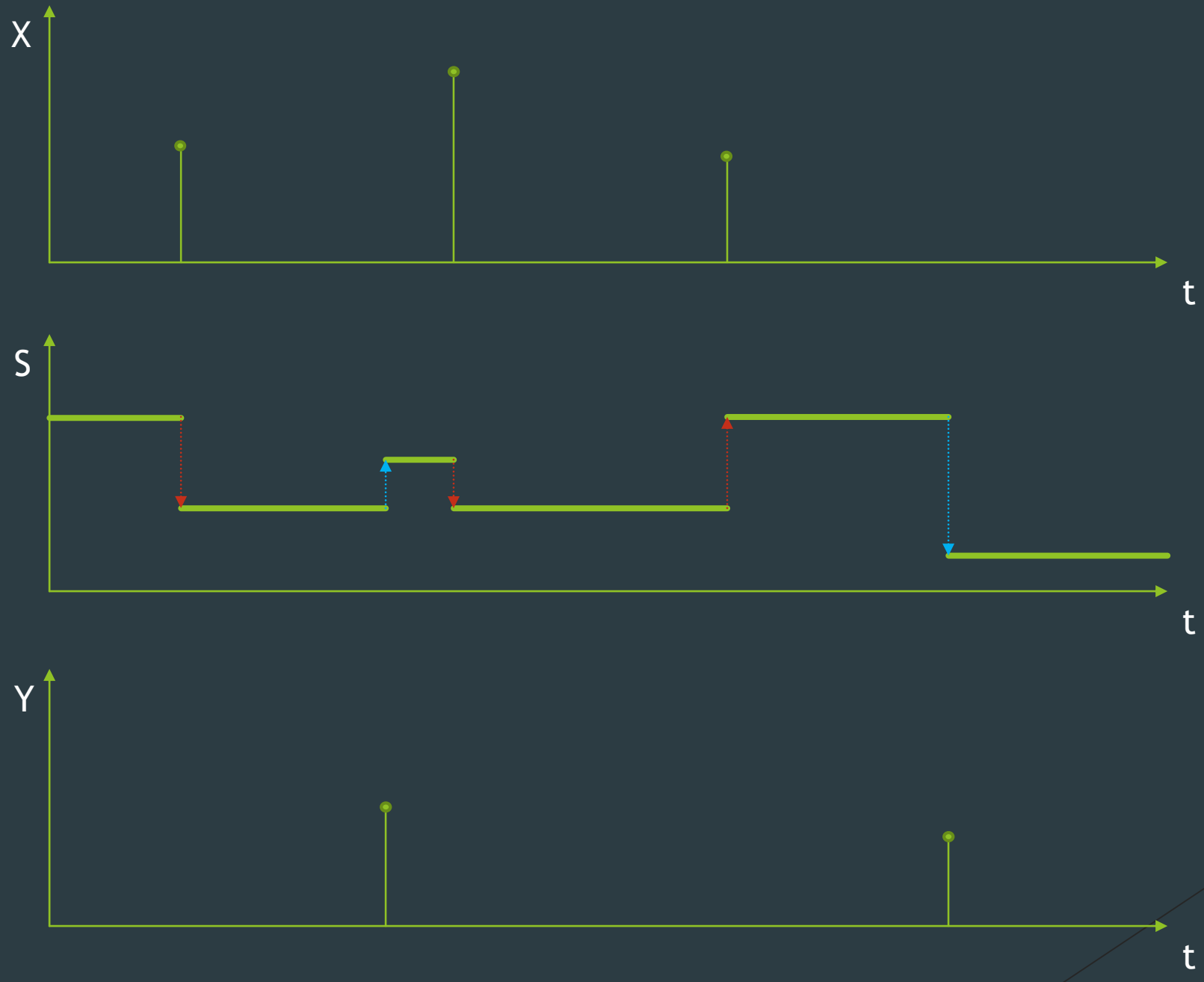
Bernard P. Zeigler
Doo Hwan Kim

Department of
Electrical and Computer Engineering
The University of Arizona
Tucson, AZ 85721
zeigler@ece.arizona.edu
dhkim@ece.arizona.edu


self
M_1
S_1
λ_{self}
(σ_1, e_1)
q_1
$t_{a,1}(\sigma_1)$
$Z_{2,1}(\lambda_2(\sigma_2))$
M_2
S_2
σ_2
e_2
(σ_2, e_2)
q_2
$t_{a,2}(\sigma_2)$
Y_2
$\lambda_2(\sigma_2)$
$Z_{2,self}$
Y_{self}
$t_a((\sigma_1,e_1),(\sigma_2,e_2))$
Y_{self}
$Z_{2,self}(\lambda_2(\sigma_2))$


Cellular Automata
State Charts
Finite State Automata
Petri Nets
Process Interaction Discrete Event
Activity Scanning Discrete Event
Event Scheduling Discrete Event
DEVS

Experimentation

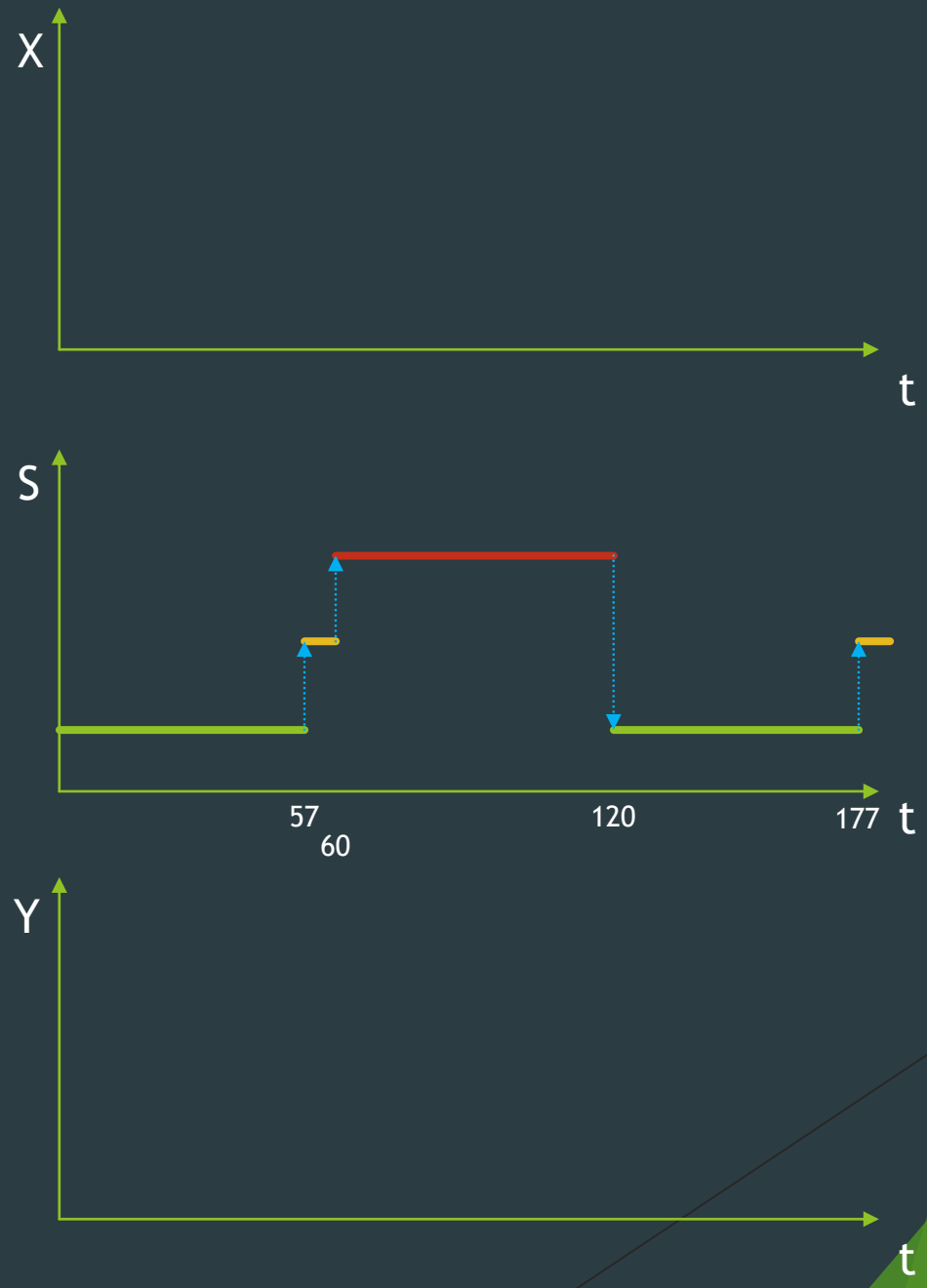

X
t
S
t
Y
t

simple_experiment.py

```
from pypdevs.simulator import Simulator

from mymodel import MyModel

model = MyModel()
simulator = Simulator(model)

simulator.setVerbose()

simulator.simulate()
```

Atomic Models


Red
60s
Yellow
3s
Green
57s


X
t
S
57
60
120
177
t
Y
t

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

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 → 60,
green → 57,
yellow → 3}

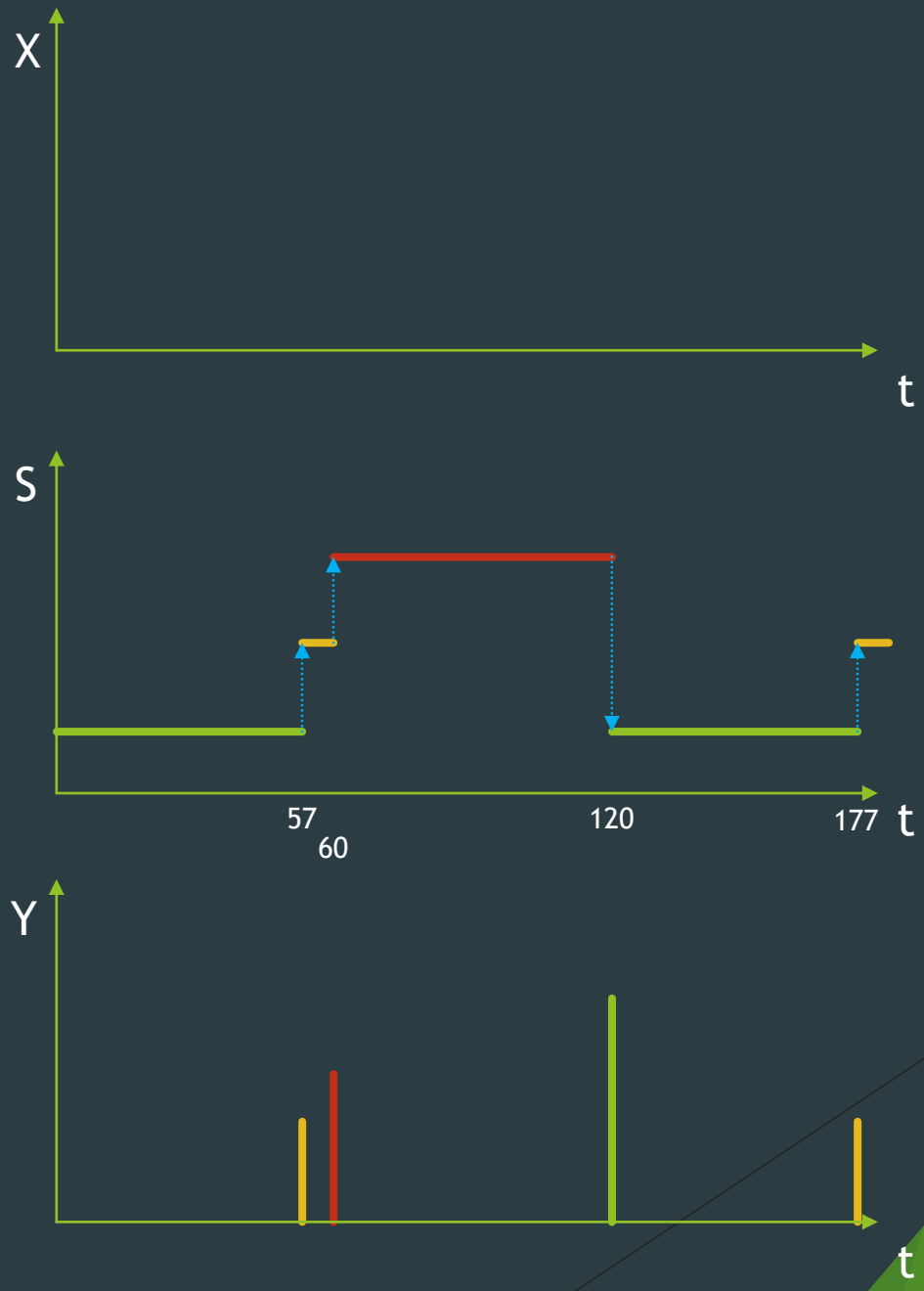
S = {red, yellow, green}
δ_{int} = { red → green,
green → yellow,
yellow → red}
ta = {red → 60,
green → 57,
yellow → 3}

```
time = 0
current_state = green
while True:
    time += ta(current_state)
    current_state = δ_int(current_state)
```

atomic_int.py

```
from pypdevs.DEVS import *

class TrafficLightAutonomous(AtomicDEVS):
    def __init__(self):
        AtomicDEVS.__init__(self, “Light”)
        self.state = “green”

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

    def timeAdvance(self):
        state = self.state
        return {“red”: 60,
                “yellow”: 3,
                “green”: 57}[state]
```


Red
60s
!red
Yellow
3s
!green
!yellow
Green
57s


X
t
S
57
60
120
177
t
Y
t

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

S = {red, yellow, green}

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

ta = {red → 60,
green → 57,
yellow → 3}

Y : set of output events

Y = {“red”, “green”, “yellow”}

$\lambda : S \rightarrow Y^b$

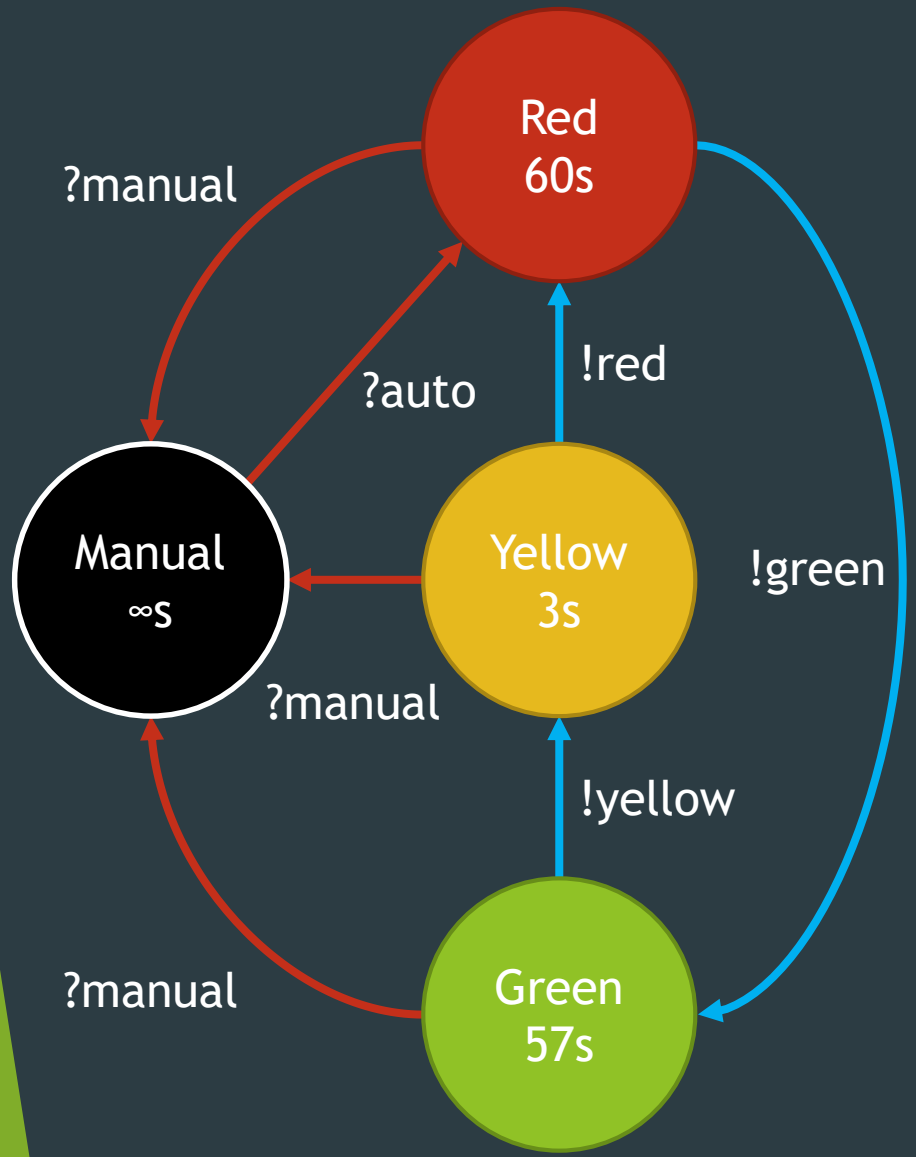
λ = { green → [“yellow”],
yellow → [“red”],
red → [“green”]}

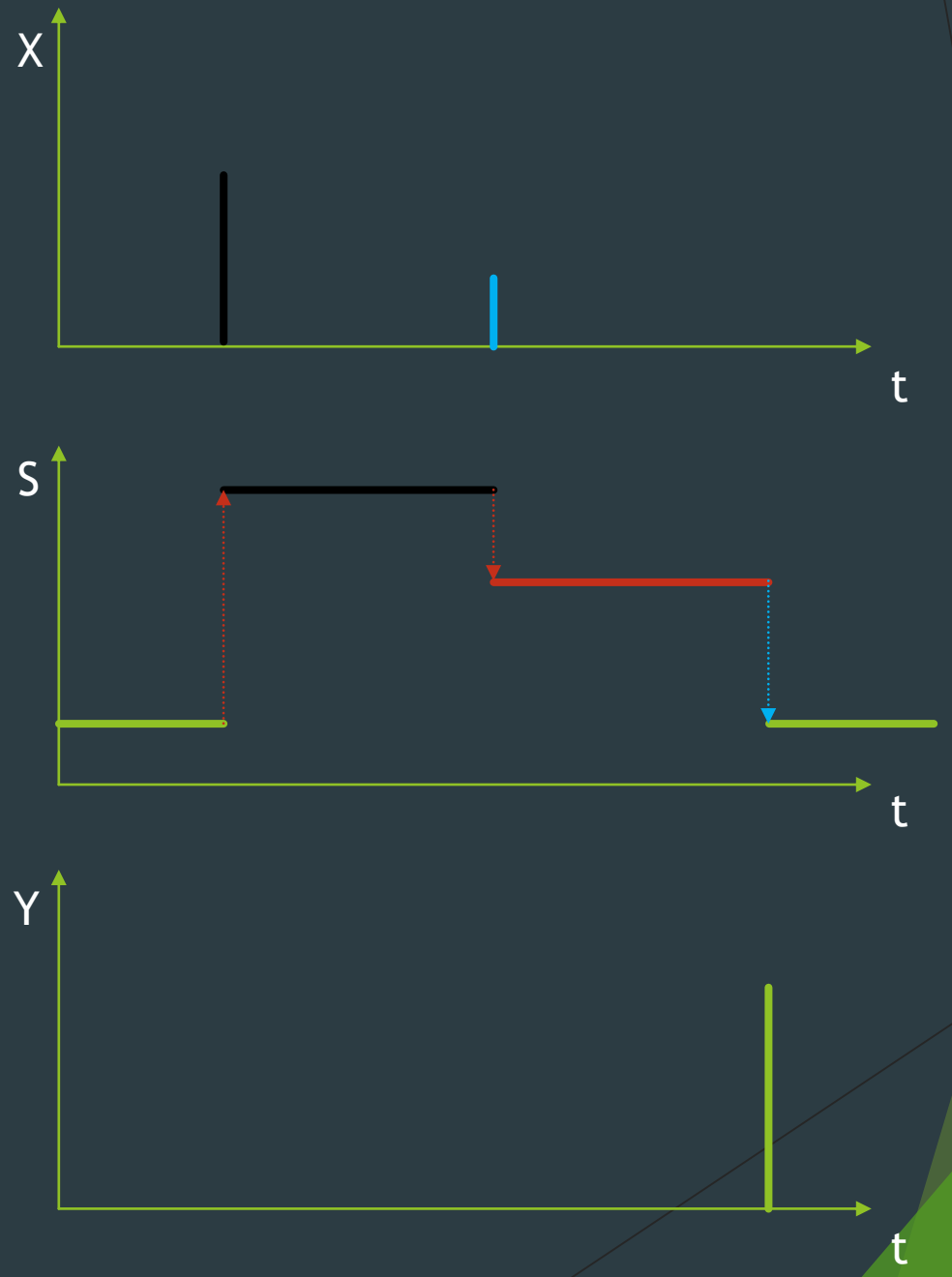
S = {red, yellow, green}
δ_{int} = { red → green,
green → yellow,
yellow → red}
ta = {red → 60,
green → 57,
yellow → 3}
Y = {“red”, “green”, “yellow”}
λ = {green → [“yellow”],
yellow → [“red”],
red → [“green”]}

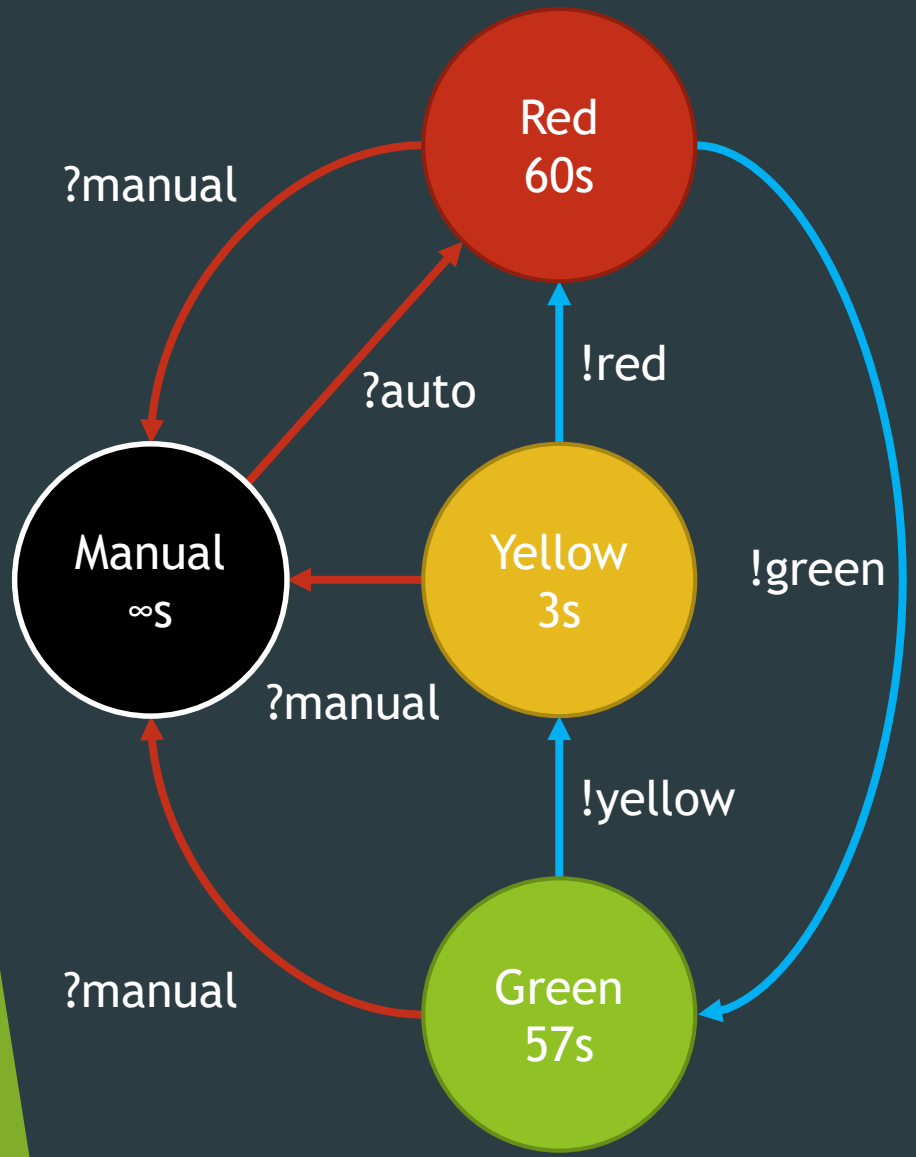
```
time = 0
current_state = green
while True:
    time += ta(current_state)
    output(λ(current_state))
    current_state = δ_int(current_state)
```

atomic_out.py

```
from pypdevs.DEVS import *

class TrafficLightWithOutput(AtomicDEVS):
    def __init__(self):
        AtomicDEVS.__init__(self, “light”)
        self.state = “green”
        self.observe = self.addOutPort(“observer”)

    …

    def outputFnc(self):
        state = self.state
        if state == “red”:
            v = “green”
        elif state == “yellow”:
            v = “red”
        elif state == “green”:
            v = “yellow”
        return {self.observe: [v]}
```


Red
60s
?manual
?auto
!red
Manual
∞s
Yellow
3s
!green
?manual
!yellow
?manual
Green
57s


X
t
S
t
Y
t



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

Y = {“red”, “green”, “yellow”}

S = {red, yellow, green, manual}

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

λ = {green → [“yellow”],
yellow → [“red”],
red → [“green”]}

ta = {red → 60,
green → 57,
yellow → 3,
manual → ∞}

X : set of input events

X = {“auto”, “manual”}

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

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

δ_{ext} = {((*, *), [“manual”]) → “manual”,
((“manual”, *), [“auto”]) → “red”}

Y = {“red”, “green”, “yellow”}
S = {red, yellow, green, manual}
δ_{int} = {red → green,
green → yellow,
yellow → red}
λ = {green → [“yellow”],
yellow → [“red”],
red → [“green”]}
ta = {red → 60,
green → 57,
yellow → 3,
manual → ∞}
X = {“auto”, “manual”}
δ_{ext} = {((*, *), [“manual”]) → “manual”,
((“manual”, *), [“auto”]) → “red”}

```
time = 0
cur_state = green
while True:
    next_time = time + ta(cur_state)
    if time_next_ev <= next_time:
        cur_state = δ_ext((cur_state, e), next_ev)
        time = time_next_ev
    else:
        time = next_time
        output(λ(current_state))
        current_state = δ_int(current_state)
```

Y = {“red”, “green”, “yellow”}
S = {red, yellow, green, manual}
δ_{int} = {red → green,
green → yellow,
yellow → red}
λ = {green → [“yellow”],
yellow → [“red”],
red → [“green”]}
ta = {red → 60,
green → 57,
yellow → 3,
manual → ∞}
X = {“auto”, “manual”}
δ_{ext} = {((*, *), [“manual”]) → “manual”,
((“manual”, *), [“auto”]) → “red”}

atomic_ext.py

```
from pypdevs.DEVS import *

class TrafficLight(AtomicDEVS):
 def __init__(self):
        AtomicDEVS.__init__(self, “light”)
        self.state = “green”
        self.observe = self.addOutPort(“observer”)
        self.interrupt = self.addInPort(“interrupt”)

    …

    def extTransition(self, inputs):
        inp = inputs[self.interrupt][0]
        if inp == “manual”:
            return “manual”
        elif inp == “auto”:
            if self.state == “manual”:
                return “red”
```

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

X : set of input events

Y : set of output events

S : set of sequential states

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

$\delta_{ext}: \mathrm{Q} \times X^b \to S$

$\lambda : S \to Y^b$

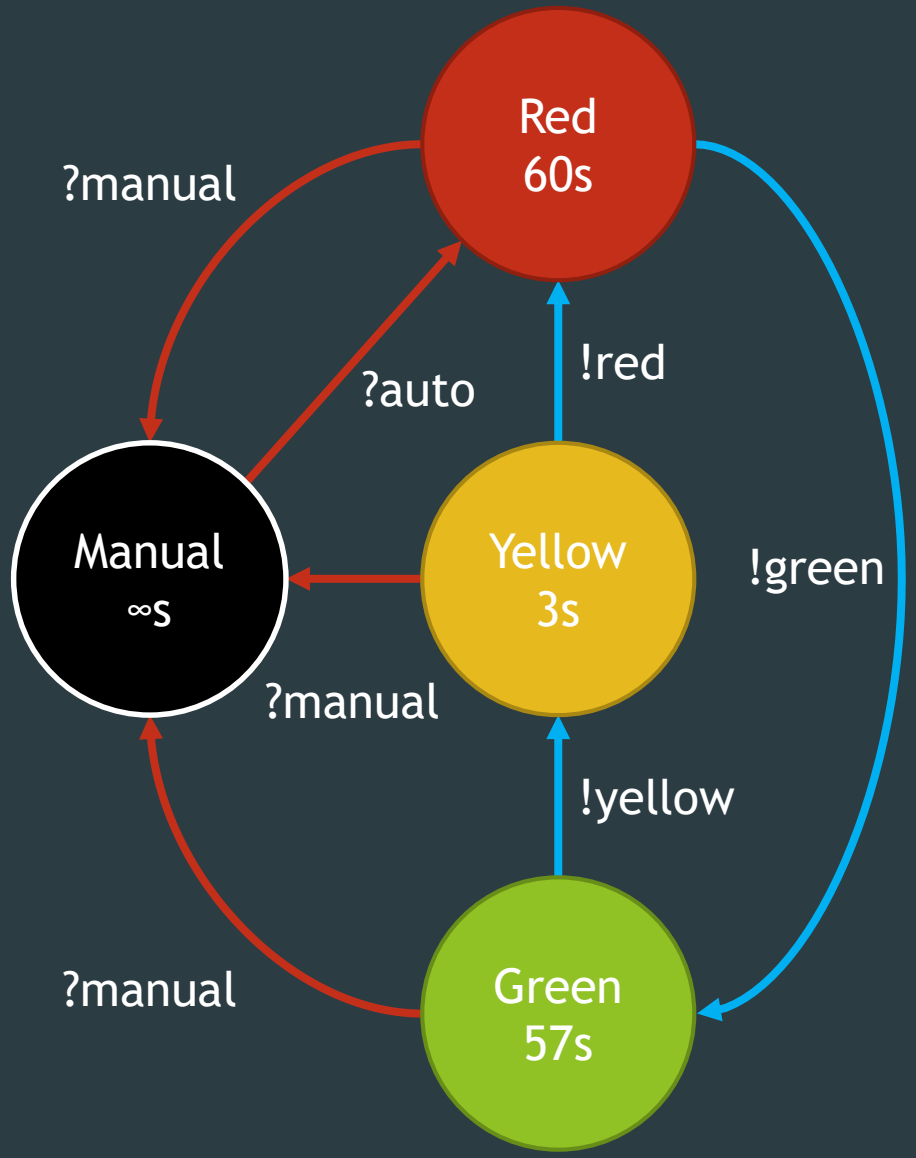
$ta : \mathrm{S} \to \mathbb{R}_{0,+\infty}$

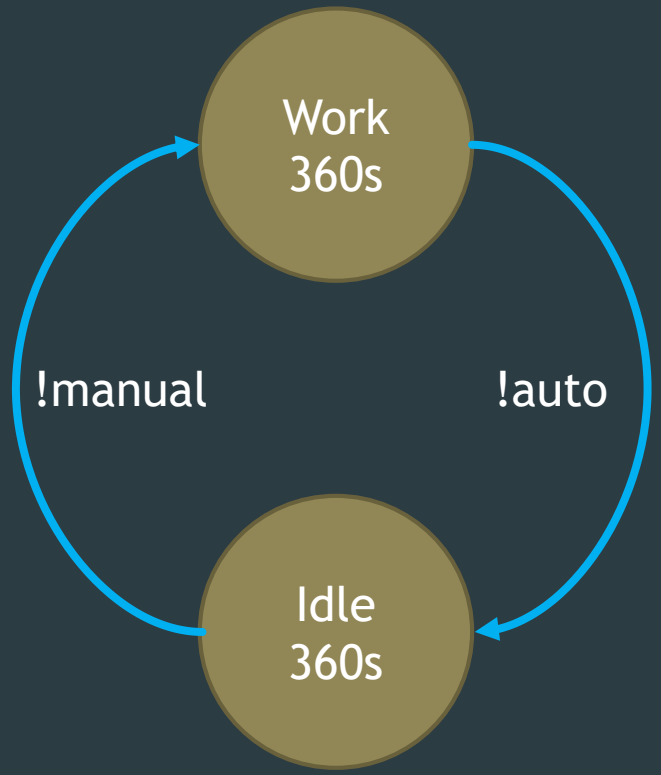
$\delta_{conf} : S \times X^b \to S$

atomic_conf.py

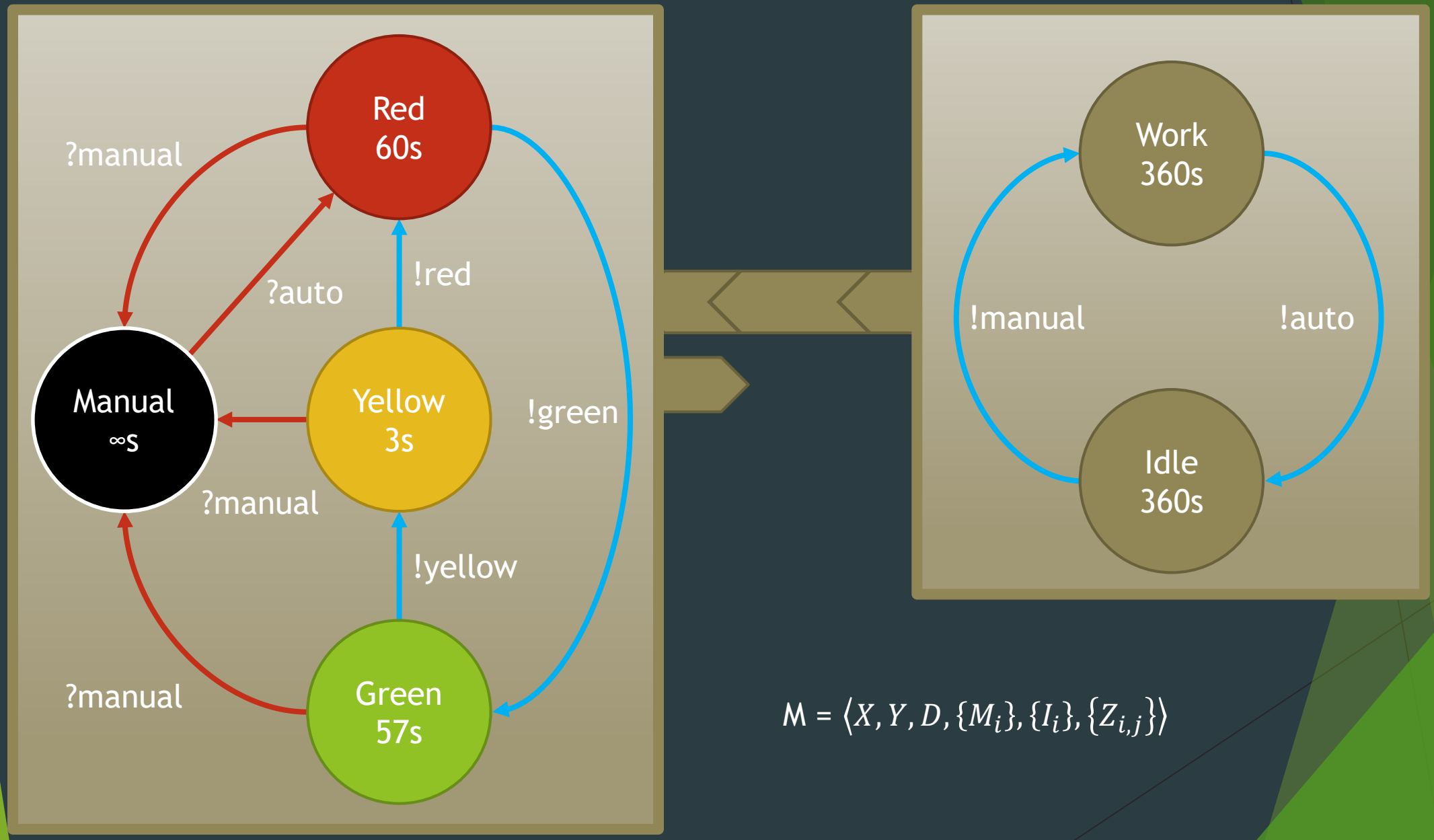
```
from pypdevs.DEVS import *

class TrafficLight(AtomicDEVS):
    …

    def confTransition(self, inputs):
        self.elapsed = 0.0
        self.state = self.intTransition()
        self.state = self.extTransition(inputs)
        return self.state
```

Coupled Models


Red
60s
?manual
?auto
!red
Manual
∞s
Yellow
3s
!green
?manual
!yellow
?manual
Green
57s


Work
360s
!manual
!auto
Idle
360s

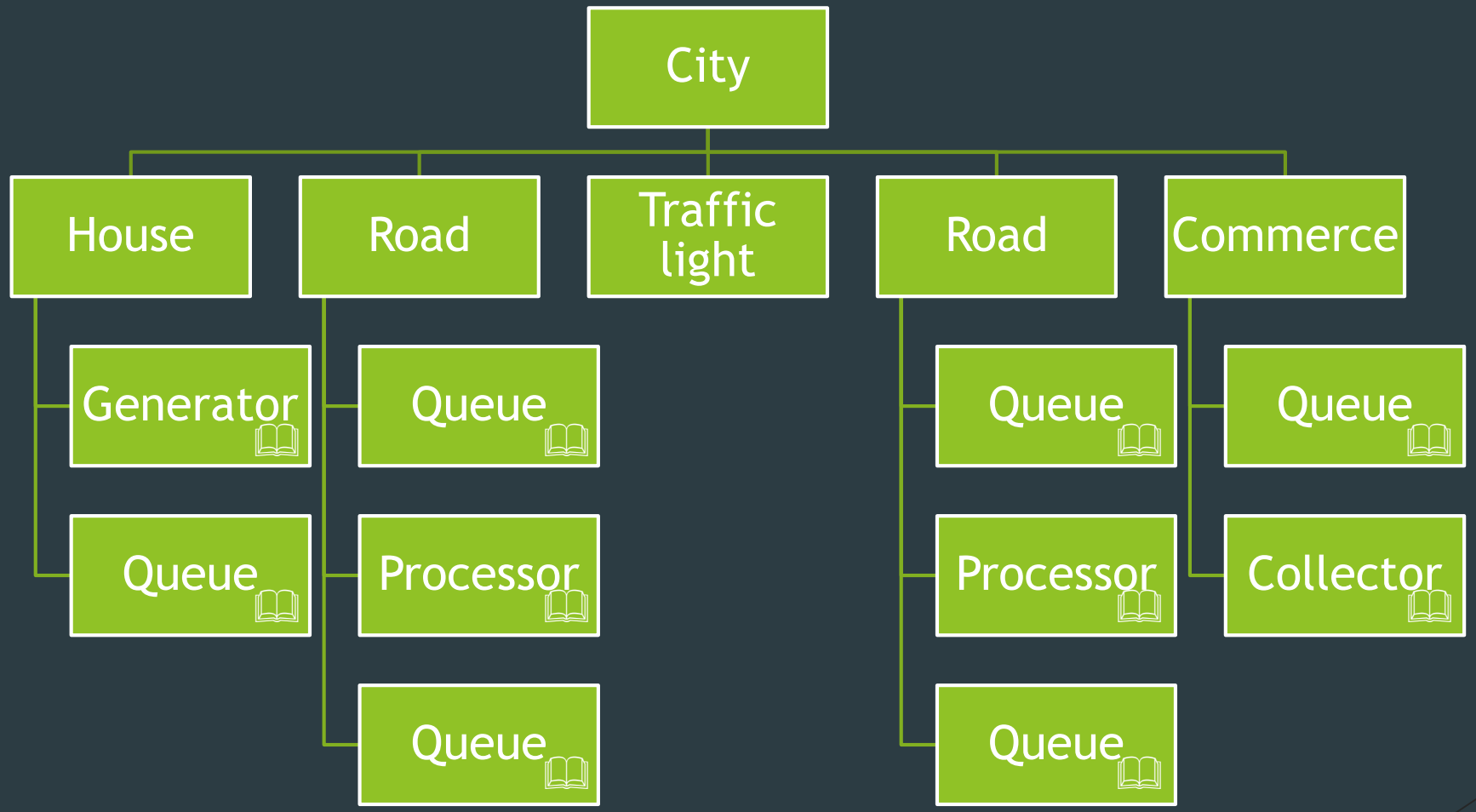

TRAFFIC

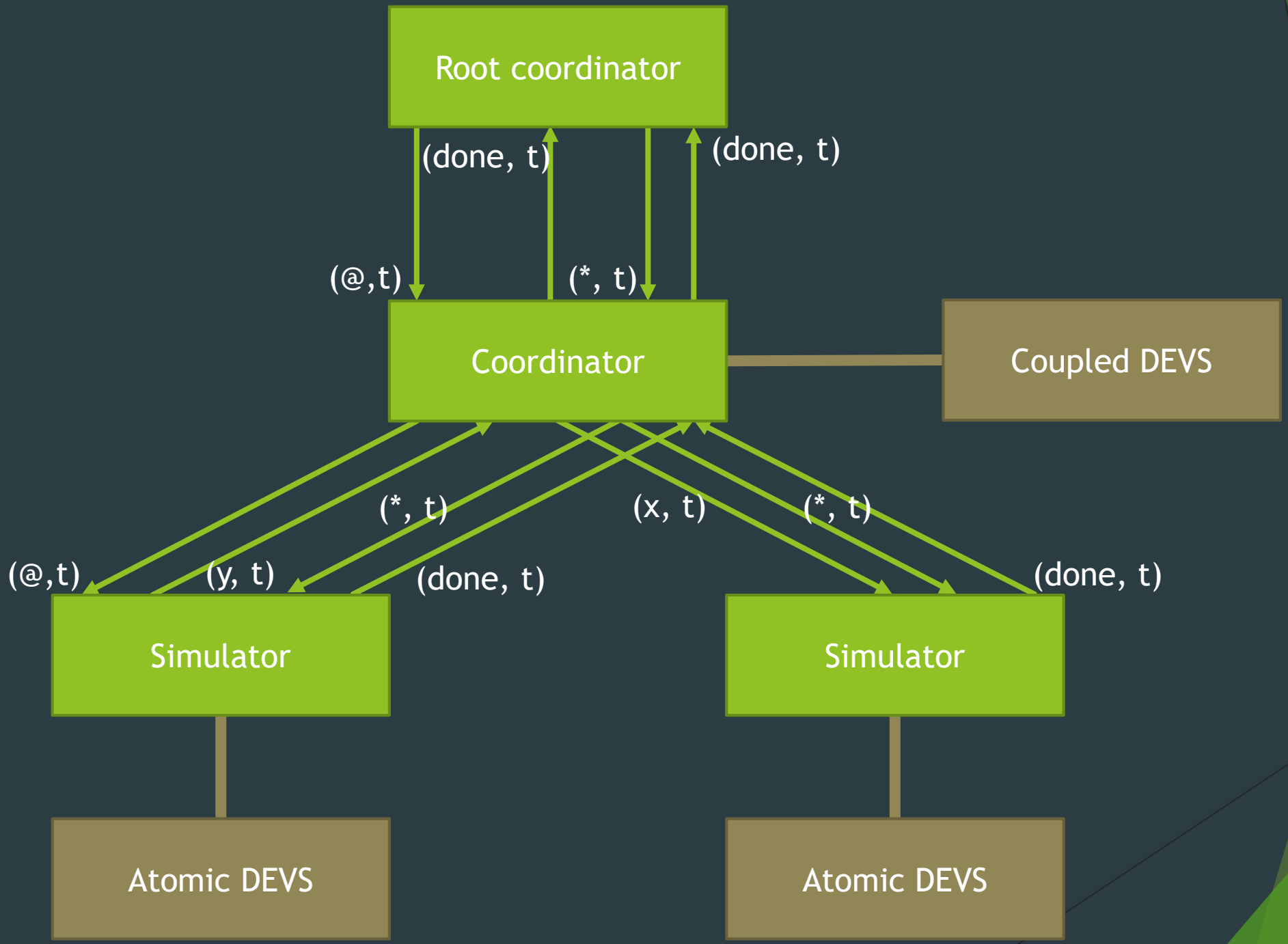


$$M = \langle X, Y, D, \{M_i\}, \{I_i\}, \{Z_{i,j}\} \rangle$$

trafficlight_system.py

```
from pypdevs.DEVS import *

from trafficlight import TrafficLight
from policeman import Policeman

class TrafficLightSystem(CoupledDEVS):
    def __init__(self):
        CoupledDEVS.__init__(self, “system”)
        self.light = self.addSubModel(TrafficLight())
        self.police = self.addSubModel(Policeman())
        self.connectPorts(self.police.out, self.light.interrupt)
```

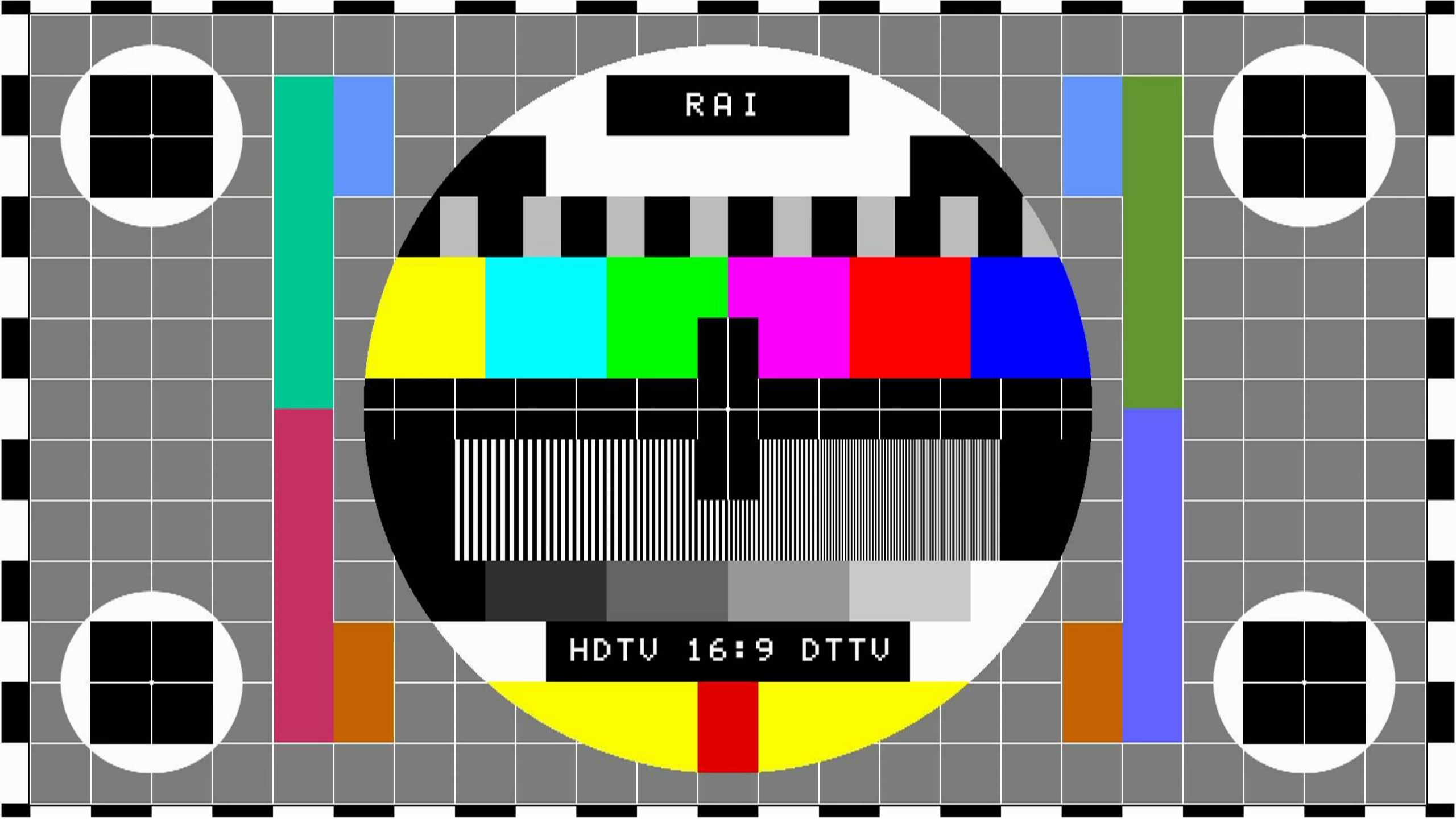

City
House
Road
Traffic light
Road
Commerce
Generator
Queue
Queue
Processor
Queue
Queue
Processor
Queue
Queue
Collector


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

Applications


GEORGE BRAZIL
Plumbing · Heating
Air Conditioning
SERVING YOUR COMMUNITY
24 HRS. A DAY

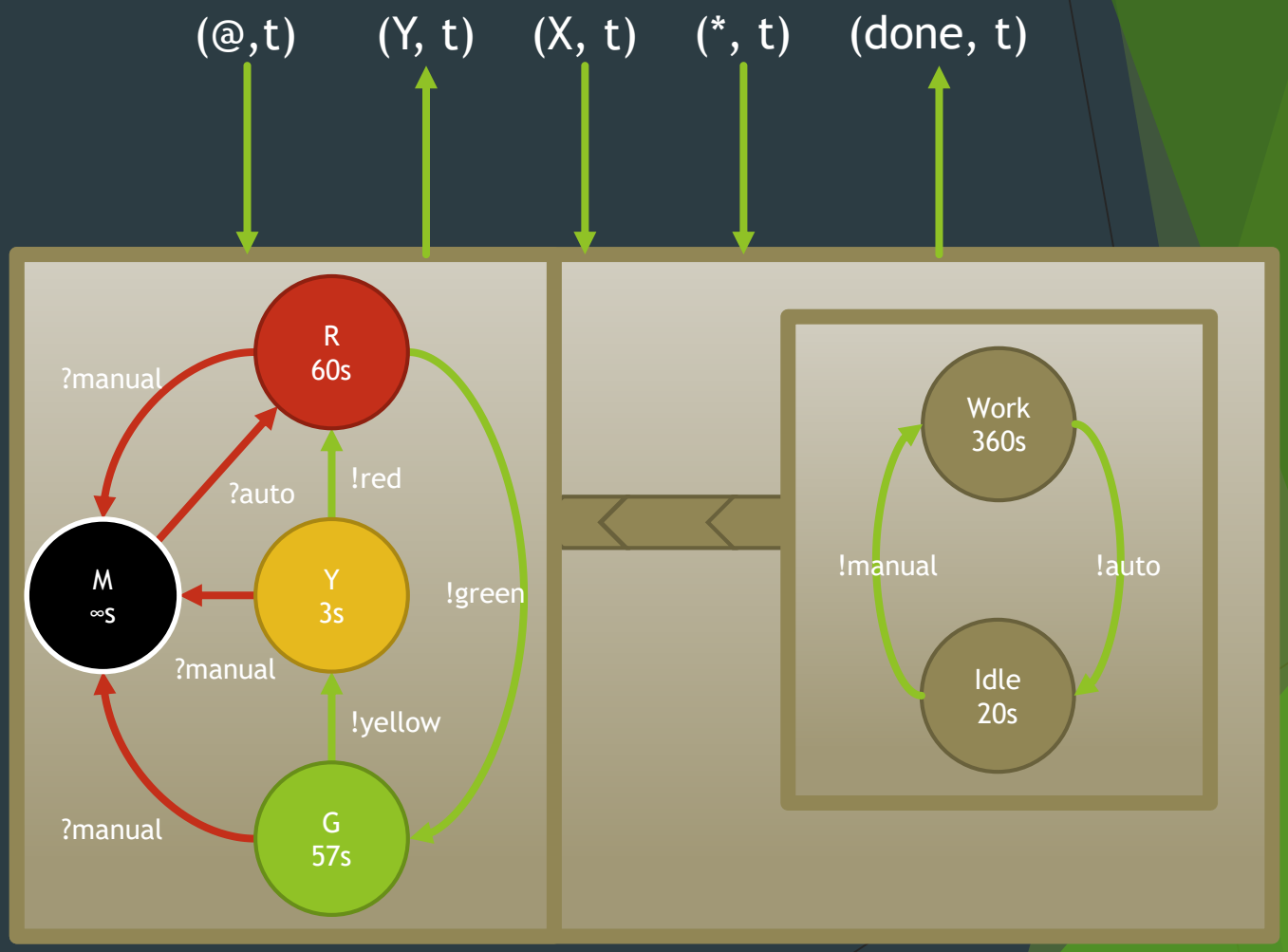

what if?


RAI
HDTV 16:9 DTTV

Conclusions

Conclusions

- Atomic DEVS
- Coupled DEVS
- Closure under coupling
- Abstract Simulator



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.

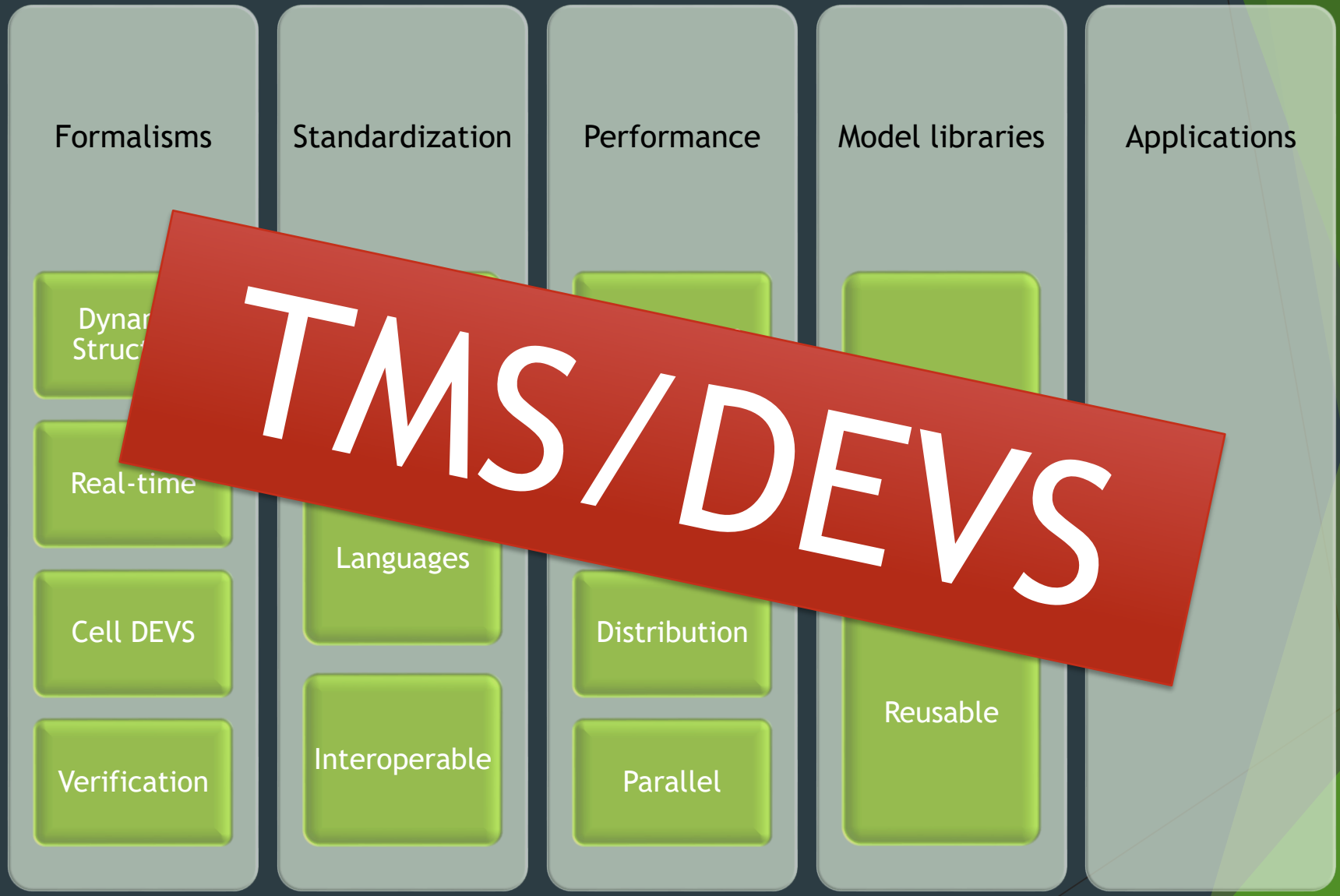
Generator

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


Formalisms
Standardization
Performance
Model libraries
Applications
Dynar
Struct
Real-time
Cell DEVS
Verification
Languages
Interoperable
Distribution
Parallel
Reusable
TMS/DEVS