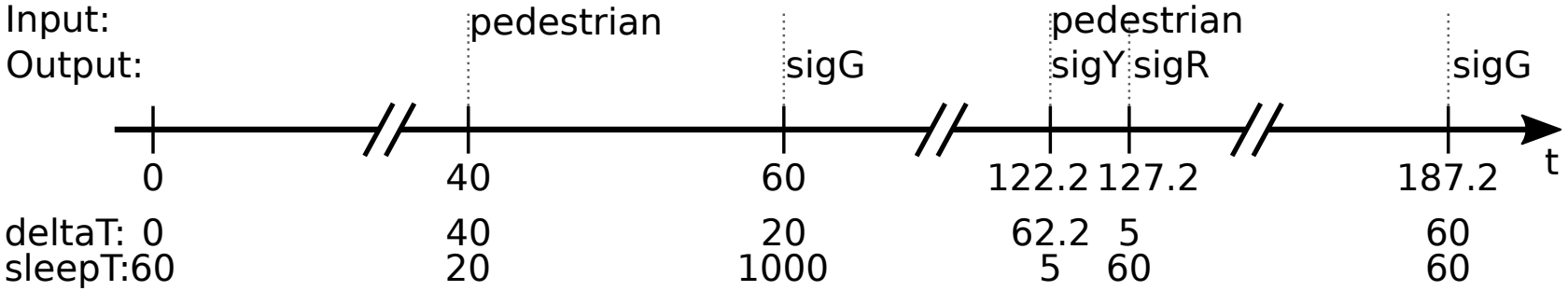


Alexander Schulz Rosengarten, *Reinhard von Hanxleden*
Kiel University

Fréderic Mallet, Robert de Simone, Julien DeAntoni
INRIA Sophia Antipolis

FDL'18 / SYNCHRON'18

Time in SCCharts

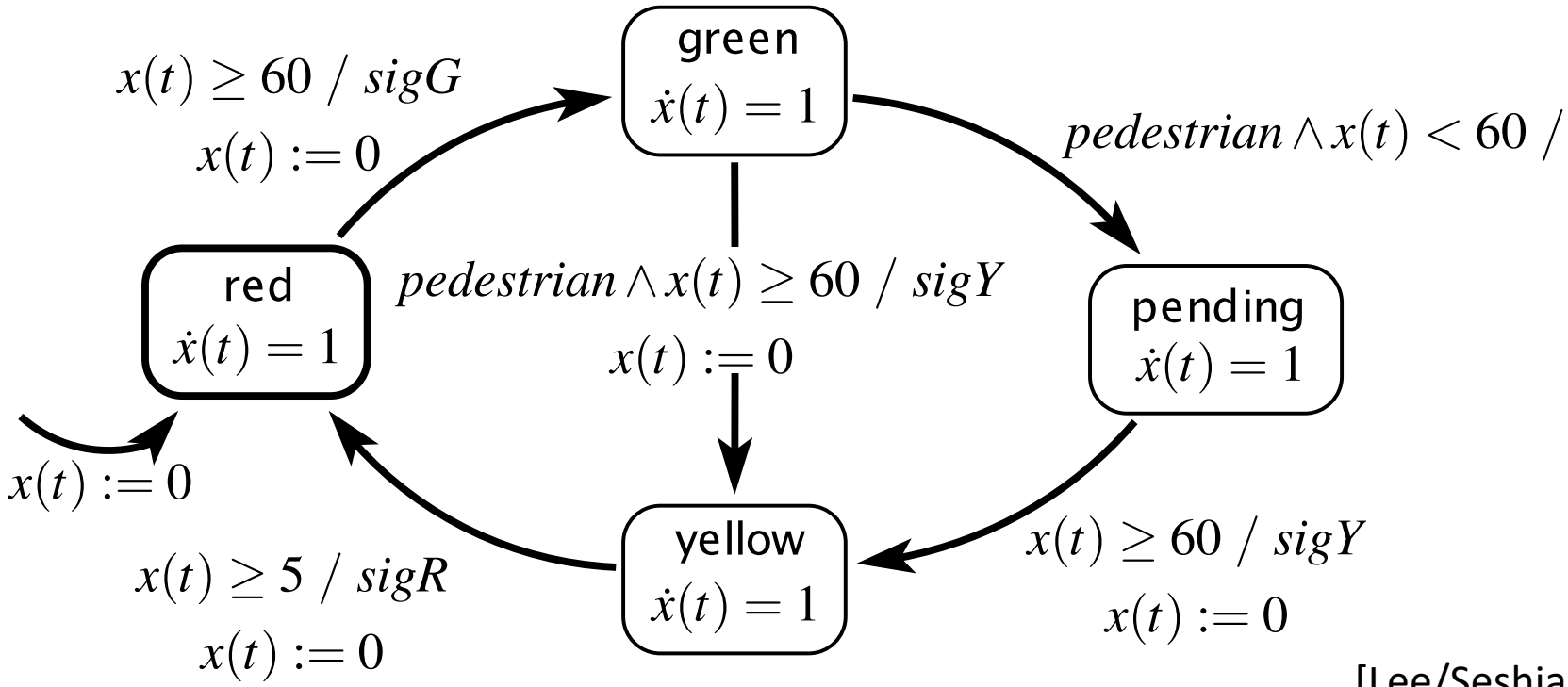
Alexander Schulz Rosengarten, *Reinhard von Hanxleden*
Kiel University

Fréderic Mallet, Robert de Simone, Julien DeAntoni
INRIA Sophia Antipolis

FDL'18 / SYNCHRON'18

Traffic Light as Timed Automaton

continuous variable: $x(t) : \mathbb{R}$
inputs: *pedestrian* : pure
outputs: *sigR*, *sigG*, *sigY* : pure

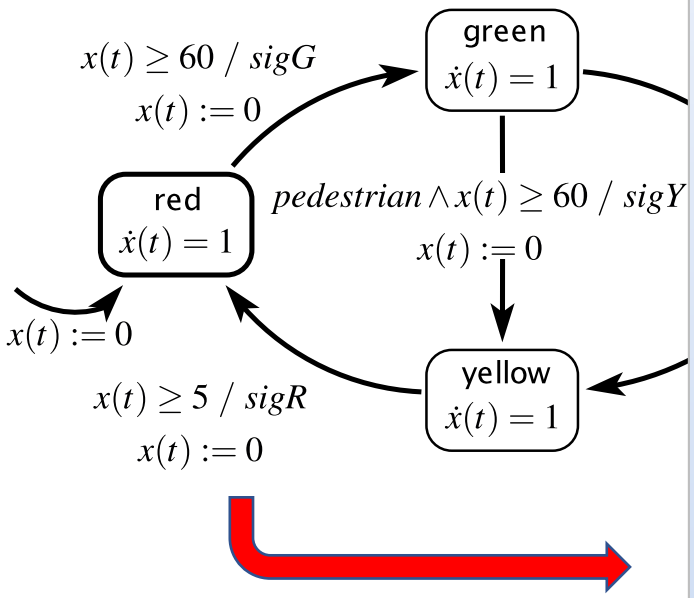


[Lee/Seshia]

Alur, Dill, *A theory of timed automata*, Theoretical Computer Science, 1994

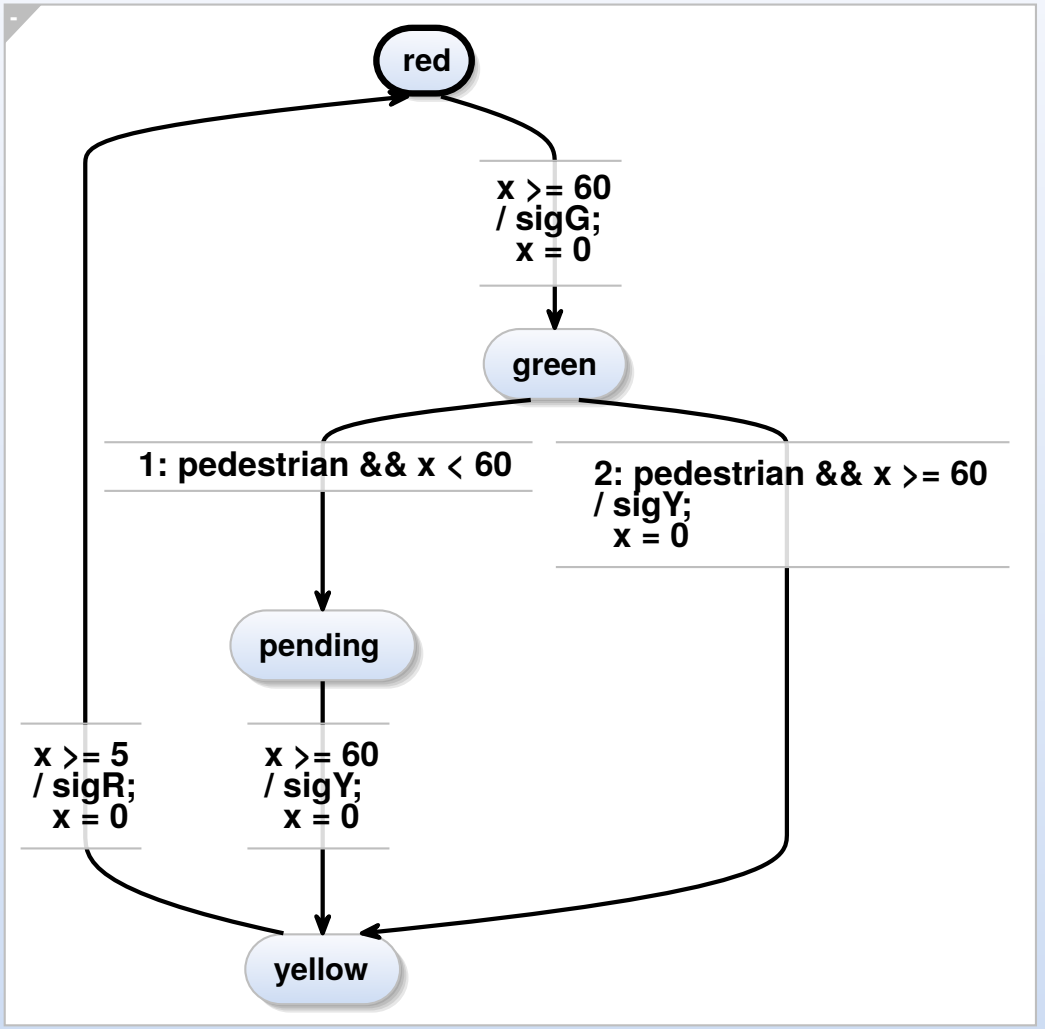
Traffic Light in SCCharts

continuous variable: $x(t) : \mathbb{R}$
inputs: *pedestrian* : pure
outputs: *sigR*, *sigG*, *sigY* : pure



TimedTrafficLight

input signal pedestrian
output signal sigR, sigG, sigY
clock x = 0



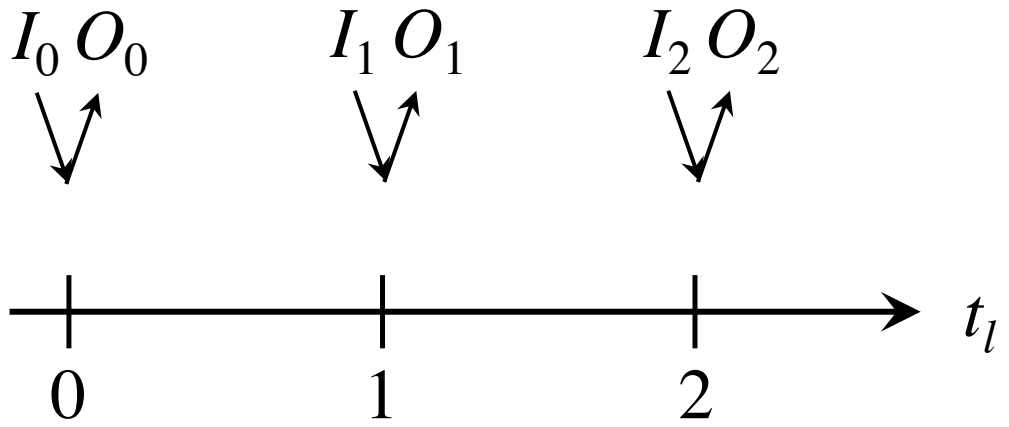
Roadmap

1. Traffic Light Example
2. Execution Models
3. Dynamic Ticks
4. Time in SCCharts: “clock”
5. Multiclocks in SCCharts: “period”
6. Demo

Roadmap

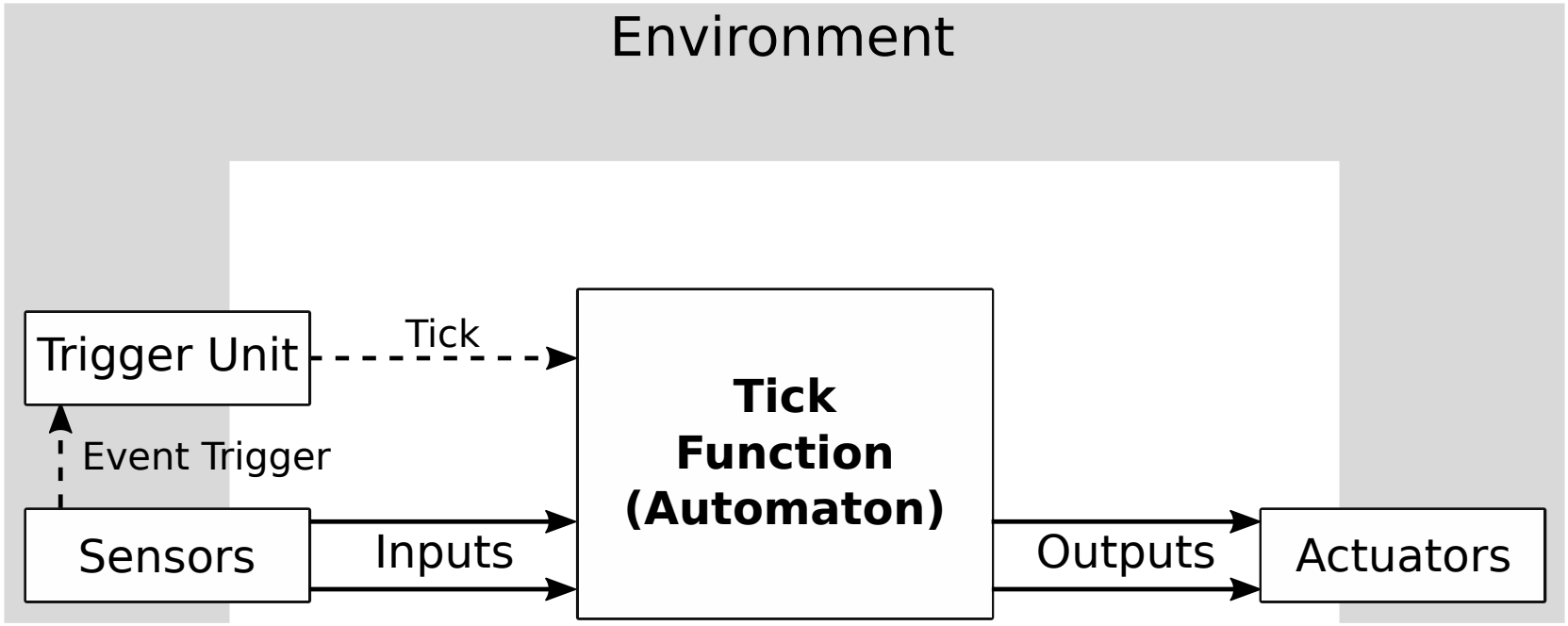
1. Traffic Light Example
2. Execution Models
3. Dynamic Ticks
4. Time in SCCharts: “clock”
5. Multiclocks in SCCharts: “period”
6. Demo

Discrete (Logical) Time in Synchronous Programming



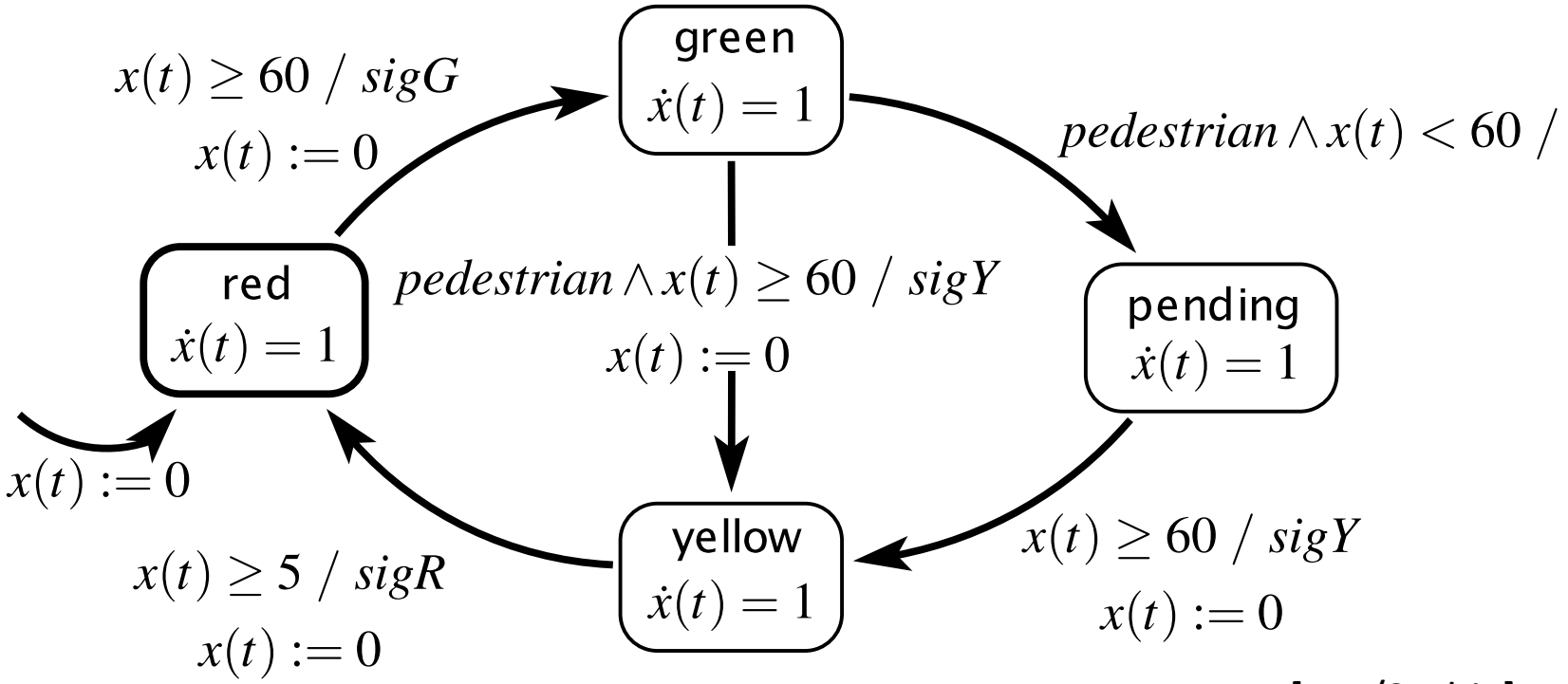
- Synchrony Hypothesis:
  Outputs are synchronous with inputs
- Computation "does not take time"
- Actual computation time does not influence result
- Sequence of outputs **determined** by inputs

Event-Triggered Execution



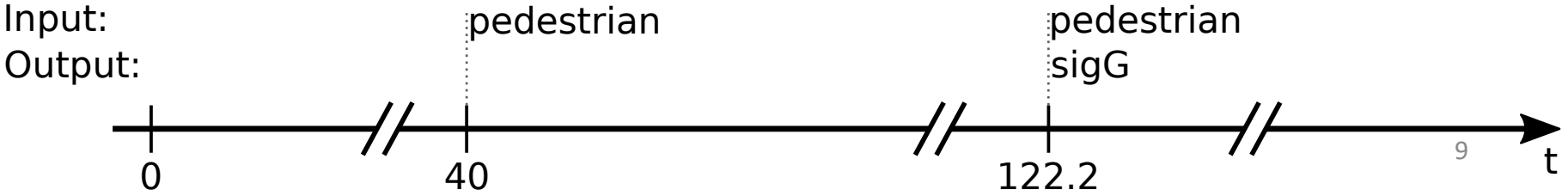
continuous variable: $x(t) : \mathbb{R}$
inputs: *pedestrian* : pure
outputs: *sigR*, *sigG*, *sigY* : pure

Assume pedestrian button pressed at t = 40 and t = 122.2



[Lee/Seshia]

Event-Triggered Execution, with initial tick at t = 0:



Synchronous Execution

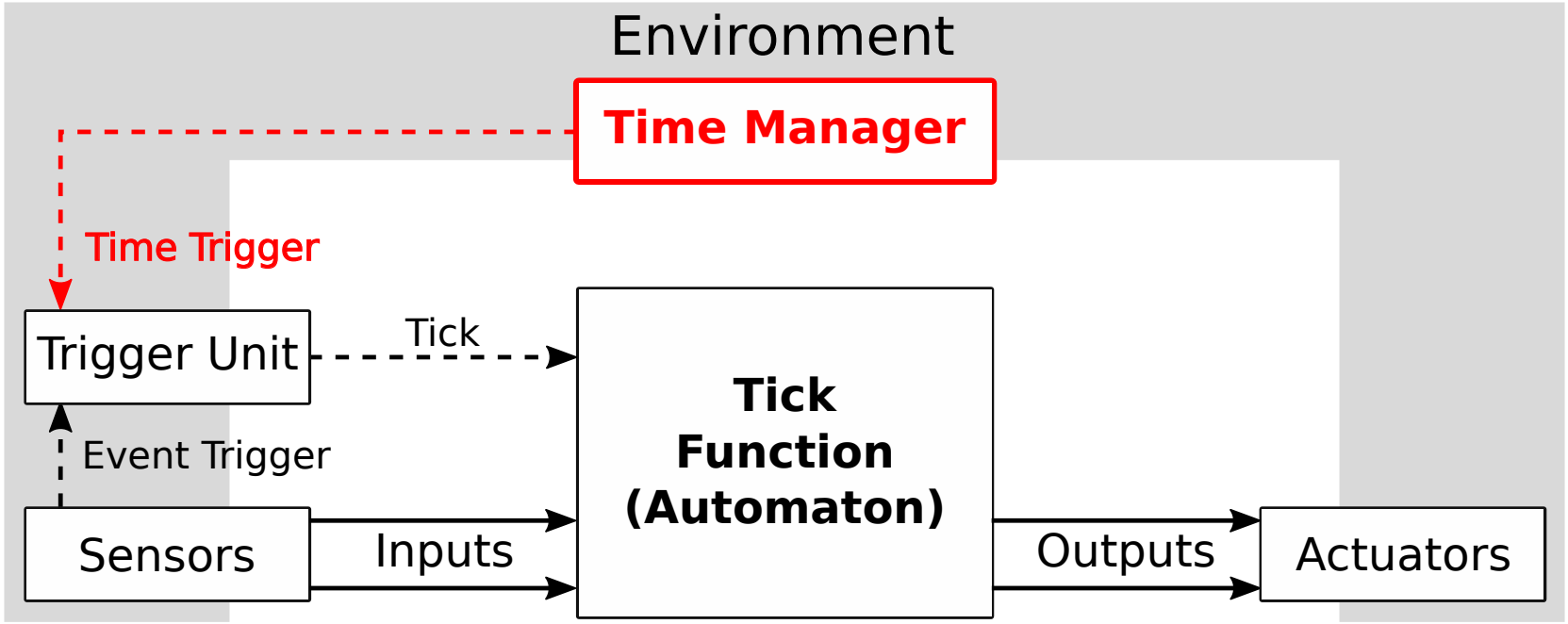
```
Initialize Memory
for each input event do
    Compute Outputs
    Update Memory
end
```

```
Initialize Memory
for each clock tick do
    Read Inputs
    Compute Outputs
    Update Memory
end
```

Fig. 1 Two common synchronous execution schemes: event driven (left) and sample driven (right).

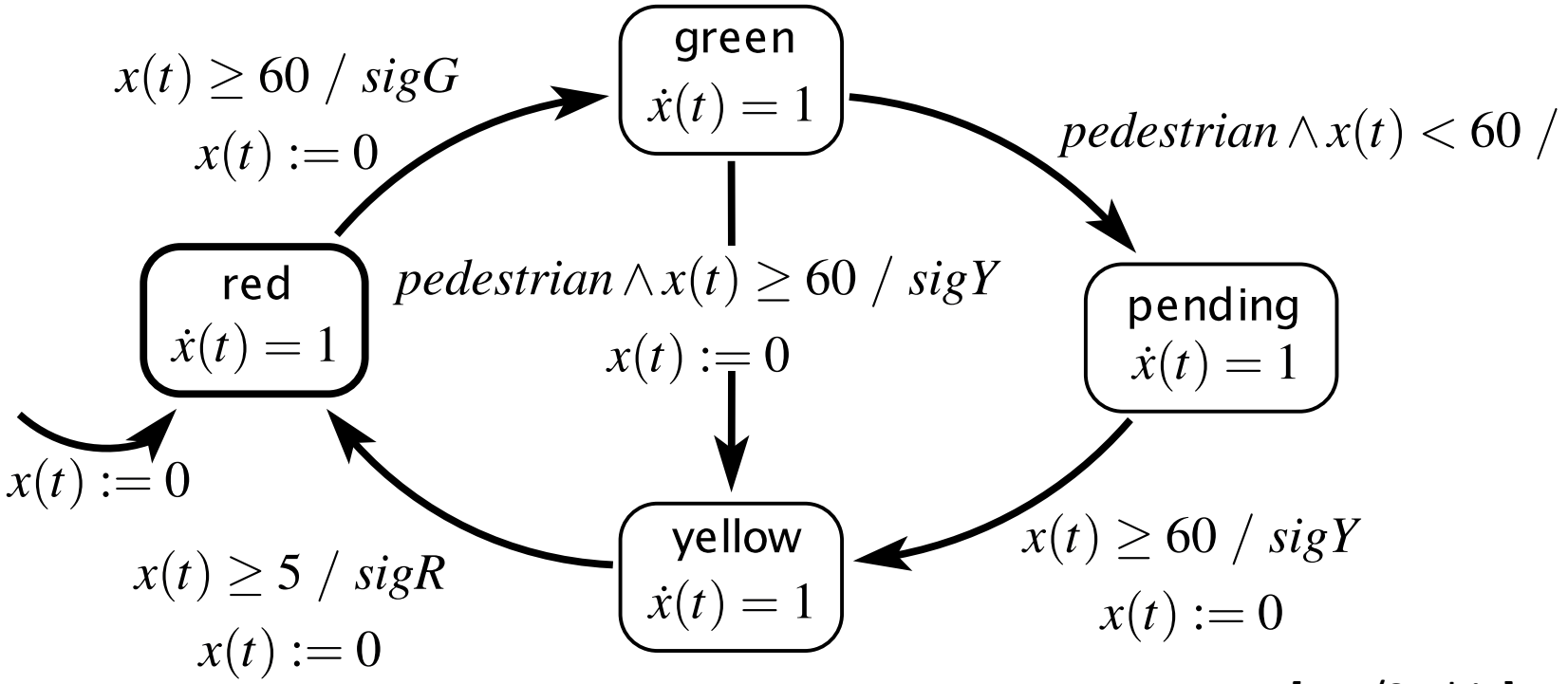
[Benveniste et al., *The Synchronous Languages Twelve Years Later*, Proc. IEEE, 2003]

Time-Triggered Execution



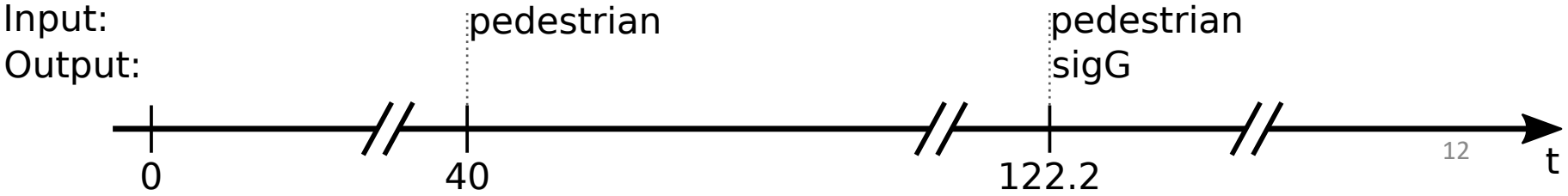
continuous variable: $x(t) : \mathbb{R}$
inputs: *pedestrian* : pure
outputs: *sigR*, *sigG*, *sigY* : pure

Assume pedestrian button pressed at t = 40 and t = 122.2



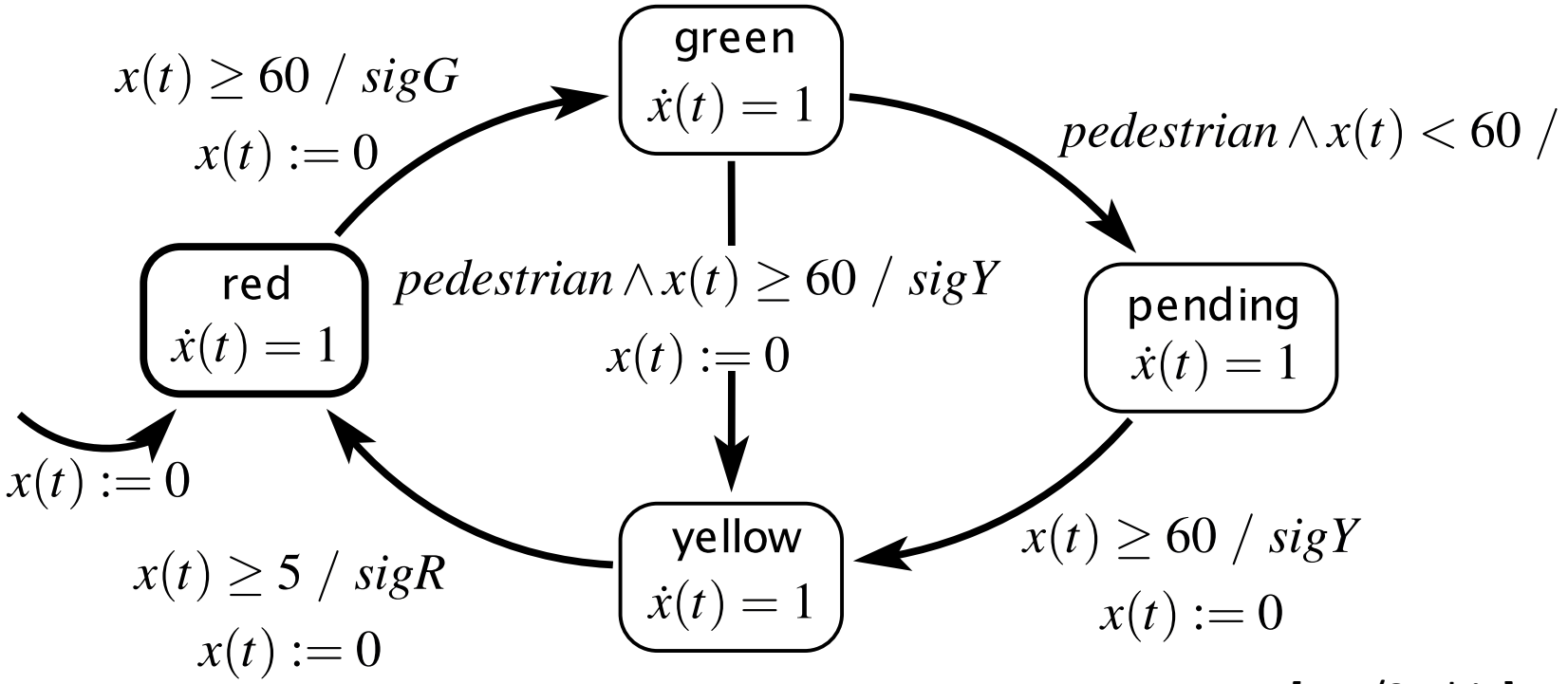
[Lee/Seshia]

Recall: Event-Triggered Execution:



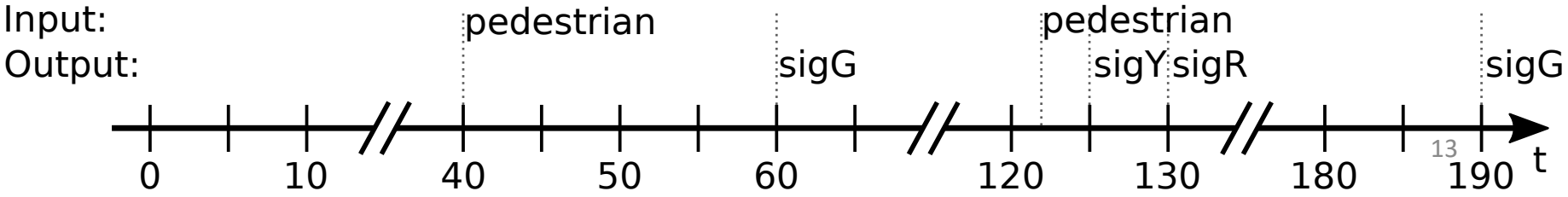
continuous variable: $x(t) : \mathbb{R}$
inputs: *pedestrian* : pure
outputs: *sigR*, *sigG*, *sigY* : pure

Assume pedestrian button pressed at t = 40 and t = 122.2



[Lee/Seshia]

Time-Triggered Execution (every 5 sec):



Multiform Notion of Time

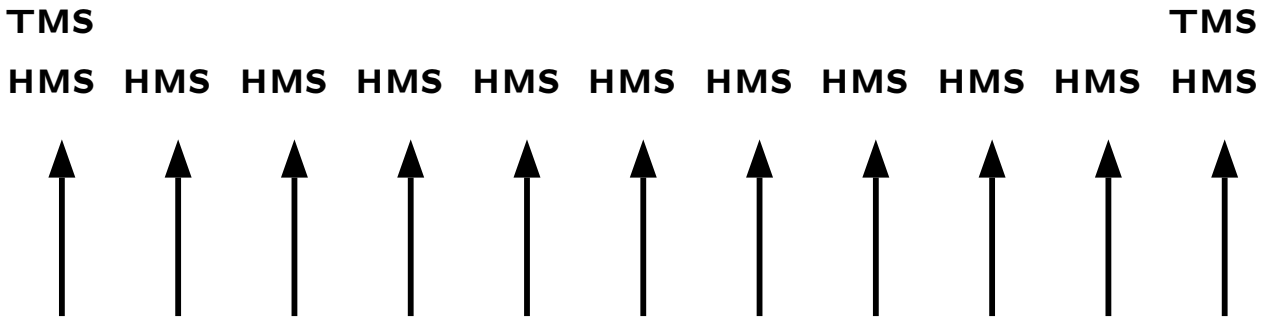
Only the simultaneity and precedence of events are considered.

This means that the physical time does not play any special role.

This is called multiform notion of time.

[https://en.wikipedia.org/wiki/Esterel]

Packaging Physical Time as Events



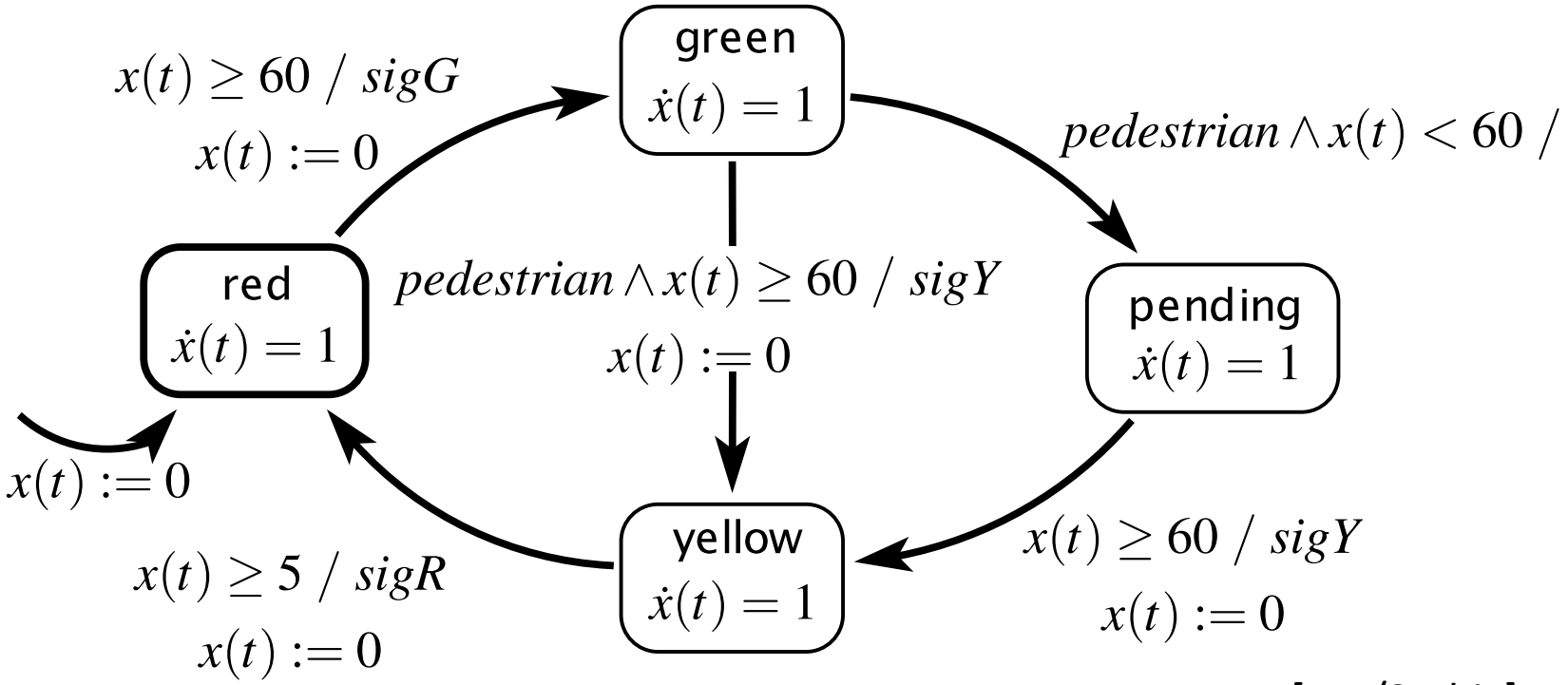
[Timothy Bourke, SYNCHRON 2009]

Event "HMS": 100 µsec have passed since last HMS

Event "TMS": 1000 µsec have passed since last TMS

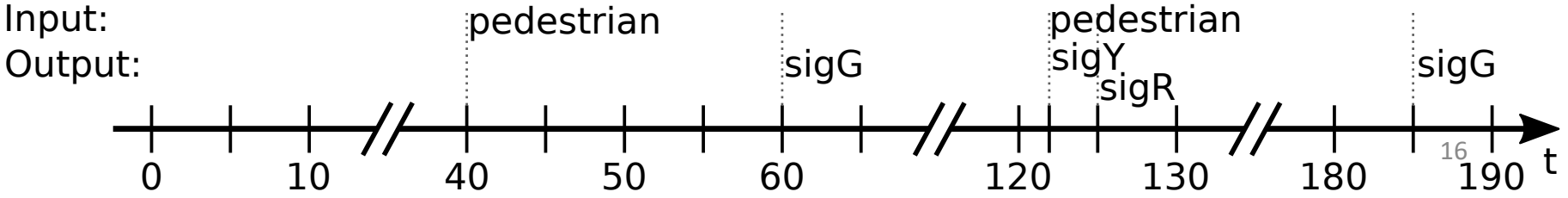
continuous variable: $x(t) : \mathbb{R}$
inputs: *pedestrian* : pure
outputs: *sigR*, *sigG*, *sigY* : pure

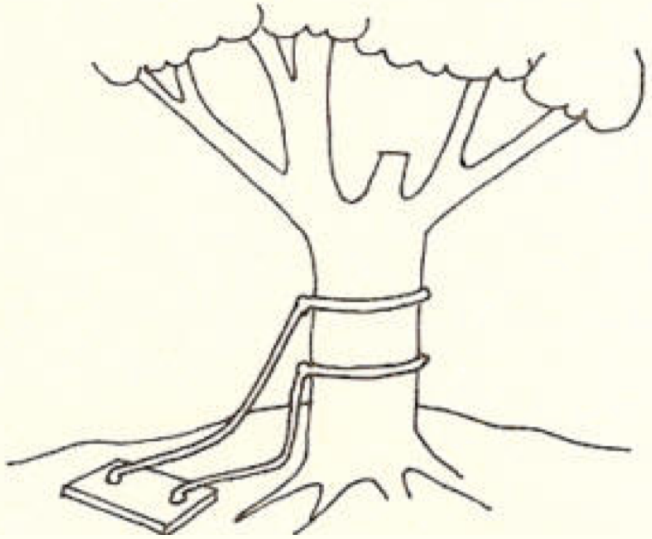
Assume pedestrian button pressed at t = 40 and t = 122.2



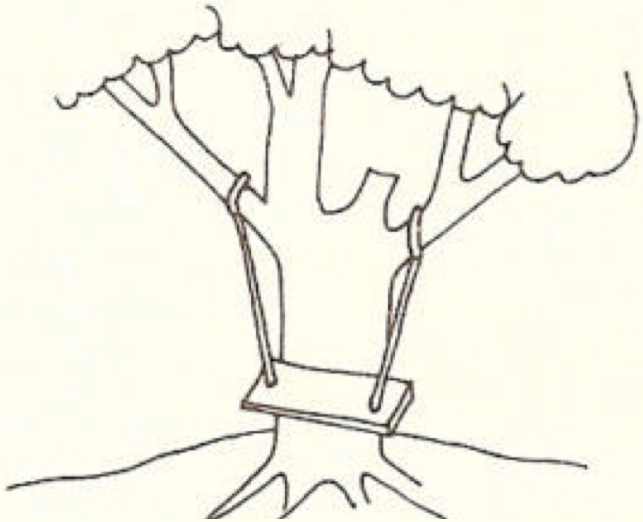
[Lee/Seshia]

Time-Event-Triggered Execution, Multiform Time:

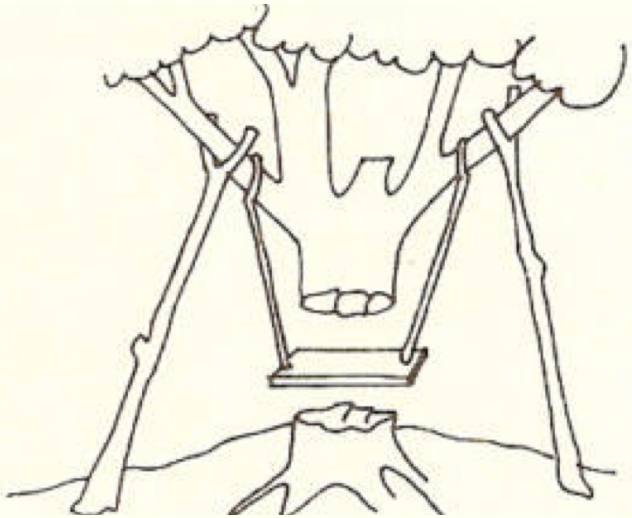




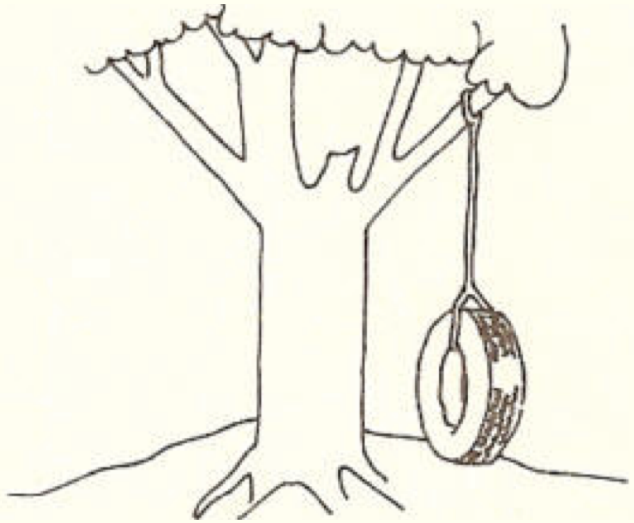
Event-Triggered



Time-Triggered



Time-Event-Triggered



Eager

ACKNOWLEDGEMENTS TO UNKNOWN AUTHOR

From the *University of London Computer Centre Newsletter* No. 53, March 1973

What the User (Probably) Wanted

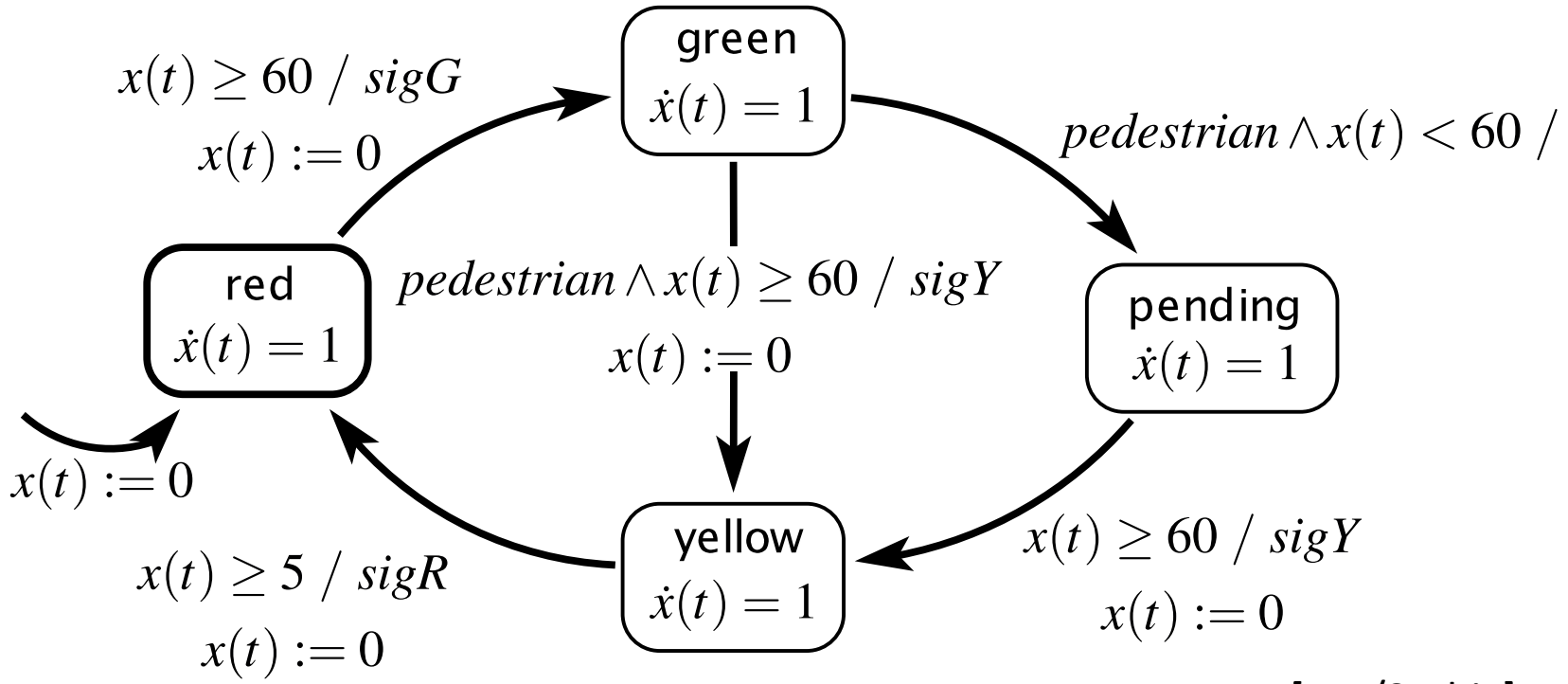
„We assume here that a transition is taken as soon as it is enabled. Other transition semantics are possible.“

[Lee/Seshia 2017]

We call this **eager** semantics.

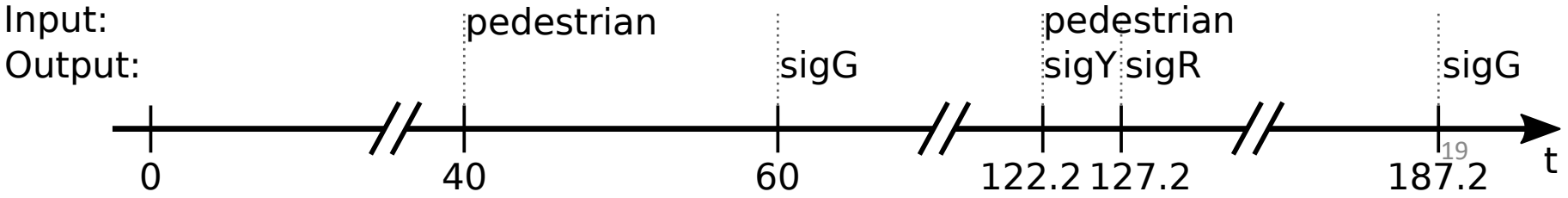
continuous variable: $x(t) : \mathbb{R}$
inputs: *pedestrian* : pure
outputs: *sigR*, *sigG*, *sigY* : pure

Assume pedestrian button pressed at t = 40 and t = 122.2



[Lee/Seshia]

Eager Semantics:



Time in SCCharts – Requirements

1. Seamless fit into synchronous paradigm
   - Still deterministic behavior – outputs fully determined by inputs
   - No changes to underlying SC (Sequentially Constructive) MoC
2. Approximate eager semantics
   - Modulo run-time variations and imperfections of physical timers
3. Scalability
   - E.g., allow arbitrary number of (concurrent) timers
4. Fine granularity
   - Gcd may be arbitrarily small, w/o performance penalty
   - E.g., may have timeouts of 1 sec and 3.1415926 msec in same model
5. Time composability
   - E.g., waiting 1 sec. twice should mean the same as waiting 2 sec's once

Time in SCCharts – Requirements

6. Preserve temporal order and simultaneity
   - E.g., timers started in same tick and running same duration should expire in same tick
7. Minimize impact of physical timer variations
   - E.g., avoid accumulations of timer imperfections
8. Give application access to physical time and tick computation time
   - Facilitates e.g. load-dependent execution modes
9. Lean, application-independent interface to environment
   - E.g., interface should not change if number of timers changes
10. Fit into Single Language-Driven Incremental Compilation (SLIC) concept
    - New timing constructs are just syntactic sugar on top of existing SCCharts
    - Transforming away timing constructs requires only local changes
    - No changes needed to compilation back-end

Roadmap

1. Traffic Light Example
2. Execution Models
3. Dynamic Ticks
4. Time in SCCharts: “clock”
5. Multiclocks in SCCharts: “period”
6. Demo

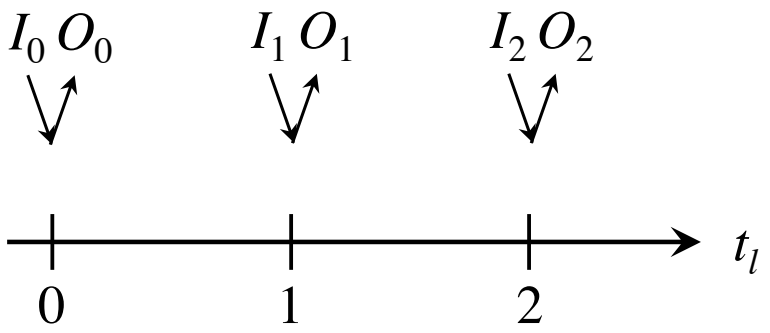
Real-Time Ticks for Synchronous Programming

Reinhard von Hanxleden (U Kiel)
Timothy Bourke (INRIA and ENS, Paris)
Alain Girault (INRIA and U Grenoble)

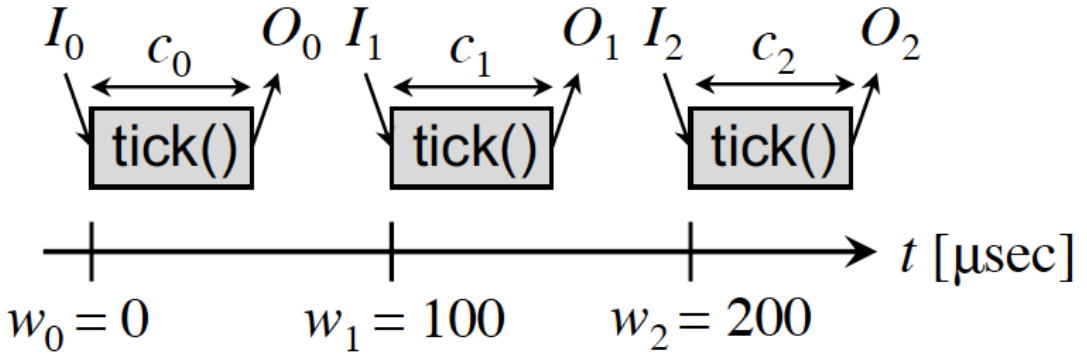
19 Sep 2017, FDL '17, Verona

Dynamic Ticks

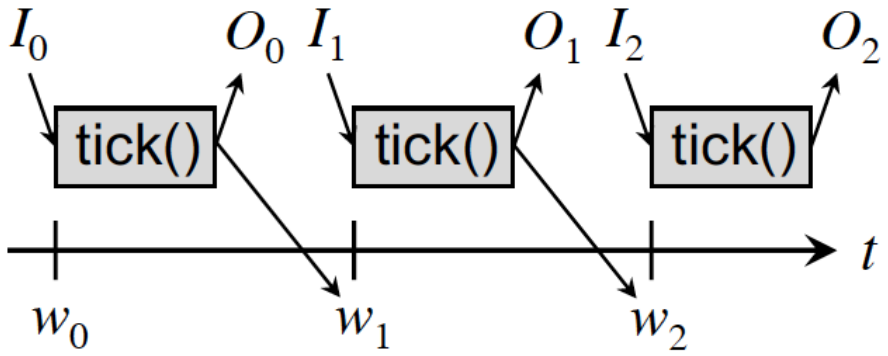
- Recall logical time:



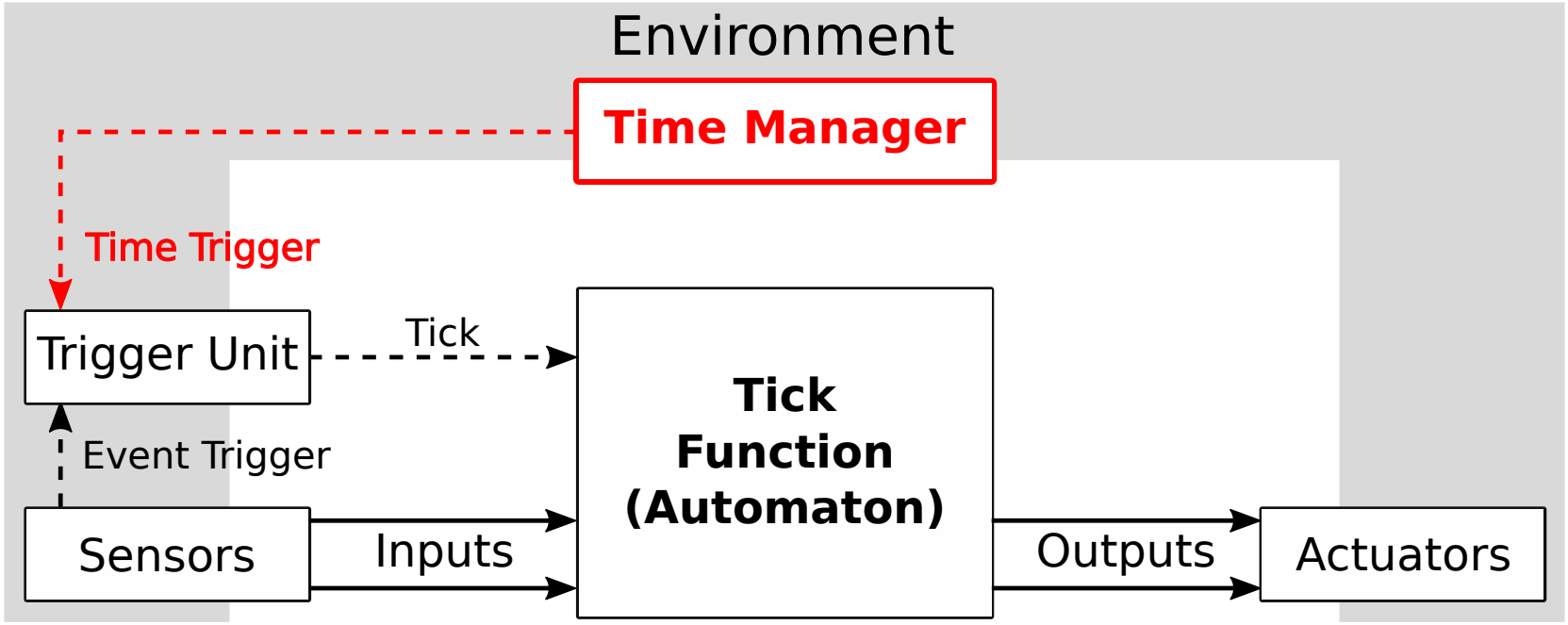
- Physical time, time-triggered:



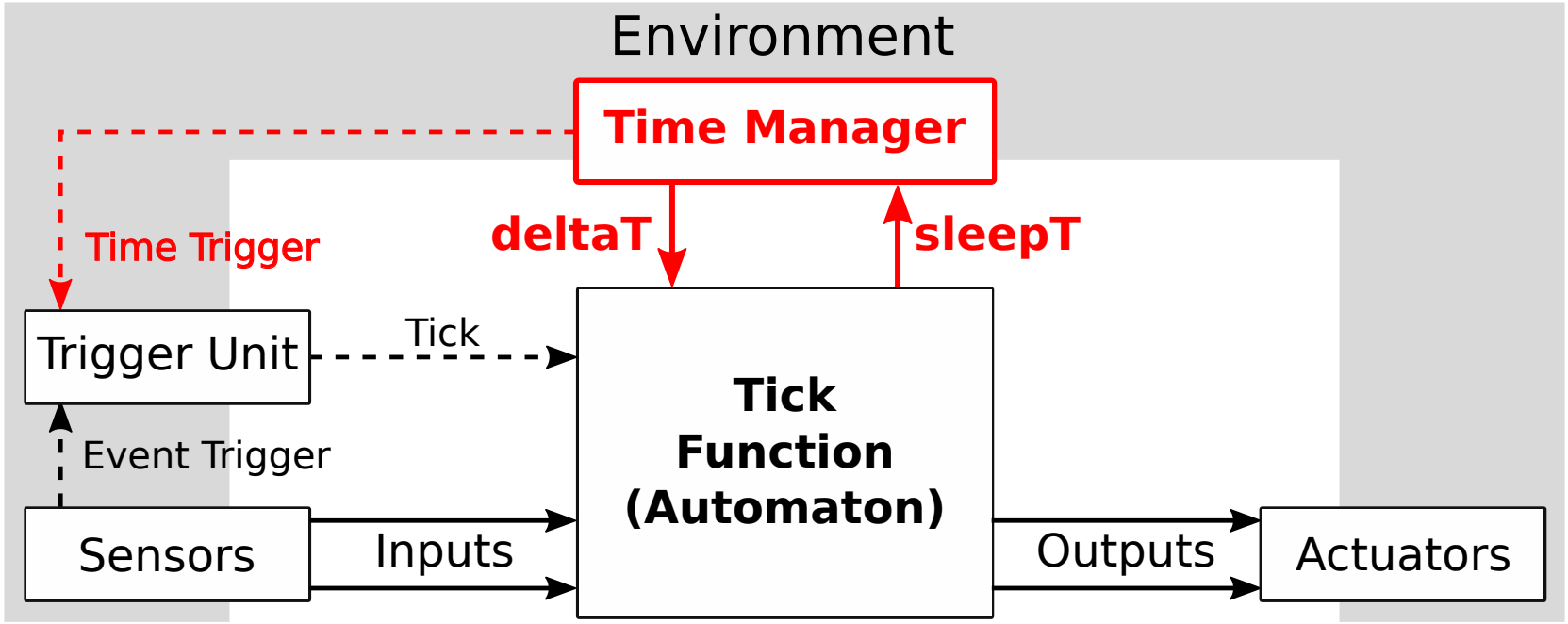
- Physical time, dynamic ticks:



Recall: Time-Triggered Execution



Eager Execution with Dynamic Ticks

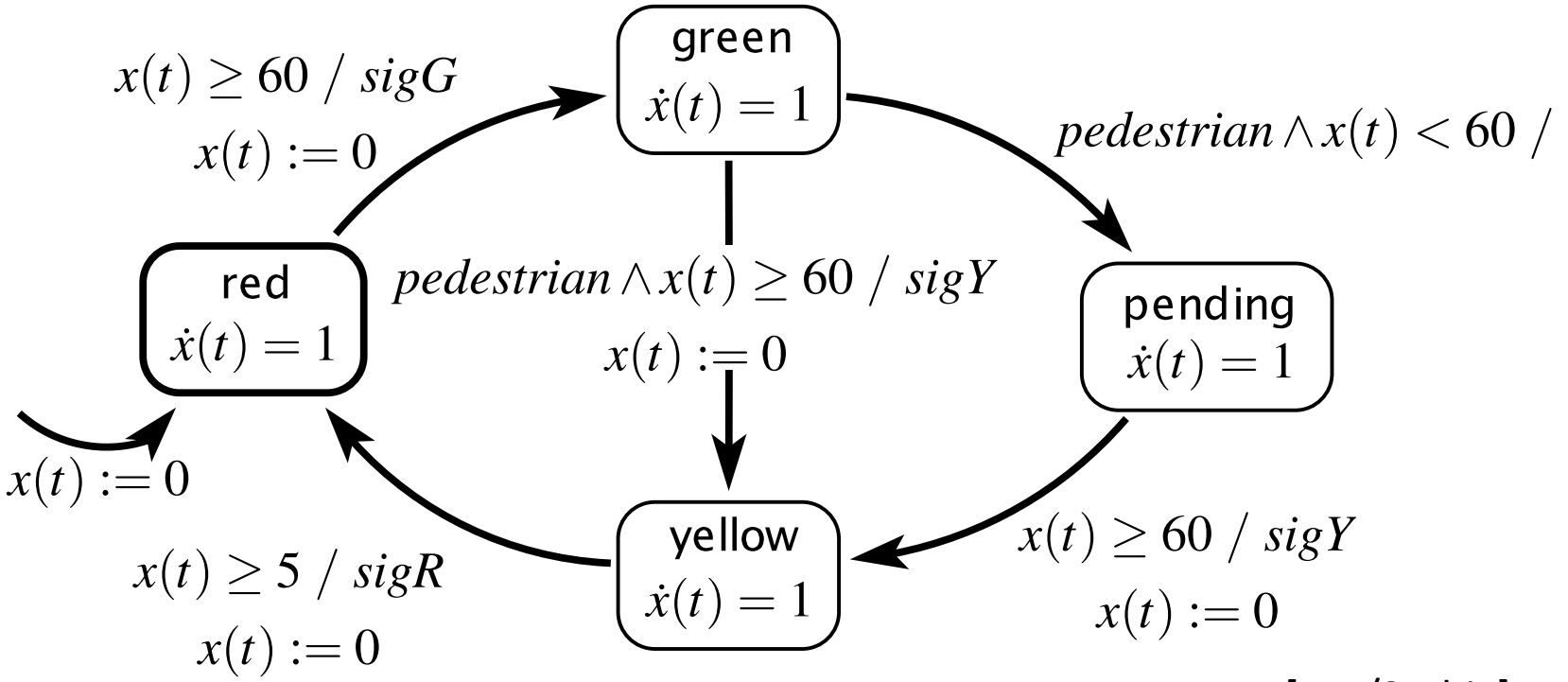


deltaT: Time since last tick

sleepT: Requested delay until next tick

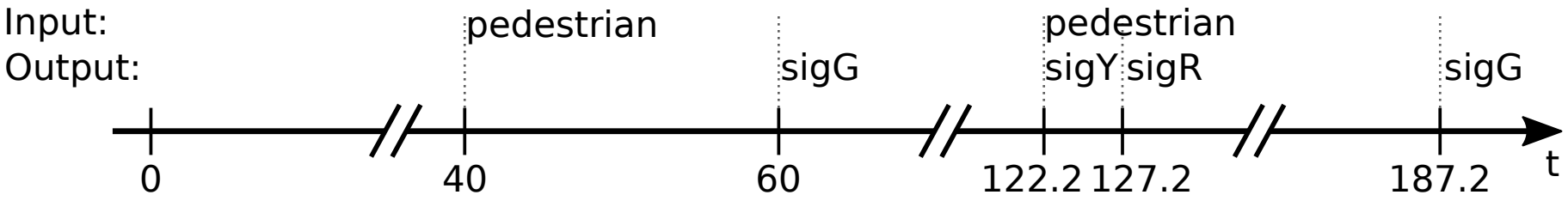
continuous variable: $x(t) : \mathbb{R}$
inputs: *pedestrian* : pure
outputs: *sigR*, *sigG*, *sigY* : pure

Assume pedestrian button pressed at t = 40 and t = 122.2



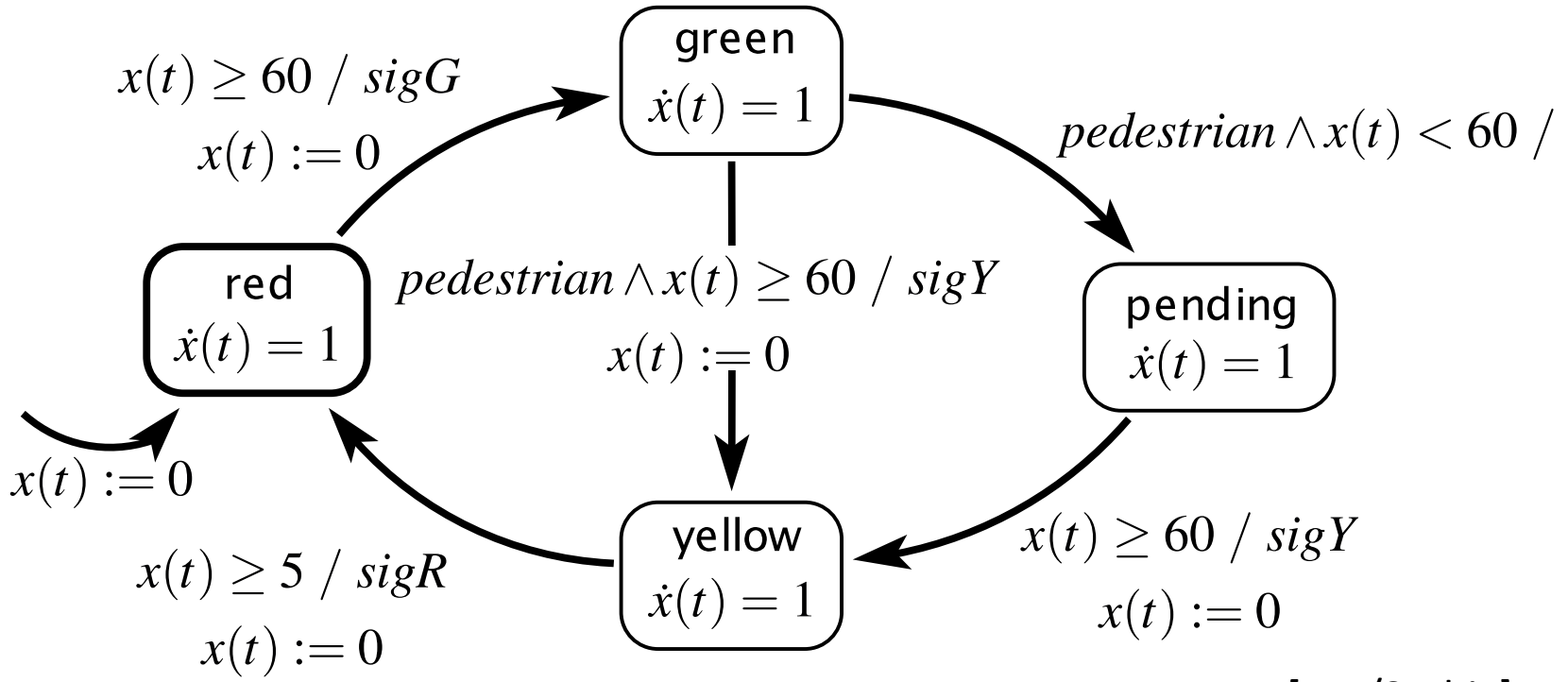
[Lee/Seshia]

Recall: Eager Semantics



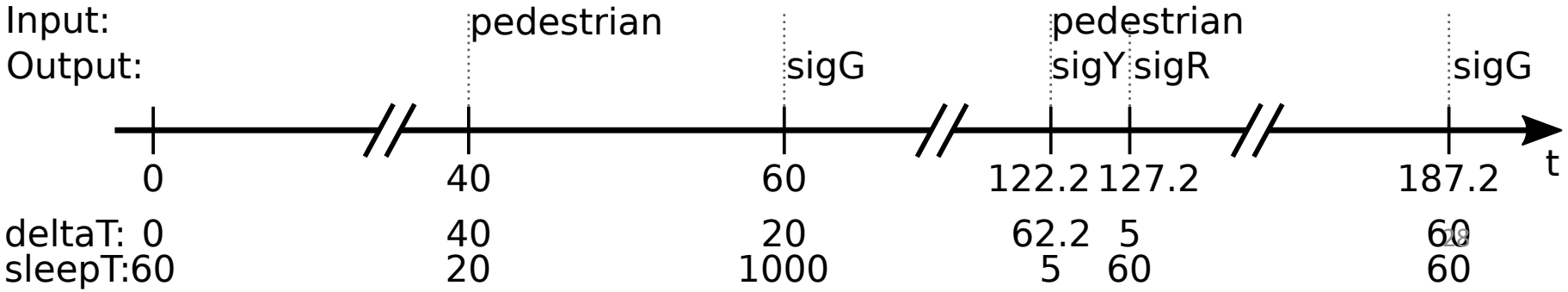
continuous variable: $x(t) : \mathbb{R}$
inputs: *pedestrian* : pure
outputs: *sigR*, *sigG*, *sigY* : pure

Assume pedestrian button pressed at t = 40 and t = 122.2



[Lee/Seshia]

Eager Execution with Dynamic Ticks:



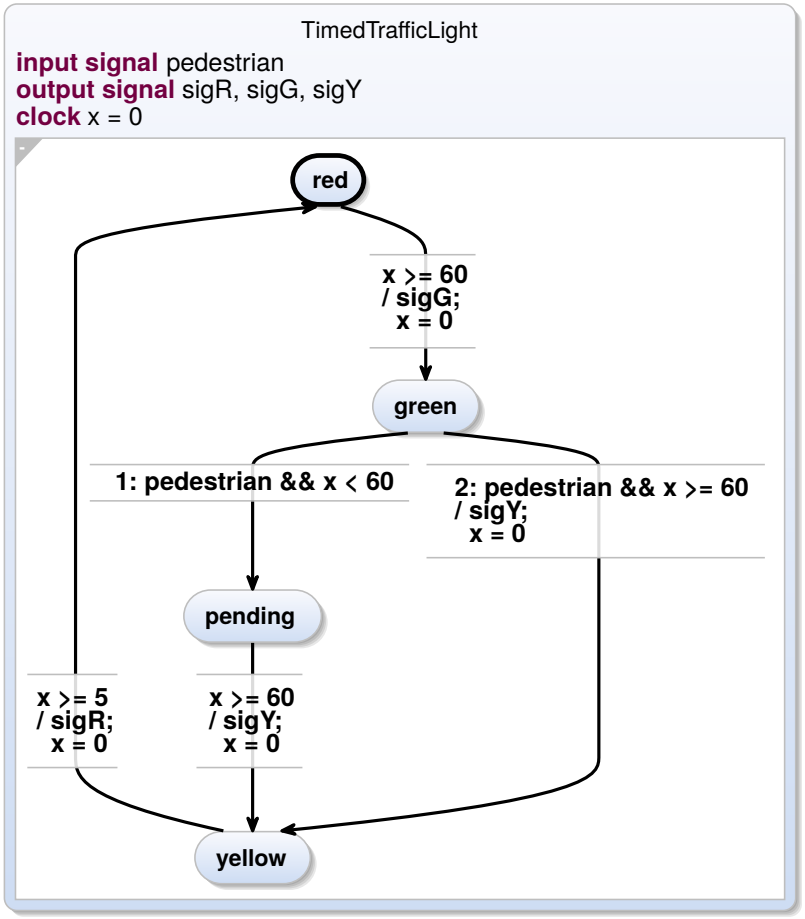
Multiform Notion of Time – Again!

- Semantically, treat clocks (time) as a unit-less number
- As in timed automata, clocks must satisfy *monotonicity* (modulo resets) and *progress*
- Current implementation maps time (clock variables) to an approximation of real numbers (float), interpreted as seconds
- However, could also map clocks to integers, interpreted as Euros spent, fathoms travelled, or beers consumed

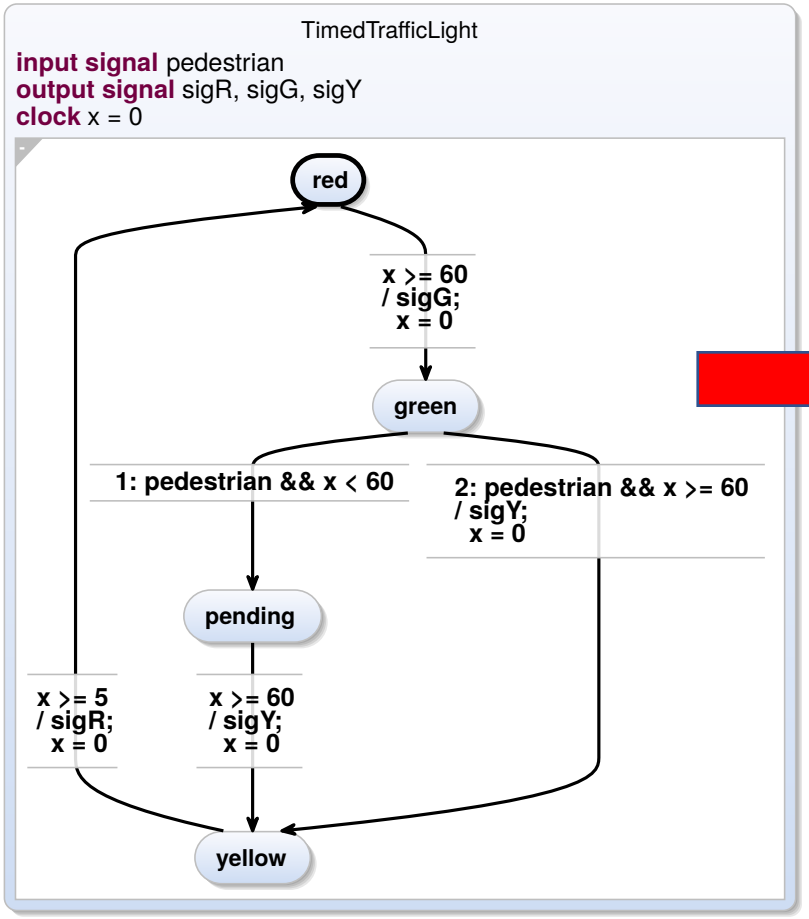
Roadmap

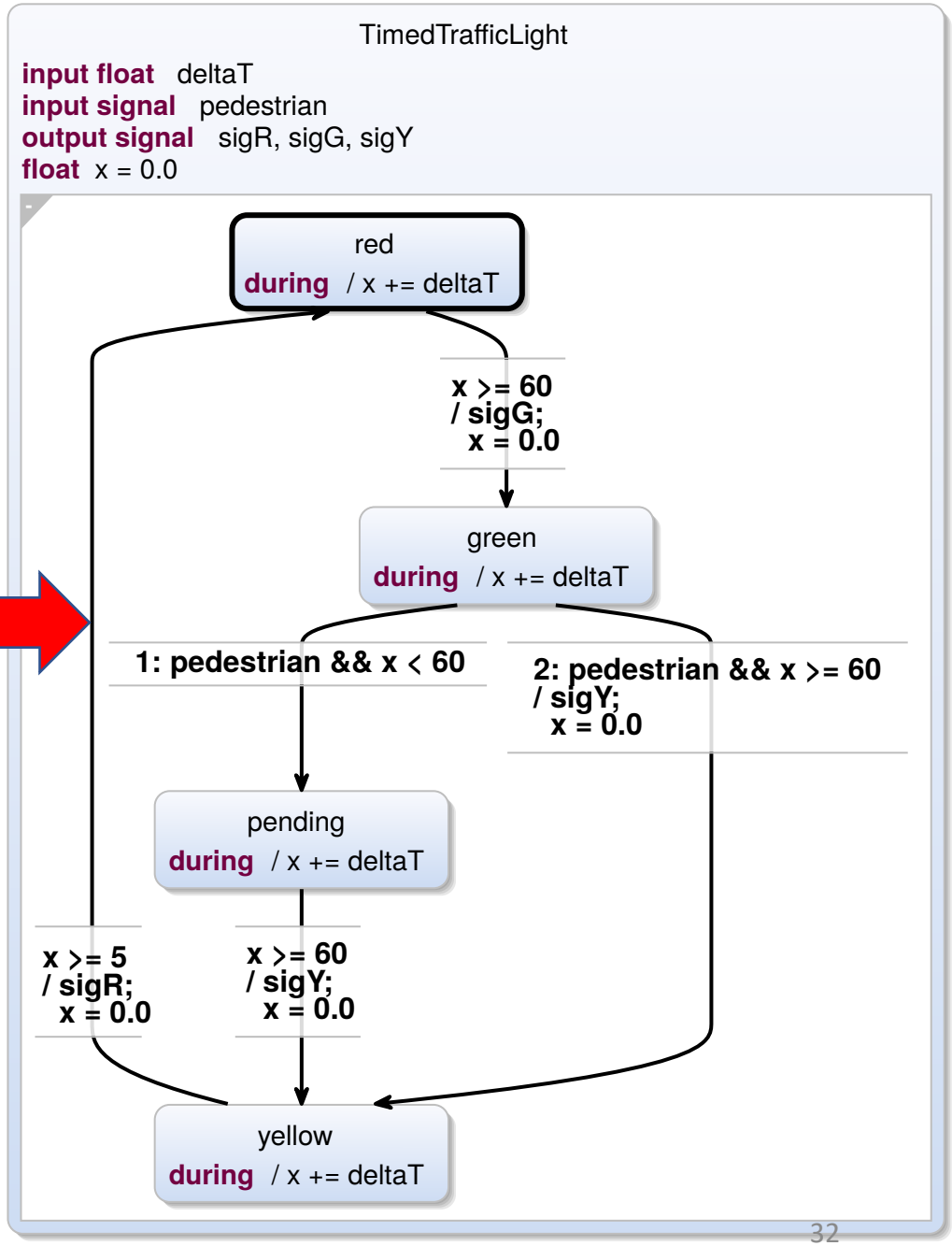
1. Traffic Light Example
2. Execution Models
3. Dynamic Ticks
4. Time in SCCharts: “clock”
5. Multiclocks in SCCharts: “period”
6. Demo

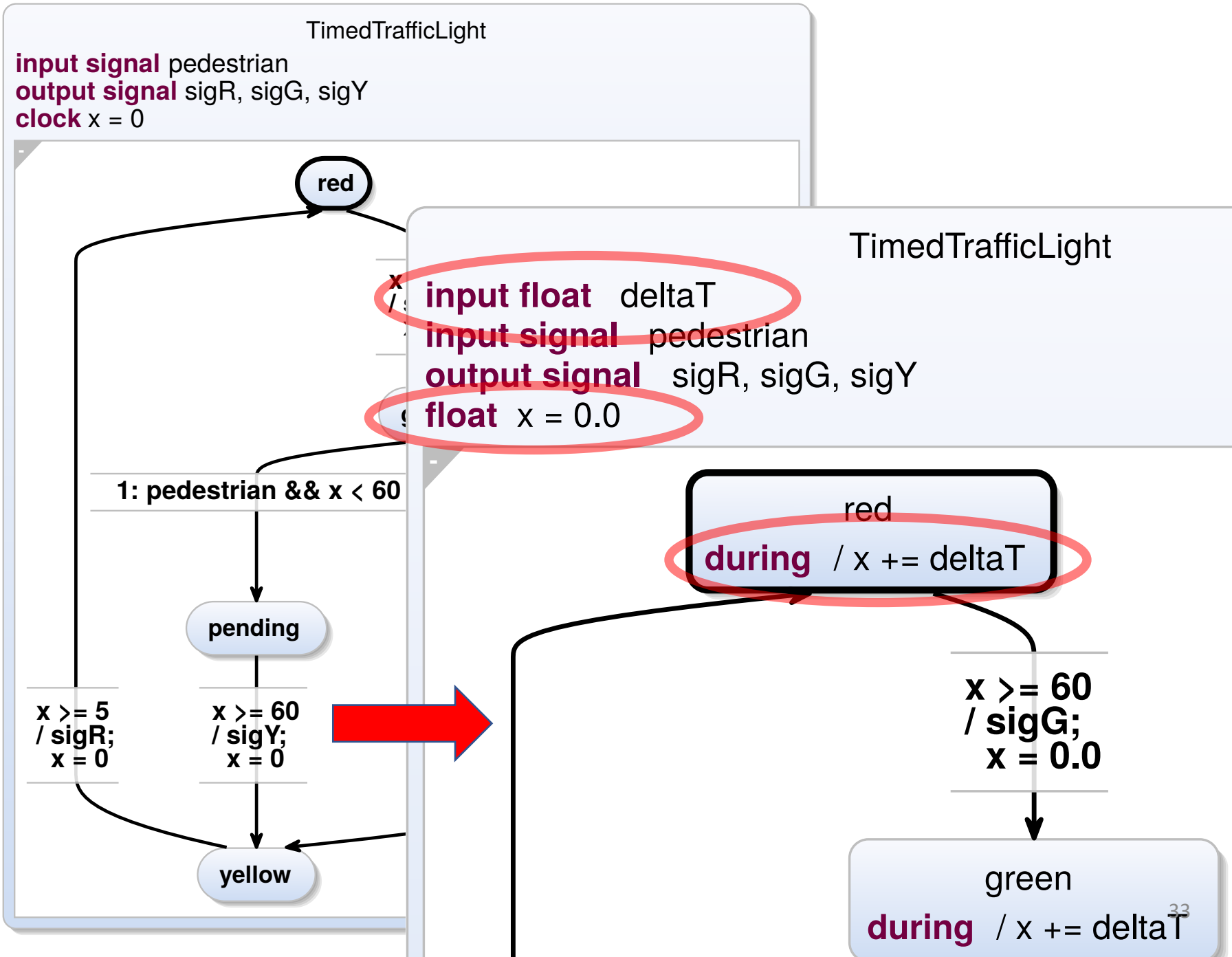
Recall: Traffic Light in SCCharts



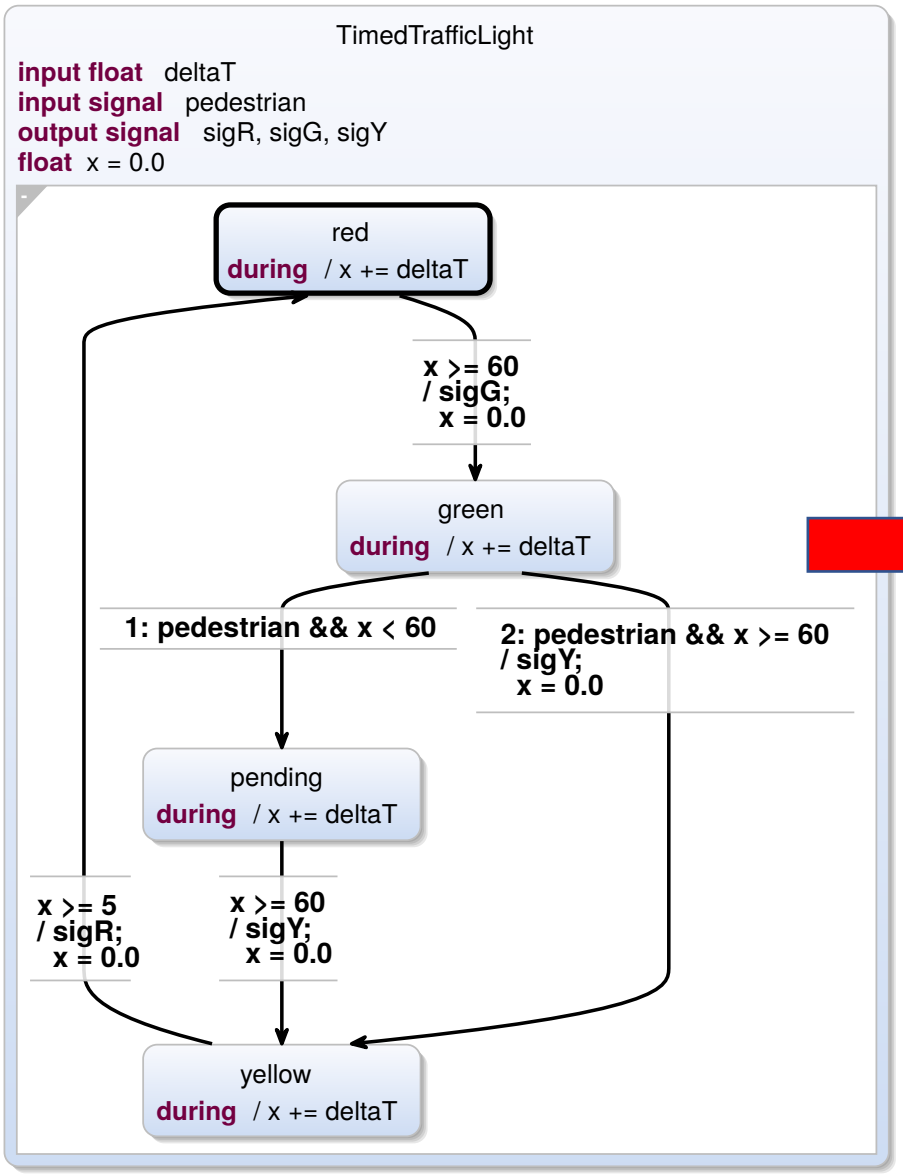
1st: Expand Clock

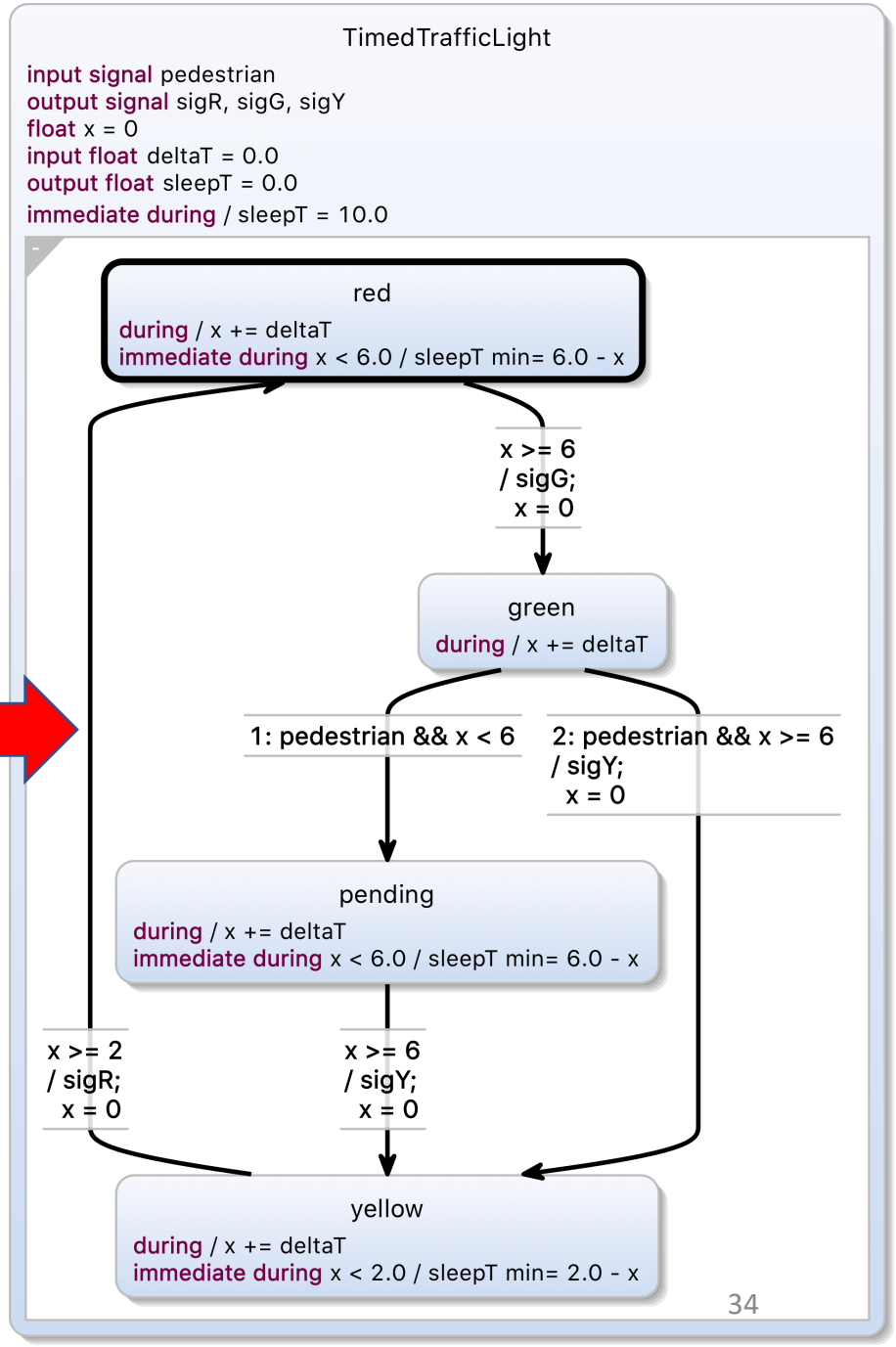





TimedTrafficLight
input signal pedestrian
output signal sigR, sigG, sigY
clock x = 0
red
1: pedestrian && x < 60
pending
x >= 5
/ sigR;
x = 0
x >= 60
/ sigY;
x = 0
yellow
TimedTrafficLight
input float deltaT
input signal pedestrian
output signal sigR, sigG, sigY
float x = 0.0
red
during / x += deltaT
x >= 60
/ sigG;
x = 0.0
green
during / x += deltaT

2nd: Add Dynamic Ticks

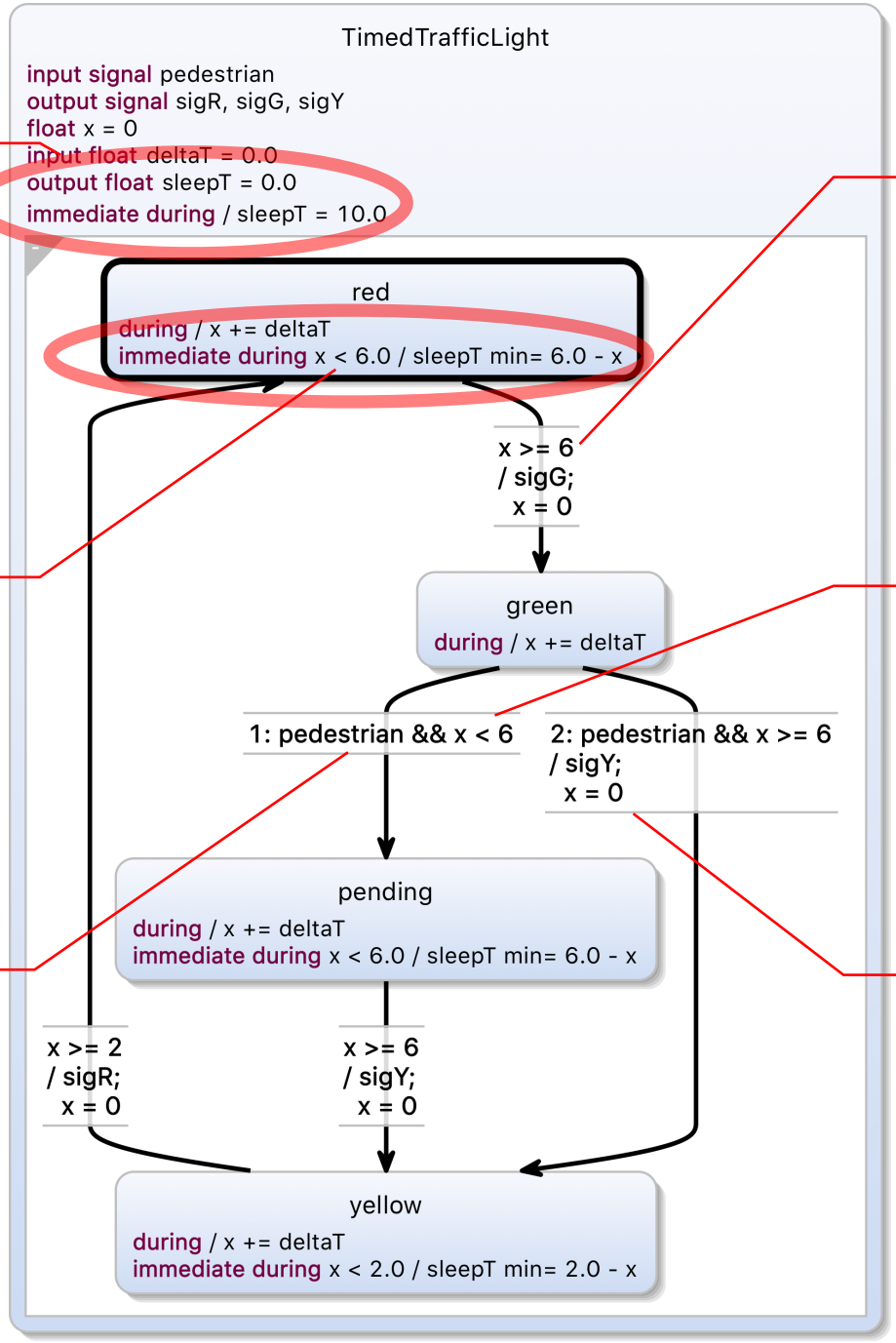




For sleepT, SC MoC orders reset and update(s)

Must guard timeout tick (missed this in paper!)

No reset here



"x ≥ 6" induces *lower timing bound* (*ltb*) of 6

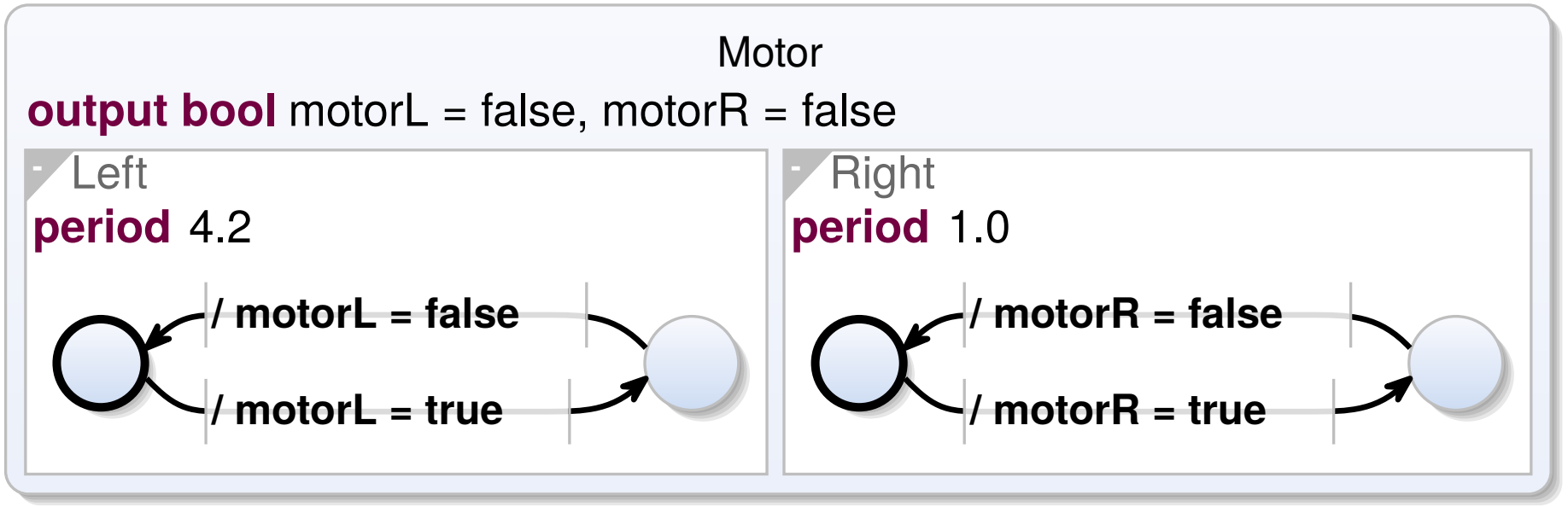
"x < 6" cancels ltb of 6

For x, SC MoC orders reset, update, read

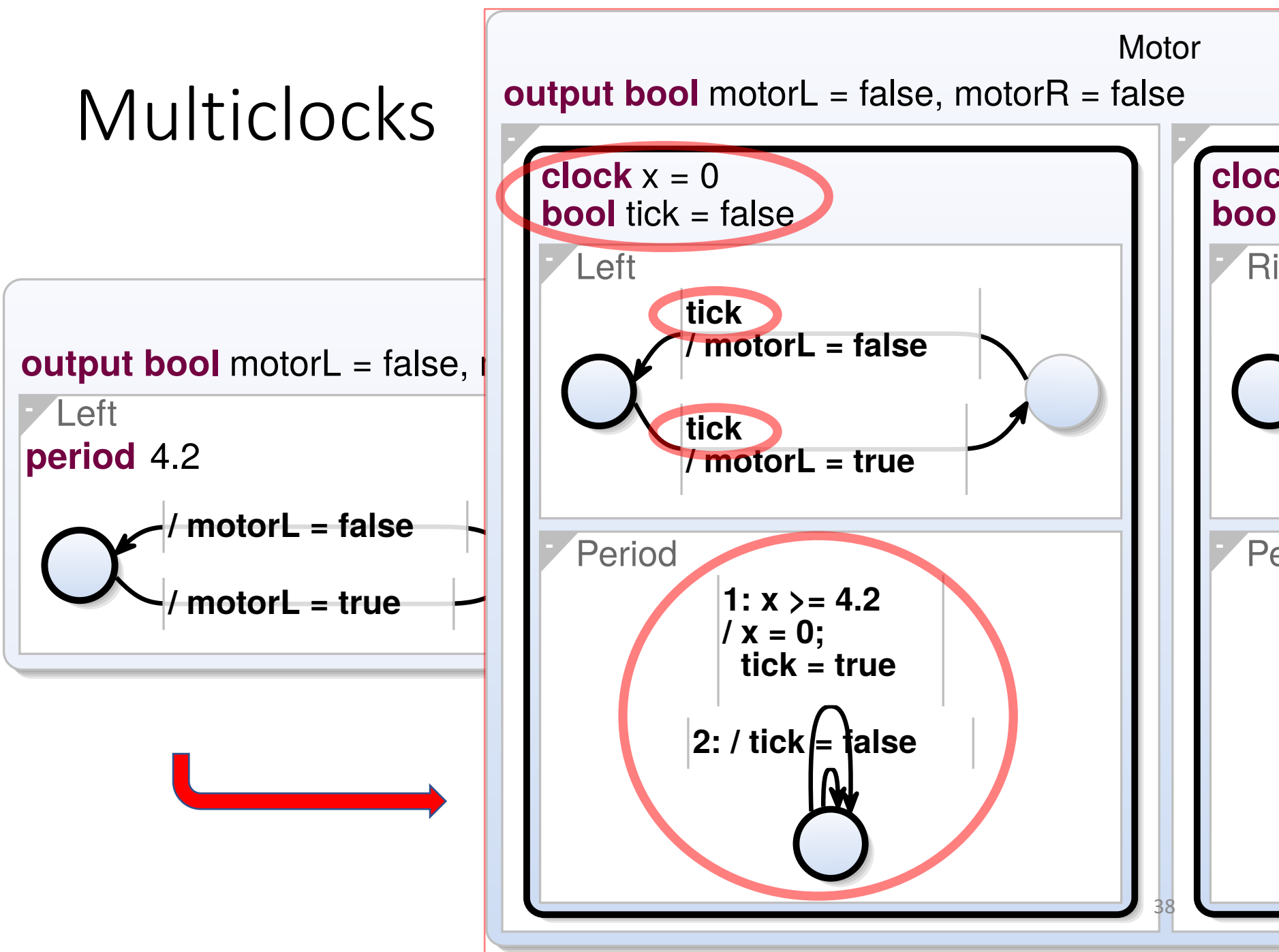
Roadmap

1. Traffic Light Example
2. Execution Models
3. Dynamic Ticks
4. Time in SCCharts: “clock”
5. Multiclocks in SCCharts: “period”
6. Demo

Multiclocks

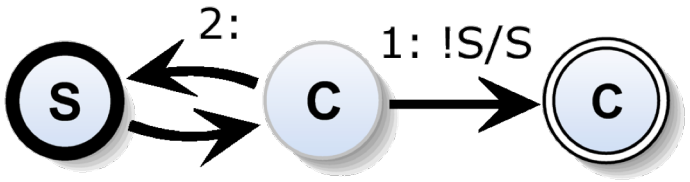


Multiclocks



Roadmap

1. Traffic Light Example
2. Execution Models
3. Dynamic Ticks
4. Time in SCCharts: “clock”
5. Multiclocks in SCCharts: “period”
6. Demo



SCCharts

http://www.sccharts.com/

KIELER

The Key to Efficient Modeling

http://www.rtsys.informatik.uni-kiel.de/en/research/kieler

ELK

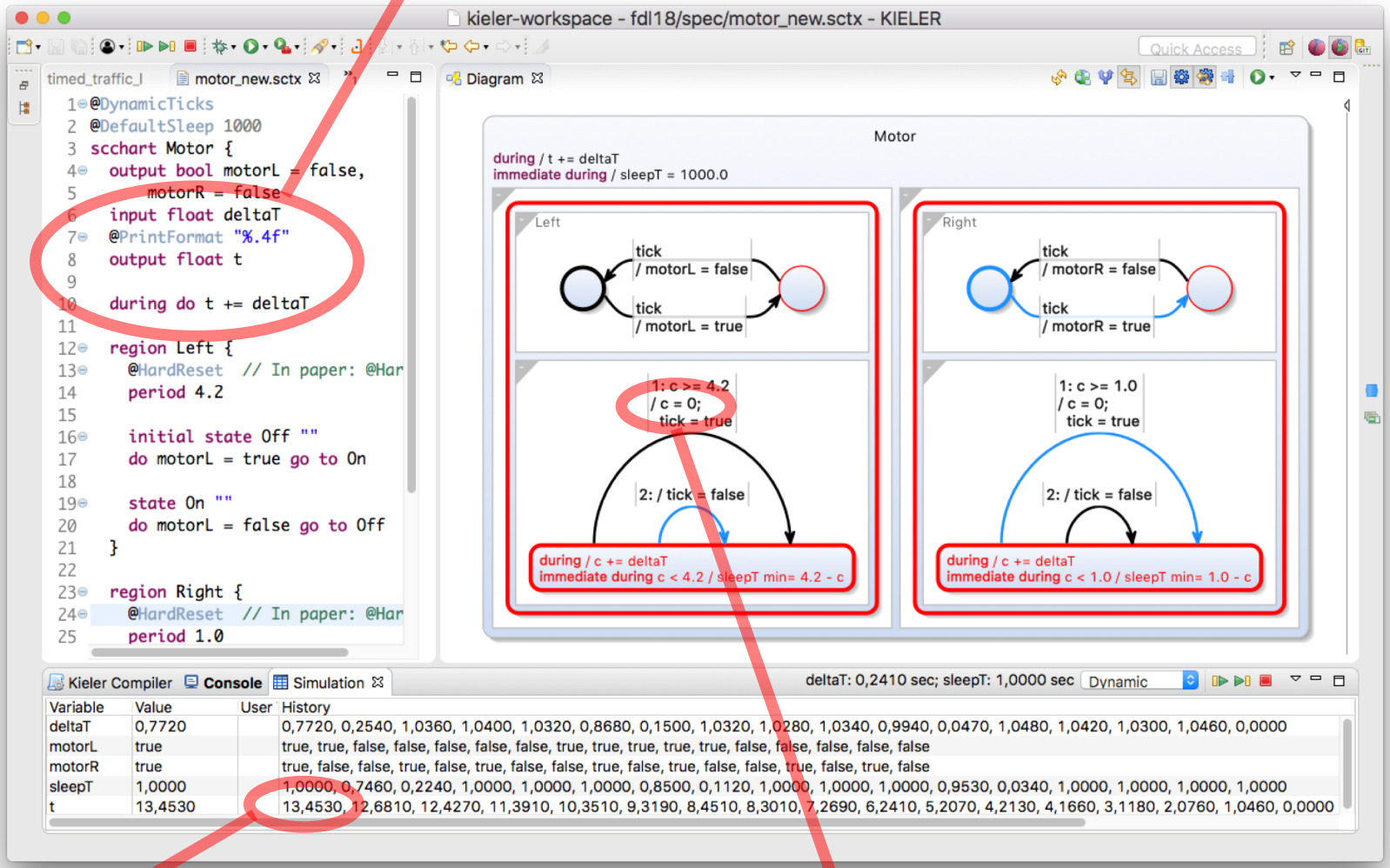
Eclipse Layout Kernel

https://www.eclipse.org/elk/

All available as open source under EPL

”avoid accumulations of timer imperfections”

Grab deltaT from environment, derive from it physical time t

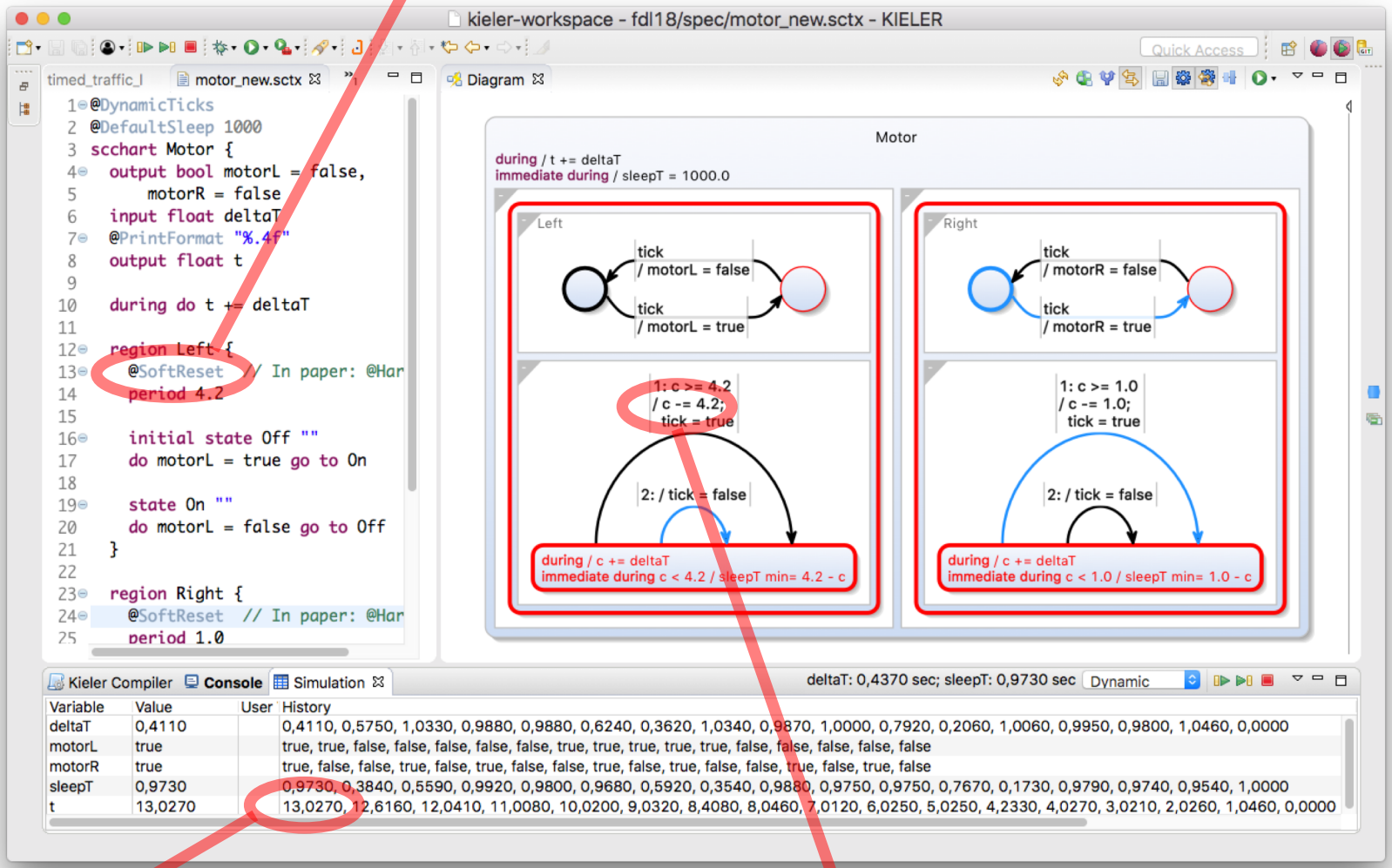


PROBLEM: After 13 sec, accumulated 0.453 sec delay!

The culprit: always reset clocks to 0!

”avoid accumulations of timer imperfections”

SOLUTION: Change @HardReset (in paper) to @SoftReset (now default)



After 13 sec, accumulated only 0.027 sec delay!

Instead of reset to 0, subtract previously requested time-out

Summary

- Timed automata used not just for verification, but also for synthesis
- Synchronous execution model cleanly contains non-determinism, at timing I/O-interface
- Can extend easily to multi-clock design
- Multiform notion of time retained – but package “time” not as events, but clocks (represented as, e.g., integers)
- Added two keywords (clock, period) as extended SCChart features
- Further annotations (@HardReset, @DefaultSleep) to control external interface
- Same concepts can be applied to other synchronous languages

ADVERTISEMENT

Forum on specification & Design Languages

2–4 September 2019 | Southampton, UK

FDL is a well-established international forum to exchange experiences and promote new trends in the application of languages, their associated design methods, and tools for the design of electronic systems. Electronic systems of interest to FDL include (but are not limited to) those that are used in Internet of Things (IoT), Cyber-Physical Systems (CPS), mixed criticality embedded systems, embedded systems for high-performance computing, automated driving and driver assistance, real-time systems, reconfigurable and secure computing.

FDL stimulates scientific and controversial discussions within and in-between scientific topics as described below. The program structure includes original research sessions, tutorials, panels, and technical discussions. “Wild and Crazy Ideas” and work in progress are welcome.

We welcome authors to submit manuscripts on topics including, but not limited to:

- Languages and formalisms in the design, test, verification, and simulation of electronic systems.
- Requirements and property specifications, models of computations, automata, networks, model- and component-based design.
- Platform modeling and abstraction, and system-level design languages.
- Synchronous and functional languages for reactive and concurrent systems.
- System design involving modern approaches such as machine learning and its verification, as well as modern computing architectures such as energy-efficient and high-performance computing, accelerators including GPUs and FPGAs, and IoT applications.
- Languages and compilers for multi/many-core and heterogeneous architectures.
- Formal methods and languages for model development and verification
- Languages in model-based design of intelligent systems and machine learning
- High-level hardware and software synthesis, virtual prototyping, and design space exploration.

Keynotes: David Broman | KTH Royal Institute of Technology, Stephen Edwards | Columbia University, Marc Pouzet | École Normale Supérieure

Submissions:

We invite **full research papers**, for oral presentation, which cover novel and complete research work supported by experimental results. We also invite **short papers**, for interactive presentations/posters, which may include "wild and crazy ideas", work in progress, case studies or industrial experience reports.

Authors should submit papers in double column, IEEE format as PDF through the submission system. A full research paper has a maximum of 8 pages, short papers may have up to 4 pages. Submitted papers must be anonymous (double blind), must describe original unpublished work, and must not be under consideration for publication elsewhere. Full research papers may be accepted as short papers.

Call for Special Sessions:

Special Sessions should focus on a topic which is of particular interest to the FDL audience. Special Sessions consist of two to four invited talks. Speakers are requested to either submit a one page abstract of their presentation, or to submit a short or full paper that goes through the regular review and publication process.

Potential organizers of a Special Session must submit a brief proposal (no more than two pages) which describes the topic, the intended audience, as well as a list of possible speakers to fdl2019@easychair.org.

Publications:

Conference proceedings will be published in electronic form with an ISSN and an ISBN number and made available on IEEE Xplore.

In addition, an edited collection of extended versions of selected best papers will be published as a book by Springer.

Accepted papers must be presented by one of the authors. A full registration for each paper is required prior to the final paper version deadline.

IEEE
IEEE Council on Electronic Design Automation
IFIP WORKING GROUP 10.5 Design and Engineering of Electronic Systems
UNIVERSITÀ di VERONA Dipartimento di INFORMATICA
UNIVERSITY OF Southampton

General Chair: Tom J Kazmierski | Univ. of Southampton
Program Chair: Reinhard von Hanxleden | Kiel Univ.
Program Co-Chair and Local Chair: Terrence Mak | Univ. of Southampton
Special Session Chair: Daniel Grosse | Univ. of Bremen
Finance Chair: Franco Fummi | Univ. of Verona
Web Chair: Florenc Demrozi | Univ. of Verona
Panel/Tutorial Chair: Carna Zivkovic | TU Kaiserslautern

Important Deadlines (AoE):

Special Sessions:	Mar 22, 2019
Abstract Deadline:	Apr 19, 2019
Paper Deadline:	Apr 26, 2019
Author Notification:	Jun 21, 2019
Final Version:	Jul 19, 2019

Website: www.fdl-conference.org | Contact: fdl2019@easychair.org

- FDL’19: Southampton, Papers wanted!
- FDL’20: Kiel, Papers and PC wanted!
- FDL’21: France, Papers, PC and GC wanted!