The Informatics of Time and Events

Gérard Berry

http://www-sop.inria.fr/members/Gerard.Berry/

Professor at Collège de France

UPMC Colloquium, *Paris, October 24th, 2012*

Agenda

1. Why discussing time and events
2. Lines and cones of time, causality
3. Multiform time
4. Concurrency models
5. Synchronous circuits and constructive logic
6. Metastability and inter-clock zone protocols
7. Asynchronous and elastic circuits
8. Synchronous languages
9. Continuous vs. discrete time in modelers
10. Conclusion

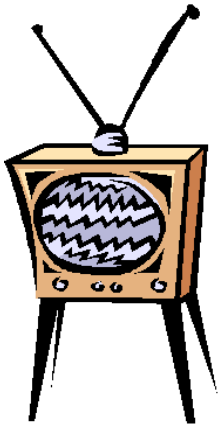
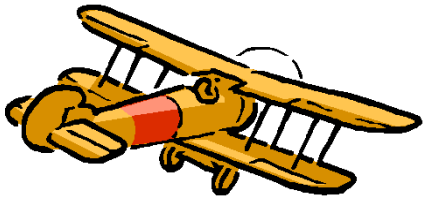
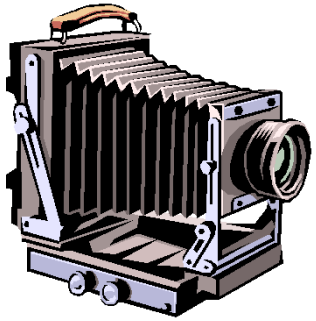
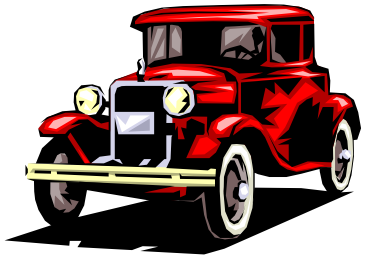
Why Discussing Time and Events

- Real-Time Embedded Systems
- Systems of Chips (SoCs)
- Simulators of physical systems
- Orchestration of Web Services
- Music composition and interpretation

Dealing with time and events is a key issue, becoming much more important in the 21th century

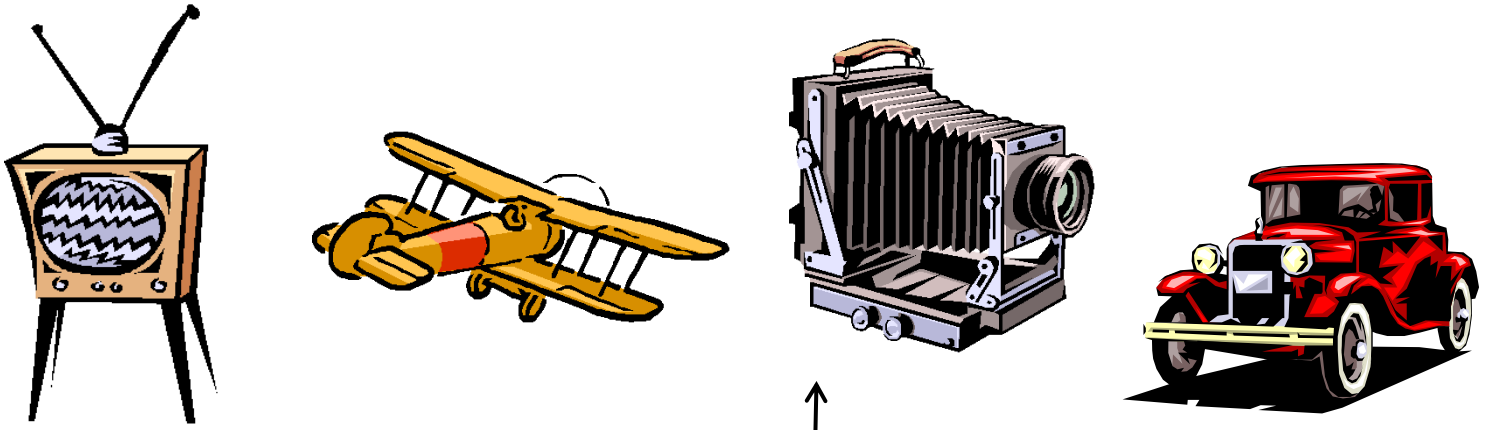
No provision in classical languages to handle them!

20th Century Embedded Systems

- Compact, single functionality
  - data-centric: continuous control, signal processing
  - control-centric: protocols, mode handlers, displays, etc.
- Mostly deterministic behavior
  - time-based control-theory
  - deterministic event handling, external-only non-determinism

20th Century Embedded Systems



Infrastructure
PLCs
µP, ECUs
simple OS
basic displays

Manual Implementation
manual mathematics
textual specification
ASM, C, ADA coding
standard compiling
extensive testing

Formal implementation
ODE math modeling
formal specification
high-level coding
automatic codegen
formal verification

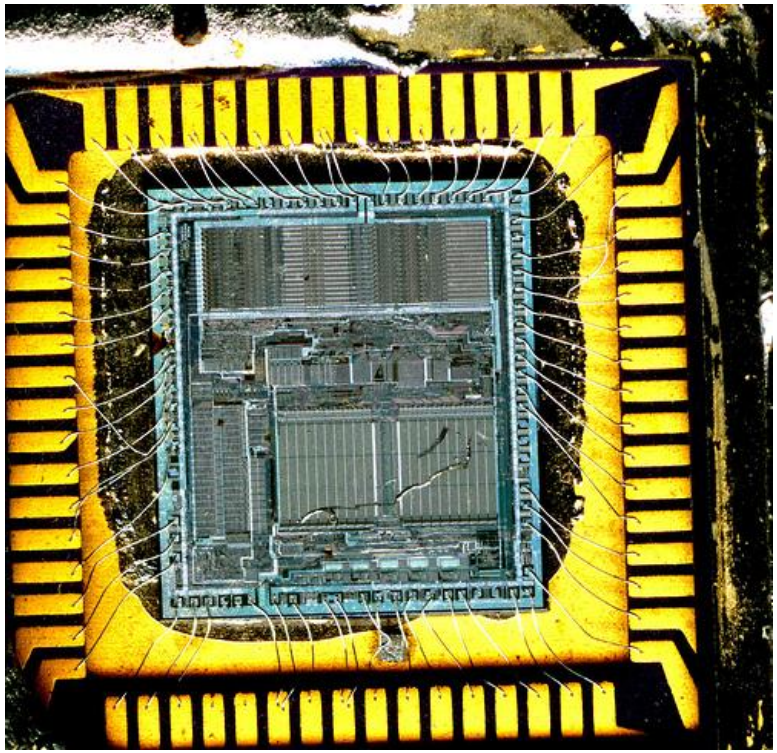
21th Century Embedded Systems

- Much more complex, distributed, non-deterministic
  - many functions on each SoC or ECU, many clocks
  - subsystem functions need to be coordinated
  - networks everywhere: NoCs, PAN, LAN, etc.

- Mix of styles
  - distributed continuous control + FSMs
  - GALS: Globally Asynchronous Locally Synchronous
  - mathematical modeling: continuous + discrete time

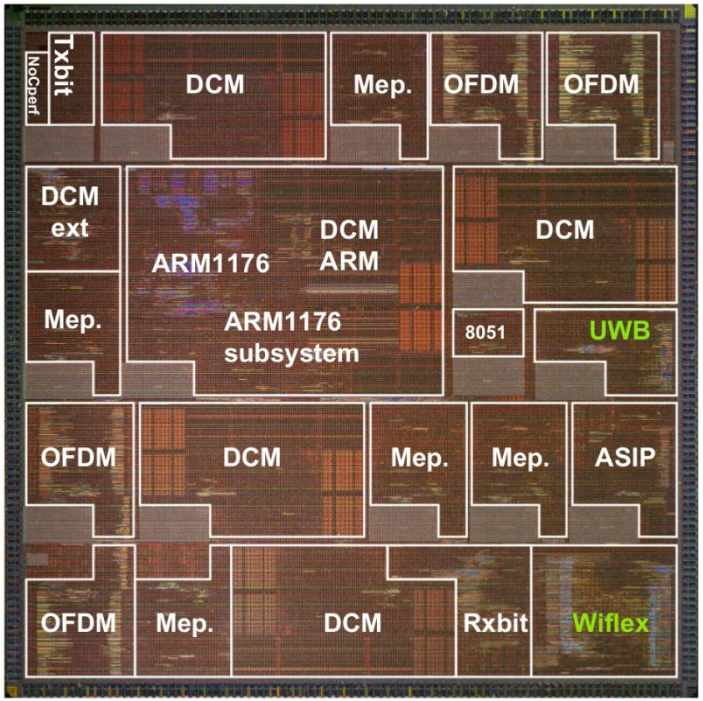
- Web-based “best effort” interfaces
  - ex.: controlling the house from a smartphone

20th Century Chips



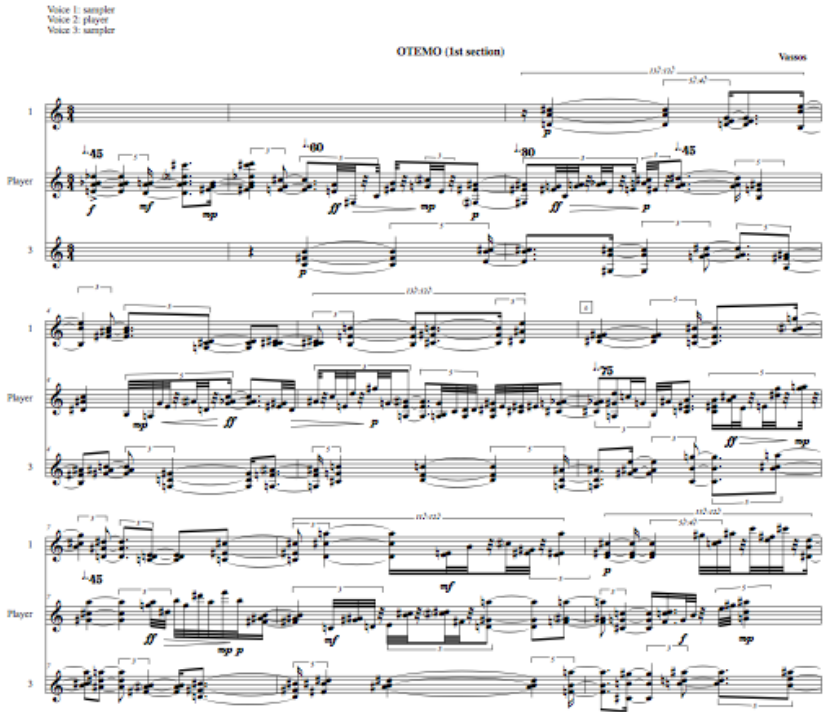
- Simple functionality
- Single clock
- Simple physics
- No power-handling

21th Century Chips

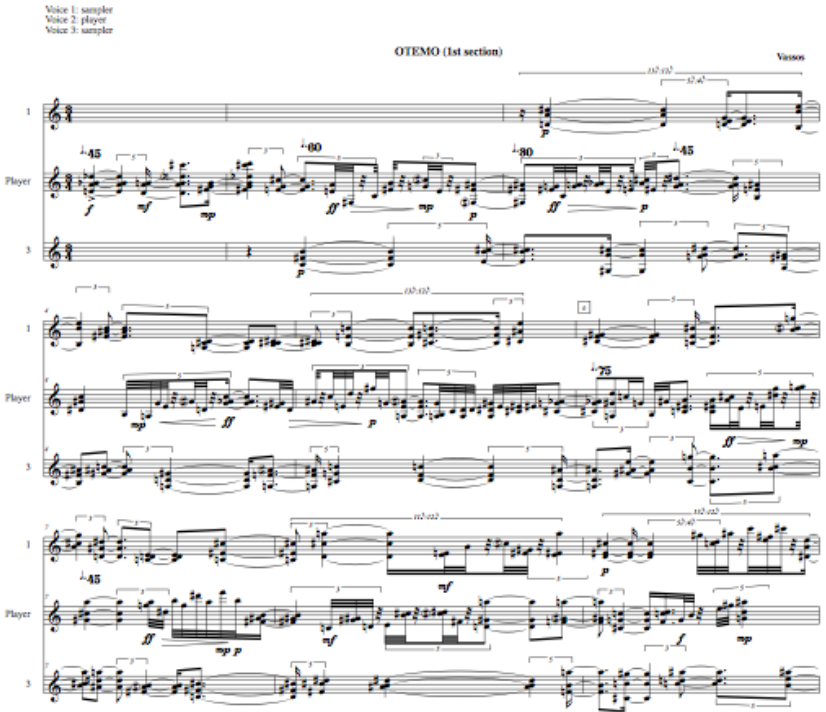


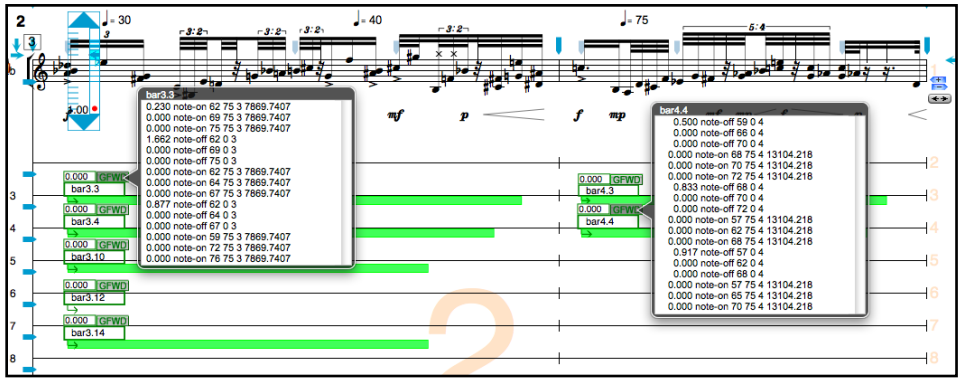
- Multiple IPs, multiple clock zones, complex power
  risks of metastability ⇒ complex communication protocols
- Need for multiple simulation levels
  build the software before the chip ⇒ TLM simulation

20th Century : Human Music and Score



21st Century: Adaptive Human / Electronics



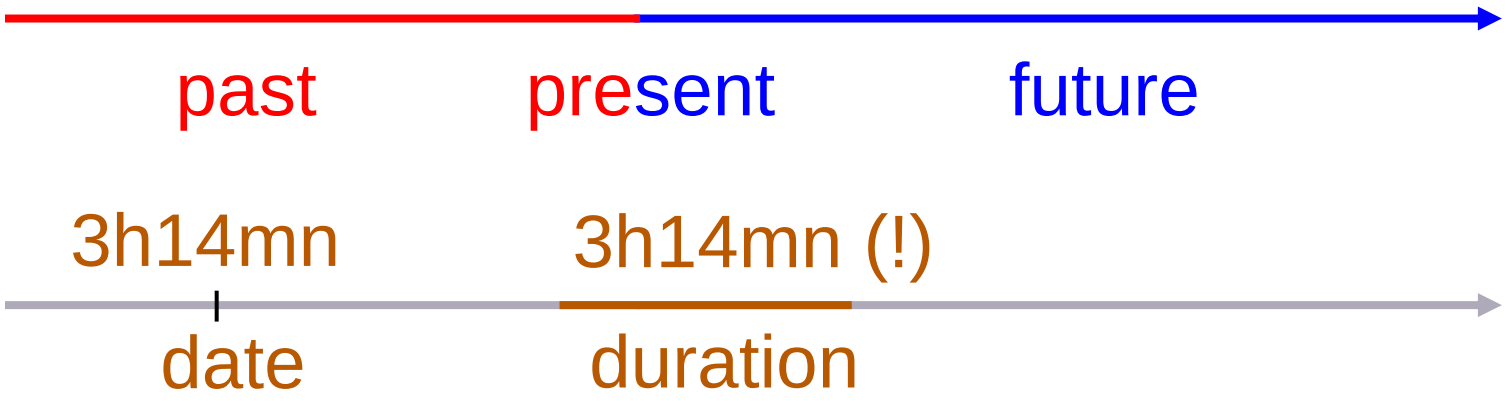


Algorithmic Score

Agenda

1. Why discussing time and events
2. Lines and cones of time, causality
3. Multiform time
4. Concurrency models
5. Synchronous circuits and constructive logic
6. Metastability and inter-clock zone protocols
7. Asynchronous and elastic circuits
8. Synchronous languages
9. Continuous vs. discrete time in modelers
10. Conclusion

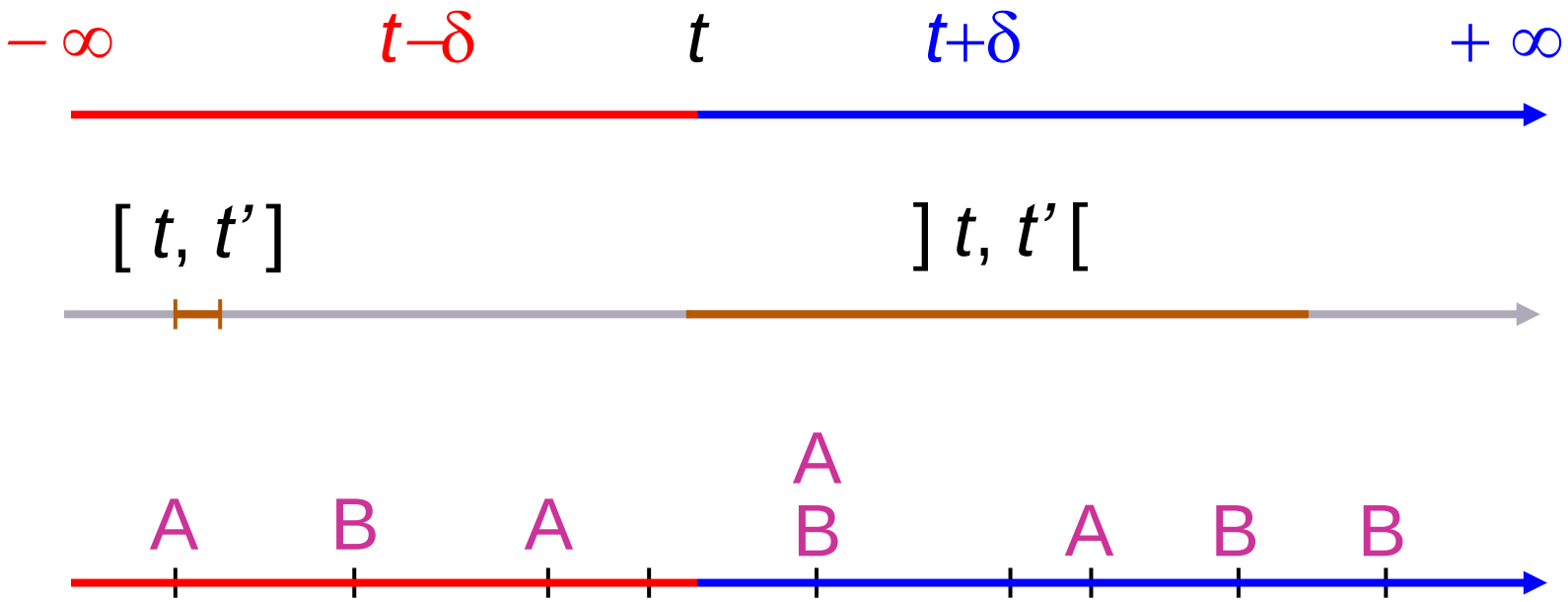
The Line of Time



- present is an instant between past and future
- Strange date numbering: 01/01 at 0h00

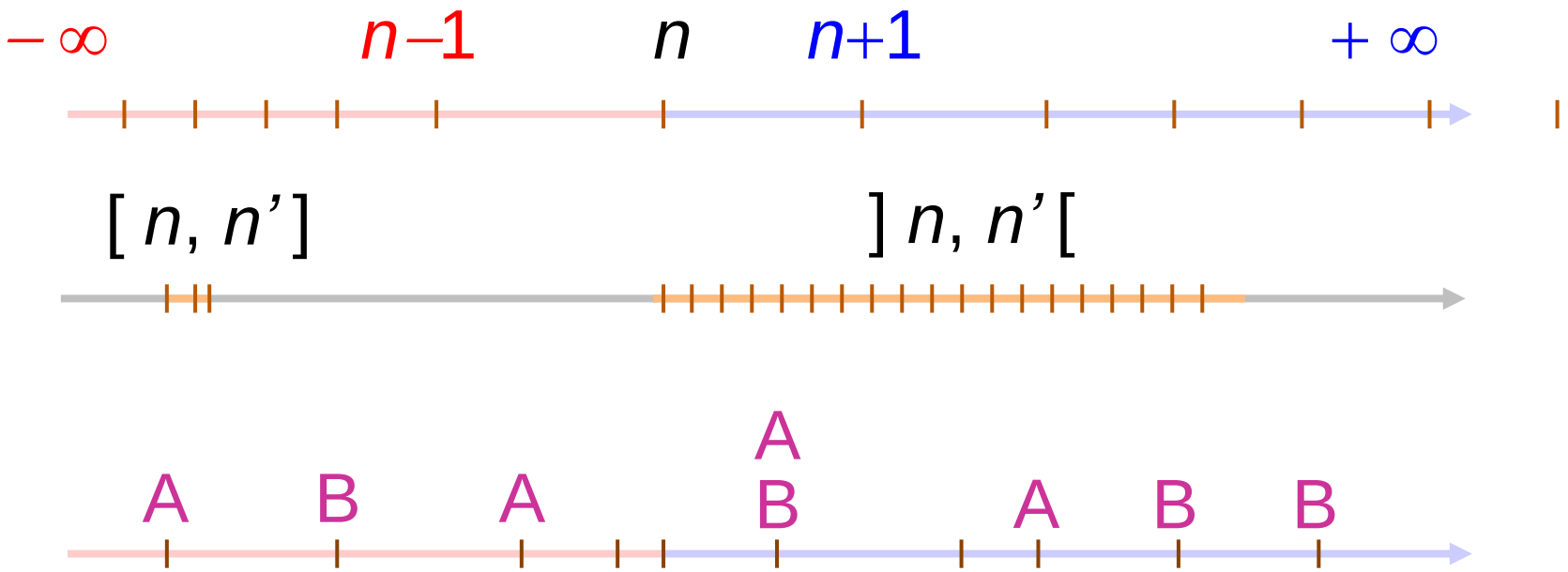
In the past, there was more future than nowadays
(le Chat)

Mathematical Continuous Time



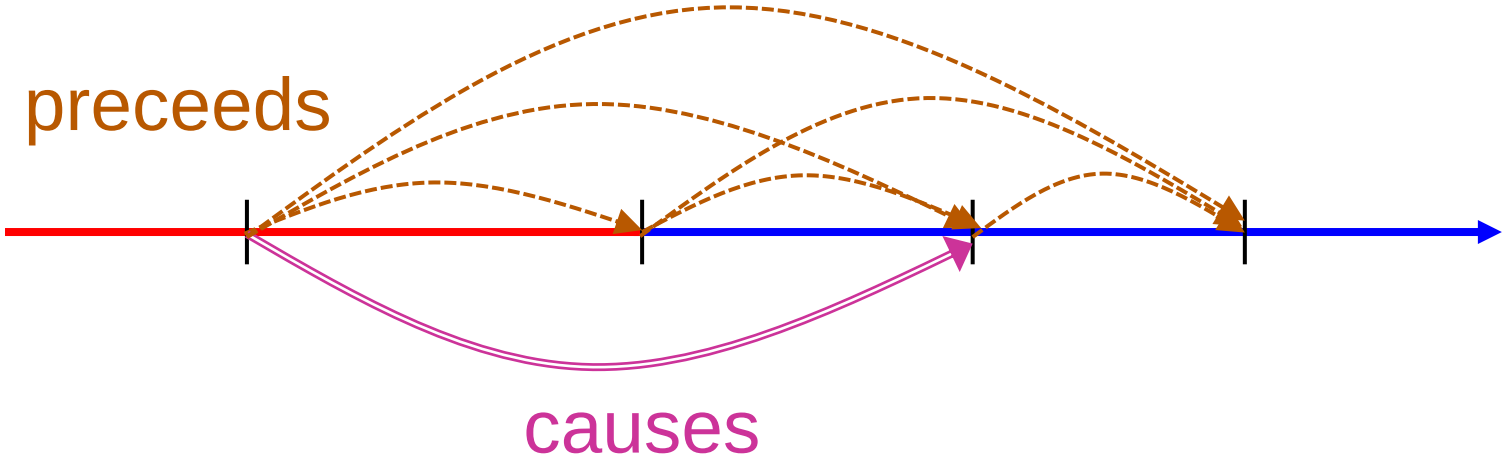
Events

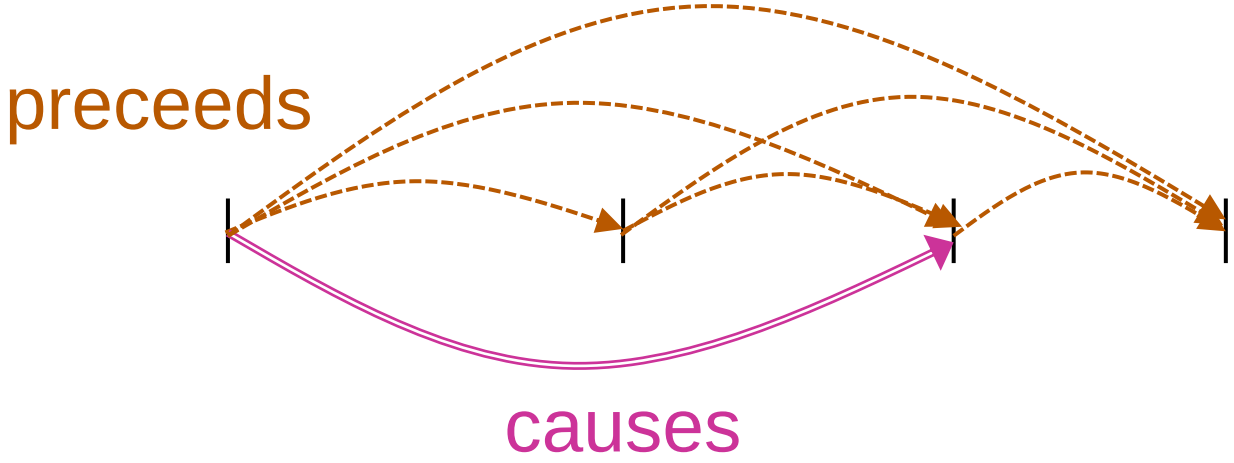
Mathematical Discrete Time



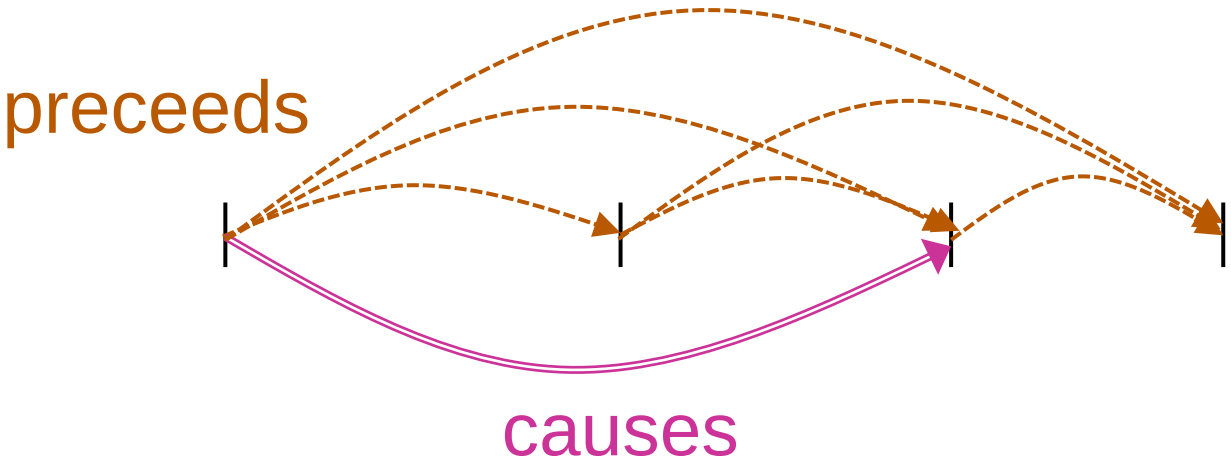
Events

Precedence vs. Causality





Linear Temporal Logic



always causes? eventually

□ (Rain ⇒ ◊ FairWeather)

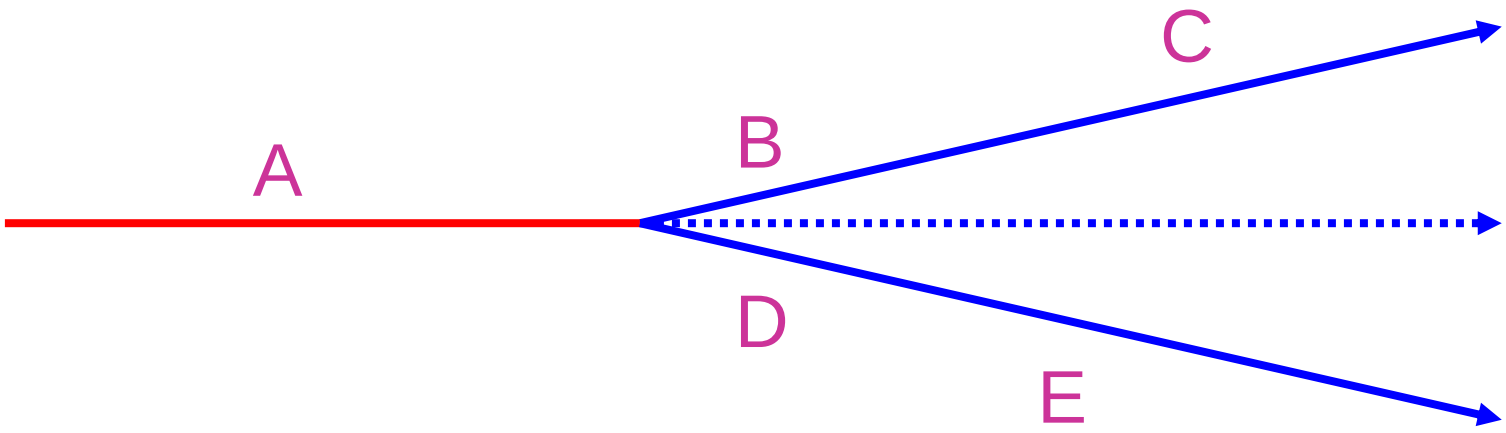
instant predicates

History and Debugging are about uncovering causality from precedence

Is Time a Physical or Logical Concept?

- In physics, is time continuous or discrete?
  - it depends on which physics !
  - discrete is an approximation of continuous, and conversely !
- Do instants have a thickness?
  - l'espace d'un instant → Einstein
- Can we act in zero-time?
  - it depends on which physics !
- How do the times of different actors compare?
  - A looong history of time measurement / adjustment
  - made much more complex by relativity theories: GPS
- Is linear time enough for reasoning about systems?
  - No, by far !

The Cone of Time



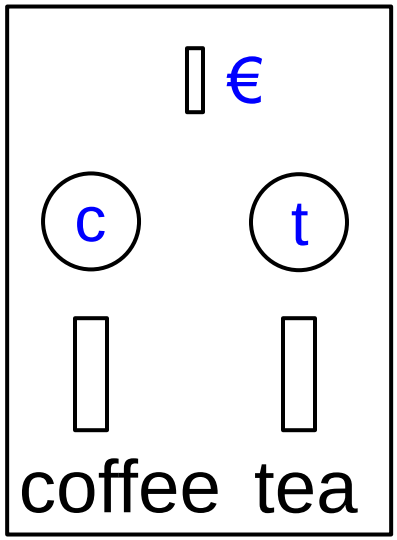
Knowing that I have done A:

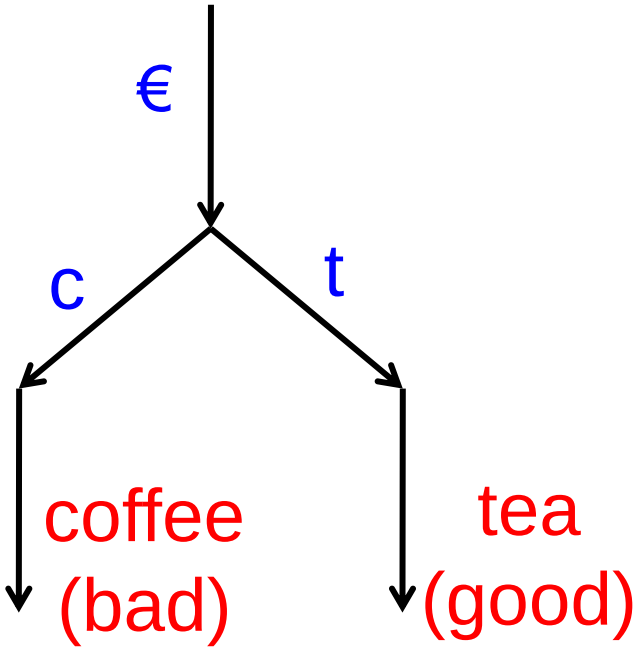
if I do B, I will get C

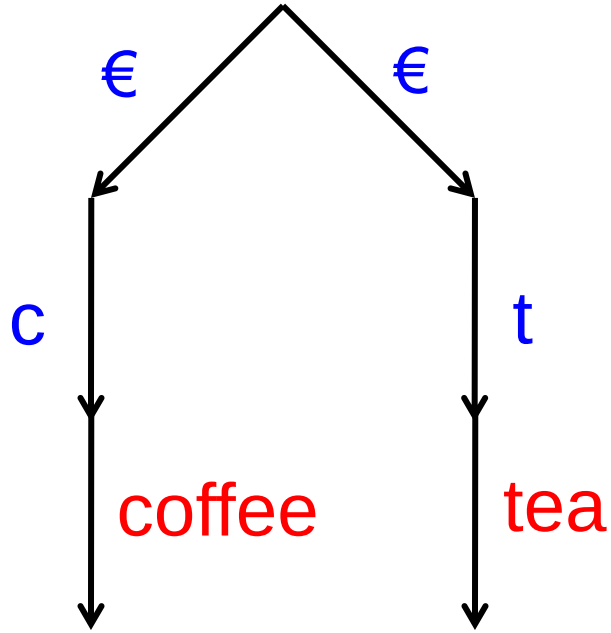
but if I do D, I will get E

$\Rightarrow$ branching-time temporal logics

British vs. French Coffee Machine





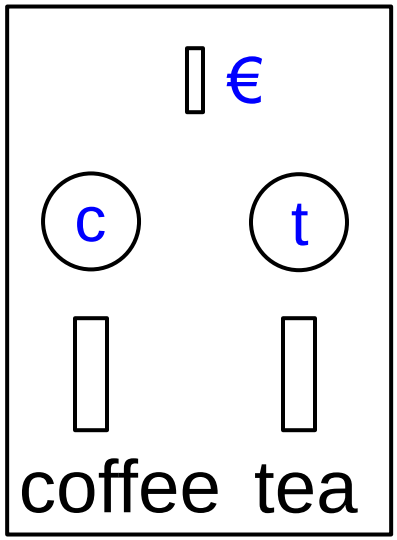


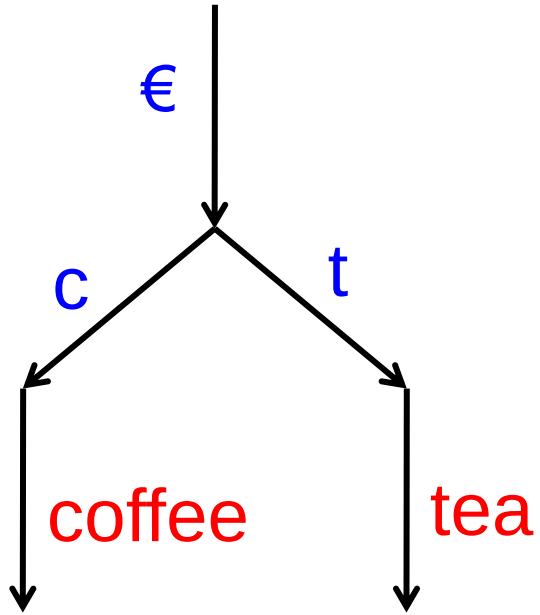
Which one do you prefer?

Equivalent w.r.t. linear-time traces

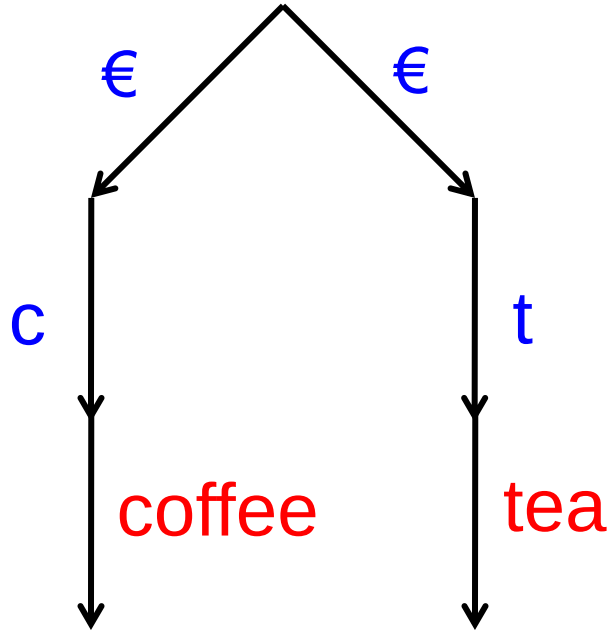
(€.c.coffee + €.t.tea)*

British vs. French Coffee Machine





OK

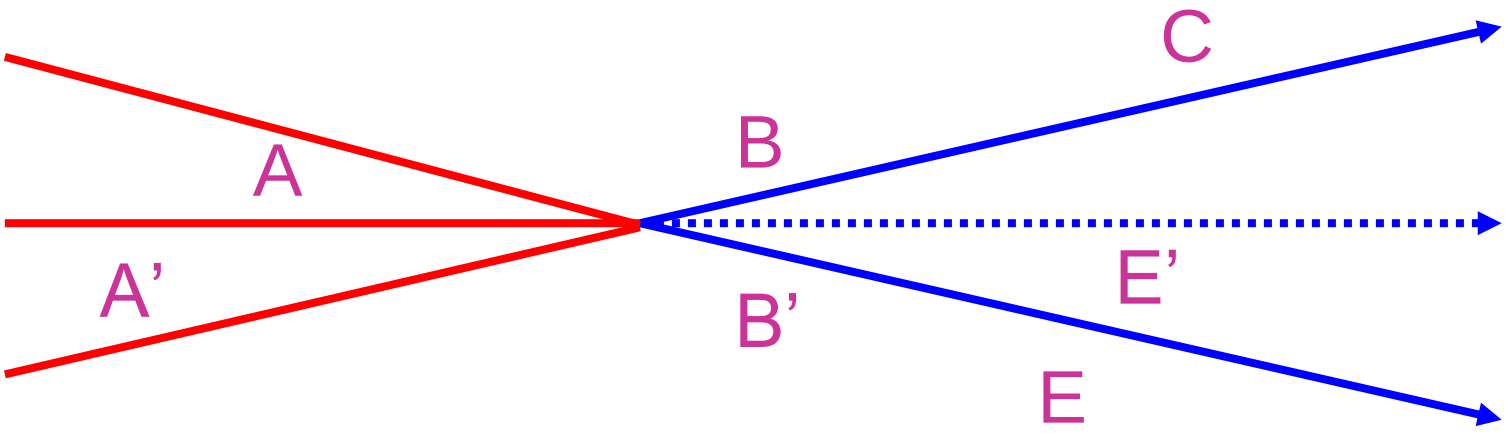


NOK

Not equivalent w.r.t. branching-time logic

$AG(€ \Rightarrow EF(coffee) \wedge EF(tea))$

The Double Cone of Time



Had I known, A’ would have been much better than A

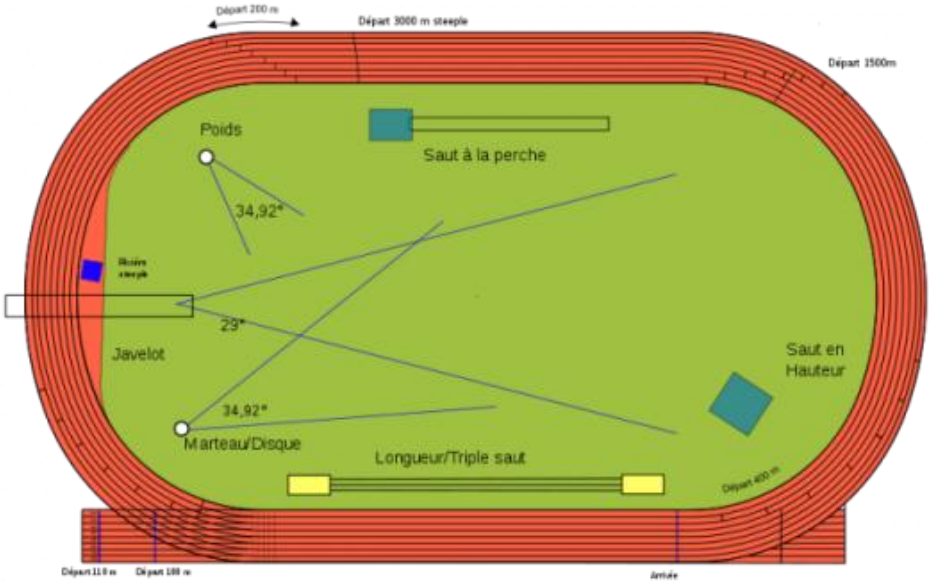
I would have got C without even doing B

and, by doing B’, I would have got E’, better than E !

Agenda

1. Why discussing time and events
2. Lines and cones of time, causality
3. Multiform time
4. Concurrency models
5. Circuits and constructive logic
6. Metastability, inter-clock zone protocols
7. Asynchronous and elastic circuits
8. Synchronous languages
9. Continuous vs. discrete time in modelers
10. Conclusion

Multiform Time: Clock Hierarchies



Second → Hour → Morning

Meter → Lap

Step

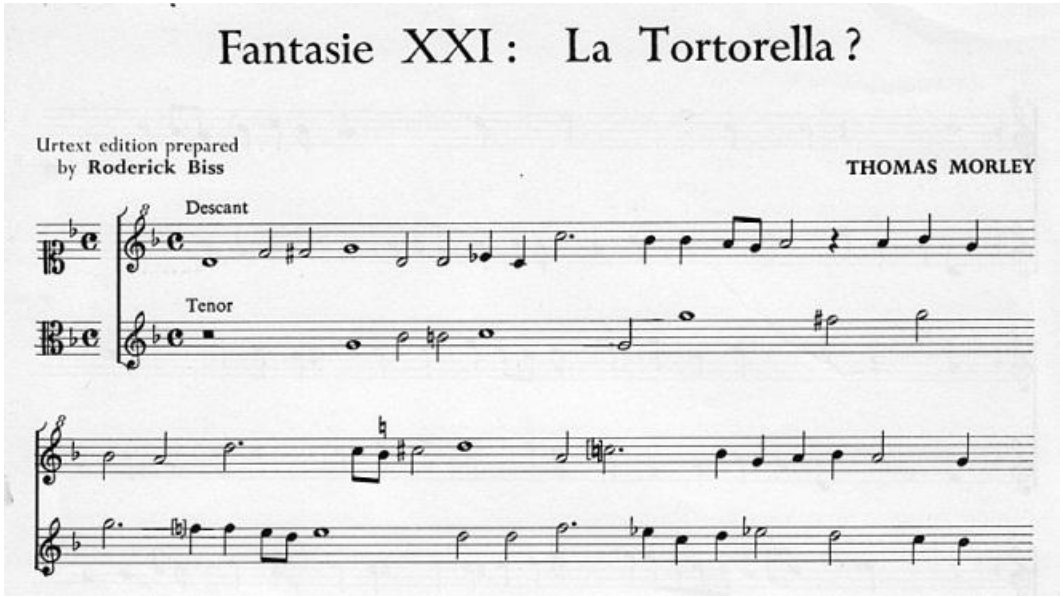
HeartBeat → HeartAttack

The Esterel Runner

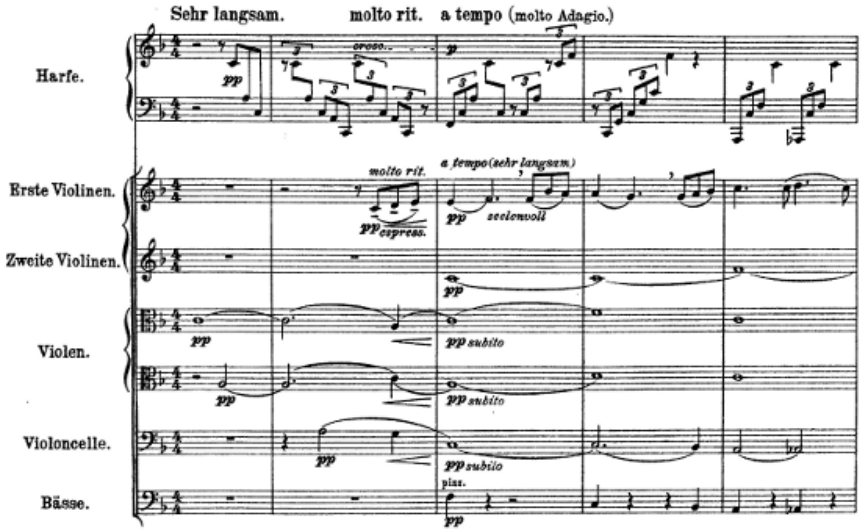
```
trap HeartAttack in
    every Morning do
        abort
            loop
                abort  run Slowly  when 100 Meter ;
                abort
                    every Step do
                        run Jump || run Breathe || CheckHeart
                    end every
                when 15 Second ;
                run FullSpeed
            each  Lap
        when  4 Lap
    end every
handle HeartAttack fo
   run RushToHospital
end trap
```

exit HeartAttack

Music Score ⇒ Multiform Time



Thomas Morley
1557-1602

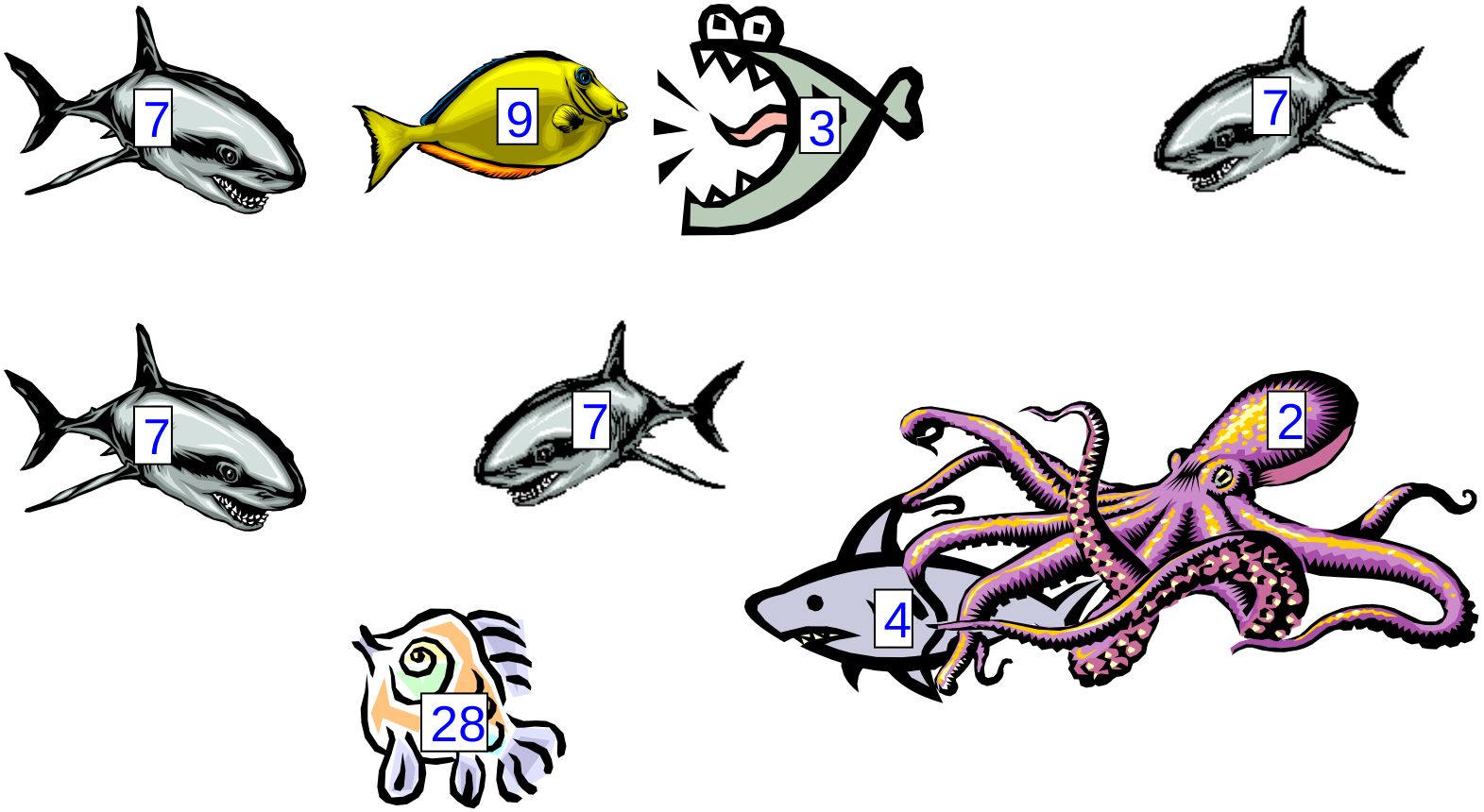


Gustav Mahler
1557-1602

Agenda

1. Why discussing time and events
2. Lines and cones of time, causality
3. Multiform time
4. Concurrency models
5. Circuits and constructive logic
6. Metastability, inter-clock zone protocols
7. Asynchronous and elastic circuits
8. Synchronous languages
9. Continuous vs. discrete time in modelers
10. Conclusion

The Asynchronous Darwin Sieve: p, kp → p



CHAM : Internet, cellular biology, etc.

The Synchrony / Vibration Model

Synchronous

Musicians and spectators neglect the speed of sound

Vibration

Acousticians deal with sound propagation

The Synchrony / Vibration Model

If the room is small enough

Vibration implements synchrony for spectators

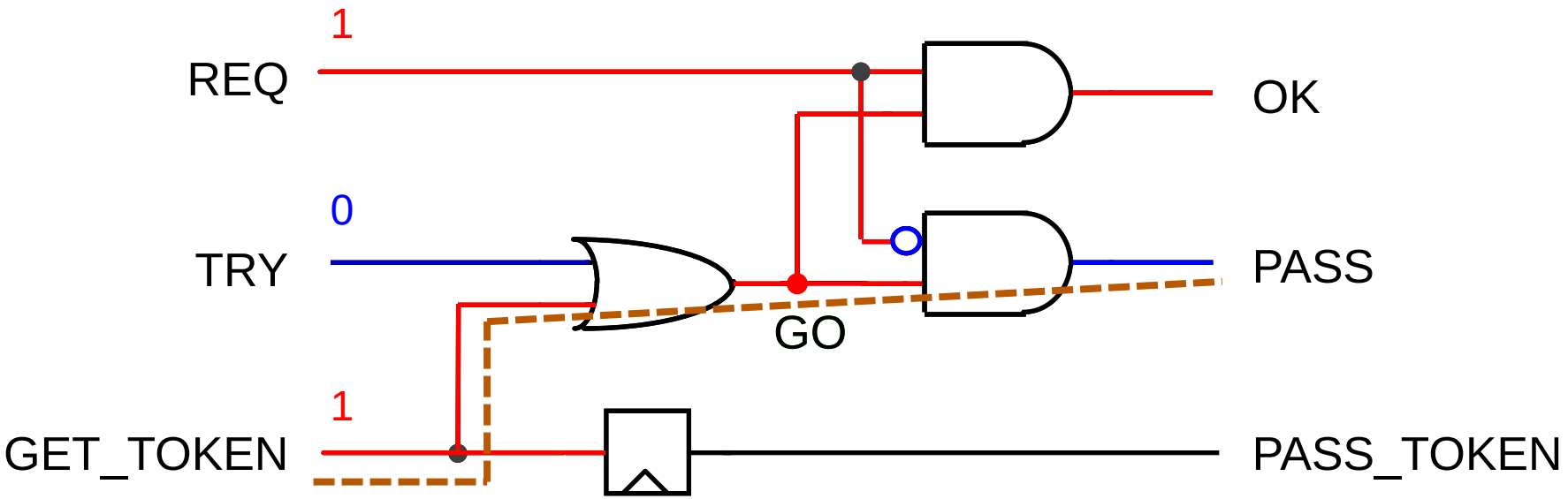
However, if the orchestra is big enough

Musicians need light + conductor to synchronize

Agenda

1. Why discussing time and events
2. Lines and cones of time, causality
3. Multiform time
4. Concurrency modes
5. Synchronous circuits and constructive logic
6. Metastability, inter-clock zone protocols
7. Asynchronous and elastic circuits
8. Synchronous languages
9. Continuous vs. discrete time in modelers
10. Conclusion

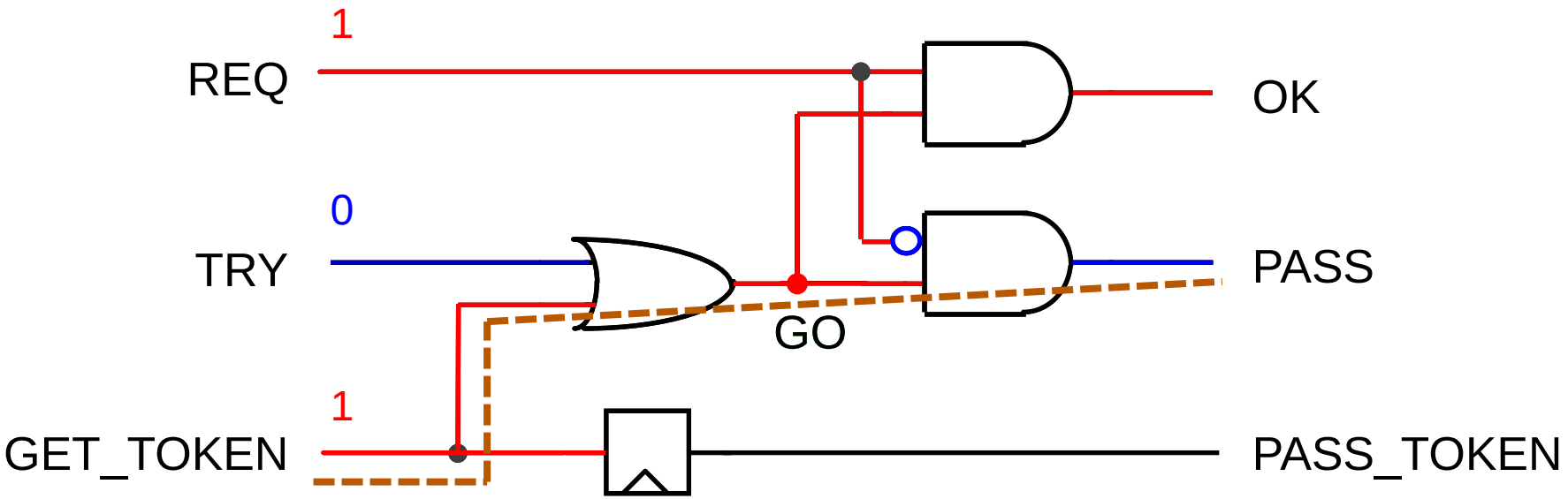
Synchronous Circuits: Vibration View



Since the network is acyclic, outputs stabilize in bounded time if inputs are kept constant

Stabilization time is determined by the critical path

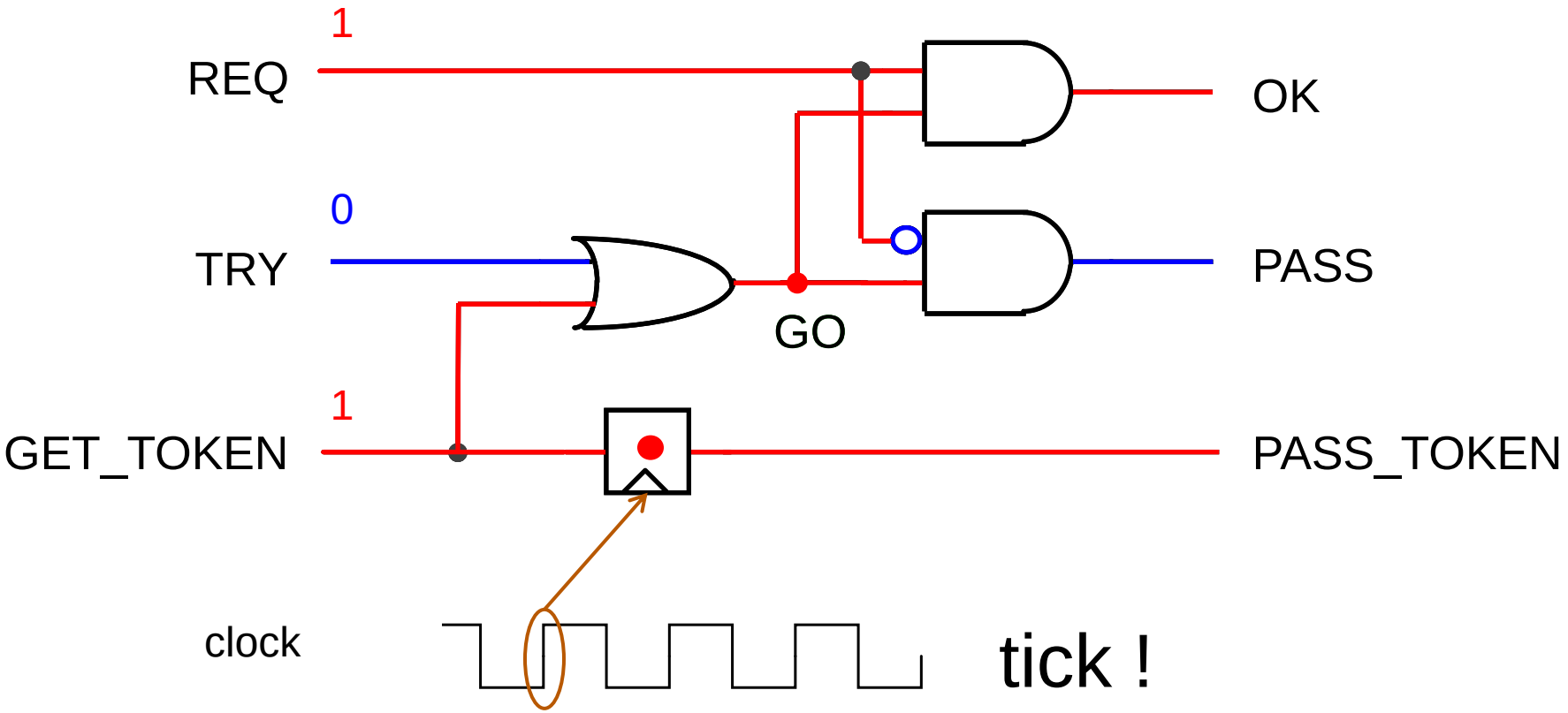
Synchronous Circuits: Synchronous View



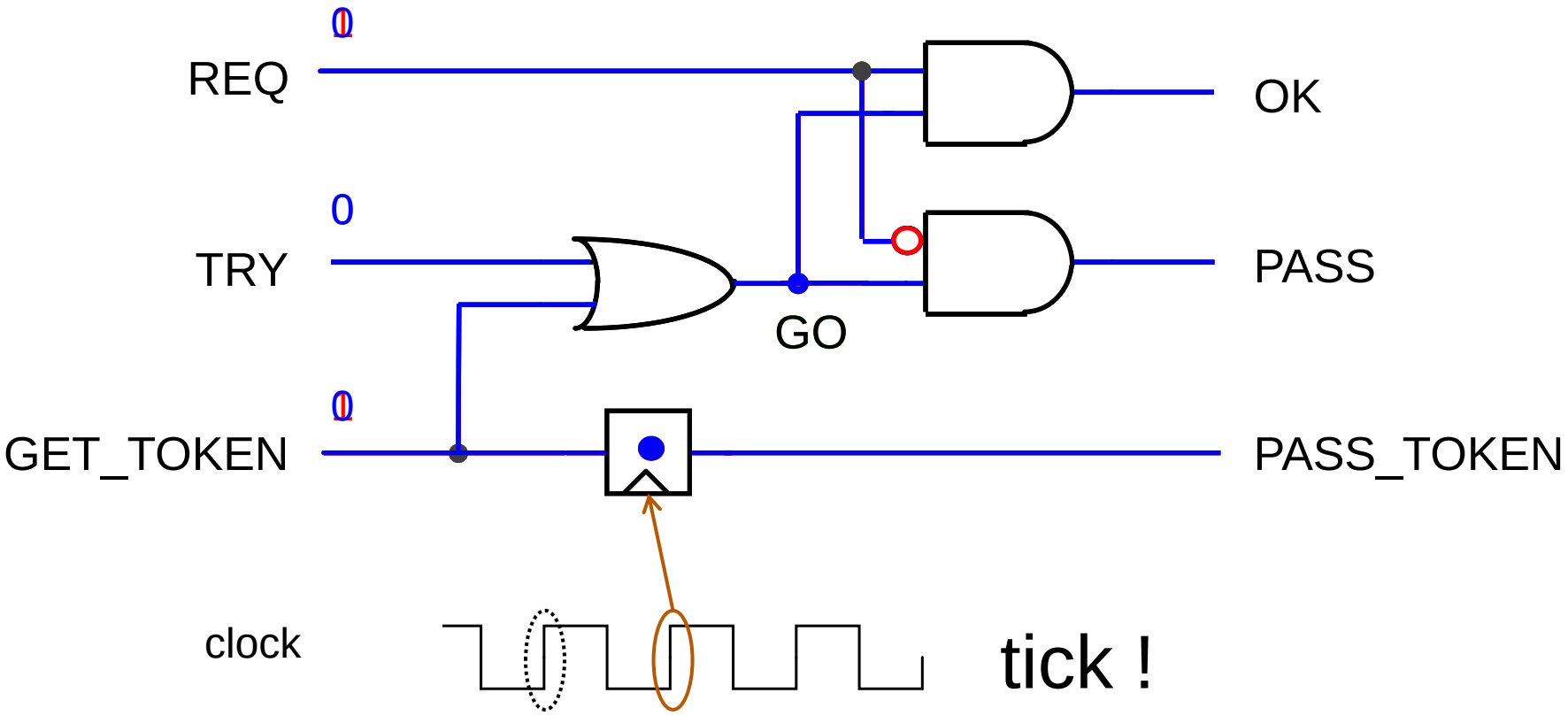
OK = REQ and GO
PASS = not REQ and GO
GO = TRY or GET_TOKEN
PASS_TOKEN = reg(GET_TOKEN)

Acyclic case:
waiting for the critical time ⇔ solving the equations

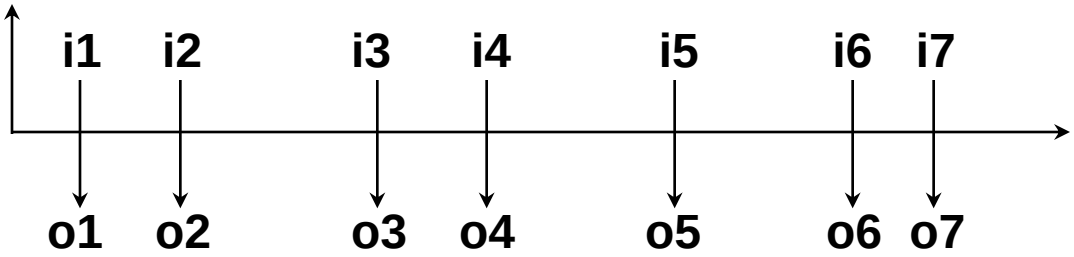
Sequential sampling



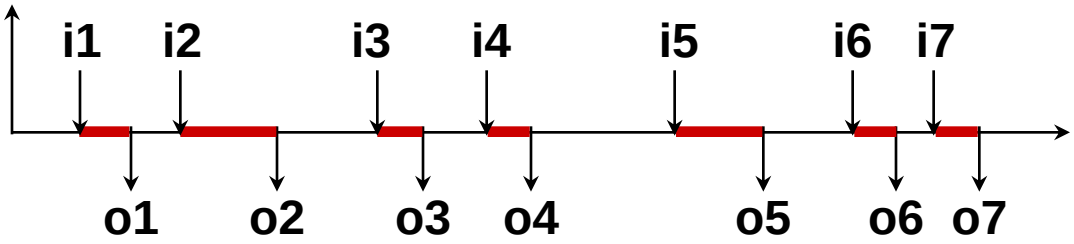
Sequential sampling



Logical Time vs. Physical Time



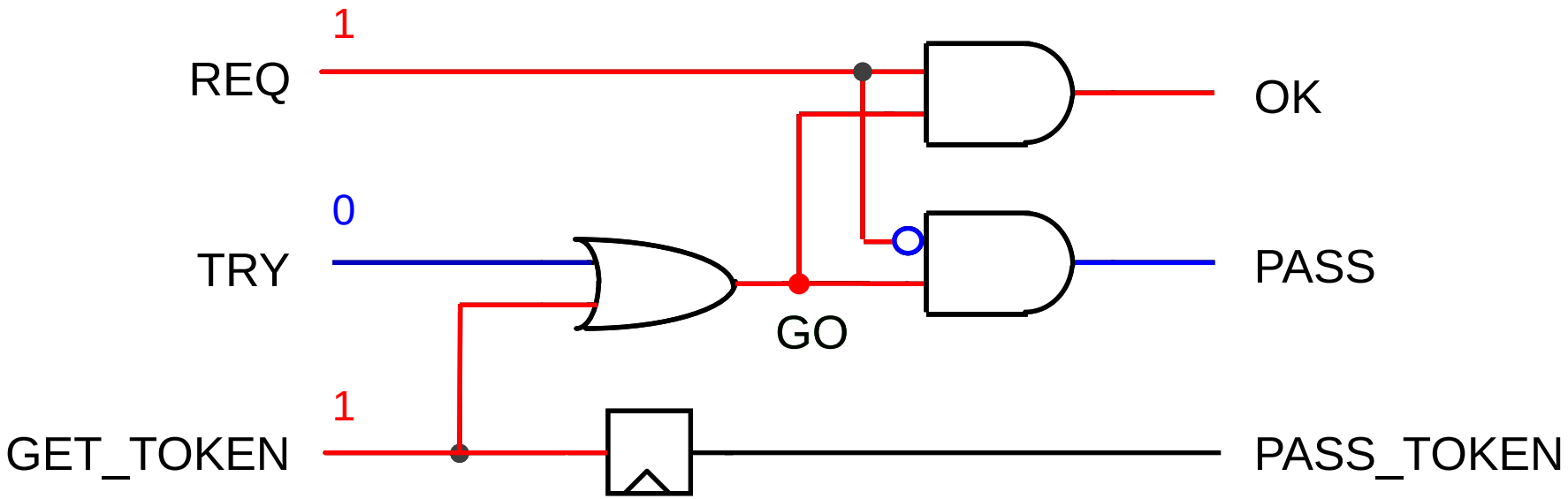
Logical time



Physical time

clock cycle OK = no overlap
(voltage / clock might be irregular)

Combinational Circuit = Proof Network



Each operator is a proof component

circuit = graph of all proofs of outputs from inputs

Constructive Boolean Propagation Logic

- Input vector $\mathcal{I}$ = inputs $\rightarrow \{0,1\}$
- Formulae: $\mathcal{I} \vdash e = b$

$$\frac{}{\mathcal{I} \vdash I = \mathcal{I}(I)} \qquad \frac{\mathcal{I} \vdash e = 0}{\mathcal{I} \vdash \text{not } e = 1} \qquad \frac{\mathcal{I} \vdash e = 1}{\mathcal{I} \vdash \text{not } e = 0}$$

$$\frac{\mathcal{I} \vdash e = 0}{\mathcal{I} \vdash e \text{ and } e' = 0} \qquad \frac{\mathcal{I} \vdash e' = 0}{\mathcal{I} \vdash e \text{ and } e' = 0} \qquad \frac{\mathcal{I} \vdash e = 1 \quad \mathcal{I} \vdash e' = 1}{\mathcal{I} \vdash e \text{ and } e' = 1}$$

$$\frac{\mathcal{I} \vdash e = 1}{\mathcal{I} \vdash e \text{ or } e' = 1} \qquad \frac{\mathcal{I} \vdash e' = 1}{\mathcal{I} \vdash e \text{ or } e' = 1} \qquad \frac{\mathcal{I} \vdash e = 0 \quad \mathcal{I} \vdash e' = 0}{\mathcal{I} \vdash e \text{ or } e' = 0}$$

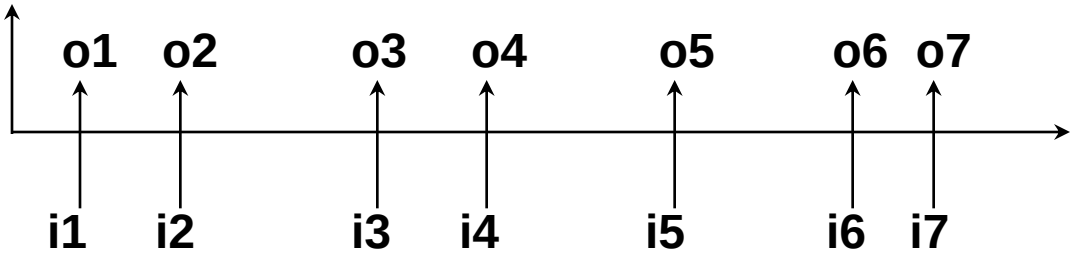
$$\frac{X = e \quad \mathcal{I} \vdash e = b}{\mathcal{I} \vdash X = b}$$

Constructive = No Excluded Middle !

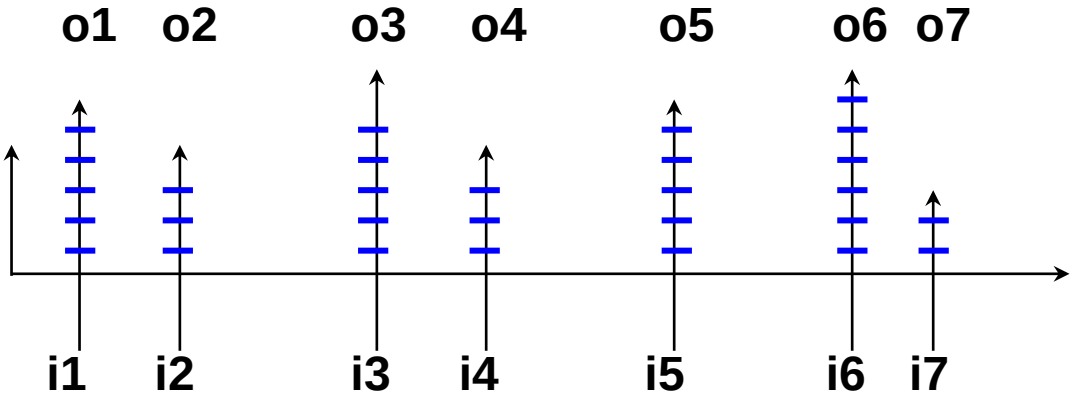
$\mathcal{I} \vdash e$ or not $e = 1$

iff $\mathcal{I} \vdash e = 0$ or $\mathcal{I} \vdash e = 1$

Logical Time vs. Constructive Time



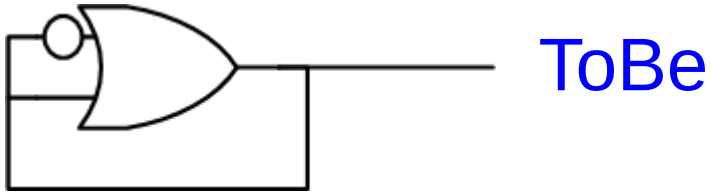
Logical time



Constructive time

Dubious Circuits

Hamlet : ToBe = ToBe or not ToBe



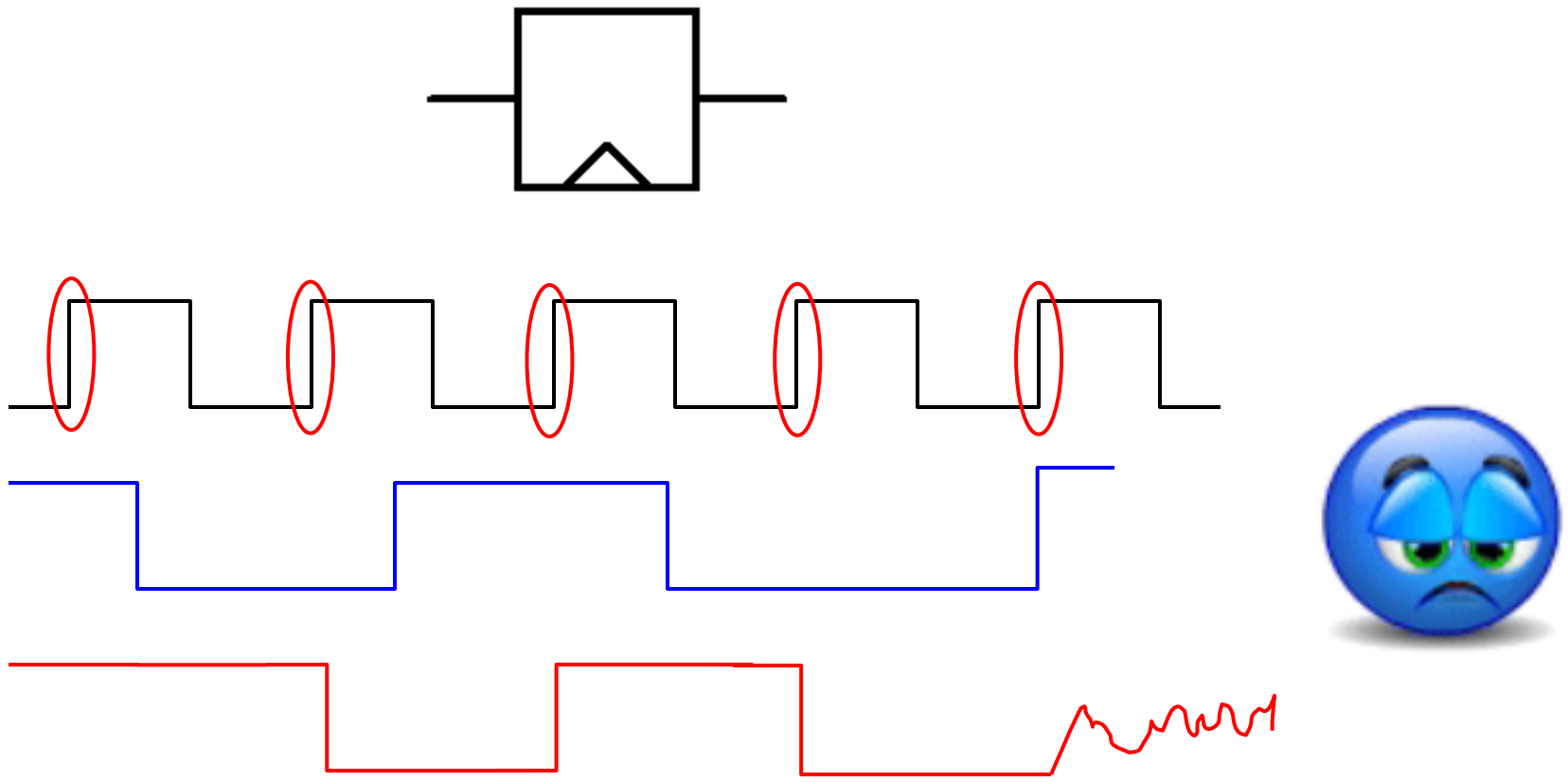
- Logically computes 1 in classical logic,
  but computes nothing in constructive logic
- Electrically stabilizes to 1 for some gate and wire delays, but not for all delays!

Theorem (Mendler-Shiple-Berry) :
constructive ⇔ electrically stable for all delays

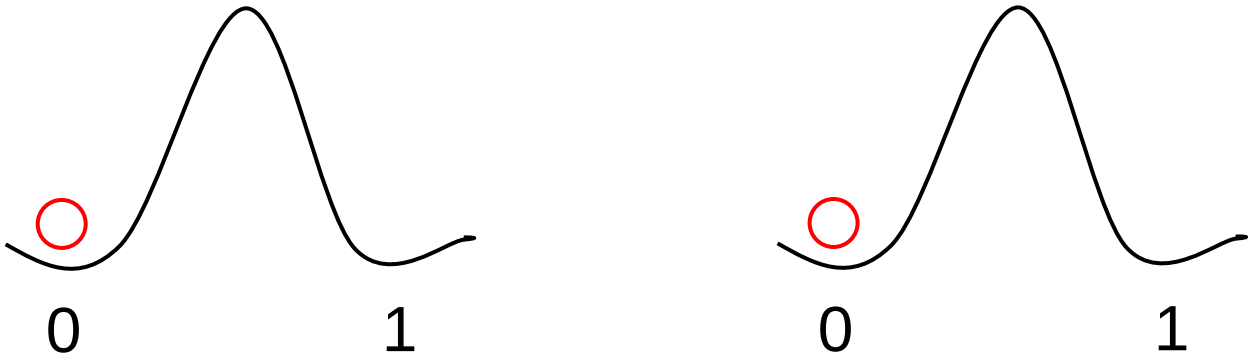
Agenda

1. Why discussing time and events
2. Lines and cones of time, causality
3. Multiform time
4. Concurrency modes
5. Synchronous circuits and constructive logic
6. Metastability and inter-clock zone protocols
7. Asynchronous and elastic circuits
8. Synchronous languages
9. Continuous vs. discrete time in modelers
10. Conclusion

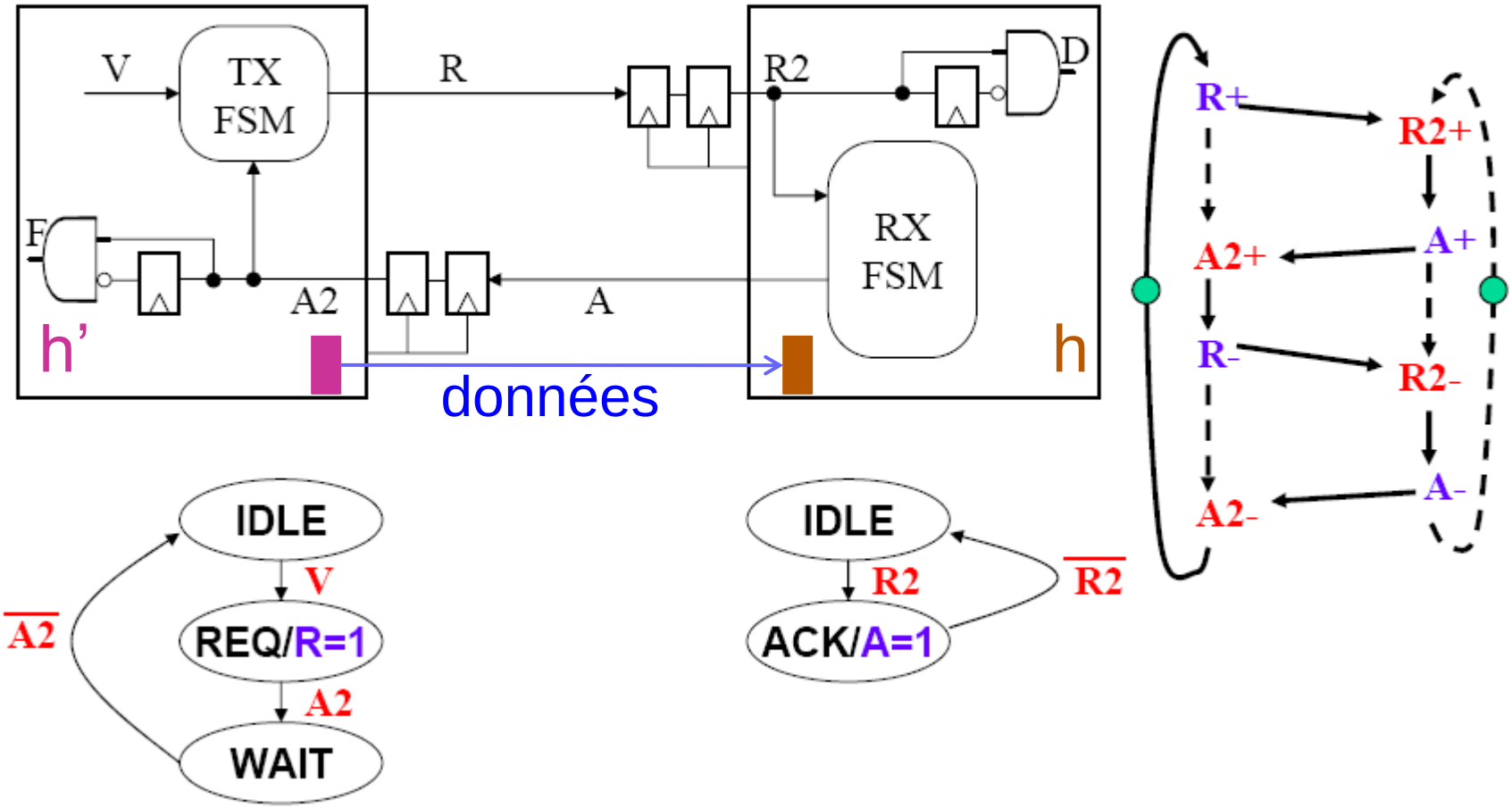
Metastability



The Ball on Hill Image



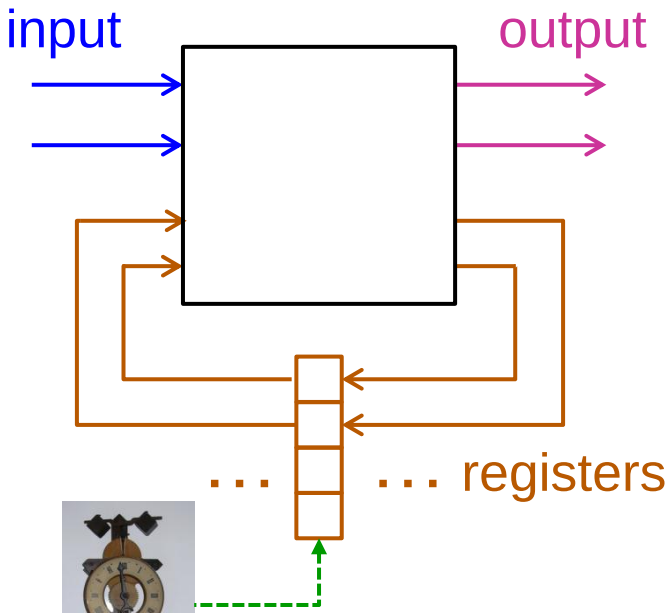
The Four Phase Synchronizer

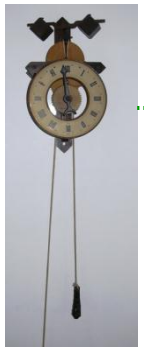


Agenda

1. Why discussing time and events
2. Lines and cones of time, causality
3. Multiform time
4. Concurrency modes
5. Synchronous circuits and constructive logic
6. Metastability and inter-clock zone protocols
7. Asynchronous and elastic circuits
8. Synchronous languages
9. Continuous vs. discrete time in modelers
10. Conclusion

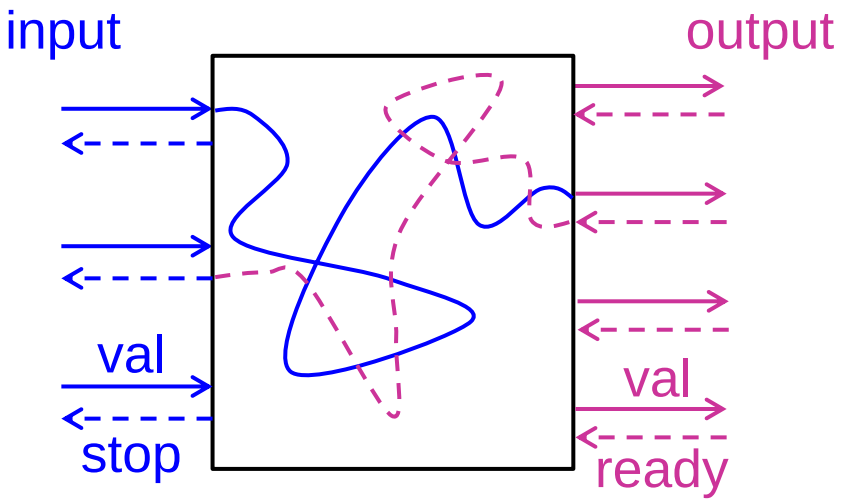
Synchronous vs. Asynchronous Circuits





synchronous
time-sensitive

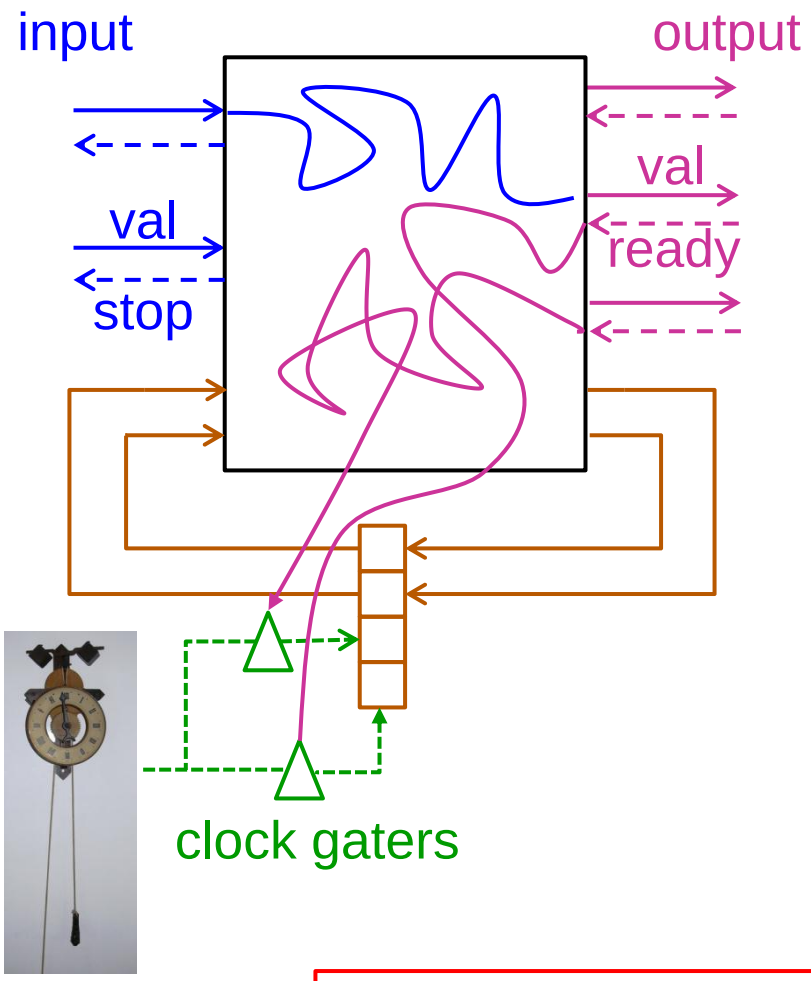
boolean logic
registers + clock
super duper CAD



asynchronous
time-insensitive

no clock, 2*wires
fancier logic
difficult CAD

Elastic Circuits



boolean logic + regs + clocks

asynchronous logic + clock gaters

synchronous CAD

time-insensitive

bubble-insensitive

⇒ cutting long lines OK

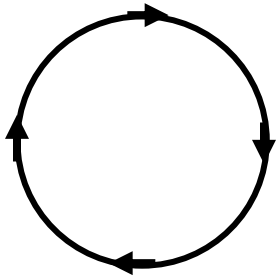
J. Cortadella, M. Kishinevsky et.al.

Agenda

1. Why discussing time and events
2. Lines and cones of time, causality
3. Multiform time
4. Concurrency modes
5. Synchronous circuits and constructive logic
6. Metastability and inter-clock zone protocols
7. Asynchronous and elastic circuits
8. Synchronous languages
9. Continuous vs. discrete time in modelers
10. Conclusion

Cycle-Based Software Synchrony

Cyclic execution



read inputs

compute reaction

generate outputs

Synchronous = Zero-delay = within the same cycle

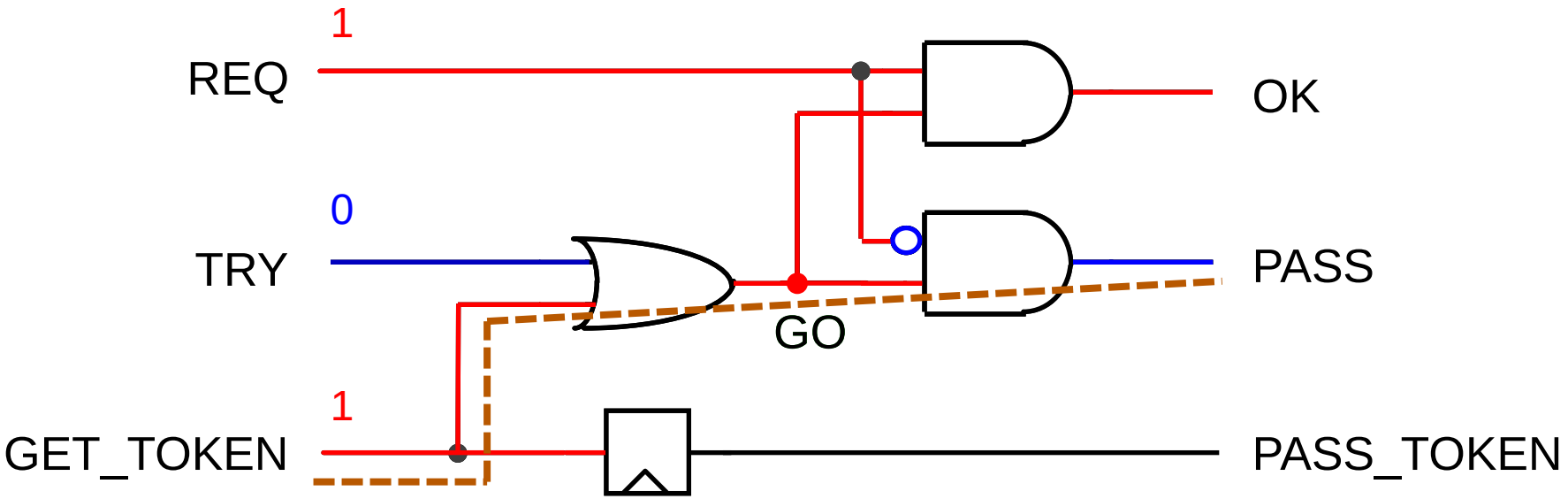
parallel propagation of control

parallel propagation of signals

Interference freedom => determinism by construction

Very elegant mathematics

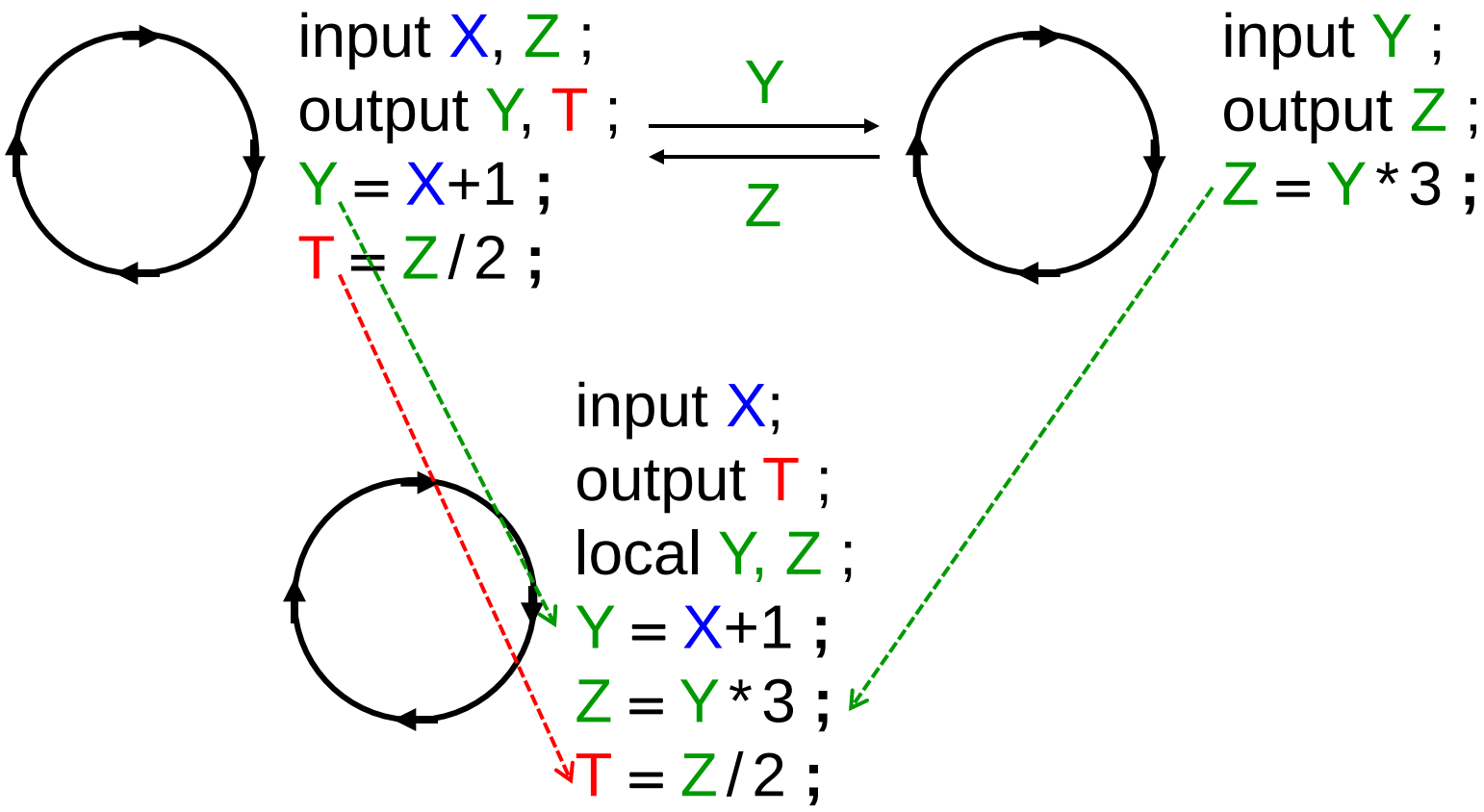
Synchronous Circuits: Simulation View



OK = REQ and GO ;
PASS = not REQ and GO ;
GO = TRY or GET_TOKEN ;
PASS_TOKEN = reg(GET_TOKEN) ;

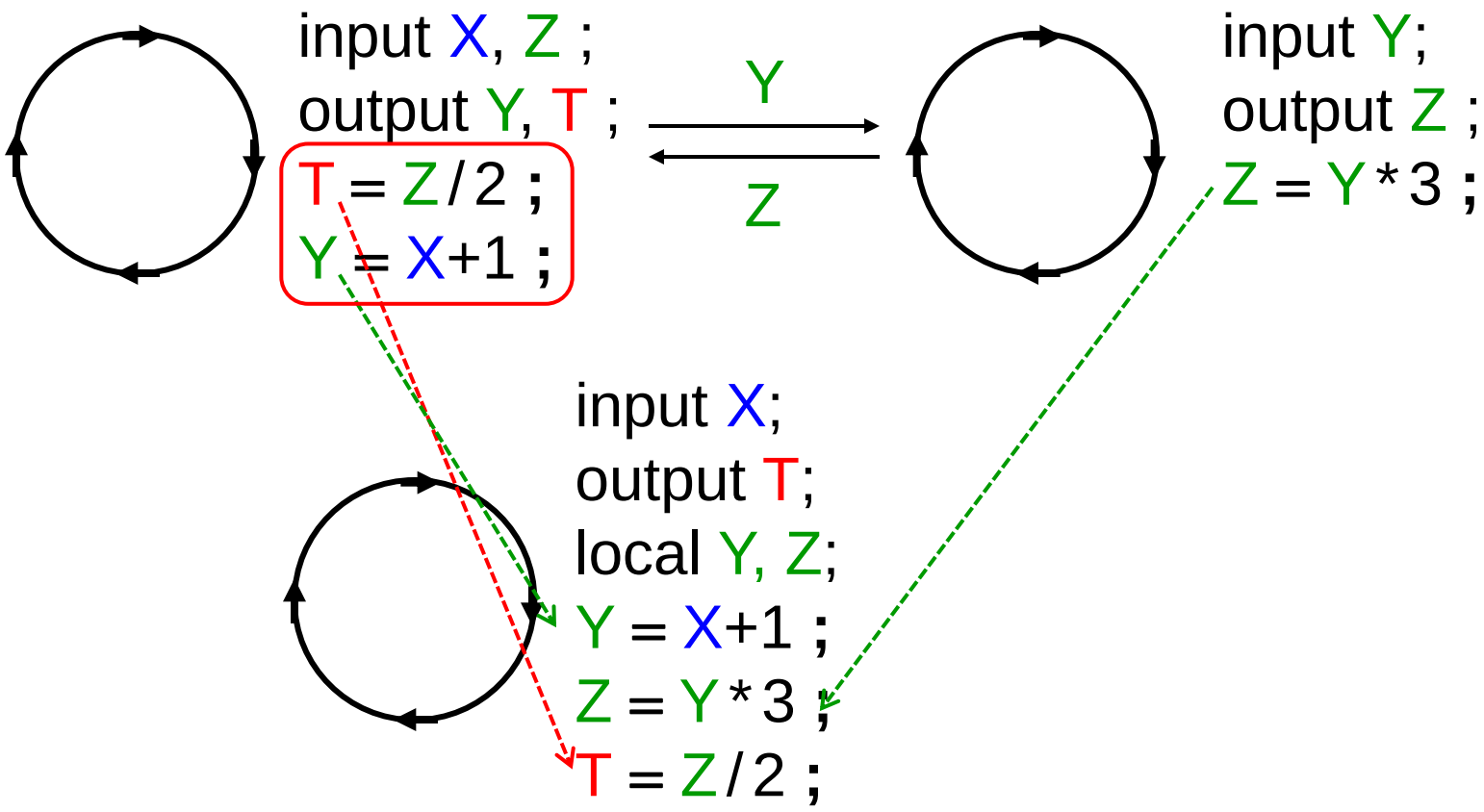
Sort equations to match electrical causal order with implementation-dependent sequential order

Resolving Concurrency by Cycle Fusion



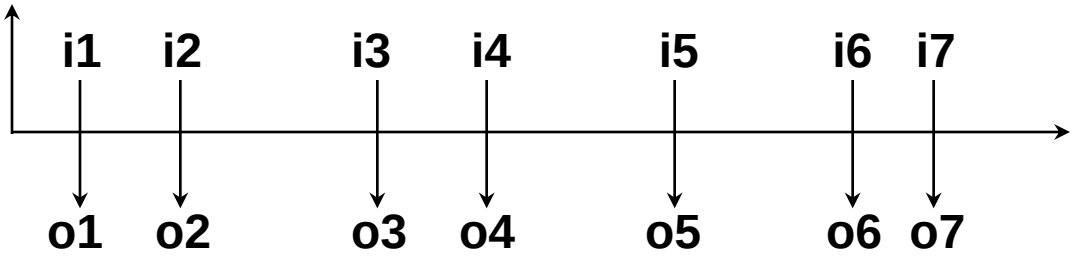
No synchronization, no deadlock, no context switch
Room size control = Worst Case Execution Time

Beware of causal vs. implementation order!

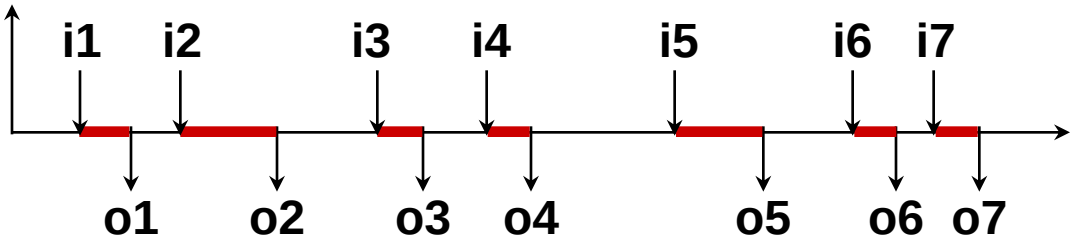


No block-level compositionality possible
Sequential order should be slave of causal order

Logical Time vs. Physical Time



Logical time



Physical time

WCET OK = no overlap

AbsInt: WCET by abstract interpretation

Lustre = Synchronous Data Flow

EventCounter

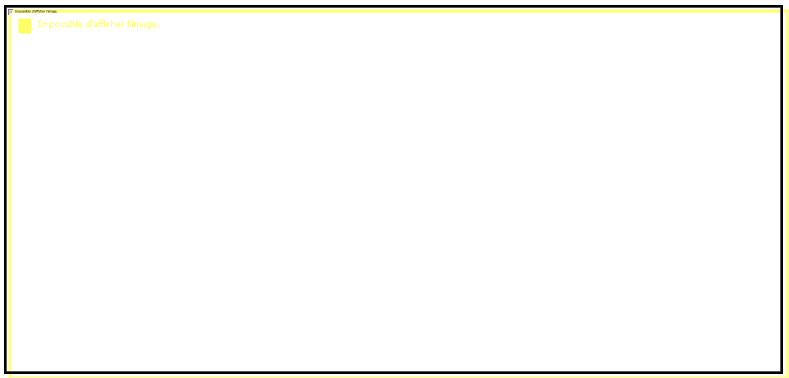
Event = false true false true true false false false true true false

Count = 0 1 1 2 3 3 3 3 4 5 5

$$\begin{cases} Count(0) = 0 \\ \forall t > 0,\ Count(t) = \begin{cases} Count(t-1)+1, & if\ \ Event(t) = true \\ Count(t-1), & otherwise \end{cases} \end{cases}$$

Count = 0 → (if Event
then pre(Count)+1
else pre(Count))
Event = false → not(pre(Event))

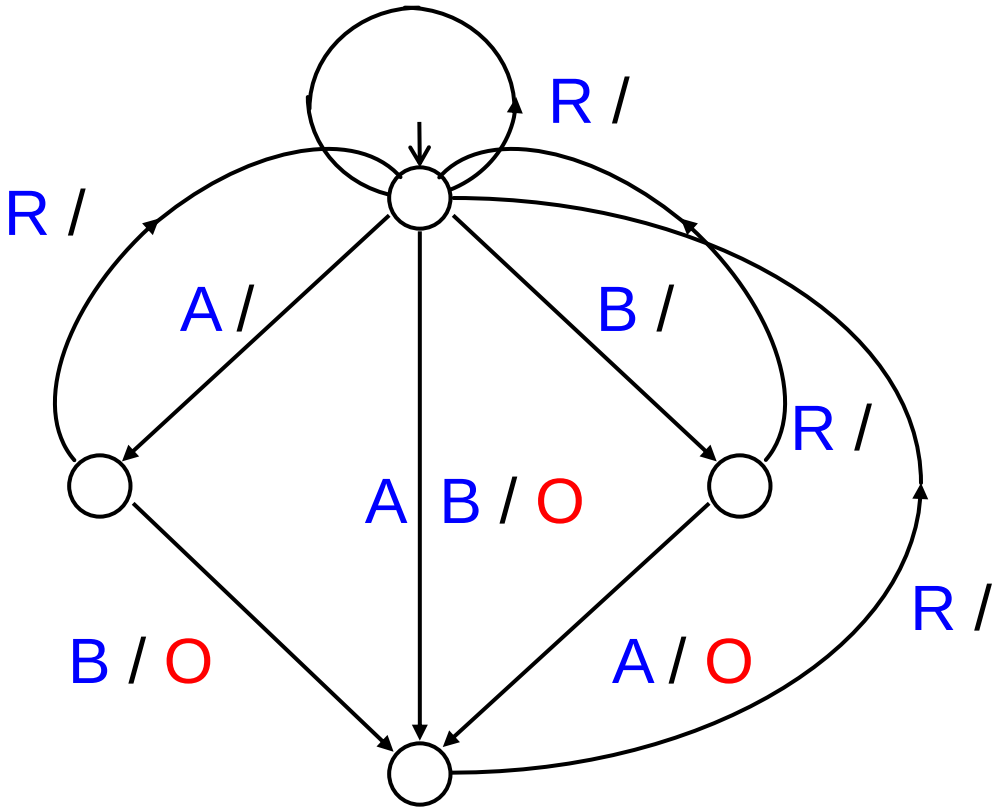
Textual Lustre



SCADE Block Diagram

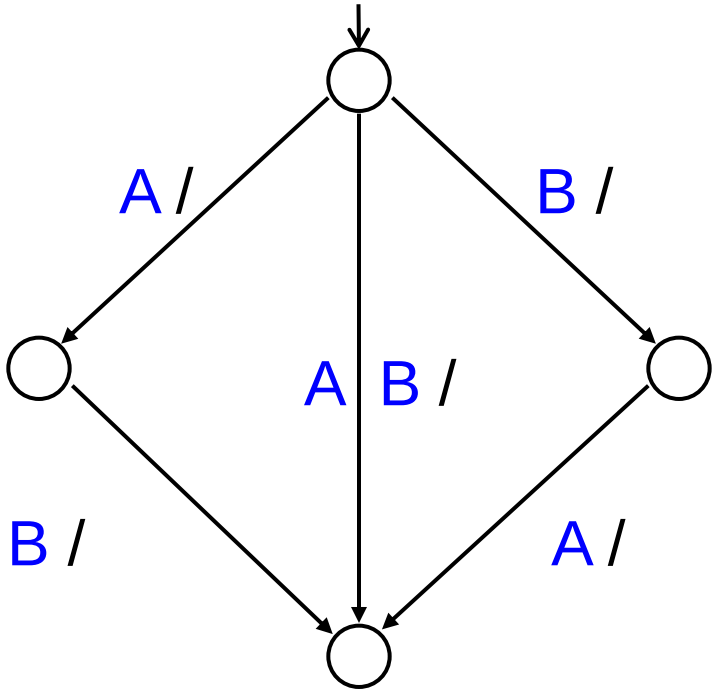
The ABRO Synchronization Example

Emit O as soon A and B have arrived
Reset this behavior each R



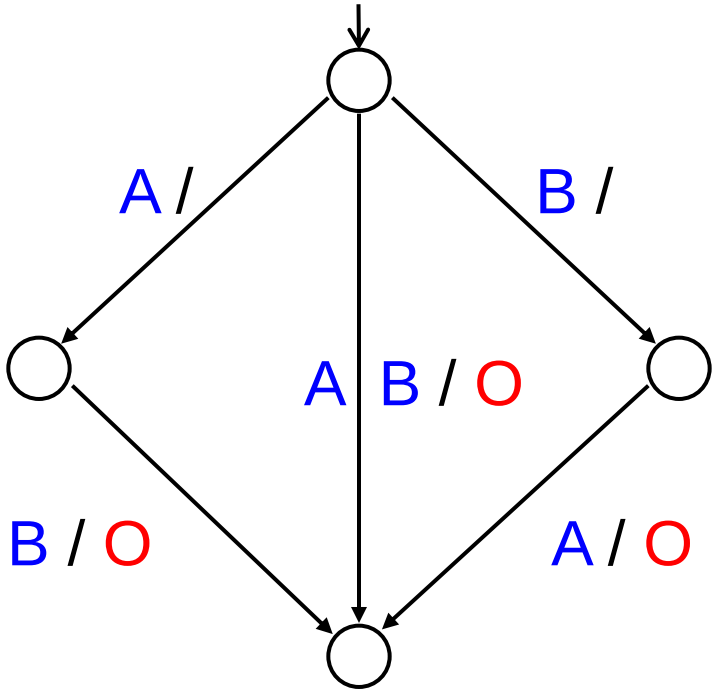
Memory Write
R : request
A : address
B : data
O : write

Esterel = Linear Specification



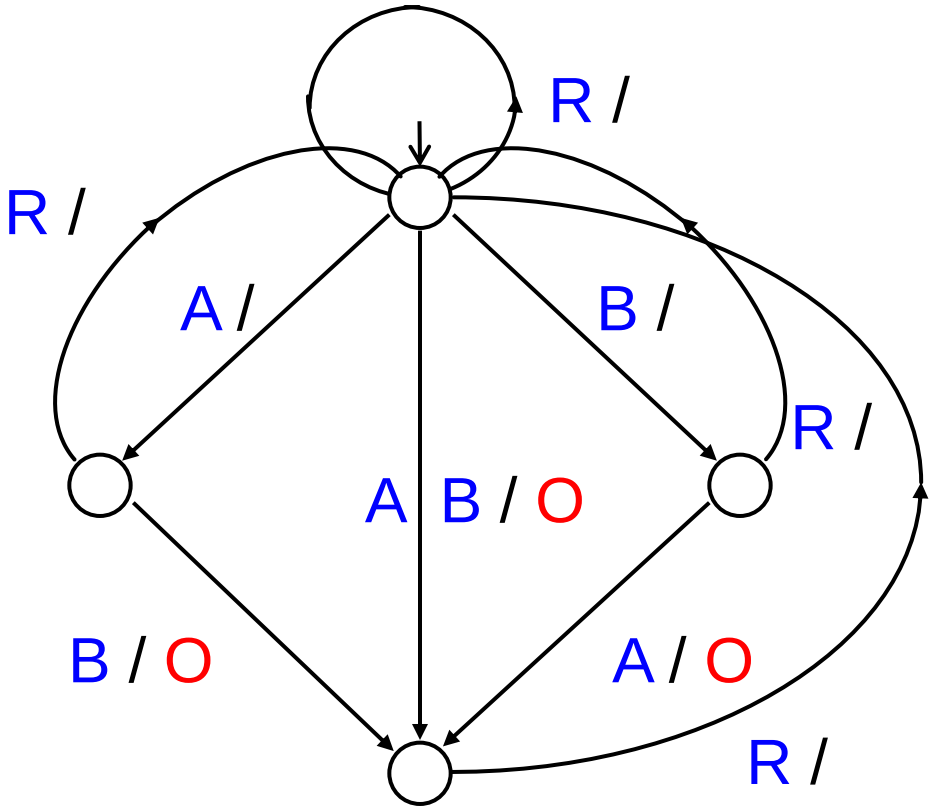
{ await A || await B };
halt

Esterel = Linear Specification



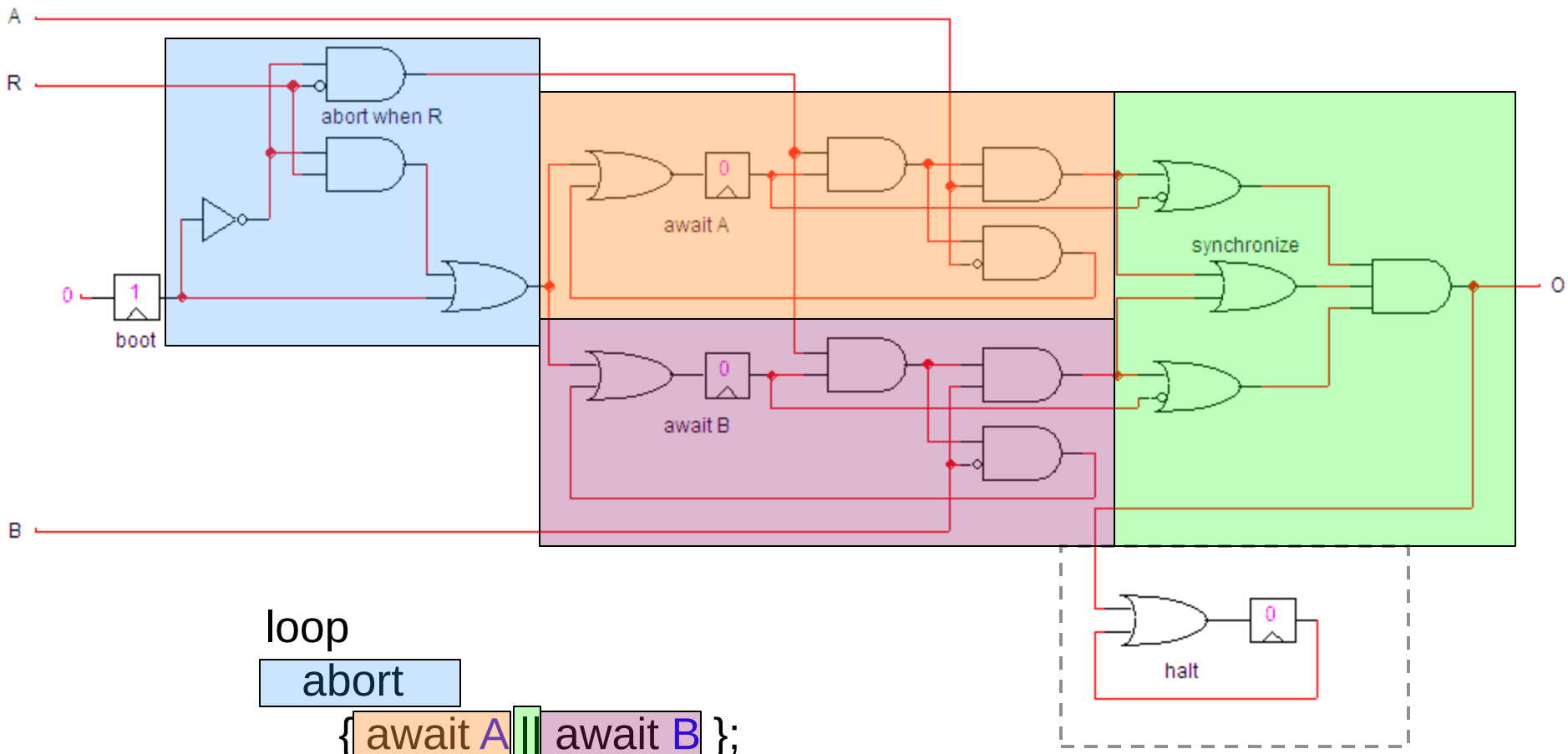

```
{ await A || await B };
emit O;
halt
```

Esterel = Linear Specification




```
loop
   abort
        { await A || await B };
        emit O;
        halt
   when R
end loop
```

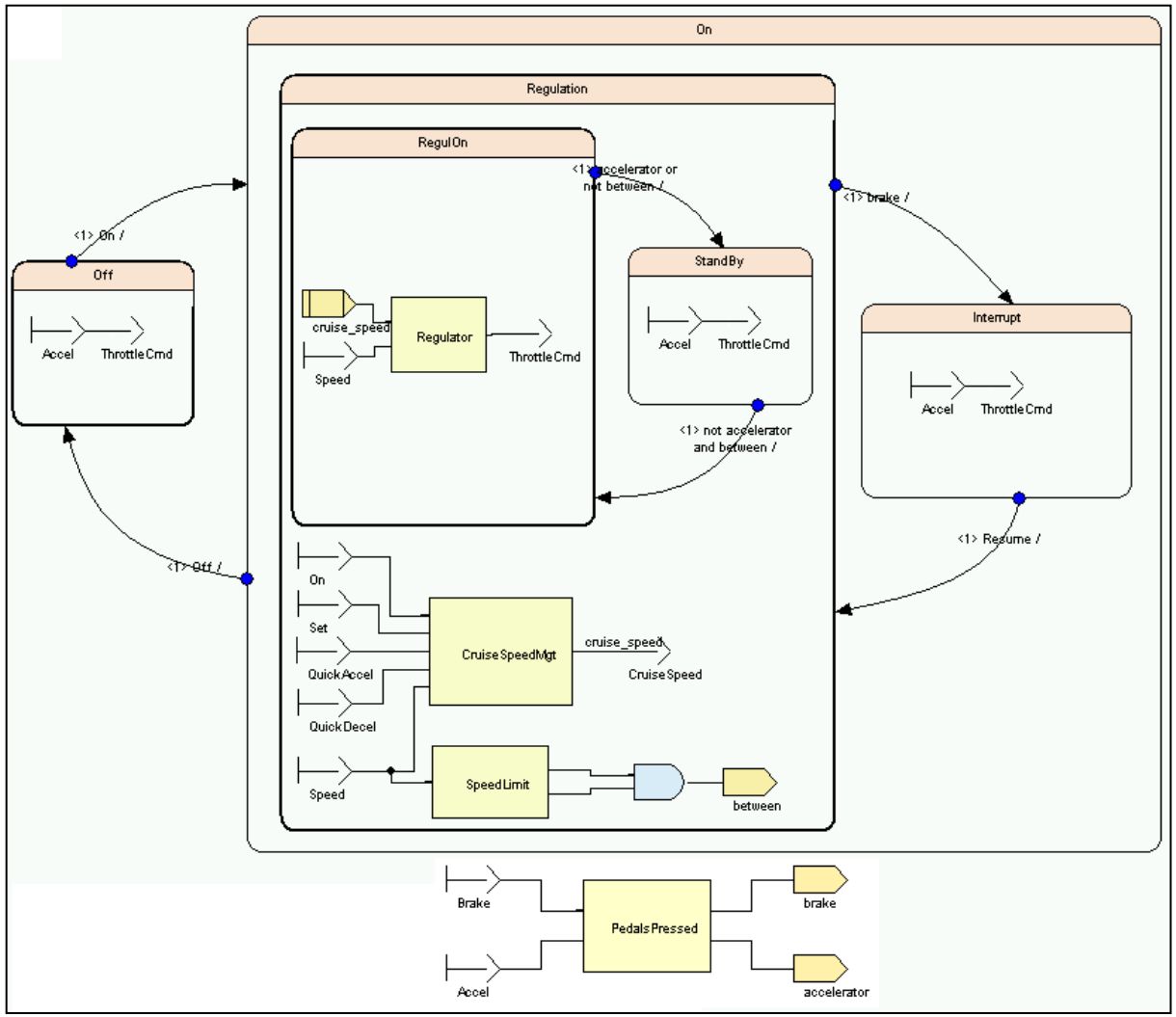
The ABRO Circuit (Proof Network)




```
loop
   abort
      { await A || await B };
      emit O ;
      halt
   when R
end loop
```

suppressed by optimization

Scade 6 for Certified Avionics = CC+FSM

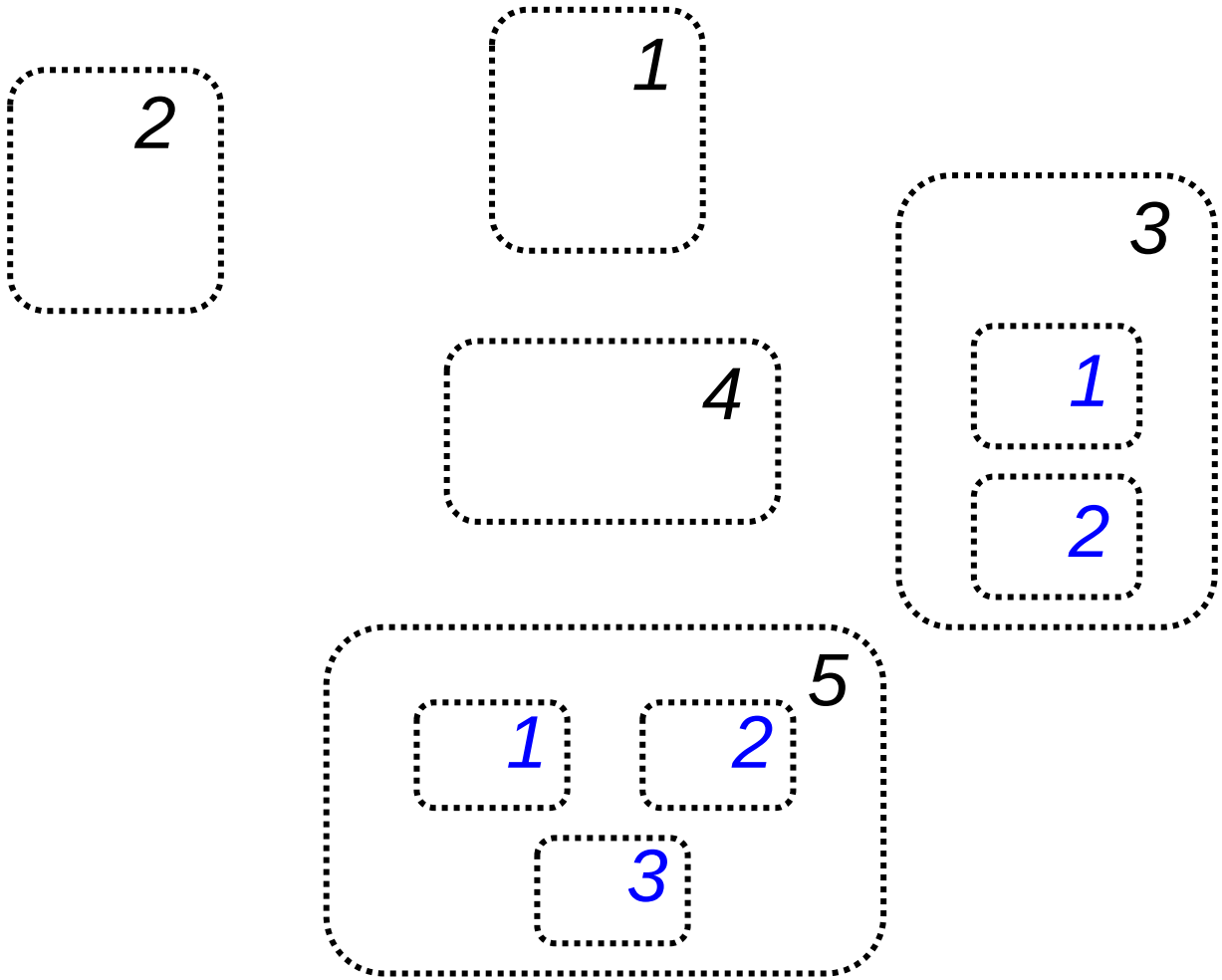


SCADE 6
Unified FSM & CC
Freely mixable in hierarchy

Functional language,
Functional array support
Formal semantics
Certified compiler

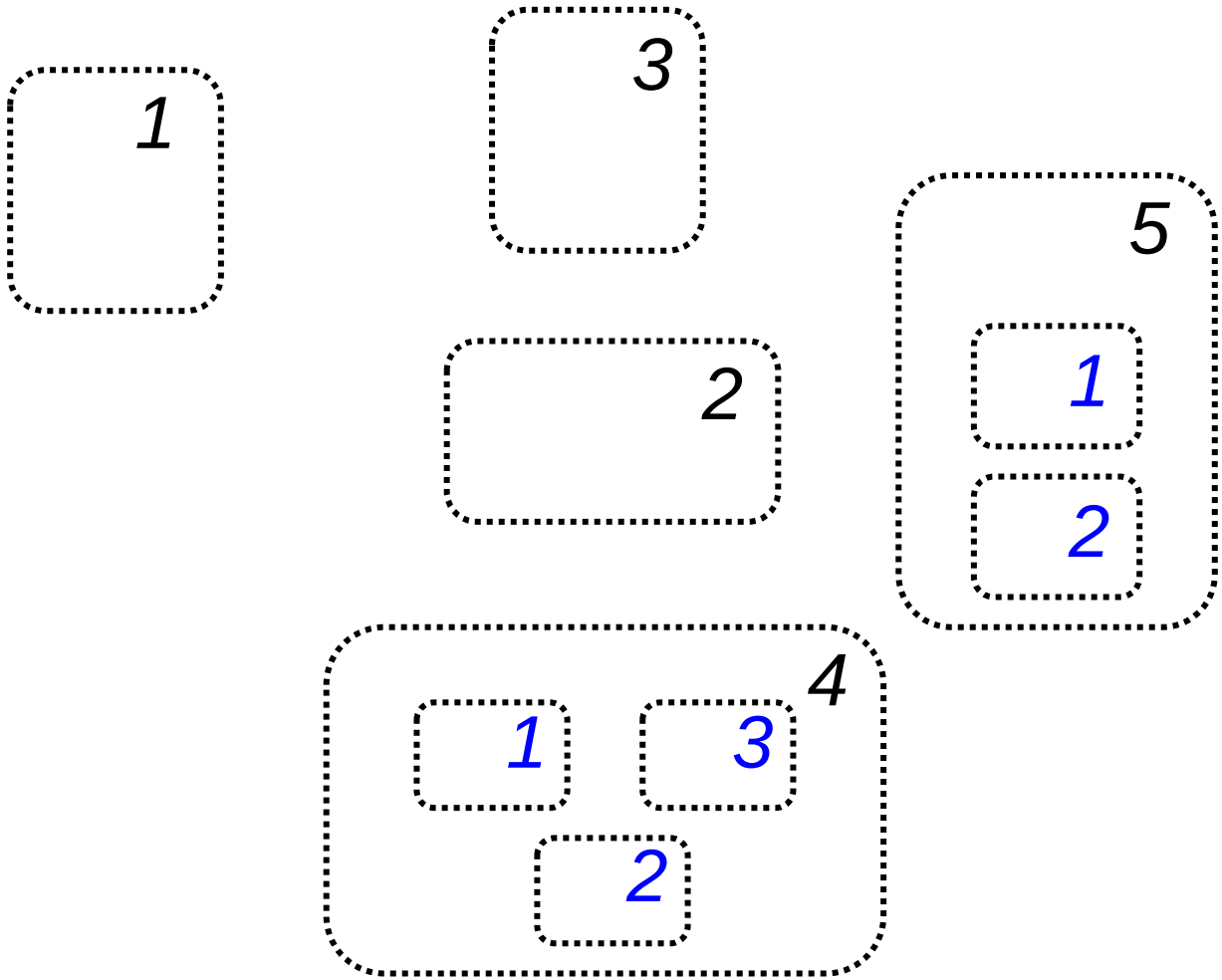
See also Ptolemy II, Ed Lee, UC Berkeley

Bad: Scheduling by Physical Position !

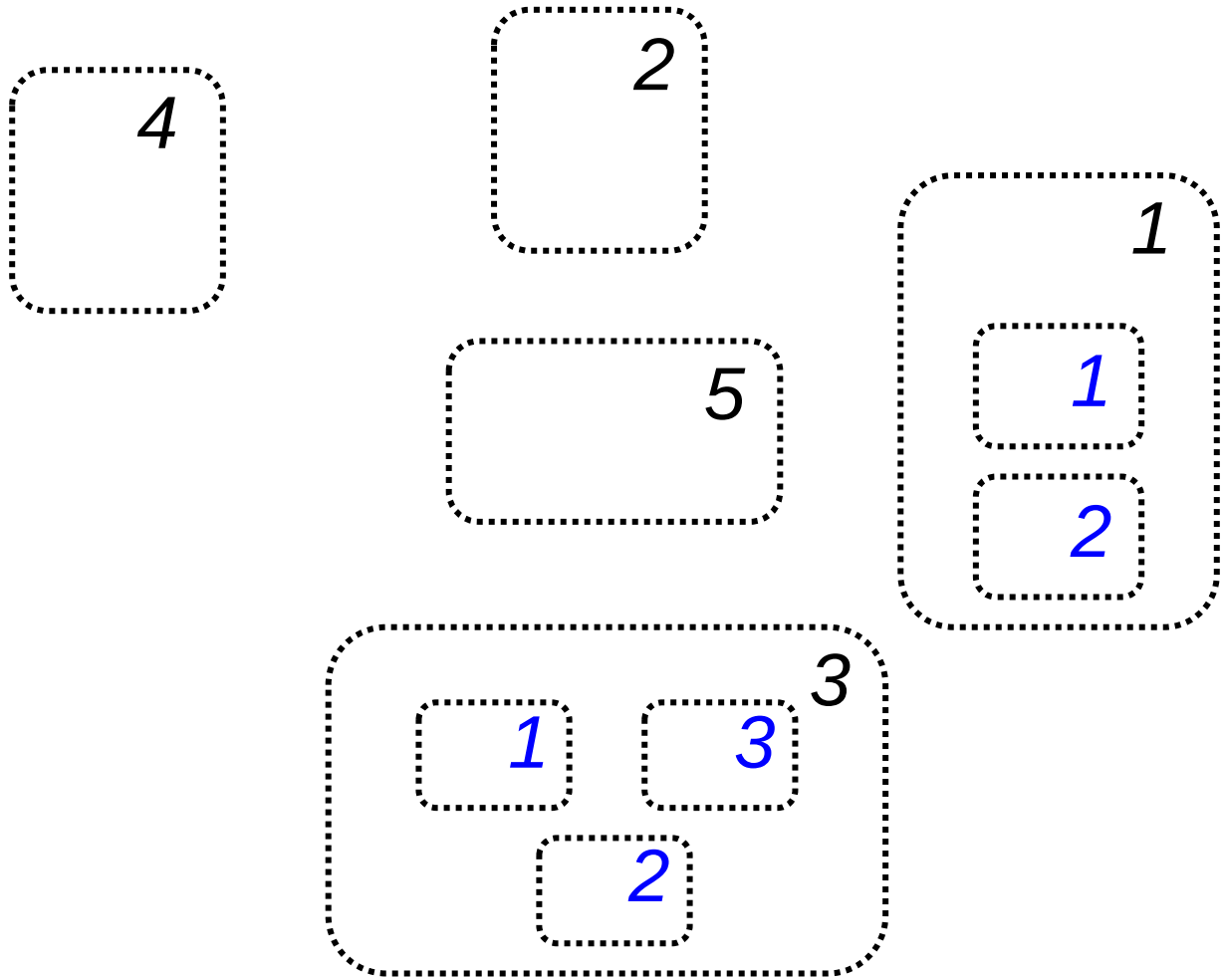


Initial choice : block-level parallelism

Bad : Scheduling by Creation Order !!



Complicated: Manual Rescheduling !!!



Plus lots of rules to renumber everything,
locally change priorities, cut & paste states, etc.

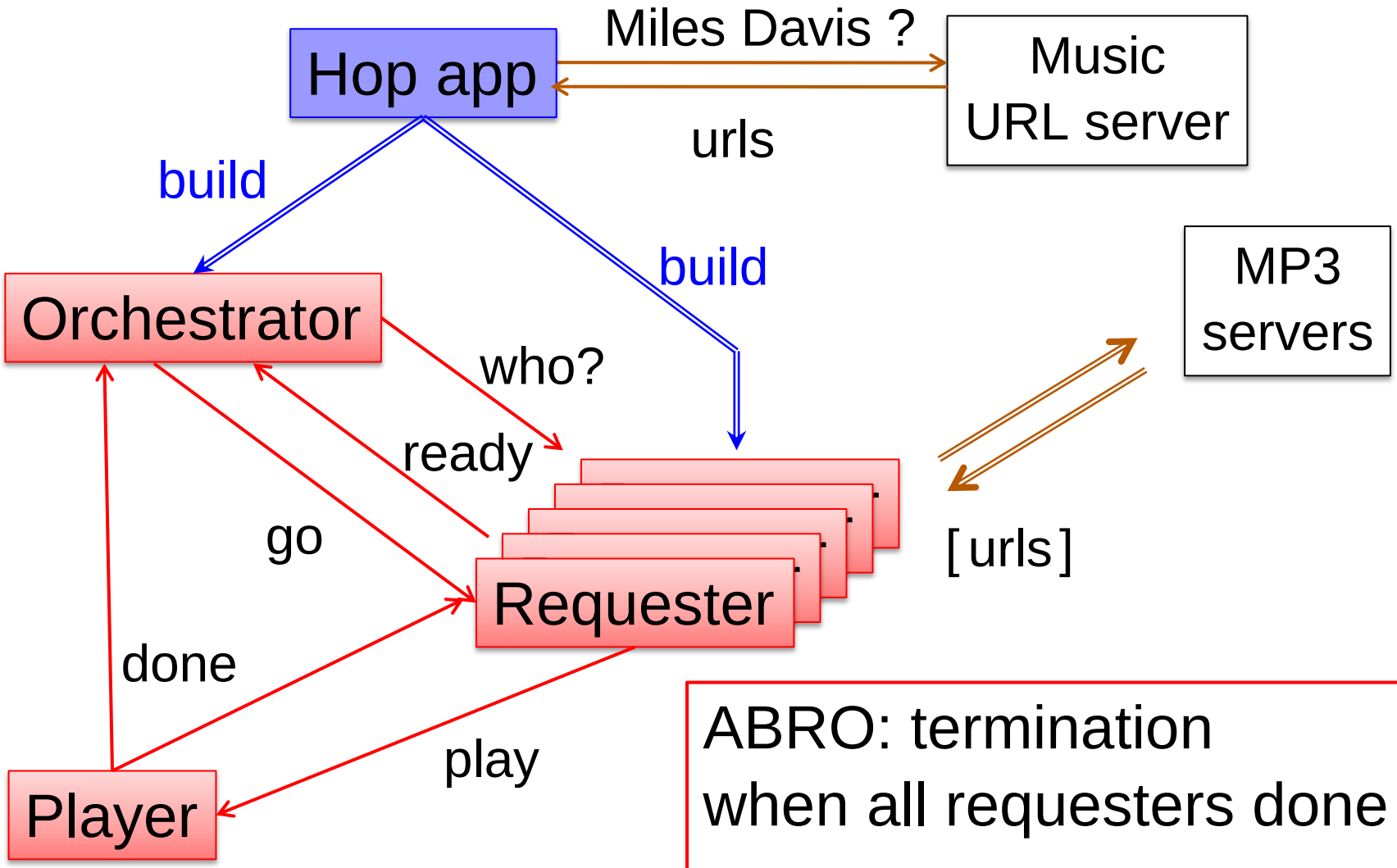
- Pollution: block-level sequential code generation makes causal dependencies depend on implementation ones

Why are parallel priorities needed at all?
Parallelism should be
associative, commutative, and modular!

Semantics and compiler handle parallelism, not users!

Cf. Lustre, Esterel, SyncCharts, SCADE 6, Ptolemy II,...

Hop + HipHop : A Web Dynamic Esterel



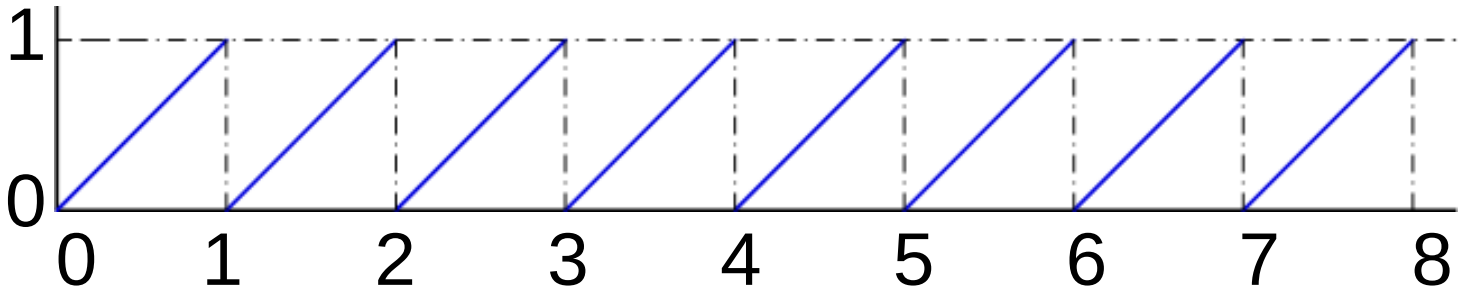
ABRO: termination
when all requesters done

Handling all sorts of errors

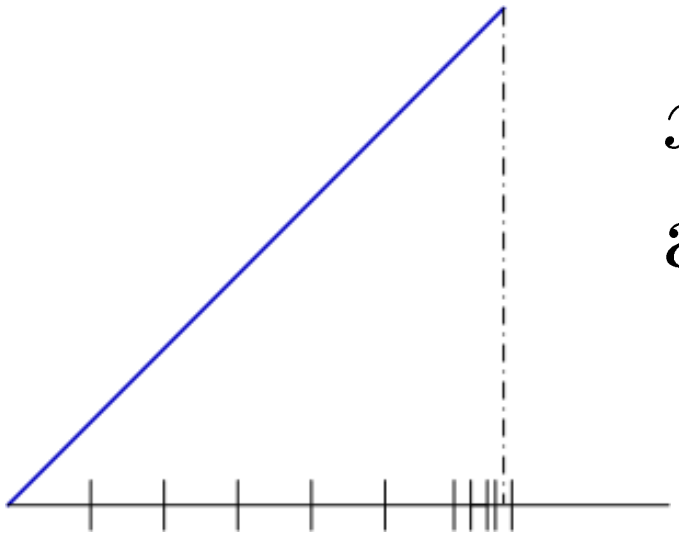
Agenda

1. Why discussing time and events
2. Lines and cones of time, causality
3. Multiform time
4. Concurrency modes
5. Synchronous circuits and constructive logic
6. Metastability and inter-clock zone protocols
7. Asynchronous and elastic circuits
8. Synchronous languages
9. Continuous vs. discrete time in modelers
10. Conclusion

Hybrid Modeling : ODE + mode transitions

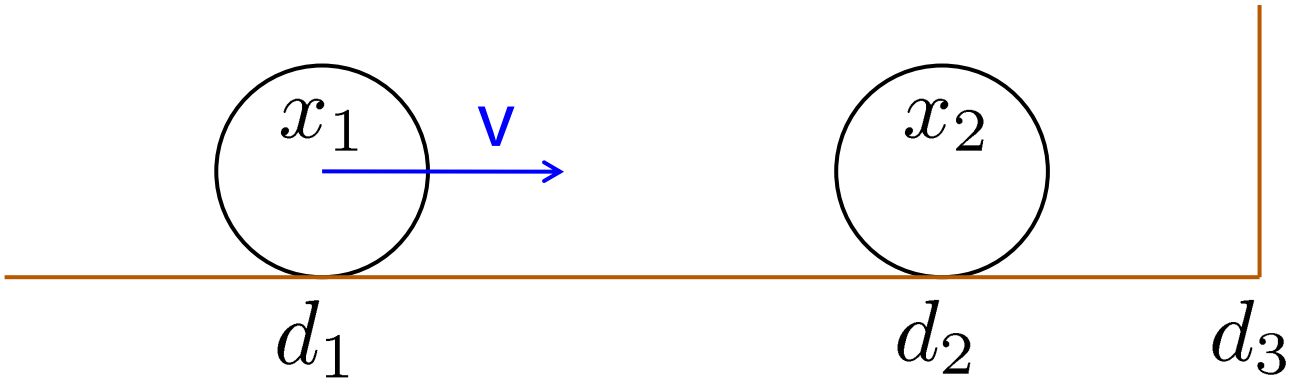


$$\begin{cases} x \ : \ \textbf{init } 0 \textbf{ reset every } [x \textbf{ up } 1 \Rightarrow 0] \\ \dot{x} = 1 \end{cases}$$



$x > 1$ detected
at time $1 + \epsilon$

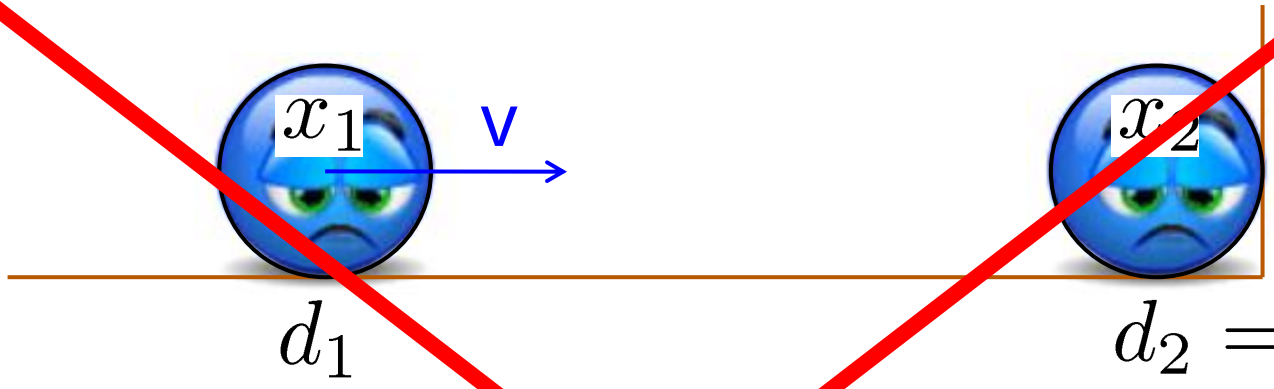
Balls on Wall



Current Modelers OK

$$
\begin{cases}
x_1 \ : \ \textbf{init}\ d_1 \\
\dot{x_1} = v_1 \\
x_2 \ : \ \textbf{init}\ d_2 \\
\dot{x_2} = v_2 \\
v_1 \ : \ \textbf{init}\ v\ \textbf{every}\ [x_1\ \textbf{up}\ x_2 \Rightarrow last(v_2)] \\
v_2 \ : \ \textbf{init}\ 0\ \textbf{every}\ [x_1\ \textbf{up}\ x_2 \Rightarrow last(v_1),\ x_2\ \textbf{up}\ d_3 \Rightarrow -last(v_2)] \\
\dot{v_1} = \dot{v_2} = 0
\end{cases}
$$

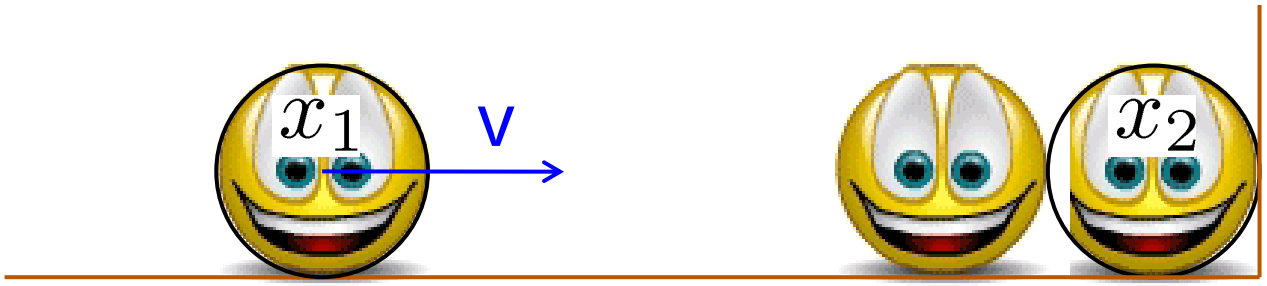
Ball Stuck on Wall – Current modelers



for some execution modes

$$
\begin{cases}
x_1 \ : \ \textbf{init}\ d_1 \\
\dot{x_1} = v_1 \\
x_2 \ : \ \textbf{init}\ d_2 \\
\dot{x_2} = v_2 \\
v_1 \ : \ \textbf{init}\ v\ \textbf{every}\ [x_1\ \textbf{up}\ x_2 \Rightarrow last(v_2)] \\
v_2 \ : \ \textbf{init}\ 0\ \textbf{every}\ [x_1\ \textbf{up}\ x_2 \Rightarrow last(v_1),\ x_2\ \textbf{up}\ d_3 \Rightarrow -last(v_2)] \\
\dot{v_1} = \dot{v_2} = 0
\end{cases}
$$

Non-Standard Constructive Analysis



ε infinitesimal

$$
\begin{array}{l}
d_1 \ \textbf{up}\ d_2 \Rightarrow v_1 = 0,\ v_2 = v \\
\rightarrow x_2 \ \textbf{up}\ d_3 \\
\Rightarrow\ v_2 = -v \\
\rightarrow x_1 \ \textbf{up}\ x_2 \\
\Rightarrow v_1 = -v,\ v_2 = 0
\end{array}
$$

$$
\left\{
\begin{array}{l}
x_1 \ : \ \textbf{init}\ d_1 \\
\dot{x_1} = v_1 \\
x_2 \ : \ \textbf{init}\ d_2 \\
\dot{x_2} = v_2 \\
v_1 \ : \ \textbf{init}\ v\ \textbf{every}\ [x_1\ \textbf{up}\ x_2 \Rightarrow last(v_2)] \\
v_2 \ : \ \textbf{init}\ 0\ \textbf{every}\ [x_1\ \textbf{up}\ x_2 \Rightarrow last(v_1),\ x_2\ \textbf{up}\ d_3 \Rightarrow -last(v_2)] \\
\dot{v_1} = \dot{v_2} = 0
\end{array}
\right.
$$

Conclusion

- There are two opposite ways to build systems
  - hack them, test them, and hope they work
  - think about them, and use well-defined tools
- Synchronous languages form a very strong basis
  - with constructive semantics as a basic principle
  - with extensive industrial development
- Their constructive semantics principle extend well
  - to GALS systems (known)
  - to continuous / discrete time modeling - new
  - to web-based control – new
  - to sound synthesis and music composition - new