Synchronous Programming

Gérard Berry

http://www.college-de-france.fr/site/en-gerard-berry/

Collège de France
Chair Algorithms, Machines, and Languages

Académie des sciences
Académie des technologies
Academia Europaea

Modelica, Prague, 17/05/2017

COLLÈGE
DE FRANCE
1530

Birth and Goals

- Invented in the early 1980's by mixed Computer Science / Control Theory French groups (Esterel in Sophia-Antipolis, Lustre / SCADE in Grenoble, Signal in Rennes),
- Goal : simplify real-time programming and make it much more rigorous and verifiable by mathematical semantics
- 1990's : process control, avionics, telecom, robotics, HW circuits, etc. → industrial
- 2001: Esterel Technologies, Esterel v7 for HW circuits, SCADE bought from Telelogic
- 2007: SCADE 6 = Lustre + Esterel for critical systems

230+ customers in 27 countries, now part of ANSYS

Esterel and Lustre

- Esterel : 1983, Ecole des Mines Sophia-Antipolis
  - dedicated to control-intensive real-time systems
  - rapidly used in telecom (Bell Labs), avionics (Dassault),…
  - several generations of mathematical semantics and academic and industrial compilers to HW and SW, all compatible with each other

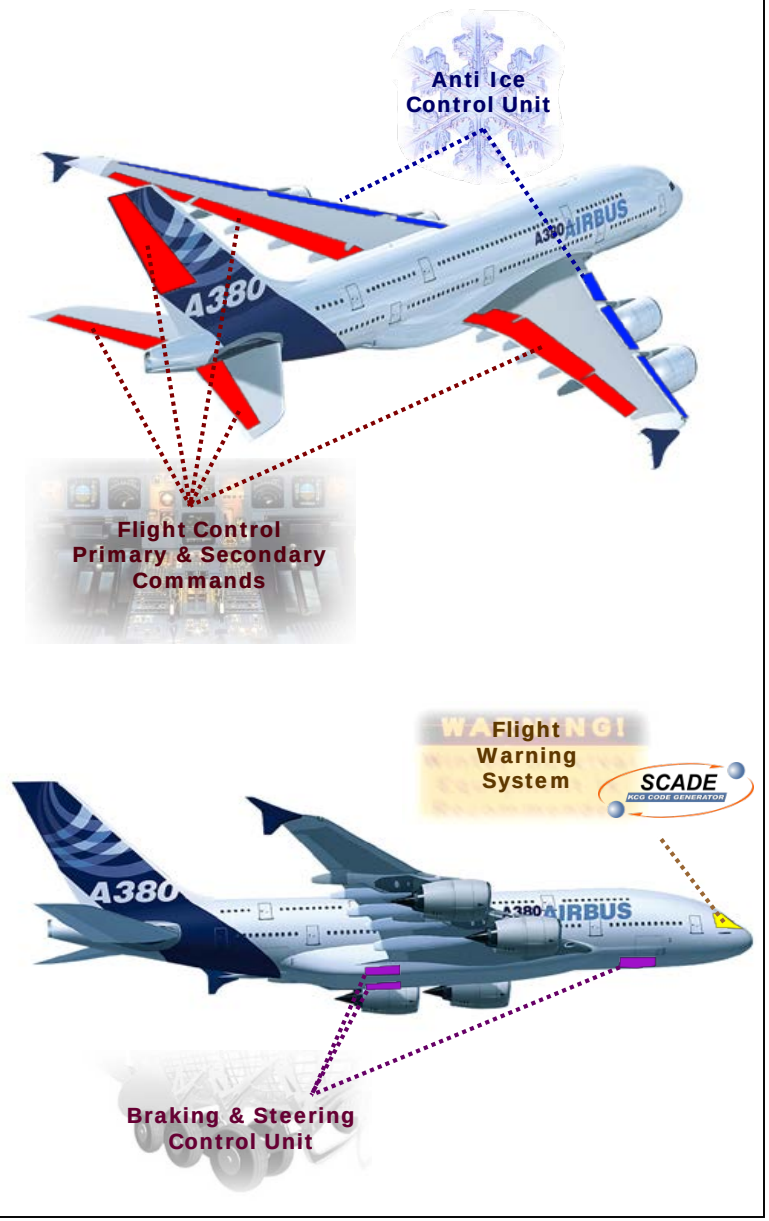
- Lustre : 1985, IMAG Grenoble (Caspi-Halbwachs)
  - dedicated to data-flow based real-time systems
  - SCADE = graphical Lustre
  - rapidly used in nuclear (Merlin-Gerin), then avionics (Airbus as a replacement of own SAO language)
  - very simple mathematical semantics, quite simple code generation

Rafale = Esterel, Airbus = SCADE

Solution 1 : ~~aerial fight~~

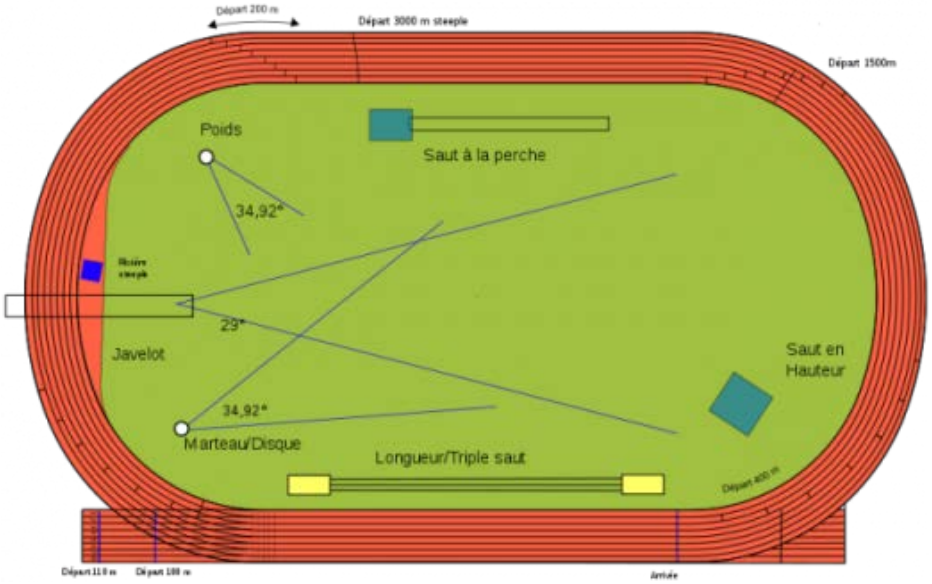
Solution 2 : unifying the langages
→ Esterel v7, SCADE 6

SCADE 5 in the Airbus A380



- Flight Control system
- Flight Warning system
- Electrical Load Management system
- Anti Icing system
- Braking and Steering system
- Cockpit Display system
- Part of ATSU (Board / Ground communication)
- FADEC (Engine Control)
- EIS2 : Specification GUI Cockpit:
  - PFD : Primary Flight Display
  - ND : Navigation Display
  - EWD : Engine Warning Display
  - SD : System Display

Temporal Event 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 do
   run RushToHospital
end trap
```

preemption

concurrency

submodule call

sequence

exception

exit HeartAttack

Code of <CheckHeart>

```
loop
    await 3 Second ;
    exit HeartAttack
each HeartBeat
```

Each HeartBeat cancels a delay 3 Second

Lustre = Synchronous Data Flow

Event counter:

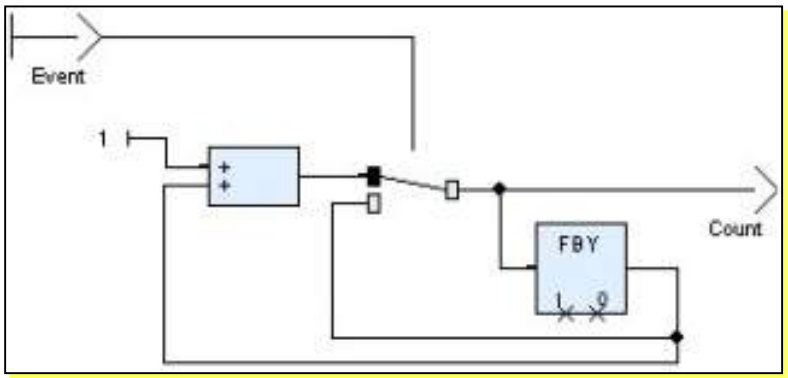
Event = false true false true true false false false true true …

Count = 0 1 1 2 3 3 3 3 4 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))
```

Lustre



SCADE

Lustre = Hierarchical Functional Language

```
node Counter (Event, Reset : bool) returns Count : int ;
let
    Count = if (true → Reset) then 0
            else if Event then pre(Count)+1
            else pre(Count))
tel
```

```
Minute = Counter (Second, pre(Minute) = 59) ;
Hour = Counter (Minute, pre(Hour) = 23) ;
```

Clocks and the Clock Calculus

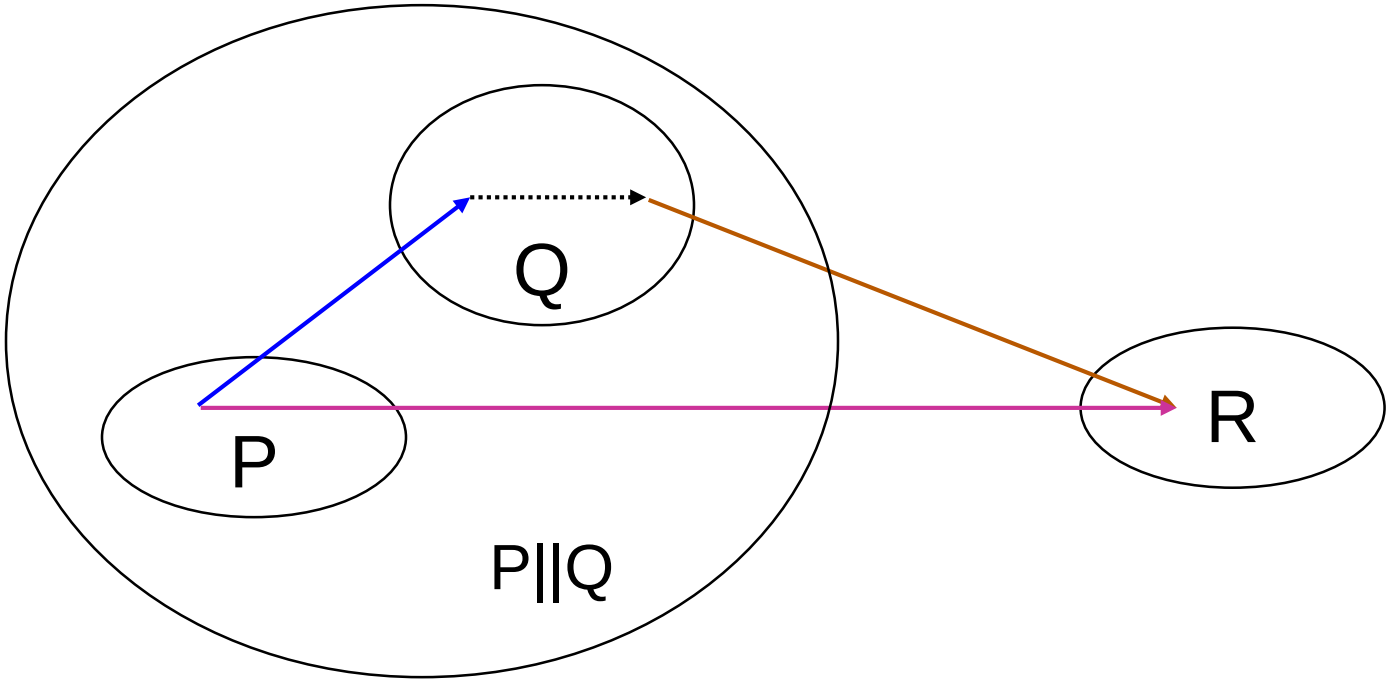
C	true	false	true	false	false	true
X	x1	x2	x3	x4	x5	x6
X when C	x1		x3			x6
pre(X when C)	nil		x1			x3
current(X when C)	x1	x1	x3	x3	x3	x6

Operations on undersampled flows

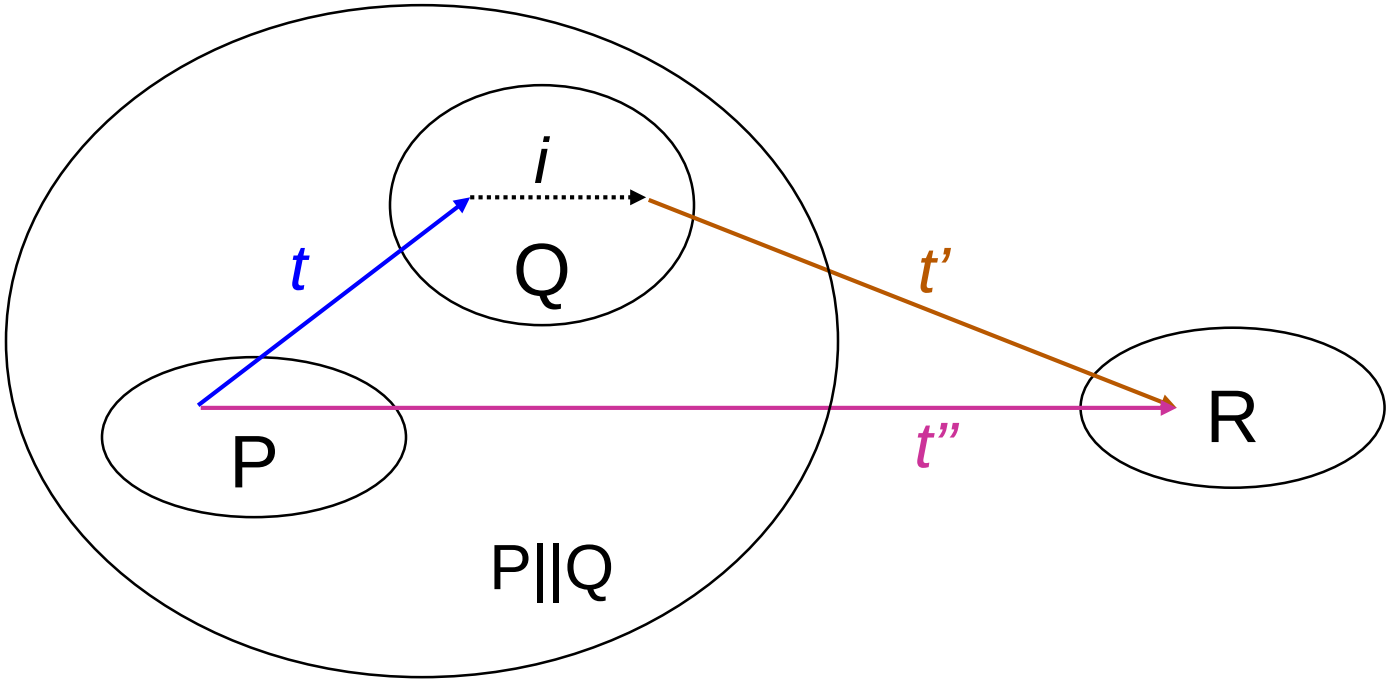
C	true	false	true	false	false	true
X	x1	x2	x3	x4	x5	y6
Y	y1	y2	y3	y4	y5	y6
Z = X when C	x1		x3			x6
T = Y when C	y1		y3			y6
Z + T	x1 + y1		x3 + y3			x6 + y6

Clock calculus : ~~X = T~~

Concurrency: the Compositionality Principle



Concurrency: the Compositionality Principle



$$t'' = t + i + t'$$

$$t'' \approx t \approx i \approx t'$$

$$\Rightarrow t \approx t + t$$

Exactly 3 solutions (in Physics and Informatics)

1. *t* arbitrary → asynchrony
2. *t* = 0 → synchrony
3. *t* predictable → vibration

asynchrony ⇒ non-determinism

synchrony ⇒ determinism

predictable ⇒ quasi-determinism

Synchrony vs. Vibration: Thick Instants

Synchrony

Musicians and spectators neglect the speed of sound

Vibration

Acousticians deal with sound propagation

Synchrony vs. Vibration: Thick Instants

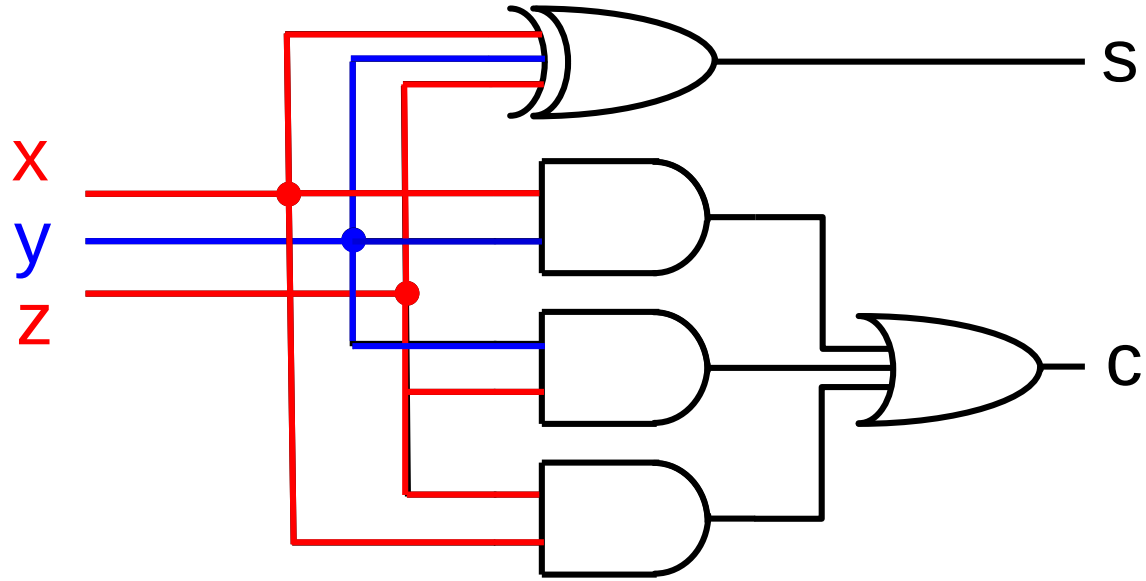
If the room is small enough

vibration implements synchrony for spectators

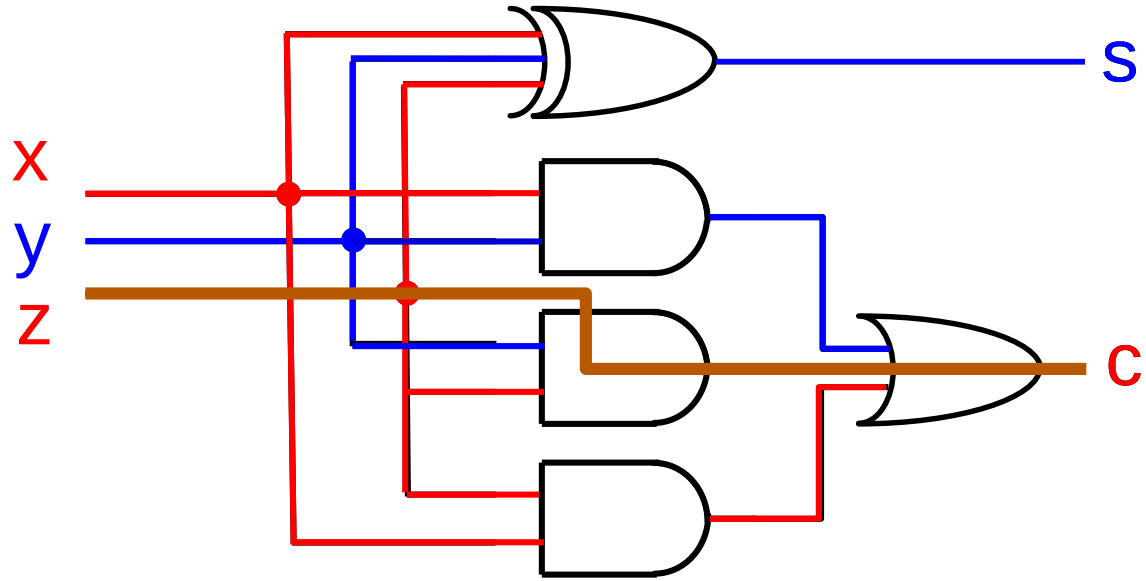
However, if the orchestra is big enough

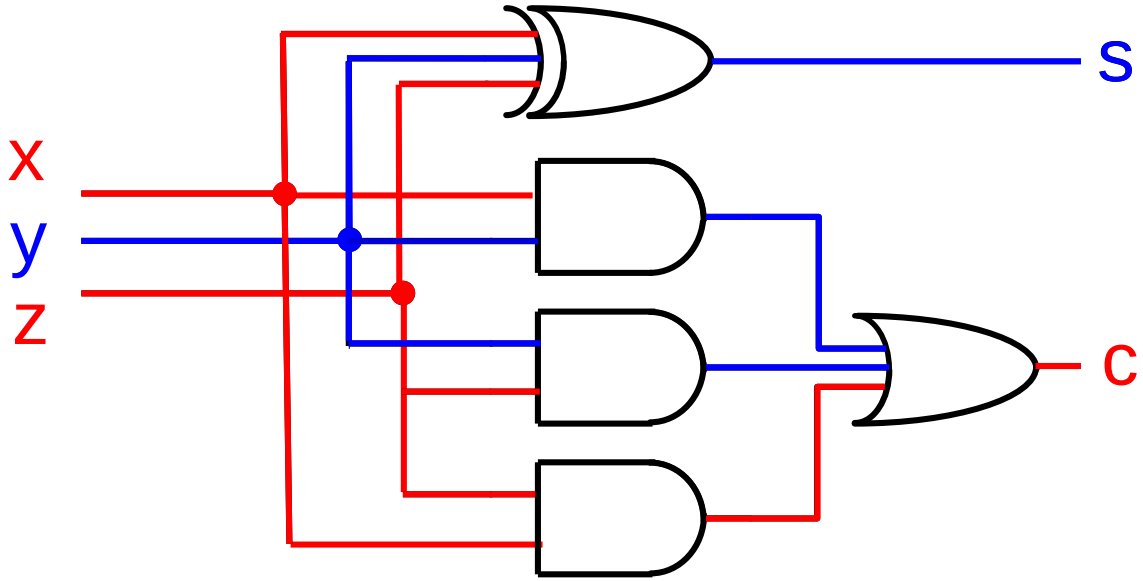
musicians need light + conductor to synchronize

Combinational Circuits: Thick Instants



Combinational Circuits: Thick Instants





s = x xor y xor z

c = (x and y) or (y and z) or (z and x)

x, y, z ⇒ s, c

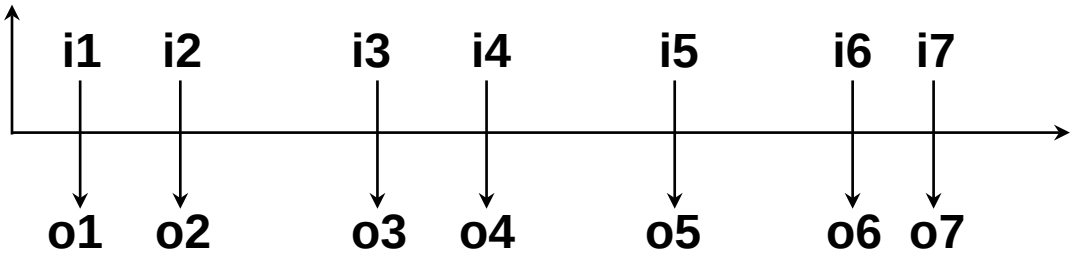
No time = Zero time

waiting for critical delay ⇔ solving the equations

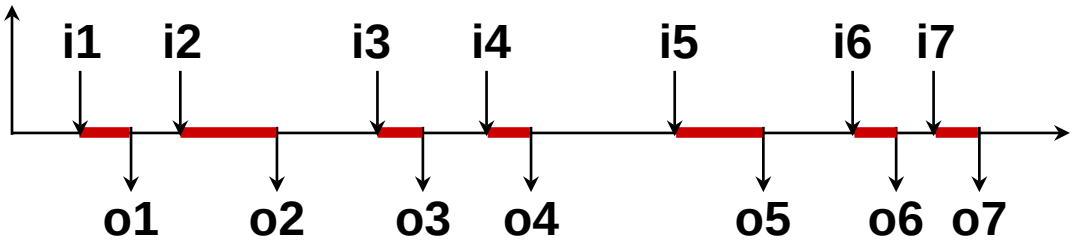
circuit = parallel vibration machine

solving parallel synchronous equations

Logical Time vs. Physical Time



Logical time



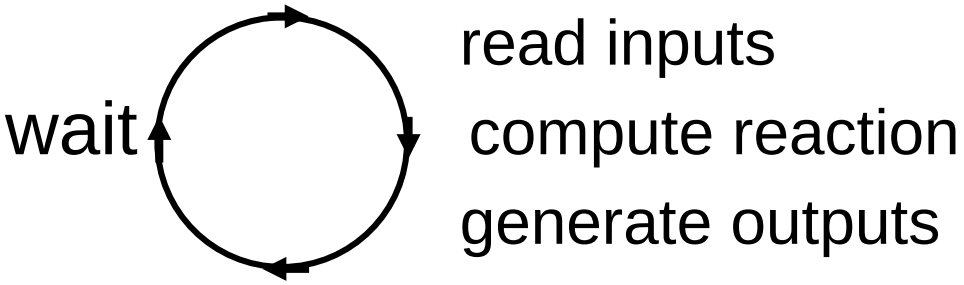
Physical time

cycle OK ⇔ no overlap

(clocks and events maybe irregular)

Cycle-Based Software Synchrony

Cyclic execution: static scheduling + 4-stroke engine



Synchronous = Zero-delay = within the same cycle

parallel propagation of control

parallel propagation of signals

Concurrency resolved at compile-time

$\Rightarrow$ determinism, no IO/computation interference

$\Rightarrow$ timing predictability (AbsInt)

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 do
   run RushToHospital
end trap
```

zero delay administration ⇒ determinism

exit HeartAttack

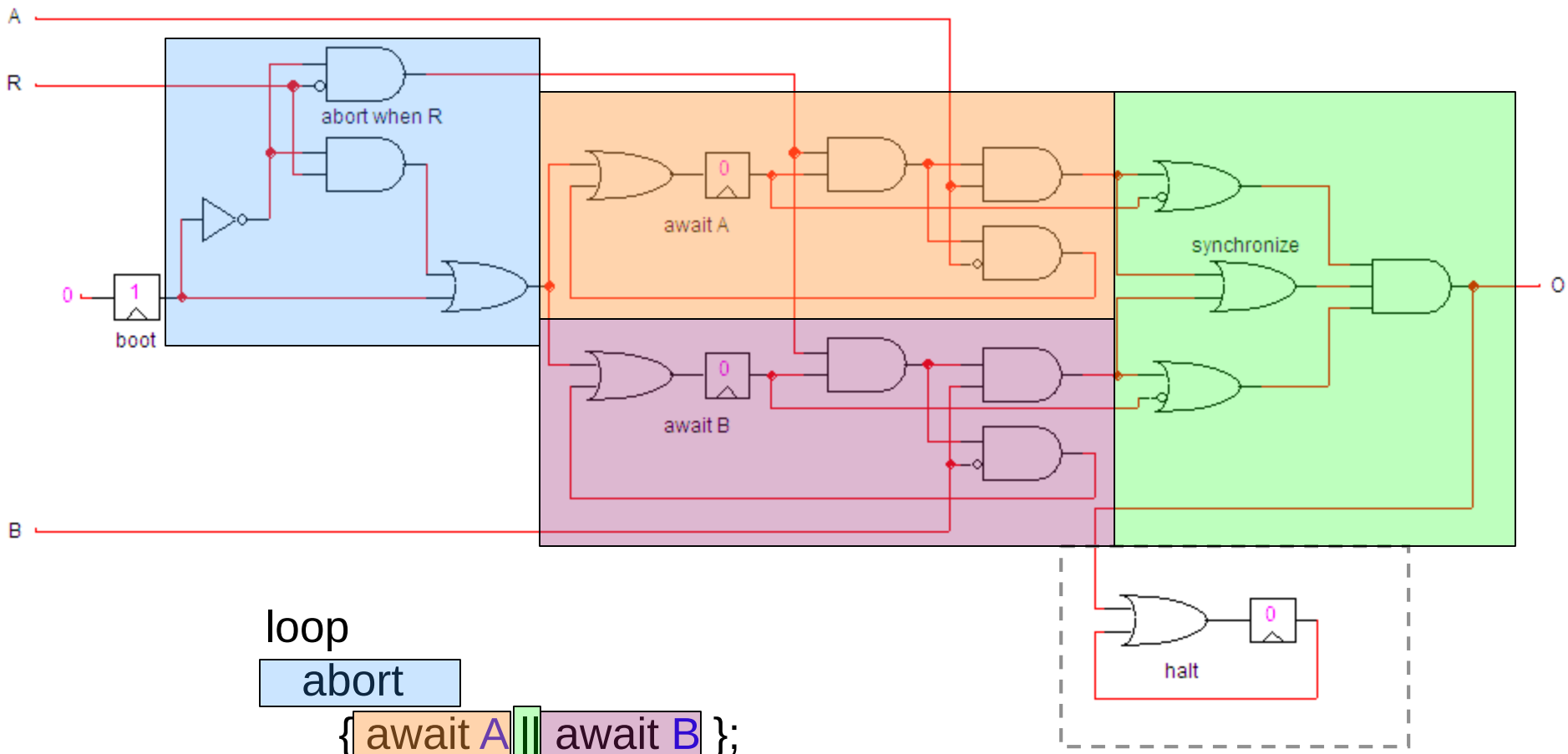
Mathematical Semantics

- All synchronous languages have fully formal and fully understandable mathematical semantics
- The mathematical semantics are fully respected by the academic and industrial compilers and by the interfaces to formal verification systems
- It becomes possible to build formally verified compilers (with Coq)
  - Esterel : L. Rieg, G. Berry
  - Lustre : T. Bourke, L. Rieg *et. al.*

Mathematical Semantics

- Esterel
  - rewriting-based constructive semantics (Plotkin's SOS rules)
  - translation to Boolean circuits, interpreted by constructive logic to handle combinational cycles when they occur
  - optimized HW generation and SW generation (either from the circuit or by direct techniques : S. Edwards, D. Potop)
- Esterel v7
  - Esterel + fancy datapath + multiclock, same semantics
  - generated circuits: better performance than human designs !
- Lustre
  - Fixpoint equations on infinite flows (= is equality)

Esterel Circuits = Proof Networks



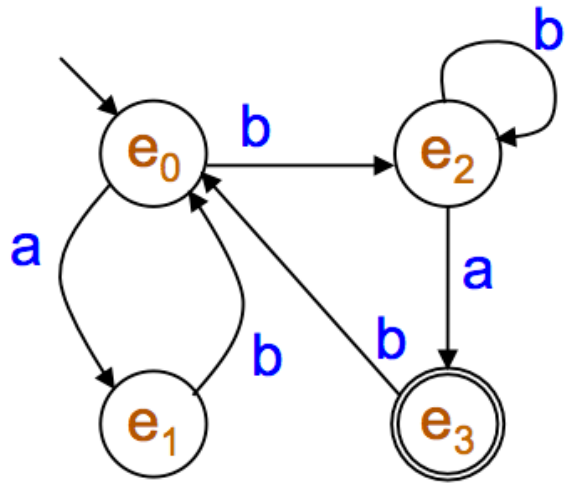

```
loop
  abort
      { await A || await B };
      emit O ;
    halt
  when R
end loop
```

suppressed
by optimization

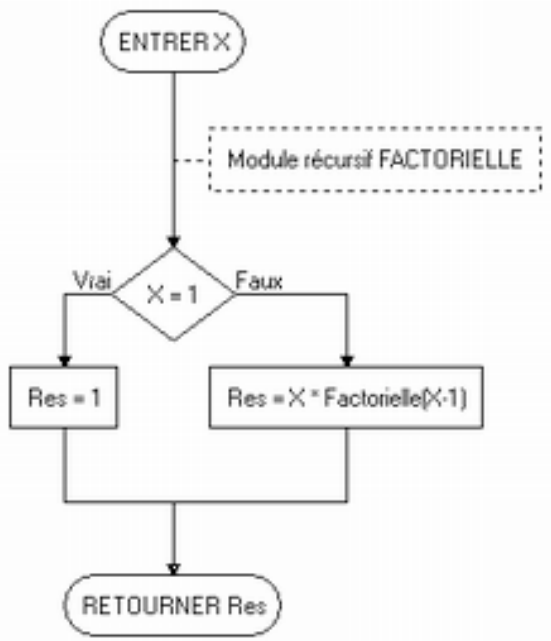
Other Synchronous Languages

- Signal (Le Guernic, Benveniste): relational Lustre
- ➢SyncCharts (C. André): graphical Esterel
- Reactive C (Boussinot), ReactiveML (L. Mandel): Esterel in C and OCaml
- SCADE 6 : SCADE + SyncCharts (Esterel Tech.)
- HipHop (C. Vidal, M. Serrano , GB) : Esterel in JavaScript for Web orchestration
- ??: optimized signal processing
- Zelus (M. Pouzet, T. Bourke): synchrony in modelers
- Ptolemy II synchronous domains (E. Lee)
- ➢SC (R. van Hanxleden) : Esterel in C + nice extensions

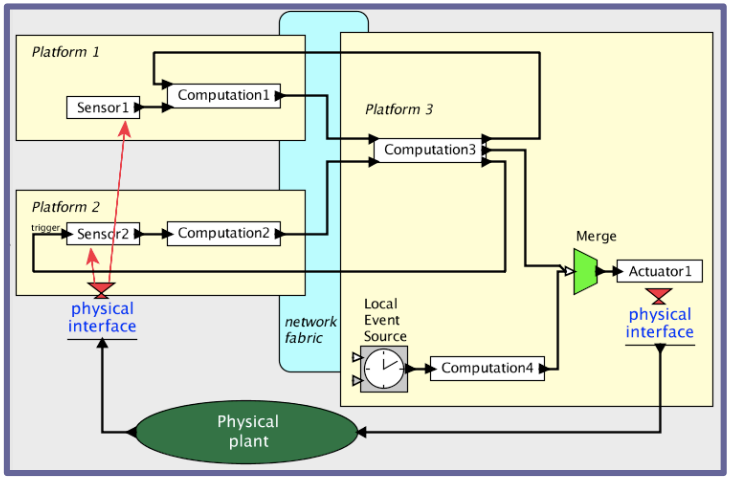
Traditional Graphical Syntax



state machines



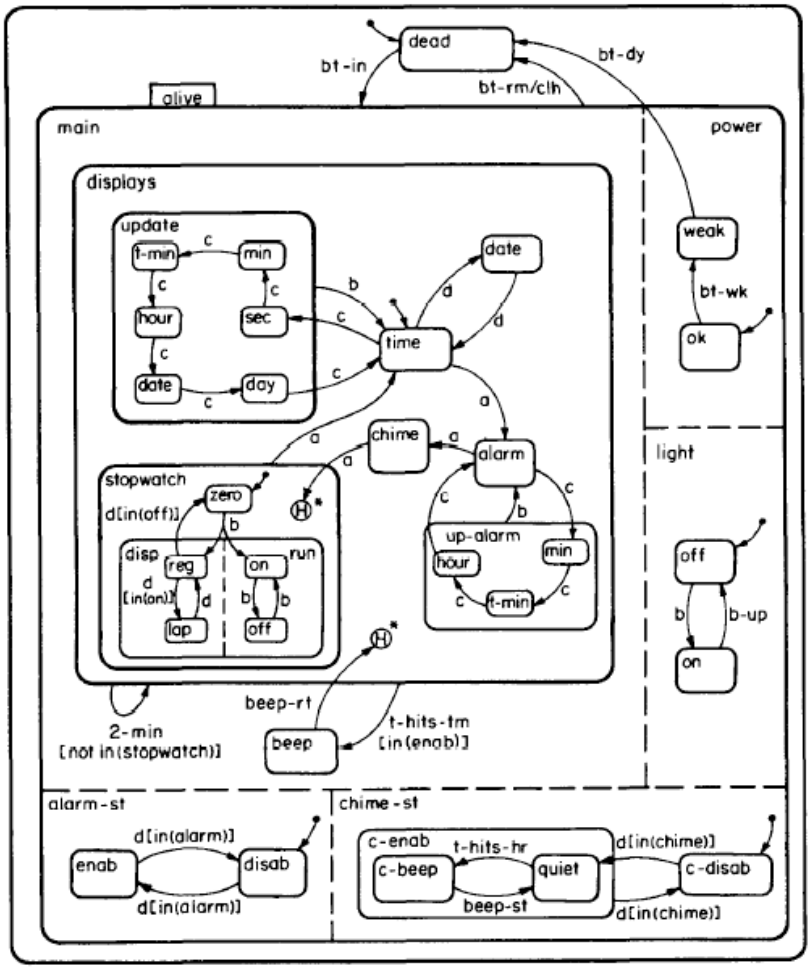
flow diagrams



block-diagrams

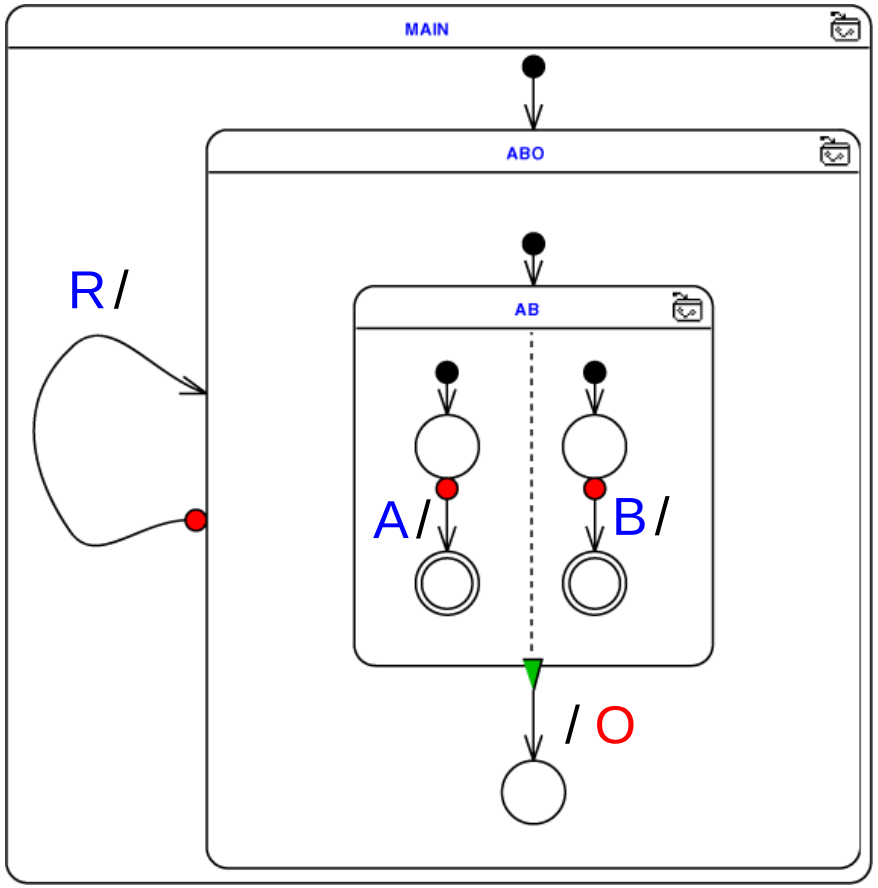
and many other diagrams,
cf. UML / SysML

Hierarchical State Machines



Statecharts, D. Harel

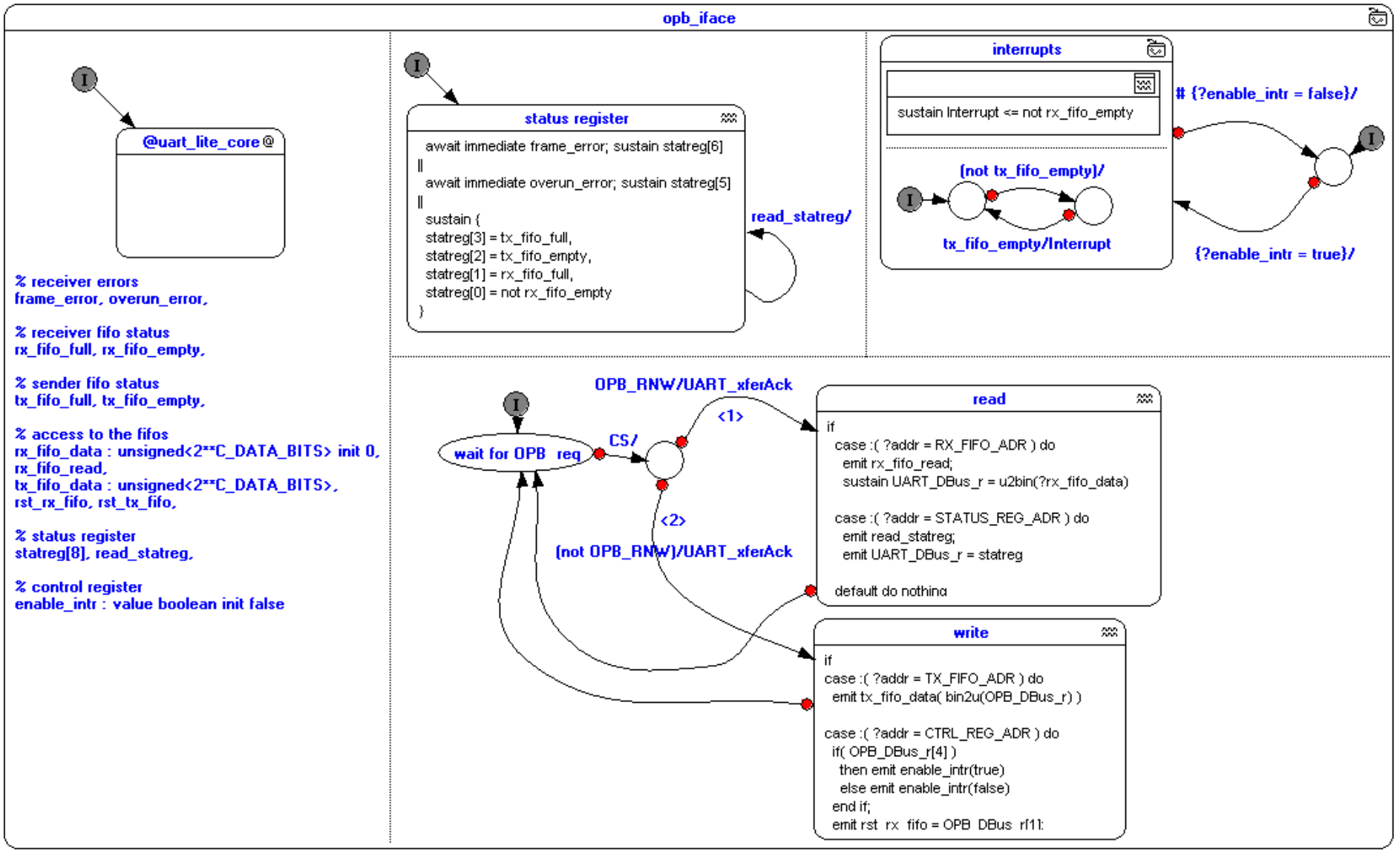
maximize expressivity



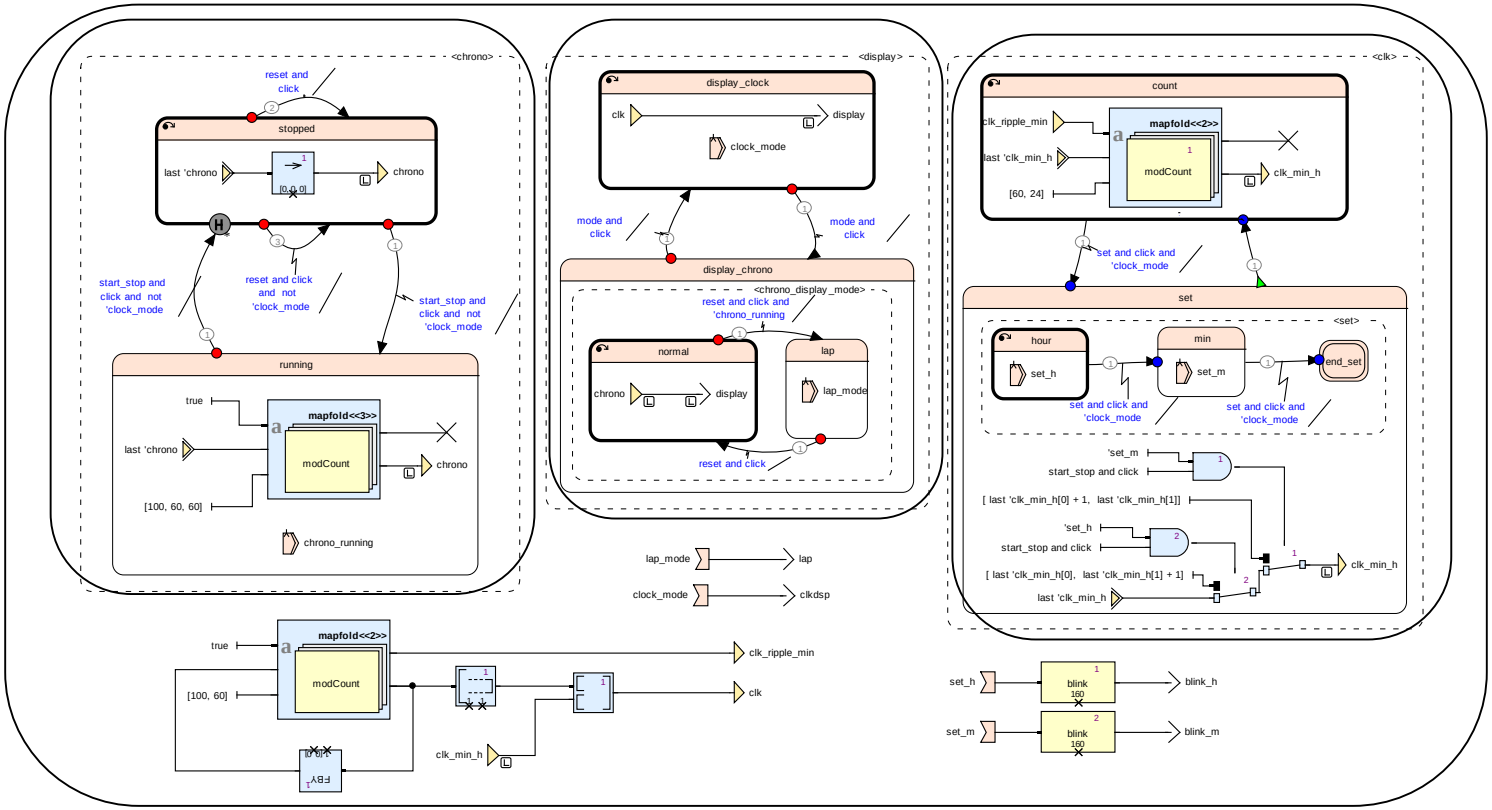
SyncCharts, C. André

based on Esterel, maximize rigor

Estrerel v7 for HW : SyncCharts + Esterel



SCADE 6 = Scade 5 + SyncCharts



SCADE 6 : parallelism is associative-commutative

Stateflow : fake parallelism for easier translation to C

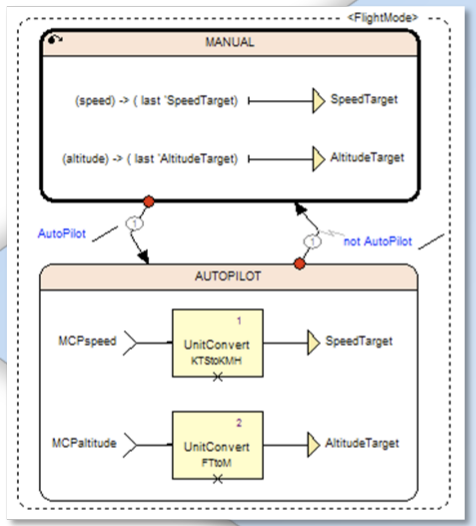
execution order defined by →↓ reading order ☹ ☹

semantics may totally change if one box is moved

other choices not better…

SCADE SUITE®

Control Software Design



PROTOTYPE & DESIGN

Model Checks

Formal Verification

Debug & Simulation

Plant Model Co-simulation (incl. FMI)

Time & Stack Optimization

HIL/SIL/PIL Integration

VERIFY

Calibration

SCADE Suite KCG

C & Ada

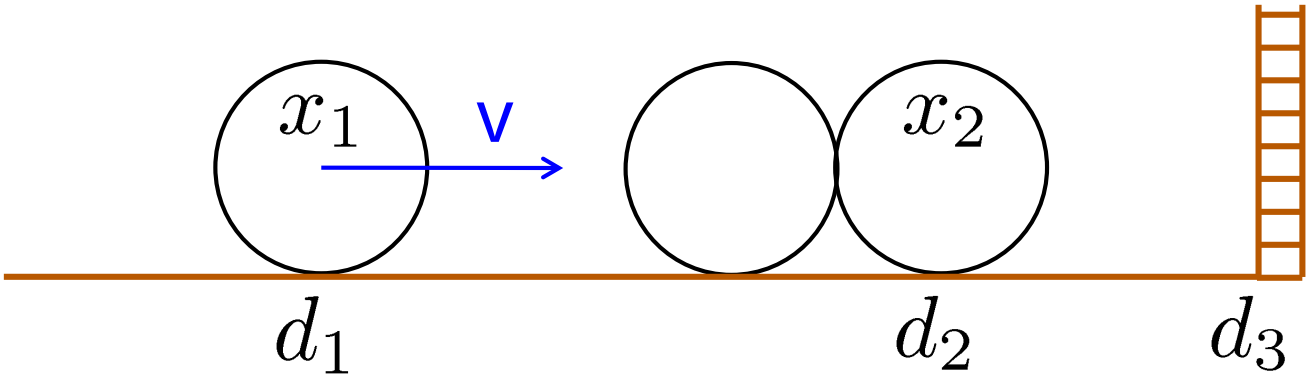
RTOS Adaptors

Object Code & Compiler Verification

DO-178B & C
IEC 61508
EN 50128
ISO 26262
Certification Kits

GENERATE

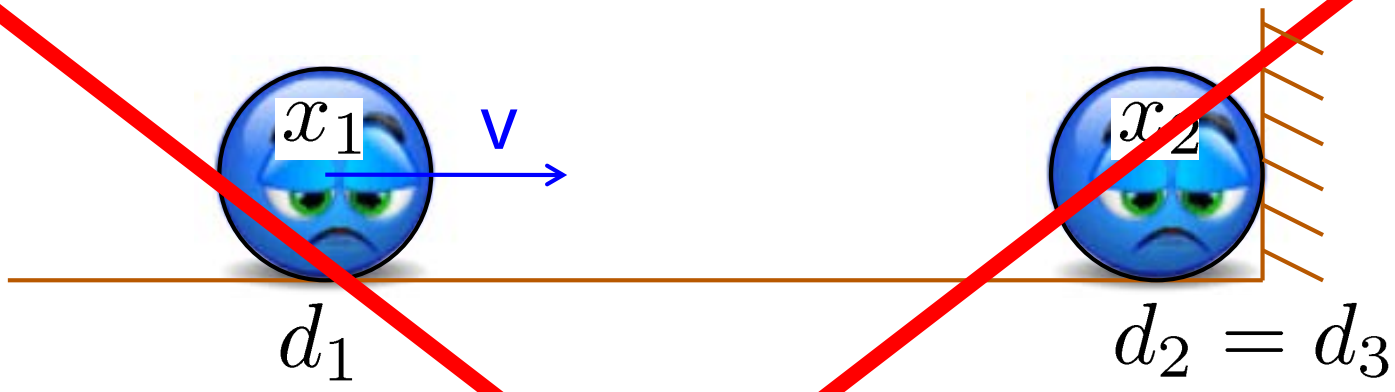
Balls On Wall – Current Modelers OK



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

shocks $\Rightarrow$ actions

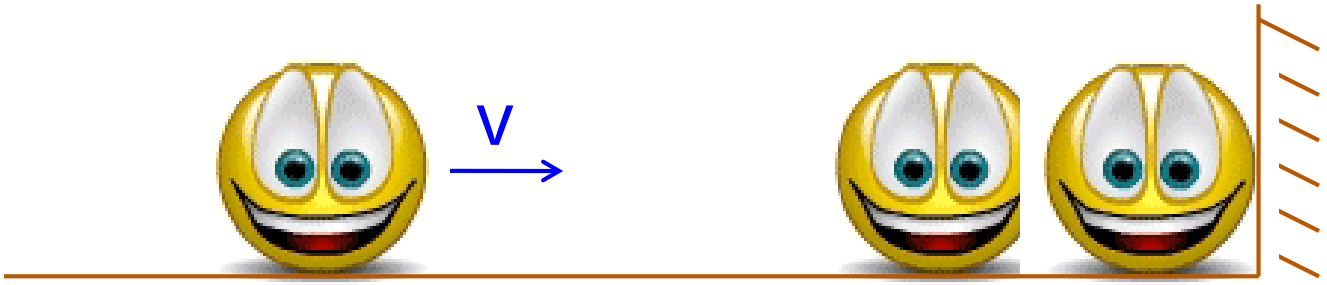
Ball Stuck on Wall – Current Modelers



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

$$
\begin{aligned}
& d_1 \ \mathbf{up}\ d_2 \Rightarrow v_1 = 0,\ v_2 = v \\
& \rightarrow x_2 \ \mathbf{up}\ d_3 \\
& \Rightarrow \ v_2 = -v \\
& \rightarrow x_1 \ \mathbf{up}\ x_2 \\
& \Rightarrow v_1 = -v,\ v_2 = 0
\end{aligned}
$$

Practical approach : Zelus by M. Pouzet and T. Bourke:
use type checking to split continuous and 0-delay actions

Conclusion

- Mature technology with many applications
- Successful transfer from Academia to Industry
- Much simpler than other programming techniques for its application domain
- Based on compromize-free mathematical semantics
- Research continues
  - Zelus (ENS Paris) for modelers
  - Sc (Kiel) for elegant and sound inclusion into C
- But Esterel now deep frozen at Synopsys…

Thank You!

Bibliography

- **The Foundations of Esterel**. G. Berry. In Proof, Language and Interaction: Essays in Honour of Robin Milner, G. Plotkin, C. Stirling and M. Tofte, editors, MIT Press, Foundations of Computing Series, 2000. [PDF]
- **The Constructive Semantics of Pure Esterel.** G. Berry. Draft book, current version 3.0, Dec. 16th, 2002. [PDF]
- **Programming and verifying critical systems by means of the synchronous data-flow programming language Lustre.** N. Halbwachs, F. Lagnier and C. Ratel. *IEEE Transactions on Software Engineering, Special Issue on the Specification and Analysis of Real-Time Systems.* September 1992. [Abstract and Postscript]
- **The synchronous languages 12 years later.** A. Benveniste, P. Caspi, S.A. Edwards, N. Halbwachs, P. Le Guernic, and R. de Simone. *Proceedings of the IEEE, Vol. 91, Nr. 1,* January 2003 [PDF]
- **SCADE: Synchronous design and validation of embedded control software**. G. Berry. In Next Generation Design and Verification of Distributed Embedded Systems, S. Ramesh and P. Sampath Ed., Springer Verlag, 2007. [PDF]
- **Compiling Esterel.** D. Potop, S. Edwards, and G. Berry. Springer, 2007.