

# Synchronous Programming : Strengths and Challenges

G rard Berry

Ecole des Mines and INRIA  
Sophia-Antipolis, France

[Gerard.Berry@inria.fr](mailto:Gerard.Berry@inria.fr)  
<http://www.esterel.org>

- Technology born in 1983-1985

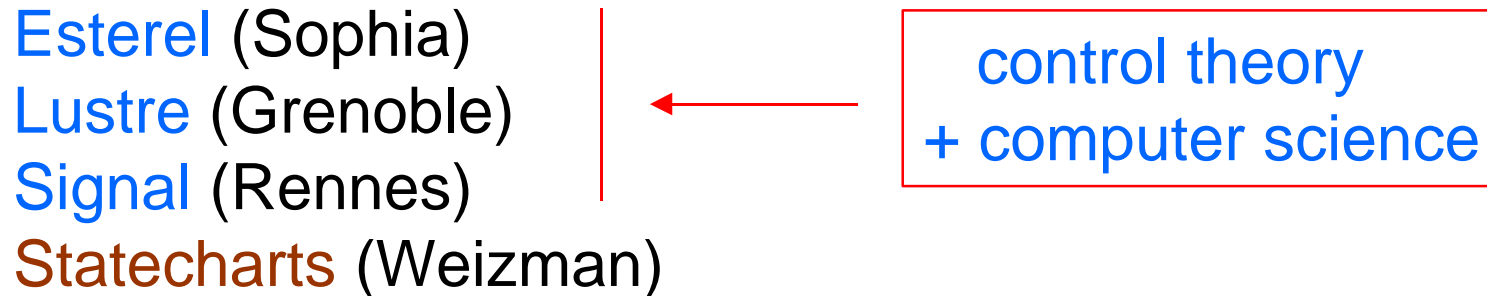
Esterel (Sophia)

Lustre (Grenoble)

Signal (Rennes)

Statecharts (Weizman)

control theory  
+ computer science



- More recently

Argos (Grenoble), SyncCharts (Nice)

Reactive C, SugarCubes (Ecole des Mines)

Ptolemy (Berkeley)

sE (GMD)

ECL (Cadence), Jester (Parades)

Lustre + ML (Paris VI)

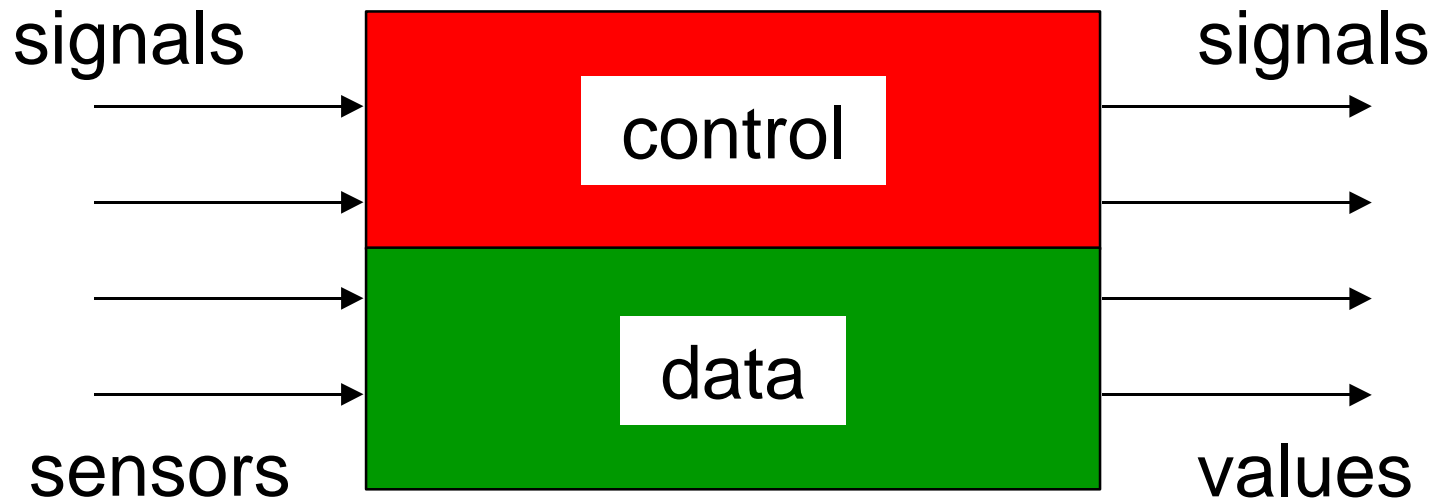
Synchronous Verilog (Cadence, Intel)

Mode Automata (Grenoble)

# Industrial products

- **Scade** (Telelogic)  
Airbus, Saab, Schneider, Volvo, ...
- **Sildex** (TNI)  
SNECMA, EDF, PSA, Nortel
- **Esterel Studio** (Esterel Technologies)  
Dassault, Thomson, Texas Instruments, VPI, ...
- **CoCentric Studio** (Synopsys)
- **ECL** (Cadence)

# Reactive Systems



Esterel  
Statecharts  
Argos  
SyncCharts

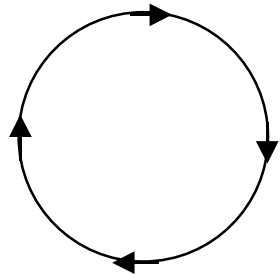


Lustre  
Signal



# Software / Hardware Synchrony

## Cycle based



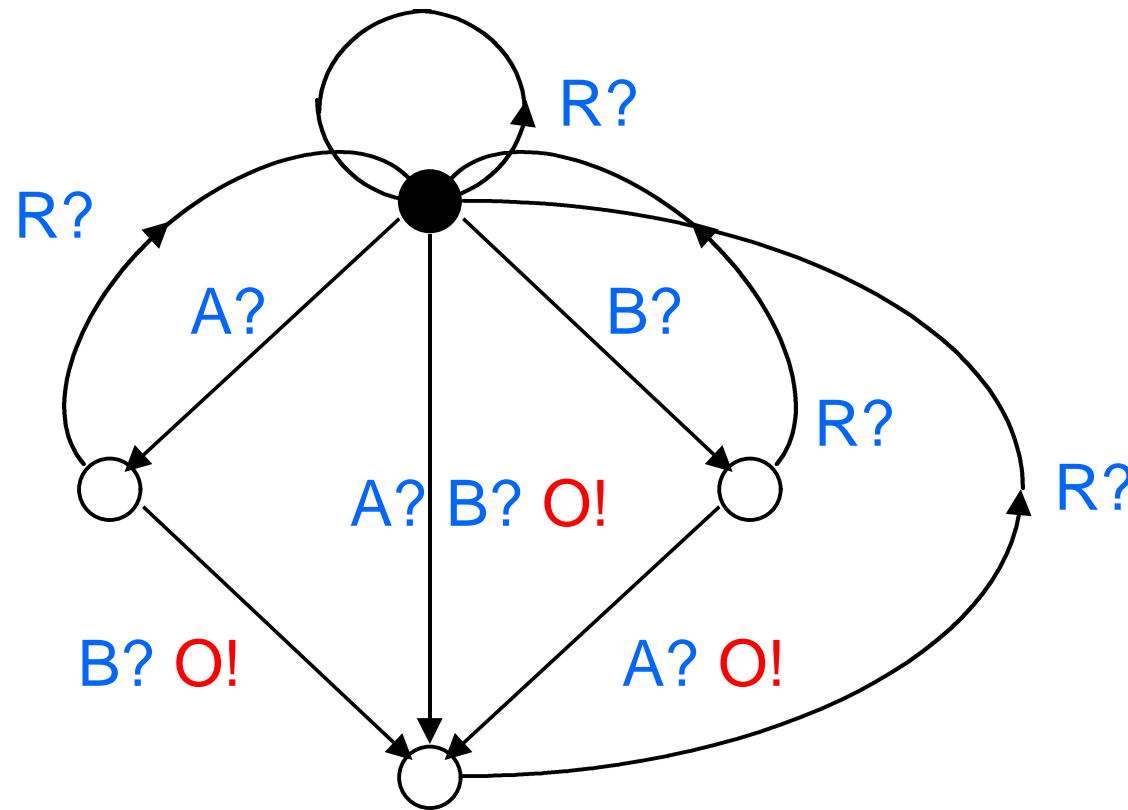
read inputs  
compute reaction  
produce outputs

**Synchronous** = within the same cycle  
control propagation  
signal propagation

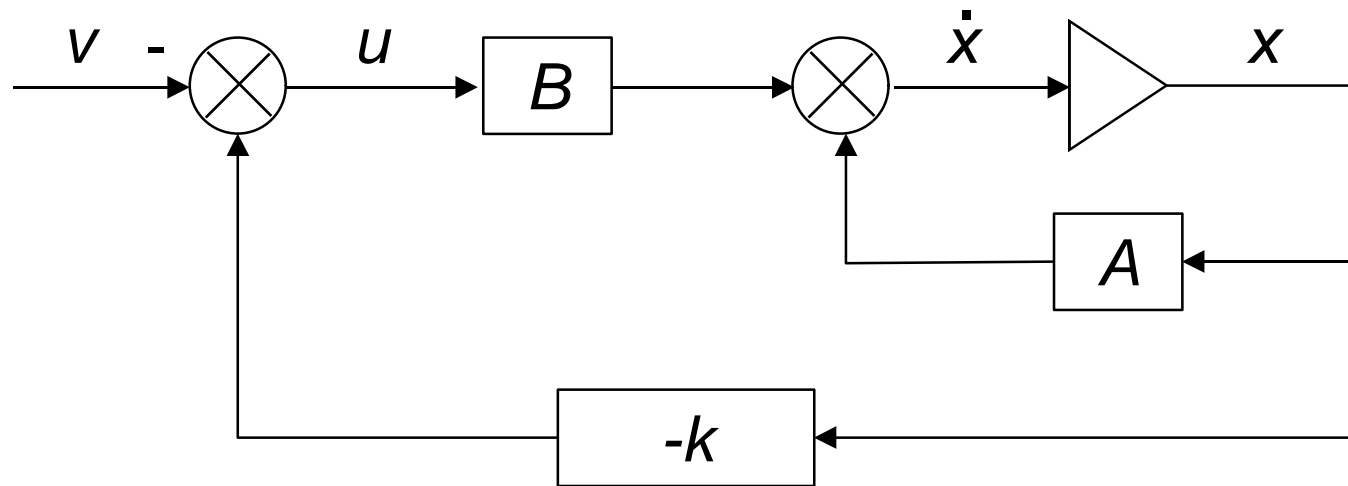
**Zero Delay** : abstract model, nice mathematics

# Automata

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

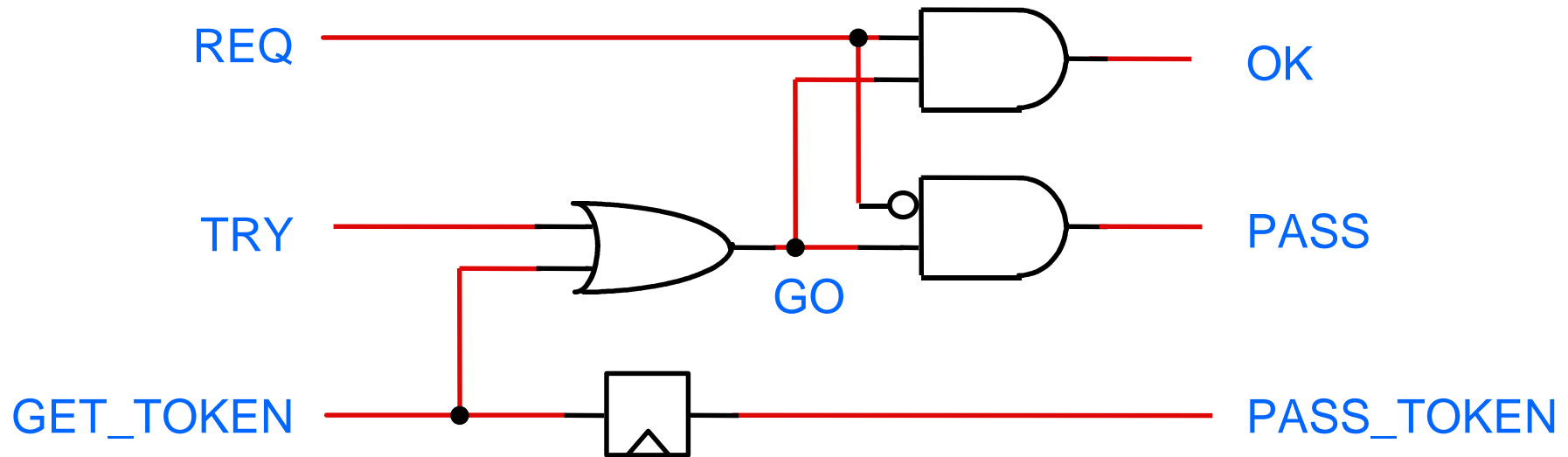


# Block Diagrams



$$\dot{x} = (A - Bk)x + Bv$$

# Sequential Circuits



OK = REQ and GO

PASS = not REQ and GO

GO = TRY or GET\_TOKEN

PASS\_TOKEN = reg(GET\_TOKEN)

# The strengths of synchrony

- Simple conceptual model  
clean languages, clean semantics
- Behavioral determinism  
easier to program and to debug  
concurrent threads are always well coordinated
- Compositionality  
a chain of communication is like a communication
- Tractable mathematics => optimization + verification  
the Boolean Calculus
- Controllable approximation of "reality"

Exactly as for good old Newtonian Mechanics!

# Constraints for Languages

## 1. Clean design

programs should mean what they say  
full construct orthogonality  
no hack, no subset

## 2. Mathematical semantics

not just formal !  
used as the golden spec for everything

## 3. Good toolset

simulation  
synthesis  
verification

# Lustre, Signal : Data-Flow

$$\left| \begin{array}{l} X_0 = 0 \\ X_{t+1} = U_{t+1} * \sin(X_t + S_{t+1} - S_t) \\ S_0 = 1 \\ S_{t+1} = \cos(S_t + U_{t+1}) \end{array} \right.$$

node **Control** ( $U$  : float) returns ( $X$  : float);

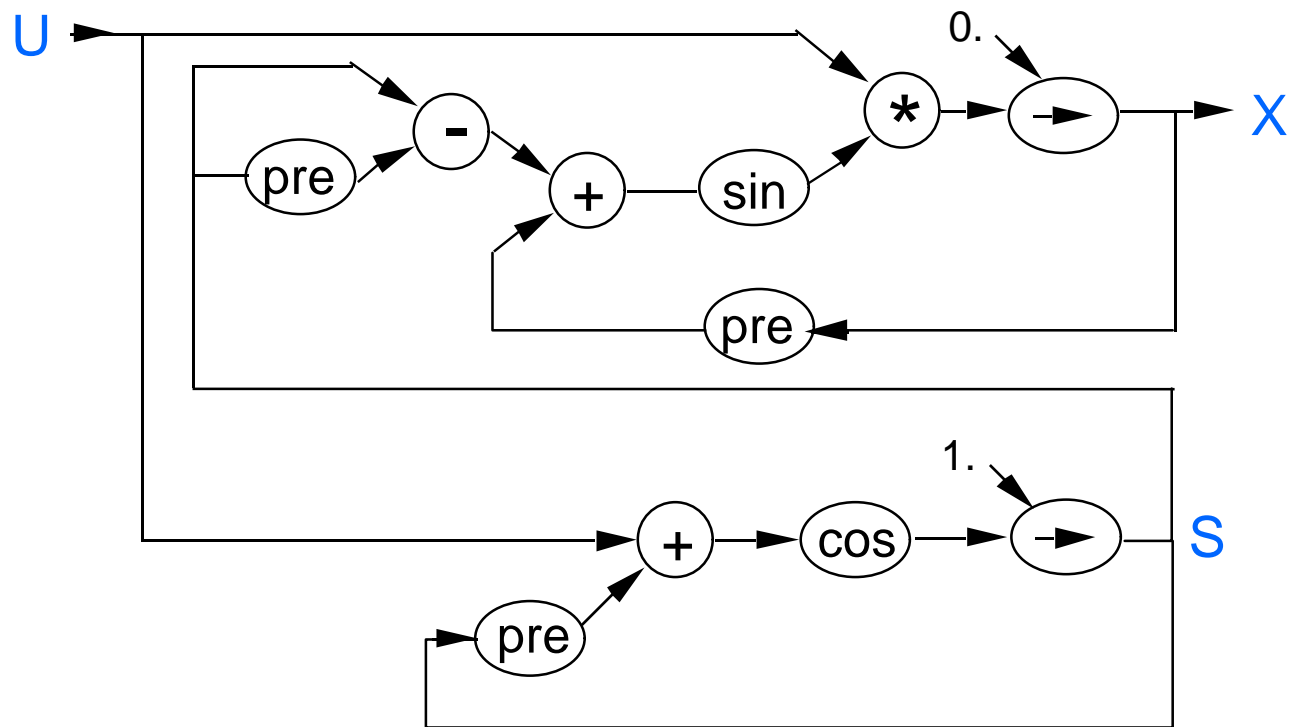
var  $S$  : float;

let

$X = 0.$  -> ( $U * \sin(\text{pre}(X) + S - \text{pre}(S))$ );

$S = 1.$  ->  $\cos(\text{pre}(S) + U)$ ;

tel





# Lustre Operators

$$X = X_0, X_1, \dots, X_n, \dots$$

$$(X + Y)_n = X_n + Y_n$$

$$\text{pre}(X) = \text{nil}, X_0, X_1, \dots, X_n, \dots$$

$$X \rightarrow Y = X_0, Y_1, Y_2, \dots, Y_n, \dots$$

$$X = 1, 2, 3, 4, 5, 6, 7, 8, \dots$$

$$C = 0, 1, 1, 0, 1, 0, 1, 1, \dots$$

$$X \text{ when } C = 2, 3, 5, 7, 8, \dots$$

# The Definition Principle

$$X = e$$

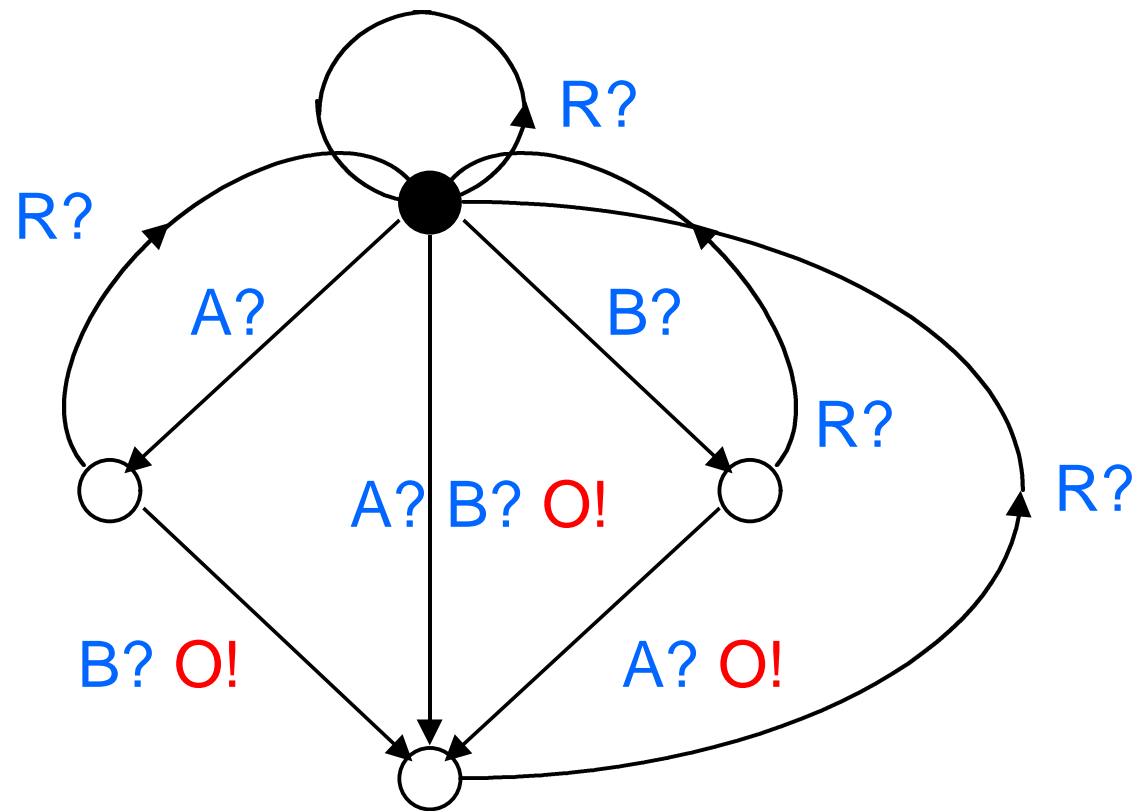
means  $X$  is equal to

i.e.  $X_n = e_n$  for all  $n$

i.e.  $X$  can be replaced by  $e$  anywhere

Clock calculus :

program executable in finite memory  
(compatible **when** undersamplings)

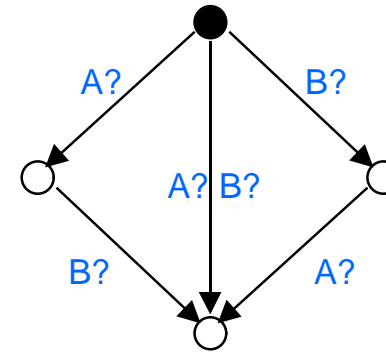


# Esterel : Write Things Once

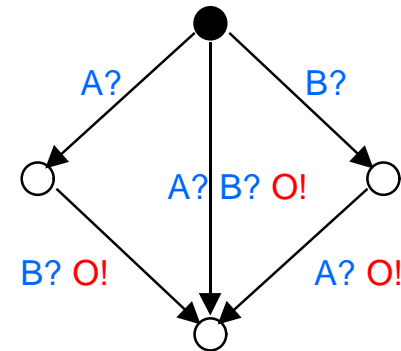
```
loop
  [ await A || await B ] ;
  emit O
each R
```

- concurrency
- sequencing
- preemption
- full orthogonality

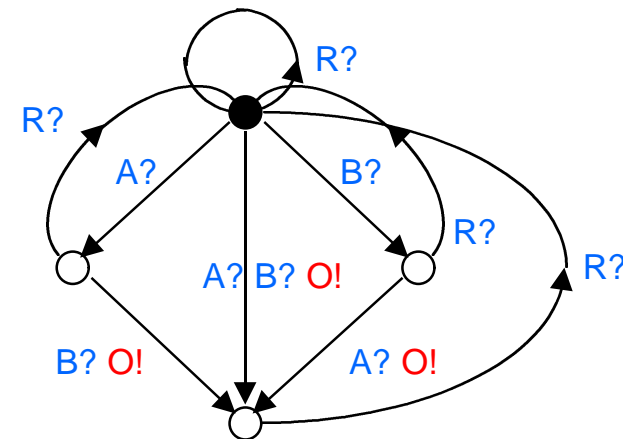
[ await **A** || await **B** ]



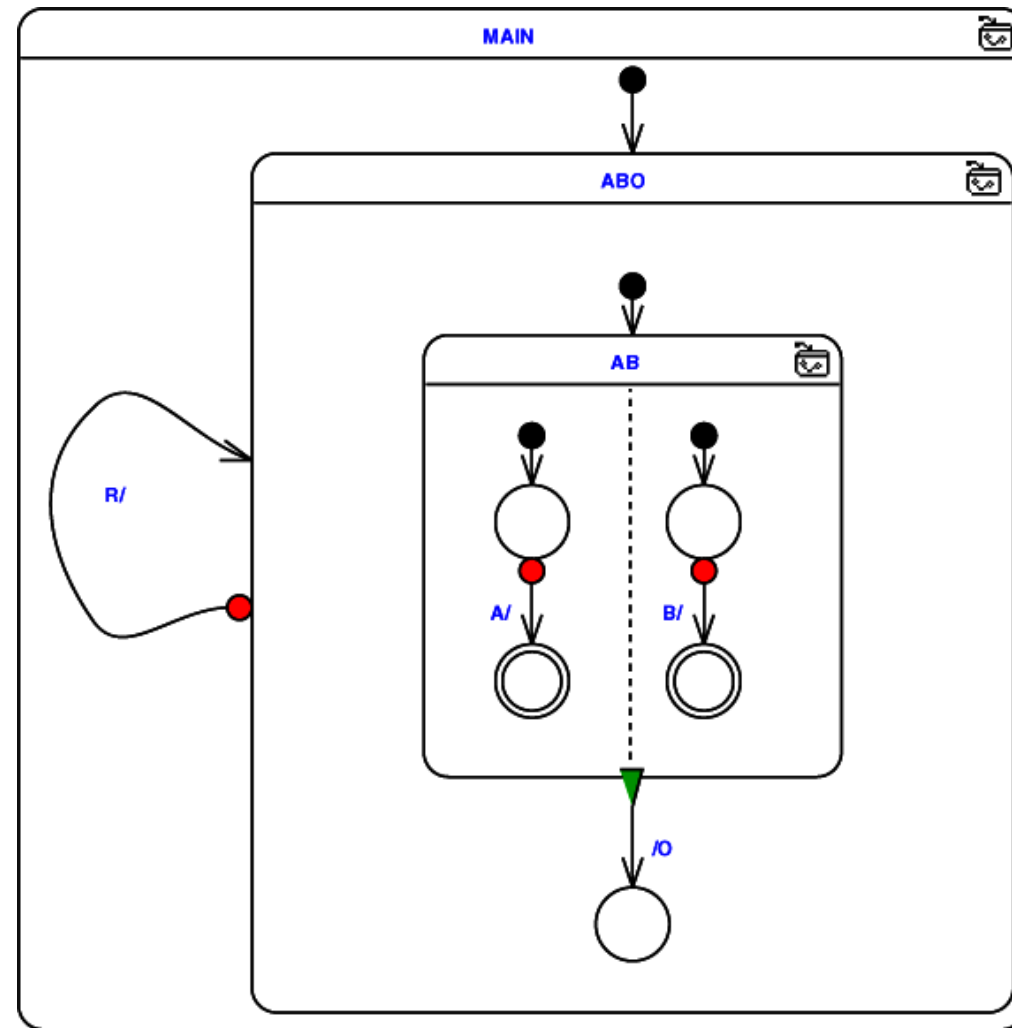
[ await **A** || await **B** ] ;  
emit **O**



loop  
  [ await **A** || await **B** ] ;  
  emit **O**  
each **R**



# SyncCharts : WTO, graphical



# Programs mean what they say

```
module SPEED : % computes exact speed
input Centimeter, Second;
relation Centimeter # Second;
output Speed : integer;

loop
  var Distance := 0 : integer in
    abort
      every Centimeter do
        Distance := Distance + 1
      end every
    when Second;
    emit Speed(Distance)
  end var
end loop
end module
```

# The Pure Esterel Kernel

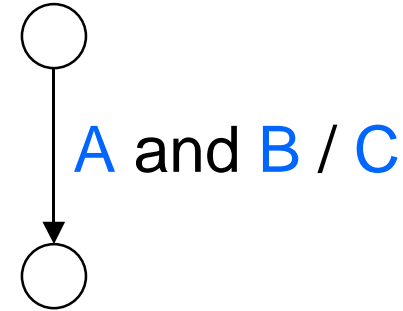
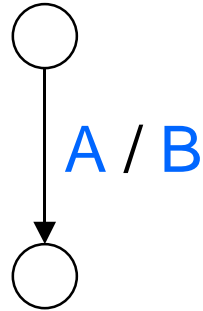
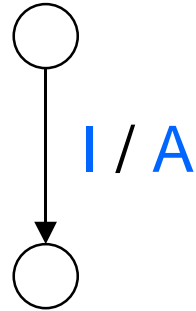
nothing	0
pause	1
emit S	! s
present S then p else q end	s ? p, q
suspend p when S	s $\supset$ p
p; q	p; q
loop p end	p*
p    q	p   q
trap T in p end	{p} $\uparrow$ p
exit T	k, k > 1
signal S in p end	p \ s



# Mathematical Semantics

- Behavioral Semantics: logical **consistency**  
A signal is present if and only if it is emitted  
Reactivity and determinism required
- Constructive Semantics: logical **constructiveness**  
Explain **why** signals are present or absent  
Reactivity and determinism implied
- Operational Semantics: **microsteps**  
Non-deterministic but **confluent** execution

Thm : Constructive = Operational < Behavioral



Lustre  
equations

$A = I$   
 $B = A$   
 $C = A \text{ and } B$

$A = B = C = I$

Esterel  
microsteps

$I = 1$   
 $\Rightarrow A = 1$   
 $\Rightarrow B = 1$   
 $\Rightarrow C = 1$

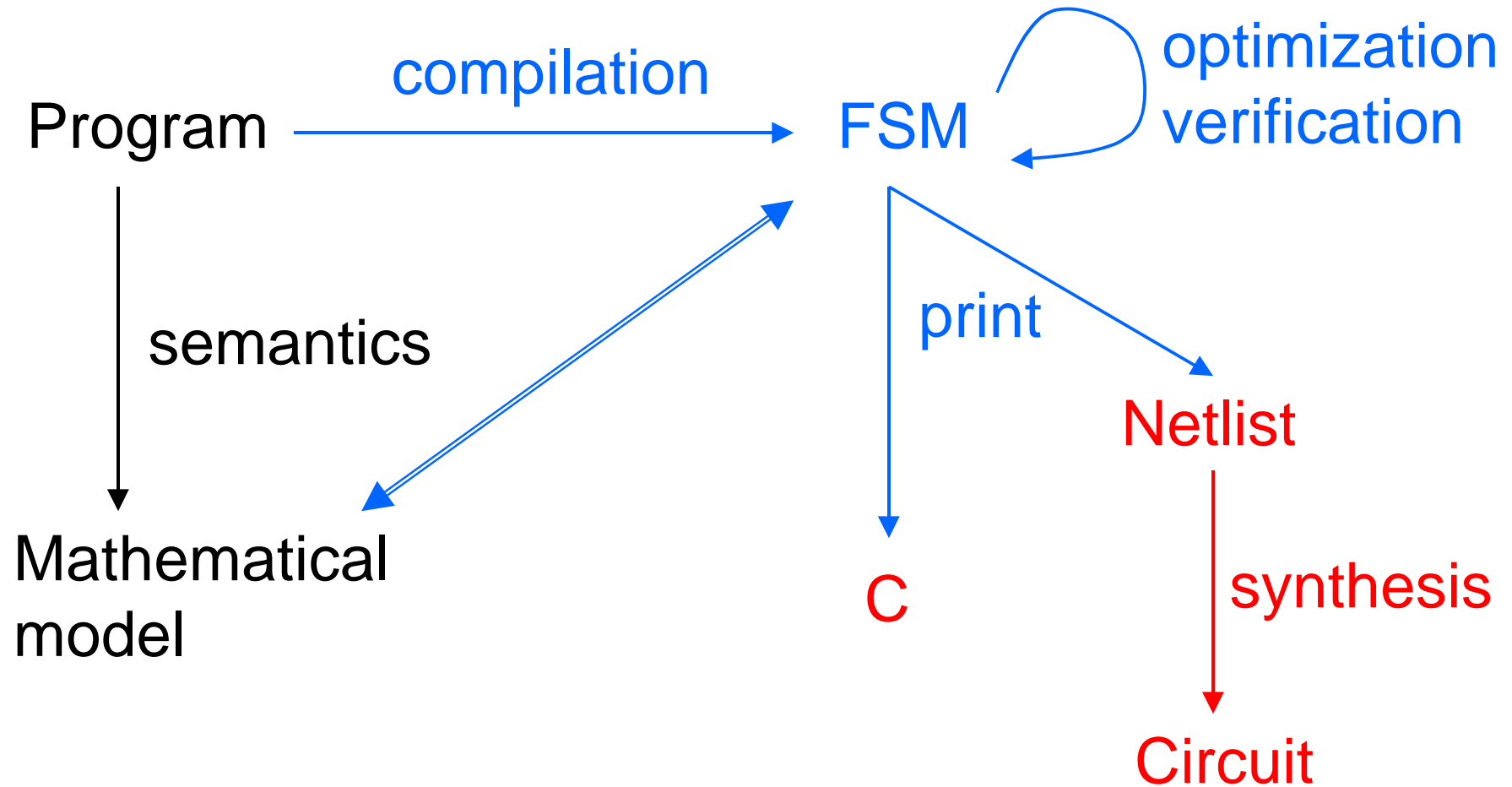
Statecharts  
microsteps

$I = 1$   
 $A = 1$   
 $B = 1$   
 that's all !

$\Leftrightarrow$

$\nRightarrow$

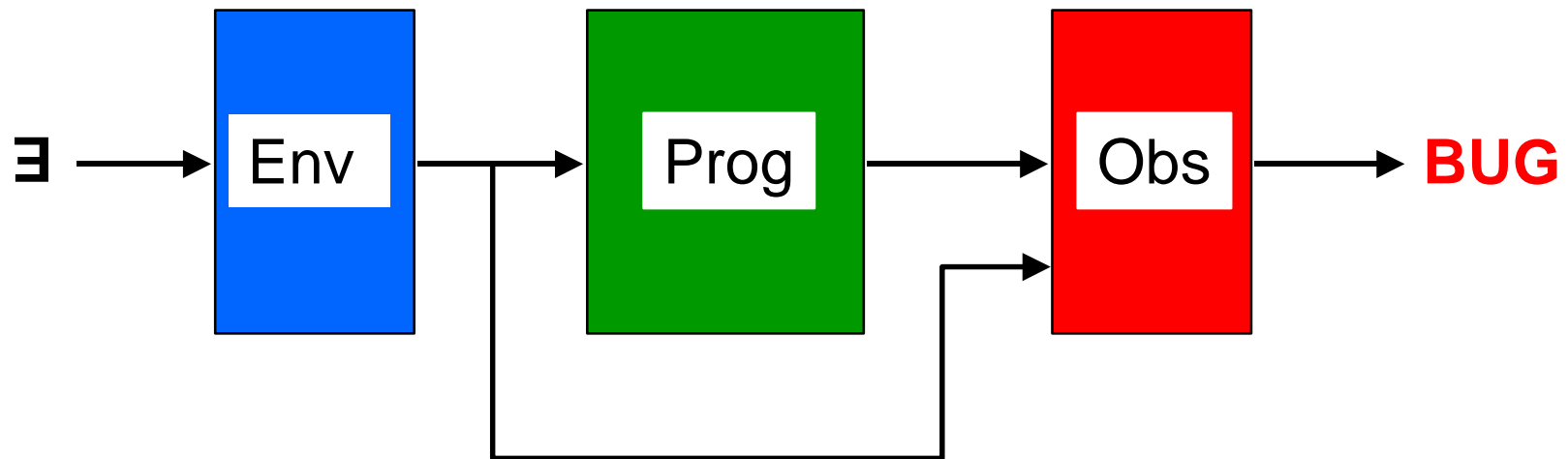
# The Design Flow



# Optimization

- Static analysis techniques
  - compute **when** to execute basic statements
  - Lustre, Signal : clock calculus
  - Esterel, SyncCharts : serial / parallel structure
- Circuit CAD techniques (Esterel, SyncCharts)
  - translate program into Boolean circuit
  - instantaneous  $\Leftrightarrow$  combinational**
  - clever state assignment techniques
  - optimize speed for hardware
  - optimize area for software !

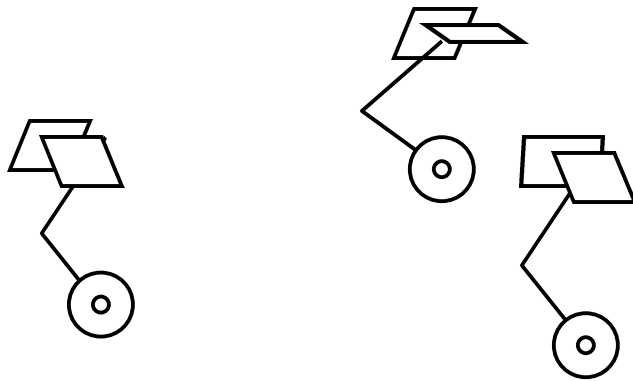
# Verification by Observers



Reachability analysis (BDD)

Specific algorithms for controllers (TiGeR)

# Landing gear (Dassault)



loosely coordinated moves  
gear / traps interaction

- gear never out without request
- gear never out unless traps locked open
- medium synchronization points OK
- if nothing gets wrong, gear fully locked in 14 seconds

# Weaknesses of Synchronous Languages

# Challenges of Synchronous Languages

- Very strong formal framework, but several languages gateways exist (dc), but can they be unified ?
- Good optimization verification tools, but still scaling problems can we gain an order of magnitude ?
- Base notions limited to relatively compact systems what about distributed systems ?  
what about systems on a chip ?
- Synchronous causality issues are difficult (Esterel) can we make them easier for the user ?

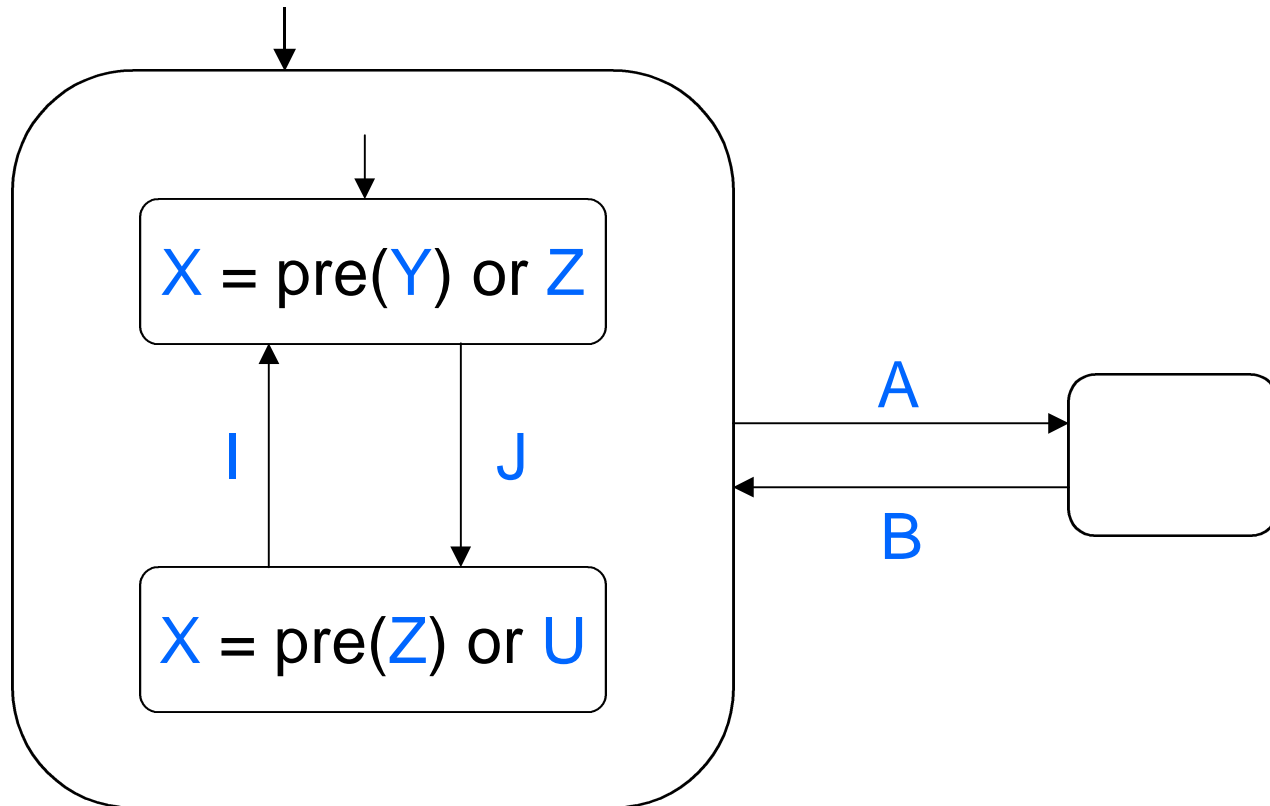


# Challenge 1 : unify the languages

- Lustre, Signal : good for data-flow
- Esterel, SyncCharts : good for control flow
- standard application : a few large control or data blocks
- modern application : a fine-grain mix
  - UMTS, military radio
    - sophisticated signal processing
    - + many mode changes
  - Modern circuits
    - fancy pipelined operator
    - + complex memory access control
- + integration with other methods, e.g. UML

# Mode automata

F. Maraninchi, Y. Rémond (Grenoble)



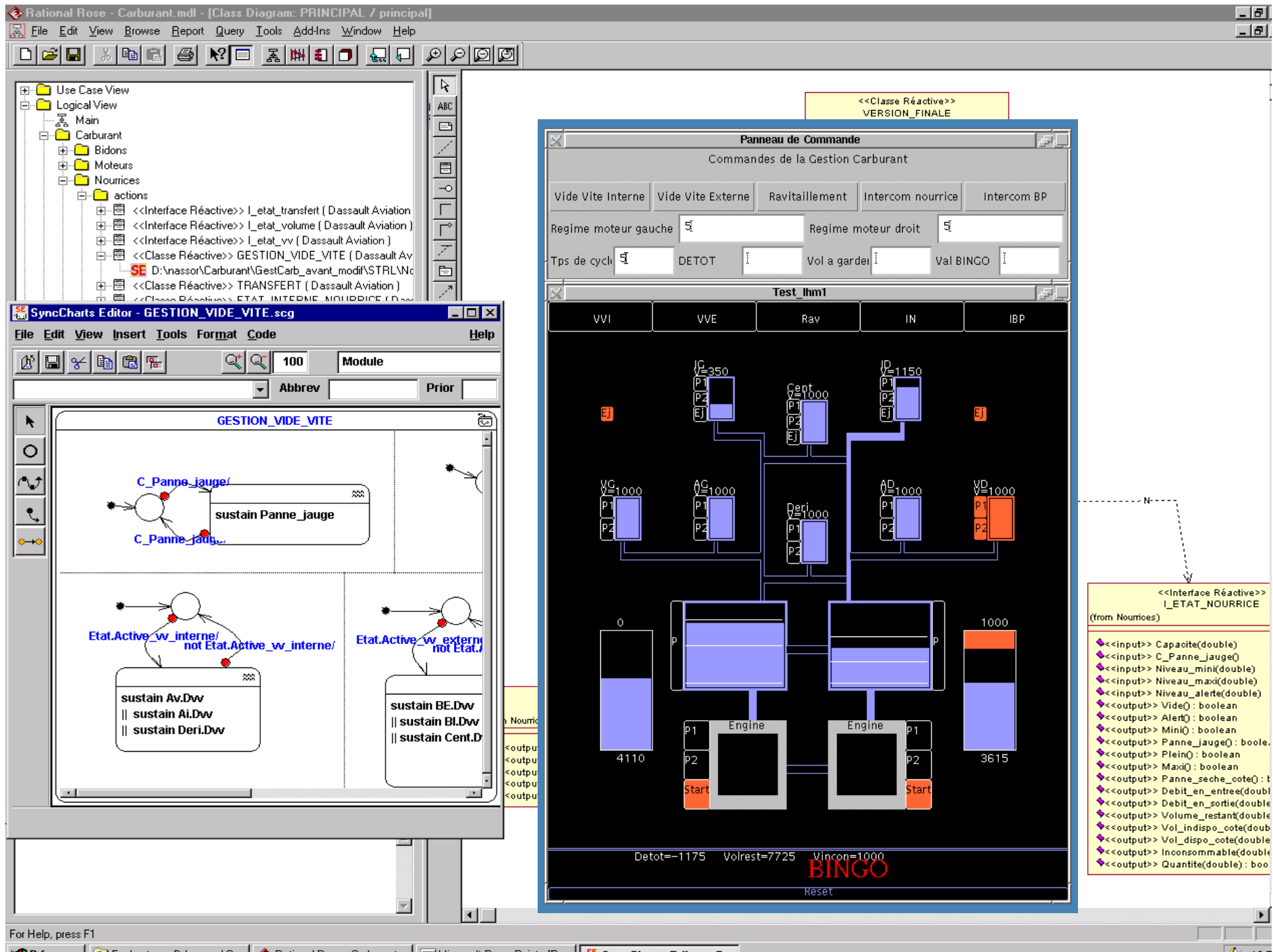
# Hardware Esterel

G. Berry, M. Kishinevsky (Intel)

```
loop
  abort
    def X = Y and Z
    def ?U = ?V + 1 when W
  when I;
  abort
    def X = pre(Y) when U
  when J
end loop
```

non-trivial



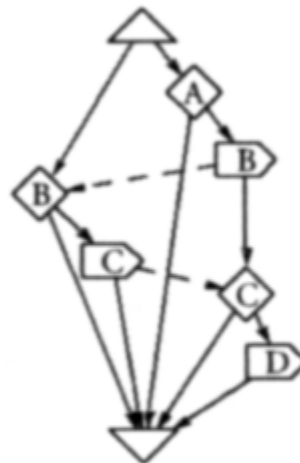


## Challenge 2 : very fast code generation

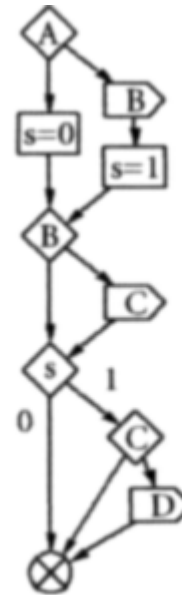
- goal : beat the human programmer
- use : simulation, golden models, embedded code
- S. Edward's (Synopsys), D. Pulou - D. Weil (France Télécom)  
fancy static scheduling of concurrency graphs

```
present A then  
  emit B;  
  present C then  
    emit D  
  end  
end  
||  
present B then  
  emit C  
end
```

(a)



(b)



(c)

```
if (A) {  
  B=1;  
  s=1;  
} else  
  s=0;  
if (B)  
  C=1;  
if (s)  
  if (C)  
    D=1;
```

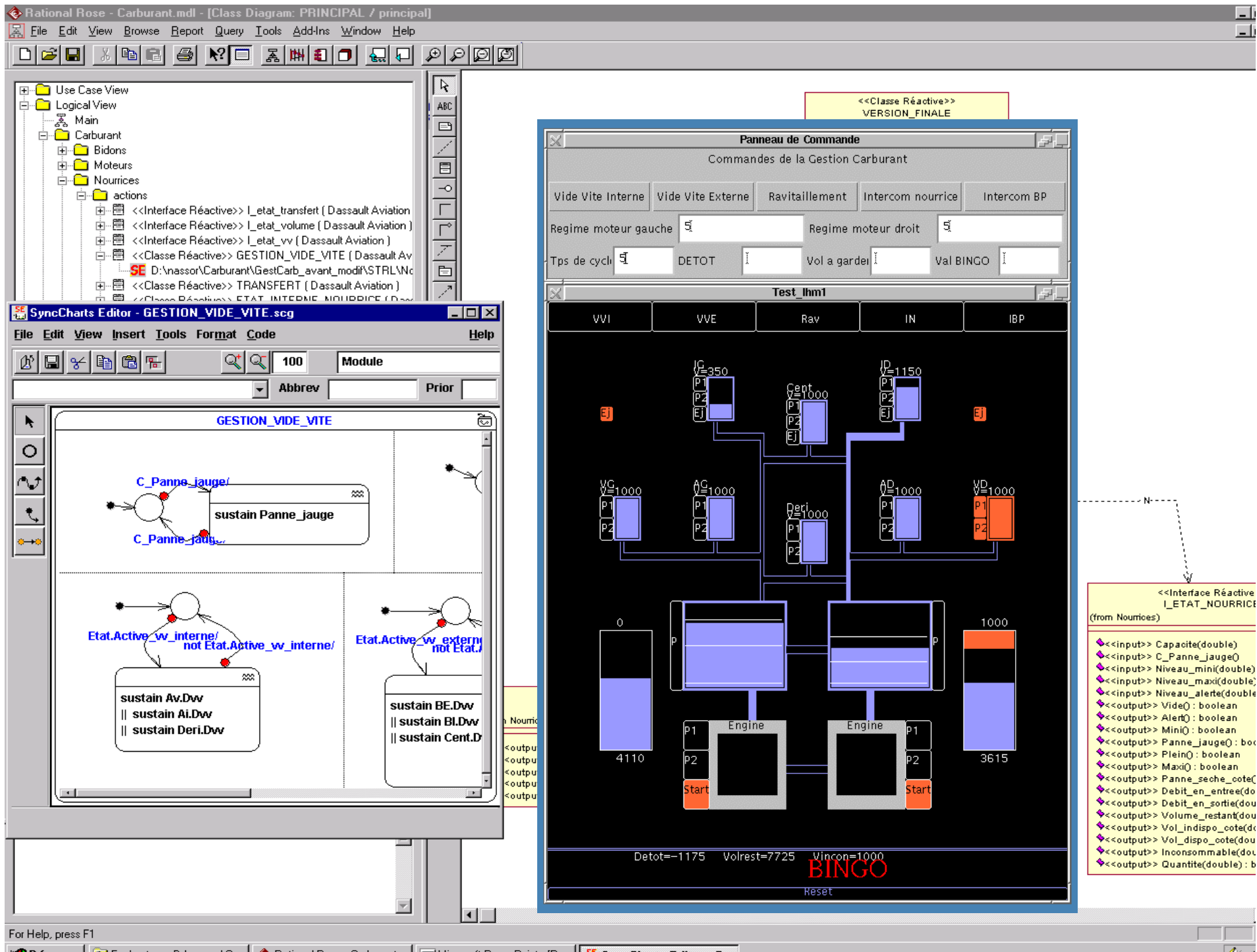
(d)

## Challenge 2' : very fast hardware generation

- goal : beat the human designer (oops...)
- use : FPGAs, glue logic, microprocessor control

state encoding  
+ information encoding  
= good register / logic balance  
+ fancy optimization games

## Challenge 2'' : very good codesign



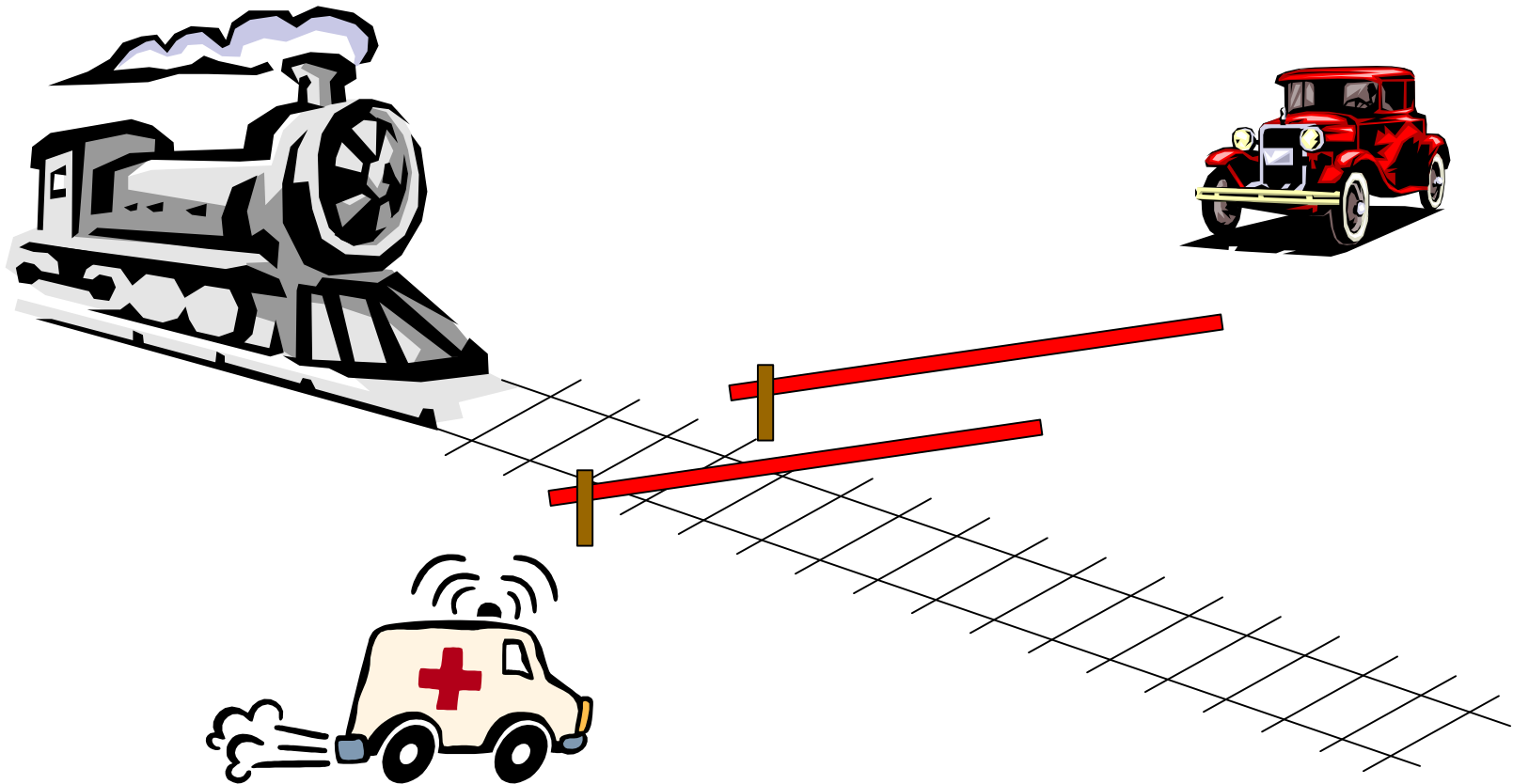
## Challenge 3 : abstraction in verification

- BDDs : reduce the number of variables
  - early quantification
  - multivalued logics
  - optimize / verify bottom-up
- Use other Boolean techniques
  - Stalmark algorithm (Prover)
  - other SAT techniques
  - linear programming techniques

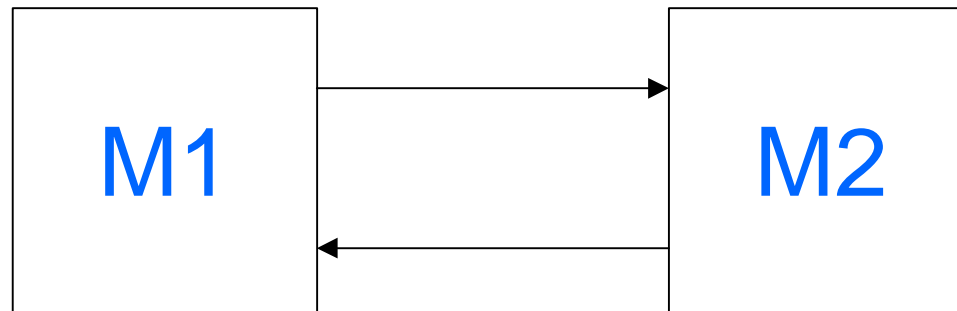


- Data abstraction

## Systems of integer inequations (Polka)



- Compositional abstraction  
assume-guarantee reasoning

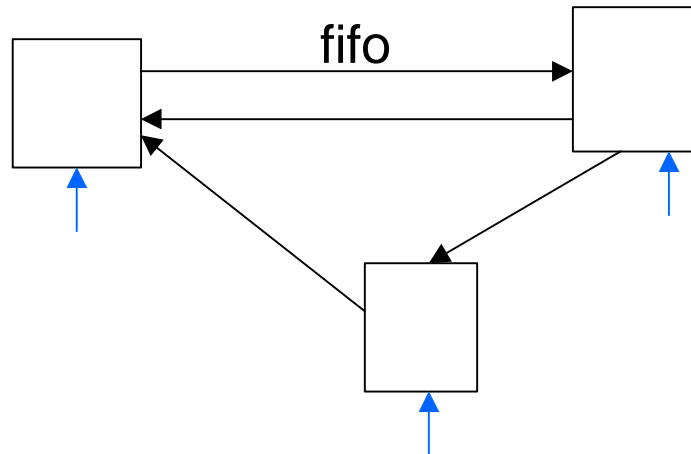


- each of **M1**, **M2** constrains the other
- prove properties of **M1** using an abstraction of **M2**  
+ symmetrical
- use this to prove global properties

easier said than done!

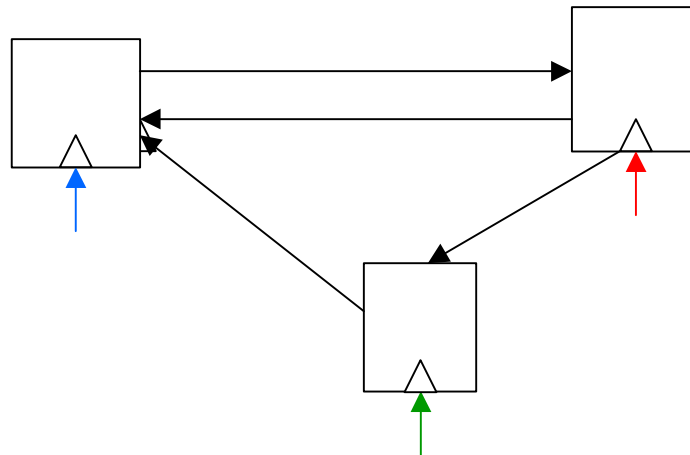
## Challenge 4 : multirate systems

- Single synchronous program, but each computing action is called at some rate : [use this for code distribution](#)



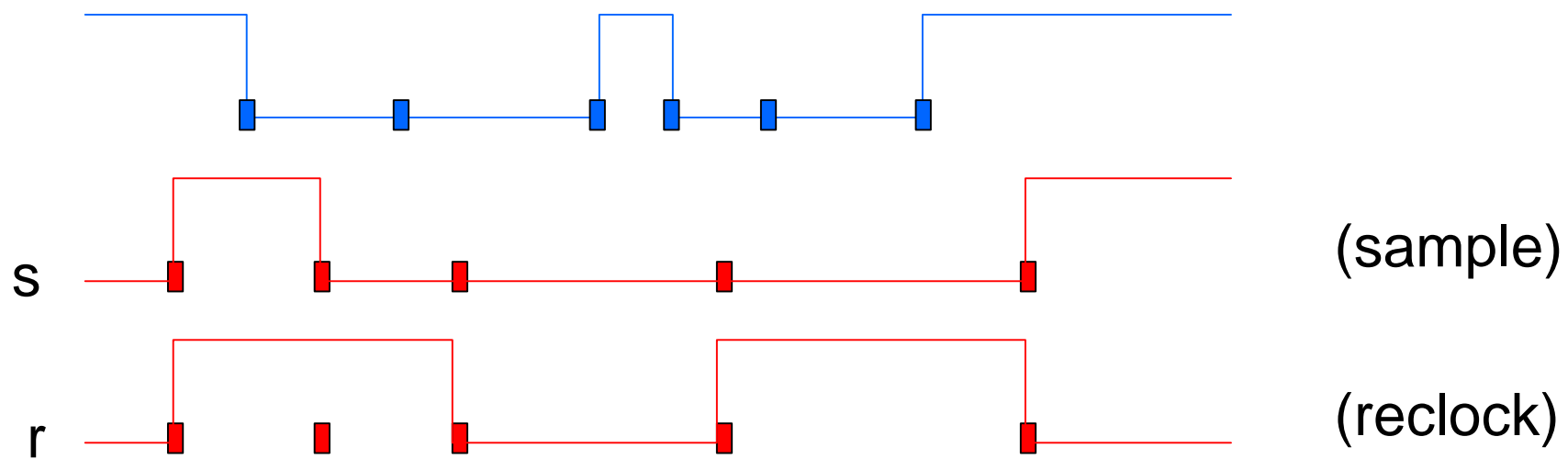
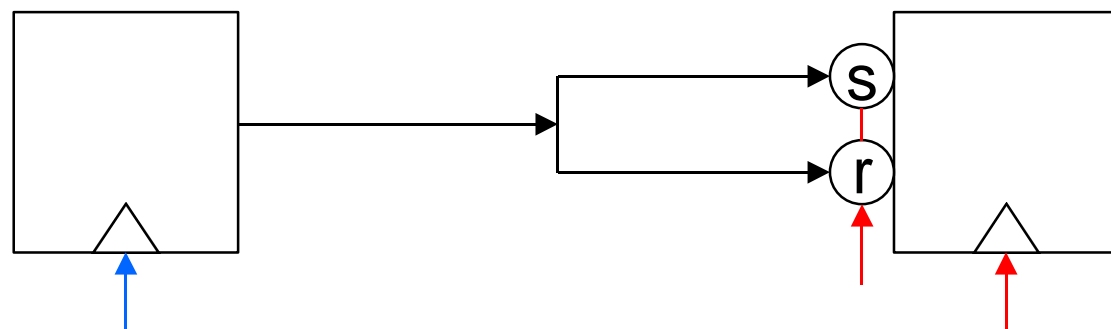
- How to minimize bandwidth?
- How to synchronize?  
[Caspi / Girault, Signal](#)
- Is there a need to synchronize?  
[endochrony / isochrony \(Benveniste, Le Guernic, Talpin\)](#)

## Challenge 4' : multiclock systems



independent  
clock zones  
no fifos

- Distributed Shannon / Nyquist theorem  
[Caspi](#)
- Multiclock Esterel : samplers and reclockers  
[Berry, Sentovich](#) (Cadence)



```
module M :  
  input sample S;  
  abort  
    run N [clock C2]  
  when S
```

```
main :  
  run M [clock C1]
```

M : - samples S on C1  
- starts N on C1  
- aborts N on C1

C2 can be arbitrary, even faster than C1

## Challenge 5 : improving causality analysis

- Classical approach : reject instantaneous feedback

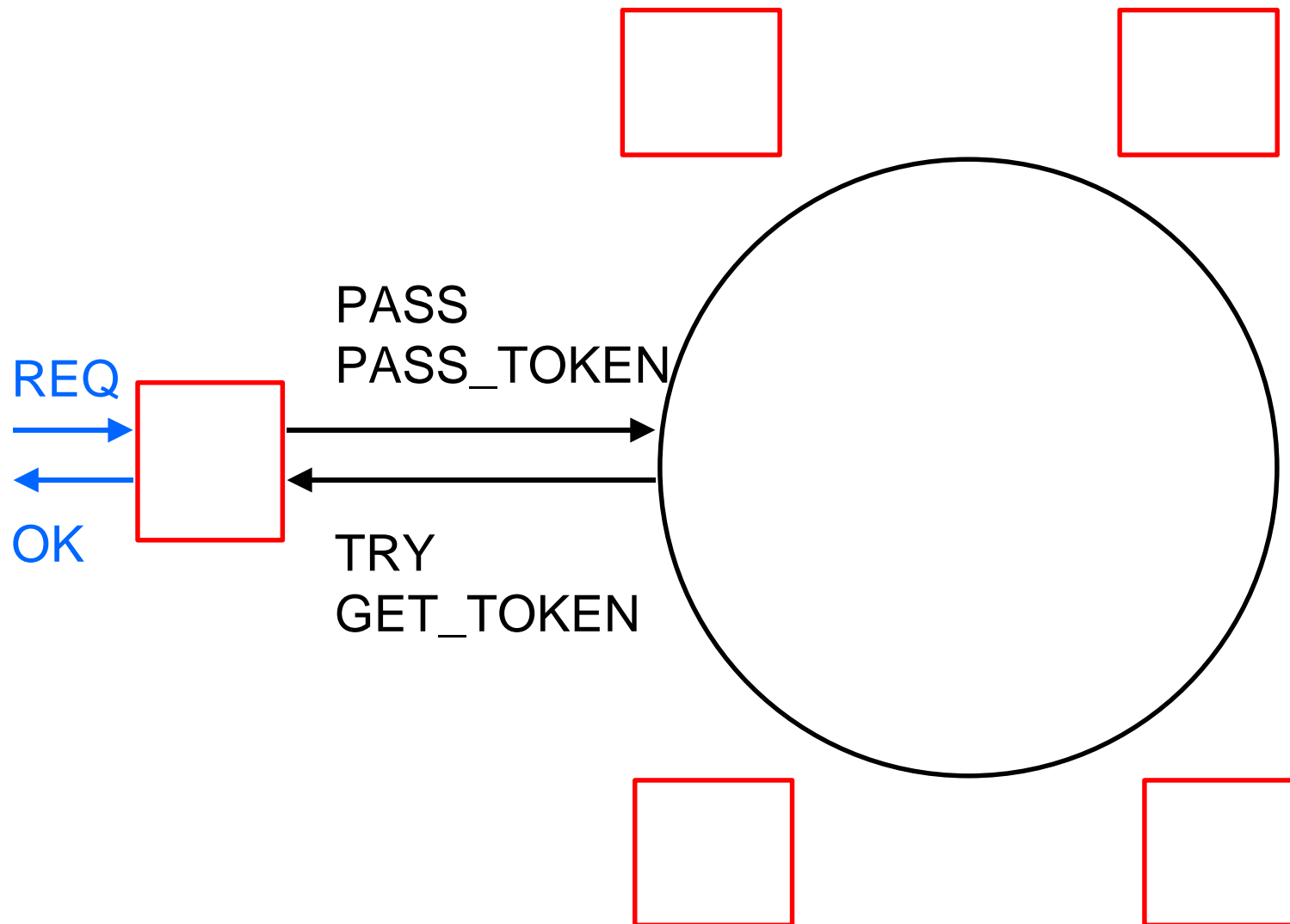
$$\begin{aligned} \cancel{X} &= \cancel{X} \\ \cancel{X} &= \text{not } \cancel{X} \end{aligned}$$

- Esterel approach : accept **good** instantaneous feedback

$$\begin{aligned} X &= I \text{ and } Y \\ Y &= \text{not } I \text{ and } X \end{aligned}$$

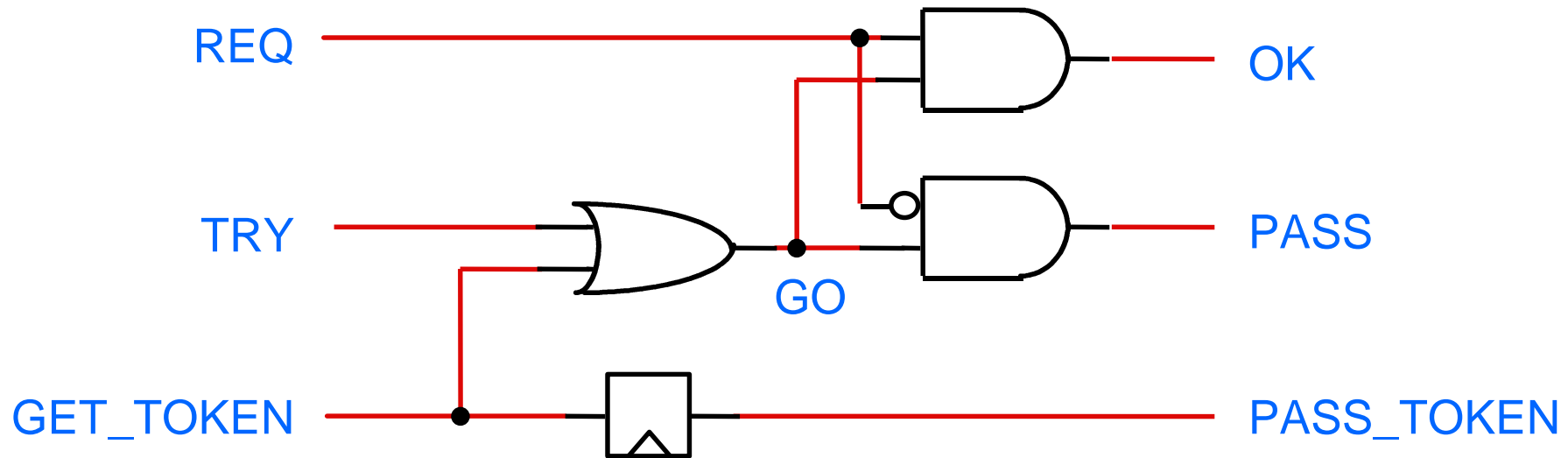
- Symmetric protocols
- operator sharing (exponential saving in size!)

# The Cyclic Bus Arbiter





# Bus Arbiter Station



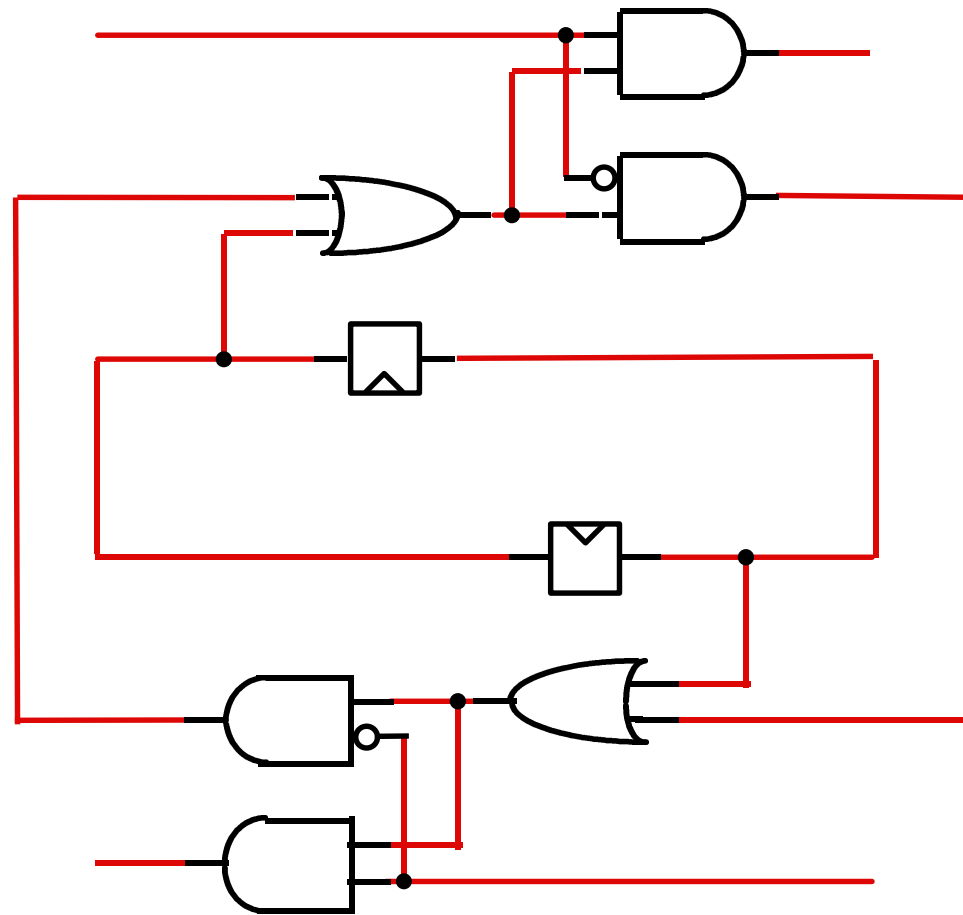
$OK = REQ \text{ and } GO$

$PASS = \text{not } REQ \text{ and } GO$

$GO = TRY \text{ or } GET\_TOKEN$

$PASS\_TOKEN = \text{reg}(GET\_TOKEN)$

# The Cyclic Bus Arbiter



- **Main theorem** : logical characterization of good cycles
- **Esterel v5** : BDD - based algorithms
  - scaling problem
  - uneasy error messages

Can we design nice syntactic restrictions which make it possible to write only good cycles?

Can we design better cycle analysis algorithms ?

# Conclusion

- Synchronous languages are parallel + understandable
- They are based on strong mathematics
- They are used in large (critical) industrial projects
- They need to be unified
- Outstanding efficiency should be attainable
- Verification is good, but one more order of magnitude needed
- Causality diagnostics need to be improved