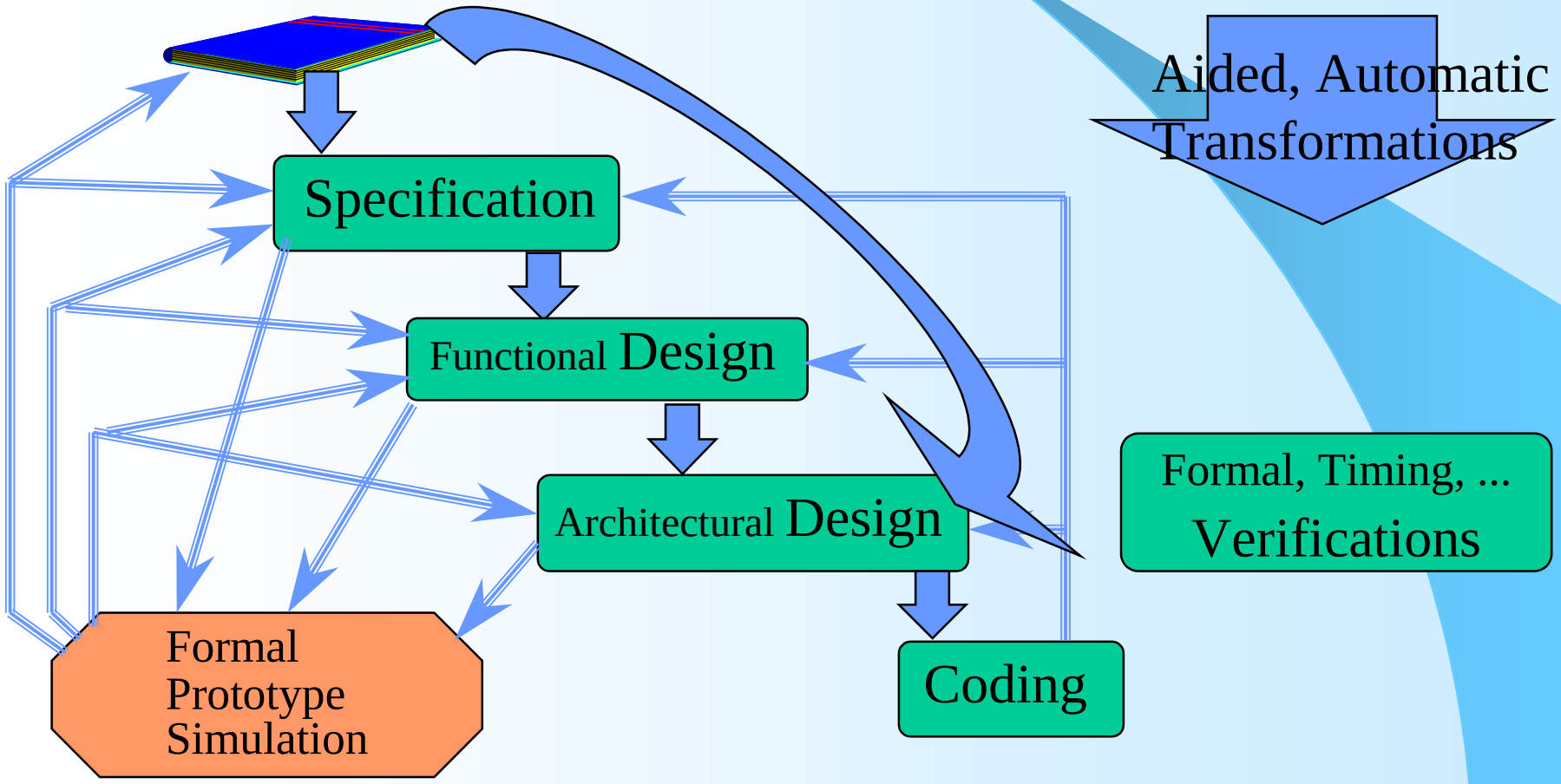
# Embedded Application
# Architecture design

## EP-ATR

Paul Le Guernic, Thierry Gautier
IRISA/INRIA EP-ATR Project

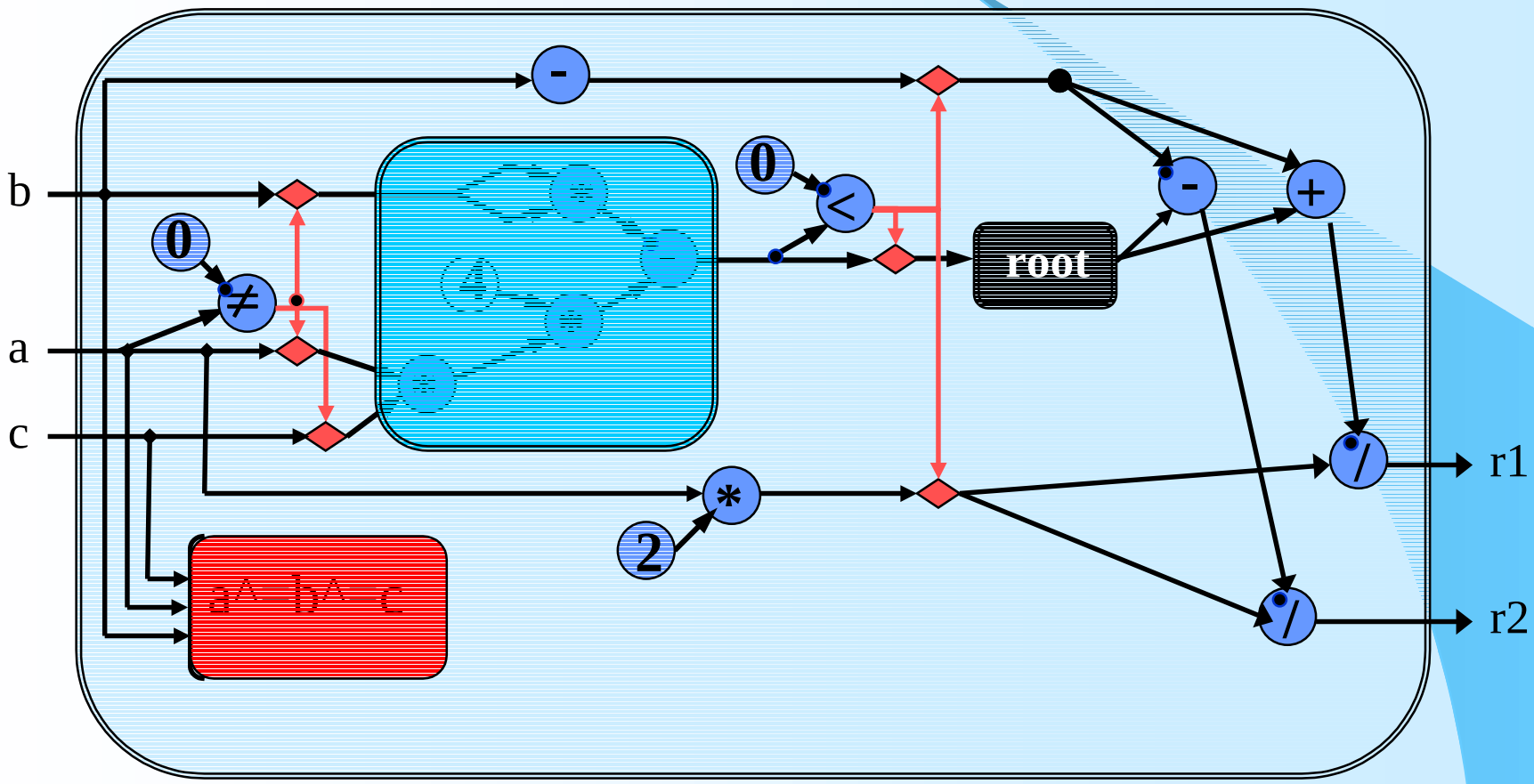
# Polychrony

- Real time and embedded system design



# The core model

## Synchronized data flow



# Contents

- Synchrony vs a-synchrony
- Signal flow model (Relational level)
- Designing with Signal
- Polychrony: functionalities

# Synchrony vs A-synchrony

- Process composition
  - A-synchronous: processes are driven by different a-synchronized clocks, each progressing at its own pace
  - Synchronised: some implicit or explicit synchronization rules are introduced (relations on communications) CCS
  - Synchronous: processes are driven by a single global clock such that an “atomic” action is activated at each tick SCCS

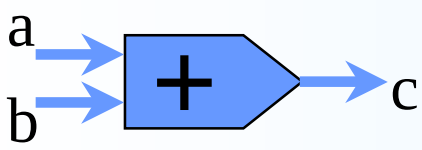
# Synchrony vs A-synchrony

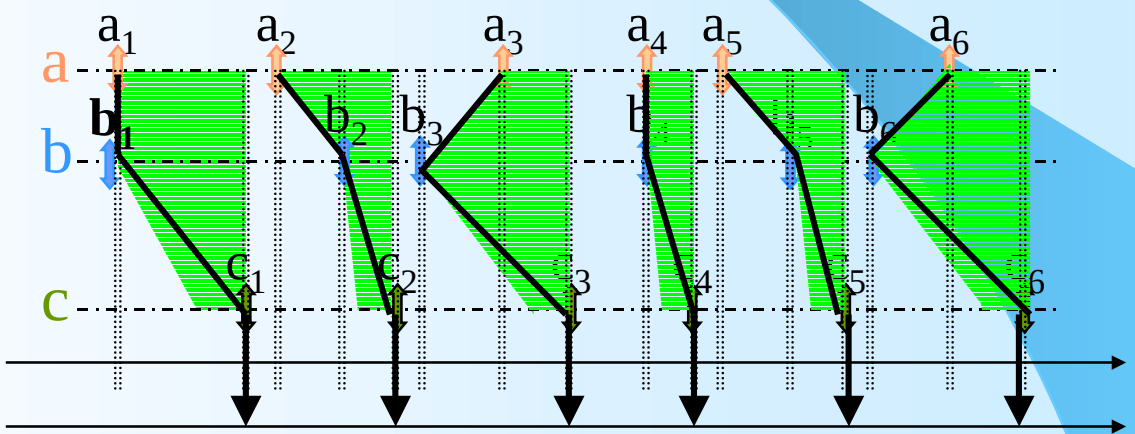
- Communication
  - A-synchronous:asynchronous interaction indicates the fact that data transfer between processes may take **unbounded** amount of time and may need **unbounded** amount of memory to hold sent and unreceived values (interaction points in ESTELLE)
  - Synchronized: some implicit or explicit synchronization rules are introduced: bounded FIFO
  - Synchronous: Communication occurs in a fixed number of ticks of each participant (RdV, signal in VHDL, Lustre, Signal including windows, StateMate,…)

# Synchrony vs A-synchrony

# Virtual discrete time abstraction

- High Level: Virtual discrete time



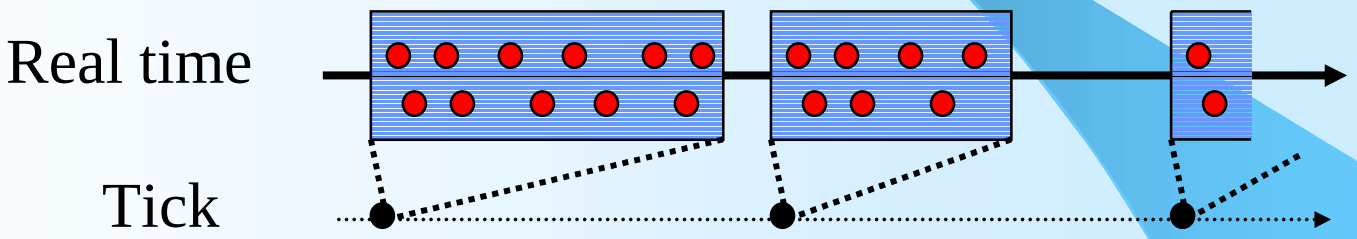

continuous
ordinal

**Reals → Floats**

# Synchrony vs A-synchrony

# Relating virtual discrete time to real time

- Consistency of synchronous hypothesis




  - **Bounded number** of internal actions
    - Lustre, Esterel, Signal: no instantaneous while

- Checking: Needs actual target architecture

# Synchrony vs A-synchrony

# Some synchronous languages Models

- $S_{t+1}=F_s(S_t, X_t)$ $O_{t+1}=F_s(S_t, X_t)$
  - VHDL Synopsis, ...
  - STATEMATE ILogix
- $S_{t+1}=F_s(S_t, X_t)$ $O_t=F_s(S_t, X_t)$
  - Esterel
  - Lustre Scade Telelogic
  - Signal Sildex TNI
  - Silage

# Signal

## a language for the core model

## Fundamental relation

X :=: Y

In this process, for all flows, X and Y have

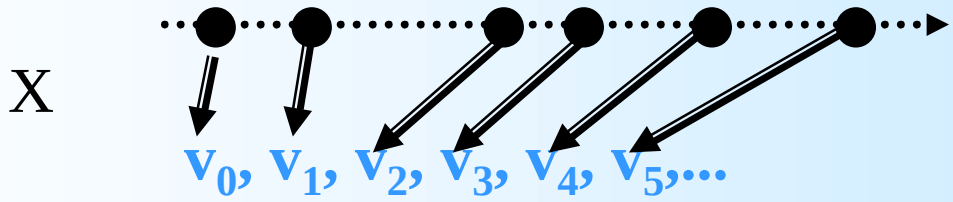
- the same discrete clock
- the same sequence of values

⇒ **Communications are synchronous** in discrete clock

Synchronization can be relaxed

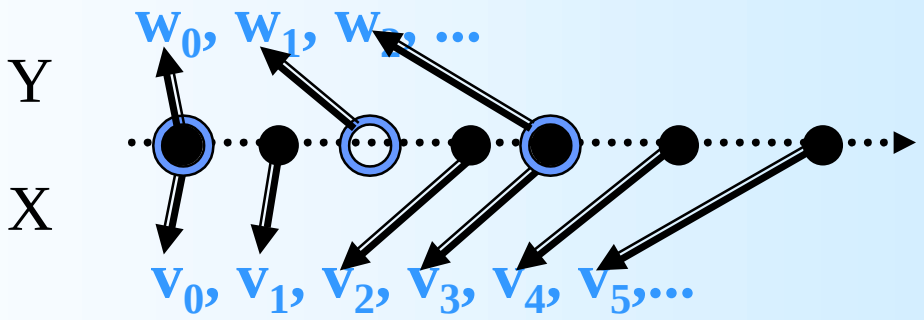
# Signal flow model
# Relational level

- A signal:
  - **a name**
  - **a discrete clock:** a discrete total order, the set of instants
  - **a sequence of values:** a value associated with each instant

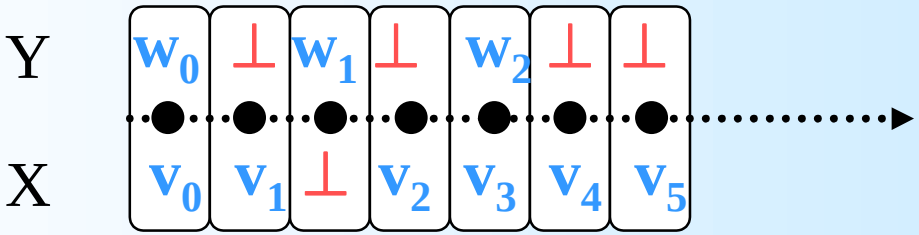


# Signal flow model
# Relational level

- A flow:
  - **a set of signals with distinct names**

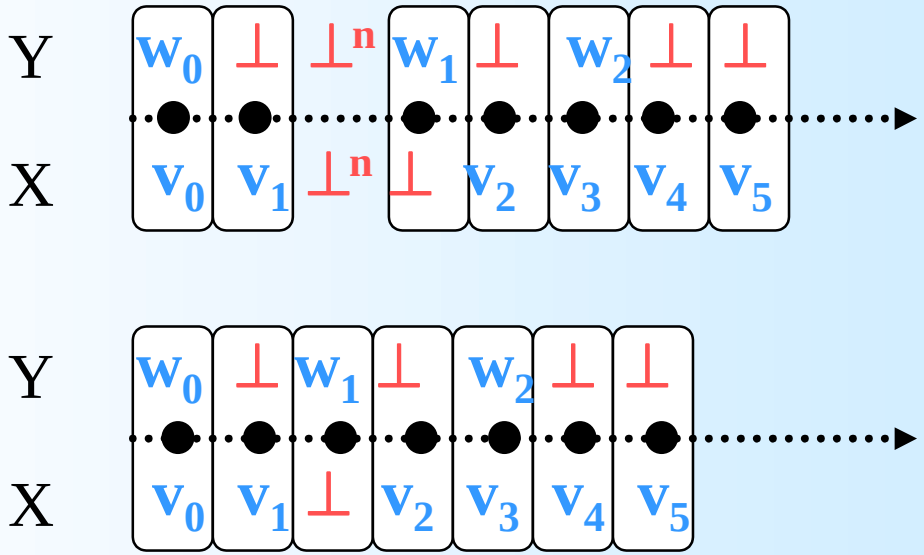



  - Using ⊥ to represent absence: events



# Signal flow model
# Relational level

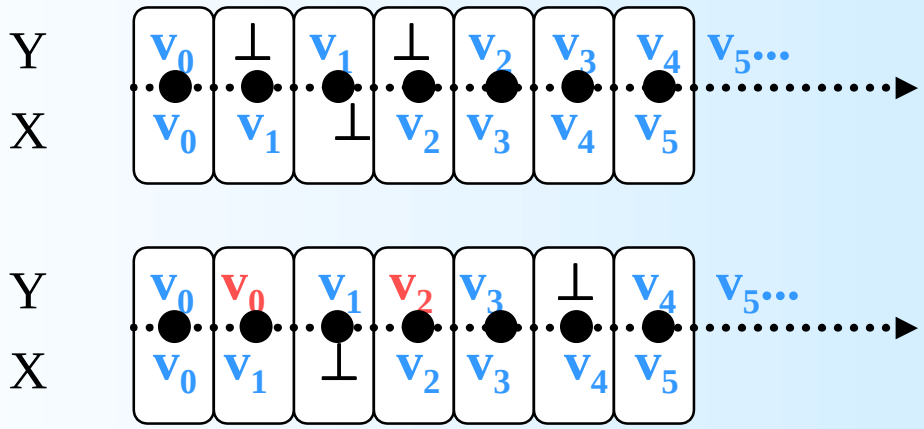
- Traces and flow:
  - **a flow is a compacted trace (silent events removed)**



# Signal flow model
# Relational level

- Process:
  - **a set of flows: a relation between a set of variables, i.e., a specification**

**Each value received in X is sent to Y**



# Signal flow model
## Monochronous functions

$Z :=: X + Y$

$\forall t >= 0\ Z_t = X_t + Y_t$

X ,Y and Z have the same discrete clock: **monochronous**
the sequence of values of Z is the one to one extension of the addition operator

# Signal flow model
## "State" function

$Y :=: X \$ 1 \text{ init } v_0$

$\forall t>0\ Y_t = X_{t-1}$

$Y_0 = v_0$

X and Y have the same discrete clock

the sequence of values of Y is the right-shifted sequence of values of X ($y_0$ is its first value)

$Y :=: X \$ Z \text{ init } V_0$

X and Y and Z have the same discrete clock

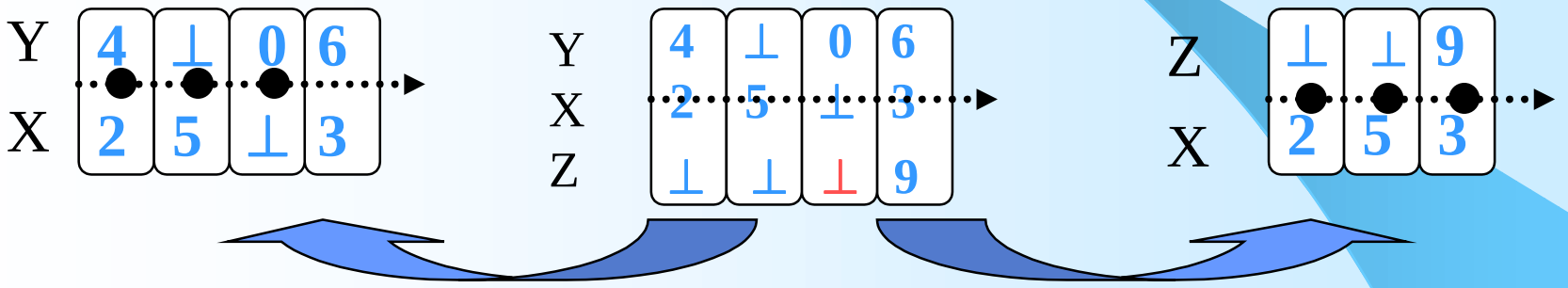
Z is a bounded integer signal (N the bound)

$V_0$ is a vector of N elements

# Signal flow model
## Process composition

$P_1 \mid P_2$

**The set of flows that satisfy both P1 and P2**

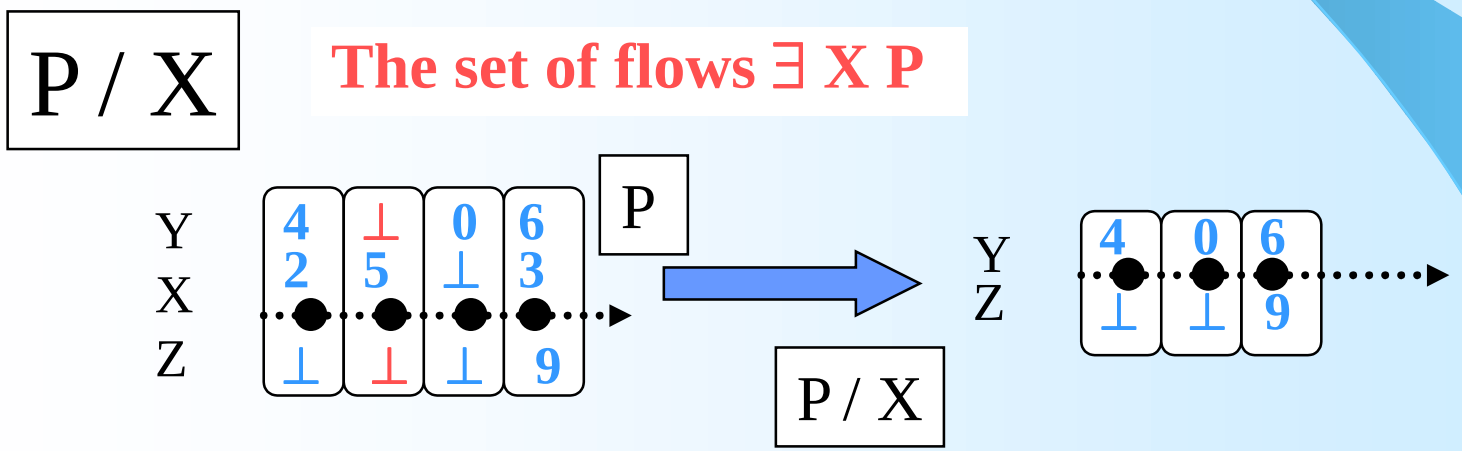



⇒ Synchronisation results from common signal name composition is **asynchronous** (or more exactly, truly parallel)

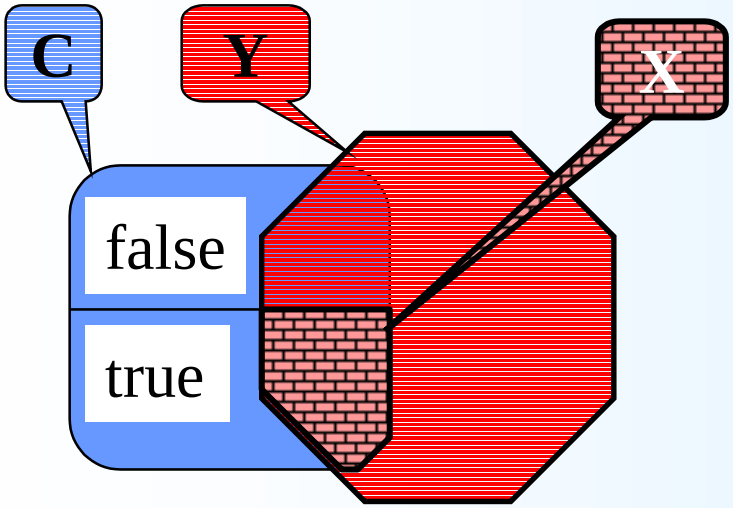
```
X :=: Y
| Z :=: U
```

Has 2 distinct, mutually unconstrained clocks

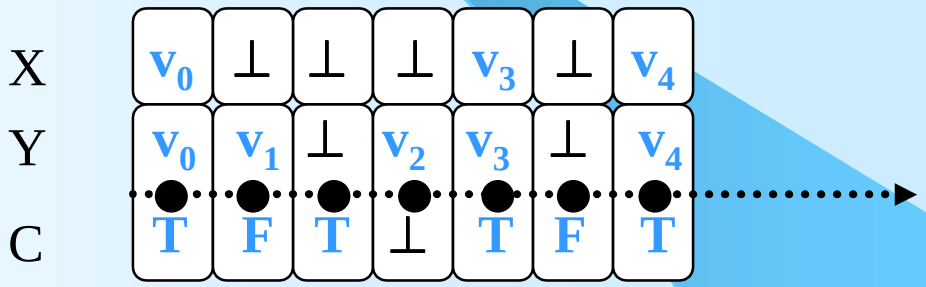
# Signal flow model
## Variable abstraction



# Signal flow model
## Signal extraction




X := Y when C

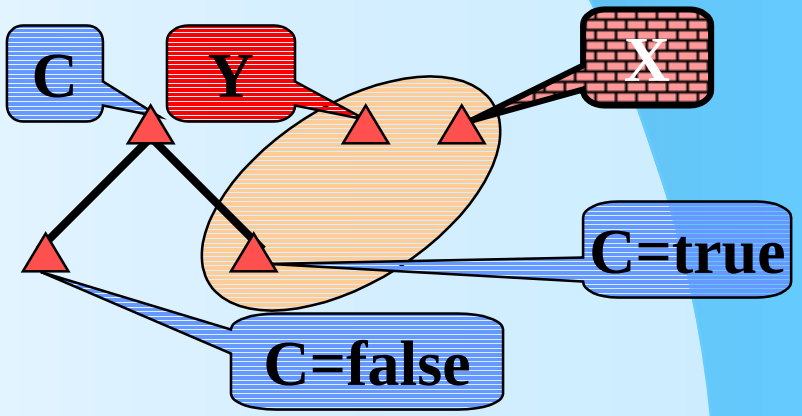



Polychrony: five clocks

3 basic

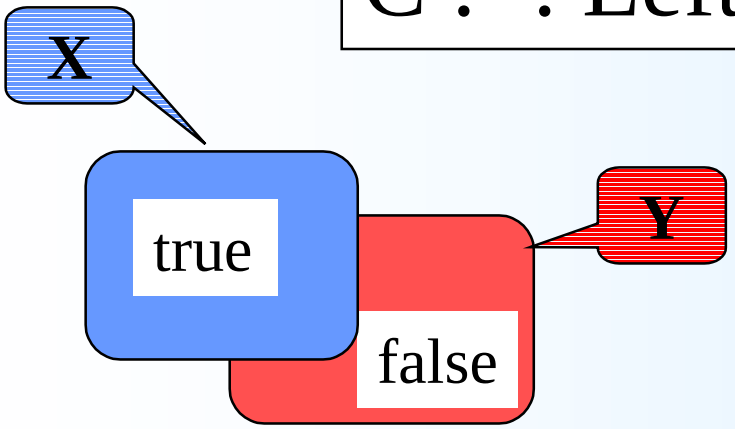
an endochronous subset

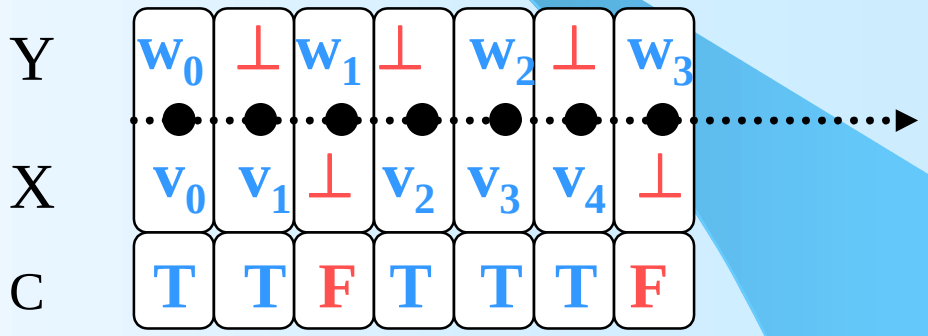
+ Clock constraints



# Signal flow model
## Boolean/clock basic relations

C :=: LeftPresent(x,y)




One can then define the event clock of X as

H :=: LeftPresent(X,X)

H :=: ^X

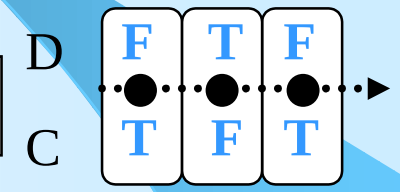
# Signal flow model
# Example: default definition

Z :=: X default Y

Can be used to specify prioritised scheduling

(|X :=: Z when C
 | Y :=: Z when not C
 | C :=: LeftPresent(X,Y)
 | Z ^= C |) / C

D :=: not C

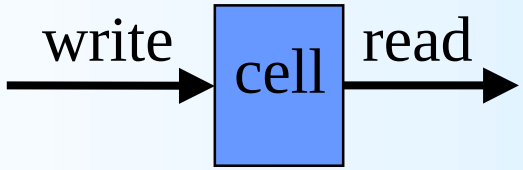



Where Z ^= C (Z and C have the same clock) is defined as

(|D :=: LeftPresent(Z,C)
 | D :=: LeftPresent(C,Z)|) / D

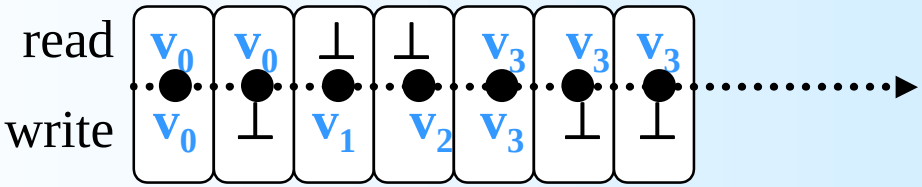
# Signal flow model

## an asynchronous/synchronous interface




```
(|write :=: CELL when ^write
 | read :=: CELL when ^read
 | CELL :=: (CELL $ 1 init v) when not LeftPresent(write, read)
 | CELL ^= (^write) default (^read) |)/CELL
```

write and read have two independent discrete clocks



# Signal flow model
# Plant specification

- Asynchronous behaviour can be described
- Non deterministic input-output behaviour ?

```
(| N :=: (N$1 init 0) +1
 | OUT :=: N when ^OUT
 |)/ N
```

N has a discrete clock which is not visible outside

⇒ OUT holds any increasing sequence of integers

# Signal flow model
# Back to as/s interface

- An interface must be defined between the synchronous program and its (a)synchronous context
- **Esterel, Statemate:**
  - no way to specify environment behaviour properties in a system (may only be documentary)
  - ⇒ standard implicit interface to detect signal occurrences
- **Signal:**
  - one can specify abstract properties on environment (x and y exclusive,… any signal program)
  - ⇒ interface is implemented as a specific protocol
  - **⇒ behaviour can be defined s.t. it remains independent of implementation architecture: endochrony**

# Signal flow model
# Back to as/s interface

- Let a property such that if **x>=0** then **y** is present
  - Esterel, Statemate: one could have a contradiction (Scade) due to global a priori sampling of inputs
  - Signal: after getting **x** and knowing that x is positive then we will eventually get **y**
  - but when x is not positive or is absent what about y?

  To answer this question in a coherent way independently of implementation architecture, y is associated with an explicit boolean clock, that can be shared by different signals

# Clock calculus

- Functions:
  - Structures the control of application
  - Solves  synchronization constrainsts
- Based on:
  - [C] ^+ [not C] = ^C,  [C] ^* [not C] = 0
  - Returns a forest of clock hierarchies
  - Uses BDD package (Berkeley, Sigali)

# Signal flow model
# Clock system

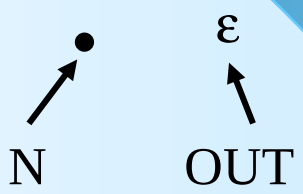
- **A clock system** on a process P defined on the variable set A is a function clk:A→ A∪{ε,•} satisfying: for any flow F in P and variable x belonging to A clk(x) ≠ ε implies that

x is present in an event of F if and only if clk(x) is true in this event (• represents the fastest clock)

- A clock system on a process P is endochronous if and only if clk(A)= A∪{•}

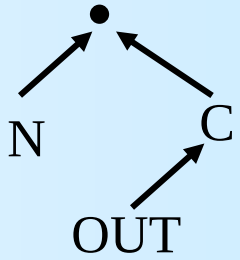
# Signal flow model
# Clock system: example

```
(| N :=: (N$1init 0) +1
 | OUT :=: N when ^OUT |)
```




Endochronous completion

```
(| N :=: (N$1 init 0) +1
 | OUT :=: N when ^OUT
 | C :=: LeftPresent(OUT,N) |)
```



# Signal flow model
# Clock system: Compiler

- The “clock calculus” solves clock equations to get the maximal endochronous sub systems
- It generates a proof obligation when it fails to solve a constraint

```
C :=: x>0
| C :=: ^C
```

Needs to prove that x<=0 never occurs

# Signal flow model
# Clock system: Compiler

- **Clock system** is the fundamental tool for many transformations
  - A function adds supplementary boolean variables to get an endochronous system
  - a function computes the clock abstraction of a process (the clock system of its interface)
  - the structure of the clock system can be used for task generation

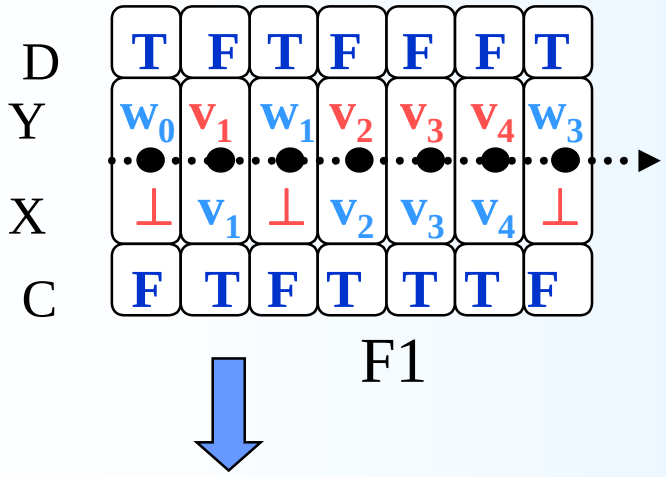
# Signal flow model

## Clock system: some available functions

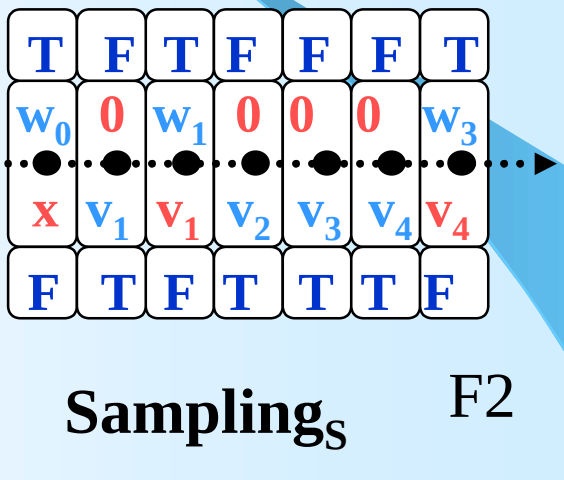
- supplementary boolean variables to get an endochronous system
- Flattening expansion: hierarchy of clocks is reduced to one level (clock(clock(x))= •
- clock abstraction of a process (the clock system of its interface)

# Signal flow model

## Refinement with Clock system

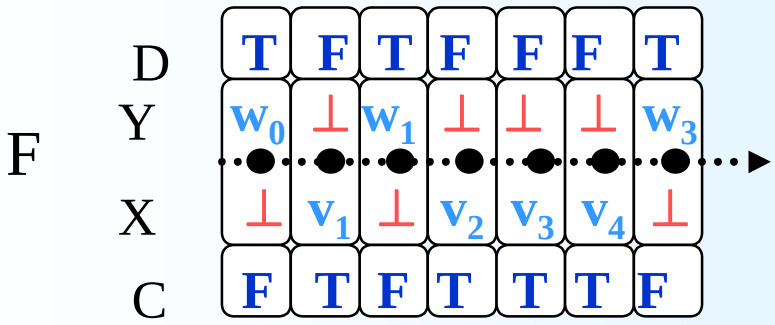



F1




F2

**Sampling$_S$**




F1 and F2 implement F

- -multiplexing (F1)
- -constant valuation (F2:Y)
- -memorization (F2:X)

# Designing with Signal
## partial order definition

- **Needed: some behavioural aspects**

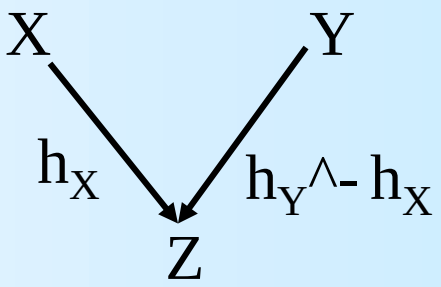
$X \xrightarrow{h} Y$

- **Definition of**

$Z := X$

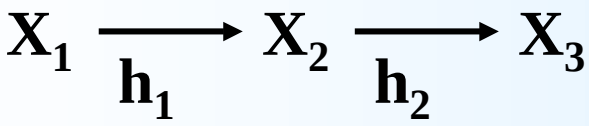
$Z :=: X \mid X \xrightarrow{\wedge x} Y$

$Z := X \text{ default } Y$



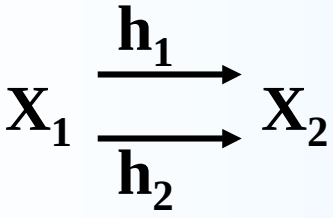
# Designing with Signal Graph properties

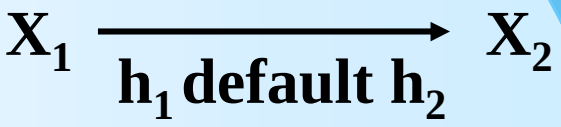
- **Sequence**





- **parallel**



# Designing with Signal Graph properties

- **Circuit detection**

$$X_1 \xrightarrow[h]{} X_1$$

- **Process abstraction:** on external signals
  - Clock equation
  - Graph abstraction
- **Optimizations**
  - needed clock
  - assignment clock
  - modification clock

# Polychrony

# Some functionalities

# State transformations

- Event/Boolean conversion
- Modifying interface for event signals
- Flattening expansion: hierarchy of clocks is reduced to one level
- Boolean State variables: defined at the master clock
- Cycle detection

# Graph normalization

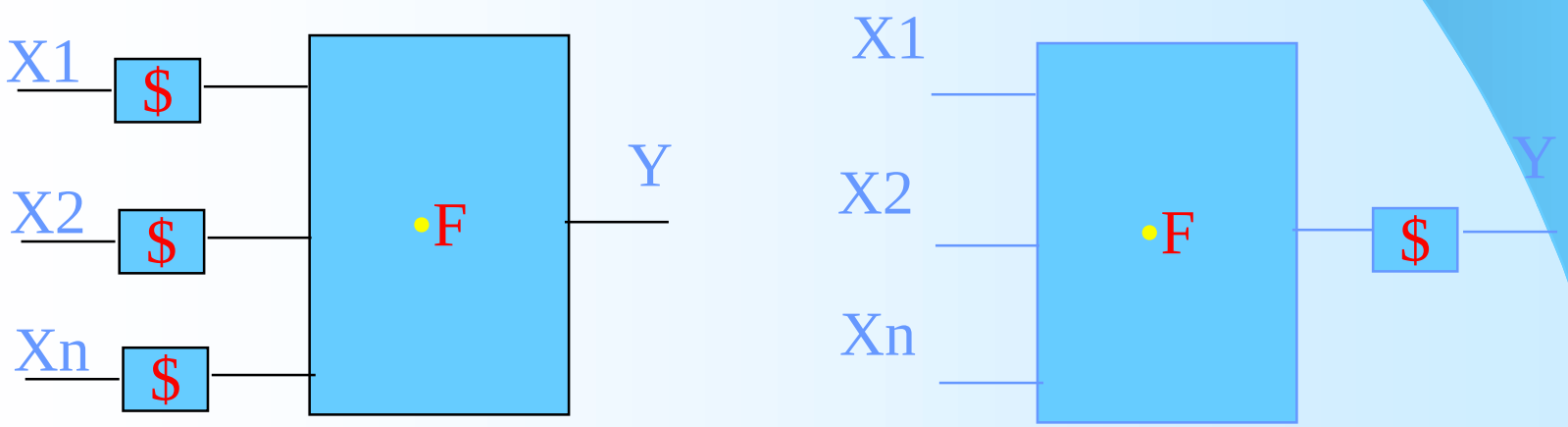
- Function:
  - Unifying signal defined by the same expression
  - Solving constraints
- Based on:
  - Rewritting of Boolean signal expressions

# RETIMING

- Minimization of the state variables
- Solving constrainsts

  UPSTREAM retiming

  DOWNSTREAM retiming



# PARTITIONING

- User partitioning
- Input  directed partitioning
- Output directed partitioning
- Control/Calculation partitioning
- State variable/other signals partitioning

# USER PARTITIONING

Reorganization

- Using PRAGMA: RunOn
- Method returns
  - a set of subgraphs using the value of the pragma
  - Clocks are assigned to subgraphs
  - The interface of each subgraph is built

# I/O directed PARTITIONING

Reorganization

- Definition:
  - X,Y are in the same set iff they depend on the same inputs
- A Program P is rewritten in
  - (| (| P1 | P2 | ….|Pn |) | SCHEDULER() |)
- How?:
  - a set of subgraphs is built, clocks are assigned to subgraphs, interface of each subgraph is built
  - The graph of the node calls is built: SCHEDULER

# CONTROL/CALCULATION partitioning

Reorganization

- A Program P is rewritten in
  - (| Pcontrol() | Pcalculations() |)
    - Pcontrol: contains clocks and boolean definitions
    - Pcalculations: contains numerical definitions
- How?:
  - Two subgraphs are built according on types of signals
  - Clocks are assigned to subgraphs
  - The interface of each subgraph is built
  - The graph of the node calls is built

# STATES/OTHERS Partitioning



- A Program P is rewritten in
  - (| P_states() | P_others() |)
    - P_states: contains memorizations definitions
    - P_others: contains others definitions
- How?:
  - Two subgraphs are built according on the criterion
  - Clocks are assigned to subgraphs
  - The interface of each subgraph is built
  - The graph of the node calls is built

# INTERFACE SYNTHESIS

Abstraction

- IO dependences
  - Transitive Closure reduced to I/O

    X --> Y at Hi
- Clocks Projection
  - I/O signal clocks
  - I/O Dependence clocks

# OPTIMIZATIONS



- Mutliplexing  delayed variables
  - (| ZX := X $ 1  |  ZZX := ZX $ 1 |)

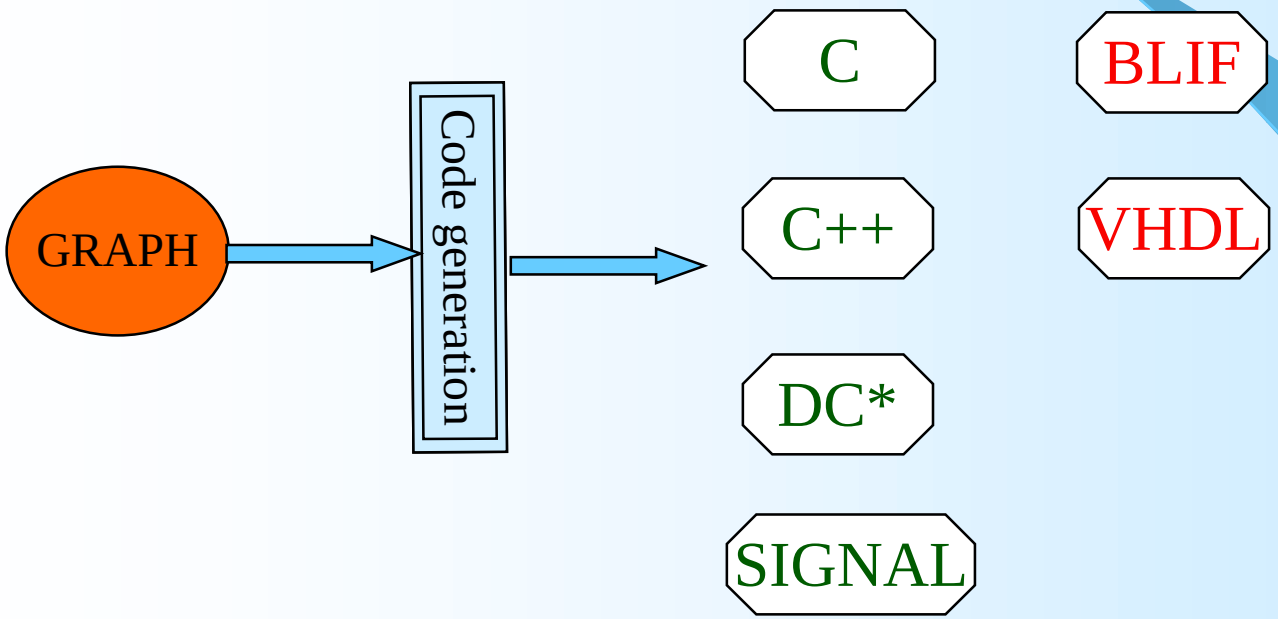
    can be implemented using an array of 3 elements without copy
- Multiplexing by signal aggregation
  - x := y when c
  - x and y can be implemented in one variable

# OPTIMIZATIONS



- Using clock calculus
  - Exclusive clocks
  - Utility clock of signals
  - Assignment clock of signals
  - Modification clock of signals

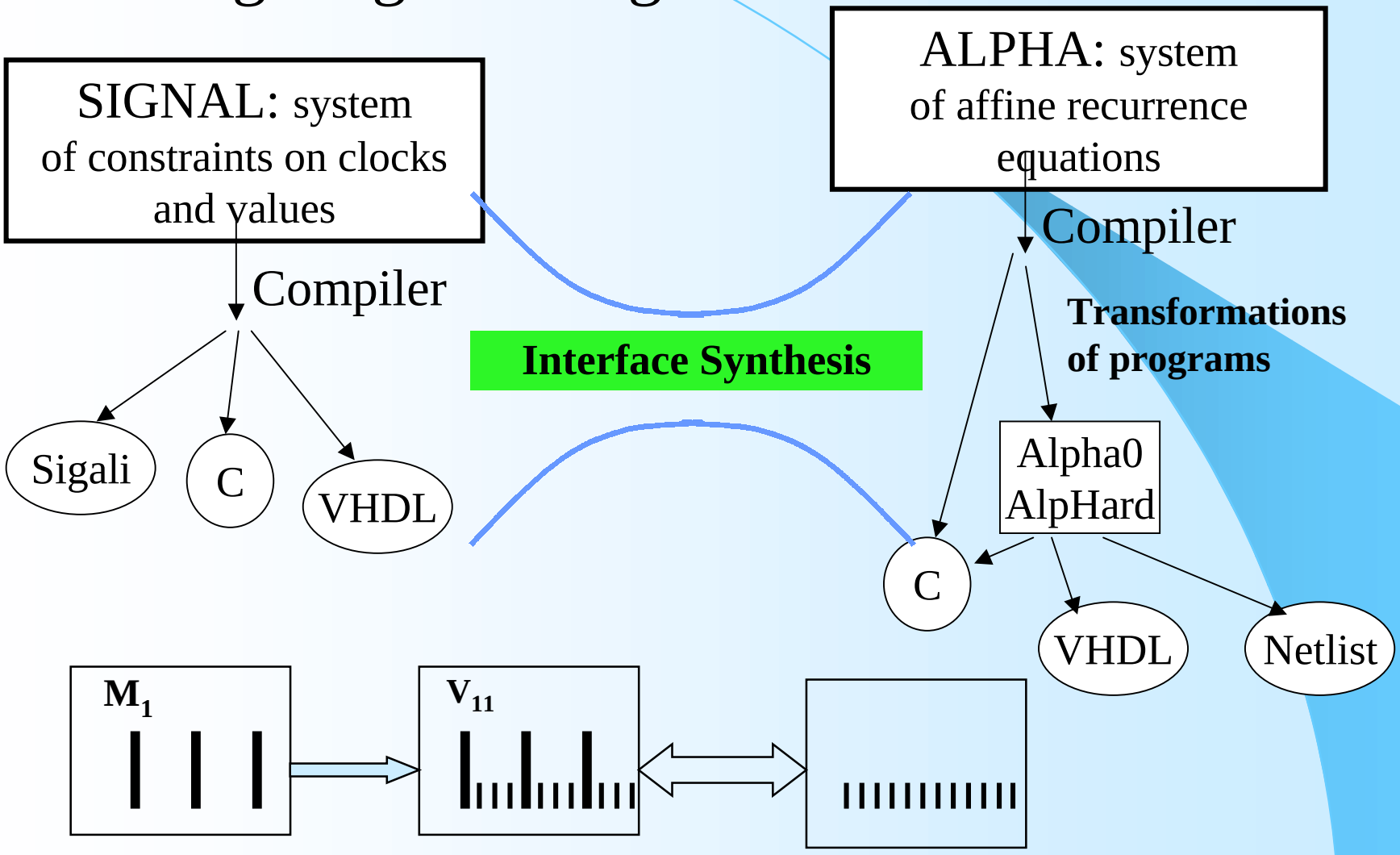
# CODE GENERATION



# Formal Verification

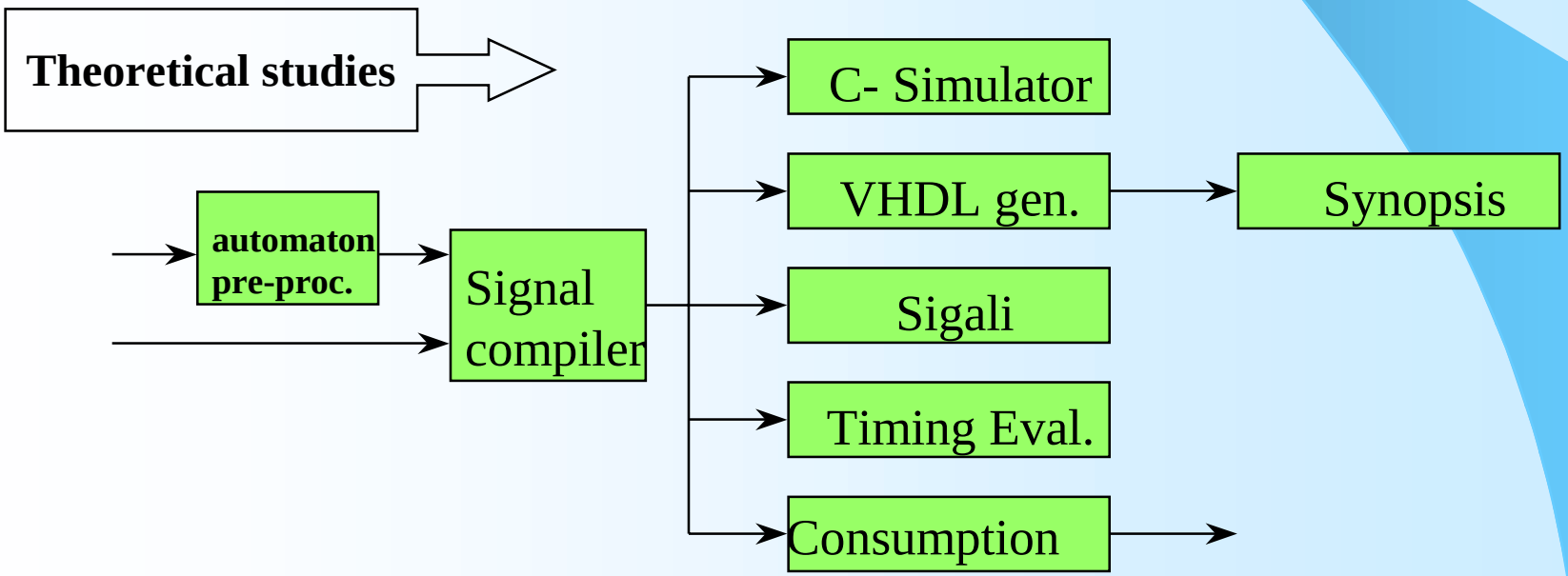
- Sigali and automata
  - Translation automata CADP
  - Studying correspondence Sigali μ-calculus

- Codesign regular/irregular



- Temporal Interpretation
  - P a program
  - A a model of target architecture
    - **Set of generators**
  - we define $T_A(P)$ model of time consumption of P on A
  - Simulator P| $T_A(P)$
- Verification of duration constraints
  - max-plus Algebra

- Hardware synthesis
  - Interpretation of absence as *don't care*
  - Clock synthesis
  - Exploitation of exclusions (clock calculus)



# Perspectives

- à la GNU-licensed version of Polychrony
- Software architecture
- Static properties
  - abstract interpretation (intervals in BDD)
- Validation
  - validation of each transformation
  - verification of the trace of their application
- Test generation
  - using controller synthesis techniques

# Complements

# INTERFACE SYNTHESIS :example

Abstraction

```
process PX=
      ( ? integer a, b, e;
          boolean c;
        ! integer y )
      (| a ^= b
       | e ^= when d ^= c
       | d := a>b
       | y := e when c when d |)
where boolean d;
end


process PX_ABSTRACT =
      ( ? integer a, b, e;
          boolean c;
        ! integer y;
          boolean d;
      )
      spec (| (| a --> e
               | b --> e
               | a --> c
               | b --> c
               | a --> y
               | b --> y
               | e --> y
               | c --> y
               | a --> d
               | b --> d
               |)
             | (| a ^= b ^= d
               | when d ^= e ^= c
               | when c ^= y
               |)
             |)

%PX_ABSTRACT%;
```

{a,b,d}

|

[d] {e, c}

|

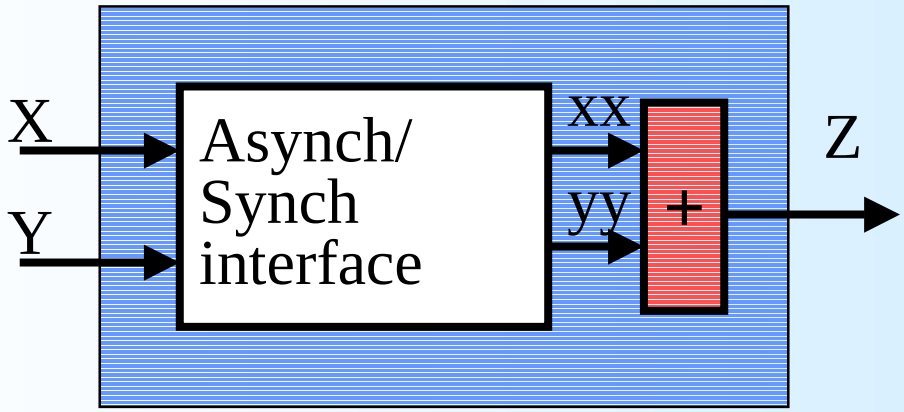
[c] {y}

d becomes an output

# Signal flow model
# De-synchronised functions

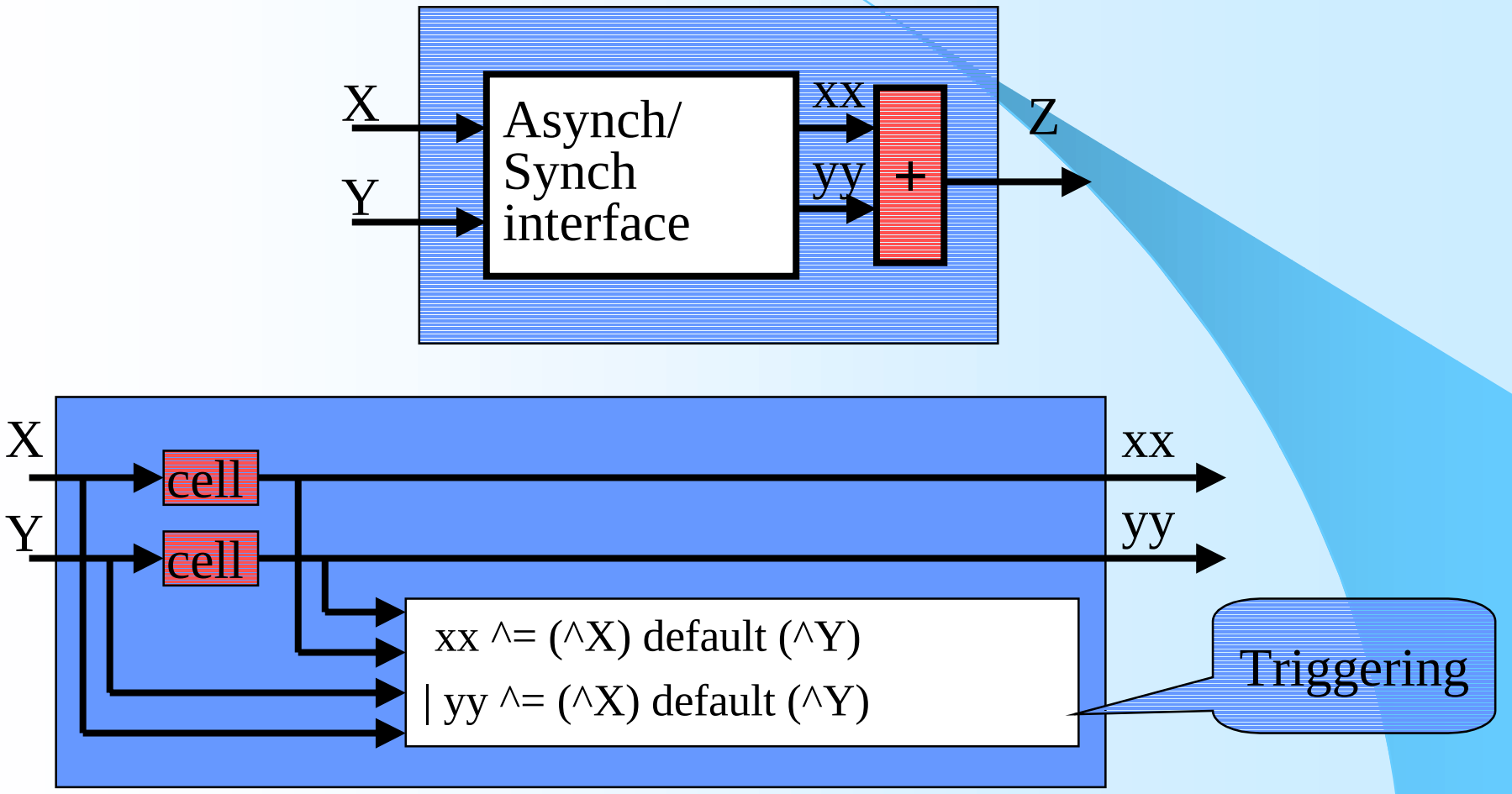
Z :=: X ~+ Y

An **asynchronous** (triggered) version of Z :=: X + Y is defined as

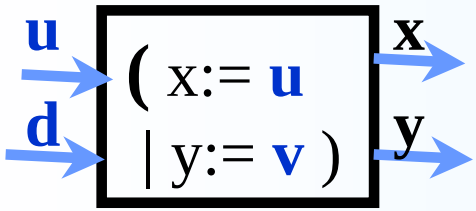


# Signal flow model

## an asynchronous/synchronous interface



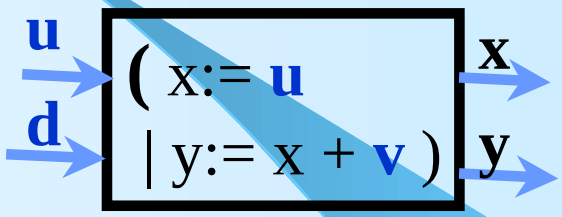
# Synchronicity in a single task




**Signal**: x and y are independent

**Lustre**: x and y are always synchronous

**Statecharts**: $x_{t+1}$ and $y_{t+1}$ are simultaneous when this action belongs to a fired transition



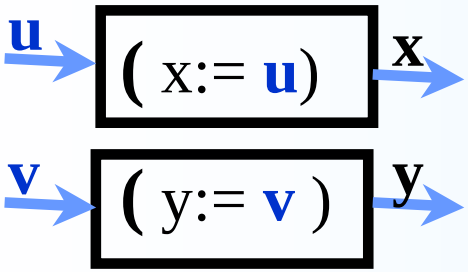

**Signal**: x and y are synchronous

**Lustre**: x and y are always synchronous

**Statecharts**: $x_{t+1}$ and $y_{t+1}$ are simultaneous when this action belongs to a fired transition

(but with a different value)

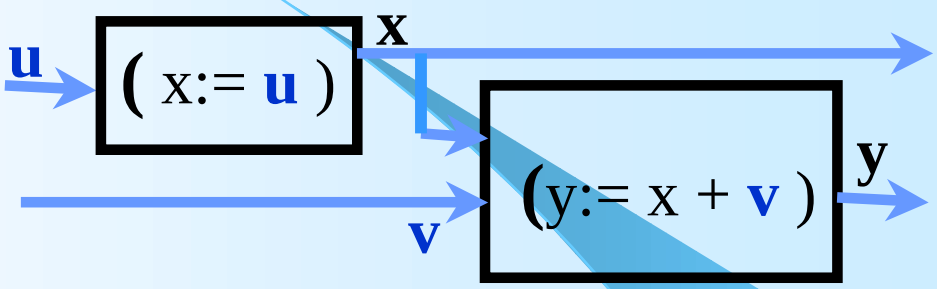
# Synchronicity in multiple task




**Signal**: x and y are independent

**Lustre**: x and y are independent

**Statecharts**: x and y are independent

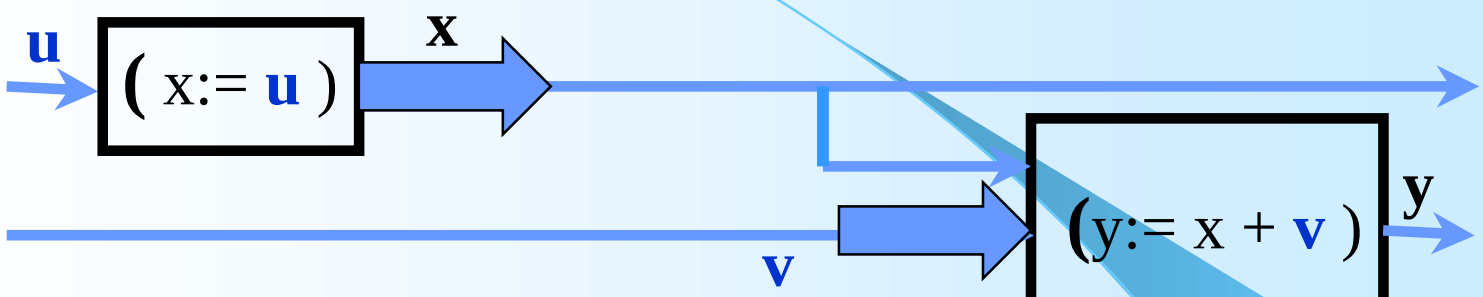



**Signal**: x and y are synchronous

**Lustre**: x and y are synchronous

**Statecharts**: $x_{t+1}$ and $y_{t+1}$ are simultaneous when this action belongs to a fired distributed transition (but with a different value)

# Relax Synchronicity
## when needed




**Signal**: x and y are no more synchronous but the flow semantics is preserved

**Lustre**: cannot be described

**StateChart**: ?