

Lectures on Desynchronization

Albert Benveniste, Benoît Caillaud, and Paul Le Guernic

IRISA

September 2003

1. Synchrony vs. asynchrony

2. Synchronous/Asynchronous Transition Systems

3. Preserving semantics: Endochrony, Isochrony

4. Discussion

SYNCHRONY

1. programs progress via **reactions** : $P = R^\omega$

$$\begin{aligned} \text{run}(P) &= \text{sequence of tuples of events} \\ &= \left\{ (x_i(1))_{i=1,\dots,k}, (x_i(2))_{i=1,\dots,k}, \dots \right\} \end{aligned}$$

2. Within a reaction, decisions can be taken based on testing for the **absence** of some signals (denoted by “ $x = \perp$ ”):

```
present S else 'stat'  
y = current x  
y := u default v
```

3. communication : instantaneous broadcast

$$P_1 \parallel P_2 \stackrel{\text{def}}{=} P_1 \cap P_2 = (R_1 \wedge R_2)^\omega$$

ASYNCHRONY

1. **no reaction**, no global clock, no global state :

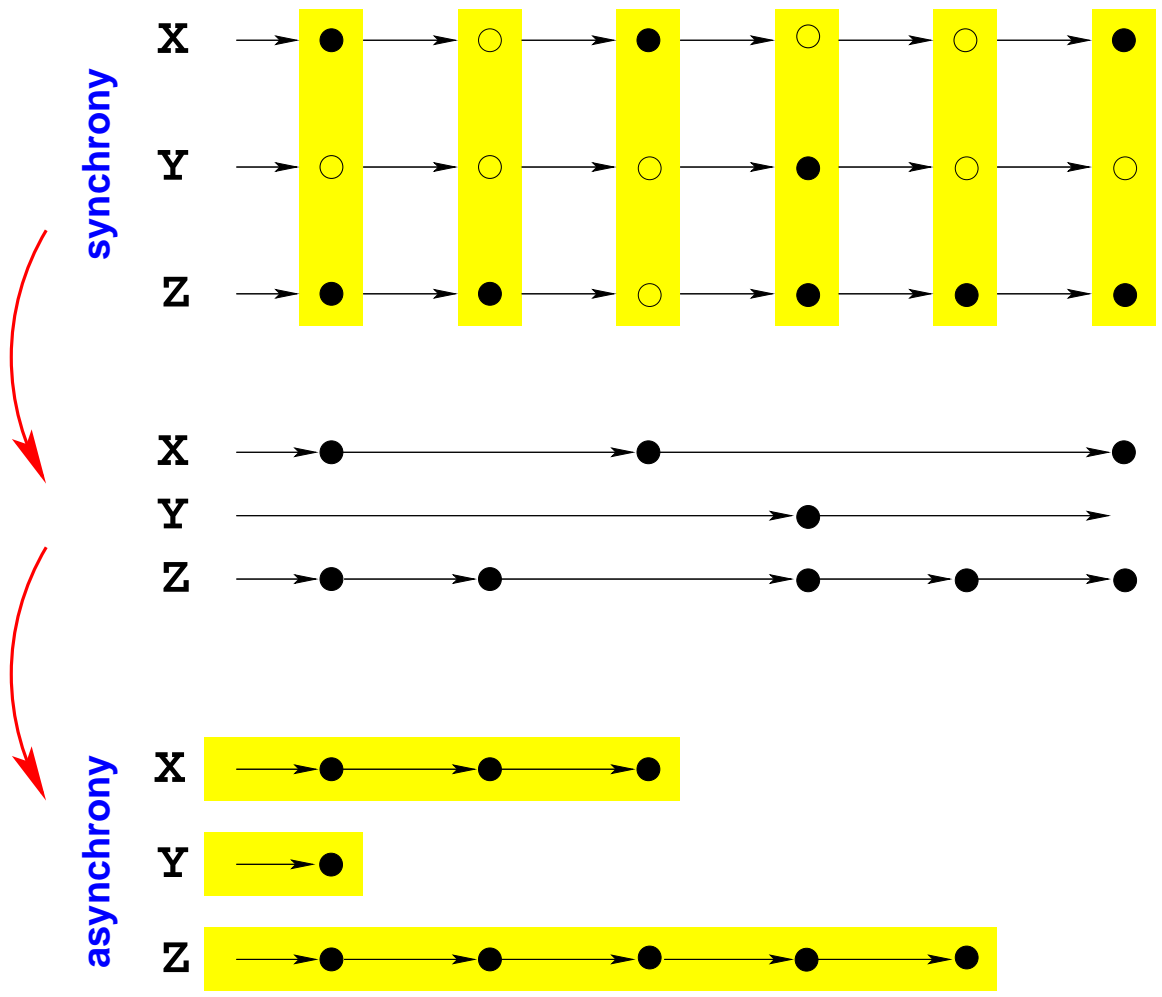
$$\begin{aligned} \text{run}(P) &= \text{tuple of sequences of events} \\ &= (\{x_i(1), x_i(2), \dots\})_{i=1, \dots, k} \\ &\triangleq \text{tuple of } \mathbf{channels} \end{aligned}$$

2. **absence cannot** be sensed and has no meaning ($\neg[x = \perp]$ holds).

3. communication : channel-per-channel unification

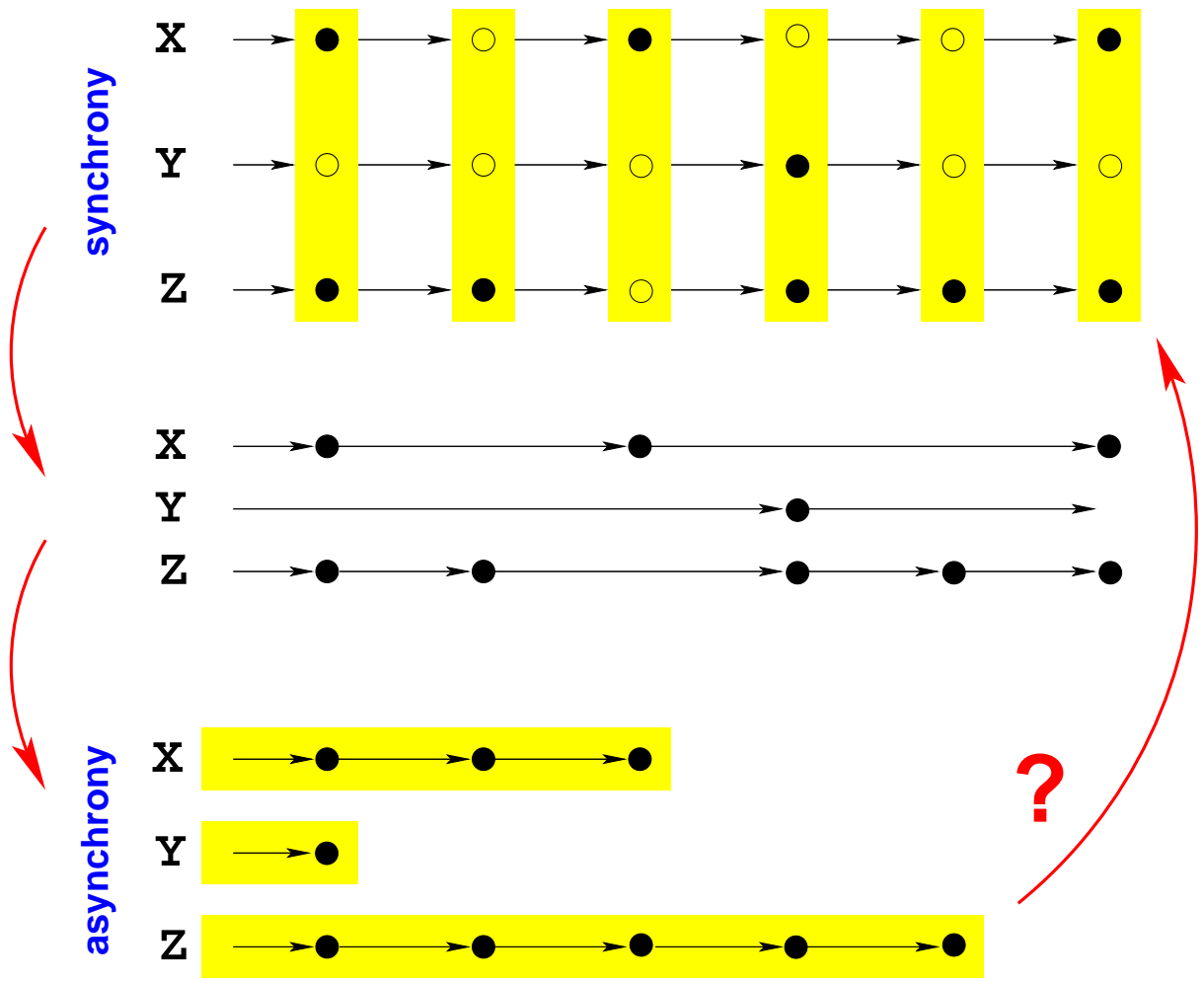
$$P_1 \parallel P_2 \triangleq P_1 \cap P_2$$

relaxing synchrony, informal definition



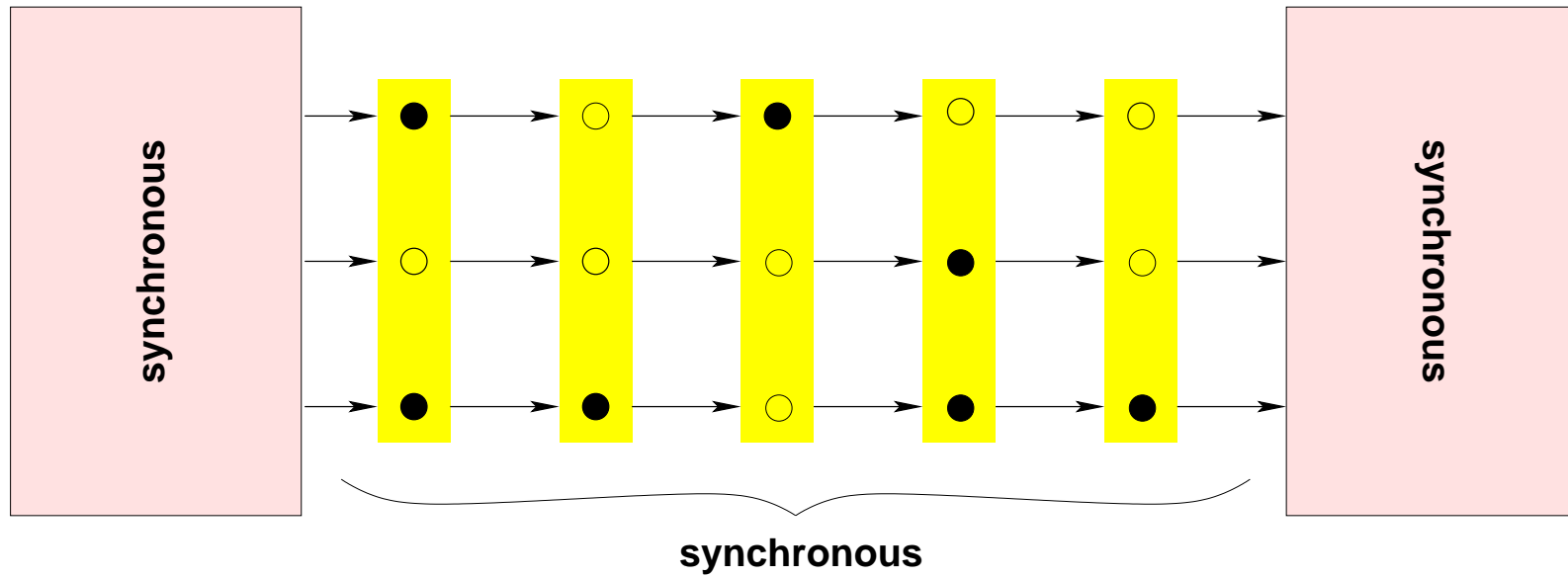
for systems *and* communications

relaxing synchrony, informal definition

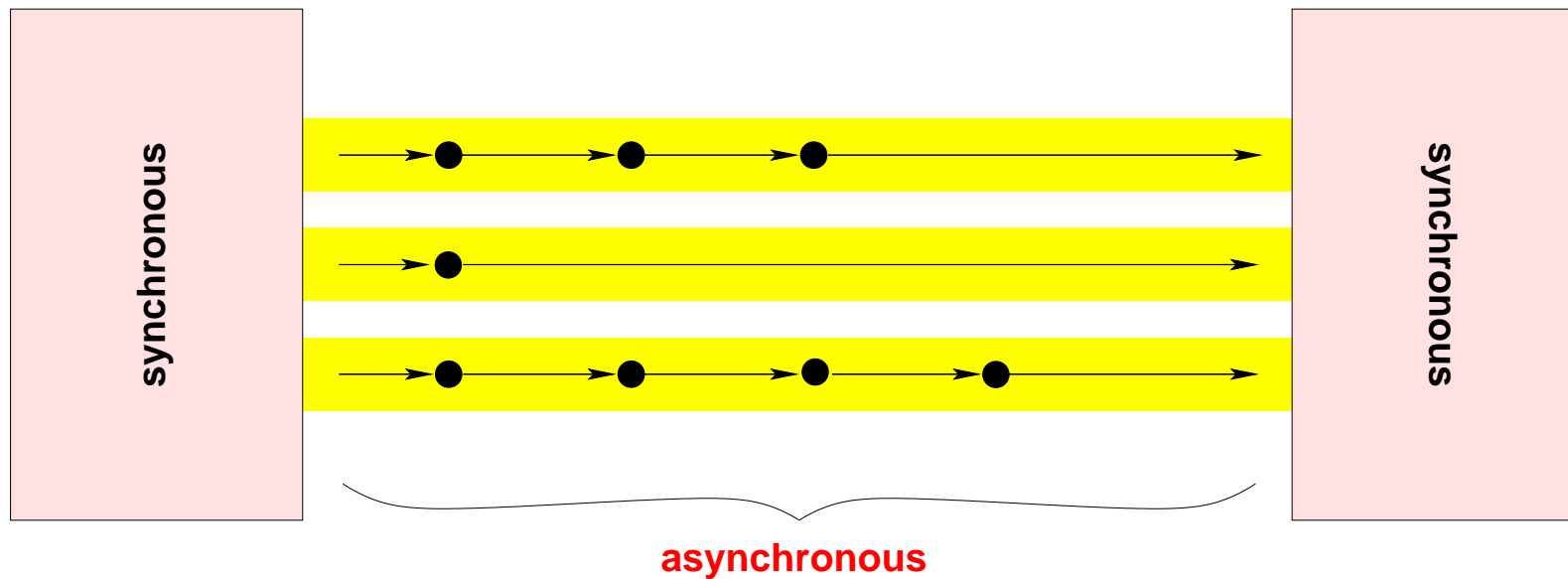


for systems *and* communications

preserving semantics, informal definition

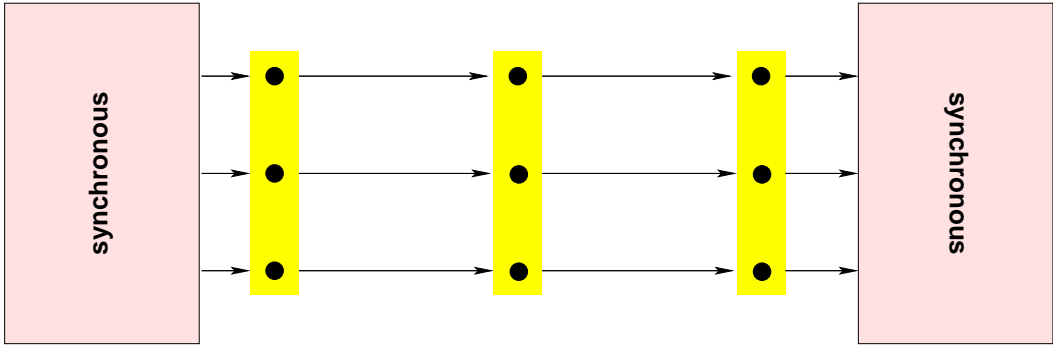


preserving semantics, informal definition

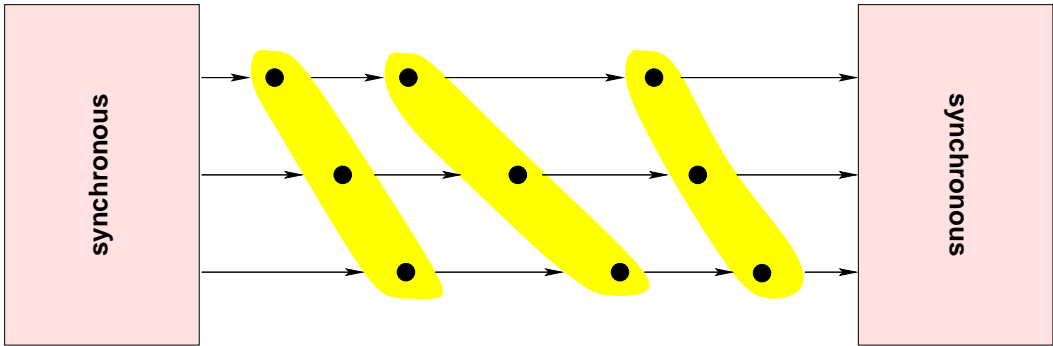


How to ensure that the two communicating components do not see the difference?

The issue of variable communication delay

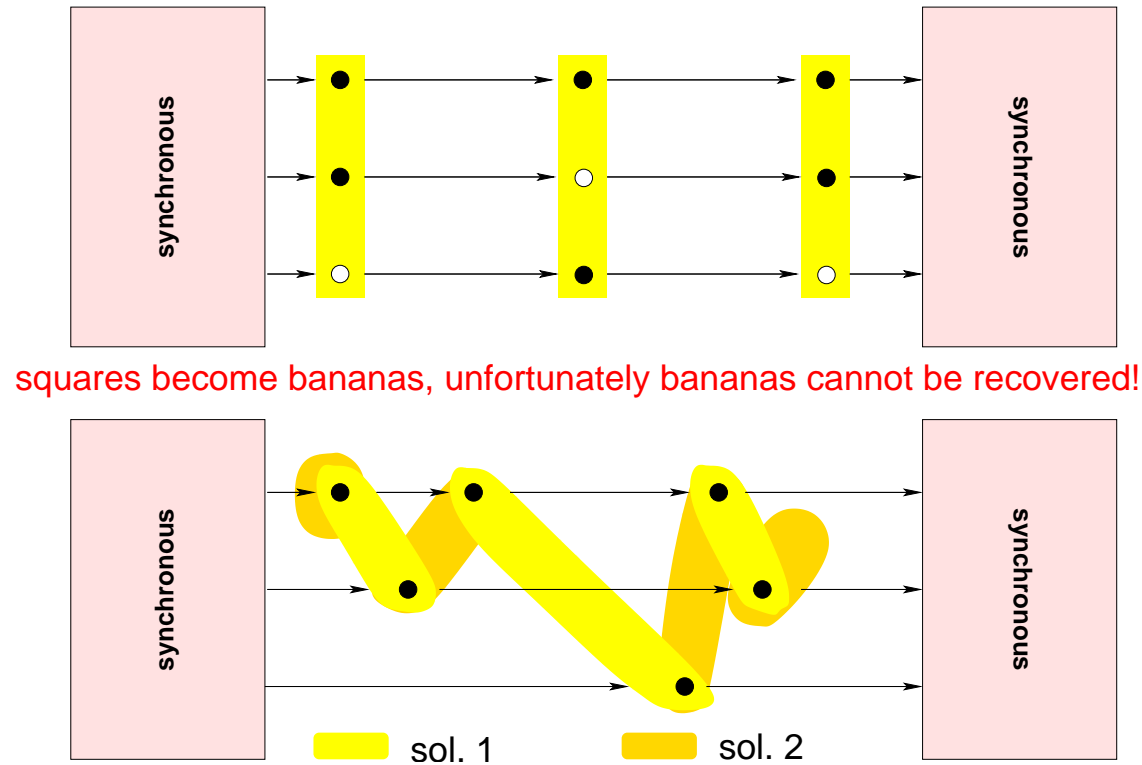


squares become bananas, but bananas can be recovered!



if \exists no absent variable, that asynchronous communications desalign reactions in physical time can be easily dealt with by compensating, at receiver, for variable latency.

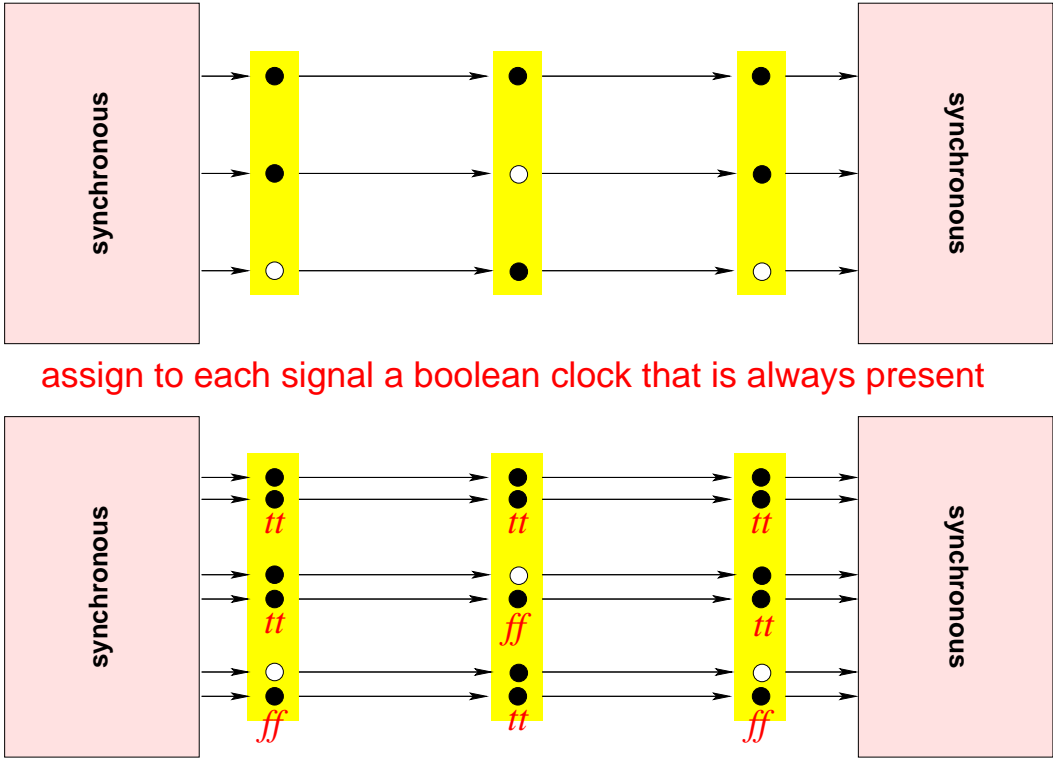
The issue of variable communication delay



if \exists no absent variable, that asynchronous communications desalign reactions in physical time can be easily dealt with by compensating, at receiver, for variable latency.

problematic if some variables are absent at some reactions.

The issue of variable communication delay



assign to each signal a boolean clock that is always present

This is a straightforward solution. However it requires 1/ to assign one signalling wire to each signal, 2/ to run computations and communications at the fastest rate, even for an architecture with slow/fast components. In the sequel we develop an algebraic approach that allows to optimize this.

1. Synchrony vs. asynchrony

2. Synchronous/Asynchronous Transition Systems

3. Preserving semantics: Endochrony, Isochrony

4. Discussion

Synchronous Transition Systems: $\Phi = \langle V, \Theta, \rho \rangle$

V : finite set of typed *variables* $\in D \cup \{\perp\}$ (“absent”)

V^- : “previous” variables

$\Theta(V)$: assertion characterizing *initial states*

$\rho(V^-, V)$: *transition relation*

states: valuation of all variables, $s = (s[v])_{v \in V}$

run: $\sigma : s_0, s_1, s_2, \dots$, sequence of states such that

$$s_0 \models \Theta(s_0) \quad \wedge \quad \forall i > 0, (s_{i-1}, s_i) \models \rho(s_{i-1}, s_i)$$

signal: $\sigma[v] : s_0[v], s_1[v], s_2[v], \dots$

Composing STS

$$\Phi_i = \langle V_i, \Theta_i, \rho_i \rangle, i = 1, 2$$

$$\Phi = \Phi_1 \parallel \Phi_2$$

$$V = V_1 \cup V_2$$

$$\Theta = \Theta_1 \wedge \Theta_2$$

$$\rho = \rho_1 \wedge \rho_2$$

$\rho_1 \wedge \rho_2$ is the conjunction of ρ_1 and ρ_2 , defined as follows. Transitions (s_1^-, s_1) and (s_2^-, s_2) are *unifiable*, written $(s_1^-, s_1) \bowtie (s_2^-, s_2)$, iff they agree on the shared variables; then (s^-, s) is the join of (s_1^-, s_1) and (s_2^-, s_2) , and $\rho_1 \wedge \rho_2$ is the set of these (s^-, s) .

Asynchronous Systems : $\Phi = \langle V, \Sigma \rangle$

V : finite set of typed *variables* $\in D$
(“absent” has no meaning)

local states: valuation of individual variables, $s[v]$

signal: $\sigma[v] : s_0[v], s_1[v], s_2[v], \dots$

run: $\sigma = \{\sigma[v], v \in V\}$, tuple of signals

Σ : set of legal runs

Composing AS

$$\Phi_i = \langle V_i, \Sigma_i \rangle, i = 1, 2$$

$$\Phi = \Phi_1 \parallel \Phi_2$$

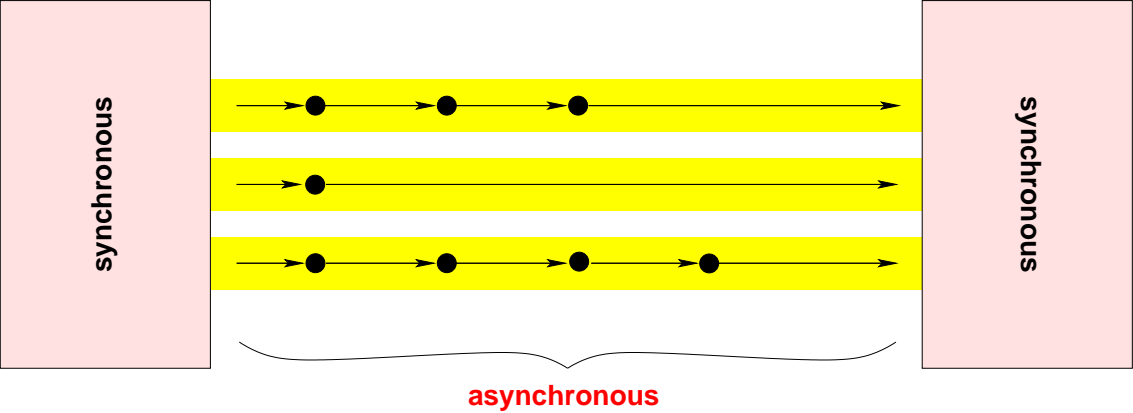
$$V = V_1 \cup V_2$$

$$\Sigma = \Sigma_1 \wedge \Sigma_2$$

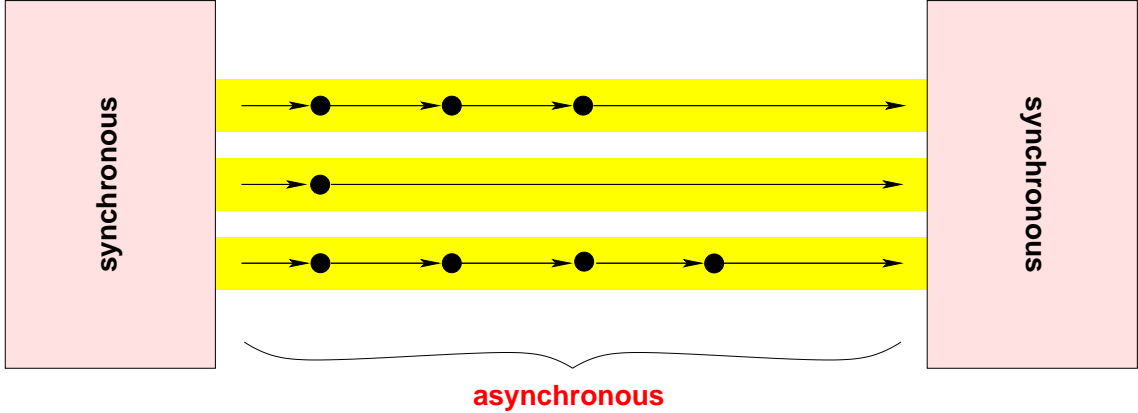
$\Sigma_1 \wedge \Sigma_2$ is the conjunction of Σ_1 and Σ_2 , defined as follows. Runs σ_1 and σ_2 are *unifiable*, written $\sigma_1 \bowtie \sigma_2$, iff they agree on the shared variables, meaning that signals associated with shared variables are equal; then σ is the join of σ_1 and σ_2 , and Σ is the set of these σ .

1. Synchrony vs. asynchrony
2. Synchronous/Asynchronous Transition Systems
3. Preserving semantics: Endochrony, Isochrony
4. Discussion

$$\mathbf{GALS} = \Phi_1 \parallel \underbrace{\text{asynch channel}}_{?} \parallel \Phi_2$$



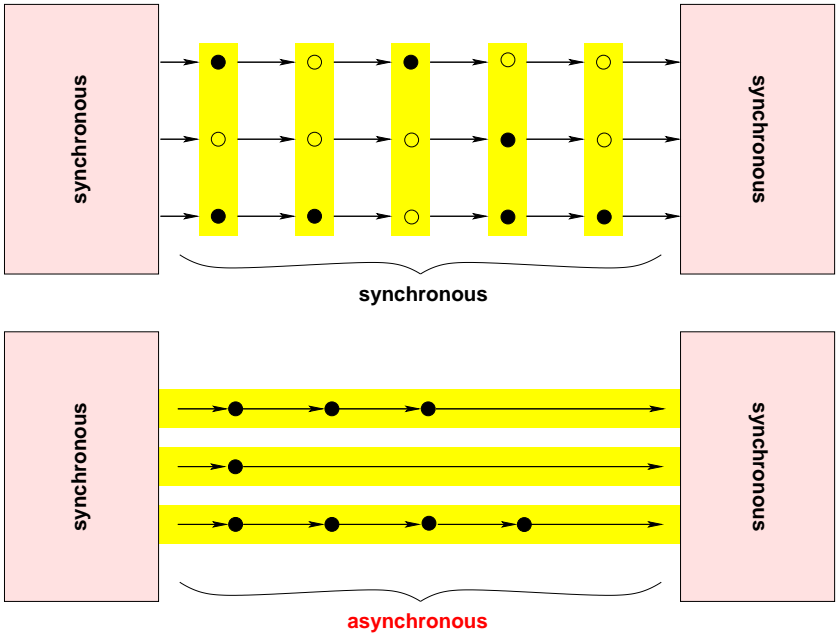
$$\mathbf{GALS} = \Phi_1 \parallel \underbrace{\text{asynch channel}}_{?} \parallel \Phi_2$$



$$\mathbf{GALS} : \Phi_1 \parallel_a C \parallel_a \Phi_2$$

σ_1 run of Φ_1 , σ_1^a desynchronized run of Φ_1 ,
 c (asynchronous) run of C ; write $\sigma_1 \parallel_a c$ iff $\sigma_1^a \parallel c$;
 then $\Phi_1 \parallel_a C$ is the set of pairs (σ_1, c) such that $\sigma_1 \parallel_a c$.

GALS: preserving semantics?



$$\Phi_1 \parallel \Phi_2 = \Phi_1 \parallel \textit{Identity} \parallel \Phi_2$$

$$\stackrel{?}{\equiv} \Phi_1 \parallel C \parallel \Phi_2$$

symbol \equiv means that both sides possess identical pairs of unifiable runs, for Φ_1 and Φ_2 .

Two key problems

Thm 1 [Emsoft2003]: $\Phi_1 \parallel \Phi_2 \equiv \Phi_1 \parallel_a C \parallel_a \Phi_2$ holds if Questions 1 & 2 below receive positive answers

QUESTION 1: is it possible, for $\Phi_i, i = 1, 2$, to reconstruct the successive reactions from a “desynchronised” run ?

$$\Phi_i \longrightarrow \Phi_i^a \stackrel{?}{\longrightarrow} \Phi_i$$

QUESTION 2: does desynchronization commute with parallel composition, for pair (Φ_1, Φ_2) ?

$$\Phi_1^a \parallel \Phi_2^a \stackrel{?}{=} (\Phi_1 \parallel \Phi_2)^a$$

QUESTION 1: endochrony [Benv. & al. 2000]

$$\Phi \longmapsto \Phi^a \stackrel{?}{\longmapsto} \Phi$$

$$\emptyset \hookrightarrow V_1 \hookrightarrow V_2 \hookrightarrow \dots \hookrightarrow V,$$

where:

$$V_1 \hookrightarrow V_2 : \text{infer presence of } V_2 \text{ from values of } V_1$$

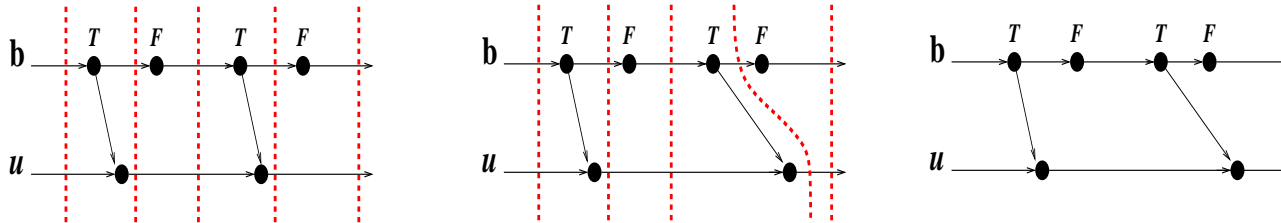
**protocol for the on-line reconstruction of successive reactions,
for endochronous STS:**

await V_1 \Rightarrow $\text{val}(V_1)$ \Rightarrow $\text{pres/abs}(V_2)$
await $\text{pres}(V_2)$ \Rightarrow $\text{val}(\text{pres}(V_2))$ \Rightarrow $\text{pres/abs}(V_3)$
etc... until V

endochrony : (counter)examples

examples :

- a single-clocked STS
- if $b \equiv T$ then get_u



counterexample :

- $\Phi_1 \parallel \Phi_2$, where the Φ_i do not communicate at all ($V_1 \cap V_2 = \emptyset$) — internal asynchrony

QUESTION 2: isochrony [Benv. & al. 2000]

$$\Phi_1^a \parallel \Phi_2^a \stackrel{?}{=} (\Phi_1 \parallel \Phi_2)^a$$

$\rho_1 \wedge \rho_2$: unifies on {absence, present values}

$\rho_1 \wedge_a \rho_2$: unifies on {present values} only

isochrony: $\rho_1 \wedge \rho_2 = \rho_1 \wedge_a \rho_2$

isochrony, details

$(\Phi_1, \Phi_2) : W = V_1 \cap V_2$, write $t = (s^-, s)$

$t_1 \bowtie t_2 : \forall w \in W, t_1[w] = t_2[w]$

$\rho_1 \wedge \rho_2 : \forall t_1 \bowtie t_2, t[v] = \begin{cases} \text{if } v \in V_i \\ \text{then } t_i[v] \end{cases}$

$t_1 \bowtie_a t_2 : s_1, s_2 \neq \perp$ and $\forall w \in W \begin{cases} s_1[w], s_2[w] \neq \perp \\ \Downarrow \\ t_1[w] = t_2[w] \end{cases}$

$\rho_1 \wedge_a \rho_2 : \forall t_1 \bowtie_a t_2, t[v] = \begin{cases} \text{if } v \in V_i \text{ and } t_i[v] \neq \perp \\ \text{then } t_i[v] \text{ else } \perp \end{cases}$

isochrony: $\rho_1 \wedge \rho_2 = \rho_1 \wedge_a \rho_2$

($s \neq \perp$ means that state s is non-silent, i.e., has some present variables in it)

isochrony : (counter)examples

examples :

- a pair (Φ_1, Φ_2) with single-clocked communications (strongly synchronous)
- $V_1 \cap V_2 = \emptyset$: a pair (Φ_1, Φ_2) with no communication at all (fully asynchronous)

counterexample :

```
[await X || await Y || Z = merge(X,Y) ]  
||  
[ emit X when C || emit Y when not C ]
```

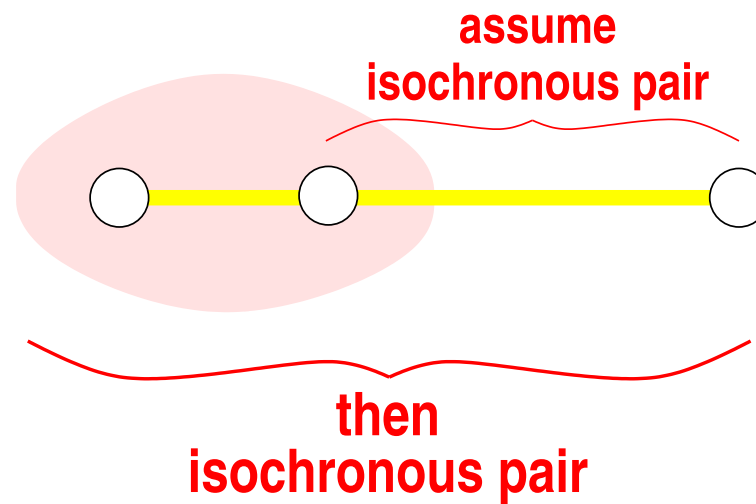
Isochrony is local

(Φ_2, Ψ) isochronous

(Φ_1, Ψ) do not interact



$(\Phi_1 \parallel \Phi_2, \Psi)$ isochronous



Globally Asynch. Locally Synch.

**Φ_1 and Φ_2 endochronous
and (Φ_1, Φ_2) isochronous**

\Downarrow by Thm 1

$$\Phi_1 \parallel \Phi_2 \equiv \Phi_1 \mathbin{a\parallel} \text{channel} \mathbin{\parallel_a} \Phi_2$$

from synchronous specification to **GALS** deployment while preserving semantics

endo/isochrony

**checkable on
synchronous program(s)**

**can be enforced
by adding protocols**

1. **Synchrony vs. asynchrony**
2. **Synchronous/Asynchronous Transition Systems**
3. **Preserving semantics: Endochrony, Isochrony**
4. **Discussion**

What about GALS with ≥ 2 components?

- So far isochrony has been defined only for pairs; for ≥ 2 components it is not enough to require isochrony for all pairs, because it is not true that (Φ_i, Φ_j) for $(i, j) = (1, 2), (2, 3), (3, 1)$ implies $((\Phi_1 \parallel \Phi_2), \Phi_3)$ isochronous.
- Thus we need to define isochrony for tuples directly, a straightforward extension. Unfortunately, whereas isochrony is tight for pairs, is it too strong for tuples — many correct GALS deployments are not isochronous, the problem arises from components involving concurrent transitions.
- [Potop-Butucaru & Caillaud 2003] have provided the right generalization of endo/isochrony for GALS networks and components possibly involving concurrent transitions — beyond the scope of these notes.

Valid Architectures for the theory to apply

1. messages shall not be lost
2. ordering within each channel shall be preserved

YES (assuming nominal, not faulty behaviour):

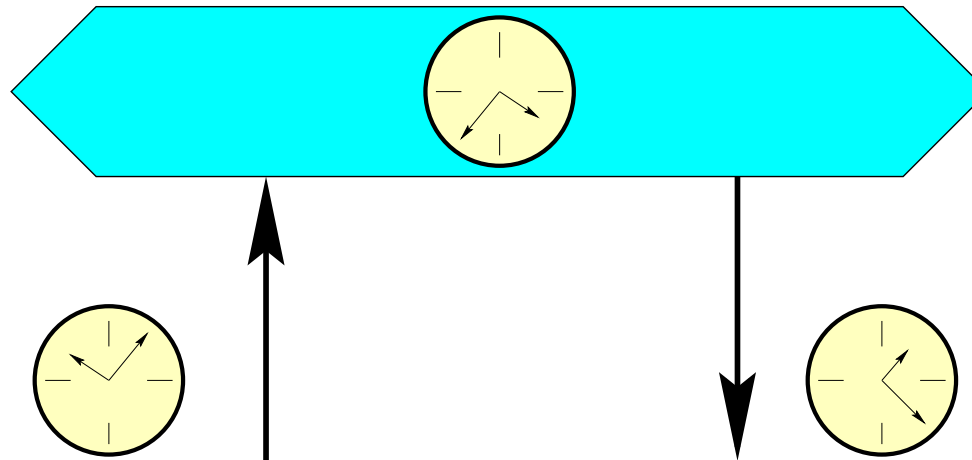
- RTOS POSIX
- point to point comm. via fifos
- “synchronous comm.” in OS
- UNIX sockets
- inter-object comm., (*dynamic instantiation*)

NO (???) :

- CAN
- ARINC
- field busses

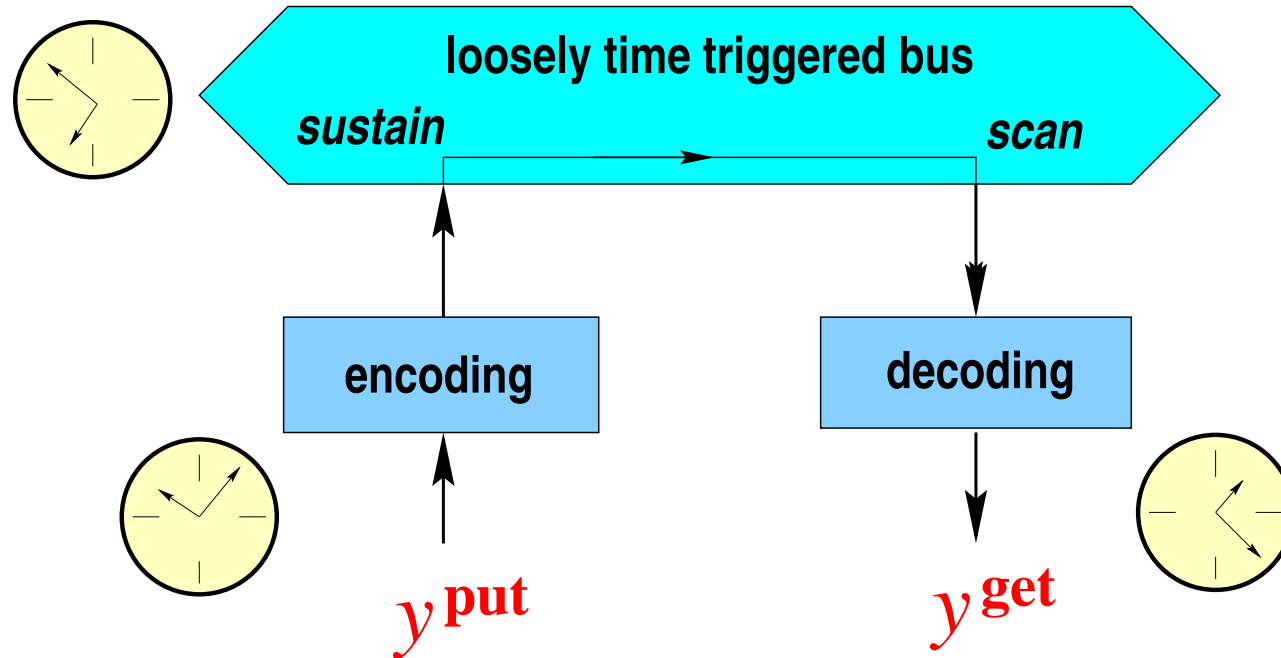
BUT ... still feasible [Emsoft2002] ...

“loosely” time triggered bus



- clocks are independent and not synchronized
- read / write are clocked & independent
- periodic bus: reads at n , writes at $n + 1$
- non blocking communication
- **robust against failures**, but...
- communications can duplicate or lose messages
⇒ our theory does not apply directly

loosely time triggered communications



Theorem :

if bus clock \gg write/read clocks (upsampling), then $y^{put} \mapsto y^{get}$ satisfies requirements for desynch.

timeliness ? timing evaluation

fault tolerance ? watchdogs & exceptions