The Quasi-Synchronous Approach to Distributed Control Systems

Paul Caspi *Verimag Laboratory*

caspi@imag.fr http://www-verimag.imag.fr

Crisys Esprit Project

http://borneo.gmd.de/~ap/crisys/

The Quasi-Synchronous Approach to Distributed Control Systems

Paul Caspi *Verimag Laboratory*

caspi@imag.fr http://www-verimag.imag.fr

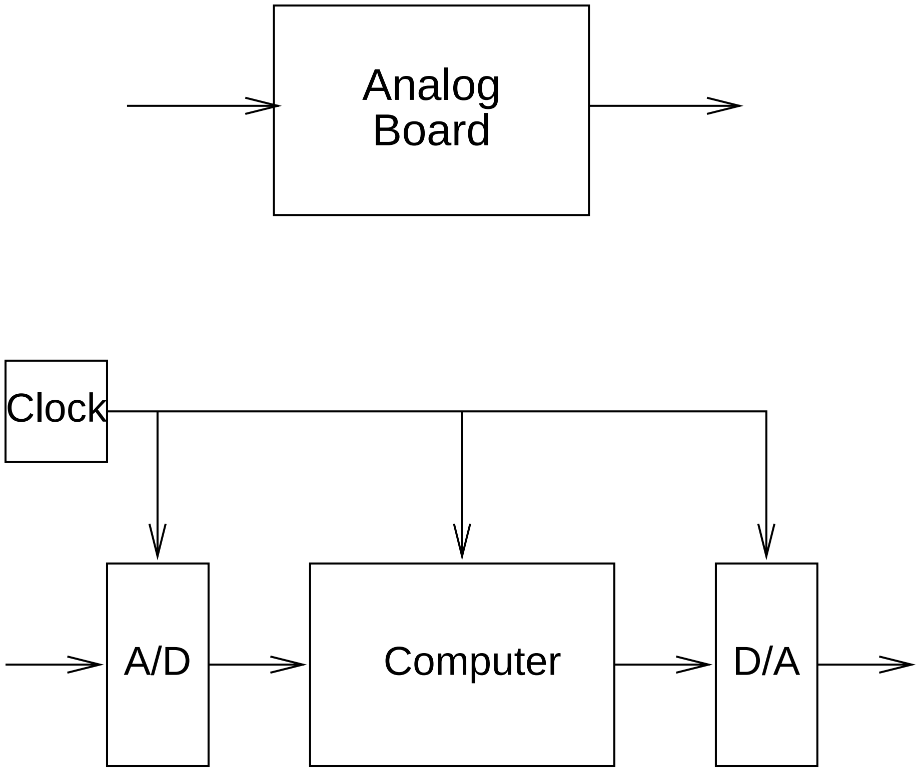
Crisys Esprit Project

http://borneo.gmd.de/~ap/crisys/

- **Where does it come from ?**
- **How to simulate it ?**
- **How to understand it ?**
- **Fault-tolerance**

Where does it come from ?

From analog boards to computers



periodic clocks

synchronous programs

Synchronous Programming

General

```
initialize state;

loop each input event

   read other inputs;
   compute outputs and state;
   emit outputs

 end loop
```

Several styles (imperative, data-flow,...)

Allow multiple simultaneous event : no performance problems

Synchronous Programming

Periodic

```
initialize state;

loop each clock

   read other inputs;
   compute outputs and state;
   emit outputs

 end loop
```

Synchronous Programming

Periodic

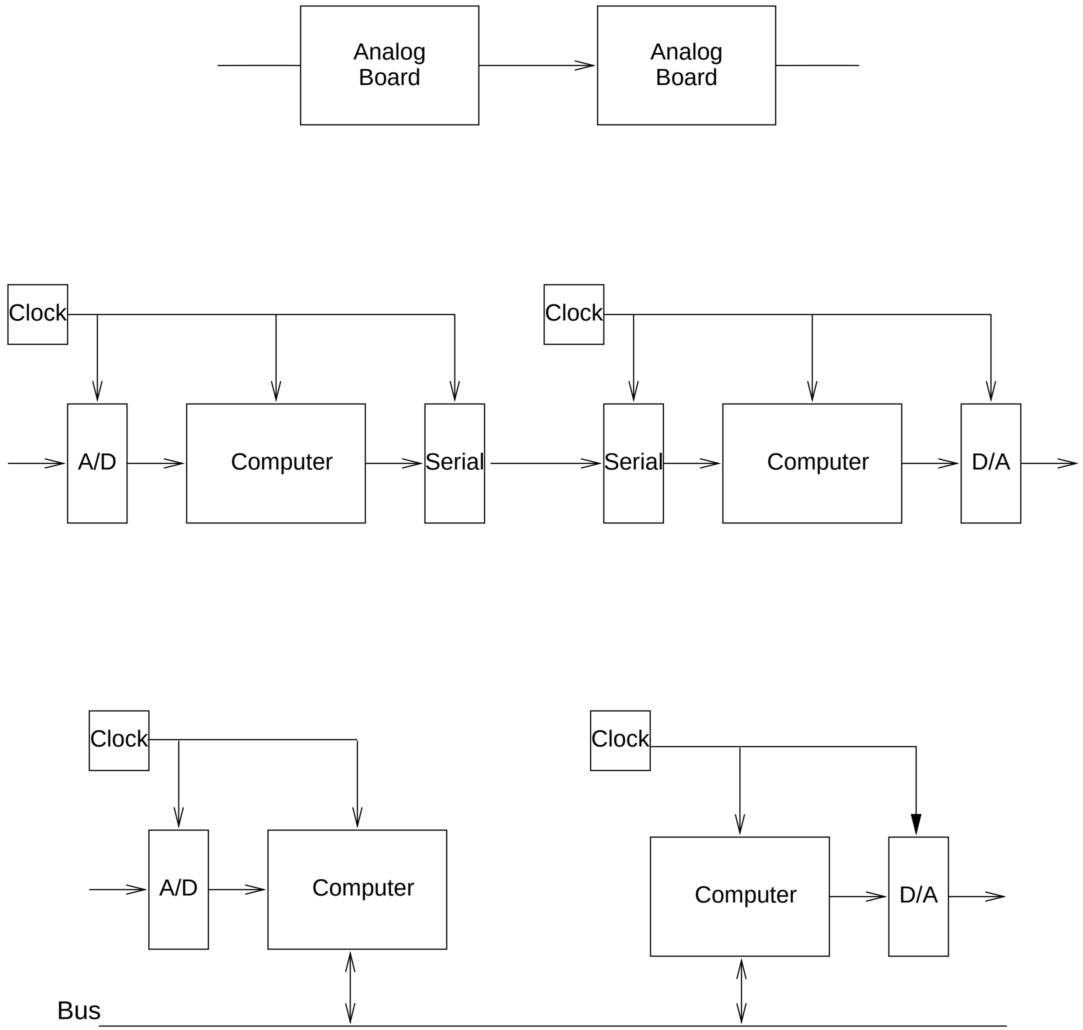
```
initialize state;

loop each clock

   read other inputs;
   compute outputs and state;
   emit outputs

 end loop
```

most applications of synchronous programming are actually periodic ones.

hybridity: sampling differential equations require periodicity!

Where does it come from ?

From networks of analog boards to local area networks



independent periodic clocks

synchronous programs

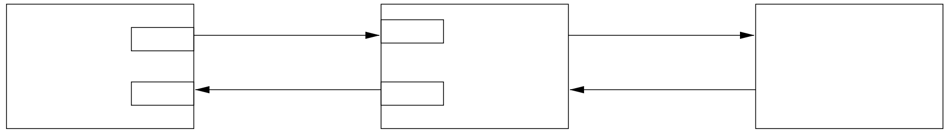
Interest

Autonomy, robustness

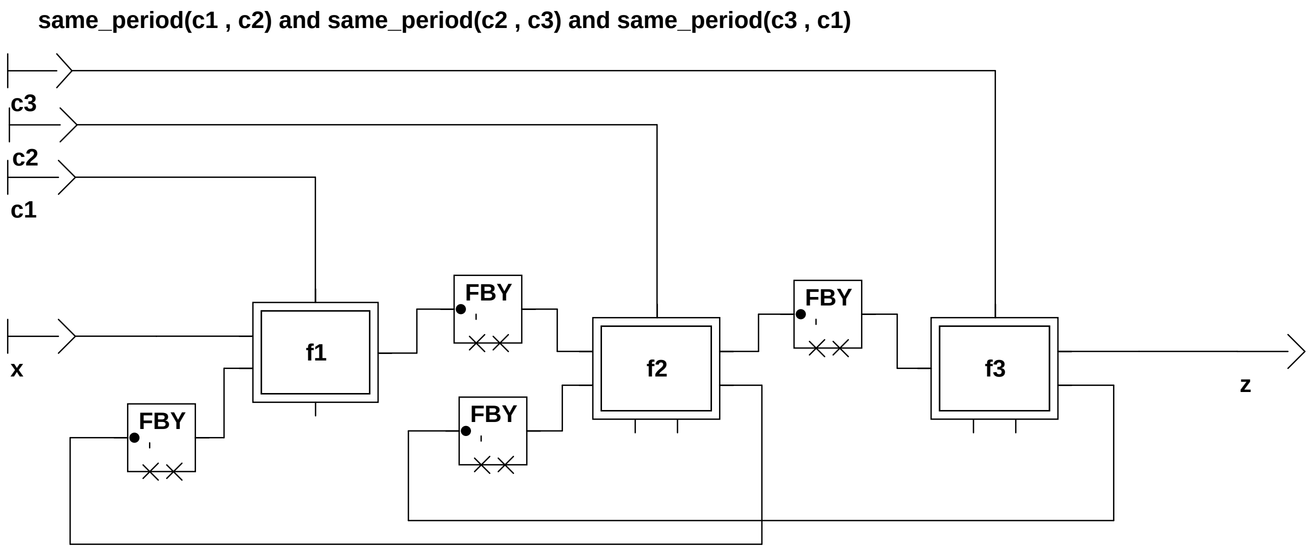
- **Each computer is a complete one, including its own clock and even possibly its own power supply.**

- **Communication between computers is non-blocking, based on periodic reads and writes, akin to periodic sampling.**

How to formalize it



Net View on chain - eq_chain



Synchronous simulation, test and verification tools apply

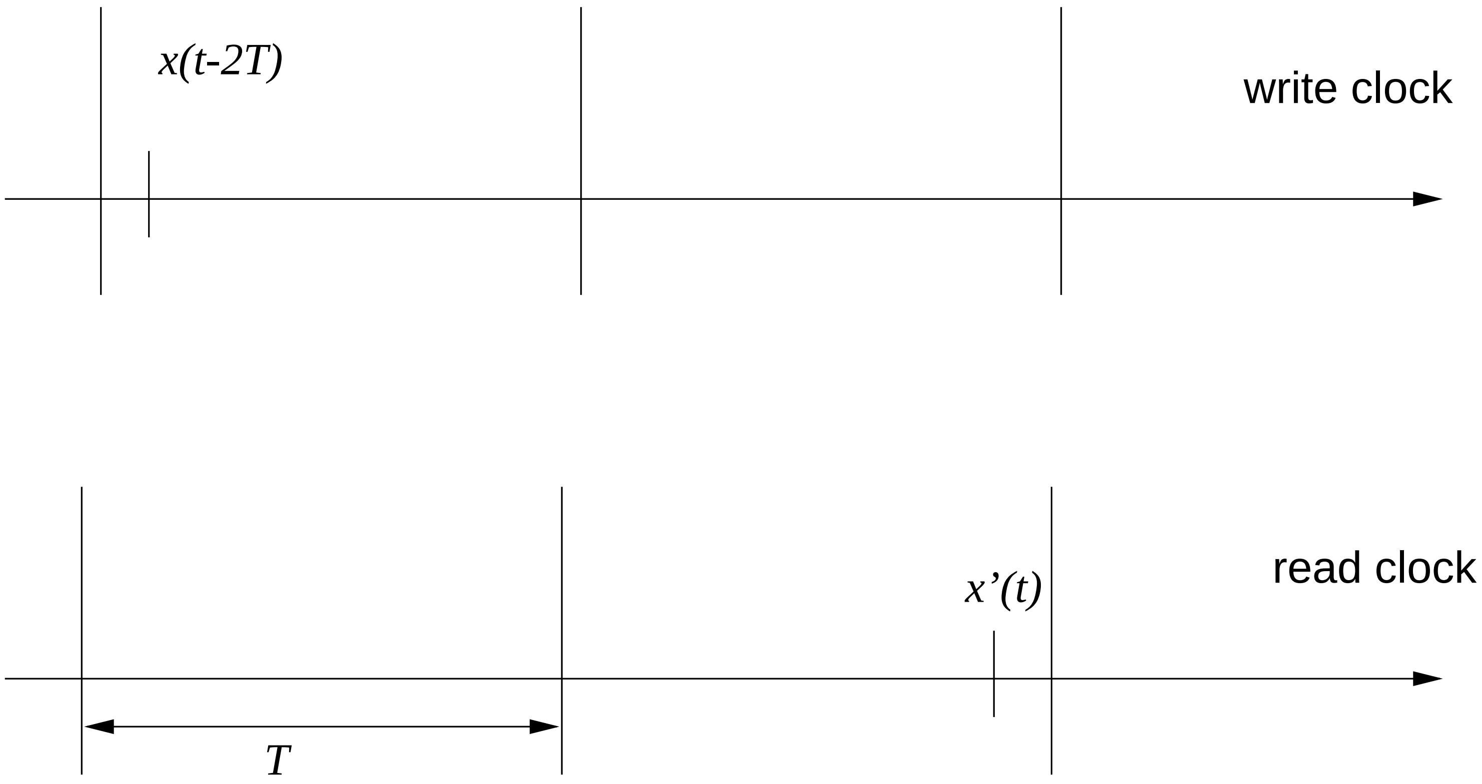
Efficiency issues ?

How to understand it ?

- **Communication Abstraction**
- **Continuous Systems**
- **Non Continuous Systems**
- **Mixed Systems**

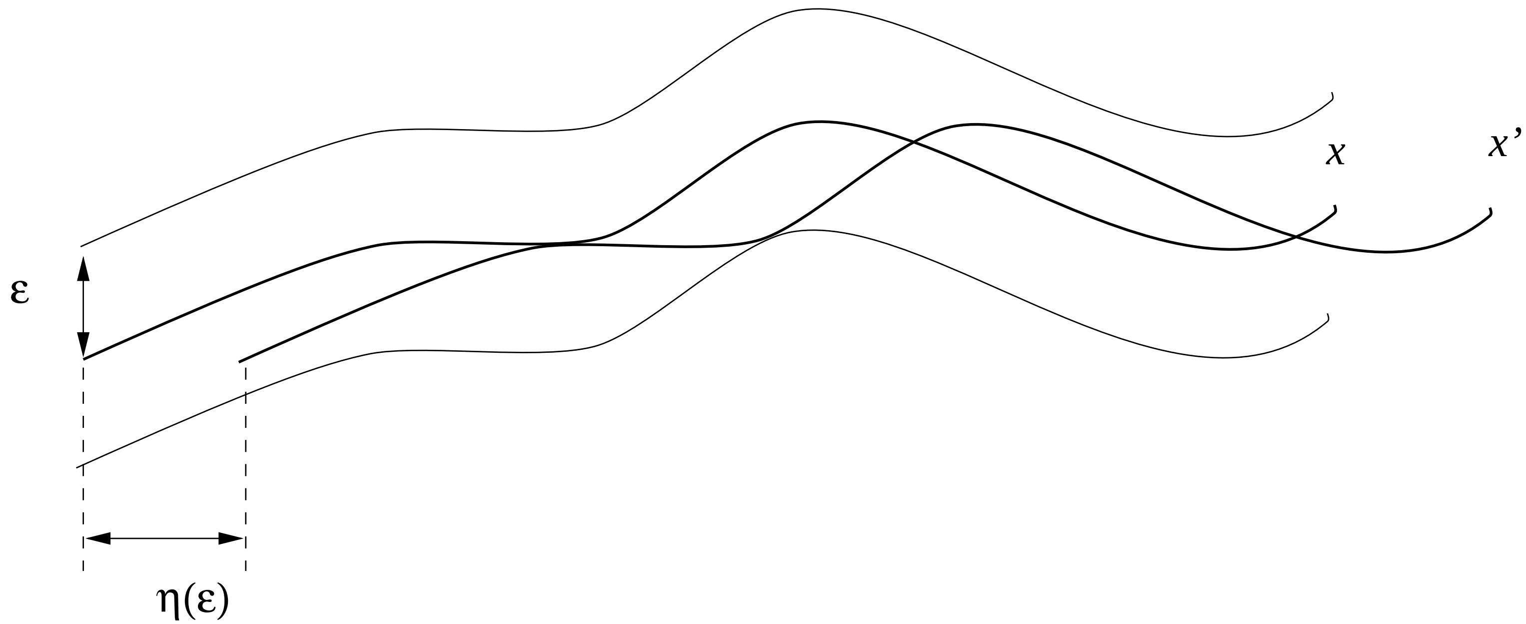
Communication Abstration

Worst situation: reads occur just before writes



Bounded communication delays

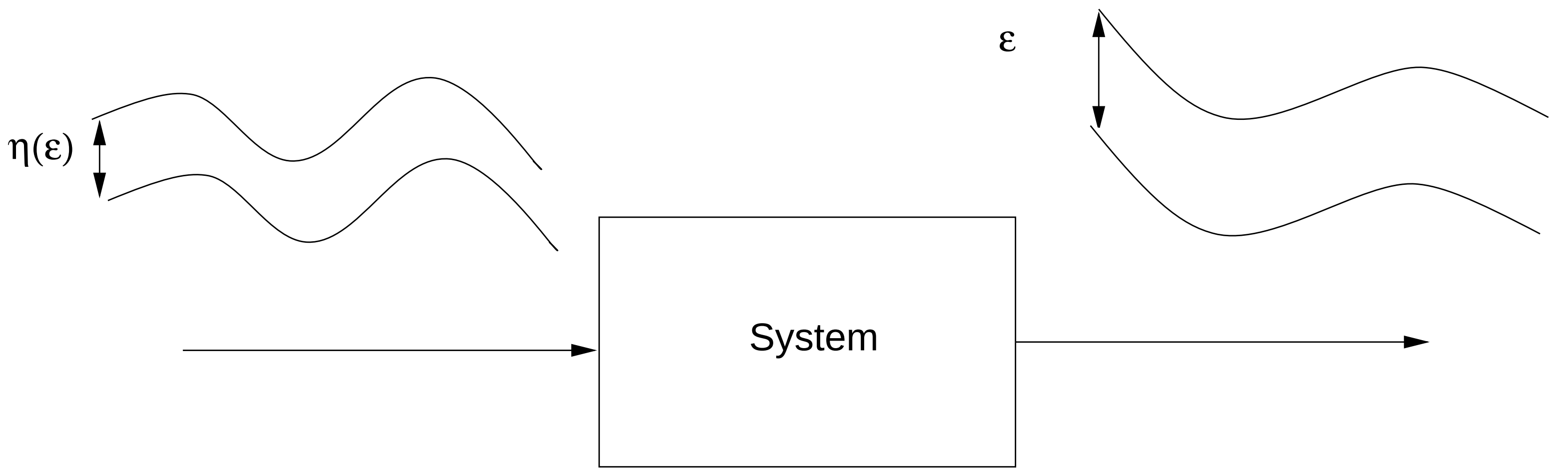
Uniformly Continuous Signals



$$\forall \varepsilon > 0, \exists \eta > 0, \forall t, t', |t - t'| \leq \eta_x \Rightarrow |x(t) - x(t')| \leq \varepsilon$$

Bounded delays yield bounded errors

Uniformly Continuous Systems



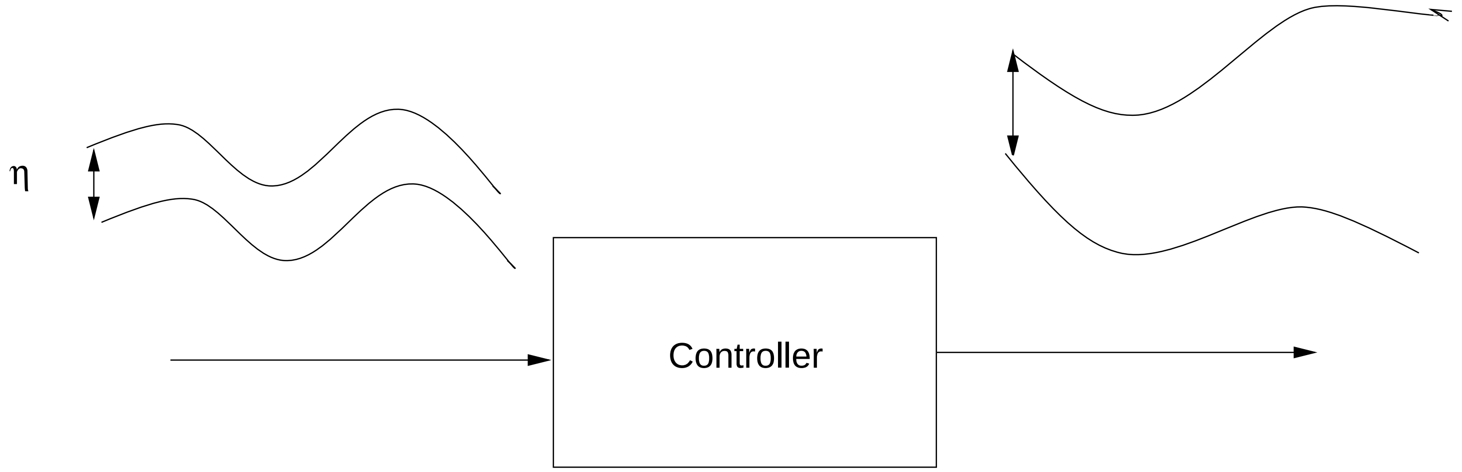
$$\forall \varepsilon > 0, \exists \eta > 0, \forall x, x', ||x - x'||_\infty \leq \eta \Rightarrow ||f(x) - f(x')||_\infty \leq \varepsilon$$

Bounded errors yield bounded errors

But . . .

Even very simple controllers are not uniformly continuous.

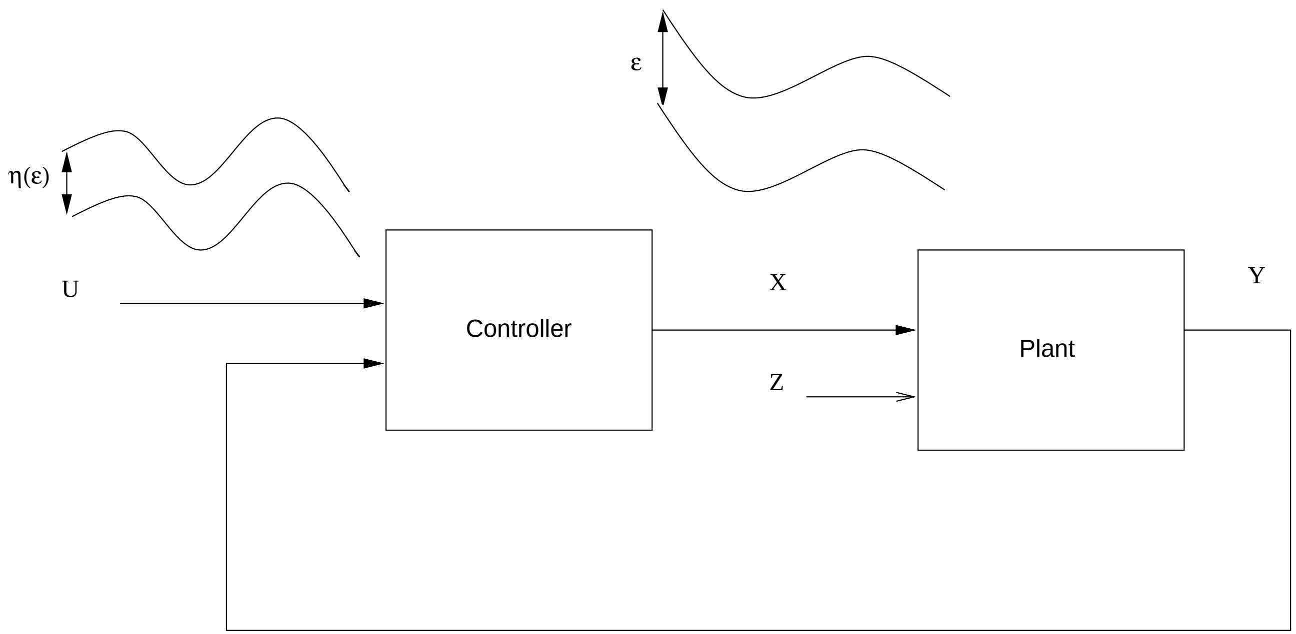
PID for instance



Bounded errors do not yield bounded errors

Stabilized Systems

The closed-loop system computes uniformly continuous signals



Bounded delays yield bounded errors

Doubts . . .

This casts a doubt on two wishful thoughts:

- **composability**
  - **system properties are the mere addition of sub-system ones**
- **separation of concerns:**
  - **automatic control people specify**
  - **computer science people implement**

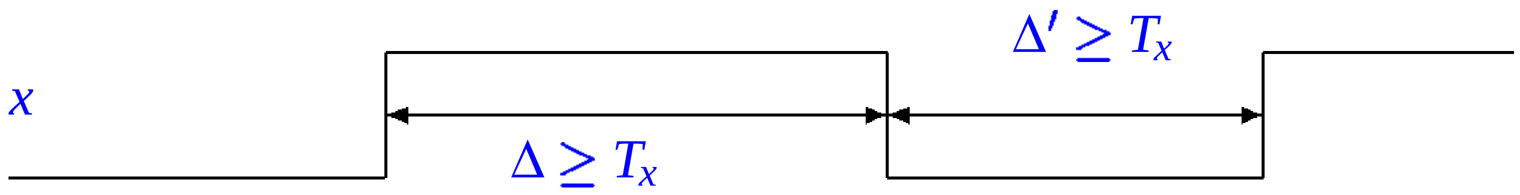
Critical control systems require a tight cooperation between both people

Non Continuous Systems

- **Combinational Systems**
- **Robust Sequential Systems**
- **Sequential Systems**

Uniform Bounded-Variability

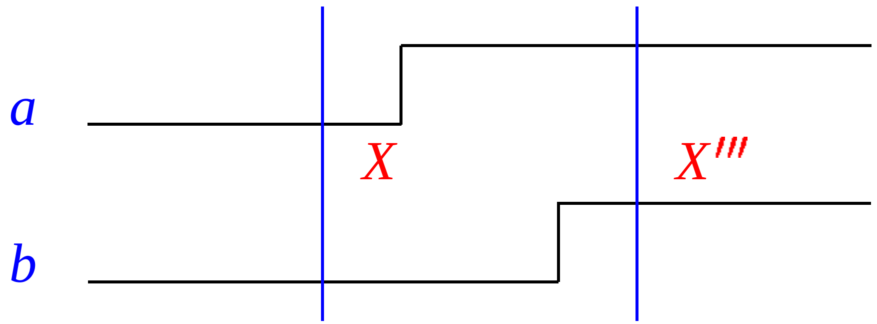
There exists a minimum stable time T_x associated with a signal x.



The analog of uniform continuity ?

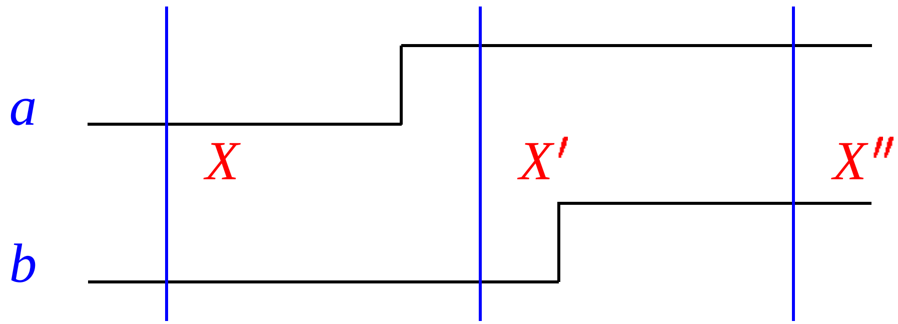
Sampling Tuples

A possible sampling



Sampling Tuples

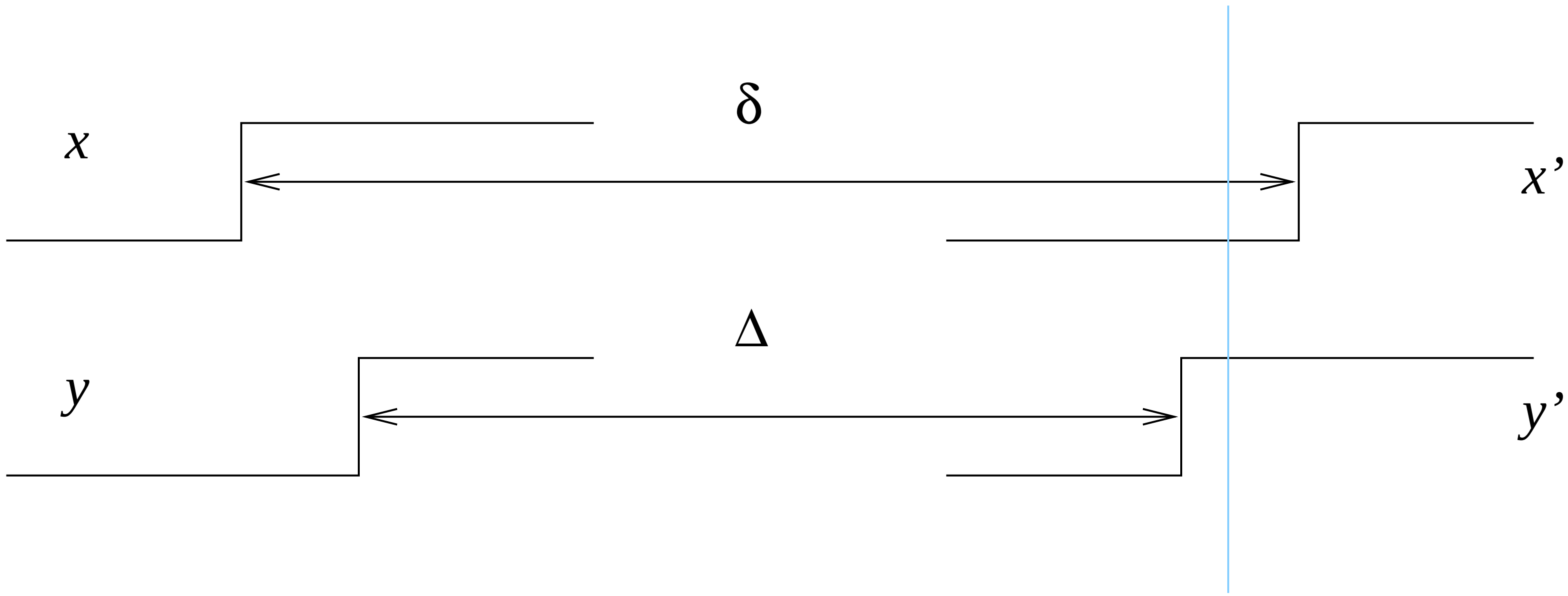
Another possible sampling



Non deterministic bounded delays

But . . .

Delays on tuples do not yield delayed tuples



Solution : Confirmation functions

Confirmation Functions

When a component of a tuple changes, wait for some $\Delta_{max} - \Delta_{min}$ time before taking it into account.

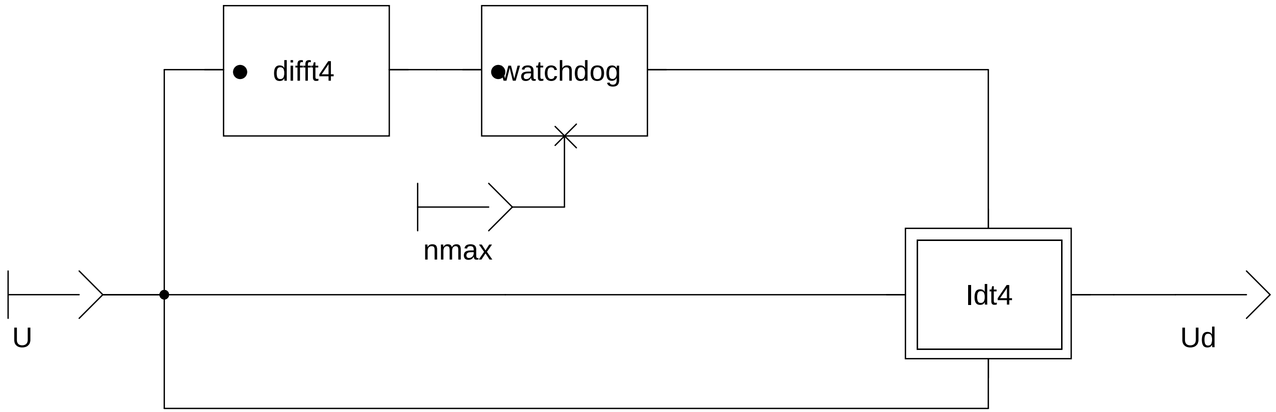
If x', y' are $(\Delta_{min}, \Delta_{max})$ bounded images of x and y,

then $confirm(x', y')$ is a delayed image of (x, y)

allows to retrieve the continuous framework

Confirmation Functions

Net View on confirm - eq_confirm



$$nmax = E\left(\frac{\Delta_{max} - \Delta_{min}}{T_{min}}\right) + 1$$

Robust Sequential Systems

idea : avoid critical races

- **between state variables : order insensitivity**
- **between inputs : confluence**

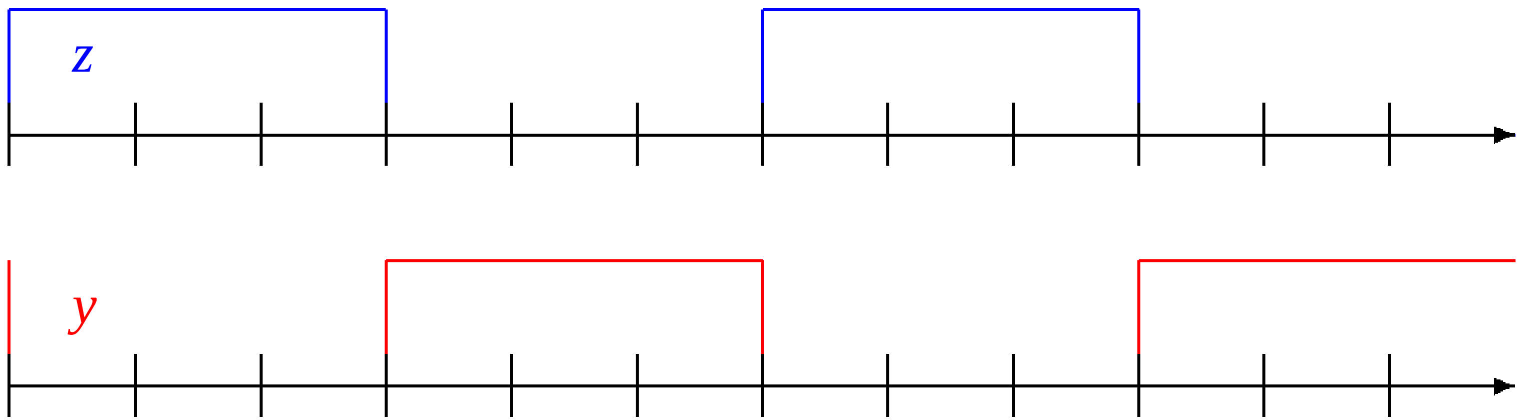
Property checker

Can robustness analysis be avoided ?

example : mutual exclusion

Property : `always not (y and z)`

a non robust solution :

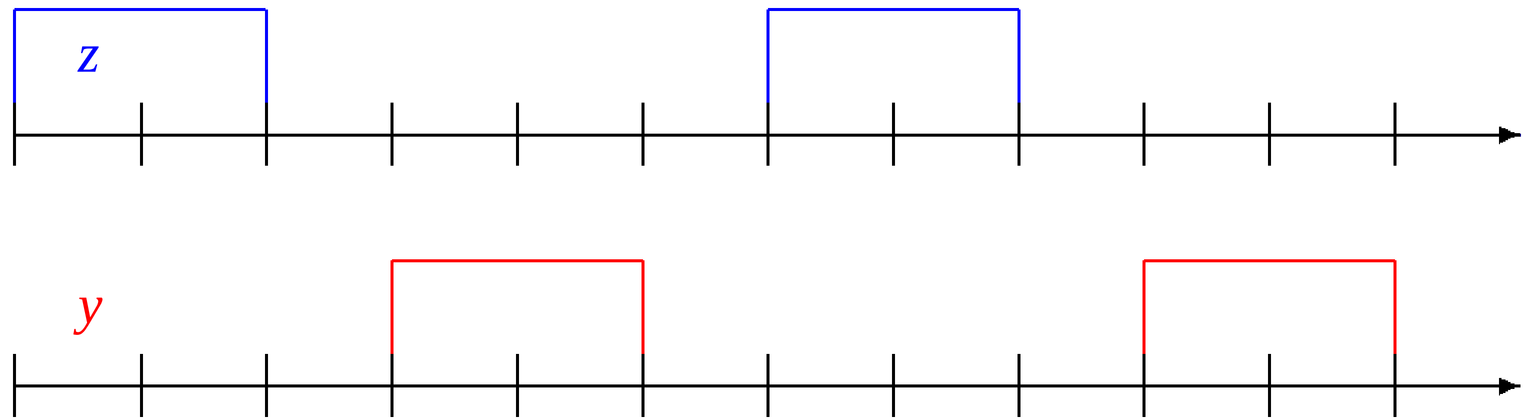


Can robustness analysis be avoided ?

example : **mutual exclusion**

Property : `always not (y and z)`

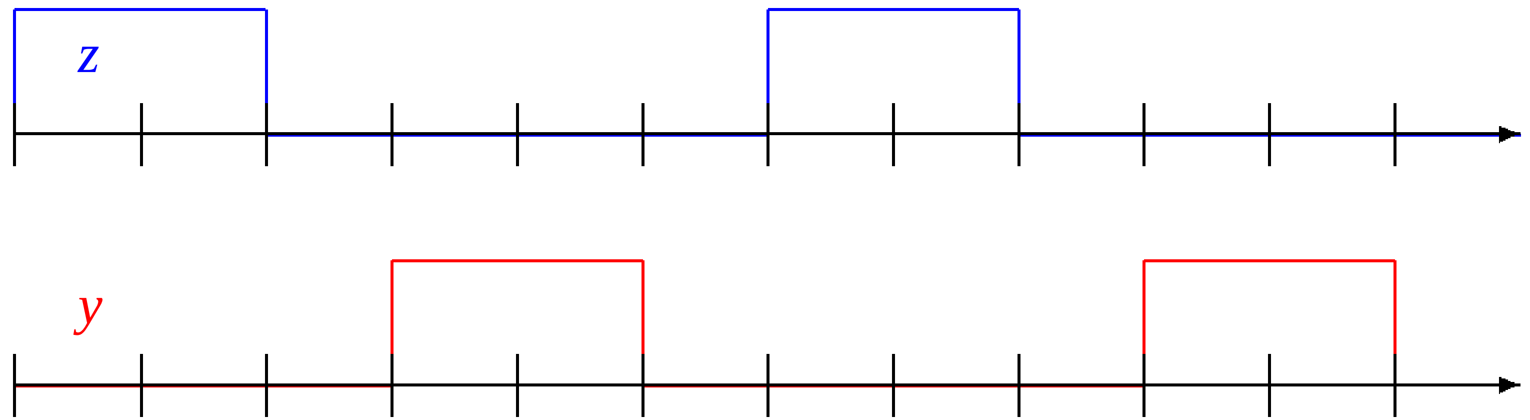
a robust solution :



same answer as for error analysis in continuous systems

Robust solutions are distributable

a robust solution :



z waits for y to go down before going up and conversely.

$$\begin{array}{l} not\ y \\ not\ z \end{array} \quad ((\rightarrow y \rightarrow not\ y)^* (\rightarrow z \rightarrow not\ z)^*)^*$$

no critical race !

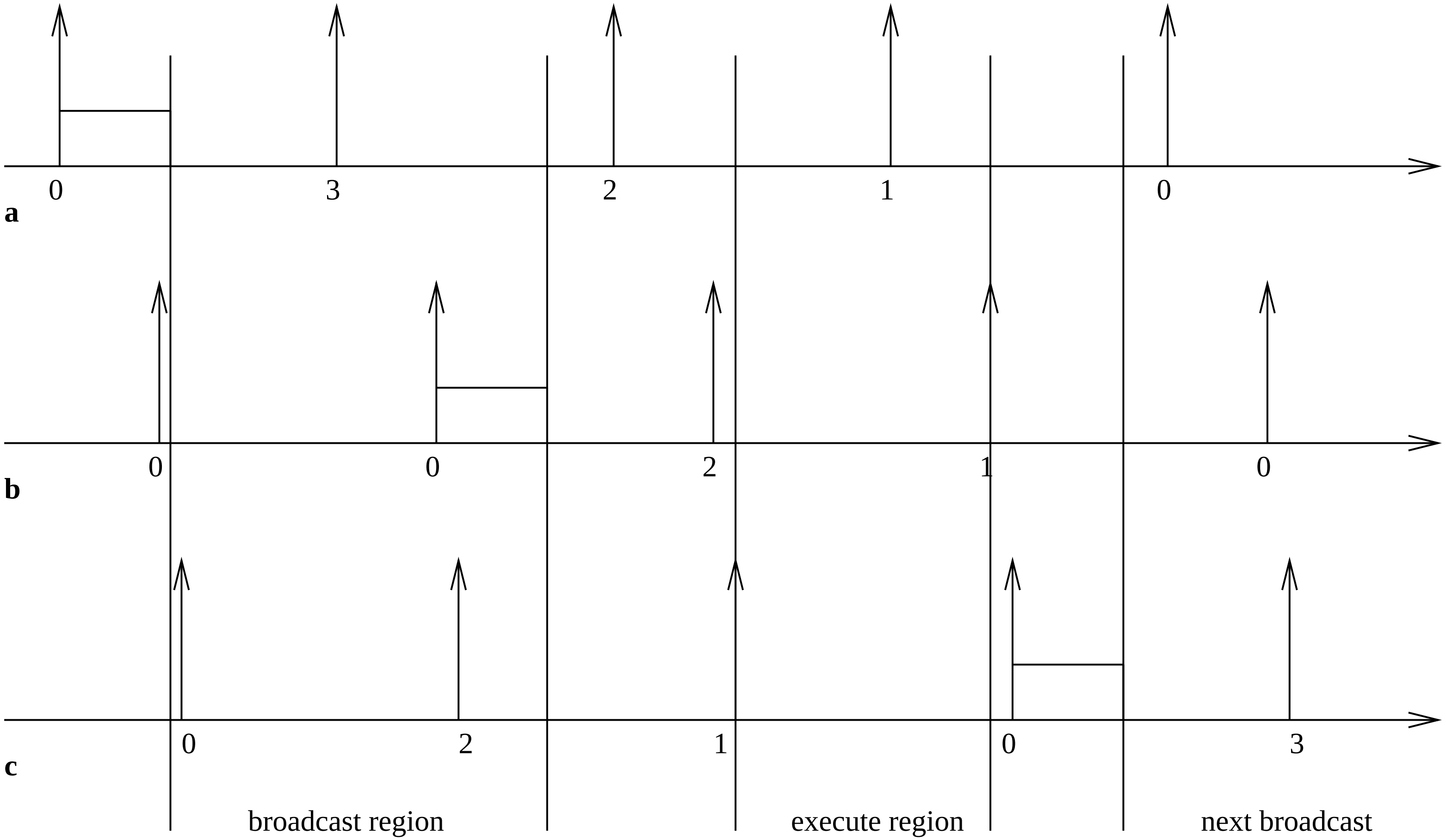
Non Robust Sequential Systems

require either soft or hard synchronization.

`Time Triggered Architecture` **for instance.**

Non Robust Sequential Systems

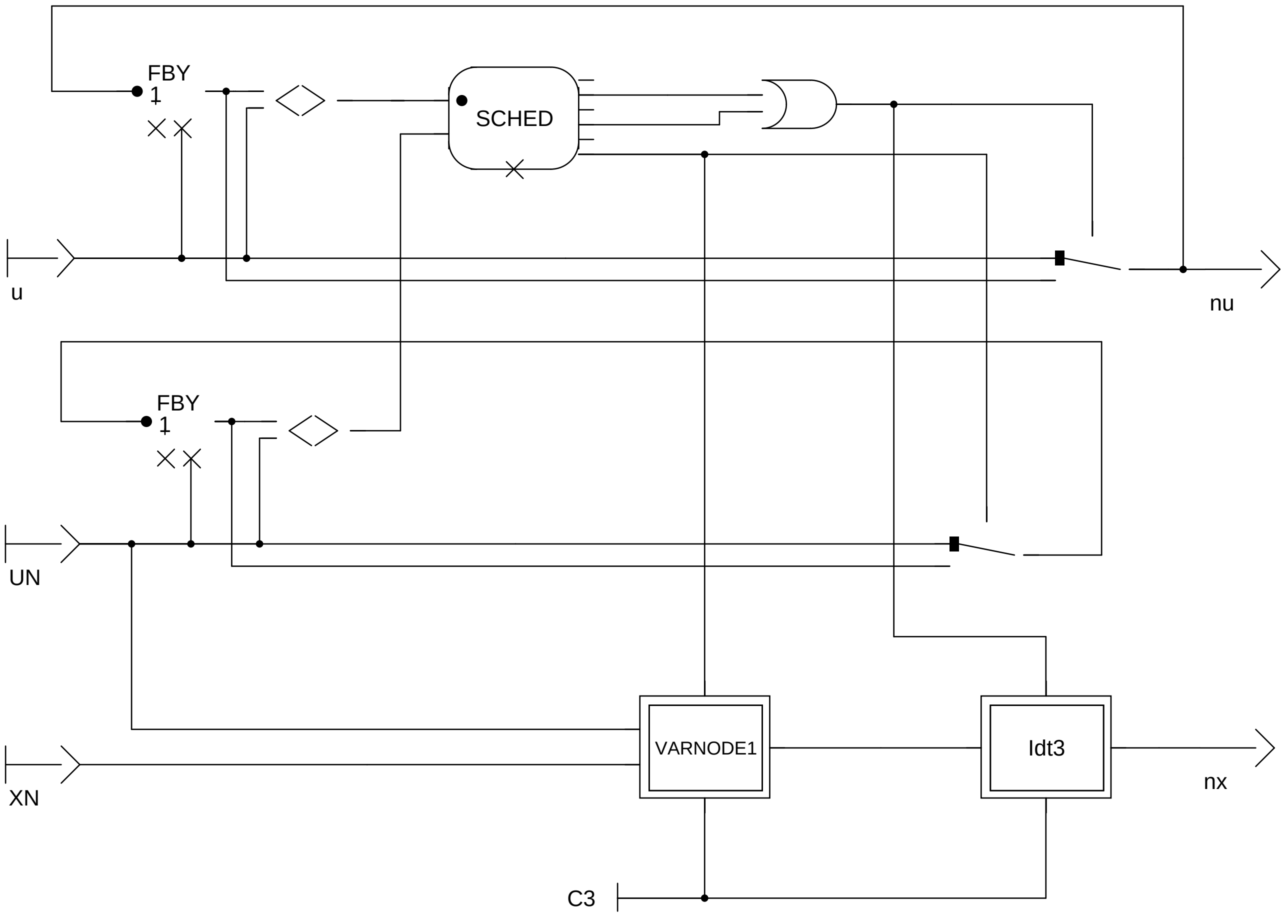
A soft synchronization algorithm



requires a speed-up by 4

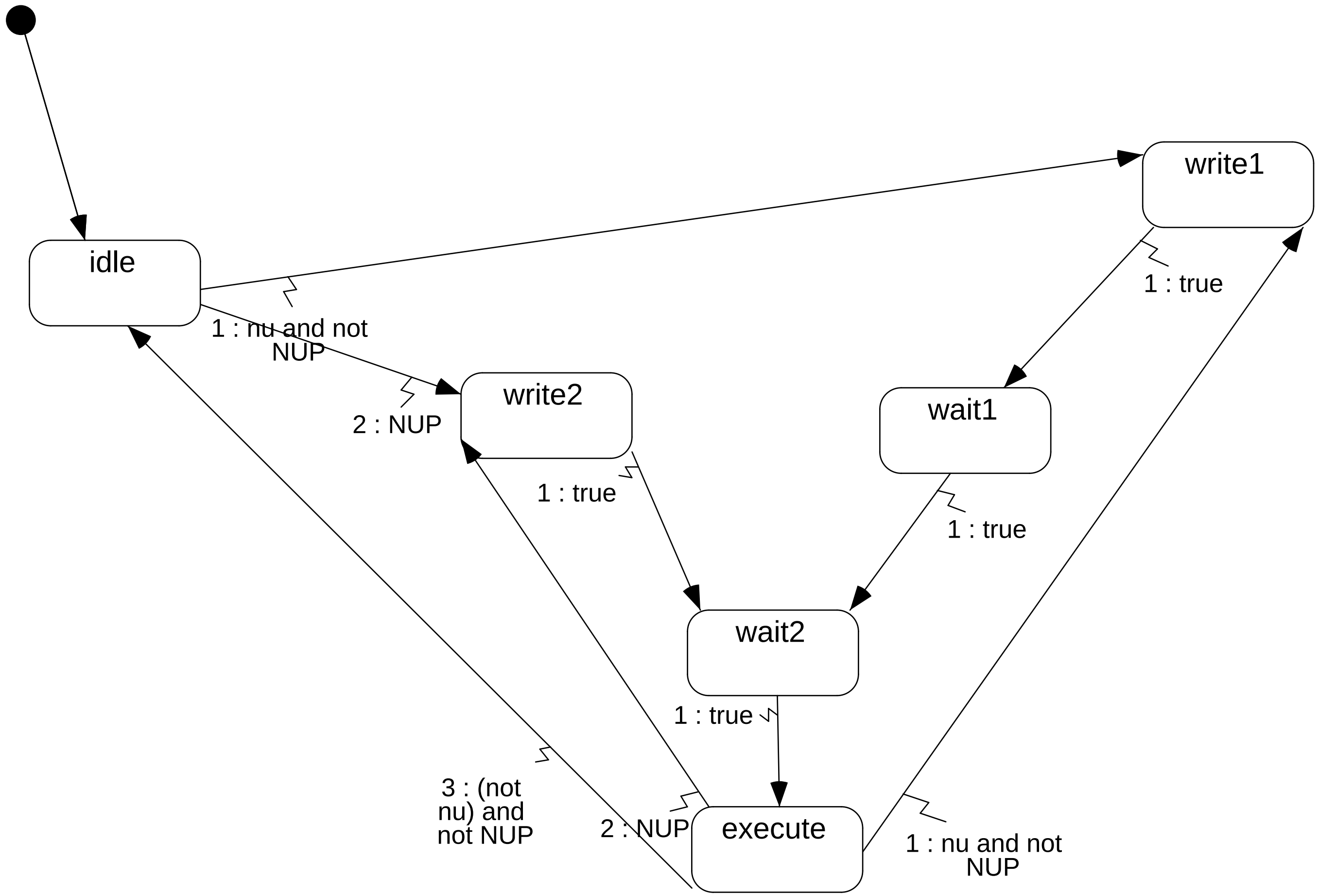
Implementation

Net View on SYNCH - eq_SYNCH



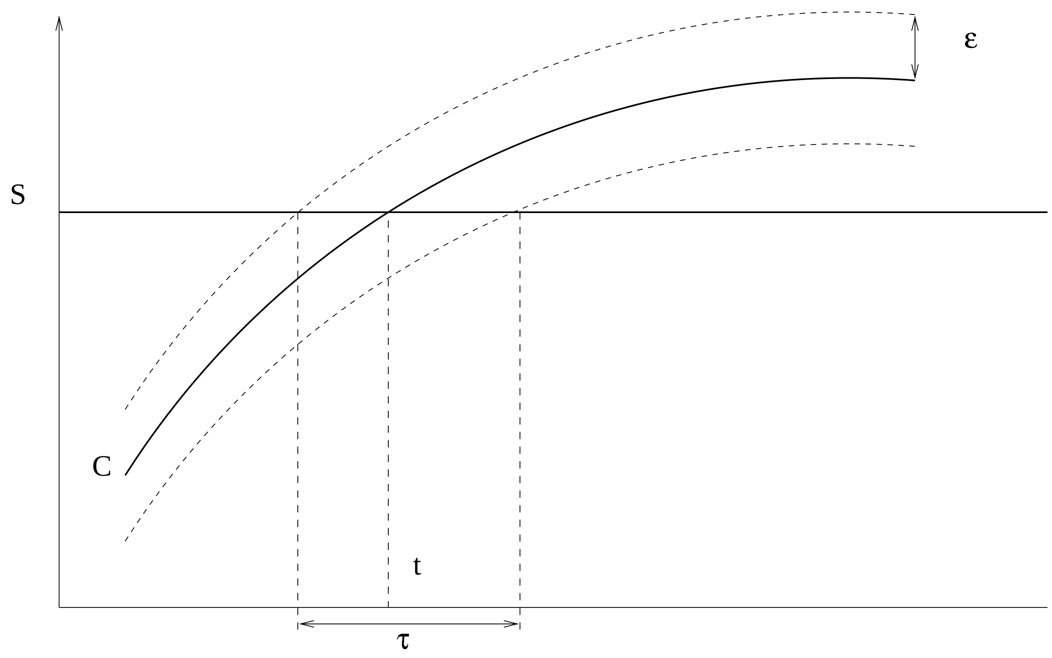
Implementation

State Machine View - SCHED



Mixed Systems

Example : Threshold crossing

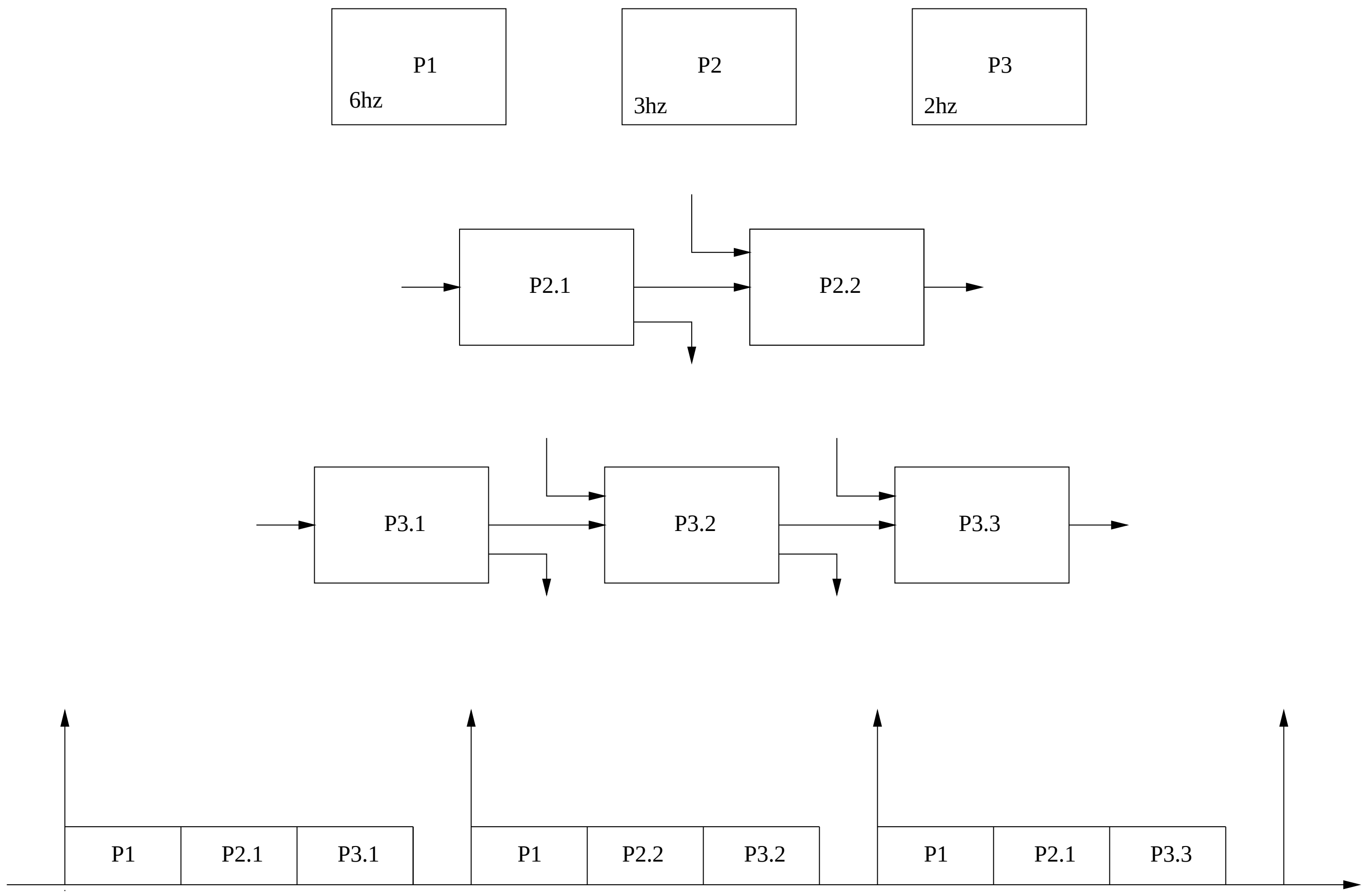


Relates errors and delays : $\tau = \dfrac{2\varepsilon}{|C'(t)|}$

This analysis too should not be skipped

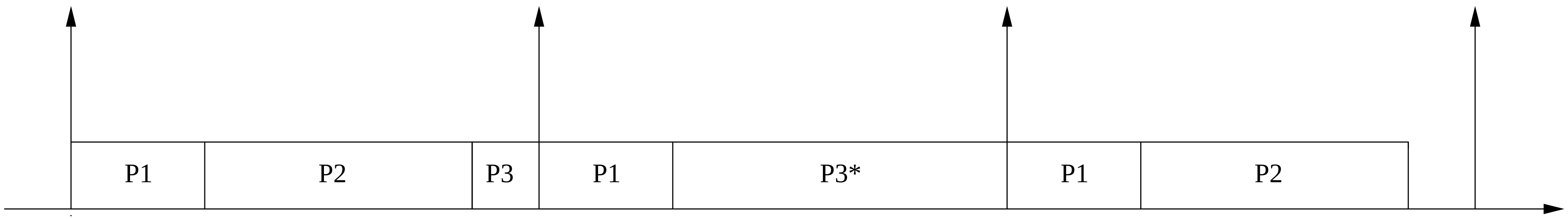
Concurrency

Actual Practices (Airbus)



Concurrency

A Crisys Proposal: earliest deadline preemptive scheduling

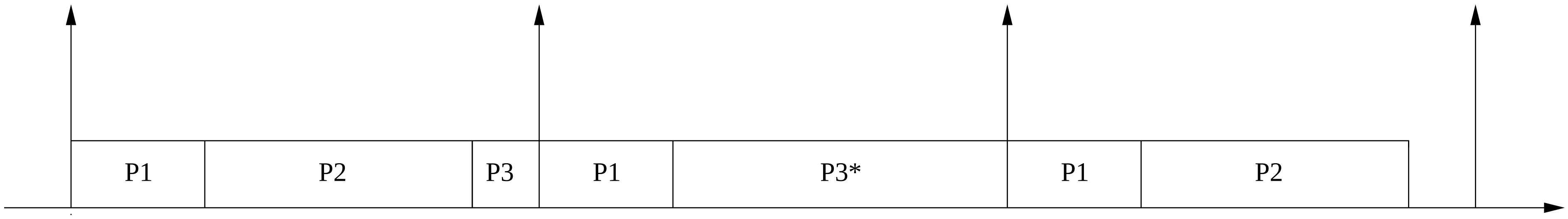


Schedulability condition

$$\sum_{i=1,n} \frac{WET_i}{T_i} < 1$$

Concurrency

A Crisys Proposal: earliest deadline preemptive scheduling



Schedulability condition

$$\sum_{i=1,n} \frac{WET_i}{T_i} < 1$$

Generalizes the synchronous program execution condition

$$WET < T$$

Concurrency

Exact functional semantics is guaranteed as soon as

Slow processes communicate with fast processes through a slow clock unit delay

c	t	f	t	f	t
x	x_0	x_1	x_2	x_3	x_4
$x \downarrow c$	x_0		x_2		x_4
$f(x \downarrow c)$	$f(x_0)$		$f(x_2)$		$f(x_4)$
$z = z_0 \Delta f(x \downarrow c)$	z_0		$f(x_0)$		$f(x_2)$
$(z_0, z) \uparrow c$	z_0	z_0	$f(x_0)$	$f(x_0)$	$f(x_2)$

Fault Tolerance

- **Continuous Computations : Threshold Voting**
  - **Units differ from more than the maximum normal error**

Fault Tolerance

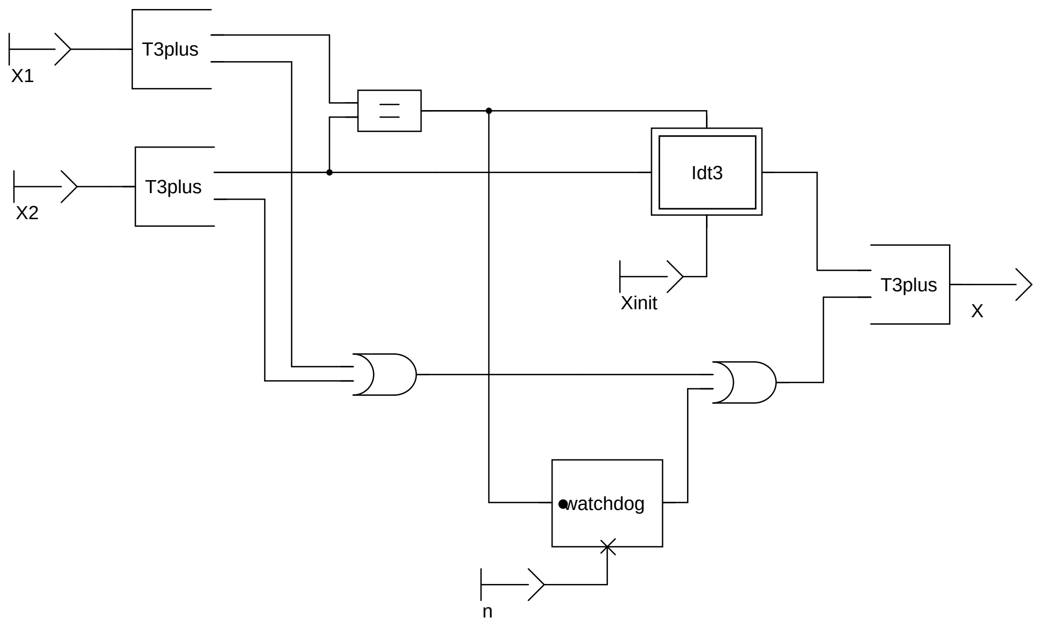
- **Continuous Computations : Threshold Voting**
  - **Units differ from more than the maximum normal error**
- **Combinational : Bounded-Delay Voting**
  - **Units differ from more than the maximum normal delay**

Fault Tolerance

- **Continuous Computations : Threshold Voting**
  - **Units differ from more than the maximum normal error**
- **Combinational : Bounded-Delay Voting**
  - **Units differ from more than the maximum normal delay**
- **Sequential Computations : 2/2 Bounded-Delay Voting**

Bounded-Delay Voters

Net View on vote2_2 - eq_vote2_2



$$n = E\left(\frac{\Delta_{max} - \Delta_{min}}{T_{min}}\right) + 1$$

Sequential Computations

Idea: vote on input and on state

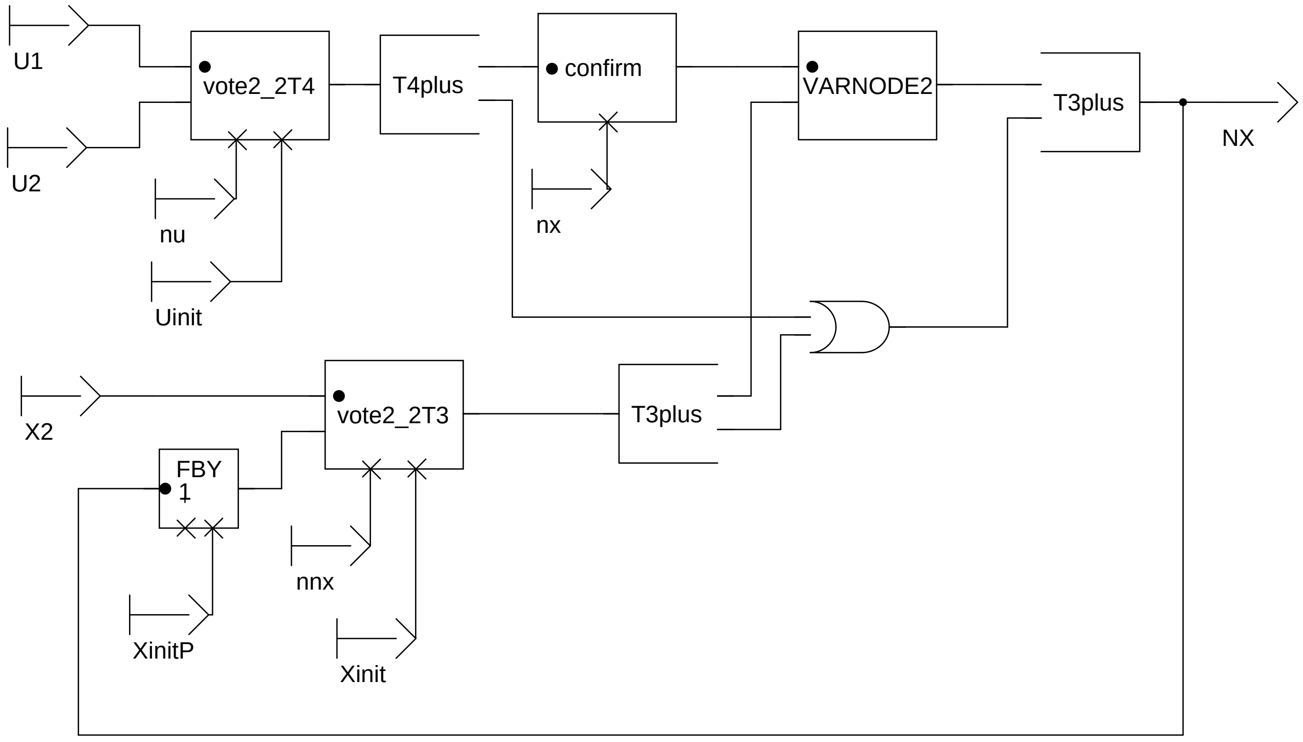
But Byzantine problems

$2/2$ **votes are not sensitive to Byzantine problems:**

- **a bad unit is only compared with a single good one:**
  - **it agrees: it looks good**
  - **it disagrees: a fault is detected.**

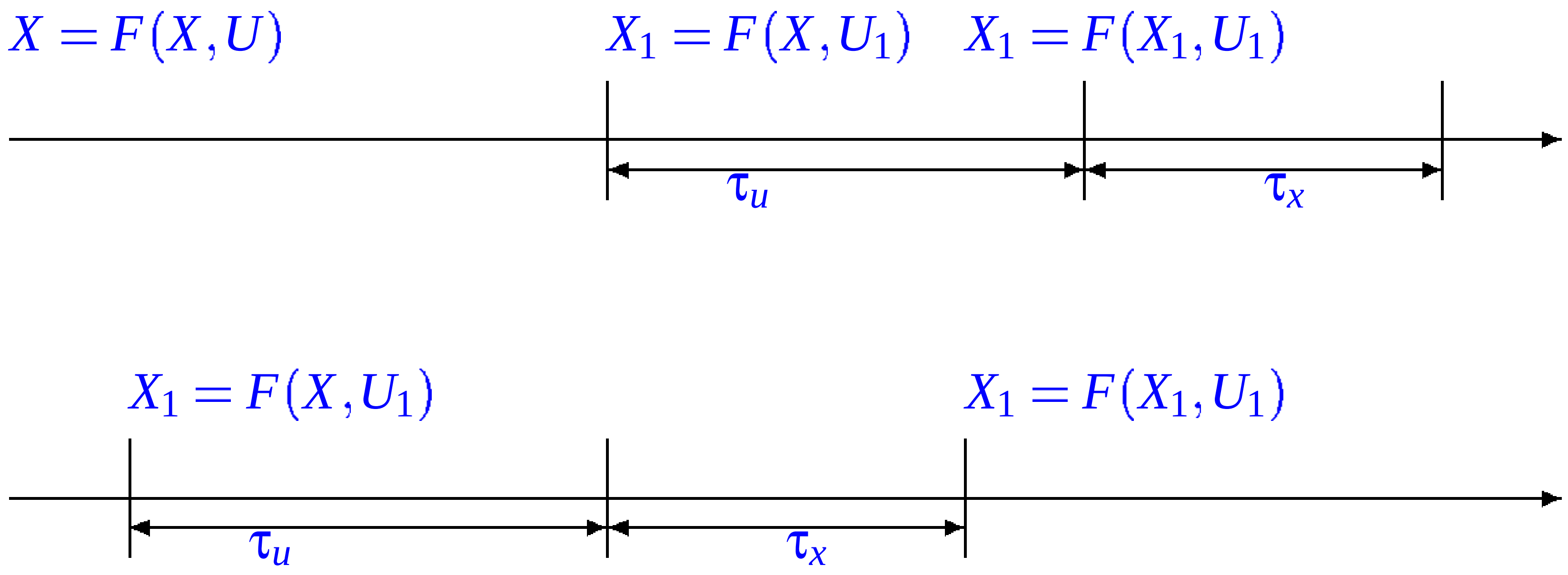
Sequential Computations: 2/2 Sequential Voters

Net View on SeqVote - eq_SeqVote



$$nx = nmax_u + nmax_x \qquad nnx = n \times nx$$

Proof Hints



Conclusion

- **Some insight on techniques used in practice.**
- **maybe useful for designers and certification authorities**

  (Crisys Esprit Project)
- **An attempt to catch the attention of the Computer Science Community on these important problems.**

Questions

- **When are clock synchronization methods useful and more efficient than the ones presented here?**

Questions

- **When are clock synchronization methods useful and more efficient than the ones presented here?**
- **How to safely encompass some event-driven computations within the approach?**

Questions

- **When are clock synchronization methods useful and more efficient than the ones presented here?**
- **How to safely encompass some event-driven computations within the approach?**
- **Are there linguistic ways to robustness (synchronous-asynchronous languages)?**

Questions

- **When are clock synchronization methods useful and more efficient than the ones presented here?**
- **How to safely encompass some event-driven computations within the approach?**
- **Are there linguistic ways to robustness (synchronous-asynchronous languages)?**
- **Is there a common framework encompassing both theories?**

continuous	discrete
uniformly continuous signals	uniform bounded variability
uniformly continuous functions	robust systems
unstable systems	sequential non robust systems