ARTIST2 - MOTIVES
Trento - Italy, February 19-23, 2007

Modeling and Design of Heterogeneous Systems

Composition and Transformation of Heterogeneous Real-Time Systems

Benoît Caillaud

IRISA / INRIA-Rennes, France

Outline

- Models of Computation and Communication (MoCC)
- Two types of MoCC heterogeneity: Architectural heterogeneity & design flow heterogeneity
- Synchronous vs. asynchronous MoCCs
- Unifiying synchronous, time-triggered & losely time-triggered MoCCs: Introducing tag systems
- From an algebra of tags to tag machines
- The power of tag system: An analysis of communication by sampling in time-sensitive distributed systems

Bibliography (1/2)

[1] A. Benveniste, B. Caillaud, L. Carloni, P. Caspi, A. Sangiovanni-Vincentelli. Heterogeneous Reactive Systems Modeling: Capturing Causality and the Correctness of Loosely Time-Triggered Architectures (LTTA). In Proceedings of the Fourth ACM International Conference on Embedded Software, EMSOFT'04, Volume September, September 2004.

[2] A. Benveniste, B. Caillaud, L. P, Carloni, A. L. Sangiovanni-Vincentelli. Tag Machines. In Proceedings of the fifth ACM International Conference on Embedded Software (Emsoft), Pages 255-263, Jersey City, NJ, USA, September 2005.

[3] A. Benveniste, B. Caillaud, L. Carloni, P. Caspi, A. Sangiovanni-Vincentelli. Communication by Sampling in Time-Sensitive Distributed Systems. In Proceedings of the Sixth Annual ACM Conference on Embedded Software, EMSOFT'06, 2006.

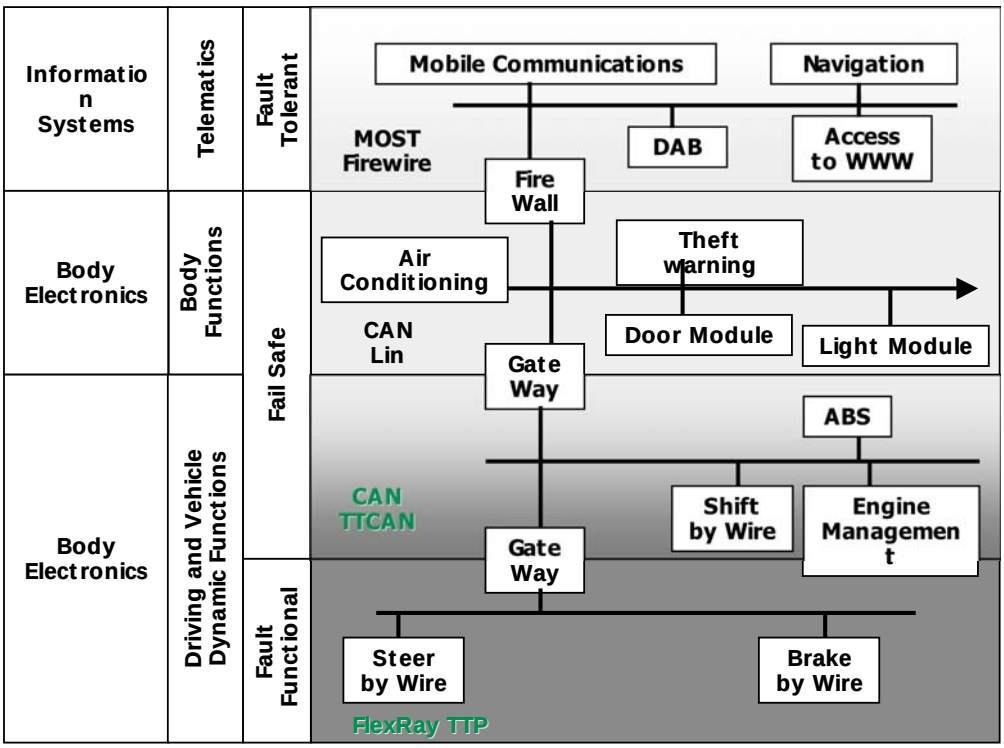
[4] S. A. Edwards, O. Tardieu. SHIM: A Deterministic Model for Heterogeneous Embedded Systems. In Proceedings of the ACM Conference on Embedded Software, EMSOFT'05. Jersey City, NJ, September 2005.

[5] D. Potop-Butucaru, B. Caillaud, A. Benveniste. Concurrency in Synchronous Systems. Formal Methods in System Design, 28(2), March 2006.

Bibliography (2/2)

[6] D. Potop-Butucaru, B. Caillaud. Correct-by-construction asynchronous implementation of modular synchronous specifications. In Proceedings of the Fifth International Conference on Application of Concurrency to System Design, ACSD 2005, 2005.

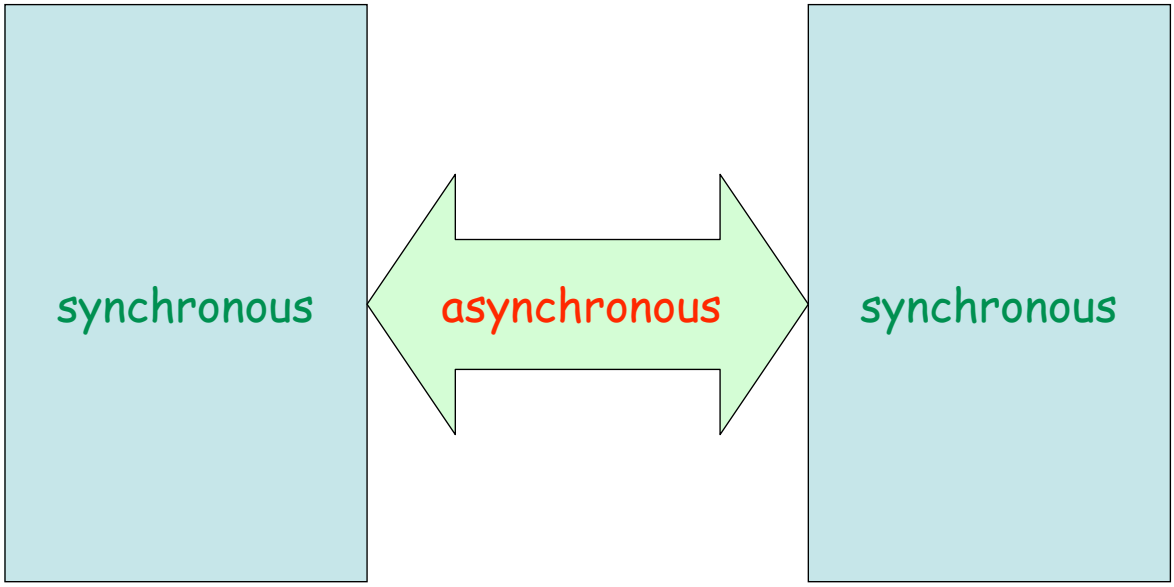
Heterogeneous architectures: automotive electronics



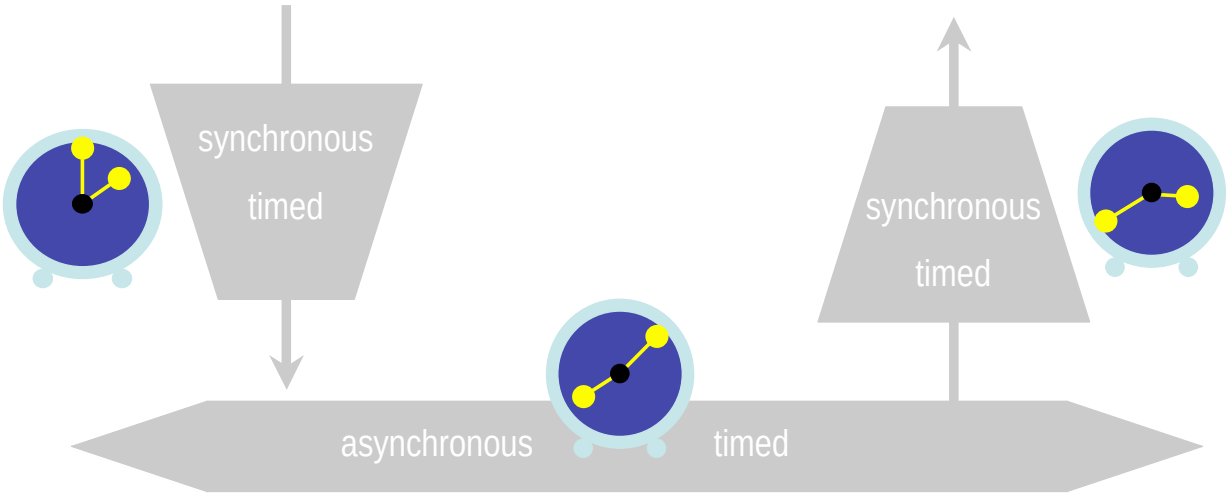
Heterogeneous models: design flow in automobile or aeronautics

- **Systems modeling (UML, MDA,…SysML)**
  - **Loose model of computation & communication**
- **Matlab/Simulink/Stateflow**
  - **Continuous time basis**
- **Statemate, synchronous languages**
- **Late assembly of imported functions**
  - **Can be in C…**
  - **Depends on OS and execution infrastructure**
- **Reuse models of deployment architecture (components)**
  - **CAN, ARINC, TTA,…**

Classes of heterogeneous systems: GALS

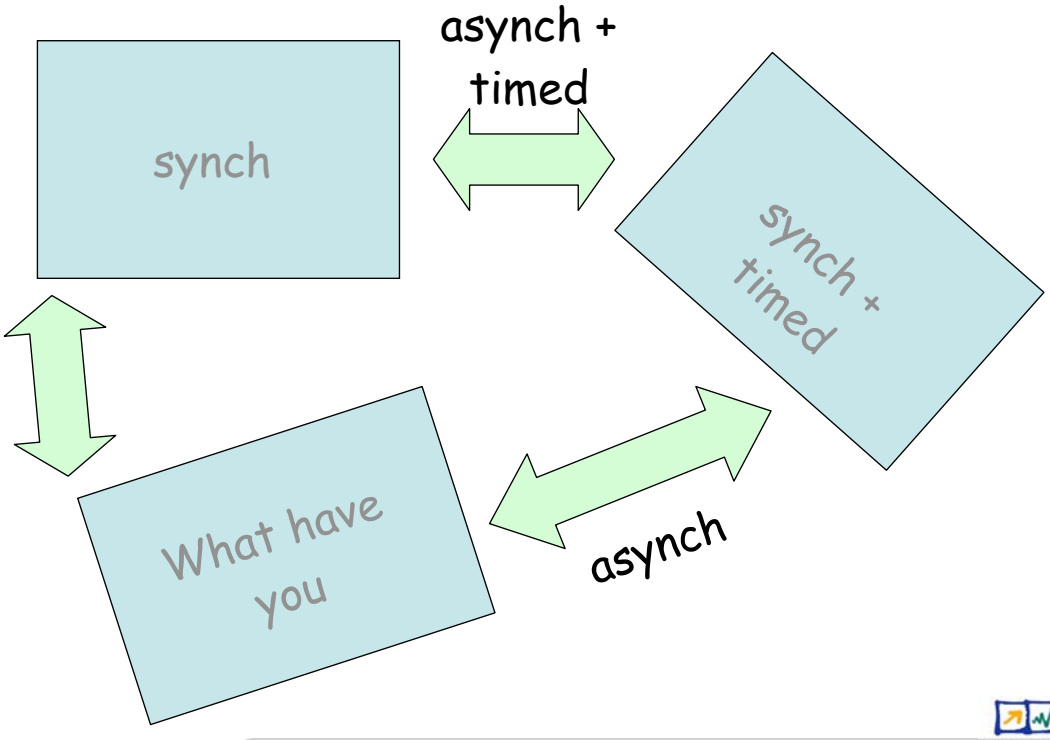


Classes of heterogeneous systems: LTTA



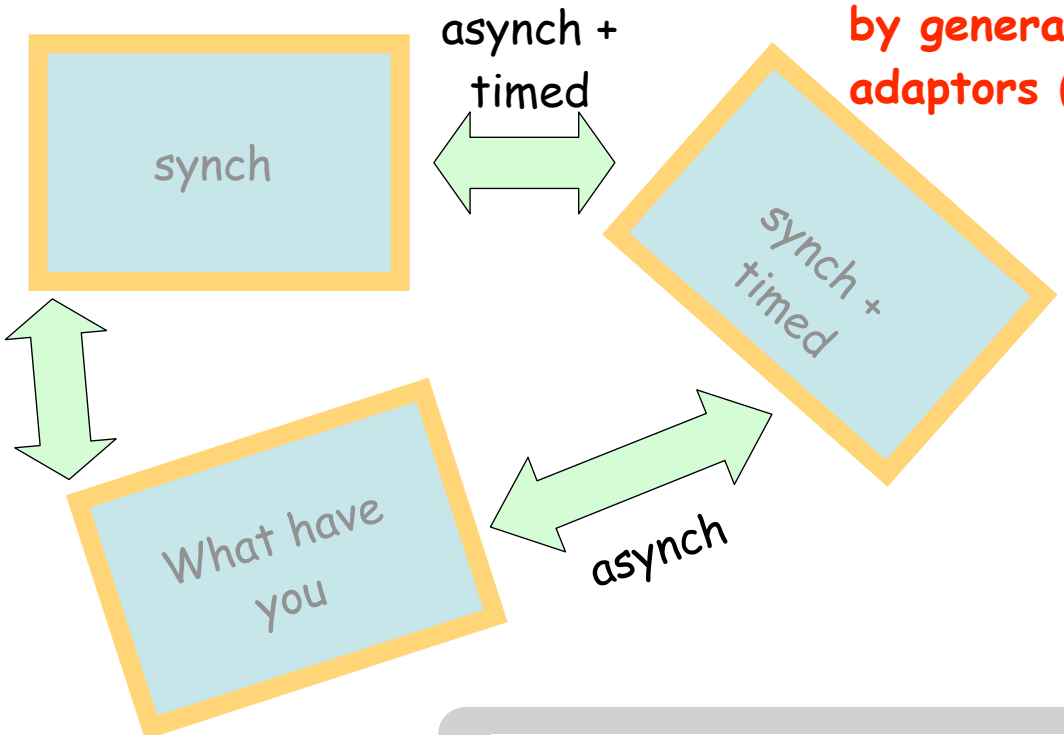
(bounded delay)

How to blend heterogeneous models while "preserving semantics"?



How to blend heterogeneous models while "preserving semantics"? idea:

by generating proper adaptors (wrappers)



Classes of heterogeneous systems: SHIM programs [S. Edwards, Emsoft'05]

- **SHIM, a language proposed by S. Edwards for HW/SW integration:**
  - ***"Rather than propose a completely new semantics for SHIM, I chose to integrate two well-known, well-established semantics: C-like imperative semantics for the SW portion of the design and RTL semantics for the HW"***

Classes of heterogeneous systems: SHIM programs

```
module timer {
  shared uint:32 counter;
  hw void count() {
    counter = counter + 1;
  }
  out void reset_timer() {
    counter = 0;
  }
  out uint get_time() {
    return counter;
  }
}
```

SHIM module

hw

sw

sw

Classes of heterogeneous systems: SHIM programs

- **SHIM, a language proposed by S. Edwards for HW/SW integration:**
  - ***"A particularly glaring issue is that SHIM models are not easy to simulate. This is due to the models themselves: the two domains run asynchronously and while the HW is timed, the SW effectively is not, meaning that the behavior of the system may be nondeterministic or at least very difficult to predict without careful modeling of SW timing, such as by using an instruction-set simulator"***

Two simple questions

1. How to keep control of the meaning of heterogeneous programs such as our SHIM example?

2. How to blend heterogeneous models while "preserving semantics"? We discuss first the GALS case

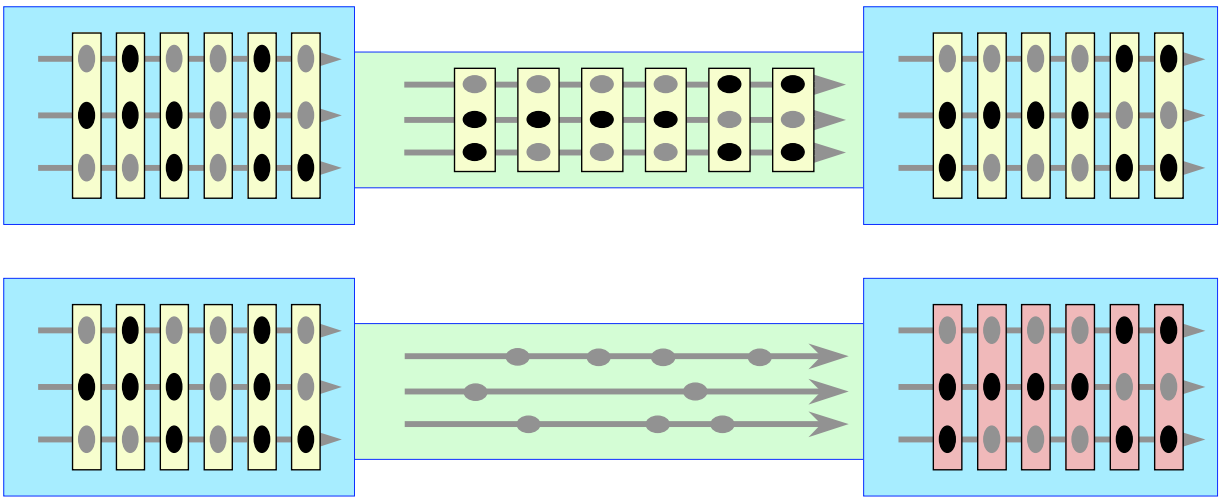
How to keep control of the meaning of heterogeneous programs such as SHIM?

```
module timer {
  shared uint:32 counter;

  hw void count() {
    counter = counter + 1;
  }

  out void reset_timer() {
    counter = 0;
  }

  out uint get_time() {
    return counter;
  }
}
```

hw : what can be a common synchro domain with sw?

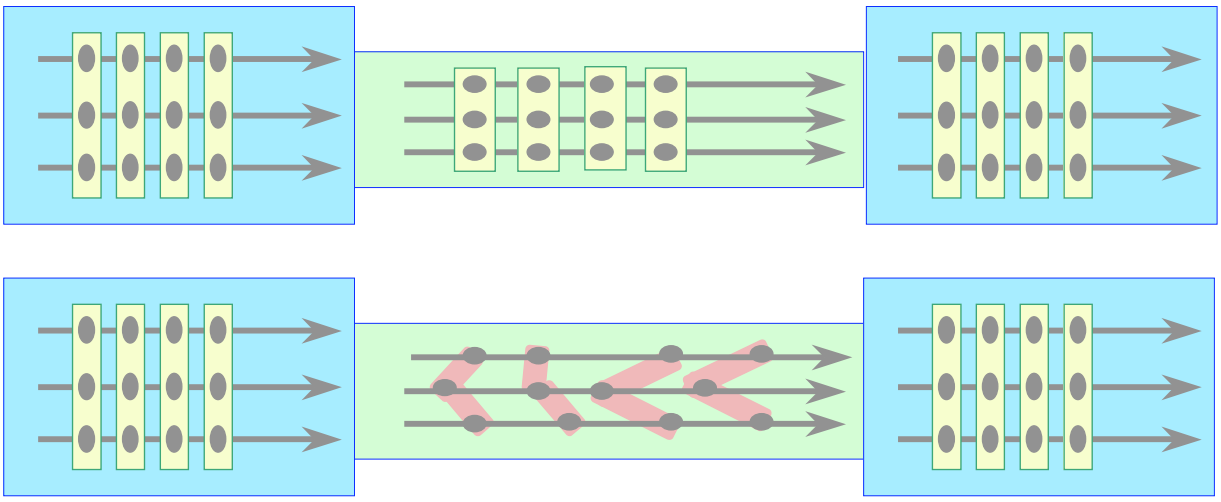
sw : what can be a common synchro domain with hw?

Adaptors for GALS: informal discussion



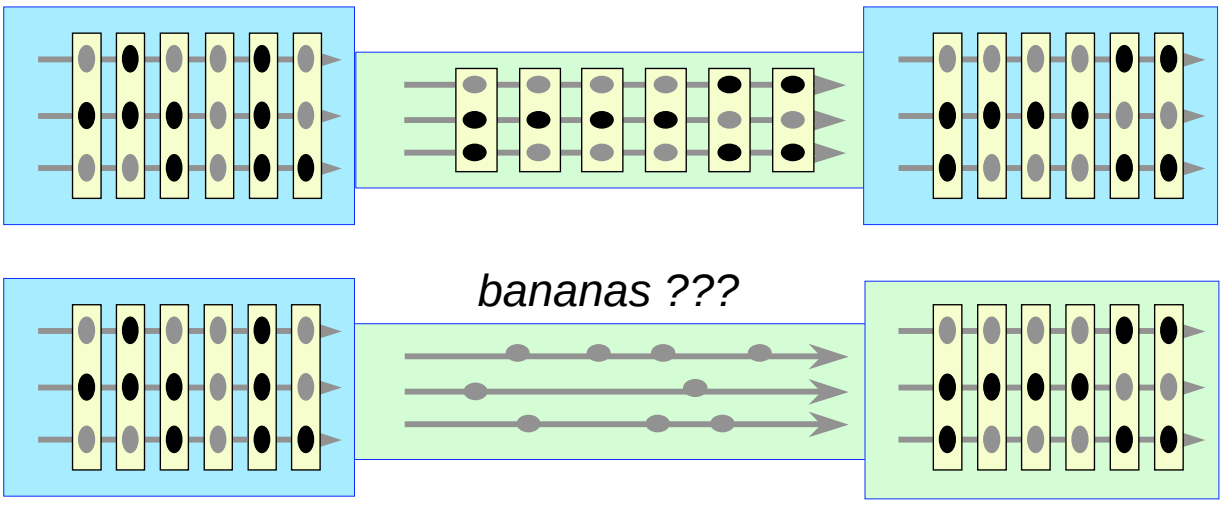
How to ensure that the two components do not behave in different ways difference when moving from synchrony to GALS?

Adaptors for GALS: informal discussion



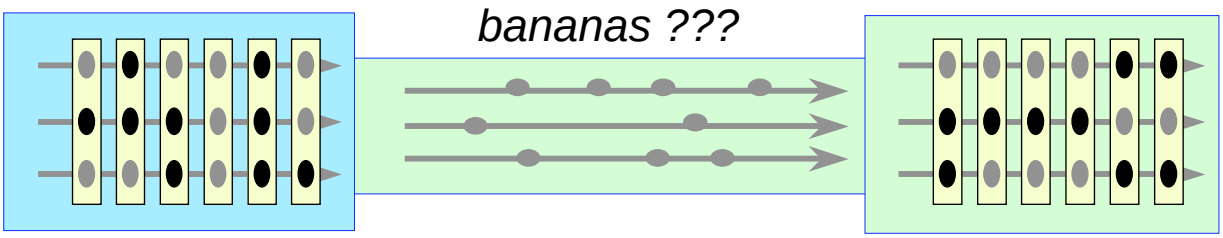
How to ensure that the two components do not see the difference when moving from synchrony to GALS? Easy if known to be single-clocked: bananas ! (latency insensitive designs and many handshake-based asynchronous HW designs make use of bananas; handshake / adaptor)

Adaptors for GALS: informal discussion



How to ensure that the two components do not see the difference when moving from synchrony to GALS? In general *bananas* can be many!

Adaptors for GALS: informal discussion



- In general *bananas* can be many!
- Naive solution: attach to each wire a boolean_clock that is present in each reaction and tells you the presence/absence for the wire (cf. hardware); then, apply previous solution
- Poor if slow/fast communications between components, because all boolean clocks must be communicated at fastest pace
- This overhead has been identified by HW people: they perform heuristic post-processing to reduce the number of handshake protocols needed. Our approach makes this algebraic and systematic

Synchronous vs. asynchronous MoCCs: Outline

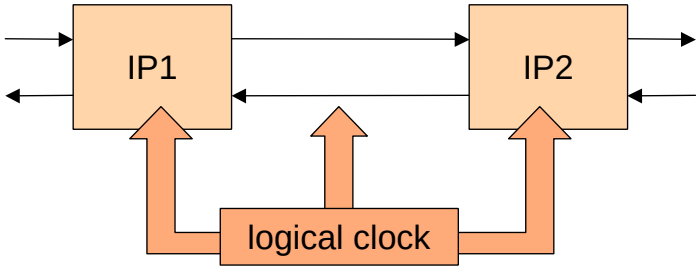
- Motivation: Asynchronous implementation of synchronous specifications
  - GALS architectures
  - Desired efficient implementation
- Formal model
  - Correctness
- Correctness criteria
  - Microstep weak endochrony
  - Microstep weak isochrony
- Conclusion

Synchrony, asynchrony, GALS

- Synchronous specification
  - Global logical clock ⇒ ease of specification & verification
  - Popular, efficient tools for system design (digital circuits, safety-critical systems)
- Distributed implementation
  - Distributed software, complex digital circuits (SoC/NoC), heterogeneous systems
  - Loosely-connected components (asynchronous FIFOs...)
- GALS architectures = good implementation model
  - Synchronous components, asynchronous communication
  - Problem: preserve semantic consistency between synchronous specification and GALS implementation

What we want (1/2)

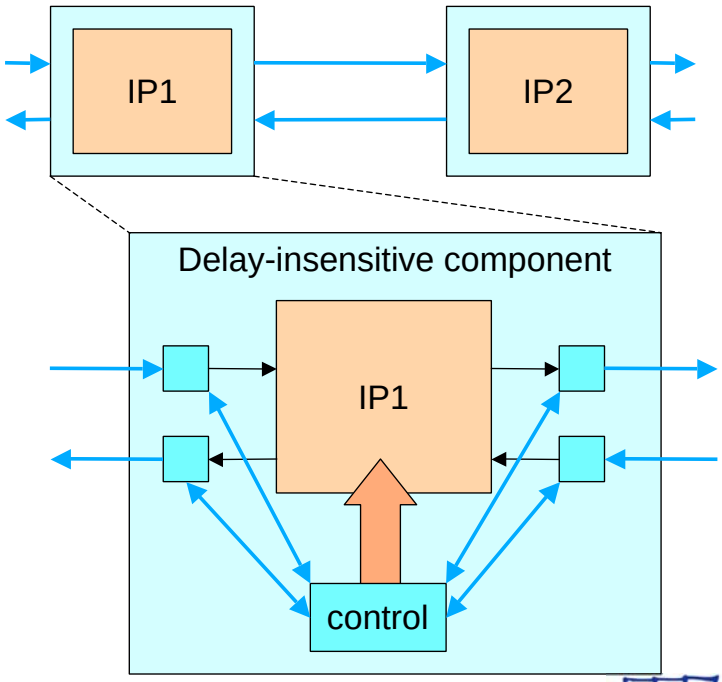
1. Take a modular synchronous specification



What we want (2/2)

1. Take a modular synchronous specification
2. Replace comm. with asynchronous FIFOs, wrappers
3. Preserve:
   - ✓ Functionality
   - ✓ Correctness
     - ⇒ No “extra” traces
     - ⇒ No deadlocks

   (Kahn processes)
   - ✓ Parallelism



Related work (1/2)

- Latency-insensitive systems
  - Carloni & Sangiovanni-Vincentelli (1999)
  - Goal: independence from communication delays
  - Global synchrony: system speed = slowest component speed
- Endo/isochronous systems
  - Benveniste, Caillaud, Le Guernic (1999)
    - Variation: Generalized latency-insensitive circuits (Singh, Theobald, 2003)
  - Goals:
    - minimize communication
    - maximize concurrency, independence between system components
  - Not compositional!

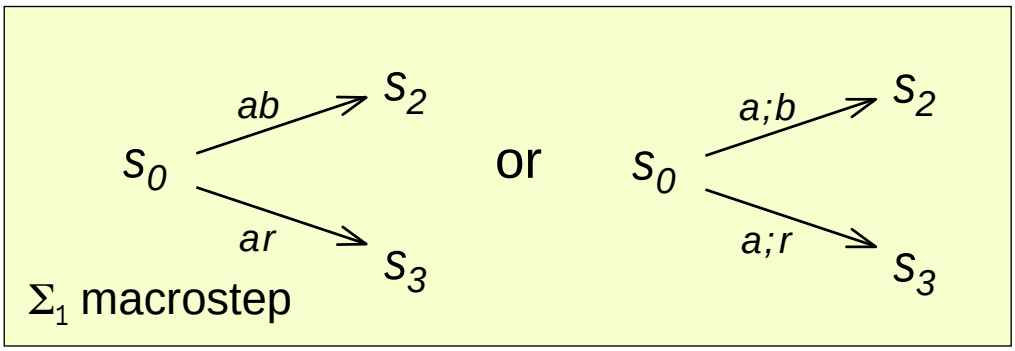
Previous work (2/2)

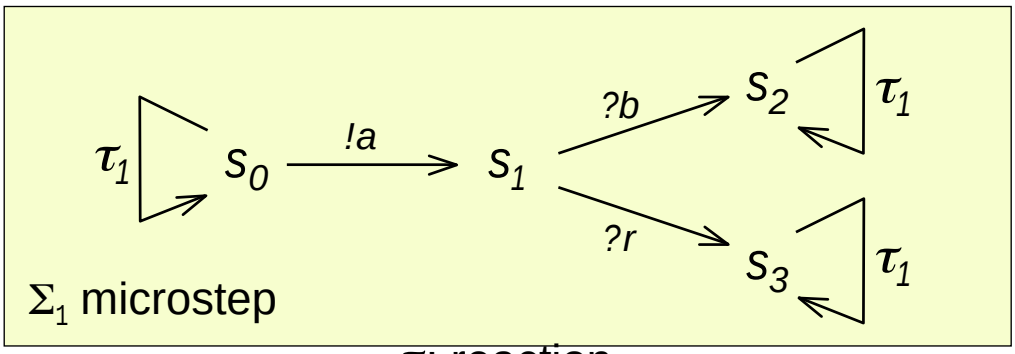
- Weak endo/isochronous systems
  - Potop, Caillaud, Benveniste (2004)
  - Goals:
    - further minimize communication by exploiting intra-component concurrency
    - **Compositionality !**
  - Synchronous Mazurkiewicz traces
  - Does not handle causality and communication deadlocks
- This work: microstep weak endo/isochronous systems
  - Goal: take into account causality and composition through read/write mechanisms

Approach presented

- Define a model such that:
  - Criteria (sufficient conditions) for the existence of delay-insensitive wrappers that preserve the semantics without adding new signals
  - Connecting through FIFOs the resulting components produces a semantics-preserving, deadlock-free GALS implementation
- How to make given components satisfy the sufficient conditions?
  - Possible solutions:
    1. Encode (part of) the “absent” events (Carloni et al.)
    2. Add new signals
    3. Decide that none is necessary due to environment constraints
- Efficient sw/hw implementation
  - Sync./async. synthesis techniques, GALS-specific communication schemes, etc.

Microstep vs. Macrostep





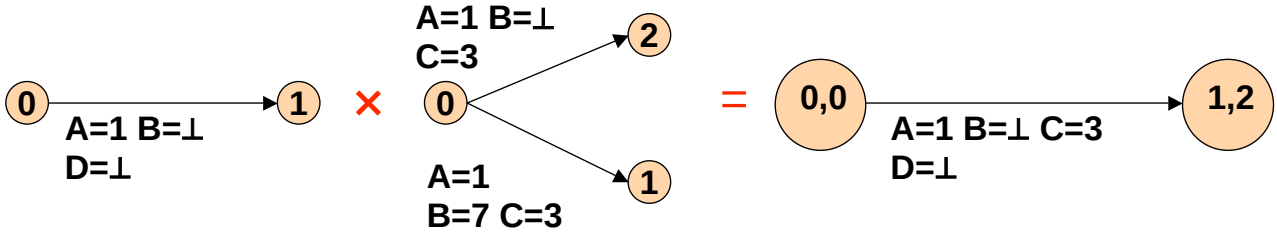
τ: reaction

The model: basic definitions

- The basics: (incomplete) automata

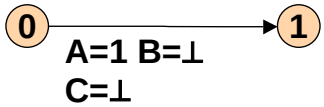
$$\Sigma = (S, s_0, V, \rightarrow),\ \rightarrow \subset S \times L(V) \times S,\ L(V) = \prod_{v \in V}(D_v \cup \perp)$$

  - Composition by synchronized product:



  - Renaming operator:

$\Sigma_1[D/C]$:



A=1 B=⊥ C=3 ≡ A=1 C=3

A=1 C=3 ≤ A = 1 B=7 C=3

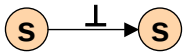
A=1 C=3 – A=1 = C=3

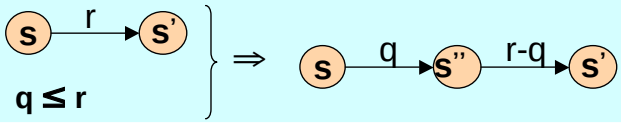
A=1 C=3 ; B=2 ; ;

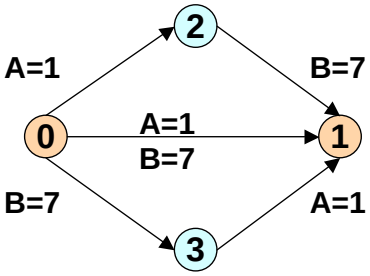
A=1 C=3 ; B=2 ; ; ⩽ A=1 C=3 ; B=2 ; ; A=2;

- Labels
- Finite runs:

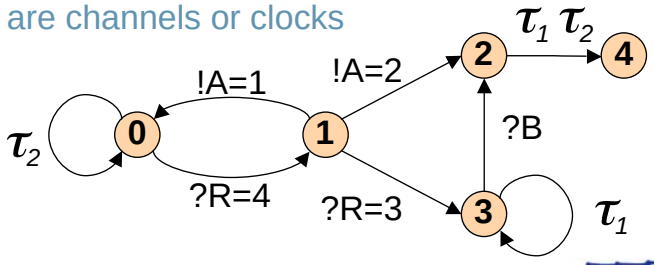
The model: basic definitions

- Generalized concurrent transition systems(GCTS)
  - Void transitions: 

  - Down closure: 

- Example: 

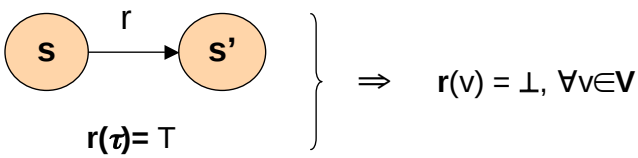
The model: I/O transition systems

- Point-to-point communication:
  - Broad/Multicast can be simulated…
  - Communication channels: c = (!c,?c) $D_{!c}=D_{?c}=D_c$
  - Dissociate emission from reception!
- Logical (local) clocks: τ τ_1… of domain $D_{clk}=\{T\}$
- I/O transition system:
  - GCTS where all variables are channels or clocks
  - Example: 

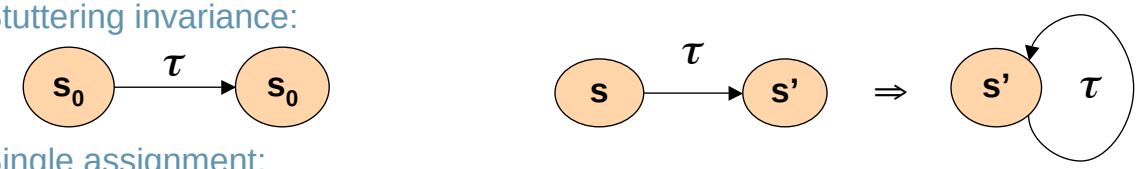
The model: synchronous systems

Synchronous system: $\Sigma = (S, s_0, V, \tau, \rightarrow)$, I/O transition system, 1-clock, s.t.:

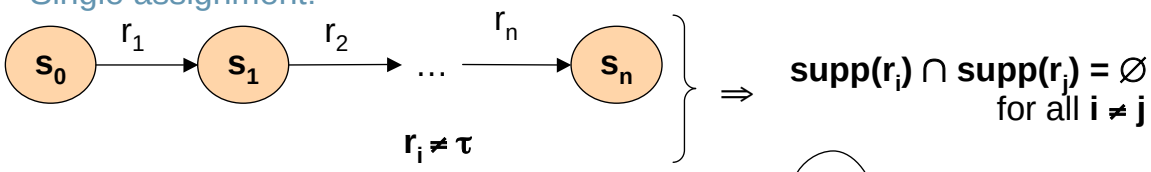
1. Clock transitions:



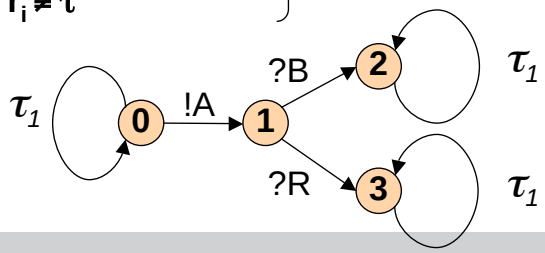
2. Stuttering invariance:



3. Single assignment:



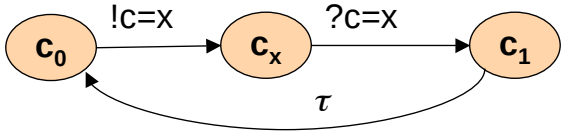
Example:



The model : composition

- Synchronous 1-place register:

SFIFO(c, τ):



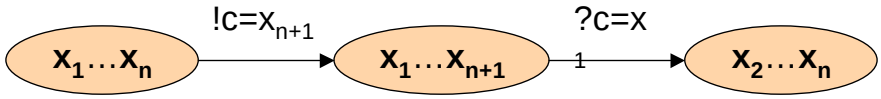
for all $x \in D_c$

- Synchronous composition (on clock τ) :

$$\Sigma_1|\Sigma_2 = \Sigma_1[\tau_1/\tau] \times \Sigma_2[\tau_2/\tau] \times \mathrm{SFIFO}(c_1, \tau) \times \ldots \times \mathrm{SFIFO}(c_n, \tau)$$

- Asynchronous FIFO:

AFIFO(c):

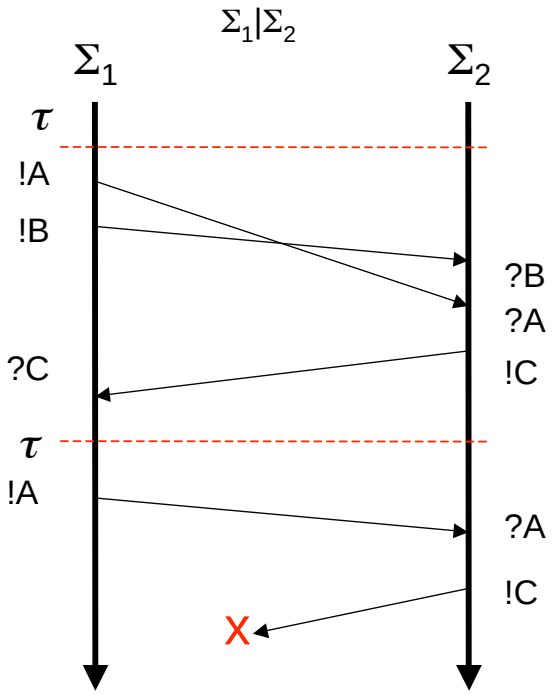


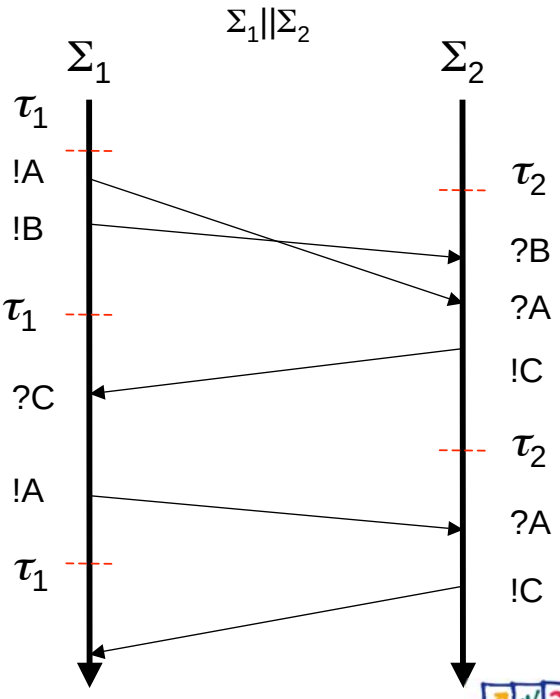
for all $x_1,\ldots,x_n,x_{n+1} \in D_c$

- Asynchronous composition:

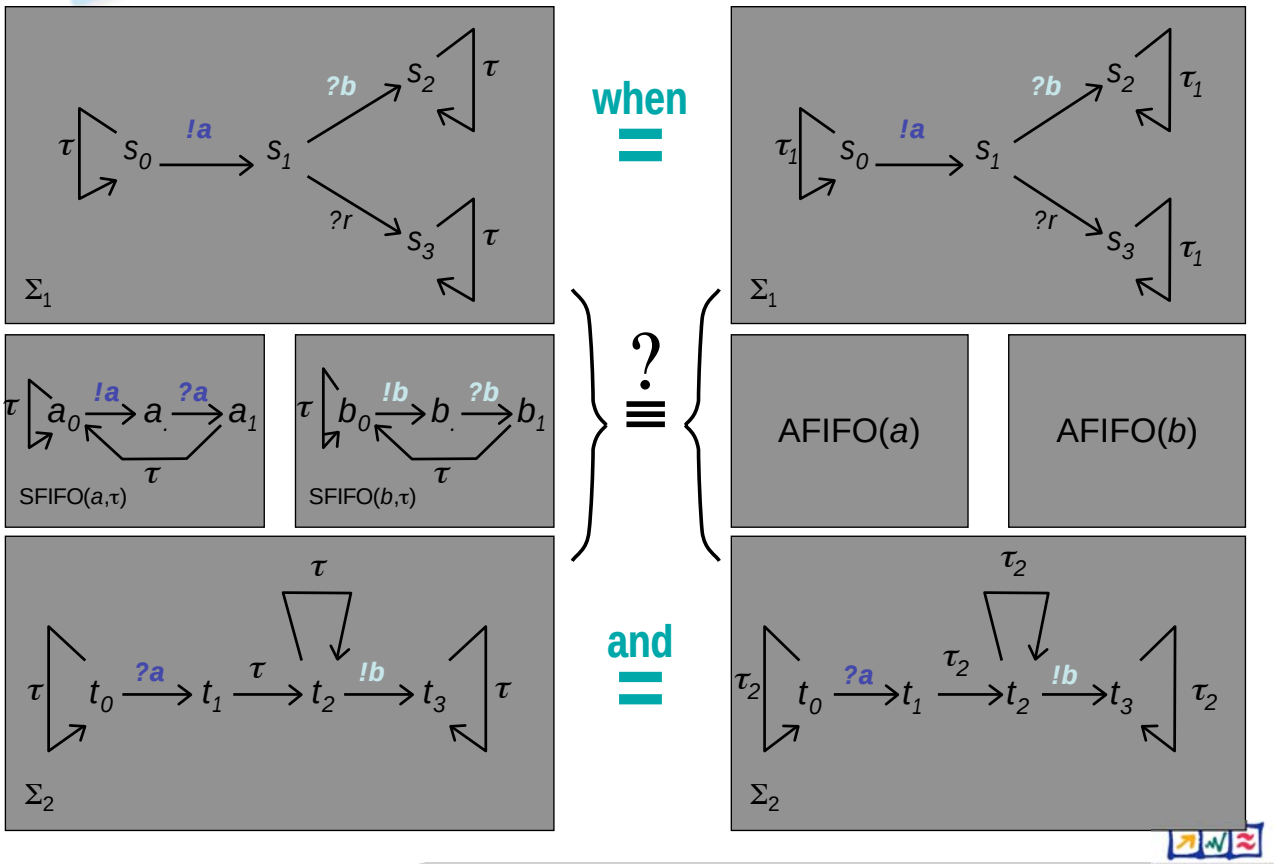
$$\Sigma_1||\Sigma_2 = \Sigma_1 \times \Sigma_2 \times \mathrm{AFIFO}(c_1) \times \ldots \times \mathrm{AFIFO}(c_n)$$

The model : composition

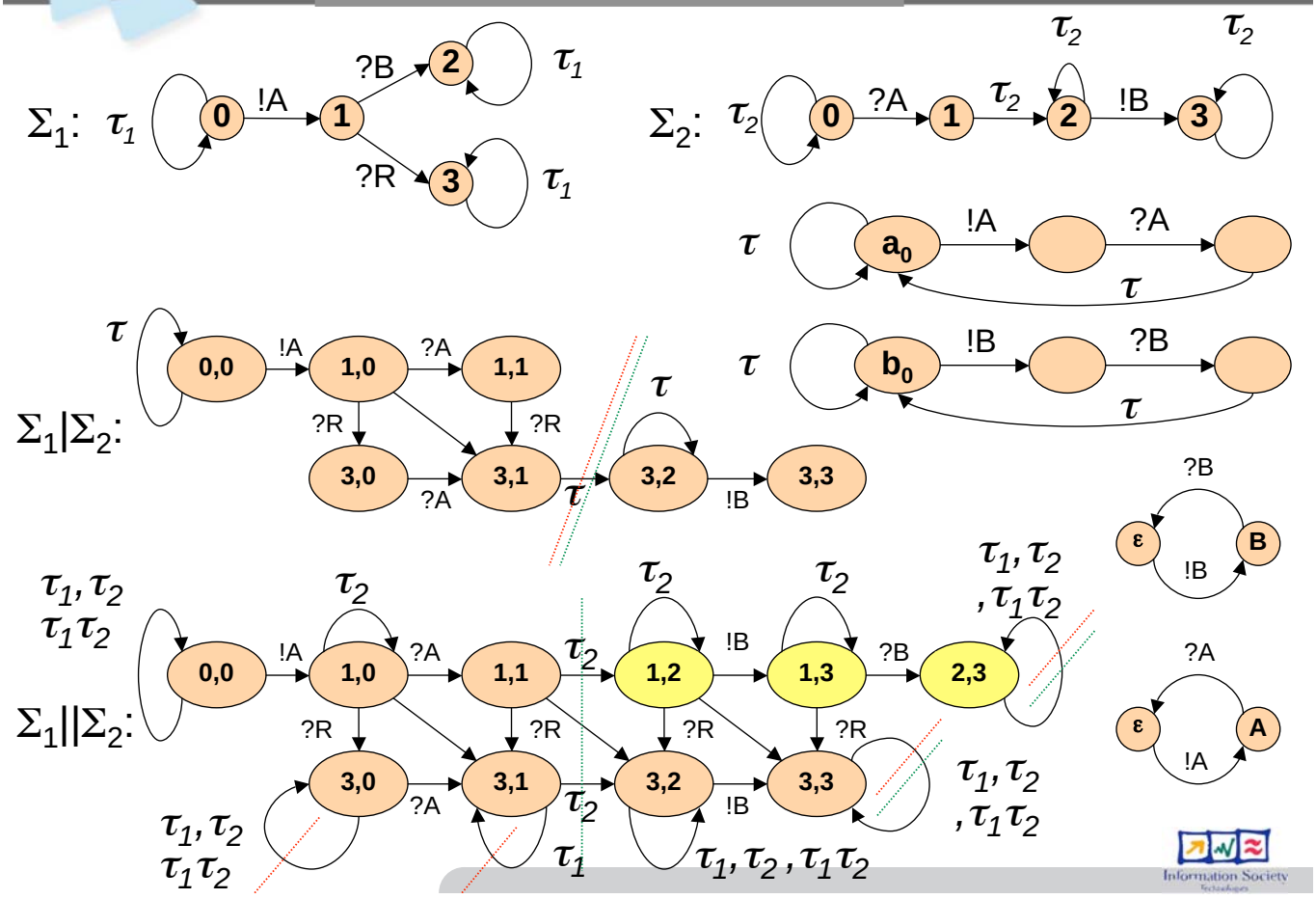




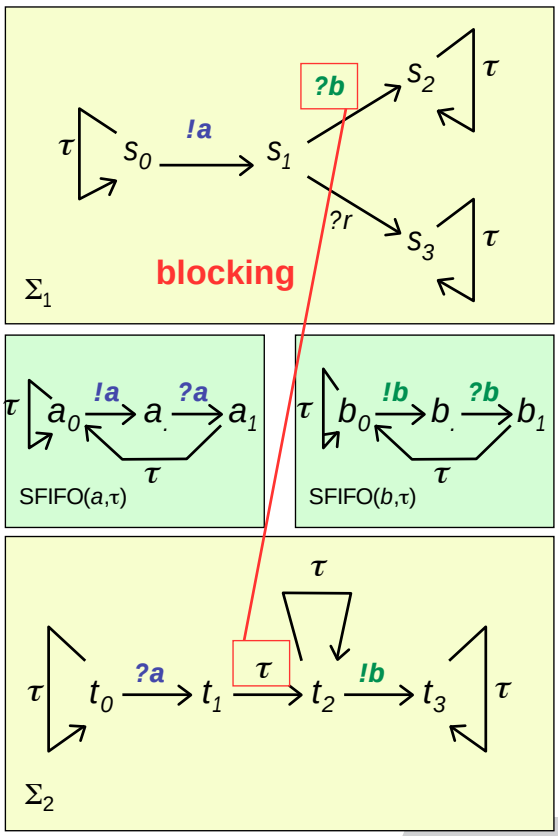
Example (1/3)



Example (2/3)



Preserving semantics $\Sigma_1|\Sigma_2 \equiv \Sigma_1||\Sigma_2$ two conditions required



weak endochrony: reconstruct reactions from asynch envir^t, *up to concurrency*

causal correctness (isochrony): *ensure that composition is non-blocking*

weak endochrony: reconstruct reactions from asynch envir^t, *up to concurrency*

Preserving semantics $\Sigma_1|\Sigma_2 \equiv \Sigma_1||\Sigma_2$ two conditions required

compositional

weak endochrony: reconstruct reactions from asynch envir[t], *up to concurrency*

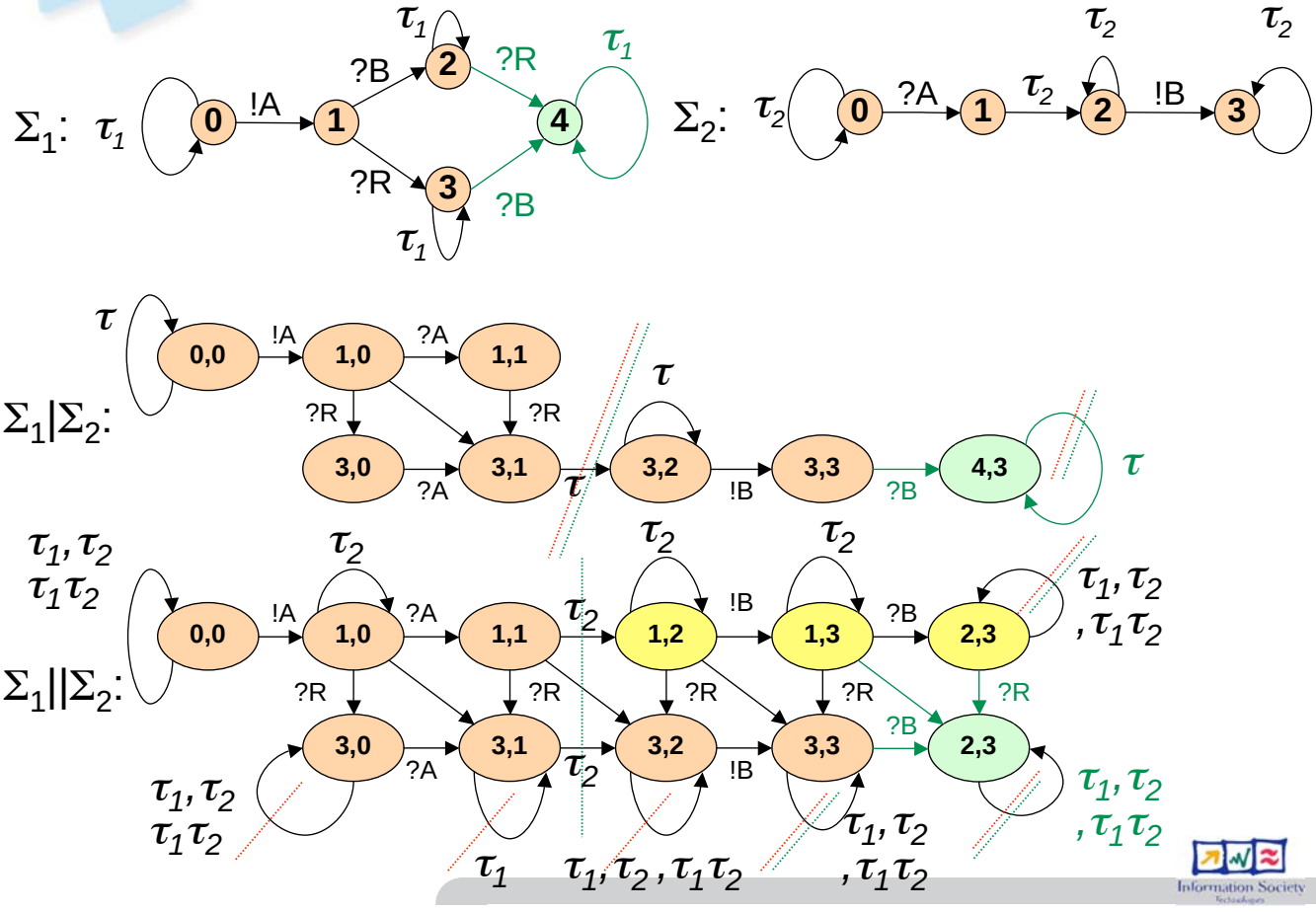
global

causal correctness (isochrony): *ensure that composition is non-blocking*

compositional

weak endochrony: reconstruct reactions from asynch envir[t], *up to concurrency*

Back to the example (3/3)



Correctness

- Some notations:

  !A=1 ; τ_1; ?A=1 ; τ_2 ; !C=3 ; ~ !A=1 ?A=1 ; $\tau_1\tau_2$; !C=3 ; τ_2 ;

  !A=1 ; τ_1; τ_2 ; !C=3 ; ≤ !A=1 ?A=1 ; τ_1 τ_2 ; !C=3 ; τ_2 ;

- Formal correctness criterion

> $\Sigma_1||\ldots||\Sigma_n$ is correct w.r.t. $\Sigma_1|\ldots|\Sigma_n$ if
>
> for all $s \in RSS(\Sigma_1|\ldots|\Sigma_n)$ and all $\phi \in Traces_{\Sigma1||\ldots||\Sigma n}(s)$
>
> there exist $\alpha \in Traces_{\Sigma1||\ldots||\Sigma n}(s)$ and $\beta \in Traces_{\Sigma1|\ldots|\Sigma n}(s)$
>
> such that $\phi \leq \alpha$ and $\alpha \sim \beta$

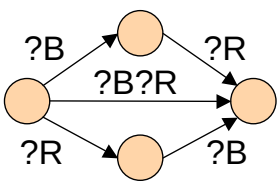
- Intuition: every trace of $\Sigma_1||\ldots||\Sigma_n$ can be completed to one that is equivalent to a synchronous trace

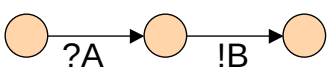
Microstep weak endochrony

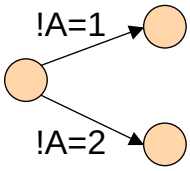
- Compositional delay-insensitivity criterion (signal absence information is not needed)
- Axioms (part 1):

A1: Determinism

A2: In every state, non-clock transitions sharing no common variable are independent







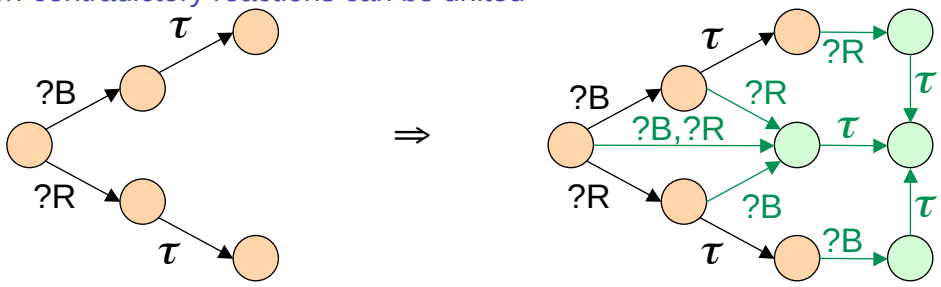
Microstep weak endochrony

- Axioms (continued):

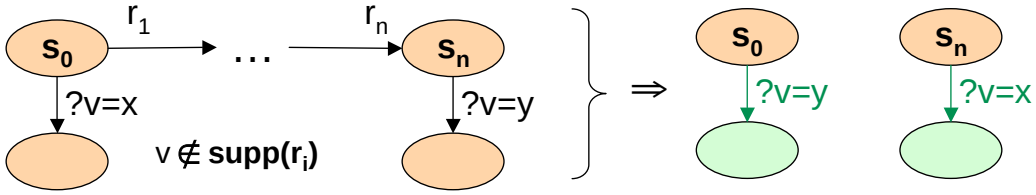
  A1: Determinism

  A2: In every state, non-clock transitions sharing no common variable are independent

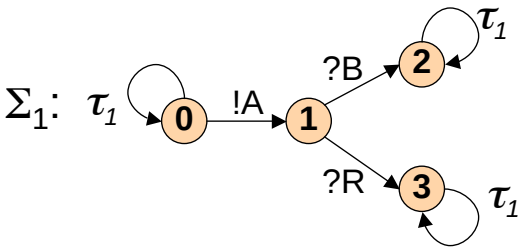
  A3: Non-contradictory reactions can be united



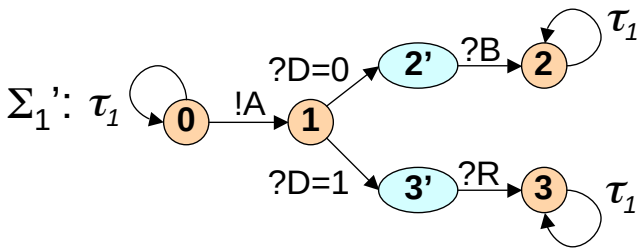
  A4: Conflict does not change with time



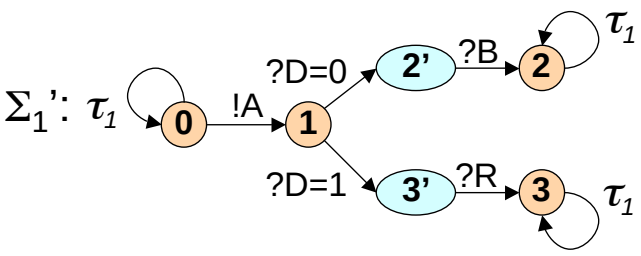
Example (1/3)

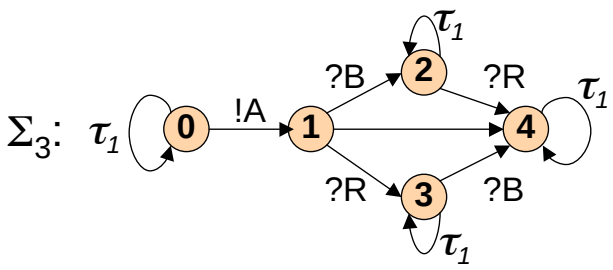


Example (2/3)



Example (3/3)





Weak non-blocking property

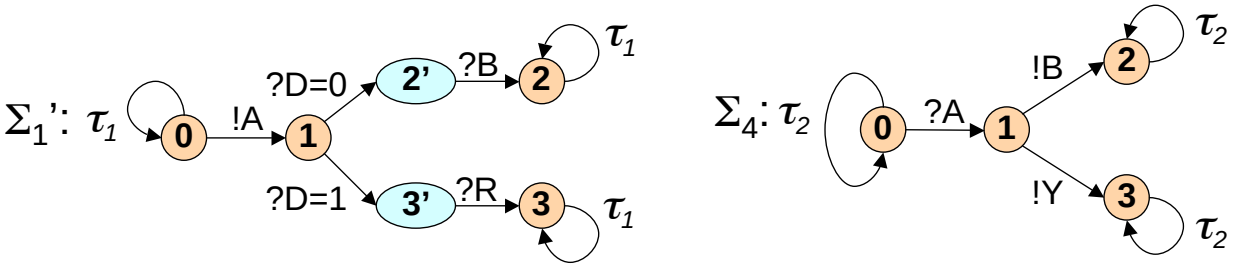
- Weak non-blocking

> $\Sigma_1,\ldots,\Sigma_n$ are weakly non-blocking iff
>
> for all $s \in RSS(\Sigma_1|\ldots|\Sigma_n)$ and all $\phi \in Traces_{\Sigma 1|\ldots|\Sigma n}(s)$
>
> maximal and containing no clock transition, there exists
>
> $\alpha \in Traces_{\Sigma 1|\ldots|\Sigma n}(s)$ non-void such that
>
> $\alpha \leqslant \phi$ and $\alpha;\tau \in Traces_{\Sigma 1|\ldots|\Sigma n}(s)$

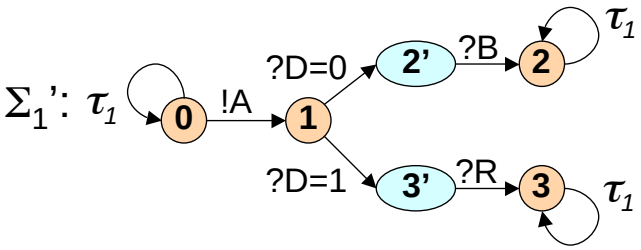
- Semantics preservation criterion

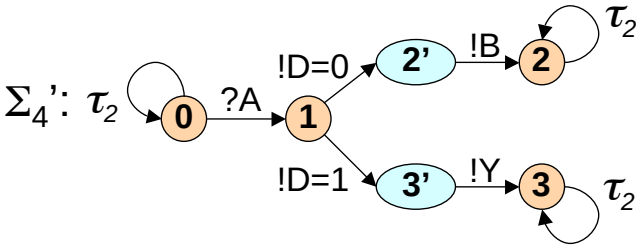
> If $\Sigma_1,\ldots,\Sigma_n$ are weak non-blocking and weak
>
> endochronous, then $\Sigma_1||\ldots||\Sigma_n$ is correct w.r.t. $\Sigma_1|\ldots|\Sigma_n$

Example (1/1)



Example (1/2)





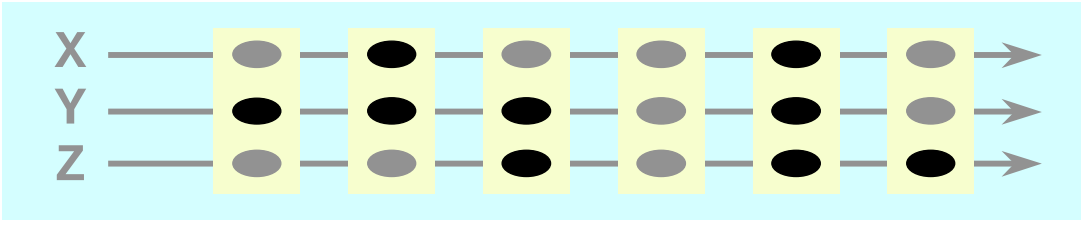
Conclusion

- Decidable criteria for GALS implementation of synchronous specifications
  - Covers causality and read/write communication
  - Compositionality, concurrency
- Future: Synthesis
  - Make synchronous automata weakly endo/isochronous. Optimality issues.
  - Heuristics for actual synchronous languages and specifications. Scaling issues (large specifications).
  - GALS circuits using asynchronous logic
  - Deal with mode changing latency
- What about timed models ?

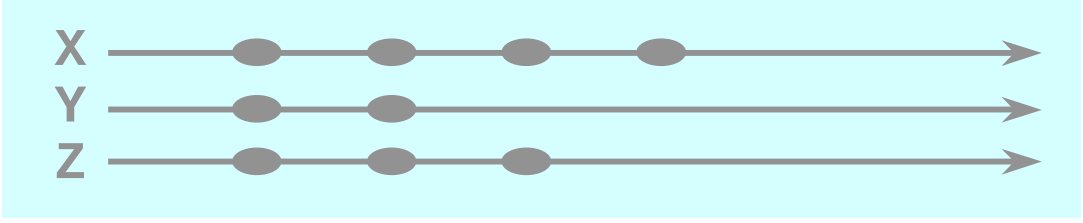
Outline

- **Tags, Tagged systems, and their ||, using drawings**
- **Some formalization**
- **Theorems and their use for LTTA**

Heterogeneous Systems as tagged systems with tag set as parameter [Emsoft'03]



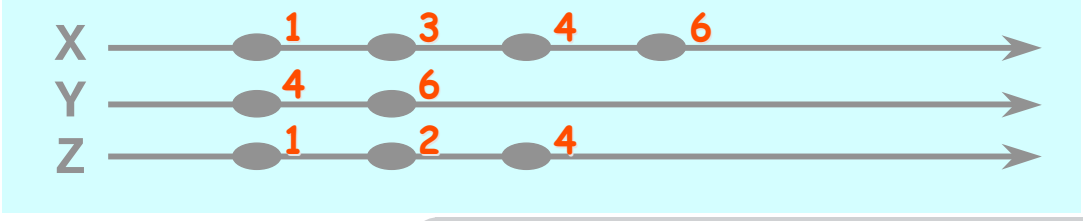
synch



=

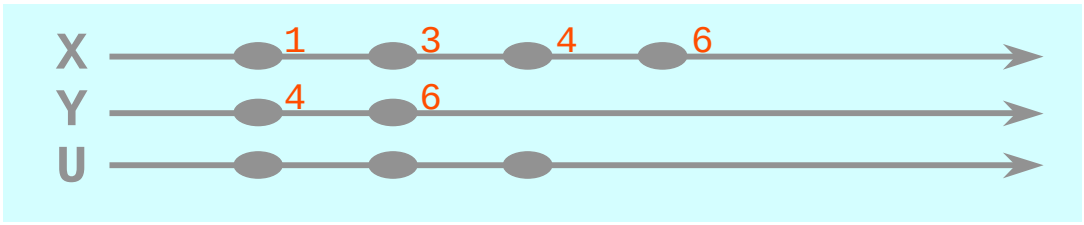
asynch

...

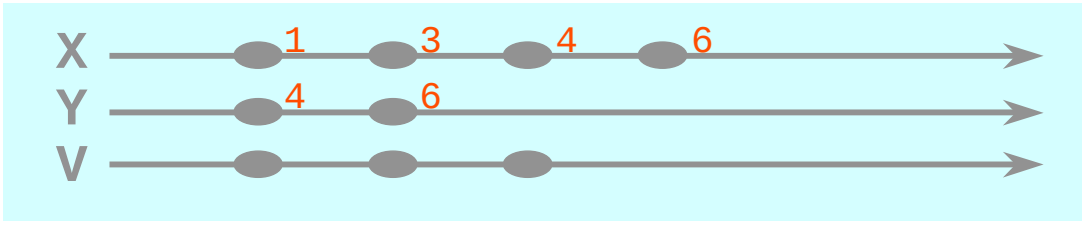


+ TAGS

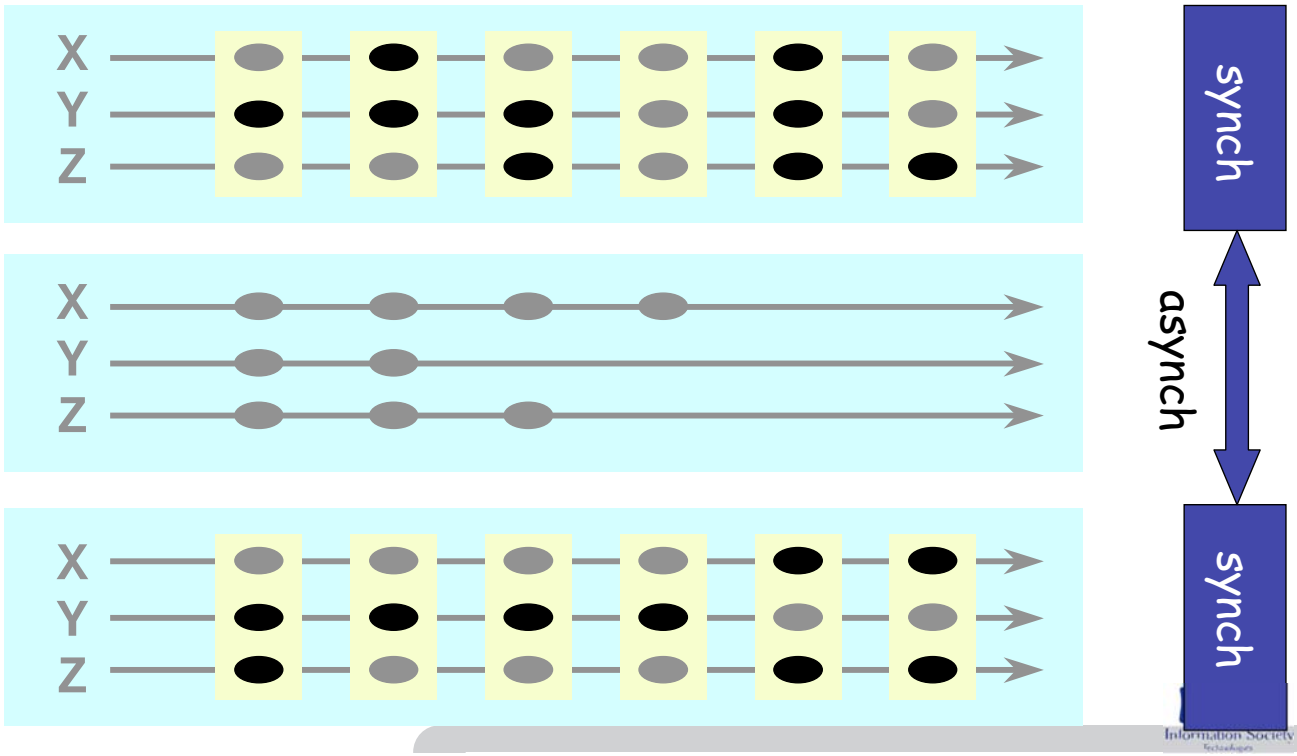
Homogeneous parallel composition: P || Q



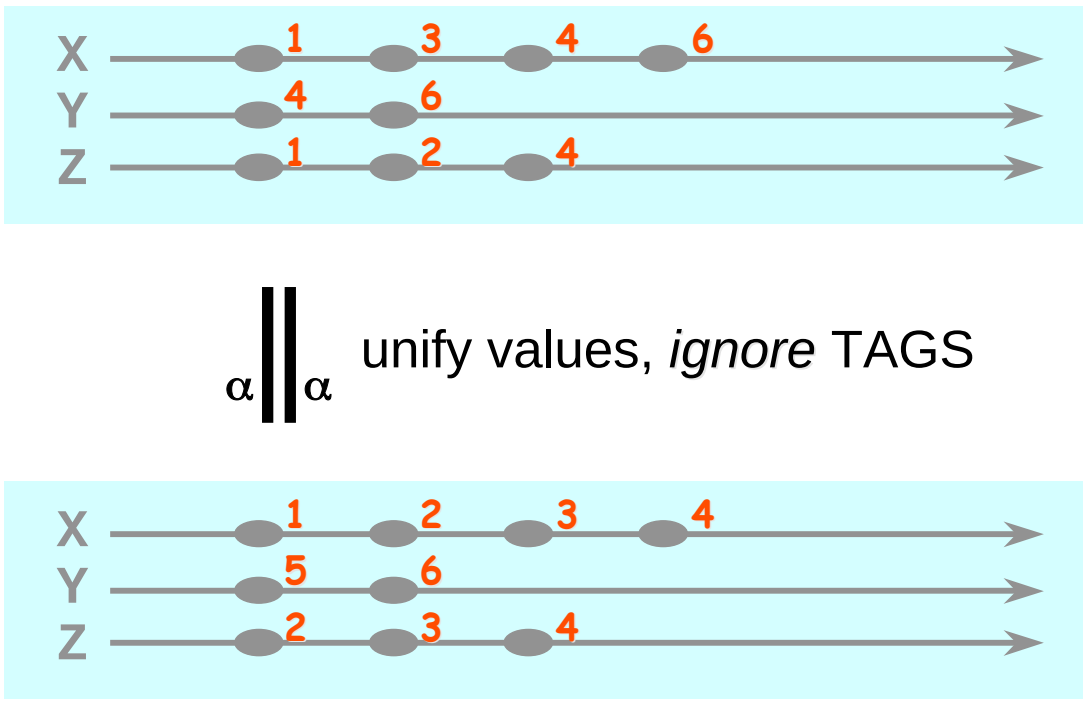
|| unify values and TAGS



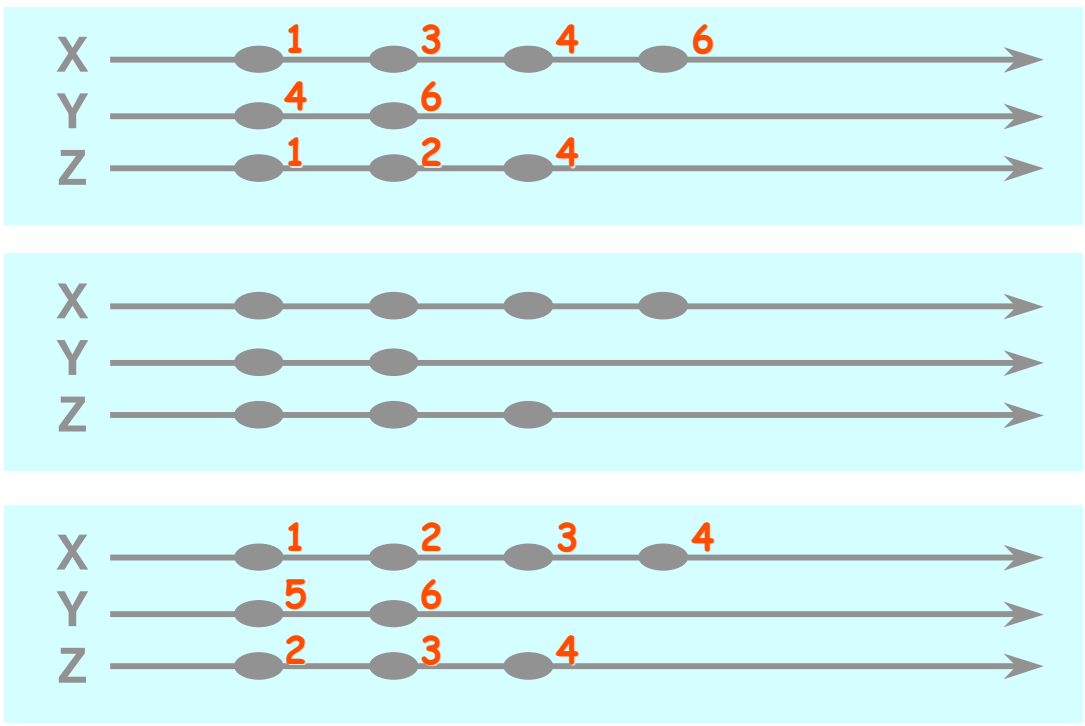
GALS as an heterogeneous system



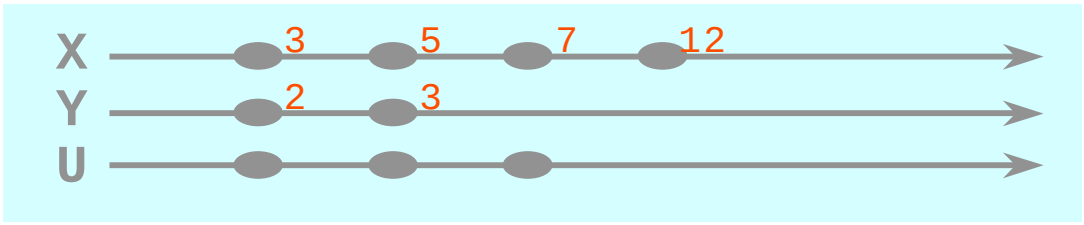
GALS as an heterogeneous system : P $\|_\alpha$ Q



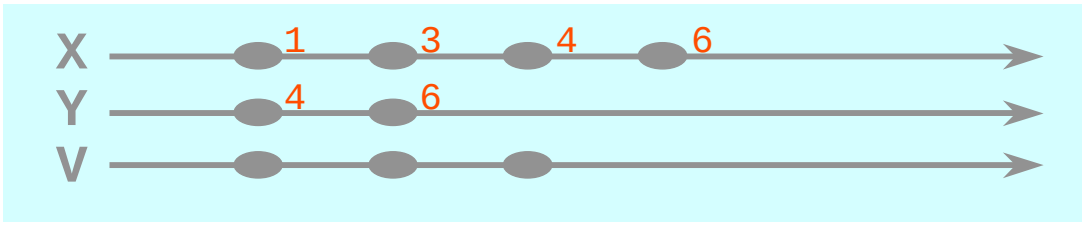
GALS as an heterogeneous system: P $\|_\alpha$ fifo $\|_\alpha$ Q



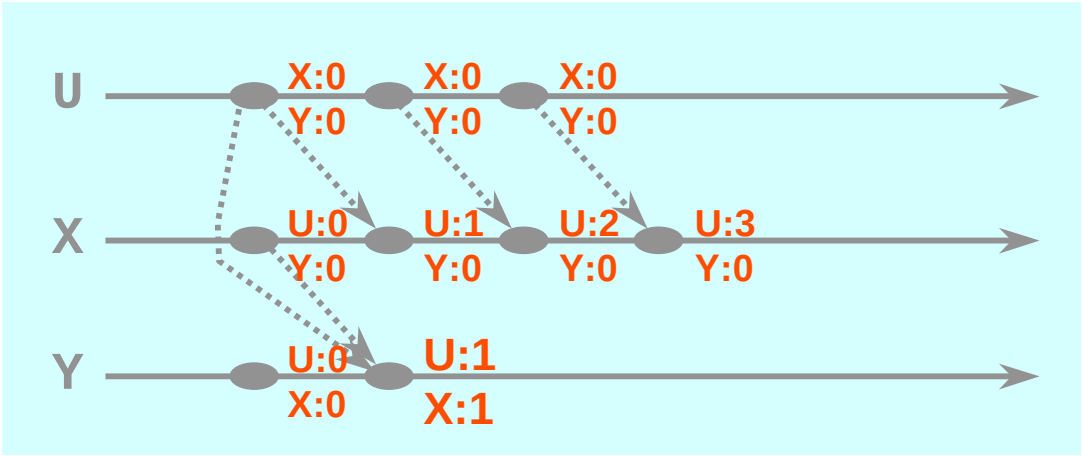
Heterogeneous parallel composition: Tags to model dates, latest wins



unify values, *combine* TAGS *with max* : (5,3) → 5

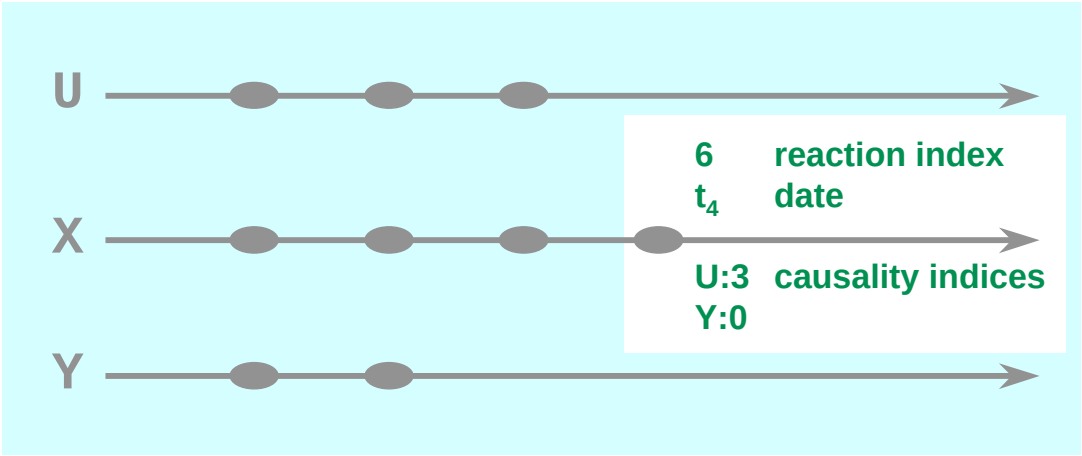


Heterogeneous parallel composition: tags to model causalities



unify values, combine TAGS componentwise with max : ((U:1),(U:3)) à (U:3)

Heterogeneous parallel composition: composite tags to combine features



- identify values, identify reaction indices, combines dates with max, combines causality indices with max componentwise

Heterogeneous parallel composition: what tags can capture

- **Synchrony, asynchrony**
- **Causalities**
- **Physical time – for performance evaluation and/or timed automata**
- **(More is conjectured: hybrid automata, probabilities)**
- **… and their combination**
- **Desynchronizing ⇔ erasing (part of) tags**

Outline

- **Motivation**
- **Tags, Tagged systems, and their ||, using drawings**
- **Some formalization**
- **Theorems and their use for LTTA**

Heterogeneous parallel composition: ingredients (1)

$$(\mathcal{T}, \leq, \sqcup)$$

(tag set, p.o., unification function)

$\sqcup$: partial function, consistent with $\leq$

Examples for $\sqcup$:

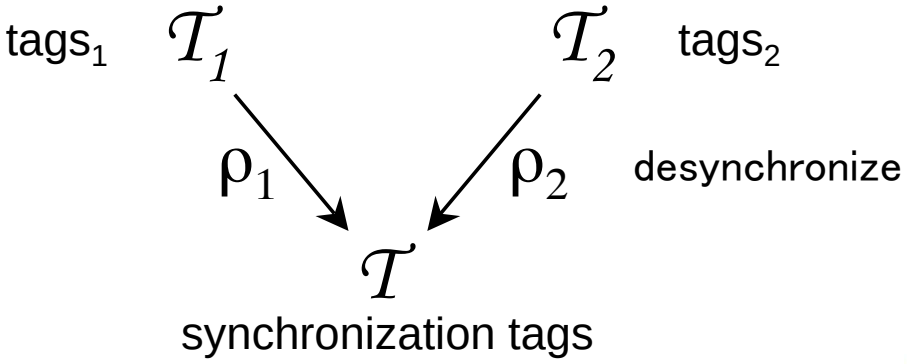
☞ unifiable iff equal: $(\tau,\tau) \rightarrow \tau$

☞ always unifiable: $(\tau,\sigma) \rightarrow \max(\tau,\sigma)$

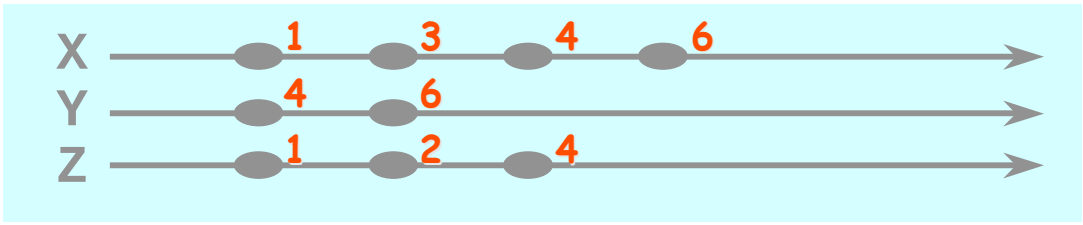
Heterogeneous parallel composition: ingredients (2)

$$P_1 \ {}_{\rho_1}||_{\rho_2} \ P_2$$

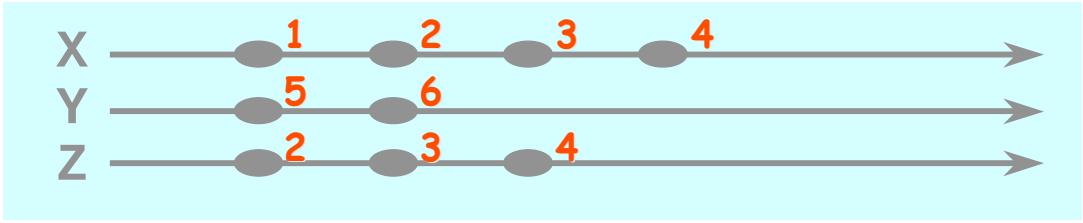
tag morphism $\rho: \mathcal{T} \longrightarrow \mathcal{T}'$ consistent with $\leq, \sqcup$

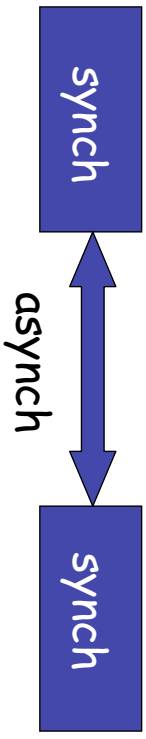


Heterogeneous parallel composition: GALS $P \ {}_{\alpha}||_{\alpha} \ Q$

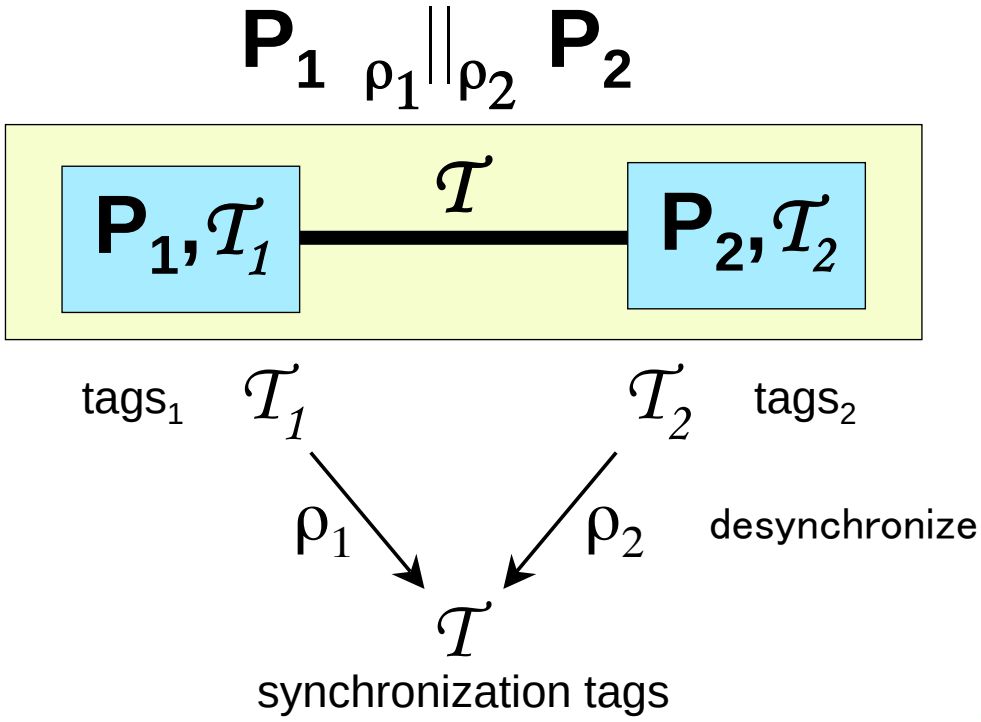


unify values, *ignore* TAGS $\alpha: \mathcal{T} \rightarrow \{.\}$

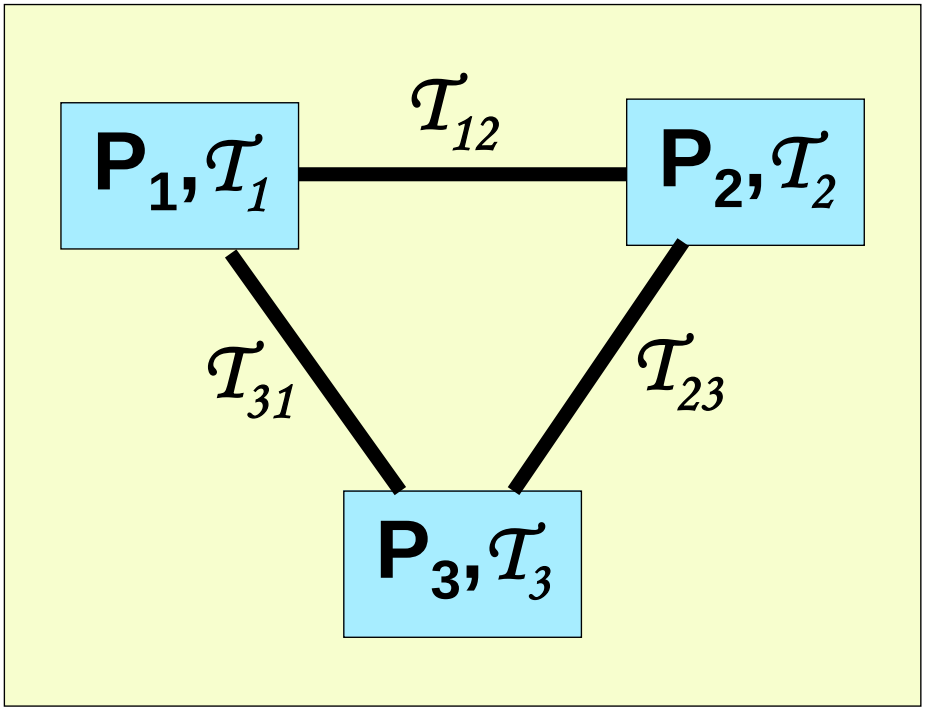




Heterogeneous parallel composition: graphical notation



Heterogeneous architectures: graphical notation

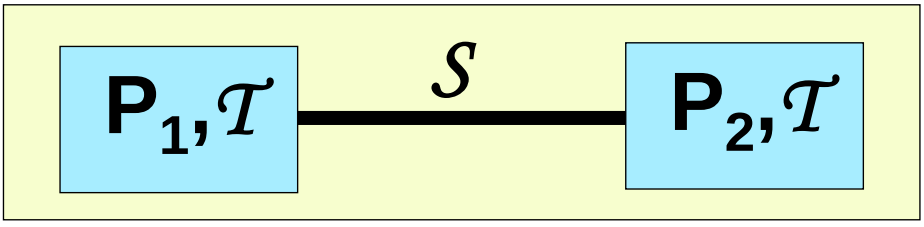


well defined!

Outline

- **Motivation**
- **Tags, Tagged systems, and their ||, using drawings**
- **Some formalization**
- **Theorems and their use for LTTA**

Heterogeneous parallel composition: theorem 1

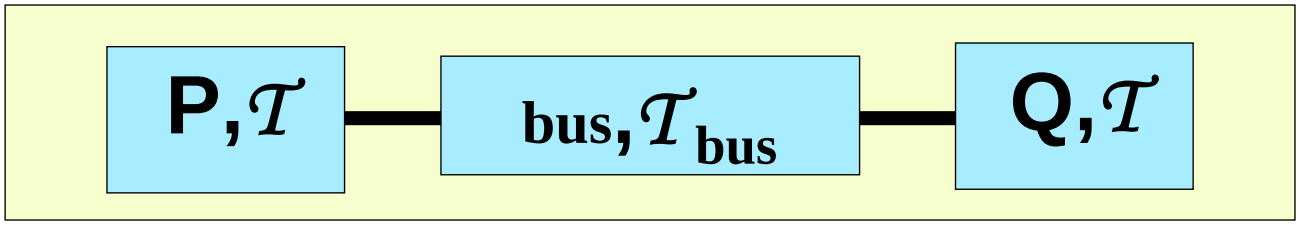


This architecture is semantics preserving if:

(i) $(P_i)_{\mathcal{S}}$ is in bijection with P_i

(ii) $(P_1 \,||\, P_2)_{\mathcal{S}} = (P_1)_{\mathcal{S}} \,||\, (P_2)_{\mathcal{S}}$

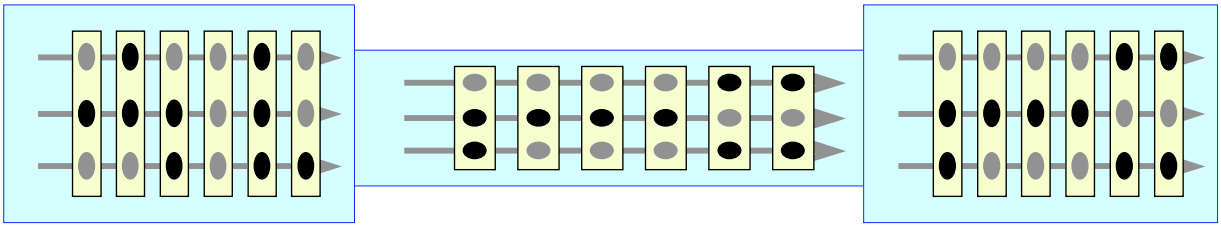
Heterogeneous parallel composition: “archi/appli separation” theorem 2

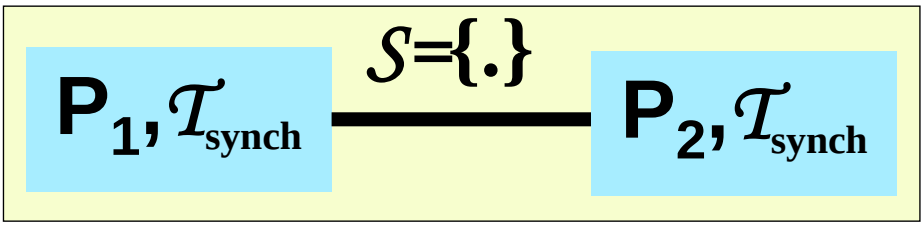


This architecture is semantics preserving if:

1. **bus** is in bijection with $(\mathbf{bus})_{\mathcal{S}}$, and
   $(\mathbf{bus})_{\mathcal{S}} = \mathbf{Id}$, where $\mathcal{T} \rightarrow \mathbf{S} \leftarrow \mathcal{T}_{\mathbf{bus}}$
2. $\mathbf{P} \,||_{\mathcal{S}}\, \mathbf{Q}$ is semantics preserving

Adaptors for GALS: apply Th. 1





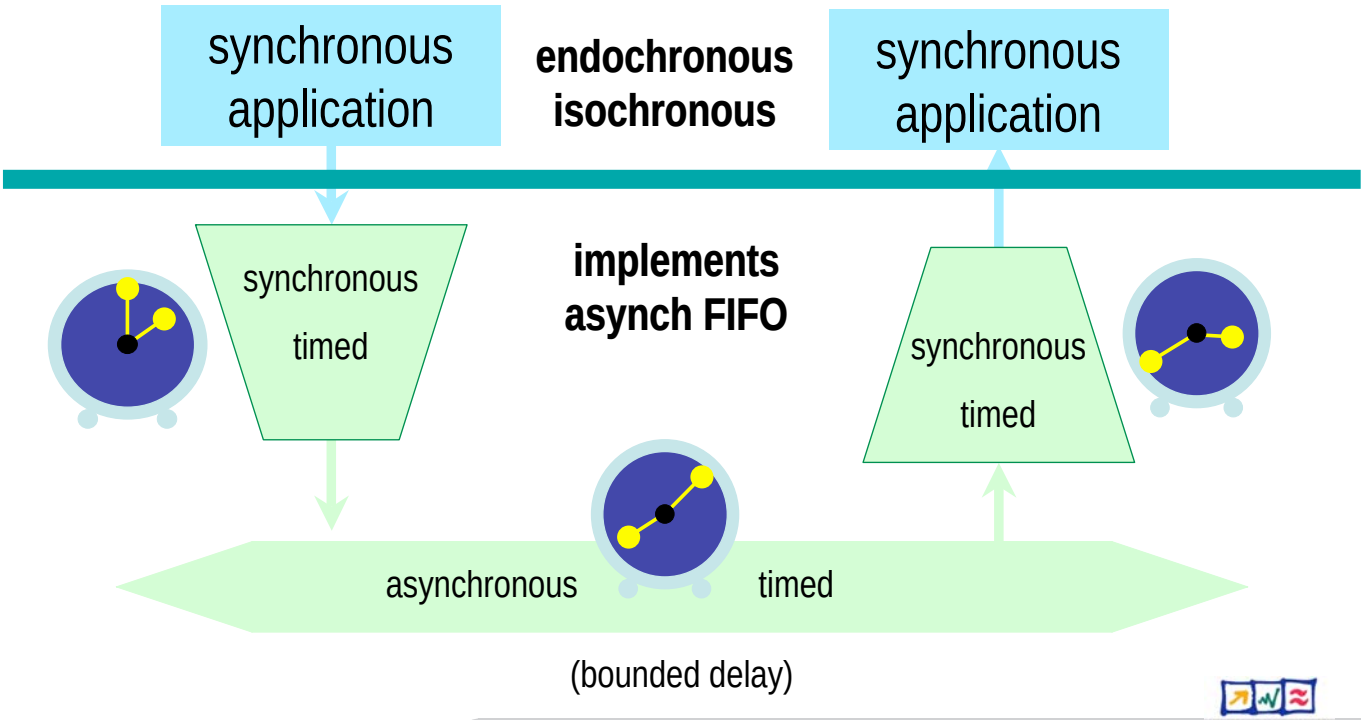
1. $(P_i)_{\mathcal{S}}$ is in bijection with P_i : endochronous
2. $(P_1 \,||\, P_2)_{\mathcal{S}} = (P_1)_{\mathcal{S}} \,||\, (P_2)_{\mathcal{S}}$: isochronous

Endochrony & Isochrony (GALS), effective properties

[BenvCaillaud2000, CaillaudPotop2004]

- A process P is endochronous when at each state the presence/absence of each variable can be inferred incrementally from the values carried by present input variables and state variables.
- Two processes P1,P2 are isochronous when at each state if each pair of shared variables that are present in both P1,P2 have the same value then all the shared variable are either present with the same value or absent
- Endochrony and isochrony are expressed in terms of transition-relations (not infinite behaviors)
  - They can be *model-checked*
  - They can be *synthesized*: for a given process P wrapper processes can be derived and composed with P to guarantee each property; wrappers provide “cheap additional signalling”

Applying theorems 2 and then 1 validates LTTA



Conclusion and perspectives

- Functional and non-functional aspects *jointly handled*, at both component- and system-level
- Design space involves both functions and execution infrastructure
- With *heterogeneous and flexible* Models of Computation and Communication (MoCC)
- **Tag systems provide the needed algebraic framework to develop ad-hoc mathematical framework for semantic based architecture analysis and synthesis**