Heterogeneous Reactive Models and Correct-by-Construction Deployment

Alberto L. Sangiovanni-Vincentelli

EECS Department
University of California at Berkeley

With A. Benveniste, L. Carloni and P. Caspi

Synchronous Model

$$P_i \equiv R_i^{\omega}$$

$$P_1 || P_2 \equiv (R_1 \wedge R_2)^{\omega}$$

- P_i: synchronous process
- R_i: set of all possible reactions of process P_i
- ω: indicates non-terminating reactions

- A synchronous process evolves according to an infinite sequence of successive reactions
- The parallel composition of two processes is the conjunction of their reactions
  - product of automata, FSM connection

Synchronous Model

- Foundation of Synchronous Languages
  - Esterel, Lustre, Signal
- Pervasive in Mathematics and Engineering
  - Discrete-Dynamic Control Systems
  - Digital Integrated Circuit Design
- When composition is possible, we can reason formally on the properties of the composite system based on the properties of its components
  - Notice: generally, functional systems are not closed under concurrent composition

Synchronous Assumption

- Communication Delay is negligible w.r.t. Computation Delay

- The system transitions between a reaction and the other instantaneously and the communication of the values from the outputs of a component to the inputs of another takes *zero time*

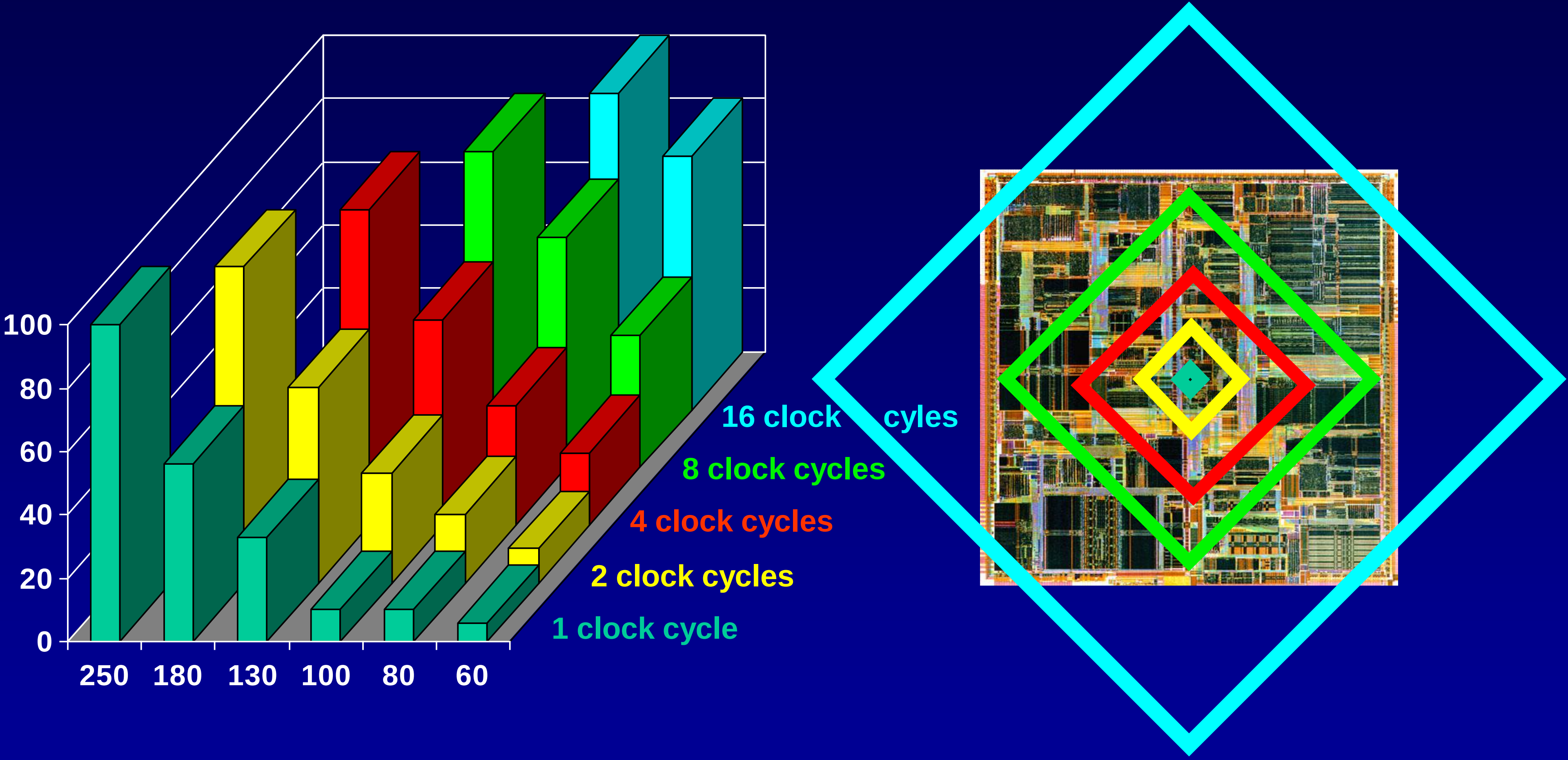
```
loop each tick
   read inputs
   compute next state
   write outputs
end loop
```

Distributed Nature of Implementations

- in **hardware**
  - with DSM technologies, the chip becomes a distributed system
  - wire delays not negligible w.r.t. transistor delays
  - on-chip communication latency is hard to estimate

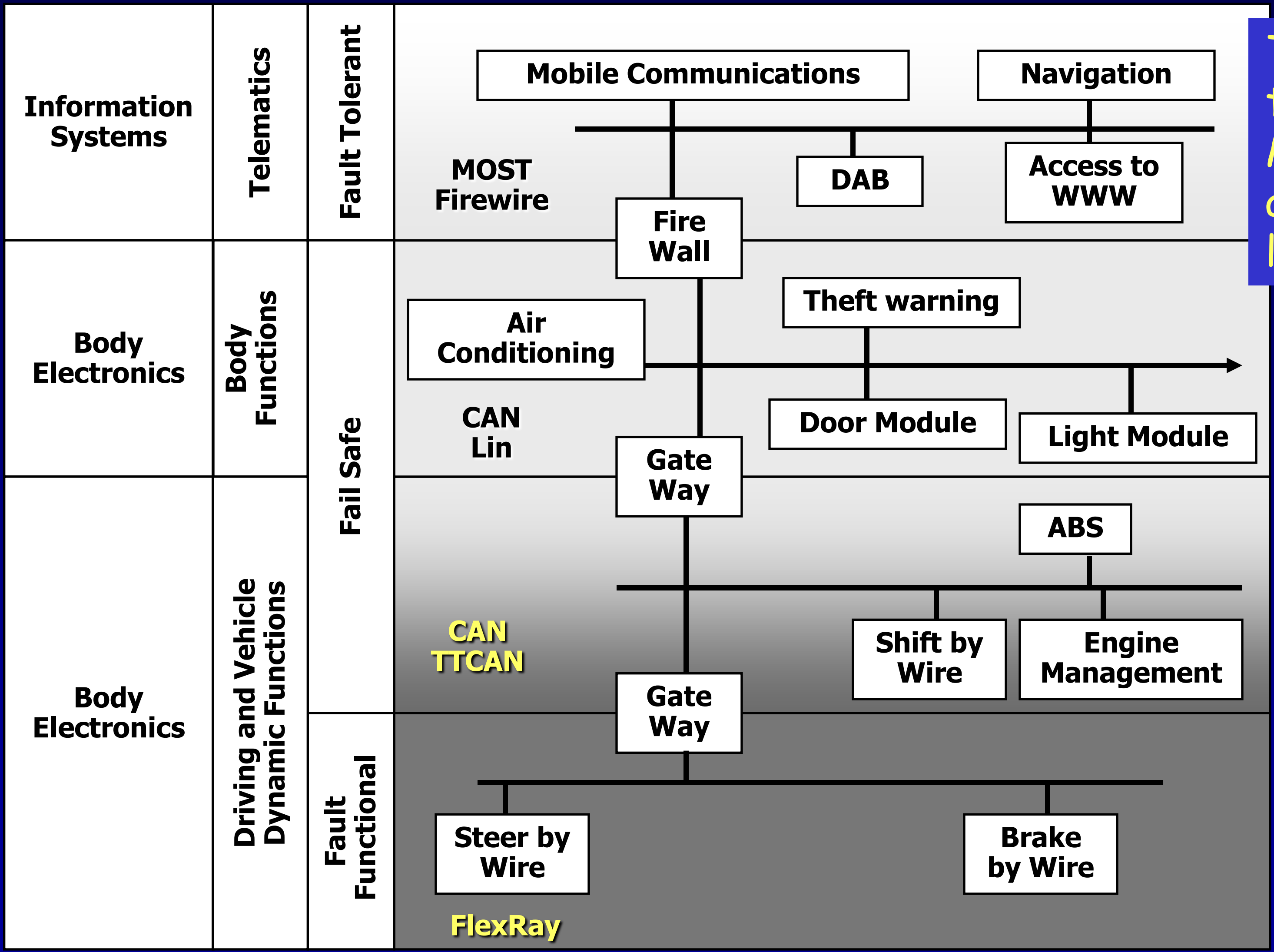
- in **(embedded) software**
  - applications with distributed nature badly matching the synchronous assumption
    - real-time safety critical embedded systems in avionics and automotive industries
    - industrial plants, transportation/power networks
  - large variations in computation/communication times
  - hard to maintain a global notion of time

DSM: Percentage of Reachable Die



- *"For a 60 nanometer process a signal can reach only 5% of the die's length in a clock cycle"* [D. Matzke,1997]
- **Cause:** Combination of high frequencies and slower wires

Electronics for the Car: A Distributed System



Information Systems

Telematics

Fault Tolerant

MOST
Firewire

Mobile Communications

Navigation

DAB

Access to WWW

Fire Wall

Body Electronics

Body Functions

Fail Safe

Air Conditioning

Theft warning

CAN
Lin

Door Module

Light Module

Gate Way

Body Electronics

Driving and Vehicle Dynamic Functions

ABS

CAN
TTCAN

Shift by Wire

Engine Management

Gate Way

Fault Functional

Steer by Wire

Brake by Wire

FlexRay

Today, more than 80 Microprocessors and millions of lines of code

Outline

- **A common formal framework for the study of**
  - Models of Computation (MOCs)
  - synchronous, asynchronous, GALS
  - event absence in modeling distributed systems
  - the de-synchronization problem

- **De-synchronization of embedded software programs**
  - properties of endochrony and isochrony

- **Concluding Remarks**

The Tagged-Signal Model [Lee & Sangiovanni '96]

- **Event : a member of $V \times T$,**
  - V : set of values, T : set of tags
- **Signal : a set of events**
  - s = { (v1, t1), (v2, t2), … , (vk, tk) }
- **Process : a subset P of the set of N-tuples of signals**
- **Behavior : a tuple of signals b = (s_1, s_2, …, s_N) which satisfies a process P**
- **System : a composition of processes P_1,… ,P_M**
  - (i.e. the intersection of their behaviors)

More on Tags

- **Tags can be a mechanism to express time**
  - across various levels of abstraction
    - Logical Time in initial specification
    - Physical (“Real”) Time in final implementation

- **But tags are essentially a tool to express constraints**
  - Coordination constraints
    - among events of the same signal
    - among signals of the same behaviors

The “Absent-Value” Event ($\perp$)

- a signal s is present at tag t when

$$\exists e=(t,d) \in s \mid d \neq \perp$$

- otherwise, s is absent at tag t

tag	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}	t_{12}	t_{13}	t_{14}
u	4	-2	5	$\perp$	-1	3	0	2	$\perp$	$\perp$	6	4	2	$\perp$
v	1	$\perp$	1	$\perp$	$\perp$	1	0	1	$\perp$	$\perp$	1	1	1	$\perp$

Assumption on the Tags of a Signal

- For any tag $t \in T$, each signal s in the system has at most one event, i.e.

$$\forall b \in B,\ \forall s \in b,$$
$$\neg[\ \exists e_1 \in s\ \exists e_2 \in s \mid tag(e_1) = tag(e_2)\ ]$$

- This implies
  - a total order < among the tags of a signal
  - a total order < among the events of a signal

Ordering Tags in a Process

- Generally, process tags are not ordered
- When used to express causality relations among signals, it is common to assume a partial order $\leq$

$$t < t' \text{ when } t \leq t' \text{ and } t \neq t'$$

- Timed System: the set T of tags (timestamps) is a totally ordered set.

$$\forall\, t,t' \; (\, t \neq t' \Rightarrow (t < t' \text{ or } t > t')\,)$$

  - the ordering among the timestamps of a signal s induces a natural order on the set of events of s

Tags

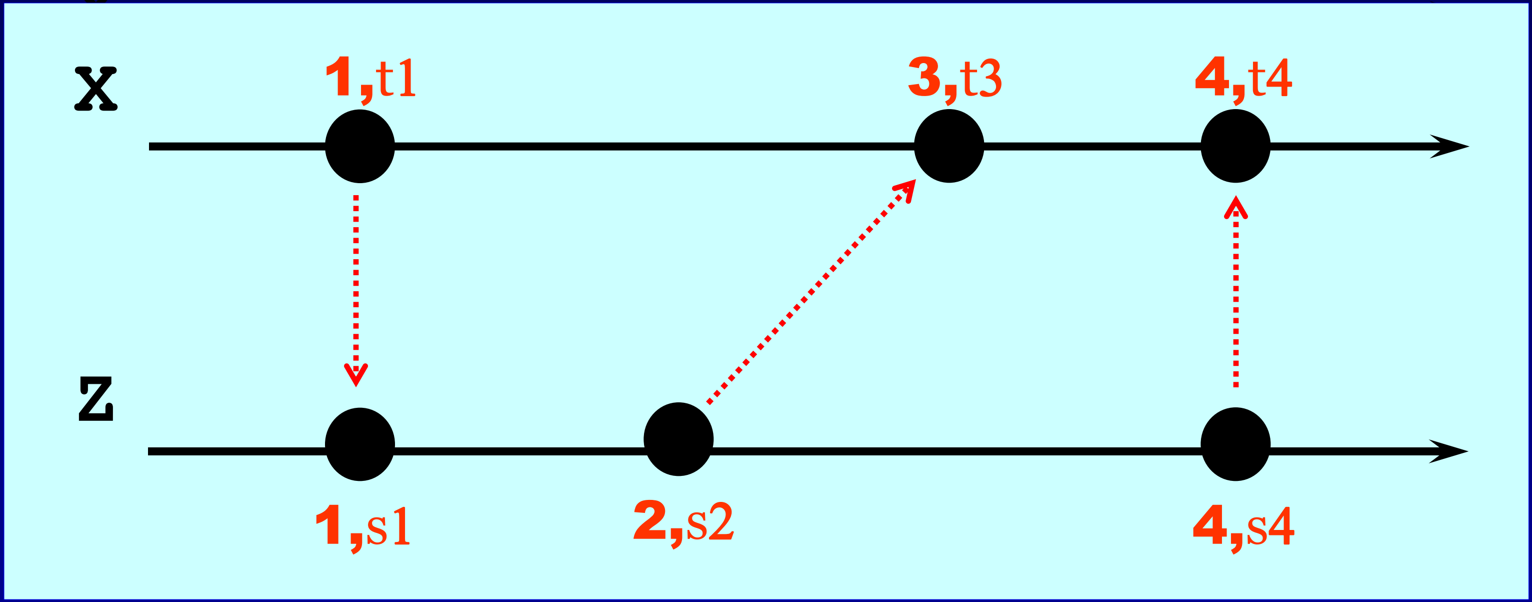
- Assumption: the tag set T is partially ordered

$$t < t' \Leftrightarrow (t \leq t' \wedge t \neq t')$$

- A clock h is a non-decreasing map N → T

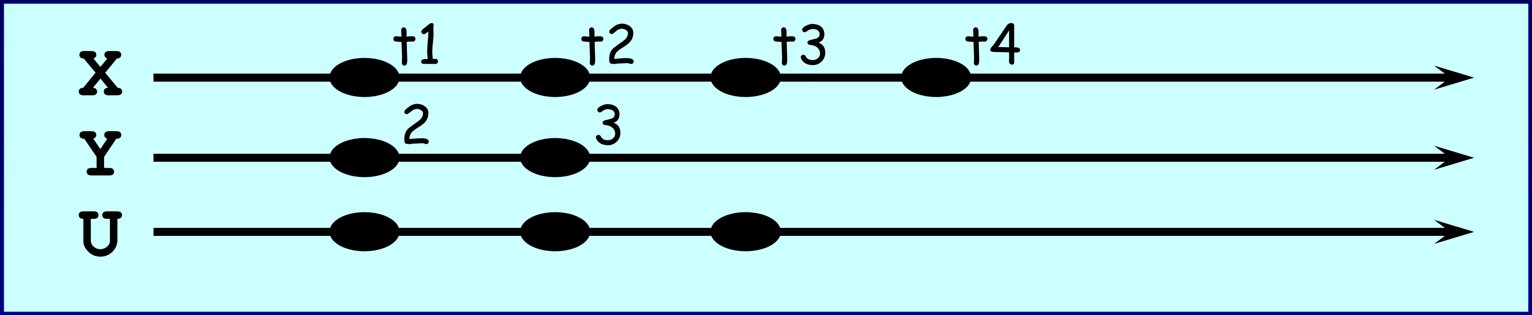
- Modeling Heterogeneity: the set T of tags can be adjusted to account for different class of systems (synchronous, asynchronous, timed,…)

TAGS generalize and heterogeneize



a TAG consisting of the triple (reaction, phys.time, causality)

TAGS generalize and heterogeneize



- TAGS can belong to any partially ordered set – tags can index reactions, can be real time R, can encode causality, can do both, etc.
- TAGS can be tuples – desynchronization can be generalized to erasing some components of the tag; yields morphisms of tag sets, we denote them by **r**, **a**
- different TAG sets can be used for different systems – we can mix synchronous systems (with tag set N) and asynchronous ones (with trivial tag set), and more

Synchronous Systems

- Time change ρ any bijective and strictly-increasing function $\rho: T \rightarrow T$
- Rt : set of all time changes over T
- P is stuttering-invariant *iff* for every behavior $b \in P$ and every time change $\rho \in Rt$

$$b\rho \in P \Leftrightarrow \{ (t,d) \in b \Leftrightarrow (\rho(t),d) \in b\rho \}$$

Stuttering-invariance → invariance under time change

Synchronous Events, Behaviors, Processes

- Synchronous Events have the same tag

$$s_1 \approx s_2 \text{ when } tag(s_1) = tag(s_2)$$

- Synchronous Signals $s_1 \approx s_2$ when

$$\forall e_i \in s_1,\ \exists e_j \in s_2\ (e_i \approx e_j) \text{ and}$$
$$\forall e_k \in s_2,\ \exists e_l \in s_1\ (e_k \approx e_l)$$

- Synchronous Behaviors

$$b_1 \approx b_2 \text{ when } \forall s_i \in b_1,\ \forall s_j \in b_2,\ (s_i \approx s_j)$$

- Synchronous Processes

$$P_1 \approx P_2 \text{ when } \forall b_i \in B(P_1),\ \forall b_j \in B(P_2),\ (b_i \approx b_j)$$

Synchronous Systems

- Stand-alone synchronous behavior b when $b \approx b$
- Synchronous process (system) P when $P \approx P$

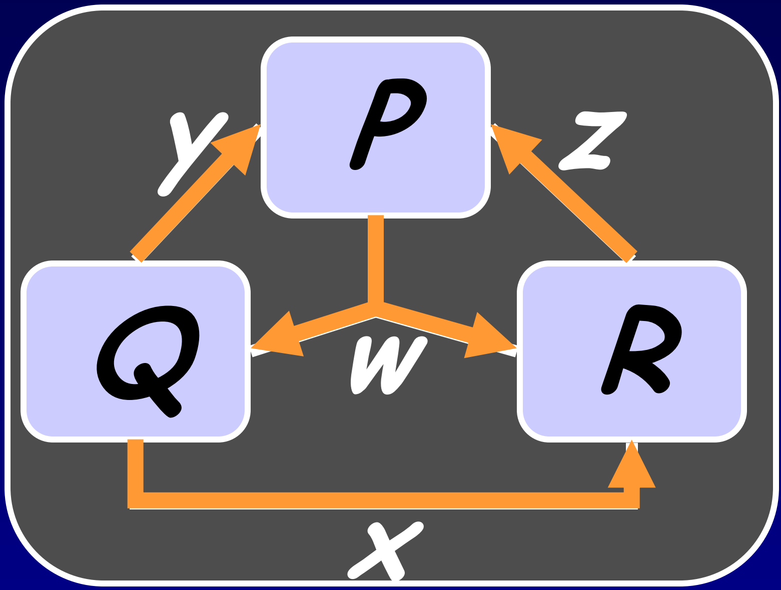
- In a synchronous system P, every signal is synchronous with every other signal

- Equivalently, for each tag t a signal has exactly one corresponding event

$$\forall b \in B(P), \forall s \in b \ \forall t \in T(P), \ (\exists! e \in s \mid tag(e)=t)$$

Synchronous Systems - Example

tag	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8
w	1	0	1	0	1	0	1	0
x	1	3	5	7	9	11	13	15
y	0	2	2	6	6	10	10	14
z	0	0	4	0	8	4	12	8



- 3 processes, 4 signals

Synchronous Languages (*Signal*)

- **Simplicity of synchronous assumption plus the power of concurrency in system specification**
- **Notion of clock of a variable**
  - a Boolean meta-variable tracking the absence/presence of a value for the corresponding variable
    - clocks: equivalence classes of simultaneously-present variables

- **The *Signal* compiler uses *clock calculus* to**

1. statically analyze every program statement
2. identify each variable's clock
3. schedule the overall computation

Asynchronous Events, Behaviors, Processes

- Asynchronous Events have different tags

  $$s_1 \cong s_2 \text{ when } tag(s_1) \neq tag(s_2)$$

- Asynchronous Signals

  $$s_1 \cong s_2 \text{ when } \forall e_i \in s_1, \forall e_j \in s_2 \ (e_i \cong e_j)$$

- Asynchronous Behaviors

  $$b_1 \cong b_2 \text{ when } \forall s_i \in b_1, \forall s_j \in b_2, \ (s_i \cong s_j)$$

- Asynchronous Processes

  $$P_1 \cong P_2 \text{ when } \forall b_i \in B(P_1), \forall b_j \in B(P_2), \ (b_i \cong b_j)$$

Asynchronous Systems

- Stand-alone asynchronous behavior b when $b \cong b$
- Asynchronous process (system) P when $P \cong P$

- In an asynchronous system P, every signal is asynchronous with every other signal
  - signals have disjoint tag sets

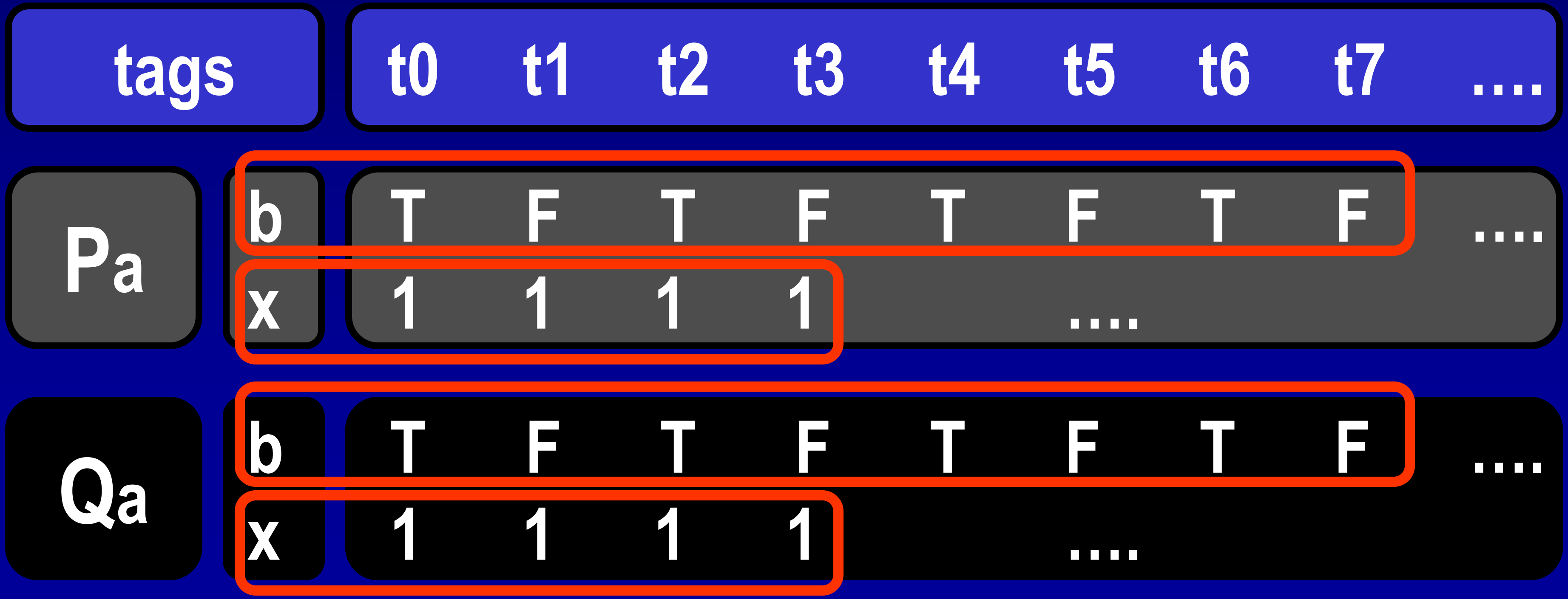
- Equivalently, for each tag t there is exactly one event across all signals

$$\forall b=(s_1,\ldots,s_M)\in B(P), \forall t\in T(P),$$

$$(\exists! e \in \cup_i s_i \mid tag(e)=t)$$

Asynchronous Systems

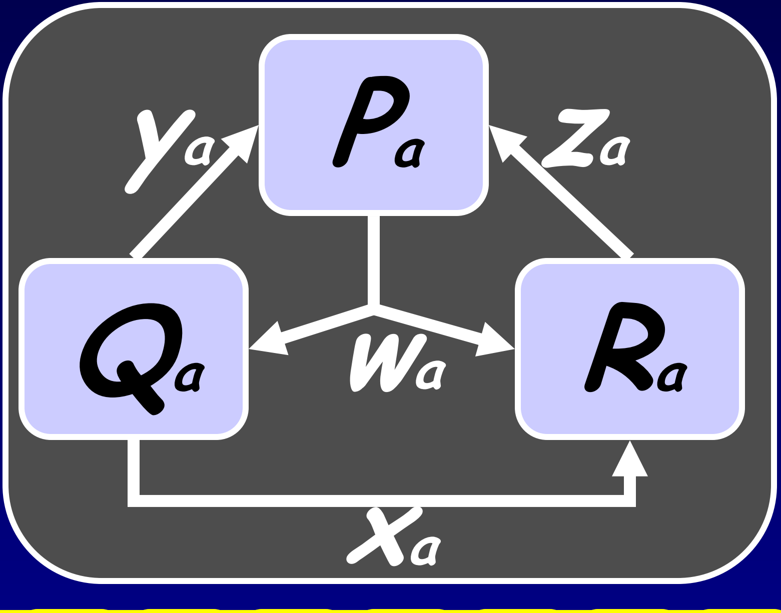
- T={.}, singleton (trivial set)
  - No global coordination information is available
  - No information on absolute/relative ordering of events
  - Absence cannot be sensed/used to exercise control
  - Composition ≡ separate unification of each common variable flow (models unbounded-FIFO communication, Kahn PNs, Rendezvous)



tags		t0	t1	t2	t3	t4	t5	t6	t7	....
Pa	b	T	F	T	F	T	F	T	F	....
	x	1	1	1	1		....			
Qa	b	T	F	T	F	T	F	T	F	....
	x	1	1	1	1		....			

Asynchronous Systems - Example

- The 3 asynchronous processes communicate by sharing signals (as in the synchronous case) but signals don't share tags



tag	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}	t_{12}	t_{13}	t_{14}	t_{15}
w_a	1				0				1				0		
x_a		1				3				5				7	
y_a			0				2				2				6
z_a				0				0				4			

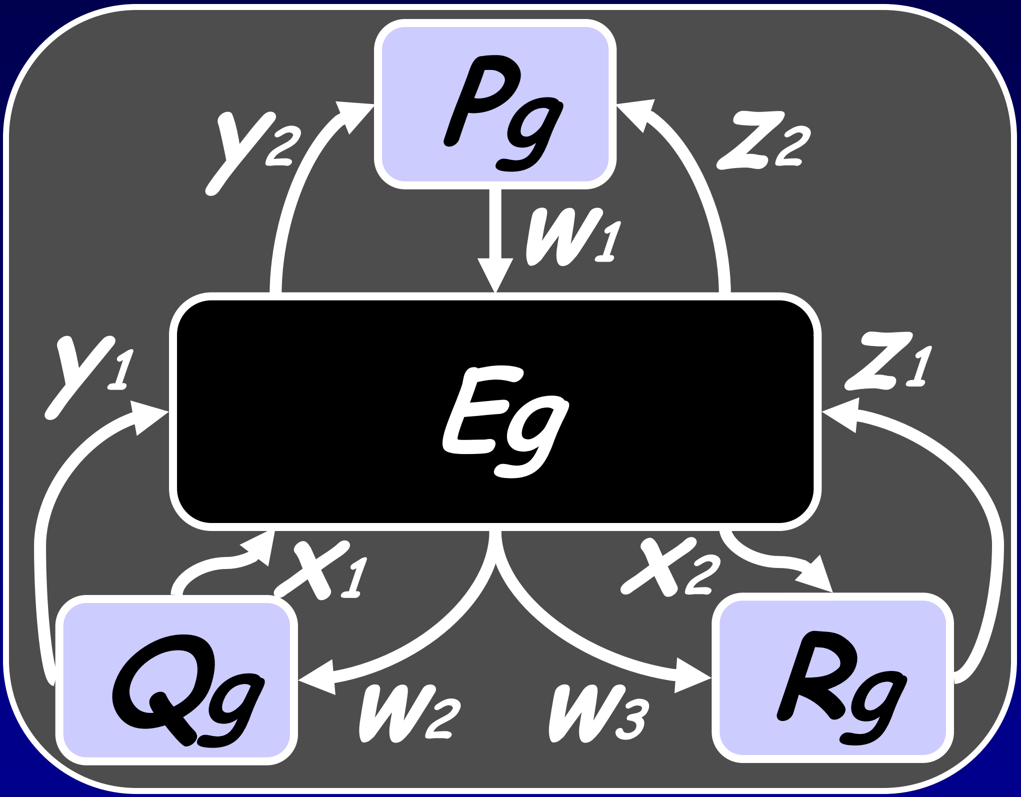
“In-Between” Systems

- Formally, the set of asynchronous systems is *not* the complement of the set of synchronous systems
  - there is an “In-Between System” set

- An element of the “In-Between Set” is
  - a process with a behavior that has both at least a pair of synchronous events (hence, it is not asynchronous) and at least a tag for which a signal does not present a corresponding event while another does (hence, it is not synchronous)

“In-Between” Systems: GALS Systems

- Computation occurs in synchronous clusters exchanging data asynchronously via a set of communication media
- Set $\mathcal{P}$ of computation processes Pg, Qg, Rg
- Media process Eg



$$\forall Pi, Pj \in \mathcal{P},\ (i=j \Rightarrow Pi \approx Pj \text{ and } i \neq j \Rightarrow Pi \cong Pj)$$

$$\forall Pi \in P, \forall b \in B(Eg),\ (\ b|V(Pi) \in B(Pi)\ )$$

GALS Systems - Example

	tag	t0	t1	t2	t3	t4	t5	t6	t7	t8	t9	t10	t11	t12	t13	t14
Pg	w1	1		⊥		⊥		0		⊥		1		⊥		⊥
	y2	⊥		0		⊥		2		⊥		⊥		⊥		2
	z2	⊥		⊥		0		⊥		4		⊥		0		⊥
Qg	w2		1				⊥				0				1	
	x1		1				3				⊥				5	
	y1		0				2				⊥				2	
Rg	w3				1				0				1			
	x2				1				3				⊥			
	z1				0				0				4			
Eg	w1	1		⊥		⊥		0		⊥		1		⊥		⊥
	w2		1				⊥				0				1	
	w3				1				0				1			
	x1		1				3				⊥				5	
	x2				1				3				⊥			
	y1		0				2				⊥				2	
	y2	⊥		0		⊥		2		⊥		⊥		⊥		2
	z1				0				0				4			
	z2	⊥		⊥		0		⊥		4		⊥		0		⊥

Modeling Communication Media

- Signal Decoupling
  - model a communication thread between two processes
- From a global viewpoint, a GALS system is a system with multiple tag sets (a multi-clock system)
  - each tag set represents dimension a that is familiar to a synchronous process and extraneous to all other synchronous processes
- Differently from synchronous systems, "unexpected" absent value events may arrive
  - computation may be badly affected if the process cannot recognize this issue

The Role of Absence

- GALS Systems
  - $t \notin T(s) \Rightarrow$ event absence
  - $t \in T(s) \wedge \exists e=(t,d) \in s \mid d=\perp \quad \Rightarrow$ value absence
  - $t \in T(s) \wedge \exists e=(t,d) \in s \mid d\neq\perp \quad \Rightarrow$ presence
- Synchronous Systems
  - event absence does not occur (value absence does)
    - signal decoupling is not necessary
    - "instantaneous" communication
    - computation and communication never overlap
- Asynchronous Systems
  - events are systematically absent
    - a present event in a signal ⇒ absences in all remaining signals!!
    - no common reference across processes
    - processes rely on *robust* handshaking protocols

Automatic Deploying of Synchronous Designs on Distributed Architectures

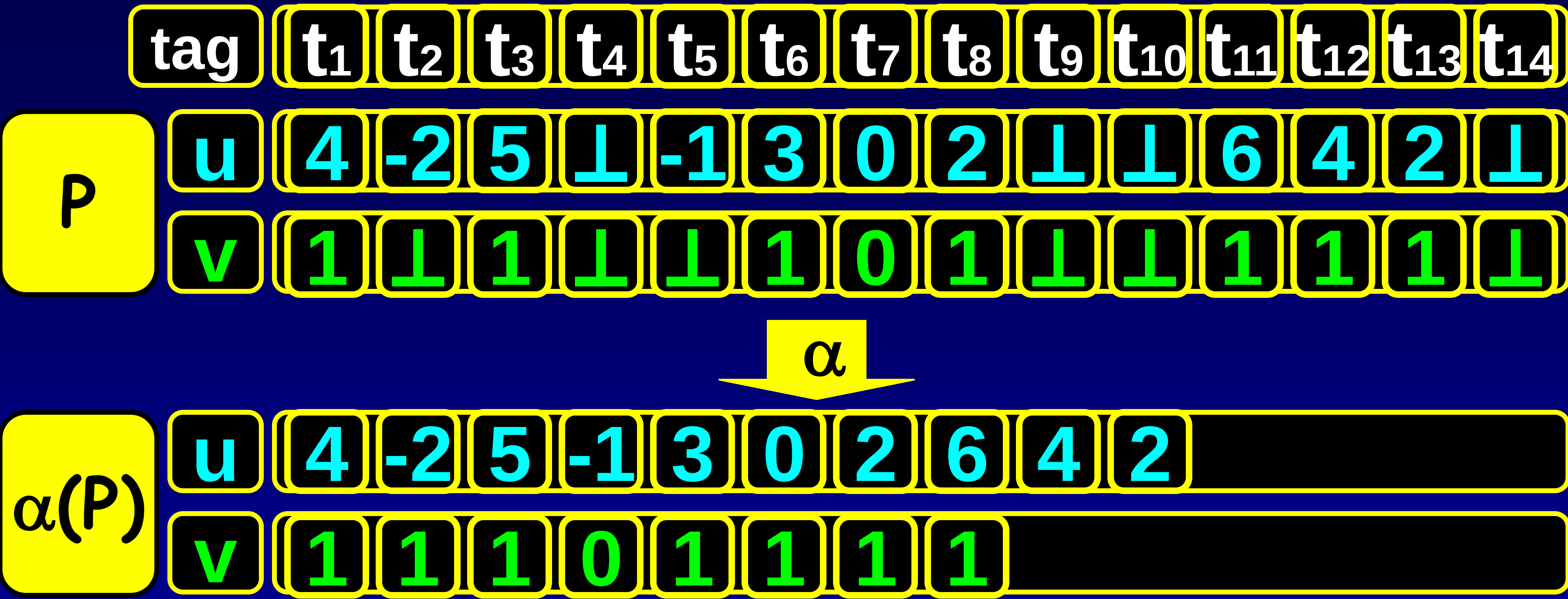
- **Need of techniques to make each process of a GALS system robust with respect to absence**
- **Under which conditions we can guarantee that**
  - sensing an absent value event when a different value was expected does not produce incorrect behavior
  - not sensing an absent value event when one is expected does not change the behavior of the system
- **Two approaches**
  - Latency-Insensitive Design
  - De-synchronization of Synchronous Programs

Semantic Equivalence

tag	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}	t_{12}	t_{13}	t_{14}
u	4	-2	5	-1	3	4	2	6	⊥	⊥	6	⊥	⊥	⊥
v	4	-2	5	-1	3	4	2	6	⊥	⊥	4	2	6	⊥

- Same sequence of values after discarding the absent events
- Semantic equivalence doesn't say anything about tags
  - two processes may be semantic equivalent even with disjoint tag sets
- The systems of the previous examples (synch., asynch., GALS) are semantic equivalent

De-synchronization of Synchronous Programs



	tag	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}	t_{12}	t_{13}	t_{14}
P	u	4	-2	5	⊥	-1	3	0	2	⊥	⊥	6	4	2	⊥
	v	1	⊥	1	⊥	⊥	1	0	1	⊥	⊥	1	1	1	⊥

α(P)	u	4	-2	5	-1	3	0	2	6	4	2
	v	1	1	1	0	1	1	1	1		

- **mapping a sequence of tuples of values in domains extended with the absent value ⊥ into a tuple of sequences of present values, one sequence per each variable**
  1. remove synchronization barriers among reactions
  2. individually compress the sequence of values for each variable

Semantic-Preserving Time Changes

- Given $P_1=(V_1,T_1)$, $P_2=(V_2,T_2)$ with time change mappings $\rho:T_1 \rightarrow T$ and $\rho:T_2 \rightarrow T$

  let $T_1=T_2$ and consider two semantics:

  -the Strong Semantics $P_1 || P_2$

  -the Weak Semantics $P_1 (\rho || \rho) P_2$

- **ρ is semantics-preserving *when* two behaviors, which compose according to the strong semantics, compose also according to the weak one, i.e.**

$$P_1 || P_2 \quad \equiv \quad P_1 (\rho||\rho) P_2$$

Theorem

- Given $P_1=(V_1,T_1)$, $P_2=(V_2,T_2)$ with $T_1=T_2$:

$$P_1 \,||\, P_2 \quad \equiv \quad P_1 \ (\rho||\rho)\ P_2$$

$$\Updownarrow$$

$\forall i \in \{1,2\}$: $(P_i)_\rho$ is in bijection with P_i

and

$$(\, P_1 \ || \ P_2 \,)_\rho \ = \ (P_1)_\rho \,||\, (P_2)_\rho$$

Endochrony & Isochrony

- A process P is endochronous when
  - for each tag t of its behaviors the presence/absence of events on all its signals can be inferred incrementally from the values of a subset of them that are guaranteed to be present at t
- Two processes P_1,P_2 are isochronous when
  - for each tag t, if there is a pair of shared signals that are present and agree on the event value, then, for each other pair of shared signals, either they are present and agree on the same value or they are 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

Conclusion

- Heterogeneous reactive systems modeled as tagged systems
- Tag sets to capture: reaction indices, physical time, causalities… and their combination
- Desynchronizing ⇔ erasing (part of) tags
- Theorems to cast semantics preserving as specific algebraic properties of tuples of systems
- To get effective algorithms for correct-by-construction deployment