# Loosely Time-Triggered Architectures (LTTA)

Albert Benveniste – INRIA

with B. Caillaud, P. Caspi, A. Sangiovanni-Vincentelli

S. Tripakis, M. di Natale, C. Pinello

Paul Caspi day, 28 september 2007

# Some historical remarks

Everybody lessoning to Paul Caspi must have noticed how quietly and slowly he speaks. Maybe his low entropic nature is the very reason for him to stick to simplest solutions, all over his career.

This lead to Lustre (« *There is little to say about Lustre, it's so simple* »)

# Some historical remarks

Everybody lessoning to Paul Caspi must have noticed how quietly and slowly he speaks. Maybe his low entropic nature is the very reason for him to stick to simplest solutions, all over his career.

This lead to Lustre (« *There is little to say about Lustre, it's so simple* »)

Around 1998, Paul warned us that ***distributed real-time control systems are not simple*** (At that time Hermann Kopetz already had proposed TTA, which, together with synchronous programming, was a simple solution.)

# Some historical remarks

Everybody lessoning to Paul Caspi must have noticed how quietly and slowly he speaks. Maybe his low entropic nature is the very reason for him to stick to simplest solutions, all over his career.

This lead to Lustre (« *There is little to say about Lustre, it's so simple* »)

Around 1998, Paul warned us that ***distributed real-time control systems are not simple*** (At that time Hermann Kopetz already had proposed TTA, which, together with synchronous programming, was a simple solution.)

Paul said:

*The engineering practice is often not TTA; still, time is the vehicle for controlling distributed real-time systems.*

*In fact, engineers seek for robustness in designs, they do not always want strictly synchronous platforms.*

*In maths, robustness refers to the ability to tolerate «small deviations»*

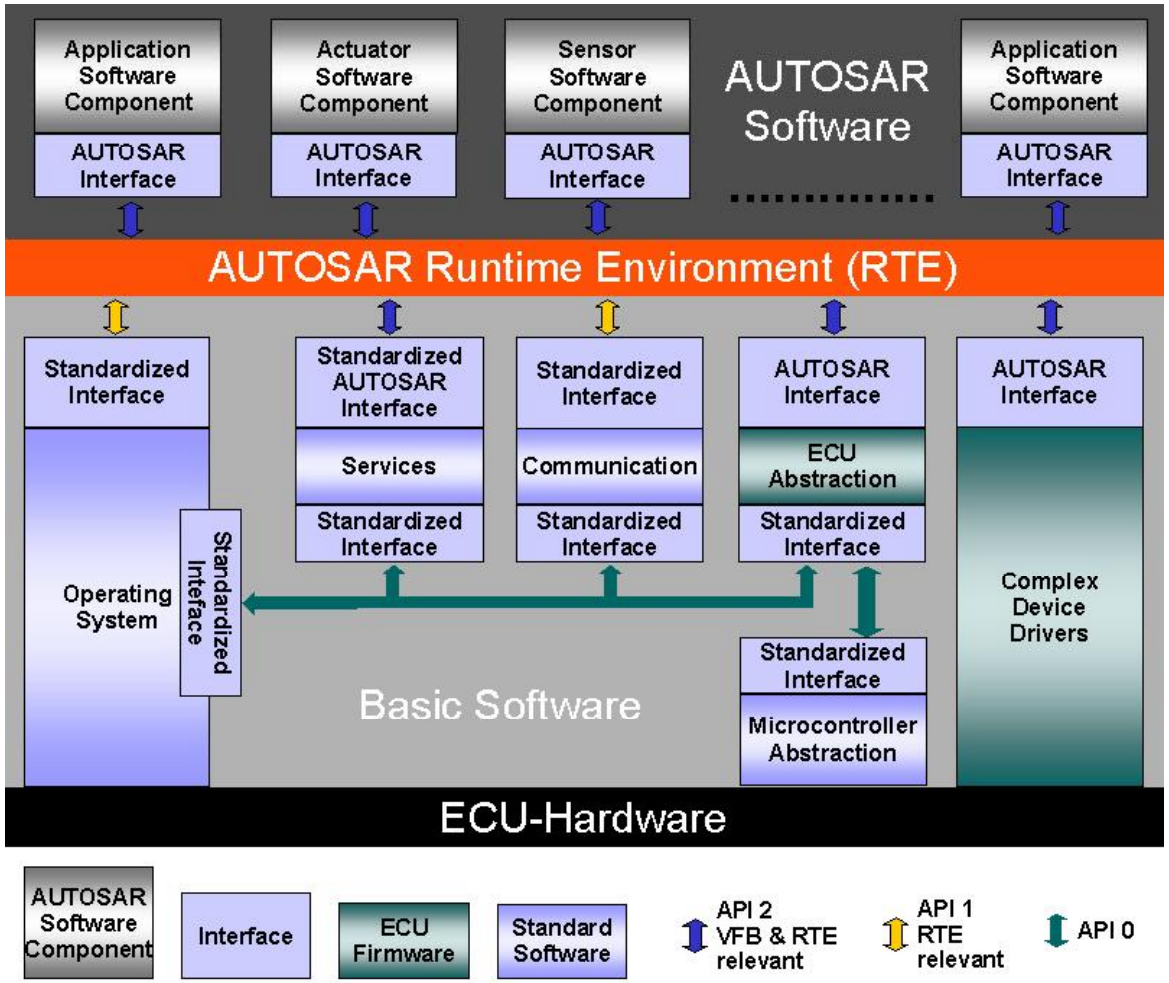
*Unfortunately, for mixed continuous/discrete systems, we do not know what « small » means.*

# What is the problem with software development for distributed embedded systems???

Here follow some slides borrowed from industrials on AUTOSAR and IMA

# Standardization

## AUTOSAR – ECU Software Architecture




**Automotive Open System Architecture (AUTOSAR):**

- Standardized, openly disclosed interfaces
- HW independent SW layer
- Transferability of functions
- Redundancy activation

**AUTOSAR RTE:**
by specifying interfaces and their communication mechanisms, the applications are decoupled from the underlying HW and Basic SW, enabling the realization of Standard Library Functions.

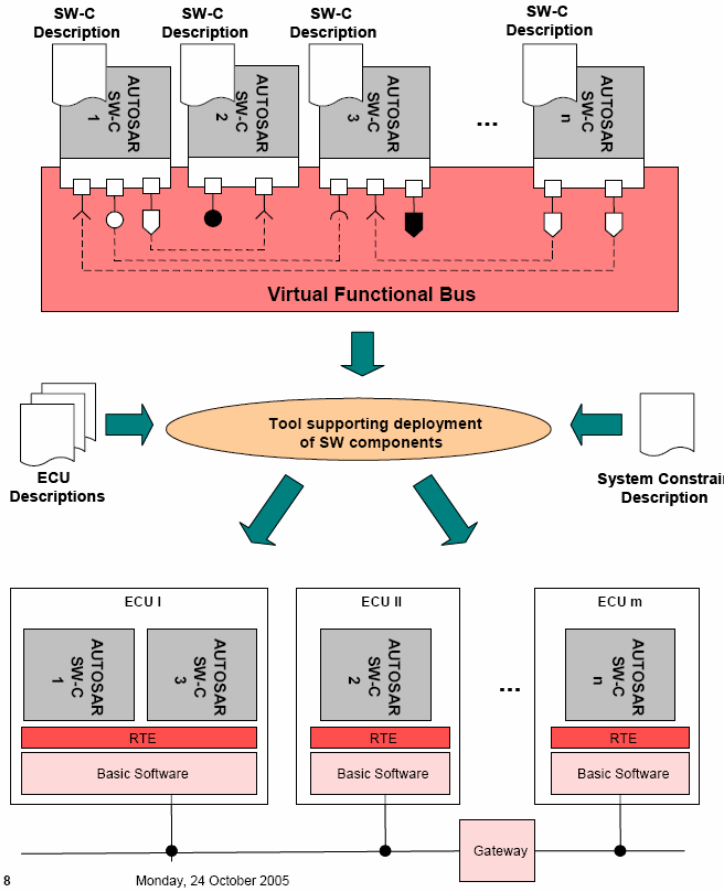




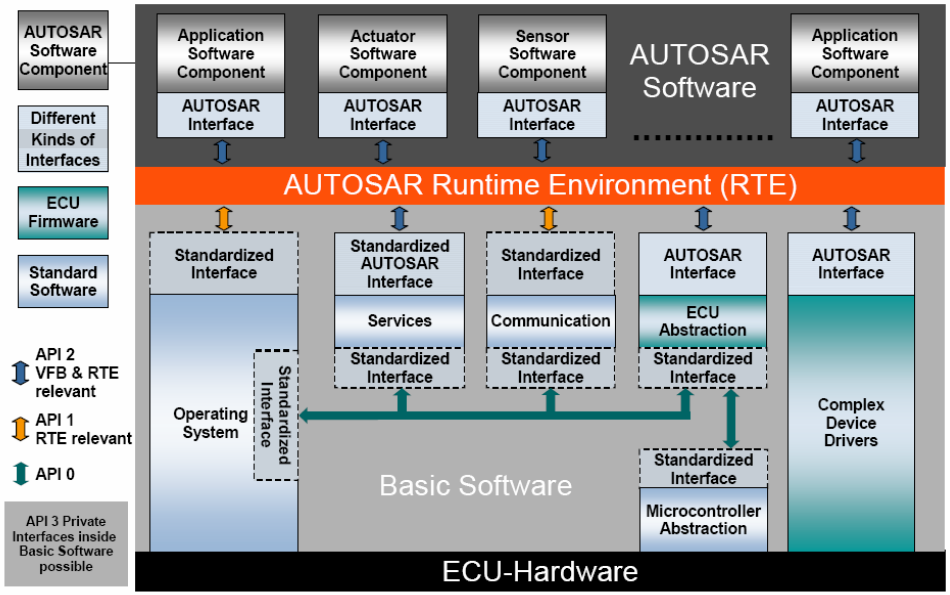
source: www.autosar.org

# Key AUTOSAR "Methodology and RTE"

- Flexible mapping of software components ...

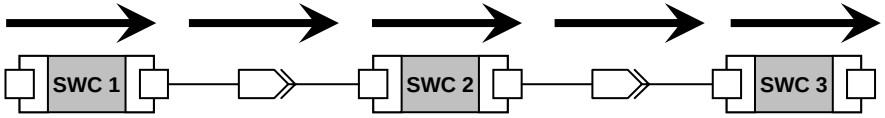



- ... enabled by standardized run-time environment (RTE)

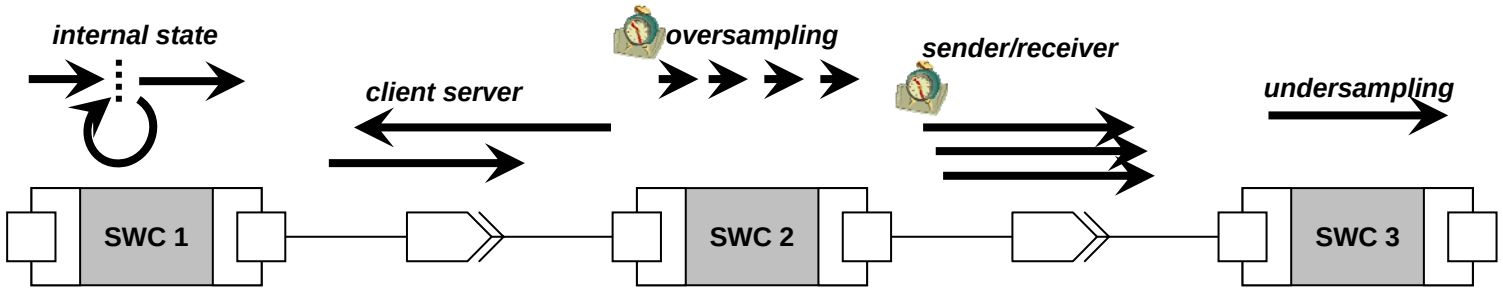


# Software Component Structure vs. Timing Dependencies

- Software component view captures "logical" dependencies (data flow)



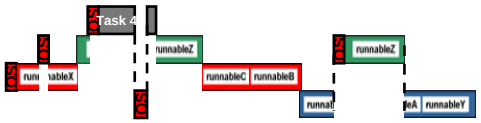

- Timing dependencies can be very different!!!
  - time-driven and event-driven activation
  - send/recv and client/server communication (remote procedure call)
  - over- / undersampling



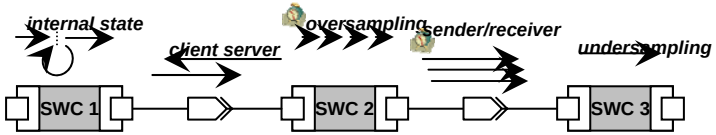
# Summary: Local Timing Effects

Complex timing

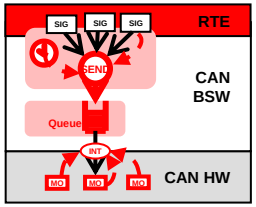
- is not directly reflected in the **software architecture**
- is induced by the **execution platform!**
  - runnables and tasks




  - timing dependencies and communication semantics




  - non-standardized drivers and middleware (BSW)




- etc...

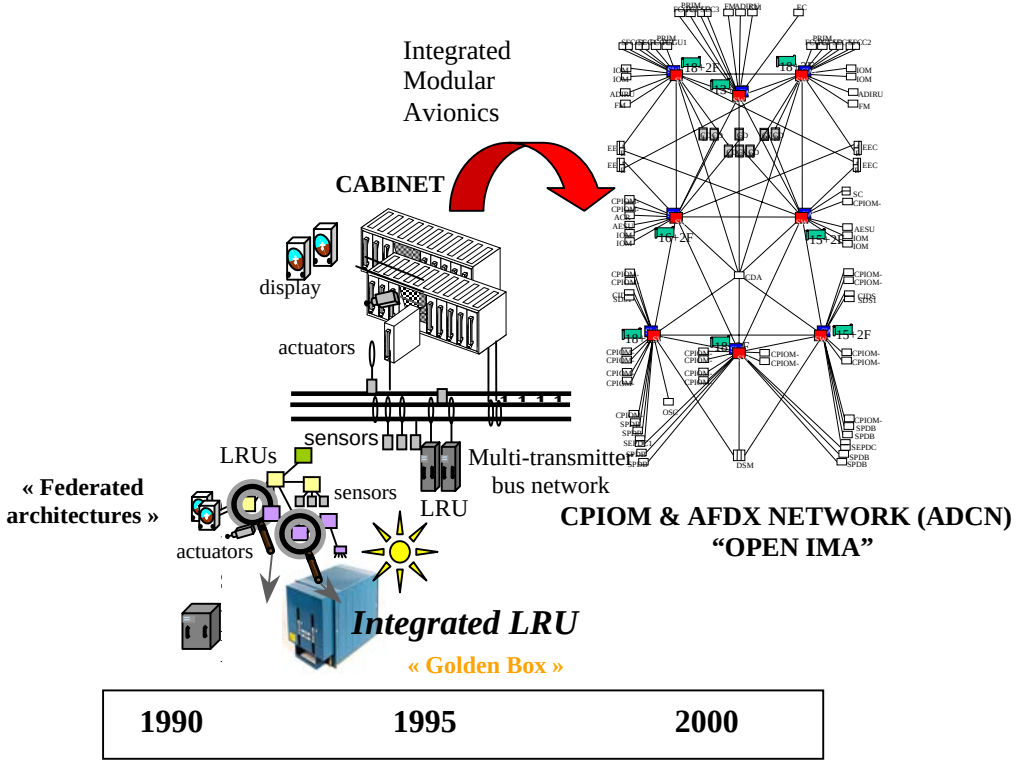



*Figure 2 – Concepts and evolution of avionic integration and modularization*

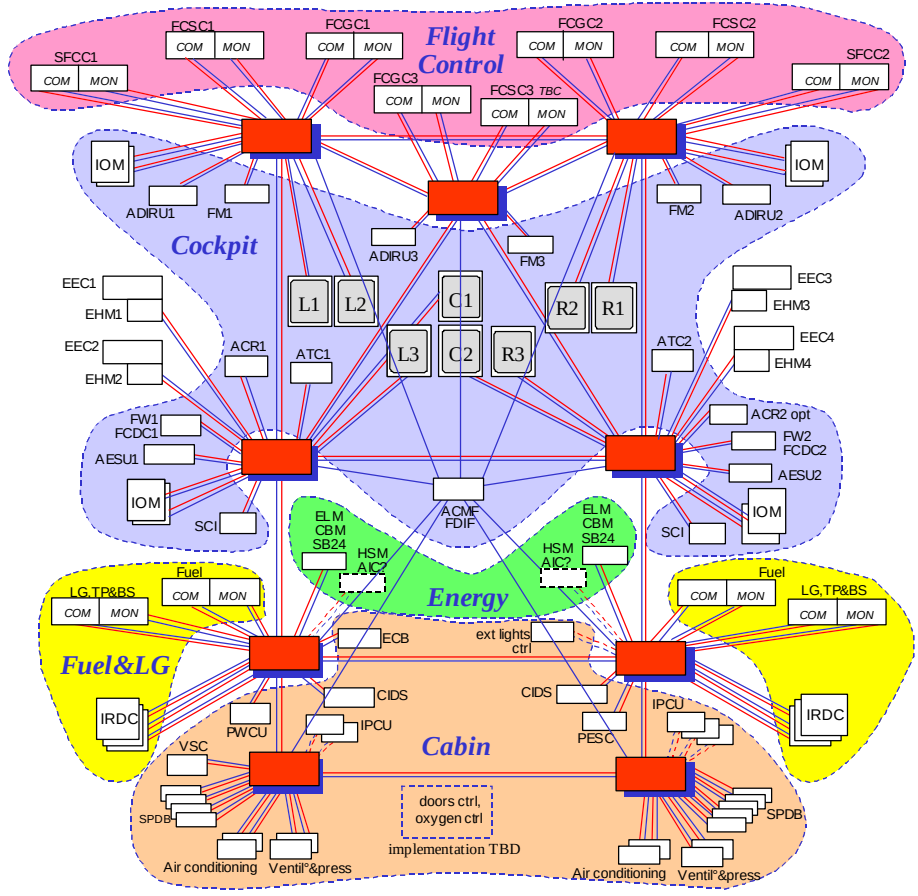



*Figure 3 – The "Open IMA" network on the A380*

# Distributed architectures for embedded systems: design criteria

- Allow for complex OEM-supplier chains:
  - Modular design, reusability
  - Migrate from the integration of *physical subsystems* to the integration of *services*
- Ensure safety:
  - Fault tolerance
  - Compartmentalization
  - Clean modular design
- Address system level safety, performance, exploitation cost, and upgradeability

# Some important consequences of architectures from control viewpoint

- Trigger of every architecture component (ECU, bus…) must be *time*, not events – otherwise the failure of one component can block other components and impair the system

# Some important consequences of architectures from control viewpoint

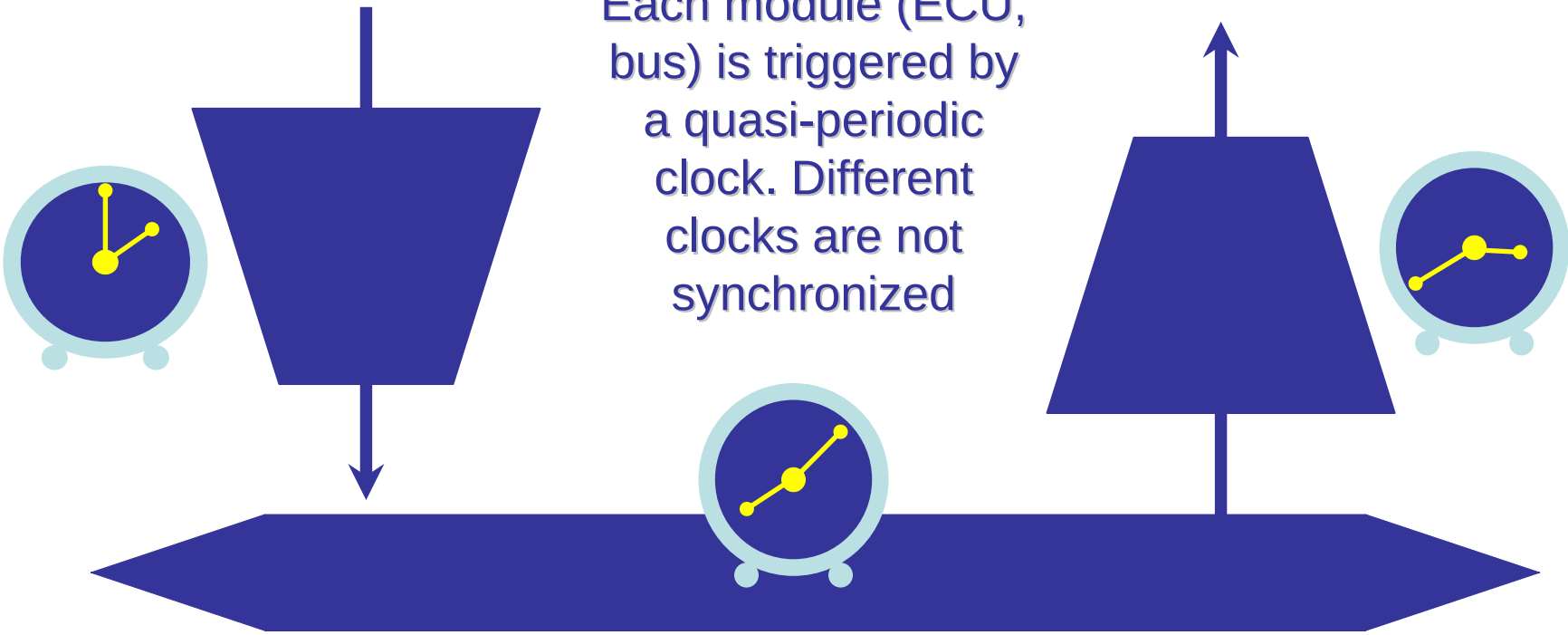
- Trigger of every architecture component (ECU, bus…) must be *time*, not events – otherwise the failure of one component can block other components and impair the system
- Kopetz Time-Triggered Architecture achieves this by offering a global architecture model of perfectly periodically sampled system, for use with Time Division Multiple Access

# Some important consequences of architectures from control viewpoint

- Trigger of every architecture component (ECU, bus…) must be *time*, not events – otherwise the failure of one component can block other components and impair the system
- Kopetz Time-Triggered Architecture achieves this by offering a global architecture model of perfectly periodically sampled system, for use with Time Division Multiple Access
- Alternative, more flexible, distributed architectures are used – e.g., by Airbus

# Loosely Time-Triggered Architecture

name invented by [Benveniste, Caillaud, Caspi, Sangiovanni-Vincentelli 2002]

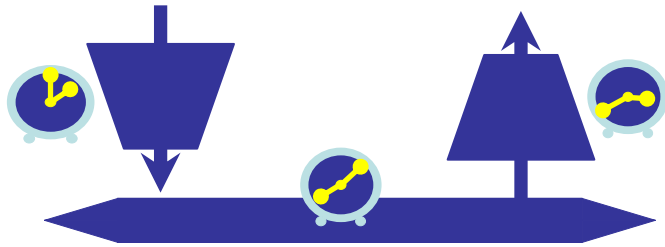



Values are sustained in write/bus/read buffers

# Instances of LTTA

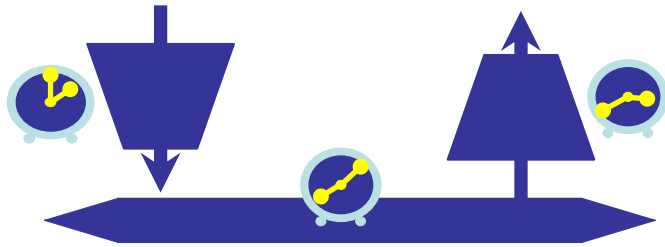
- Airbus: ARINC 653 + AFDX switched Ethernet
  - Not quite used for distributed continuous control yet
  - up to and included A380, flight control loops are deployed on a single computer
  - still, redundancy brings distributed architectures through voters
  - In the future (A350?), highly distributed control loops with distributed intelligence will be considered
- LTTA is the architecture of choice for distributed control with wireless comm. (flight formations)
- LTTA is actually found under many different names in distributed control

# Use for continuous control



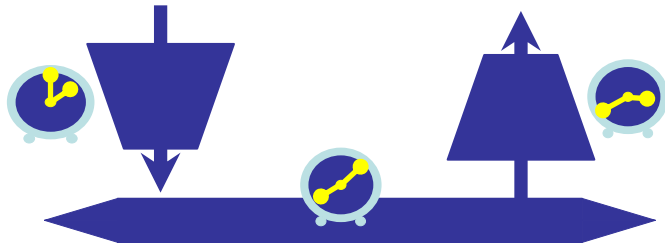
- Have all clocks deviating from an *ideal* periodic clock by:
  - Bounded drift: $T-d \leq \tau_n \leq T+D$ where $\tau_n$ is the actual, possibly time-varying, period, and $d,D$ are small w.r.t. $T$
  - Bounded jitter
- Just implement continuous control by autonomous sampling according to the local clocks

# Use for continuous control



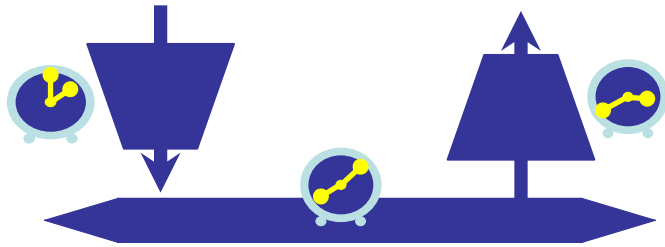
- Have all clocks deviating from an *ideal* periodic clock by:
  - Bounded drift: $T-d \leq \tau_n \leq T+D$ where $\tau_n$ is the actual, possibly time-varying, period, and $d,D$ are small w.r.t. $T$
  - Bounded jitter
- Just implement continuous control by autonomous sampling according to the local clocks
- Simple and elegant; seems like a very good way of sampling continuous control; yet, $\neq$ from control design with delta operators

# Use for continuous control

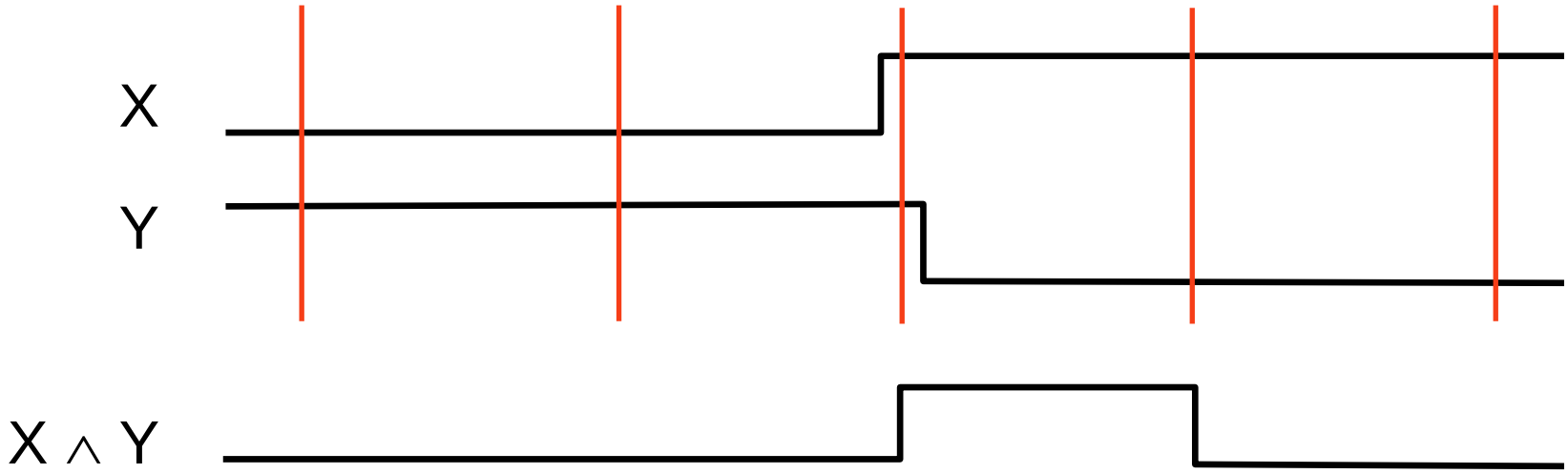


- Have all clocks deviating from an *ideal* periodic clock by:
  - Bounded drift: $T-d \leq \tau_n \leq T+D$ where $\tau_n$ is the actual, possibly time-varying, period, and $d,D$ are small w.r.t. $T$
  - Bounded jitter
- Just implement continuous control by autonomous sampling according to the local clocks
- **Is this architecture model covered by existing robust control design frameworks? No**

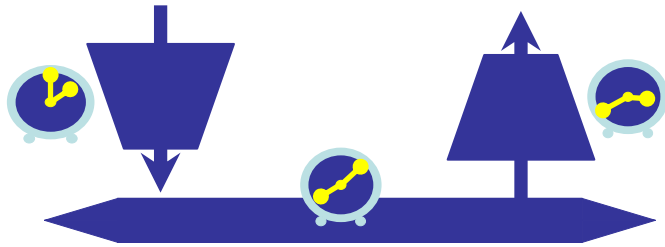
# Use for discrete control



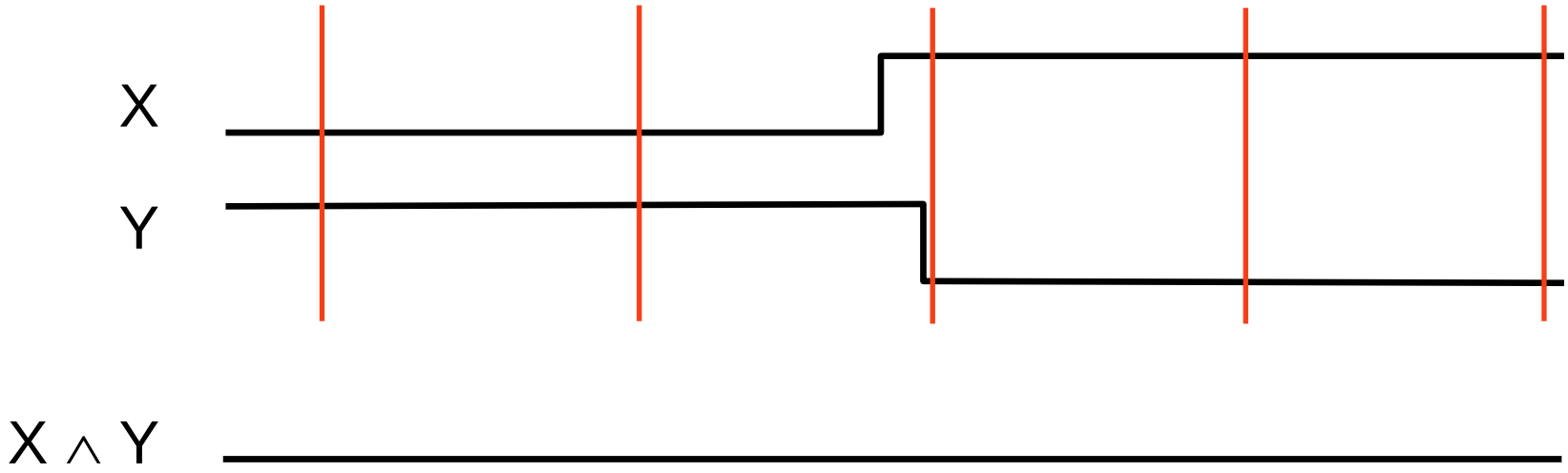
- Problem: continuity of trajectories in continuous control is essential in justifying the autonomous sampling technique; unfortunately, discrete systems (automata) are not robust w.r.t. sampling uncertainties:



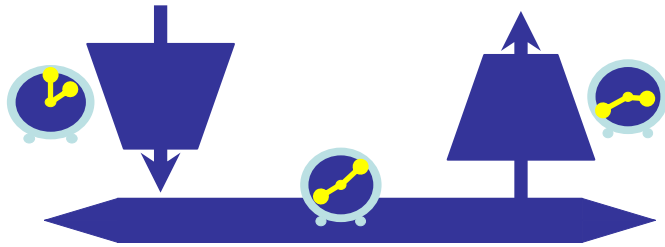
# Use for discrete control



- Problem: continuity of trajectories in continuous control is essential in justifying the autonomous sampling technique; unfortunately, discrete systems (automata) are not robust w.r.t. sampling uncertainties:



# Use for discrete control

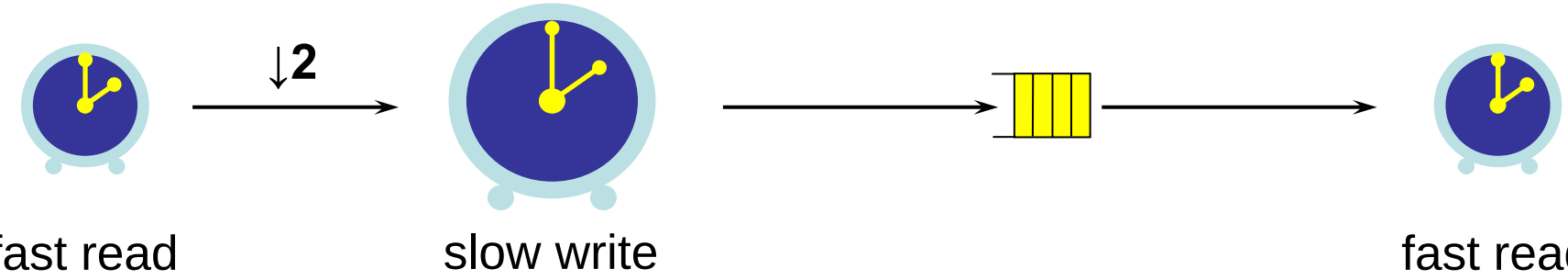


- Solution: develop a middleware that offers
  - a distributed, global, and uniform *logical* time,
  - with bounded relative jitter between different sites
- Key building blocks of this middleware follow

# LTTA middleware: key idea (1)

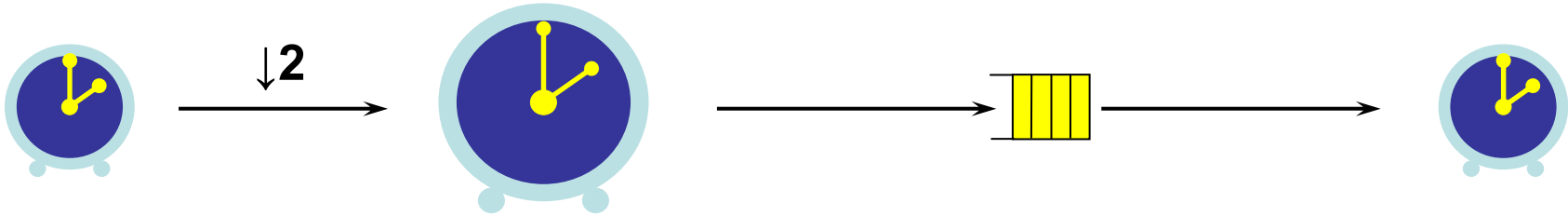
- Let **[w → r]** denote the result of a *reader* r sampling the successive values posted by a *writer* w.
  - If r samples more frequently than w, no data will be lost; repetitions can be taken care of by marking data with an alternating bit.
- **[[$s_1$ → $s_2$] → $s_3$]** is the cascade of $s_1$ transmitting to $s_2$, which then transmits to $s_3$.
  - Problem: seems to require cascaded slow-downs, thus prohibiting bi-directional communications.

# LTTA middleware: key idea (2) clock regeneration



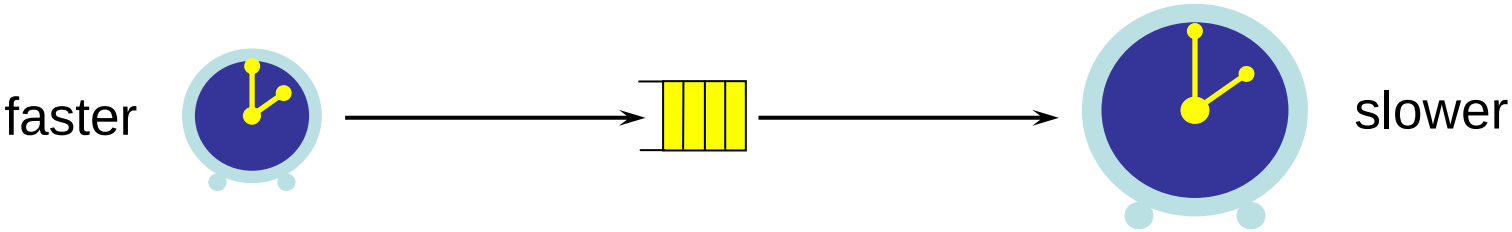

- The two sites possess loosely synchronized clocks, triggering *fast reads*
- They write at a clock downsampled by a factor 2 $\Rightarrow$ no data is lost
- Duplications are cleaned up by using an alternating bit

# LTTA middleware: key idea (2) clock regeneration




- The two sites possess loosely synchronized clocks, triggering *fast reads*
- They write at a clock downsampled by a factor 2 ⇒ no data is lost
- Duplications are cleaned up by using an alternating bit

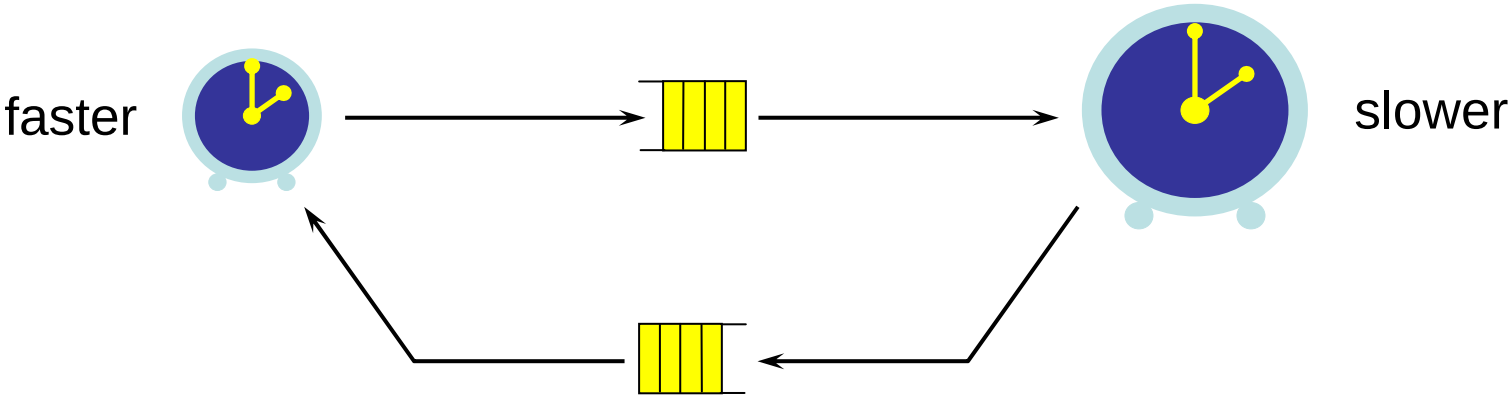
# LTTA middleware: key idea (3)
# a feedback mechanism for traffic shaping




**If nothing done, the faster will overload the slower**

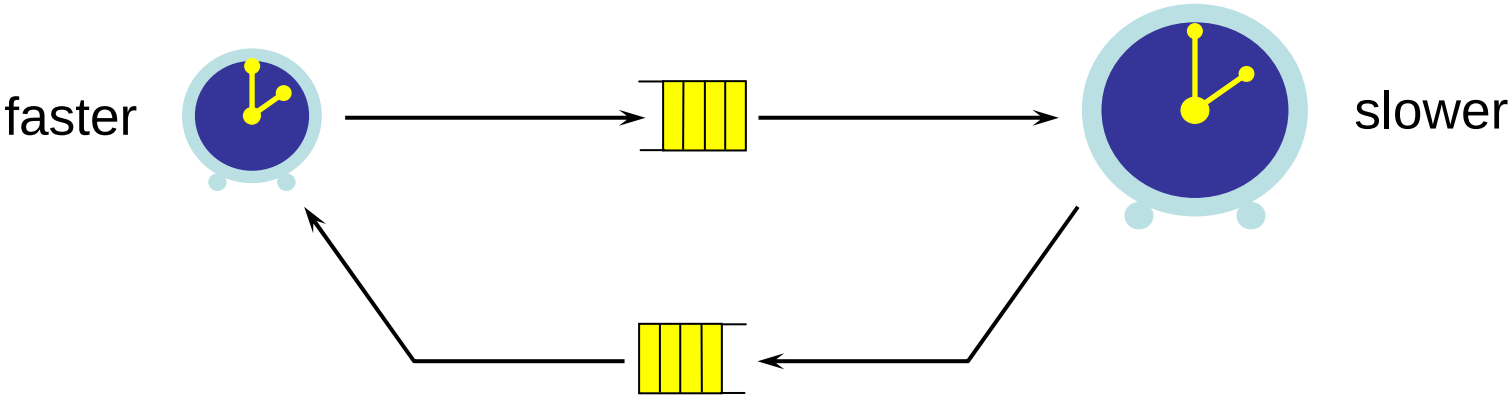
**What can be done?**

# LTTA middleware: key idea (3)
# a feedback mechanism for traffic shaping



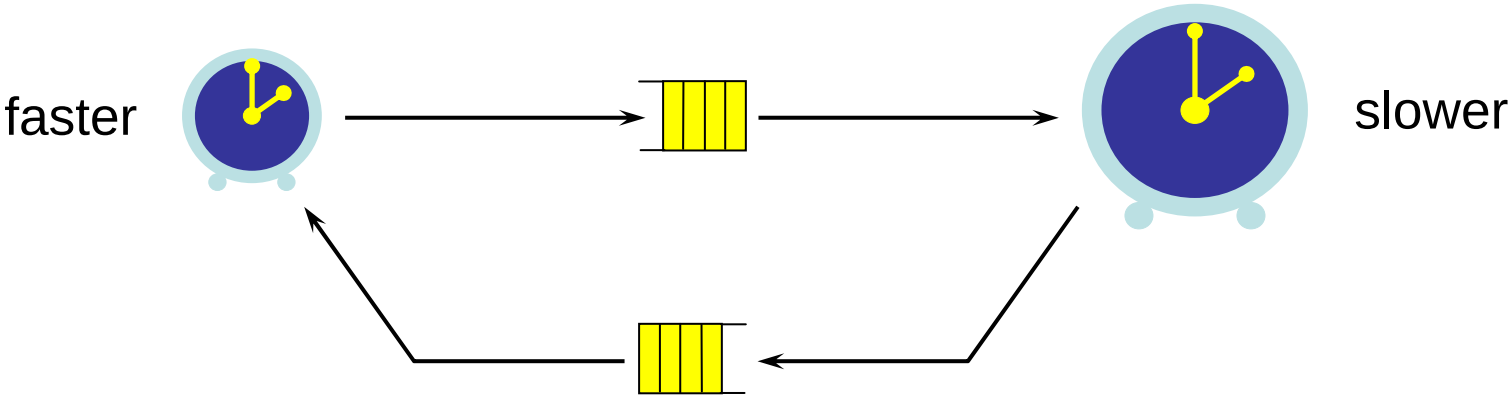

1. Assume a reverse communication channel

# LTTA middleware: key idea (3)
# a feedback mechanism for traffic shaping




1. Assume a reverse communication channel
2. Monitor the excess counter
3. Skip emission when excess counter gets close to buffer capacity

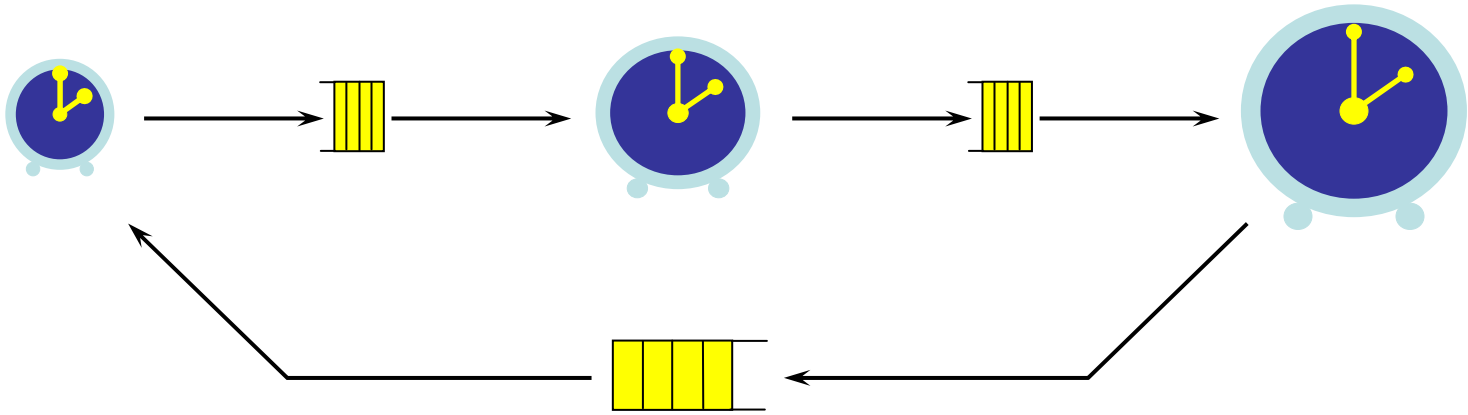
# LTTA middleware: key idea (3)
# a feedback mechanism for traffic shaping




1. Assume a reverse communication channel
2. Monitor the excess counter
3. Skip emission when excess counter gets close to buffer capacity

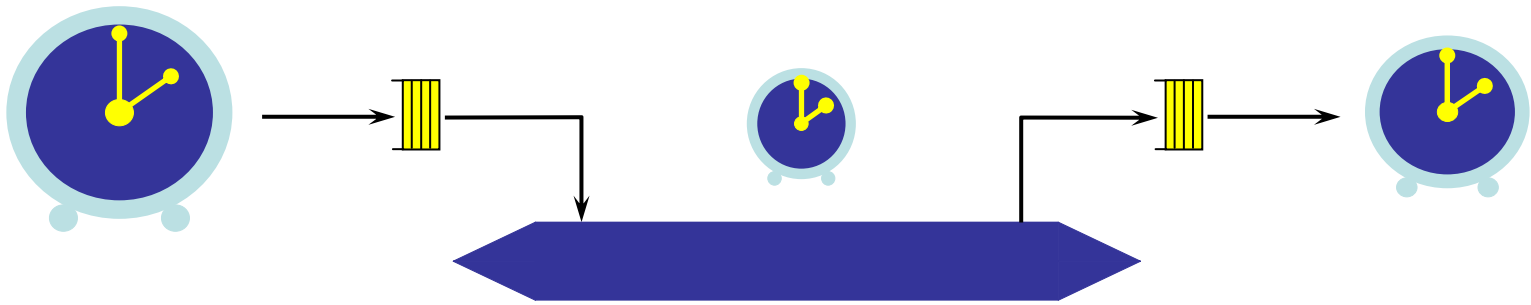
**Result: this feedback loop preserves data flows and ensures bounded relative jitter**

# LTTA middleware: key idea (3) feedback mechanism can be cascaded



**Result: this feedback loop preserves data flows and ensures bounded relative jitter**

# LTTA middleware: key idea (4)
# LTT bus

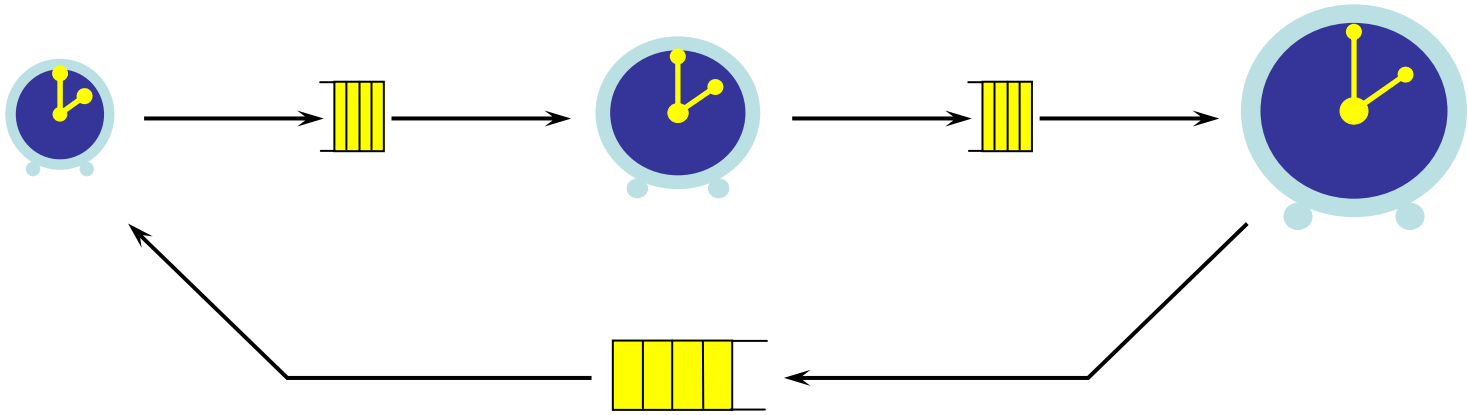


If the bus is faster than both writer and reader,

and the integer part of the ratio $T_w / T_r$ is smaller than the bus period,

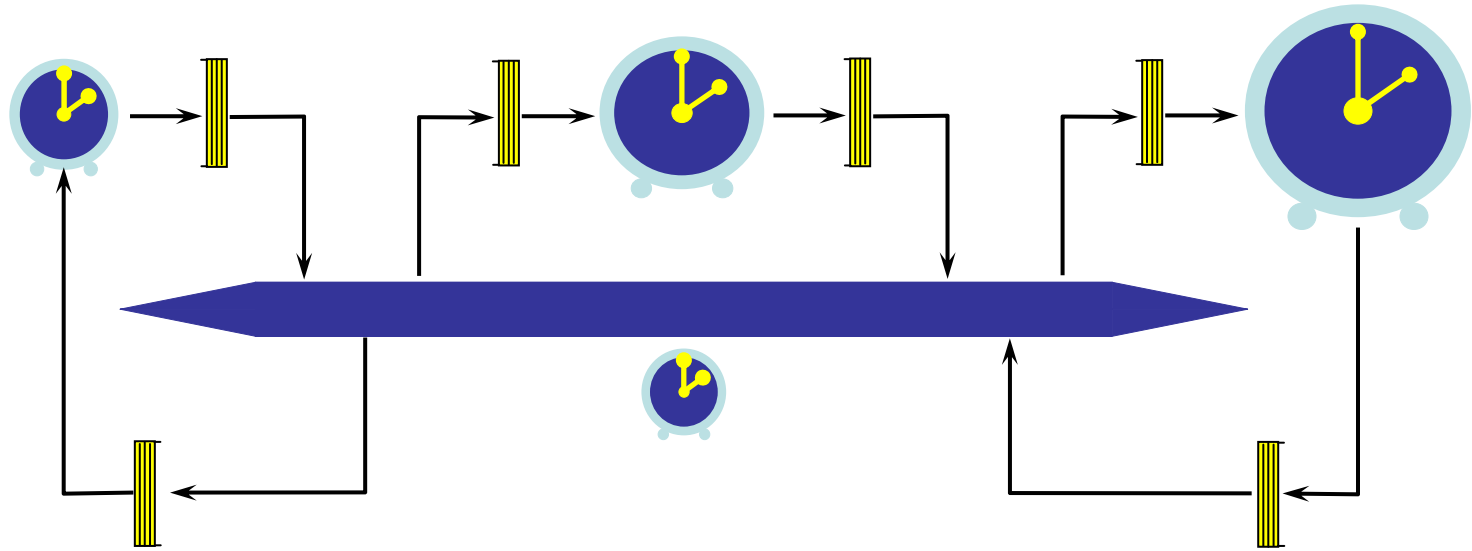
then $[[w \rightarrow b] \rightarrow r]$ preserves the data flows

# LTTA middleware: combining ideas (3) and (4)



1. Start from overlay LTT network

# LTTA middleware: combining ideas (3) and (4)



1. Start from overlay LTT network
2. Replace direct buffer communication by LTT bus communication

# LTTA middleware, a key tool: the excess counter monitoring algorithm

- Using the same excess counter monitoring algorithm

$$N_t = max\,[\,N_{t-1} + X_t\,,\,0\,]\,,\ \text{where}\ X_t \in \{-1,0,+1\}$$

  various services can be developed:
  - Access control mechanisms
  - Bus guardians
  - Voters

- Here we use this algorithm for monitoring strict bounds. This algorithm is in fact originating from quality control in sequential statistics, where it is used to detect changes in populations. Thus we are prepared to lift LTTA to QoS based adaptive systems.

# Concluding remarks

- So far we have satisfactory solutions
  - for distributed continuous control
  - for discrete control
- However, these solutions do not seem easily compatible
- How to get a global solution? Still open.

# Concluding remarks

- So far we have satisfactory solutions
  - for distributed continuous control
  - for discrete control
- However, these solutions do not seem easily compatible
- How to get a global solution? Still open.
- Paul Caspi has proposed ideas to bring in topology, for mixed continuous/discrete systems, where time is subject to jitter
- This looks like a very promising topic for his active retirement and we all are confident that Paul will remain a rising and shining star for ever