this talk is posted at http://chess.eecs.berkeley.edu/pubs/472.html

Component Architectures for Time-Sensitive Systems

Part 2

Edward A. Lee

Robert S. Pepper Distinguished Professor and

The Onassis Foundation Science Lecture Series
The 2008 Lectures in Computer Science
Embedded Networked Systems: Theory and Applications

With thanks to Thomas Huning Feng, Yang Zhao, and Ye (Rachel) Zhou

Heraklion, Crete
July 24-28, 2008

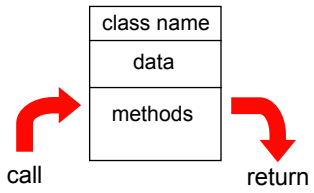
Our Solution

Reintroduce time into the core abstractions:

- *Foundations:* Timed computational semantics.
- *Bottom up*: Make timing repeatable.
- *Top down*: Timed, concurrent components.
- *Holistic*: Model engineering.

Object Oriented vs. Actor Oriented

The established: Object-oriented:

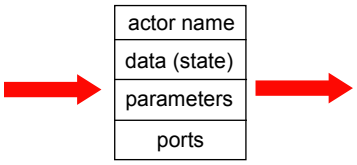


What flows through an object is sequential control

Things happen to objects

The alternative: Actor oriented:

Actors make things happen



What flows through an object is evolving data

Our Agenda

I will show a particular approach to the design of concurrent and distributed time-sensitive systems that is an actor-oriented component technology.

The approach is called PTIDES (pronounced “tides”), for Programming Temporally Integrated Distributed Embedded Systems.

[1] Y. Zhao, E. A. Lee, and J. Liu, "A Programming Model for Time-Synchronized Distributed Real-Time Systems," in Real-Time and Embedded Technology and Applications Symposium (RTAS), Bellevue, WA, USA, 2007.

[2] T. H. Feng, E. A. Lee, H. D. Patel, and J. Zou, "Toward an Effective Execution Policy for Distributed Real-Time Embedded Systems," in 14th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), St. Louis, MO, USA, 2008.

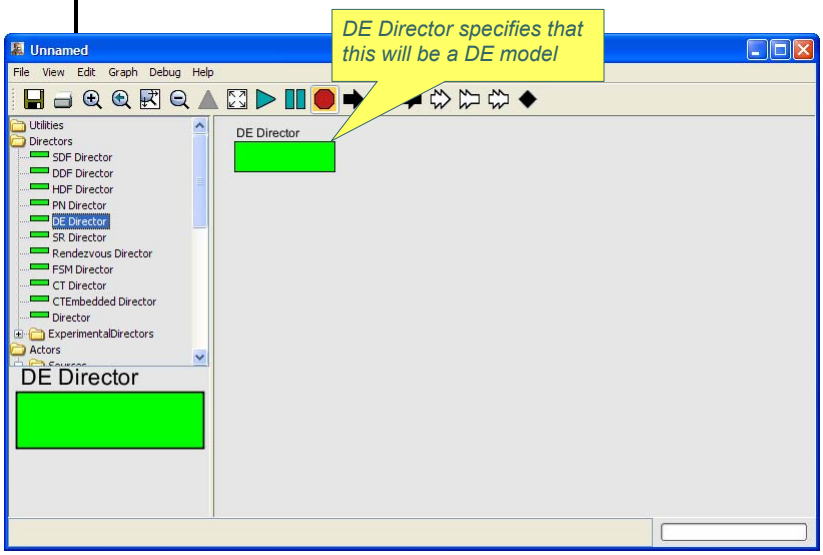
Our Approach is based on Discrete Events (DE)

- Concurrent actors
- Exchange time-stamped messages

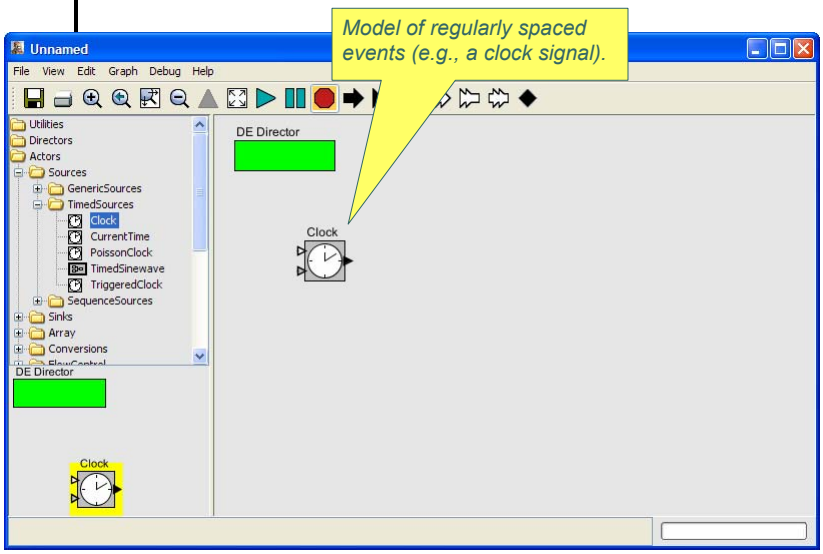
A correct execution is one where every actor reacts to input events in time-stamp order.

Time stamps are in “model time,” which typically bears no relationship to “real time” (wall-clock time).

Example

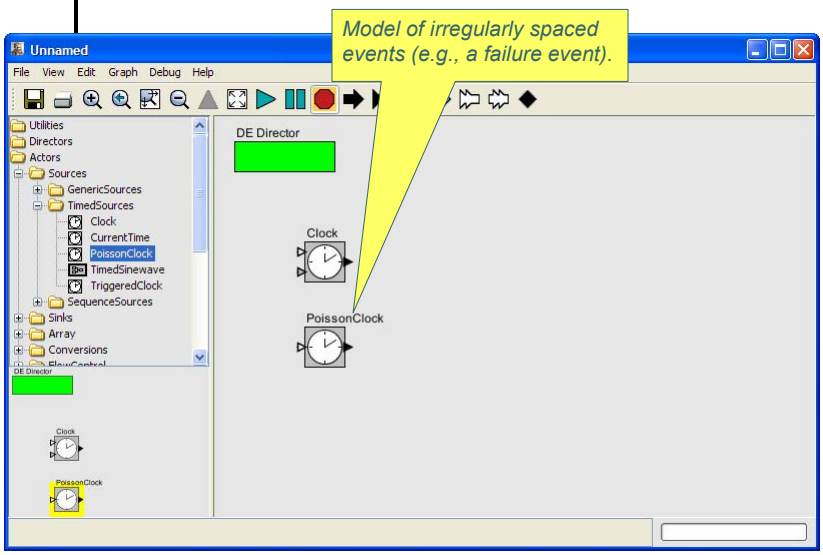


Example



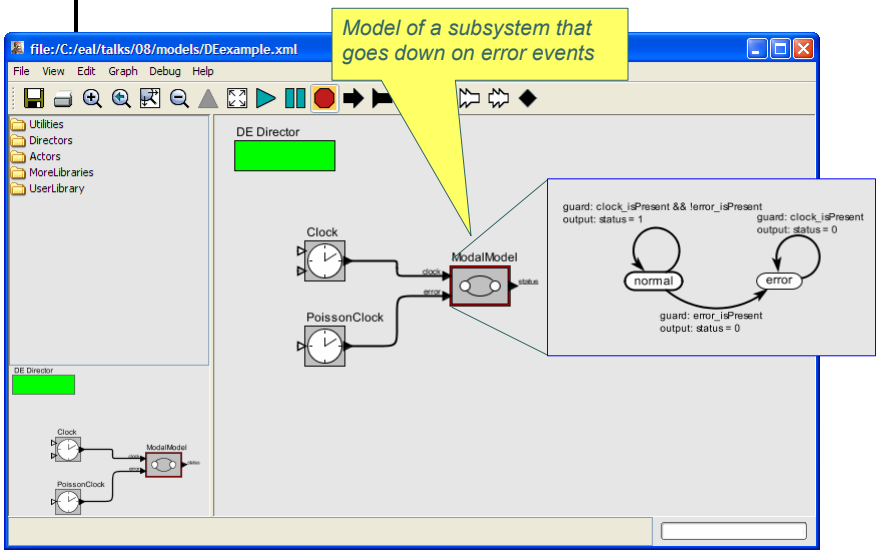
Lee, Berkeley 7

Example



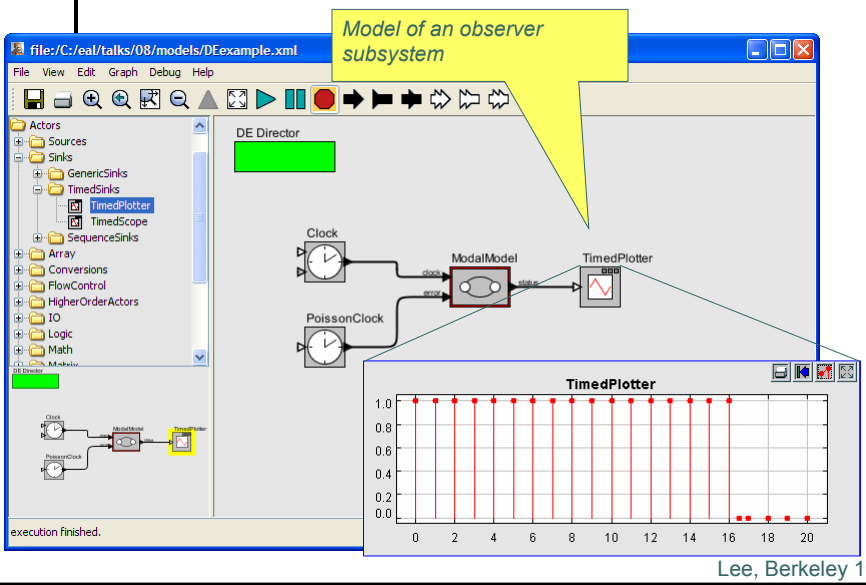
Lee, Berkeley 8

Example



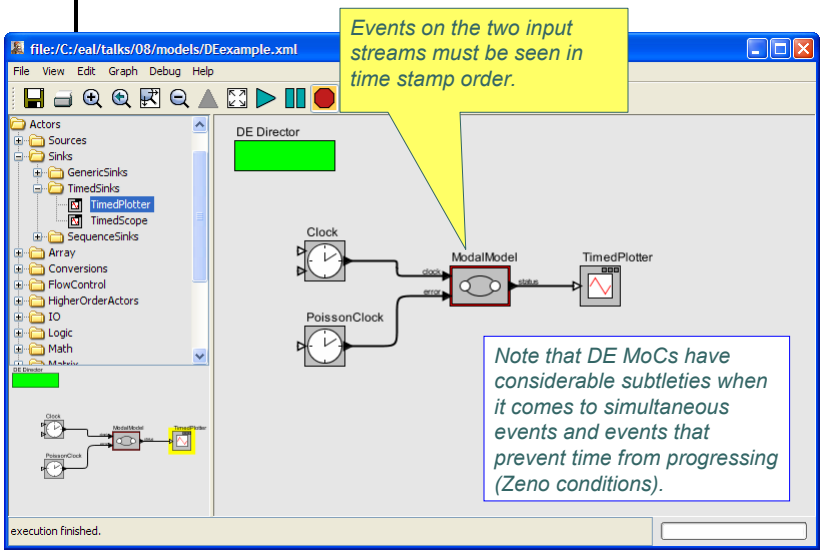
Lee, Berkeley 9

Example

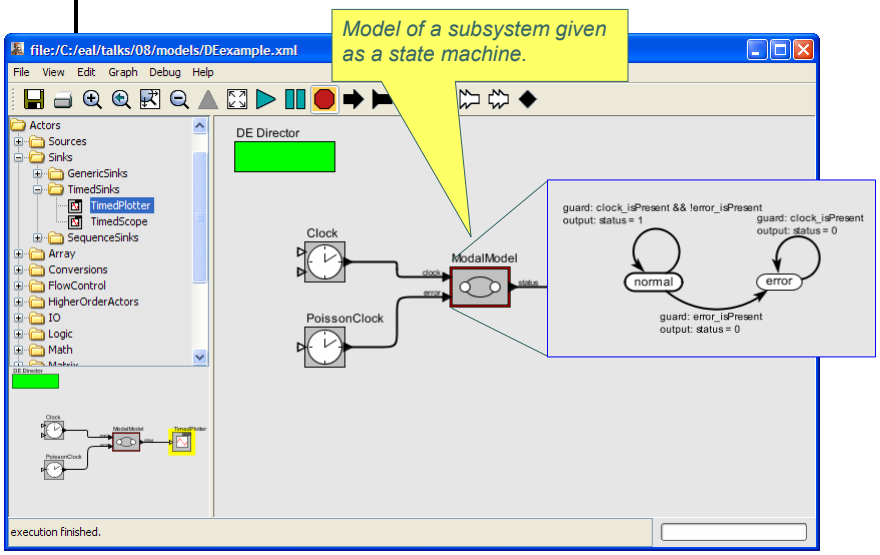


Lee, Berkeley 10

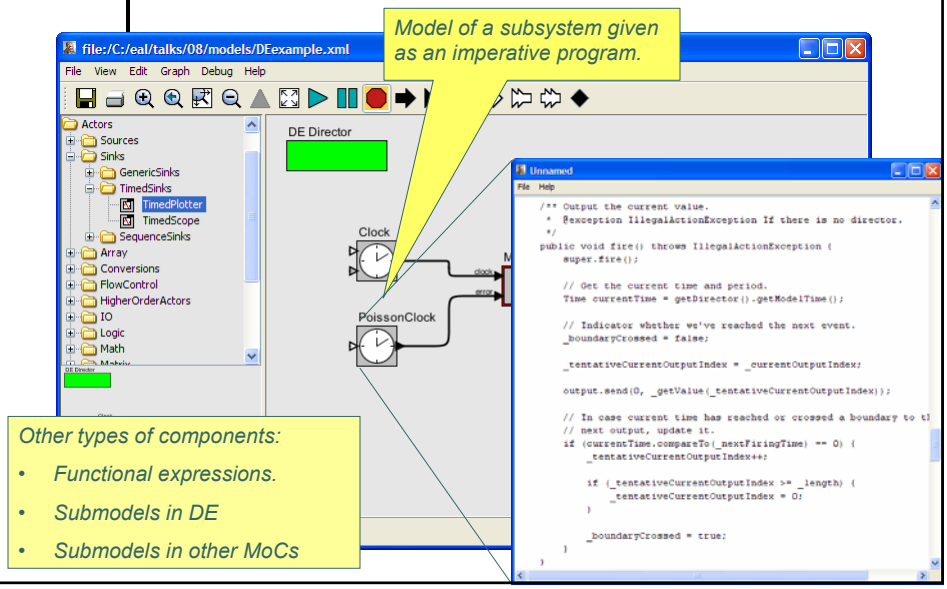
Example



This is a Component Technology



This is a Component Technology



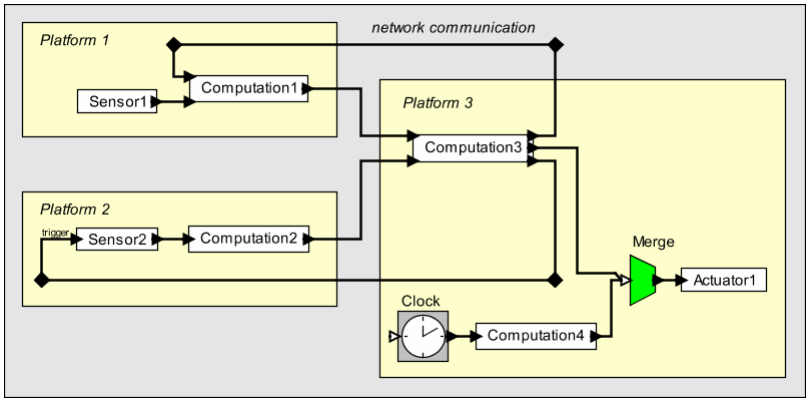
Using DE Semantics in Distributed Real-Time Systems

- DE is usually a simulation technology.
- Distributing DE is done for acceleration.
- Hardware design languages (e.g. VHDL) use DE where time stamps are literally interpreted as real time, or abstractly as ticks of a physical clock.

- We are using DE for distributed real-time software, binding time stamps to real time only where necessary.
- *PTIDES*: Programming Temporally Integrated Distributed Embedded Systems

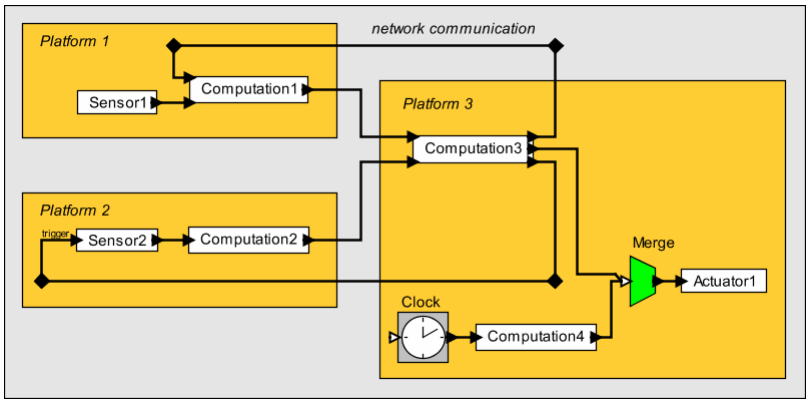
PTIDES: Programming Temporally Integrated Distributed Embedded Systems

Consider a simpler scenario:



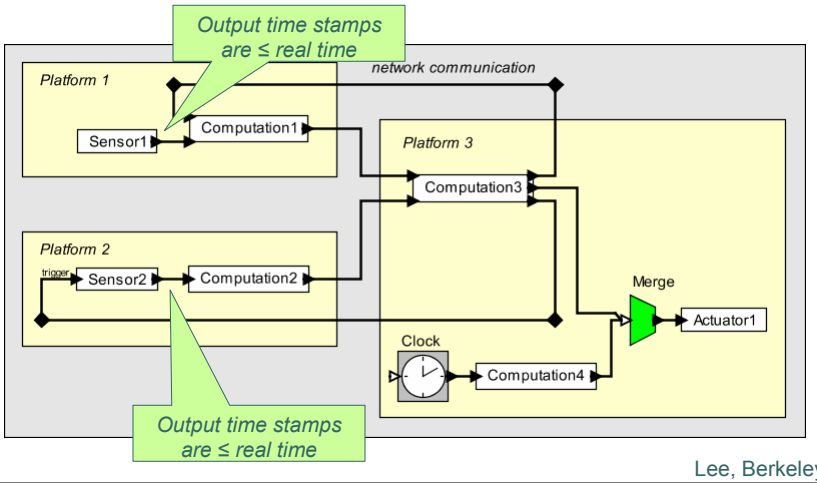
PTIDES: Programming Temporally Integrated Distributed Embedded Systems

Assumption: Wall clocks on the distributed platforms are synchronized to some known precision (e.g. NTP, IEEE 1588)



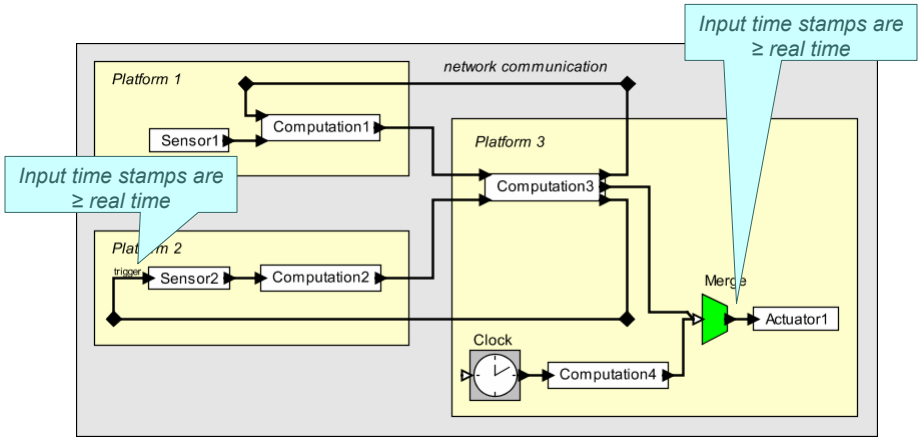
PTIDES: Programming Temporally Integrated Distributed Embedded Systems

Bind model time to real time at the *sensors*:



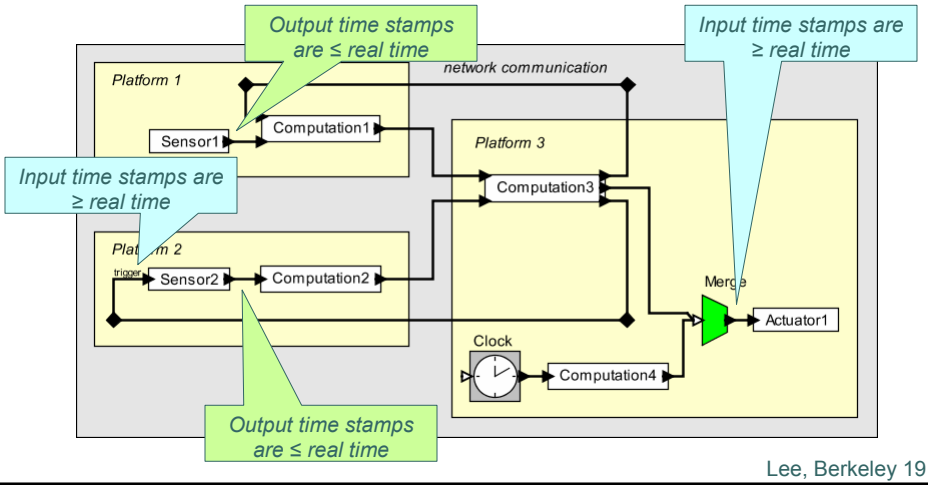
PTIDES: Programming Temporally Integrated Distributed Embedded Systems

Bind model time to real time at the *actuators*:



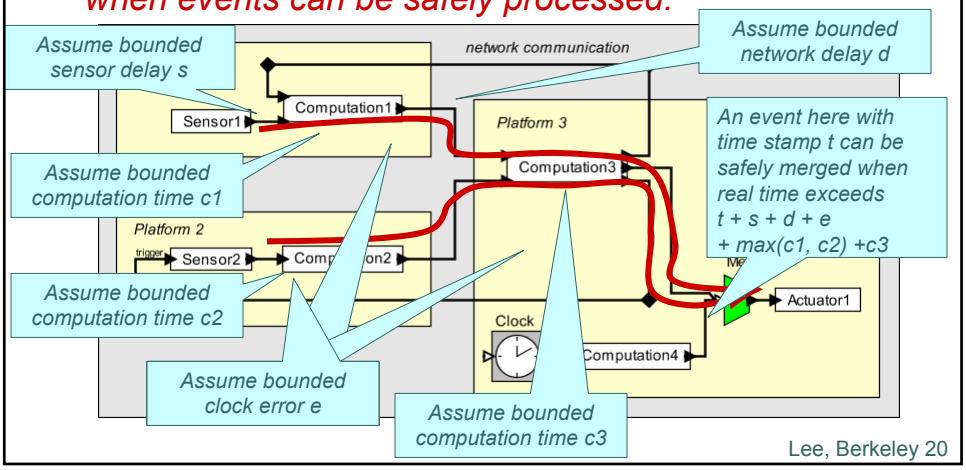
PTIDES: Programming Temporally Integrated Distributed Embedded Systems

Schedulability is not violating these timing inequalities.



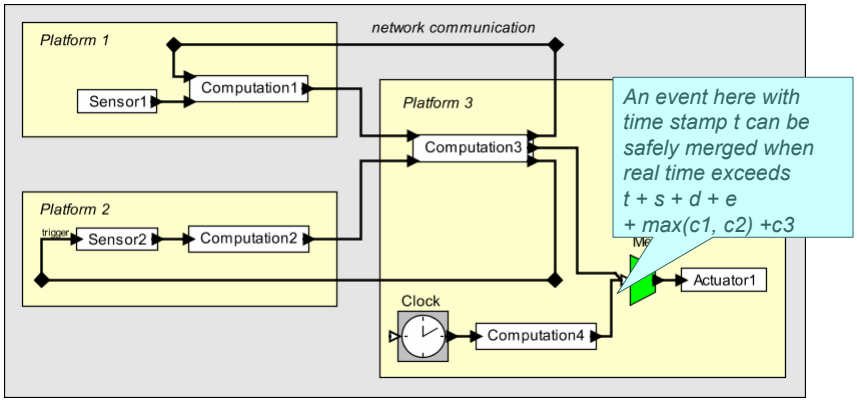
PTIDES: Programming Temporally Integrated Distributed Embedded Systems

PTIDES uses static causality analysis to determine when events can be safely processed.



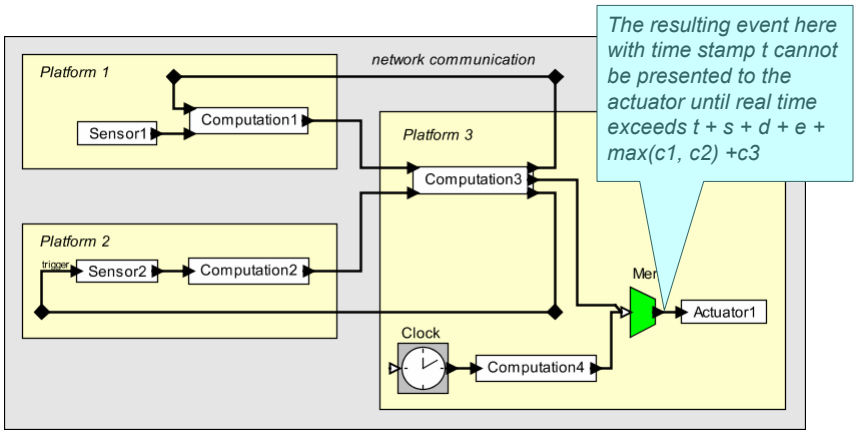
PTIDES: Programming Temporally Integrated Distributed Embedded Systems

The execution model prevents remote processes from blocking local ones, and does not require backtracking.



PTIDES: Programming Temporally Integrated Distributed Embedded Systems

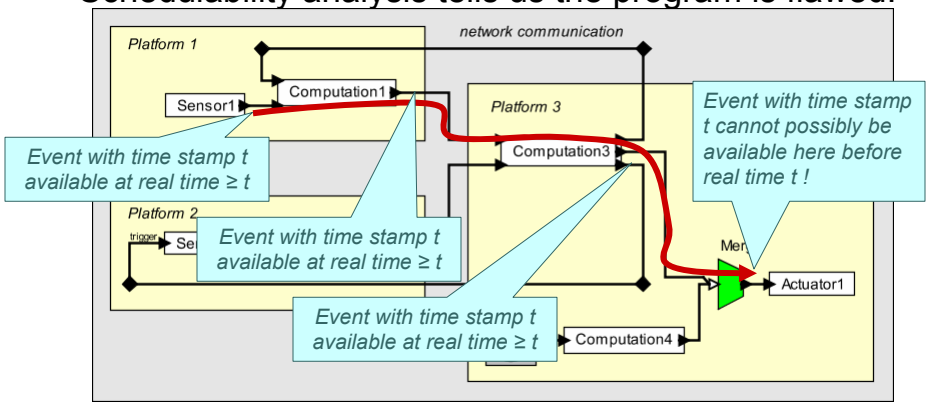
However, this program is not schedulable!



PTIDES: Programming Temporally Integrated Distributed Embedded Systems

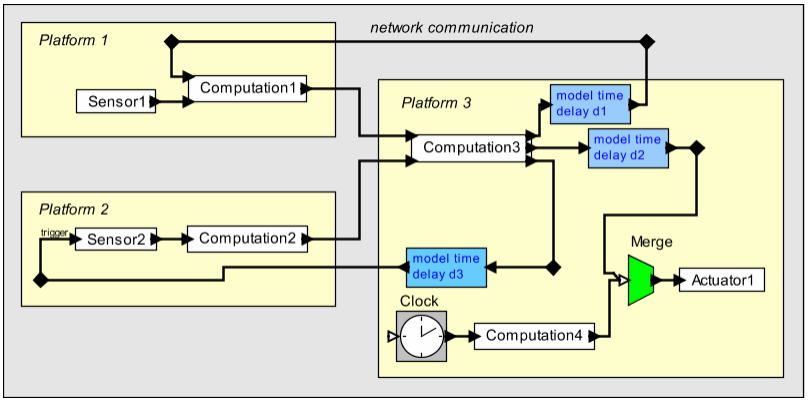
Remote events also trigger real-time violations.

Schedulability analysis tells us the program is flawed.



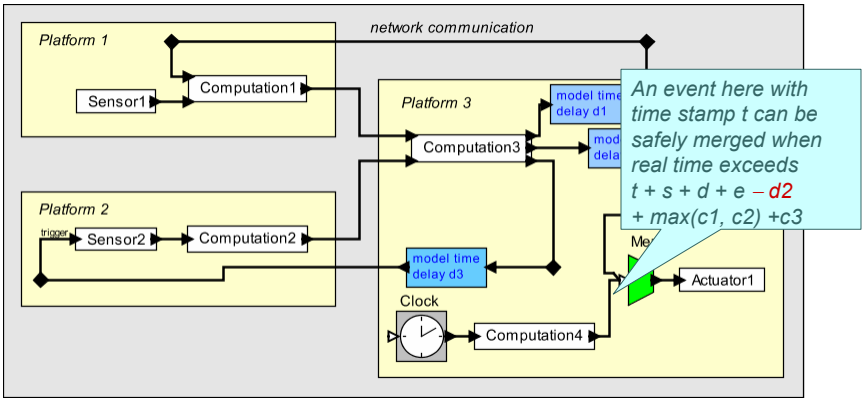
PTIDES: Programming Temporally Integrated Distributed Embedded Systems

The program can be fixed with actors that increment the time stamps (model-time delays).



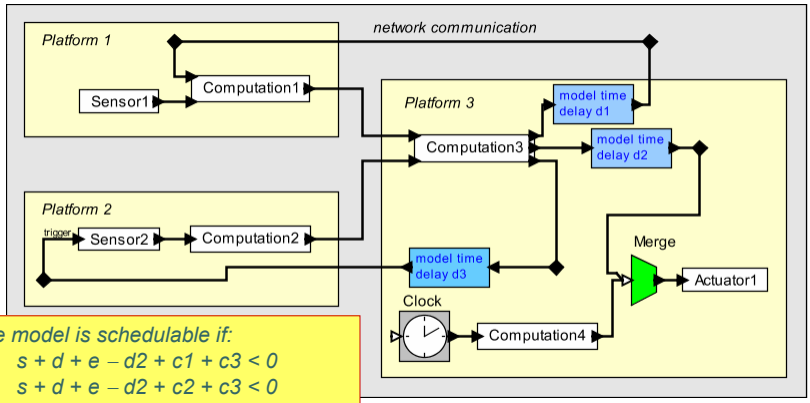
PTIDES: Programming Temporally Integrated Distributed Embedded Systems

This relaxes scheduling constraints...



PTIDES: Programming Temporally Integrated Distributed Embedded Systems

Through static analysis we can derive sufficient conditions for schedulability…

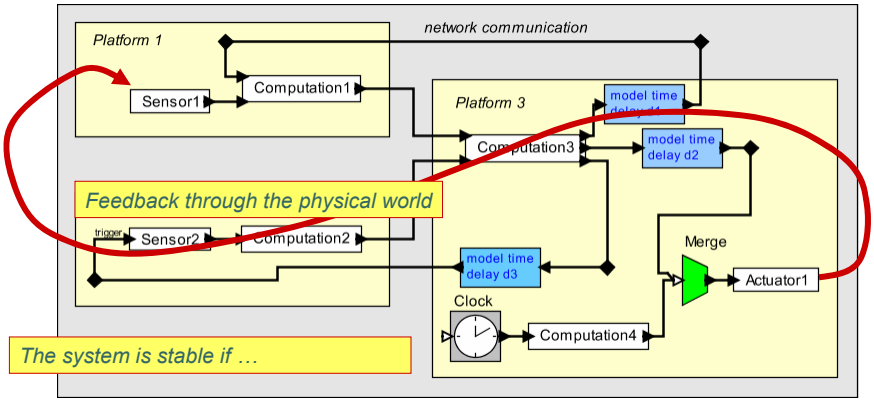


The model is schedulable if:

1) $s + d + e - d2 + c1 + c3 < 0$
2) $s + d + e - d2 + c2 + c3 < 0$
3) …

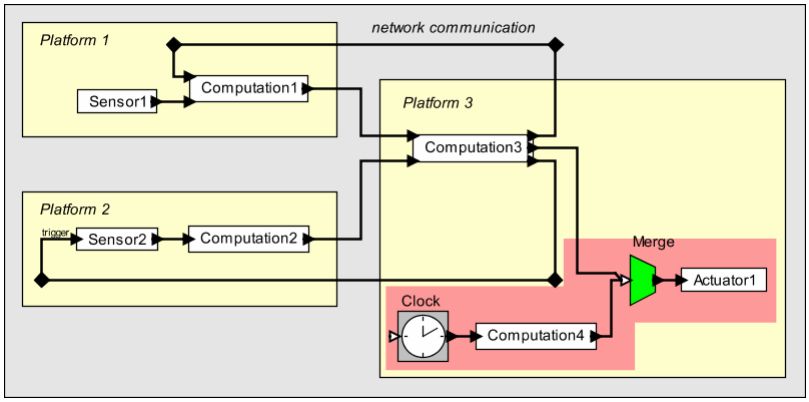
PTIDES: Programming Temporally Integrated Distributed Embedded Systems

… and being explicit about time delays means that we can analyze control system dynamics…



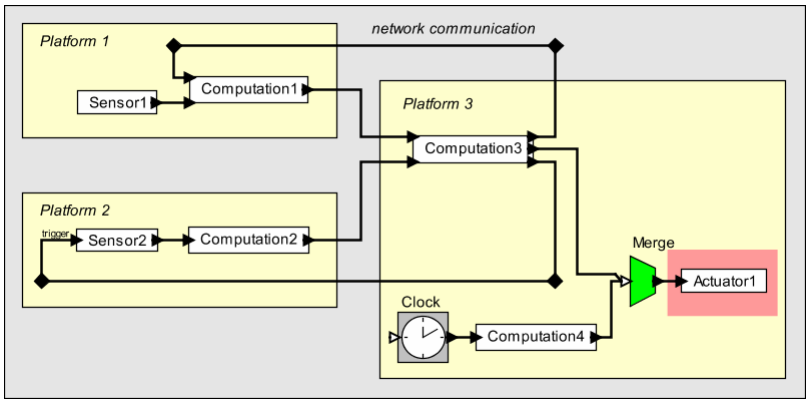
Compare with Classical Distributed DE Simulation Technologies

Conservative distributed DE (Chandy & Misra) would block actuation unnecessarily.



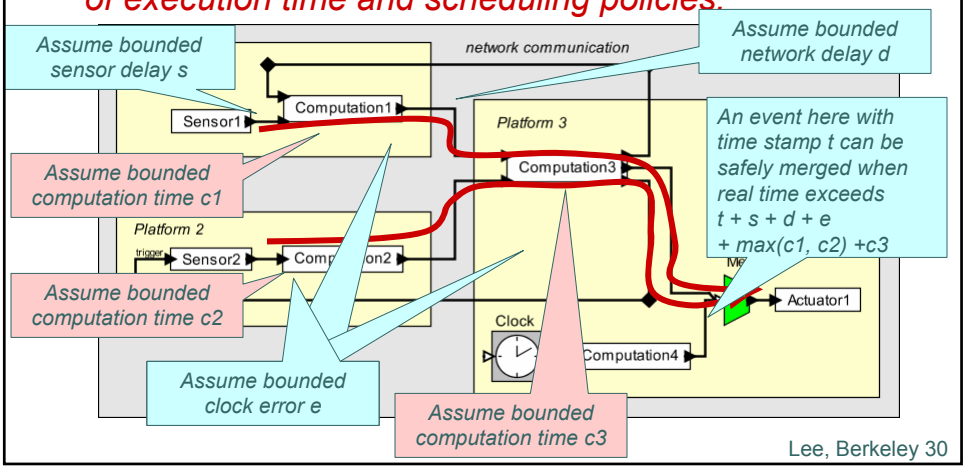
Compare with Classical Distributed DE Simulation Technologies

Optimistic distributed DE (Jefferson) would require being able to roll back the physical world.



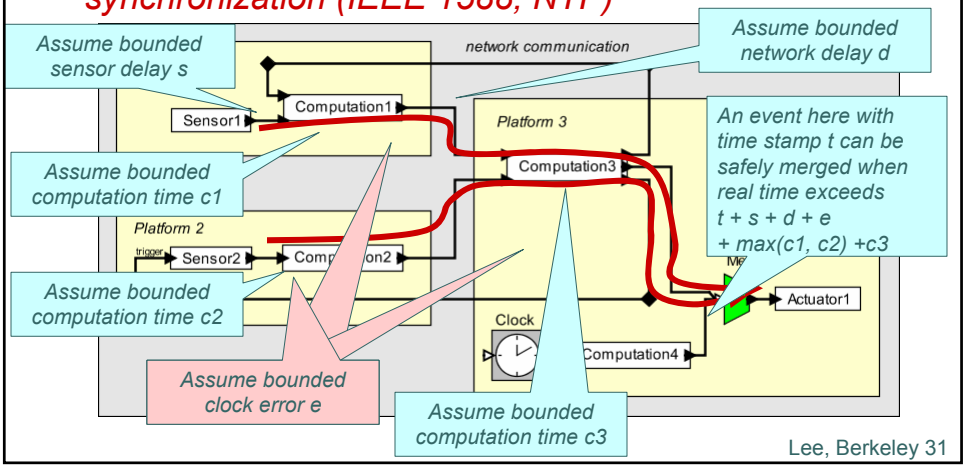
But this schedulability analysis is not quite as easy as it might look

Bounding computation time requires careful analysis of execution time and scheduling policies.



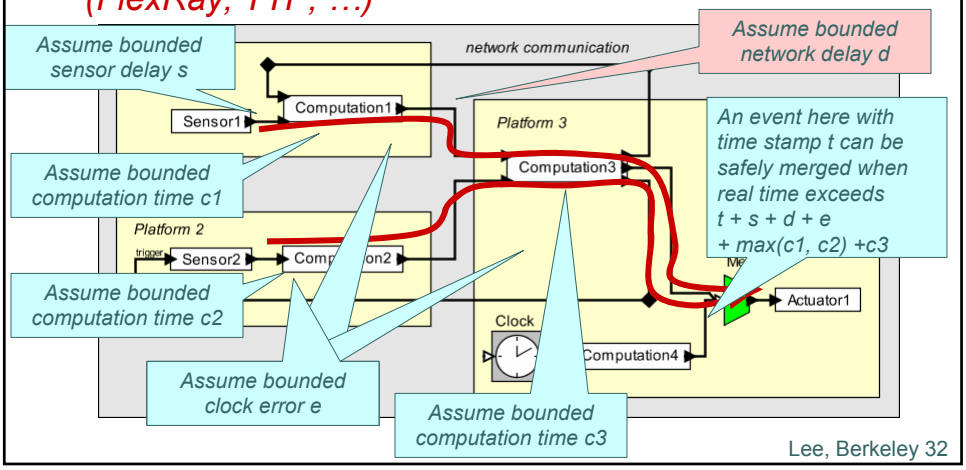
But this schedulability analysis is not quite as easy as it might look

Bounding clock error requires network time synchronization (IEEE 1588, NTP)



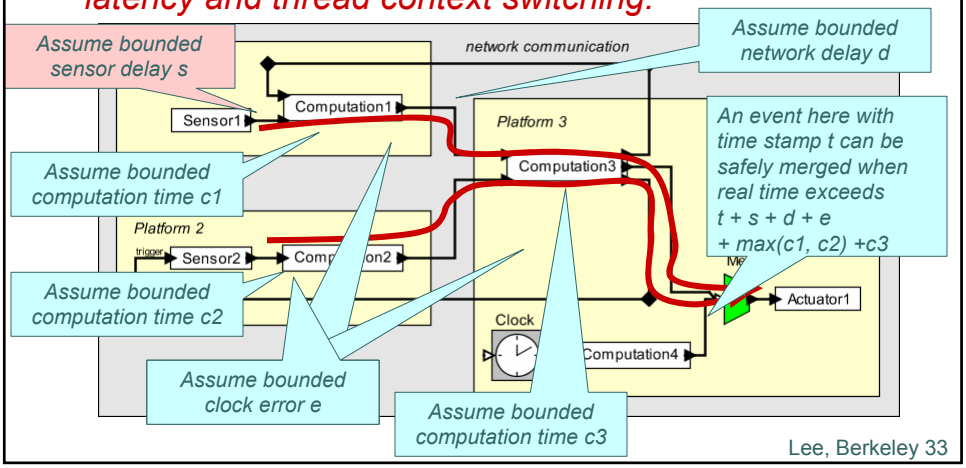
But this schedulability analysis is not quite as easy as it might look

Bounding network delay requires a real-time network (FlexRay, TTP, …)



But this schedulability analysis is not quite as easy as it might look

Bounding sensor delay requires bounding interrupt latency and thread context switching.



Making this Systematic

Levels of analysis:

1. Assume zero execution time for actors (exposes modeling errors)
2. Assume known worst-case execution time (WCET) for actors, unbounded compute resources (exposes complexity problems).
3. Assume WCET and a scheduling policy over finite resources (exposes resources limitations).

Exposing Modeling Errors

Levels of analysis:

1. Assume zero execution time for actors (exposes modeling errors)
2. Assume known worst-case execution time (WCET) for actors, unbounded compute resources (exposes inadequate compute speed).
3. Assume WCET and a scheduling policy over finite resources (exposes resources limitations).

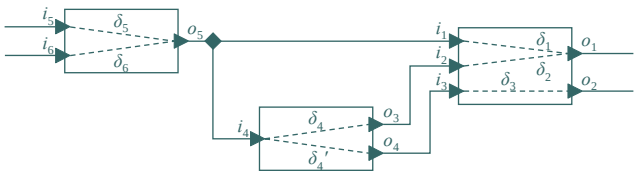
Do this using *causality interfaces*.

[1] Y. Zhou and E. A. Lee, "Causality Interfaces for Actor Networks," ACM Transactions on Embedded Computing Systems (TECS), April 2008.

Causality Interfaces

$\delta : P \times P \rightarrow R^+ \cup \{\infty\}$ yields the minimum model-time delay between any two ports (a *causality interface*).

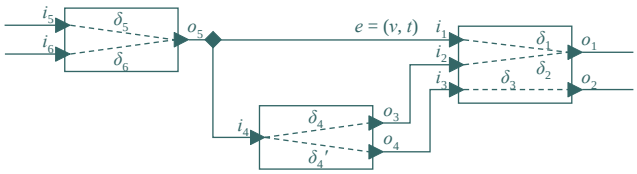
(P – set of ports; R^+ – set of non-negative real numbers)



Infer causality from causality interfaces using a min-plus algebra.

Example: $\delta(i_5, o_1) = \min\{\delta_5+\delta_1, \delta_5+\delta_4+\delta_2\}$, where $\delta_1, \ldots, \delta_6 \in R^+$ are pre-defined.

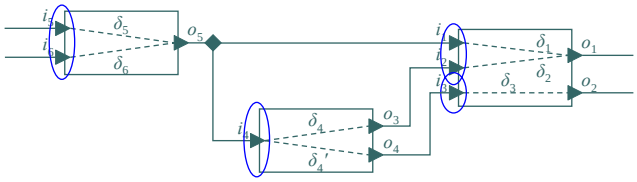
From Causality Interfaces to an Execution Strategy



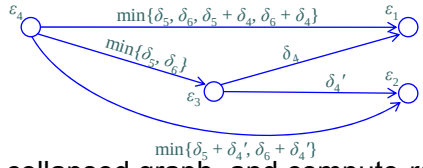
When is it safe to process $e = (v, t)$ at i_1?

1. future events at i_1, i_2 and i_3 have time stamps ≥ t (conventional), or
2. future events at i_1 and i_2 have time stamps ≥ t, or
3. future events at i_1 have time stamps ≥ t, and future events at i_2 depend on events at i_4 with time stamps ≥ $t - \delta_4$, or
4. future events at i_1 and i_2 depend on events at i_5 and i_6 with time stamps ≥ $t - \min\{\delta_5, \delta_6, \delta_5 + \delta_4, \delta_6 + \delta_4\}$.

Relevant Dependency [Ye Zhou]



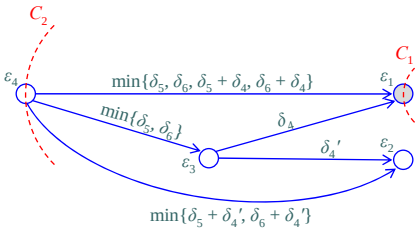
$i \sim i'$ iff they are input of the same actor and affect a common output. An *equivalence class* is a transitive closure of ~.



Construct a collapsed graph, and compute *relevant dependency* between equivalence classes.

$$d(\varepsilon', \varepsilon) = \min_{i' \in \varepsilon',\, i \in \varepsilon} \{\delta(i', i)\}$$

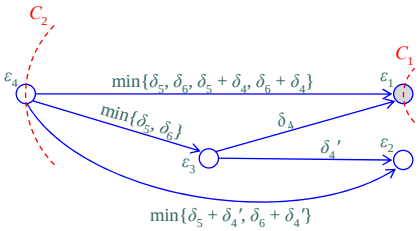
Dependency Cut [T. Feng, Y. Zhou, J. Zou]



A *dependency cut for ε* is a minimal but complete set of equivalence classes that needs to be considered to process an event at *ε*.

Example: C_1 and C_2 are both dependency cuts for ε_1.

Choosing a Dependency Cut

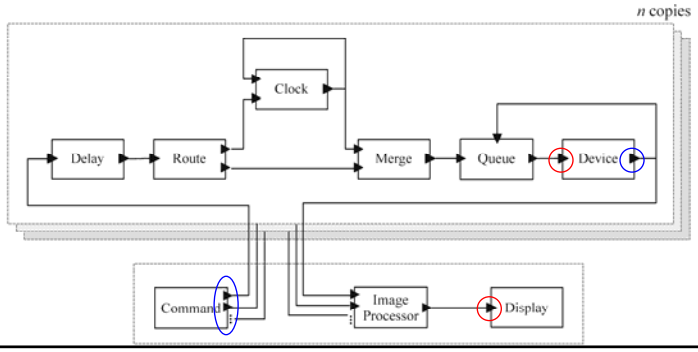


Determine earliest event $e = (v, t)$ at ε_1 safe to process

- If we choose C_1: all unprocessed events at ε_1 will have time stamps ≥ t.
- If we choose C_2: for any $\varepsilon \in C_2$, all unprocessed events at in ε_1 depend on events at ε with time stamps ≥ $t - d(\varepsilon, \varepsilon_1)$.
- We can freely choose a dependency cut.

Reference Application: Distributed Cameras

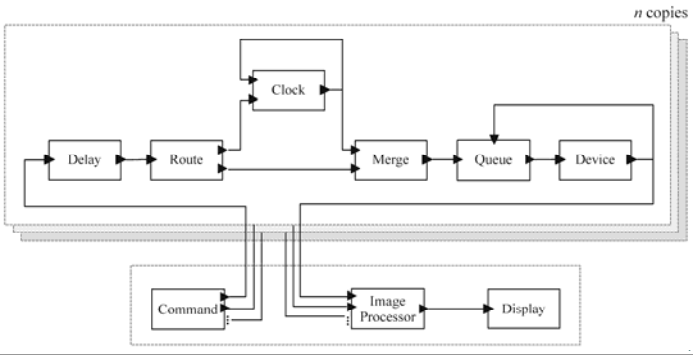
- *n* cameras located around a football field, all connected to a central computer.
- Events at blue ports satisfy $t \leq \tau$
  (t – time stamp of any event; τ – real time)
- Events at red ports satisfy $t \geq \tau$



Berkeley 41

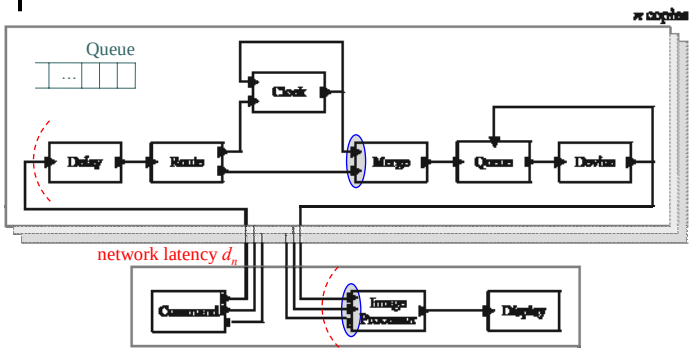
Problems to solve

- Make event-processing decisions locally
- Guarantee timely command delivery to the Devices
- Guarantee real-time update at the Display
- Tolerate images loss or corruption at Image Processor



Berkeley 42

Choose Dependency Cuts at Platform Boundaries



- n + 1 platforms with synchronized clocks (IEEE 1588).
- Choose dependency cuts at platform boundary.
- A queue stores events local to the platform.
- At real time τ, future events have time stamps ≥ $\tau - d_n$.

Time-sensitive computation is significantly different from other computation

Some misleading statements:

- *"Computing takes time"*
- *"Time is a resource"*
- *"Time is a non-functional property"*
- *"Real time is a quality of service problem"*