# Cyber-Physical Systems

## A Fundamental Intellectual Challenge


Edward A. Lee

*Robert S. Pepper Distinguished Professor*
*UC Berkeley*

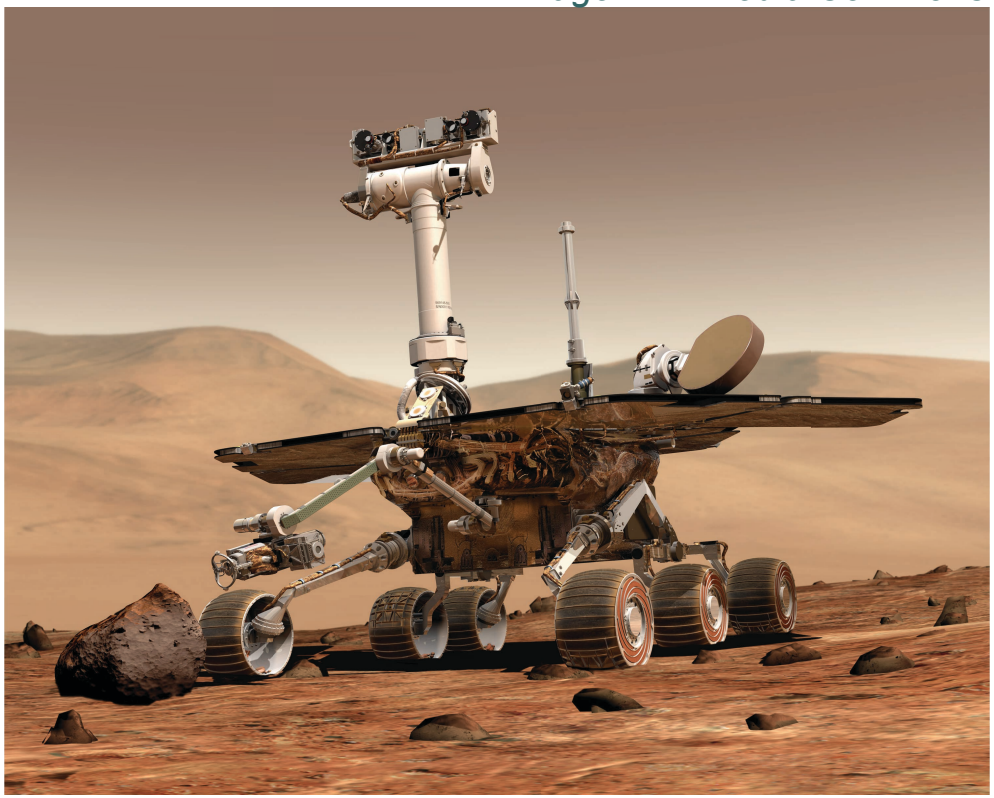

Invited Talk
College de France

December 11, 2013.
Paris, France

# Cyber-Physical Systems

*Orchestrating networked computational resources and physical systems.*

*Image: Wikimedia Commons*



*Roots:*

- Coined around 2006 by Helen Gill at the National Science Foundation in the US
- ***Cyberspace**: attributed William Gibson, who used the term in the novel Neuromancer.*
- ***Cybernetics**: coined by Norbert Wiener in 1948, to mean the* conjunction of control and communication.

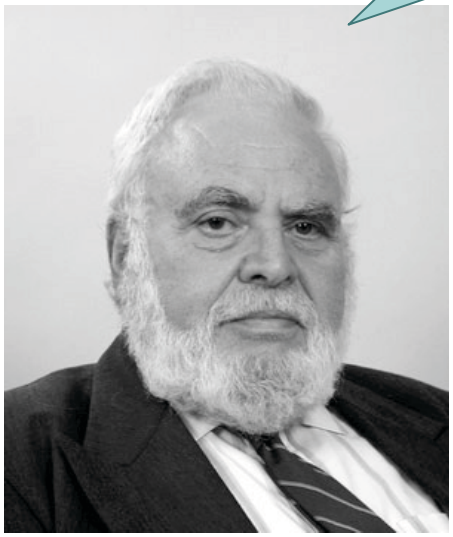
# Outline

1. Engineering Models for CPS
2. Time
3. Some Promising Approaches

# Models vs. Reality

*Solomon Golomb: Mathematical models – Uses and limitations. Aeronautical Journal 1968*

*You will never strike oil by drilling through the map!*



*Solomon Wolf Golomb (1932) mathematician and engineer and a professor of electrical engineering at the University of Southern California. Best known to the general public and fans of mathematical games as the inventor of polyominoes, the inspiration for the computer game Tetris. He has specialized in problems of combinatorial analysis, number theory, coding theory and communications.*

*But this does not, in any way, diminish the value of a map!*

# The Kopetz Principle

*Prof. Dr. Hermann Kopetz*

Many (predictive) properties that we assert about systems (determinism, timeliness, reliability, safety) are in fact not properties of an *implemented* system, but rather properties of a *model* of the system.

We can make definitive statements about *models*, from which we can *infer* properties of system realizations. The validity of this inference depends on *model fidelity*, which is always approximate.

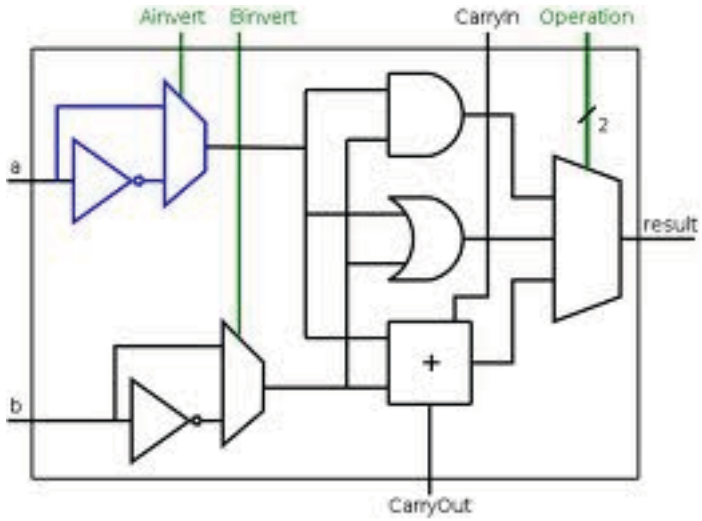
(paraphrased)

# Determinate Models

Physical System


*Image: Wikimedia Commons*

*Model*




*Synchronous digital logic*

# Determinate Models

## Physical System


*Image: Wikimedia Commons*

## *Model*

```
/** Reset the output receivers, which are the inside receivers of
 *  the output ports of the container.
 *  @exception IllegalActionException If getting the receivers fails.
 */
private void _resetOutputReceivers() throws IllegalActionException {
    List<IOPort> outputs = ((Actor) getContainer()).outputPortList();
    for (IOPort output : outputs) {
        if (_debugging) {
            _debug("Resetting inside receivers of output port: "
                    + output.getName());
        }
        Receiver[][] receivers = output.getInsideReceivers();
        if (receivers != null) {
            for (int i = 0; i < receivers.length; i++) {
                if (receivers[i] != null) {
                    for (int j = 0; j < receivers[i].length; j++) {
                        if (receivers[i][j] instanceof FSMReceiver) {
                            receivers[i][j].reset();
                        }
                    }
                }
            }
        }
    }
}
```

*Single-threaded imperative programs*

# Determinate Models

## Physical System


*Image: Wikimedia Commons*

## *Model*

```
module Timer:
input R, SEC;
output L, S;
Loop
  weak abort
    await 3 SEC;
    [
      sustain S
    ||
      await 5 SEC;
      sustain L
    ]
   when R;
  end
end module
```

*[S. Edwards, Columbia U.]*

*Synchronous language programs*

# Determinate Models

Physical System

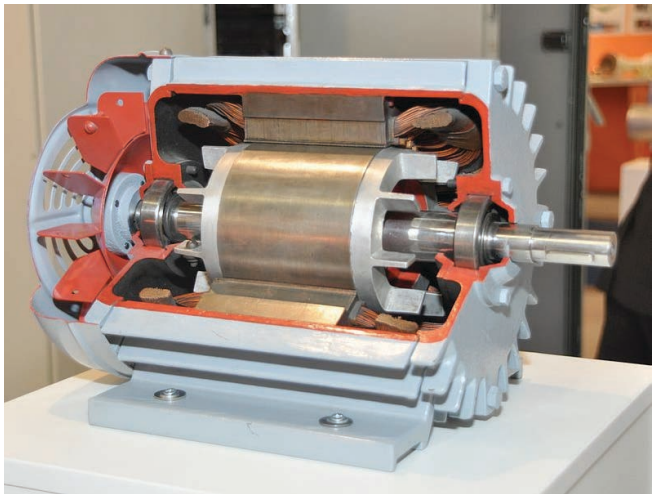


*Image: Wikimedia Commons*

*Model*


$$\dot{\mathbf{x}}(t) = \dot{\mathbf{x}}(0) + \frac{1}{M}\int_0^t \mathbf{F}(\tau)d\tau$$

*Differential Equations*

# A Major Problem for CPS: Combinations are Nondeterminate

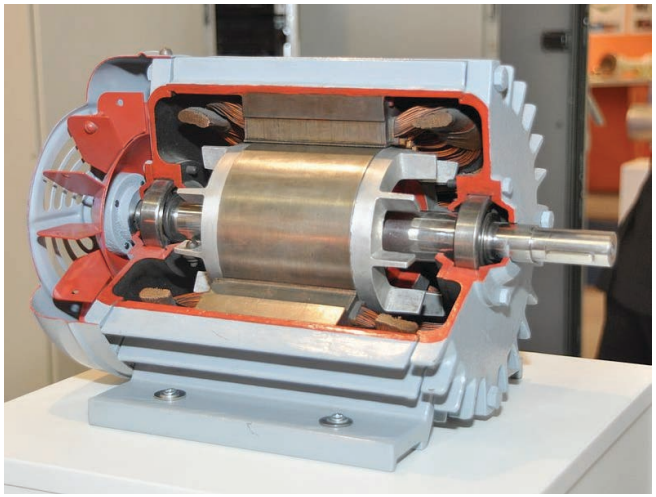
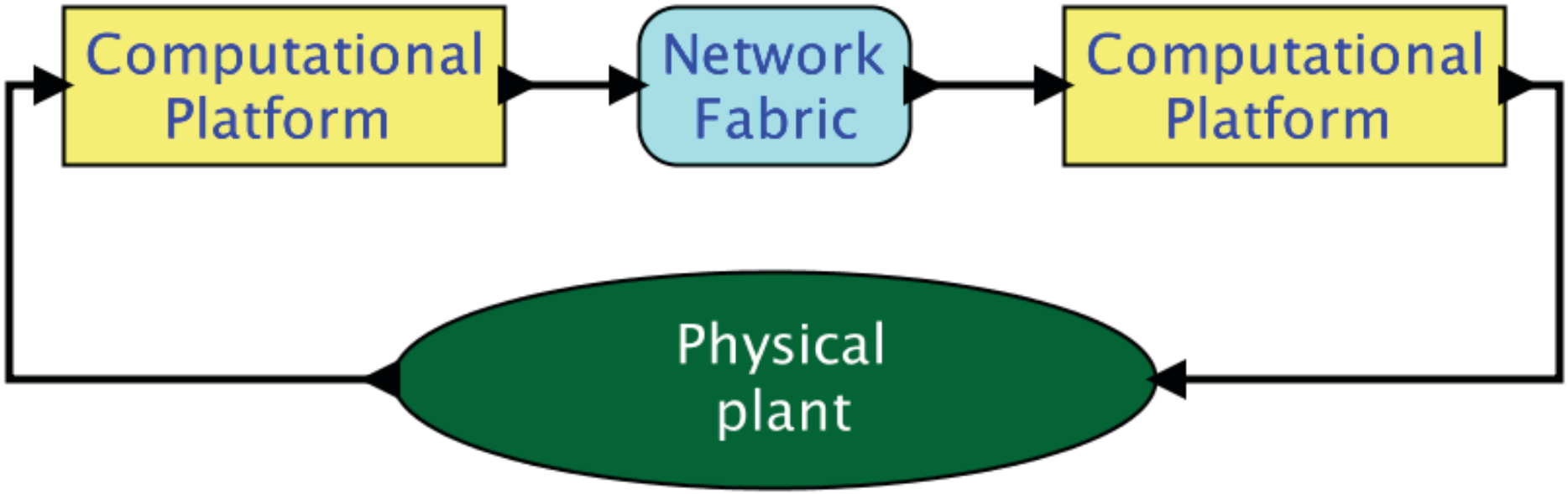



*Image: Wikimedia Commons*

```
/** Reset the output receivers, which are the inside receivers of
 *  the output ports of the container.
 *  @exception IllegalActionException If getting the receivers fails.
 */
private void _resetOutputReceivers() throws IllegalActionException {
    List<IOPort> outputs = ((Actor) getContainer()).outputPortList();
    for (IOPort output : outputs) {
        if (_debugging) {
            _debug("Resetting inside receivers of output port: "
                    + output.getName());
        }
        Receiver[][] receivers = output.getInsideReceivers();
        if (receivers != null) {
            for (int i = 0; i < receivers.length; i++) {
                if (receivers[i] != null) {
                    for (int j = 0; j < receivers[i].length; j++) {
                        if (receivers[i][j] instanceof FSMReceiver) {
                            receivers[i][j].reset();
                        }
                    }
                }
            }
        }
    }
}
```


$$\dot{\mathbf{x}}(t) = \dot{\mathbf{x}}(0) + \frac{1}{M}\int_0^t \mathbf{F}(\tau)d\tau$$

Schematic of a simple CPS:



Computation given in an untimed, imperative language. Physical plant modeled with ODEs or DAEs

```
void initTimer(void) {
    SysTickPeriodSet(SysCtlClockGet() / 1000);
    SysTickEnable();
    SysTickIntEnable();
}
volatile uint timer_count = 0;
void ISR(void) {
    if(timer_count != 0) {
        timer_count--;
    }
}
int main(void) {
    SysTickIntRegister(&ISR);
    .. // other init
    timer_count = 2000;
    initTimer();
    while(timer_count != 0) {
        ... code to run for 2 seconds
    }
    ... // other code
}
```

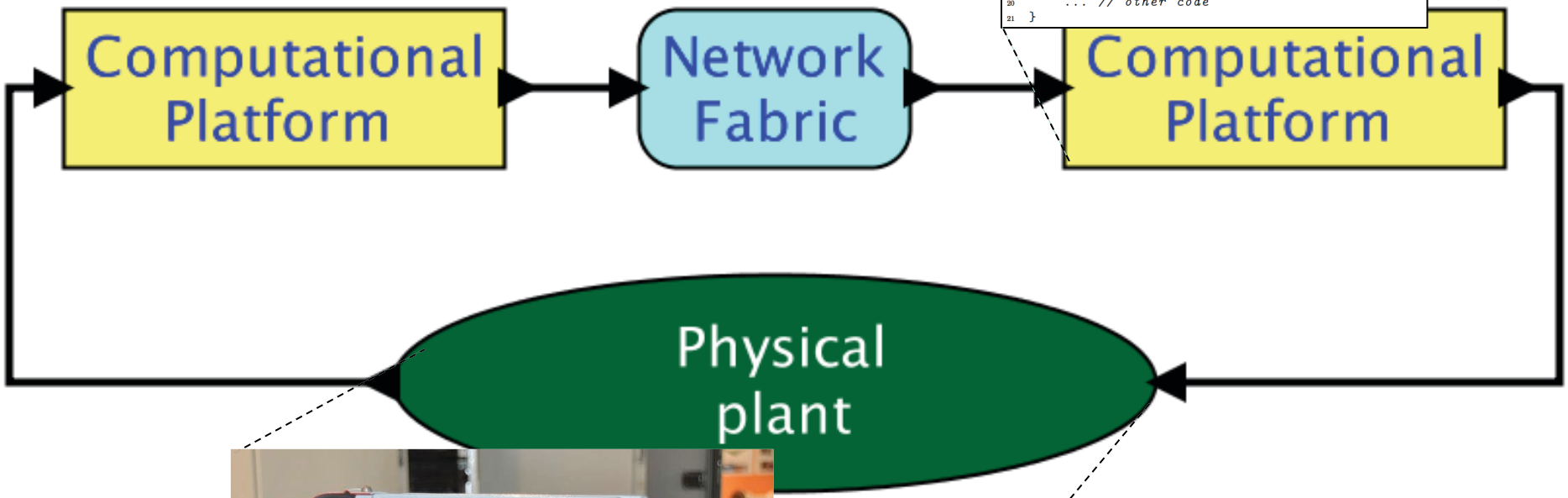
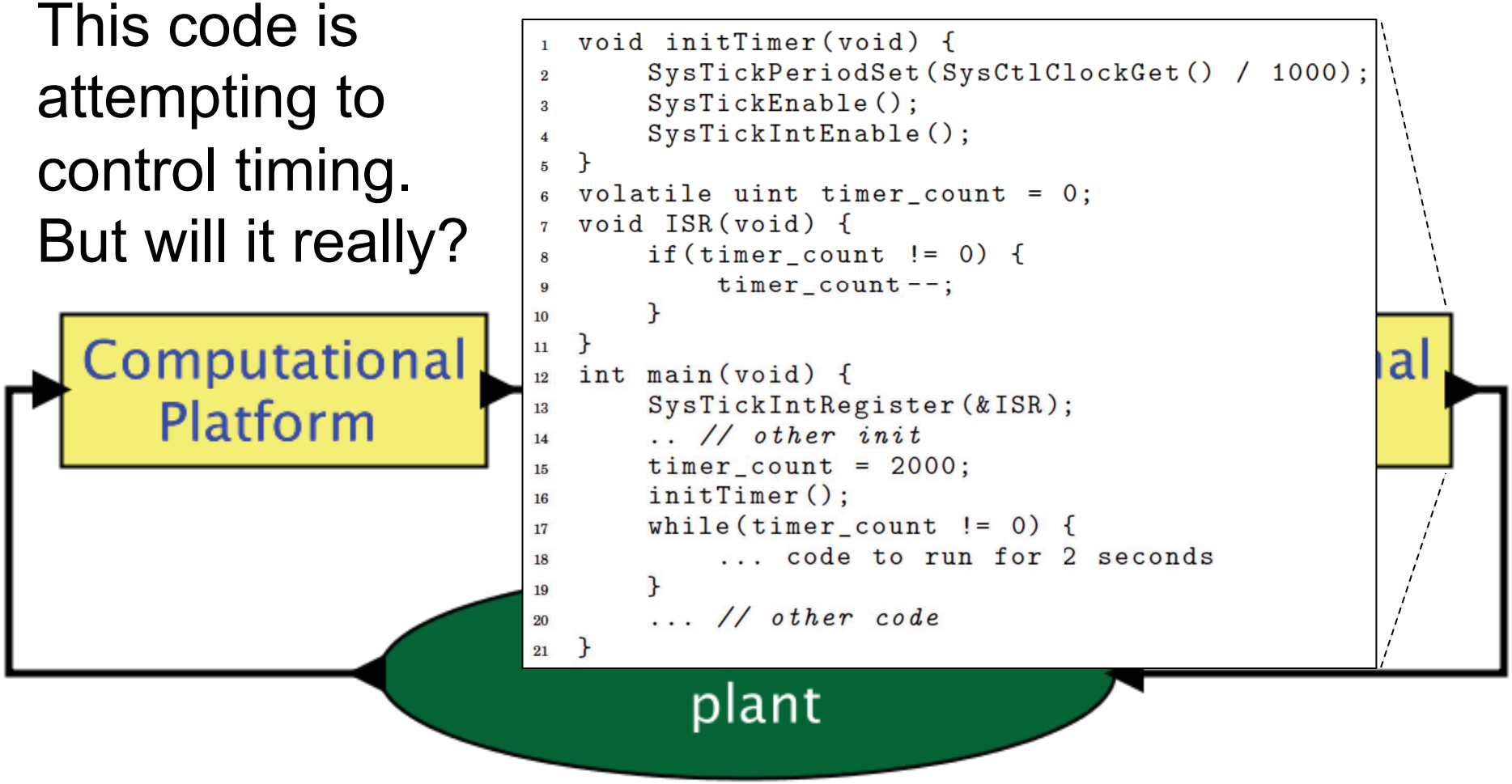





Image: Wikimedia Commons

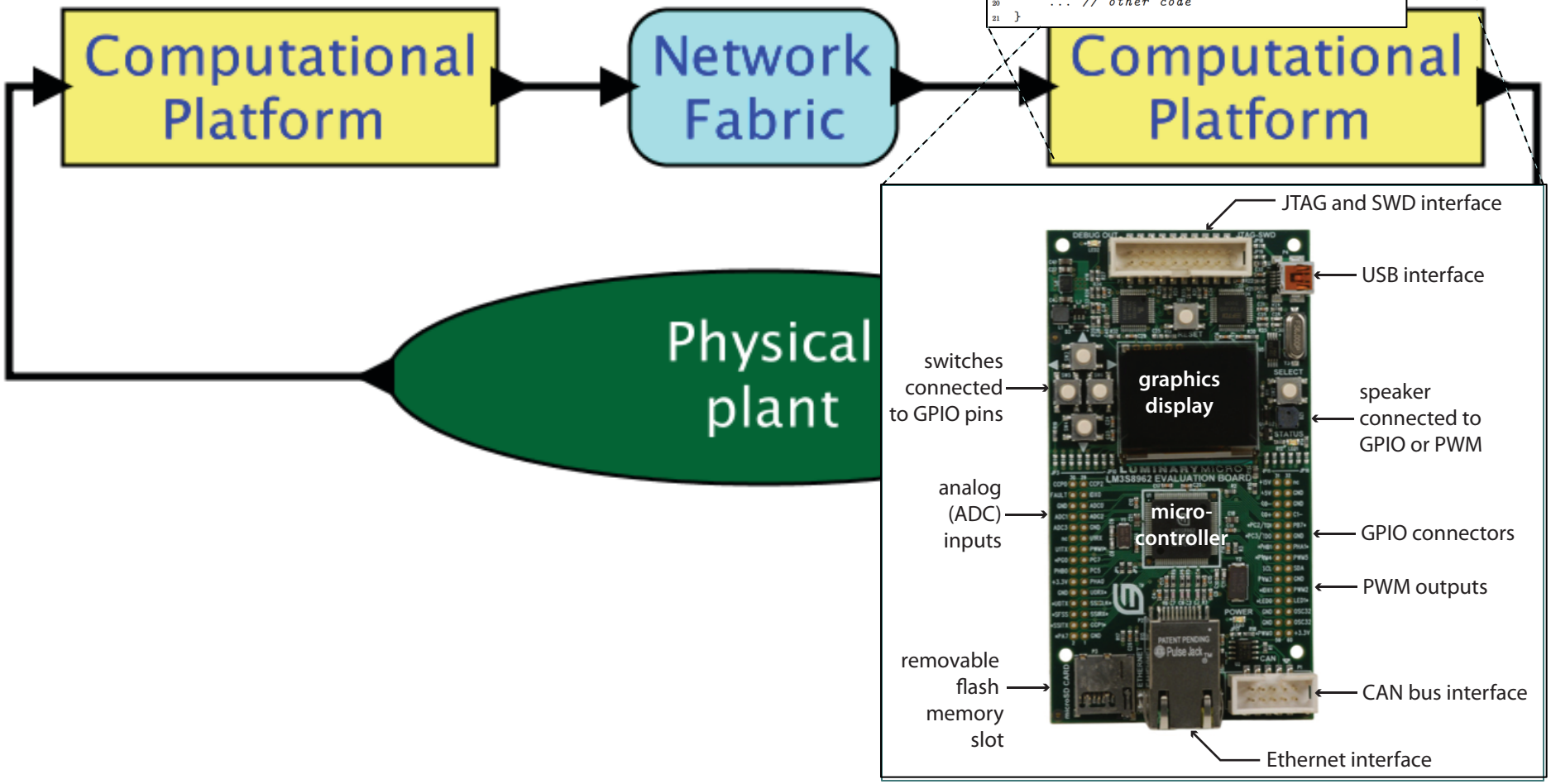
This code is attempting to control timing. But will it really?




```
1  void initTimer(void) {
2      SysTickPeriodSet(SysCtlClockGet() / 1000);
3      SysTickEnable();
4      SysTickIntEnable();
5  }
6  volatile uint timer_count = 0;
7  void ISR(void) {
8      if(timer_count != 0) {
9          timer_count--;
10     }
11 }
12 int main(void) {
13     SysTickIntRegister(&ISR);
14     .. // other init
15     timer_count = 2000;
16     initTimer();
17     while(timer_count != 0) {
18         ... code to run for 2 seconds
19     }
20     ... // other code
21 }
```

# Timing behavior emerges from the combination of the program and the hardware platform.

```
void initTimer(void) {
    SysTickPeriodSet(SysCtlClockGet() / 1000);
    SysTickEnable();
    SysTickIntEnable();
}
volatile uint timer_count = 0;
void ISR(void) {
    if(timer_count != 0) {
        timer_count--;
    }
}
int main(void) {
    SysTickIntRegister(&ISR);
    .. // other init
    timer_count = 2000;
    initTimer();
    while(timer_count != 0) {
        ... code to run for 2 seconds
    }
    ... // other code
}
```




Stellaris LM3S8962 evaluation board (Luminary Micro 2008, now Texas Instruments)

# Consequences

When precise control over timing is needed, designs are brittle. Small changes in the hardware, software, or environment can cause big, unexpected changes in timing. Results:

- System behavior emerges only at system integration.
- Manufacturers stockpile parts to suffice for the complete production run of a product.
- Manufacturers cannot leverage improvements in the hardware (e.g. weight, power).
- Any change forces re-testing and re-certifying.
- Designs are over provisioned, increasing cost, weight, and energy usage.

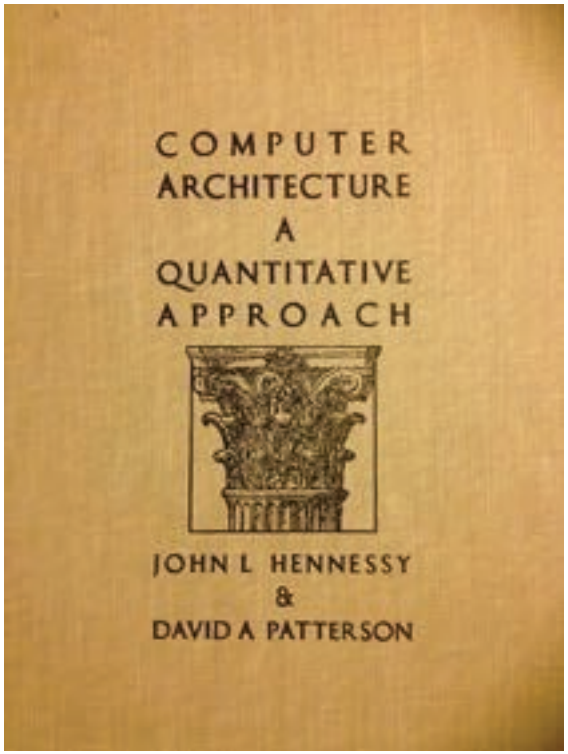
# A Key Challenge:
# Timing is not Part of Software Semantics

***Correct*** *execution of a program in C, C#, Java, Haskell, OCaml, Esterel, etc. has nothing to do with how long it takes to do anything. Nearly all our computation and networking abstractions are built on this premise.*

Programmers have to step *outside* the programming abstractions to specify timing behavior.

Programmers have no map!

# Computer Science has not *ignored* timing…




*The first edition of Hennessy and Patterson (1990) revolutionized the field of computer architecture by making performance metrics the dominant criterion for design.*

*Today, for computers, timing is merely a performance metric.*

*It needs to be a correctness criterion.*

# Correctness criteria

We can safely assert that line 8 does not execute

```
1  void foo(int32_t x) {
2       if (x > 1000) {
3            x = 1000;
4       }
5       if (x > 0) {
6            x = x + 1000;
7            if (x < 0) {
8                 panic();
9            }
10      }
11 }
```

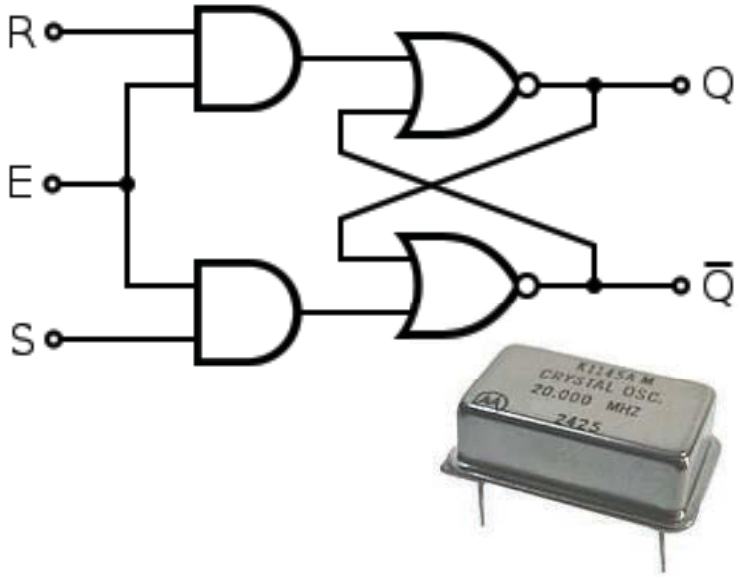
(In C, we need to separately ensure that no other thread or ISR can overwrite the stack, but in more modern languages, such assurance is provided by construction.)

*We can develop **absolute confidence** in the software, in that only a **hardware failure** is an excuse.*

*But not with regards to timing!!*

# The hardware out of which we build computers is capable of delivering “correct” computations and precise timing…

The synchronous digital logic abstraction removes the messiness of transistors.




*… but the overlaying software abstractions discard the timing precision.*

```
// Perform the convolution.
for (int i=0; i<10; i++) {
  x[i] = a[i]*b[j-i];
  // Notify listeners.
  notify(x[i]);
}
```

# Challenge # 1

Can we change programming models so that a *correct* execution of a program always delivers the same temporal behavior (with high precision) at the subsystem I/O?

*i.e. we need determinate CPS models with high fidelity implementations*

# Challenge # 2

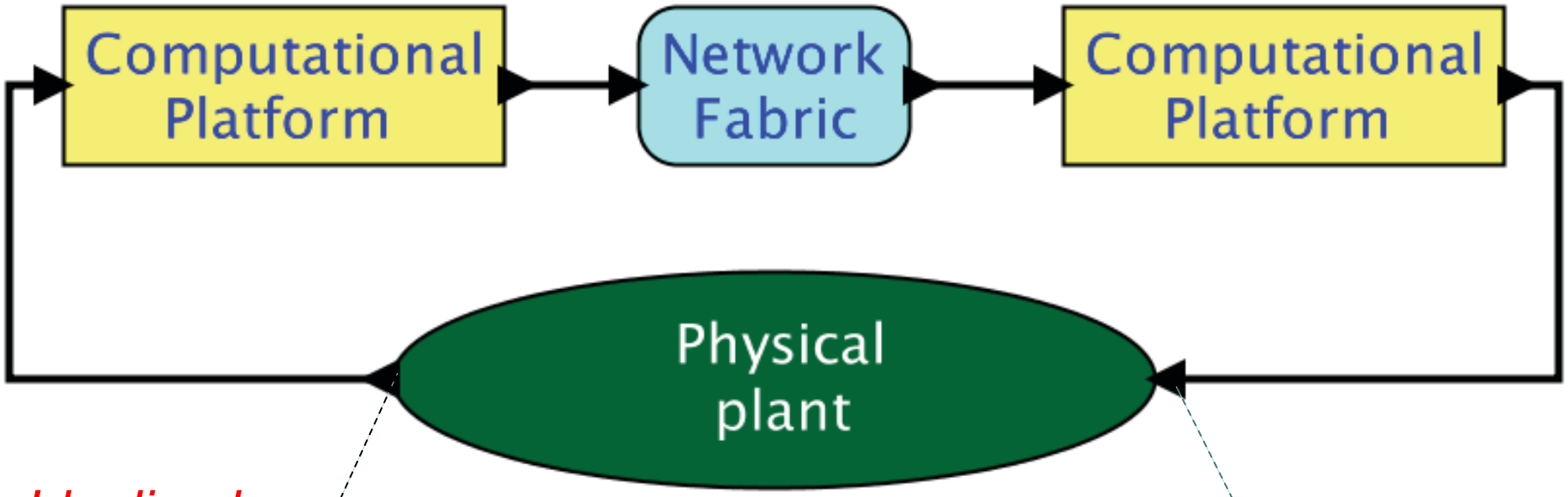
How can we overcome the powerful inertia created by existing languages, tools, and methodologies to allow innovation that may change key abstractions?

*i.e. we need open minds*

# Outline

1. Engineering Models for CPS
2. Time
3. Some Promising Approaches

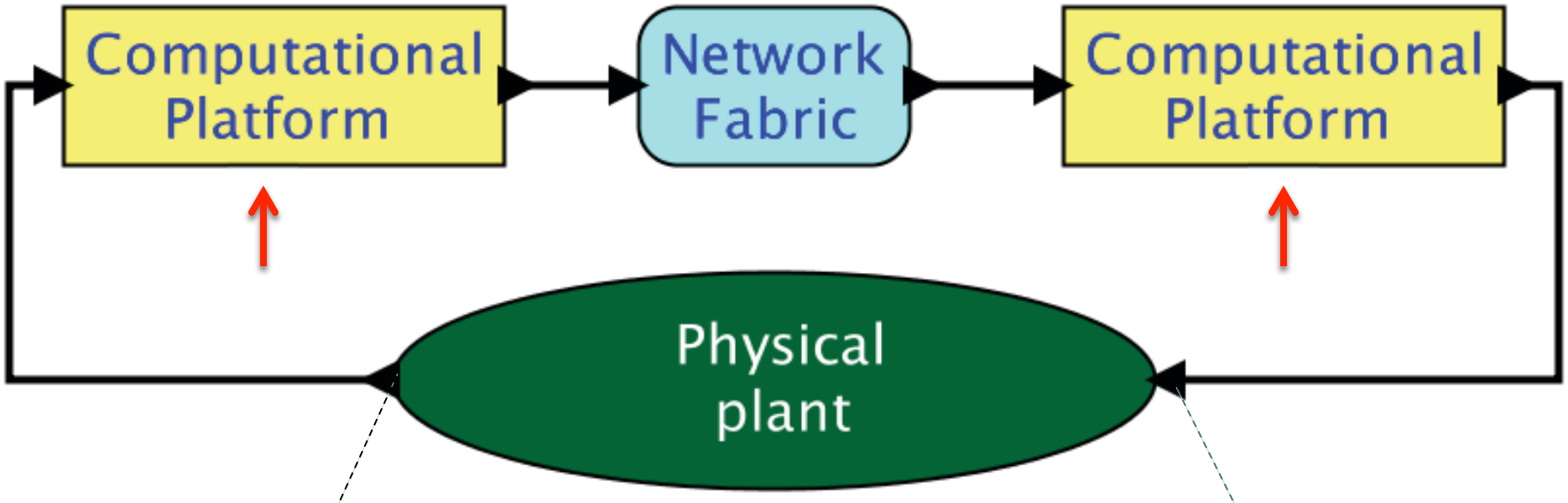
For CPS the very notion of *time* is subtle.




*Idealized Newtonian notion of time.*

$$\dot{\theta}(t) = \dot{\theta}(0) + \frac{1}{I}\int_0^t \mathbf{T}(\tau)d\tau$$

Computational platforms have no access to *t.*
Instead, local measurements of time are used.




$$\dot{\theta}(t) = \dot{\theta}(0) + \frac{1}{I}\int_0^t \mathbf{T}(\tau)d\tau$$

# There are naïve answers out there

- Uniform, global Newtonian time:

$$\dot{\theta}(t) = \dot{\theta}(0) + \frac{1}{I}\int_0^t \mathbf{T}(\tau)d\tau$$

- Floating point numbers:

```
double time;
```

# A Major Emerging Opportunity: Clock Synchronization

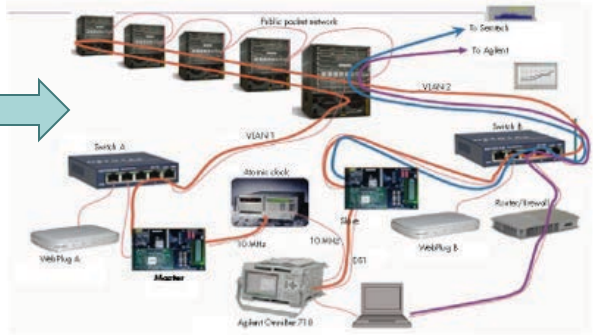
Clock synchronization is going to change the world (again)


*Gregorian Calendar (BBC history)*

1500s
days

*Musée d'Orsay clock (Wikimedia Commons)*

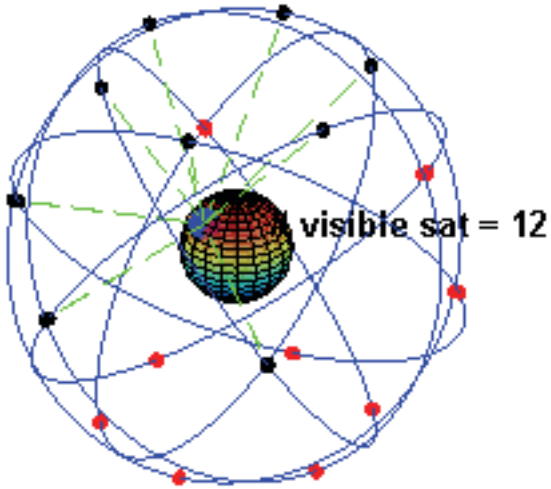
1800s
seconds



*2005: first IEEE 1588 plugfest*

2000s
nanoseconds

# Global Positioning System




Provides ~100ns accuracy to devices with outdoor access.

*Images: Wikimedia Commons*

# Precision Time Protocols (PTP) IEEE 1588 on Ethernet

*Press Release October 1, 2007*

National Semiconductor
*The Sight & Sound of Information*

NEWS RELEASE

**For More Information Contact**

**Media Contact**
Naomi Mitchell
**National Semiconductor**
(408) 721-2142
naomi.mitchell@nsc.com

**Reader Information**
Design Support Group
(800) 272-9959
www.national.com


**Industry's First Ethernet Transceiver with IEEE 1588 PTP Hardware Support from National Semiconductor Delivers Outstanding Clock Accuracy**

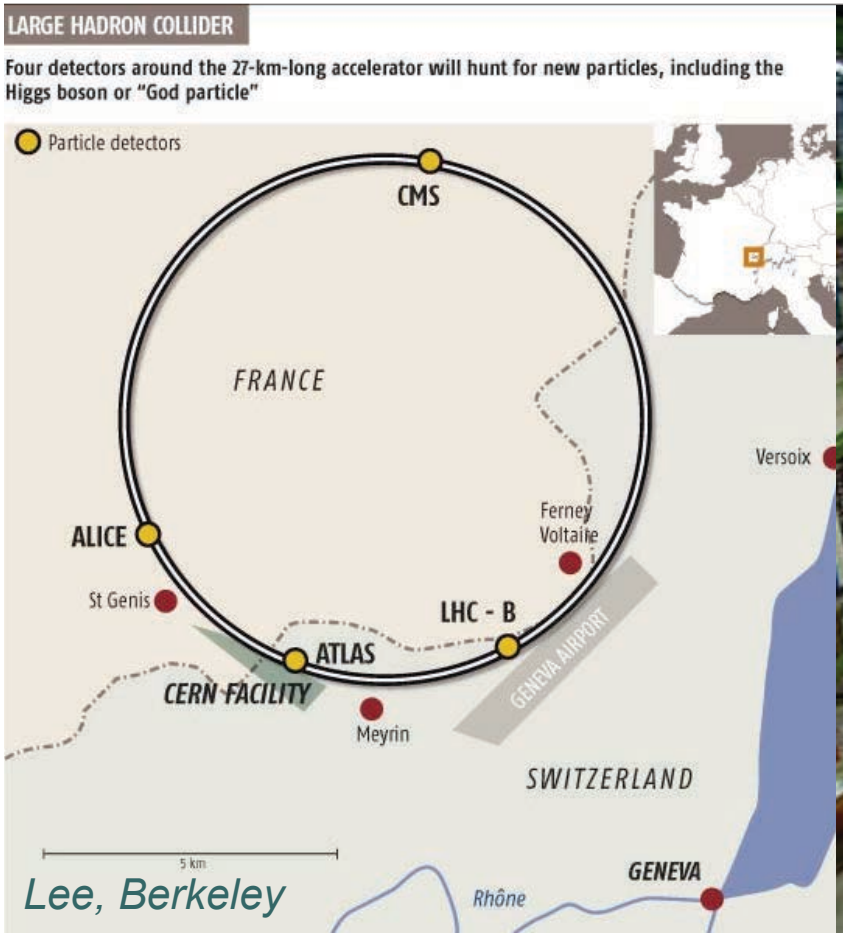
**Using DP83640, Designers May Choose Any Microcontroller, FPGA or ASIC to Achieve 8- Nanosecond Precision with Maximum System Flexibility**

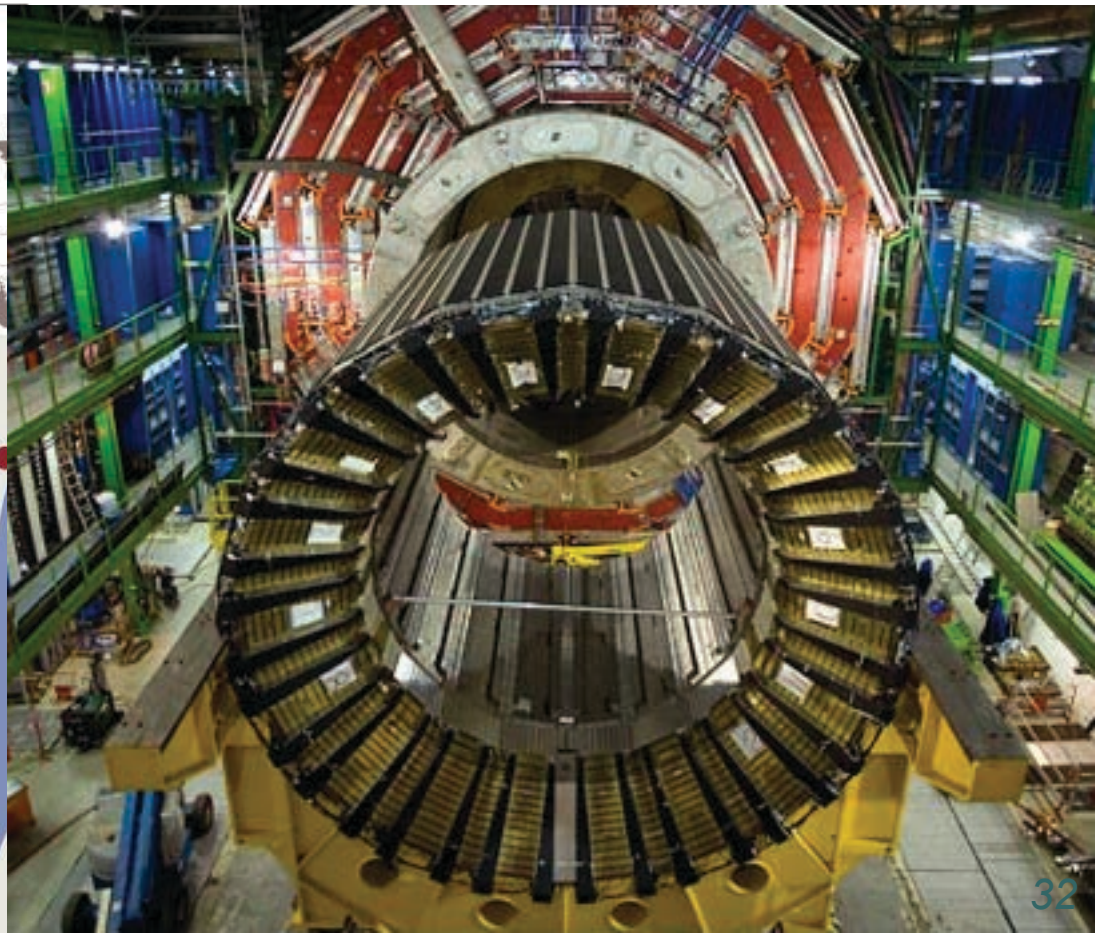
**It is becoming routine for physical network interfaces (PHY) to provide hardware support for PTPs.**

With this first generation PHY, clocks on a LAN agree on the current time of day to within 8ns, far more precise than GPS older techniques like NTP.

# An Extreme Example: The Large Hadron Collider

The WhiteRabbit project at CERN is synchronizing the clocks of computers 10 km apart to within about 80 psec using a combination of GPS, IEEE 1588 PTP and synchronous ethernet.



# Clock Synchronization Enables:

- Energy efficiency
- Coordination, even without communication
- Security
- Resource management
- Determinism

*… but I will skip this story in the interest of time…*

# Challenge # 3

Can we develop a model of time that is consistent with the realities of time measurement and clock synchronization and also with the engineering
models used for physical systems?

*i.e. we need a semantics of time*

# Outline

1. Engineering Models for CPS
2. Time
3. Some Promising Approaches

# Some Promising Approaches

- Superdense time
- PRET machines
- PTIDES for distributed real-time systems

# Superdense Time

*For heterogeneous mixtures of dynamics:*

- Continuously evolving state in time
  - Continuous-time systems

*Physical Dynamics*

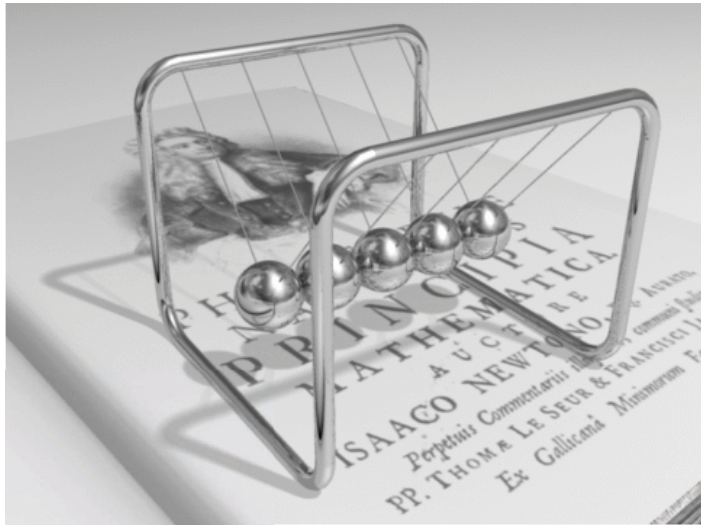
- Discretely evolving state in time
  - Discrete-time systems
  - Discrete-event systems
  - Synchronous systems

*Physical Events*
*Software Controllers*
*Signal Processing*

- Sequentially evolving state
  - Imperative programs

*Software*

Image by Dominique Toussaint
GNU Free Documentation License




# Consider Physical Events

## Momentum of the second ball:

Conventional: $p: \mathbb{R} \rightarrow \mathbb{R}$

$$p(t) = \begin{cases} P & \text{if } t = \tau \\ 0 & \text{otherwise} \end{cases}$$

# Flaws with the Conventional Model

1. Discretizing the momentum by sampling yields a signal that is indistinguishable from a continuous signal.

2. Momentum is not conserved. At the time of collision, all three middle balls have equal momentum summing to three times the momentum of the first ball before the collision.

$$p(t) = \begin{cases} P & \text{if } t = \tau \\ 0 & \text{otherwise} \end{cases}$$

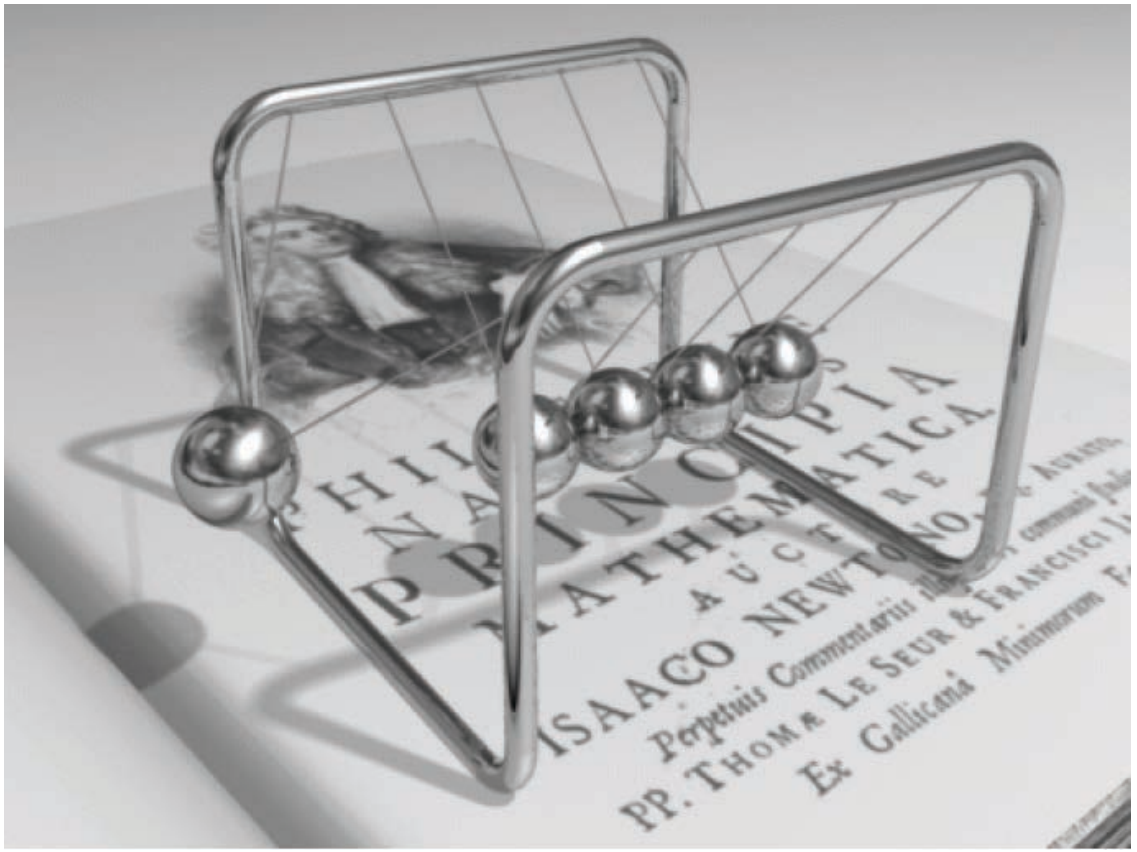



Image by Dominique Toussaint
GNU Free Documentation License

# Improvements with Superdense Time

1. Discretizing by sampling yields a signal that is semantically distinct from any continuous signal.

2. Momentum is conserved.

3. Signals can be piecewise continuous, enabling use of conventional ODE solvers between discontinuities.

$$p(t,m) = \begin{cases} P & \text{if } t = \tau, m = 1 \\ 0 & \text{otherwise} \end{cases}$$

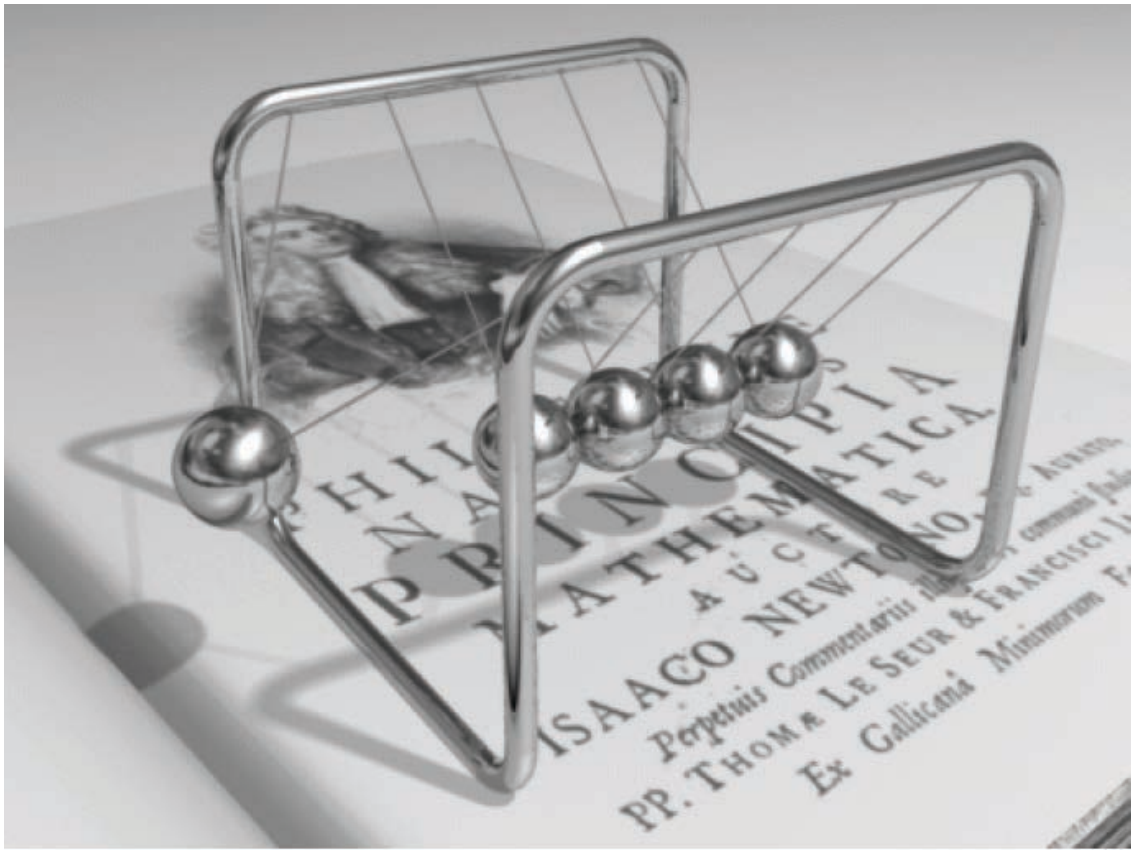
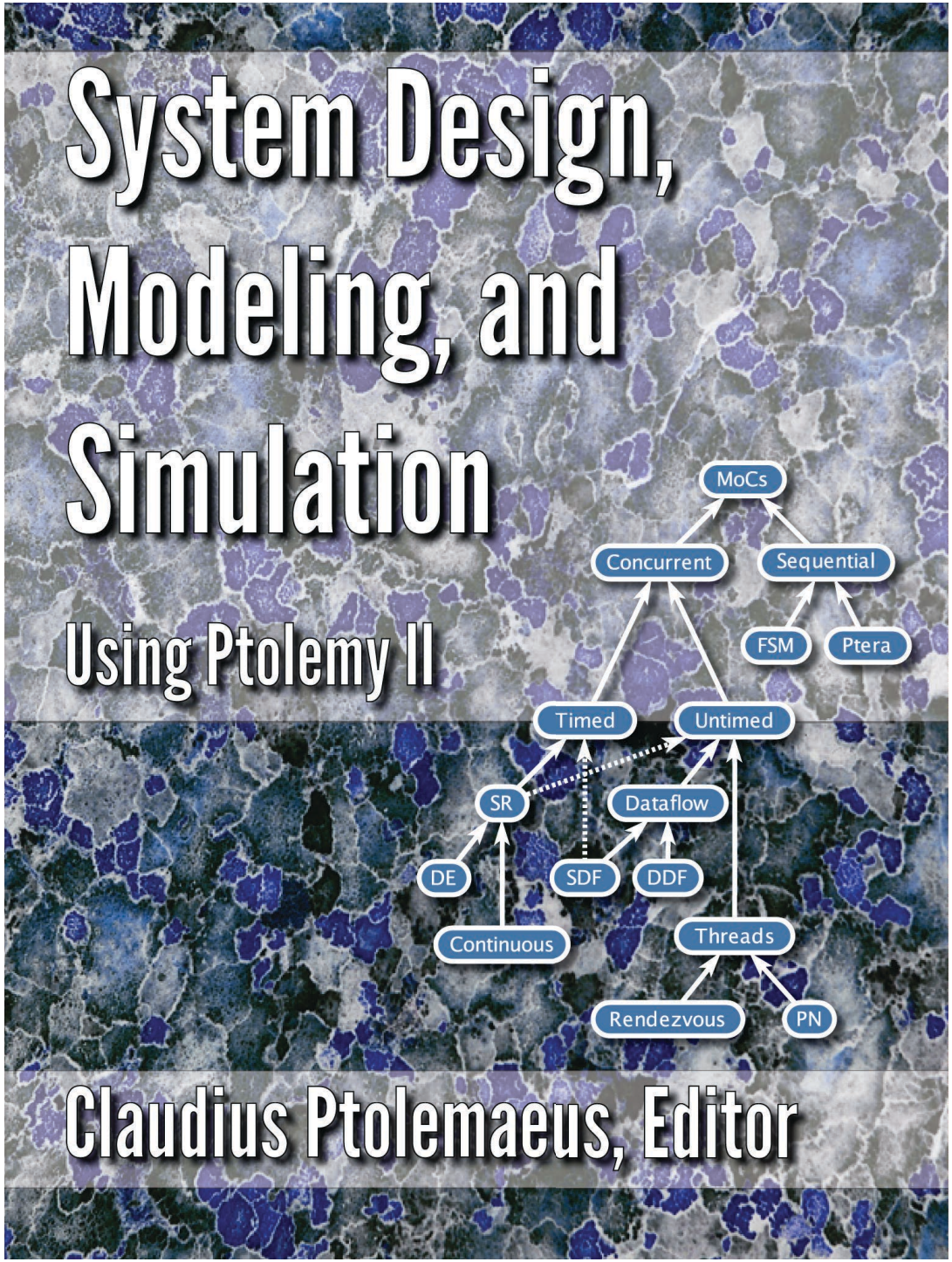



Image by Dominique Toussaint
GNU Free Documentation License

# Superdense Time

Provides a principled way to mix discrete events and untimed sequences (software) with continuous dynamics.

See the Ptolemy book, Chapter 1.
http://ptolemy.org/systems



# Some Promising Approaches

- Superdense time
- PRET machines
- PTIDES for distributed real-time systems

# PRET Machines

- **PRE**cision-**T**imed processors = **PRET**
- **P**redictable, **RE**peatable **T**iming = **PRET**
- **P**erformance *with* **RE**peatable **T**iming = **PRET**

```
// Perform the convolution.
for (int i=0; i<10; i++) {
  x[i] = a[i]*b[j-i];
  // Notify listeners.
  notify(x[i]);
}
```

*Computing*

**+**

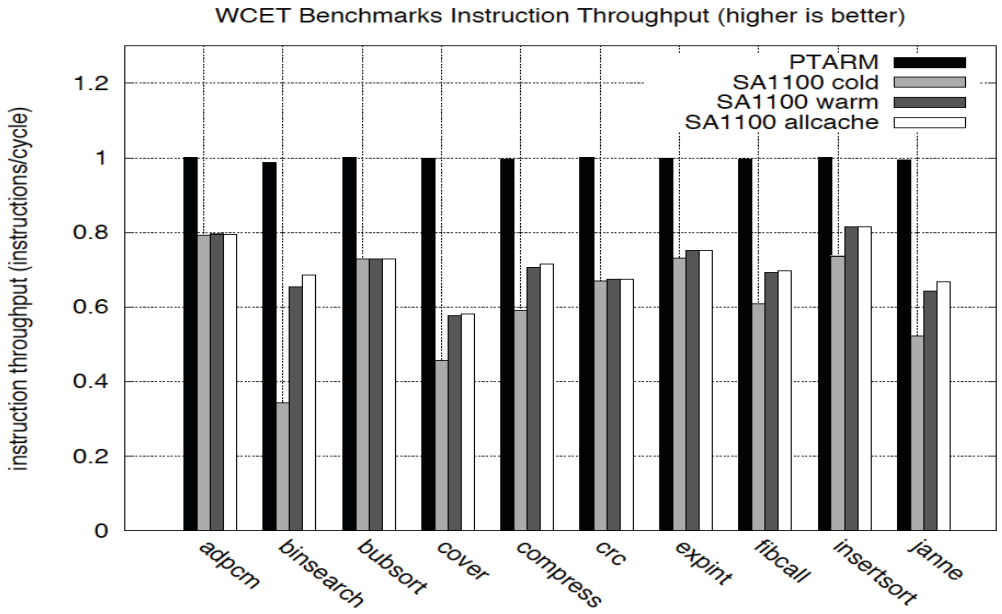
*With time*

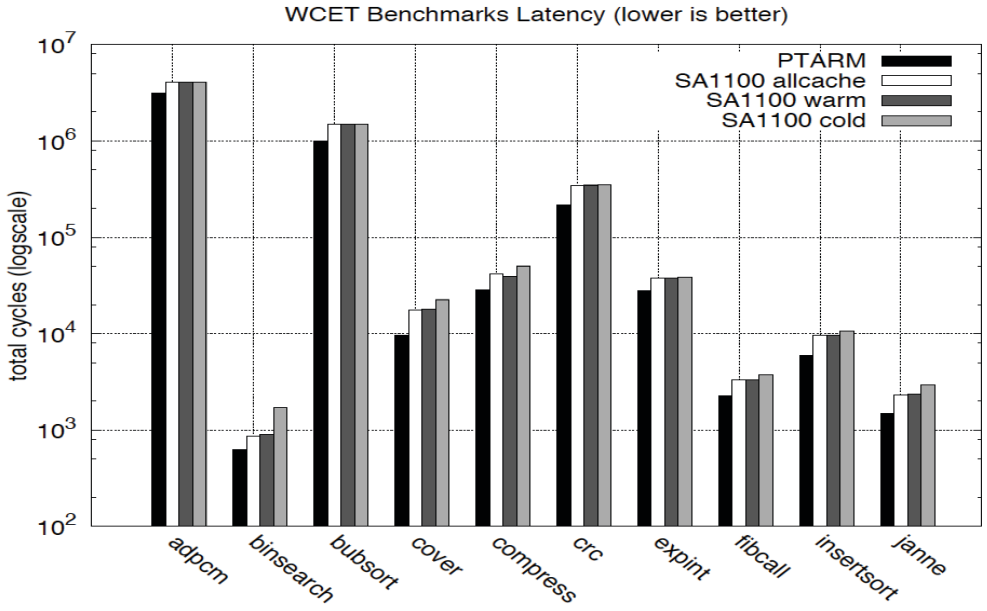
**= PRET**

# The Bottom Line

In microarchitecture design, we have shown that you do not need to sacrifice performance to get control over timing.

*… but I will skip this story in the interest of time…*



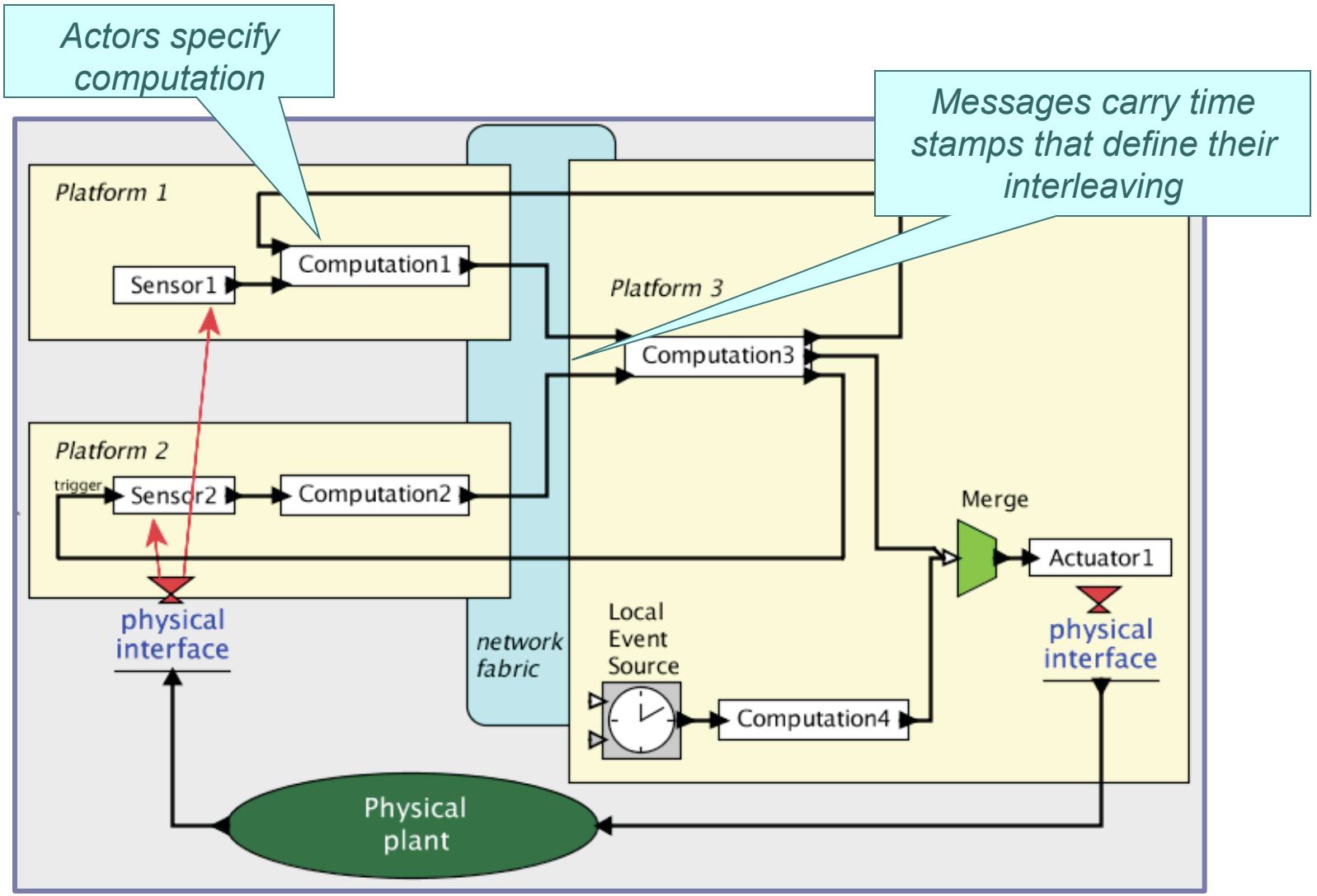

*[Isaac Liu, PhD Thesis, May, 2012]*

# Some Promising Approaches

- Superdense time
- PRET machines
- PTIDES for distributed real-time systems

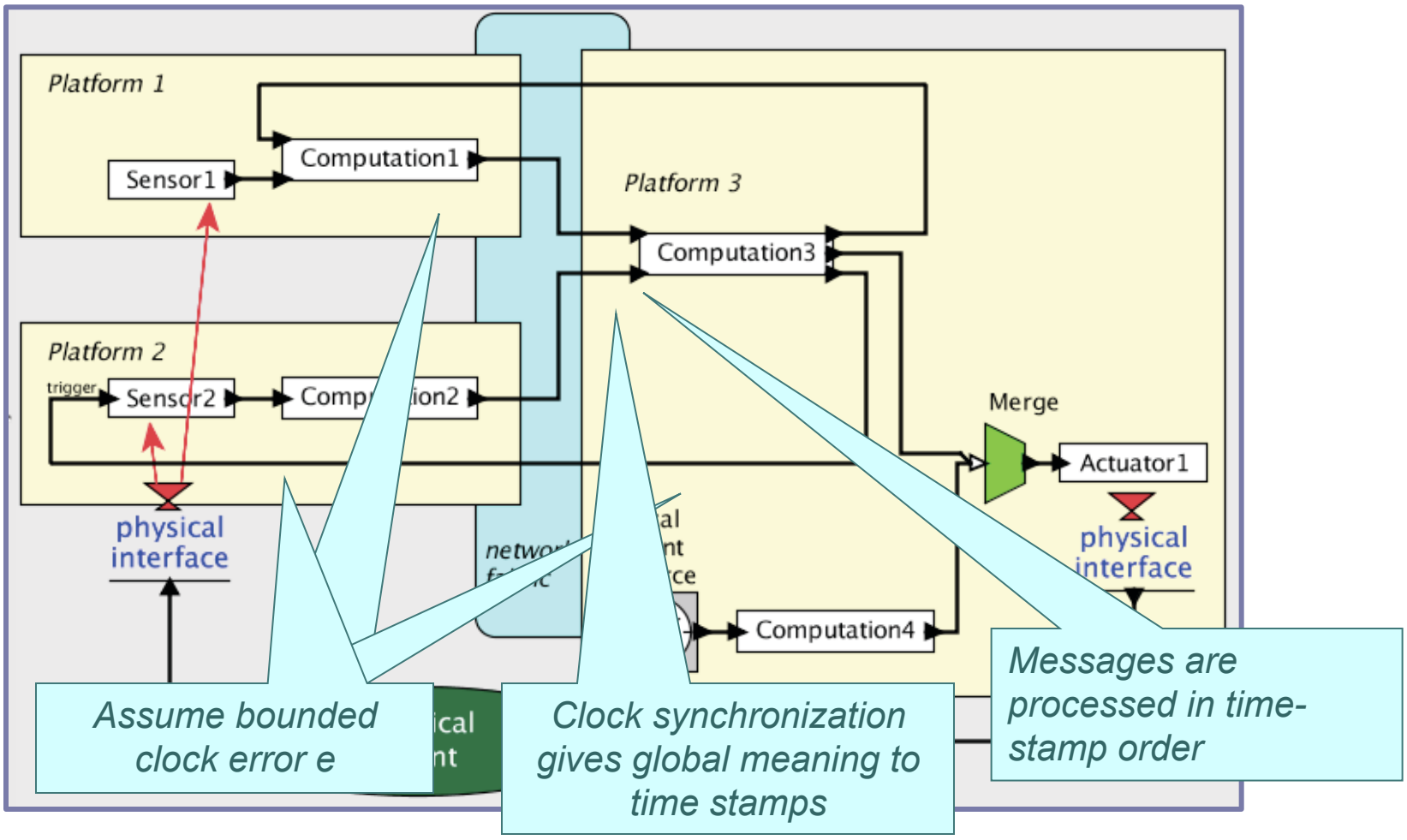
# Ptides: Programming Temporally Integrated Distributed Embedded Systems

## First step: Time-stamped messages.

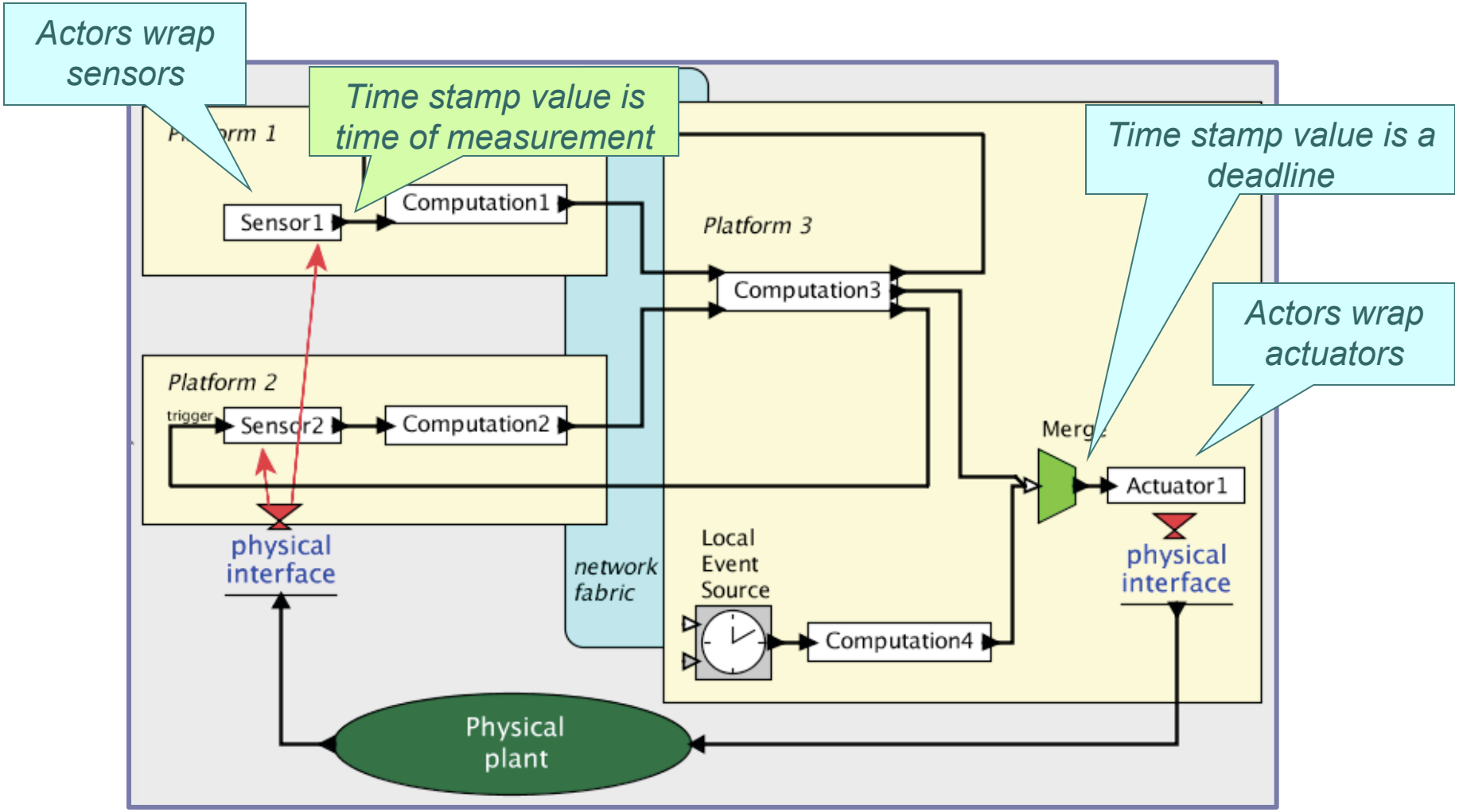


# Ptides: Second step: Network time synchronization

GPS, NTP, IEEE 1588, time-triggered busses, … they all work. We just need to bound the clock synchronization error.

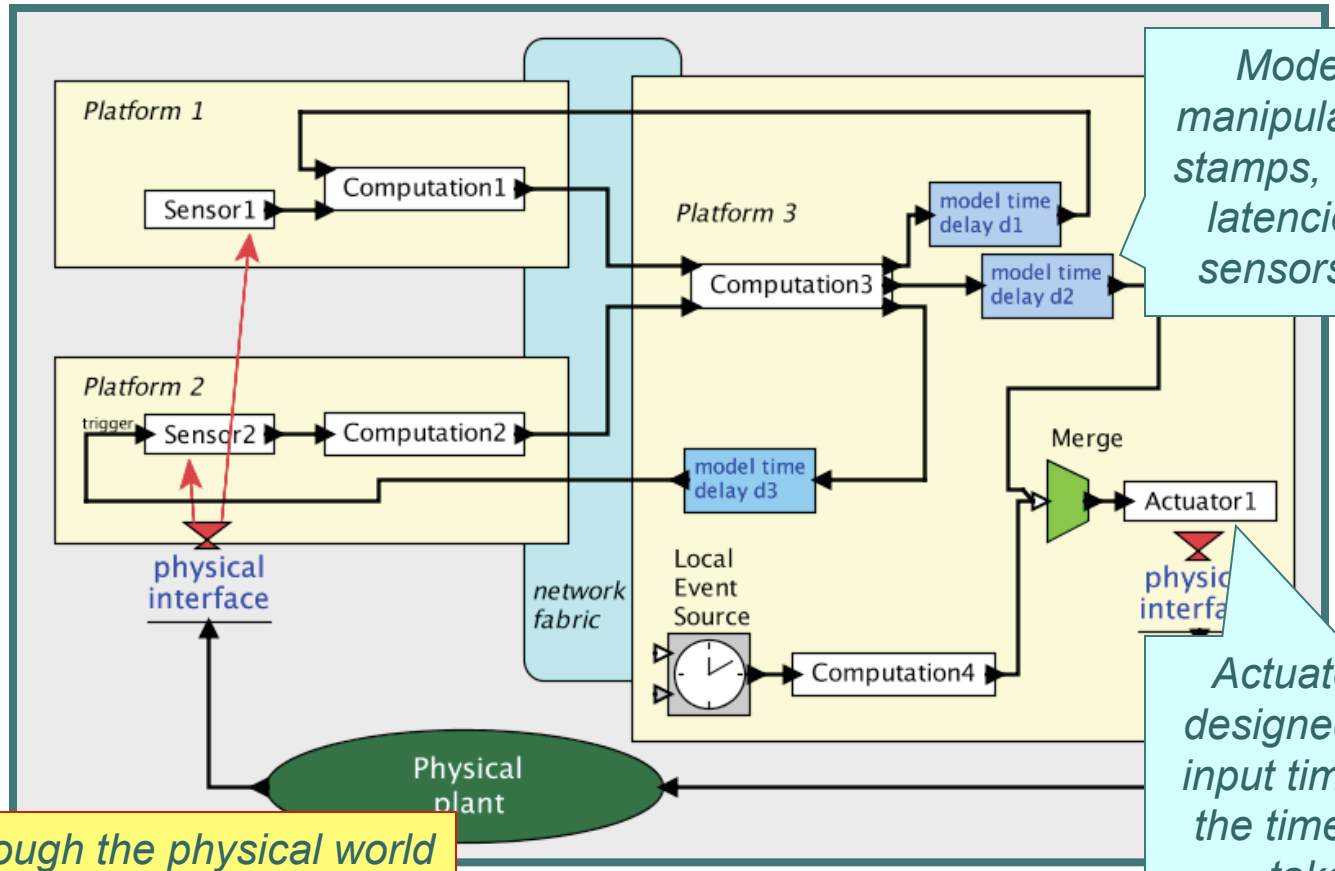


# Ptides: Third step:
# Bind time stamps to real time at sensors and actuators



# Ptides: Fourth step: Specify latencies in the model

*Global latencies between sensors and actuators become controllable, which enables analysis of system dynamics.*




*Model includes manipulations of time stamps, which control latencies between sensors and actors*

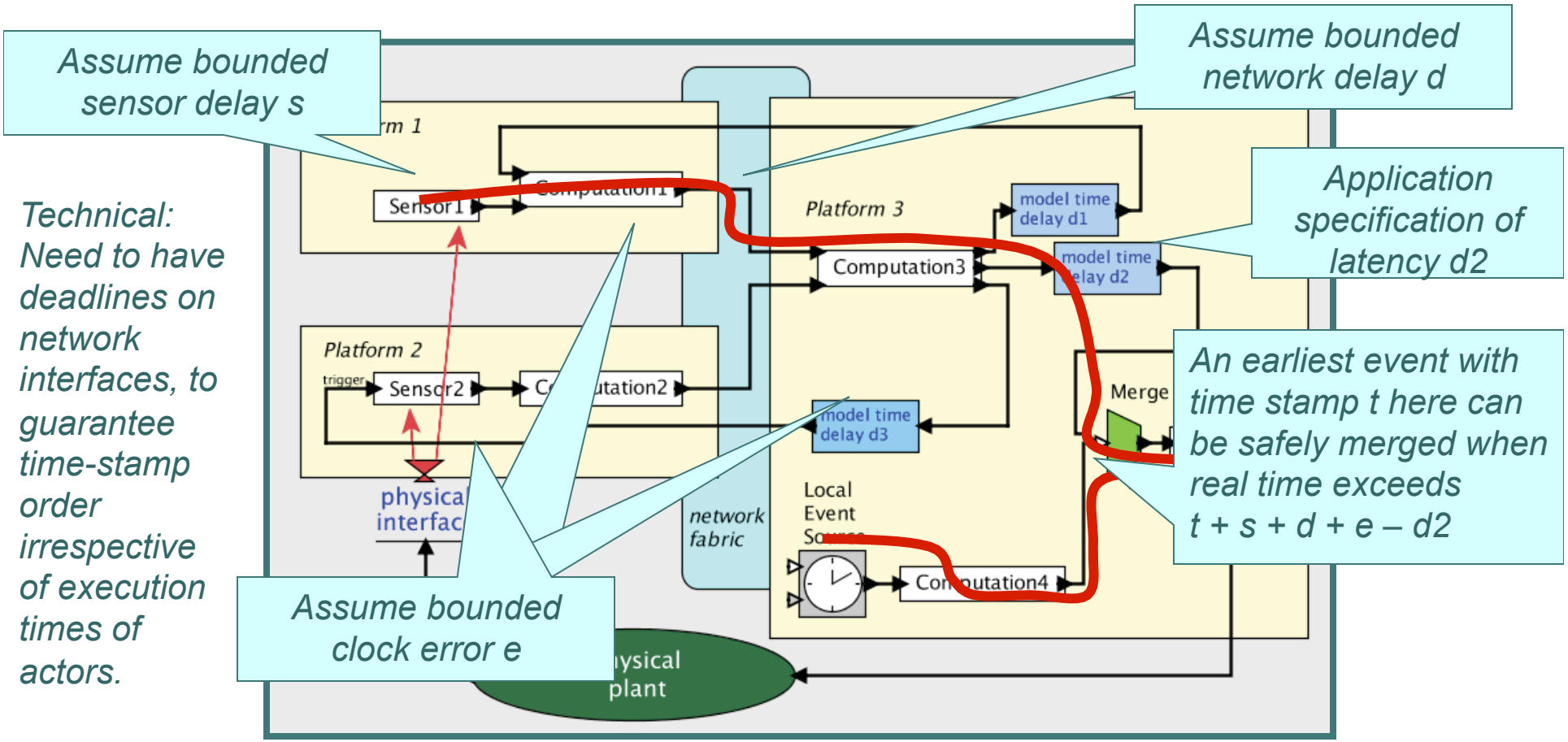
*Actuators may be designed to interpret input time stamps as the time at which to take action.*

*Feedback through the physical world*

# Ptides: Fifth step
# Safe-to-process analysis (ensures determinacy)

*Safe-to-process analysis guarantees that events are processed in time-stamp order, given some assumptions.*

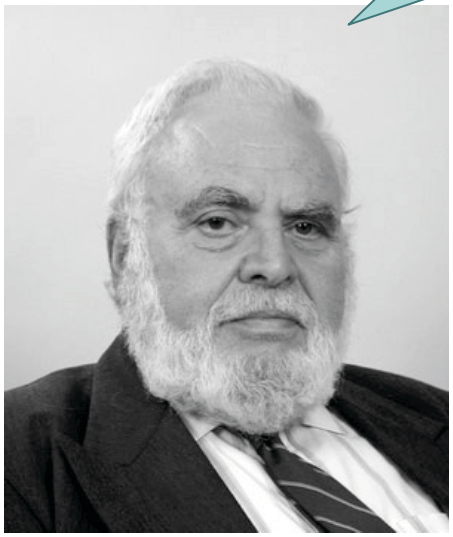
*Technical: Need to have deadlines on network interfaces, to guarantee time-stamp order irrespective of execution times of actors.*



# So Many Assumptions?

*Recall Solomon Wolf Golomb:*

*You will never strike oil by drilling through the map!*



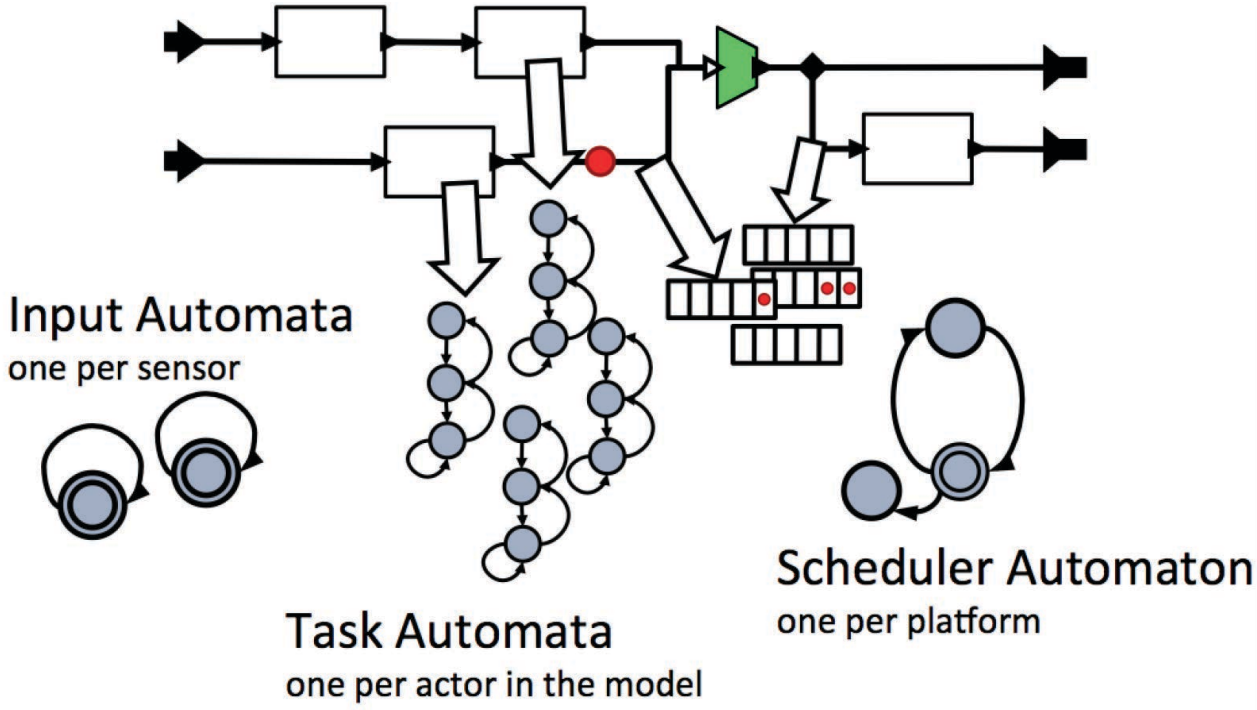
*All of the assumptions are achievable with today's technology, and in fact are **requirements** anyway for hard-real-time systems. The Ptides model makes the assumptions explicit.*

*Violations of the assumptions are detectable as out-of-order events and can be treated as **faults**.*

# Ptides Schedulability Analysis

## Determine *whether* deadlines can be met

The problem turns out to be decidable for a large class of models.



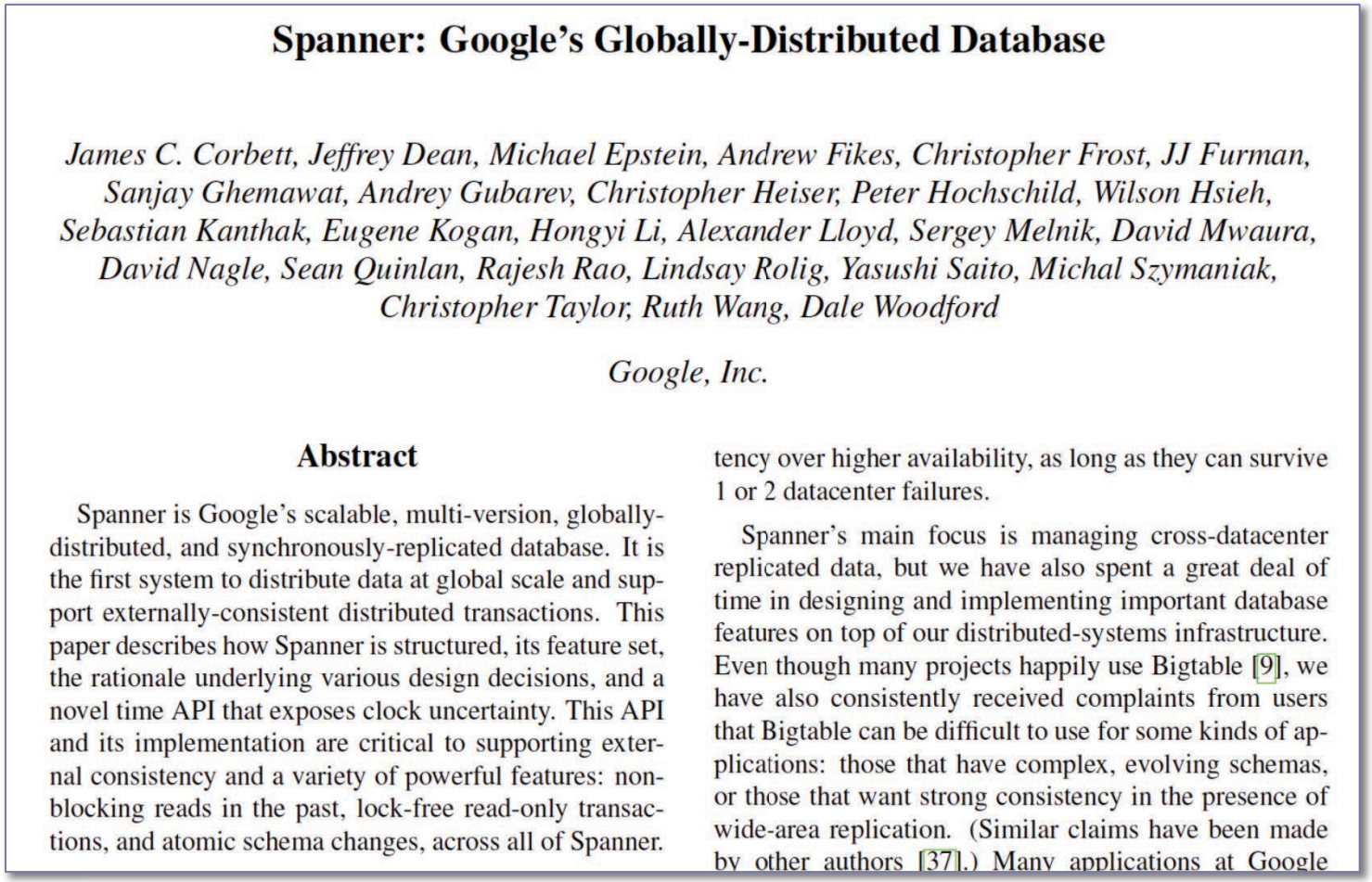

**On the Schedulability of Real-Time Discrete-Event Systems***

Eleftherios Matsikoudis Christos Stergiou Edward A. Lee

EMSOFT 2013

# Google Spanner

Google independently developed a very similar technique and applied it to distributed databases.



**Spanner: Google's Globally-Distributed Database**

*James C. Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, JJ Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, Wilson Hsieh, Sebastian Kanthak, Eugene Kogan, Hongyi Li, Alexander Lloyd, Sergey Melnik, David Mwaura, David Nagle, Sean Quinlan, Rajesh Rao, Lindsay Rolig, Yasushi Saito, Michal Szymaniak, Christopher Taylor, Ruth Wang, Dale Woodford*

*Google, Inc.*

**Abstract**

Spanner is Google's scalable, multi-version, globally-distributed, and synchronously-replicated database. It is the first system to distribute data at global scale and support externally-consistent distributed transactions. This paper describes how Spanner is structured, its feature set, the rationale underlying various design decisions, and a novel time API that exposes clock uncertainty. This API and its implementation are critical to supporting external consistency and a variety of powerful features: non-blocking reads in the past, lock-free read-only transactions, and atomic schema changes, across all of Spanner.

tency over higher availability, as long as they can survive 1 or 2 datacenter failures.

Spanner's main focus is managing cross-datacenter replicated data, but we have also spent a great deal of time in designing and implementing important database features on top of our distributed-systems infrastructure. Even though many projects happily use Bigtable [9], we have also consistently received complaints from users that Bigtable can be difficult to use for some kinds of applications: those that have complex, evolving schemas, or those that want strong consistency in the presence of wide-area replication. (Similar claims have been made by other authors [37].) Many applications at Google

Proceedings of OSDI 2012

# Ptides is a Change in Philosophy

The implementation platform affects timing in a distributed real-time system.

**Conventional approach**: Specify functionality, implementation architecture, and mapping. Timing emerges from the combination.

**Ptides approach**: Specify temporal behavior. Then verify that it is met by a candidate implementation architecture.

Ptides offers a *deterministic*
model of computation
for distributed real-time systems.

http://chess.eecs.berkeley.edu/ptides

# Challenge # 4

How to define interfaces between components that bridge engineering disciplines and clarify requirements and expectations?

*We need a discipline of "model engineering"*

Promising approaches:

- Heterogeneous MoCs
- Aspect-oriented modeling

*… but I will skip this story in the interest of time…*

# Four Big Challenges

1. Determinate CPS models
2. Open minds about languages and tools
3. A semantics of time
4. A discipline of “model engineering”

*Raffaello Sanzio da Urbino – The Athens School* *Image: Wikimedia Commons*

# Acknowledgements


- David Broman (PRET)
- Patricia Derler (PTIDES)
- John Eidson (PTIDES, clock synchronization)
- Isaac Liu (PRET)
- Xiaojun Liu (Time)
- Slobodan Matic (PTIDES)
- Eleftherios D. Matsikoudis (Time)
- Christos Stergiou (PTIDES)
- Stavros Tripakis (Modeling)
- Yang Zhao (PTIDES)
- Haiyang Zheng (Time)
- Michael Zimmer (PRET)
- Jia Zou (PTIDES)

Plus: The entire Ptolemy II Pteam