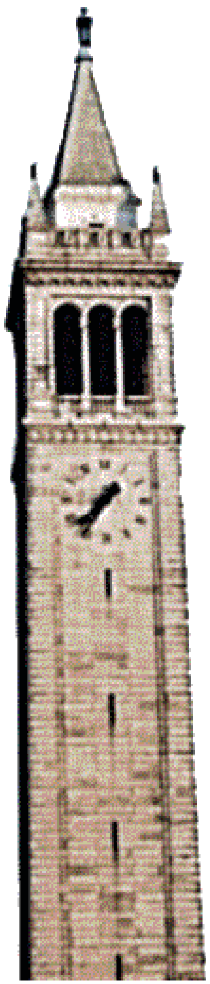


The Internet of Important Things

Edward A. Lee

Robert S. Pepper Distinguished Professor
UC Berkeley

Keynote

Time Sensitive Networks and Applications (TSNA)
April 28-29, 2015.
Santa Clara, CA

The Context for this Talk: Cyber-Physical Systems or The Internet of Important Things (IoIT)

Leveraging Internet technology in cyber-physical systems.

***Challenges**:*

- **Isolated networks** are reliable, predictable, and controllable. But they lose the benefits of **connectedness**.
- ***Safety** is the most critical design requirement.*
- ***Security** is essential, particularly w.r.t. how it impacts safety.*
- ***Privacy** (protection of data) is required.*

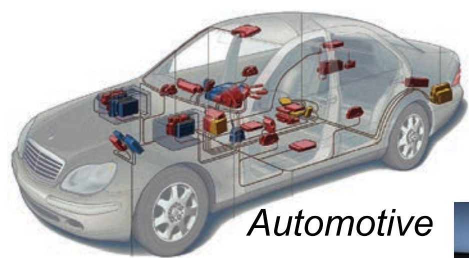
This Bosch Rexroth printing press is a cyber-physical factory using Ethernet and TCP/IP with high-precision clock synchronization (IEEE 1588) on an isolated LAN.

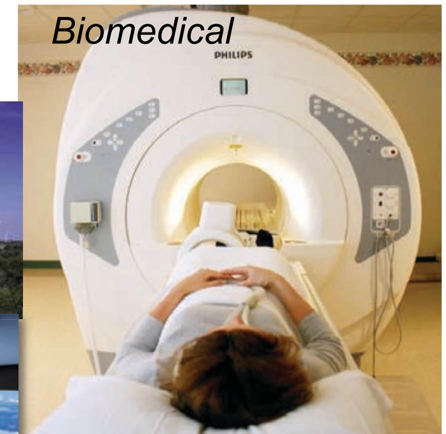
IoIT and CPS

Underlie much of the industrial economy

It’s not just information technology anymore:

- Cyber + *Physical*
- Computation + *Dynamics*
- Security + *Safety*

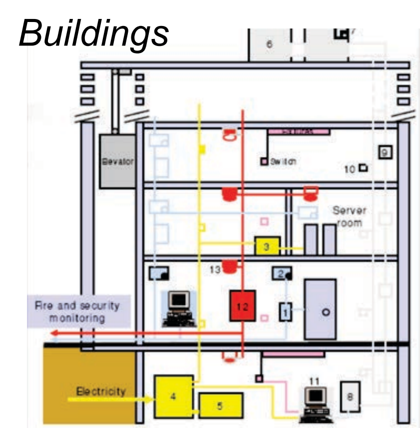




Contradictions:

- Algorithms vs. *Dynamics*
- Economies of scale (cloud) vs. *Locality (fog)*
- High performance vs. *Low Energy*
- Asynchrony vs. *Coordination/Cooperation*
- Adaptability vs. *Repeatability*
- High connectivity vs. *Security and Privacy*
- Scalability vs. *Reliability and Predictability*
- Open vs. *Proprietary*
- Laws and Regulations vs. *Technical Possibilities*

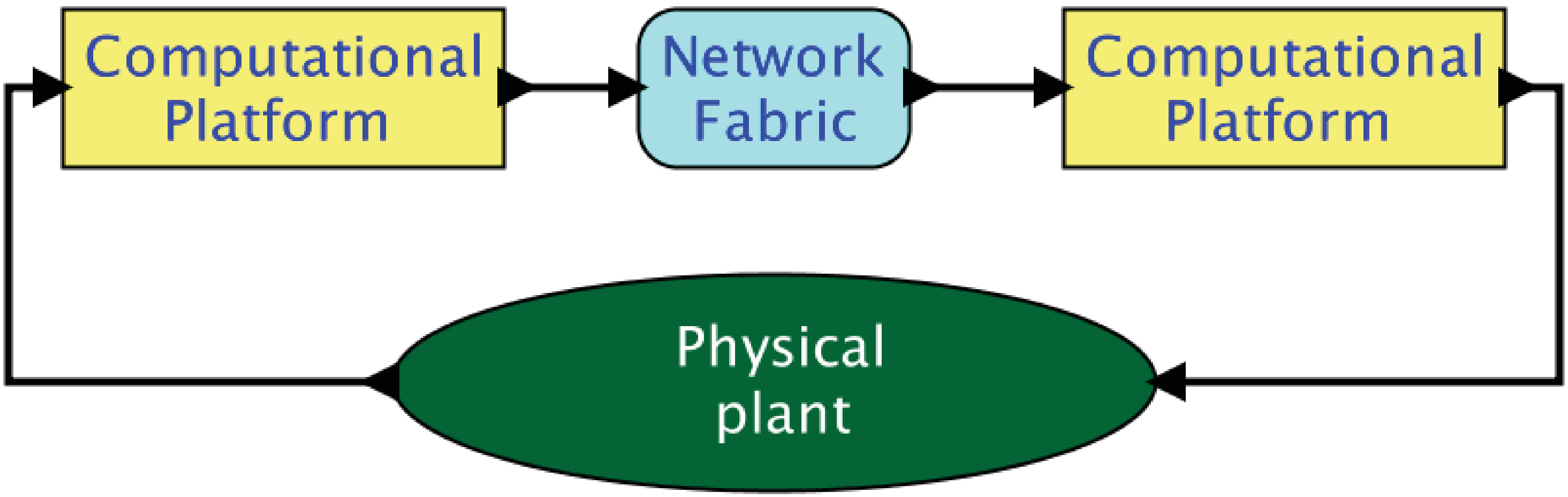
Manufacturing



Innovation:

Cyber-physical systems are fundamentally different from computational systems and from physical systems. They require new engineering models that embrace temporal dynamics and algorithmic computation.

Schematic of a simple CPS



In CPS, "cyber" == "software" and "physical" == "not software". Digital hardware sits in a gray area…

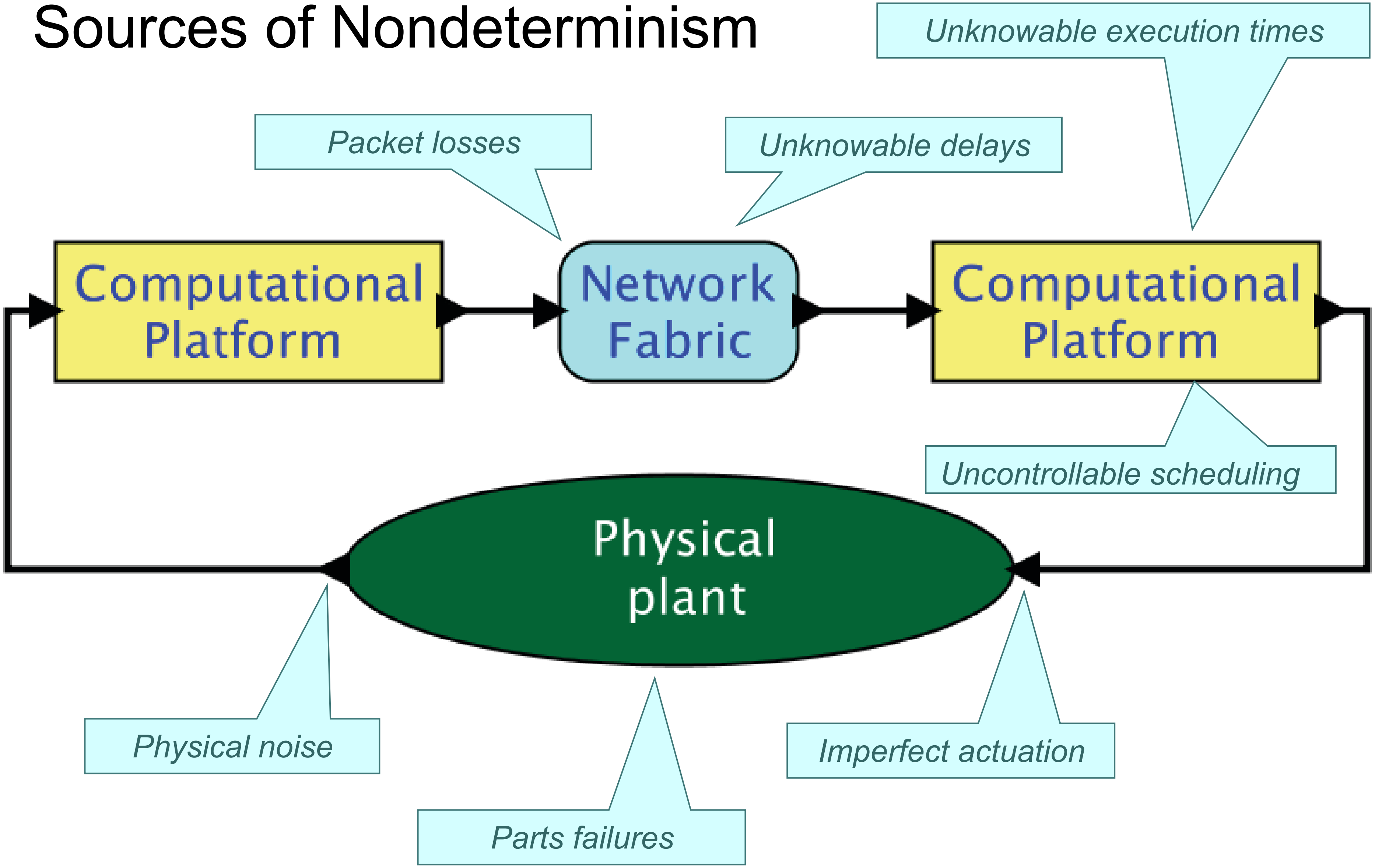
The Theme of This Talk

Determinacy

or

Better Engineering through Better Models

Sources of Nondeterminism

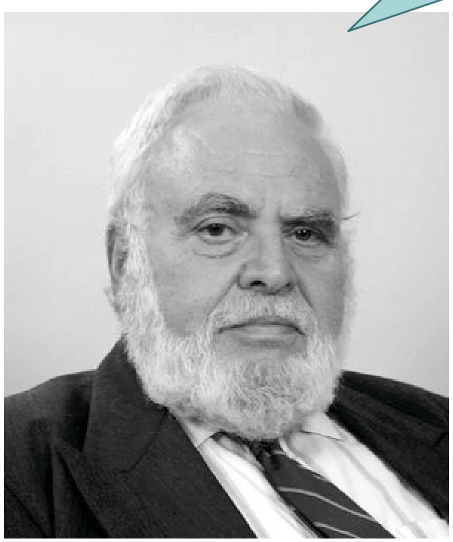


In the face of such nondeterminism, does it make sense to talk about deterministic models for cyber-physical systems?

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.

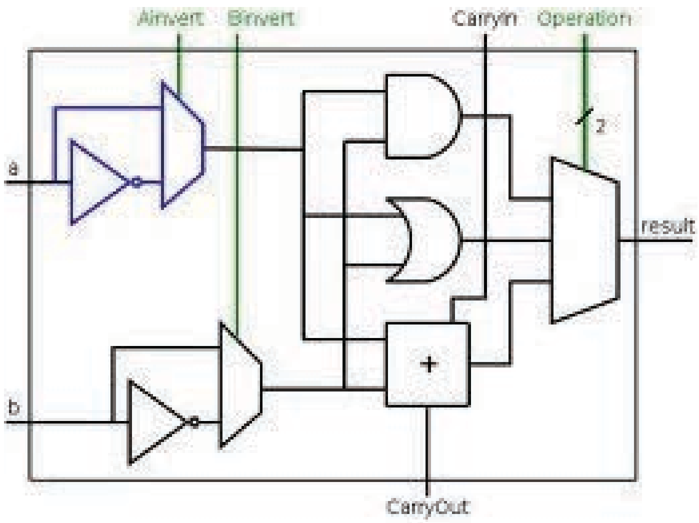
(paraphrased)

Deterministic Models of Nondeterministic Systems

Physical System

Model

Image: Wikimedia Commons



Synchronous digital logic

Deterministic Models of Nondeterministic Systems

Physical System

Image: Wikimedia Commons

Model

Integer Register-Register Operations

RISC-V defines several arithmetic R-type operations. All operations read the *rs1* and *rs2* registers as source operands and write the result into register *rd*. The *funct* field selects the type of operation.

31–27	26–22	21–17	16–7	6–0
rd	rs1	rs2	funct10	opcode
5	5	5	10	7
dest	src1	src2	ADD/SUB/SLT/SLTU	OP
dest	src1	src2	AND/OR/XOR	OP
dest	src1	src2	SLL/SRL/SRA	OP
dest	src1	src2	ADDW/SUBW	OP-32
dest	src1	src2	SLLW/SRLW/SRAW	OP-32

Waterman, et al., The RISC-V Instruction Set Manual, UCB/EECS-2011-62, 2011

Instruction Set Architectures (ISAs)

Deterministic Models of Nondeterministic Systems

Physical System

Image: Wikimedia Commons

Model

```java
/** 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

Deterministic Models of Nondeterministic Systems

Physical System

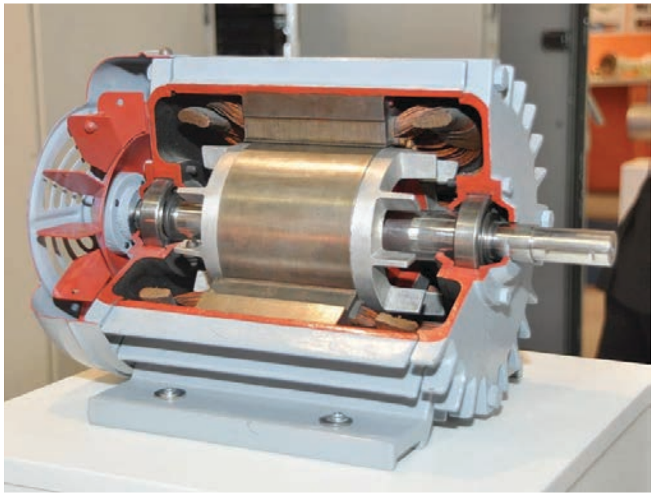
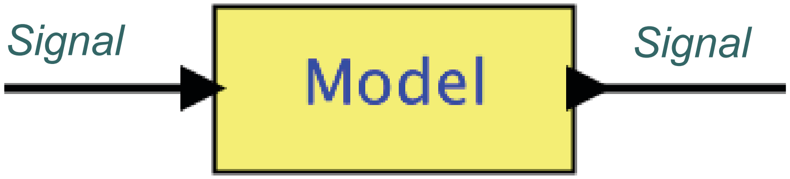


Image: Wikimedia Commons

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 of these Models are Nondeterministic

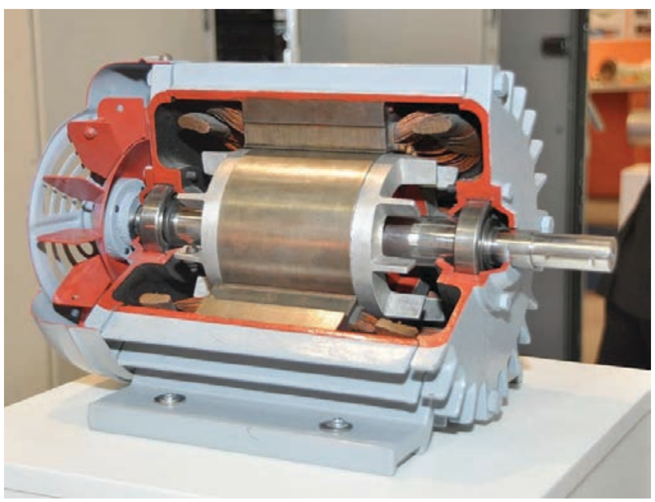
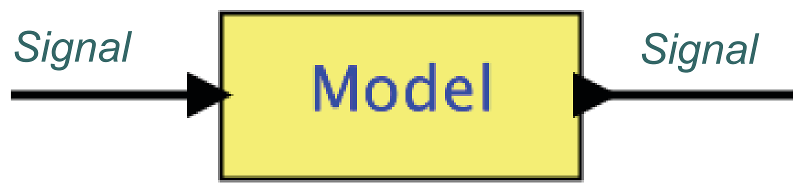


Image: Wikimedia Commons

```java
/** 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();
                        }
                    }
                }
            }
        }
    }
}
```



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

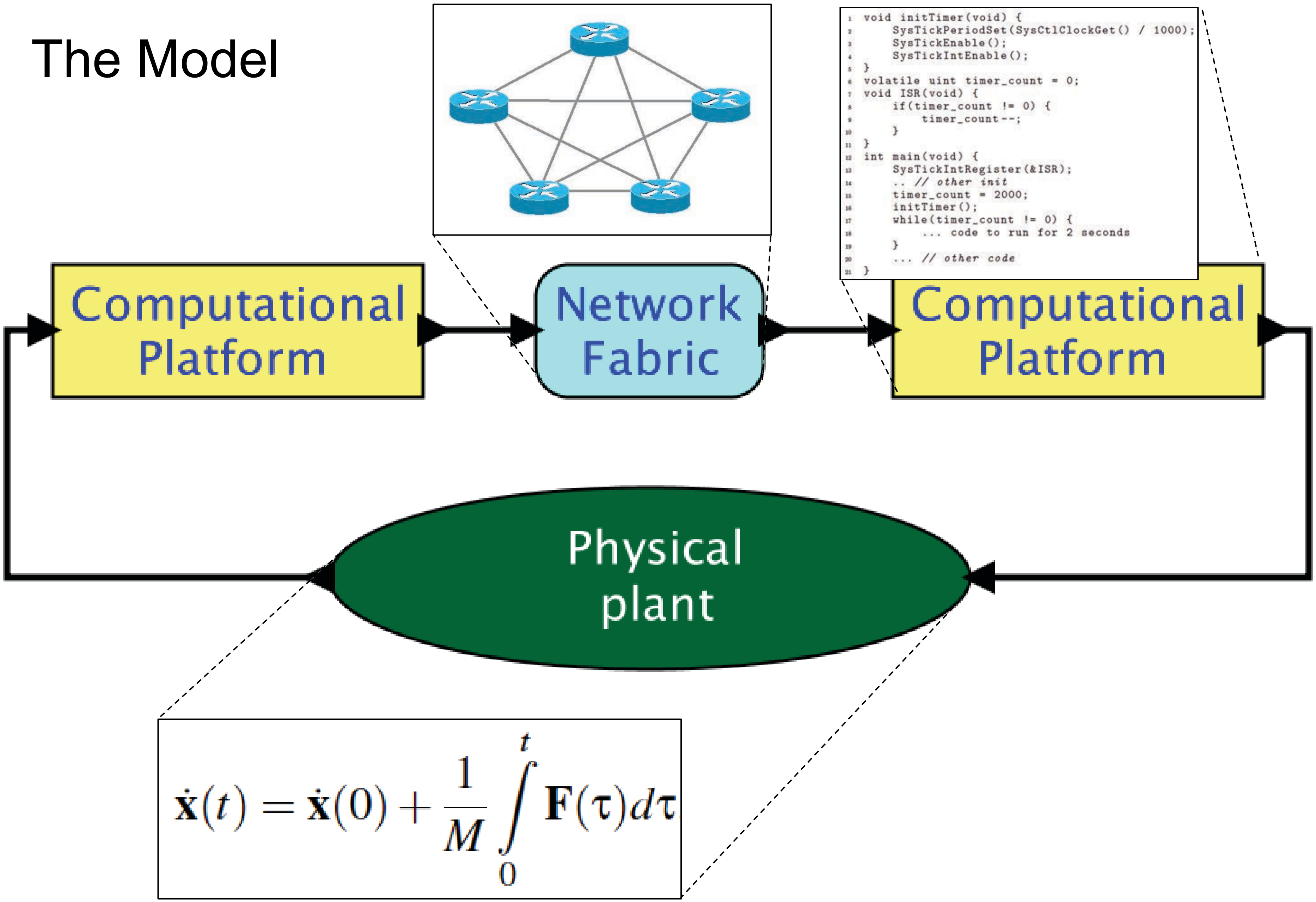
A Key Challenge:
Timing is not Part of Software Semantics

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.

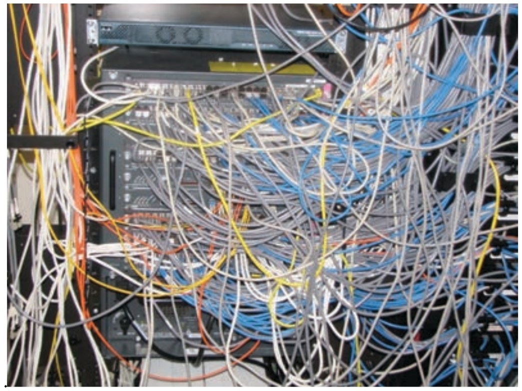
Programmers have no map!

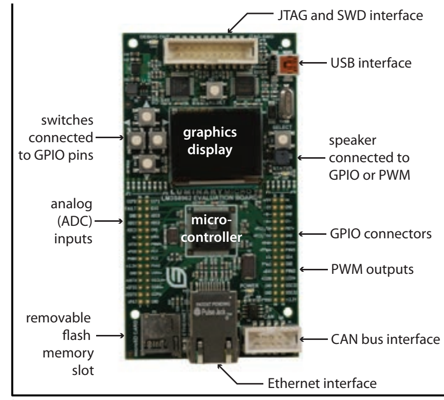
The Model



Computational Platform
Network Fabric
Computational Platform
Physical plant
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
}
$\dot{\mathbf{x}}(t) = \dot{\mathbf{x}}(0) + \frac{1}{M}\int_0^t \mathbf{F}(\tau)d\tau$

The Reality





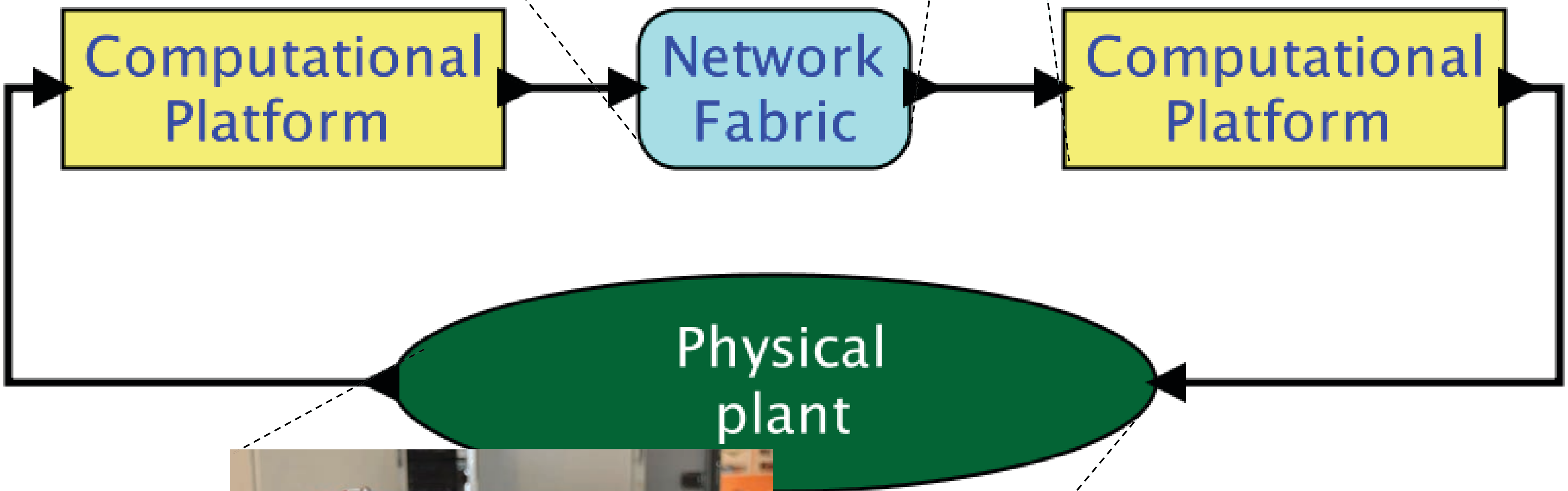
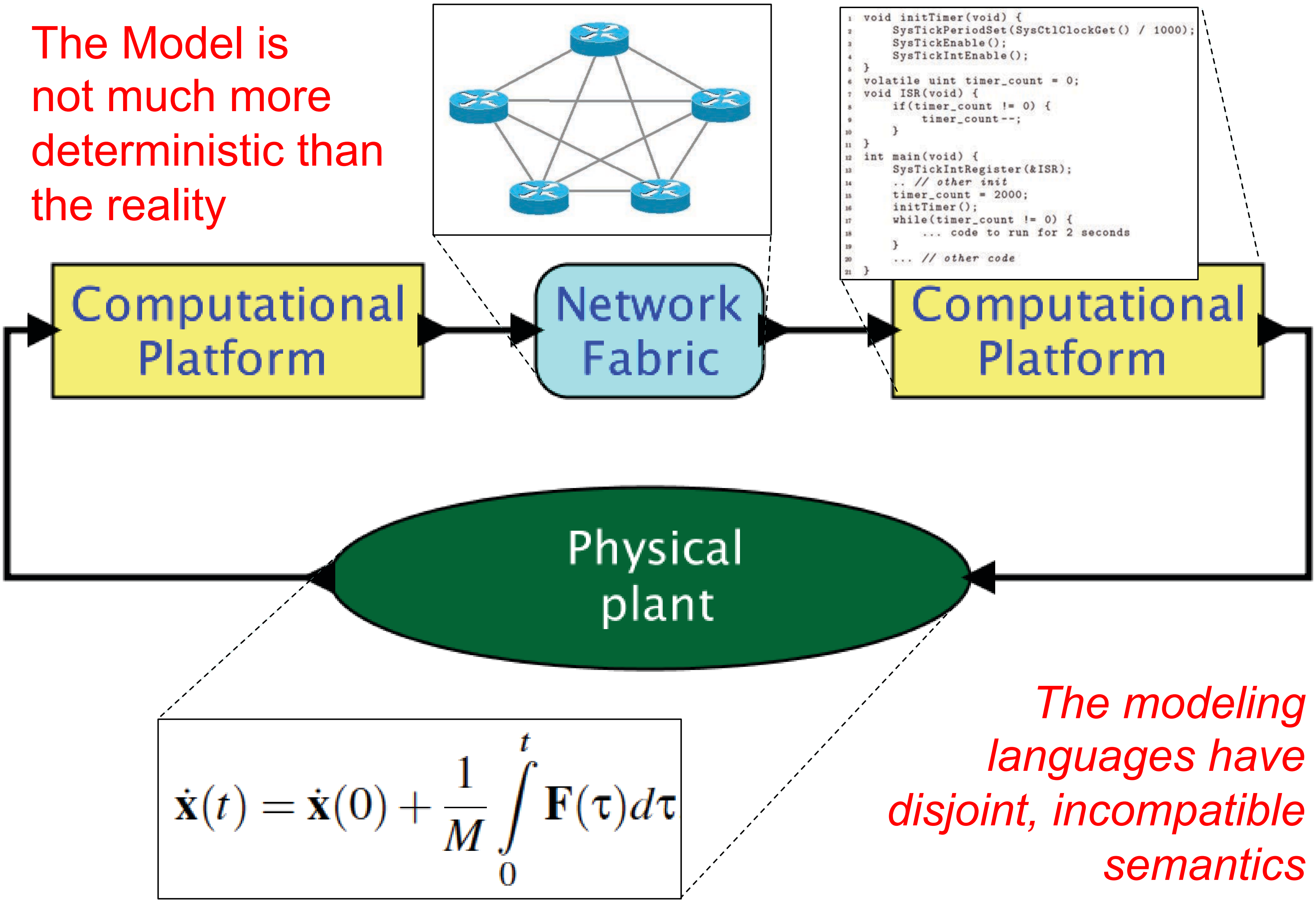
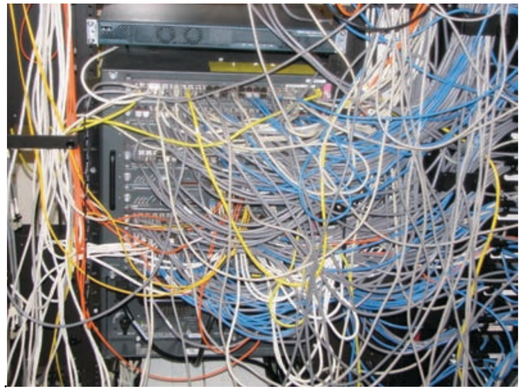
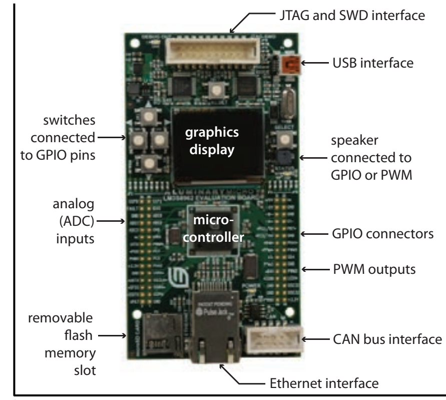


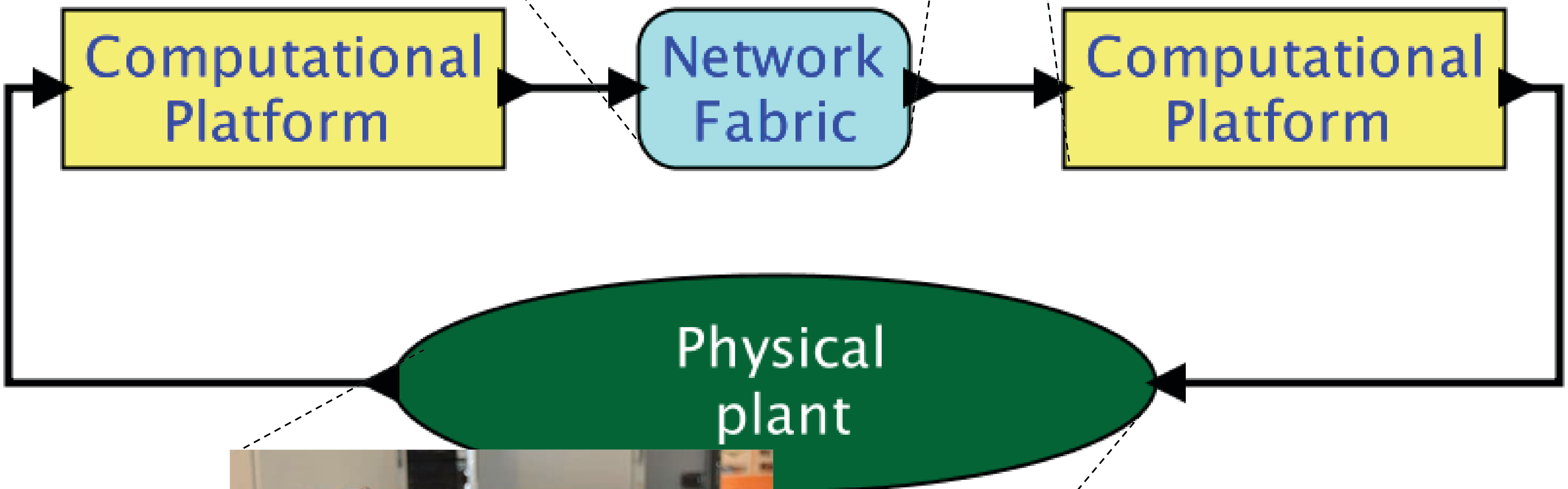
Image: Wikimedia Commons


The Model is not much more deterministic than the reality
Computational Platform
Network Fabric
Computational Platform
Physical plant
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
}
$\dot{\mathbf{x}}(t) = \dot{\mathbf{x}}(0) + \frac{1}{M}\int_0^t \mathbf{F}(\tau)d\tau$
The modeling languages have disjoint, incompatible semantics

System dynamics emerges from the physical realization

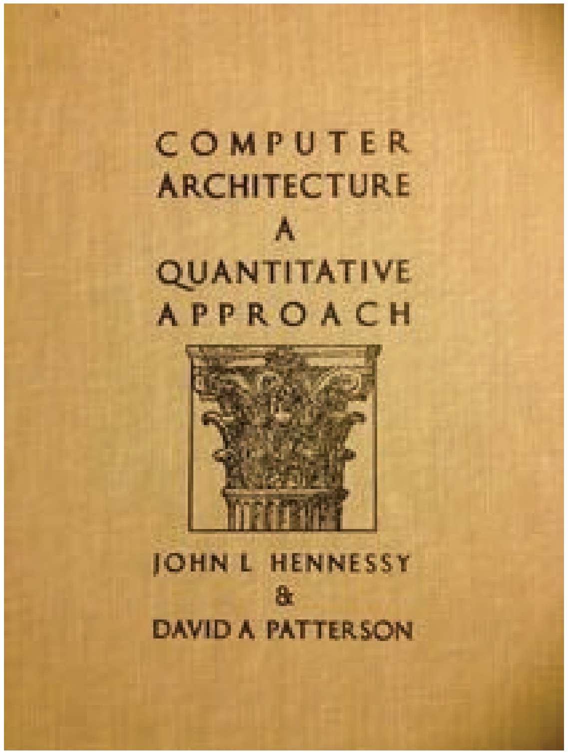






… leading to a "prototype and test" style of design

Computer Science has not *completely* 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

(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.)

```
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 }
```

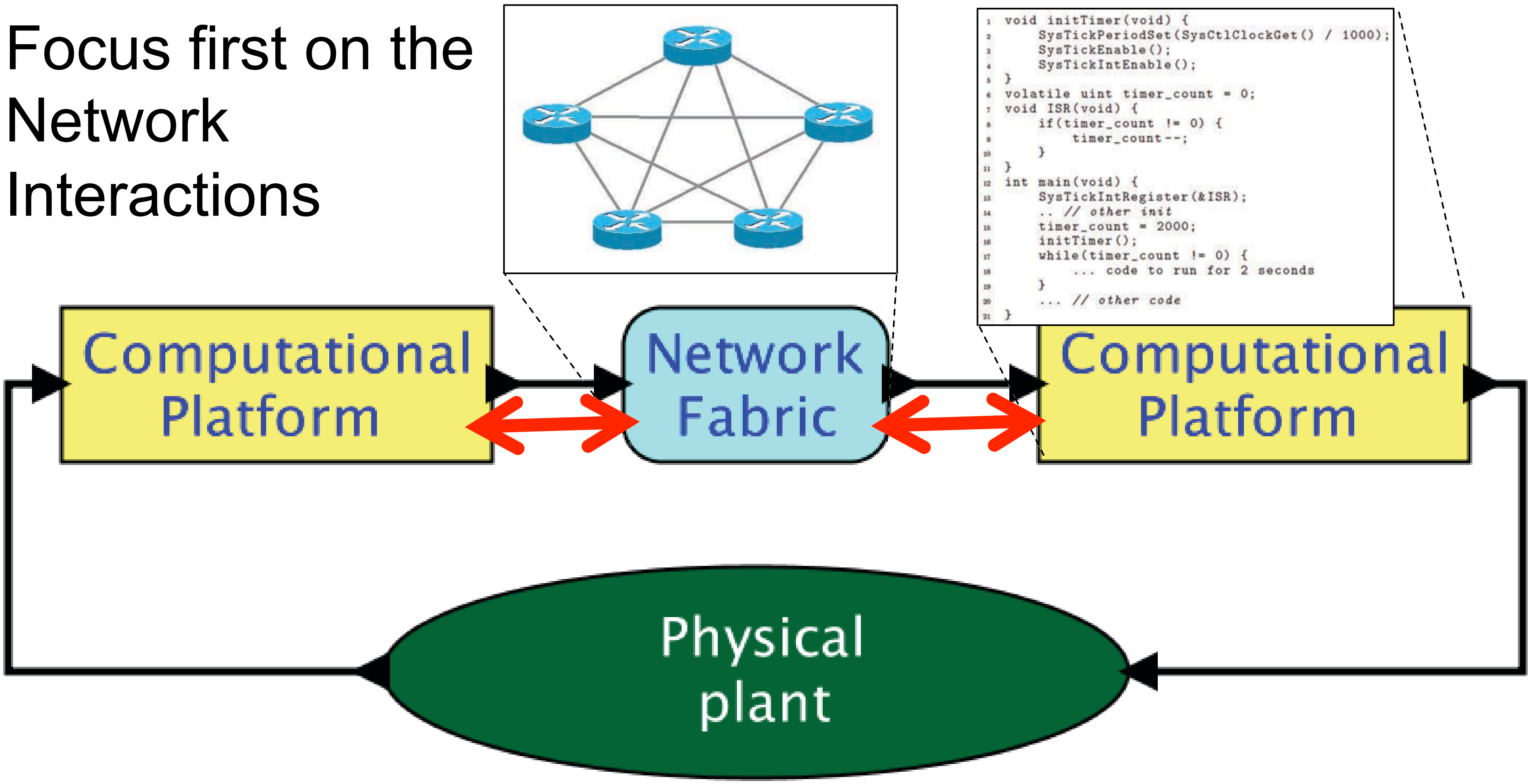
We can develop ***absolute confidence*** *in the software, in that only a* ***hardware failure*** *is an excuse.*

But not with regards to timing!!

Research Efforts at Berkeley
Better Engineering through Better Models

- PTIDES: distributed real-time software
  - Deterministic timing of distributed CPS
- PRET machines
  - Deterministic timing at the processor level
- Accessors
  - Principled composition of networked components
- Open-source software
  - Ptolemy II
- Model-based design (iCyPhy)
  - Interfaces (e.g. FMI), contracts, aspects, …
- Semantics
  - Timed models of computation,

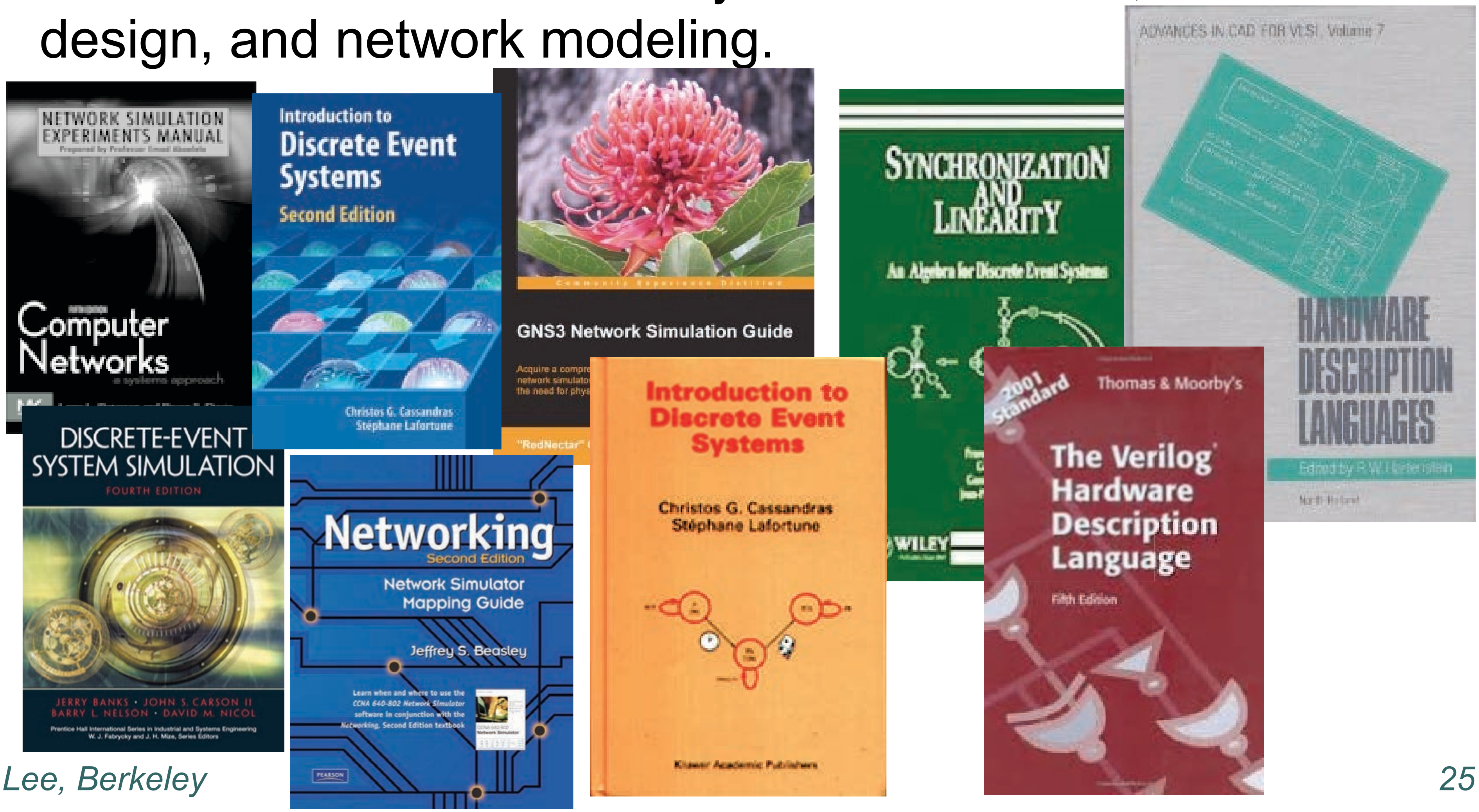
Focus first on the Network Interactions



We also developed *deterministic models* for distributed real-time software, using a technique called PTIDES.

Our Proposal: Discrete-Event Semantics + Synchronized Clocks

DE models have been widely used simulation, hardware design, and network modeling.



Using Discrete Event Semantics in Distributed Real-Time Systems

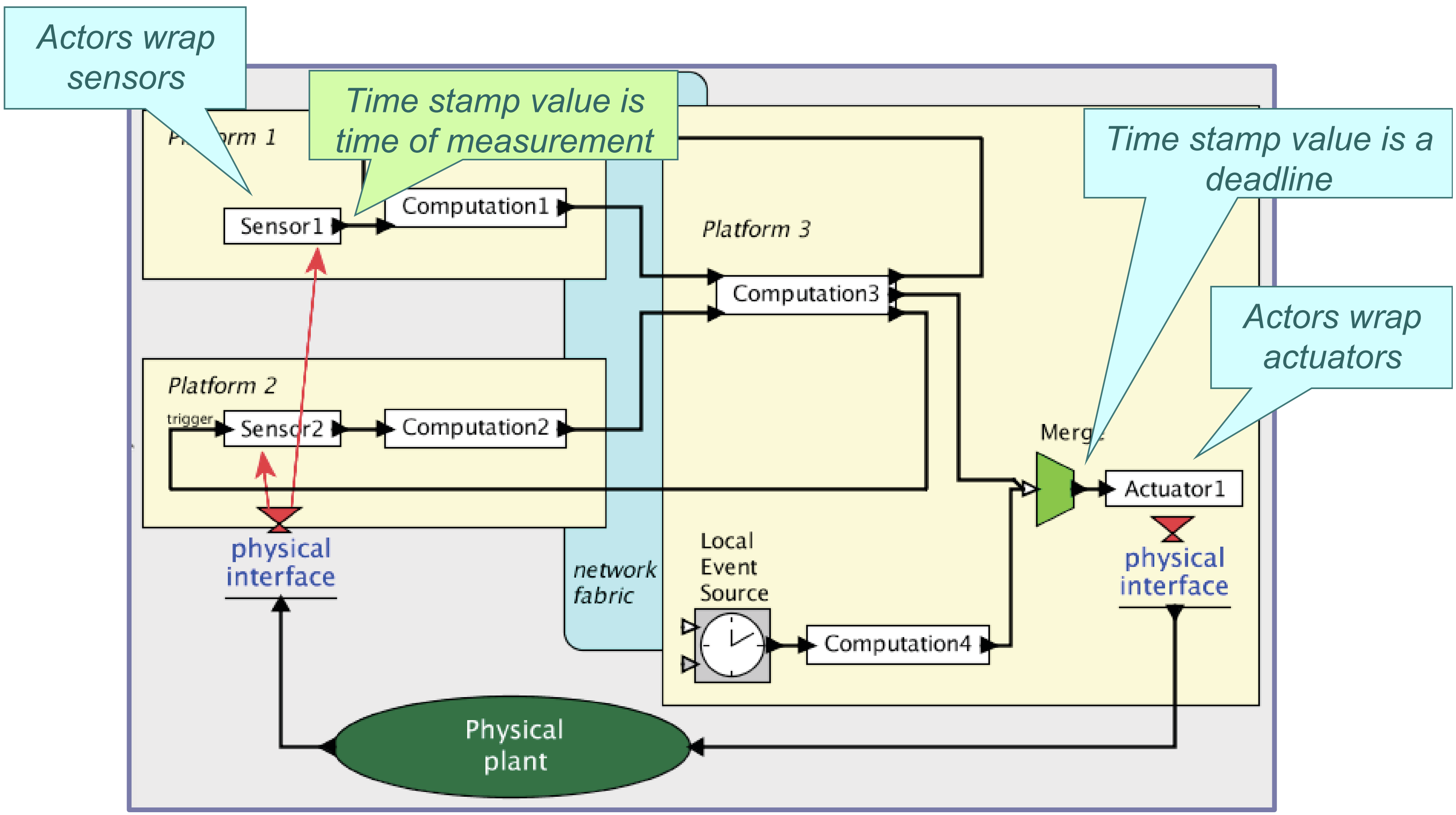
- DE is usually used for simulation (HDLs, network simulators, …)
- Distributing DE is done to accelerate simulation.

We are using DE for distributed real-time software, binding time stamps to real time only where necessary.

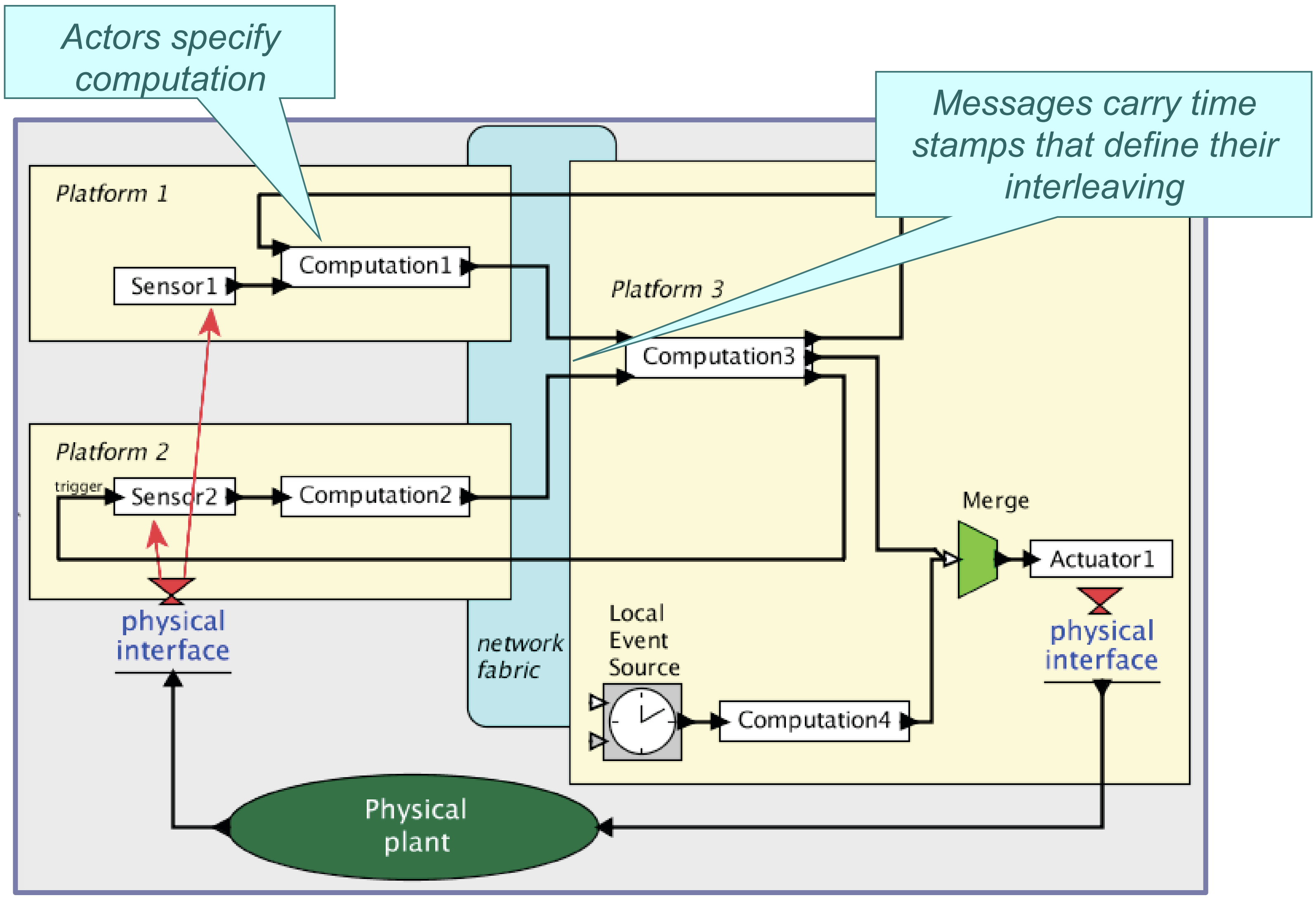
PTIDES: Programming Temporally Integrated Distributed Embedded Systems

Y. Zhao, E.A. Lee, J. Liu, “A Programming Model for Time-Synchronized Distributed Real-Time Systems,” *Proc. Real-Time and Embedded Technology and Applications Symposium (RTAS)*, IEEE, 2007, pp. 259 - 268.

Ptides: First step:
Time stamps bind to real time at sensors and actuators

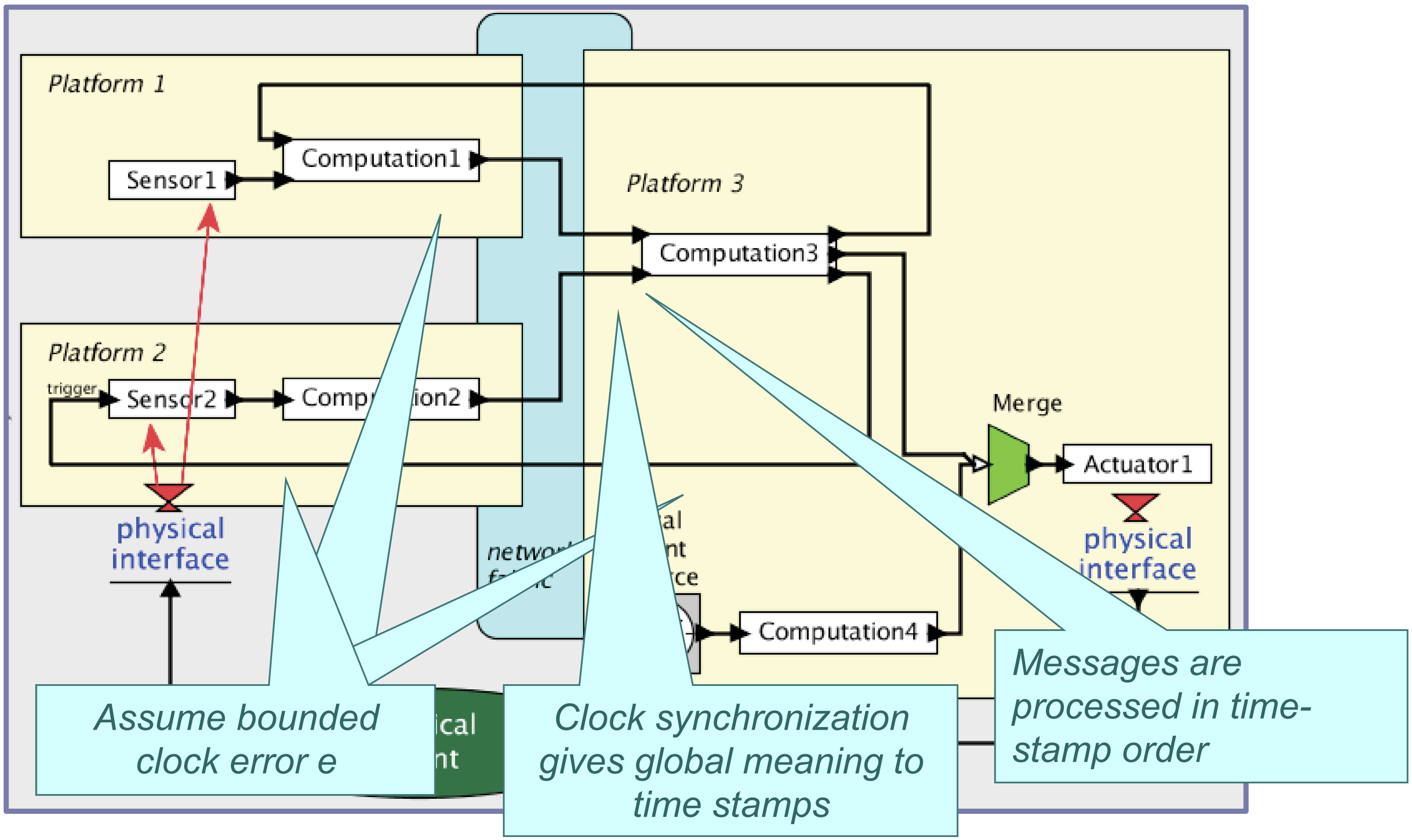


Ptides: Second step: Time-stamped messages.



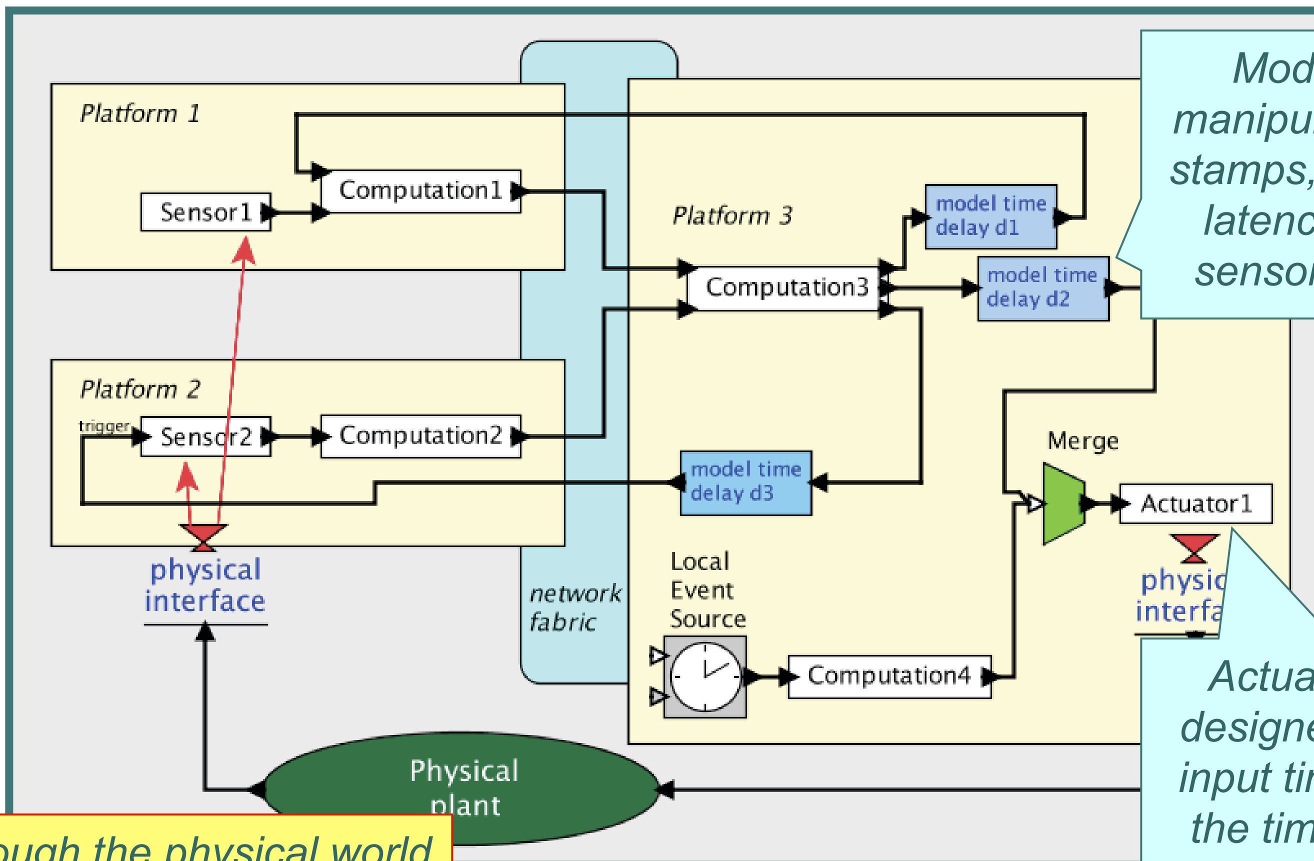
Ptides: Third step: Network clock synchronization

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



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

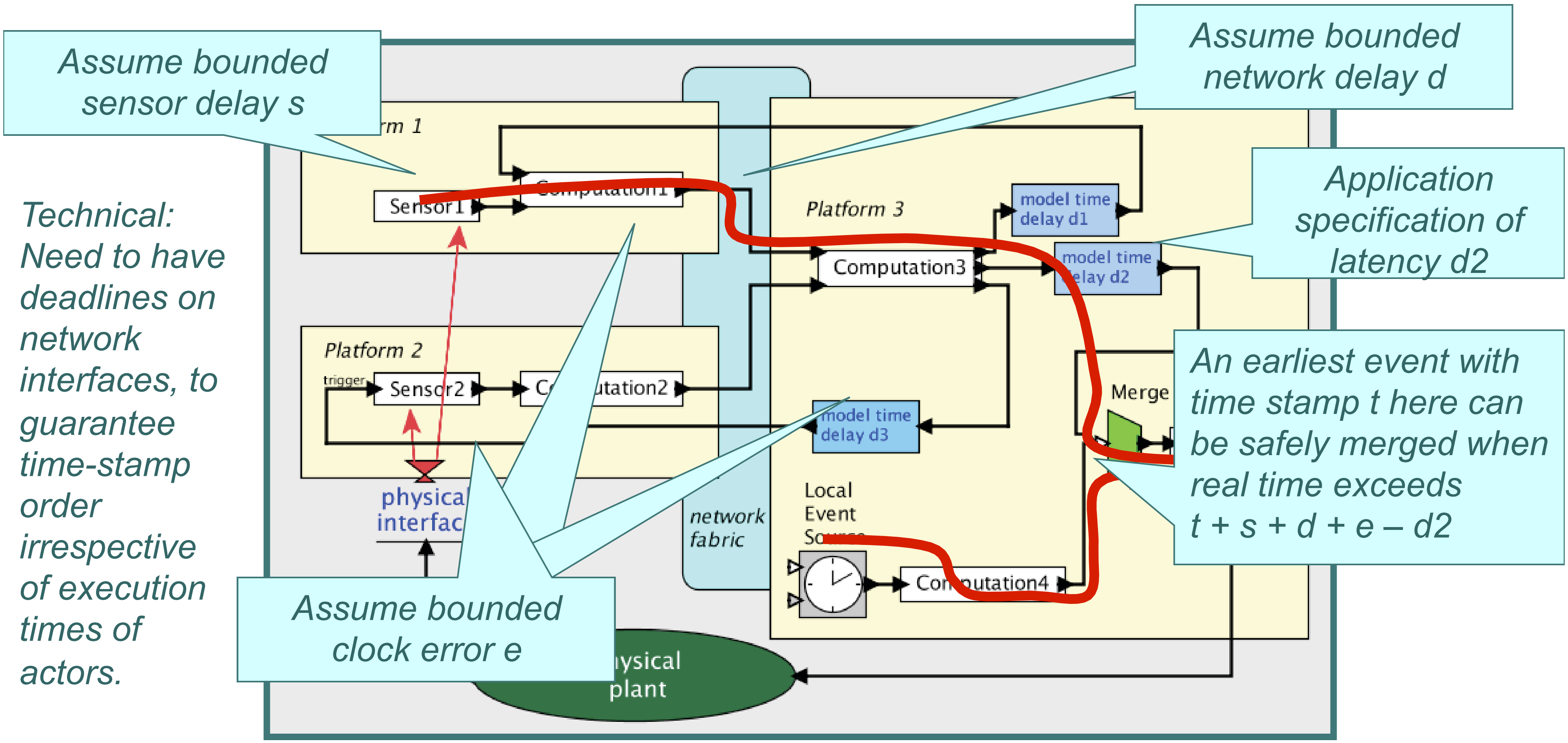
Actuators may be designed to interpret input time stamps as the time at which to take action.

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.

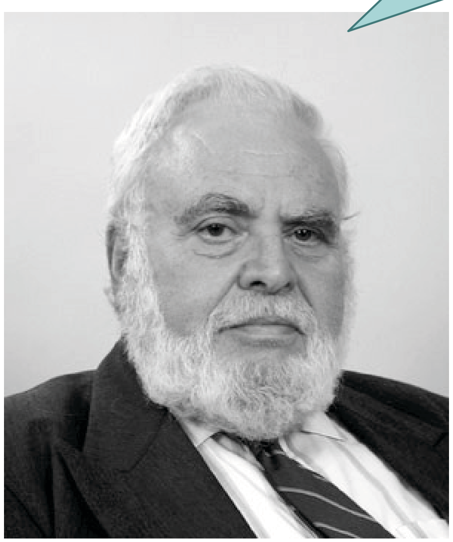
Technical: Need to have deadlines on network interfaces, to guarantee time-stamp order irrespective of execution times of actors.



So Many Assumptions?

Recall Solomon Wolf Golomb:

You will never strike oil by drilling through the map!



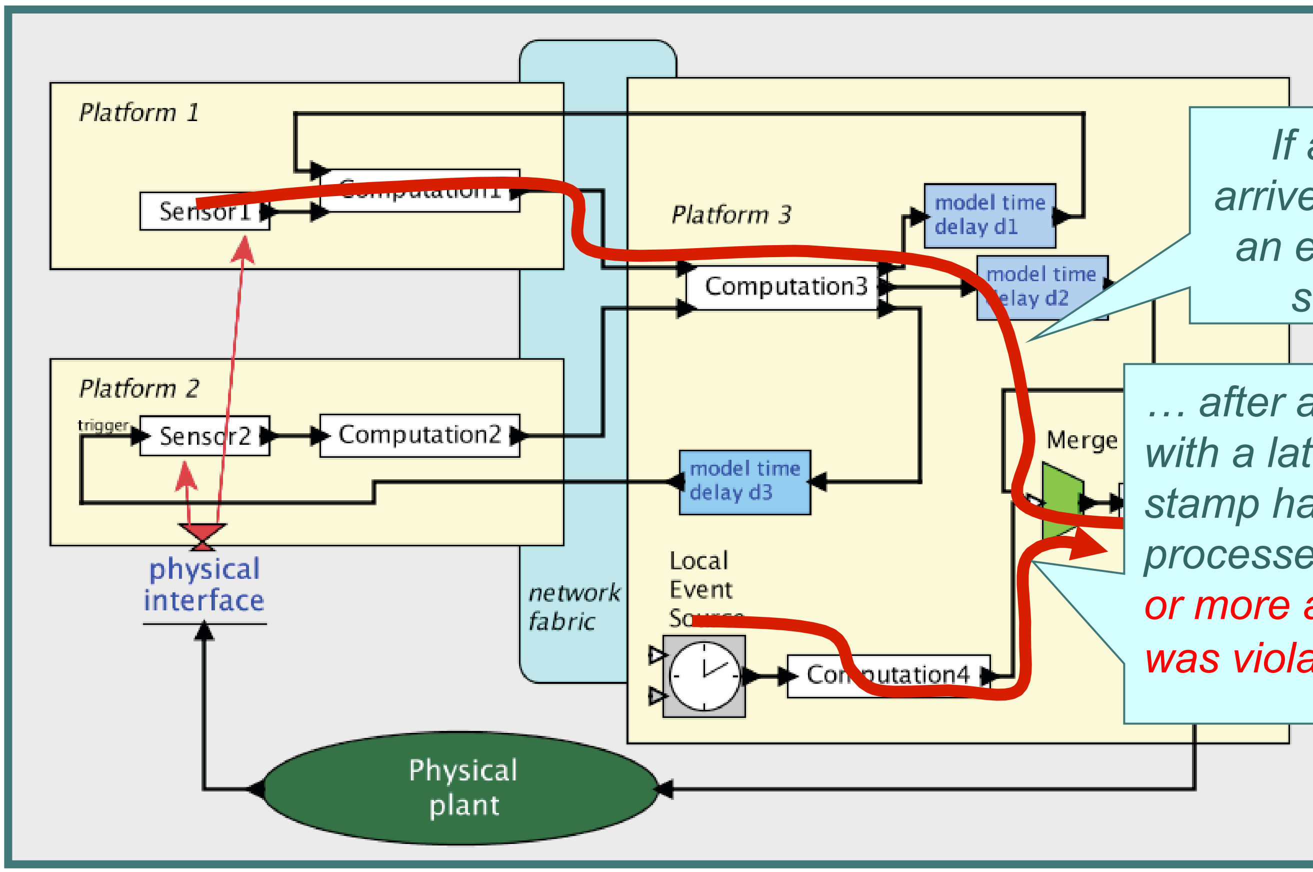
*All of the assumptions are achievable with today's technology, and in fact are **requirements** anyway for hard-real-time systems. The Ptides model makes the assumptions explicit.*

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

Handling Faults

A “fault” is a violation of assumptions in the model.

As with any model, the physical world may not conform to its rules. Violations should be treated as faults.

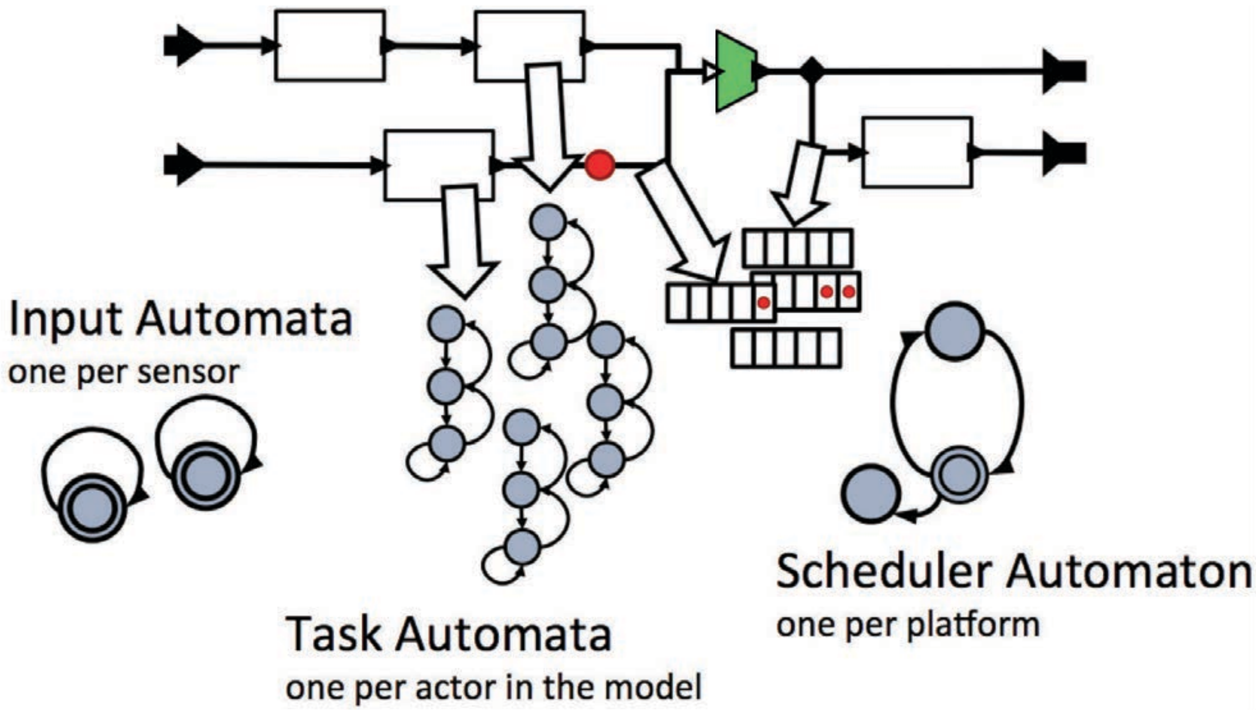


If an event arrives here with an earlier time stamp…

… after an event here with a later time stamp has been processed, then one or more assumptions was violated.

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

Google Spanner

Record update comes in. Time stamp t_1.

Query for the same record comes in. Time stamp t_2.

Distributed database with redundant storage and query handling across data centers.

Google Spanner

Record update comes in. Time stamp t_1.

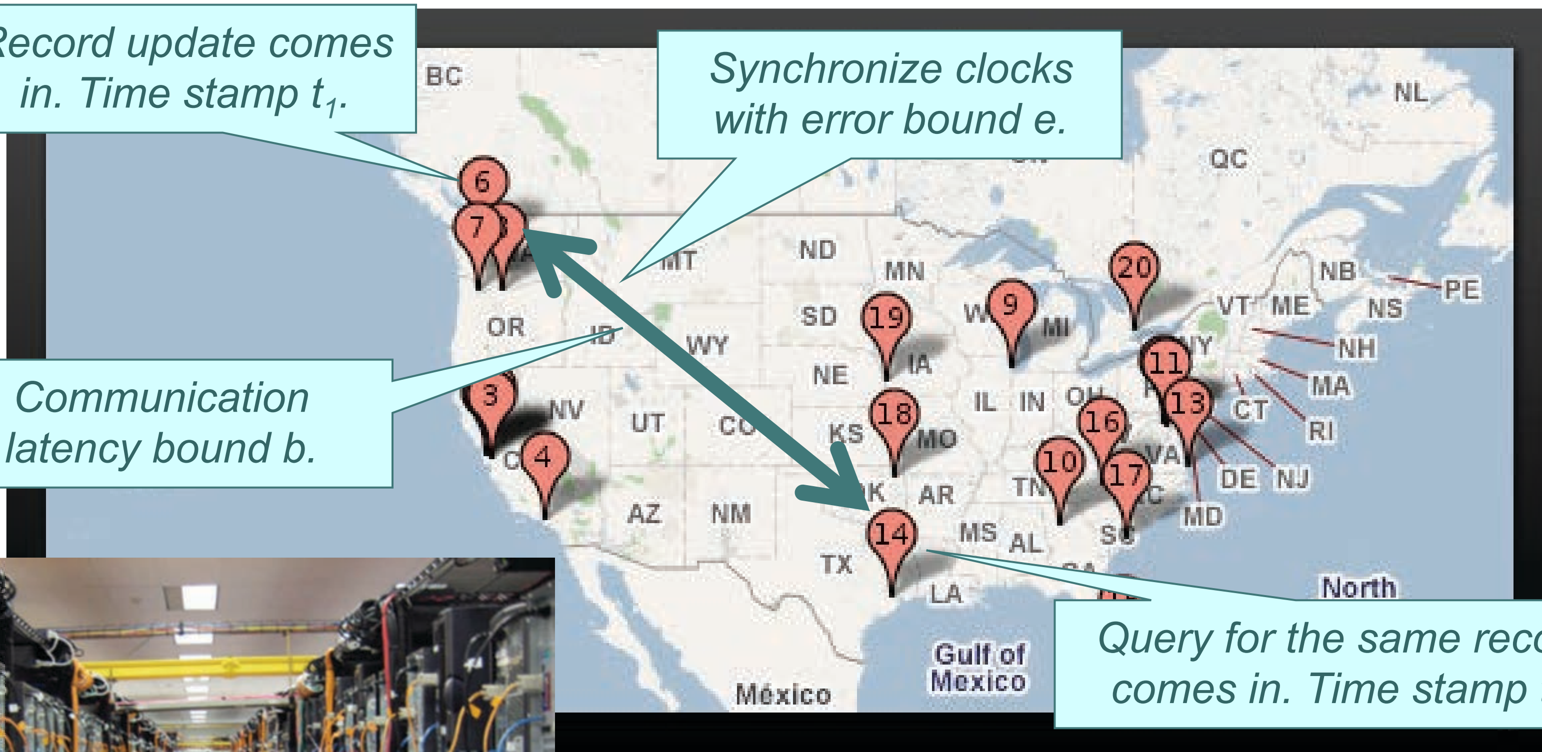
Query for the same record comes in. Time stamp t_2.

If $t_2 < t_1$, the query response should be the pre-update value. Otherwise, it should be the post-update value.

Google Spanner: When to Respond?

Record update comes in. Time stamp t_1.

Synchronize clocks with error bound e.



Communication latency bound b.

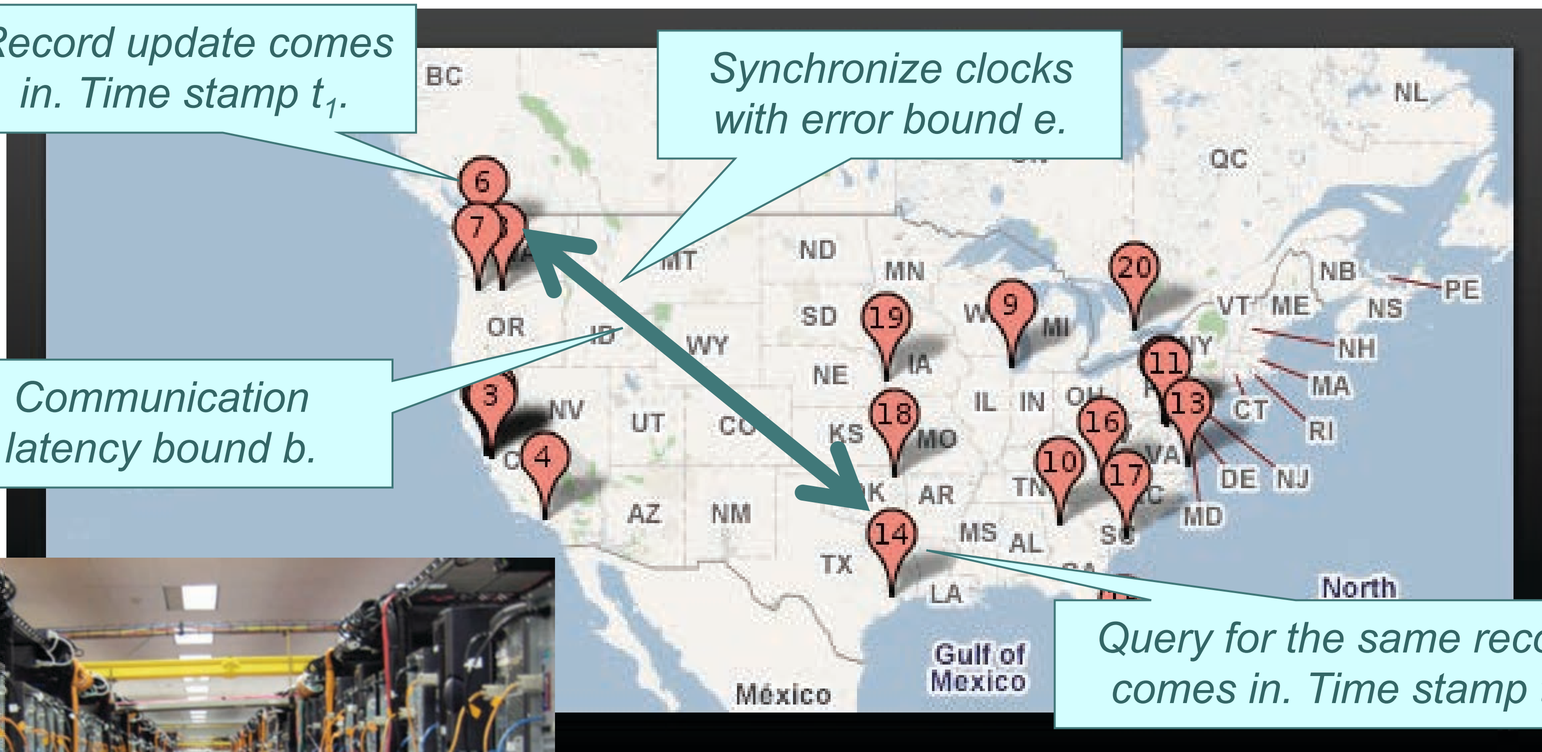
Query for the same record comes in. Time stamp t_2.

When the local clock time exceeds $t_2 + e + d$, issue the current record value as a response.

Google Spanner: Fault!

Record update comes in. Time stamp t_1.

Synchronize clocks with error bound e.



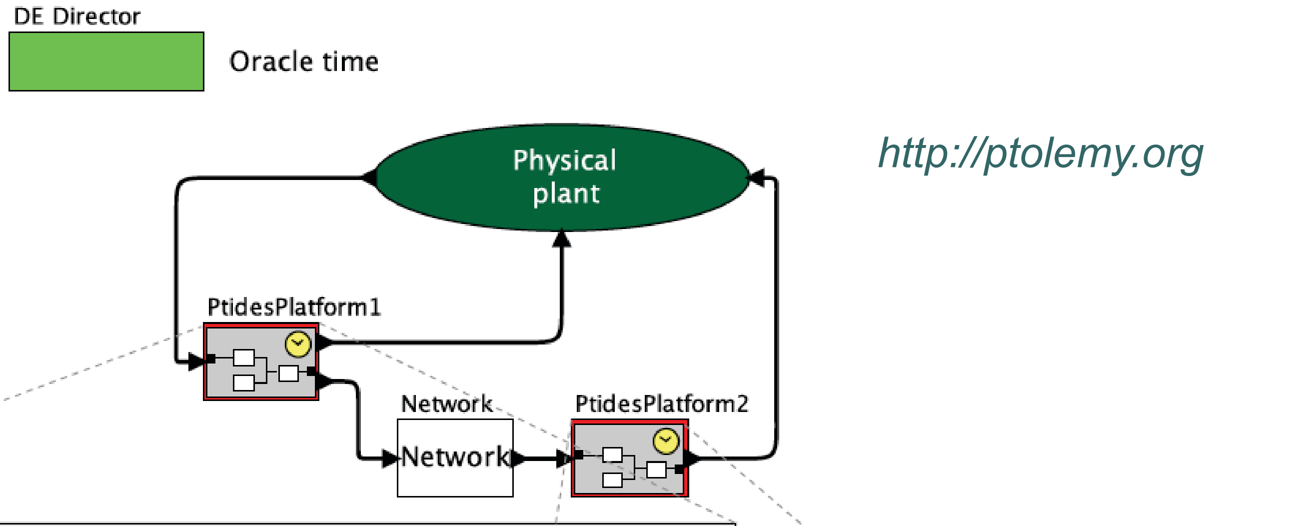
Communication latency bound b.

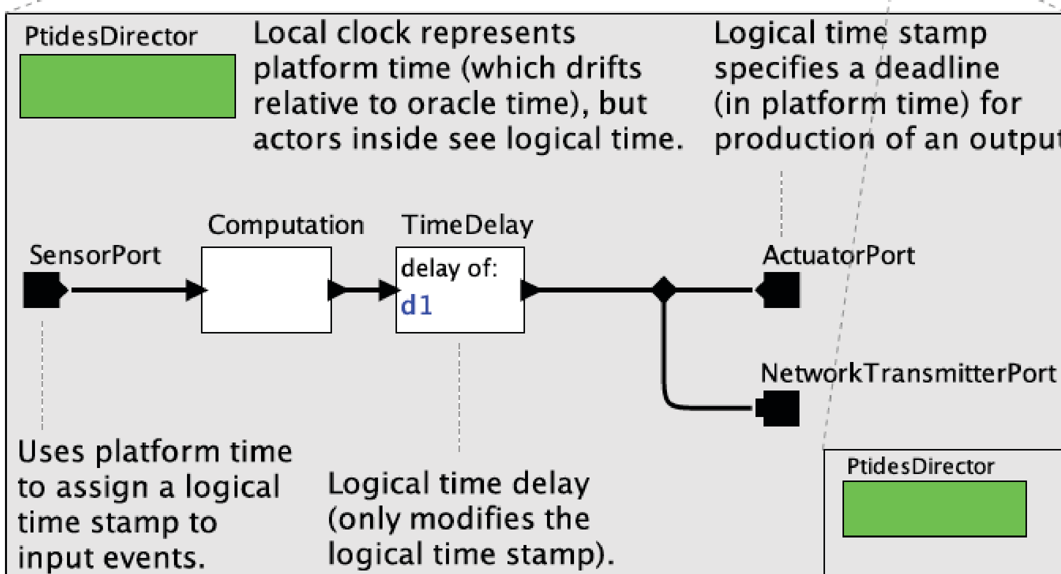
Query for the same record comes in. Time stamp t_2.

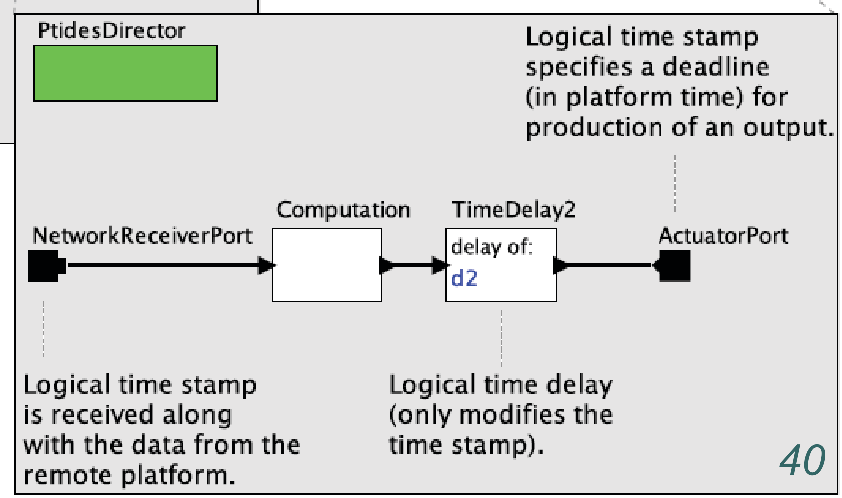
If after sending a response, we receive a record update with time stamp $t_1 < t_2$ declare a fault. Spanner handles this with a transaction schema.

Ptides in Ptolemy II

http://ptolemy.org







Open-source modeling and simulation environment, with Ptides support created by Patricia Derler.

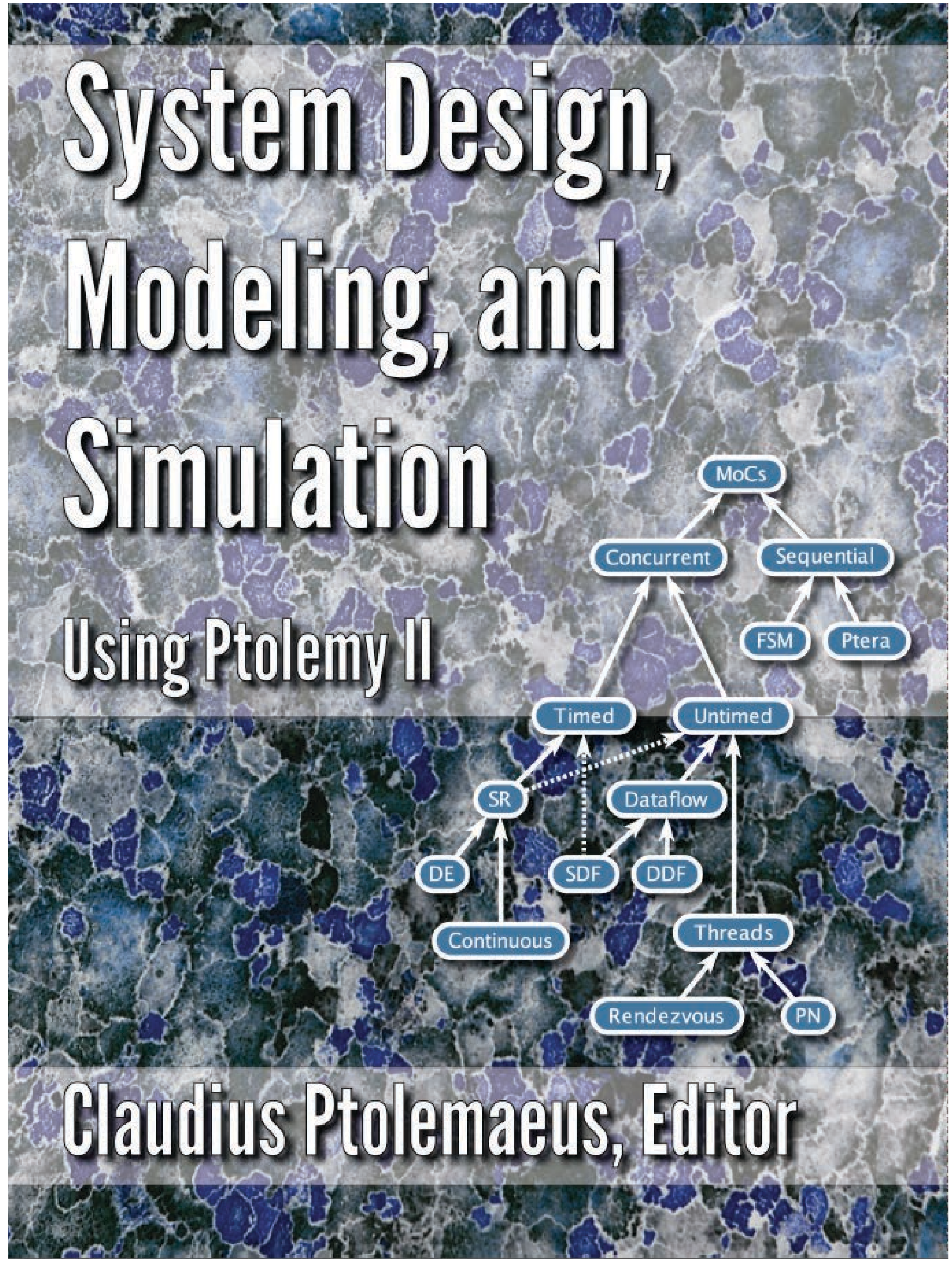
See Book

See

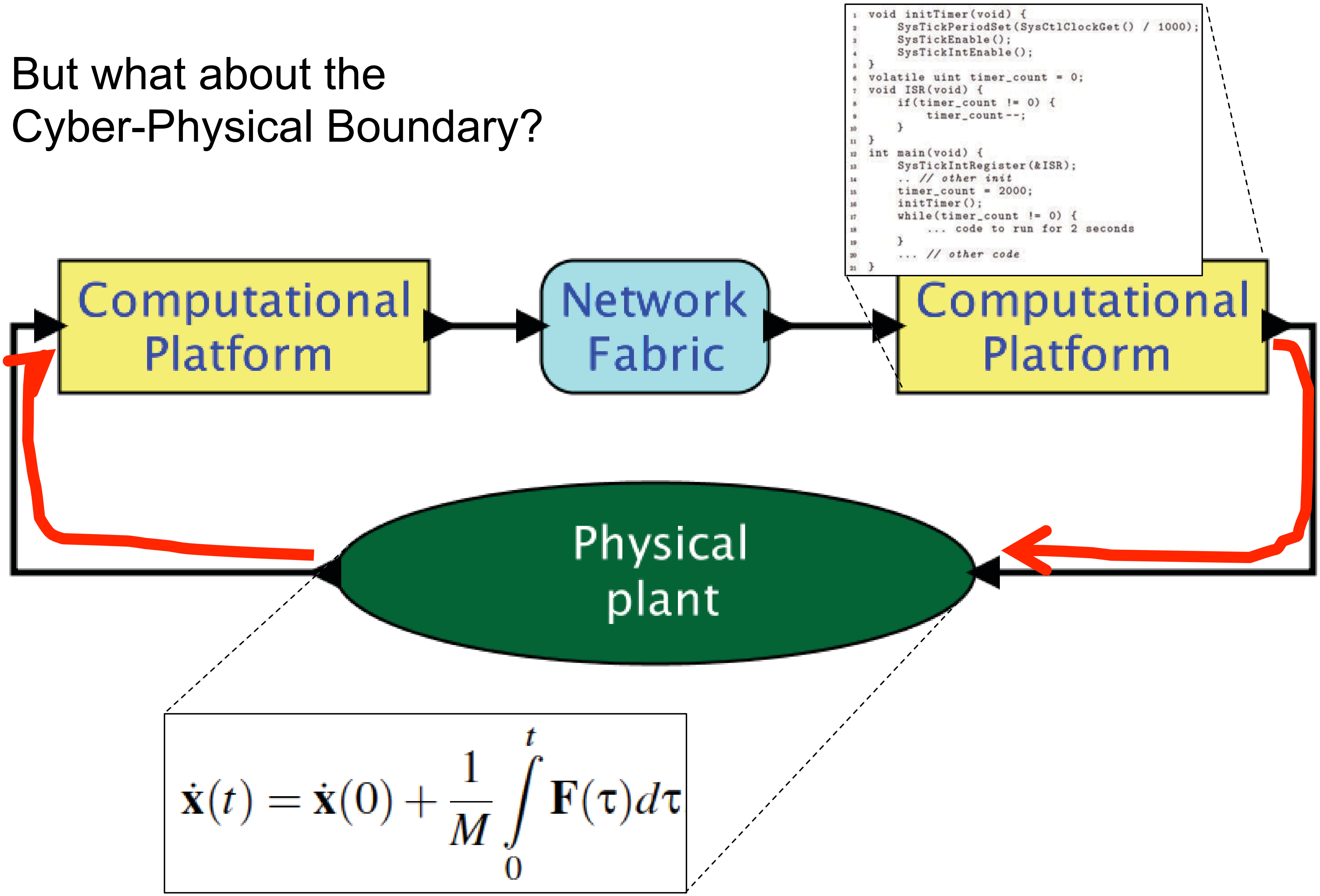
- Chapter 8:
  Discrete-Event Models
- Chapter 10:
  Modeling Timed Systems

Free download at:

http://ptolemy.org/systems



But what about the Cyber-Physical Boundary?

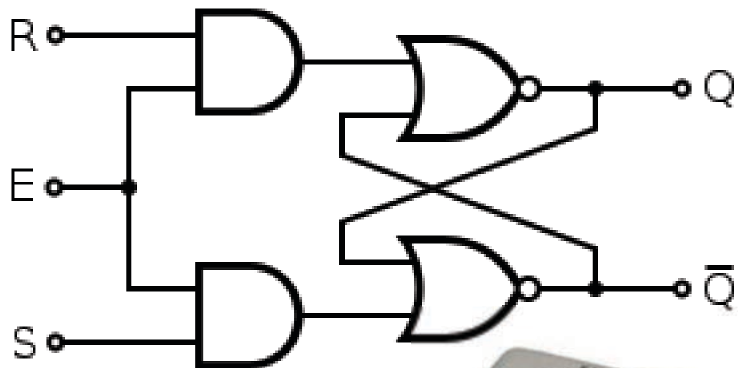


Research Efforts
Better Engineering through Better Models

- PTIDES: distributed real-time software
  - Deterministic timing of distributed CPS
- PRET machines
  - Deterministic timing at the processor level
- Accessors
  - Principled composition of networked components
- Open-source software
  - Ptolemy II
- Model-based design (iCyPhy)
  - Interfaces (e.g. FMI), contracts, aspects, …
- Semantics
  - Timed models of computation,

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]);
}
```

PRET Machines – Giving Programs the Capabilities their Hardware Already Has.

- **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

Major Challenges

and existence proofs that they can be met

- Pipelines
  - fine-grain multithreading
- Memory hierarchy
  - memory controllers with controllable latency
- I/O
  - threaded interrupts, with bounded effects on timing

Major Challenges, Yes, but Leading to Major Opportunities

- Improved determinism
- Better testability
- Reduced energy consumption
- Reduced overdesign (cost, weight)
- Improved confidence and safety
- Substitutable hardware

PRET Publications

PRET ISA Realizations:

- PRET1, Sparc-based
  - [Lickly et al., CASES, 2008]
- PTARM, ARM-based
  - [Liu et al., ICCD, 2012]
- FlexPRET, RISC-V-based
  - [Zimmer et al., RTAS, 2014]

PRET Applications:

- *Control systems*
  - *[Bui et al., RTCSA 2010]*
- *Computational fluid dynamics*
  - *[Liu et al., FCCM, 2012]*

PRET for Security:

- *Eliminating side-channel attacks*
  - *[Lie & McGrogan, Report 2009]*

PRET Memory Systems:

- *DRAM controller*
  - *[Reineke et al., CODES+ISSS 2011]*
- *Scratchpad managment*
  - *[Kim et al., RTAS, 2014]*
- *Mixed criticality DRAM controller*
  - *[Kim et al., RTAS 2015]*

PRET Principle:

- *The case for PRET*
  - *[Edwards & Lee, DAC 2007]*
- *PRET ISA extensions*
  - *[Edwards at al., ICCD 2009]*
- *Temporal isolation*
  - *[Bui et al., DAC, 2011]*
- *Design challenges*
  - *[Broman et al., ESLsyn, 2013]*
- *Cyber-physical systems*
  - *[Lee., Sensors, 2015]*

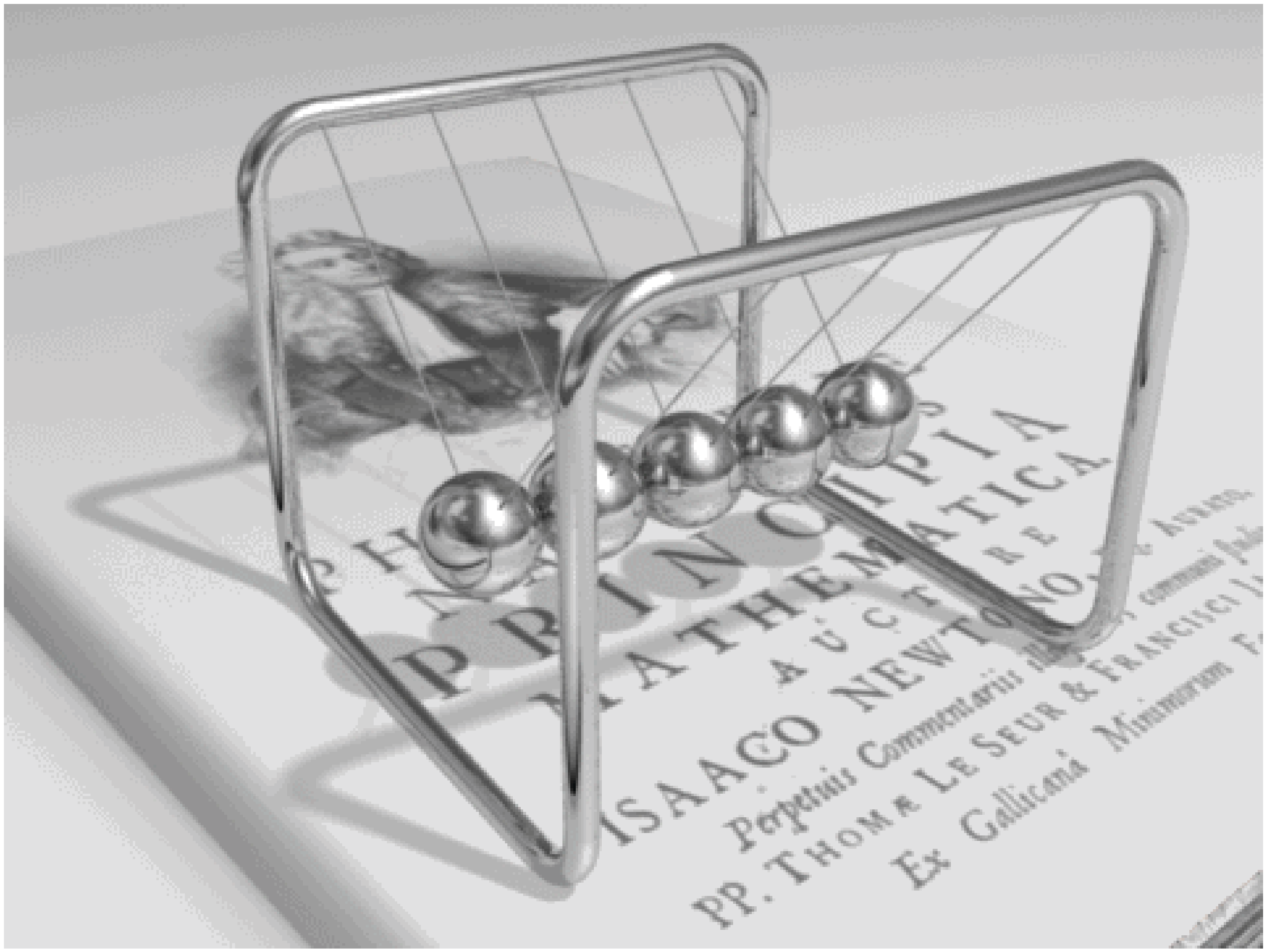
One Last Comment…
Model Fidelity

- In *science*, a good model matches well the behavior of the physical world.
- In *engineering*, a good physical implementation matches well the behavior of the model.

In engineering, model fidelity is a two-way street!

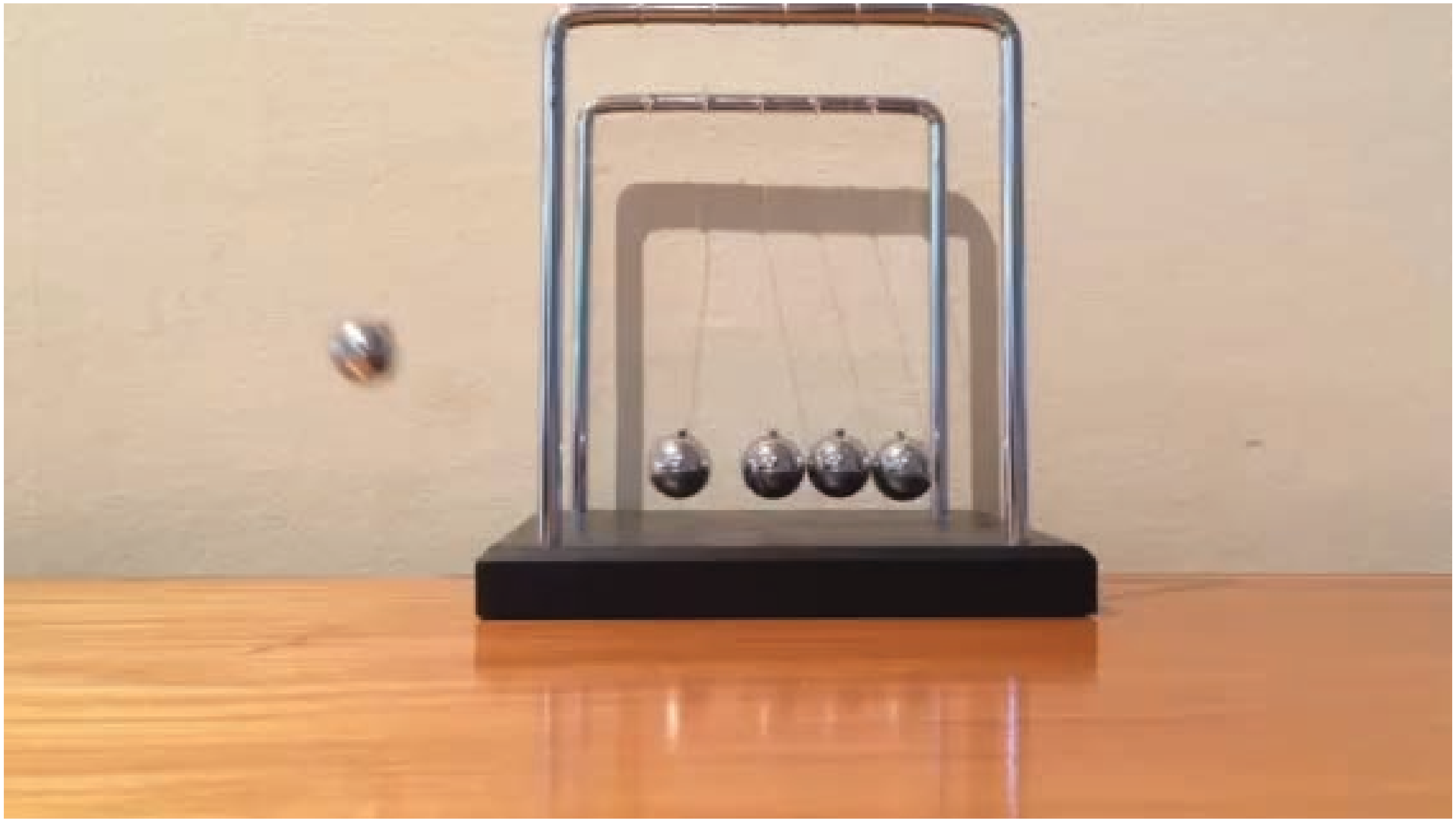
For a model to be useful, it is necessary (but not sufficient) to be able to be able to construct a faithful physical realization.

A Model



PRINCIPIA
MATHEMATICA
AUCTORE
ISAACO NEWTONO
Perpetuis Commentariis
PP. THOMÆ LE SEUR & FRANCISCI
Ex Gallicanâ Minimorum

A Physical Realization



Model Fidelity

- To a *scientist*, the model is flawed.
- To an *engineer*, the physical realization is flawed.

I'm an engineer…

Ptides and PRET offer less flawed physical realizations.

Determinism?

For a model to be useful, it is necessary (but not sufficient) to be able to be able to construct a faithful physical realization.

- The real world is highly unpredictable.
- So, are deterministic models useful?
  - Is synchronous digital logic useful?
  - Are Instruction-Set Architectures useful?
  - Single-threaded imperative programs?
  - Differential equations?

Determinism?

Deterministic models do not eliminate the need for for robust, fault-tolerant designs.

In fact, they *enable* such designs, because they make it much clearer what it means to have a fault!

Conclusions

Today, timing behavior in computers emerges from the physical realization.

Tomorrow, timing behavior will be part of the programming abstractions and their hardware realizations.

Special Thanks to:

- *Particia Derler*
- *John Eidson*
- *Slobodan Matic*
- *Hiren Patel*
- *Jan Reineke*
- *Yang Zhao*
- *Jia Zou*

See: Lee, "The Past, Present, and Future of Cyber-Physical Systems: A Focus on Models," Sensors, 15(3), February, 2015. (Open Access)

Raffaello Sanzio da Urbino – The Athens School

Image: Wikimedia Commons