Leveraging Synchronized Clocks in Cyber-Physical Systems

Edward A. Lee

Robert S. Pepper Distinguished Professor
UC Berkeley

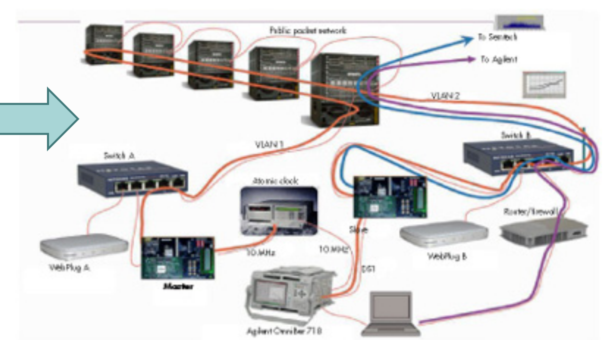
Invited Keynote Talk
WSTS 2014,
Workshop on Synchronization in Telecommunications Systems

June 11, 2014.
San Jose, CA, USA

Clock synchronization advances have happened before.

Gregorian Calendar (BBC history)

Musée d'Orsay clock (Wikimedia Commons)



2005: first IEEE 1588 plugfest

1500s
days

1800s
seconds

2000s
microseconds

Clock Synchronization Enables:

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

In this talk, I will focus on leveraging clock synchronization to provide deterministic models for cyber-physical systems.

A Cyber-Physical System Printing Press

Bosch-Rexroth

Hundreds of microcontrollers and an Ethernet network are orchestrated with precisions on the order of microseconds.

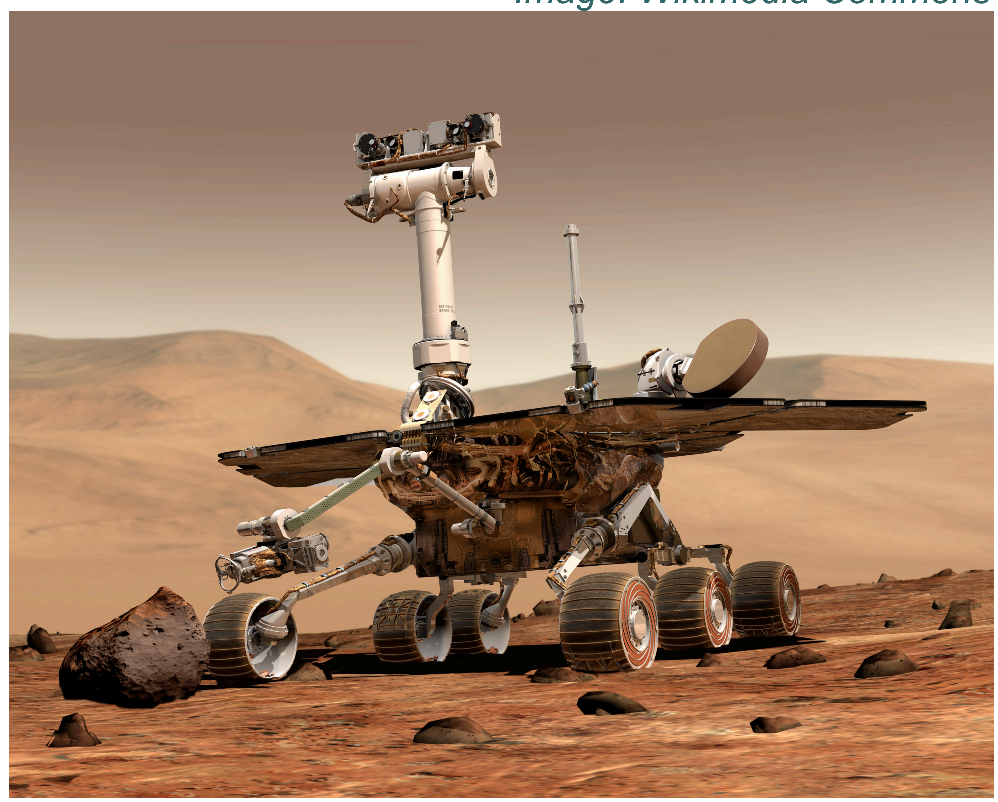
Software for such systems can be developed in a completely new way.

Clock synchronization enables tightly coordinated actions and reliable networking with bounded latency, despite using TCP/IP.

Cyber-Physical Systems

Orchestrating networked computational resources and physical systems.

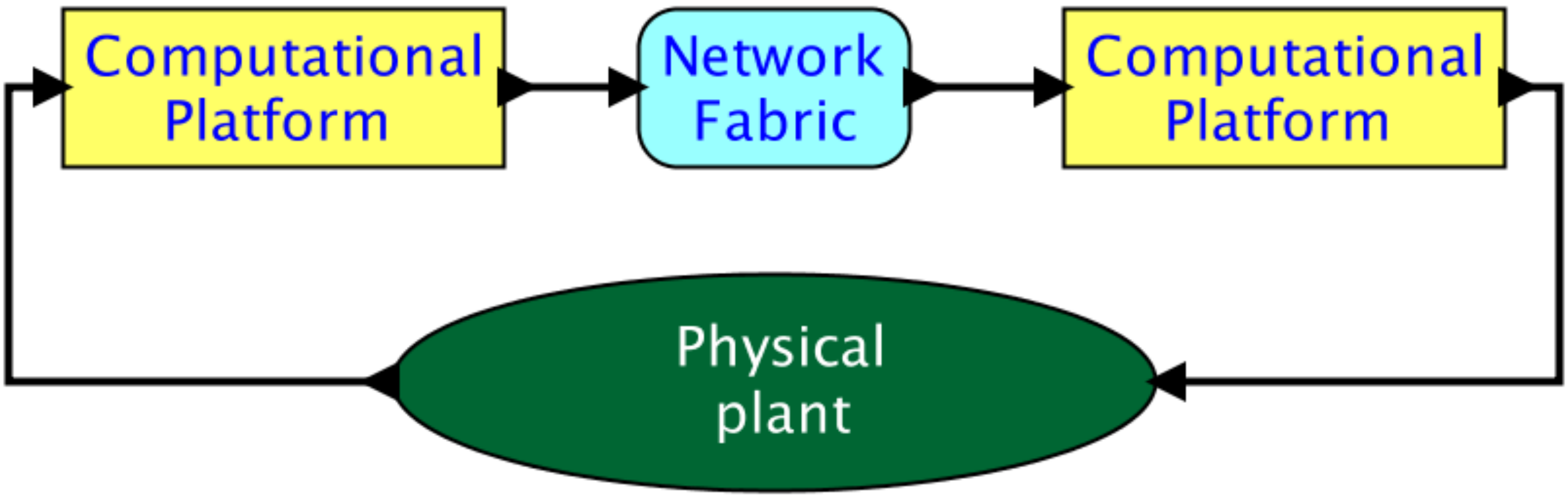
Image: Wikimedia Commons



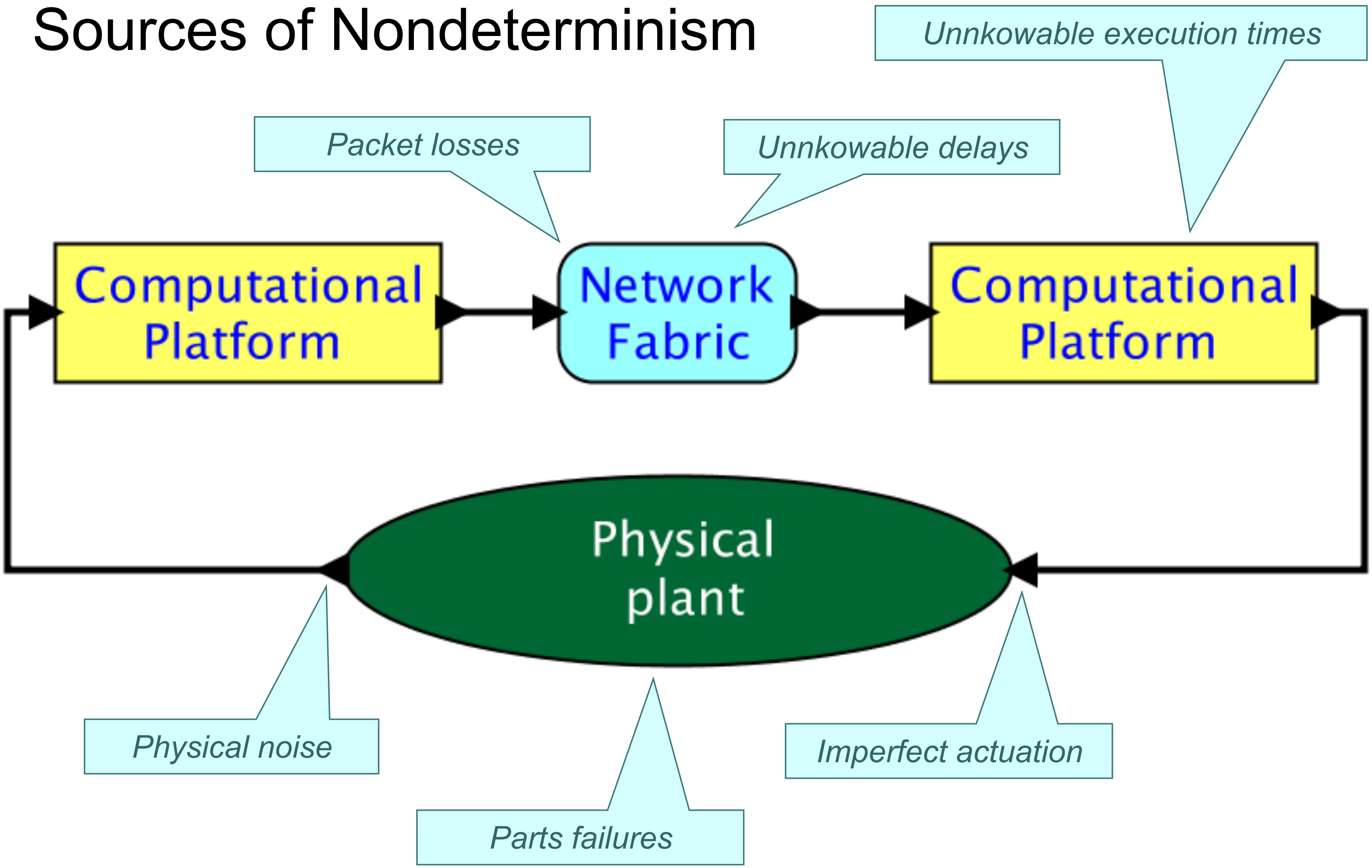
Roots:

- 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.

Schematic of a simple CPS:



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.

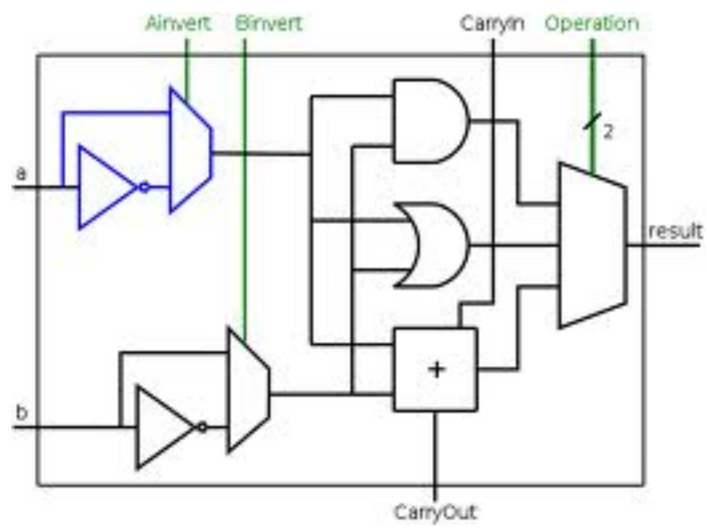
(paraphrased)

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

```
/** 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

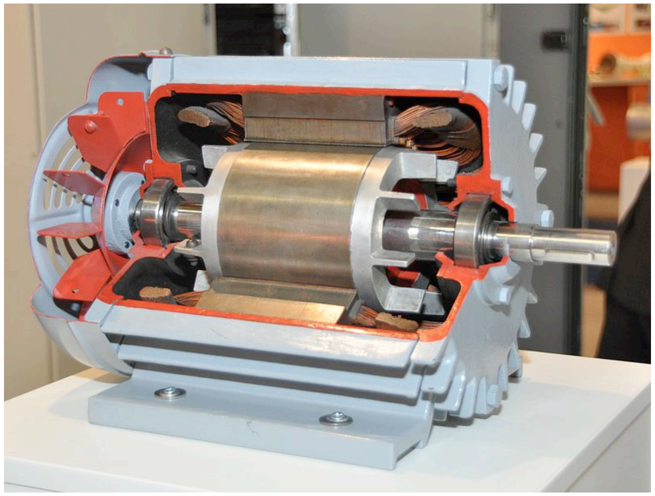


Image: Wikimedia Commons

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

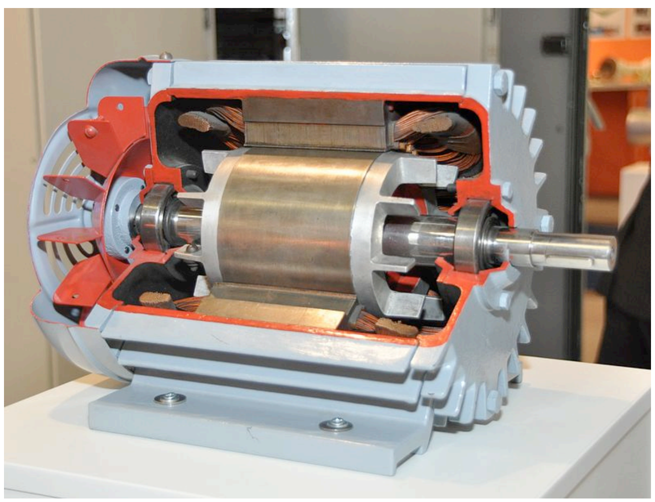
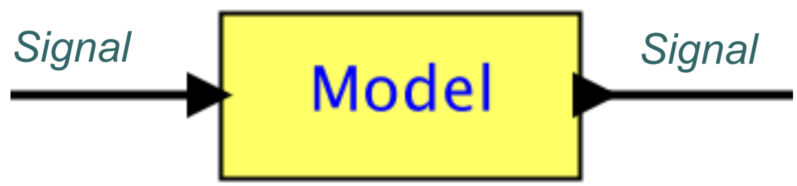


Image: Wikimedia Commons

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



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

The Model

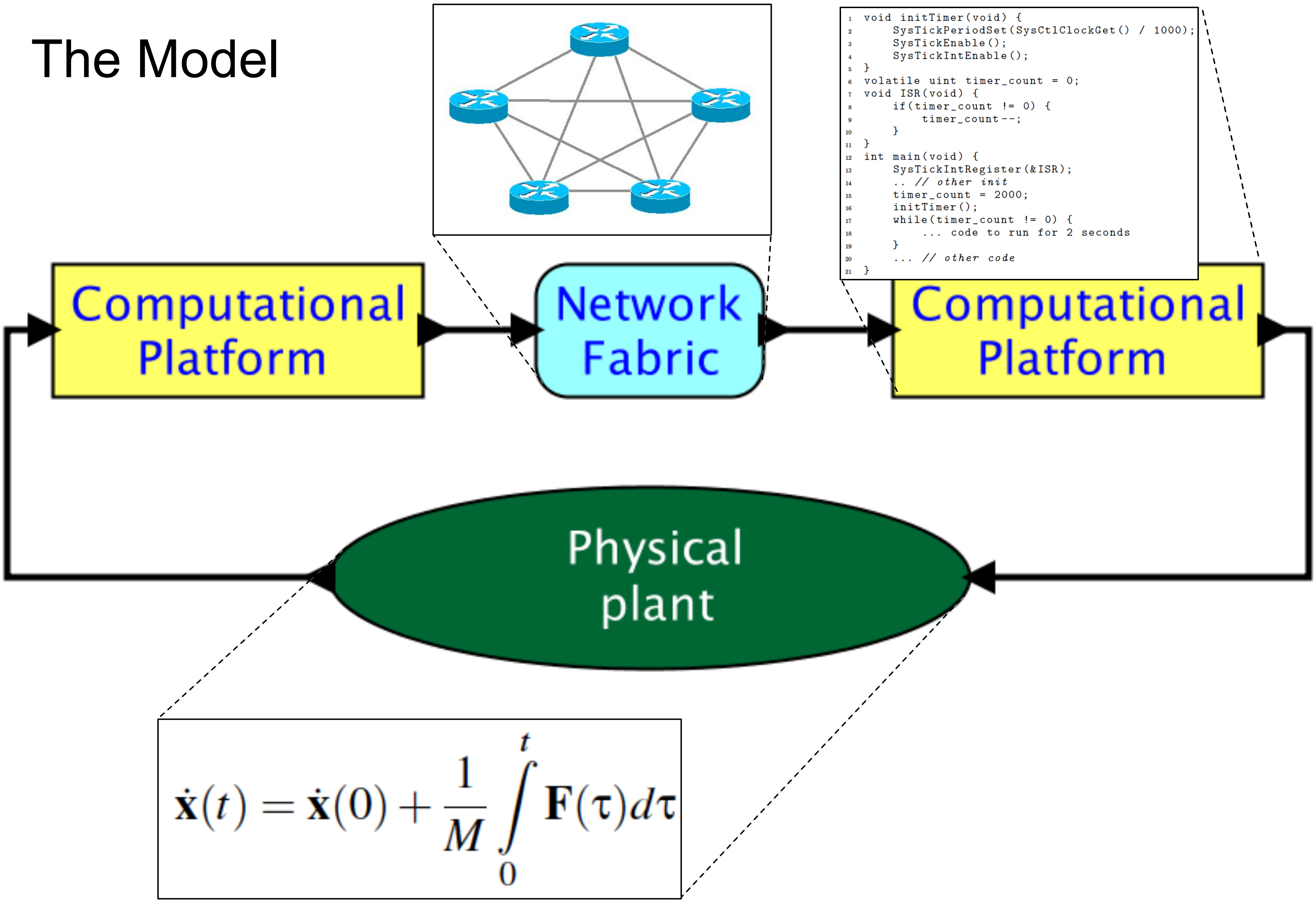
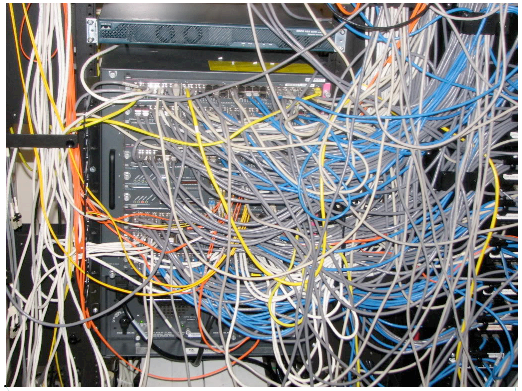
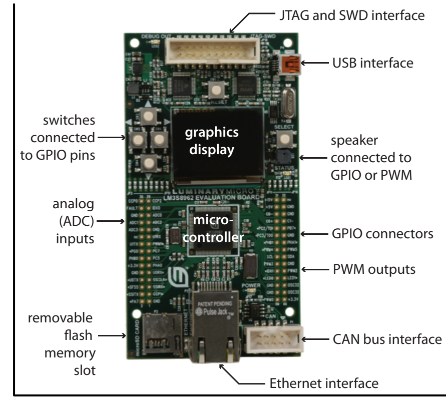


Image: Wikimedia Commons

The Reality





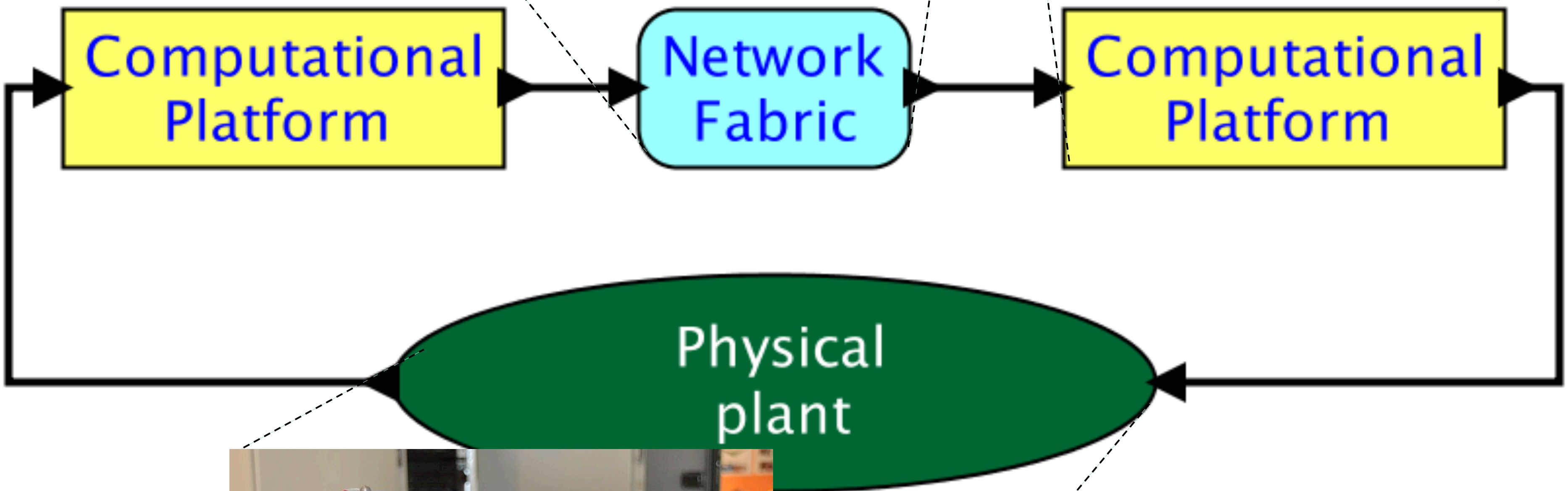
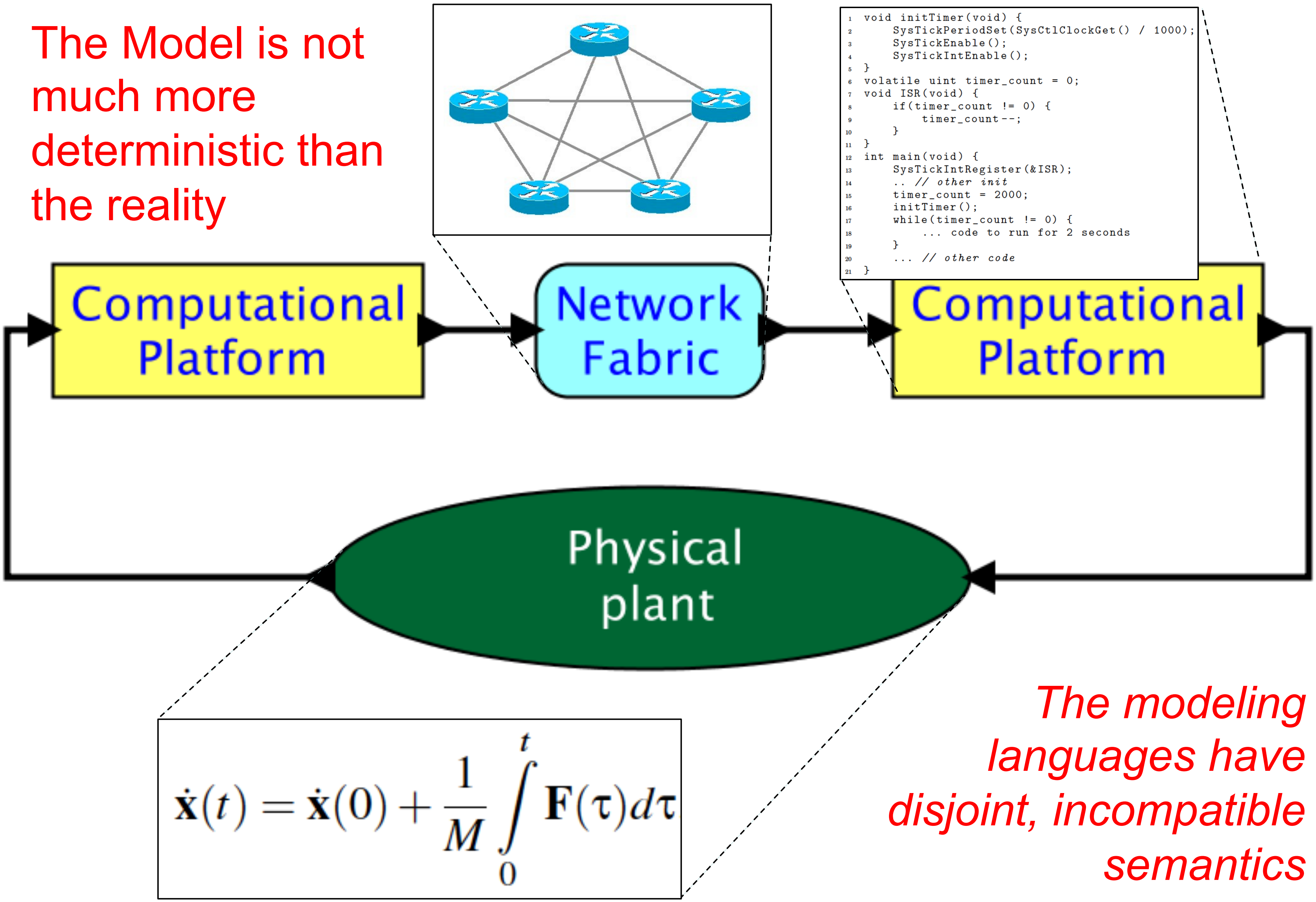


Image: Wikimedia Commons

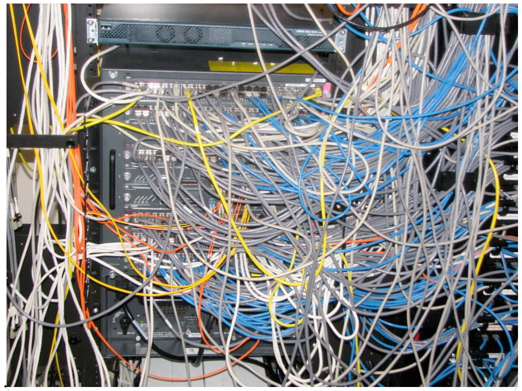
The Model is not much more deterministic than the reality

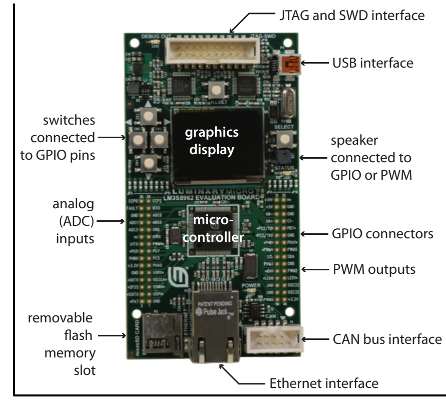


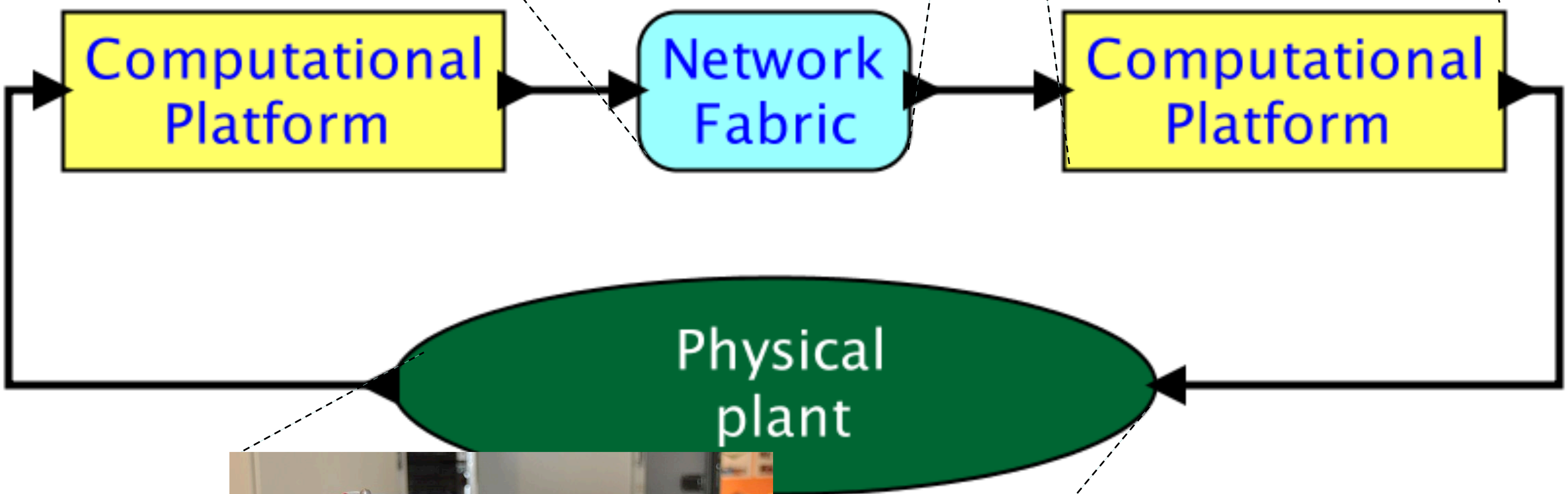
The modeling languages have disjoint, incompatible semantics

Image: Wikimedia Commons

System dynamics emerges from the physical realization





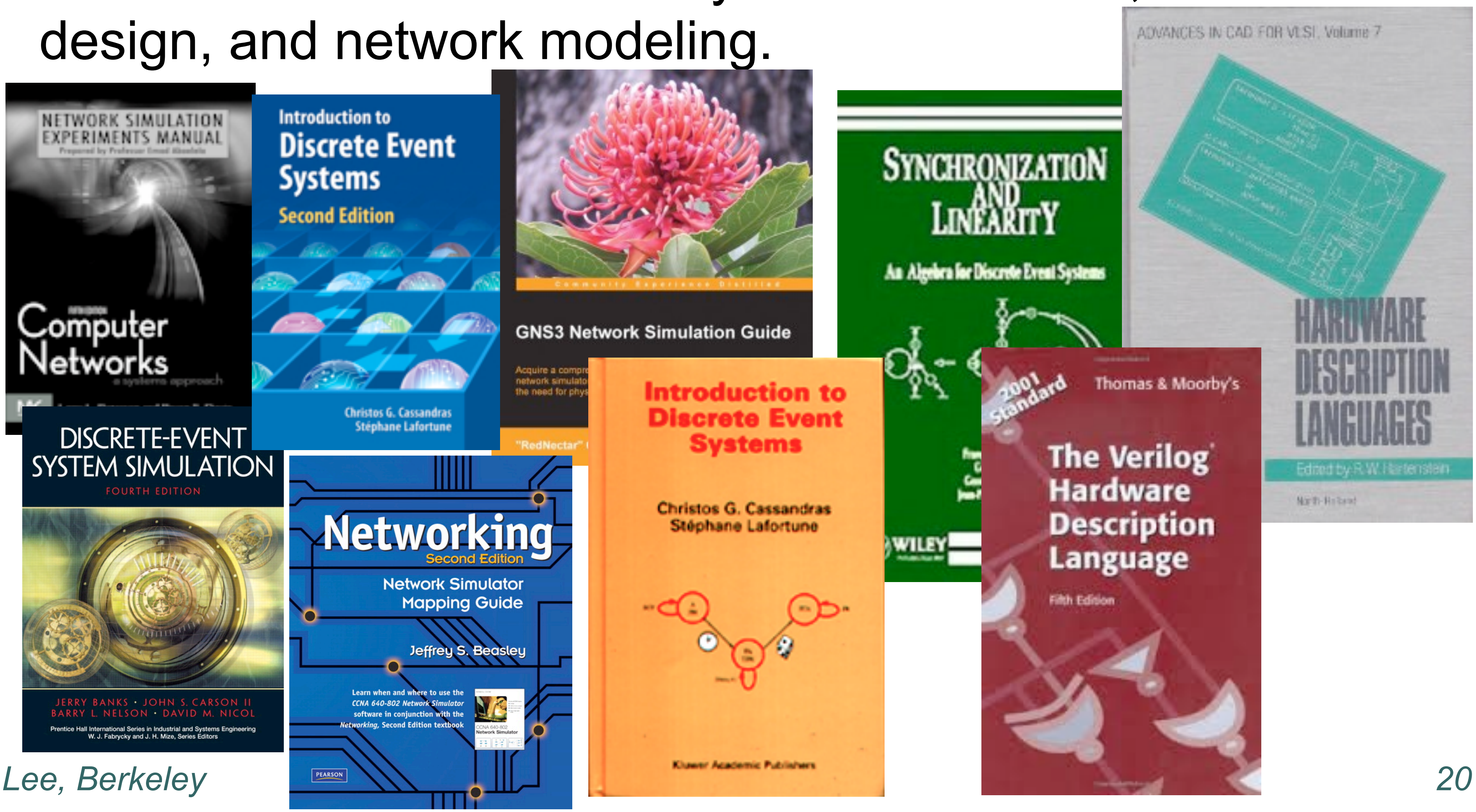


… leading to a "prototype and test" style of design

Image: Wikimedia Commons

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

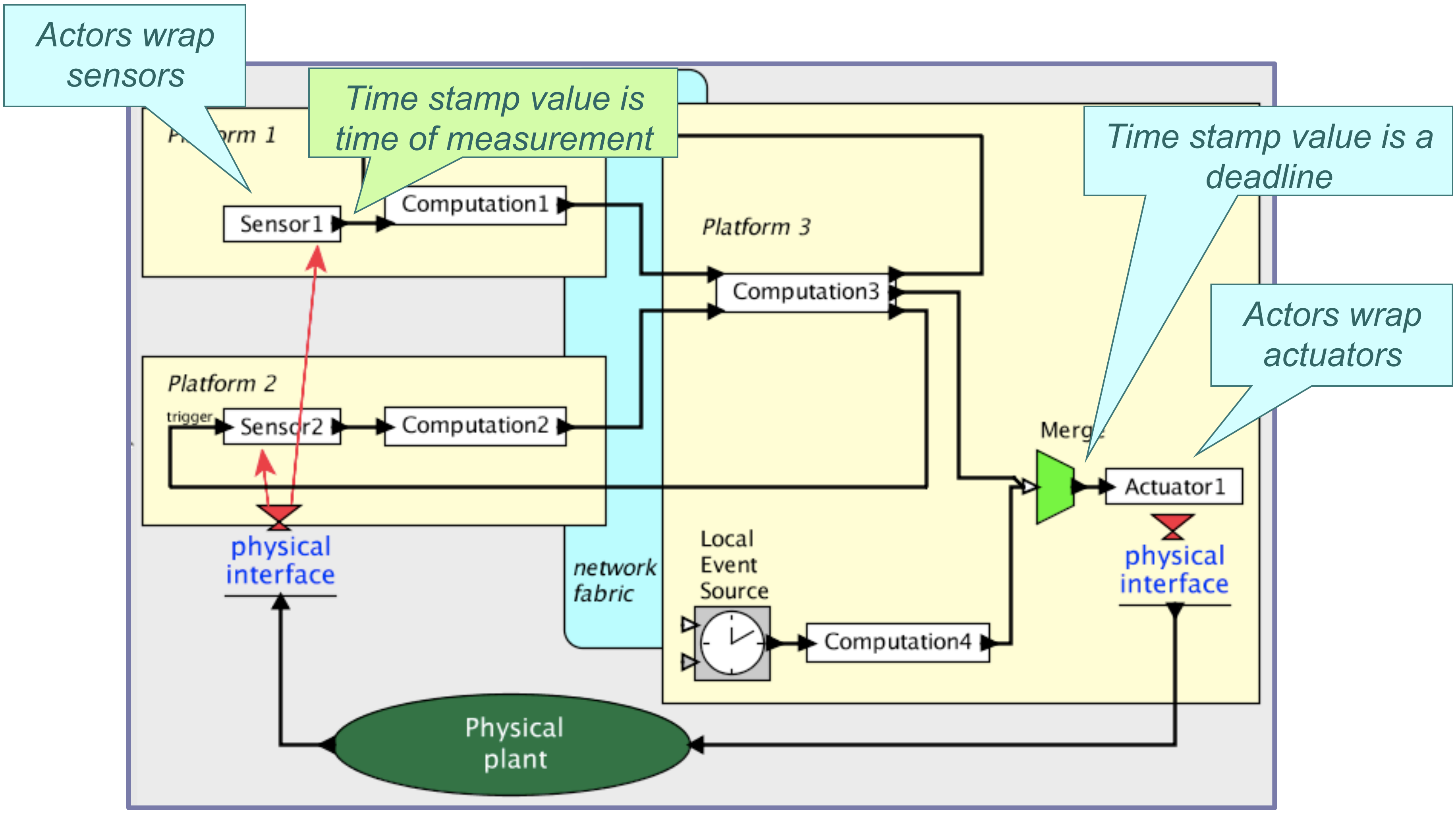
- DE is usually used for simulation (HDLs, network simulators, …)
- Distributing DE is done to accelerate simulation.

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

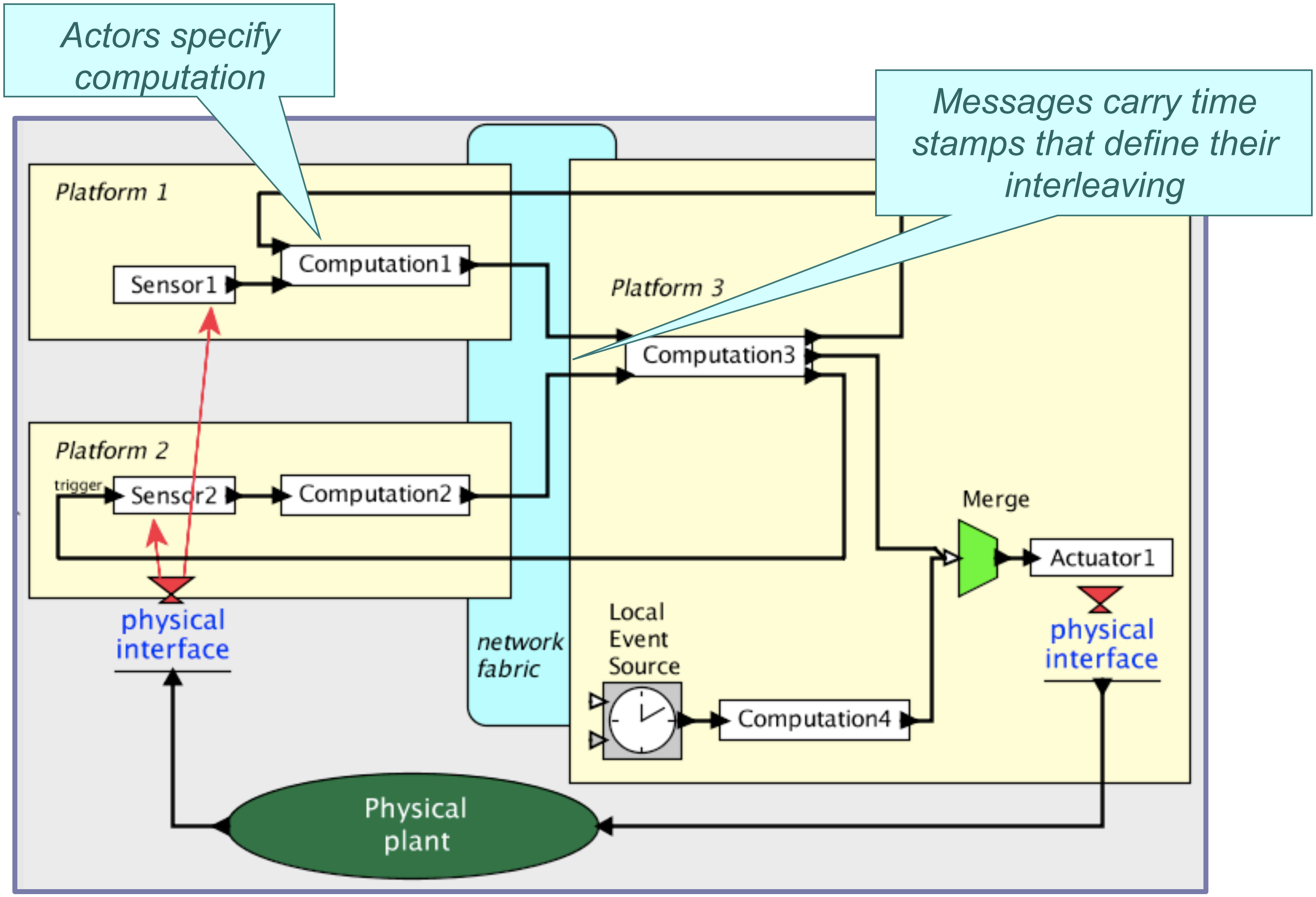
- ***PTIDES***: Programming Temporally Integrated Distributed Embedded Systems

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

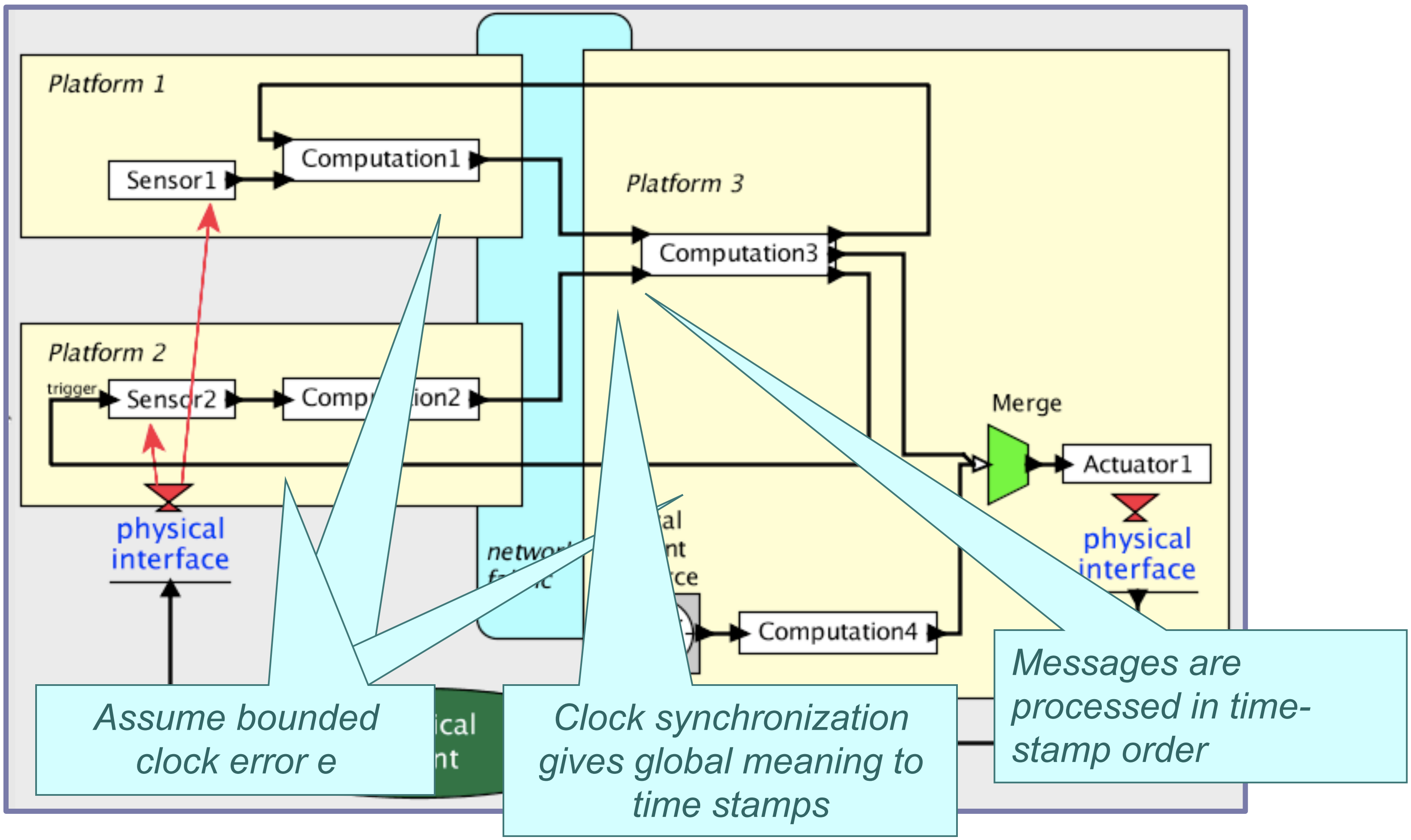


Ptides: Second step: Time-stamped messages.



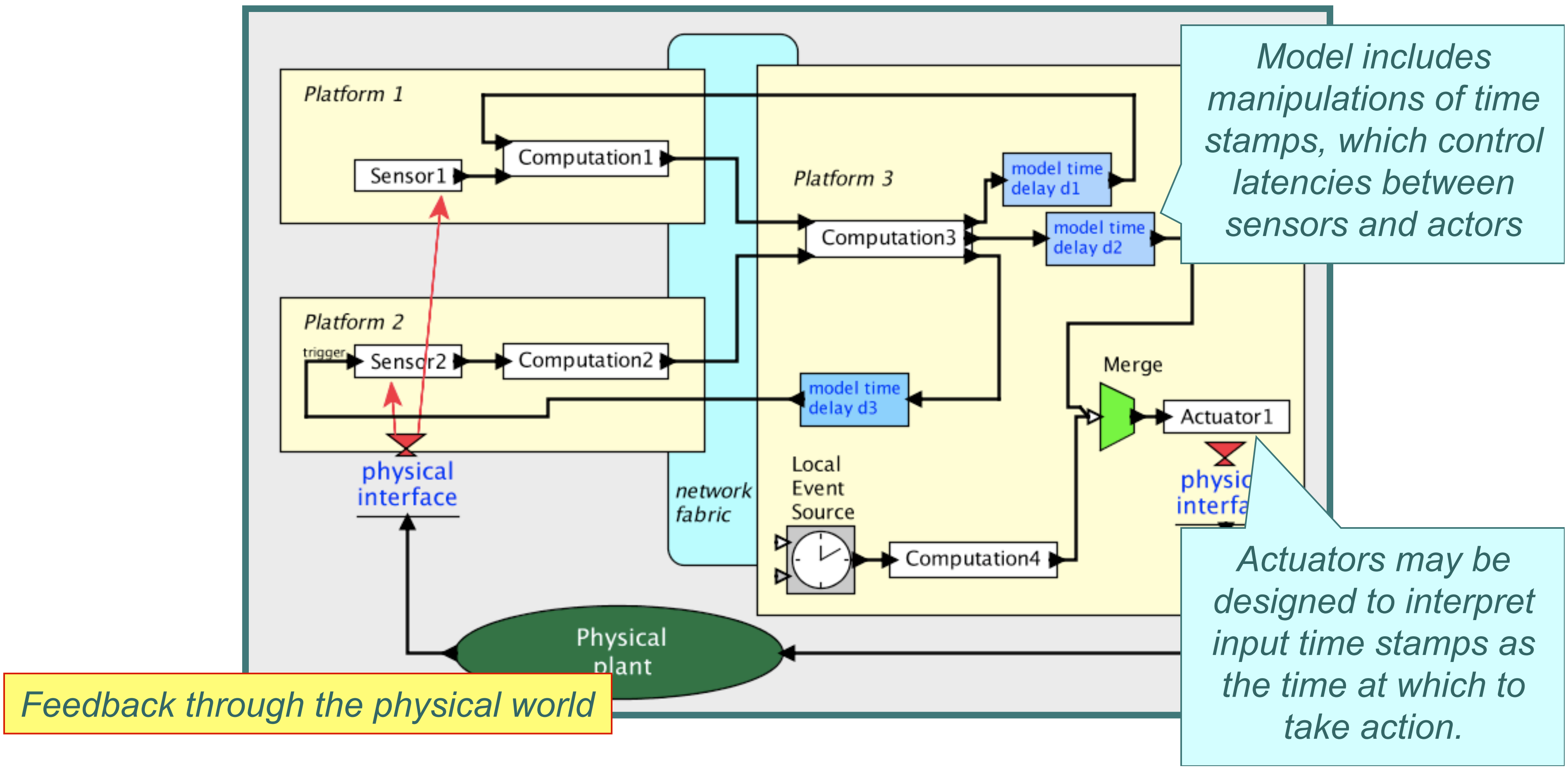
Ptides: Third step: Network clock synchronization

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



Ptides: Fourth step:
Specify latencies in the model

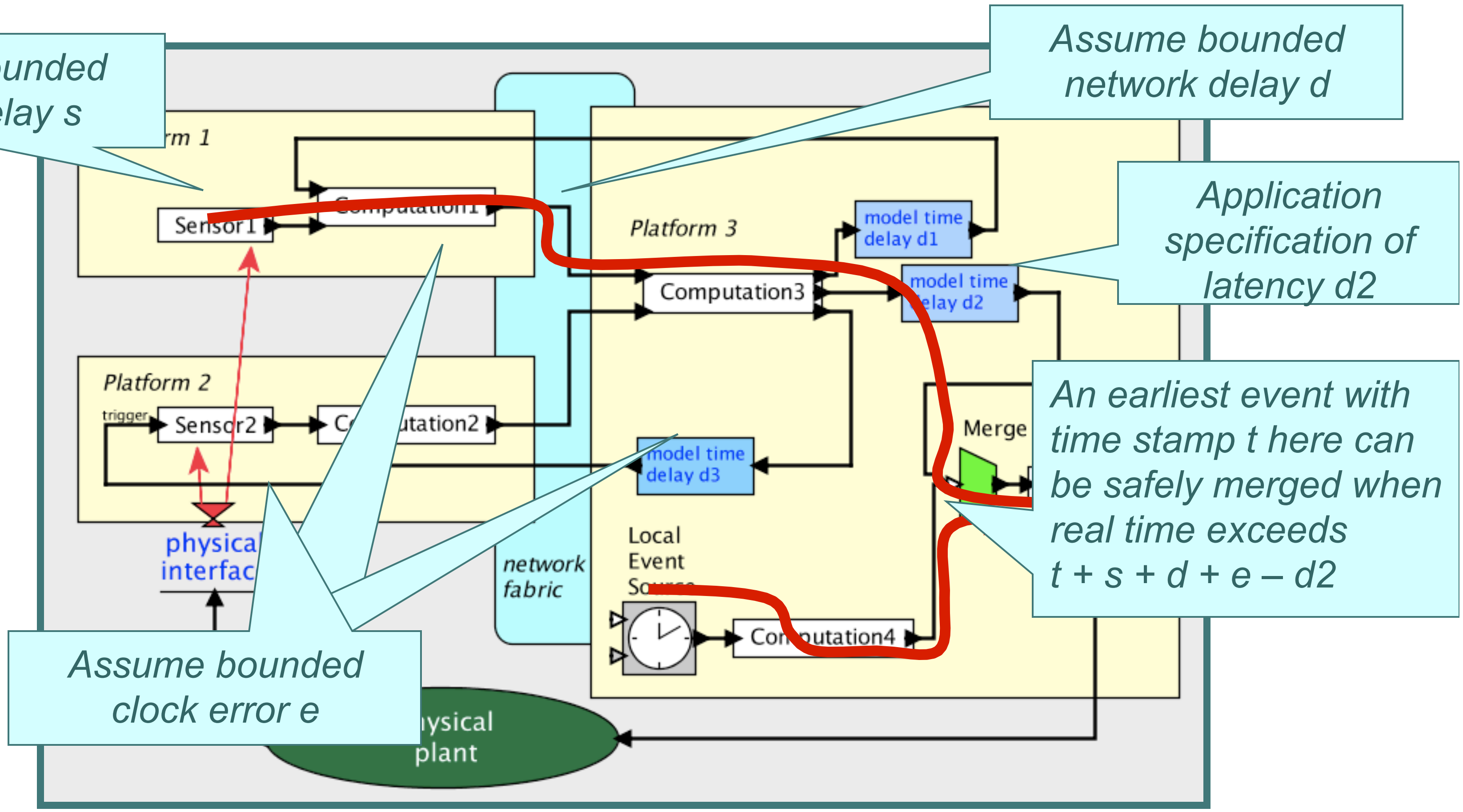
Global latencies between sensors and actuators become controllable, which enables analysis of system dynamics.



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!

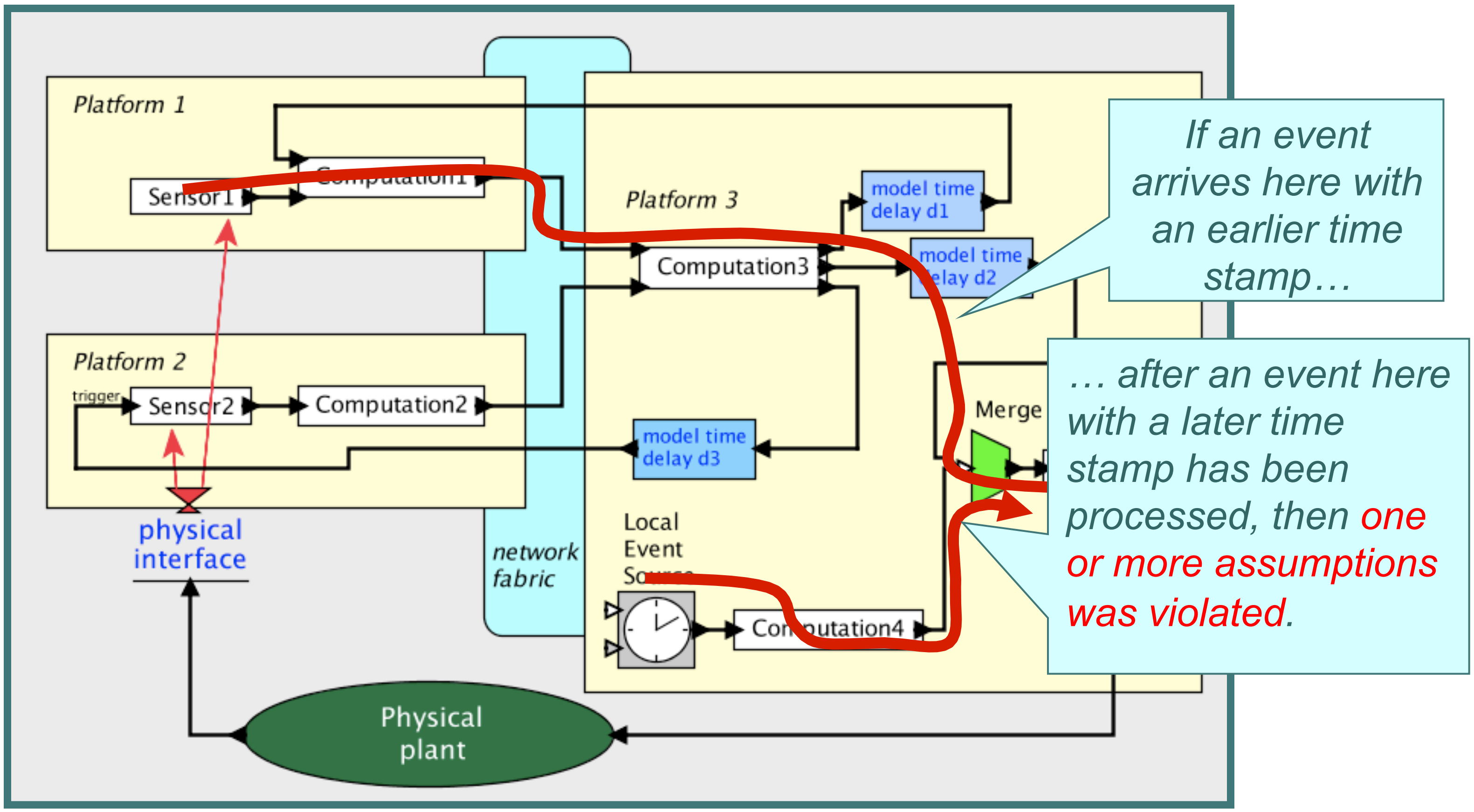
*All of the assumptions are achievable with today's technology, and in fact are **requirements** anyway for hard-real-time systems. The Ptides model makes the assumptions explicit.*

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

Handling Faults

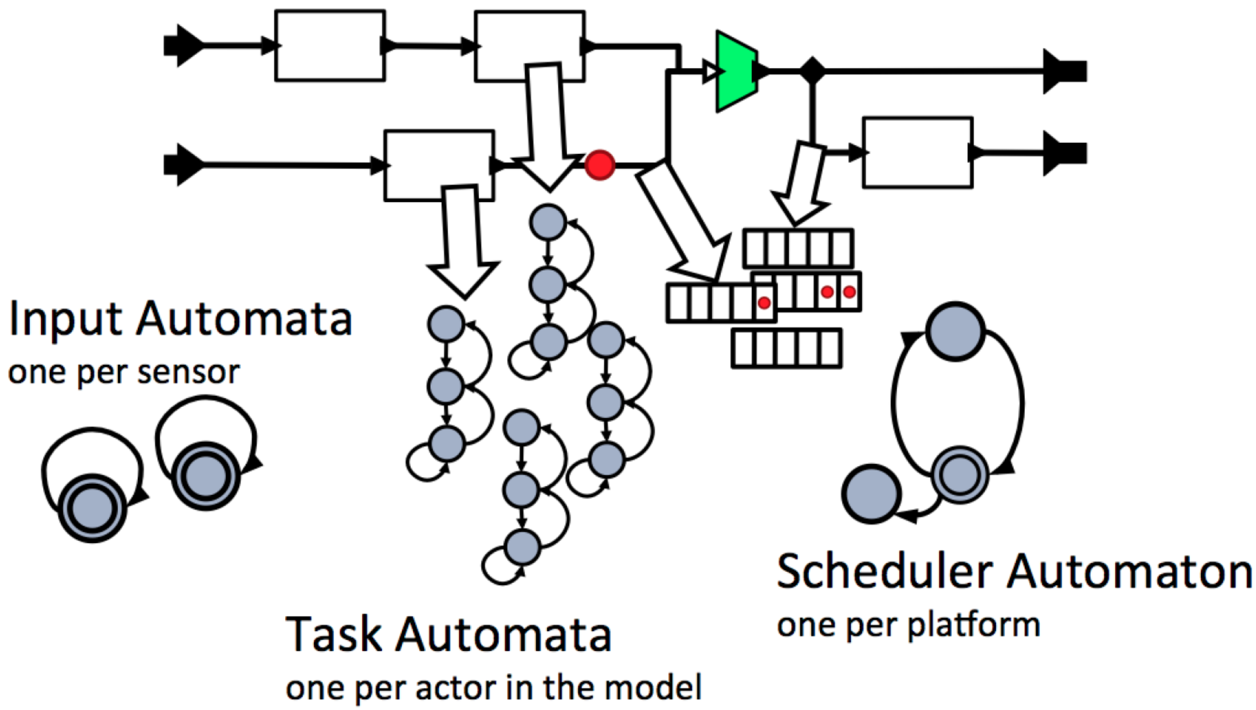
A “fault” is a violation of assumptions in the model.

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



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.

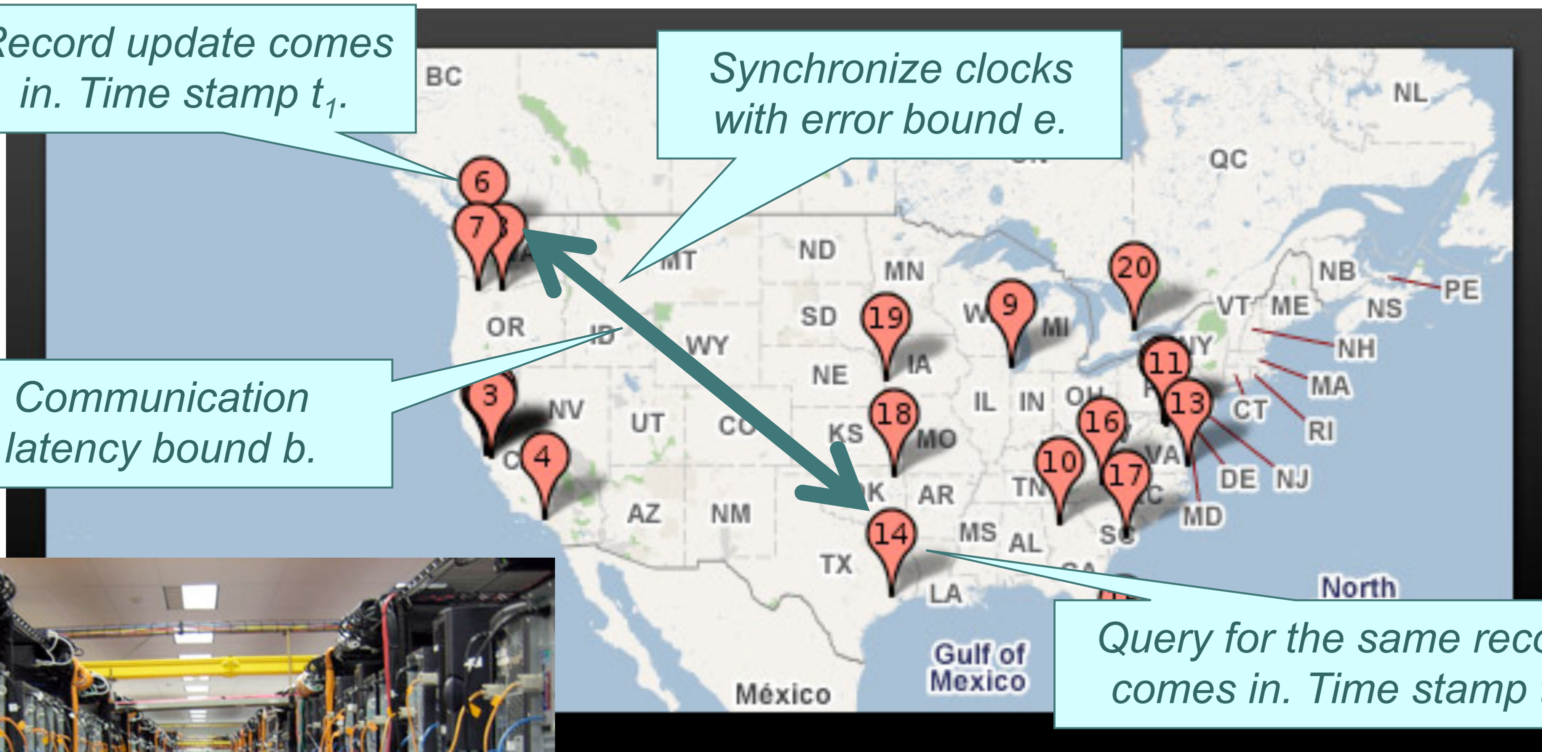
Google Spanner

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.

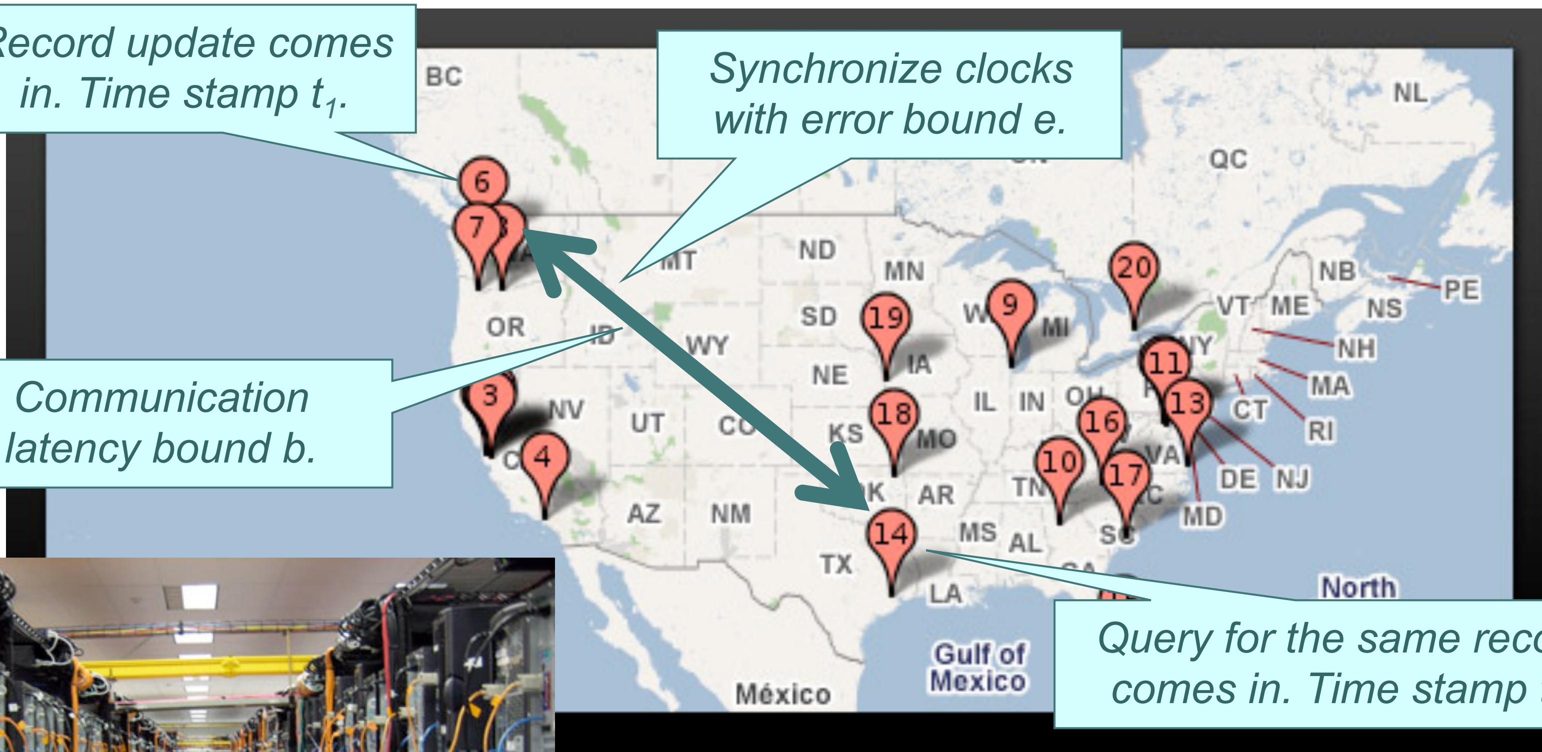
Synchronize clocks with error bound e.

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.

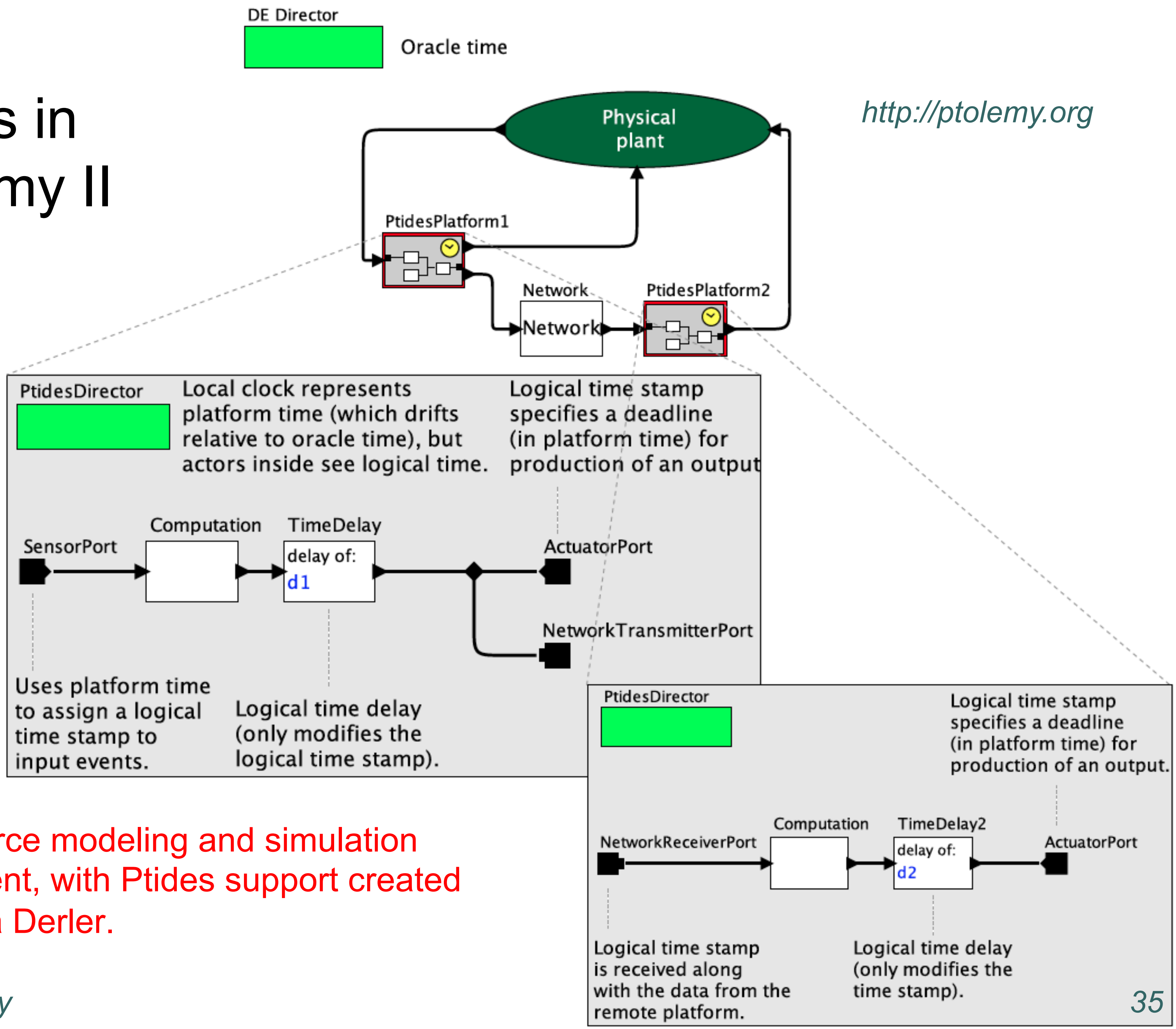
Communication latency bound b.

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.

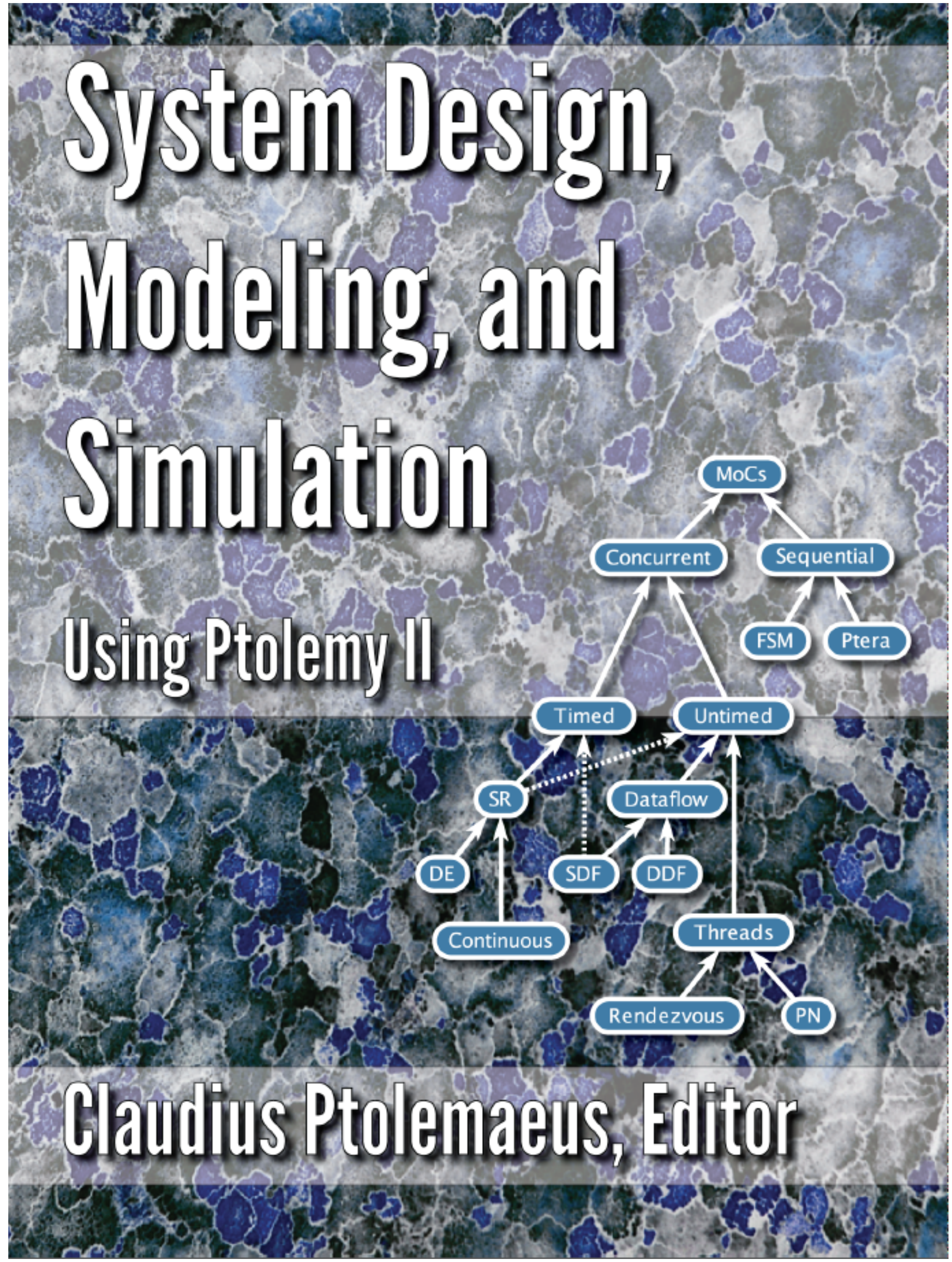
See Book

See

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

Free download at:

http://ptolemy.org/systems



Ptides is a Change in Philosophy

The implementation architecture (processors, networks, software) affects the behavior of any cyber-physical 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

Conclusion

Today, timing behavior in programs and networks emerges from the physical realization.

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

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