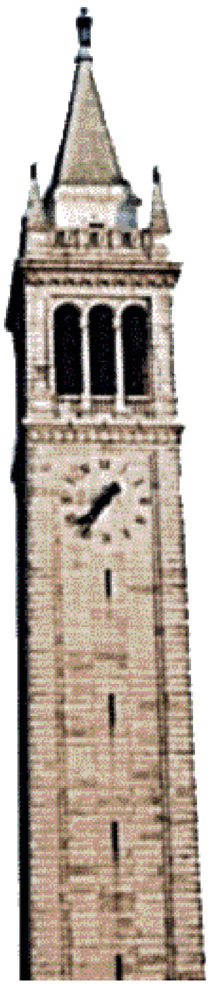


The Internet of *Important* Things

Edward A. Lee

Robert S. Pepper Distinguished Professor
UC Berkeley

Keynote

Software Engineering and Formal Methods (SEFM)
September 11, 2015.
York, England, UK

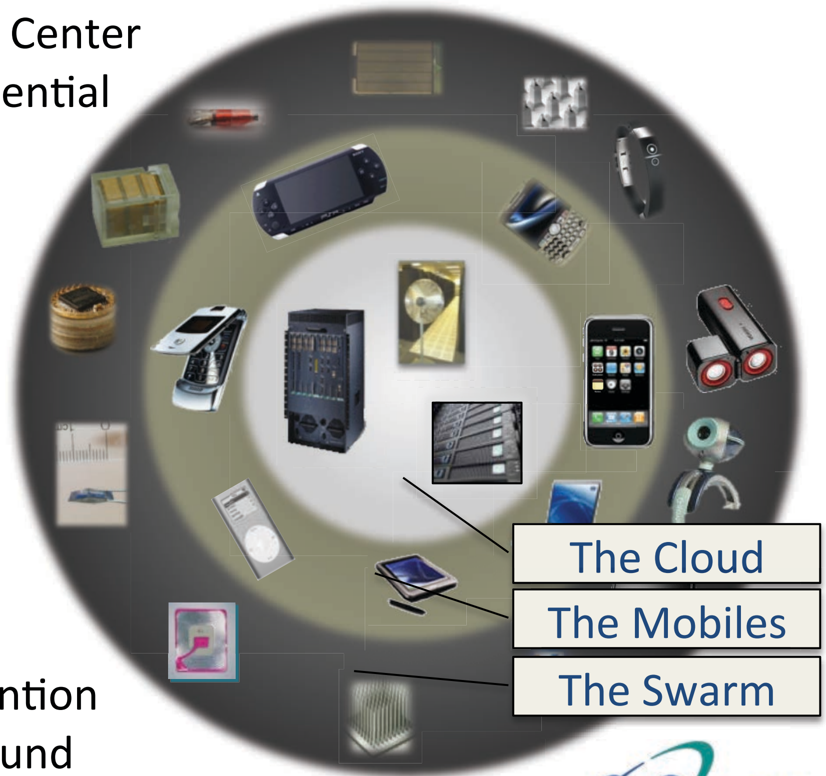
The TerraSwarm Research Center 2013-2018

What it is:

The TerraSwarm Research Center is addressing the huge potential (and associated risks) of pervasive integration of smart, networked sensors and actuators into our connected world.

The Goal

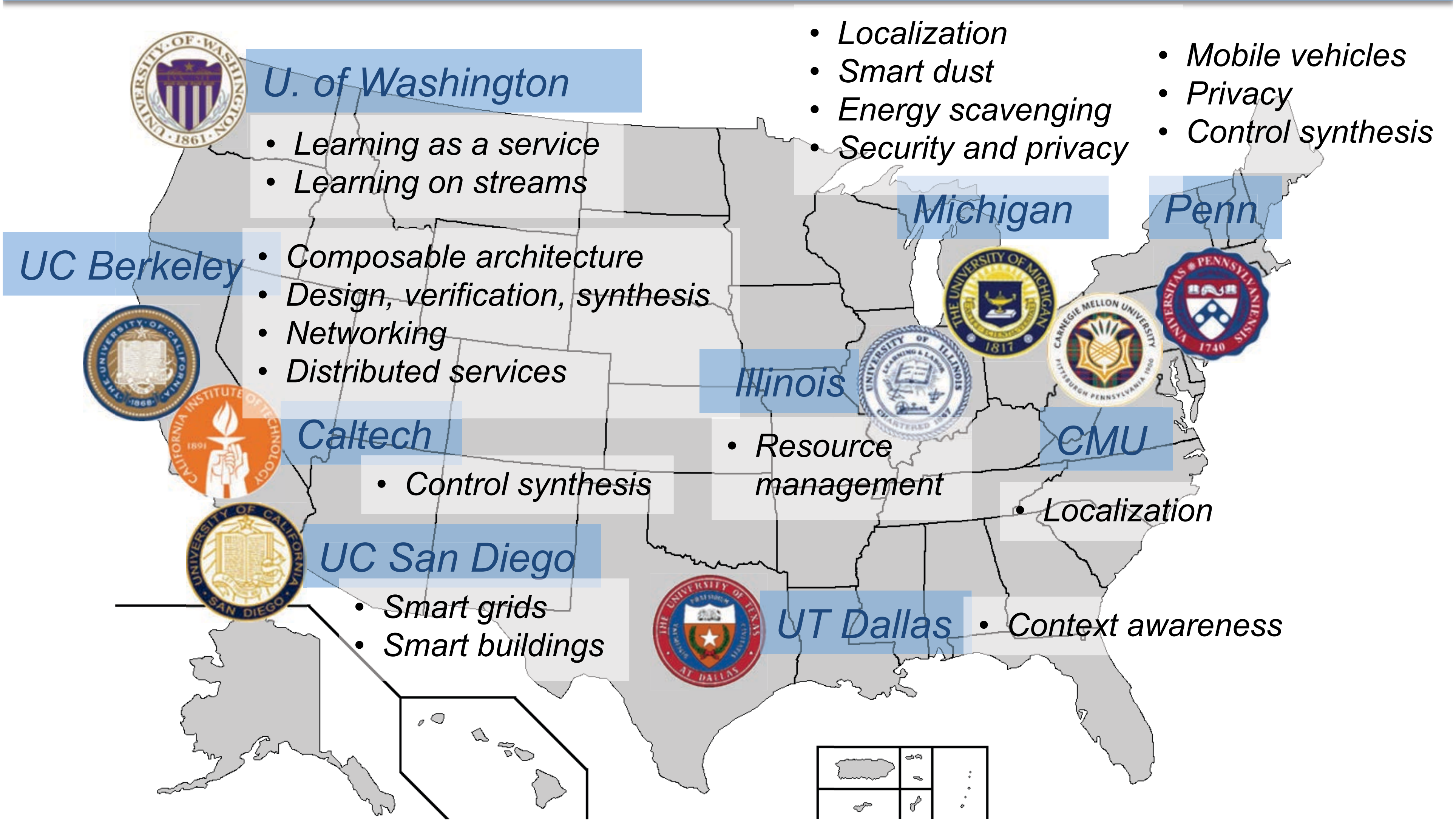
To lead the world in development of the platforms, methodologies, and tools that enable invention of creative, secure, and sound applications using networked sensors and actuators.



The Sponsors:

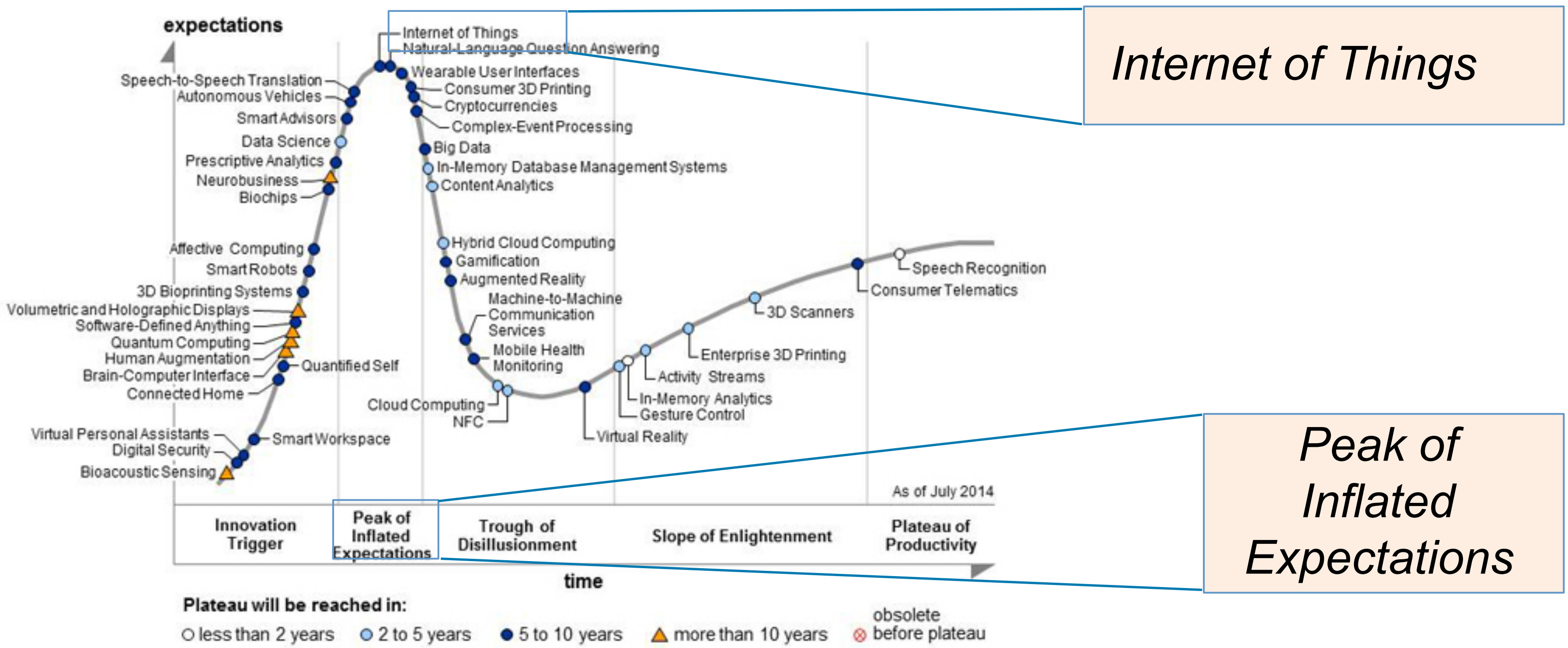
DARPA, GLOBALFOUNDRIES, IBM, intel, Micron, Raytheon, Texas Instruments, United Technologies

TerraSwarm Sites



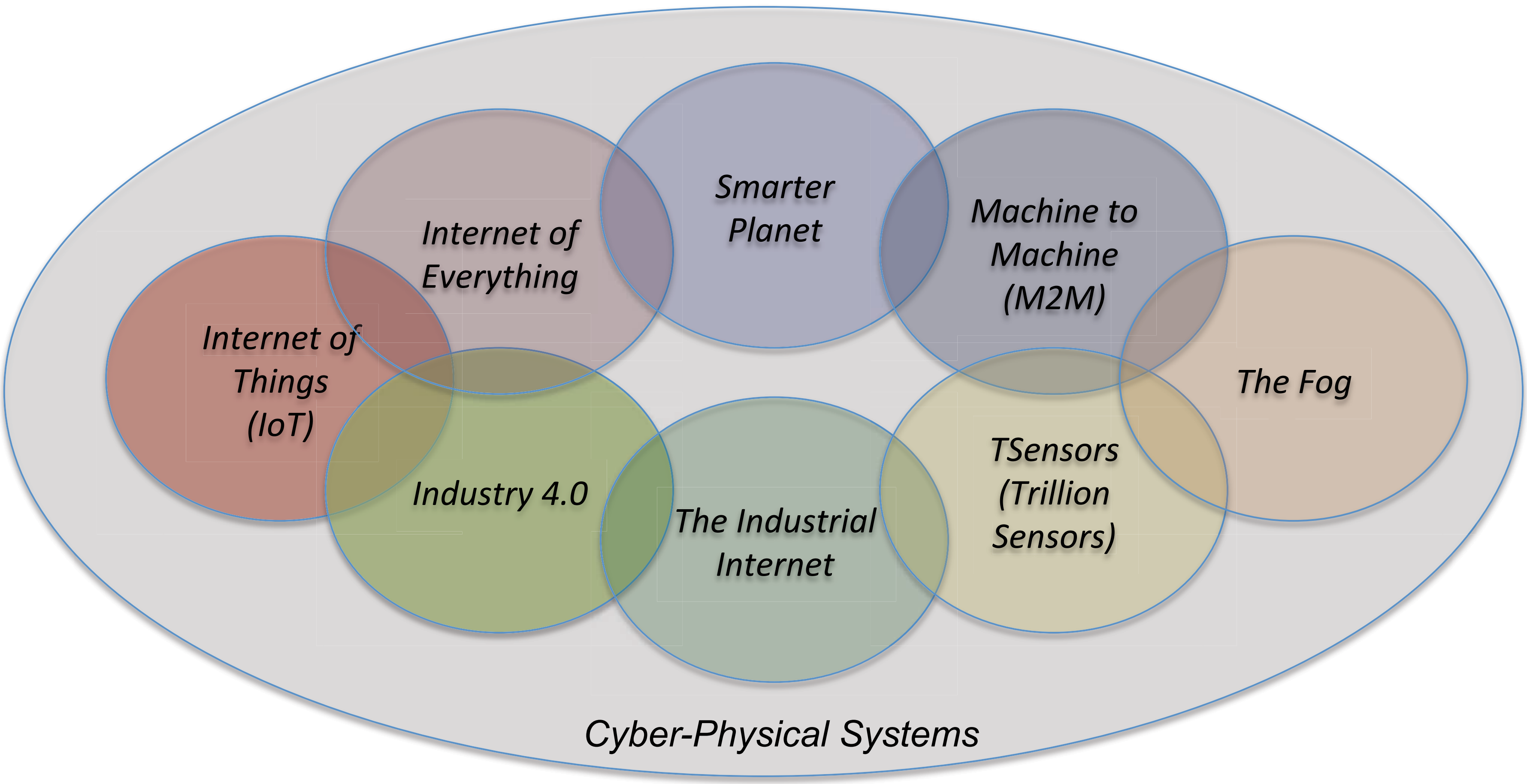
Buzzword du jour: The Internet of Things

Using Internet technology to interact with physical devices (“things”).



http://www.gartner.com/technology/research/hype-cycles/

… but the idea has been around for a while…



Our Focus: Cyber-Physical Systems (CPS) The Internet of *Important* Things (IoIT)

Using Internet technology to interact with physical devices ("things").

We are interested in systems where safety and reliability loom large.

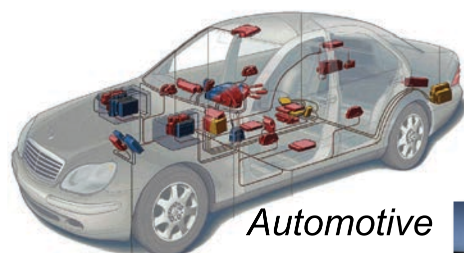
This Bosch Rexroth printing press is a cyber-physical factory using Ethernet and TCP/IP with high-precision clock synchronization (IEEE 1588) on an isolated LAN.

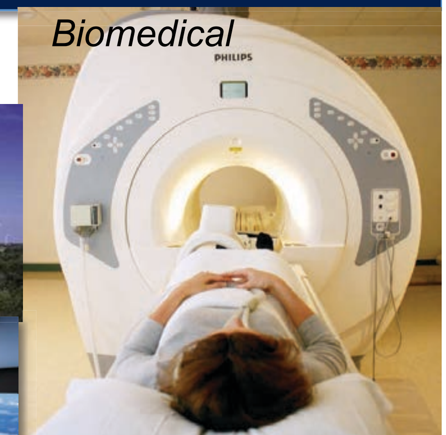
IoIT and CPS

Underlie much of the industrial economy

It’s not just information technology anymore:

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

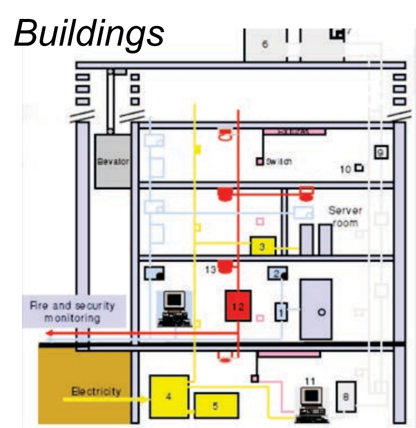




Contradictions:

- 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



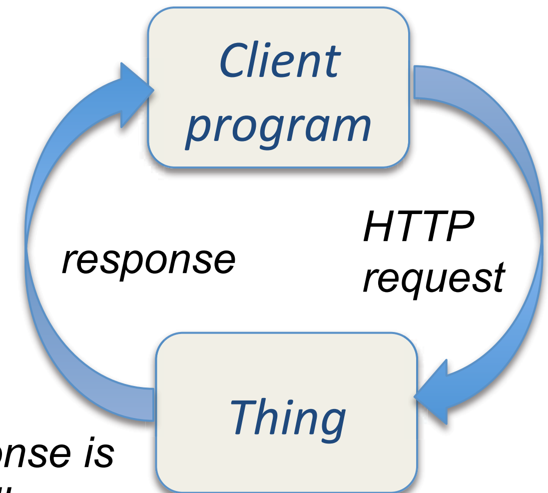
Innovation:

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.

A Common IoT Design Pattern: REST with AACs

A RESTful service [Fielding & Taylor 2002] is accessed using a design pattern common on the web that we call *Asynchronous Atomic Callbacks* (AAC).

In the Web, AAC is widely used. It is central to many popular internet programming frameworks such as Node.js & Vert.x, and to CPS frameworks such as TinyOS.



Response is typically asynchronous to avoid blocking the client program.

URL encodes all state info (credentials, commands, etc.)

Response handler executes atomically.

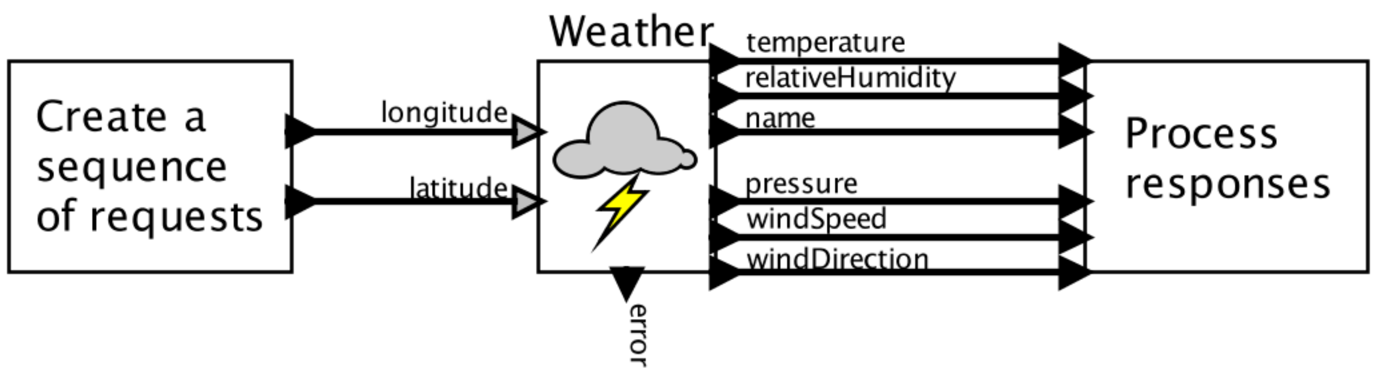
Example in JavaScript

```
// Import a module providing network services
var http = require("http”);
// Construct a URL encoding a request
var url = "http://foo.com/deviceID/...";
// Issue the request and provide a callback
http.get(url, function(response) {
    // ... handle the response ...
});
```

The callback function will be called atomically some time later when the server response arrives.

Another Common Design Pattern: *Actors*

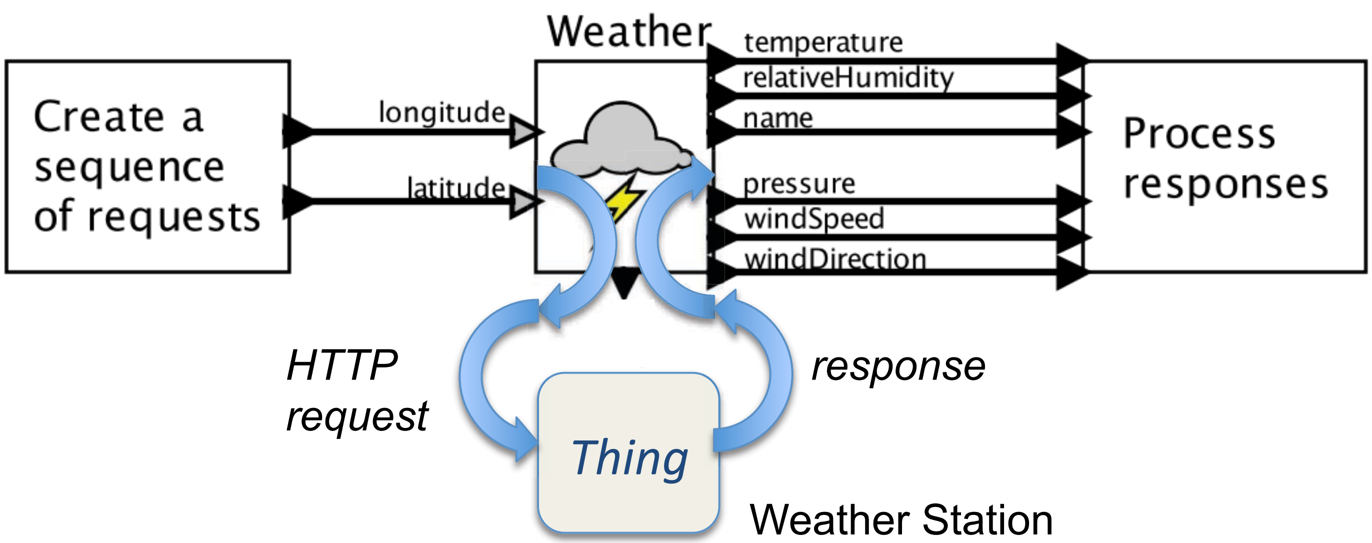
Streaming requests:



Sequence of requests for a service (a *stream*) triggers a sequence of responses.

Actors and AAC

Streaming requests:



But the responses may not come back in the same order as the requests!

This is a rudimentary *timing problem.*

Timing Problems Loom Large in The Internet of *Important* Things (IoIT)

The order and timing of events matters a lot when interacting with physical processes.

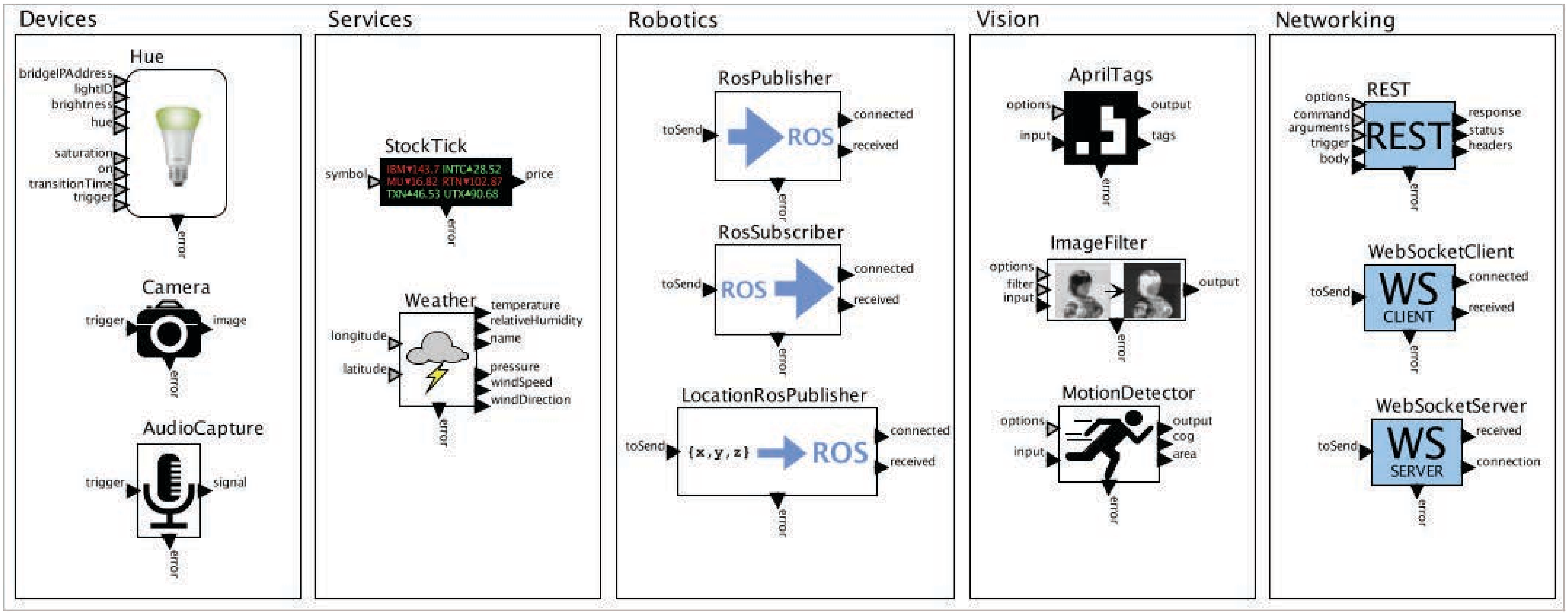
The system at the right orchestrates hundreds of microcontrollers to deposit ink on paper flying through the printer at 100 kmh with micron precision.

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.

Lee, Berkeley

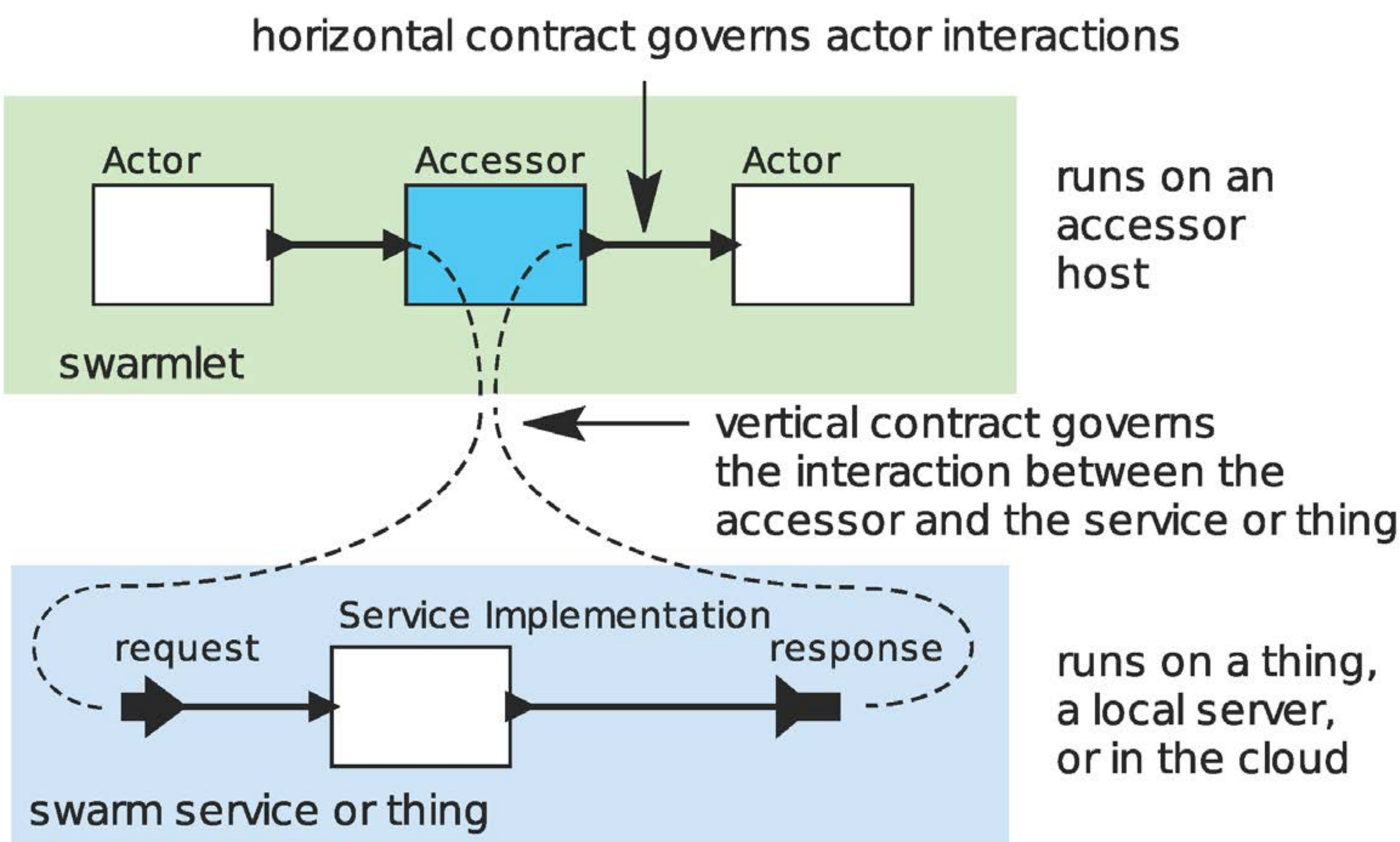
Accessors

Actor-oriented wrappers for networked devices and services. Some examples:



Accessors

Bridging actors with networked access



E.g. time-stamped events processed in time-stamp order (a discrete-event (DE) model of computation (MoC)).

E.g. asynchronous atomic callbacks (AAC).

Formal Methods

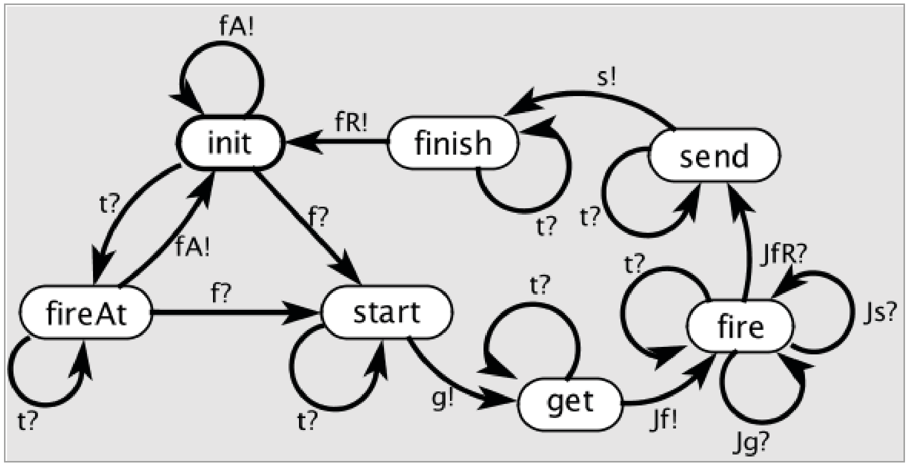
- Semantics of timed systems
  - Fixed point theorems, contractions, causality, and constructiveness on generalized ultrametric semilattices [Matsikoudis & Lee, 2013/14/15].

- Interface theories
  - Type systems, behavioral interfaces, and concurrent composition [Lohstroh & Lee, 2015, Lee & Xiong 2003].

Behavioral Interfaces to Codify Contracts

- Encode the possible sequences of interactions using *interface automata* [de Alfaro & Henzinger, 2001].
- Use automated tools to compose the automata, checking for incompatibilities.

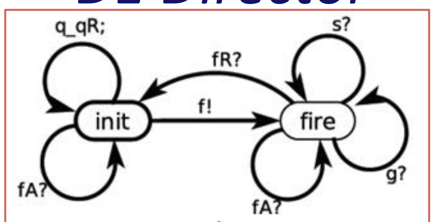
Example automaton for an accessor



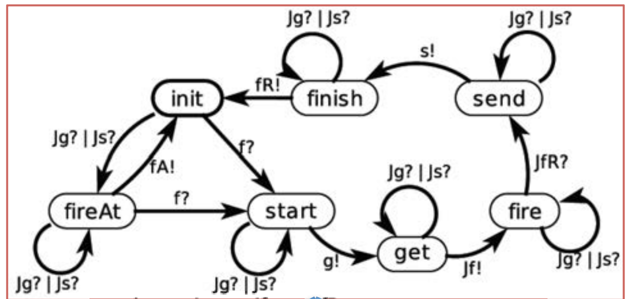
See: Lohstroh and Lee, "An Interface Theory for the Internet of Things," Proc. of SEFM, 2015.

Checking Compatibility of Contracts

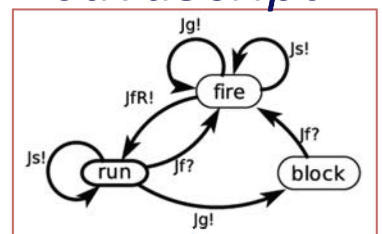
DE Director



Accessor

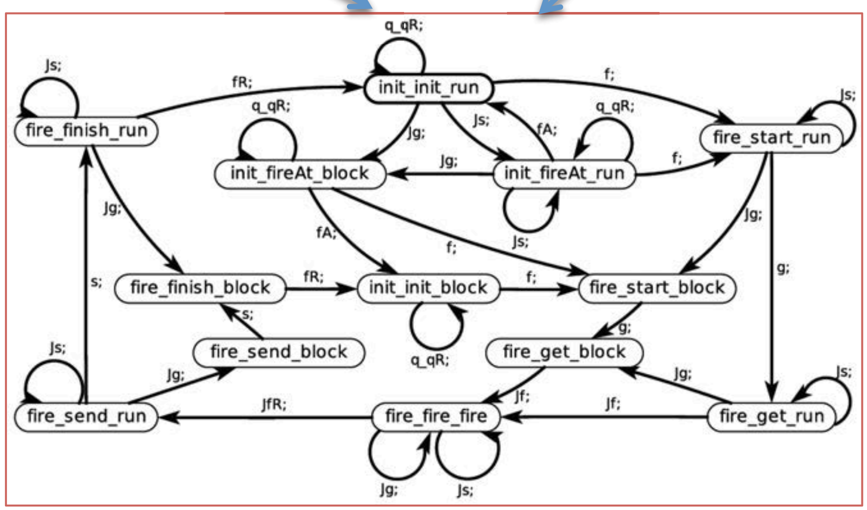


JavaScript



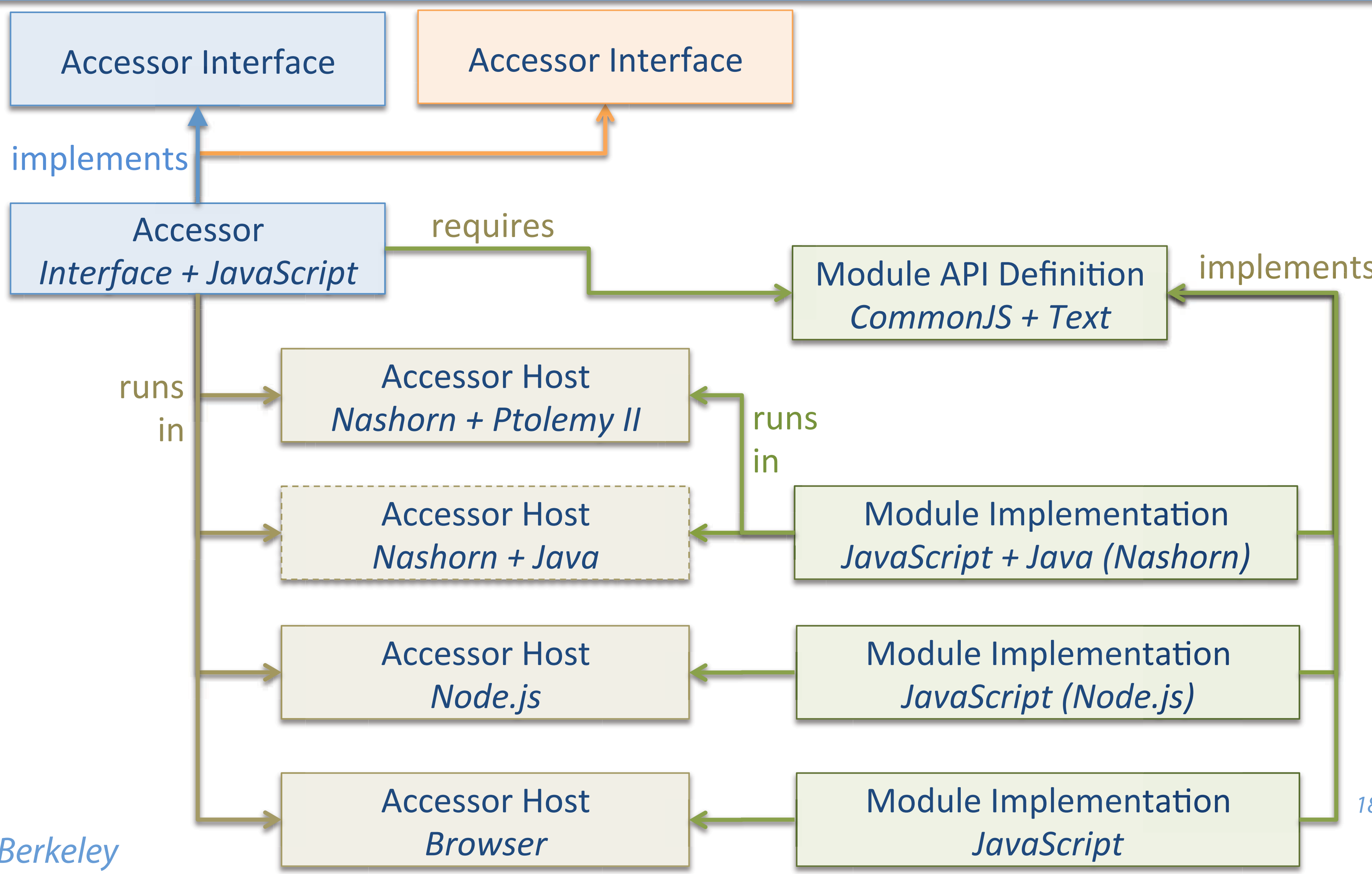
Horizontal Contract

Vertical Contract



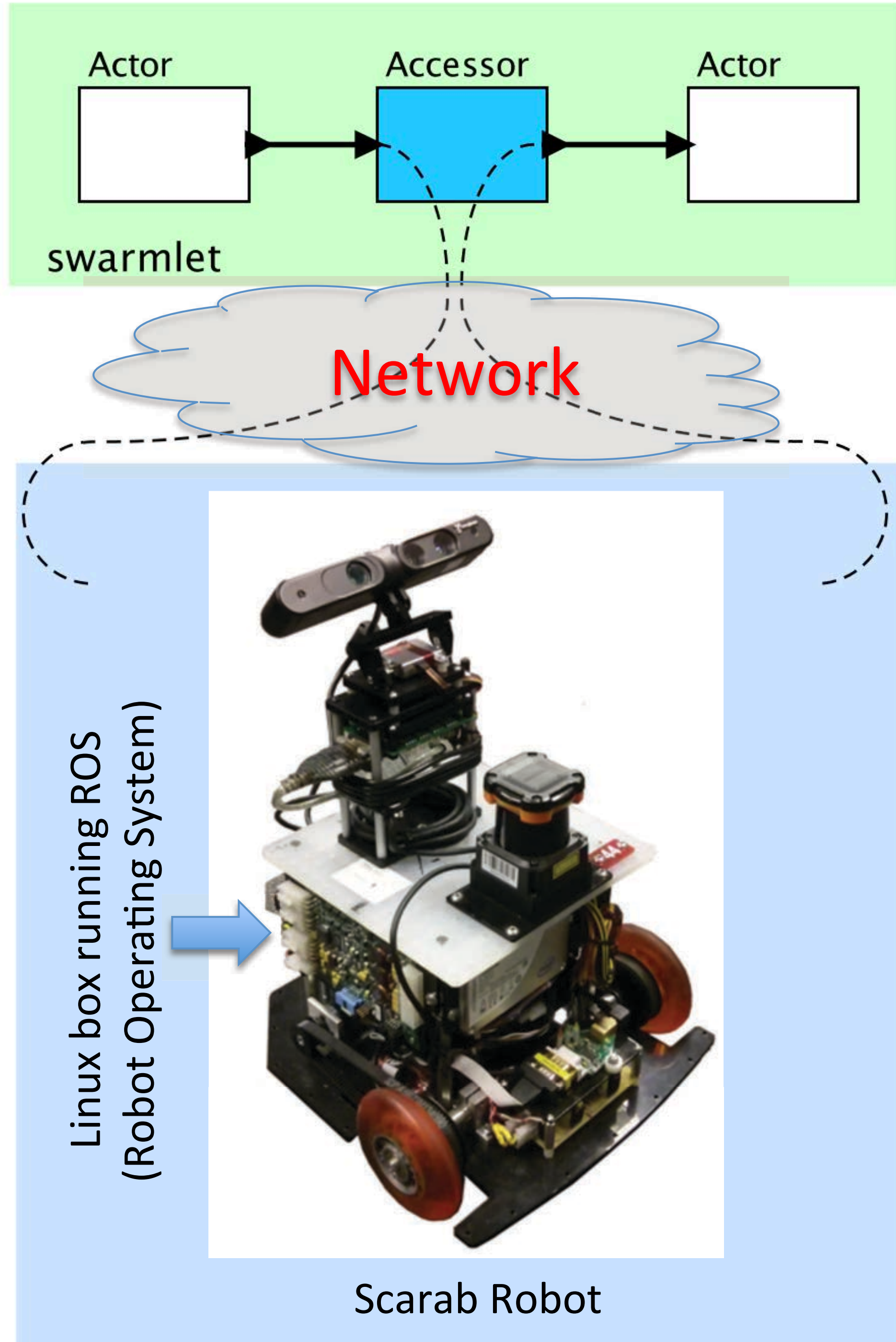
Lohstroh & Lee, "An Interface Theory for the Internet of Things," Proc. of SEFM, 2015.

Accessor Architecture Today
Version 0.1a



Using Accessors

Today, the TerraSwarm project is demonstrating the use of accessors to compose services to orchestrate a fleet of robots.

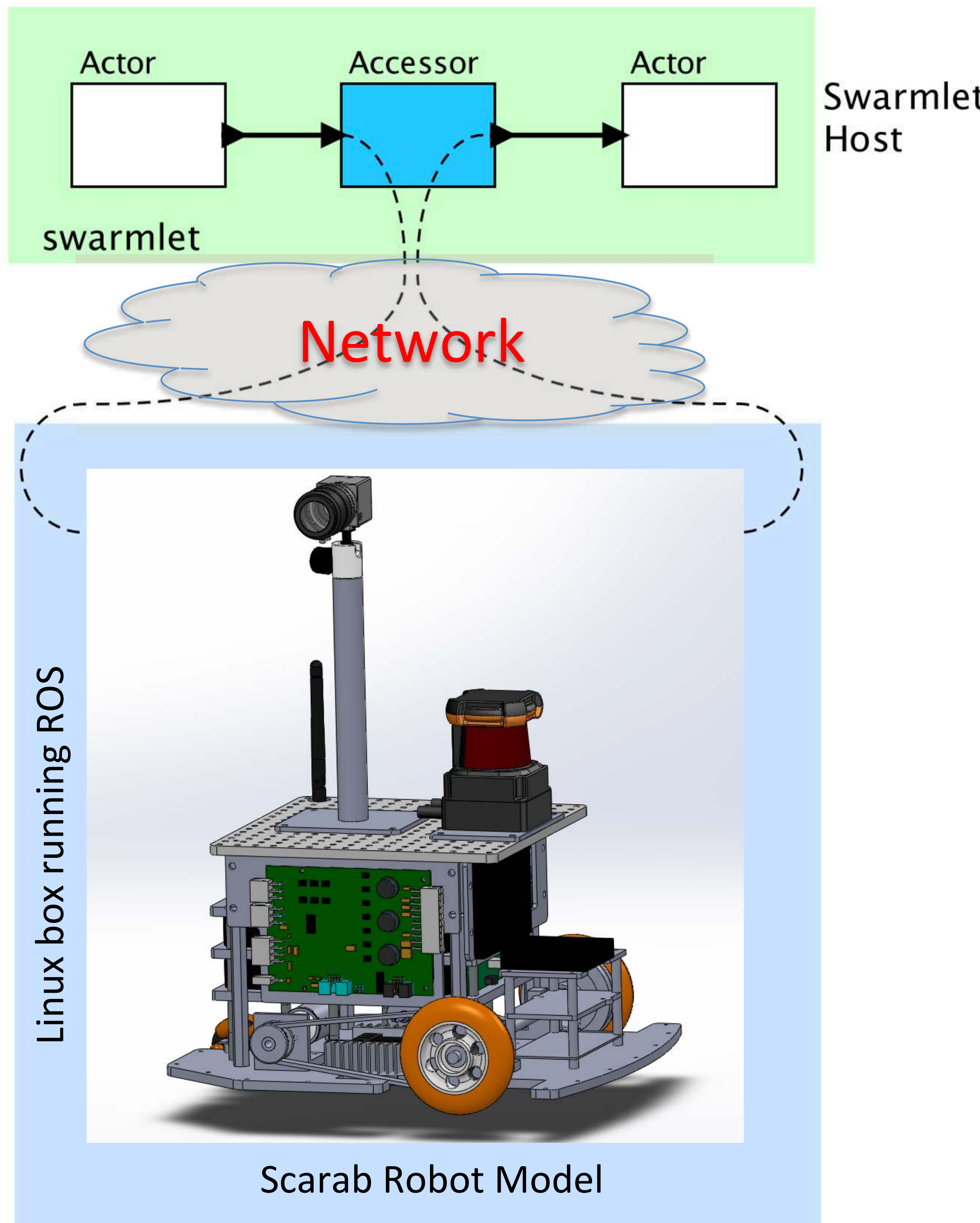


Swarmlet Host

In St. Louis, MO, at DARPA's *Wait, What?* Exposition.

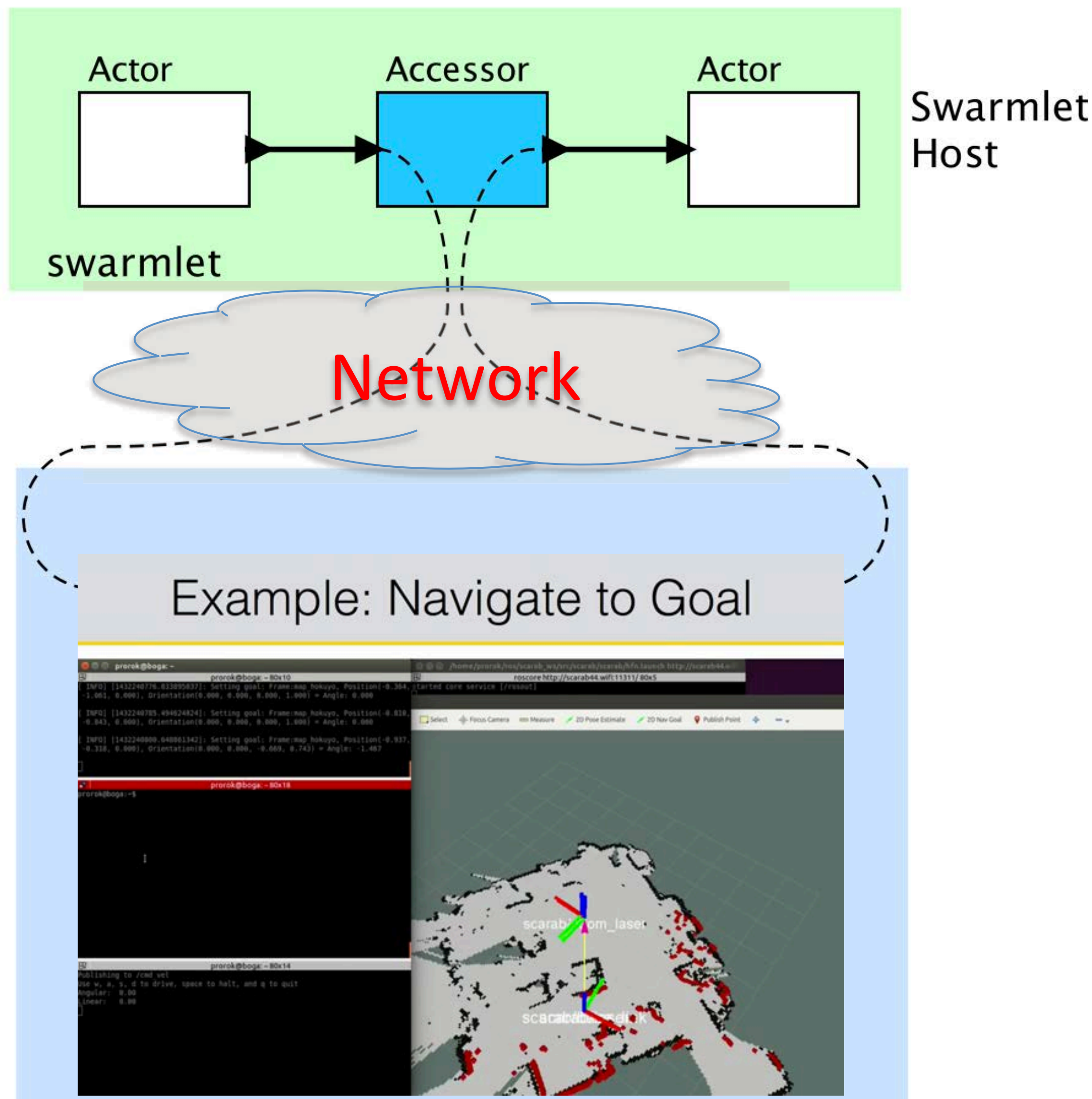
Accessors and Models

Because the interfaces are formal objects, virtual prototypes and simulations behave the same as a (correct) implementation.



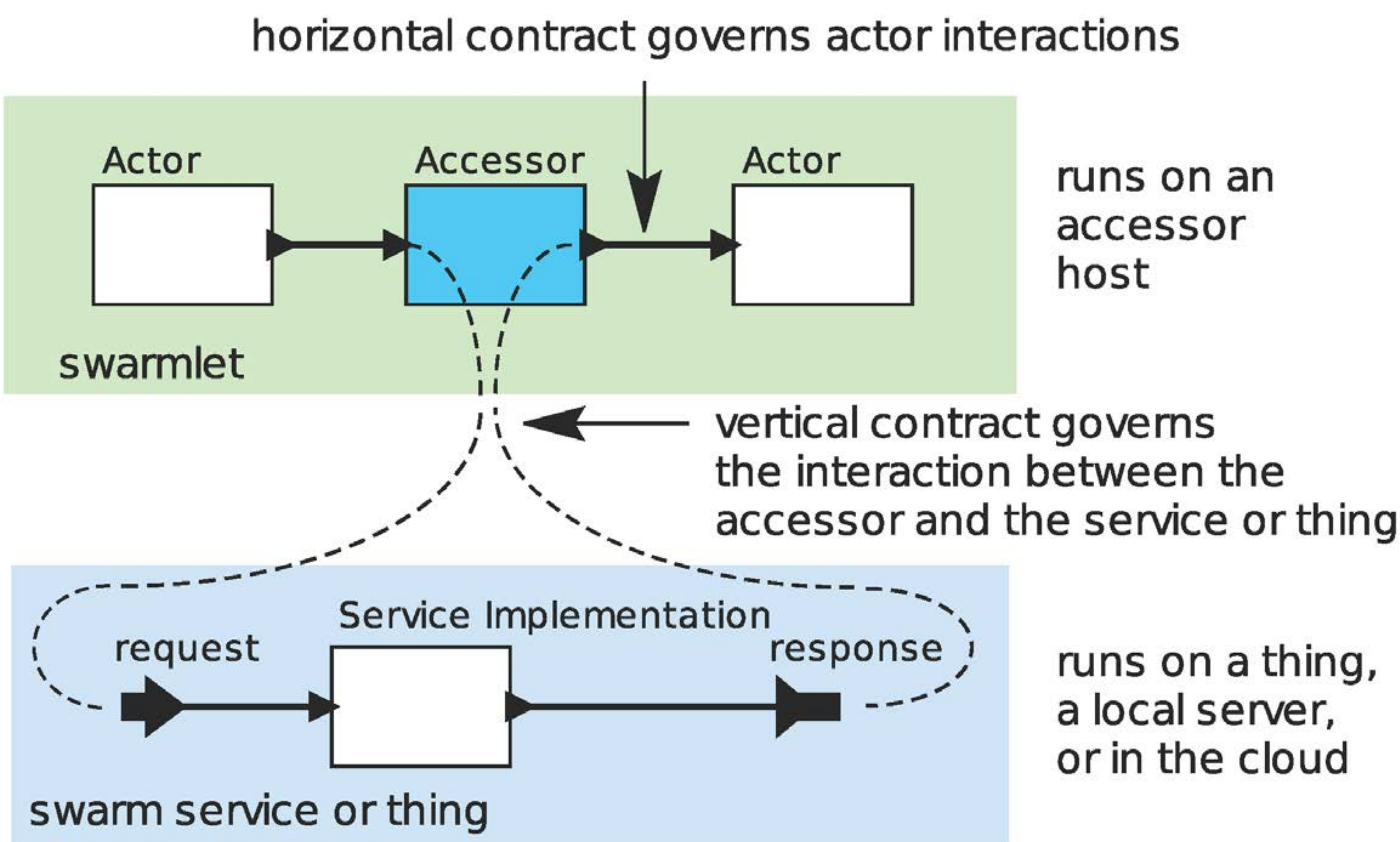
Accessors and Models

Because the interfaces are formal objects, virtual prototypes and simulations behave the same as a (correct) implementation.



Scarab robot simulation has the same interface as the robot itself

Focus on the horizontal contract



Time-stamped events processed in time-stamp order (a discrete-event (DE) model of computation (MoC)).

E.g. asynchronous atomic callbacks (AAC).

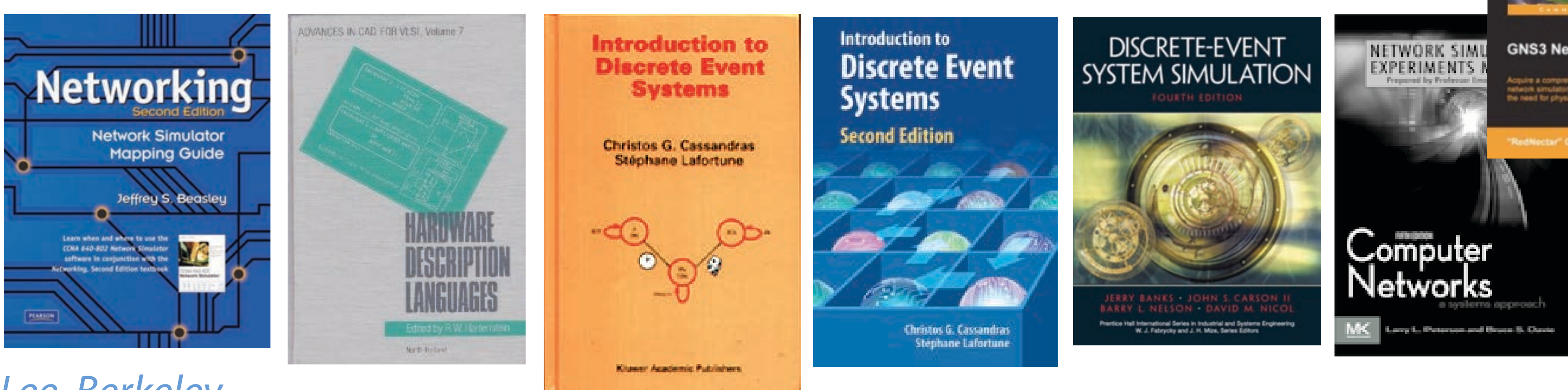
The DE MoC

Time-stamped events that are processed in time-stamp order.

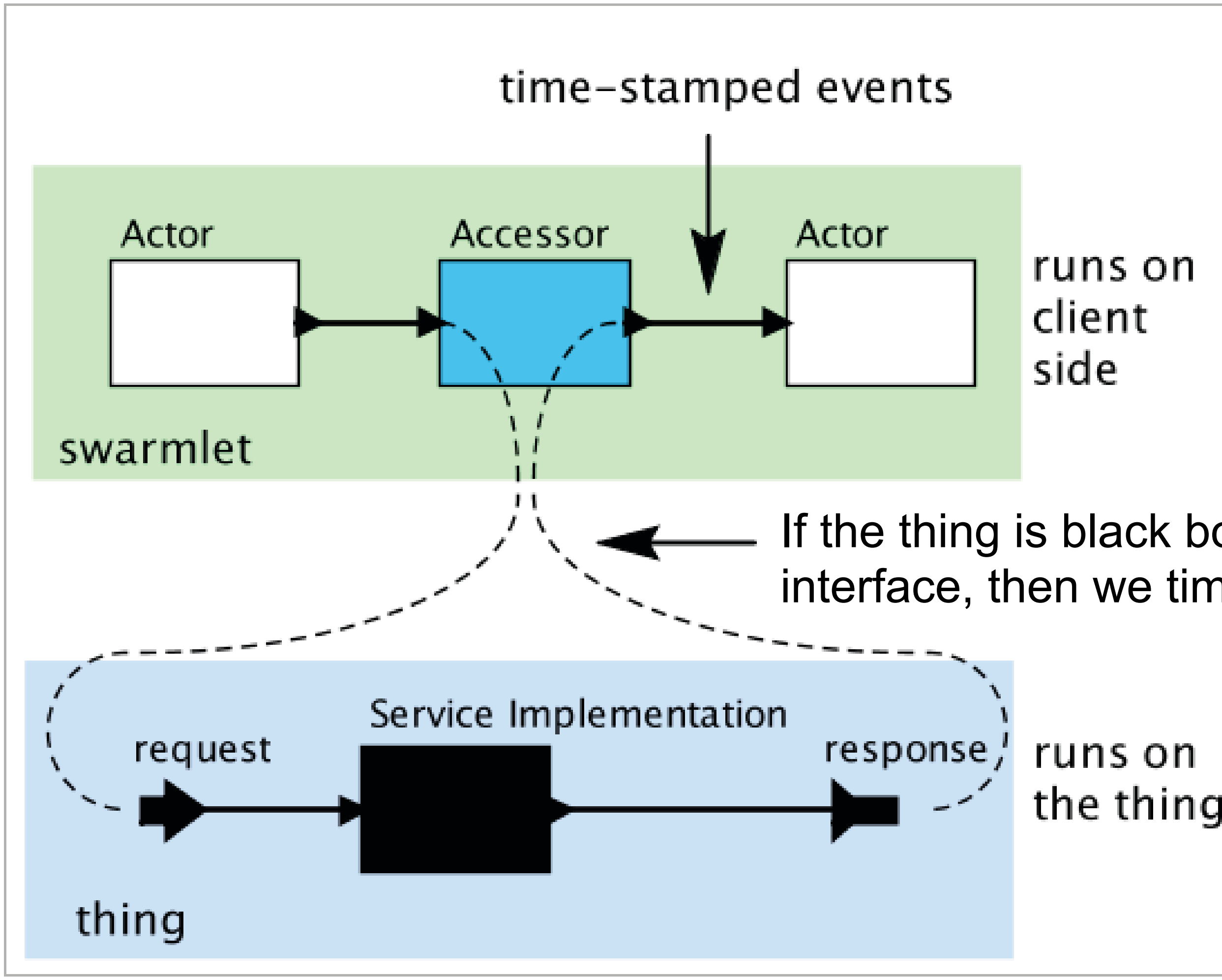
This MoC is widely used in simulation and HDLs.

Given time-stamped inputs, it is a *deterministic* concurrent MoC.

A few texts that use the DE MoC



DE & Determinism



If the thing is black box with a RESTful interface, then we time stamp the response

Networks of causal actors are deterministic under the DE MoC.

… but if we can design the thing, we can do much better!

Ptides – A Robust Distributed DE MoC for IoIT Applications

in Proceedings of the 13th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 07), Bellevue, WA, United States.

A Programming Model for Time-Synchronized Distributed Real-Time Systems

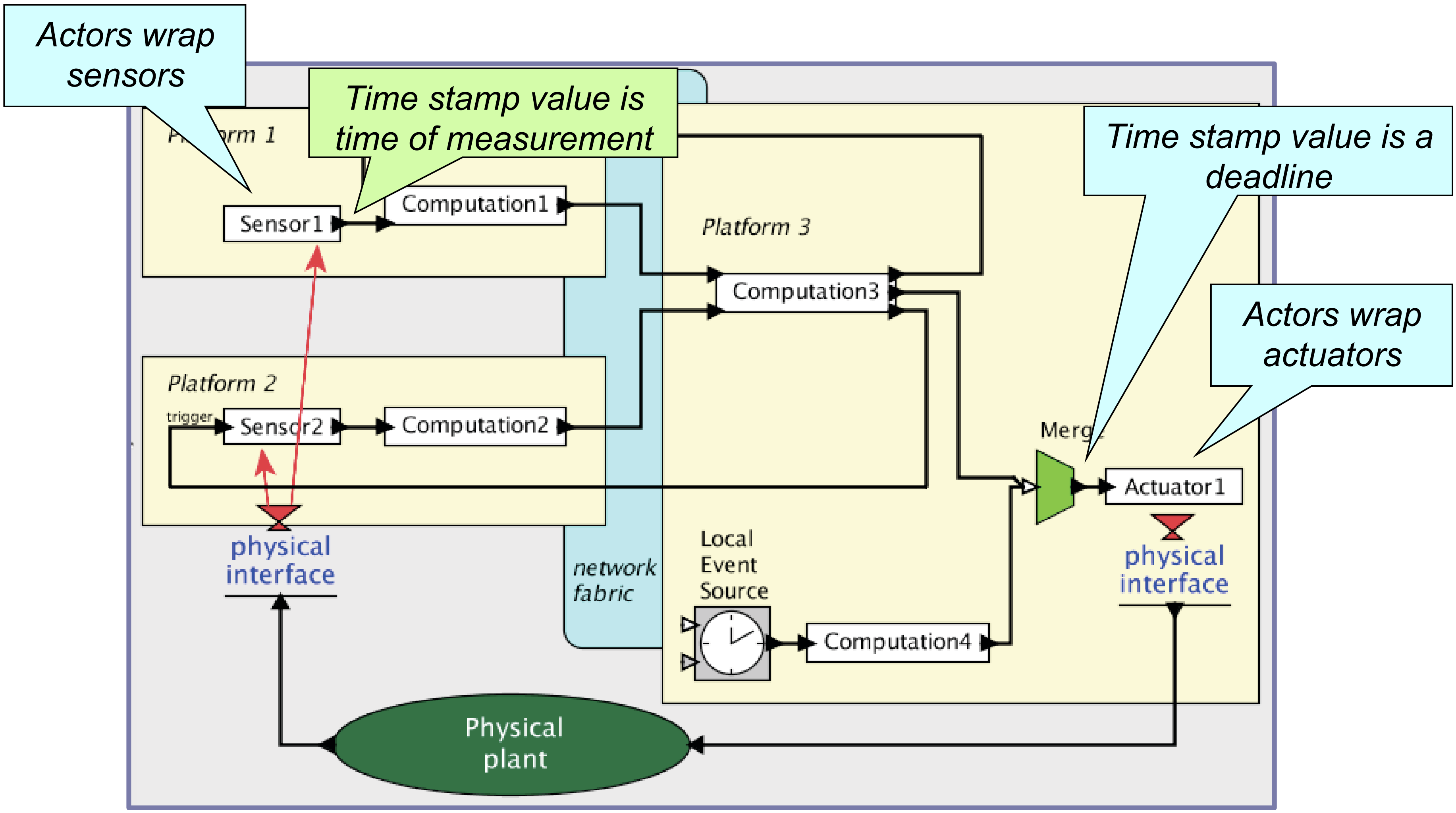
Yang Zhao
EECS Department
UC Berkeley

Jie Liu
Microsoft Research
One Microsoft Way

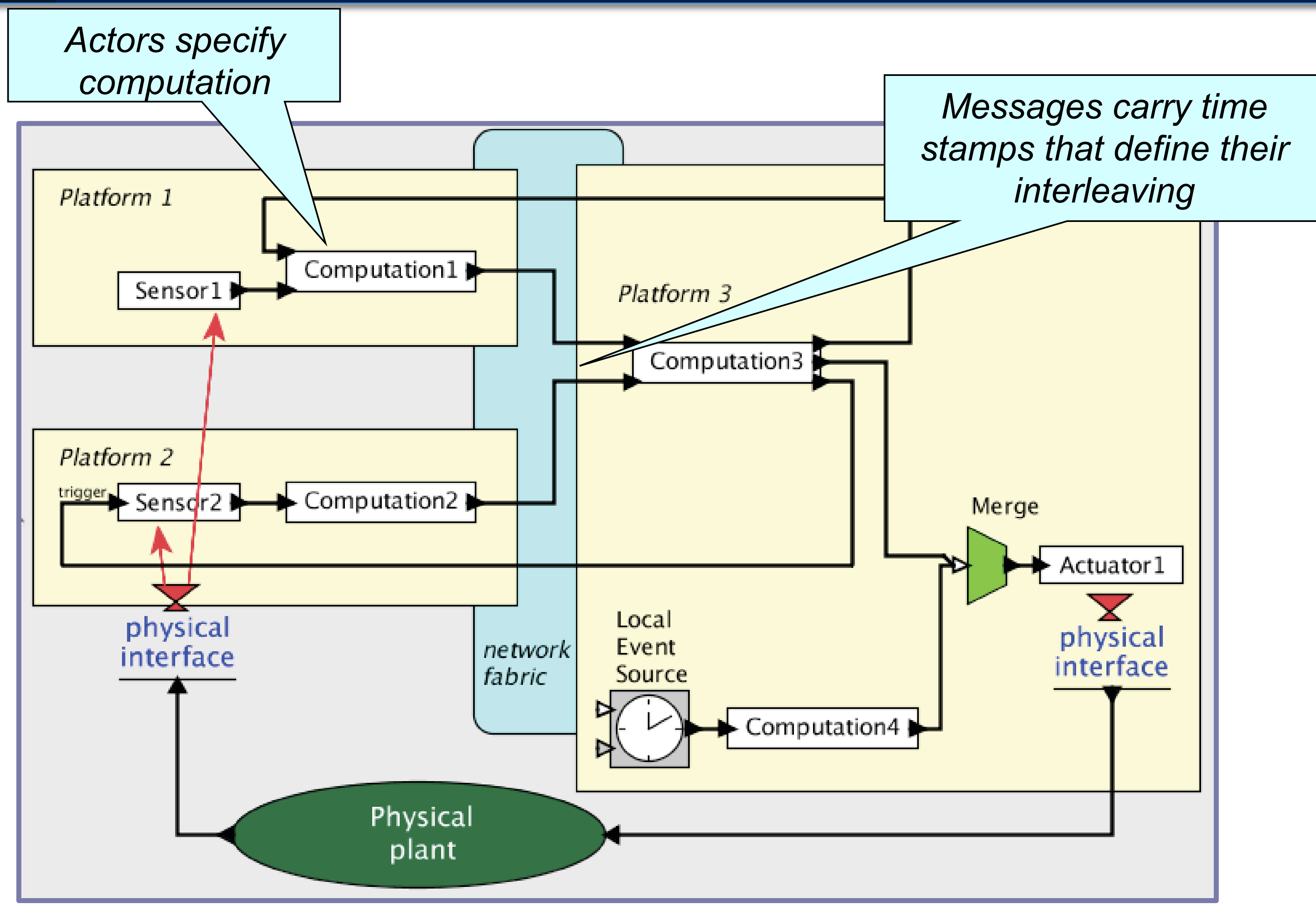
Edward A. Lee
EECS Department
UC Berkeley

Abstract: Discrete-event (DE) models are formal system specifications that have analyzable deterministic behaviors. Using a global, consistent notion of time, DE components communicate via time-stamped events. DE models have primarily been used in performance modeling and simulation, where time stamps are a modeling property bearing no relationship to real time during execution of the model. In this paper, we extend DE models with the capability of relating certain events to physical time…

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

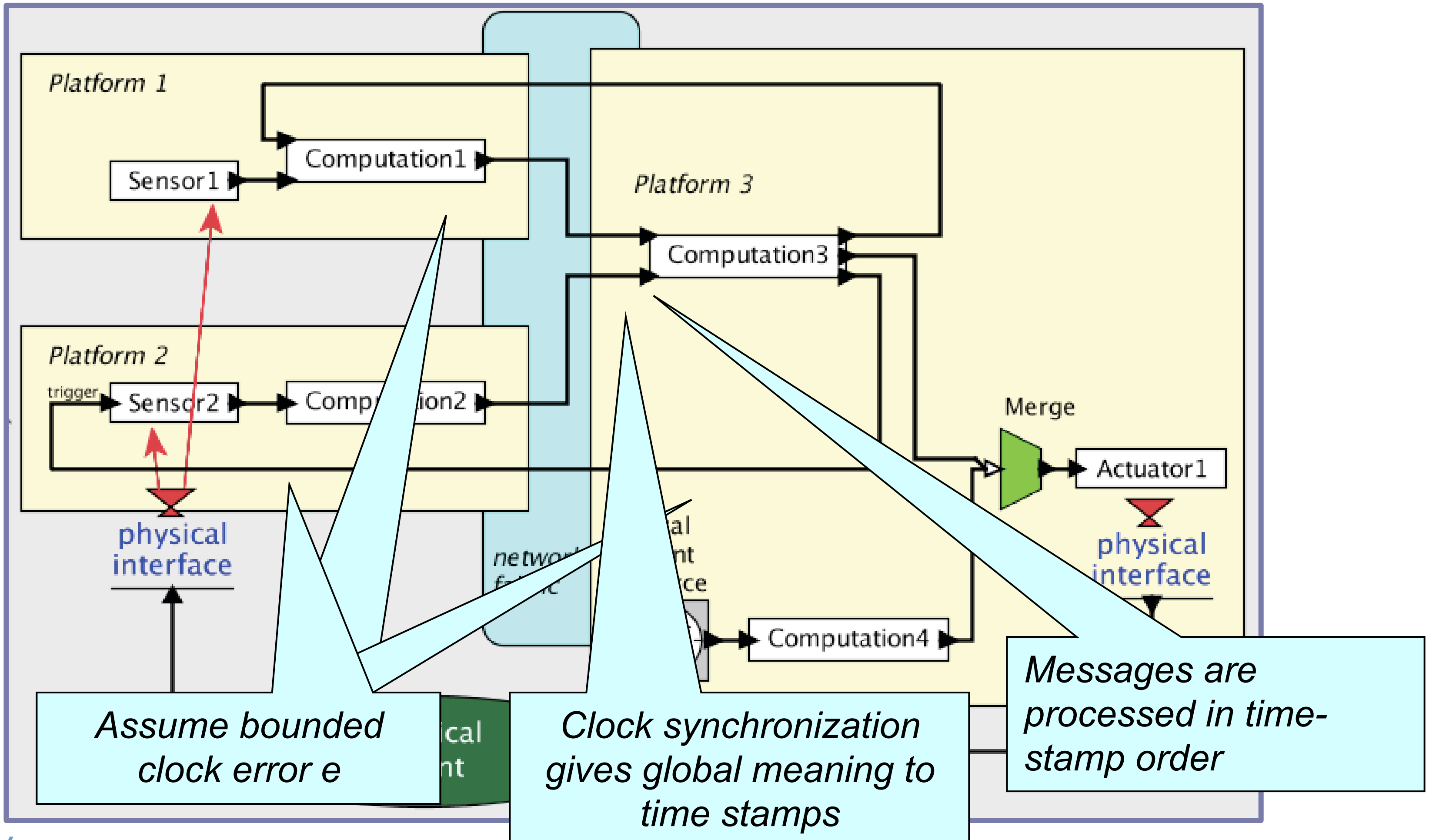


Ptides: Second step: Time-stamped messages.



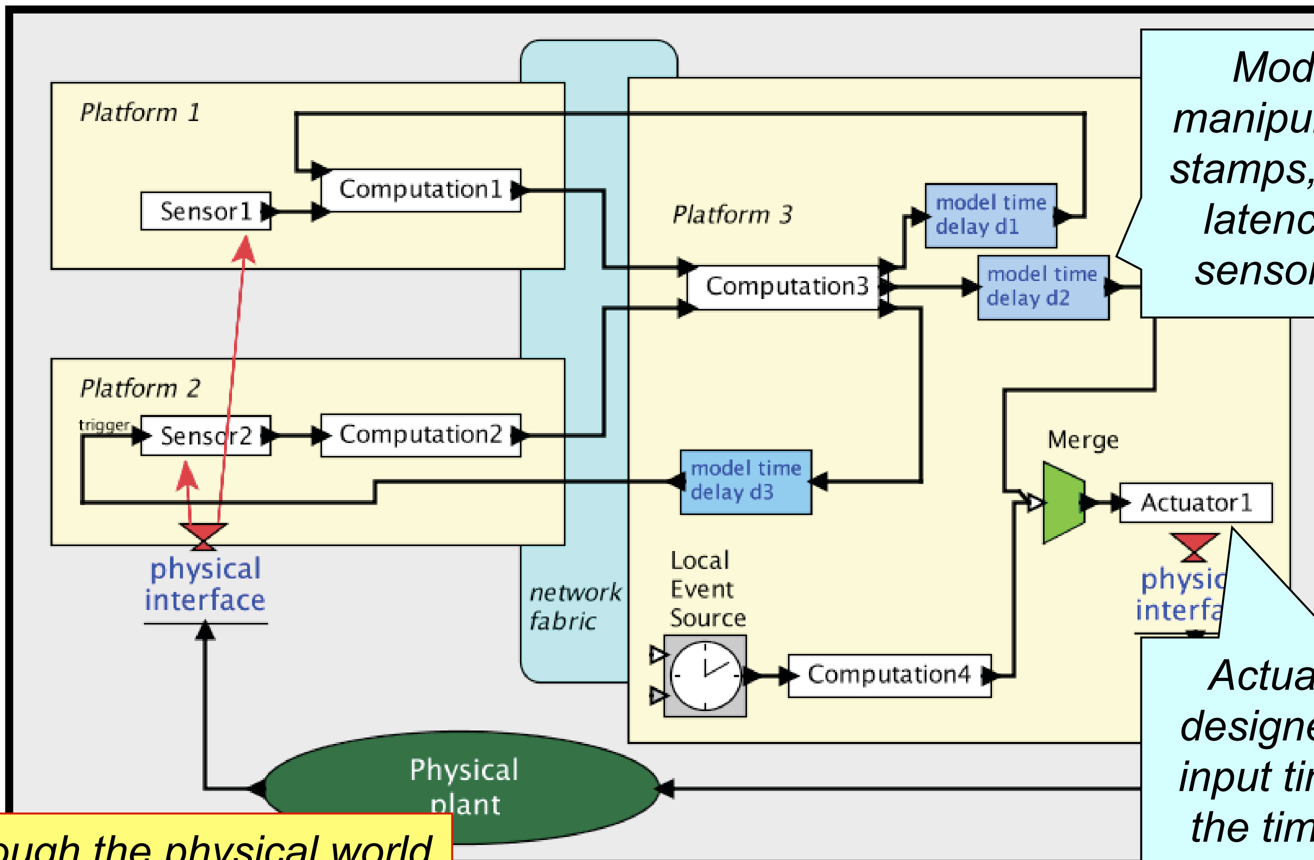
Ptides: Third step: Network clock synchronization

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)

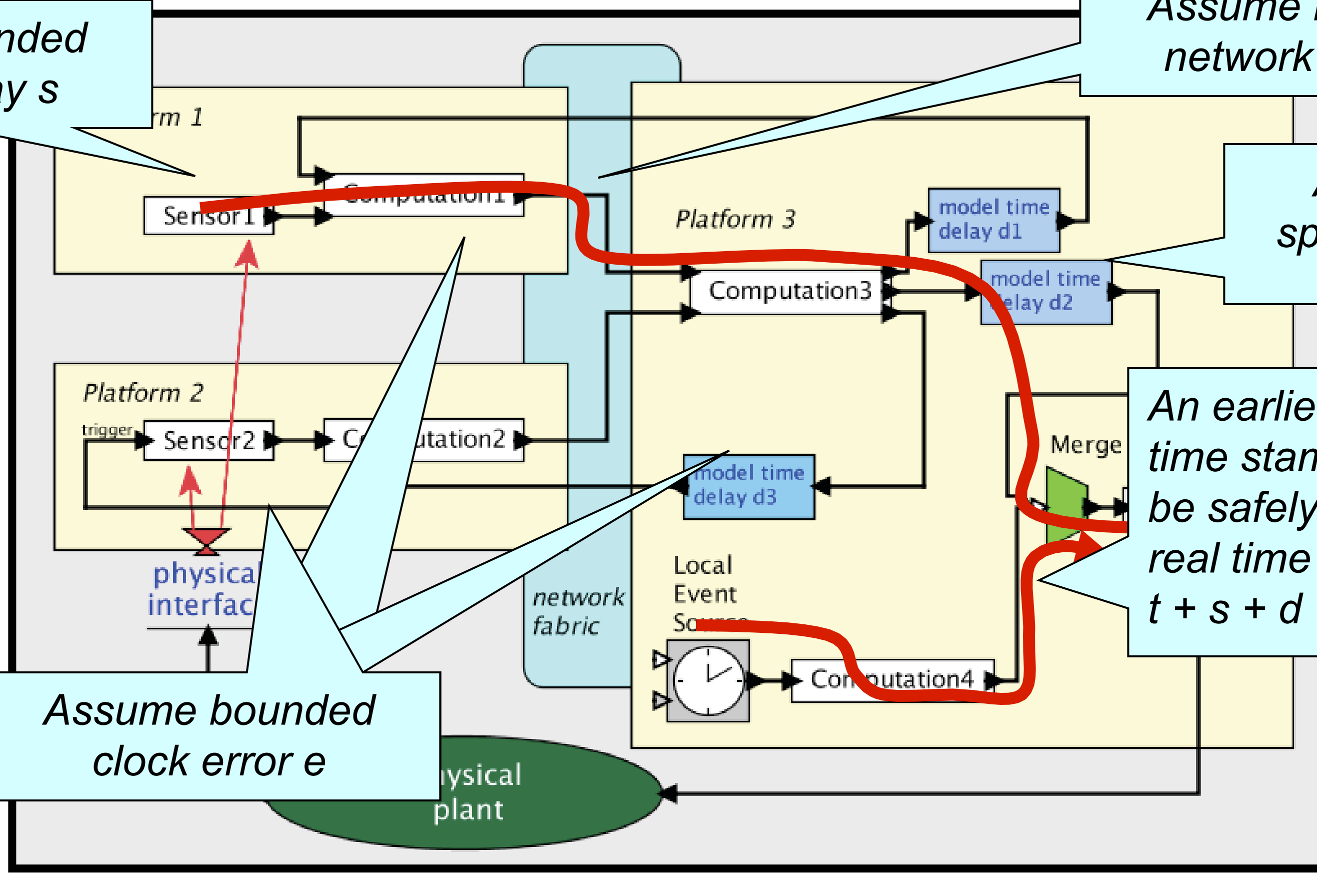
Safe-to-process analysis guarantees that events are processed in time-stamp order, given some assumptions.

Assume bounded sensor delay s

Assume bounded network delay d

Application specification of latency d2

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



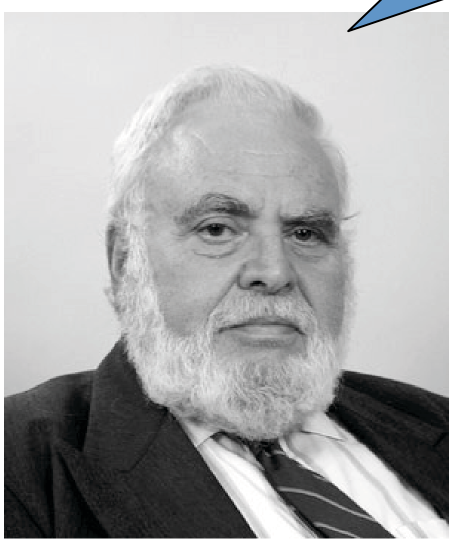
An earliest event with time stamp t here can be safely merged when real time exceeds t + s + d + e – d2

Assume bounded clock error e

So Many Assumptions?

Solomon Wolf Golomb: On keeping the model distinct from the thing being modeled:

You will never strike oil by drilling through the map!



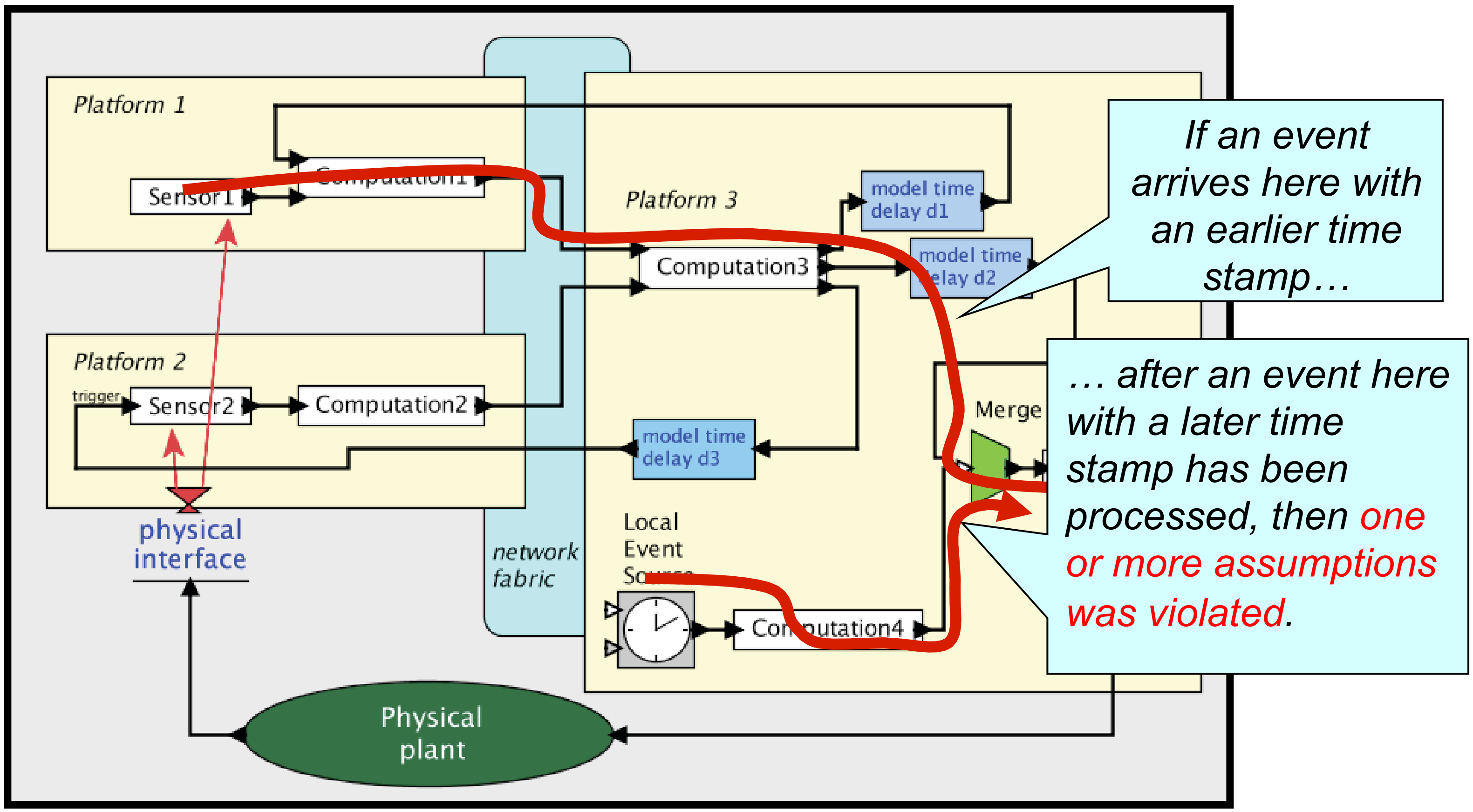
*All of the assumptions are achievable with today's technology, and in fact are **requirements** anyway for hard-real-time systems. The Ptides model makes the assumptions explicit.*

*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.



Google Spanner – A Reinvention of PTIDES

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 – A Reinvention of PTIDES

Update to a record 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.

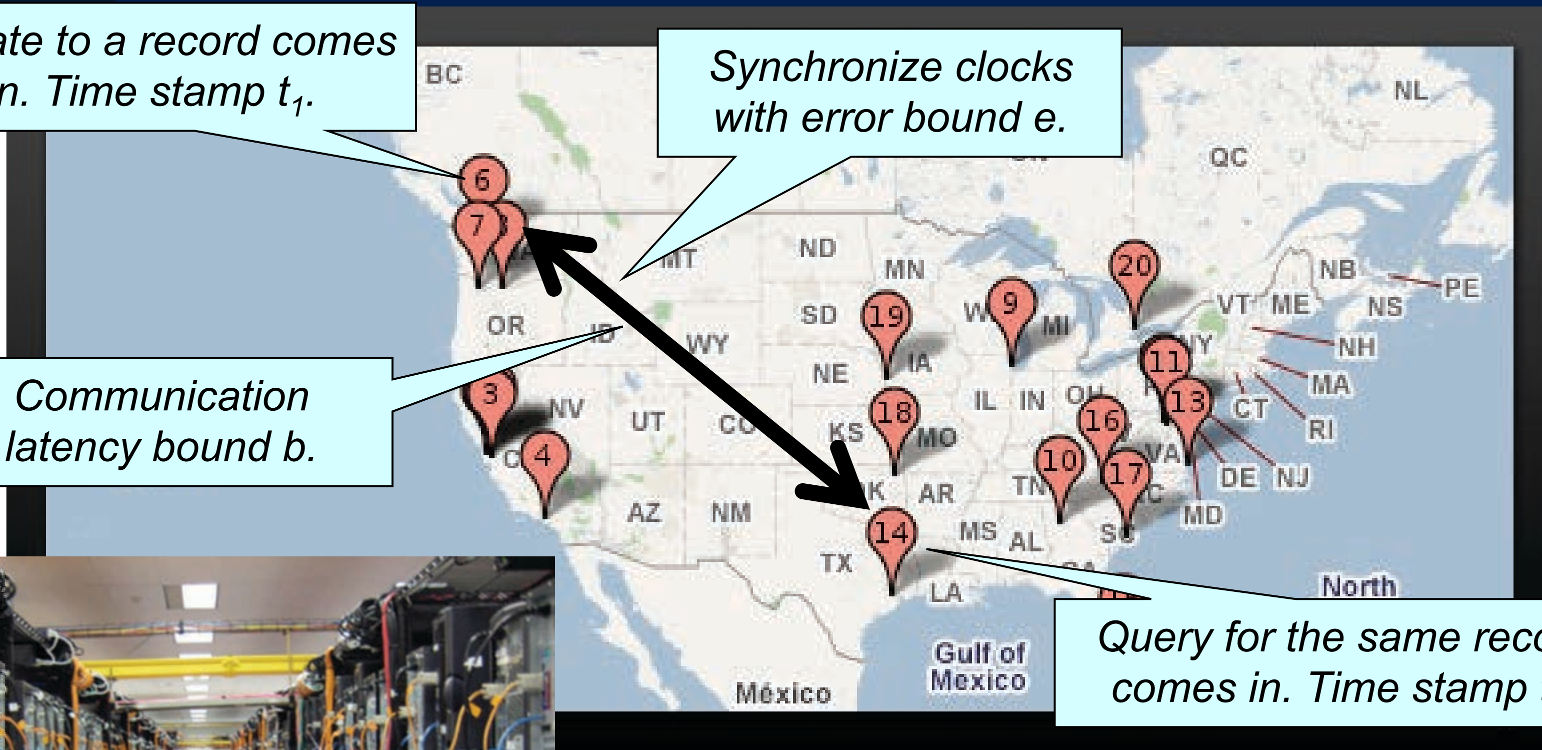
Google Spanner – A Reinvention of PTIDES

Update to a record 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?



Update to a record 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.

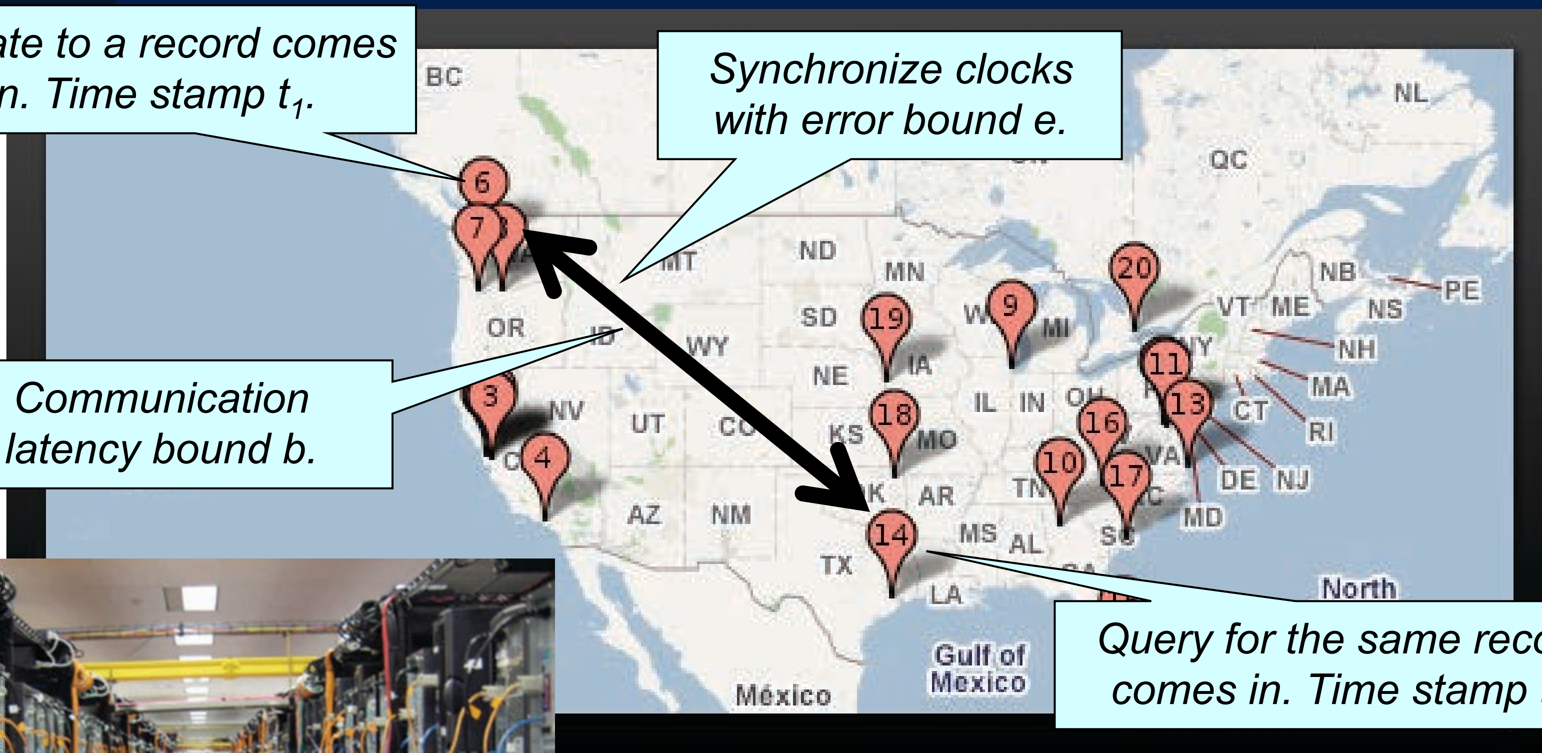
Google Spanner: Fault!

Update to a record 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.

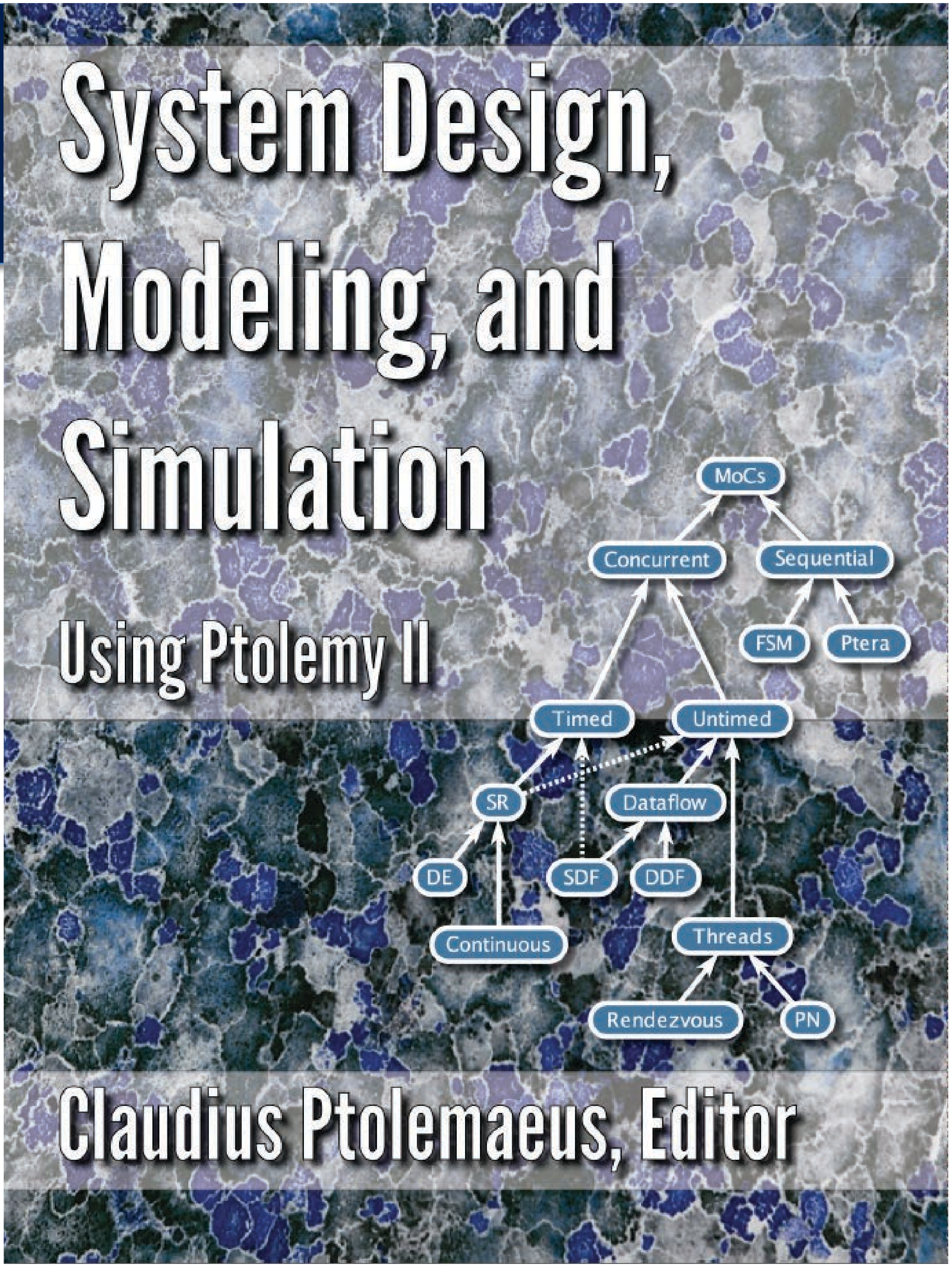
See Book

See

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

Free download at:

http://ptolemy.org/systems



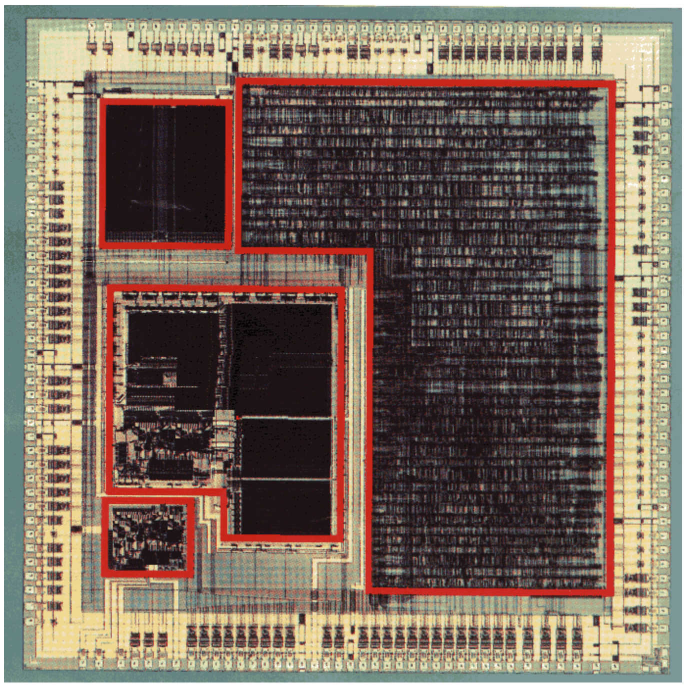
Determinism? Really?

IoIT applications operate in an intrinsically nondeterministic world.

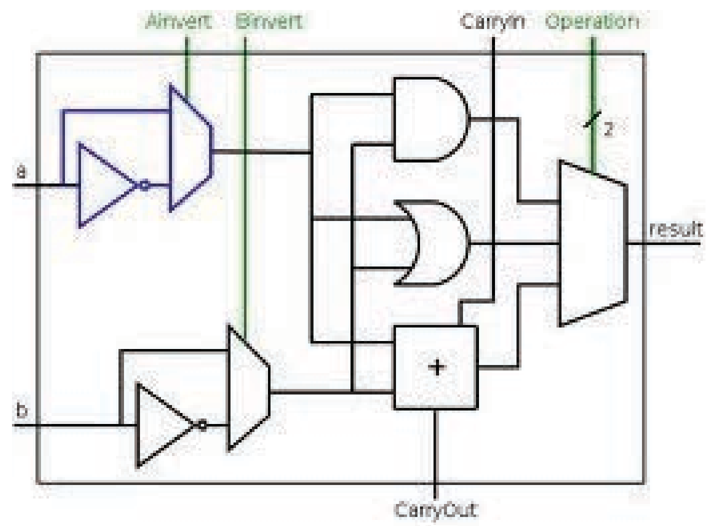
Does it really make sense to insist on deterministic models?

Deterministic Models of Nondeterministic Systems

Physical System



Model



Synchronous digital logic

Deterministic Models of Nondeterministic Systems

Physical System

Model

Image: Wikimedia Commons

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

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

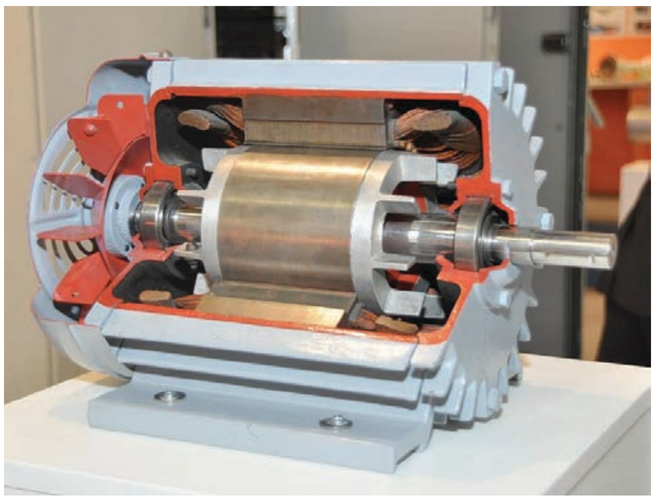
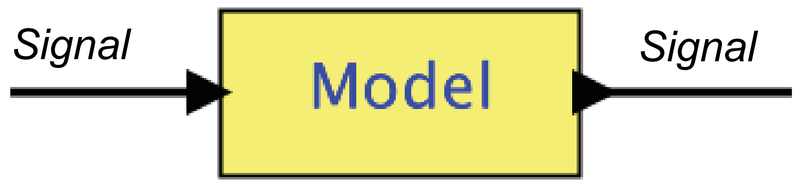


Image: Wikimedia Commons

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

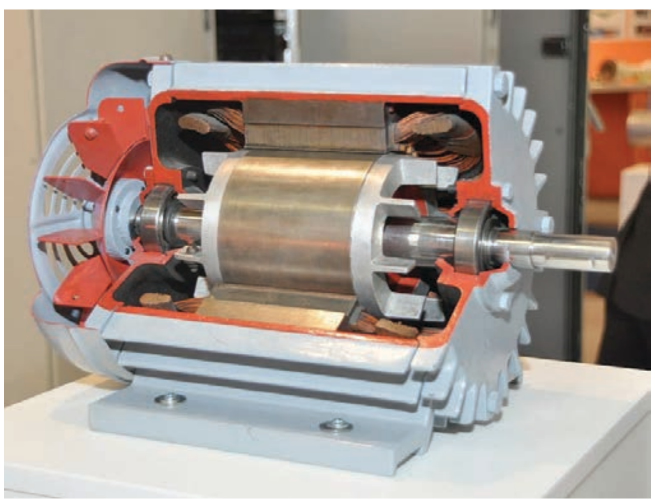
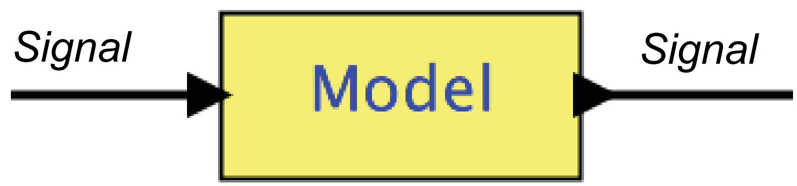


Image: Wikimedia Commons

```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();
                        }
                    }
                }
            }
        }
    }
}
```



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

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!

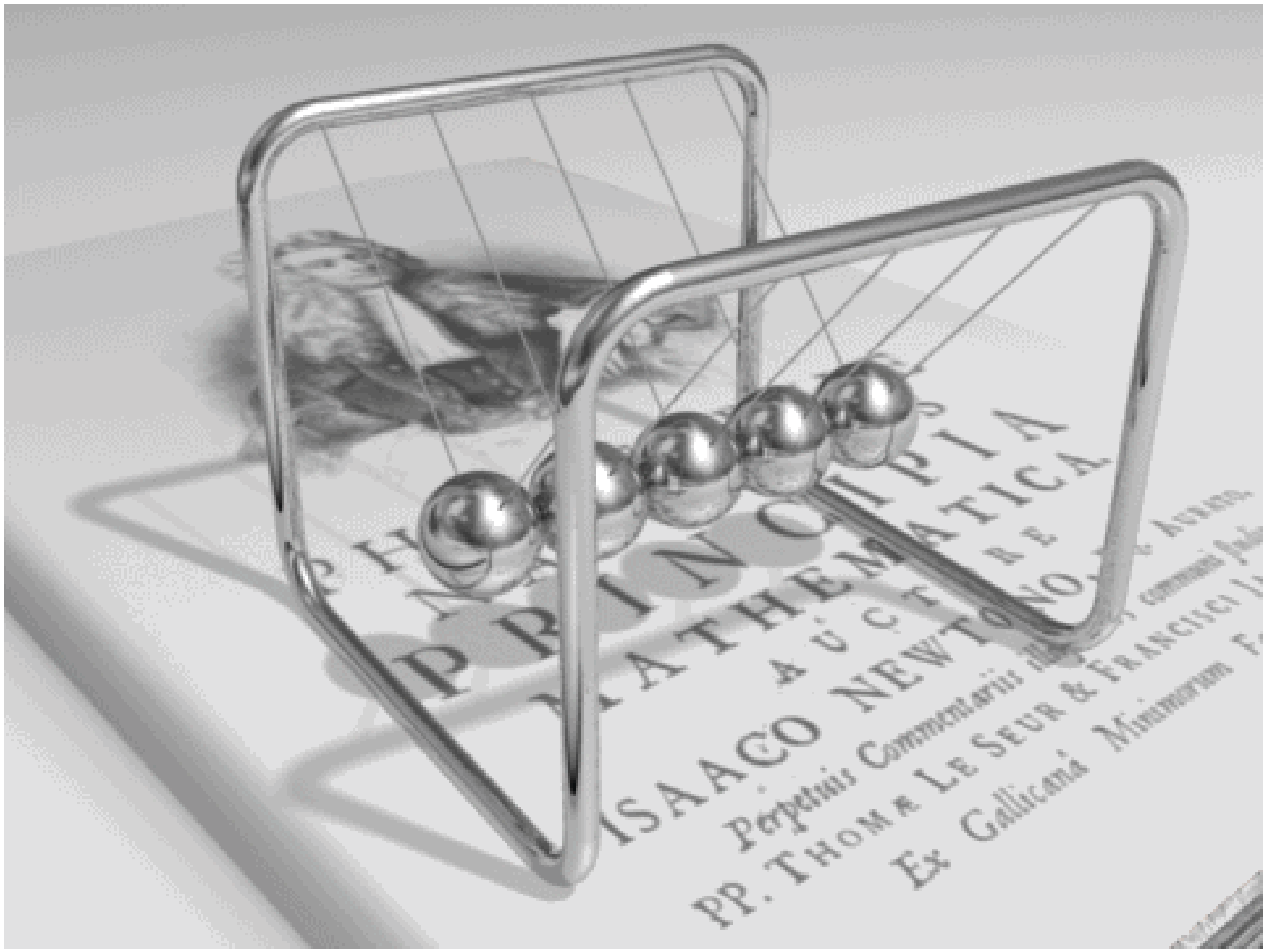
One Last Comment… Model Fidelity
… or … How often will faults occur?

- 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.

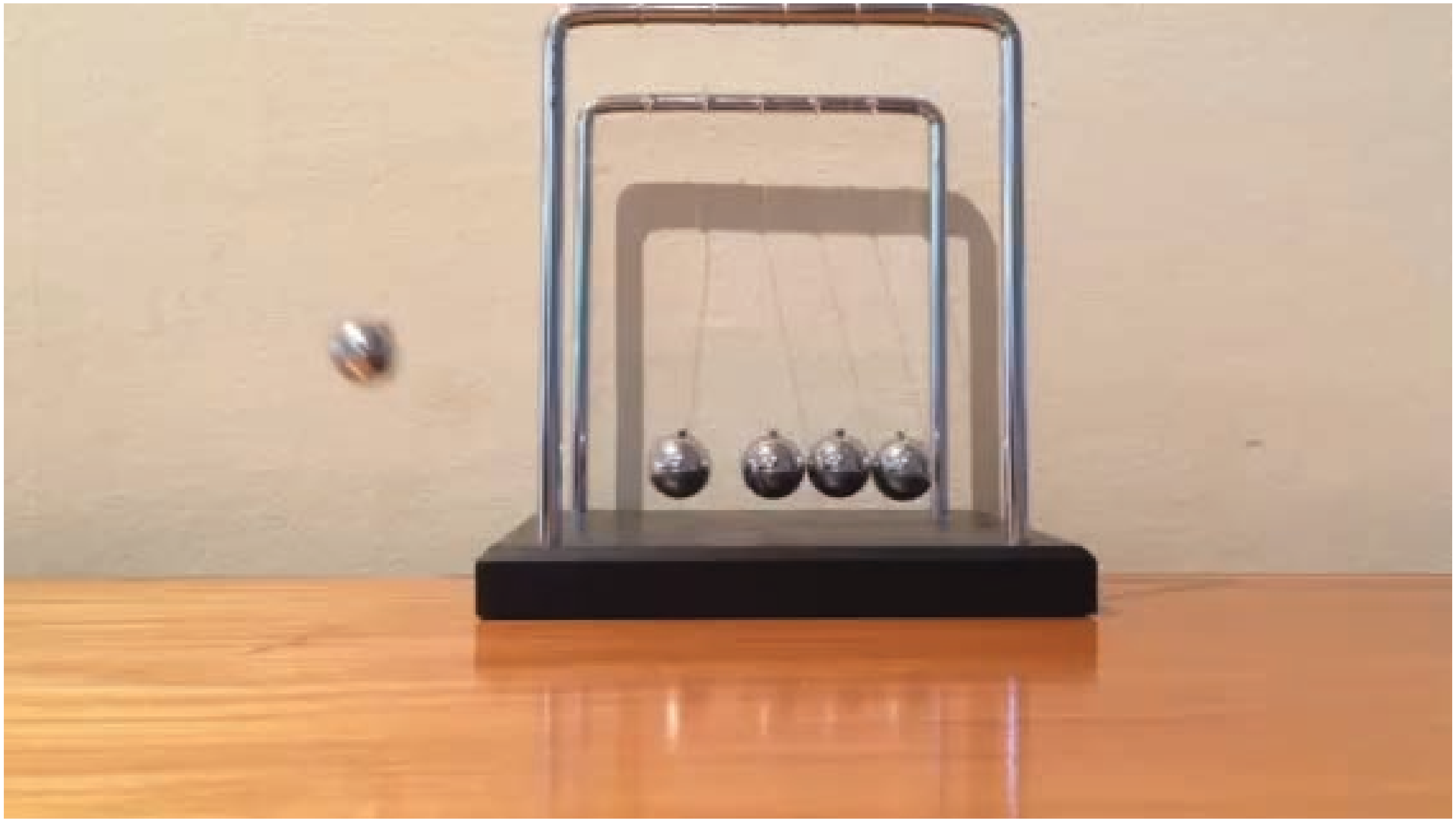
In engineering, model fidelity is a two-way street!

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


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 offers better models with less flawed physical realizations.

Determinism?

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

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

Conclusion

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

IoIT and CPS demand new ways of modeling. They require models that embrace some notion(s) of *time*, and models with *strong formal properties*.

Raffaello Sanzio da Urbino – The Athens School