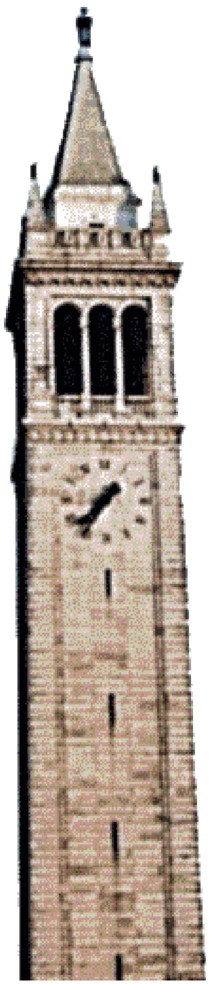


Accessors:
An Open Architecture for the Internet of Things

Edward A. Lee
Robert S. Pepper Distinguished Professor
UC Berkeley

69th IFIP WG 10.4 Meeting and Workshop on Internet of Things
10-14 January, 2016 – Aspen/Snowmass, US

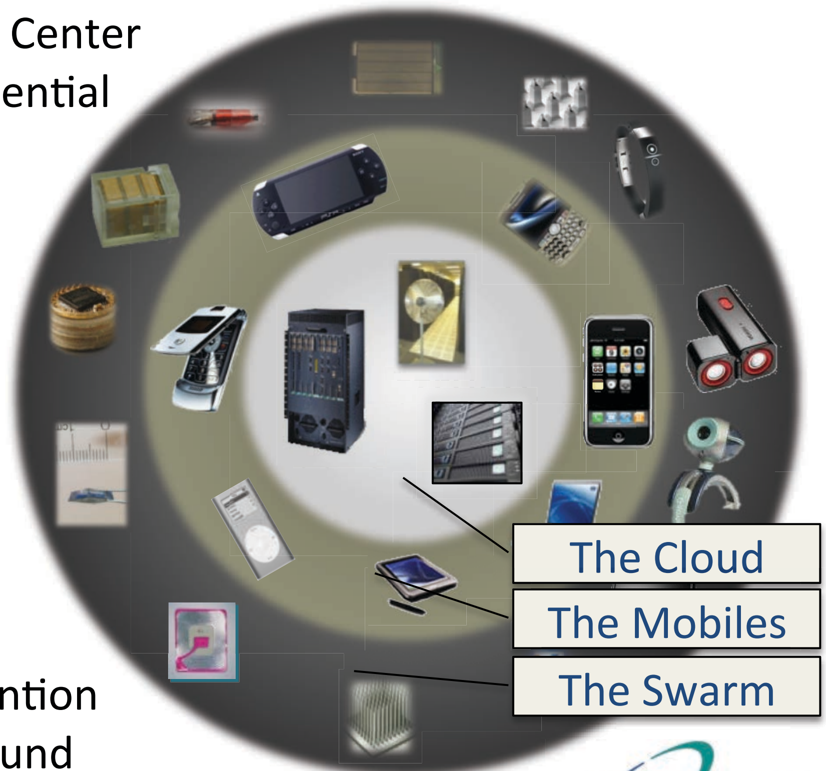
Context:
The TerraSwarm Research Center (2013-2018)

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

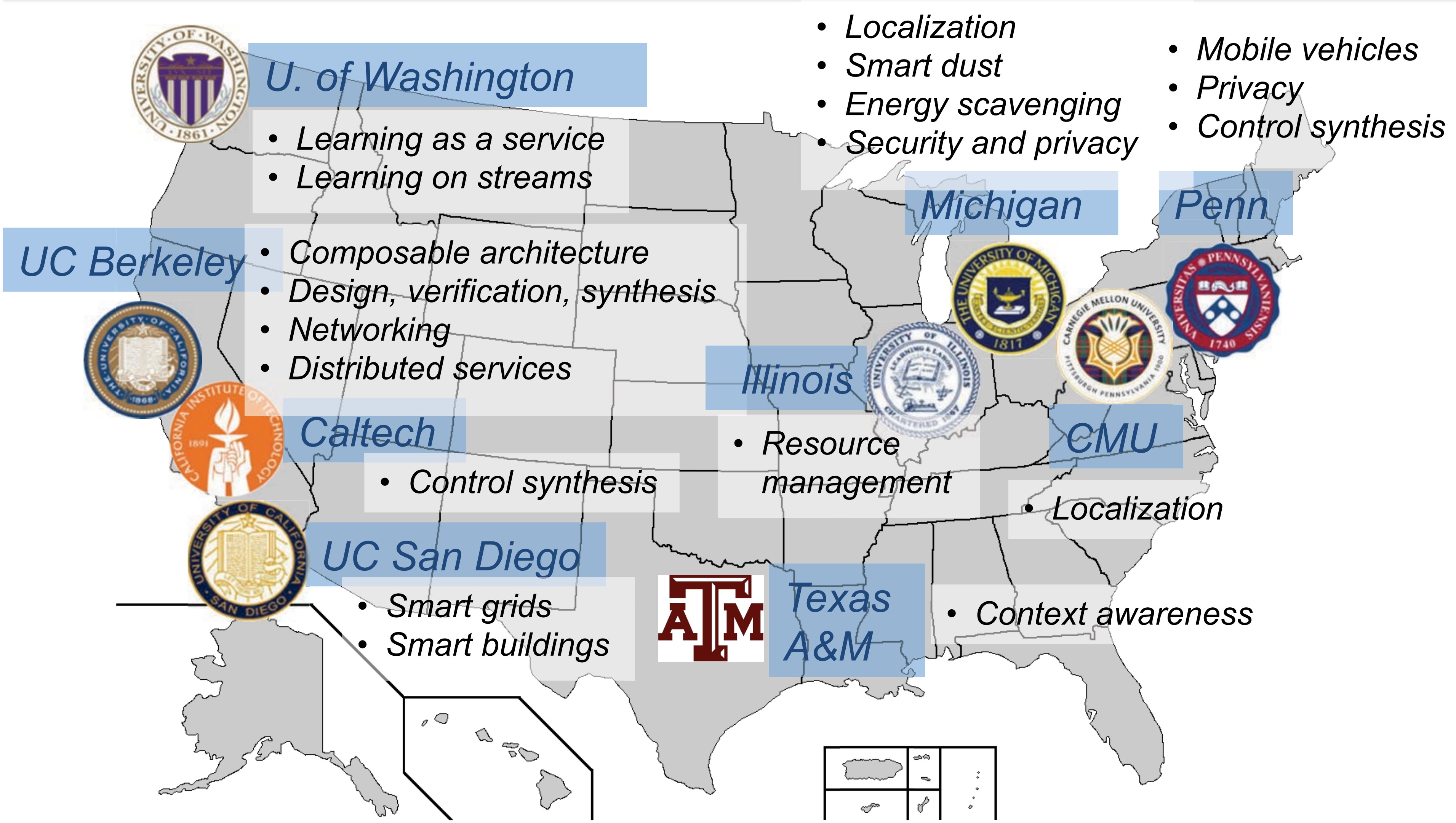
To lead the world in development of the platforms, methodologies, and tools that enable invention of creative, secure, and sound applications using networked sensors and actuators.



The Sponsors:

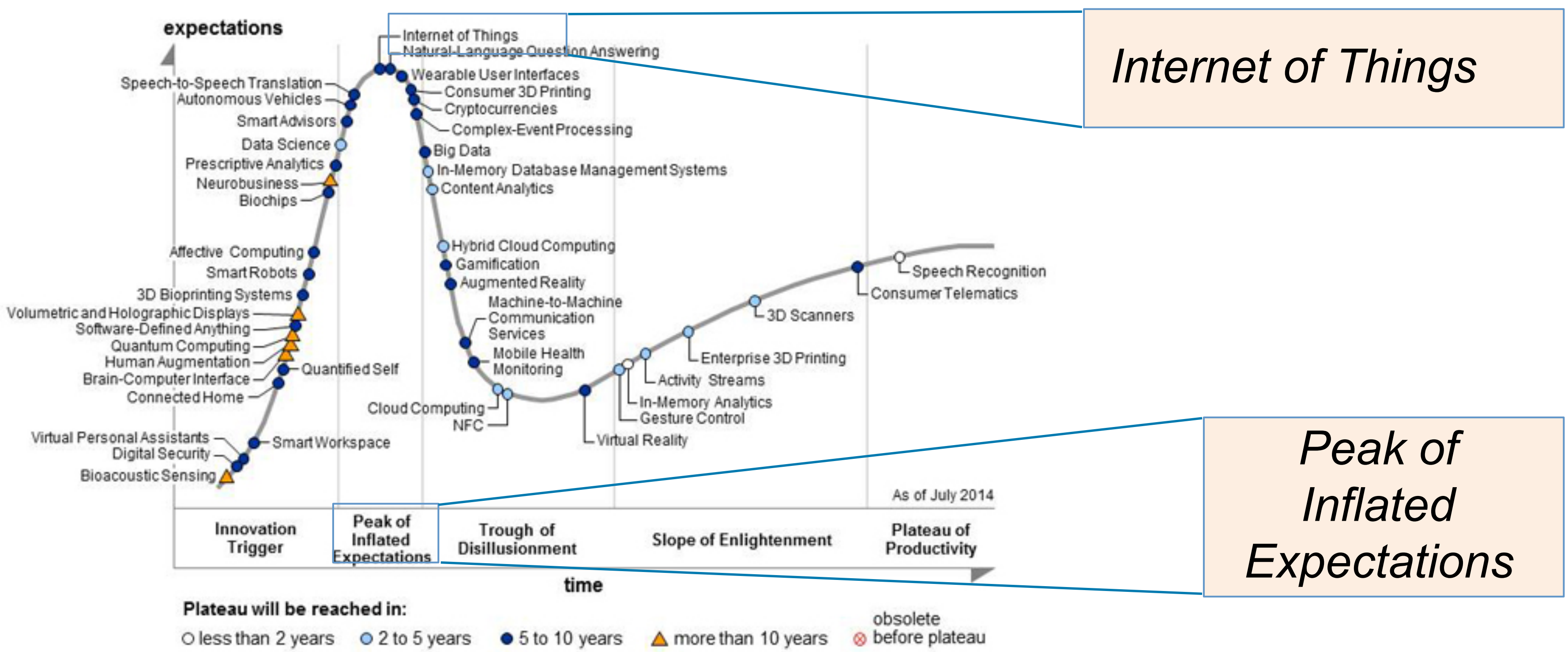
DARPA
GLOBAL FOUNDRIES
IBM
intel
Micron
Raytheon
TEXAS INSTRUMENTS
United Technologies

TerraSwarm Sites



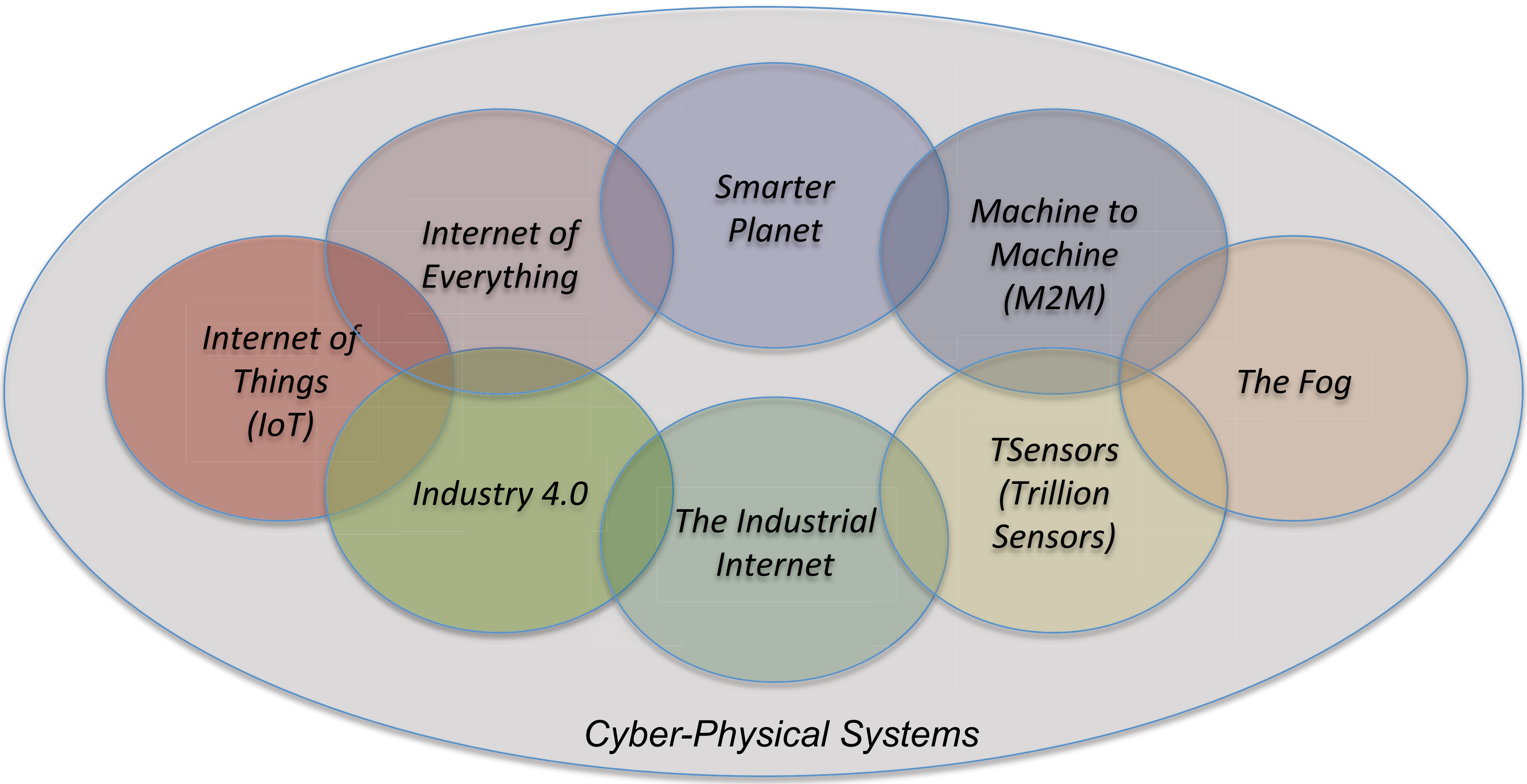
Buzzword du jour:
The Internet of Things

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…

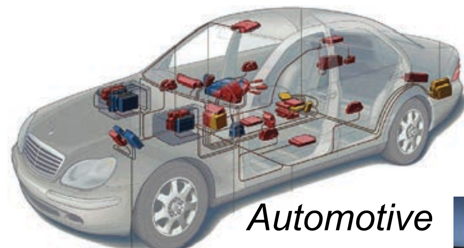


CPS

Underlies much of the industrial economy

It’s not just information technology anymore:

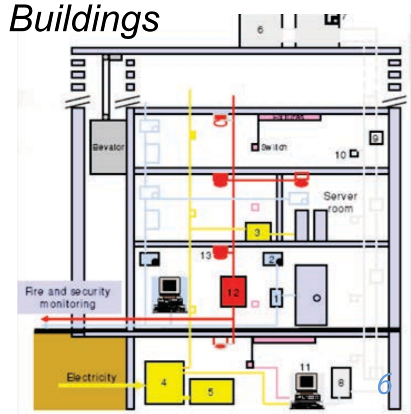
- Cyber + *Physical*
- Computation + *Dynamics*
- Security + *Safety*


Automotive

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.

IoT Background at Berkeley

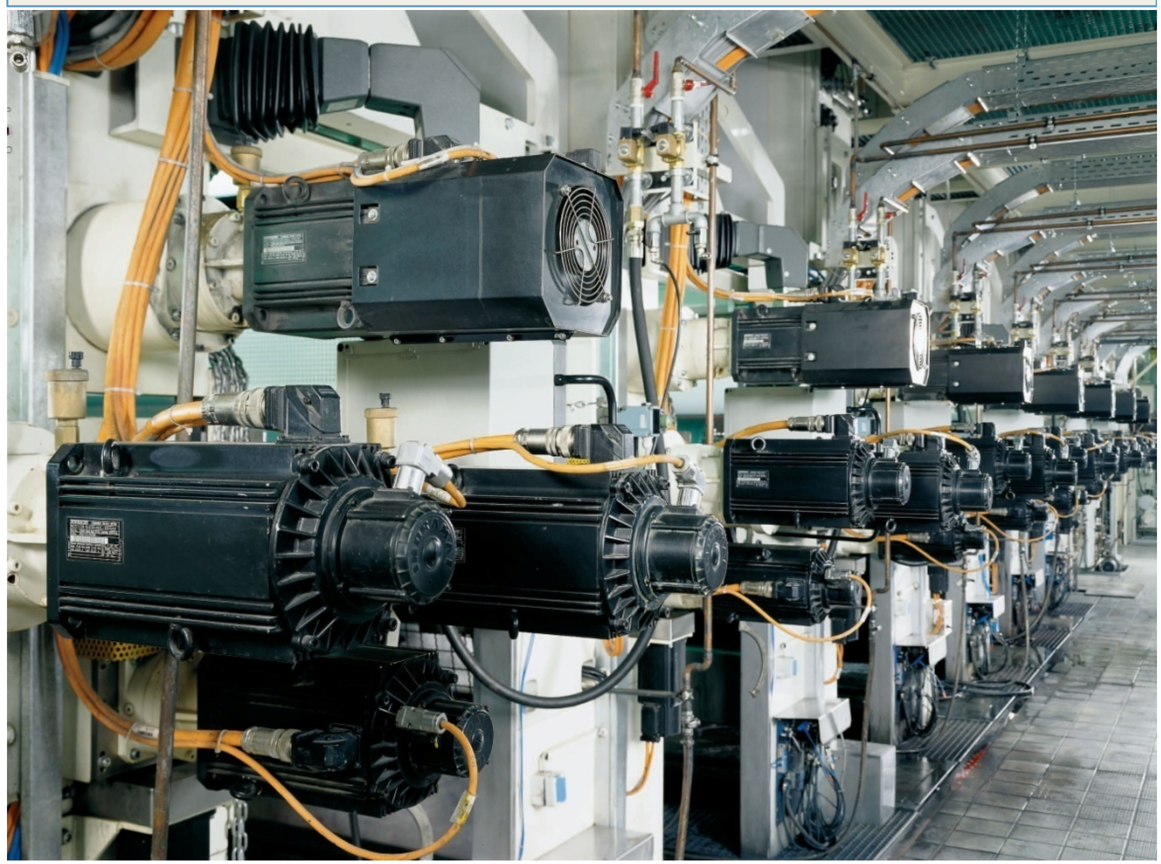
- Smart Dust (Pister, 1996-)
- Berkeley Wireless Research Center - Picoradio (Rabaey, 1998-)
- TinyOS (Culler, 1999-)
- Oceanstore (Kubiatowicz, 2000-)
- Cyber-Physical Systems (Lee, Sastry, Tomlin, 2006-)
- Wireless HART (Pister, 2006-)
- 6LoWPAN (Culler, 2007-)
- OpenWSN (Pister, 2009-)
- The Ubiquitous SwarmLab (Rabaey, 2011-)
- Software-defined buildings (Culler, 2013-)
- The TerraSwarm Research Center (2013-)
- …

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.

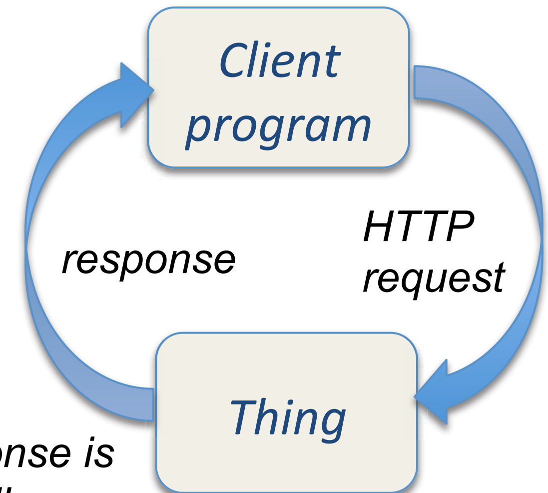


Lee, Berkeley

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) (also called the *Reactor Pattern*).

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.

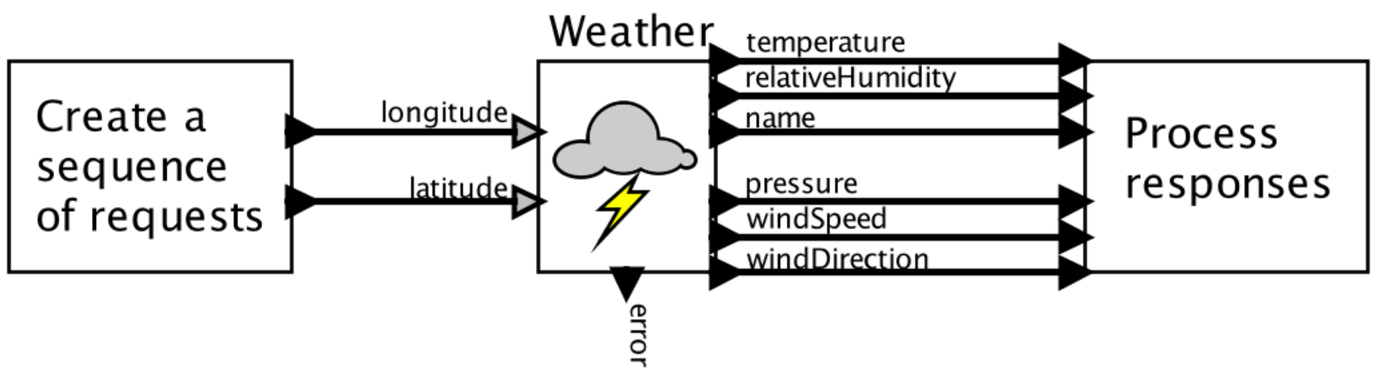
Example in JavaScript

```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:

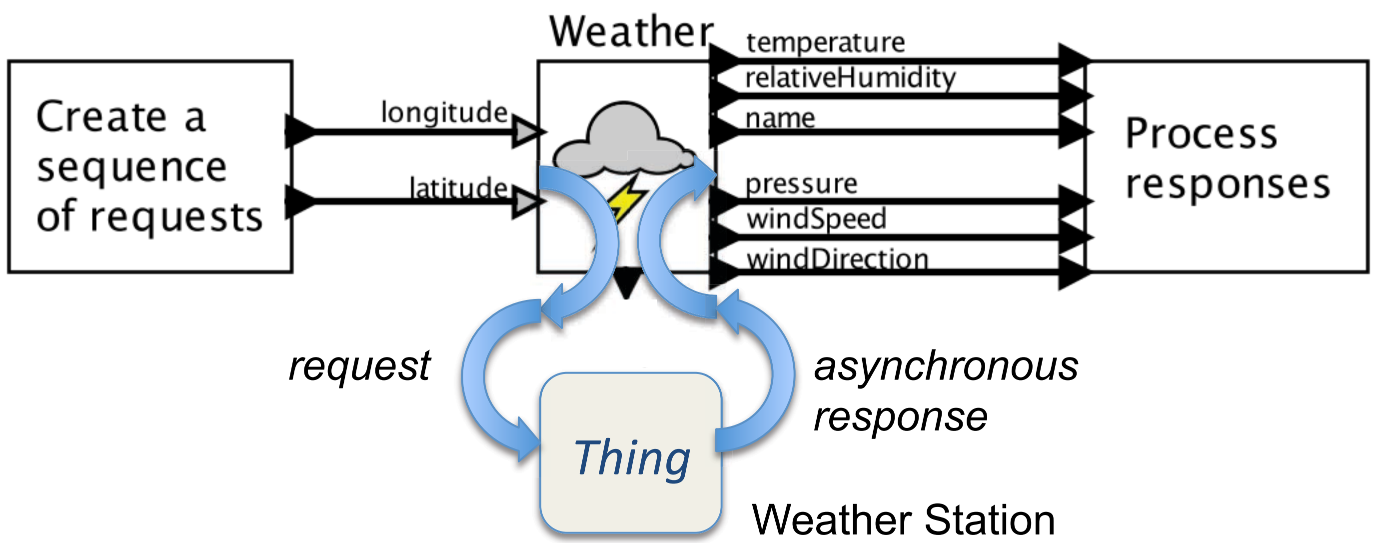


Sequence of requests for a service (a *stream*) triggers a sequence of responses.

Actors embrace concurrency and scale well.

Actors and AAC

Streaming requests:



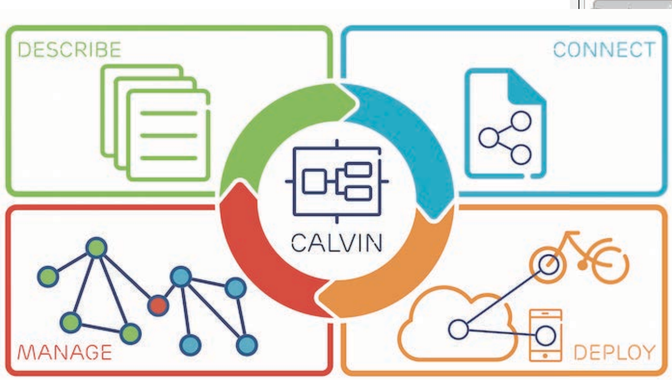
This is the essence of *accessors*, a design pattern for IoT that embraces concurrency, asynchrony, and atomicity.

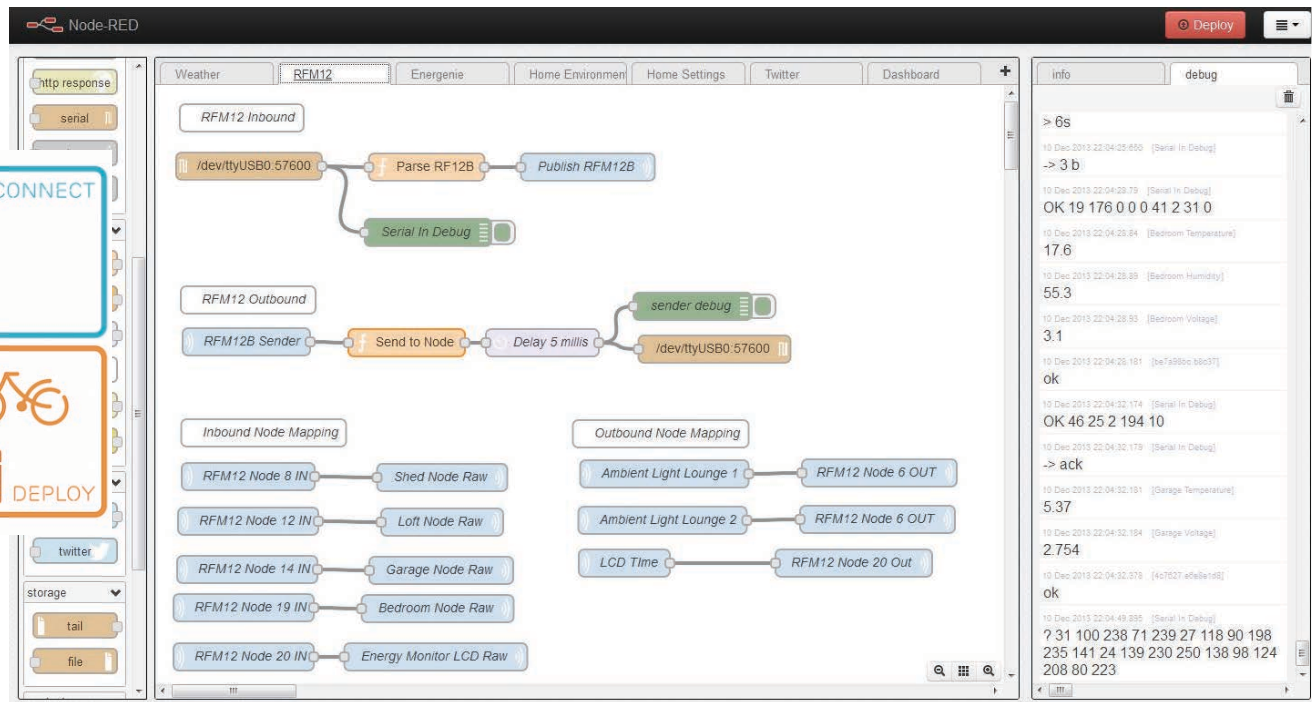
We are not alone pursuing this approach

Notable efforts:

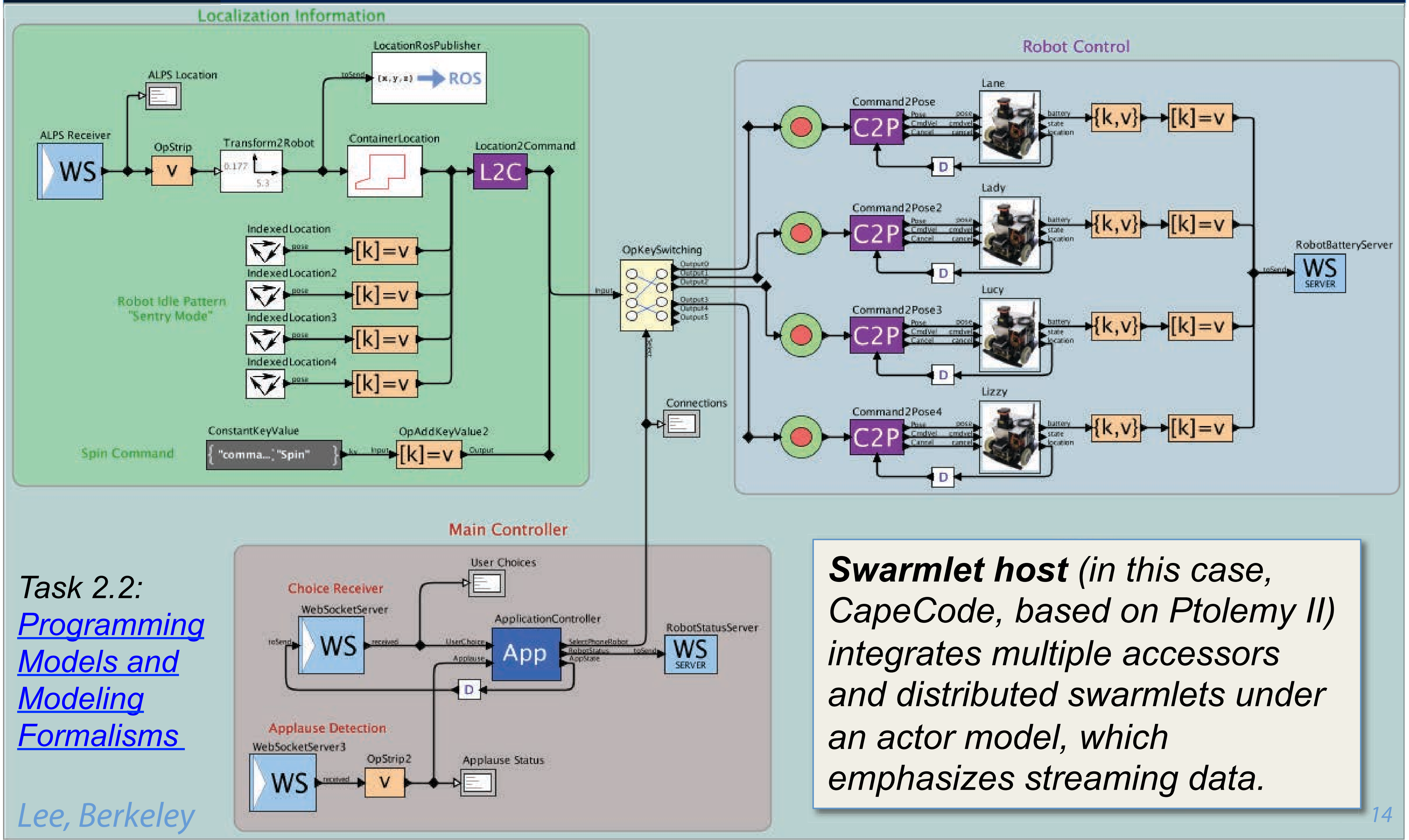
- Node Red (IBM)
- Calvin (Ericsson)

From: “Home Automation with Node Red, JeeNodes and Open Energy Monitor,” Dom Bramley's Blog of Maximo and the 'Internet of Things', IBM Developer Works, Dec., 2013.





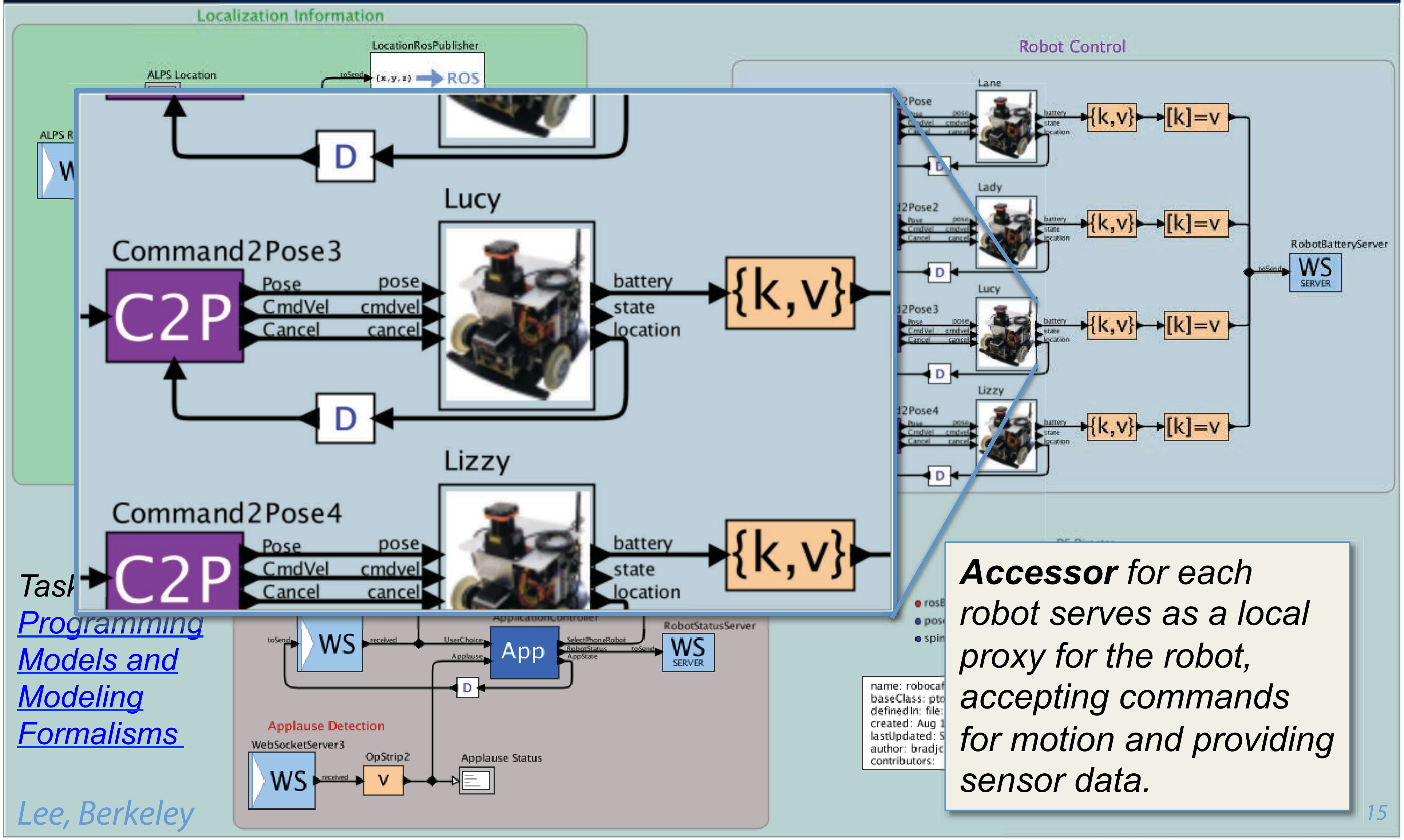
Accessors in the RoboCafe at the DARPA Wait, What? Demo [Dutta, Kumar, Lee, Rowe]



Task 2.2: [*Programming Models and Modeling Formalisms*]

Swarmlet host *(in this case, CapeCode, based on Ptolemy II) integrates multiple accessors and distributed swarmlets under an actor model, which emphasizes streaming data.*

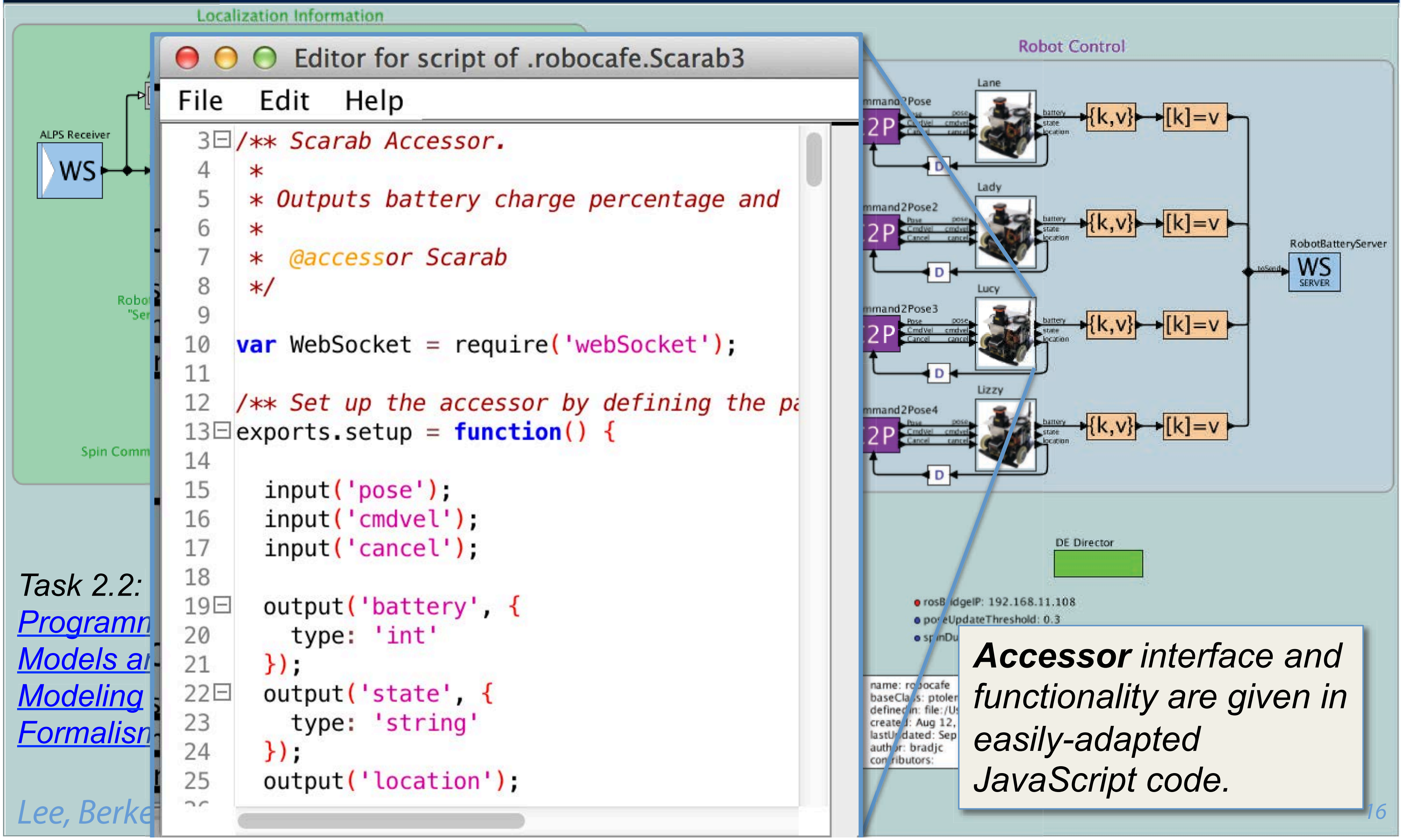
Accessors in the RoboCafe at the DARPA Wait, What? Demo *[Dutta, Kumar, Lee, Rowe]*



Task [Programming Models and Modeling Formalisms]

Accessor *for each robot serves as a local proxy for the robot, accepting commands for motion and providing sensor data.*

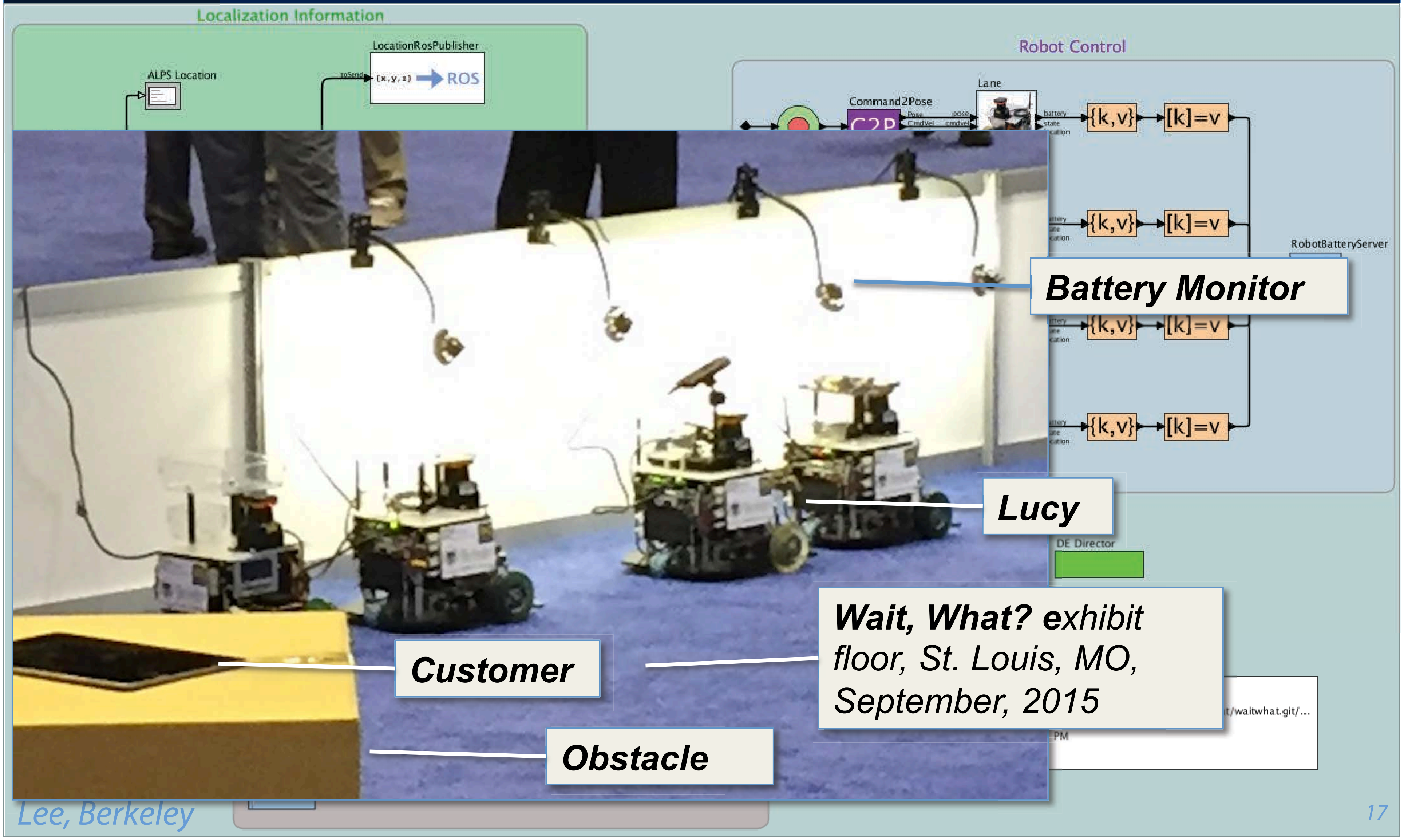
Accessors in the RoboCafe at the DARPA Wait, What? Demo *[Dutta, Kumar, Lee, Rowe]*



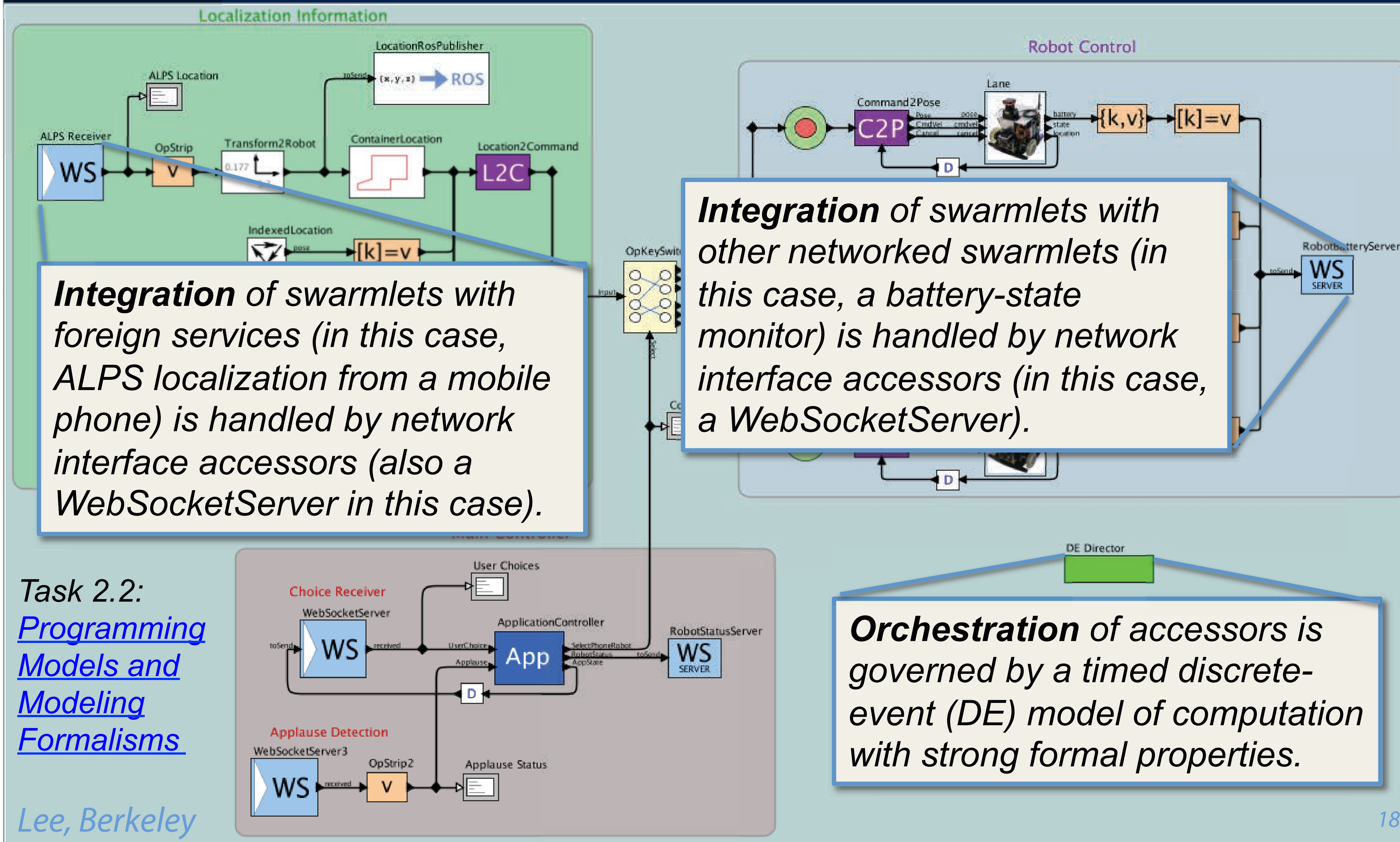
Task 2.2:
Programm
Models a
Modeling
Formalism

***Accessor** interface and functionality are given in easily-adapted JavaScript code.*

Accessors in the RoboCafe at the DARPA Wait, What? Demo *[Dutta, Kumar, Lee, Rowe]*



Accessors in the RoboCafe at the DARPA Wait, What? Demo [Dutta, Kumar, Lee, Rowe]



Integration *of swarmlets with foreign services (in this case, ALPS localization from a mobile phone) is handled by network interface accessors (also a WebSocketServer in this case).*

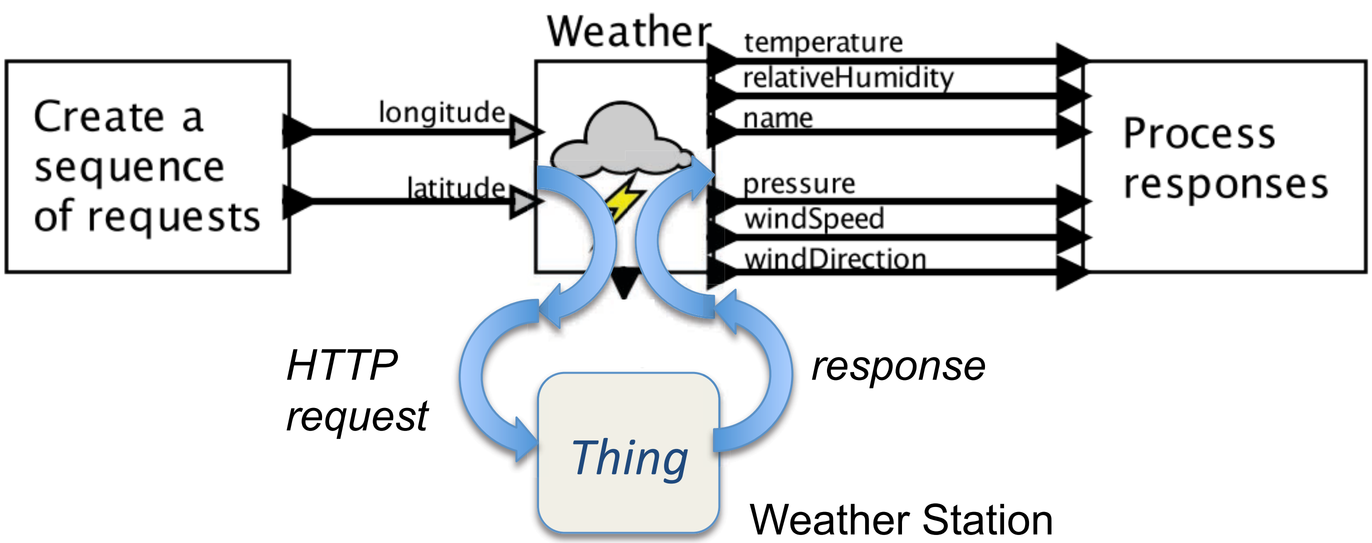
Integration *of swarmlets with other networked swarmlets (in this case, a battery-state monitor) is handled by network interface accessors (in this case, a WebSocketServer).*

Orchestration *of accessors is governed by a timed discrete-event (DE) model of computation with strong formal properties.*

Task 2.2:
Programming Models and Modeling Formalisms

Actors and AAC: Challenges

Example of a potential problem:

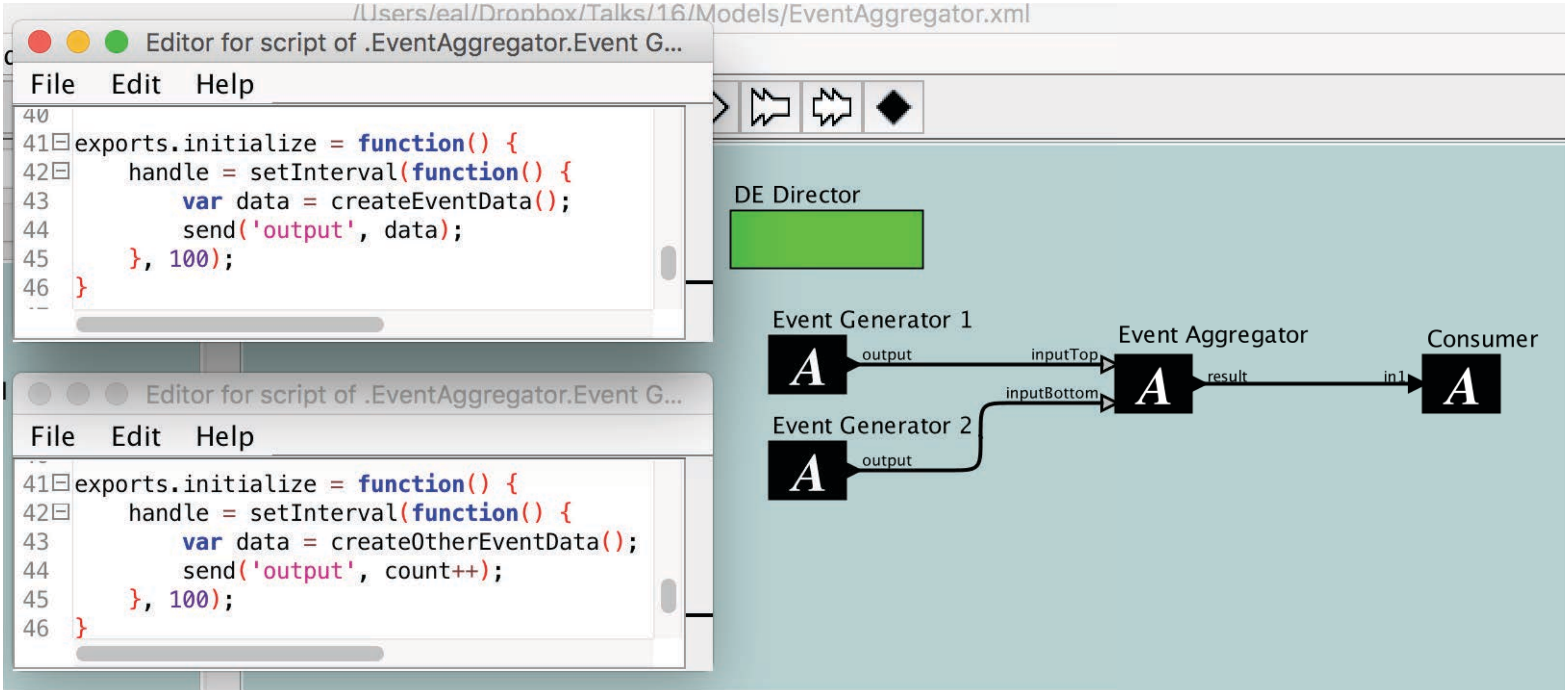


The responses may not come back in the same order as the requests!

This is a rudimentary *timing problem.*

Another Timing Problem

Coordinated timing:



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

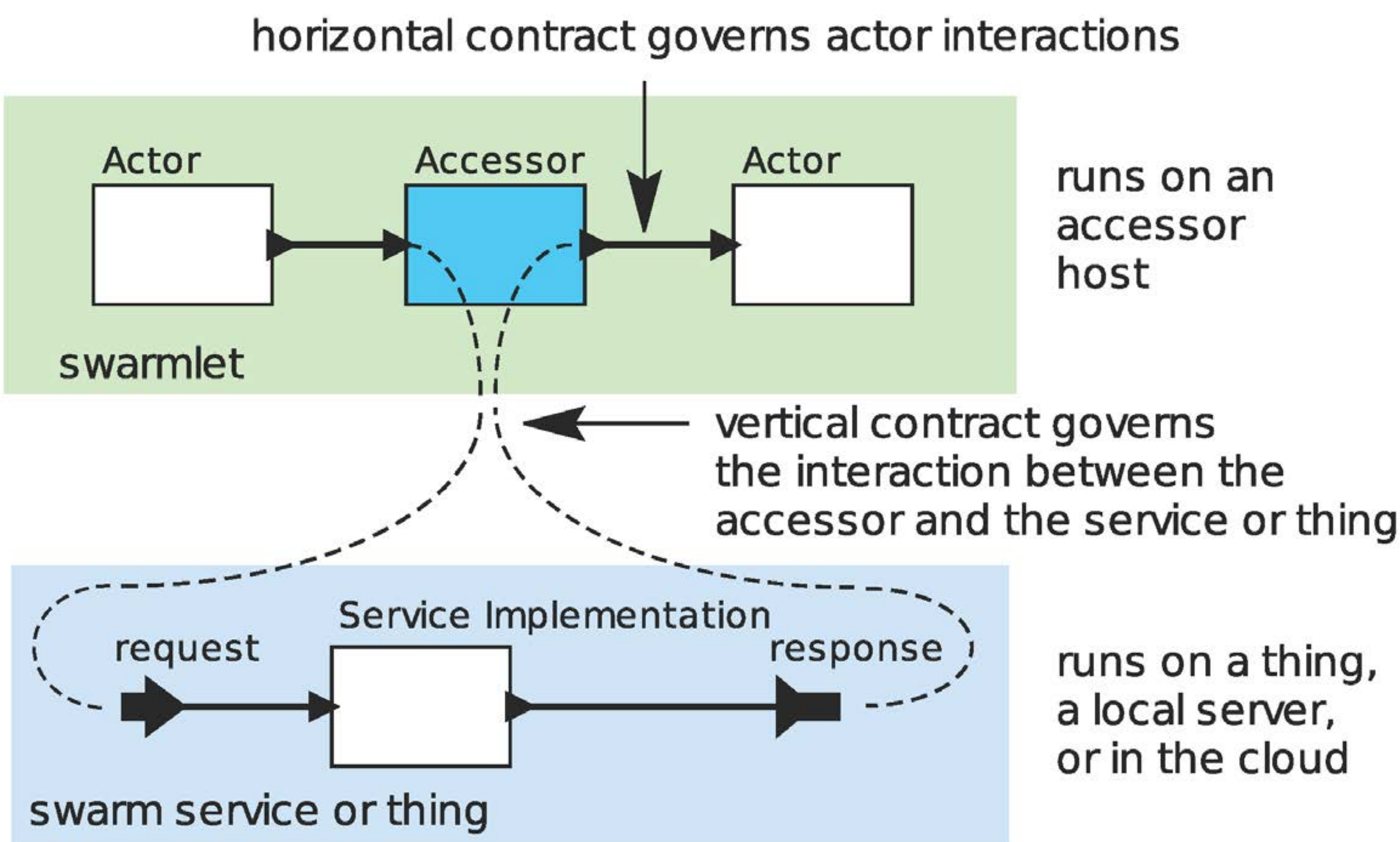
The order and timing of events matters a lot when interacting with physical processes.

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

Focus on Interfaces

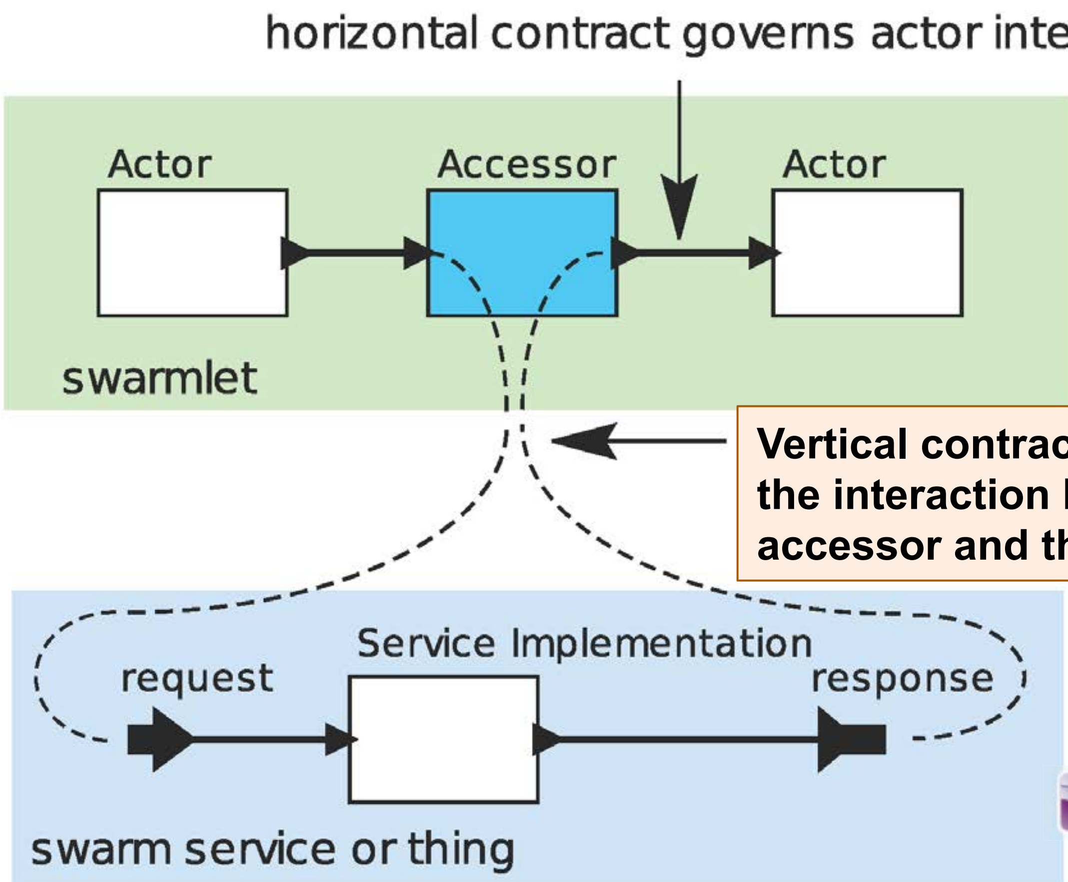


Standardization can occur with either the horizontal contract or the vertical contract.

E.g. asynchronous atomic callbacks (AAC).

Vertical Contract Standards

Focus on over-the-wire protocols

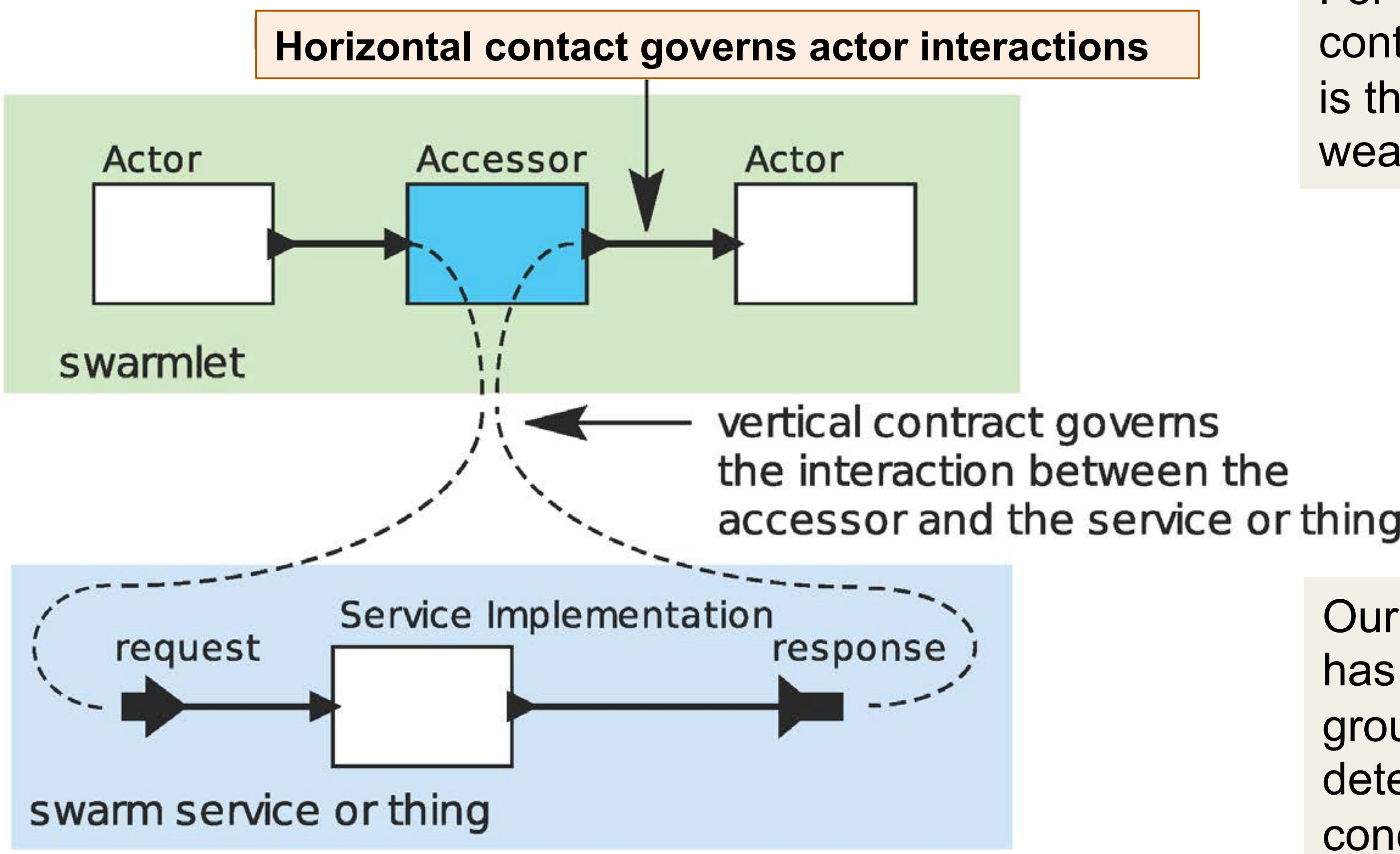


Vertical contract governs the interaction between the accessor and the service or thing.

- HTTP
- WebSockets
- CoAP
- XMPP
- MQTT
- UPnP
- DDS
- …

Horizontal Contract (Standards?)

Providing a *local proxy* for a *remote service*

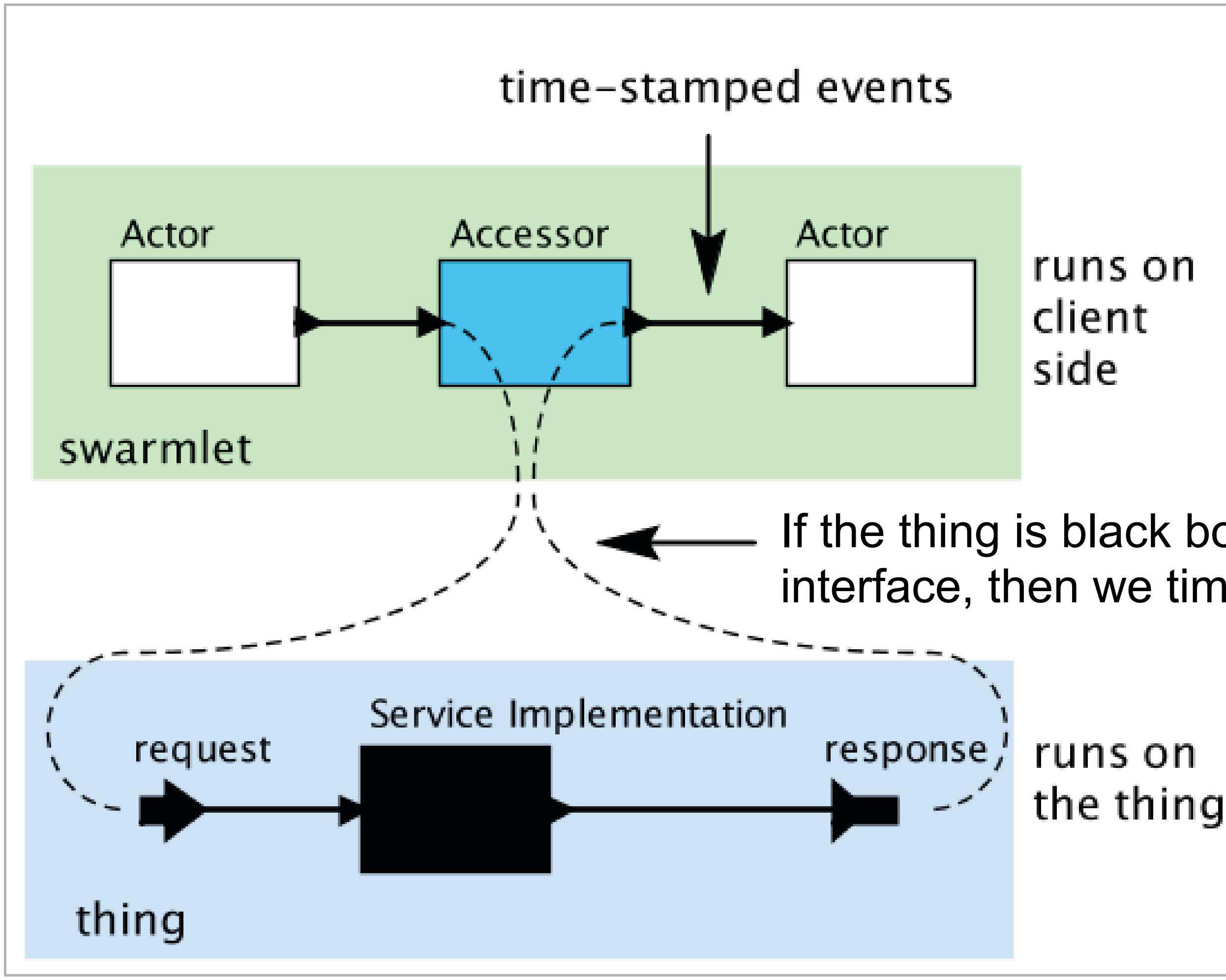


For horizontal contracts, my opinion is that current work is weak.

Our accessors work has put a stake in the ground that insists on deterministic concurrency models for composition of accessors.

Focus on the horizontal contract Discrete Event MoC

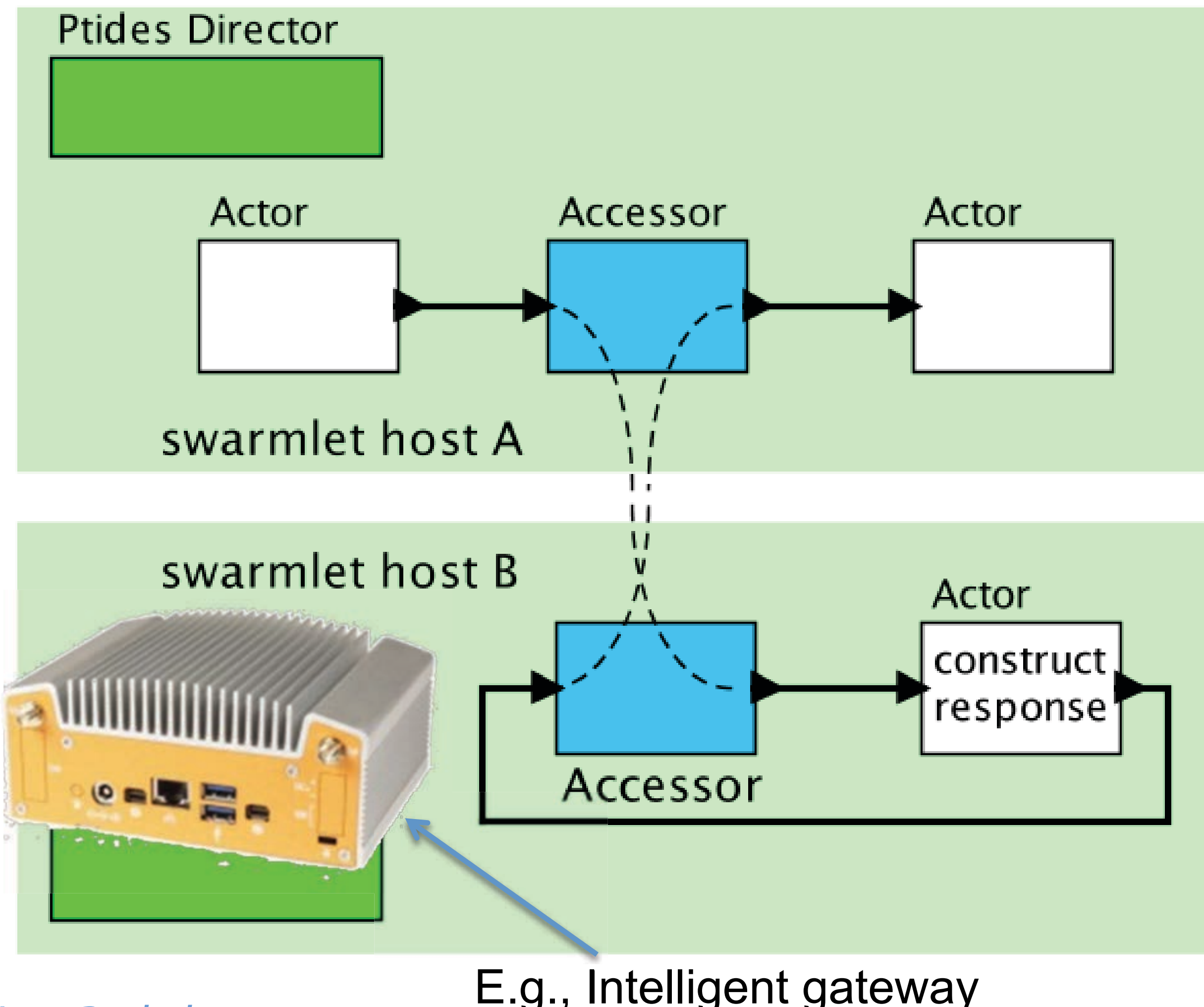
We use time-stamped events processed in time-stamp order, a discrete-event (DE) model of computation (MoC).



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

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

Distributed Swarmlets using Accessors



Leveraging time stamps and synchronized clocks, we can achieve **deterministic** distributed MoCs.

See:

- PTIDES [2007]
- Google Spanner [2012]

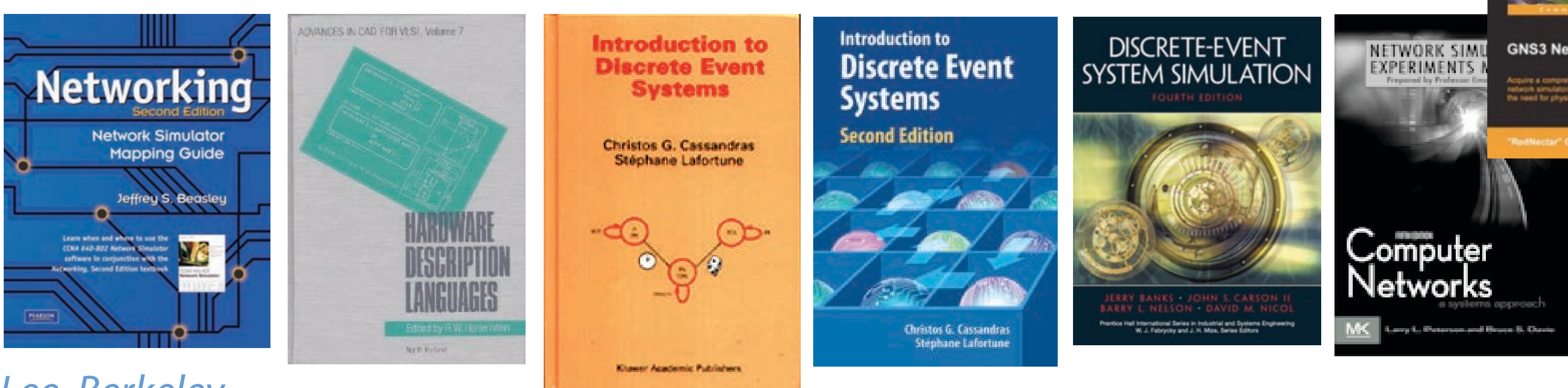
The DE MoC

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



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…

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.

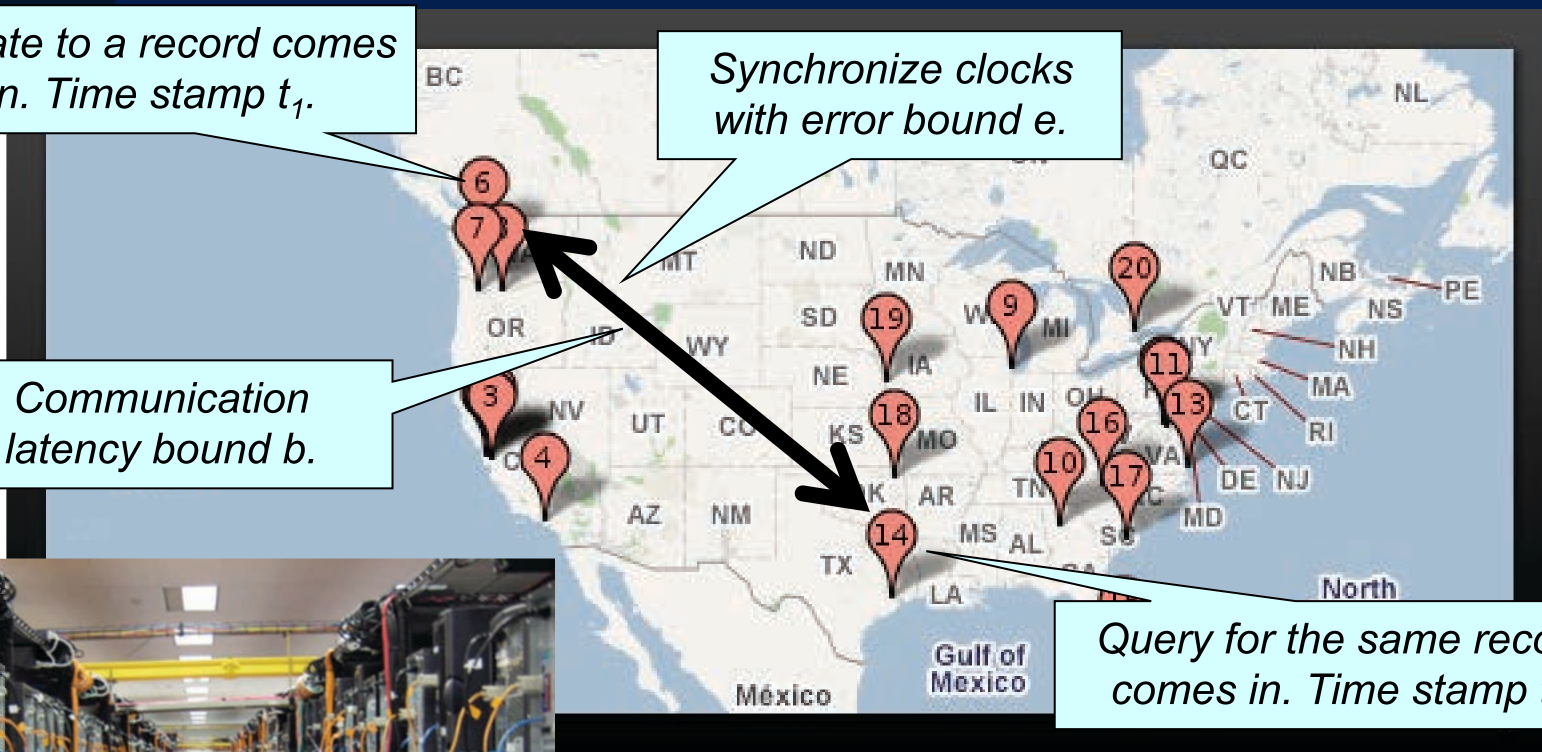
If $t_2 < t_1$, the query response should be the pre-update value. Otherwise, it should be the post-update value.

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.

Consequences of the PTIDES/Spanner Model

If *inputs* to the database system are time-stamped queries, and if the communication latency and clock synchronization bounds are respected at runtime, then the distributed database is a *deterministic DE system*.

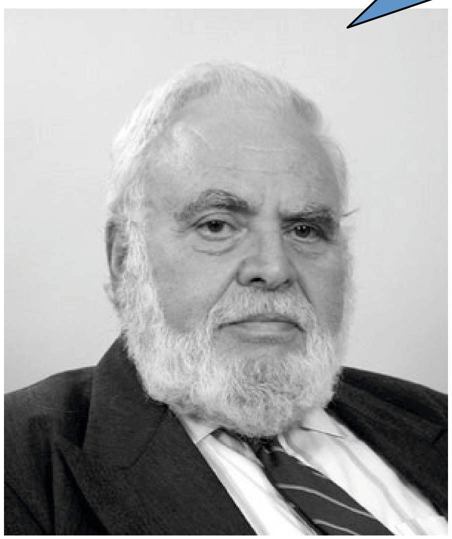
Determinism? Really?

IoIT applications operate in an intrinsically nondeterministic world.

Does it really make sense to insist on deterministic models?

Keep Clear the Distinction Between the Model and its Target

You will never strike oil by drilling through the map!



Solomon Wolf Golomb

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

Engineers all too often conflate the model with its target.

Determinacy

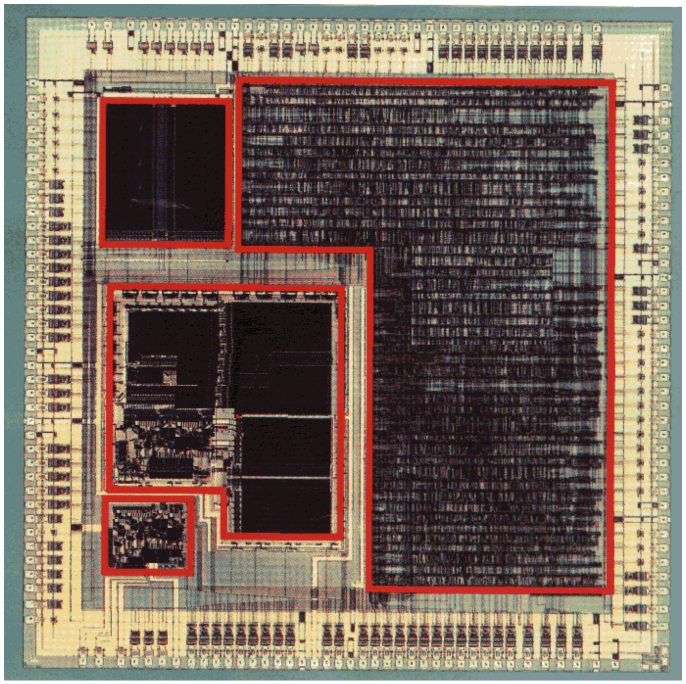
Some modeling frameworks support the construction of *deterministic* models.

A model is *deterministic* if, given the initial state and the inputs, the model defines exactly one behavior.

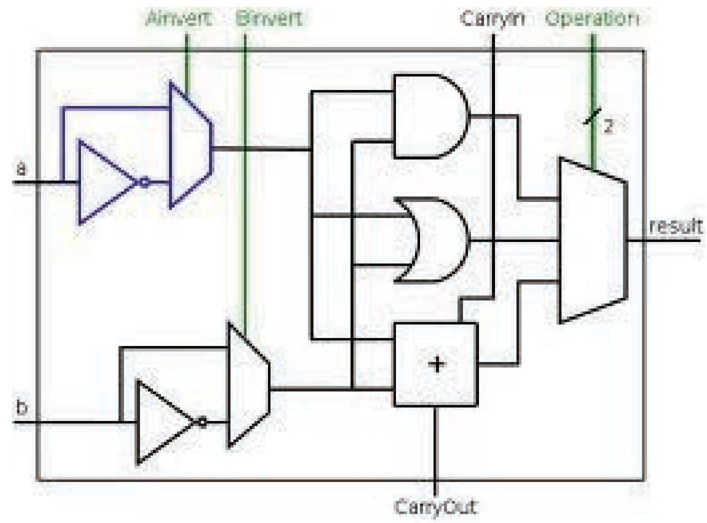
Deterministic modeling frameworks have proven extremely valuable in the past.

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

Model

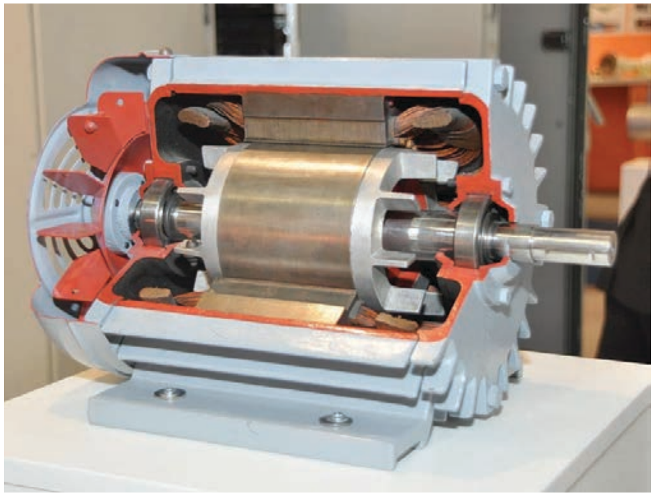
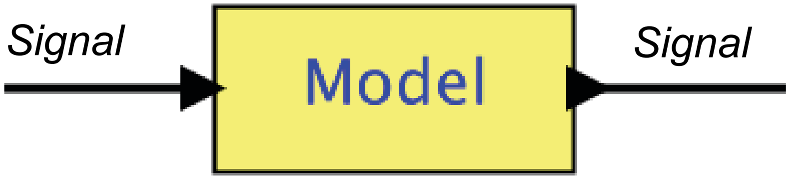


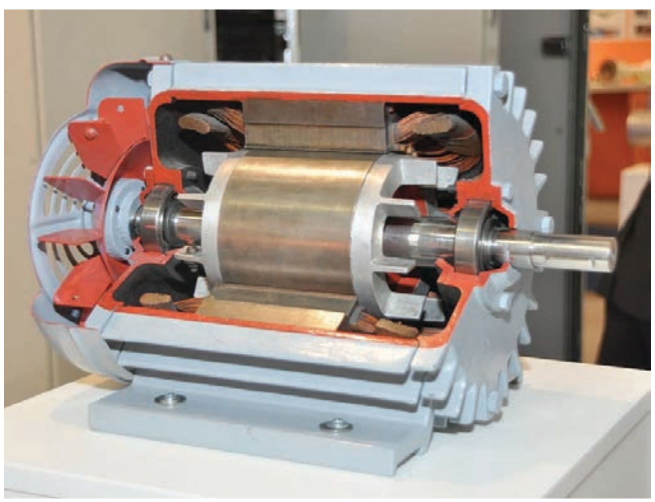
Image: Wikimedia Commons



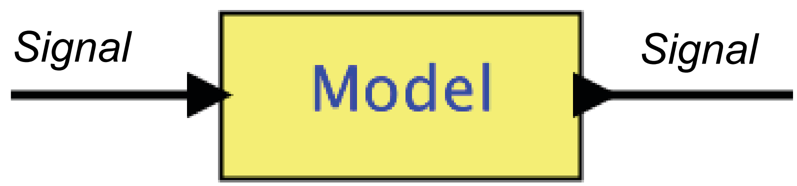
$$\dot{\mathbf{x}}(t) = \dot{\mathbf{x}}(0) + \frac{1}{M}\int_0^t \mathbf{F}(\tau)d\tau$$

Differential Equations

A Major Problem for CPS: Combinations of these Models are Nondeterministic



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



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

Image: Wikimedia Commons

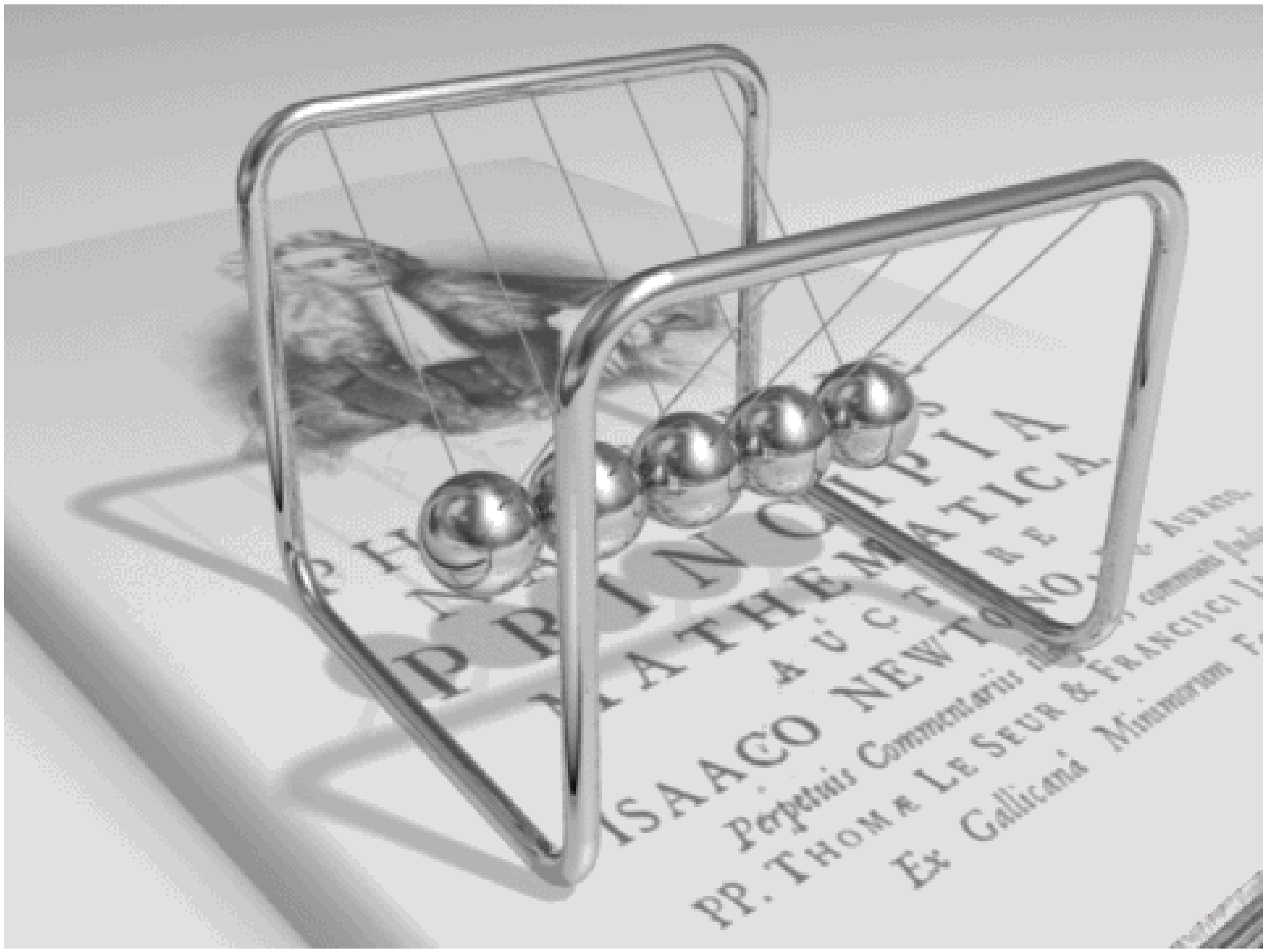
The Value of Models

- In *science*, the value of a *model* lies in how well its behavior matches that of the physical system.
- In *engineering*, the value of the *physical system* lies in how well its behavior matches that of the model.

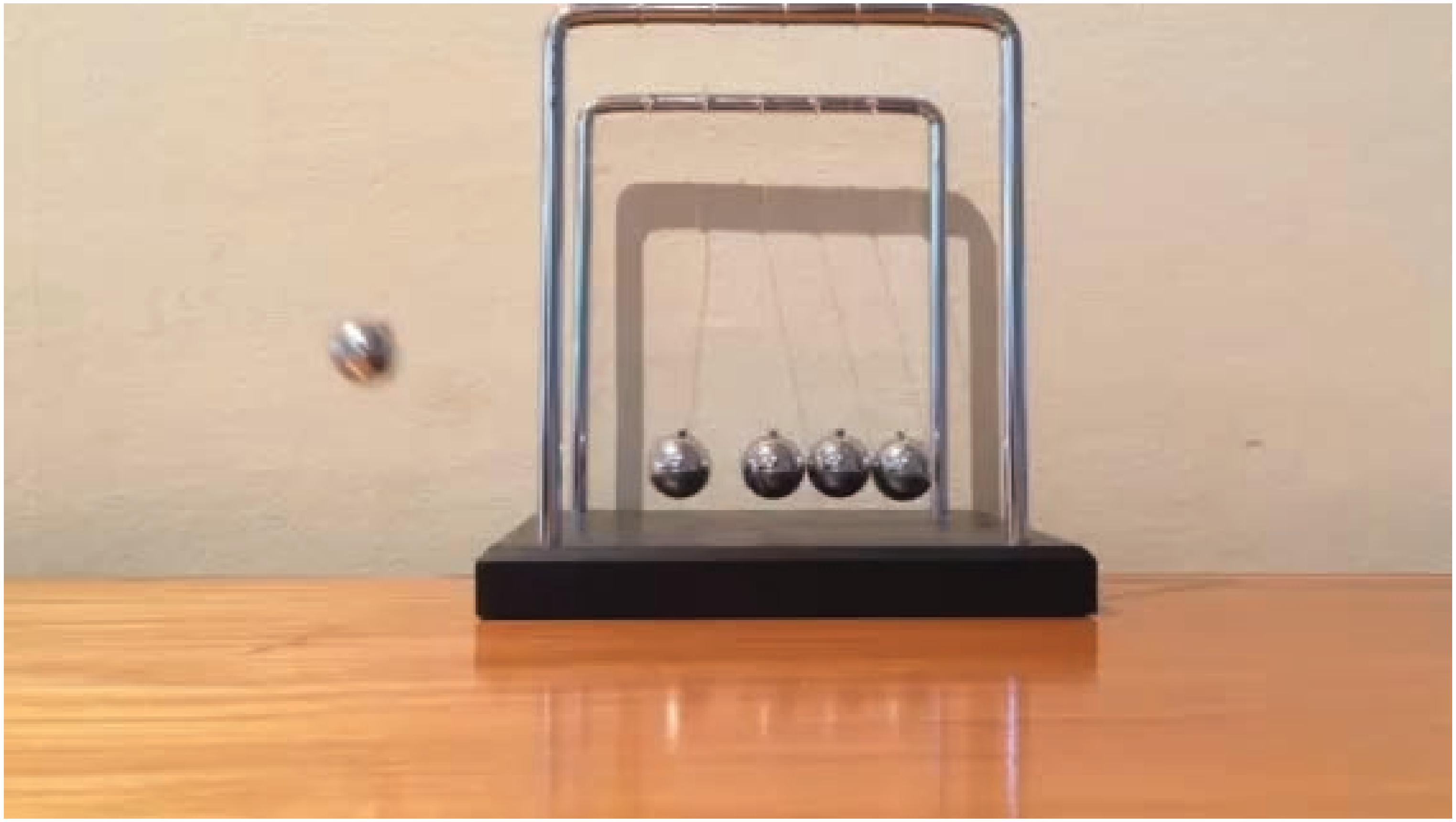
In engineering, model fidelity is a two-way street!

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…

Changing the Question

The question is *not* whether deterministic models can describe the behavior of cyber-physical systems (with high fidelity).

The question is whether we can build cyber-physical systems whose behavior matches that of a deterministic model (with high probability).

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!

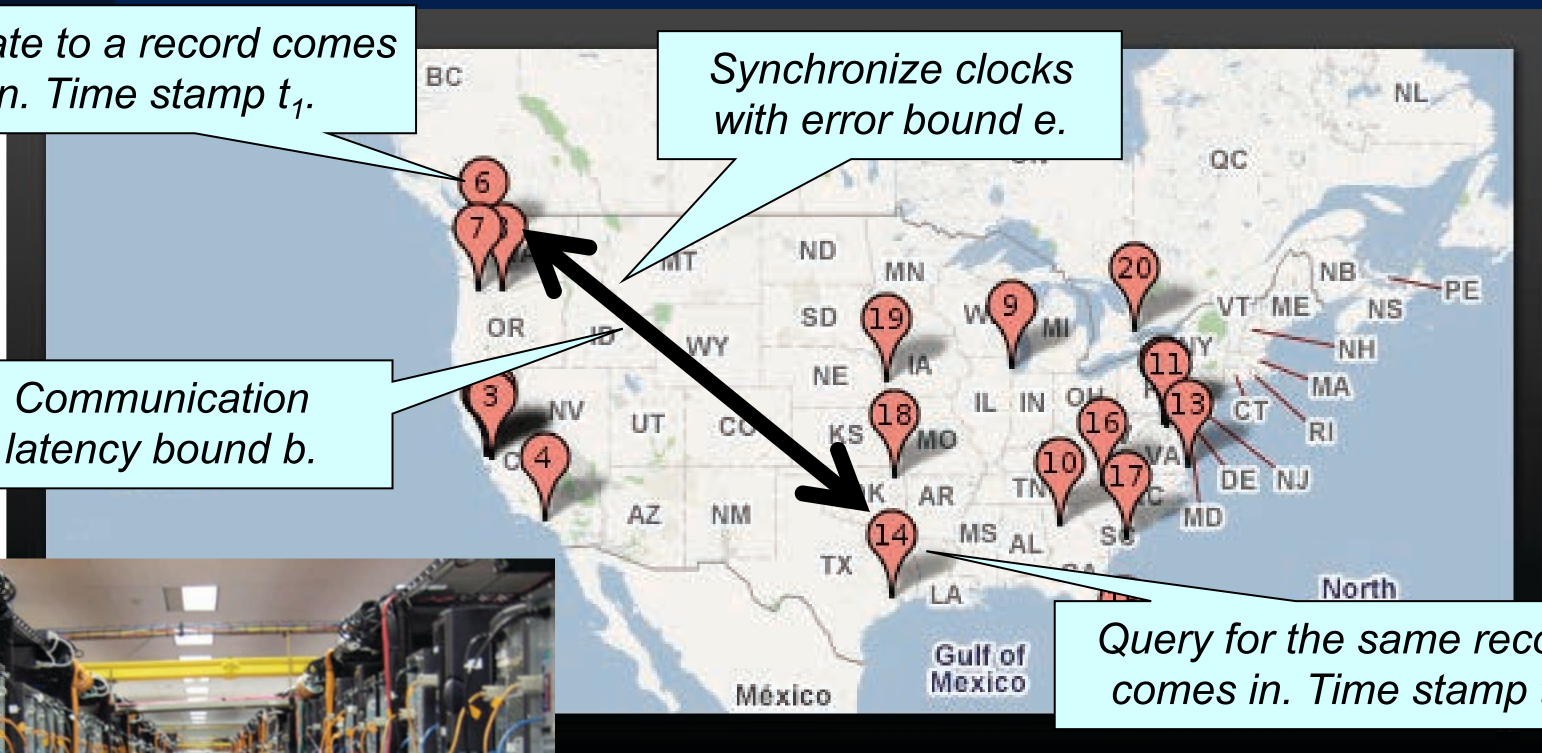
Google Spanner: Fault!

Update to a record comes in. Time stamp t_1.

Synchronize clocks with error bound e.

Communication latency bound b.

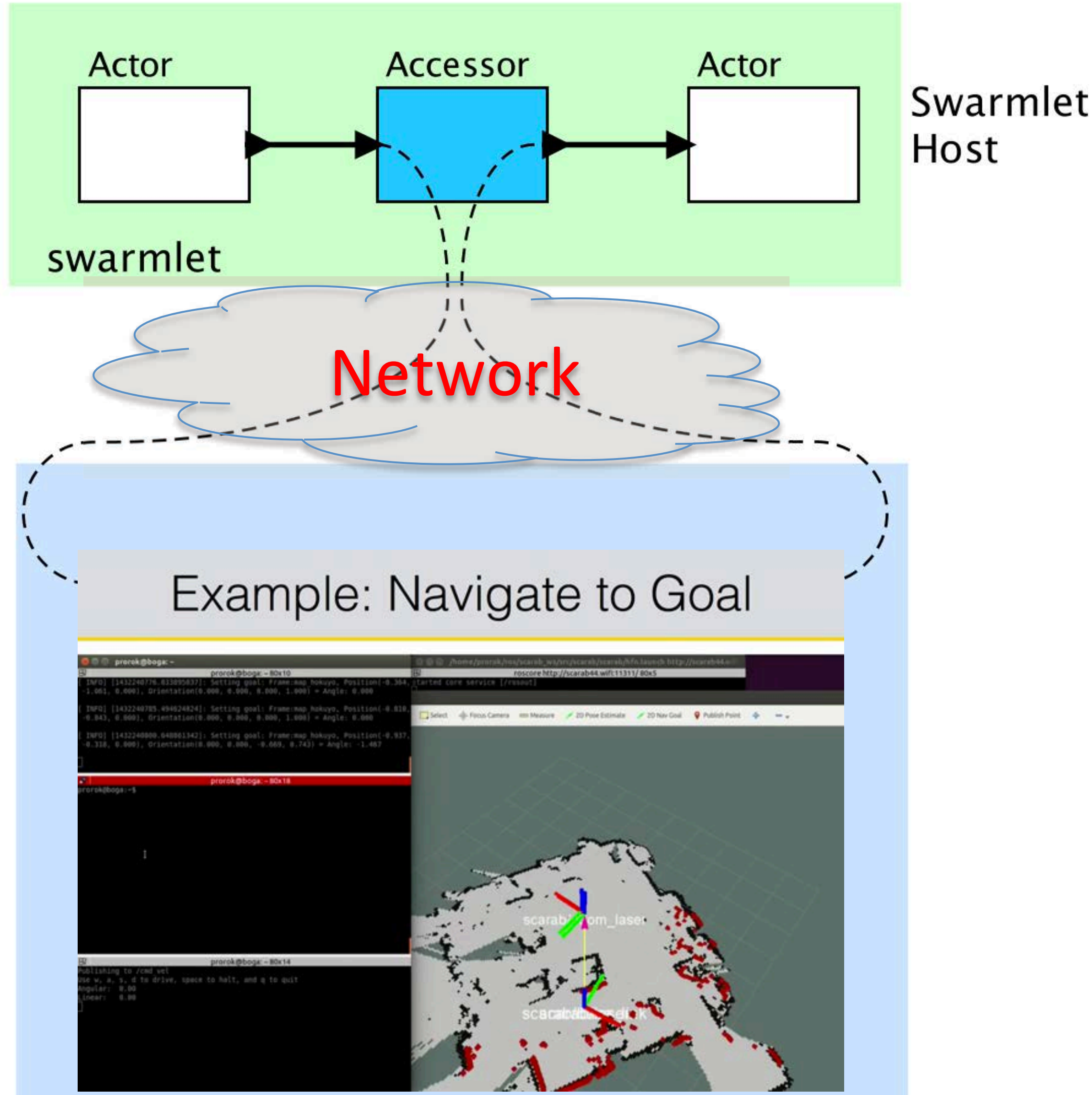
Query for the same record comes in. Time stamp t_2.



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.

Possible Evidence

The DARPA Wait, What? Demo was developed almost entirely in simulation with most components not deployed to real robots until two weeks before the demo.



Scarab robot simulation has the same interface as the robot itself

Conclusion

- IoT is not so new.
- IoIT is a really interesting problem area.
- Modern concurrency models are useful:
  - AAC (or the Reactor Pattern)
  - Actors
- They can be combined (accessors)
- But for IoIT, they beg for more determinism
- PTIDES shows that deterministic models for distributed CPS applications are practical.