A Personal View of Real-Time Computing

Edward A. Lee

Professor of the Graduate School

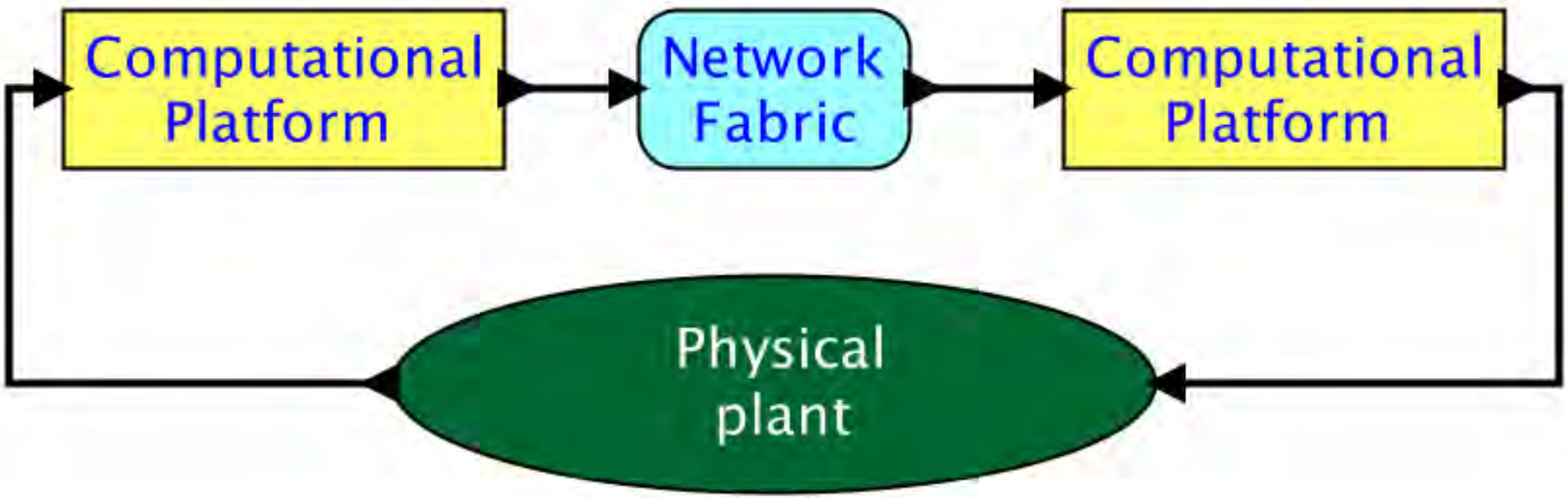
Distinguished Lecture Series

ECE Department, George Washington University,

Washington DC, April 22, 2019

University of California at Berkeley

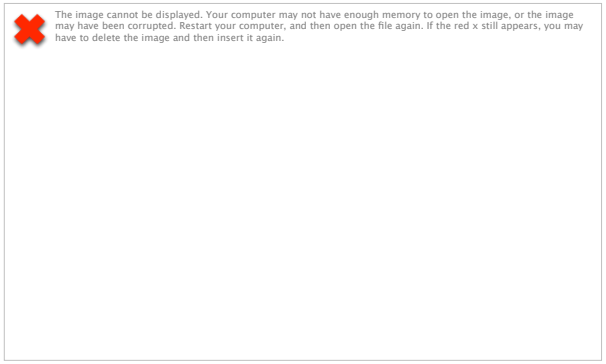
Cyber Physical Systems



Predictability requires determinacy and depends on timing, including execution times and network delays.

What is Real Time?

- fast computation
- prioritized scheduling
- computation on streaming data
- bounded execution time
- temporal semantics in programs
- temporal semantics in networks



These are very different from one another.
We have to decide which to focus on.

Achieving Real Time

- overengineering
- using old technology
- response-time analysis
- real-time operating systems (RTOSs)
- specialized networks
- extensive testing and validation


AIRBUS A380
F-WWOW
A380
787
BOEING

Achieving Real Time in Practice

- overengineering
- using old technology
- response-time analysis
- real-time operating systems (RTOSs)
- specialized networks
- extensive testing and validation

Maybe we can do better?

The Challenge: Timing is not part of Software Semantics

Correct execution *of a program in all widely used programming languages, and* ***correct delivery*** *of a network message in all general-purpose networks has nothing to do with how long it takes to do anything.*

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

Semantics is the Meaning of a Model

```
// Source code is a model:
for (int i=0; i<10; i++) {
  x[i] = a[i]*b[j-i];
  notify(x[i]);
}
```

A model is any description of a system that is not the thing-in-itself.
(*das Ding an sich* in Kantian philosophy).

MODELICA

AADL

thread group
thread
subprogram
Execution Platform
device
memory
bus
processor
package

DE Dire

MicrostepDelay
D

Ptolemy

Garage
vehicles: 10
timeDeployed: 1.0
timeBetweenDeployments: 1.0
3
out
in
pause
full

Track
length: 5.0
speed: 1.0
minimumSpacing: 1.0
capacity: 2
2

Router

Simulink

cruisecontrolonoff
File Edit View Simulation Format Tools Help
Enabled Subsystem
Sine Wave
Constant
Gain
Merge
Enabled Subsystem1
Car model (F = ma)
Desired speed (0 to 60 at time 1)
1/mass
Integrator
Integrator1
Position
Speed
Simple Cruise Control System
Ready
100%
ode45

Timing of programs emerges from the implementation

- Pipeline hazards
- Cache effects
- Variable DRAM latencies
- Speculative execution
- Interrupts
- Forwarding
- Dynamic voltage/frequency
- …

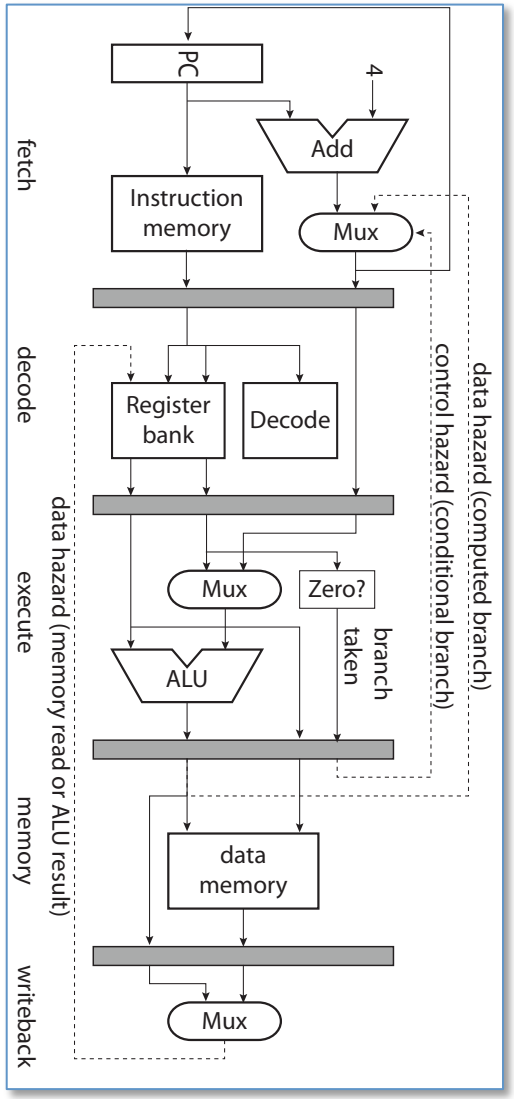
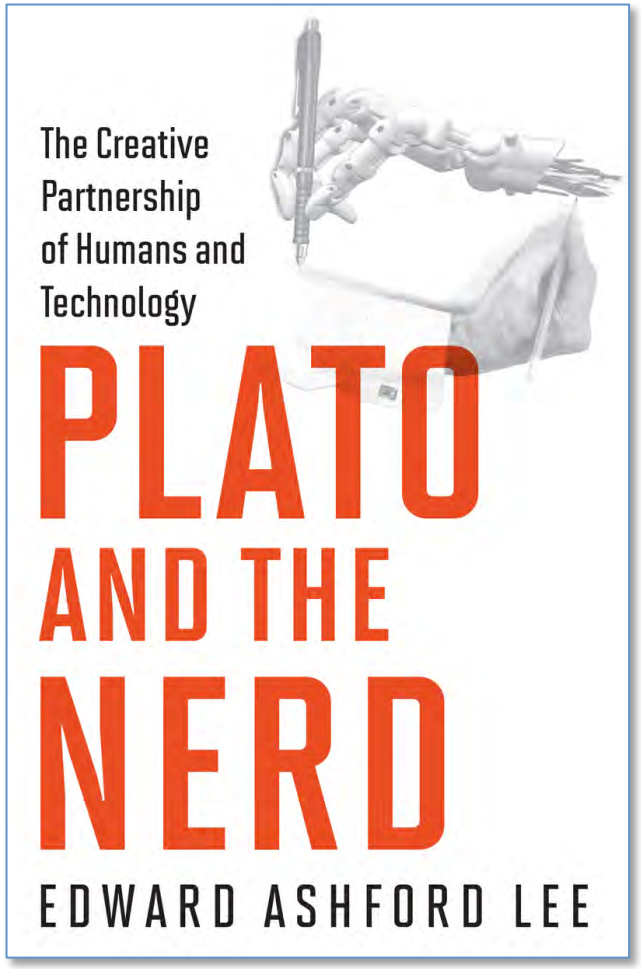


Image from Lee & Seshia,
Introduction to Embedded Systems
MIT Press, 2017

An Epiphany


The Creative
Partnership
of Humans and
Technology
PLATO
AND THE
NERD
EDWARD ASHFORD LEE


PLATO
AND THE
NERD
EDWARD ASHFORD LEE

The Value of Models

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

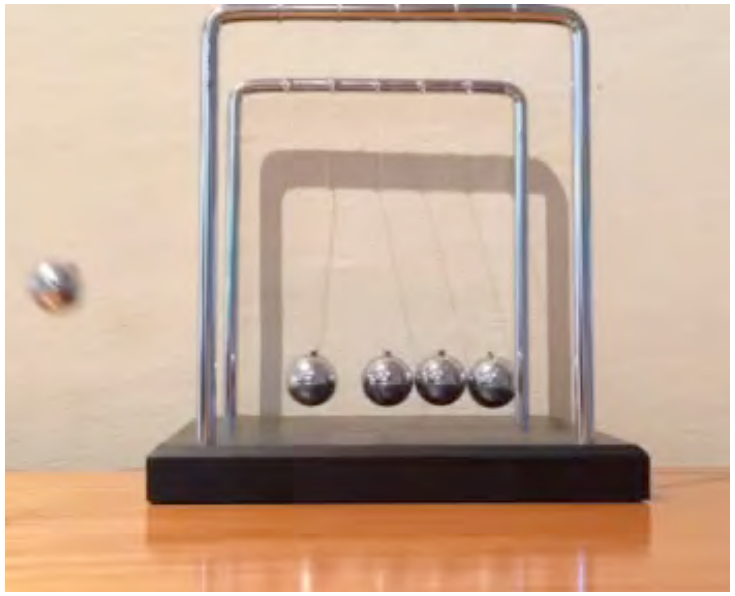
- In *engineering*, the value of the *physical system* lies in how well its behavior matches that of the model.

A scientist asks, "Can I make a model for this thing?"
An engineer asks, "Can I make a thing for this model?"

Models vs. Reality

$$x(t) = x(0) + \int_0^t v(\tau)d\tau$$
$$v(t) = v(0) + \frac{1}{m}\int_0^t F(\tau)d\tau$$

The model



The target (the thing being modeled).

In this example, the *modeling framework* is calculus and Newton's laws.

Fidelity is how well the model and its target match

A Model

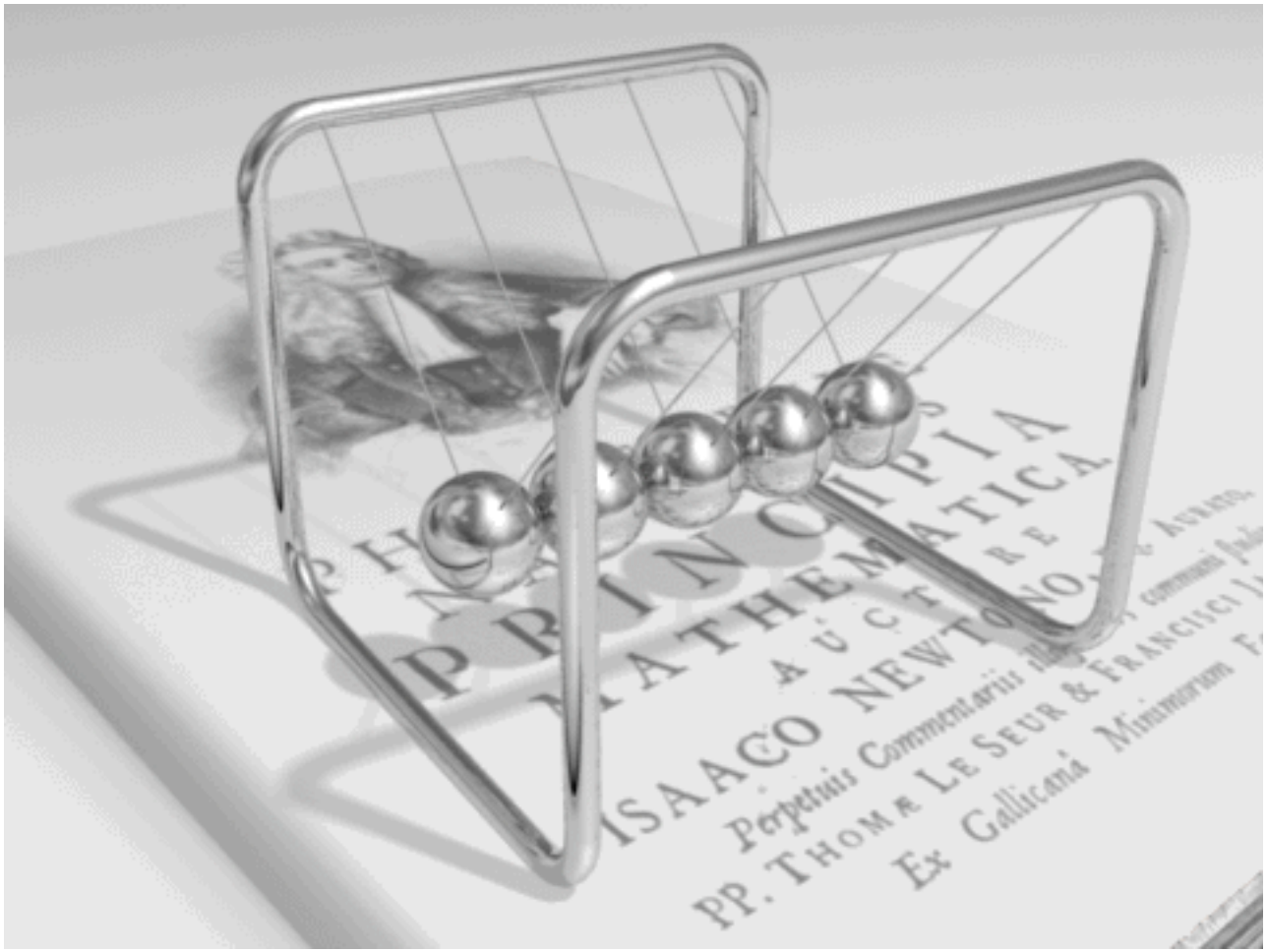
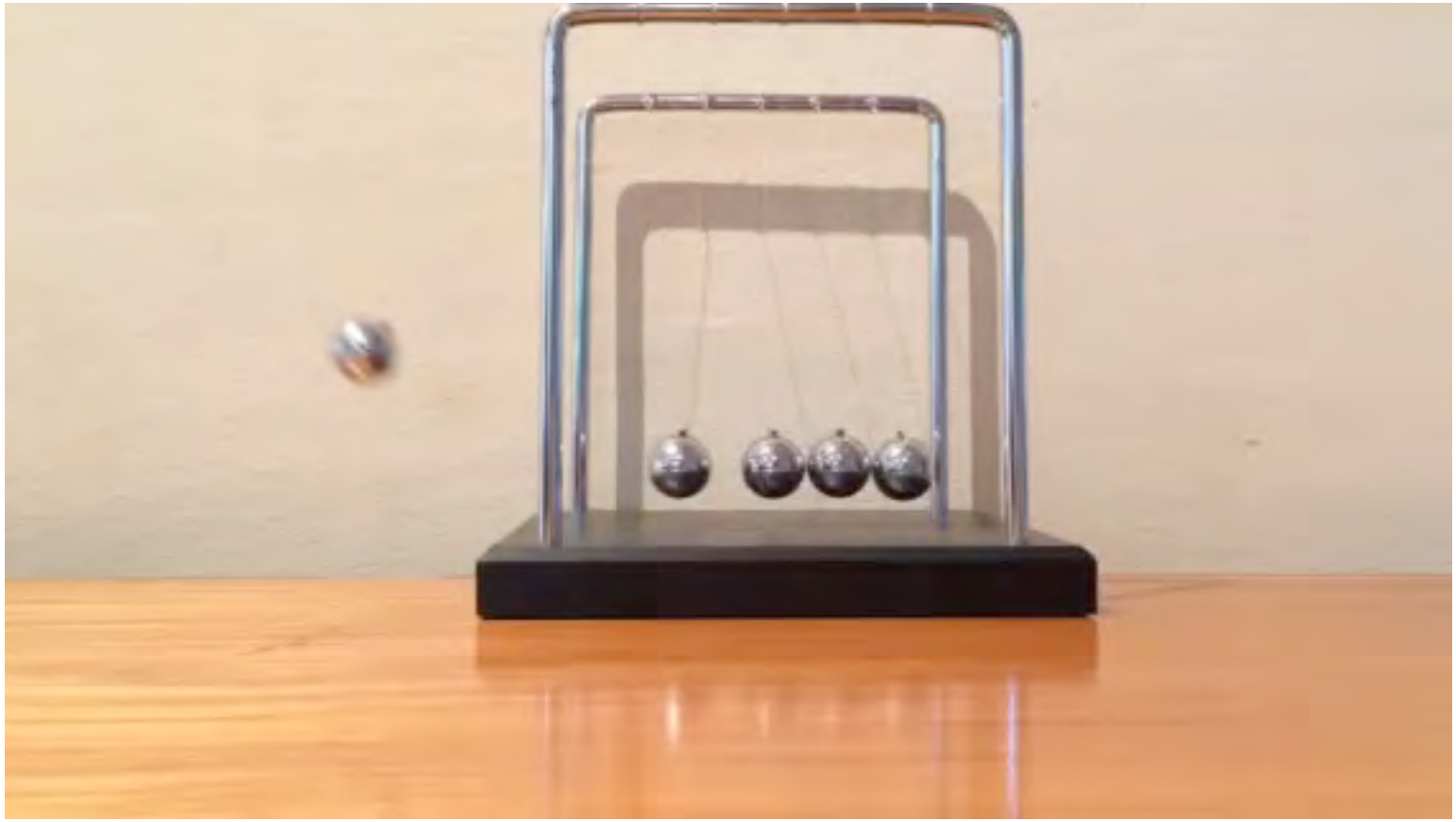


Image by Dominique Toussaint, GNU Free Documentation License, Version 1.2 or later.

A Physical Realization



Model Fidelity

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

I'm an engineer…

Useful Models and Useful Things

"Essentially, all models are wrong,
but some are useful."

Box, G. E. P. and N. R. Draper, 1987: *Empirical Model-Building and Response Surfaces*. Wiley Series in Probability and Statistics, Wiley.

"Essentially, all system implementations
are wrong, but some are useful."

Lee and Sirjani, "What good are models," FACS 2018.

The Value of Simulation

"Simulation is doomed to succeed."

[anonymous]

Could this statement be confusing engineering models for scientific ones?

Lee and Sirjani, "What good are models," FACS 2018.

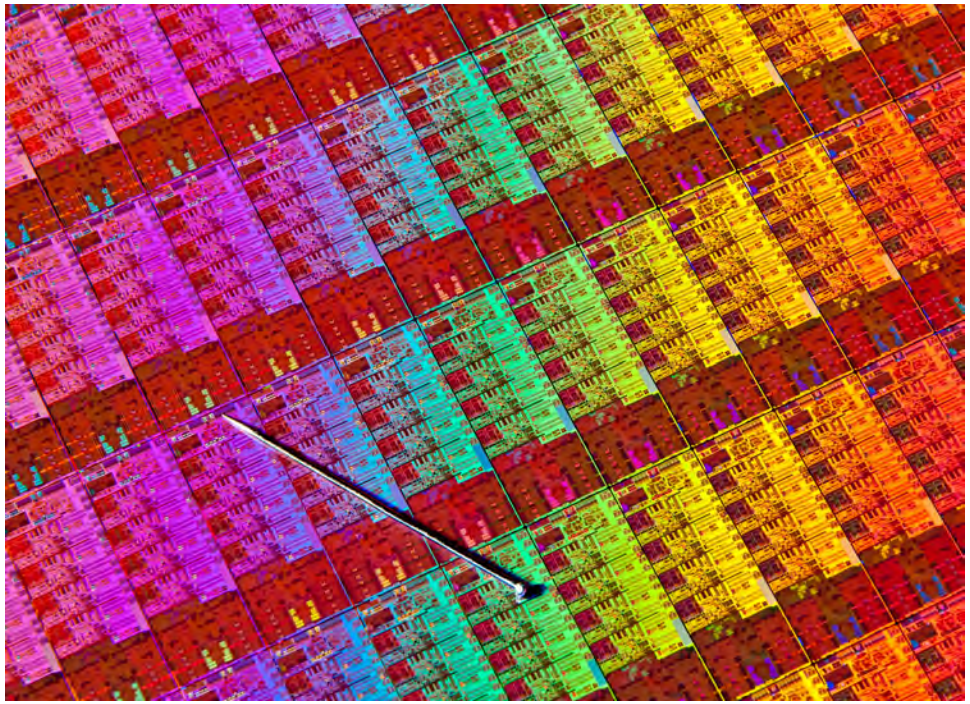
Changing the Question

Is the question whether we can build models describing the behavior of real-time systems?

Or

Is the question whether we can build real-time systems with behavior matching our models?

Consider Chip Design

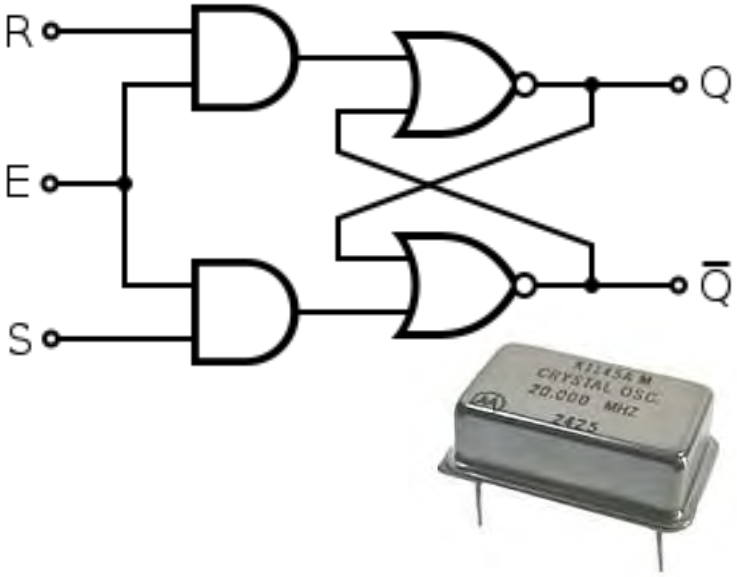


Intel Haswell, each with 1.4 billion transistors

A piece of silicon that doesn't behave like the model is just beach sand.

The hardware out of which we build computers is capable of delivering "correct" computations *and* precise timing…

Synchronous digital logic delivers precise, repeatable timing.



… but the overlaying software abstractions discard timing.

```
// 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 Software the Capabilities its Hardware Already Has.

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

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

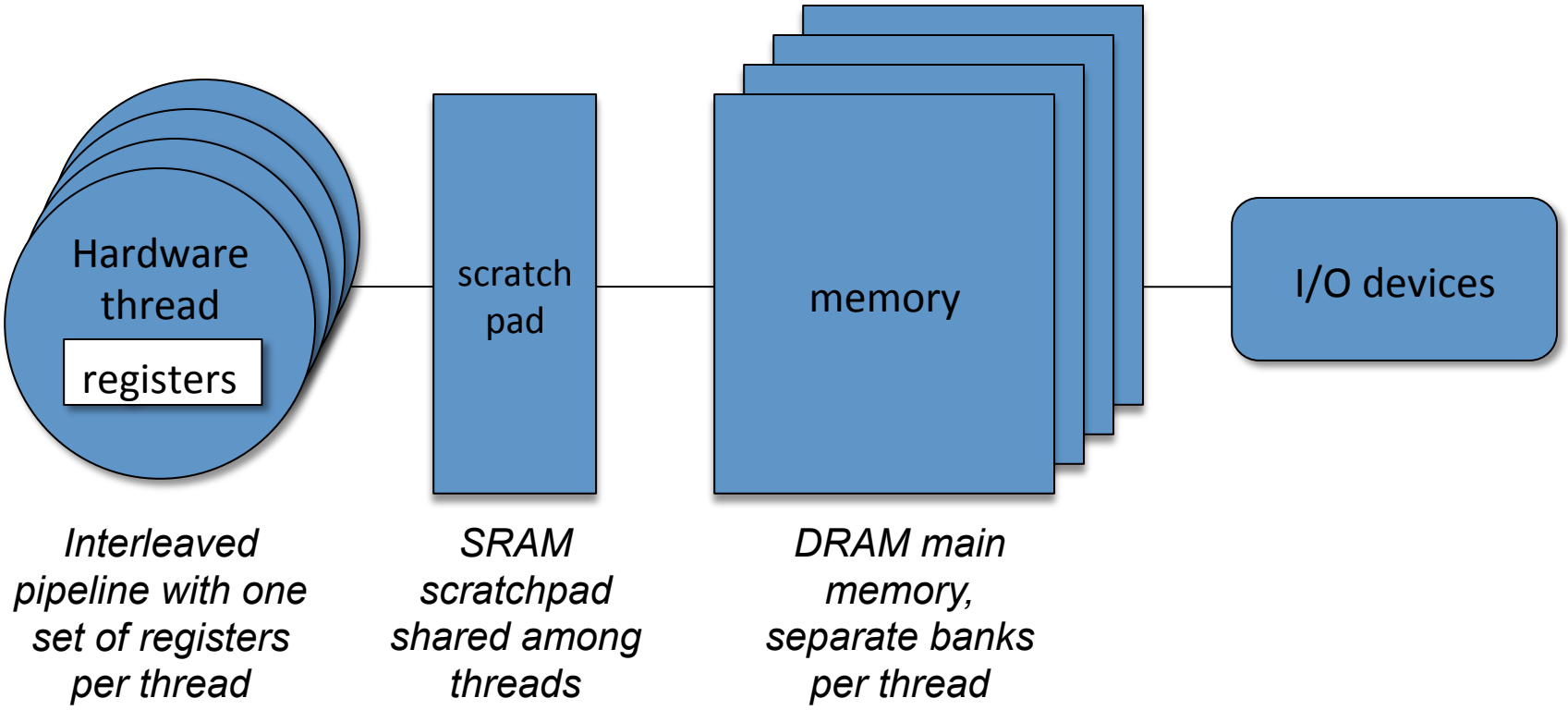
- Pipelines
  - fine-grain multithreading
- Memory hierarchy
  - memory controllers with controllable latency
- I/O
  - threaded interrupts with zero effect on timing

Three Generations of PRET Machines at Berkeley

- PRET1, Sparc-based (simulation only)
  - [Lickly et al., CASES, 2008]
- PTARM, ARM-based (FPGA implementation)
  - [Liu et al., ICCD, 2012]
- FlexPRET, RISC-V-based (FPGA + simulation)
  - [Zimmer et al., RTAS, 2014, PhD Thesis 2015]

Our Second Generation PRET

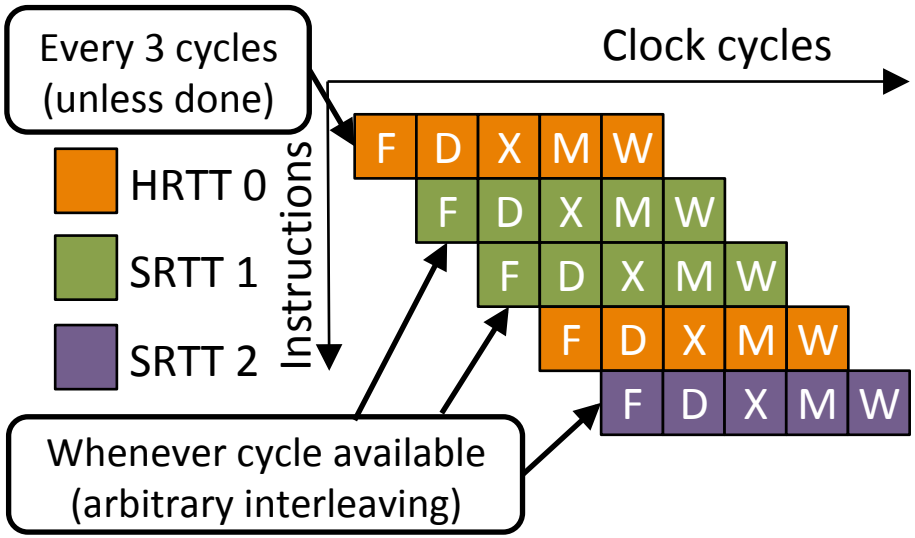
PTArm, a soft core on a
Xilinx Virtex 5 FPGA (2012)



Isaac Liu, PhD Thesis, 2012

Our Third-Generation PRET: Open-Source FlexPRET (Zimmer 2014/15)

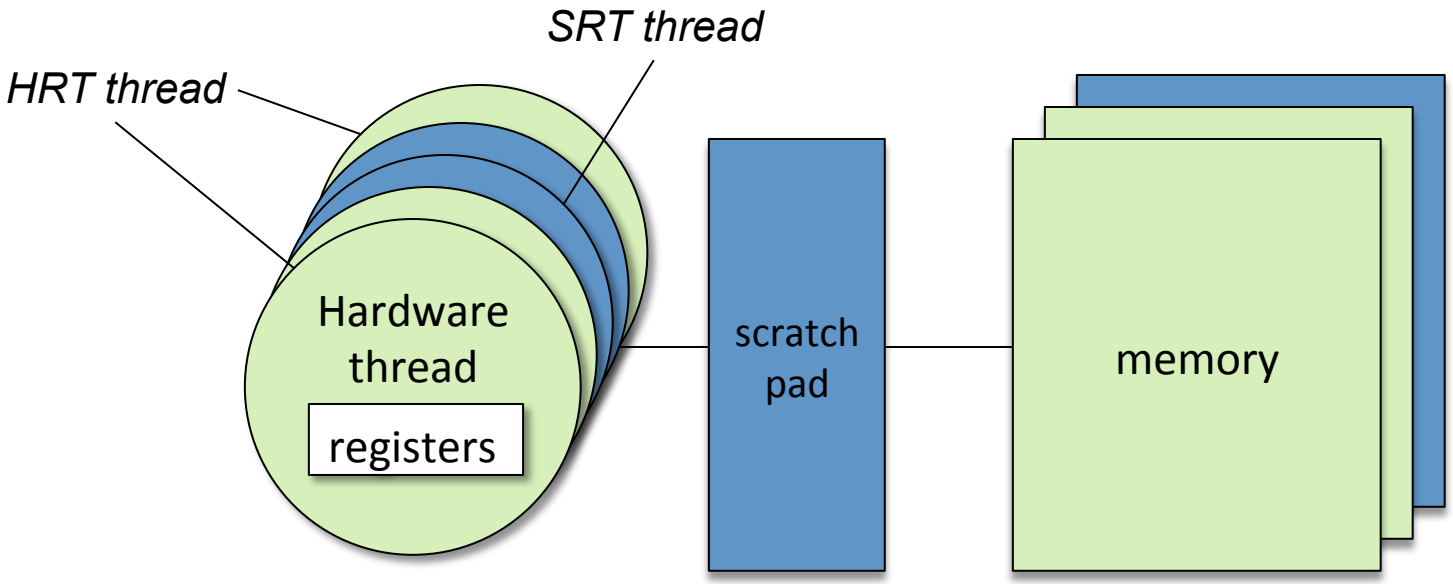
- 32-bit, 5-stage thread interleaved pipeline, RISC-V ISA
  - **Hard real-time HW threads**: scheduled at constant rate for isolation and repeatability.
  - **Soft real-time HW threads**: share all available cycles for efficiency.
- Deployed on Xilinx FPGA



Digilent Atlys (Spartan 6) and NI myRIO (Zync)

FlexPRET

Hard-Real-Time (HRT) Threads
Interleaved with Soft-Real-Time (SRT) Threads



HRT threads have deterministic timing. SRT threads share remaining cycles

SRAM scratchpad shared among threads

DRAM main memory provides deterministic latency for HRT threads. Conventional behavior for the rest.

Fact

The real-time performance of a FlexPRET machine is never worse than that of a conventional machine.

Proof: A FlexPRET machine *is* a conventional machine if the memory-mapped registers controlling HRT and SRT threads is set to have only one thread, a SRT thread.

The Cost (Worst Case)

A **baseline RISC-V** without any complex instructions (floating point, integer division, packed instructions) can be realized on an FPGA with 580 flip flops and 2,788 LUTs.

A **4-thread FlexPRET** can be realized with 908 flip flops and 3,943 LUTs, an increase of 56% and 41% respectively.

Percentage is much lower with floating point, division, etc.

[Zimmer, Broman, Shaver, Lee, RTAS 2014]

About Interrupts

"[M]any a systems programmer's grey hair bears witness to the fact that we should not talk lightly about the logical problems created by that feature"

- Edsger Dijkstra (1972)

Interrupts

- Nondeterministically interleaved with program
- Make response time > execution time
- Disrupt cache and branch predictors
- Overhead of context switching

- For WCET analysis, have to disable interrupts
- Disabling interrupts increases variability in response time

Interrupts

Scientific solution:

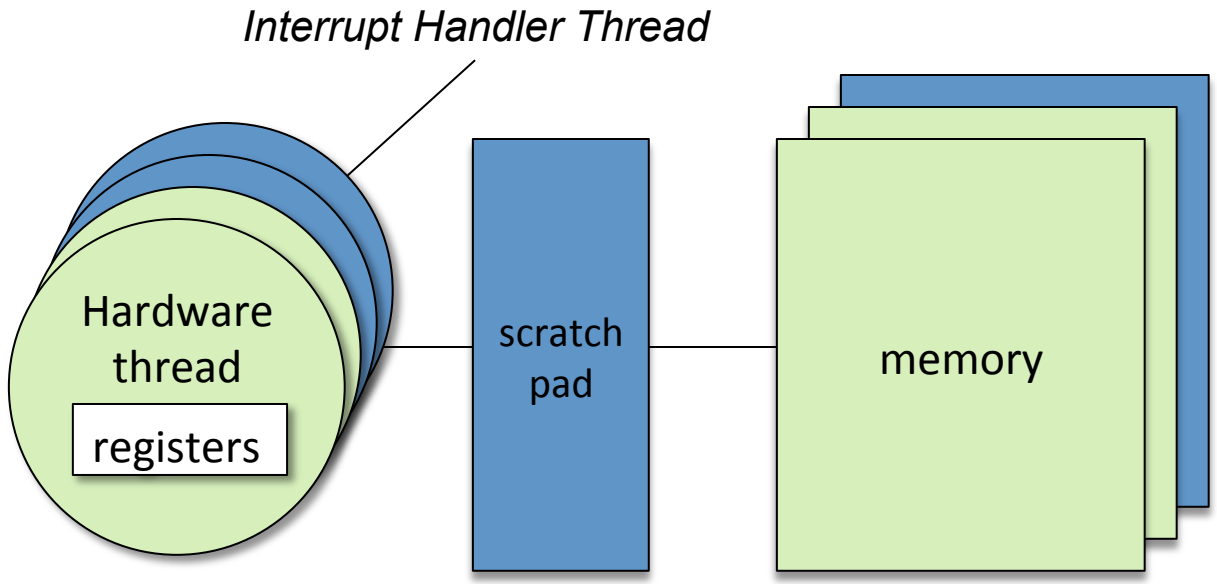
- Model all these effects

Engineering solution:

- Eliminate all these effects

The latter is what PRET machines do.

FlexPRET I/O
Interrupt Handler Thread Option



Such interrupts have
no effect on HRT threads, and
bounded effect on SRT threads!

A similar strategy is also used by XMOS, but with less isolation.

Abstract PRET Machines (APM)

Abstract PRET Machines

Invited TCRTS award paper

Edward A. Lee (award recipient) Jan Reineke Michael Zimmer

RTSS, 2017, Paris.

This paper shows that achieving deterministic response times that meet deadlines, when that is feasible, comes at *no cost* in worst-case response times.

This is shown for a task model of *N* sporadic independent tasks with deadlines.

Intuition

- *N* sporadic real-time tasks with minimum interarrival time T_i, deadlines D_i, and WCET C_i.

Theorem: When $T_i = D_i$, PRET yields deterministic response times no worse than the worst case response time of a conventional architecture.

When $T_i > D_i$, if any processor can deliver deterministic response times, PRET will, with worst case response time no worse than a conventional architecture.

Benefits of PRET
(Even if you don't care about determinism)

- **Very low context switch overhead**
  - Up to the number of hardware threads.
  - Conventional overhead above that.
- ***Improved* performance**
  - Can eliminate pipeline bubbles.
- **High-precision timing instructions**
  - Nanoseconds of precision are possible.
- **Tighter execution-time analysis**
  - Especially with *more* concurrency.

Benefits of PRET
(If you take advantage of determinism)

- **Modularity**
  - Non-interference between tasks (even with interrupts).
- **Exactness**
  - Get not just WCET, but *actual response* time.
- **Repeatability**
  - Works in the field like on the bench.
- **Complexity**
  - More hard-real-time tasks is better than fewer.
- **Certifiability**
  - Every *correct* execution of the software gives the same behavior.
- **Energy**
  - Reduce voltage and frequency to the minimum to meet deadlines.

Achieving Real Time in Practice

- ✔ overengineering
- ✔ using old technology
- ✔ response-time analysis
- real-time operating systems (RTOSs)
- specialized networks
- ✔ extensive testing and validation

What about the programming model?

Engineering Models for Real-Time Cyber-Physical Systems

- **PRET**: time-deterministic architectures
  - http://chess.eecs.berkeley.edu/pret
- **PTIDES**: distributed real-time software
  - http://chess.eecs.berkeley.edu/ptides

These enable models with tightly controlled timing and deterministic behaviors.

We have shown that that these models are practically realizable at reasonable cost.

Roots of the Idea

Using Time Instead of Timeout for Fault-Tolerant Distributed Systems

LESLIE LAMPORT
SRI International

A general method is described for implementing a distributed system with any desired degree of fault-tolerance. Instead of relying upon explicit timeouts, processes execute a simple clock-driven algorithm. Reliable clock synchronization and a solution to the Byzantine Generals Problem are assumed.

Categories and Subject Descriptors: C.2.4 [**Computer-Communications Networks**]: Distributed Systems—*network operating systems*; D.1.3 [**Programming Techniques**]: Concurrent Programming; D.4.1 [**Operating Systems**]: Process Management—*synchronization*; D.4.3 [**Operating Systems**]: File Systems Management—*distributed file systems*; D.4.5 [**Operating Systems**]: Reliability—*fault-tolerance*; D.4.7 [**Operating Systems**]: Organization and Design—*distributed systems*; *real-time systems*

General Terms: Design, Reliability

Additional Key Words and Phrases: Clocks, transaction commit, timestamps, interactive consistency, Byzantine Generals Problem

ACM Transactions on Programming Languages and Systems, 1984.

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

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

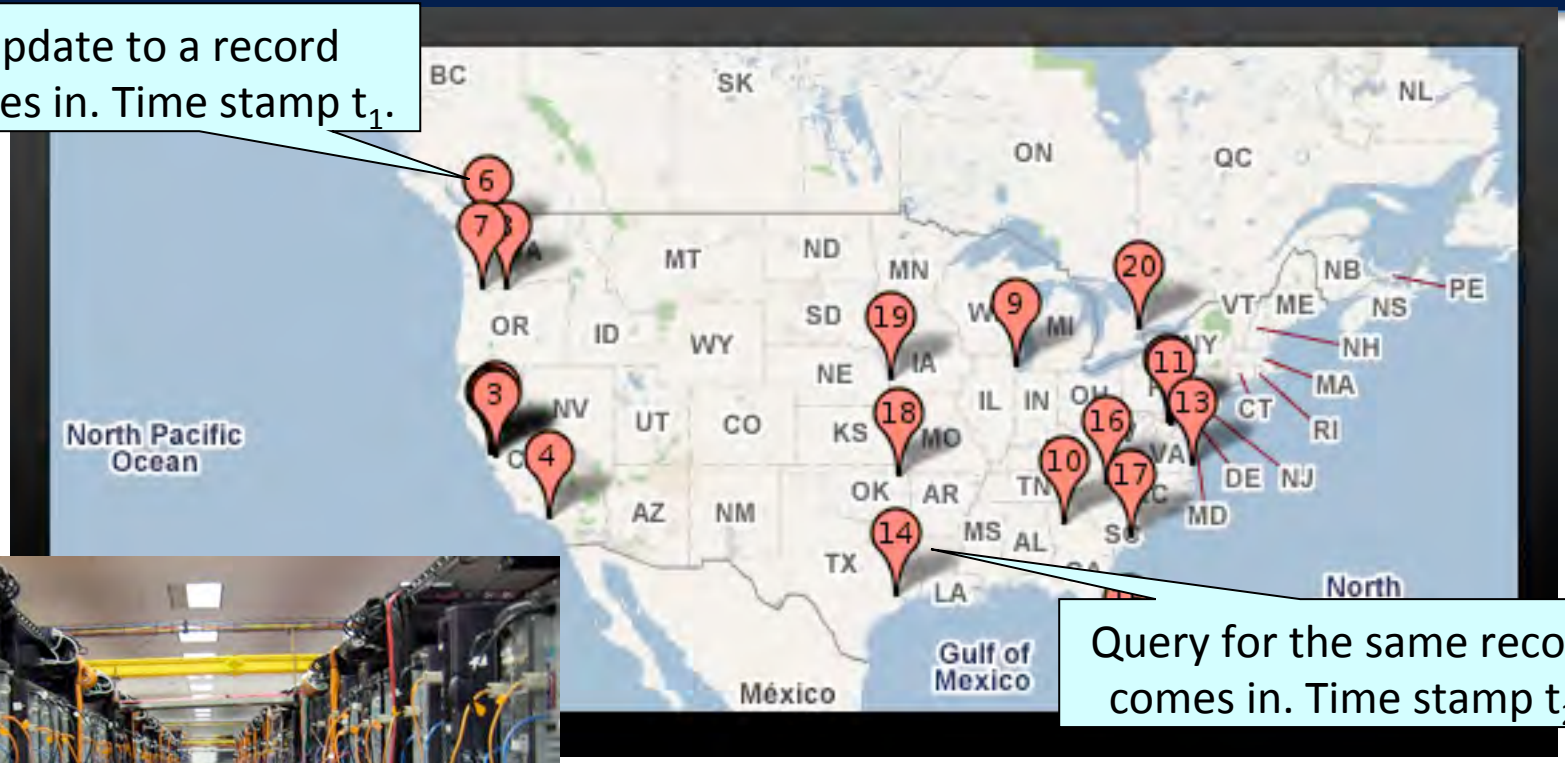
Spanner is Google's scalable, multi-version, globally-distributed, and synchronously-replicated database. It is the first system to distribute data at global scale and support externally-consistent distributed transactions. This paper describes how Spanner is structured, its feature set, the rationale 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

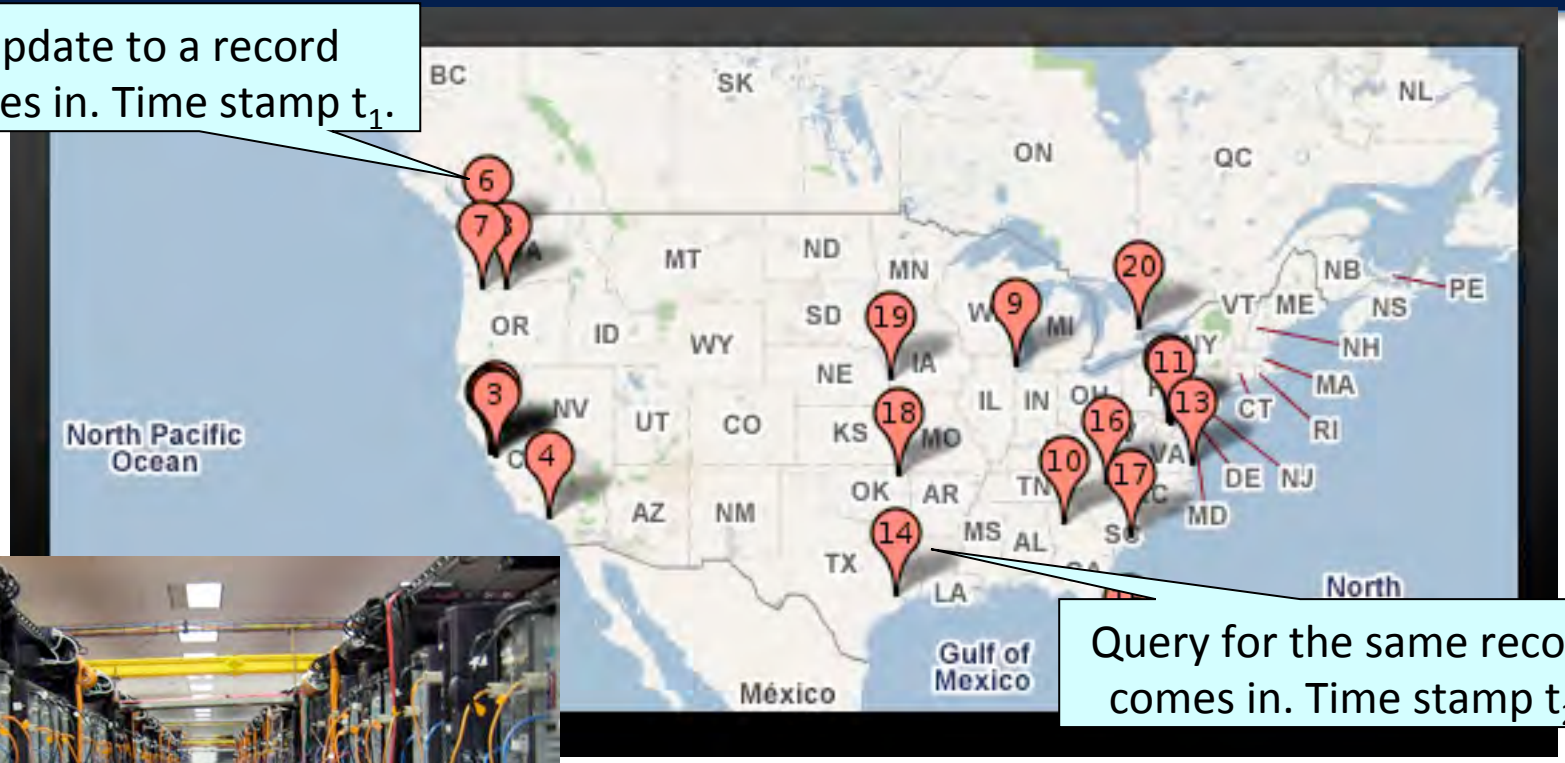


Distributed database with redundant storage and query handling across data centers.

Lee, Berkeley

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.

Lee, Berkeley

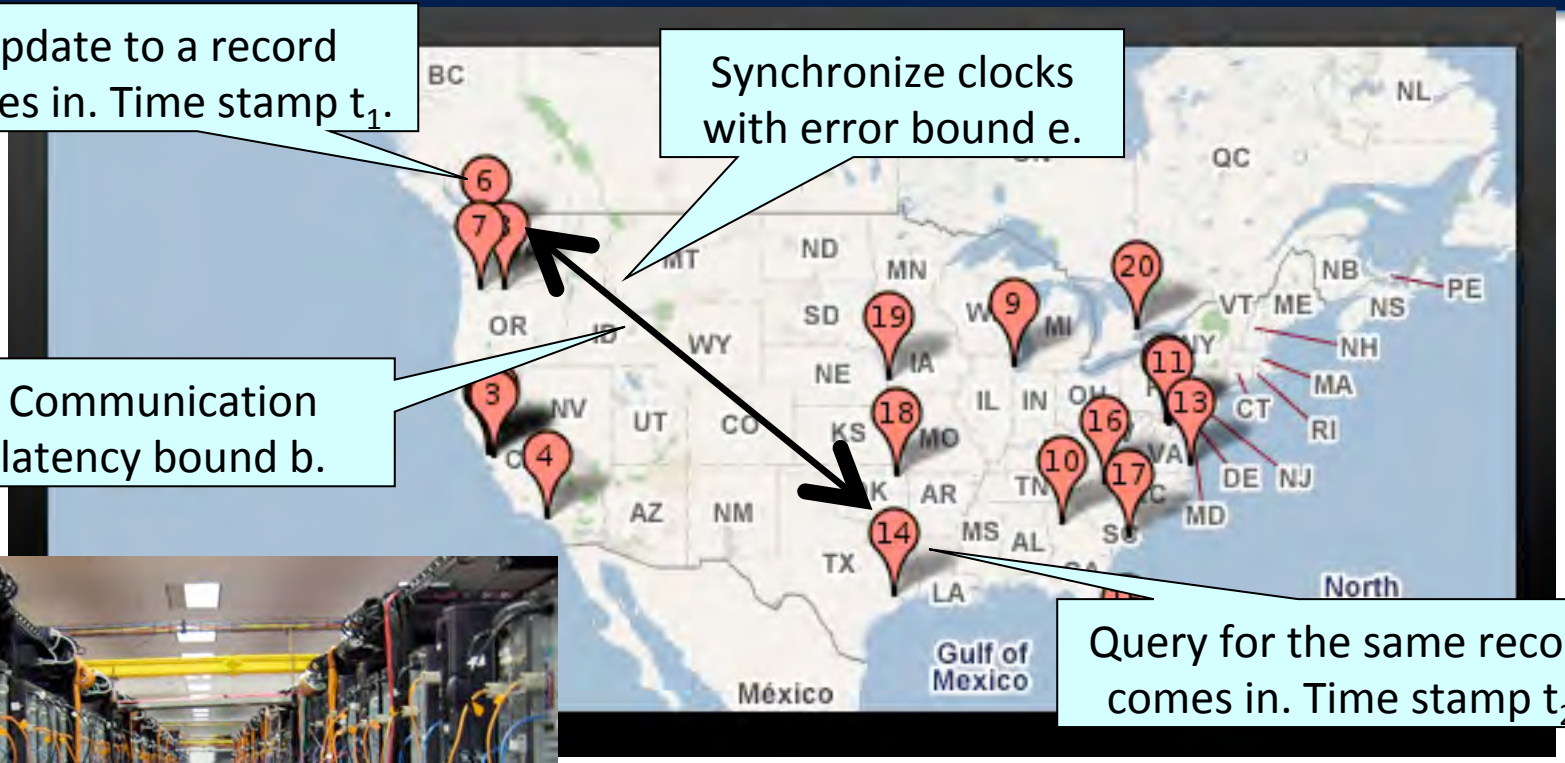
Google Spanner: When to Respond?

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 + b, issue the current record value as a response.

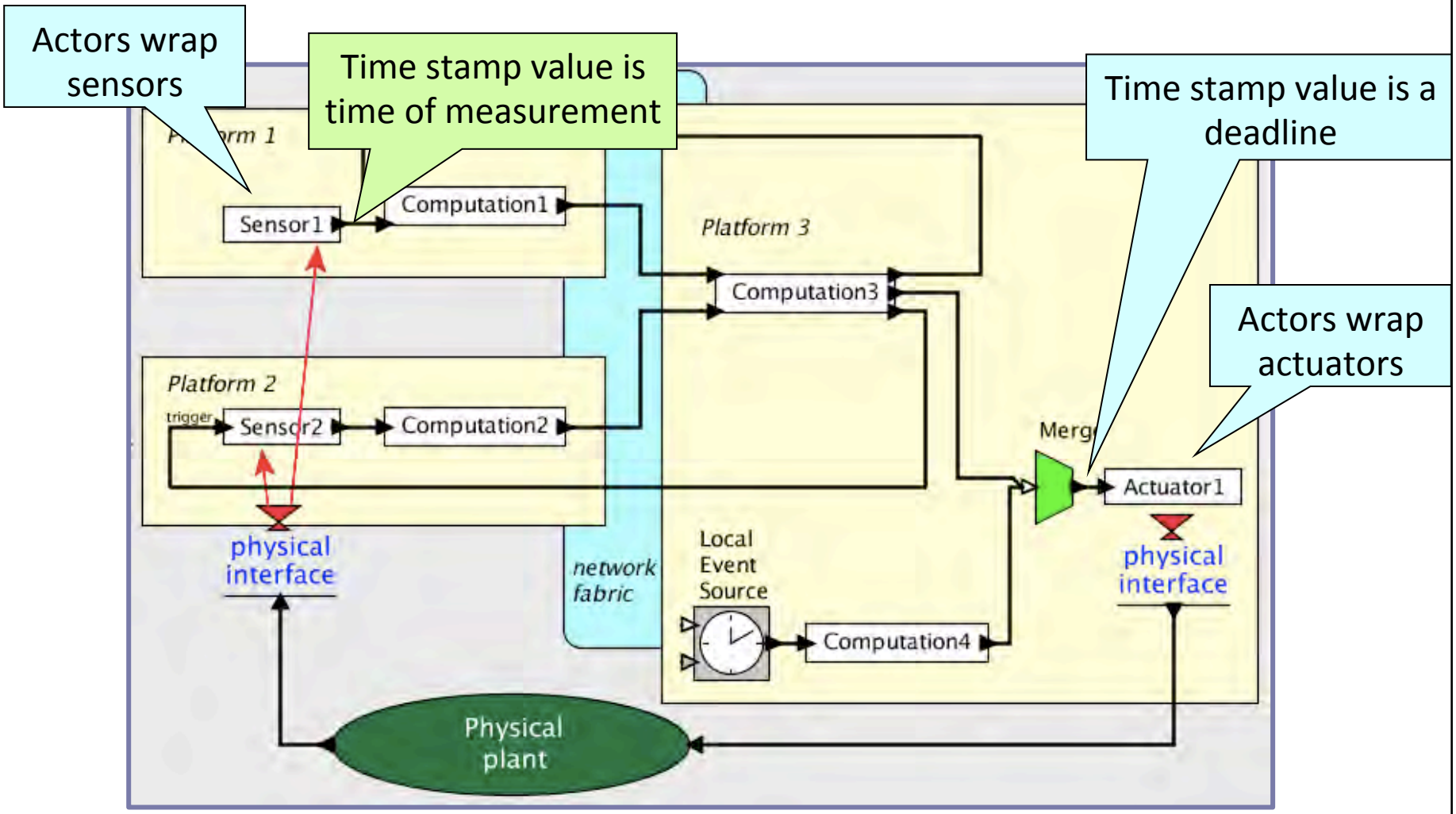
Lee, Berkeley

Google Spanner: Fault!

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.

Lee, Berkeley

Ptides Applies this Idea to Distributed Real-Time Systems



Ptides: Deterministic Distributed Real-Time

Assume bounds on:

- *clock synchronization error*
- *network latency*

then ***events are processed in time-stamp order*** at every component. If in addition we assume

- *bounds on execution time*

then events are delivered to actuators on time.

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

PTIDES Requires Synchronized Clocks with Bounded Error

Every engineered design makes assumptions about its execution platform.

Ubiquitous clock synchronization gives us a new and powerful tool.

Lingua Franca

A meta-language for PRET, Ptides, and predictable concurrent systems in general.

Invited: Actors Revisited for Time-Critical Systems

Marten Lohstroh
UC Berkeley, USA

Martin Schoeberl
TU Denmark, Denmark

Andrés Goens
TU Dresden, Germany

Armin Wasicek
Avast, USA

Christopher Gill
Washington Univ., St. Louis, USA

Marjan Sirjani
Mälardalen Univ., Sweden

Edward A. Lee
UC Berkeley, USA

ABSTRACT

Programming time-critical systems is notoriously difficult. In this paper we propose an actor-oriented programming model with a semantic notion of time and a deterministic coordination semantics based on discrete events to exercise precise control over both the computational and timing aspects of the system behavior.

2 ACTORS

The actor model was introduced by Hewitt [6] in the early 70s. Since then, the use of actors has proliferated in programming languages [1, 2], coordination languages [14, 15], distributed systems [7, 11], and simulation and verification engines [13, 17]. Actors have much in common with objects—a paradigm focused on reducing code replication and increasing modularity via data

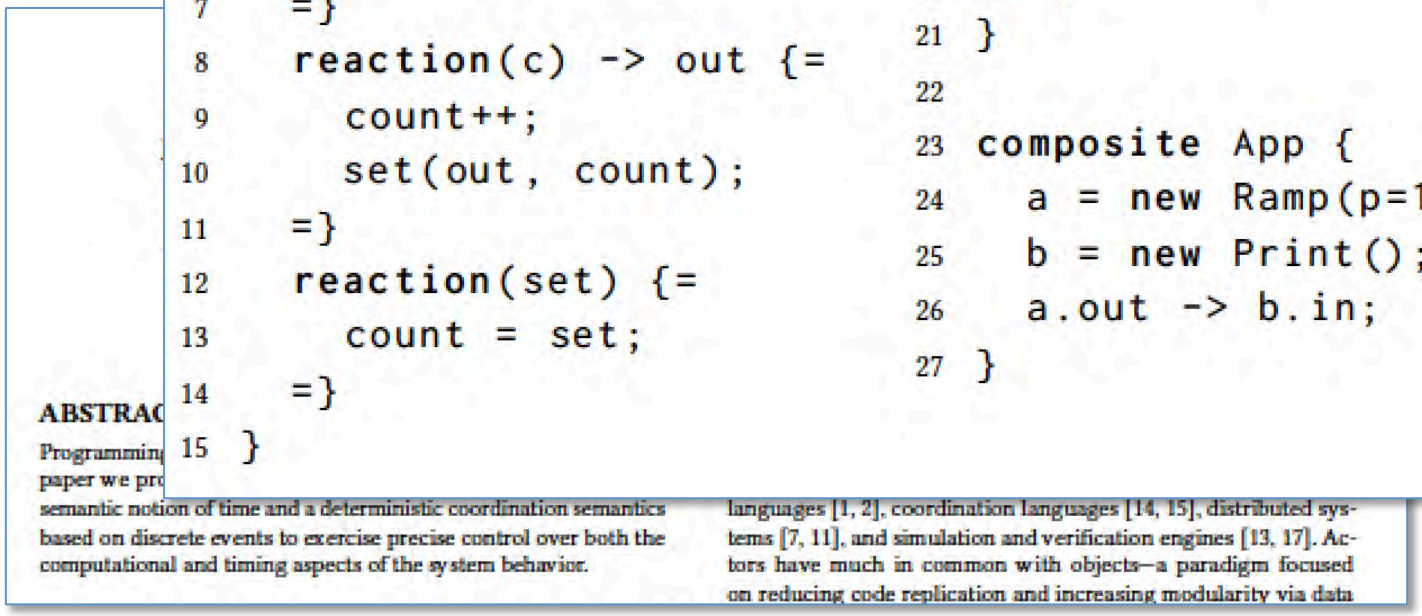
To Appear, Design Automation Conference (DAC), June, 2019.

Lingua Franca

A meta-l
predictal

```
1  reactor Ramp(p:int(10)) {
2    input set:int;
3    output out:int;
4    clock c(p);
5    constructor {=
6      int count = 0;
7    =}
8    reaction(c) -> out {=
9      count++;
10     set(out, count);
11   =}
12   reaction(set) {=
13     count = set;
14   =}
15 }
```

```
16 reactor Print {
17   input in:int;
18   reaction(in) {=
19     printf("%d\n", in);
20   =}
21 }
22
23 composite App {
24   a = new Ramp(p=100);
25   b = new Print();
26   a.out -> b.in;
27 }
```



ABSTRAC

Programming
paper we pr
semantic notion of time and a deterministic coordination semantics based on discrete events to exercise precise control over both the computational and timing aspects of the system behavior.

languages [1, 2], coordination languages [14, 15], distributed systems [7, 11], and simulation and verification engines [13, 17]. Actors have much in common with objects—a paradigm focused on reducing code replication and increasing modularity via data

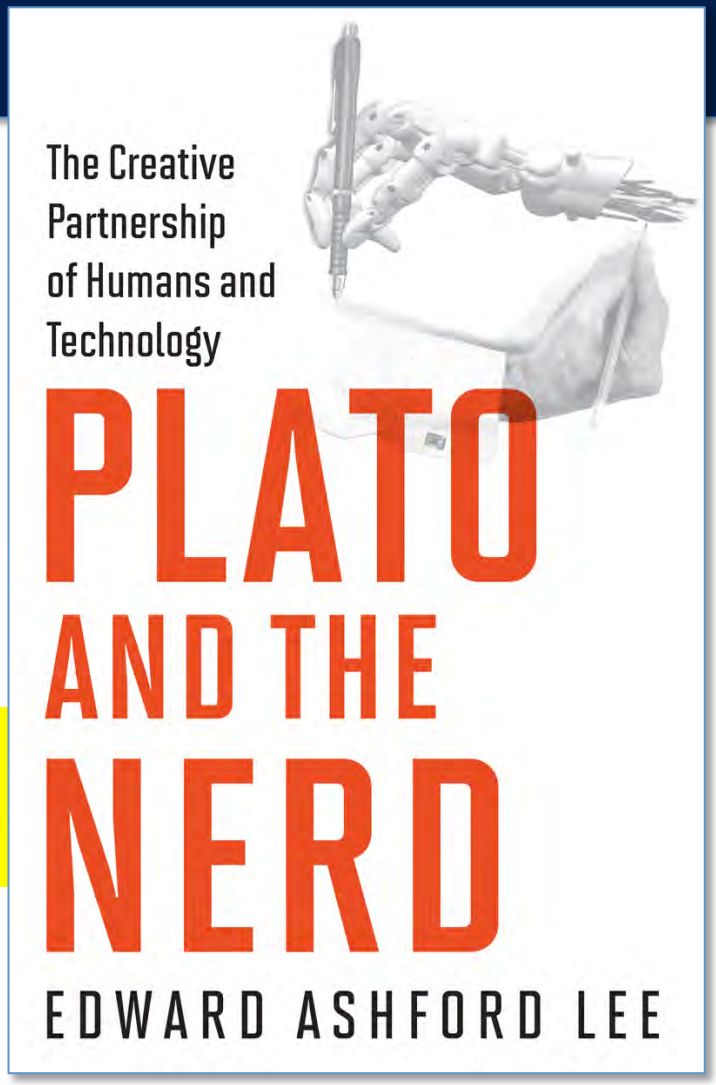
To Appear, Design Automation Conference (DAC), June, 2019.

Conclusion

- 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.

My message:
Do less science and more engineering.

http://ptolemy.berkeley.edu/pret
http://ptolemy.berkeley.edu/ptides



http://platoandthenerd.org