Resurrecting Laplace's Demon: The Case for Deterministic Models

Edward A. Lee

Robert S. Pepper Distinguished Professor
UC Berkeley

Keynote Talk: Workshop on Modeling in Software Engineering (MiSE)

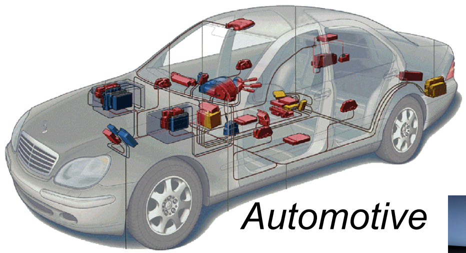
Part of the International Conference on Software Engineering (ICSE)

May 16-17, 2016
Austin, TX

Focus on Cyber-Physical Systems Full of Contradictory Requirements

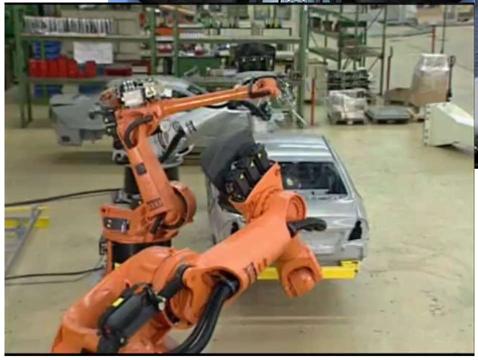
It's not just information technology anymore:

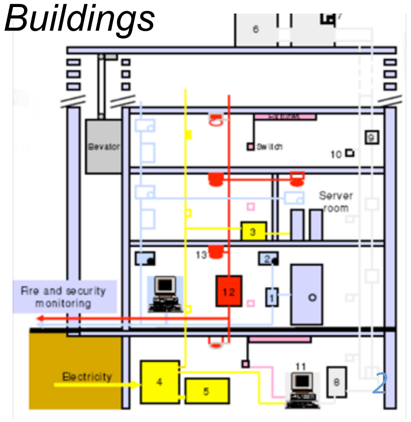
- Cyber + *Physical*
- Computation + *Dynamics*
- Security + *Safety*



Contradictions:

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





Innovation:

Cyber-physical systems require new engineering methods and models to address these contradictions.

Lee, Berkeley

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

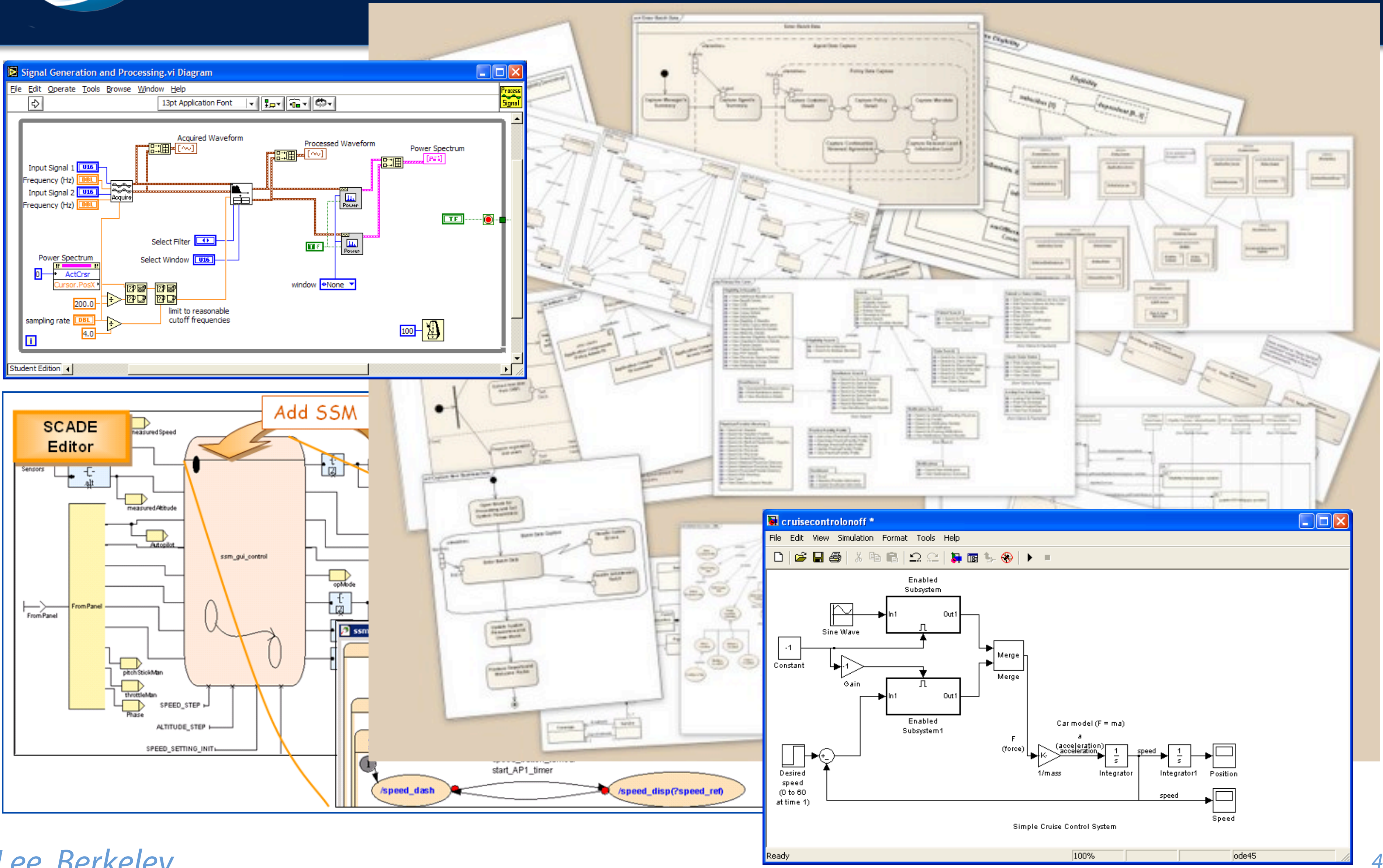
Industrial automation example from 2008: Bosch-Rexroth printing press.

The term “IoT” includes the technical solution “Internet technology” in the problem statement “connect things”.

The term CPS does not.

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.

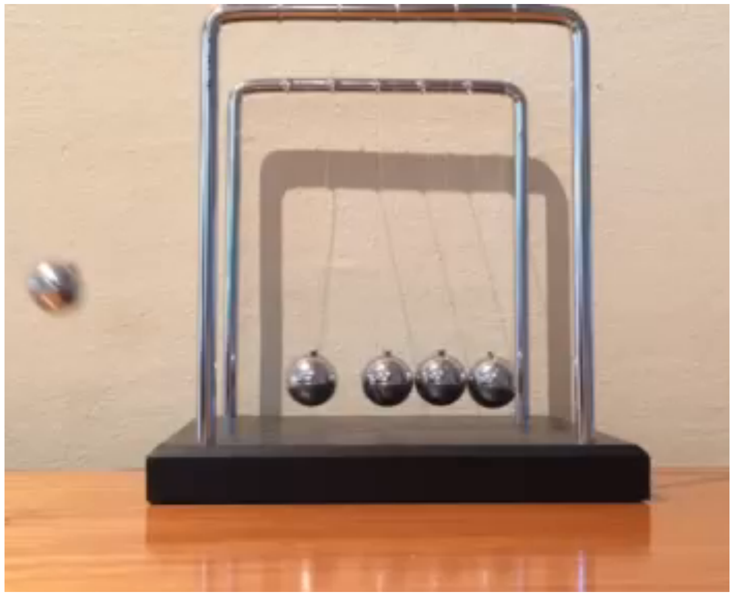
Focus on Models



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

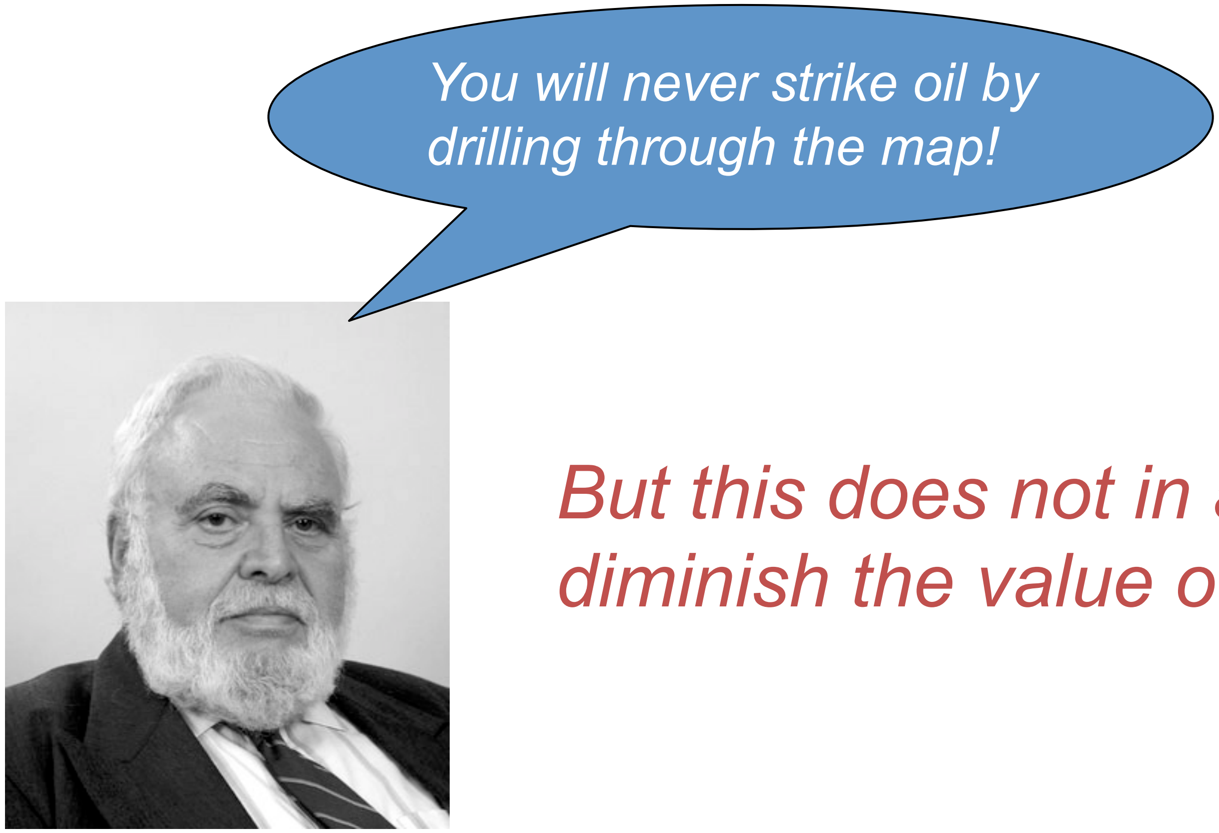


The target (the thing being modeled).

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

Fidelity is how well the model and its target match

Engineers often confuse the model with its target



Solomon Wolf Golomb

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

Determinacy

Some of the most valuable models are *deterministic*.

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

Deterministic models have proven extremely valuable in the past.

Laplace's Demon

"We may regard the present state of the universe as the effect of its past and the cause of its future. An intellect which at a certain moment would know all forces that set nature in motion, and all positions of all items of which nature is composed, if this intellect were also vast enough to submit these data to analysis, it would embrace in a single formula the movements of the greatest bodies of the universe and those of the tiniest atom; for such an intellect nothing would be uncertain and the future just like the past would be present before its eyes."

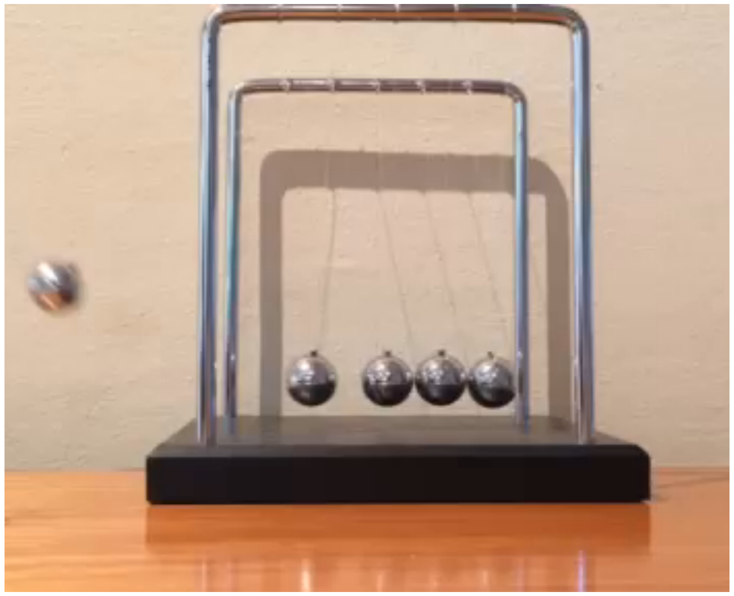
— Pierre Simon Laplace, A Philosophical Essay on Probabilities

Pierre-Simon Laplace (1749–1827). Posthumous portrait by Joan-Baptiste Paulin Guérin, 1838

But hasn't Laplace's demon been debunked?

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

Deterministic model



Nondeterministic target

Laplace reflected a period of optimism in science that tended to equate the model with the truth.

He was drilling through a map!

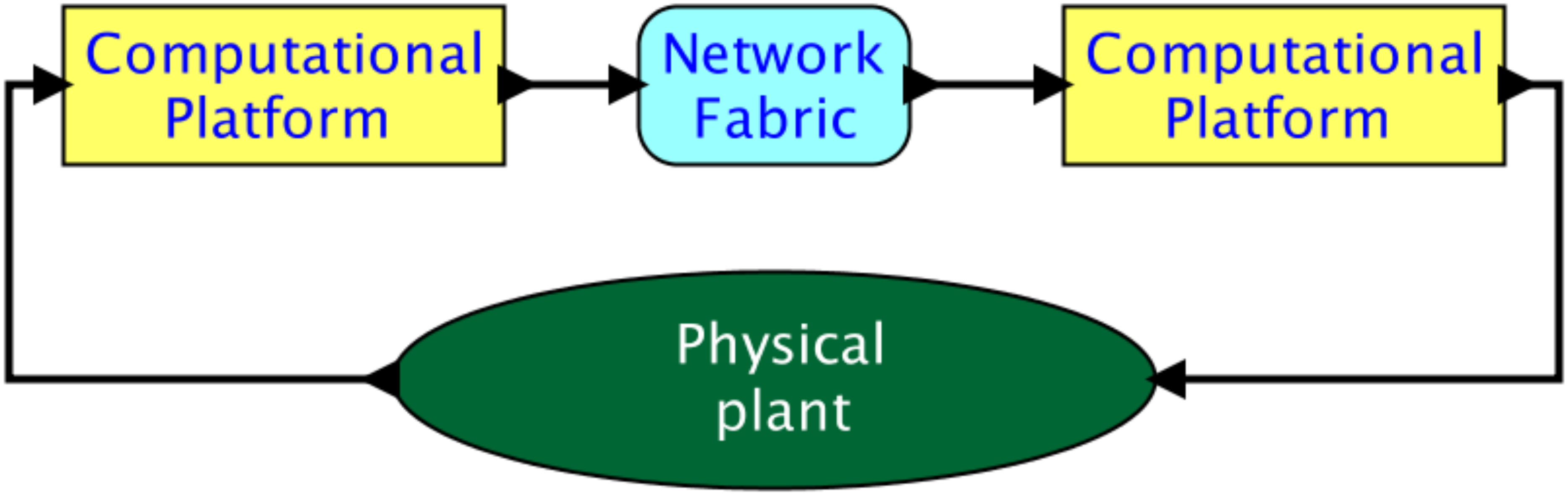
Lee, Berkeley

A Model Need not be *True* to be *Useful*

"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.

Schematic of a simple CPS



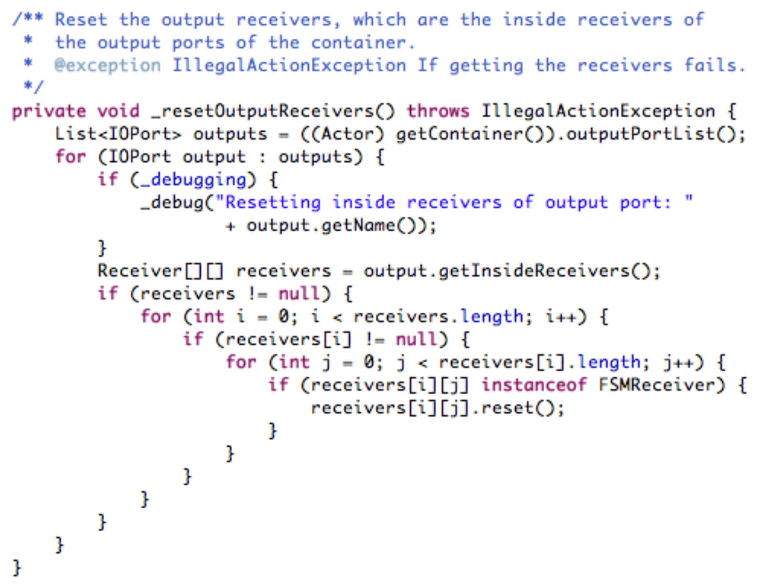
What kinds of models should we use?

Let's look at the most successful kinds of models from the cyber and the physical worlds.

Software is a Model

Physical System

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

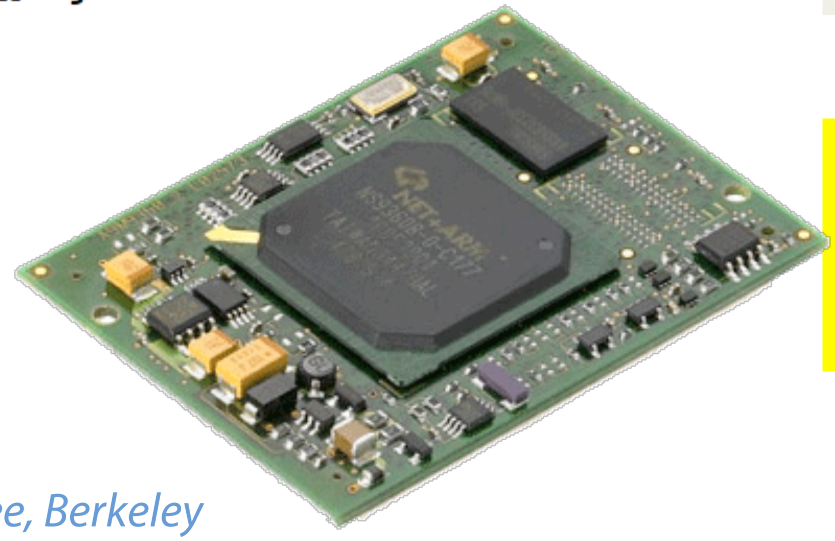
Single-threaded imperative programs are deterministic models

Consider single-threaded imperative programs

```
void foo(int32_t x) {
      if (x > 1000) {
            x = 1000;
      }
      if (x > 0) {
            x = x + 1000;
            if (x < 0) {
                  panic();
            }
      }
}
```

This program defines exactly one behavior, given the input x.

Note that the modeling framework (the C language, in this case) defines "behavior" and "input."



The target of the model is nondeterministic (electrons and holes sloshing around in silicon).

Software relies on another deterministic model that abstracts the hardware

Physical System

Model

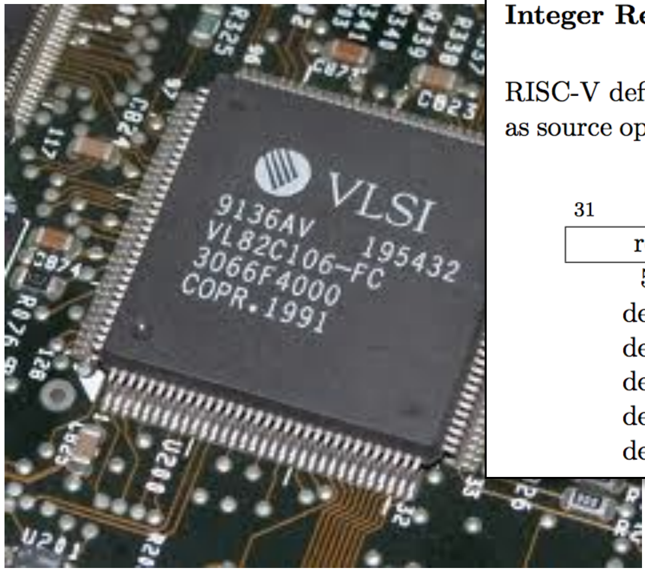
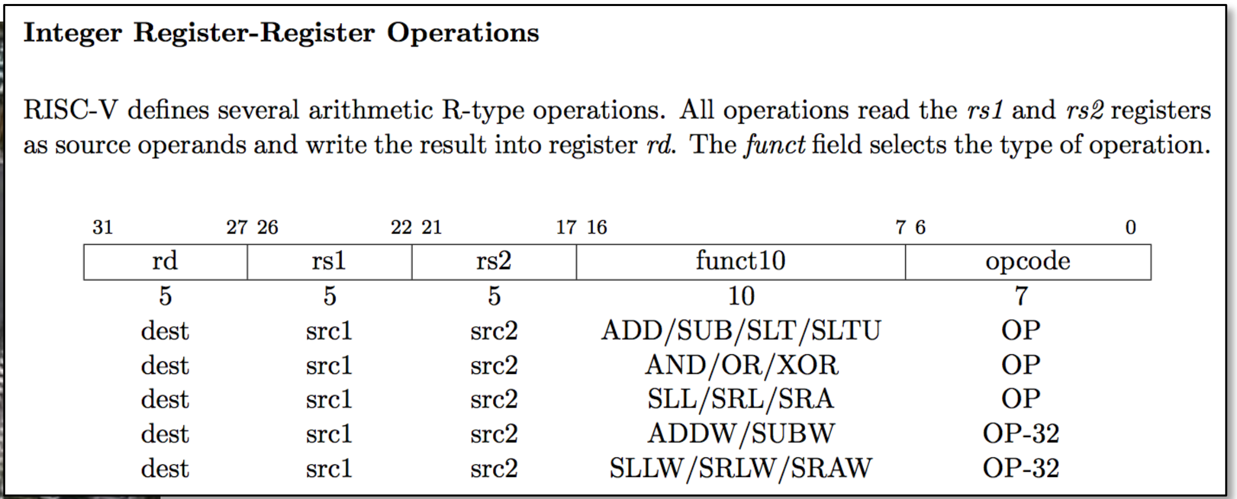


Image: Wikimedia Commons



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

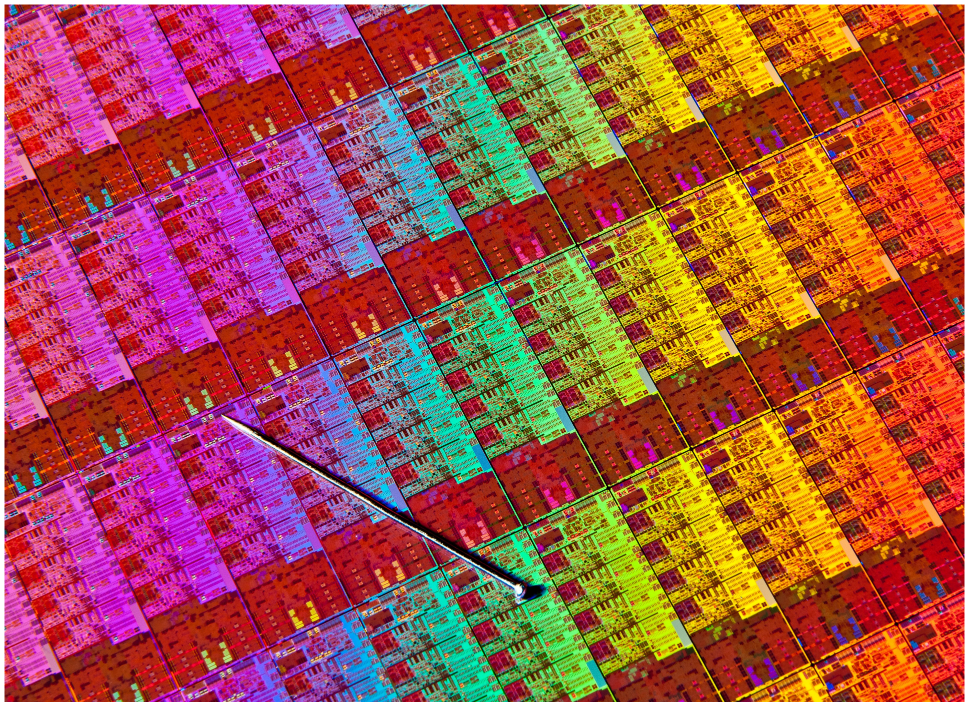
Waterman, et al., The RISC-V Instruction Set Manual, UCB/EECS-2011-62, 2011

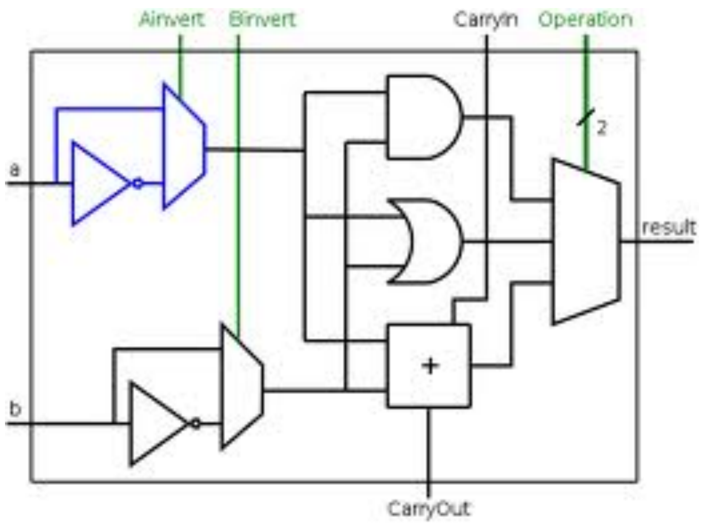
Instruction Set Architectures (ISAs) are deterministic models

… which relies on yet another deterministic model

Physical System

Model





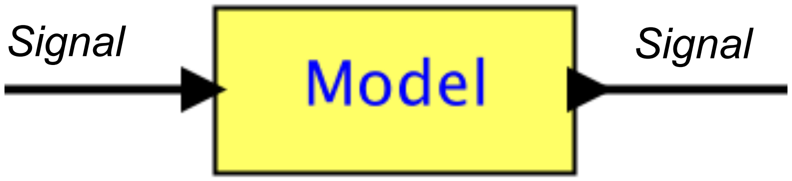
Synchronous digital logic is a deterministic model

Deterministic Models for the Physical Side of CPS

Physical System

Model

Image: Wikimedia Commons

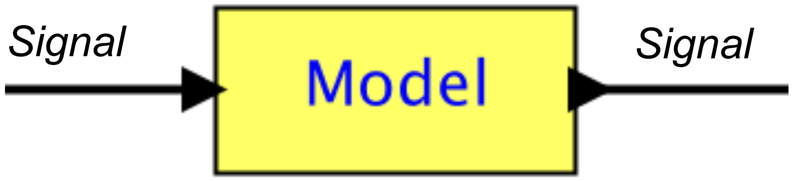


$$\dot{\mathbf{x}}(t) = \dot{\mathbf{x}}(0) + \frac{1}{M}\int_{0}^{t} \mathbf{F}(\tau)d\tau$$

Differential Equations are deterministic models

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

```
void initTimer(void) {
    SysTickPeriodSet(SysCtlClockGet() / 1000);
    SysTickEnable();
    SysTickIntEnable();
}
volatile uint timer_count = 0;
void ISR(void) {
    if(timer_count != 0) {
        timer_count--;
    }
}
int main(void) {
    SysTickIntRegister(&ISR);
    .. // other init
    timer_count = 2000;
    initTimer();
    while(timer_count != 0) {
        ... code to run for 2 seconds
    }
    ... // other code
}
```



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

Lee, Berkeley   Image: Wikimedia Commons

Timing is not Part of Software and Network 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.

Embedded software designers have no map!

Determinism? Really?

CPS applications operate in an intrinsically nondeterministic world.

Does it really make sense to insist on deterministic models?

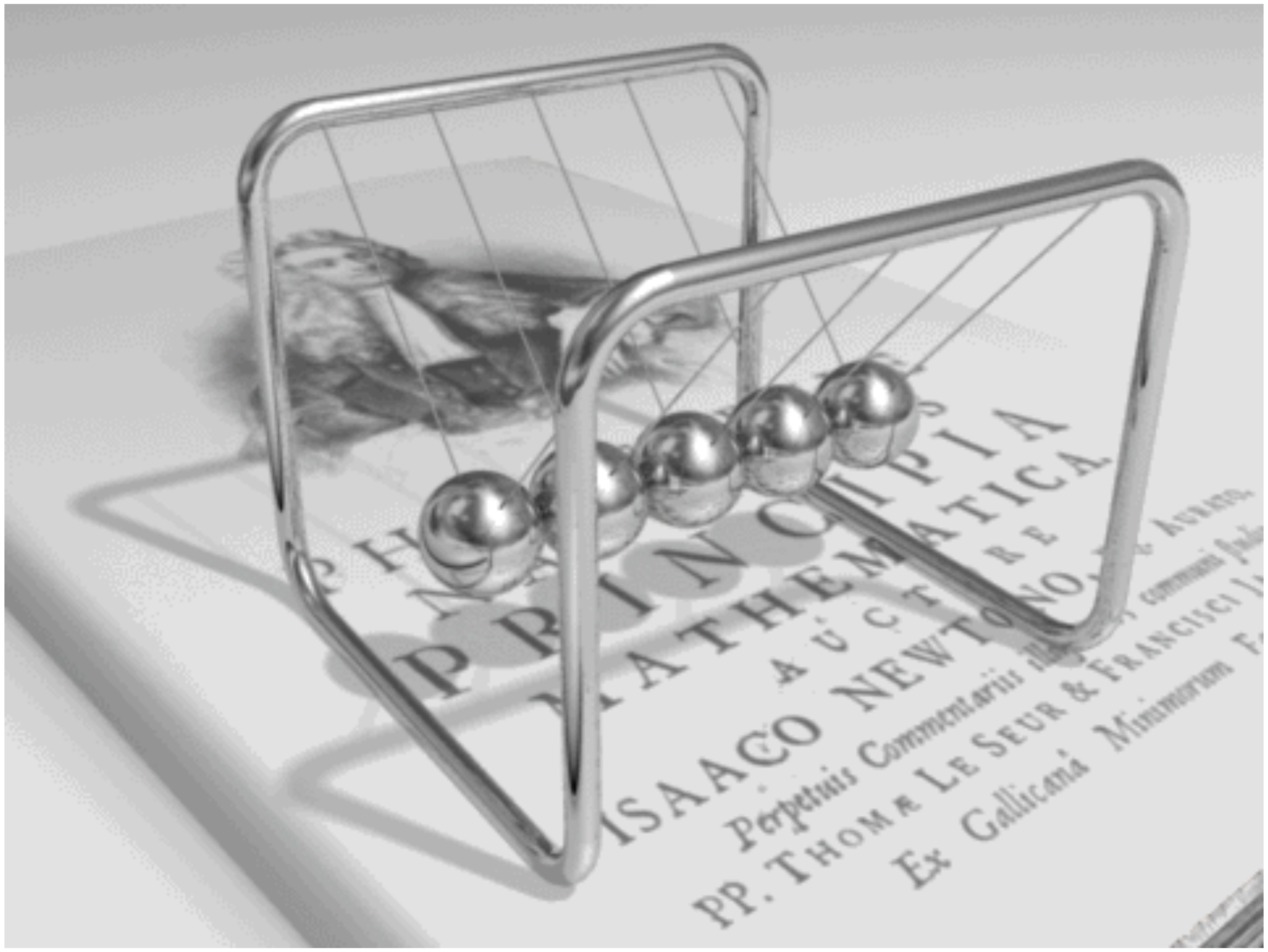
The Value of Models

- 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.

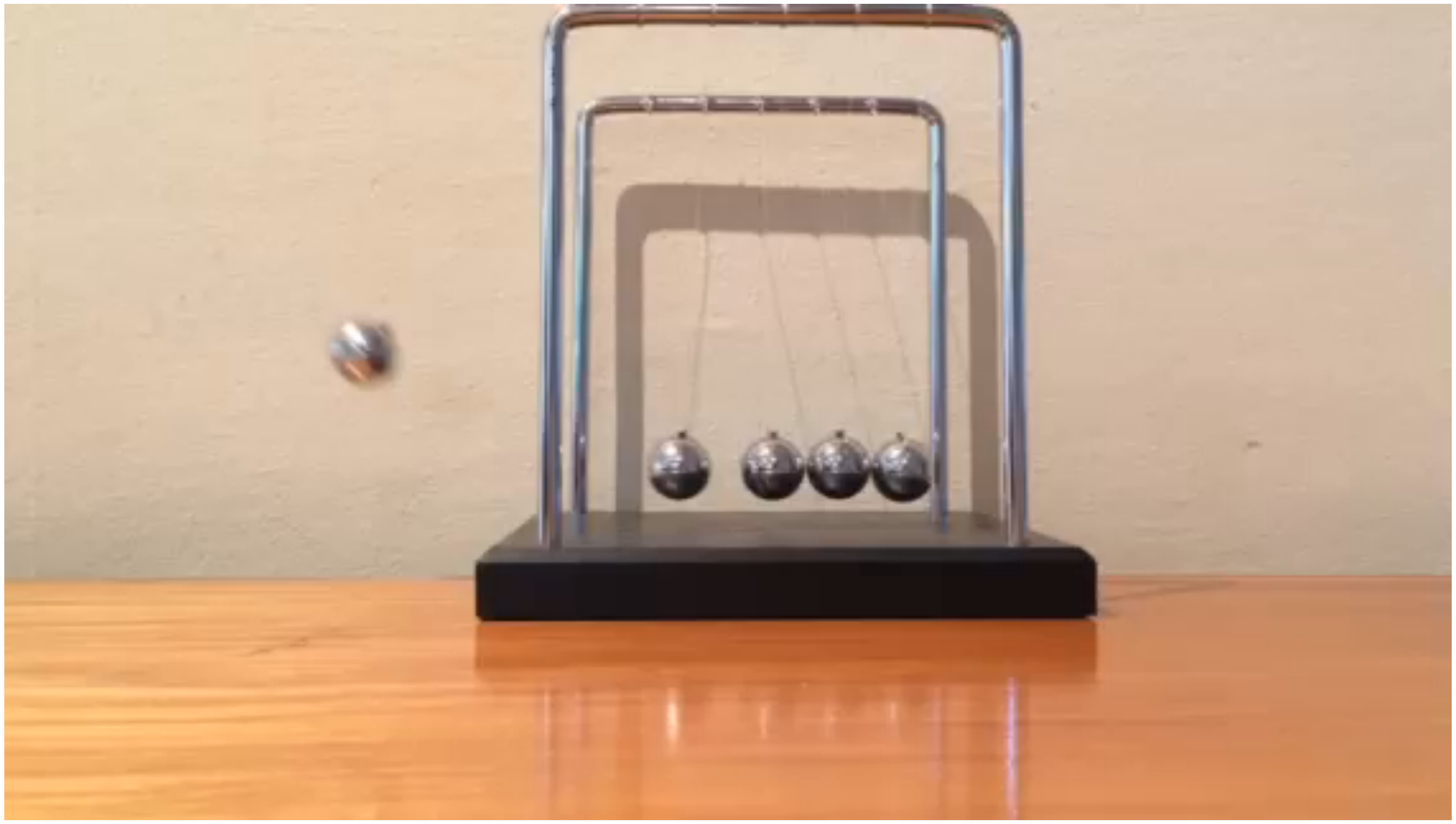
In engineering, model fidelity is a two-way street!

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


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

A Physical Realization



Model Fidelity

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

I'm an engineer…

For CPS, we need to Change 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?
What about Resilience? Adaptability?

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!

Interim Conclusion

We have to fix the models!

But how?

Example of the Benefits of Embracing Temporal Semantics

Despite using TCP/IP on Ethernet, this network achieves highly reliable bounded latency.

TSN (time-sensitive networks) technology is starting to become pervasive…

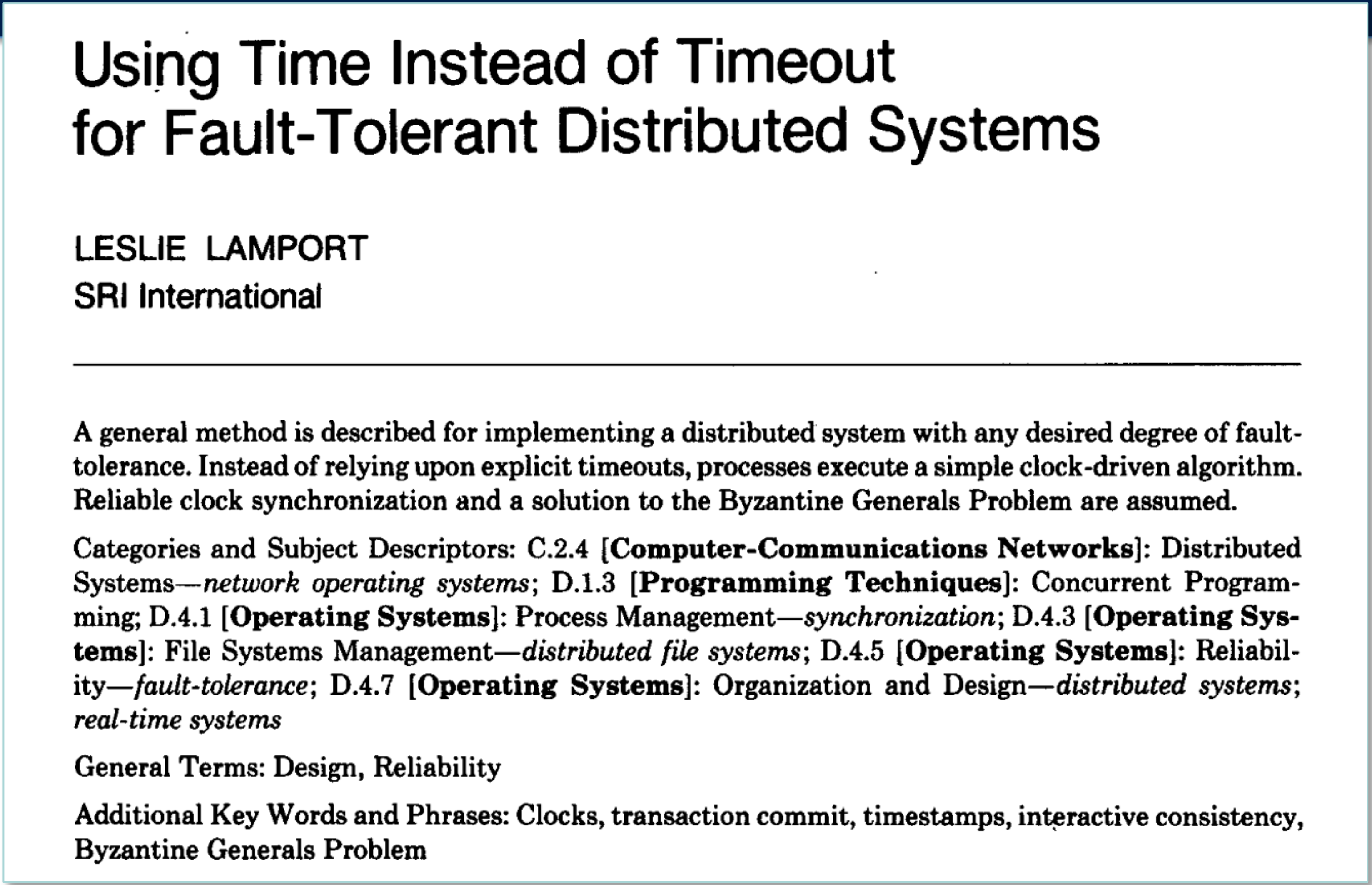
*This Bosch Rexroth printing press is a cyber-physical factory using Ethernet and TCP/IP with **high-precision clock synchronization (IEEE 1588)** on an **isolated LAN**.*

Existence Proofs

Deterministic models with highly faithful implementations exist for distributed real-time systems.

- PTIDES: distributed real-time software
  - Deterministic timing across networks
- PRET machines
  - Deterministic timing at the processor level

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.

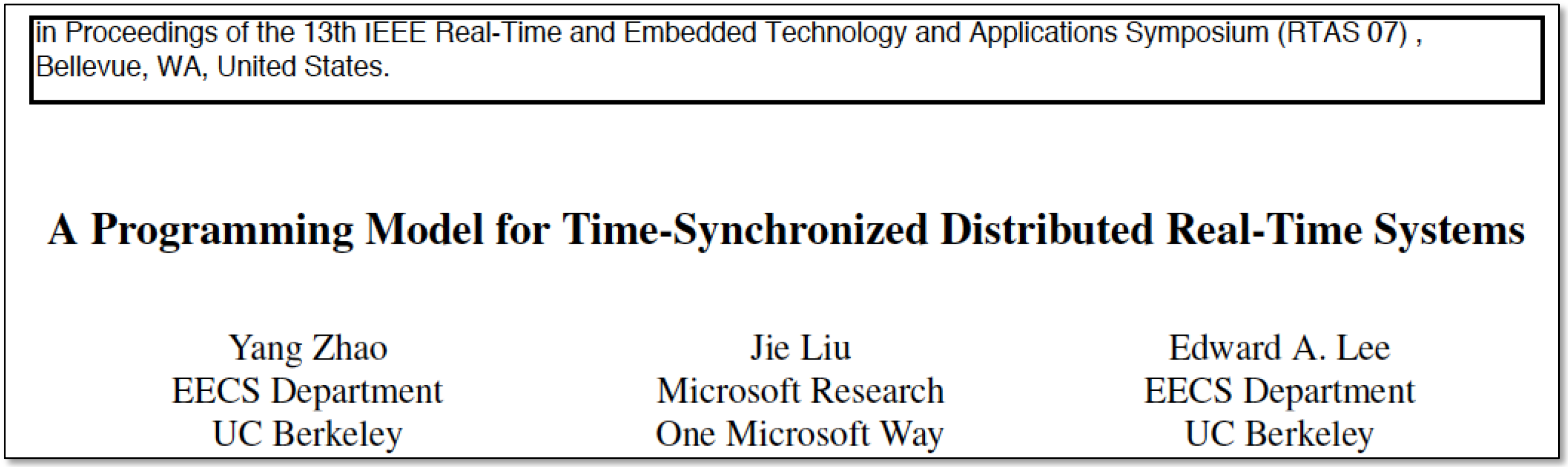
Categories and Subject Descriptors: C.2.4 [**Computer-Communications Networks**]: Distributed Systems—*network operating systems*; D.1.3 [**Programming Techniques**]: Concurrent Programming; D.4.1 [**Operating Systems**]: Process Management—*synchronization*; D.4.3 [**Operating Systems**]: File Systems 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

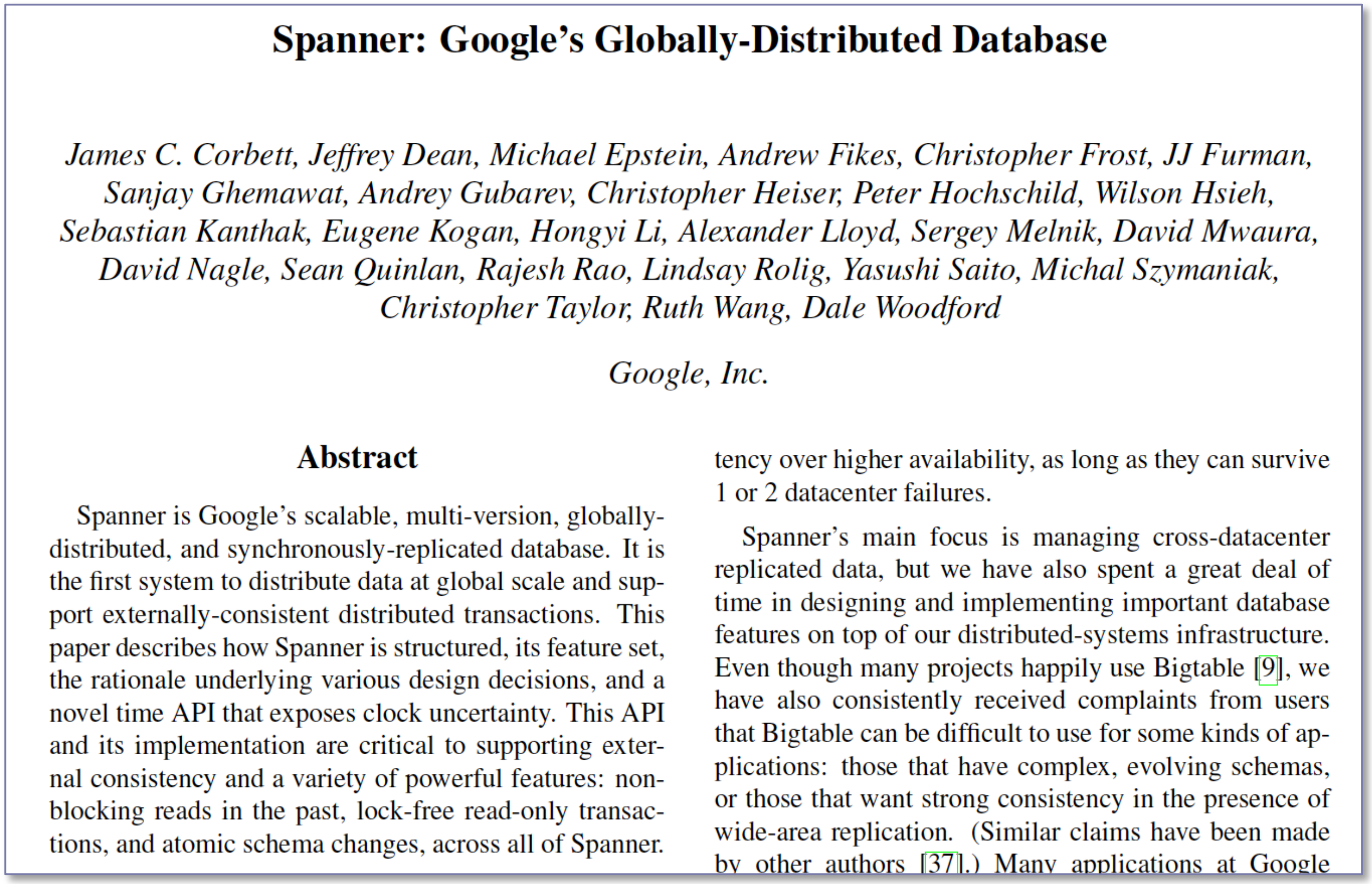
Jie Liu
Microsoft Research
One Microsoft Way

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

PTIDES: Discrete-Event Semantics + Synchronized Clocks + Sensors and Actuators

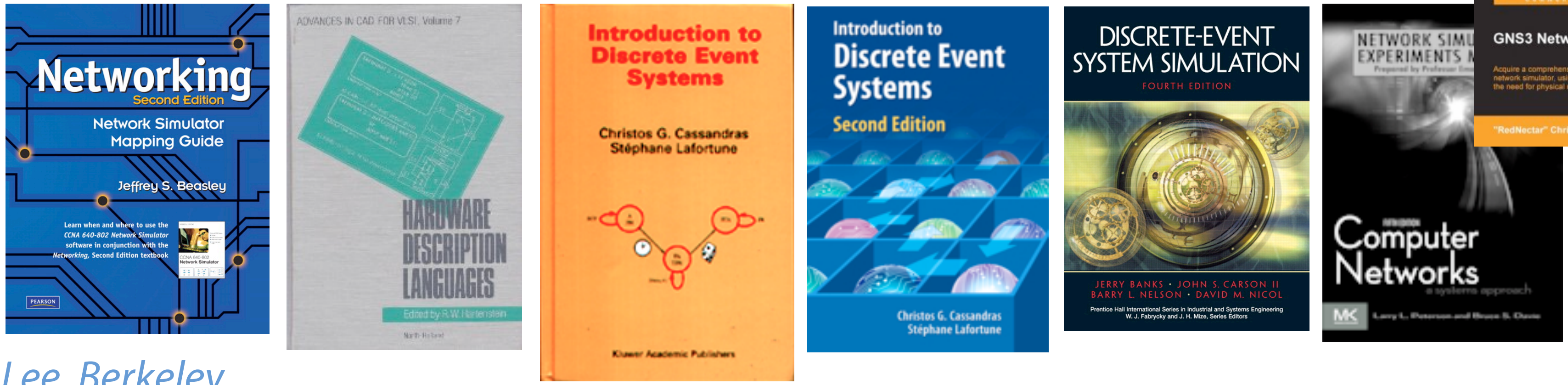
Time-stamped events that are processed in time-stamp order.

This MoC is widely used in simulation and HDLs.

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

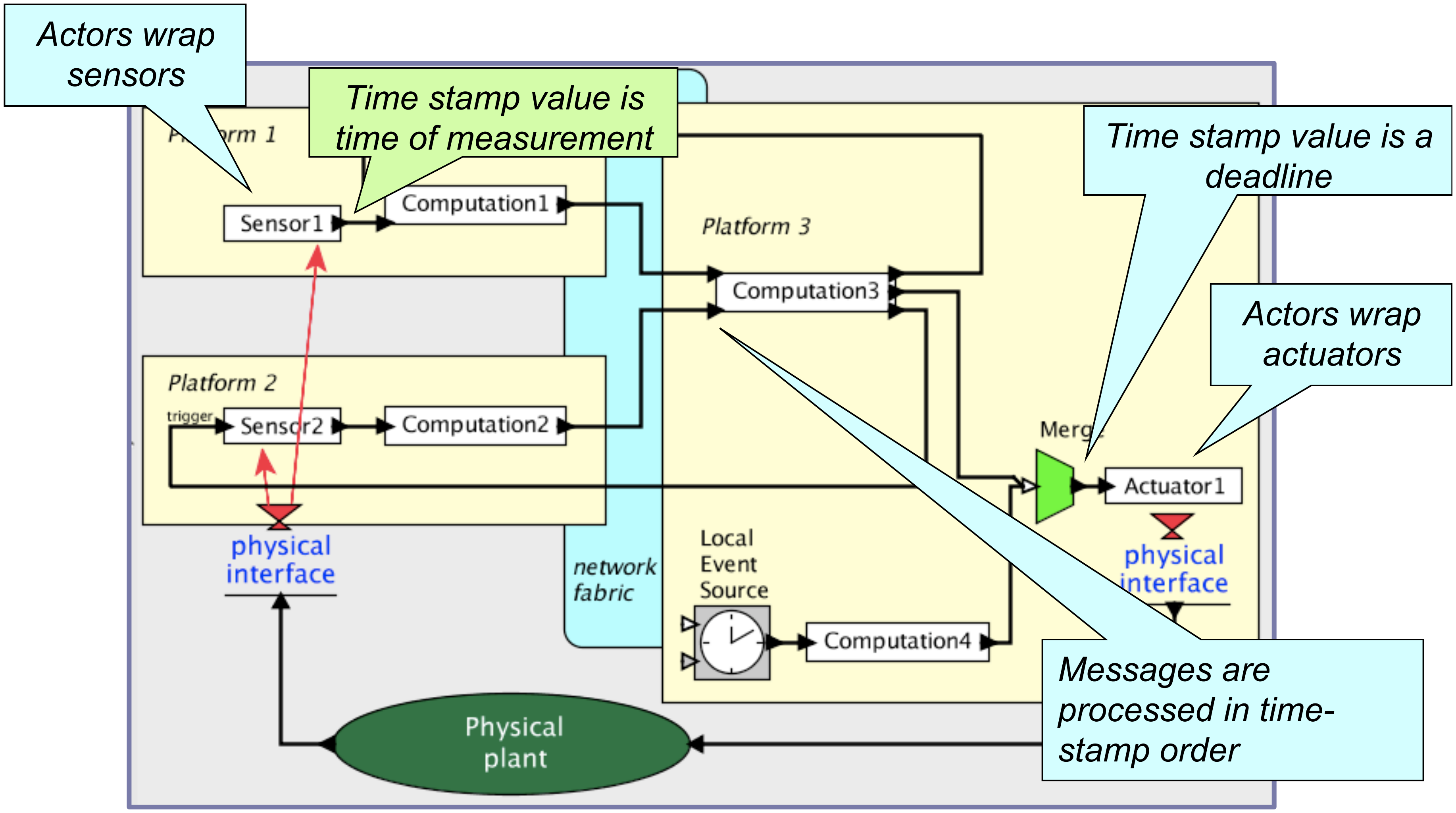
A few texts that use the DE MoC

Synchronization and Linearity
An Algebra for Discrete Event Systems
Wiley
2001 Standard
Thomas & Moorby's
The Verilog Hardware Description Language
Fifth Edition
GNS3 Network Simulation Guide



Networking
Second Edition
Network Simulator Mapping Guide
Jeffrey S. Beasley
Hardware Description Languages
Introduction to Discrete Event Systems
Christos G. Cassandras
Stéphane Lafortune
Kluwer Academic Publishers
Introduction to Discrete Event Systems
Second Edition
Discrete-Event System Simulation
Fourth Edition
Jerry Banks · John S. Carson II
Barry L. Nelson · David M. Nicol
Computer Networks

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



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

So Many Assumptions?

You will never strike oil by drilling through the map!

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

Non-Synchronized Clocks

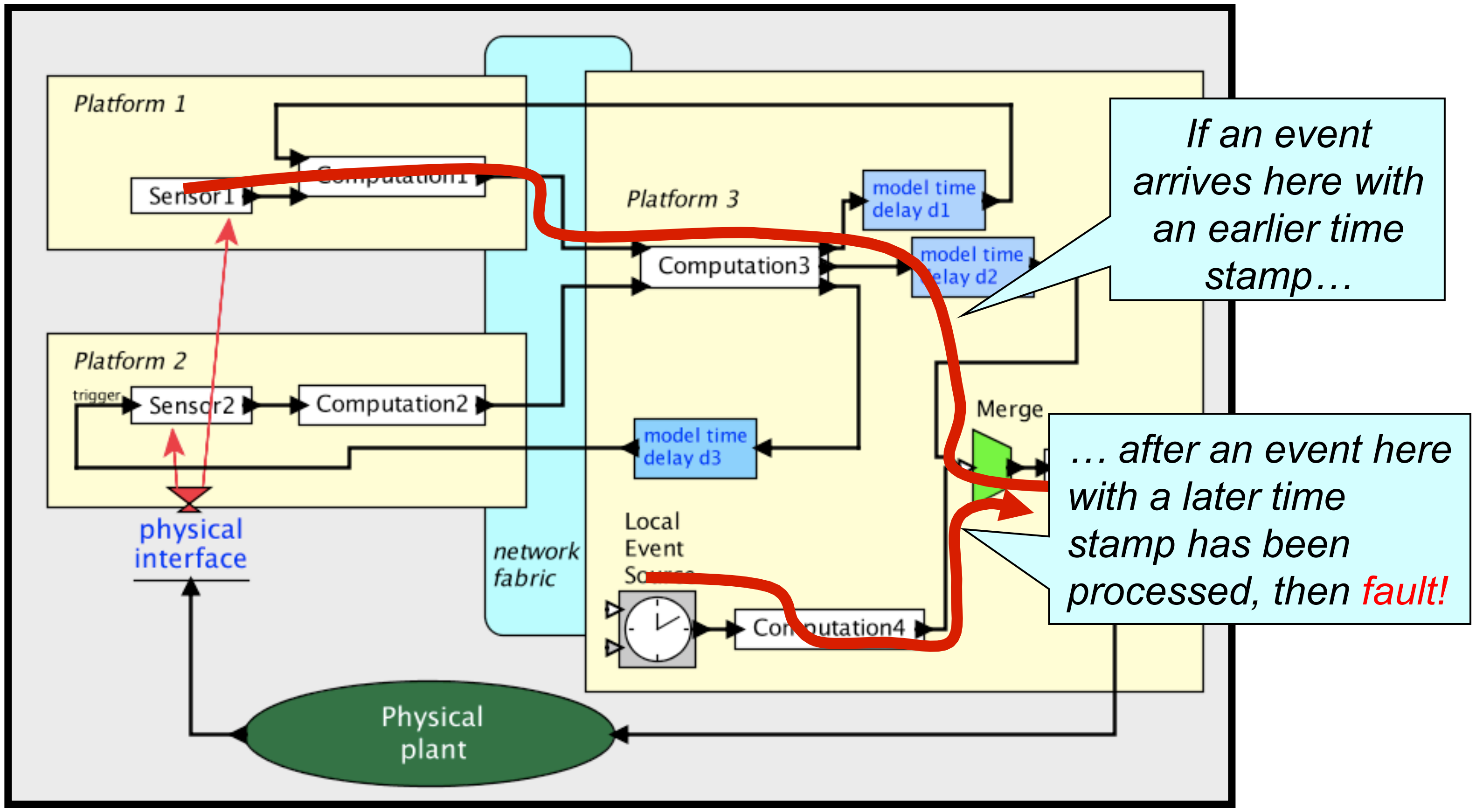
*Violations of the requirements are detectable as out-of-order events and can be treated as **faults**.*

Lee, Berkeley

Handling Faults

A fault manifests as out-of-order events.

Occurrence of a fault implies one or more of the assumptions was violated.



Ptides

High-precision clock synchronization will become ubiquitous. Networks will become able to “guarantee” bounded latencies.

The time is right.

But what about the execution time of software?

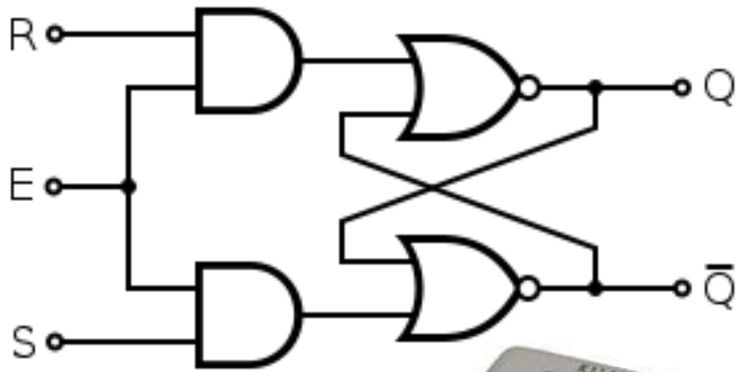
Existence Proofs

Deterministic models with highly faithful implementations exist for distributed real-time systems.

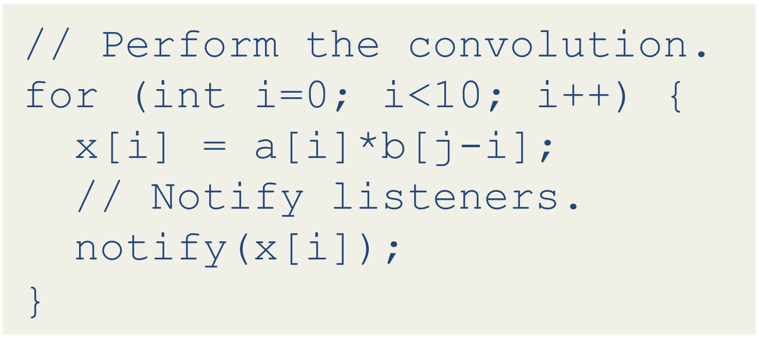
- PTIDES: distributed real-time software
  - Deterministic timing across networks
- PRET machines
  - Deterministic timing at the processor level

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

The synchronous digital logic abstraction removes the messiness of transistors.



… but the overlaying software abstractions discard the timing precision.



```
// Perform the convolution.
for (int i=0; i<10; i++) {
  x[i] = a[i]*b[j-i];
  // Notify listeners.
  notify(x[i]);
}
```

PRET Machines – Giving 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 bounded effects on timing

PRET Publications

PRET ISA Realizations:

- PRET1, Sparc-based
  - [Lickly et al., CASES, 2008]
- PTARM, ARM-based
  - [Liu et al., ICCD, 2012]
- FlexPRET, RISC-V-based
  - [Zimmer et al., RTAS, 2014, 2015]

PRET Applications:

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

PRET for Security:

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

PRET Memory Systems:

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

PRET Principle:

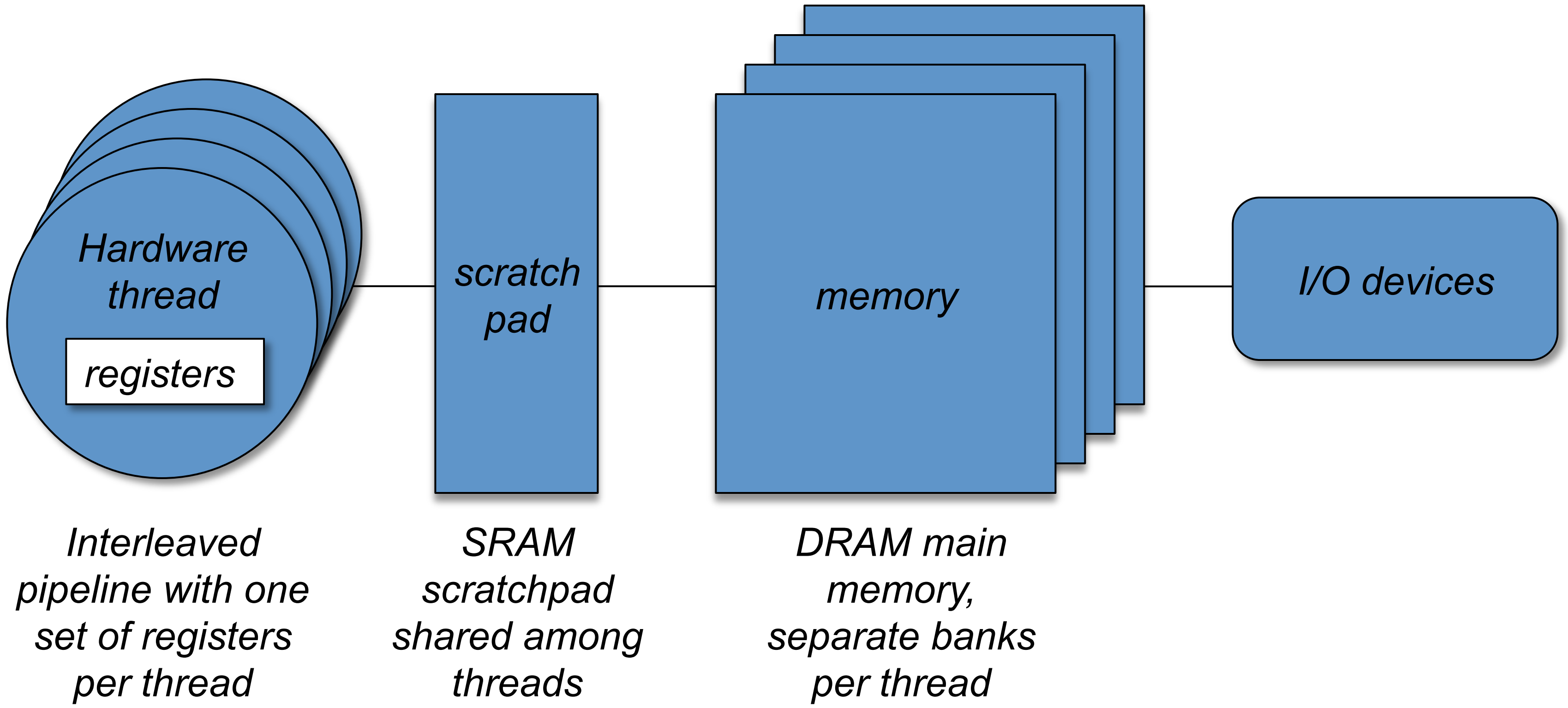
- *The case for PRET*
  - *[Edwards & Lee, DAC 2007]*
- *PRET ISA extensions*
  - *[Edwards at al., ICCD 2009]*
- *Temporal isolation*
  - *[Bui et al., DAC, 2011]*
- *Design challenges*
  - *[Broman et al., ESLsyn, 2013]*
- *Cyber-physical systems*
  - *[Lee., Sensors, 2015]*

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)



Interleaved pipeline with one set of registers per thread

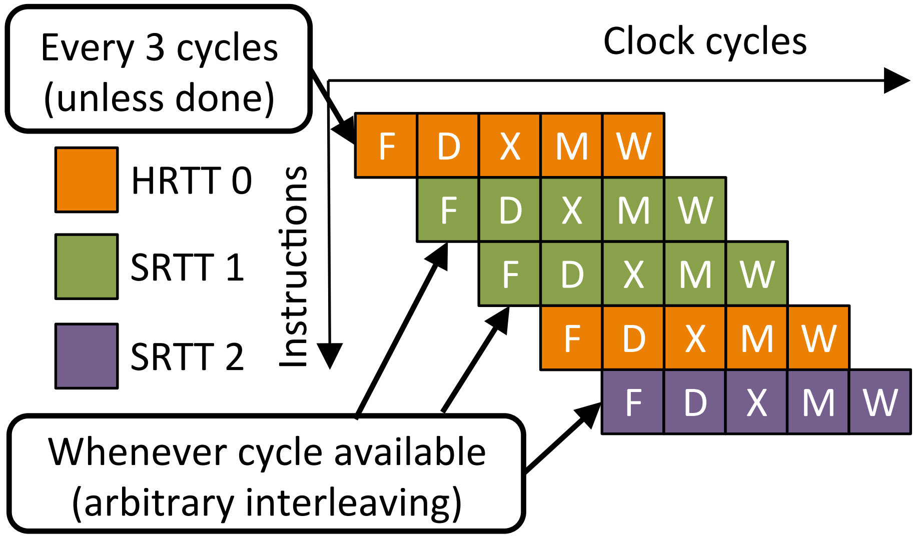
SRAM scratchpad shared among threads

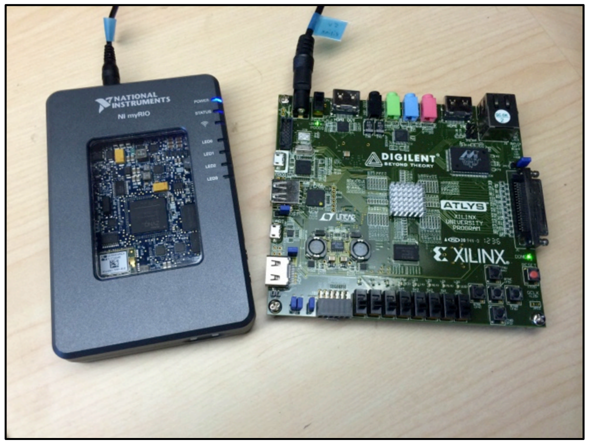
DRAM main memory, separate banks per thread

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 (area comparable to Microblaze)

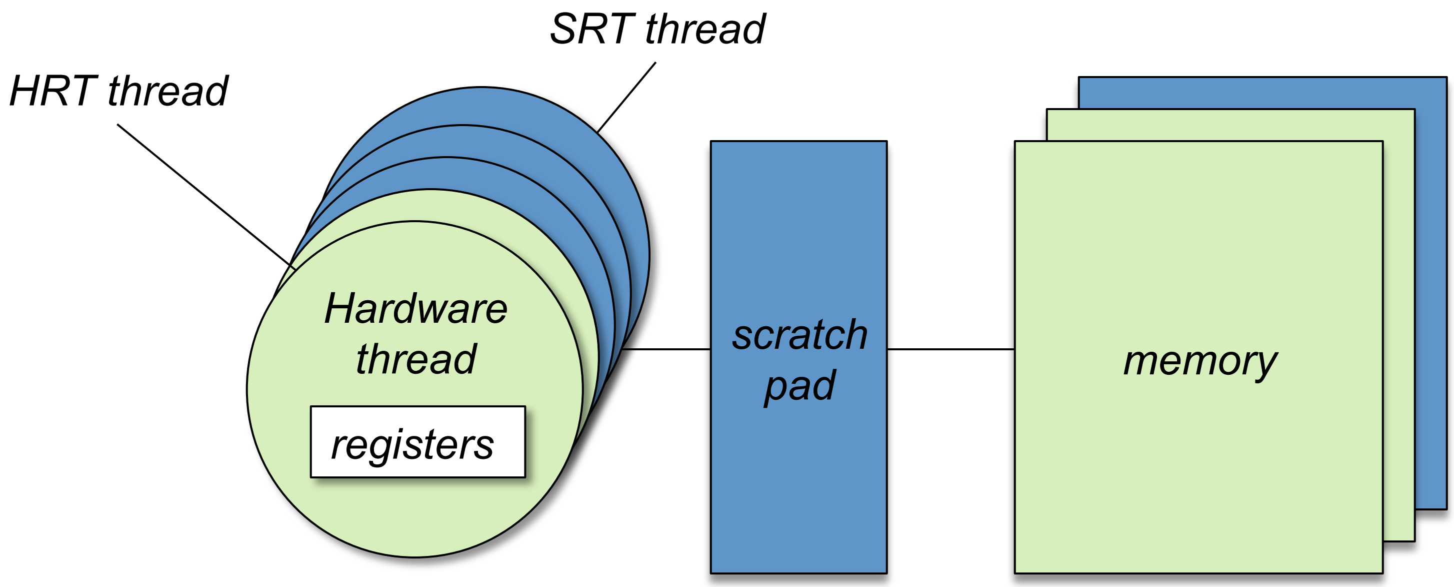




Digilent Atlys (Spartan 6) and NI myRIO (Zync)

FlexPRET

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



HRT threads have deterministic timing. SRT threads share available cycles

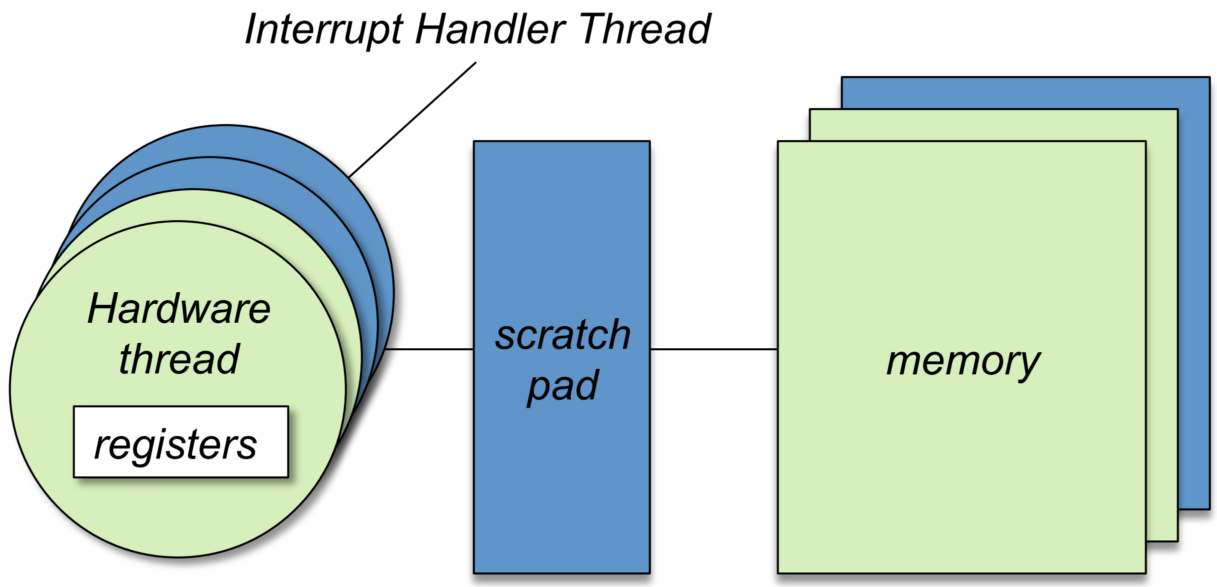
SRAM scratchpad shared among threads

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

Michael Zimmer

FlexPRET I/O

Interrupt-Driven I/O is notorious for disrupting timing



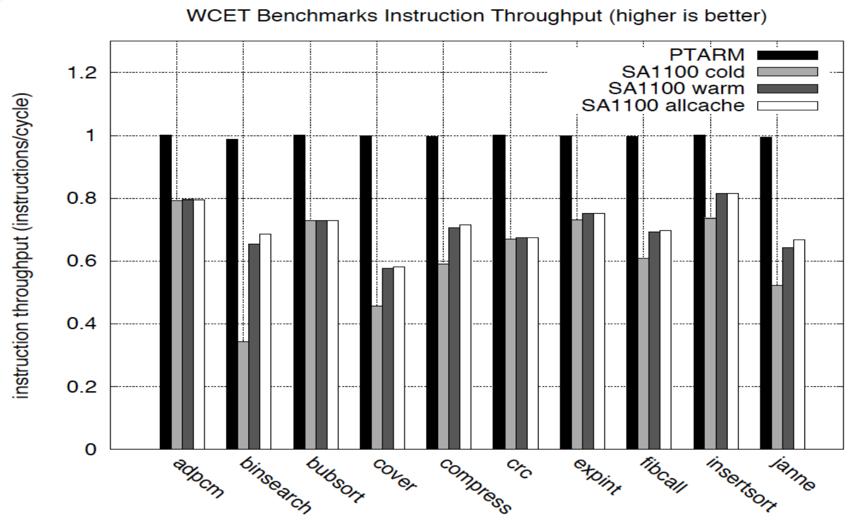
Interrupts have
no effect on HRT threads, and
bounded effect on SRT threads!

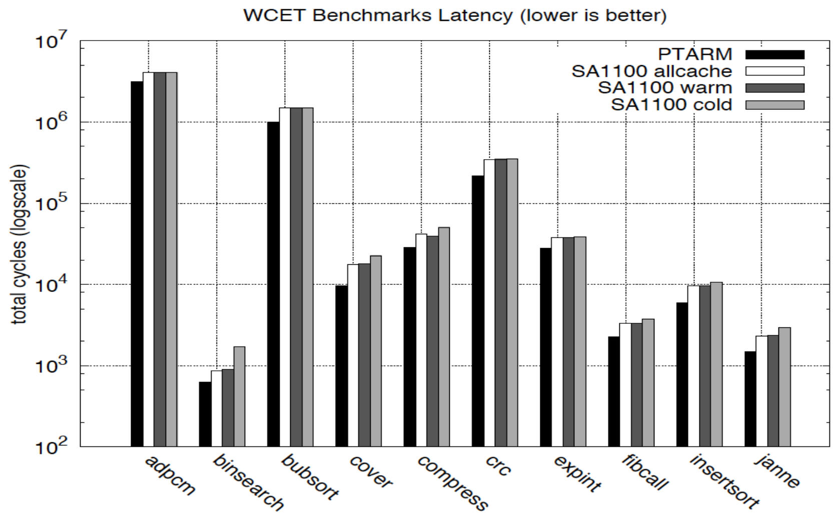
Michael Zimmer

Performance Cost?

No!

The PRET project has shown that you do not need to sacrifice performance to get control over timing.





[Isaac Liu, PhD Thesis, May, 2012]

FlexPRET Shows:

- Not only is there no performance cost for appropriate workloads, but there is also no performance cost for inappropriate workloads!

- Pipelining, memory hierarchy, and interrupt-driven I/O can all be done without losing timing determinacy!

Conclusion

Deterministic models with **highly faithful implementations** exist for distributed real-time systems.

- PTIDES: distributed real-time software
  - Deterministic timing across networks
- PRET machines
  - Deterministic timing at the processor level

We have run out of excuses for sloppy engineering!