Time for High-Confidence Cyber-Physical Systems

Edward A. Lee

Robert S. Pepper Distinguished Professor
UC Berkeley

Invited Plenary Talk

Performance Metrics for Intelligent Systems
(PerMIS'12) Workshop
University of Maryland
March 20-22, 2012.

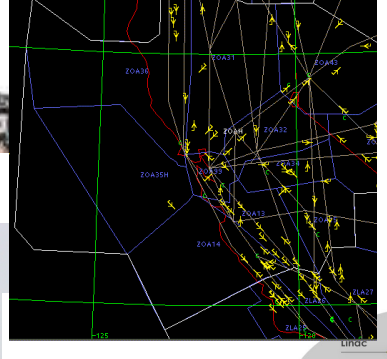
Key Collaborators on work shown here:

- *Steven Edwards*
- *Jeff Jensen*
- *Sungjun Kim*
- *Isaac Liu*
- *Slobodan Matic*
- *Hiren Patel*
- *Jan Reinke*
- *Sanjit Seshia*
- *Mike Zimmer*
- *Jia Zou*

Cyber-Physical Systems (CPS):

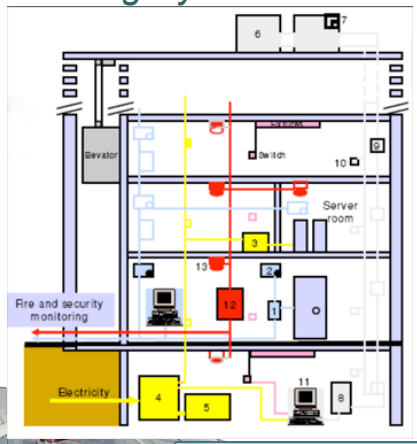
Orchestrating networked computational resources with physical systems

Avionics



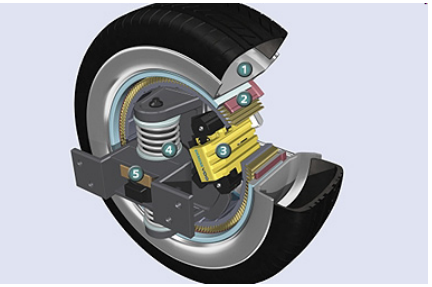
Transportation (Air traffic control at SFO)

Building Systems

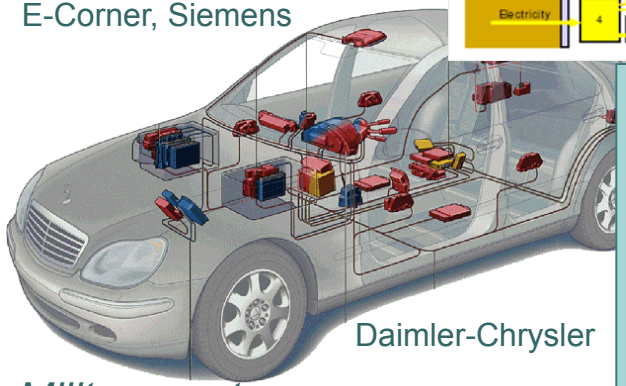


Telecommunications

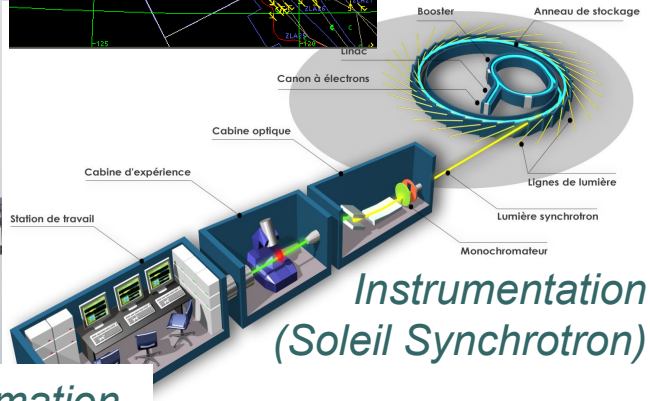
Automotive



E-Corner, Siemens

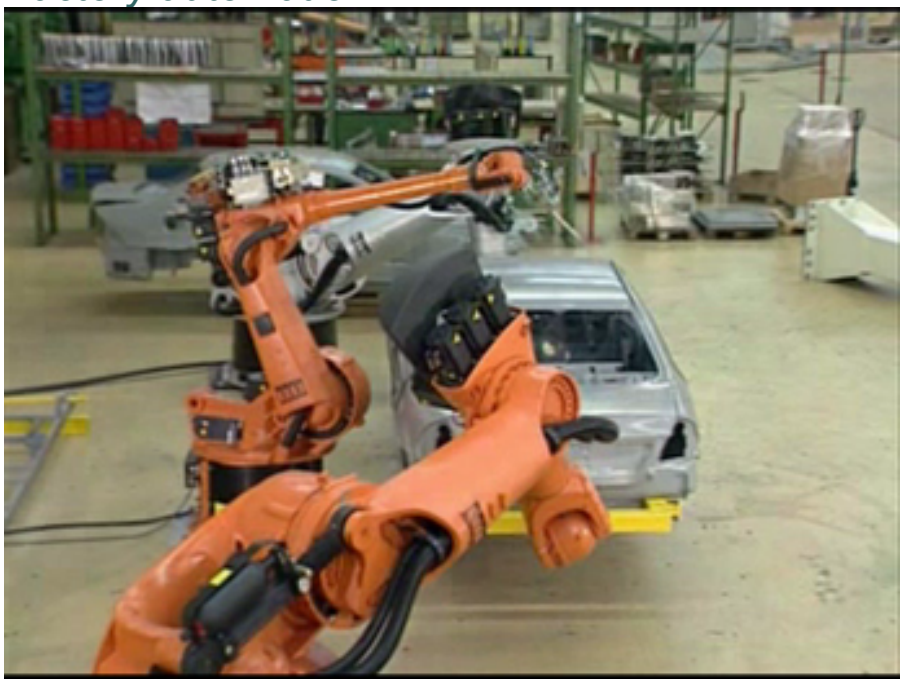


Daimler-Chrysler



Instrumentation (Soleil Synchrotron)

Factory automation

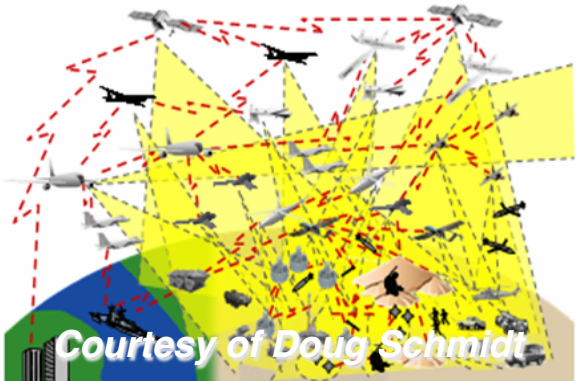


Courtesy of Kuka Robotics Corp.

Power generation and distribution

Courtesy of General Electric

Military systems:



Claim

For CPS, *programs* do not adequately specify *behavior*.

Corollary: *Performance* of program execution may not be a good metric.

A Story

The Boeing 777 was Boeing's first fly-by-wire aircraft, controlled by software. It is deployed, appears to be reliable, and is succeeding in the marketplace. Therefore, it must be a success. However…

Boeing was forced to purchase and store an advance supply of the microprocessors that will run the software, sufficient to last for the estimated 50 year production run of the aircraft and another many years of maintenance.

Why?

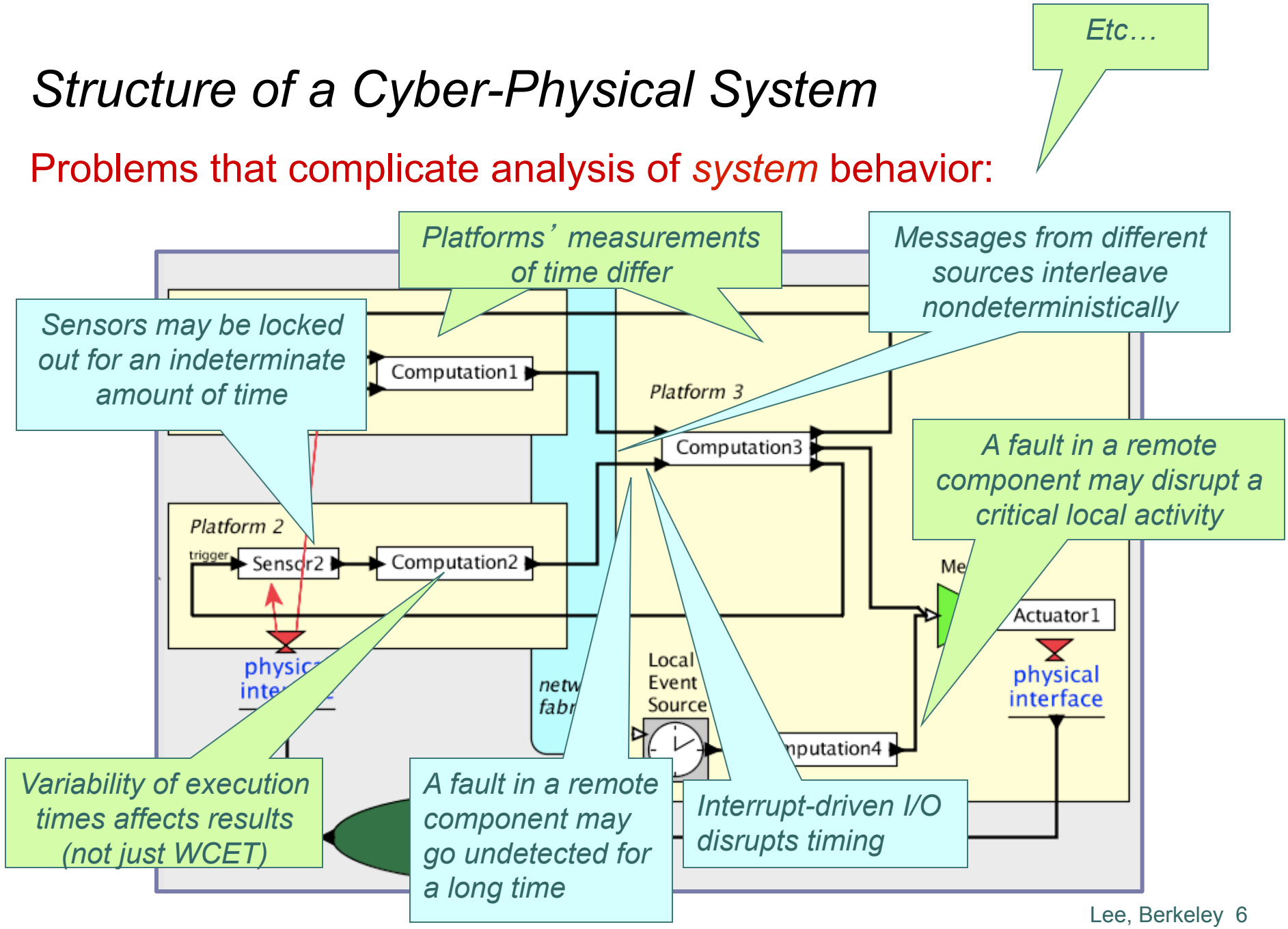
Lesson from this example:

Apparently, the software does not specify the behavior that has been validated and certified!

Unfortunately, this problem is very common, even with less safety-critical, certification-intensive applications. Validation is done on complete system implementations, not on software.

Structure of a Cyber-Physical System

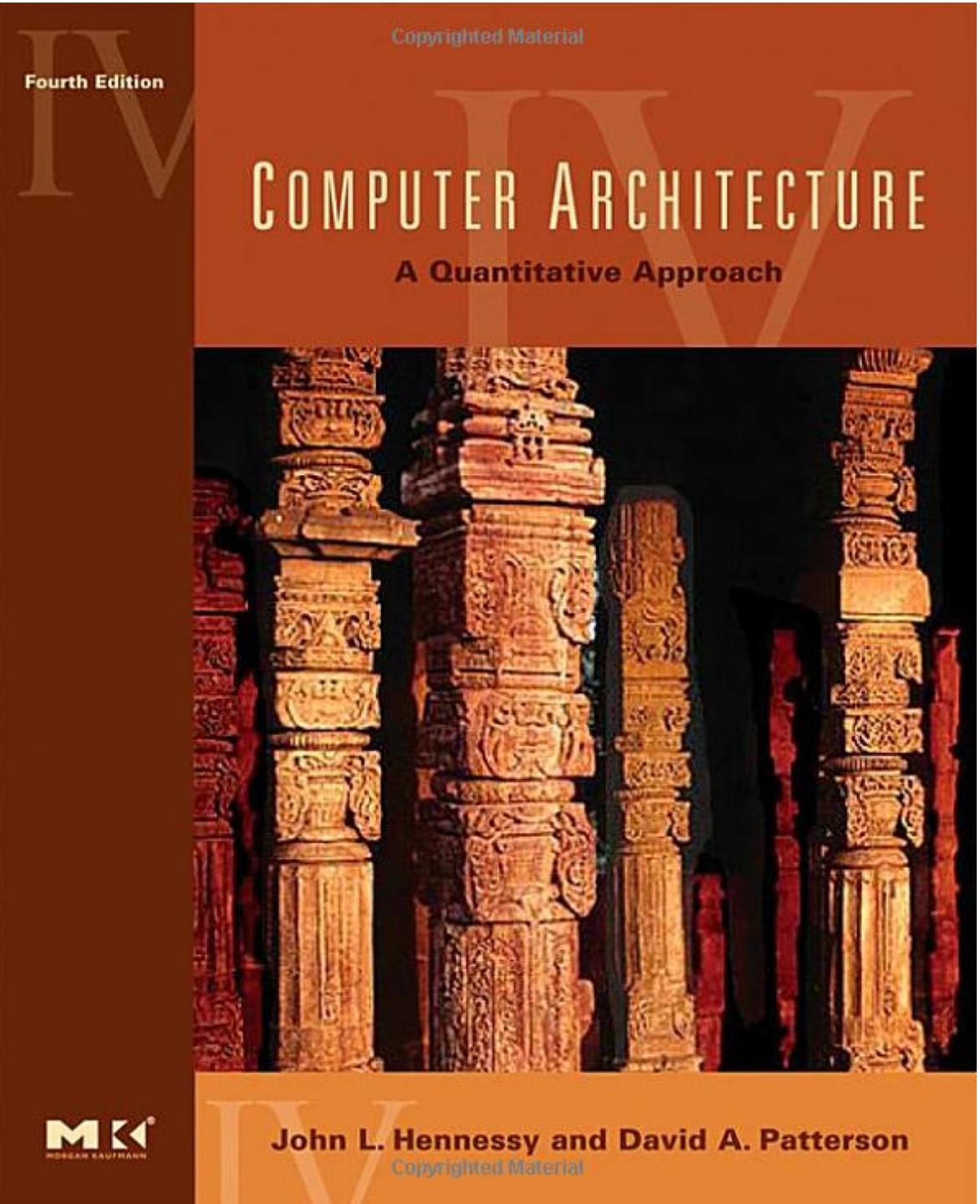
Problems that complicate analysis of *system* behavior:



A Key Challenge:
Timing is not Part of Software Semantics

Correct execution of a program in C, C#, Java, Haskell, OCaml, etc. has nothing to do with how long it takes to do anything. All our computation and networking abstractions are built on this premise.

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



The first edition of Hennessy and Patterson (1990) revolutionized the field of computer architecture by making performance metrics the dominant criterion for design.

Today, for computers, timing is merely a performance metric.

It needs to be a correctness criterion.

Execution-time analysis, by itself, does not solve the problem!

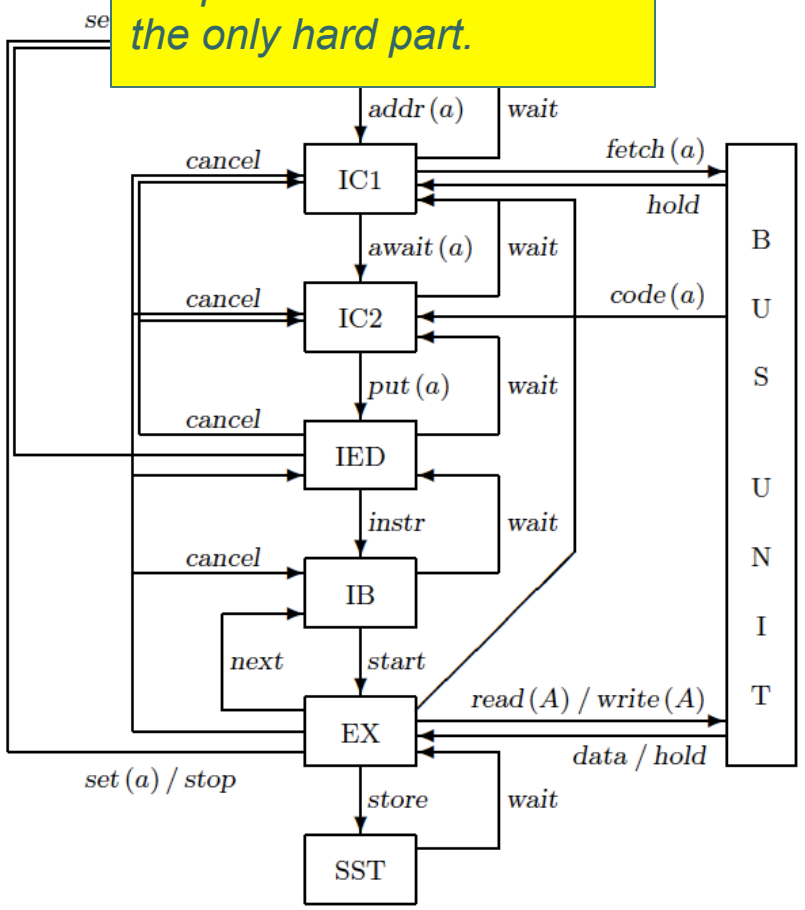
Analyzing software for timing behavior requires:

- Paths through the program (undecidable)
- Detailed model of microarchitecture
- Detailed model of the memory system
- Complete knowledge of execution context
- Many constraints on preemption/concurrency
- Lots of time and effort

And the result is valid only for that exact hardware and software!

Fundamentally, the ISA of the processor has failed to provide an adequate abstraction.

Our first goal is to reduce the problem so that this is the only hard part.



Wilhelm, et al. (2008). "The worst-case execution-time problem - overview of methods and survey of tools." ACM TECS 7 (3): p1-53.

Part 1: PRET Machines

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

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

+

= PRET

Computing *With time*

Dual Approach

- Rethink the ISA
  - Timing has to be a *correctness* property not a *performance* property.
- Implementation has to allow for multiple realizations and efficient realizations of the ISA
  - Repeatable execution times
  - Repeatable memory access times

Example of one sort of mechanism we would like:

```
tryin (500ms) {
   // Code block
} catch {
    panic();
}
```

```
jmp_buf  buf;

if ( !setjmp(buf) ){
  set_time r1, 500ms
  exception_on_expire r1, 0
  // Code block
  deactivate_exception 0
} else {
    panic();
}

exception_handler_0 () {
     longjmp(buf)
}
```

If the code block takes longer than 500ms to run, then the panic() procedure will be invoked.

But then we would like to verify that panic() is never invoked!

Pseudocode showing how this might be implemented today. The result is very platform dependent.

Extending an ISA with Timing Semantics

[V1] Best effort:

```
set_time r1, 1s
// Code block
delay_until r1
```

[V2] Late miss detection

```
set_time r1, 1s
// Code block
branch_expired r1, <target>
delay_until r1
```

[V3] Immediate miss detection

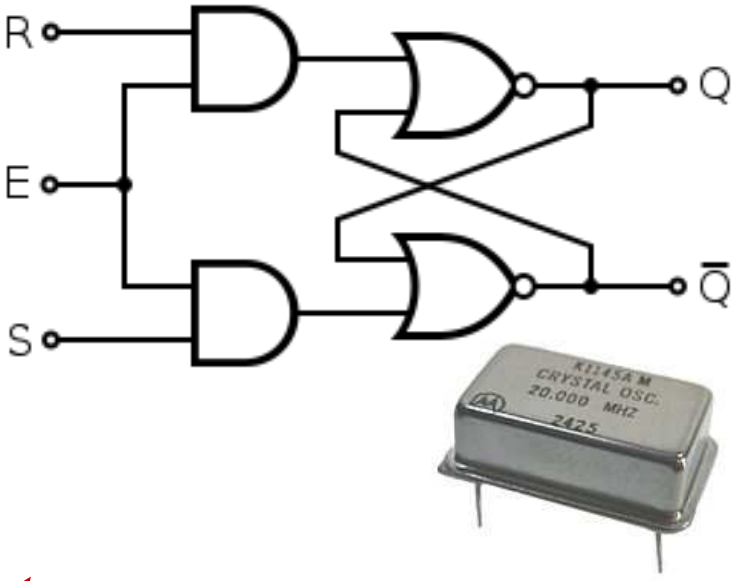
```
set_time r1, 1s
exception_on_expire r1, 1
// Code block
deactivate_exception 1
delay_until r1
```

[V4] Exact execution:

```
set_time r1, 1s
// Code block
MTFD r1
```

To provide timing guarantees, we need implementations that deliver repeatable timing

Fortunately, electronics technology delivers highly reliable and precise timing…



… but the overlaying software abstractions discard it. Chip architects heavily exploit the lack of temporal semantics.

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

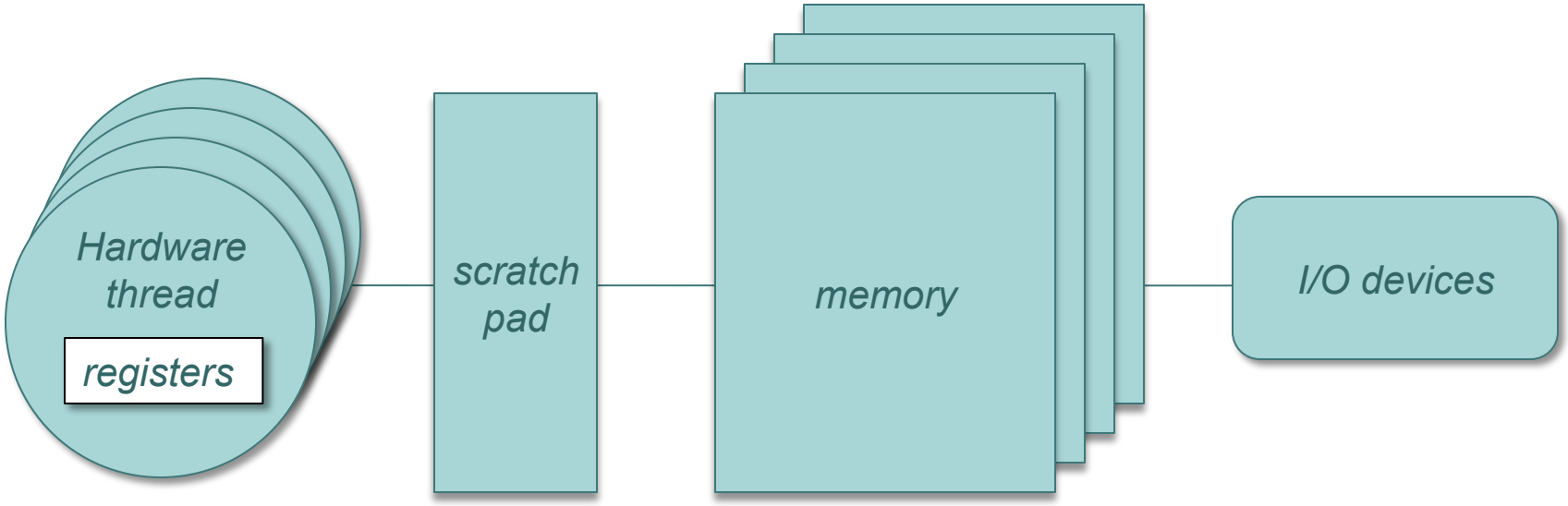
To deliver repeatable timing, we have to rethink the microarchitecture

Challenges:

- Pipelining
- Memory hierarchy
- I/O (DMA, interrupts)
- Power management (clock and voltage scaling)
- On-chip communication
- Resource sharing (e.g. in multicore)

Our Current PRET Architecture

PTArm, a soft core on a
Xilinx Virtex 5 and 6 FPGA



Status of the PRET project

- Results:
  - PTArm implemented on Xilinx Virtex 5 FPGA.
  - UNISIM simulator of the PTArm facilitates experimentation.
  - DRAM controller with repeatable timing and DMA support.
  - PRET-like utilities implemented on COTS Arm.

- Much still to be done:
  - Realize MTFD, interrupt I/O, compiler toolchain, scratchpad management, etc.

A Key Next Step: Parametric PRET Architectures

```
set_time r1, 1s
// Code block
MTFD r1
```

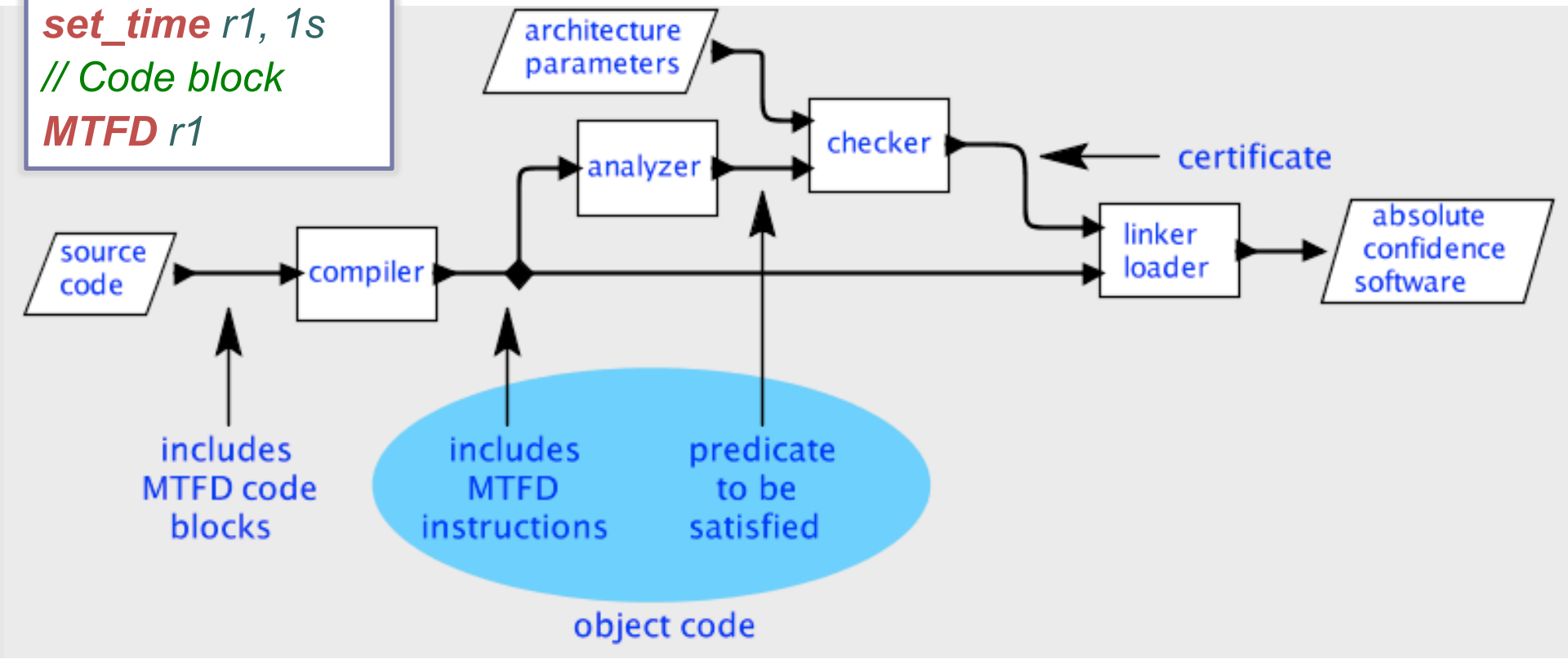
ISA that admits a variety of implementations:

- Variable clock rates and energy profiles
- Variable number of cycles per instruction
- Latency of memory access varying by address
- Varying sizes of memory regions
- …

A given program may meet deadlines on only some realizations of the same parametric PRET ISA.

Realizing the MTFD instruction on a parametric PRET machine

```
set_time r1, 1s
// Code block
MTFD r1
```



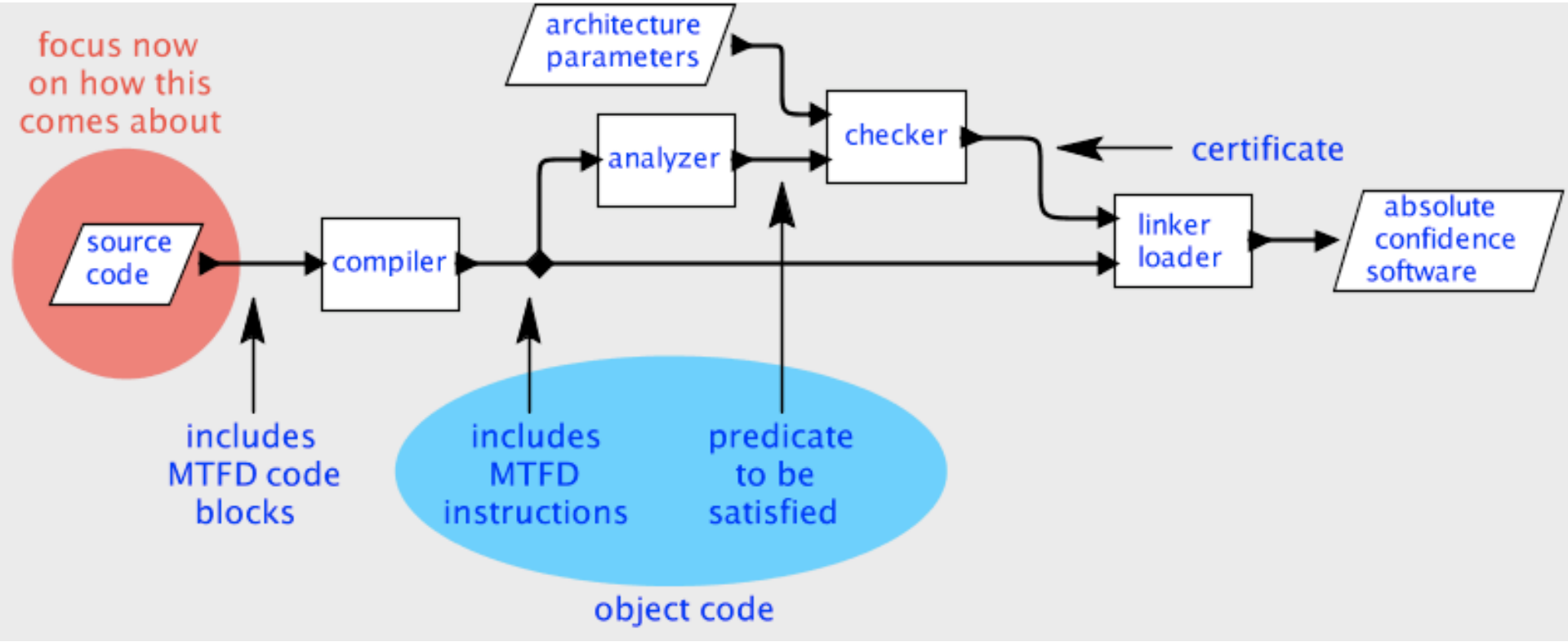
The goal is to make software that will run correctly on a variety of implementations of the ISA, and that correctness can be checked for each implementation.

PRET Publications

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

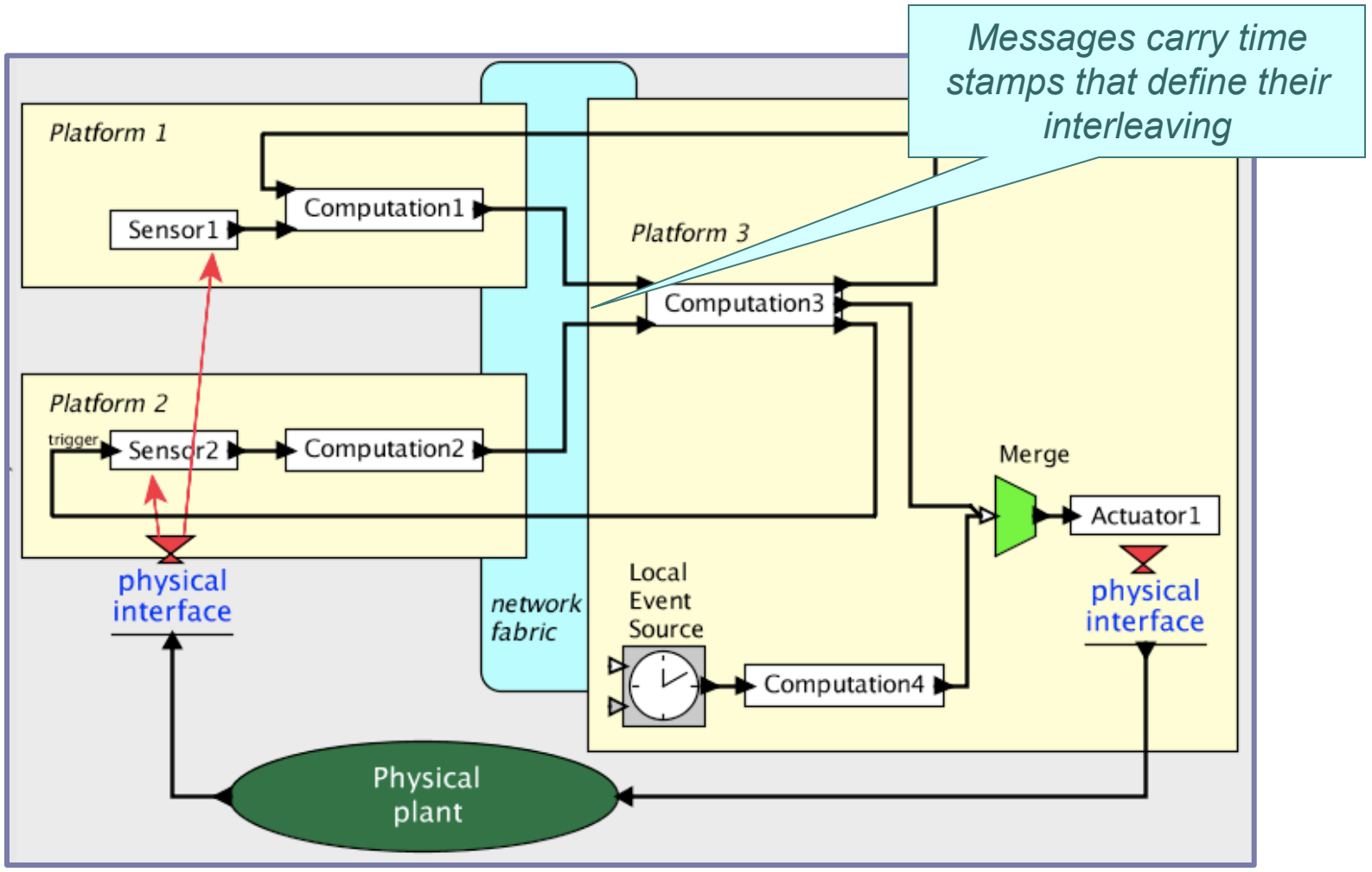
- S. Edwards and E. A. Lee, "**The Case for the Precision Timed (PRET) Machine**," in the *Wild and Crazy Ideas* Track of DAC, June 2007.
- B. Lickly, I. Liu, S. Kim, H. D. Patel, S. A. Edwards and E. A. Lee, "**Predictable programming on a precision timed architecture**," CASES 2008.
- S. Edwards, S. Kim, E. A. Lee, I. Liu, H. Patel and M. Schoeberl, "**A Disruptive Computer Design Idea: Architectures with Repeatable Timing**," ICCD 2009.
- D. Bui, H. Patel, and E. Lee, "**Deploying hard real-time control software on chip-multiprocessors**," RTCSA 2010.
- Bui, E. A. Lee, I. Liu, H. D. Patel and J. Reineke, "**Temporal Isolation on Multiprocessing Architectures**," DAC 2011.
- J. Reineke, I. Liu, H. D. Patel, S. Kim, E. A. Lee, **PRET DRAM Controller: Bank Privatization for Predictability and Temporal Isolation** (to appear), CODES +ISSS, Taiwan, October, 2011.
- S. Bensalem, K. Goossens, C. M. Kirsch, R. Obermaisser, E. A. Lee, J. Sifakis, **Time-Predictable and Composable Architectures for Dependable Embedded Systems**, Tutorial Abstract (to appear), EMSOFT, Taiwan, October, 2011

Part 2: How to get the Source Code?

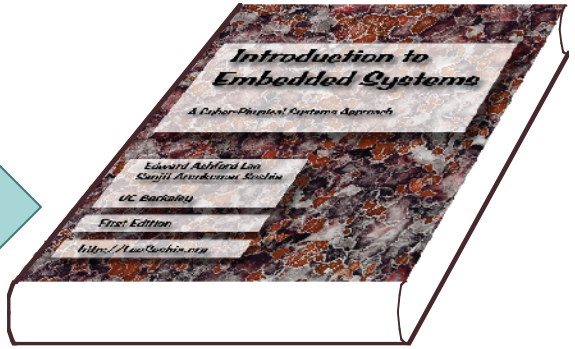


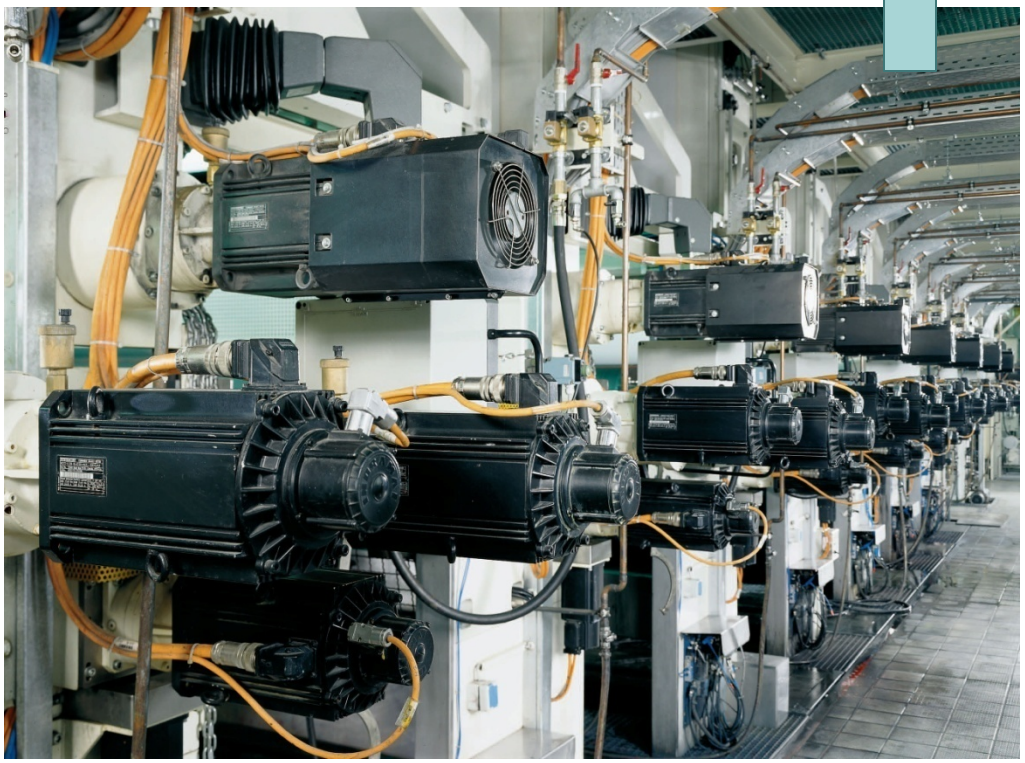
The input (mostly likely C) will ideally be generated from a model, like Simulink or SCADE. The model specifies temporal behavior at a higher level than code blocks, and it specifies a concurrency model that can limit preemption points. However, Simulink and SCADE have naïve models of time.

Ptides: Programming model for distributed real-time systems, using time-stamped messages.



A CPS Problem and Potential Ptides Application: Printing Press





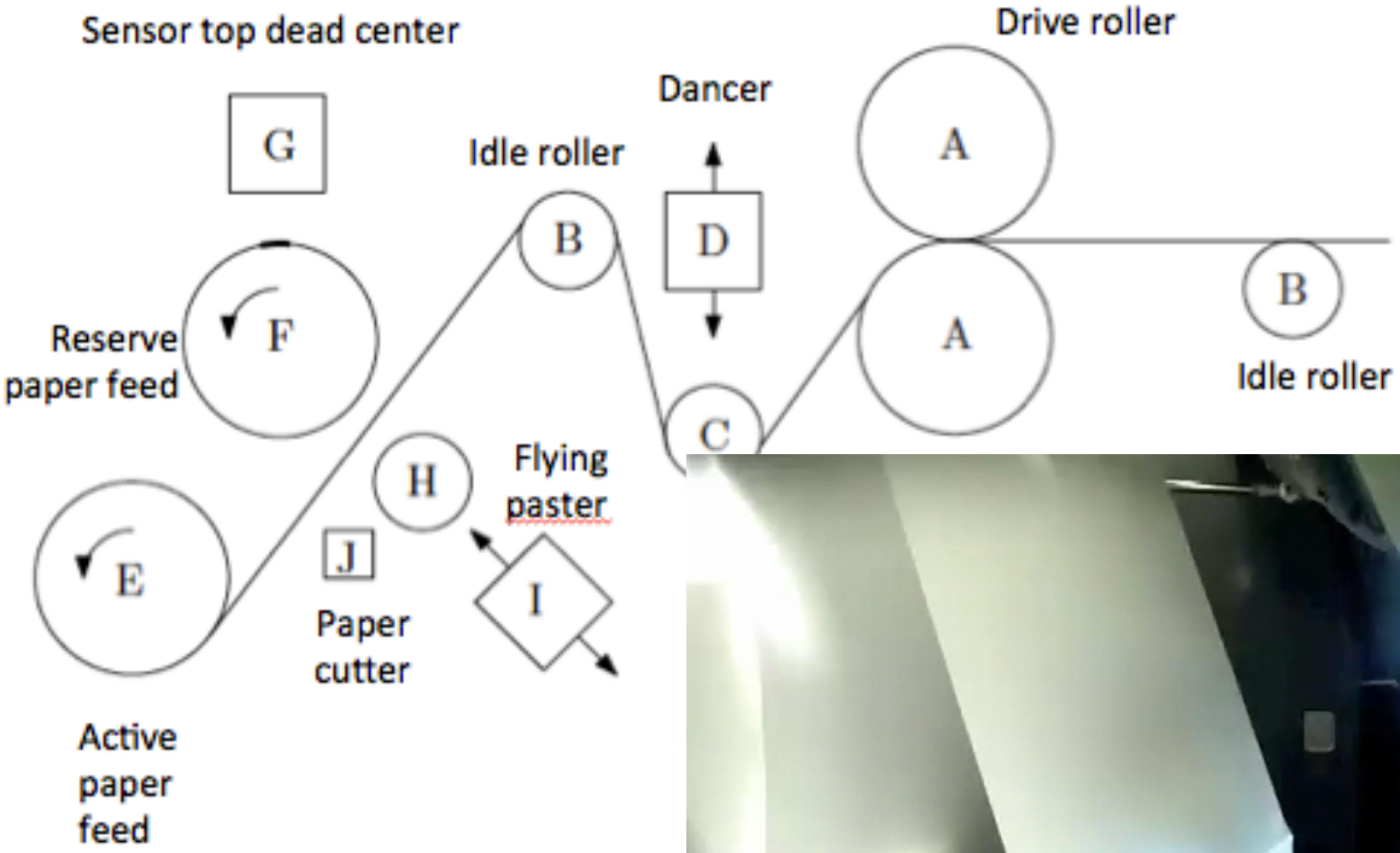
Bosch-Rexroth

- Application aspects
  - local (control)
  - distributed (coordination)
  - global (modes)
- Open standards (Ethernet)
  - Synchronous, Time-Triggered
  - IEEE 1588 time-sync protocol
- High-speed, high precision
  - Speed: 1 inch/ms
  - Precision: 0.01 inch

    -> Time accuracy: 10us

Goal: Orchestrated networked resources built with **sound design principles** on **suitable abstractions**

Example – Flying Paster



Source: http://offsetpressman.blogspot.com/2011/03/how-flying-paster-works.htr

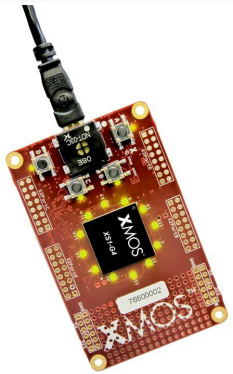
Source: http://offsetpressman.blogspot.com/2011/03/how-flying-paster-works.html

Flying Paster

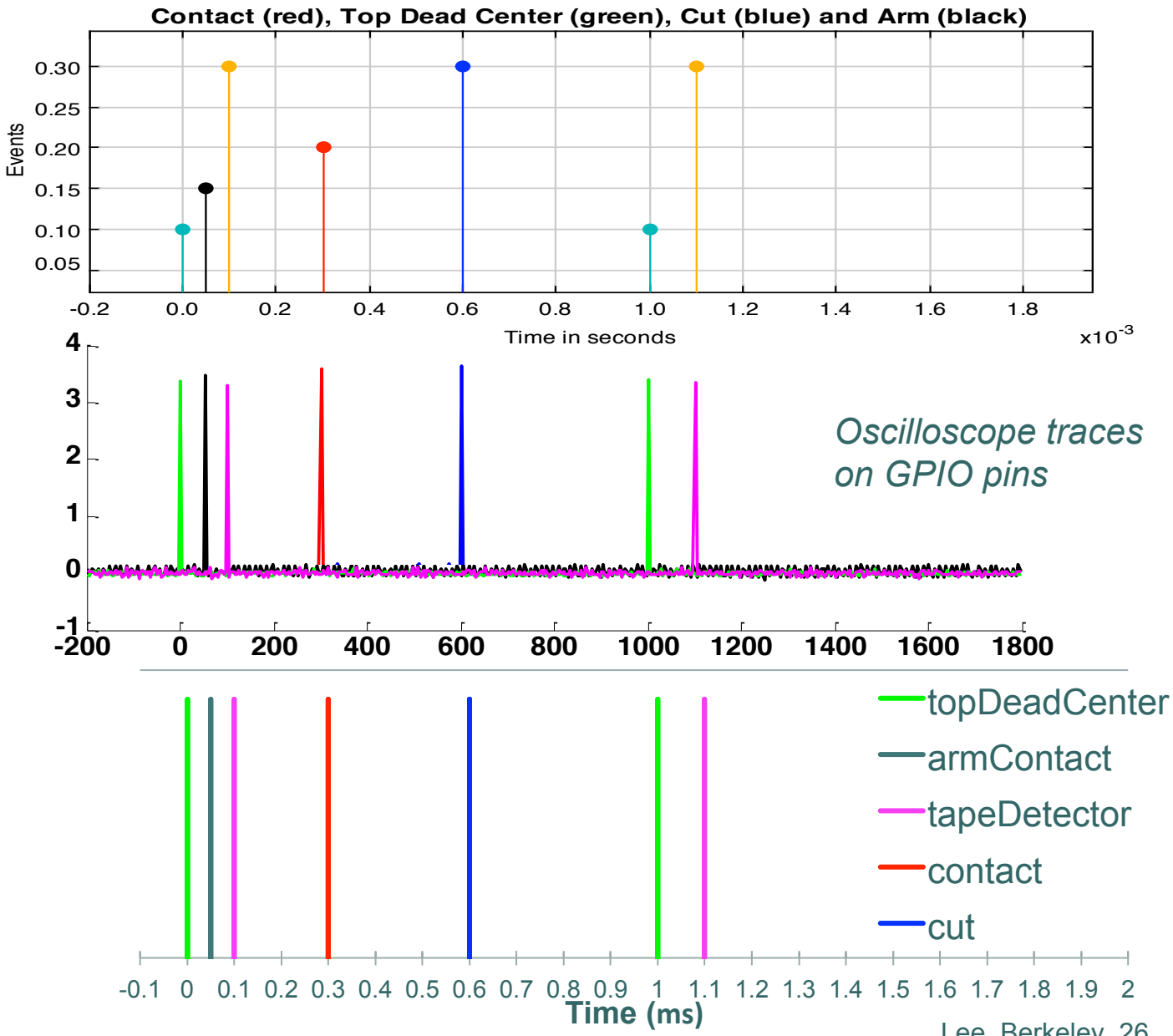
What this needs is *correct* timing, not *high performance.*

Simulation

Renesas



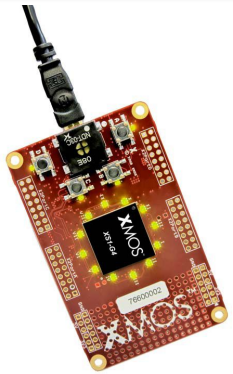
XMOS



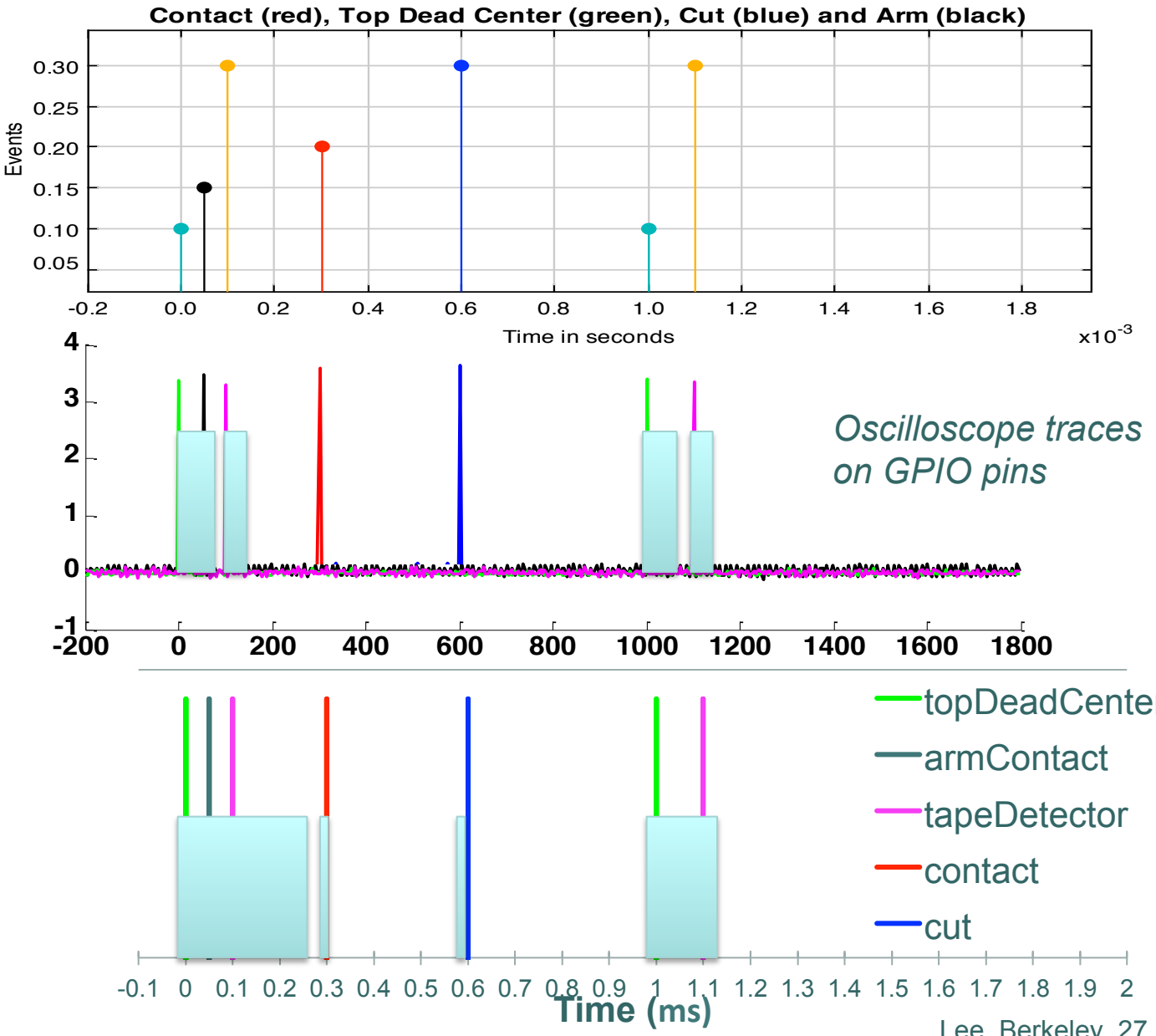
We have demonstrated platform independent timing for programs automatically generated from model.

Simulation

Renesas



XMOS



Ptides Publications

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

- Y. Zhao, J. Liu, E. A. Lee, "**A Programming Model for Time-Synchronized Distributed Real-Time Systems**," RTAS 2007.
- T. H. Feng and E. A. Lee, "**Real-Time Distributed Discrete-Event Execution with Fault Tolerance**," RTAS 2008.
- P. Derler, E. A. Lee, and S. Matic, "**Simulation and implementation of the ptides programming model**," DS-RT 2008.
- J. Zou, S. Matic, E. A. Lee, T. H. Feng, and P. Derler, "**Execution strategies for Ptides, a programming model for distributed embedded systems**," RTAS 2009.
- J. Zou, J. Auerbach, D. F. Bacon, E. A. Lee, "**PTIDES on Flexible Task Graph: Real-Time Embedded System Building from Theory to Practice**," LCTES 2009.
- J. C. Eidson, E. A. Lee, S. Matic, S. A. Seshia and J. Zou, "**Time-centric Models For Designing Embedded Cyber-physical Systems**," ACES-MB 2010.
- J. C. Eidson, E. A. Lee, S. Matic, S. A. Seshia, and J. Zou, **Distributed Real-Time Software for Cyber-Physical Systems**, To appear in *Proceedings of the IEEE* special issue on CPS, December, 2011.

Implications for Performance Metrics

- Performance metrics for computing have been oversimplified:
  - Minimize execution time on standard benchmarks.
  - Minimize energy or power on standard benchmarks.

- Ideas for revised performance metrics:
  - Timing precision (or variability) at I/O connections.
  - Precision and reliability of time synchronization.
  - Minimize energy subject to meeting timing constraints.
  - Timing variability across platform implementations.

  This will require new benchmarks!

Overview References:

- *Lee. **Computing needs time**. CACM, 52(5):70–79, 2009*
- *Eidson et. al, Distributed Real-Time Software for Cyber-Physical Systems, Proc. of the IEEE, January, 2012.*

Conclusions

Today, timing behavior is a property only of *realizations* of software systems.

Tomorrow, timing behavior will be a semantic property of *programs* and *models*.

Raffaello Sanzio da Urbino – The Athens School