



Time for High-Confidence Software Systems

Edward A. Lee

*Robert S. Pepper Distinguished Professor
UC Berkeley*

Keynote talk

*11th Annual Conference on
High Confidence Software and Systems,
Annapolis, Maryland, May 1-6, 2011*

Key Collaborators:

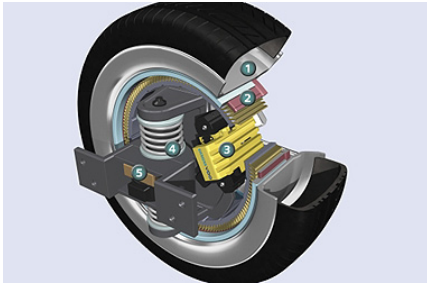
- *Steven Edwards*
- *Sungjun Kim*
- *Isaac Liu*
- *Slobodan Matic*
- *Jan Reinke*
- ***Sanjit Seshia***
- *Mike Zimmer*
- *Jia Zou*

This talk:

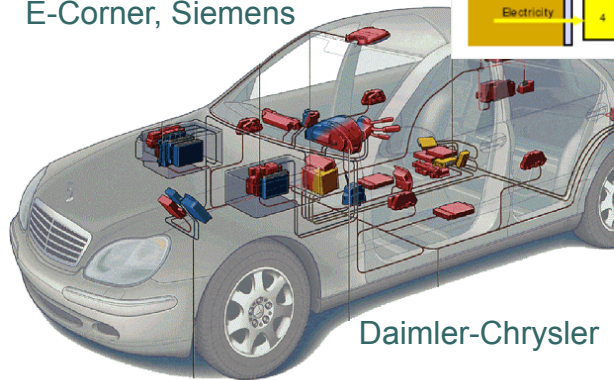
- Verifying software is not enough.
- We need to verify *systems*.
- And this requires rethinking software abstractions.
- Particularly: it's about time.

Cyber-Physical Systems (CPS): *Orchestrating networked computational resources with physical systems*

Automotive



E-Corner, Siemens



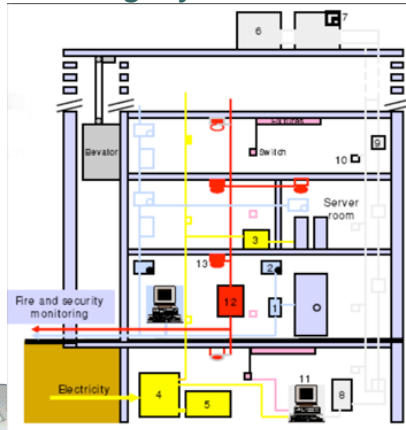
Daimler-Chrysler

Military systems:

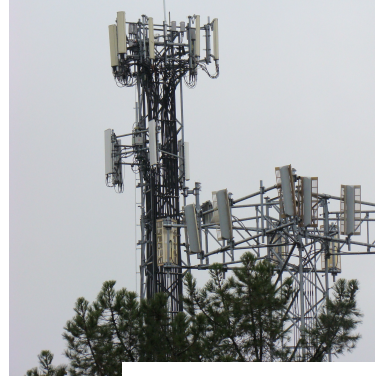


Courtesy of Doug Schmidt

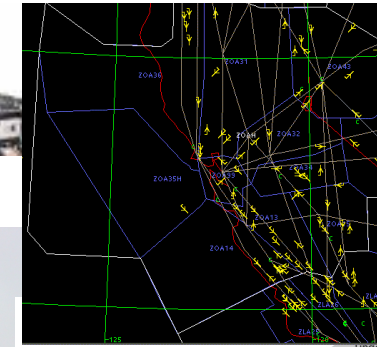
Building Systems



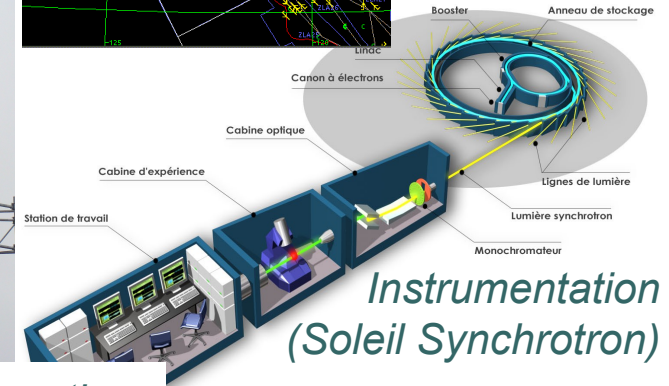
Telecommunications



Avionics

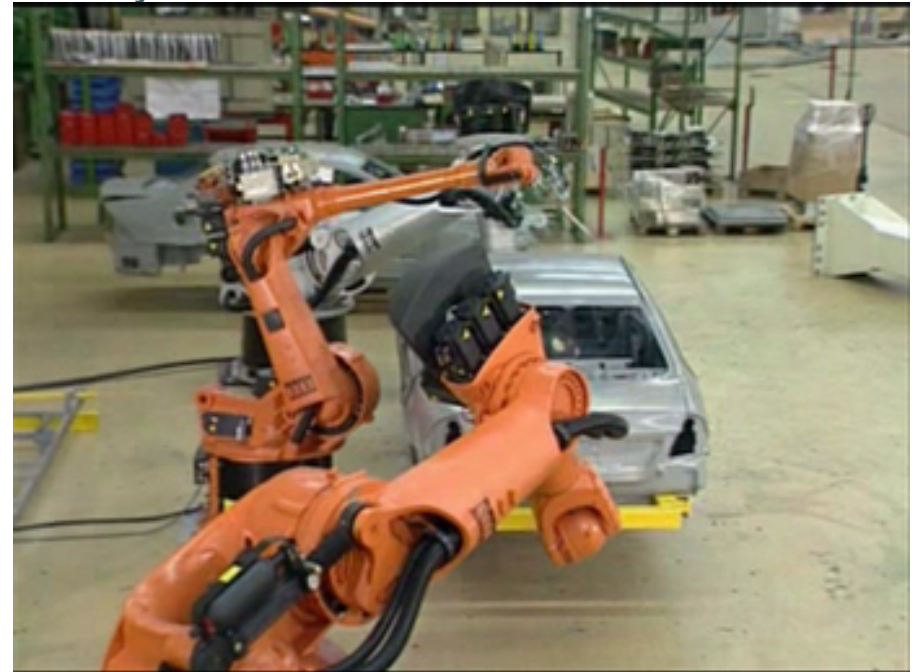


Transportation
(Air traffic
control at
SFO)



Instrumentation
(Soleil Synchrotron)

Factory automation



Courtesy of Kuka Robotics Corp.

Power generation and distribution



Courtesy of
General Electric

Claim

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

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:



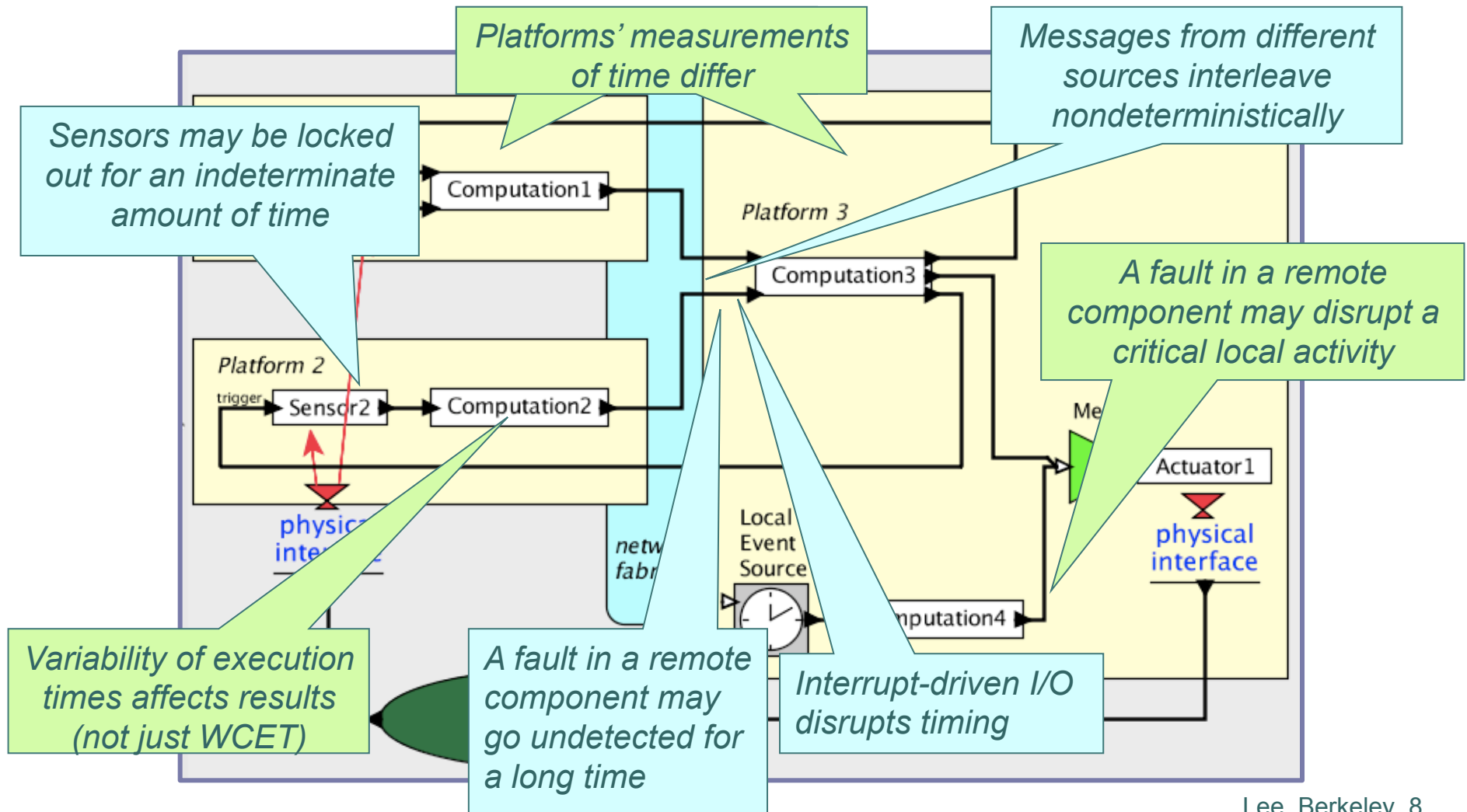
Apparently, the software does not specify the behavior that has been validated and certified!

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

Etc...

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.

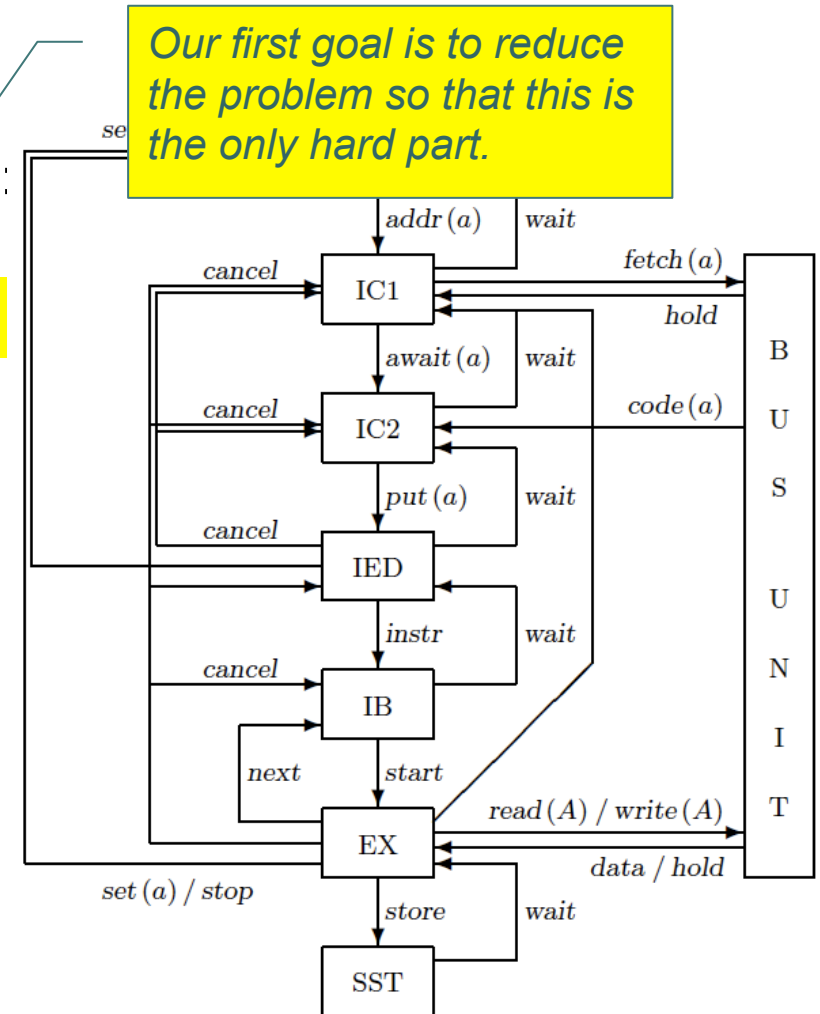
Execution-time analysis, by itself, does not solve the problem!

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.



C. Ferdinand et al., "Reliable and precise WCET determination for a real-life processor." EMSOFT 2001.

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 the mechanism in a mix of C and assembly.

Extending an ISA with Timing Semantics

[V1] Best effort:

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

[V3] Immediate miss detection

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

[V2] Late miss detection

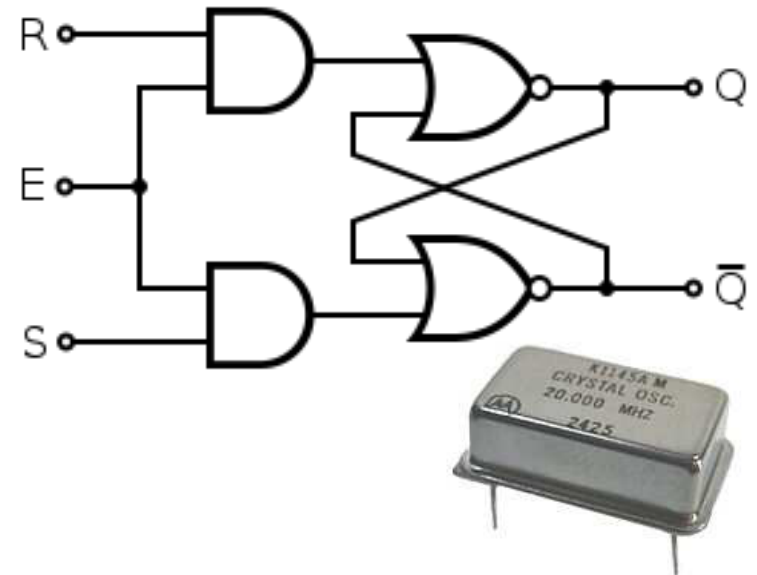
```
set_time r1, 1s  
// Code block  
branch_expired r1, <target>  
delay_until r1
```

[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.

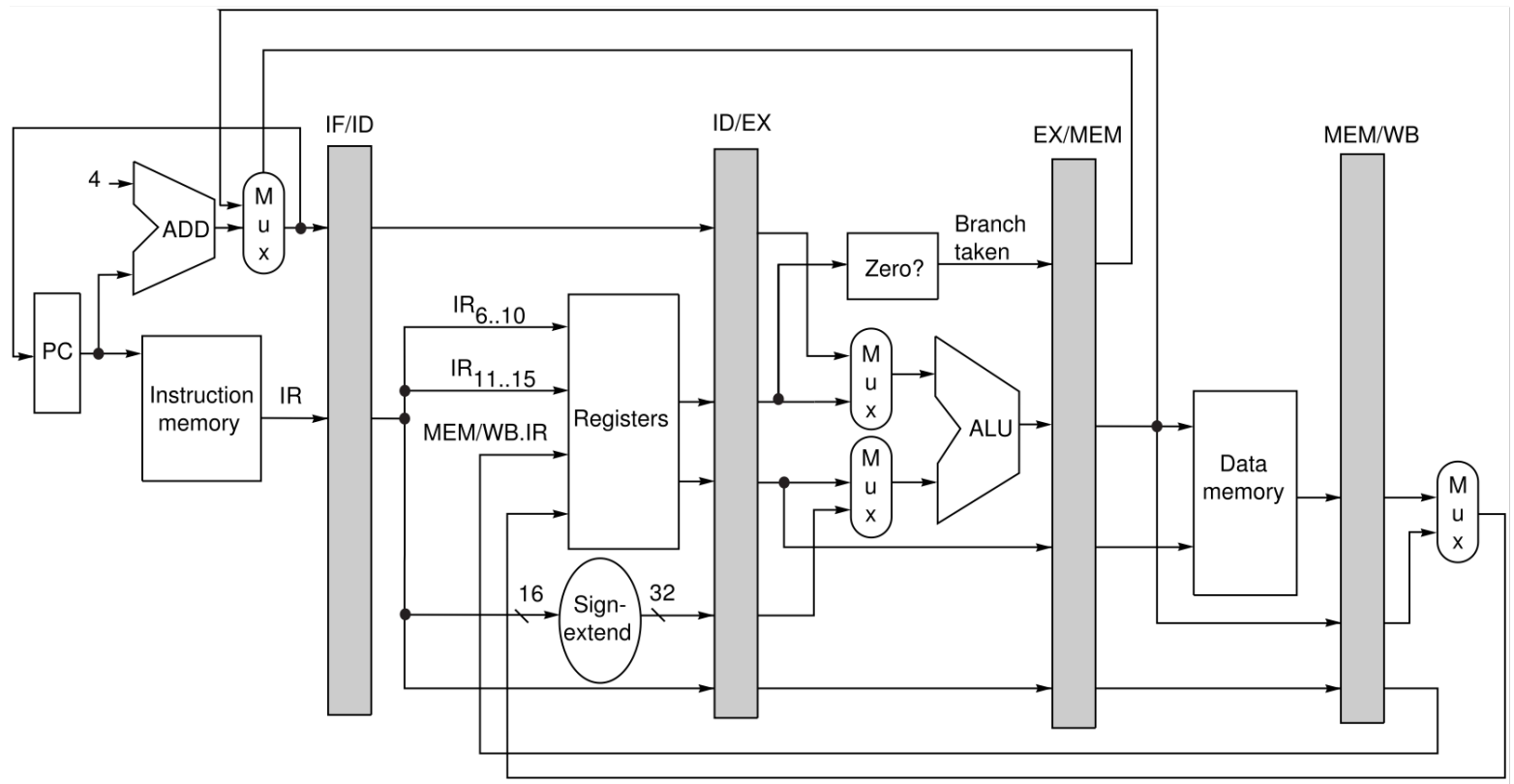
```
// Perform the convolution.
for (int i=0; i<10; i++) {
    x[i] = a[i]*b[j-i];
    // Notify listeners.
    notify(x[i]);
}
```


To deliver repeatable timing, we have to rethink the microarchitecture

Challenges:

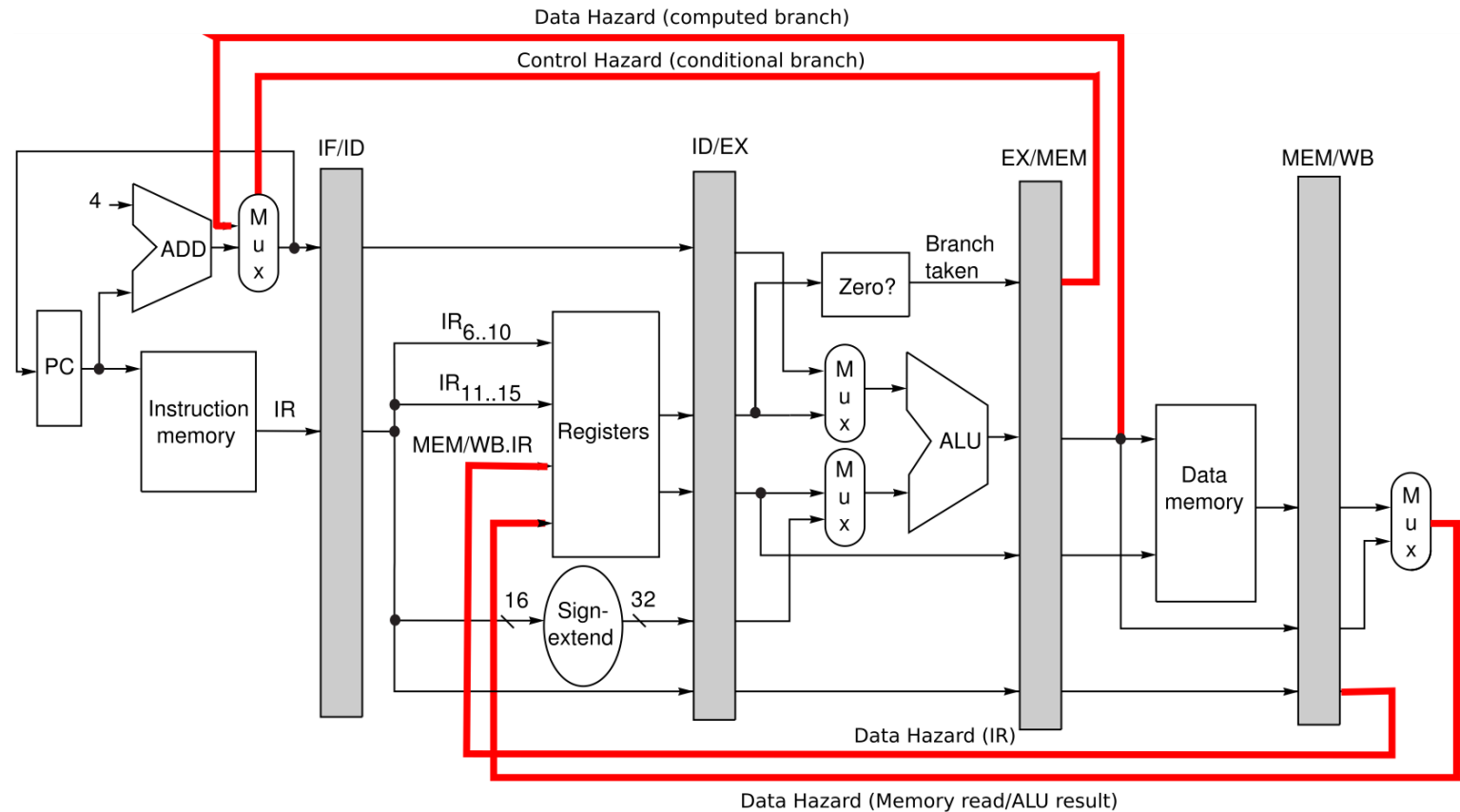
- Pipelining
- Memory hierarchy
- I/O (DMA, interrupts)
- Power management (clock and voltage scaling)
- On-chip communication
- Resource sharing (e.g. in multicore)

First Problem: Pipelining



Hennessey and Patterson, Computer Architecture: A Quantitative Approach, 4th edition, 2007.

Pipeline Hazards



Hennessey and Patterson, Computer Architecture: A Quantitative Approach, 4th edition, 2007.

An Alternative: Pipeline Interleaving

Traditional pipeline:

T0: cmp %g2, 9

T0: bg, a 40011b8

T0: add %i1, %i2, %l3

t	t+1	t+2	t+3	t+4	t+5	t+6	t+7	t+8	t+9	t+10	t+11
F	D	R	E	M	W						
	F	D	D	D	R	E	M	W			
				F	F	D	R	E	M	W	

Stall pipeline

Dependencies result in complex timing behaviors

Thread-interleaved pipeline:

T0: cmp %g2, 9

T1: add %o0, %g1, %g2

T2: sub %g1, %g2, %g1

T3: bn 430011a0

T4: ld [%fp + -12], %g1

T5: cmp %g1, 4

T0: bg, a 40011b8

T1: cmp %g1, 4

t	t+1	t+2	t+3	t+4	t+5	t+6	t+7	t+8	t+9	t+10	t+11
F	D	R	E	M	W						
	F	D	R	E	M	W					
		F	D	R	E	M	W				
			F	D	R	E	M	W			
				F	D	R	E	M	W		
					F	D	R	E	M	W	
						F	D	R	E	M	W
							F	D	R	E	M

Repeatable timing behavior of instructions

Pipeline Interleaving

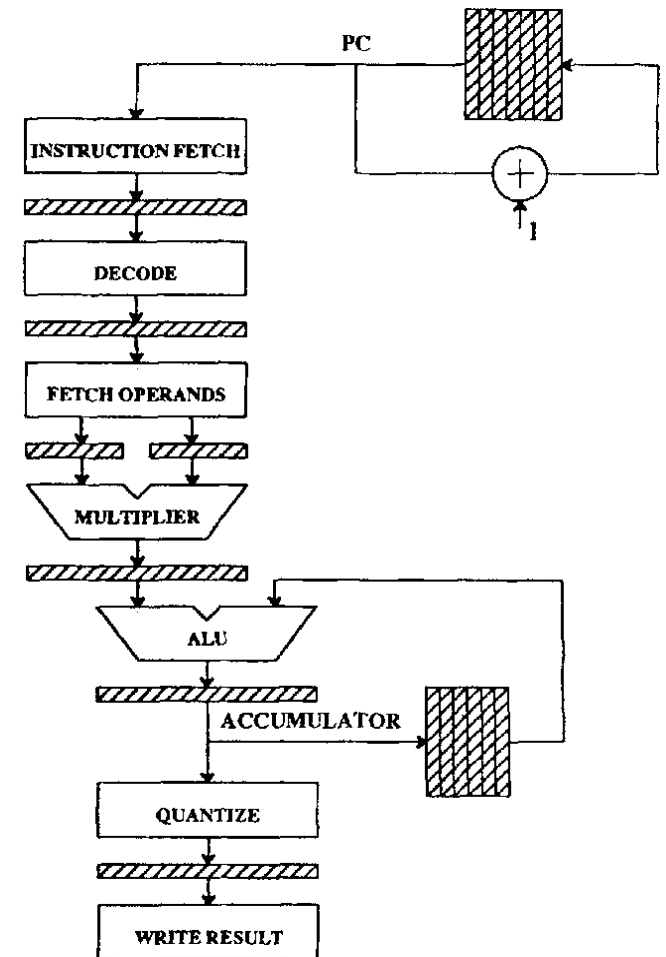
(Aka Hardware threads,
related to hyperthreading)

- History:

- CDC 6600
- Denelcore HEP
- ...
- Sandbridge Sandblaster
- XMOS

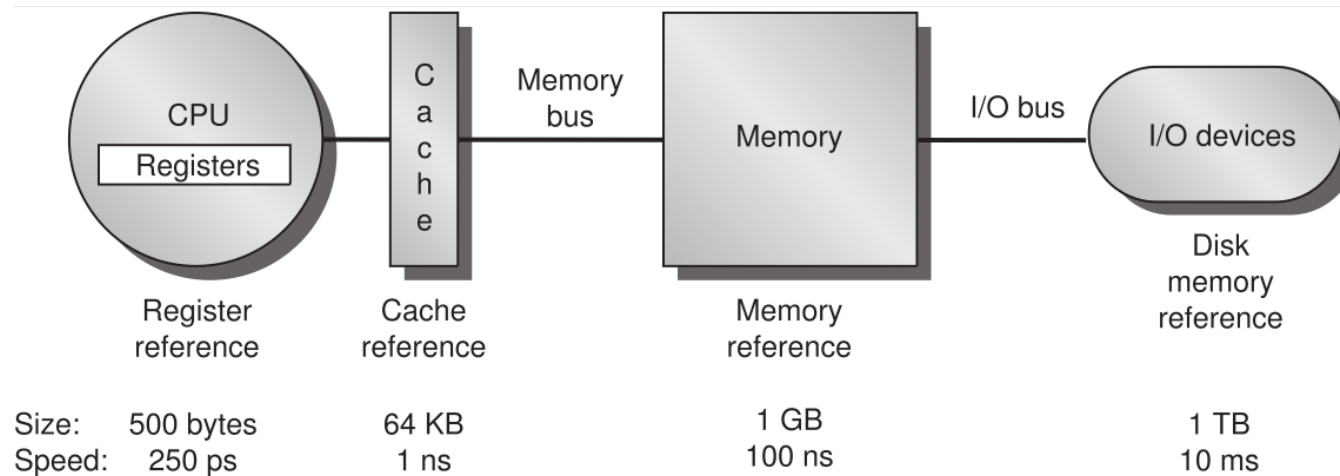
- Tradeoffs:

- + Simpler hardware (faster clocks)
- + Repeatable timing
- + Interference-free multithreading
- Slower single-thread performance



Lee and Messerschmitt, Pipeline Interleaved Programmable DSPs, ASSP-35(9), 1987.

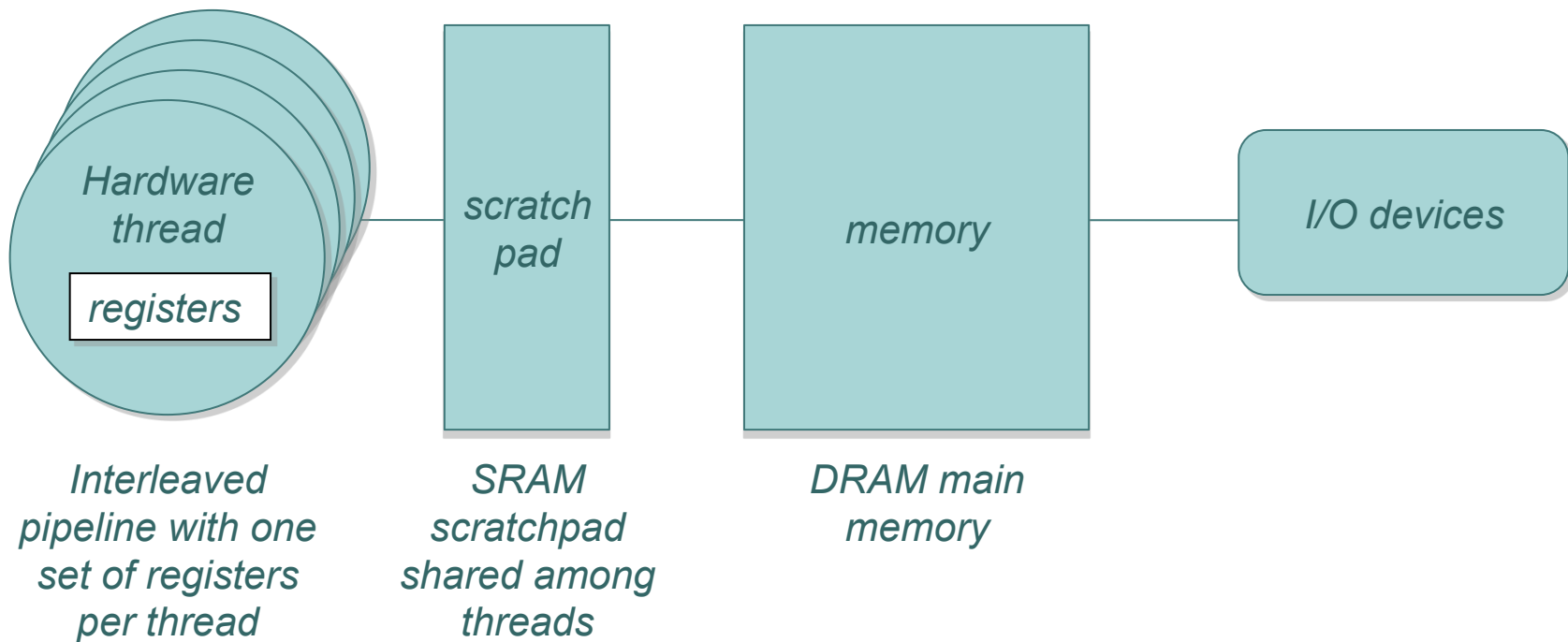
Second Problem: Memory Hierarchy



Hennessey and Patterson, Computer Architecture: A Quantitative Approach, 4th edition, 2007.

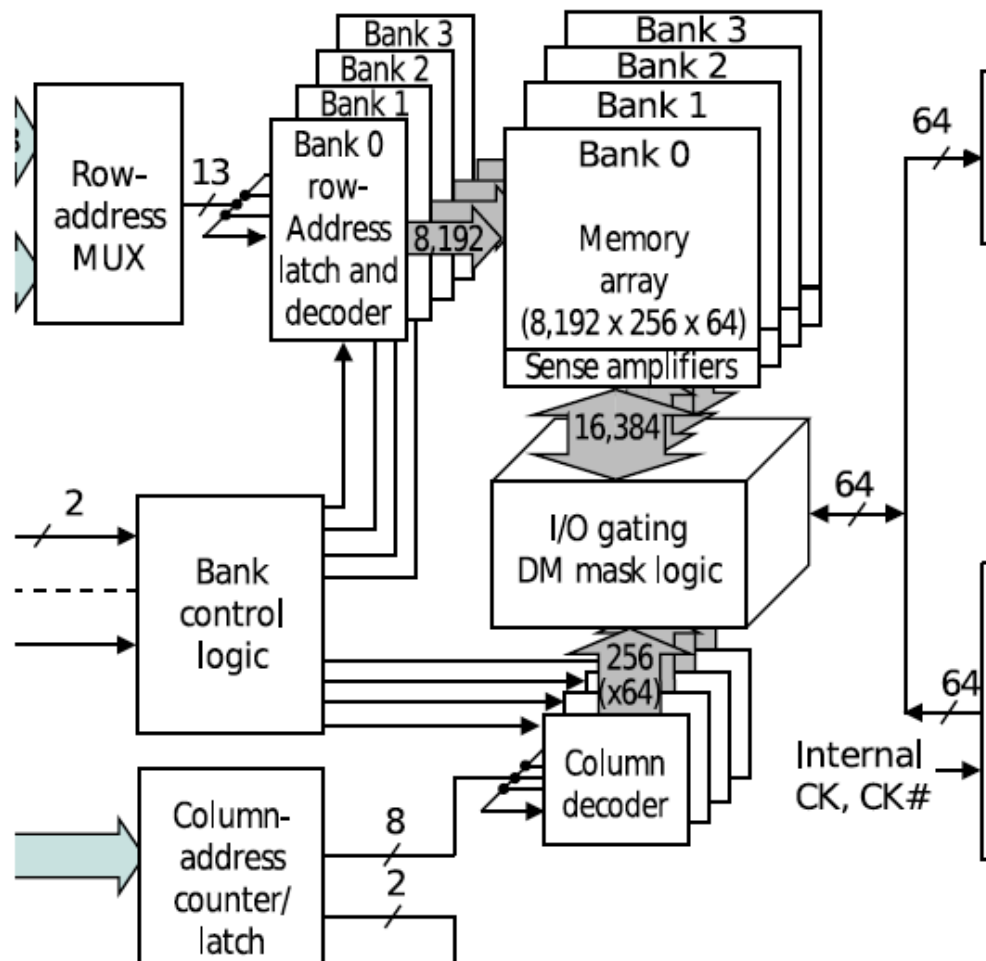
- Register file is a temporary memory under program control.
 - *Why is it so small?* *Instruction word size.*
 - Cache is a temporary memory under hardware control.
 - *Why is replacement strategy application independent?*
Separation of concerns.
- PRET principle: any temporary memory is under program control.**

PRET principle implies using a scratchpad rather than a cache.

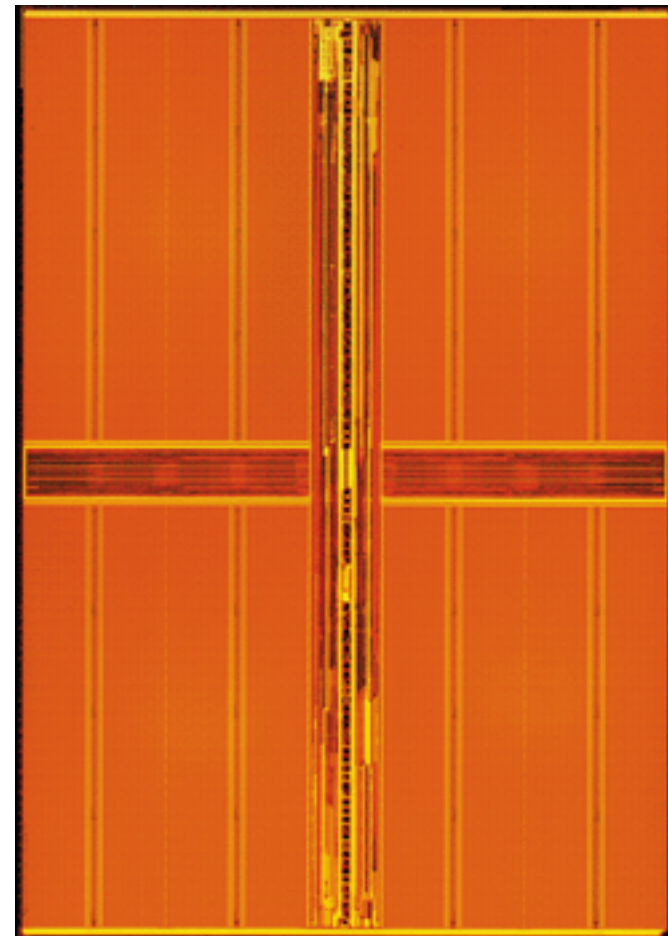


What about the main memory?

Opportunity: DRAM for main memory is highly parallel



Micron corp.



DDR2: Four pipelined banks

DDR3: Eight pipelined banks

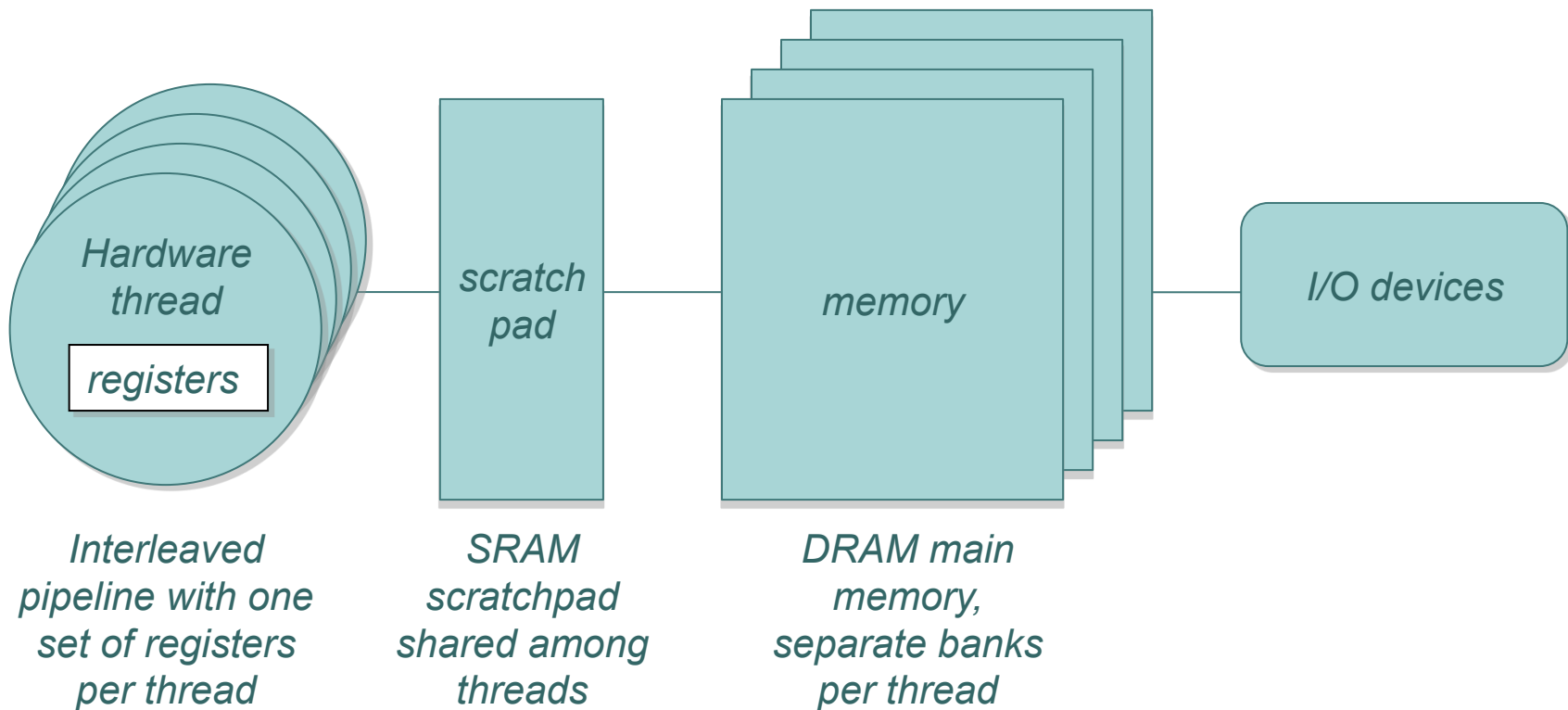
DDRn: 2^n pipelined banks?

Resulting PRET Architecture

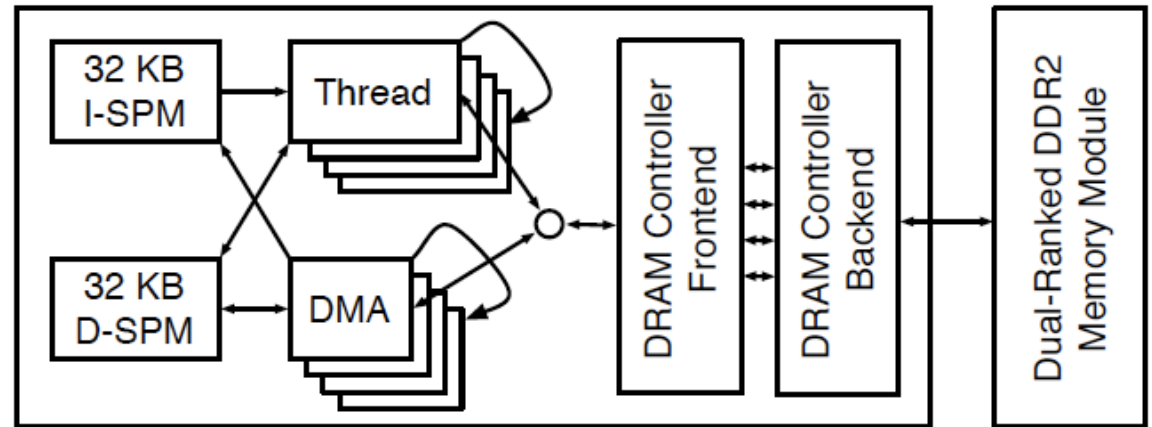
We have realized this in *PTArm*,
a soft core on a Xilinx Virtex 5 FPGA

*Note inverted memory
compared to multicore!*

*Fast, close memory is
shared, slow remote
memory is private!*



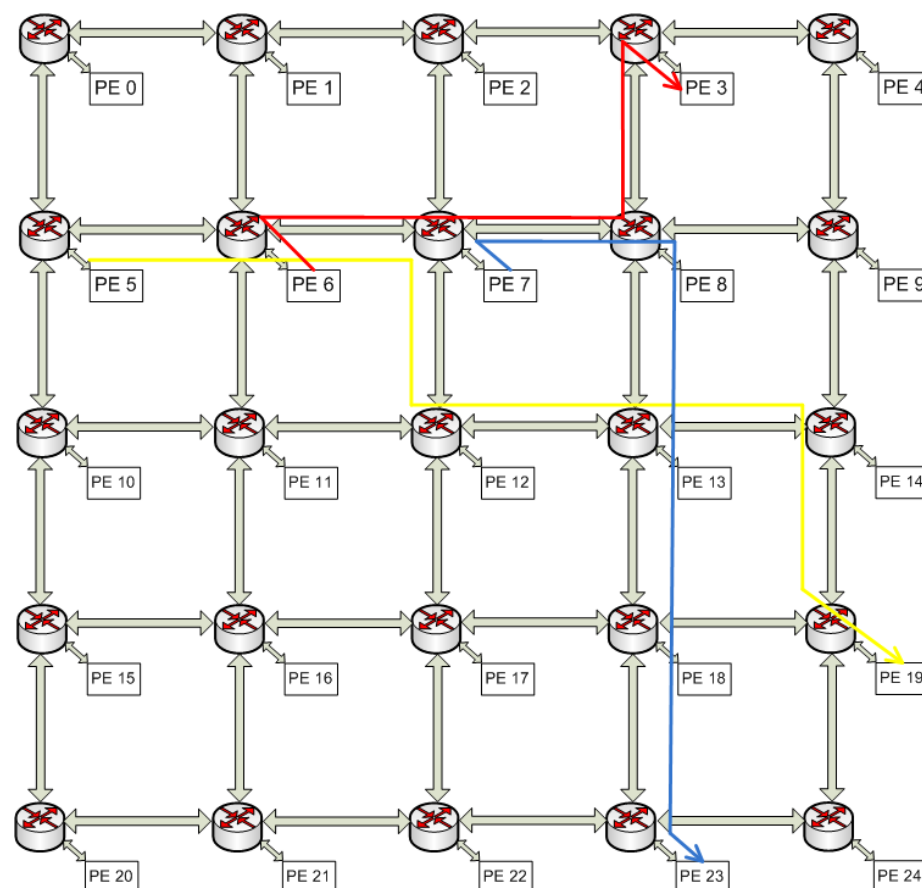
Memory Architecture in PTArm goes further



- Dual ported instruction/data scratchpads
- Load/stores can go to scratchpads or main memory
- DMA
 - One DMA unit per hardware thread
 - Thread can initiate DMA scratchpad-main transfers
 - Thread continues executing from scratchpad
 - Thread blocks on access to either DRAM or the affected region of the scratchpad until DMA is complete.
- DRAM refreshes are software controlled

Multicore PRET

In today's multicore architectures, one thread can disrupt the timing of another thread *even if they are running on different cores and are not communicating!*



Our preliminary work shows that control over timing enables conflict-free routing of messages in a network on chip, making it possible to have non-interfering programs on a multicore PRET.

Status of the PRET project

- MURI & NSF funding has run out.
 - Therefore, the project has successfully concluded.
- 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.

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

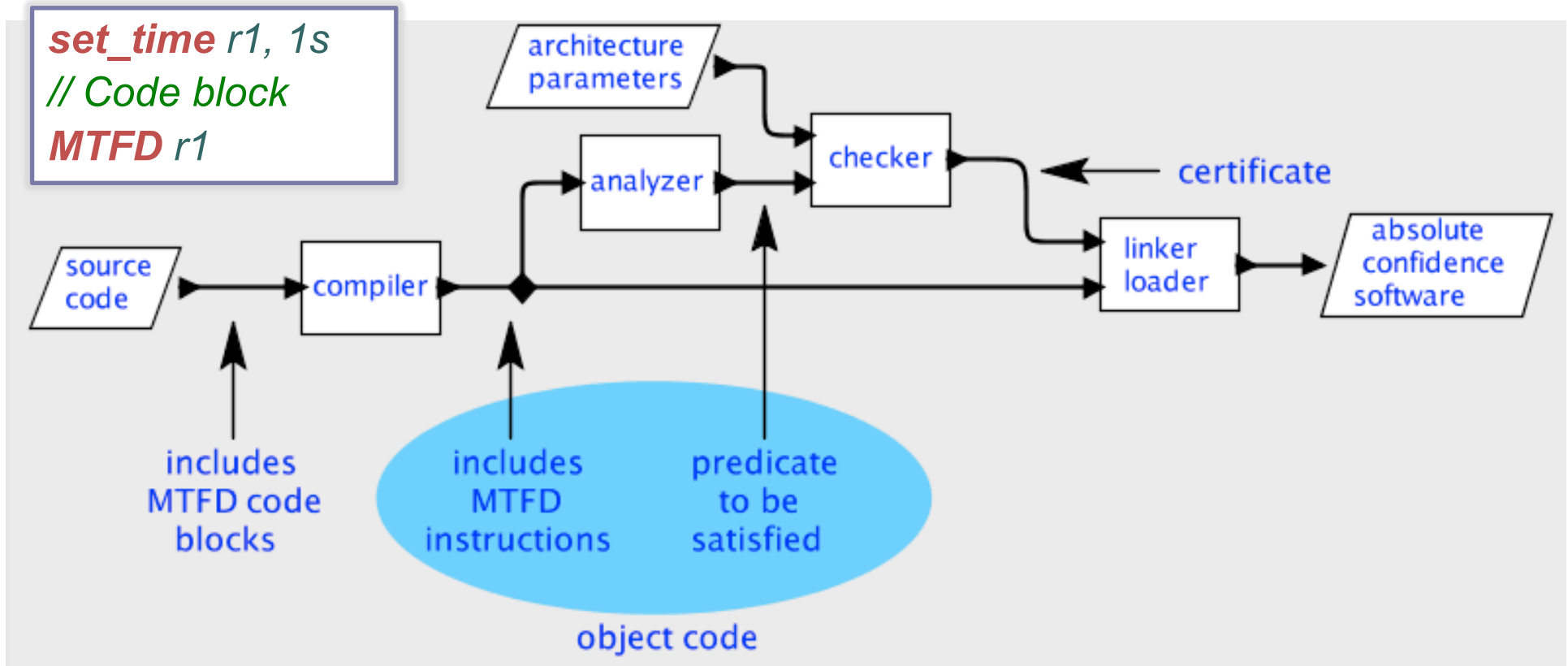
A Key Next Step: Parametric PRET Architectures

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

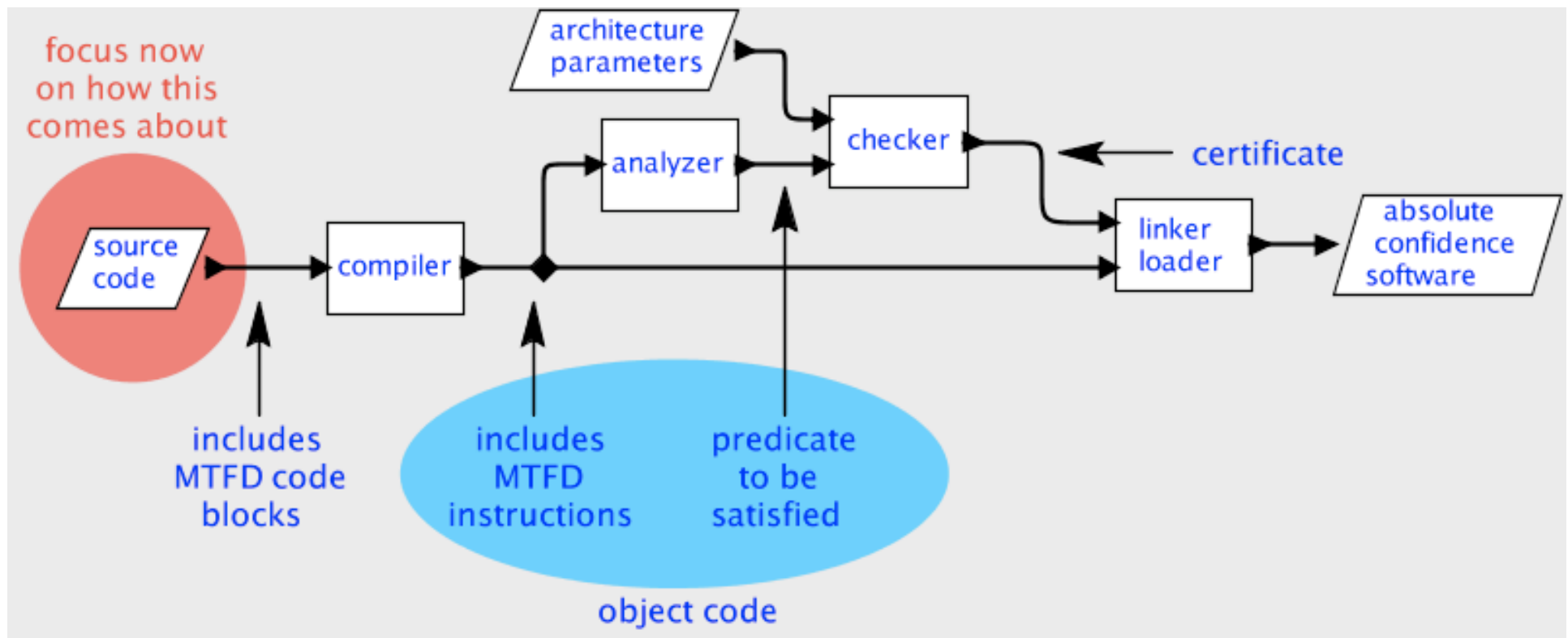


The goal is to make software that will run correctly on a variety of implementations of the ISA, and that correctness can be checked for each implementation.

PRET Publications

- Bui, E. A. Lee, I. Liu, H. D. Patel and J. Reineke, “**Temporal Isolation on Multiprocessing Architectures**,” DAC 2011.
- D. Bui, H. Patel, and E. Lee, “**Deploying hard real-time control software on chip-multiprocessors**,” RTCSA 2010.
- 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.
- 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 and E. A. Lee, “**The Case for the Precision Timed (PRET) Machine**,” in the *Wild and Crazy Ideas* Track of DAC, June 2007.

Part 2: How to get the Source Code?



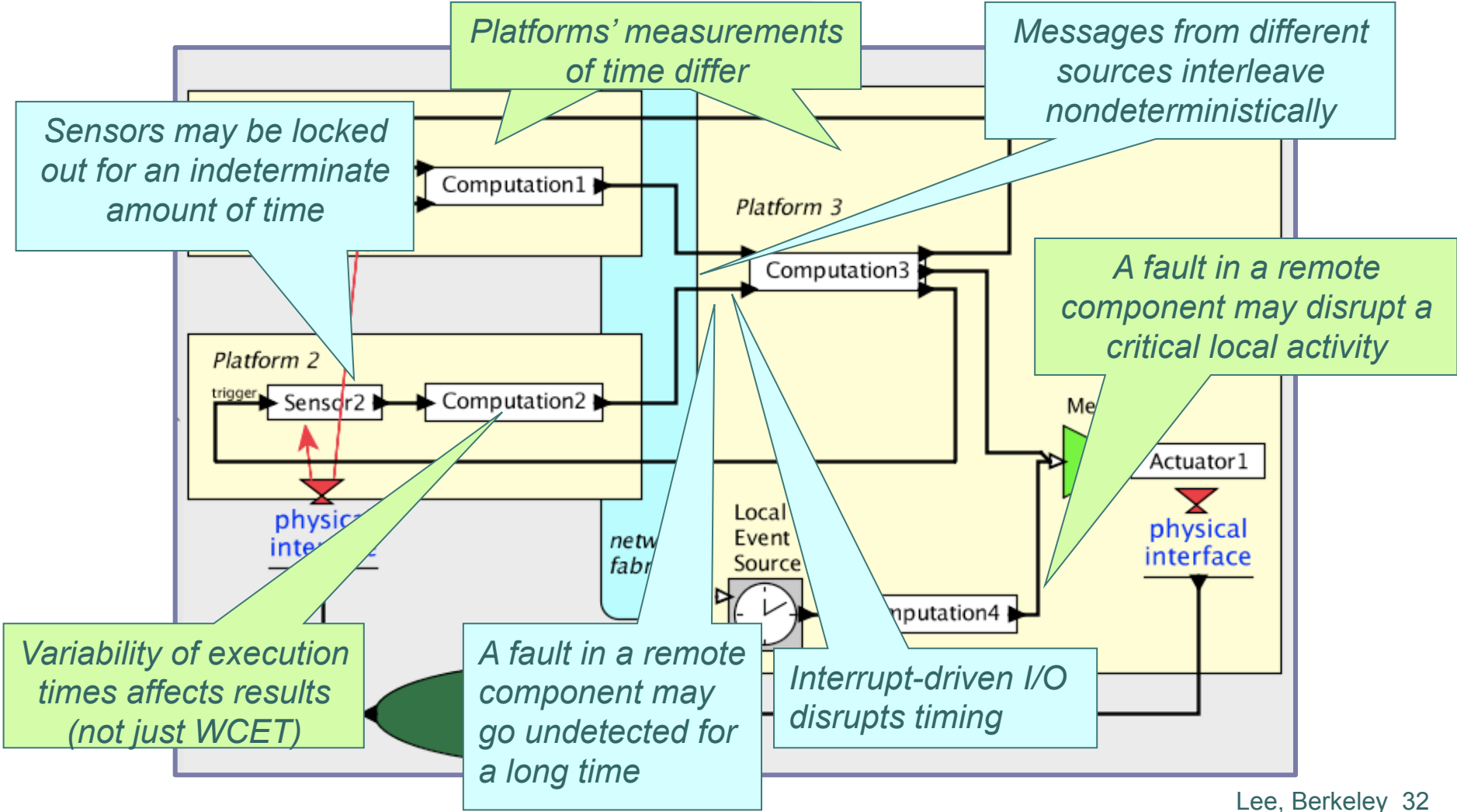
The input (mostly likely C) will ideally be generated from a model, like Simulink or SCADE. The model specifies temporal behavior at a higher level than code blocks, and it specifies a concurrency model that can limit preemption points.

However, Simulink and SCADE have naïve models of time.

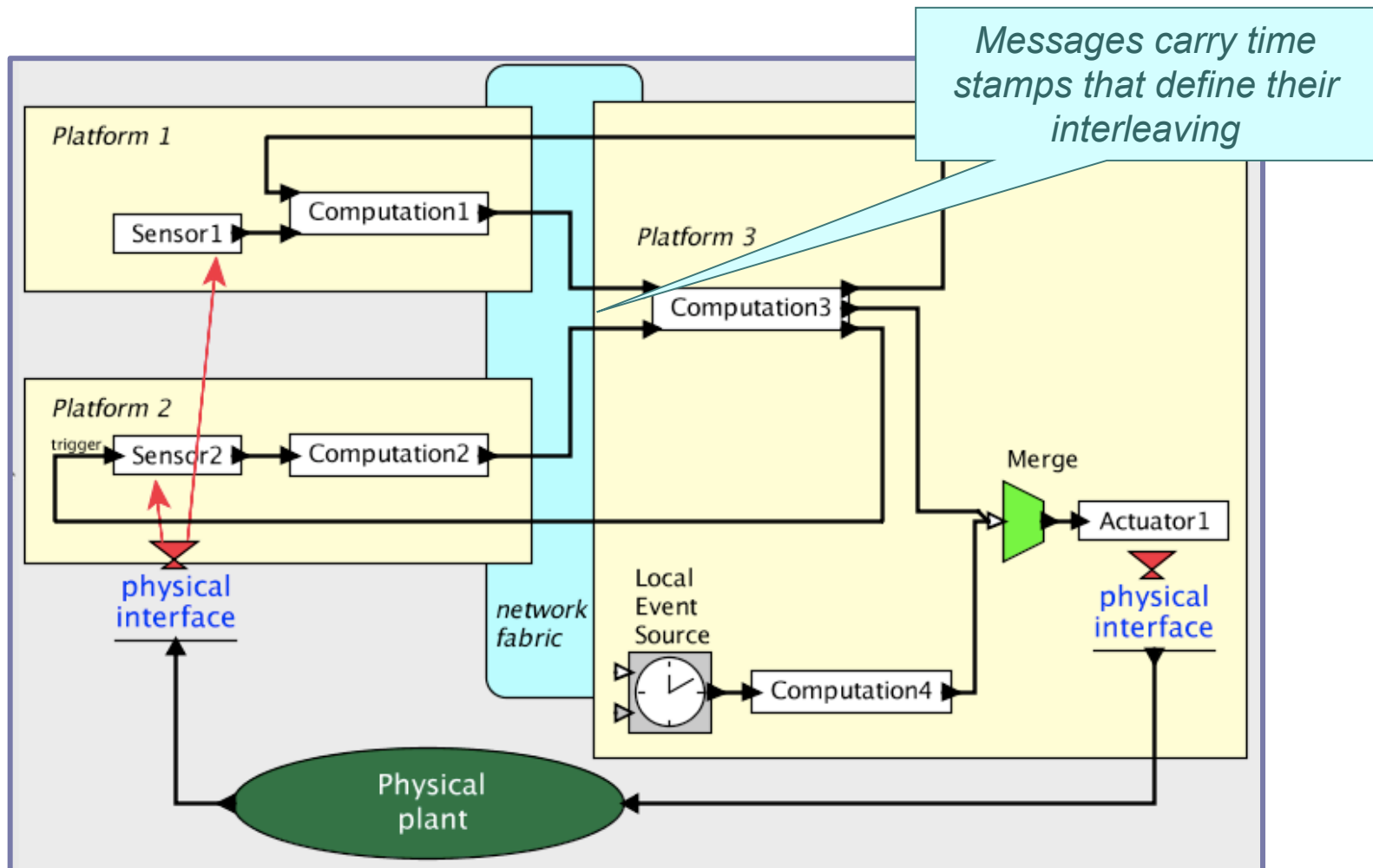
Etc...

Recall Structure of a Cyber-Physical System

Problems that complicate analysis of system behavior:

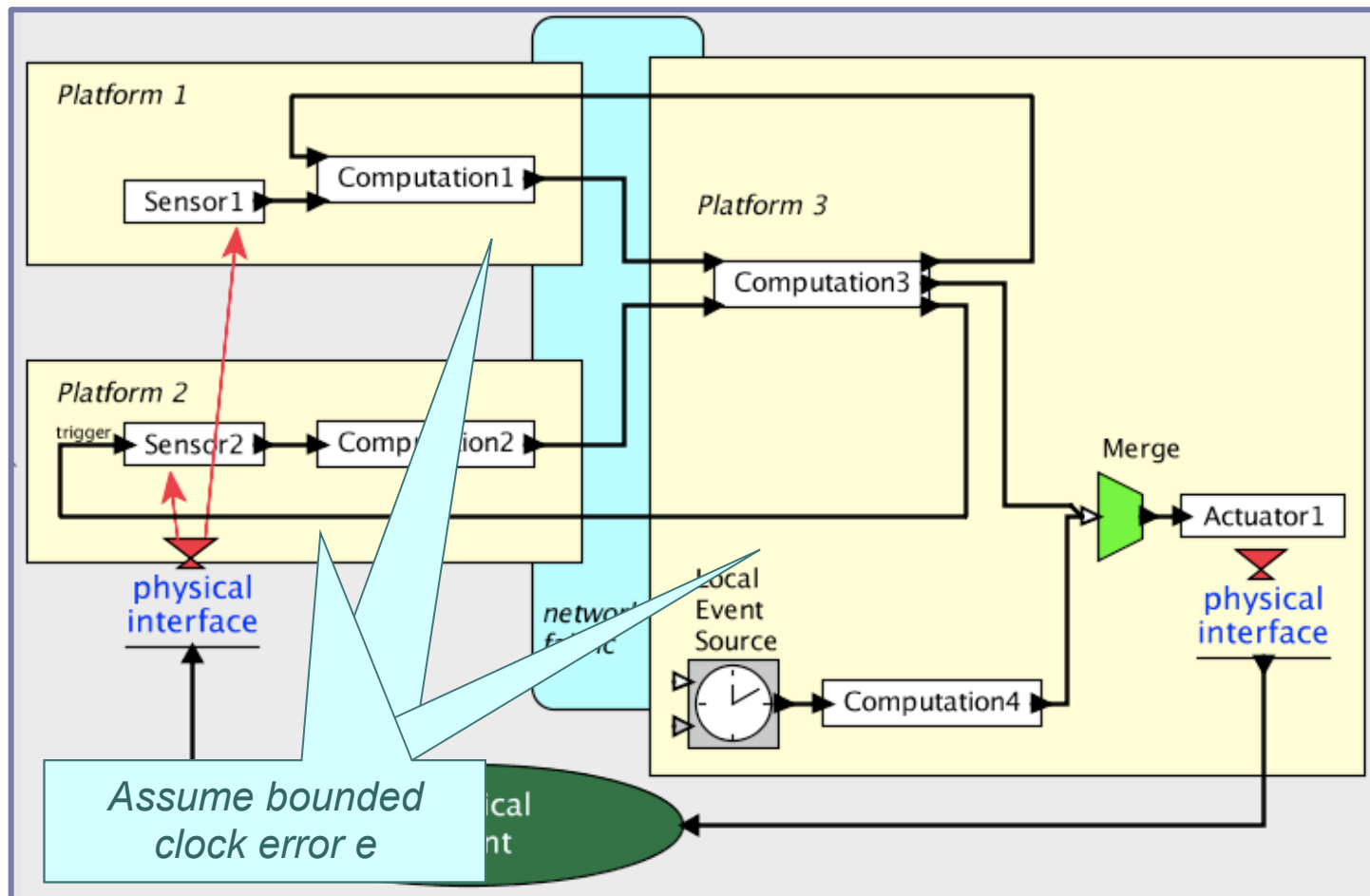


Ptides: First step:
Time-stamped messages.



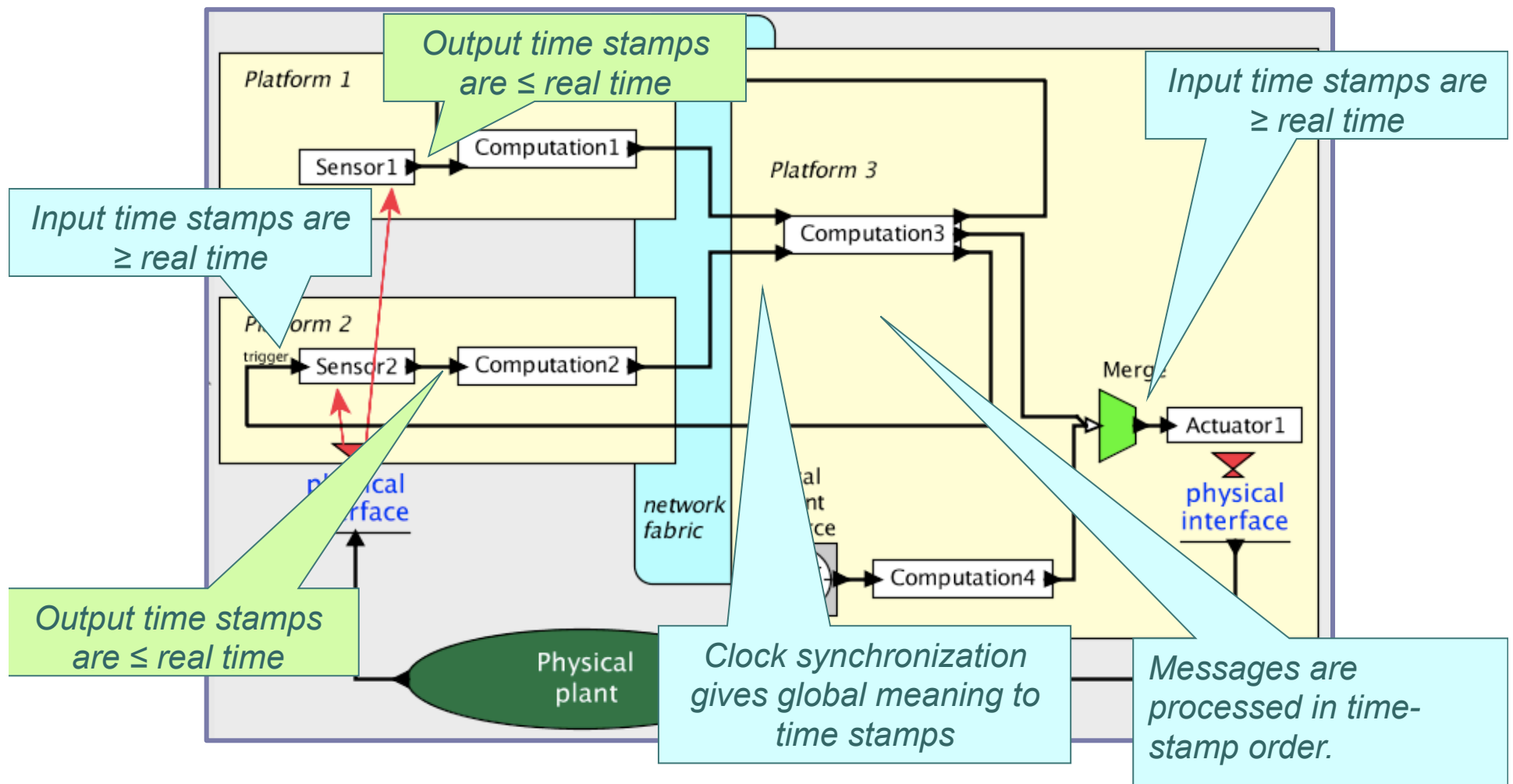
Ptides: Second step: Network time synchronization

GPS, NTP, IEEE 1588, time-triggered busses, etc., all provide some form of common time base. These are becoming fairly common.



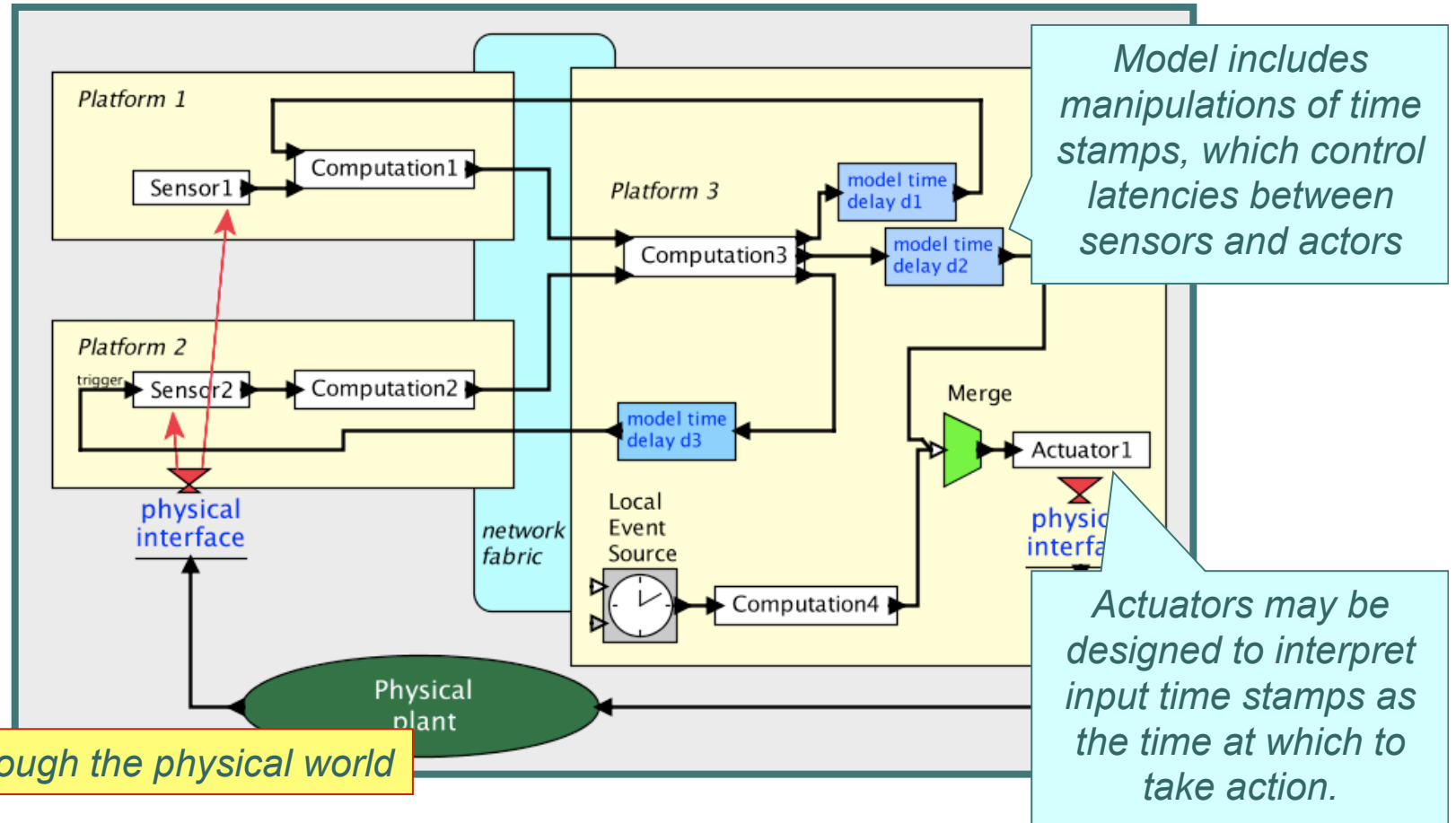
Ptides: Third step:

Bind time stamps to real time at sensors and actuators



Ptides: Fourth step: Specify latencies in the model

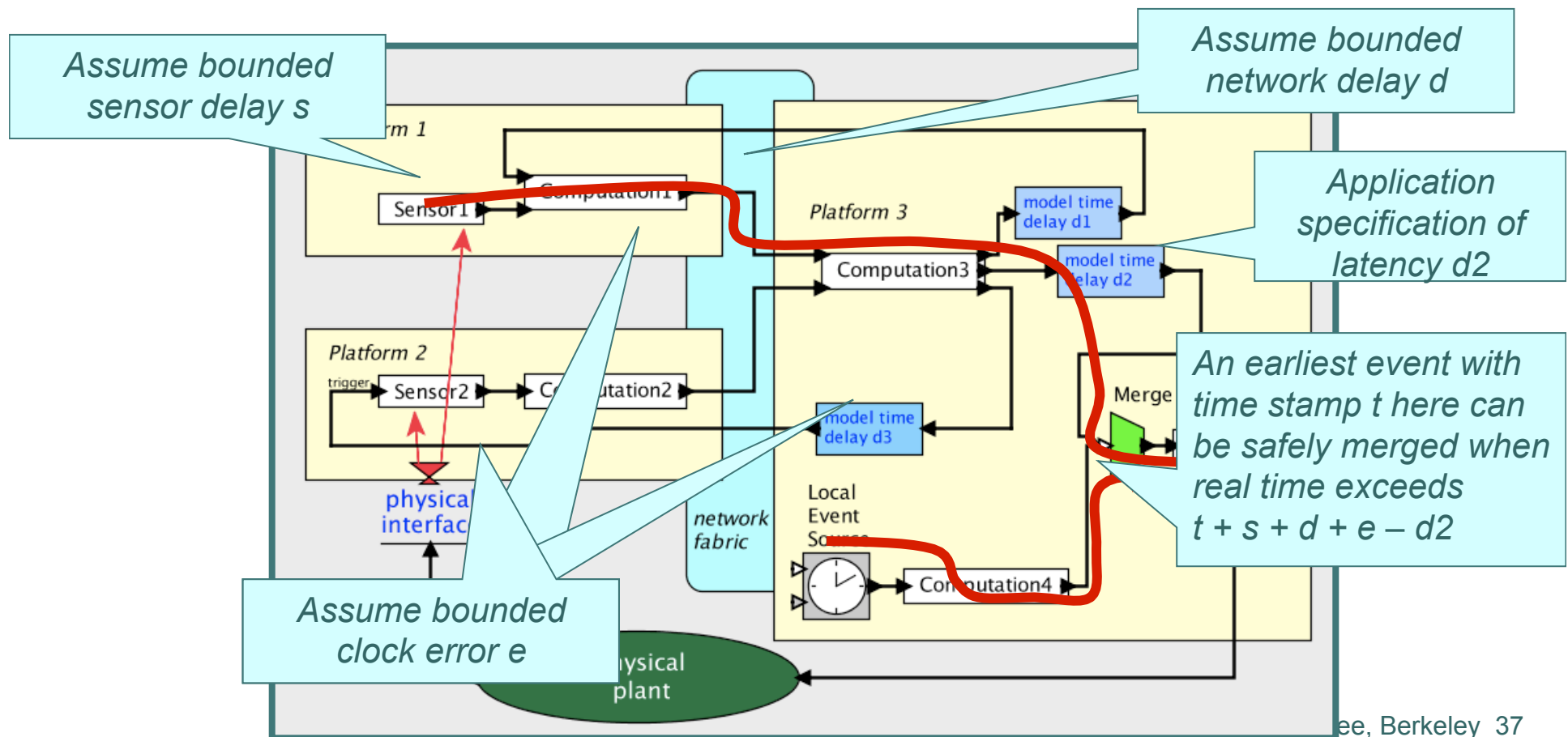
Global latencies between sensors and actuators become controllable, which enables analysis of system dynamics.



Ptides: Fifth step

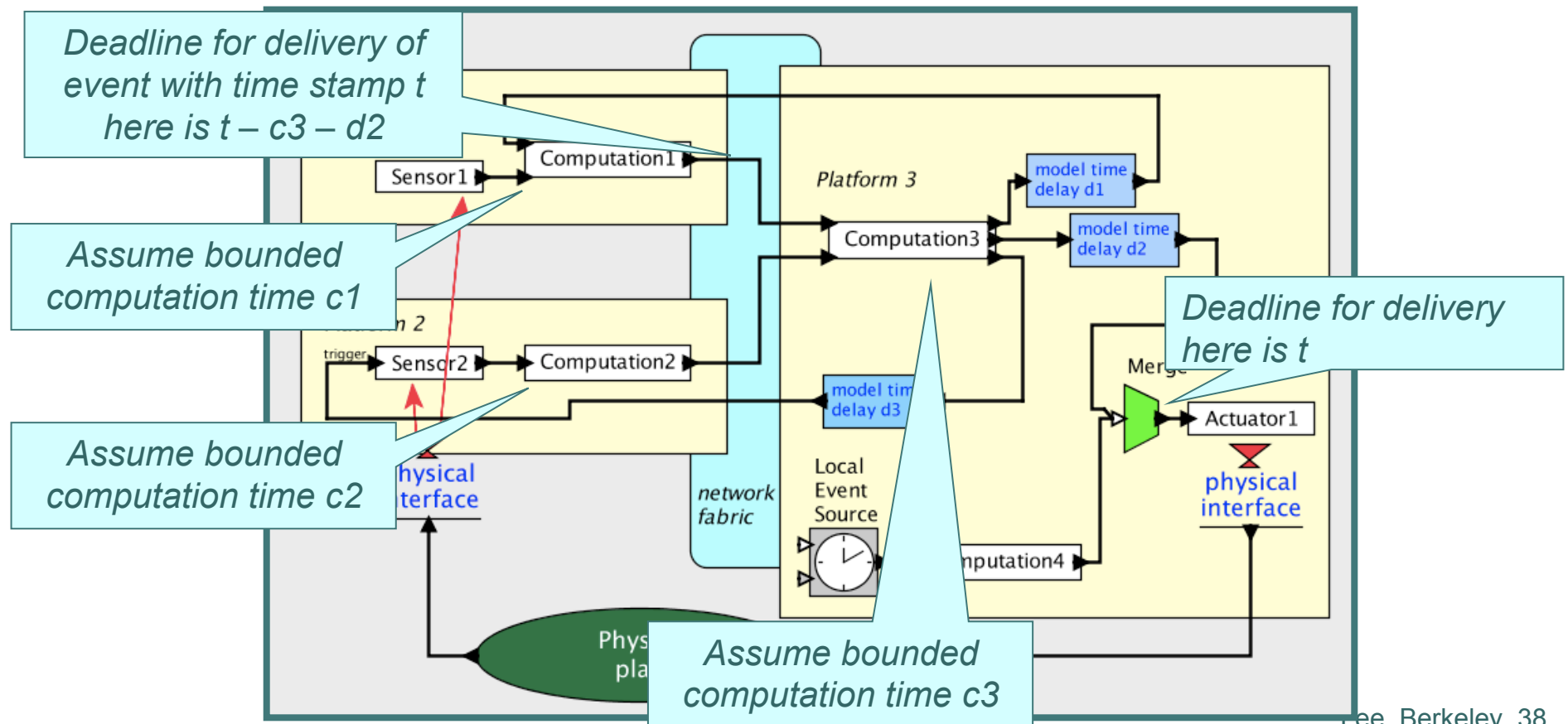
Safe-to-process analysis (ensures determinacy)

Safe-to-process analysis guarantees that the generated code obeys time-stamp semantics (events are processed in time-stamp order), given some assumptions.



Determine whether *deadlines* can be met

Schedulability analysis incorporates computation times to determine whether we can guarantee that deadlines are met.



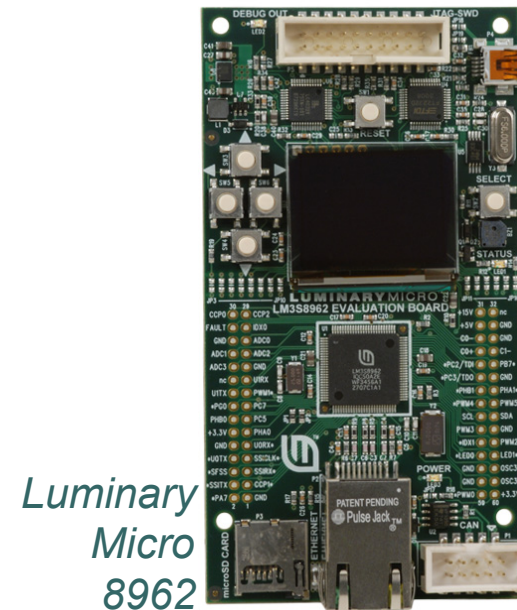
PtidyOS: A lightweight microkernel supporting Ptidies semantics

Current prototype runs on a COTS Arm platform (Luminary Micro) with rudimentary support for IEEE 1588 network time synchronization. Occupies about 16 kbytes of memory.

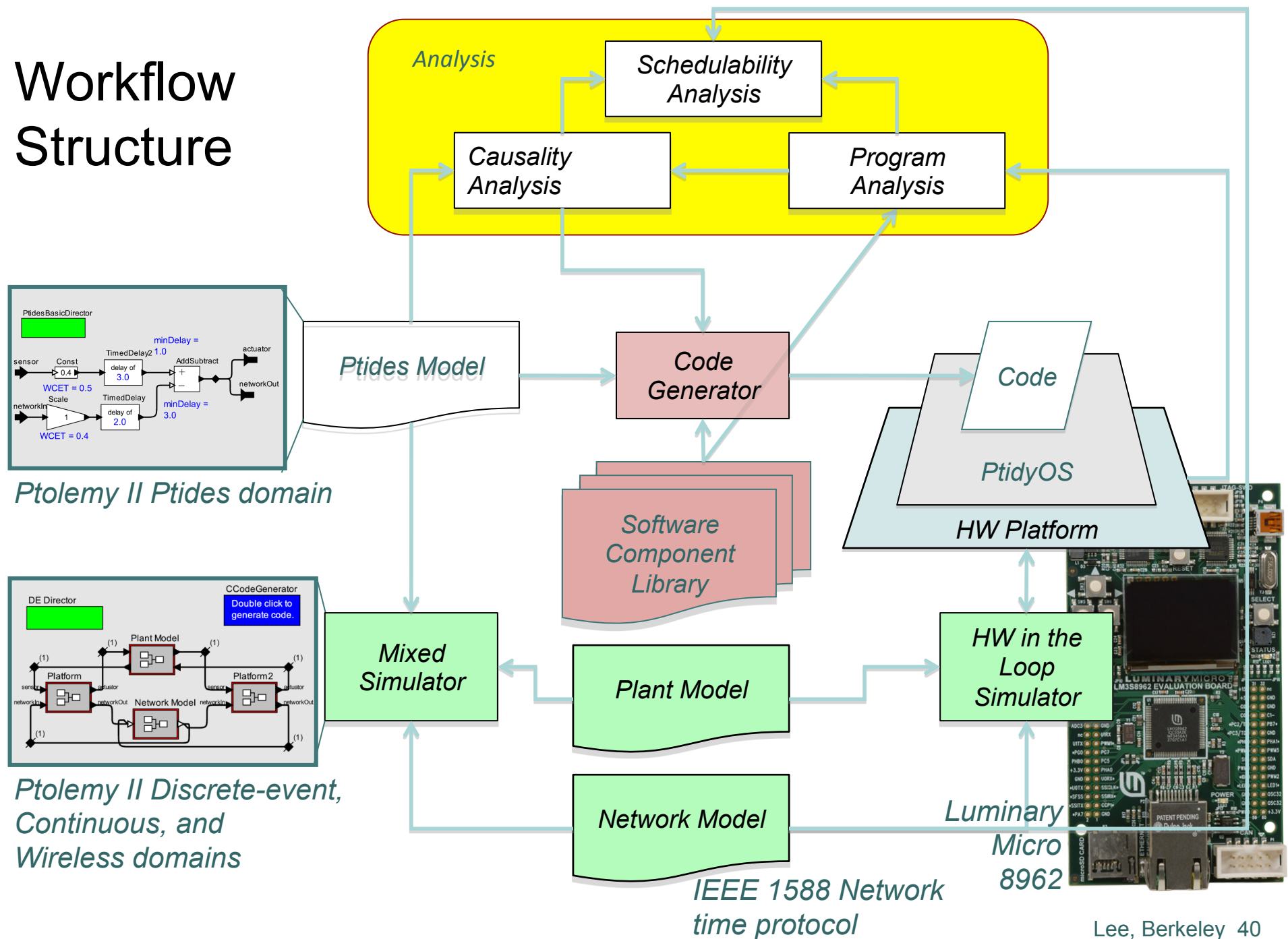
Currently porting to Renesas and PRET platforms.

An interesting property of PtidyOS is that despite being highly concurrent, preemptive, and EDF-based, it does not require threads.

A single stack is sufficient!

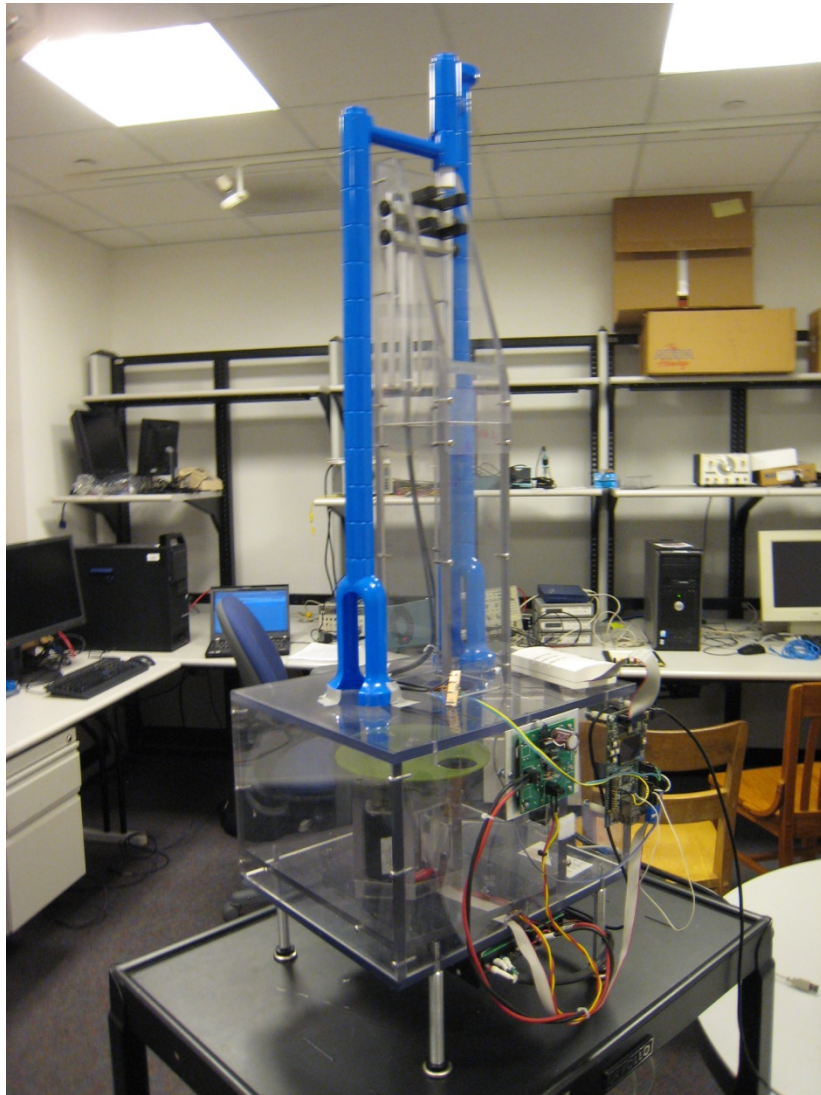


Workflow Structure



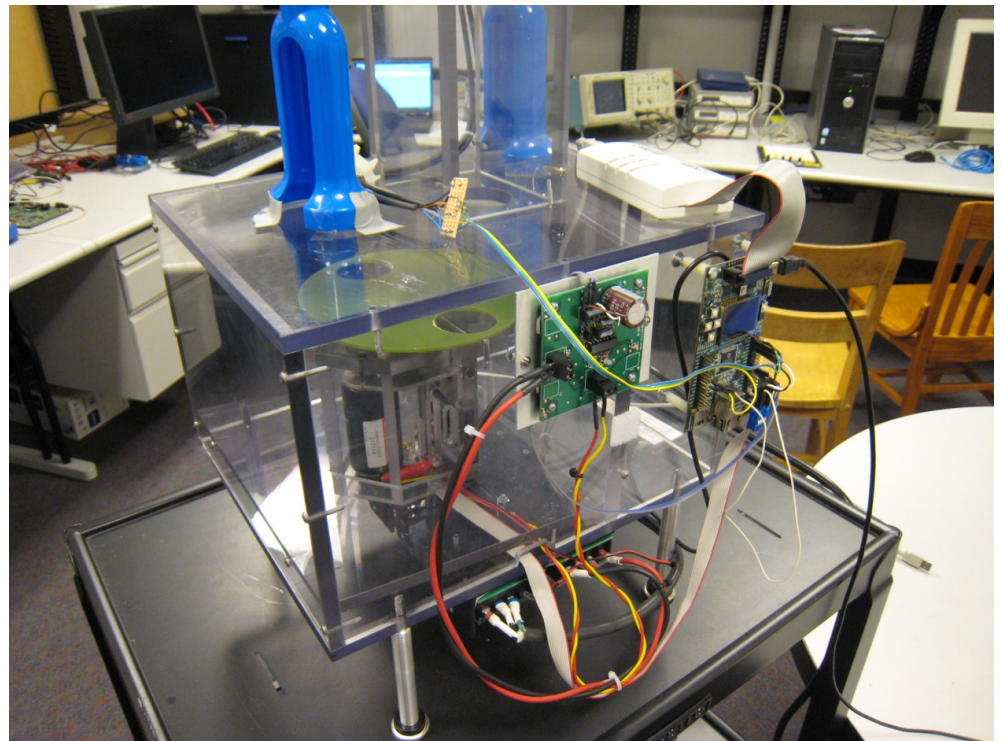
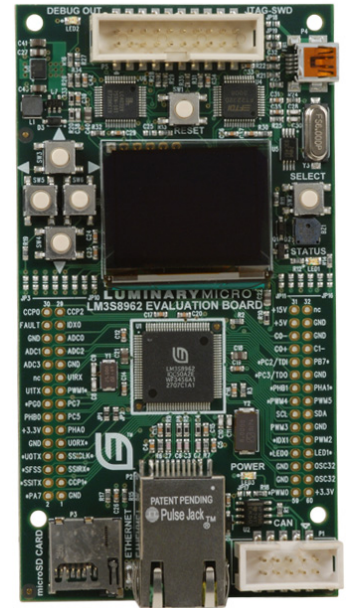
A Test Case for PtidyOS

This device, designed by Jeff Jensen, mixes periodic, quasi-periodic, and sporadic real-time events.

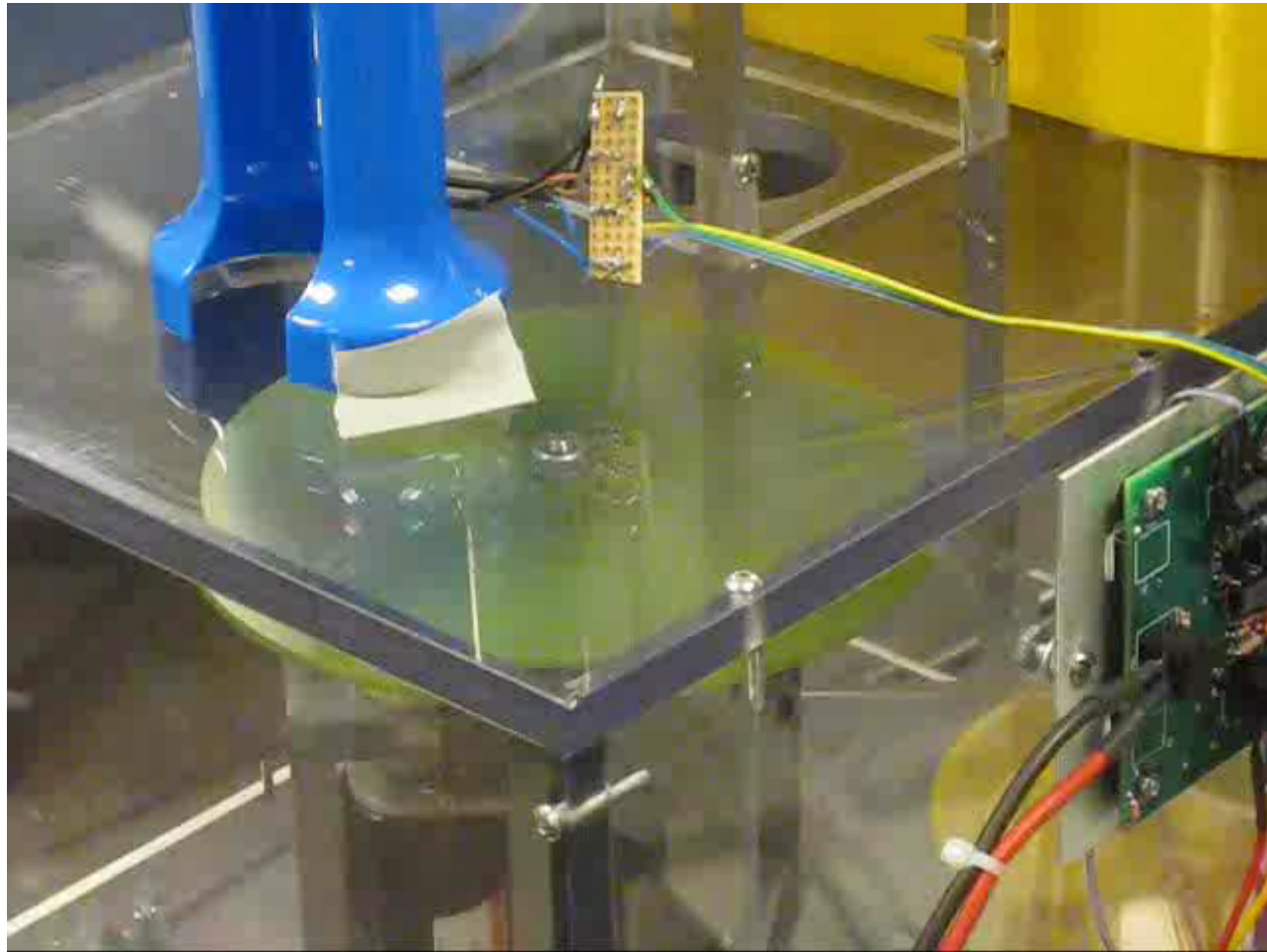


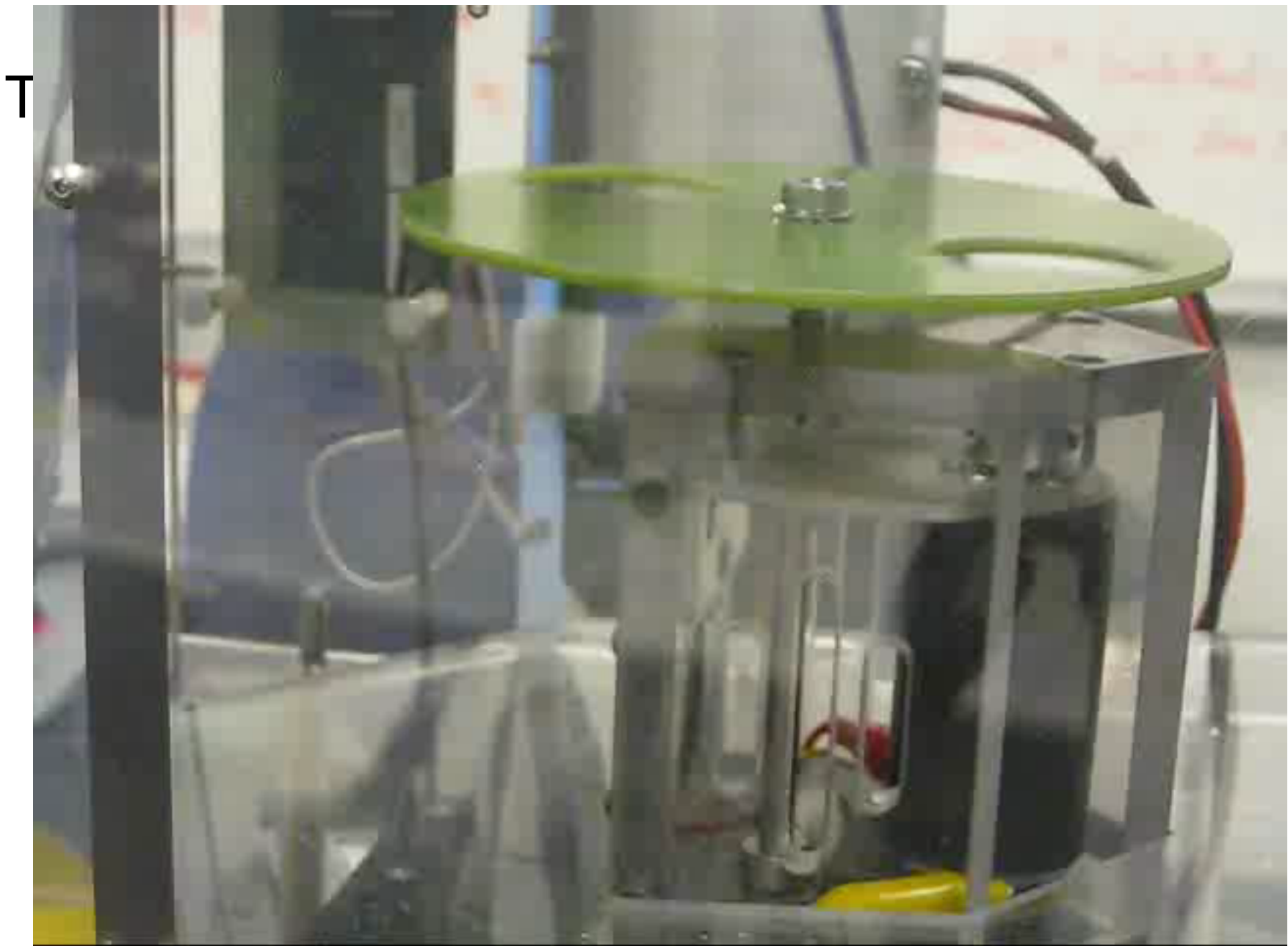
Tunneling Ball Device

- sense ball*
- track disk*
- adjust trajectory*



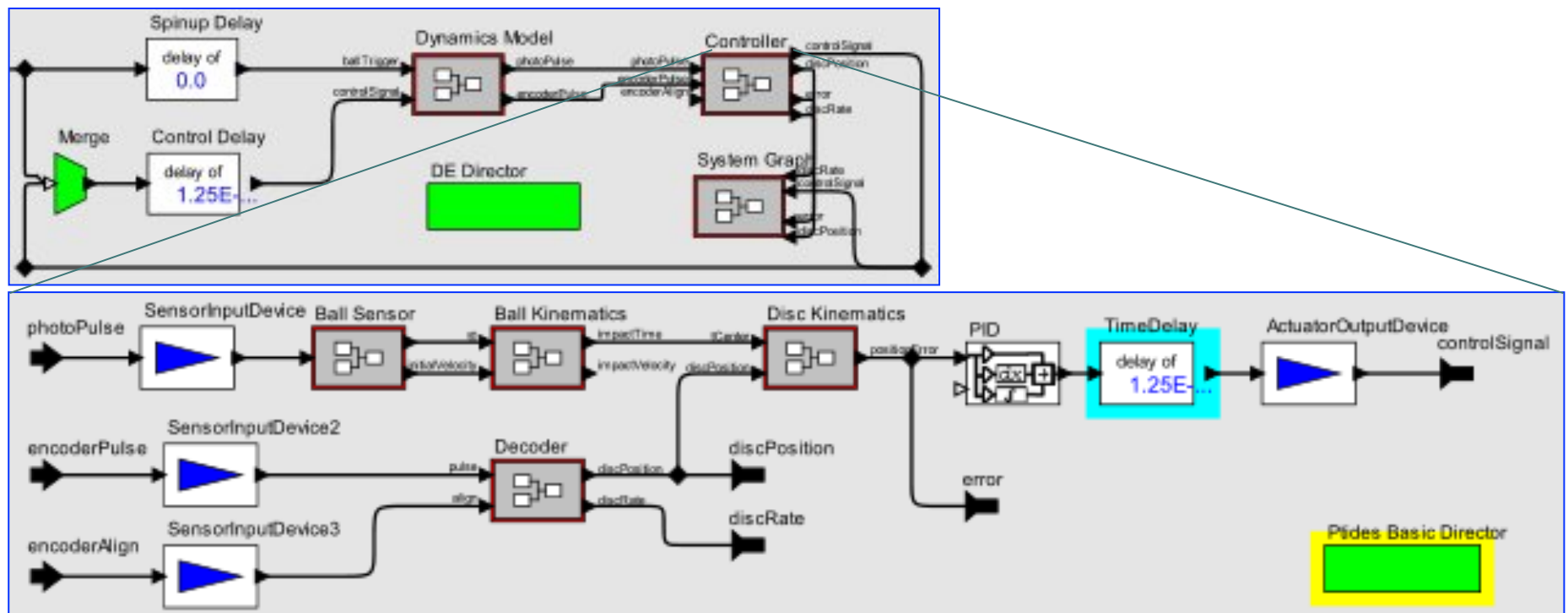
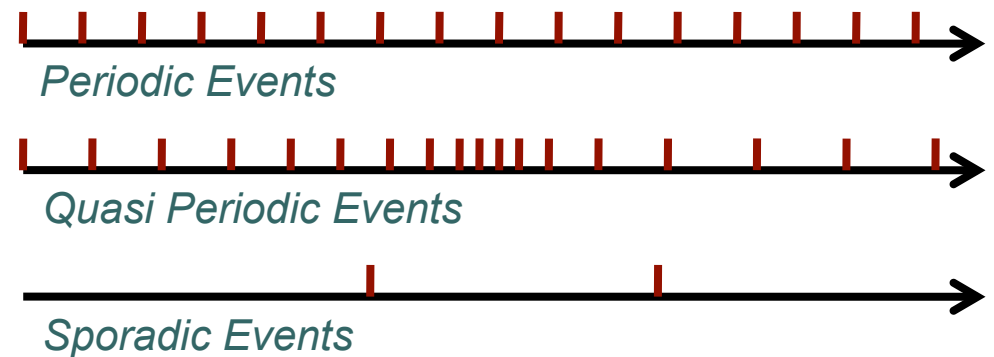
Tunneling Ball Device in Action





Tunneling Ball Device

Mixed event sequences



Ptides Project Status

- Seed funding from ARL got the project going.
- Ongoing NSF effort (CPS Medium)
 - Sanjit Seshia focused on WCET & schedulability analysis
 - Ptolemy II-based simulator supports multiform clocks
 - PtidyOS being prepped for open-source release

Ptides Publications

- 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.

Overview Reference:

E. A. Lee. *Computing needs time*. CACM, 52(5):70–79, 2009

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*



Distributed PTIDES Relies on Network Time Synchronization with Bounded Error

Press Release October 1, 2007



NEWS RELEASE

For More Information Contact

Media Contact

Naomi Mitchell
National Semiconductor
(408) 721-2142
naomi.mitchell@nsc.com

Reader Information

Design Support Group
(800) 272-9959
www.national.com

Industry's First Ethernet Transceiver with IEEE 1588 PTP Hardware Support from National Semiconductor Delivers Outstanding Clock Accuracy

Using DP83640, Designers May Choose Any Microcontroller, FPGA or ASIC to Achieve 8- Nanosecond Precision with Maximum System Flexibility



This may become routine!

With this PHY, clocks on a LAN agree on the current time of day to within 8ns, far more precise than older techniques like NTP.

A question we are addressing at Berkeley: How does this change how we develop distributed CPS software?

An Extreme Example: The Large Hadron Collider

The WhiteRabbit project at CERN is synchronizing the clocks of computers 10 km apart to within about 80 psec using a combination of IEEE 1588 PTP and synchronous ethernet.

