Compositional Timing in Concurrent, Parallel, and Distributed Real-Time Systems

Edward A. Lee

Robert S. Pepper Distinguished Professor
UC Berkeley

Invited Keynote Talk

3rd Workshop on
Compositional Theory and Technology for
Real-Time Embedded Systems

November 30, 2010, San Diego, CA, USA
(co-located with RTSS 2010)

Thanks to:

- *Danica Chang*
- *David Culler*
- *Gage Eads*
- *Stephen Edwards*
- *John Eidson*
- *Jeff Jensen*
- *Sungjun Kim*
- *Isaac Liu*
- *Slobodan Matic*
- *Hiren Patel*
- *Jan Reineke*
- *Alberto Sangiovanni-Vincentelli*
- *Jia Zou*

Abstract

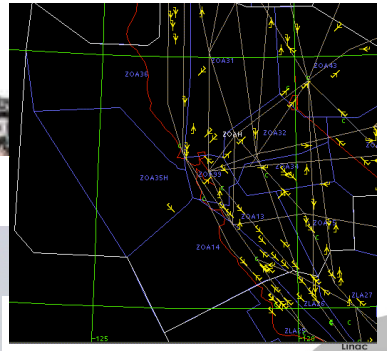
Edward A. Lee
Professor, UC Berkeley

In general-purpose computing, concurrency is achieved by relatively coarse-grained time multiplexing of shared resources. In this talk, I explore alternatives that provide temporal isolation between concurrent tasks on a processor, enabling the sharing of resources with repeatable timing behavior. The approach builds on the concept of PREcision Timed (PRET) machines, which introduce temporal semantics into an instruction-set architecture. I then explore how repeatable timing can facilitate efficient parallel execution in suitably-designed multicore architectures and in distributed real-time systems.

Cyber-Physical Systems (CPS):

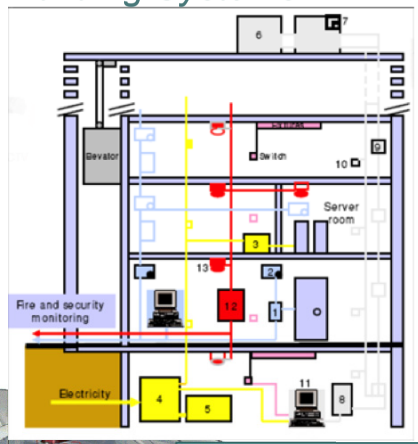
Orchestrating networked computational resources with physical systems

Avionics



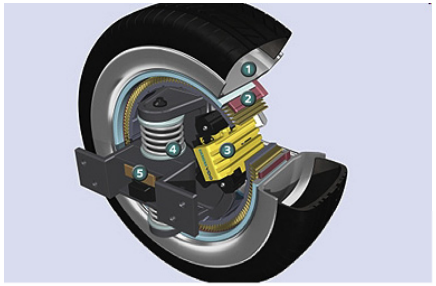
Transportation (Air traffic control at SFO)

Building Systems

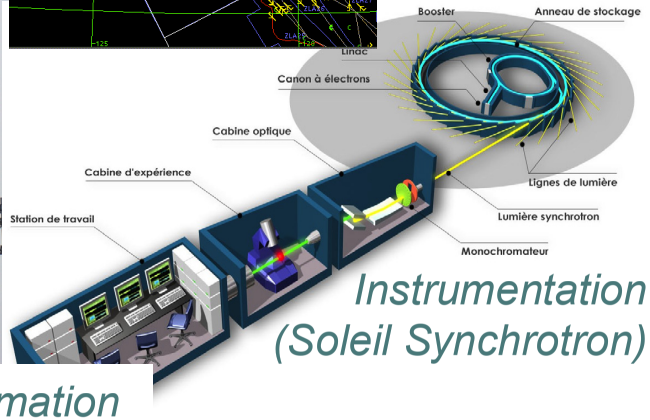


Telecommunications

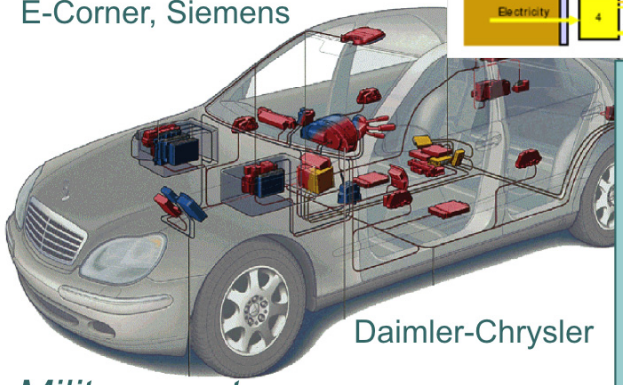
Automotive



E-Corner, Siemens

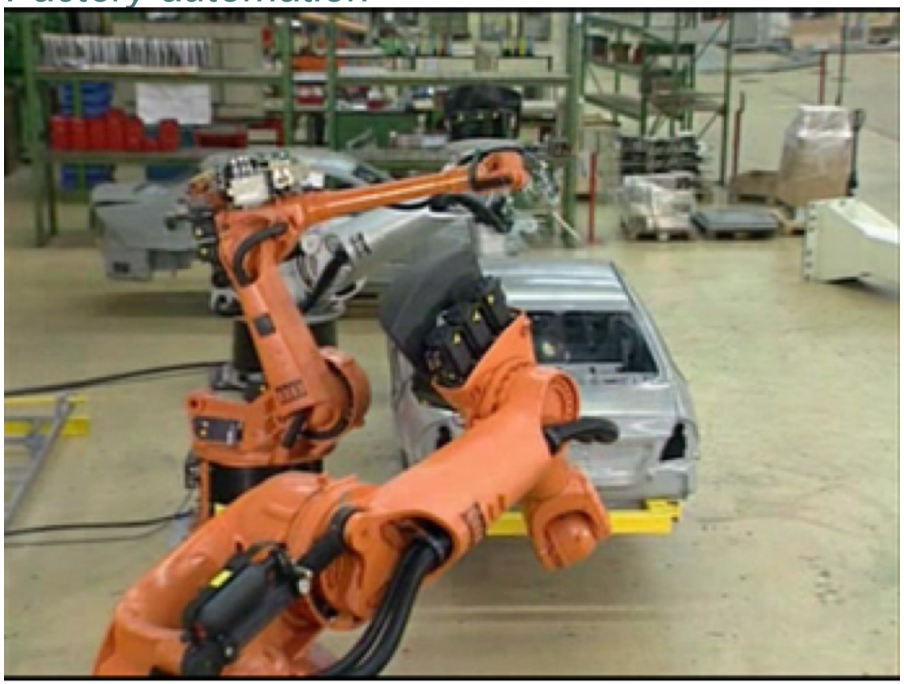


Instrumentation (Soleil Synchrotron)



Daimler-Chrysler

Factory automation

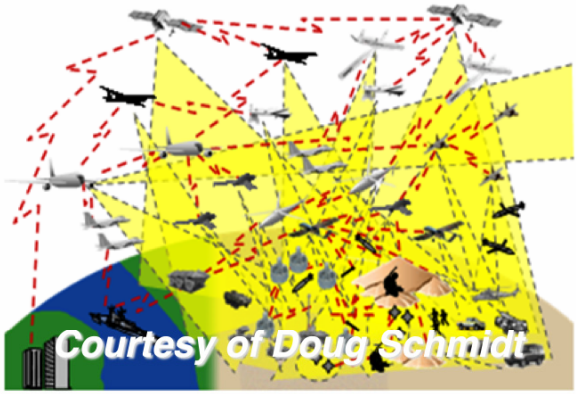


Courtesy of Kuka Robotics Corp.

Power generation and distribution

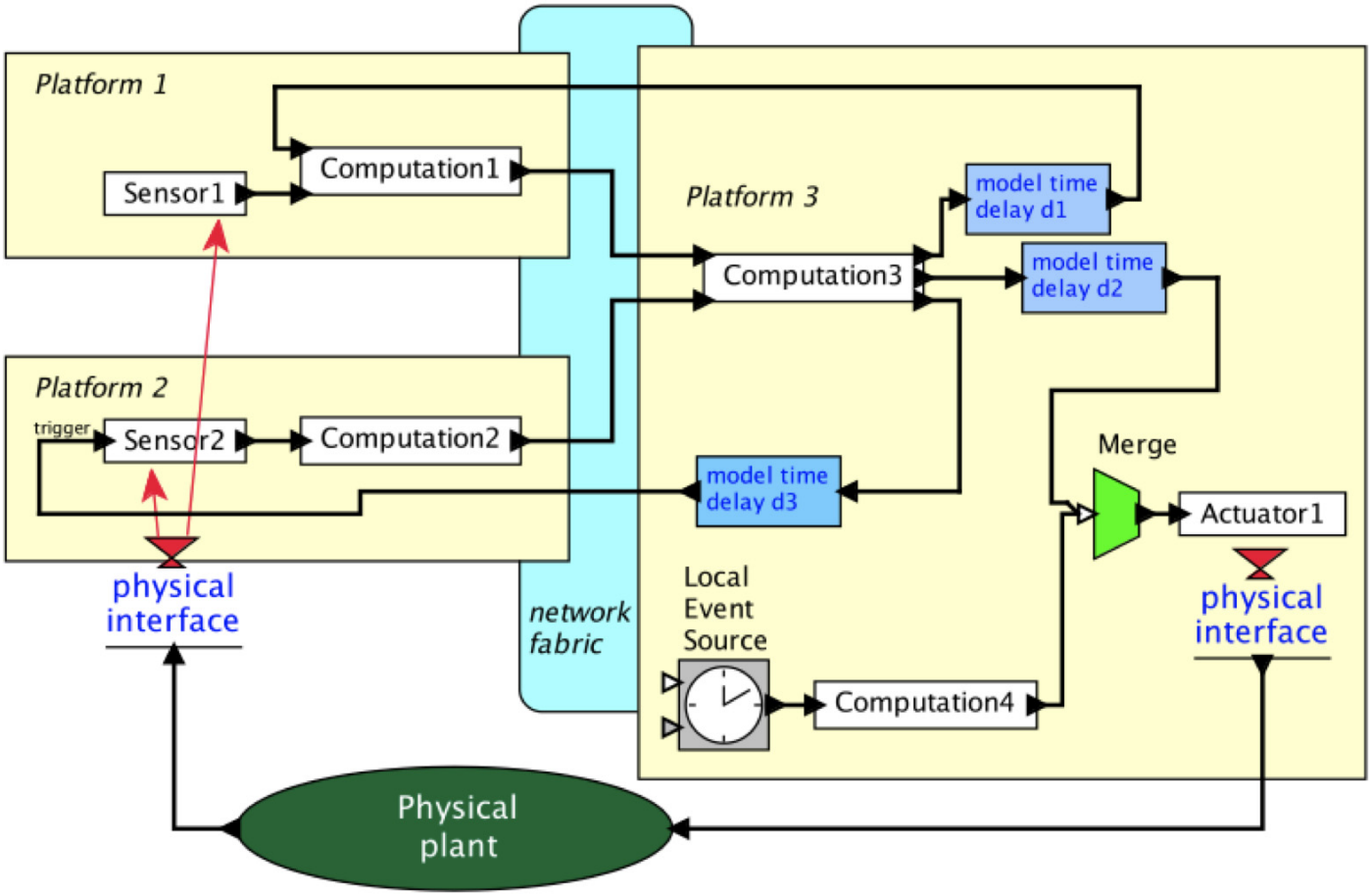
Courtesy of General Electric

Military systems:



Approaching the CPS Challenge

Physicalizing the cyber (PtC): to endow software and network components with abstractions and interfaces that represent their physical properties, such as dynamics in time.



Cyberizing the Physical (CtP): to endow physical subsystems with cyber-like abstractions and interfaces

The Problem We Address

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

Today, correct execution of a program has nothing to do with how long it takes to do anything.

We are correcting this by introducing precise and repeatable timing into core computing infrastructure.

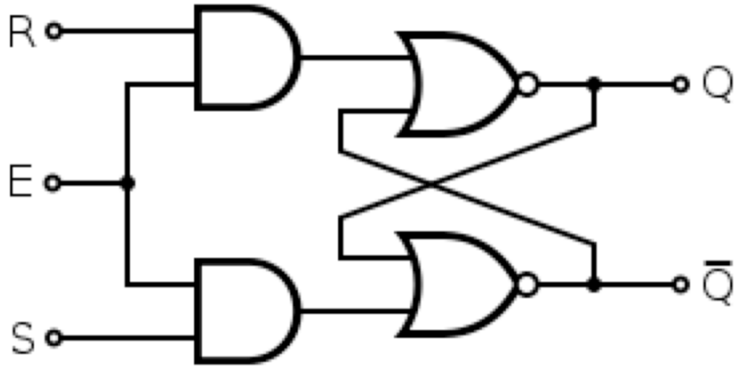
- Timing becomes a property of a *program*, not just a property of particular hardware executing the program.
- Timing of programs becomes predictable, repeatable, and portable.
- Behavior of *systems* becomes predictable, repeatable, and portable.

Problems that Arise Today

- Execution time analysis requires very detailed models of processors.
- Execution time analysis is intractable for many real programs and processors.
- Multicore interactions disrupt timing.
- Network interactions disrupt timing.Pipeline stalls and speculation make timing of programs hard to predict and hard to control.
- Memory hierarchy (caches) introduce timing variability and timing interference across tasks.

Consequence: System behavior is hard to predict and control.

These problems are not intrinsic to the technology!



Electronics technology delivers highly reliable and precise timing…

20.000 MHz (± 100 ppm)

… and the overlaying software abstractions discard it.

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

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

PRET Machines: Make Timing a Semantic Property of Computers

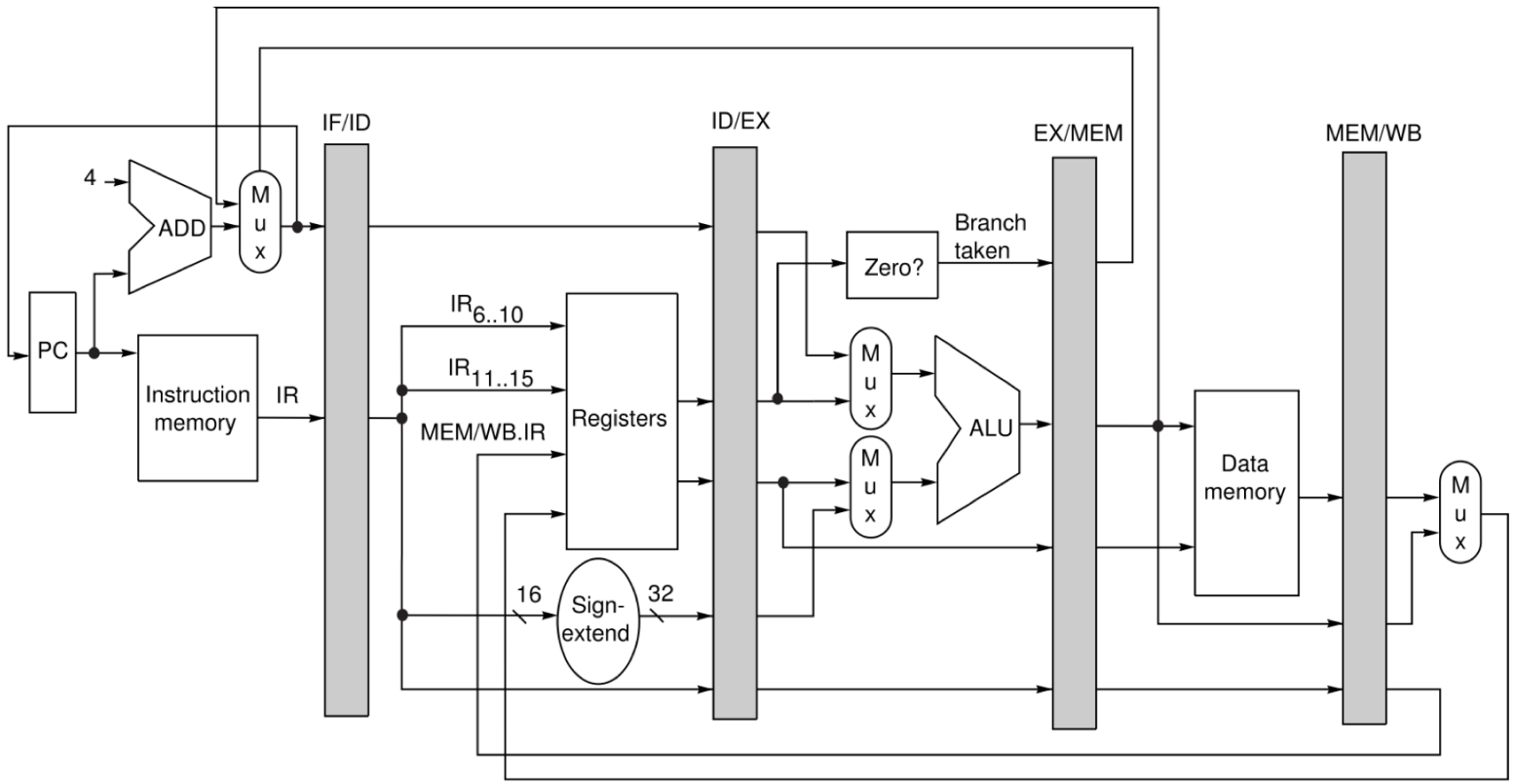
Make temporal behavior as important as logical function.

Timing precision with performance: Challenges:

- ISAs with timing (repeatable instr. timing? deadline instructions?)
- Deep pipelines (interleaving?)
- Memory hierarchy (scratchpads? DRAM banks?)
- Predictable memory management (Metronome?)
- Languages with timing (discrete events? Giotto?)
- Predictable concurrency (synchronous languages?)
- Composable timed components (actor-oriented?)
- Multicore PRET (conflict-free routing?)
- Precision networks (TTA? Time synchronization?)

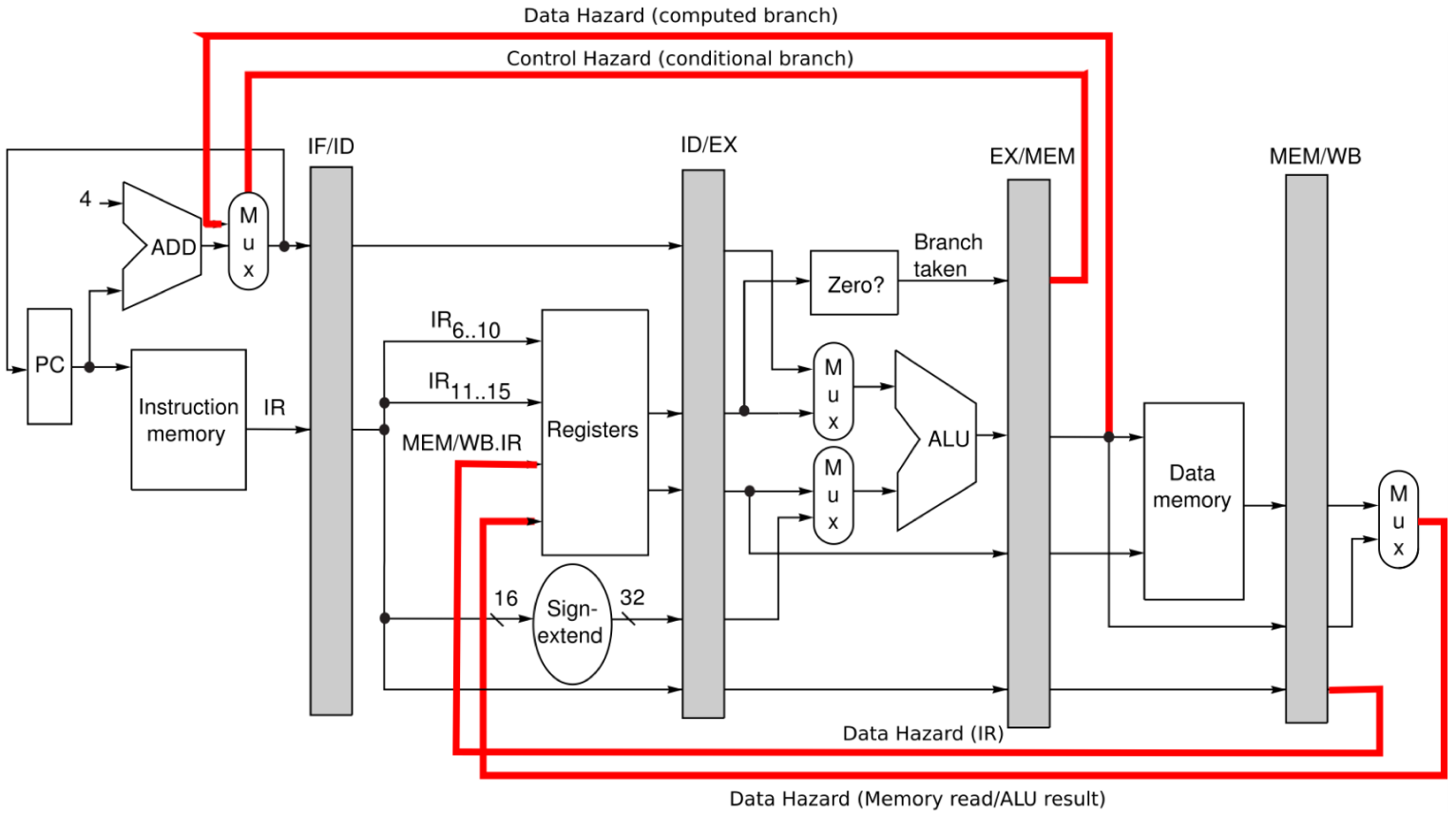
Edwards and Lee, "**The Case for the Precision Timed (PRET) Machine**," *Wild and Crazy Ideas* Track, *Design Automation Conference* (DAC), June 2007.

Pipelining



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

Pipeline Hazards

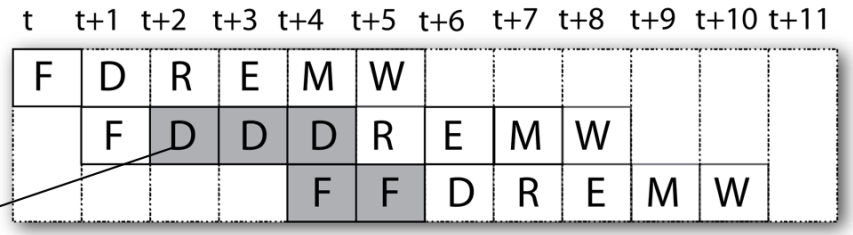


Hennessey and Patterson, Computer Architecture: A Quantitative Approach, 4th edition, 2007.

An Alternative: Pipeline Interleaving

Traditional pipeline:

T0: cmp %g2, 9
T0: bg, a 40011b8
T0: add %i1, %i2, %l3

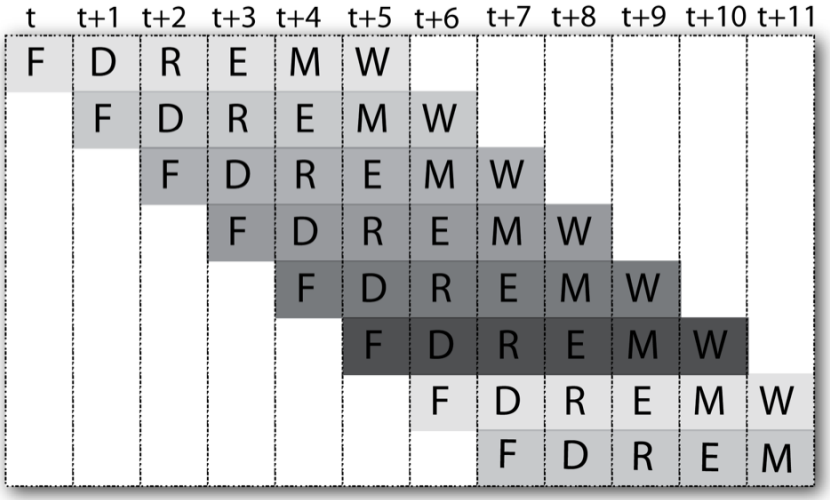


Stall pipeline

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

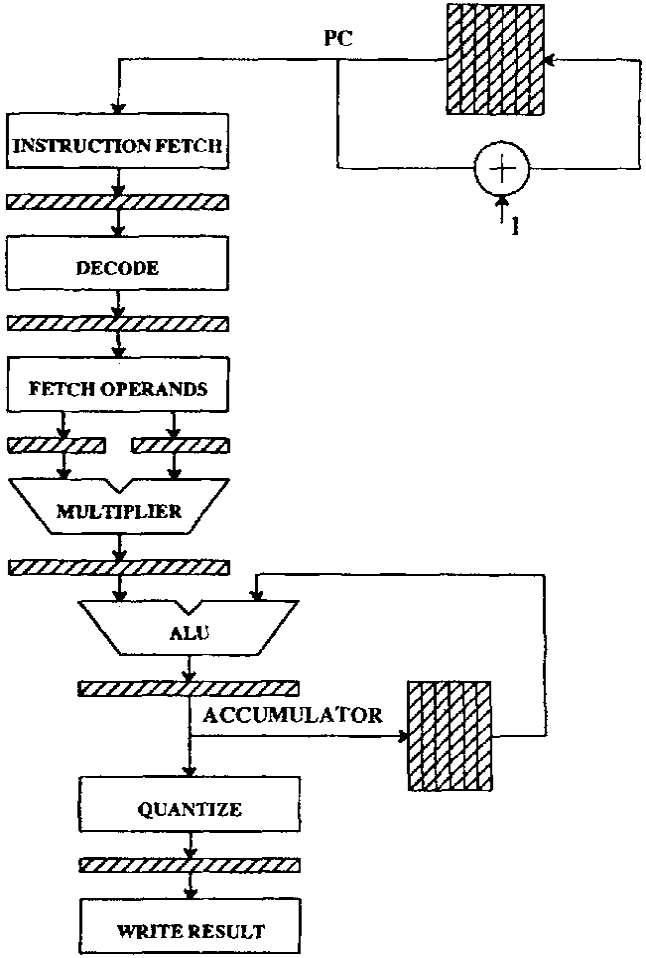


Repeatable timing behavior of instructions

Pipeline Interleaving

An old idea:

- 1960s:
  - CDC 6600
  - Denelcore HEP
- ...
- 2000s
  - Sandbridge Sandblaster (John Glossner, et al.)
  - XMOS (David May, et al.)



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

*There are various detractors. See Ungerer, T., B. Robic and J. Silc (2003). "A survey of processors with explicit multithreading." Computing Surveys **35(1): 29-63.***

Pipeline Interleaving Enables Precise, Regular Timing, and Compositional Timing

- If interrupts are always handled by thread 1, then the timing of threads 2 through *n* is not affected by I/O.

- XMOS approach: Activate thread *n*+1 to handle interrupts. This does affect the timing of other threads, but much more smoothly than standard methods.

Our solution:
PRET Machines

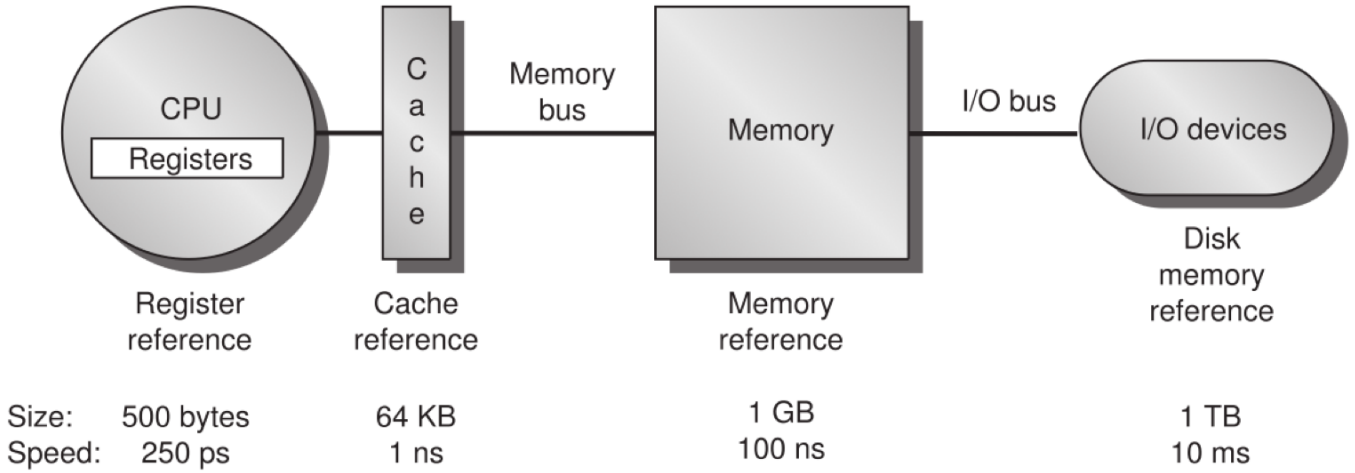
Make temporal behavior as important as logical function.

Timing precision with performance: Challenges:

- ISAs with timing (repeatable instr. timing? deadline instructions?)
- Deep pipelines (interleaving?)
- Memory hierarchy (scratchpads? DRAM banks?)
- Predictable memory management (Metronome?)
- Languages with timing (discrete events? Giotto?)
- Predictable concurrency (synchronous languages?)
- Composable timed components (actor-oriented?)
- Multicore PRET (conflict-free routing?)
- Precision networks (TTA? Time synchronization?)

Edwards and Lee, "**The Case for the Precision Timed (PRET) Machine**," *Wild and Crazy Ideas* Track, *Design Automation Conference* (DAC), June 2007.

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 is application independent?*

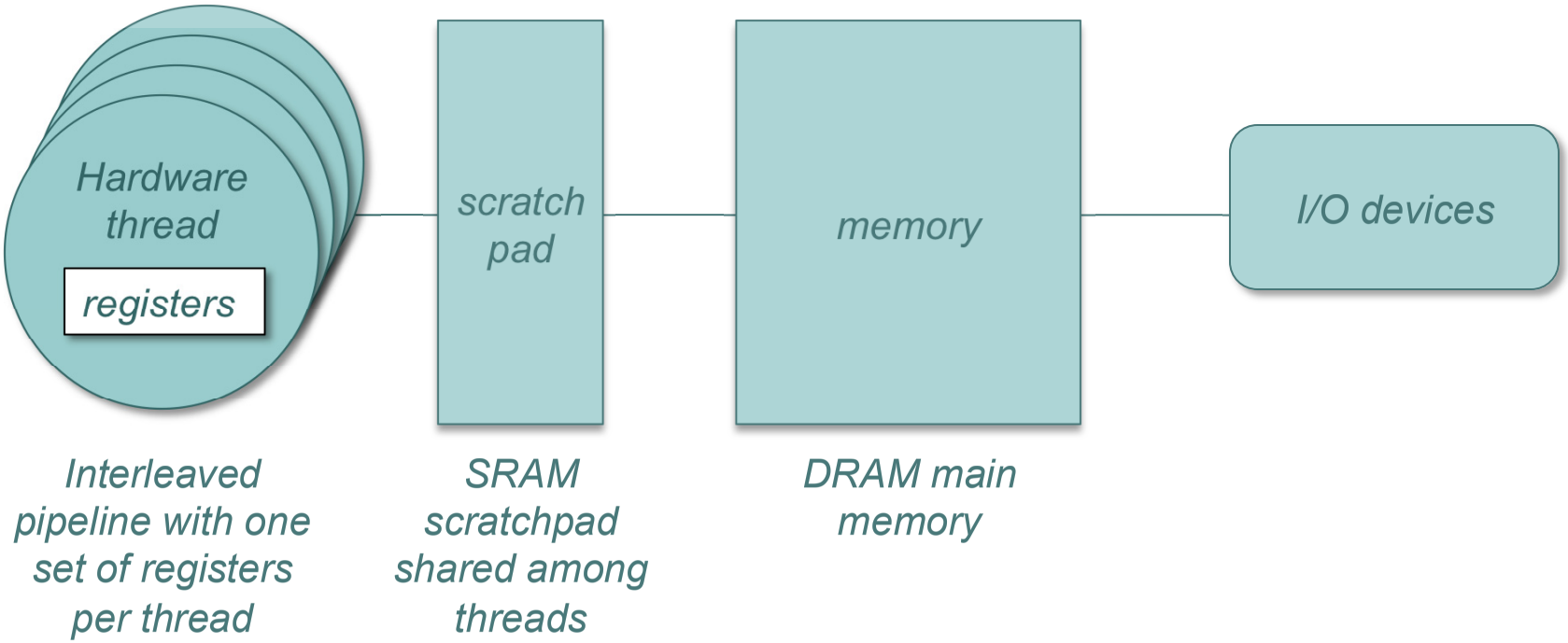
Separation of concerns.

PRET principle: all temporary memory is under program control.

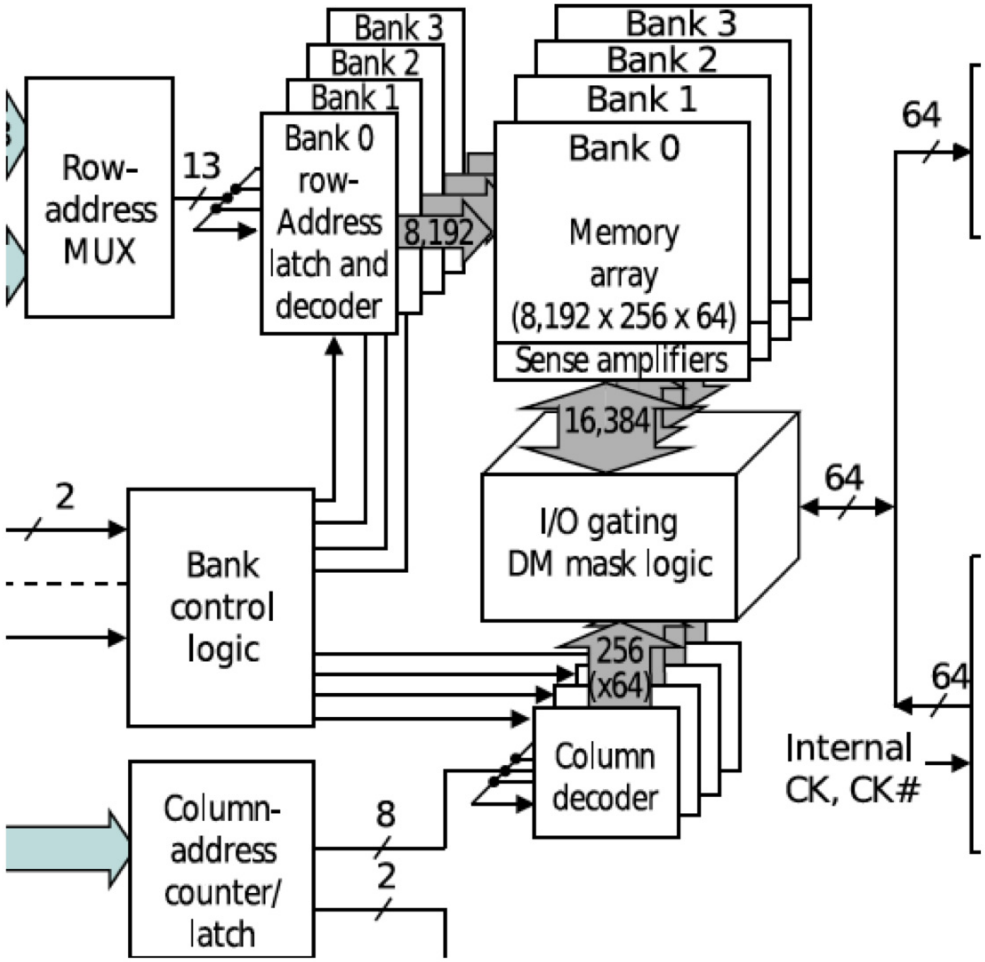
Dynamic RAM vs Static RAM

- Static RAM (SRAM) is
  - low capacity: each cell has six transistors
  - low access latency
  - typically on-chip in lower levels of memory hierarchy
- Dynamic RAM (DRAM) is
  - high capacity: one transistor and one capacitor
  - higher access latency
  - typically off-chip in higher levels of memory hierarchy (with some recent exceptions)

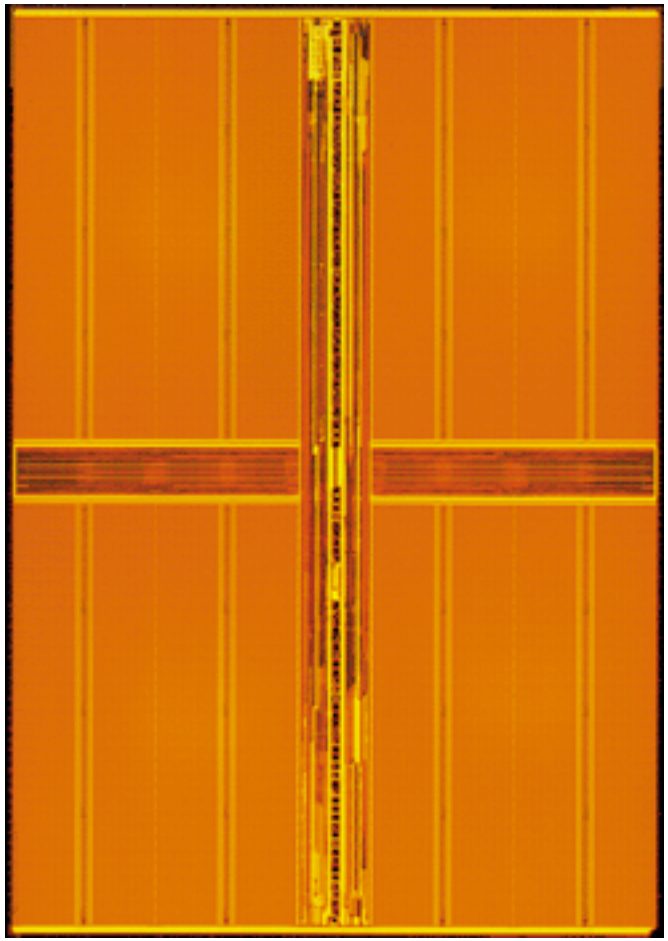
One Possible PRET Architecture



What about Main Memory? Modern DRAMs:



Micron corp.



DDR2: Four pipelined banks
DDR3: Eight pipelined banks
DDRn: 2^n pipelined banks?

General-Purpose DRAM Controllers

- Timing is hard to predict even for single client:
  - Timing of request depends on past requests:
    - Request to same/different bank?
    - Request to open/closed row within bank?
    - Controller might reorder requests to minimize latency
  - Controllers dynamically schedule refreshes

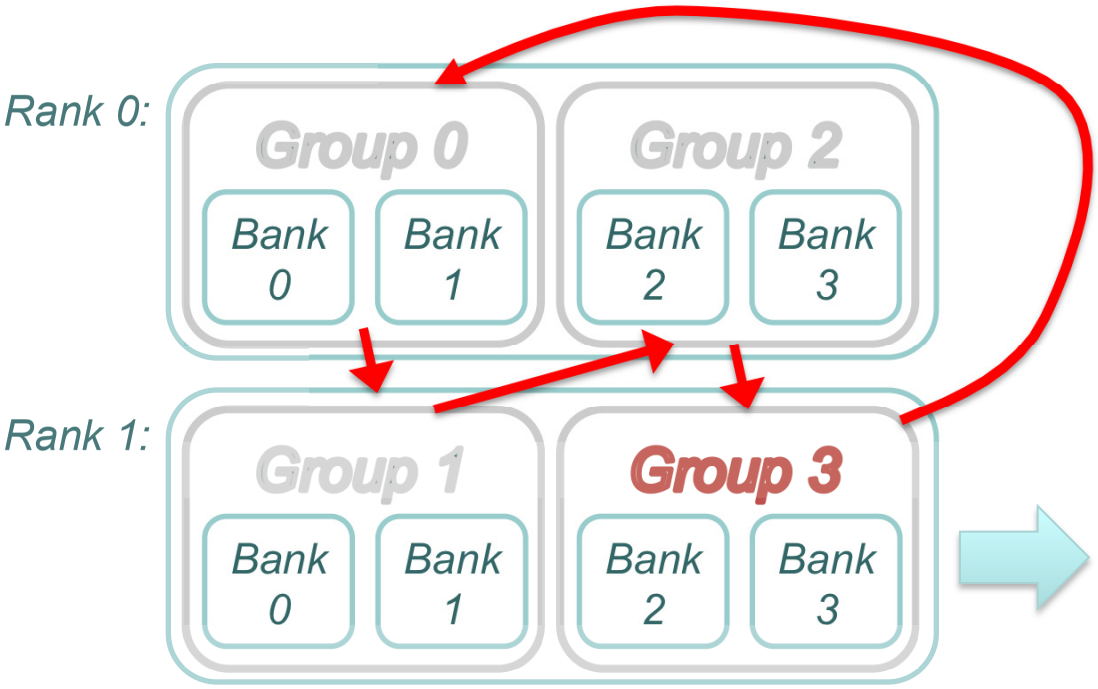
The result is non-composable timing.

Predictable DRAM Controllers: Predator (TU Eindhoven) and AMC (Barcelona)

- Schedule predefined patterns:
  - Decouple access timing from execution history
  - Still schedule refreshes dynamically
  - Allow to determine worst-case access timing (pessimistic regarding refreshes)
- Composable arbitration:
  - Round robin (AMC)
  - Latency-rate servers + Buffers to hide interaction between clients (Predator)

PRET DRAM Controller [Kim and Reineke]

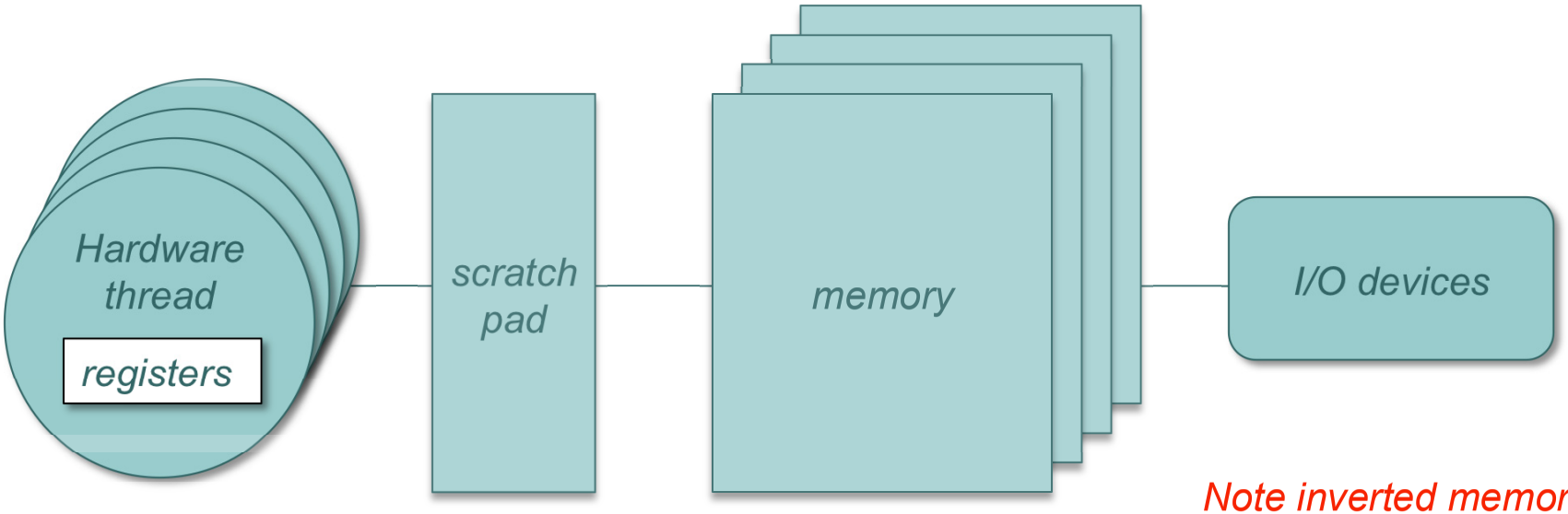
- Exploit internal structure of DRAM module:
  - Consists of 4-8 banks in 1-2 ranks
    - Share only command and data bus, otherwise independent
  - Partition into four groups of banks in alternating ranks
  - Cycle through groups in a time-triggered fashion



- *Successive accesses to same group obey timing constraints*
- *Reads/Writes to different groups do not interfere*

Provides four independent and predictable resources

Resulting PRET Architecture



Interleaved pipeline with one set of registers per thread

SRAM scratchpad shared among threads

DRAM main memory, separate banks per thread

Note inverted memory compared to multicore!

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

PRET Machines: Make Timing a Semantic Property of Computers

Make temporal behavior as important as logical function.

Timing precision with performance: Challenges:

- ISAs with timing (repeatable instr. timing? deadline instructions?)
- Deep pipelines (interleaving?)
- Memory hierarchy (scratchpads? DRAM banks?)
- Predictable memory management (Metronome?)
- Languages with timing (discrete events? Giotto?)
- Predictable concurrency (synchronous languages?)
- Composable timed components (actor-oriented?)
- Multicore PRET (conflict-free routing?)
- Precision networks (TTA? Time synchronization?)

Edwards and Lee, "**The Case for the Precision Timed (PRET) Machine**," *Wild and Crazy Ideas* Track, *Design Automation Conference* (DAC), June 2007.

Extending an ISA with Timing Instructions

[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
```

Exporting the Timed Semantics to a Low-Level Language (like C)

Example:

```
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)
}
```

This realizes variant 3, "immediate miss detection," using setjmp and longjmp to handle timing exceptions. setjmp() returns 0 when directly invoked, and returns non zero when invoked by longjmp().

If the code block takes longer than 500ms to run, then exception 0 will be thrown, and the handler code will run longjmp, which will return control flow to setjmp, but returning non zero. The else statement will then be run, causing the panic procedure to be called.

This pseudocode is neither C-level nor assembly, but is meant to explain an assembly-level implementation.

Summary of ISA extensions

- [V1] Execute a block of code taking at least a specified *time* [Ip & Edwards, 2006]
- [V2] Do [V1], and then conditionally branch if the specified *time* was exceeded.
- [V3] Do [V1], but if the specified *time* is exceeded during execution of the block, branch immediately to an exception handler.
- [V4] Execute a block of code taking exactly the specified *time*. MTFD

Variants:

•For V2 – V4, may not impose minimum execution time.

•*Time* may be literal (seconds) or abstract (cycles).

A Brief Word on Multicore PRET Machines

Make temporal behavior as important as logical function.

Timing precision with performance: Challenges:

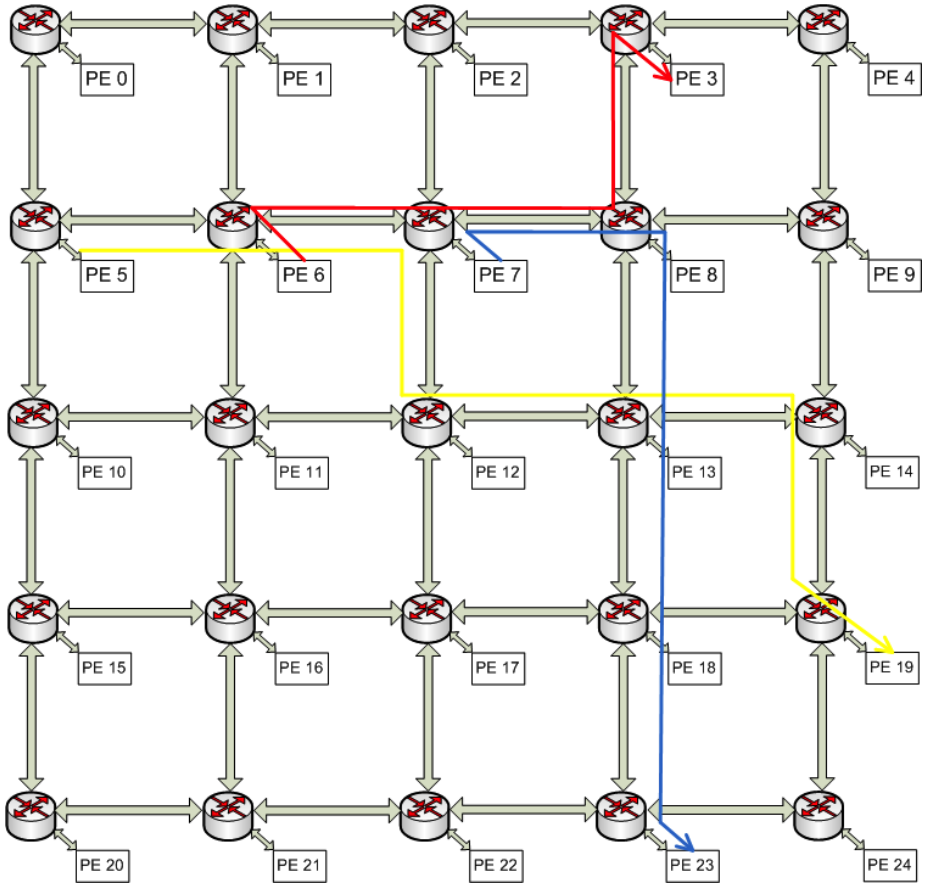
- ISAs with timing (repeatable instr. timing? deadline instructions?)
- Deep pipelines (interleaving?)
- Memory hierarchy (scratchpads? DRAM banks?)
- Predictable memory management (Metronome?)
- Languages with timing (discrete events? Giotto?)
- Predictable concurrency (synchronous languages?)
- Composable timed components (actor-oriented?)
- Multicore PRET (conflict-free routing?)
- Precision networks (TTA? Time synchronization?)

Edwards and Lee, "**The Case for the Precision Timed (PRET) Machine**," *Wild and Crazy Ideas* Track, *Design Automation Conference* (DAC), June 2007.

Multicore PRET

A preliminary project by Dai Bui shows that control over timing enables conflict-free routing of messages in a network on chip.

This means that it becomes possible to have programs on a multicore PRET with compositional timing!

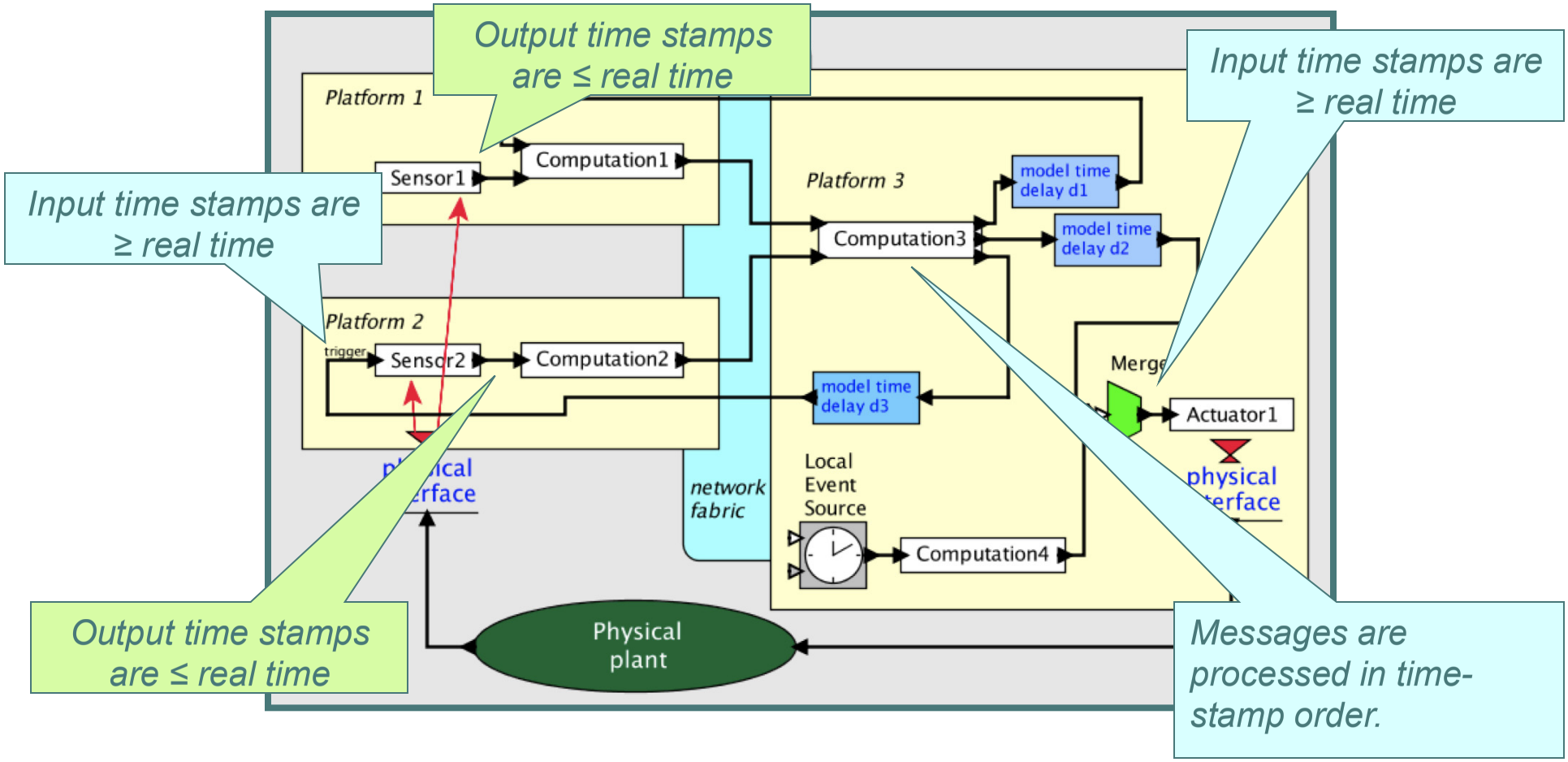


A Few of the (Many) Remaining Challenges and Opportunities

- DRAM designs today compromise efficiency even with private banks (e.g. write-after-read latencies)
- Interleaved pipelines may not be the best choice for power optimization
- Exposing timing properties in programming models (completely absent in today's languages)
- I/O mechanisms that do not disrupt repeatable timing
- More work needed on multicore.
- …

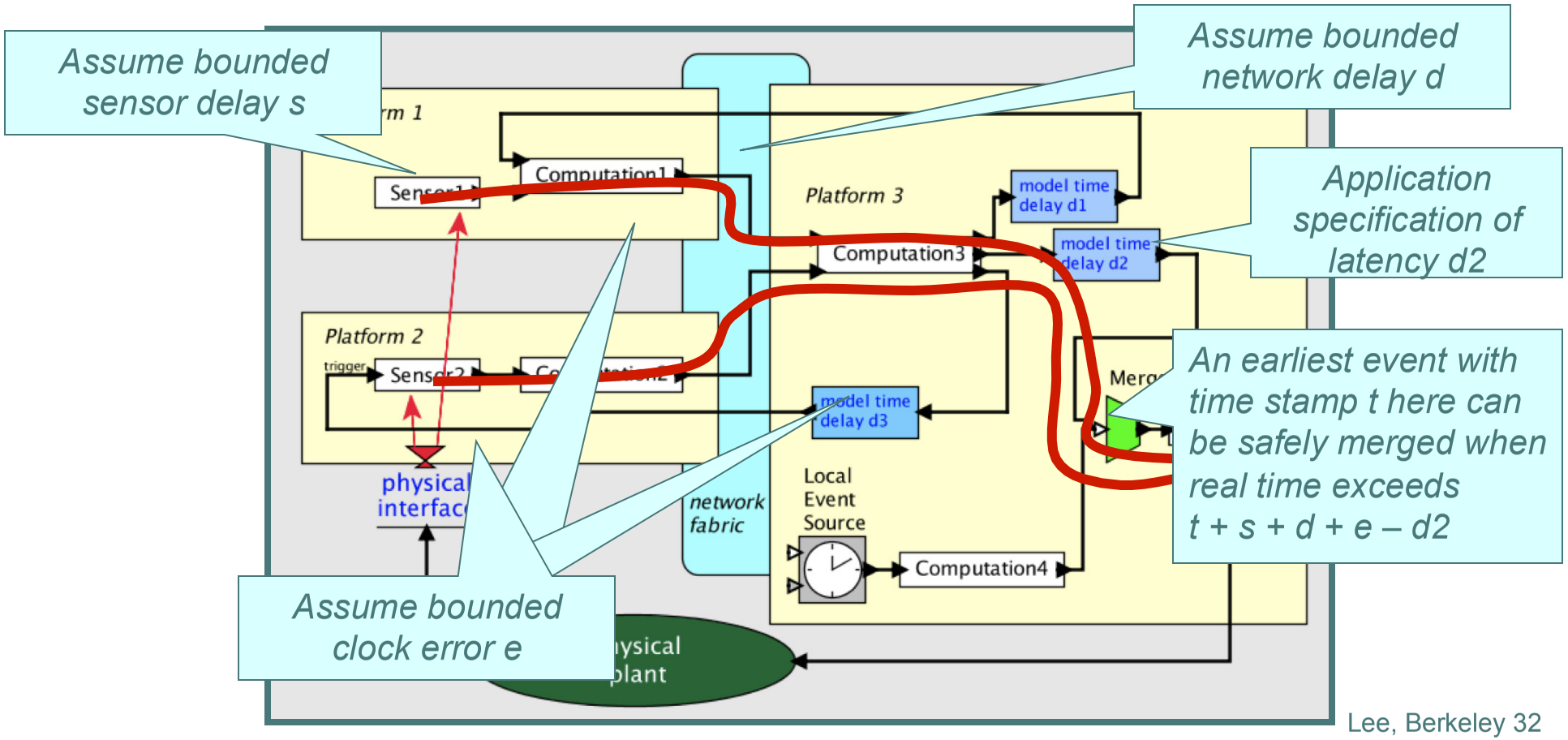
A Top Down Approach:
Make Timing a Semantic Property of Software Components

PTIDES: Distributed execution under discrete-event semantics, with "model time" and "real time" bound at sensors and actuators.



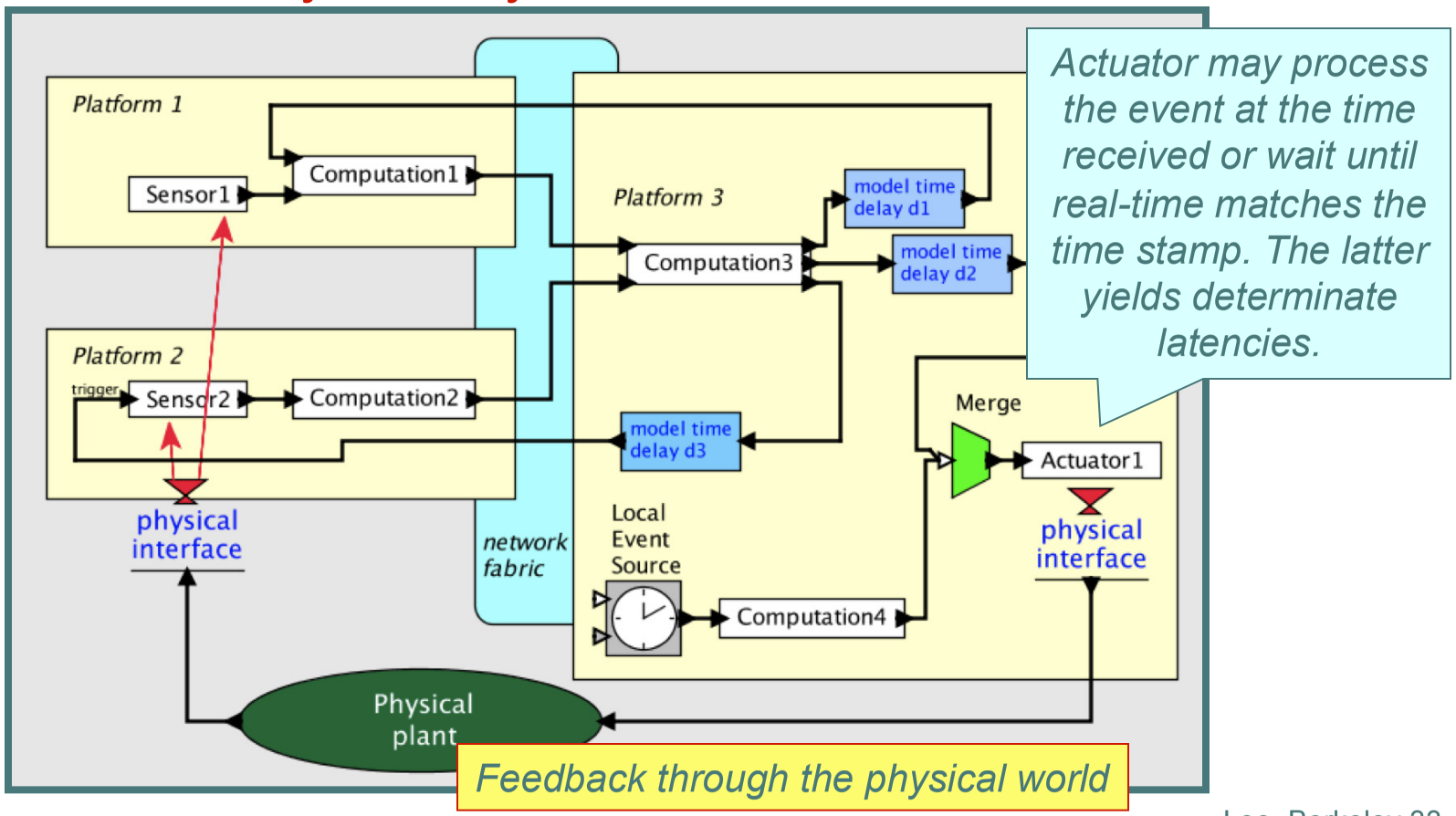
PTIDES: Programming Temporally Integrated Distributed Embedded Systems

PTIDES uses static causality analysis to determine when events can be safely processed (preserving DE semantics).



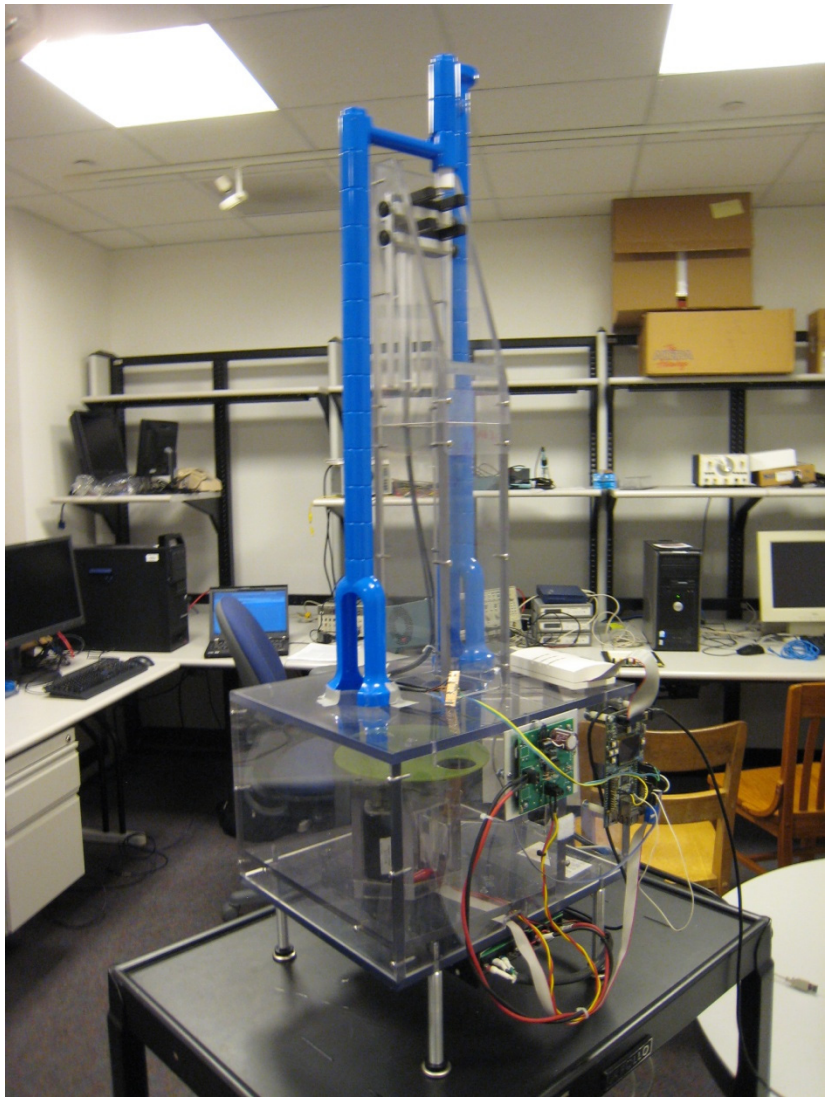
PTIDES: Programming Temporally Integrated Distributed Embedded Systems

… and being explicit about time delays means that we can analyze control system dynamics…

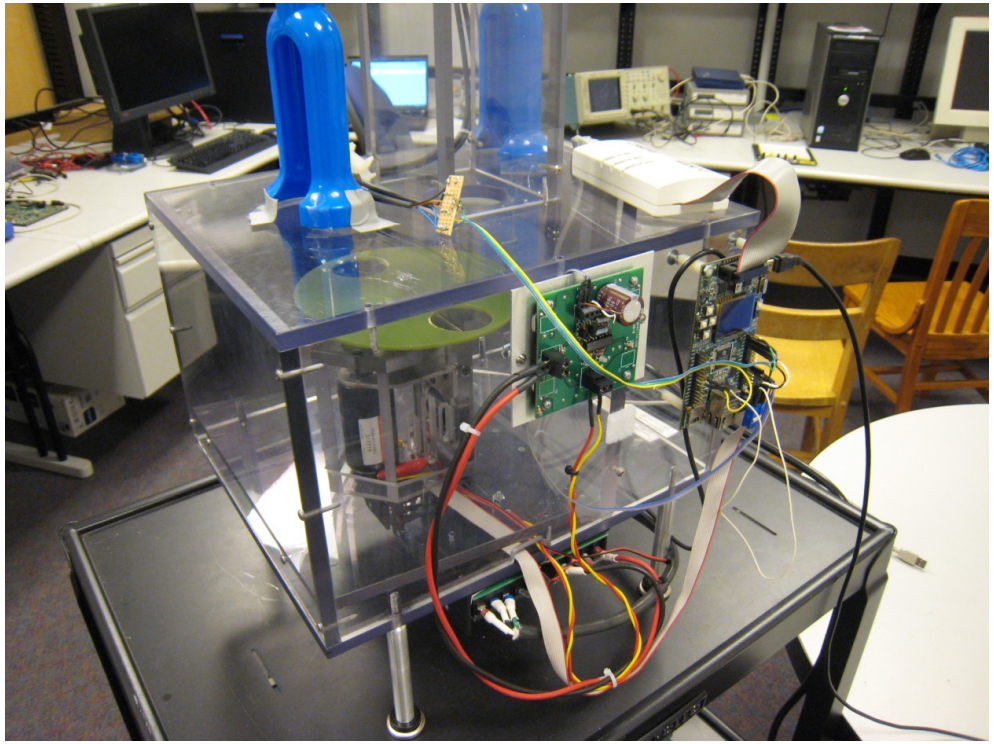


This device was designed by Jeff Jensen, now at National Instruments.

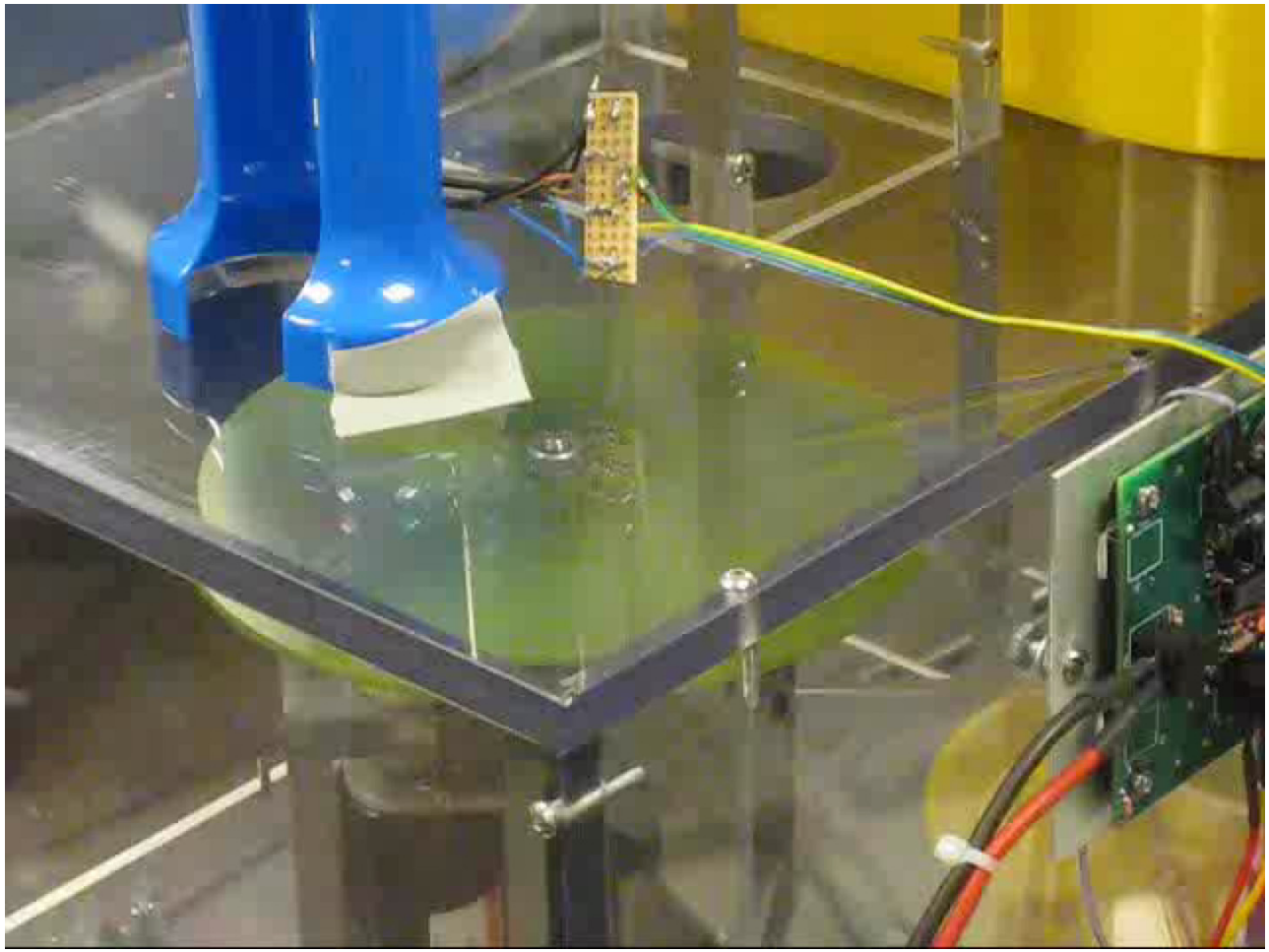
First Test Case



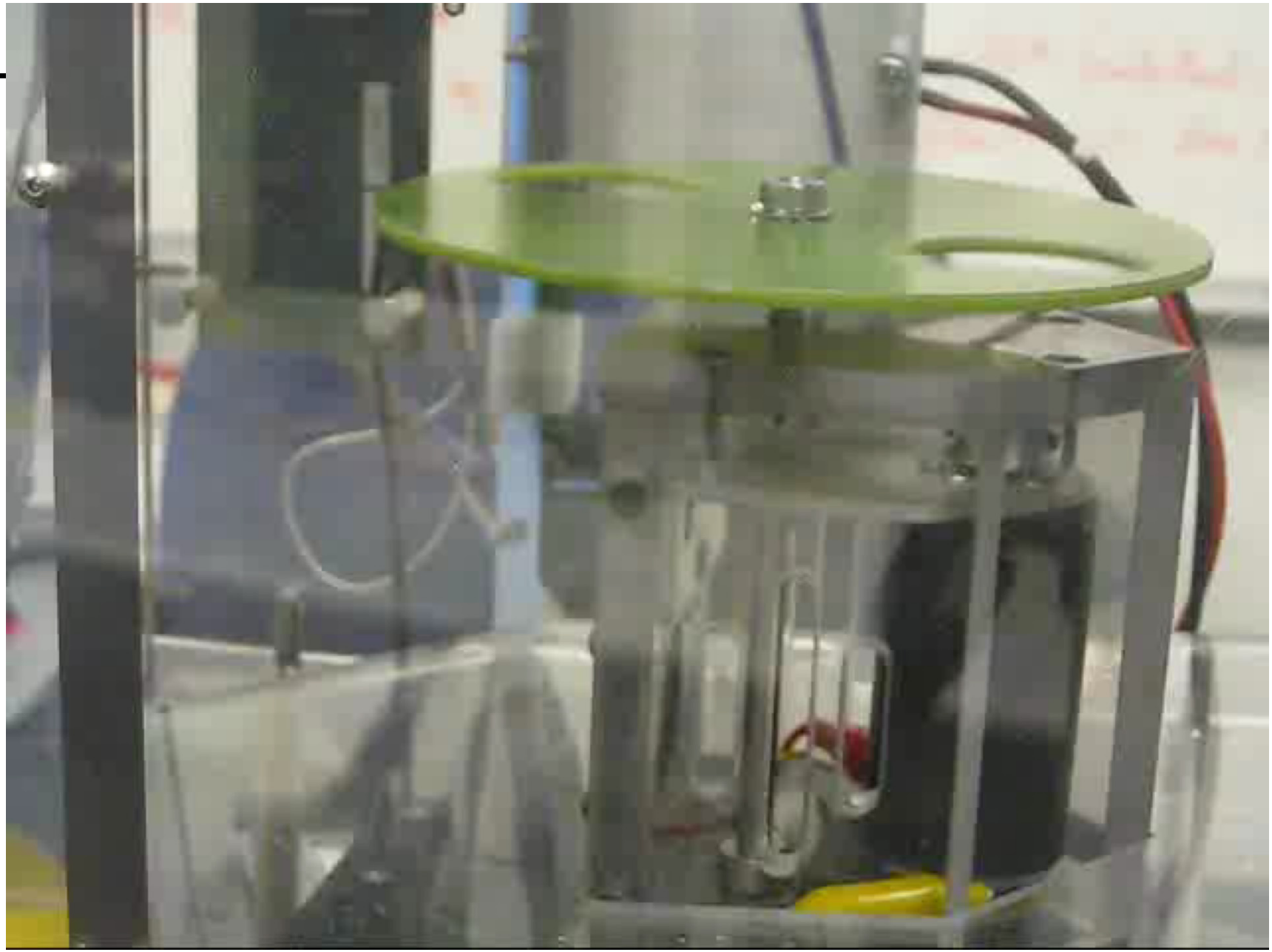
- *Tunneling Ball Device*
  - *sense ball*
  - *track disk*
  - *adjust trajectory*



Tunneling Ball Device in Action

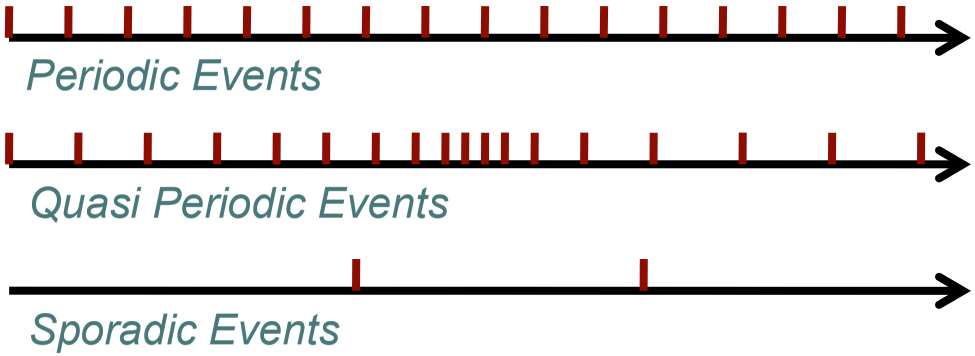


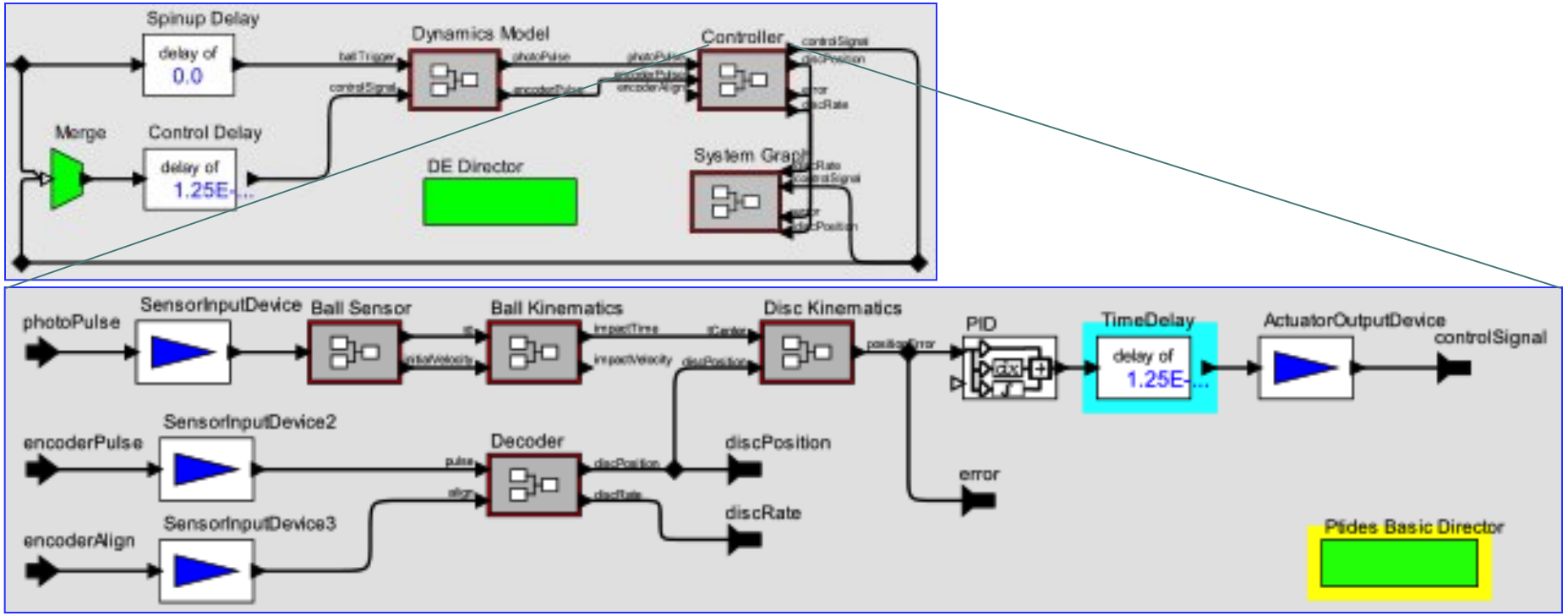
T



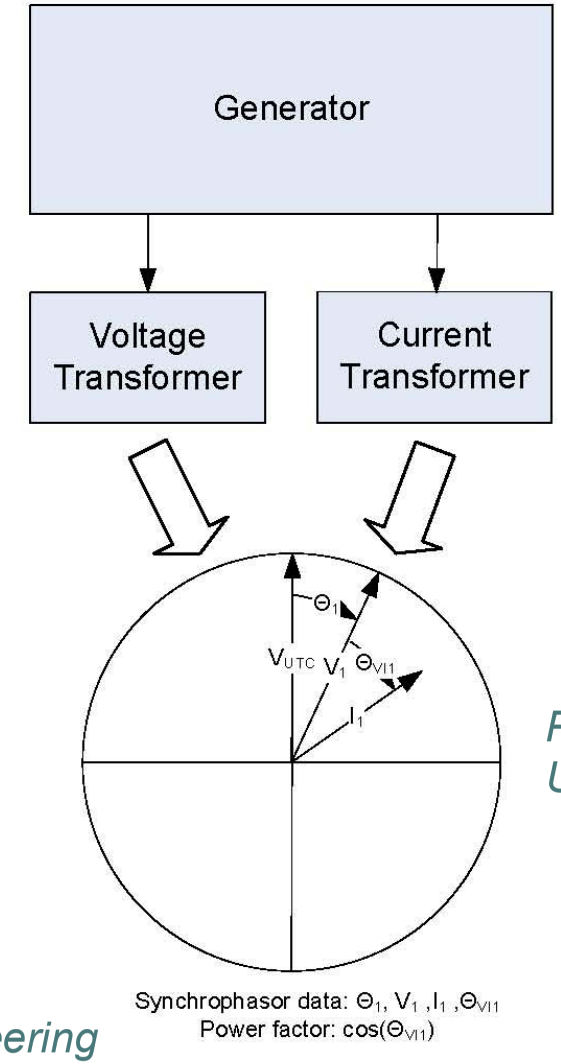
Tunneling Ball Device

Mixed event sequences



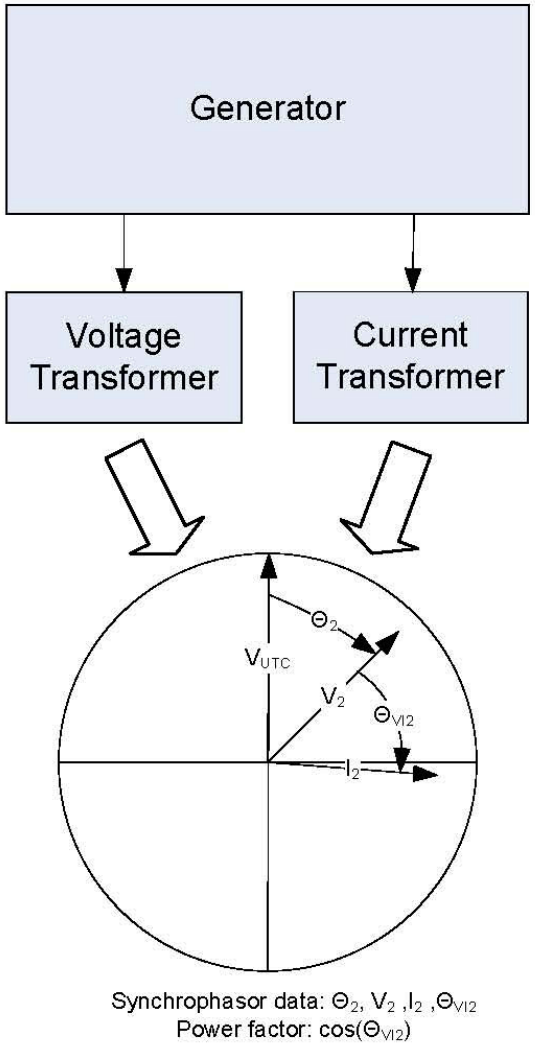


Second Test Case: Distributed Synchrophasor Measurement & Control



Synchrophasor data: Θ_1, V_1 ,I_1 ,Θ_{VI1}
Power factor: cos(Θ_{VI1})

Power swing and Unstability detection



Synchrophasor data: Θ_2, V_2 ,I_2 ,Θ_{VI2}
Power factor: cos(Θ_{VI2})

Thanks to Vaselin Skendzic, Schweitzer Engineering

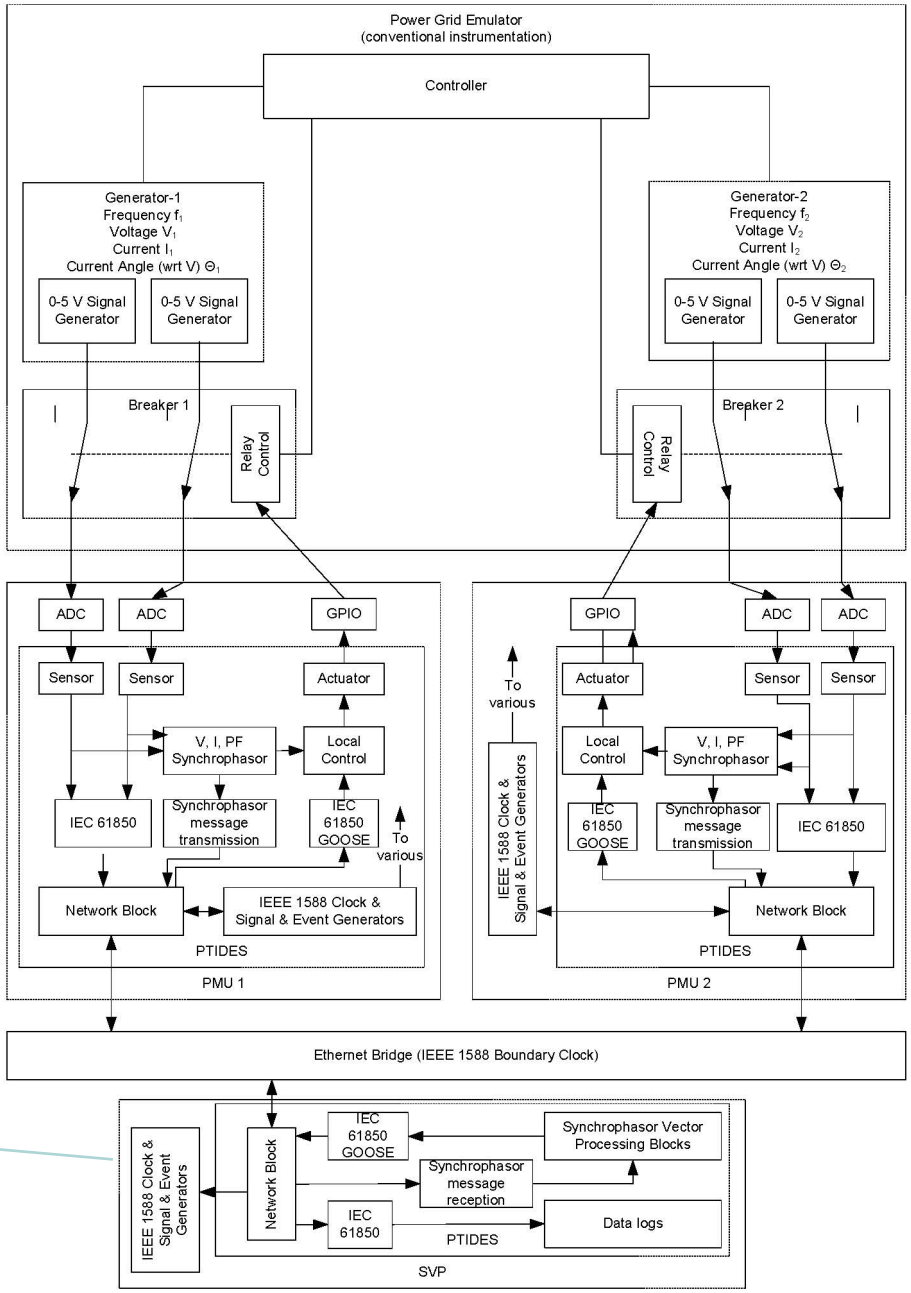
Experiment Diagram

Grid emulator built with National Instruments PXI

'Primary Measurement Unit (PMU) built with Renesas demo boards with DP83640

Ethernet bridge or 1588 boundary/transparent clock

Synchrophasor Vector Processing unit (SVP) built with Renesas demo board with DP83640



Thanks to Vaselin Skendzic, Schweitzer Engineering

The question addressed by the PTIDES project:

If you assume that computers on a network can agree on the current time of day within some bounded error, how does this change how we develop distributed real-time software?

Our answer: It changes everything!

Our approach: Model-based design based on distributed discrete-event (DE) models with synthesis of embedded software.

Distributed PTIDES Relies on Network Time Synchronization with Bounded Error

Press Release October 1, 2007

National Semiconductor
The Sight & Sound of Information

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

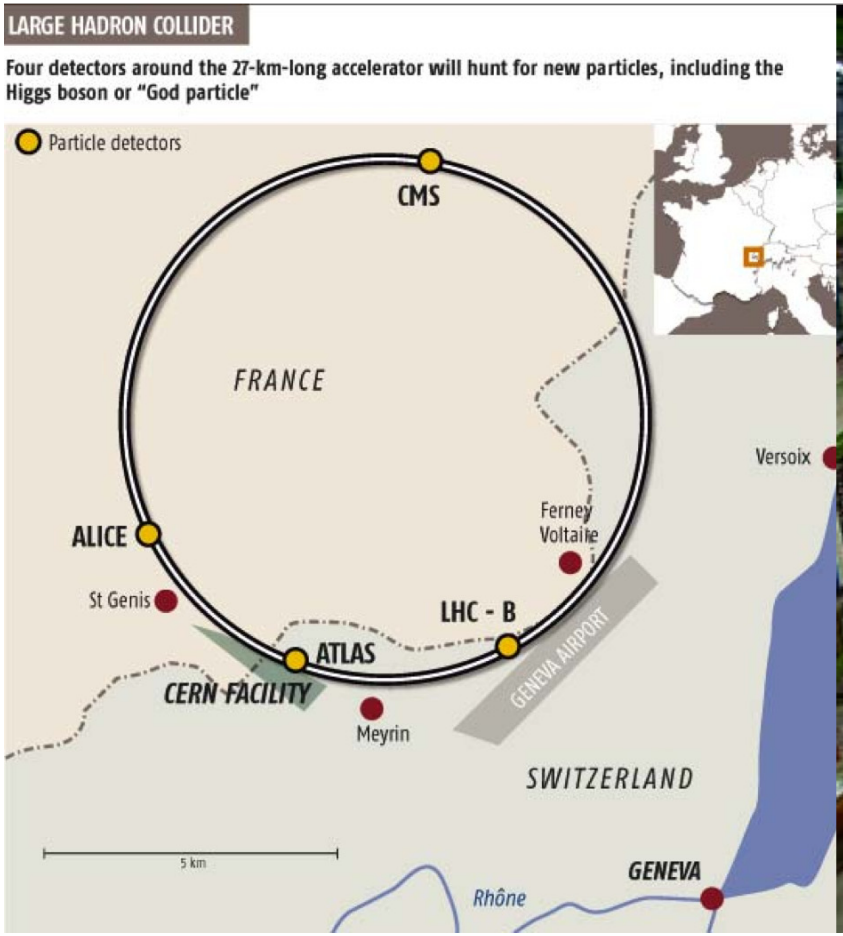
This may become routine!

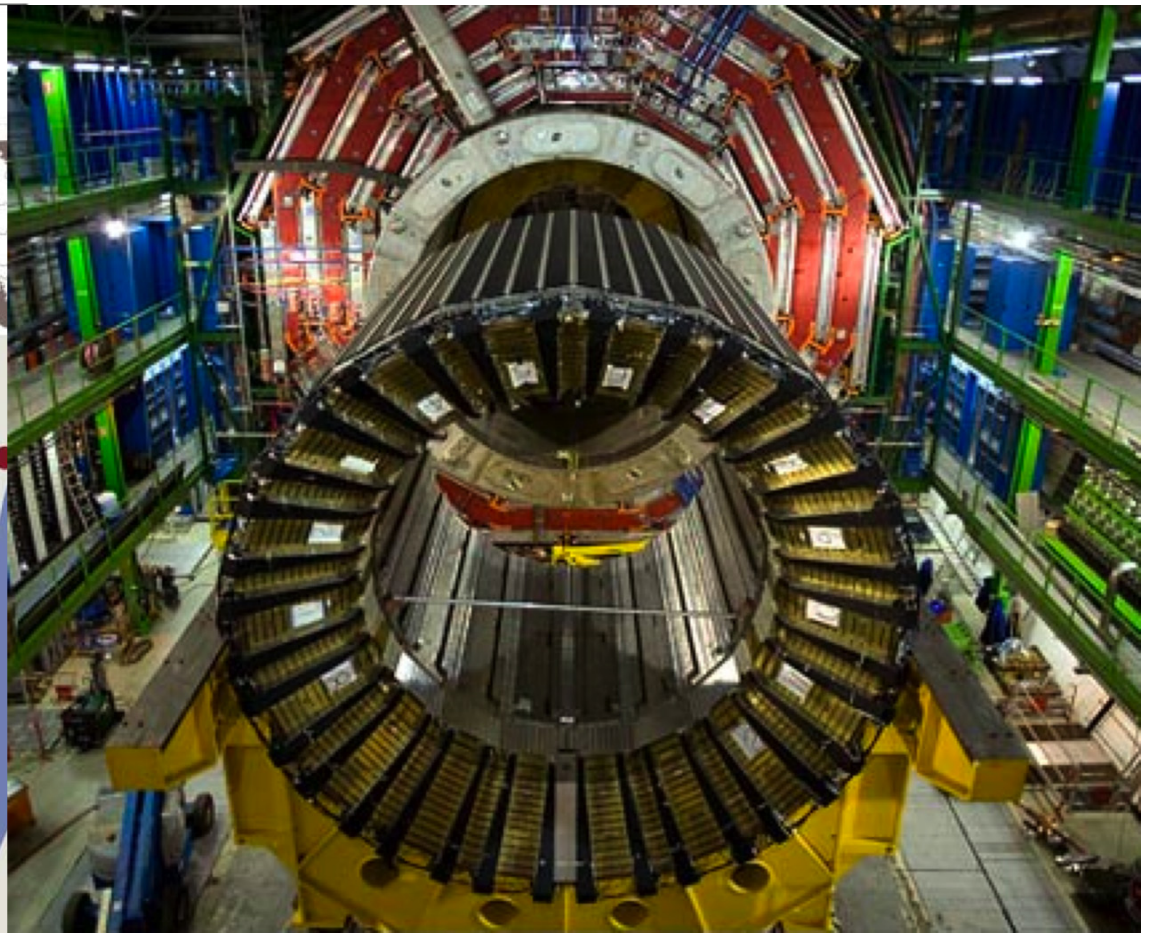
With this PHY, clocks on a LAN agree on the current time of day to within 8ns, far more precise than older techniques like NTP.

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

An Extreme Example: The Large Hadron Collider

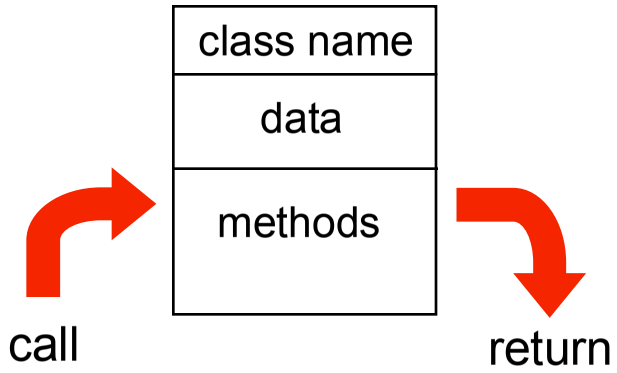
The WhiteRabbit project at CERN is synchronizing the clocks of computers 10 km apart to within about 80 psec using a combination of IEEE 1588 PTP and synchronous ethernet.





More Generally than PTIDES: *Rethinking Software Components to Admit Time*. Object Oriented vs. Actor Oriented

The established: Object-oriented:

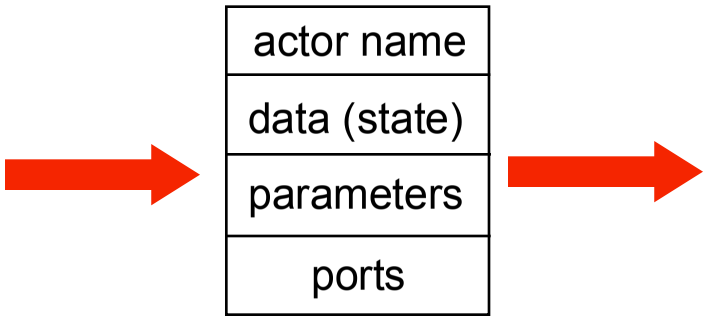


What flows through an object is sequential control

Things happen to objects

The alternative: Actor oriented:

Actors make things happen



What flows through an object is evolving data

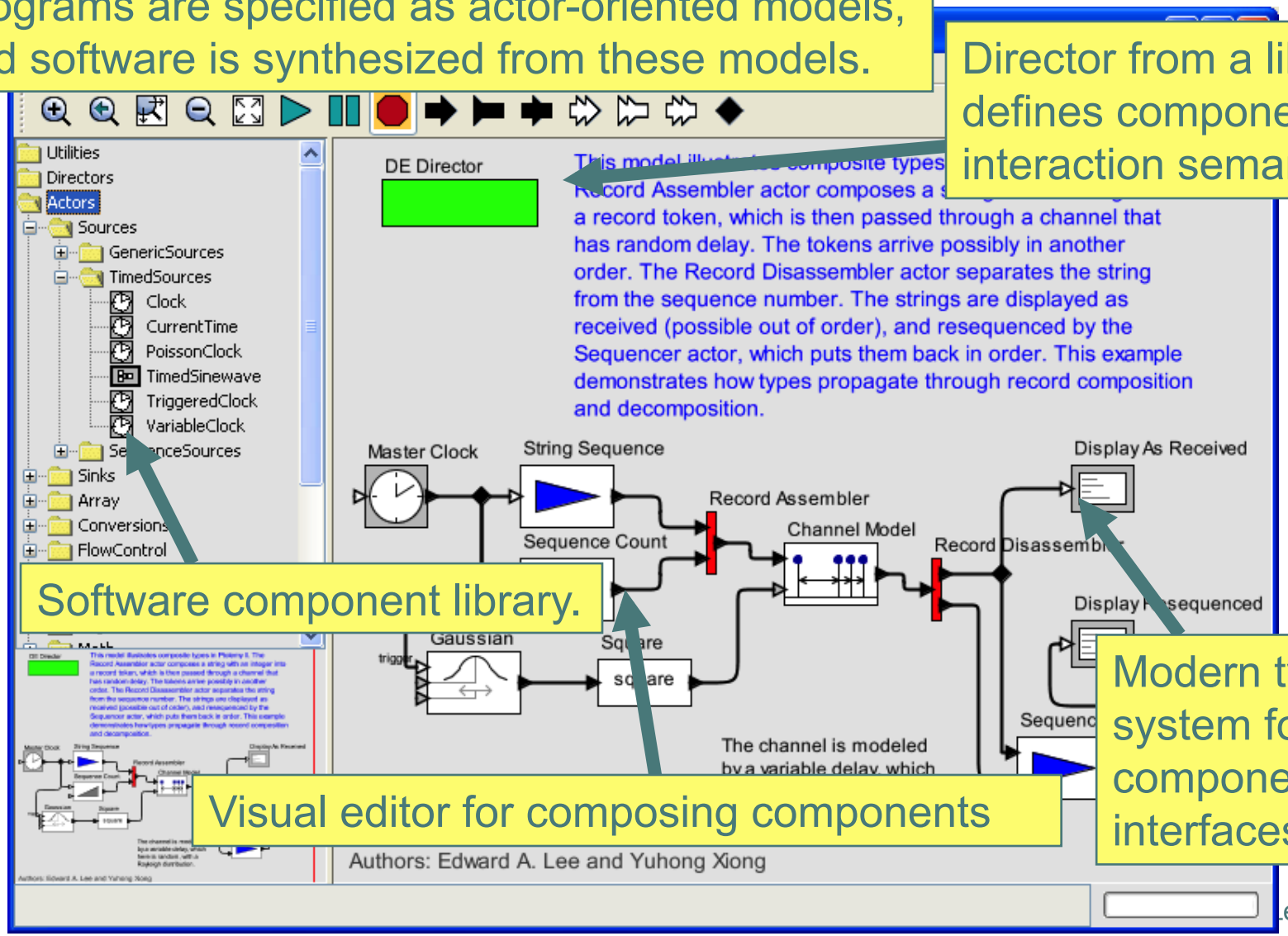
Examples of Actor-Oriented Systems

- UML 2 and SysML (activity diagrams)
- ASCET (time periods, interrupts, priorities, preemption, shared variables)
- Autosar (software components w/ sender/receiver interfaces)
- Simulink (continuous time, The MathWorks)
- LabVIEW (structured dataflow, National Instruments)
- SCADE (synchronous, based on Lustre and Esterel)
- CORBA event service (distributed push-pull)
- ROOM and UML-2 (dataflow, Rational, IBM)
- VHDL, Verilog (discrete events, Cadence, Synopsys, ...)
- Modelica (continuous time, constraint-based, Linkoping)
- OPNET (discrete events, Opnet Technologies)
- SDL (process networks)
- Occam (rendezvous)
- SPW (synchronous dataflow, Cadence, CoWare)
- …

The semantics of these differ considerably in their approaches to concurrency and time. Some are loose (ambiguous) and some rigorous. Some are strongly actor-oriented, while some retain much of the flavor (and flaws) of threads.

Ptolemy II: Our Laboratory for Experiments with Actor-Oriented Design

Programs are specified as actor-oriented models, and software is synthesized from these models.



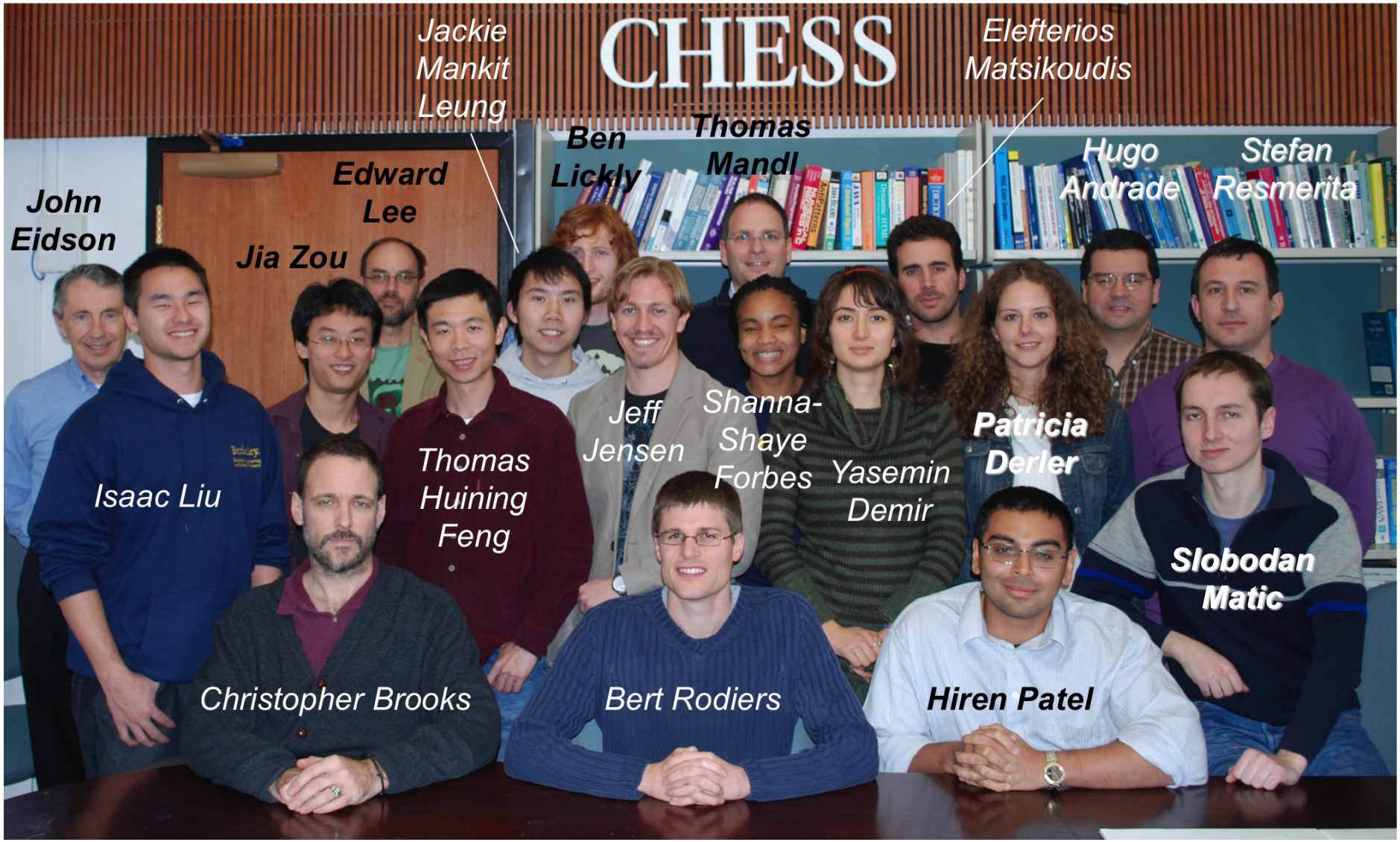
Conclusions

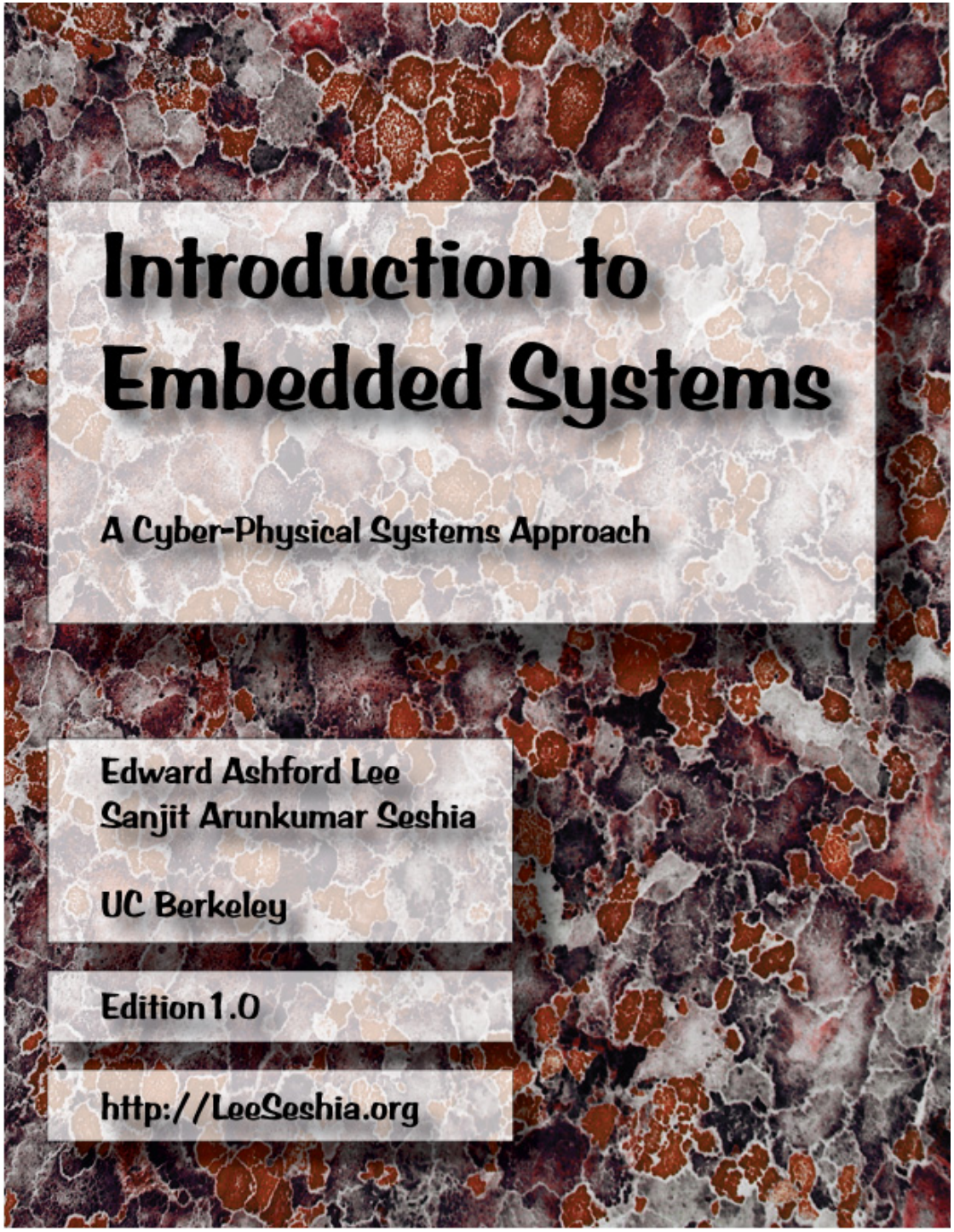
Today, timing is a property only of *realizations* of computational systems.

Tomorrow, timing will be a semantic property of computational *models*.

Raffaello Sanzio da Urbino – The Athens School

The Ptolemy Pteam





New Text: Lee & Seshia: Introduction to Embedded Systems - A Cyber-Physical Systems Approach

http://LeeSeshia.org/

This book strives to identify and introduce the durable intellectual ideas of embedded systems as a technology and as a subject of study. The emphasis is on modeling, design, and analysis of cyber-physical systems, which integrate computing, networking, and physical processes.