Verifying Real-Time Software is Not Reasonable (Today)

Edward A. Lee

Robert S. Pepper Distinguished Professor
UC Berkeley

Invited Plenary Talk

Haifa Verification Conference (HVC)
Haifa, Israel
November 6-8, 2012.

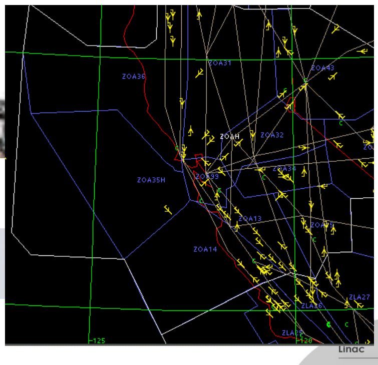
Key Collaborators on work shown here:

- *David Broman*
- *Patricia Derler*
- *Steven Edwards*
- *Isaac Liu*
- *Hiren Patel*
- *Jan Reinke*
- *Sanjit Seshia*
- *Mike Zimmer*
- *Jia Zou*

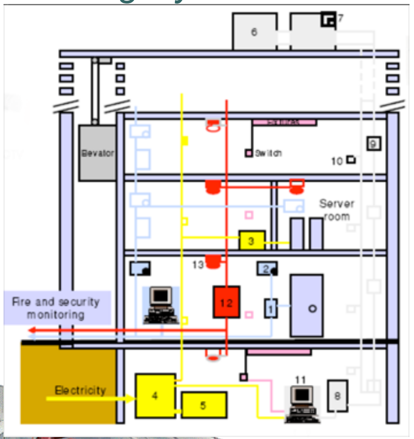
Cyber-Physical Systems (CPS):

Orchestrating networked computational resources with physical systems

Avionics

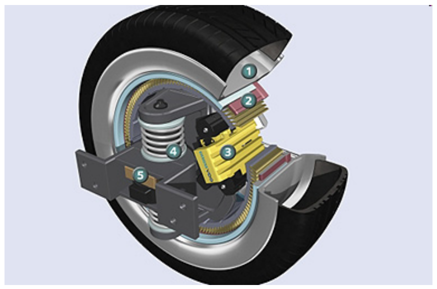

Transportation (Air traffic control at SFO)

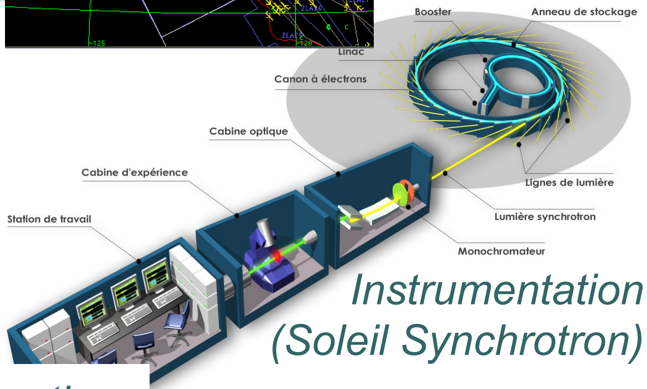
Building Systems



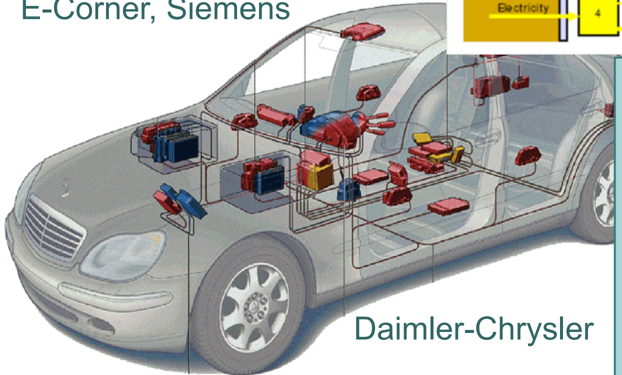
Telecommunications

Automotive


E-Corner, Siemens



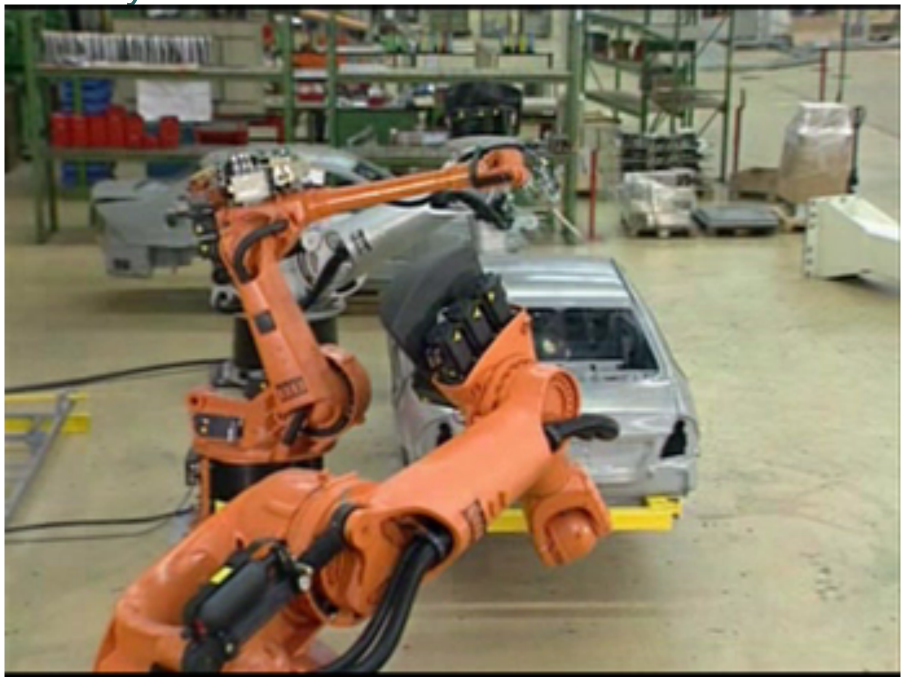
Instrumentation (Soleil Synchrotron)


Daimler-Chrysler

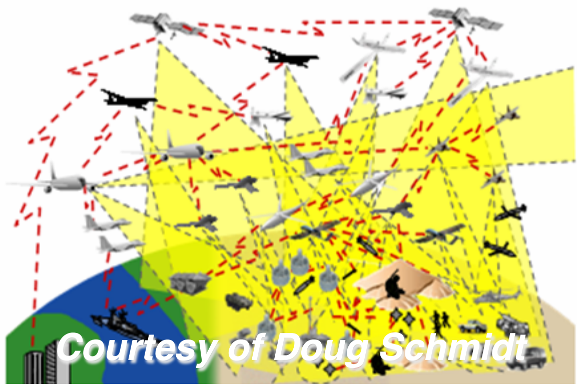
Power generation and distribution

Courtesy of General Electric

Factory automation


Courtesy of Kuka Robotics Corp.

Military systems:

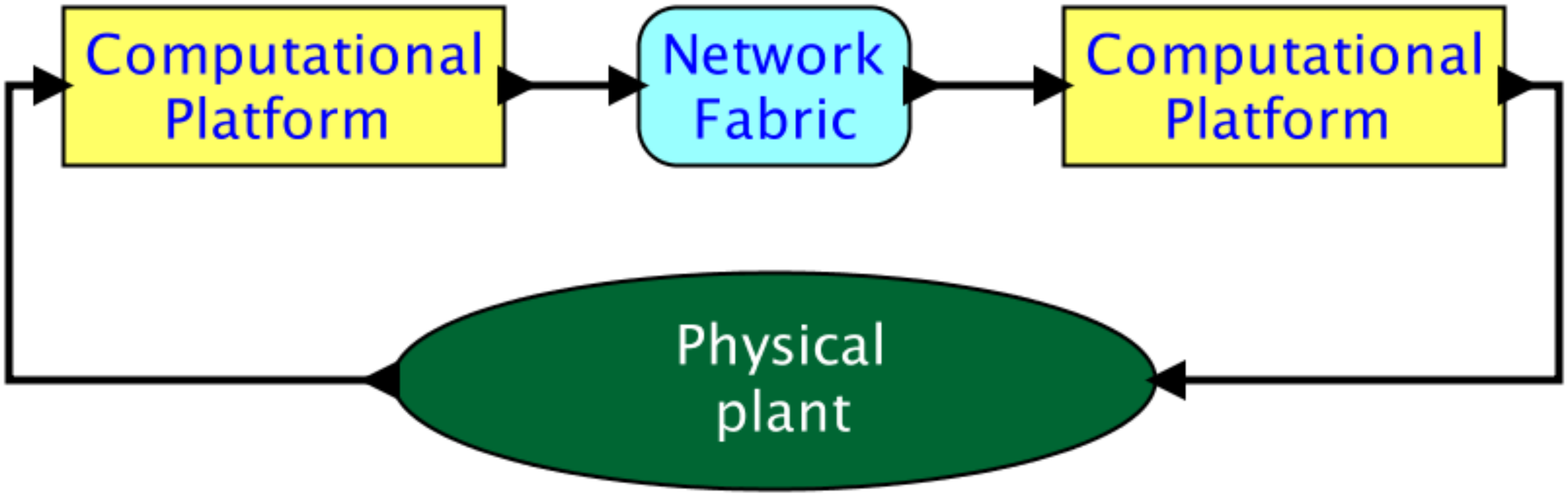

Courtesy of Doug Schmidt

Claim

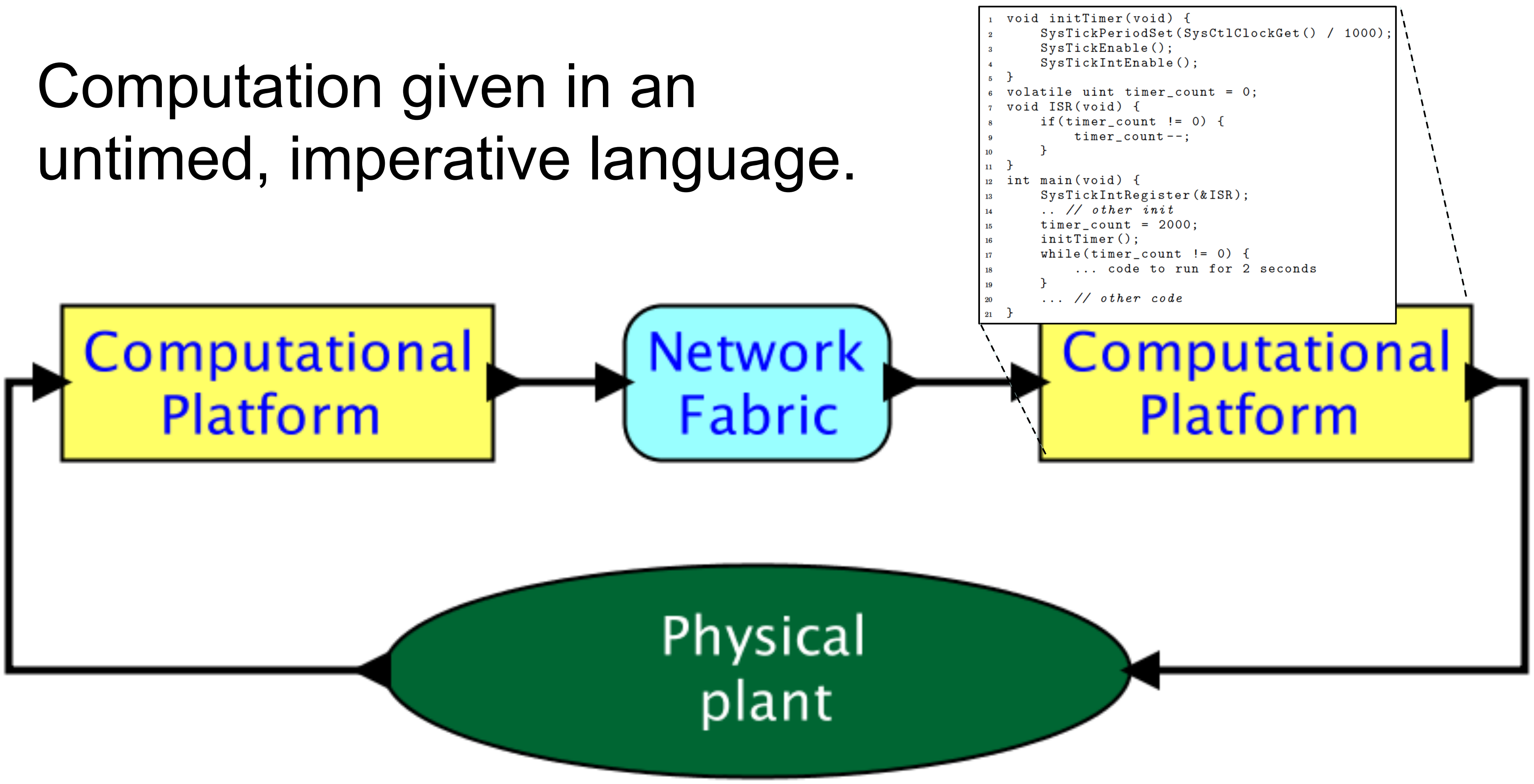
For CPS, *programs* do not adequately specify *timing behavior*.

Corollary: Verifying *software* for CPS is not a reasonable way to ensure timing behavior.

Schematic of a simple CPS:

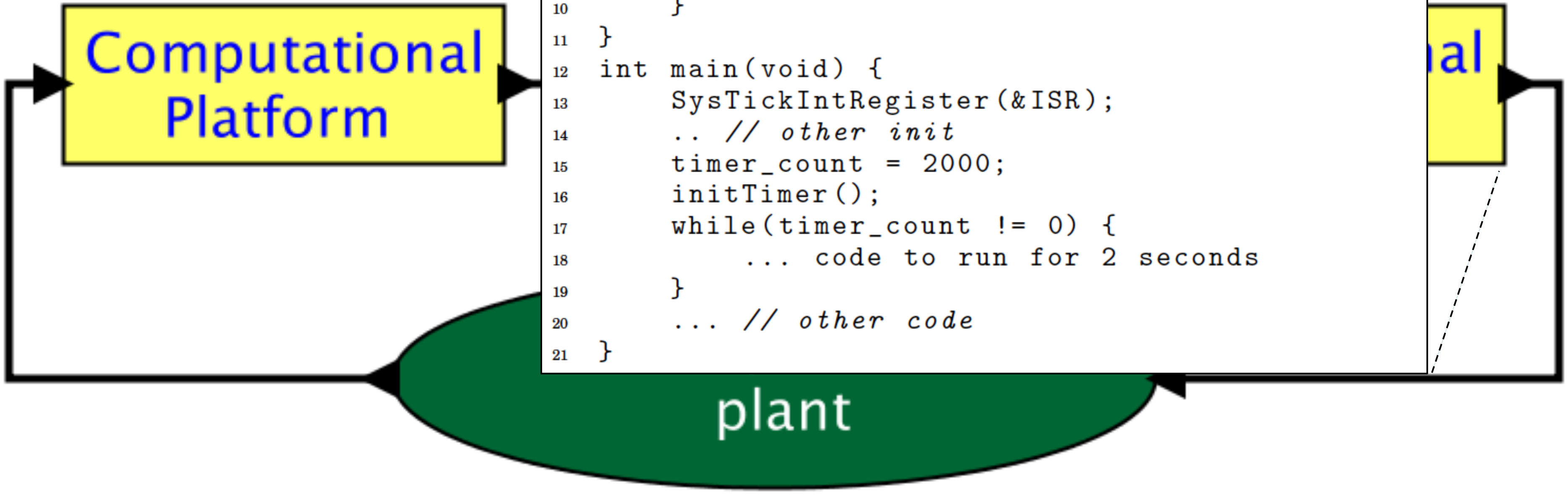


Computation given in an untimed, imperative language.



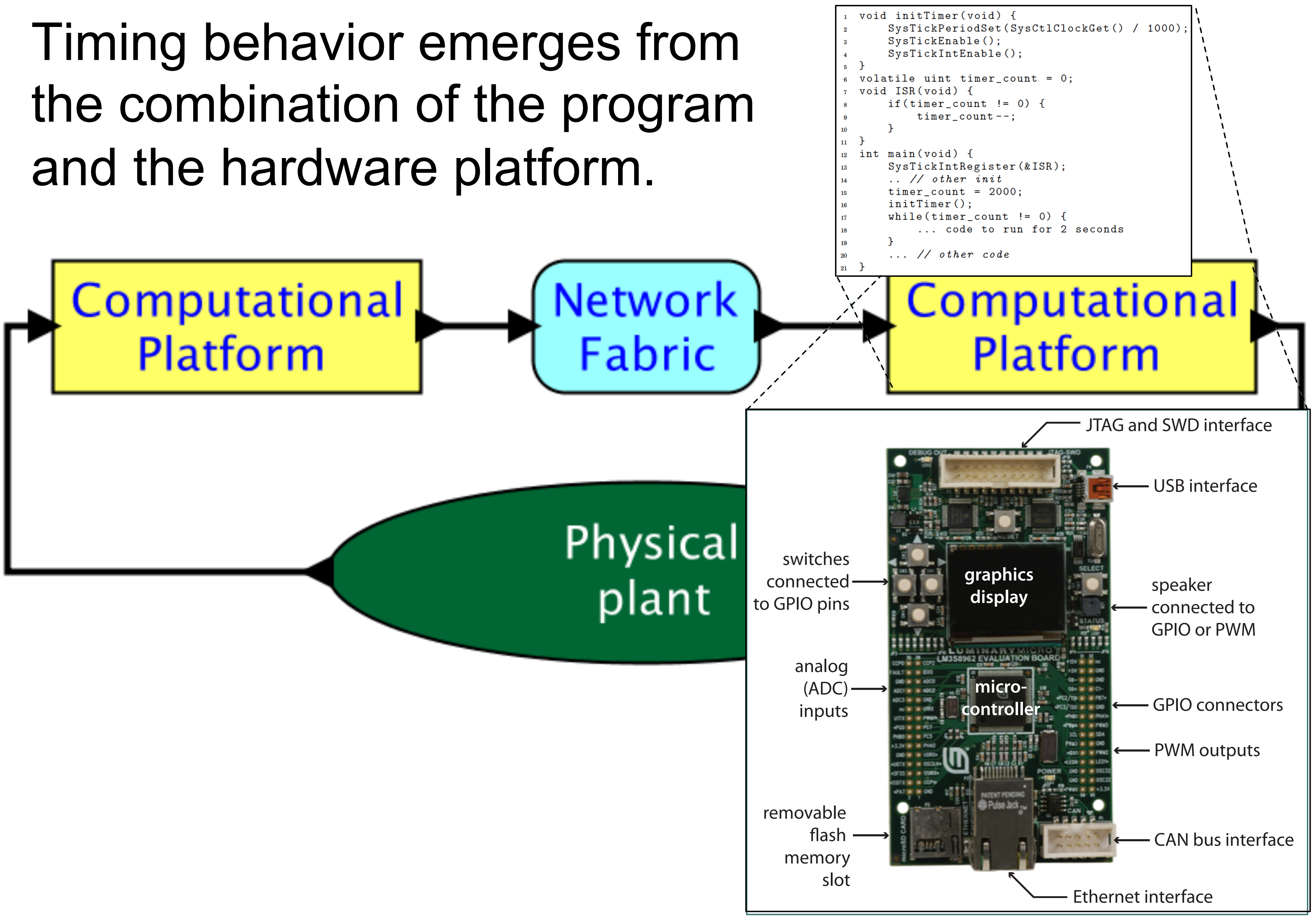
```
void initTimer(void) {
    SysTickPeriodSet(SysCtlClockGet() / 1000);
    SysTickEnable();
    SysTickIntEnable();
}
volatile uint timer_count = 0;
void ISR(void) {
    if(timer_count != 0) {
        timer_count--;
    }
}
int main(void) {
    SysTickIntRegister(&ISR);
    .. // other init
    timer_count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

This code is attempting to control timing. But will it really?



```
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
}
```

Timing behavior emerges from the combination of the program and the hardware platform.



```
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
}
```

Consequences

When precise control over timing is needed, designs are brittle. Small changes in the hardware, software, or environment can cause big, unexpected changes in timing. Testing has to be redone. Results:

- Manufacturers frequently stockpile parts to suffice for the complete production run of a product.
- Manufacturers cannot take advantage of improvements in the hardware (e.g. weight, power). The cost of re-testing and re-certifying is too high.
- Designs are over provisioned, increasing cost, weight, and energy usage.

Guidance for Verification: The Kopetz Principle

Prof. Dr. Hermann Kopetz

Many (predictive) properties that we assert about systems (determinism, timeliness, reliability) are in fact not properties of an *implemented* system, but rather properties of a *model* of the system.

We can make definitive statements about *models*, from which we can *infer* properties of system realizations. The validity of this inference depends on *model fidelity*, which is always approximate.

(paraphrased)

Models vs. Reality

Solomon Golomb: Mathematical models – Uses and limitations. Aeronautical Journal 1968

You will never strike oil by drilling through the map!

Solomon Wolf Golomb (1932) mathematician and engineer and a professor of electrical engineering at the University of Southern California. Best known to the general public and fans of mathematical games as the inventor of polyominoes, the inspiration for the computer game Tetris. He has specialized in problems of combinatorial analysis, number theory, coding theory and communications.

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

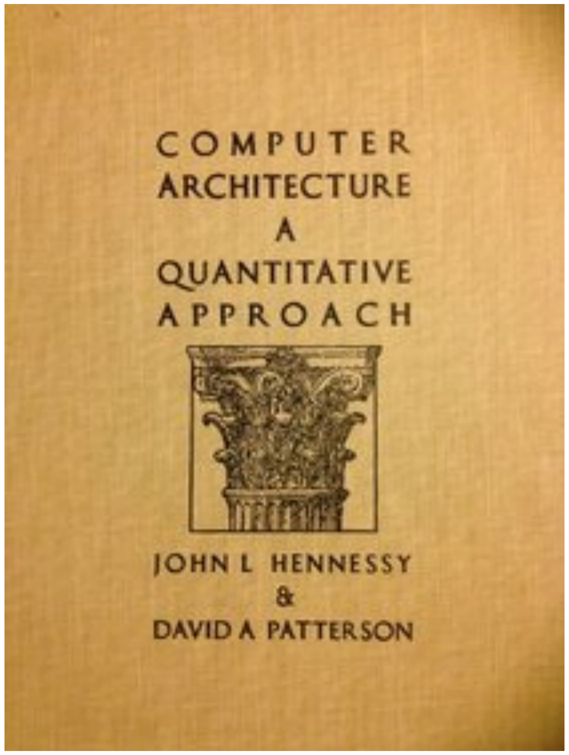
A Key Challenge:
Timing is not Part of Software Semantics

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

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

Programmers have no map!

Computer Science has not *ignored* timing…



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.

Correctness criteria

We can safely assert that line 8 does not execute

(In C, we need to separately ensure that no other thread or ISR can overwrite the stack, but in more modern languages, such assurance is provided by construction.)

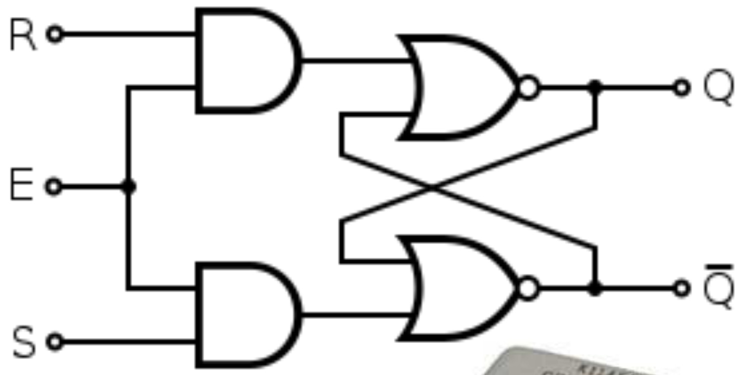
```
1   void foo(int32_t x) {
2           if (x > 1000) {
3                   x = 1000;
4           }
5           if (x > 0) {
6                   x = x + 1000;
7                   if (x < 0) {
8                           panic();
9                   }
10          }
11  }
```

*We can develop **absolute confidence** in the software, in that only a **hardware failure** is an excuse.*

But not with regards to timing!!

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

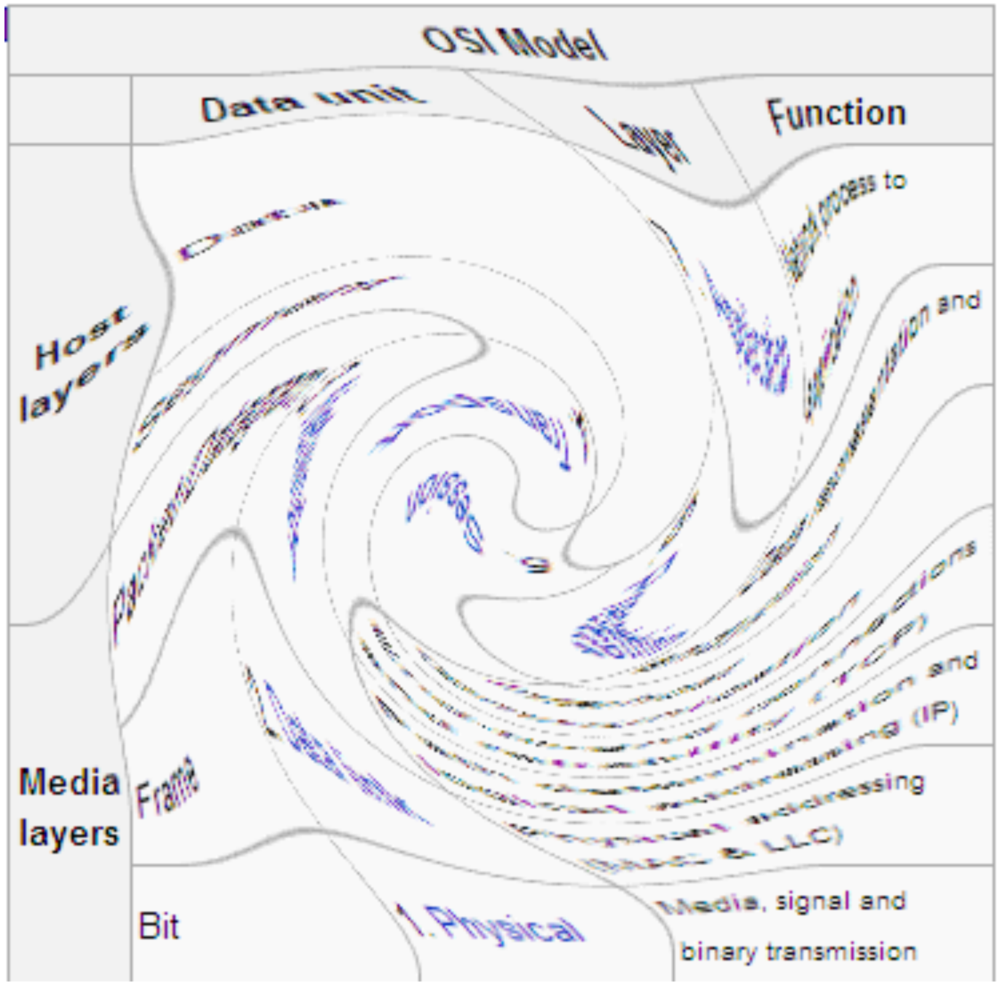
The synchronous digital logic abstraction removes the messiness of transistors.



… 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]);
}
```

As with processors, for networks, timing is a *performance metric*, not a *correctness criterion*



The point of these abstraction layers is to isolate a system designer from the details of the implementation below.

In today's networks, timing emerges from the details of the implementation.

Even QoS-aware networks (e.g. AVB) derive timing properties from packet priorities & network topology.

Consequences

Verification asserts properties of *models,* not properties of real systems.

Verification can only assert properties expressed in the modeling semantics.

If the model says nothing about timing, then timing properties cannot be verified.

Software as-a Model

C program specifying timed behavior.

```
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
}
```

Software as-a Model

C program specifying timed behavior.

Within the semantics of C, how long will this code run?

```
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
}
```

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

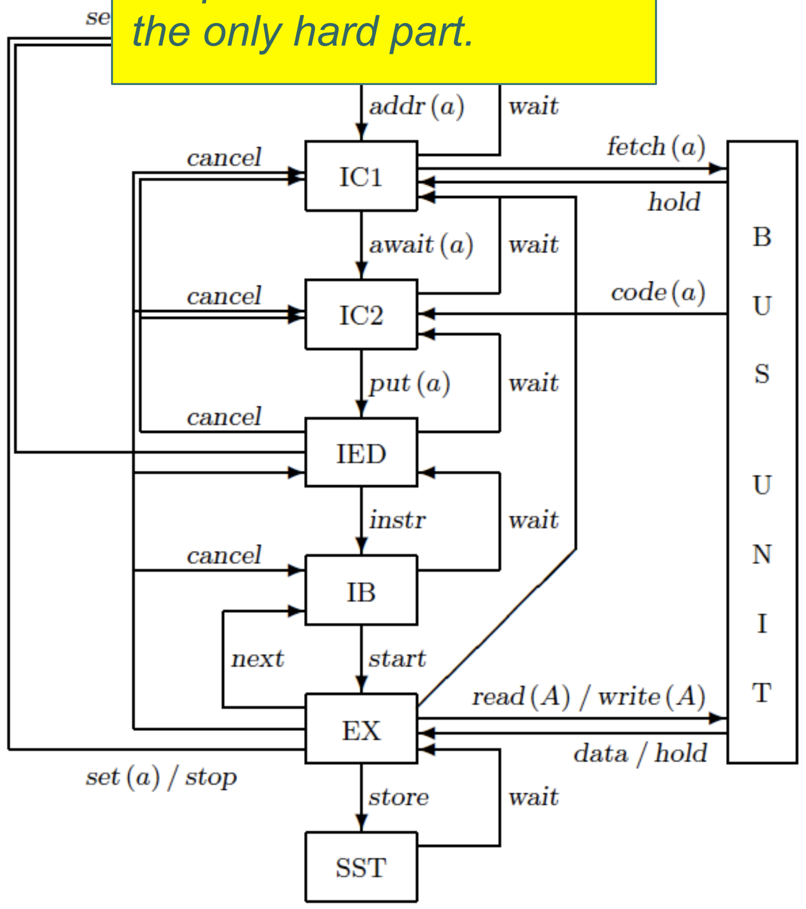
Our first goal is to reduce the problem so that this is the only hard part.

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.



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

First Part of Our Solution: 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)
}
```

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

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!

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
```

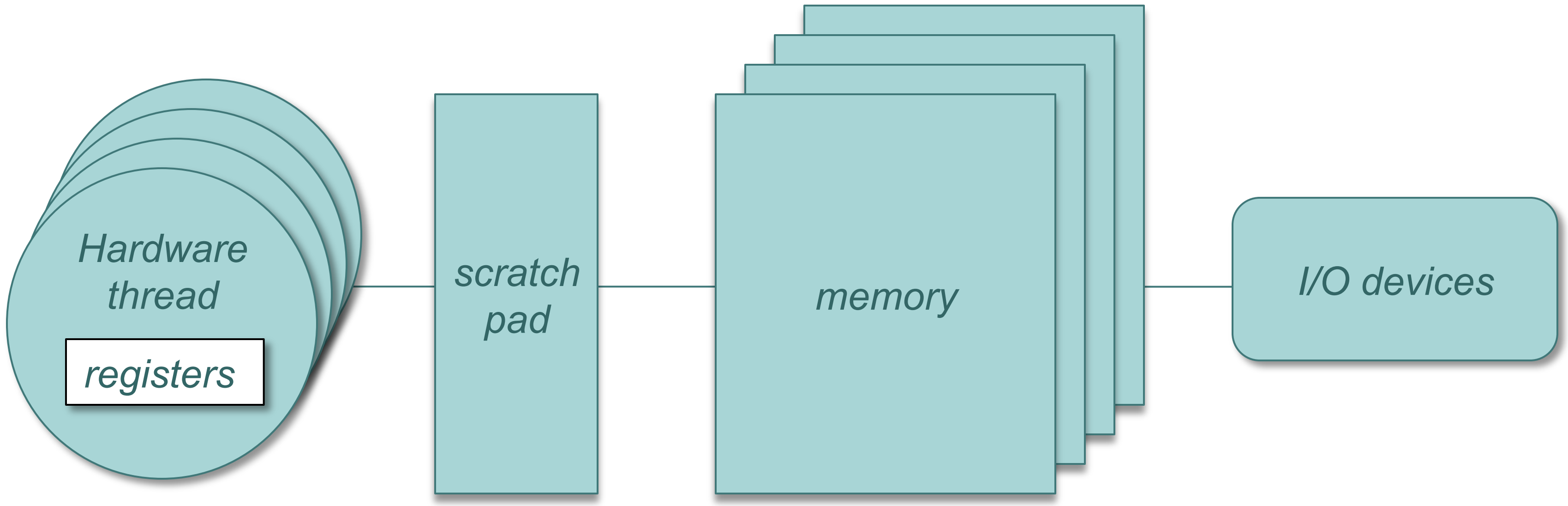
To deliver repeatable timing, we have to rethink the microarchitecture

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



Interleaved pipeline with one set of registers per thread

SRAM scratchpad shared among threads

DRAM main memory, separate banks per thread

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.

Performance Cost?

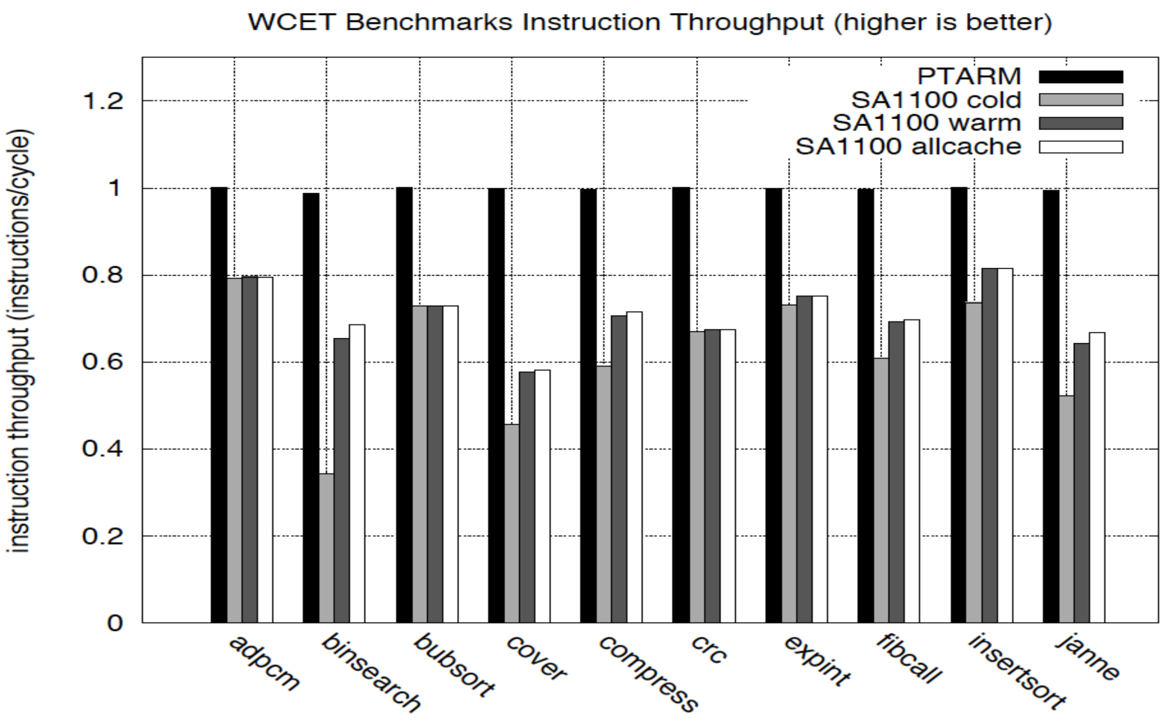
No!

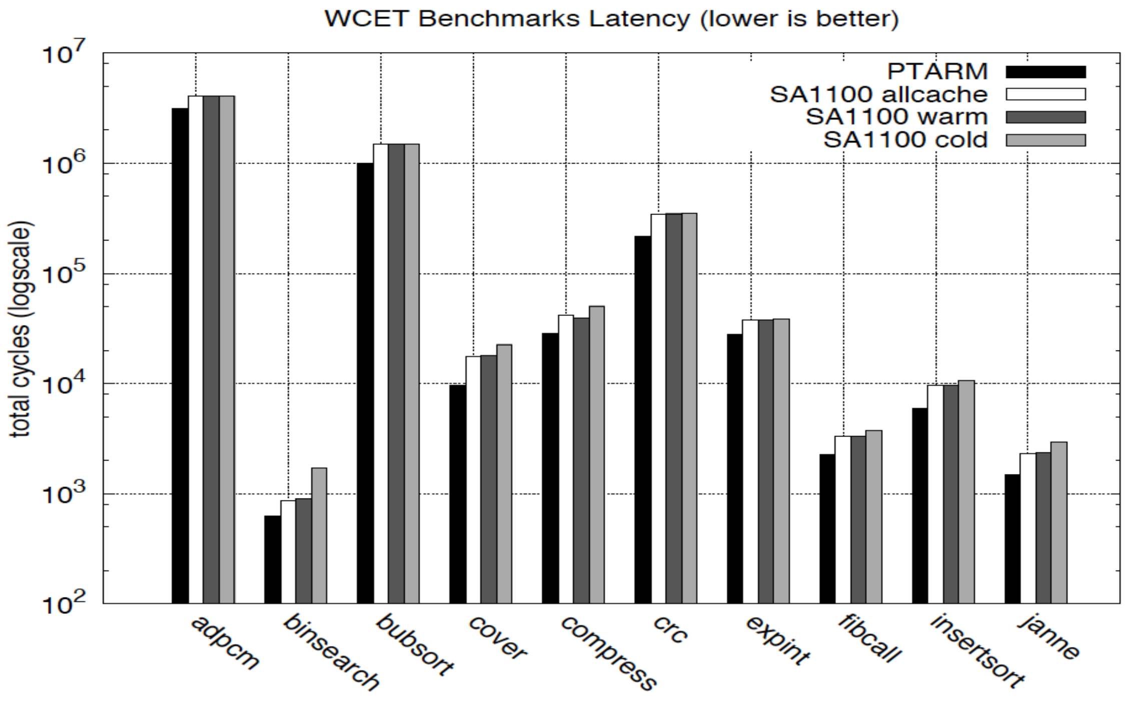
Comparing PTARM against SimIT-ARM simulator (StrongARM 1100) [Qin & Malik] over Malardalen WCET benchmarks [Gustafsson…].

Given enough concurrency, the PTARM beats the StrongARM on every benchmark!

Moreover, our simpler pipeline can probably be clocked faster.

[Isaac Liu, PhD Thesis, May, 2012]





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

A Key Next Step:
Parametric PRET Architectures

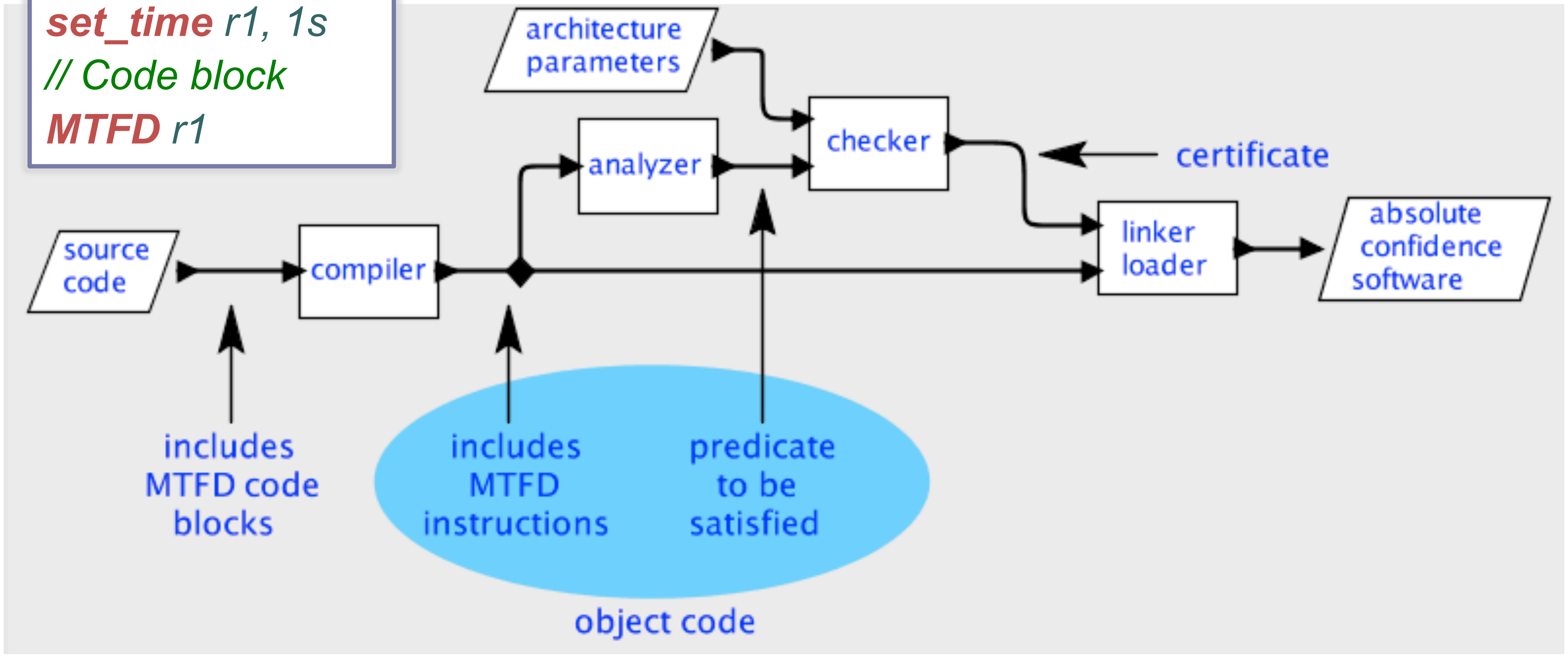
ISA that admits a variety of implementations:

- 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

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


architecture parameters
checker
certificate
analyzer
source code
compiler
linker loader
absolute confidence software
includes MTFD code blocks
includes MTFD instructions
predicate to be satisfied
object code

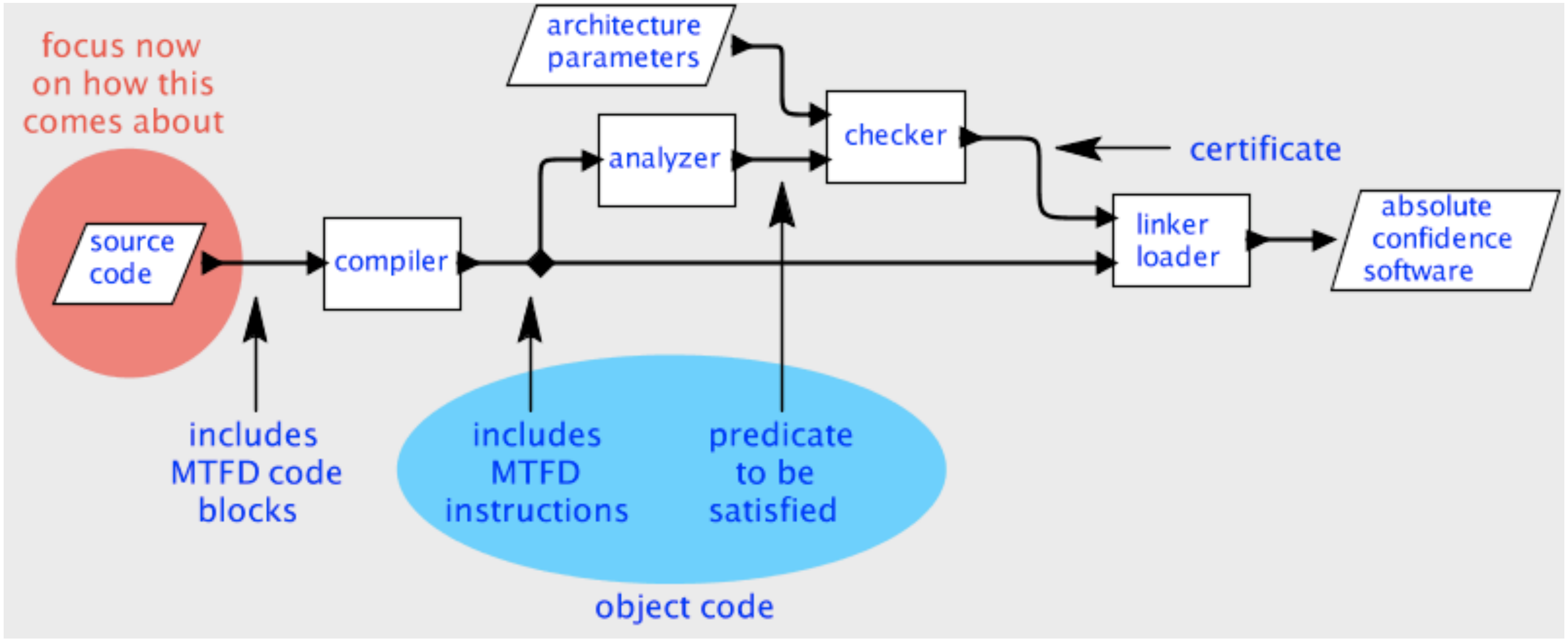
The goal is to make software that will run correctly on a variety of implementations of the ISA, and that correctness can be checked for each implementation.

PRET Publications

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

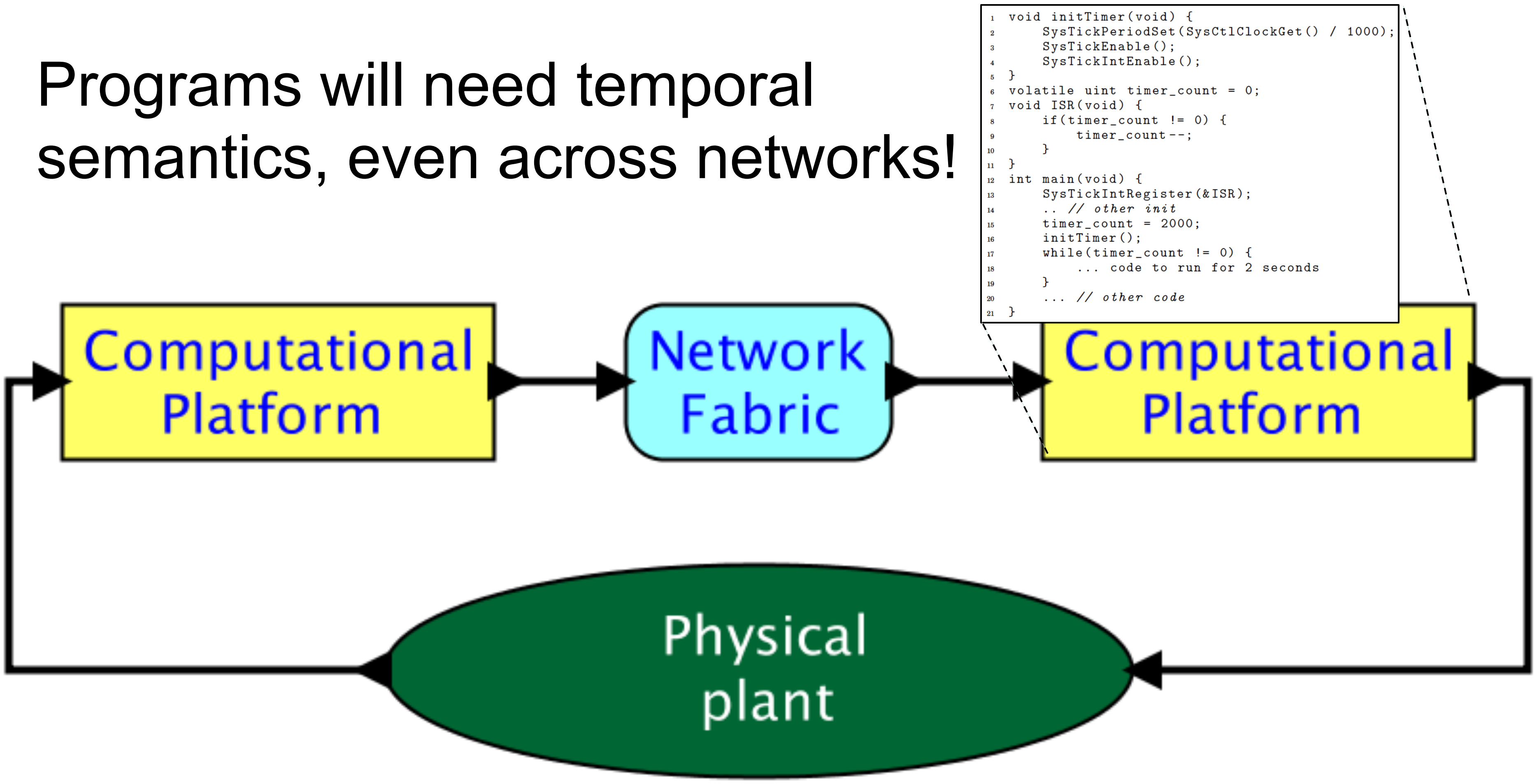
- S. Edwards and E. A. Lee, "**The Case for the Precision Timed (PRET) Machine**," in the *Wild and Crazy Ideas* Track of DAC, June 2007.
- B. Lickly, I. Liu, S. Kim, H. D. Patel, S. A. Edwards and E. A. Lee, "**Predictable programming on a precision timed architecture**," CASES 2008.
- S. Edwards, S. 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**, 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, EMSOFT, Taiwan, October, 2011

Part 2: How to get the Source Code?


focus now on how this comes about
architecture parameters
checker
certificate
analyzer
source code
compiler
linker loader
absolute confidence software
includes MTFD code blocks
includes MTFD instructions
predicate to be satisfied
object 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.

Programs will need temporal semantics, even across networks!



```
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
}
```

Our premise: Synchronizing clocks across a network at high precision is becoming reality.

How does this change the design of distributed real-time software?

A Time Synchronization Revolution
Happening now!

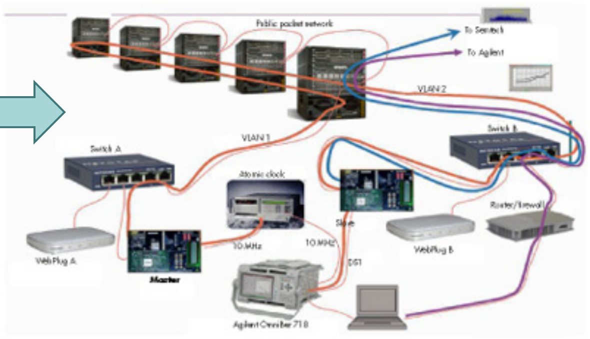
Time synchronization is going to change the world (again)

Gregorian Calendar (BBC history)

1500s
days

Lackawanna Railroad Station, 1907, Hoboken. Photograph by Alicia Dudek

1800s
seconds



2005: first IEEE 1588 plugfest

2000s
nanoseconds

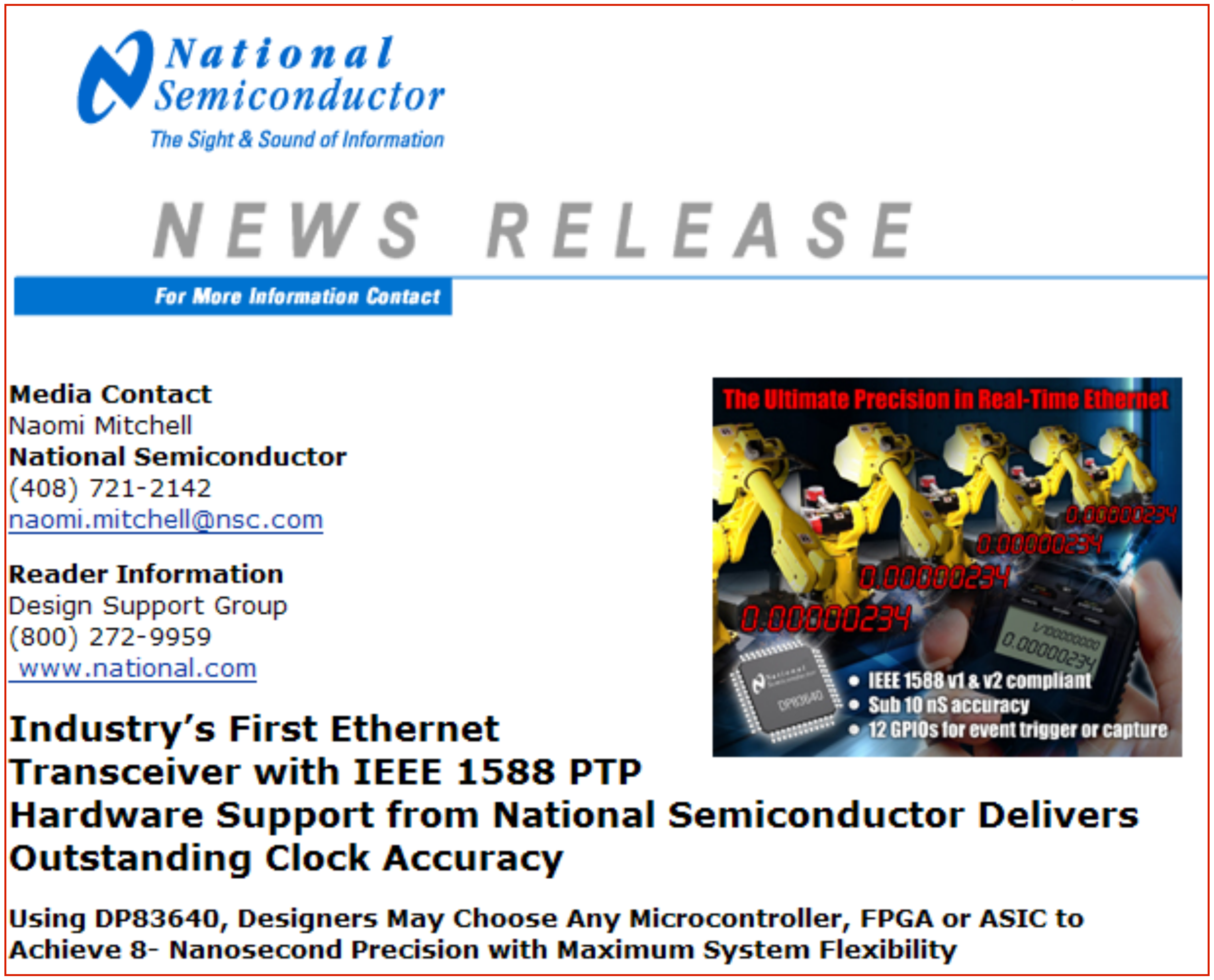
Today’s networks

“On August 12, 1853, two trains on the Providence & Worcester Railroad were headed toward each other on a single track. The conductor of one train thought there was time to reach the switch to a track to Boston before the approaching train was scheduled to pass through. But the conductor's watch was slow. As his speeding train rounded a blind curve, it collided head-on with the other train—fourteen people were killed. The public was outraged. All over New England, railroads ordered more reliable watches for their conductors and issued stricter rules for running on time.”

Source: National Museum of American History

Precision Time Protocols (PTP) IEEE 1588 on Ethernet

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

The Ultimate Precision in Real-Time Ethernet
0.00000234
0.00000234
0.00000234
0.00000234
0.00000234
IEEE 1588 v1 & v2 compliant
Sub 10 nS accuracy
12 GPIOs for event trigger or capture

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 is becoming 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?

A Cyber-Physical System
Printing Press

Bosch-Rexroth

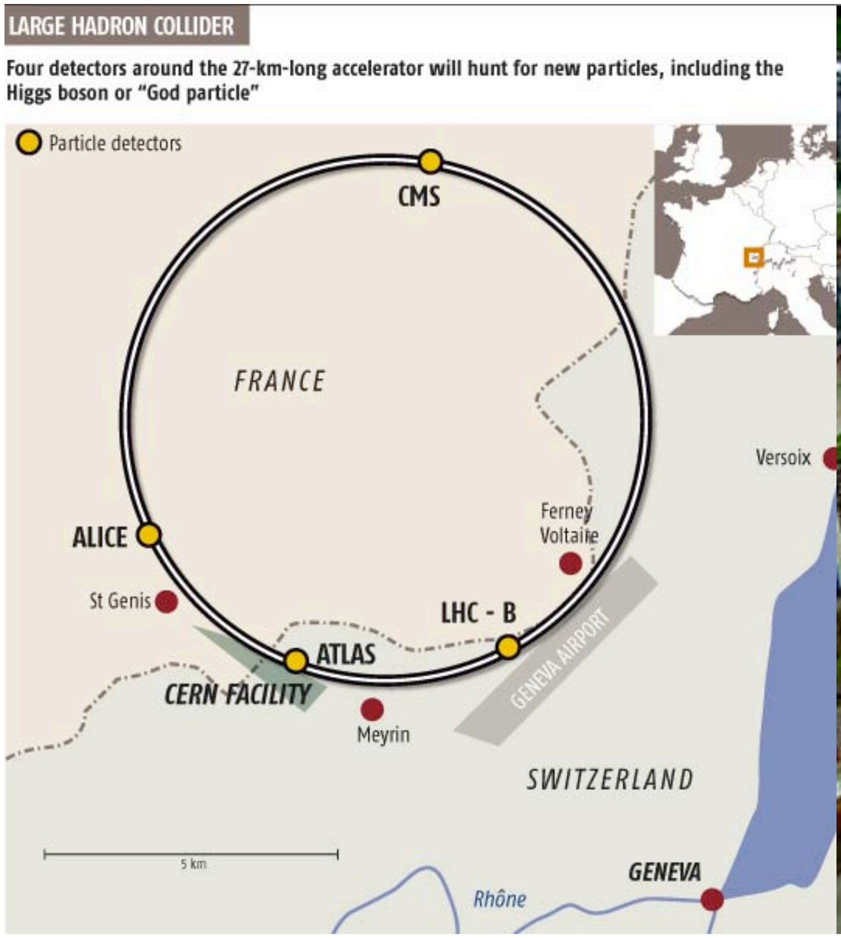
Hundreds of microcontrollers and an Ethernet network are orchestrated with precisions on the order of microseconds.

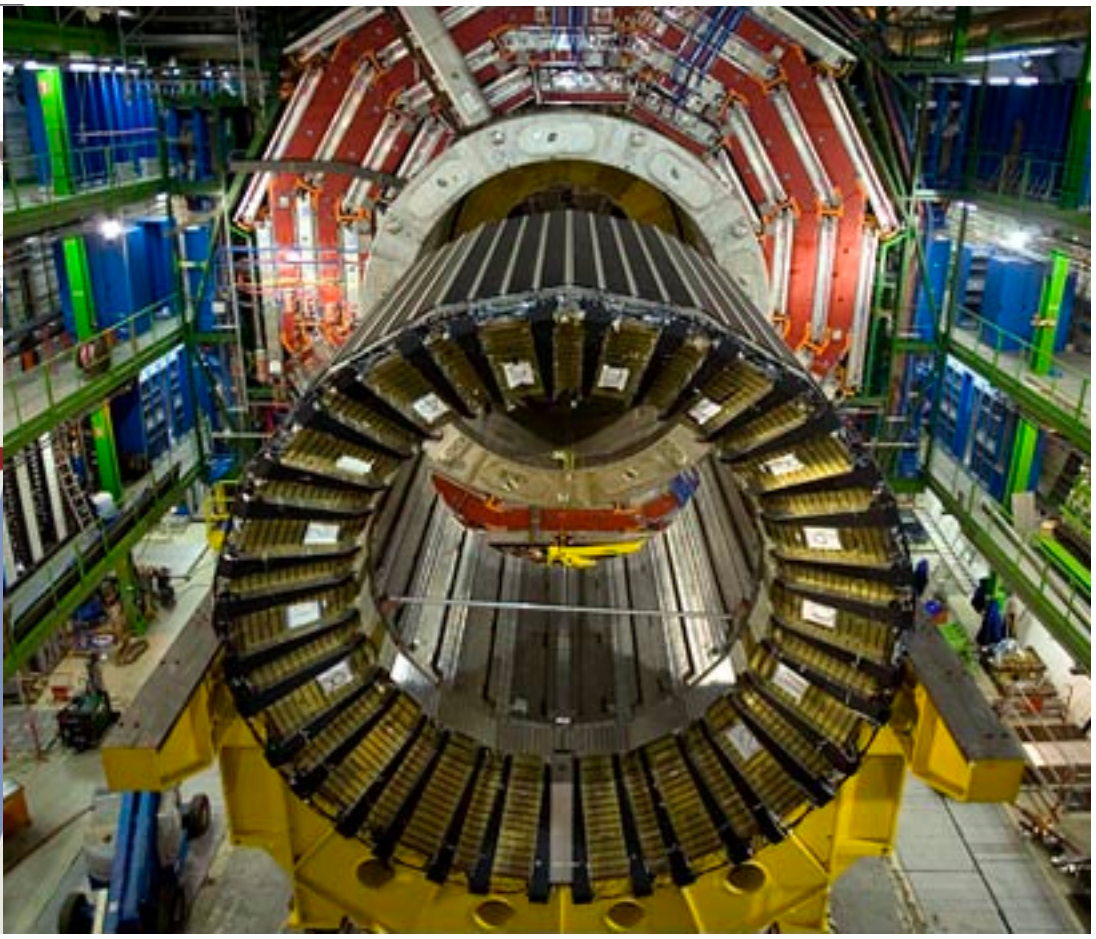
Software for such systems can be developed in a completely new way.

Time synchronization enables tightly coordinated actions and reliable networking with bounded latency.

An Extreme Example: The Large Hadron Collider

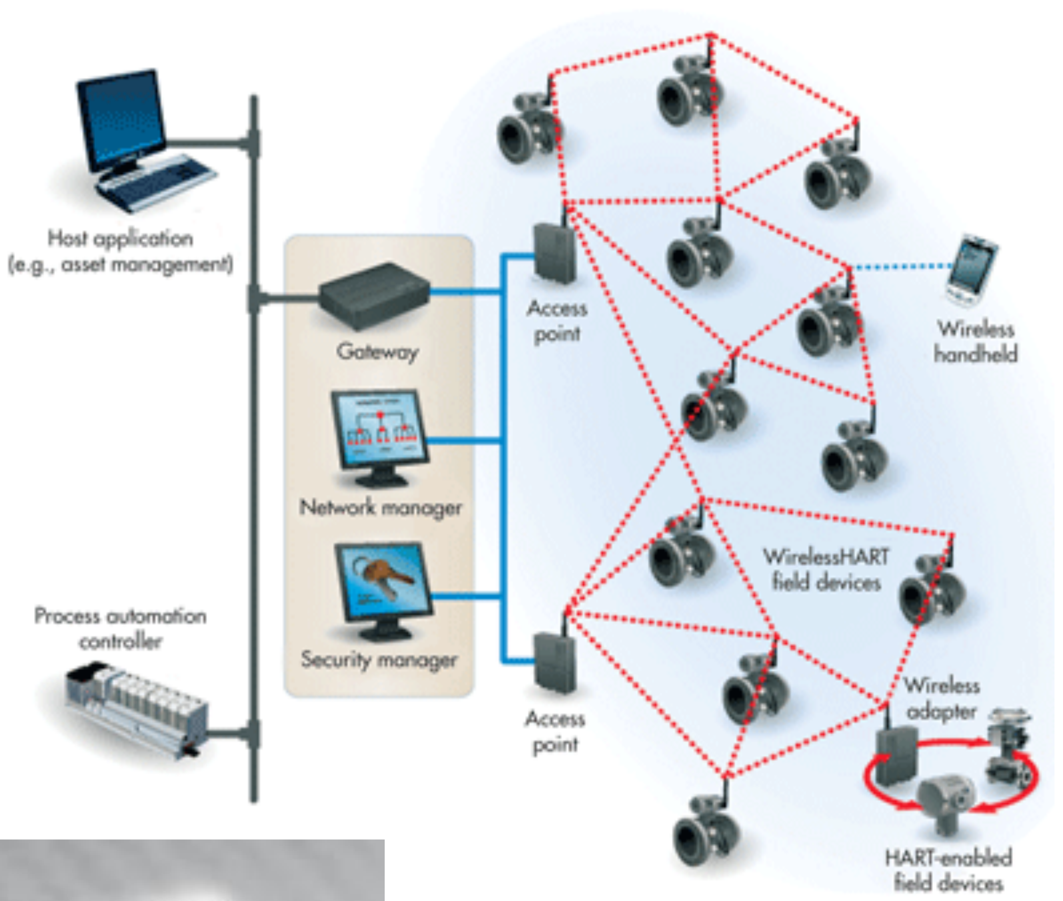
The WhiteRabbit project at CERN is synchronizing the clocks of computers 10 km apart to within about 80 psec using a combination of IEEE 1588 PTP and synchronous ethernet.

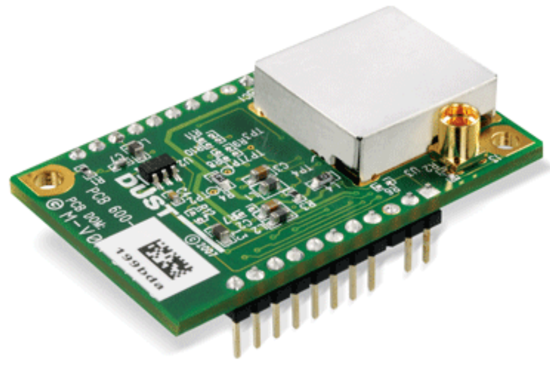




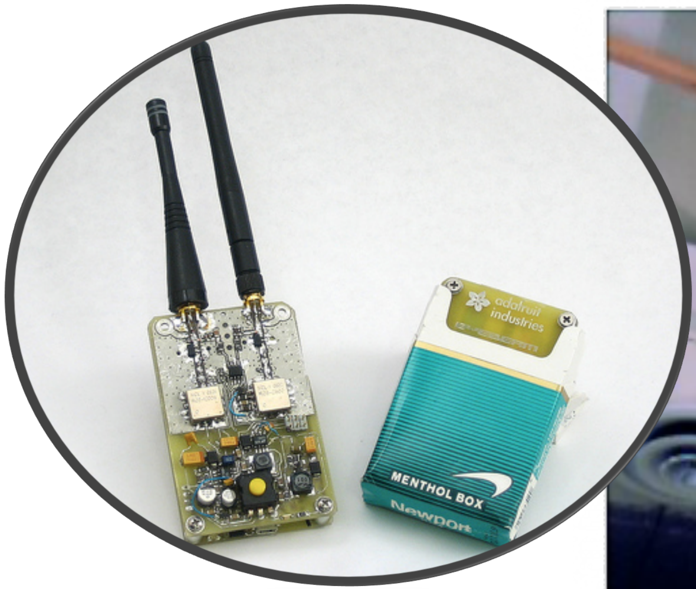
Wireless

Wireless HART uses Time Synchronized Mesh Protocol (TSMP) in a Mote-on-Chip (MoC), from Dust Networks Inc.





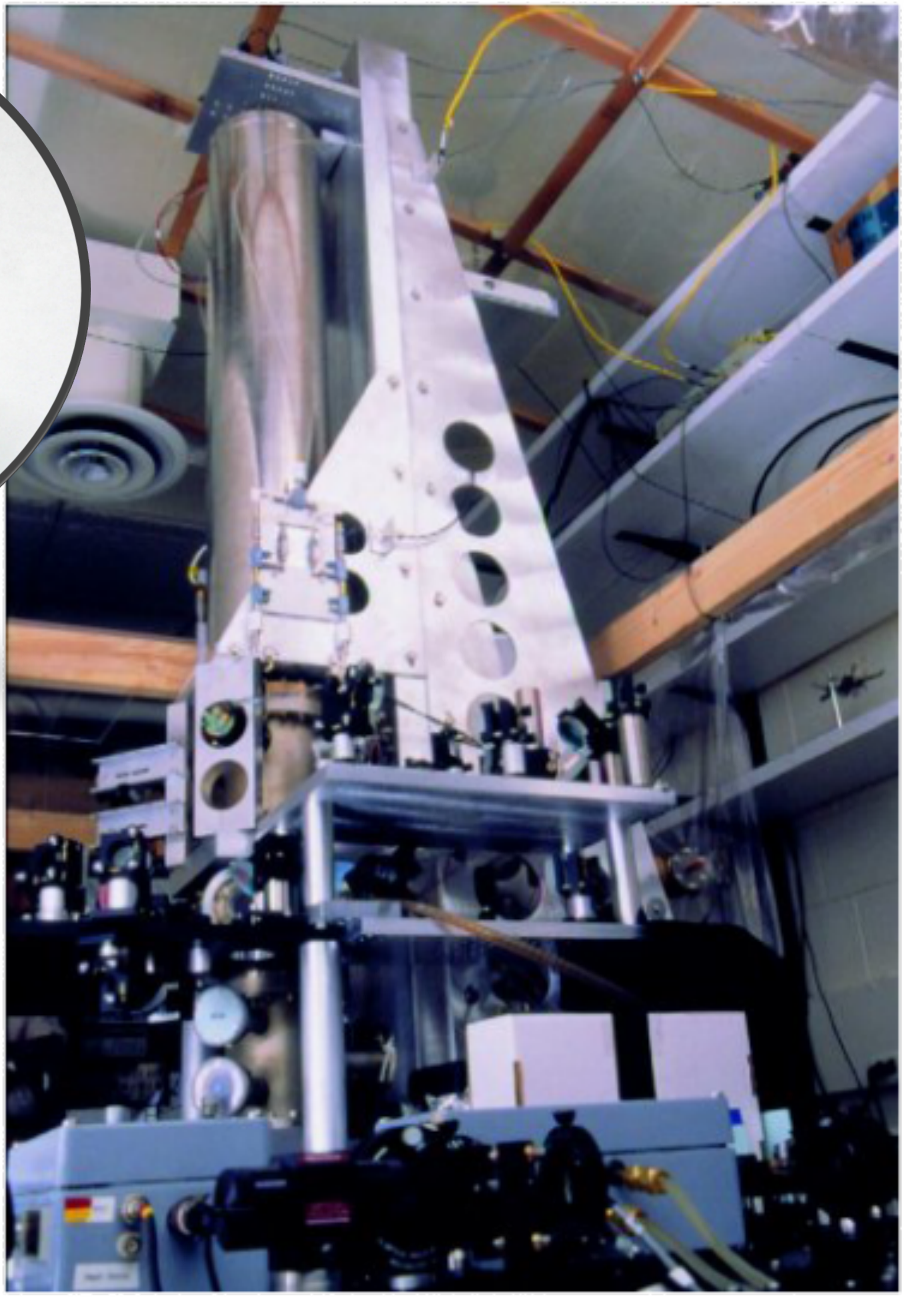
Security



GPS Jammer, courtesy of Kyle D. Wesson, UT Austin

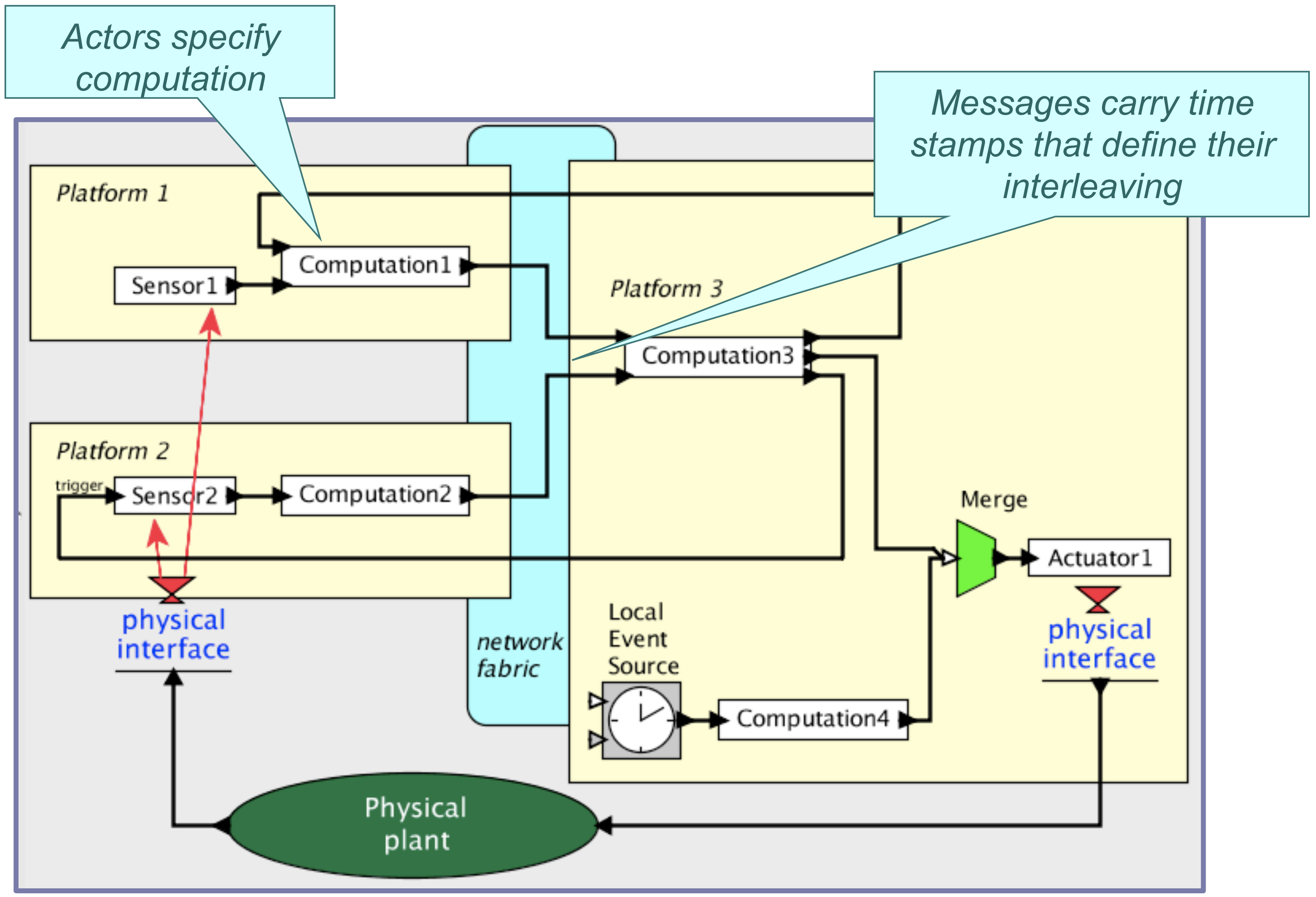
With stable local clocks you can:

- Prevent packet losses.
- Detect hardware failures.
- Detect denial of service.
- Detect GPS and PTP spoofing.
- Coordinate w/out communication.

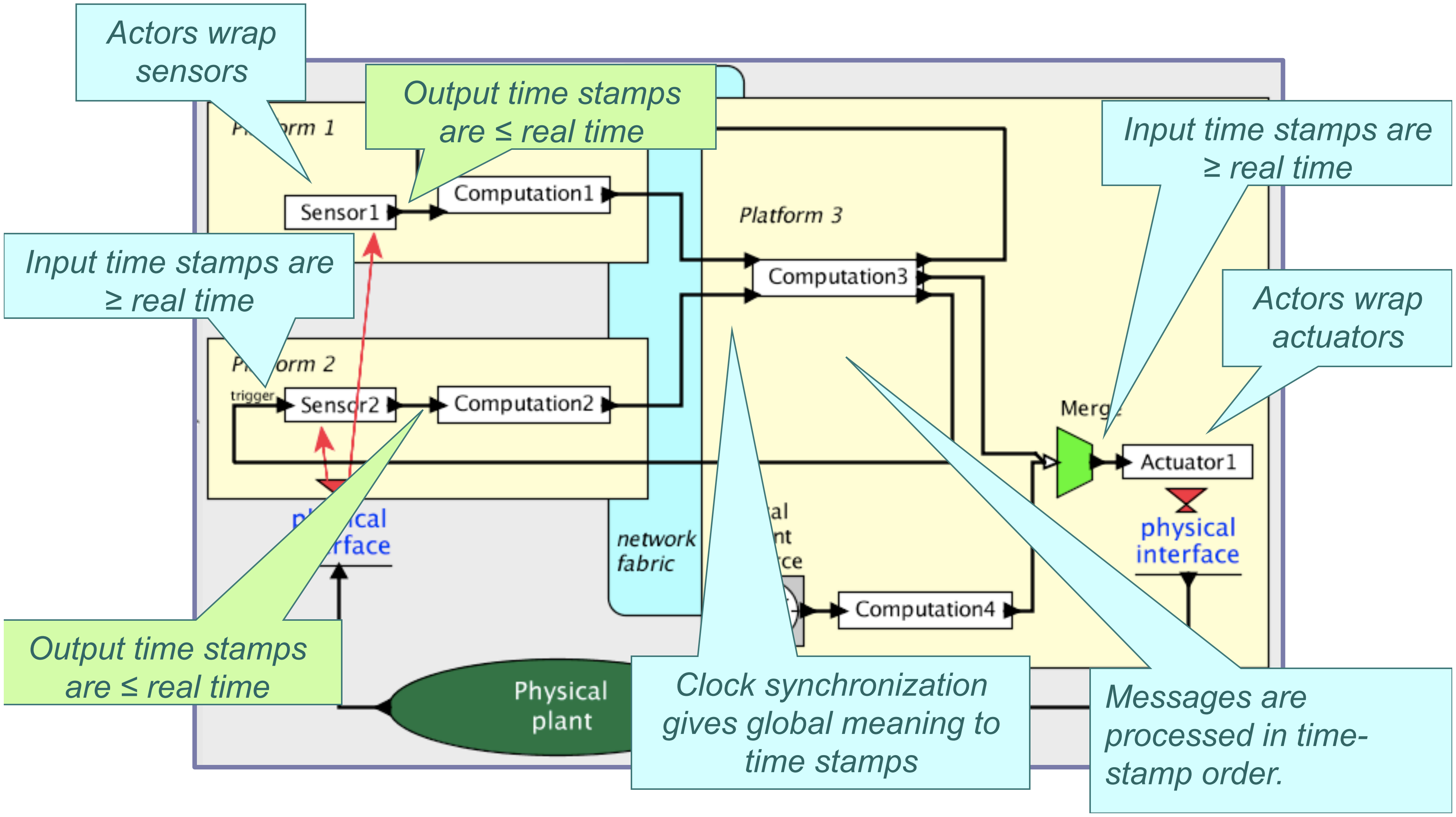


NIST-F1 Atomic Clock Public Doman Photo

Ptides: Programming Temporally Integrated Distributed Embedded Systems
Time-stamped messages.



Ptides:
Bind time stamps to real time at sensors and actuators



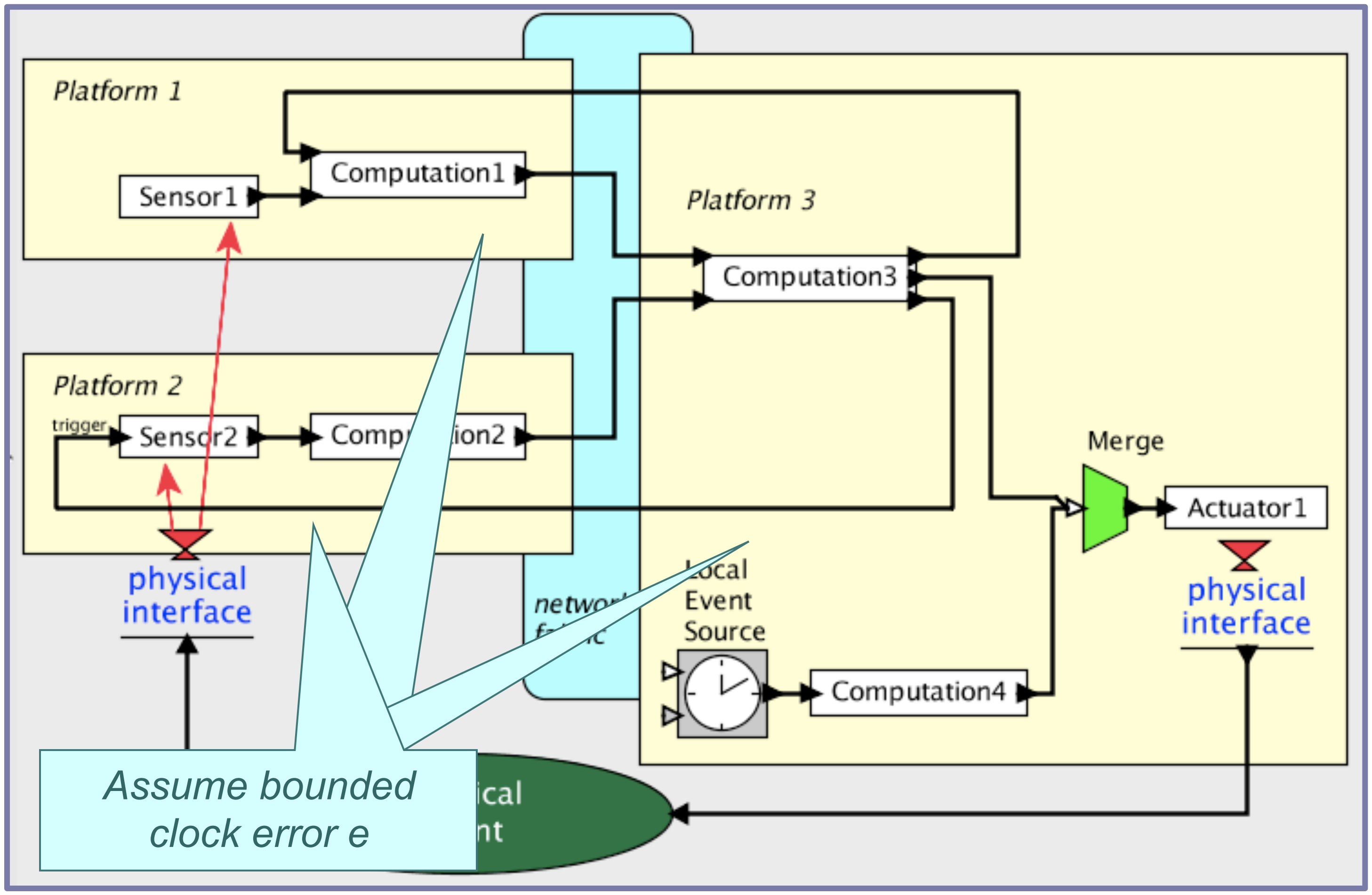
Actors wrap sensors
Output time stamps are ≤ real time
Input time stamps are ≥ real time
Actors wrap actuators
Input time stamps are ≥ real time
Output time stamps are ≤ real time
Clock synchronization gives global meaning to time stamps
Messages are processed in time-stamp order.
Platform 1
Sensor1
Computation1
Platform 3
Computation3
Platform 2
trigger
Sensor2
Computation2
Merge
Actuator1
physical interface
physical interface
network fabric
Computation4
Physical plant

Principles of the Model

- Assume bounded time synchronization error
  - Gives global meaning to time stamps.
  - Violations are detectable (when they matter).
- Assume bounded network latency
  - Ensures visibility of remote events.
  - Violations are detectable (when they matter).
- Achieve determinate distributed computation
  - Given time-stamped input events, the model will always produce the same time stamped output events (with above assumptions).

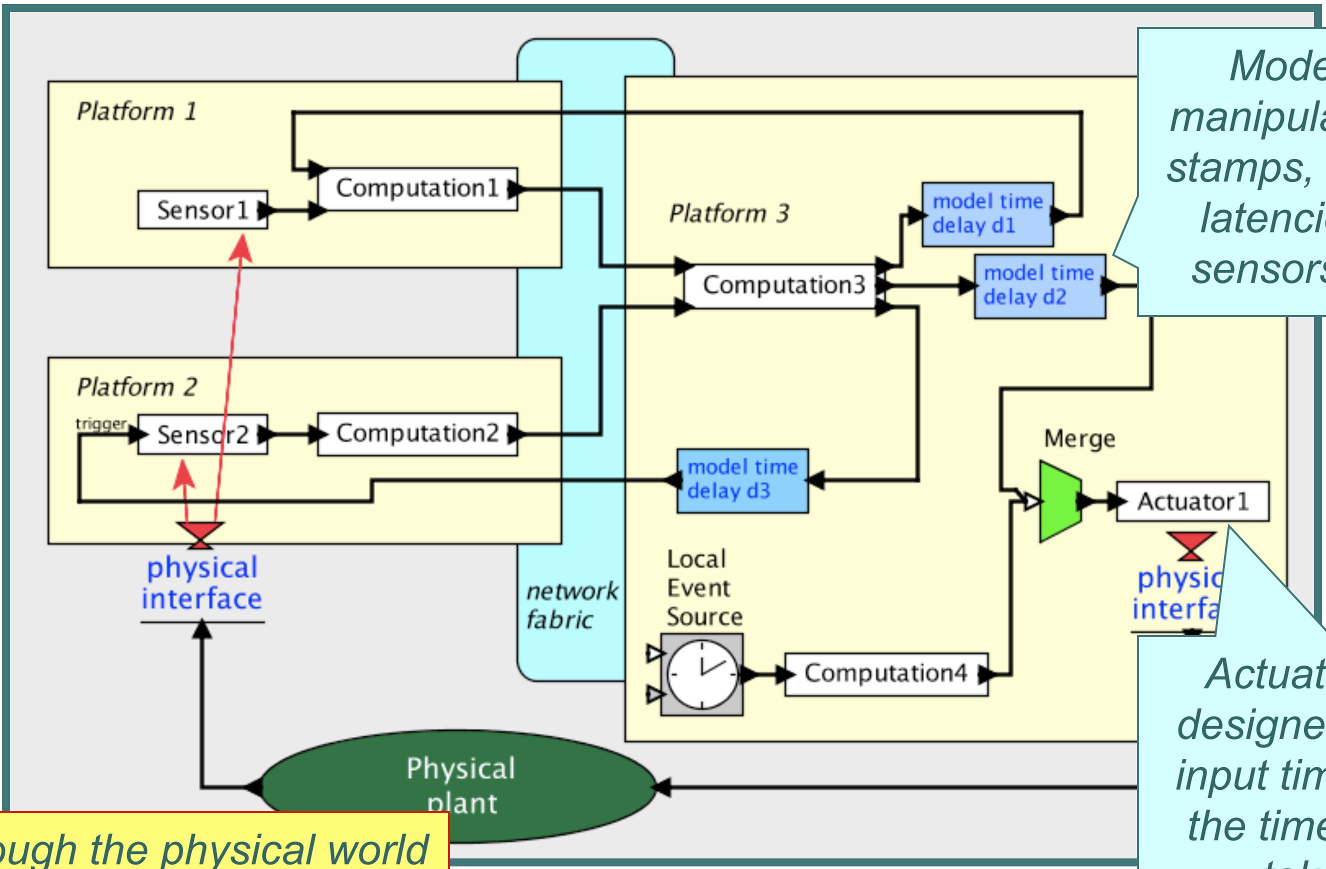
Ptides: Second step: Network time synchronization

GPS, NTP, IEEE 1588, time-triggered busses, … they all work. We just need to bound the clock synchronization error.



Ptides: Specify latencies in the model

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



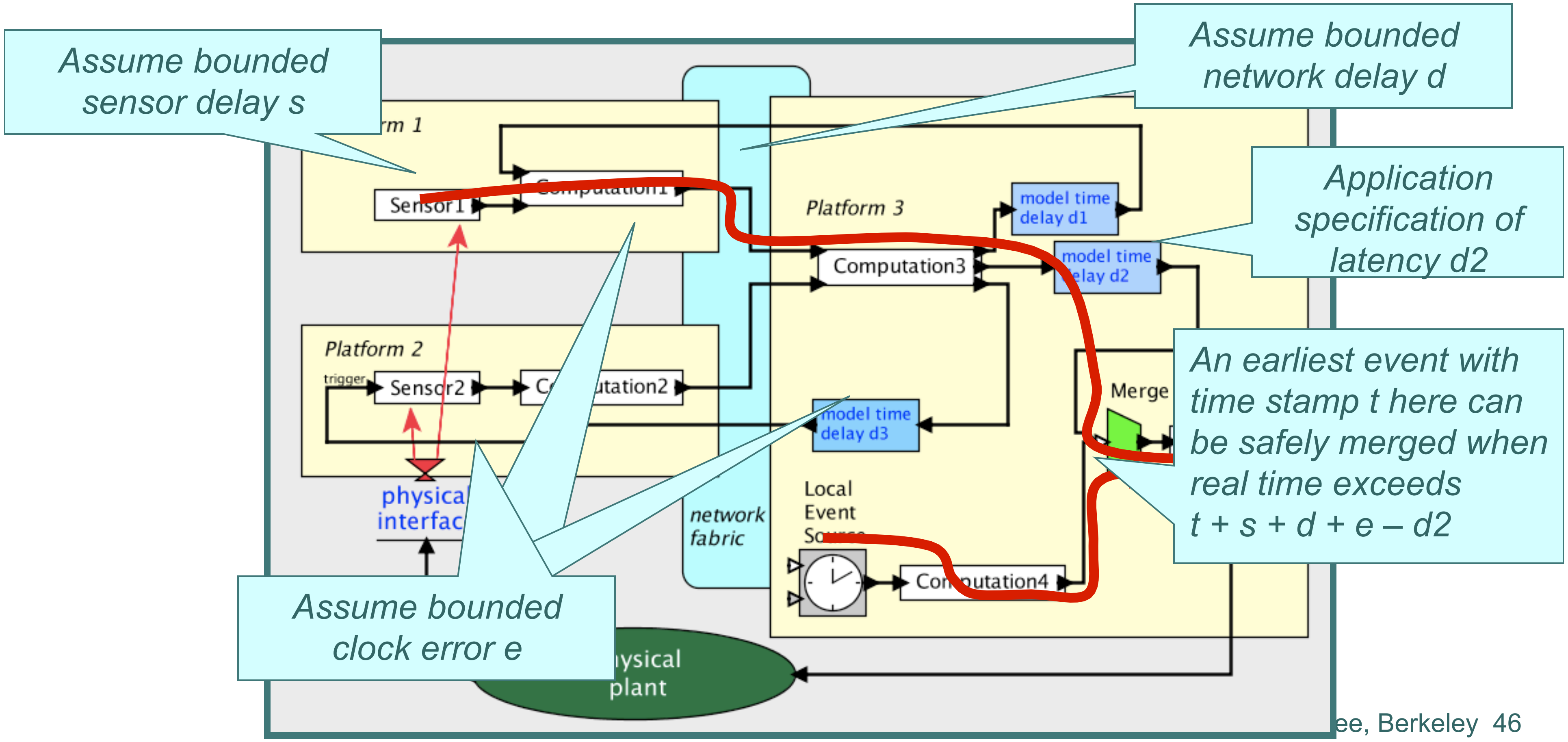
Model includes manipulations of time stamps, which control latencies between sensors and actors

Actuators may be designed to interpret input time stamps as the time at which to take action.

Feedback through the physical world

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.

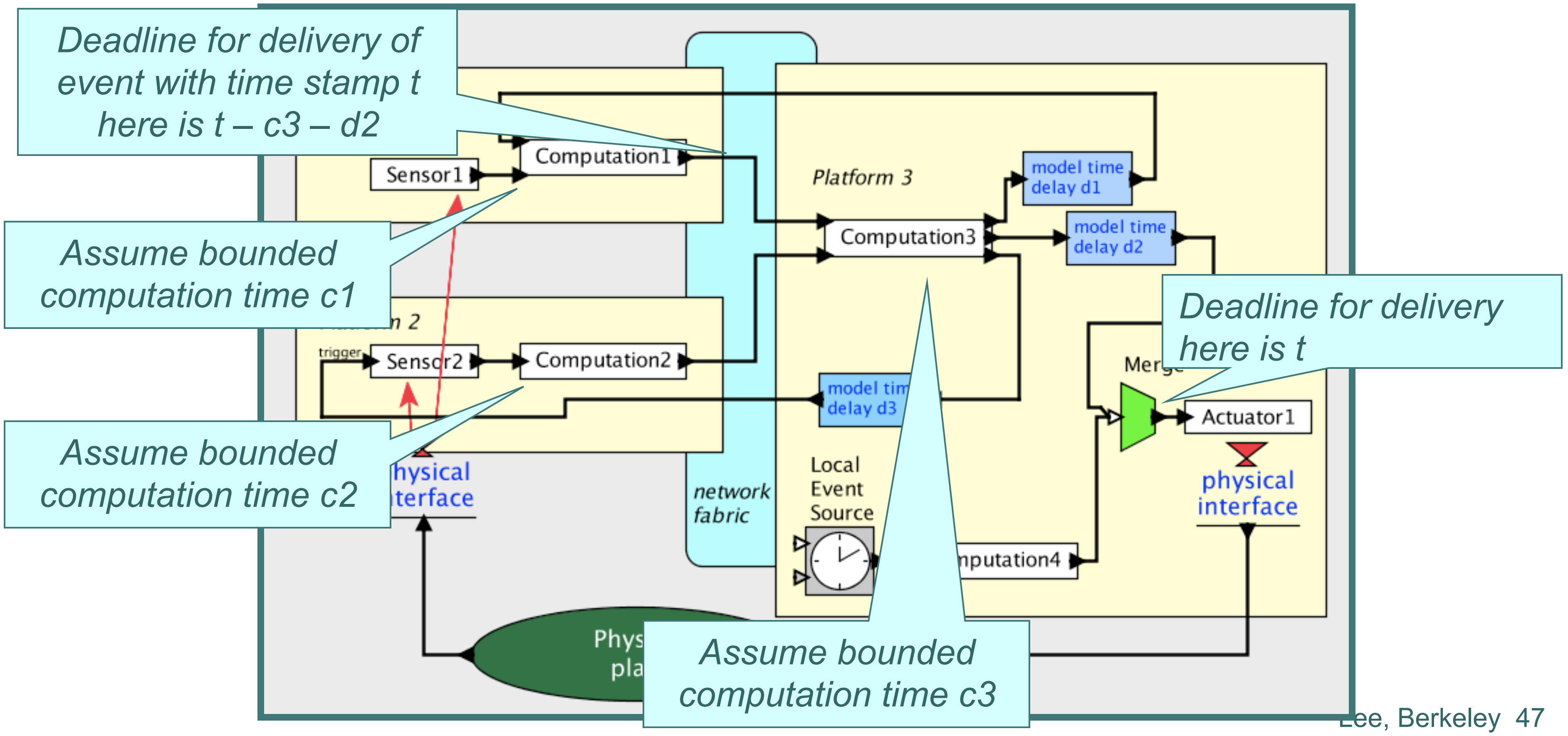


Assume bounded sensor delay s
Assume bounded network delay d
Application specification of latency d2
An earliest event with time stamp t here can be safely merged when real time exceeds t + s + d + e – d2
Assume bounded clock error e
Platform 1
Sensor1
Computation1
Platform 2
trigger
Sensor2
Computation2
physical interface
Platform 3
model time delay d1
model time delay d2
model time delay d3
Computation3
Merge
network fabric
Local Event Source
Computation4
physical plant

Ptides Schedulability Analysis

Determine *whether* deadlines can be met

Schedulability analysis incorporates computation times to determine whether we can guarantee that deadlines are met. If execution times of components are bounded, this turns out to be decidable!



Conclusions

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.*

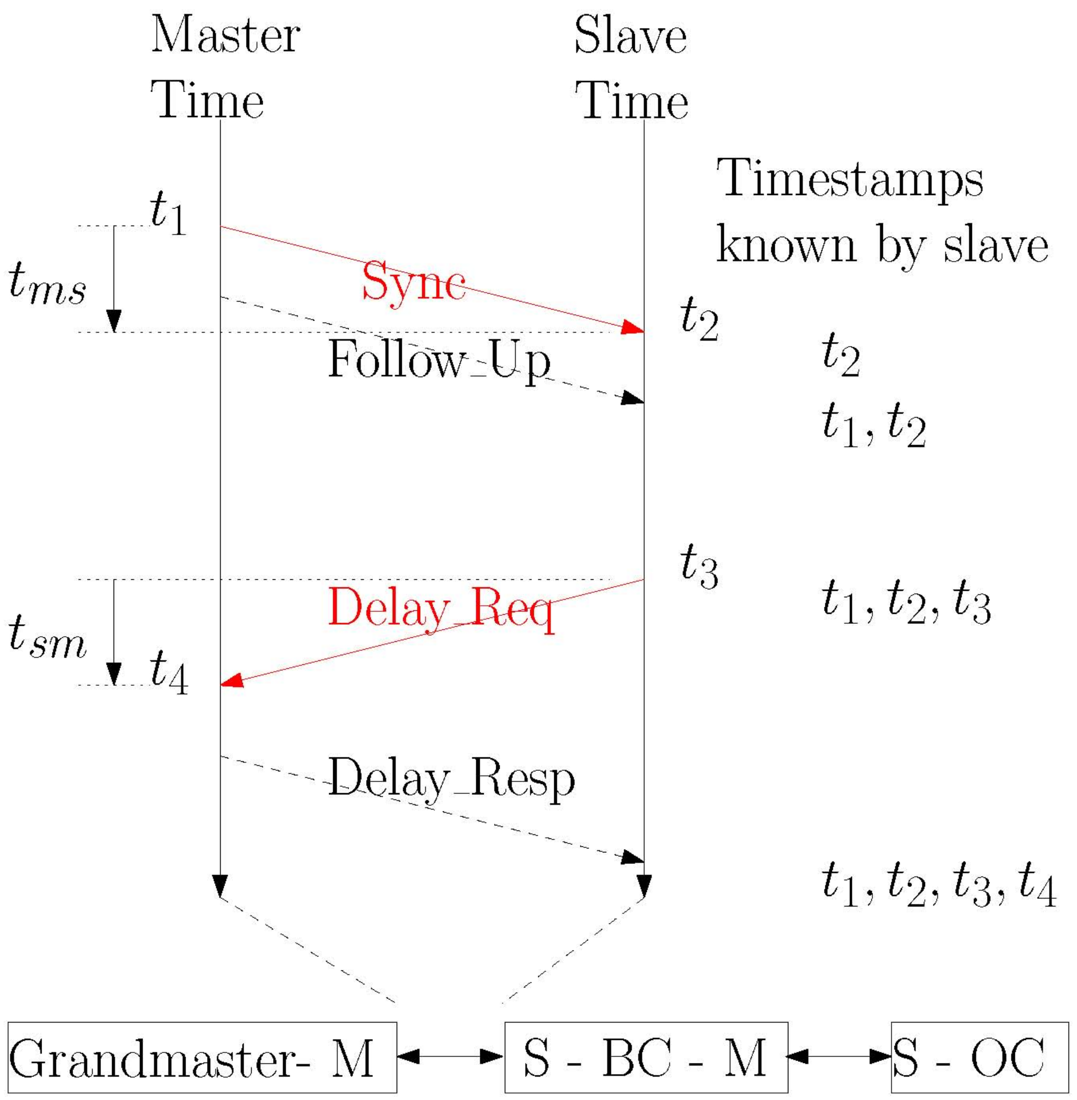
Today, timing emerges from *realizations* of systems.

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

Raffaello Sanzio da Urbino – The Athens School

Thank you!

How PTP Synchronization works



If link is symmetric:

$$\mathit{Offset} = t_{slave} - t_{master} = [(t_2 - t_1) - (t_4 - t_3)]/2 = [t_{ms} - t_{sm}]/2$$

$$\mathit{Propagation\ time} = [(t_2 - t_1) + (t_4 - t_3)]/2 = [t_{ms} + t_{sm}]/2$$