Precision Timed Microprocessors

Edward A. Lee

Professor of the Graduate School

Systèmes embarqués et traitement de l'information (SETI)

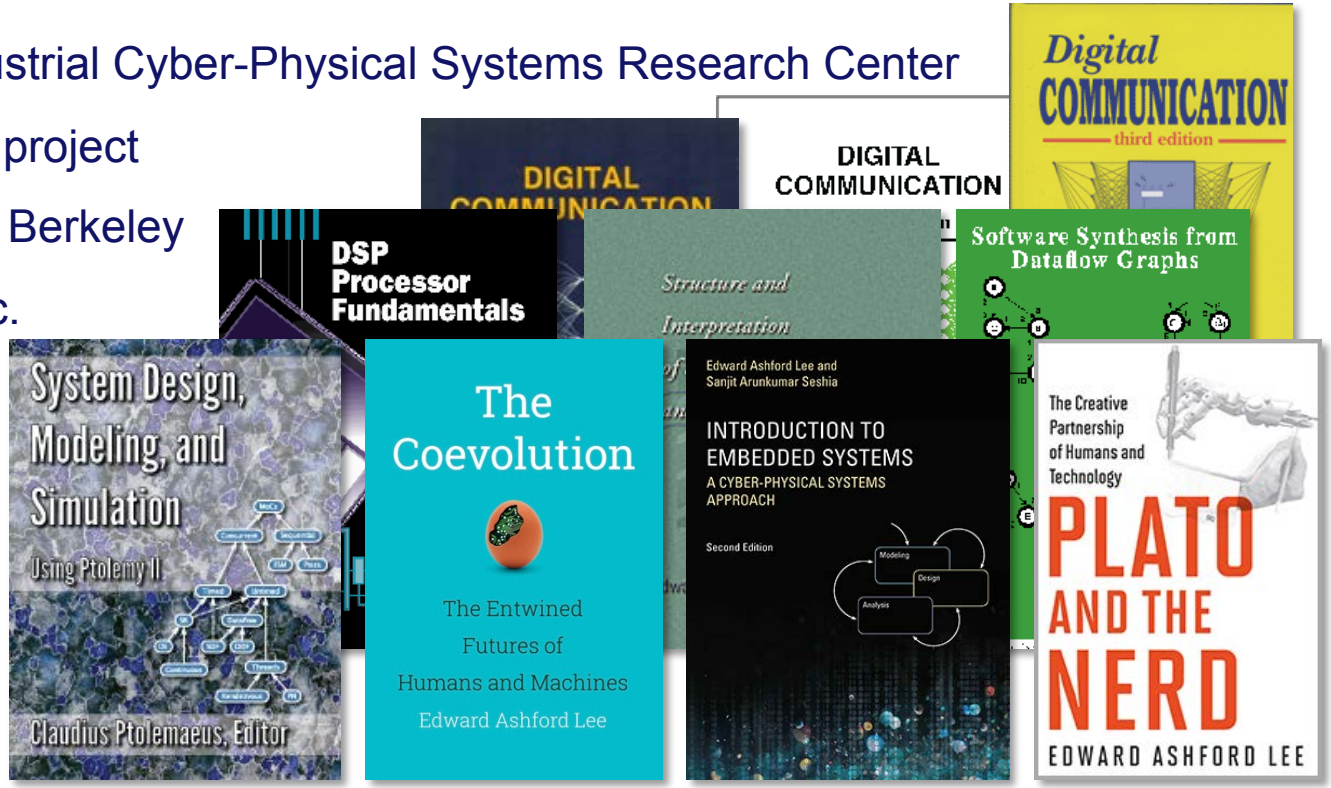
Université Paris-Saclay

Saclay, France, January 24, 2020

University of California at Berkeley

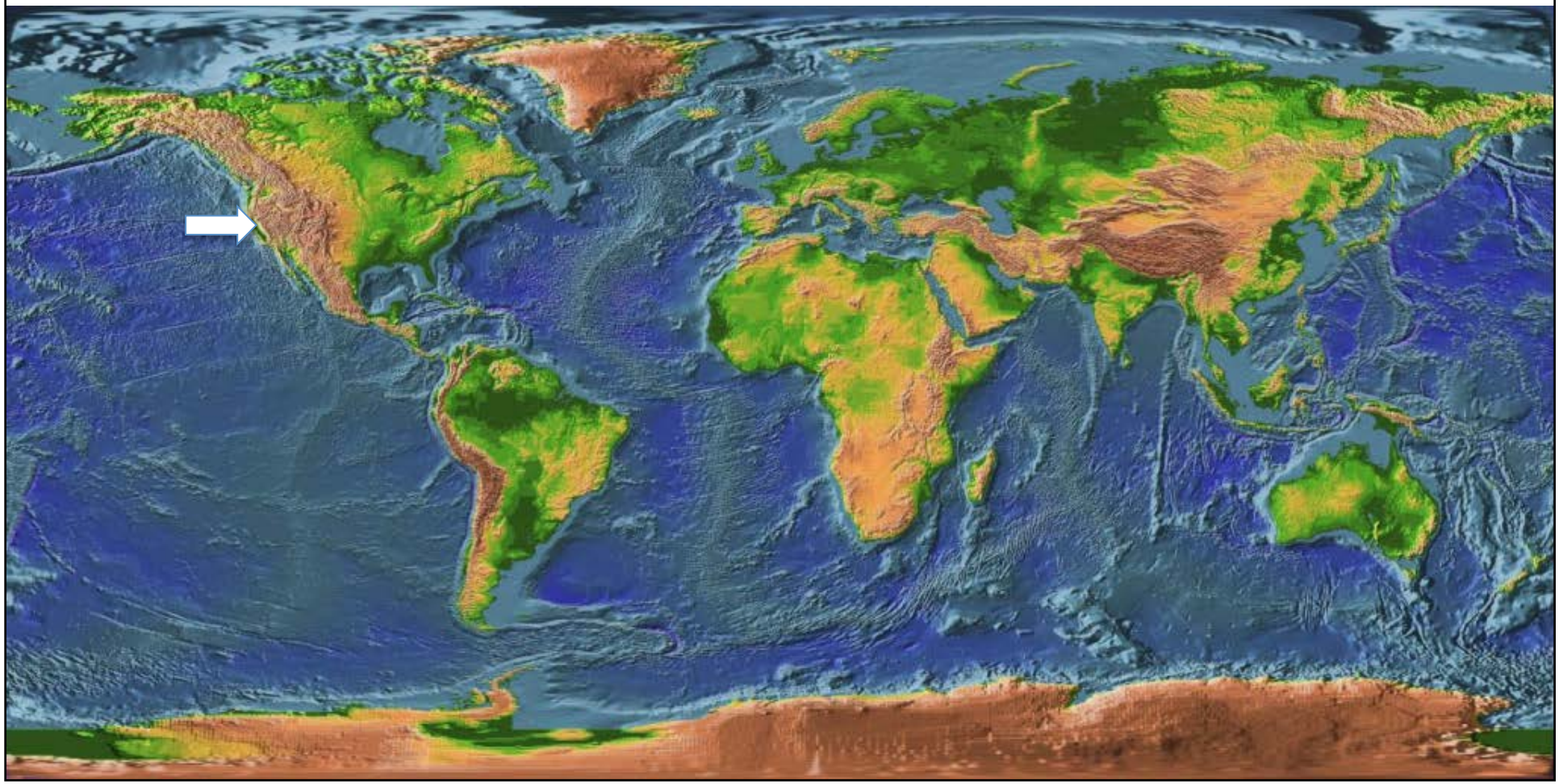
Introducing Edward A. Lee

- BS (Yale), SM (MIT), PhD (Berkeley)
- Bell Labs in the early 1980s
- Berkeley EECS faculty since 1986
- Working on embedded software since 1978
- Director of iCyPhy, Industrial Cyber-Physical Systems Research Center
- Director of the Ptolemy project
- Former Chair of EECS, Berkeley
- Co-founder of BDTI, Inc.
- Books…



http://ptolemy.org/~eal
eal@berkeley.edu

Location



The University of California at Berkeley

Disclaimer

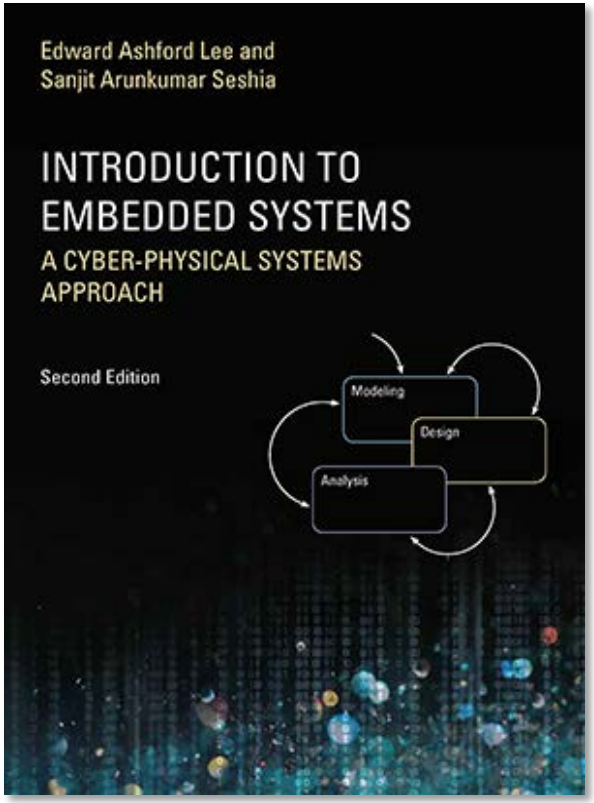
This is not a survey of the field.

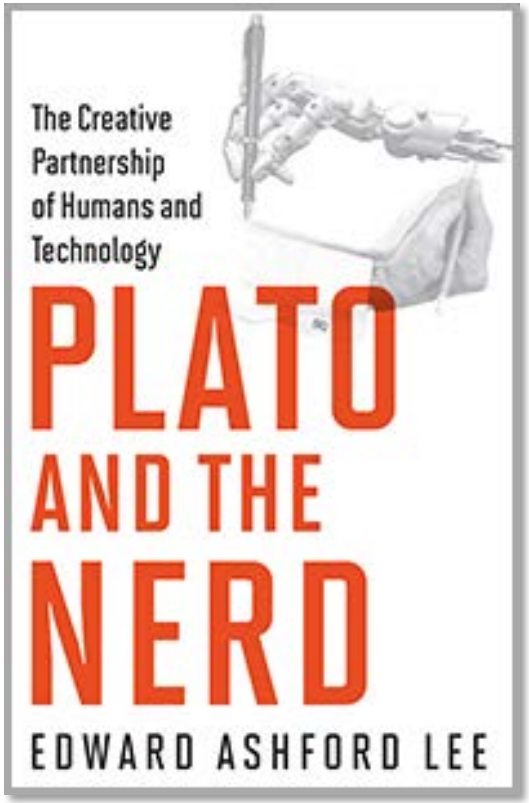
I will give you a narrow Berkeley view with a lot of opinions and personal perspectives.

Key References

https://ptolemy.berkeley.edu/projects/chess/pret/

&





These slides:
http://ptolemy.org/~eal/presentations/Lee_PrecisionTimedMicroprocessors_Saclay.pdf

These slides

http://ptolemy.org/~eal/presentations/Lee_PrecisionTimedMicroprocessors_Saclay.pdf

Outline

- Cyber-Physical Systems
- Real Time
- Timing in Software
- Science and Engineering
- Precision Timed Machines
- Mixed Criticality and FlexPRET
- I/O and Interrupts
- Cost and Performance
- Programming Models
- Parametric PRET Machines

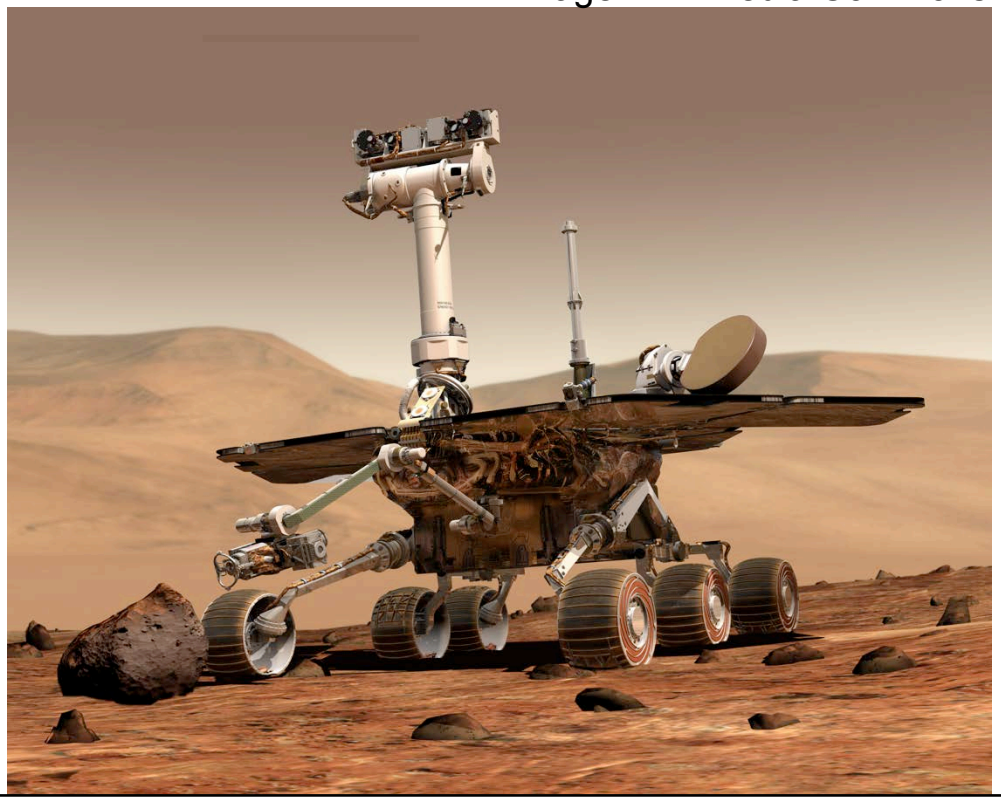
Cyber-Physical Systems

Orchestrating networked computational resources and physical systems.

Roots:

- Term coined around 2006 by Helen Gill at the National Science Foundation in the US.
- **Cyberspace**: attributed William Gibson, who used the term in the novel Neuromancer.
- **Cybernetics**: coined by Norbert Wiener in 1948, to mean the conjunction of control and communication.

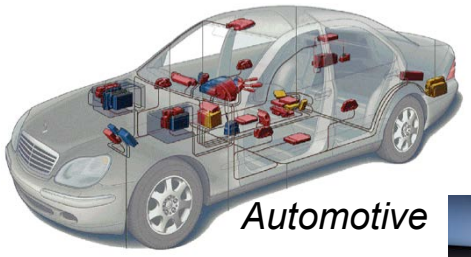
Image: Wikimedia Commons



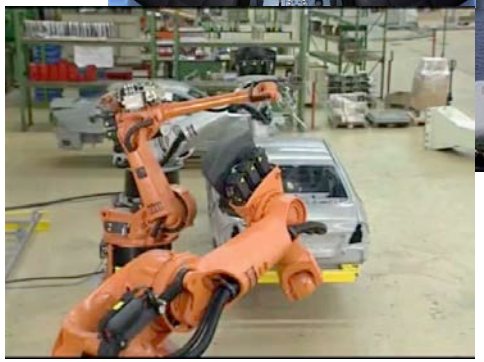
Cyber-Physical Systems

Not just information technology:

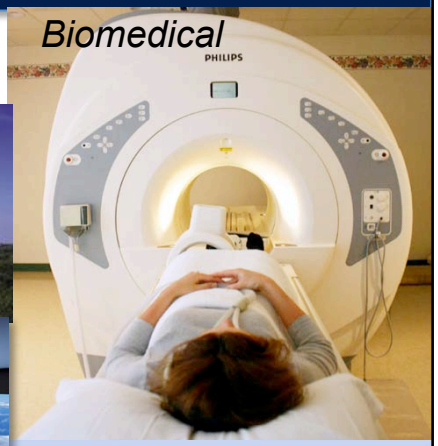
- Cyber + Physical
- Computation + Dynamics
- Security + Safety

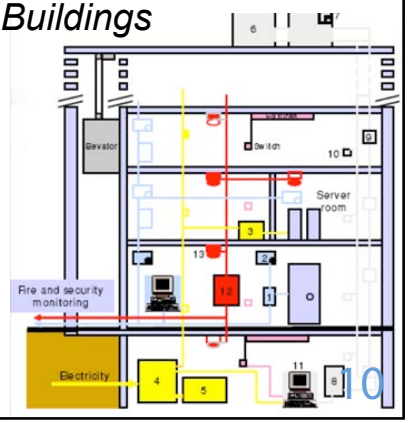


Automotive



Manufacturing





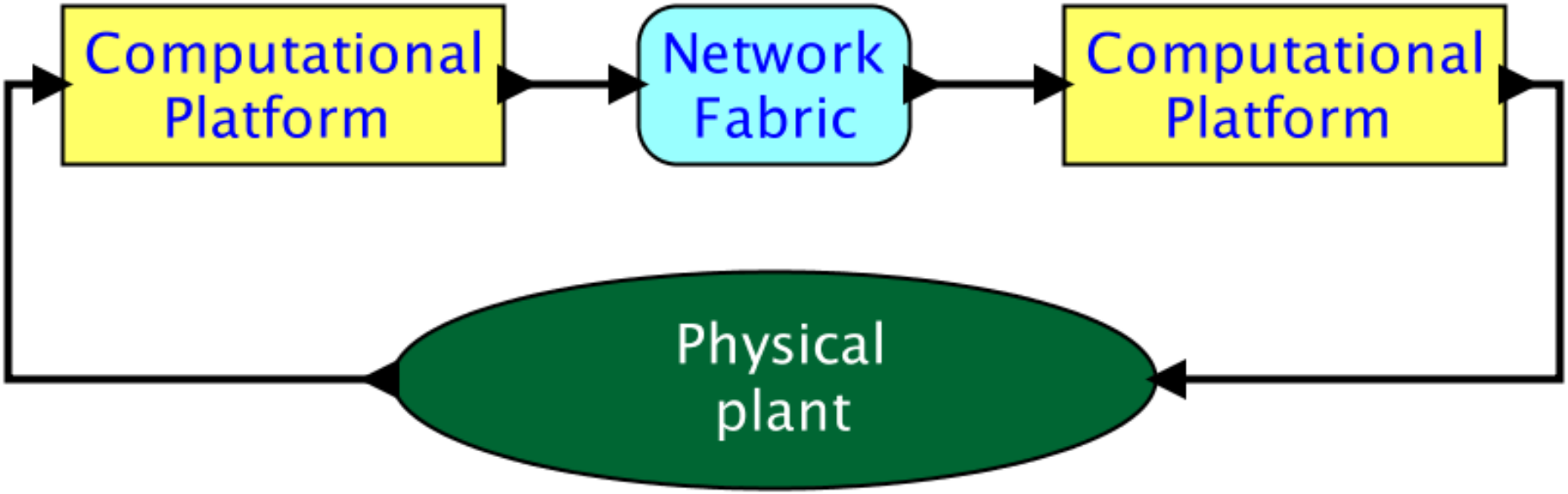
Properties:

- Highly dynamic
- Safety critical
- Uncertain environment
- Physically distributed
- Sporadic connectivity
- Resource constrained

We need engineering models and methodologies for dependable cyber-physical systems.

Lee, Berkeley

Cyber Physical Systems



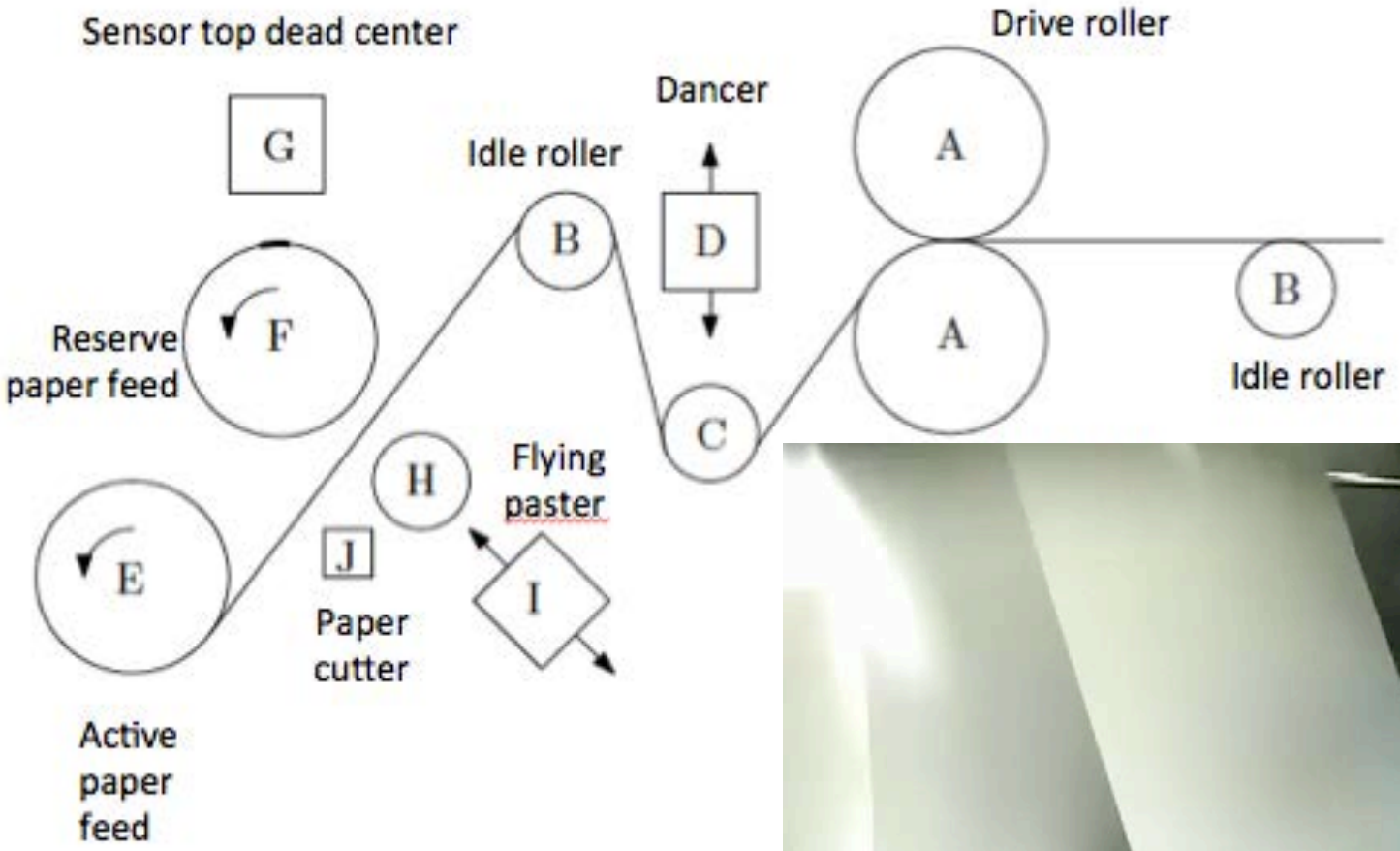
Predictability requires determinacy and depends on timing, including execution times and network delays.

Example

Hundreds of microcontrollers orchestrating depositing ink and slicing paper flying through the machine at 100 km/hr.

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

Example – Flying Paster

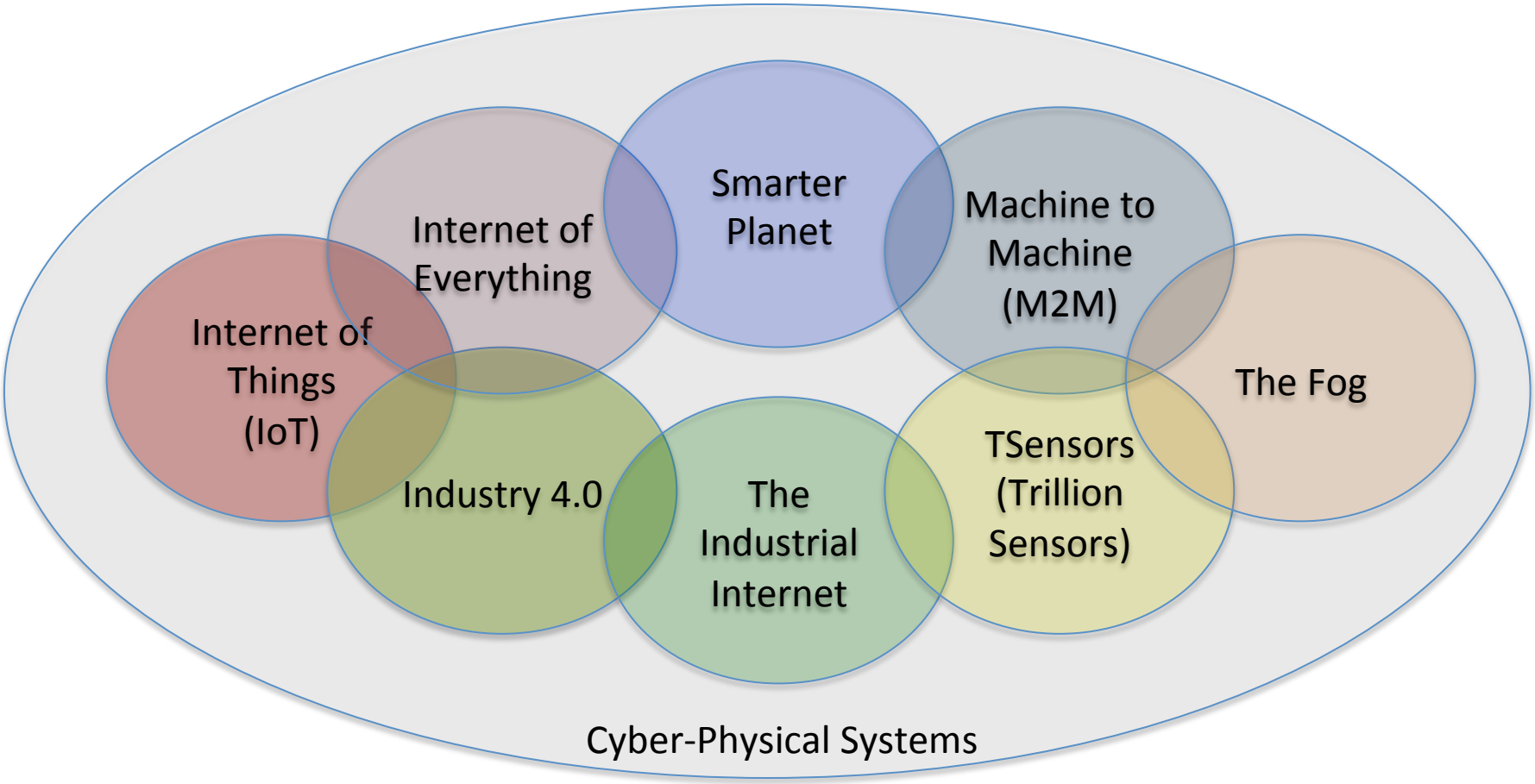


Source: http://offsetpressman.blogspot.com/2011/03/how-flying-paster-works.htr

Example – Flying Paster

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

CPS-related terms



Outline

- Cyber-Physical Systems
- Real Time
- Timing in Software
- Science and Engineering
- Precision Timed Machines
- Mixed Criticality and FlexPRET
- I/O and Interrupts
- Cost and Performance
- Programming Models
- Parametric PRET Machines

What is Real Time?

- fast computation
- prioritized scheduling
- computation on streaming data
- bounded execution time
- temporal semantics in programs
- temporal semantics in networks

What is Real Time?

- fast computation
- prioritized scheduling
- computation on streaming data
- bounded execution time
- temporal semantics in programs
- temporal semantics in networks

These are very different from one another.
We have to decide which to focus on.

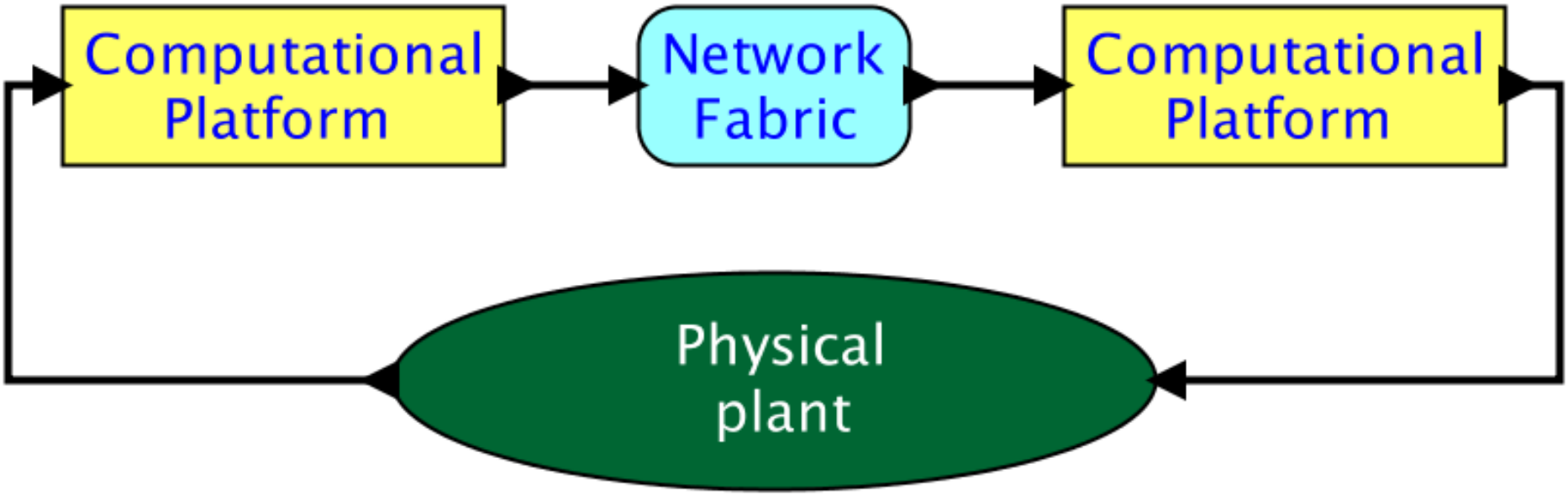
Achieving Real Time

- overengineering
- using old technology
- response-time analysis
- real-time operating systems (RTOSs)
- specialized networks
- extensive testing and validation

Outline

- Cyber-Physical Systems
- Real Time
- Timing in Software
- Science and Engineering
- Precision Timed Machines
- Mixed Criticality and FlexPRET
- I/O and Interrupts
- Cost and Performance
- Programming Models
- Parametric PRET Machines

Schematic of a simple CPS:



Computation given in an untimed, imperative language. Physical plant modeled with ODEs or DAEs

```
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
}
```

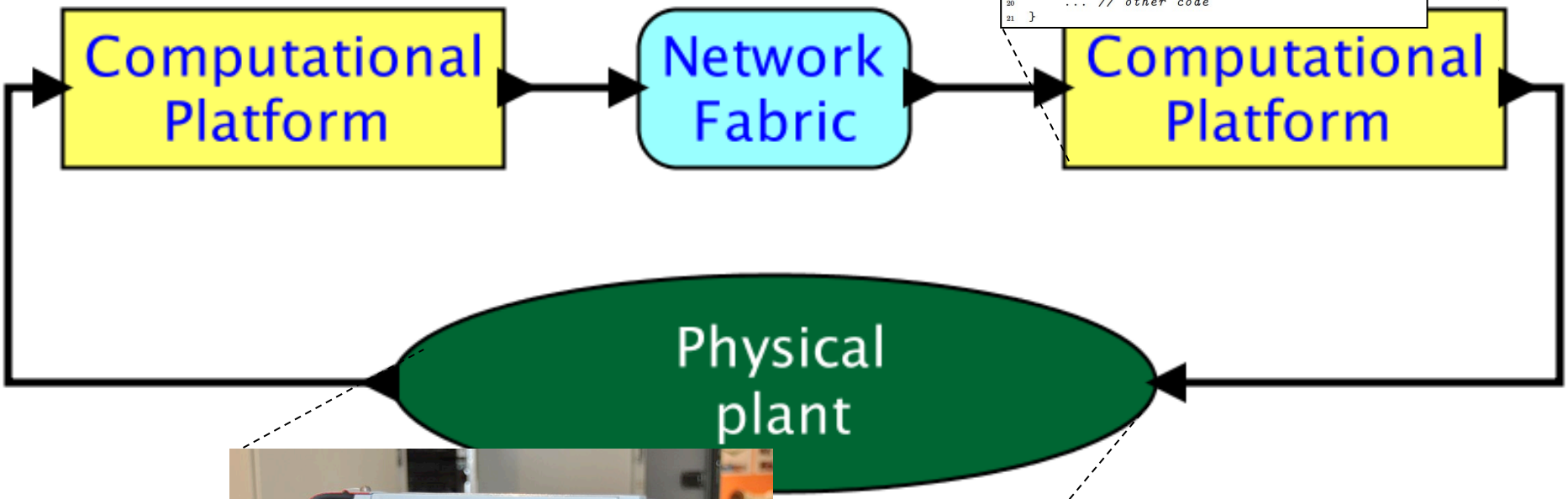
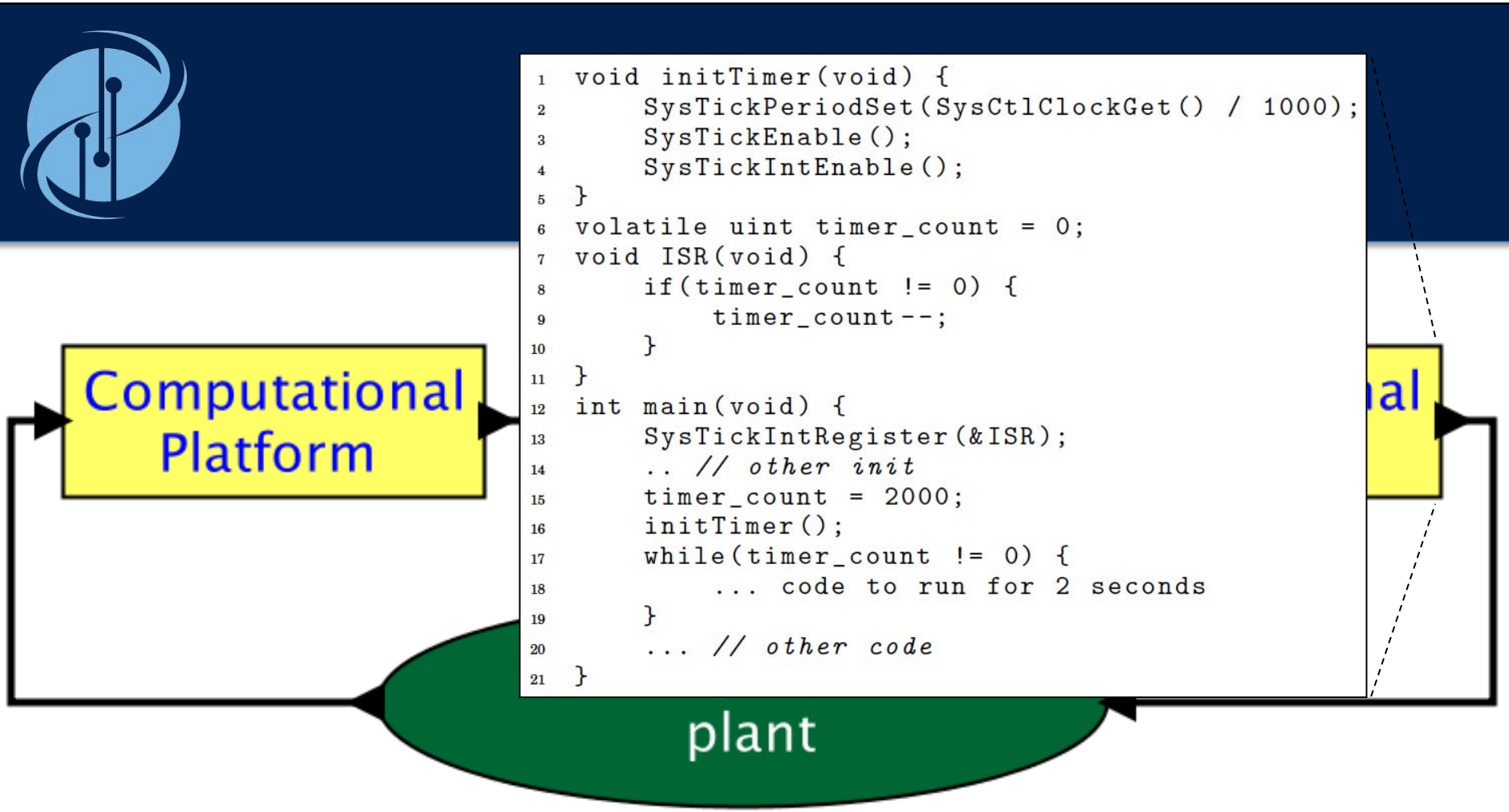


Image: Wikimedia Commons

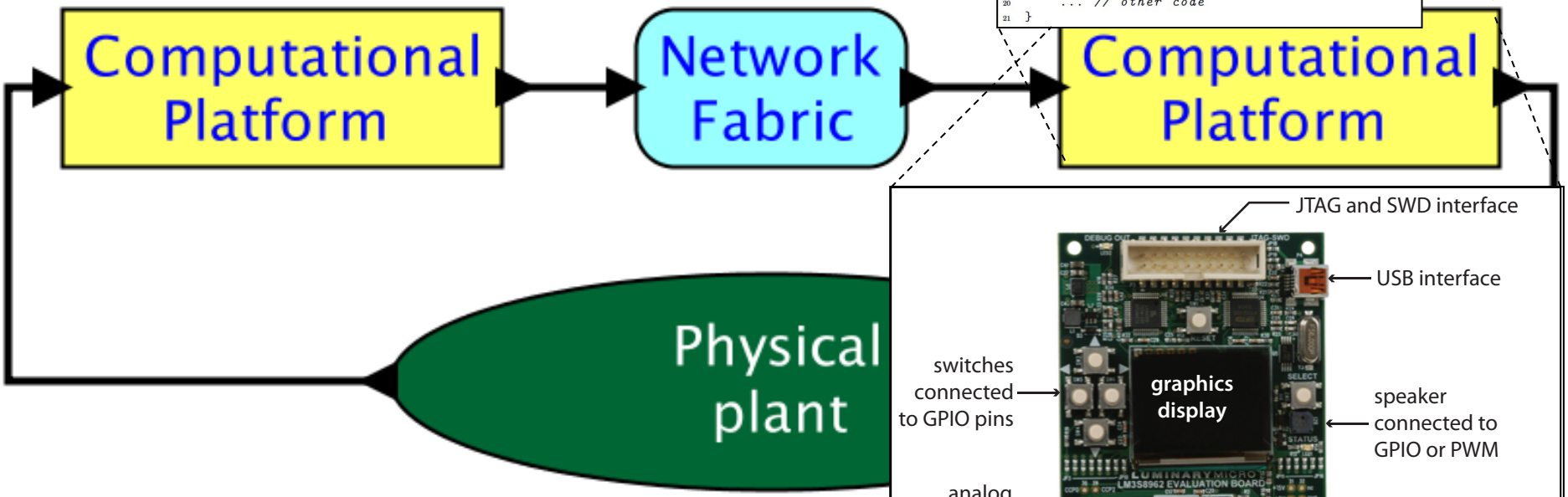



```
1  void initTimer(void) {
2      SysTickPeriodSet(SysCtlClockGet() / 1000);
3      SysTickEnable();
4      SysTickIntEnable();
5  }
6  volatile uint timer_count = 0;
7  void ISR(void) {
8      if(timer_count != 0) {
9          timer_count--;
10     }
11 }
12 int main(void) {
13     SysTickIntRegister(&ISR);
14     .. // other init
15     timer_count = 2000;
16     initTimer();
17     while(timer_count != 0) {
18         ... code to run for 2 seconds
19     }
20     ... // other code
21 }
```

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

Emergent Timing

```
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.

Stellaris LM3S8962 evaluation board (Luminary Micro 2008, now Texas Instruments)

Frozen Designs

Everything about the design, down to wire lengths and microprocessor chips, must be frozen at the time of design.

CCA 2.0
Boeing Dreamscape

Timing is not part of software and network semantics

Correct execution *of a program in all widely used programming languages, and* ***correct delivery*** *of a network message in all general-purpose networks has nothing to do with how long it takes to do anything.*

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

Contrast with correctness criteria in software

We can safely assert that line 8 does not execute, *regardless of the choice of microprocessor*!

```
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!!

Achieving Real Time

- overengineering
- using old technology
- model the processors for response-time analysis
- real-time operating systems (RTOSs)
- specialized networks
- extensive testing and validation

Timing of programs emerges from the implementation

- Pipeline hazards
- Cache effects
- Variable DRAM latencies
- Speculative execution
- Interrupts
- Forwarding
- Dynamic voltage/frequency
- …

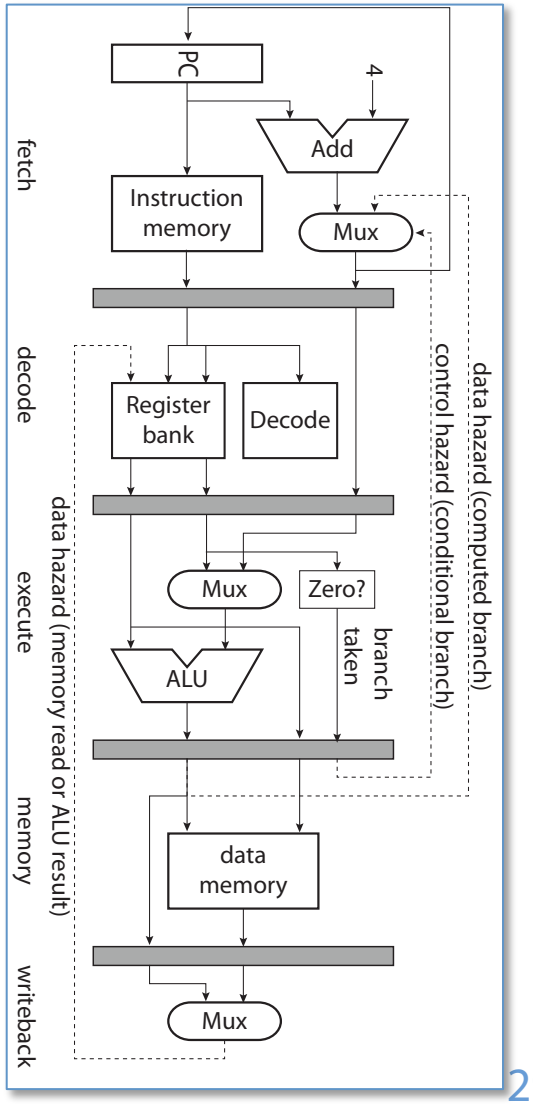


Image from Lee & Seshia,
Introduction to Embedded Systems
MIT Press, 2017

WCET

ACM Tr. on Embedded Computing Systems, April 2008

The Worst-Case Execution-Time Problem—Overview of Methods and Survey of Tools

REINHARD WILHELM
Saarland University
JAKOB ENGBLOM
Virtutech AB
ANDREAS ERMEDAHL
Mälardalen University
NIKLAS HOLSTI
Tidorum Ltd.
STEPHAN THESING
Saarland University

TULIKA MITRA
National University of Singapore
FRANK MUELLER
North Carolina State University
ISABELLE PUAUT
IRISA
PETER PUSCHNER
TU Vienna
JAN STASCHULAT
TU Braunschweig
and
PER STENSTRÖM
Chalmers University of Technology

REINHOLD HECKMANN

The determination of upper bounds on execution times, commonly called worst-case execution times (WCETs), is a necessary step in the development and validation process for hard real-time systems. **This problem is hard if the underlying processor architecture has components such as caches, pipelines, branch prediction, and other speculative components**.

Modeling the Processor

Timing analysis requires detailed info about:

- The pipeline
- Cache management hardware
- Branch prediction hardware
- Bus arbitration hardware
- Memory management hardware
- DRAM architecture

When successful, the model is valid only for a particular piece of silicon, not a family of chips.

Is the Analysis Valid?

Almost all analysis techniques become invalid unless interrupts are disabled.

This means:

- No operating system
- No packet network
- I/O is by polling only

Program with Interrupts

```
1  void initTimer(void) {
2      SysTickPeriodSet(SysCtlClockGet() / 1000);
3      SysTickEnable();
4      SysTickIntEnable();
5  }
6  volatile uint timer_count = 0;
7  void ISR(void) {
8      if(timer_count != 0) {
9          timer_count--;
10     }
11 }
12 int main(void) {
13     SysTickIntRegister(&ISR);
14     .. // other init
15     timer_count = 2000;
16     initTimer();
17     while(timer_count != 0) {
18         ... code to run for 2 seconds
19     }
20     ... // other code
21 }
```

The Cost of Polling

Consider a situation where a rare event (e.g., fire detection) requires a quick response (e.g., within 500 µs). Without interrupts, this implies:

- No task can take more than 500 µs
  - *Every* task becomes time critical.
- The sensor has to be polled every 500 µs.
  - Network and bus traffic and processor cycles wasted

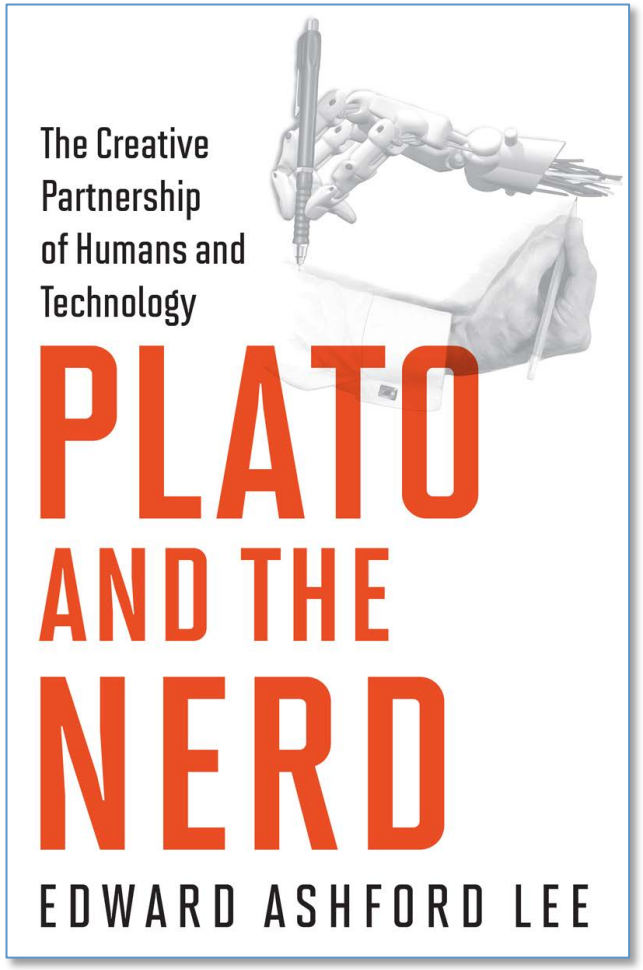
Summary of the State of the Art

- Avoid modern technology (operating systems, programming languages, networks, multicore).
- Model the silicon you use in excruciating detail, beyond what is documented.
- Break your tasks into tiny chunks and prove they do not run longer than 500 µs.
- Schedule everything statically and periodically, even for rare events and non-critical tasks.
- Don’t change *anything* after validating the design.

Outline

- Cyber-Physical Systems
- Real Time
- Timing in Software
- Science and Engineering
- Precision Timed Machines
- Mixed Criticality and FlexPRET
- I/O and Interrupts
- Cost and Performance
- Programming Models
- Parametric PRET Machines

An Epiphany


The Creative Partnership of Humans and Technology
PLATO AND THE NERD
EDWARD ASHFORD LEE


PLATO AND THE NERD
EDWARD ASHFORD LEE

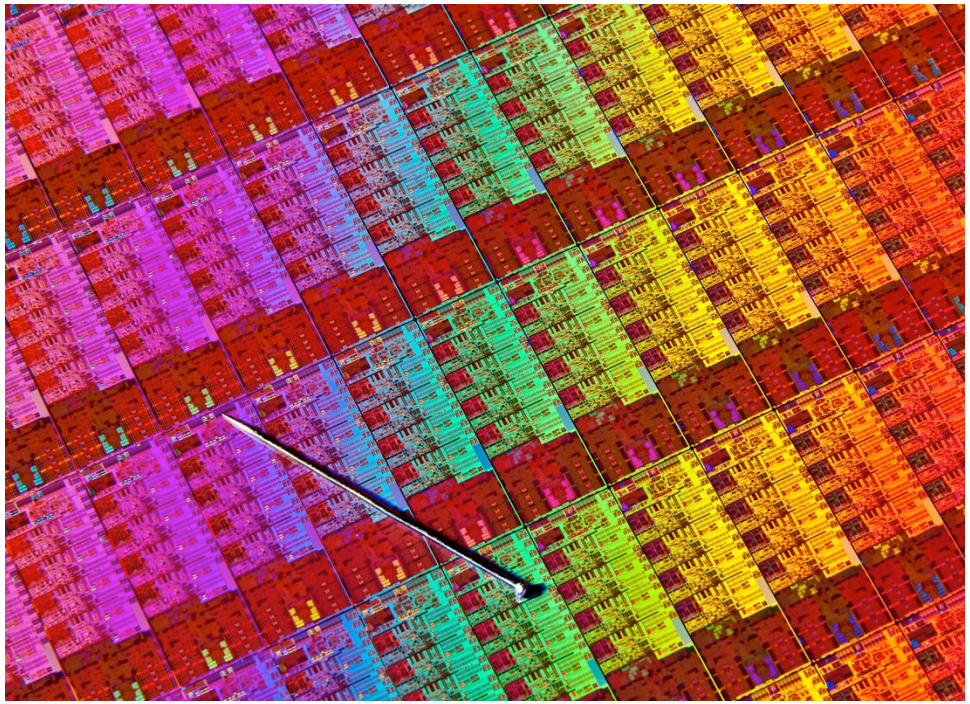
The Value of Models

- In *science*, the value of a *model* lies in how well its behavior matches that of the physical system.

- In *engineering*, the value of the *physical system* lies in how well its behavior matches that of the model.

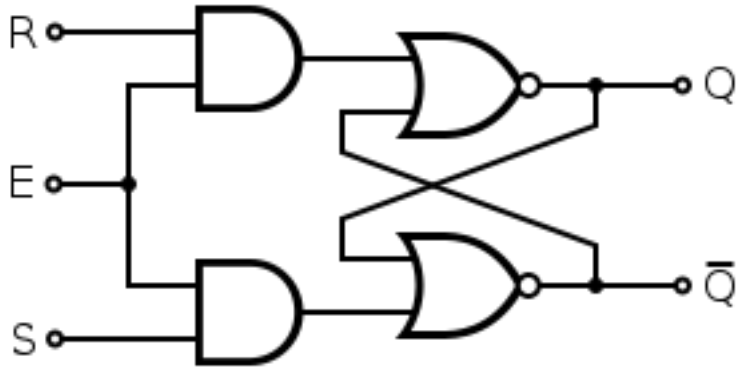
A scientist asks, "Can I make a model for this thing?"
An engineer asks, "Can I make a thing for this model?"

Consider Chip Design



Intel Haswell, each with 1.4 billion transistors

A piece of silicon that doesn't behave like the model is just beach sand.

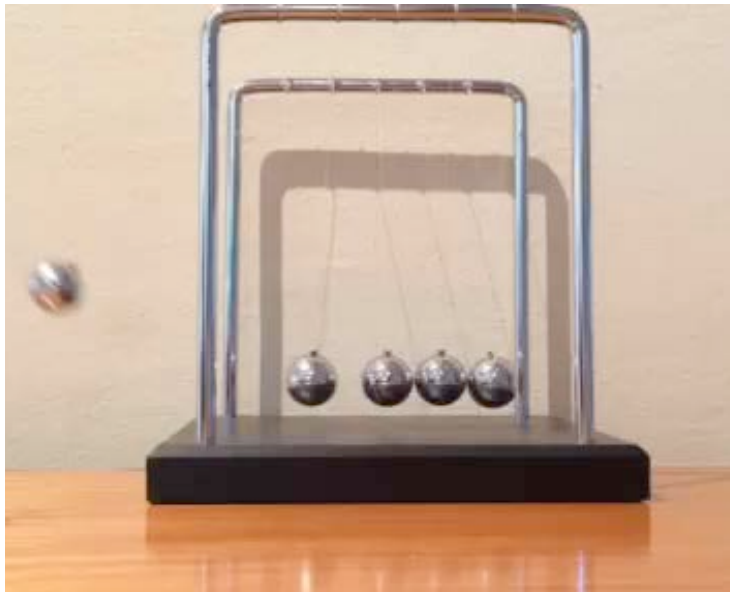


Models vs. Reality

$$x(t) = x(0) + \int_0^t v(\tau)d\tau$$

$$v(t) = v(0) + \frac{1}{m}\int_0^t F(\tau)d\tau$$

The model



The target (the thing being modeled).

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

Fidelity is how well the model and its target match

A Model

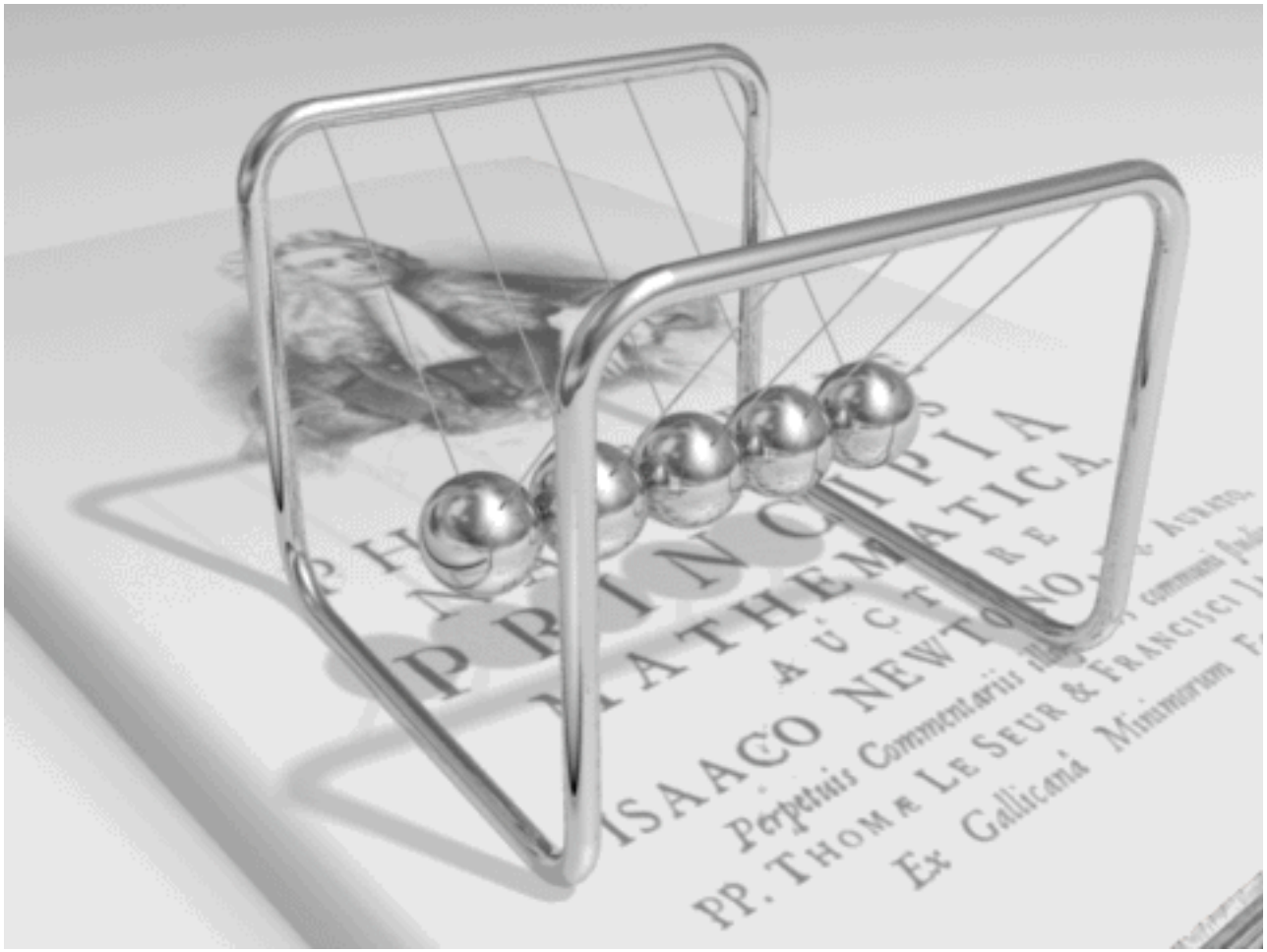
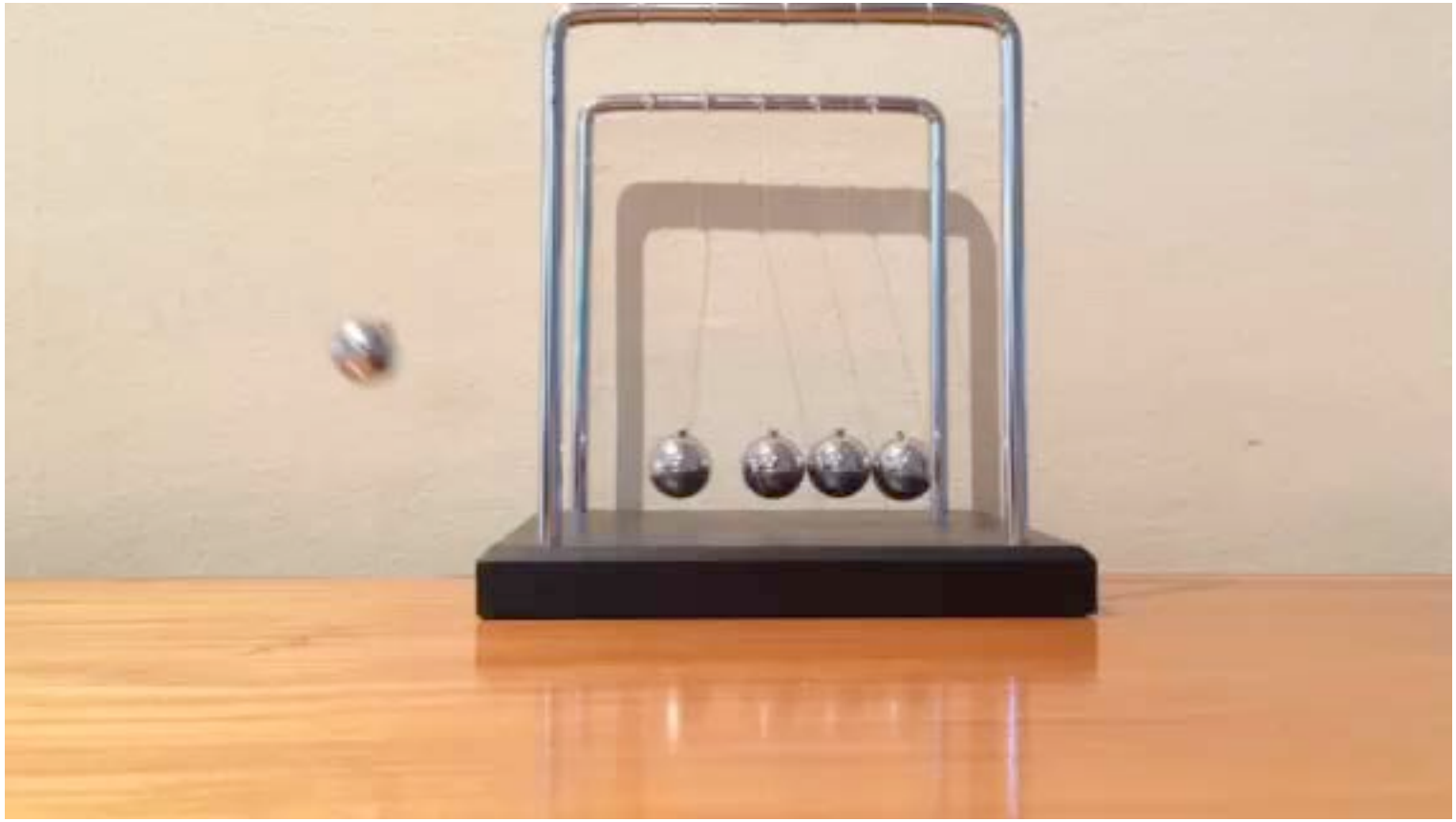


Image by Dominique Toussaint, GNU Free Documentation License, Version 1.2 or later.

A Physical Realization



Model Fidelity

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

I'm an engineer…

Perhaps we should be making our realizations more faithful to our models rather than the other way around?

Useful Models and Useful Things

"Essentially, all models are wrong,
but some are useful."

Box, G. E. P. and N. R. Draper, 1987: *Empirical Model-Building and Response Surfaces*. Wiley Series in Probability and Statistics, Wiley.

"Essentially, all system implementations
are wrong, but some are useful."

Lee and Sirjani, "What good are models," FACS 2018.

The Value of Simulation

"Simulation is doomed to succeed."

[anonymous]

Could this statement be confusing engineering models for scientific ones?

Lee and Sirjani, "What good are models," FACS 2018.

Changing the Question

Is the question whether our models describe the behavior of real-time systems (with high fidelity)?

Or

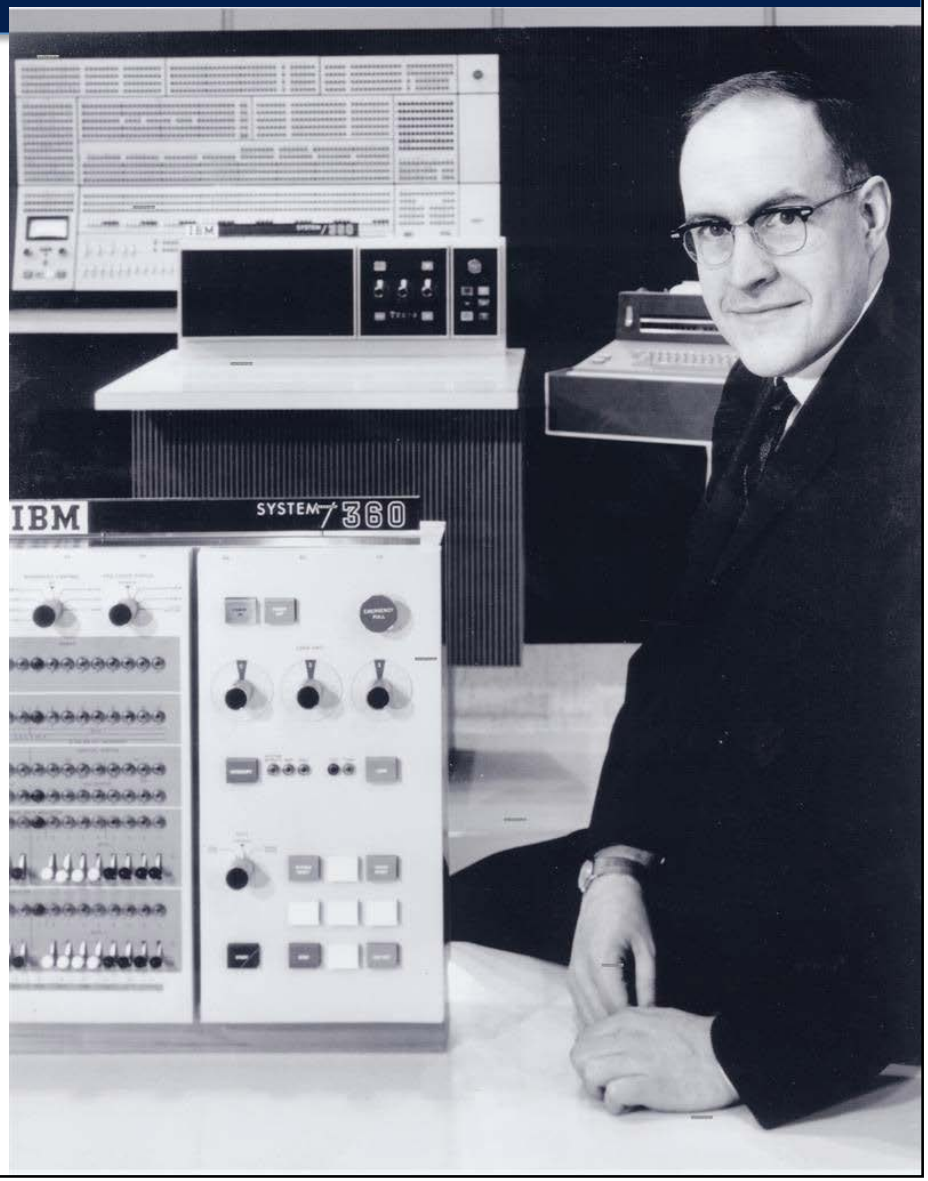
Is the question whether we can build real-time systems where behavior matches that of our models (with high probability)?

Outline

- Cyber-Physical Systems
- Real Time
- Timing in Software
- Science and Engineering
- Precision Timed Machines
- Mixed Criticality and FlexPRET
- I/O and Interrupts
- Cost and Performance
- Programming Models
- Parametric PRET Machines

Instruction Set Architecture (ISA)

The concept of an ISA hasn't changed much since the 1960s.



Fred Brooks

Photo courtesy of computerhistory.org

Instruction Set Architecture (ISA)

Physical System

Image: Wikimedia Commons

Model

Integer Register-Register Operations

RISC-V defines several arithmetic R-type operations. All operations read the *rs1* and *rs2* registers as source operands and write the result into register *rd*. The *funct* field selects the type of operation.

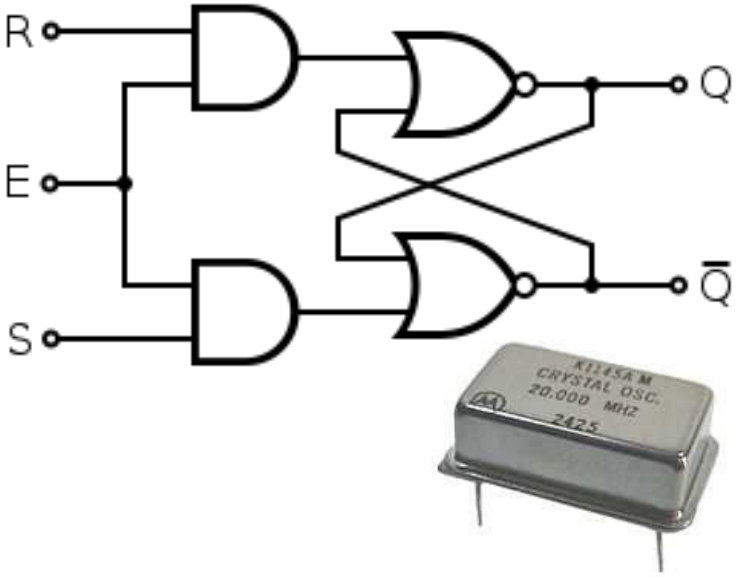
31 27	26 22	21 17	16 7	6 0
rd	rs1	rs2	funct10	opcode
5	5	5	10	7
dest	src1	src2	ADD/SUB/SLT/SLTU	OP
dest	src1	src2	AND/OR/XOR	OP
dest	src1	src2	SLL/SRL/SRA	OP
dest	src1	src2	ADDW/SUBW	OP-32
dest	src1	src2	SLLW/SRLW/SRAW	OP-32

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

An ISA is a deterministic model of the behavior of hardware

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

Synchronous digital logic delivers precise, repeatable timing.



… but the overlaying software abstractions discard timing.

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

PRET Machines – Giving Software the Capabilities its Hardware Already Has.

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

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

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

Computing

+

With time

= PRET

Major Challenges

and existence proofs that they can be met

- Pipelines
  - fine-grain multithreading
- Memory hierarchy
  - memory controllers with controllable latency
- I/O
  - threaded interrupts with zero effect on timing

Pipeline Hazards

- Read after write, e.g.
  - i1. R2 <- R5+R3
  - i2. R4 <- R2+R3

Pipeline bubble

- Write after read
- Write after write
- Branch based on result

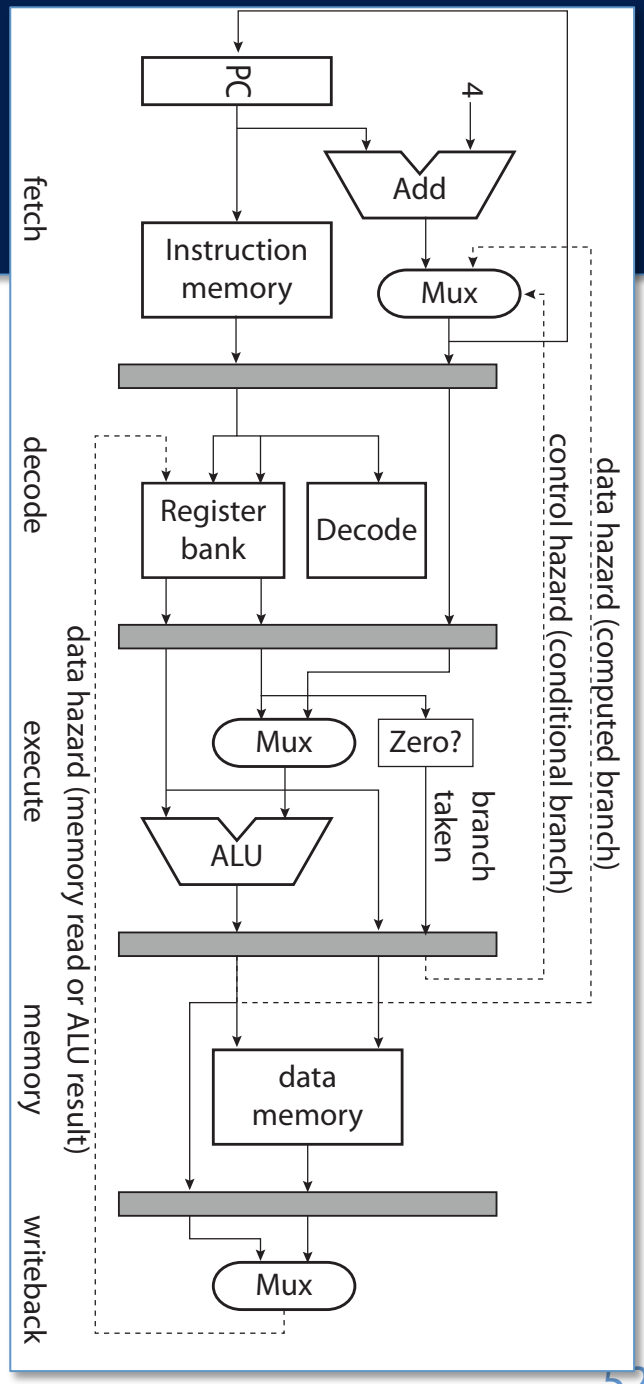


Image from Lee & Seshia,
Introduction to Embedded Systems
MIT Press, 2017

Three Generations of PRET Machines at Berkeley

- PRET1, Sparc-based (simulation only)
  - [Lickly et al., CASES, 2008]
- PTARM, ARM-based (FPGA implementation)
  - [Liu et al., ICCD, 2012]
- FlexPRET, RISC-V-based (FPGA + simulation)
  - [Zimmer et al., RTAS, 2014, PhD Thesis 2015]

Multiple Register Banks

All our PRET machines have multiple register banks, each providing a "**hardware thread**."

Cost in hardware is modest.

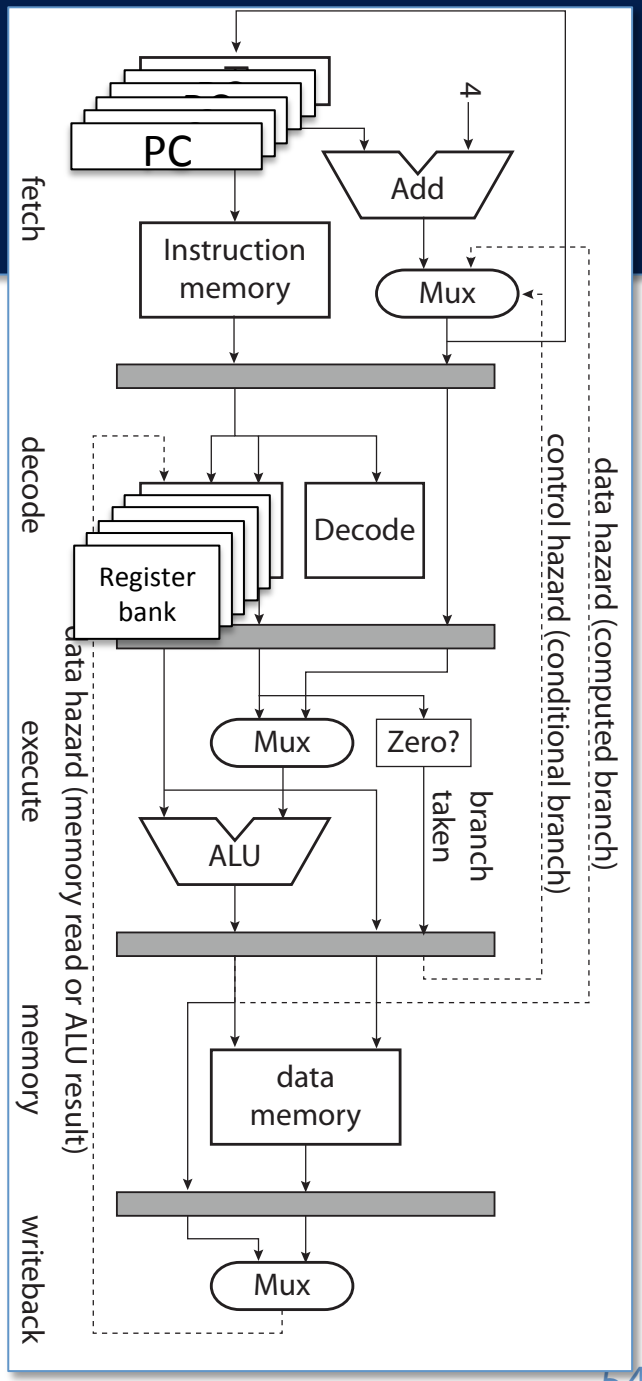


Image from Lee & Seshia,
Introduction to Embedded Systems
MIT Press, 2017

Multiple Register Banks

If independent tasks are interleaved through the pipeline, hazards disappear.

Can eliminate pipeline bubbles.

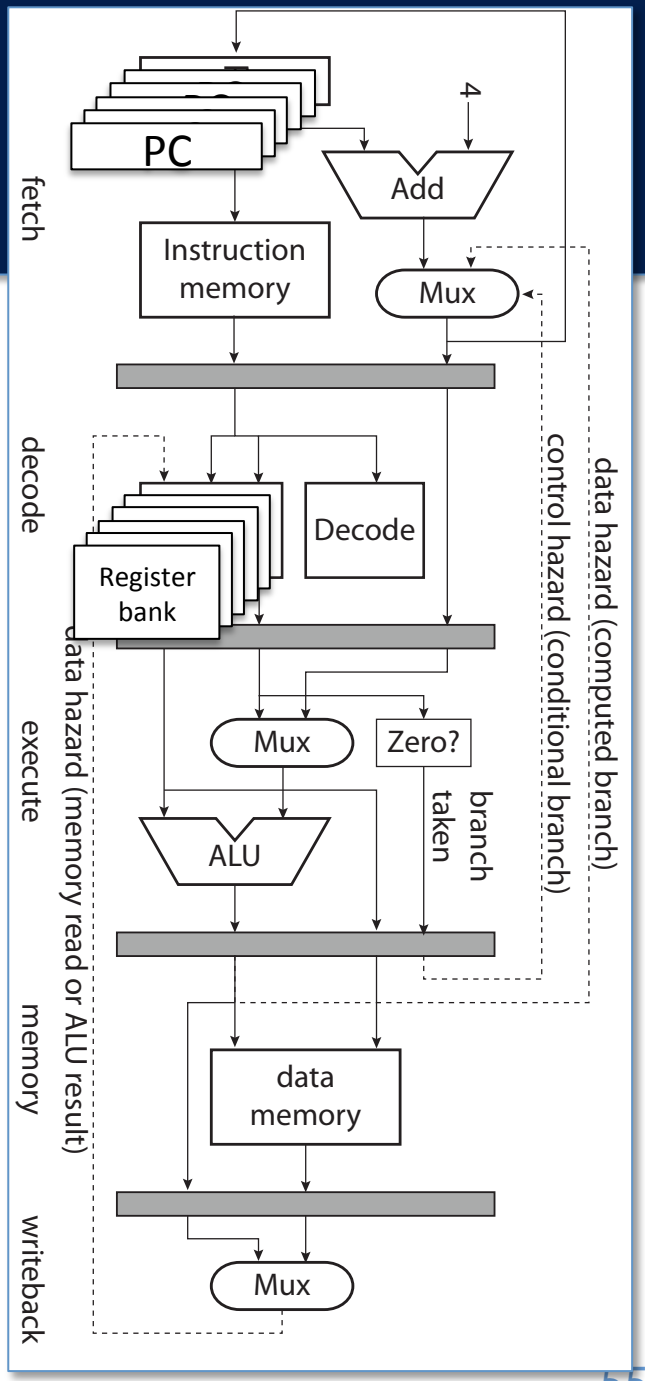


Image from Lee & Seshia,
Introduction to Embedded Systems
MIT Press, 2017

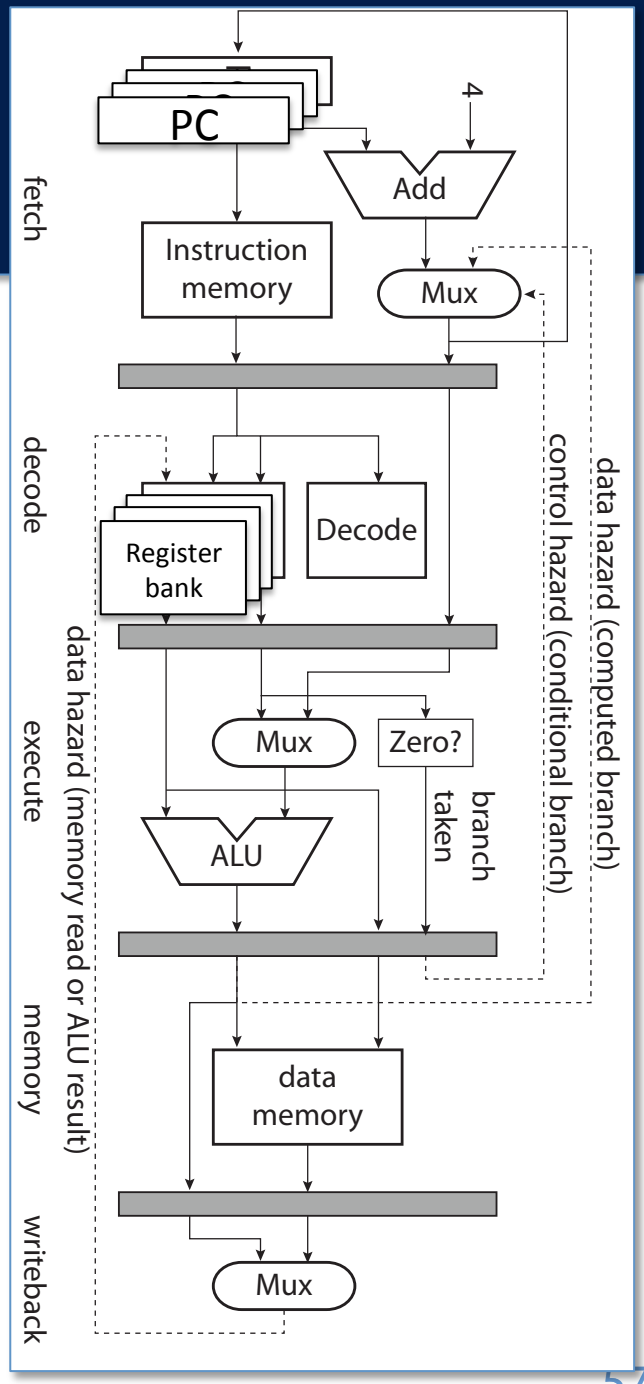
Pipeline Interleaving is Old

First used in the CDC 6600.

2nd Generation PRET

PTArm, a soft core on a
Xilinx Virtex 5 FPGA (2012)

- Four hardware threads
- Deterministic instruction timing
- Shared SRAM memory
- Partitioned DRAM memory
- Precise timing instructions



DRAM Memories

The latency of a DRAM memory access depends on the address of the previous access.

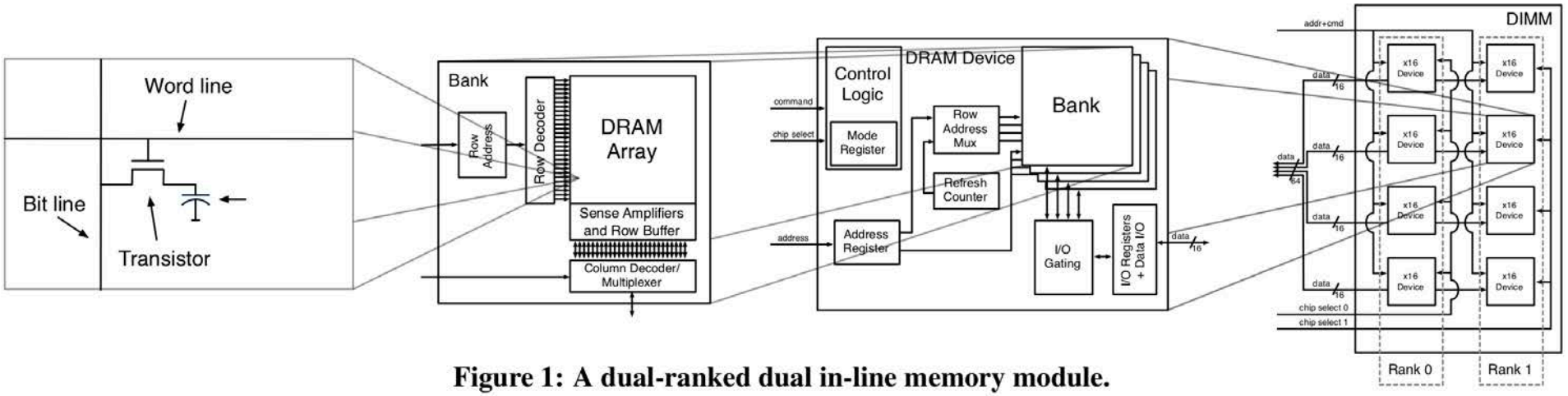
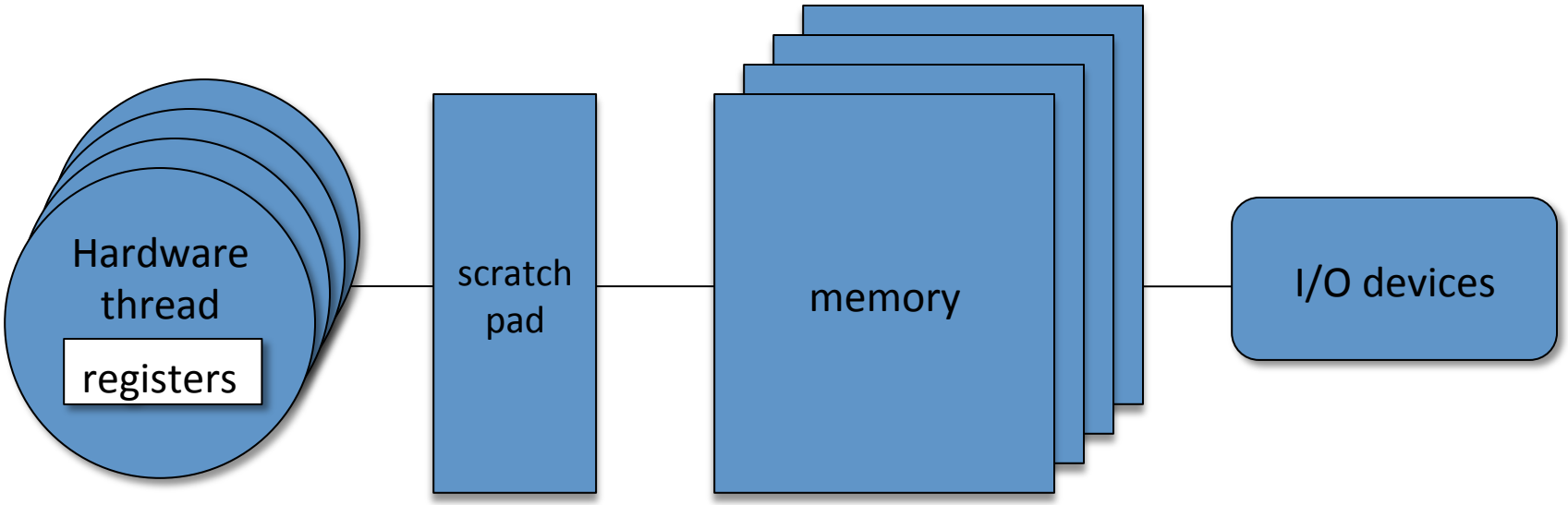


Figure 1: A dual-ranked dual in-line memory module.

Assigning banks to hardware threads can make this deterministic.

Reineke, Liu, Patel, Kim, & Lee. "PRET DRAM Controller: Bank Privatization for Predictability and Temporal Isolation," CODES+ISSS '11.

2nd Generation PRET Summary



Interleaved pipeline with one set of registers per thread

SRAM scratchpad shared among threads

DRAM main memory, separate banks per thread

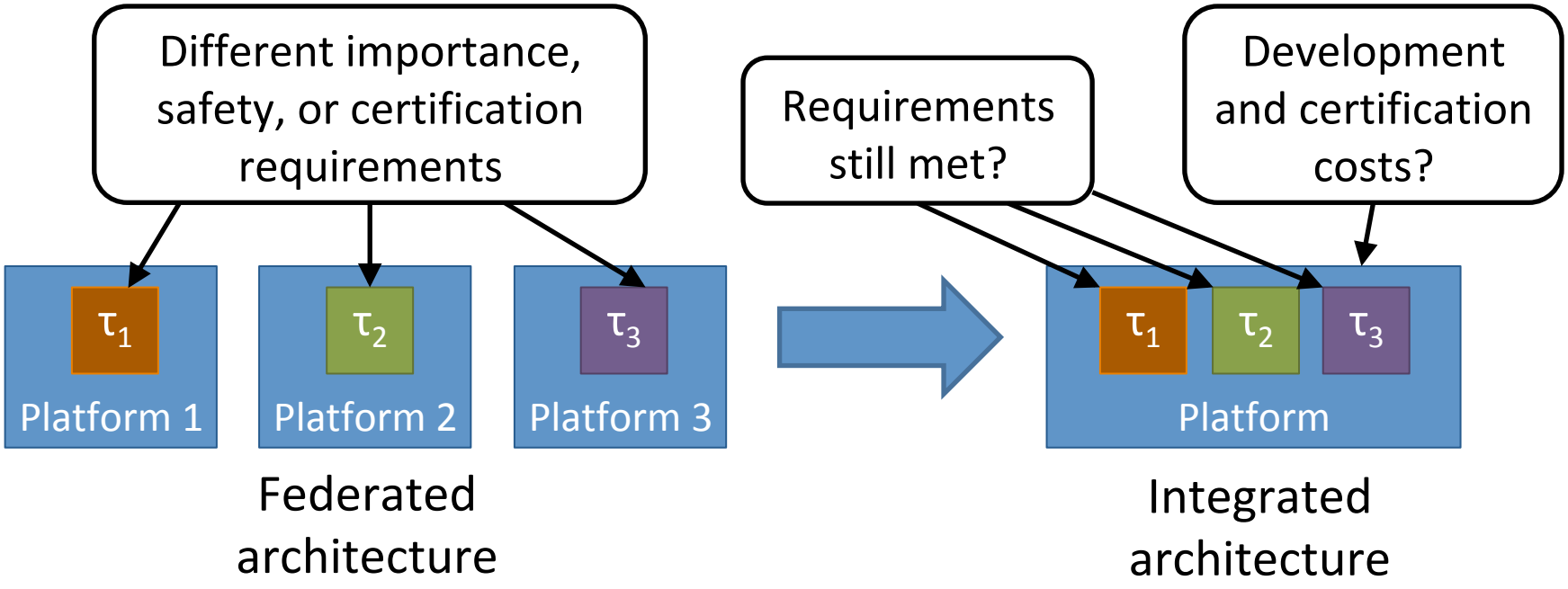
Isaac Liu, PhD Thesis, 2012

Outline

- Cyber-Physical Systems
- Real Time
- Timing in Software
- Science and Engineering
- Precision Timed Machines
- Mixed Criticality and FlexPRET
- I/O and Interrupts
- Cost and Performance
- Programming Models
- Parametric PRET Machines

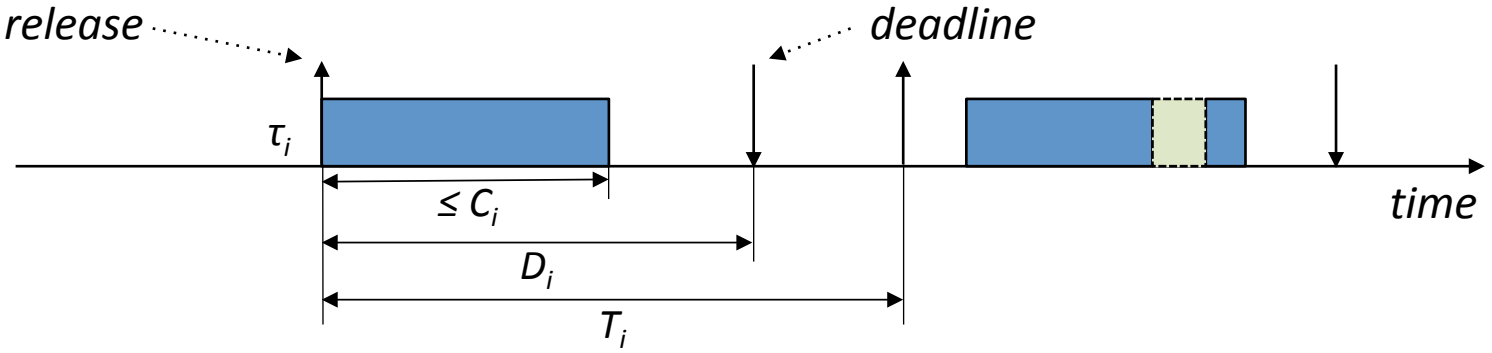
Mixed Criticality Systems

- Increasingly complex functionality
  - **Cost** and size, weight and power (SWaP) concerns
  - E.g. High-end cars with 70-100+ ECUs



HRT and SRT

- Set of independent, periodic **tasks τ_i**, each assigned a **criticality level**



- Each criticality level has a requirement regarding **deadline importance**

hard real-time (HRT) tasks:
should never miss deadlines

higher criticality levels

soft real-time (SRT) tasks:
less utility if deadline missed

lower criticality levels

Tradeoffs

Hard real-time tasks

Isolation
(spatial and temporal)
independent behavior

+ Modularity
+ Independent verification

Timing predictability
Safe and tight bounds on worst-case execution time (WCET)

+ Verification
+ Less over provisioning

Soft real-time tasks

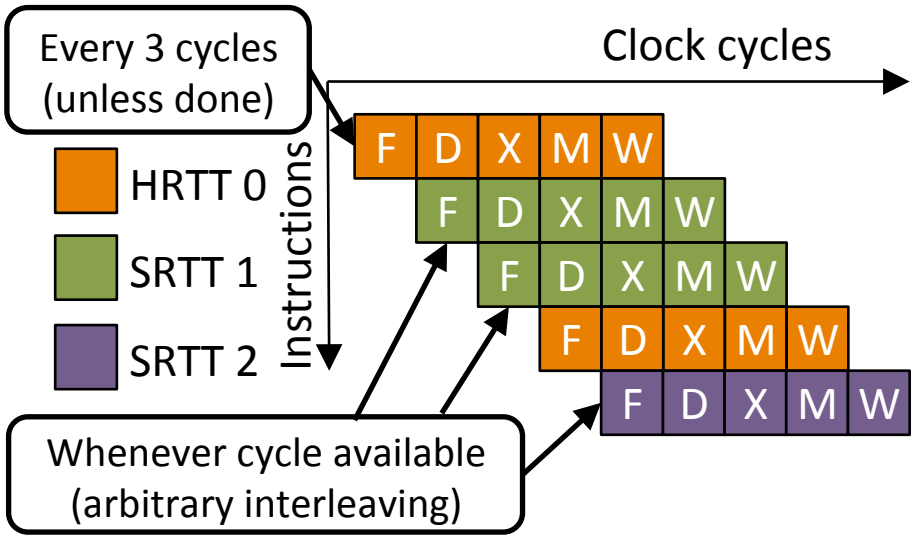
Efficient processor utilization

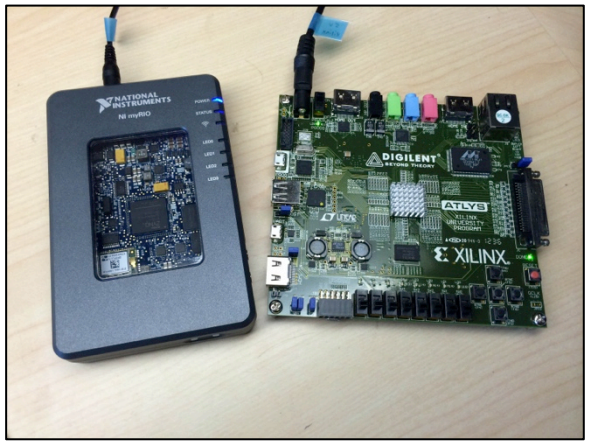
+ Cost

FlexPRET makes these tradeoffs at the task level and not processor level.

3rd Generation PRET: Open-Source FlexPRET (Zimmer 2014/15)

- 32-bit, 5-stage thread interleaved pipeline, RISC-V ISA
  - **Hard real-time HW threads**: scheduled at constant rate for isolation and repeatability.
  - **Soft real-time HW threads**: share all available cycles for efficiency.
- Deployed on Xilinx FPGA



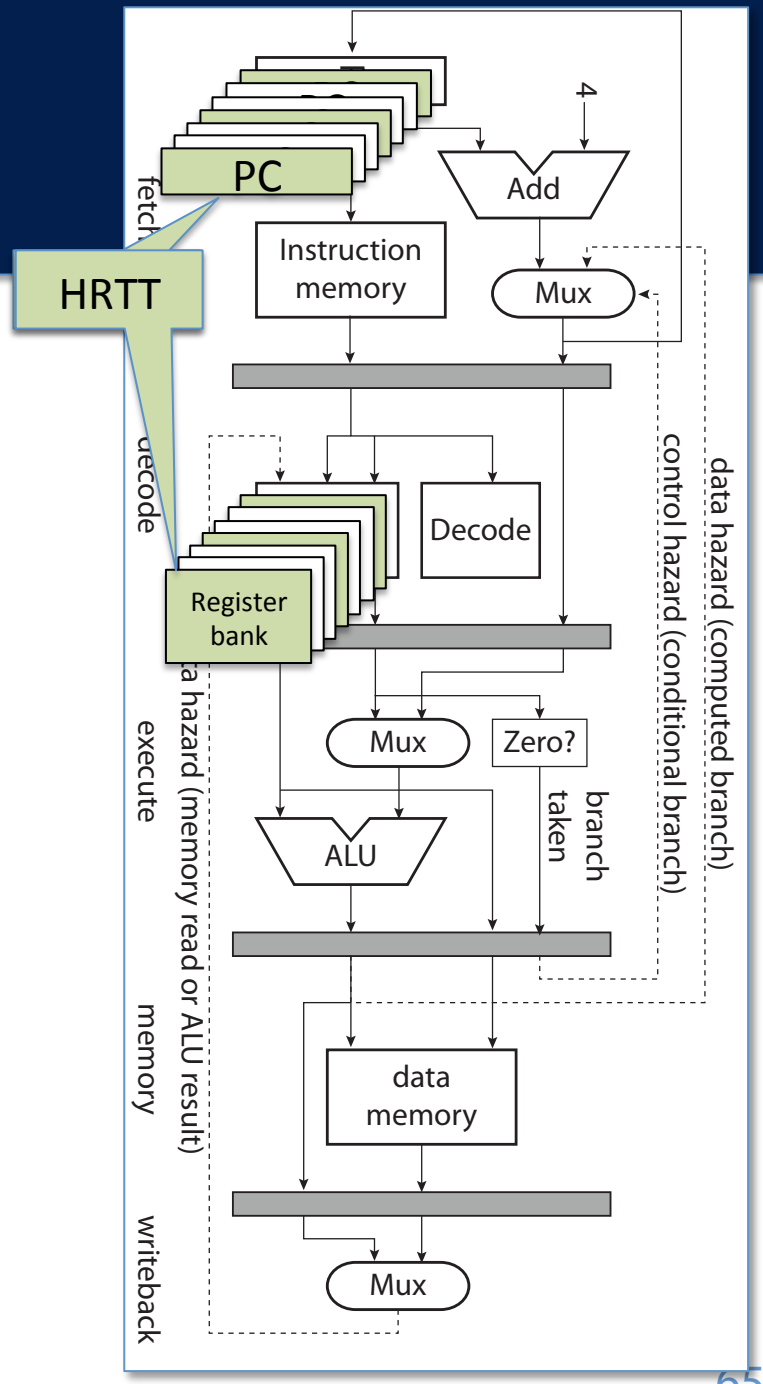


Digilent Atlys (Spartan 6) and NI myRIO (Zync)

Hard and Soft Real-Time Threads

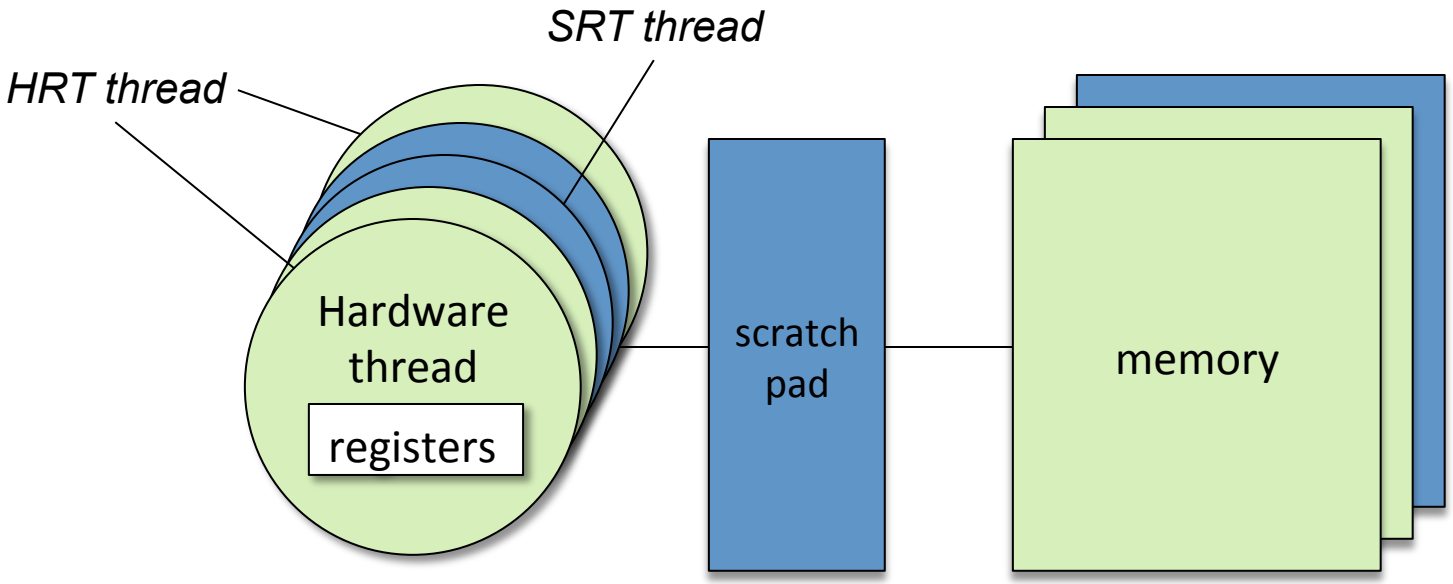
HRTTs (when active) are issued periodically according to a fixed schedule.

SRTTs use all remaining cycles.



FlexPRET

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



HRT threads have deterministic timing. SRT threads share remaining cycles

SRAM scratchpad shared among threads

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

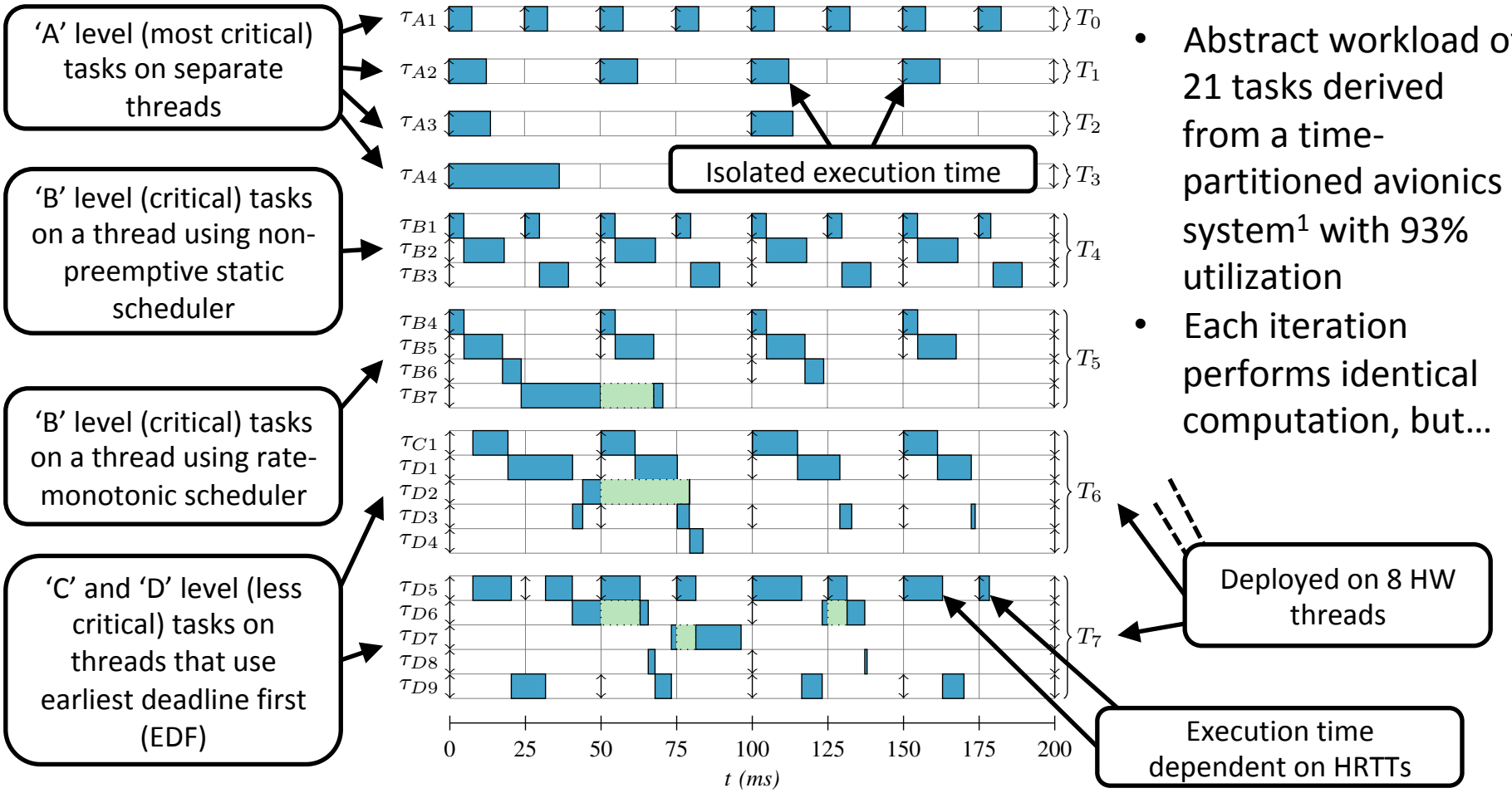
Michael Zimmer

FlexPRET Summary

- RISC V ISA
- 5-stage, fine-grained multithreaded processor designed for mixed-criticality systems.
- Implemented with Chisel. Compiles to FPGA and a C++ cycle-accurate simulator.
- Class: 100-200MHz 32-bit embedded processor.
- Open source: https://github.com/pretis/flexpret

Avionics Mixed Criticality Test Case

Zimmer, PhD Thesis, 2015.



- Abstract workload of 21 tasks derived from a time-partitioned avionics system[1] with 93% utilization
- Each iteration performs identical computation, but…

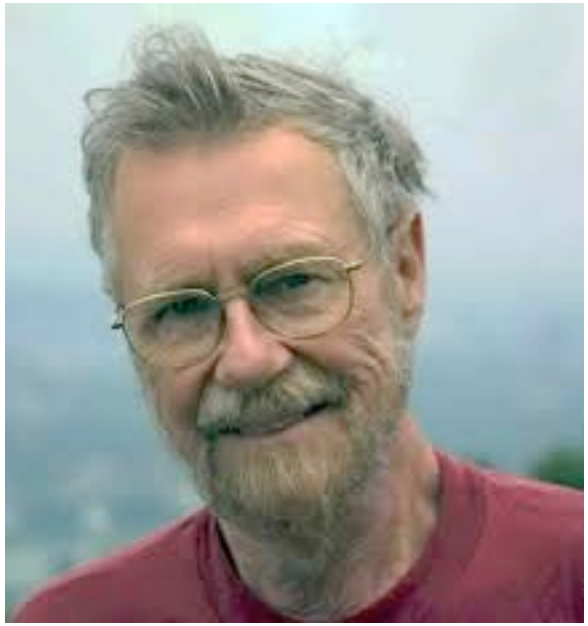
S. Vestal, "Preemptive Scheduling of Multi-criticality Systems with Varying Degrees of Execution Time Assurance," in RTSS 2007.

Outline

- Cyber-Physical Systems
- Real Time
- Timing in Software
- Science and Engineering
- Precision Timed Machines
- Mixed Criticality and FlexPRET
- I/O and Interrupts
- Cost and Performance
- Programming Models
- Parametric PRET Machines

About Interrupts

"[M]any a systems programmer's grey hair bears witness to the fact that we should not talk lightly about the logical problems created by that feature"



- Edsger Dijkstra (1972)

Interrupts

- Nondeterministically interleaved with program
- Make response time > execution time
- Disrupt cache and branch predictors
- Overhead of context switching

- For WCET analysis, have to disable interrupts
- Disabling interrupts increases variability in response time and forces all tasks to be small.

Interrupts

Scientific solution:

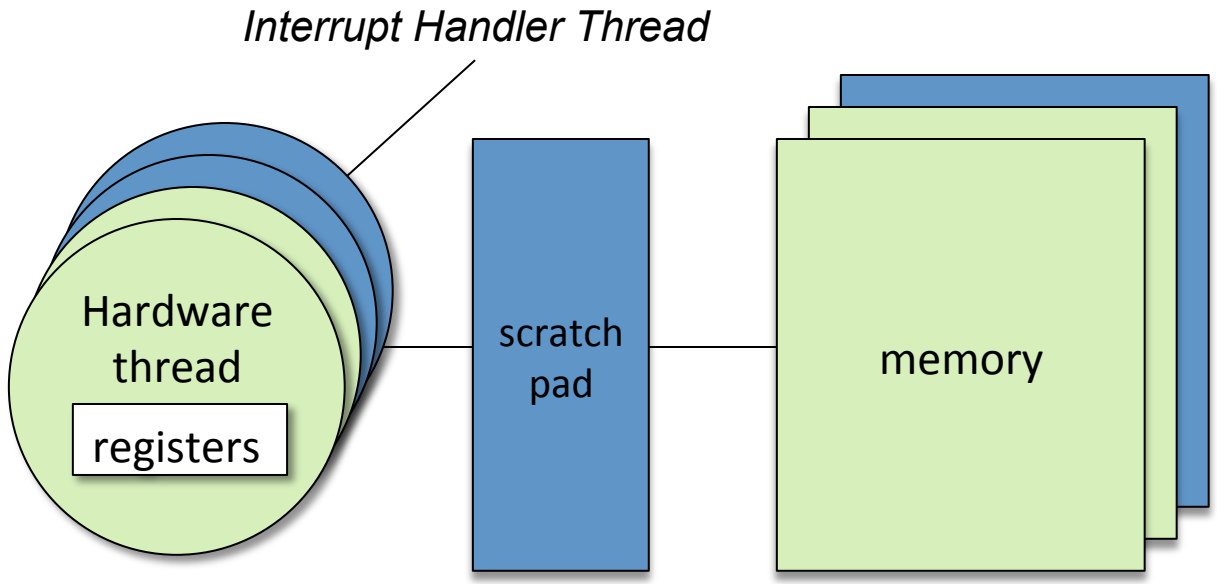
- Model all these effects

Engineering solution:

- Eliminate all these effects

The latter is what PRET machines do.

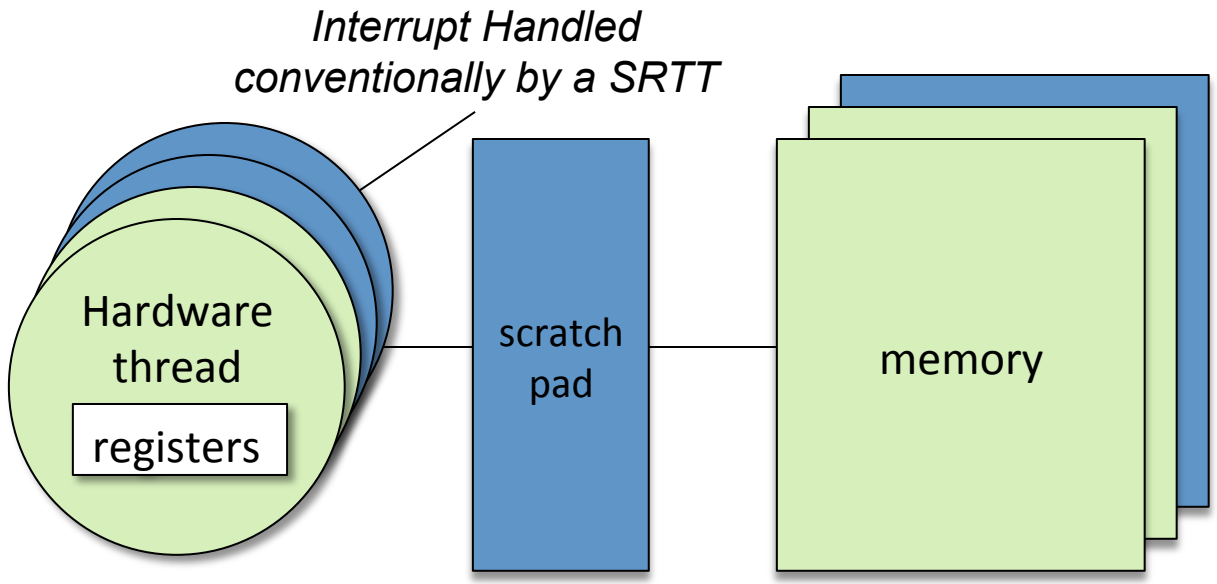
FlexPRET I/O
Interrupt Handler Thread Option 1



Such interrupts have
no effect on HRT threads, and
bounded effect on SRT threads!

A similar strategy is also used by XMOS, but with less isolation.

FlexPRET I/O
Interrupt Handler Thread Option 2



Such interrupts have *no effect* on HRT threads, and *bounded effect* on SRT threads other than the handling thread.

Outline

- Cyber-Physical Systems
- Real Time
- Timing in Software
- Science and Engineering
- Precision Timed Machines
- Mixed Criticality and FlexPRET
- I/O and Interrupts
- Cost and Performance
- Programming Models
- Parametric PRET Machines

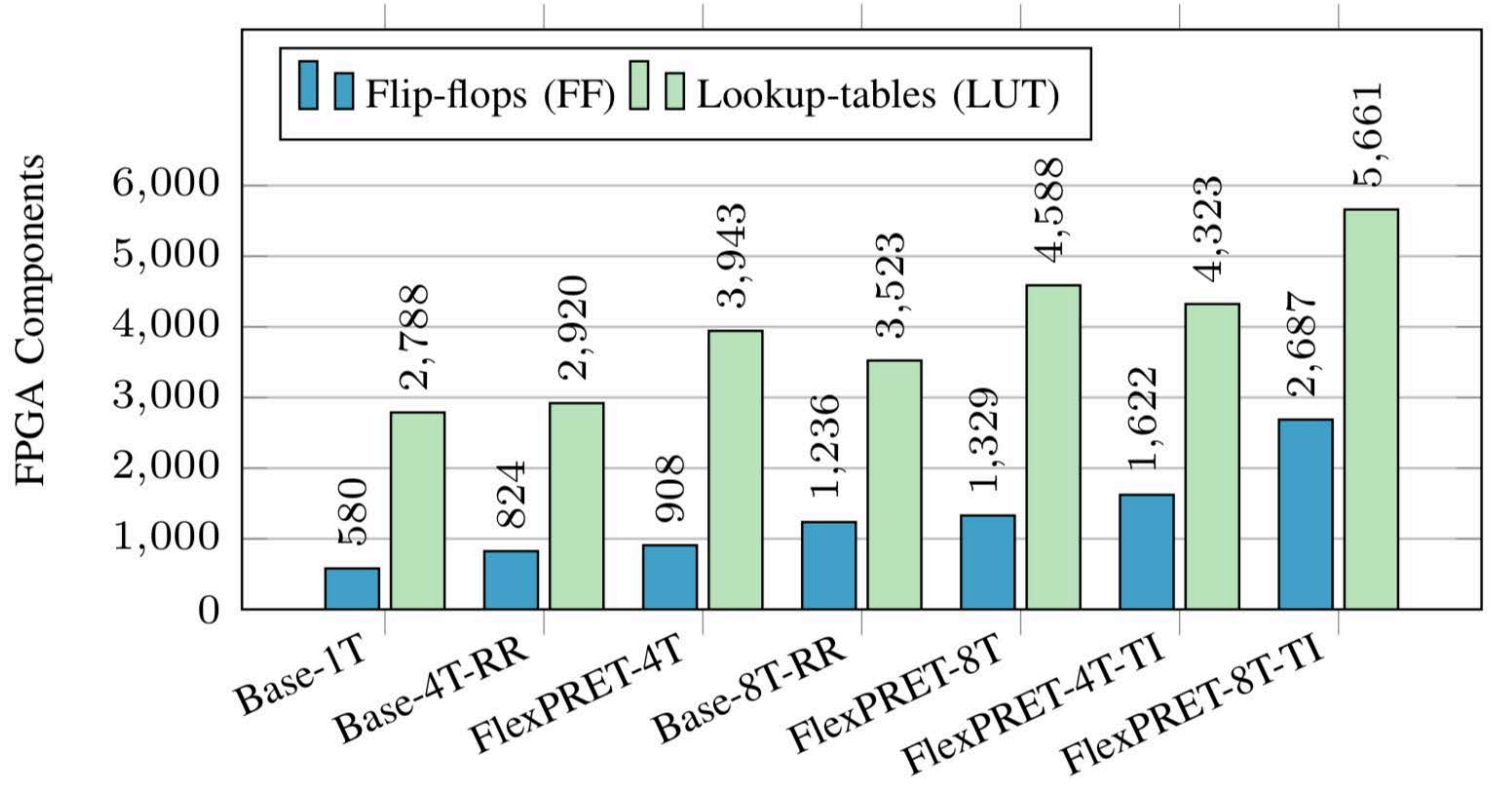
Fact

The real-time performance of a FlexPRET machine is never worse than that of a conventional machine.

Proof: A FlexPRET machine *is* a conventional machine if the memory-mapped registers controlling HRT and SRT threads is set to have only one thread, a SRT thread.

The Cost

Size:



[Zimmer, Broman, Shaver, Lee, RTAS 2014]

The Cost

A **baseline RISC-V** without any complex instructions (floating point, integer division, packed instructions) can be realized on an FPGA with 580 flip flops and 2,788 LUTs.

A **4-thread FlexPRET** can be realized with 908 flip flops and 3,943 LUTs, an increase of 56% and 41% respectively.

Percentage is much lower with floating point, division, etc.

[Zimmer, Broman, Shaver, Lee, RTAS 2014]

Abstract PRET Machines (APM)

Abstract PRET Machines

Invited TCRTS award paper

Edward A. Lee (award recipient) Jan Reineke Michael Zimmer

RTSS, 2017, Paris.

This paper shows that achieving deterministic response times that meet deadlines, when that is feasible, comes at *no cost* in worst-case response times.

This is shown for a task model of *N* sporadic independent tasks with deadlines.

Intuition

- *N* sporadic real-time tasks with minimum interarrival time T_i, deadlines D_i, and WCET C_i.

Theorem: When $T_i = D_i$, PRET yields deterministic response times no worse than the worst case response time of a conventional architecture.

When $T_i > D_i$, if any processor can deliver deterministic response times, PRET will, with worst case response time no worse than a conventional architecture.

Bottom Line

At modest hardware cost, FlexPRET offers the possibly of isolated hard-real-time tasks that are unaffected by interrupts and other real-time tasks.

This comes at *no cost* in performance.

Considering the elimination of pipeline bubbles, for some workloads, it *improves* performance.

Benefits of PRET
(Even if you don't care about determinism)

- **Very low context switch overhead**
  - Up to the number of hardware threads.
  - Conventional overhead above that.
- **Tighter WCET analysis**
  - Particularly when activating enough threads to eliminate pipeline bubbles and memory access order dependencies.
- **No longer need to be restricted to polling I/O**
  - Effect of interrupts is bounded.

Benefits of PRET
(If you take advantage of determinism)

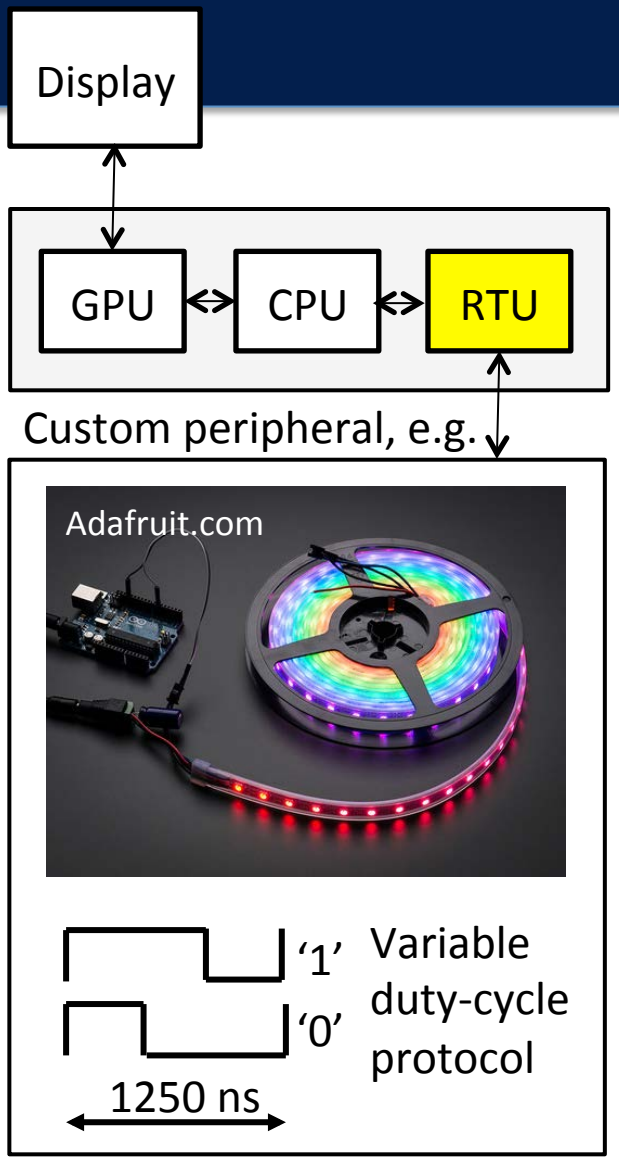
- **Modularity**
  - Non-interference between tasks.
  - Interrupts have *exactly no effect* on hard-real-time tasks.
- **Exactness**
  - Can get not just WCET, but actual ET.
  - Not just ET, but *response* time.
- **Repeatability**
  - Works in the field like on the bench.
  - Event ordering can be made invariant.
- **Complexity**
  - More hard-real-time tasks is better than fewer.
- **Certifiability**
  - Every *correct* execution of the software gives the same behavior.
- **Energy**
  - Reduce voltage and frequency to the bare minimum to meet deadlines.

FlexPRET and Energy

- With enough concurrency in the application, every cycle performs useful work (no speculation, no pipeline bubbles, no memory stalls)
- Flexibility for logical operating frequency of each thread (load balancing)
- Predictability enables tighter bounds on worst-case execution time (WCET)
- Voltage and frequency can be reduced until all deadlines are barely met.

One way to Make PRET Widespread
Real-Time Units (RTUs)

- Offload timing-critical functions to the RTU
  - Compare with dedicated hardware
- Software peripherals
  - Bit-banging for custom protocols
- Software API: OpenRT?
  - Richer interface for smart sensors/actuators



So why isn't every modern processor a FlexPRET?

Possibilities:

- Our claims look too good to be true.
- Programming models that can take advantage of this are missing.

Outline

- Cyber-Physical Systems
- Real Time
- Timing in Software
- Science and Engineering
- Precision Timed Machines
- Mixed Criticality and FlexPRET
- I/O and Interrupts
- Cost and Performance
- Programming Models
- Parametric PRET Machines

Basic Timing Control in PRET

Extended RISC-V ISA with timing instructions. E.g. Hard-real-time periodic task:

internal clock value (in nanoseconds)

```
time r;
get_time(r);
while(1) {
    add_ms(r, 10);
    exception_on_expire(r);
    task();
    deactivate_exception();
    delay_until(r);
}
```

Compute future time

Execution interrupted at time r (deadline miss)

If here, no deadline miss so deactivate

Wait until next period, **SRTT can use allocated cycles**

Bui, Lee, Liu, Patel, and Reineke, "Temporal isolation on multiprocessing architectures," DAC 2011.

Four Patterns of Timed Code Blocks

[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
```

Higher-Level Programming Models

The above patterns are rather low level. Come next week for higher-level programming models that map beautifully to PRET (as well as to conventional architectures).

But Wait…

The whole point of an ISA is that the same program does the same thing on multiple hardware realizations.

Isn't this incompatible with deterministic timing?

Outline

- Cyber-Physical Systems
- Real Time
- Timing in Software
- Science and Engineering
- Precision Timed Machines
- Mixed Criticality and FlexPRET
- I/O and Interrupts
- Cost and Performance
- Programming Models
- Parametric PRET Machines

Parametric PRET Machines

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

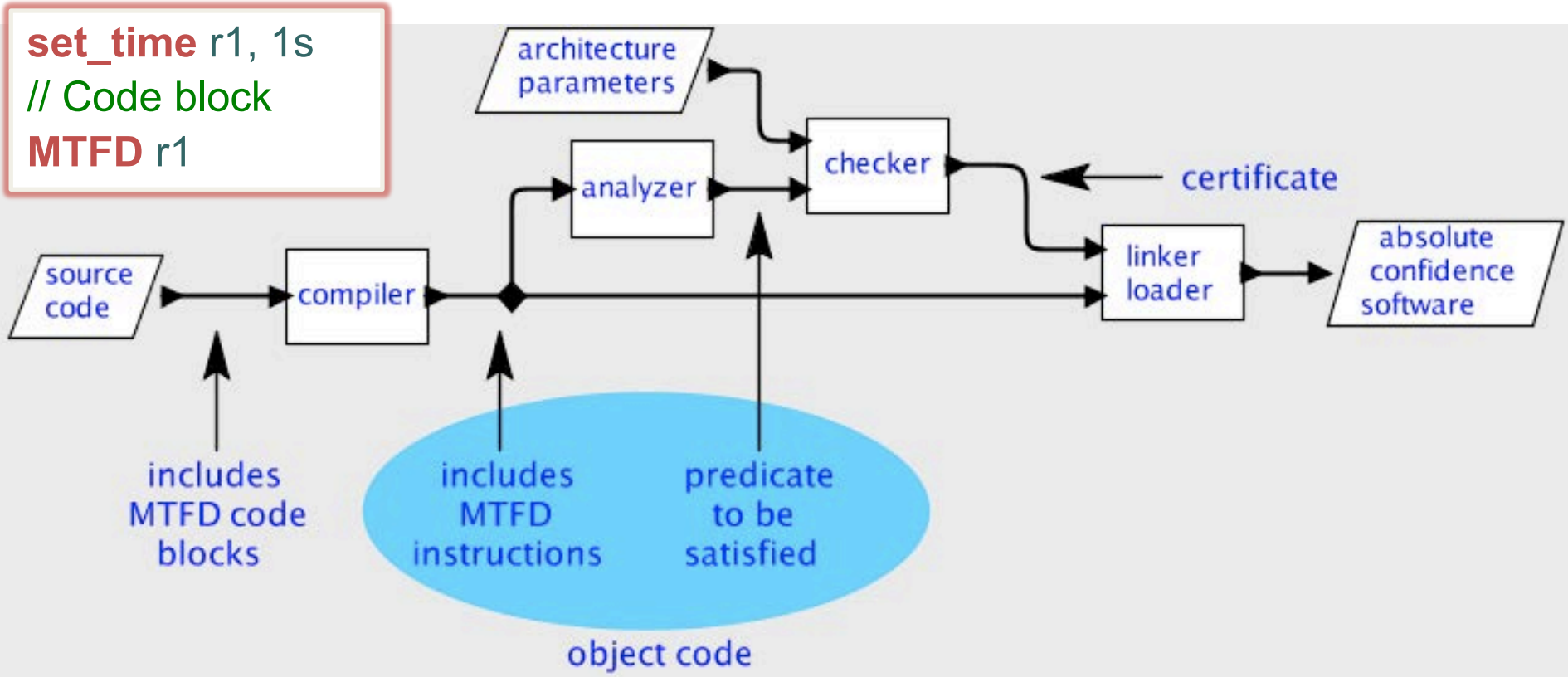
ISA that admits a variety of implementations:

- 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 many implementations of the ISA, and that correctness can be checked for each implementation.

Research Opportunities

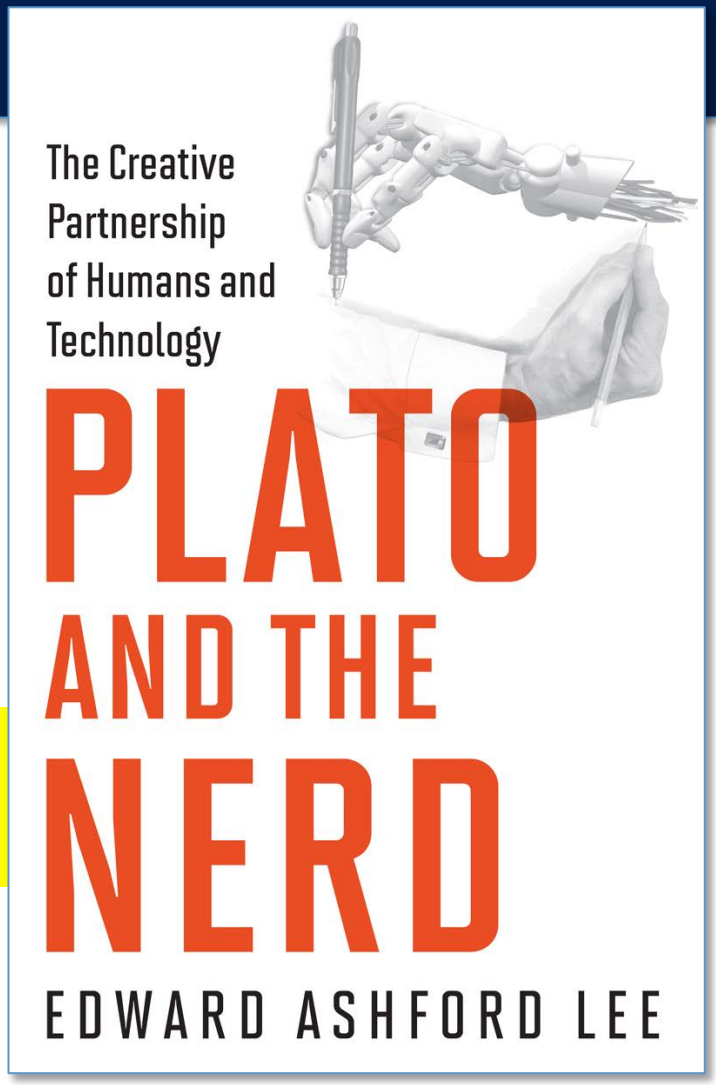
- Exploiting potential reduction in energy
- Programming models with temporal semantics
- Synthesizing HRT thread schedules on the fly
- OS support for dynamic HRT thread instantiation

Conclusion

- In *science*, the value of a *model* lies in how well its behavior matches that of the physical system.
- In *engineering*, the value of the *physical system* lies in how well its behavior matches that of the model.

My message:
Do less science and more engineering.

http://ptolemy.berkeley.edu/pret
http://ptolemy.berkeley.edu/ptides



http://platoandthenerd.org