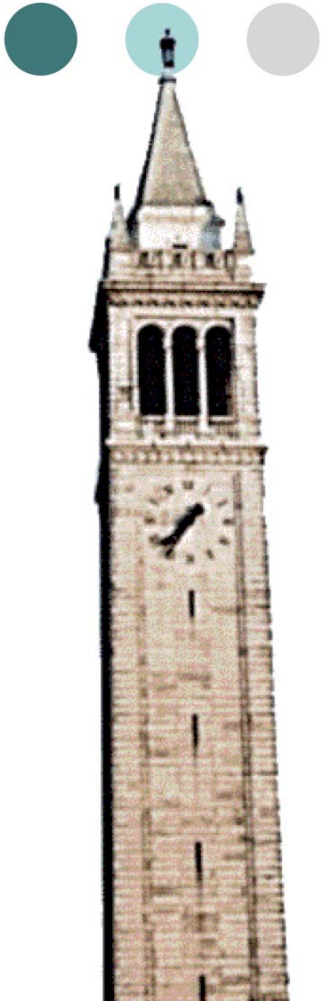


Architectures with Repeatable Timing for Cyber-Physical Systems

Edward A. Lee

Robert S. Pepper Distinguished Professor, UC Berkeley

and

- *Stephen A. Edwards (Columbia University)*
- *Sungjun Kim (Columbia University)*
- *Isaac Liu (UC Berkeley)*
- *Hiren D. Patel (Waterloo)*
- *Martin Schoeberl (Vienna University of Technology)*

Invited Keynote Talk

Workshop on Cyber-Physical Systems (part of ES Week)

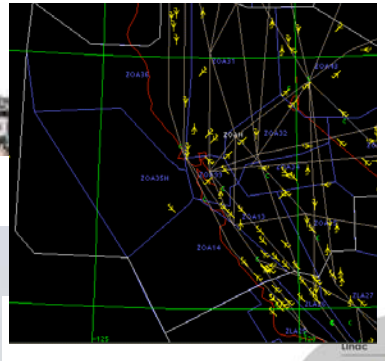
Grenoble, France, October 16, 2009

With thanks to NSF for sponsorship of this work.

Cyber-Physical Systems (CPS):

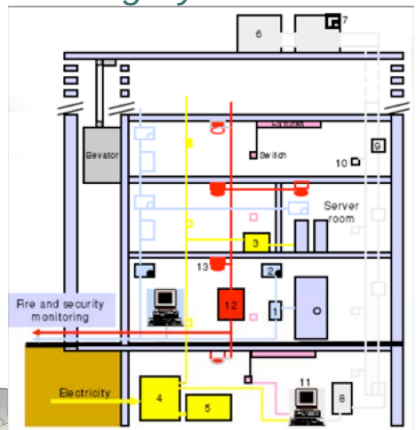
Orchestrating networked computational resources with physical systems

Avionics



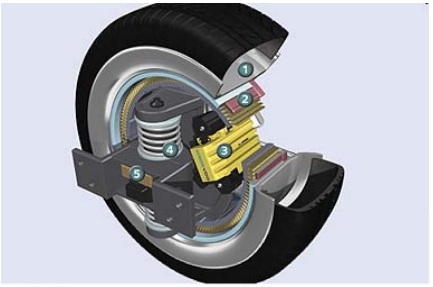
Transportation (Air traffic control at SFO)

Building Systems

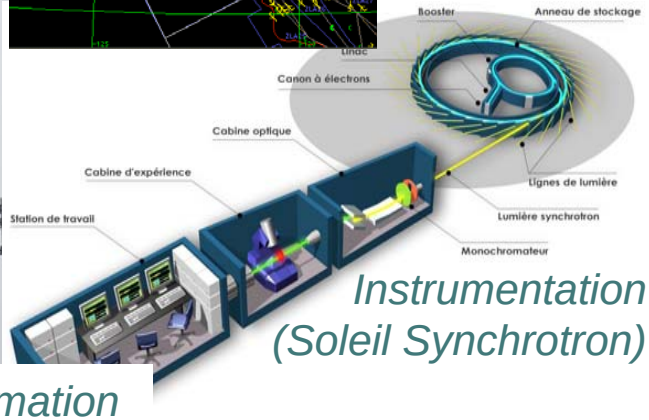


Telecommunications

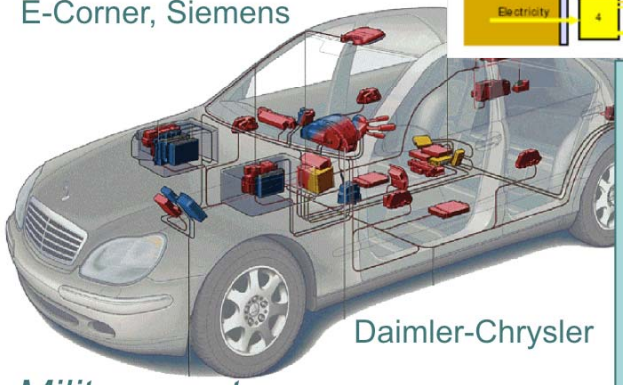
Automotive



E-Corner, Siemens

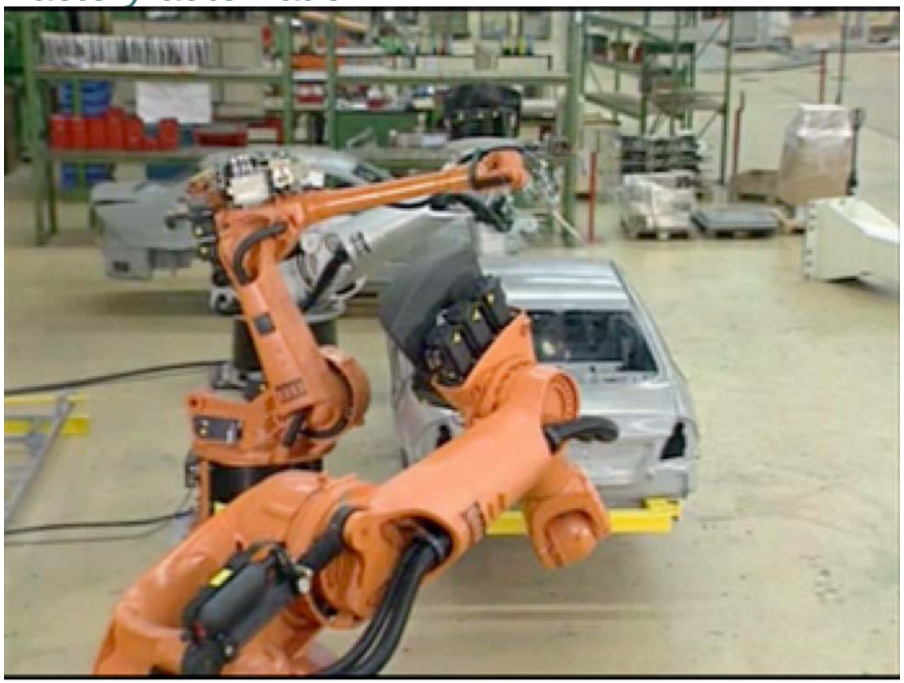


Instrumentation (Soleil Synchrotron)



Daimler-Chrysler

Factory automation

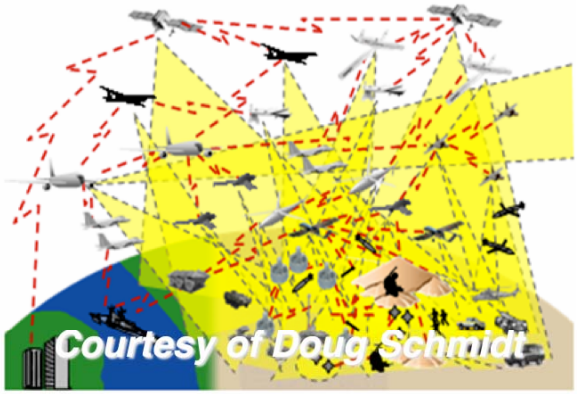


Courtesy of Kuka Robotics Corp.

Power generation and distribution

Courtesy of General Electric

Military systems:



Where CPS Differs from the traditional embedded systems problem:

- *The traditional embedded systems problem:*

  Embedded software is software on small computers. The technical problem is one of optimization (coping with limited resources).

- *The CPS problem:*

  Computation and networking integrated with physical processes. The technical problem is managing dynamics, time, and concurrency in networked computational + physical systems.

This is not about small systems!

A Key Challenge on the Cyber Side: Real-Time Software

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

Timing of programs is not repeatable, except at very coarse granularity.

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

Techniques Exploiting the Fact that Time is Irrelevant

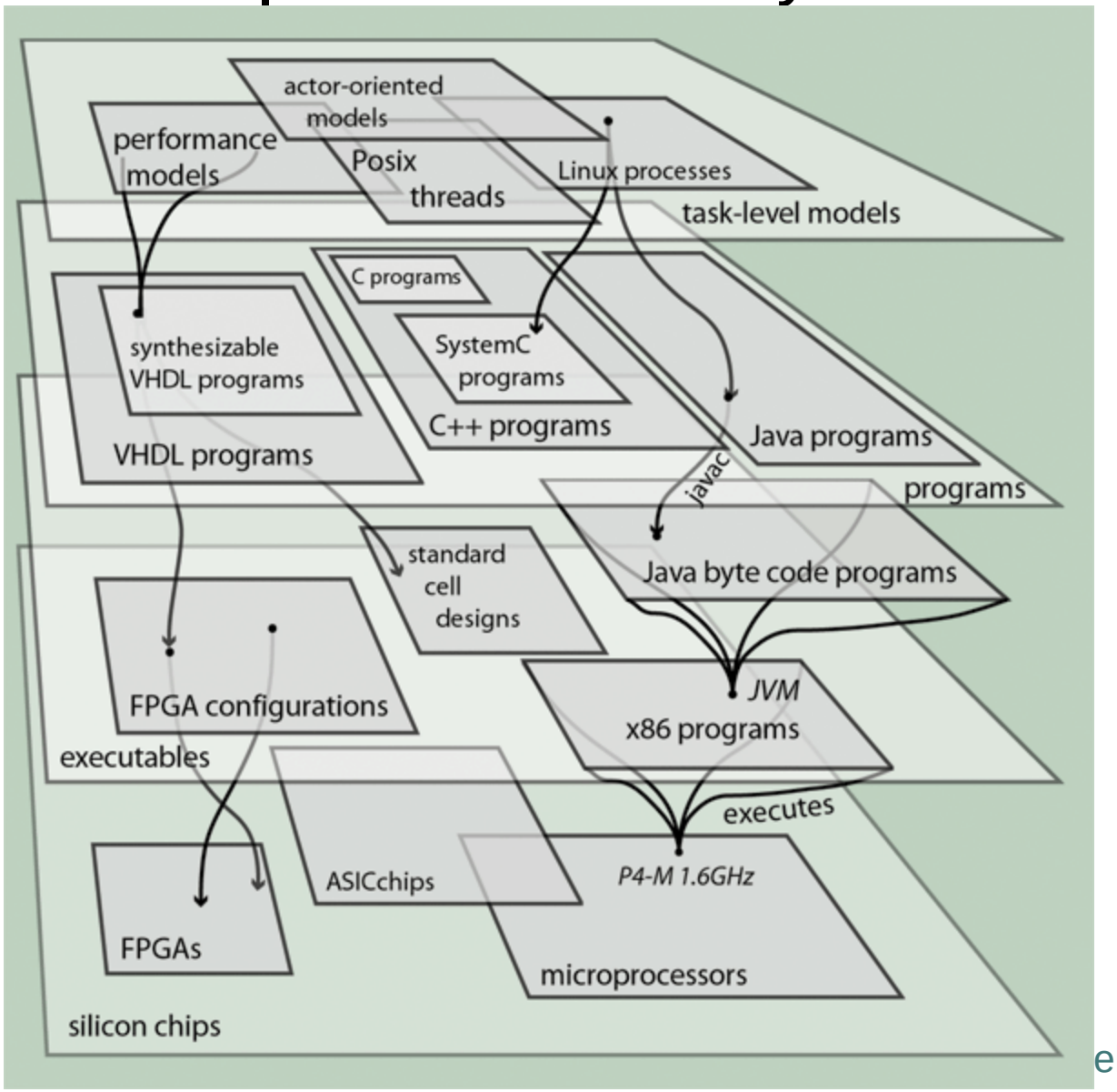
- Programming languages
- Virtual memory
- Caches
- Dynamic dispatch
- Speculative execution
- Power management (voltage scaling)
- Memory management (garbage collection)
- Just-in-time (JIT) compilation
- Multitasking (threads and processes)
- Component technologies (OO design)
- Networking (TCP)
- …

A Story

A “fly by wire” aircraft, expected to be made for 50 years, requires a 50-year stockpile of the hardware components that execute the software.

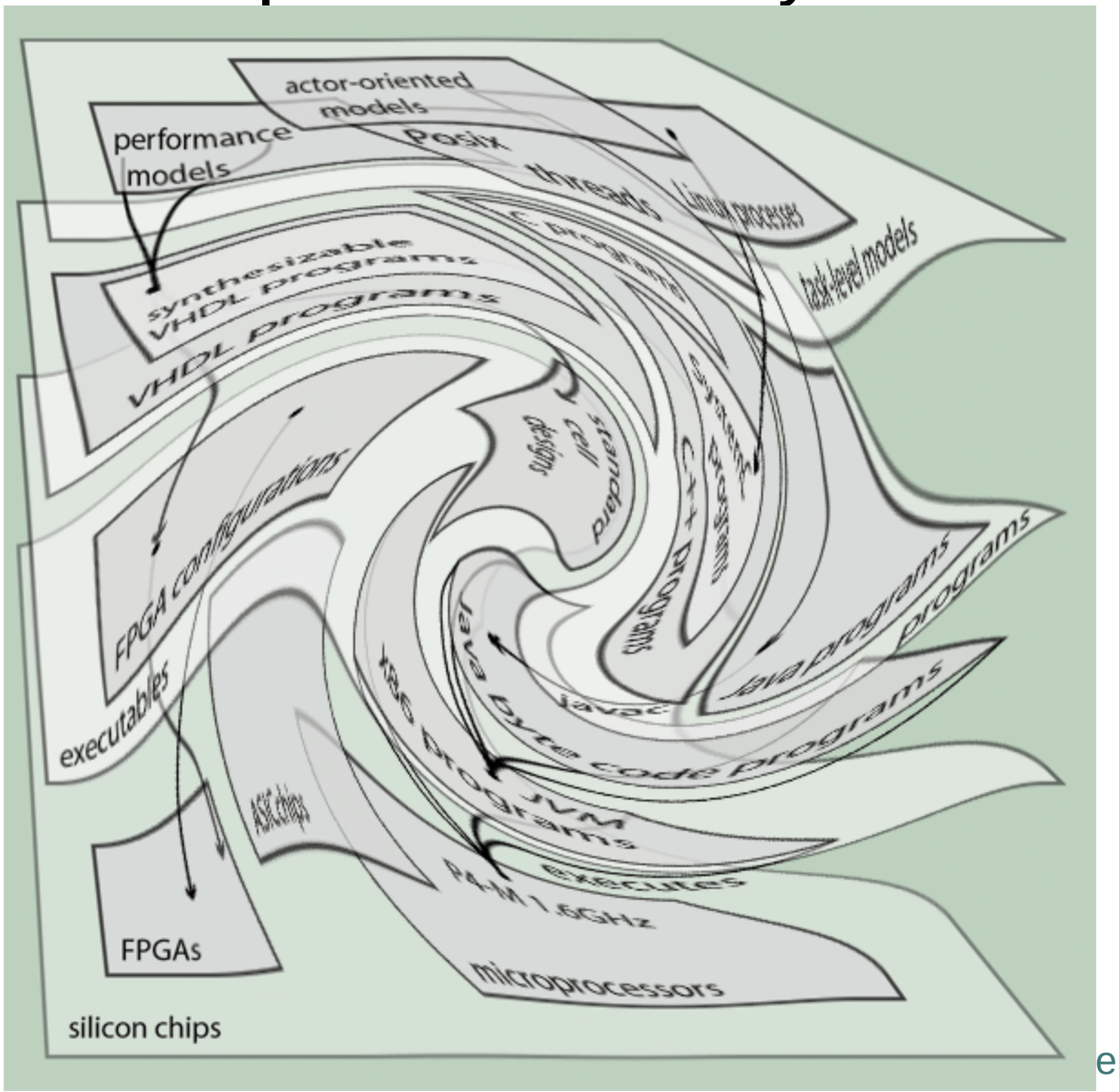
All must be made from the same mask set on the same production line. Even a slight change or “improvement” might affect timing and require the software to be re-certified.

Abstraction Layers



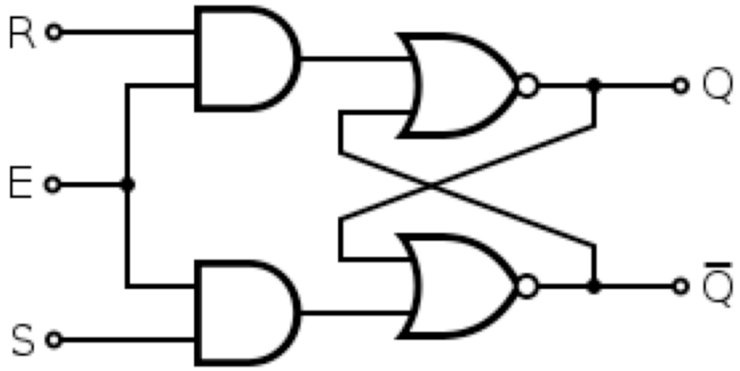
The purpose for an abstraction is to hide details of the implementation below and provide a platform for design from above.

Abstraction Layers



Every abstraction layer has failed for time-sensitive applications.

Is the problem intrinsic in the technology?



Electronics technology delivers highly repeatable and precise timing…

20.000 MHz (± 100 ppm)

… and the overlaying software abstractions discard it.

Case in point: What is an ISA?

- A collection of instructions.
- Each one changes the state of the processor in a well-defined way.
- **The *ISA strong guarantee***:
  - Given a known initial state.
  - Execute a sequence of instructions.
  - Next execute an instruction that observes the processor state.
  - The observed state is equivalent to one produced by a sequential execution of exactly every instruction that preceded it in the sequence.
- Architects are very clever at preserving this guarantee without precisely doing sequential execution.
- And the guarantee says nothing about timing.

Definitions

- **Correct execution**: preserves semantics (strong guarantee).
- **Repeatable property** of a program: every correct execution has the property, given the same inputs (this requires a model of "inputs").
- **Conventional Turing-Church** (**CTC**) computation:
  - inputs included in the initial state of the processor
  - sequence of instructions
  - outputs are included in the final state
- Outputs of a CTC computation are repeatable in today's processors
  - Note that before the IBM 360, even many CTC programs ran correctly on only one computer.

How Many Apps are Conventional Turing-Church (CTC) Computations?

- No multithreading.
- No I/O during execution.

How many applications?

... not many ...

Yet that's what we've designed computers to do!

Our stab at a solution: Precision-Time (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?)
- 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.

Timing in the ISA

Add to the strong guarantee:

- Repeatable timing of each instruction.

  This need not be fixed across realizations of the ISA, but it must be specified for each realization, so that tools can analyze timing.

```
deadi $t0, 10
...                    Block 1
deadi $t0, 8
...                    Block 2
deadi $t0, 0
...
```

Add timing instructions:

- Force a block to take a minimum amount of time.
- Branch and/or exception on exceeding this minimum.

C Code with Exceptions

```
int main(){
  ...
  tryin (500ms) {
    while (true) {
      x = f(x);
    }
  } catch {
    printf("Converged: %d\n", x);
  }
  ...
}
```

Our stab at a solution: Precision-Time (PRET) Machines

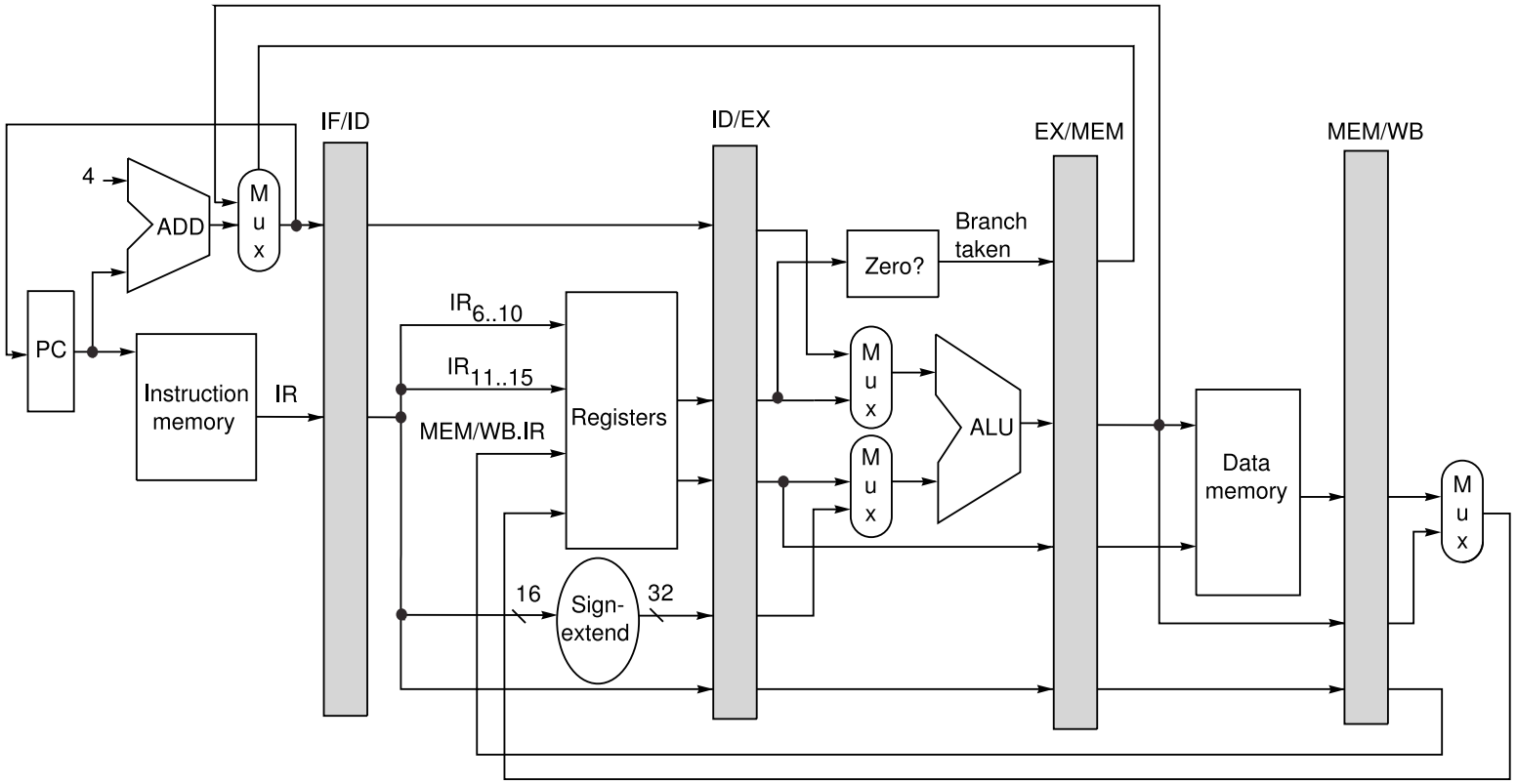
Make temporal behavior as important as logical function.

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?)
- Precision networks (TTA? Time synchronization?)

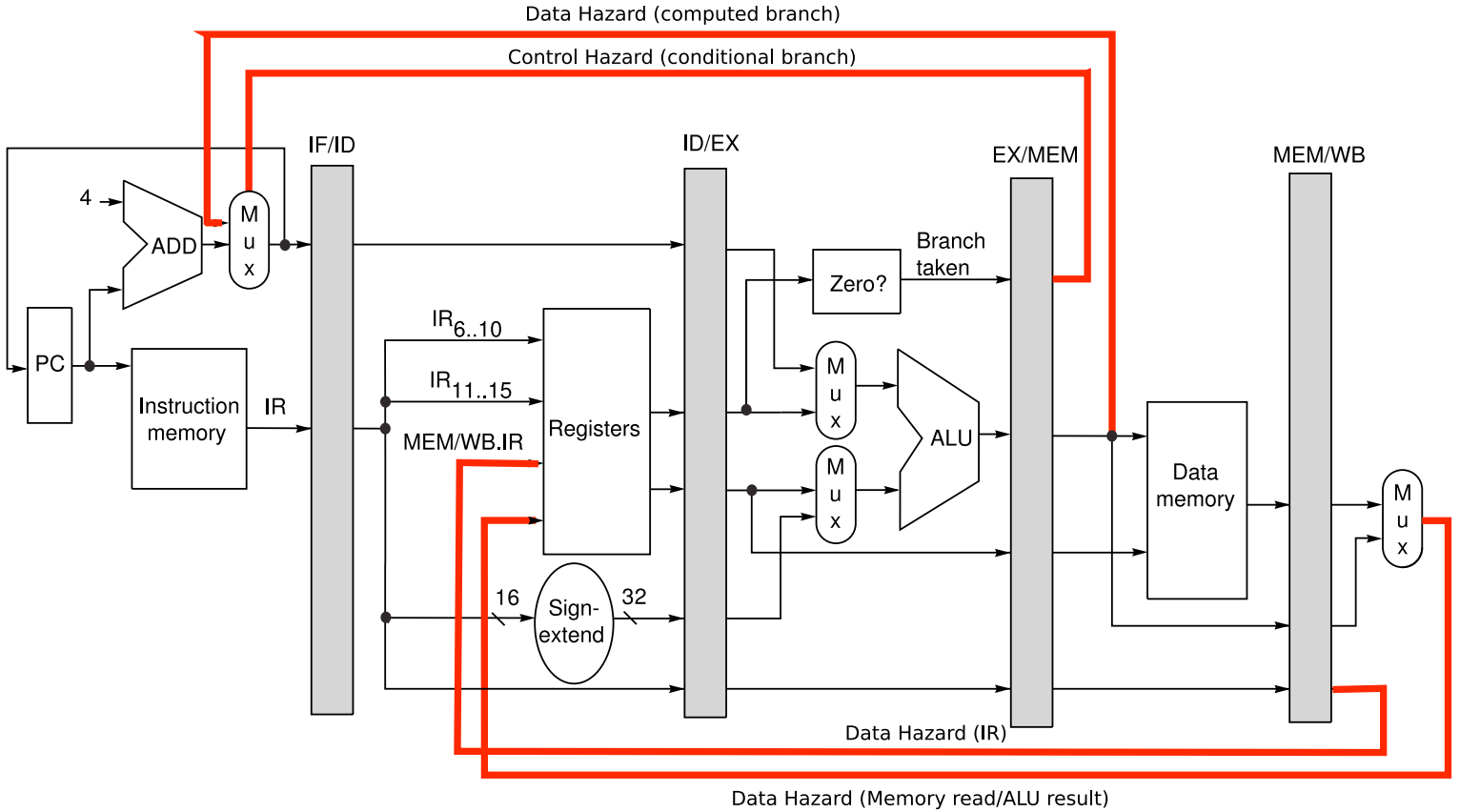
Edwards and Lee, "**The Case for the Precision Timed (PRET) Machine**," *Wild and Crazy Ideas* Track, *Design Automation Conference* (DAC), June 2007.

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.

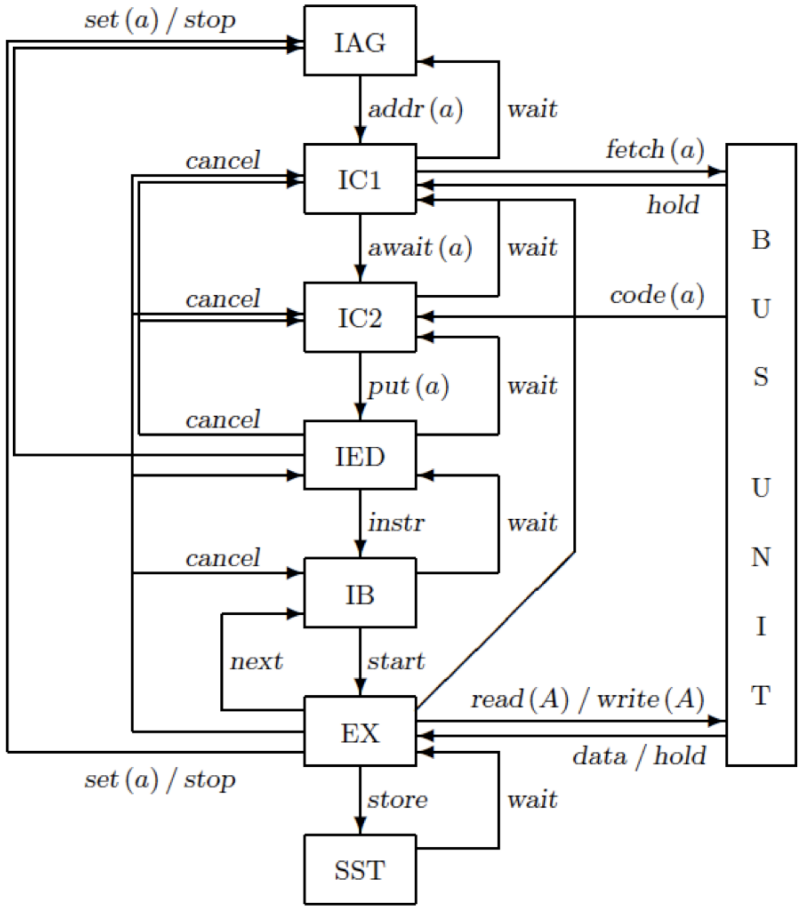
Forwarding reduces stalls, but complicates hardware, and makes timing non-repeatable and hard to analyze.

Example execution time analysis of:
- Motorola ColdFire
- Two coupled pipelines (7-stage)
- Shared instruction & data cache
- Artificial example from Airbus
- Twelve independent tasks
- Simple control structures
- Cache/Pipeline interaction

leads to large integer linear programming problem

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

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

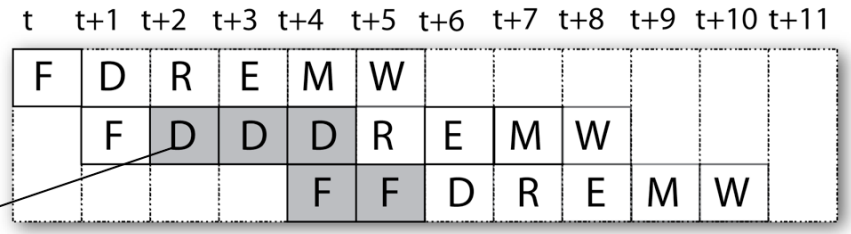


C. Ferdinand et al., “Reliable and precise WCET determination for a real-life processor.” EMSOFT 2001.

An Alternative: Pipeline Interleaving

Traditional pipeline:

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

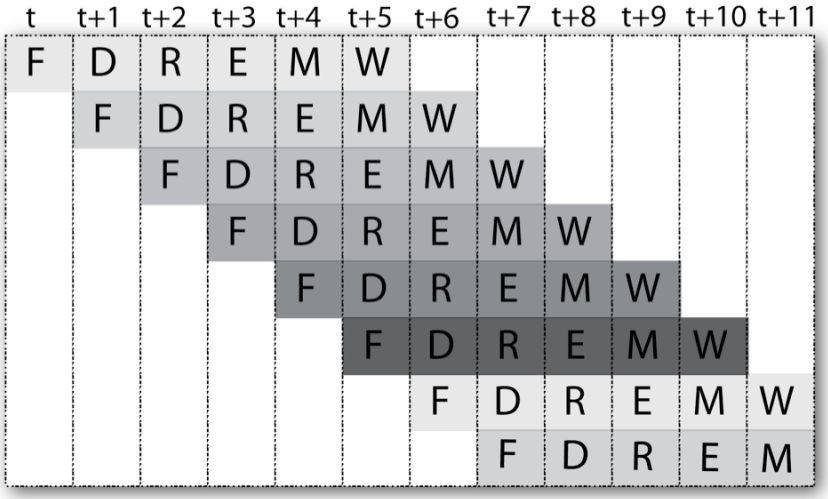


Stall pipeline

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
```

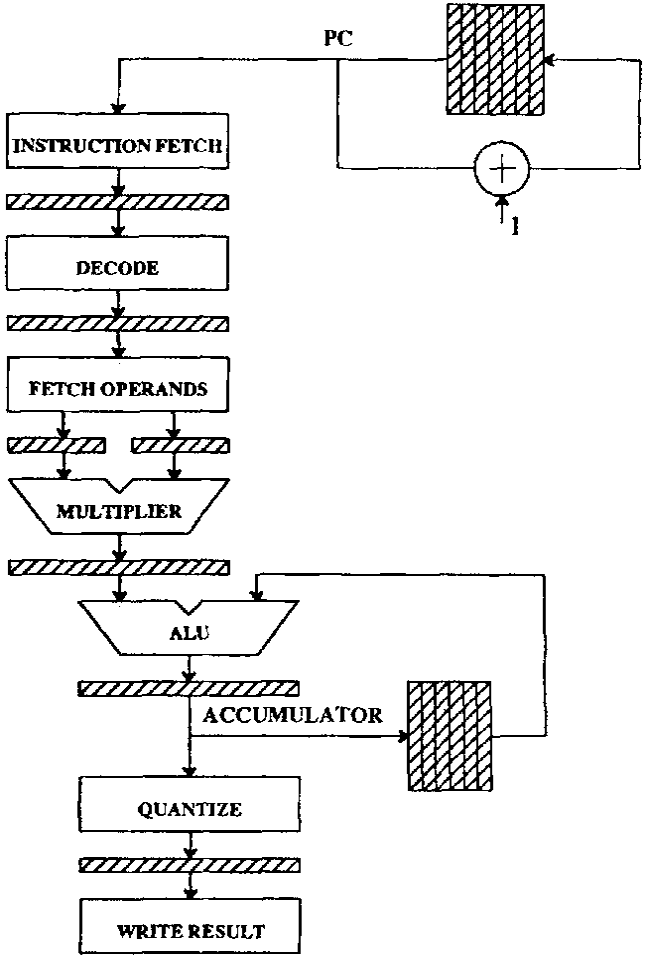


Repeatable timing behavior of instructions

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.***

Our stab at a solution: Precision-Time (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?)
- Precision networks (TTA? Time synchronization?)

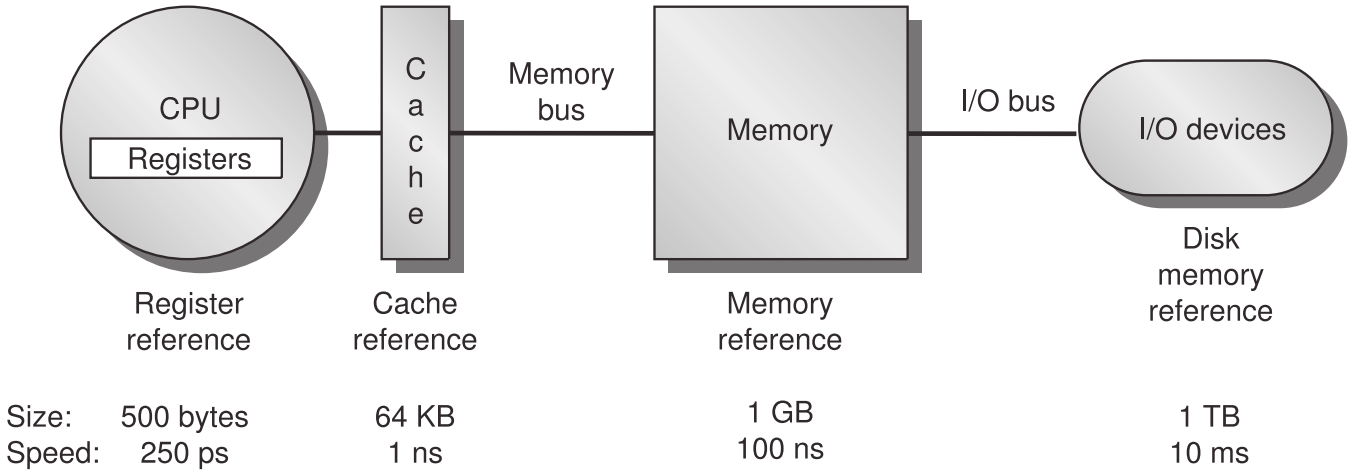
Edwards and Lee, "**The Case for the Precision Timed (PRET) Machine**," *Wild and Crazy Ideas* Track, *Design Automation Conference* (DAC), June 2007.

Forget the datapath…

“It’s the Memory, Stupid!”

R. Sites. *Microprocessor Report*, Aug. 1996.

Memory Hierarchy



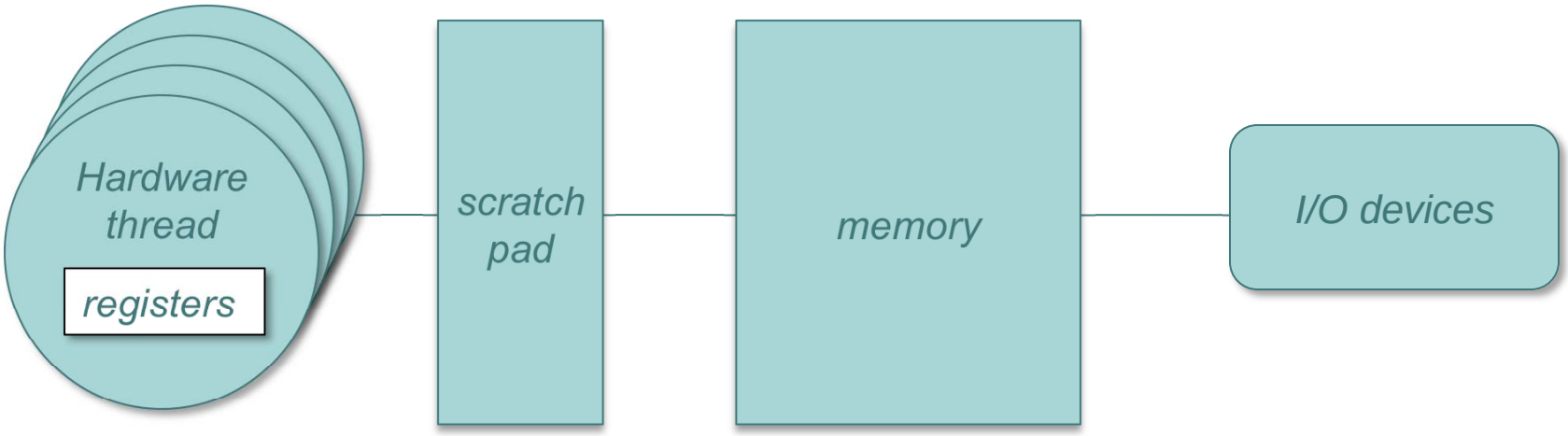
Hennessey and Patterson, Computer Architecture: A Quantitative Approach, 4th edition, 2007.

- 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?*

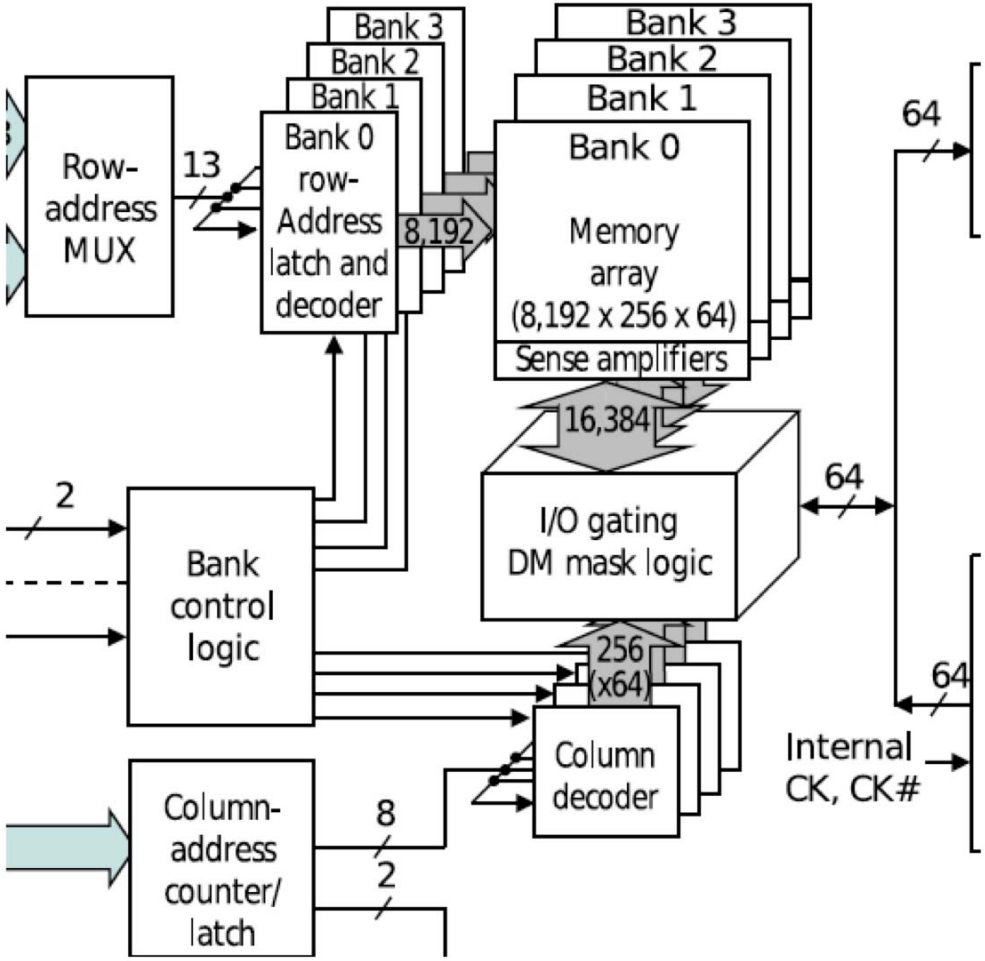
Separation of concerns.

PRET principle: any temporary memory is under program control.

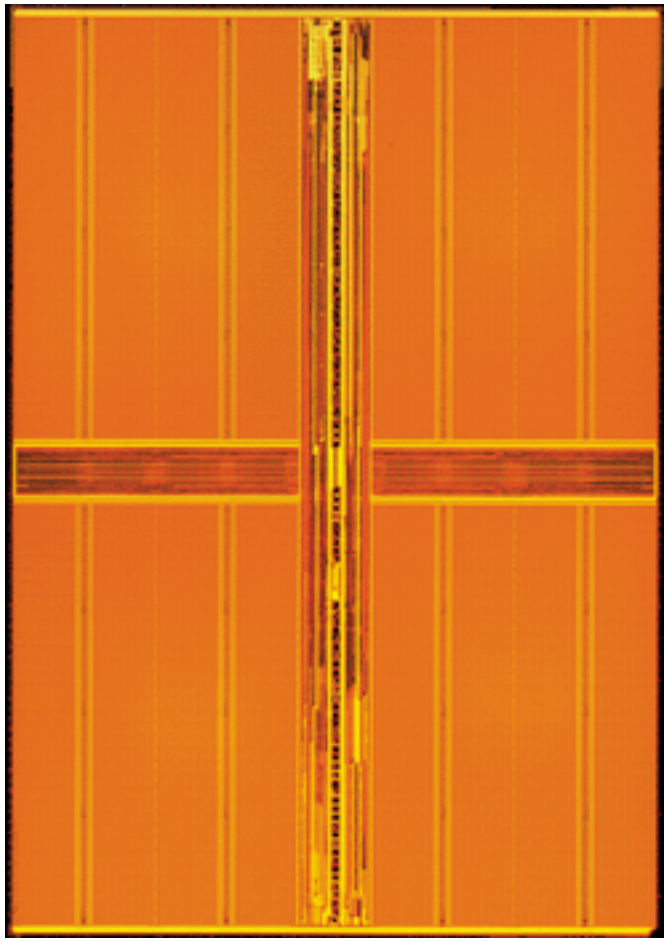
One Possible PRET Architecture



What about Main Memory? Modern DRAMs:

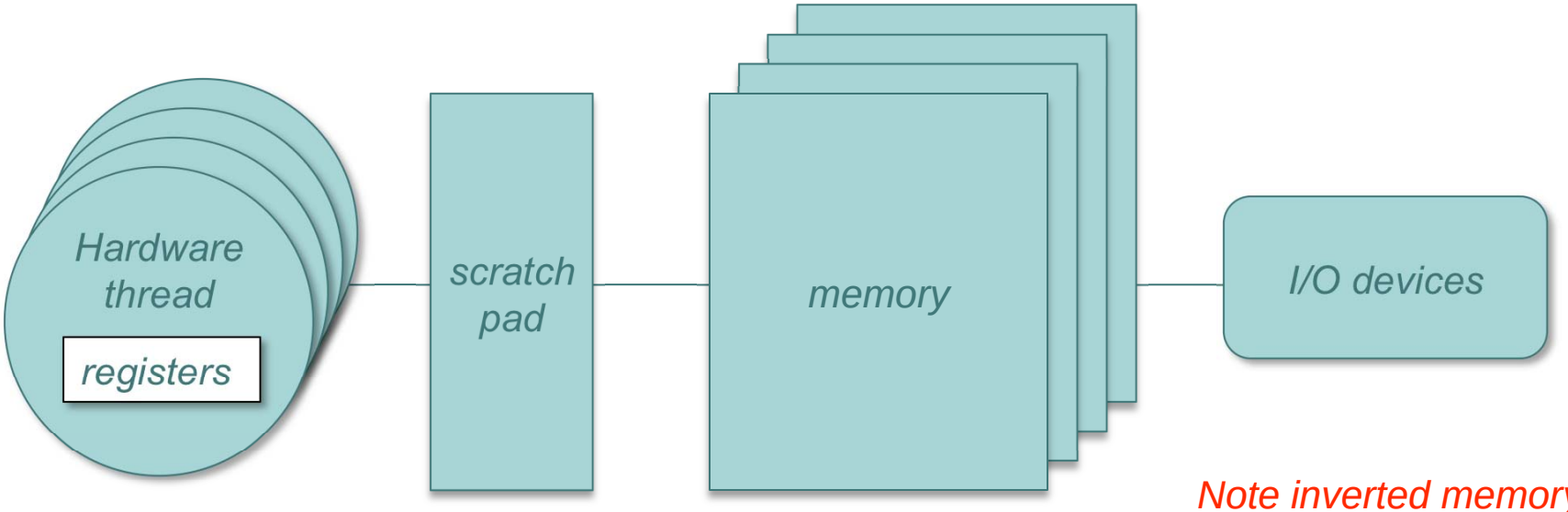


Micron corp.



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

One Possible 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!

A Few of the (Many) Remaining Challenges and Opportunities

- DRAM designs today foil timing repeatability even with private banks (e.g. write-after-read latencies)
- Interleaved pipelines may not be the best choice for power optimization
- How to expose timing properties in programming models (completely absent in today's languages)
- Need I/O mechanisms that do not disrupt repeatable timing
- Multicore networks-on-chip may benefit dramatically from repeatable timing
- …

Conclusion

- Repeatable timing could be very useful
- Requires *timed semantics in the ISA*, and a different approach to:
  - pipelining,
  - memory hierarchy,
  - multicore
  - I/O
  - networking
  - programming models

Full employment for computer architects!

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