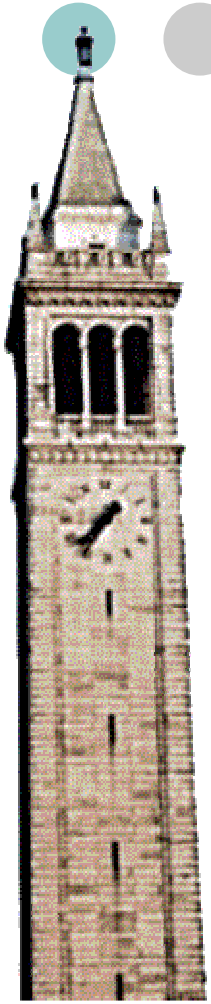


Precision-Timed (PRET) Machines

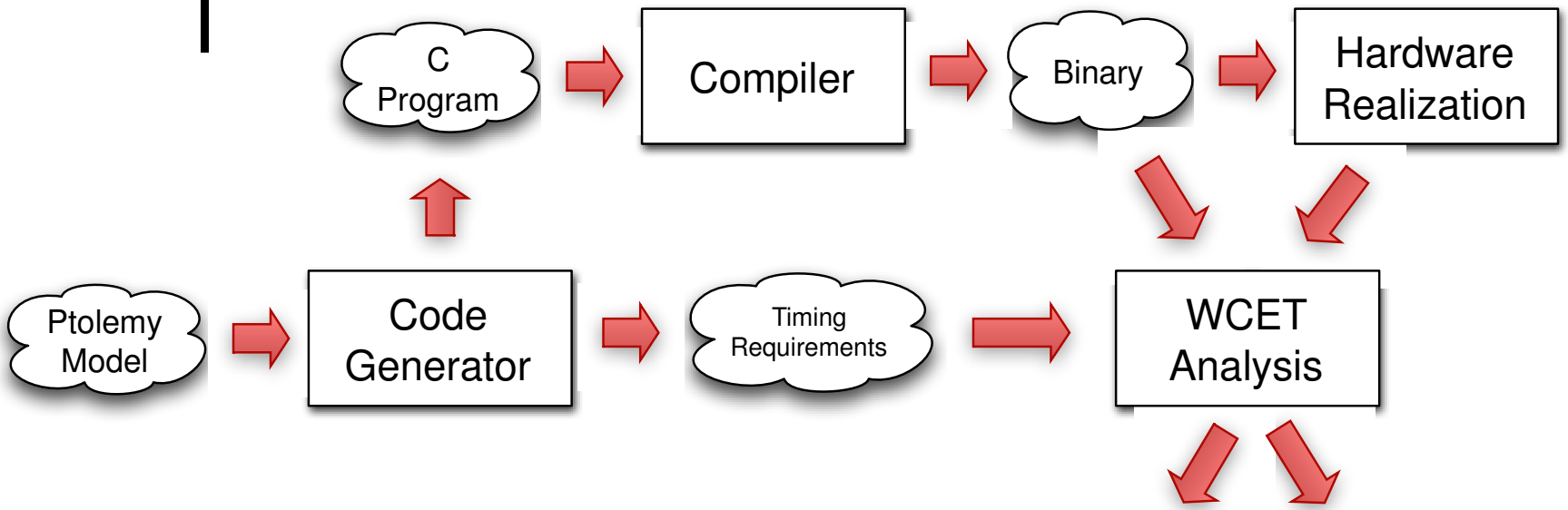
Stephen A. Edwards	Columbia University
Sungjun Kim	Columbia University
Edward A. Lee	UC Berkeley
Ben Lickly	UC Berkeley
Isaac Liu	UC Berkeley
Hiren D. Patel	University of Waterloo
Jan Reineke <speaker>	UC Berkeley

Designing Next-Generation Real-Time Streaming Systems

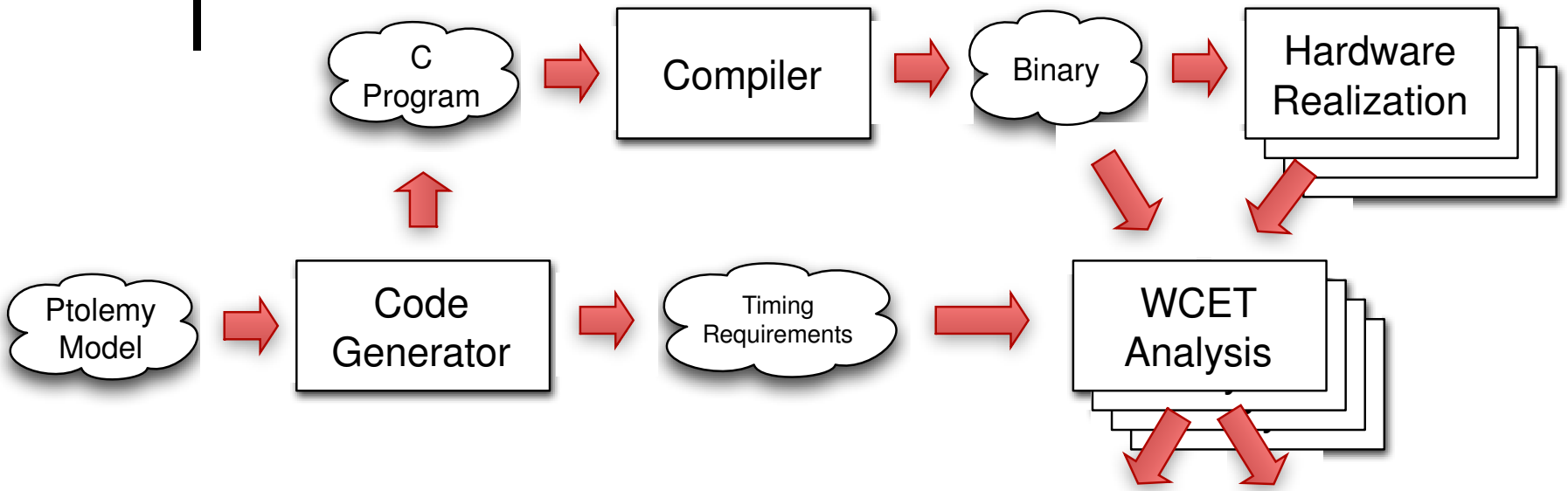
Tutorial at ESWEEK 2011

Taipei, Taiwan, October 9th, 2011

Current Timing Verification Process



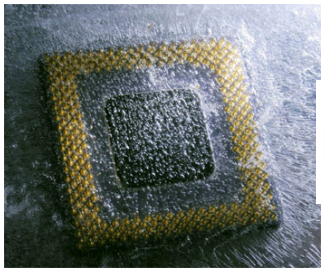
Current Timing Verification Process



- New Architecture ➔ New Analysis & Recertification
- For modern architectures:

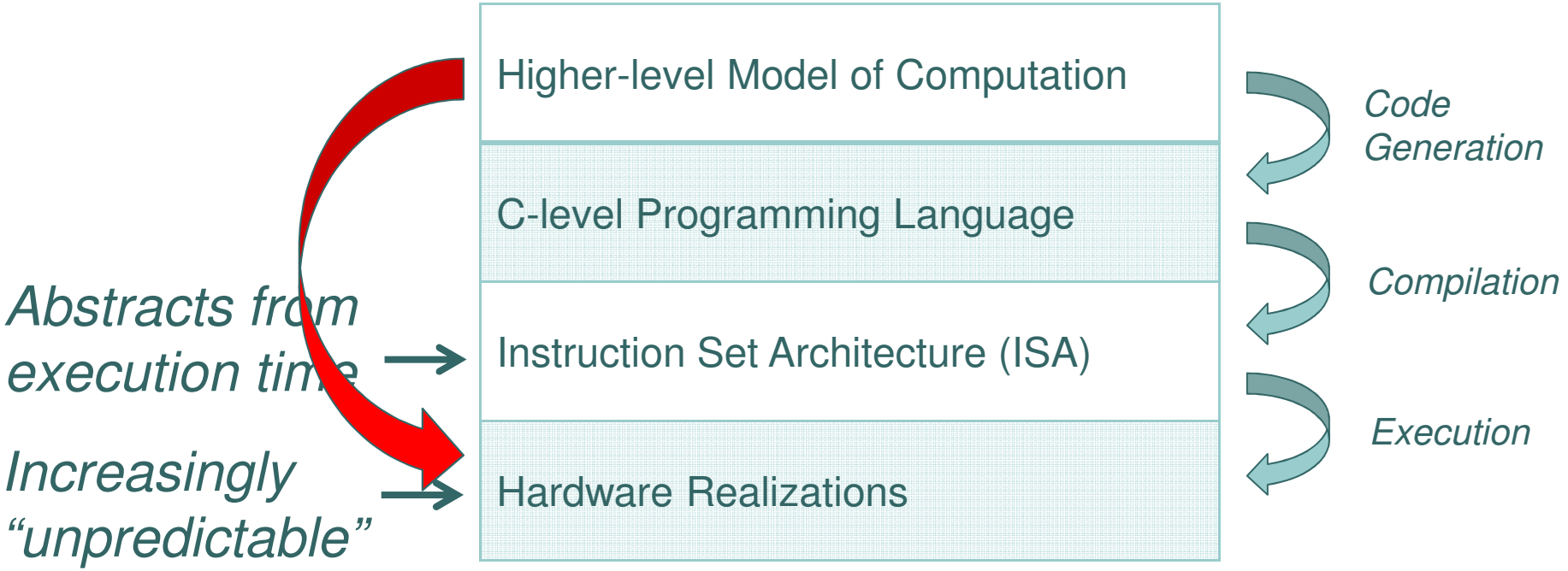
Extremely time-consuming, costly and error-prone

Boeing: 40 years supply of



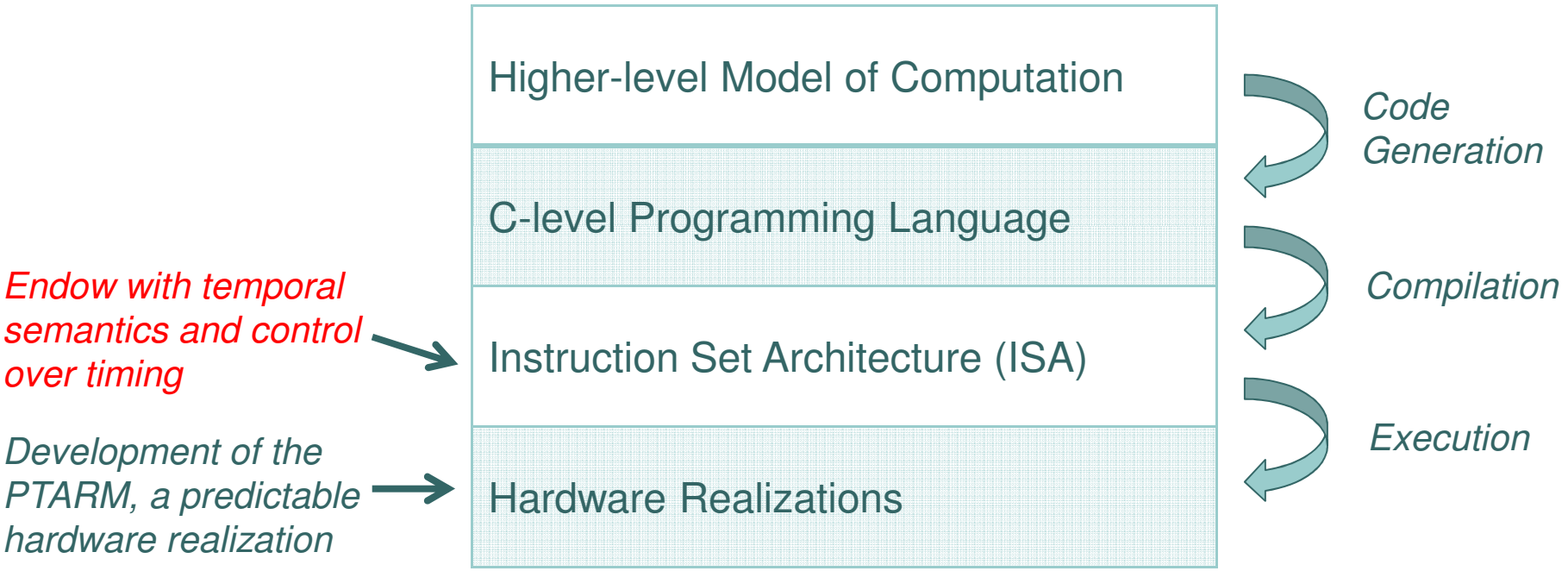
Reineke et al., Berkeley 3

Lack of Suitable Abstractions for Real-Time Systems



Agenda of this PRET (and this presentation)

Agenda of this PRET (and this presentation)

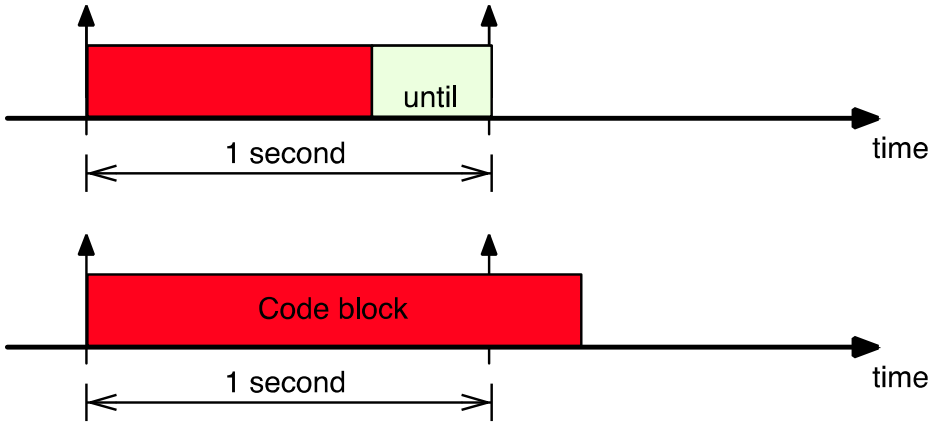


Adding Control over Timing to the ISA

Capability 1: “delay until”

Some possible capabilities in an ISA:

- [C1] Execute a block of code taking at least a specified *time*



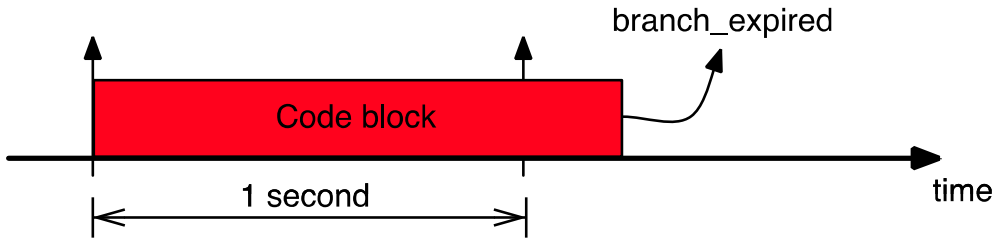
Where could this be useful?

- Finishing early is not always better:
  - Scheduling Anomalies (Graham’s anomalies)
  - Communication protocols may expect periodic behavior
  - …

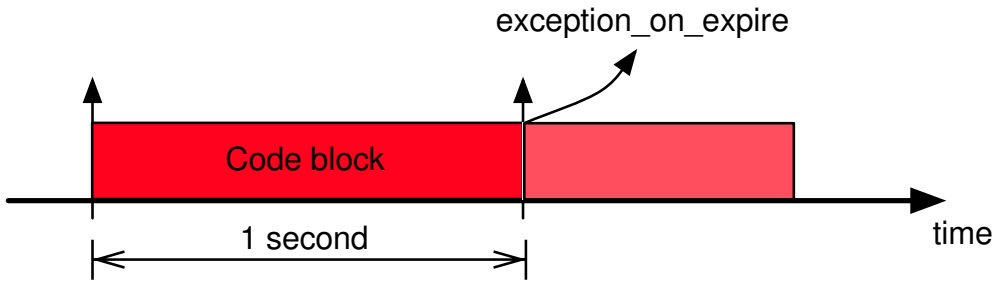
Adding Control over Timing to the ISA

Capabilities 2+3: “late” and “immediate miss detection”

- [C2] Conditionally branch if the specified *time* was exceeded.



- [C3] If the specified *time* is exceeded during execution of the block, branch immediately to an exception handler.



Applications of Variants 2+3

“late” and “immediate miss detection”

- [C3] “immediate miss detection”:
  - Runtime detection of missed deadlines to initiate error handling mechanisms
  - Anytime algorithms
  - However: unknown state after exception is taken
- [C2] “late miss detection”:
  - No problems with unknown state of system
  - Change parameters of algorithm to meet future deadlines

PRET Assembly Instructions Supporting these Four Capabilities

set_time %r, <val>

- loads current time + <val> into %r

delay_until %r

- stall until current time >= %r

branch_expired %r, <target>

- branch to target if current time > %r

exception_on_expire %r, <id>

- arm processor to throw exception <id> when current time > %r

deactivate_exception <id>

- disarm the processor for exception <id>

Controlled Timing in Assembly Code

[C1] Delay until:

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

[C2] Late miss detection

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

[C3] Immediate miss detection

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

MTFD – Meet the F(inal) Deadline

- Capability [C1] ensures that a block of code takes **at least** a given time.
- [C4] "MTFD": Execute a block of code taking **at most** the specified time.

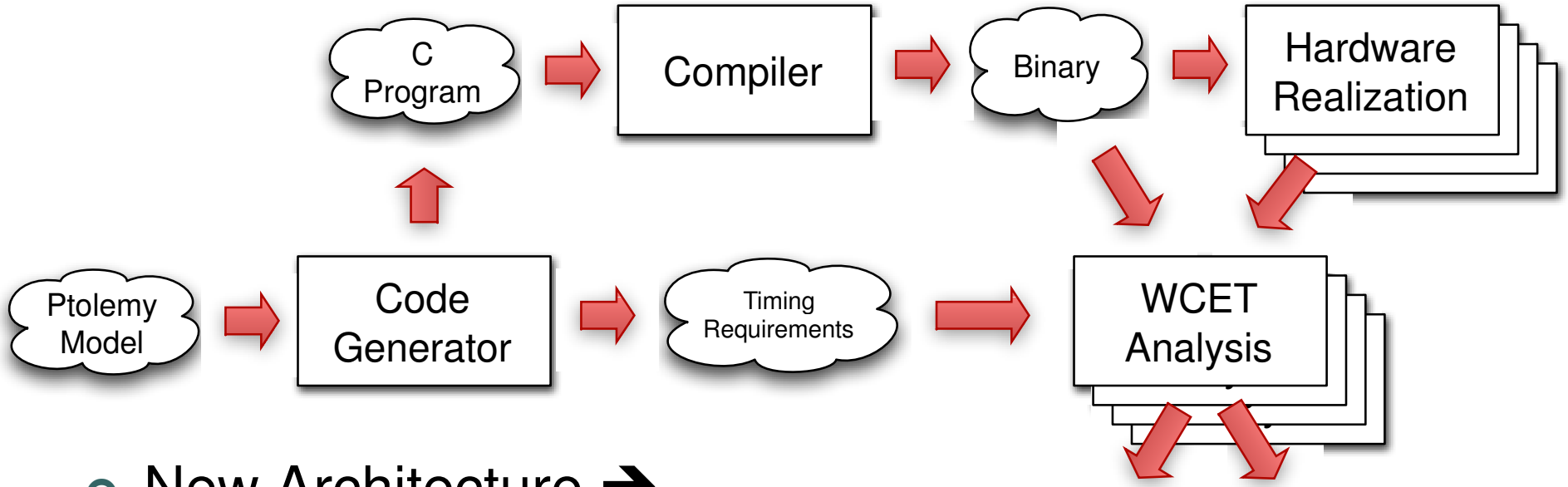
Being arbitrarily "slow" is always possible and "easy".

But what about being "fast"?

[C4] Exact execution:

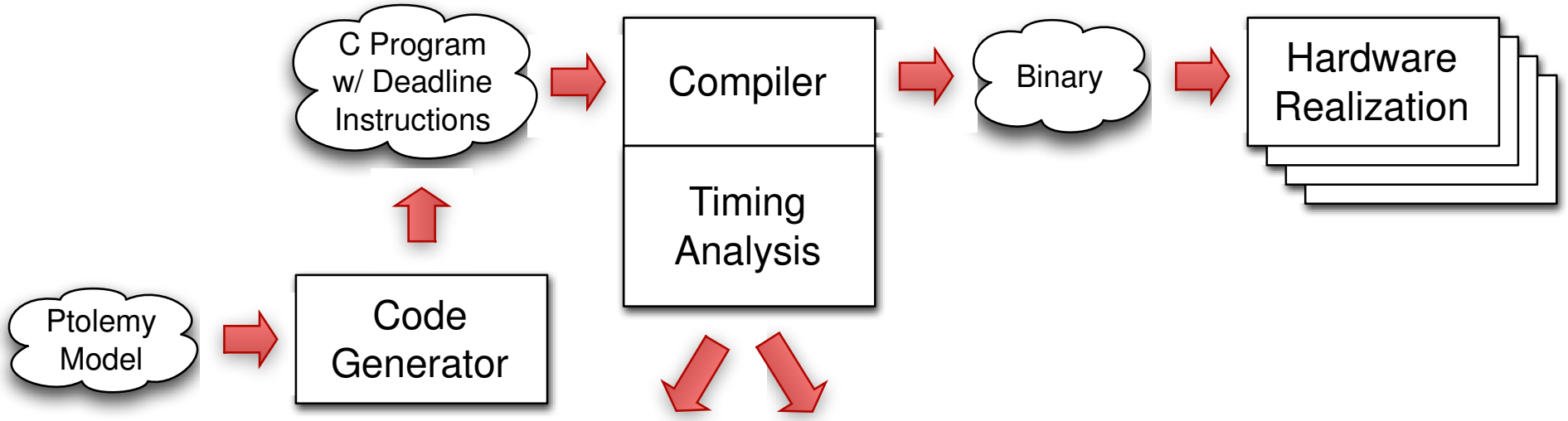
```
set_time r1, 1s
// Code block
MTFD r1
delay_until r1
```

Current Timing Verification Process



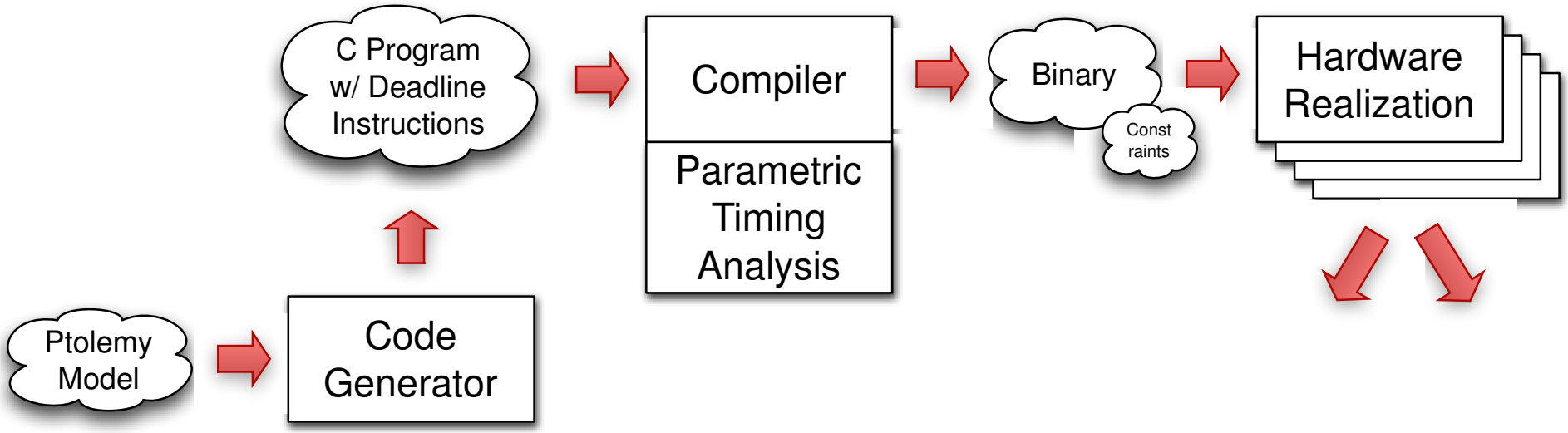
- New Architecture ➔ Recertification
- Extremely time-consuming and costly

The Future Timing Verification Process



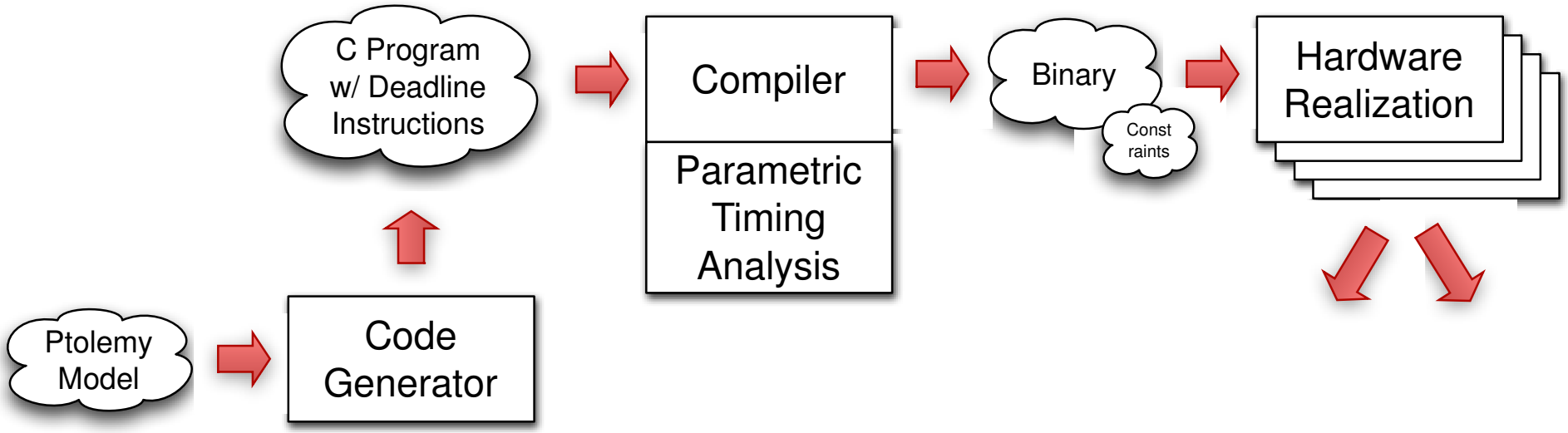
- Timing is property of ISA
- Compiler can check constraints once and for all
- Downside: little flexibility in development of hardware realizations

The Future Timing Verification Process: Flexibility through Parameterization



- ISA leaves more freedom to implementations through a parameterized timing model
- Compiler generates constraints on parameters which are sufficient to meet the timing constraints
- Parametric timing analysis is ongoing work

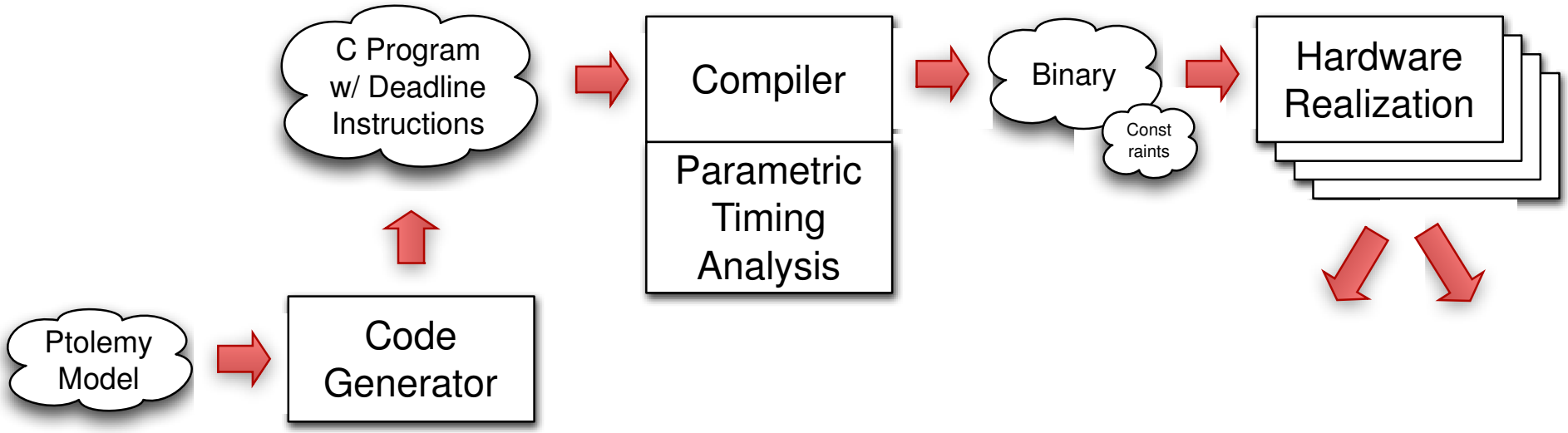
The Future Timing Verification Process: Flexibility through Parameterization



Possible parameters:

- Latencies of different components, such as the pipeline, scratchpad memory, main memory, buses
- Sizes of buffers, such as scratchpad memories or caches.

The Future Timing Verification Process: Flexibility through Parameterization



Challenge: Parameterization should allow for:

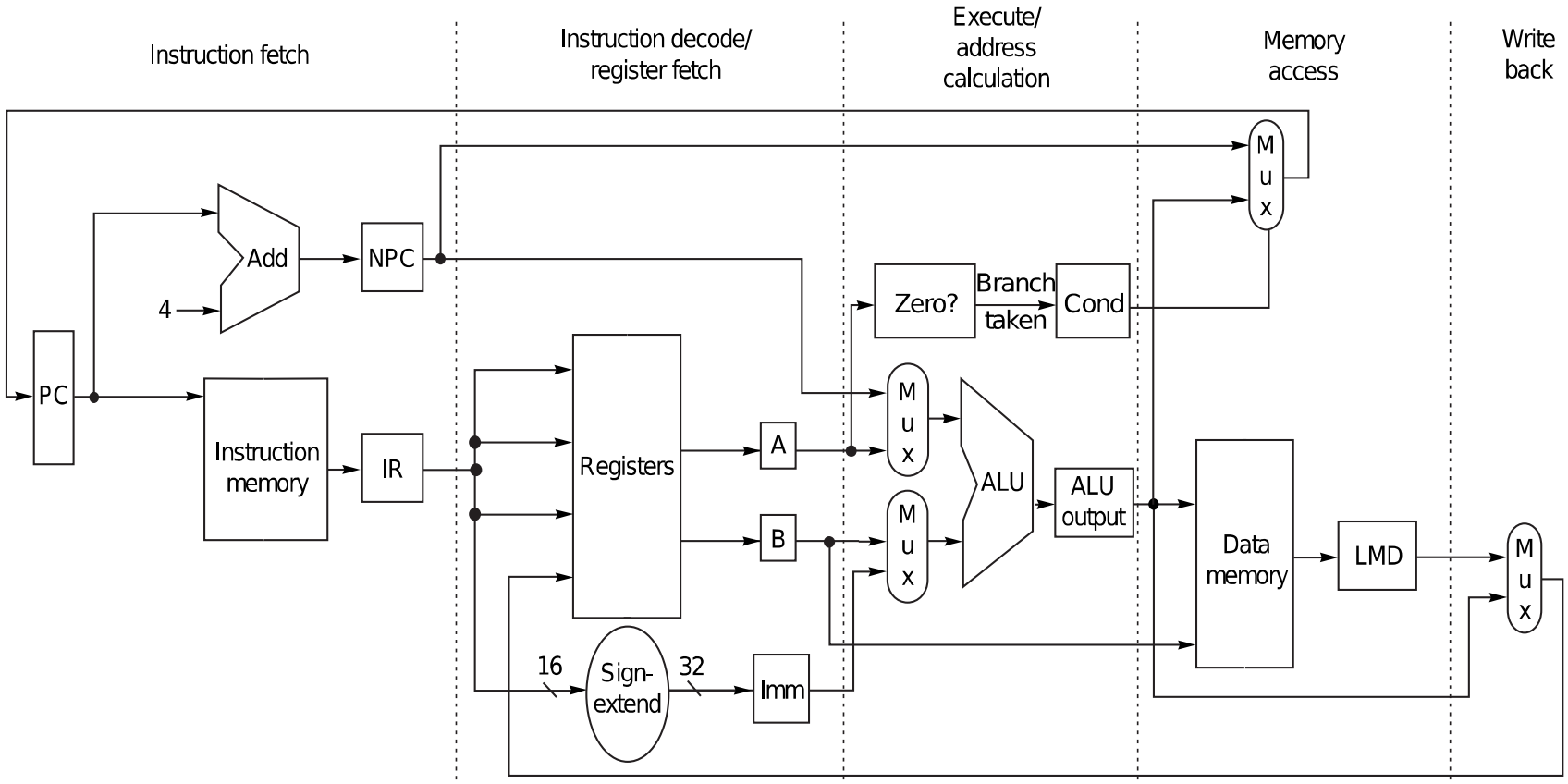
- Efficient and accurate parametric timing analysis, and
- Admit a wide variety of cost-efficient hardware realizations.

Agenda of this PRET (and this presentation)

Hardware Realizations: Challenges to deliver predictable timing

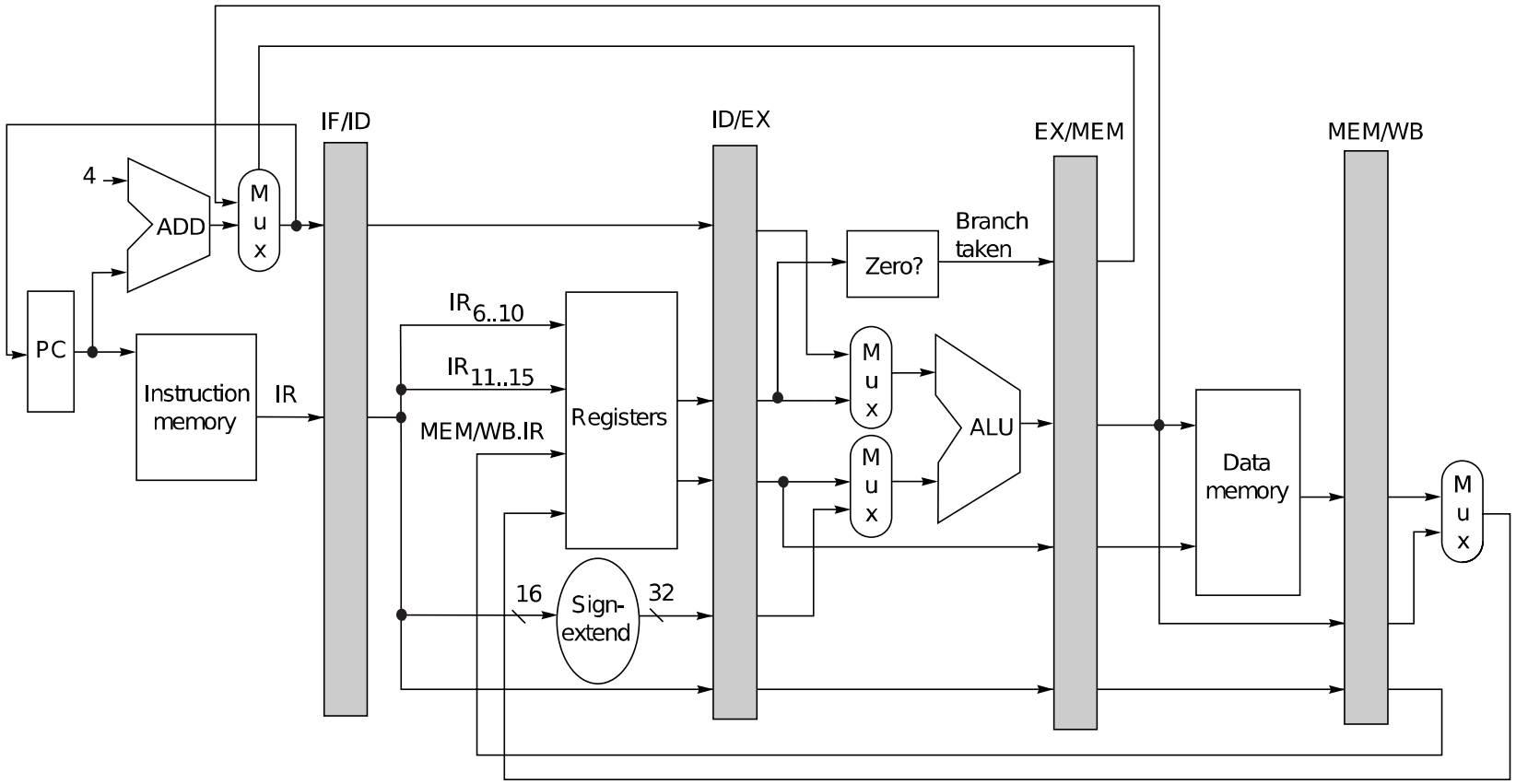
- Pipelining
- Memory hierarchy: Caches, DRAM
- On-chip communication
- I/O (DMA, interrupts)
- Resource sharing (e.g. in multicore architectures)

First Problem: Pipelining



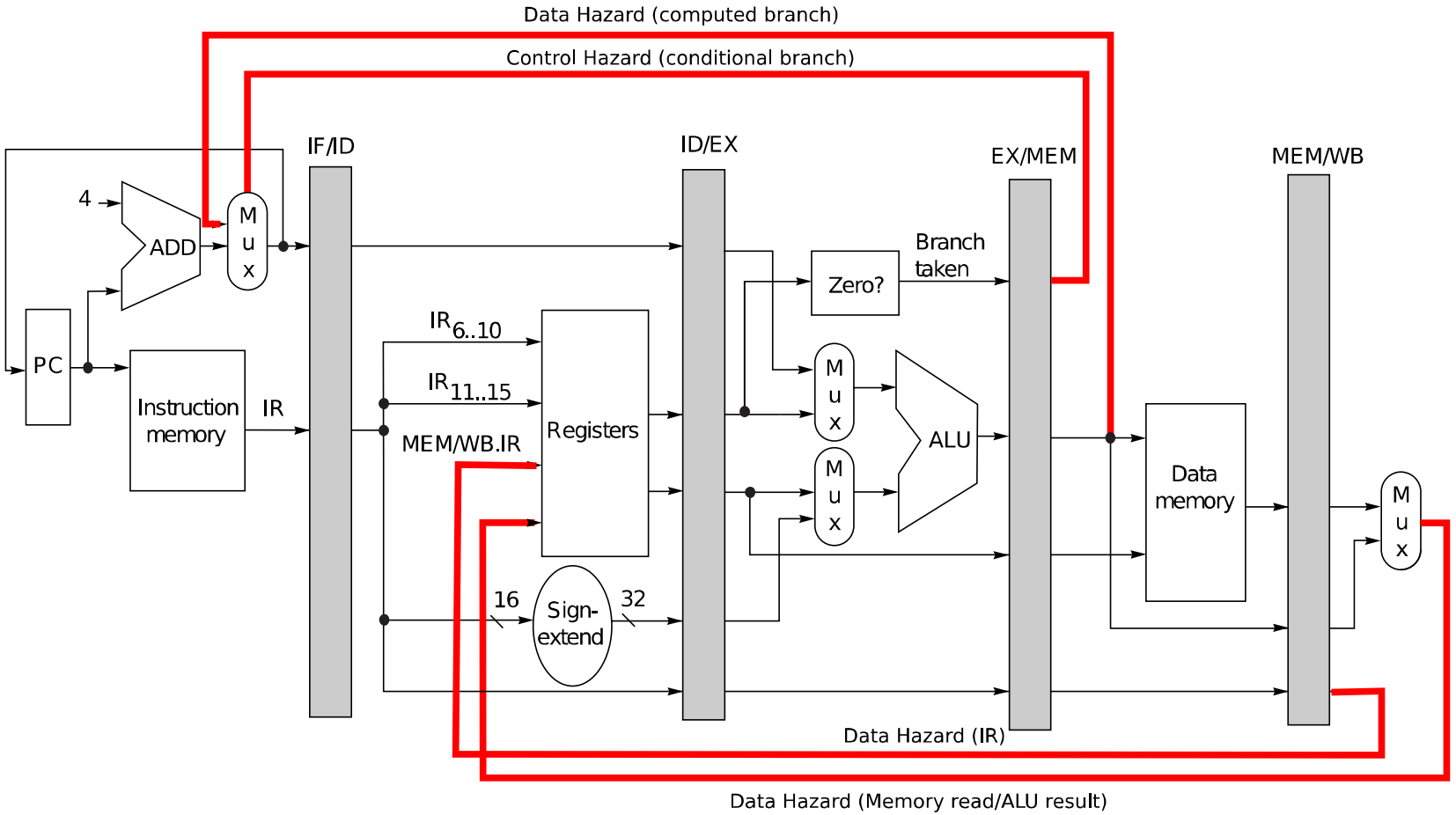
from Hennessy and Patterson, Computer Architecture: A Quantitative Approach, 2007.

First Problem: Pipelining



from Hennessy and Patterson, Computer Architecture: A Quantitative Approach, 2007.

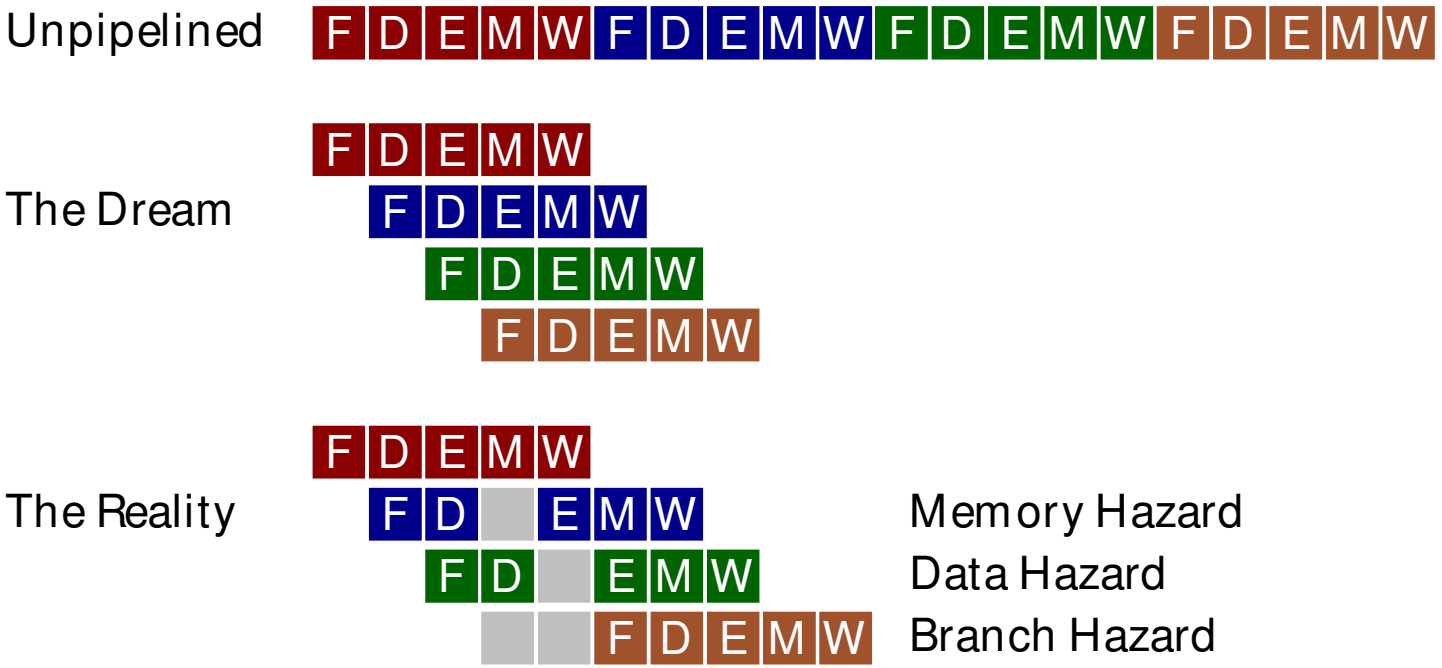
Pipelining: Hazards



from Hennessy and Patterson, Computer Architecture: A Quantitative Approach, 2007.

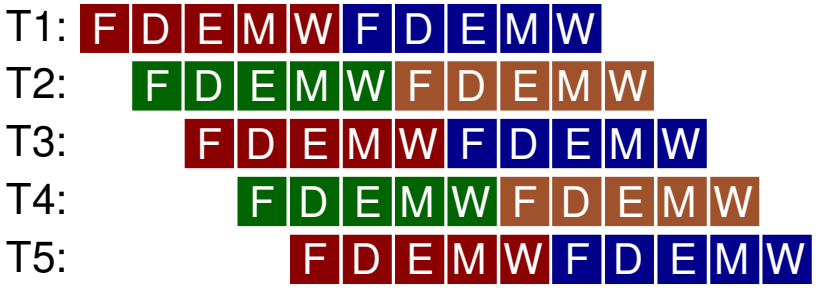
Forwarding helps, but not all the time…

```
LD R1, 45(r2)
DADD R5, R1, R7
BE R5, R3, R0
ST R5, 48(R2)
```



Our Solution: Thread-interleaved Pipelines

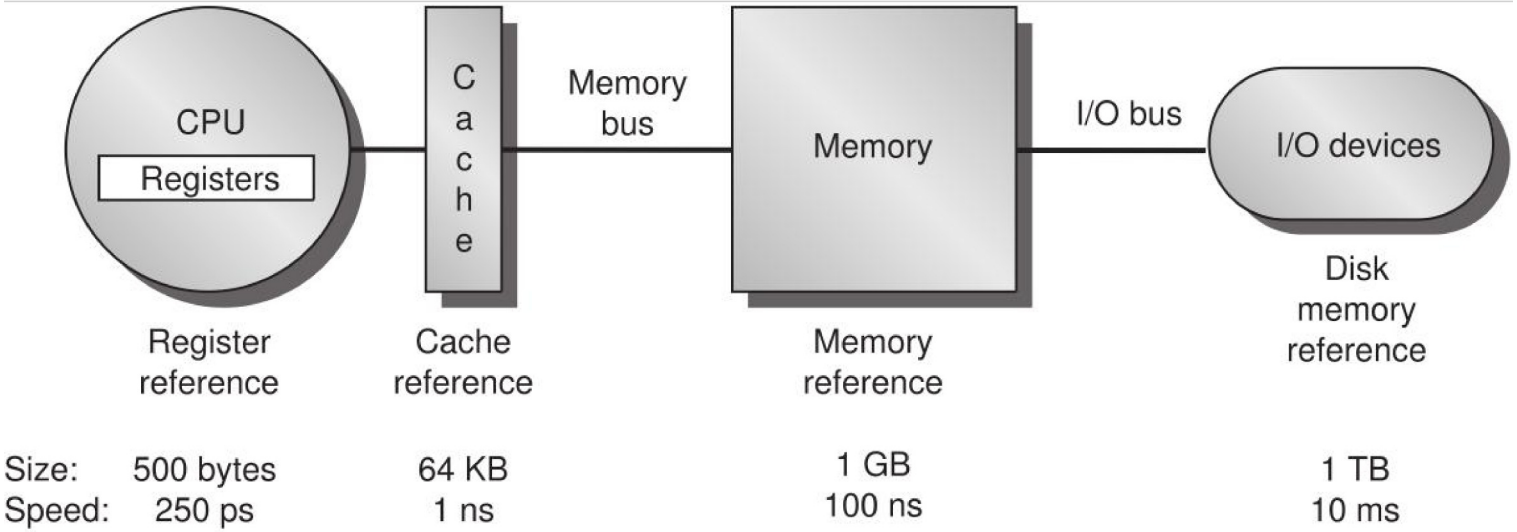
+



Each thread occupies only one stage of the pipeline at a time
→ No hazards; perfect utilization of pipeline
→ Simple hardware implementation (no forwarding, etc.)

Drawback: reduced single-thread performance

Second Problem: Memory Hierarchy

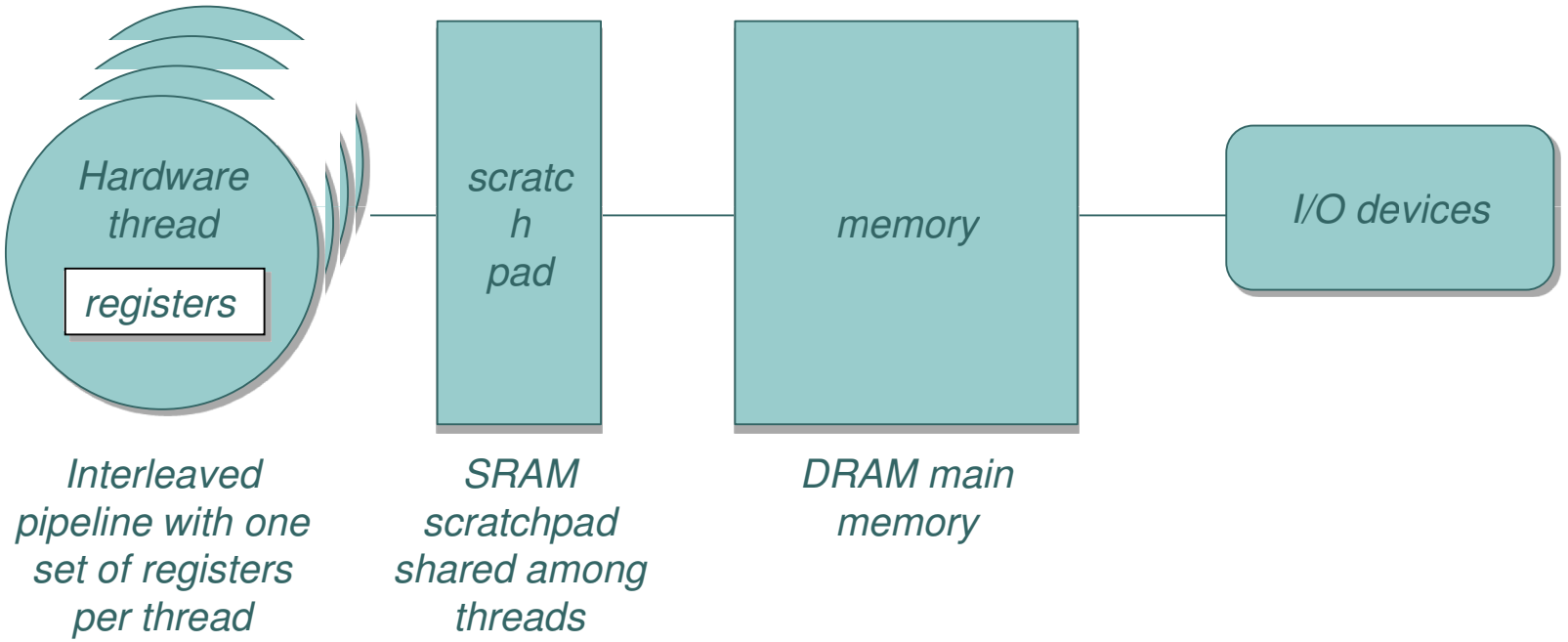


from Hennessy and Patterson, Computer Architecture: A Quantitative Approach, 2007.

- Register file is a temporary memory under program control.
- Cache is a temporary memory under hardware control.

PRET principle: any temporary memory is under program control.

PRET principles implies Scratchpad in place of cache

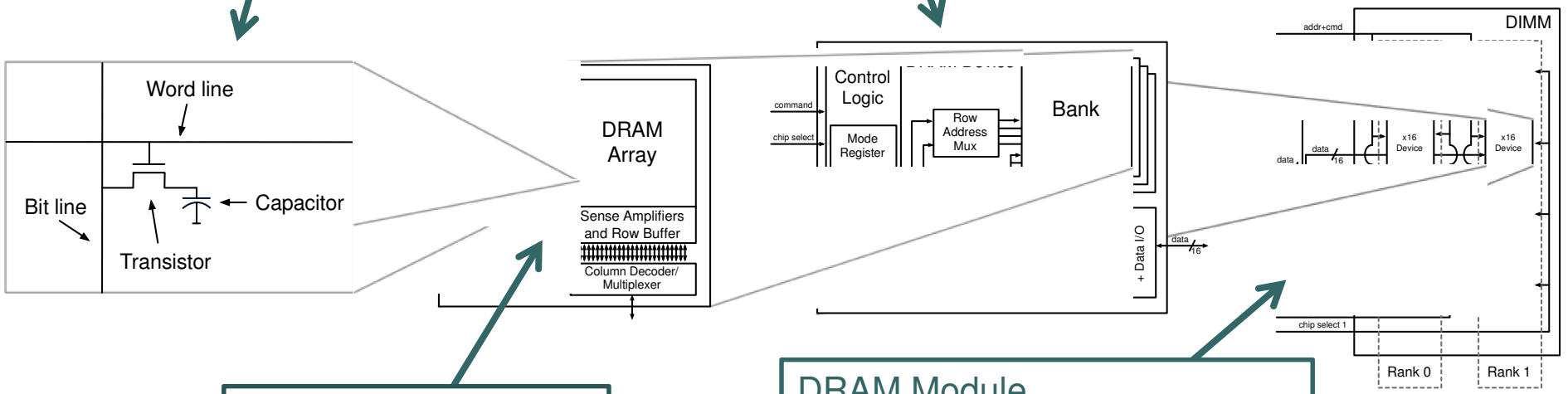


What about the main memory? Dynamic RAM Organization Overview

DRAM Cell
leaks charge ➔ needs to be refreshed (every 7.8µs for DDR2/DDR3)
therefore "dynamic"

DRAM Device
Set of DRAM banks +
- *Control logic*
- *I/O gating*

Accesses to banks can be pipelined, however I/O + control logic are shared



DRAM Bank
= Array of DRAM Cells
+ Sense Amplifiers and Row Buffer
Sharing of sense amplifiers and row buffer

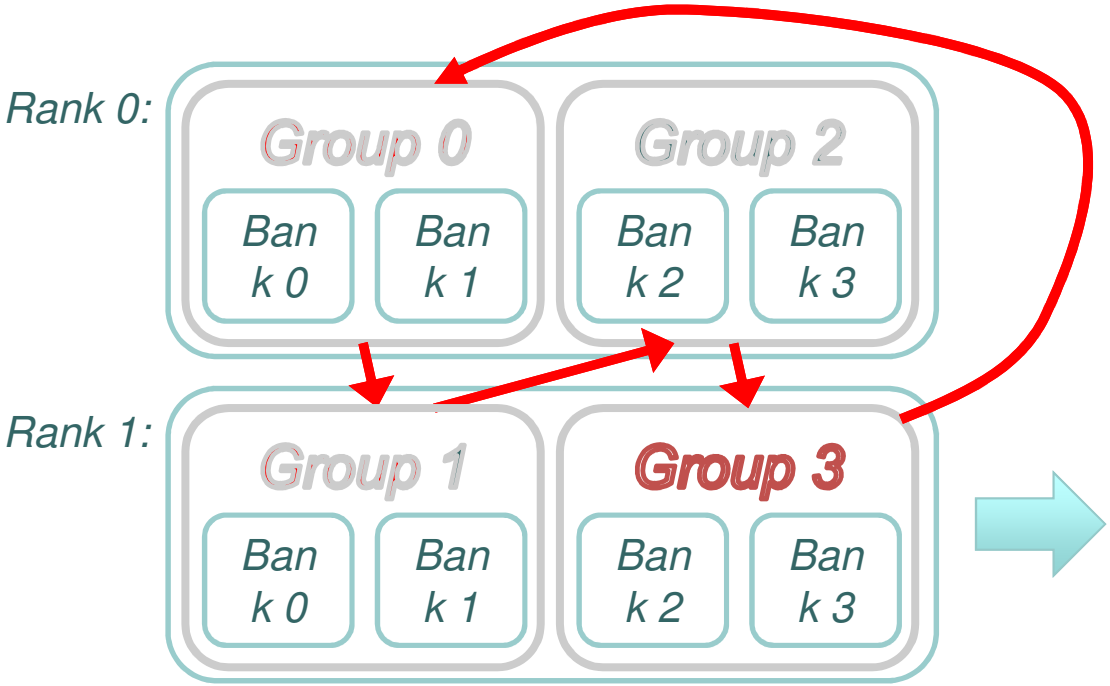
DRAM Module
Collection of DRAM Devices
- *rank = groups of devices that operate in unison*
- *Ranks share data/address/command bus*

DRAM Timing Constraints

- DRAM Memory Controllers have to conform to different timing constraints
- Almost all of these constraints are due to **competition for resources** at different levels:
  - Within the DRAM banks:
    rows are sharing sense amplifiers
  - Within a DRAM device:
    sharing of I/O gating and control logic
  - Between different ranks:
    sharing of data/address/command busses

PRET DRAM Controller: Exploiting 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

General-Purpose DRAM Controller vs PRET DRAM Controller

General-Purpose Controller

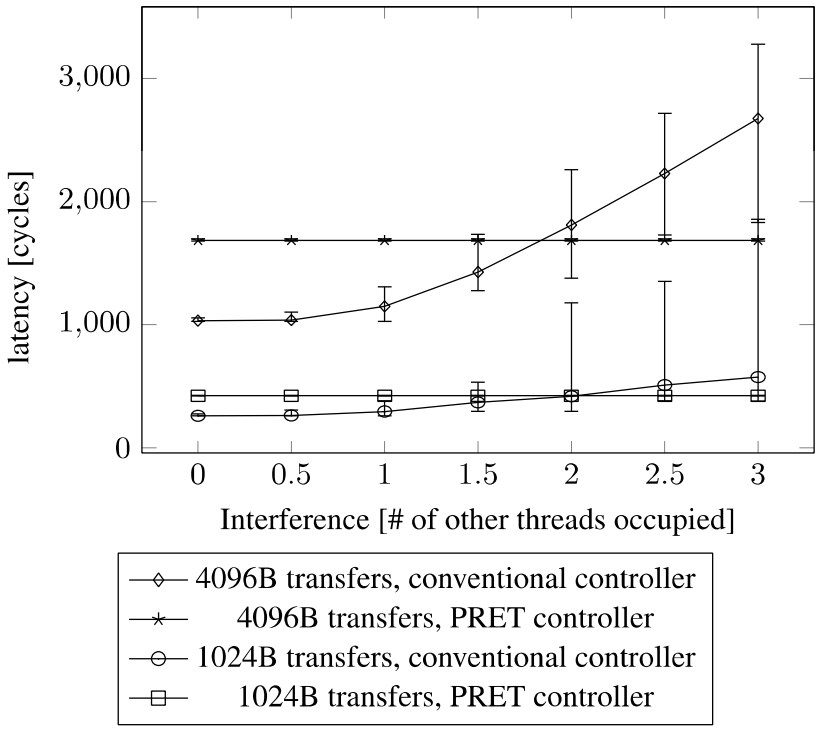
- Abstracts DRAM as a single shared resource
- Schedules refreshes dynamically
- Schedules commands dynamically
- “Open page” policy speculates on locality

PRET DRAM Controller

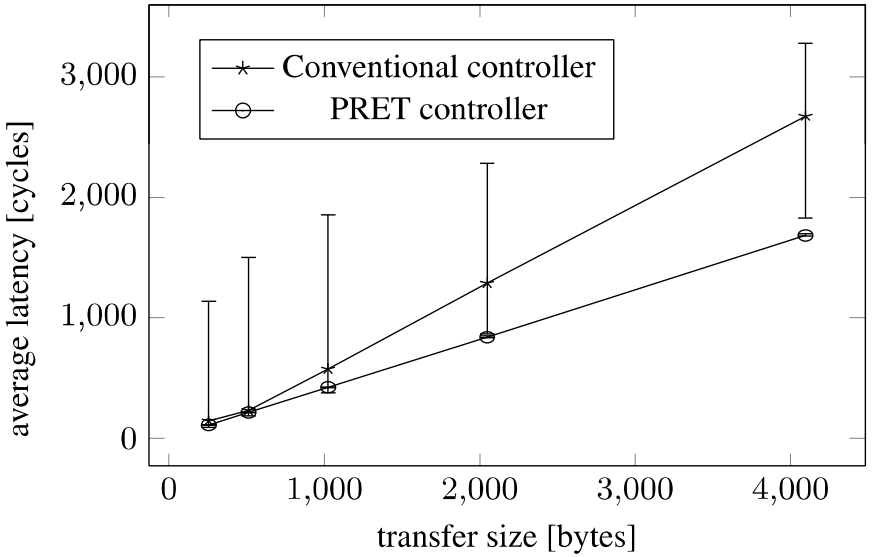
- Abstracts DRAM as multiple independent resources
- Refreshes as reads: shorter interruptions
- Defer refreshes: improves perceived latency
- Follows periodic, time-triggered schedule
- “Closed page” policy: access-history independence

Conventional DRAM Controller vs PRET DRAM Controller: Latency Evaluation

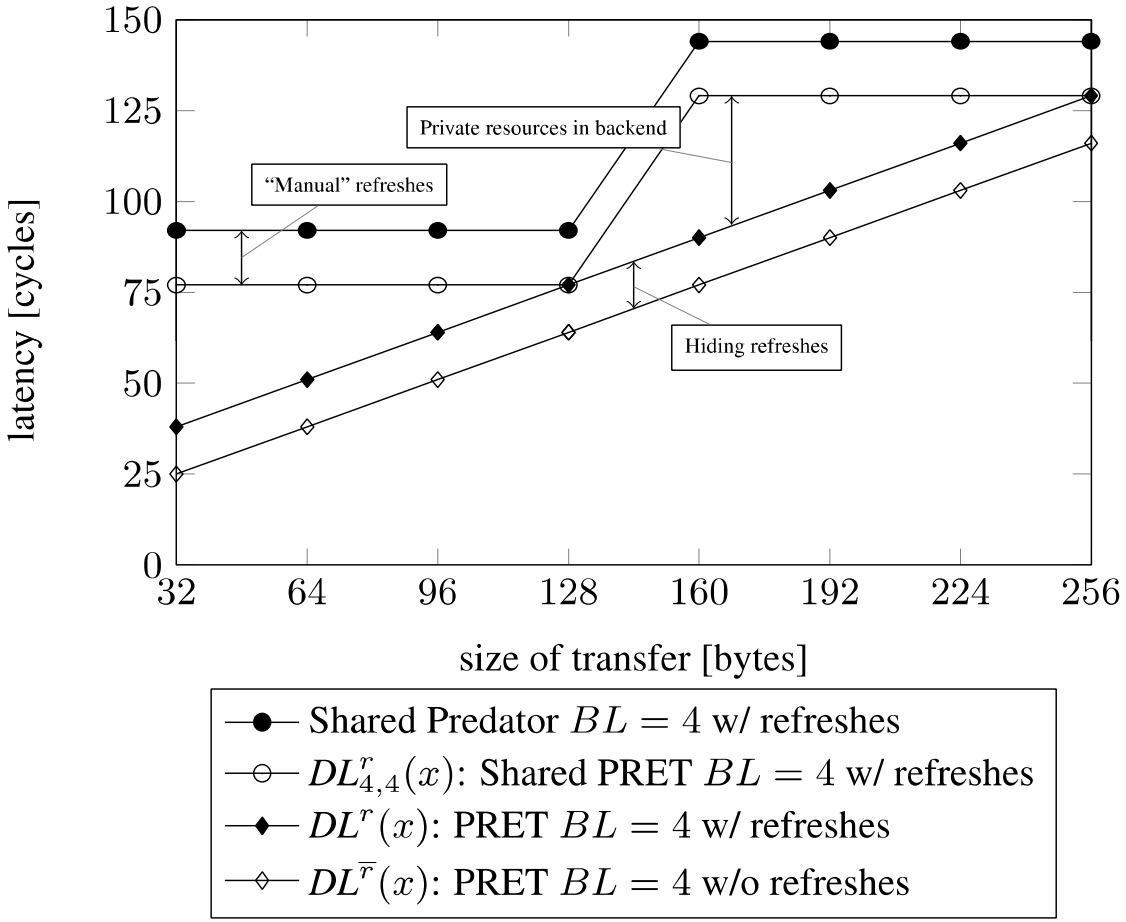
Varying Interference:



Varying Transfer Size:



PRET DRAM Controller vs Predator



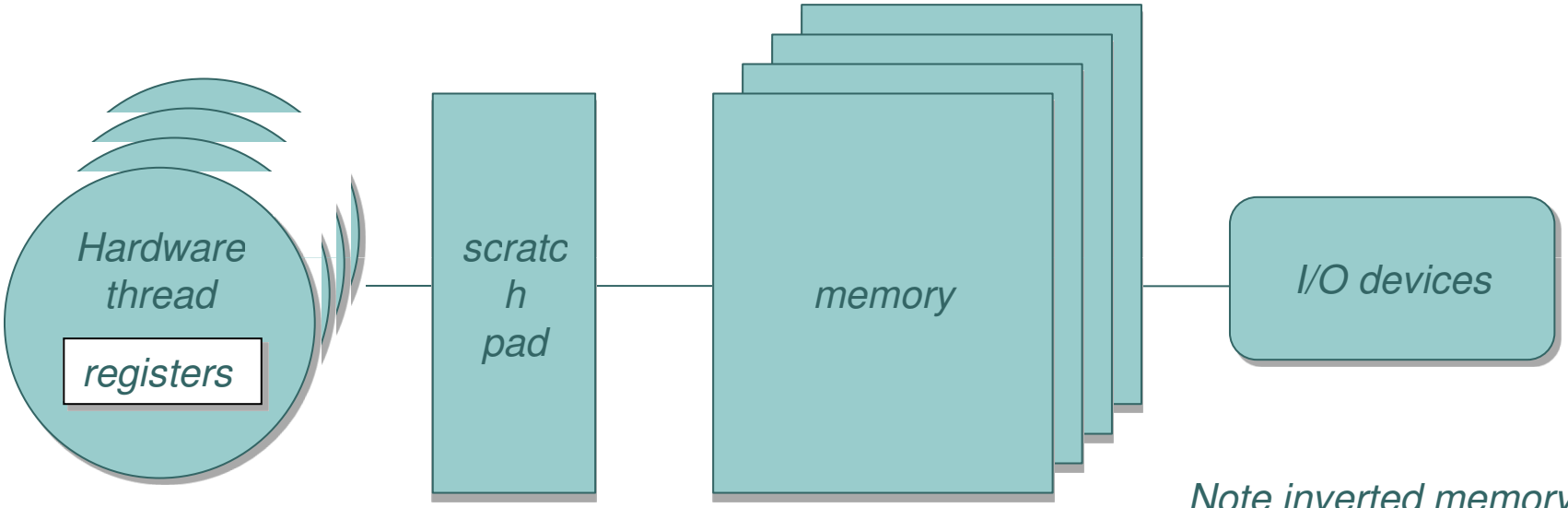
Predator:

- *abstracts DRAM as single resource*
- *uses standard refresh mechanism*

➔ *PRET's worst-case access latency of small transfers is smaller than Predator's*

➔ *PRET's drawback: memory is private*

PTARM Memory Hierarchy



Interleaved pipeline with one set of registers per thread

SRAM scratchpad shared among threads

DRAM main memory, separate banks per thread

Note inverted memory hierarchy!

Conclusions

- Real-time computing needs real-time abstractions
- Potential for significant improvements in worst-case performance of some hardware realizations
- For more information on PRET: *http://chess.eecs.berkeley.edu/pret/*

Raffaello Sanzio da Urbino – The Athens School

References

- [CODES '11] Jan Reineke, Isaac Liu, Hiren D. Patel, Sungjun Kim, Edward A. Lee, PRET DRAM Controller: Bank Privatization for Predictability and Temporal Isolation, *International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, October, 2011.
- [DAC '11] Dai Nguyen Bui, Edward A. Lee, Isaac Liu, Hiren D. Patel, Jan Reineke, Temporal Isolation on Multiprocessing Architectures, *Design Automation Conference (DAC)*, June, 2011.
- [Asilomar '10] Isaac Liu, Jan Reineke, and Edward A. Lee, PRET Architecture Supporting Concurrent Programs with Composable Timing Properties, in *Signals, Systems, and Computers (ASILOMAR)*, Conference Record of the Forty Fourth Asilomar Conference, November 2010, Pacific Grove, California.
- [ICCD '09] Stephen A. Edwards, Sungjun Kim, Edward A. Lee, Isaac Liu, Hiren D. Patel, Martin Schoeberl. A Disruptive Computer Design Idea: Architectures with Repeatable Timing, Proceedings of International Conference on Computer Design (ICCD), IEEE, Lake Tahoe, CA, 4-7 October, 2009.
- [CASES '08] Ben Lickly, Isaac Liu, Sungjun Kim, Hiren D. Patel, Stephen A. Edwards and Edward A. Lee, "Predictable Programming on a Precision Timed Architecture," in Proceedings of International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES), Piscataway, NJ, pp. 137-146, IEEE Press, October, 2008.
- [UCB-TR '08] Hiren D. Patel, Ben Lickly, Bas Burgers and Edward A. Lee, "A Timing Requirements-Aware Scratchpad Memory Allocation Scheme for a Precision Timed Architecture," EECS Department, University of California, Berkeley, Technical Report No. UCB/EECS-2008-115, September 12, 2008.