PREcision Timed (PRET) Architecture

Isaac Liu
Advisor – Edward A. Lee

Dissertation Talk
April 24, 2012
Berkeley, CA

Acknowledgements

- Many people were involved in this project:
  - Edward A. Lee – UC Berkeley
  - David Broman – UC Berkeley
  - Ben Lickly – UC Berkeley
  - Hiren Patel – University of Waterloo
  - Jan Reineke – Saarland University
  - Stephen Edwards – Columbia University
  - Sungjun Kim – Columbia University
  - Matt Viele – Drivven Inc.
  - Gerald Wang – National Instruments
  - Hugo Andrade – National Instruments
  - And many more…

Cyber Physical Systems

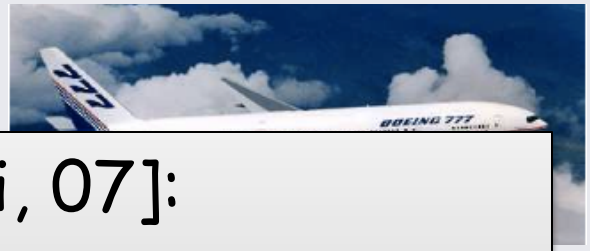
Two key characteristics of physical processes

- Inherently Concurrency
- Uncontrollable passage of time

Instrumentation

E-Corner, Siemens

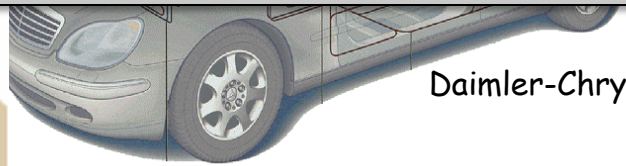
Avionics:



Key Challenges [Sangiovanni-Vincentelli, 07]:

- Composability ← Concurrency
- Timing Predictability ← Passage of Time
- Dependability

Courtesy of Doug Schmidt

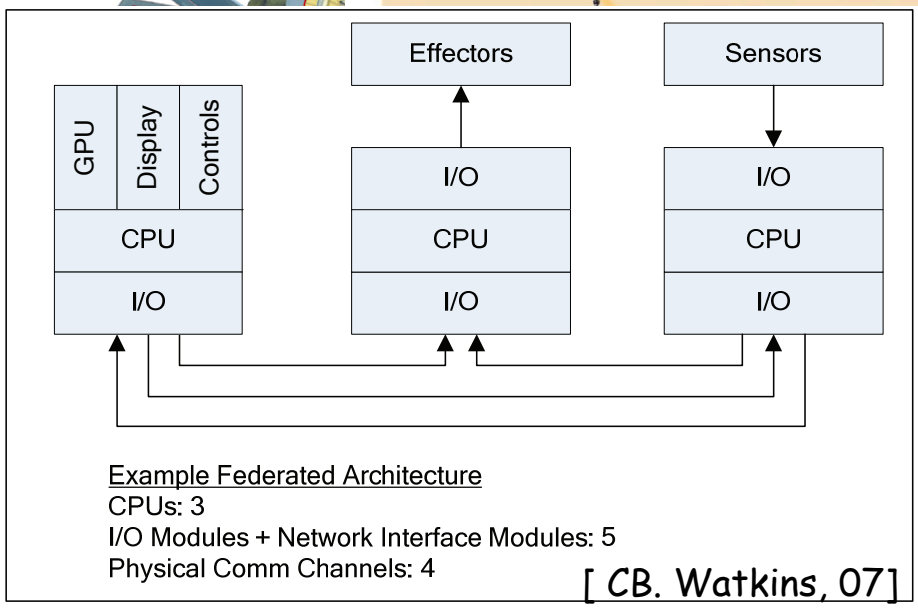


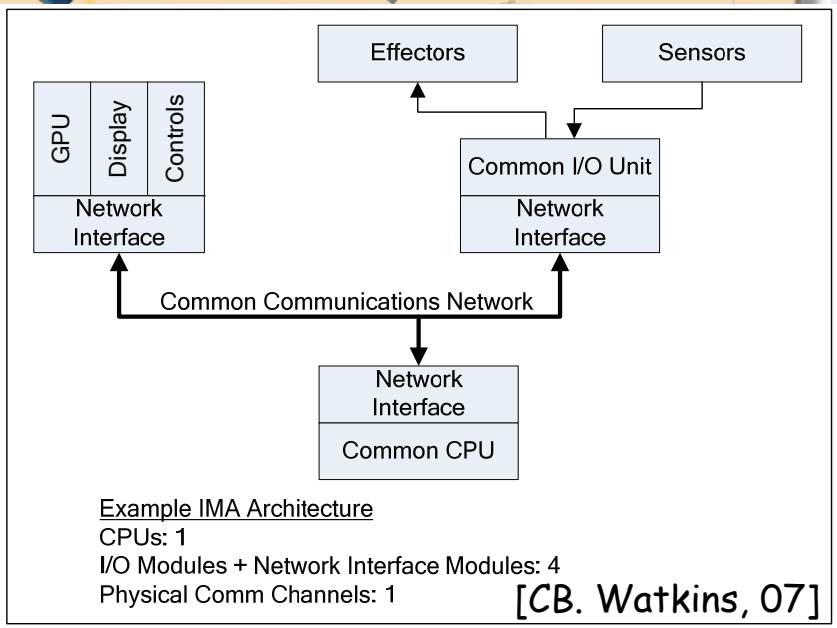
Daimler-Chrysler

Composability

AUTOSAR AUTOMOTIVE OPEN SYSTEM ARCHITECTURE

IMA - Integrated Modular Avionics





[Sangiovanni-Vincentelli, ee249 lecture 1]

Timing Predictability

How long does it take to execute the following code?

Let's assume we know n

```
for (i = 1; i < n; i++)
    if ( a[i] > b[i] )
        c[i] = c[i-1] + a[i];
    else
        c[i] = c[i-1] + b[i];
```

10

Branch predicted correctly?

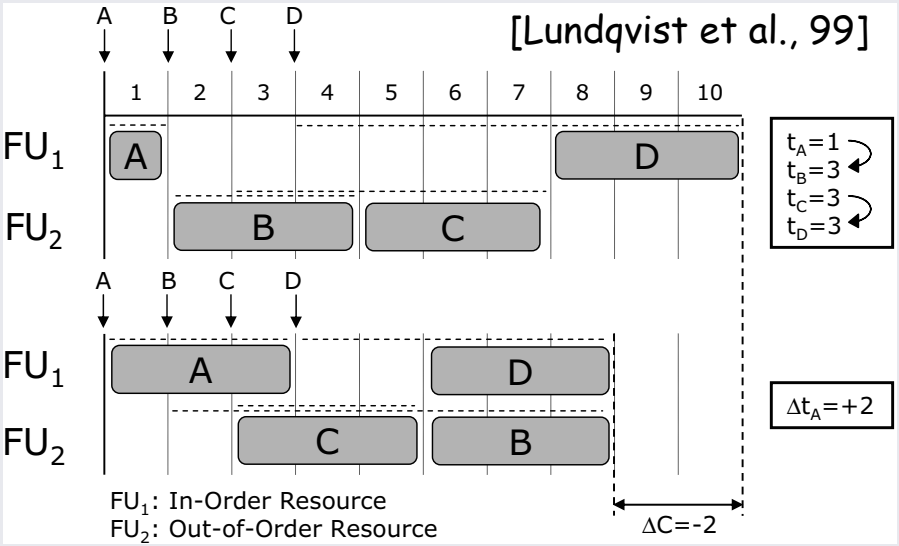
Cache Hit? Miss?

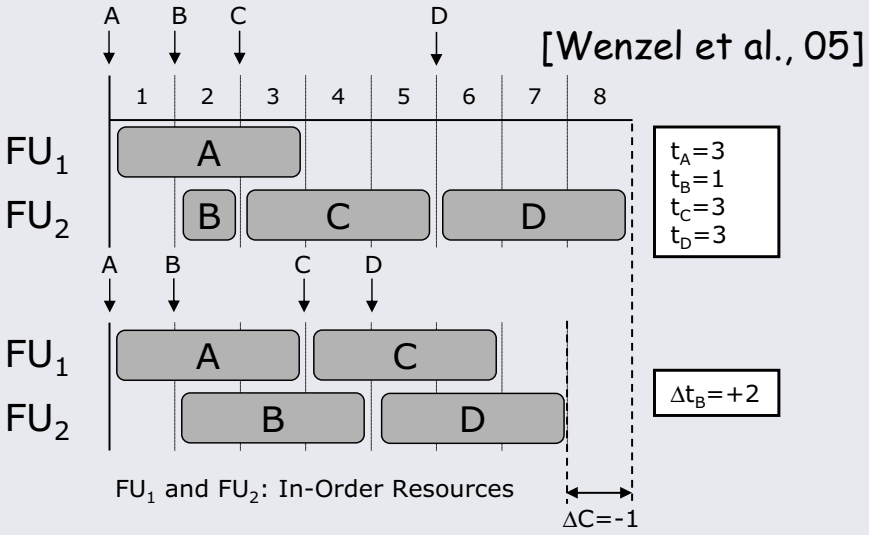
Data Dependency

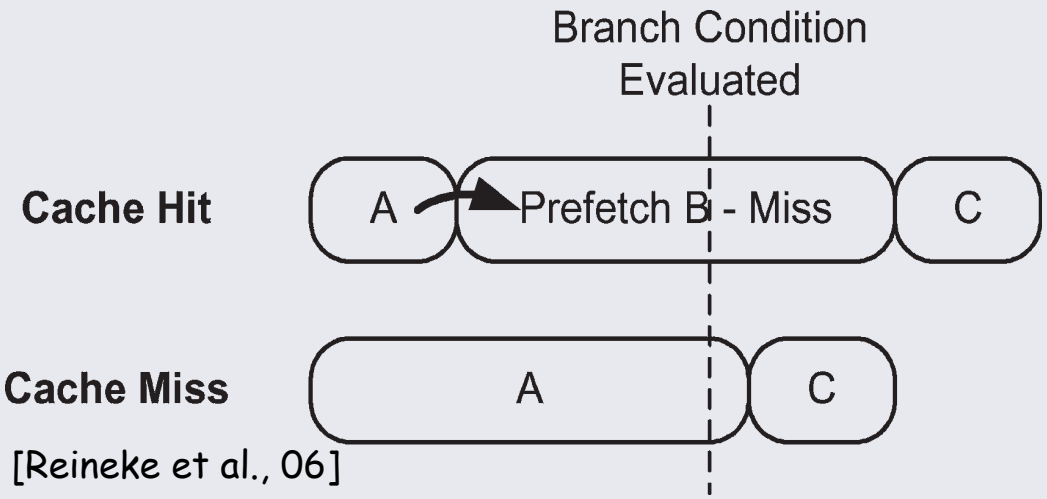
Out of order execution? Multithreading?

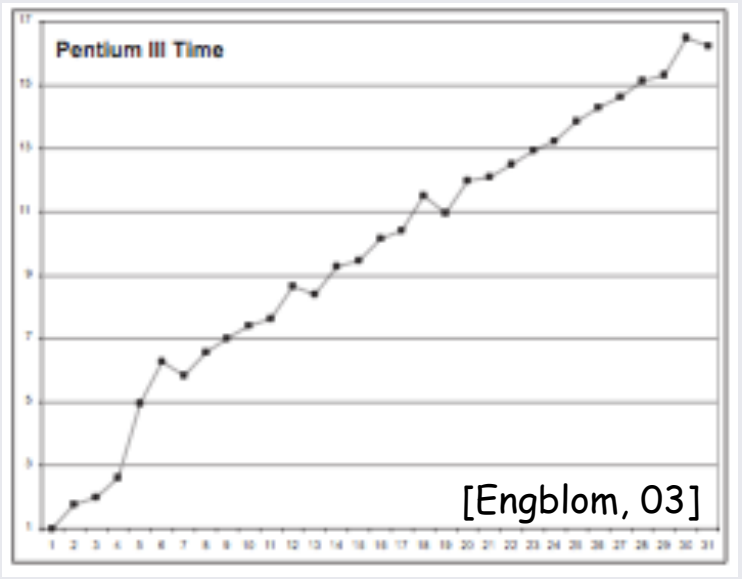
Assume branch mispredict, cache miss?

Timing Anomalies









Challenges in WCET Analysis

- "However, both the precision of the results and the efficiency of the analysis methods are **highly dependent** on the predictability of the execution platform. In fact, **the architecture** determines whether a static timing analysis is practically feasible at all and whether the most precise obtainable results are precise enough."

(Emphasis added) **[Wilhelm, 03]**

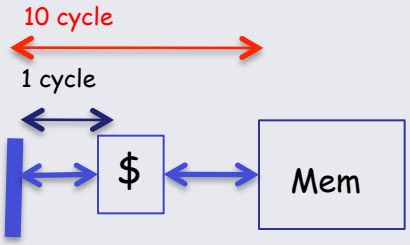
Heckmann et al., *The influence of processor architecture on the design and the results of wcet tools*, IEEE 03

Contribution

- Propose an architecture that allows for timing predictability and composable resource sharing without sacrificing performance.

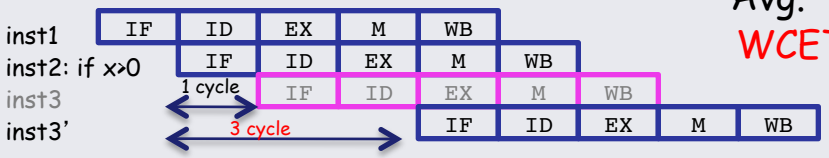
Architecture Improvements

Cache



Avg. Time
WCET

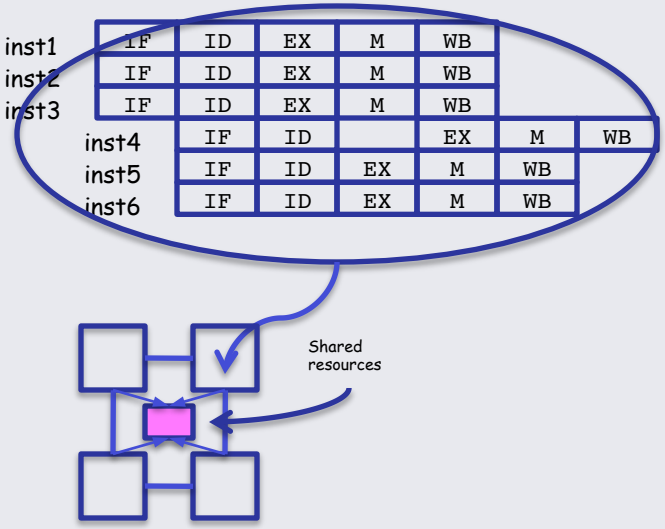
Pipelines



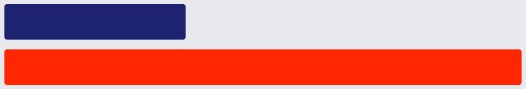
Avg. Time
WCET

Superscalar Out of Order

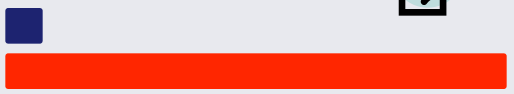
Multicore



Avg. Time
WCET

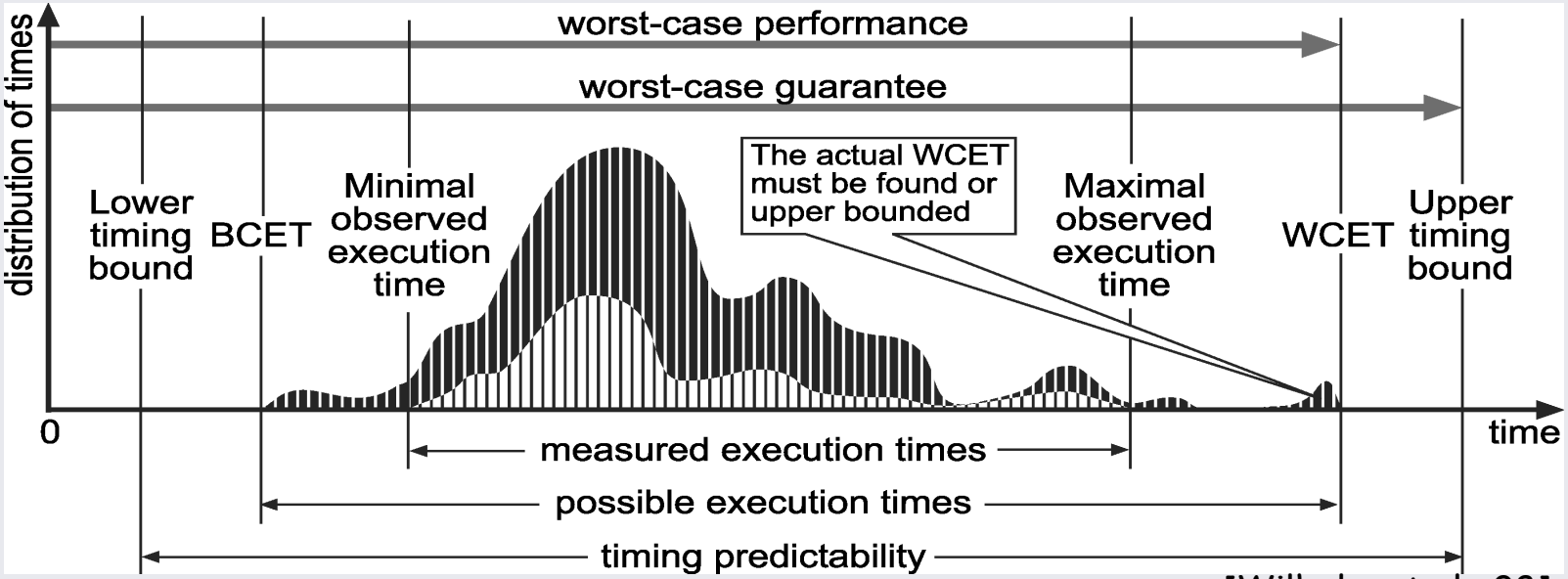


Avg. Time
WCET



[Courtesy of Sami Yehia, Thales]

Execution Time Variance



[Wilhelm et al., 08]

"Future applications, including safety-critical and active-safety ones, need shorter latencies and time determinism - **reduced jitter** - to increase performance."

[Sangiovanni-Vincentelli, 07]

Related Work

- Modifying Modern Processors
  - Superscalar [Rochange et al., 05], [Whitham et al., 08]
  - VLIW [Yan et al., 08]
  - Multithreading [Kreuzinger et al., 00], [El-Haj-Mahmoud et al., 05]
  - SMT [Barre et al., 08], [Mische et al., 08], [Metzlaff et al., 08]
- WCET Analysis
  - Pipeline Analysis [Schneider et al., 99], [Ferdinand et al., 01], [Lagenbach et al., 02], [Kirner et al. 09] …
  - Cache Analysis [Heckmann et al., 03], [Reineke et al., 07] …
- Stack Based Architecture
  - Java Optimized Processor [Schoeberl, 06]

Precision Timed Architecture

Summary of architectural features:

Traditional	PRET
Deep out-of-order pipelines (Instructional level parallelism)	Thread-interleaved pipelines (Thread level parallelism)
Caches (Hardware replacement policy)	Scratchpads (Software controlled replacement)
Best effort DRAM Controller	Predictable DRAM Controller

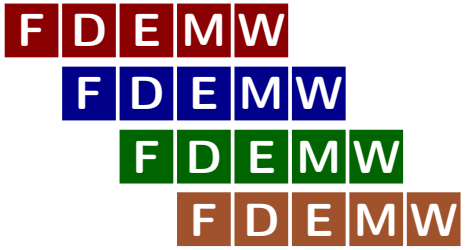
See S. Edwards and E. A. Lee, "**The Case for the Precision Timed (PRET) Machine**," in the *Wild and Crazy Ideas* Track of the *Design Automation Conference* (DAC), June 2007.

Pipelining

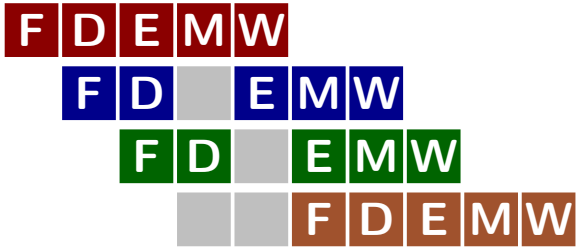
```
LD R1, 45(r2)
DADD R5, R1, R7
BE R5, R3, R0
ST R5, 48(R2)
```

Unpipelined

The Dream



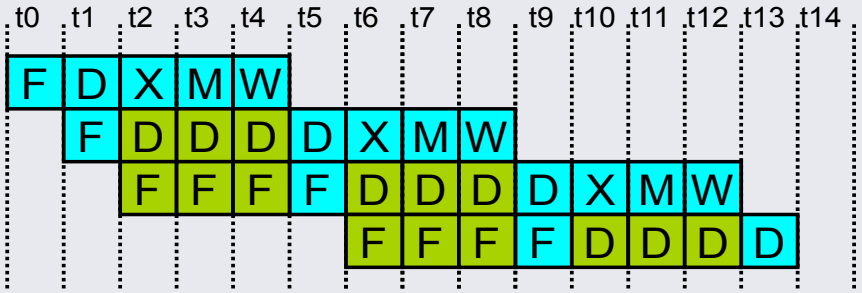
The Reality

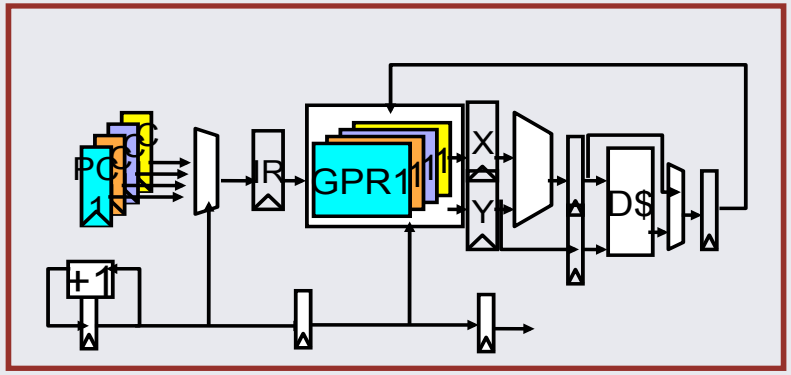


Memory Hazard
Data Hazard
Branch Hazard

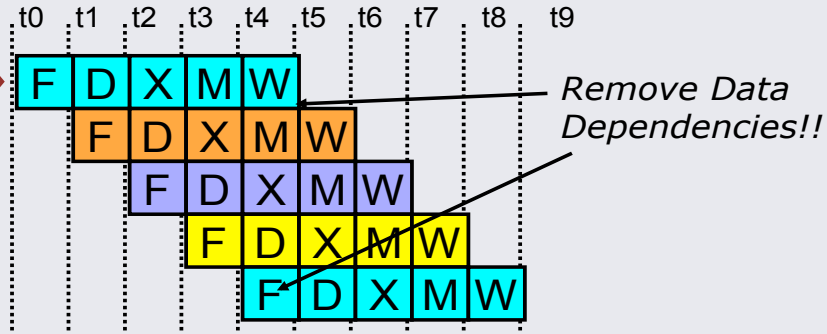
Edwards, RePP 09

Interleaved Pipeline





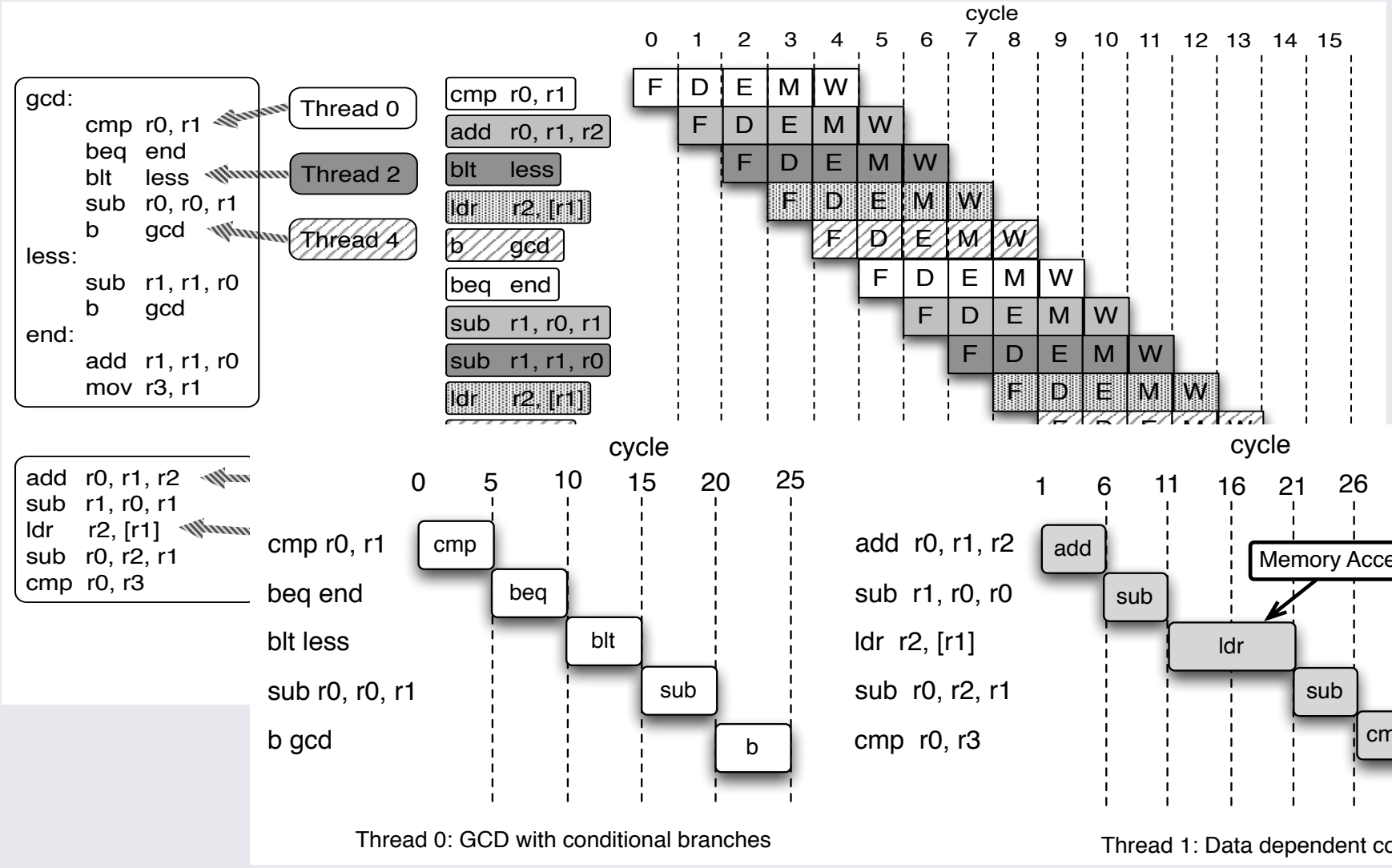
- Denelcor, HEP (1981), Lee and Messerschmitt, DSP (1987), CDC 6000 (1961)…



[Asonavic, CS252 lecture F07]

Also called Fine Grained Multithreading!

Thread Interleaved Execution



Interleaved Pipeline

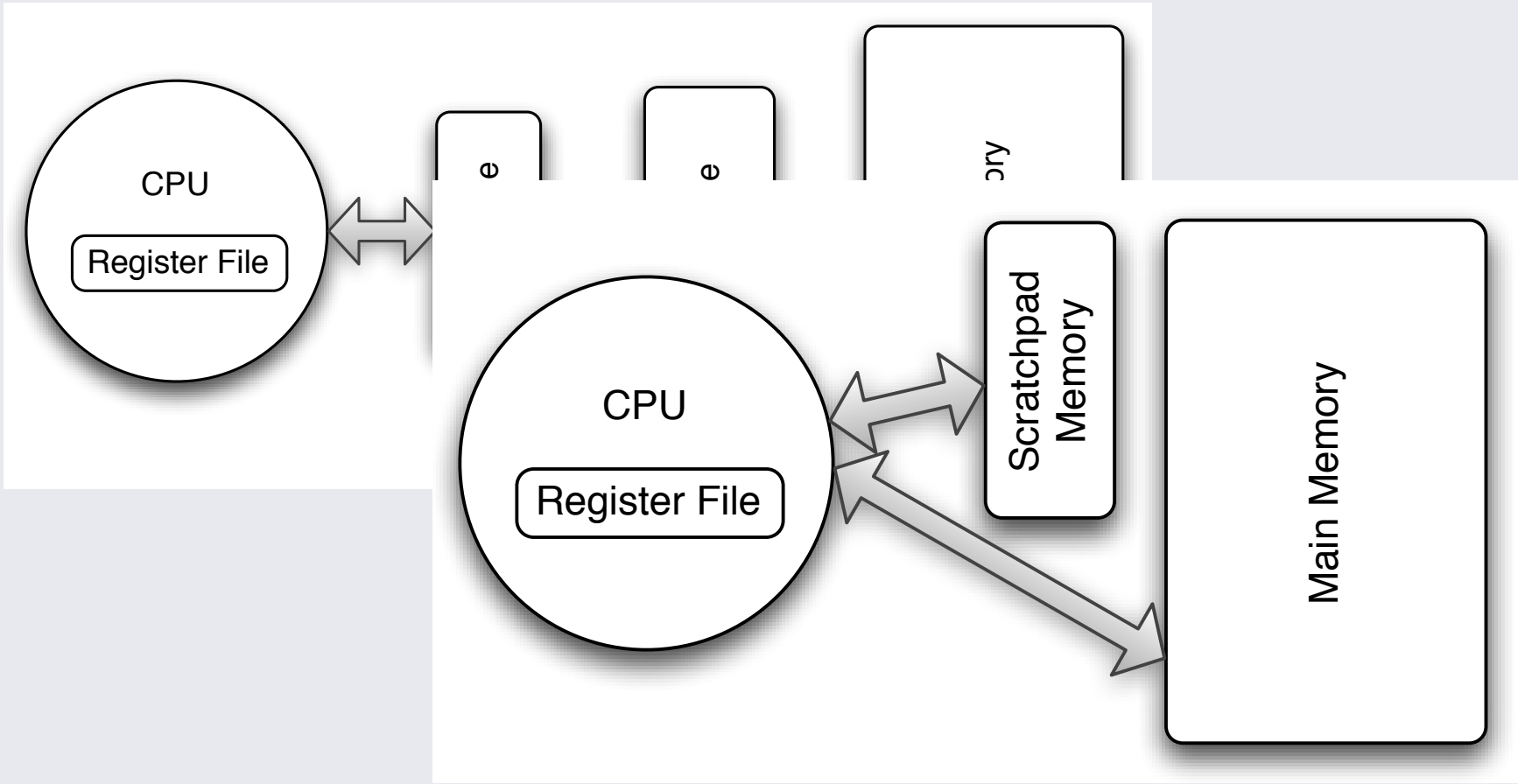
Trade-offs:

- Need enough concurrency to utilize processor
- Favor throughput over latency

However…

- Simpler WCET analysis (Timing Predictability)
- Interference free multiple context execution (Composability)
- Simple pipeline design (Energy, Cost…)
- Improved throughput and clock rate (Performance)

Memory Hierarchy



Use Scratchpads instead of Caches!

Scratchpads

Trade-offs:

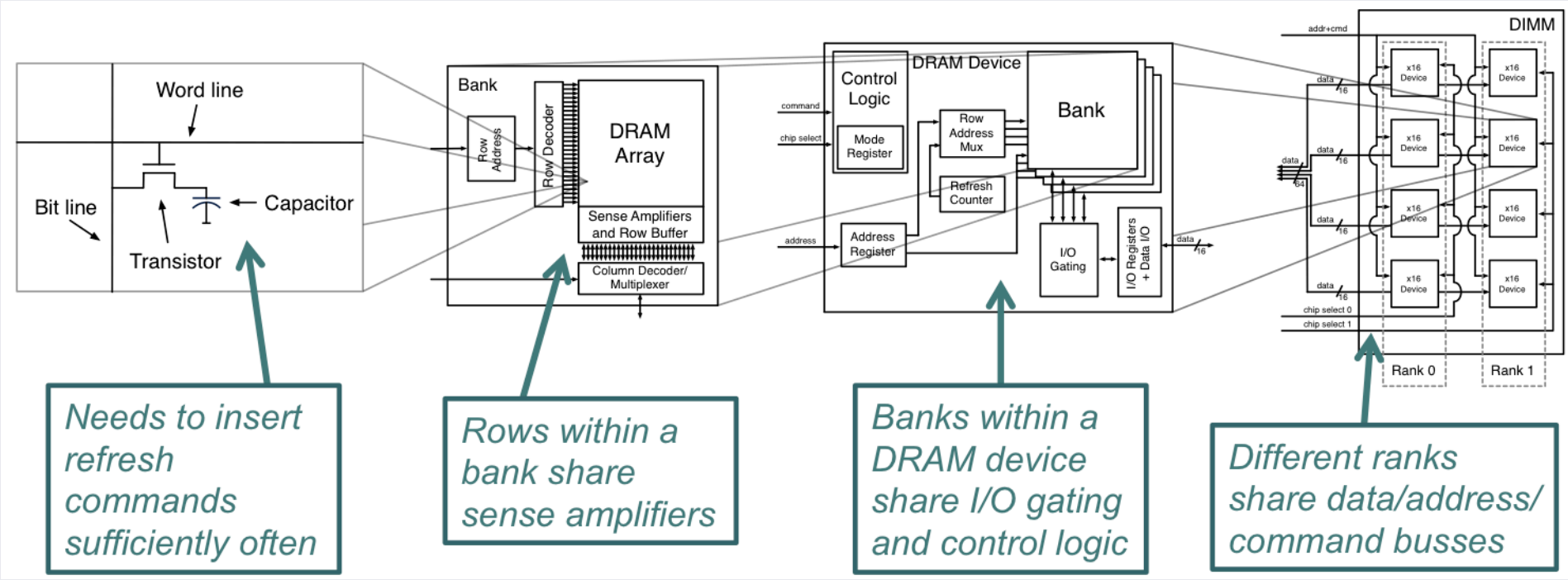
- Need explicit management from the software (compiler/programmer)

However…

- Simpler WCET analysis (Timing Predictability)
- Customize to workload (Performance)
- Simple circuit design (Energy, Cost…)

Main Memory

DRAMs:

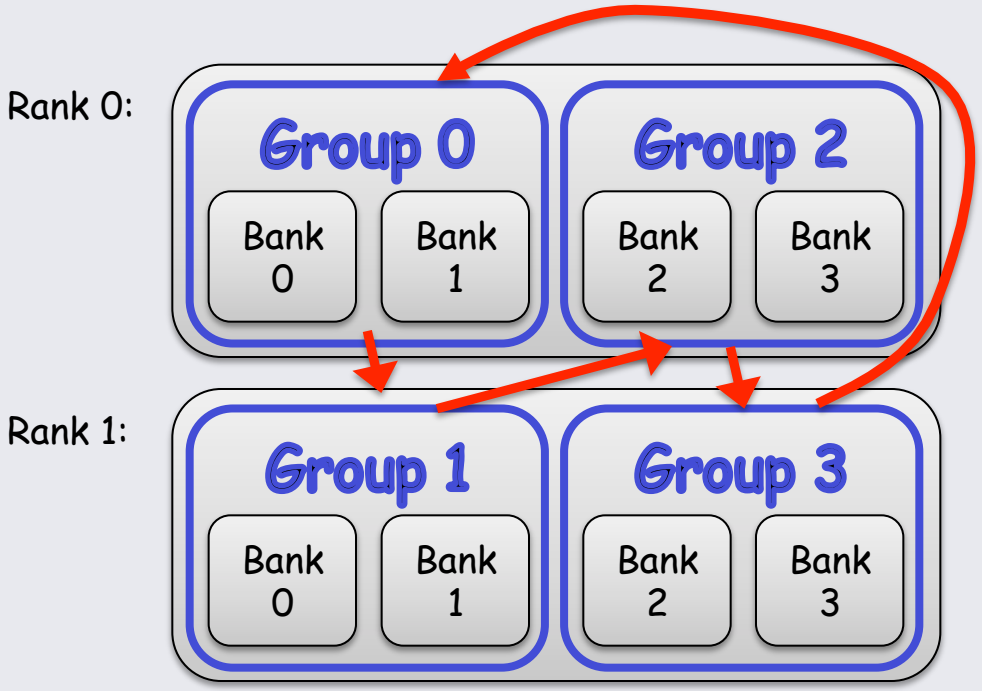


Two key problems:

- Bank Conflicts
- DRAM Refresh

→ Variable Access Times

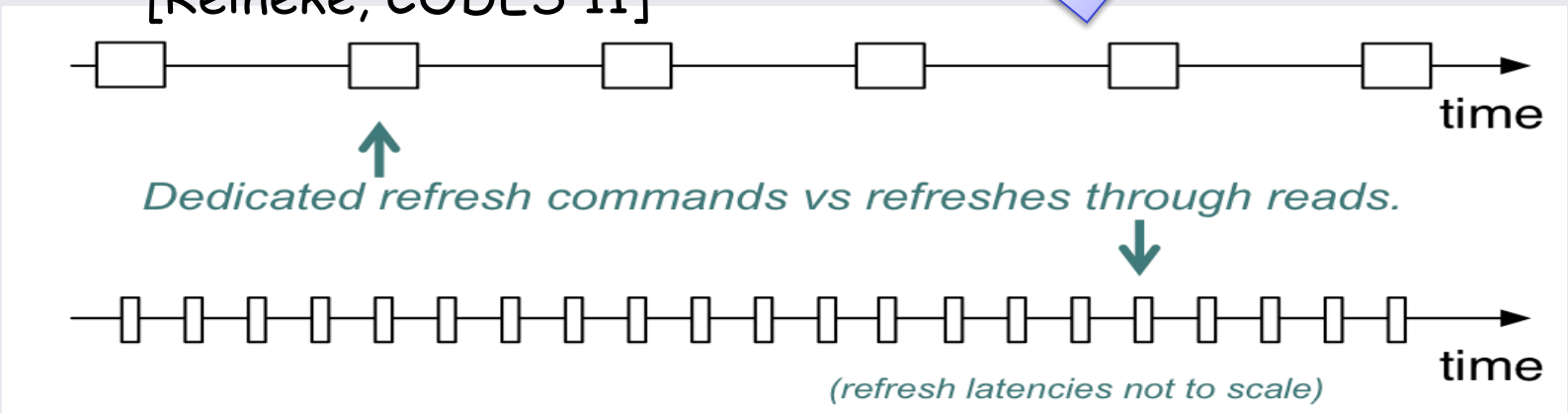
PRET DRAM Controller



Provides four independent and predictable resources

Allows for predictable refreshes

[Reineke, CODES 11]



Main Memory

Trade-Offs:

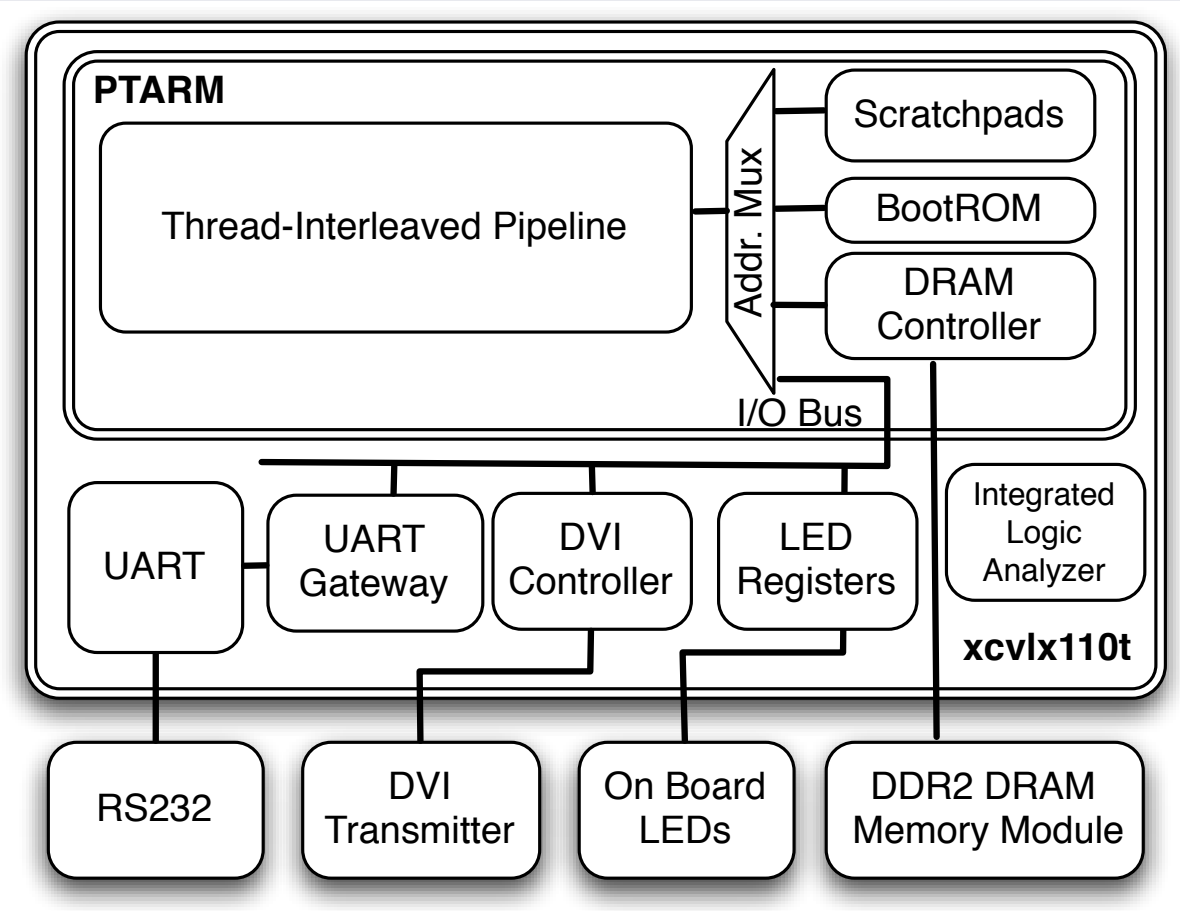
- Shared memory on scratchpad
- Longer average memory latencies

However…

- Predictable access latencies (Timing Predictability)
- Better throughput and latency when fully utilized (Performance)

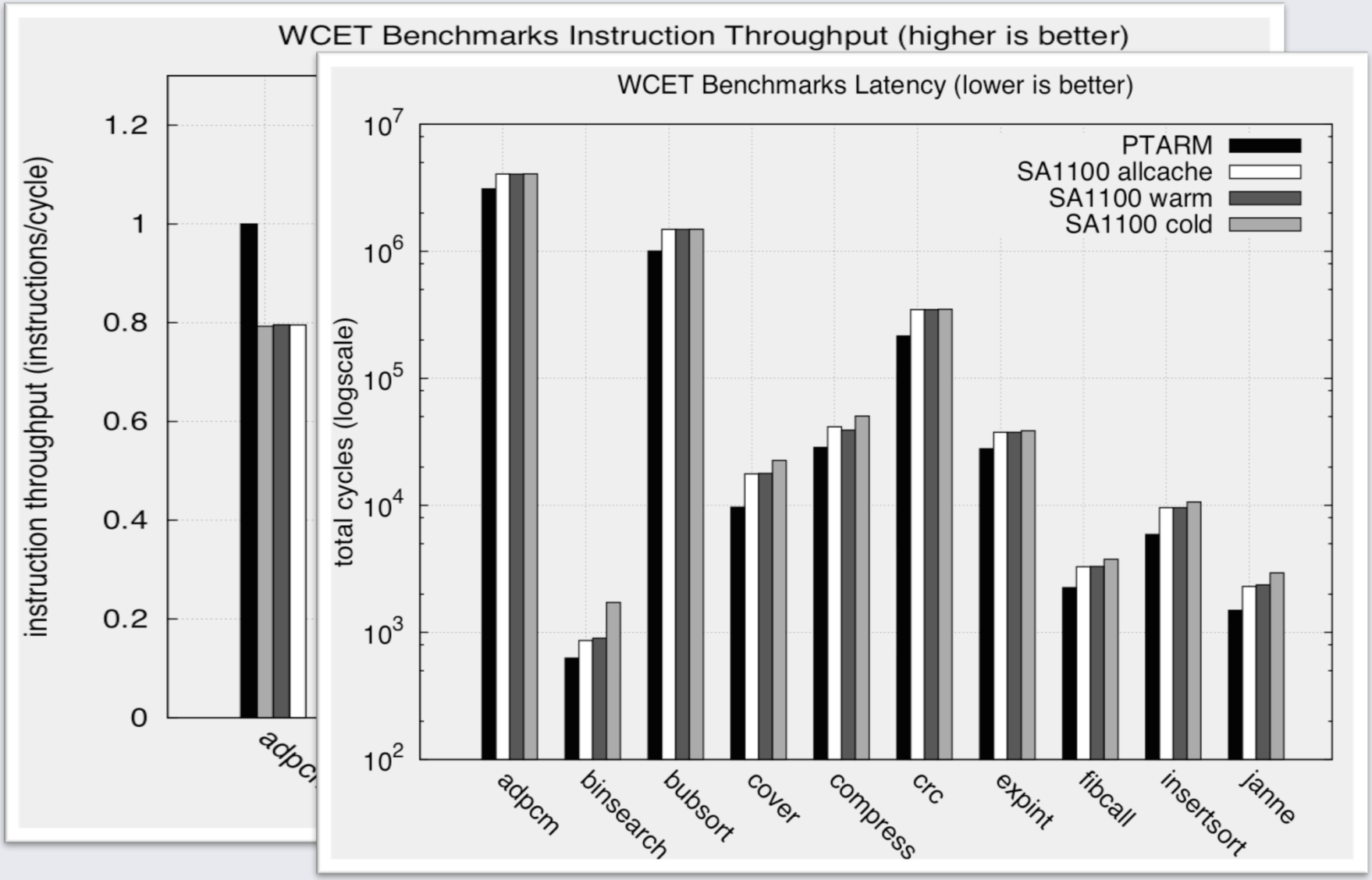
Reineke et al., PRET DRAM Controller: Bank Privatization for Predictability and Temporal Isolation, CODES 11

PTARM



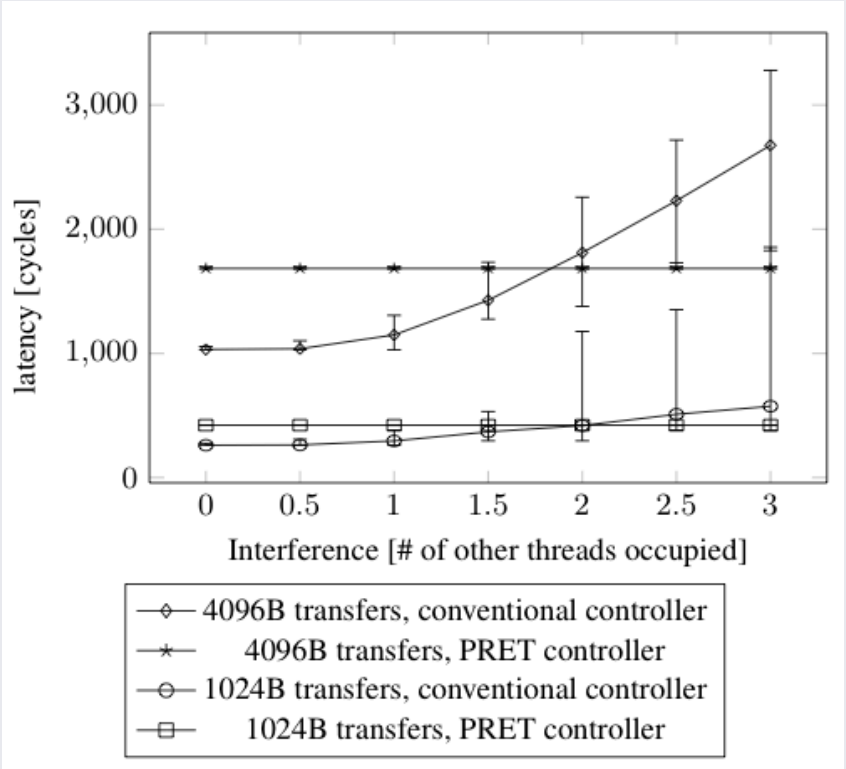
Download at http://chess.eecs.berkeley.edu/pret

Pipeline Performance

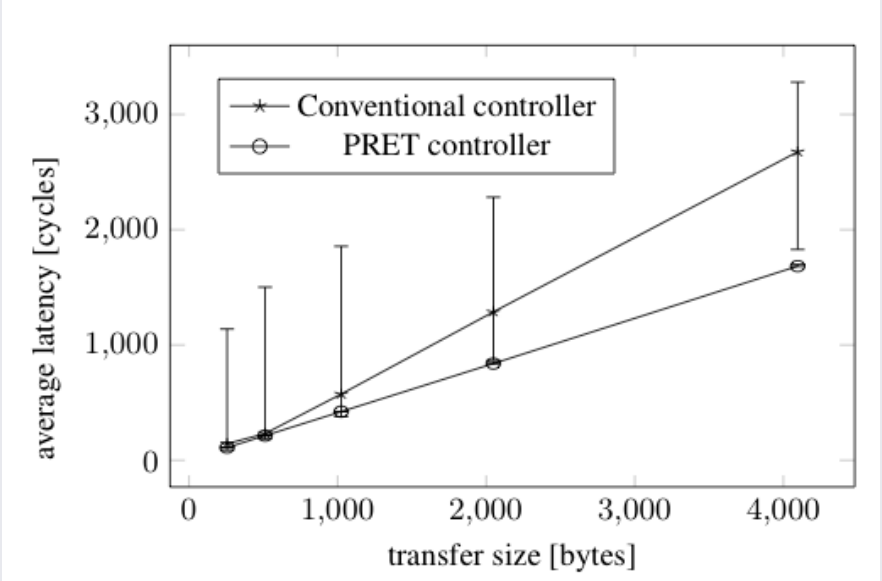


DRAM Performance

Varying Interference



Varying Bandwidth



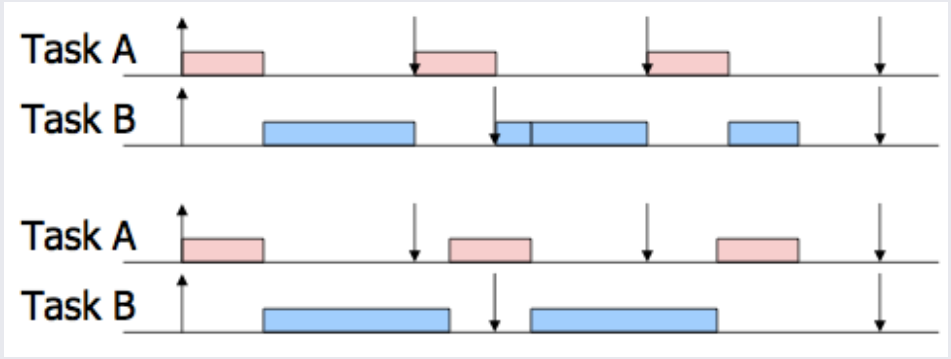
Reineke et al., PRET DRAM Controller: Bank Privatization for Predictability and Temporal Isolation, CODES 11

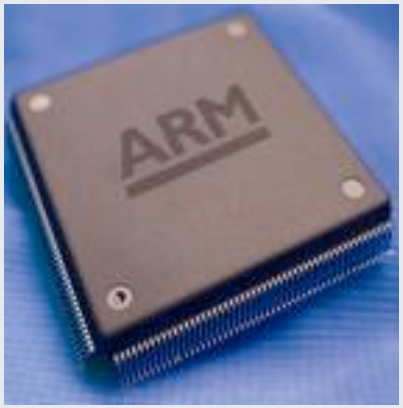
Contribution

- Propose an architecture that allows for timing predictability and composable resource sharing without sacrificing performance.
  - Use architectural techniques that provides composability and timing predictability

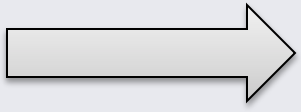
- **Expose "time" in the Instruction Set Architecture.**

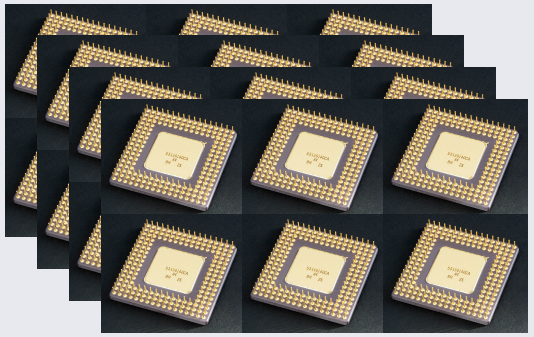
Current Methods





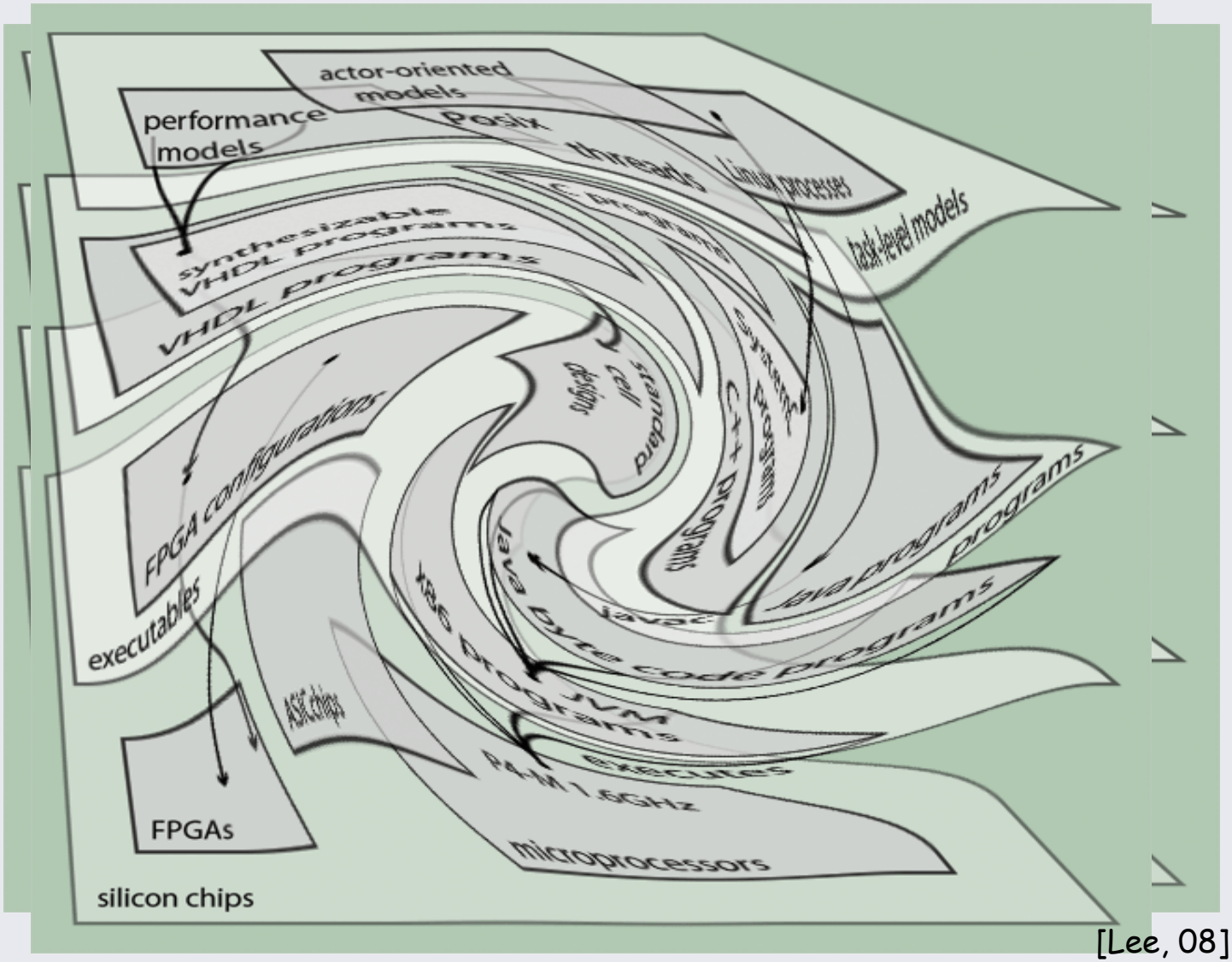
Fly-by-wire aircraft controlled by <u>software.</u>





They have to purchase and store microprocessors for at least 50 years production and maintenance…

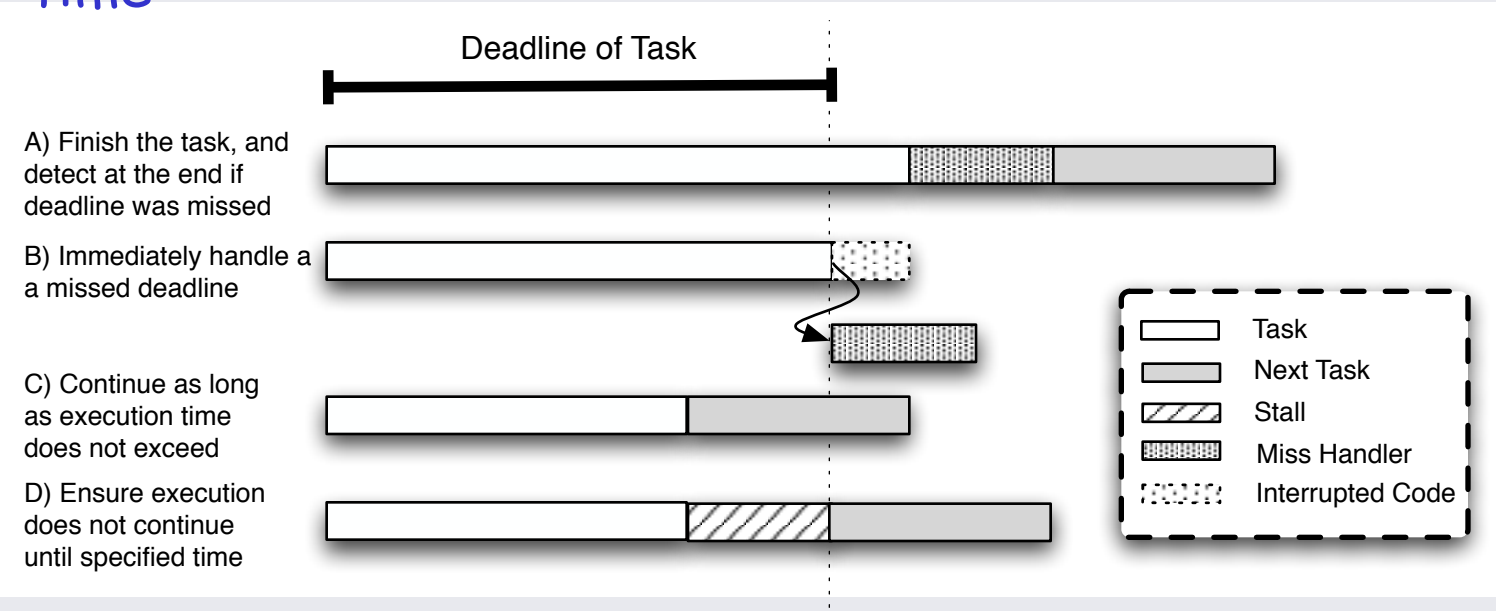
Levels of Abstraction



[Lee, 08]

ISA with "time"

- Extend Instruction Set with timing instructions that specify and control timing behaviors of *code blocks*.
  - Assume a "platform clock" synchronous with the execution of instructions
  - Timing instructions use platform clock to control execution time

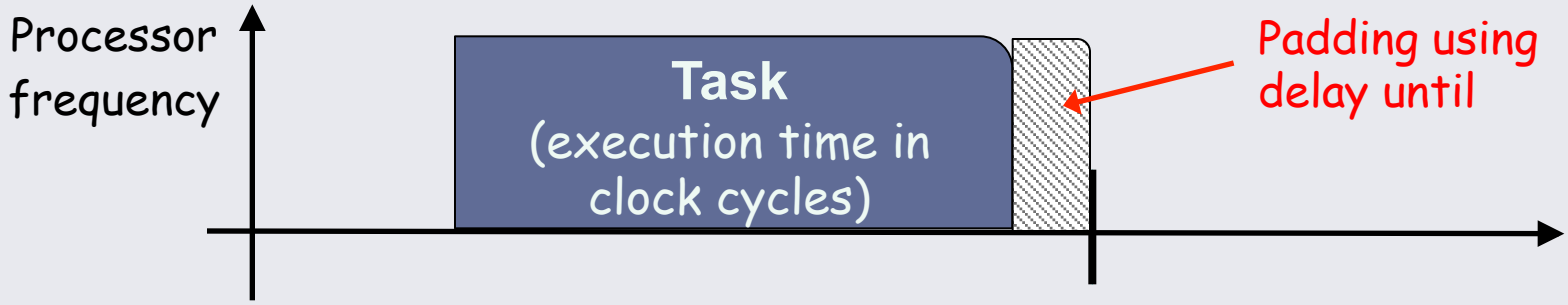


Timing Control

New instruction *get time* (gt)

```
gt          r1, r2          ; get time (ns)
  -- Code block --
adds        r2, r2, #500    ; add 500 ns
adc         r1, r1, #0      ; add with carry (time in 2 32-bit reg)
du          r1, r2          ; delay until 500ns have elapsed
```

New instruction *delay until* (du)



Where could this be useful?

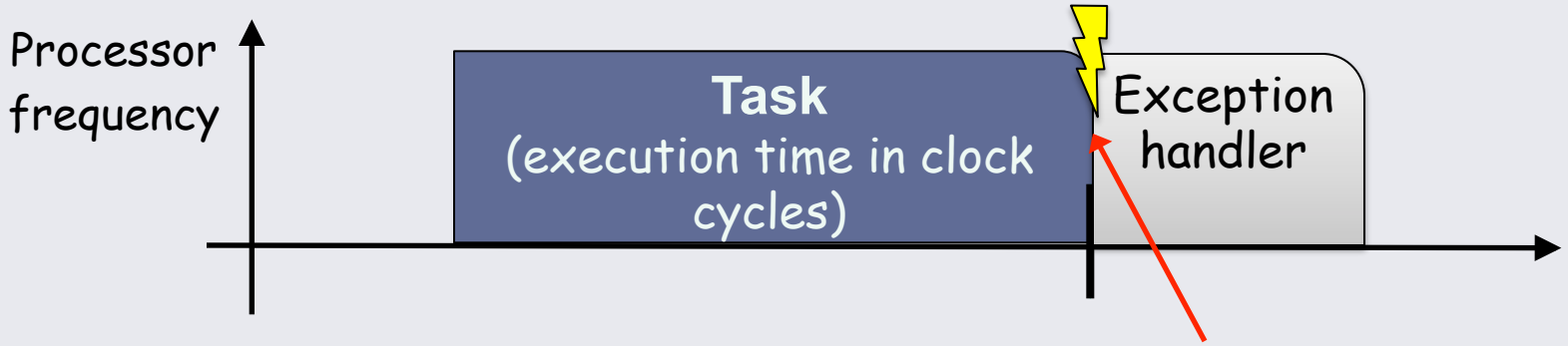
- Finishing early is not always better:
  - Scheduling Anomalies (Graham's anomalies)
  - Communication protocols and External synchronization

Timing Exceptions

New instruction *exception on expire* (ee)

```
gt          r1, r2          ; get time (ns)
adds        r2, r2, #500    ; add 500 ns
adc         r1, r1, #0      ; add with carry (time in 2 32-bit reg)
ee          r1, r2          ; register timer exception
  -- Code block --
de                          ; deactivate exception
```

New instruction *deactivate exception* (de)

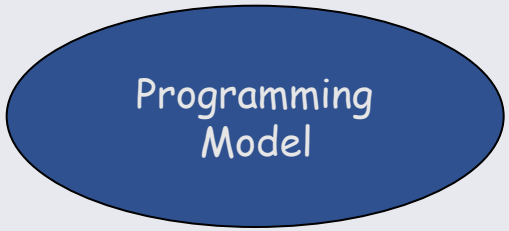


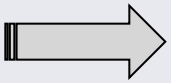
Where could this be useful?

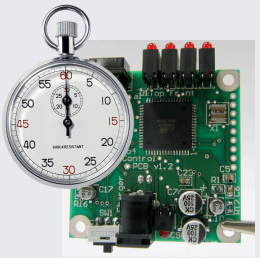
- Immediate deadline miss detection

ISA with "time"

Traditional Approach

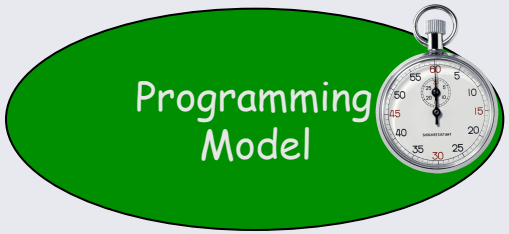




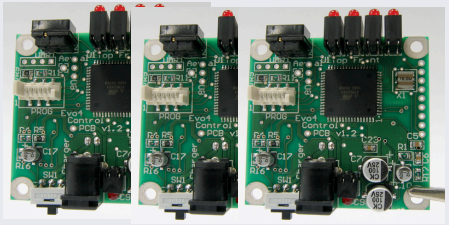


Timing Dependent on the Hardware Platform

Our Objective



Make time an engineering abstraction within the programming model



Timing is independent of the hardware platform (within certain constraints)

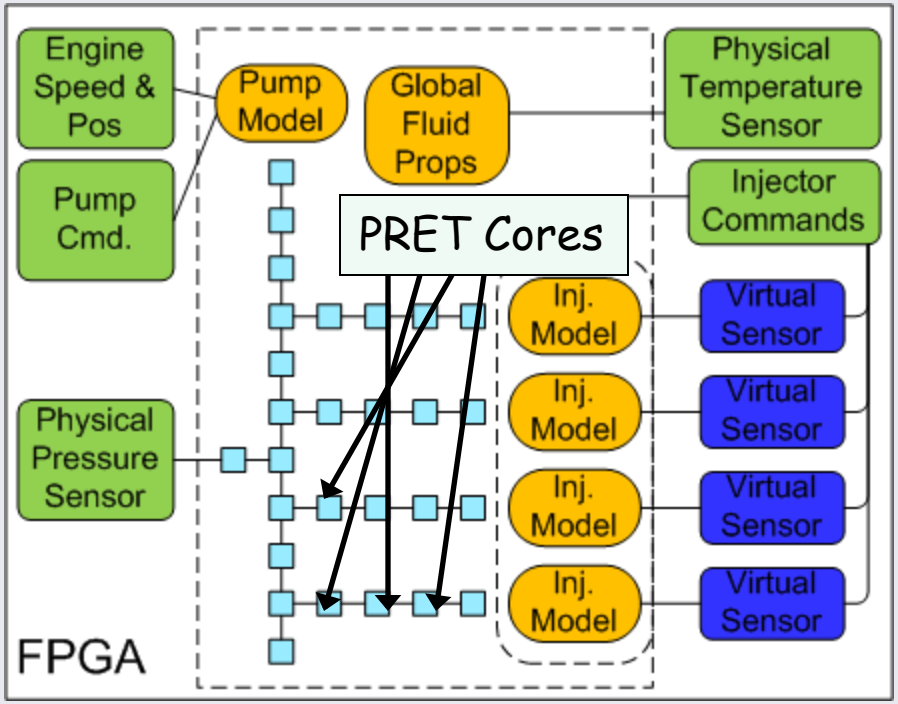
A Timing Requirements-Aware Scratchpad Memory Allocation Scheme for a Precision Timed Architecture [Patel et al. 08]

Contribution

- Propose an architecture that allows for timing predictability and composable resource sharing without sacrificing performance.
  - Use architectural techniques that provides composability and timing predictability

- **Expose "time" in the Instruction Set Architecture.**
  - ISA extensions to specify temporal properties

Real-Time Engine Fuel Rail Simulation

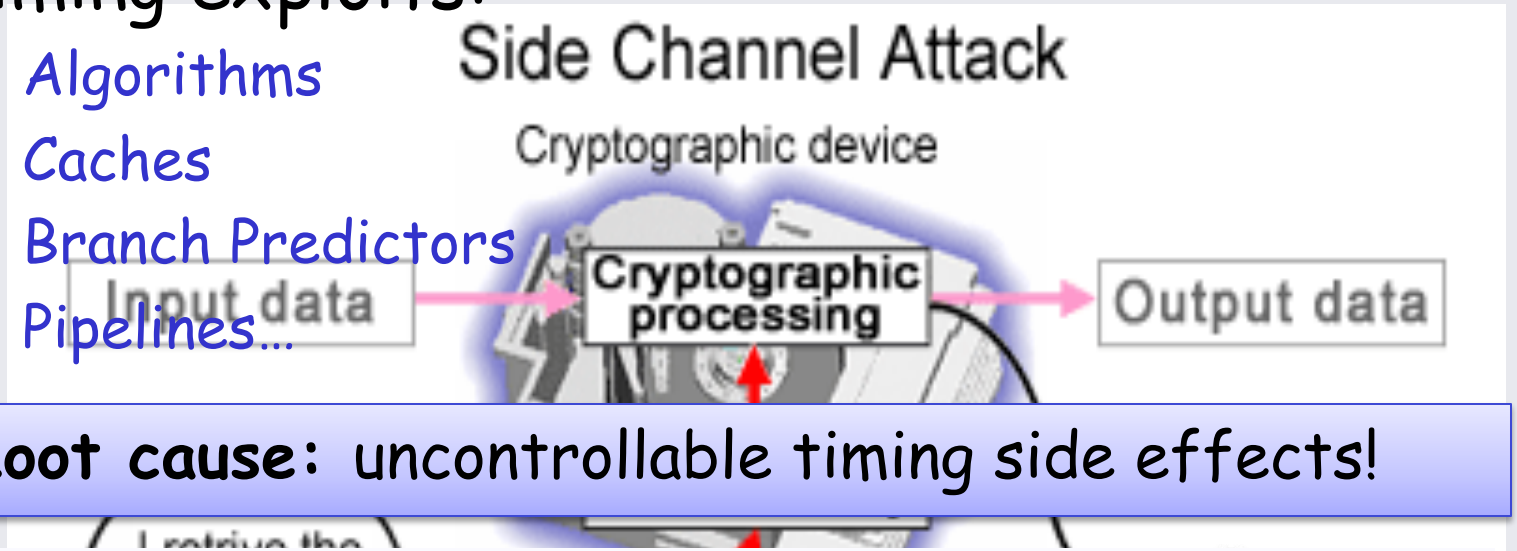


- 1D CFD Simulation – Network of Pipes
- Real-Time requirements: 5.33us
- Common Fuel Rail: 234 nodes

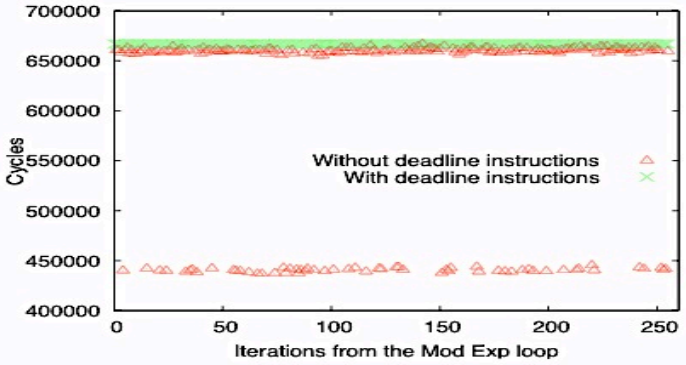
Implemented on Xilinx V6 FPGA

Timing Side-Channel Attacks

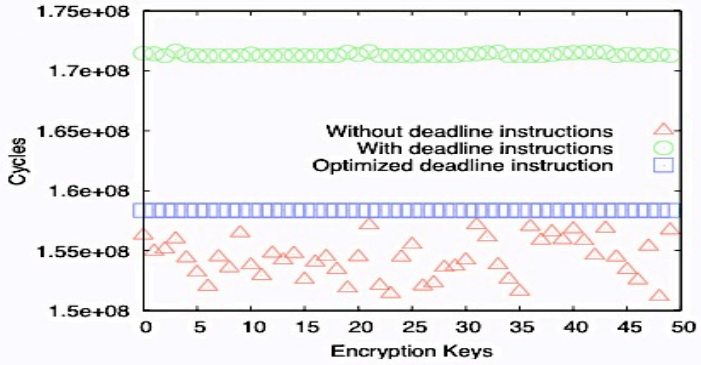
- Timing exploits:
  - Algorithms
  - Caches
  - Branch Predictors
  - Pipelines…



Root cause: uncontrollable timing side effects!



(a) Run time of Modular Exponent operation



(b) Run time of RSA operation

Summary

- Problem Statement:
  - Conventional methods are limiting the scaling of Cyber Physical Systems design because of its lack of precise timing control and analysis
- Solution:
  - To rethink the design of the bottom layers of abstraction, with emphasis on temporal predictability for Cyber Physical Systems
- Outcome of Research:
  - Precision Timed Architecture (PRET) for timing predictability and composability with ISA extensions for expressing temporal properties
  - To propose changes in the abstraction layer to expose "timing" thru lower layers, and propose a computer architecture that focuses on timing predictability and composability for Cyber Physical Systems.

Publications

- **Liu**, Viele, Wang, Lee, Andrade, A Heterogeneous Architecture for Evaluating Real-Time One Dimensional Computational Fluid Dynamics, FCCM 12
- Reineke, **Liu**, Patel, Kim, Lee, PRET DRAM Controller: Bank Privatization for Predictability and Temporal Isolation, CODES 11
- Bui, Lee, **Liu**, Patel, Reineke, Temporal Isolation on Multiprocessing Architectures, DAC 11
- **Liu**, Reineke, Lee, A PRET Architecture Supporting Concurrent Programs with Composable Timing Properties, ACSSC 10
- Edwards, Kim, Lee, **Liu**, Patel, Schoeberl, A Disruptive Computer Design Idea: Architectures with Repeatable Timing, ICCD 09
- **Liu**, Lickly, Patel, Lee, Poster Abstract: Timing Instructions - ISA Extensions for Timing Guarantees, RTAS 09
- **Liu** and McGrogan. Elimination of Side Channel Attacks on a Precision Timed Architecture, Technical Report, UCB 2009
- Lickly, **Liu**, Kim, Patel, Edwards, Lee, Predictable Programming on a Precision Timed Architecture, CASES 08

Thank You

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

Questions?

BACKUP SLIDES

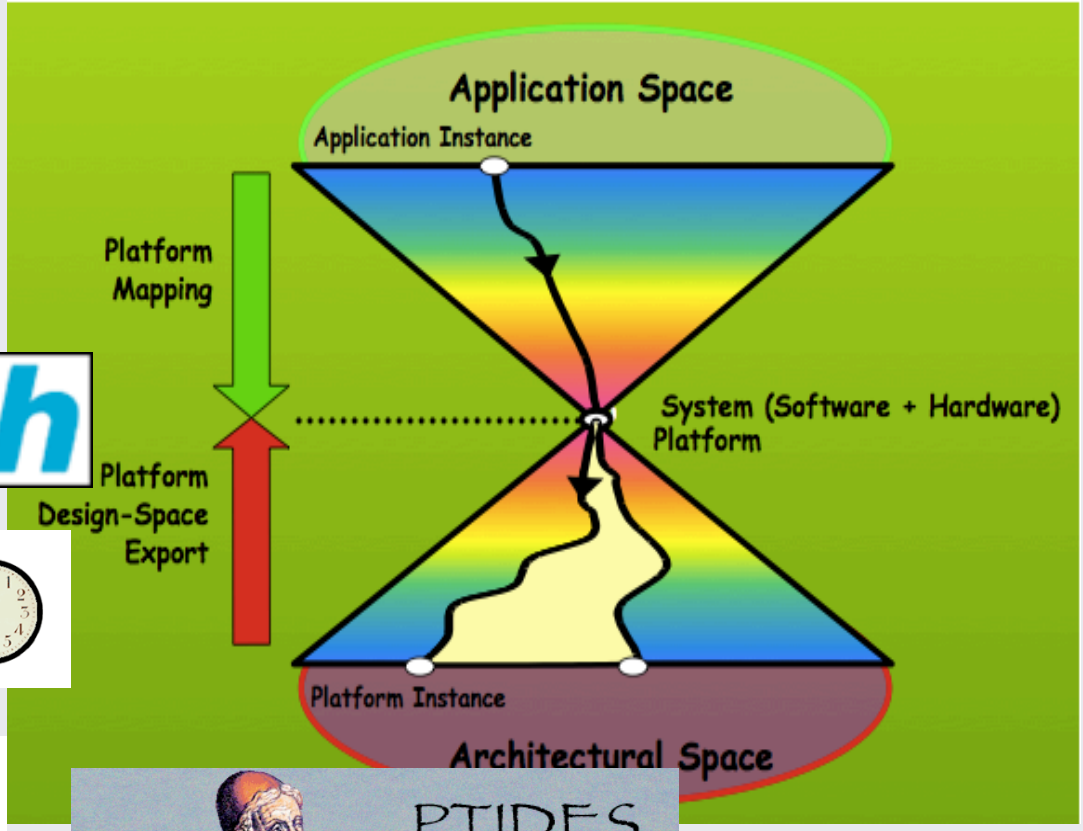
Thank You

- Qual Committee
- Edward A. Lee - Berkeley
- Hiren Patel – Univ. of Waterloo
- Martin Schoeberl – Univ. of Denmark
- Stephen A. Edwards – Columbia Univ.
- Ben Lickly, Sungjun Kim
- John Eidson, Marc Geilen, Sami Yehia (Thales), Maarten Wiggers, Jan Reineke, Slobodon Matic, Jia Zou
- Christopher Brooks, Mary Stewart
- My Family

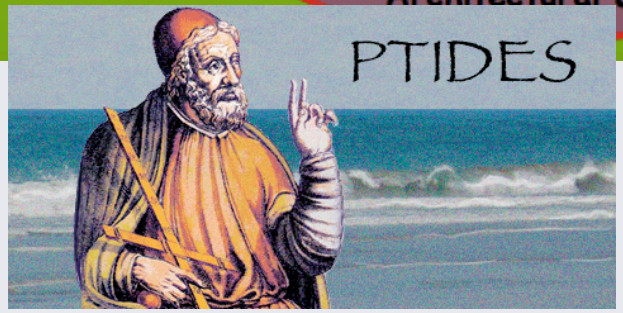
Research Efforts In All Fronts

TTTech

GIOTTO



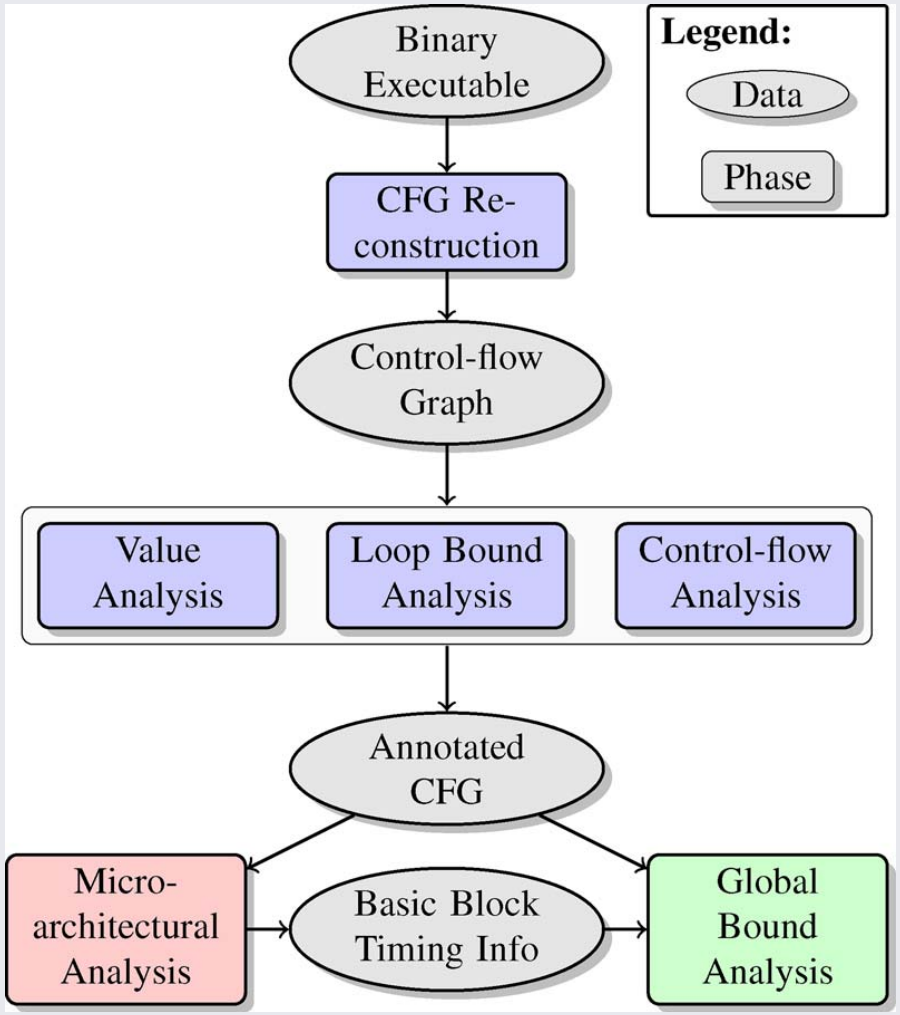
ESTEREL Technologies
An ISO 9001:2008 Certified Company



Definitions

- Predictability
  - The ability to analyze the execution time
- Repeatability
  - The ability to repeat the execution given the same inputs
- Composability
  - The functional and temporal behavior of an application is the same, irrespective of the presence or absence of other applications
- Robust
  - Small changes in input leads to small changes in output

WCET Analysis



Computer Architecture

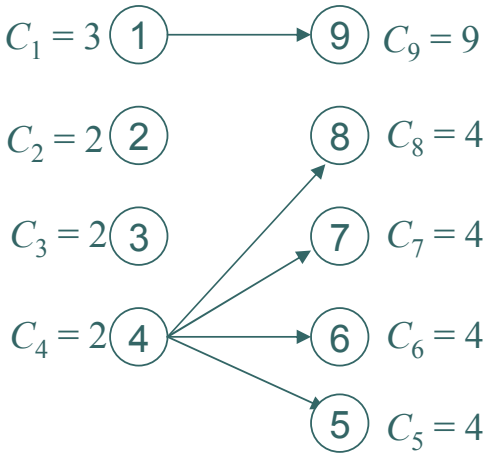
- Typical metrics in processor design for Embedded Systems
  - Performance (Average Case)
  - Power
  - Area (Size)
  - Compiler Support (Developmental Effort)
  - Cost
  - Multiple Context
  - Analyzability

Branch Prediction Anomaly Experiment

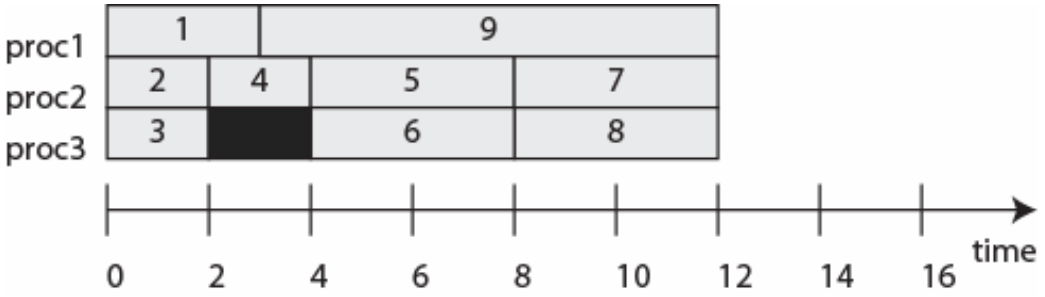
```
for{k=1; k<32; k++) {
  starttimer();
  for(n=0; n < 10000000; n++)    // OUTER LOOP
    {
      for(i=0; i < k; i++)  // INNER LOOP
        {
           __nop();   // Some compiler-dependent way to get a nop
        }
    }
  stoptimer();
  recordtime();
}
```

Richard's Anomalies

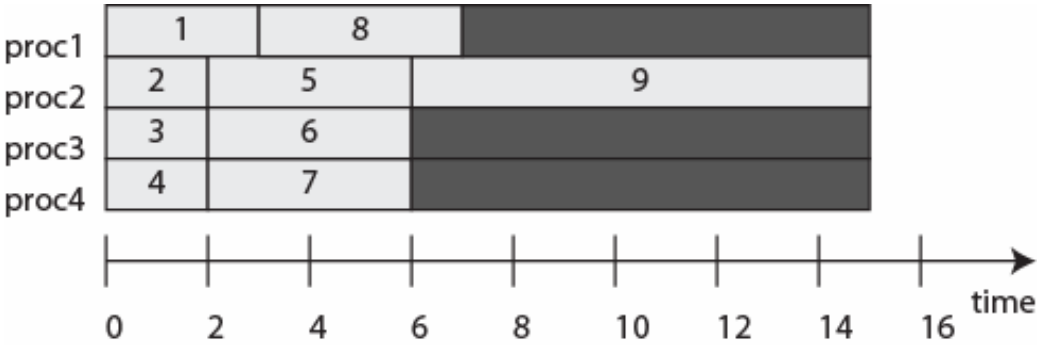
Increasing the number of processors



9 tasks with precedences and the shown execution times, where lower numbered tasks have higher priority than higher numbered tasks. Optimal 3 processor schedule:

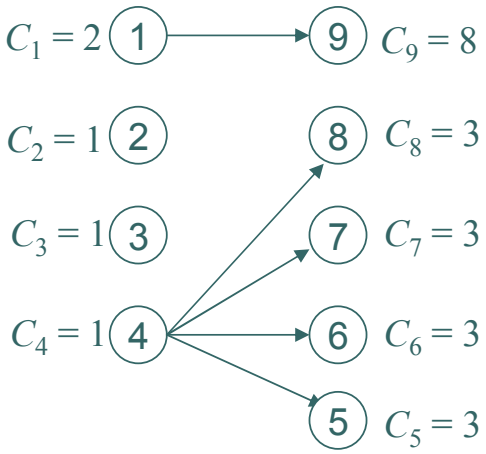


The optimal schedule with four processors has a longer execution time.

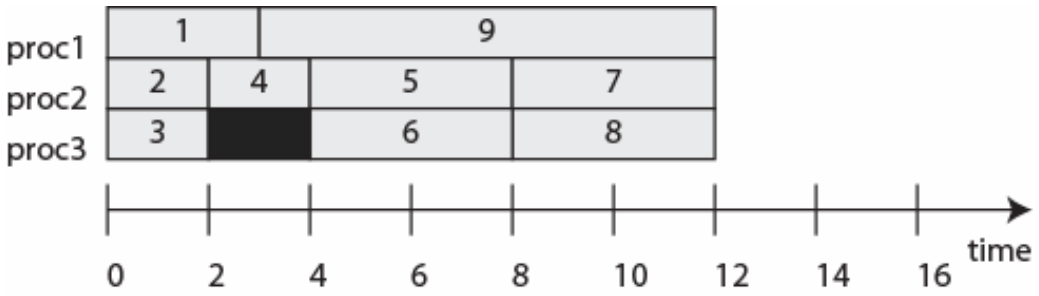


Richard's Anomalies

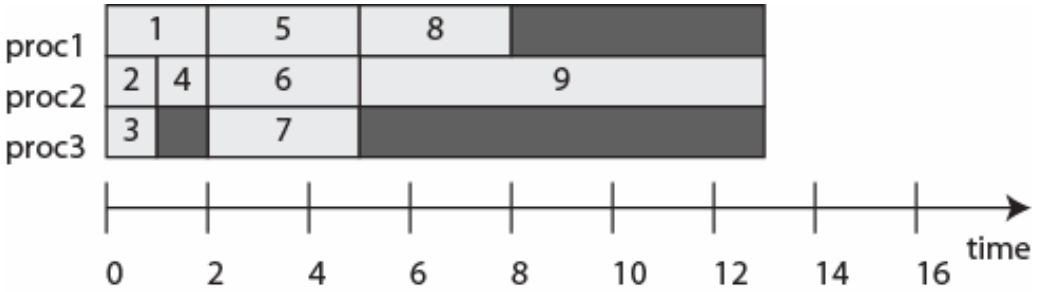
Reducing all execution times by 1



9 tasks with precedences and the shown execution times, where lower numbered tasks have higher priority than higher numbered tasks. Optimal 3 processor schedule:

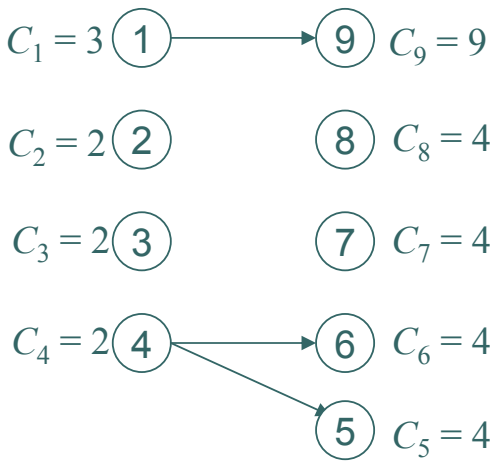


Reducing the computation times by 1 also results in a longer execution time.

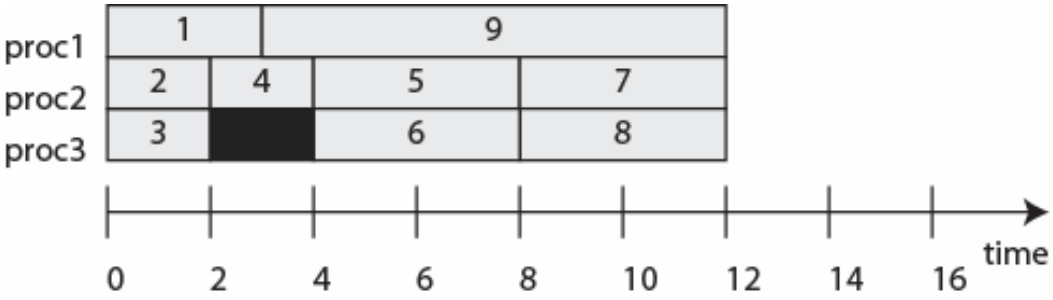


Richard's Anomalies

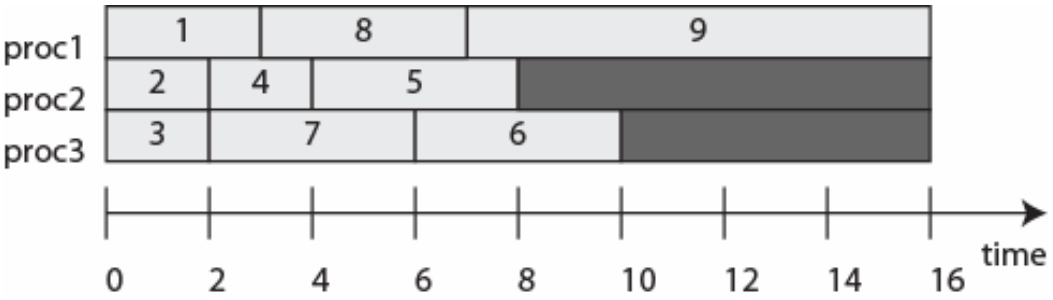
Removing precedence constraints



9 tasks with precedences and the shown execution times, where lower numbered tasks have higher priority than higher numbered tasks. Optimal 3 processor schedule:



Weakening precedence constraints can also result in a longer schedule.



Progress

Work Completed:

- SPARC instruction set simulator
  - C++ cycle accurate simulator
- PTARM architecture
  - Synthesizable VHDL ARM core
  - VGA controller and Serial Communication

Work in Progress:

- WCET analysis tool (~2 weeks)
- Benchmarking the pipeline (~ 1 semester)
- Scratchpad allocation with timed programming models (~1 semester)
- Proof of concept workflow (~ 1 semesters)

Contribution

- Expose "time" in the abstraction layers.
  - ISA extensions to specify temporal properties

- Propose an architecture that allows for timing predictability and composable resource sharing.