18-447
Computer Architecture
Lecture 1: Introduction and Basics

Prof. Onur Mutlu
Carnegie Mellon University
Spring 2014, 1/13/2014

I Hope You Are Here for This

18-213/243

"C" as a model of computation

Programmer's view of a computer system works

- How does an assembly program end up executing as digital logic?
- What happens in-between?
- How is a computer designed using logic gates and wires to satisfy specific goals?

Architect/microarchitect's view: How to design a computer that meets system design goals.

Choices critically affect both the SW programmer and the HW designer

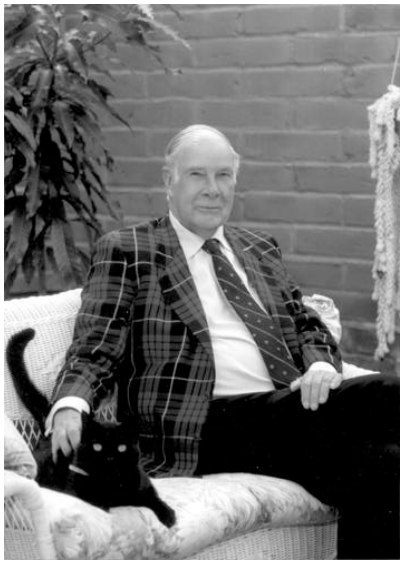
HW designer's view of a computer system works

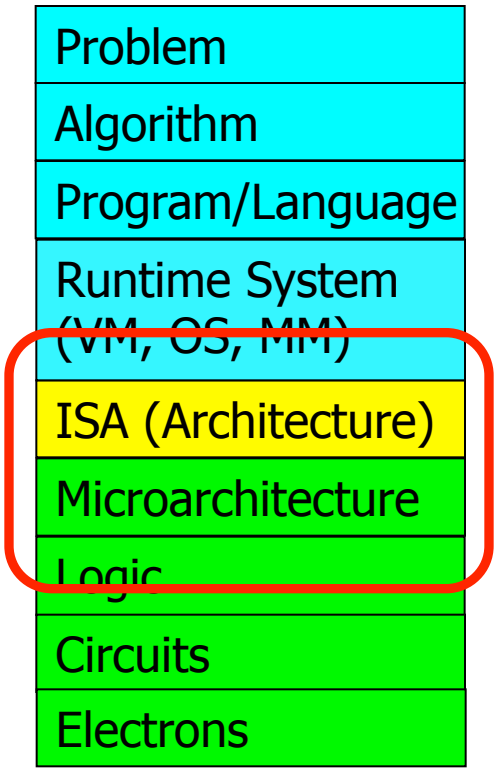
18-240

Digital logic as a model of computation

Levels of Transformation

"The purpose of computing is insight" (*Richard Hamming*)
We gain and generate insight by solving problems
How do we ensure problems are solved by electrons?





The Power of Abstraction

- Levels of transformation create abstractions
  - Abstraction: A higher level only needs to know about the interface to the lower level, not how the lower level is implemented
  - E.g., high-level language programmer does not really need to know what the ISA is and how a computer executes instructions
- Abstraction improves productivity
  - No need to worry about decisions made in underlying levels
  - E.g., programming in Java vs. C vs. assembly vs. binary vs. by specifying control signals of each transistor every cycle
- Then, why would you want to know what goes on underneath or above?

Crossing the Abstraction Layers

- As long as everything goes well, not knowing what happens in the underlying level (or above) is not a problem.

- What if
  - The program you wrote is running slow?
  - The program you wrote does not run correctly?
  - The program you wrote consumes too much energy?

- What if
  - The hardware you designed is too hard to program?
  - The hardware you designed is too slow because it does not provide the right primitives to the software?

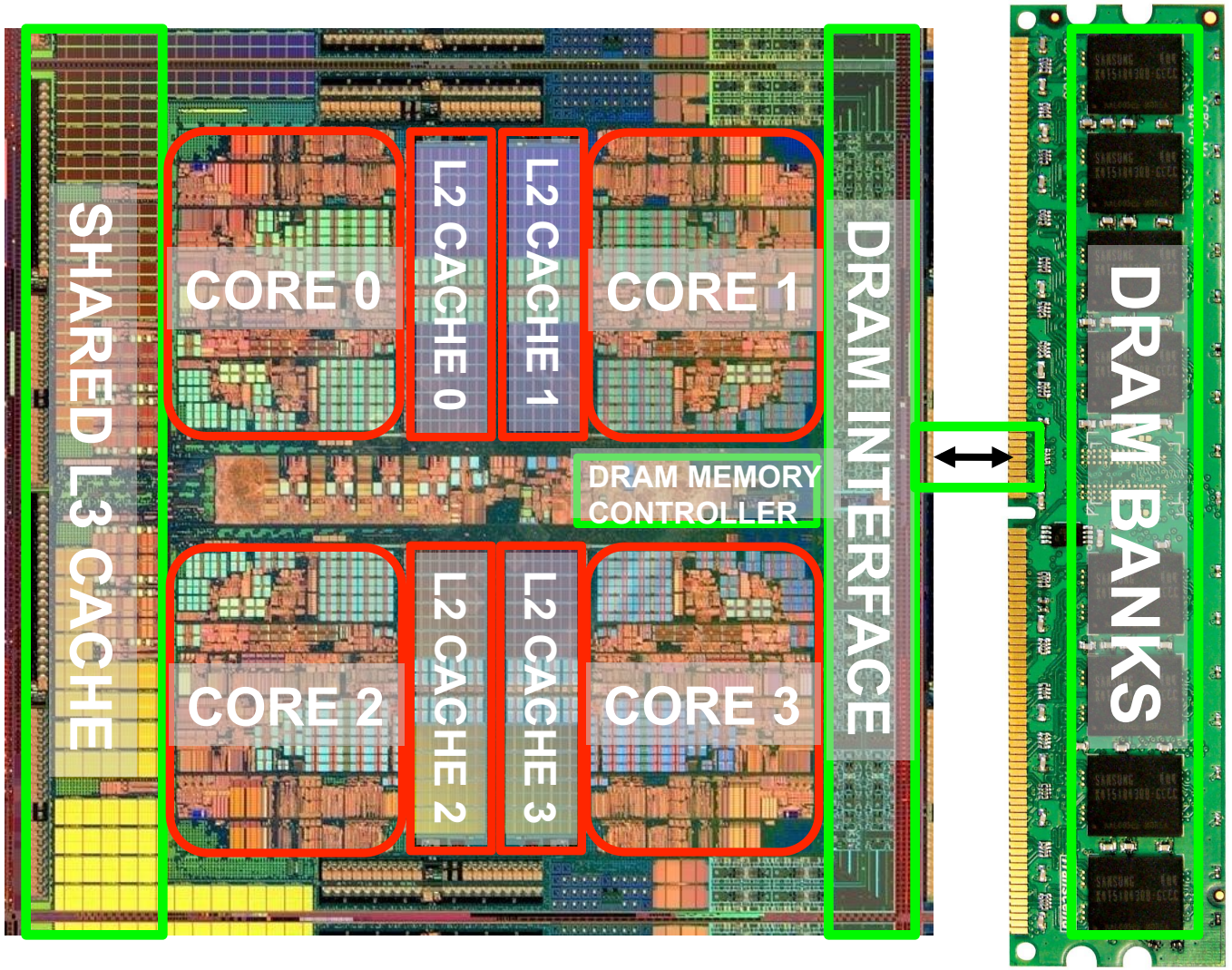
- What if
  - You want to design a much more efficient and higher performance system?

Crossing the Abstraction Layers

- Two key goals of this course are
  - to understand how a processor works underneath the software layer and how decisions made in hardware affect the software/programmer
  - to enable you to be comfortable in making design and optimization decisions that cross the boundaries of different layers and system components

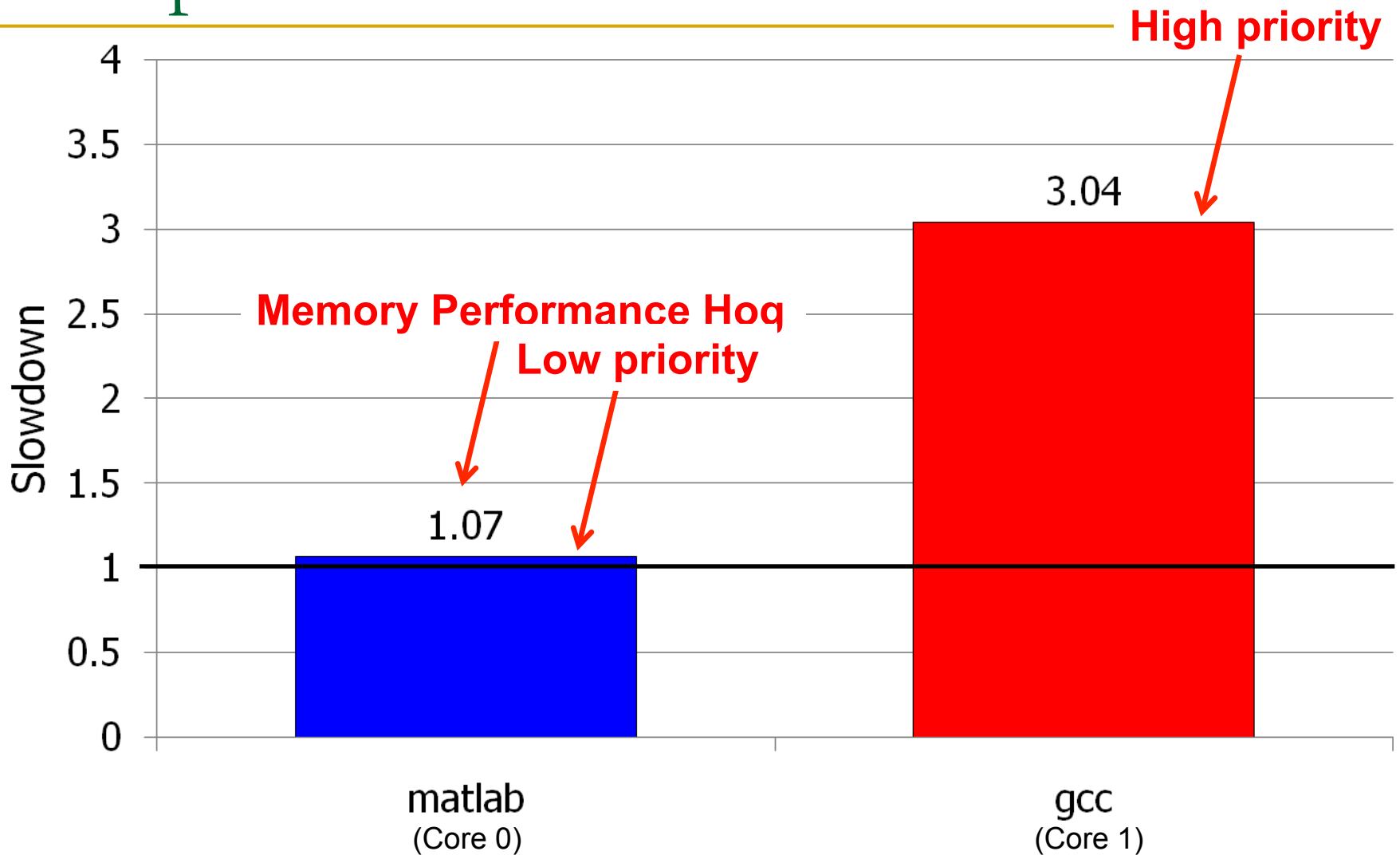
An Example: Multi-Core Systems

Multi-Core Chip



*Die photo credit: AMD Barcelona

Unexpected Slowdowns in Multi-Core



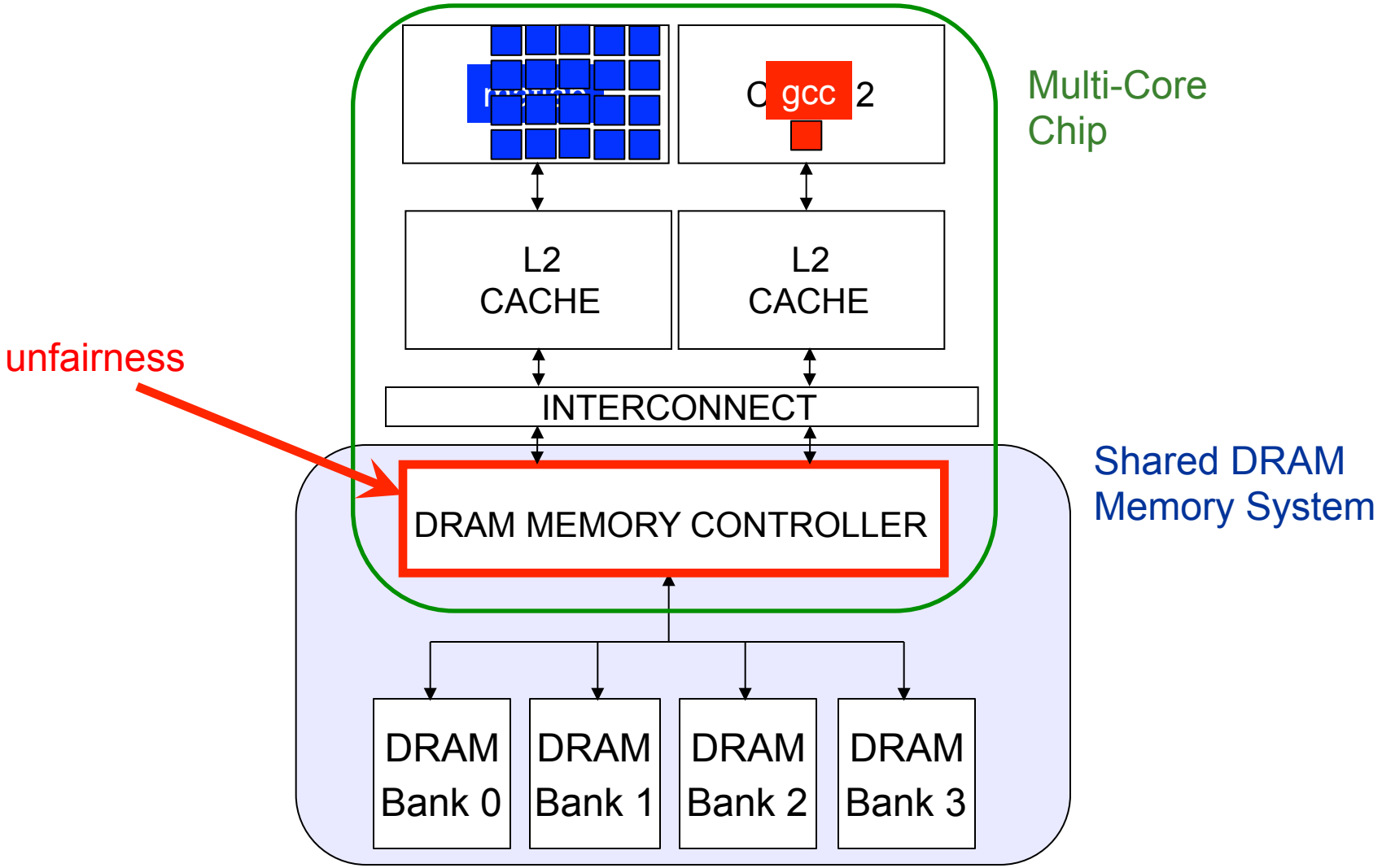
Moscibroda and Mutlu, "Memory performance attacks: Denial of memory service in multi-core systems," USENIX Security 2007.

A Question or Two

- Can you figure out why there is a disparity in slowdowns if you do not know how the processor executes the programs?

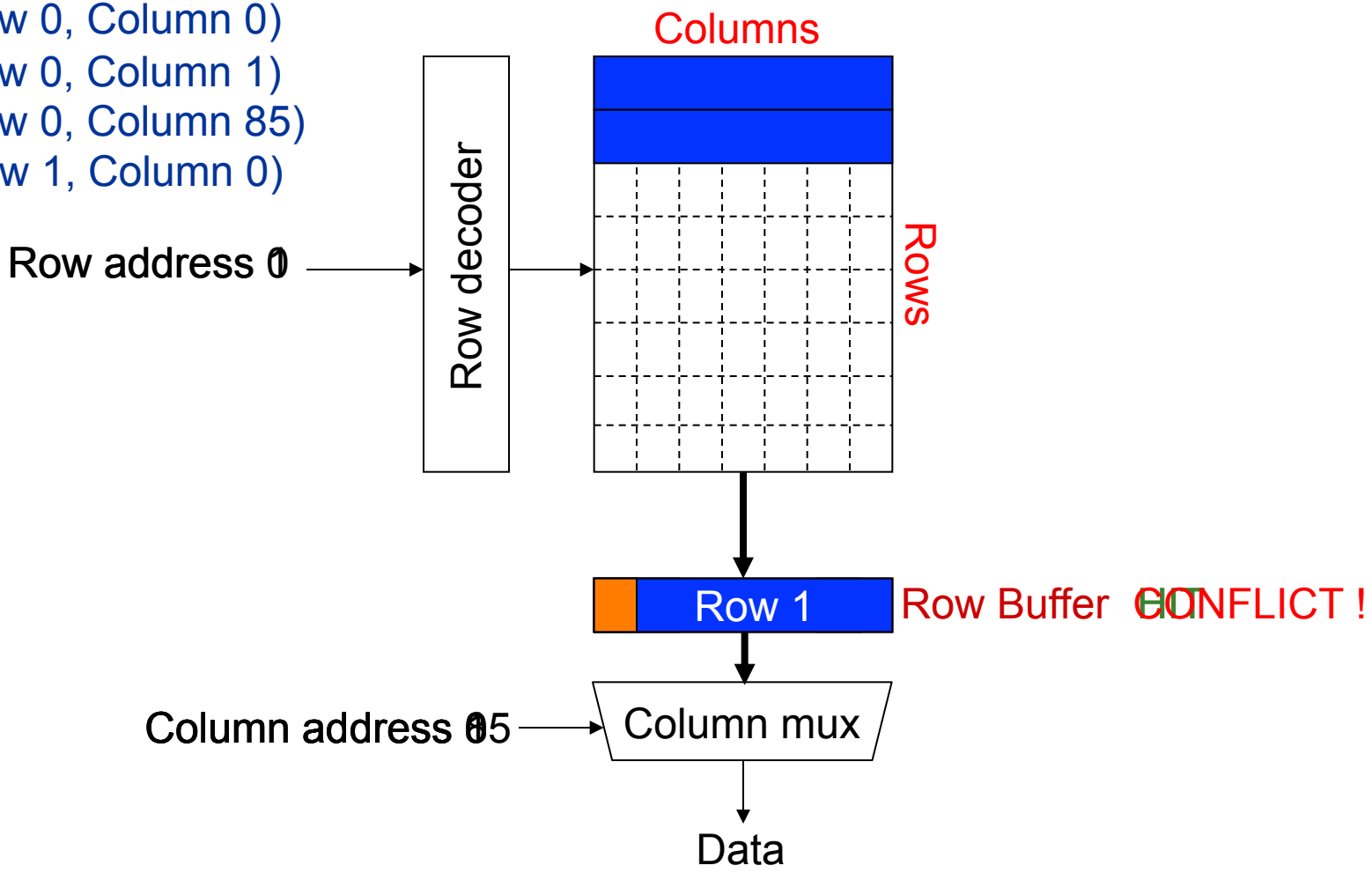
- Can you fix the problem without knowing what is happening "underneath"?

Why the Disparity in Slowdowns?



DRAM Bank Operation

Access Address:
(Row 0, Column 0)
(Row 0, Column 1)
(Row 0, Column 85)
(Row 1, Column 0)



DRAM Controllers

- A row-conflict memory access takes significantly longer than a row-hit access
- Current controllers take advantage of the row buffer
- Commonly used scheduling policy (FR-FCFS) [Rixner 2000]*
  (1) Row-hit first: Service row-hit memory accesses first
  (2) Oldest-first: Then service older accesses first
- This scheduling policy aims to maximize DRAM throughput

*Rixner et al., "Memory Access Scheduling," ISCA 2000.
*Zuravleff and Robinson, "Controller for a synchronous DRAM …," US Patent 5,630,096, May 1997.

The Problem

- Multiple threads share the DRAM controller
- DRAM controllers designed to maximize DRAM throughput

- DRAM scheduling policies are thread-unfair
  - Row-hit first: unfairly prioritizes threads with high row buffer locality
    - Threads that keep on accessing the same row
  - Oldest-first: unfairly prioritizes memory-intensive threads

- DRAM controller vulnerable to denial of service attacks
  - Can write programs to exploit unfairness

A Memory Performance Hog

```
// initialize large arrays A, B

for (j=0; j<N; j++) {
    index = j*linesize;
    A[index] = B[index];
    …
}
```

streaming

```
// initialize large arrays A, B

for (j=0; j<N; j++) {
    index = rand();
    A[index] = B[index];
    …
}
```

random

STREAM

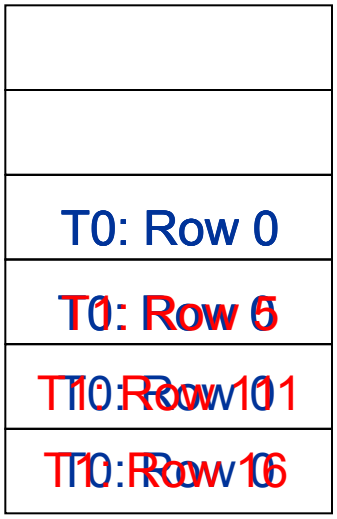
- Sequential memory access
- Very high row buffer locality (96% hit rate)
- Memory intensive

RANDOM

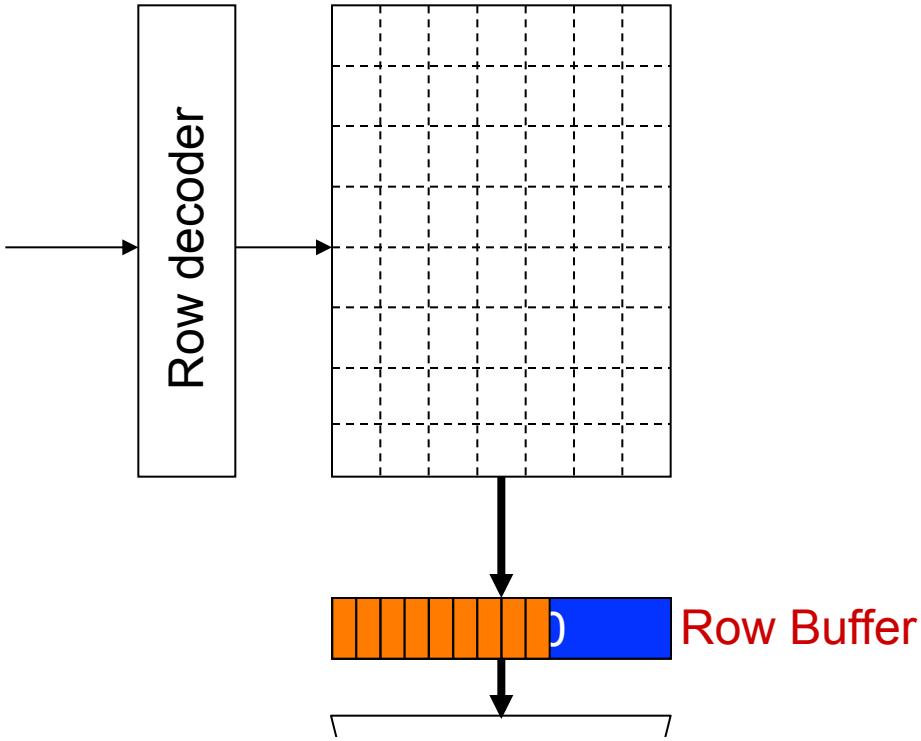
- Random memory access
- Very low row buffer locality (3% hit rate)
- Similarly memory intensive

Moscibroda and Mutlu, "Memory Performance Attacks," USENIX Security 2007.

What Does the Memory Hog Do?



Memory Request Buffer



Row size: 8KB, cache block size: 64B

128 (8KB/64B) requests of T0 serviced before T1

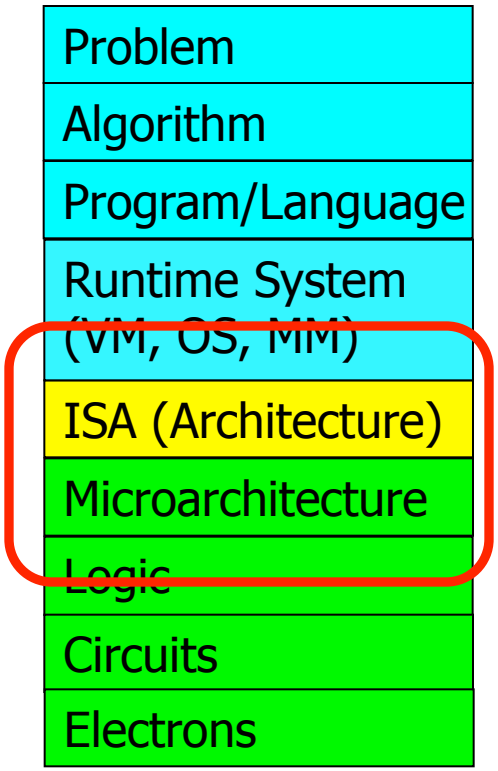
Moscibroda and Mutlu, "Memory Performance Attacks," USENIX Security 2007.

Now That We Know What Happens Underneath

- How would you solve the problem?

- What is the right place to solve the problem?
  - Programmer?
  - System software?
  - Compiler?
  - Hardware (Memory controller)?
  - Hardware (DRAM)?
  - Circuits?

- Two other goals of this course:
  - Enable you to think critically
  - Enable you to think broadly



If You Are Interested … Further Readings

- Onur Mutlu and Thomas Moscibroda,
  "Stall-Time Fair Memory Access Scheduling for Chip Multiprocessors"
  *Proceedings of the 40th International Symposium on Microarchitecture (**MICRO**)*, pages 146-158, Chicago, IL, December 2007. Slides (ppt)

- Sai Prashanth Muralidhara, Lavanya Subramanian, Onur Mutlu, Mahmut Kandemir, and Thomas Moscibroda,
  "Reducing Memory Interference in Multicore Systems via Application-Aware Memory Channel Partitioning"
  *Proceedings of the 44th International Symposium on Microarchitecture (**MICRO**)*, Porto Alegre, Brazil, December 2011. Slides (pptx)

Takeaway

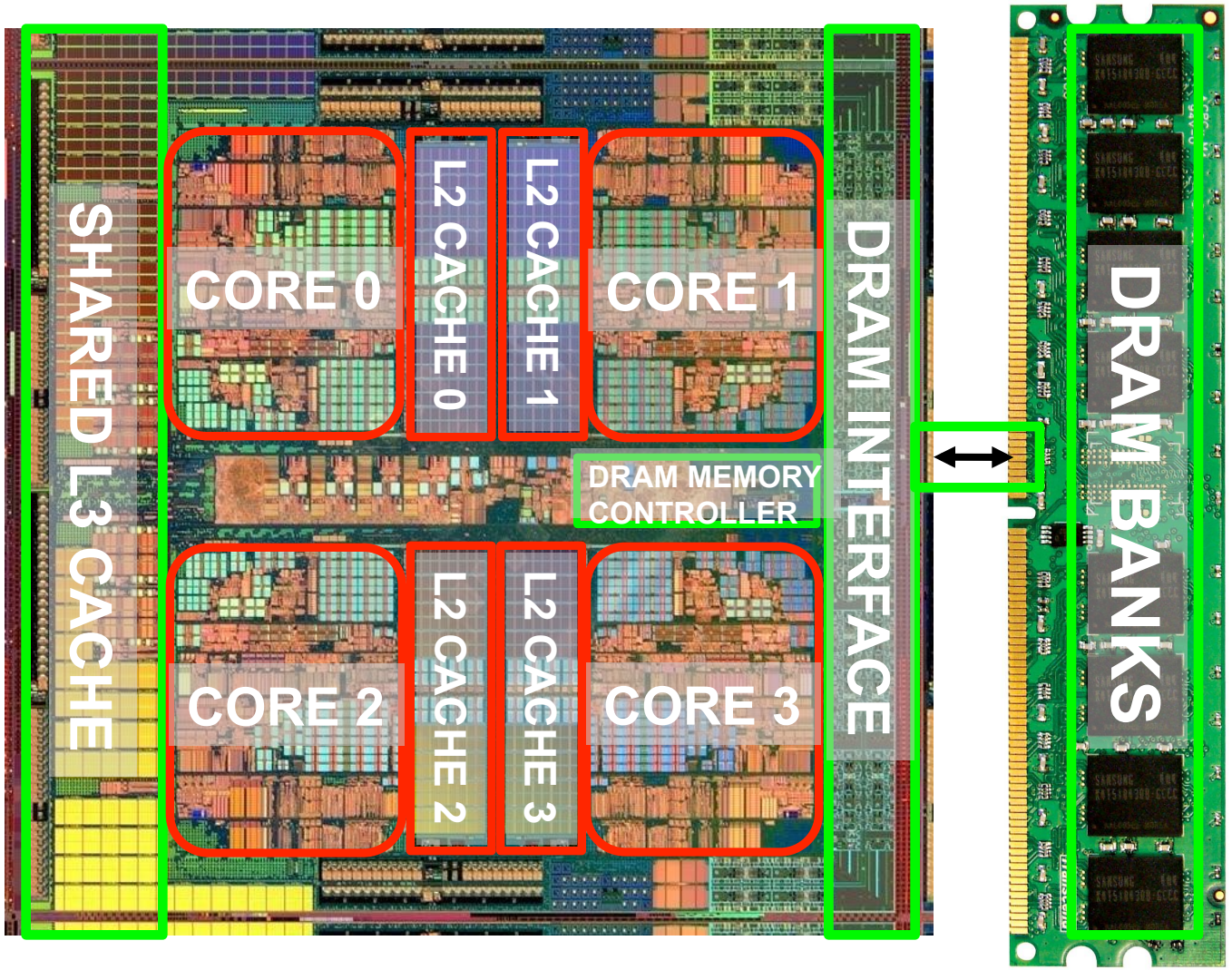
- Breaking the abstraction layers (between components and transformation hierarchy levels) and knowing what is underneath enables you to solve problems

Another Example

- DRAM Refresh

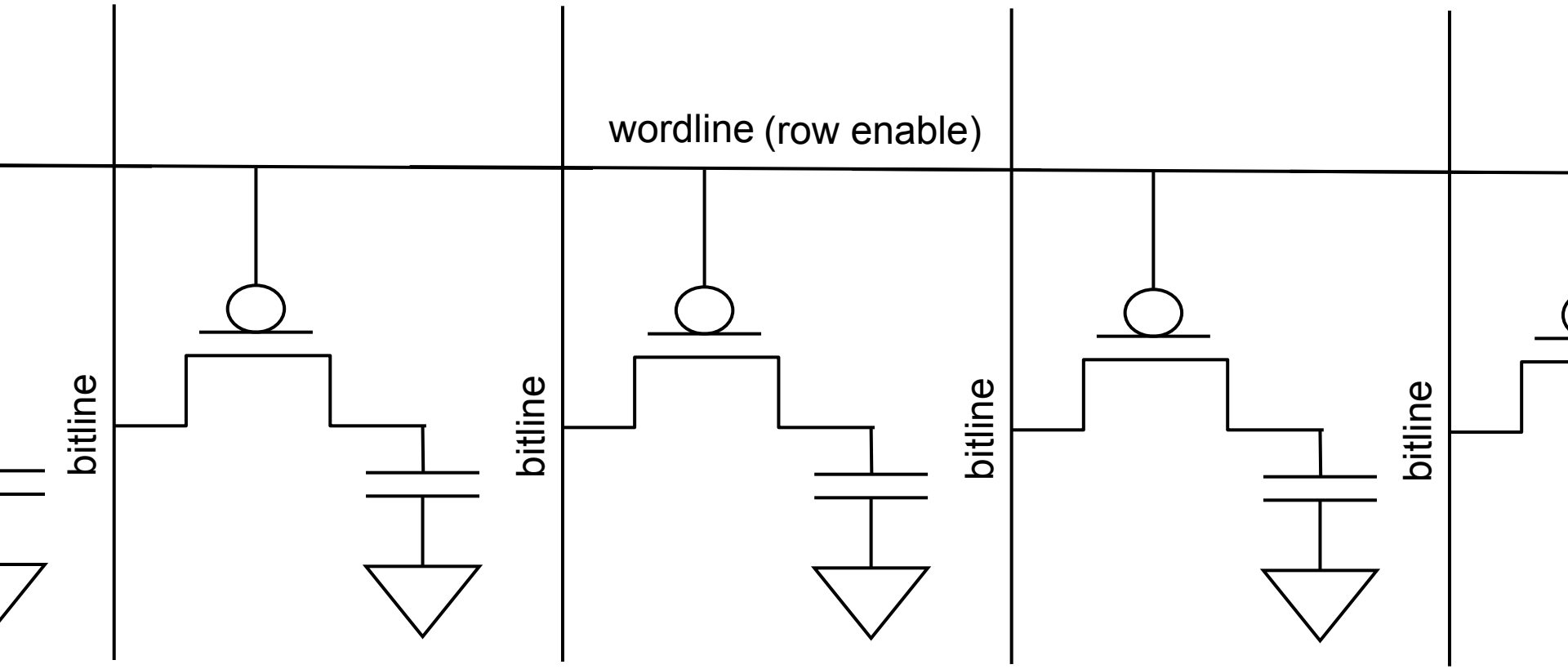
DRAM in the System

Multi-Core Chip



*Die photo credit: AMD Barcelona

A DRAM Cell



- A DRAM cell consists of a capacitor and an access transistor
- It stores data in terms of charge in the capacitor
- A DRAM chip consists of (10s of 1000s of) rows of such cells

DRAM Refresh

- DRAM capacitor charge leaks over time

- The memory controller needs to refresh each row periodically to restore charge
  - Activate each row every N ms
  - Typical N = 64 ms

- Downsides of refresh

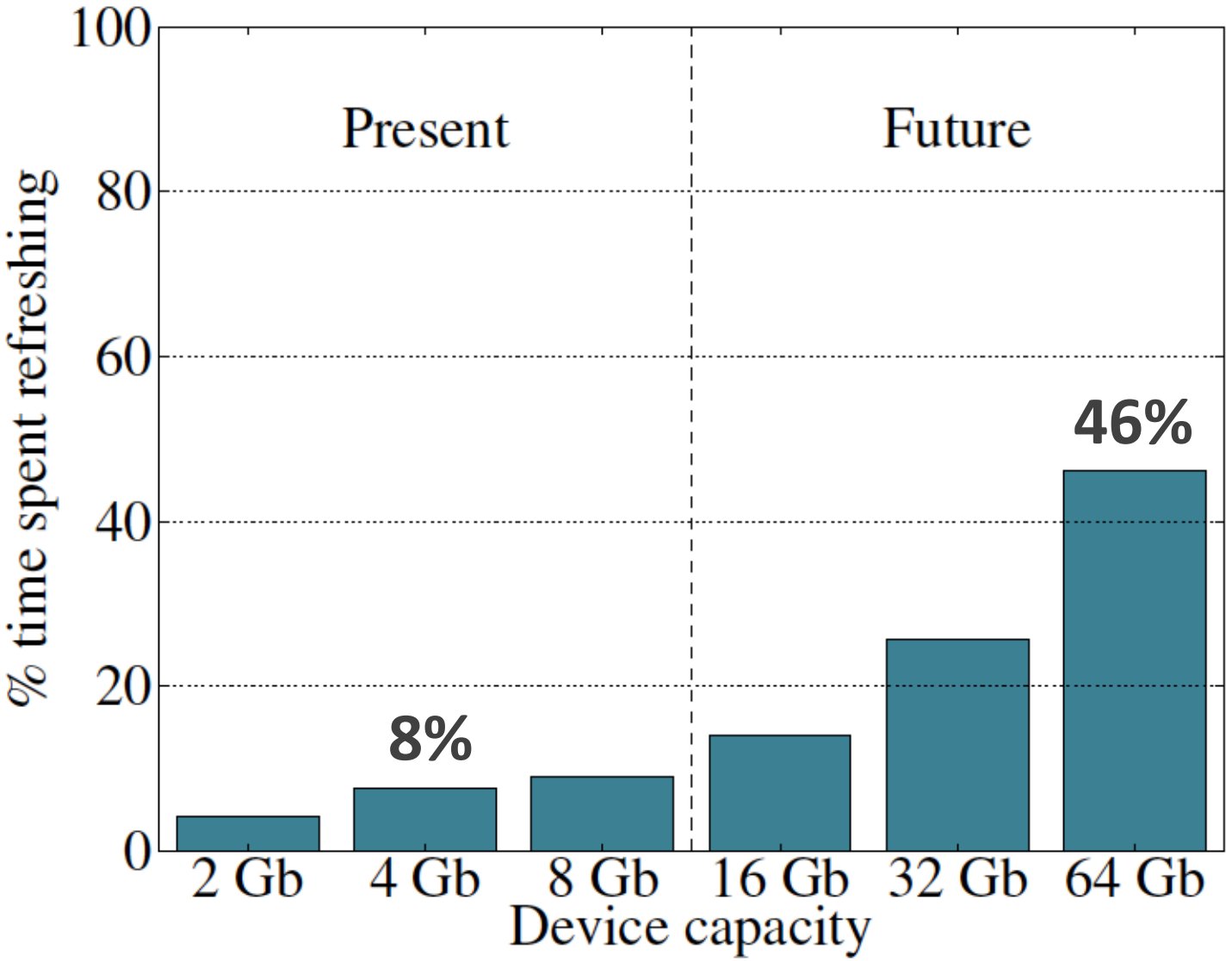
  -- Energy consumption: Each refresh consumes energy

  -- Performance degradation: DRAM rank/bank unavailable while refreshed

  -- QoS/predictability impact: (Long) pause times during refresh

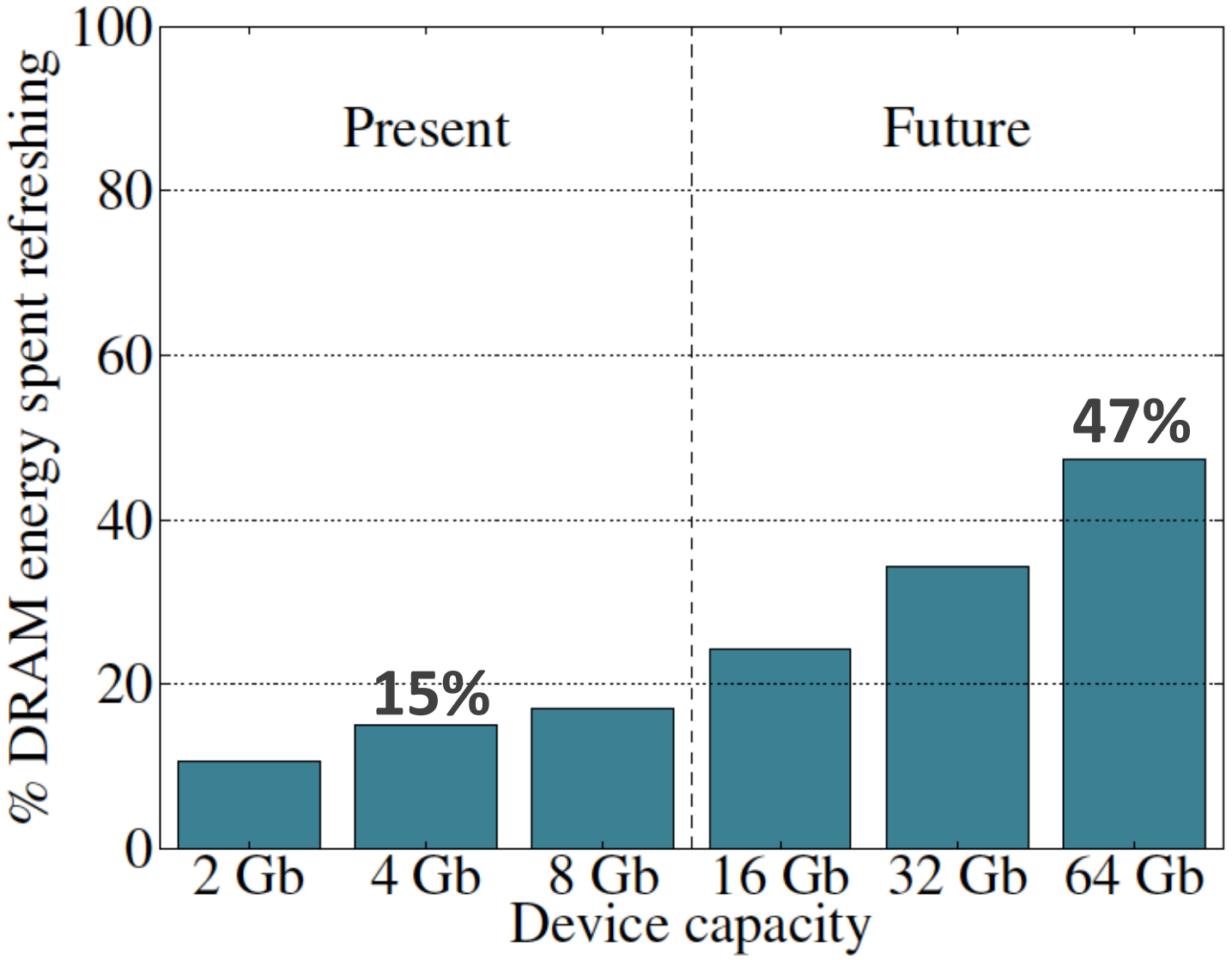
  -- Refresh rate limits DRAM capacity scaling

Refresh Overhead: Performance



Liu et al., "RAIDR: Retention-Aware Intelligent DRAM Refresh," ISCA 2012.

Refresh Overhead: Energy



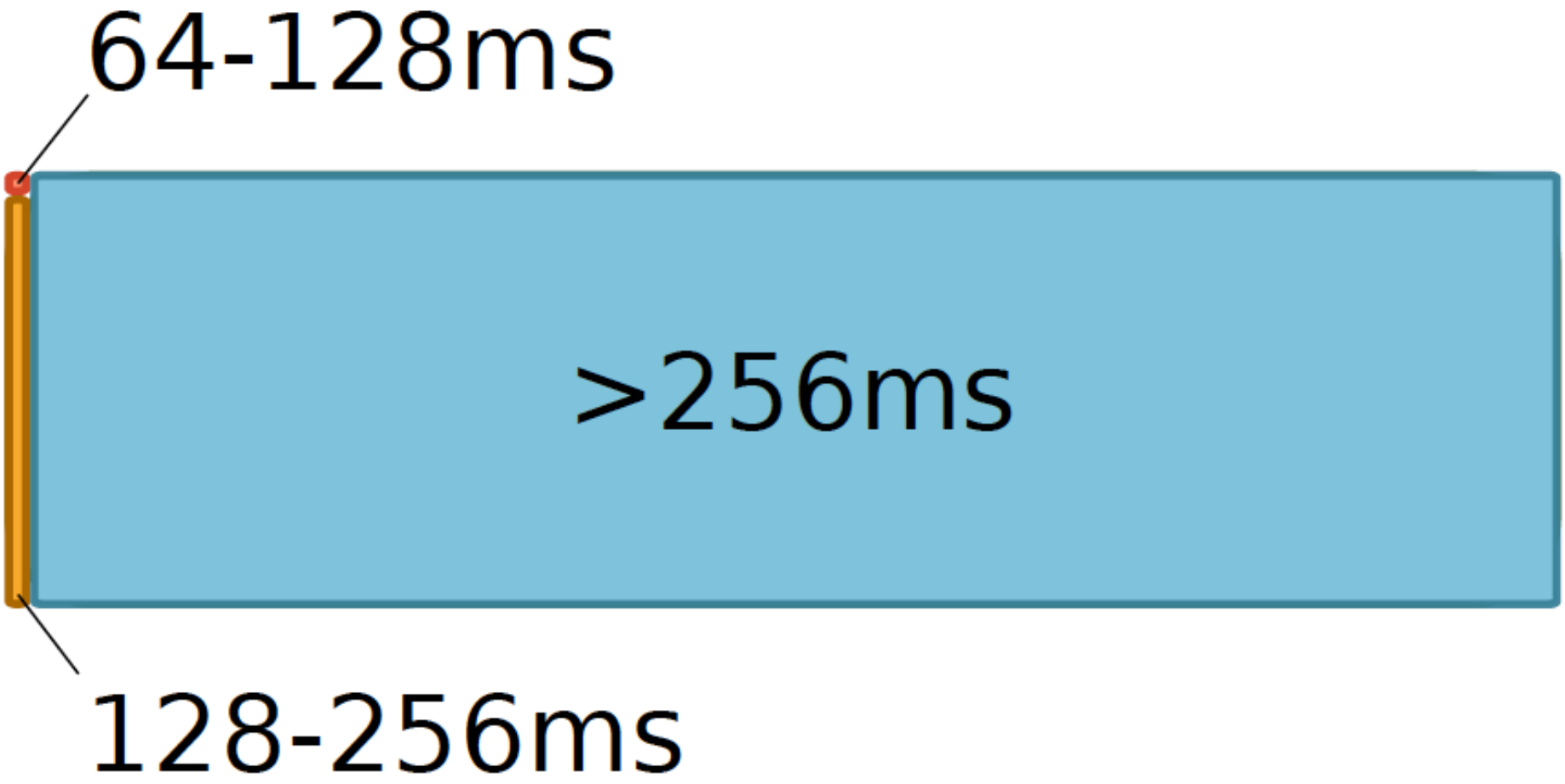
Liu et al., "RAIDR: Retention-Aware Intelligent DRAM Refresh," ISCA 2012.

How Do We Solve the Problem?

- Do we need to refresh all rows every 64ms?

- What if we knew what happened underneath and exposed that information to upper layers?

Underneath: Retention Time Profile of DRAM



Taking Advantage of This Profile

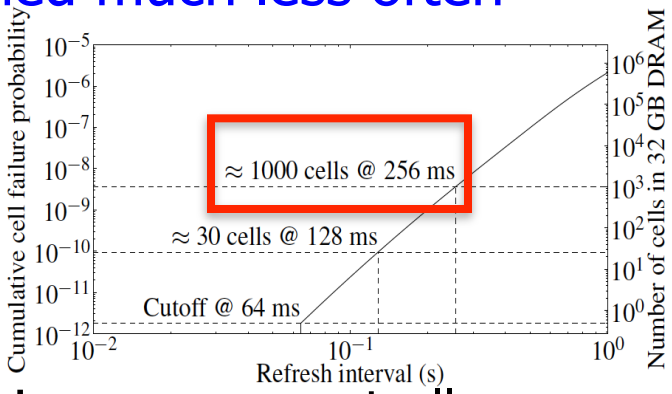
- Expose this retention time profile information to
  - the memory controller
  - the operating system
  - the programmer?
  - the compiler?

- How much information to expose?
  - Affects hardware/software overhead, power consumption, verification complexity, cost

- How to determine this profile information?
  - Also, who determines it?

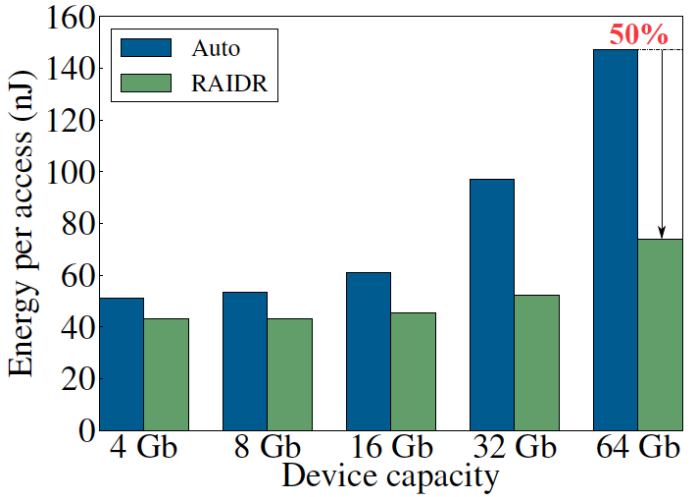
An Example: RAIDR

- Observation: Most DRAM rows can be refreshed much less often without losing data **[Kim+, EDL'09][Liu+ ISCA'13]**



- Key idea: Refresh rows containing weak cells more frequently, other rows less frequently
  1. Profiling: Profile retention time of all rows
  2. Binning: Store rows into bins by retention time in memory controller
     Efficient storage with Bloom Filters (only 1.25KB for 32GB memory)
  3. Refreshing: Memory controller refreshes rows in different bins at different rates

- Results: 8-core, 32GB, SPEC, TPC-C, TPC-H
  - 74.6% refresh reduction @ 1.25KB storage
  - ~16%/20% DRAM dynamic/idle power reduction
  - ~9% performance improvement
  - Benefits increase with DRAM capacity



Liu et al., "RAIDR: Retention-Aware Intelligent DRAM Refresh," ISCA 2012.

If You Are Interested … Further Readings

- Jamie Liu, Ben Jaiyen, Richard Veras, and Onur Mutlu,
  "RAIDR: Retention-Aware Intelligent DRAM Refresh"
  *Proceedings of the 39th International Symposium on Computer Architecture (**ISCA**)*, Portland, OR, June 2012. Slides (pdf)

- Onur Mutlu,
  "Memory Scaling: A Systems Architecture Perspective"
  *Technical talk at MemCon 2013 (**MEMCON**)*, Santa Clara, CA, August 2013.
  Slides (pptx) (pdf) Video

Takeaway

- Breaking the abstraction layers (between components and transformation hierarchy levels) and knowing what is underneath enables you to solve problems and design better future systems

- Cooperation between multiple components and layers can enable more effective solutions and systems

Recap: Some Goals of 447

- Teach/enable/empower you you to:
  - Understand how a processor works
  - Implement a simple processor (with not so simple parts)
  - Understand how decisions made in hardware affect the software/programmer as well as hardware designer
  - Think critically (in solving problems)
  - Think broadly across the levels of transformation
  - Understand how to analyze and make tradeoffs in design

Agenda

- Intro to 18-447
  - Course logistics, info, requirements
  - What 447 is about
  - Lab assignments
  - Homeworks, readings, etc
- Assignments for the next two weeks
  - Homework 0 (due Jan 22)
  - Homework 1 (due Jan 29)
  - Lab 1 (due Jan 24)
- Basic concepts in computer architecture

Handouts for Today

- Online
  - Homework 0
  - Syllabus

Course Info: Who Are We?

- Instructor: Prof. Onur Mutlu
  - onur@cmu.edu
  - Office: CIC 4105
  - Office Hours: W 2:30-3:30pm (or by appointment)
  - http://www.ece.cmu.edu/~omutlu
  - PhD from UT-Austin, worked at Microsoft Research, Intel, AMD
  - Research and teaching interests:
    - Computer architecture, hardware/software interaction
    - Many-core systems
    - Memory and storage systems
    - Improving programmer productivity
    - Interconnection networks
    - Hardware/software interaction and co-design (PL, OS, Architecture)
    - Fault tolerance
    - Hardware security
    - Algorithms and architectures for bioinformatics, genomics, health applications

Course Info: Who Are We?

- Teaching Assistants
  - Rachata Ausavarungnirun
    - rachata@cmu.edu
    - Office hours: Wed 4:30-6:30pm
  - Varun Kohli
    - vkohli@andrew.cmu.edu
    - Office hours: Thu 4.30-6.30pm
  - Xiao Bo Zhao
    - xiaoboz@andrew.cmu.edu
    - Office hours: Tue 4:30-6:30pm
  - Paraj Tyle
    - ptyle@cmu.edu
    - Office hours: Fri 3-5pm

Your Turn

- Who are you?

- Homework 0 (absolutely required)
  - Your opportunity to tell us about yourself
  - Due Jan 22 (midnight)
  - Attach your picture (absolutely required)
  - Submit via AFS

- All grading predicated on receipt of Homework 0

Where to Get Up-to-date Course Info?

- Website: http://www.ece.cmu.edu/~ece447
  - Lecture notes
  - Project information
  - Homeworks
  - Course schedule, handouts, papers, FAQs
- Your email
- Me and the Tas
- Piazza

Lecture and Lab Locations, Times

- Lectures:
  - MWF 12:30-2:20pm
  - Scaife Hall 219
  - Attendance is for your benefit and is therefore important
  - Some days, we may have recitation sessions or guest lectures

- Recitations:
  - T 10:30am-1:20pm, Th 1:30-4:20pm, F 6:30-9:20pm
  - Hamerschlag Hall 1303
  - You can attend *any* session
  - Goals: to enhance your understanding of the lecture material, help you with homework assignments, exams, and labs, and get one-on-one help from the TAs on the labs.

Tentative Course Schedule

- Tentative schedule is in syllabus
- To get an idea of topics, you can look at last year's schedule, lectures, videos, etc:
  - http://www.ece.cmu.edu/~ece447/s13

- But don't believe the "static" schedule
- Systems that perform best are usually dynamically scheduled
  - Static vs. Dynamic scheduling
  - Compile time vs. Run time

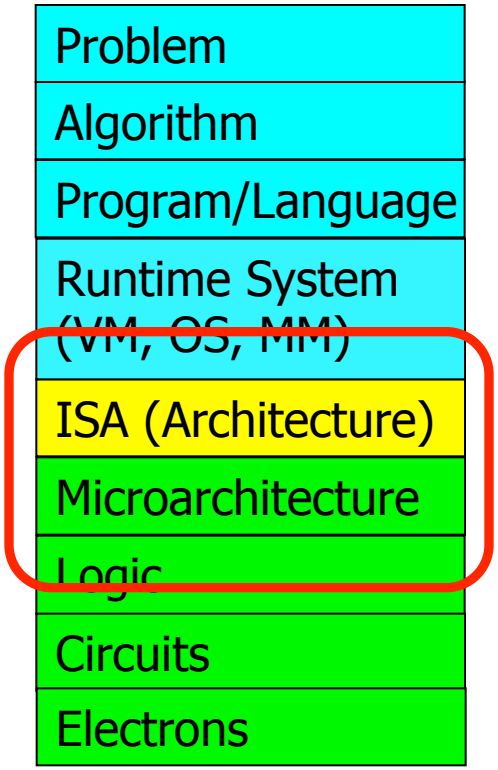
What Will You Learn

- **Computer Architecture:** The science and art of designing, selecting, and interconnecting hardware components and designing the hardware/software interface to create a computing system that meets functional, performance, energy consumption, cost, and other specific goals.

- **Traditional definition:** "The term *architecture* is used here to describe the attributes of a programmer, i.e., the conceptual s behavior as distinct from the orga and controls, the logic design, and implementation." *Gene Amdahl*, IB 1964

Dr. Amdahl holding a 100gate LSI air-cooled chip. On his desk is a circuit board with the chips on it. This circuit board was for an Amdahl 470 V/6 (photograph dated March 1973).

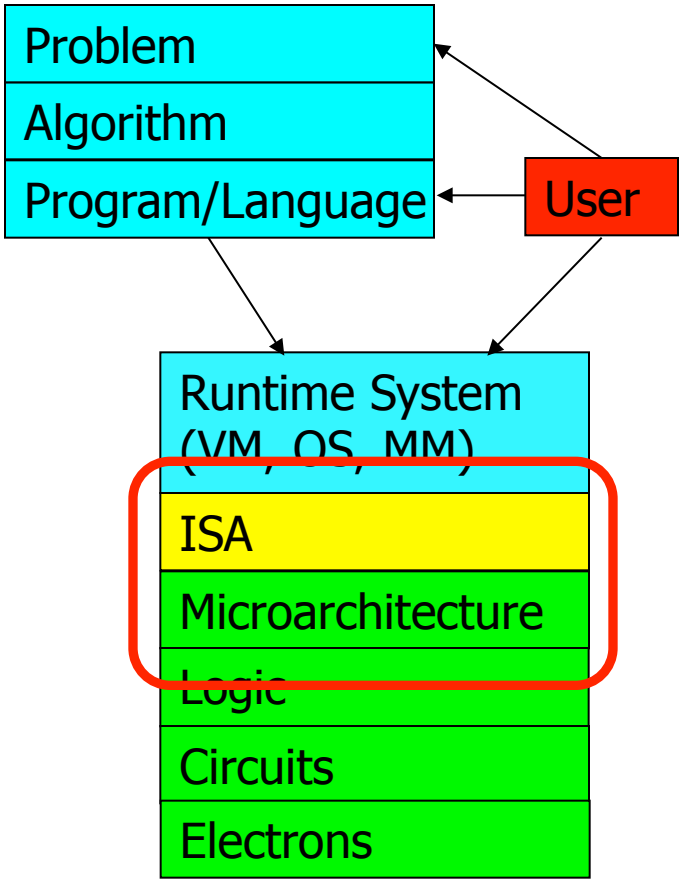
Computer Architecture in Levels of Transformation



- Read: Patt, "Requirements, Bottlenecks, and Good Fortune: Agents for Microprocessor Evolution," Proceedings of the IEEE 2001.

Levels of Transformation, Revisited

- A user-centric view: computer designed for users



- The entire stack should be optimized for user

What Will You Learn?

- Fundamental principles and tradeoffs in designing the hardware/software interface and major components of a modern programmable microprocessor
  - Focus on state-of-the-art (and some recent research and trends)
  - Trade-offs and how to make them
- How to design, implement, and evaluate a functional modern processor
  - Semester-long lab assignments
  - A combination of RTL implementation and higher-level simulation
  - Focus is on functionality (and some focus on "how to do even better")
- How to dig out information, think critically and broadly
- How to work even harder!

Course Goals

- Goal 1: To familiarize those interested in computer system design with both fundamental operation principles and design tradeoffs of processor, memory, and platform architectures in today's systems.
  - Strong emphasis on fundamentals and design tradeoffs.

- Goal 2: To provide the necessary background and experience to design, implement, and evaluate a modern processor by performing hands-on RTL and C-level implementation.
  - Strong emphasis on functionality and hands-on design.

A Note on Hardware vs. Software

- This course is classified under "Computer Hardware"

- However, you will be much more capable if you master both hardware and software (and the interface between them)
  - Can develop better software if you understand the underlying hardware
  - Can design better hardware if you understand what software it will execute
  - Can design a better computing system if you understand both

- This course covers the HW/SW interface and microarchitecture
  - We will focus on tradeoffs and how they affect software

What Do I Expect From You?

- Required background: 240 (digital logic, RTL implementation, Verilog), 213/243 (systems, virtual memory, assembly)

- Learn the material thoroughly
  - attend lectures, do the readings, do the homeworks
- Do the work & work hard
- Ask questions, take notes, participate
- Perform the assigned readings
- **Come to class on time**
- Start early – do not procrastinate
- If you want feedback, come to office hours

- Remember "Chance favors the prepared mind." (Pasteur)

What Do I Expect From You?

- How you prepare and manage your time is very important
- There will be an assignment due almost every week
  - 7 Labs and 7 Homework Assignments
- This will be a heavy course
  - However, you will learn a lot of fascinating topics and understand how a microprocessor actually works (and how it can be made to work better)

How Will You Be Evaluated?

- Six Homeworks: 10%
- Seven Lab Assignments: 35%
- Midterm I: 15%
- Midterm II: 15%
- Final: 25%
- Our evaluation of your performance: 5%
  - Participation counts
  - Doing the readings counts

More on Homeworks and Labs

- Homeworks
  - Do them to truly understand the material, not to get the grade
  - Content from lectures, readings, labs, discussions
  - All homework writeups *must* be your own work, written up individually and independently
    - However, you can discuss with others
  - No late homeworks accepted

- Labs
  - These will take time.
  - You need to start early and work hard.
  - Labs will be done individually unless specified otherwise.
  - A total of five late lab days per semester allowed.

A Note on Cheating and Academic Dishonesty

- Absolutely no form of cheating will be tolerated
- You are all adults and we will treat you so
- See syllabus, CMU Policy, and ECE Academic Integrity Policy
  - Linked from syllabus
- Cheating → Failing grade (no exceptions)
  - And, perhaps more

Homeworks for Next Two Weeks

- Homework 0
  - Due next Wednesday (Jan 22)

- Homework 1
  - Due Wednesday Jan 29
  - ARM warmup, ISA concepts, basic performance evaluation

Lab Assignment 1

- A functional C-level simulator for a subset of the ARM ISA
- Due Friday Jan 24, at the end of the Friday recitation session

- Start early, you will have a lot to learn
- Homework 1 and Lab 1 are synergistic
  - Homework questions are meant to help you in the Lab

Readings for Next Time (Wednesday)

- Patt, "Requirements, Bottlenecks, and Good Fortune: Agents for Microprocessor Evolution," Proceedings of the IEEE 2001.
- Mutlu and Moscibroda, "Memory Performance Attacks: Denial of Memory Service in Multi-core Systems," USENIX Security Symposium 2007.

- P&P Chapter 1 (Fundamentals)
- P&H Chapters 1 and 2 (Intro, Abstractions, ISA, MIPS)

- Reference material throughout the course
  - ARM Reference Manual
  - (less so) x86 Reference Manual

A Note on Books

- None required

- But, I expect you to be resourceful in finding and doing the readings…

Recitations This and Next Week

- ARM ISA Tutorial
  - Rachata, Varun, Xiao, Paraj
  - You can attend any recitation session