18-447
Computer Architecture
Lecture 1: Introduction and Basics

Prof. Onur Mutlu
Carnegie Mellon University
Spring 2013, 1/14/2013

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

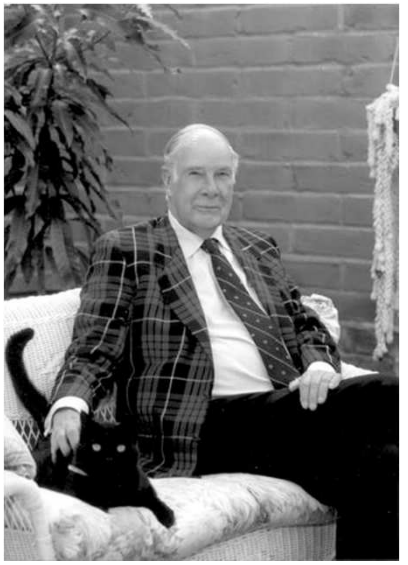
HW designer's view of a computer system works

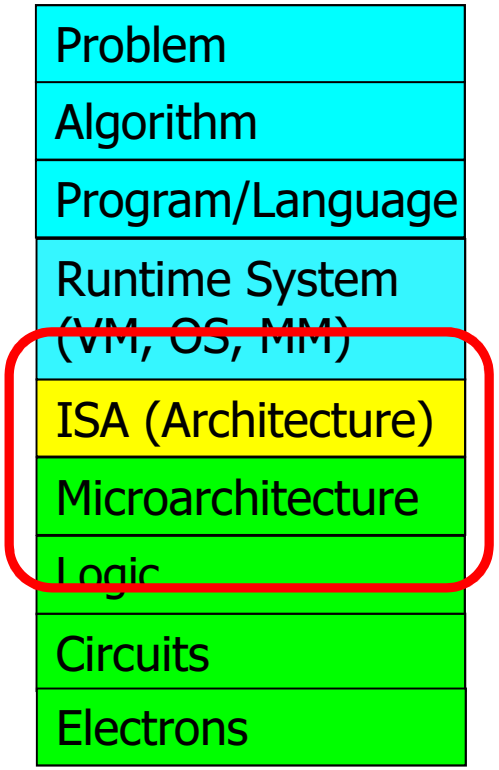
Digital logic as a model of computation

18-240

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

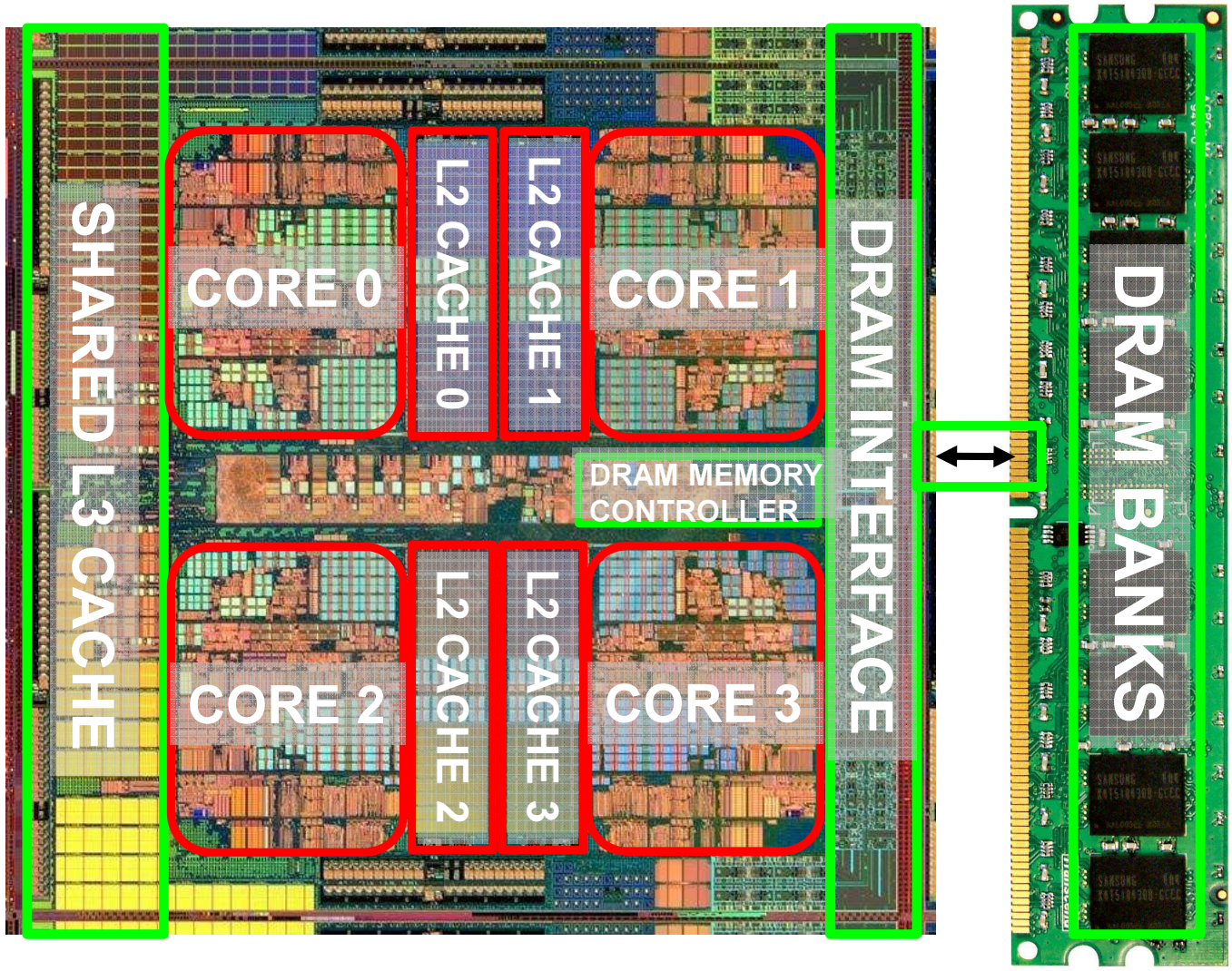
- Levels of transformation create abstractions
  - Abstraction: A higher level only needs to know about the interface to the lower level, not how the lower level is implemented
  - E.g., high-level language programmer does not really need to know what the ISA is and how a computer executes instructions
- 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?
- One goal of this course is to understand how a processor works underneath the software layer and how decisions made in hardware affect the software/programmer

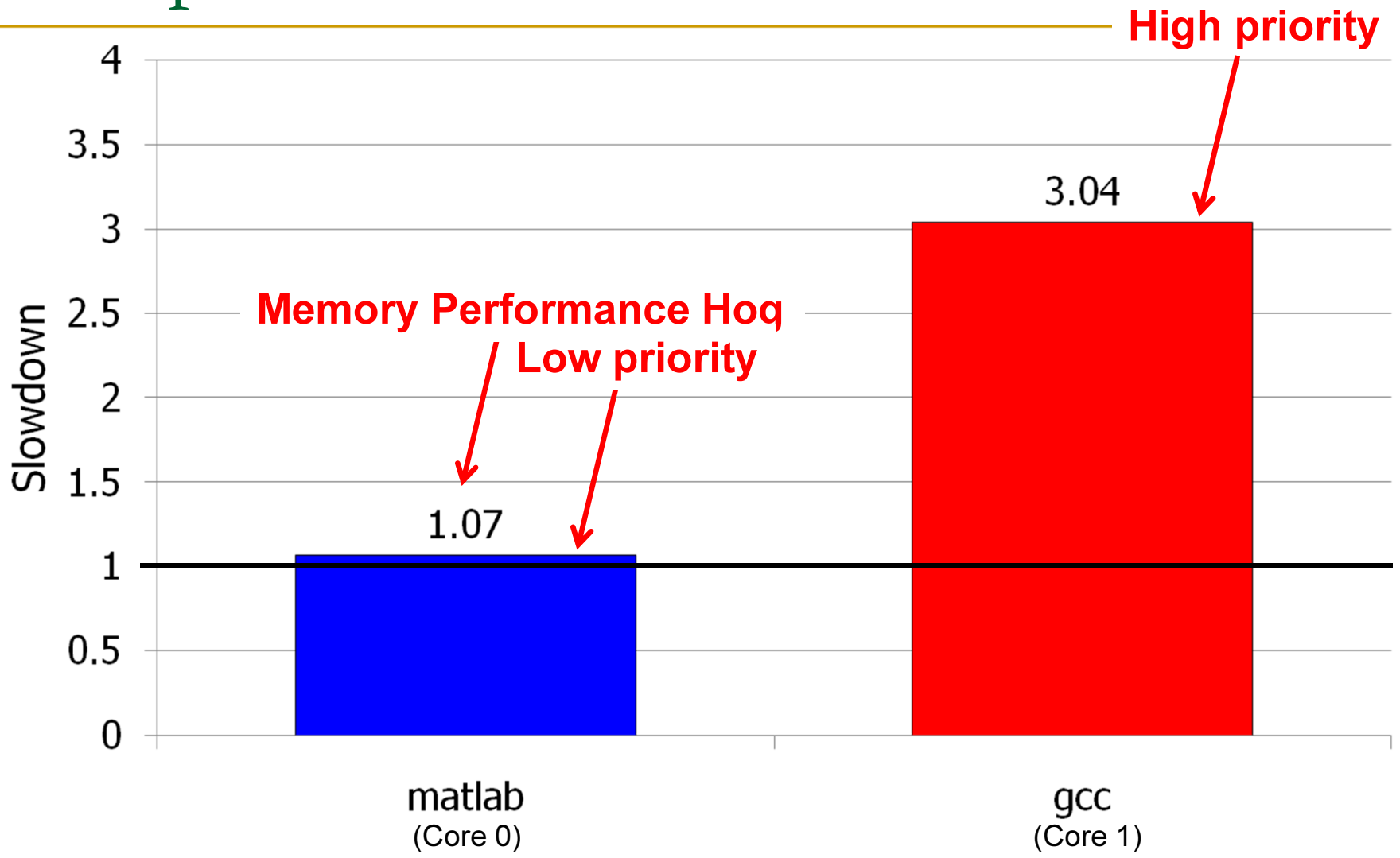
An Example: Multi-Core Systems

Multi-Core Chip



*Die photo credit: AMD Barcelona

Unexpected Slowdowns in Multi-Core



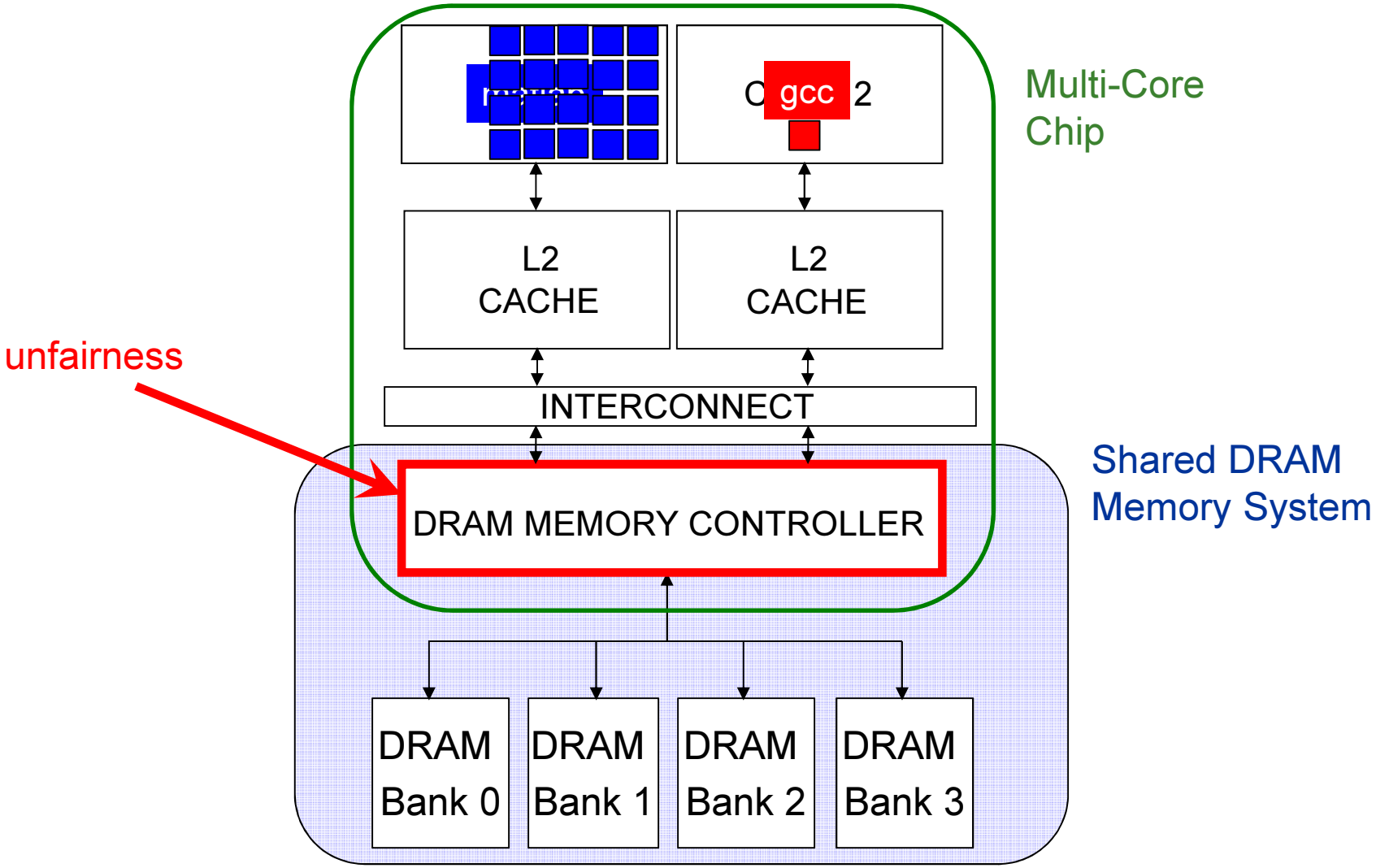
Moscibroda and Mutlu, "Memory performance attacks: Denial of memory service in multi-core systems," USENIX Security 2007.

A Question or Two

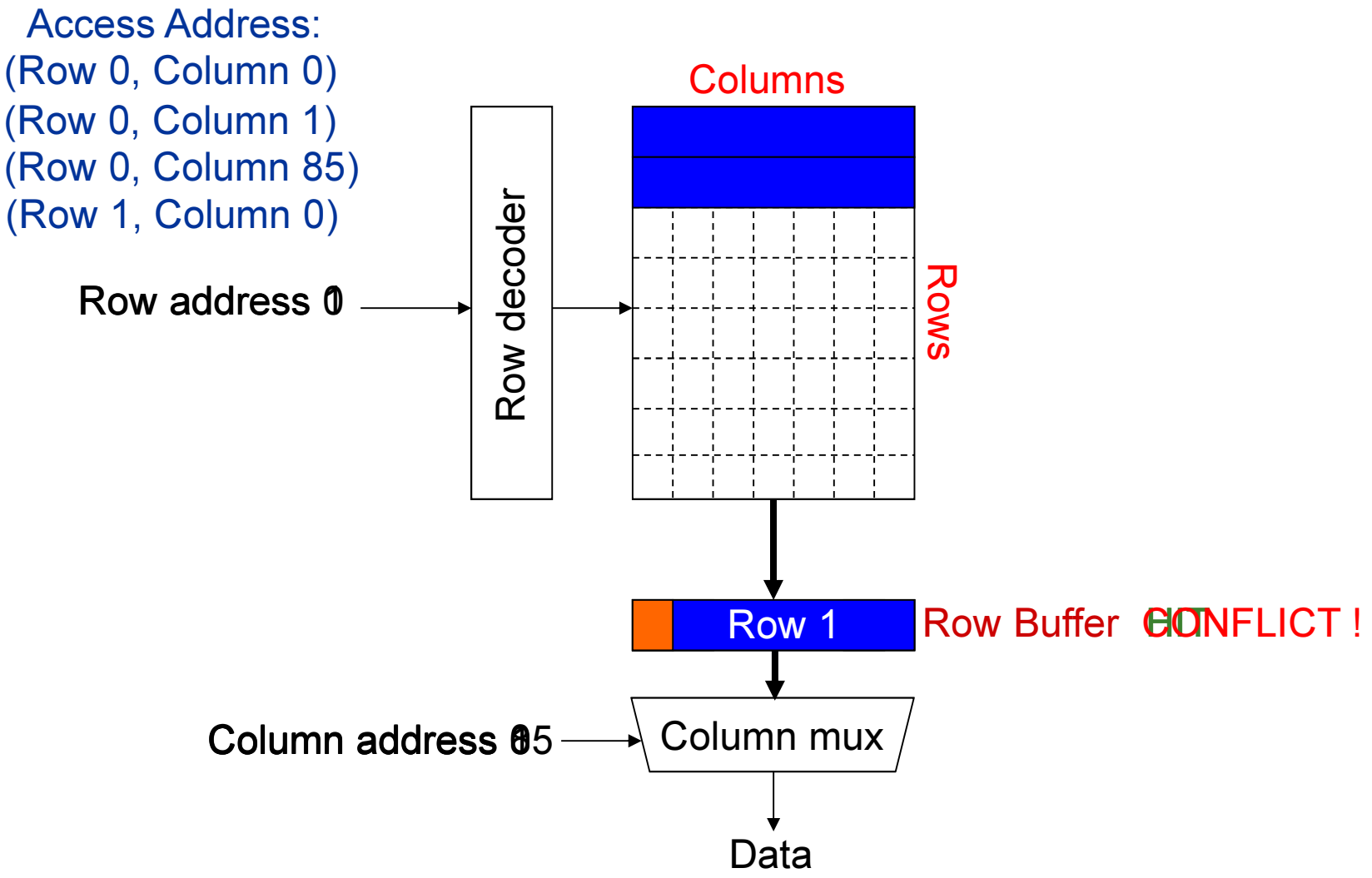
- Can you figure out why there is a disparity in slowdowns if you do not know how the processor executes the programs?

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

Why the Disparity in Slowdowns?



DRAM Bank Operation



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

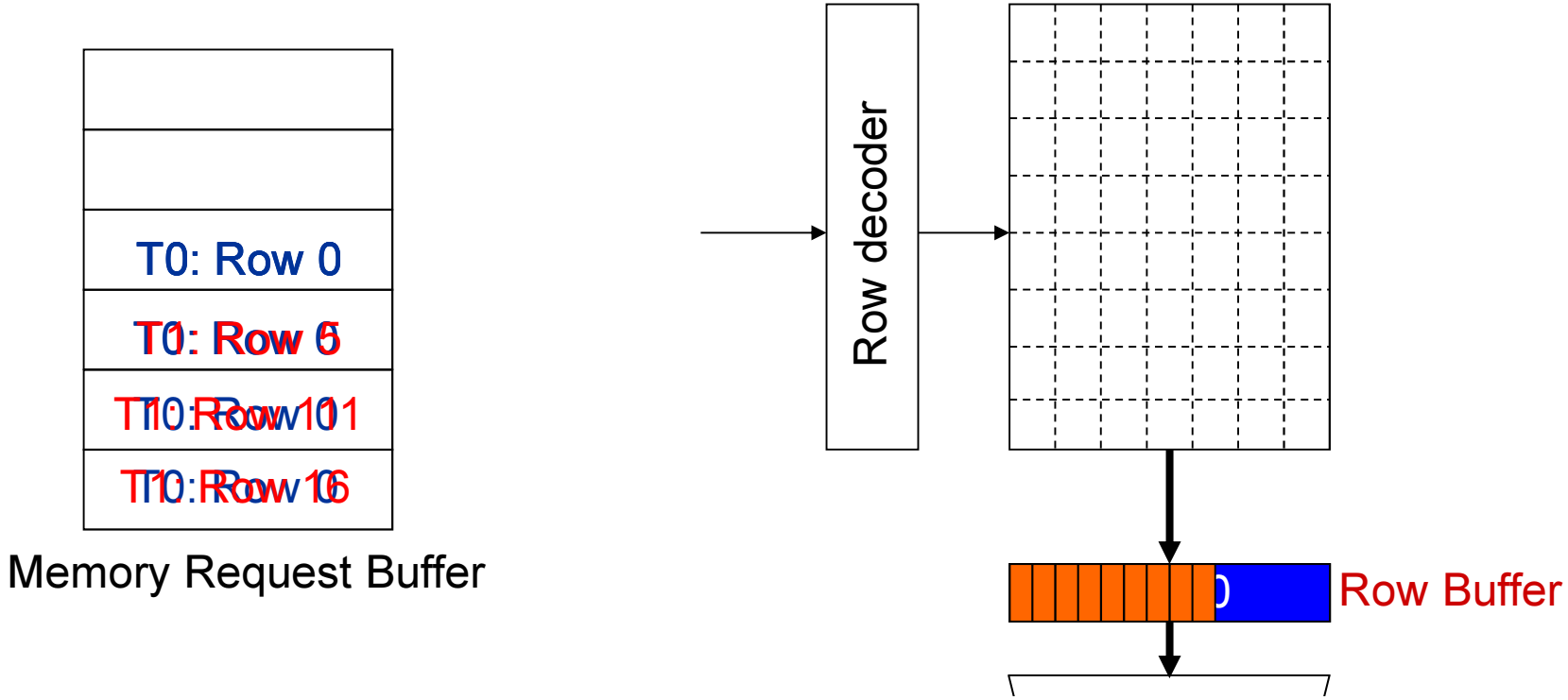
- Sequential memory access
- Very high row buffer locality (96% hit rate)
- Memory intensive

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?



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

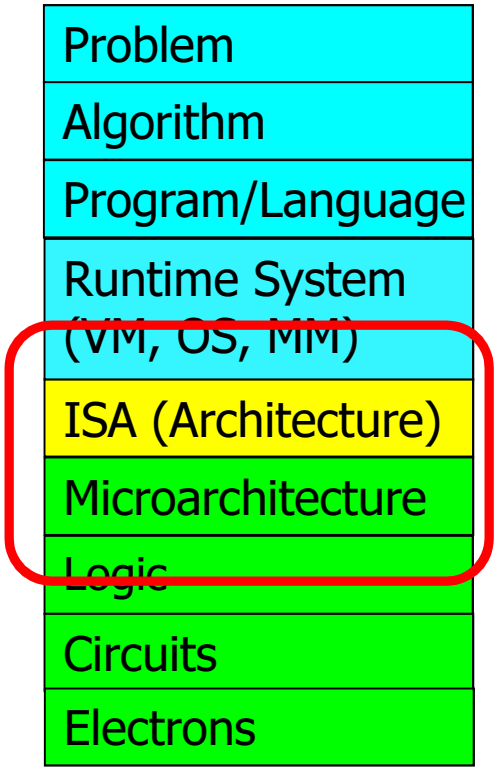
128 (8KB/64B) requests of T0 serviced before T1

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:
  - Make you think critically
  - Make you think broadly

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 23)
  - Homework 1 (due Jan 28)
  - Lab 1 (due Feb 1)

- Basic concepts in computer architecture

Handouts

- Make sure you get a copy of
  - Homework 0
  - Syllabus (online)

Course Info: Who Are We?

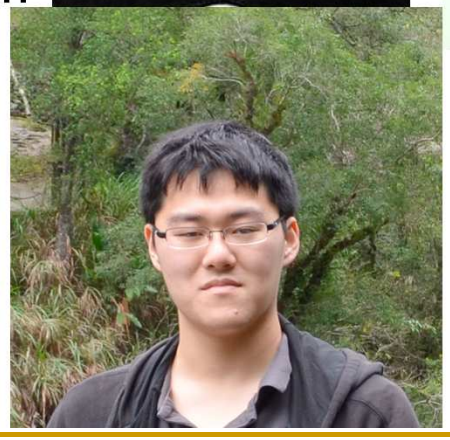
- Instructor: Prof. Onur Mutlu
  - onur@cmu.edu
  - Office: Hamerschlag Hall A305 (and CIC 4105)
  - Office Hours: W 2:30-3:30pm (or by appointment)
  - http://www.ece.cmu.edu/~omutlu
  - PhD from UT-Austin, worked at Microsoft Research, Intel, AMD
  - Research and teaching interests:
    - Computer 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

  - Yoongu Kim
    - yoongukim@cmu.edu
    - Office hours: Fri 4-6pm

  - Justin Meza
    - meza@cmu.edu
    - Office hours: Thu 11am-1pm



  - Jason Lin
    - jasonli1@andrew.cmu.edu
    - Office hours: TBD

Your Turn

- Who are you?

- Homework 0
  - Your opportunity to tell us about yourself
  - Due Jan 23, before class
  - Return hard copy
  - Attach your picture

- 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
- Blackboard
  - Upload homeworks here except for HW0
- Your email
- Me and the TAs

Lecture and Lab Locations, Times

- Lectures:
  - MWF 12:30-2:20pm
  - Hamerschlag Hall B103
  - Attendance will not be enforced but is for your benefit to learn
  - Some days, we will have recitation sessions

- Labs:
  - T 10:30am-1:20pm
  - Th 1:30-4:20pm
  - F 6:30-9:20pm
  - Hamerschlag Hall 1303
  - Attendance not required except for check-off of labs
  - You can attend any lab session to ask questions or check off

Tentative Course Schedule

- Tentative schedule is in syllabus
- To get an idea of topics, you can look at last year's schedule:
  - http://www.ece.cmu.edu/~ece447/s12/doku.php

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

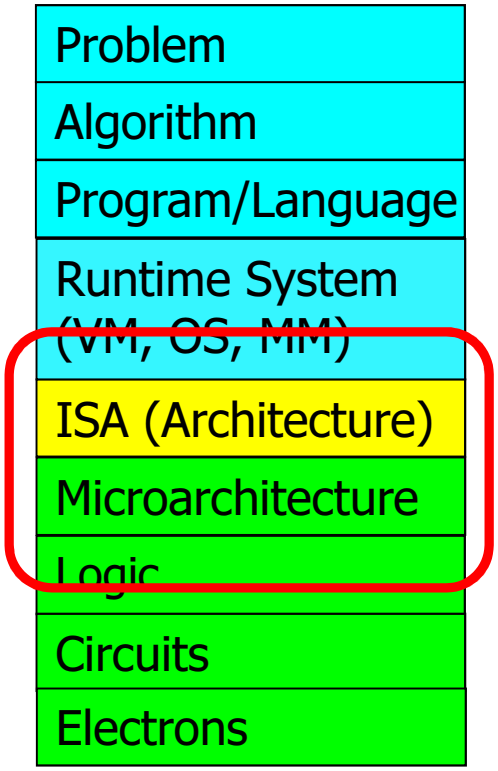
What Will You Learn

- **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*, IE 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).

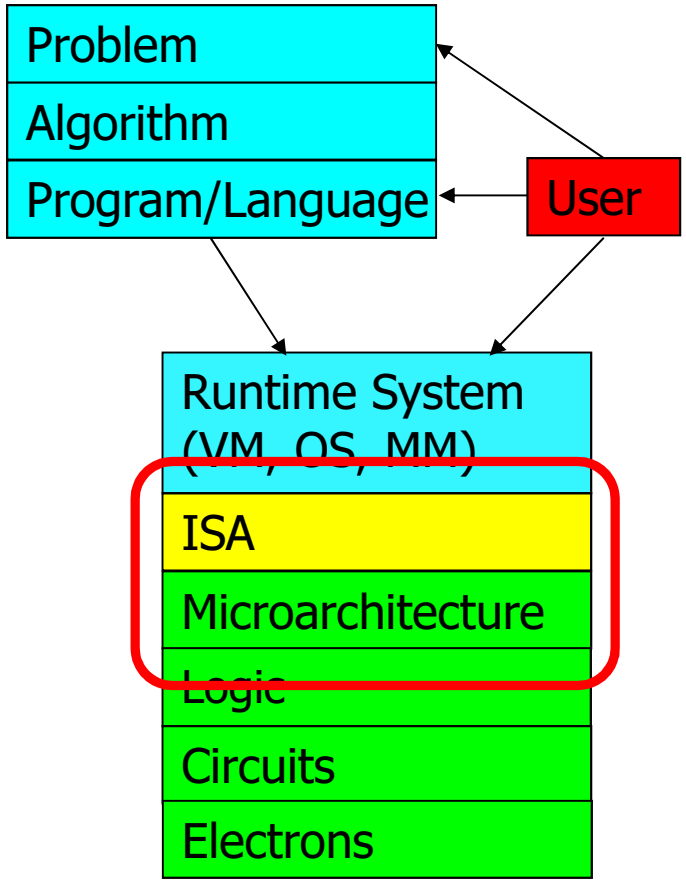
Computer Architecture in Levels of Transformation



- 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.
  - Assigned lab slots are for check-off only.
  - 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 Monday (Jan 23), right before lecture

- Homework 1
  - Due Monday Jan 28, right before lecture, on Blackboard
  - MIPS warmup, ISA concepts, basic performance evaluation

Lab Assignment 1

- A functional C-level simulator for a subset of the MIPS ISA
- Due Friday Feb 1, at the end of Friday lab

- 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
  - MIPS R4000 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…

Discussion/Recitation for Next Time

- MIPS ISA Tutorial
  - Justin, Yoongu, and Jason
  - Time(s) to be determined

We did not cover the following slides in lecture.
These are for your preparation for the next lecture.

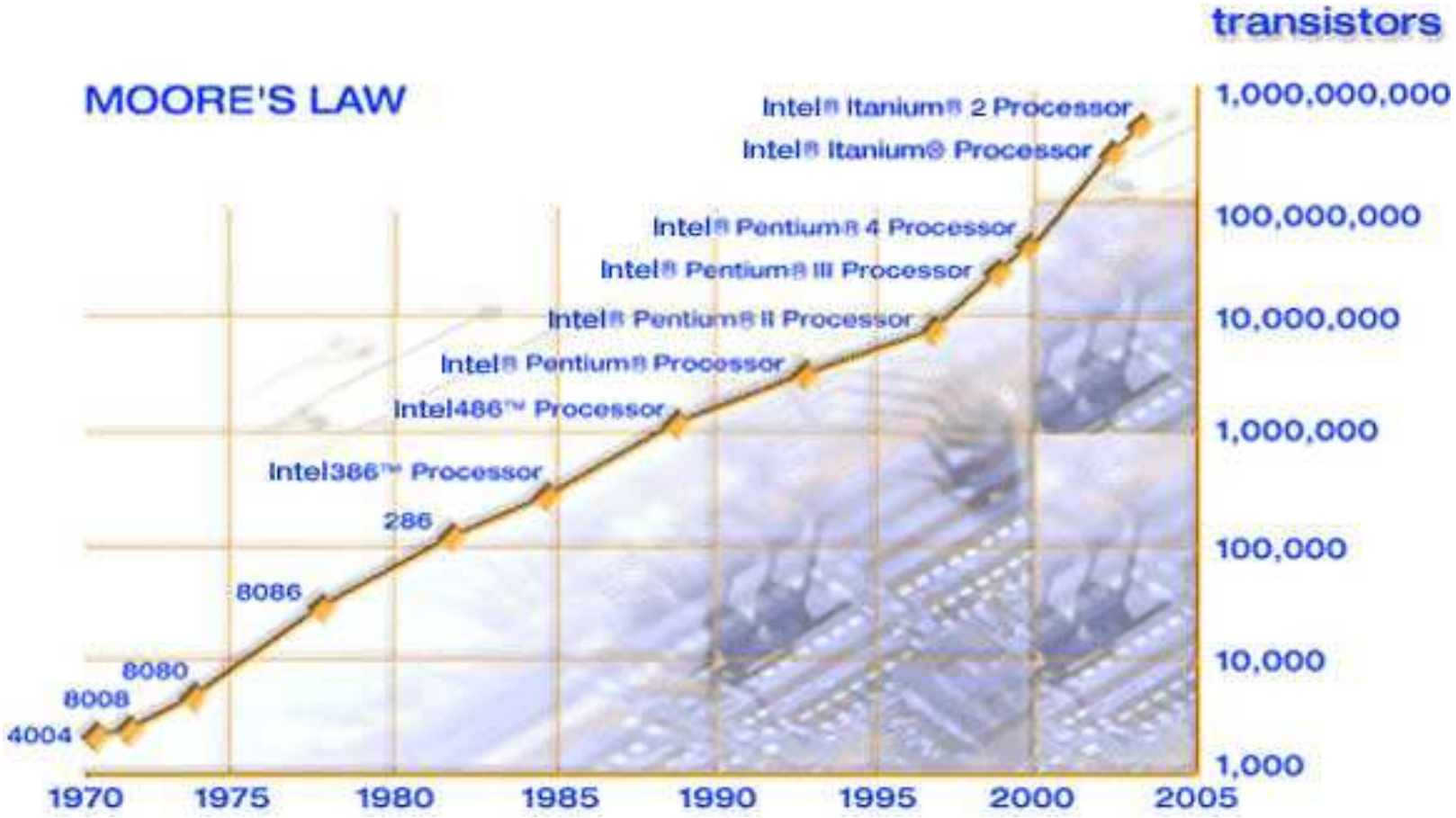
Why Study Computer Architecture?

What is 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.

- We will soon distinguish between the terms *architecture*, and *microarchitecture*.

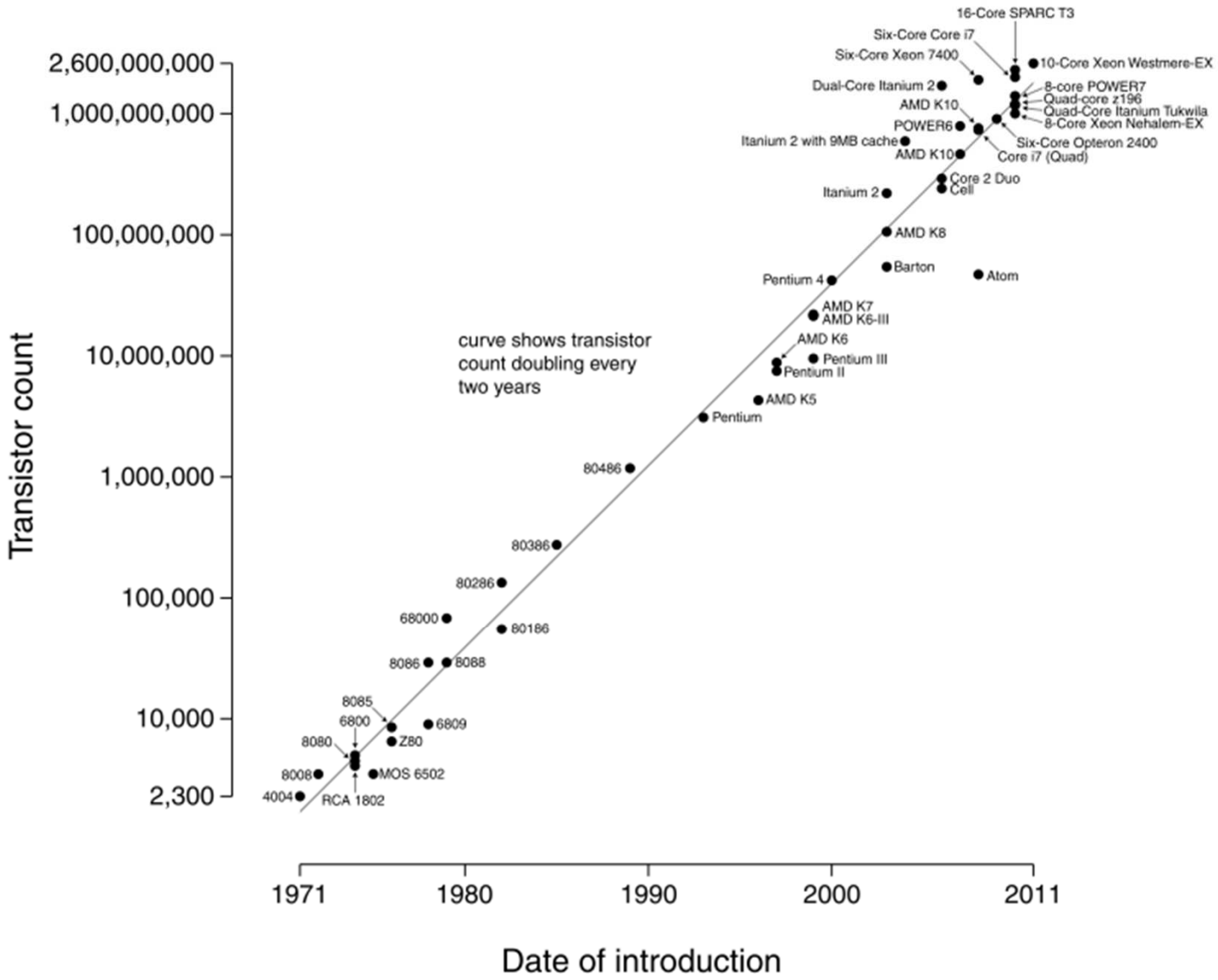
Moore's Law



Moore, "Cramming more components onto integrated circuits," Electronics Magazine, 1965. Component counts double every year

Image source: Intel

Microprocessor Transistor Counts 1971-2011 & Moore's Law



Number of transistors on an integrated circuit doubles ~ every two years

Image source: Wikipedia

What Do We Use These Transistors for?

Why Study Computer Architecture?

- Enable better systems: make computers faster, cheaper, smaller, more reliable, …
  - By exploiting advances and changes in underlying technology/circuits
- Enable new applications
  - Life-like 3D visualization 20 years ago?
  - Virtual reality?
  - Personal genomics?
- Enable better solutions to problems
  - Software innovation is built into trends and changes in computer architecture
    - > 50% performance improvement per year has enabled
- Understand why computers work the way they do

Computer Architecture Today

- Today is a very exciting time to study computer architecture
- Industry is in a large paradigm shift (to multi-core)
- Many problems motivating and caused by the shift
  - Power/energy constraints
  - Complexity of design → multi-core
  - Technology scaling → new technologies
  - Memory wall/gap
  - Reliability wall/issues
  - Programmability wall/problem

- You can revolutionize the way computers are built, if you understand both the hardware and the software (and change each accordingly)
  - Book: Kuhn, "The Structure of Scientific Revolutions" (1962)