Computer Architecture
Lecture 1: Introduction and Basics

Dr. Eman Gawish
Nile University
Fall 2020

OUR VISION

- The vision of the School is to be a world-class school, recognized as one of the top in the region in research, education and entrepreneurship.

OUR MISSION

The mission of the school is to contribute to the development of cultural values and to information technology-driven economies in the region through the pursuit of education, research, innovation and entrepreneurship at the highest levels of excellence by:

- Offering leading edge undergraduate and graduate programs and executive education.
- Carrying-out interdisciplinary research.
- Collaborating with distinguished international universities and research institutions.
- Committing to a strong linkage with business, Government, NGOs and industry, and dedication to serve the community.

CSCI 311 – Computer Architecture

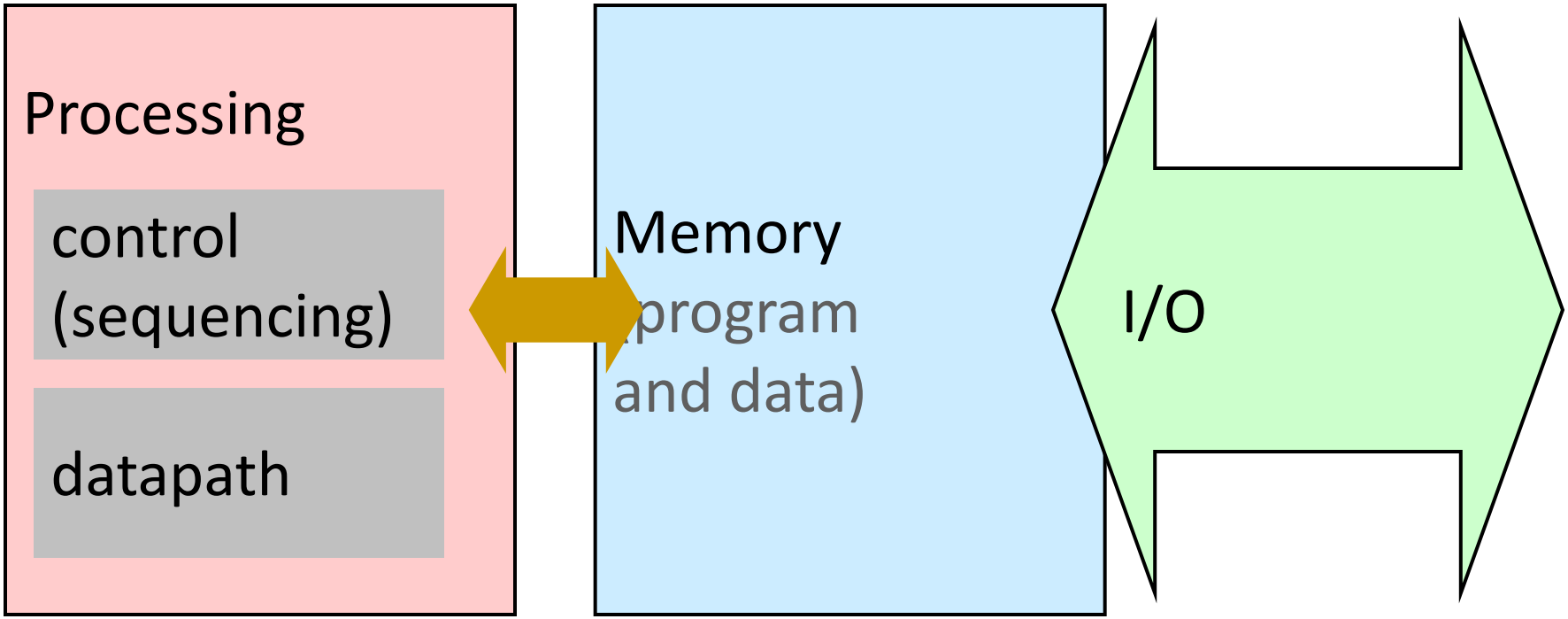
- Course Evaluation
  - Attendance and participation (5%)
  - To attend more than 80% of classes is mandatory to take the final exam
  - Lab exercises (12%)
  - Lab exercise integration (8%)
  - 4 Quizzes (15%)
  - 4 Assignments/Mini-projects (15%)
  - Mid-term exam (10%)
  - Final exam (15%)
  - Course project (15%)
  - Papers review (5 %)

CSCI 311 Computer Architecture

- Instructor
  - Dr Eman Gawish
    - Email: EGawish@nu.edu.eg
- Teaching Assistant:
  - Amira Tarek Email: Amtarek@nu.edu.eg
- Text Book:
  - "Computer Organization and Design: The Hardware/Software Interface", Fourth Edition by Patterson and Hennessy, Morgan Kaufmann/Elsvier.
  - Introduction to Computing Systems: From Bits and Gates to C and Beyond, Second Edition by Patt and Patel, McGraw-Hill
- What is written on board is binding.

What is A Computer?

- We will cover all three components



The Von Neumann Model/Architecture

- Also called *stored program computer* (instructions in memory). Two key properties:

- Stored program
  - Instructions stored in a linear memory array
  - Memory is unified between instructions and data
    - The interpretation of a stored value depends on the control signals

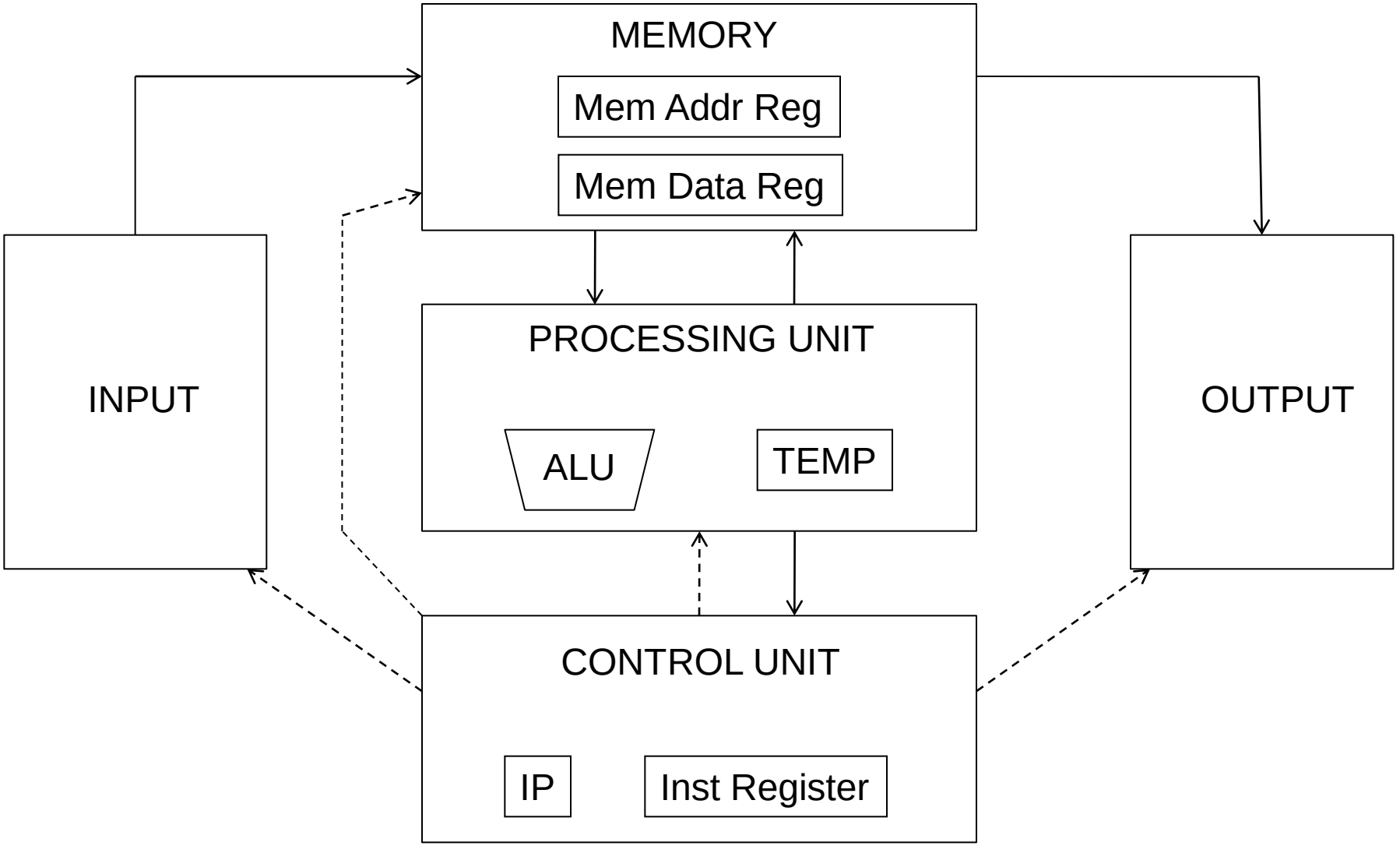
When is a value interpreted as an instruction?

- Sequential instruction processing
  - One instruction processed (fetched, executed, and completed) at a time
  - Program counter (instruction pointer) identifies the current instr.
  - Program counter is advanced sequentially except for control transfer instructions

The Von Neumann Model/Architecture

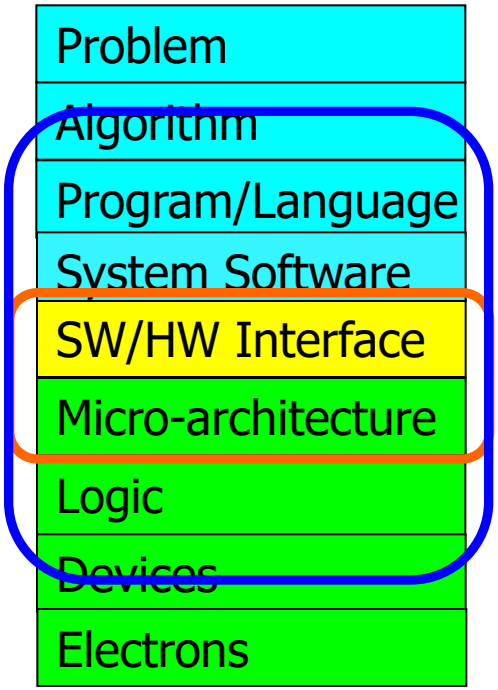
- Recommended reading
  - Burks, Goldstein, von Neumann, "Preliminary discussion of the logical design of an electronic computing instrument," 1946.
  - Patt and Patel book, Chapter 4, "The von Neumann Model"
- Stored program
- Sequential instruction processing

The Von Neumann Model (of a Computer)



The Transformation Hierarchy

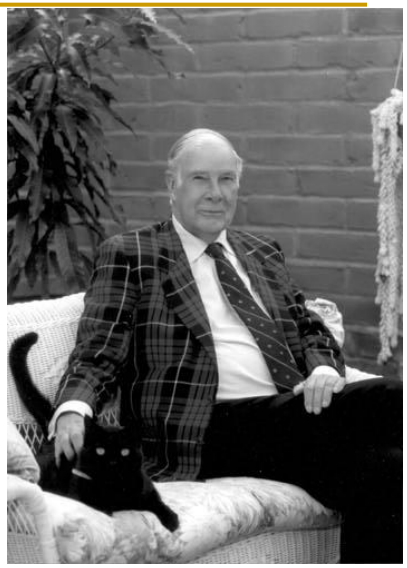
Computer Architecture (expanded view)



Computer Architecture (narrow view)

Levels of Transformation

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

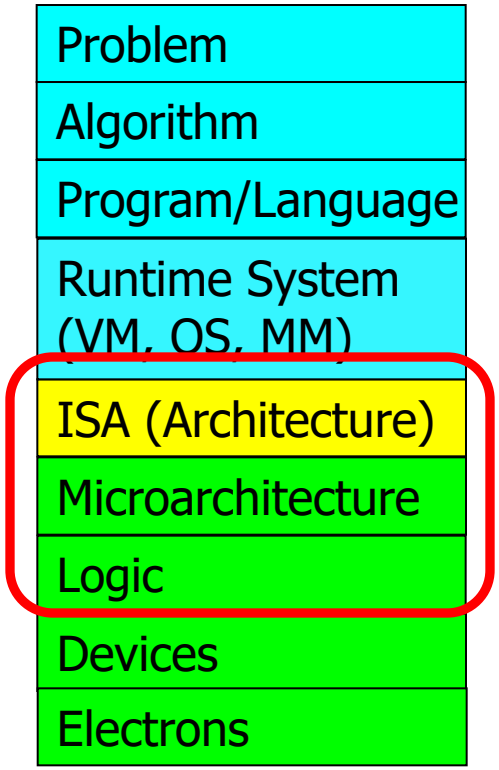


Algorithm

Step-by-step procedure that is **guaranteed to terminate** where **each step is precisely stated** and **can be carried out by a computer**

- **Finiteness**
- **Definiteness**
- **Effective computability**

Many algorithms for the same problem



ISA
(Instruction Set Architecture)

Interface/contract between SW and HW.

What the programmer assumes hardware will satisfy.

Microarchitecture
An implementation of the ISA

Digital logic circuits
Building blocks of micro-arch (e.g., gates)

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

A Note on Hardware vs. Software

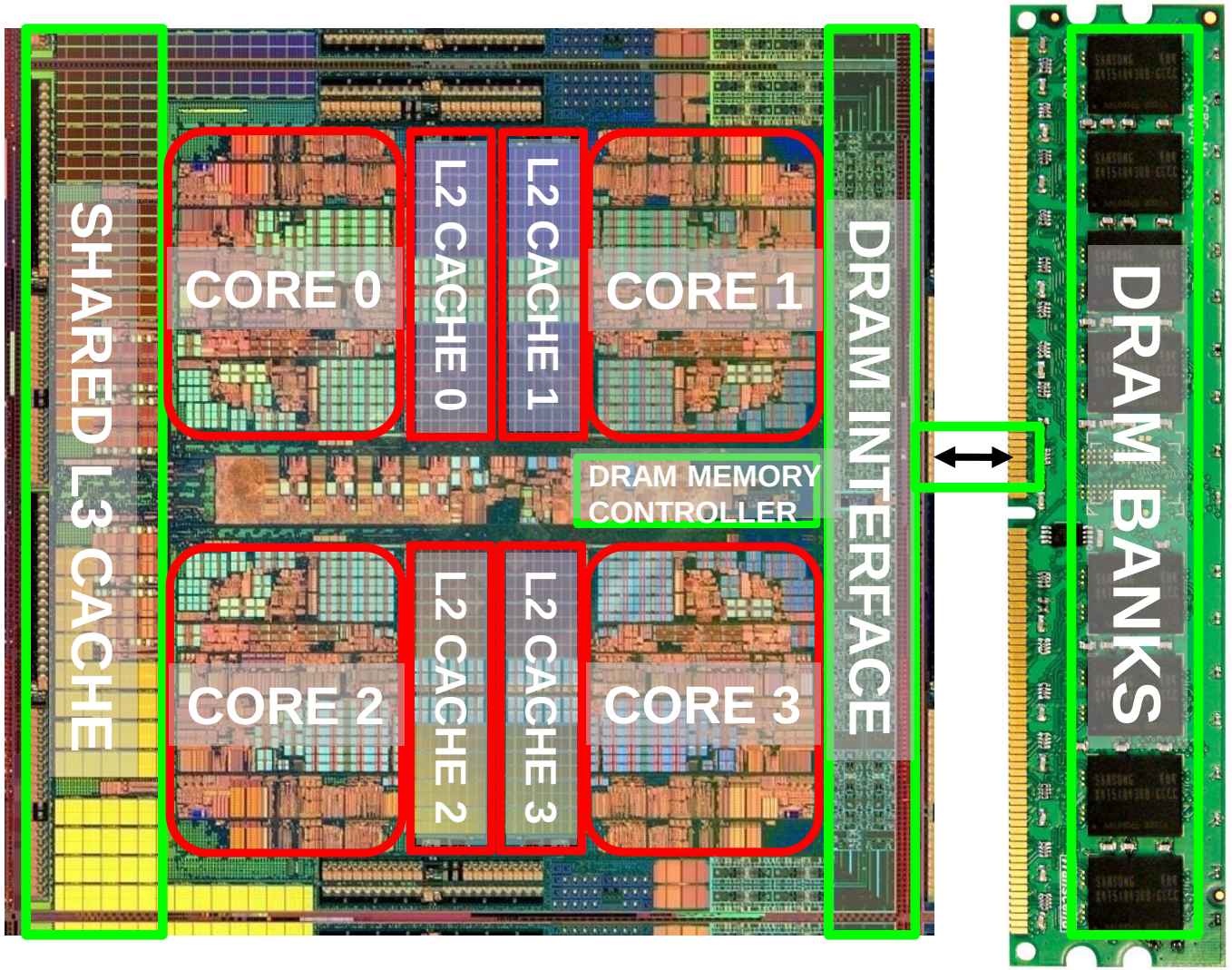
- This course is classified under "Computer Hardware"

- 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

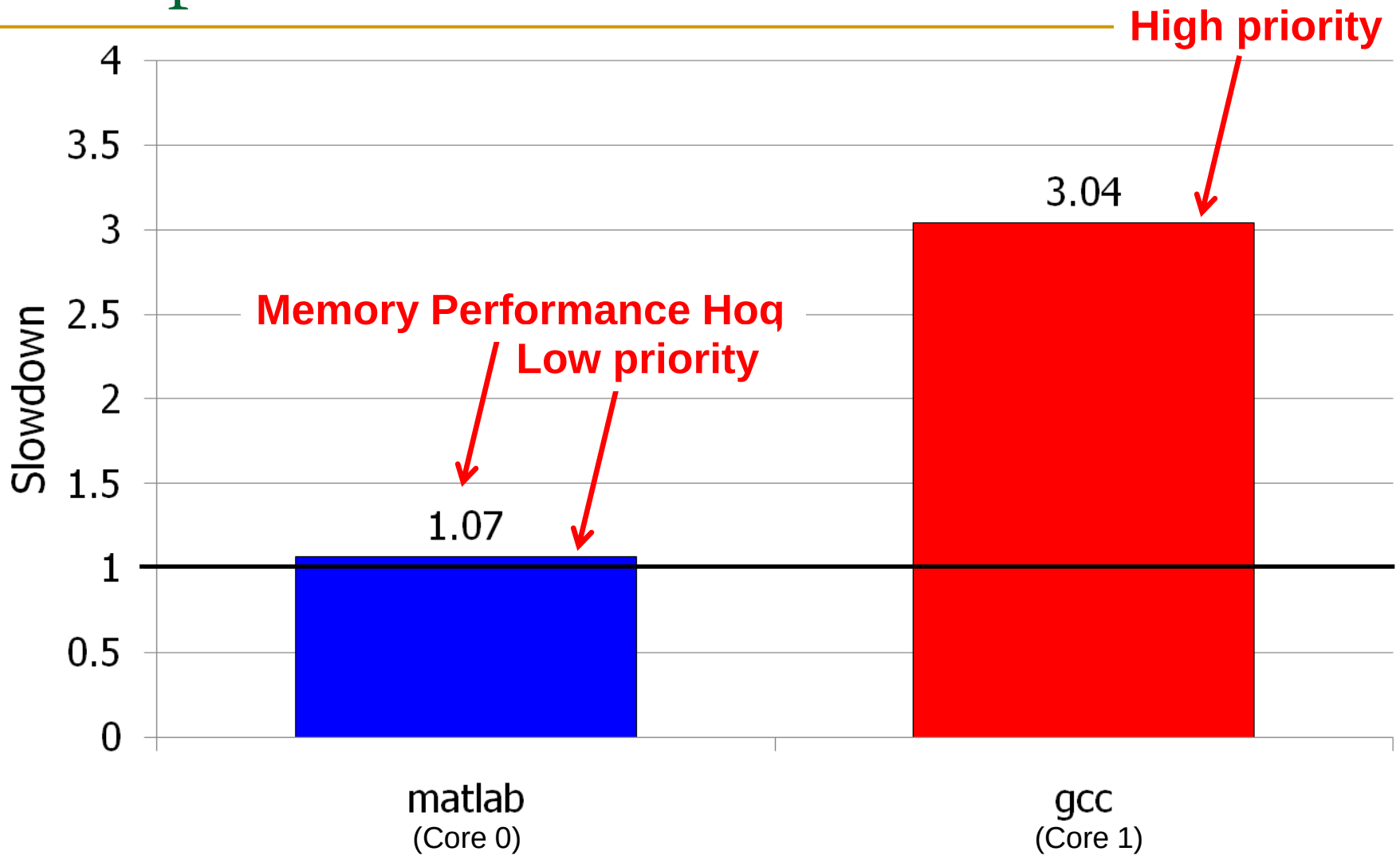
An Example: Multi-Core Systems

Multi-Core Chip



*Die photo credit: AMD Barcelona

Unexpected Slowdowns in Multi-Core



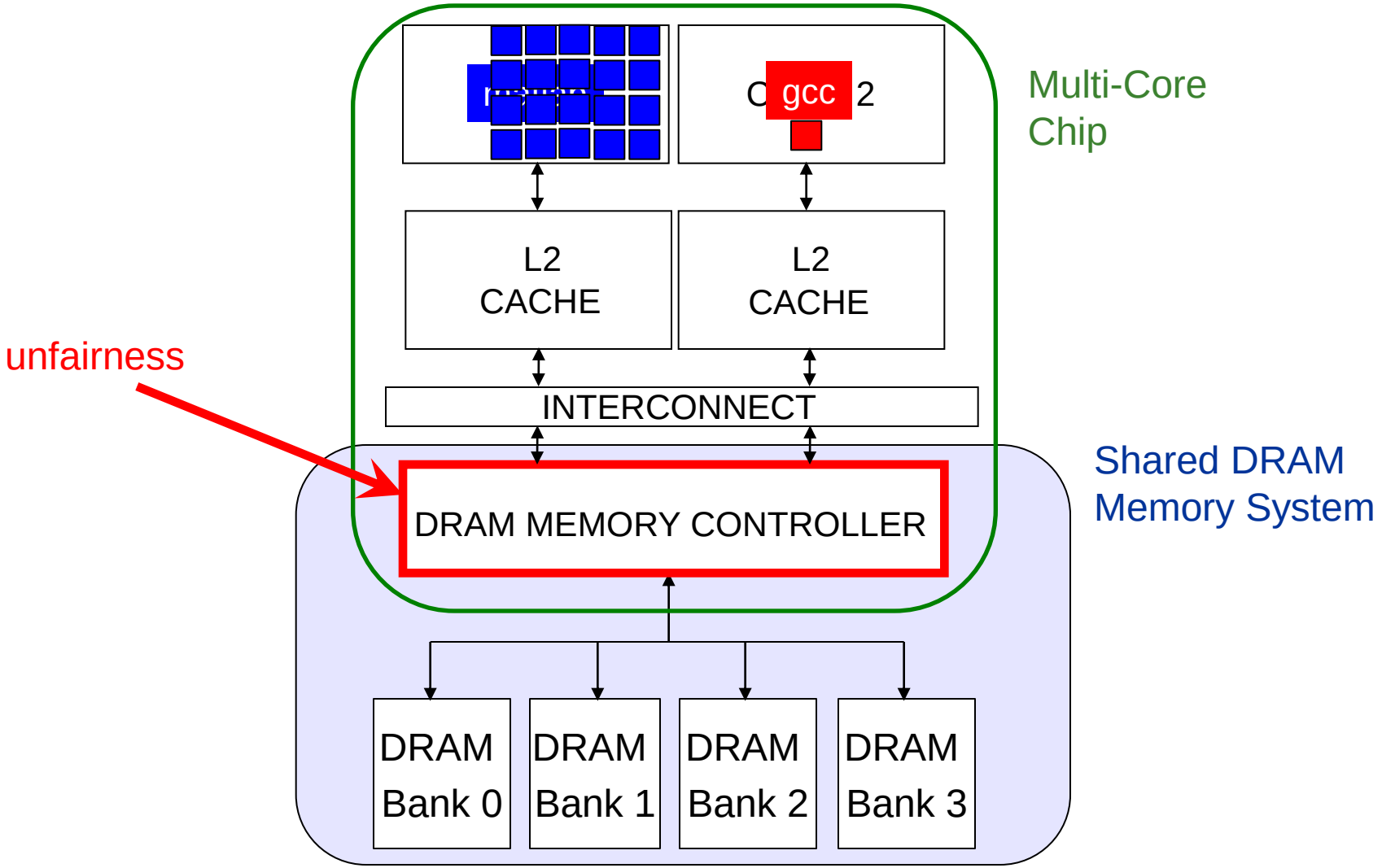
Moscibroda and Mutlu, "Memory performance attacks: Denial of memory service in multi-core systems," USENIX Security 2007.

A Question or Two

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

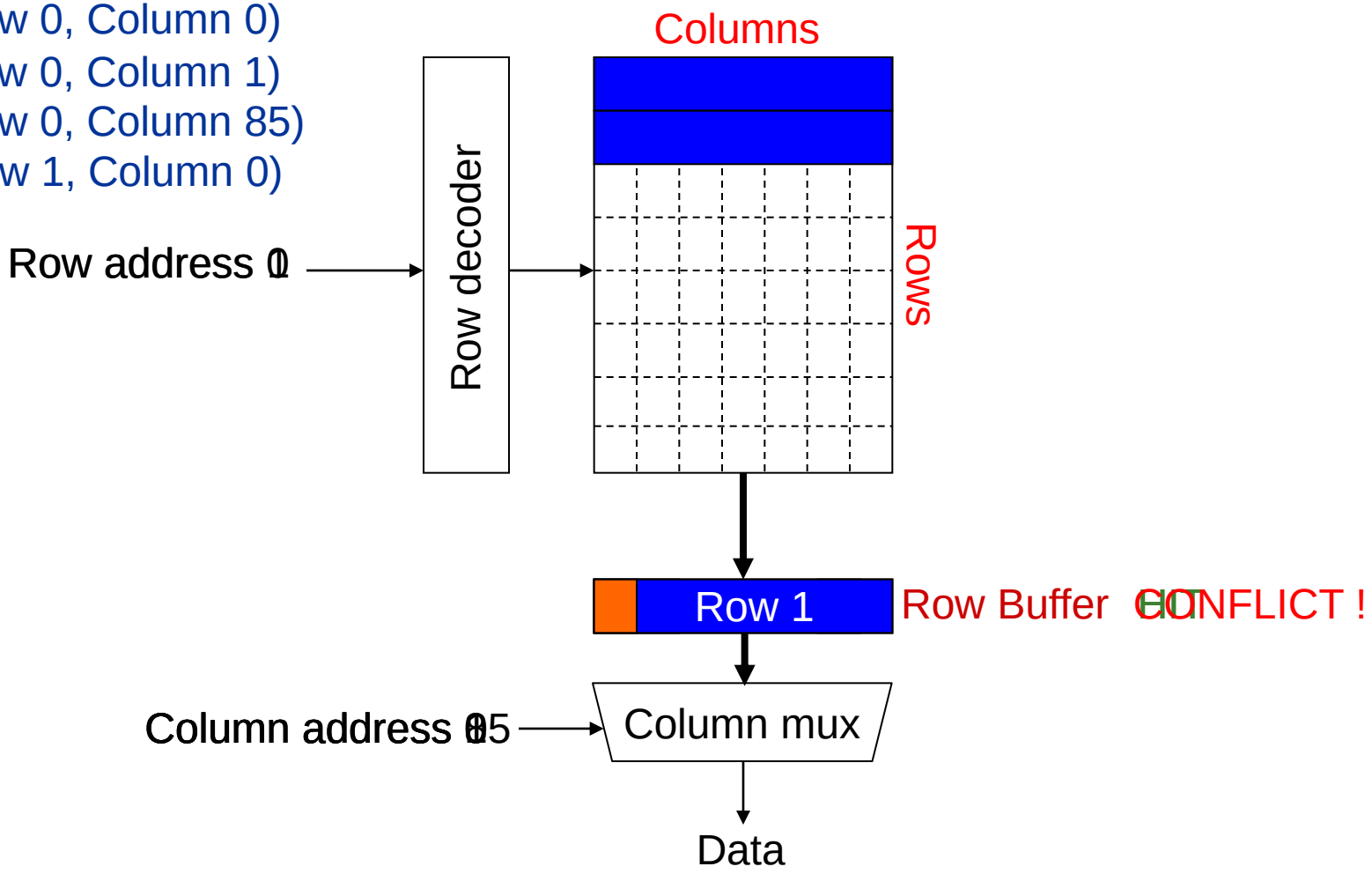
- Can you fix the problem without knowing what is happening “underneath”?

Why the Disparity in Slowdowns?

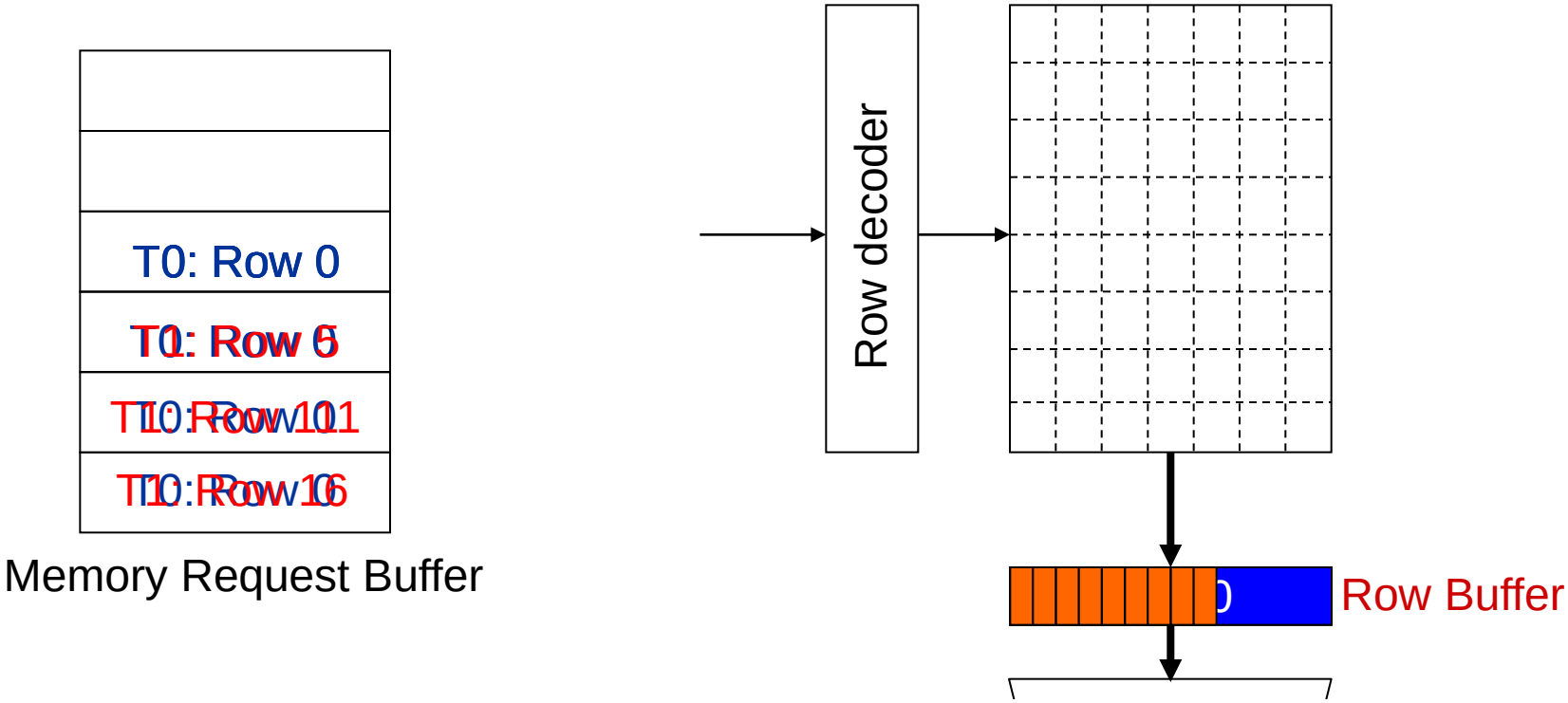


DRAM Bank Operation

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



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.

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 applications share the DRAM controller
- DRAM controllers designed to maximize DRAM data throughput

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

- 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;    streaming
    A[index] = B[index];
    …
}
```

```
// initialize large arrays A, B

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

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

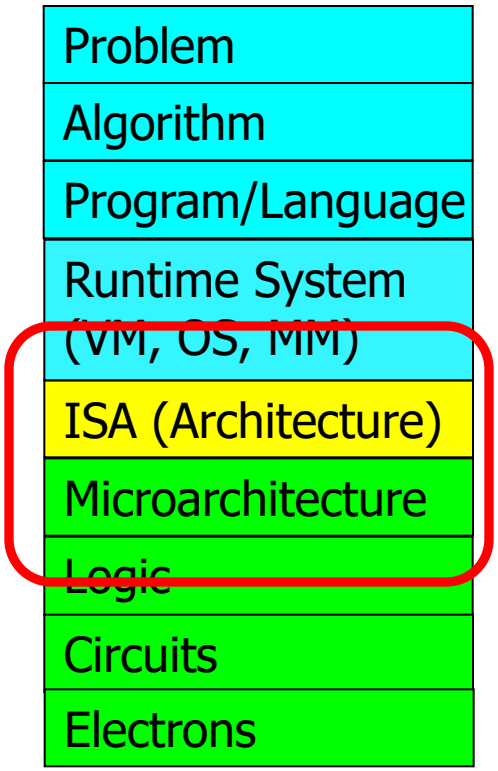
Moscibroda and Mutlu, "Memory Performance Attacks," USENIX Security 2007.

Now That We Know What Happens Underneath

- How would you solve the problem?

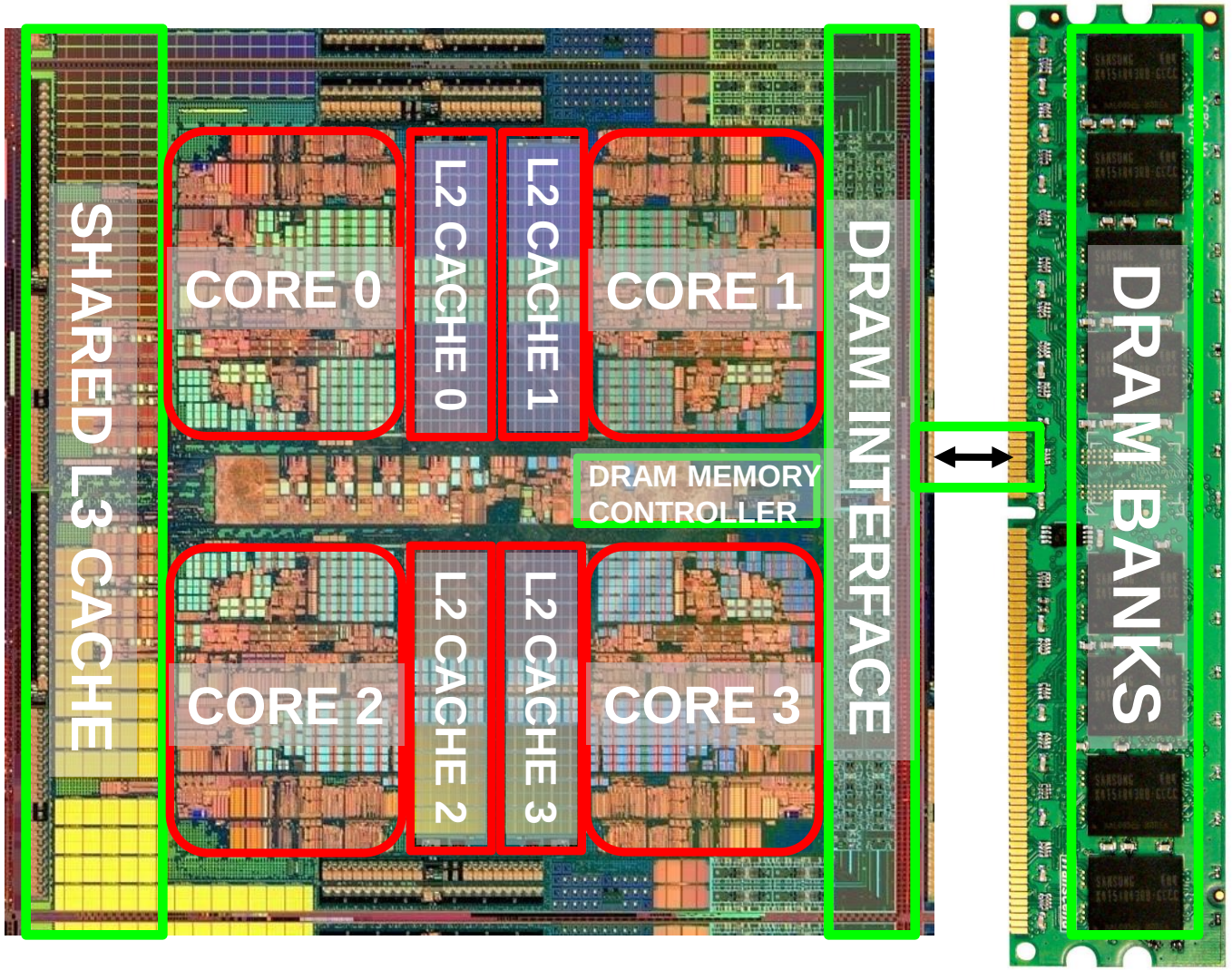
- What is the right place to solve the problem?
  - Programmer?
  - System software?
  - Compiler?
  - Hardware (Memory controller)?
  - Hardware (DRAM)?
  - Circuits?

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



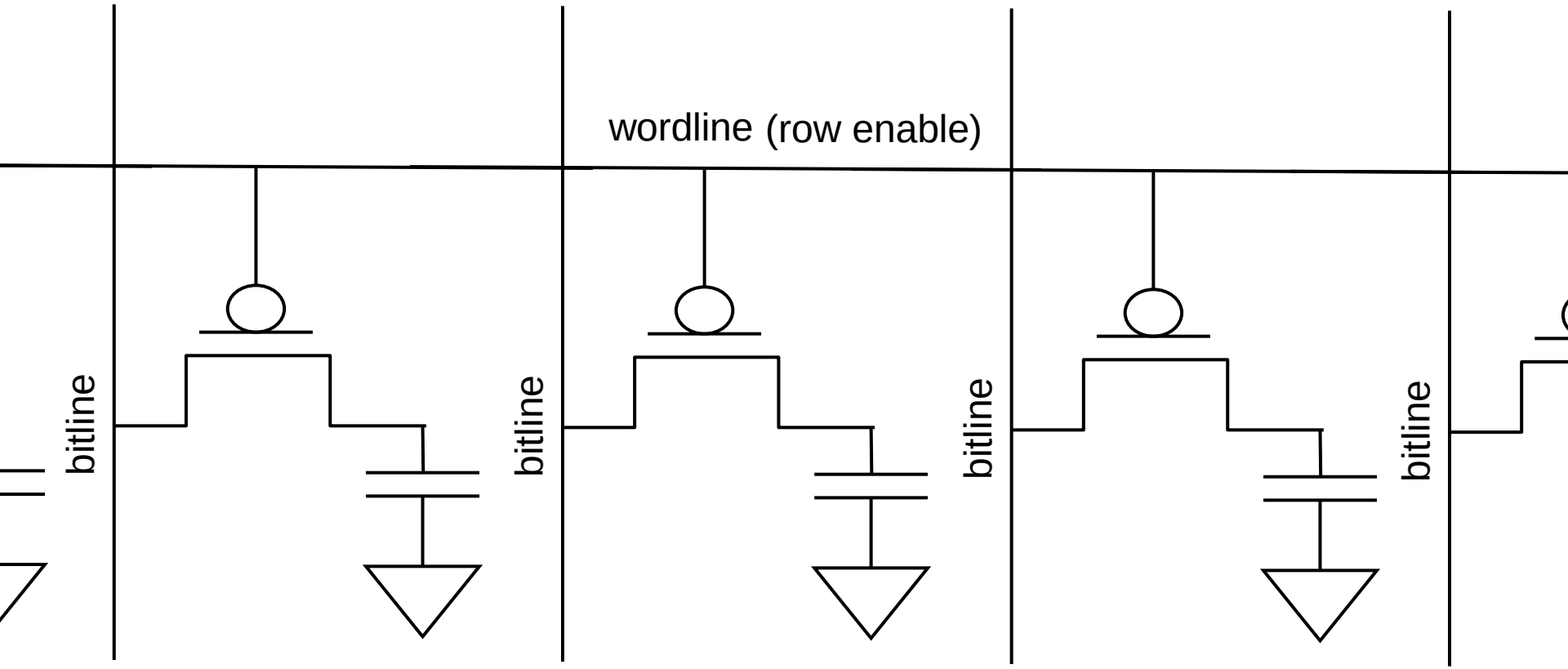
DRAM in the System

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

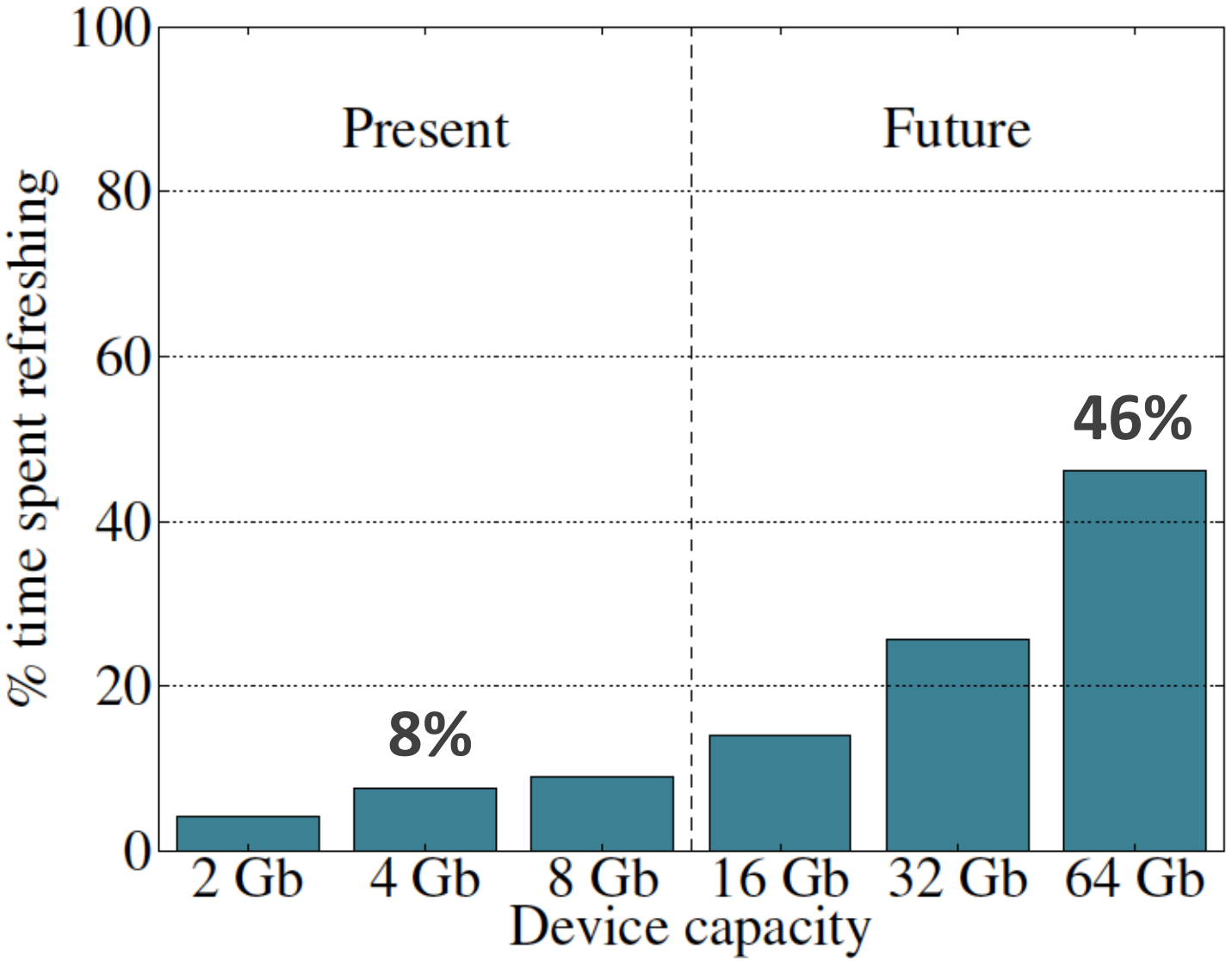
  -- Energy consumption: Each refresh consumes energy

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

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

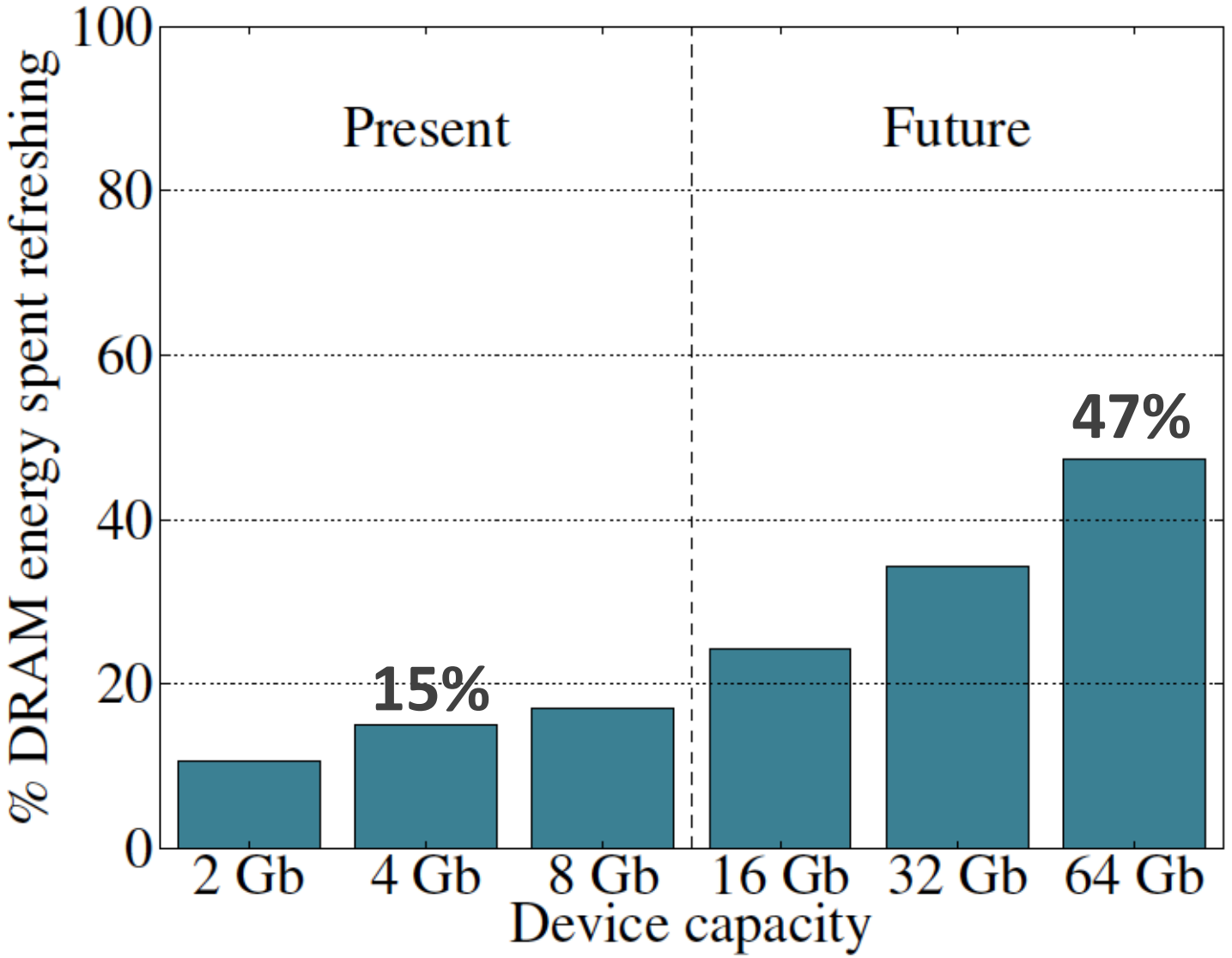
  -- Refresh rate limits DRAM capacity scaling

Refresh Overhead: Performance



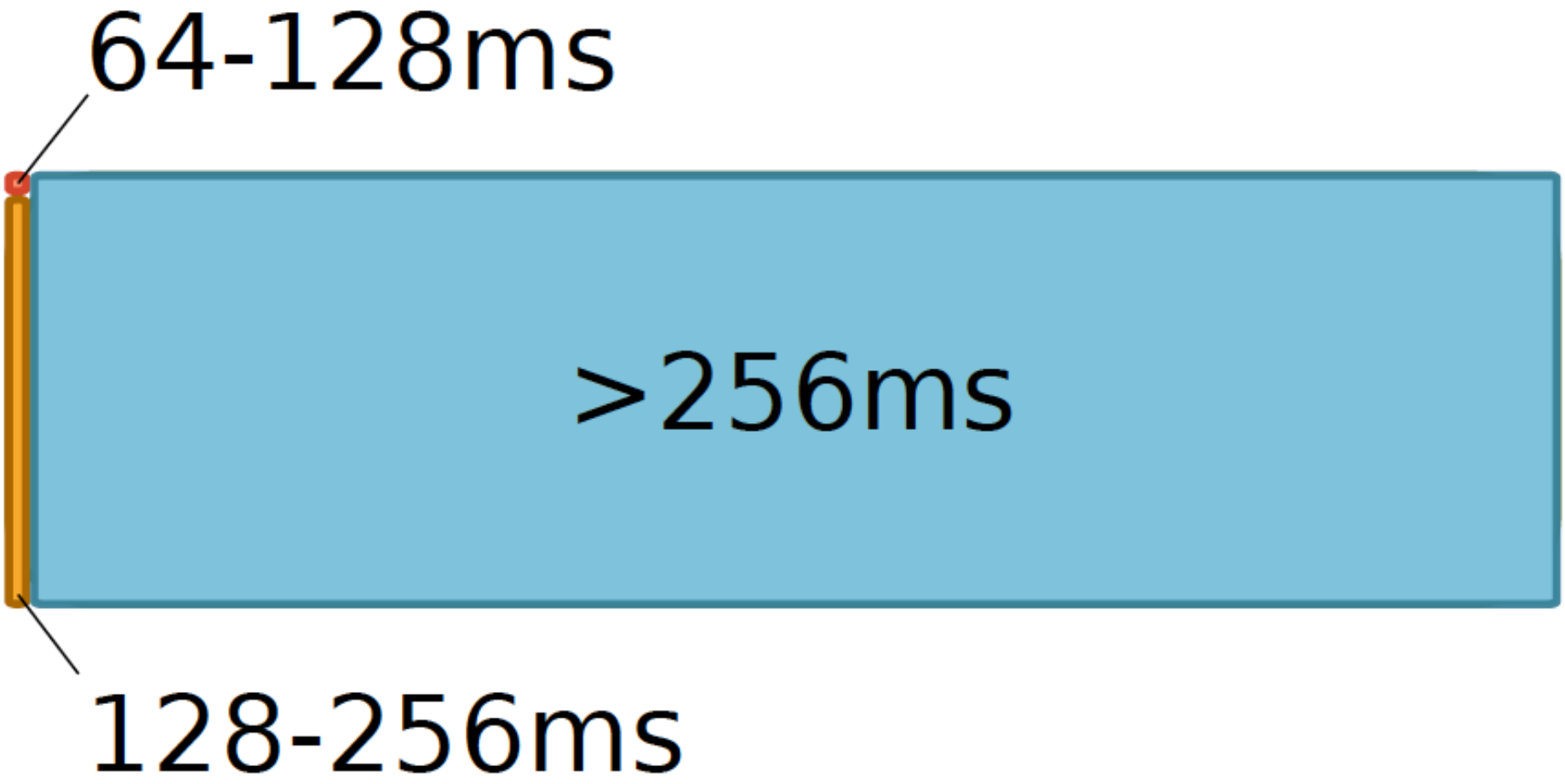
Liu et al., "RAIDR: Retention-Aware Intelligent DRAM Refresh," ISCA 2012.

Refresh Overhead: Energy



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

Underneath: Retention Time Profile of DRAM



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

Taking Advantage of This Profile

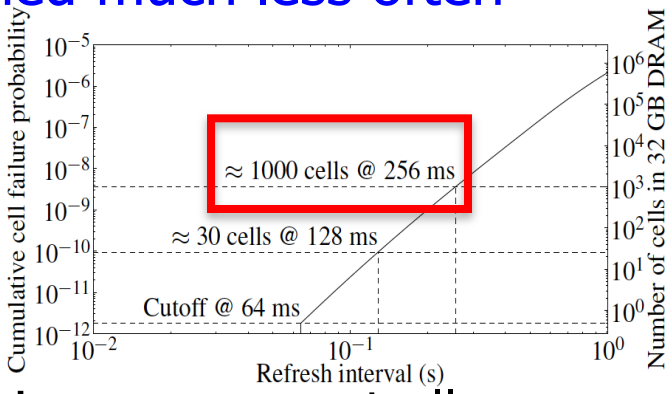
- Expose this retention time profile information to
  - the memory controller
  - the operating system
  - the programmer?
  - the compiler?

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

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

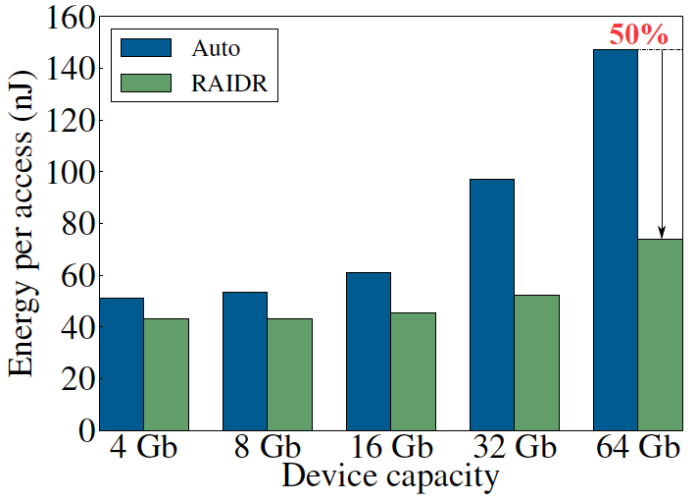
An Example: RAIDR

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



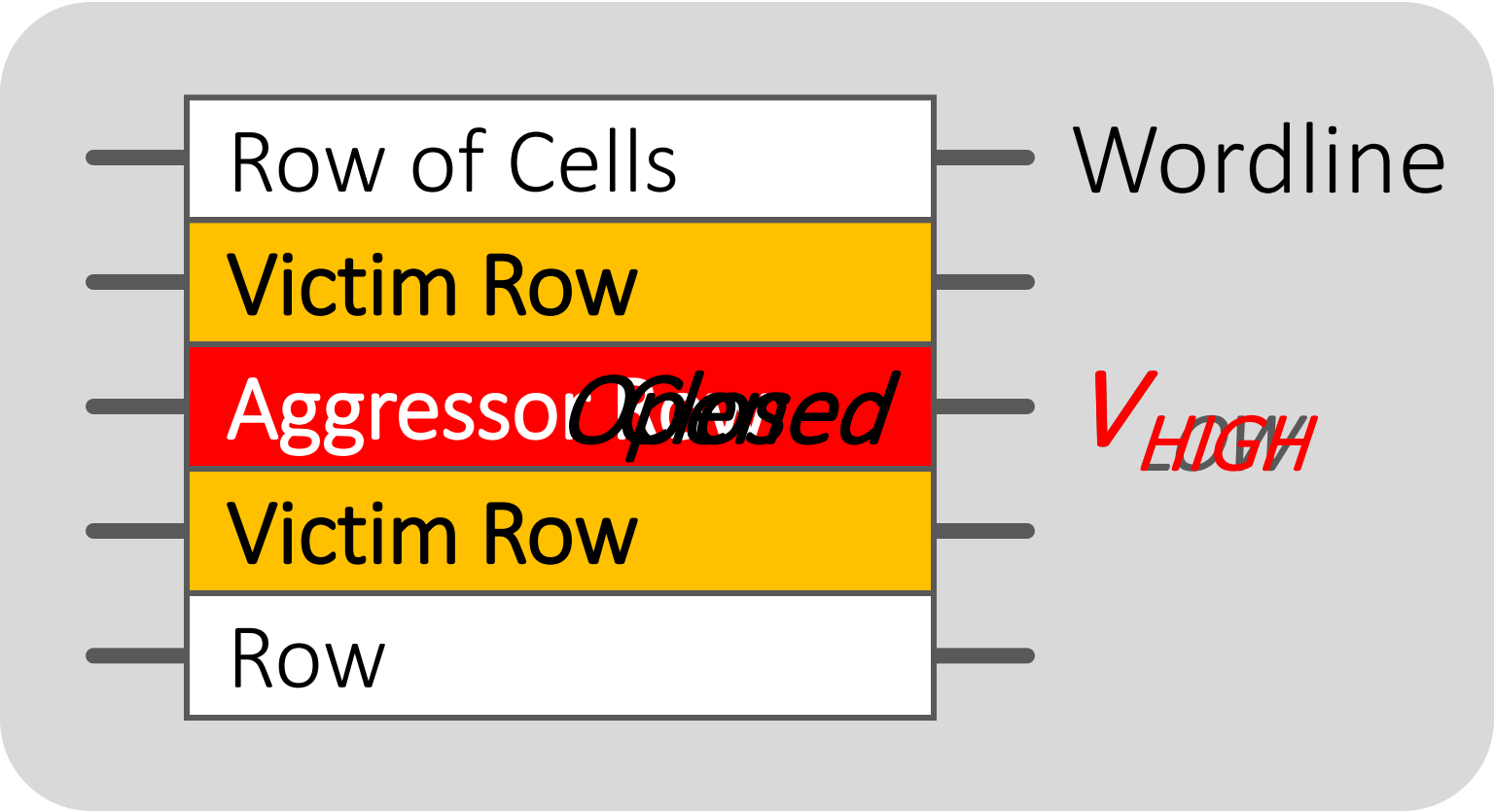
- Key idea: Refresh rows containing weak cells more frequently, other rows less frequently
  1. Profiling: Profile retention time of all rows
  2. Binning: Store rows into bins by retention time in memory controller
     Efficient storage with Bloom Filters (only 1.25KB for 32GB memory)
  3. Refreshing: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

Disturbance Errors in Modern DRAM

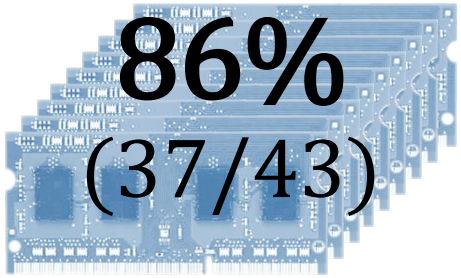


Repeatedly opening and closing a row enough times within a refresh interval induces disturbance errors in adjacent rows in most real DRAM chips you can buy today

Kim+, "Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors," ISCA 2014.

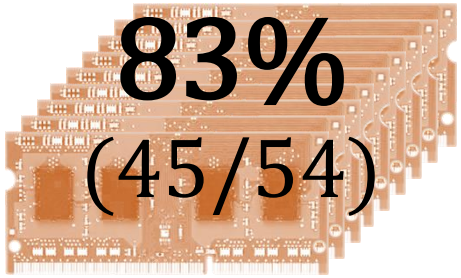
Most DRAM Modules Are At Risk

A company



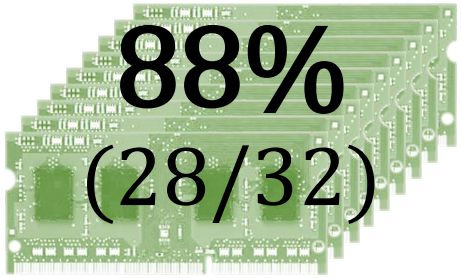
Up to
1.0×10^7
errors

B company



Up to
2.7×10^6
errors

C company

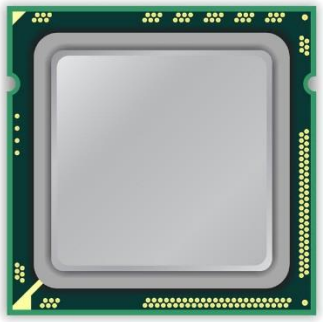


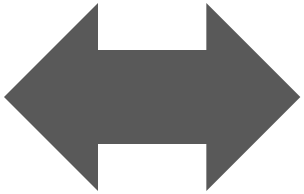
Up to
3.3×10^5
errors

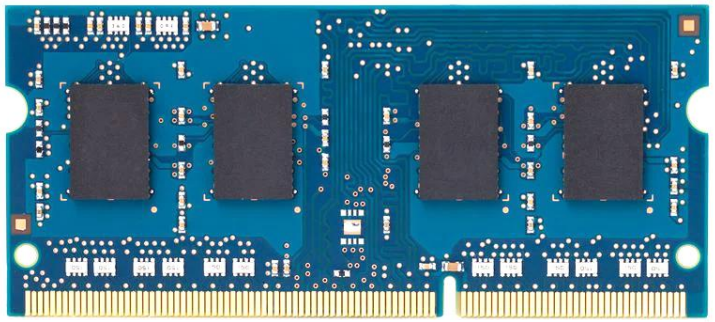
Kim+, "Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors," ISCA 2014.

x86 CPU

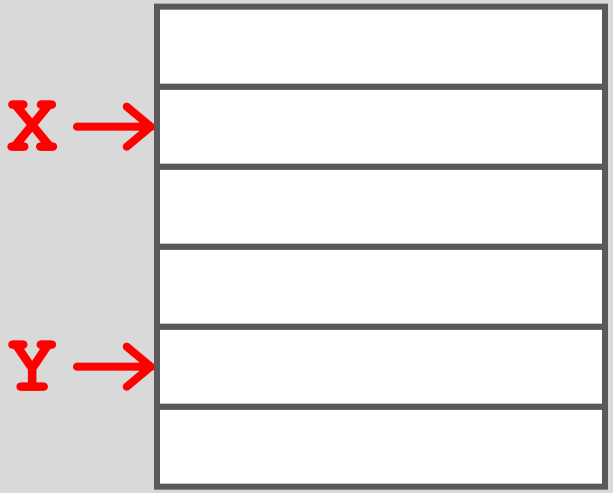
DRAM Module





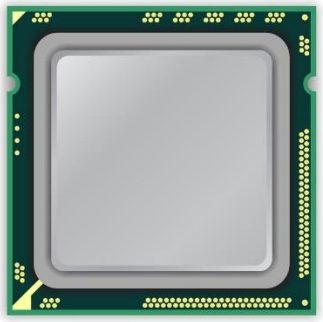


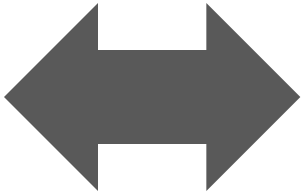
```
loop:
   mov (X), %eax
   mov (Y), %ebx
   clflush (X)
   clflush (Y)
   mfence
   jmp loop
```

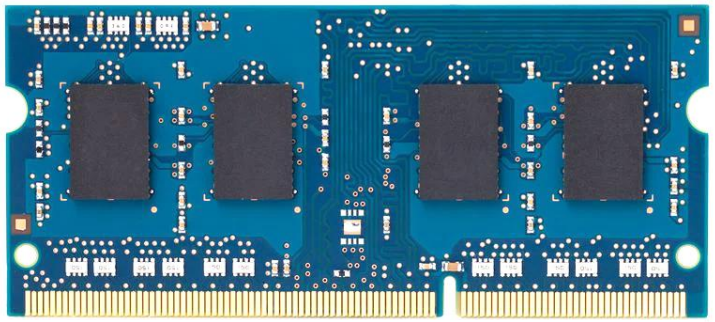


x86 CPU

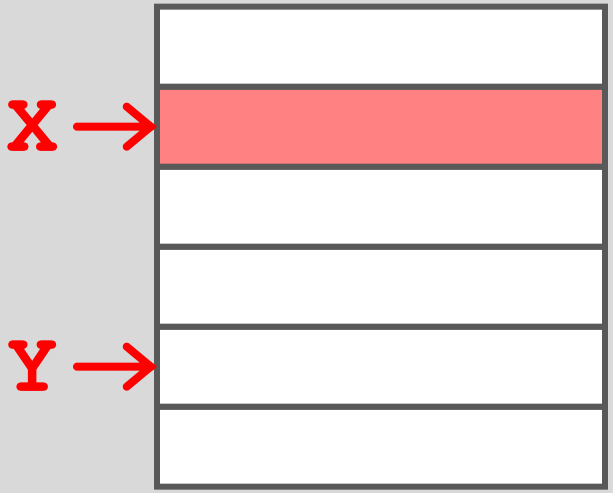
DRAM Module





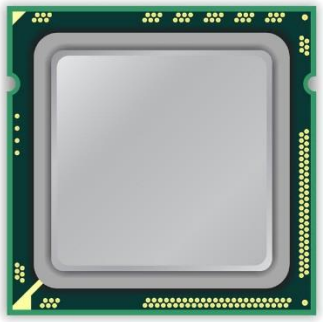


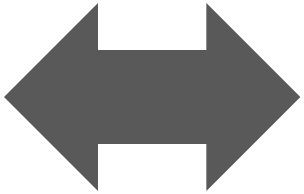
```
loop:
   mov (X), %eax
   mov (Y), %ebx
   clflush (X)
   clflush (Y)
   mfence
   jmp loop
```

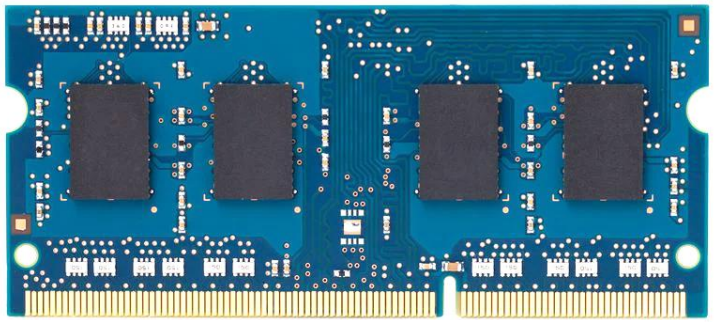


x86 CPU

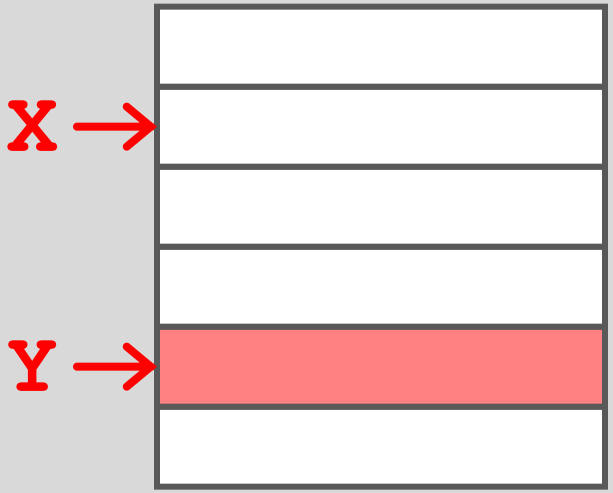
DRAM Module



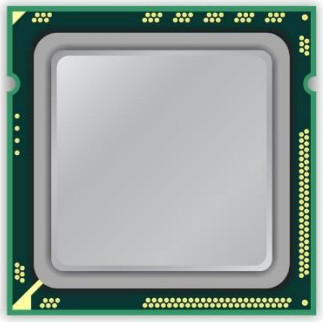




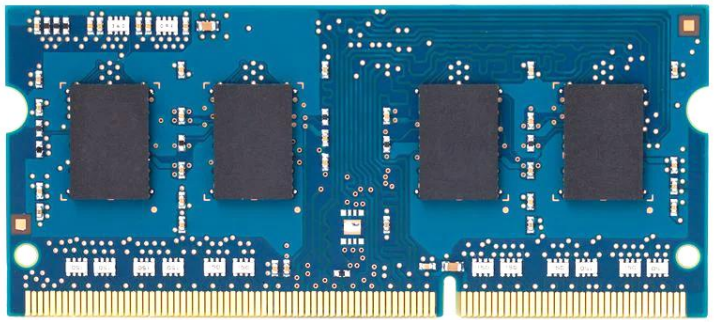
```
loop:
   mov (X), %eax
   mov (Y), %ebx
   clflush (X)
   clflush (Y)
   mfence
   jmp loop
```



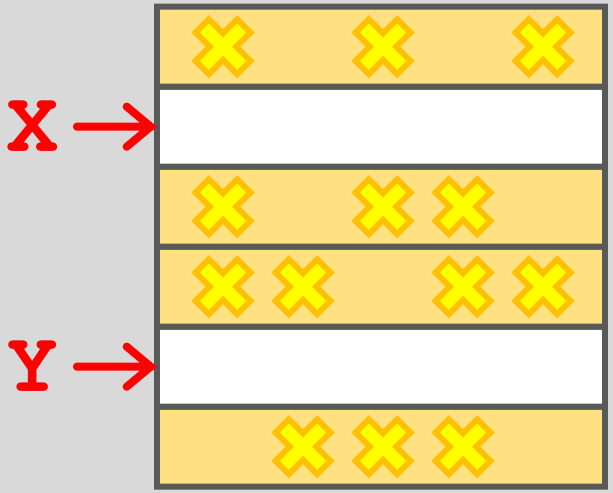
x86 CPU



DRAM Module



```
loop:
   mov (X), %eax
   mov (Y), %ebx
   clflush (X)
   clflush (Y)
   mfence
   jmp loop
```



Observed Errors in Real Systems

CPU Architecture	Errors	Access-Rate
Intel Haswell (2013)	22.9K	12.3M/sec
Intel Ivy Bridge (2012)	20.7K	11.7M/sec
Intel Sandy Bridge (2011)	16.1K	11.6M/sec
AMD Piledriver (2012)	59	6.1M/sec

- *A real reliability & security issue*
- *In a more controlled environment, we can induce as many as ten million disturbance errors*

Kim+, "Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors," ISCA 2014.

RowHammer Security Attack Example

- "Rowhammer" is a problem with some recent DRAM devices in which repeatedly accessing a row of memory can cause bit flips in adjacent rows (Kim et al., ISCA 2014).
  - Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors (Kim et al., ISCA 2014)
- they tested a selection of laptops and found that a subset of them exhibited the problem.
- they built two working privilege escalation exploits that use this effect. Exploiting the DRAM rowhammer bug to gain kernel privileges (Seaborn, 2015)
- One exploit uses rowhammer-induced bit flips to gain kernel privileges on x86-64 Linux when run as an unprivileged userland process.
- When run on a machine vulnerable to the rowhammer problem, the process was able to induce bit flips in page table entries .
- It was able to use this to gain write access to its own page table, and hence gain read-write access to all of physical memory.

Security Implications

It’s like breaking into an apartment by repeatedly slamming a neighbor’s door until the vibrations open the door you were after

Some Potential Solutions

- Make better DRAM chips — Cost
- Refresh frequently — Power, Performance
- Sophisticated Error Correction — Cost, Power
- Access counters — Cost, Power, Complexity

Recap: Some Goals of This Course

- Teach/enable/empower you to:
  - Understand how a computing platform (processor + memory + interconnect) works
  - Implement a simple platform (with not so simple parts), with a focus on the processor and memory
  - 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

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 system as seen by the programmer, i.e., the conceptual structure and functional behavior as distinct from the organization of the dataflow and controls, the logic design, and the physical implementation." *Gene Amdahl*, IBM Journal of R&D, April 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).

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, design tradeoffs, key current/future issues
  - Strong emphasis on looking backward, forward, up and down

- 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, hands-on design & implementation, and efficiency.
  - Strong emphasis on making things work, realizing ideas