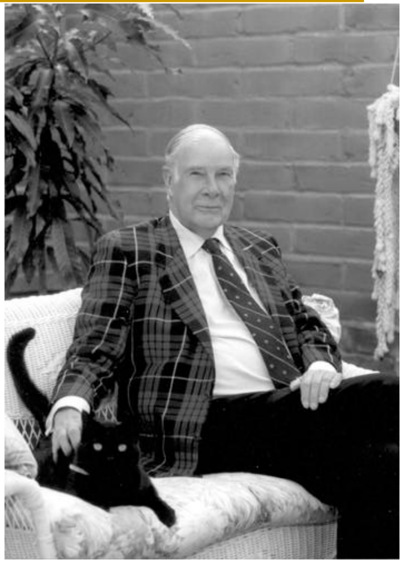
Computer Architecture
Lecture 3c:
Memory Performance Attacks

Prof. Onur Mutlu
ETH Zürich
Fall 2021
7 October 2021

Recall: 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?

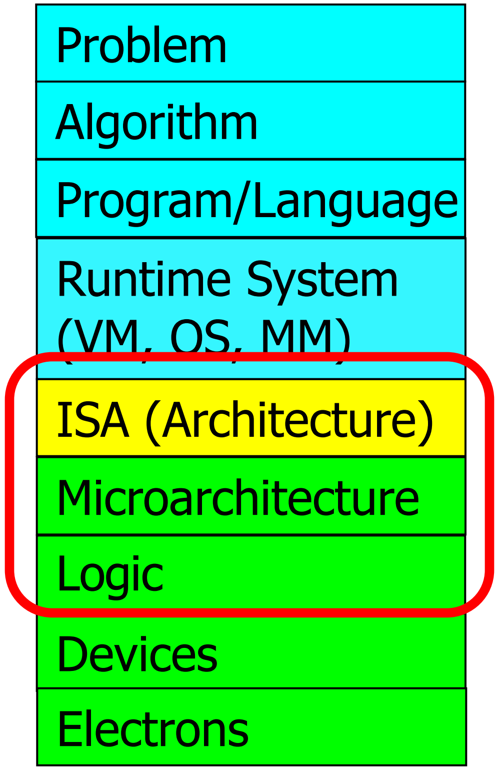


Algorithm

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)

Recall: 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?

Recall: Crossing the Abstraction Layers

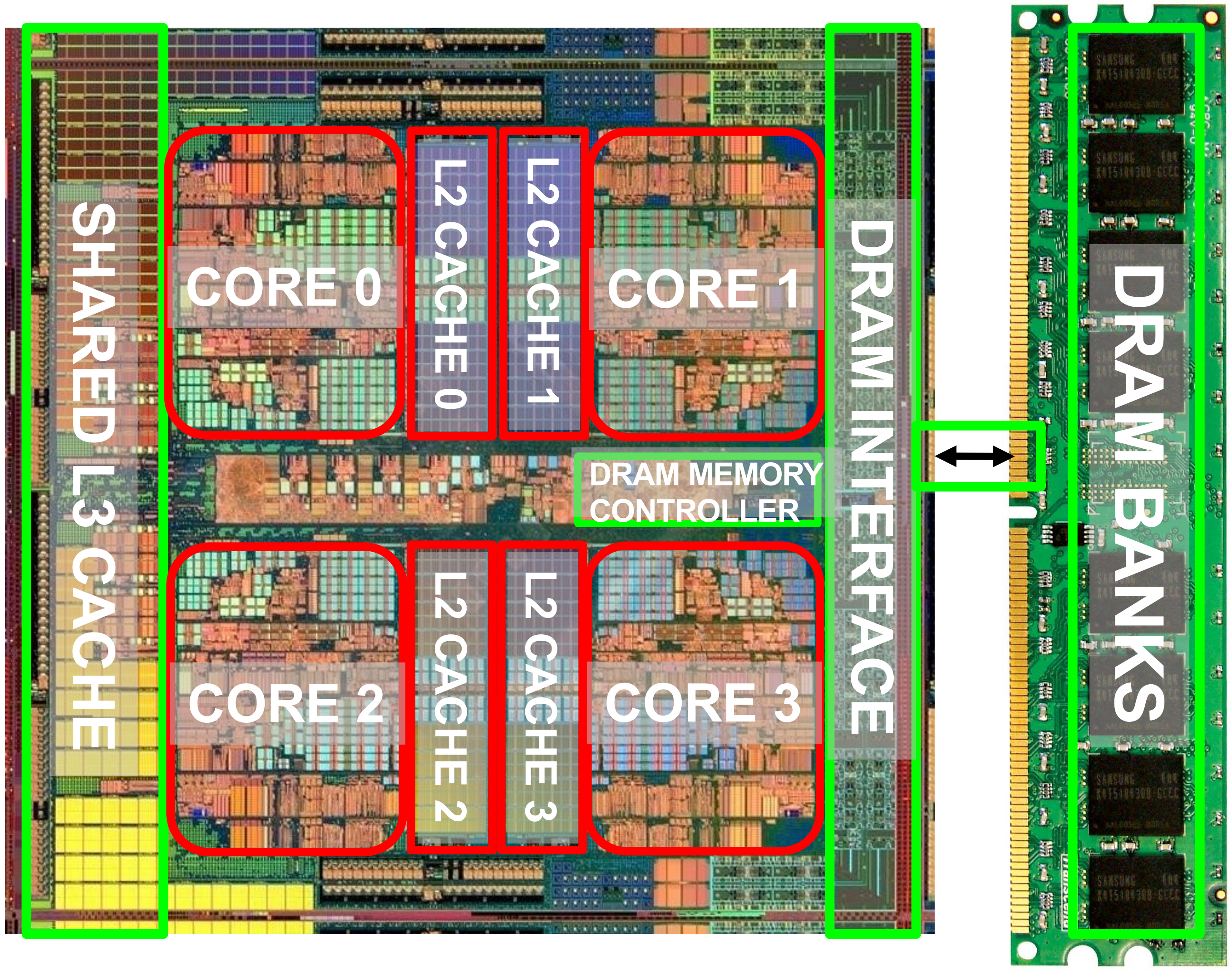
- As long as everything goes well, not knowing what happens underneath (or above) is not a problem.
- What if
  - The program you wrote is running slow?
  - The program you wrote does not run correctly?
  - The program you wrote consumes too much energy?
  - Your system just shut down and you have no idea why?
  - Someone just compromised your system and you have no idea how?
- 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?

Recall: Crossing the Abstraction Layers

- Two key goals of this course are
  - to understand how a computing system 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

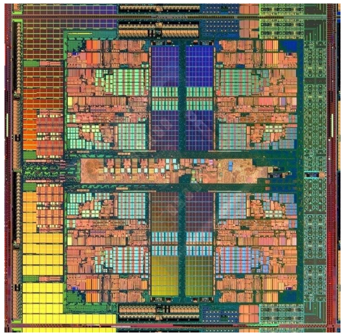
Multi-Core Chip

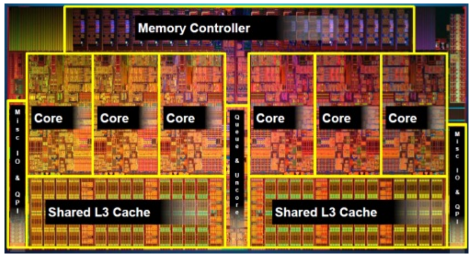


*Die photo credit: AMD Barcelona

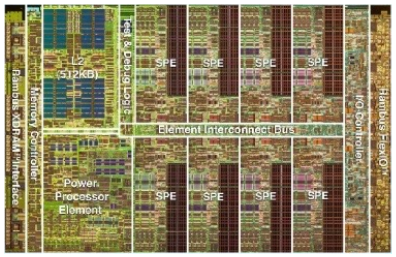
A Trend: Many Cores on Chip

- Simpler and lower power than a single large core
- Parallel processing on single chip → faster, new applications

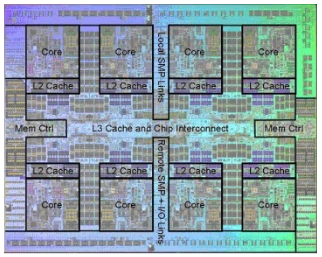

AMD Barcelona
4 cores



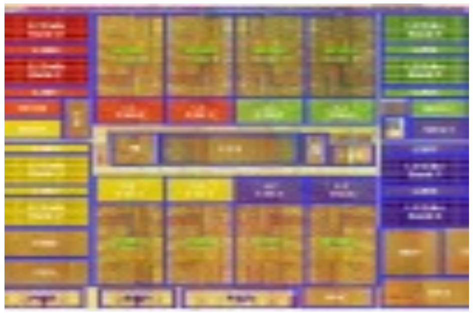
Intel Core i7
8 cores

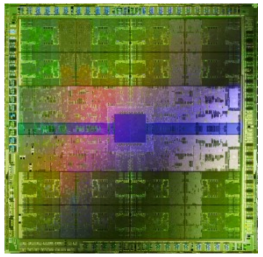


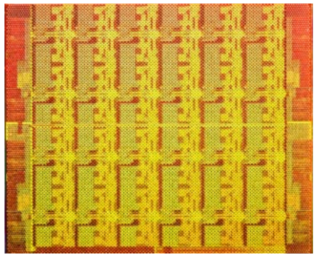
IBM Cell BE
8+1 cores

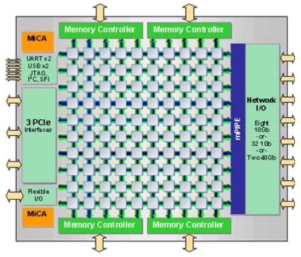


IBM POWER7
8 cores


Sun Niagara II
8 cores

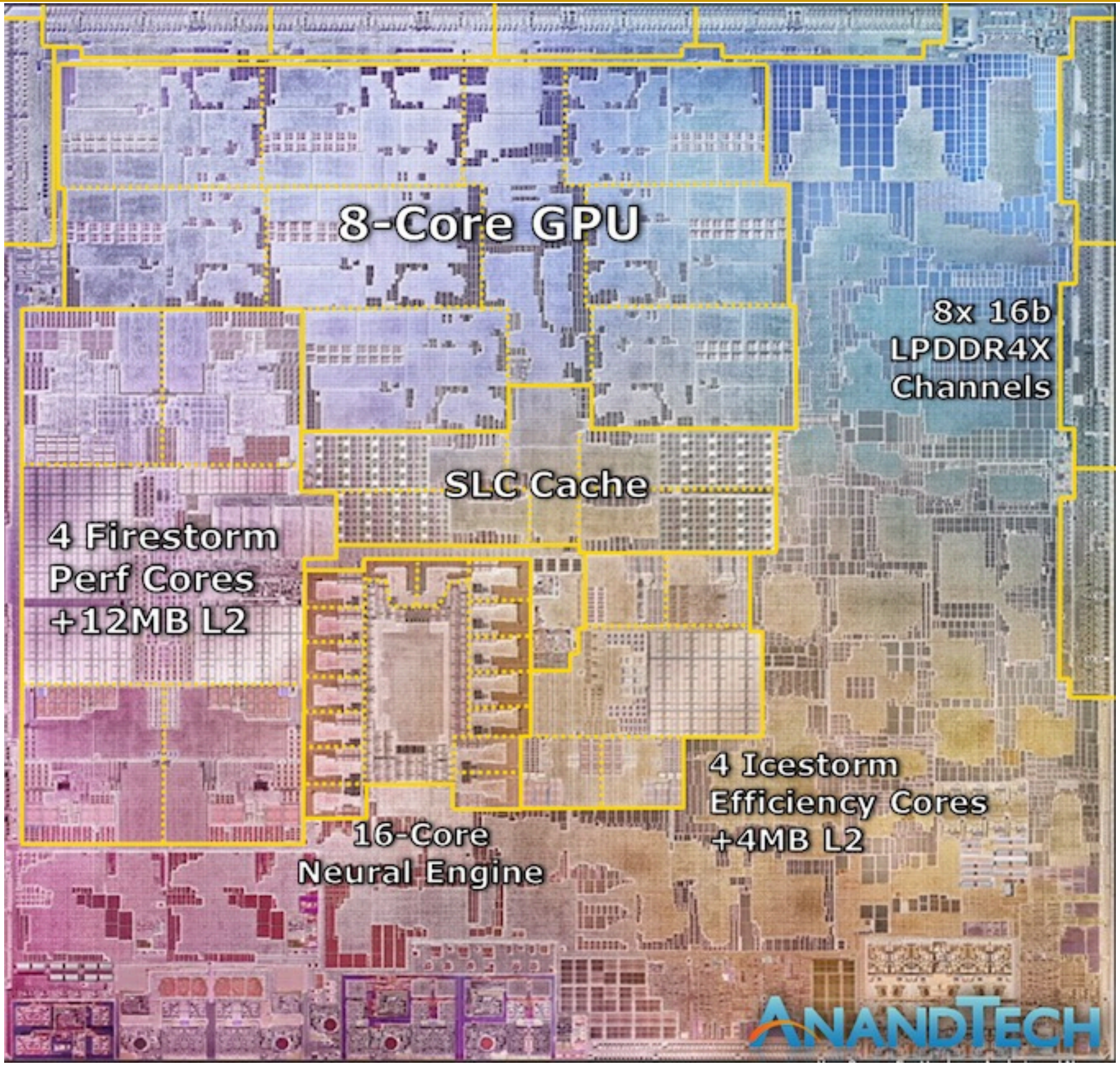

Nvidia Fermi
448 “cores”


Intel SCC
48 cores, networked



Tilera TILE Gx
100 cores, networked

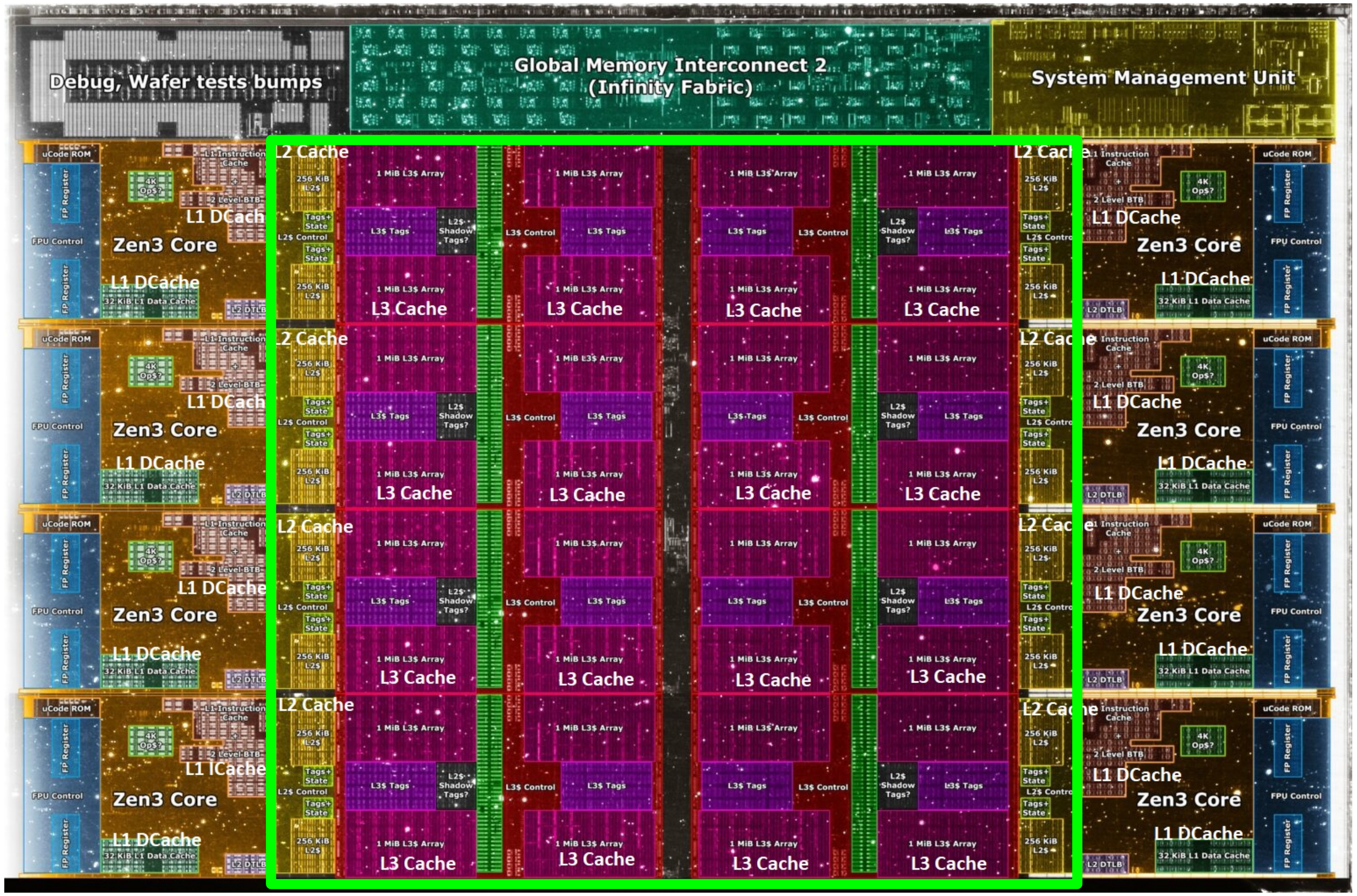
More Recent Multi-Core Systems (I)



Apple M1, 2021

Source: https://www.anandtech.com/show/16252/mac-mini-apple-m1-tested

More Recent Multi-Core Systems (II)



AMD Ryzen 5000, 2020

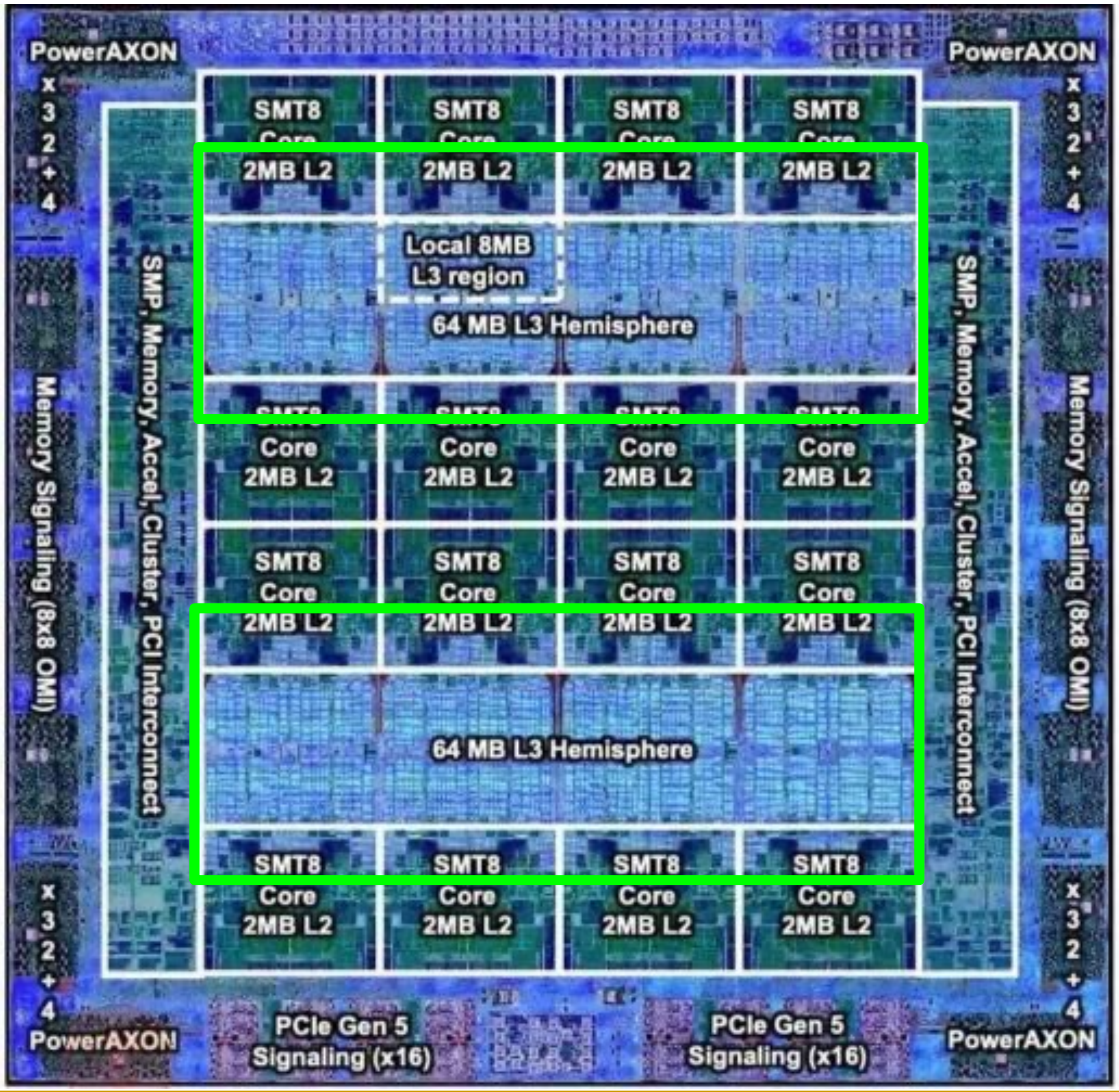
Core Count:
8 cores/16 threads

L1 Caches:
32 KB per core

L2 Caches:
512 KB per core

L3 Cache:
32 MB shared

More Recent Multi-Core Systems (III)



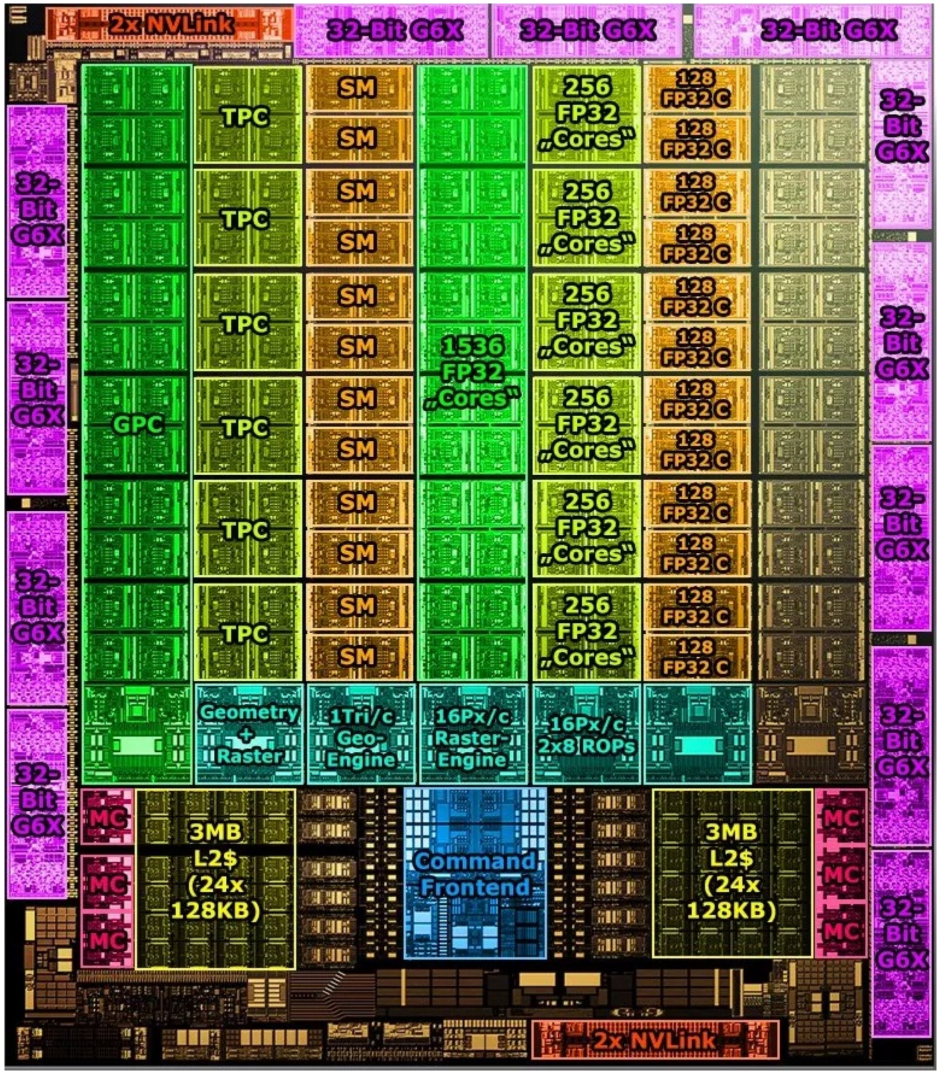
IBM POWER10, 2020

Cores:
15-16 cores,
8 threads/core

L2 Caches:
2 MB per core

L3 Cache:
120 MB shared

More Recent Multi-Core Systems (IV)



Nvidia Ampere, 2020

Cores:

128 Streaming Multiprocessors

L1 Cache or Scratchpad:

192KB per SM

Can be used as L1 Cache and/or Scratchpad

L2 Cache:

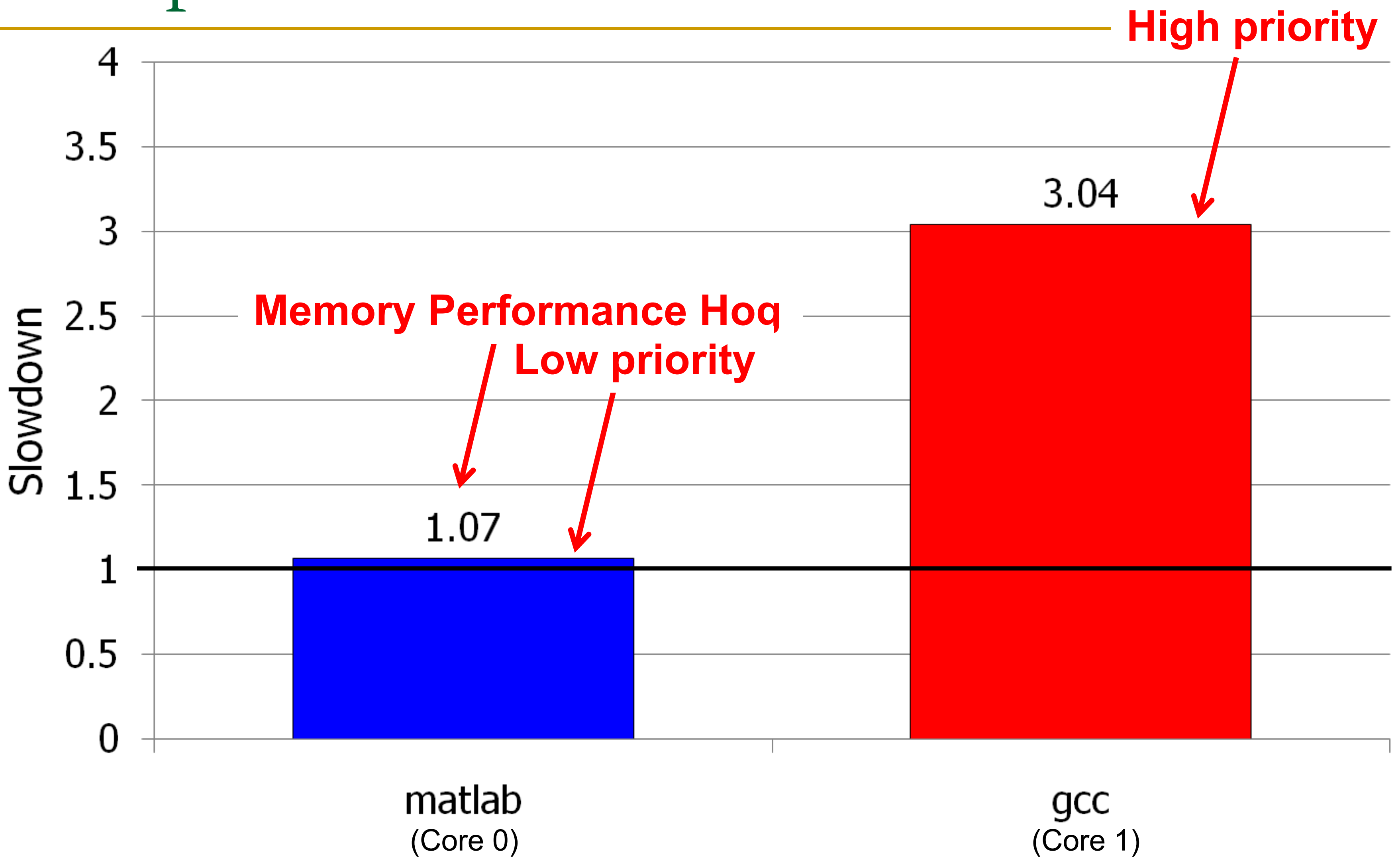
40 MB shared

Many Cores on Chip

- What we want:
  - N times the system performance with N times the cores

- What do we get today?

Unexpected Slowdowns in Multi-Core



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

Three Questions

- Can you figure out why the applications slow down if you do not know the underlying system and how it works?

- 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”?

Three Questions: Rephrased & Concise

- Why is there any slowdown?
- Why is there a disparity in slowdowns?
- How can we solve the problem if we do not want that disparity?

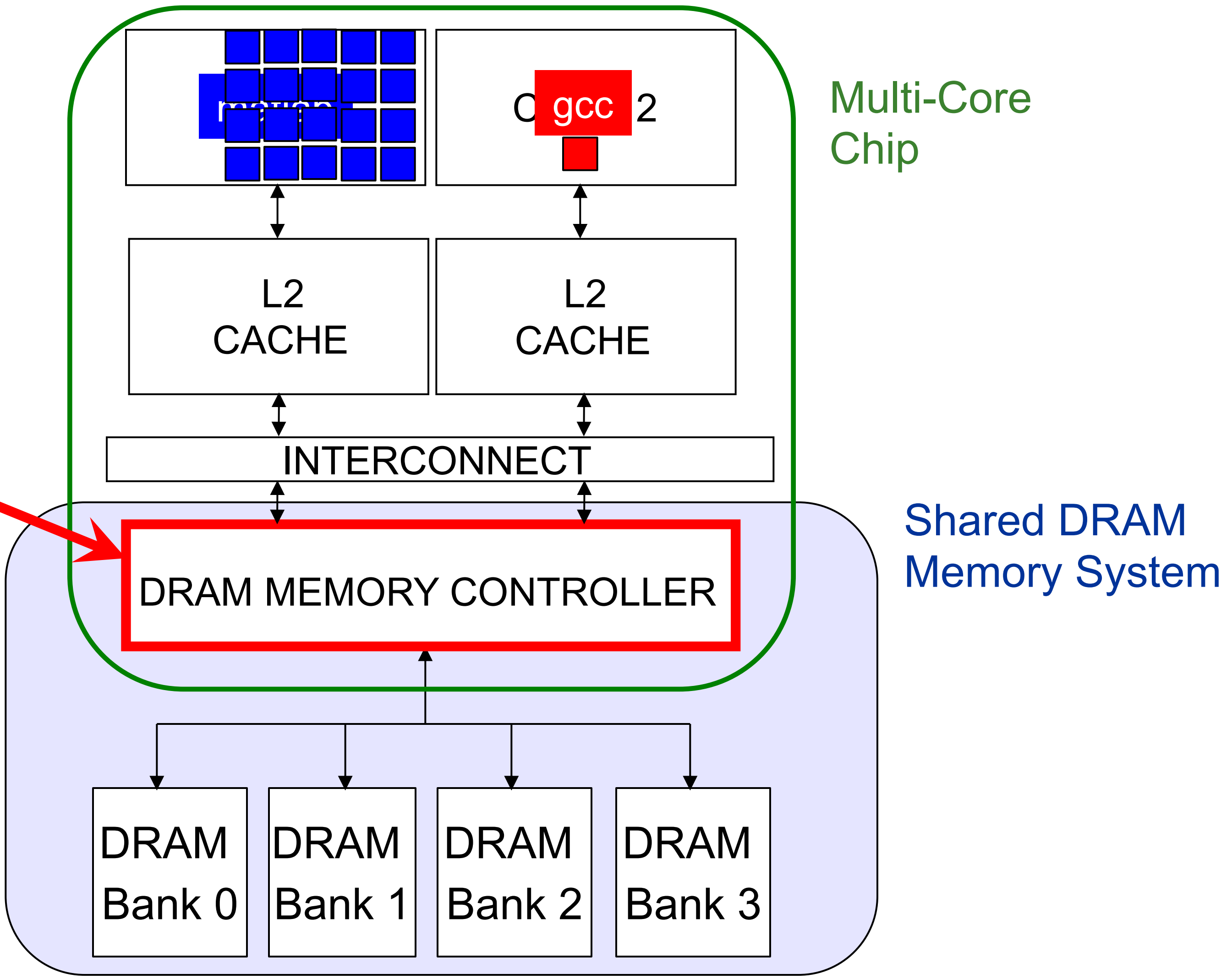
Why Is This Important?

- We want to execute applications in parallel in multi-core systems → consolidate more and more (for efficiency)
  - Cloud computing
  - Mobile phones
  - Automotive systems

- We want to mix different types of applications together
  - those requiring QoS guarantees (e.g., video, pedestrian detection)
  - those that are important but less so
  - those that are less important

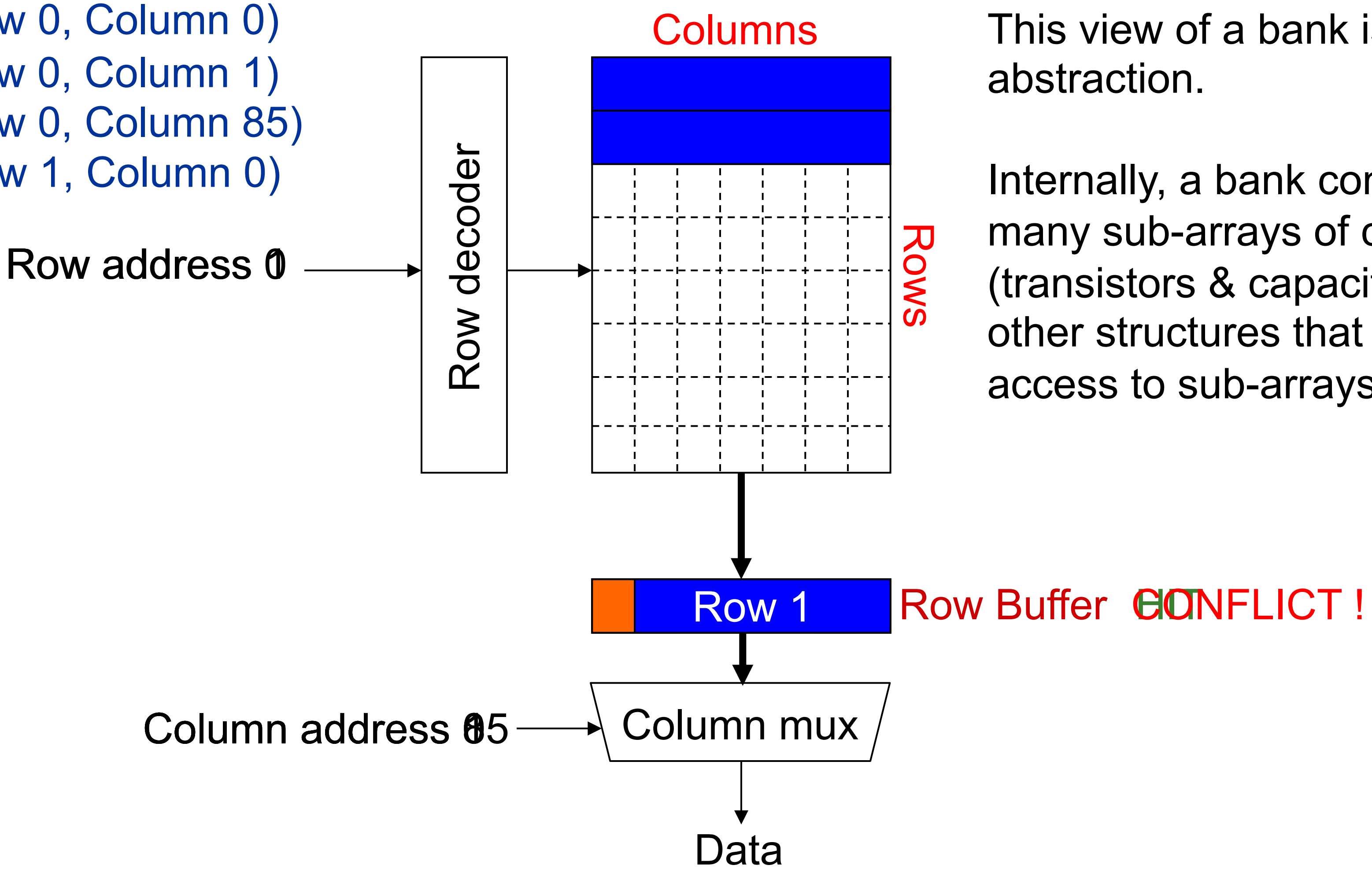
- We want the system to be controllable **and** high performance

Why the Disparity in Slowdowns?



Digging Deeper: DRAM Bank Operation

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



This view of a bank is an abstraction.

Internally, a bank consists of many sub-arrays of cells (transistors & capacitors) and other structures that enable access to sub-arrays & cells

DRAM Controllers

- A row-conflict memory access takes significantly longer than a row-hit access

- Current controllers take advantage of this fact

- 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];     (in sequence)
    …
}
```

```
// initialize large arrays A, B

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

STREAM

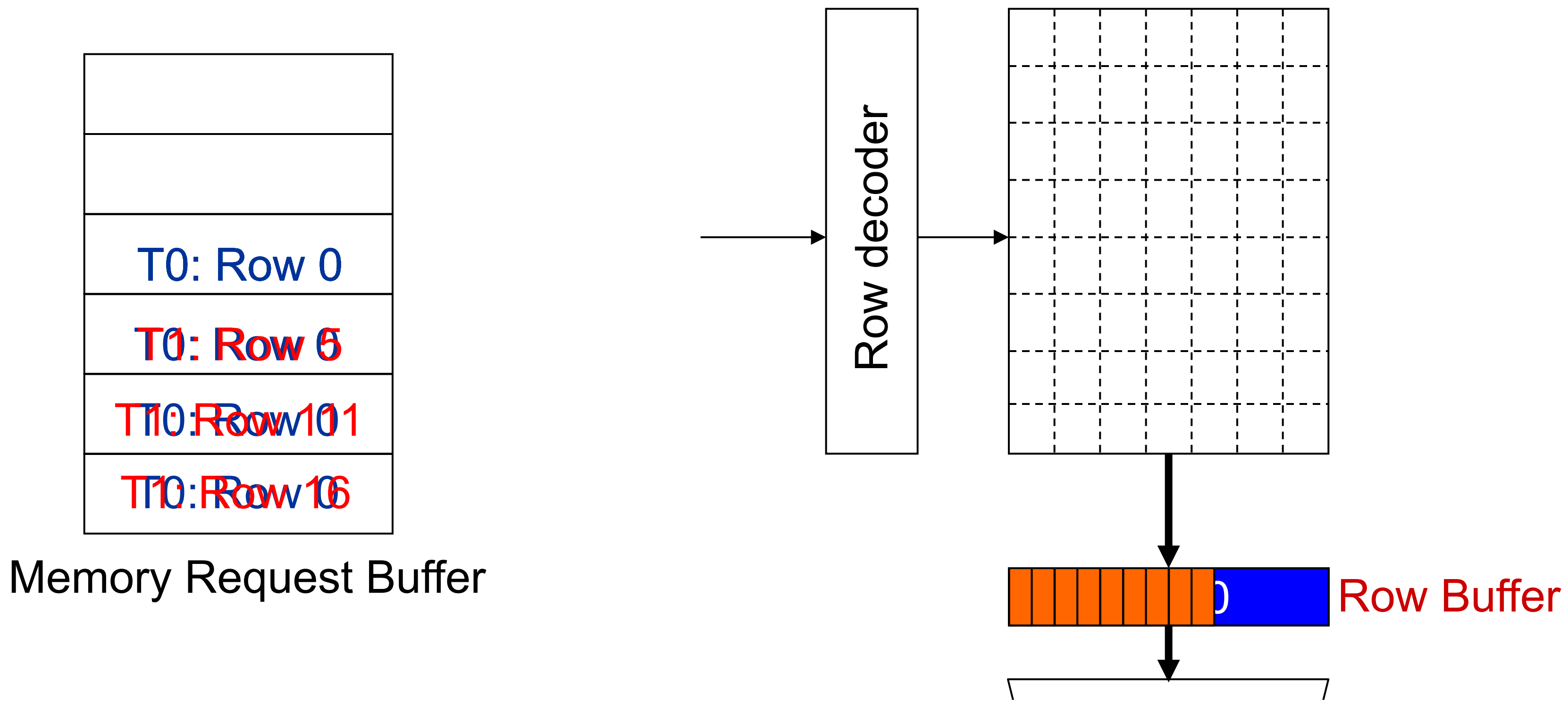
- Sequential memory access
- Very high row buffer locality (96% hit rate)
- Memory intensive

RANDOM

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

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

What Does the Memory Hog Do?

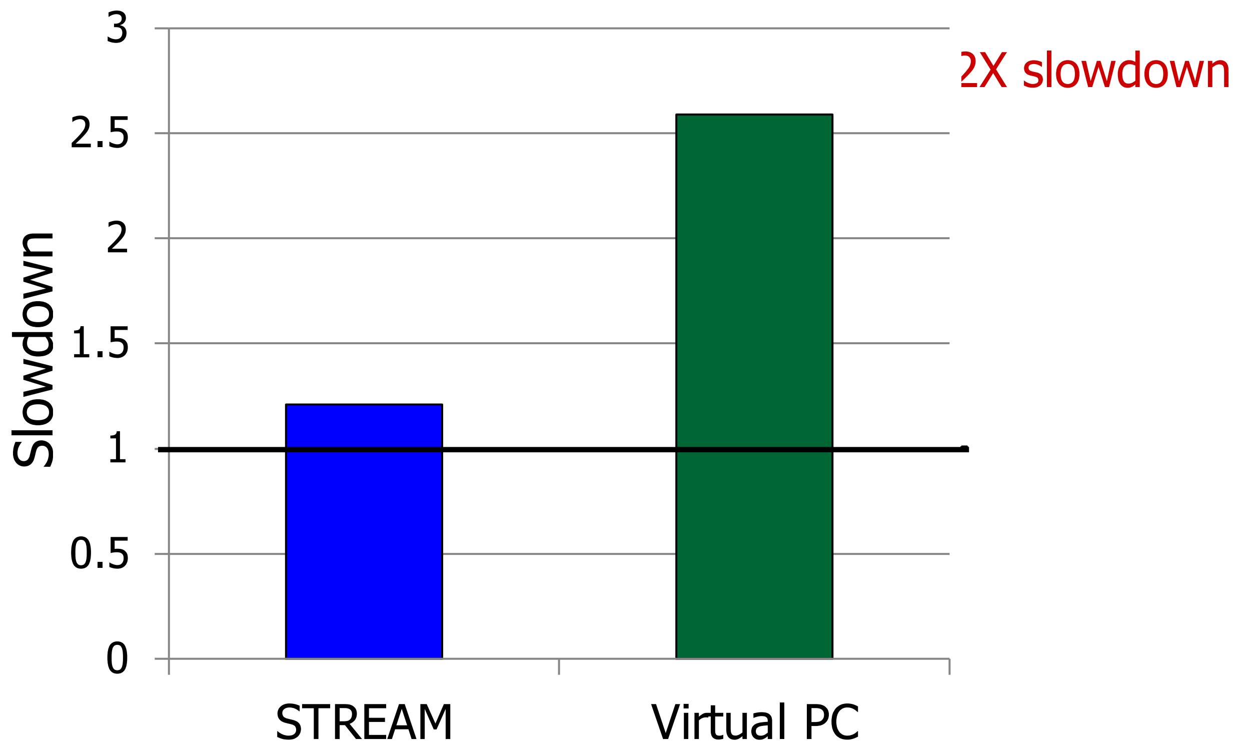


Row size: 8KB, request size: 64B

128 (8KB/64B) requests of STREAM serviced before a single request of RANDOM

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

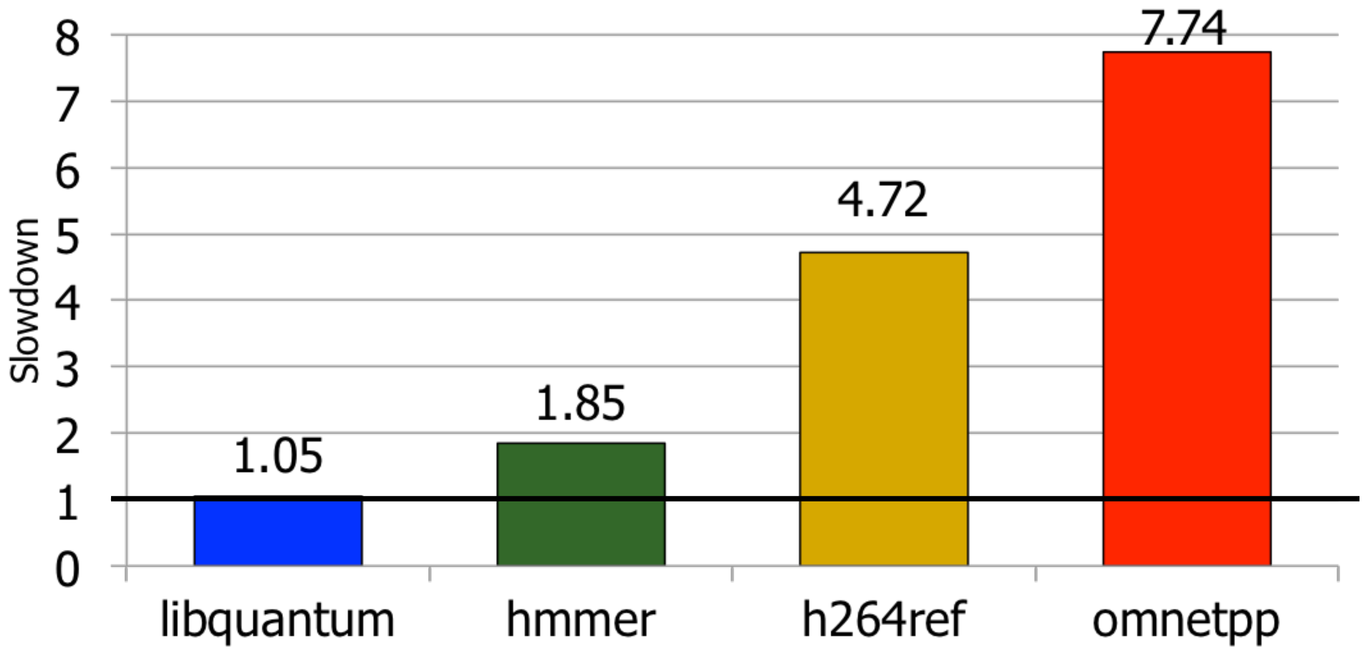
Effect of the Memory Performance Hog



Results on Intel Pentium D running Windows XP
(Similar results for Intel Core Duo and AMD Turion, and on Fedora Linux)

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

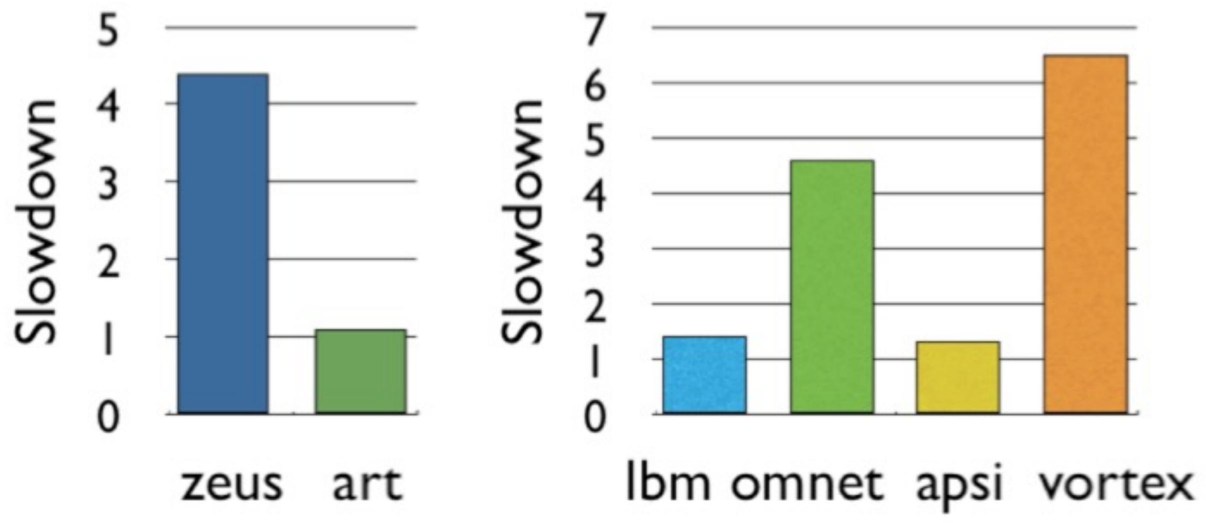
Greater Problem with More Cores



- Vulnerable to denial of service (DoS)
- Unable to enforce priorities or SLAs
- Low system performance

Uncontrollable, unpredictable system

Greater Problem with More Cores



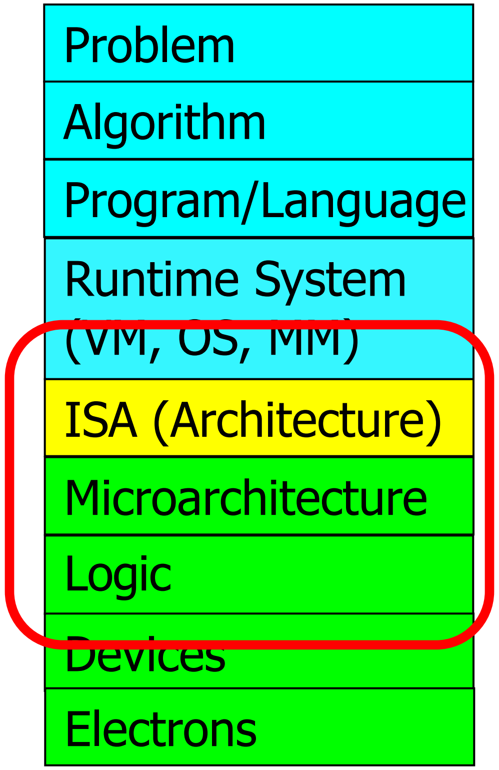
- Vulnerable to denial of service (DoS)
- Unable to enforce priorities or SLAs
- Low system performance

Uncontrollable, unpredictable system

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?

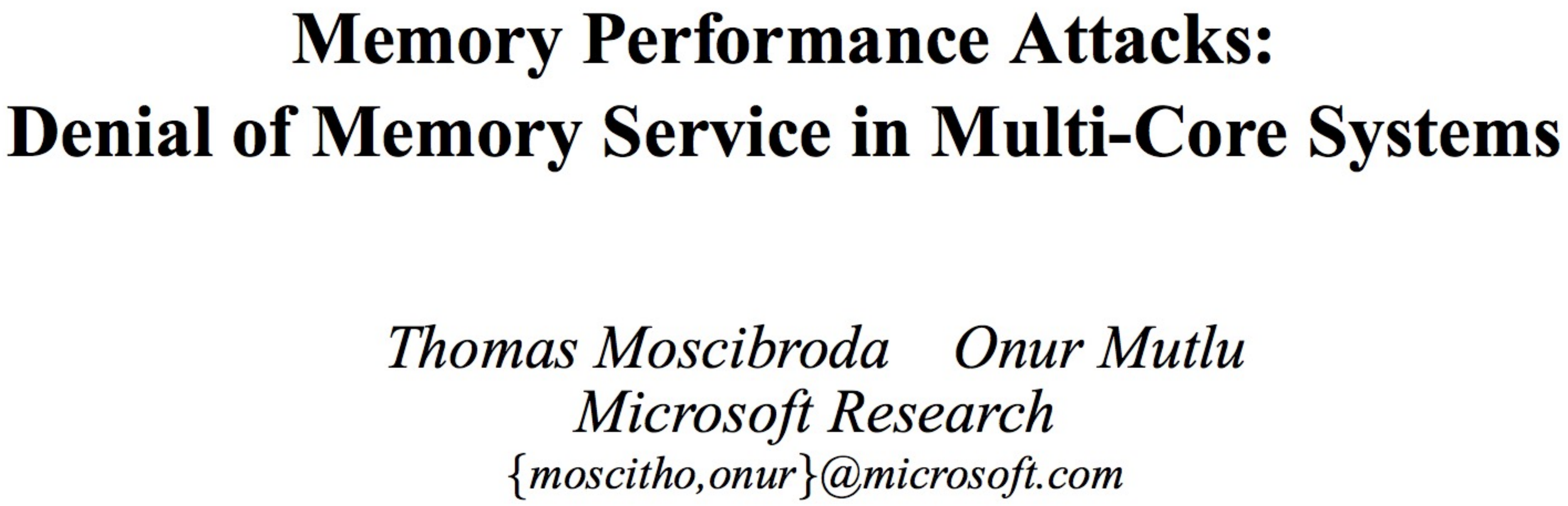


- Two other goals of this course:
  - Enable you to think critically
  - Enable you to think broadly

Reading on Memory Performance Attacks

- Thomas Moscibroda and Onur Mutlu,
  ["Memory Performance Attacks: Denial of Memory Service in Multi-Core Systems"]
  *Proceedings of the 16th USENIX Security Symposium (**USENIX SECURITY**)*,
  pages 257-274, Boston, MA, August 2007. Slides (ppt)

- One potential reading for your Homework 1 assignment



Memory Performance Attacks:
Denial of Memory Service in Multi-Core Systems

Thomas Moscibroda   *Onur Mutlu*
Microsoft Research
{moscitho,onur}@microsoft.com

Conclusions [USENIX Security'07]

- Introduced the notion of memory performance attacks in shared DRAM memory systems
- Unfair DRAM scheduling is the cause of the vulnerability
- More severe problem in future many-core systems

- We provide a novel definition of DRAM fairness
  - Threads should experience equal slowdowns
- New DRAM scheduling algorithm enforces this definition
  - Effectively prevents memory performance attacks

- Preventing attacks also improves system throughput!

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)

- Onur Mutlu and Thomas Moscibroda,
  "Parallelism-Aware Batch Scheduling: Enhancing both Performance and Fairness of Shared DRAM Systems"
  *Proceedings of the 35th International Symposium on Computer Architecture (**ISCA**)* [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)

A Recent Solution: BLISS

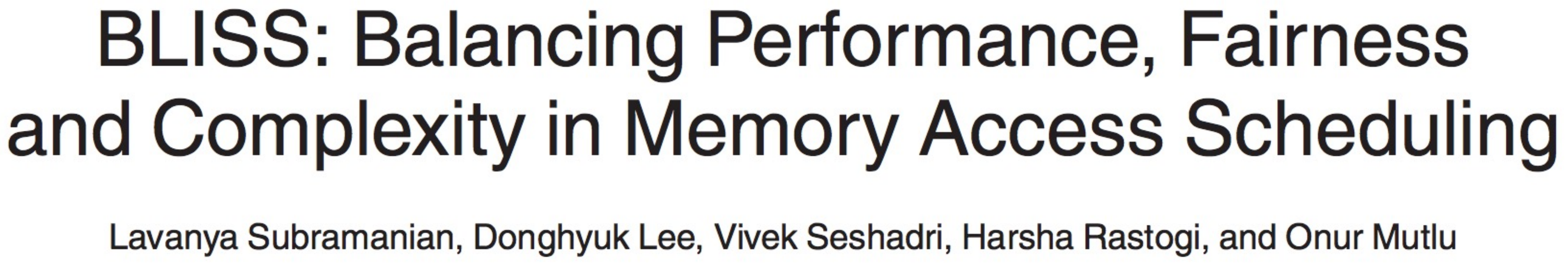
- Lavanya Subramanian, Donghyuk Lee, Vivek Seshadri, Harsha Rastogi, and Onur Mutlu,
"The Blacklisting Memory Scheduler: Achieving High Performance and Fairness at Low Cost"
Proceedings of the 32nd IEEE International Conference on Computer Design (***ICCD***), Seoul, South Korea, October 2014.
[Slides (pptx) (pdf)]

The Blacklisting Memory Scheduler:
Achieving High Performance and Fairness at Low Cost

Lavanya Subramanian, Donghyuk Lee, Vivek Seshadri, Harsha Rastogi, Onur Mutlu
Carnegie Mellon University
{lsubrama,donghyu1,visesh,harshar,onur}@cmu.edu

More on BLISS: Longer Version

- Lavanya Subramanian, Donghyuk Lee, Vivek Seshadri, Harsha Rastogi, and Onur Mutlu,
  "BLISS: Balancing Performance, Fairness and Complexity in Memory Access Scheduling"
  IEEE Transactions on Parallel and Distributed Systems (***TPDS***), to appear in 2016. arXiv.org version, April 2015.
  An earlier version as *SAFARI Technical Report*, TR-SAFARI-2015-004, Carnegie Mellon University, March 2015.
  [Source Code]



BLISS: Balancing Performance, Fairness and Complexity in Memory Access Scheduling

Lavanya Subramanian, Donghyuk Lee, Vivek Seshadri, Harsha Rastogi, and Onur Mutlu

Many Potential Solutions w/ Tradeoffs

Computer Architecture

Lecture 11a: Memory Controllers

Prof. Onur Mutlu

ETH Zürich

Fall 2020

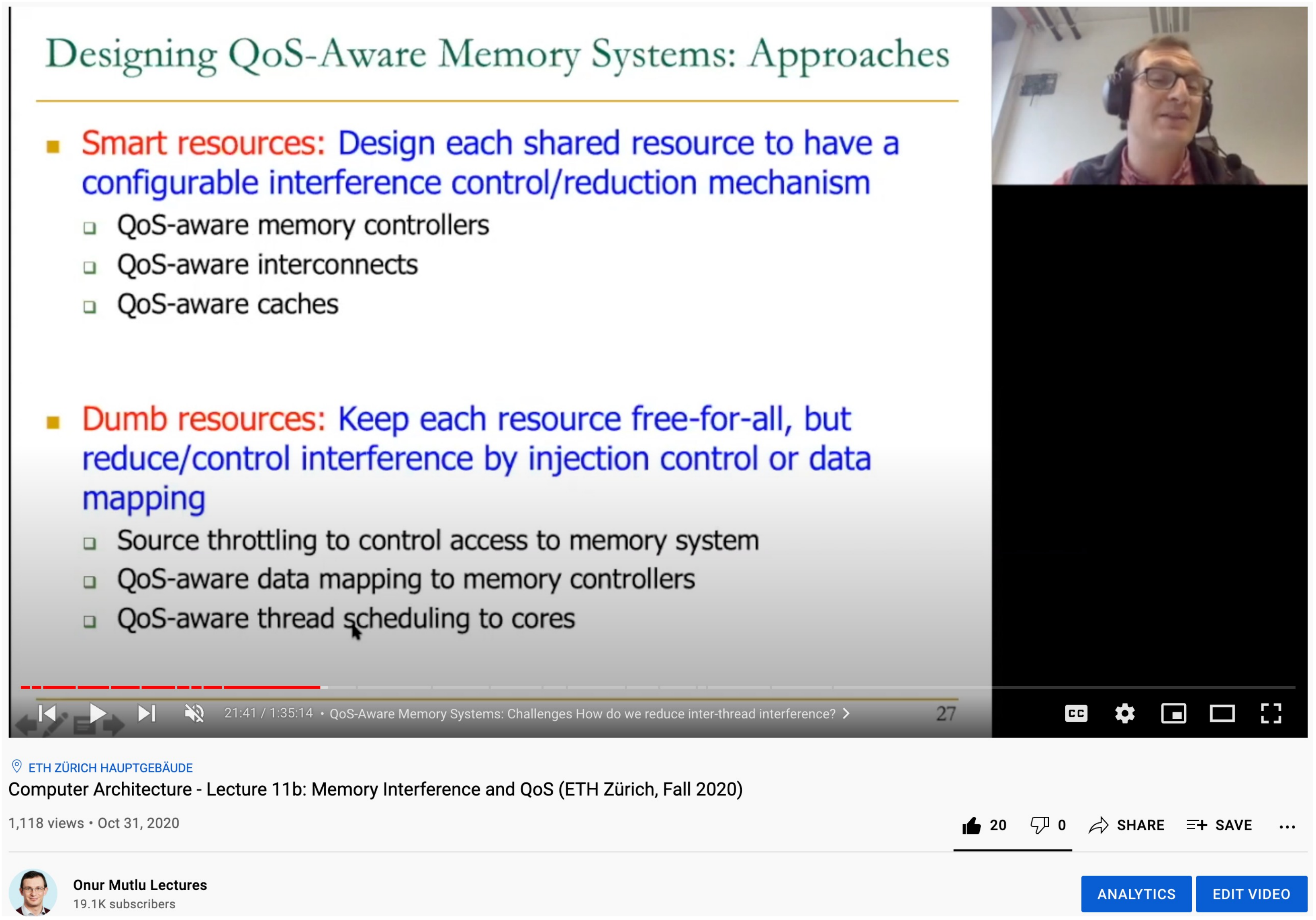
29 October 2020

Computer Architecture - Lecture 11a: Memory Controllers (ETH Zürich, Fall 2020)

5,226 views • Oct 31, 2020

https://www.youtube.com/watch?v=TeG773OgiMQ&list=PL5Q2soXY2Zi9xidyIgBxUz7xRPS-wisBN&index=20

Many Potential Solutions w/ Tradeoffs



Designing QoS-Aware Memory Systems: Approaches

- Smart resources: Design each shared resource to have a configurable interference control/reduction mechanism
  - QoS-aware memory controllers
  - QoS-aware interconnects
  - QoS-aware caches
- Dumb resources: Keep each resource free-for-all, but reduce/control interference by injection control or data mapping
  - Source throttling to control access to memory system
  - QoS-aware data mapping to memory controllers
  - QoS-aware thread scheduling to cores

21:41 / 1:35:14 • QoS-Aware Memory Systems: Challenges How do we reduce inter-thread interference?

27

ETH ZÜRICH HAUPTGEBÄUDE

Computer Architecture - Lecture 11b: Memory Interference and QoS (ETH Zürich, Fall 2020)

1,118 views • Oct 31, 2020

https://www.youtube.com/watch?v=0nnI807nCkc&list=PL5Q2soXY2Zi9xidyIgBxUz7xRPS-wisBN&index=21

Memory Channel Partitioning

- 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)

Reducing Memory Interference in Multicore Systems via Application-Aware Memory Channel Partitioning

Sai Prashanth Muralidhara
Pennsylvania State University
smuralid@cse.psu.edu

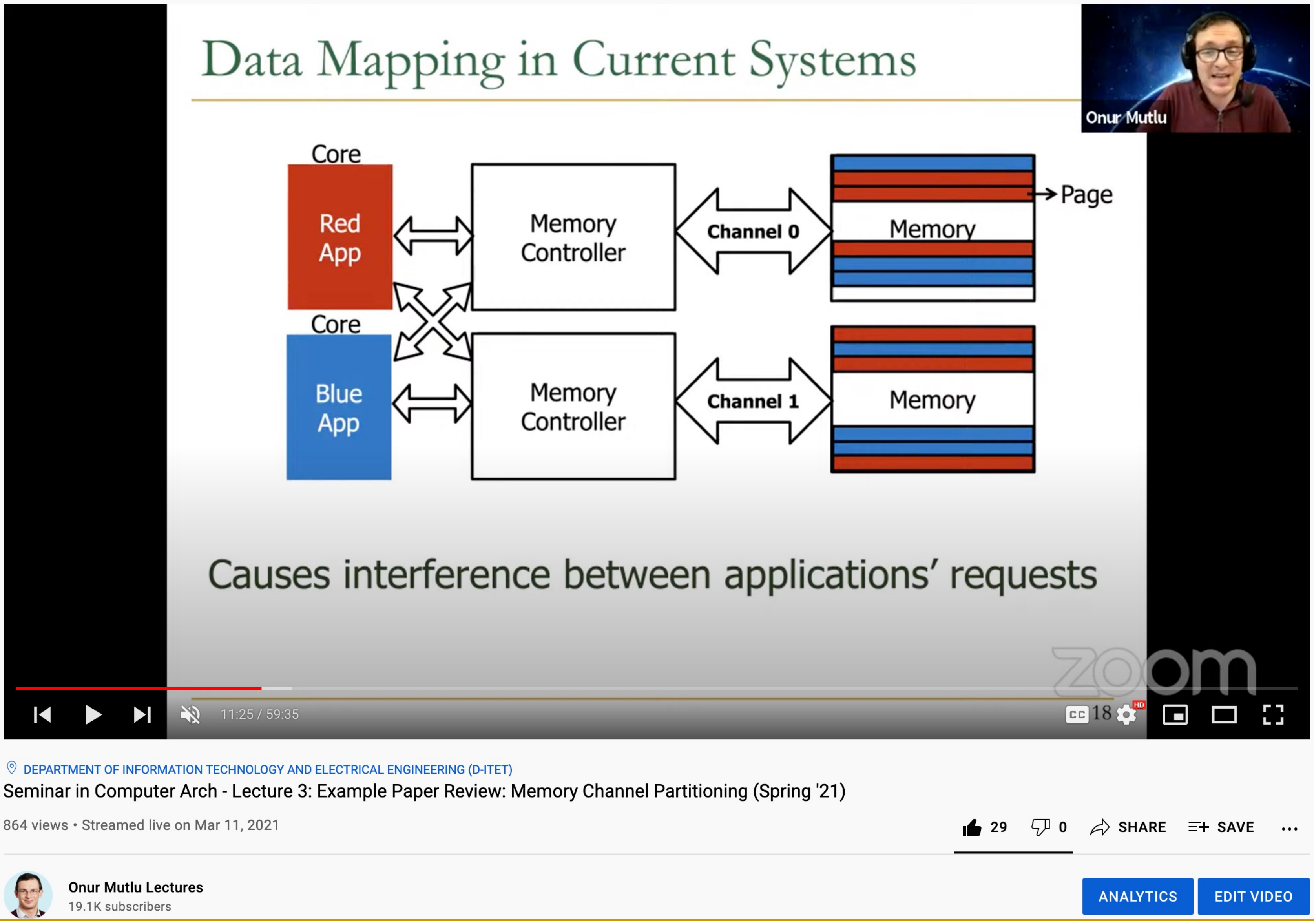
Lavanya Subramanian
Carnegie Mellon University
lsubrama@ece.cmu.edu

Onur Mutlu
Carnegie Mellon University
onur@cmu.edu

Mahmut Kandemir
Pennsylvania State University
kandemir@cse.psu.edu

Thomas Moscibroda
Microsoft Research Asia
moscitho@microsoft.com

Memory Channel Partitioning



Data Mapping in Current Systems

Causes interference between applications' requests

DEPARTMENT OF INFORMATION TECHNOLOGY AND ELECTRICAL ENGINEERING (D-ITET)

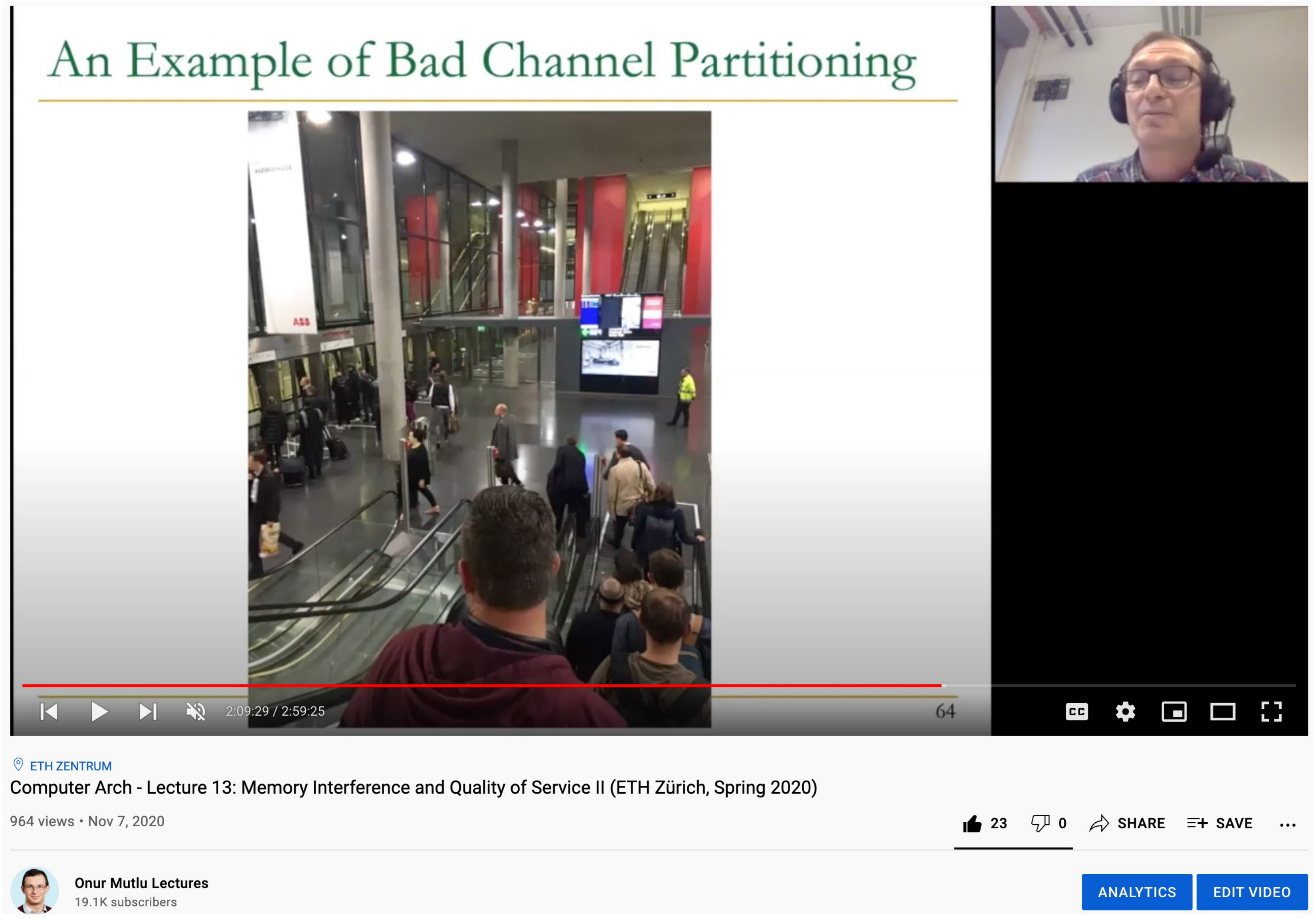
Seminar in Computer Arch - Lecture 3: Example Paper Review: Memory Channel Partitioning (Spring '21)

864 views • Streamed live on Mar 11, 2021

Onur Mutlu Lectures
19.1K subscribers

https://www.youtube.com/watch?v=MfEMpsnB93E&list=PL5Q2soXY2Zi_awYdjmWVIUegsbY7TPGW4&index=3

Many Potential Solutions w/ Tradeoffs



An Example of Bad Channel Partitioning

2:09:29 / 2:59:25

64

ETH ZENTRUM

Computer Arch - Lecture 13: Memory Interference and Quality of Service II (ETH Zürich, Spring 2020)

964 views • Nov 7, 2020

Onur Mutlu Lectures

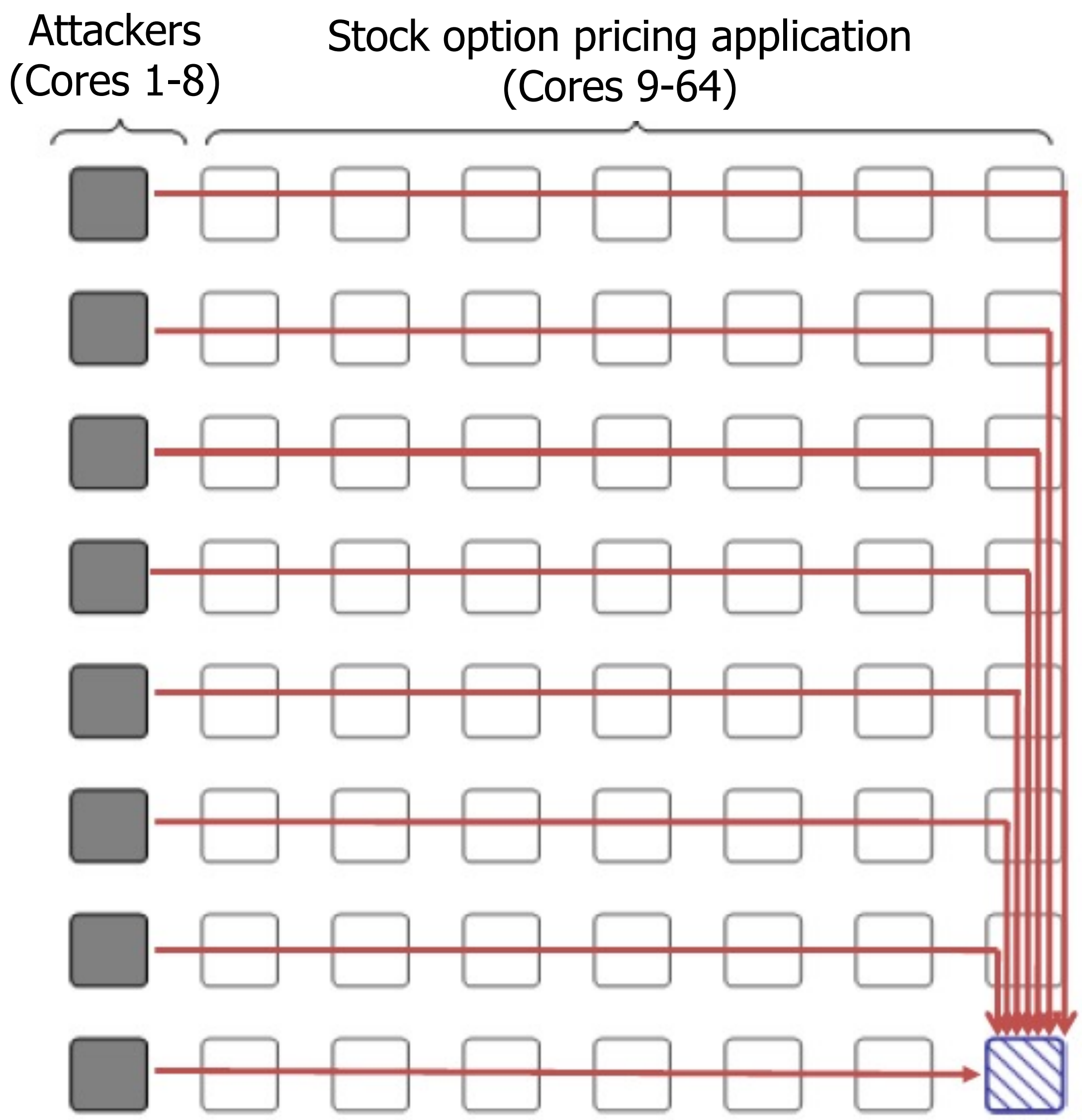
19.1K subscribers

https://www.youtube.com/watch?v=Axye9VqQT7w&list=PL5Q2soXY2Zi9xidyIgBxUz7xRPS-wisBN&index=26

Distributed DoS in Networked Multi-Core Systems

Cores connected via packet-switched routers on chip

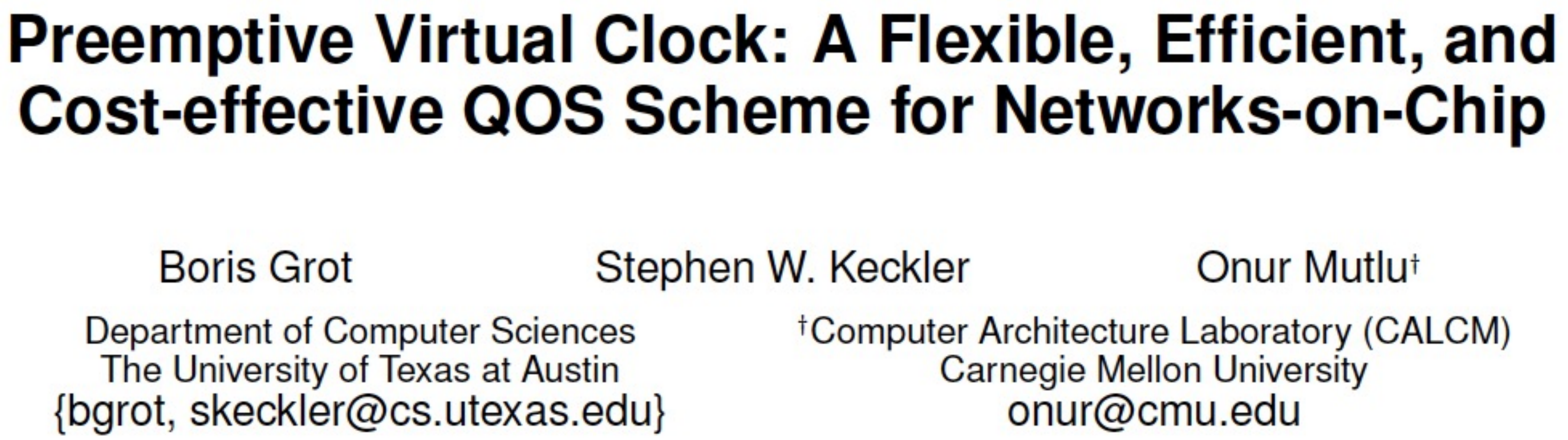
~5000X latency increase



Grot, Hestness, Keckler, Mutlu, "Preemptive virtual clock: A Flexible, Efficient, and Cost-effective QOS Scheme for Networks-on-Chip," MICRO 2009.

More on Interconnect Based Starvation

- Boris Grot, Stephen W. Keckler, and Onur Mutlu, **"Preemptive Virtual Clock: A Flexible, Efficient, and Cost-effective QOS Scheme for Networks-on-Chip"** *Proceedings of the 42nd International Symposium on Microarchitecture* (***MICRO***), pages 268-279, New York, NY, December 2009. Slides (pdf)



Preemptive Virtual Clock: A Flexible, Efficient, and Cost-effective QOS Scheme for Networks-on-Chip

Boris Grot   Stephen W. Keckler   Onur Mutlu[†]

Department of Computer Sciences
The University of Texas at Austin
{bgrot, skeckler@cs.utexas.edu}

[†]Computer Architecture Laboratory (CALCM)
Carnegie Mellon University
onur@cmu.edu

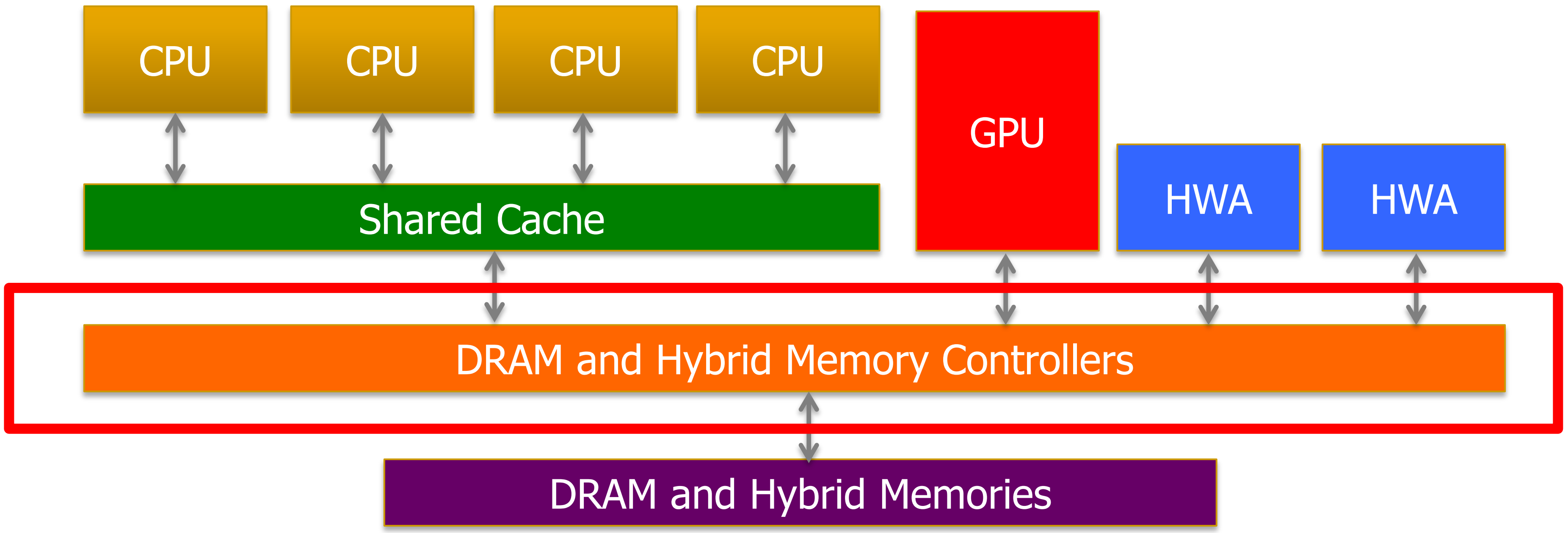
Takeaway

Breaking the abstraction layers (between components and transformation hierarchy levels)

and knowing what is underneath

enables you to **understand** and **solve** problems

Memory Control is Getting More Complex



- Heterogeneous agents: CPUs, GPUs, and HWAs
- Main memory interference between CPUs, GPUs, HWAs

Many goals, many constraints, many metrics …

Computer Architecture
Lecture 3c:
Memory Performance Attacks

Prof. Onur Mutlu
ETH Zürich
Fall 2021
7 October 2021