Digital Design & Computer Arch.

Lecture 24a: Multi-Core Caches

Prof. Onur Mutlu

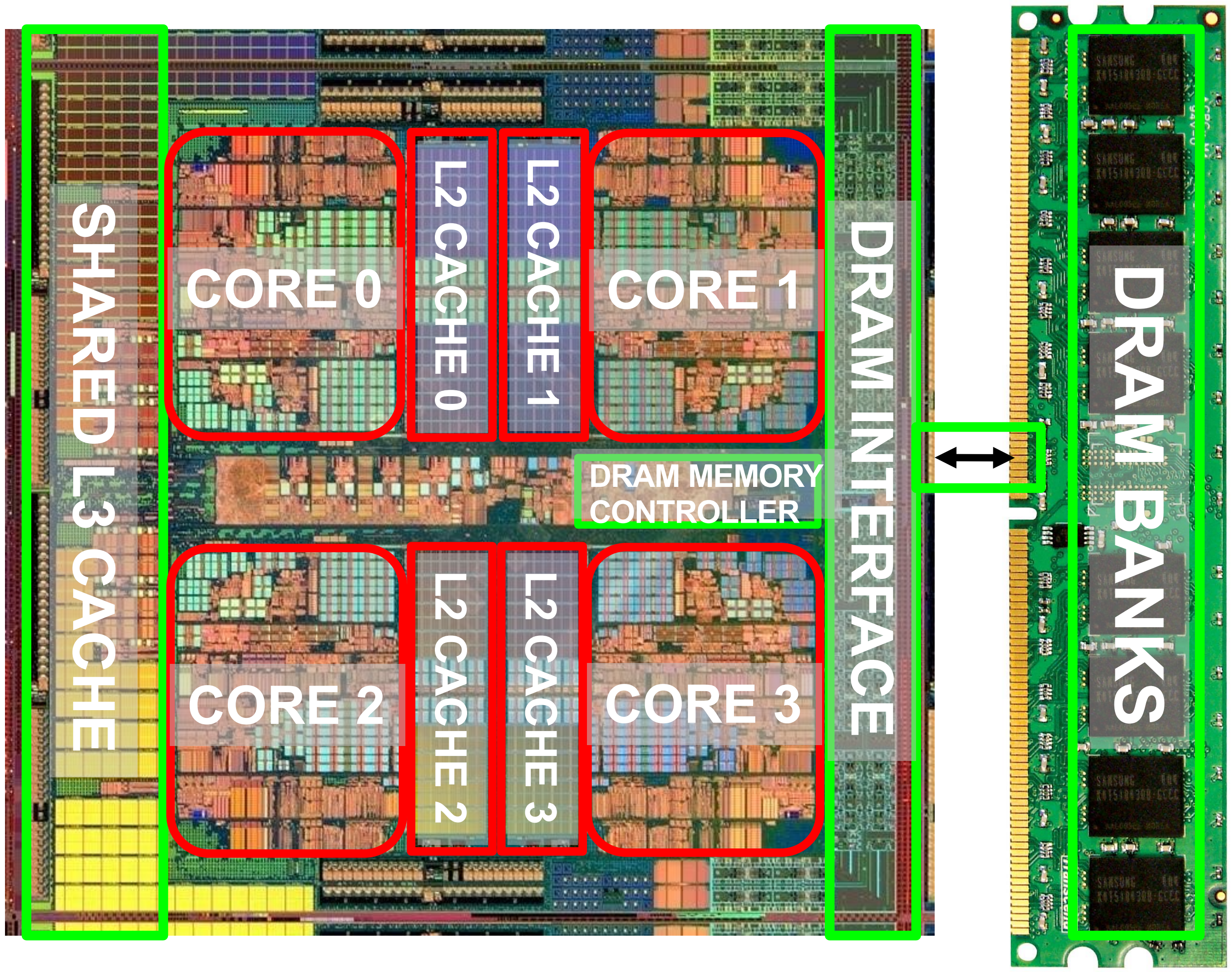
ETH Zürich

Spring 2022

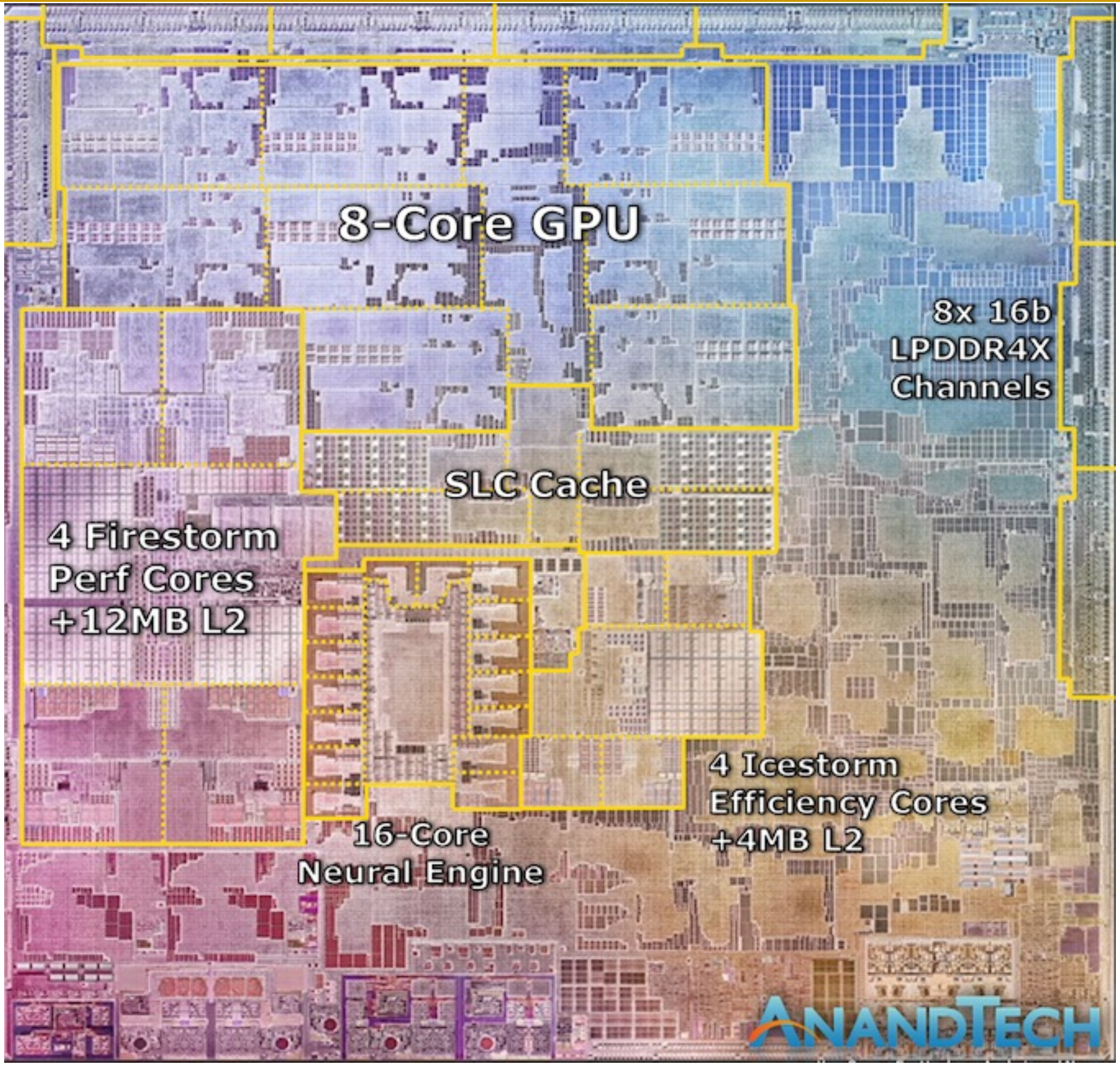
27 May 2022

Multi-Core Issues in Caching

Caches in a Multi-Core System



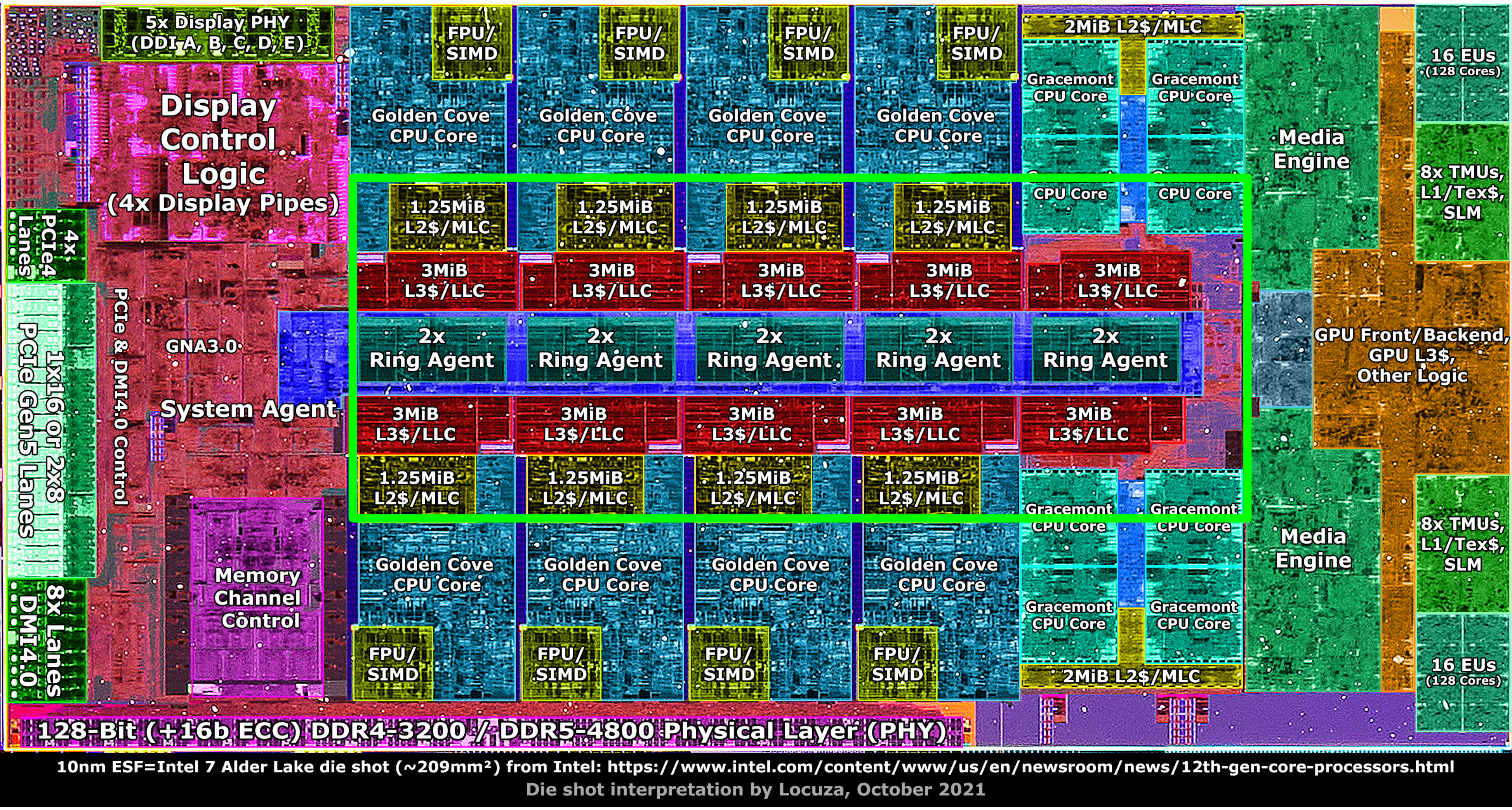
Caches in a Multi-Core System



Apple M1, 2021

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

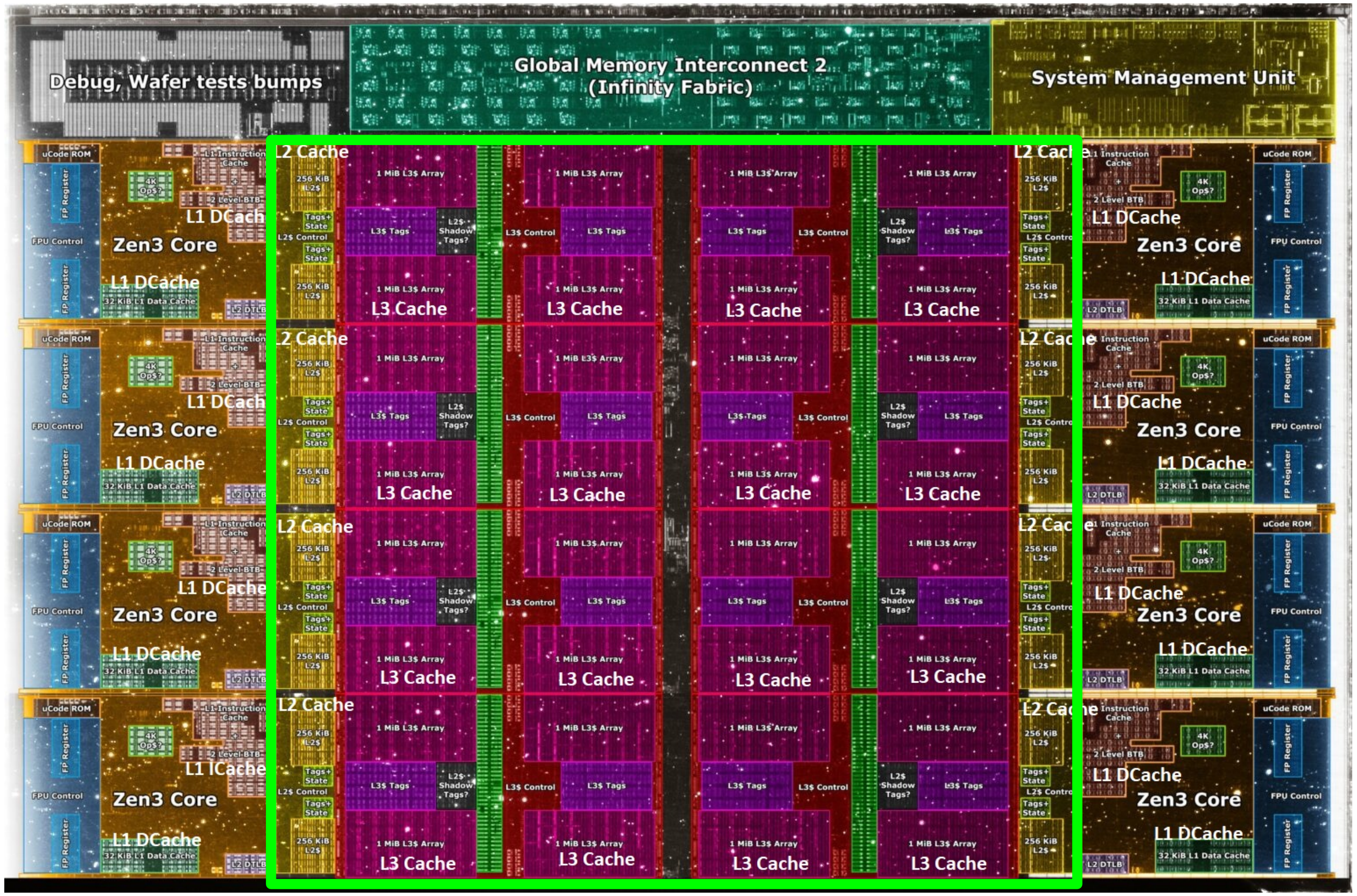
Caches in a Multi-Core System


5x Display PHY (DDI A, B, C, D, E)
Display Control Logic (4x Display Pipes)
4x PCIe4 Lanes
1x16 or 2x8 PCIe Gen5 Lanes
PCIe & DMI4.0 Control
GNA3.0
System Agent
Memory Channel Control
8x Lanes DMI4.0
FPU/ SIMD
Golden Cove CPU Core
1.25MiB L2$/MLC
3MiB L3$/LLC
2x Ring Agent
2MiB L2$/MLC
Gracemont CPU Core
Media Engine
16 EUs (128 Cores)
8x TMUs, L1/Tex$, SLM
GPU Front/Backend, GPU L3$, Other Logic
128-Bit (+16b ECC) DDR4-3200 / DDR5-4800 Physical Layer (PHY)
10nm ESF=Intel 7 Alder Lake die shot (~209mm²) from Intel: https://www.intel.com/content/www/us/en/newsroom/news/12th-gen-core-processors.html
Die shot interpretation by Locuza, October 2021

Intel Alder Lake, 2021

Source: https://twitter.com/Locuza_/status/1454152714930331652

Caches in a Multi-Core System



Core Count:
8 cores/16 threads

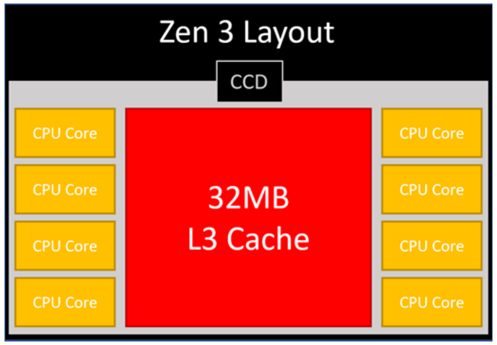
L1 Caches:
32 KB per core

L2 Caches:
512 KB per core

L3 Cache:
32 MB shared

AMD Ryzen 5000, 2020

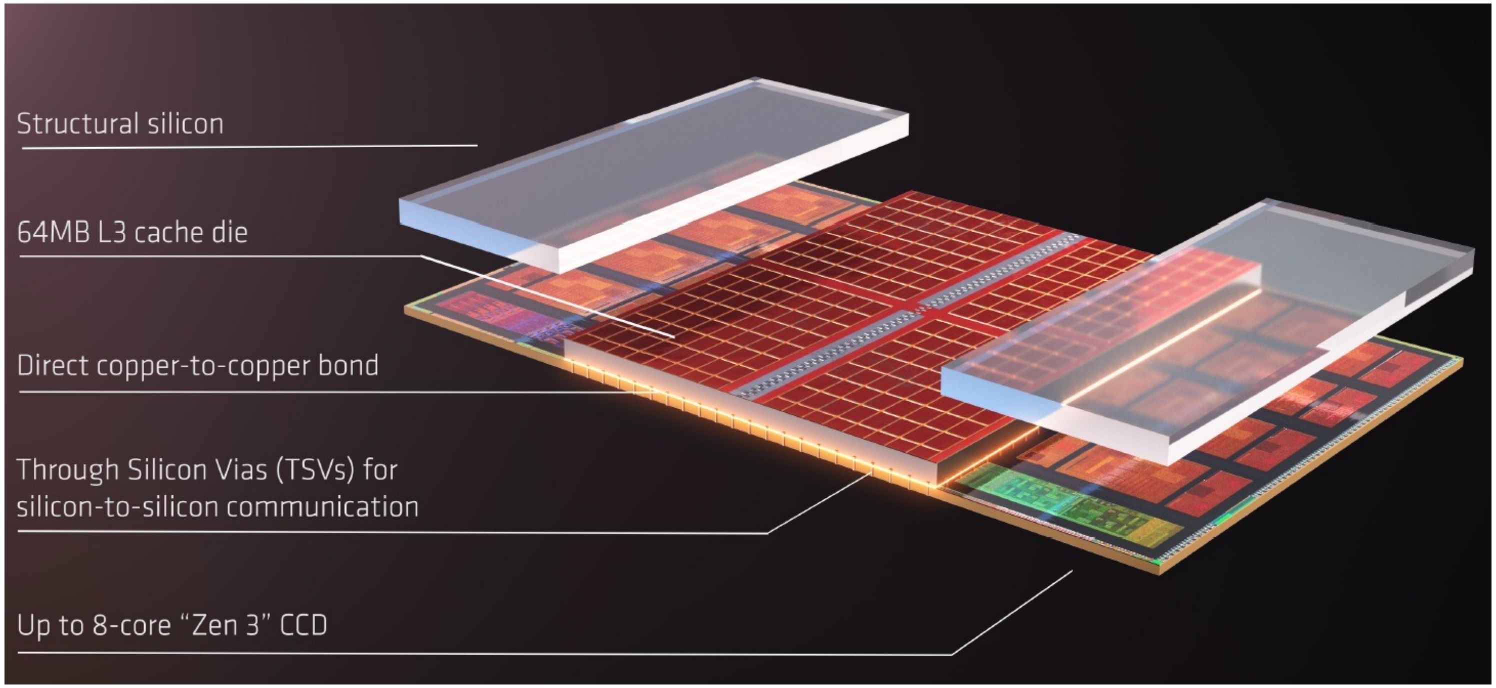
Caches in a Multi-Core System



https://community.microcenter.com/discussion/5134/comparing-zen-3-to-zen-2

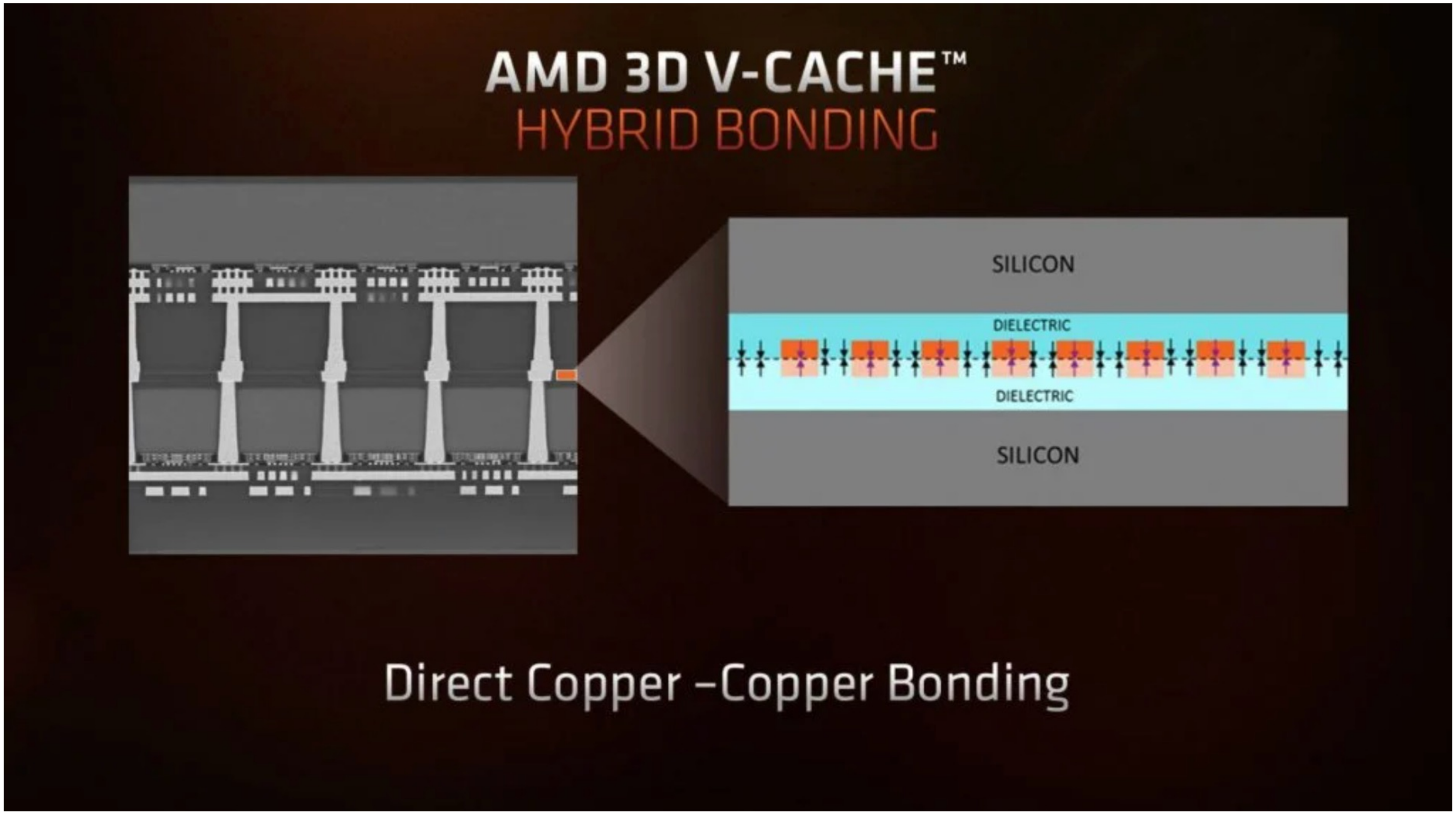
AMD increases the L3 size of their 8-core Zen 3 processors from 32 MB to 96 MB

Additional 64 MB L3 cache die stacked on top of the processor die
- Connected using Through Silicon Vias (TSVs)
- Total of 96 MB L3 cache


Structural silicon
64MB L3 cache die
Direct copper-to-copper bond
Through Silicon Vias (TSVs) for silicon-to-silicon communication
Up to 8-core "Zen 3" CCD

https://youtu.be/gqAYMx34euU
https://www.tech-critter.com/amd-keynote-computex-2021/

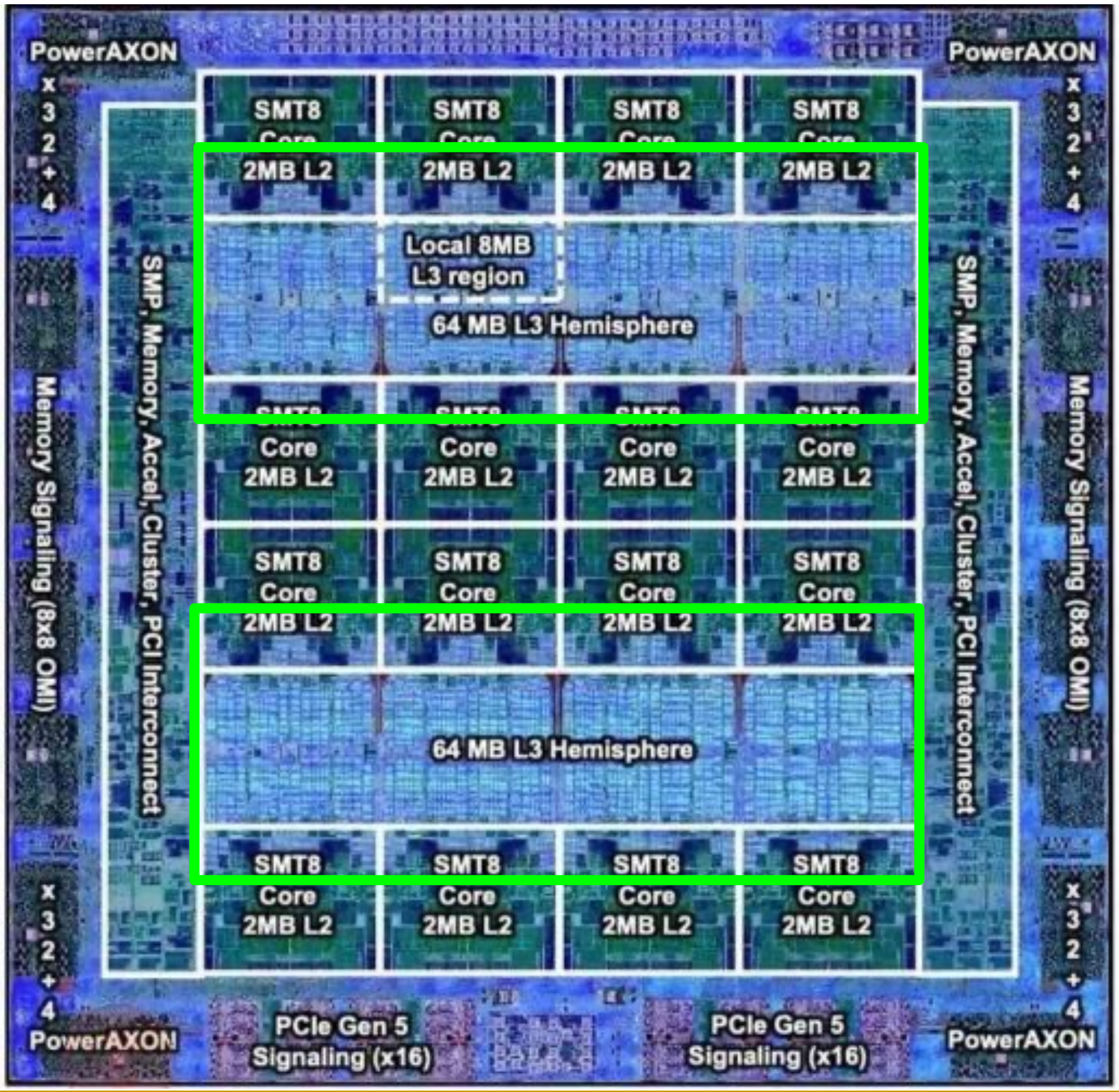
3D Stacking Technology: Example



AMD Ryzen 7 5800X3D: The 3D V-Cache in detail (4)
Source: AMD

https://www.pcgameshardware.de/Ryzen-7-5800X3D-CPU-278064/Specials/3D-V-Cache-Release-1393125/

Caches in a Multi-Core System



IBM POWER10, 2020

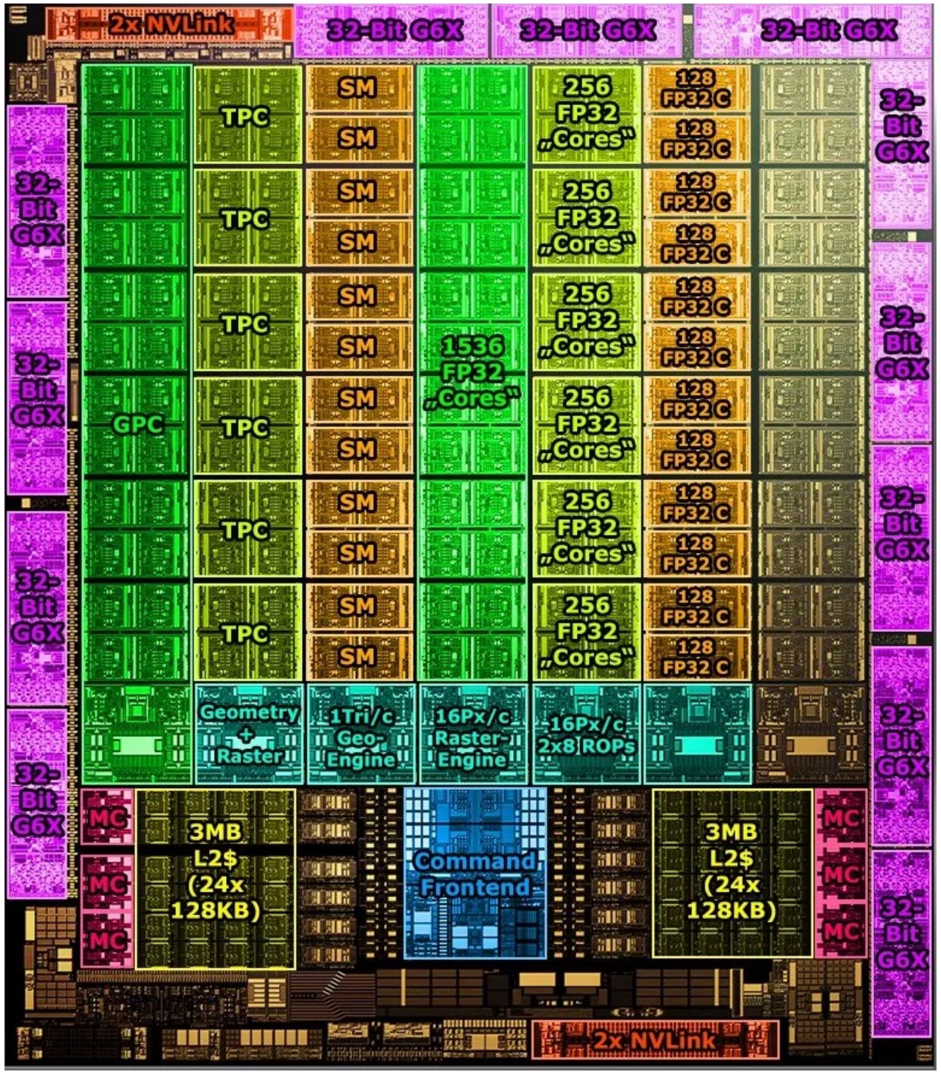
Cores:
15-16 cores,
8 threads/core

L2 Caches:
2 MB per core

L3 Cache:
120 MB shared

https://www.it-techblog.de/ibm-power10-prozessor-mehr-speicher-mehr-tempo-mehr-sicherheit/09/2020/

Caches in a Multi-Core System



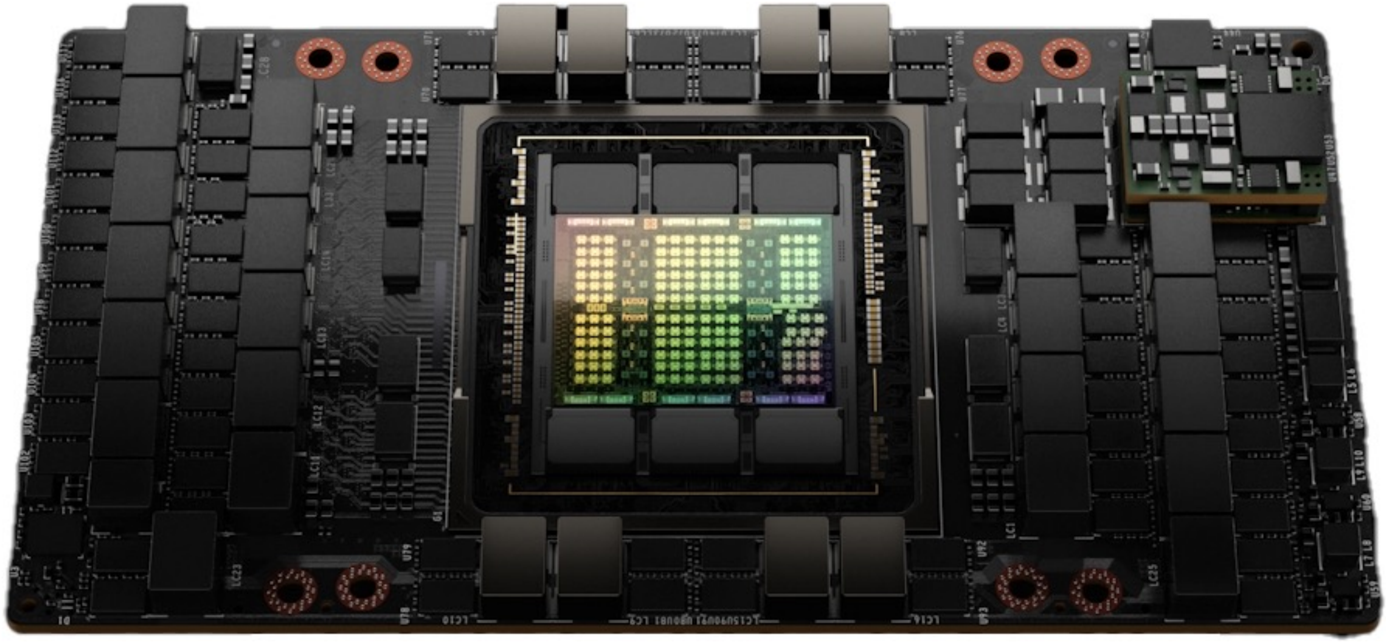
Nvidia Ampere, 2020

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

Caches in a Multi-Core System



Nvidia Hopper, 2022

Cores:

144 Streaming Multiprocessors

L1 Cache or Scratchpad:

256KB per SM

Can be used as L1 Cache and/or Scratchpad

L2 Cache:

60 MB shared

https://developer.nvidia.com/blog/nvidia-hopper-architecture-in-depth/

Caches in Multi-Core Systems

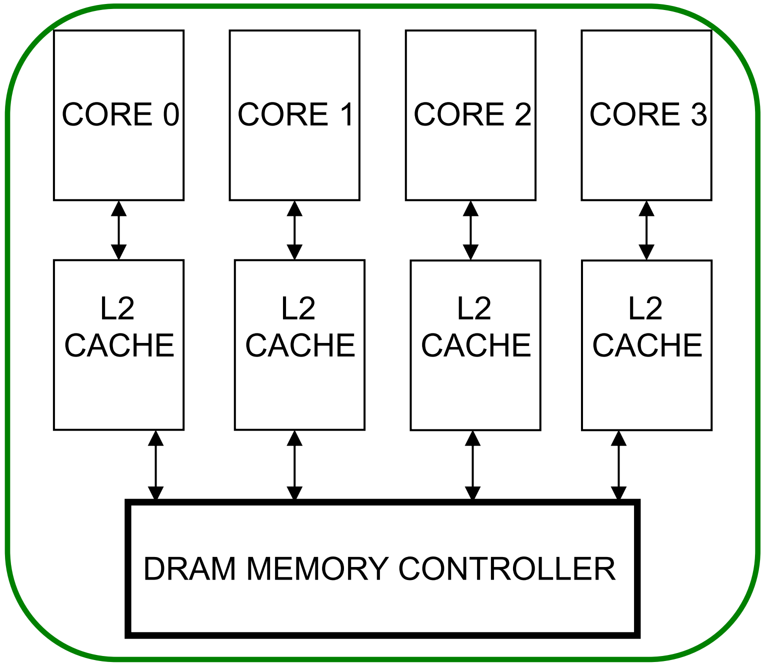
- Cache efficiency becomes even more important in a multi-core/multi-threaded system
  - Memory bandwidth is at premium
  - Cache space is a limited resource across cores/threads

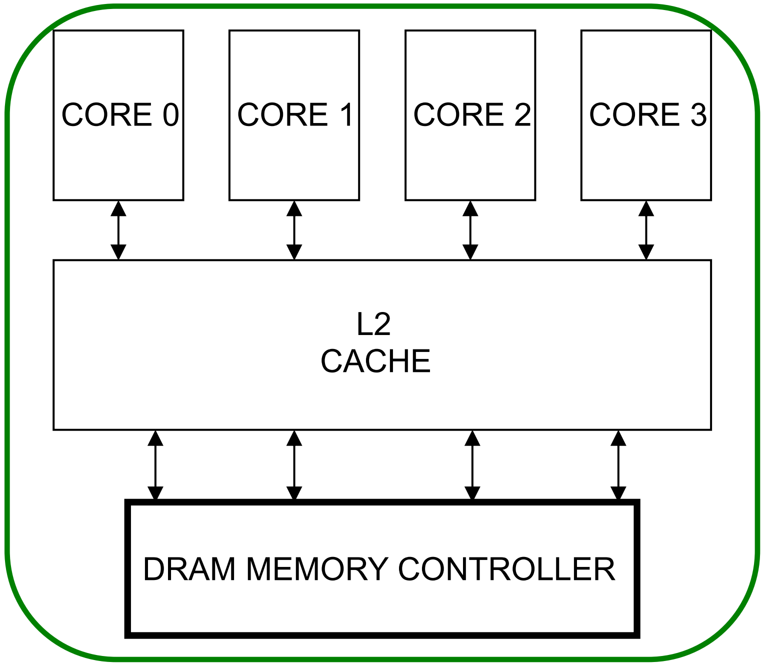
- How do we design the caches in a multi-core system?

- Many decisions and questions
  - Shared vs. private caches
  - How to maximize performance of the entire system?
  - How to provide QoS & predictable perf. to different threads in a shared cache?
  - Should cache management algorithms be aware of threads?
  - How should space be allocated to threads in a shared cache?
  - Should we store data in compressed format in some caches?
  - How do we do better reuse prediction & management in caches?

Private vs. Shared Caches

- Private cache: Cache belongs to one core (a shared block can be in multiple caches)
- Shared cache: Cache is shared by multiple cores





Resource Sharing Concept and Advantages

- Idea: Instead of dedicating a hardware resource to a hardware context, allow multiple contexts to use it
  - Example resources: functional units, pipeline, caches, buses, memory
- Why?

+ Resource sharing improves utilization/efficiency → throughput
  - When a resource is left idle by one thread, another thread can use it; no need to replicate shared data

+ Reduces communication latency
  - For example, data shared between multiple threads can be kept in the same cache in multithreaded processors

+ Compatible with the shared memory programming model

Resource Sharing Disadvantages

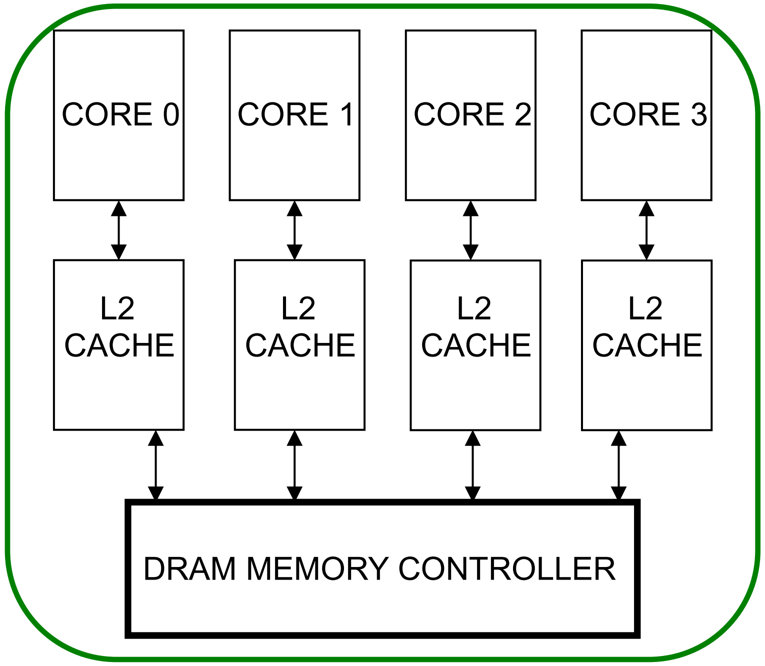
- Resource sharing results in contention for resources
  - When the resource is not idle, another thread cannot use it
  - If space is occupied by one thread, another thread needs to re-occupy it

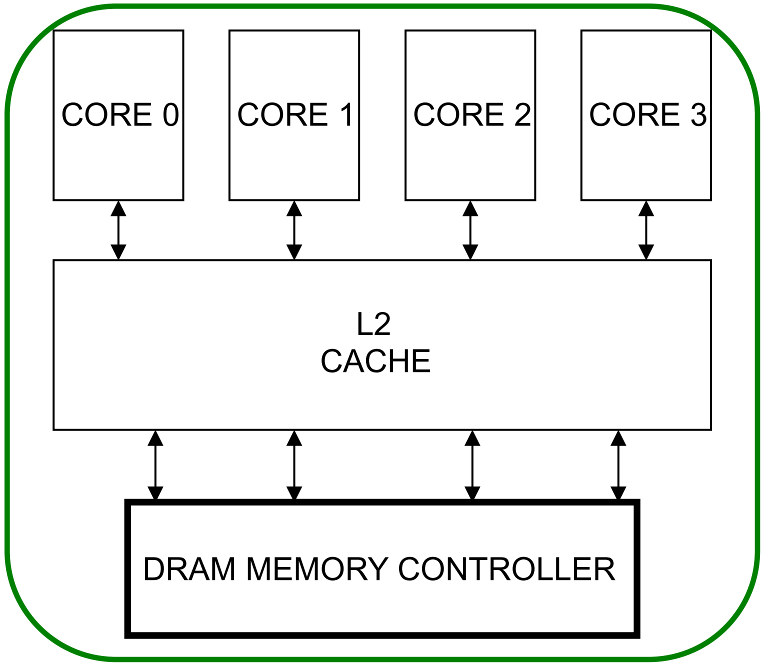
- Sometimes reduces each or some thread's performance
  - Thread performance can be worse than when it is run alone
- Eliminates performance isolation → inconsistent performance across runs
  - Thread performance depends on co-executing threads
- Uncontrolled (free-for-all) sharing degrades QoS
  - Causes unfairness, starvation

Need to efficiently and fairly utilize shared resources

Private vs. Shared Caches

- Private cache: Cache belongs to one core (a shared block can be in multiple caches)
- Shared cache: Cache is shared by multiple cores



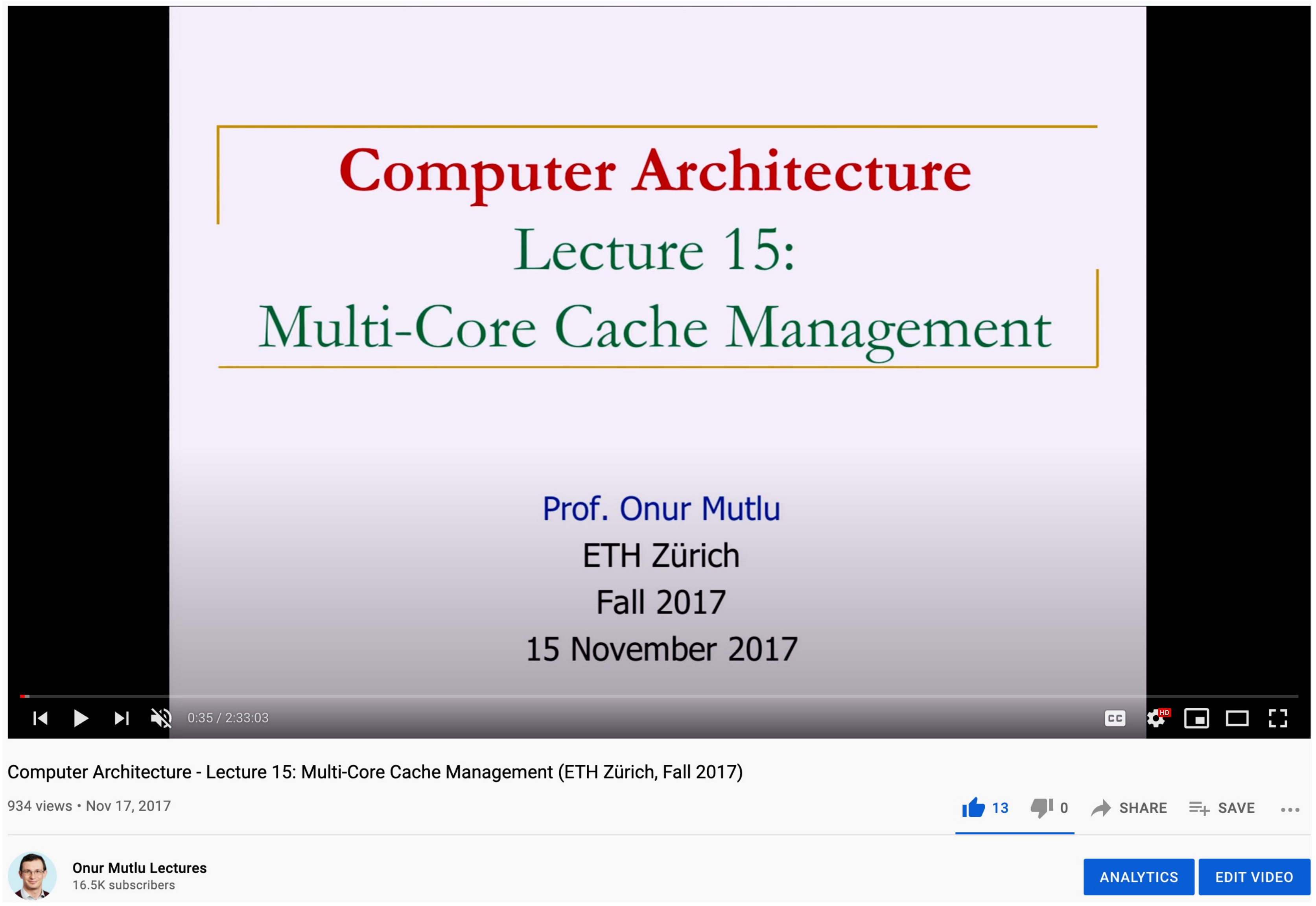


Shared Caches Between Cores

- Advantages:
  - High effective capacity
  - Dynamic partitioning of available cache space
    - No fragmentation due to static partitioning
    - If one core does not utilize some space, another core can
  - Easier to maintain coherence (a cache block is in a single location)

- Disadvantages
  - Slower access (cache not tightly coupled with the core)
  - Cores incur conflict misses due to other cores' accesses
    - Misses due to inter-core interference
    - Some cores can destroy the hit rate of other cores
  - Guaranteeing a minimum level of service (or fairness) to each core is harder (how much space, how much bandwidth?)

Lectures on Multi-Core Cache Management



Computer Architecture
Lecture 15:
Multi-Core Cache Management

Prof. Onur Mutlu
ETH Zürich
Fall 2017
15 November 2017

Computer Architecture - Lecture 15: Multi-Core Cache Management (ETH Zürich, Fall 2017)

934 views • Nov 17, 2017

https://www.youtube.com/watch?v=7_Tqlw8gxOU&list=PL5Q2soXY2Zi9OhoVQBXYFIZywZXCPl4M_&index=17

Lectures on Multi-Core Cache Management

Page Coloring
Physical memory divided into colors
Colors map to different cache sets
Cache partitioning
Ensure two threads are allocated pages of different colors
Memory page
Cache
Thread A
Thread B
54:39 / 1:07:14

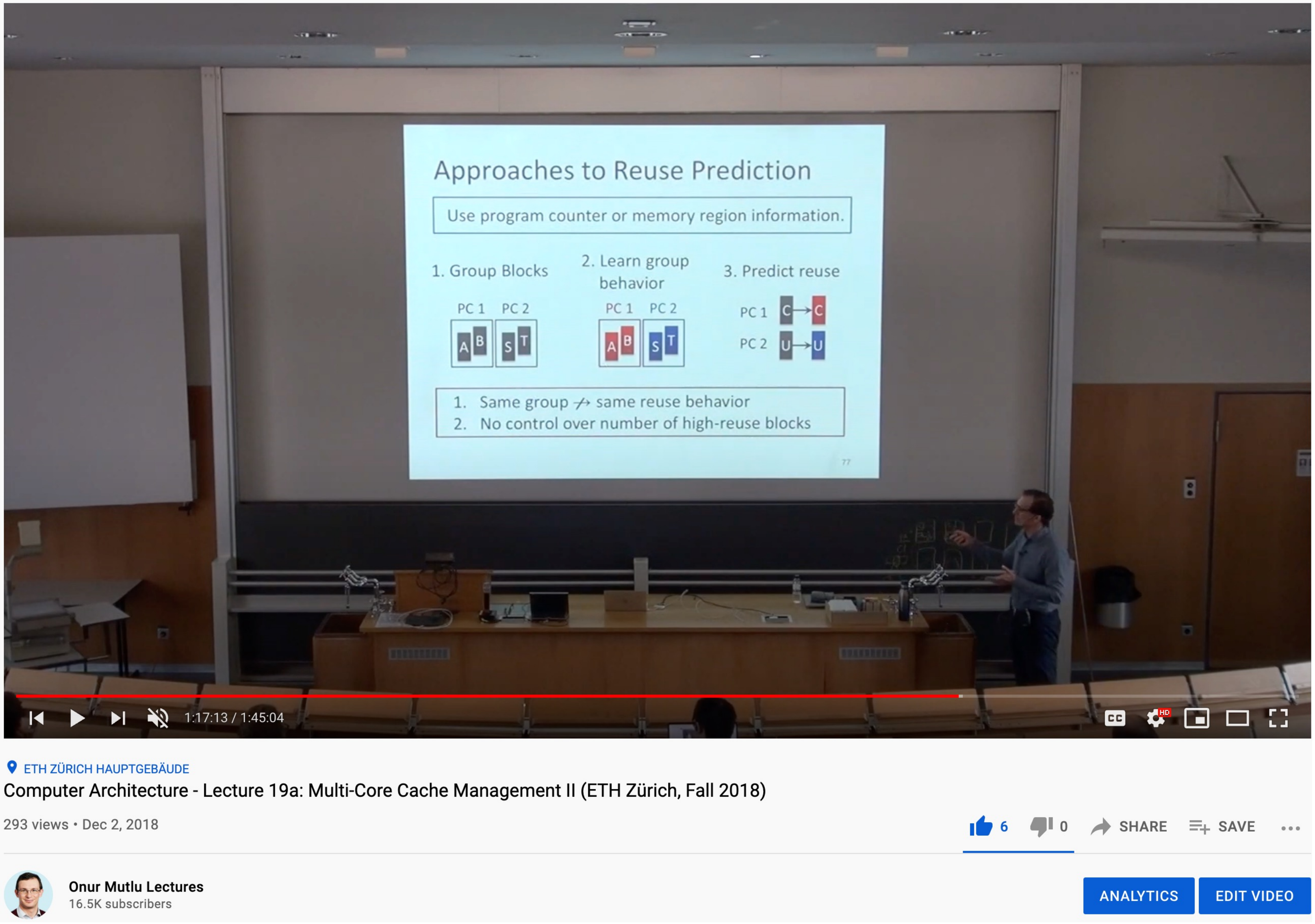
ETH ZÜRICH HAUPTGEBÄUDE

Computer Architecture - Lecture 18b: Multi-Core Cache Management (ETH Zürich, Fall 2018)

744 views • Nov 23, 2018

https://www.youtube.com/watch?v=c9FhGRB3HoA&list=PL5Q2soXY2Zi9JXe3ywQMhylk_d5dI-TM7&index=29

Lectures on Multi-Core Cache Management



Approaches to Reuse Prediction

Use program counter or memory region information.

1. Group Blocks
2. Learn group behavior
3. Predict reuse

1. Same group ↛ same reuse behavior
2. No control over number of high-reuse blocks

ETH ZÜRICH HAUPTGEBÄUDE

Computer Architecture - Lecture 19a: Multi-Core Cache Management II (ETH Zürich, Fall 2018)

293 views • Dec 2, 2018

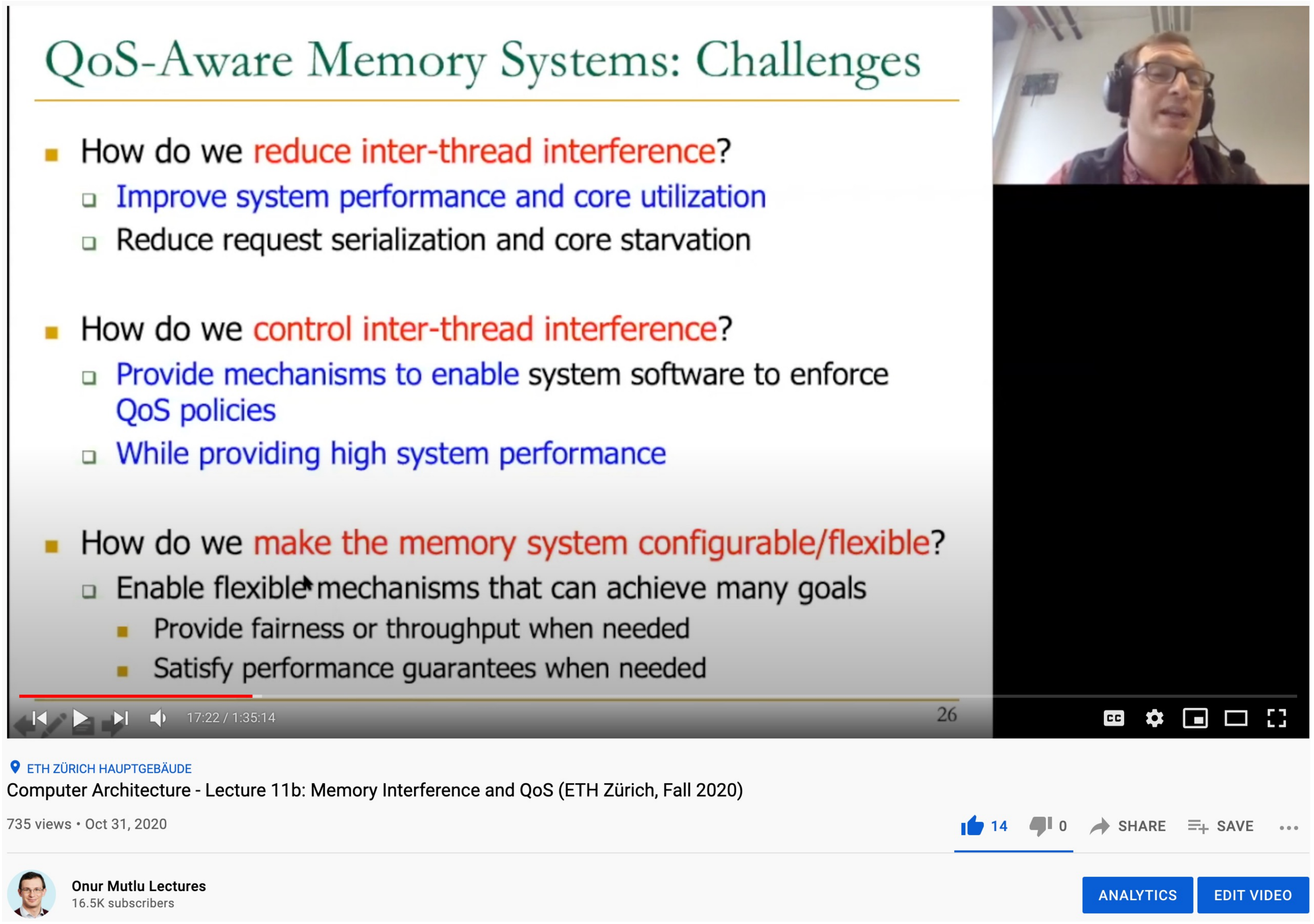
Onur Mutlu Lectures
16.5K subscribers

https://www.youtube.com/watch?v=Siz86__PD4w&list=PL5Q2soXY2Zi9JXe3ywQMhylk_d5dI-TM7&index=30

Lectures on Multi-Core Cache Management

- Computer Architecture, Fall 2018, Lecture 18b
  - Multi-Core Cache Management (ETH, Fall 2018)
  - https://www.youtube.com/watch?v=c9FhGRB3HoA&list=PL5Q2soXY2Zi9JXe3ywQMhylk_d5dI-TM7&index=29
- Computer Architecture, Fall 2018, Lecture 19a
  - Multi-Core Cache Management II (ETH, Fall 2018)
  - https://www.youtube.com/watch?v=Siz86__PD4w&list=PL5Q2soXY2Zi9JXe3ywQMhylk_d5dI-TM7&index=30
- Computer Architecture, Fall 2017, Lecture 15
  - Multi-Core Cache Management (ETH, Fall 2017)
  - https://www.youtube.com/watch?v=7_Tqlw8gxOU&list=PL5Q2soXY2Zi9OhoVQBXYFIZywZXCPl4M_&index=17

Lectures on Memory Resource Management



QoS-Aware Memory Systems: Challenges

- How do we reduce inter-thread interference?
  - Improve system performance and core utilization
  - Reduce request serialization and core starvation
- How do we control inter-thread interference?
  - Provide mechanisms to enable system software to enforce QoS policies
  - While providing high system performance
- How do we make the memory system configurable/flexible?
  - Enable flexible mechanisms that can achieve many goals
    - Provide fairness or throughput when needed
    - Satisfy performance guarantees when needed

ETH ZÜRICH HAUPTGEBÄUDE

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

735 views • Oct 31, 2020

Onur Mutlu Lectures
16.5K subscribers

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

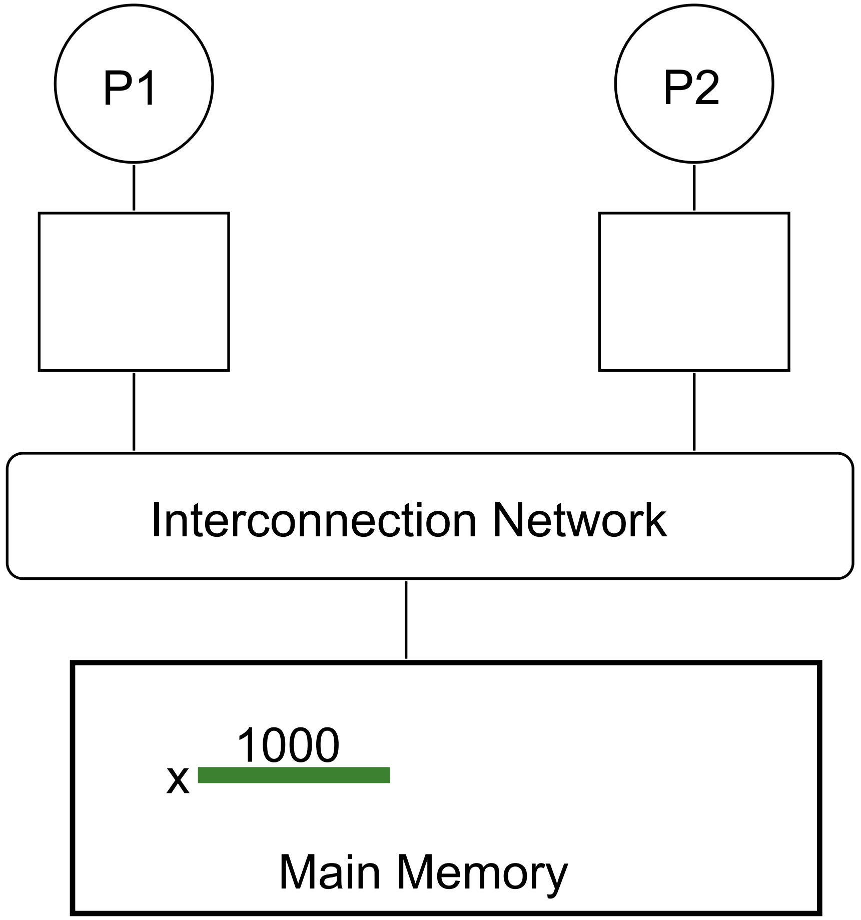
Lectures on Memory Resource Management

- Computer Architecture, Fall 2020, Lecture 11a
  - Memory Controllers (ETH, Fall 2020)
  - https://www.youtube.com/watch?v=TeG773OgiMQ&list=PL5Q2soXY2Zi9xidyIgBxUz7xRPS-wisBN&index=20
- Computer Architecture, Fall 2020, Lecture 11b
  - Memory Interference and QoS (ETH, Fall 2020)
  - https://www.youtube.com/watch?v=0nnI807nCkc&list=PL5Q2soXY2Zi9xidyIgBxUz7xRPS-wisBN&index=21
- Computer Architecture, Fall 2020, Lecture 13
  - Memory Interference and QoS II (ETH, Fall 2020)
  - https://www.youtube.com/watch?v=Axye9VqQT7w&list=PL5Q2soXY2Zi9xidyIgBxUz7xRPS-wisBN&index=26
- Computer Architecture, Fall 2020, Lecture 2a
  - Memory Performance Attacks (ETH, Fall 2020)
  - https://www.youtube.com/watch?v=VJzZbwgBfy8&list=PL5Q2soXY2Zi9xidyIgBxUz7xRPS-wisBN&index=2

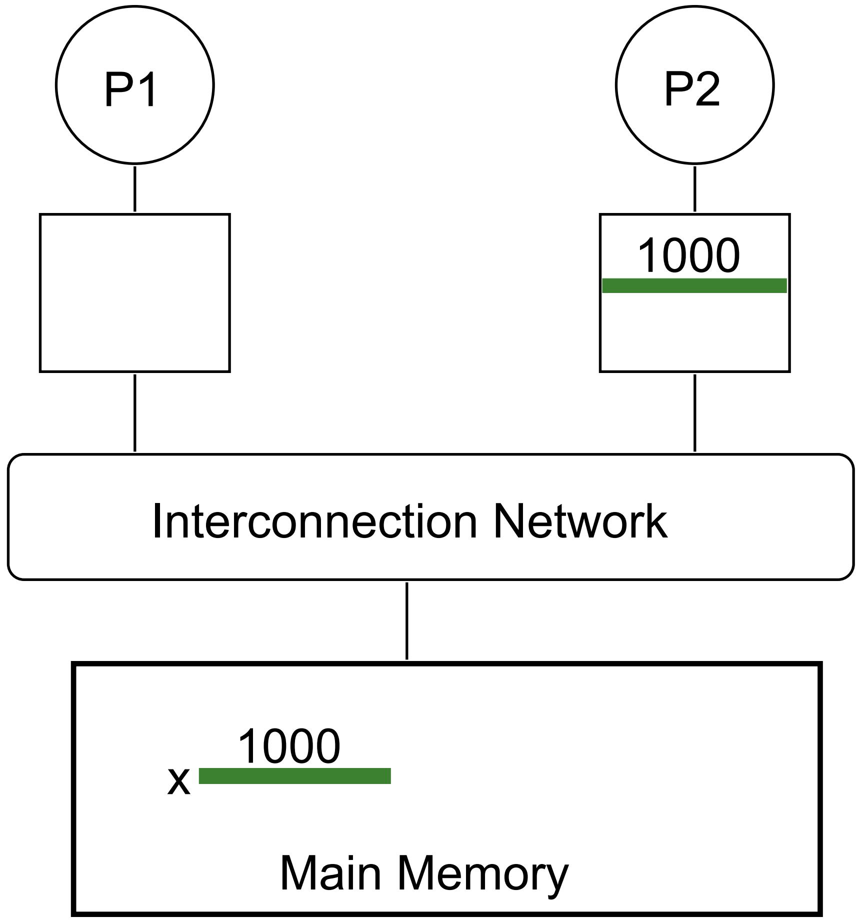
Cache Coherence

Cache Coherence

- Basic question: If multiple processors cache the same block, how do they ensure they all see a consistent state?

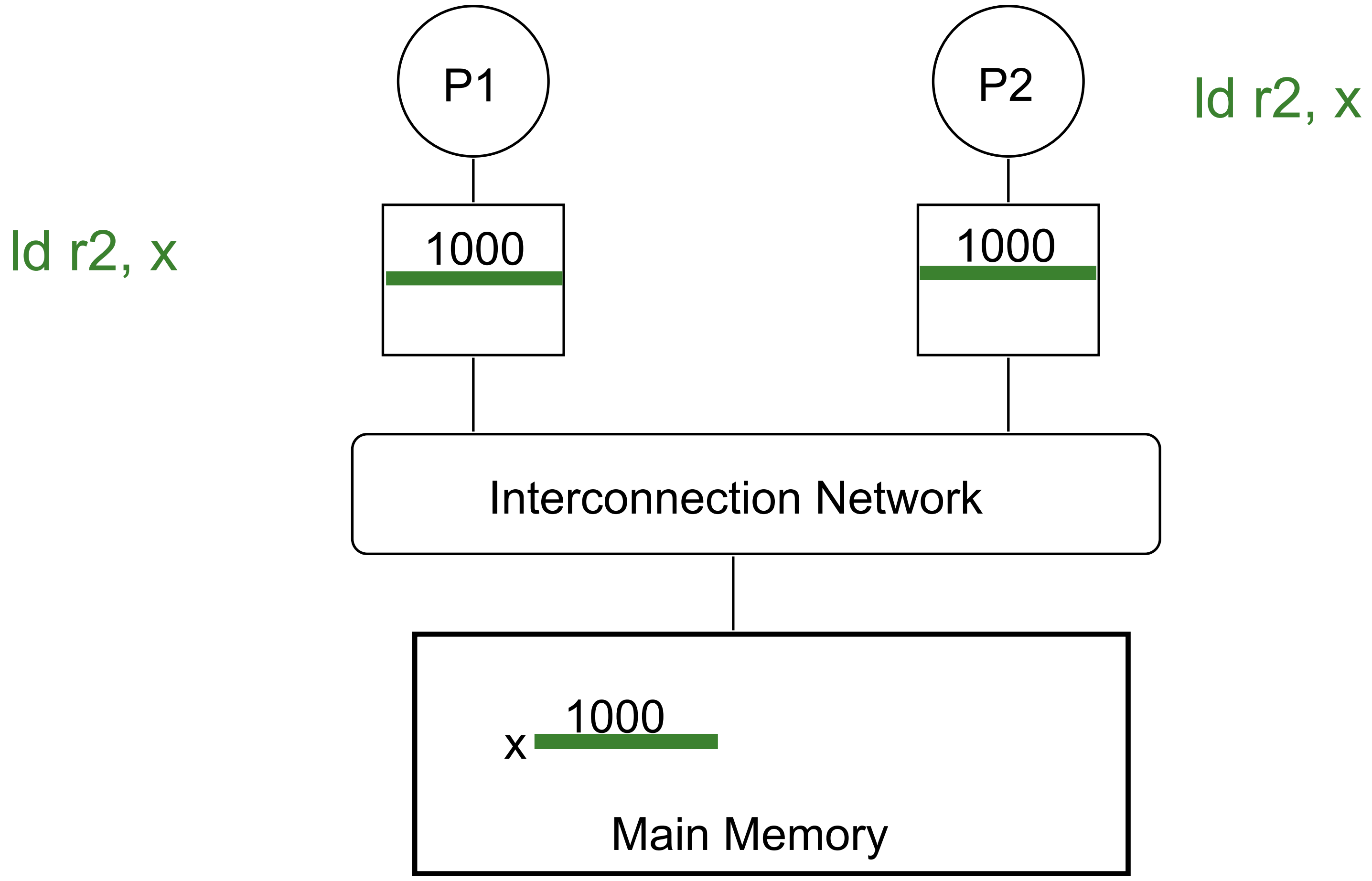


The Cache Coherence Problem

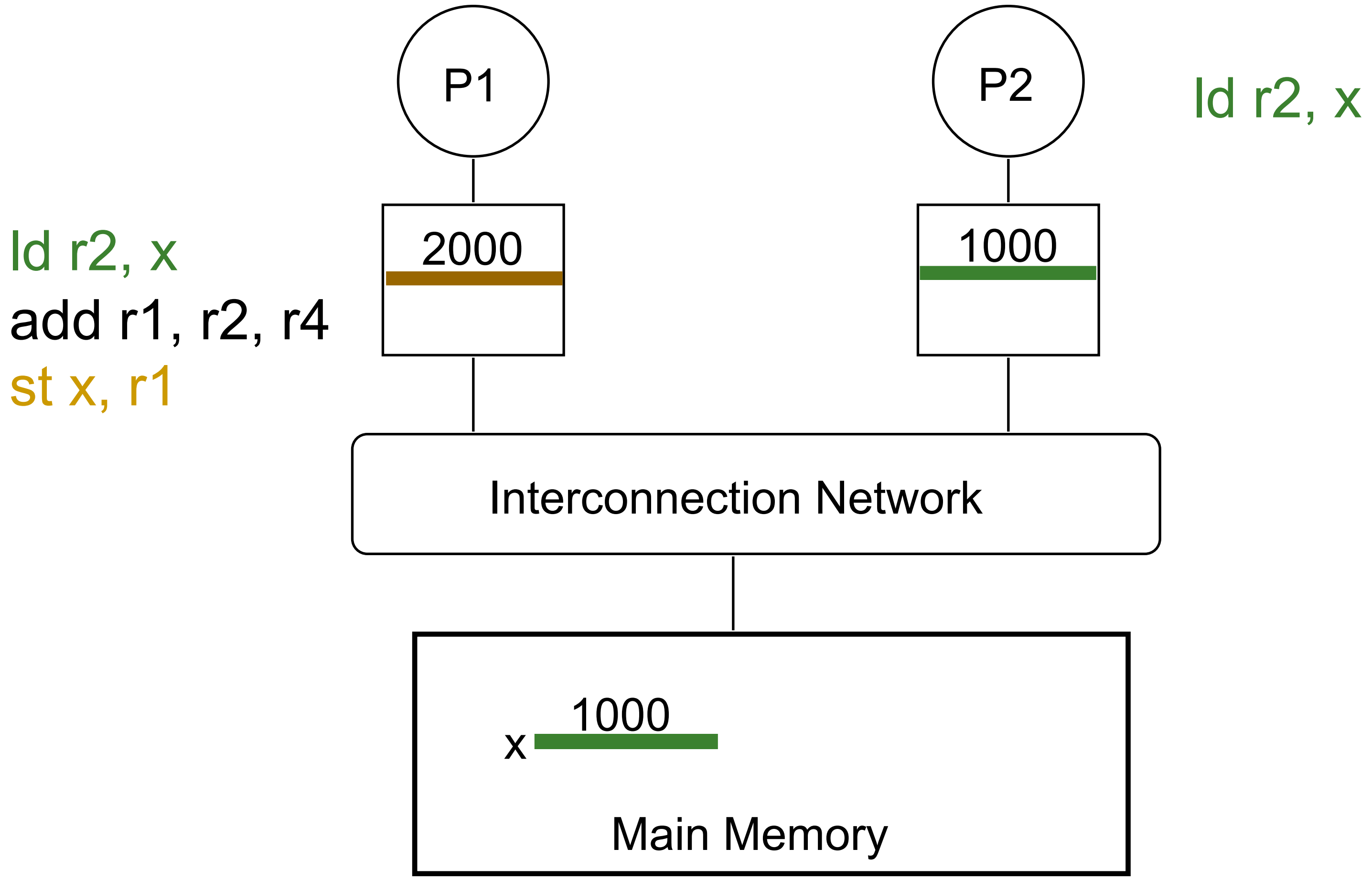


ld r2, x

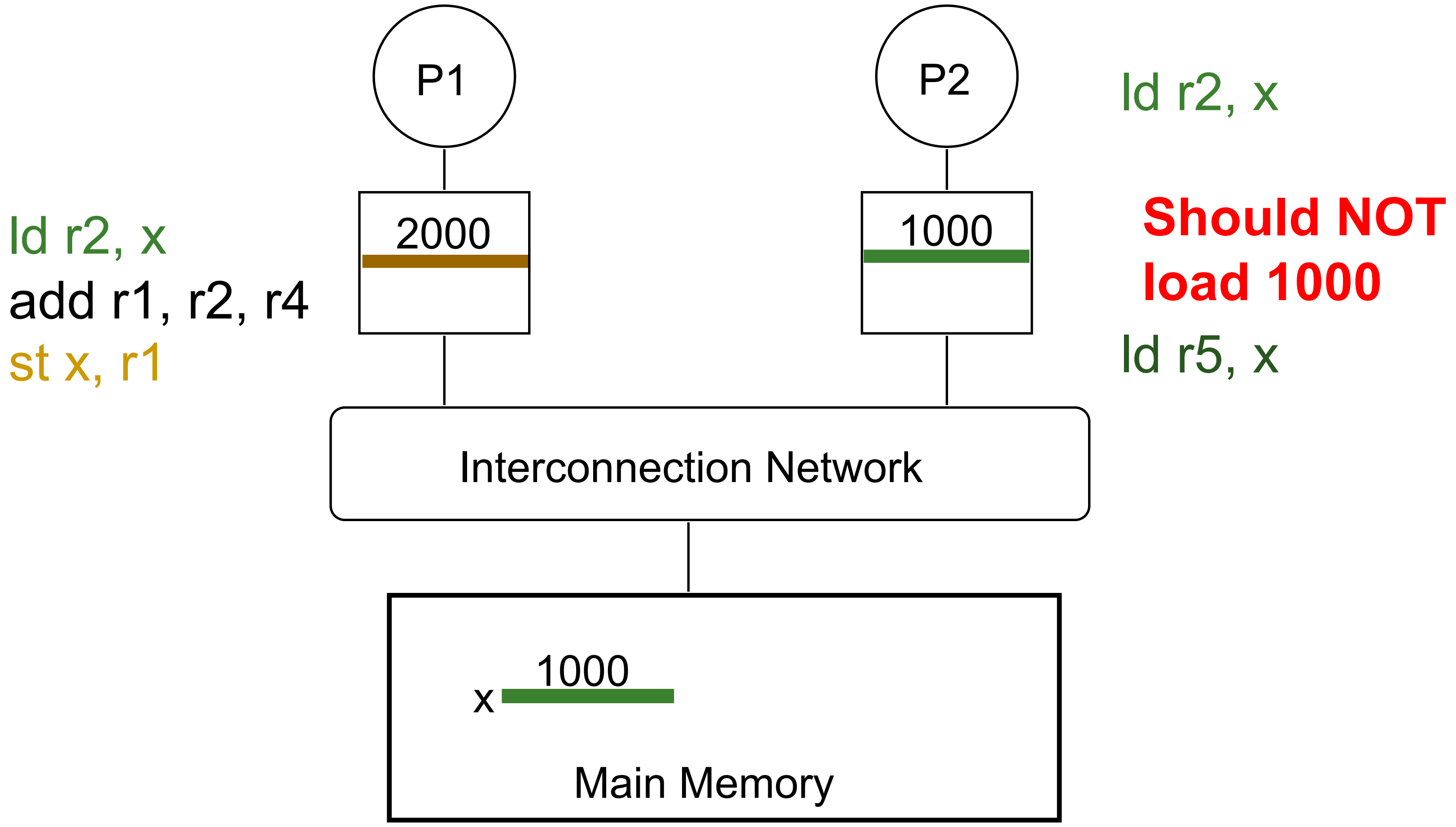
The Cache Coherence Problem



The Cache Coherence Problem

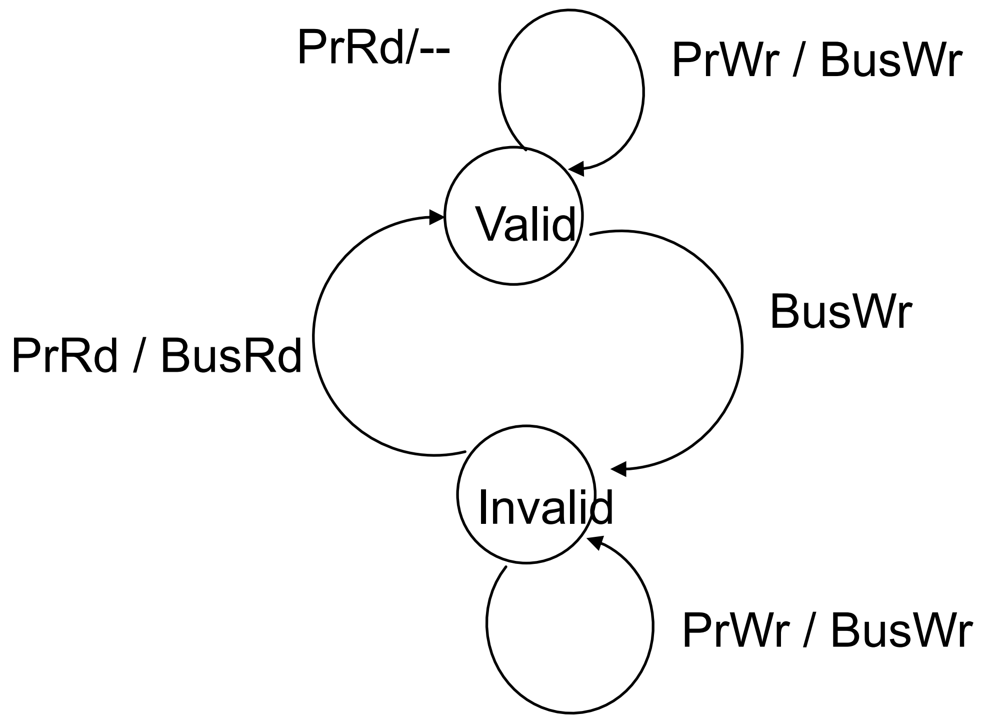


The Cache Coherence Problem



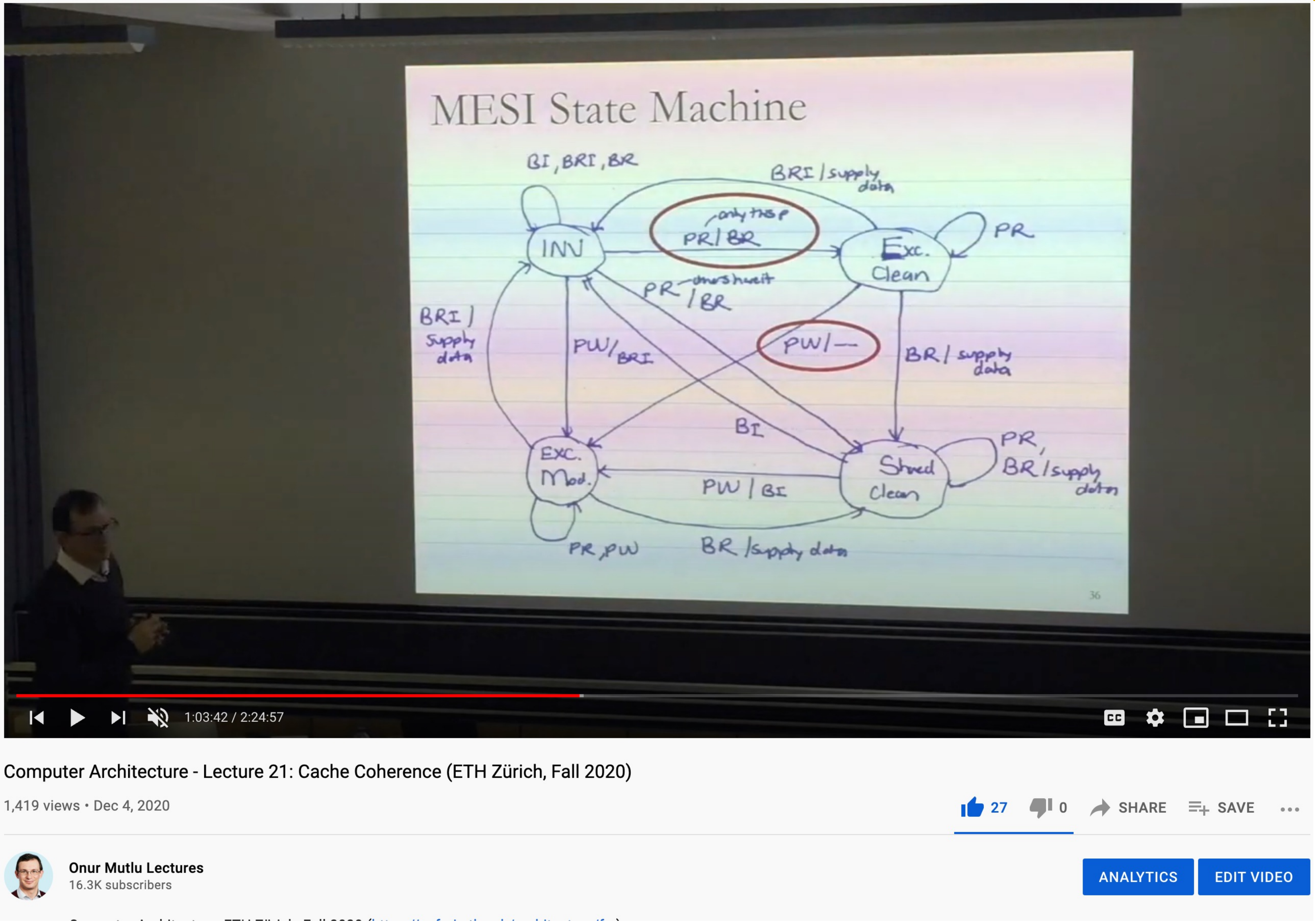
A Very Simple Coherence Scheme (VI)

- Idea: All caches "snoop" (observe) each other's write/read operations. If a processor writes to a block, all others invalidate the block.
- A simple protocol:



- Write-through, no-write-allocate cache
- Actions of the local processor on the cache block: PrRd, PrWr,
- Actions that are broadcast on the bus for the block: BusRd, BusWr

Lecture on Cache Coherence



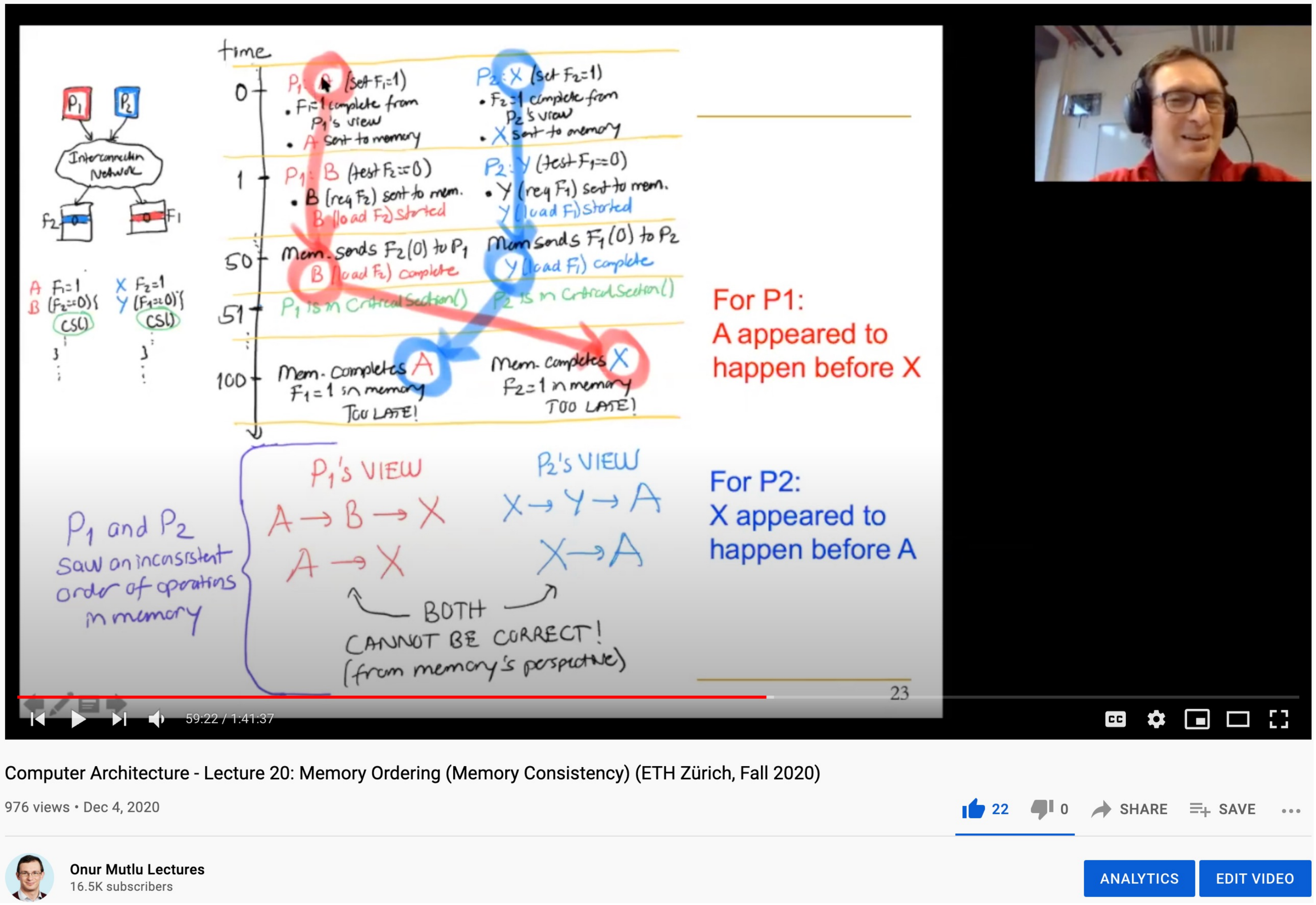
Computer Architecture - Lecture 21: Cache Coherence (ETH Zürich, Fall 2020)

1,419 views • Dec 4, 2020

Onur Mutlu Lectures
16.3K subscribers

https://www.youtube.com/watch?v=T9WlyezeaII&list=PL5Q2soXY2Zi9xidyIgBxUz7xRPS-wisBN&index=38

Lecture on Memory Ordering & Consistency



For P1:
A appeared to happen before X

For P2:
X appeared to happen before A

Computer Architecture - Lecture 20: Memory Ordering (Memory Consistency) (ETH Zürich, Fall 2020)

976 views • Dec 4, 2020

22 0 SHARE SAVE

Onur Mutlu Lectures
16.5K subscribers

ANALYTICS EDIT VIDEO

https://www.youtube.com/watch?v=Suy09mzTbiQ&list=PL5Q2soXY2Zi9xidyIgBxUz7xRPS-wisBN&index=37

Lecture on Cache Coherence & Consistency

- Computer Architecture, Fall 2020, Lecture 21
  - Cache Coherence (ETH, Fall 2020)
  - https://www.youtube.com/watch?v=T9WlyezeaII&list=PL5Q2soXY2Zi9xidyIgBxUz7xRPS-wisBN&index=38
- Computer Architecture, Fall 2020, Lecture 20
  - Memory Ordering & Consistency (ETH, Fall 2020)
  - https://www.youtube.com/watch?v=Suy09mzTbiQ&list=PL5Q2soXY2Zi9xidyIgBxUz7xRPS-wisBN&index=37
- Computer Architecture, Spring 2015, Lecture 28
  - Memory Consistency & Cache Coherence (CMU, Spring 2015)
  - https://www.youtube.com/watch?v=JfjT1a0vi4E&list=PL5PHm2jkkXmi5CxxI7b3JCL1TWybTDtKq&index=32
- Computer Architecture, Spring 2015, Lecture 29
  - Cache Coherence (CMU, Spring 2015)
  - https://www.youtube.com/watch?v=X6DZchnMYcw&list=PL5PHm2jkkXmi5CxxI7b3JCL1TWybTDtKq&index=33

Prefetching

Digital Design & Computer Arch.

Lecture 24a: Multi-Core Caches

Prof. Onur Mutlu

ETH Zürich
Spring 2022
27 May 2022

These Slides Are for You to Study. Next Lecture May Cover Them.

Prefetching

- Idea: Fetch the data before it is needed (i.e. pre-fetch) by the program

- Why?
  - Memory latency is high. If we can prefetch accurately and early enough we can reduce/eliminate that latency.
  - Can eliminate compulsory cache misses
  - Can it eliminate all cache misses? Capacity, conflict?

- Involves predicting which address will be needed in the future
  - Works if programs have predictable miss address patterns

Prefetching and Correctness

- Does a misprediction in prefetching affect correctness?

- No, prefetched data at a “mispredicted” address is simply not used
- There is no need for state recovery
  - In contrast to branch misprediction or value misprediction

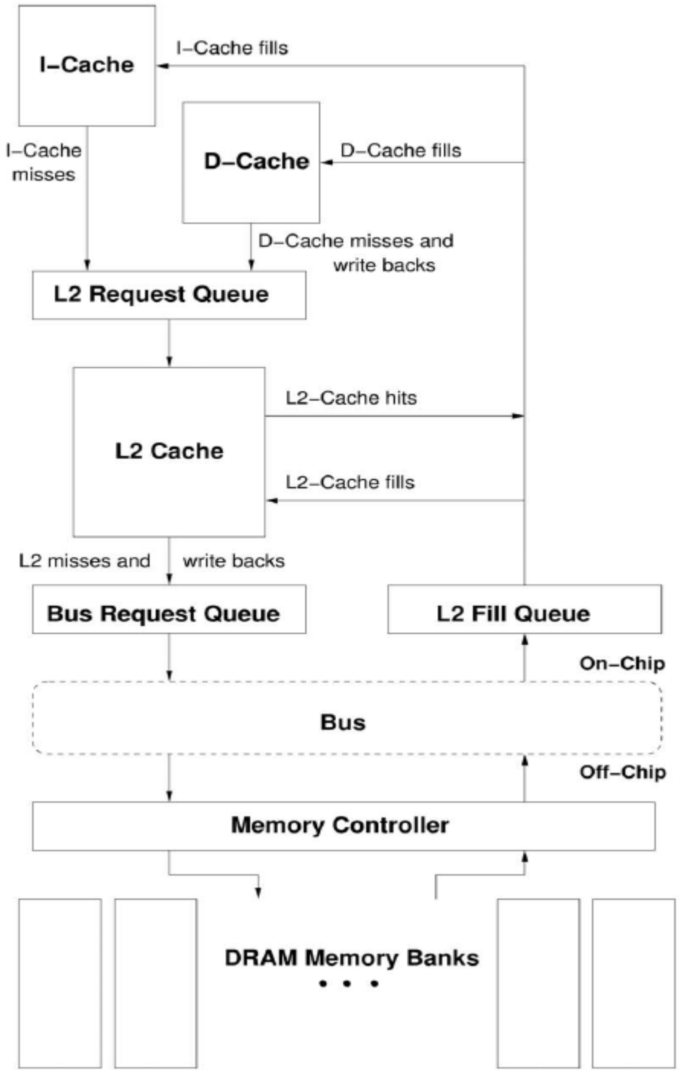
Basics

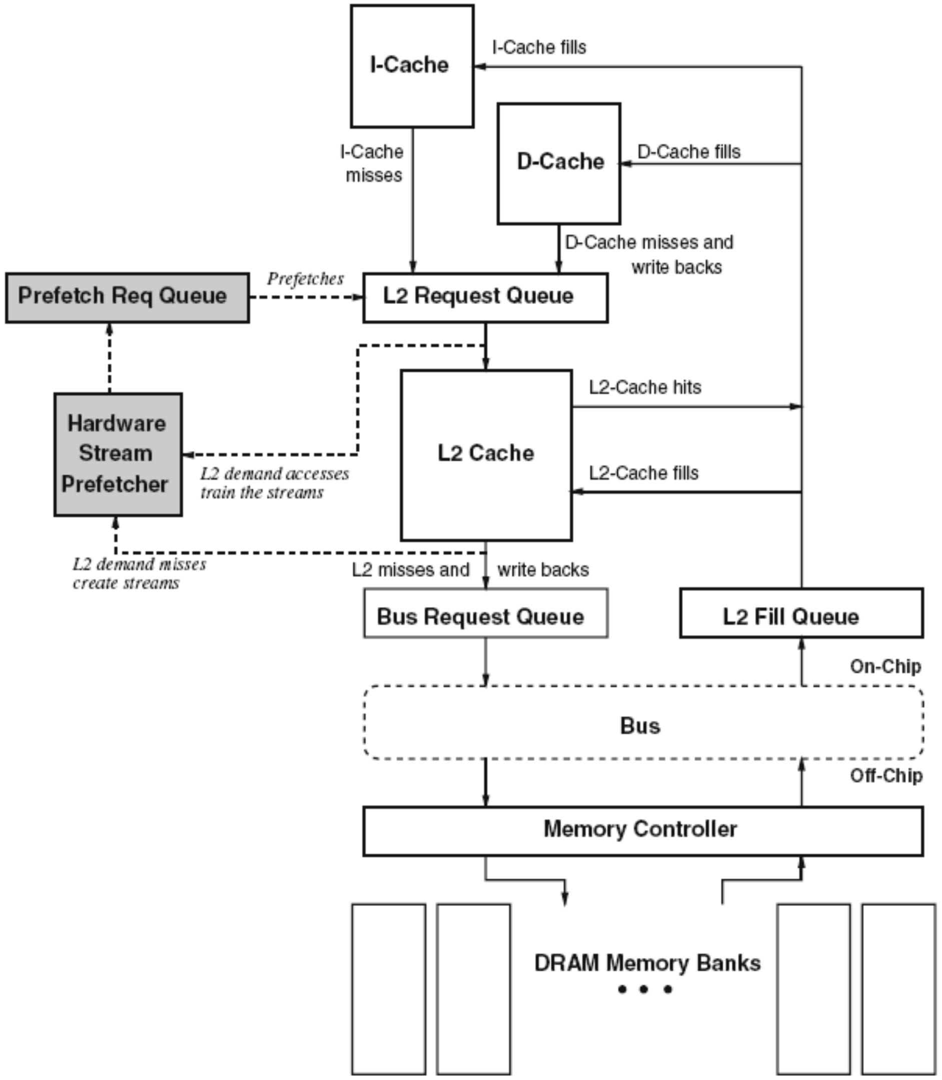
- In modern systems, prefetching is usually done in cache block granularity

- Prefetching is a technique that can reduce both
  - Miss rate
  - Miss latency

- Prefetching can be done by
  - Hardware
  - Compiler
  - Programmer
  - System

How a HW Prefetcher Fits in the Memory System





Prefetching: The Four Questions

- What
  - What addresses to prefetch (i.e., address prediction algorithm)
- When
  - When to initiate a prefetch request (early, late, on time)
- Where
  - Where to place the prefetched data (caches, separate buffer)
  - Where to place the prefetcher (which level in memory hierarchy)
- How
  - How does the prefetcher operate and who operates it (software, hardware, execution/thread-based, cooperative, hybrid)

Challenges in Prefetching: How

- Software prefetching
  - ISA provides prefetch instructions
  - Programmer or compiler inserts prefetch instructions (effort)
  - Usually works well only for “regular access patterns”

- Hardware prefetching
  - Hardware monitors processor accesses
  - Memorizes or finds patterns/strides
  - Generates prefetch addresses automatically

- Execution-based prefetchers
  - A “thread” is executed to prefetch data for the main program
  - Can be generated by either software/programmer or hardware

Important: Prefetcher Performance

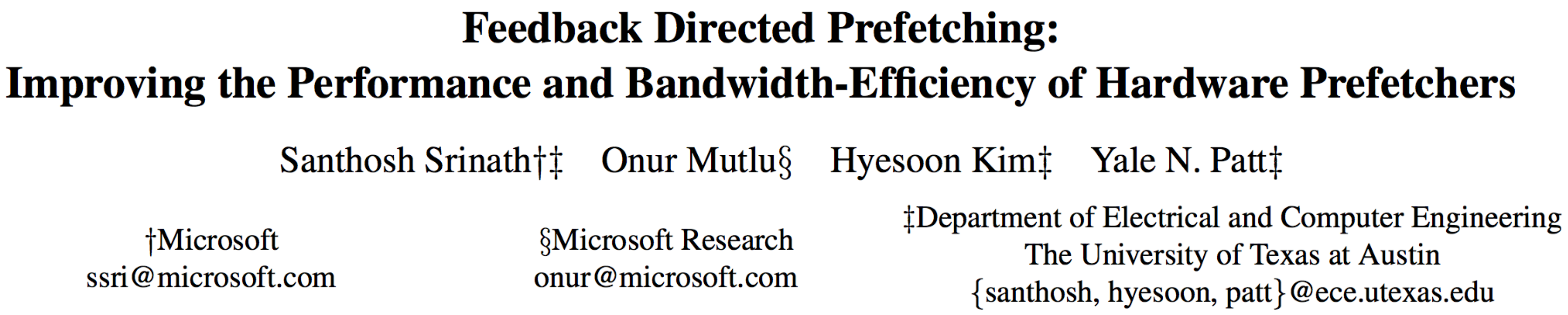
- Accuracy (used prefetches / sent prefetches)
- Coverage (prefetched misses / all misses)
- Timeliness (on-time prefetches / used prefetches)

- Bandwidth consumption
  - Memory bandwidth consumed with prefetcher / without prefetcher
  - Good news: Can utilize idle bus bandwidth (if available)

- Cache pollution
  - Extra demand misses due to prefetch placement in cache
  - More difficult to quantify but affects performance

Recommended Paper

- Santhosh Srinath, Onur Mutlu, Hyesoon Kim, and Yale N. Patt, **"Feedback Directed Prefetching: Improving the Performance and Bandwidth-Efficiency of Hardware Prefetchers"** *Proceedings of the 13th International Symposium on High-Performance Computer Architecture* (***HPCA***), pages 63-74, Phoenix, AZ, February 2007. Slides (ppt)
  One of the five papers nominated for the Best Paper Award by the Program Committee.



Feedback Directed Prefetching:
Improving the Performance and Bandwidth-Efficiency of Hardware Prefetchers

Santhosh Srinath†‡ Onur Mutlu§ Hyesoon Kim‡ Yale N. Patt‡

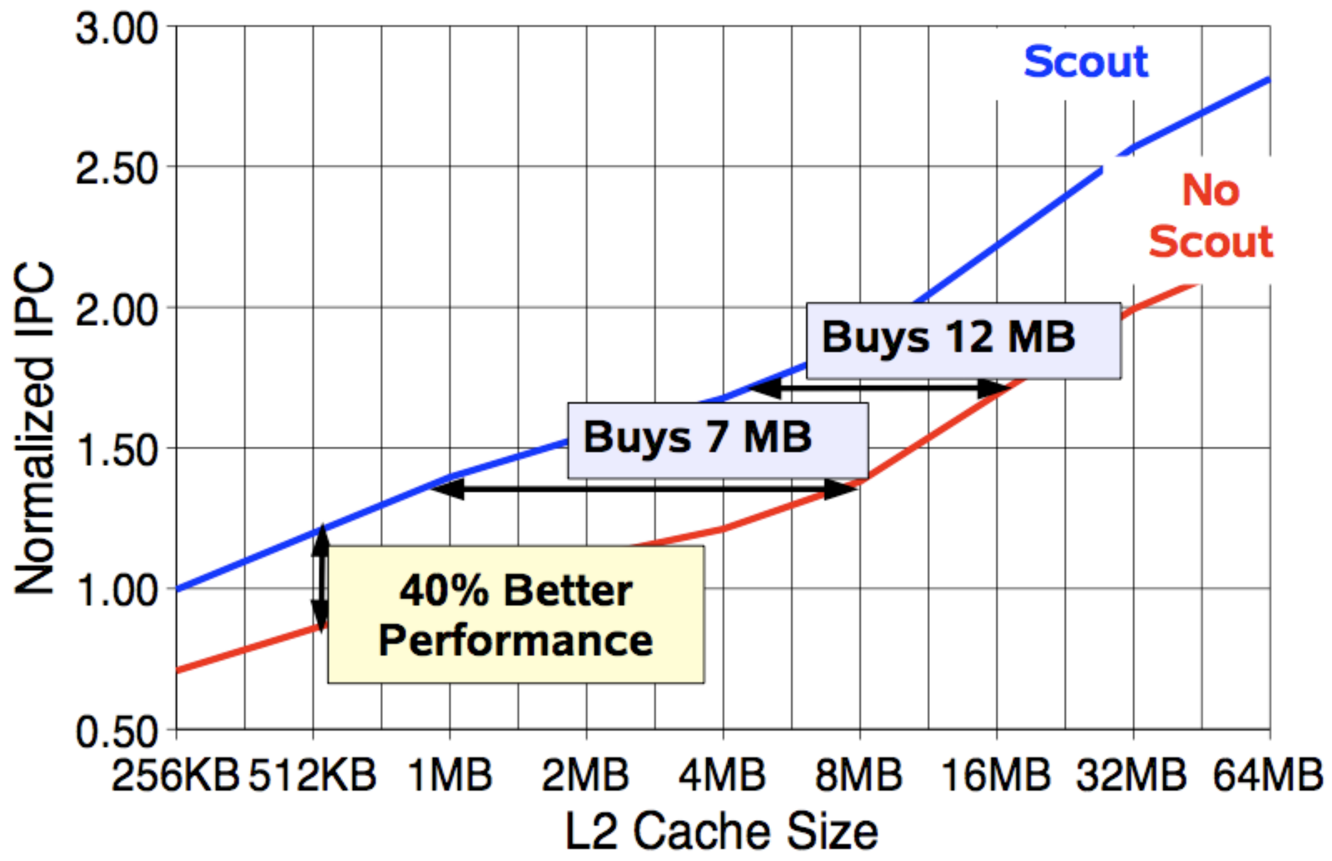
†Microsoft
ssri@microsoft.com

§Microsoft Research
onur@microsoft.com

‡Department of Electrical and Computer Engineering
The University of Texas at Austin
{santhosh, hyesoon, patt}@ece.utexas.edu

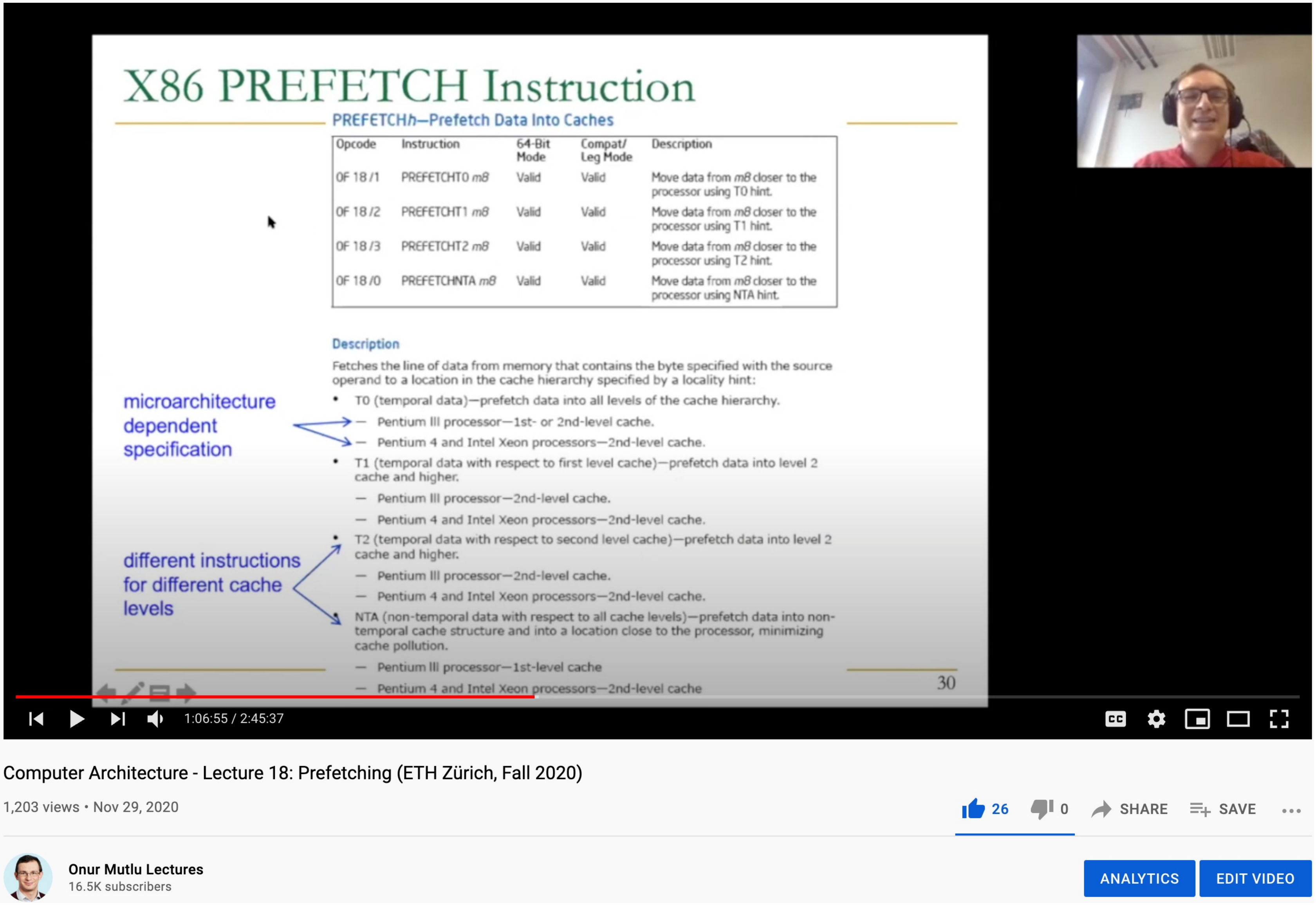
Effect of Runahead Prefetching in Sun ROCK

- Shailender Chaudhry talk, Aug 2008.



Effective prefetching can improve performance and reduce hardware cost

Lectures on Prefetching (I)



X86 PREFETCH Instruction

PREFETCH*h*—Prefetch Data Into Caches

Opcode	Instruction	64-Bit Mode	Compat/ Leg Mode	Description
0F 18 /1	PREFETCHT0 *m8*	Valid	Valid	Move data from *m8* closer to the processor using T0 hint.
0F 18 /2	PREFETCHT1 *m8*	Valid	Valid	Move data from *m8* closer to the processor using T1 hint.
0F 18 /3	PREFETCHT2 *m8*	Valid	Valid	Move data from *m8* closer to the processor using T2 hint.
0F 18 /0	PREFETCHNTA *m8*	Valid	Valid	Move data from *m8* closer to the processor using NTA hint.

Description

Fetches the line of data from memory that contains the byte specified with the source operand to a location in the cache hierarchy specified by a locality hint:

- T0 (temporal data)—prefetch data into all levels of the cache hierarchy.
  - Pentium III processor—1st- or 2nd-level cache.
  - Pentium 4 and Intel Xeon processors—2nd-level cache.
- T1 (temporal data with respect to first level cache)—prefetch data into level 2 cache and higher.
  - Pentium III processor—2nd-level cache.
  - Pentium 4 and Intel Xeon processors—2nd-level cache.
- T2 (temporal data with respect to second level cache)—prefetch data into level 2 cache and higher.
  - Pentium III processor—2nd-level cache.
  - Pentium 4 and Intel Xeon processors—2nd-level cache.
- NTA (non-temporal data with respect to all cache levels)—prefetch data into non-temporal cache structure and into a location close to the processor, minimizing cache pollution.
  - Pentium III processor—1st-level cache
  - Pentium 4 and Intel Xeon processors—2nd-level cache

microarchitecture dependent specification

different instructions for different cache levels

30

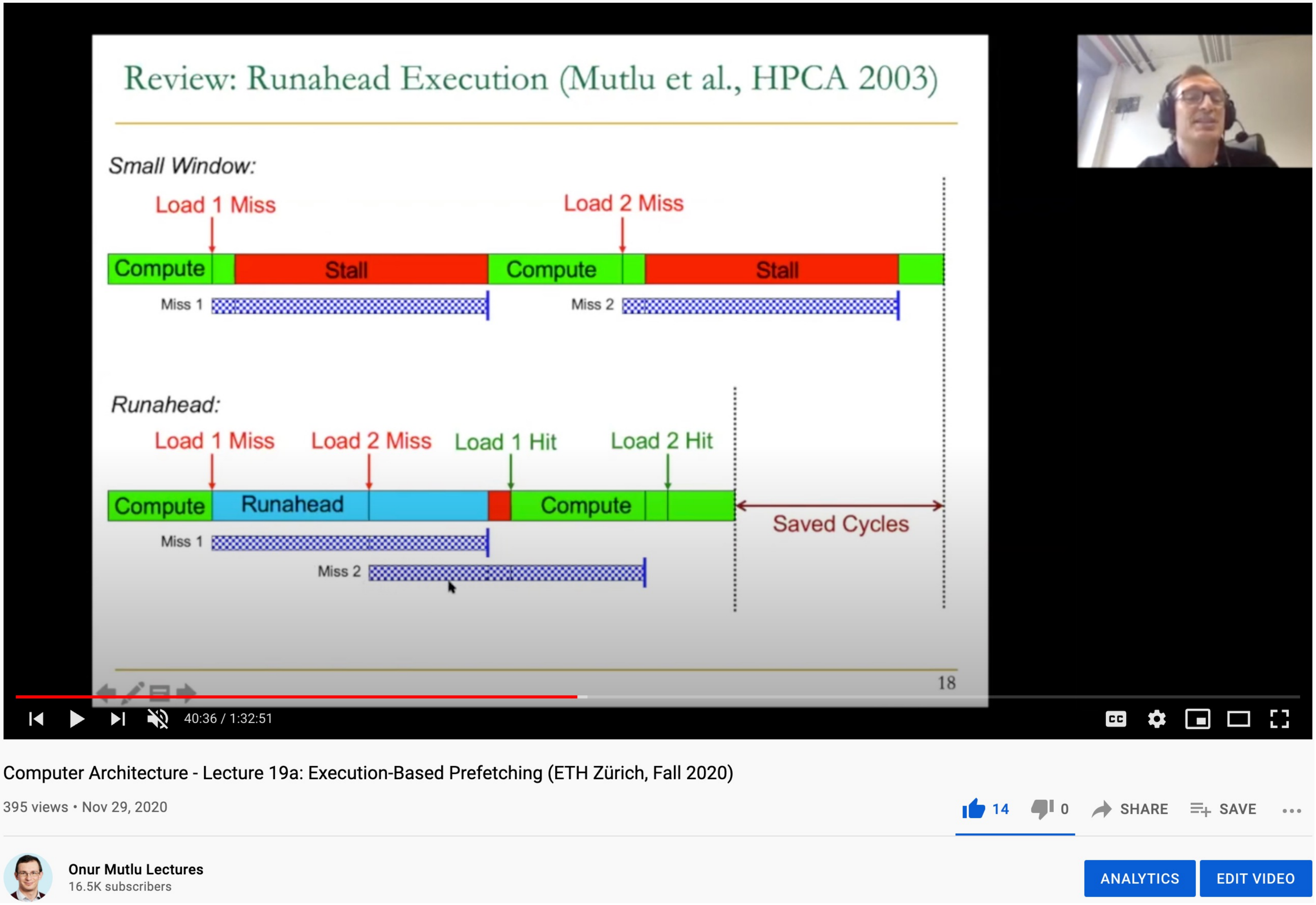
1:06:55 / 2:45:37

Computer Architecture - Lecture 18: Prefetching (ETH Zürich, Fall 2020)

1,203 views • Nov 29, 2020

https://www.youtube.com/watch?v=xZmDyj0g3Pw&list=PL5Q2soXY2Zi9xidyIgBxUz7xRPS-wisBN&index=33

Lectures on Prefetching (II)



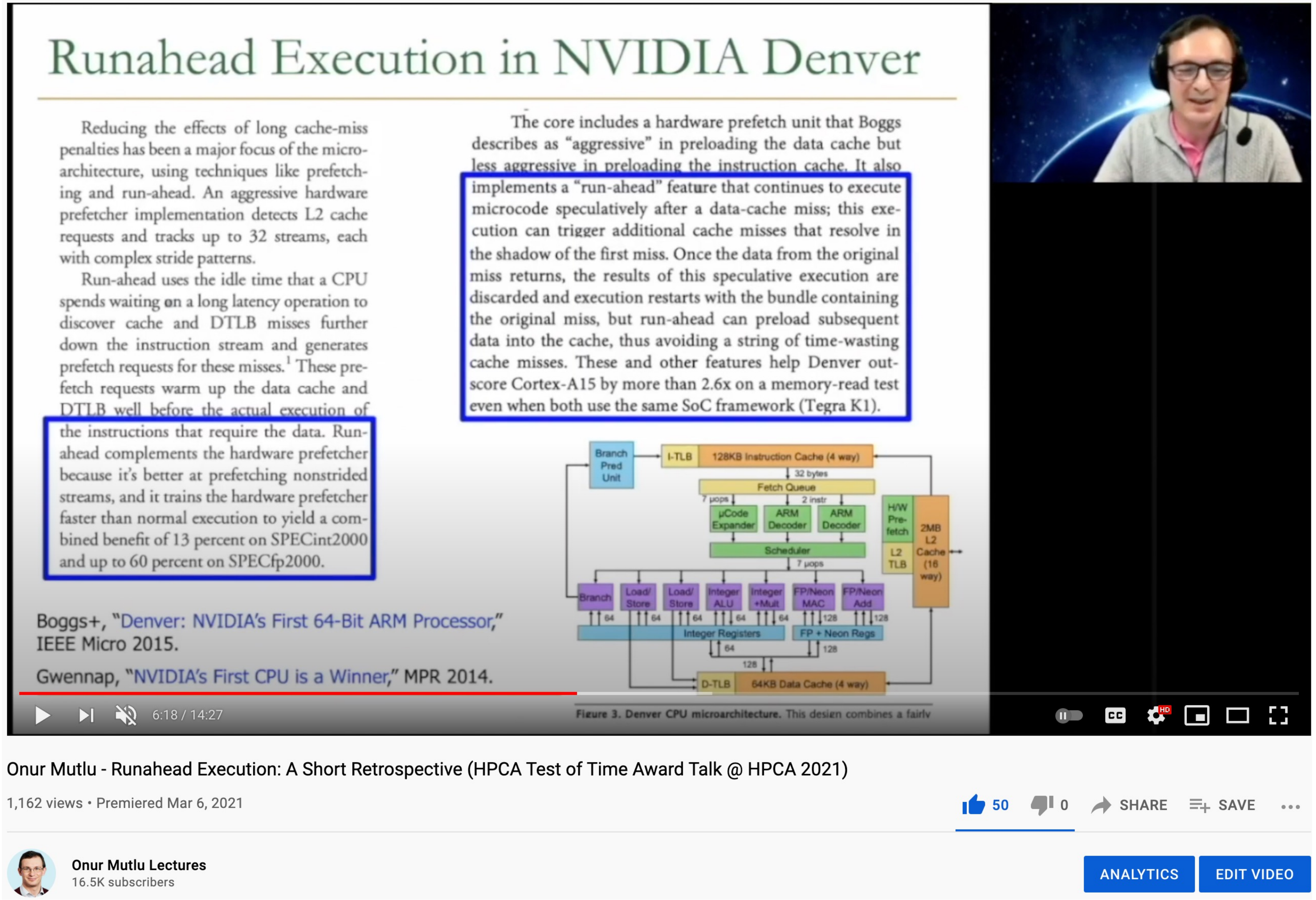
Computer Architecture - Lecture 19a: Execution-Based Prefetching (ETH Zürich, Fall 2020)

395 views • Nov 29, 2020

Onur Mutlu Lectures
16.5K subscribers

https://www.youtube.com/watch?v=zPewo6IaJ_8&list=PL5Q2soXY2Zi9xidyIgBxUz7xRPS-wisBN&index=34

Lectures on Prefetching (III)



Runahead Execution in NVIDIA Denver

Reducing the effects of long cache-miss penalties has been a major focus of the microarchitecture, using techniques like prefetching and run-ahead. An aggressive hardware prefetcher implementation detects L2 cache requests and tracks up to 32 streams, each with complex stride patterns.

Run-ahead uses the idle time that a CPU spends waiting on a long latency operation to discover cache and DTLB misses further down the instruction stream and generates prefetch requests for these misses.[1] These prefetch requests warm up the data cache and DTLB well before the actual execution of the instructions that require the data. Run-ahead complements the hardware prefetcher because it's better at prefetching nonstrided streams, and it trains the hardware prefetcher faster than normal execution to yield a combined benefit of 13 percent on SPECint2000 and up to 60 percent on SPECfp2000.

The core includes a hardware prefetch unit that Boggs describes as "aggressive" in preloading the data cache but less aggressive in preloading the instruction cache. It also implements a "run-ahead" feature that continues to execute microcode speculatively after a data-cache miss; this execution can trigger additional cache misses that resolve in the shadow of the first miss. Once the data from the original miss returns, the results of this speculative execution are discarded and execution restarts with the bundle containing the original miss, but run-ahead can preload subsequent data into the cache, thus avoiding a string of time-wasting cache misses. These and other features help Denver outscore Cortex-A15 by more than 2.6x on a memory-read test even when both use the same SoC framework (Tegra K1).

Figure 3. Denver CPU microarchitecture. This design combines a fairly

Boggs+, "Denver: NVIDIA's First 64-Bit ARM Processor," IEEE Micro 2015.

Gwennap, "NVIDIA's First CPU is a Winner," MPR 2014.

Onur Mutlu - Runahead Execution: A Short Retrospective (HPCA Test of Time Award Talk @ HPCA 2021)

1,162 views • Premiered Mar 6, 2021

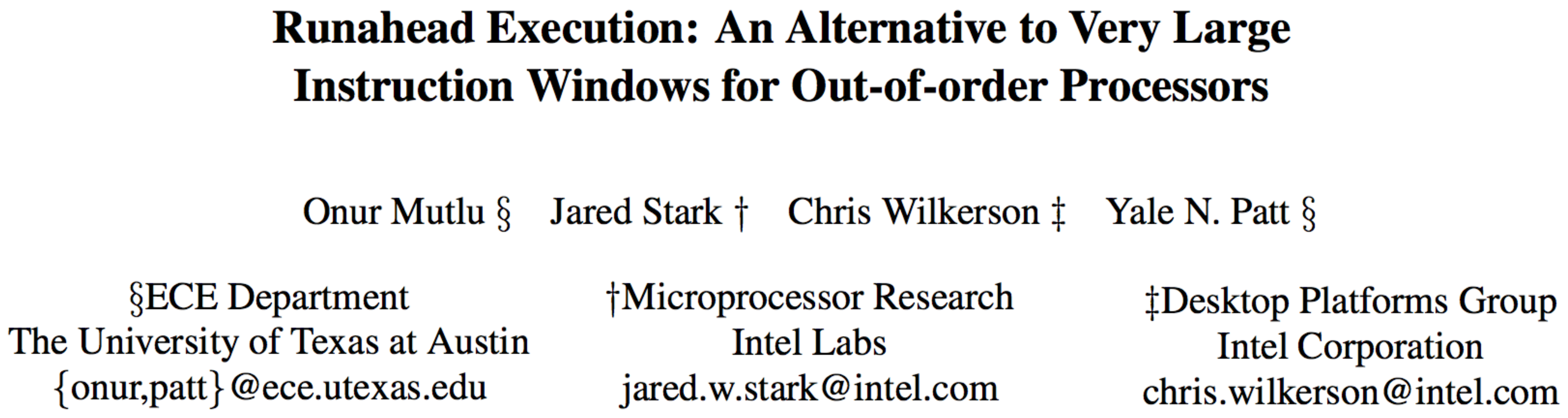
https://www.youtube.com/watch?v=KFCOecRQTIc

Lectures on Prefetching

- Computer Architecture, Fall 2020, Lecture 18
  - Prefetching (ETH, Fall 2020)
  - https://www.youtube.com/watch?v=xZmDyj0g3Pw&list=PL5Q2soXY2Zi9xidyIgBxUz7xRPS-wisBN&index=33
- Computer Architecture, Fall 2020, Lecture 19a
  - Execution-Based Prefetching (ETH, Fall 2020)
  - https://www.youtube.com/watch?v=zPewo6IaJ_8&list=PL5Q2soXY2Zi9xidyIgBxUz7xRPS-wisBN&index=34
- Computer Architecture, Spring 2015, Lecture 25
  - Prefetching (CMU, Spring 2015)
  - https://www.youtube.com/watch?v=ibPL7T9iEwY&list=PL5PHm2jkkXmi5CxxI7b3JCL1TWybTDtKq&index=29
- Computer Architecture, Spring 2015, Lecture 26
  - More Prefetching (CMU, Spring 2015)
  - https://www.youtube.com/watch?v=TUFins4z6o4&list=PL5PHm2jkkXmi5CxxI7b3JCL1TWybTDtKq&index=30

Some Readings on Prefetching

- Onur Mutlu, Jared Stark, Chris Wilkerson, and Yale N. Patt,
"Runahead Execution: An Alternative to Very Large Instruction Windows for Out-of-order Processors"
Proceedings of the 9th International Symposium on High-Performance Computer Architecture (***HPCA***), pages 129-140, Anaheim, CA, February 2003. Slides (pdf)
One of the 15 computer arch. papers of 2003 selected as Top Picks by IEEE Micro. HPCA Test of Time Award (awarded in 2021).
[Lecture Slides (pptx) (pdf)]
[Lecture Video (1 hr 54 mins)]
[Retrospective HPCA Test of Time Award Talk Slides (pptx) (pdf)]
[Retrospective HPCA Test of Time Award Talk Video (14 minutes)]



Runahead Execution: An Alternative to Very Large Instruction Windows for Out-of-order Processors

Onur Mutlu § Jared Stark † Chris Wilkerson ‡ Yale N. Patt §

§ECE Department
The University of Texas at Austin
{onur,patt}@ece.utexas.edu

†Microprocessor Research
Intel Labs
jared.w.stark@intel.com

‡Desktop Platforms Group
Intel Corporation
chris.wilkerson@intel.com

Some Readings on Prefetching

- Onur Mutlu, Jared Stark, Chris Wilkerson, and Yale N. Patt, **"Runahead Execution: An Effective Alternative to Large Instruction Windows"** *IEEE Micro, Special Issue: Micro's Top Picks from Microarchitecture Conferences* (***MICRO TOP PICKS***), Vol. 23, No. 6, pages 20-25, November/December 2003.

RUNAHEAD EXECUTION:
AN EFFECTIVE ALTERNATIVE TO
LARGE INSTRUCTION WINDOWS