Scalable Many-Core Memory Systems Lecture 4, Topic 3: Memory Interference and QoS-Aware Memory Systems

Prof. Onur Mutlu
http://www.ece.cmu.edu/~omutlu
onur@cmu.edu

HiPEAC ACACES Summer School 2013
July 18, 2013

Carnegie Mellon

SAFARI

What Will You Learn in This Course?

- Scalable Many-Core Memory Systems
  - July 15-19, 2013

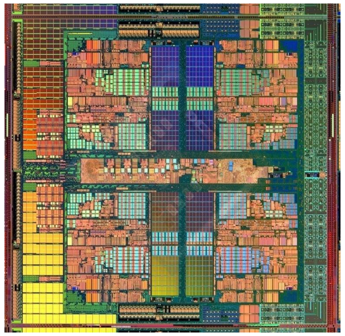
- Topic 1: Main memory basics, DRAM scaling
- Topic 2: Emerging memory technologies and hybrid memories
- Topic 3: Main memory interference and QoS
- Topic 4 (unlikely): Cache management
- Topic 5 (unlikely): Interconnects

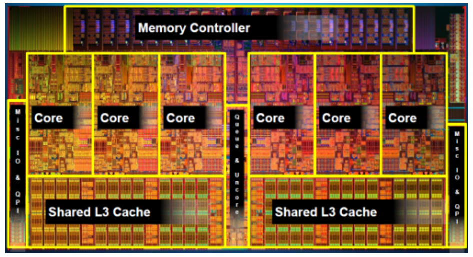
- Major Overview Reading:
  - Mutlu, "Memory Scaling: A Systems Architecture Perspective," IMW 2013.

Main Memory Interference

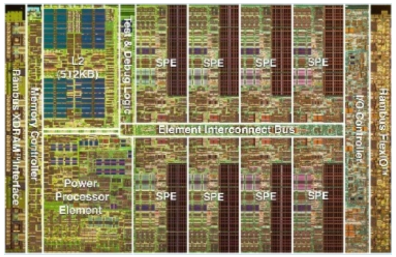
Trend: Many Cores on Chip

- Simpler and lower power than a single large core
- Large scale parallelism on chip

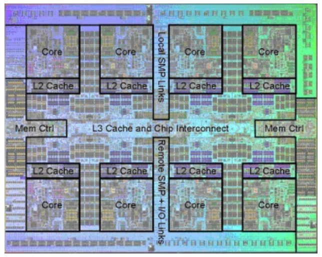

AMD Barcelona
4 cores



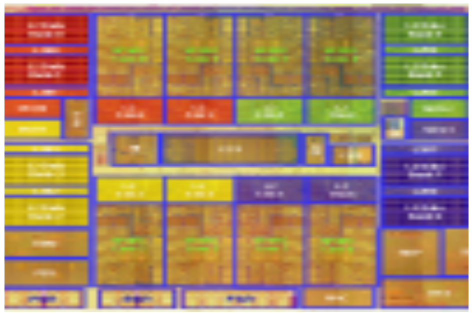
Intel Core i7
8 cores

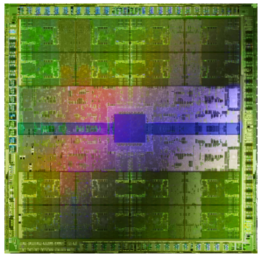


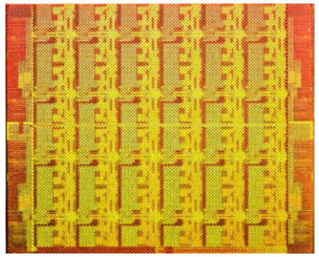
IBM Cell BE
8+1 cores

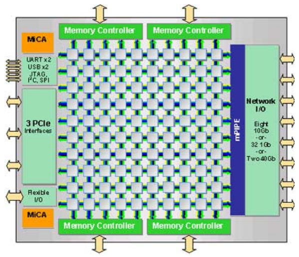


IBM POWER7
8 cores


Sun Niagara II
8 cores


Nvidia Fermi
448 "cores"


Intel SCC
48 cores, networked



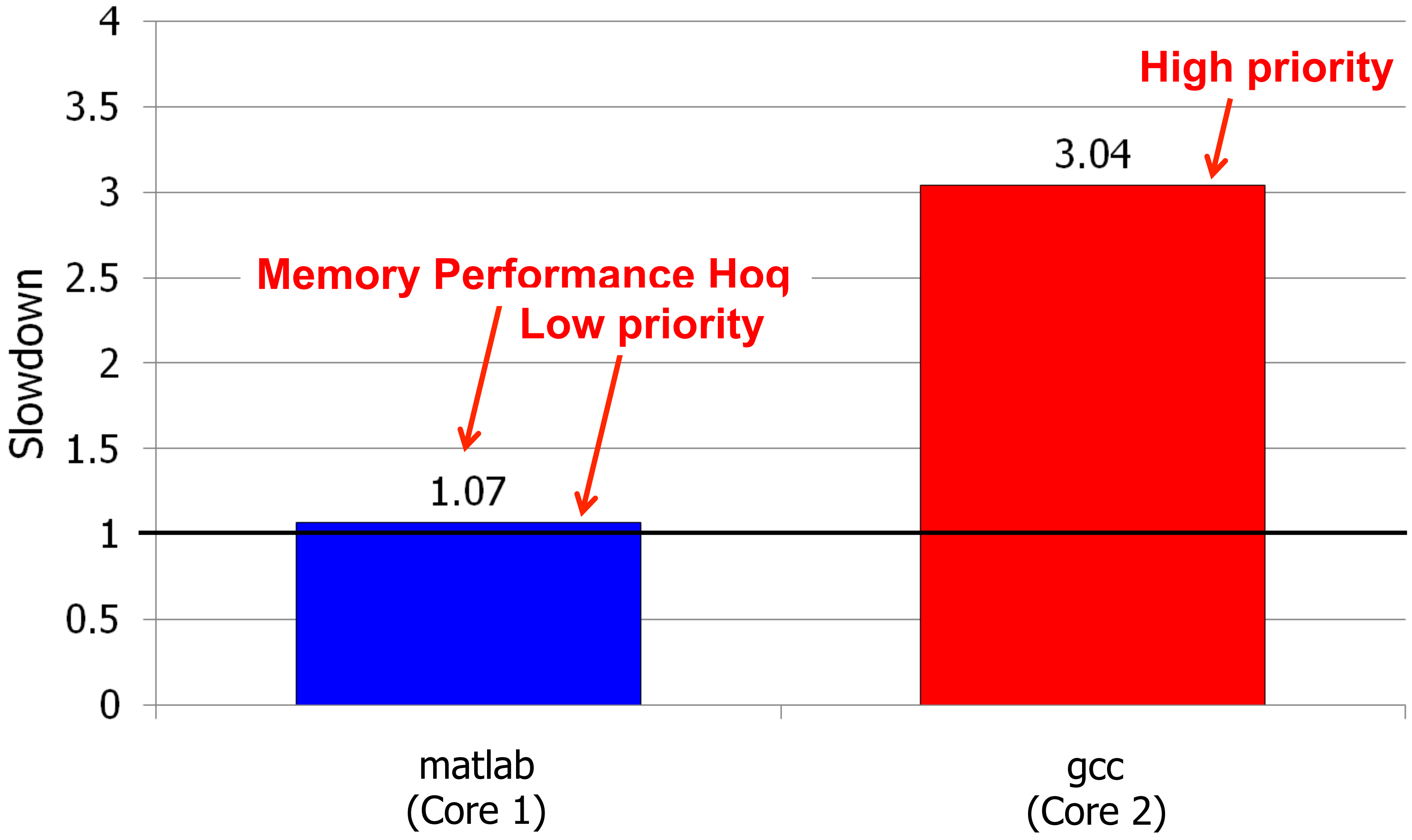
Tilera TILE Gx
100 cores, networked

Many Cores on Chip

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

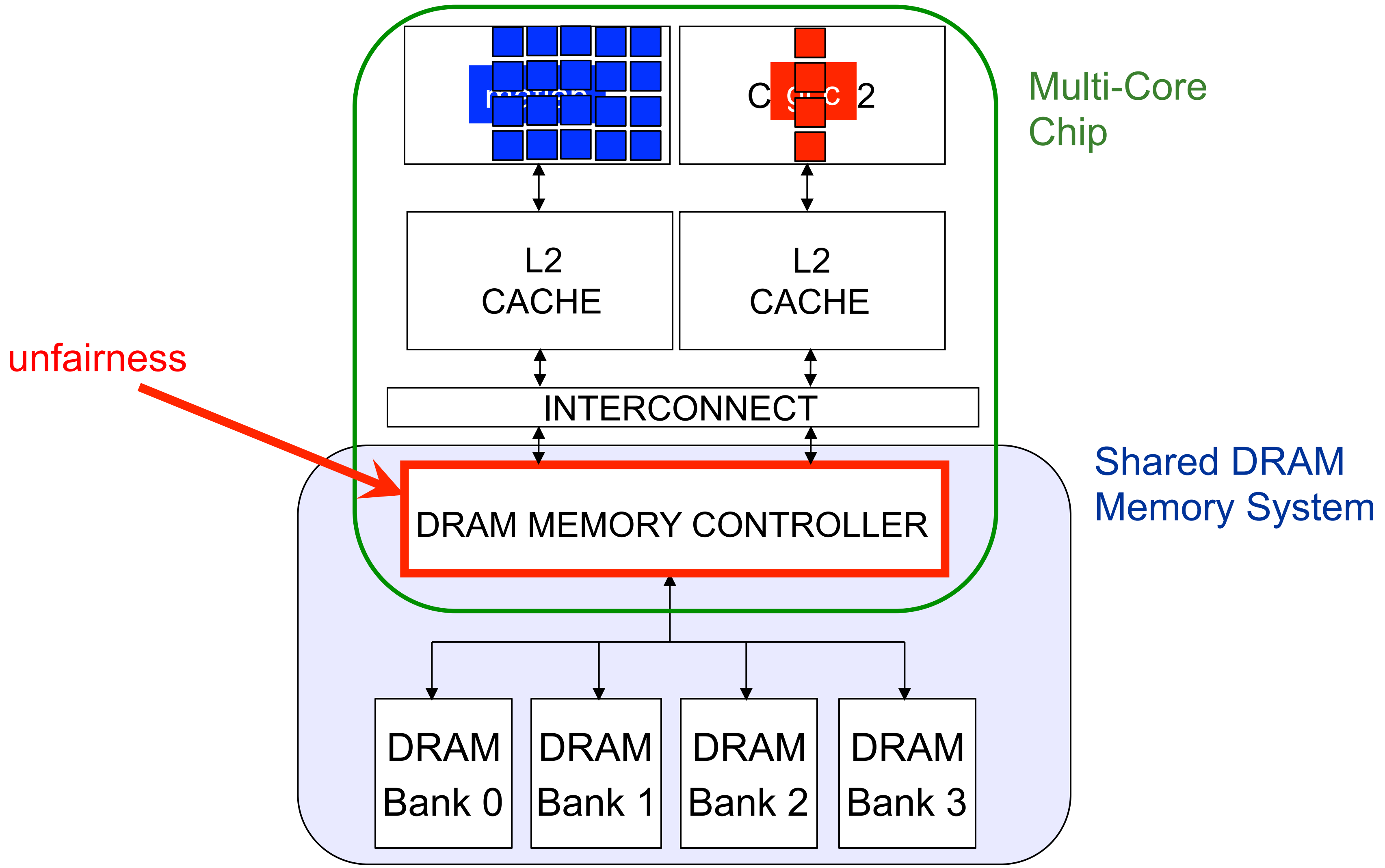
- What do we get today?

Unfair Slowdowns due to Interference

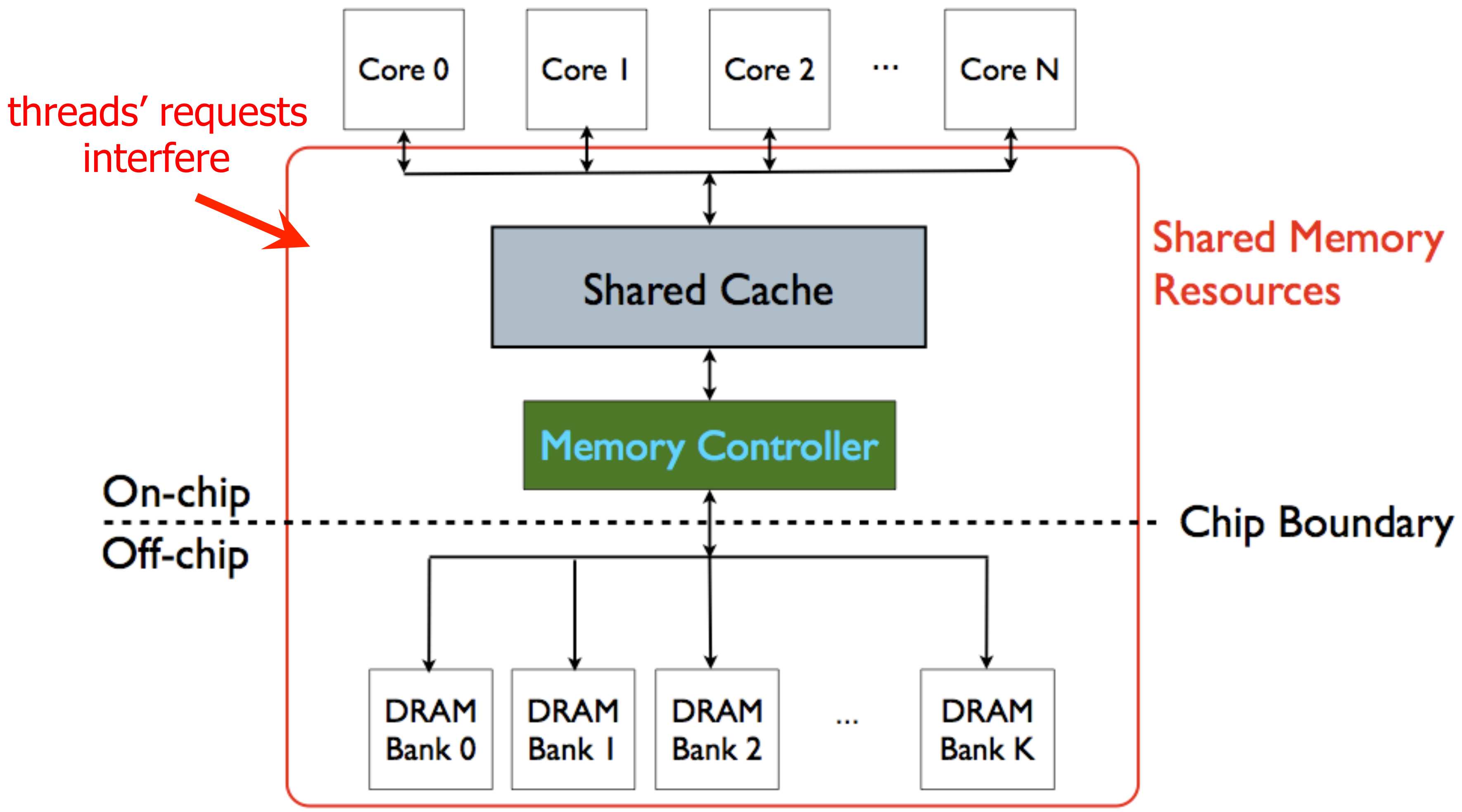


Moscibroda and Mutlu, "Memory performance attacks: Denial of memory service in multi-core systems," USENIX Security 2007.

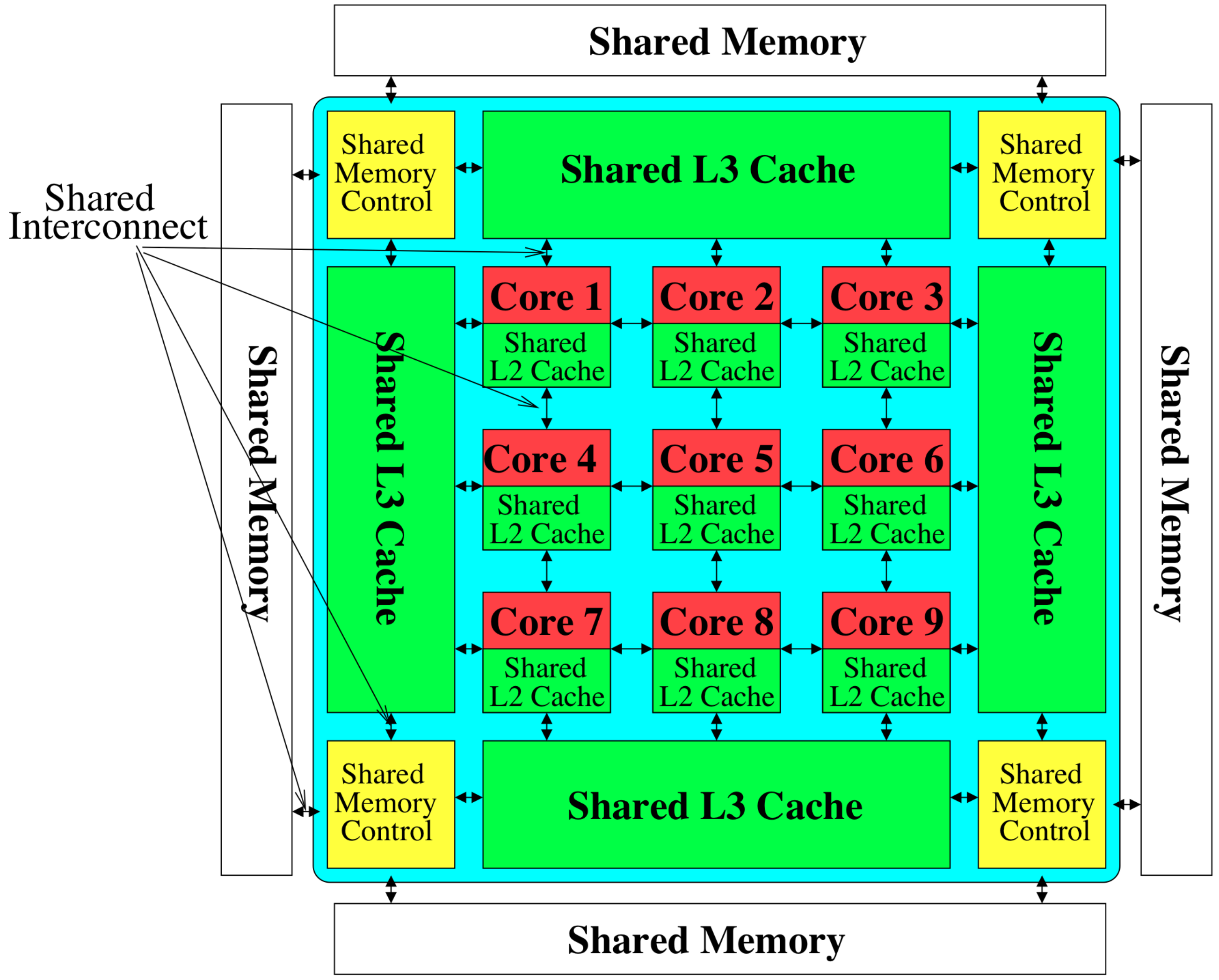
Uncontrolled Interference: An Example



Memory System is the Major Shared Resource



Much More of a Shared Resource in Future



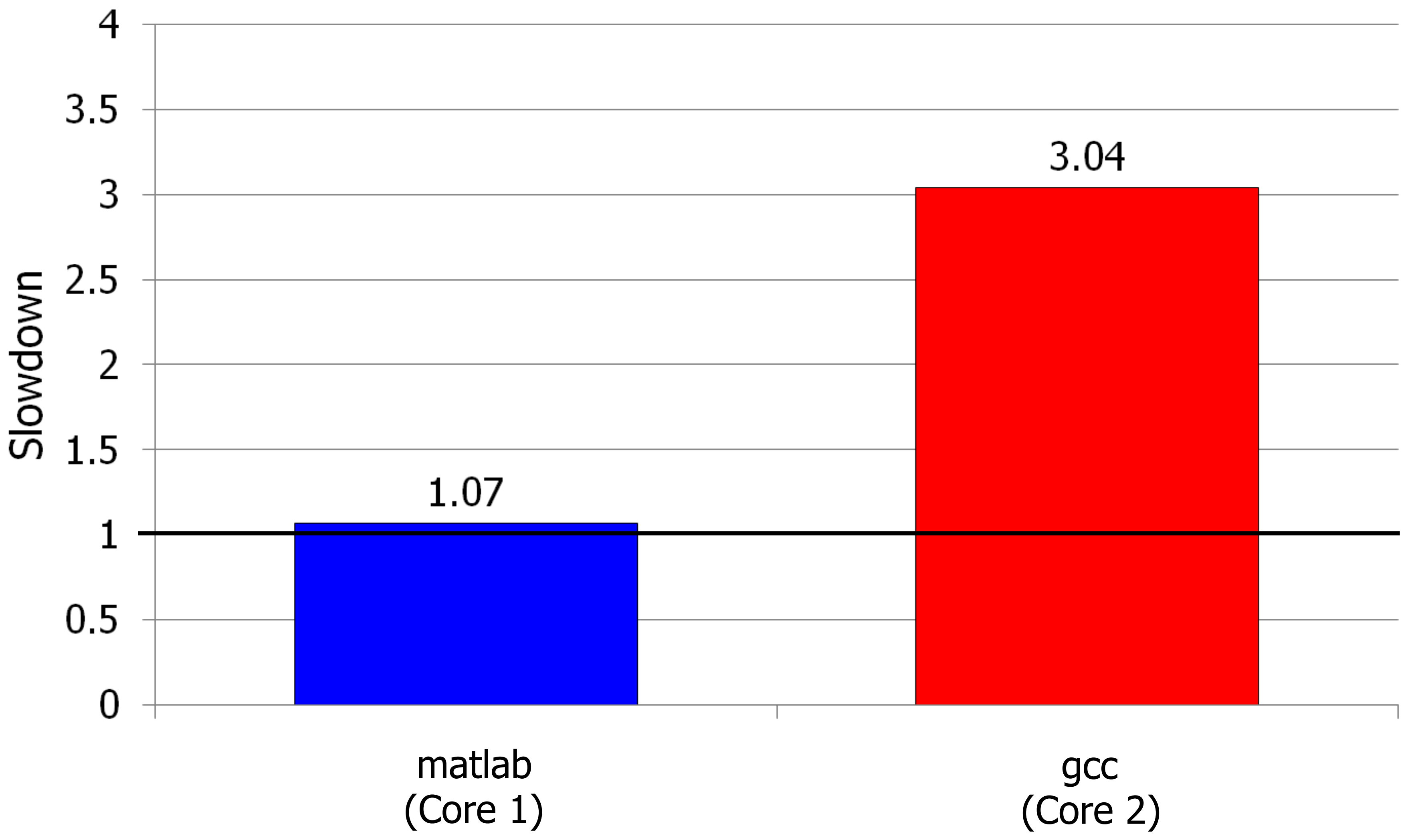
SAFARI

Inter-Thread/Application Interference

- Problem: Threads share the memory system, but memory system does not distinguish between threads' requests

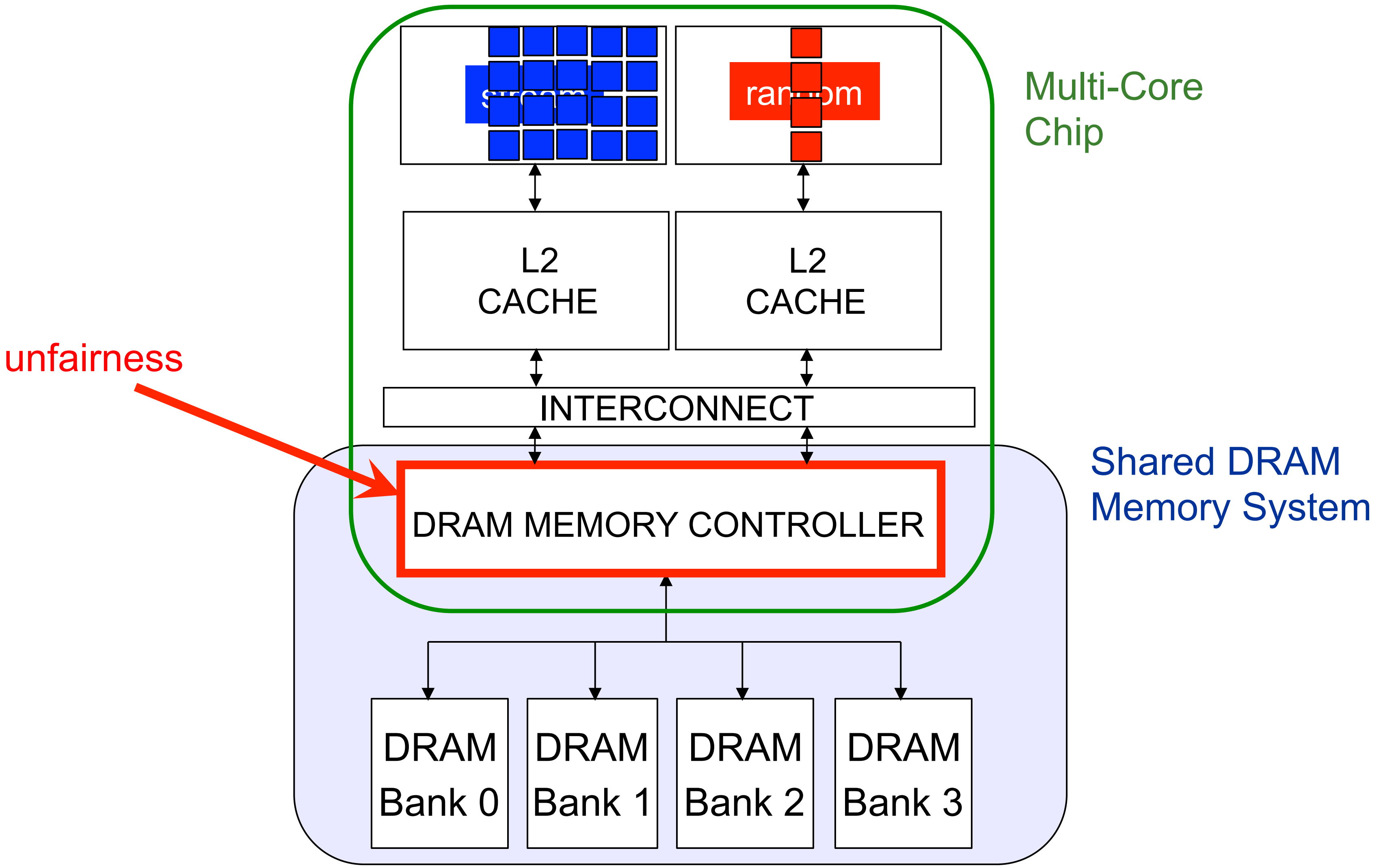
- Existing memory systems
  - Free-for-all, shared based on demand
  - Control algorithms thread-unaware and thread-unfair
  - Aggressive threads can deny service to others
  - Do not try to reduce or control inter-thread interference

Unfair Slowdowns due to Interference



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

Uncontrolled Interference: An Example



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

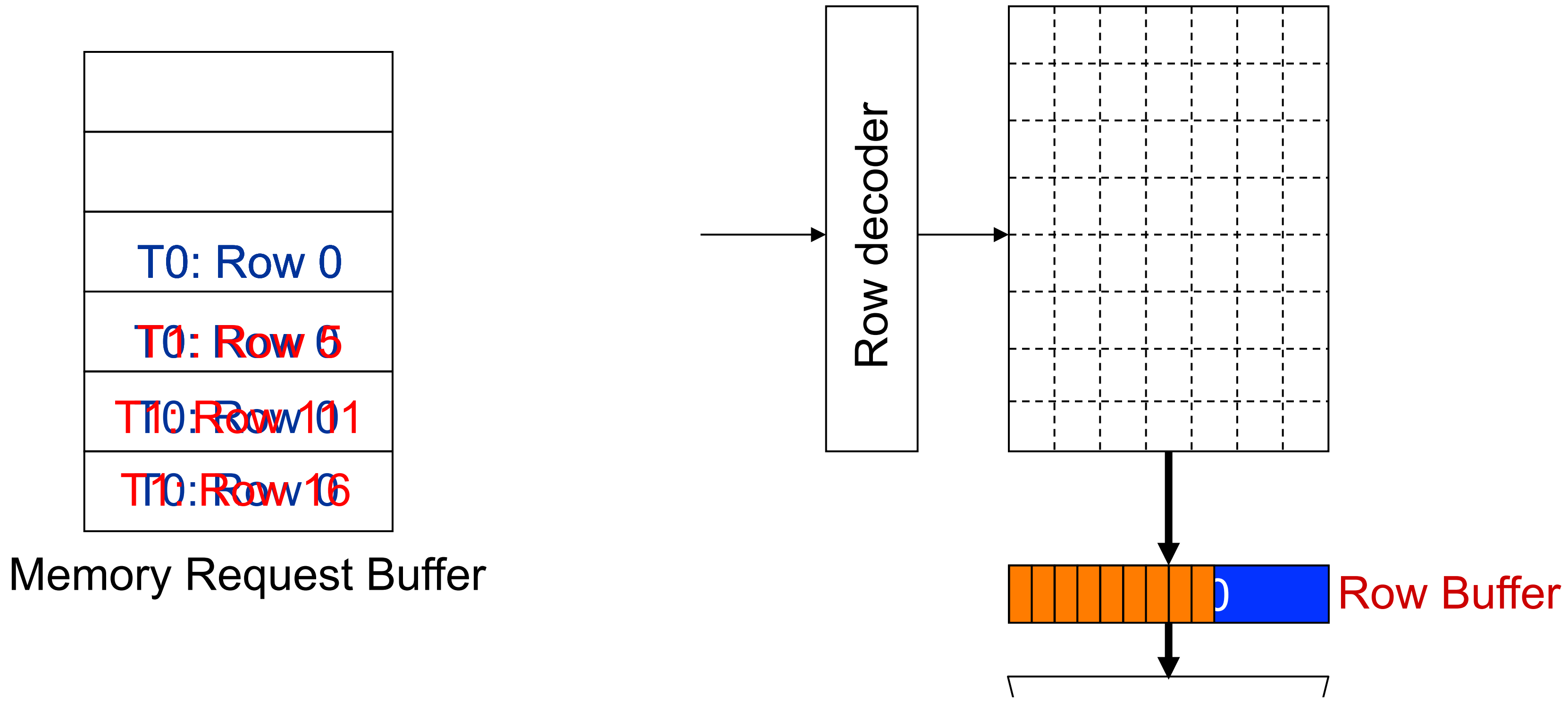
- Sequential memory access
- Very high row buffer locality (96% hit rate)
- Memory intensive

RANDOM

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

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

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]*

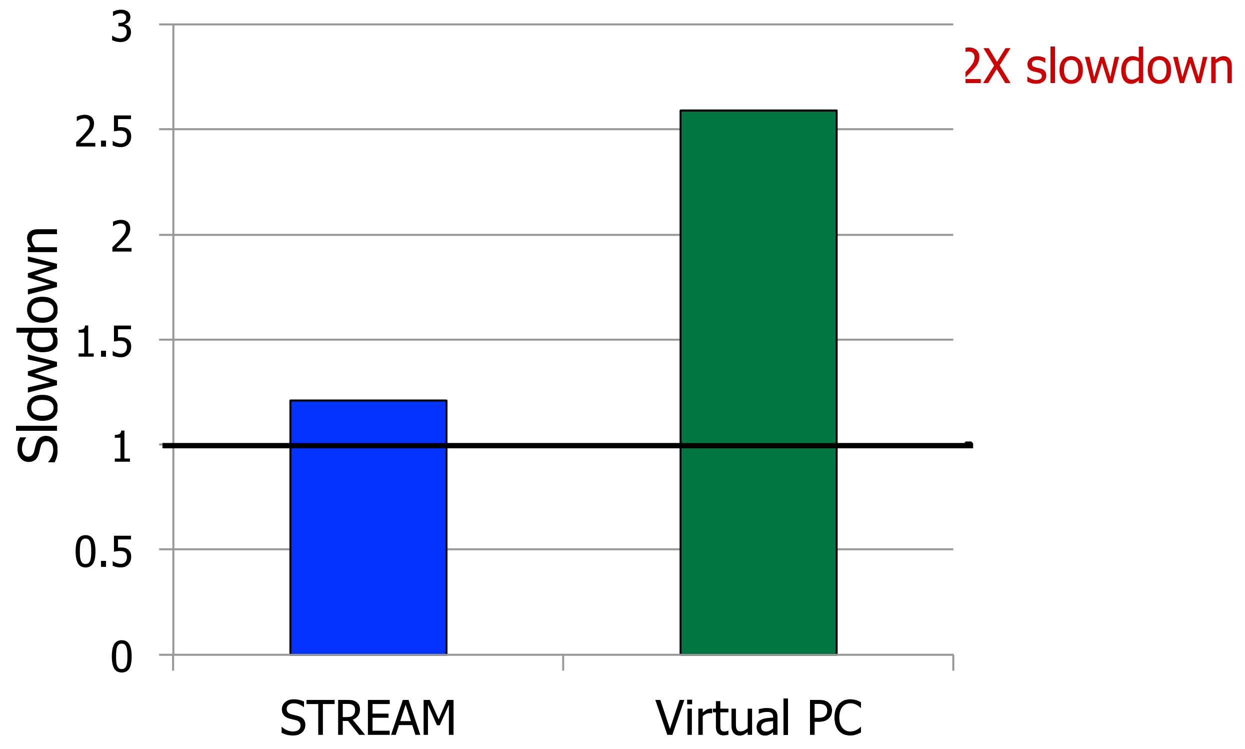
  (1) Row-hit first: Service row-hit memory accesses first

  (2) Oldest-first: Then service older accesses first

- This scheduling policy aims to maximize DRAM throughput
  - But, it is unfair when multiple threads share the DRAM system

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

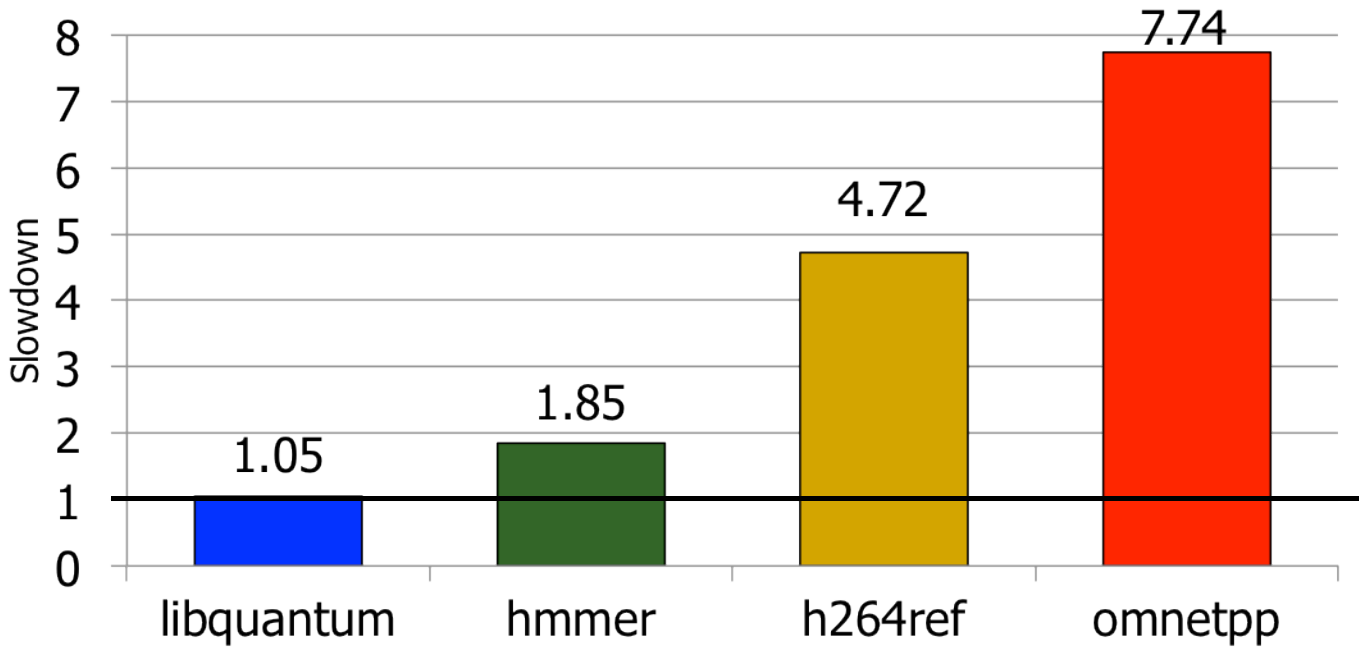
Effect of the Memory Performance Hog



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.

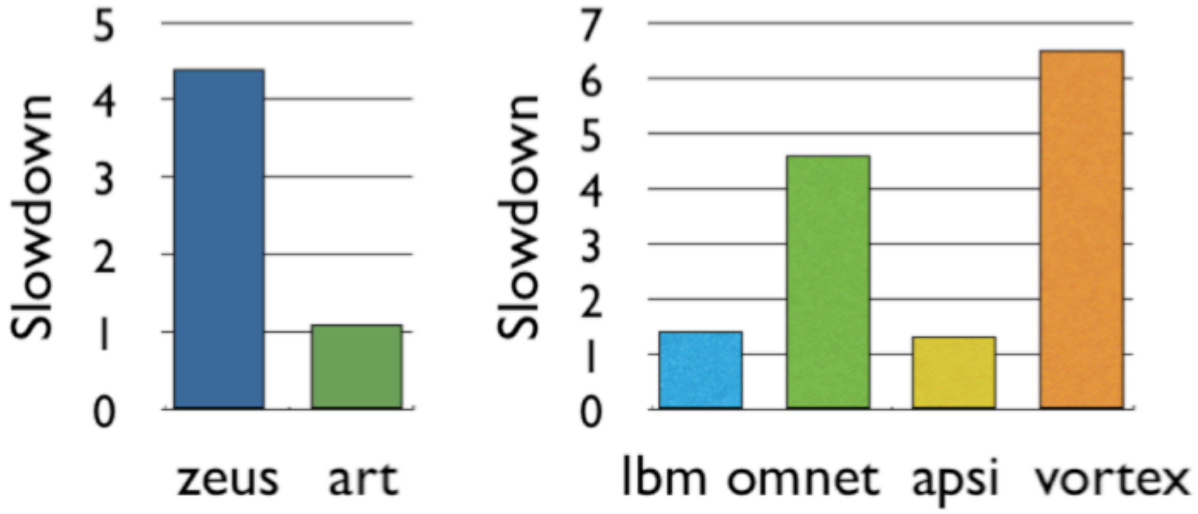
Greater Problem with More Cores



- 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

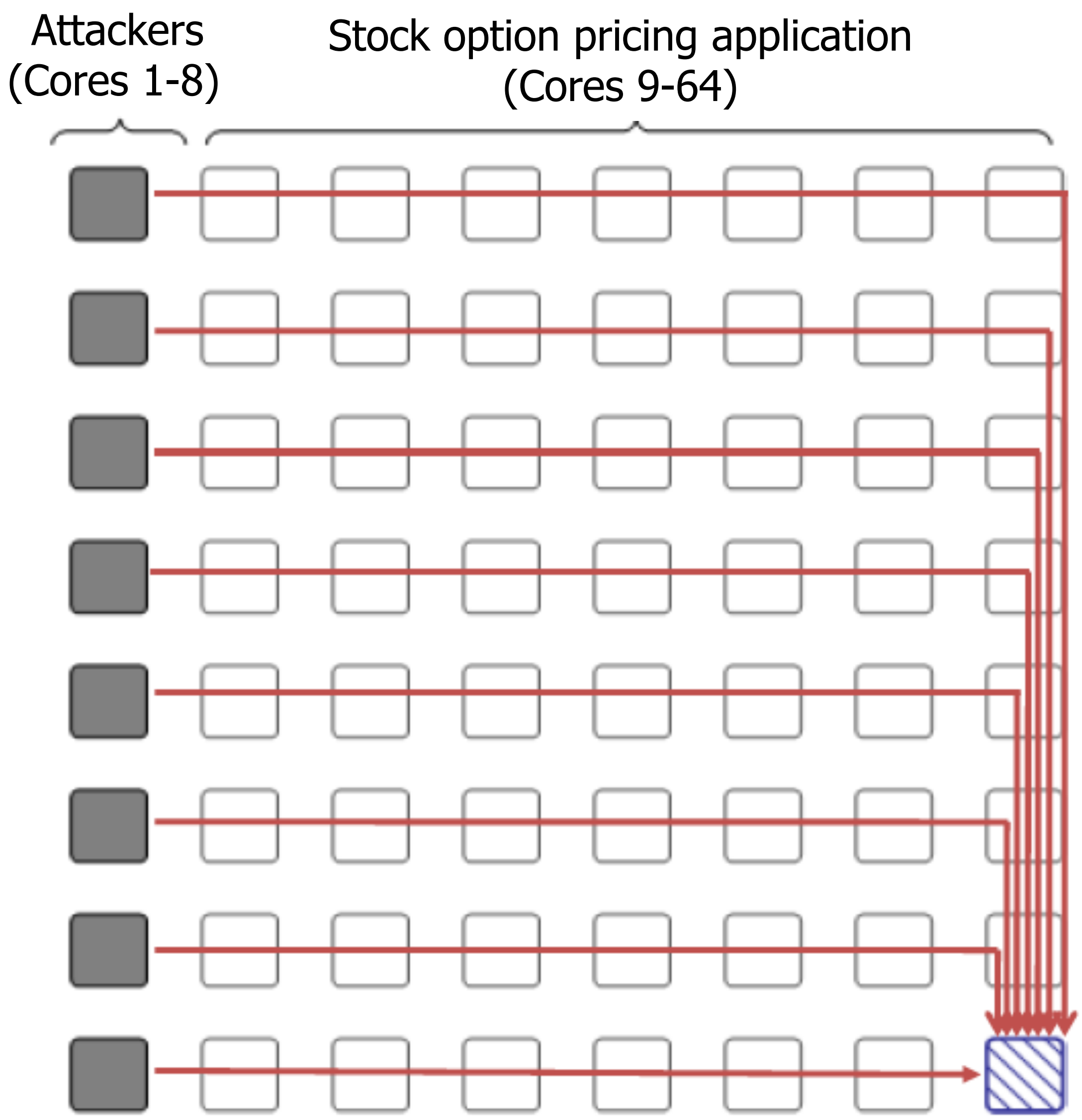
Uncontrollable, unpredictable system

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.



How Do We Solve The Problem?

- Inter-thread interference is uncontrolled in all memory resources
  - Memory controller
  - Interconnect
  - Caches

- We need to control it
  - i.e., design an interference-aware (QoS-aware) memory system

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

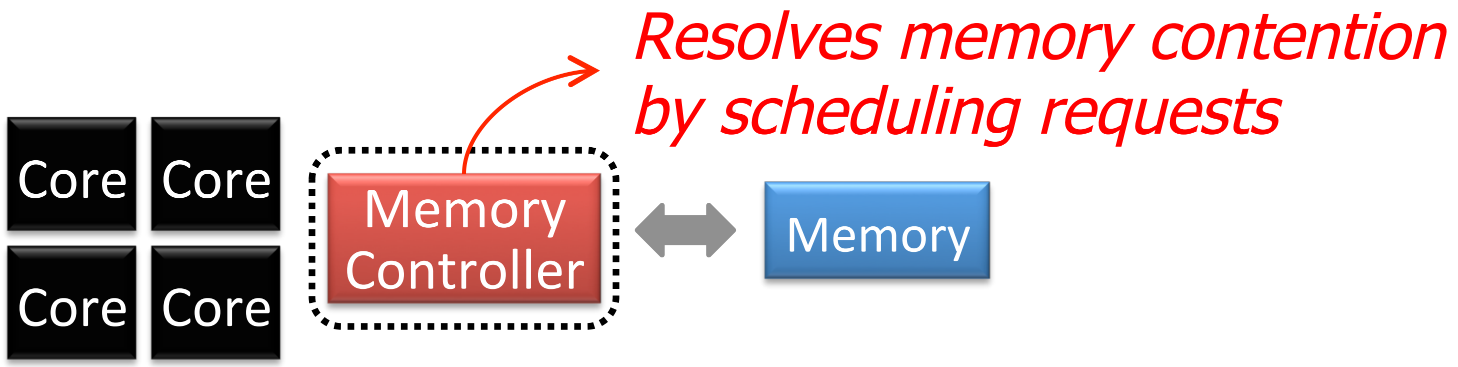
- How do we make the memory system configurable/flexible?
  - Enable flexible mechanisms that can achieve many goals
    - Provide fairness or throughput when needed
    - Satisfy performance guarantees when needed

Designing QoS-Aware Memory Systems: Approaches

- Smart resources: Design each shared resource to have a configurable interference control/reduction mechanism
  - QoS-aware memory controllers [Mutlu+ MICRO'07] [Moscibroda+, Usenix Security'07] [Mutlu+ ISCA'08, Top Picks'09] [Kim+ HPCA'10] [Kim+ MICRO'10, Top Picks'11] [Ebrahimi+ ISCA'11, MICRO'11] [Ausavarungnirun+, ISCA'12][Subramanian+, HPCA'13]
  - QoS-aware interconnects [Das+ MICRO'09, ISCA'10, Top Picks '11] [Grot+ MICRO'09, ISCA'11, Top Picks '12]
  - 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 [Ebrahimi+ ASPLOS'10, ISCA'11, TOCS'12] [Ebrahimi+ MICRO'09] [Nychis+ HotNets'10] [Nychis+ SIGCOMM'12]
  - QoS-aware data mapping to memory controllers [Muralidhara+ MICRO'11]
  - QoS-aware thread scheduling to cores [Das+ HPCA'13]

QoS-Aware Memory Scheduling



- How to schedule requests to provide
  - High system performance
  - High fairness to applications
  - Configurability to system software

- Memory controller needs to be aware of threads

QoS-Aware Memory Scheduling: Evolution

QoS-Aware Memory Scheduling: Evolution

- Stall-time fair memory scheduling [Mutlu+ MICRO'07]
  - Idea: Estimate and balance thread slowdowns
  - Takeaway: Proportional thread progress improves performance, especially when threads are "heavy" (memory intensive)
- Parallelism-aware batch scheduling [Mutlu+ ISCA'08, Top Picks'09]
  - Idea: Rank threads and service in rank order (to preserve bank parallelism); batch requests to prevent starvation
  - Takeaway: Preserving within-thread bank-parallelism improves performance; request batching improves fairness
- ATLAS memory scheduler [Kim+ HPCA'10]
  - Idea: Prioritize threads that have attained the least service from the memory scheduler
  - Takeaway: Prioritizing "light" threads improves performance

QoS-Aware Memory Scheduling: Evolution

- Thread cluster memory scheduling [Kim+ MICRO'10]
  - Idea: Cluster threads into two groups (latency vs. bandwidth sensitive); prioritize the latency-sensitive ones; employ a fairness policy in the bandwidth sensitive group
  - Takeaway: Heterogeneous scheduling policy that is different based on thread behavior maximizes both performance and fairness

- Integrated Memory Channel Partitioning and Scheduling [Muralidhara+ MICRO'11]
  - Idea: Only prioritize very latency-sensitive threads in the scheduler; mitigate all other applications' interference via channel partitioning
  - Takeaway: Intelligently combining application-aware channel partitioning and memory scheduling provides better performance than either

QoS-Aware Memory Scheduling: Evolution

- Parallel application memory scheduling [Ebrahimi+ MICRO'11]
  - Idea: Identify and prioritize limiter threads of a multithreaded application in the memory scheduler; provide fast and fair progress to non-limiter threads
  - Takeaway: Carefully prioritizing between limiter and non-limiter threads of a parallel application improves performance

- Staged memory scheduling [Ausavarungnirun+ ISCA'12]
  - Idea: Divide the functional tasks of an application-aware memory scheduler into multiple distinct stages, where each stage is significantly simpler than a monolithic scheduler
  - Takeaway: Staging enables the design of a scalable and relatively simpler application-aware memory scheduler that works on very large request buffers

QoS-Aware Memory Scheduling: Evolution

- MISE [Subramanian+ HPCA'13]
  - Idea: Estimate the performance of a thread by estimating its change in memory request service rate when run alone vs. shared → use this simple model to estimate slowdown to design a scheduling policy that provides predictable performance or fairness
  - Takeaway: Request service rate of a thread is a good proxy for its performance; alone request service rate can be estimated by giving high priority to the thread in memory scheduling for a while

QoS-Aware Memory Scheduling: Evolution

- Prefetch-aware shared resource management [Ebrahimi+ ISCA'12] [Ebrahimi+ MICRO'09] [Lee+ MICRO'08]
  - Idea: Prioritize prefetches depending on how they affect system performance; even accurate prefetches can degrade performance of the system
  - Takeaway: Carefully controlling and prioritizing prefetch requests improves performance and fairness

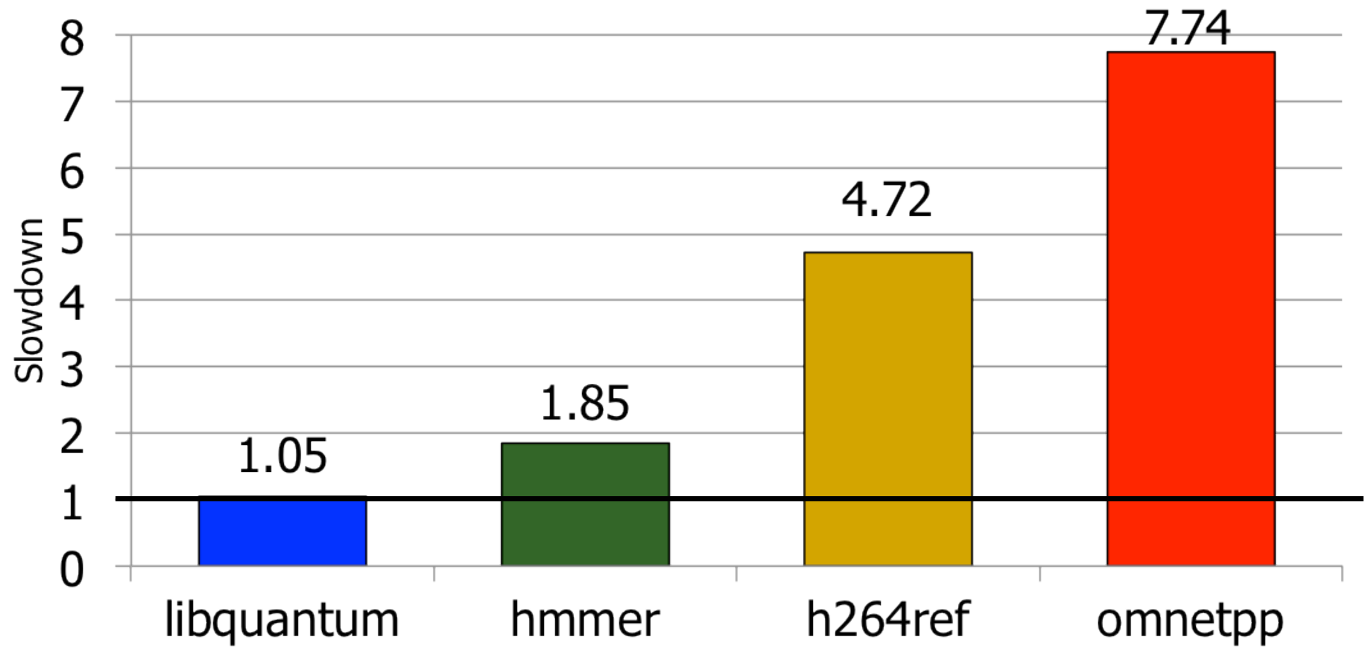
- DRAM-Aware last-level cache policies and write scheduling [Lee+ HPS Tech Report'10] [Lee+ HPS Tech Report'10]
  - Idea: Design cache eviction and replacement policies such that they proactively exploit the state of the memory controller and DRAM (e.g., proactively evict data from the cache that hit in open rows)
  - Takeaway: Coordination of last-level cache and DRAM policies improves performance and fairness

Stall-Time Fair Memory Scheduling

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

STFM Micro 2007 Talk

The Problem: Unfairness



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

Uncontrollable, unpredictable system

How Do We Solve the Problem?

- Stall-time fair memory scheduling [Mutlu+ MICRO'07]
- Goal: Threads sharing main memory should experience similar slowdowns compared to when they are run alone → fair scheduling
    - Also improves overall system performance by ensuring cores make "proportional" progress
- Idea: Memory controller estimates each thread's slowdown due to interference and schedules requests in a way to balance the slowdowns
- Mutlu and Moscibroda, "Stall-Time Fair Memory Access Scheduling for Chip Multiprocessors," MICRO 2007.

Stall-Time Fairness in Shared DRAM Systems

- A DRAM system is fair if it equalizes the slowdown of equal-priority threads relative to when each thread is run alone on the same system

- DRAM-related stall-time: The time a thread spends waiting for DRAM memory
- ST_{shared}: DRAM-related stall-time when the thread runs with other threads
- ST_{alone}: DRAM-related stall-time when the thread runs alone
- **Memory-slowdown = ST_{shared}/ST_{alone}**
  - Relative increase in stall-time

- *Stall-Time Fair Memory scheduler (STFM)* aims to equalize Memory-slowdown for interfering threads, without sacrificing performance
  - Considers inherent DRAM performance of each thread
  - Aims to allow proportional progress of threads

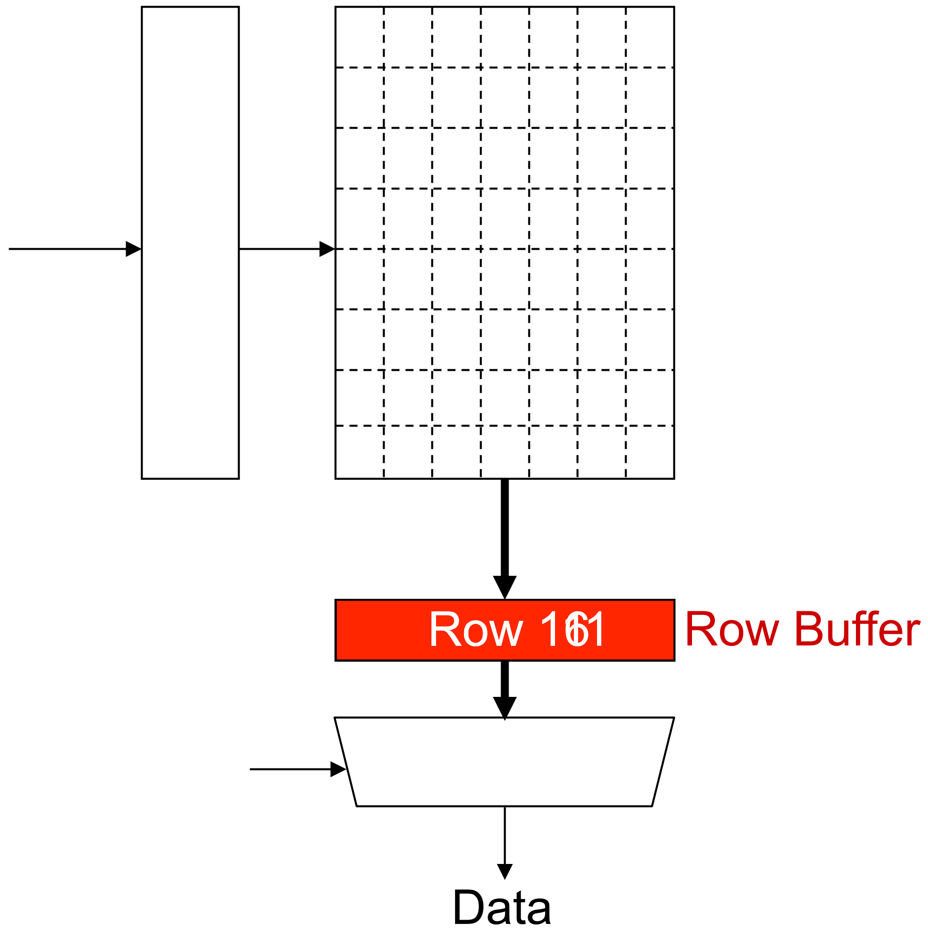
STFM Scheduling Algorithm [MICRO' 07]

- For each thread, the DRAM controller
  - Tracks ST_{shared}
  - Estimates ST_{alone}
- Each cycle, the DRAM controller
  - Computes Slowdown = ST_{shared}/ST_{alone} for threads with legal requests
  - Computes unfairness = MAX Slowdown / MIN Slowdown
- If unfairness $< \alpha$
  - Use DRAM throughput oriented scheduling policy
- If unfairness $\geq \alpha$
  - Use fairness-oriented scheduling policy
    - (1) requests from thread with MAX Slowdown first
    - (2) row-hit first , (3) oldest-first

How Does STFM Prevent Unfairness?

T0: Row 0
T1: Row 5
T0: Row 0
T1: Row 111
T0: Row 0
T1: Row 16

T0 Slowdown	1.04
T1 Slowdown	1.06
Unfairness	1.04
α	1.05



STFM Pros and Cons

- Upsides:
  - Identifies fairness as an issue in multi-core memory scheduling
  - Good at providing fairness
  - Being fair improves performance

- Downsides:
  - Does not handle all types of interference
  - Somewhat complex to implement
  - Slowdown estimations can be incorrect

Parallelism-Aware Batch Scheduling

Onur Mutlu and Thomas Moscibroda,
"Parallelism-Aware Batch Scheduling: Enhancing both Performance and Fairness of Shared DRAM Systems"
35th International Symposium on Computer Architecture (***ISCA***),
pages 63-74, Beijing, China, June 2008. Slides (ppt)

PAR-BS ISCA 2008 Talk