

# Seminar in Computer Architecture

## Meeting 3b: Example Paper Presentation

Prof. Onur Mutlu

ETH Zürich

Spring 2020

12 March 2020

- 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)*, pages 63-74, Beijing, China, June 2008.  
[\[Summary\]](#) [\[Slides \(ppt\)\]](#)

## **Parallelism-Aware Batch Scheduling:**

## **Enhancing both Performance and Fairness of Shared DRAM Systems**

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

# We Will Do This Differently

---

- I will give a “conference talk”
- You can ask questions and analyze what I described

# Parallelism-Aware Batch Scheduling

## Enhancing both Performance and Fairness of Shared DRAM Systems

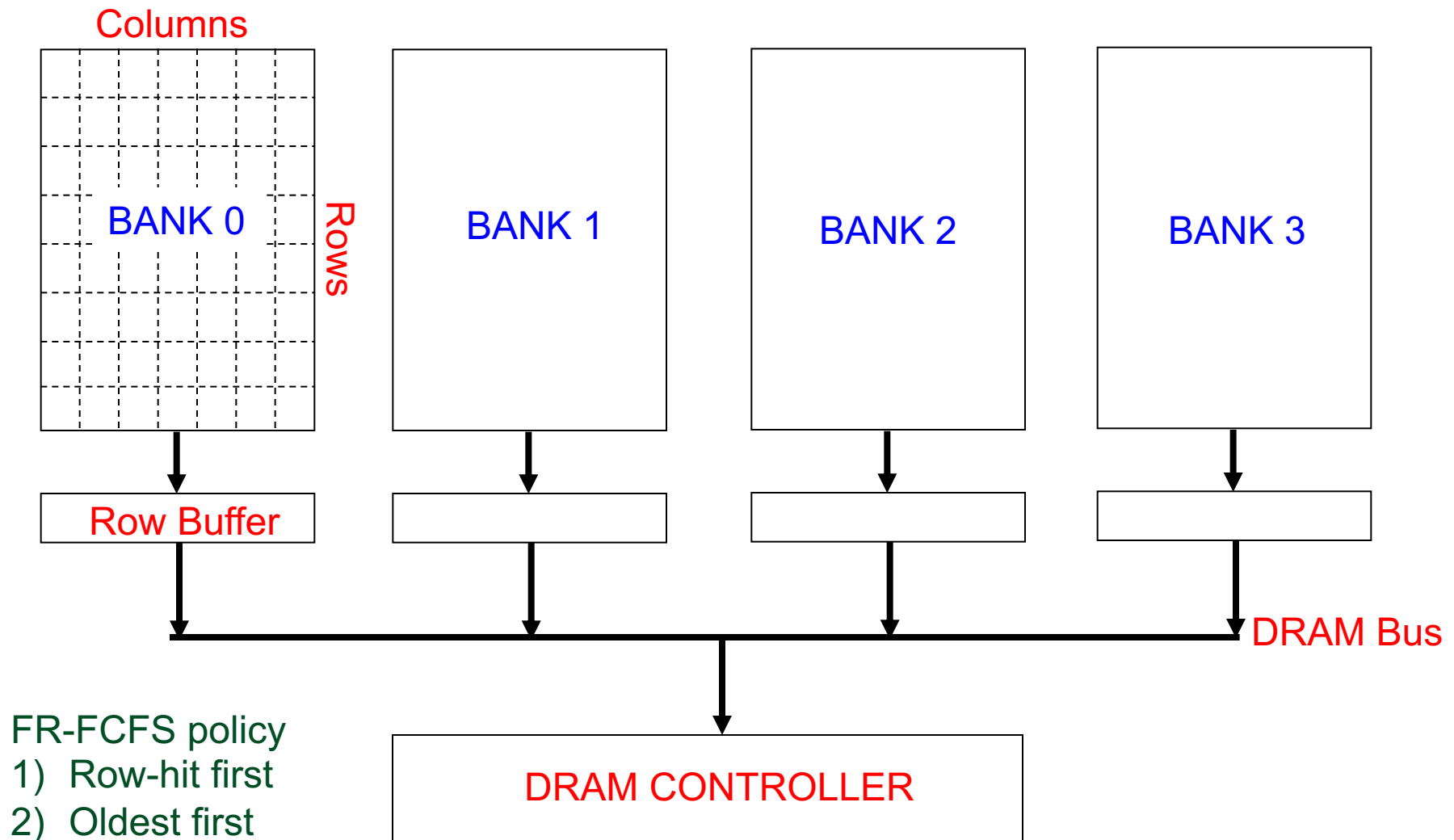
Onur Mutlu and Thomas Moscibroda  
Computer Architecture Group  
Microsoft Research

# Outline

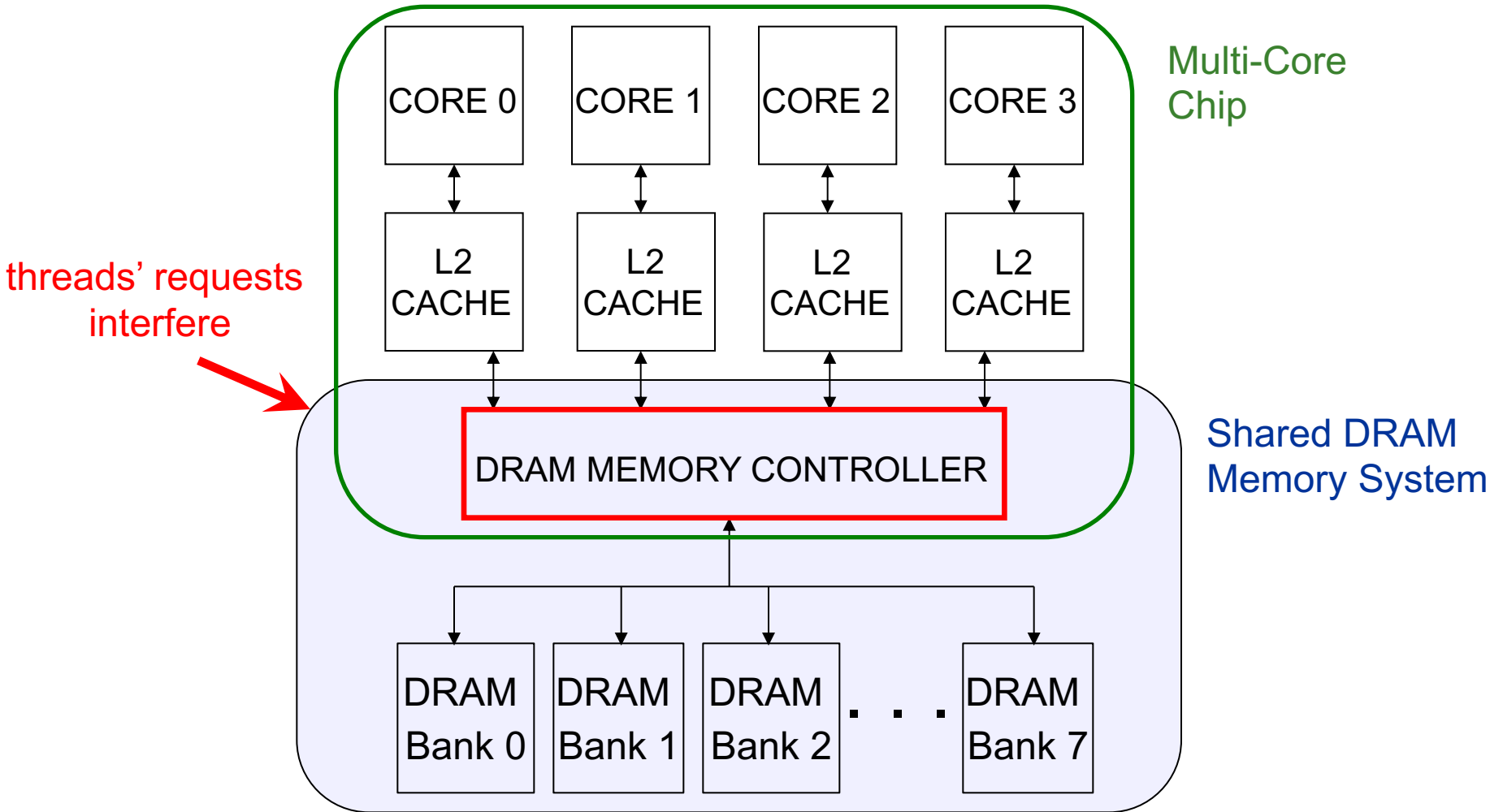
---

- Background and Goal
- Motivation
  - Destruction of Intra-thread DRAM Bank Parallelism
- Parallelism-Aware Batch Scheduling
  - Batching
  - Within-batch Scheduling
- System Software Support
- Evaluation
- Summary

# The DRAM System



# Multi-Core Systems



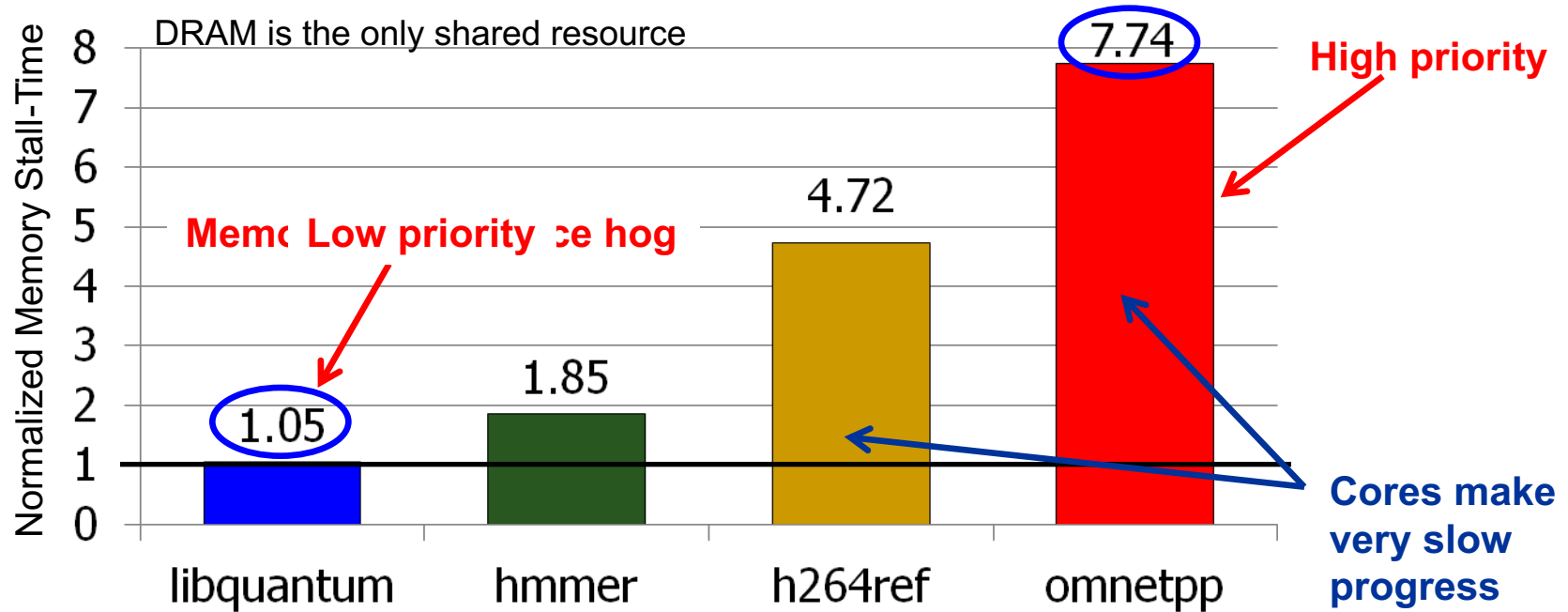
# Inter-thread Interference in the DRAM System

---

- Threads delay each other by causing resource contention:
  - Bank, bus, row-buffer conflicts [MICRO 2007]
- Threads can also destroy each other's **DRAM bank parallelism**
  - Otherwise parallel requests can become serialized
- Existing DRAM schedulers are unaware of this interference
- They simply aim to maximize DRAM throughput
  - Thread-unaware and thread-unfair
  - **No intent to service each thread's requests in parallel**
  - FR-FCFS policy: 1) row-hit first, 2) oldest first
    - Unfairly prioritizes threads with high row-buffer locality



# Consequences of Inter-Thread Interference in DRAM



- Unfair slowdown of different threads [MICRO 2007]
- System performance loss [MICRO 2007]
- Vulnerability to denial of service [USENIX Security 2007]
- Inability to enforce system-level thread priorities [MICRO 2007]

# Our Goal

---

- Control inter-thread interference in DRAM
- Design a shared DRAM scheduler that
  - provides **high system performance**
    - preserves each thread's **DRAM bank parallelism**
  - provides **fairness to threads** sharing the DRAM system
    - equalizes memory-slowdowns of equal-priority threads
  - is **controllable and configurable**
    - enables different service levels for threads with different priorities

# Outline

---

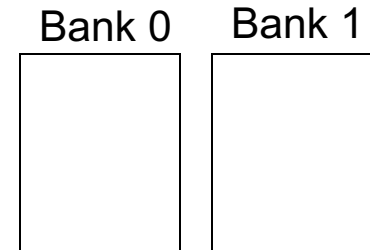
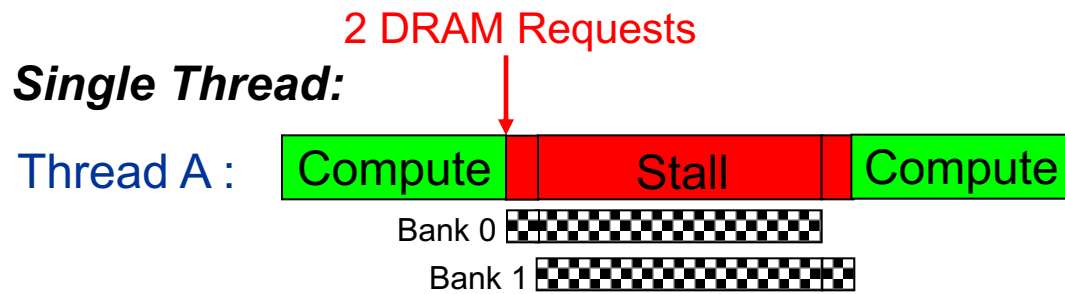
- Background and Goal
- Motivation
  - Destruction of Intra-thread DRAM Bank Parallelism
- Parallelism-Aware Batch Scheduling
  - Batching
  - Within-batch Scheduling
- System Software Support
- Evaluation
- Summary

# The Problem

---

- Processors try to tolerate the latency of DRAM requests by generating multiple outstanding requests
  - Memory-Level Parallelism (MLP)
  - Out-of-order execution, non-blocking caches, runahead execution
- Effective only if the DRAM controller actually services the multiple requests in parallel in DRAM banks
- Multiple threads share the DRAM controller
- DRAM controllers are not aware of a thread's MLP
  - Can service each thread's outstanding requests serially, not in parallel

# Bank Parallelism of a Thread



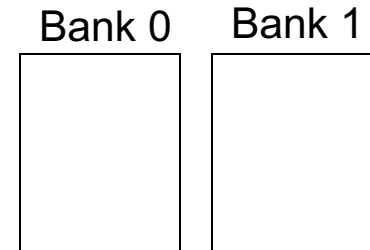
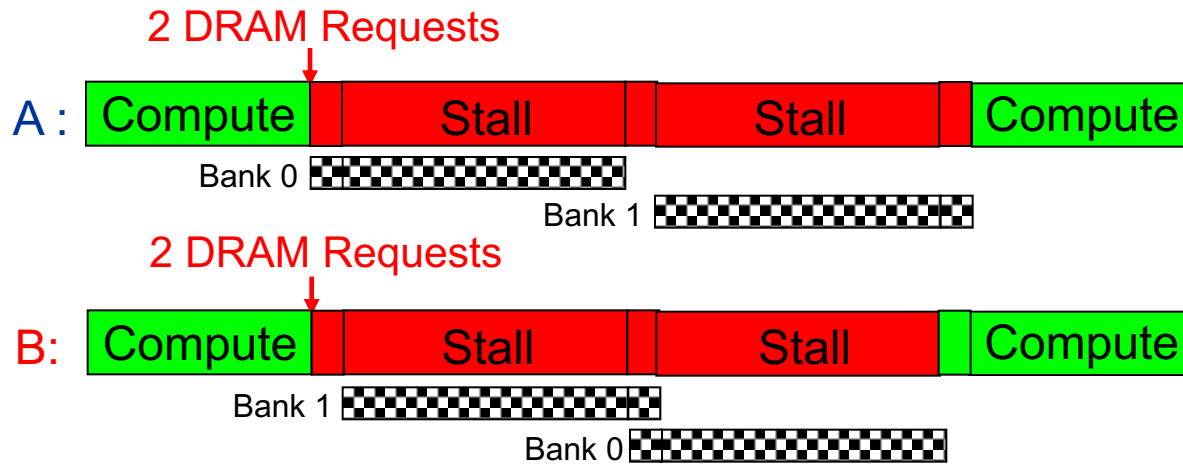
Thread A: Bank 0, Row 1

Thread A: Bank 1, Row 1

Bank access latencies of the two requests overlapped  
Thread stalls for ~ONE bank access latency

# Bank Parallelism Interference in DRAM

## Baseline Scheduler:



Thread A: Bank 0, Row 1

Thread B: Bank 1, Row 99

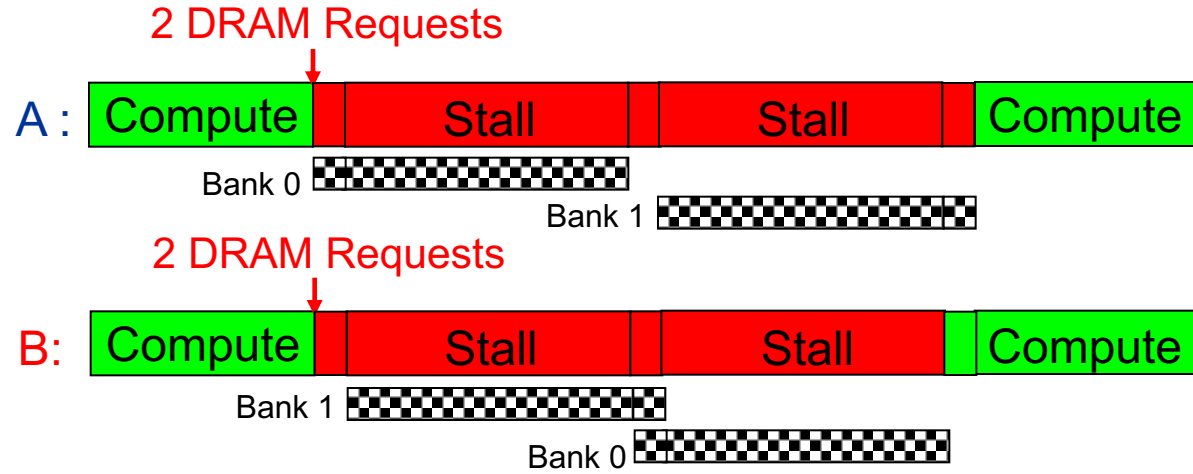
Thread B: Bank 0, Row 99

Thread A: Bank 1, Row 1

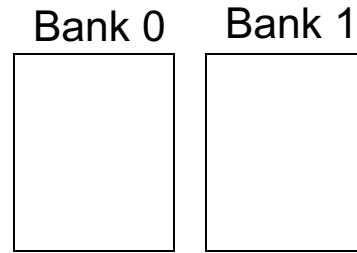
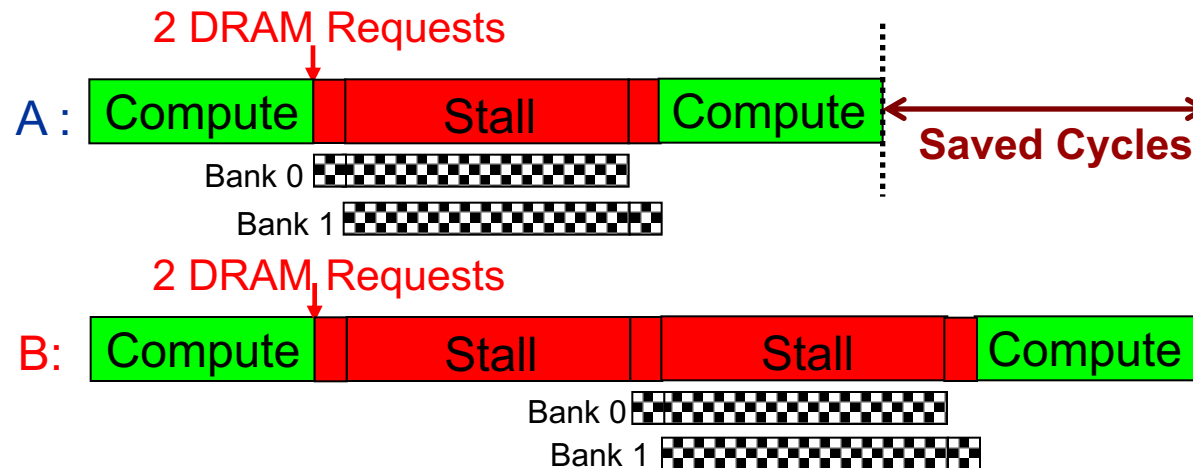
Bank access latencies of each thread serialized  
Each thread stalls for ~TWO bank access latencies

# Parallelism-Aware Scheduler

## Baseline Scheduler:



## Parallelism-aware Scheduler:



Thread A: Bank 0, Row 1

Thread B: Bank 1, Row 99

Thread B: Bank 0, Row 99

Thread A: Bank 1, Row 1

**Average stall-time:  
~1.5 bank access latencies**

# Outline

---

- Background and Goal
- Motivation
  - Destruction of Intra-thread DRAM Bank Parallelism
- **Parallelism-Aware Batch Scheduling (PAR-BS)**
  - Request Batching
  - Within-batch Scheduling
- System Software Support
- Evaluation
- Summary



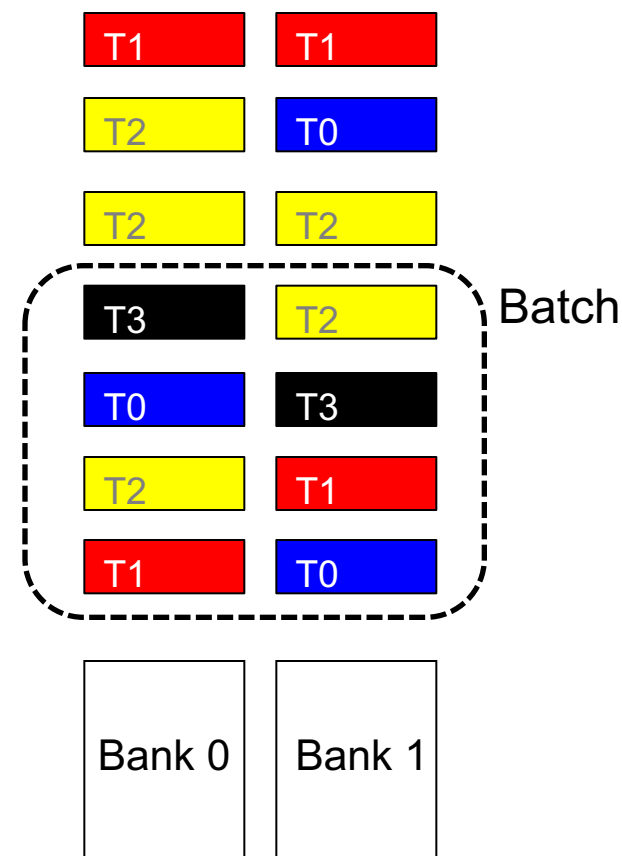
# Parallelism-Aware Batch Scheduling (PAR-BS)

## ■ Principle 1: Parallelism-awareness

- ❑ Schedule requests from a thread (to different banks) back to back
- ❑ Preserves each thread's bank parallelism
- ❑ But, this can cause starvation...

## ■ Principle 2: Request Batching

- ❑ Group a fixed number of oldest requests from each thread into a "batch"
- ❑ Service the batch before all other requests
- ❑ Form a new batch when the current one is done
- ❑ Eliminates starvation, provides fairness
- ❑ Allows parallelism-awareness within a batch



# PAR-BS Components

---

- Request batching
- Within-batch scheduling
  - Parallelism aware


# Request Batching

---

- Each memory request has a bit (*marked*) associated with it
- Batch formation:
  - Mark up to *Marking-Cap* oldest requests per bank for each thread
  - Marked requests constitute the batch
  - Form a new batch when no marked requests are left
- Marked requests are prioritized over unmarked ones
  - No reordering of requests across batches: **no starvation, high fairness**
- **How to prioritize requests within a batch?**

# Within-Batch Scheduling

---

- Can use any existing DRAM scheduling policy
  - FR-FCFS (row-hit first, then oldest-first) exploits row-buffer locality
- But, we also want to preserve intra-thread bank parallelism
  - Service each thread's requests back to back
- Scheduler computes a ~~row~~ **ranking of threads** when the batch is formed 
  - Higher-ranked threads are prioritized over lower-ranked ones
  - Improves the likelihood that requests from a thread are serviced in parallel by different banks
    - Different threads prioritized in the same order across ALL banks

# How to Rank Threads within a Batch

---

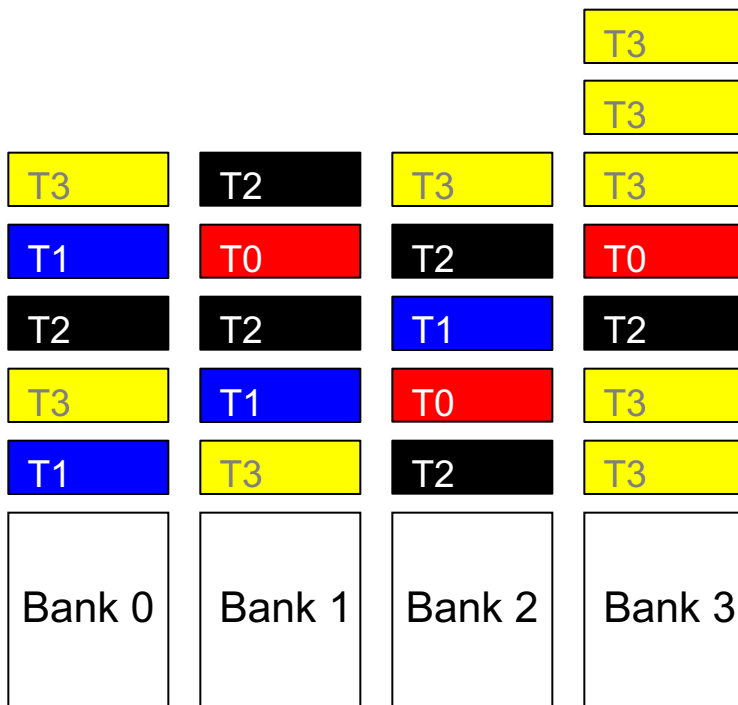
- Ranking scheme affects system throughput and fairness
- Maximize system throughput
  - Minimize average stall-time of threads within the batch
- Minimize unfairness (Equalize the slowdown of threads)
  - Service threads with inherently low stall-time early in the batch
  - Insight: delaying memory non-intensive threads results in high slowdown
- Shortest stall-time first (shortest job first) ranking
  - Provides optimal system throughput [Smith, 1956]\*
  - Controller estimates each thread's stall-time within the batch
  - Ranks threads with shorter stall-time higher

---

\* W.E. Smith, "Various optimizers for single stage production," Naval Research Logistics Quarterly, 1956.

# Shortest Stall-Time First Ranking

- Maximum number of marked requests to any bank (max-bank-load)
  - Rank thread with lower max-bank-load higher (~ low stall-time)
- Total number of marked requests (total-load)
  - Breaks ties: rank thread with lower total-load higher

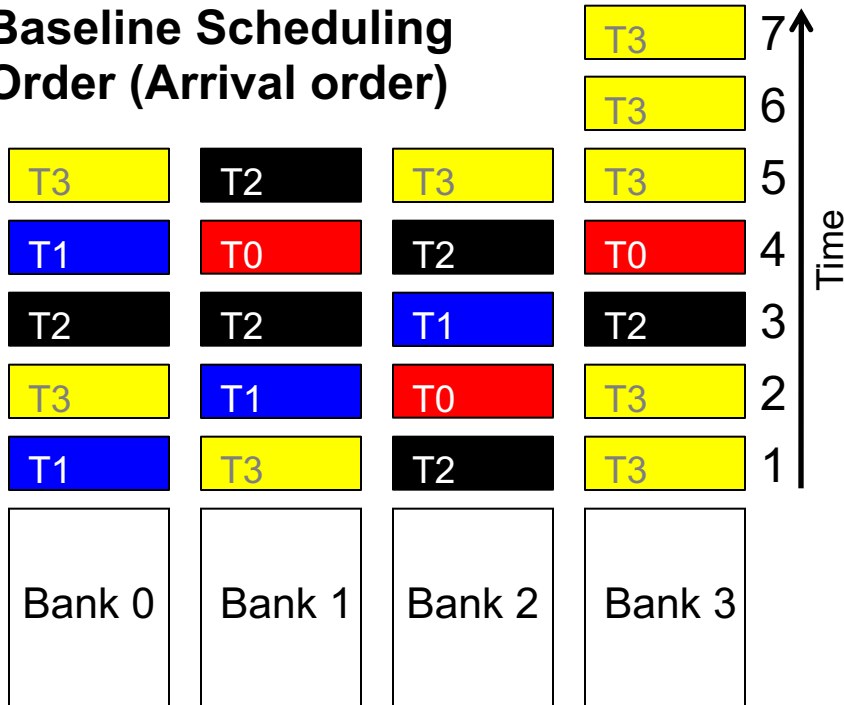


	max-bank-load	total-load

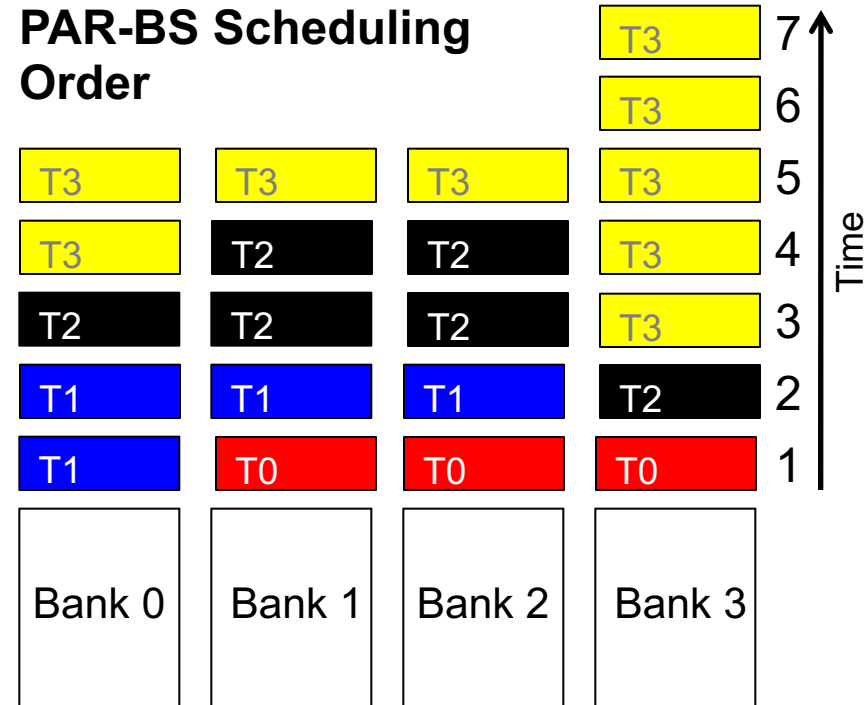
**Ranking:**  
**T0 > T1 > T2 > T3**

# Example Within-Batch Scheduling Order

**Baseline Scheduling Order (Arrival order)**



**PAR-BS Scheduling Order**



**Ranking: T0 > T1 > T2 > T3**

	<b>T0</b>	<b>T1</b>	<b>T2</b>	<b>T3</b>
Stall times				

**AVG: 5 bank access latencies**

	<b>T0</b>	<b>T1</b>	<b>T2</b>	<b>T3</b>
Stall times				

**AVG: 3.5 bank access latencies**

# Putting It Together: PAR-BS Scheduling Policy

---

## ■ PAR-BS Scheduling Policy

(1) Marked requests first

Batching

(2) Row-hit requests first

(3) Higher-rank thread first (shortest stall-time first)

Parallelism-aware  
within-batch  
scheduling

(4) Oldest first

## ■ Three properties:

- Exploits row-buffer locality **and** intra-thread bank parallelism
- Work-conserving
  - Services unmarked requests to banks without marked requests
- Marking-Cap is important
  - Too small cap: destroys row-buffer locality
  - Too large cap: penalizes memory non-intensive threads

## ■ Many more trade-offs analyzed in the paper



# Hardware Cost

---

- <1.5KB storage cost for
  - 8-core system with 128-entry memory request buffer
- No complex operations (e.g., divisions)
- Not on the critical path
  - Scheduler makes a decision only every DRAM cycle

# Outline

---

- Background and Goal
- Motivation
  - Destruction of Intra-thread DRAM Bank Parallelism
- Parallelism-Aware Batch Scheduling
  - Batching
  - Within-batch Scheduling
- System Software Support
- Evaluation
- Summary

# System Software Support

---

- OS conveys **each thread's priority level** to the controller
  - Levels 1, 2, 3, ... (highest to lowest priority)
- **Controller enforces priorities** in two ways
  - Mark requests from a thread with priority X only every Xth batch
  - Within a batch, higher-priority threads' requests are scheduled first
- **Purely opportunistic service**
  - Special very low priority level L
  - Requests from such threads never marked
- Quantitative analysis in paper

# Outline

---

- Background and Goal
- Motivation
  - Destruction of Intra-thread DRAM Bank Parallelism
- Parallelism-Aware Batch Scheduling
  - Batching
  - Within-batch Scheduling
- System Software Support
- Evaluation
- Summary

# Evaluation Methodology

---

- 4-, 8-, 16-core systems
  - x86 processor model based on Intel Pentium M
  - 4 GHz processor, 128-entry instruction window
  - 512 Kbyte per core private L2 caches, 32 L2 miss buffers
- Detailed DRAM model based on Micron DDR2-800
  - 128-entry memory request buffer
  - 8 banks, 2Kbyte row buffer
  - 40ns (160 cycles) row-hit round-trip latency
  - 80ns (320 cycles) row-conflict round-trip latency
- Benchmarks
  - Multiprogrammed SPEC CPU2006 and Windows Desktop applications
  - 100, 16, 12 program combinations for 4-, 8-, 16-core experiments

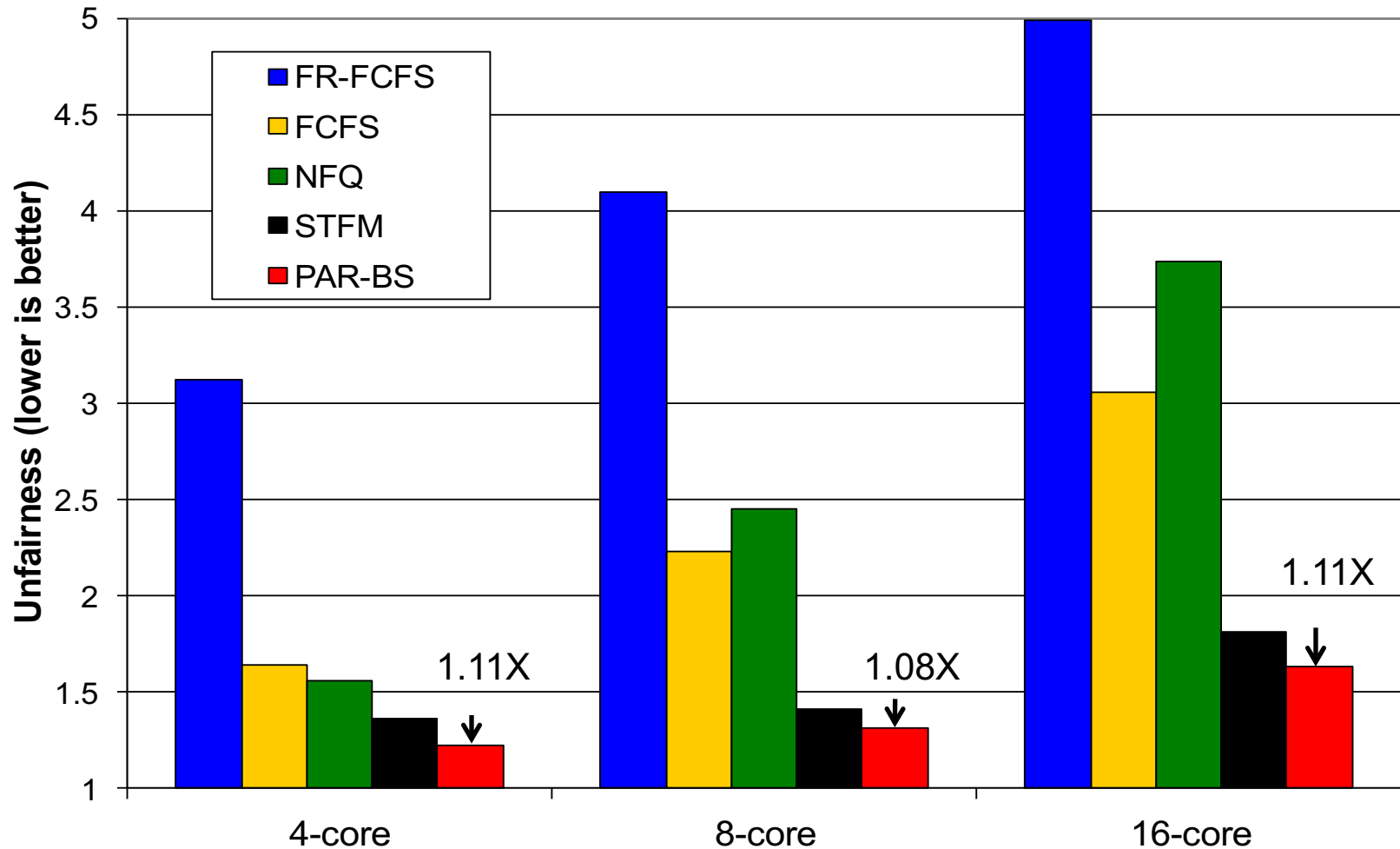
# Comparison with Other DRAM Controllers

---

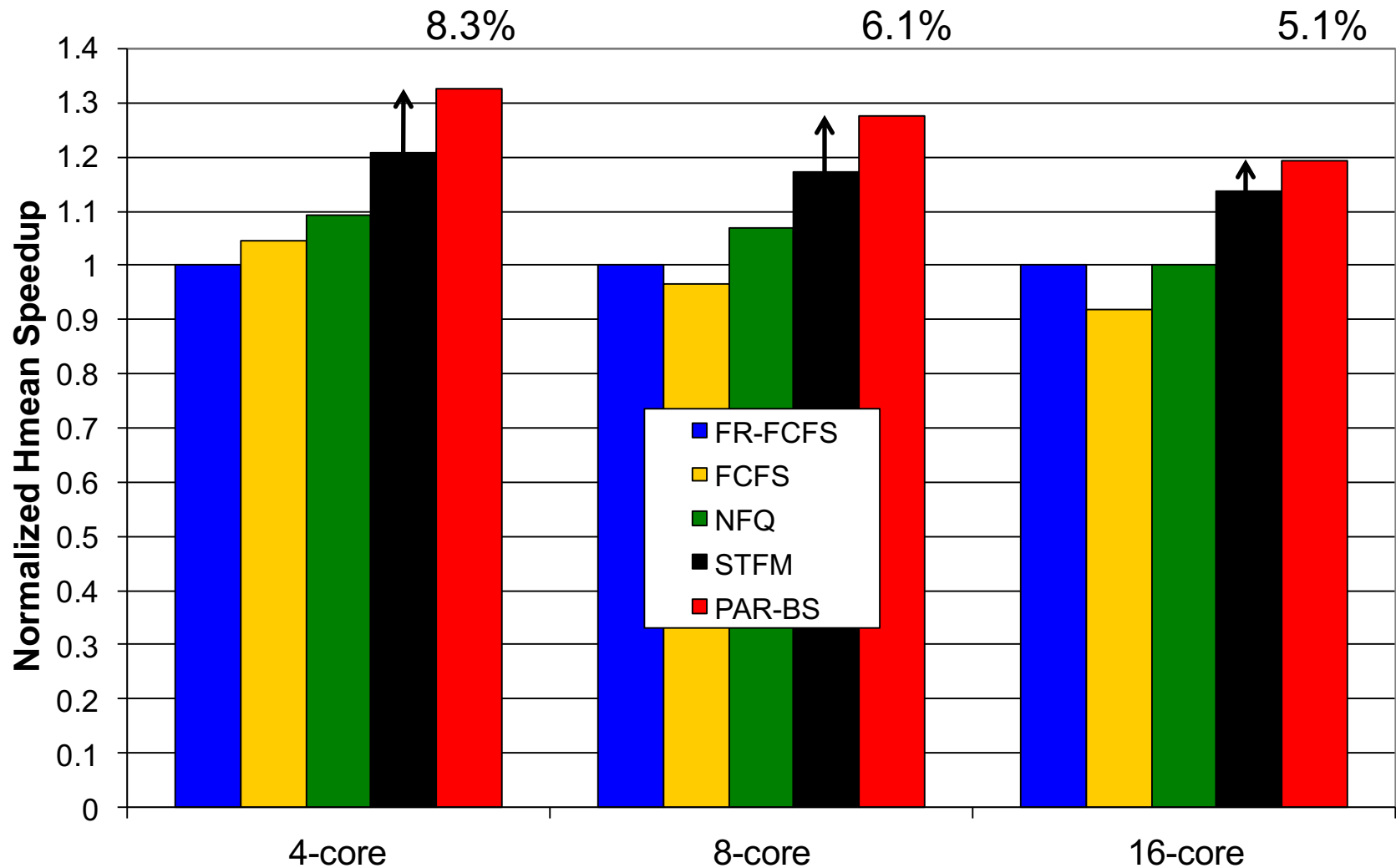
- **Baseline FR-FCFS** [Zuravleff and Robinson, US Patent 1997; Rixner et al., ISCA 2000]
  - Prioritizes row-hit requests, older requests
  - Unfairly penalizes threads with low row-buffer locality, memory non-intensive threads
- **FCFS** [Intel Pentium 4 chipsets]
  - Oldest-first; low DRAM throughput
  - Unfairly penalizes memory non-intensive threads
- **Network Fair Queueing (NFQ)** [Nesbit et al., MICRO 2006]
  - Equally partitions DRAM bandwidth among threads
  - Does not consider inherent (baseline) DRAM performance of each thread
  - Unfairly penalizes threads with high bandwidth utilization [MICRO 2007]
  - Unfairly prioritizes threads with bursty access patterns [MICRO 2007]
- **Stall-Time Fair Memory Scheduler (STFM)** [Mutlu & Moscibroda, MICRO 2007]
  - Estimates and balances thread slowdowns relative to when run alone
  - Unfairly treats threads with inaccurate slowdown estimates
  - Requires multiple (approximate) arithmetic operations

# Unfairness on 4-, 8-, 16-core Systems

Unfairness = MAX Memory Slowdown / MIN Memory Slowdown [MICRO 2007]



# System Performance (Hmean-speedup)





# Outline

---

- Background and Goal
- Motivation
  - Destruction of Intra-thread DRAM Bank Parallelism
- Parallelism-Aware Batch Scheduling
  - Batching
  - Within-batch Scheduling
- System Software Support
- Evaluation
- Summary

# Summary

---

- Inter-thread interference can destroy each thread's DRAM bank parallelism
  - ❑ Serializes a thread's requests → reduces system throughput
  - ❑ Makes techniques that exploit memory-level parallelism less effective
  - ❑ Existing DRAM controllers unaware of intra-thread bank parallelism
- A new approach to fair and high-performance DRAM scheduling
  - ❑ **Batching**: Eliminates starvation, allows fair sharing of the DRAM system
  - ❑ **Parallelism-aware thread ranking**: Preserves each thread's bank parallelism
  - ❑ **Flexible and configurable**: Supports system-level thread priorities → QoS policies
- **PAR-BS provides better fairness and system performance than previous DRAM schedulers**

Thank you. Questions?

# Parallelism-Aware Batch Scheduling

## Enhancing both Performance and Fairness of Shared DRAM Systems

Onur Mutlu and Thomas Moscibroda  
Computer Architecture Group  
Microsoft Research

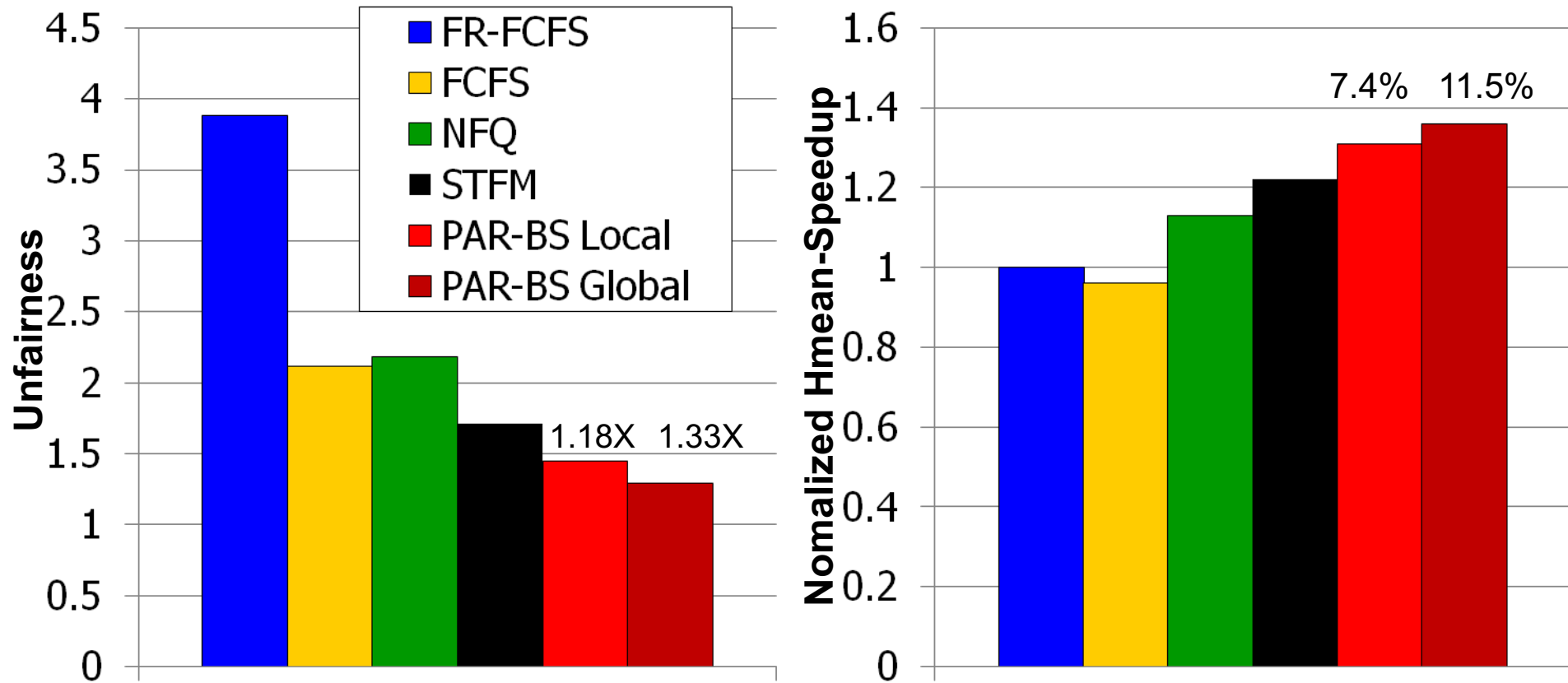
# Backup Slides

# Multiple Memory Controllers (I)

---

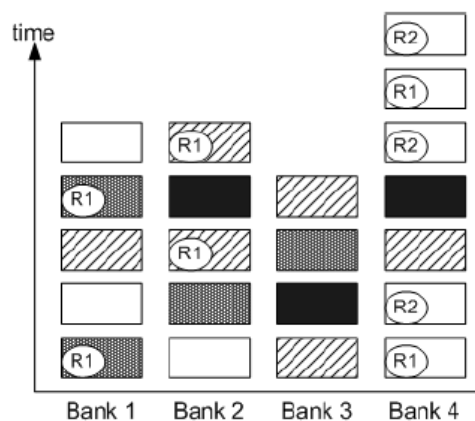
- **Local ranking:** Each controller uses PAR-BS independently
  - Computes its own ranking based on its local requests
- **Global ranking:** Meta controller that computes a global ranking across all controllers based on global information
  - Only needs to track bookkeeping info about each thread's requests to the banks in each controller
- The difference between the ranking computed by each scheme depends on the balance of the distribution of requests to each controller
  - Balanced → Local and global rankings are similar

# Multiple Memory Controllers (II)

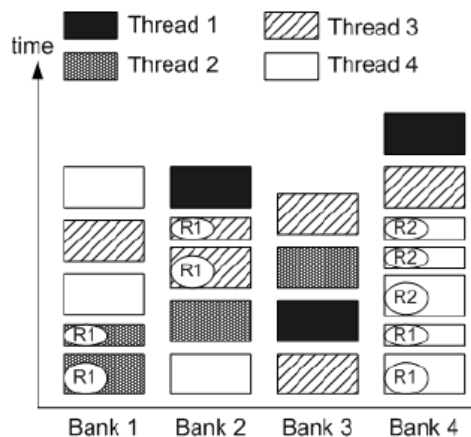


16-core system, 4 memory controllers

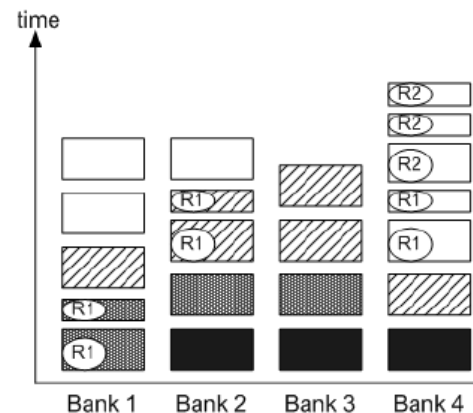
# Example with Row Hits



(a) Arrival order (and FCFS schedule)



(b) FR-FCFS schedule



(c) PAR-BS schedule

	Stall time		Stall time		Stall time
Thread 1	4	Thread 1	5.5	Thread 1	1
Thread 2	4	Thread 2	3	Thread 2	2
Thread 3	5	Thread 3	4.5	Thread 3	4
Thread 4	7	Thread 4	4.5	Thread 4	5.5
<b>AVG</b>	<b>5</b>	<b>AVG</b>	<b>4.375</b>	<b>AVG</b>	<b>3.125</b>



End of Backup Slides

# Now Your Turn to Analyze...

---

- Background, Problem & Goal
- Novelty
- Key Approach and Ideas
- Mechanisms (in some detail)
- Key Results: Methodology and Evaluation
- Summary
- Strengths
- Weaknesses
- Thoughts and Ideas
- Takeaways
- Open Discussion

# PAR-BS Pros and Cons

---

- Upsides:
  - ❑ First scheduler to address bank parallelism destruction across multiple threads
  - ❑ Simple mechanism (vs. STFM)
  - ❑ Batching provides fairness
  - ❑ Ranking enables parallelism awareness
- Downsides:
  - ❑ Does not always prioritize the latency-sensitive applications
  - ❑ Deadline guarantees?
  - ❑ Complexity?
- Some ideas implemented in real SoC memory controllers

# More on PAR-BS

---

- 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)*, pages 63-74, Beijing, China, June 2008.  
[\[Summary\]](#) [\[Slides \(ppt\)\]](#)

## **Parallelism-Aware Batch Scheduling:**

## **Enhancing both Performance and Fairness of Shared DRAM Systems**

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

# Seminar in Computer Architecture

## Meeting 3b: Example Paper Presentation

Prof. Onur Mutlu

ETH Zürich

Spring 2020

12 March 2020