15-740/18-740
Computer Architecture
Lecture 20: Main Memory II

Prof. Onur Mutlu

Carnegie Mellon University

Today

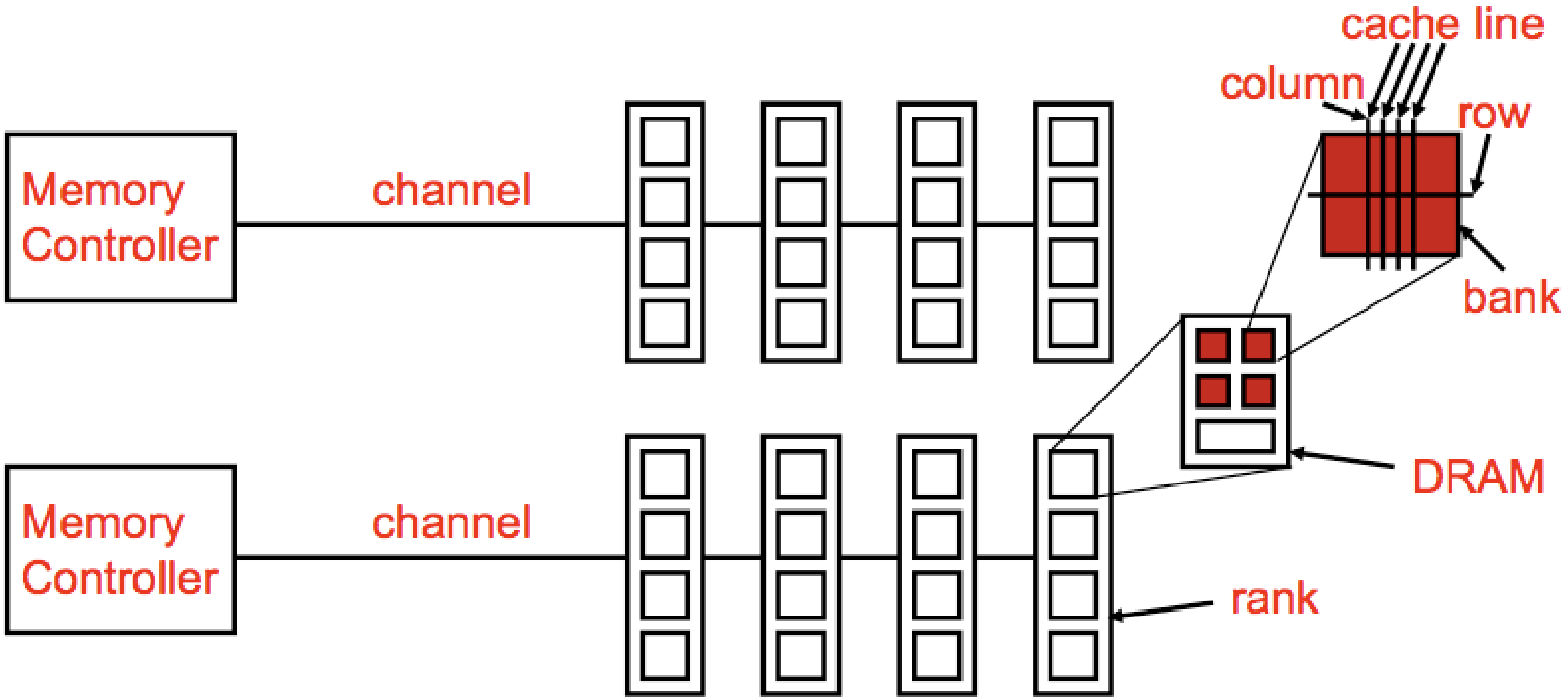
- SRAM vs. DRAM
- Interleaving/Banking
- DRAM Microarchitecture
  - Memory controller
  - Memory buses
  - Banks, ranks, channels, DIMMs
  - Address mapping: software vs. hardware
  - DRAM refresh
- Memory scheduling policies
- Memory power/energy management
- Multi-core issues
  - Fairness, interference
  - Large DRAM capacity

Readings

- Required:
  - Mutlu and Moscibroda, "Parallelism-Aware Batch Scheduling: Enabling High-Performance and Fair Memory Controllers," IEEE Micro Top Picks 2009.
  - Mutlu and Moscibroda, "Stall-Time Fair Memory Access Scheduling for Chip Multiprocessors," MICRO 2007.

- Recommended:
  - Zhang et al., "A Permutation-based Page Interleaving Scheme to Reduce Row-buffer Conflicts and Exploit Data Locality," MICRO 2000.
  - Lee et al., "Prefetch-Aware DRAM Controllers," MICRO 2008.
  - Rixner et al., "Memory Access Scheduling," ISCA 2000.

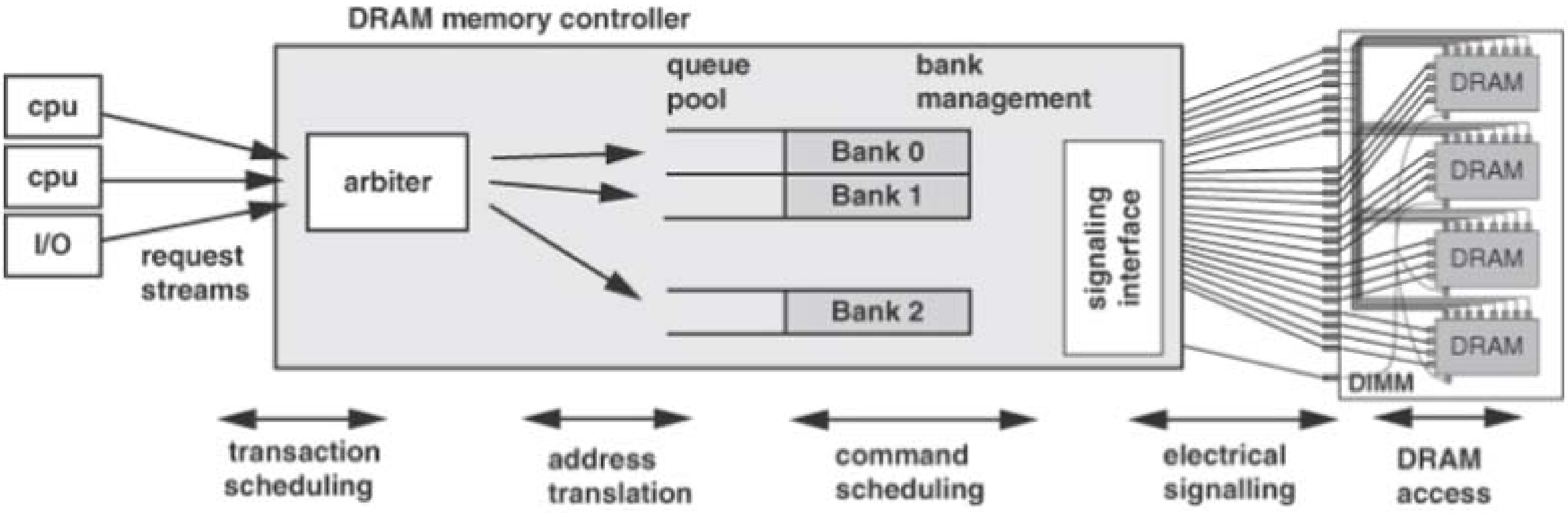
Review: Generalized Memory Structure



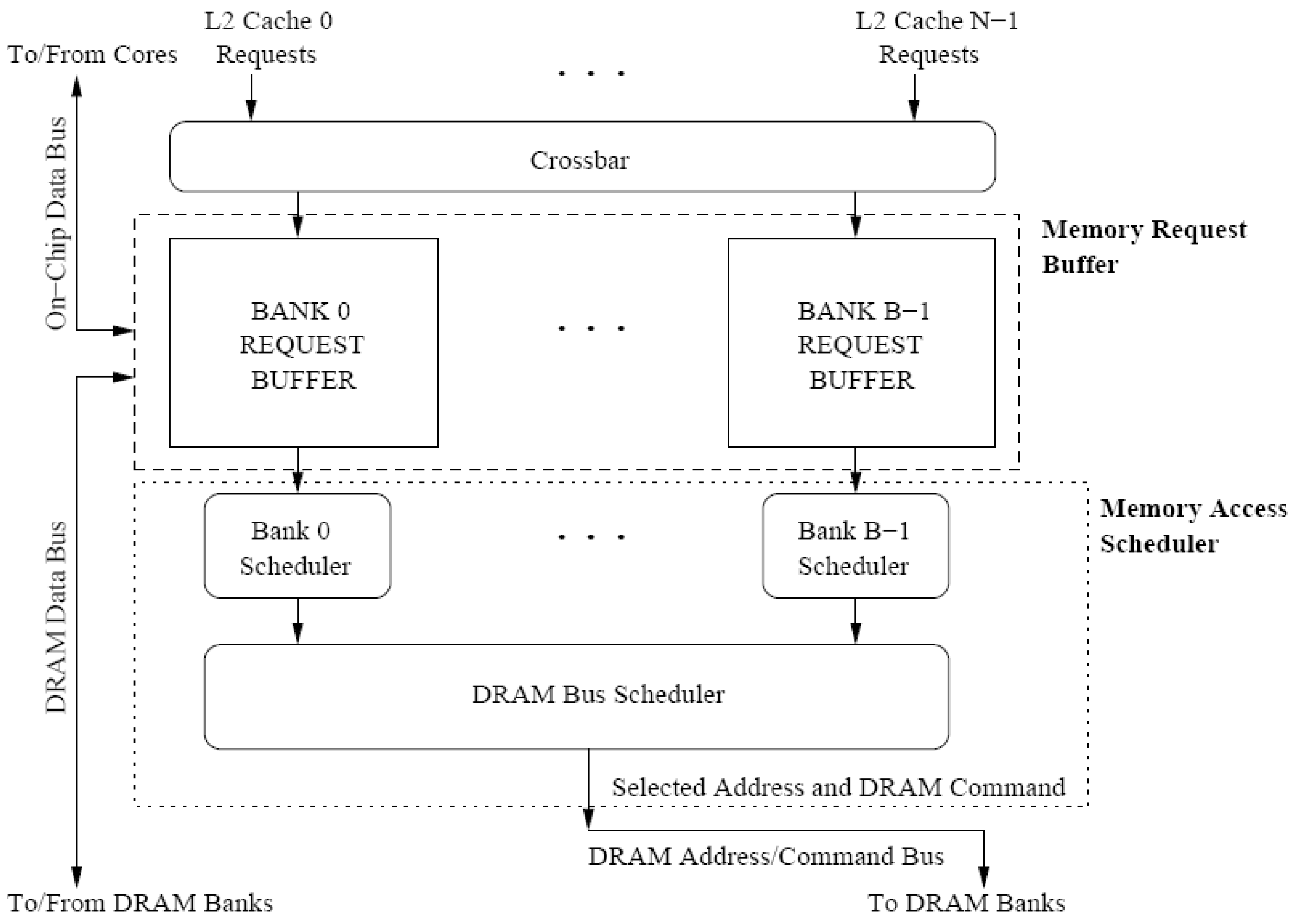
Review: DRAM Controller

- Purpose and functions
  - Ensure correct operation of DRAM (refresh)
  - Service DRAM requests while obeying timing constraints of DRAM chips
    - Constraints: resource conflicts (bank, bus, channel), minimum write-to-read delays
    - Translate requests to DRAM command sequences
  - Buffer and schedule requests to improve performance
    - Reordering and row-buffer management
  - Manage power consumption and thermals in DRAM
    - Turn on/off DRAM chips, manage power modes

DRAM Controller (II)



A Modern DRAM Controller



DRAM Scheduling Policies (I)

- FCFS (first come first served)
  - Oldest request first

- FR-FCFS (first ready, first come first served)

  1. Row-hit first
  2. Oldest first

  Goal: Maximize row buffer hit rate → maximize DRAM throughput

  - Actually, scheduling is done at the command level
    - Column commands (read/write) prioritized over row commands (activate/precharge)
    - Within each group, older commands prioritized over younger ones

DRAM Scheduling Policies (II)

- A scheduling policy is essentially a prioritization order

- Prioritization can be based on
  - Request age
  - Row buffer hit/miss status
  - Request type (prefetch, read, write)
  - Requestor type (load miss or store miss)
  - Request criticality
    - Oldest miss in the core?
    - How many instructions in core are dependent on it?

Row Buffer Management Policies

- Open row
  - Keep the row open after an access

  + Next access might need the same row → row hit

  -- Next access might need a different row → row conflict, wasted energy

- Closed row
  - Close the row after an access (if no other requests already in the request buffer need the same row)

  + Next access might need a different row → avoid a row conflict

  -- Next access might need the same row → extra activate latency

- Adaptive policies
  - Predict whether or not the next access to the bank will be to the same row

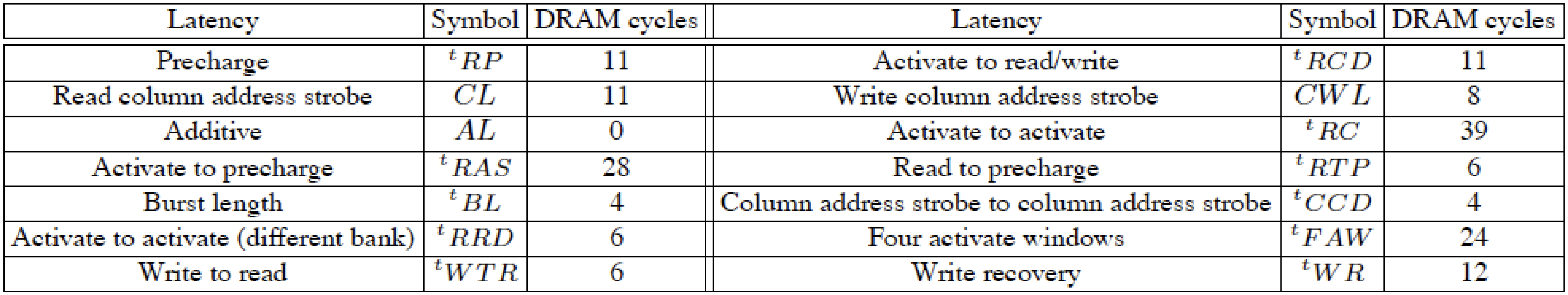
Open vs. Closed Row Policies

Policy	First access	Next access	Commands needed for next access
Open row	Row 0	Row 0 (row hit)	Read
Open row	Row 0	Row 1 (row conflict)	Precharge + Activate Row 1 + Read
Closed row	Row 0	Row 0 – access in request buffer (row hit)	Read
Closed row	Row 0	Row 0 – access not in request buffer (row closed)	Activate Row 0 + Read + Precharge
Closed row	Row 0	Row 1 (row closed)	Activate Row 1 + Read + Precharge

Why are DRAM Controllers Difficult to Design?

- Need to obey DRAM timing constraints for correctness
  - There are many (50+) timing constraints in DRAM
  - tWTR: Minimum number of cycles to wait before issuing a read command after a write command is issued
  - tRC: Minimum number of cycles between the issuing of two consecutive activate commands to the same bank
  - ...
- Need to keep track of many resources to prevent conflicts
  - Channels, banks, ranks, data bus, address bus, row buffers
- Need to handle DRAM refresh
- Need to optimize for performance (in the presence of constraints)
  - Reordering is not simple
  - Predicting the future?

Why are DRAM Controllers Difficult to Design?



Latency	Symbol	DRAM cycles	Latency	Symbol	DRAM cycles
Precharge	^{t}RP	11	Activate to read/write	^{t}RCD	11
Read column address strobe	CL	11	Write column address strobe	CWL	8
Additive	AL	0	Activate to activate	^{t}RC	39
Activate to precharge	^{t}RAS	28	Read to precharge	^{t}RTP	6
Burst length	^{t}BL	4	Column address strobe to column address strobe	^{t}CCD	4
Activate to activate (different bank)	^{t}RRD	6	Four activate windows	^{t}FAW	24
Write to read	^{t}WTR	6	Write recovery	^{t}WR	12

Table 4. DDR3 1600 DRAM timing specifications

- From Lee et al., "DRAM-Aware Last-Level Cache Writeback: Reducing Write-Caused Interference in Memory Systems," HPS Technical Report, April 2010.

DRAM Power Management

- DRAM chips have power modes
- Idea: When not accessing a chip power it down

- Power states
  - Active (highest power)
  - All banks idle
  - Power-down
  - Self-refresh (lowest power)

- State transitions incur latency during which the chip cannot be accessed

Multi-Core Issues (I)

- Memory controllers, pins, and memory banks are shared

- Pin bandwidth is not increasing as fast as number of cores
  - Bandwidth per core reducing

- Different threads executing on different cores interfere with each other in the main memory system

- Threads delay each other by causing resource contention:
  - Bank, bus, row-buffer conflicts → reduced DRAM throughput
- Threads can also destroy each other's DRAM bank parallelism
  - Otherwise parallel requests can become serialized

Effects of Inter-Thread Interference in DRAM

- Queueing/contention delays
  - Bank conflict, bus conflict, channel conflict, ...

- Additional delays due to DRAM constraints
  - Called “protocol overhead”
  - Examples
    - Row conflicts
    - Read-to-write and write-to-read delays

- Loss of intra-thread parallelism

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, ISCA'00]

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

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

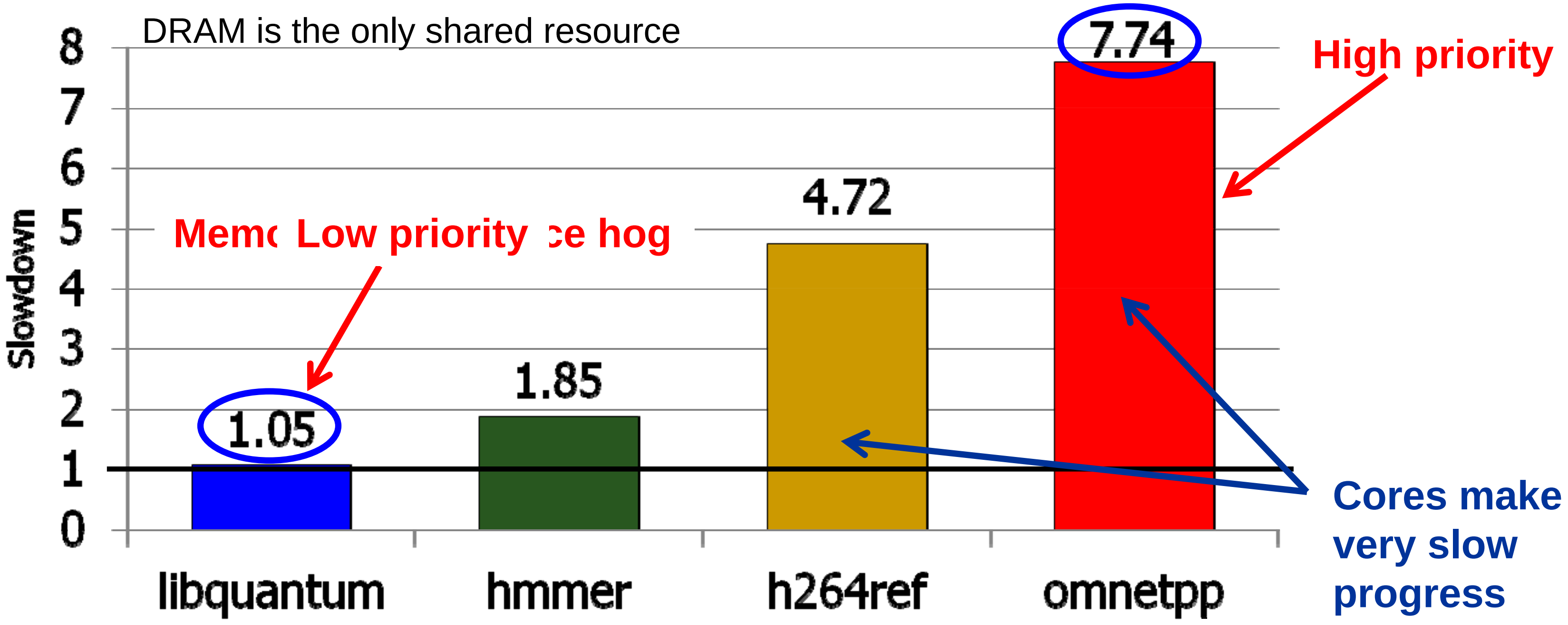
- This scheduling policy aims to maximize DRAM throughput
  - But, it is unfair when multiple threads share the DRAM system

Inter-Thread Interference in DRAM

- Multiple threads share the DRAM controller
- DRAM controllers are designed to maximize DRAM throughput
- Existing DRAM controllers are unaware of inter-thread interference in DRAM system

- DRAM scheduling policies are thread-unaware and unfair
  - Row-hit first: unfairly prioritizes threads with high row buffer locality
    - Streaming threads
    - Threads that keep on accessing the same row
  - Oldest-first: unfairly prioritizes memory-intensive threads

Consequences of Inter-Thread Interference in DRAM



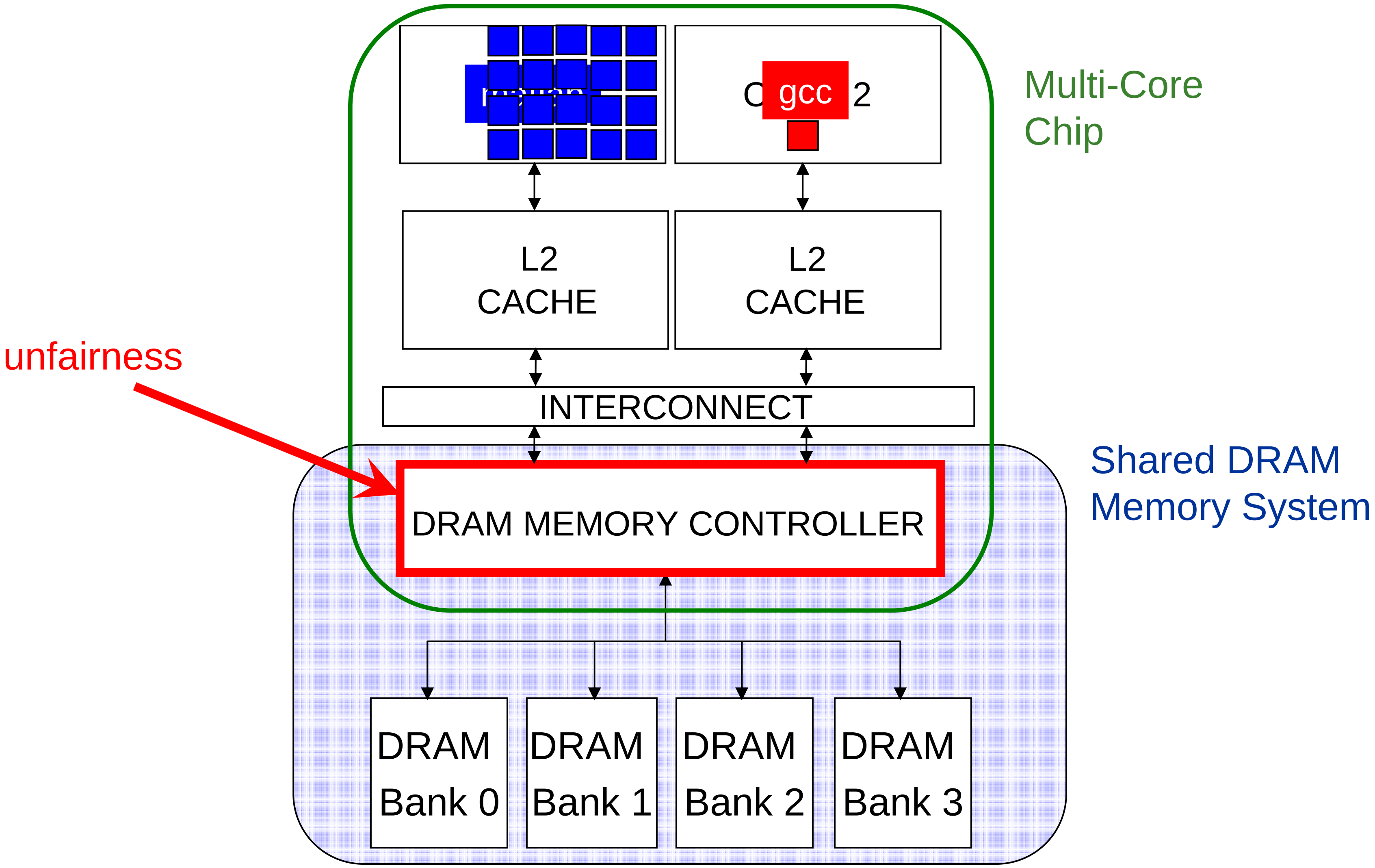
High priority

Memc Low priority ce hog

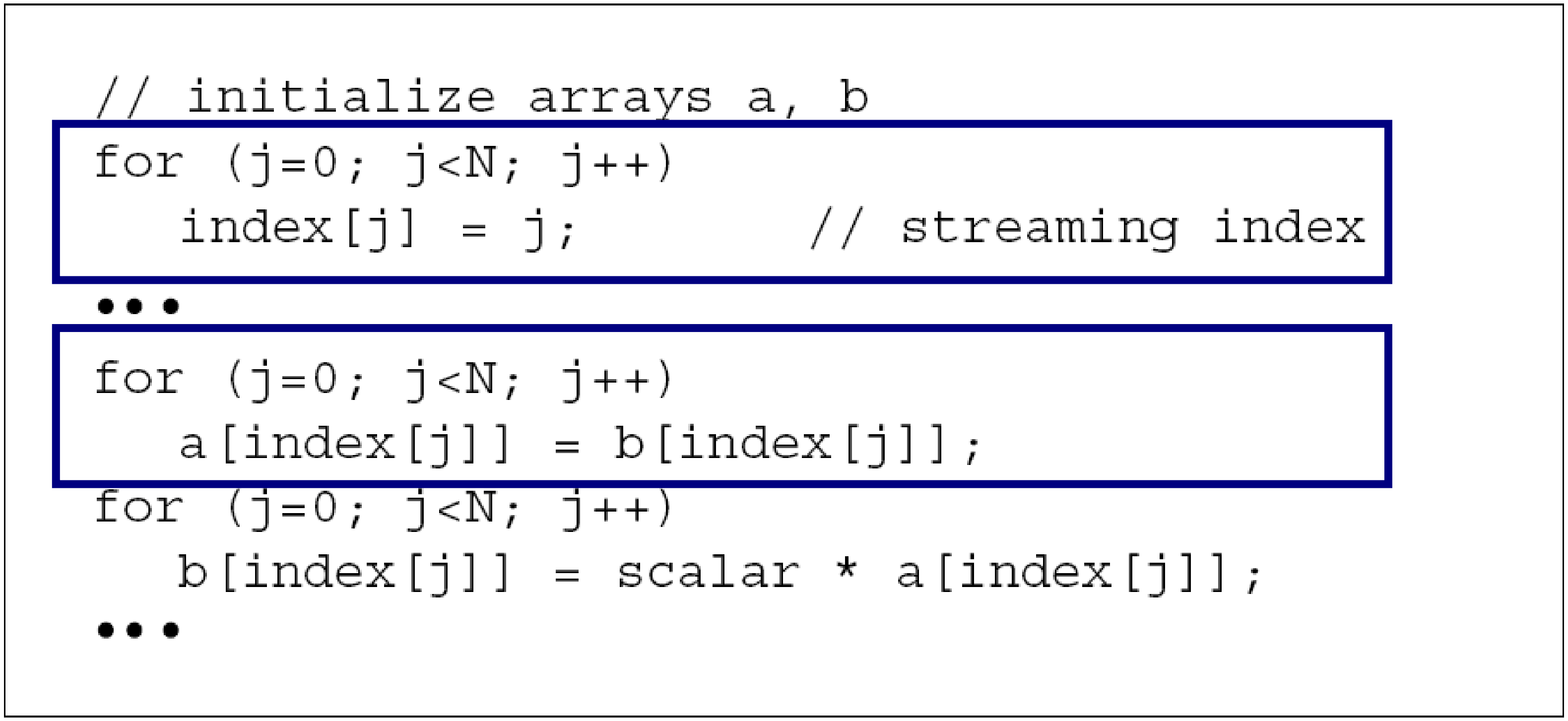
Cores make very slow progress

- Unfair slowdown of different threads
- System performance loss
- Vulnerability to denial of service
- Inability to enforce system-level thread priorities

Why the Disparity in Slowdowns?



An Example Memory Performance Hog

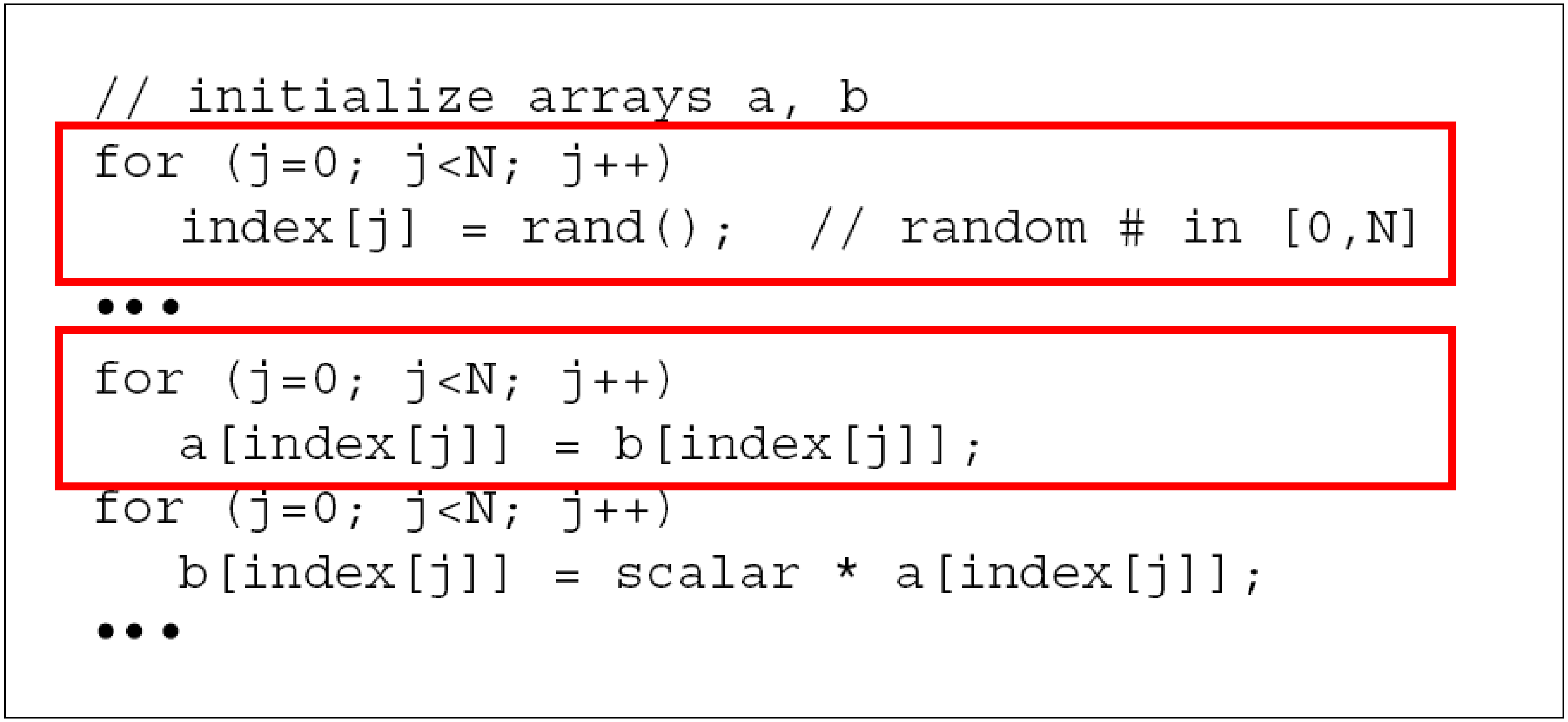


```
// initialize arrays a, b
for (j=0; j<N; j++)
   index[j] = j;          // streaming index
•••
for (j=0; j<N; j++)
   a[index[j]] = b[index[j]];
for (j=0; j<N; j++)
   b[index[j]] = scalar * a[index[j]];
•••
```

STREAM

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

A Co-Scheduled Application

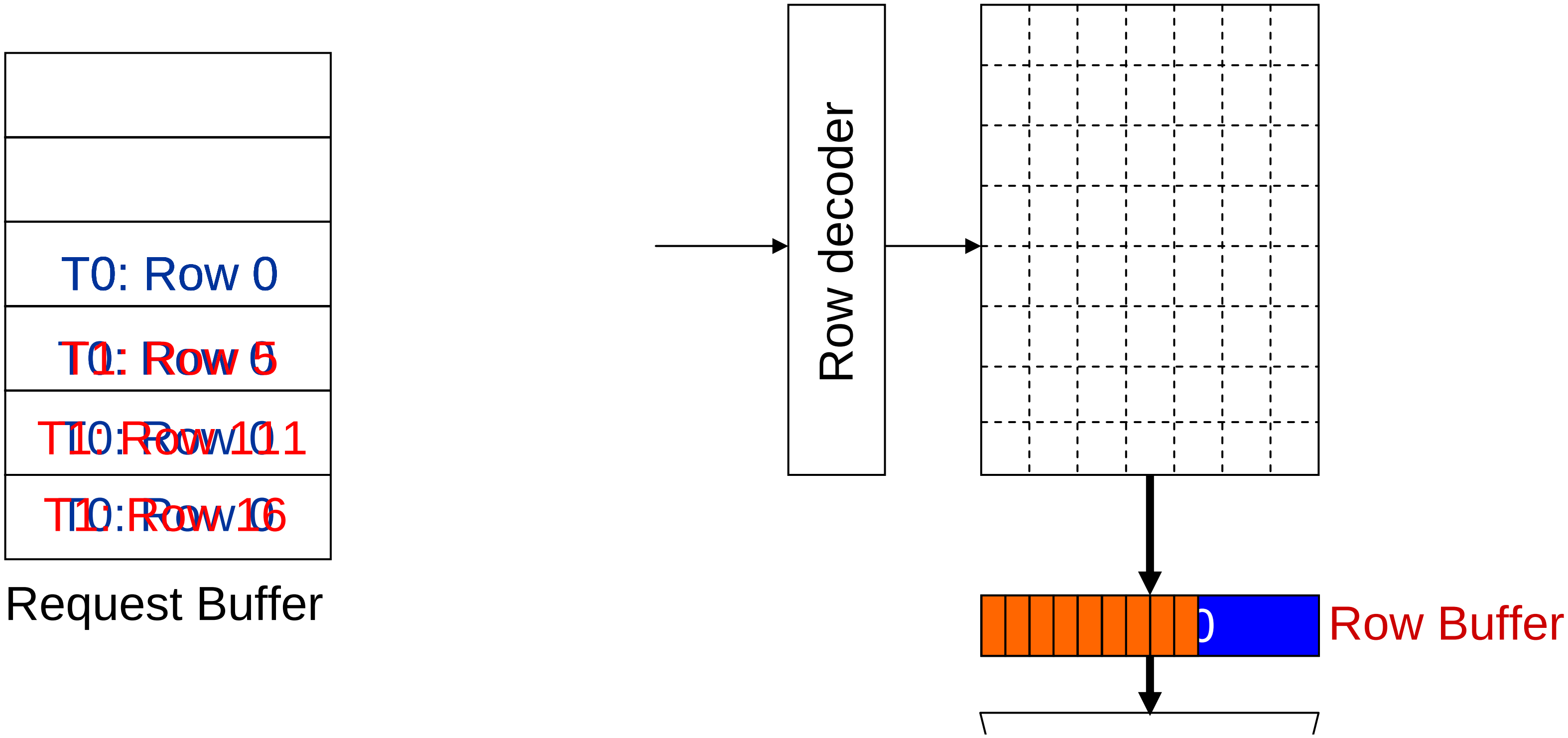


```
// initialize arrays a, b
for (j=0; j<N; j++)
   index[j] = rand();   // random # in [0,N]
•••
for (j=0; j<N; j++)
   a[index[j]] = b[index[j]];
for (j=0; j<N; j++)
   b[index[j]] = scalar * a[index[j]];
•••
```

RDARRAY

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

What does the MPH do?



Row size: 8KB, cache block size: 64B

128 (8KB/64B) requests of T0 serviced before T1

A Multi-Core DRAM Controller

- Should control inter-thread interference in DRAM

- Properties of a good multi-core DRAM controller:

  - provides high system performance
    - preserves each thread’s DRAM bank parallelism
    - efficiently utilizes the scarce memory bandwidth

  - provides fairness to threads sharing the DRAM system
    - Substrate for providing performance guarantees to different cores

  - is controllable and configurable by system software
    - enables different service levels for threads with different priorities

Stall-Time Fair
Memory Access Scheduling

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}

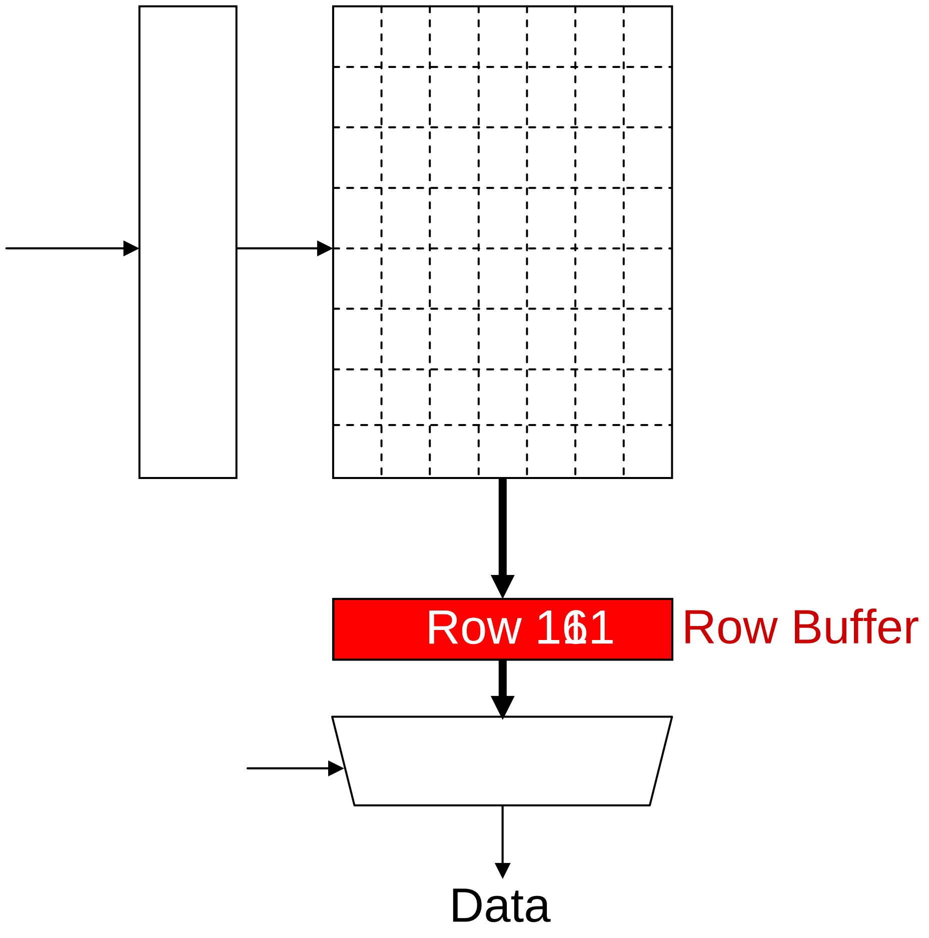
- Each cycle, the DRAM controller
  - Computes Slowdown = ST_{shared}/ST_{alone} for threads with legal requests
  - Computes unfairness = MAX Slowdown / MIN Slowdown

- If unfairness < α
  - Use DRAM throughput oriented scheduling policy
- If unfairness ≥ α
  - 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	1.06
α	1.05



STFM Implementation

- Tracking ST_{shared}
  - Increase ST_{shared} if the thread cannot commit instructions due to an outstanding DRAM access

- Estimating ST_{alone}
  - Difficult to estimate directly because thread not running alone
  - Observation: $ST_{alone} = ST_{shared} - ST_{interference}$
  - Estimate $ST_{interference}$: Extra stall-time due to interference
  - Update $ST_{interference}$ when a thread incurs delay due to other threads
    - When a row buffer hit turns into a row-buffer conflict
      (keep track of the row that would have been in the row buffer)
    - When a request is delayed due to bank or bus conflict

Support for System Software

- System-level thread weights (priorities)
  - OS can choose thread weights to satisfy QoS requirements
  - Larger-weight threads should be slowed down less

  - OS communicates thread weights to the memory controller
  - Controller scales each thread's slowdown by its weight
  - Controller uses weighted slowdown used for scheduling
    - Favors threads with larger weights

- α: Maximum tolerable unfairness set by system software
  - Don't need fairness? Set α large.
  - Need strict fairness? Set α close to 1.
  - Other values of α: trade off fairness and throughput

Parallelism-Aware Batch Scheduling

Mutlu and Moscibroda, "Parallelism-Aware Batch Scheduling: ...," ISCA 2008, IEEE Micro Top Picks 2009.

Another Problem due to Interference

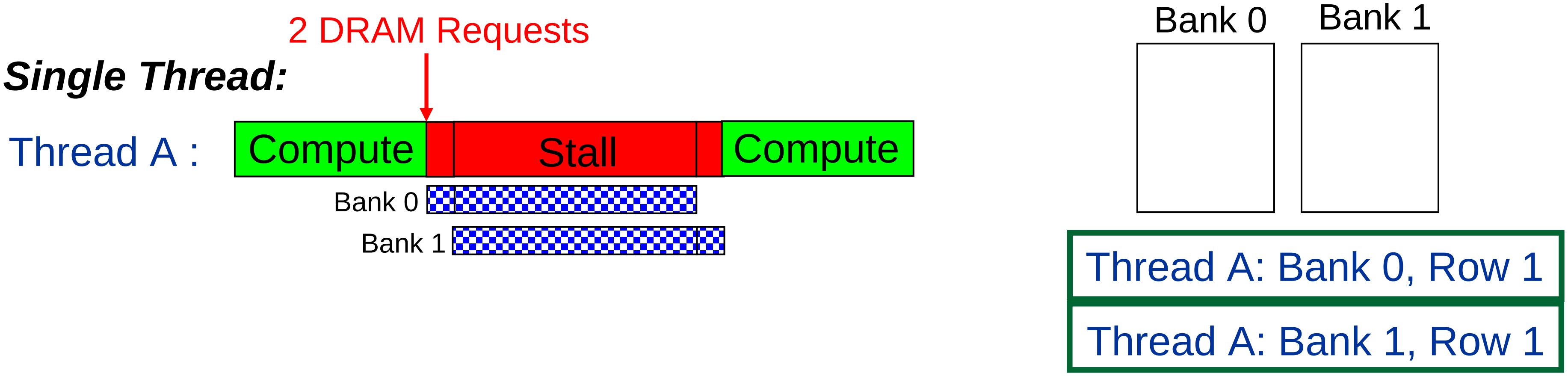
- Processors try to tolerate the latency of DRAM requests by generating multiple outstanding requests
  - Memory-Level Parallelism (MLP)
  - Out-of-order execution, non-blocking caches, runahead execution

- 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

Single Thread:



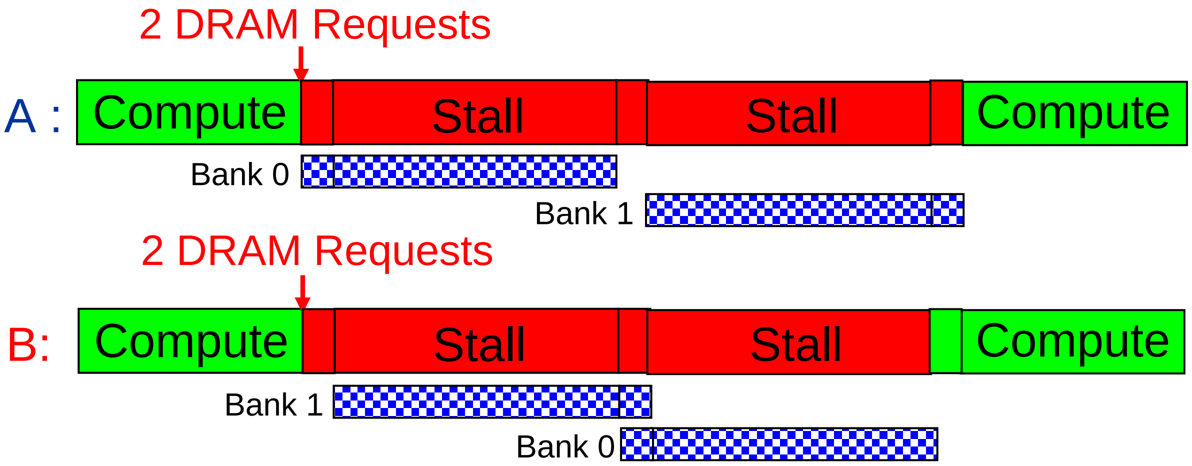
Thread A: Bank 0, Row 1

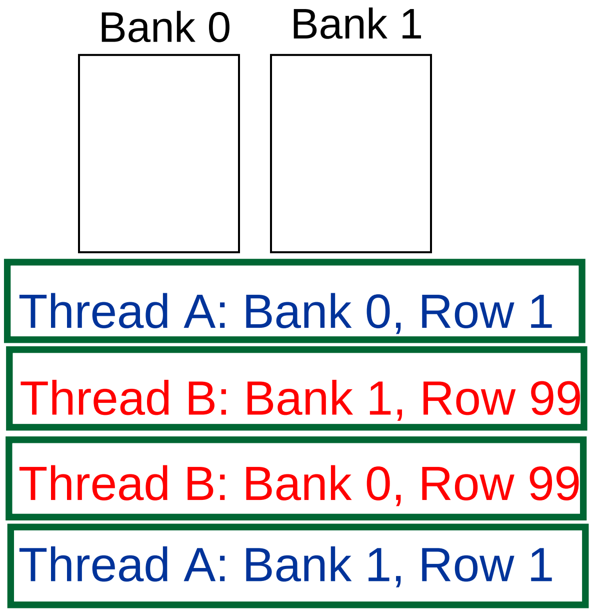
Thread A: Bank 1, Row 1

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

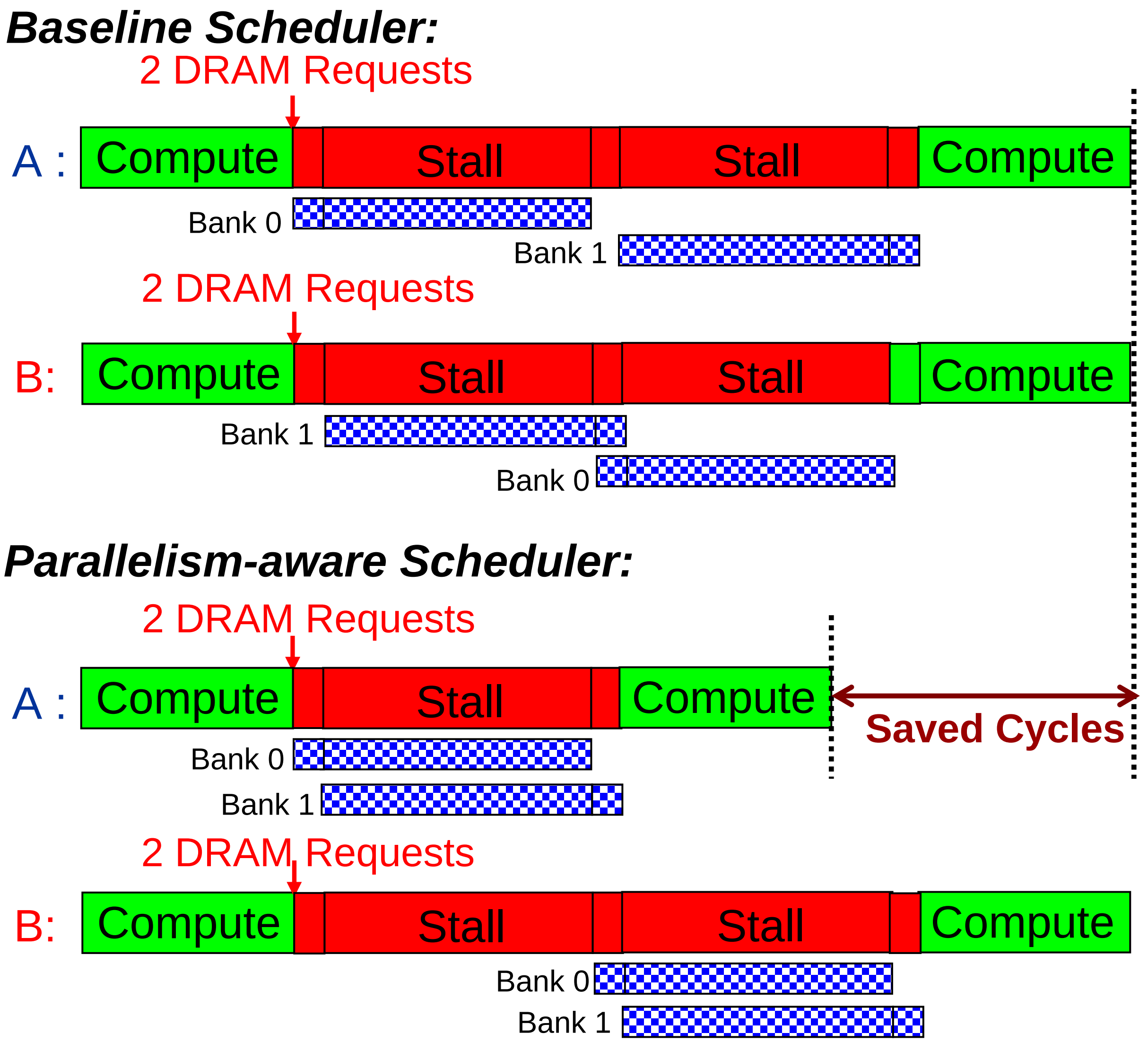
Thread B: Bank 0, Row 99

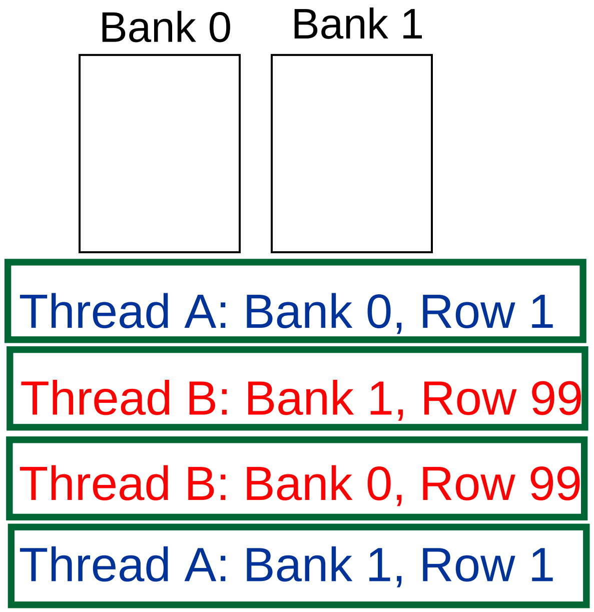
Thread A: Bank 1, Row 1

Bank access latencies of each thread serialized

Each thread stalls for ~TWO bank access latencies

Parallelism-Aware Scheduler





Thread A: Bank 0, Row 1

Thread B: Bank 1, Row 99

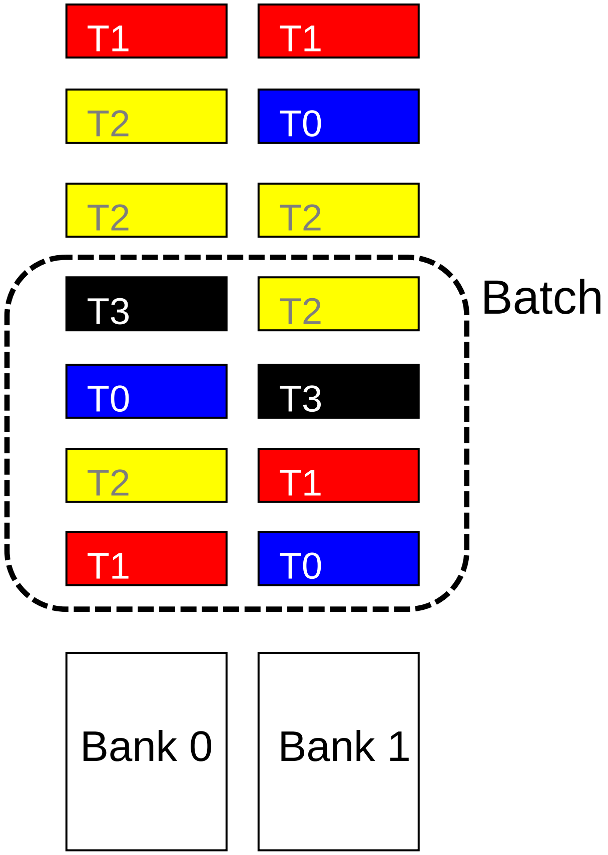
Thread B: Bank 0, Row 99

Thread A: Bank 1, Row 1

Average stall-time: ~1.5 bank access latencies

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

HOW?

- Scheduler computes a 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

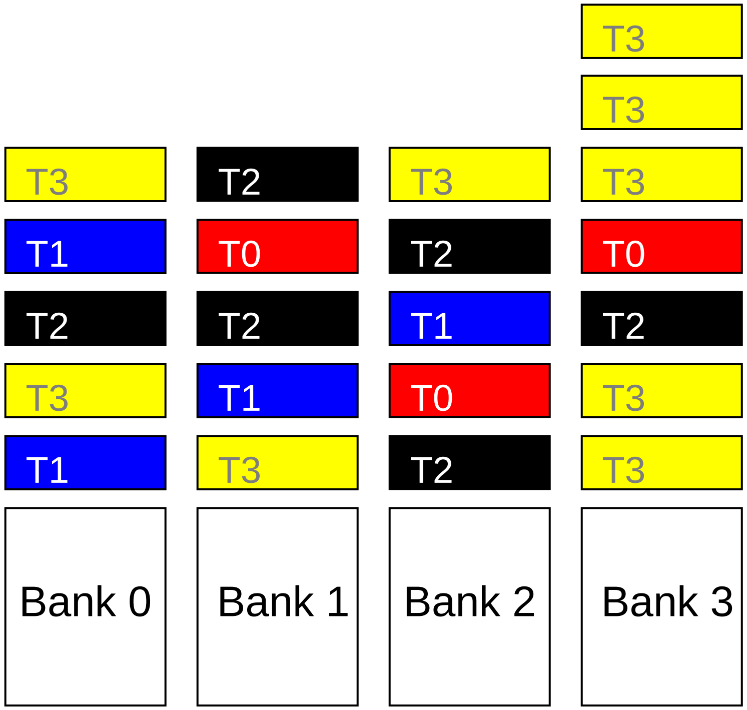
- Maximize system throughput
  - Minimize average stall-time of threads within the batch
- Minimize unfairness (Equalize the slowdown of threads)
  - Service threads with inherently low stall-time early in the batch
  - Insight: delaying memory non-intensive threads results in high slowdown

- 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



T3
T1
T2
T3
T1
Bank 0

T2
T0
T2
T1
T3
Bank 1

T3
T2
T1
T0
T2
Bank 2

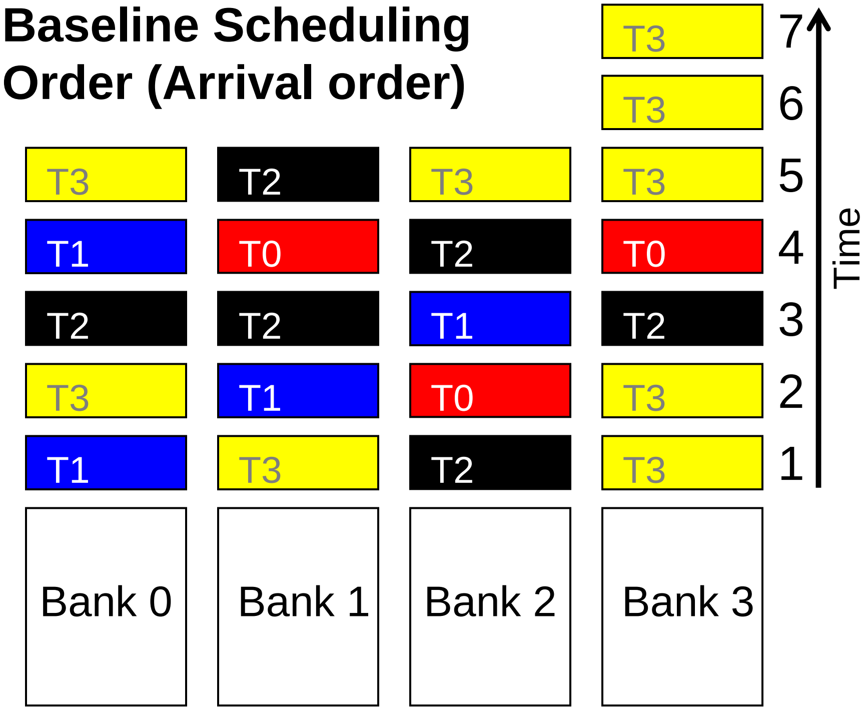
T3
T3
T3
T0
T2
T3
T3
Bank 3

	max-bank-load	total-load

Ranking:
T0 > T1 > T2 > T3

Example Within-Batch Scheduling Order

Baseline Scheduling Order (Arrival order)

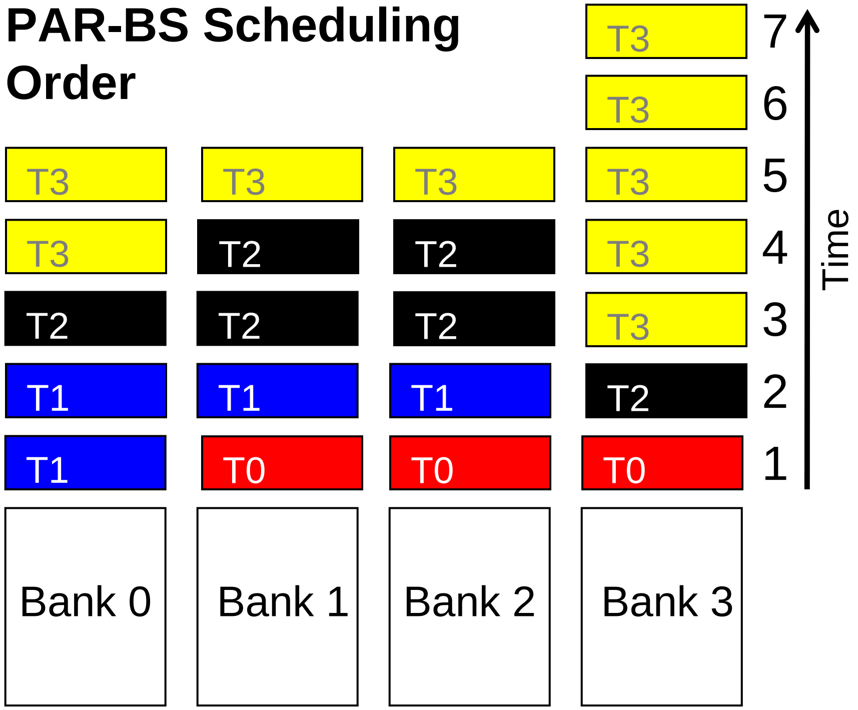


			T3	7
			T3	6
T3	T2	T3	T3	5
T1	T0	T2	T0	4
T2	T2	T1	T2	3
T3	T1	T0	T3	2
T1	T3	T2	T3	1
Bank 0	Bank 1	Bank 2	Bank 3	

	T0	T1	T2	T3
Stall times				

AVG: 5 bank access latencies

PAR-BS Scheduling Order



			T3	7
			T3	6
T3	T3	T3	T3	5
T3	T2	T2	T3	4
T2	T2	T2	T3	3
T1	T1	T1	T2	2
T1	T0	T0	T0	1
Bank 0	Bank 1	Bank 2	Bank 3	

Ranking: T0 > T1 > T2 > T3

	T0	T1	T2	T3
Stall times				

AVG: 3.5 bank access latencies

Putting It Together: PAR-BS Scheduling Policy

- PAR-BS Scheduling Policy
  - (1) Marked requests first
  - (2) Row-hit requests first
  - (3) Higher-rank thread first (shortest stall-time first)
  - (4) Oldest first

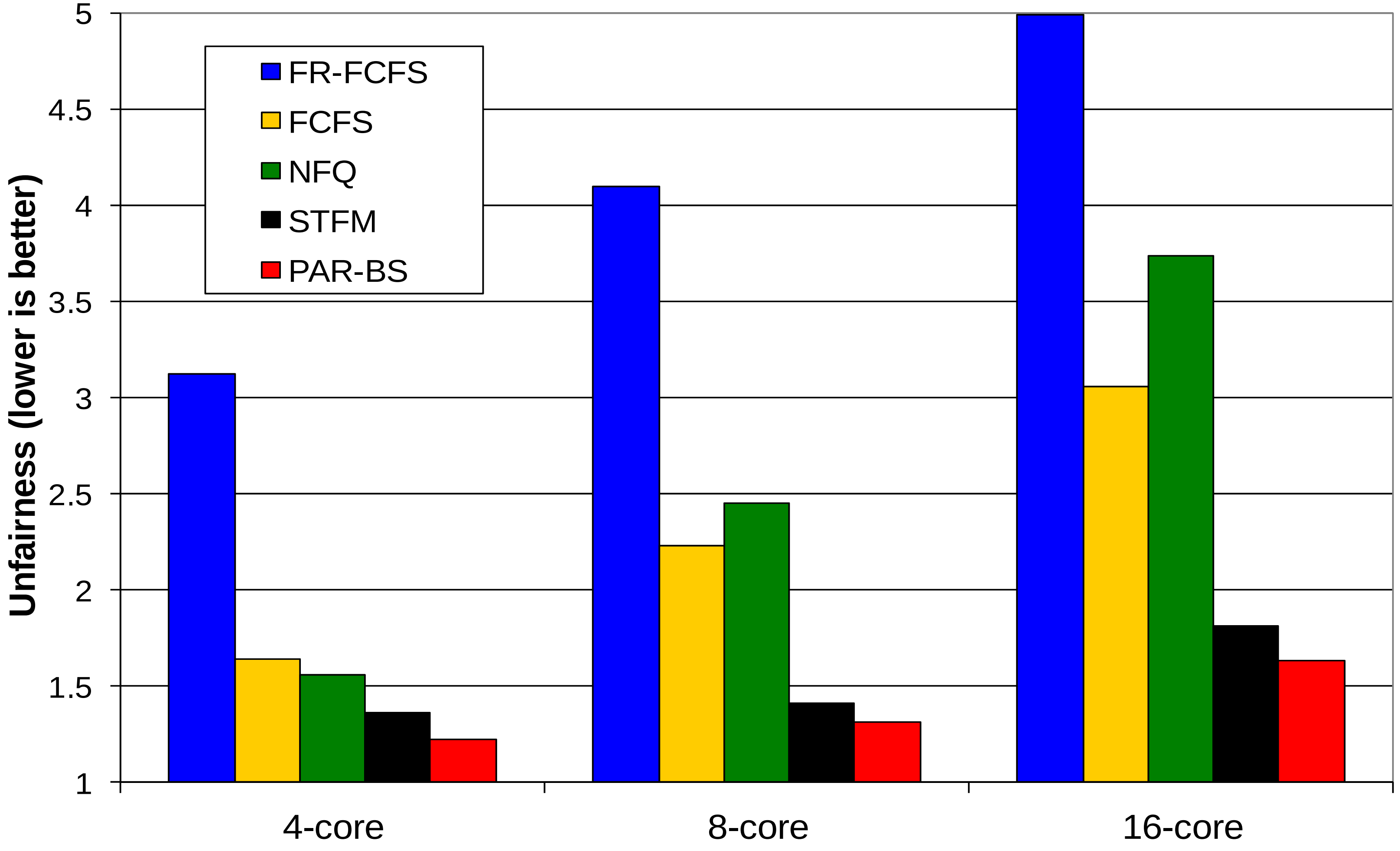
Batching

Parallelism-aware within-batch scheduling

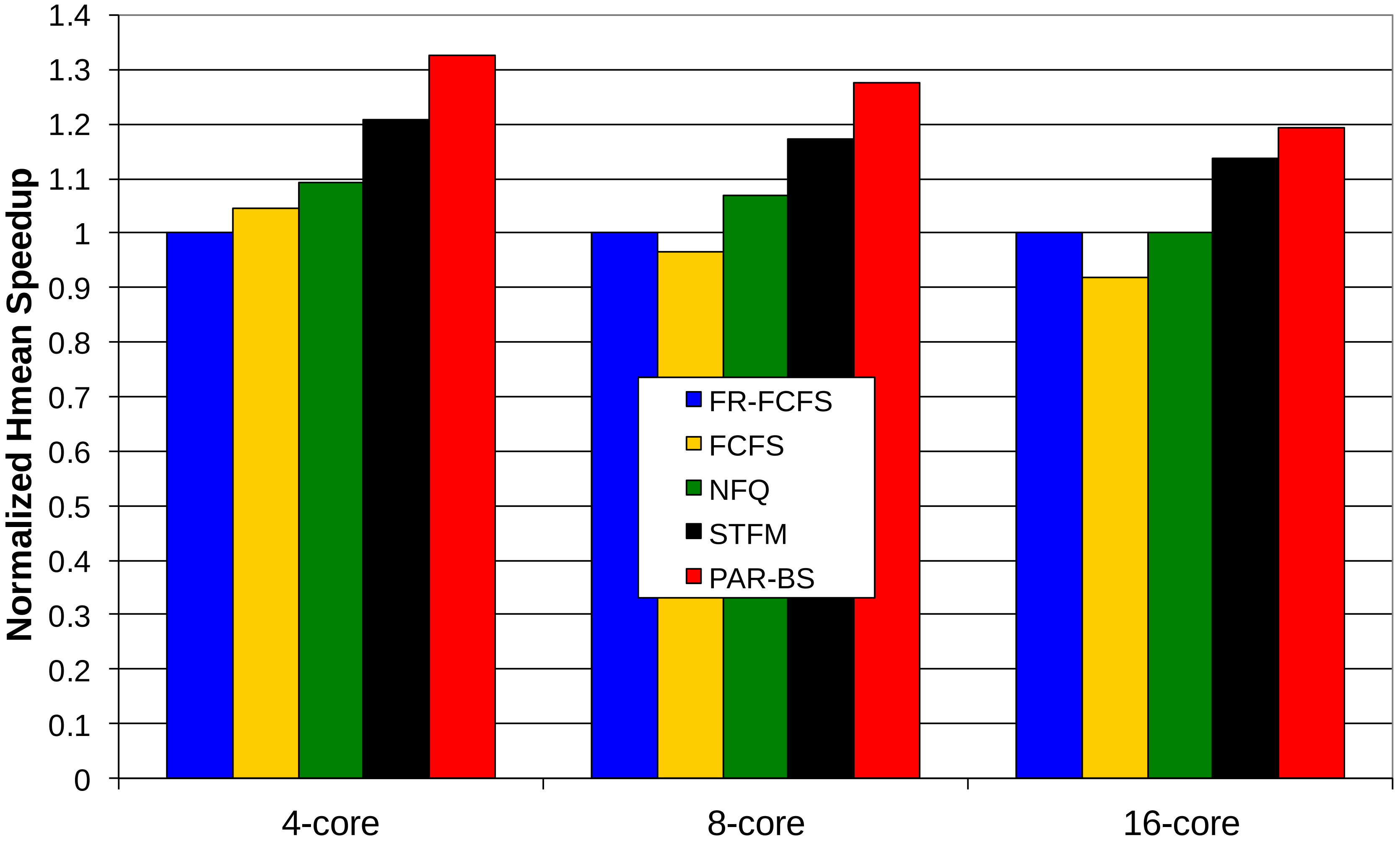
- Three properties:
  - Exploits row-buffer locality **and** intra-thread bank parallelism
  - Work-conserving
    - Services unmarked requests to banks without marked requests
  - Marking-Cap is important
    - Too small cap: destroys row-buffer locality
    - Too large cap: penalizes memory non-intensive threads
- Trade-offs analyzed in [ISCA 2008]

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

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



System Performance


Normalized Hmean Speedup
4-core
8-core
16-core
FR-FCFS
FCFS
NFQ
STFM
PAR-BS