DRAM COMMAND SCHEDULING

Mahdi Nazm Bojnordi

Assistant Professor

School of Computing

University of Utah

Overview

- Upcoming deadline
  - Tonight: homework assignment will be posted
- This lecture
  - Basics of DRAM controllers
  - Memory access scheduling
  - Bank level parallel scheduler
  - Thread cluster scheduling
  - Self optimizing scheduler

DRAM Controller

- Translate memory read/write requests to DRAM commands
  - DRAM controller enforces all of the timing constraints

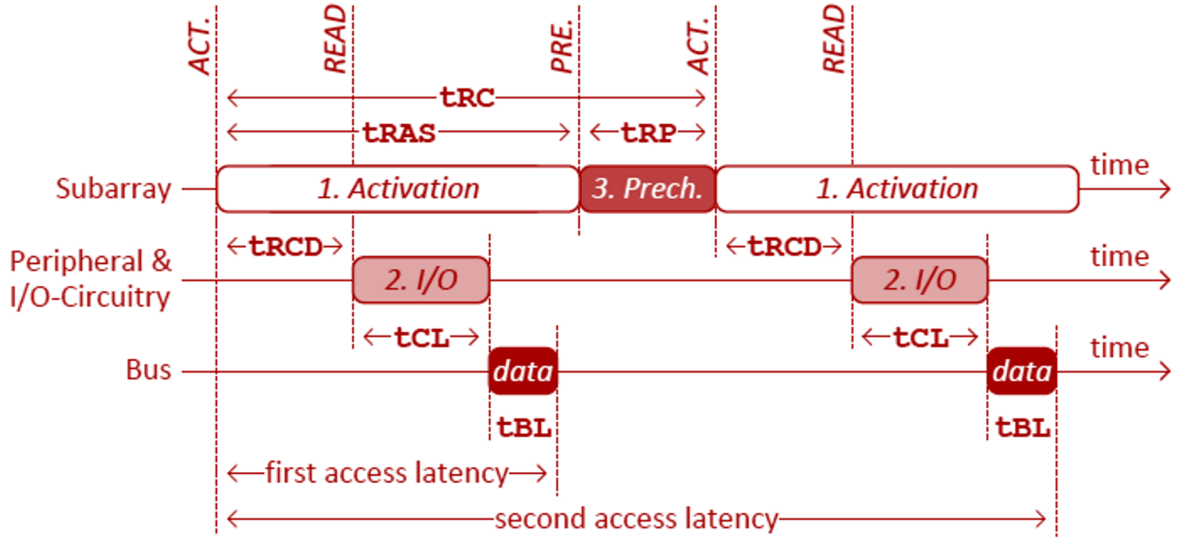


Figure 5. Three Phases of DRAM Access

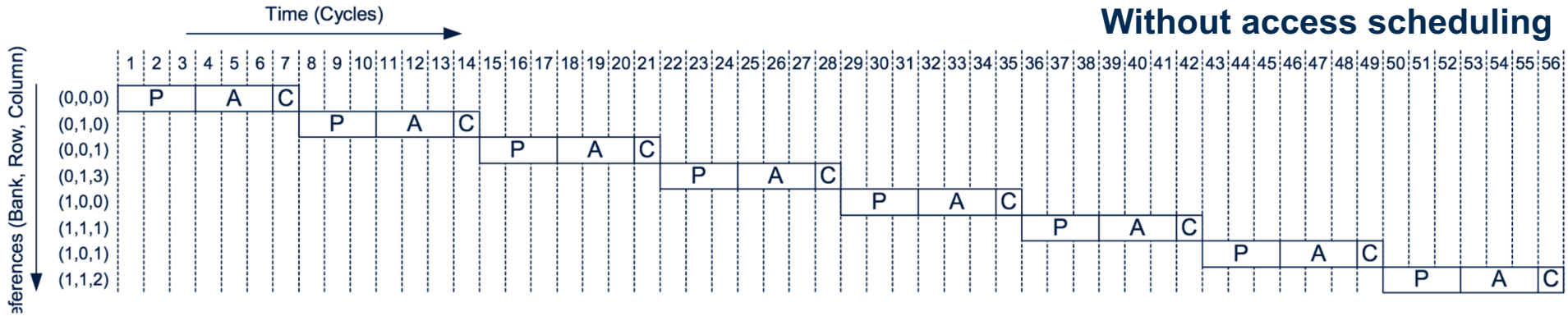
Table 2. Timing Constraints (DDR3-1066) [43]

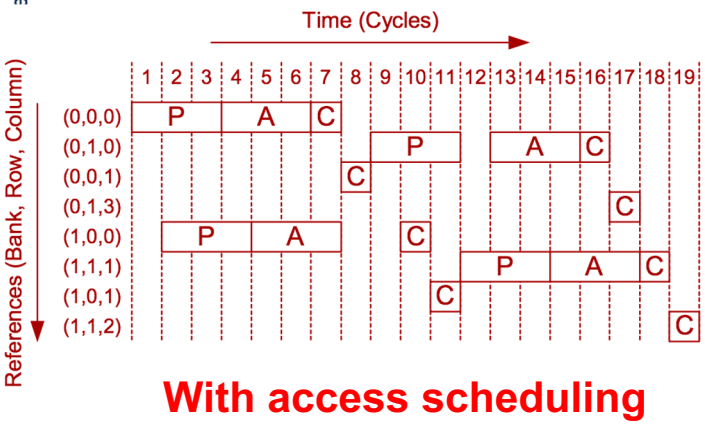
Phase	Commands	Name	Value
1	ACT → READ ACT → WRITE	tRCD	15ns
	ACT → PRE	tRAS	37.5ns
2	READ → *data* WRITE → *data*	tCL tCWL	15ns 11.25ns
	data burst	tBL	7.5ns
3	PRE → ACT	tRP	15ns
1 & 3	ACT → ACT	tRC (tRAS+tRP)	52.5ns

[slide ref: Mutlu]

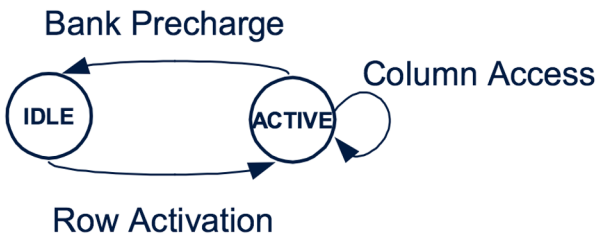
Memory Access Scheduling

- Command scheduling significantly reduces DRAM access time





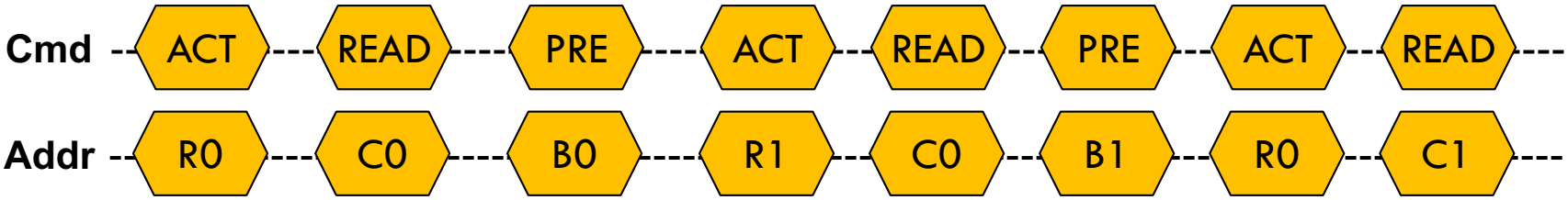
With access scheduling



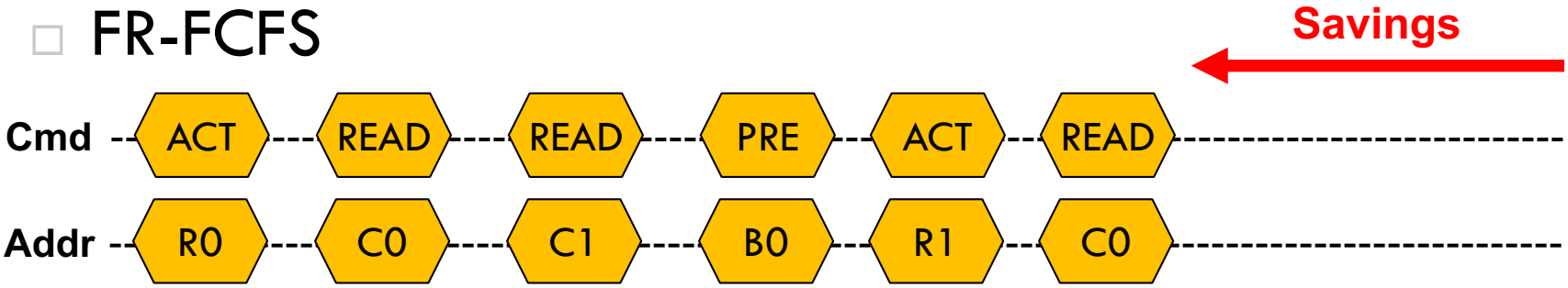
[Rixner'00]

FCFS vs. FR-FCFS

- Single bank memory
  - READ(B0,R0,C0) READ(B0,R1,C0) READ(B0,R0,C1)
- FCFS

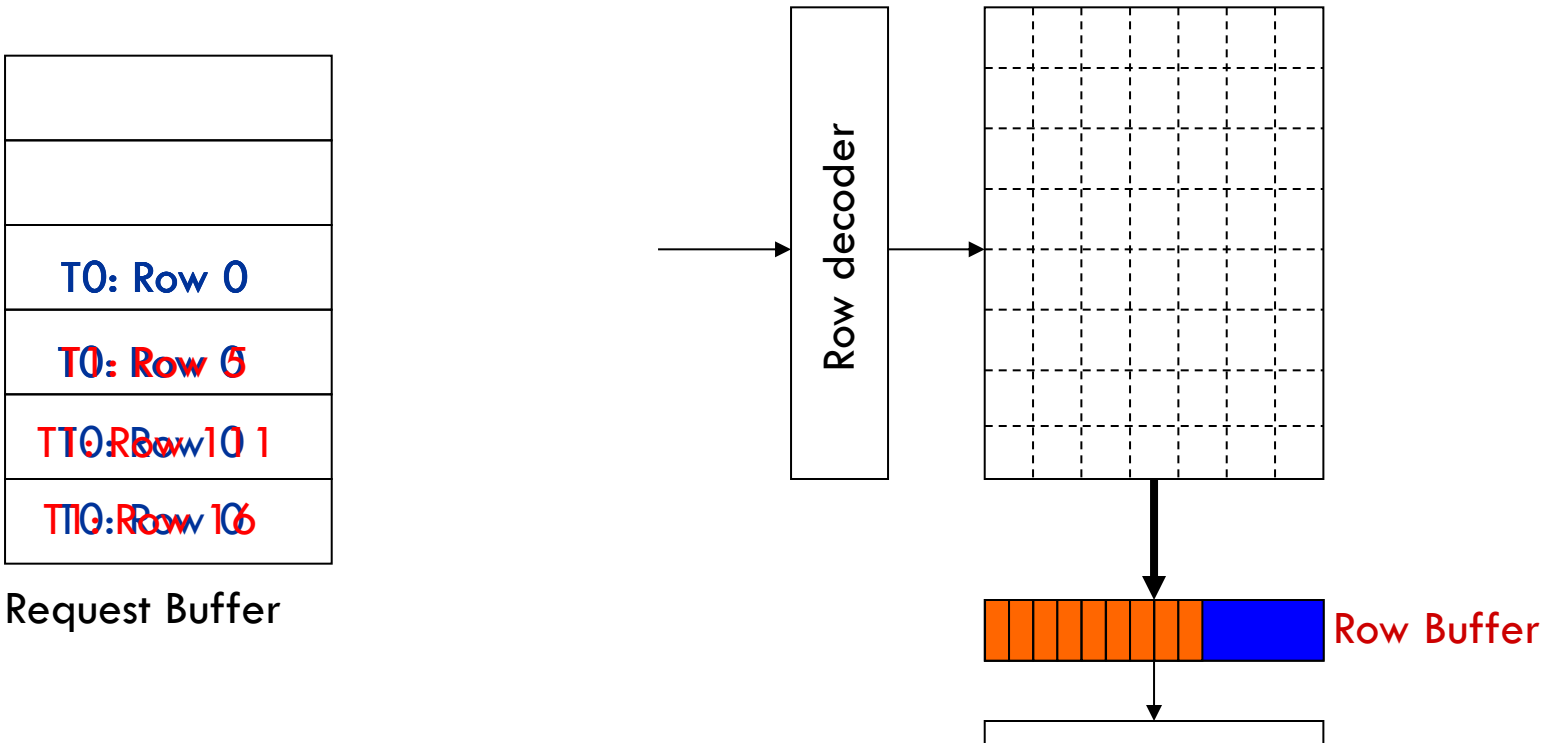


- FR-FCFS



FR-FCFS Scheduling

FR-FCFS policy: 1) row-hit first, 2) oldest first

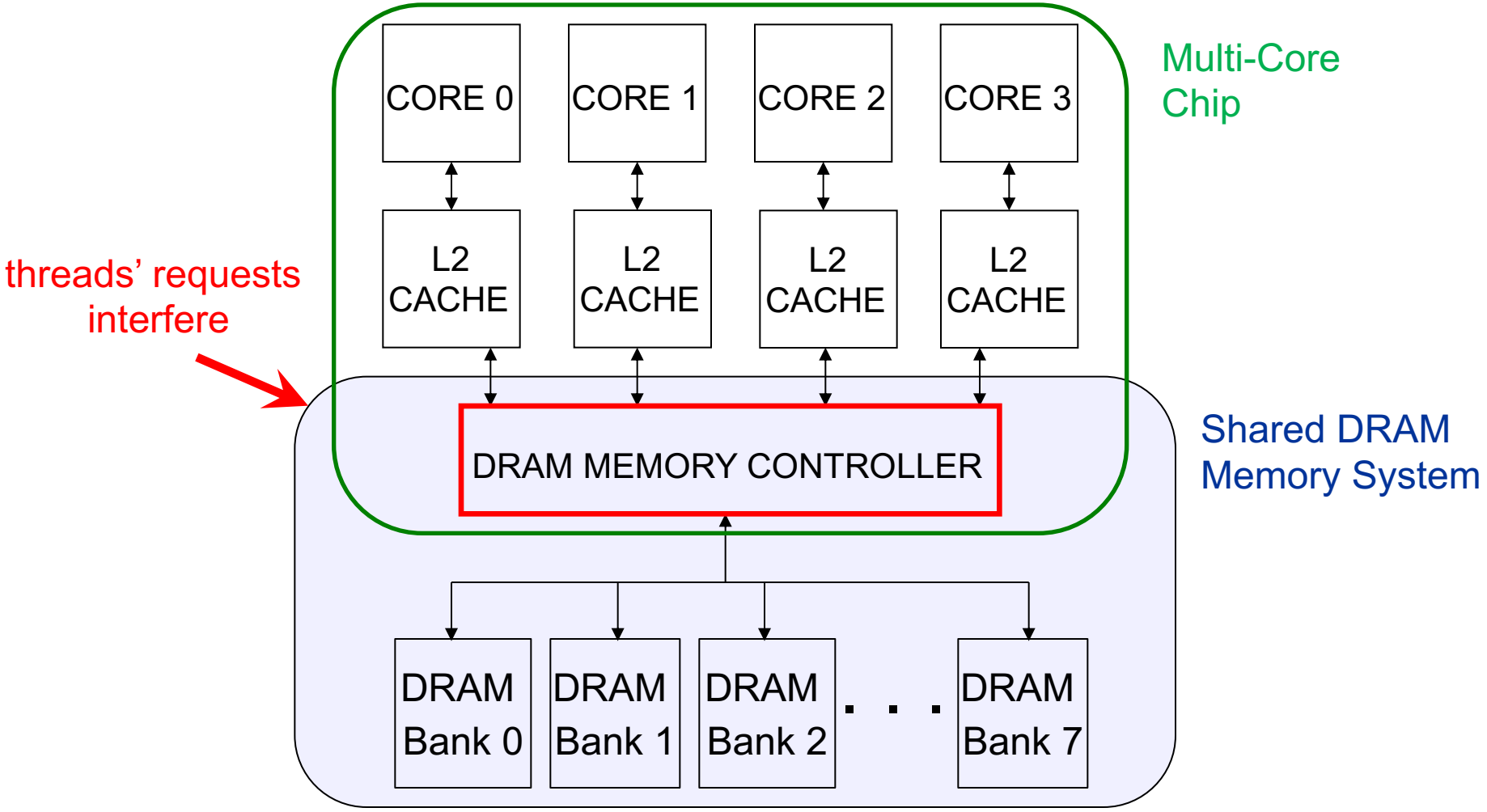


Row size: 8KB, cache block size: 64B

128 requests of T0 serviced before T1

[Mutlu'07]

Multi-Core Systems



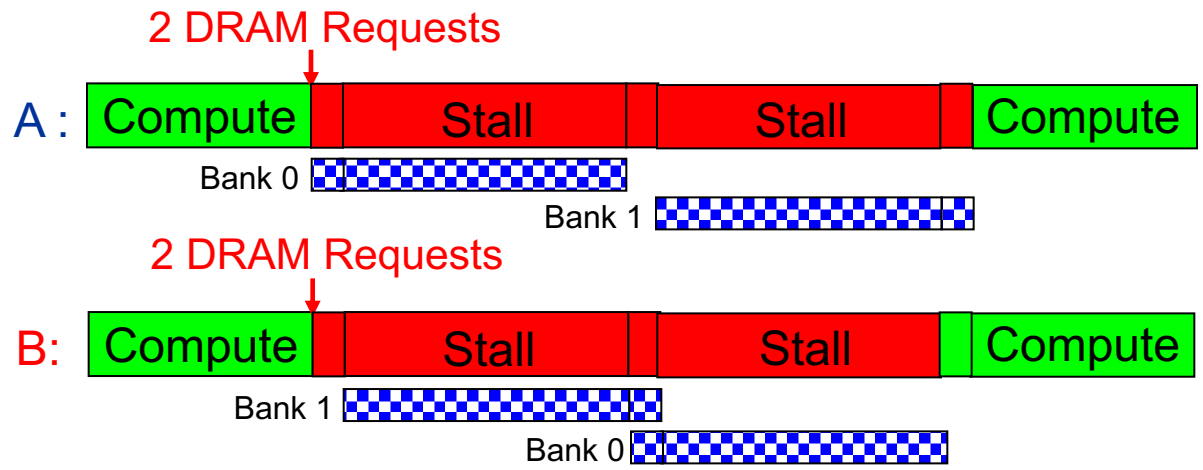
Interference in the DRAM System

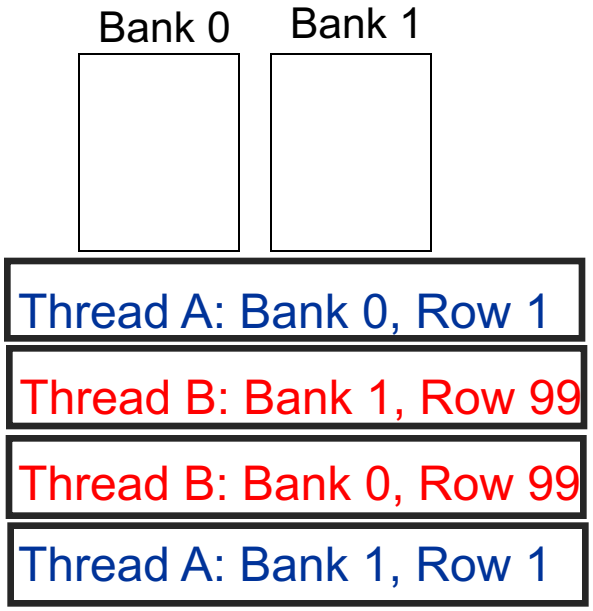
- Threads delay each other by causing resource contention
  - Bank, bus, row-buffer conflicts [MICRO 2007]
- Threads can also destroy each other's DRAM bank parallelism
  - Otherwise parallel requests can become serialized

- Traditional DRAM schedulers were unaware of inter-thread 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

Bank Parallelism Interference

Baseline Scheduler:

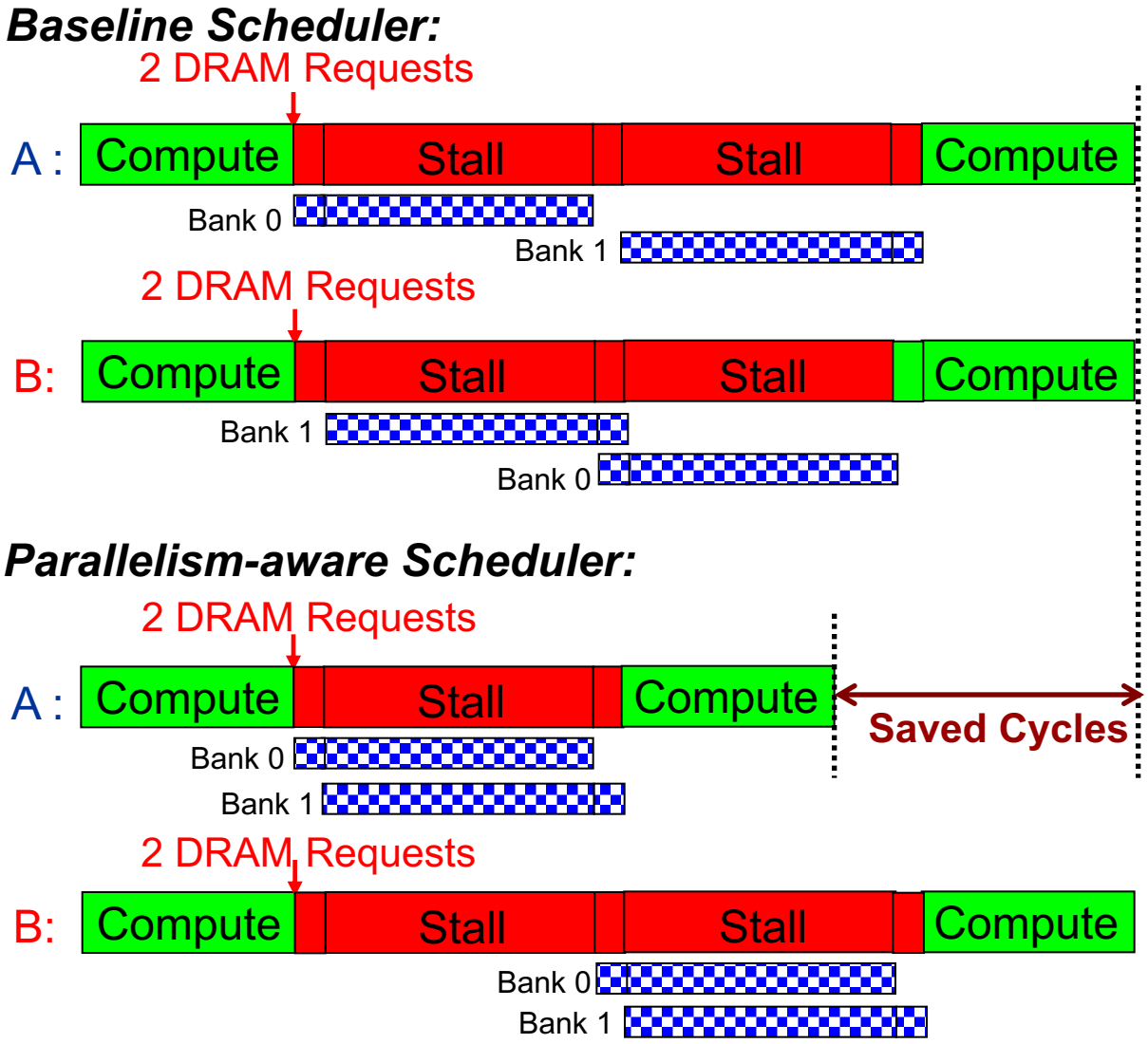


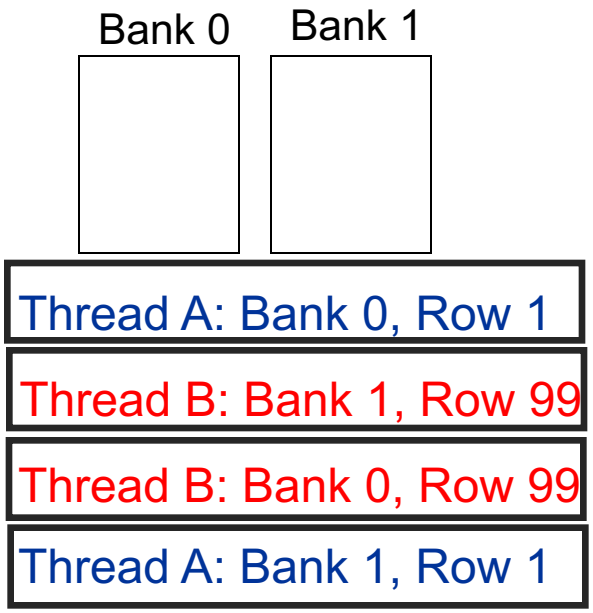


Bank access latencies of each thread serialized
Each thread stalls for ~TWO bank access latencies

[Mutlu'08]

Parallelism-Aware Scheduler



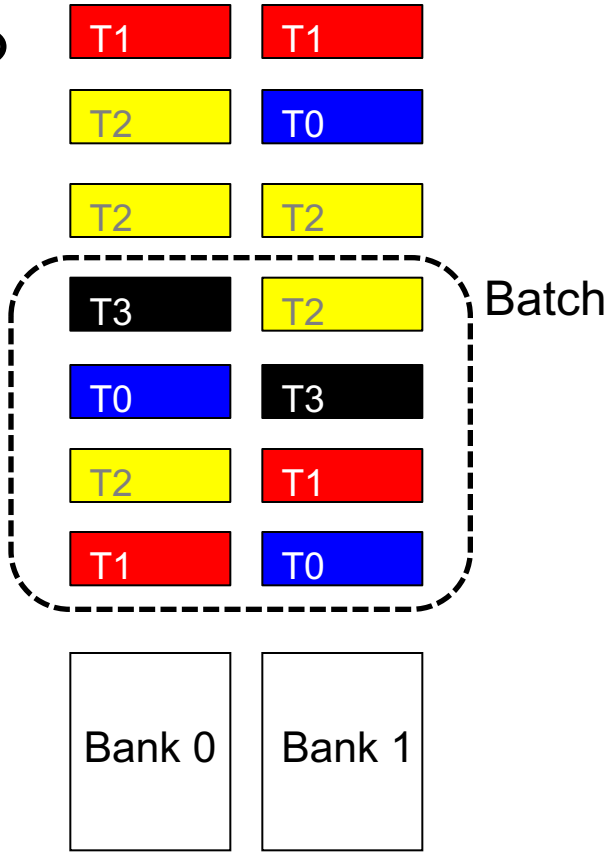


Average stall-time: ~1.5 bank access latencies

[Mutlu'08]

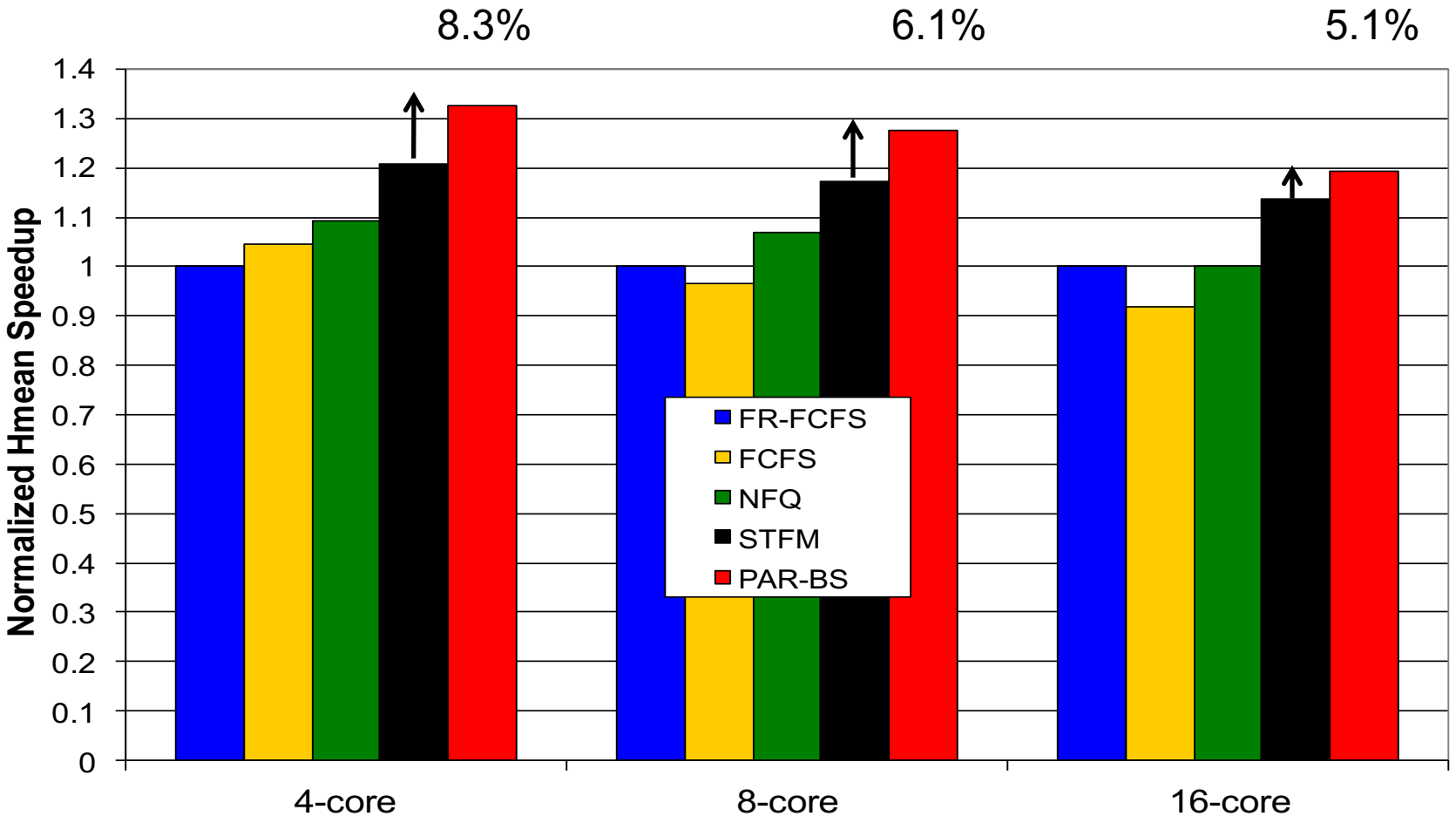
Parallelism-Aware Batch Scheduling

- Schedule requests from a thread (to different banks) back to back
  - Preserves each thread's bank parallelism
  - This can cause starvation…
- Group a fixed number of oldest requests from each thread into a "batch"
  - Service the batch before all other requests



[Mutlu'08]

System Performance



[Mutlu'08]

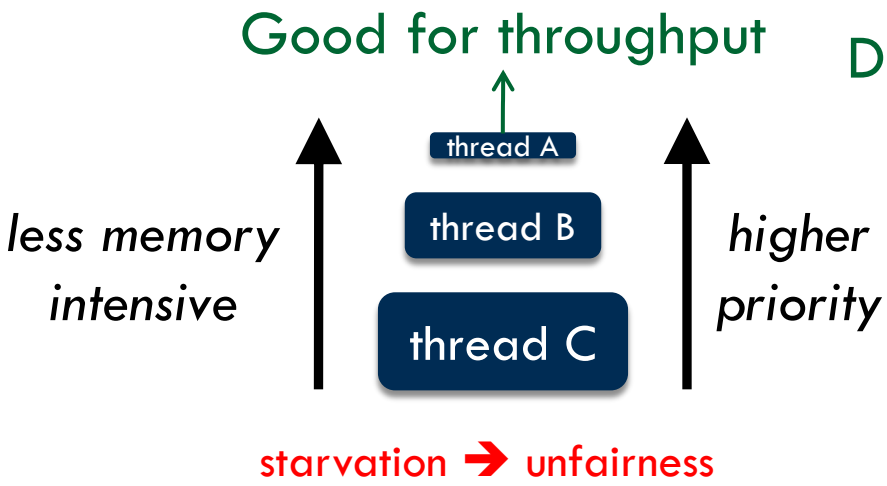
Problem: Conflicting Objectives

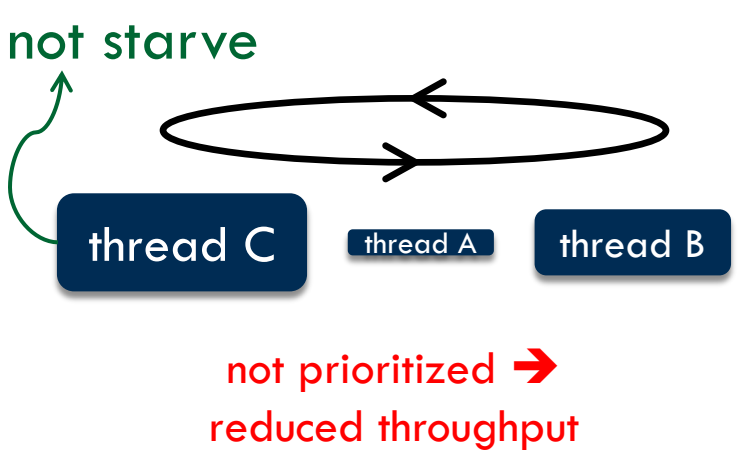
Throughput biased approach

Prioritize less memory-intensive threads

Fairness biased approach

Take turns accessing memory

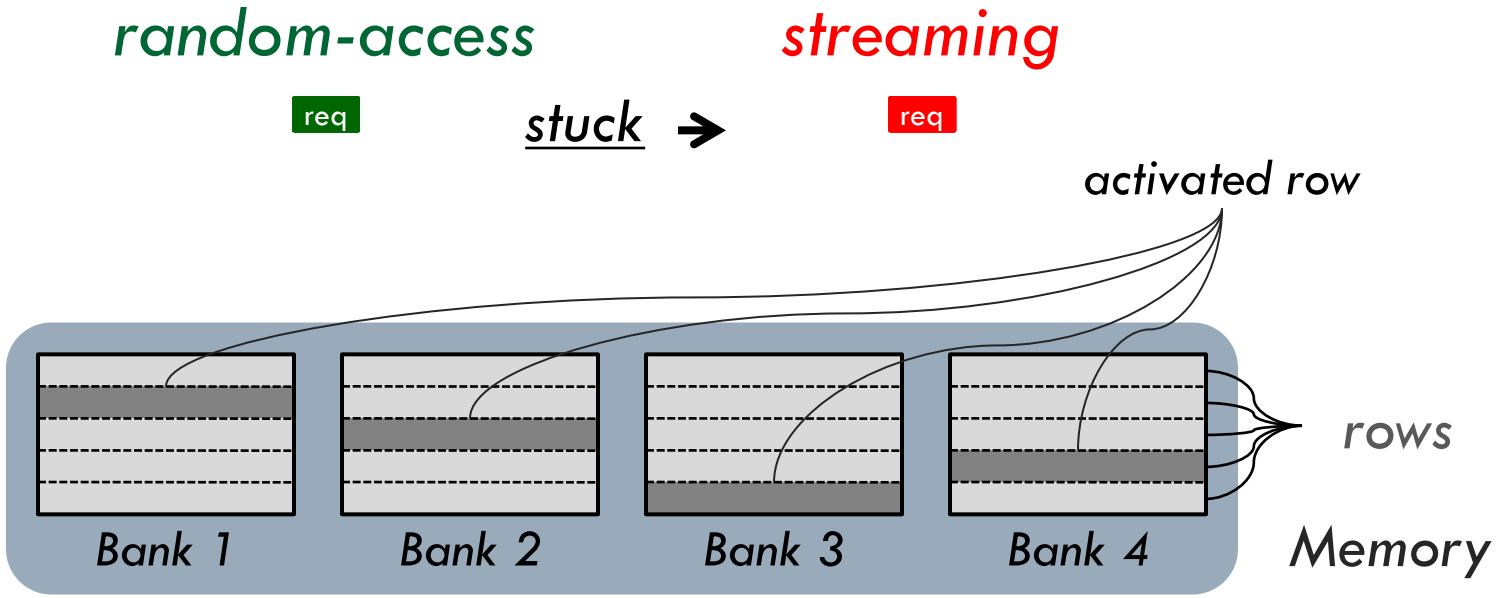




Single policy for all threads is insufficient

[Kim'09]

Threads Are Different



- All requests parallel
- High bank-level parallelism

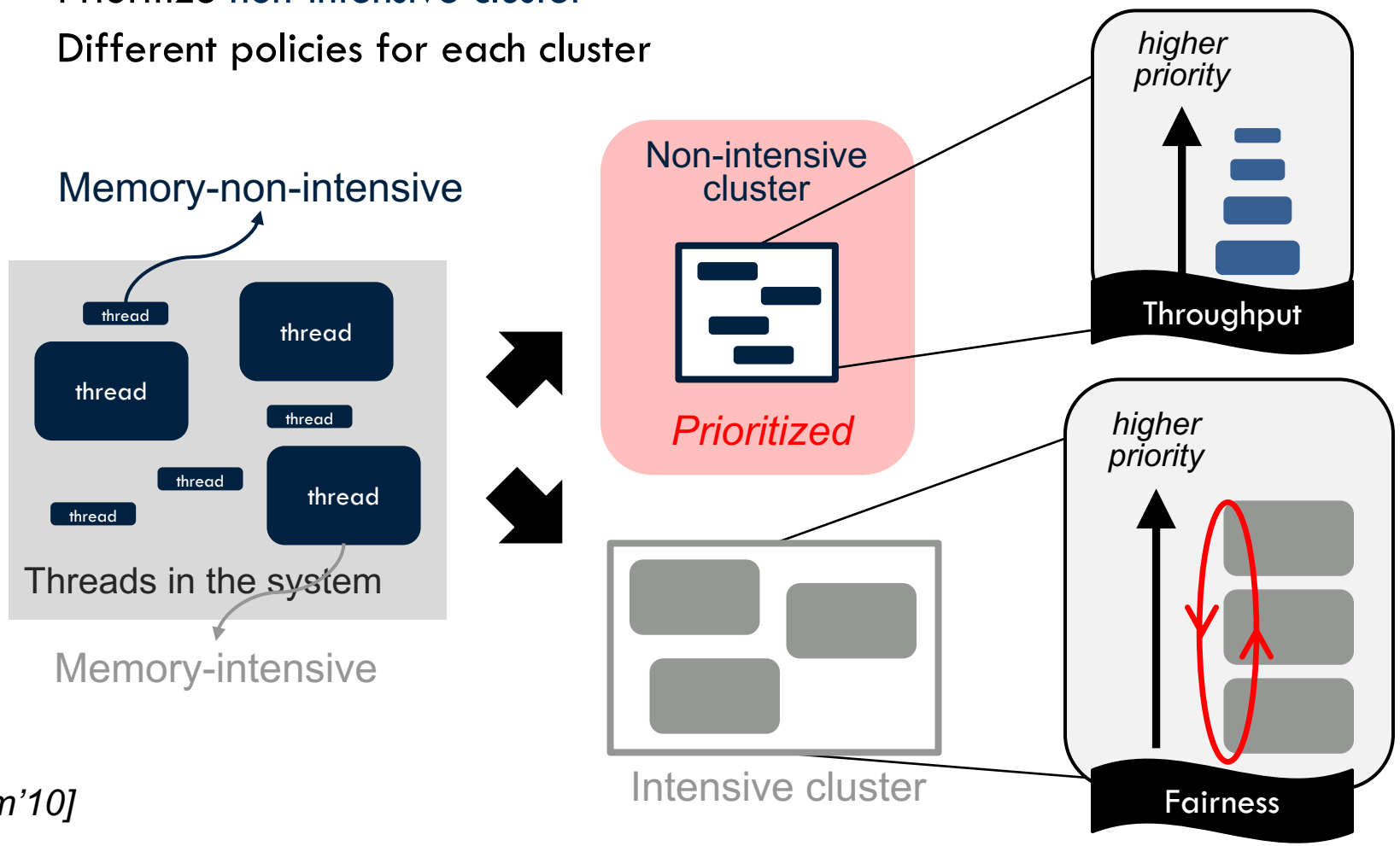
- All requests → Same row
- High row-buffer locality

Vulnerable to interference

[Kim'10]

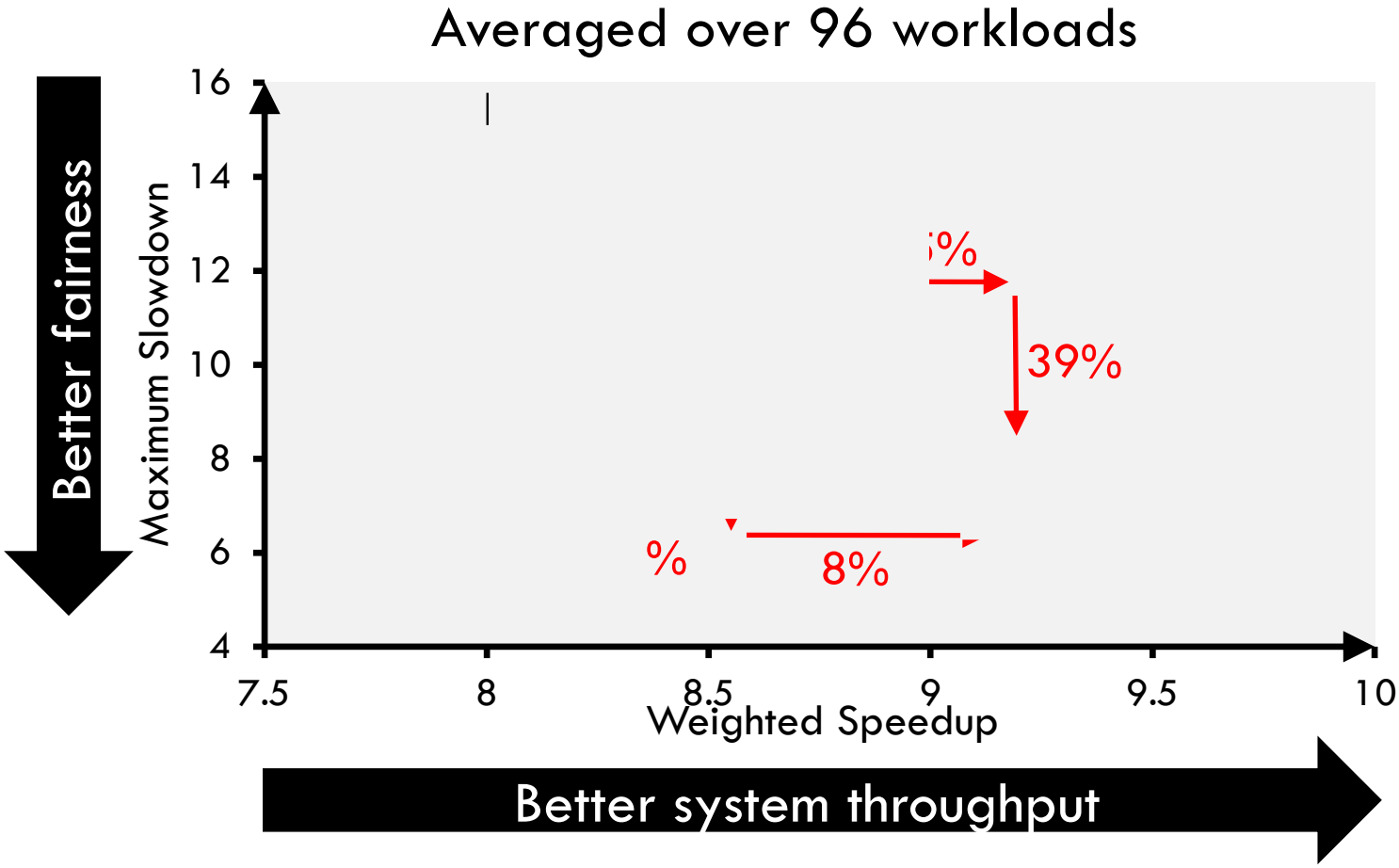
Thread Cluster Memory Scheduling

1. Group threads into two *clusters*
2. Prioritize non-intensive cluster
3. Different policies for each cluster



[Kim'10]

Results: Fairness vs. Throughput



TCM provides best fairness and system throughput

[Kim'09]

Self-Optimizing DRAM Controller

- Problem: DRAM controllers difficult to design
  - It is difficult for human designers to design a policy that can adapt itself very well to different workloads and different system conditions
- Idea: Design a memory controller that adapts its scheduling policy decisions to workload behavior and system conditions using machine learning
- Observation: Reinforcement learning maps nicely to memory control
- Design: Memory controller is a reinforcement learning agent that dynamically and continuously learns and employs the best scheduling policy

[Ipek'08]

Self-Optimizing DRAM Controller

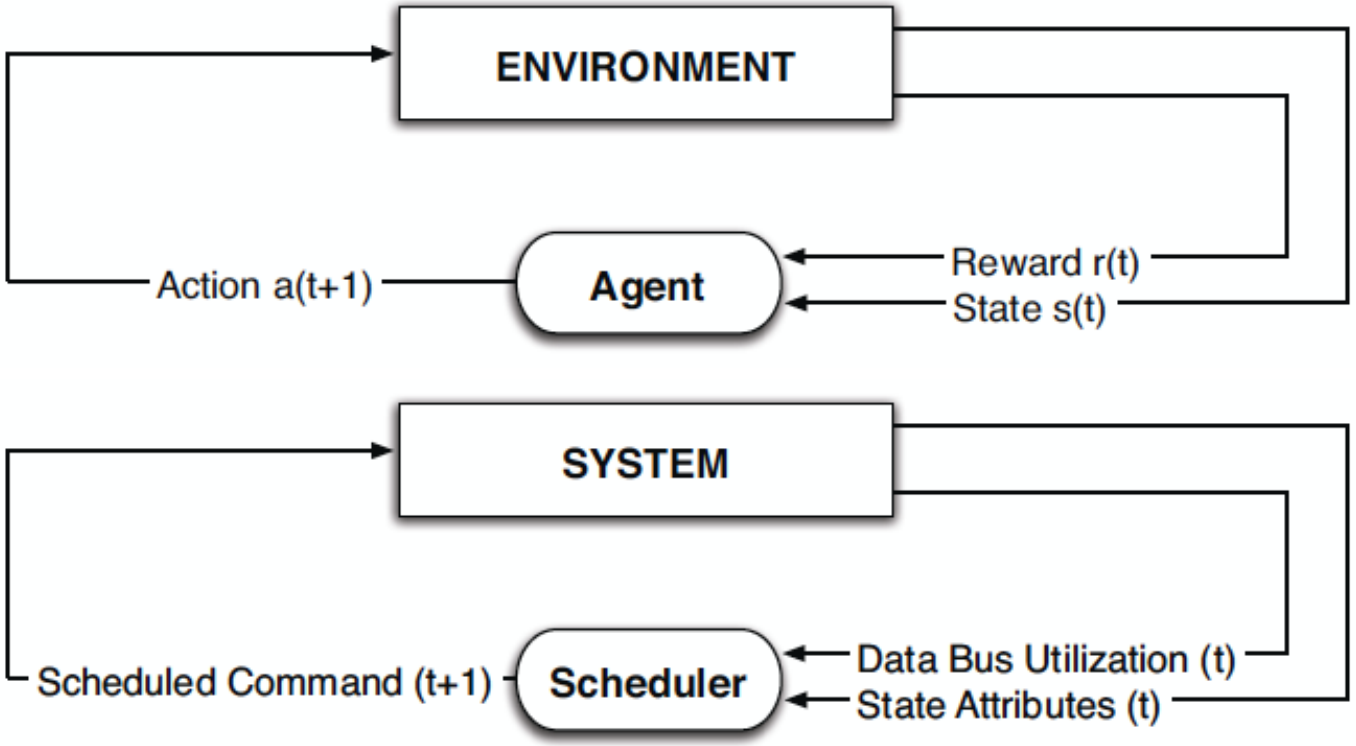


Figure 2: (a) Intelligent agent based on reinforcement learning principles; (b) DRAM scheduler as an RL-agent

[Ipek'08]

Self-Optimizing DRAM Controller

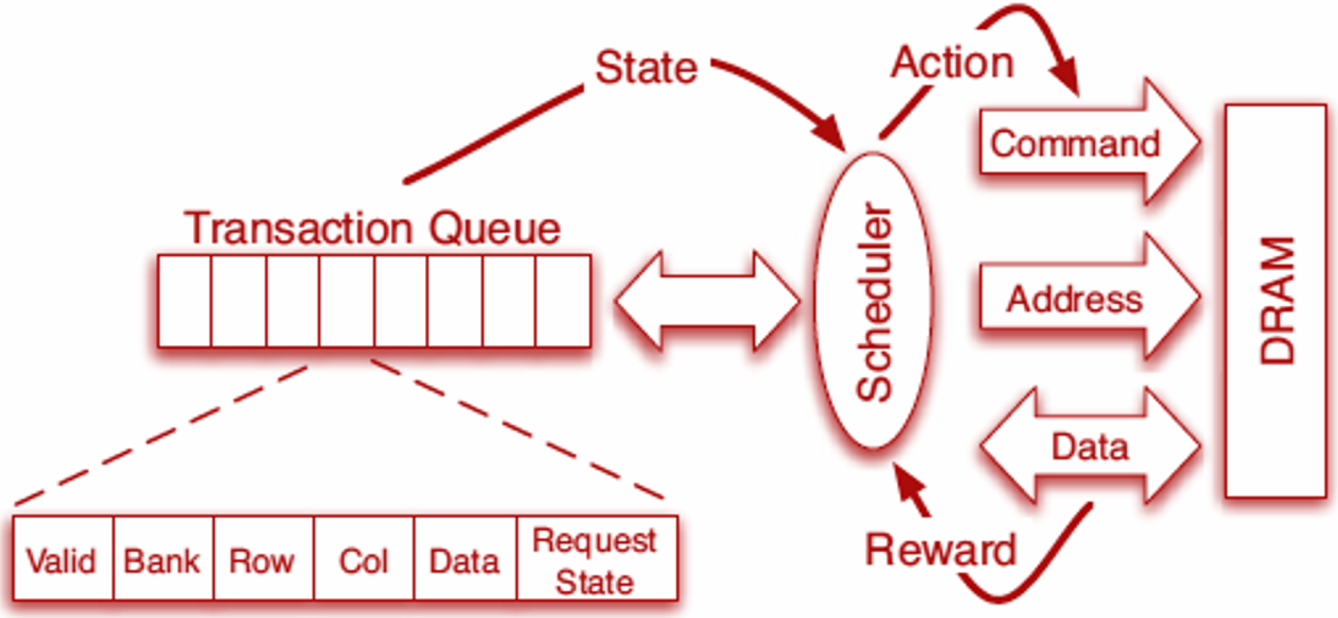


Figure 4: High-level overview of an RL-based scheduler.

[Ipek'08]

Self-Optimizing DRAM Controller

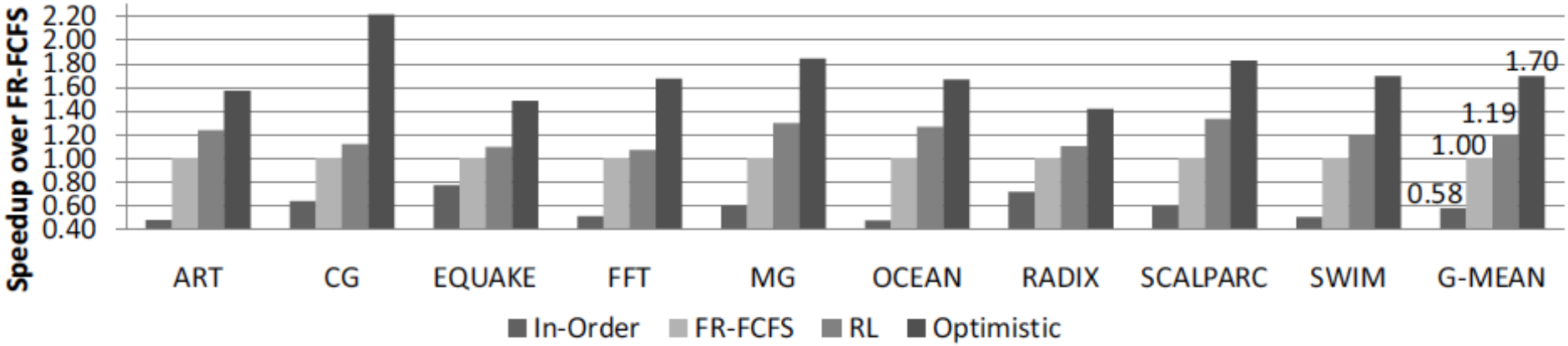


Figure 7: Performance comparison of in-order, FR-FCFS, RL-based, and optimistic memory controllers

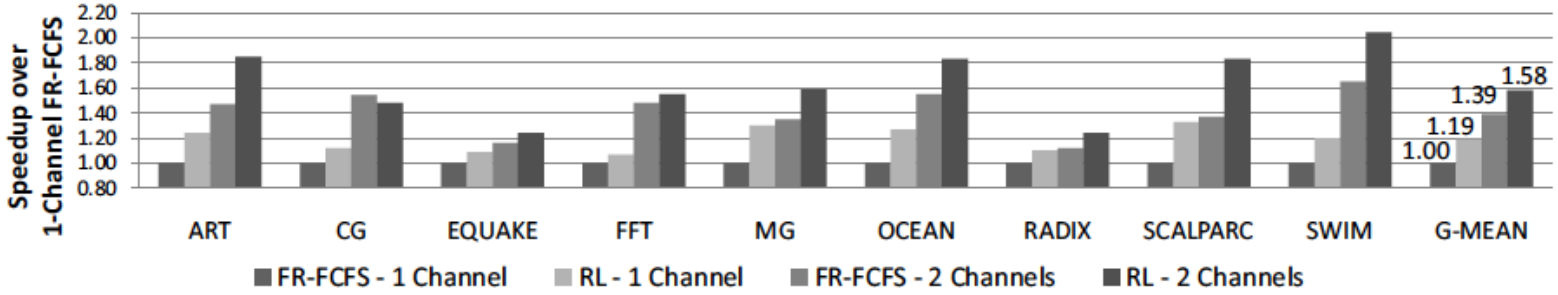


Figure 15: Performance comparison of FR-FCFS and RL-based memory controllers on systems with 6.4GB/s and 12.8GB/s peak DRAM bandwidth

[Ipek'08]