

Computer Architecture

Lecture 13a: Memory Controllers

Prof. Onur Mutlu

ETH Zürich

Fall 2019

31 October 2019

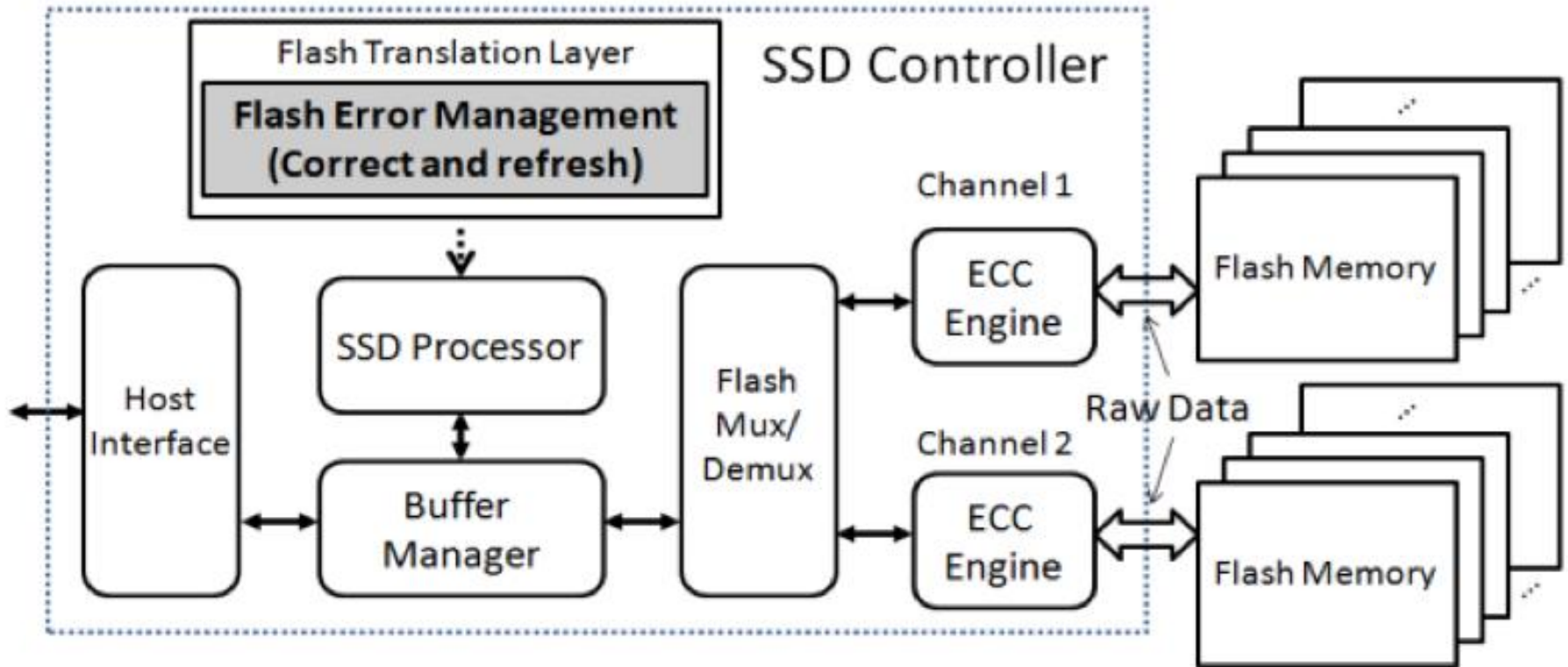
Memory Controllers

DRAM versus Other Types of Memories

- Long latency memories have similar characteristics that need to be controlled.
- The following discussion will use DRAM as an example, but many scheduling and control issues are similar in the design of controllers for other types of memories
 - Flash memory
 - Other emerging memory technologies
 - Phase Change Memory
 - Spin-Transfer Torque Magnetic Memory
 - These other technologies can place other demands on the controller

Flash Memory (SSD) Controllers

- Similar to DRAM memory controllers, except:
 - They are flash memory specific
 - They do much more: error correction, garbage collection, page remapping, ...



Another View of the SSD Controller

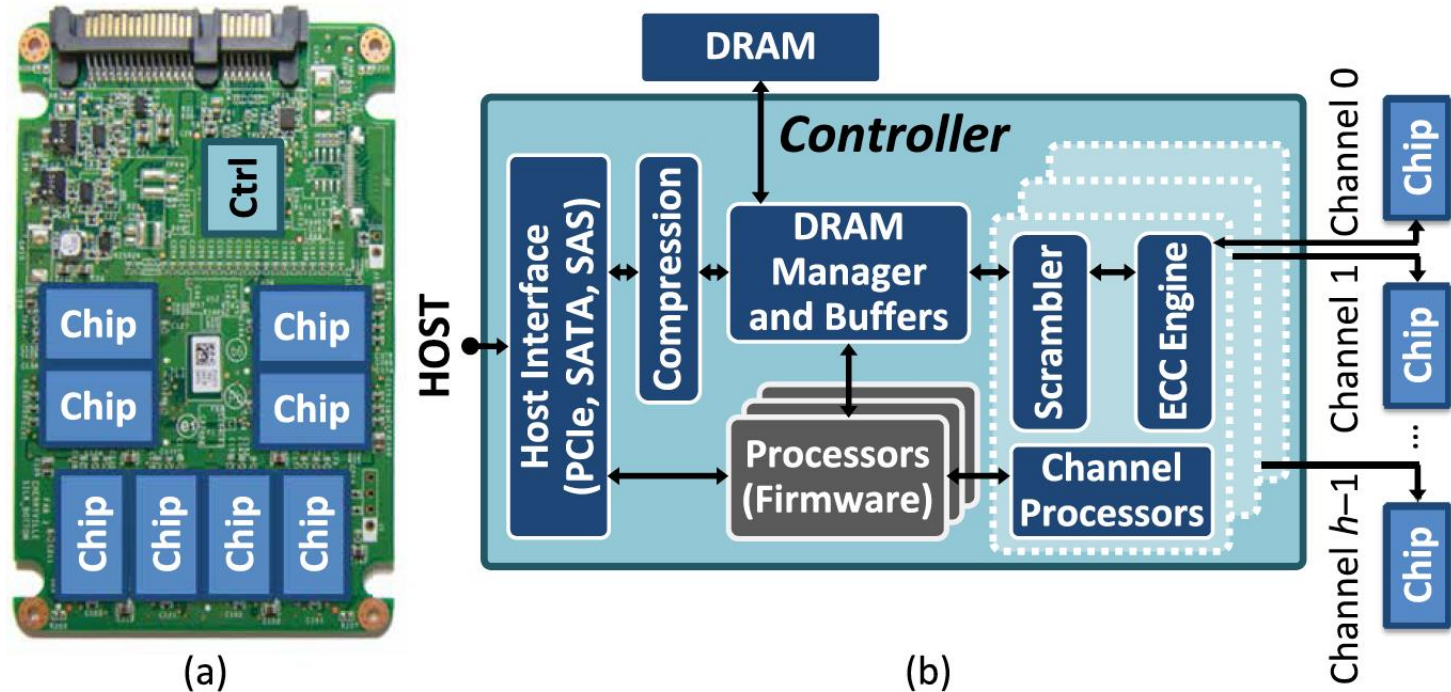


Fig. 1. (a) SSD system architecture, showing controller (Ctrl) and chips. (b) Detailed view of connections between controller components and chips.

On Modern SSD Controllers (I)



Proceedings of the IEEE, Sept. 2017

Error Characterization, Mitigation, and Recovery in Flash-Memory-Based Solid-State Drives

This paper reviews the most recent advances in solid-state drive (SSD) error characterization, mitigation, and data recovery techniques to improve both SSD's reliability and lifetime.

By YU CAI, SAUGATA GHOSE, ERICH F. HARATSCH, YIXIN LUO, AND ONUR MUTLU

On Modern SSD Controllers (II)

- Arash Tavakkol, Juan Gomez-Luna, Mohammad Sadrosadati, Saugata Ghose, and Onur Mutlu,
"MQSim: A Framework for Enabling Realistic Studies of Modern Multi-Queue SSD Devices"
Proceedings of the 16th USENIX Conference on File and Storage Technologies (FAST), Oakland, CA, USA, February 2018.
[[Slides \(pptx\)](#)] [[pdf](#)]
[[Source Code](#)]

MQSim: A Framework for Enabling Realistic Studies of Modern Multi-Queue SSD Devices

Arash Tavakkol[†], Juan Gómez-Luna[†], Mohammad Sadrosadati[†], Saugata Ghose[‡], Onur Mutlu^{†‡}
[†]*ETH Zürich* [‡]*Carnegie Mellon University*

On Modern SSD Controllers (III)

- Arash Tavakkol, Mohammad Sadrosadati, Saugata Ghose, Jeremie Kim, Yixin Luo, Yaohua Wang, Nika Mansouri Ghiasi, Lois Orosa, Juan G. Luna and Onur Mutlu,

"FLIN: Enabling Fairness and Enhancing Performance in Modern NVMe Solid State Drives"

Proceedings of the 45th International Symposium on Computer Architecture (ISCA), Los Angeles, CA, USA, June 2018.

[[Slides \(pptx\)](#)] [[pdf](#)] [[Lightning Talk Slides \(pptx\)](#)] [[pdf](#)]

[[Lightning Talk Video](#)]

FLIN: Enabling Fairness and Enhancing Performance in Modern NVMe Solid State Drives

Arash Tavakkol[†] Mohammad Sadrosadati[†] Saugata Ghose[‡] Jeremie S. Kim^{‡†} Yixin Luo[‡]
Yaohua Wang^{†§} Nika Mansouri Ghiasi[†] Lois Orosa^{†*} Juan Gómez-Luna[†] Onur Mutlu^{†‡}
[†]*ETH Zürich* [‡]*Carnegie Mellon University* [§]*NUDT* ^{*}*Unicamp*

DRAM Types

- DRAM has different types with different interfaces optimized for different purposes
 - ❑ Commodity: DDR, DDR2, DDR3, DDR4, ...
 - ❑ Low power (for mobile): LPDDR1, ..., LPDDR5, ...
 - ❑ High bandwidth (for graphics): GDDR2, ..., GDDR5, ...
 - ❑ Low latency: eDRAM, RDRAM, ...
 - ❑ 3D stacked: WIO, HBM, HMC, ...
 - ❑ ...
- Underlying microarchitecture is fundamentally the same
- A flexible memory controller can support various DRAM types
- This complicates the memory controller
 - ❑ Difficult to support all types (and upgrades)

DRAM Types (circa 2015)

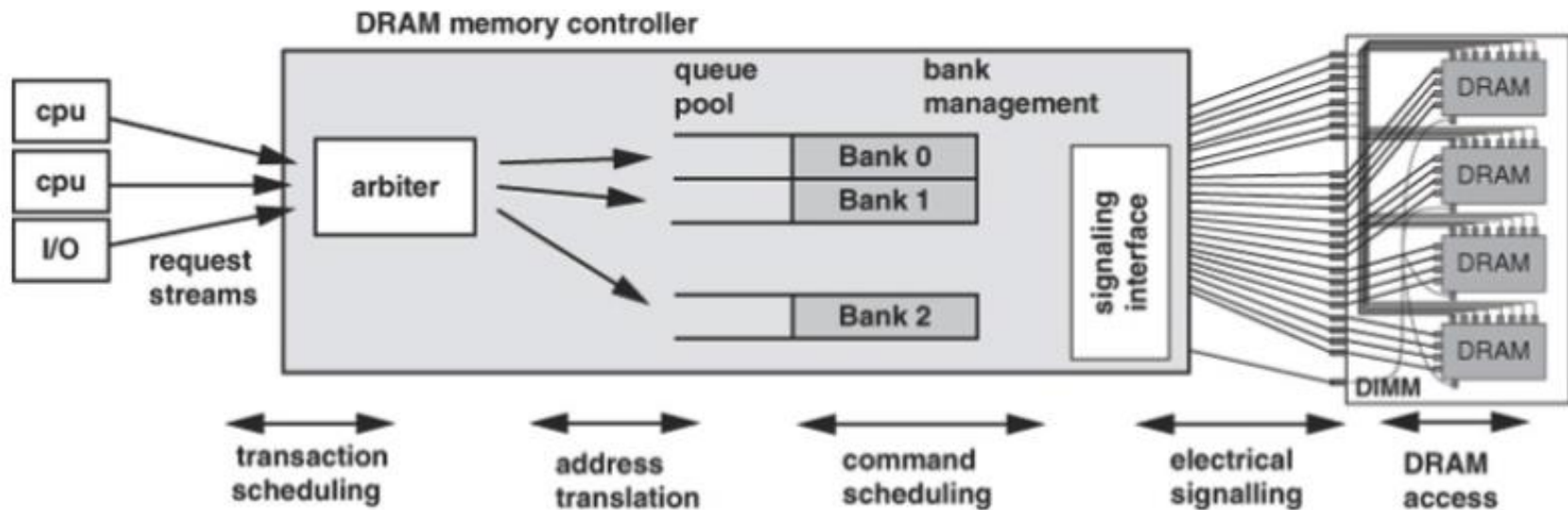
<i>Segment</i>	<i>DRAM Standards & Architectures</i>
Commodity	DDR3 (2007) [14]; DDR4 (2012) [18]
Low-Power	LPDDR3 (2012) [17]; LPDDR4 (2014) [20]
Graphics	GDDR5 (2009) [15]
Performance	eDRAM [28], [32]; RLDram3 (2011) [29]
3D-Stacked	WIO (2011) [16]; WIO2 (2014) [21]; MCDRAM (2015) [13]; HBM (2013) [19]; HMC1.0 (2013) [10]; HMC1.1 (2014) [11]
Academic	SBA/SSA (2010) [38]; Staged Reads (2012) [8]; RAIDR (2012) [27]; SALP (2012) [24]; TL-DRAM (2013) [26]; RowClone (2013) [37]; Half-DRAM (2014) [39]; Row-Buffer Decoupling (2014) [33]; SARP (2014) [6]; AL-DRAM (2015) [25]

Table 1. Landscape of DRAM-based memory

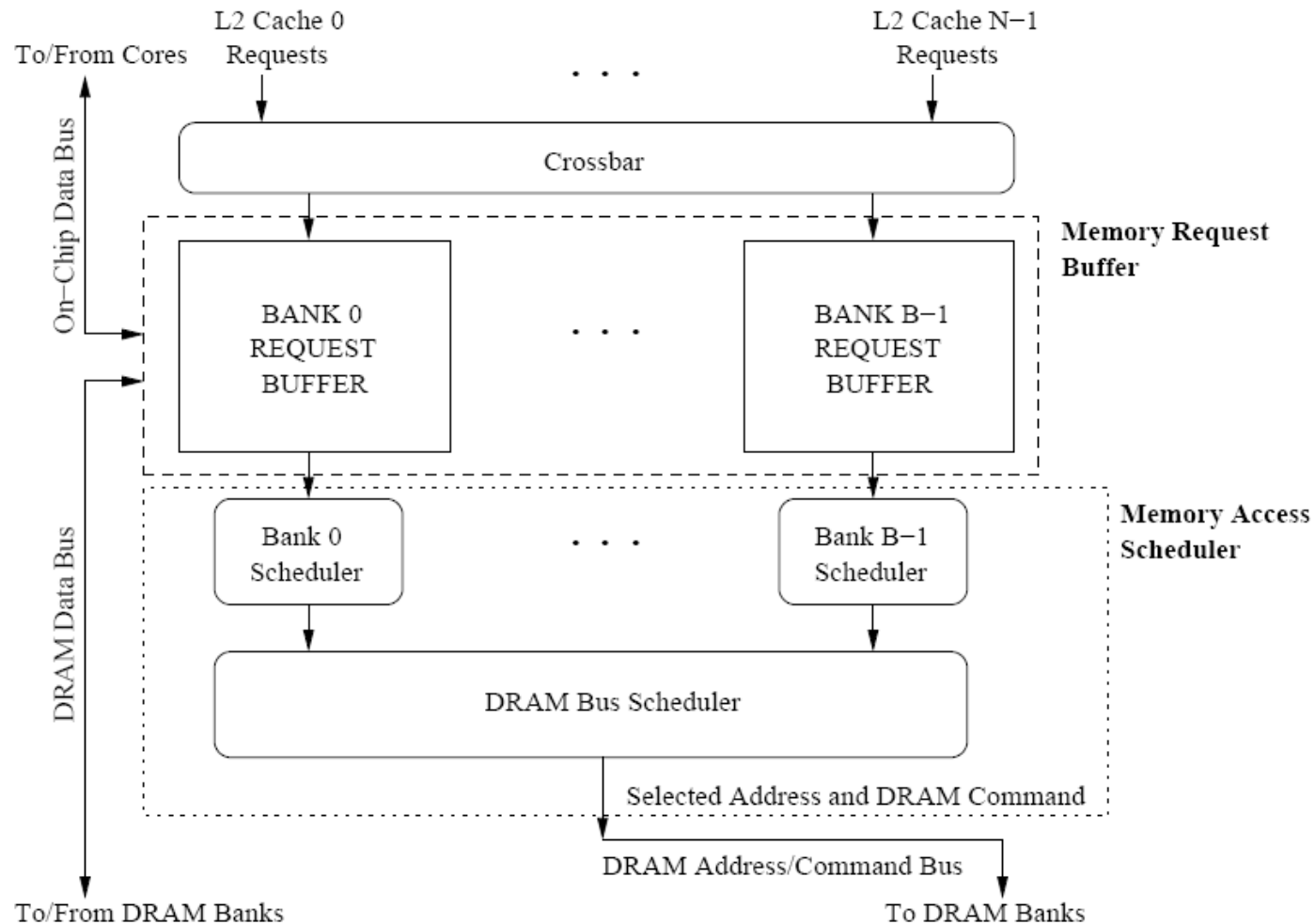
DRAM Controller: Functions

- Ensure correct operation of DRAM (refresh and timing)
- 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 for high performance + QoS
 - Reordering, row-buffer, bank, rank, bus management
- Manage power consumption and thermals in DRAM
 - Turn on/off DRAM chips, manage power modes

A Modern DRAM Controller (I)



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

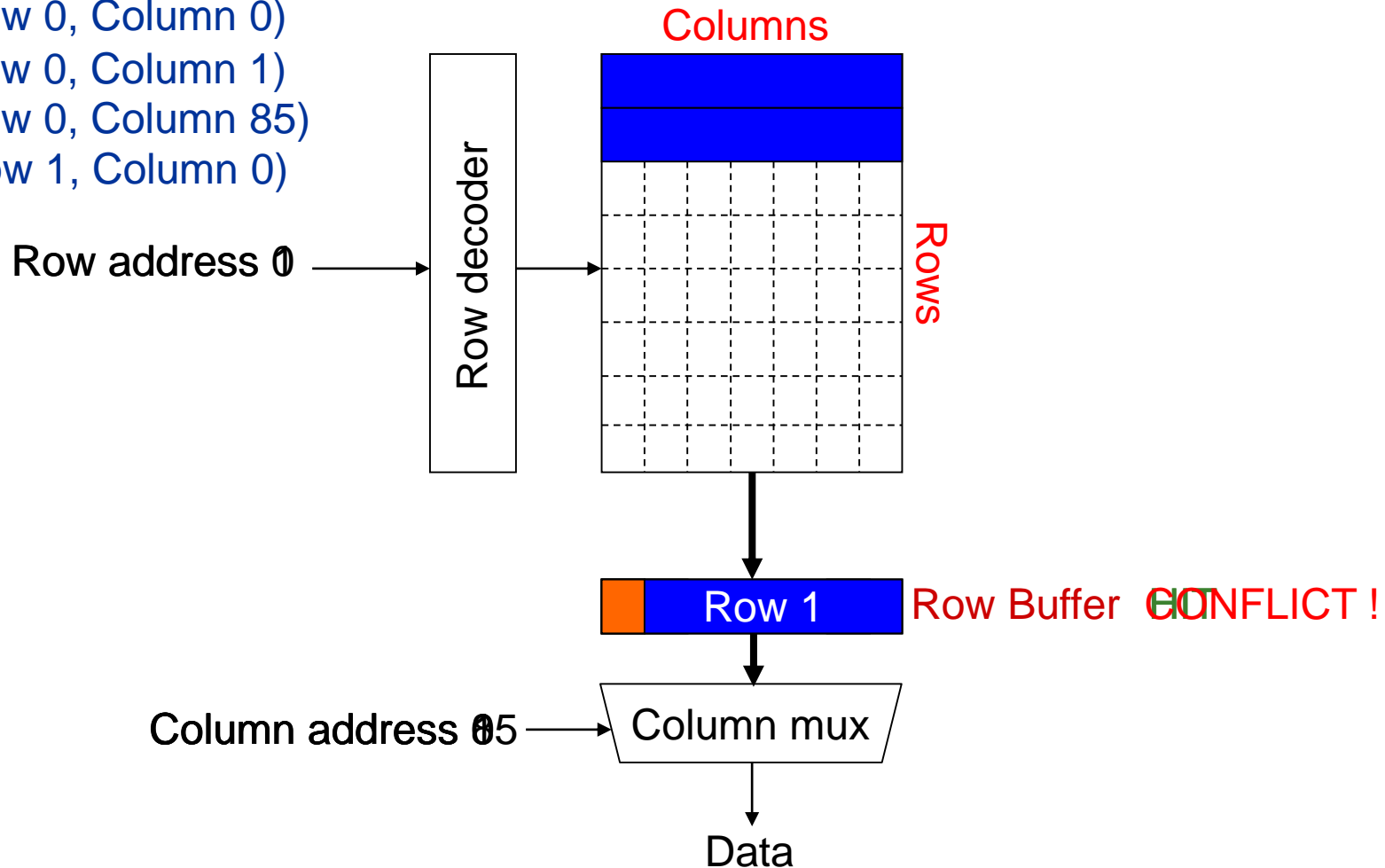
Goal: Maximize row buffer hit rate → **maximize DRAM throughput**

- 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

Review: DRAM Bank Operation

Access Address:
(Row 0, Column 0)
(Row 0, Column 1)
(Row 0, Column 85)
(Row 1, Column 0)



DRAM Scheduling Policies (II)

- A scheduling policy is a request 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?
 - Will it stall the processor?
 - Interference caused to other cores
 - ...

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 and act accordingly

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

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

Difficulty of DRAM Control

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 **manage power** consumption
- Need to **optimize performance & QoS** (in the presence of constraints)
 - Reordering is not simple
 - Fairness and QoS needs complicates the scheduling problem

Many DRAM Timing Constraints

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.

More on DRAM Operation

- Kim et al., "A Case for Exploiting Subarray-Level Parallelism (SALP) in DRAM," ISCA 2012.
- Lee et al., "Tiered-Latency DRAM: A Low Latency and Low Cost DRAM Architecture," HPCA 2013.

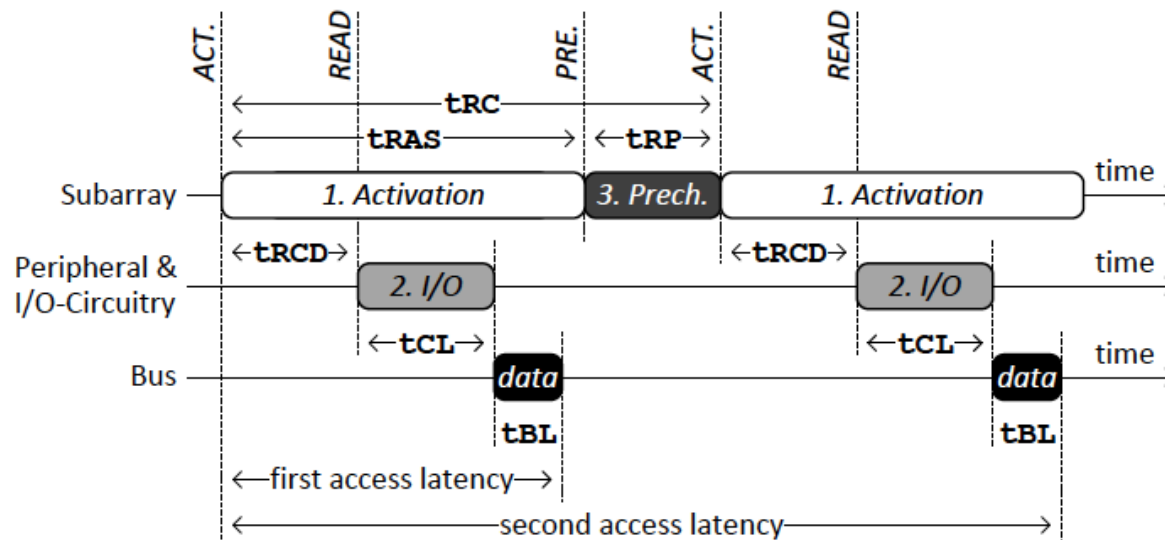


Figure 5. Three Phases of DRAM Access

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

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

Why So Many Timing Constraints? (I)

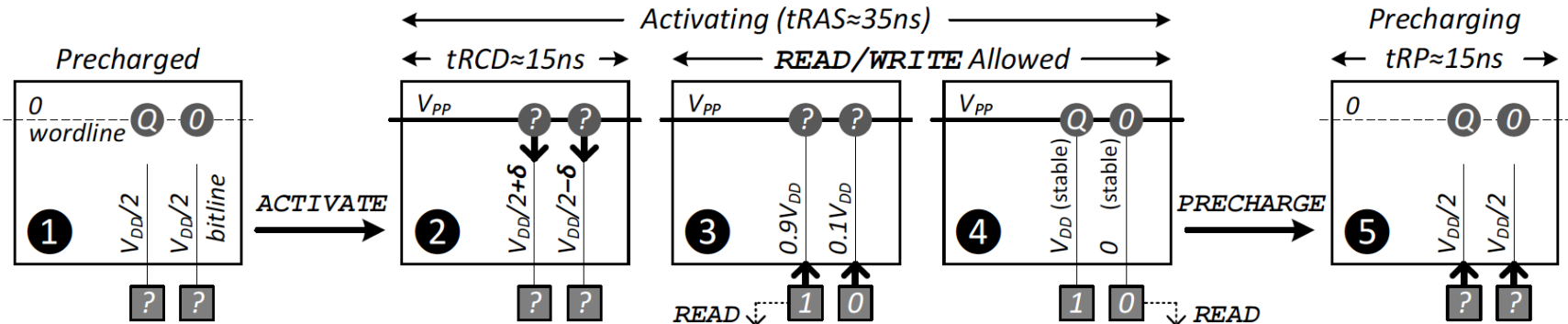


Figure 4. DRAM bank operation: Steps involved in serving a memory request [17] ($V_{PP} > V_{DD}$)

Category	RowCmd↔RowCmd			RowCmd↔ColCmd			ColCmd↔ColCmd			ColCmd→DATA	
Name	t_{RC}	t_{RAS}	t_{RP}	t_{RCD}	t_{RTP}	t_{WR}^*	t_{CCD}	t_{RTW}^\dagger	t_{WTR}^*	CL	CWL
Commands	A→A	A→P	P→A	A→R/W	R→P	W*→P	R(W)→R(W)	R→W	W*→R	R→DATA	W→DATA
Scope	Bank	Bank	Bank	Bank	Bank	Bank	Channel	Rank	Rank	Bank	Bank
Value (ns)	~50	~35	13-15	13-15	~7.5	15	5-7.5	11-15	~7.5	13-15	10-15

A: ACTIVATE– P: PRECHARGE– R: READ– W: WRITE

* Goes into effect after the last write *data*, not from the WRITE command

† Not explicitly specified by the JEDEC DDR3 standard [18]. Defined as a function of other timing constraints.

Table 1. Summary of DDR3-SDRAM timing constraints (derived from Micron’s 2Gb DDR3-SDRAM datasheet [33])

Kim et al., “A Case for Exploiting Subarray-Level Parallelism (SALP) in DRAM,” ISCA 2012.

Why So Many Timing Constraints? (II)

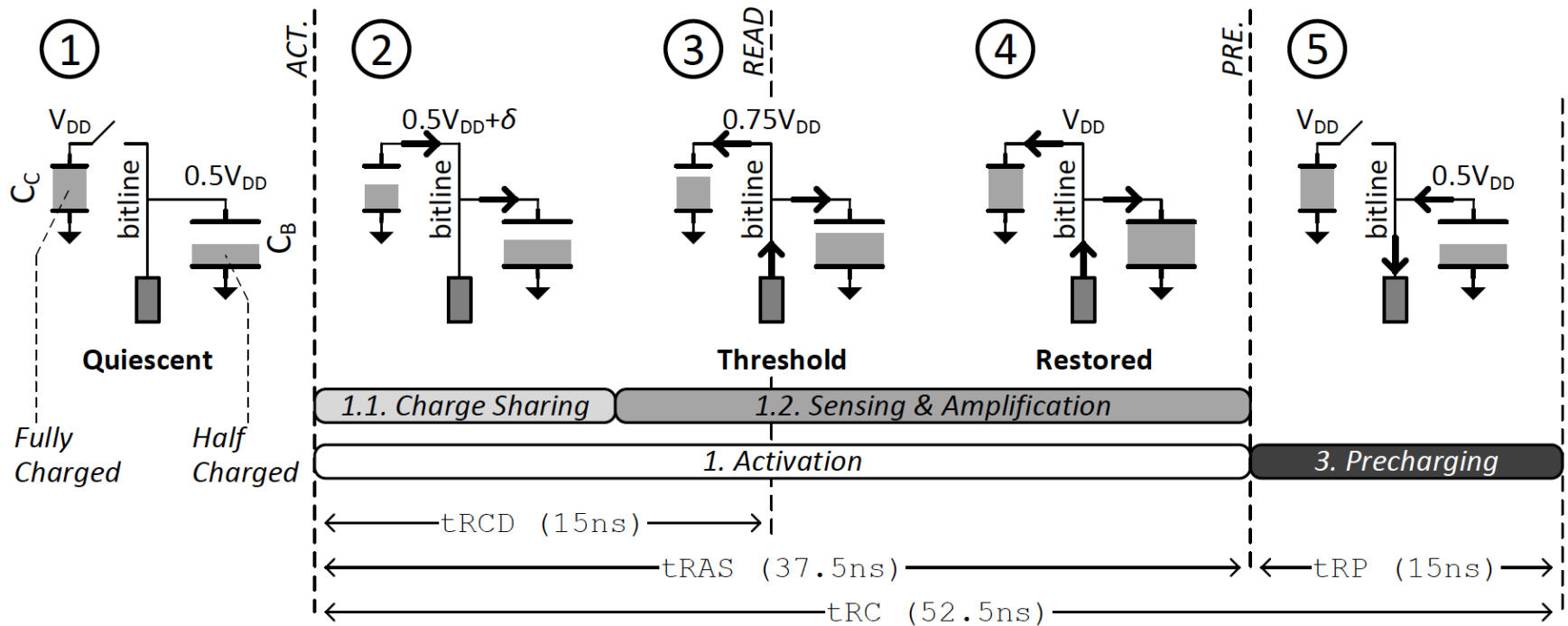


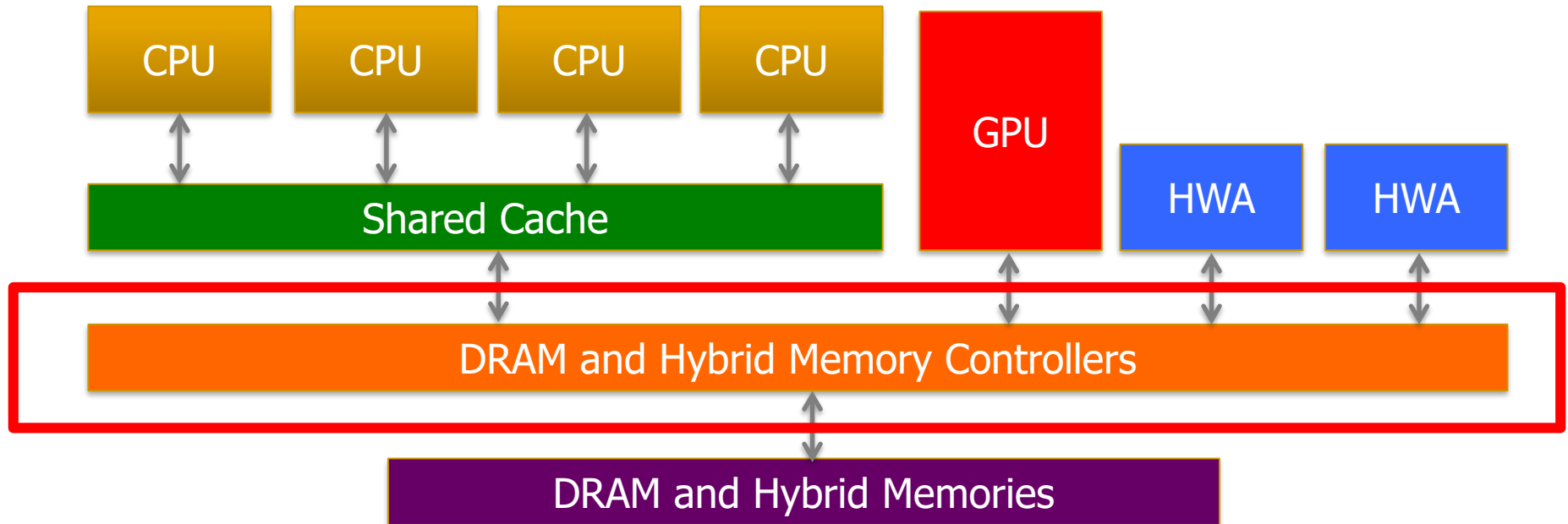
Figure 6. Charge Flow Between the Cell Capacitor (C_C), Bitline Parasitic Capacitor (C_B), and the Sense-Amplifier ($C_B \approx 3.5C_C$ [39])

Lee et al., "Tiered-Latency DRAM: A Low Latency and Low Cost DRAM Architecture," HPCA 2013.

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

Phase	Commands	Name	Value
1	ACT → READ	t_{RCD}	15ns
	ACT → WRITE		
	ACT → PRE	t_{RAS}	37.5ns
2	READ → data	t_{CL}	15ns
	WRITE → data	t_{CWL}	11.25ns
	data burst	t_{BL}	7.5ns
3	PRE → ACT	t_{RP}	15ns
1 & 3	ACT → ACT	t_{RC} ($t_{RAS} + t_{RP}$)	52.5ns

DRAM Controller Design Is Becoming More Difficult



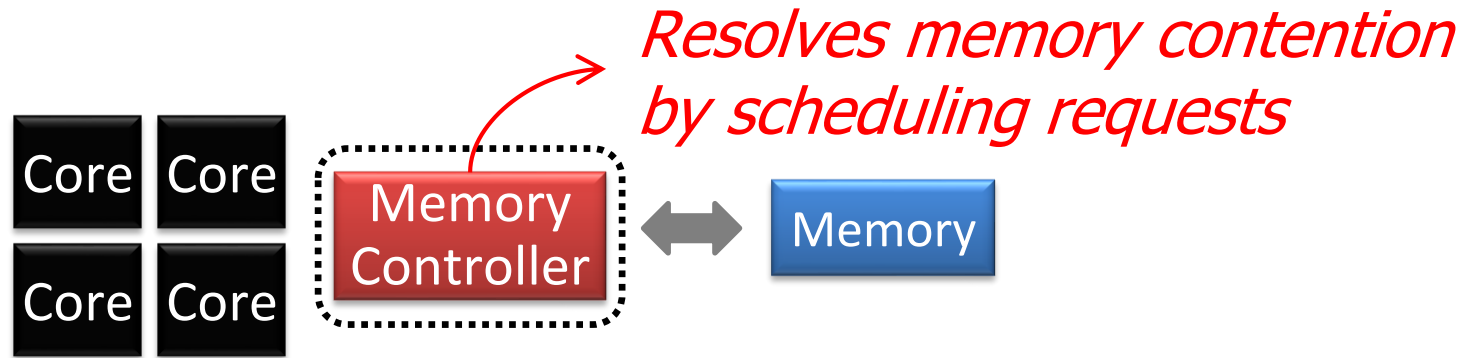
- Heterogeneous agents: CPUs, GPUs, and HWAs
- Main memory interference between CPUs, GPUs, HWAs
- Many timing constraints for various memory types
- Many goals at the same time: performance, fairness, QoS, energy efficiency, ...

Reality and Dream

- Reality: It is difficult to design a policy that maximizes performance, QoS, energy-efficiency, ...
 - Too many things to think about
 - Continuously changing workload and system behavior

- Dream: Wouldn't it be nice if the DRAM controller automatically found a good scheduling policy on its own?

Memory Controller: Performance Function



How to schedule requests to maximize system performance?

Self-Optimizing DRAM Controllers

- Problem: DRAM controllers are 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: A memory controller that adapts its scheduling policy 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
 - It dynamically and continuously learns and employs the best scheduling policy to maximize long-term performance.

Self-Optimizing DRAM Controllers

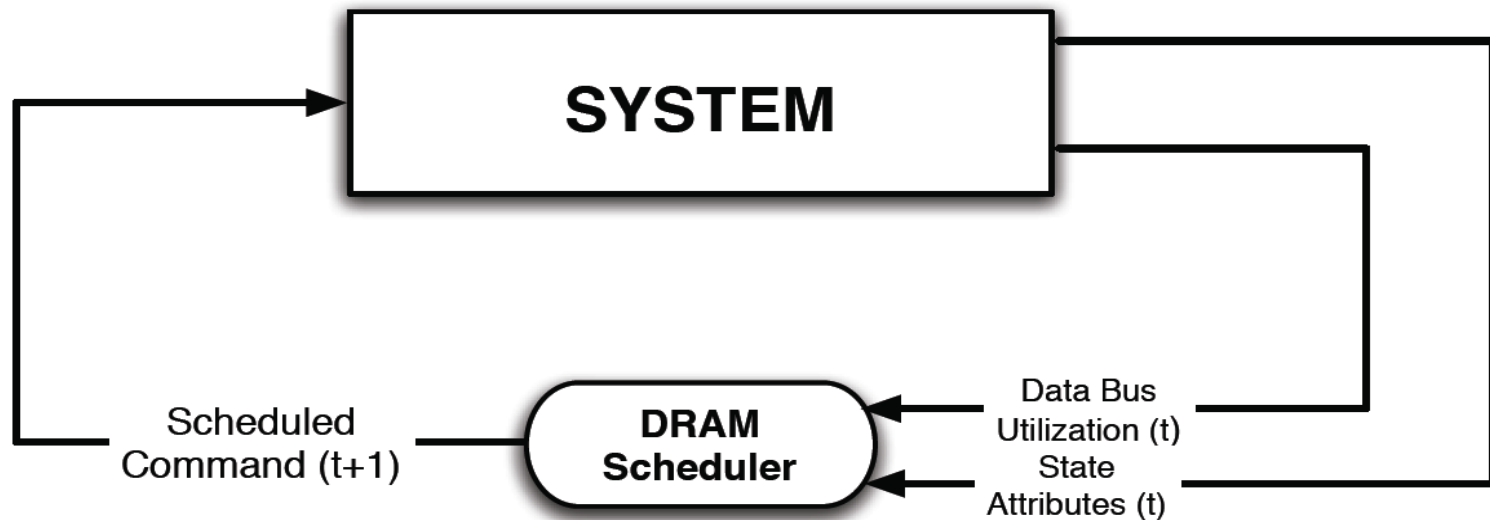


Goal: Learn to choose actions to maximize $r_0 + \gamma r_1 + \gamma^2 r_2 + \dots$ ($0 \leq \gamma < 1$)

Figure 2: (a) Intelligent agent based on reinforcement learning principles;

Self-Optimizing DRAM Controllers

- Dynamically adapt the memory scheduling policy via interaction with the system at runtime
 - Associate system states and actions (commands) with long term reward values: **each action at a given state leads to a learned reward**
 - **Schedule command with highest estimated long-term reward value in each state**
 - **Continuously update reward values for $\langle \text{state}, \text{action} \rangle$ pairs based on feedback from system**



Self-Optimizing DRAM Controllers

- Engin Ipek, Onur Mutlu, José F. Martínez, and Rich Caruana,
"Self Optimizing Memory Controllers: A Reinforcement Learning Approach"

Proceedings of the 35th International Symposium on Computer Architecture (ISCA), pages 39-50, Beijing, China, June 2008.

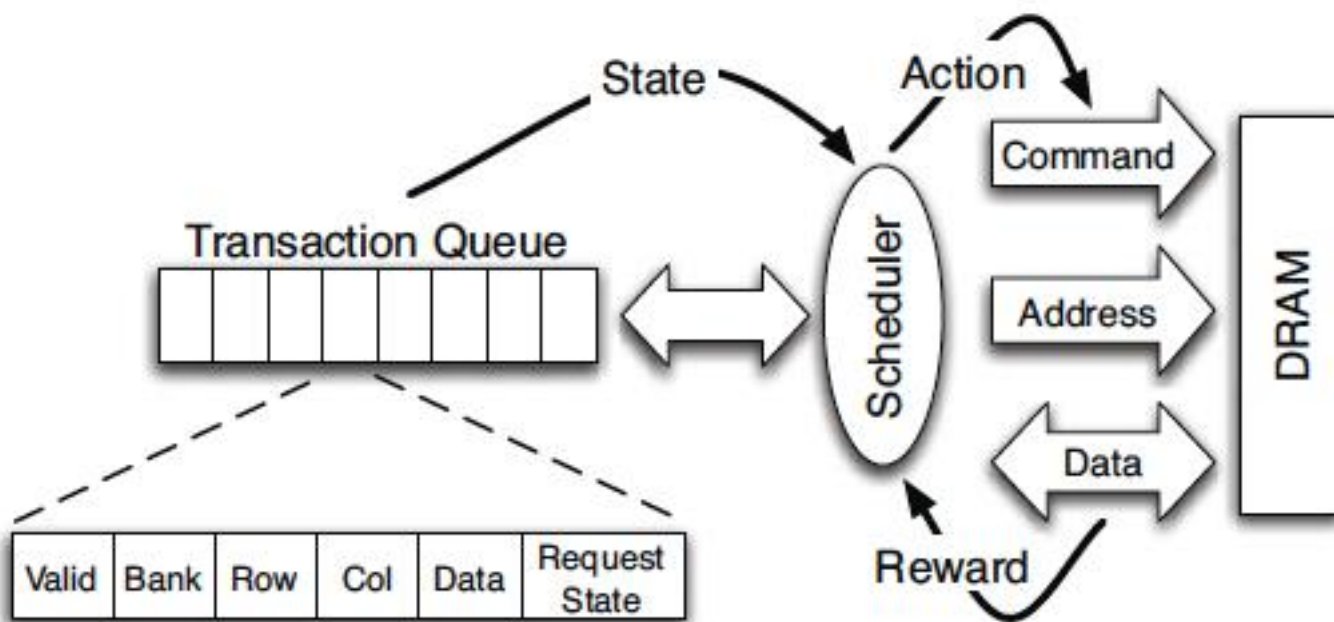


Figure 4: High-level overview of an RL-based scheduler.

States, Actions, Rewards

❖ Reward function

- +1 for scheduling Read and Write commands
- 0 at all other times

Goal is to maximize long-term data bus utilization

❖ State attributes

- Number of reads, writes, and load misses in transaction queue
- Number of pending writes and ROB heads waiting for referenced row
- Request's relative ROB order

❖ Actions

- Activate
- Write
- Read - load miss
- Read - store miss
- Precharge - pending
- Precharge - preemptive
- NOP

Performance Results

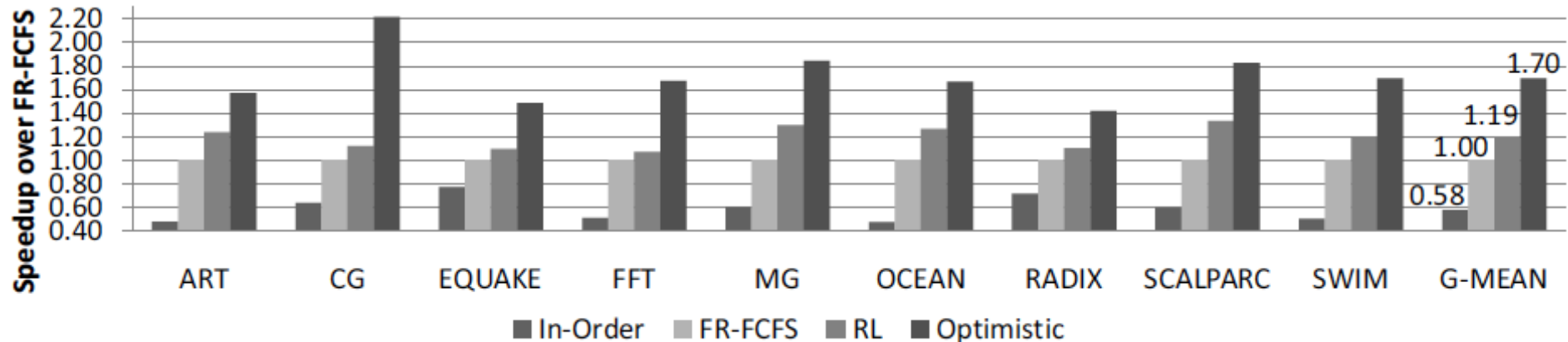


Figure 7: Performance comparison of in-order, FR-FCFS, RL-based, and optimistic memory controllers

Large, robust performance improvements over many human-designed policies

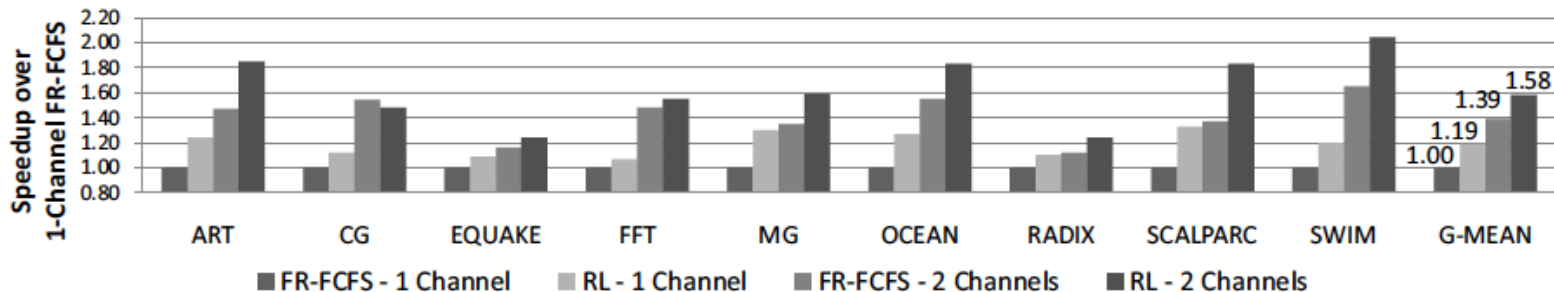


Figure 15: Performance comparison of FR-FCFS and RL-based memory controllers on systems with 6.4GB/s and 12.8GB/s peak DRAM bandwidth

Self Optimizing DRAM Controllers

+ Continuous learning in the presence of changing environment

+ Reduced designer burden in finding a good scheduling policy.

Designer specifies:

1) What system variables might be useful

2) What target to optimize, but not how to optimize it

-- How to specify different objectives? (e.g., fairness, QoS, ...)

-- Hardware complexity?

-- Design mindset and flow

More on Self-Optimizing DRAM Controllers

- Engin Ipek, Onur Mutlu, José F. Martínez, and Rich Caruana,
"Self Optimizing Memory Controllers: A Reinforcement Learning Approach"
Proceedings of the 35th International Symposium on Computer Architecture (ISCA), pages 39-50, Beijing, China, June 2008.

Self-Optimizing Memory Controllers: A Reinforcement Learning Approach

Engin İpek^{1,2} Onur Mutlu² José F. Martínez¹ Rich Caruana¹

¹Cornell University, Ithaca, NY 14850 USA

²Microsoft Research, Redmond, WA 98052 USA

Self-Optimizing (Data-Driven) Computing Architectures

System Architecture Design Today

- Human-driven
 - Humans design the policies (how to do things)
- Many (too) simple, short-sighted policies all over the system
- No automatic data-driven policy learning
- (Almost) no learning: cannot take lessons from past actions

**Can we design
fundamentally intelligent architectures?**

An Intelligent Architecture

- Data-driven
 - Machine learns the “best” policies (how to do things)
- Sophisticated, workload-driven, changing, far-sighted policies
- Automatic data-driven policy learning
- All controllers are intelligent data-driven agents

**We need to rethink design
(of all controllers)**

Computer Architecture

Lecture 13a: Memory Controllers

Prof. Onur Mutlu

ETH Zürich

Fall 2019

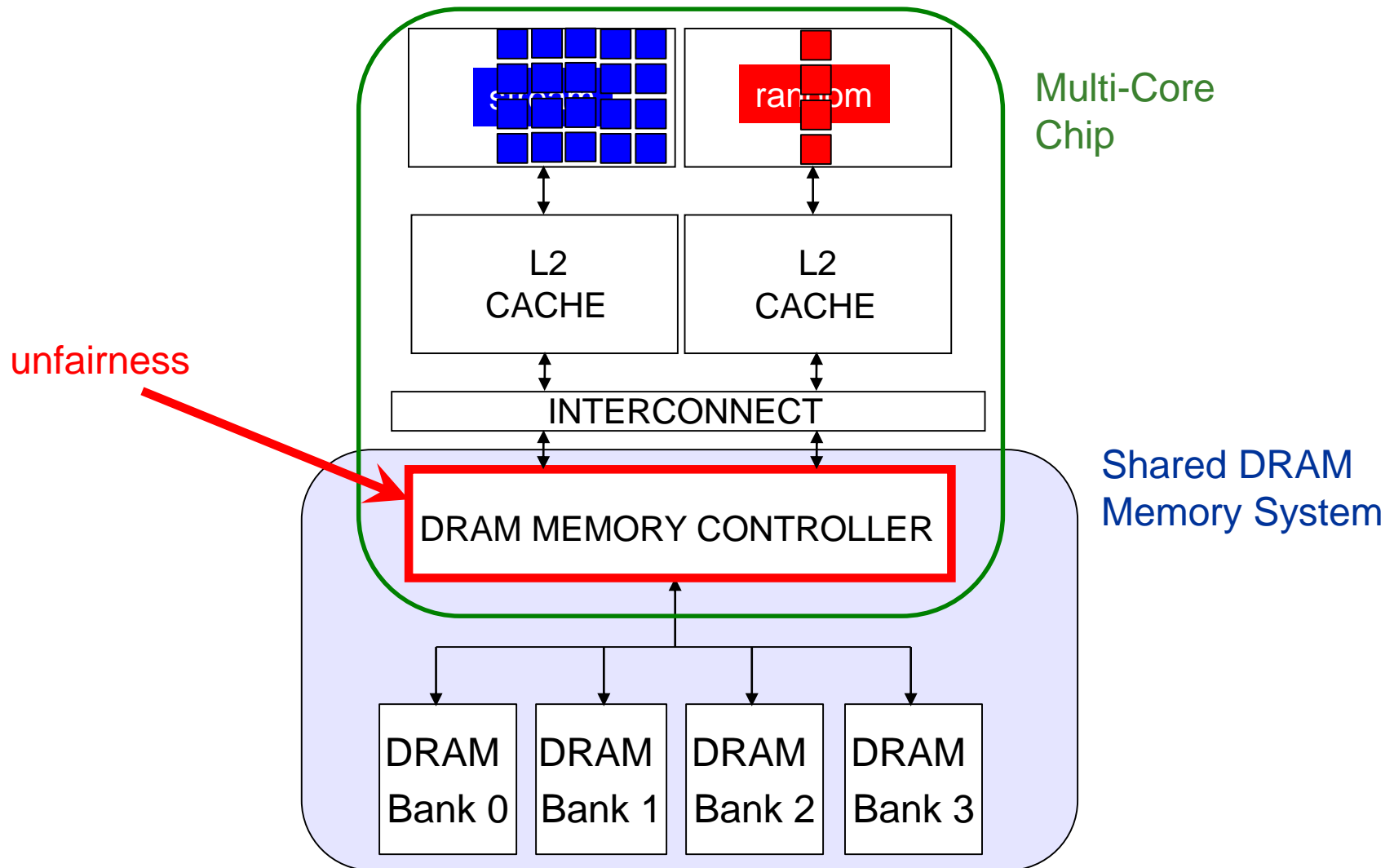
31 October 2019

Memory Interference

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

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];  
    ...  
}
```

STREAM

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

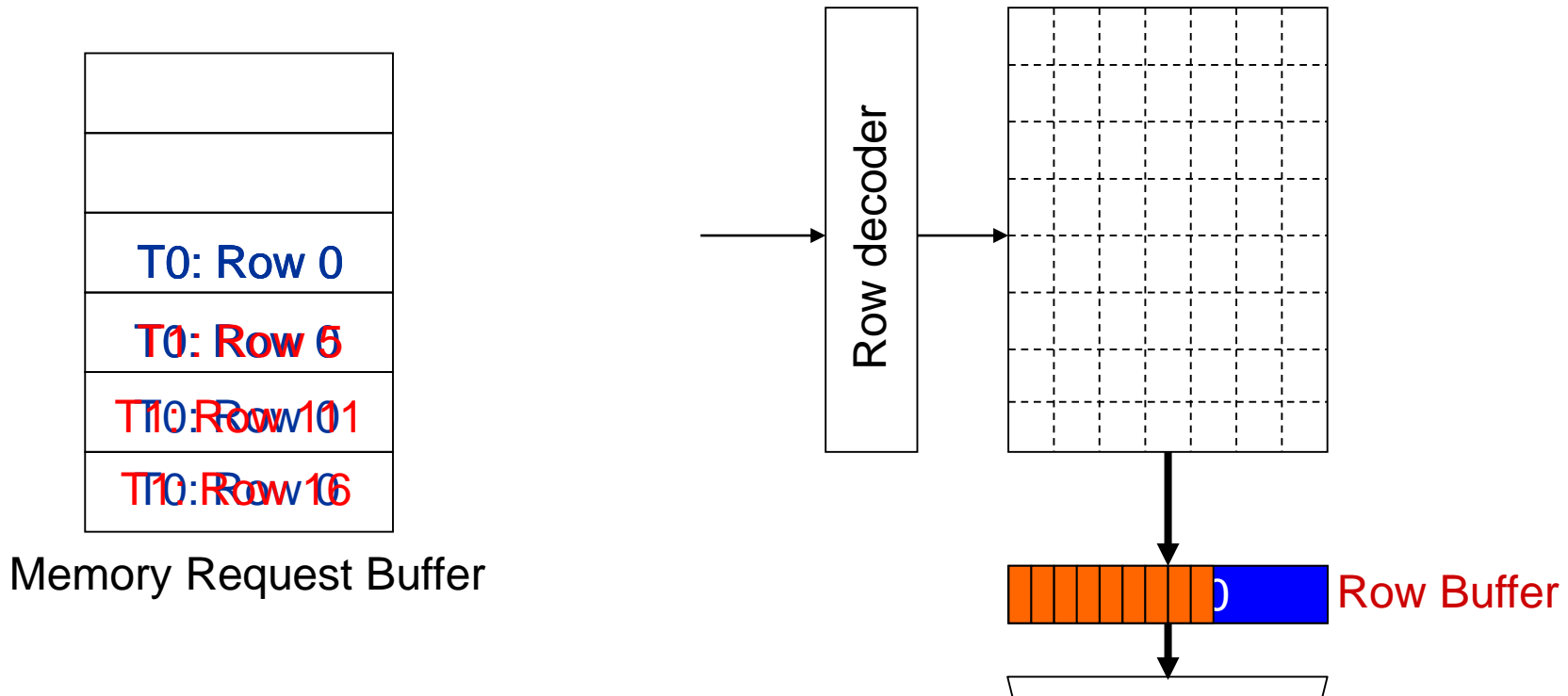
```
// initialize large arrays A, B  
for (j=0; j<N; j++) {  
    index = rand(); random  
    A[index] = B[index];  
    ...  
}
```

RANDOM

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

Moscibroda and Mutlu, “[Memory Performance Attacks](#),” USENIX Security 2007.

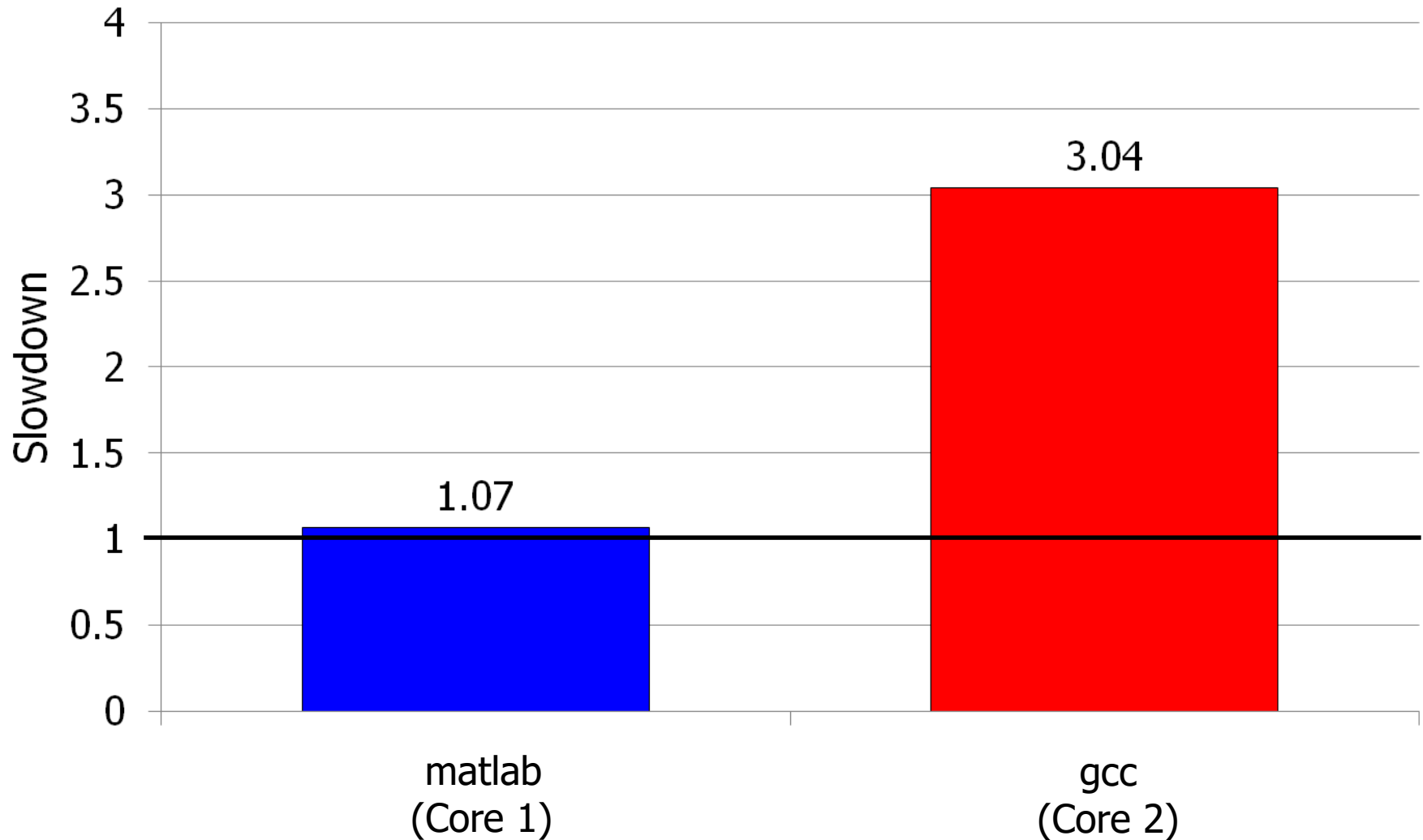
What Does the Memory Hog Do?



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.

Unfair Slowdowns due to Interference



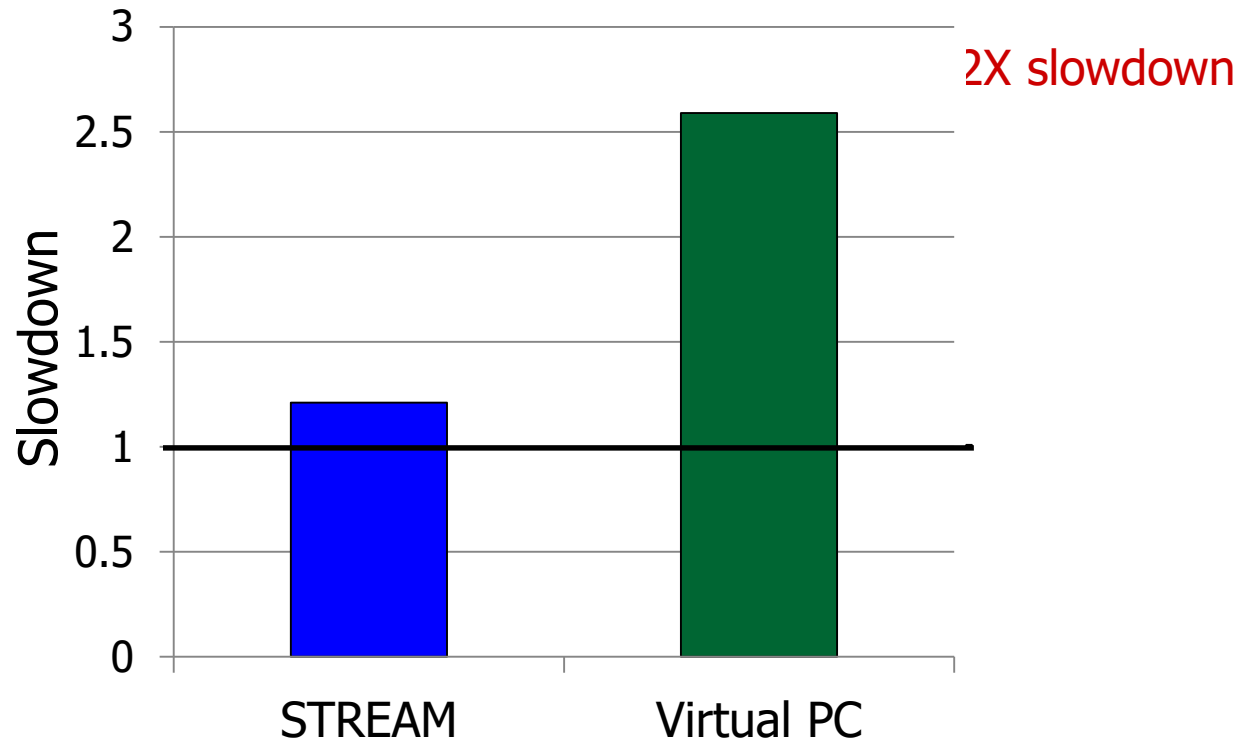
DRAM Controllers

- 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.

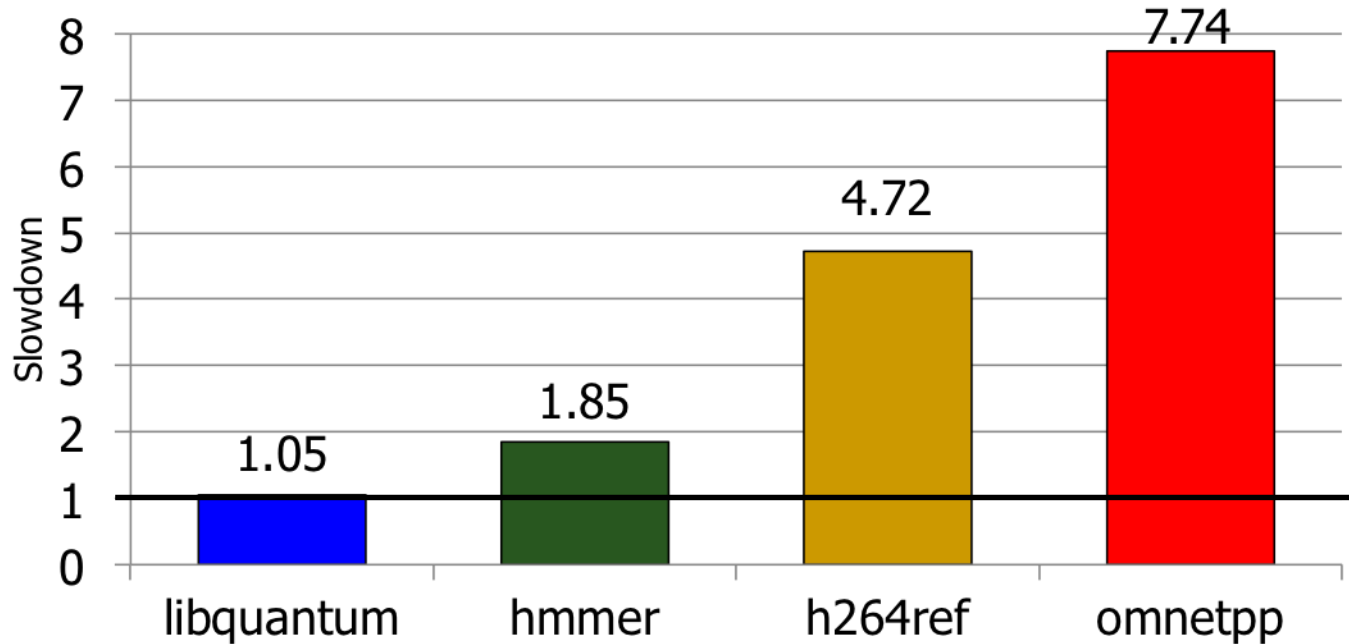
Effect of the Memory Performance Hog



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.

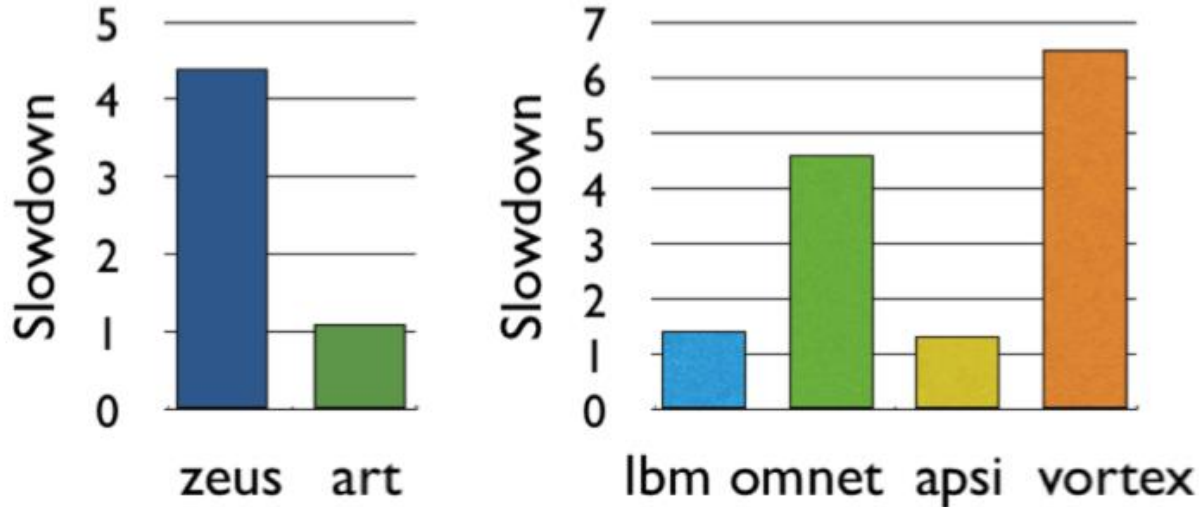
Greater Problem with More Cores



- 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

More on Memory Performance Attacks

- Thomas Moscibroda and Onur Mutlu,
"Memory Performance Attacks: Denial of Memory Service in Multi-Core Systems"
*Proceedings of the 16th USENIX Security Symposium (**USENIX SECURITY**), pages 257-274, Boston, MA, August 2007. Slides (ppt)*

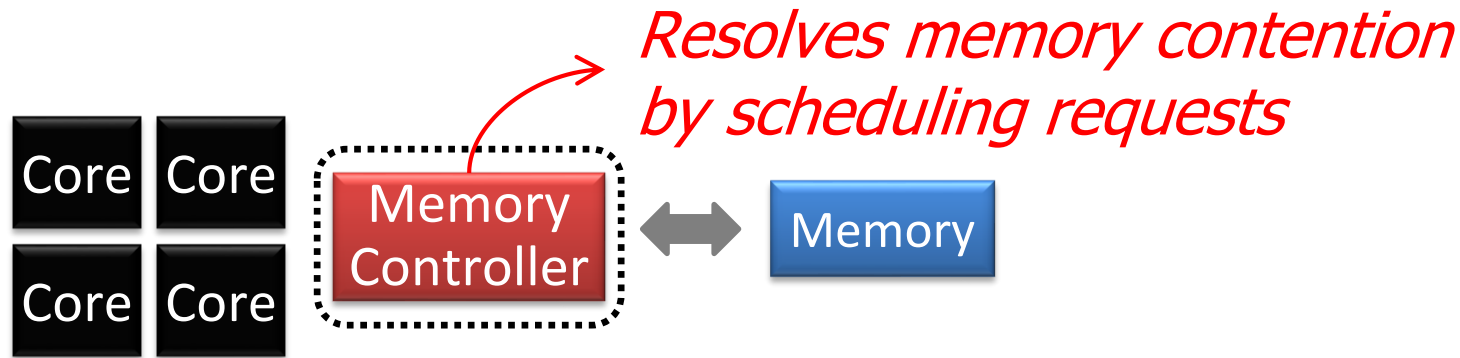
Memory Performance Attacks: Denial of Memory Service in Multi-Core Systems

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

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 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: Readings (I)

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

Stall-Time Fair Memory Access Scheduling for Chip Multiprocessors

Onur Mutlu Thomas Moscibroda

Microsoft Research
{onur,moscitho}@microsoft.com

QoS-Aware Memory: Readings (II)

- 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

QoS-Aware Memory: Readings (III)

- Yoongu Kim, Dongsu Han, Onur Mutlu, and Mor Harchol-Balter, **"ATLAS: A Scalable and High-Performance Scheduling Algorithm for Multiple Memory Controllers"**
Proceedings of the 16th International Symposium on High-Performance Computer Architecture (HPCA), Bangalore, India, January 2010. [Slides \(pptx\)](#)

ATLAS: A Scalable and High-Performance Scheduling Algorithm for Multiple Memory Controllers

Yoongu Kim Dongsu Han Onur Mutlu Mor Harchol-Balter
Carnegie Mellon University

QoS-Aware Memory: Readings (IV)

- Yoongu Kim, Michael Papamichael, Onur Mutlu, and Mor Harchol-Balter,

"Thread Cluster Memory Scheduling: Exploiting Differences in Memory Access Behavior"

*Proceedings of the 43rd International Symposium on Microarchitecture (**MICRO**), pages 65-76, Atlanta, GA, December 2010. [Slides \(pptx\)](#) [\(pdf\)](#)*

Thread Cluster Memory Scheduling: Exploiting Differences in Memory Access Behavior

Yoongu Kim

yoonguk@ece.cmu.edu

Michael Papamichael

papamix@cs.cmu.edu

Onur Mutlu

onur@cmu.edu

Mor Harchol-Balter

harchol@cs.cmu.edu

Carnegie Mellon University

QoS-Aware Memory: Readings (V)

- Lavanya Subramanian, Donghyuk Lee, Vivek Seshadri, Harsha Rastogi, and Onur Mutlu,
"The Blacklisting Memory Scheduler: Achieving High Performance and Fairness at Low Cost"
Proceedings of the 32nd IEEE International Conference on Computer Design (ICCD), Seoul, South Korea, October 2014.
[[Slides \(pptx\)](#) ([pdf](#))]

The Blacklisting Memory Scheduler: Achieving High Performance and Fairness at Low Cost

Lavanya Subramanian, Donghyuk Lee, Vivek Seshadri, Harsha Rastogi, Onur Mutlu
Carnegie Mellon University
{lsubrama,donghyu1,visesh,harshar,onur}@cmu.edu

QoS-Aware Memory: Readings (VI)

- Lavanya Subramanian, Donghyuk Lee, Vivek Seshadri, Harsha Rastogi, and Onur Mutlu,
"BLISS: Balancing Performance, Fairness and Complexity in Memory Access Scheduling"
IEEE Transactions on Parallel and Distributed Systems (***TPDS***), to appear in 2016. [arXiv.org version](#), April 2015.
[An earlier version](#) as *SAFARI Technical Report*, TR-SAFARI-2015-004, Carnegie Mellon University, March 2015.
[\[Source Code\]](#)

BLISS: Balancing Performance, Fairness and Complexity in Memory Access Scheduling

Lavanya Subramanian, Donghyuk Lee, Vivek Seshadri, Harsha Rastogi, and Onur Mutlu

QoS-Aware Memory: Readings (VII)

- Rachata Ausavarungnirun, Kevin Chang, Lavanya Subramanian, Gabriel Loh, and Onur Mutlu,
"Staged Memory Scheduling: Achieving High Performance and Scalability in Heterogeneous Systems"
Proceedings of the 39th International Symposium on Computer Architecture (ISCA), Portland, OR, June 2012. Slides (pptx)

Staged Memory Scheduling: Achieving High Performance and Scalability in Heterogeneous Systems

Rachata Ausavarungnirun[†] Kevin Kai-Wei Chang[†] Lavanya Subramanian[†] Gabriel H. Loh[‡] Onur Mutlu[†]

[†]Carnegie Mellon University
{rachata,kevincha,lsubrama,onur}@cmu.edu

[‡]Advanced Micro Devices, Inc.
gabe.loh@amd.com

QoS-Aware Memory: Readings (VIII)

- Hiroyuki Usui, Lavanya Subramanian, Kevin Kai-Wei Chang, and Onur Mutlu,
"DASH: Deadline-Aware High-Performance Memory Scheduler for Heterogeneous Systems with Hardware Accelerators"
ACM Transactions on Architecture and Code Optimization (TACO), Vol. 12, January 2016.
Presented at the 11th HiPEAC Conference, Prague, Czech Republic, January 2016.
[Slides (pptx)] [pdf]
[Source Code]

DASH: Deadline-Aware High-Performance Memory Scheduler for Heterogeneous Systems with Hardware Accelerators

HIROYUKI USUI, LAVANYA SUBRAMANIAN, KEVIN KAI-WEI CHANG,
and ONUR MUTLU, Carnegie Mellon University

QoS-Aware Memory: Readings (IX)

- Lavanya Subramanian, Vivek Seshadri, Yoongu Kim, Ben Jaiyen, and Onur Mutlu,
"MISE: Providing Performance Predictability and Improving Fairness in Shared Main Memory Systems"
Proceedings of the 19th International Symposium on High-Performance Computer Architecture (HPCA), Shenzhen, China, February 2013. [Slides \(pptx\)](#)

MISE: Providing Performance Predictability and Improving Fairness in Shared Main Memory Systems

Lavanya Subramanian

Vivek Seshadri

Yoongu Kim

Ben Jaiyen

Onur Mutlu

Carnegie Mellon University

QoS-Aware Memory: Readings (X)

- Lavanya Subramanian, Vivek Seshadri, Arnab Ghosh, Samira Khan, and Onur Mutlu,
"The Application Slowdown Model: Quantifying and Controlling the Impact of Inter-Application Interference at Shared Caches and Main Memory"
Proceedings of the 48th International Symposium on Microarchitecture (MICRO), Waikiki, Hawaii, USA, December 2015.
[[Slides \(pptx\)](#)] [[pdf](#)] [[Lightning Session Slides \(pptx\)](#)] [[pdf](#)] [[Poster \(pptx\)](#)] [[pdf](#)]
[[Source Code](#)]

The Application Slowdown Model: Quantifying and Controlling the Impact of Inter-Application Interference at Shared Caches and Main Memory

Lavanya Subramanian*§ Vivek Seshadri* Arnab Ghosh*†
Samira Khan*‡ Onur Mutlu*

*Carnegie Mellon University §Intel Labs †IIT Kanpur ‡University of Virginia