Computer Architecture

Lecture 5: DRAM Operation, Memory Control & Memory Latency

Prof. Onur Mutlu

ETH Zürich

Fall 2017

4 October 2017

High-Level Summary of Last Lecture

- Enabling High Bandwidth Memories
- Main Memory System: A Broad Perspective
- DRAM Fundamentals and Operation

Agenda for Today

- DRAM Operation Continued
- Memory Controllers
- Memory Latency

Lab 1 is Out

- Data Cache
- Implement a Data Cache in a Pipelined Processor
- A lot of extra credit opportunity.
- It should be a lot of fun.
- Due 18 October.

The Main Memory System and DRAM

Required Readings on DRAM

- DRAM Organization and Operation Basics
  - Sections 1 and 2 of: Lee et al., "Tiered-Latency DRAM: A Low Latency and Low Cost DRAM Architecture," HPCA 2013.

    https://people.inf.ethz.ch/omutlu/pub/tldram_hpca13.pdf

  - Sections 1 and 2 of Kim et al., "A Case for Subarray-Level Parallelism (SALP) in DRAM," ISCA 2012.

    https://people.inf.ethz.ch/omutlu/pub/salp-dram_isca12.pdf

- DRAM Refresh Basics
  - Sections 1 and 2 of Liu et al., "RAIDR: Retention-Aware Intelligent DRAM Refresh," ISCA 2012. https://people.inf.ethz.ch/omutlu/pub/raidr-dram-refresh_isca12.pdf

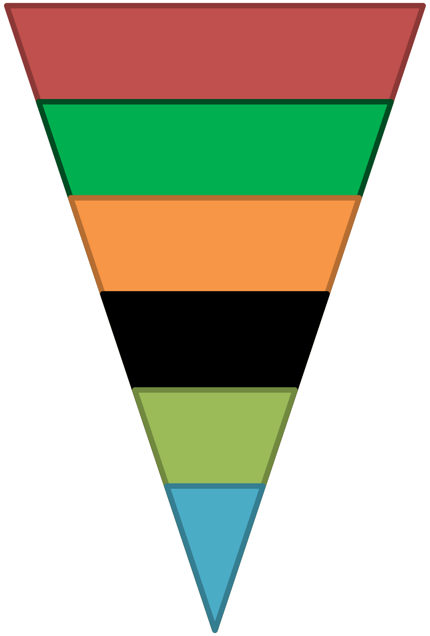
Reading on Simulating Main Memory

- How to evaluate future main memory systems?
- An open-source simulator and its brief description

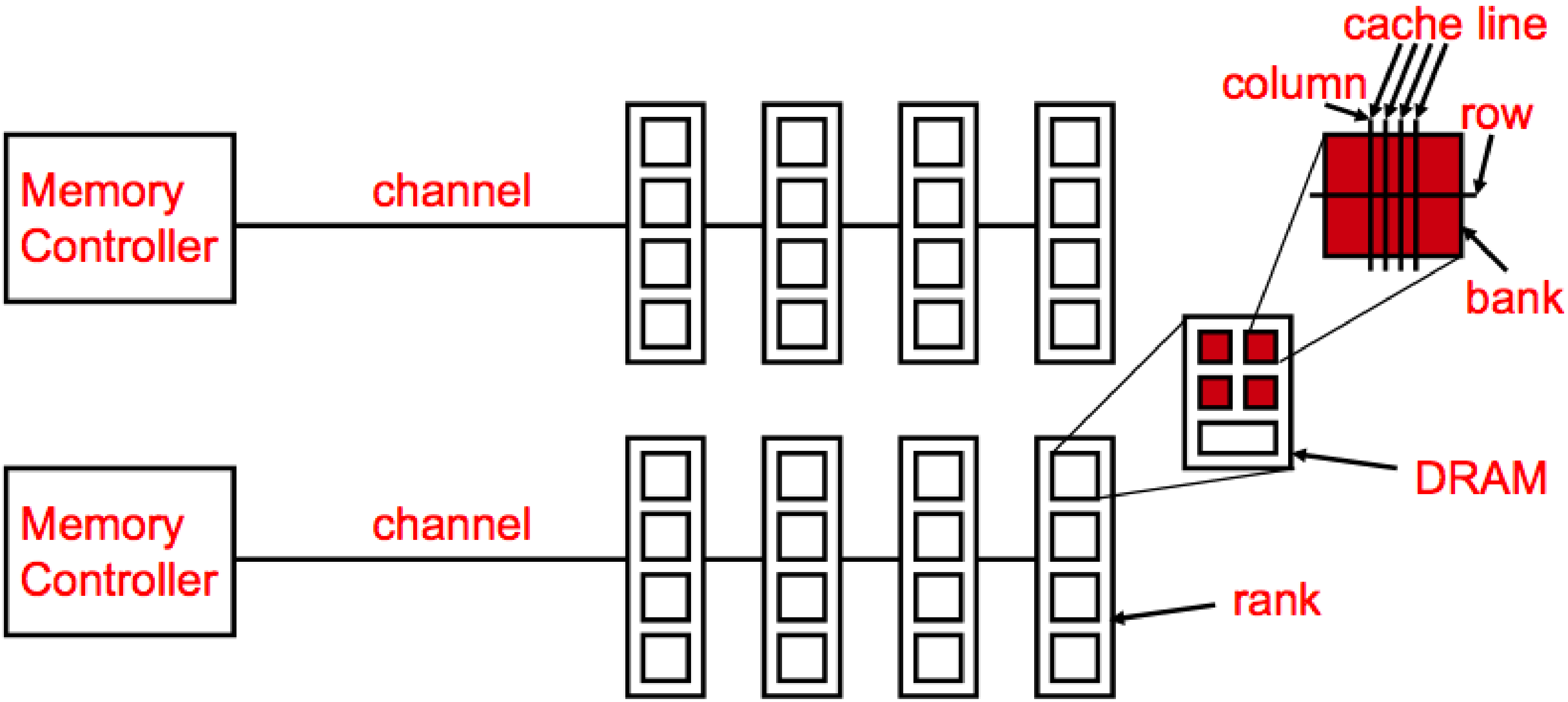
- Yoongu Kim, Weikun Yang, and Onur Mutlu,
"Ramulator: A Fast and Extensible DRAM Simulator"
IEEE Computer Architecture Letters ***(CAL)***, March 2015.
[Source Code]

Review: DRAM Subsystem Organization

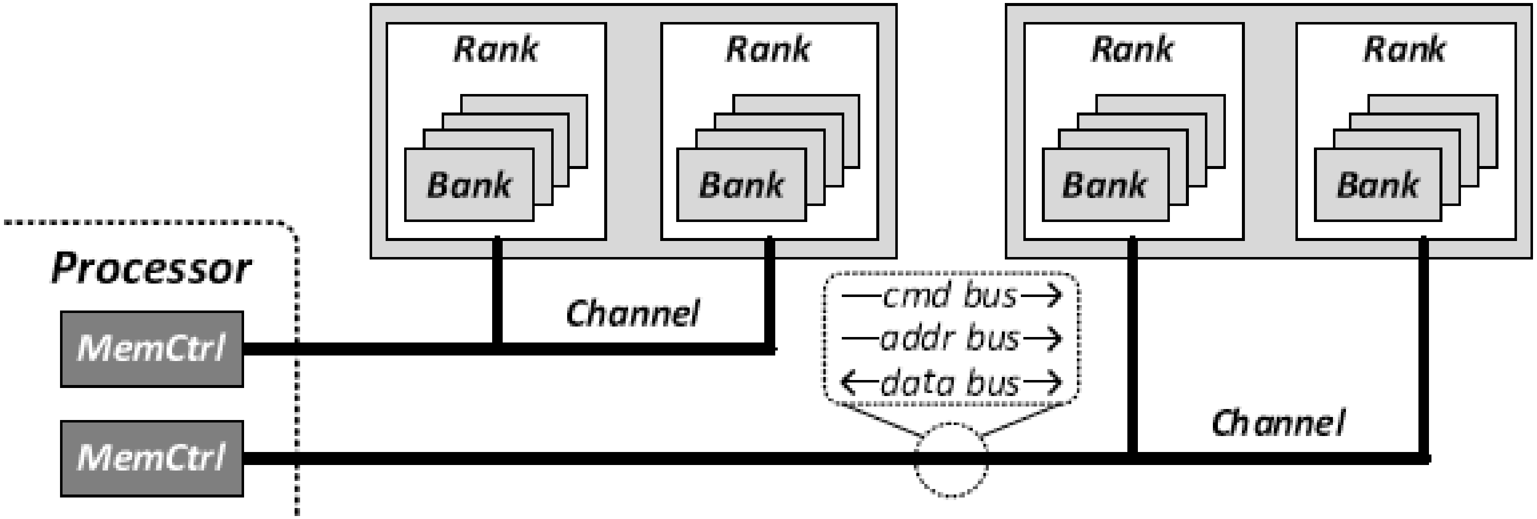
- Channel
- DIMM
- Rank
- Chip
- Bank
- Row/Column
- Cell



Review: Generalized Memory Structure



Review: Generalized Memory Structure

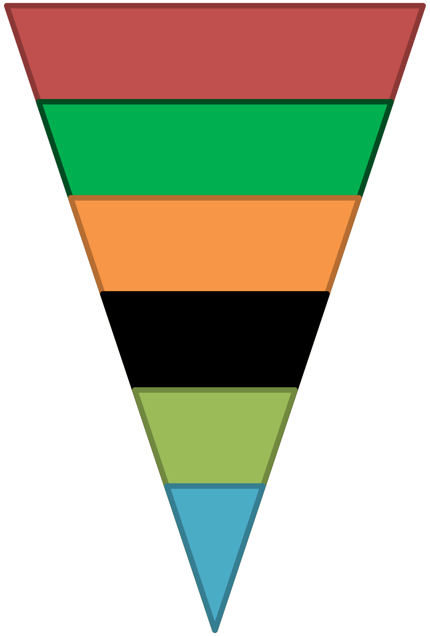


Kim+, "A Case for Exploiting Subarray-Level Parallelism in DRAM," ISCA 2012.

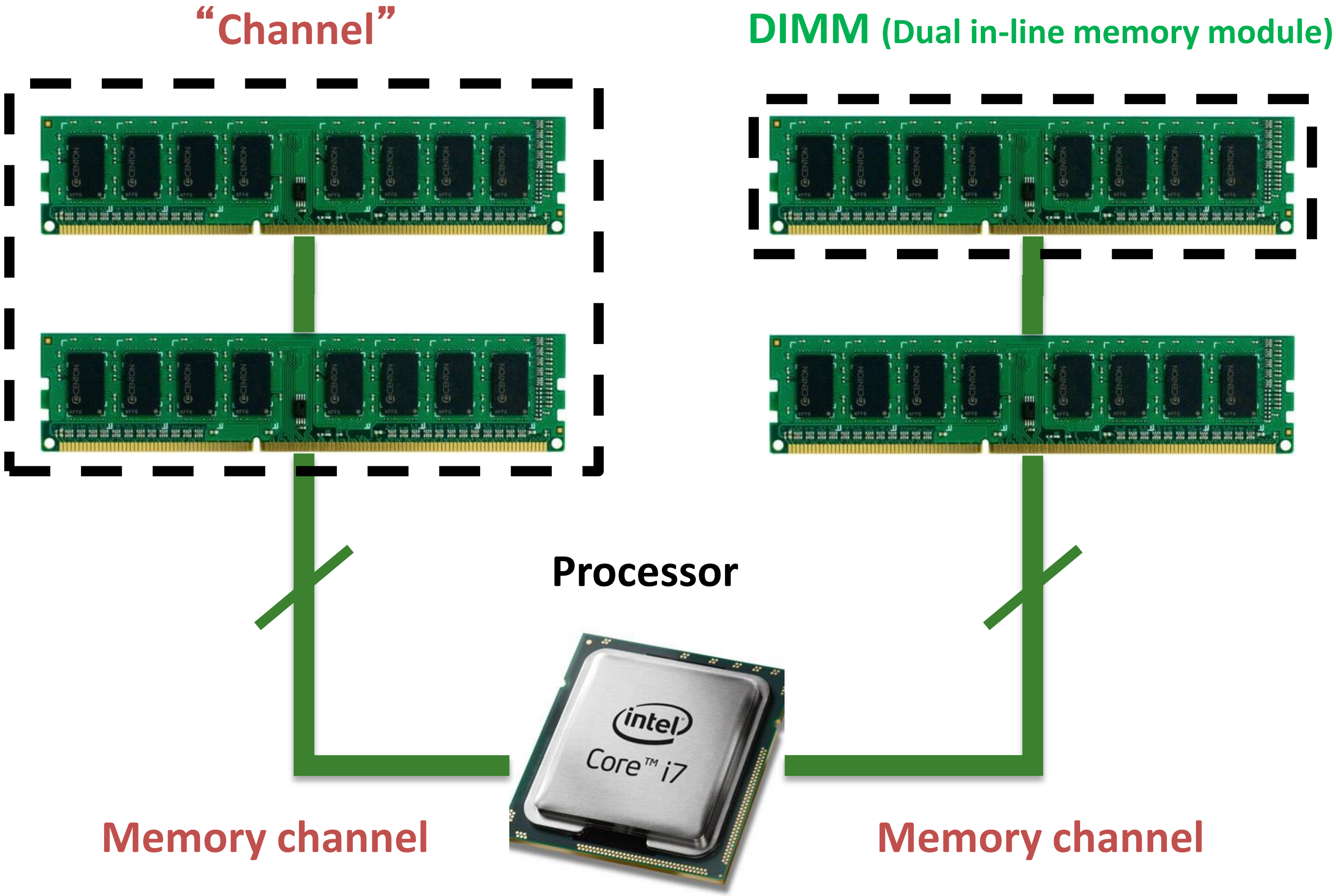
The DRAM Subsystem
The Top Down View

DRAM Subsystem Organization

- Channel
- DIMM
- Rank
- Chip
- Bank
- Row/Column
- Cell



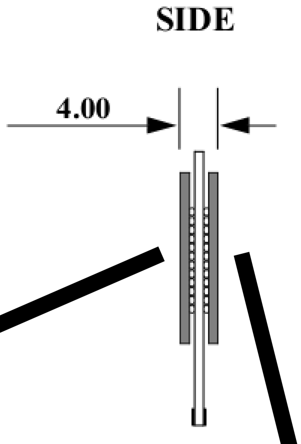
The DRAM subsystem



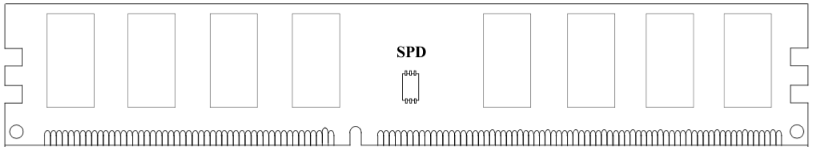
Breaking down a DIMM

DIMM (Dual in-line memory module)

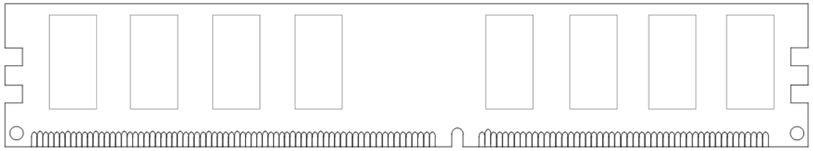
Side view



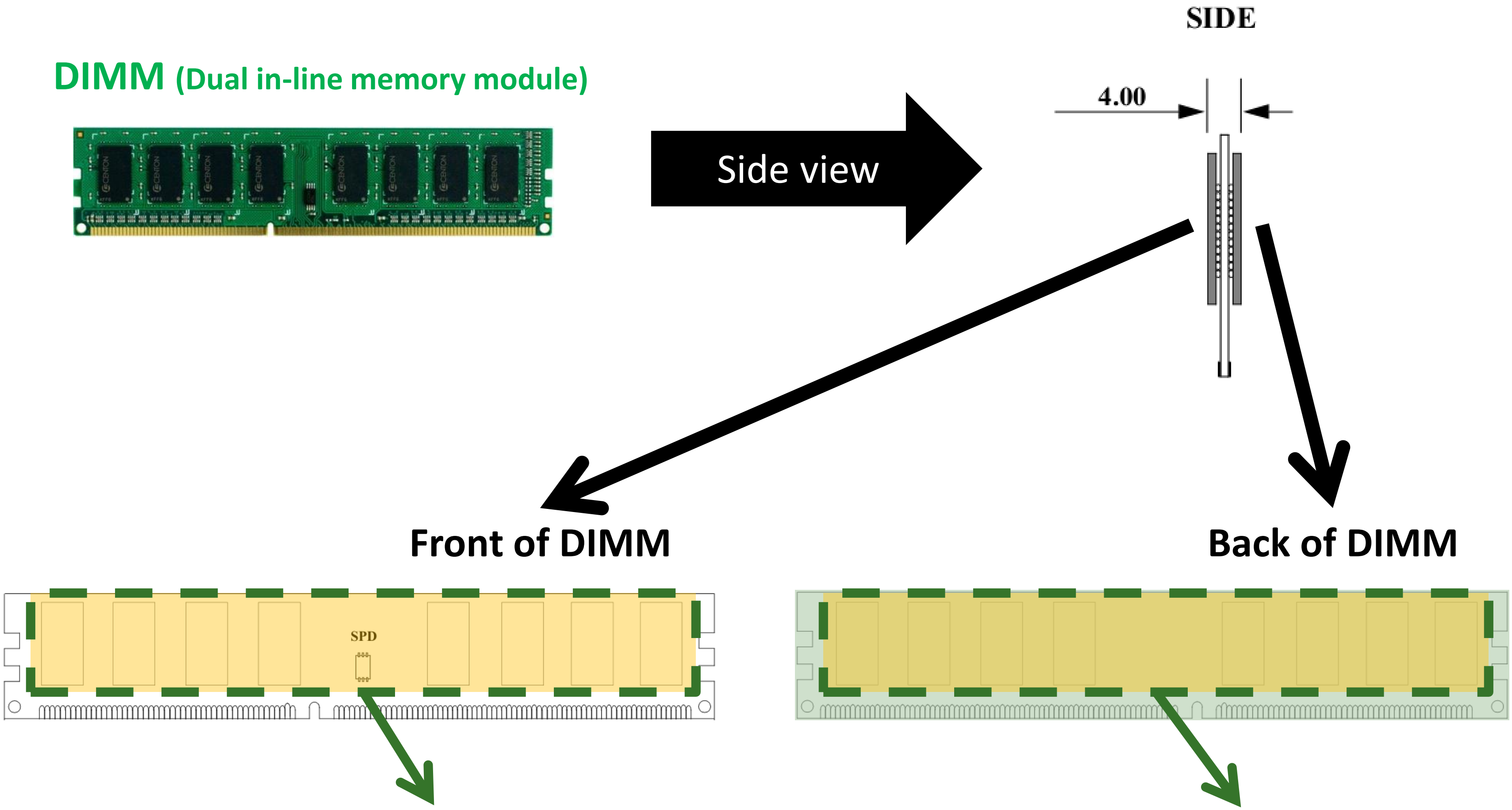
Front of DIMM



Back of DIMM



Breaking down a DIMM



DIMM **(Dual in-line memory module)**

Side view

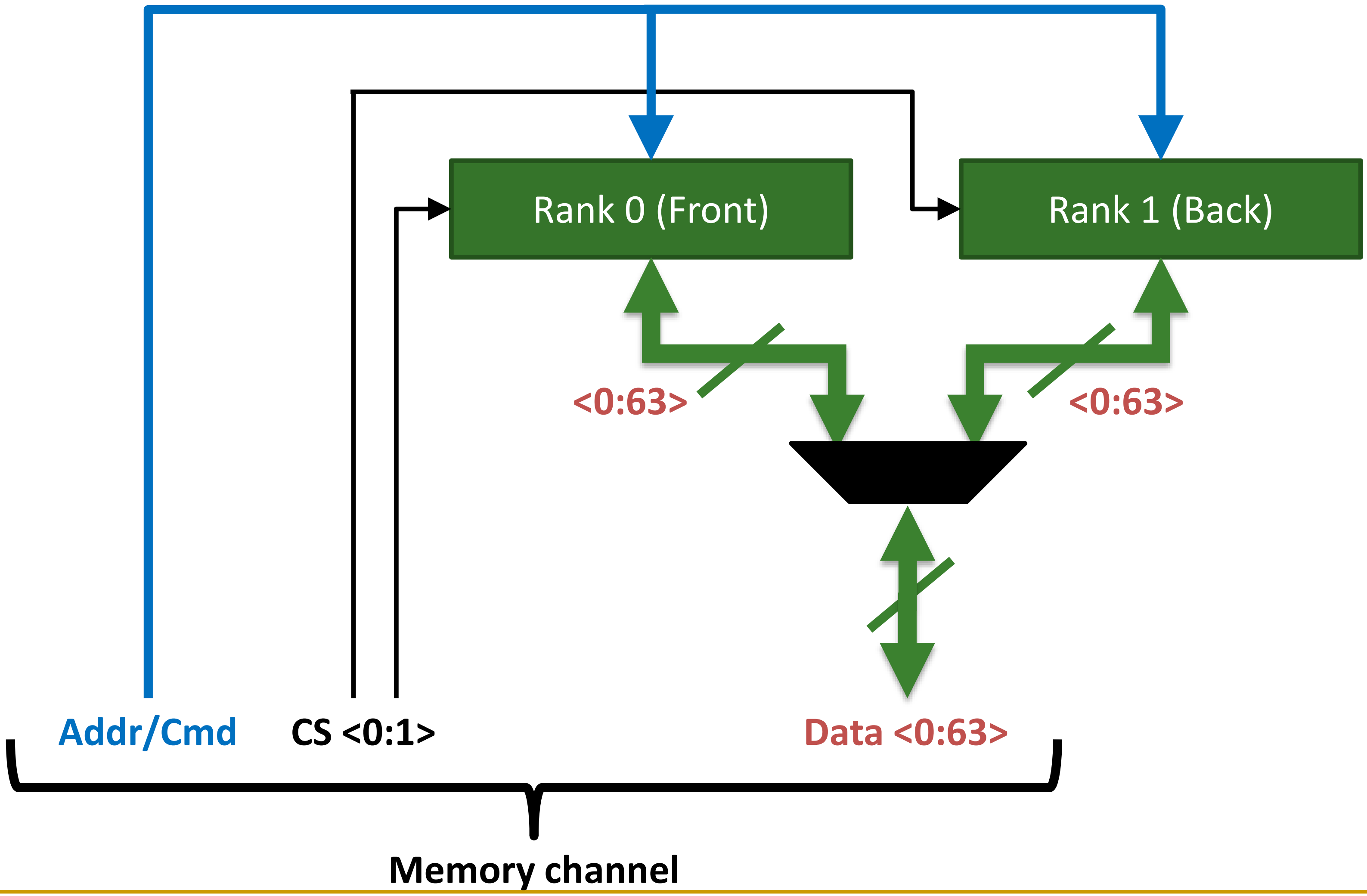
Front of DIMM

Back of DIMM

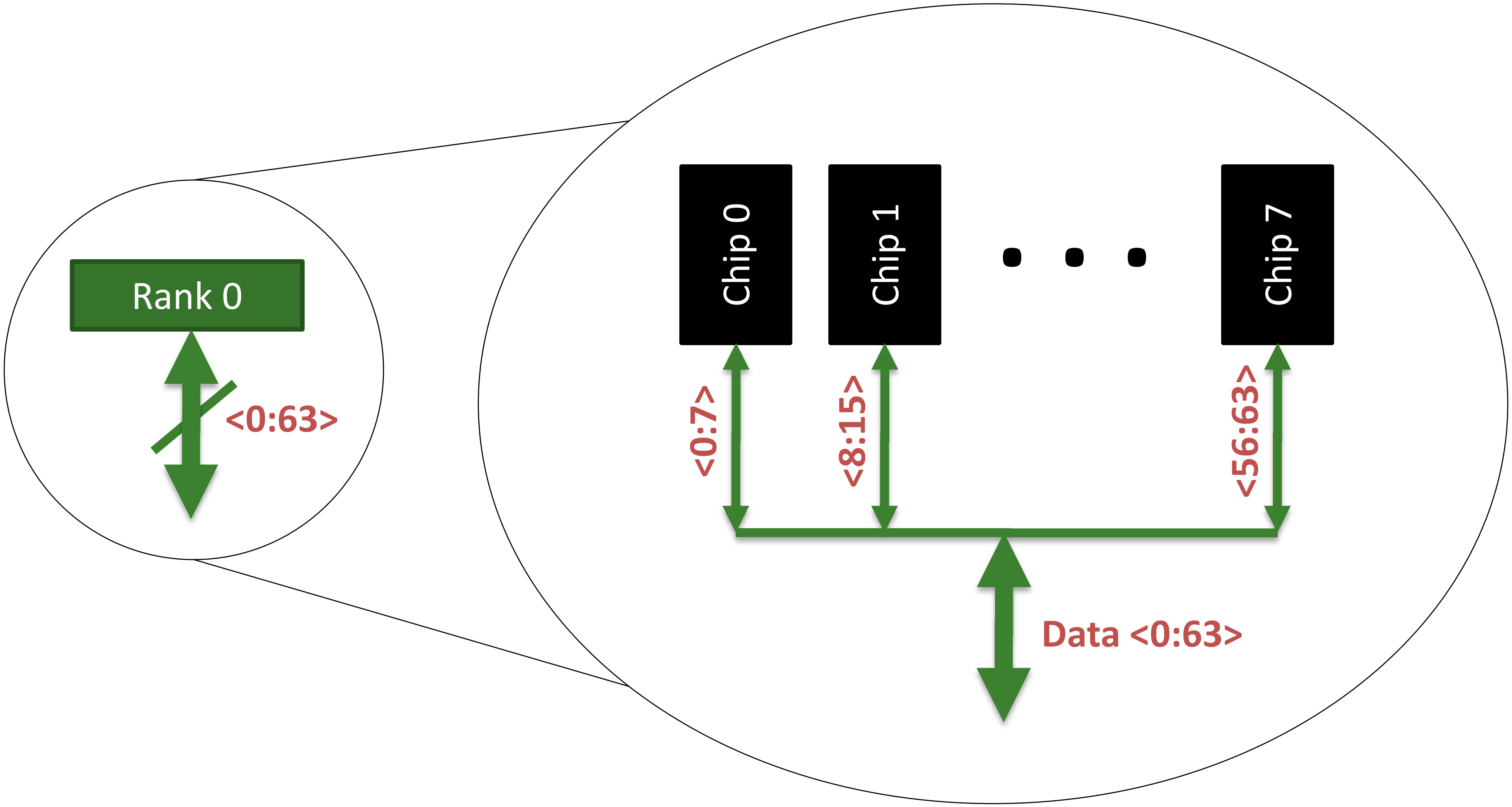
Rank 0: collection of 8 chips

Rank 1

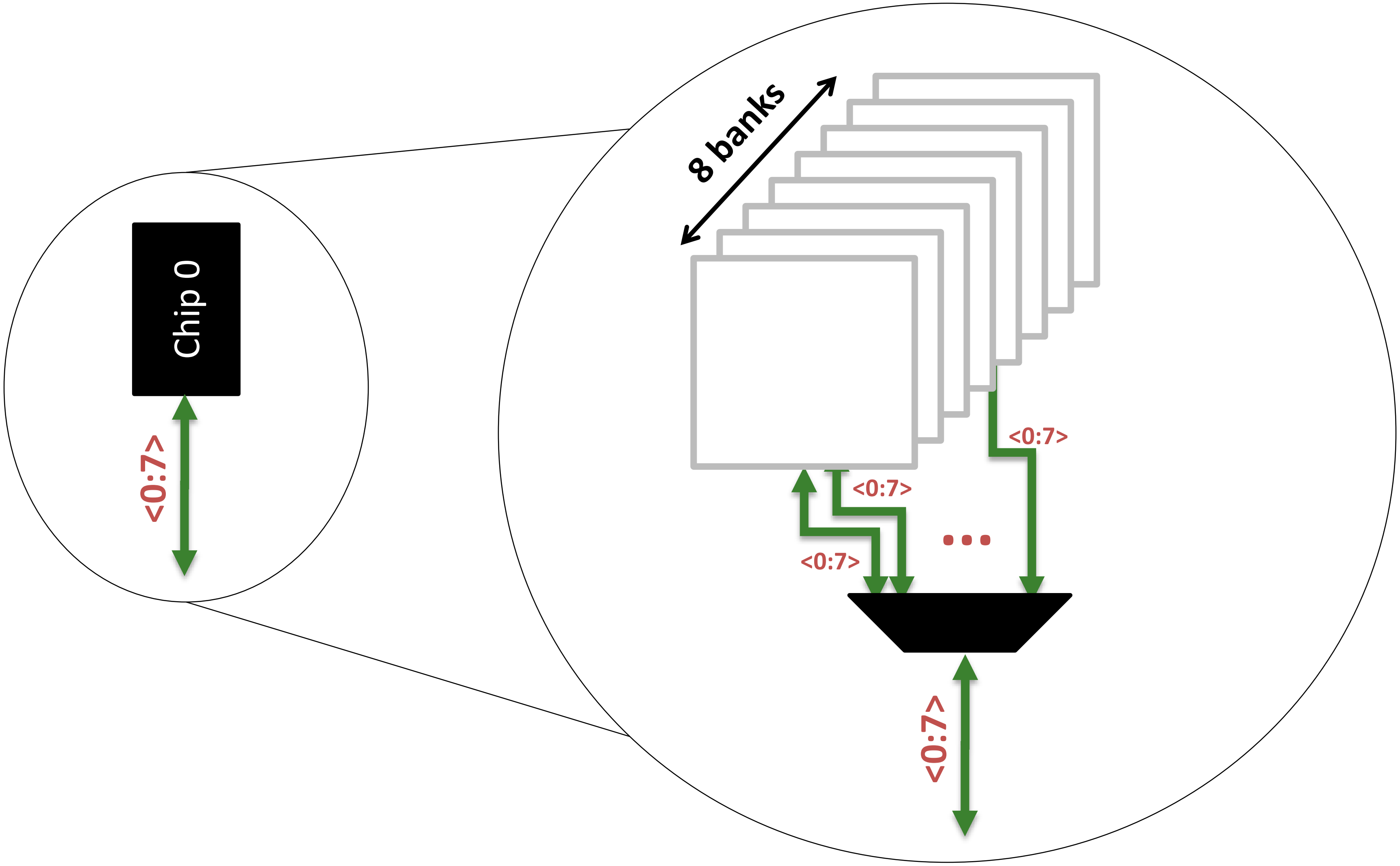
Rank


Rank 0 (Front)
Rank 1 (Back)
<0:63>
<0:63>
Addr/Cmd
CS <0:1>
Data <0:63>
Memory channel

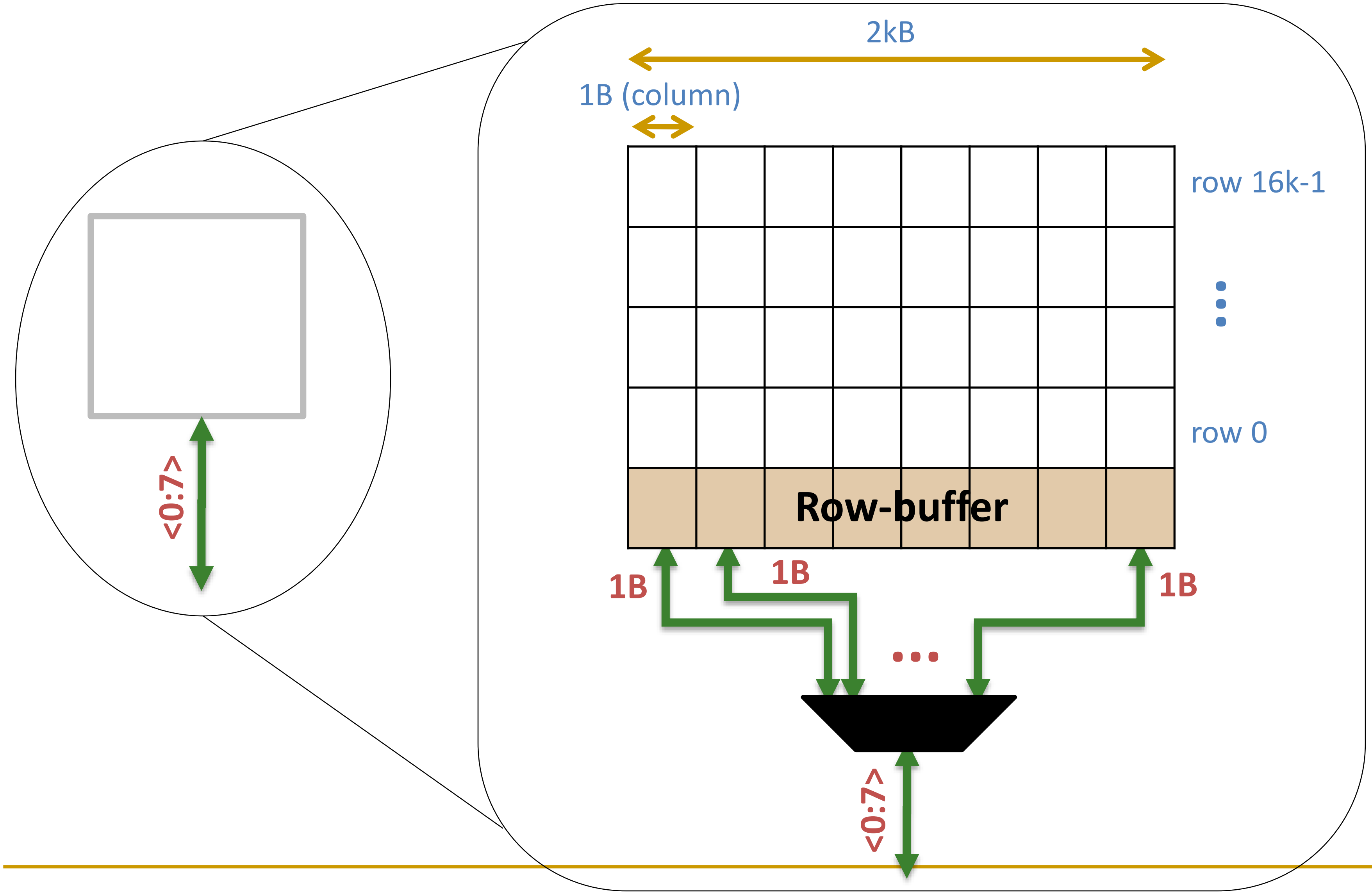
Breaking down a Rank



Breaking down a Chip

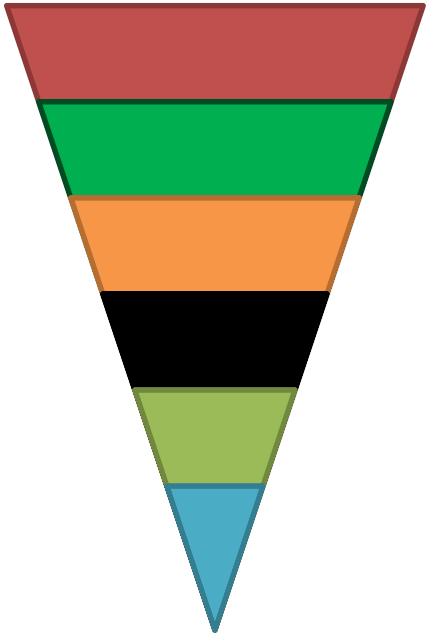


Breaking down a Bank

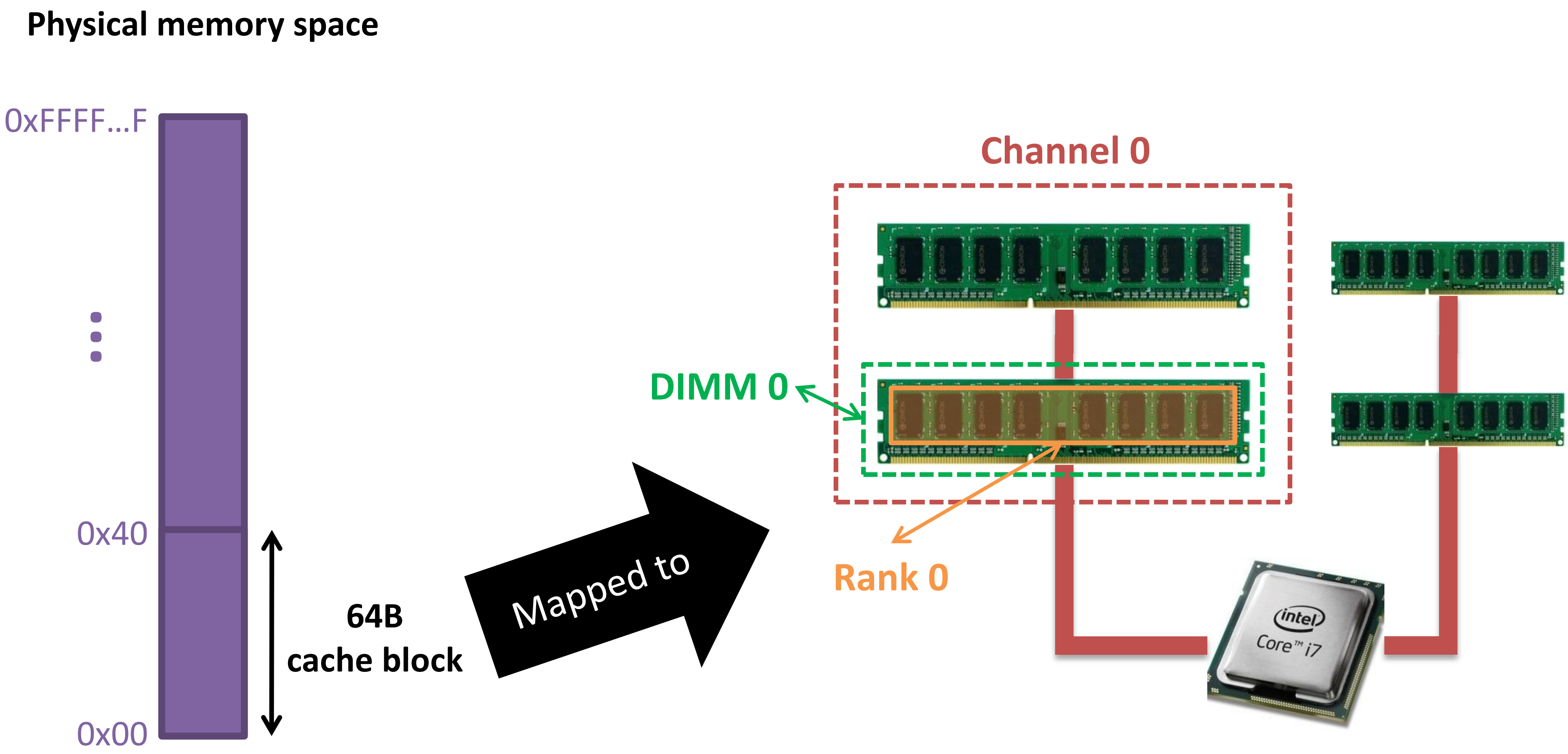


DRAM Subsystem Organization

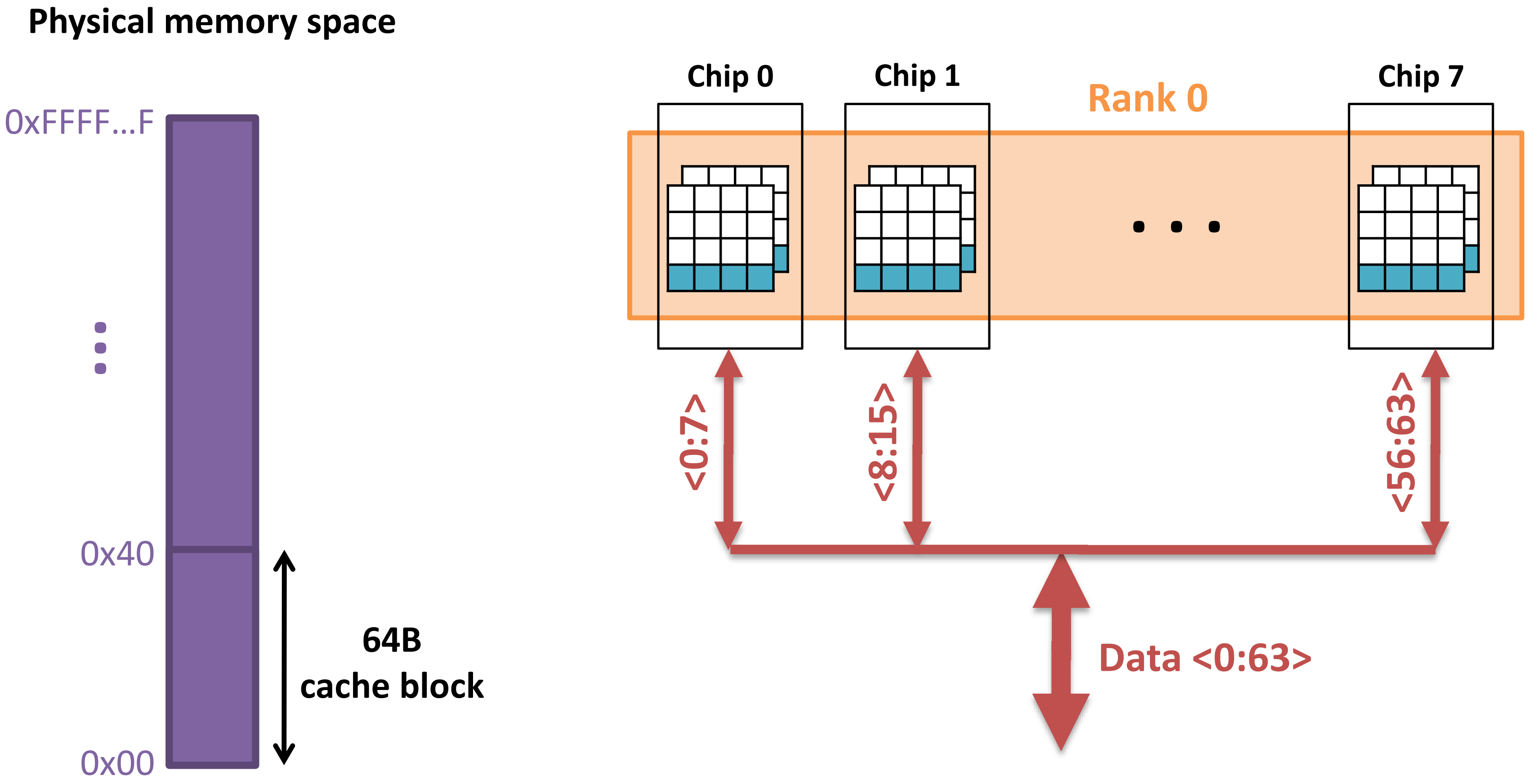
- Channel
- DIMM
- Rank
- Chip
- Bank
- Row/Column
- Cell



Example: Transferring a cache block

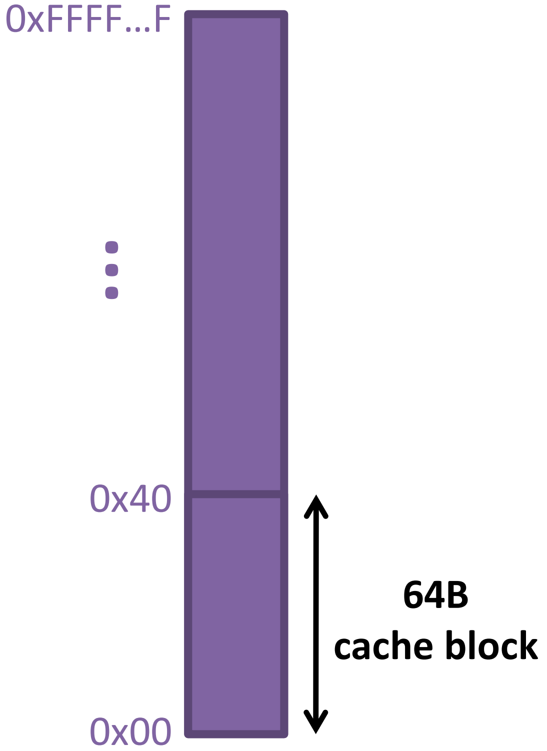


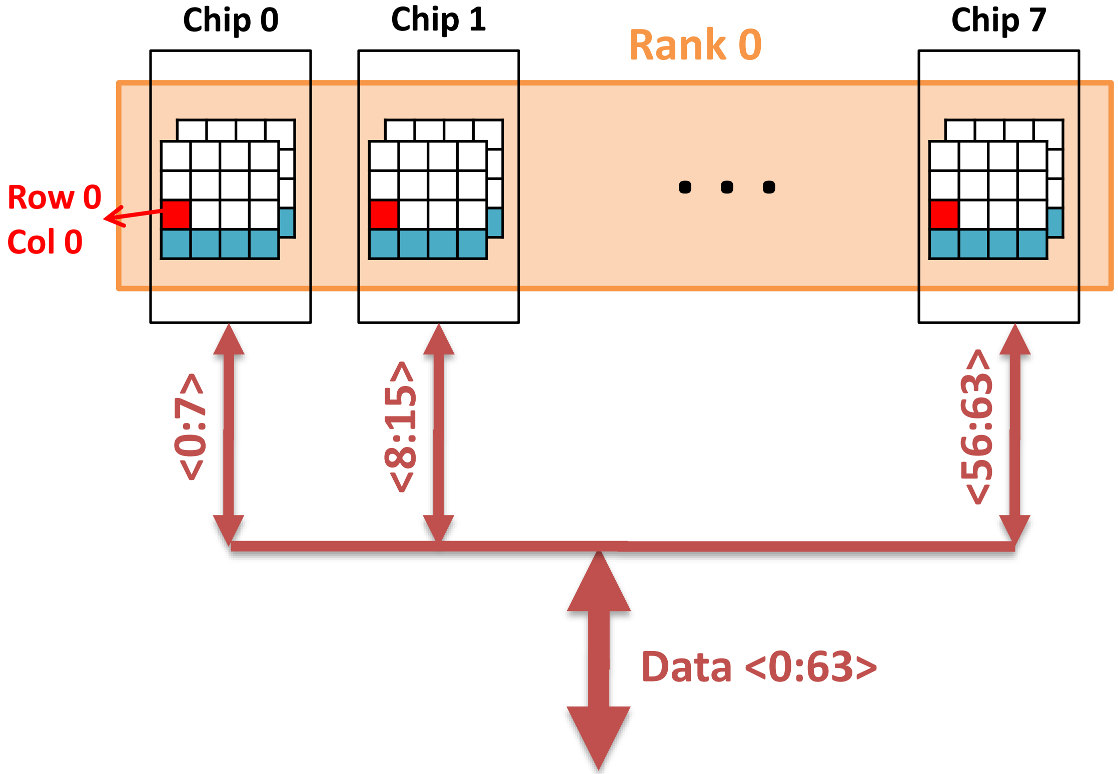
Example: Transferring a cache block



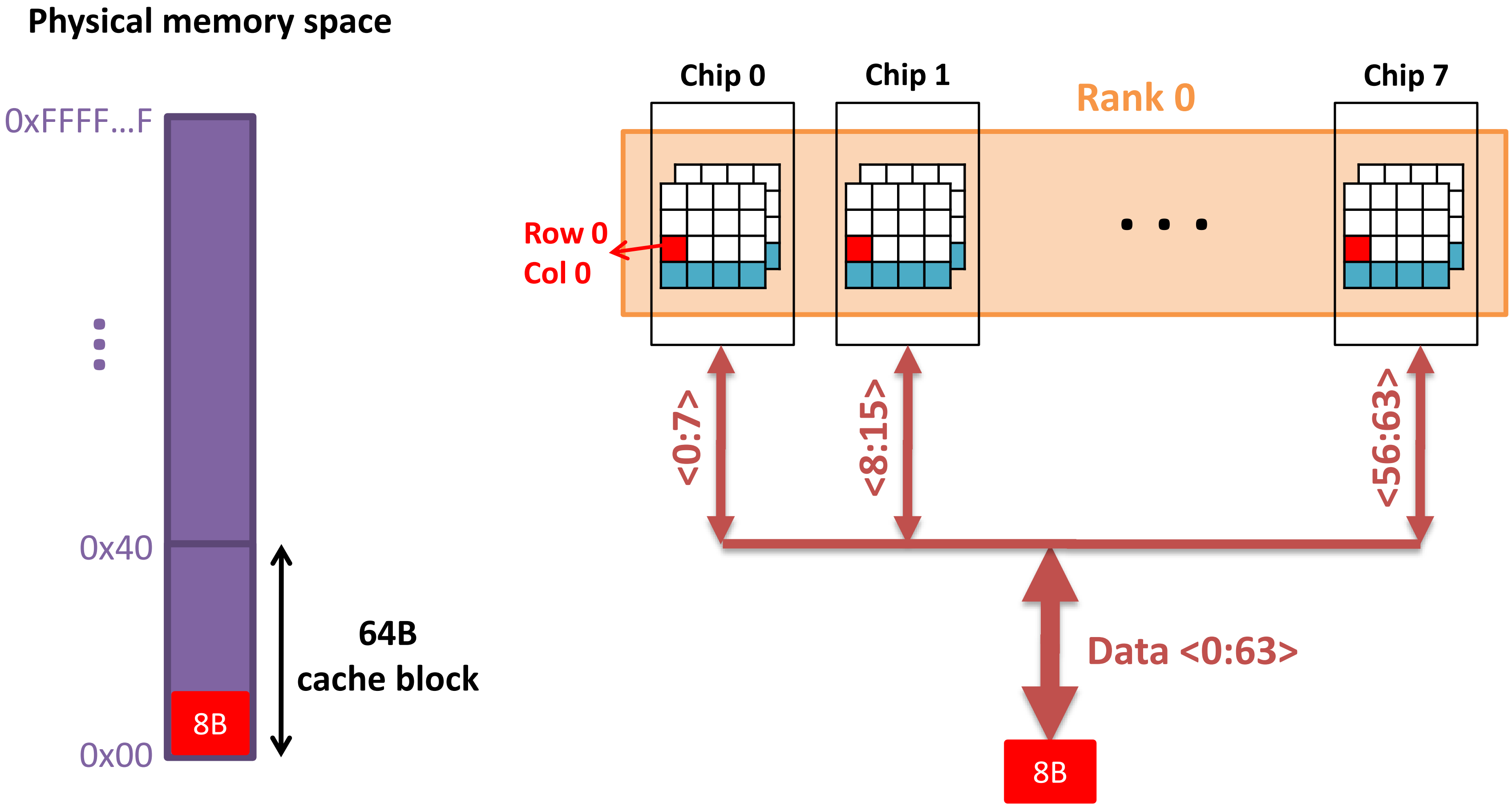
Example: Transferring a cache block

Physical memory space

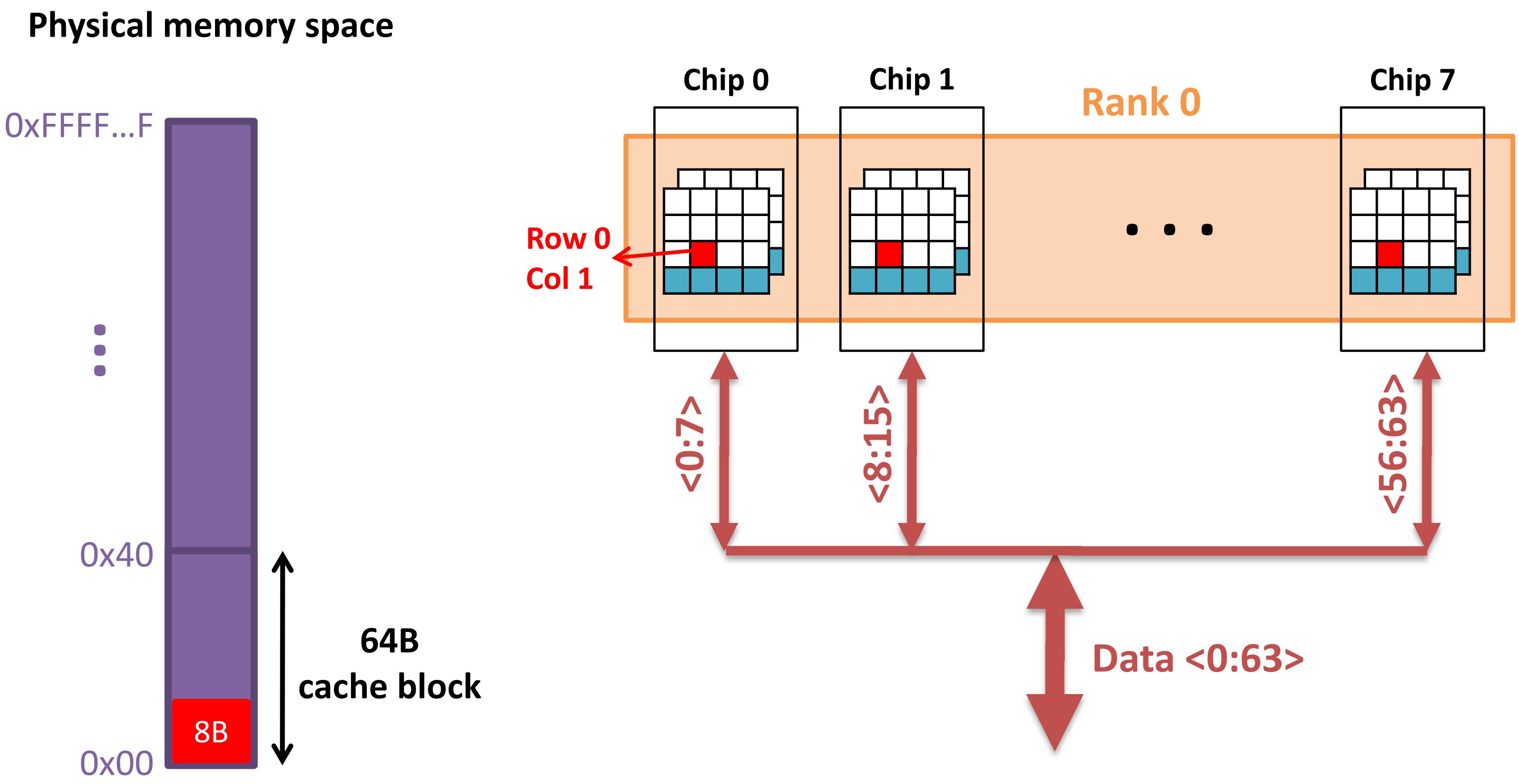




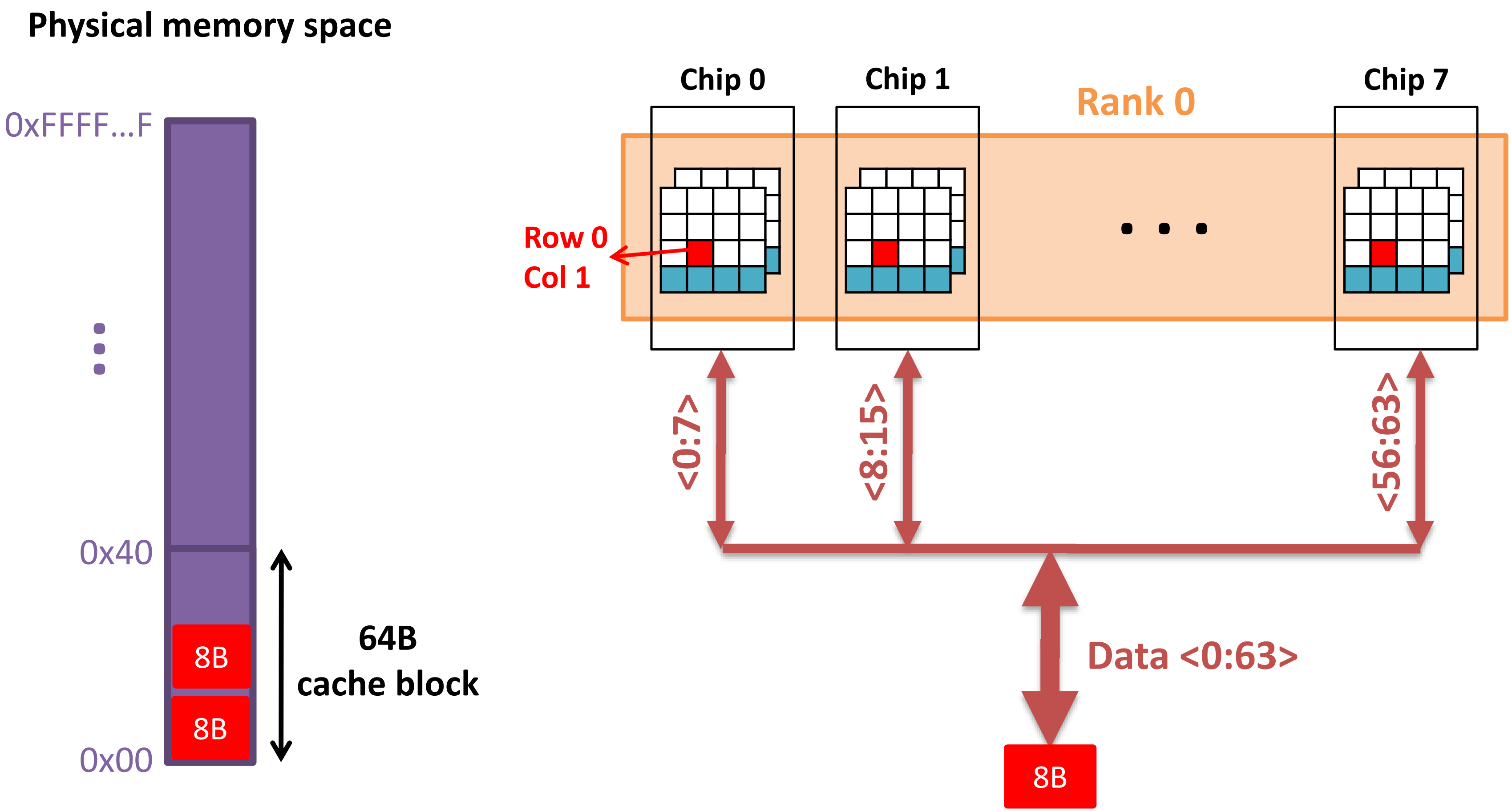
Example: Transferring a cache block



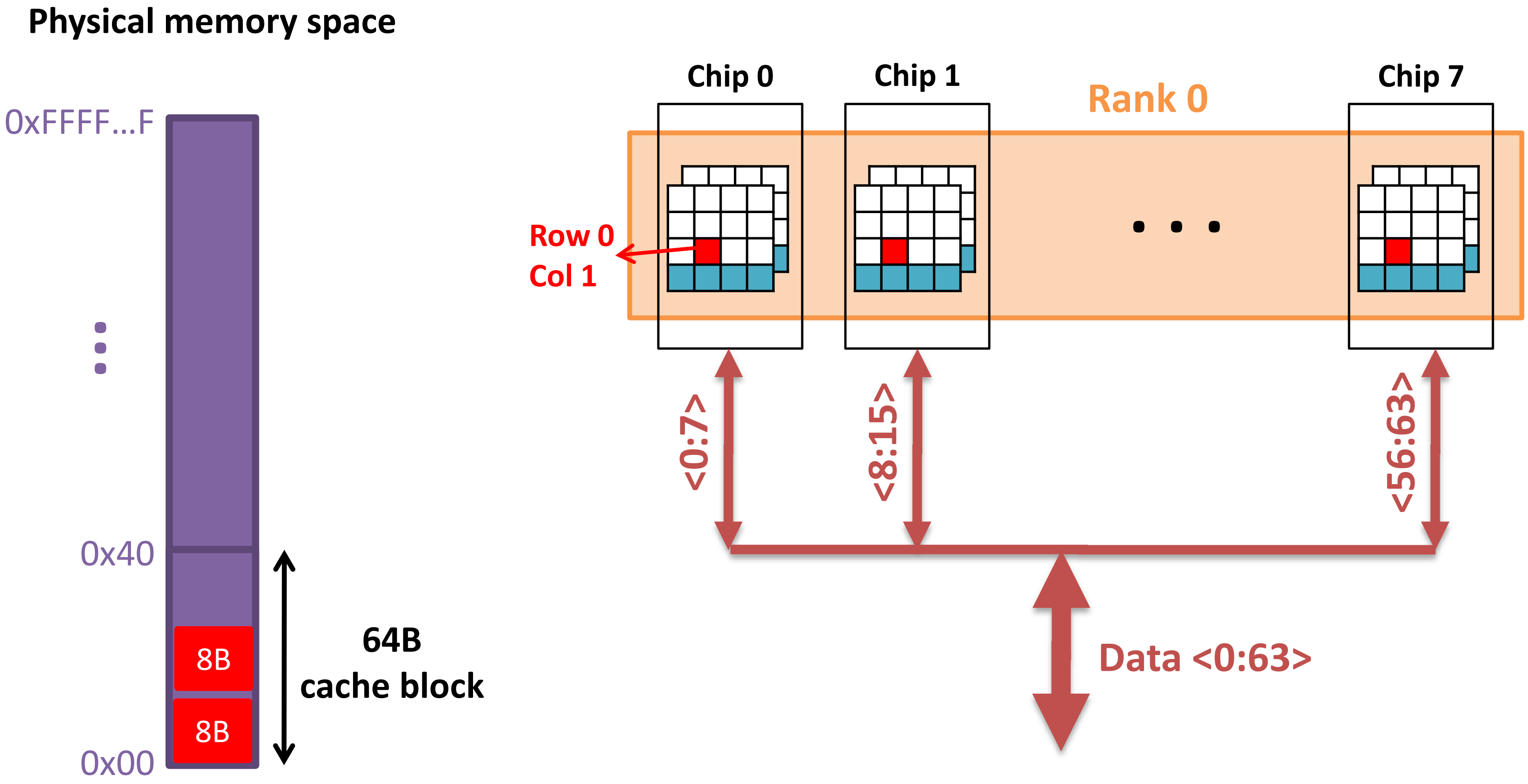
Example: Transferring a cache block



Example: Transferring a cache block



Example: Transferring a cache block



A 64B cache block takes 8 I/O cycles to transfer.

During the process, 8 columns are read sequentially.

Latency Components: Basic DRAM Operation

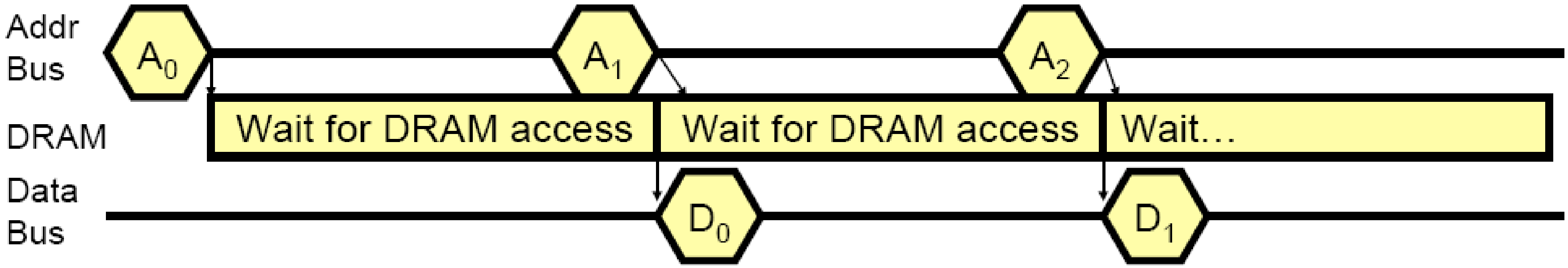
- CPU → controller transfer time
- Controller latency
  - Queuing & scheduling delay at the controller
  - Access converted to basic commands
- Controller → DRAM transfer time
- DRAM bank latency
  - Simple CAS (column address strobe) if row is "open" OR
  - RAS (row address strobe) + CAS if array precharged OR
  - PRE + RAS + CAS (worst case)
- DRAM → Controller transfer time
  - Bus latency (BL)
- Controller to CPU transfer time

Multiple Banks (Interleaving) and Channels

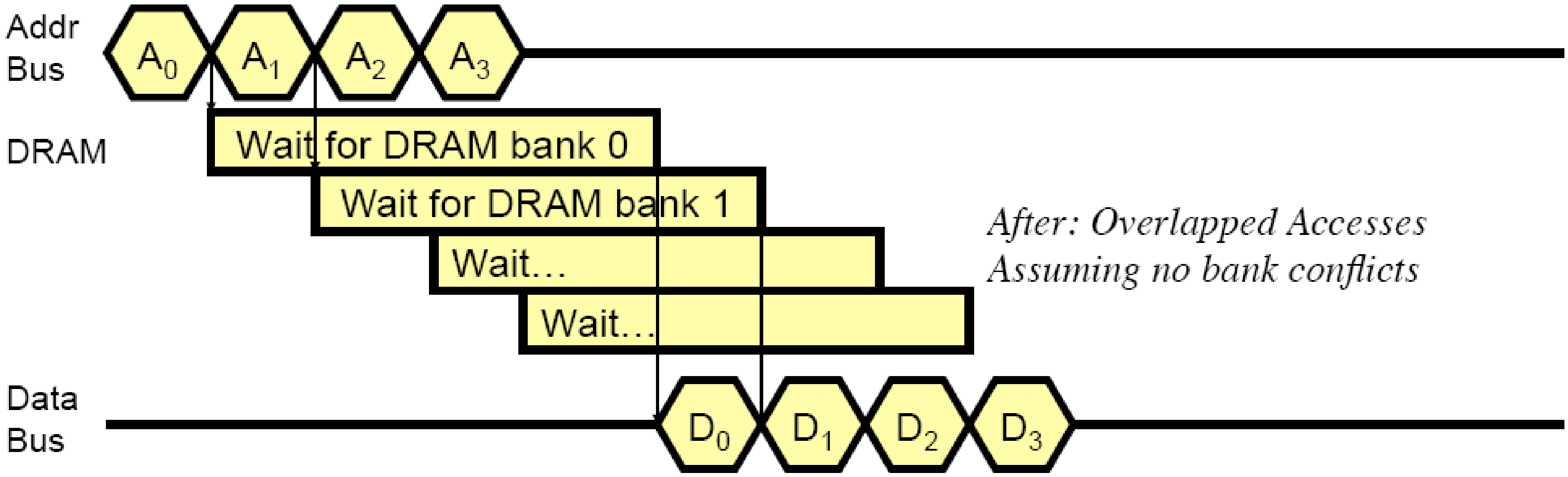
- Multiple banks
  - Enable concurrent DRAM accesses
  - Bits in address determine which bank an address resides in
- Multiple independent channels serve the same purpose
  - But they are even better because they have separate data buses
  - Increased bus bandwidth

- Enabling more concurrency requires reducing
  - Bank conflicts
  - Channel conflicts
- How to select/randomize bank/channel indices in address?
  - Lower order bits have more entropy
  - Randomizing hash functions (XOR of different address bits)

How Multiple Banks Help



Before: No Overlapping
Assuming accesses to different DRAM rows



After: Overlapped Accesses
Assuming no bank conflicts

Address Mapping (Single Channel)

- Single-channel system with 8-byte memory bus
  - 2GB memory, 8 banks, 16K rows & 2K columns per bank

- Row interleaving
  - Consecutive rows of memory in consecutive banks

Row (14 bits)	Bank (3 bits)	Column (11 bits)	Byte in bus (3 bits)

  - Accesses to consecutive cache blocks serviced in a pipelined manner

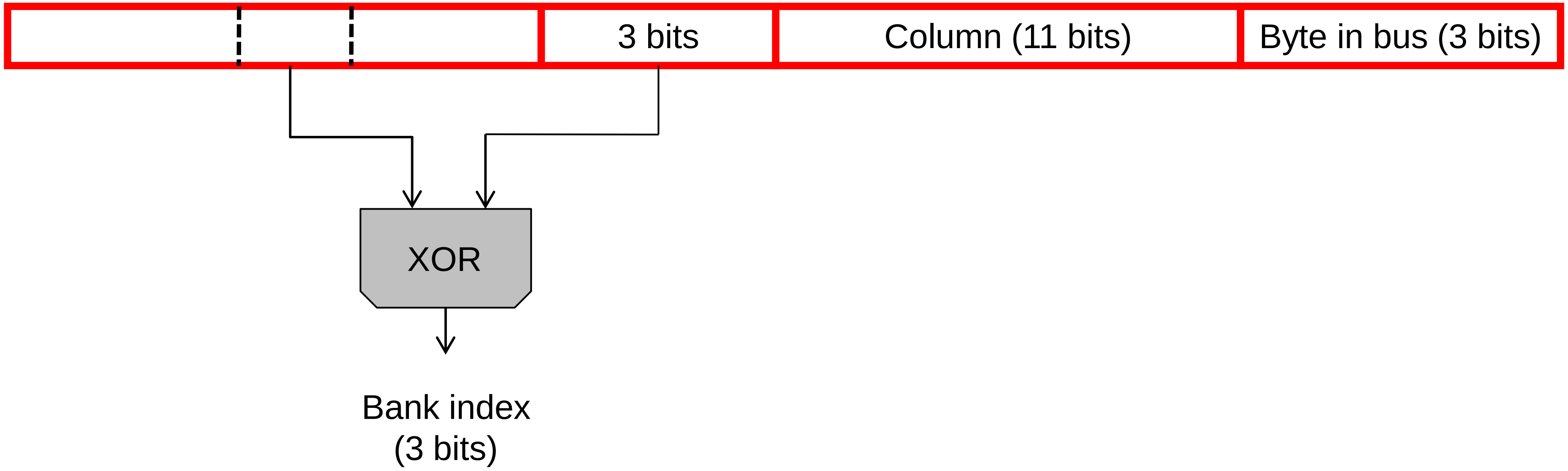
- Cache block interleaving
  - Consecutive cache block addresses in consecutive banks
  - 64 byte cache blocks

Row (14 bits)	High Column	Bank (3 bits)	Low Col.	Byte in bus (3 bits)
	8 bits		3 bits	

  - Accesses to consecutive cache blocks can be serviced in parallel

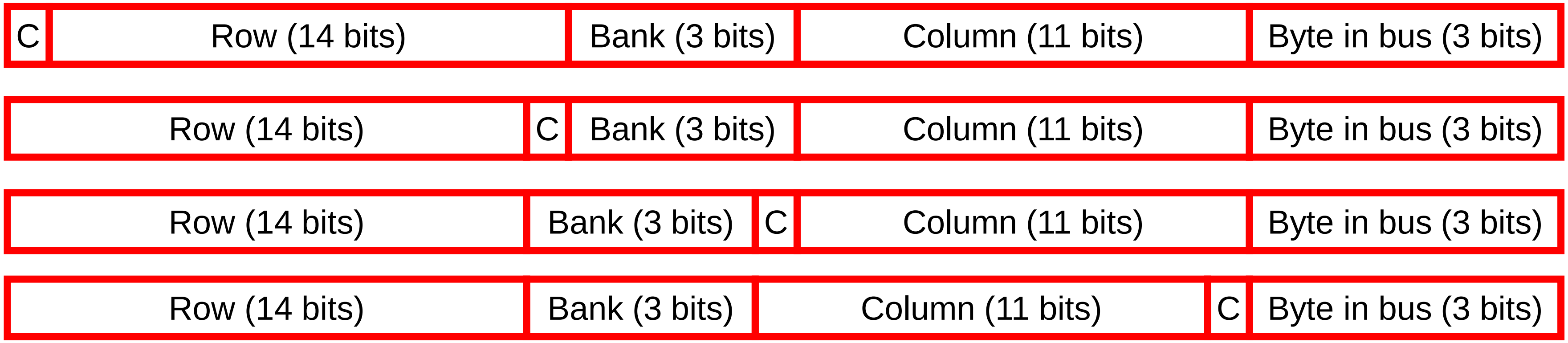
Bank Mapping Randomization

- DRAM controller can randomize the address mapping to banks so that bank conflicts are less likely



- Reading:
  - Rau, "Pseudo-randomly Interleaved Memory," ISCA 1991.

Address Mapping (Multiple Channels)

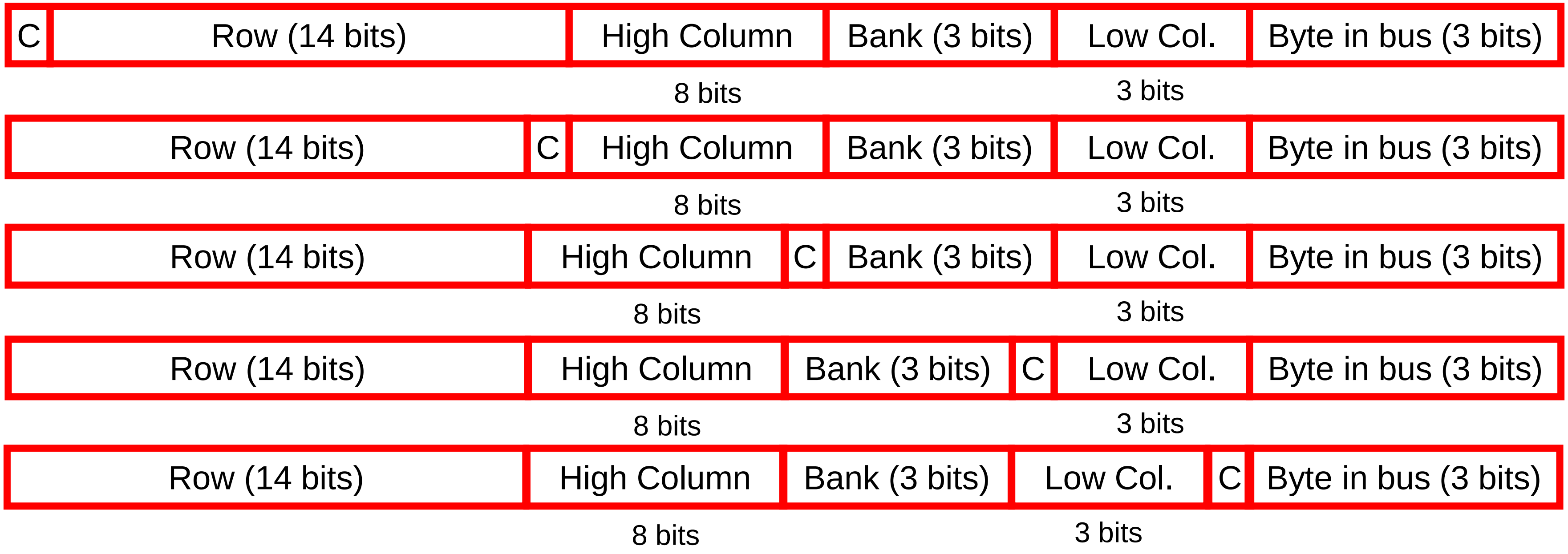

C Row (14 bits) Bank (3 bits) Column (11 bits) Byte in bus (3 bits)

Row (14 bits) C Bank (3 bits) Column (11 bits) Byte in bus (3 bits)

Row (14 bits) Bank (3 bits) C Column (11 bits) Byte in bus (3 bits)

Row (14 bits) Bank (3 bits) Column (11 bits) C Byte in bus (3 bits)

- Where are consecutive cache blocks?


C Row (14 bits) High Column Bank (3 bits) Low Col. Byte in bus (3 bits)

8 bits 3 bits

Row (14 bits) C High Column Bank (3 bits) Low Col. Byte in bus (3 bits)

8 bits 3 bits

Row (14 bits) High Column C Bank (3 bits) Low Col. Byte in bus (3 bits)

8 bits 3 bits

Row (14 bits) High Column Bank (3 bits) C Low Col. Byte in bus (3 bits)

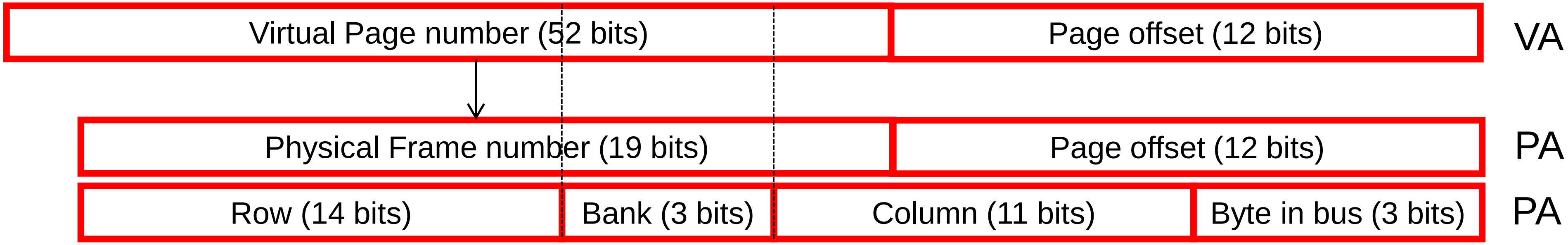
8 bits 3 bits

Row (14 bits) High Column Bank (3 bits) Low Col. C Byte in bus (3 bits)

8 bits 3 bits

Interaction with Virtual→Physical Mapping

- Operating System influences where an address maps to in DRAM



Virtual Page number (52 bits) | Page offset (12 bits) | VA

Physical Frame number (19 bits) | Page offset (12 bits) | PA

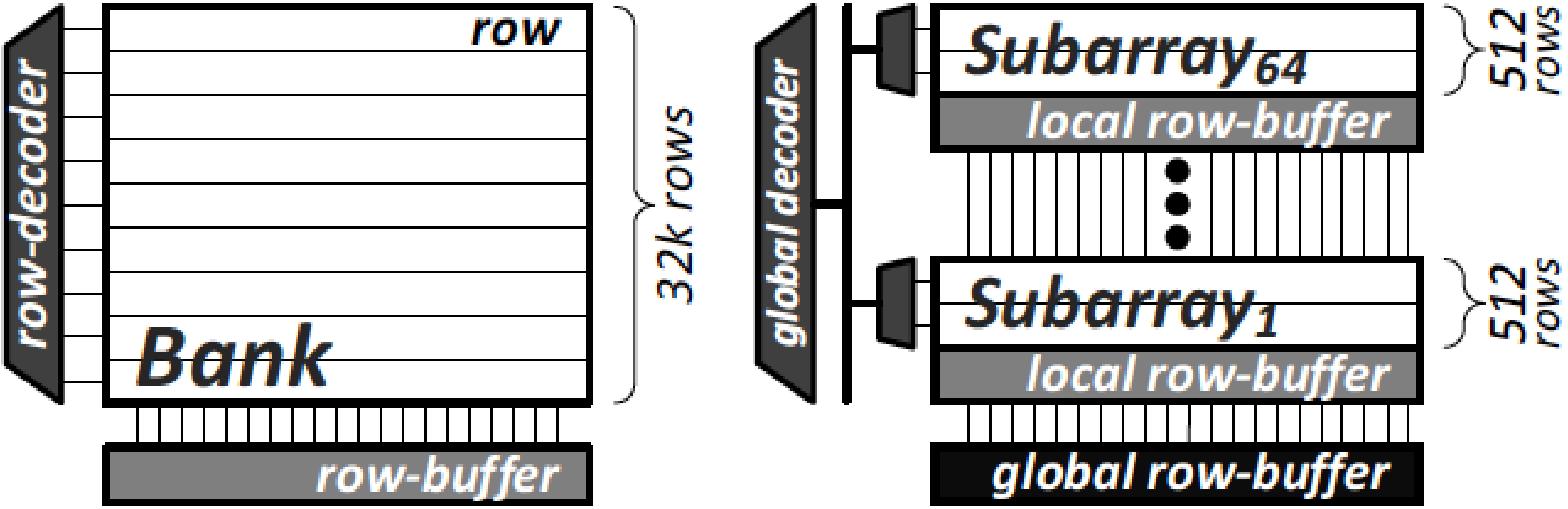
Row (14 bits) | Bank (3 bits) | Column (11 bits) | Byte in bus (3 bits) | PA

- Operating system can influence which bank/channel/rank a virtual page is mapped to.

- It can perform page coloring to
  - Minimize bank conflicts
  - Minimize inter-application interference **[Muralidhara+ MICRO'11]**
  - Minimize latency in the network **[Das+ HPCA'13]**

More on Reducing Bank Conflicts

- Read Sections 1 through 4 of:
  - Kim et al., "A Case for Exploiting Subarray-Level Parallelism in DRAM," ISCA 2012.



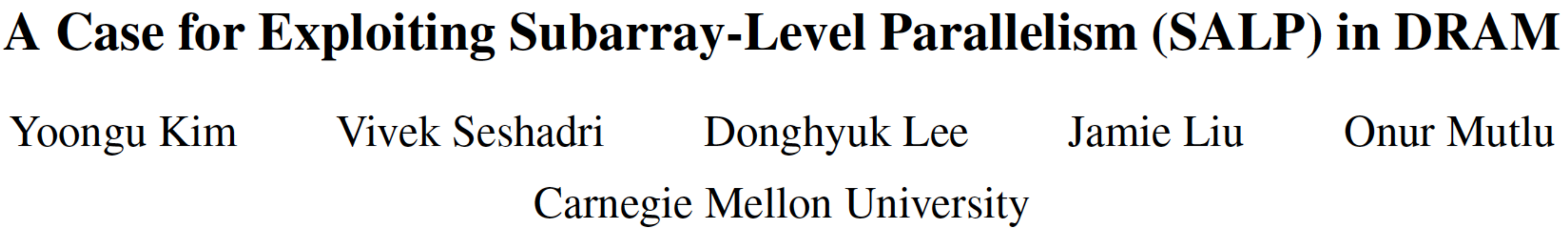
(a) Logical abstraction (b) Physical implementation

Figure 1. DRAM bank organization

Required Reading on DRAM

- Yoongu Kim, Vivek Seshadri, Donghyuk Lee, Jamie Liu, and Onur Mutlu, **"A Case for Exploiting Subarray-Level Parallelism (SALP) in DRAM"**
  *Proceedings of the 39th International Symposium on Computer Architecture (**ISCA**)*, Portland, OR, June 2012. Slides (pptx)

- Sections 1-2 are required

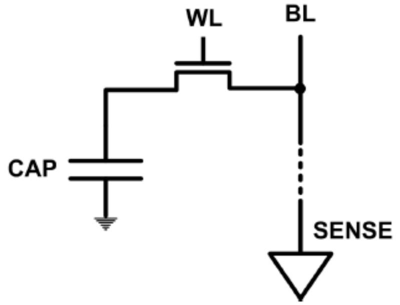


A Case for Exploiting Subarray-Level Parallelism (SALP) in DRAM

Yoongu Kim Vivek Seshadri Donghyuk Lee Jamie Liu Onur Mutlu
Carnegie Mellon University

DRAM Refresh (I)

- DRAM capacitor charge leaks over time
- The memory controller needs to read each row periodically to restore the charge
  - Activate + precharge each row every N ms
  - Typical N = 64 ms



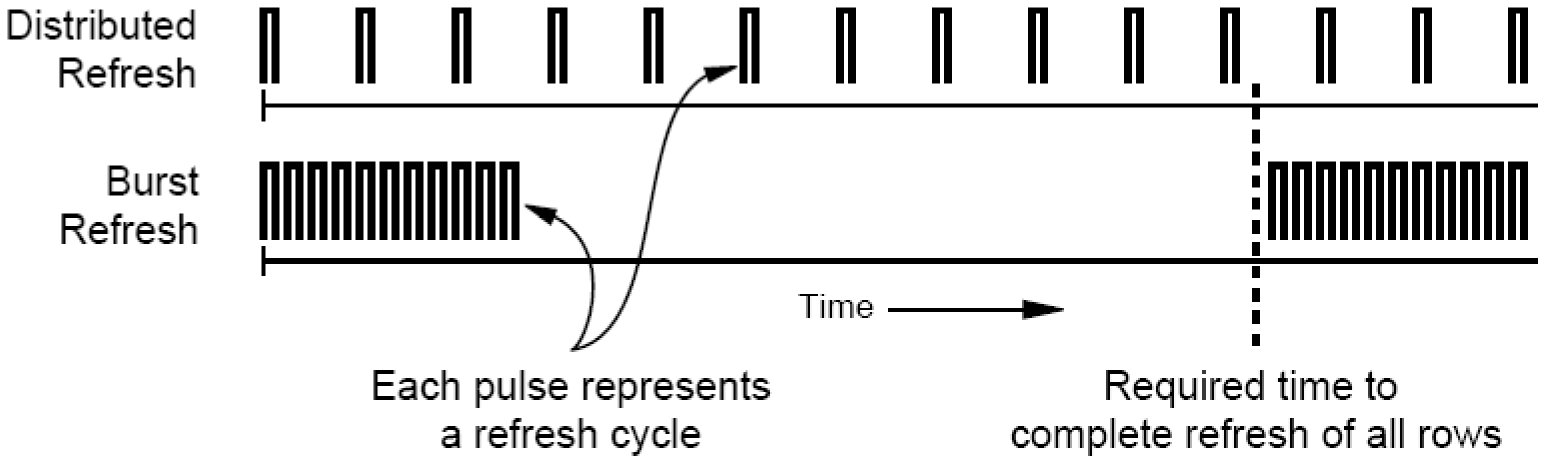
- Implications on performance?

-- DRAM bank unavailable while refreshed

-- Long pause times: If we refresh all rows in burst, every 64ms the DRAM will be unavailable until refresh ends

- Burst refresh: All rows refreshed immediately after one another
- Distributed refresh: Each row refreshed at a different time, at regular intervals

DRAM Refresh (II)


Distributed Refresh
Burst Refresh
Time
Each pulse represents a refresh cycle
Required time to complete refresh of all rows

- Distributed refresh eliminates long pause times
- How else we can reduce the effect of refresh on performance?
  - Can we reduce the number of refreshes?

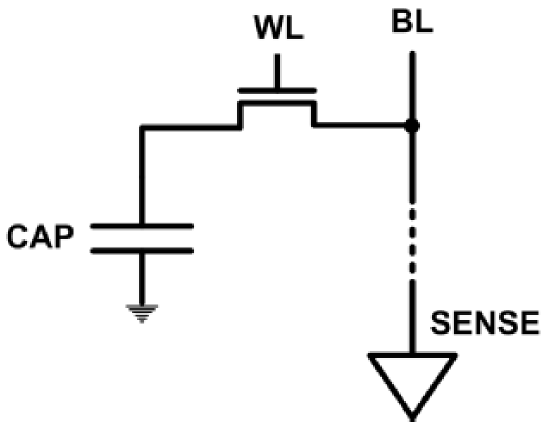
Downsides of DRAM Refresh

-- Energy consumption: Each refresh consumes energy

-- Performance degradation: DRAM rank/bank unavailable while refreshed

-- QoS/predictability impact: (Long) pause times during refresh

-- Refresh rate limits DRAM density scaling



Liu et al., "RAIDR: Retention-aware Intelligent DRAM Refresh," ISCA 2012.

Memory Controllers

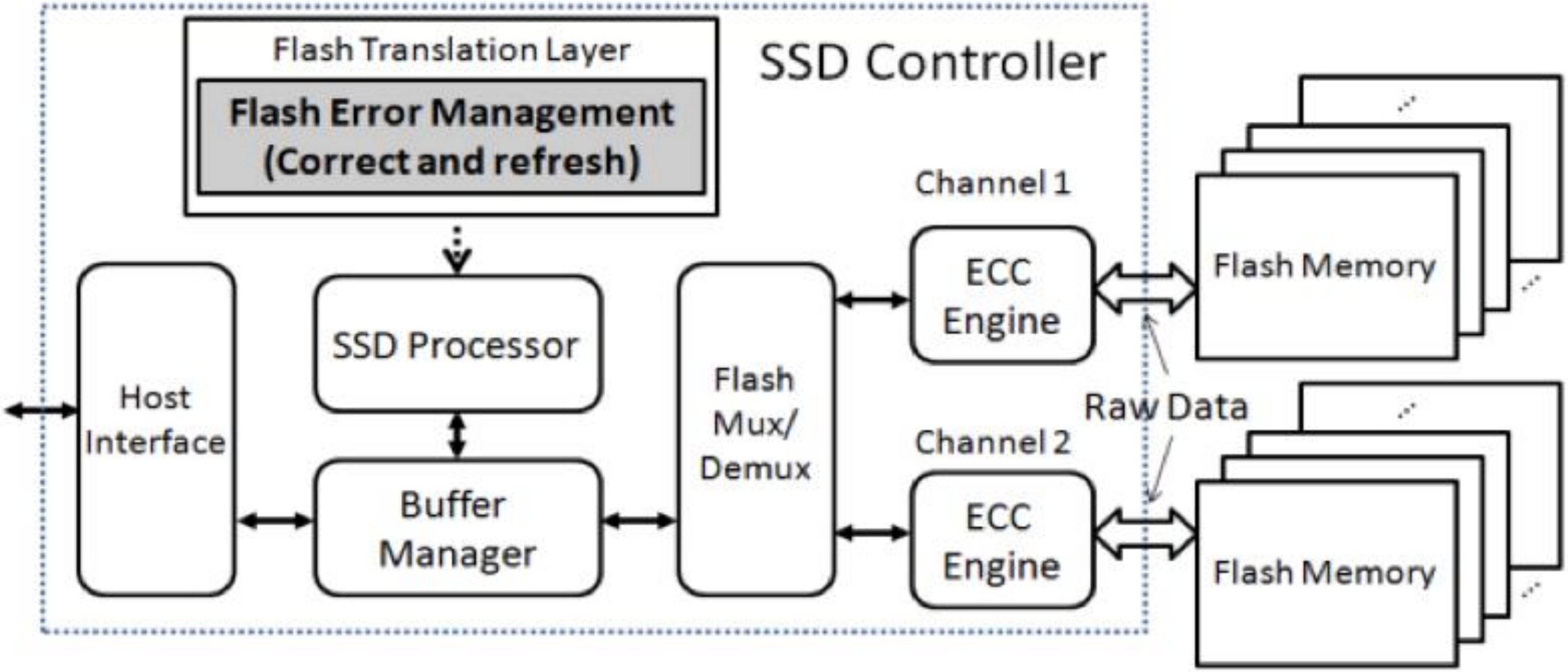
DRAM versus Other Types of Memories

- 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, …



Cai+, "Flash Correct-and-Refresh: Retention-Aware Error Management for Increased Flash Memory Lifetime", ICCD 2012.

Another View of the SSD Controller

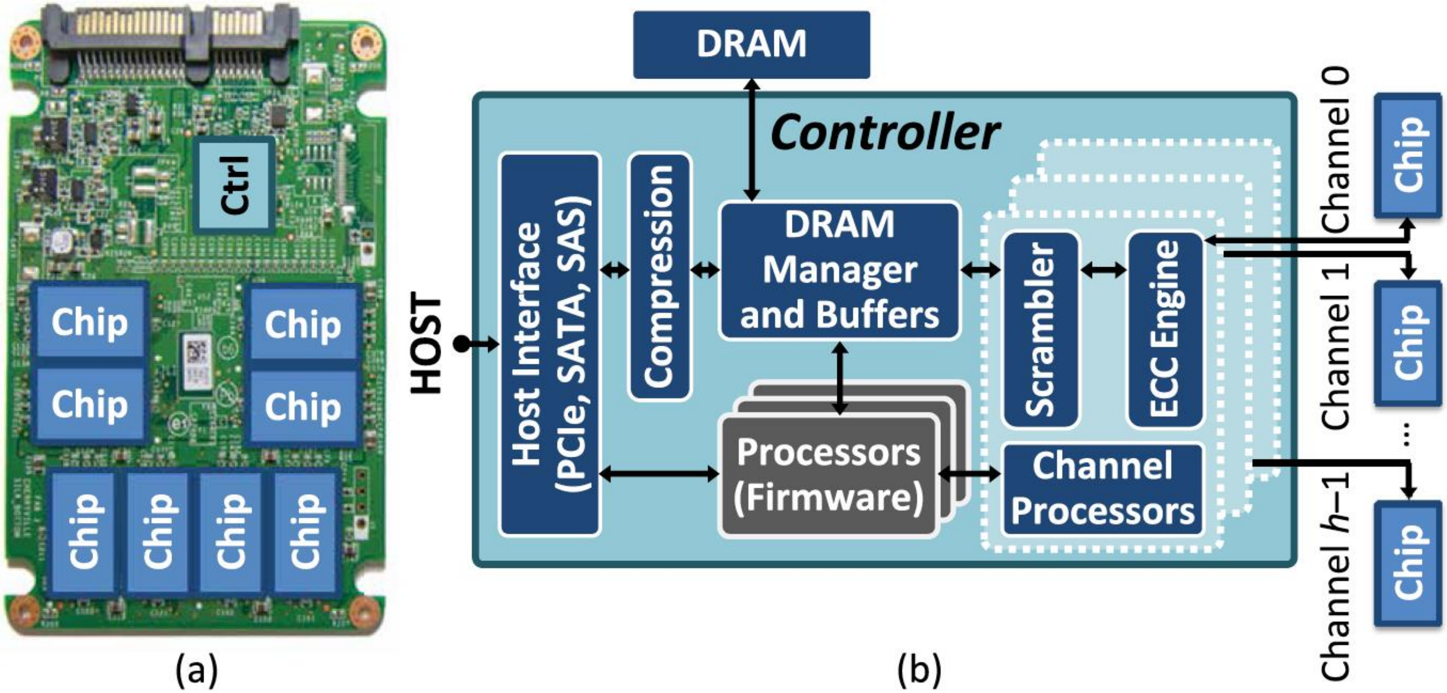


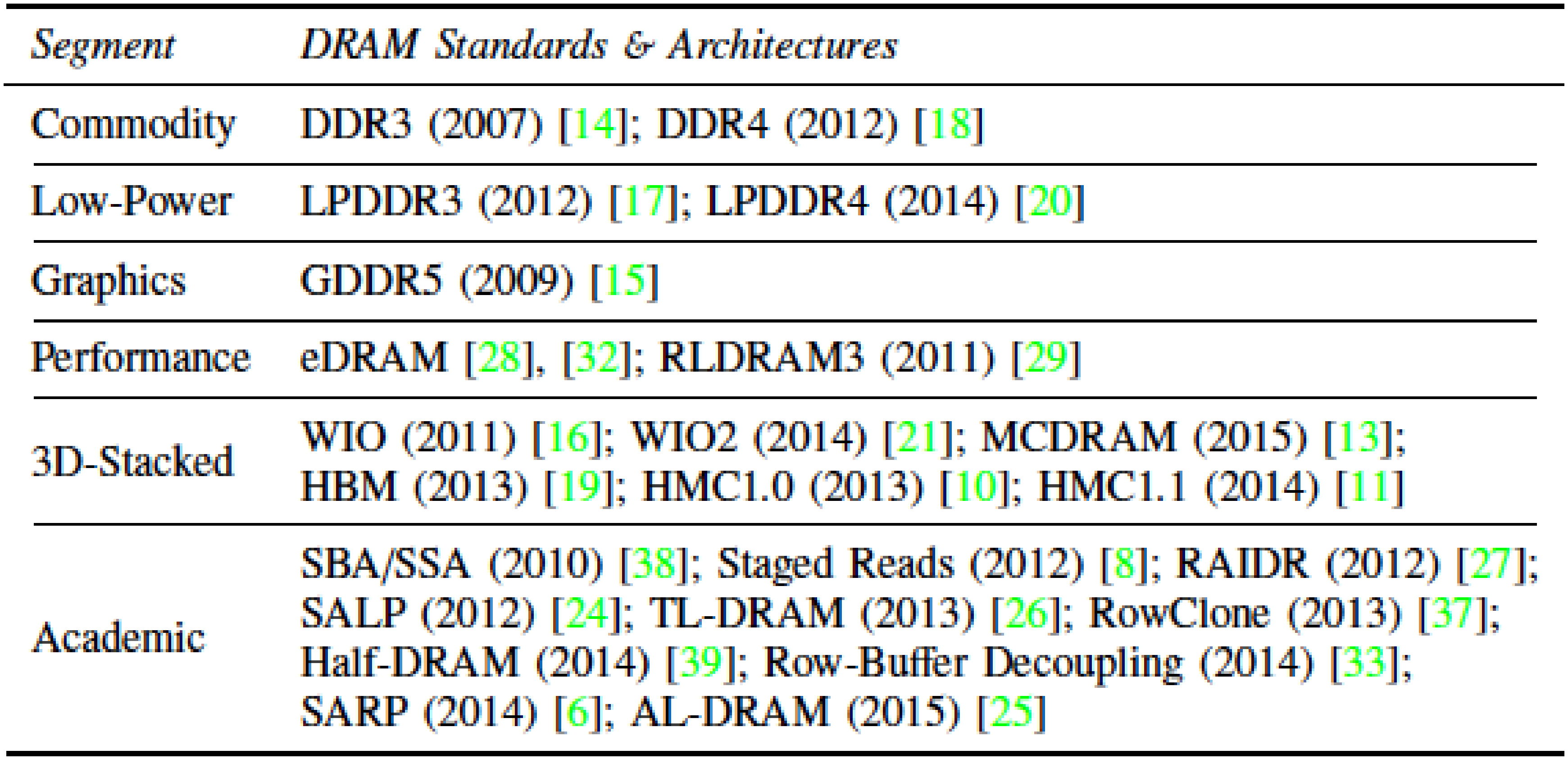
Fig. 1. ***(a) SSD system architecture, showing controller (Ctrl) and chips. (b) Detailed view of connections between controller components and chips.***

Cai+, "Error Characterization, Mitigation, and Recovery in Flash Memory Based Solid State Drives," Proc. IEEE 2017.

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, RLDRAM, …
  - 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)



Segment	*DRAM Standards & Architectures*
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

Kim et al., "Ramulator: A Fast and Extensible DRAM Simulator," IEEE Comp Arch Letters 2015.

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

DRAM Controller: Where to Place

- In chipset

  + More flexibility to plug different DRAM types into the system

  + Less power density in the CPU chip

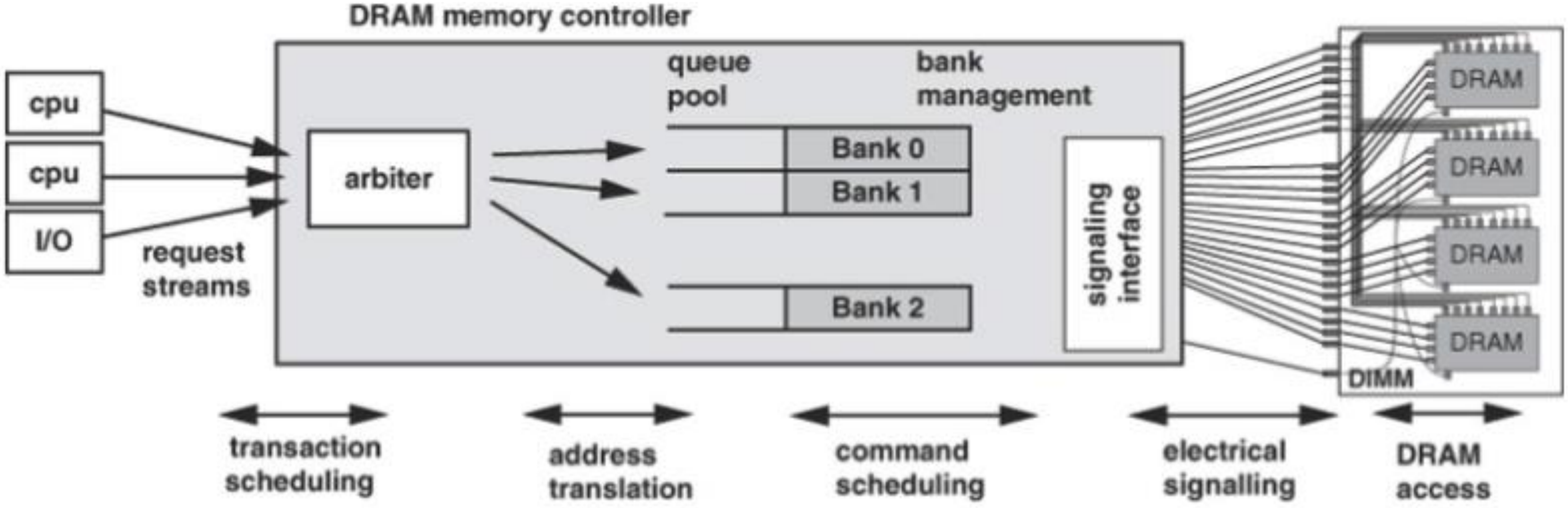
- On CPU chip

  + Reduced latency for main memory access

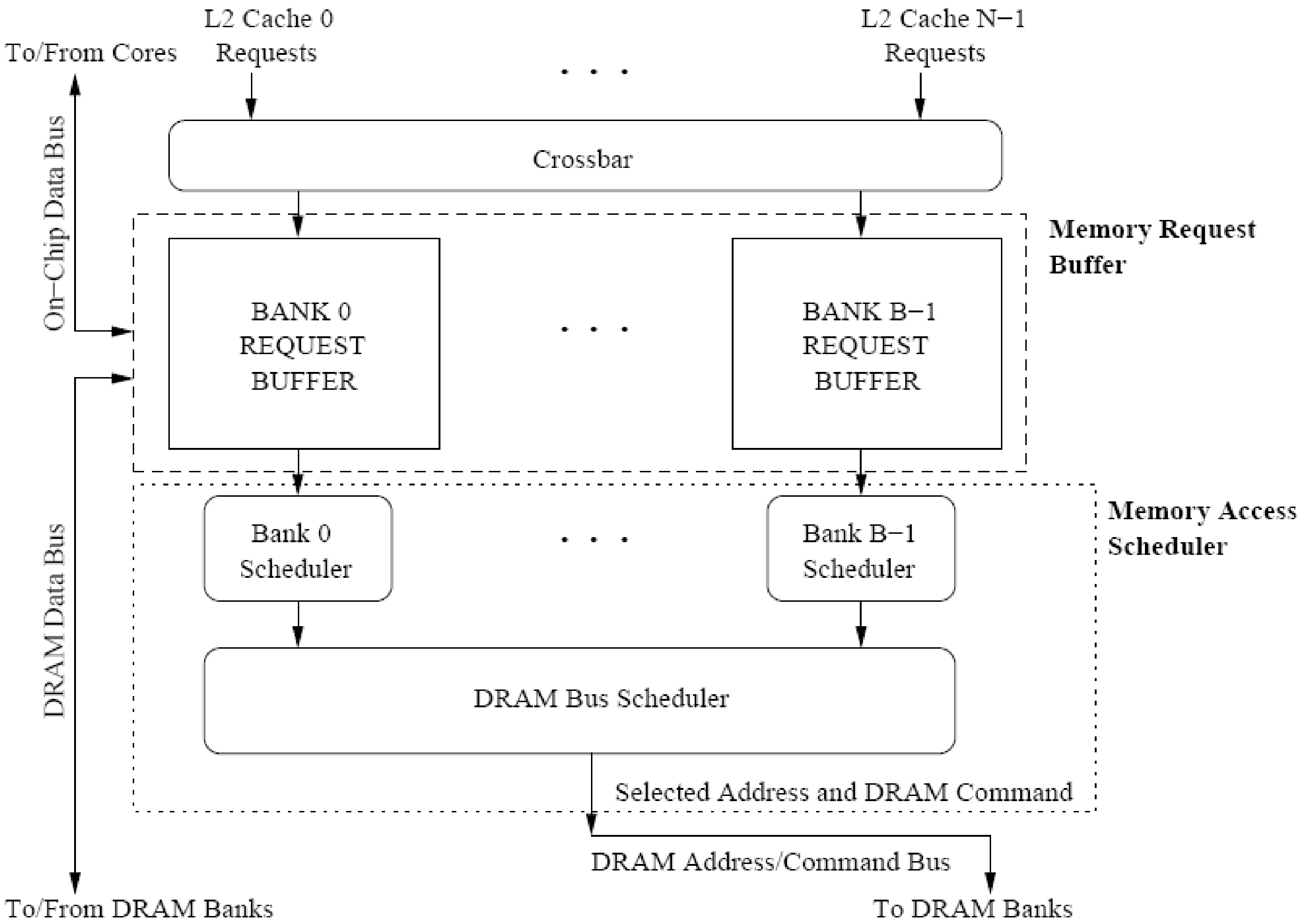
  + Higher bandwidth between cores and controller

  - More information can be communicated (e.g. request’s importance in the processing core)

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)

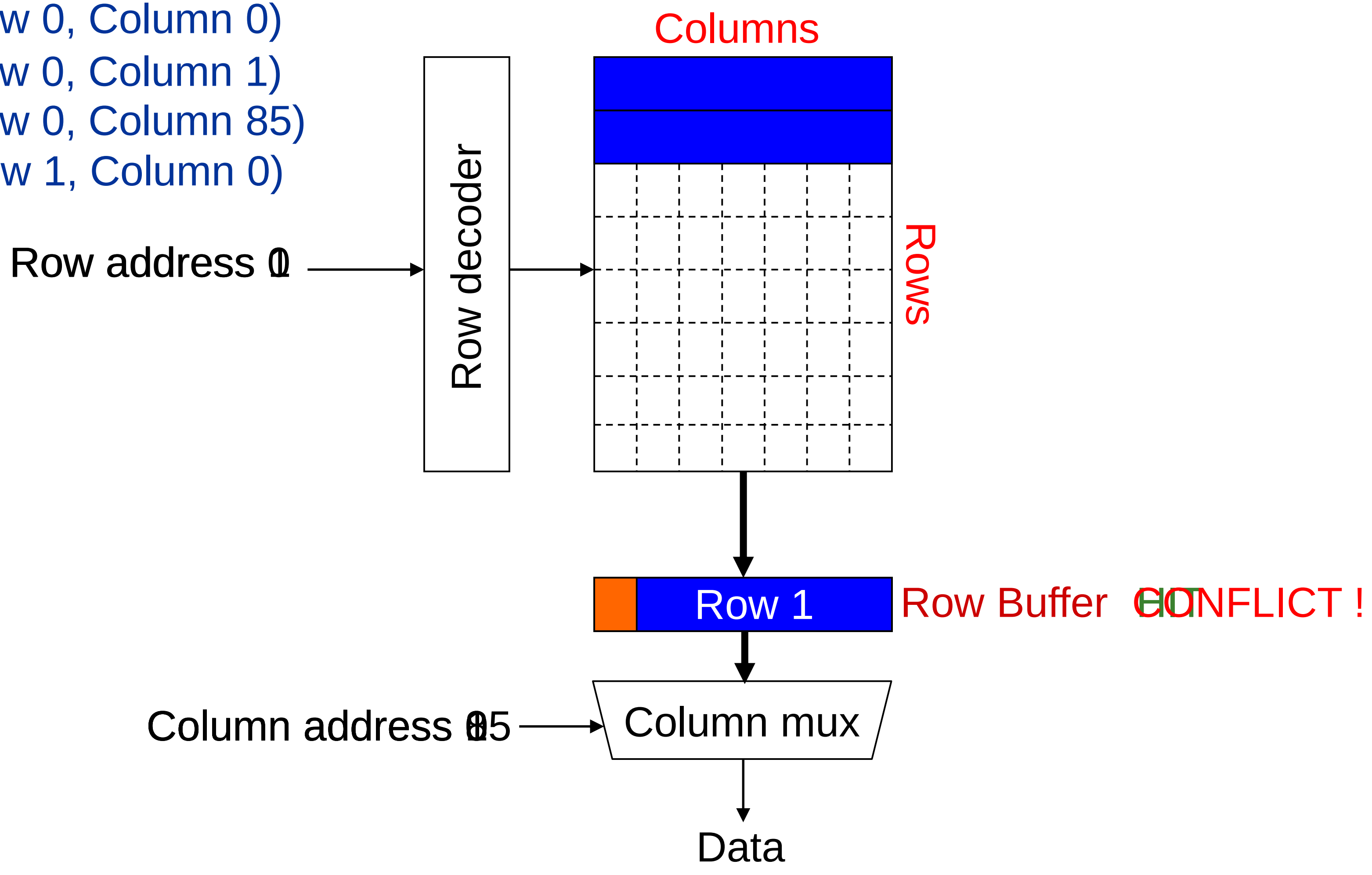
  1. Row-hit first
  2. Oldest first

  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

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)

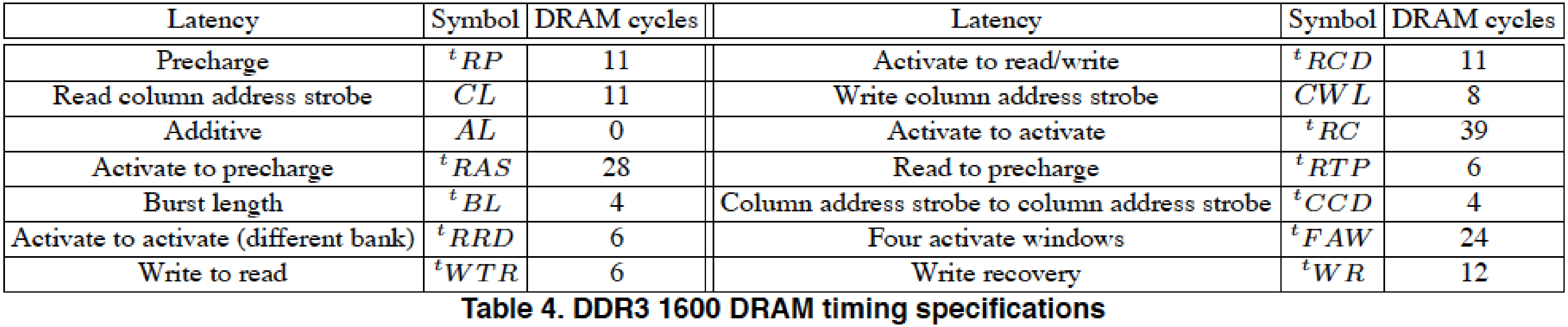
- Tradeoff: State transitions incur latency during which the chip cannot be accessed

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.

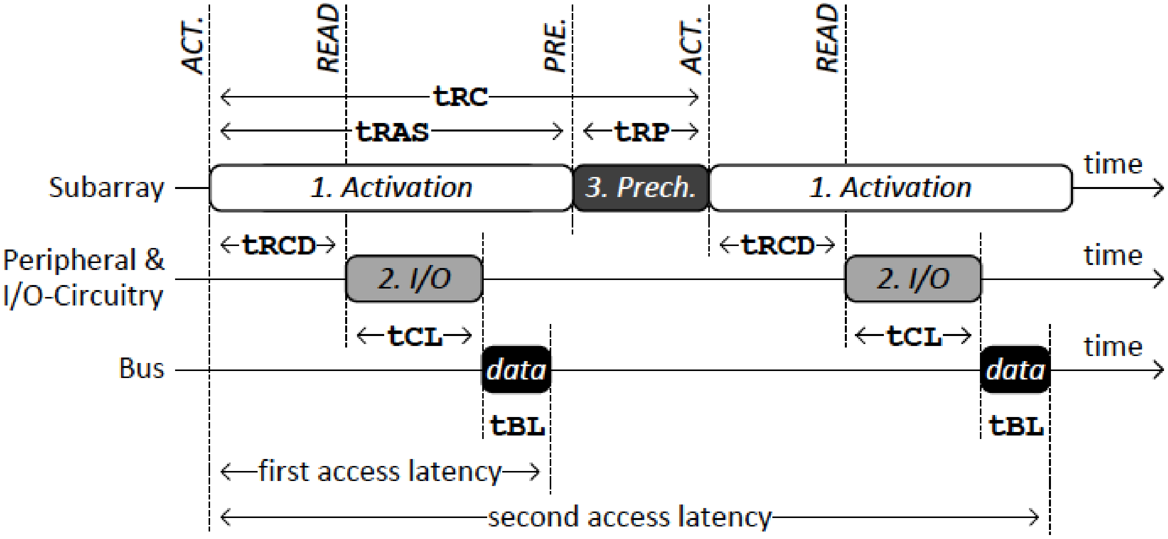


Figure 5. Three Phases of DRAM Access

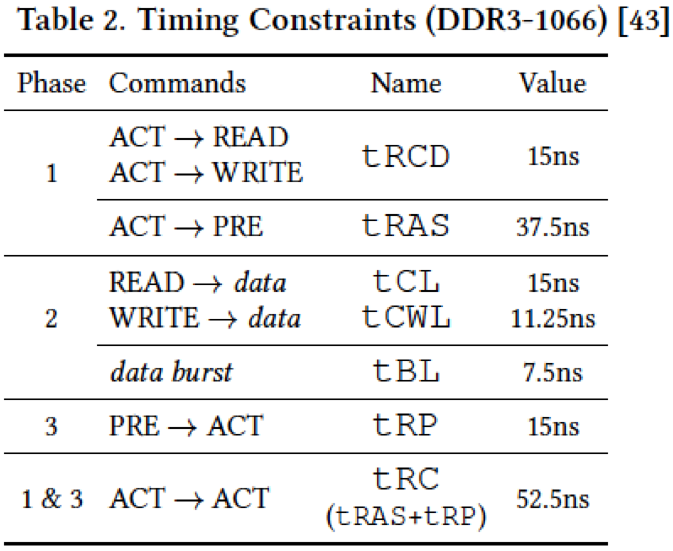


Table 2. Timing Constraints (DDR3-1066) [43]

Phase	Commands	Name	Value
1	ACT → READ 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

Why So Many Timing Constraints? (I)

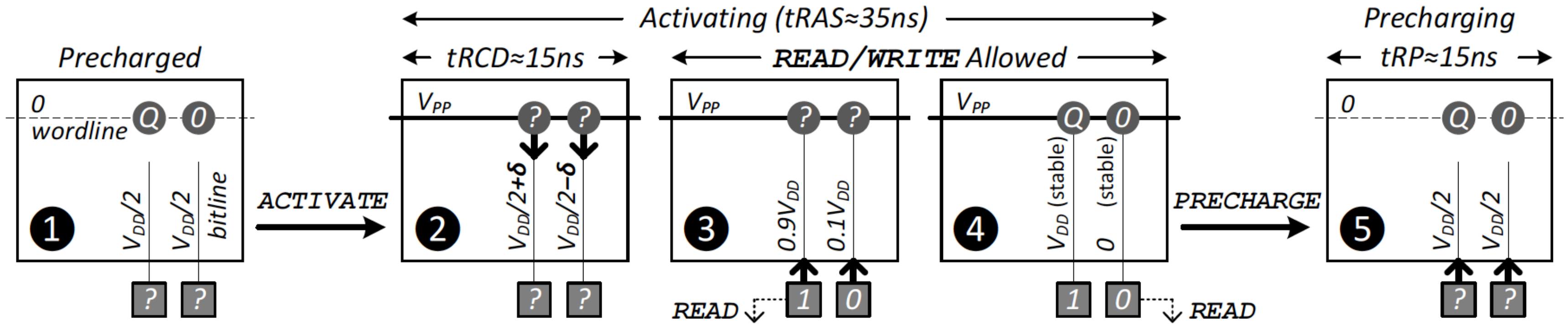


Figure 4. DRAM bank operation: Steps involved in serving a memory request [17] ($V_{PP} > V_{DD}$)

Category	RowCmd↔RowCmd			RowCmd↔ColCmd			ColCmd↔ColCmd			ColCmd→DATA	
Name	*tRC*	*tRAS*	*tRP*	*tRCD*	*tRTP*	*tWR**	*tCCD*	*tRTW*†	*tWTR**	*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)

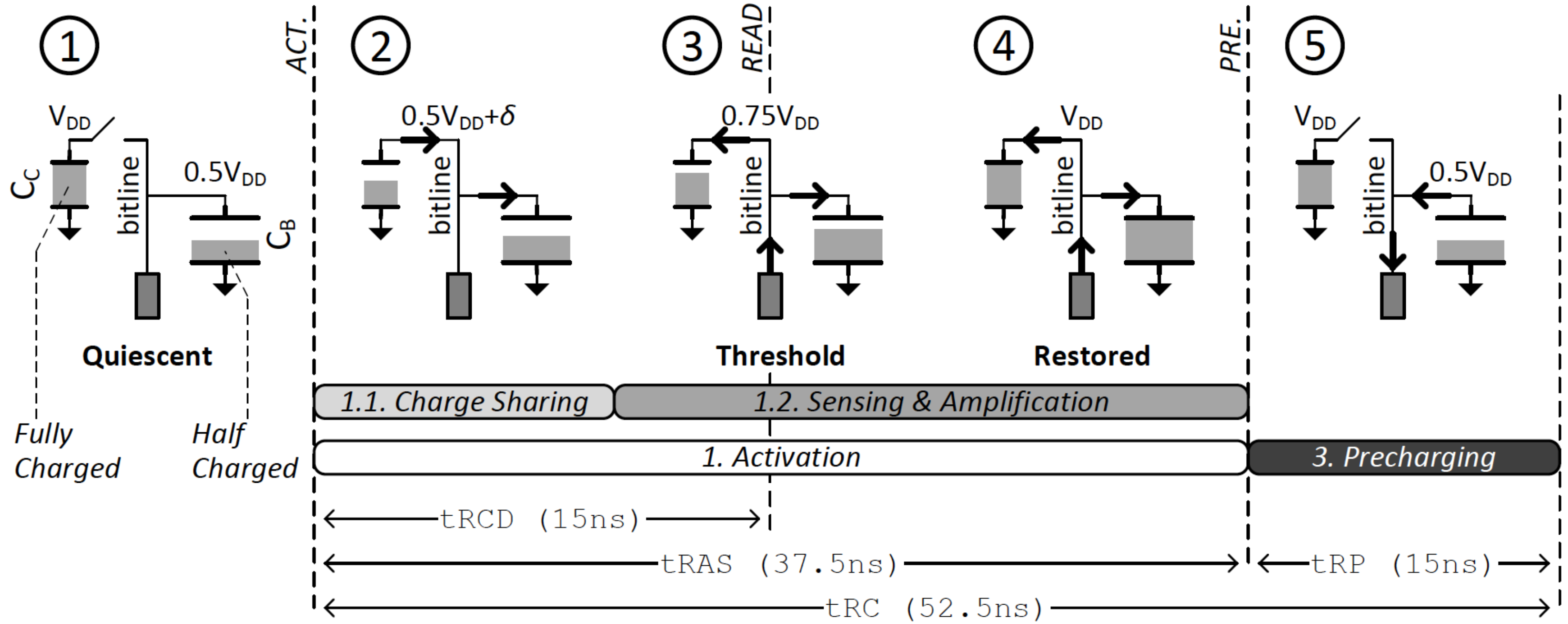
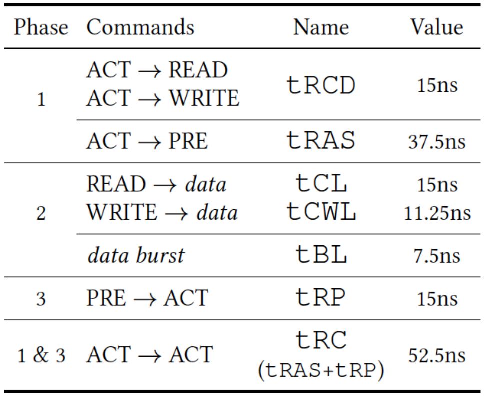


Figure 6. Charge Flow Between the Cell Capacitor (C_C), Bitline Parasitic Capacitor (C_B), and the Sense-Amplifier ($C_B \approx 3.5C_C$ [39])

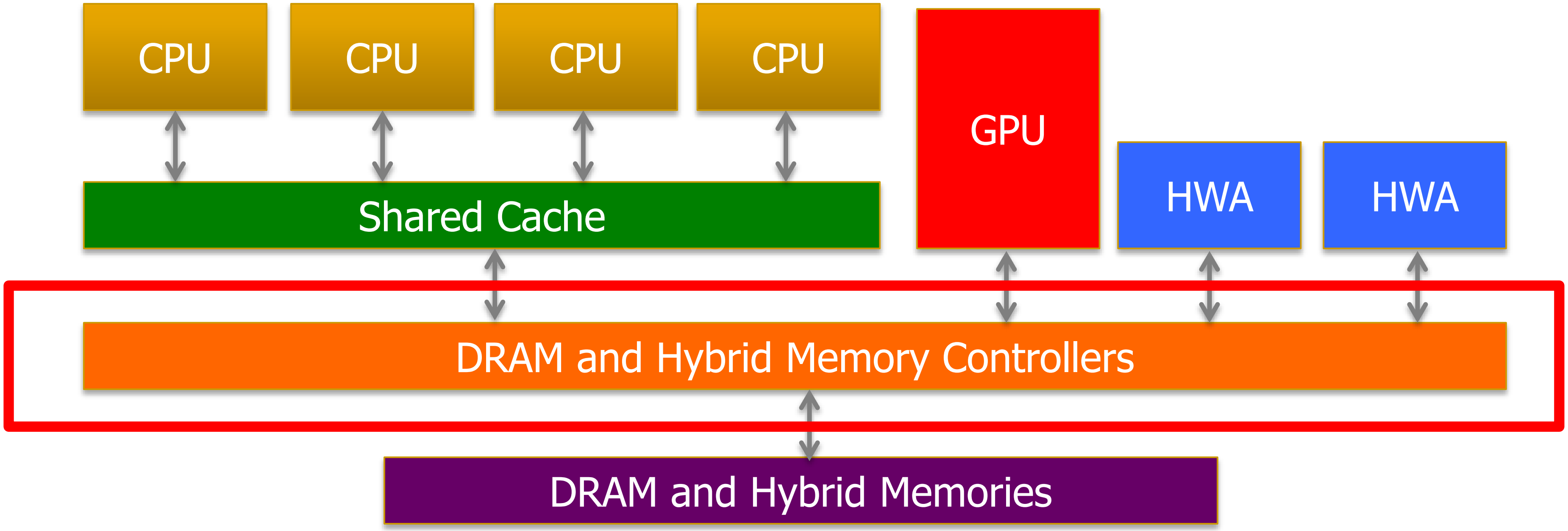
Lee et al., "Tiered-Latency DRAM: A Low Latency and Low Cost DRAM Architecture," HPCA 2013.

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



Phase	Commands	Name	Value
1	ACT → READ 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

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 difficult to optimize all these different constraints while maximizing performance, QoS, energy-efficiency, …

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

Self-Optimizing DRAM Controllers

- 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

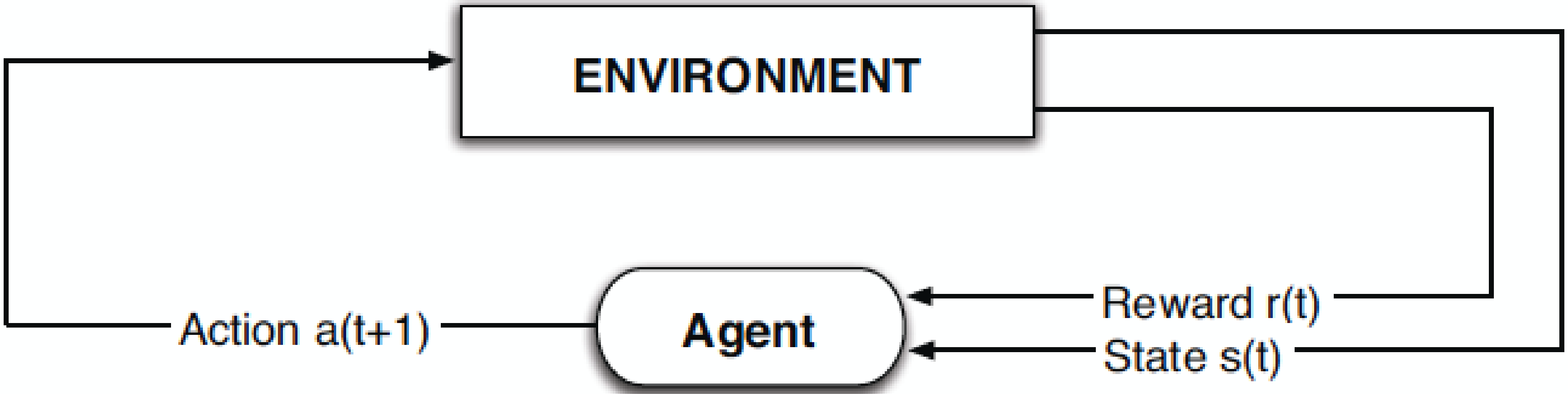
- Idea: Design a memory controller that adapts its scheduling policy decisions to workload behavior and system conditions using machine learning.

- 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+, "Self Optimizing Memory Controllers: A Reinforcement Learning Approach," ISCA 2008.

Self-Optimizing DRAM Controllers

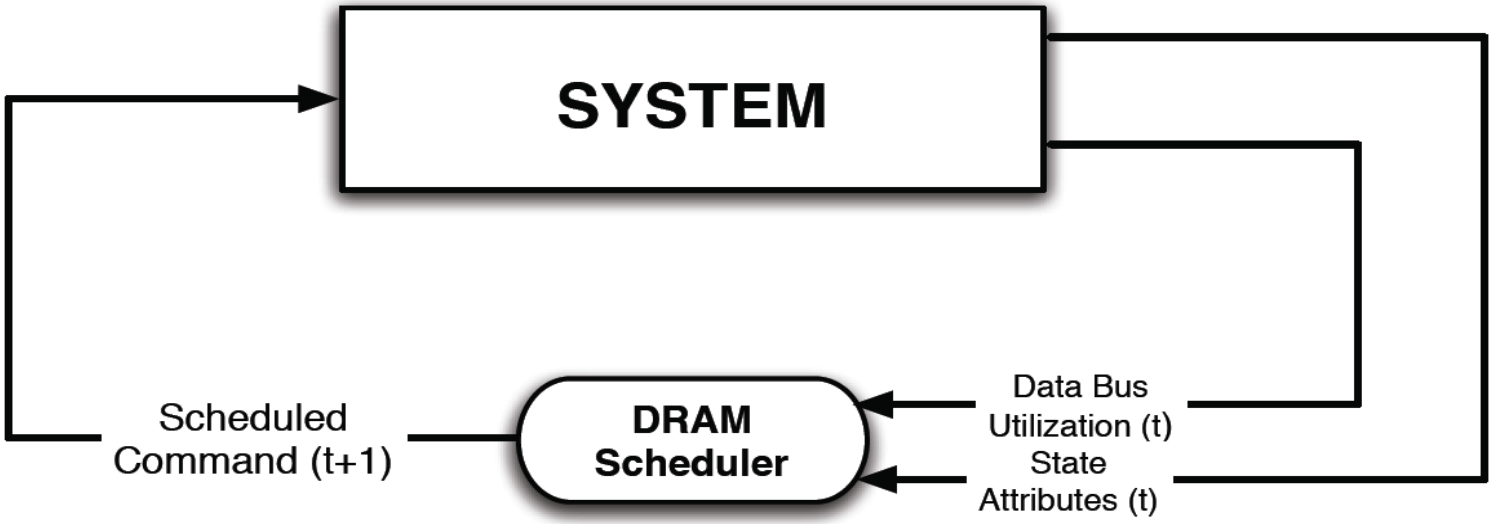


Goal: Learn to choose actions to maximize $r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots$ ($0 \leq \gamma < 1$)

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 <state, action> 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.

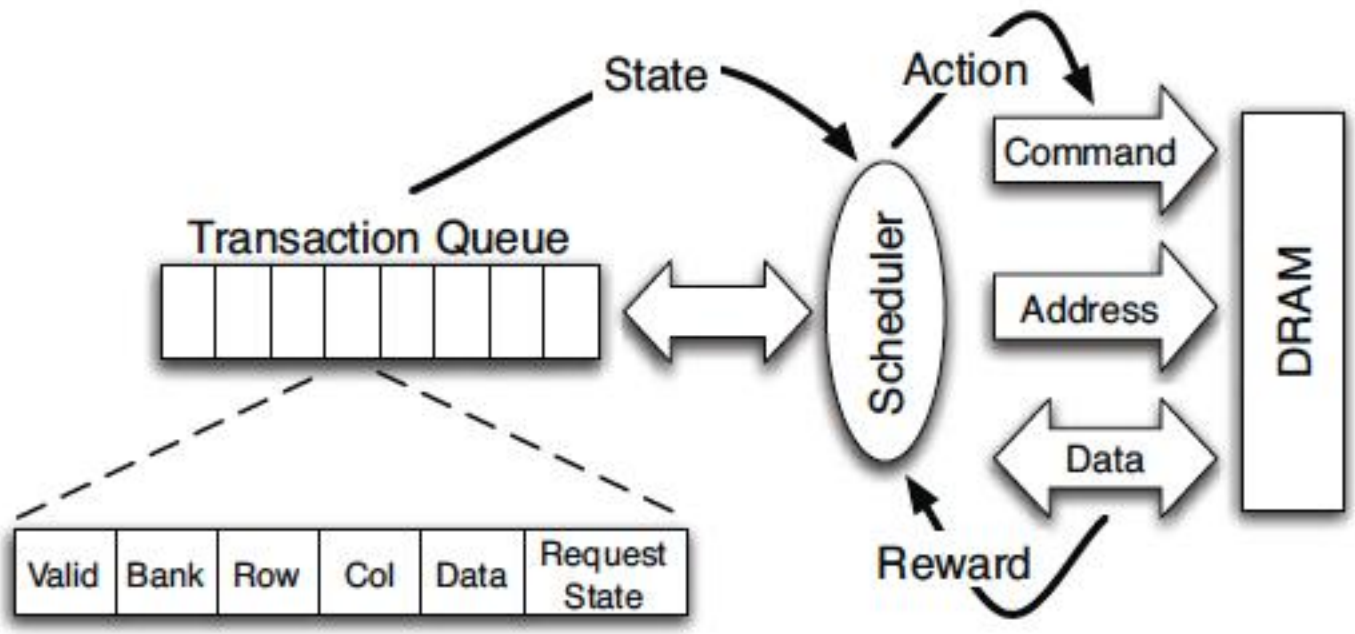


Figure 4: High-level overview of an RL-based scheduler.

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

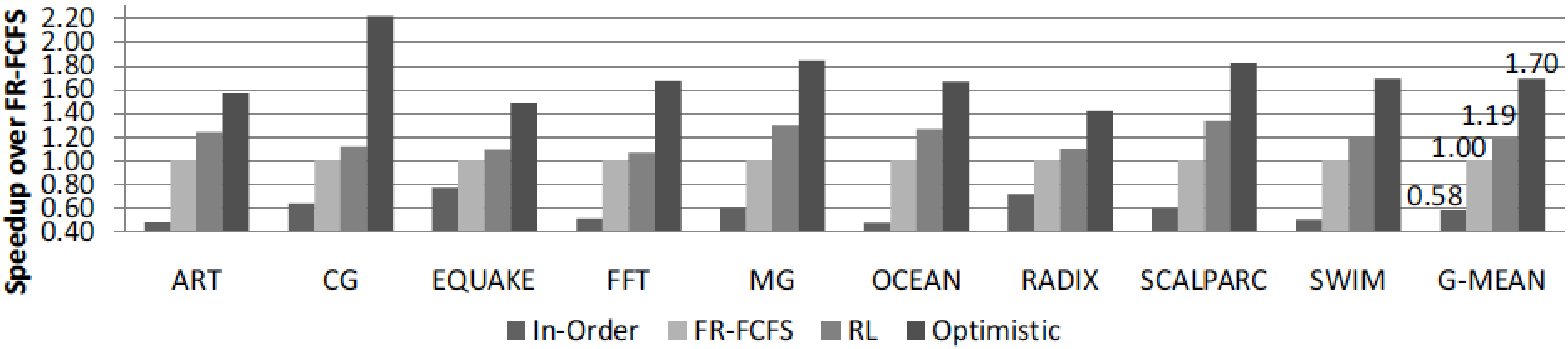


Figure 7: Performance comparison of in-order, FR-FCFS, RL-based, and optimistic memory controllers

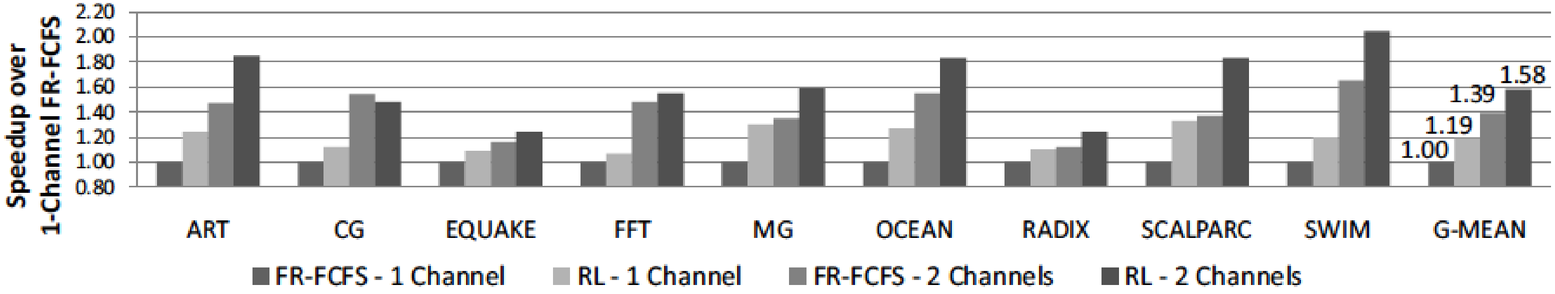


Figure 15: Performance comparison of FR-FCFS and RL-based memory controllers on systems with 6.4GB/s and 12.8GB/s peak DRAM bandwidth

Self Optimizing DRAM Controllers

- Advantages

  + Adapts the scheduling policy dynamically to changing workload behavior and to maximize a long-term target

  + Reduces the designer's 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

- Disadvantages and Limitations

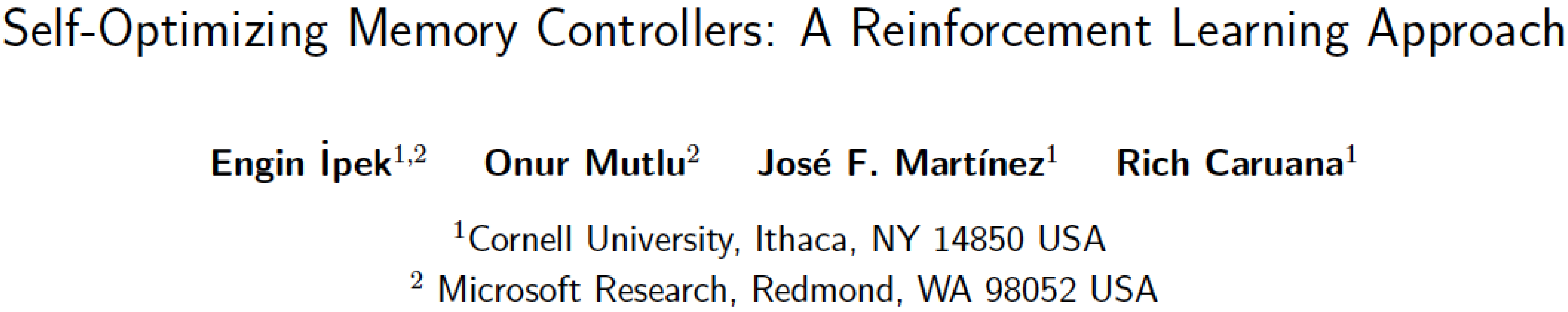
  -- Black box: designer much less likely to implement what she cannot easily reason about

  -- How to specify different reward functions that can achieve different objectives? (e.g., fairness, QoS)

  -- Hardware complexity?

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 Mutlu[2] José F. Martínez[1] Rich Caruana[1]

[1]Cornell University, Ithaca, NY 14850 USA
[2] Microsoft Research, Redmond, WA 98052 USA

Evaluating New Ideas for New (Memory) Architectures

Potential Evaluation Methods

- How do we assess an idea will improve a target metric X?

- A variety of evaluation methods are available:
    - Theoretical proof
    - Analytical modeling/estimation
    - Simulation (at varying degrees of abstraction and accuracy)
    - Prototyping with a real system (e.g., FPGAs)
    - Real implementation

The Difficulty in Architectural Evaluation

- The answer is usually workload dependent
  - E.g., think caching
  - E.g., think pipelining
  - E.g., think any idea we talked about (RAIDR, Mem. Sched., …)

- Workloads change

- System has many design choices and parameters
  - Architect needs to decide many ideas and many parameters for a design
  - Not easy to evaluate all possible combinations!

- System parameters may change

Simulation: The Field of Dreams

Dreaming and Reality

- An architect is in part a dreamer, a creator
- Simulation is a key tool of the architect
- Simulation enables
  - The exploration of many dreams
  - A reality check of the dreams
  - Deciding which dream is better
- Simulation also enables
  - The ability to fool yourself with false dreams

Why High-Level Simulation?

- Problem: RTL simulation is intractable for design space exploration → too time consuming to design and evaluate
    - Especially over a large number of workloads
    - Especially if you want to predict the performance of a good chunk of a workload on a particular design
    - Especially if you want to consider many design choices
        - Cache size, associativity, block size, algorithms
        - Memory control and scheduling algorithms
        - In-order vs. out-of-order execution
        - Reservation station sizes, ld/st queue size, register file size, …
        - …

- Goal: Explore design choices quickly to see their impact on the workloads we are designing the platform for

Different Goals in Simulation

- Explore the design space quickly and see what you want to
  - potentially implement in a next-generation platform
  - propose as the next big idea to advance the state of the art
  - the goal is mainly to see relative effects of design decisions

- Match the behavior of an existing system so that you can
  - debug and verify it at cycle-level accuracy
  - propose small tweaks to the design that can make a difference in performance or energy
  - the goal is very high accuracy

- Other goals in-between:
  - Refine the explored design space without going into a full detailed, cycle-accurate design
  - Gain confidence in your design decisions made by higher-level design space exploration

Tradeoffs in Simulation

- Three metrics to evaluate a simulator
  - Speed
  - Flexibility
  - Accuracy

- Speed: How fast the simulator runs (xIPS, xCPS, slowdown)
- Flexibility: How quickly one can modify the simulator to evaluate different algorithms and design choices?
- Accuracy: How accurate the performance (energy) numbers the simulator generates are vs. a real design (Simulation error)

- The relative importance of these metrics varies depending on where you are in the design process (what your goal is)

Trading Off Speed, Flexibility, Accuracy

- Speed & flexibility affect:
  - How quickly you can make design tradeoffs

- Accuracy affects:
  - How good your design tradeoffs may end up being
  - How fast you can build your simulator (simulator design time)

- Flexibility also affects:
  - How much human effort you need to spend modifying the simulator

- You can trade off between the three to achieve design exploration and decision goals

High-Level Simulation

- Key Idea: Raise the abstraction level of modeling to give up some accuracy to enable speed & flexibility (and quick simulator design)

- Advantage

  + Can still make the right tradeoffs, and can do it quickly

  + All you need is modeling the key high-level factors, you can omit corner case conditions

  + All you need is to get the "relative trends" accurately, not exact performance numbers

- Disadvantage

  -- Opens up the possibility of potentially wrong decisions

  -- How do you ensure you get the "relative trends" accurately?

Simulation as Progressive Refinement

- High-level models (Abstract, C)
- …
- Medium-level models (Less abstract)
- …
- Low-level models (RTL with everything modeled)
- …
- Real design

- As you refine (go down the above list)
  - Abstraction level reduces
  - Accuracy (hopefully) increases (not necessarily, if not careful)
  - Flexibility reduces; Speed likely reduces except for real design
  - You can loop back and fix higher-level models

Making The Best of Architecture

- A good architect is comfortable at all levels of refinement
  - Including the extremes

- A good architect knows when to use what type of simulation
  - And, more generally, what type of evaluation method

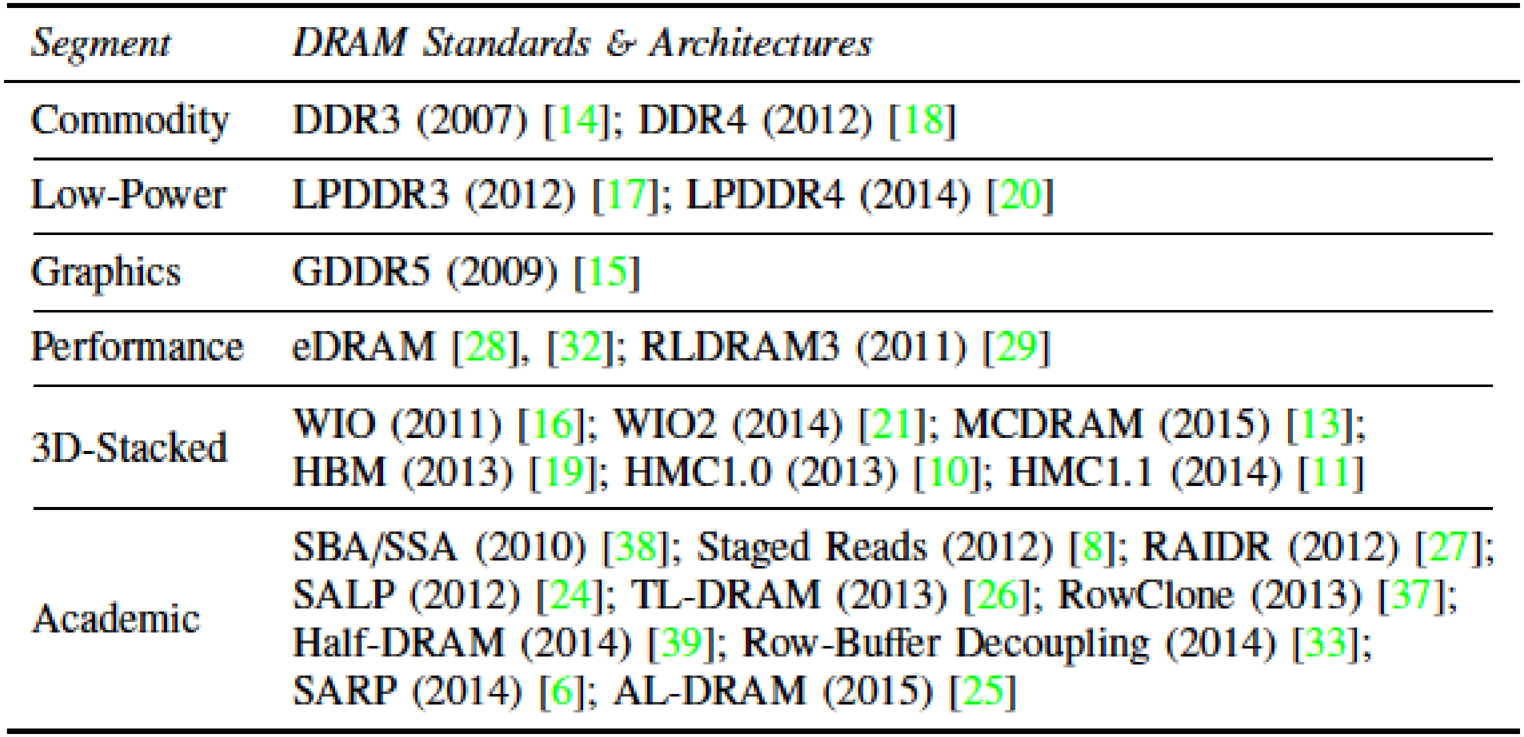
- Recall: A variety of evaluation methods are available:
  - Theoretical proof
  - Analytical modeling
  - Simulation (at varying degrees of abstraction and accuracy)
  - Prototyping with a real system (e.g., FPGAs)
  - Real implementation

Ramulator: A Fast and Extensible DRAM Simulator

[IEEE Comp Arch Letters’15]

Ramulator Motivation

- DRAM and Memory Controller landscape is changing
- Many new and upcoming standards
- Many new controller designs
- A fast and easy-to-extend simulator is very much needed

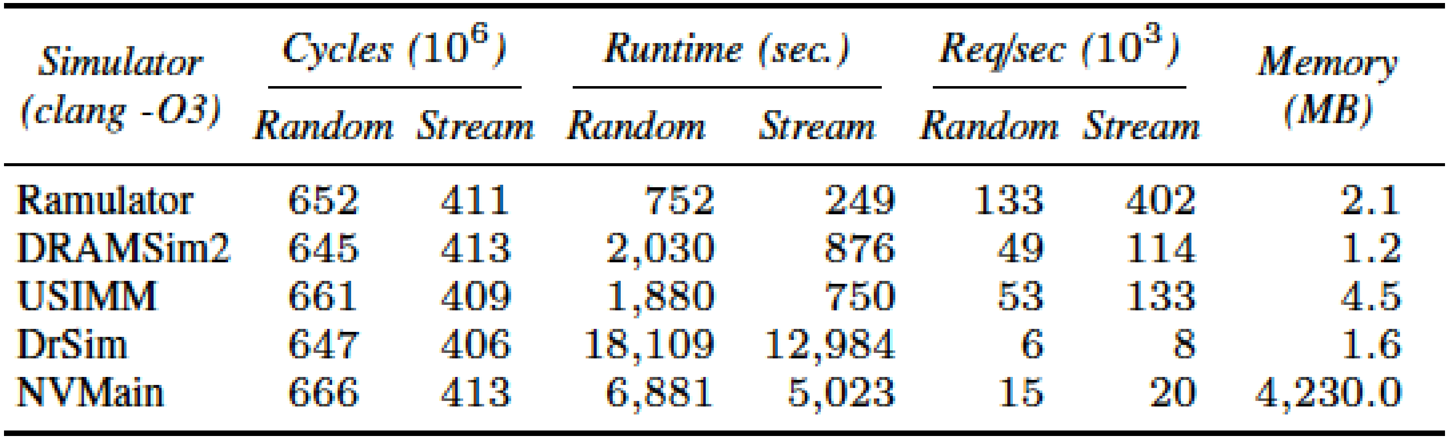


Segment	*DRAM Standards & Architectures*
Commodity	DDR3 (2007) [14]; DDR4 (2012) [18]
Low-Power	LPDDR3 (2012) [17]; LPDDR4 (2014) [20]
Graphics	GDDR5 (2009) [15]
Performance	eDRAM [28], [32]; RLDRAM3 (2011) [29]
3D-Stacked	WIO (2011) [16]; WIO2 (2014) [21]; MCDRAM (2015) [13]; HBM (2013) [19]; HMC1.0 (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

Ramulator

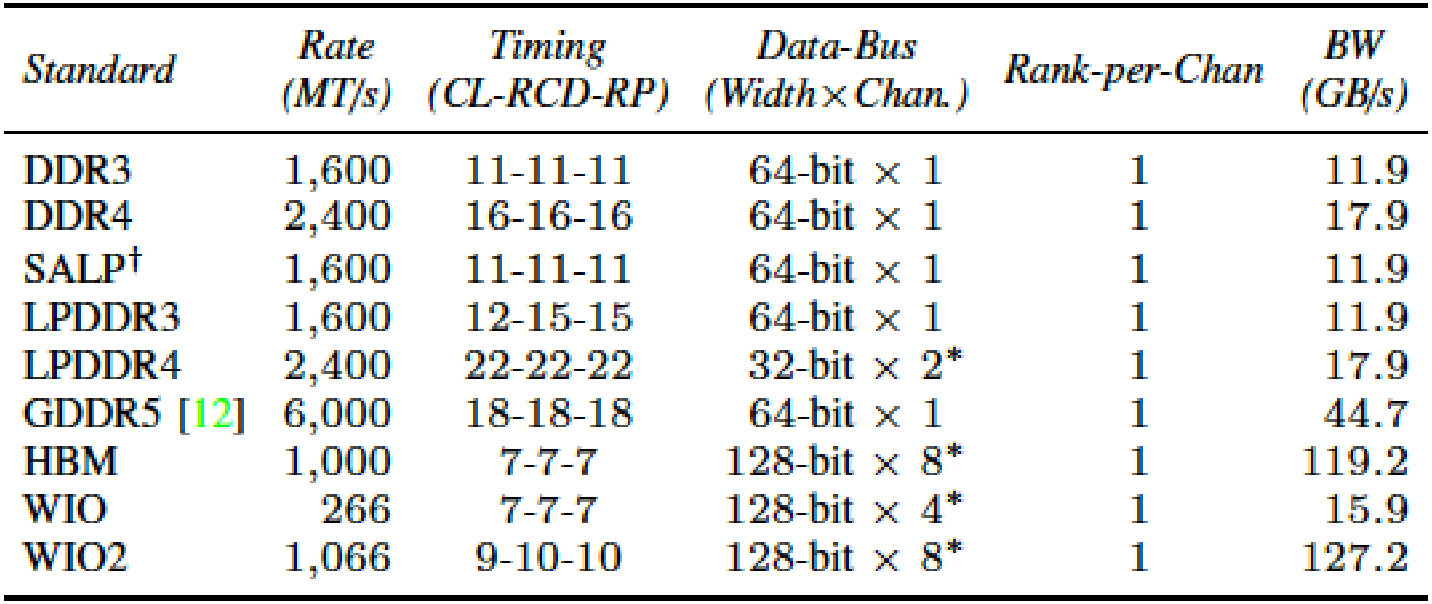
- Provides out-of-the box support for many DRAM standards:
  - DDR3/4, LPDDR3/4, GDDR5, WIO1/2, HBM, plus new proposals (SALP, AL-DRAM, TLDRAM, RowClone, and SARP)
- ~2.5X faster than fastest open-source simulator
- Modular and extensible to different standards



Simulator (clang -O3)	*Cycles (10^6)*		*Runtime (sec.)*		*Req/sec (10^3)*		*Memory (MB)*
	Random	*Stream*	*Random*	*Stream*	*Random*	*Stream*	
Ramulator	652	411	752	249	133	402	2.1
DRAMSim2	645	413	2,030	876	49	114	1.2
USIMM	661	409	1,880	750	53	133	4.5
DrSim	647	406	18,109	12,984	6	8	1.6
NVMain	666	413	6,881	5,023	15	20	4,230.0

Table 3. Comparison of five simulators using two traces

Case Study: Comparison of DRAM Standards



Standard	*Rate (MT/s)*	*Timing (CL-RCD-RP)*	*Data-Bus (Width×Chan.)*	*Rank-per-Chan*	*BW (GB/s)*
DDR3	1,600	11-11-11	64-bit × 1	1	11.9
DDR4	2,400	16-16-16	64-bit × 1	1	17.9
SALP†	1,600	11-11-11	64-bit × 1	1	11.9
LPDDR3	1,600	12-15-15	64-bit × 1	1	11.9
LPDDR4	2,400	22-22-22	32-bit × 2*	1	17.9
GDDR5 [12]	6,000	18-18-18	64-bit × 1	1	44.7
HBM	1,000	7-7-7	128-bit × 8*	1	119.2
WIO	266	7-7-7	128-bit × 4*	1	15.9
WIO2	1,066	9-10-10	128-bit × 8*	1	127.2

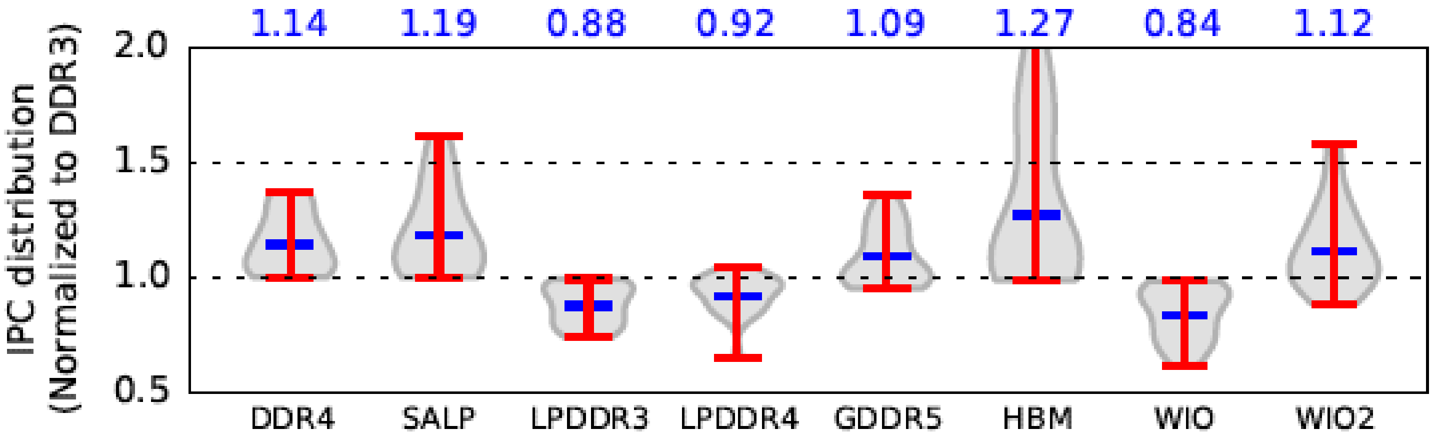


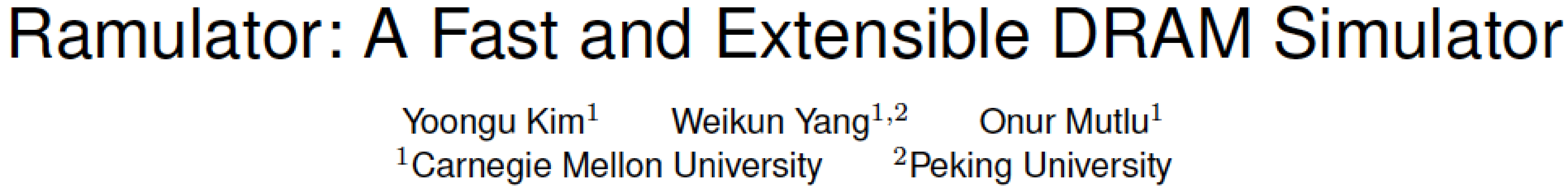
Figure 2. Performance comparison of DRAM standards

Across 22 workloads, simple CPU model

Ramulator Paper and Source Code

- Yoongu Kim, Weikun Yang, and Onur Mutlu, **"Ramulator: A Fast and Extensible DRAM Simulator"** *IEEE Computer Architecture Letters* (***CAL***), March 2015. [Source Code]

- Source code is released under the liberal MIT License
  - https://github.com/CMU-SAFARI/ramulator



Ramulator: A Fast and Extensible DRAM Simulator

Yoongu Kim[1] Weikun Yang[1,2] Onur Mutlu[1]
[1]Carnegie Mellon University [2]Peking University

Extra Credit Assignment

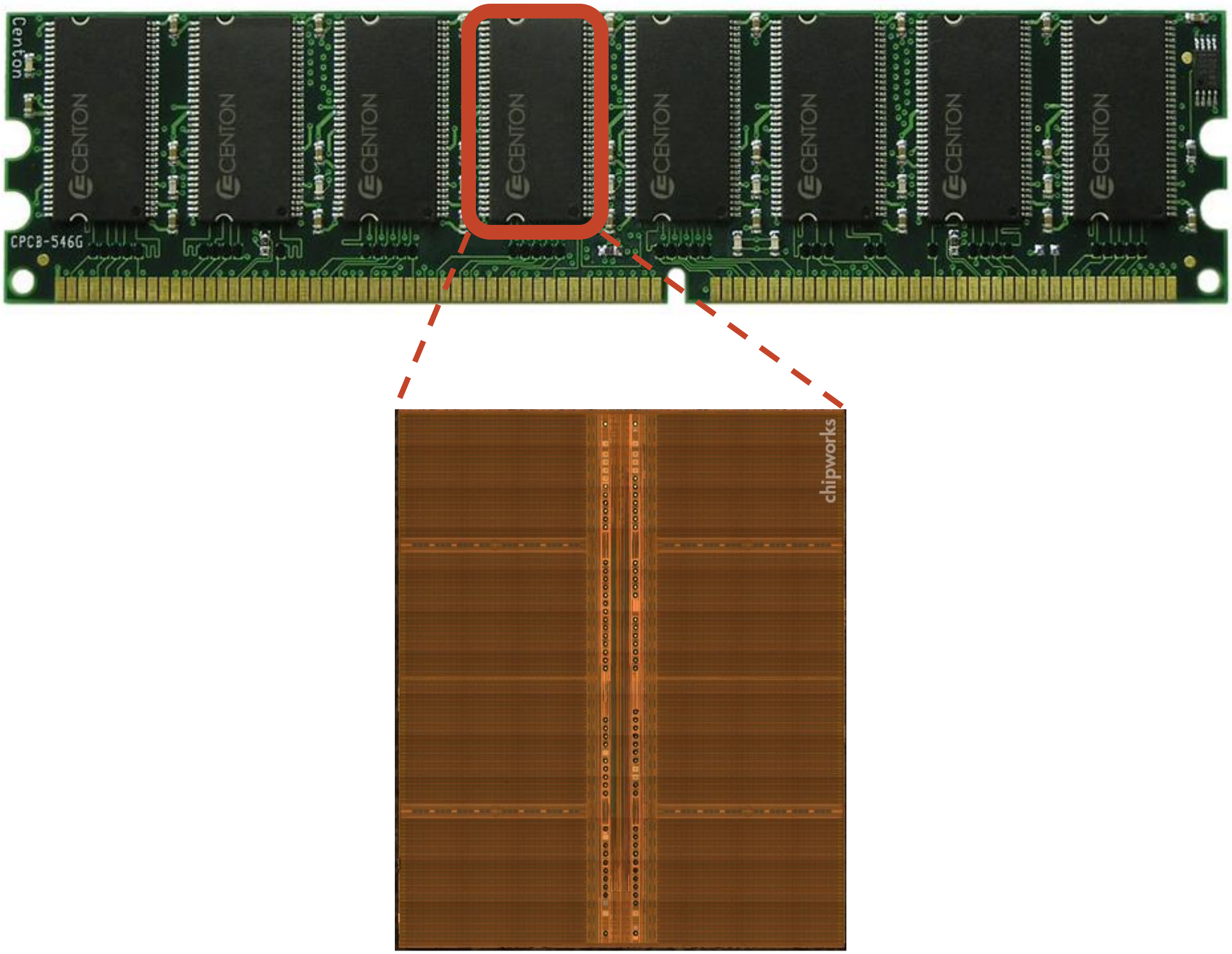
- Review the Ramulator paper
  - Online on our review site

- Download and run Ramulator
  - Compare DDR3, DDR4, SALP, HBM for the libquantum benchmark (provided in Ramulator repository)
  - Upload your brief report to Moodle

- This may become part of a future homework

Memory Latency: Fundamental Tradeoffs

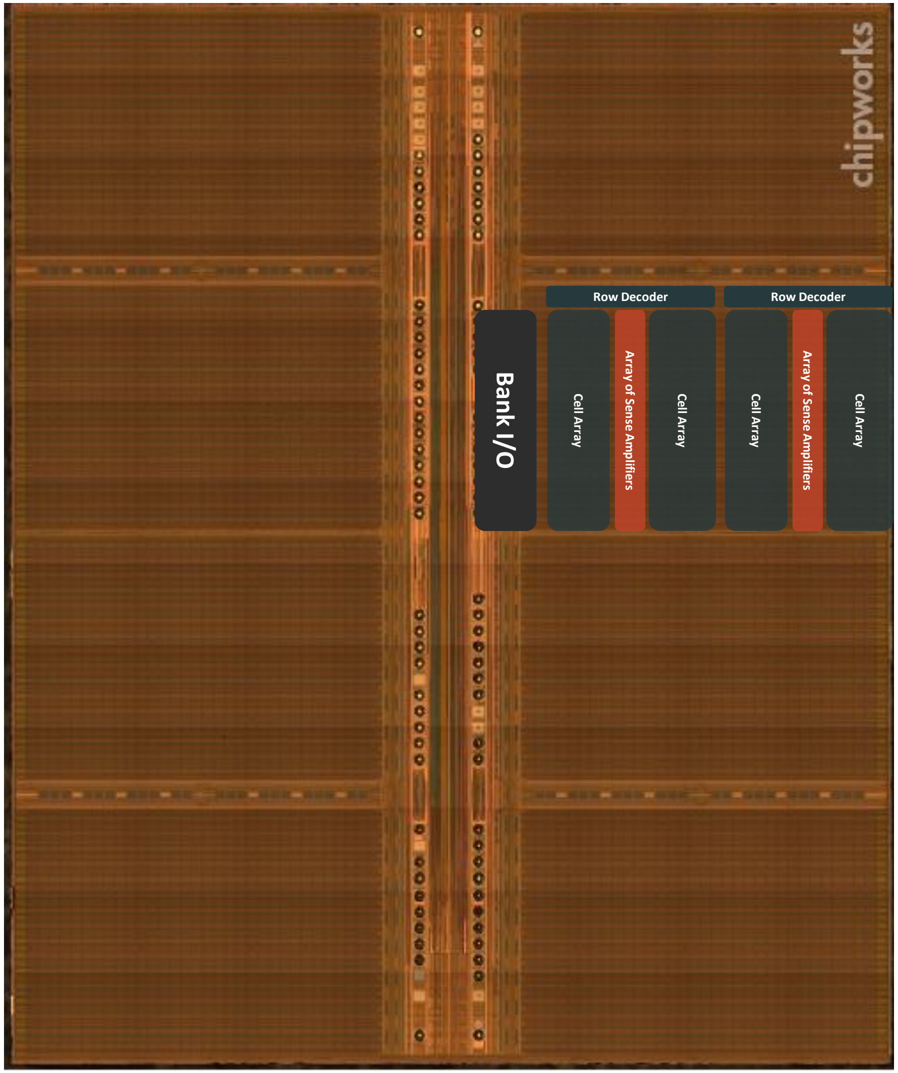
DRAM Module and Chip



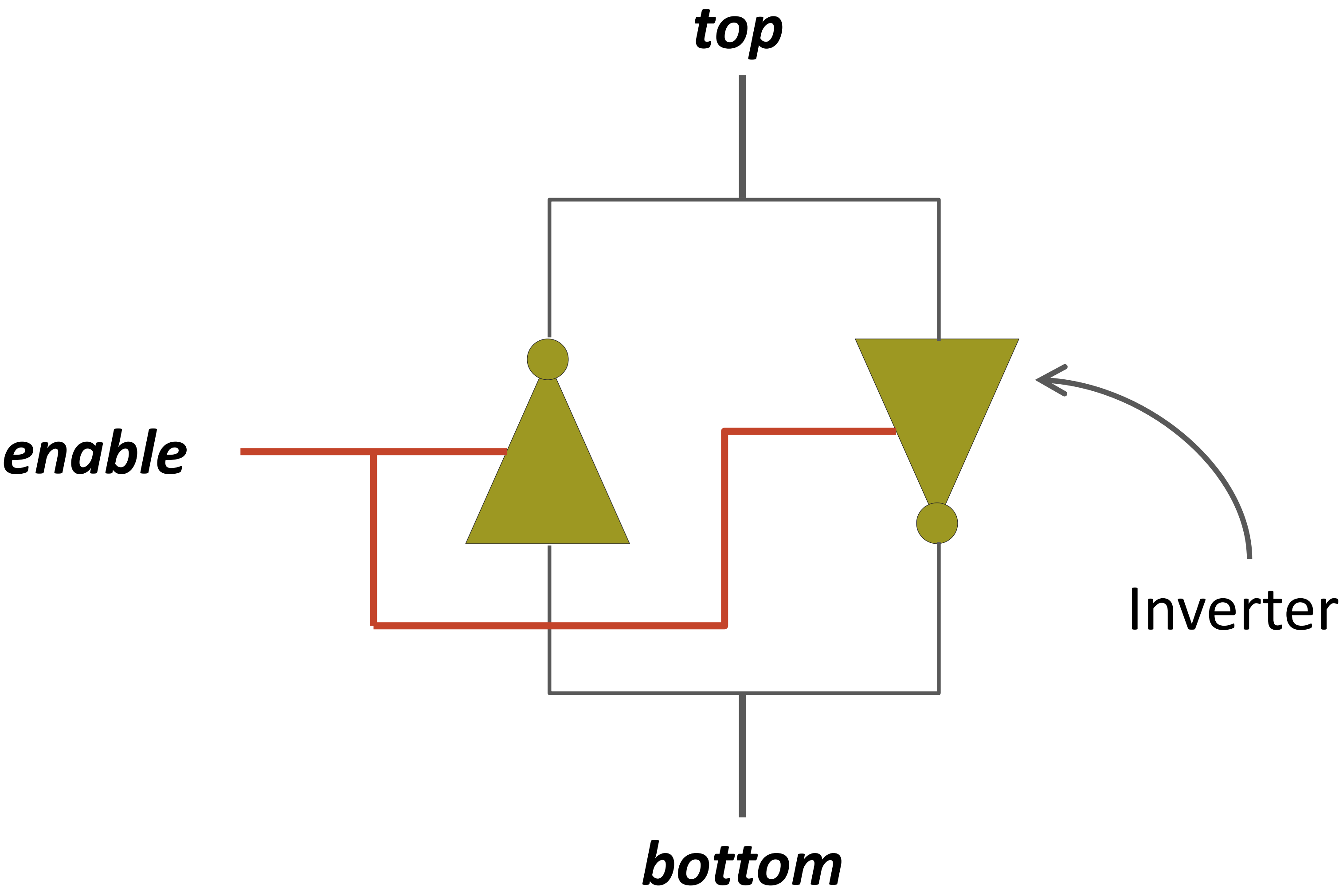
Goals

- Cost
- Latency
- Bandwidth
- Parallelism
- Power
- Energy
- Reliability
- …

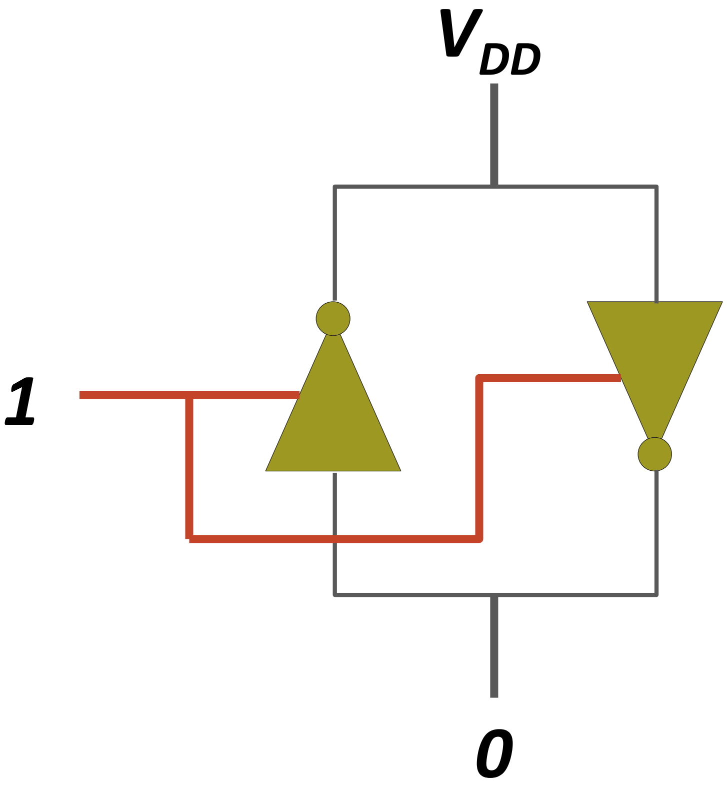
DRAM Chip


chipworks
Row Decoder
Row Decoder
Bank I/O
Cell Array
Array of Sense Amplifiers
Cell Array
Cell Array
Array of Sense Amplifiers
Cell Array

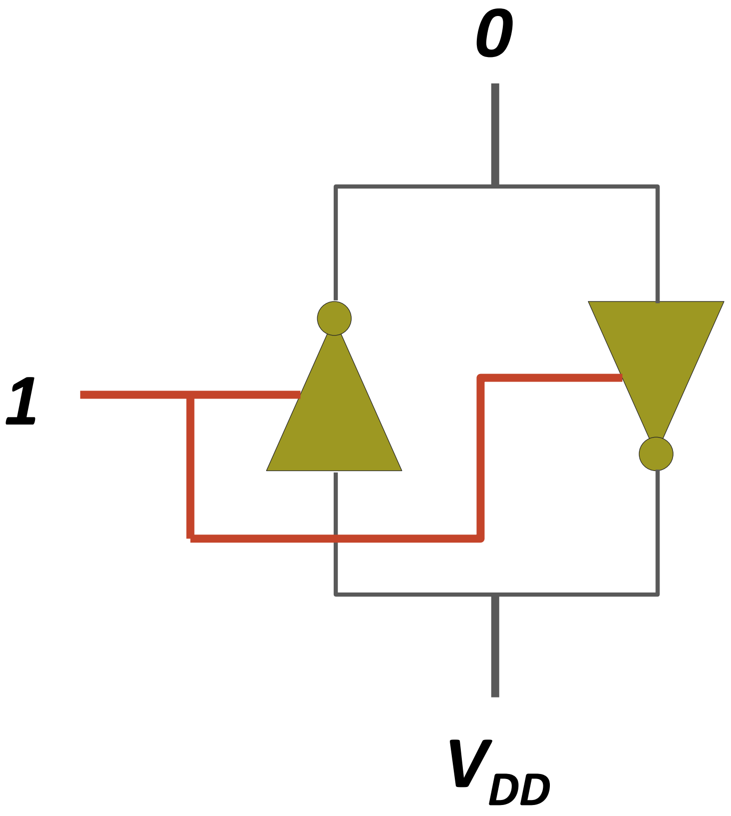
Sense Amplifier



Sense Amplifier – Two Stable States

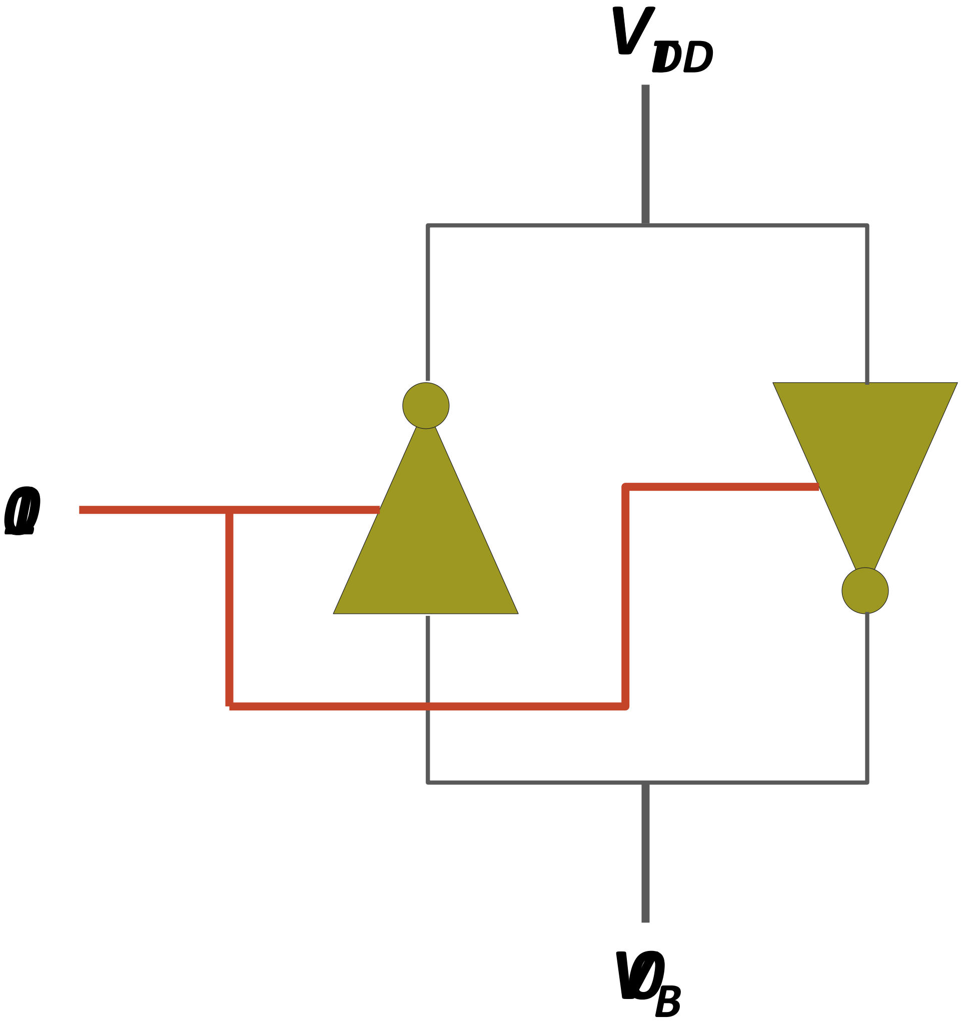


Logical "1"



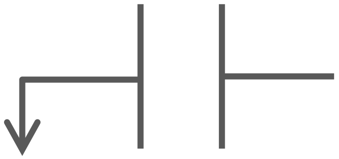
Logical "0"

Sense Amplifier Operation



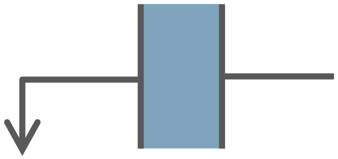
$V_T > V_B$

DRAM Cell – Capacitor



Empty State

Logical “0”



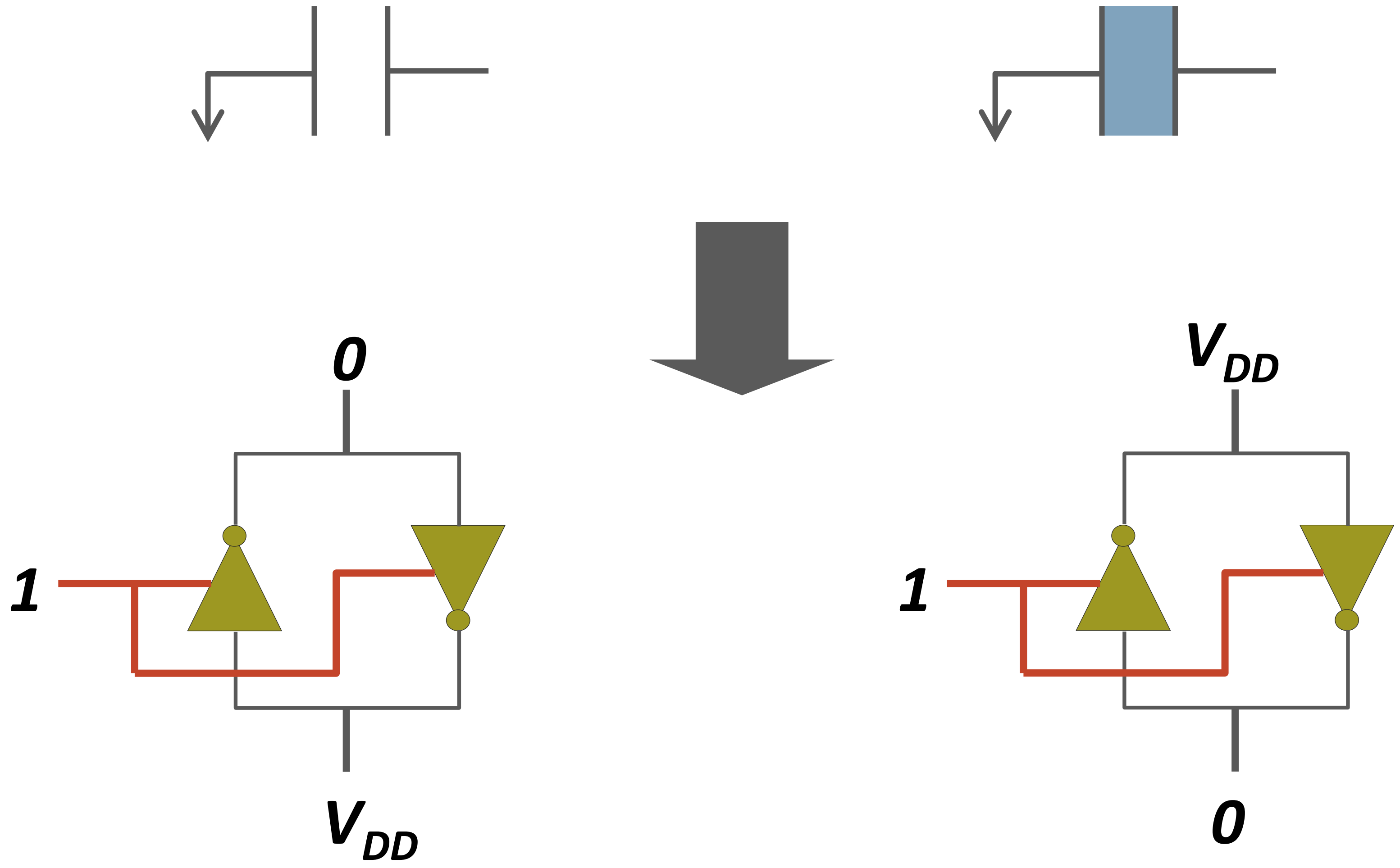
Fully Charged State

Logical “1”

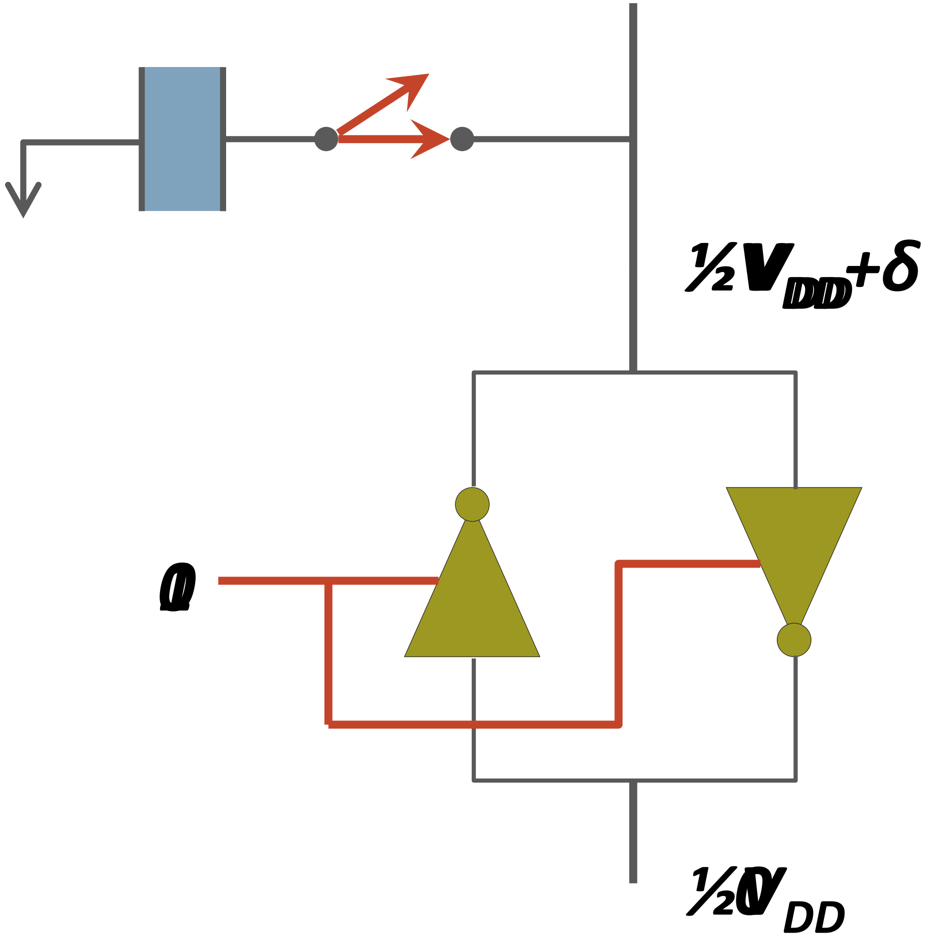
1 Small – Cannot drive circuits

2 Reading destroys the state

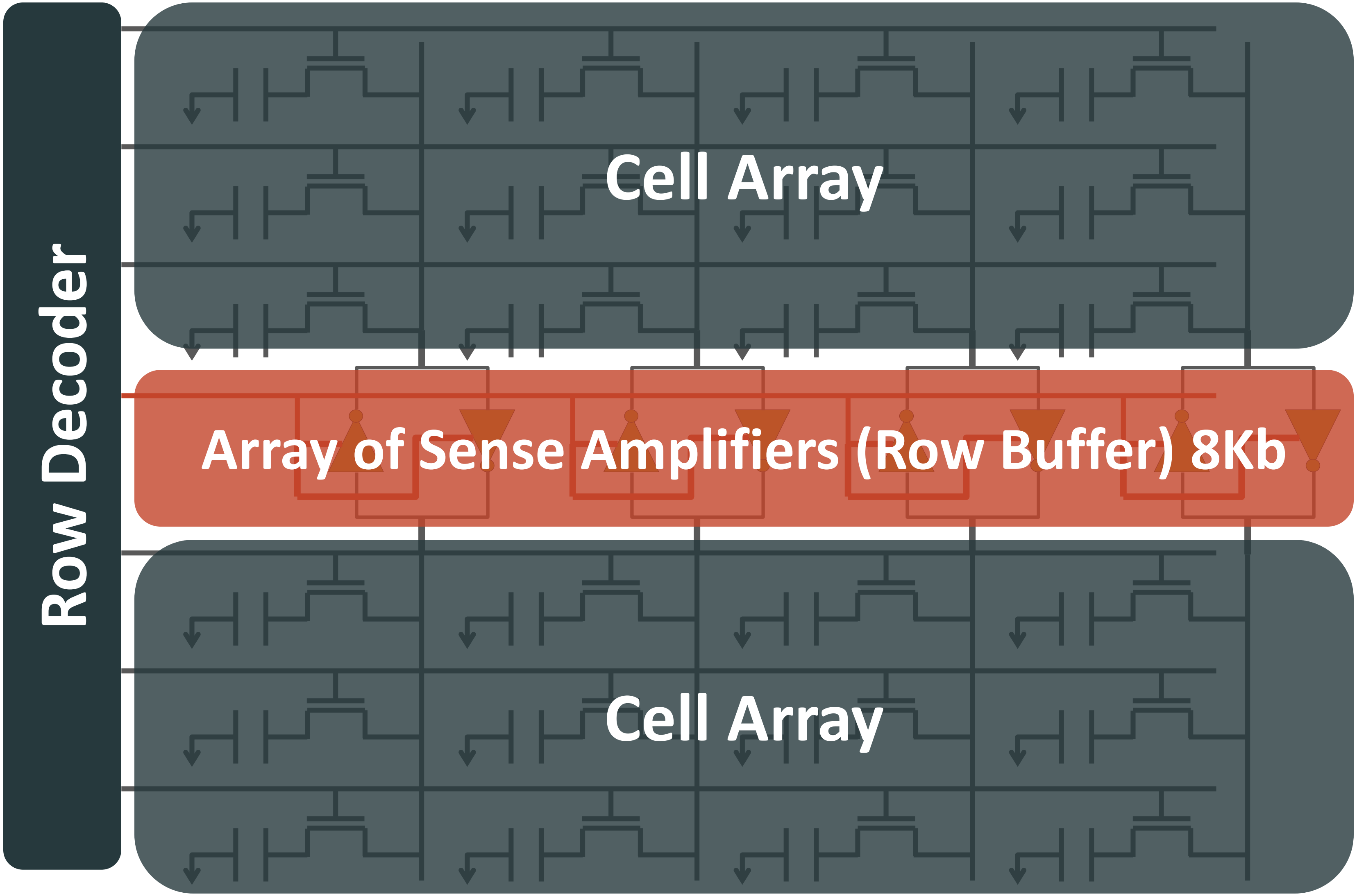
Capacitor to Sense Amplifier



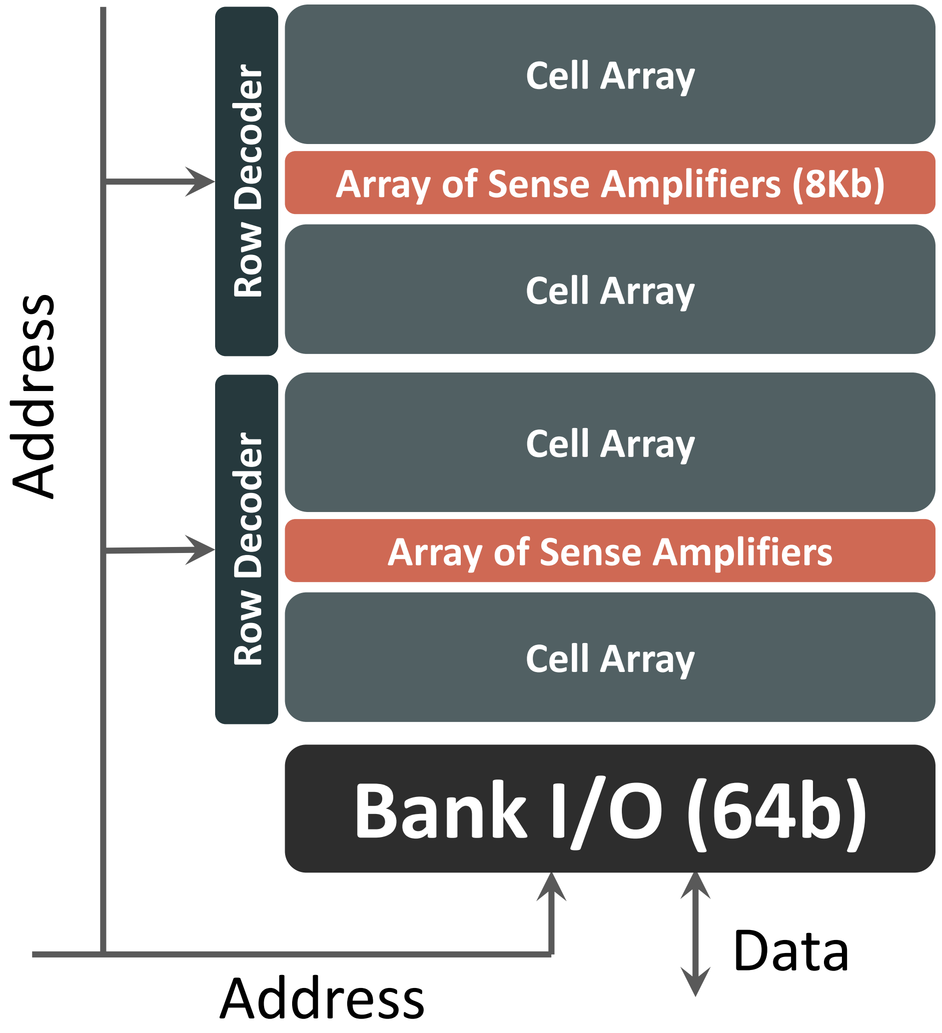
DRAM Cell Operation



DRAM Subarray – Building Block for DRAM Chip

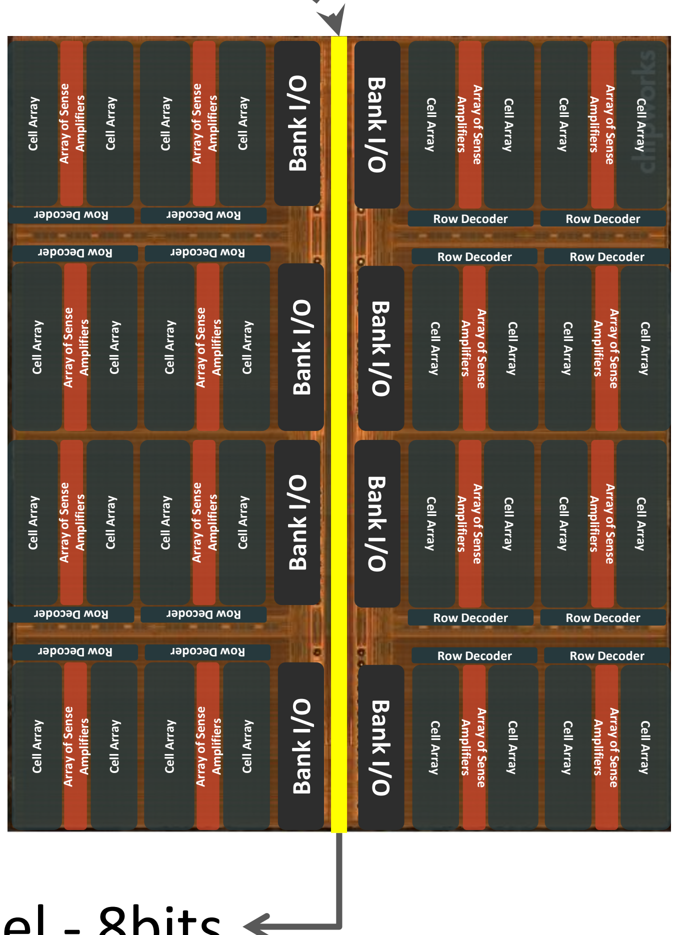


DRAM Bank


Address
Row Decoder
Cell Array
Array of Sense Amplifiers (8Kb)
Cell Array
Row Decoder
Cell Array
Array of Sense Amplifiers
Cell Array
Bank I/O (64b)
Data
Address

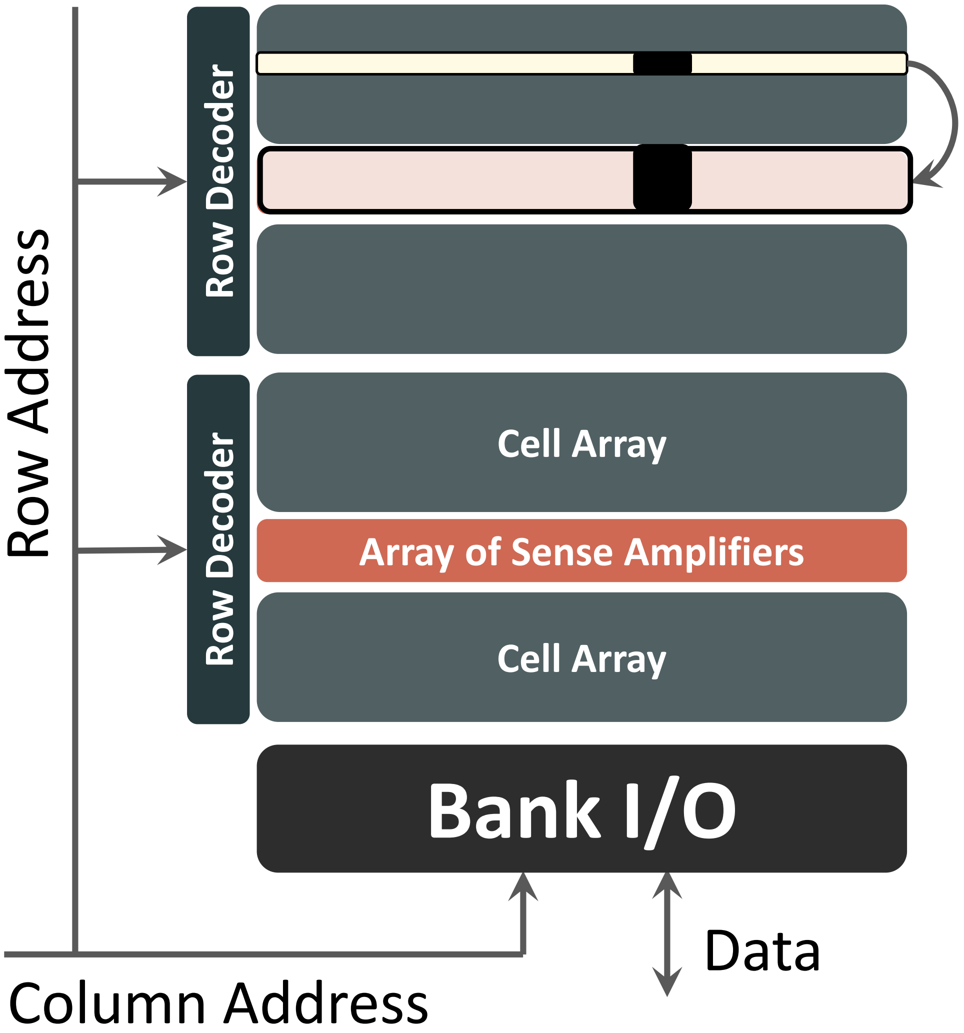
DRAM Chip

Shared internal bus



Memory channel - 8bits

DRAM Operation

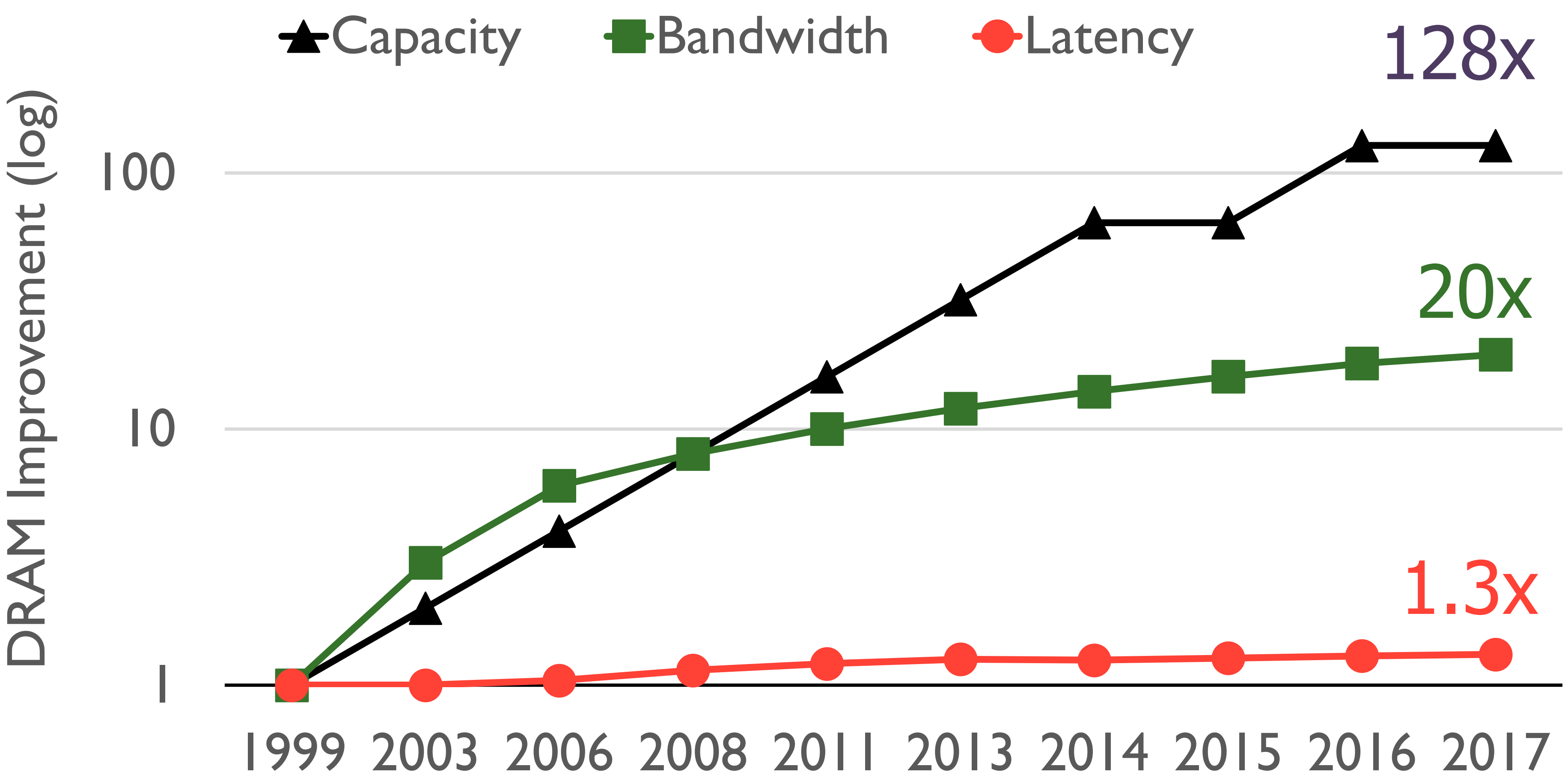


1 ACTIVATE Row

2 READ/WRITE Column

3 PRECHARGE

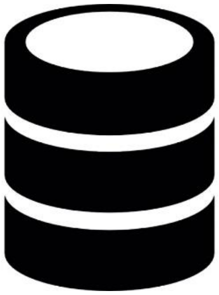
Memory Latency Lags Behind



Memory latency remains almost constant

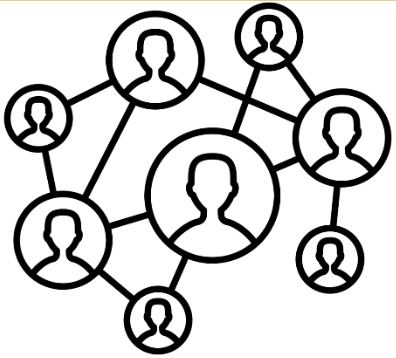
SAFARI

DRAM Latency Is Critical for Performance



In-memory Databases

[Mao+, EuroSys'12;
Clapp+ (**Intel**), IISWC'15]

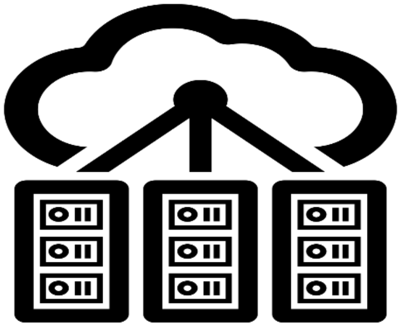


Graph/Tree Processing

[Xu+, IISWC'12; Umuroglu+, FPL'15]

In-Memory Data Analytics

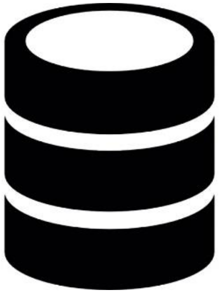
[Clapp+ (**Intel**), IISWC'15;
Awan+, BDCloud'15]



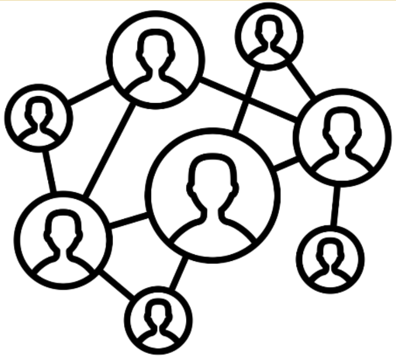
Datacenter Workloads

[Kanev+ (**Google**), ISCA'15]

DRAM Latency Is Critical for Performance



In-memory Databases

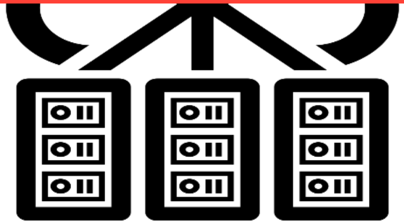


Graph/Tree Processing

Long memory latency → performance bottleneck

In-Memory Data Analytics

[Clapp+ (**Intel**), IISWC'15;
Awan+, BDCloud'15]



Datacenter Workloads

[Kanev+ (**Google**), ISCA'15]

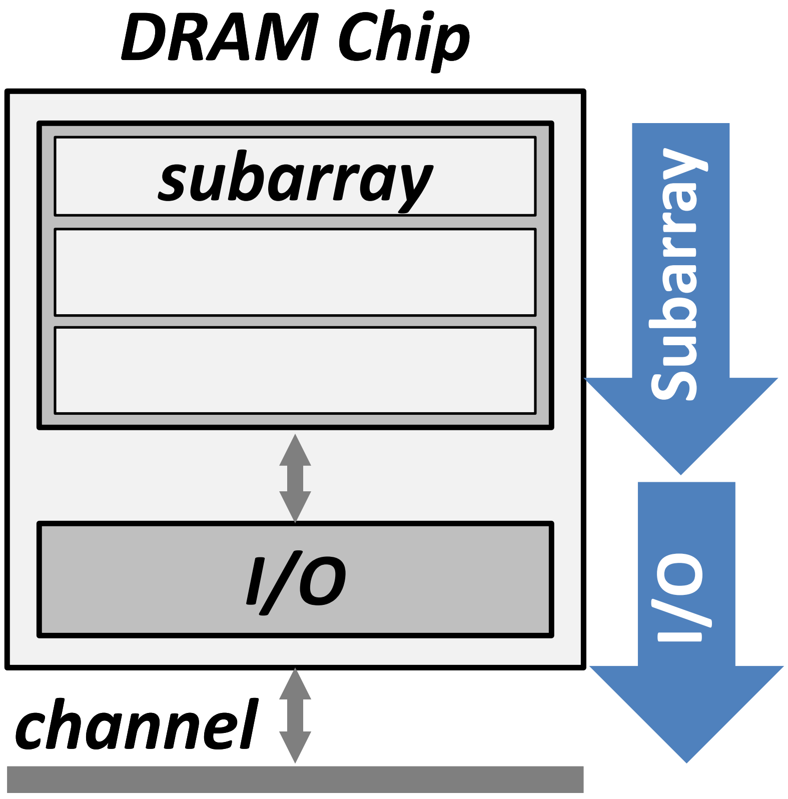
What Causes the Long DRAM Latency?

Why the Long Latency?

- Reason 1: Design of DRAM Micro-architecture
  - Goal: Maximize capacity/area, not minimize latency

- Reason 2: "One size fits all" approach to latency specification
  - Same latency parameters for all temperatures
  - Same latency parameters for all DRAM chips (e.g., rows)
  - Same latency parameters for all parts of a DRAM chip
  - Same latency parameters for all supply voltage levels
  - Same latency parameters for all application data
  - …

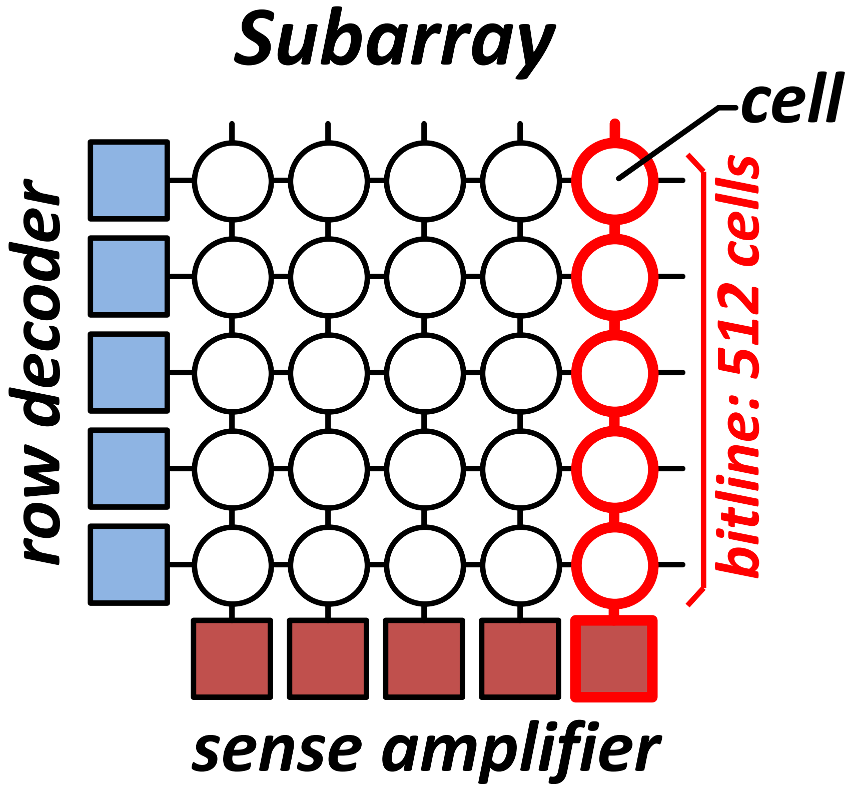
What Causes the Long Latency?



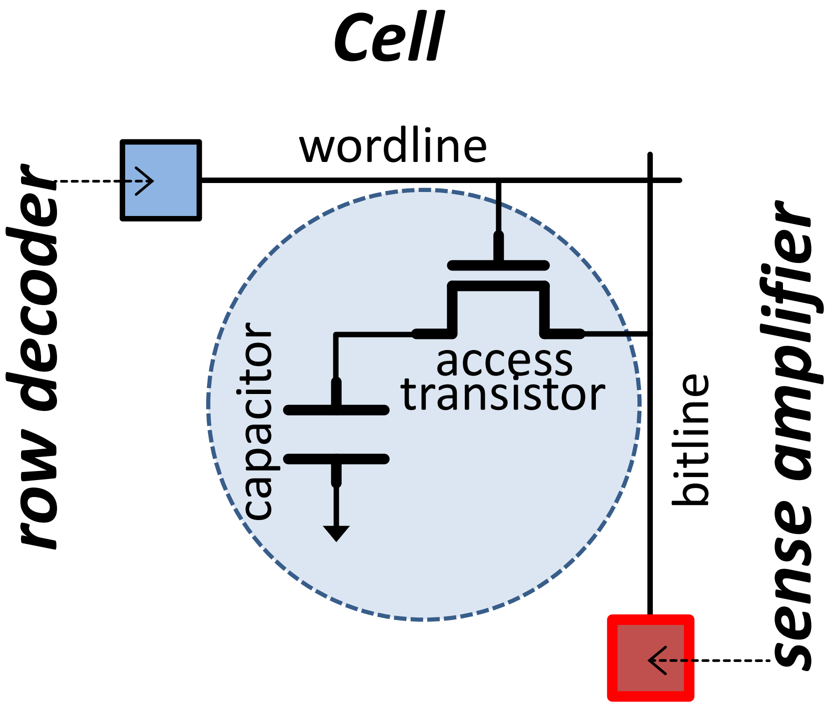
DRAM Latency = Subarray Latency + I/O Latency

Dominant

Why is the Subarray So Slow?



Subarray

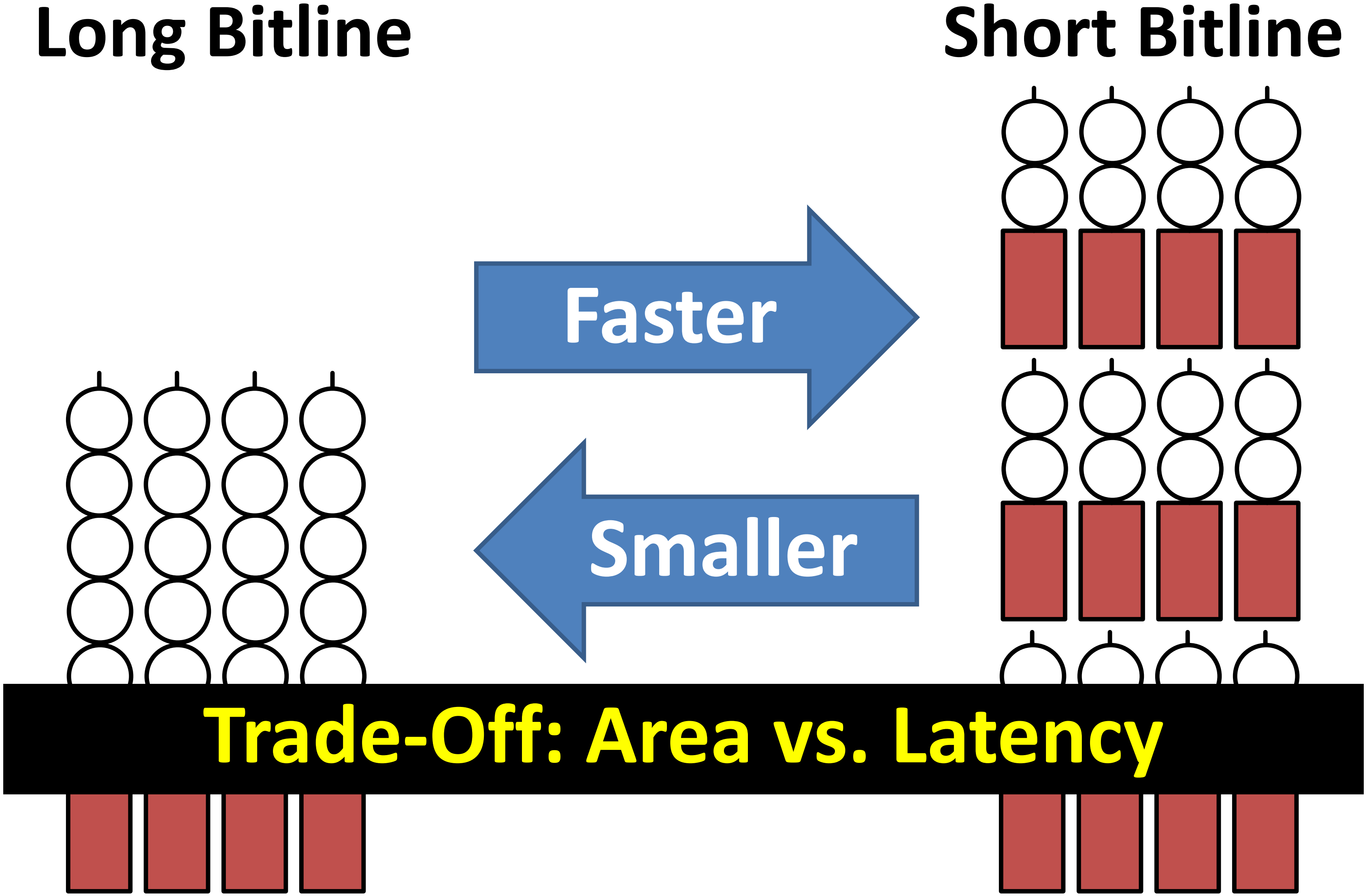


Cell

large sense amplifier

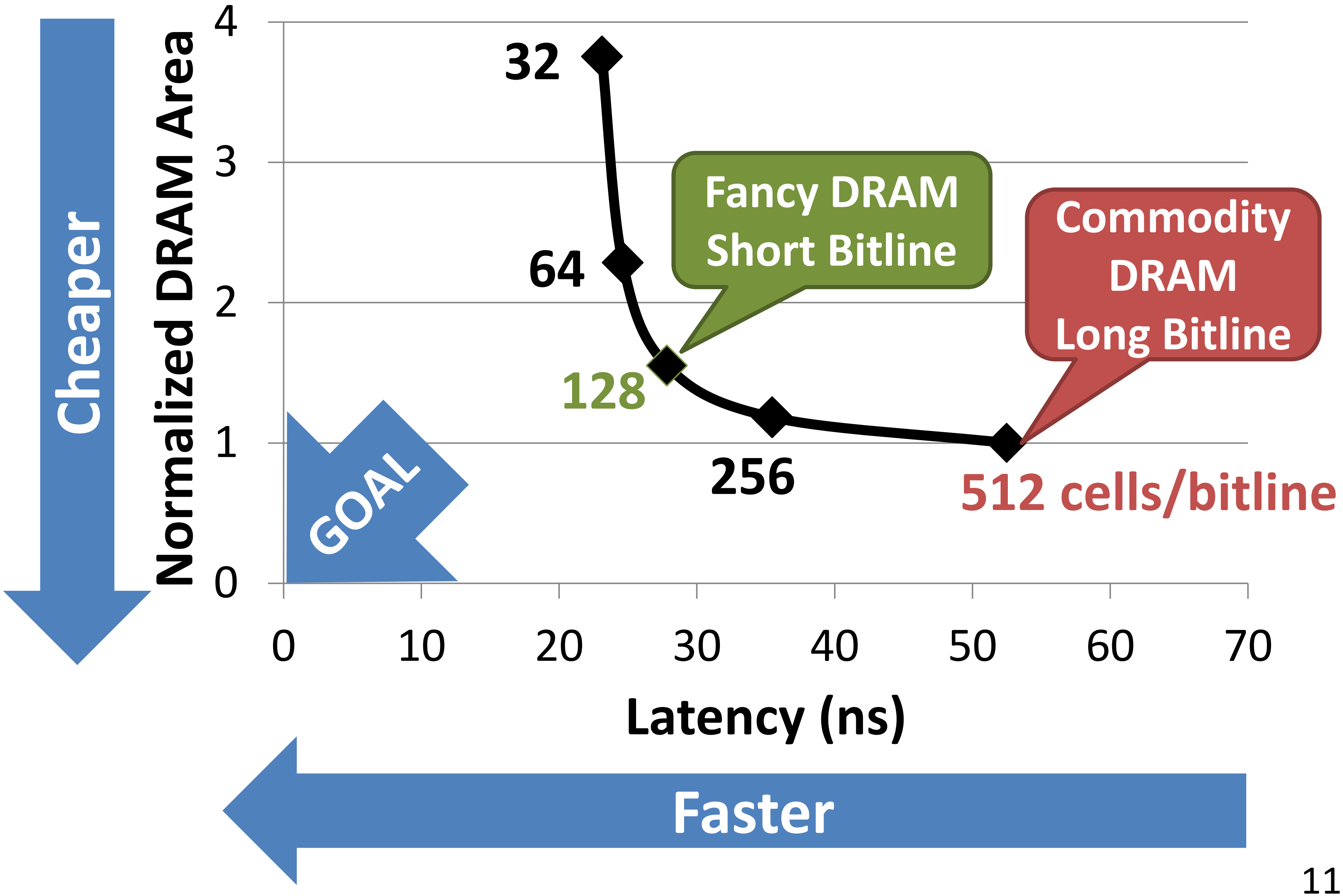
- Long bitline
  - Amortizes sense amplifier cost → Small area
  - Large bitline capacitance → High latency & power

Trade-Off: Area (Die Size) vs. Latency



Trade-Off: Area vs. Latency

Trade-Off: Area (Die Size) vs. Latency



Approximating the Best of Both Worlds

Long Bitline

Our Proposal

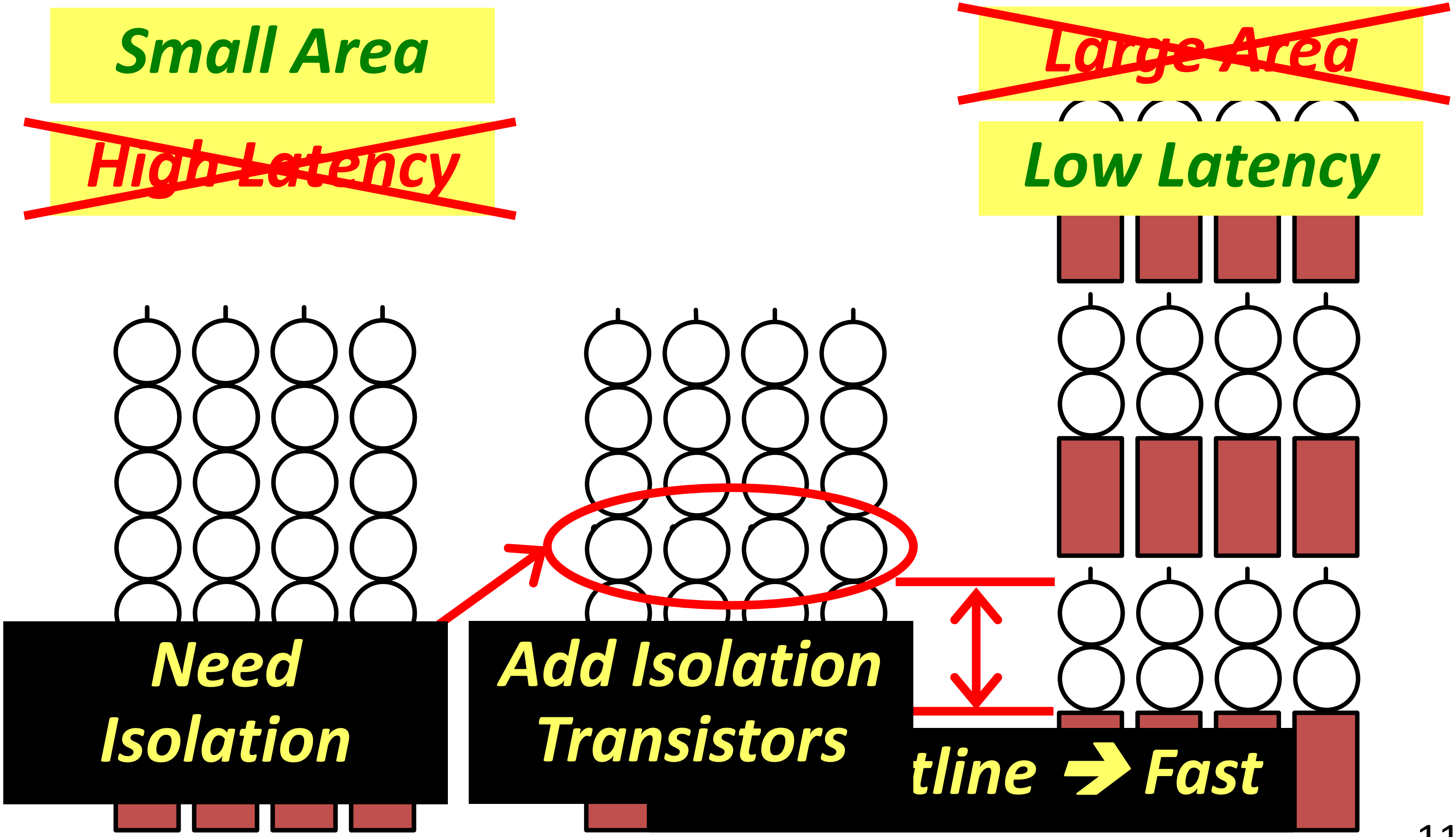
Short Bitline

Small Area

~~High Latency~~

~~Large Area~~

Low Latency



Need Isolation

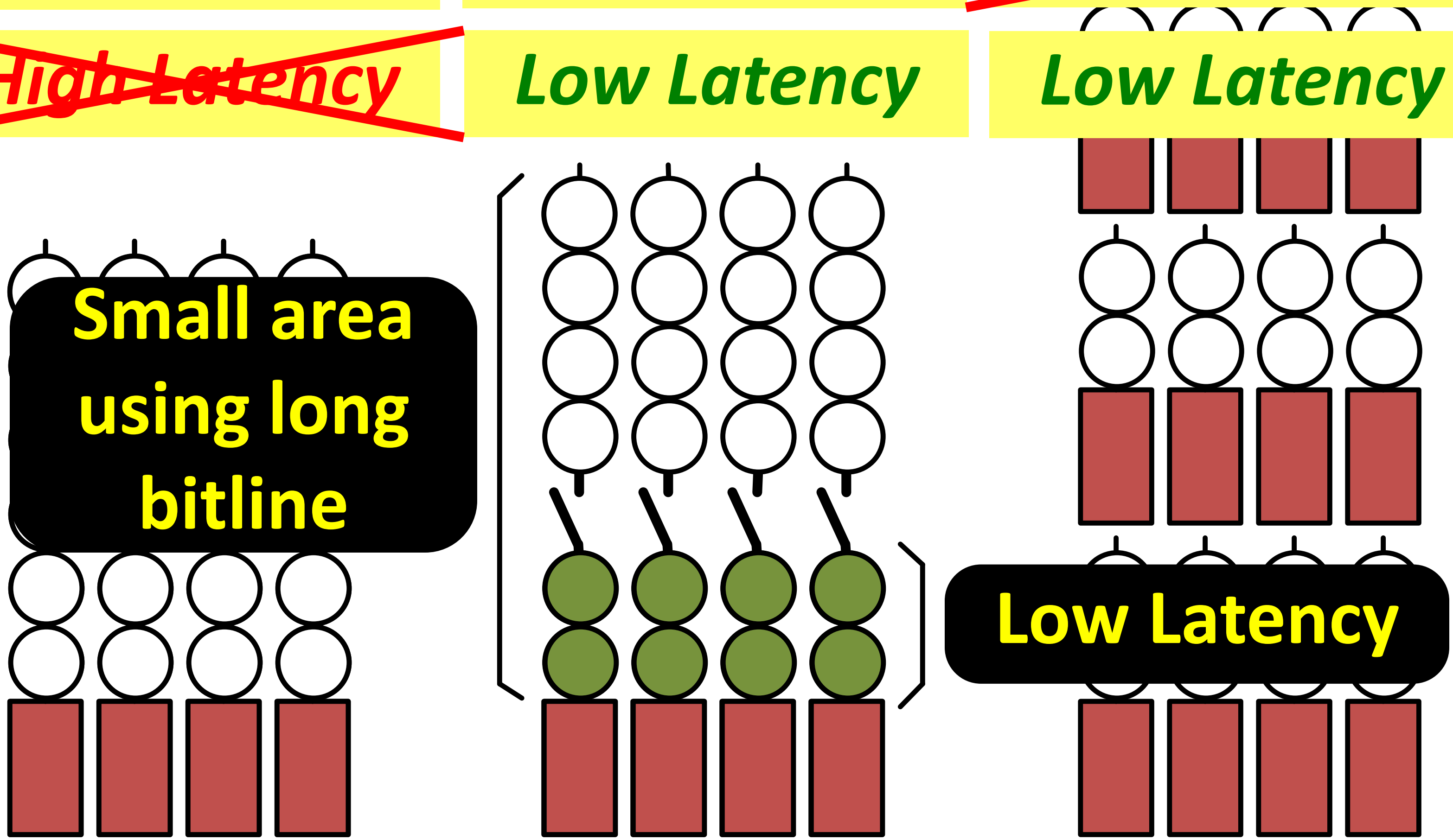
Add Isolation Transistors

tline → Fast

Approximating the Best of Both Worlds

Long Bitlin Tiered-Latency DRAM ıort Bitline

Small Area	Small Area	~~Large Area~~
~~High Latency~~	Low Latency	Low Latency



Small area using long bitline

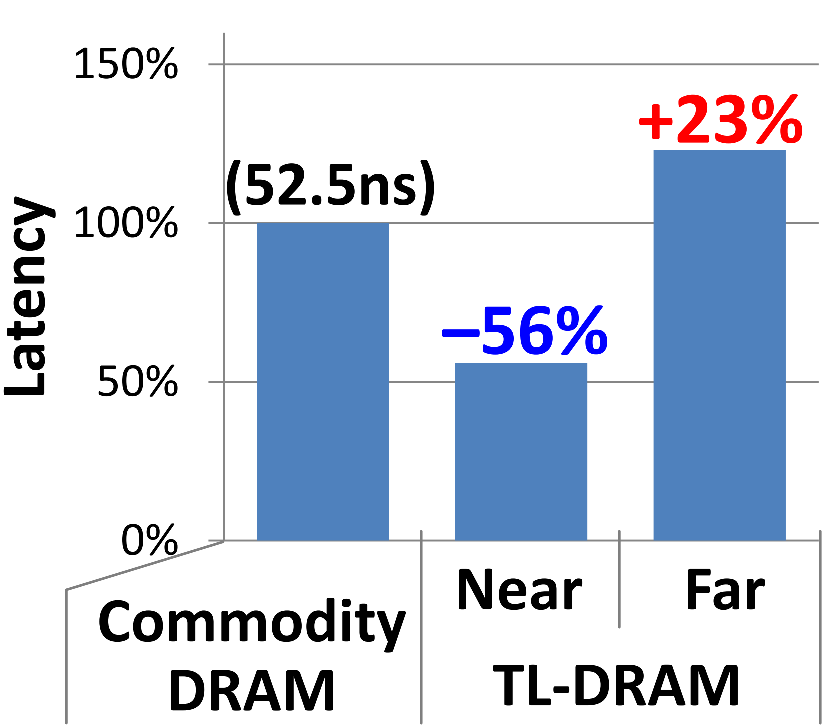
Low Latency

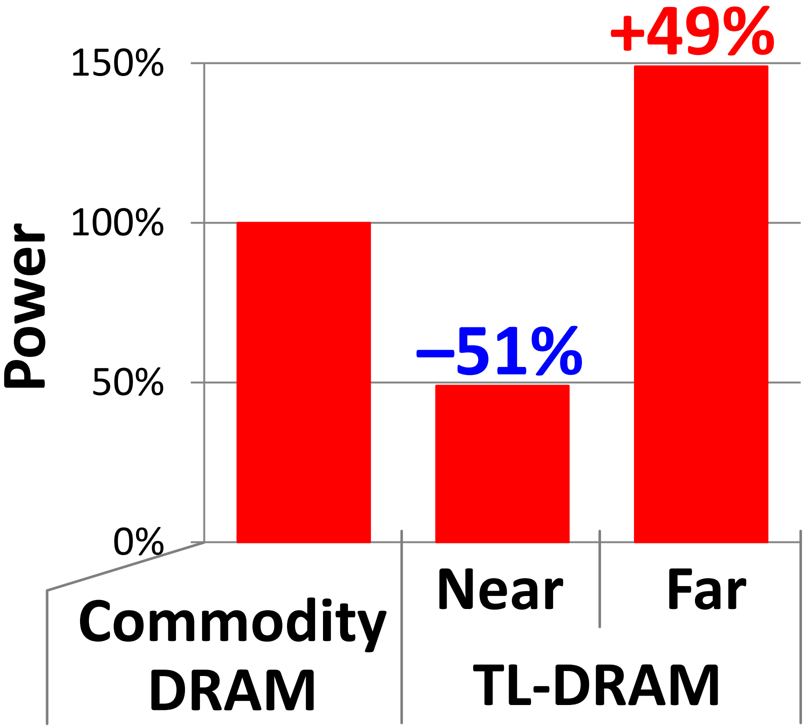
Latency, Power, and Area Evaluation

- **Commodity DRAM:** 512 cells/bitline
- **TL-DRAM:** 512 cells/bitline
  - Near segment: 32 cells
  - Far segment: 480 cells
- **Latency Evaluation**
  - SPICE simulation using circuit-level DRAM model
- **Power and Area Evaluation**
  - DRAM area/power simulator from Rambus
  - DDR3 energy calculator from Micron

Commodity DRAM vs. TL-DRAM [HPCA 2013]

- **DRAM Latency** (tRC)
- **DRAM Power**

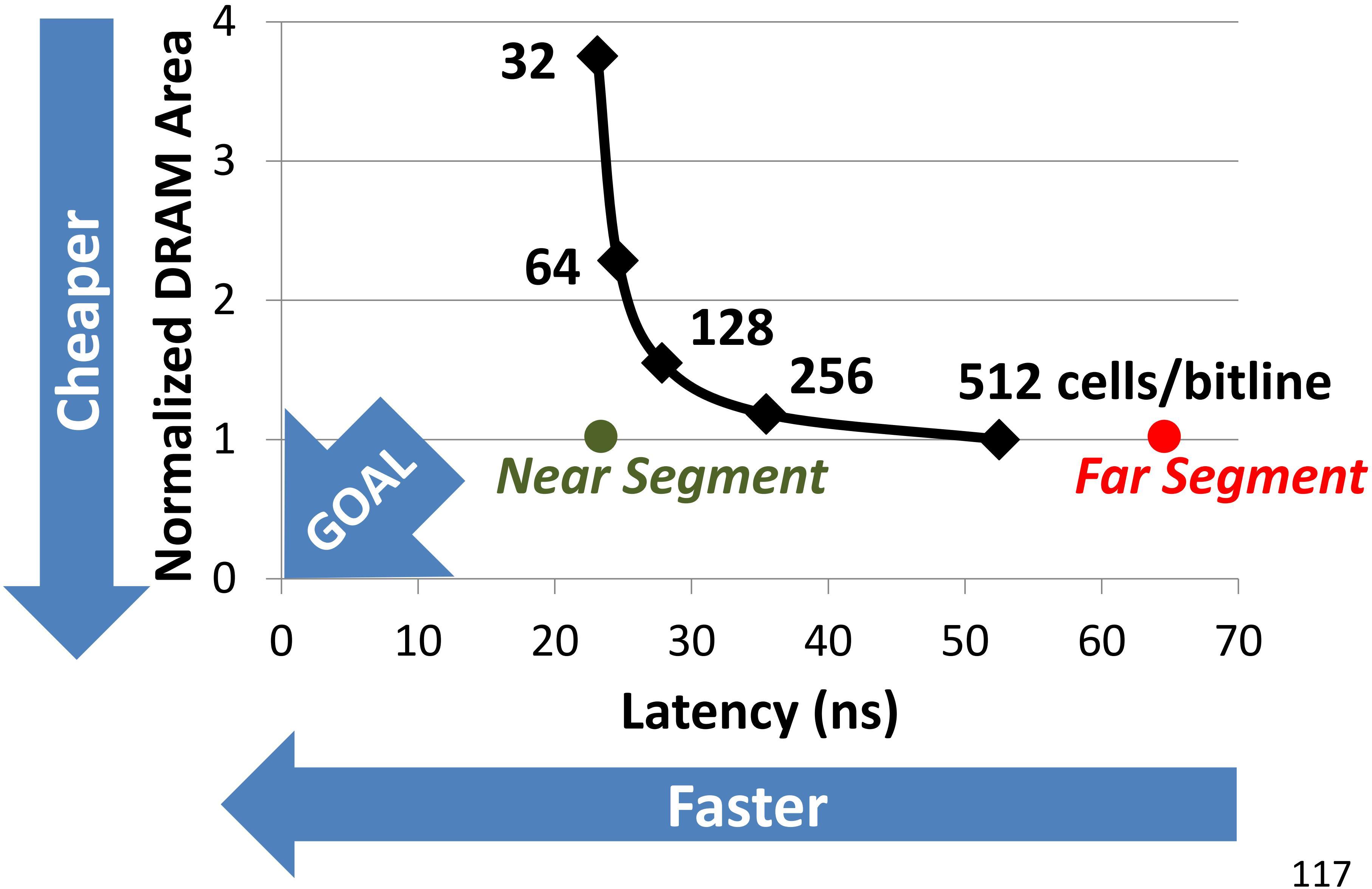




- **DRAM Area Overhead**

  ~3%: mainly due to the isolation transistors

Trade-Off: Area (Die-Area) vs. Latency



Leveraging Tiered-Latency DRAM

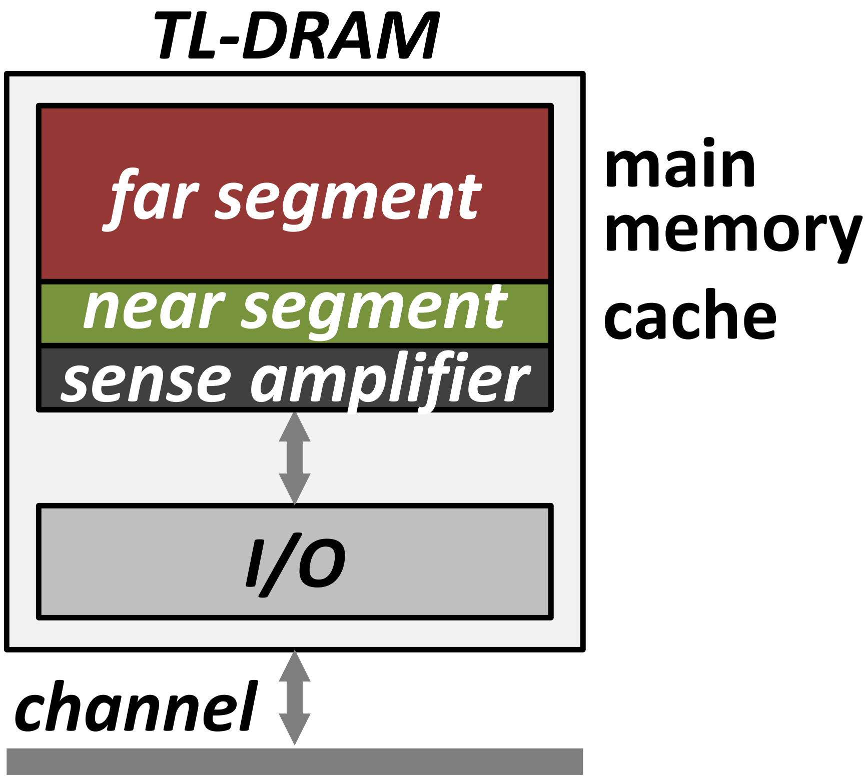
- TL-DRAM is a ***substrate*** that can be leveraged by the hardware and/or software

- Many potential uses

1. Use near segment as hardware-managed ***inclusive*** cache to far segment
2. Use near segment as hardware-managed ***exclusive*** cache to far segment
3. Profile-based page mapping by operating system
4. Simply replace DRAM with TL-DRAM

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

Near Segment as Hardware-Managed Cache

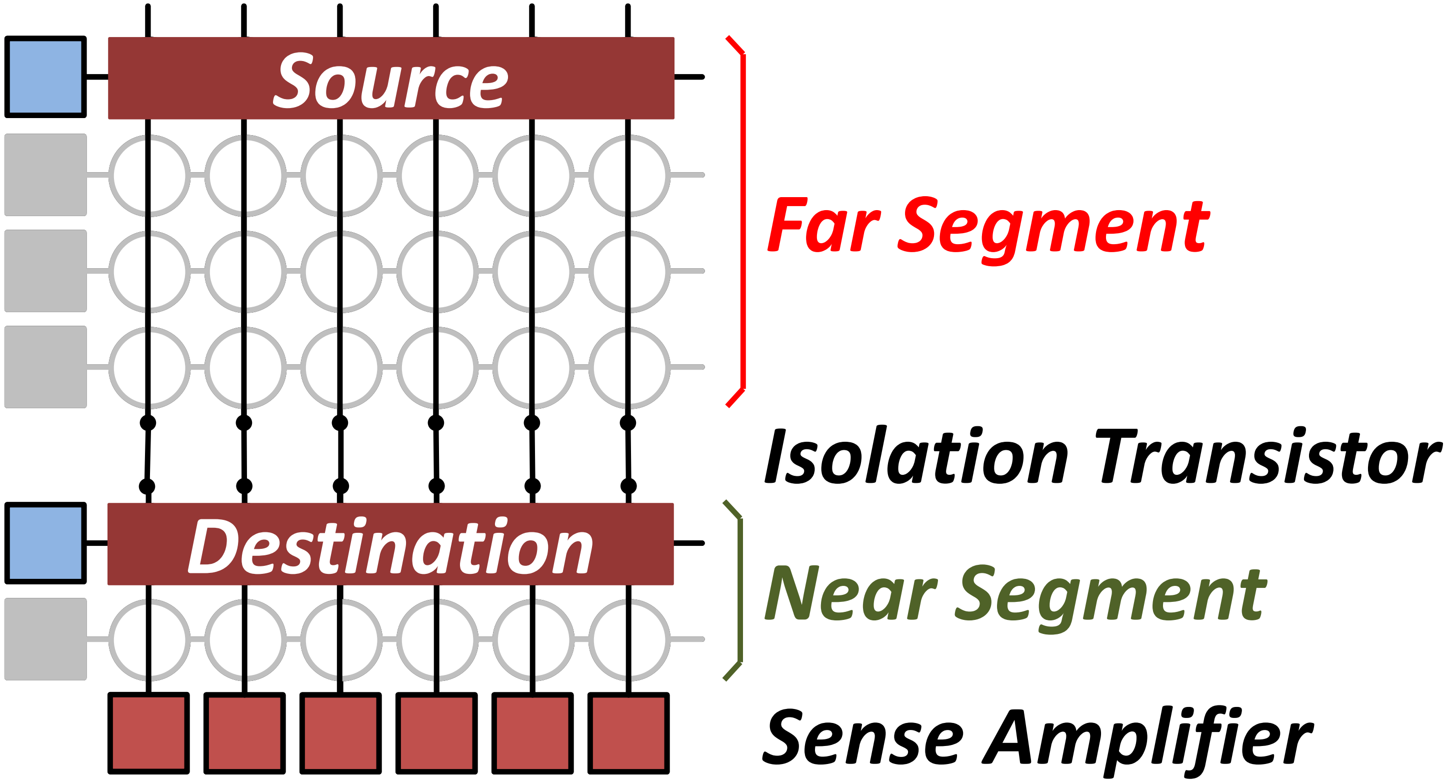


- **Challenge 1:** How to efficiently migrate a row between segments?
- **Challenge 2:** How to efficiently manage the cache?

Inter-Segment Migration

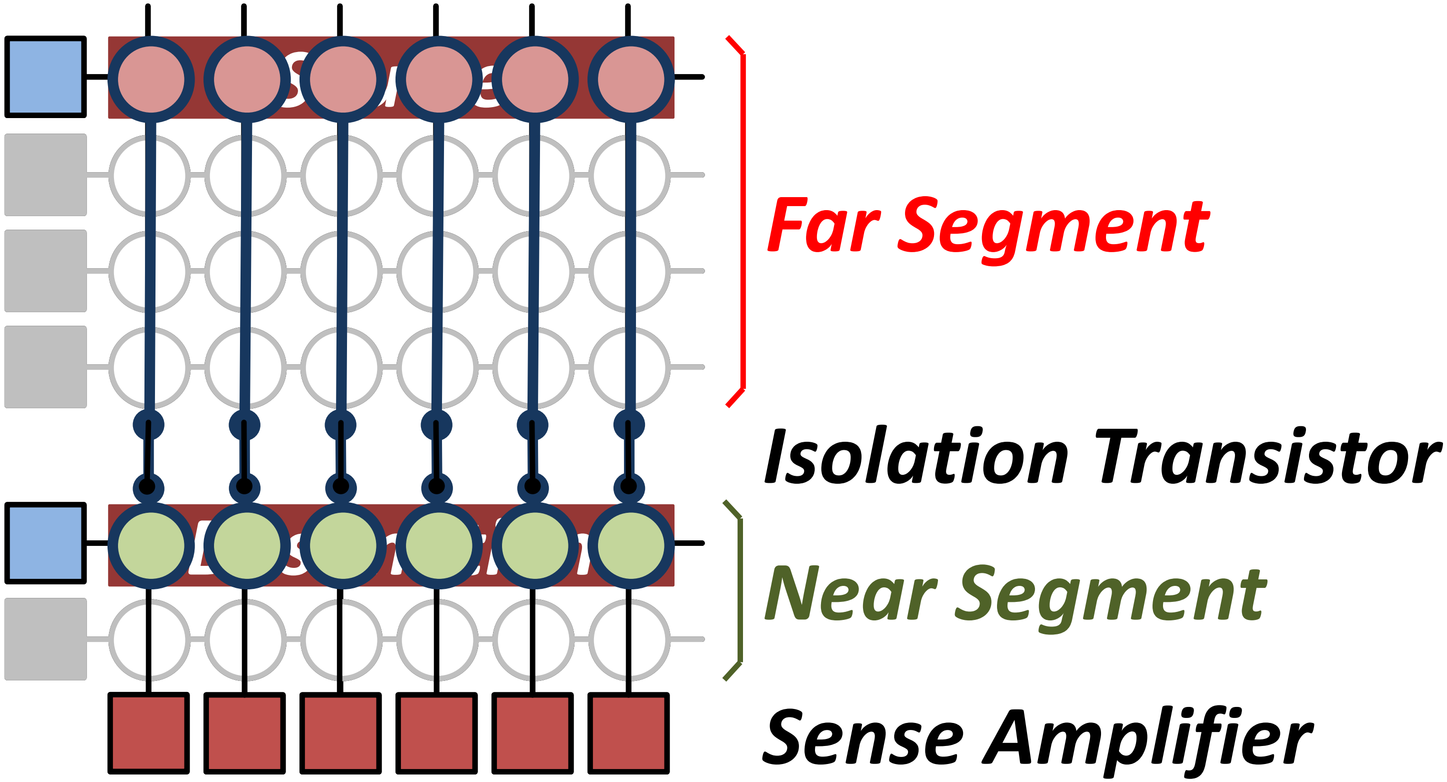
- **Goal:** Migrate source row into destination row
- **Naïve way:** Memory controller reads the source row *byte by byte* and writes to destination row *byte by byte*

→ ***High latency***



Inter-Segment Migration

- **Our way:**
  - Source and destination cells ***share bitlines***
  - Transfer data from source to destination across ***shared bitlines*** concurrently

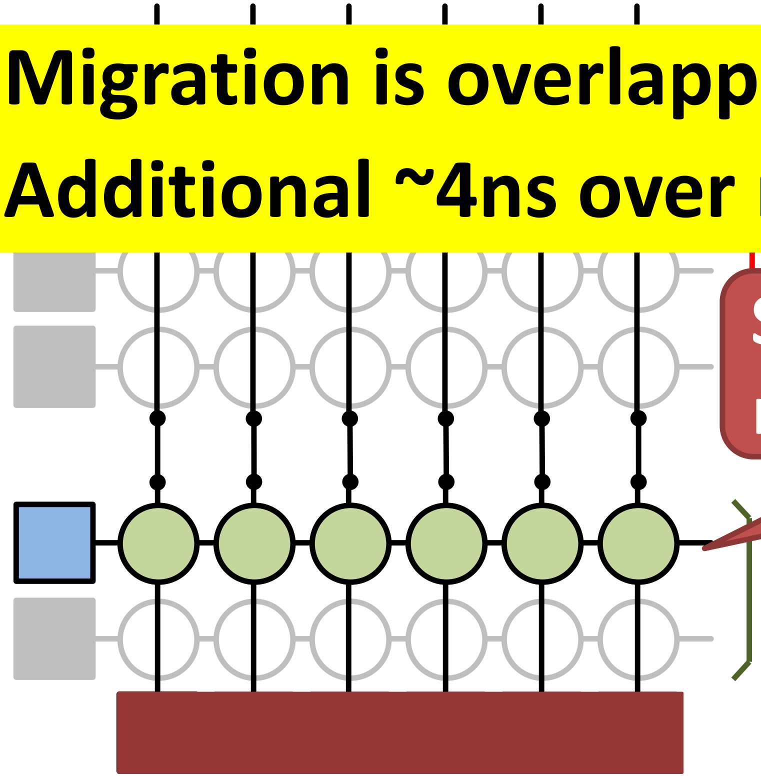


Inter-Segment Migration

- **Our way:**
  - Source and destination cells ***share bitlines***
  - Transfer data from sou
    shared bitlines concu

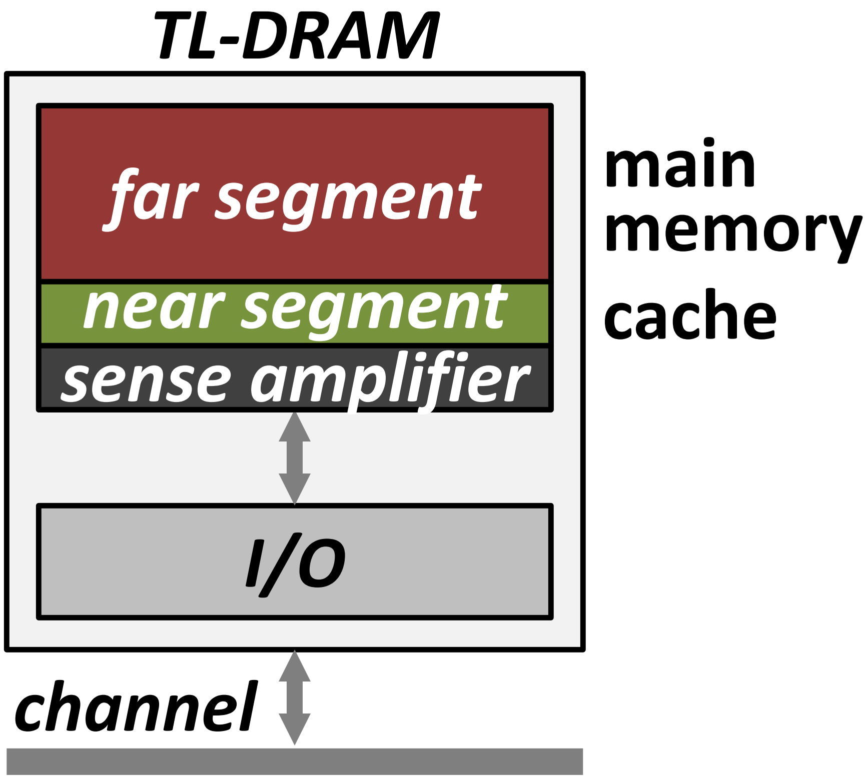
Step 1: Activate source row

Migration is overlapped with source row access
Additional ~4ns over row access latency



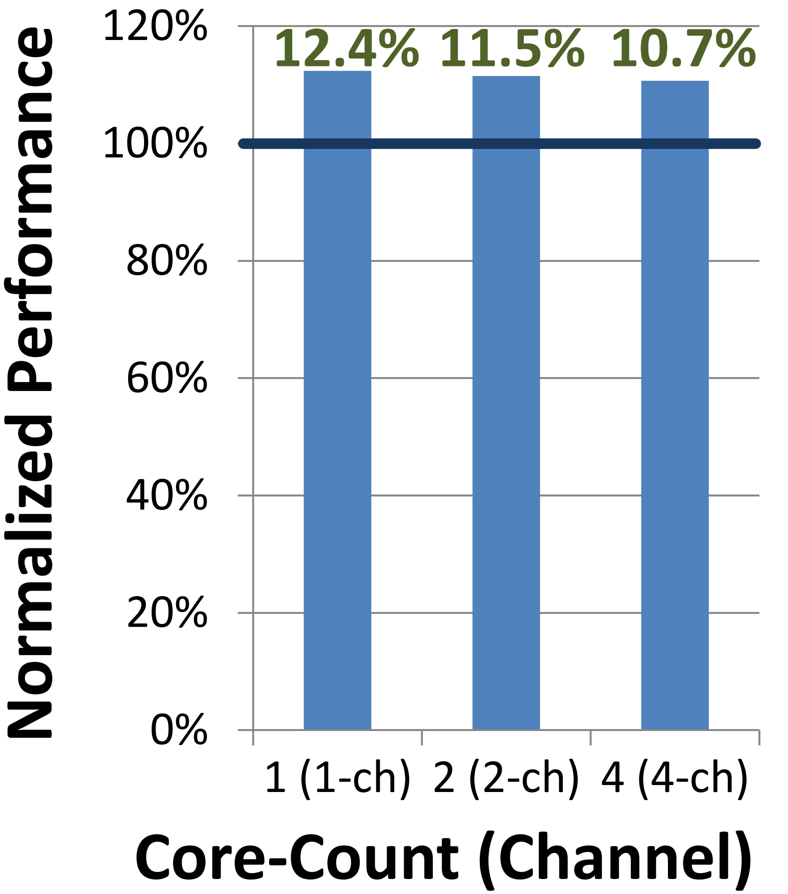
Step 2: Activate destination row to connect cell and bitline

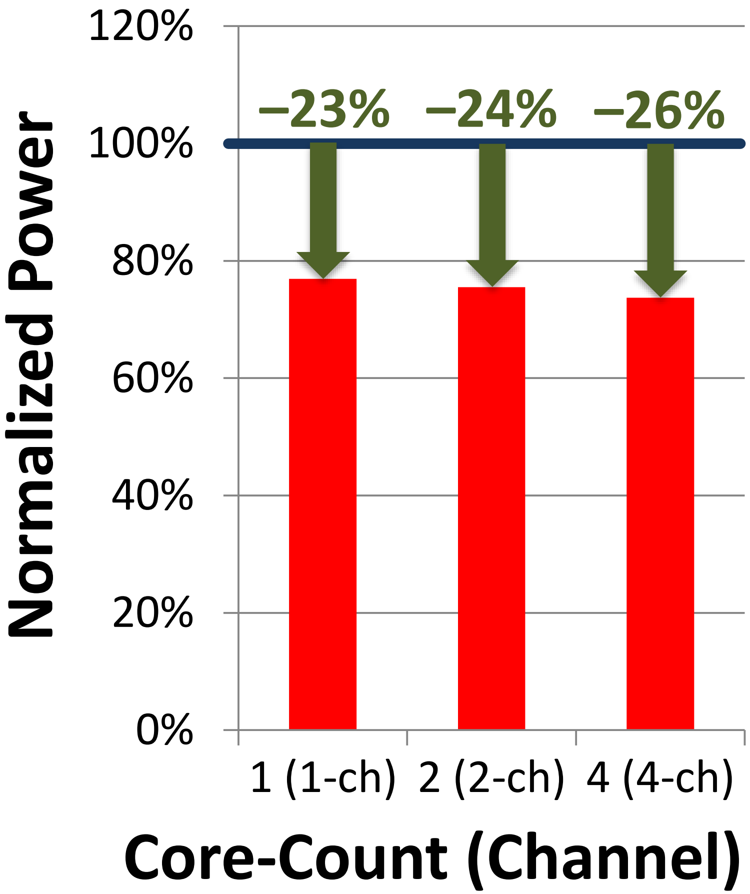
Near Segment as Hardware-Managed Cache



- **Challenge 1:** How to efficiently migrate a row between segments?
- **Challenge 2:** How to efficiently manage the cache?

Performance & Power Consumption

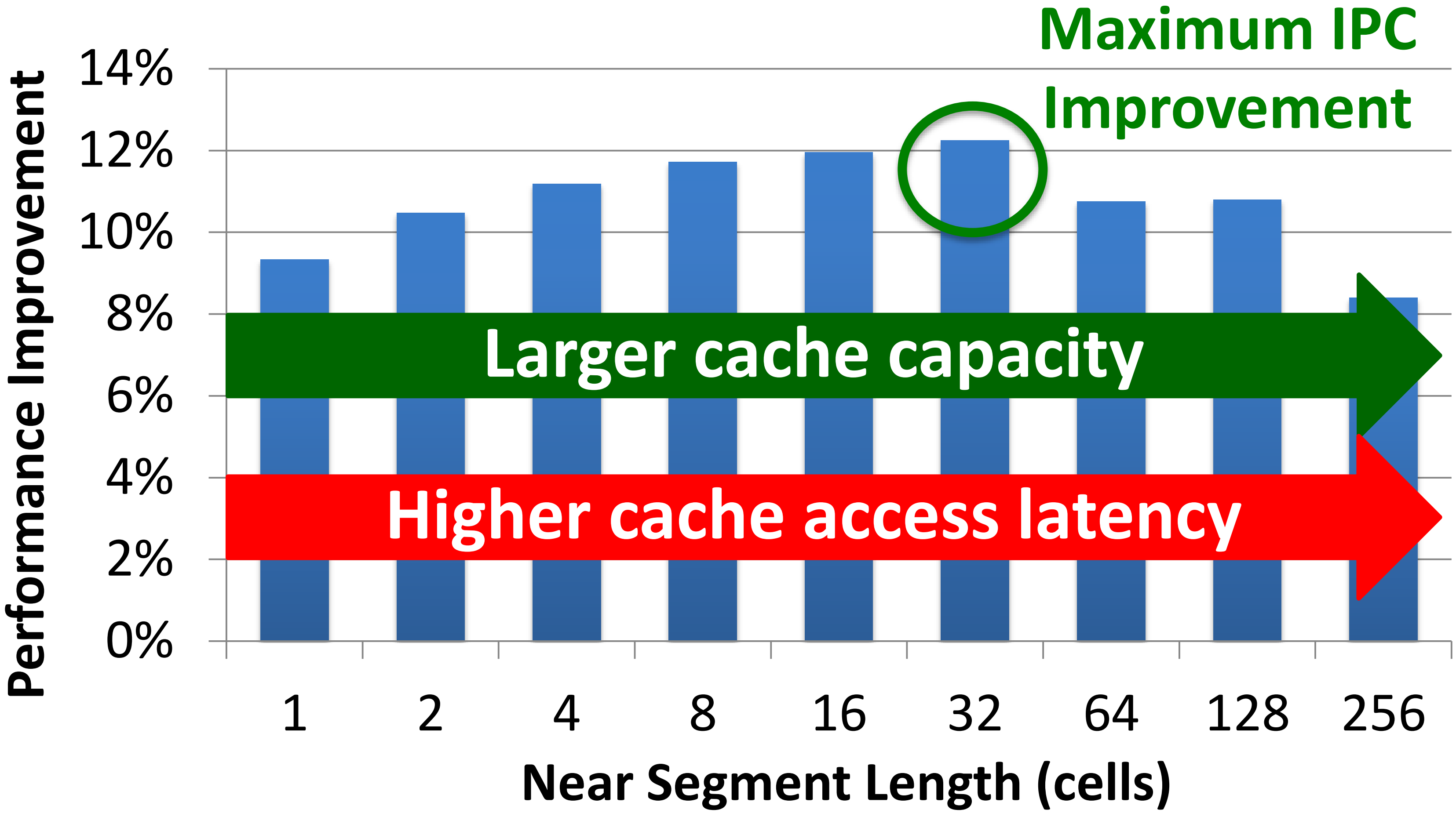




Using near segment as a cache improves performance and reduces power consumption

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

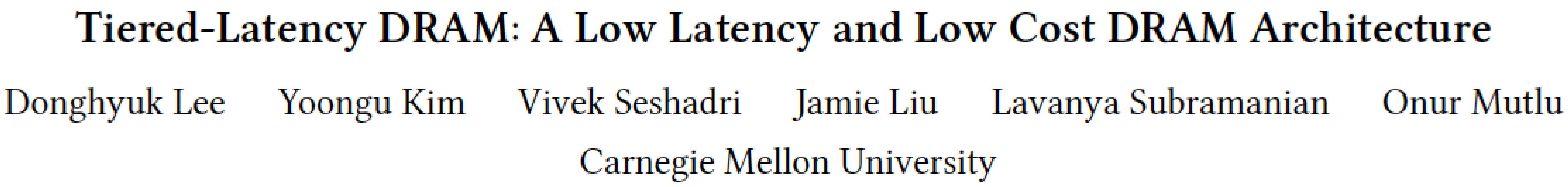
Single-Core: Varying Near Segment Length



By adjusting the near segment length, we can trade off cache capacity for cache latency

More on TL-DRAM

- Donghyuk Lee, Yoongu Kim, Vivek Seshadri, Jamie Liu, Lavanya Subramanian, and Onur Mutlu, **"Tiered-Latency DRAM: A Low Latency and Low Cost DRAM Architecture"** *Proceedings of the 19th International Symposium on High-Performance Computer Architecture (**HPCA**)*, Shenzhen, China, February 2013. Slides (pptx)



Tiered-Latency DRAM: A Low Latency and Low Cost DRAM Architecture

Donghyuk Lee Yoongu Kim Vivek Seshadri Jamie Liu Lavanya Subramanian Onur Mutlu

Carnegie Mellon University

We did not cover the following slides in lecture.
These are for your preparation for the next lecture.

Computer Architecture

Lecture 5: DRAM Operation, Memory Control & Memory Latency

Prof. Onur Mutlu

ETH Zürich

Fall 2017

4 October 2017

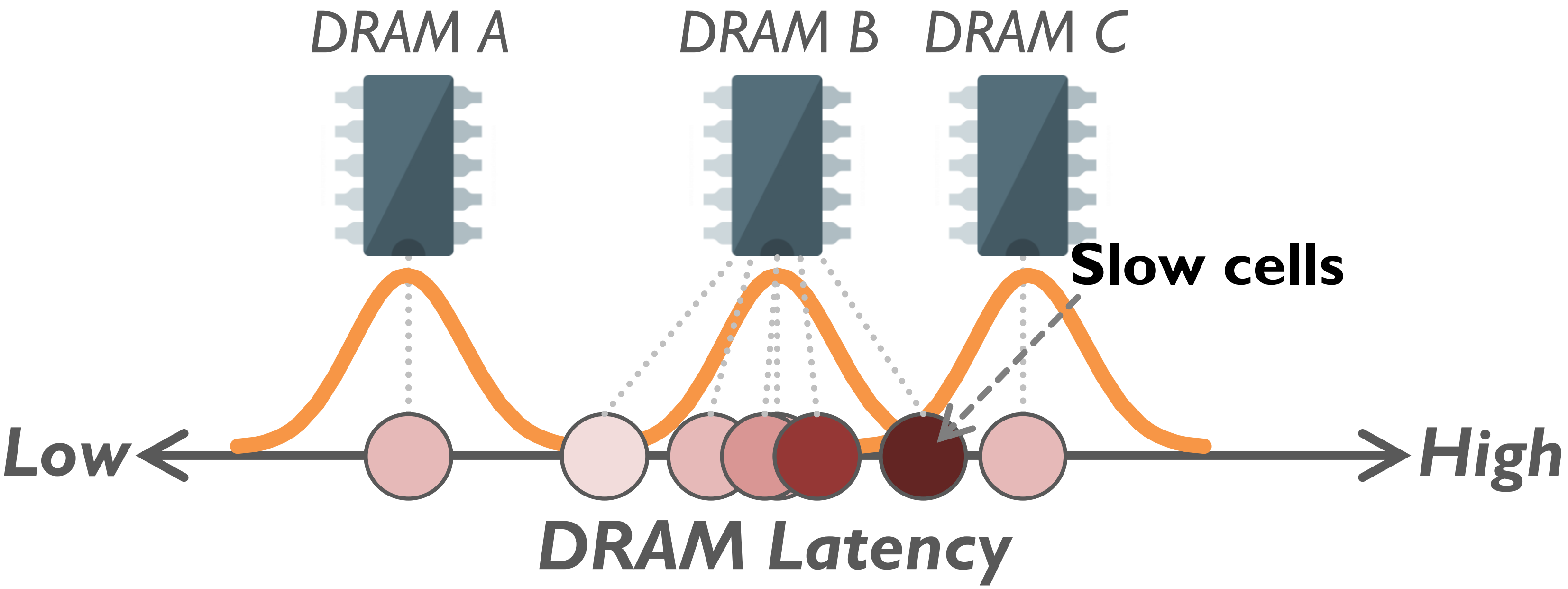
Why the Long Latency?

- Design of DRAM uArchitecture
  - Goal: Maximize capacity/area, not minimize latency

- "One size fits all" approach to latency specification
  - Same latency parameters for all temperatures
  - Same latency parameters for all DRAM chips (e.g., rows)
  - Same latency parameters for all parts of a DRAM chip
  - Same latency parameters for all supply voltage levels
  - Same latency parameters for all application data
  - …

Latency Variation in Memory Chips

Heterogeneous manufacturing & operating conditions → latency variation in timing parameters



What Else Causes the Long Memory Latency?

- **Conservative timing margins!**

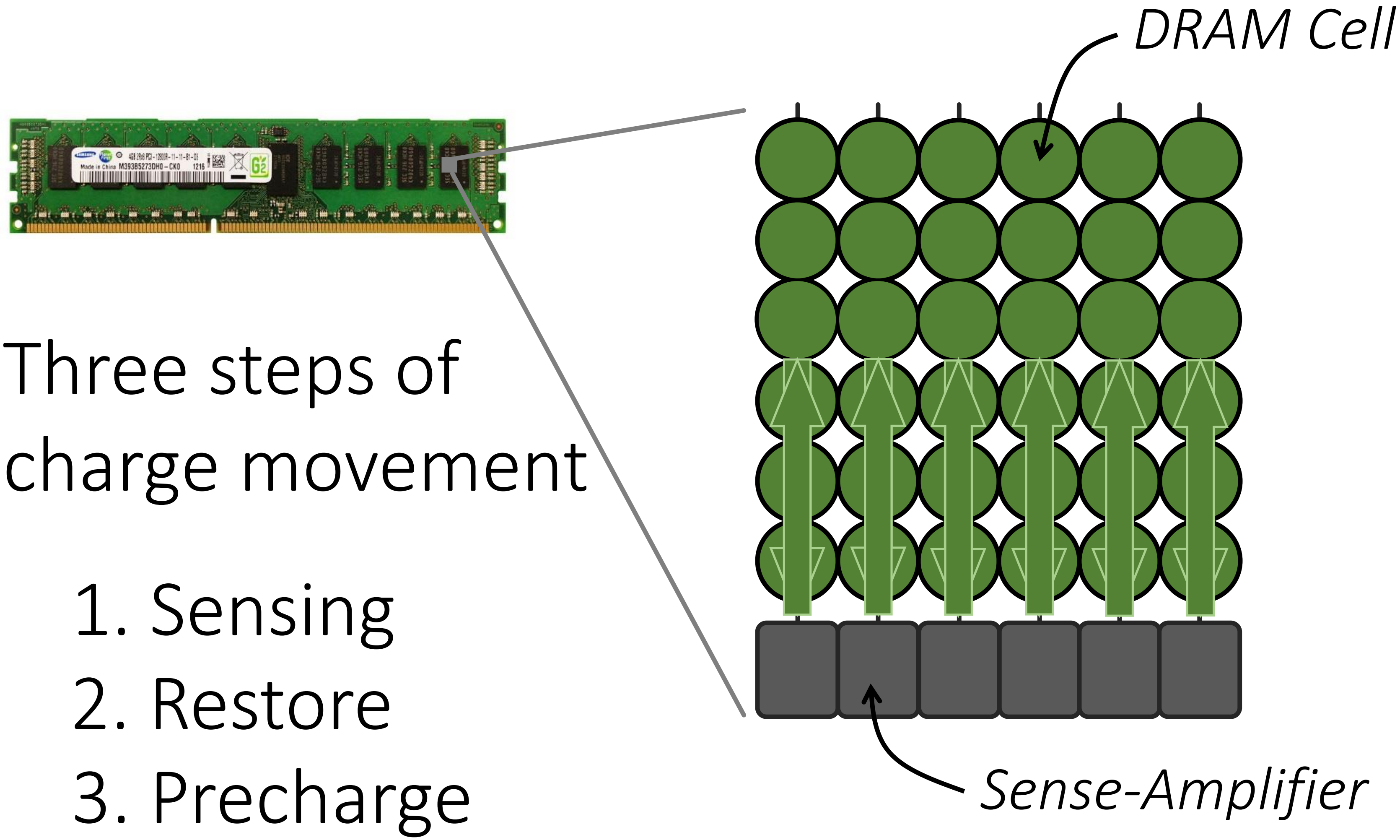
- DRAM timing parameters are set to cover the worst case

- Worst-case temperatures
  - 85 degrees vs. common-case
  - to enable a wide range of operating conditions
- Worst-case devices
  - DRAM cell with smallest charge across any acceptable device
  - to tolerate process variation at acceptable yield

- This leads to large timing margins for the common case

Understanding and Exploiting Variation in DRAM Latency

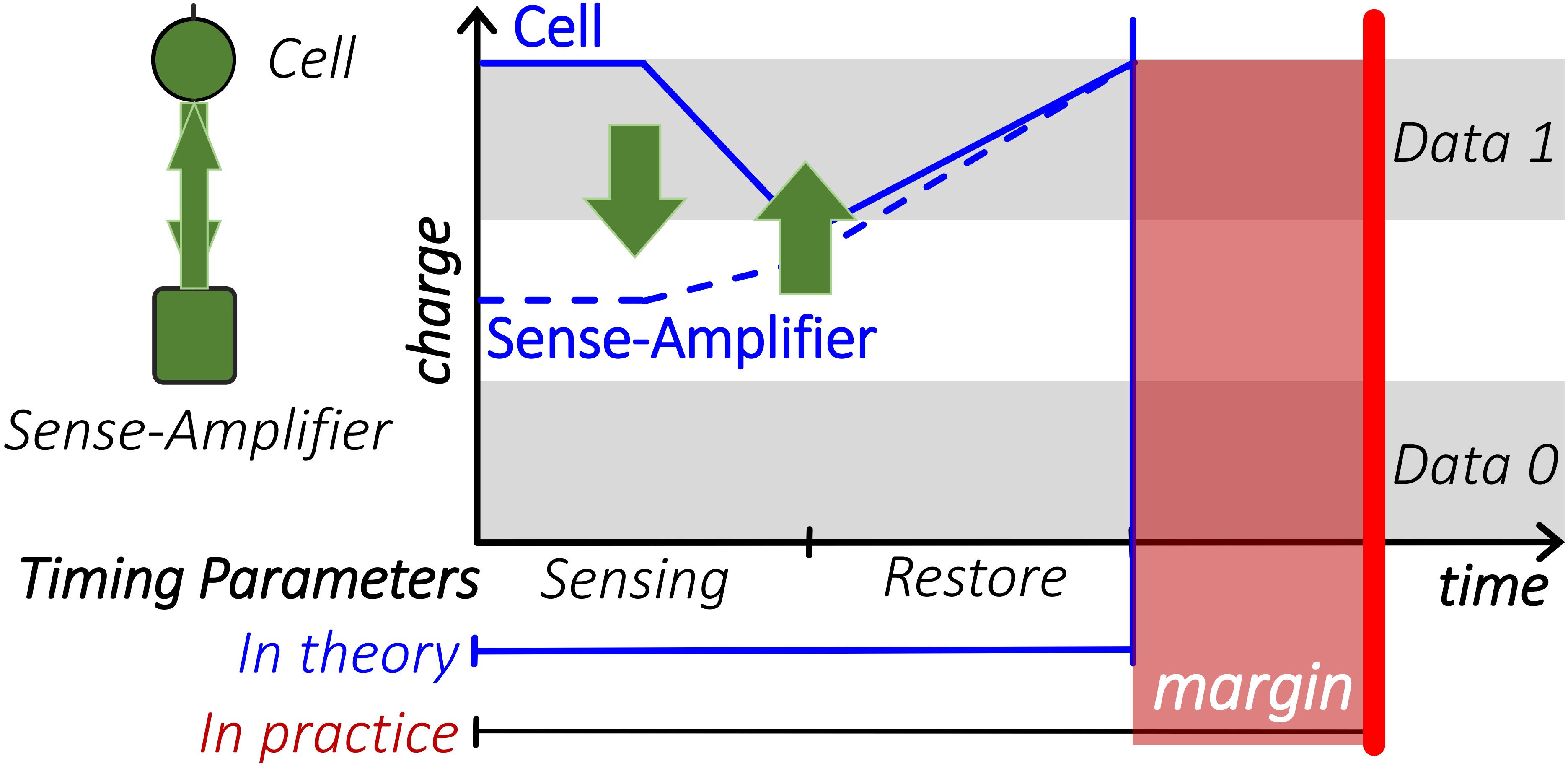
DRAM Stores Data as Charge



Three steps of charge movement

1. Sensing
2. Restore
3. Precharge

DRAM Charge over Time



Why does DRAM need the extra timing margin?

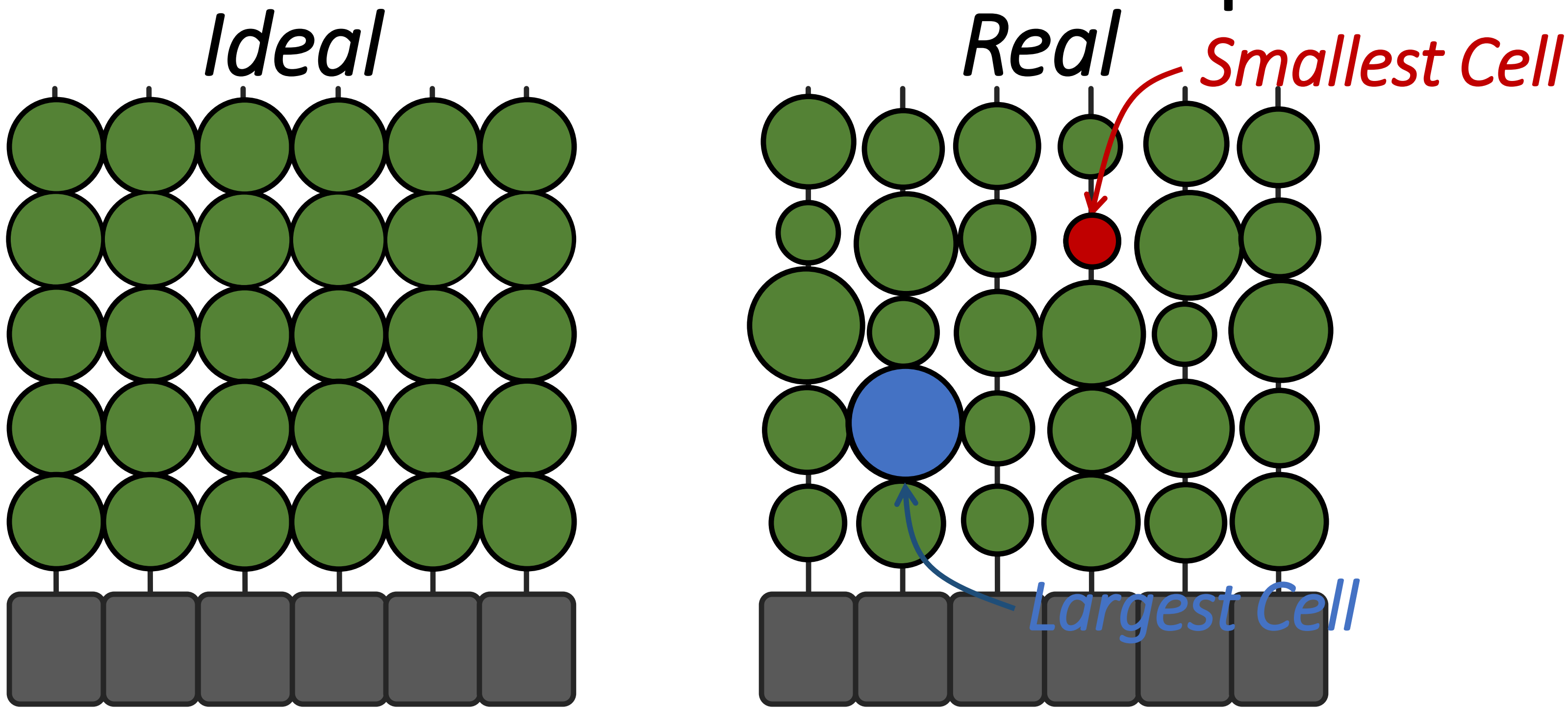
Two Reasons for Timing Margin

1. Process Variation

- DRAM cells are not equal
- Leads to extra timing margin for a cell that can store a large amount of charge

2. Temperature Dependence

DRAM Cells are Not Equal

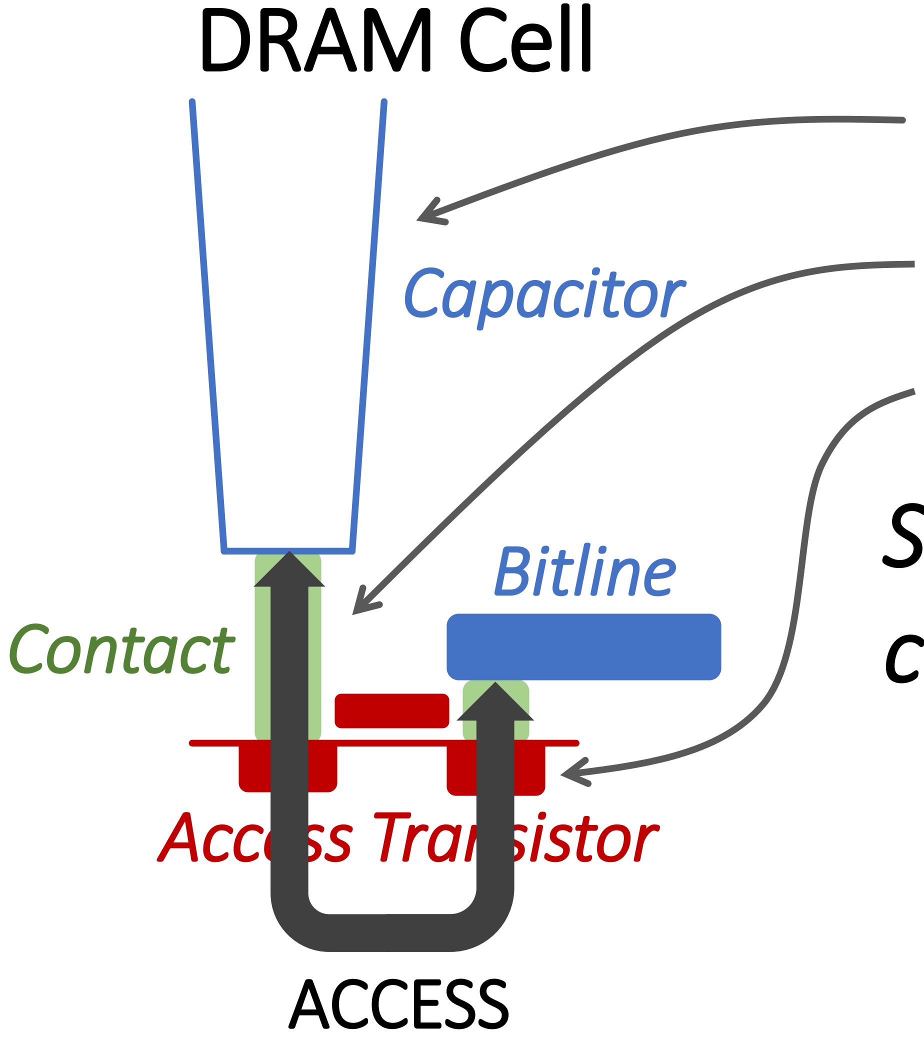


Same Size →
Same Charge →
Same Latency

Different Size →
Different Charge →
Different Latency

Large variation in cell size →
Large variation in charge →
Large variation in access latency

Process Variation



❶ Cell Capacitance

❷ Contact Resistance

❸ Transistor Performance

Small cell can store small charge

- *Small cell capacitance*
- *High contact resistance*
- *Slow access transistor*

➔ ***High access latency***

Two Reasons for Timing Margin

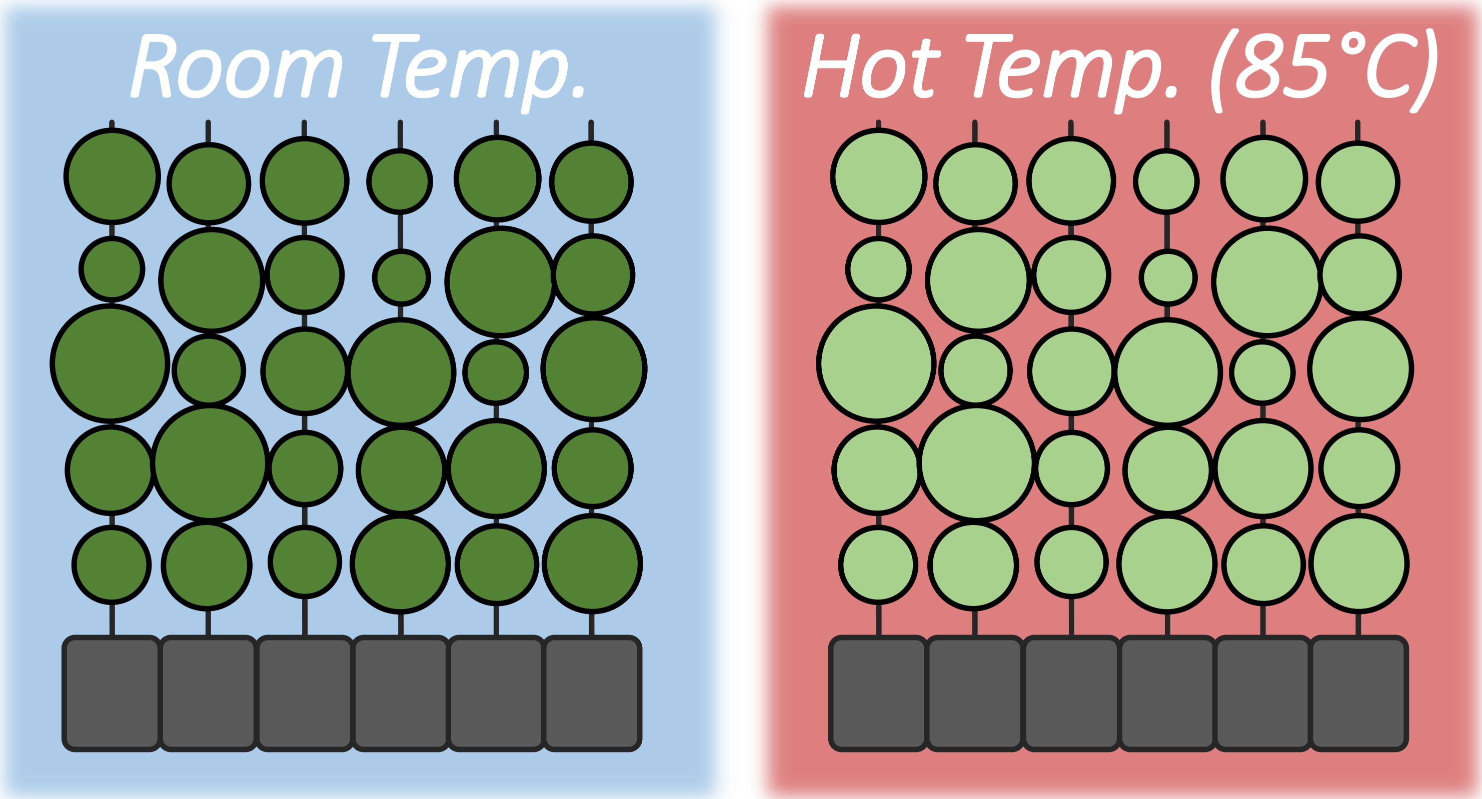
1. Process Variation

- DRAM cells are not equal
- Leads to extra timing margin for a cell that can store a large amount of charge

2. Temperature Dependence

- DRAM leaks more charge at higher temperature
- Leads to extra timing margin for cells that operate at low temperature

Charge Leakage Temperature



Small Leakage

Large Leakage

Cells store small charge at high temperature and large charge at low temperature
→ Large variation in access latency

DRAM Timing Parameters

- *DRAM timing parameters are dictated by* ***the worst-case***
  - The smallest cell with the smallest charge **in all DRAM products**
  - Operating at **the highest temperature**

- ***Large timing margin for the common-case***

Adaptive-Latency DRAM **[HPCA 2015]**

- Idea: Optimize DRAM timing for the common case
  - Current temperature
  - Current DRAM module

- Why would this reduce latency?

  - A DRAM cell can store much more charge in the common case (low temperature, strong cell) than in the worst case

  - More charge in a DRAM cell
    → Faster sensing, charge restoration, precharging
    → Faster access (read, write, refresh, …)

Lee+, "Adaptive-Latency DRAM: Optimizing DRAM Timing for the Common-Case," HPCA 2015.

Extra Charge → Reduced Latency

1. Sensing

   Sense **cells with extra charge** faster

   → Lower sensing latency

2. Restore

   No need to fully restore **cells with extra charge**

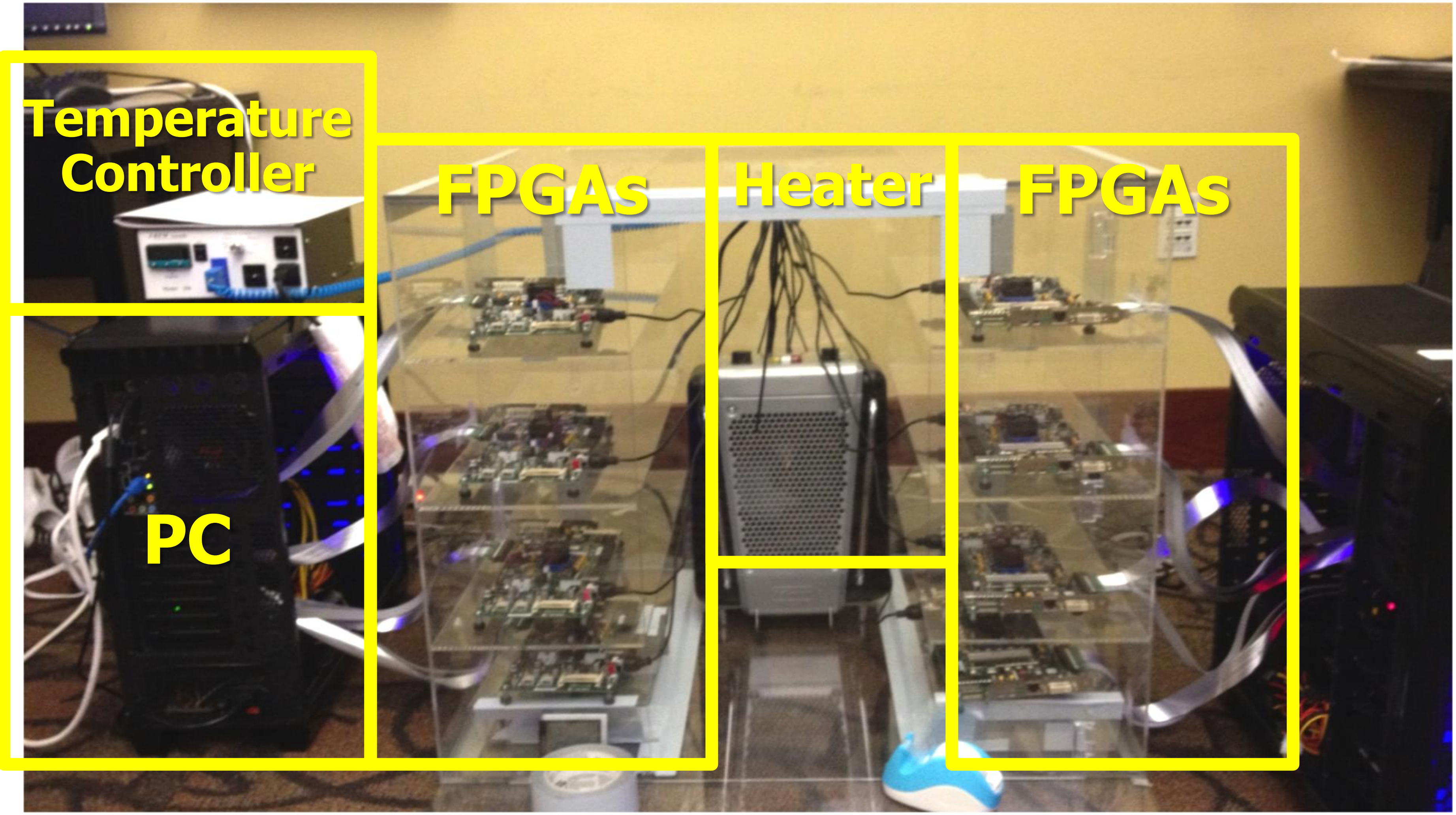
   → Lower restoration latency

3. Precharge

   No need to fully precharge bitlines for **cells with extra charge**

   → Lower precharge latency

DRAM Characterization Infrastructure



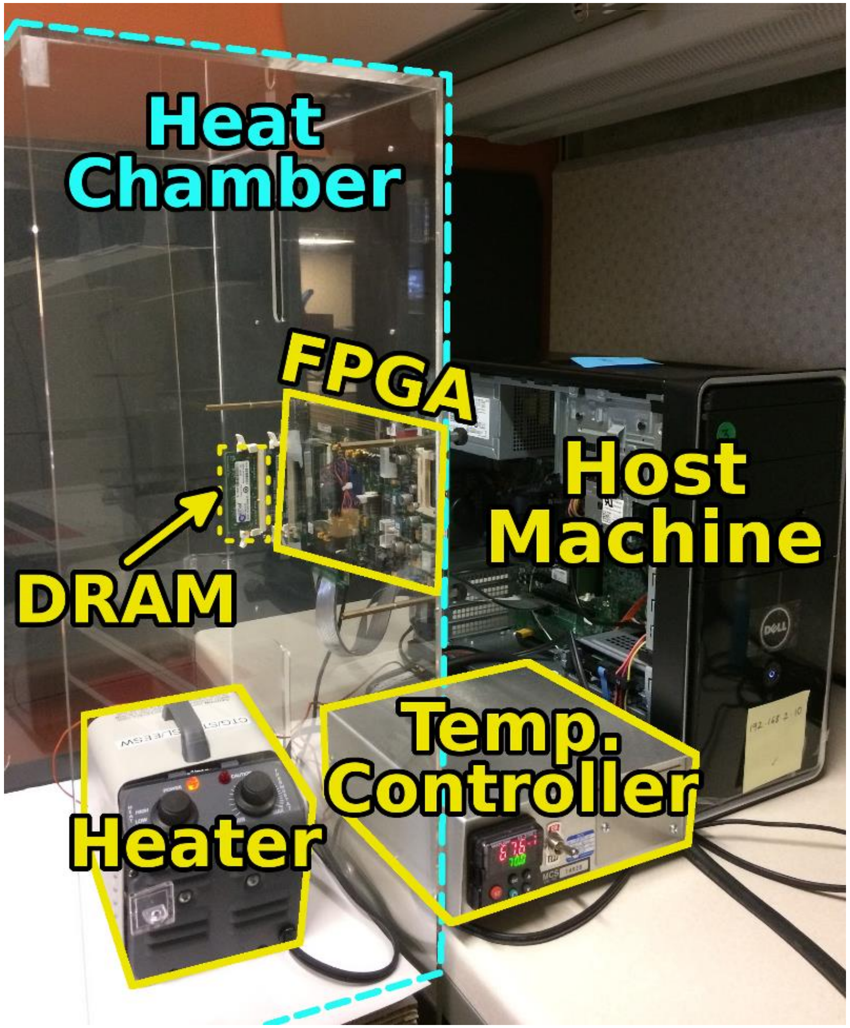
Kim+, "Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors," ISCA 2014.

DRAM Characterization Infrastructure

- Hasan Hassan et al., **SoftMC: A Flexible and Practical Open-Source Infrastructure for Enabling Experimental DRAM Studies**, HPCA 2017.

- **Flexible**
- **Easy to Use (C++ API)**
- **Open-source**

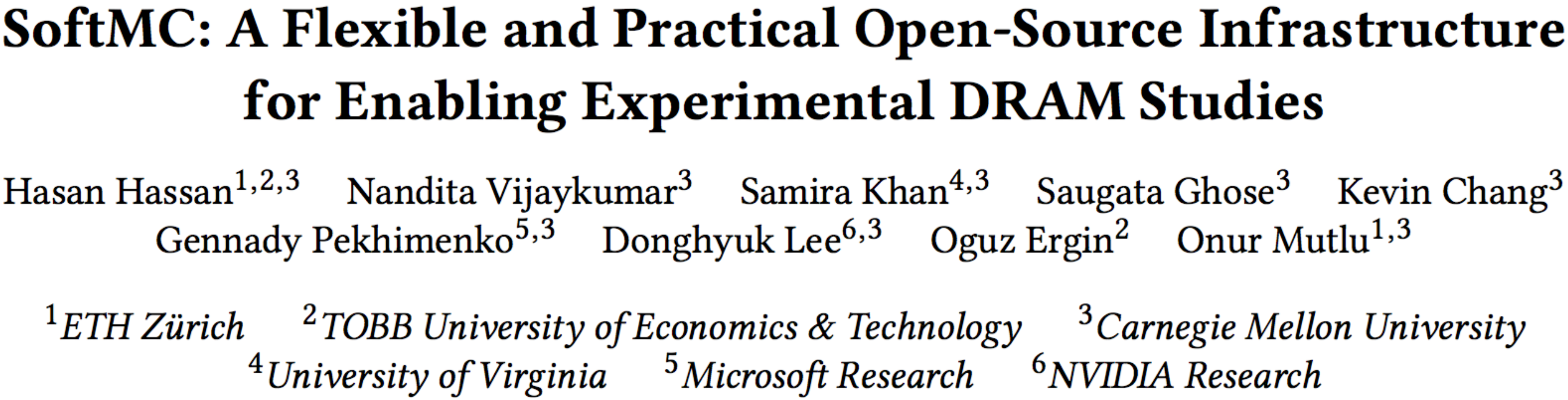
github.com/CMU-SAFARI/SoftMC



SAFARI

SoftMC: Open Source DRAM Infrastructure

- https://github.com/CMU-SAFARI/SoftMC



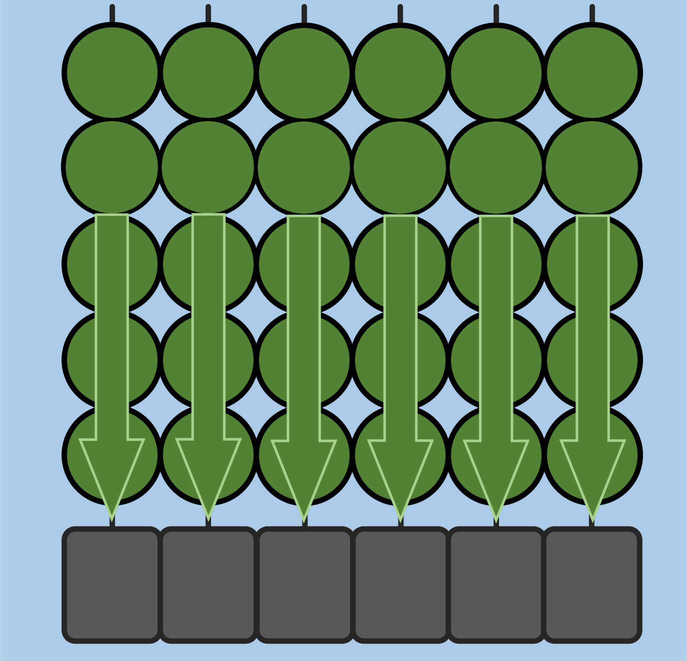
SoftMC: A Flexible and Practical Open-Source Infrastructure for Enabling Experimental DRAM Studies

Hasan Hassan[1,2,3] Nandita Vijaykumar[3] Samira Khan[4,3] Saugata Ghose[3] Kevin Chang[3]
Gennady Pekhimenko[5,3] Donghyuk Lee[6,3] Oguz Ergin[2] Onur Mutlu[1,3]

[1]*ETH Zürich* [2]*TOBB University of Economics & Technology* [3]*Carnegie Mellon University*
[4]*University of Virginia* [5]*Microsoft Research* [6]*NVIDIA Research*

Observation 1. Faster Sensing

Typical DIMM at Low Temperature



More Charge

Strong Charge Flow

Faster Sensing

115 DIMM Characterization

Timing

(tRCD)

17% ↓

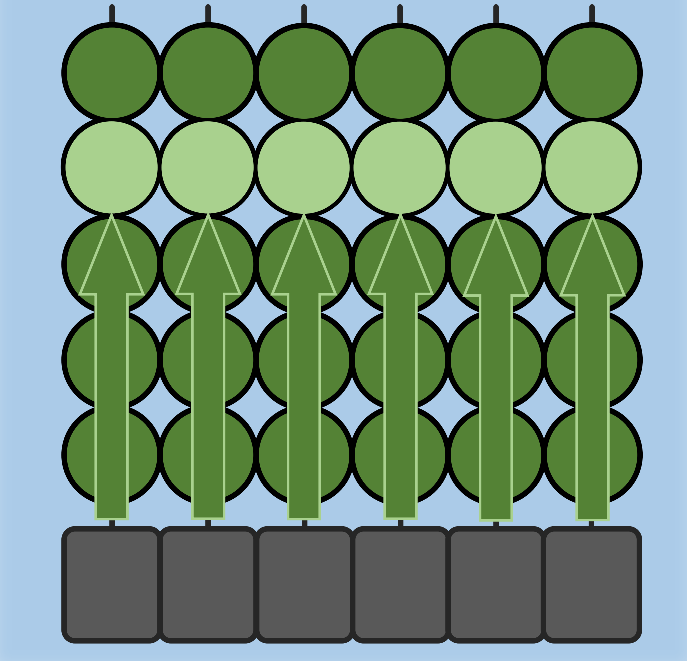
No Errors

Typical DIMM at Low Temperature

➔ More charge ➔ Faster sensing

Observation 2. Reducing Restore Time

Typical DIMM at Low Temperature



Less Leakage ➔ Extra Charge

No Need to Fully Restore Charge

115 DIMM Characterization

Read `(tRAS)`

37% ↓

Write `(tWR)`

54% ↓

No Errors

Typical DIMM at lower temperature

➔ More charge ➔ Restore time reduction

SAFARI

AL-DRAM

- *Key idea*
  - **Optimize DRAM timing parameters online**

- *Two components*
  - DRAM manufacturer provides multiple sets of reliable DRAM timing parameters at different temperatures for each DIMM
  - System monitors DRAM temperature & uses appropriate DRAM timing parameters

Lee+, "Adaptive-Latency DRAM: Optimizing DRAM Timing for the Common-Case," HPCA 2015.

DRAM Temperature

- *DRAM temperature measurement*
  - Server cluster: Operates at under 34°C
  - Desktop: Operates at under 50°C
  - *DRAM standard optimized for 85 ℃*

DRAM operates at low temperatures in the common-case

- *Previous works – Maintain low DRAM temperature*
  - David+ ICAC 2011
  - Liu+ ISCA 2007
  - Zhu+ ITHERM 2008

Latency Reduction Summary of 115 DIMMs

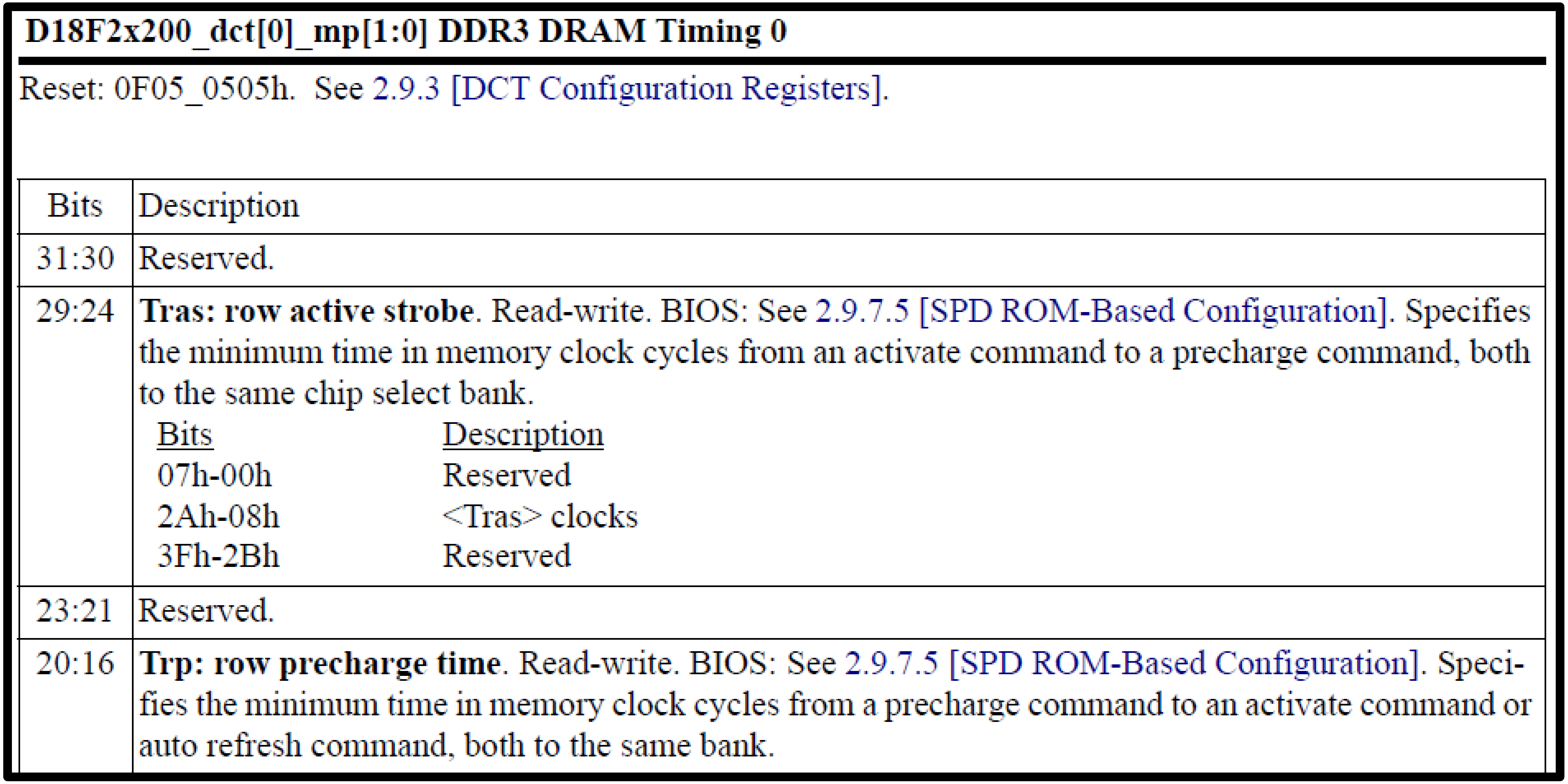
- *Latency reduction for read & write (55°C)*
  - *Read Latency: **32.7%***
  - *Write Latency: **55.1%***

- *Latency reduction for each timing parameter (55°C)*
  - *Sensing: **17.3%***
  - *Restore: **37.3%** (read), **54.8%** (write)*
  - *Precharge: **35.2%***

Lee+, "Adaptive-Latency DRAM: Optimizing DRAM Timing for the Common-Case," HPCA 2015.

AL-DRAM: Real System Evaluation

- *System*
  - *– CPU: AMD 4386 (8 Cores, 3.1GHz, 8MB LLC)*

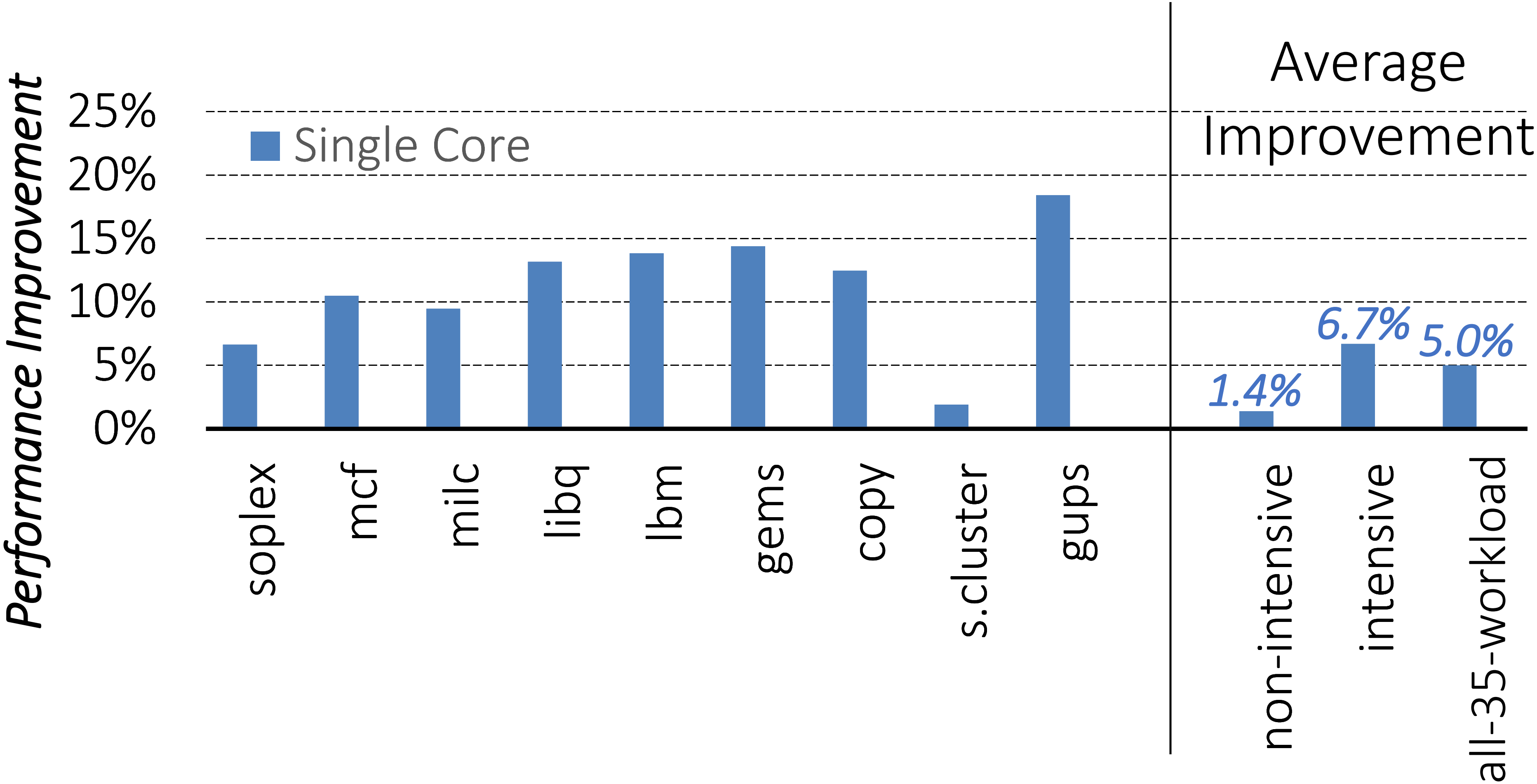


D18F2x200_dct[0]_mp[1:0] DDR3 DRAM Timing 0

Reset: 0F05_0505h. See 2.9.3 [DCT Configuration Registers].

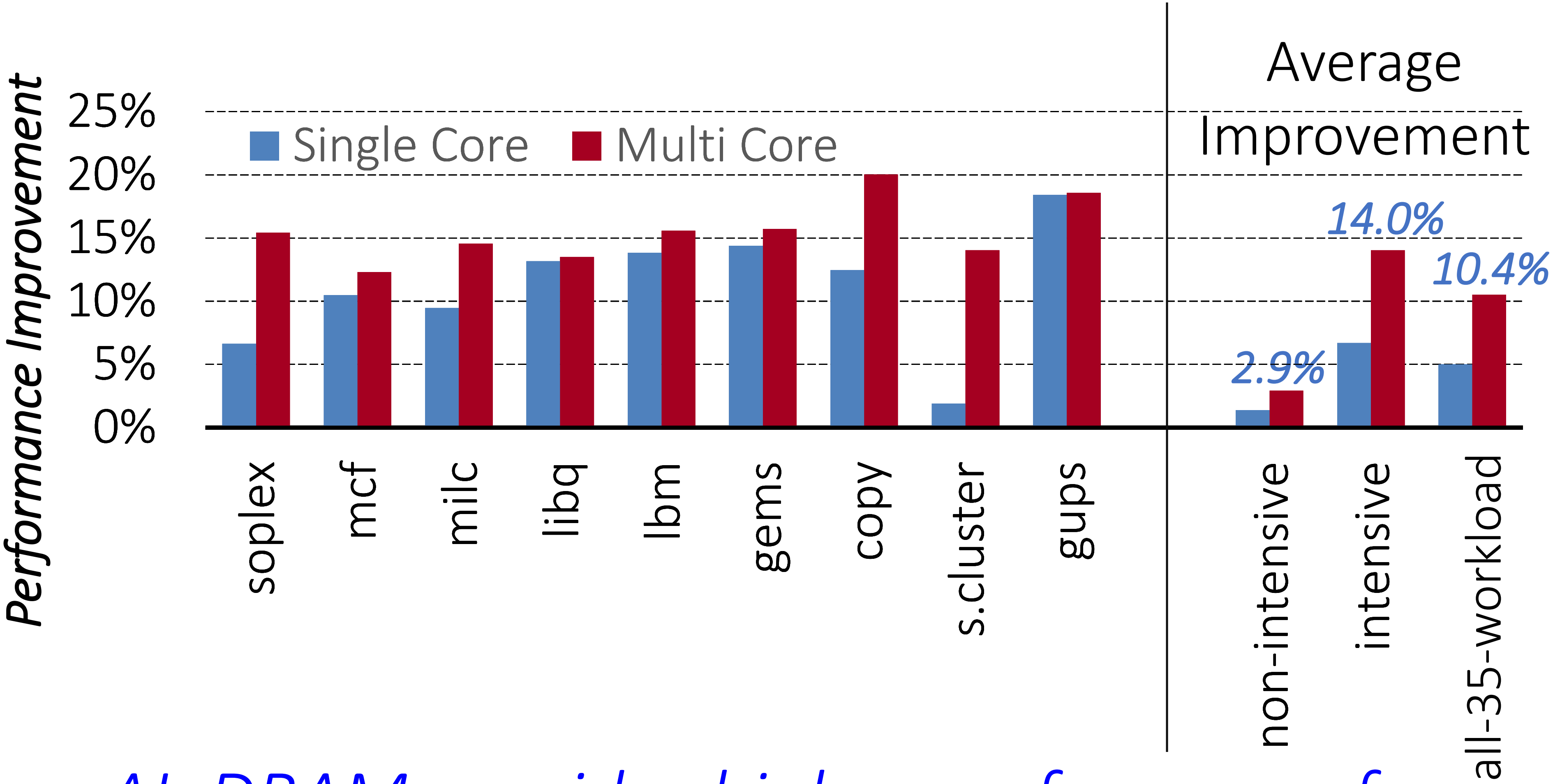
Bits	Description
31:30	Reserved.
29:24	**Tras: row active strobe**. Read-write. BIOS: See 2.9.7.5 [SPD ROM-Based Configuration]. Specifies the minimum time in memory clock cycles from an activate command to a precharge command, both to the same chip select bank. Bits Description 07h-00h Reserved 2Ah-08h <Tras> clocks 3Fh-2Bh Reserved
23:21	Reserved.
20:16	**Trp: row precharge time**. Read-write. BIOS: See 2.9.7.5 [SPD ROM-Based Configuration]. Specifies the minimum time in memory clock cycles from a precharge command to an activate command or auto refresh command, both to the same bank.

AL-DRAM: Single-Core Evaluation



*AL-DRAM **improves performance** on a real system*

AL-DRAM: Multi-Core Evaluation



AL-DRAM provides higher performance for ***multi-programmed & multi-threaded workloads***

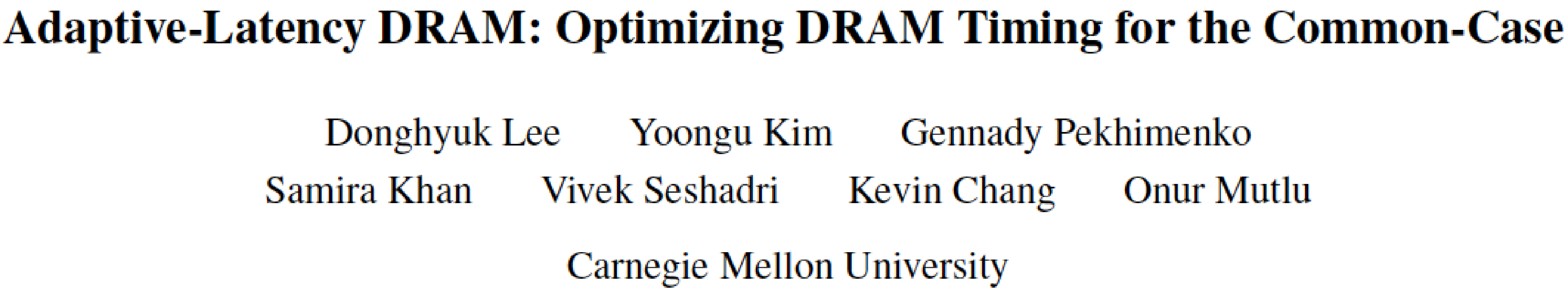
SAFARI

Reducing Latency Also Reduces Energy

- AL-DRAM reduces DRAM power consumption by 5.8%
- Major reason: reduction in row activation time

More on AL-DRAM

- Donghyuk Lee, Yoongu Kim, Gennady Pekhimenko, Samira Khan, Vivek Seshadri, Kevin Chang, and Onur Mutlu,
"Adaptive-Latency DRAM: Optimizing DRAM Timing for the Common-Case"
*Proceedings of the 21st International Symposium on High-Performance Computer Architecture (**HPCA**)*, Bay Area, CA, February 2015.
[Slides (pptx) (pdf)] [Full data sets]

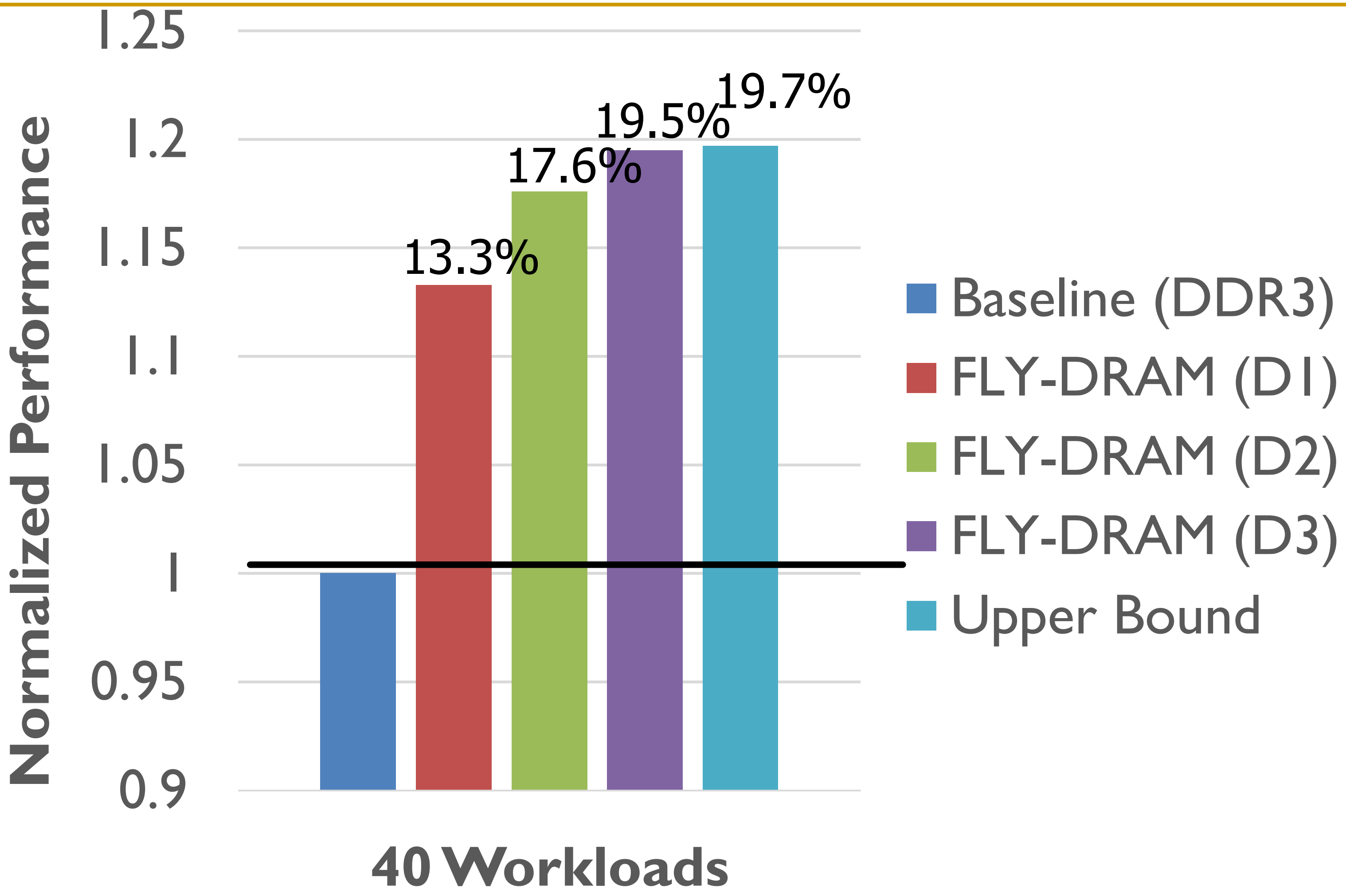


Adaptive-Latency DRAM: Optimizing DRAM Timing for the Common-Case

Donghyuk Lee Yoongu Kim Gennady Pekhimenko
Samira Khan Vivek Seshadri Kevin Chang Onur Mutlu

Carnegie Mellon University

Heterogeneous Latency within A Chip



Chang+, "**Understanding Latency Variation in Modern DRAM Chips: Experimental Characterization, Analysis, and Optimization"**," SIGMETRICS 2016.

Analysis of Latency Variation in DRAM Chips

- Kevin Chang, Abhijith Kashyap, Hasan Hassan, Samira Khan, Kevin Hsieh, Donghyuk Lee, Saugata Ghose, Gennady Pekhimenko, Tianshi Li, and Onur Mutlu,
"Understanding Latency Variation in Modern DRAM Chips: Experimental Characterization, Analysis, and Optimization"
*Proceedings of the ACM International Conference on Measurement and Modeling of Computer Systems (**SIGMETRICS**)*, Antibes Juan-Les-Pins, France, June 2016.
[Slides (pptx) (pdf)]
[Source Code]

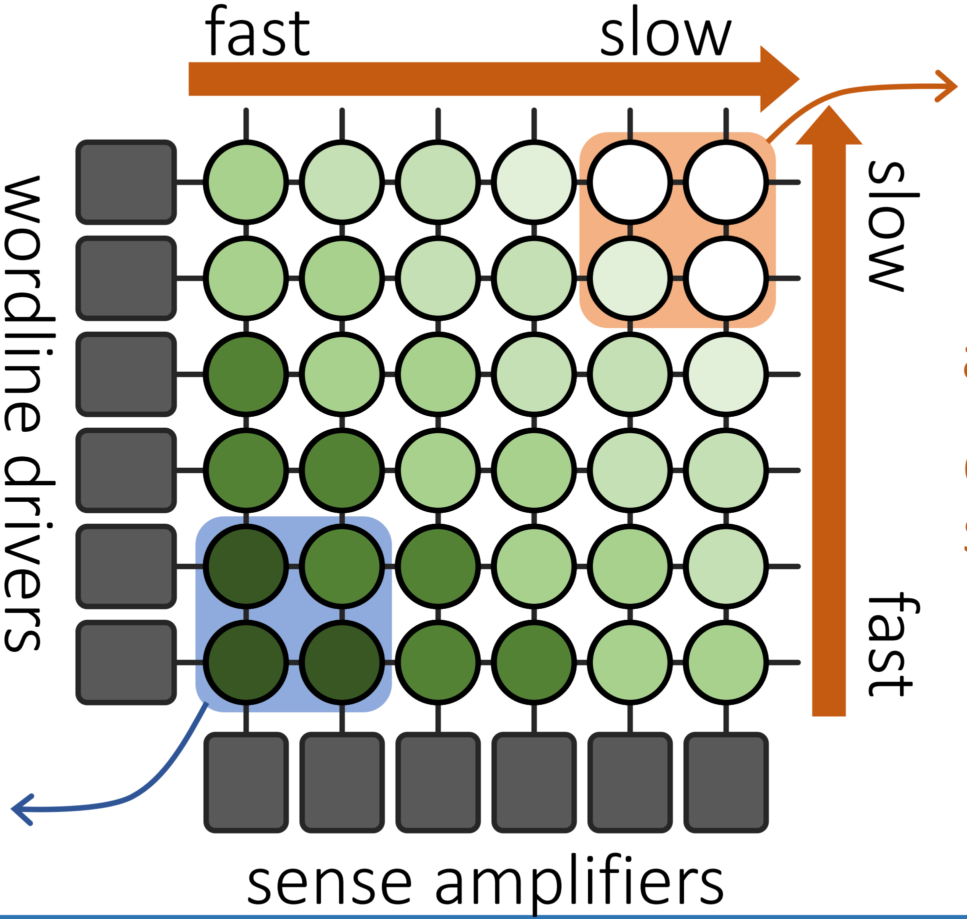
Understanding Latency Variation in Modern DRAM Chips: Experimental Characterization, Analysis, and Optimization

Kevin K. Chang[1] Abhijith Kashyap[1] Hasan Hassan[1,2]
Saugata Ghose[1] Kevin Hsieh[1] Donghyuk Lee[1] Tianshi Li[1,3]
Gennady Pekhimenko[1] Samira Khan[4] Onur Mutlu[5,1]
[1]Carnegie Mellon University [2]TOBB ETÜ [3]Peking University [4]University of Virginia [5]ETH Zürich

What Is Design-Induced Variation?

across column

distance from wordline driver



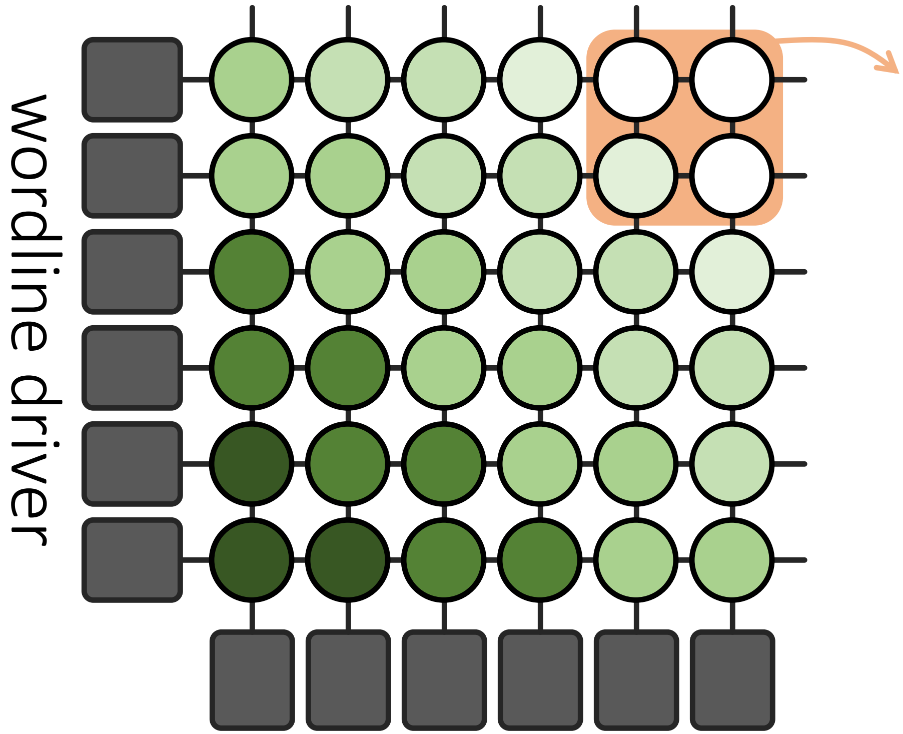
across row

distance from sense amplifier

Systematic variation in cell access times caused by the ***physical organization*** of DRAM

DIVA Online **Profiling**

Design-**I**nduced-**V**ariation-**A**ware



sense amplifier

Profile *only slow regions* to determine min. latency

→ *Dynamic* & *low cost* latency optimization

DIVA Online **Profiling**

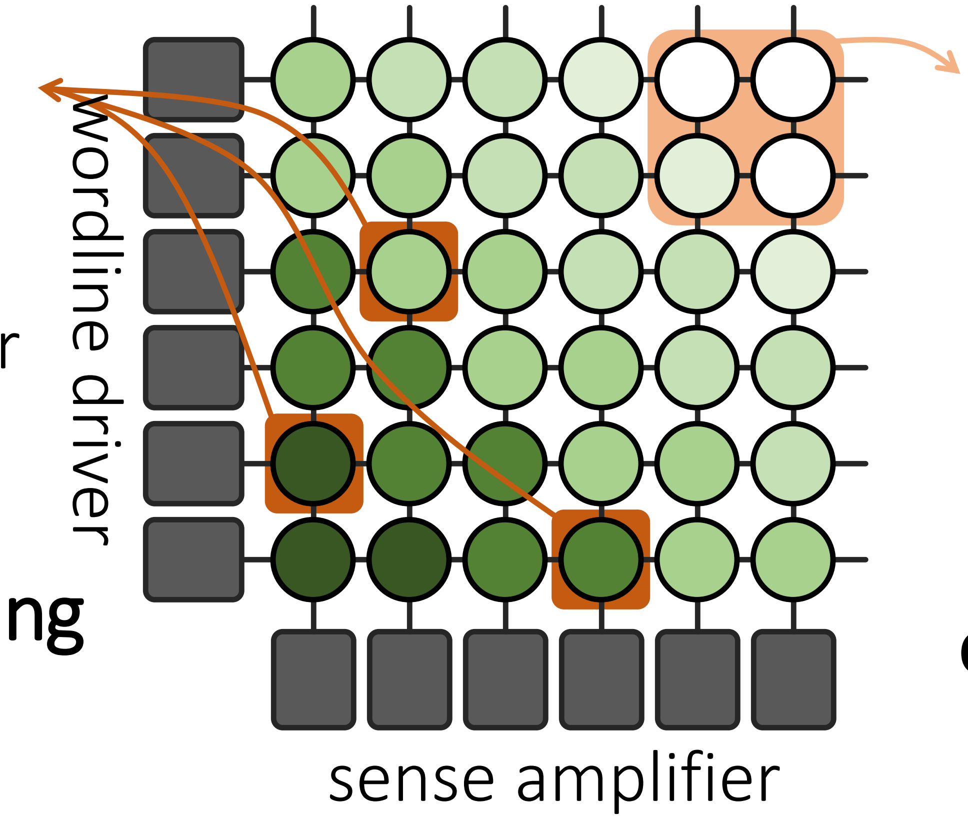
Design-**I**nduced-**V**ariation-**A**ware

slow cells

process variation

random error

error-correcting code



inherently slow

design-induced variation

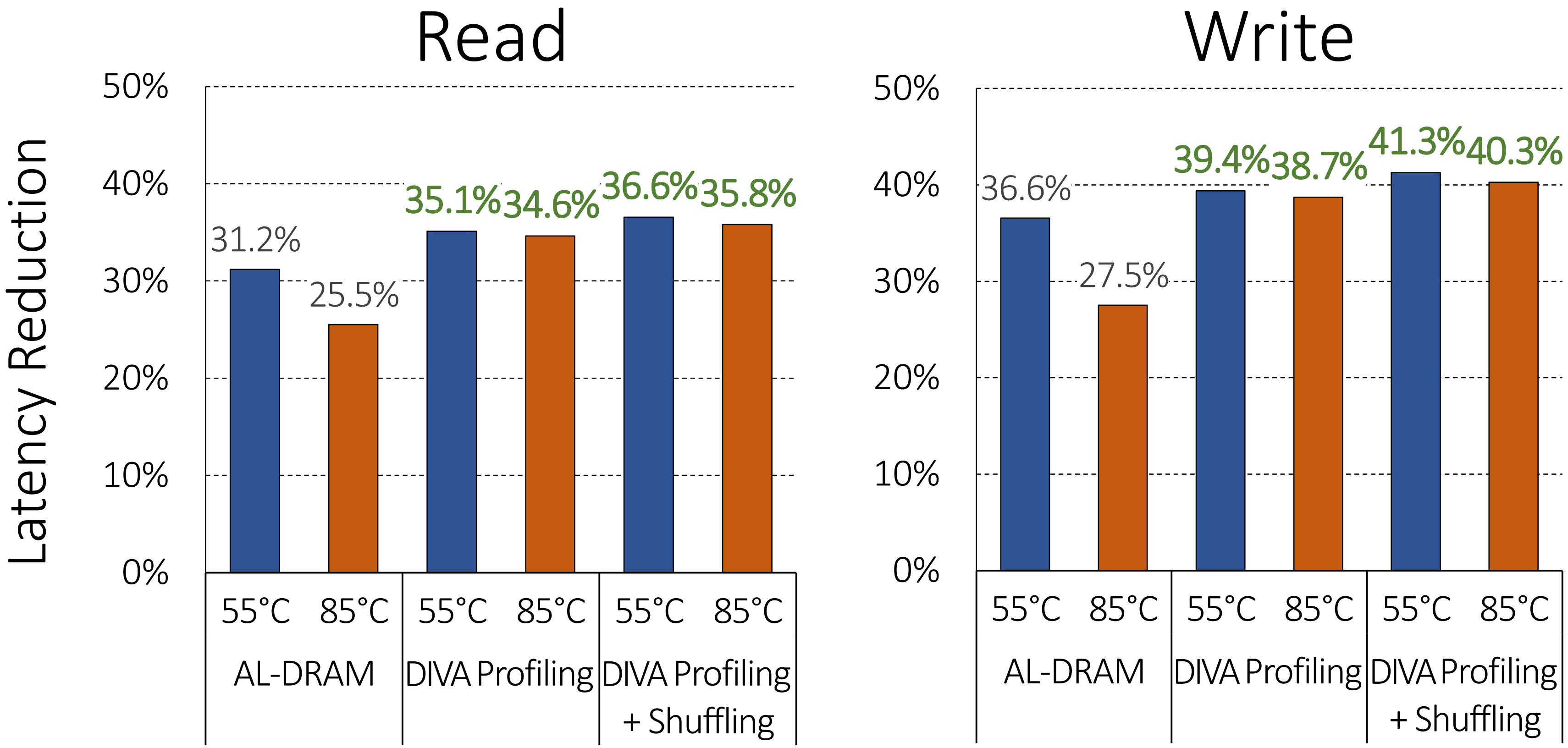
localized error

online profiling

Combine error-correcting codes & online profiling

→ Reliably reduce DRAM latency

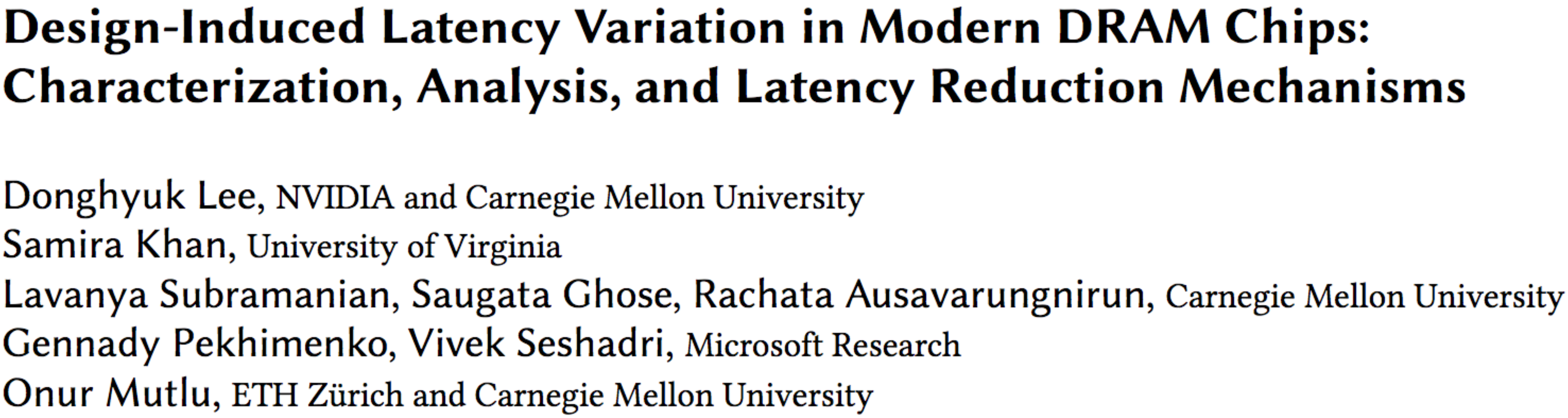
DIVA-DRAM Reduces Latency



DIVA-DRAM ***reduces latency more aggressively*** and uses ECC to correct random slow cells

Design-Induced Latency Variation in DRAM

- Donghyuk Lee, Samira Khan, Lavanya Subramanian, Saugata Ghose, Rachata Ausavarungnirun, Gennady Pekhimenko, Vivek Seshadri, and Onur Mutlu, **"Design-Induced Latency Variation in Modern DRAM Chips: Characterization, Analysis, and Latency Reduction Mechanisms"** *Proceedings of the ACM International Conference on Measurement and Modeling of Computer Systems (**SIGMETRICS**)*, Urbana-Champaign, IL, USA, June 2017.



Design-Induced Latency Variation in Modern DRAM Chips: Characterization, Analysis, and Latency Reduction Mechanisms

Donghyuk Lee, NVIDIA and Carnegie Mellon University
Samira Khan, University of Virginia
Lavanya Subramanian, Saugata Ghose, Rachata Ausavarungnirun, Carnegie Mellon University
Gennady Pekhimenko, Vivek Seshadri, Microsoft Research
Onur Mutlu, ETH Zürich and Carnegie Mellon University

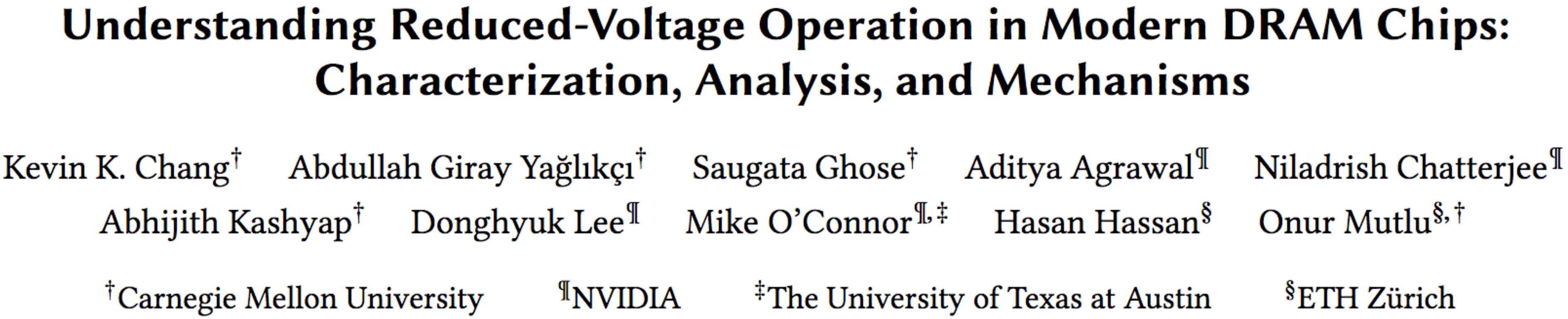
Voltron: Exploiting the Voltage-Latency-Reliability Relationship

Executive Summary

- DRAM (memory) power is significant in today's systems
  - Existing low-voltage DRAM reduces voltage **conservatively**
- Goal: Understand and exploit the reliability and latency behavior of real DRAM chips under ***aggressive reduced-voltage operation***
- Key experimental observations:
  - Huge voltage margin -- Errors occur beyond some voltage
  - Errors exhibit spatial locality
  - Higher operation latency mitigates voltage-induced errors
- **Voltron**: A new DRAM energy reduction mechanism
  - Reduce DRAM voltage **without introducing errors**
  - Use a **regression model** to select voltage that does not degrade performance beyond a chosen target → 7.3% system energy reduction

Analysis of Latency-Voltage in DRAM Chips

- Kevin Chang, A. Giray Yaglikci, Saugata Ghose, Aditya Agrawal, Niladrish Chatterjee, Abhijith Kashyap, Donghyuk Lee, Mike O'Connor, Hasan Hassan, and Onur Mutlu,
  "Understanding Reduced-Voltage Operation in Modern DRAM Devices: Experimental Characterization, Analysis, and Mechanisms"
  *Proceedings of the ACM International Conference on Measurement and Modeling of Computer Systems (**SIGMETRICS**)*, Urbana-Champaign, IL, USA, June 2017.



Understanding Reduced-Voltage Operation in Modern DRAM Chips: Characterization, Analysis, and Mechanisms

Kevin K. Chang[†] Abdullah Giray Yağlıkçı[†] Saugata Ghose[†] Aditya Agrawal[¶] Niladrish Chatterjee[¶]
Abhijith Kashyap[†] Donghyuk Lee[¶] Mike O'Connor[¶,‡] Hasan Hassan[§] Onur Mutlu[§,†]

[†]Carnegie Mellon University [¶]NVIDIA [‡]The University of Texas at Austin [§]ETH Zürich

And, What If …

- … we can sacrifice reliability of some data to access it with even lower latency?

Challenge and Opportunity for Future

Fundamentally

Low Latency

Computing Architectures

Computer Architecture

Lecture 5: DRAM Operation, Memory Control & Memory Latency

Prof. Onur Mutlu

ETH Zürich

Fall 2017

4 October 2017