Computer Architecture

Lecture 5: Main Memory and DRAM Fundamentals

Prof. Onur Mutlu
ETH Zürich

Fall 2018

3 October 2018

Last Lecture

- Wrap-up Caches
- Main Memory and Its Importance
- Main Memory Trends and Challenges

Agenda for This Lecture

- Wrap-up Main Memory Challenges
- Main Memory Fundamentals
- DRAM Basics and Operation
- Memory Controllers
- Simulation
- Memory Latency

The Main Memory System

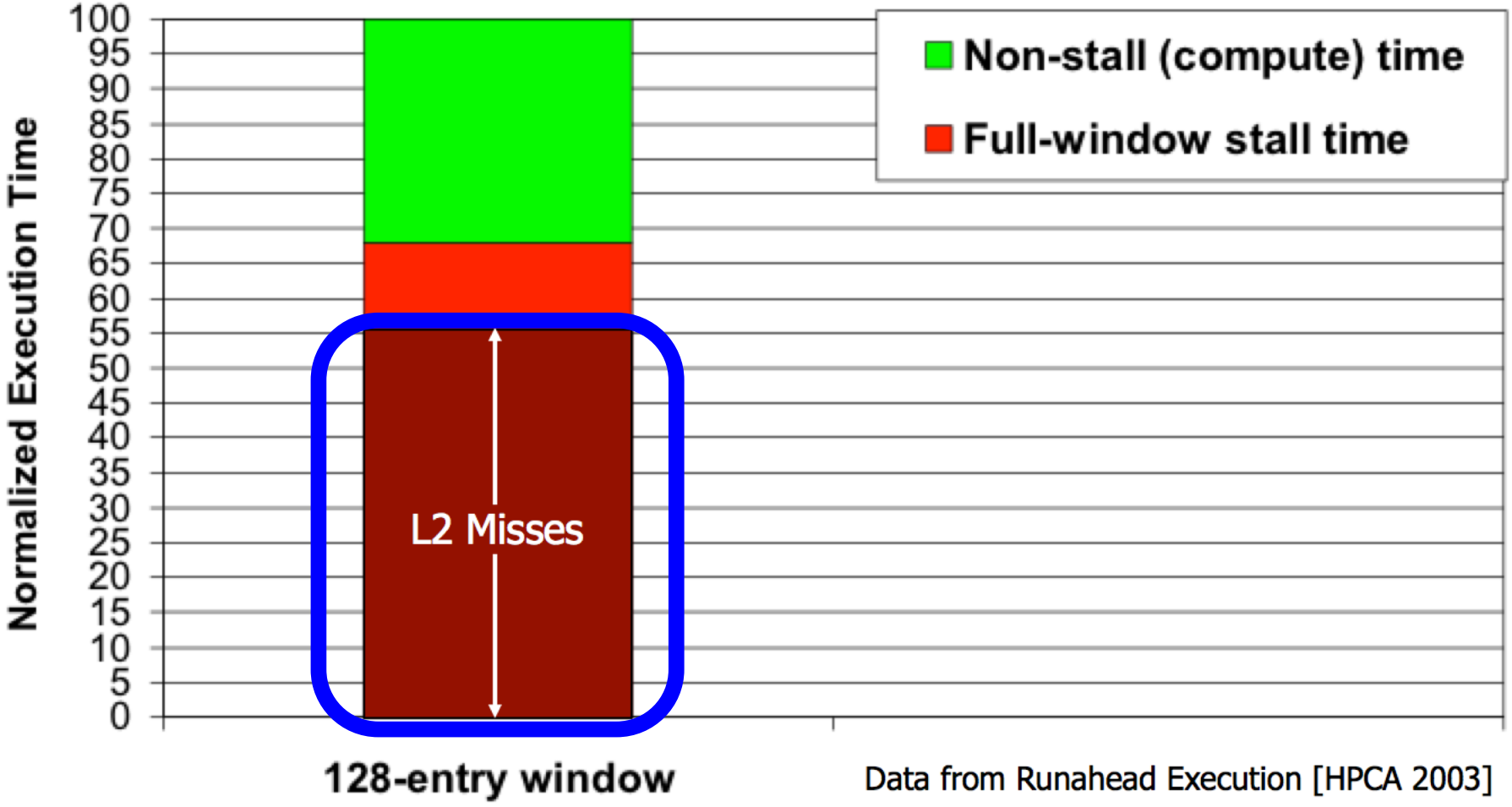
Why Is Memory So Important? (Especially Today)

Importance of Main Memory

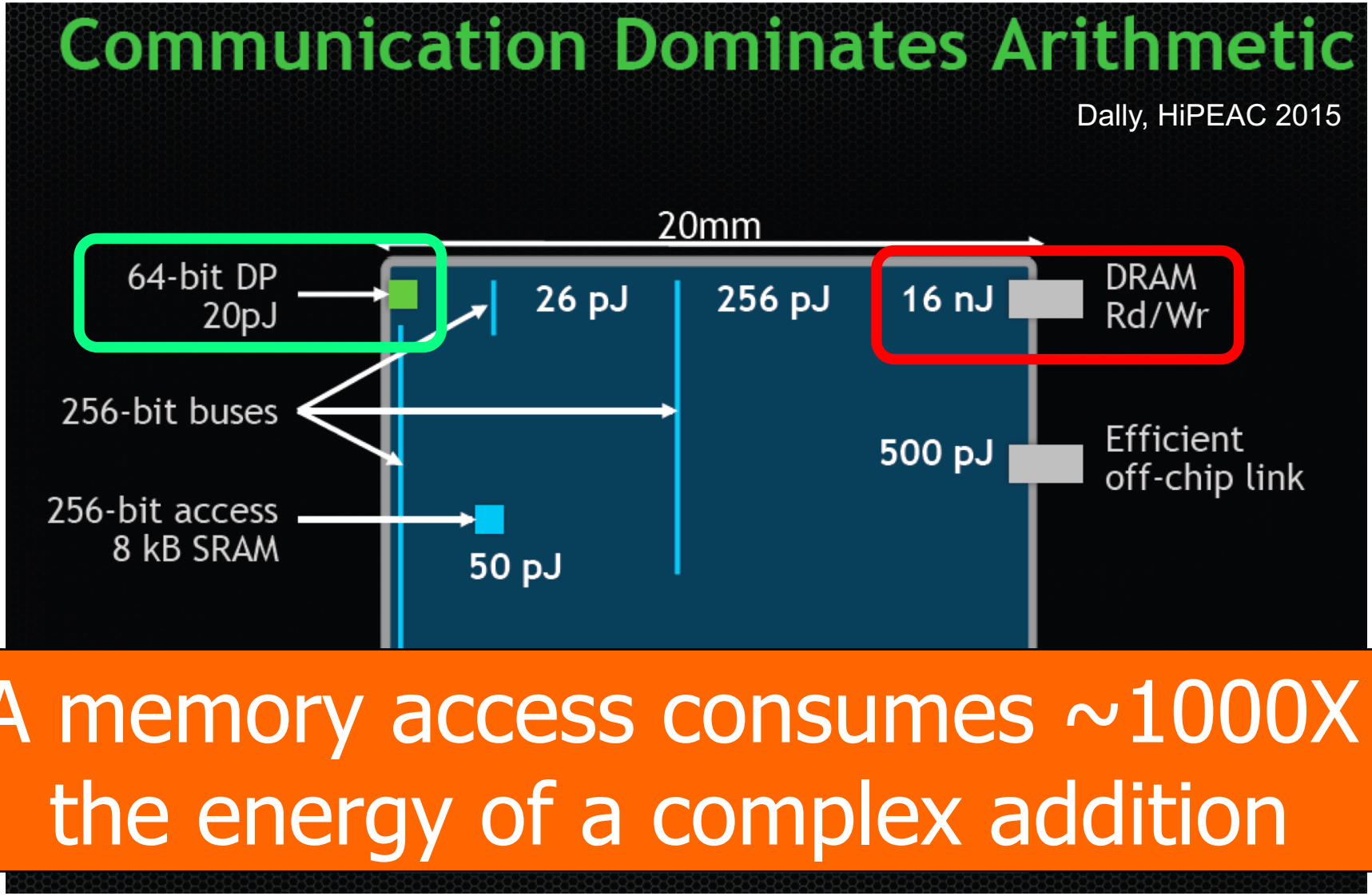
- The Performance Perspective
- The Energy Perspective
- The Reliability/Security Perspective
- Trends/Challenges/Opportunities in Main Memory

The Performance Perspective

- "**It's the Memory, Stupid!**" (Richard Sites, MPR, 1996)



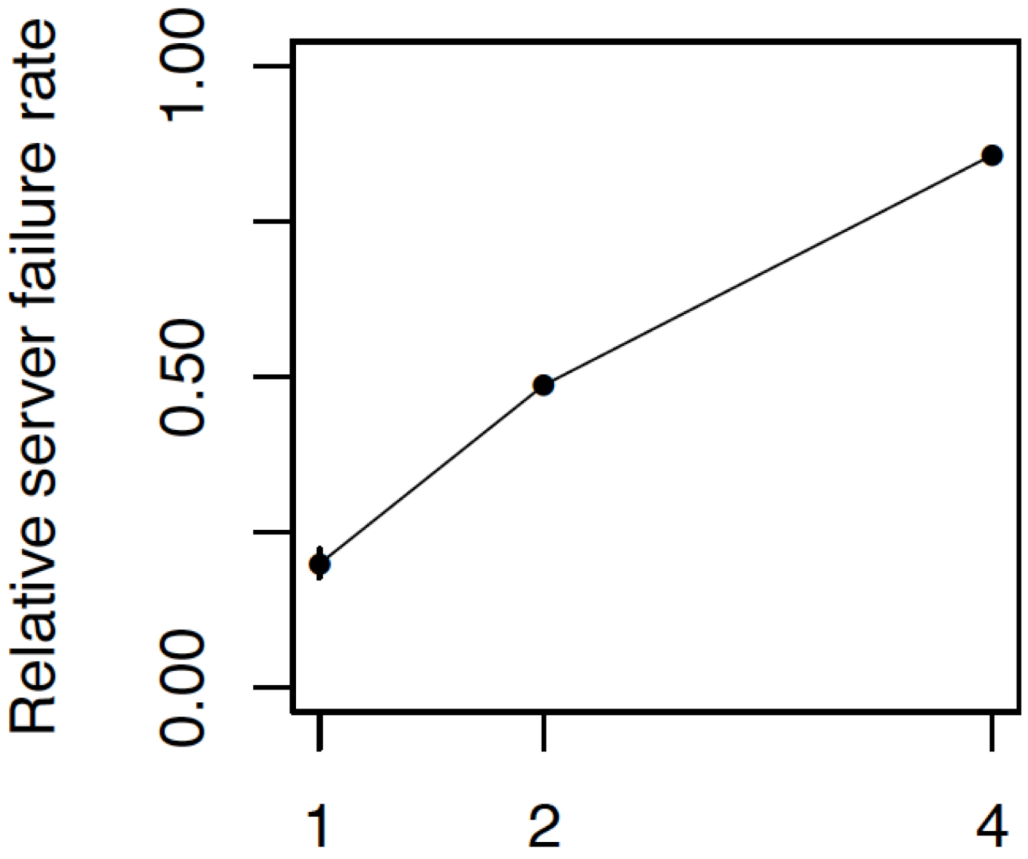
The Energy Perspective



A memory access consumes ~1000X the energy of a complex addition

The Reliability Perspective

- Data from all of Facebook's servers worldwide
- Meza+, "Revisiting Memory Errors in Large-Scale Production Data Centers," DSN'15.



Intuition: quadratic increase in capacity

The Security Perspective

It's like breaking into an apartment by repeatedly slamming a neighbor's door until the vibrations open the door you were after

The Reliability & Security Perspectives

- Onur Mutlu,
  "The RowHammer Problem and Other Issues We May Face as Memory Becomes Denser"
  *Invited Paper in Proceedings of the Design, Automation, and Test in Europe Conference (**DATE**)*, Lausanne, Switzerland, March 2017.
  [Slides (pptx) (pdf)]

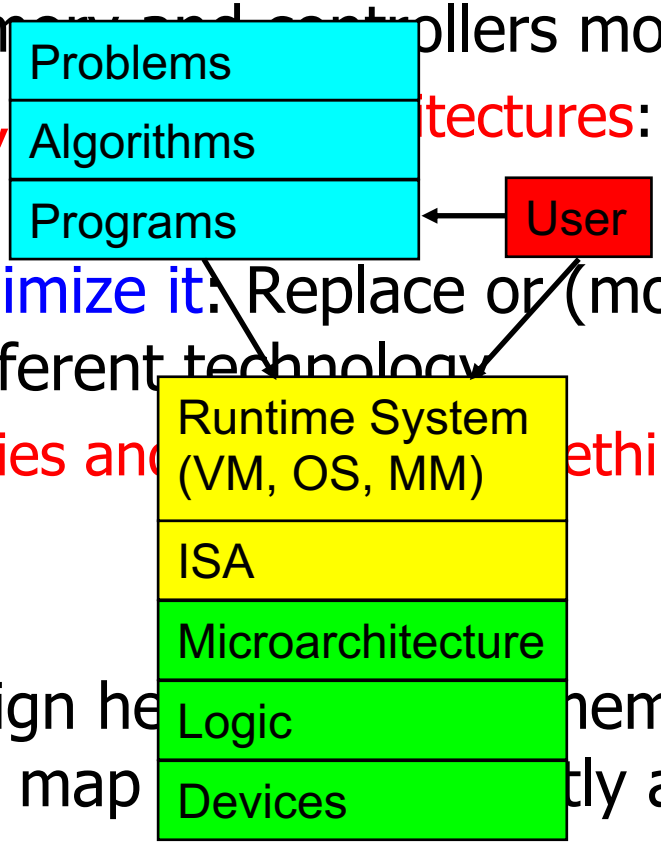
The RowHammer Problem
and Other Issues We May Face as Memory Becomes Denser

Onur Mutlu
ETH Zürich
onur.mutlu@inf.ethz.ch
https://people.inf.ethz.ch/omutlu

Trends, Challenges, and Opportunities in Main Memory

How Do We Solve The Memory Problem?

- Fix it: Make mem[illegible]llers more intelligent
  - New interfaces, [illegible]tectures: system-mem codesign
- Eliminate or minimize it: Replace or (more likely) augment DRAM with a different technology
  - New technologies an[illegible]ethinking of memory & storage
- Embrace it: Design he[illegible]nemories (none of which are perfect) and map [illegible]tly across them
  - New models for data management and maybe usage



Solutions (to memory scaling) require software/hardware/device cooperation

Solution 1: New Memory Architectures

- Overcome memory shortcomings with
  - Memory-centric system design
  - Novel memory architectures, interfaces, functions
  - Better waste management (efficient utilization)

- Key issues to tackle
  - Enable reliability at low cost → high capacity
  - Reduce energy
  - Reduce latency
  - Improve bandwidth
  - Reduce waste (capacity, bandwidth, latency)
  - Enable computation close to data

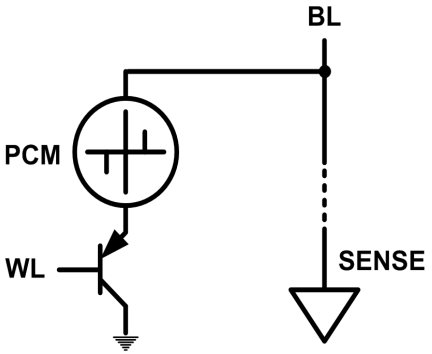
Solution 1: New Memory Architectures

- Liu+, "RAIDR: Retention-Aware Intelligent DRAM Refresh," ISCA 2012.
- Kim+, "A Case for Exploiting Subarray-Level Parallelism in DRAM," ISCA 2012.
- Lee+, "Tiered-Latency DRAM: A Low Latency and Low Cost DRAM Architecture," HPCA 2013.
- Liu+, "An Experimental Study of Data Retention Behavior in Modern DRAM Devices," ISCA 2013.
- Seshadri+, "RowClone: Fast and Efficient In-DRAM Copy and Initialization of Bulk Data," MICRO 2013.
- Pekhimenko+, "Linearly Compressed Pages: A Main Memory Compression Framework," MICRO 2013.
- Chang+, "Improving DRAM Performance by Parallelizing Refreshes with Accesses," HPCA 2014.
- Khan+, "The Efficacy of Error Mitigation Techniques for DRAM Retention Failures: A Comparative Experimental Study," SIGMETRICS 2014.
- Luo+, "Characterizing Application Memory Error Vulnerability to Optimize Data Center Cost," DSN 2014.
- Kim+, "Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors," ISCA 2014.
- Lee+, "Adaptive-Latency DRAM: Optimizing DRAM Timing for the Common-Case," HPCA 2015.
- Qureshi+, "AVATAR: A Variable-Retention-Time (VRT) Aware Refresh for DRAM Systems," DSN 2015.
- Meza+, "Revisiting Memory Errors in Large-Scale Production Data Centers: Analysis and Modeling of New Trends from the Field," DSN 2015.
- Kim+, "Ramulator: A Fast and Extensible DRAM Simulator," IEEE CAL 2015.
- Seshadri+, "Fast Bulk Bitwise AND and OR in DRAM," IEEE CAL 2015.
- Ahn+, "A Scalable Processing-in-Memory Accelerator for Parallel Graph Processing," ISCA 2015.
- Ahn+, "PIM-Enabled Instructions: A Low-Overhead, Locality-Aware Processing-in-Memory Architecture," ISCA 2015.
- Lee+, "Decoupled Direct Memory Access: Isolating CPU and IO Traffic by Leveraging a Dual-Data-Port DRAM," PACT 2015.
- Seshadri+, "Gather-Scatter DRAM: In-DRAM Address Translation to Improve the Spatial Locality of Non-unit Strided Accesses," MICRO 2015.
- Lee+, "Simultaneous Multi-Layer Access: Improving 3D-Stacked Memory Bandwidth at Low Cost," TACO 2016.
- Hassan+, "ChargeCache: Reducing DRAM Latency by Exploiting Row Access Locality," HPCA 2016.
- Chang+, "Low-Cost Inter-Linked Subarrays (LISA): Enabling Fast Inter-Subarray Data Migration in DRAM," HPCA 2016.
- Chang+, "Understanding Latency Variation in Modern DRAM Chips Experimental Characterization, Analysis, and Optimization," SIGMETRICS 2016.
- Khan+, "PARBOR: An Efficient System-Level Technique to Detect Data Dependent Failures in DRAM," DSN 2016.
- Hsieh+, "Transparent Offloading and Mapping (TOM): Enabling Programmer-Transparent Near-Data Processing in GPU Systems," ISCA 2016.
- Hashemi+, "Accelerating Dependent Cache Misses with an Enhanced Memory Controller," ISCA 2016.
- Boroumand+, "LazyPIM: An Efficient Cache Coherence Mechanism for Processing-in-Memory," IEEE CAL 2016.
- Pattnaik+, "Scheduling Techniques for GPU Architectures with Processing-In-Memory Capabilities," PACT 2016.
- Hsieh+, "Accelerating Pointer Chasing in 3D-Stacked Memory: Challenges, Mechanisms, Evaluation," ICCD 2016.
- Hashemi+, "Continuous Runahead: Transparent Hardware Acceleration for Memory Intensive Workloads," MICRO 2016.
- Khan+, "A Case for Memory Content-Based Detection and Mitigation of Data-Dependent Failures in DRAM"," IEEE CAL 2016.
- Hassan+, "SoftMC: A Flexible and Practical Open-Source Infrastructure for Enabling Experimental DRAM Studies," HPCA 2017.
- Mutlu, "The RowHammer Problem and Other Issues We May Face as Memory Becomes Denser," DATE 2017.
- Lee+, "Design-Induced Latency Variation in Modern DRAM Chips: Characterization, Analysis, and Latency Reduction Mechanisms," SIGMETRICS 2017.
- Chang+, "Understanding Reduced-Voltage Operation in Modern DRAM Devices: Experimental Characterization, Analysis, and Mechanisms," SIGMETRICS 2017.
- Patel+, "The Reach Profiler (REAPER): Enabling the Mitigation of DRAM Retention Failures via Profiling at Aggressive Conditions," ISCA 2017.
- Seshadri and Mutlu, "Simple Operations in Memory to Reduce Data Movement," ADCOM 2017.
- Liu+, "Concurrent Data Structures for Near-Memory Computing," SPAA 2017.
- Khan+, "Detecting and Mitigating Data-Dependent DRAM Failures by Exploiting Current Memory Content," MICRO 2017.
- Seshadri+, "Ambit: In-Memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology," MICRO 2017.
- Avoid DRAM:
  - Seshadri+, "The Evicted-Address Filter: A Unified Mechanism to Address Both Cache Pollution and Thrashing," PACT 2012.
  - Pekhimenko+, "Base-Delta-Immediate Compression: Practical Data Compression for On-Chip Caches," PACT 2012.
  - Seshadri+, "The Dirty-Block Index," ISCA 2014.
  - Pekhimenko+, "Exploiting Compressed Block Size as an Indicator of Future Reuse," HPCA 2015.
  - Vijaykumar+, "A Case for Core-Assisted Bottleneck Acceleration in GPUs: Enabling Flexible Data Compression with Assist Warps," ISCA 2015.
  - Pekhimenko+, "Toggle-Aware Bandwidth Compression for GPUs," HPCA 2016.

Solution 2: Emerging Memory Technologies

- Some emerging **resistive** memory technologies seem more scalable than DRAM (and they are non-volatile)

- Example: Phase Change Memory
  - Data stored by changing phase of material
  - Data read by detecting material's resistance
  - Expected to scale to 9nm (2022 [ITRS 2009])
  - Prototyped at 20nm (Raoux+, IBM JRD 2008)
  - Expected to be denser than DRAM: can store multiple bits/cell

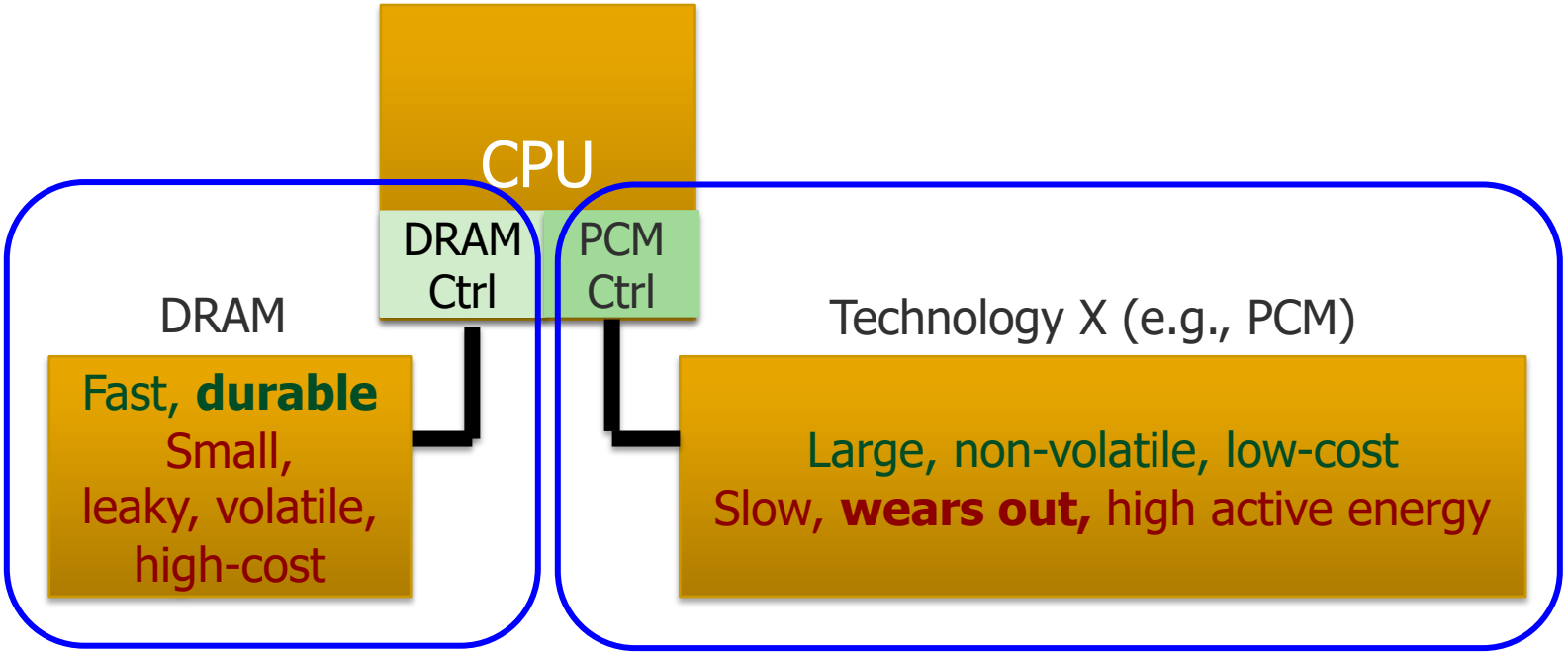


- But, emerging technologies have (many) shortcomings
  - Can they be enabled to replace/augment/surpass DRAM?

Solution 2: Emerging Memory Technologies

- Lee+, "Architecting Phase Change Memory as a Scalable DRAM Alternative," ISCA'09, CACM'10, IEEE Micro'10.
- Meza+, "Enabling Efficient and Scalable Hybrid Memories," IEEE Comp. Arch. Letters 2012.
- Yoon, Meza+, "Row Buffer Locality Aware Caching Policies for Hybrid Memories," ICCD 2012.
- Kultursay+, "Evaluating STT-RAM as an Energy-Efficient Main Memory Alternative," ISPASS 2013.
- Meza+, "A Case for Efficient Hardware-Software Cooperative Management of Storage and Memory," WEED 2013.
- Lu+, "Loose Ordering Consistency for Persistent Memory," ICCD 2014.
- Zhao+, "FIRM: Fair and High-Performance Memory Control for Persistent Memory Systems," MICRO 2014.
- Yoon, Meza+, "Efficient Data Mapping and Buffering Techniques for Multi-Level Cell Phase-Change Memories," TACO 2014.
- Ren+, "ThyNVM: Enabling Software-Transparent Crash Consistency in Persistent Memory Systems," MICRO 2015.
- Chauhan+, "NVMove: Helping Programmers Move to Byte-Based Persistence," INFLOW 2016.
- Li+, "Utility-Based Hybrid Memory Management," CLUSTER 2017.
- Yu+, "Banshee: Bandwidth-Efficient DRAM Caching via Software/Hardware Cooperation," MICRO 2017.

Combination: Hybrid Memory Systems

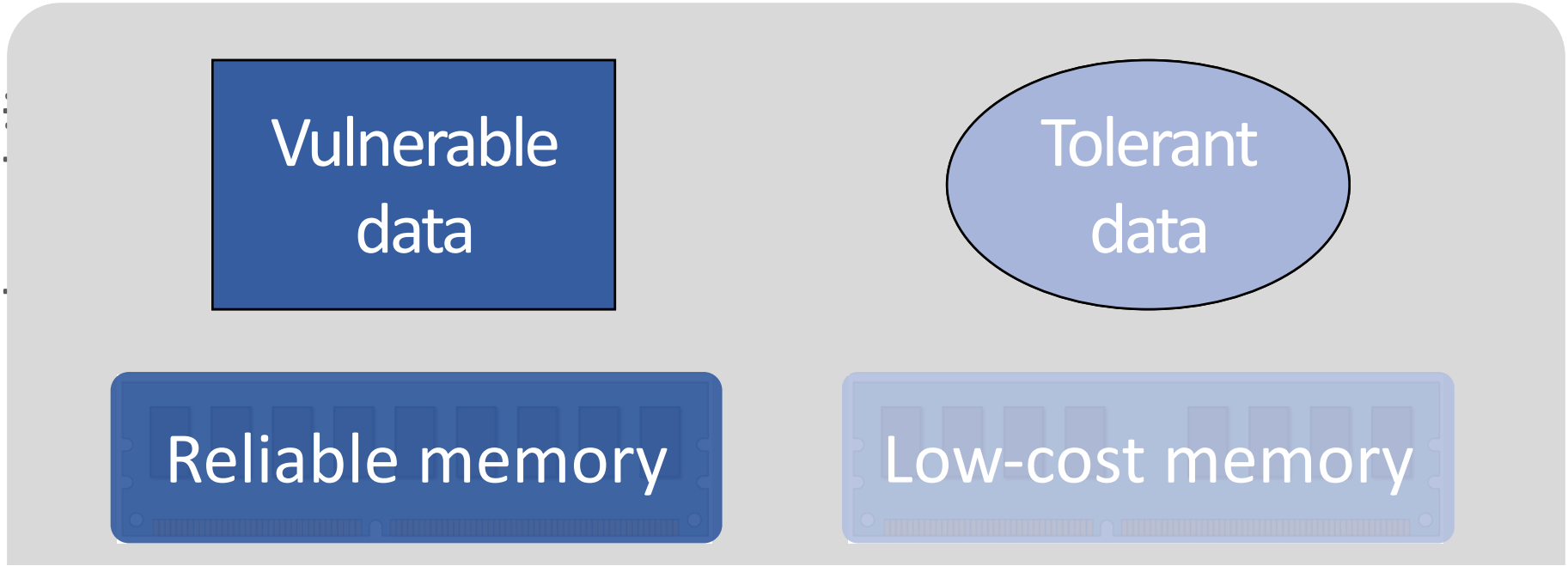


Hardware/software manage data allocation and movement
to achieve the best of multiple technologies

Meza+, "Enabling Efficient and Scalable Hybrid Memories," IEEE Comp. Arch. Letters, 2012.

Yoon, Meza et al., "Row Buffer Locality Aware Caching Policies for Hybrid Memories," ICCD 2012 Best Paper Award.

Exploiting Memory Error Tolerance with Hybrid Memory Systems



On Microsoft's Web Search workload

Reduces server hardware cost by 4.7 %

Achieves single server availability target of 99.90 %

Heterogeneous-**R**eliability **M**emory [DSN 2014]

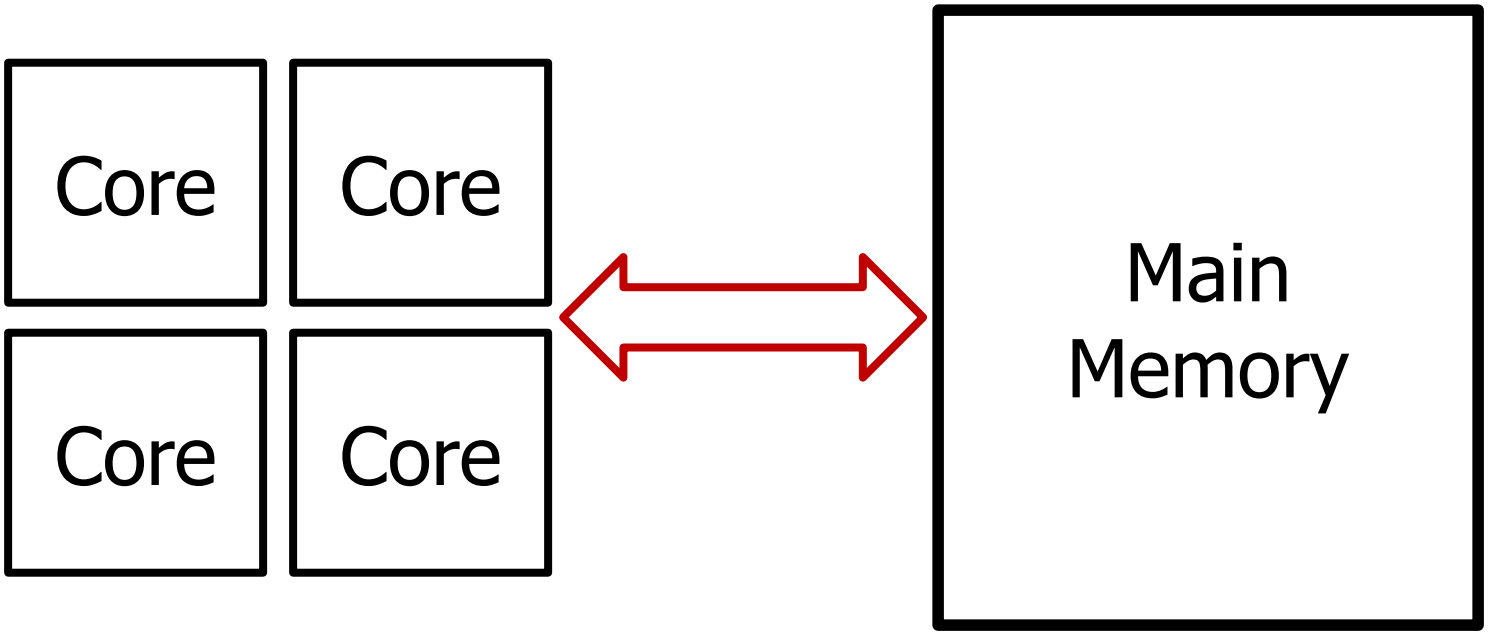
More on Heterogeneous Reliability Memory

- Yixin Luo, Sriram Govindan, Bikash Sharma, Mark Santaniello, Justin Meza, Aman Kansal, Jie Liu, Badriddine Khessib, Kushagra Vaid, and Onur Mutlu, **"Characterizing Application Memory Error Vulnerability to Optimize Data Center Cost via Heterogeneous-Reliability Memory"** *Proceedings of the 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (**DSN**)*, Atlanta, GA, June 2014. [Summary] [Slides (pptx) (pdf)] [Coverage on ZDNet]

Characterizing Application Memory Error Vulnerability to Optimize Datacenter Cost via Heterogeneous-Reliability Memory

Yixin Luo Sriram Govindan* Bikash Sharma* Mark Santaniello* Justin Meza
Aman Kansal* Jie Liu* Badriddine Khessib* Kushagra Vaid* Onur Mutlu
Carnegie Mellon University, yixinluo@cs.cmu.edu, {meza, onur}@cmu.edu
*Microsoft Corporation, {srgovin, bsharma, marksan, kansal, jie.liu, bkhessib, kvaid}@microsoft.com

An Orthogonal Issue: Memory Interference



Cores' interfere with each other when accessing shared main memory

Uncontrolled interference leads to many problems (QoS, performance)

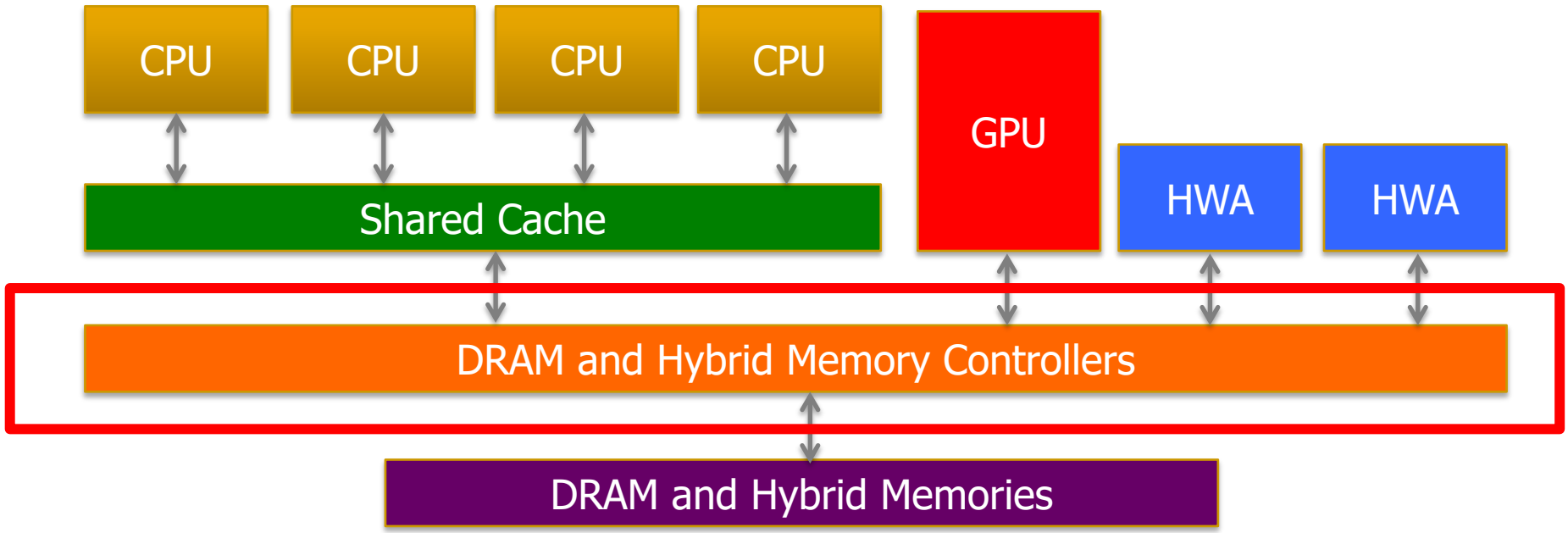
An Orthogonal Issue: Memory Interference

- Problem: Memory interference between cores is uncontrolled
  → unfairness, starvation, low performance
  → uncontrollable, unpredictable, vulnerable system

- Solution: QoS-Aware Memory Systems
  - Hardware designed to provide a configurable fairness substrate
    - Application-aware memory scheduling, partitioning, throttling
  - Software designed to configure the resources to satisfy different QoS goals

- QoS-aware memory systems can provide predictable performance and higher efficiency

Goal: Predictable Performance in Complex Systems



- Heterogeneous agents: CPUs, GPUs, and HWAs
- Main memory interference between CPUs, GPUs, HWAs

How to allocate resources to heterogeneous agents to mitigate interference and provide predictable performance?

Strong Memory Service Guarantees

- Goal: Satisfy performance/SLA requirements in the presence of shared main memory, heterogeneous agents, and hybrid memory/storage

- Approach:
  - Develop techniques/models to accurately estimate the performance loss of an application/agent in the presence of resource sharing
  - Develop mechanisms (hardware and software) to enable the resource partitioning/prioritization needed to achieve the required performance levels for all applications
  - All the while providing high system performance

- Subramanian et al., "MISE: Providing Performance Predictability and Improving Fairness in Shared Main Memory Systems," HPCA 2013.
- Subramanian et al., "The Application Slowdown Model," MICRO 2015.

How Can We Fix the Memory Problem & Design (Memory) Systems of the Future?

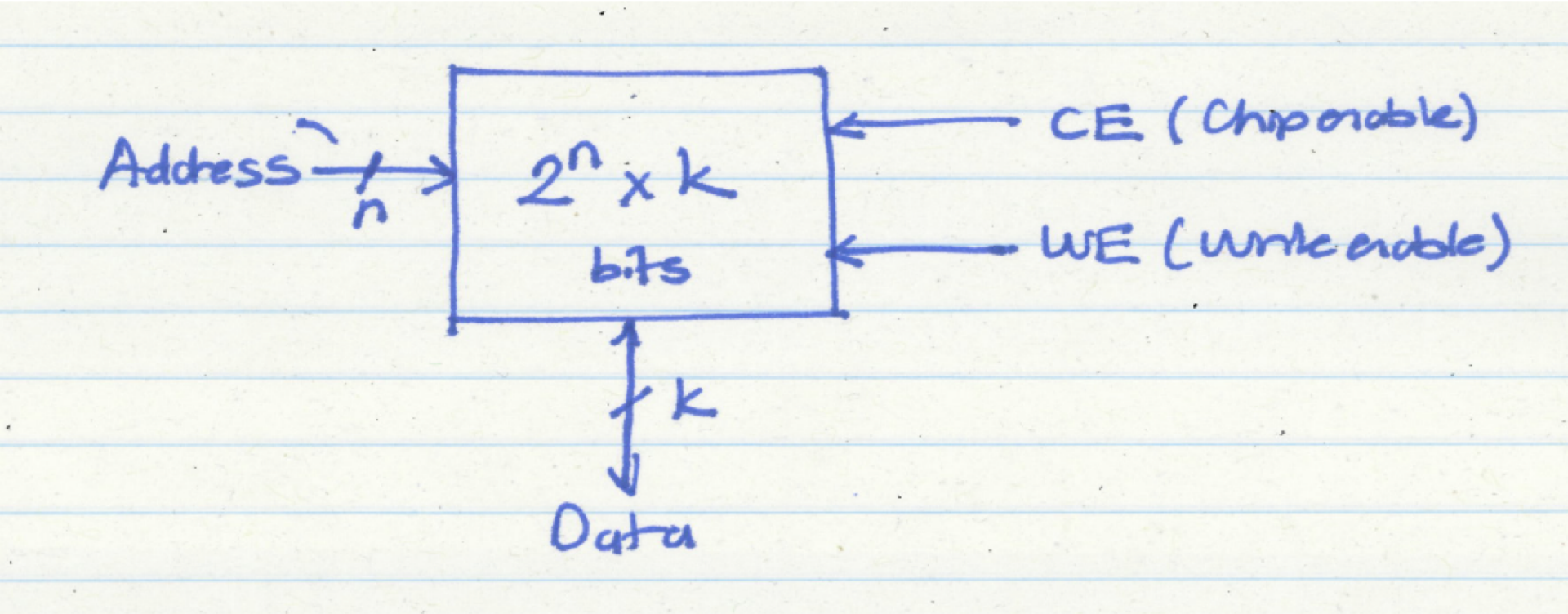
Plan of Action

- We first need to understand the principles of:
  - Memory and DRAM
  - Memory controllers
  - Techniques for reducing and tolerating memory latency
  - Potential memory technologies that can compete with DRAM
  - How to evaluate new ideas in memory systems

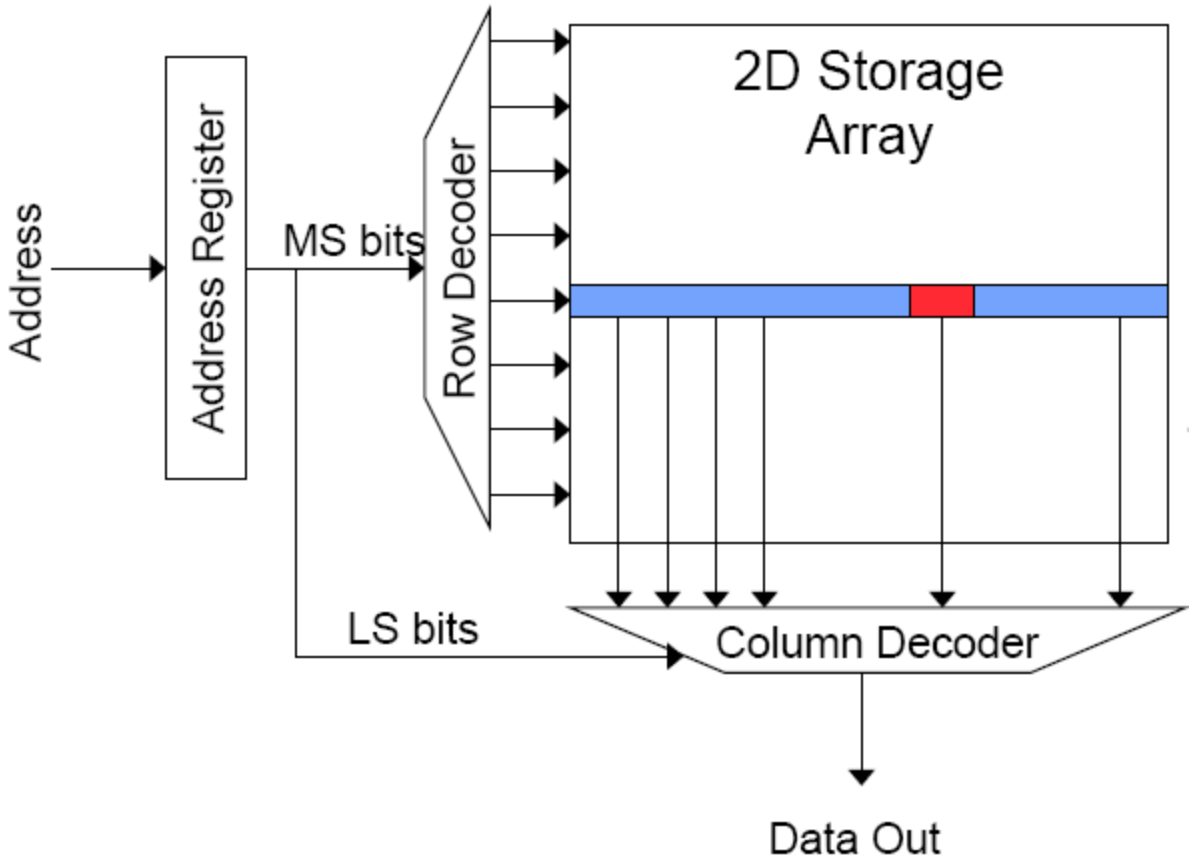
- This is what we will cover in the next lectures

Main Memory Fundamentals

The Memory Chip/System Abstraction



Review: Memory Bank Organization



- Read access sequence:

1. Decode row address & drive word-lines

2. Selected bits drive bit-lines
   - Entire row read

3. Amplify row data

4. Decode column address & select subset of row
   - Send to output

5. Precharge bit-lines
   - For next access

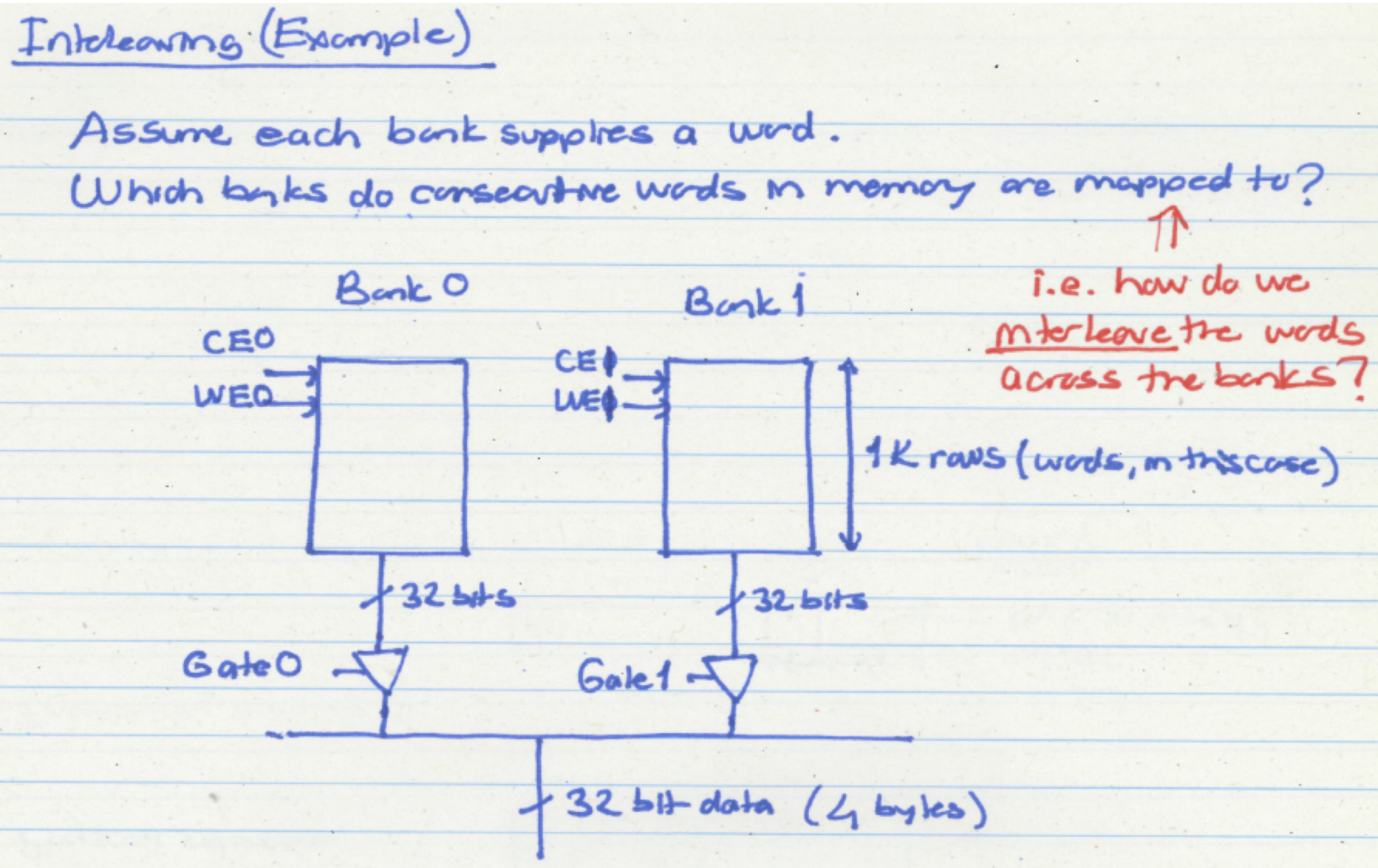
Some Fundamental Concepts (I)

- Physical address space
  - Maximum size of main memory: total number of uniquely identifiable locations
- Physical addressability
  - Minimum size of data in memory can be addressed
  - Byte-addressable, word-addressable, 64-bit-addressable
  - Microarchitectural addressability depends on the abstraction level of the implementation
- Alignment
  - Does the hardware support unaligned access transparently to software?
- Interleaving

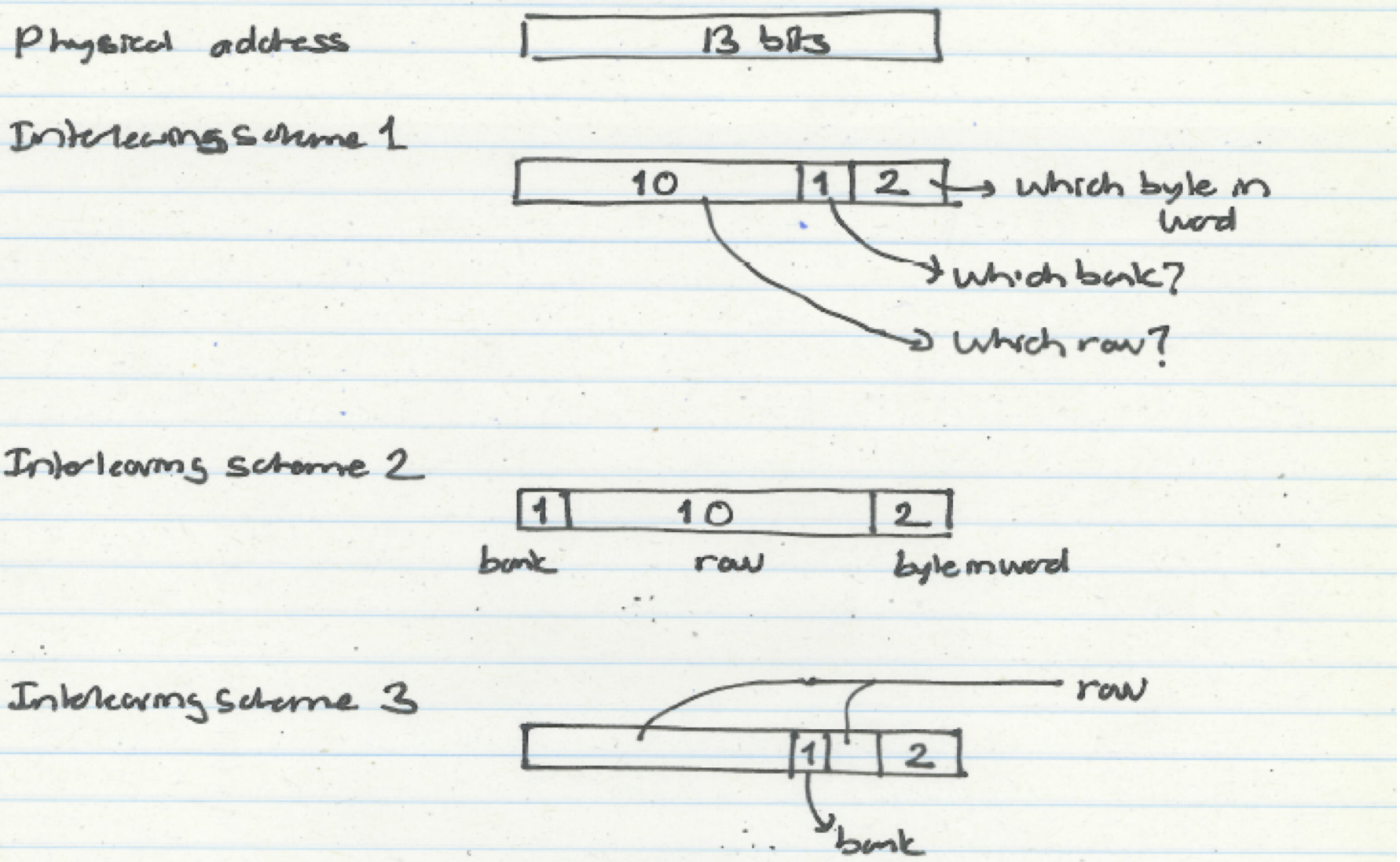
Some Fundamental Concepts (II)

- Interleaving (banking)
  - Problem: a single monolithic memory array takes long to access and does not enable multiple accesses in parallel
  - Goal: Reduce the latency of memory array access and enable multiple accesses in parallel
  - Idea: Divide the array into multiple banks that can be accessed independently (in the same cycle or in consecutive cycles)
    - Each bank is smaller than the entire memory storage
    - Accesses to different banks can be overlapped
  - A Key Issue: How do you map data to different banks? (i.e., how do you interleave data across banks?)

Interleaving



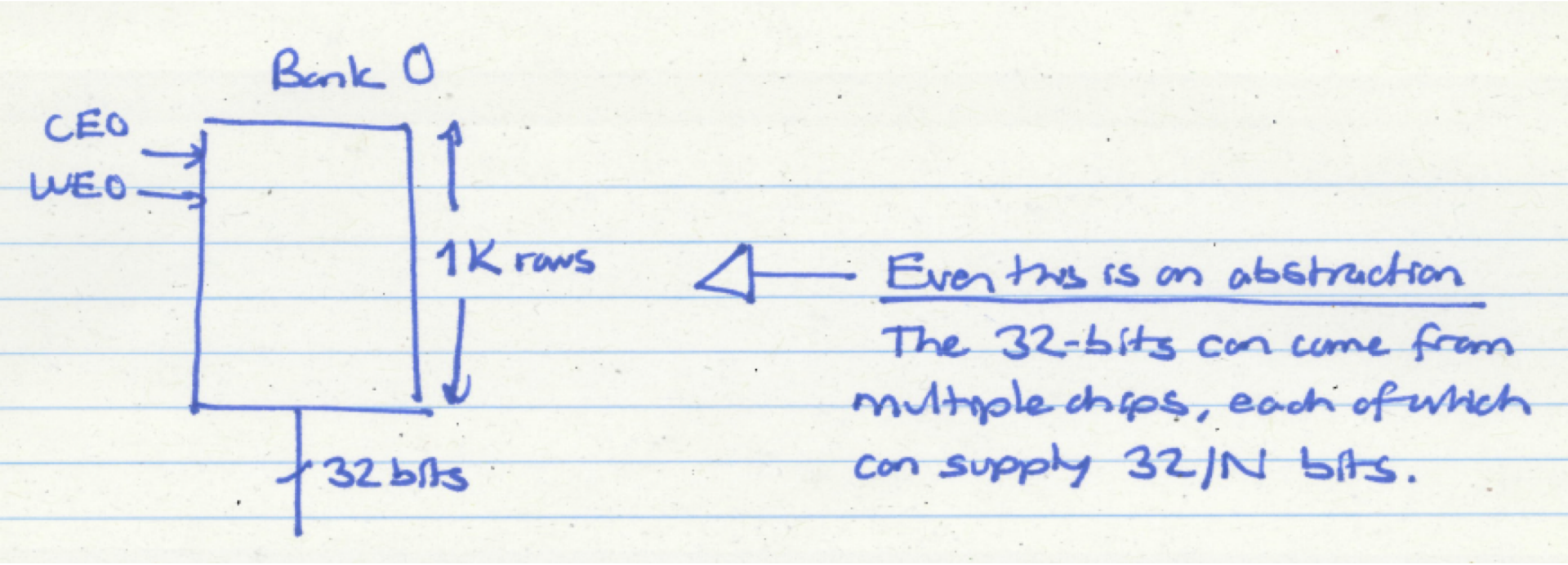
Interleaving Options



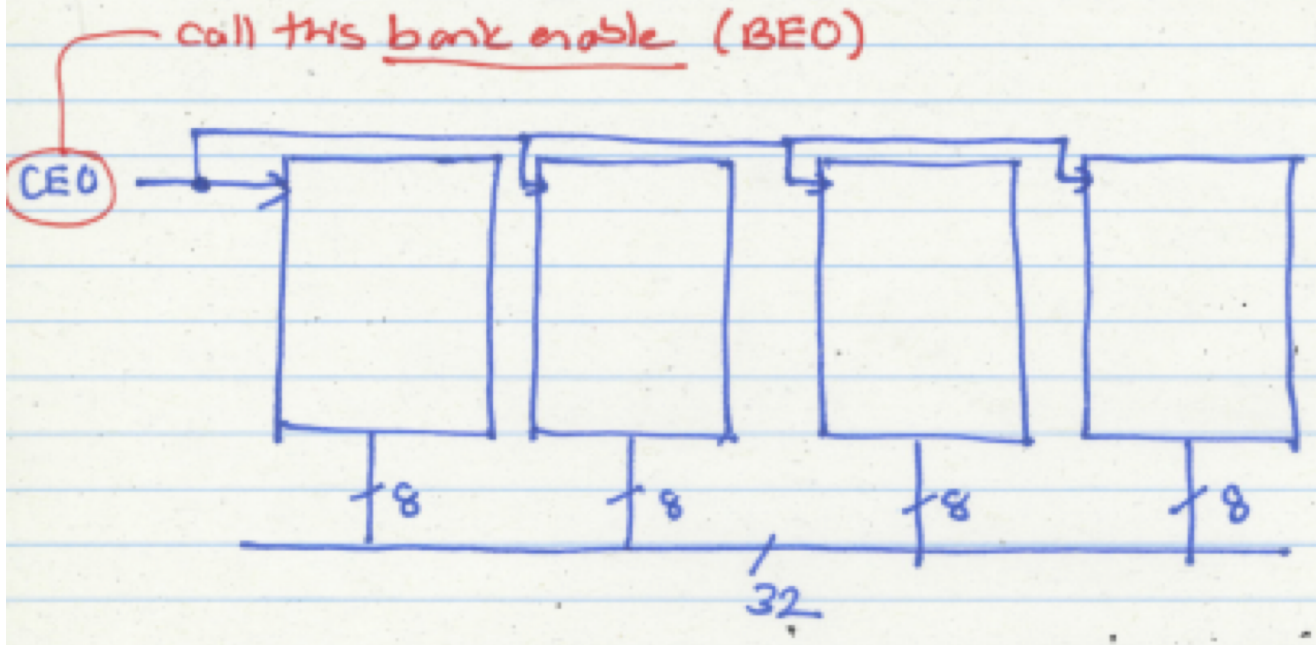
Some Questions/Concepts

- Remember CRAY-1 with 16 banks [From Digital Circuits]
  - 11 cycle bank latency; banks share address/data buses
  - Consecutive words in memory in consecutive banks (word interleaving)
  - 1 access can be started (and finished) per cycle
- Can banks be operated *fully* in parallel?
  - Multiple accesses started per cycle?
- What is the cost of this?
  - We have seen it earlier
- Modern superscalar processors have L1 data caches with multiple, fully-independent banks; DRAM banks share buses

The Bank Abstraction



Rank



This is called a "rank." (only bank 0 of the rank shown here)

Rank: A set of chips that respond to the same command & same address at the same time with different pieces of the requested data

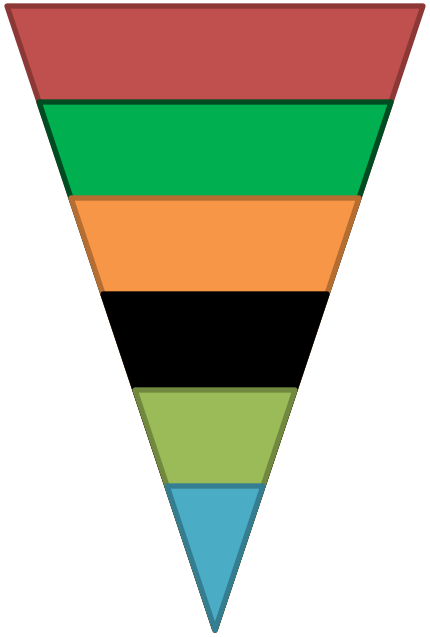
Why? Producing an 8-bit/pin chip cheaper than producing a 32-bit/pin chip

Idea: Produce an 8-bit/pin chip, but control/operate them as a rank so that we can get 32 bits in a single read.

The DRAM Subsystem

DRAM Subsystem Organization

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

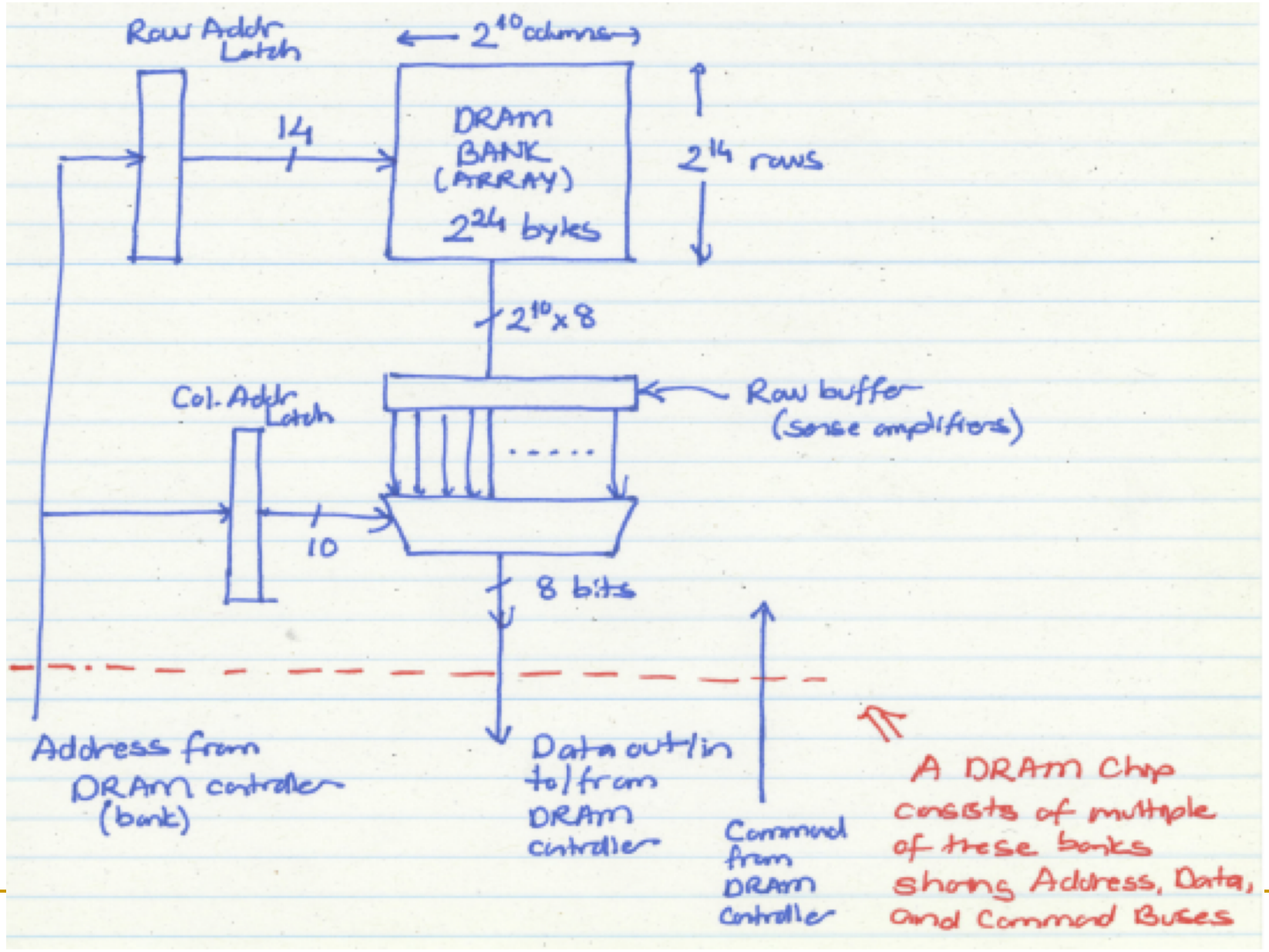


Page Mode DRAM

- A DRAM bank is a 2D array of cells: rows x columns
- A "DRAM row" is also called a "DRAM page"
- "Sense amplifiers" also called "row buffer"

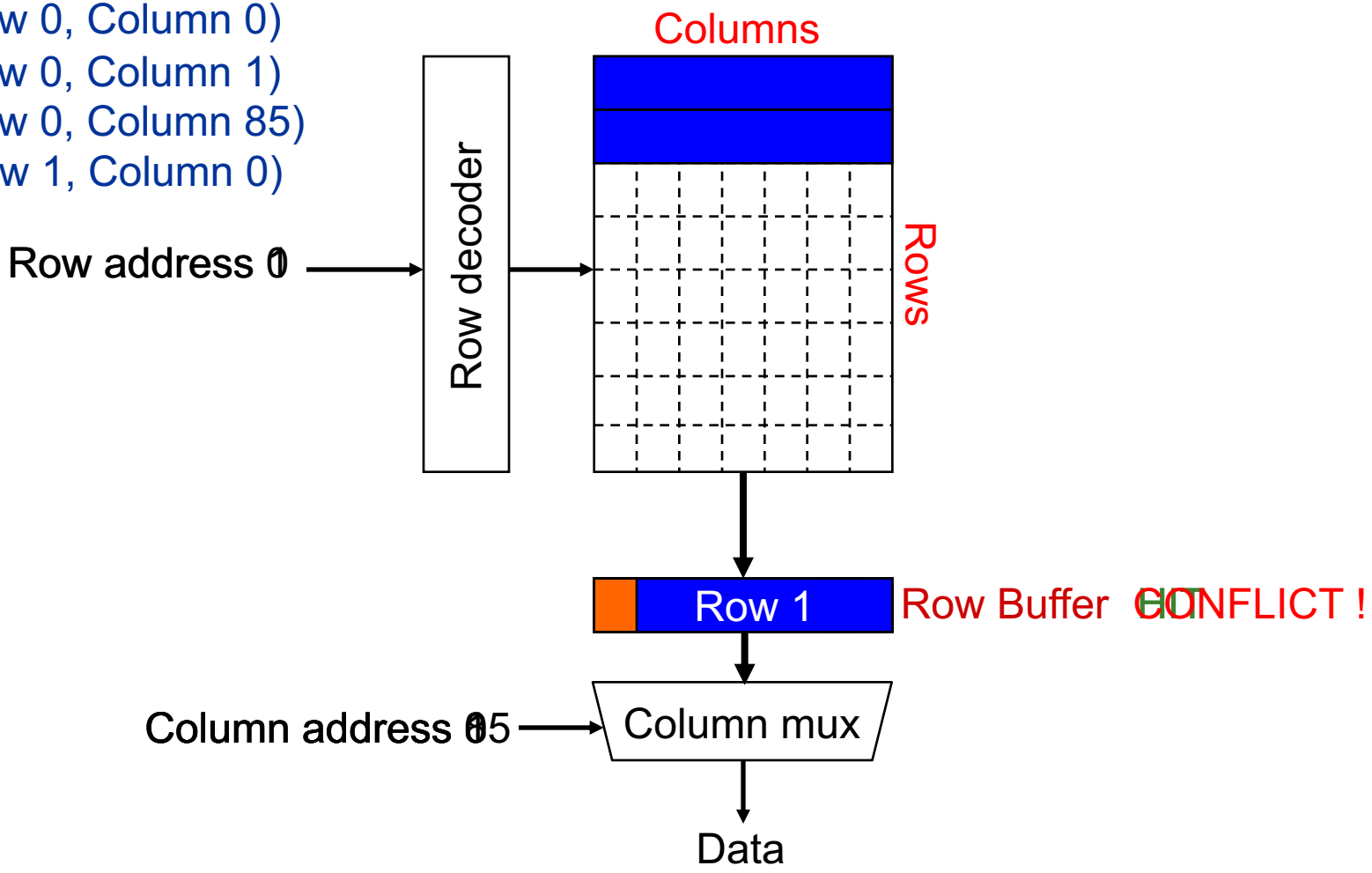
- Each address is a <row,column> pair
- Access to a "closed row"
  - Activate command opens row (placed into row buffer)
  - Read/write command reads/writes column in the row buffer
  - Precharge command closes the row and prepares the bank for next access
- Access to an "open row"
  - No need for activate command

The DRAM Bank Structure



DRAM Bank Operation

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

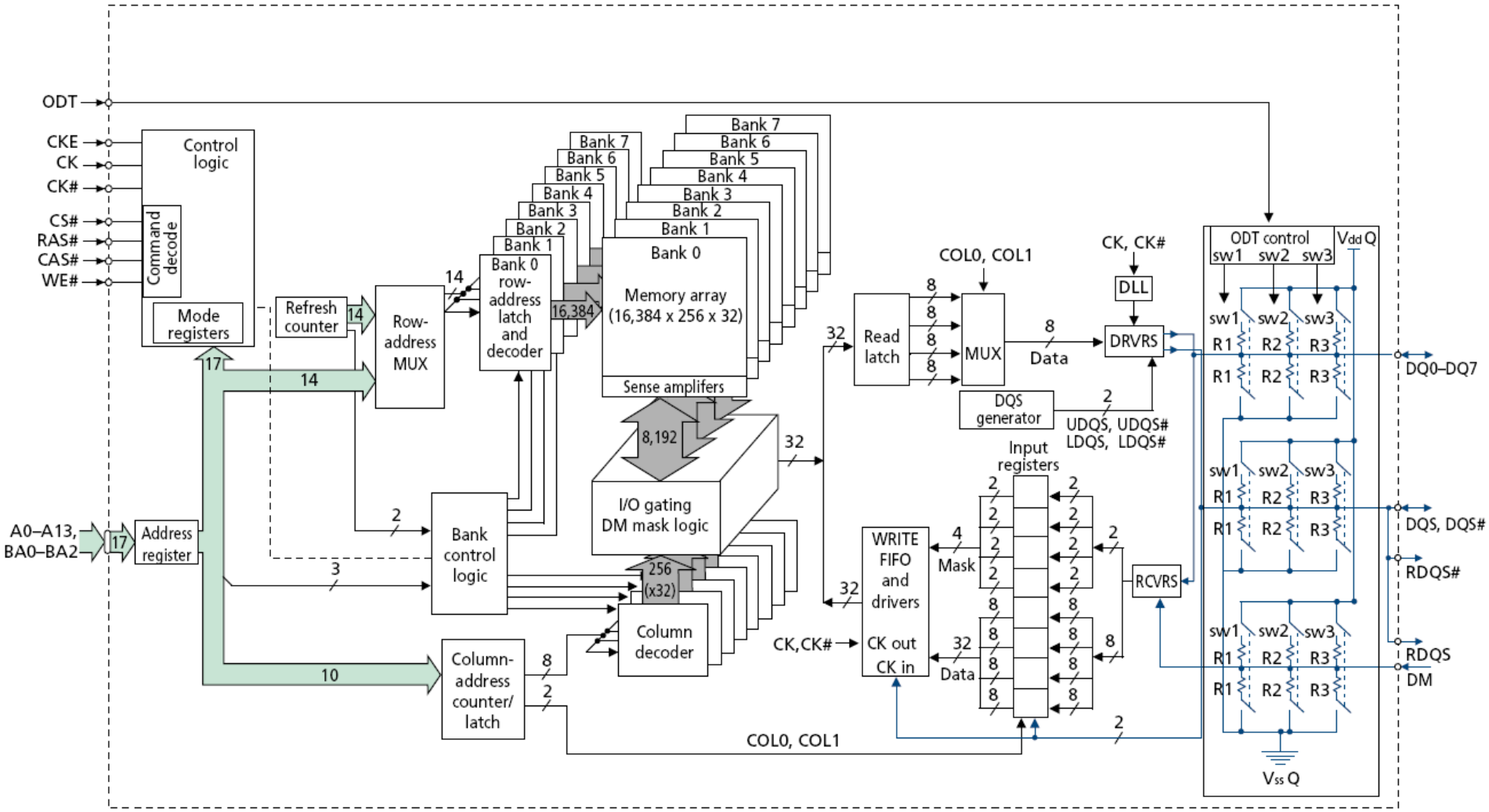


The DRAM Chip

- Consists of multiple banks (8 is a common number today)
- Banks share command/address/data buses
- The chip itself has a narrow interface (4-16 bits per read)

- Changing the number of banks, size of the interface (pins), whether or not command/address/data buses are shared has significant impact on DRAM system cost

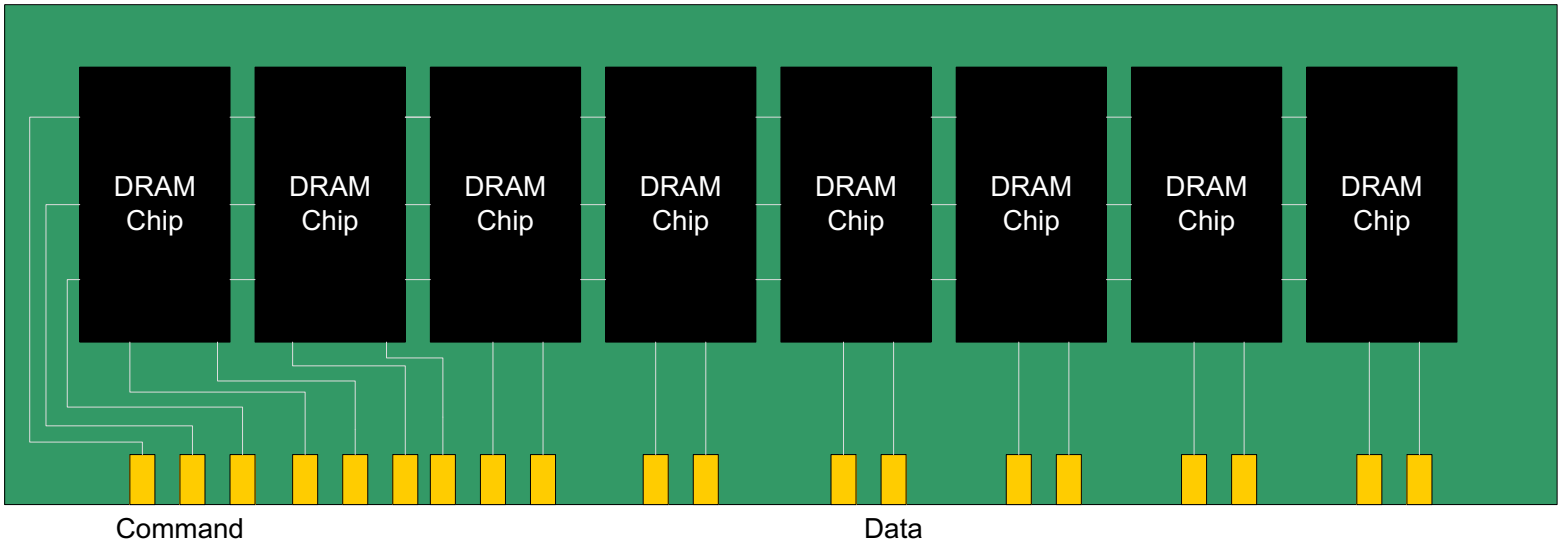
128M x 8-bit DRAM Chip



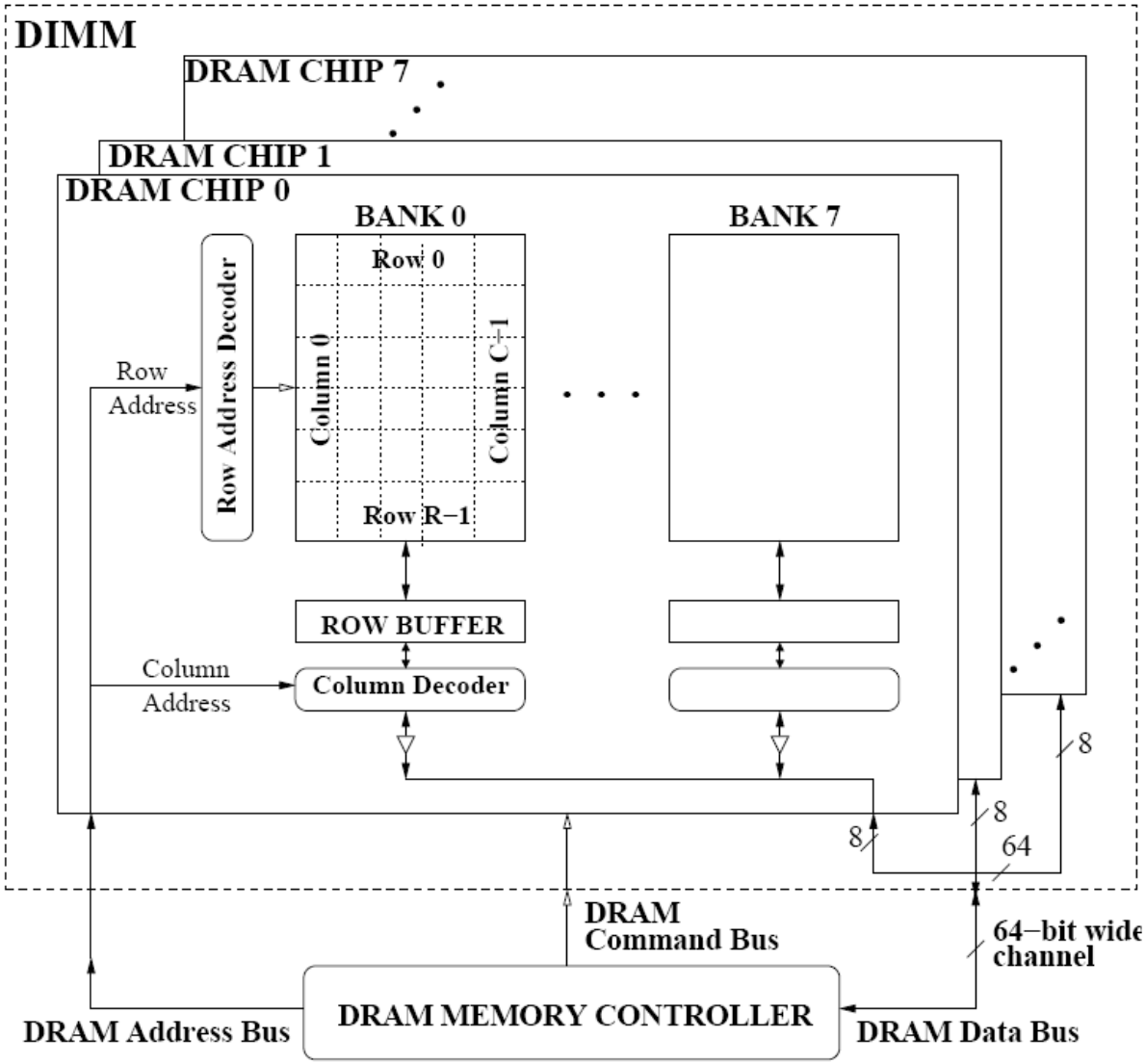
DRAM Rank and Module

- Rank: Multiple chips operated together to form a wide interface
- All chips comprising a rank are controlled at the same time
  - Respond to a single command
  - Share address and command buses, but provide different data
- A DRAM module consists of one or more ranks
  - E.g., DIMM (dual inline memory module)
  - This is what you plug into your motherboard
- If we have chips with 8-bit interface, to read 8 bytes in a single access, use 8 chips in a DIMM

A 64-bit Wide DIMM (One Rank)

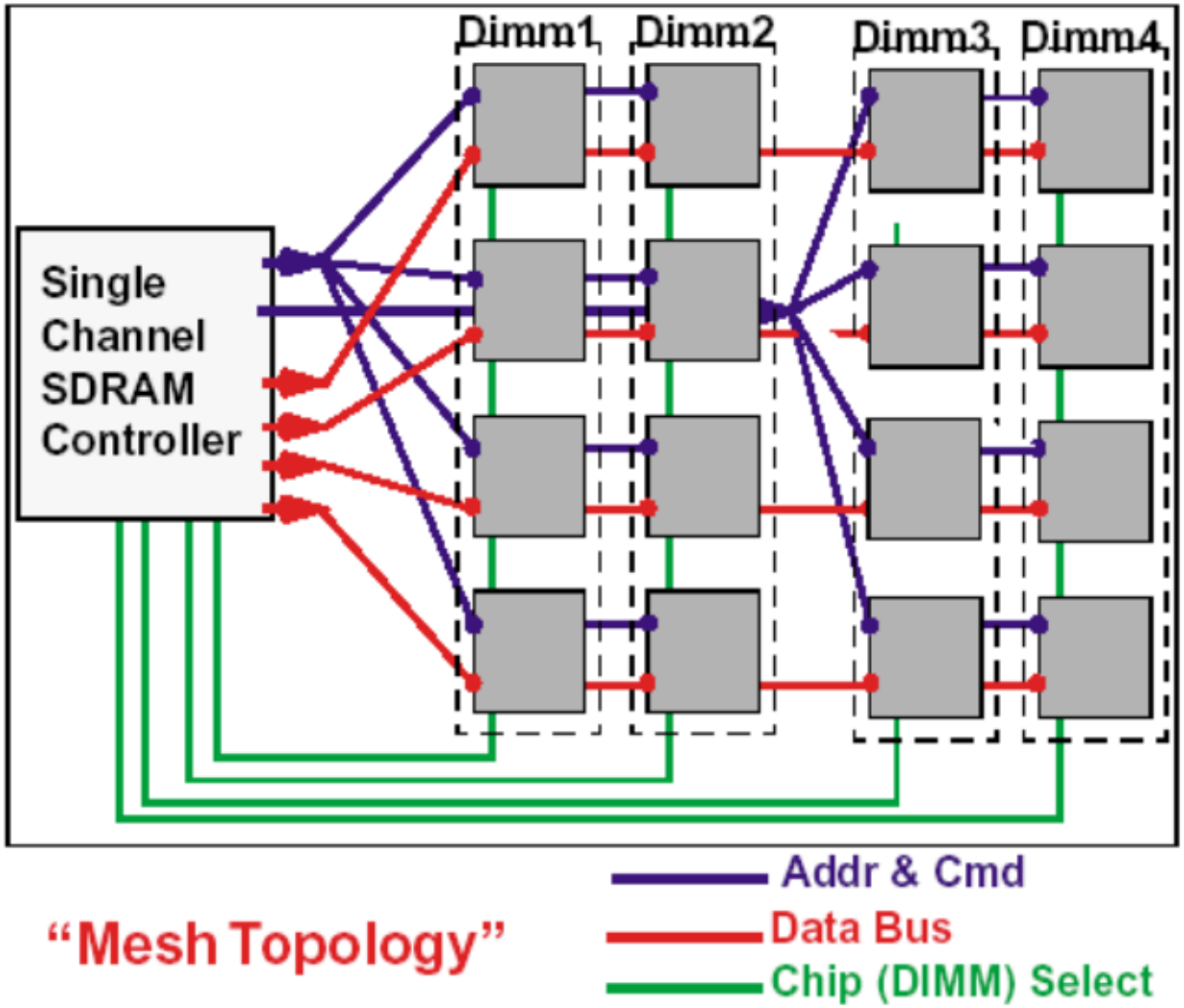


A 64-bit Wide DIMM (One Rank)



- Advantages:
  - Acts like a high-capacity DRAM chip with a wide interface
  - Flexibility: memory controller does not need to deal with individual chips
- Disadvantages:
  - Granularity: Accesses cannot be smaller than the interface width

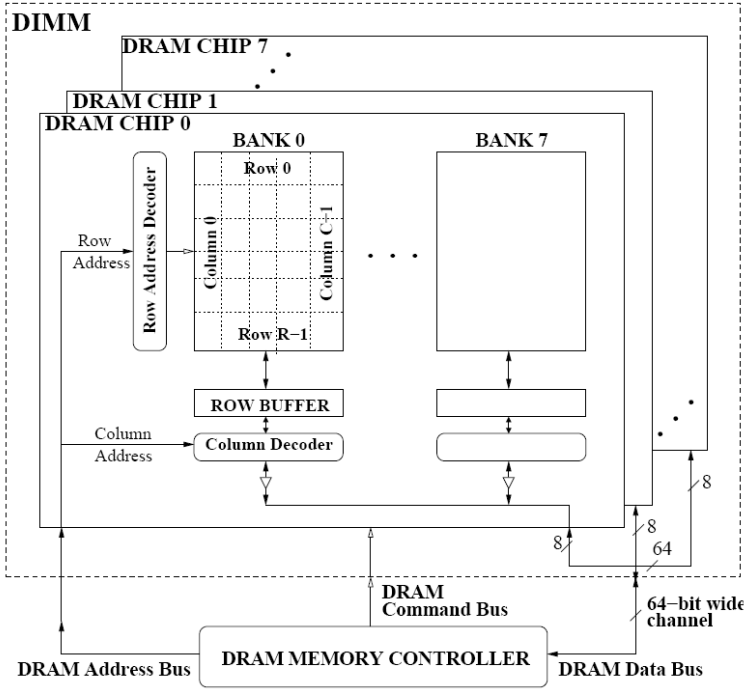
Multiple DIMMs

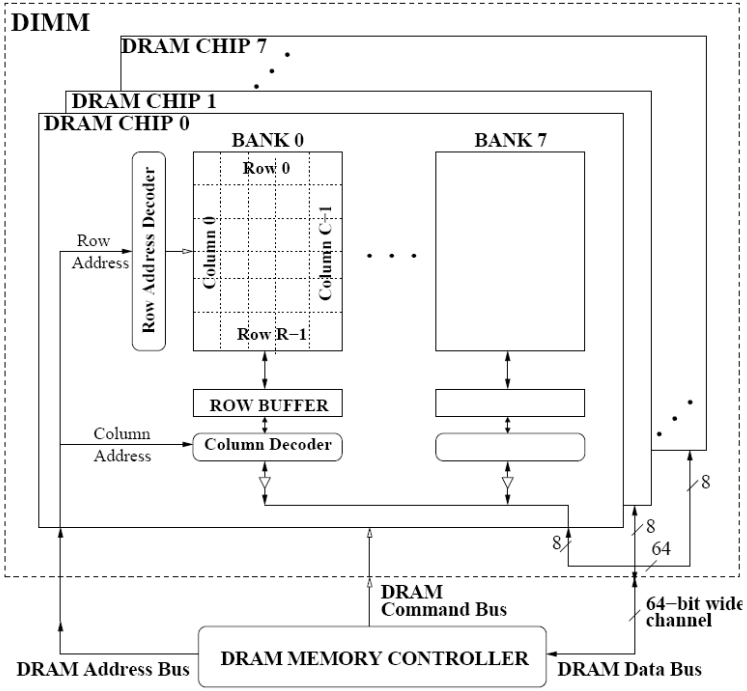


- Advantages:
  - Enables even higher capacity
- Disadvantages:
  - Interconnect complexity and energy consumption can be high

  → Scalability is limited by this

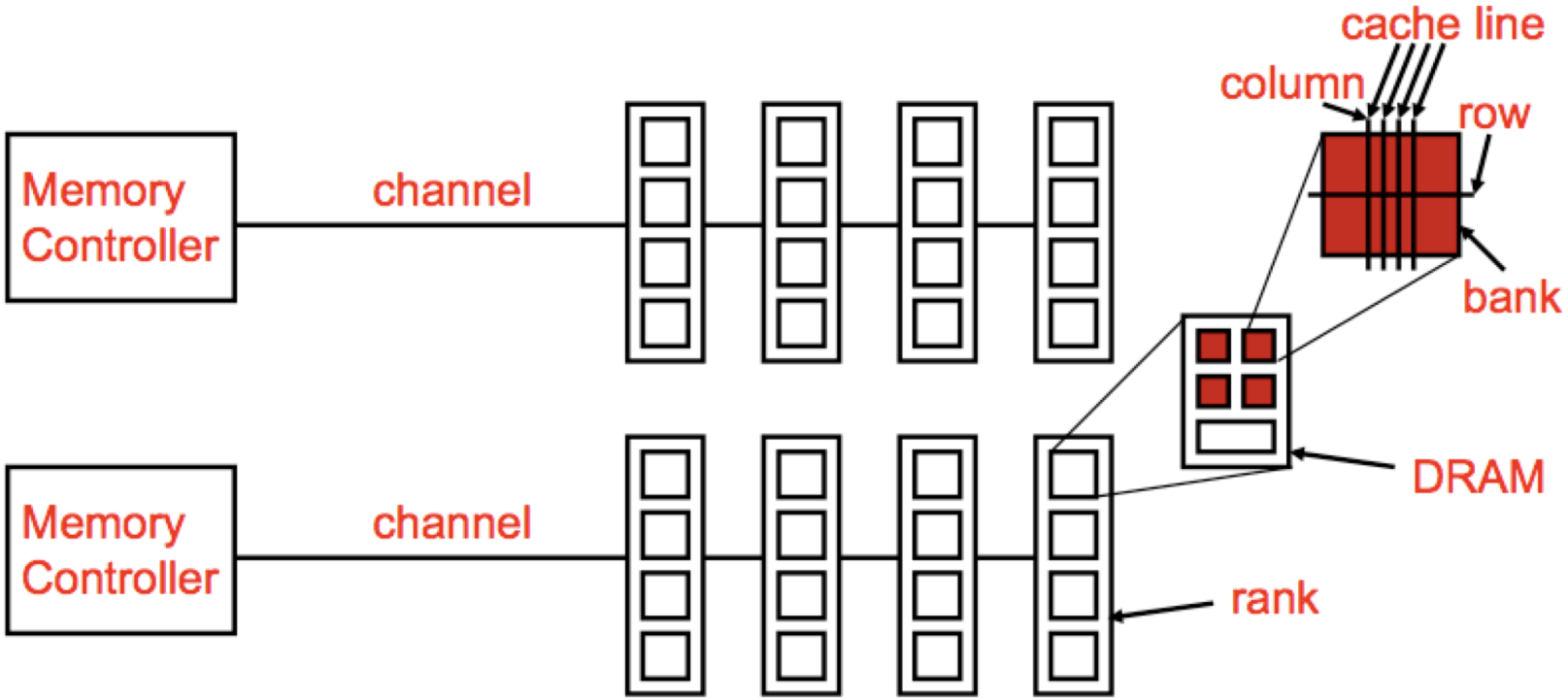
DRAM Channels



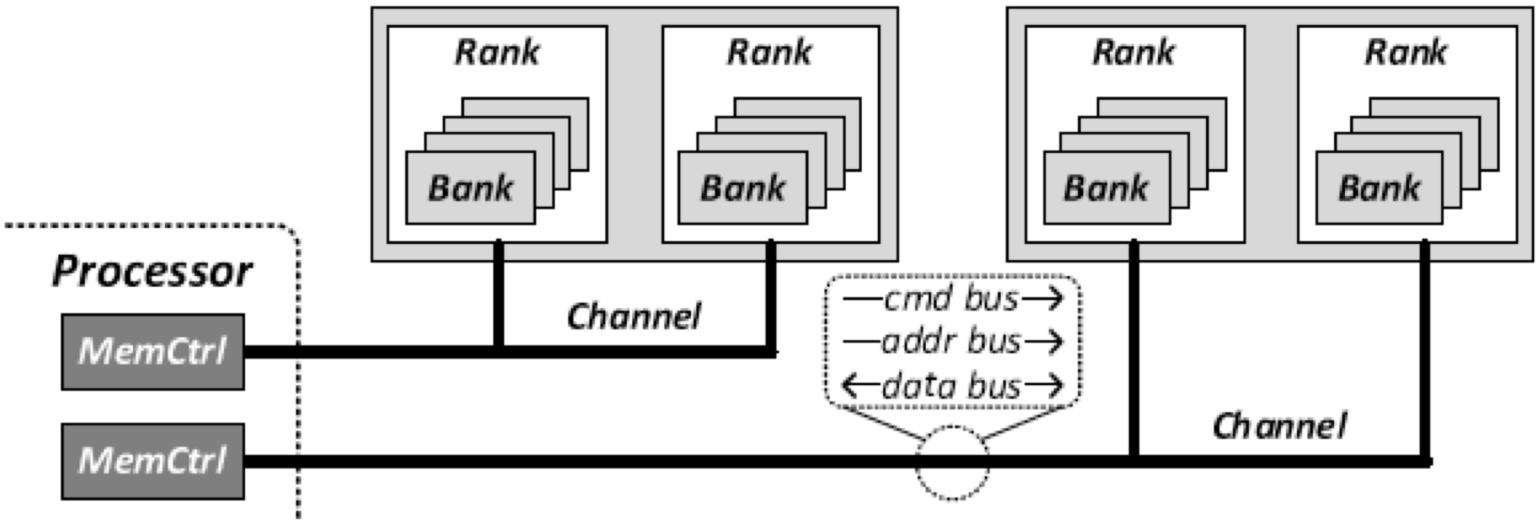


- 2 Independent Channels: 2 Memory Controllers (Above)
- 2 Dependent/Lockstep Channels: 1 Memory Controller with wide interface (Not shown above)

Generalized Memory Structure



Generalized Memory Structure



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

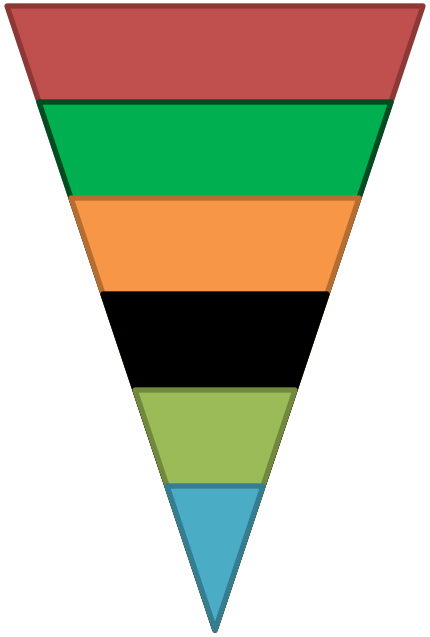
Required Readings on DRAM

- 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

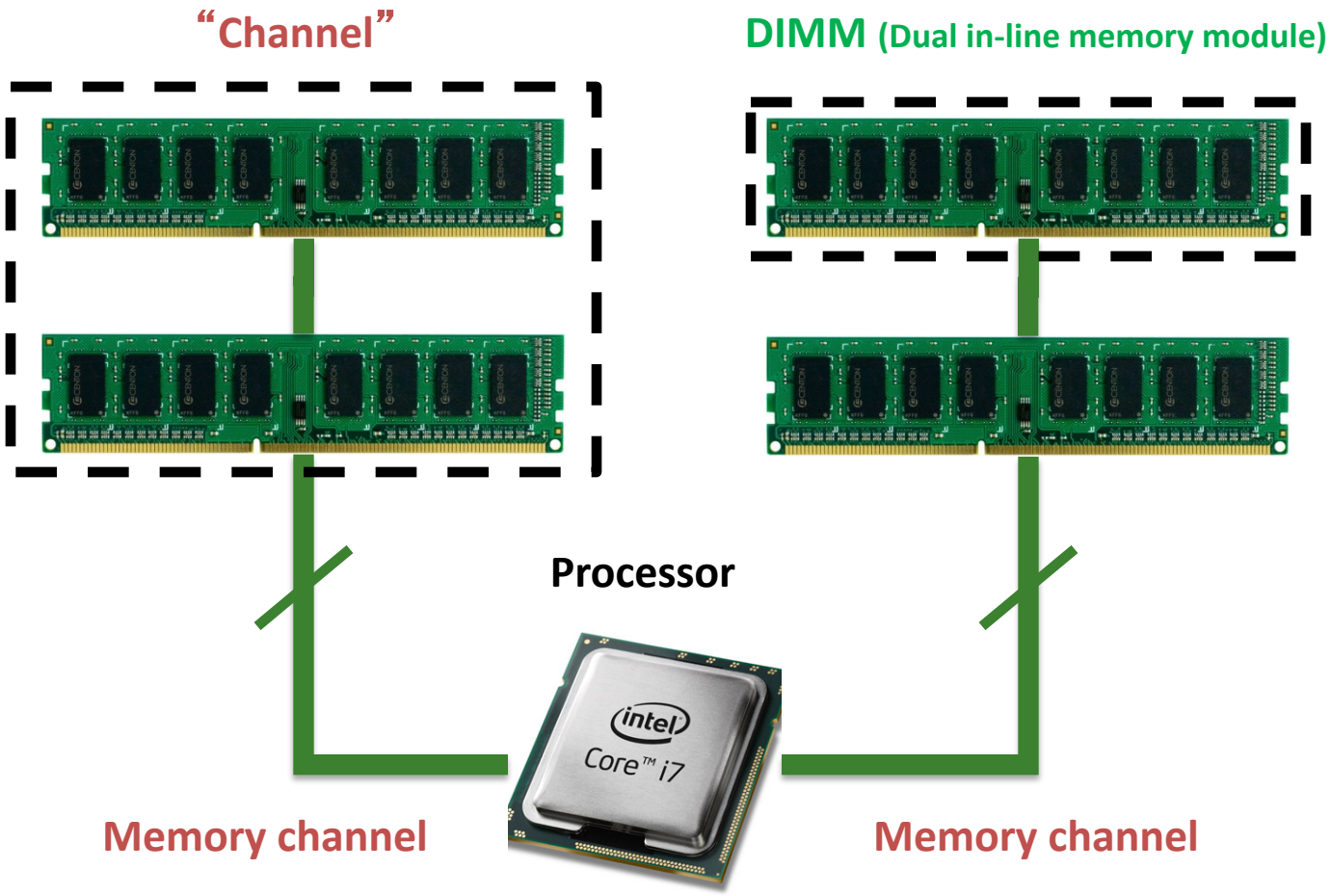
The DRAM Subsystem
The Top Down View

DRAM Subsystem Organization

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



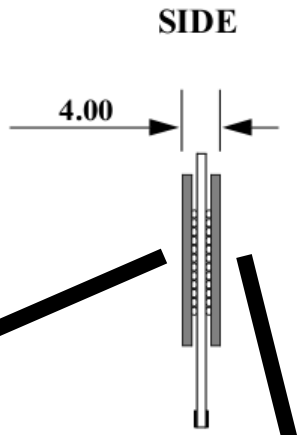
The DRAM subsystem



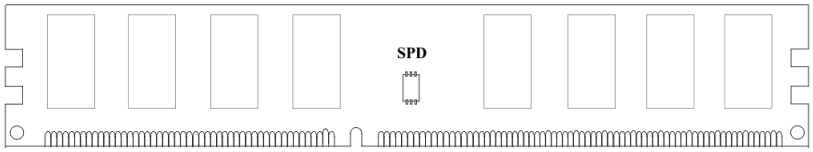
Breaking down a DIMM

DIMM (Dual in-line memory module)

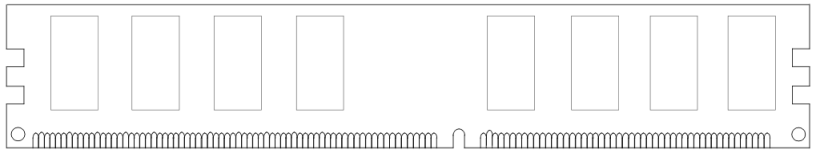
Side view



Front of DIMM



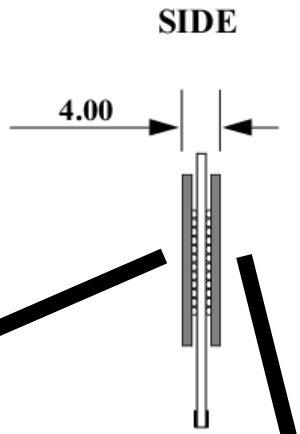
Back of DIMM



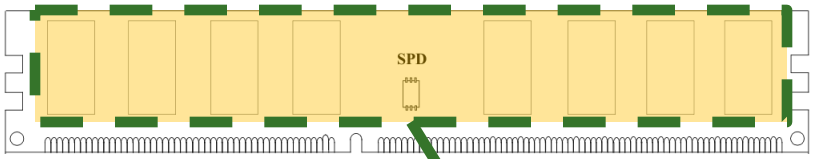
Breaking down a DIMM

DIMM (Dual in-line memory module)

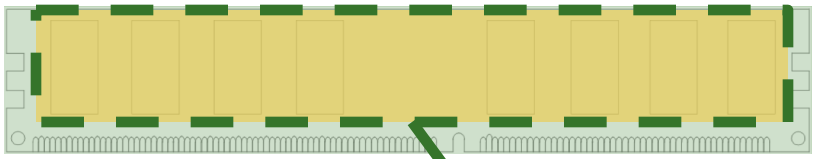
Side view



Front of DIMM



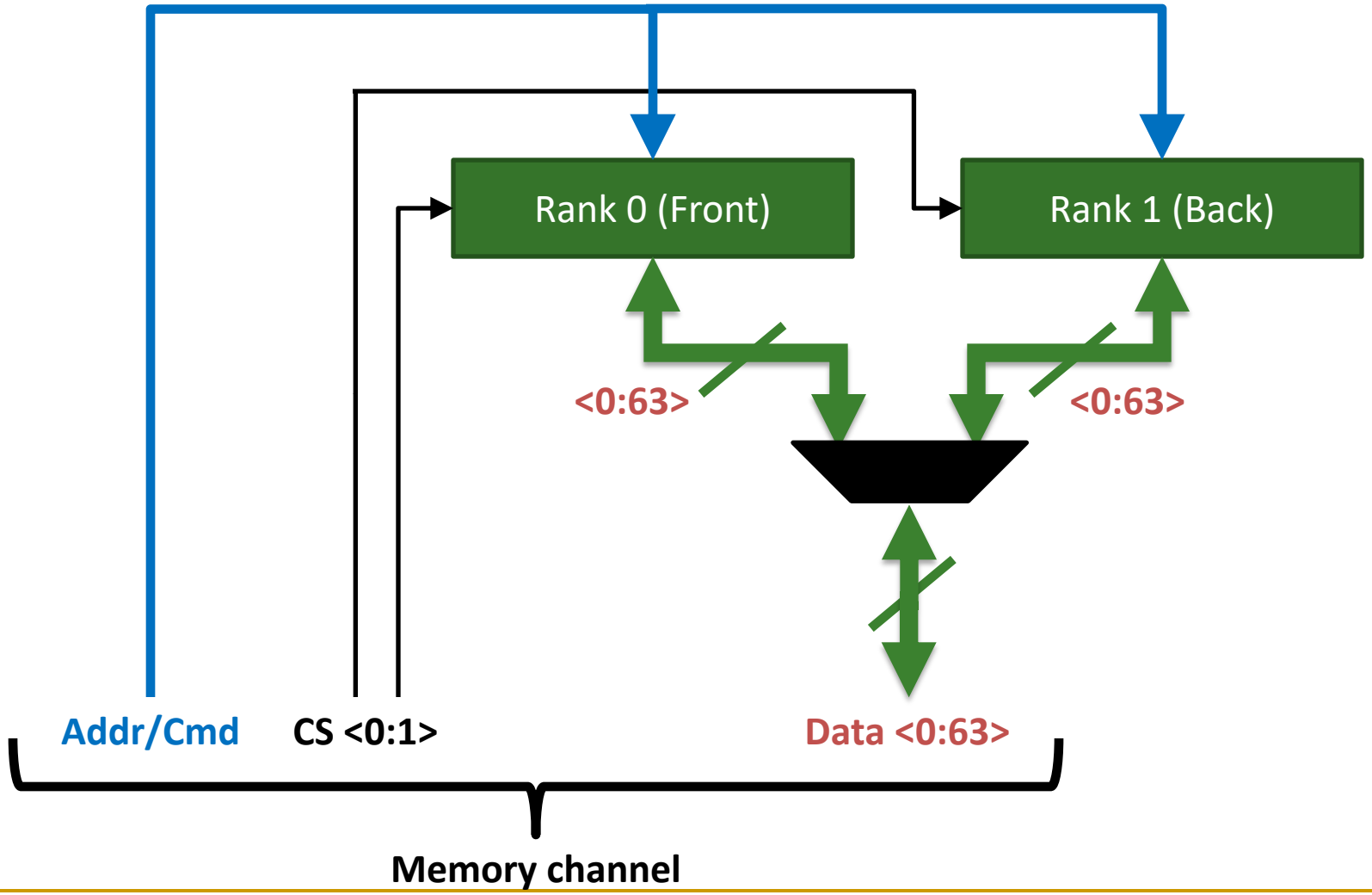
Back of DIMM



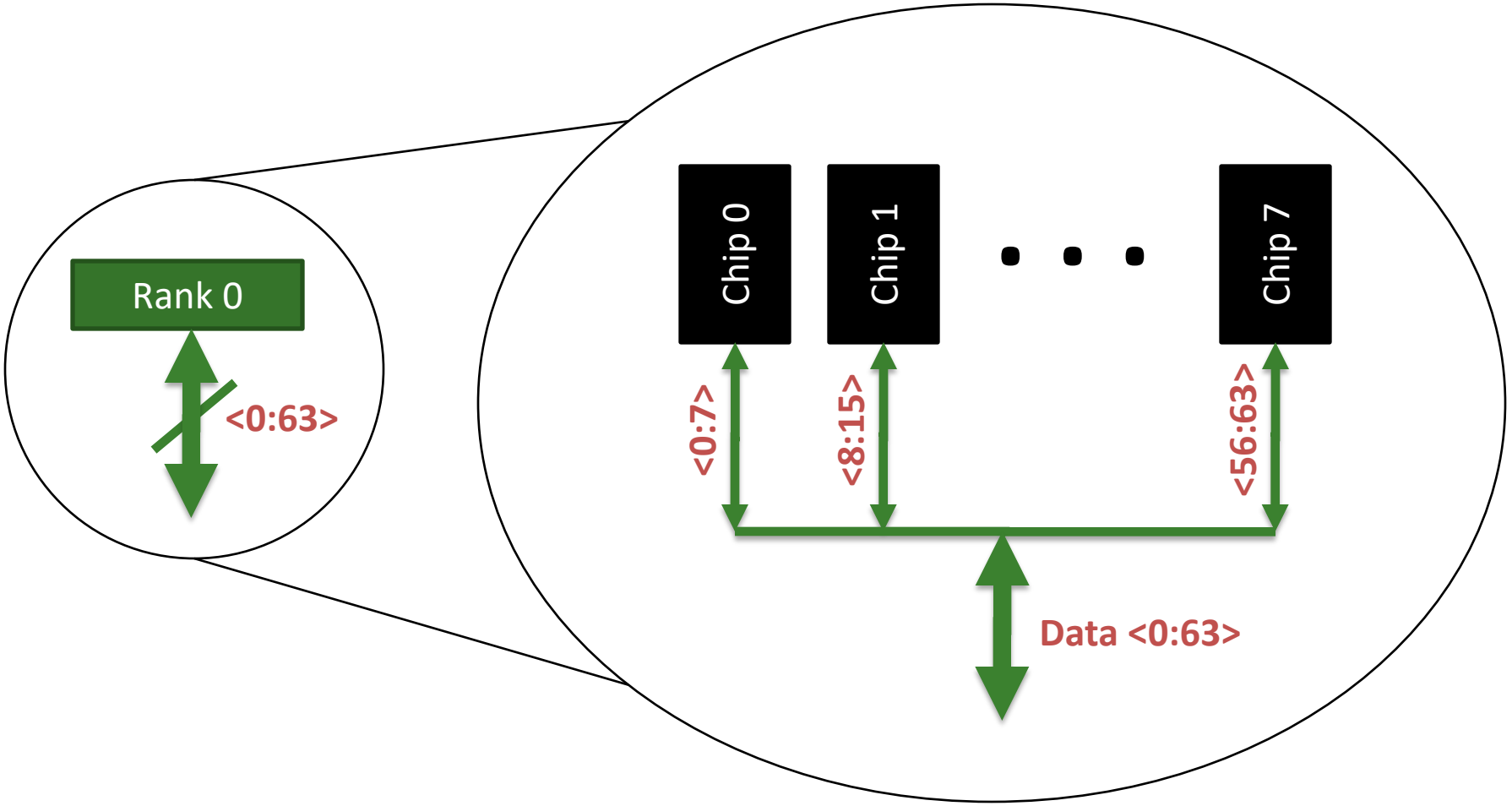
Rank 0: collection of 8 chips

Rank 1

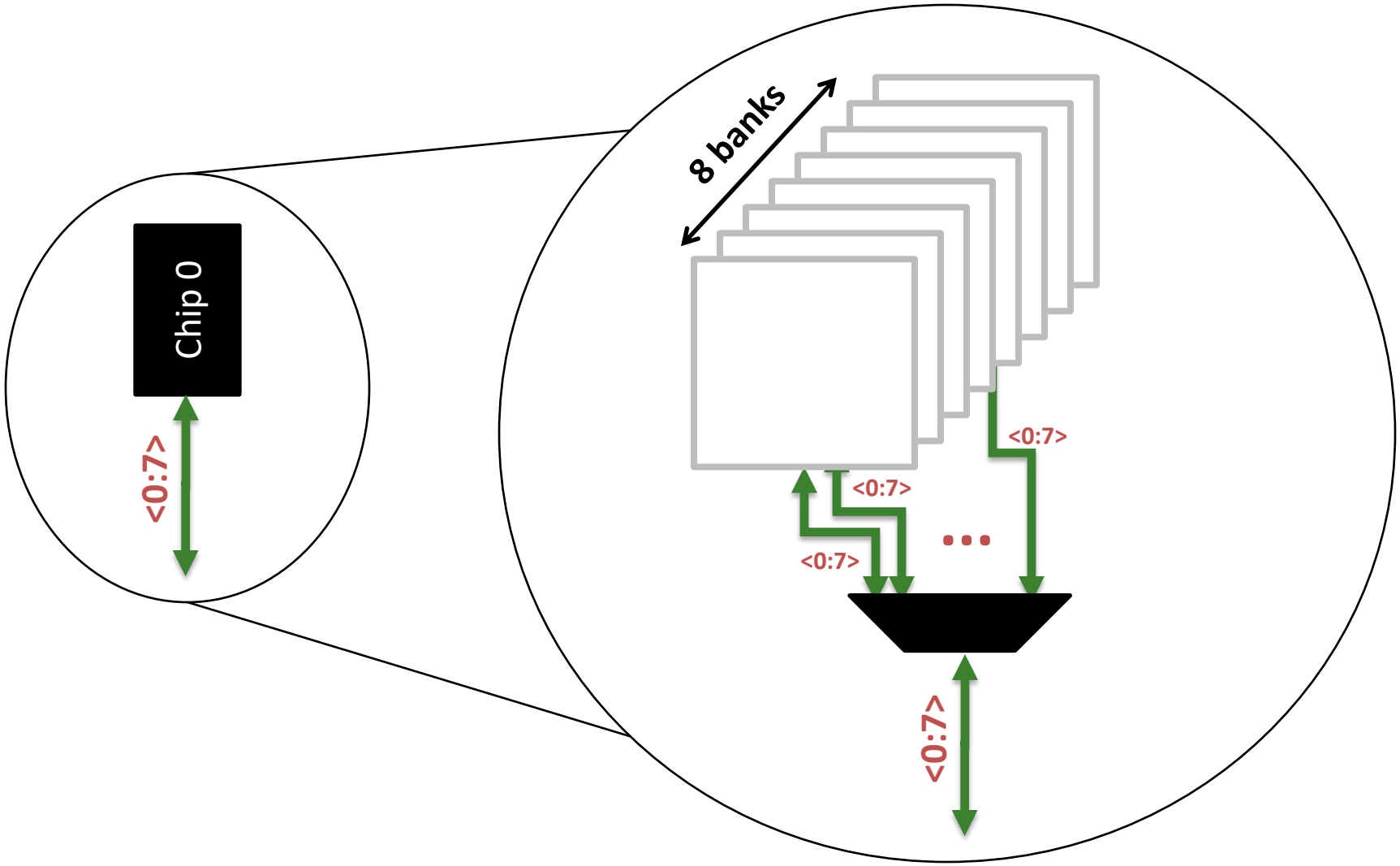
Rank


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

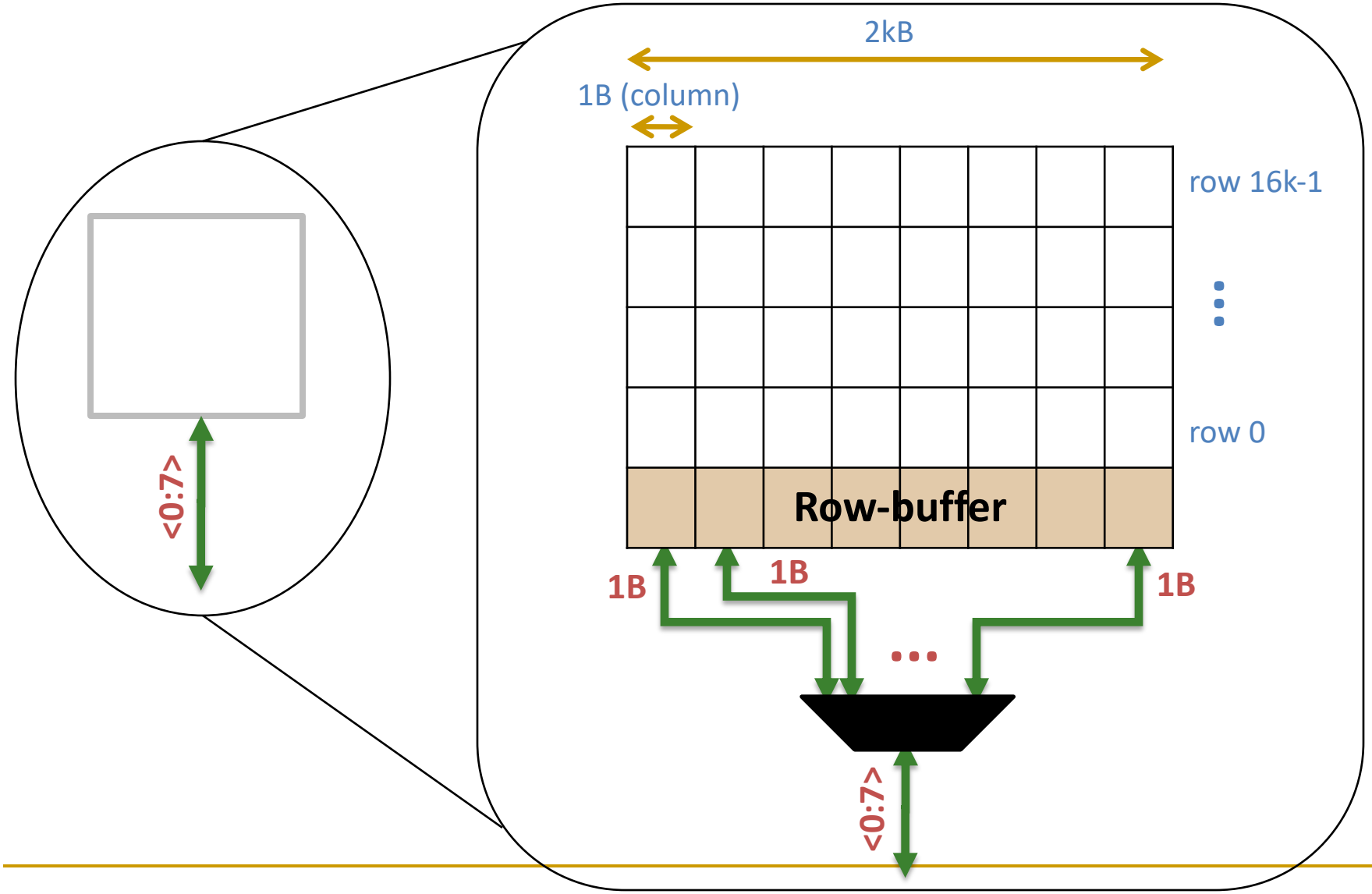
Breaking down a Rank



Breaking down a Chip

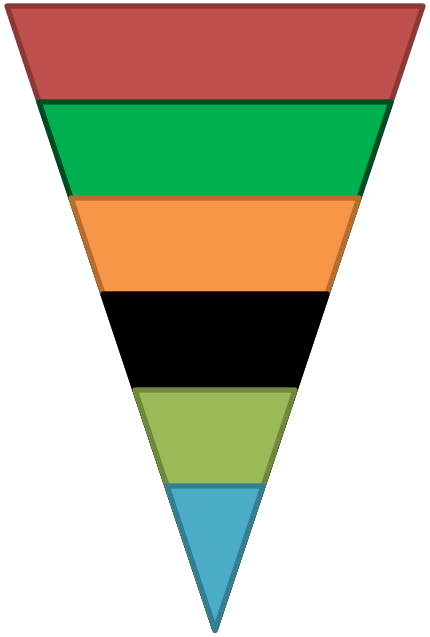


Breaking down a Bank

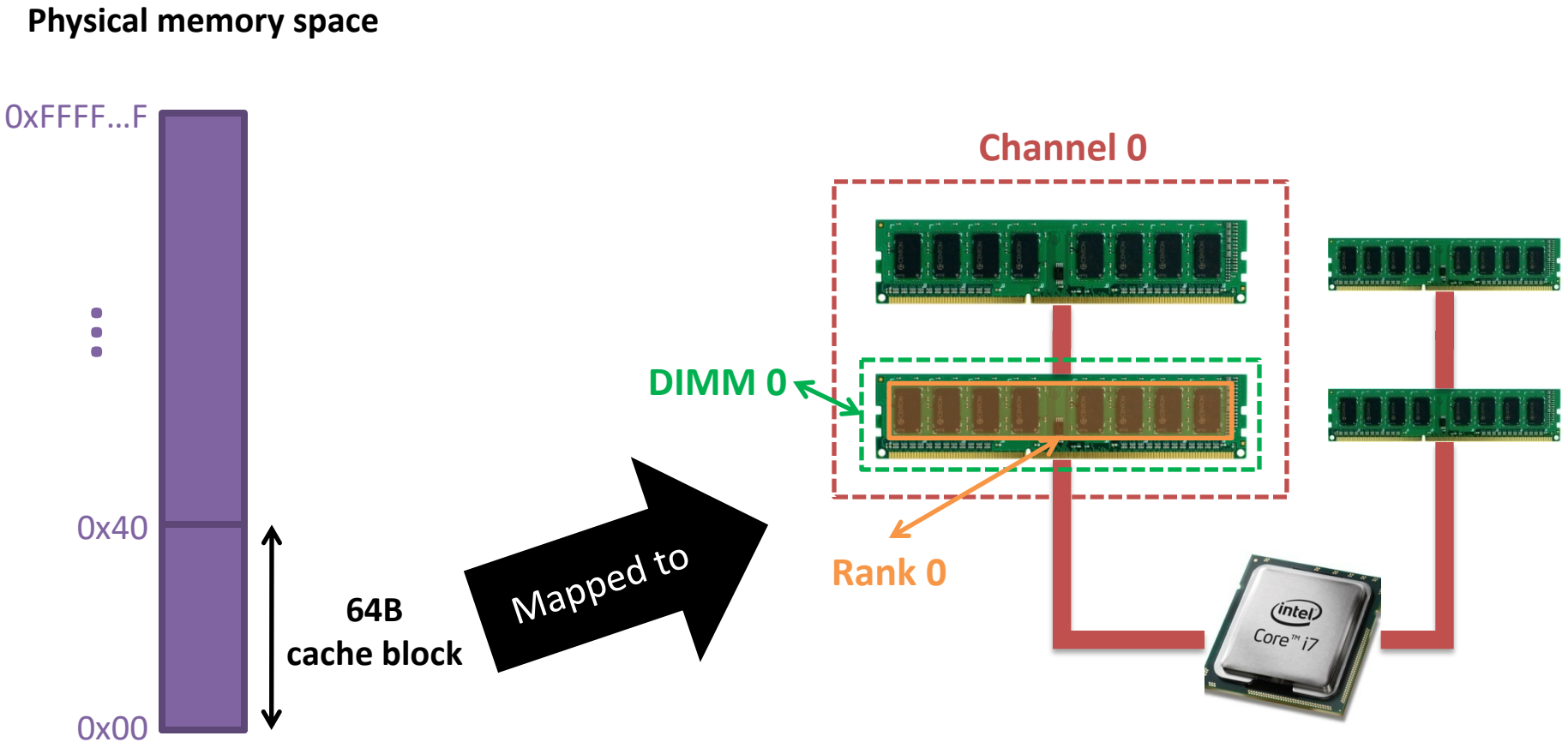


DRAM Subsystem Organization

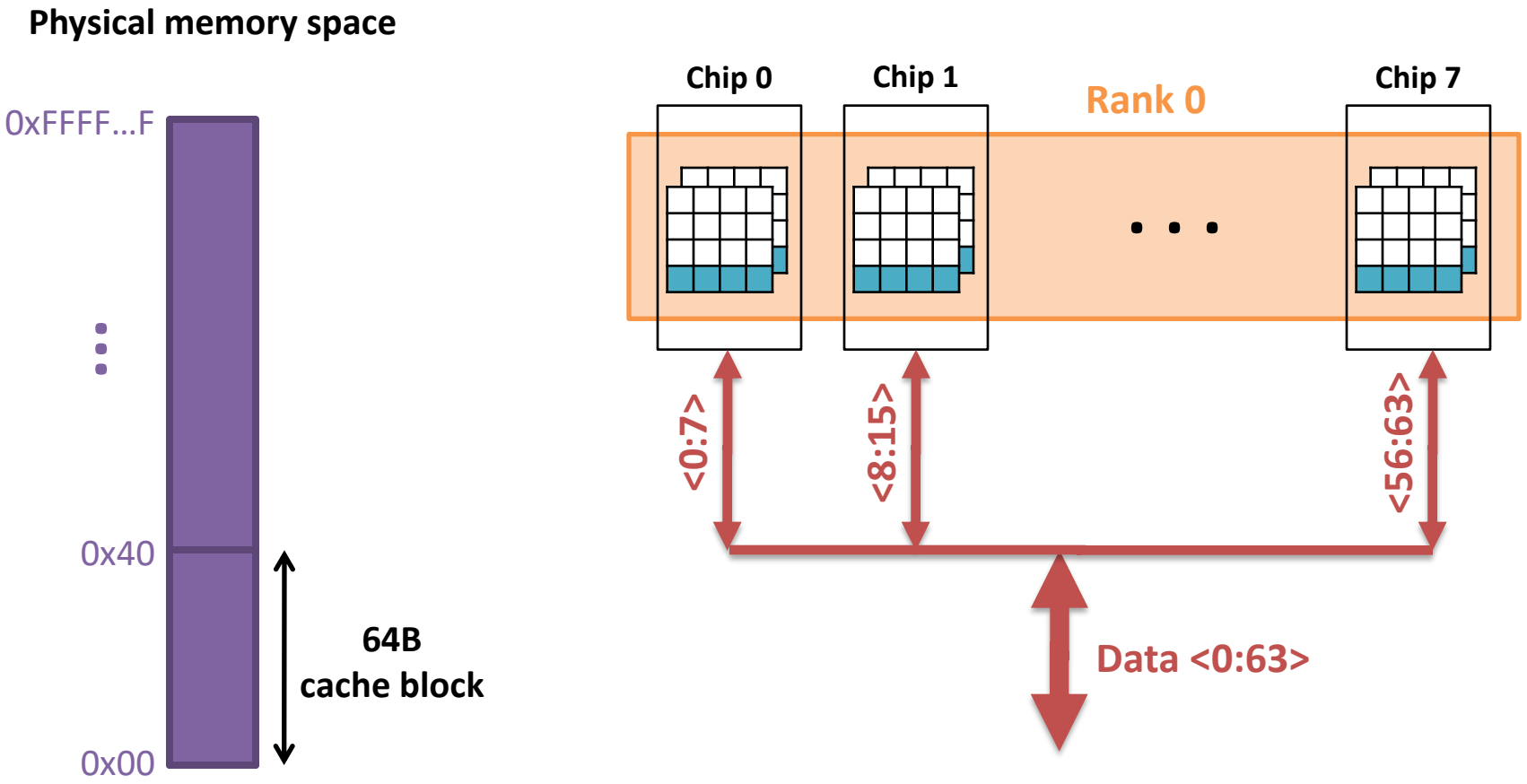
- Channel
- DIMM
- Rank
- Chip
- Bank
- Row/Column
- Cell



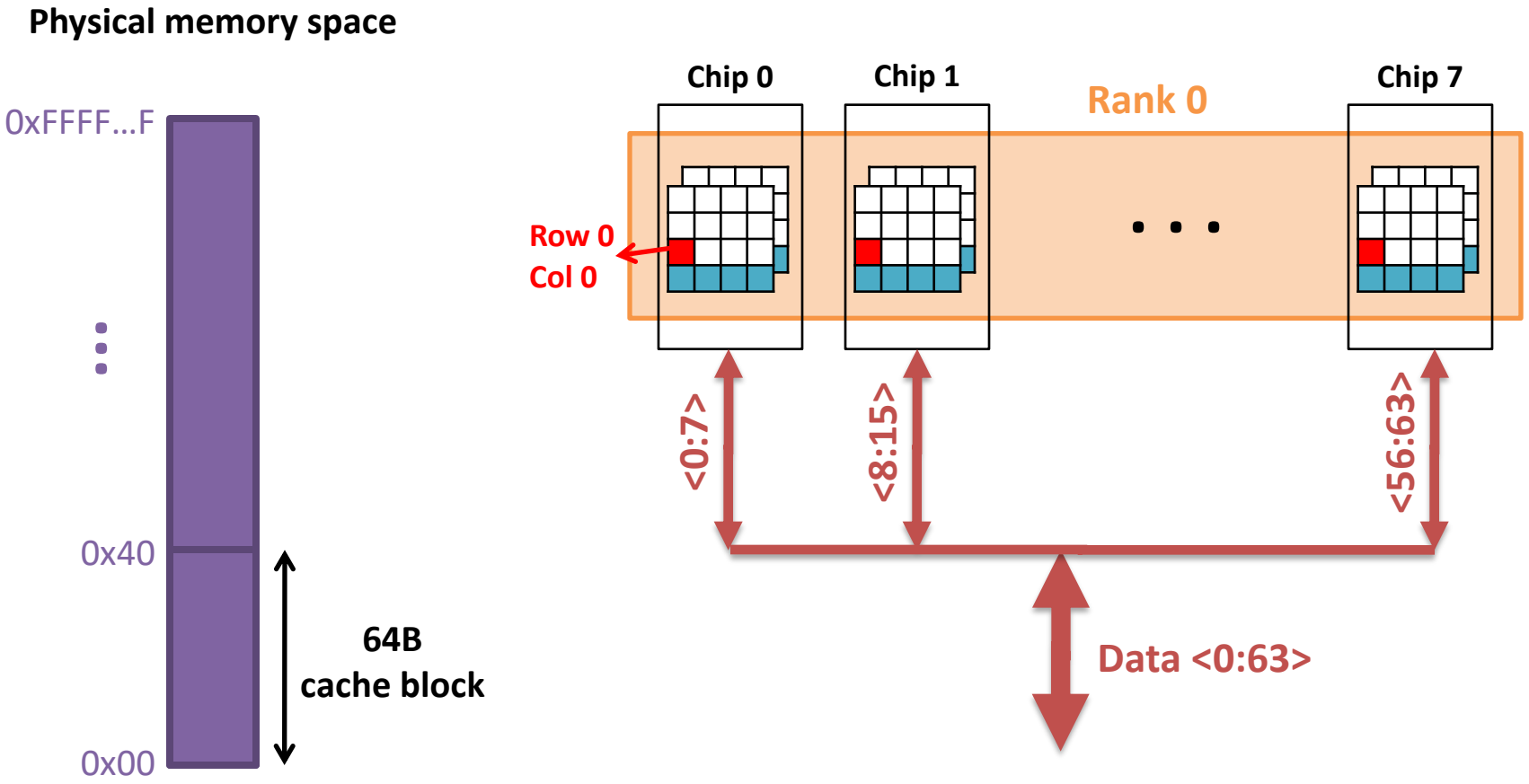
Example: Transferring a cache block



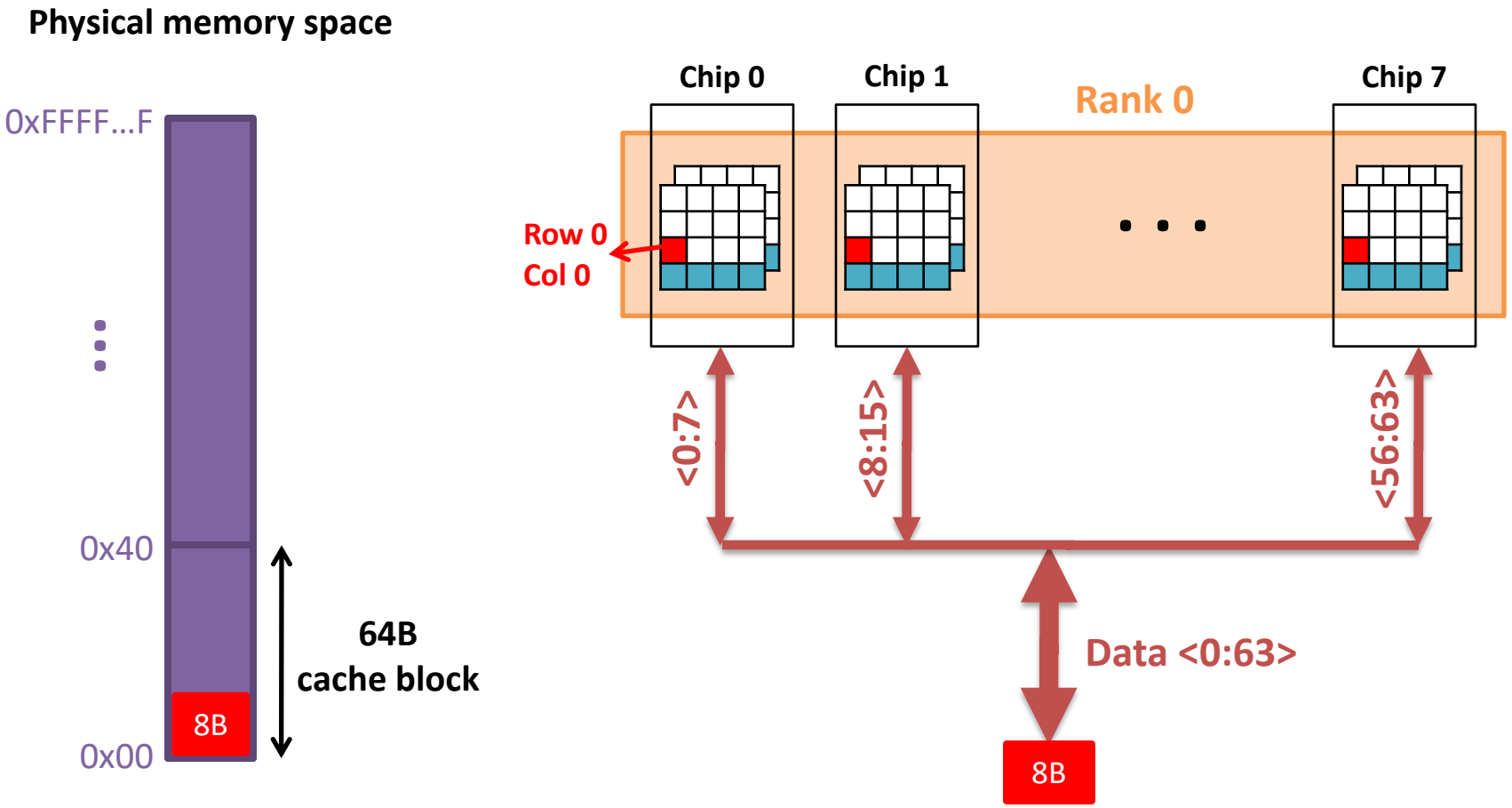
Example: Transferring a cache block



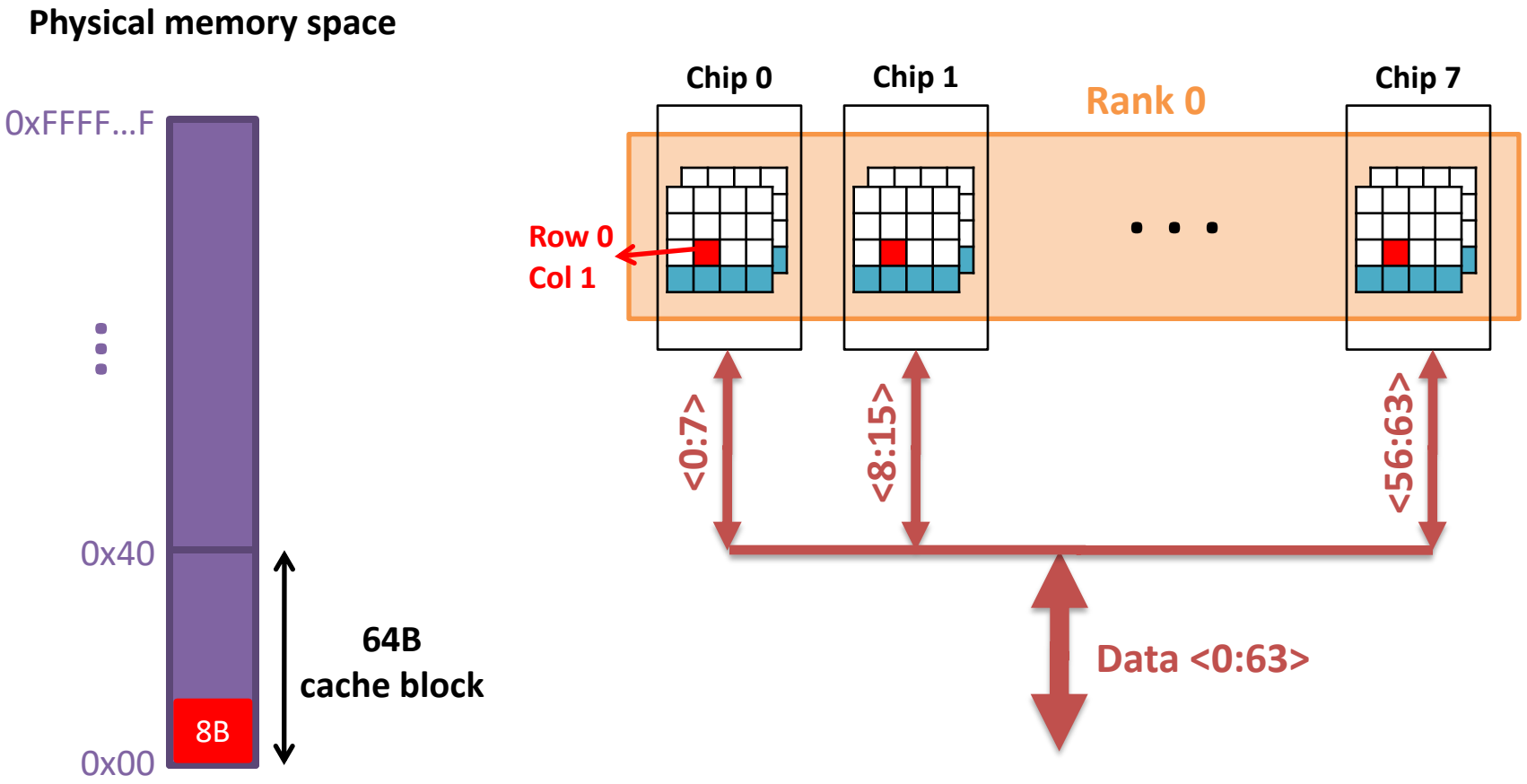
Example: Transferring a cache block



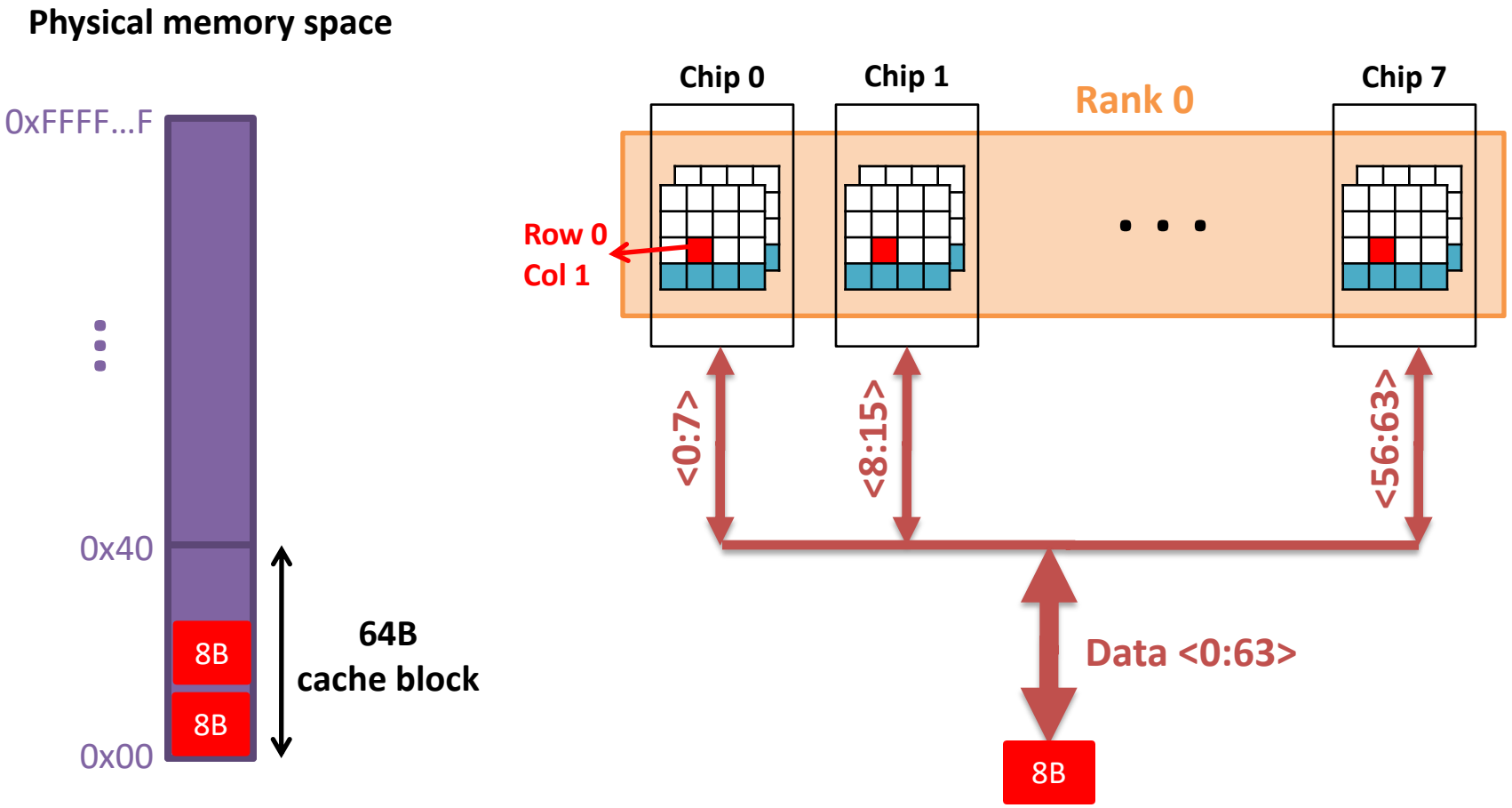
Example: Transferring a cache block



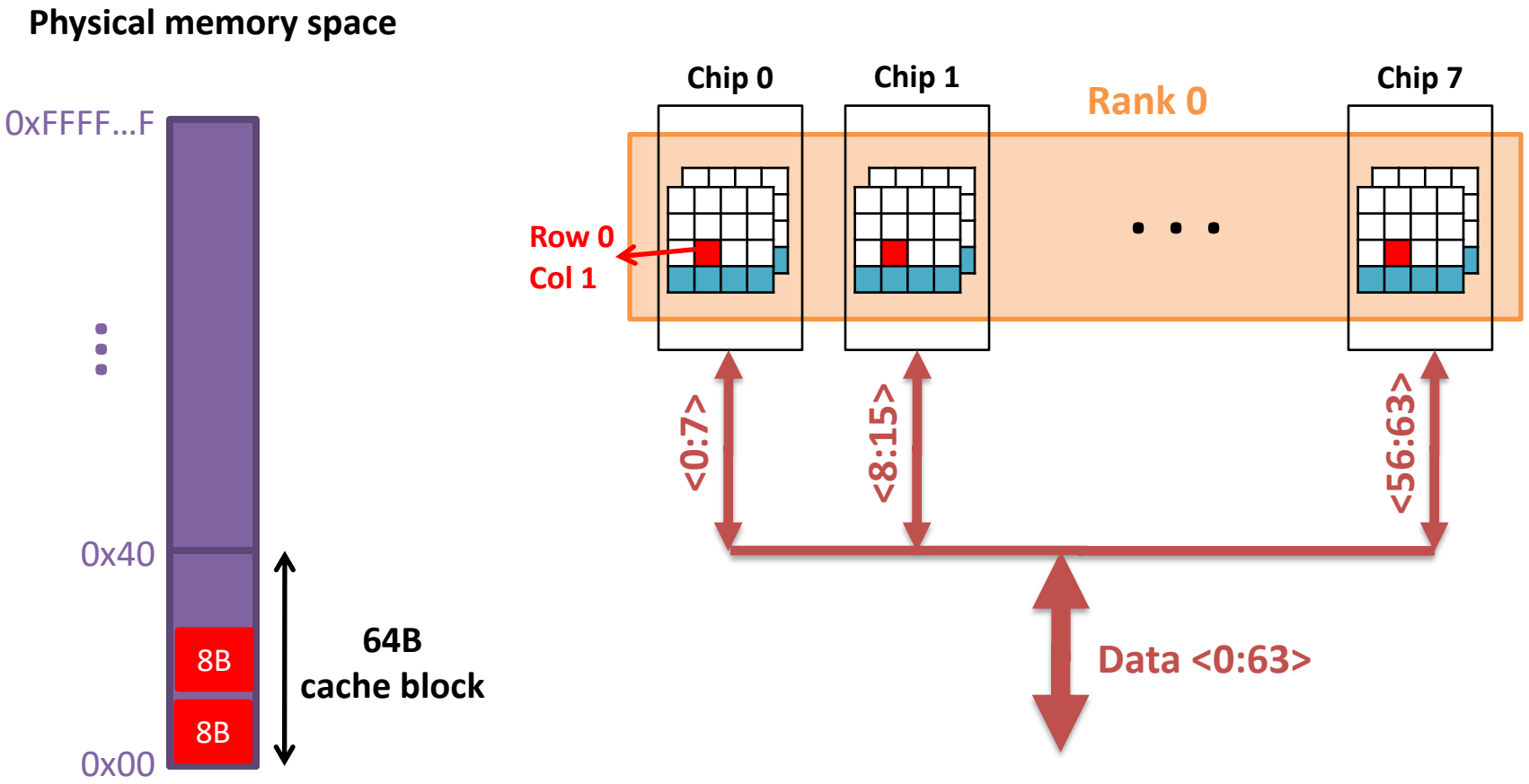
Example: Transferring a cache block



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

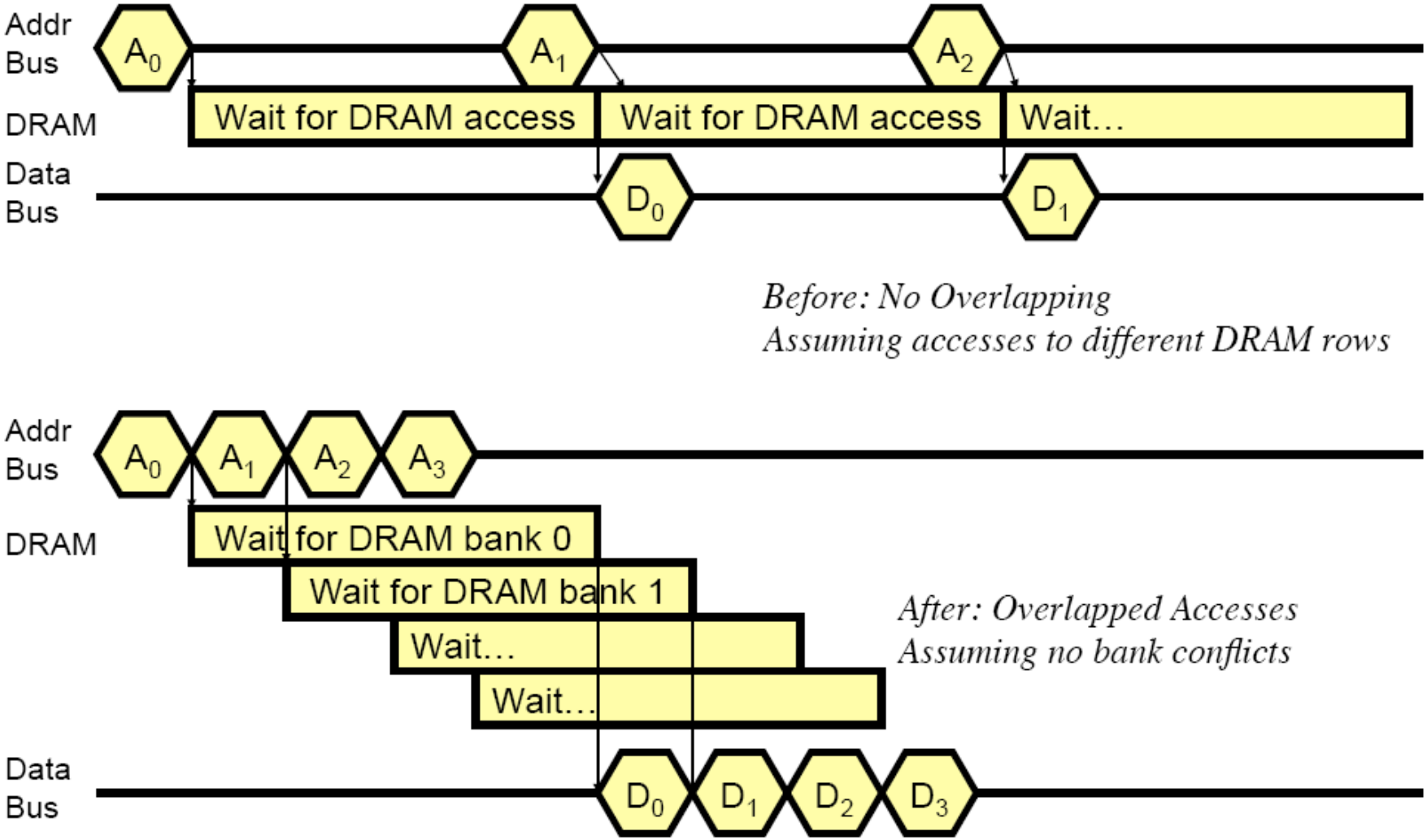
- CPU → controller transfer time
- Controller latency
  - Queuing & scheduling delay at the controller
  - Access converted to basic commands
- Controller → DRAM transfer time
- DRAM bank latency
  - Simple CAS (column address strobe) if row is "open" OR
  - RAS (row address strobe) + CAS if array precharged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

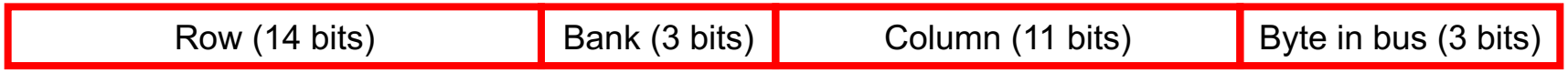
- Enabling more concurrency requires reducing
  - Bank conflicts
  - Channel conflicts
- How to select/randomize bank/channel indices in address?
  - Lower order bits have more entropy
  - Randomizing hash functions (XOR of different address bits)

How Multiple Banks Help

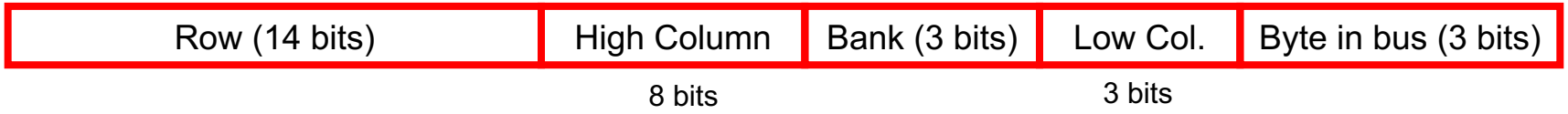


Address Mapping (Single Channel)

- 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



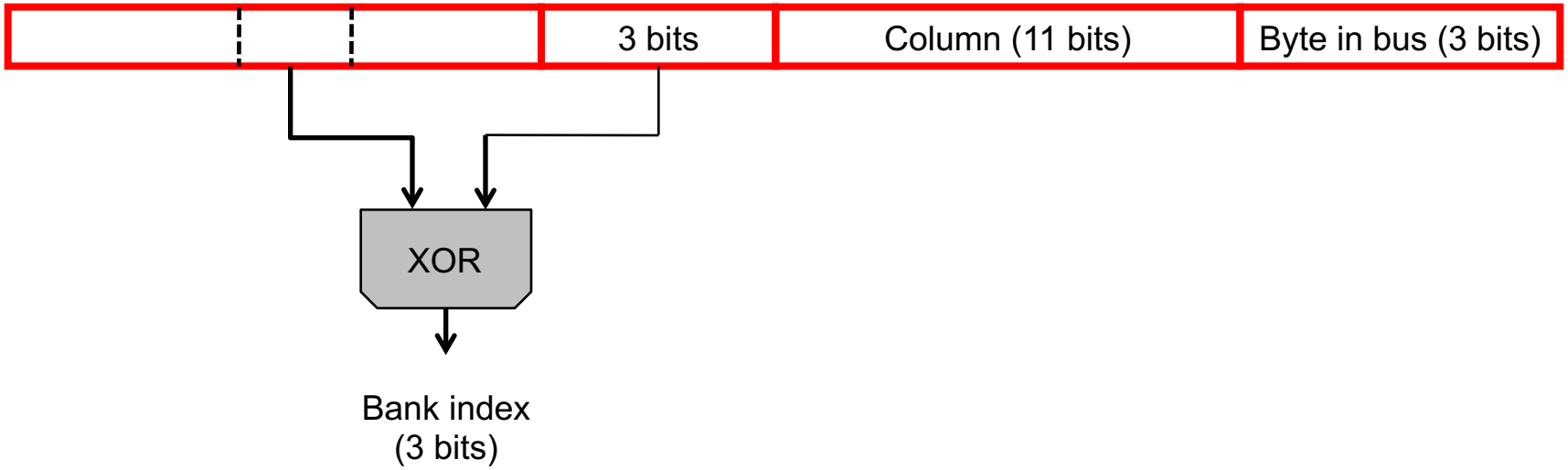
  - Accesses to consecutive cache blocks serviced in a pipelined manner
- Cache block interleaving
  - Consecutive cache block addresses in consecutive banks
  - 64 byte cache blocks



  - Accesses to consecutive cache blocks can be serviced in parallel

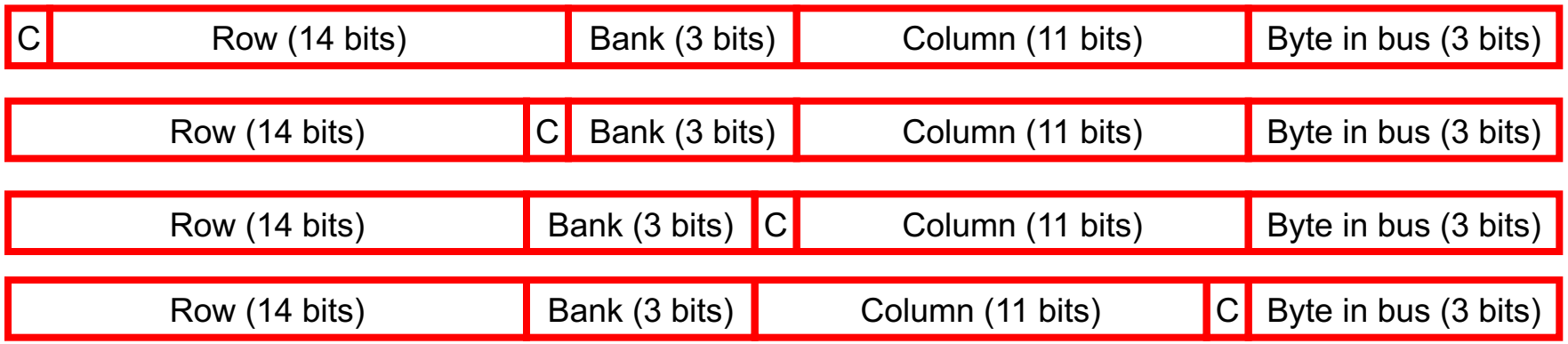
Bank Mapping Randomization

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

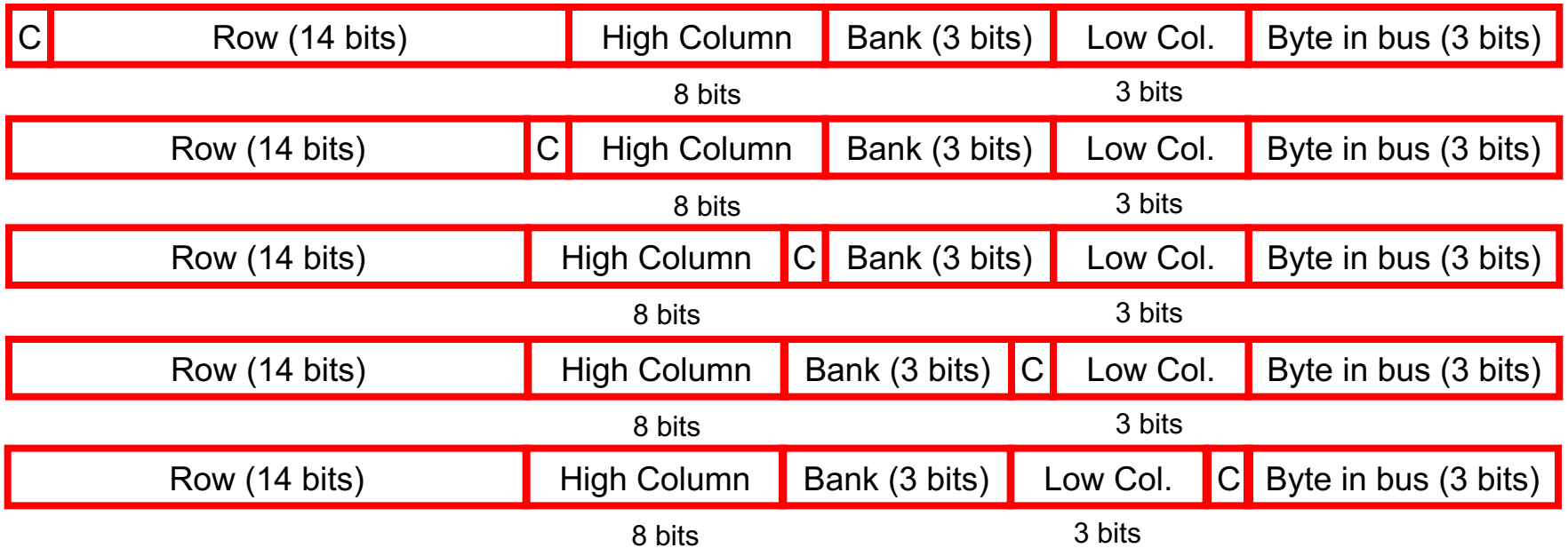


- Reading:
  - Rau, "Pseudo-randomly Interleaved Memory," ISCA 1991.

Address Mapping (Multiple Channels)

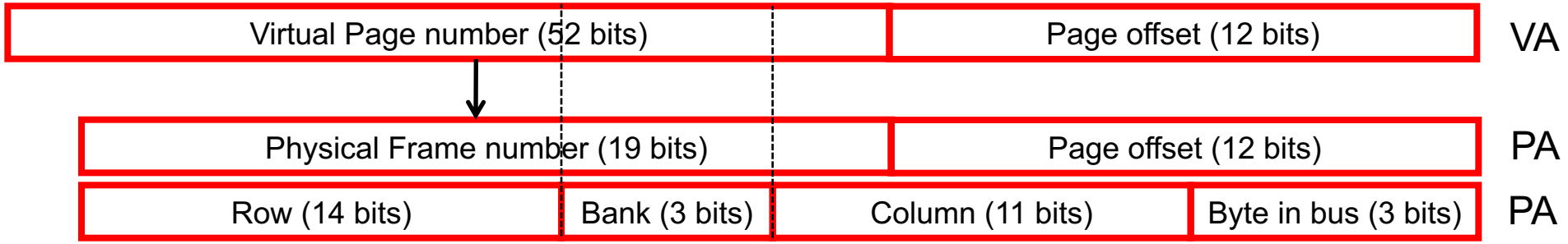


- Where are consecutive cache blocks?



Interaction with Virtual→Physical Mapping

- Operating System influences where an address maps to in DRAM



- 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]**

Memory Channel Partitioning

- Sai Prashanth Muralidhara, Lavanya Subramanian, Onur Mutlu, Mahmut Kandemir, and Thomas Moscibroda, **"Reducing Memory Interference in Multicore Systems via Application-Aware Memory Channel Partitioning"** *Proceedings of the 44th International Symposium on Microarchitecture (**MICRO**)*, Porto Alegre, Brazil, December 2011. Slides (pptx)

Reducing Memory Interference in Multicore Systems via Application-Aware Memory Channel Partitioning

Sai Prashanth Muralidhara
Pennsylvania State University
smuralid@cse.psu.edu

Lavanya Subramanian
Carnegie Mellon University
lsubrama@ece.cmu.edu

Onur Mutlu
Carnegie Mellon University
onur@cmu.edu

Mahmut Kandemir
Pennsylvania State University
kandemir@cse.psu.edu

Thomas Moscibroda
Microsoft Research Asia
moscitho@microsoft.com

Application-to-Core Mapping

- Reetuparna Das, Rachata Ausavarungnirun, Onur Mutlu, Akhilesh Kumar, and Mani Azimi,
  "Application-to-Core Mapping Policies to Reduce Memory System Interference in Multi-Core Systems"
  *Proceedings of the 19th International Symposium on High-Performance Computer Architecture (**HPCA**)*, Shenzhen, China, February 2013.
  Slides (pptx)

Application-to-Core Mapping Policies
to Reduce Memory System Interference in Multi-Core Systems

Reetuparna Das* Rachata Ausavarungnirun† Onur Mutlu† Akhilesh Kumar‡ Mani Azimi‡
University of Michigan* Carnegie Mellon University† Intel Labs‡

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.

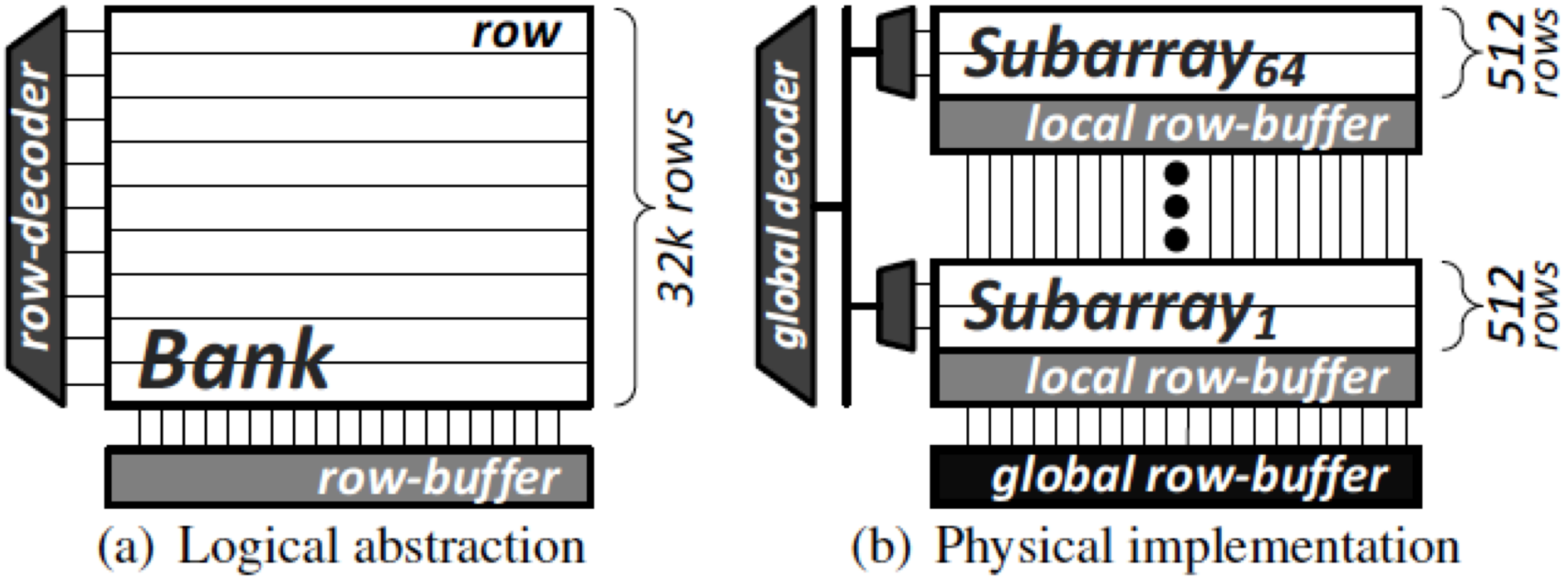


Figure 1. DRAM bank organization

Subarray Level Parallelism

- 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)

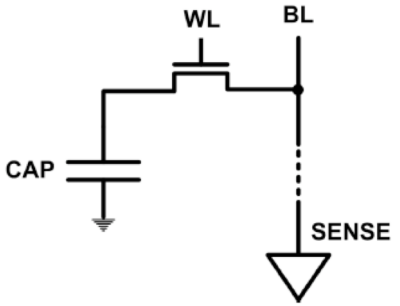
A Case for Exploiting Subarray-Level Parallelism (SALP) in DRAM

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



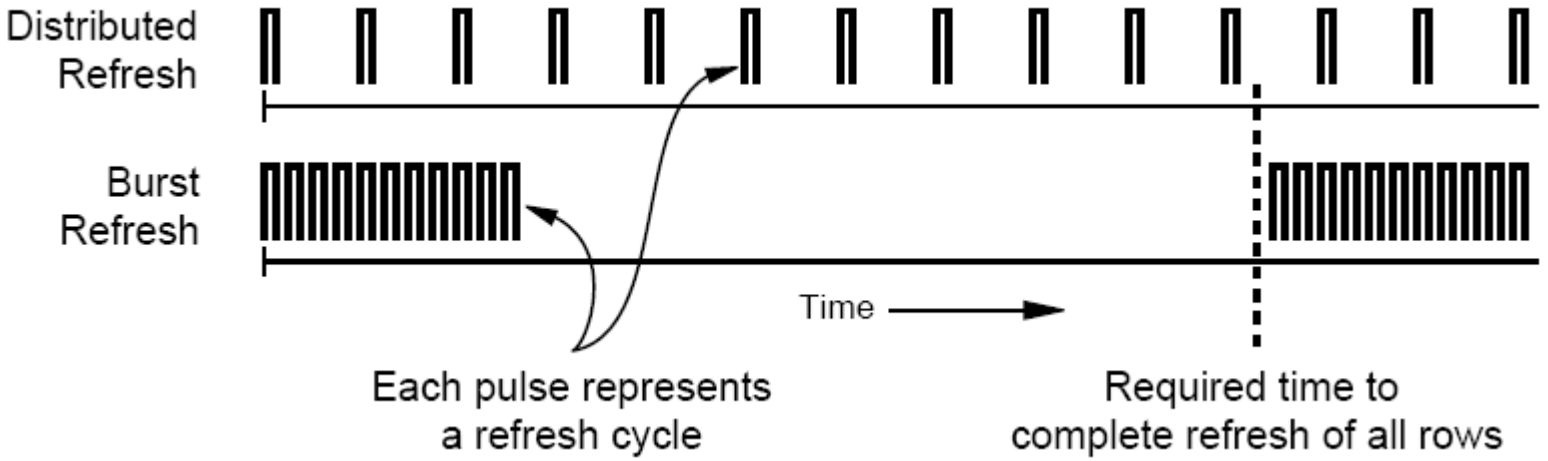
- Implications on performance?

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

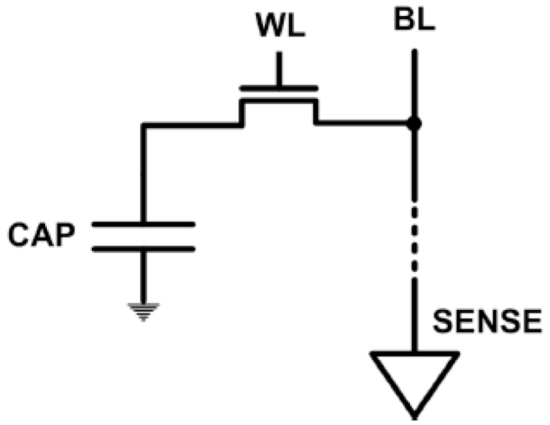
Downsides of DRAM Refresh

-- 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.

More on DRAM Refresh

- Jamie Liu, Ben Jaiyen, Richard Veras, and Onur Mutlu, **"RAIDR: Retention-Aware Intelligent DRAM Refresh"** *Proceedings of the 39th International Symposium on Computer Architecture (**ISCA**)*, Portland, OR, June 2012. Slides (pdf)

RAIDR: Retention-Aware Intelligent DRAM Refresh

Jamie Liu Ben Jaiyen Richard Veras Onur Mutlu
Carnegie Mellon University

DRAM Retention Analysis

- Jamie Liu, Ben Jaiyen, Yoongu Kim, Chris Wilkerson, and Onur Mutlu, **"An Experimental Study of Data Retention Behavior in Modern DRAM Devices: Implications for Retention Time Profiling Mechanisms"** *Proceedings of the 40th International Symposium on Computer Architecture (**ISCA**)*, Tel-Aviv, Israel, June 2013. Slides (ppt) Slides (pdf)

An Experimental Study of Data Retention Behavior in Modern DRAM Devices: Implications for Retention Time Profiling Mechanisms

Jamie Liu*
Carnegie Mellon University
5000 Forbes Ave.
Pittsburgh, PA 15213
jamiel@alumni.cmu.edu

Ben Jaiyen*
Carnegie Mellon University
5000 Forbes Ave.
Pittsburgh, PA 15213
bjaiyen@alumni.cmu.edu

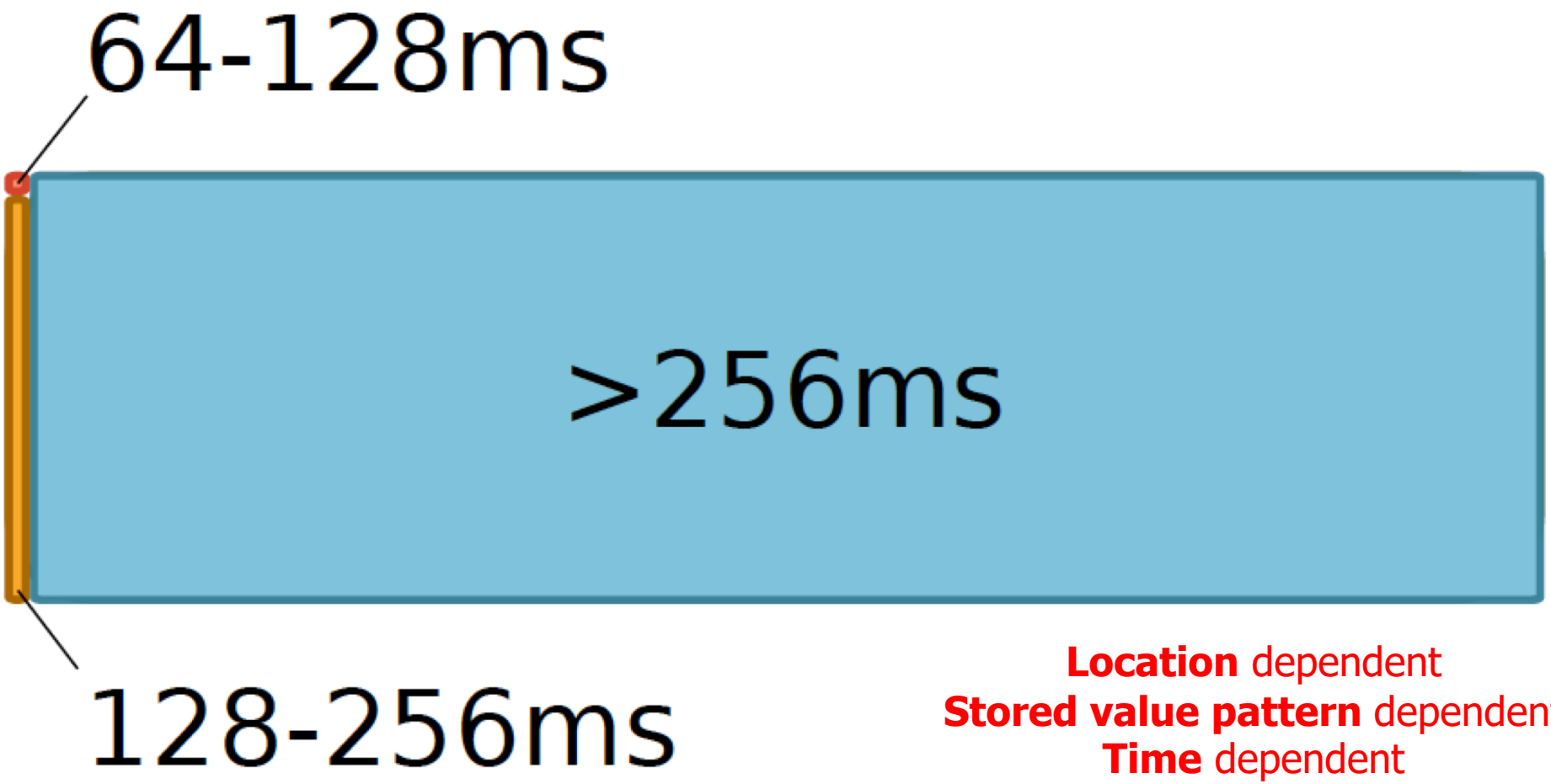
Yoongu Kim
Carnegie Mellon University
5000 Forbes Ave.
Pittsburgh, PA 15213
yoonguk@ece.cmu.edu

Chris Wilkerson
Intel Corporation
2200 Mission College Blvd.
Santa Clara, CA 95054
chris.wilkerson@intel.com

Onur Mutlu
Carnegie Mellon University
5000 Forbes Ave.
Pittsburgh, PA 15213
onur@cmu.edu

Data Retention in Memory [Liu et al., ISCA 2013]

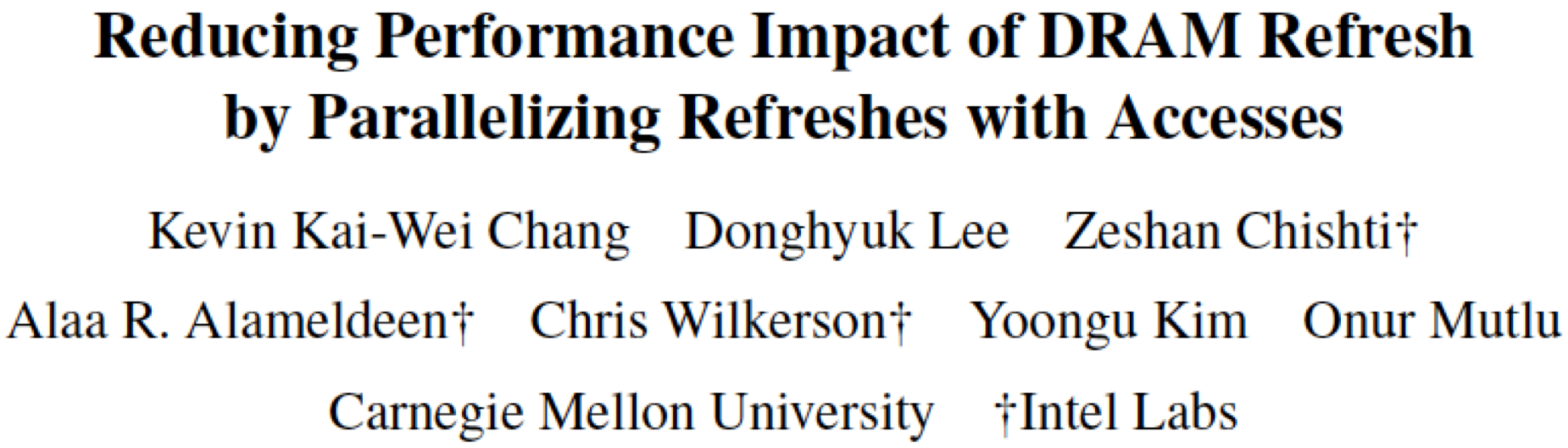
- Data Retention Time Profile of DRAM looks like this:



Location dependent
Stored value pattern dependent
Time dependent

DRAM Refresh-Access Parallelization

- Kevin Chang, Donghyuk Lee, Zeshan Chishti, Alaa Alameldeen, Chris Wilkerson, Yoongu Kim, and Onur Mutlu,
  "Improving DRAM Performance by Parallelizing Refreshes with Accesses"
  *Proceedings of the 20th International Symposium on High-Performance Computer Architecture (**HPCA**)*, Orlando, FL, February 2014.
  [Summary] [Slides (pptx) (pdf)]



Reducing Performance Impact of DRAM Refresh by Parallelizing Refreshes with Accesses

Kevin Kai-Wei Chang Donghyuk Lee Zeshan Chishti†

Alaa R. Alameldeen† Chris Wilkerson† Yoongu Kim Onur Mutlu

Carnegie Mellon University †Intel Labs

Memory Controllers

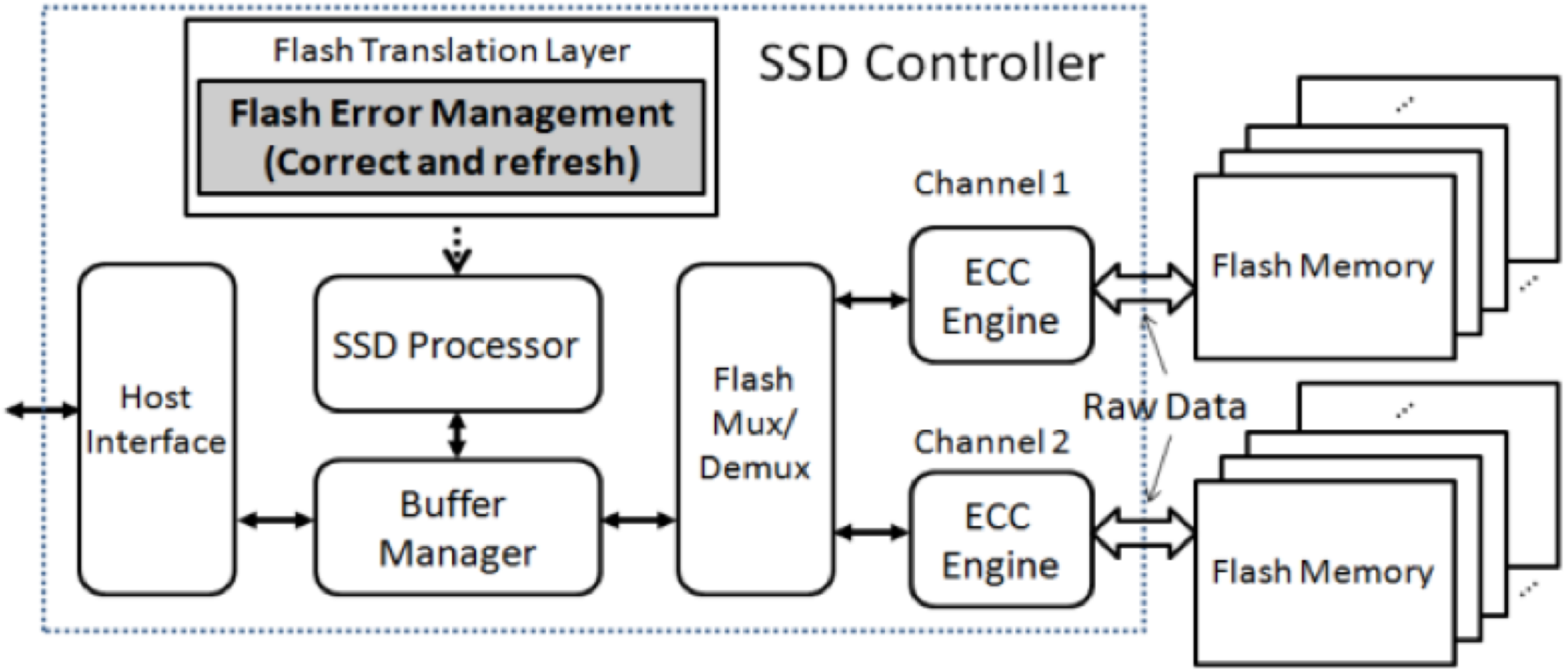
DRAM versus Other Types of Memories

- 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

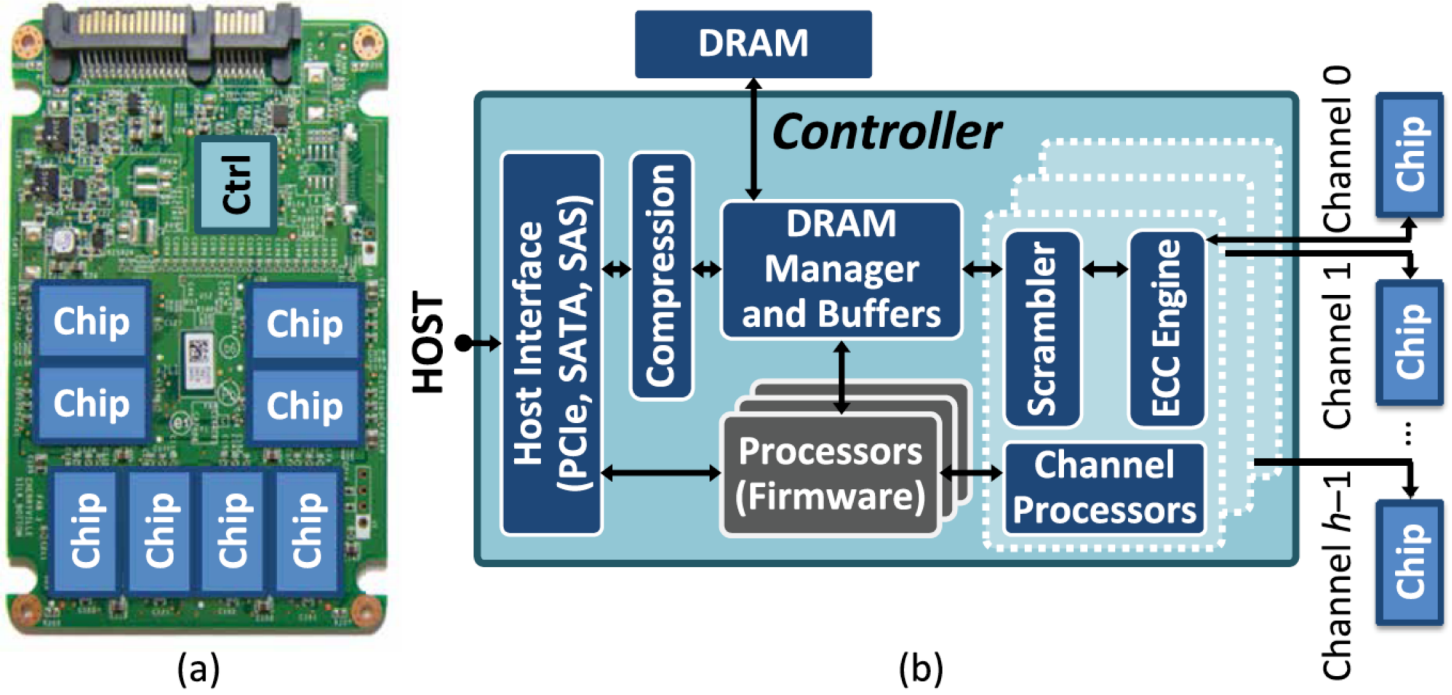


Fig. 1. ***(a) SSD system architecture, showing controller (Ctrl) and chips. (b) Detailed view of connections between controller components and chips.***

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

https://arxiv.org/pdf/1711.11427.pdf

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

https://arxiv.org/pdf/1706.08642

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, 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]

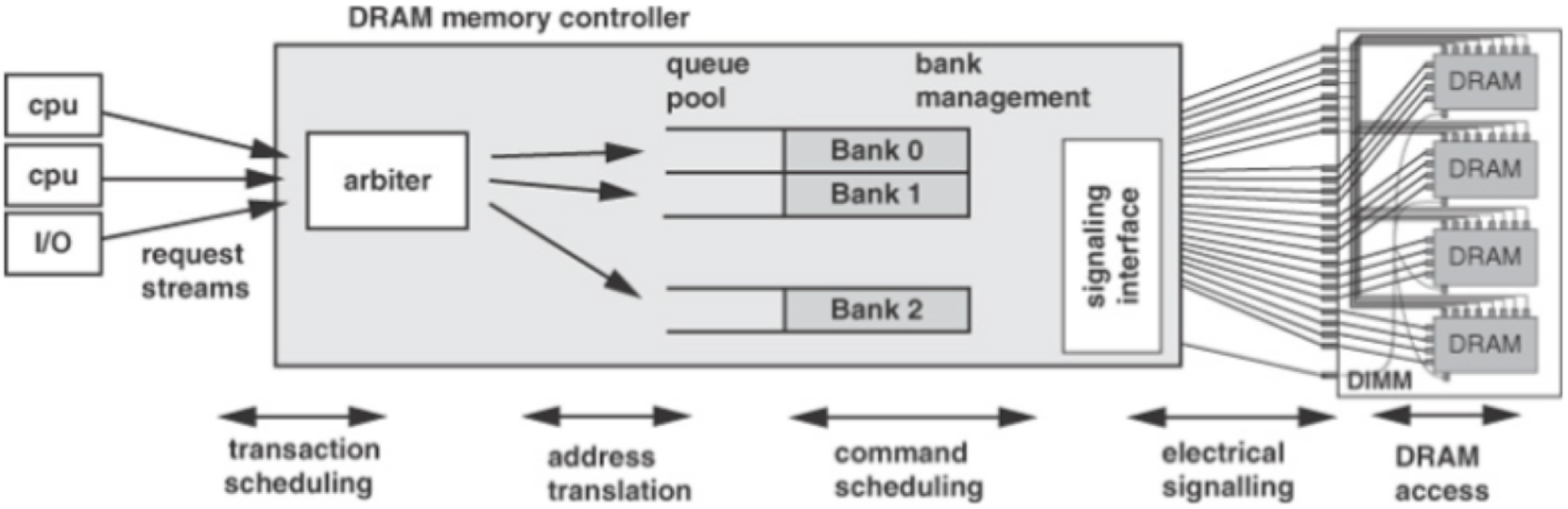
Table 1. Landscape of DRAM-based memory

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

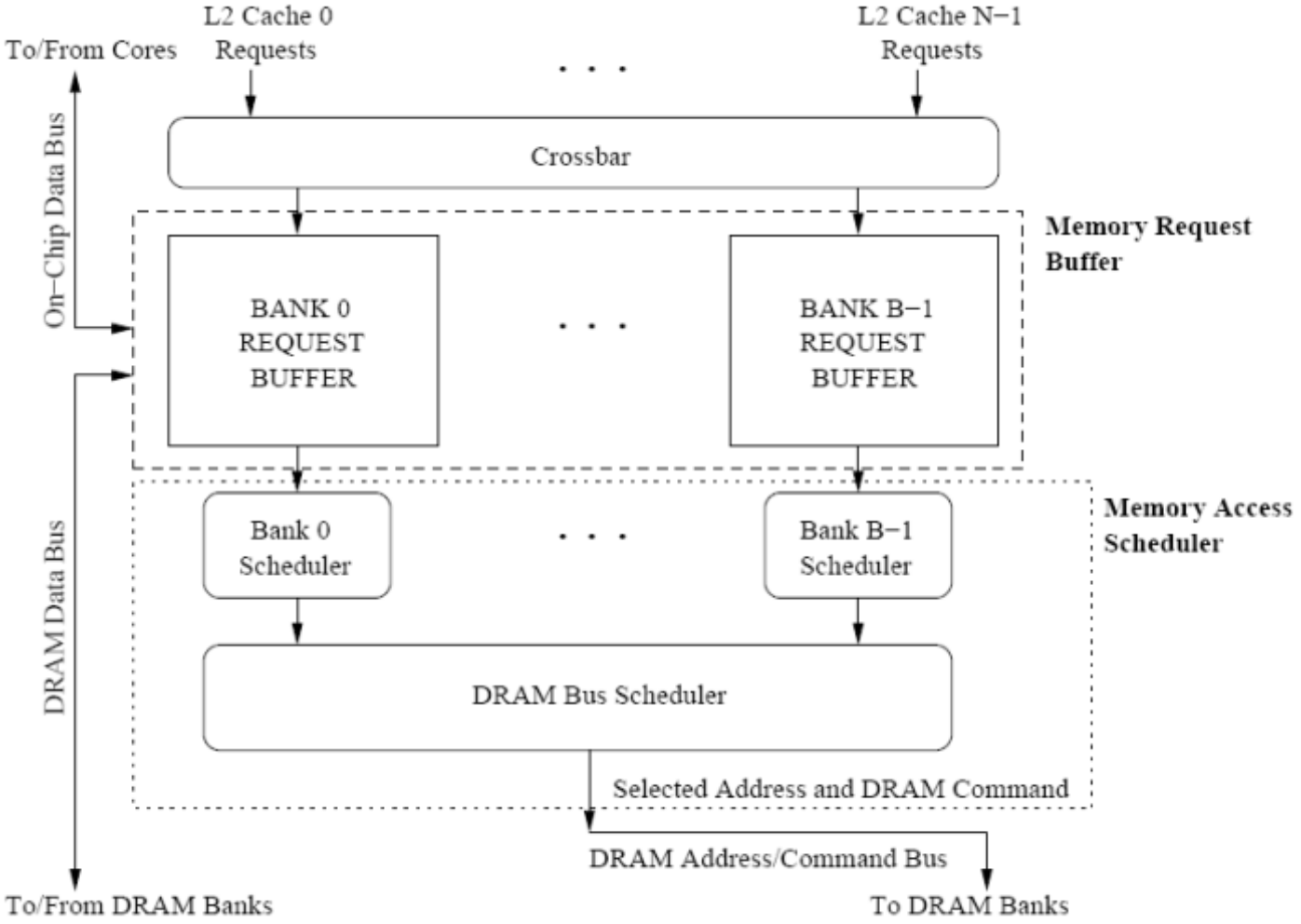
DRAM Controller: Functions

- 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

A Modern DRAM Controller (I)



A Modern DRAM Controller



Mutlu+, "Stall-Time Fair Memory Scheduling," MICRO 2007.

DRAM Scheduling Policies (I)

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

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

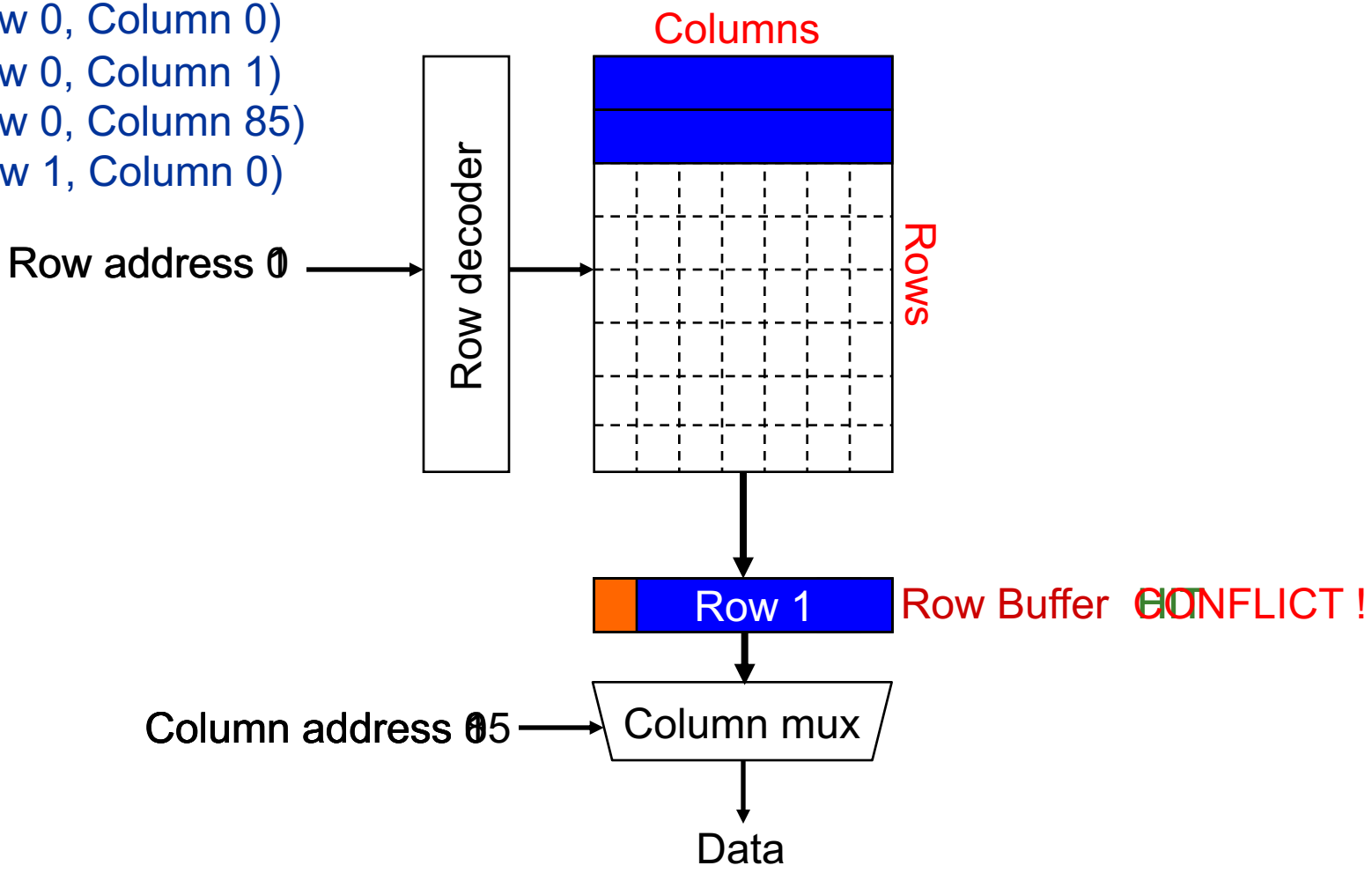
  1. Row-hit first
  2. Oldest first

  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	^tRP	11	Activate to read/write	^tRCD	11
Read column address strobe	CL	11	Write column address strobe	CWL	8
Additive	AL	0	Activate to activate	^tRC	39
Activate to precharge	^tRAS	28	Read to precharge	^tRTP	6
Burst length	^tBL	4	Column address strobe to column address strobe	^tCCD	4
Activate to activate (different bank)	^tRRD	6	Four activate windows	^tFAW	24
Write to read	^tWTR	6	Write recovery	^tWR	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.

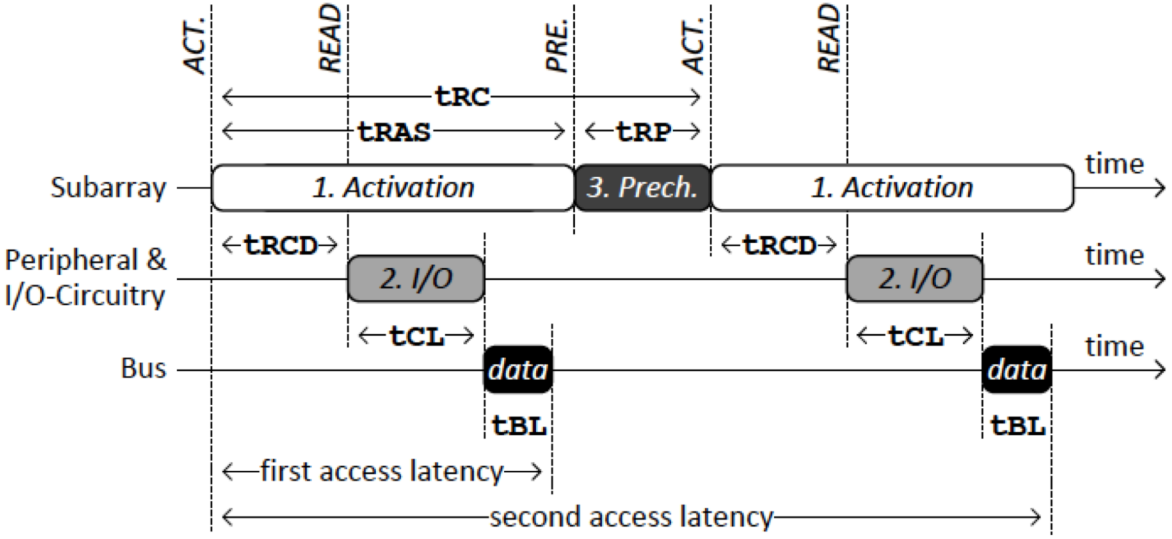


Figure 5. Three Phases of DRAM Access

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)

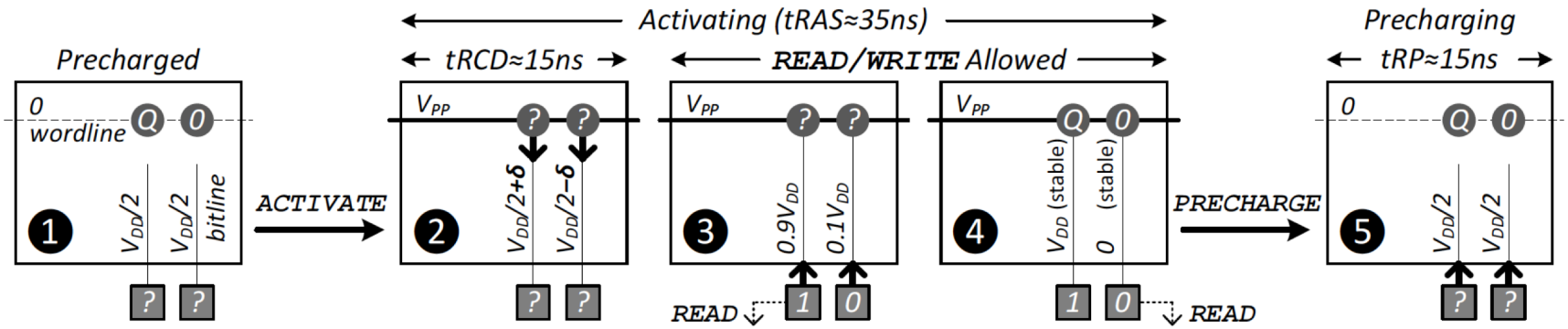


Figure 4. DRAM bank operation: Steps involved in serving a memory request [17] ($V_{PP} > V_{DD}$)

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)

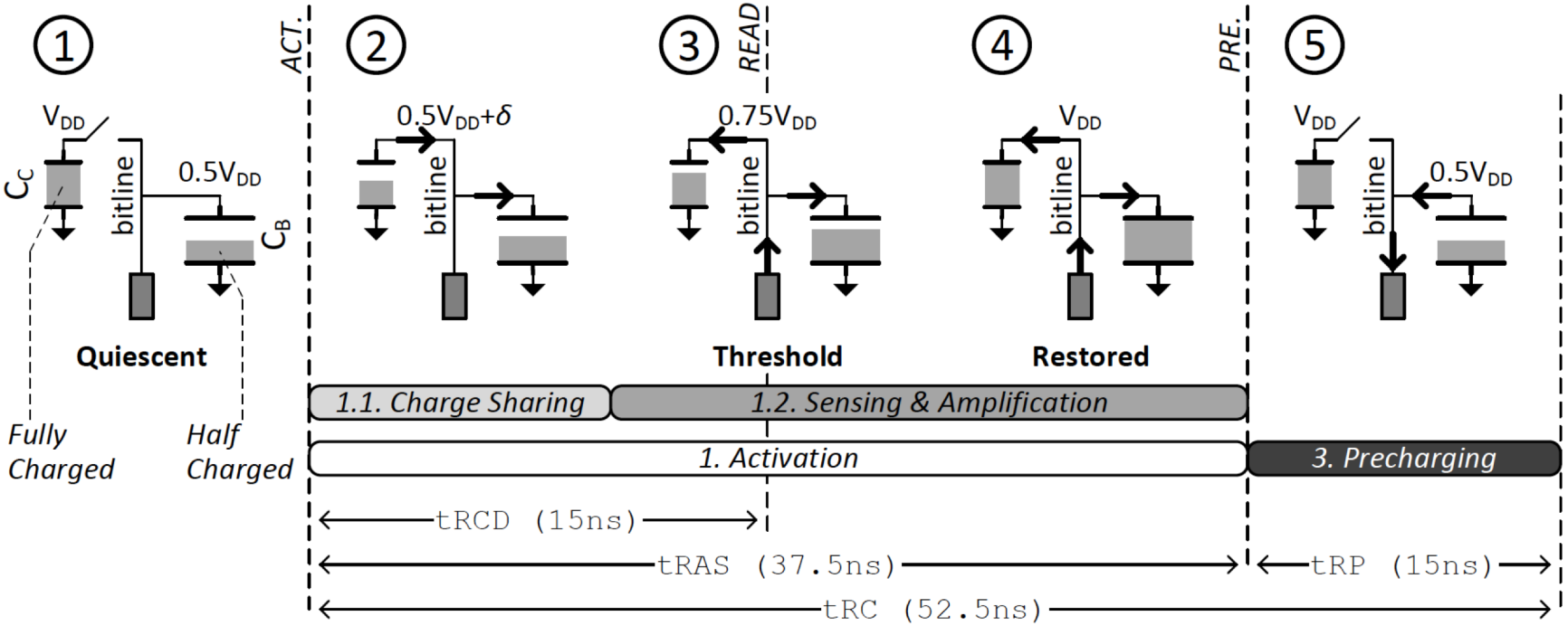


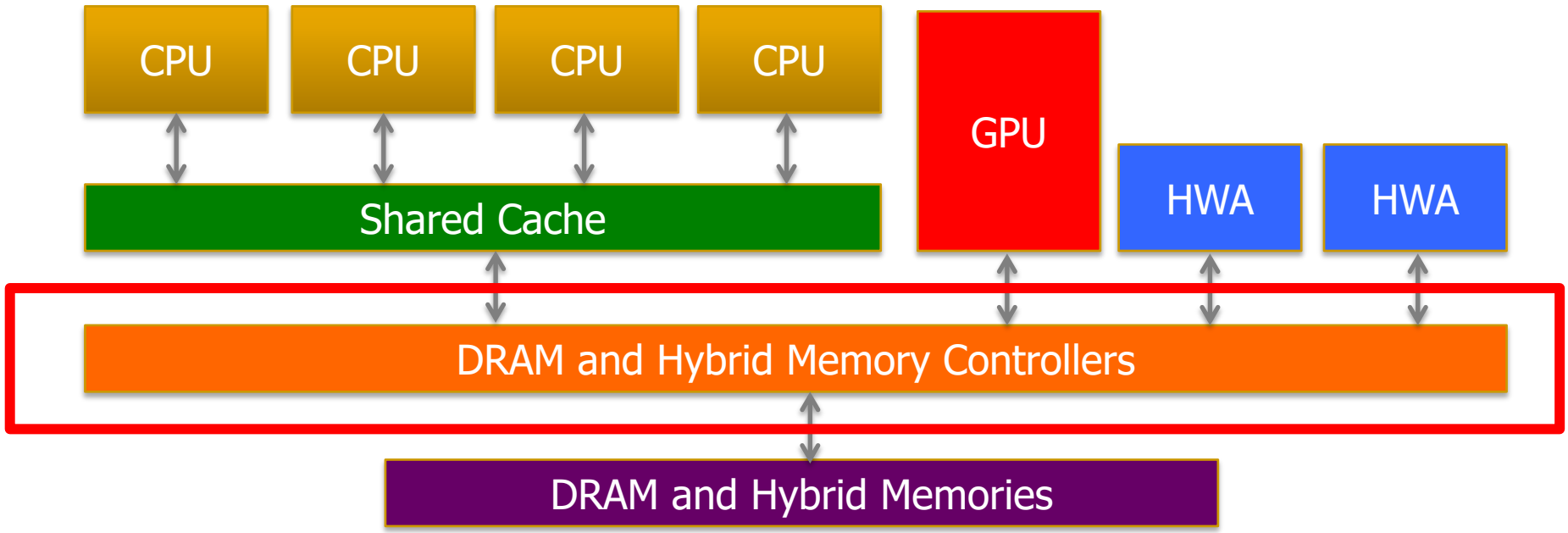
Figure 6. Charge Flow Between the Cell Capacitor (C_C), Bitline Parasitic Capacitor (C_B), and the Sense-Amplifier ($C_B \approx 3.5C_C$ [39])

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

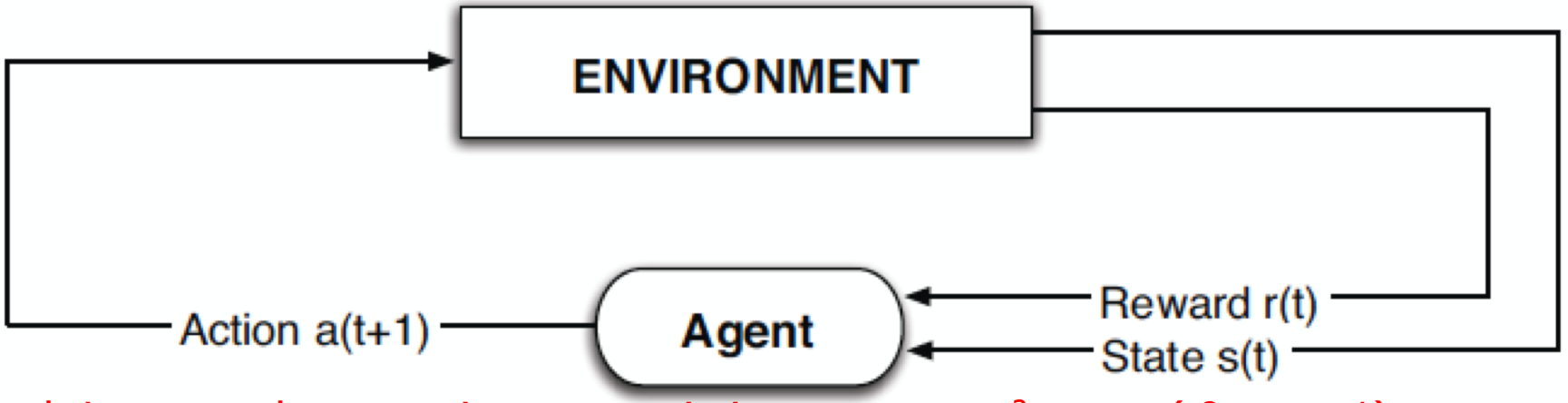
- Idea: Design a memory controller that adapts its scheduling policy decisions to workload behavior and system conditions using machine learning.

- 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

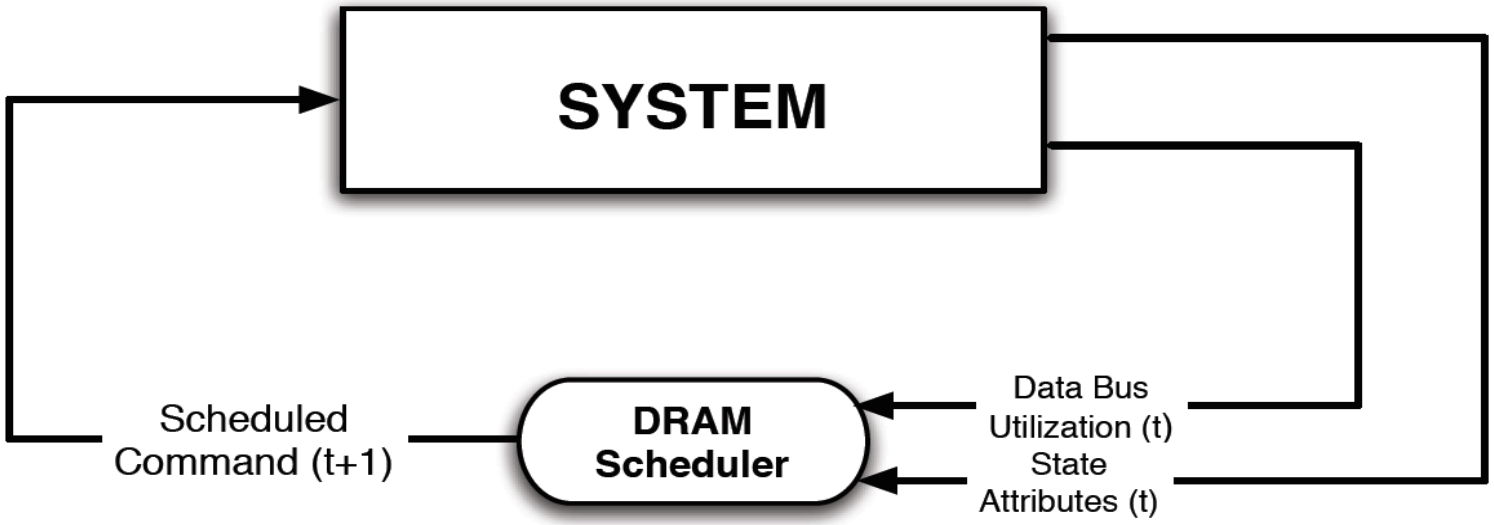


Goal: Learn to choose actions to maximize $r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots$ ($0 \leq \gamma < 1$)

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.

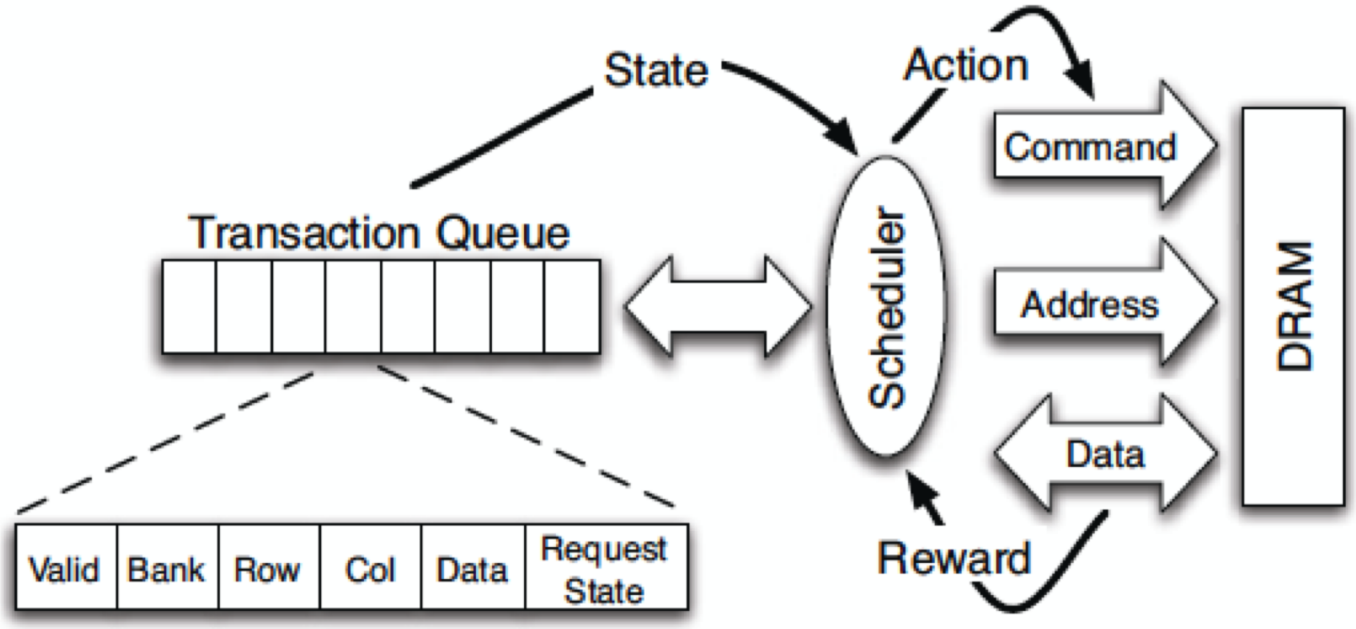


Figure 4: High-level overview of an RL-based scheduler.

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

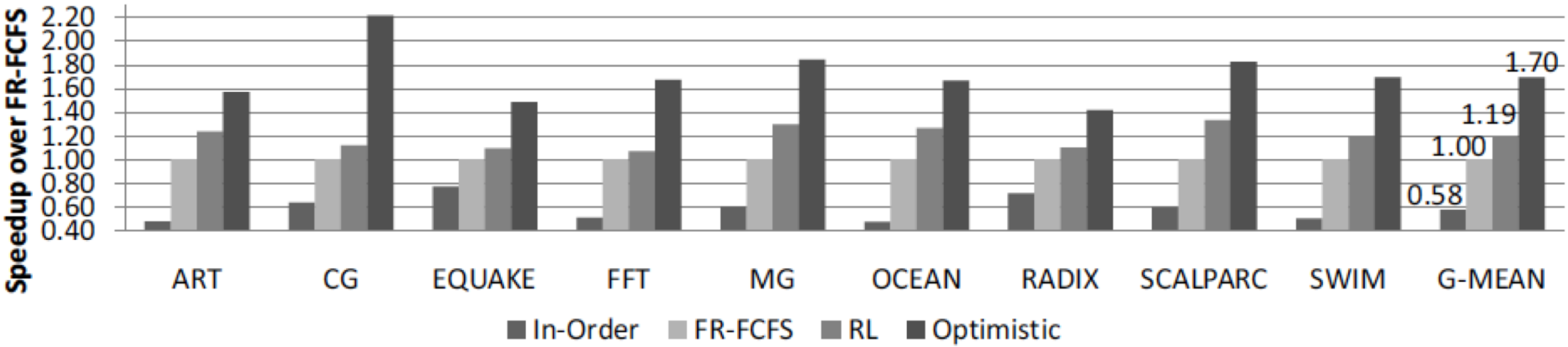


Figure 7: Performance comparison of in-order, FR-FCFS, RL-based, and optimistic memory controllers

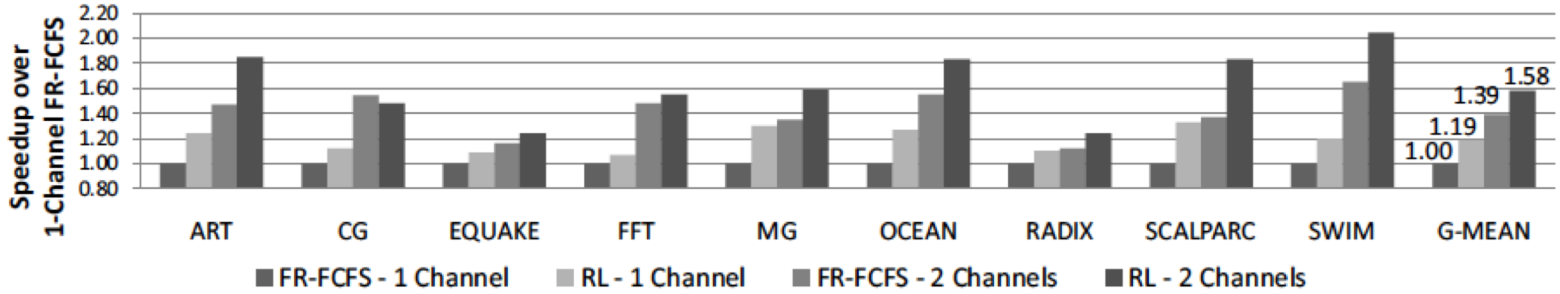


Figure 15: Performance comparison of FR-FCFS and RL-based memory controllers on systems with 6.4GB/s and 12.8GB/s peak DRAM bandwidth

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

Computer Architecture

Lecture 5: Main Memory and DRAM Fundamentals

Prof. Onur Mutlu
ETH Zürich

Fall 2018

3 October 2018