

18-740/640

Computer Architecture

Lecture 8: Main Memory System

Prof. Onur Mutlu

Carnegie Mellon University

Fall 2015, 9/28/2015

Required Readings

➤ Required Reading Assignment:

- **Sec. 1 & 3 of B. Jacob, "The Memory System: You Can't Avoid It, You Can't Ignore It, You Can't Fake It," Synthesis Lectures on Computer Architecture, 2009.**

➤ Recommended References:

- O. Mutlu and L. Subramanian, "Research Problems and Opportunities in Memory Systems," Supercomputing Frontiers and Innovations, 2015.
- Lee et al., "Phase Change Technology and the Future of Main Memory," IEEE Micro, Jan/Feb 2010.
- Y. Kim, W. Yang, O. Mutlu, "Ramulator: A Fast and Extensible DRAM Simulator," IEEE Computer Architecture Letters, May 2015.

State-of-the-art in Main Memory...

- Onur Mutlu and Lavanya Subramanian,
"Research Problems and Opportunities in Memory Systems"
Invited Article in Supercomputing Frontiers and Innovations
(*SUPERFRI*), 2015.

Recommended 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.

http://users.ece.cmu.edu/~omutlu/pub/tldram_hpca13.pdf

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

http://users.ece.cmu.edu/~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.

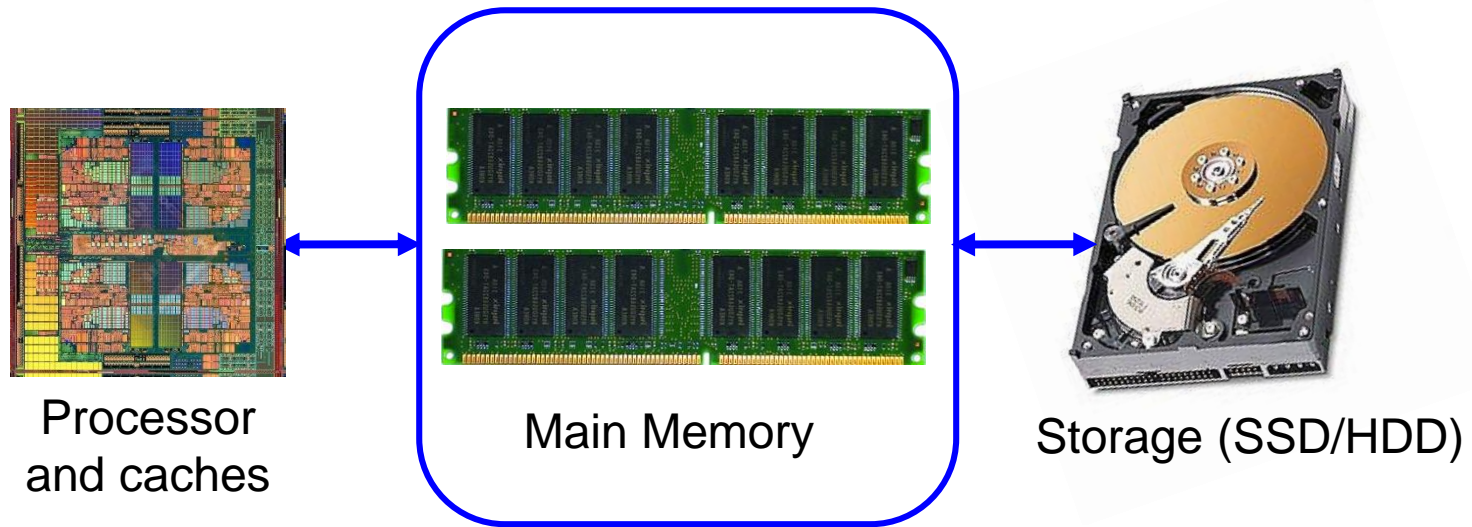
http://users.ece.cmu.edu/~omutlu/pub/raidr-dram-refresh_isca12.pdf

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]

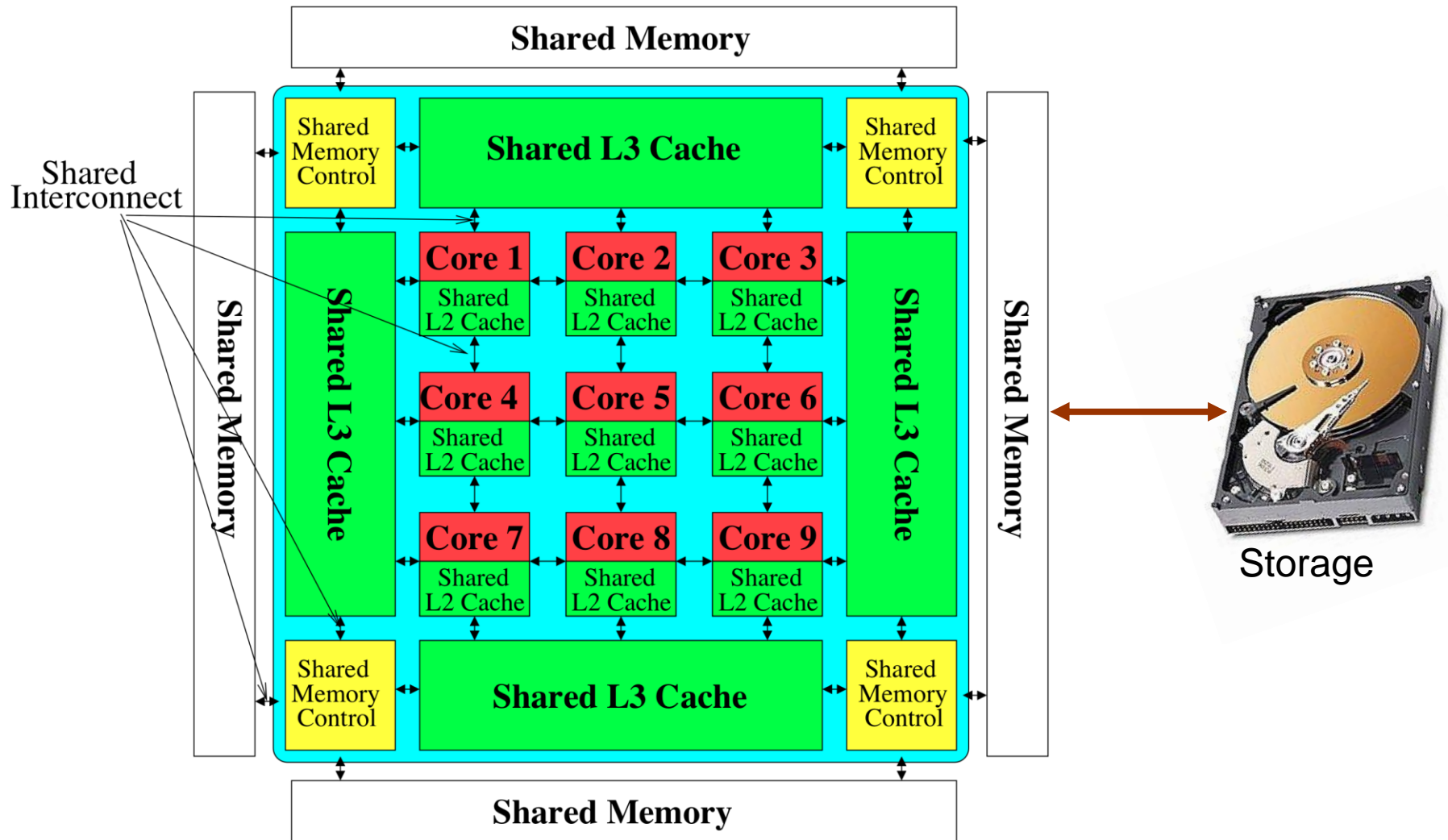
Why Is Main Memory So Important Especially Today?

The Main Memory System



- Main memory is a critical component of all computing systems: server, mobile, embedded, desktop, sensor
- Main memory system must scale (in *size, technology, efficiency, cost, and management algorithms*) to maintain performance growth and technology scaling benefits

Memory System: A *Shared Resource* View



State of the Main Memory System

- Recent technology, architecture, and application trends
 - lead to new requirements
 - exacerbate old requirements
- DRAM and memory controllers, as we know them today, are (will be) unlikely to satisfy all requirements
- Some emerging non-volatile memory technologies (e.g., PCM) enable new opportunities: memory+storage merging
- We need to rethink the main memory system
 - to fix DRAM issues and enable emerging technologies
 - to satisfy all requirements

Major Trends Affecting Main Memory (I)

- Need for main memory capacity, bandwidth, QoS increasing
- Main memory energy/power is a key system design concern
- DRAM technology scaling is ending

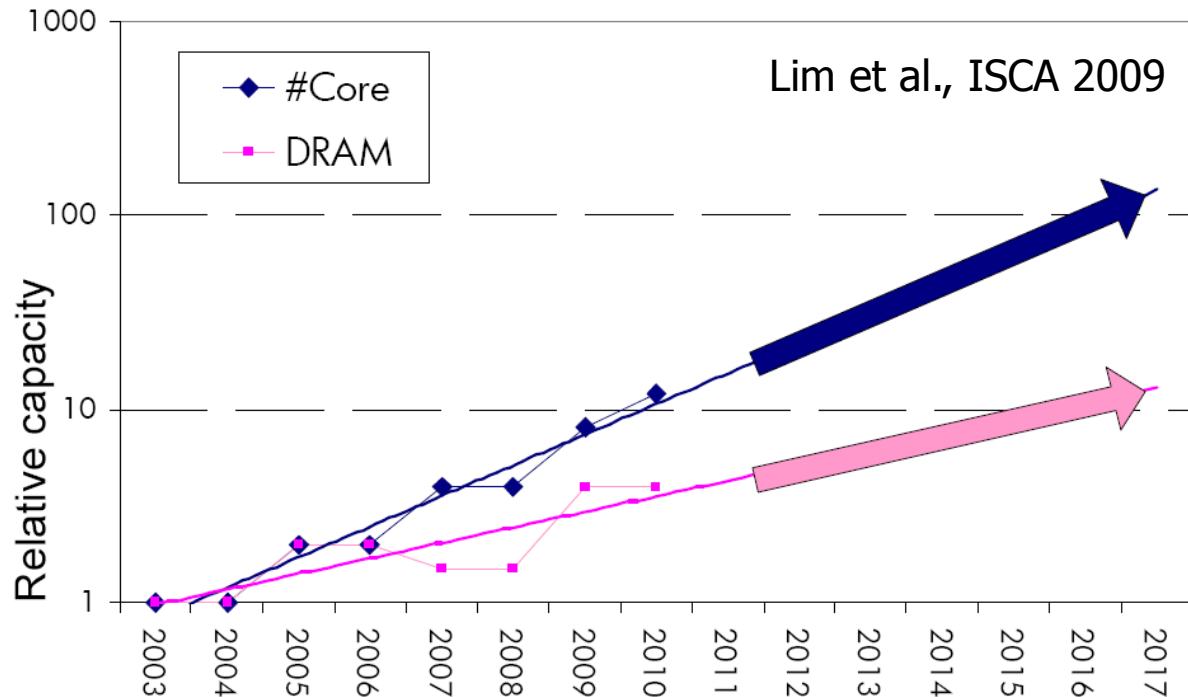
Major Trends Affecting Main Memory (II)

- Need for main memory capacity, bandwidth, QoS increasing
 - **Multi-core**: increasing number of cores/agents
 - **Data-intensive applications**: increasing demand/hunger for data
 - **Consolidation**: cloud computing, GPUs, mobile, heterogeneity
- Main memory energy/power is a key system design concern
- DRAM technology scaling is ending

Example: The Memory Capacity Gap

Core count doubling ~ every 2 years

DRAM DIMM capacity doubling ~ every 3 years



- *Memory capacity per core* expected to drop by 30% every two years
- Trends worse for *memory bandwidth per core*!

Major Trends Affecting Main Memory (III)

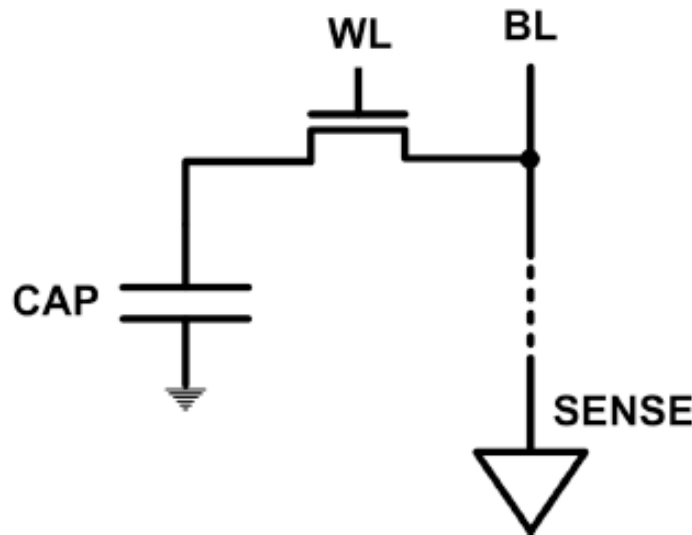
- Need for main memory capacity, bandwidth, QoS increasing
- Main memory energy/power is a key system design concern
 - ~40-50% energy spent in off-chip memory hierarchy [Lefurgy, IEEE Computer 2003]
 - DRAM consumes power even when not used (periodic refresh)
- DRAM technology scaling is ending

Major Trends Affecting Main Memory (IV)

- Need for main memory capacity, bandwidth, QoS increasing
- Main memory energy/power is a key system design concern
- DRAM technology scaling is ending
 - ITRS projects DRAM will not scale easily below X nm
 - Scaling has provided many benefits:
 - higher capacity (density), lower cost, lower energy

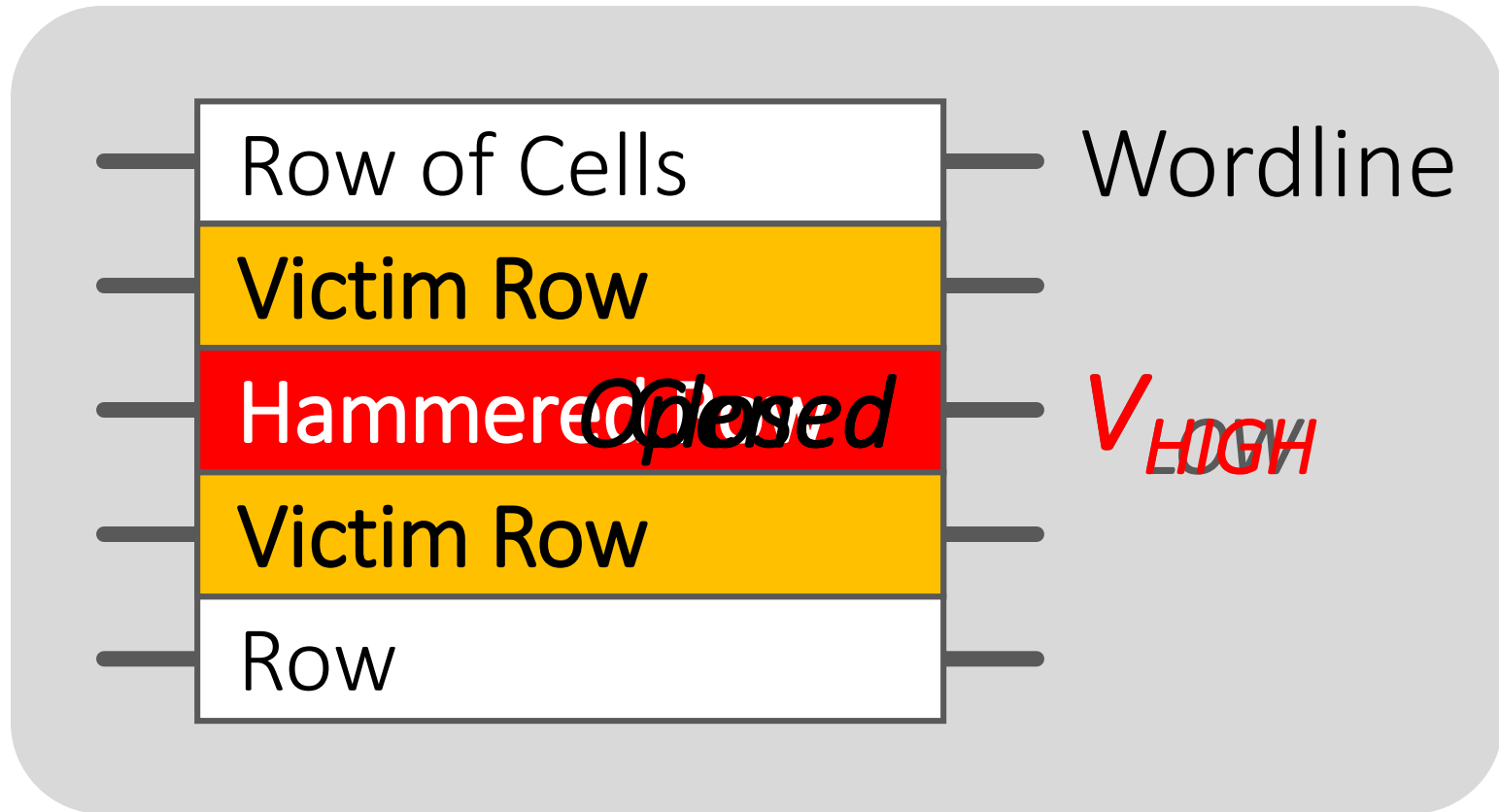
The DRAM Scaling Problem

- DRAM stores charge in a capacitor (charge-based memory)
 - Capacitor must be large enough for reliable sensing
 - Access transistor should be large enough for low leakage and high retention time
 - Scaling beyond 40-35nm (2013) is challenging [ITRS, 2009]



- DRAM capacity, cost, and energy/power hard to scale

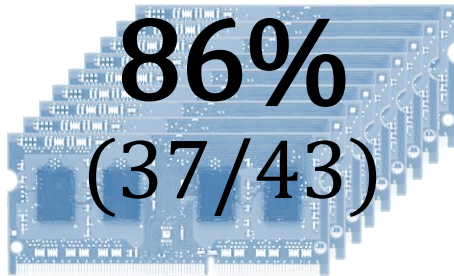
An Example of the DRAM Scaling Problem



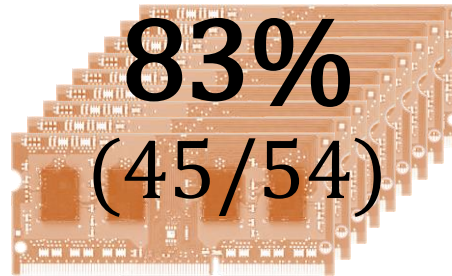
Repeatedly opening and closing a row enough times within a refresh interval induces **disturbance errors** in adjacent rows in **most real DRAM chips you can buy today**

Most DRAM Modules Are at Risk

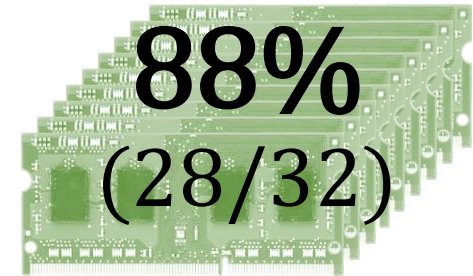
A company



B company



C company

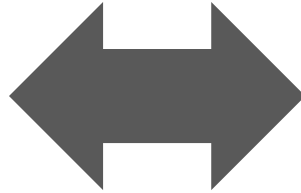
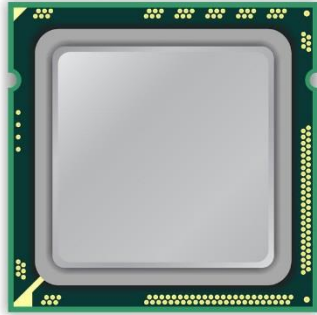


Up to
 1.0×10^7
errors

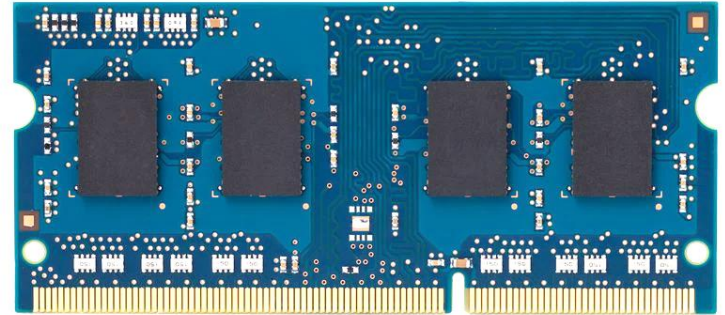
Up to
 2.7×10^6
errors

Up to
 3.3×10^5
errors

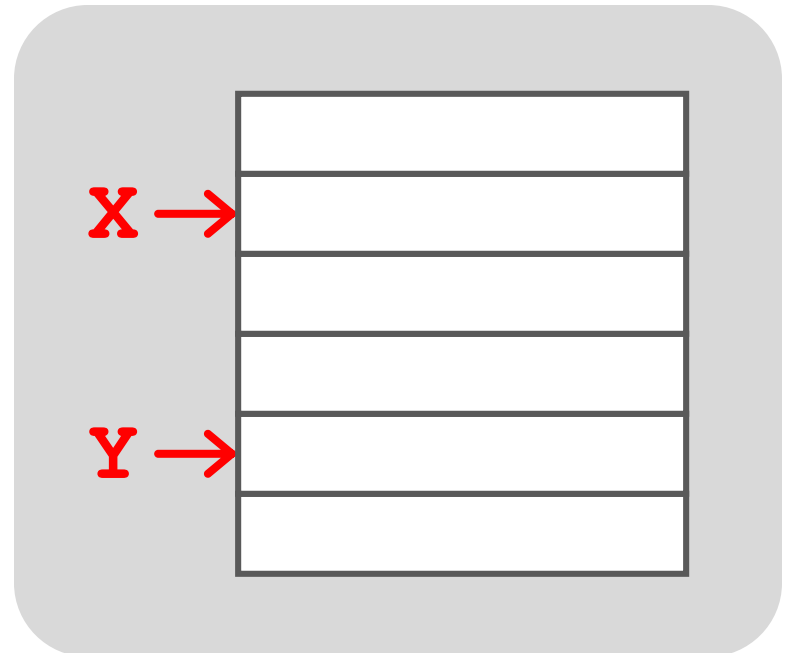
x86 CPU



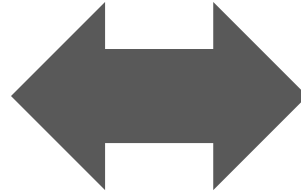
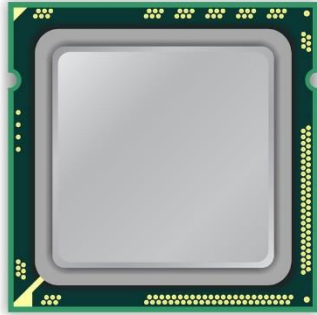
DRAM Module



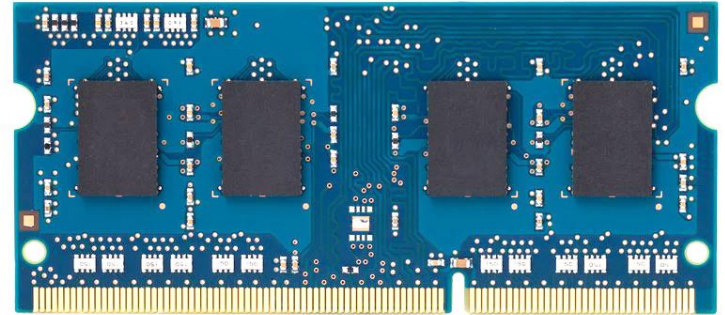
```
loop:  
  mov  (X), %eax  
  mov  (Y), %ebx  
  clflush (X)  
  clflush (Y)  
  mfence  
  jmp  loop
```



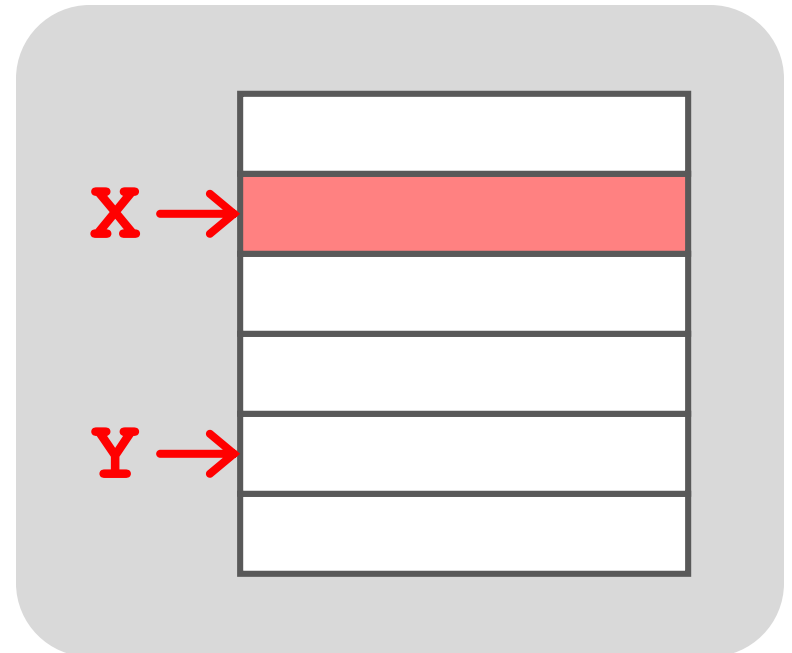
x86 CPU



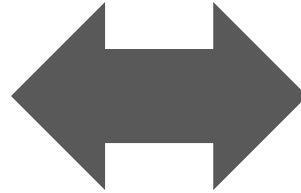
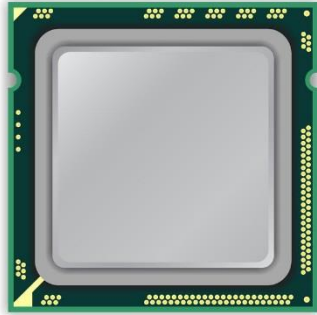
DRAM Module



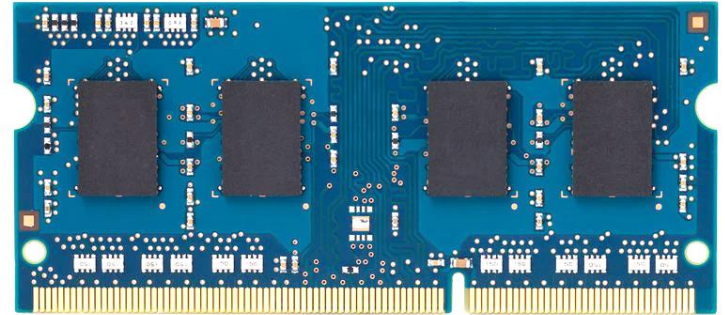
```
loop:  
  mov  (X),  %eax  
  mov  (Y),  %ebx  
  clflush (X)  
  clflush (Y)  
  mfence  
  jmp  loop
```



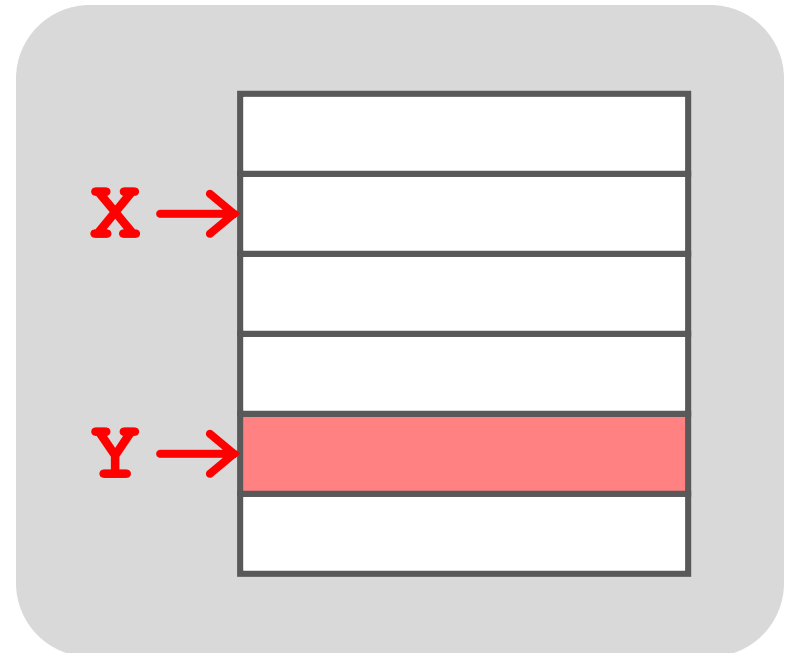
x86 CPU



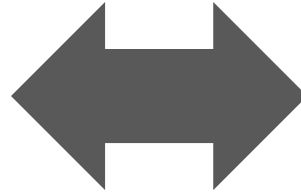
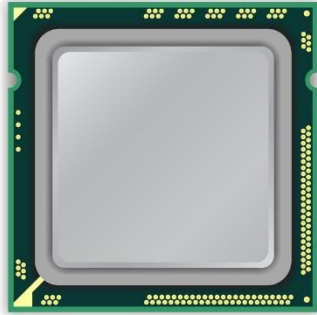
DRAM Module



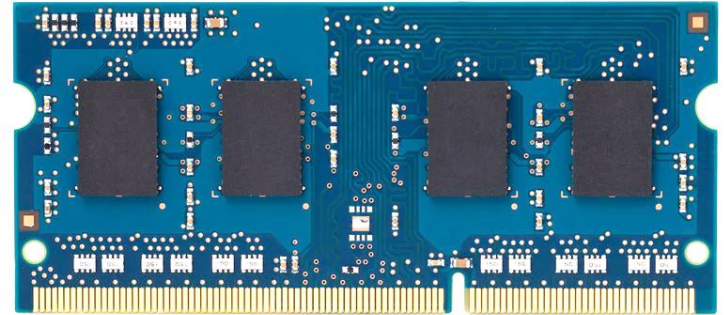
```
loop:  
  mov  (X), %eax  
  mov  (Y), %ebx  
  clflush (X)  
  clflush (Y)  
  mfence  
  jmp  loop
```



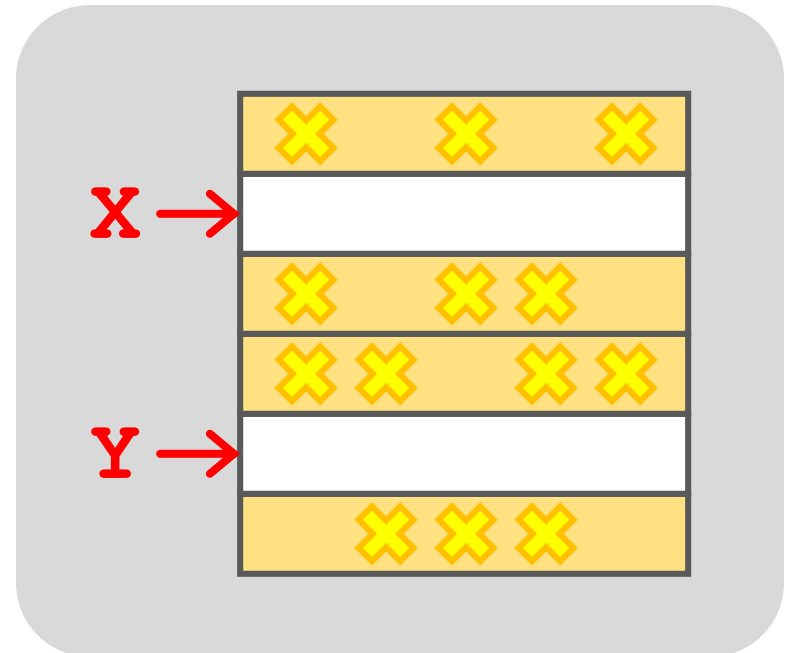
x86 CPU



DRAM Module



```
loop:  
  mov  (X),  %eax  
  mov  (Y),  %ebx  
  clflush (X)  
  clflush (Y)  
  mfence  
  jmp  loop
```

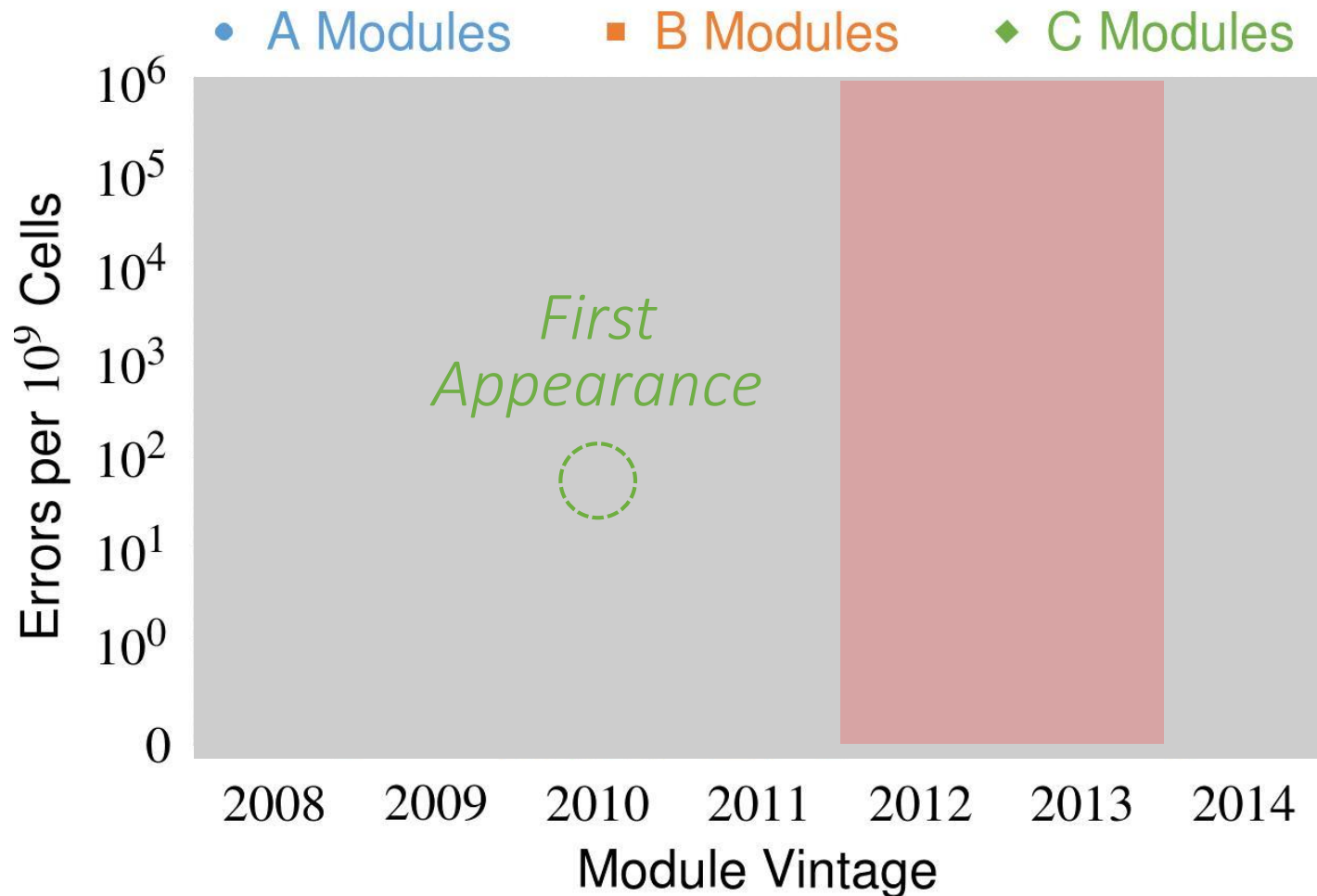


Observed Errors in Real Systems

CPU Architecture	Errors	Access-Rate
Intel Haswell (2013)	22.9K	12.3M/sec
Intel Ivy Bridge (2012)	20.7K	11.7M/sec
Intel Sandy Bridge (2011)	16.1K	11.6M/sec
AMD Piledriver (2012)	59	6.1M/sec

- *A real reliability & security issue*
- *In a more controlled environment, we can induce as many as **ten million** disturbance errors*

Errors *vs.* Vintage



All modules from 2012–2013 are vulnerable

Experimental DRAM Testing Infrastructure



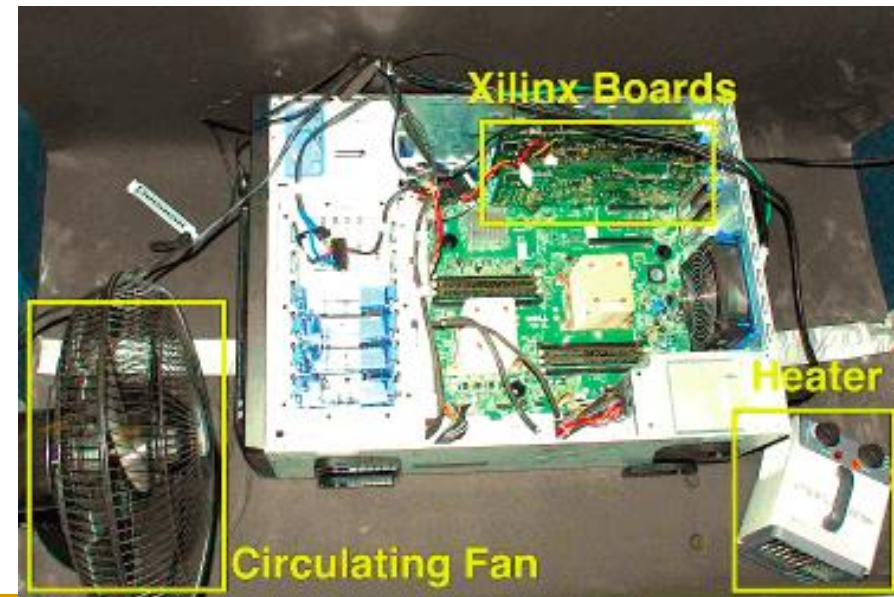
An Experimental Study of Data Retention Behavior in Modern DRAM Devices: Implications for Retention Time Profiling Mechanisms (Liu et al., ISCA 2013)

The Efficacy of Error Mitigation Techniques for DRAM Retention Failures: A Comparative Experimental Study (Khan et al., SIGMETRICS 2014)

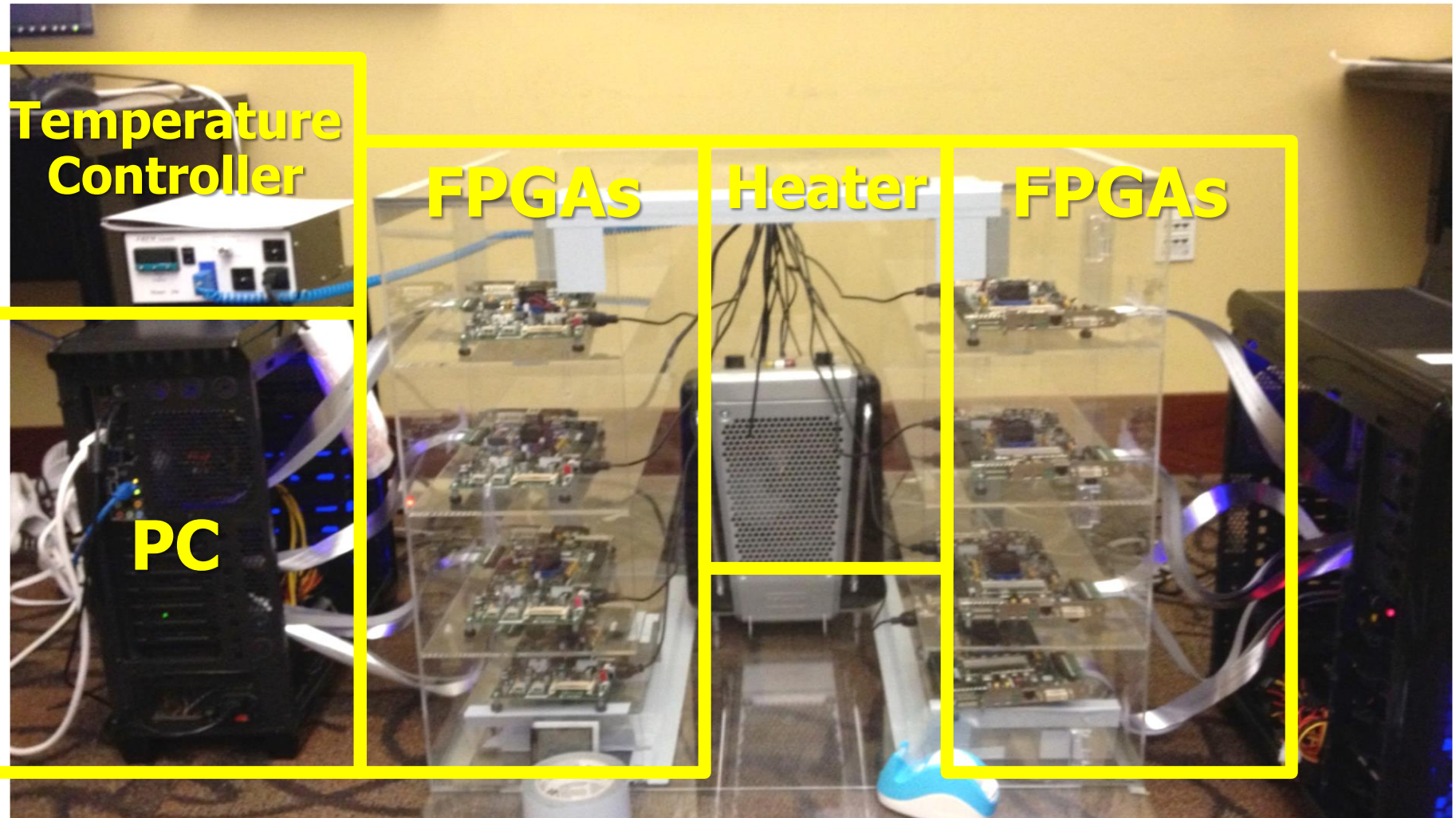
Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors (Kim et al., ISCA 2014)

Adaptive-Latency DRAM: Optimizing DRAM Timing for the Common-Case (Lee et al., HPCA 2015)

AVATAR: A Variable-Retention-Time (VRT) Aware Refresh for DRAM Systems (Qureshi et al., DSN 2015)



Experimental Infrastructure (DRAM)



RowHammer Characterization Results

1. Most Modules Are at Risk
2. Errors vs. Vintage
3. Error = Charge Loss
4. Adjacency: Aggressor & Victim
5. Sensitivity Studies
6. Other Results in Paper
7. Solution Space

One Can Take Over an Otherwise-Secure System

Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors

Abstract. Memory isolation is a key property of a reliable and secure computing system — an access to one memory address should not have unintended side effects on data stored in other addresses. However, as DRAM process technology

Project Zero

Flipping Bits in Memory Without Accessing Them:
An Experimental Study of DRAM Disturbance Errors
(Kim et al., ISCA 2014)

News and updates from the Project Zero team at Google

Exploiting the DRAM rowhammer bug to
gain kernel privileges (Seaborn, 2015)

Monday, March 9, 2015

Exploiting the DRAM rowhammer bug to gain kernel privileges

RowHammer Security Attack Example

- “Rowhammer” is a problem with some recent DRAM devices in which repeatedly accessing a row of memory can cause bit flips in adjacent rows (Kim et al., ISCA 2014).
 - Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors (Kim et al., ISCA 2014)
- We tested a selection of laptops and found that a subset of them exhibited the problem.
- We built two working privilege escalation exploits that use this effect.
 - Exploiting the DRAM rowhammer bug to gain kernel privileges (Seaborn, 2015)
- One exploit uses rowhammer-induced bit flips to gain kernel privileges on x86-64 Linux when run as an unprivileged userland process.
- When run on a machine vulnerable to the rowhammer problem, the process was able to induce bit flips in page table entries (PTEs).
- It was able to use this to gain write access to its own page table, and hence gain read-write access to all of physical memory.

Security Implications



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

Recap: The DRAM Scaling Problem

DRAM Process Scaling Challenges

❖ Refresh

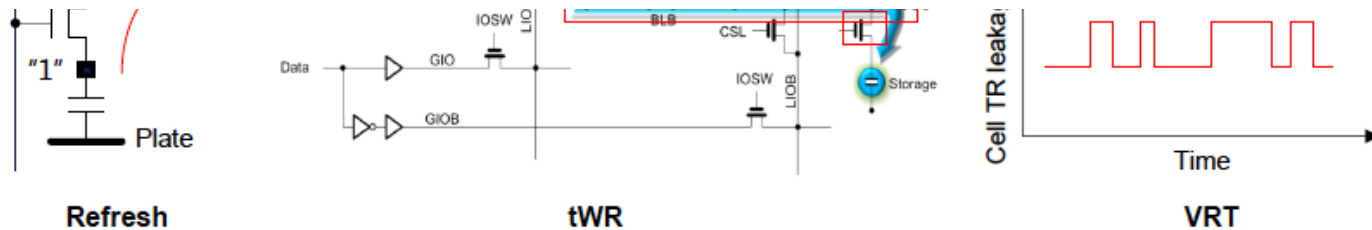
- Difficult to build high-aspect ratio cell capacitors decreasing cell capacitance

THE MEMORY FORUM 2014

Co-Architecting Controllers and DRAM to Enhance DRAM Process Scaling

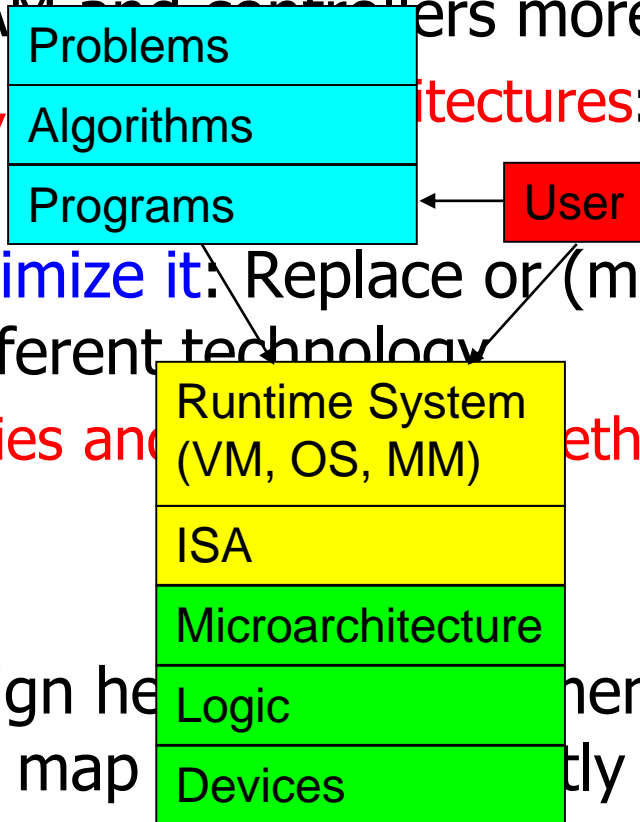
Uksong Kang, Hak-soo Yu, Churoo Park, *Hongzhong Zheng,
**John Halbert, **Kuljit Bains, SeongJin Jang, and Joo Sun Choi

*Samsung Electronics, Hwasung, Korea / *Samsung Electronics, San Jose / **Intel*



How Do We Solve The Problem?

- **Fix it:** Make DRAM and controllers more intelligent
 - **New interfaces, architectures:** system-DRAM codesign
- **Eliminate or minimize it:** Replace or (more likely) augment DRAM with a different technology
 - **New technologies and storage**
- **Embrace it:** Design heterogeneous memories (none of which are perfect) and map applications directly across them
 - **New models for data management and maybe usage**



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

Solution 1: Fix DRAM

- Overcome DRAM shortcomings with
 - ❑ System-DRAM co-design
 - ❑ Novel DRAM architectures, interface, functions
 - ❑ Better waste management (efficient utilization)

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

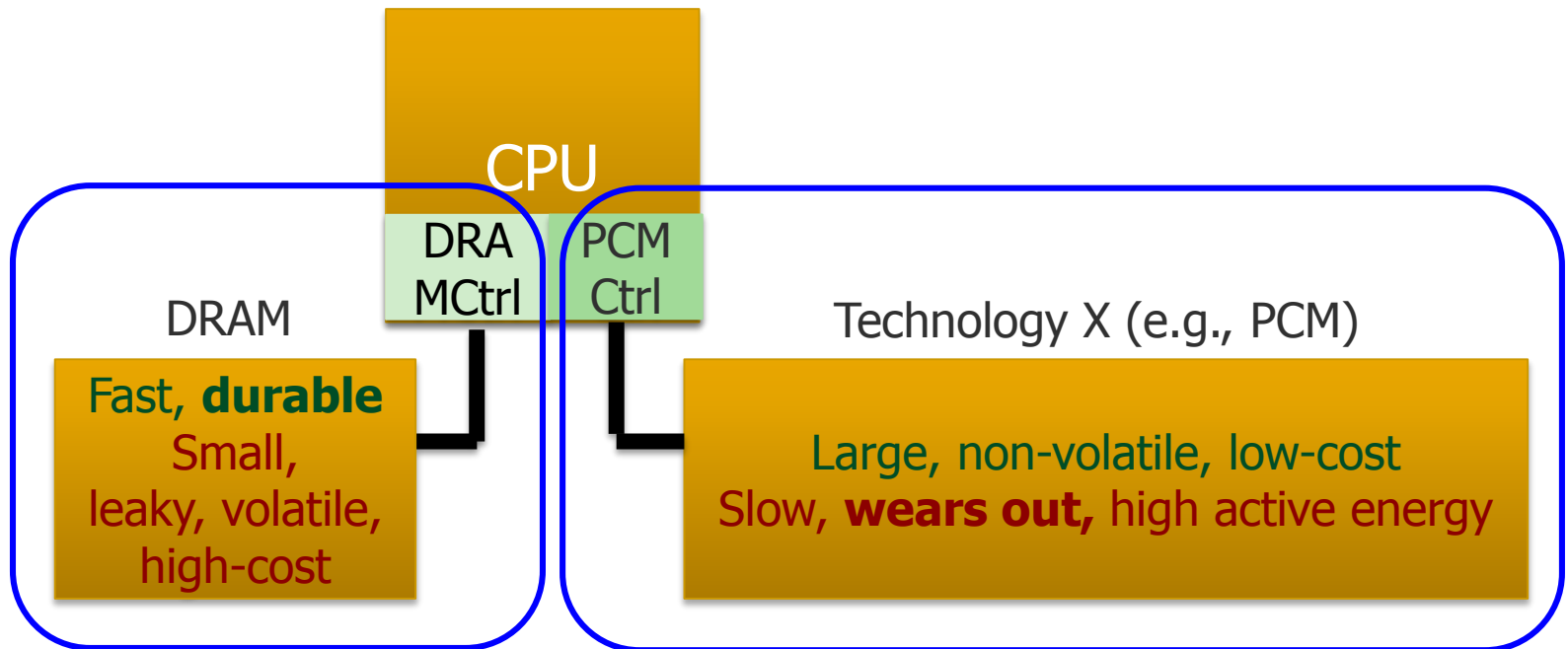
Solution 1: Fix DRAM

- 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.
 - 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.
-

Solution 2: Emerging Memory Technologies

- Some emerging resistive memory technologies seem more scalable than DRAM (and they are non-volatile)
- Example: Phase Change Memory
 - ❑ Expected to scale to 9nm (2022 [ITRS])
 - ❑ Expected to be denser than DRAM: can store multiple bits/cell
- But, emerging technologies have shortcomings as well
 - ❑ Can they be enabled to replace/augment/surpass DRAM?
- Lee+, “Architecting Phase Change Memory as a Scalable DRAM Alternative,” ISCA’09, CACM’10, 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,” ACM TACO 2014.
- Ren+, “Dual-Scheme Checkpointing: A Software-Transparent Mechanism for Supporting Crash Consistency in Persistent Memory Systems,” MICRO 2015.

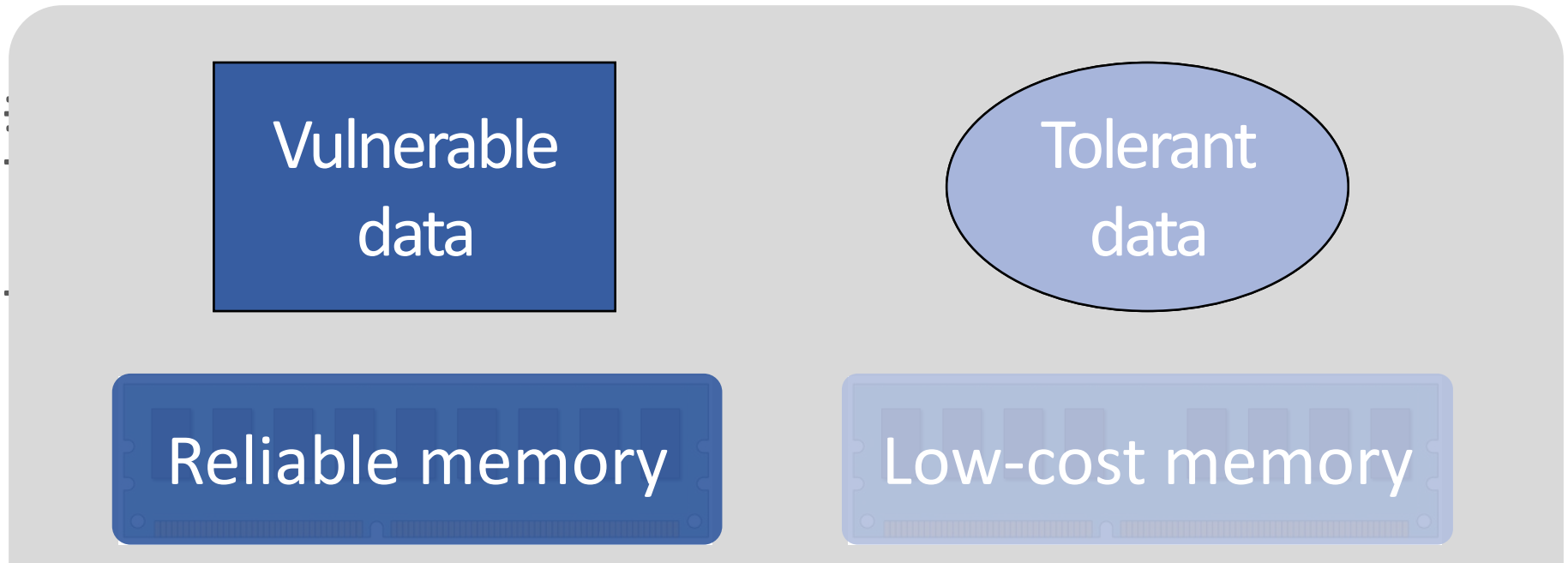
Solution 3: Hybrid Memory Systems



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



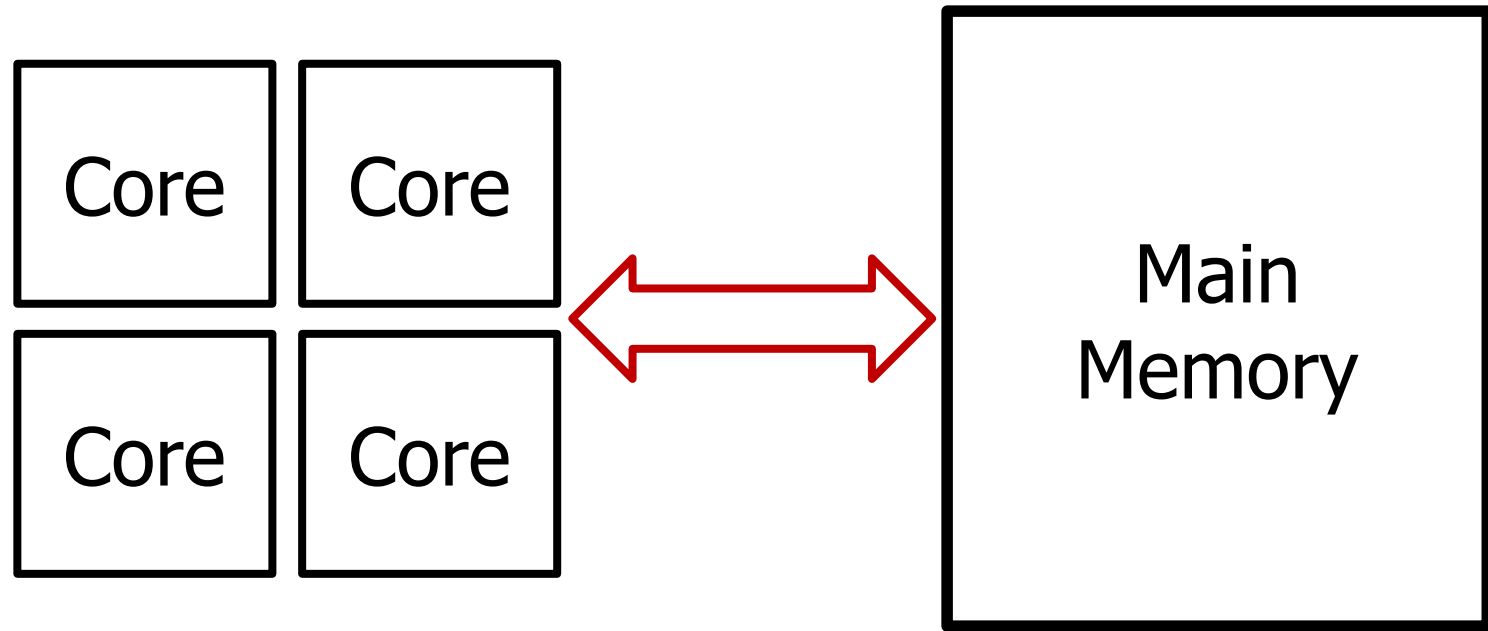
On Microsoft's Web Search workload

Reduces server hardware **cost** by **4.7 %**

Achieves single server **availability** target of **99.90 %**

Heterogeneous-Reliability Memory [DSN 2014]

An Orthogonal Issue: Memory Interference

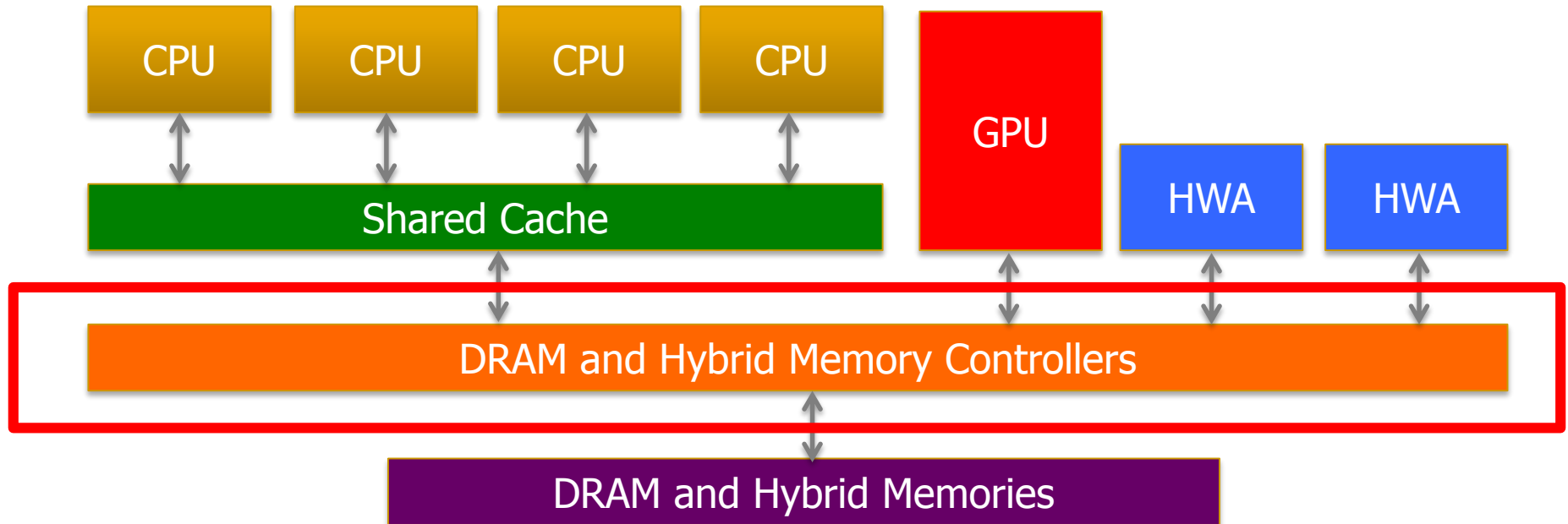


Cores' interfere with each other when accessing shared main memory

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

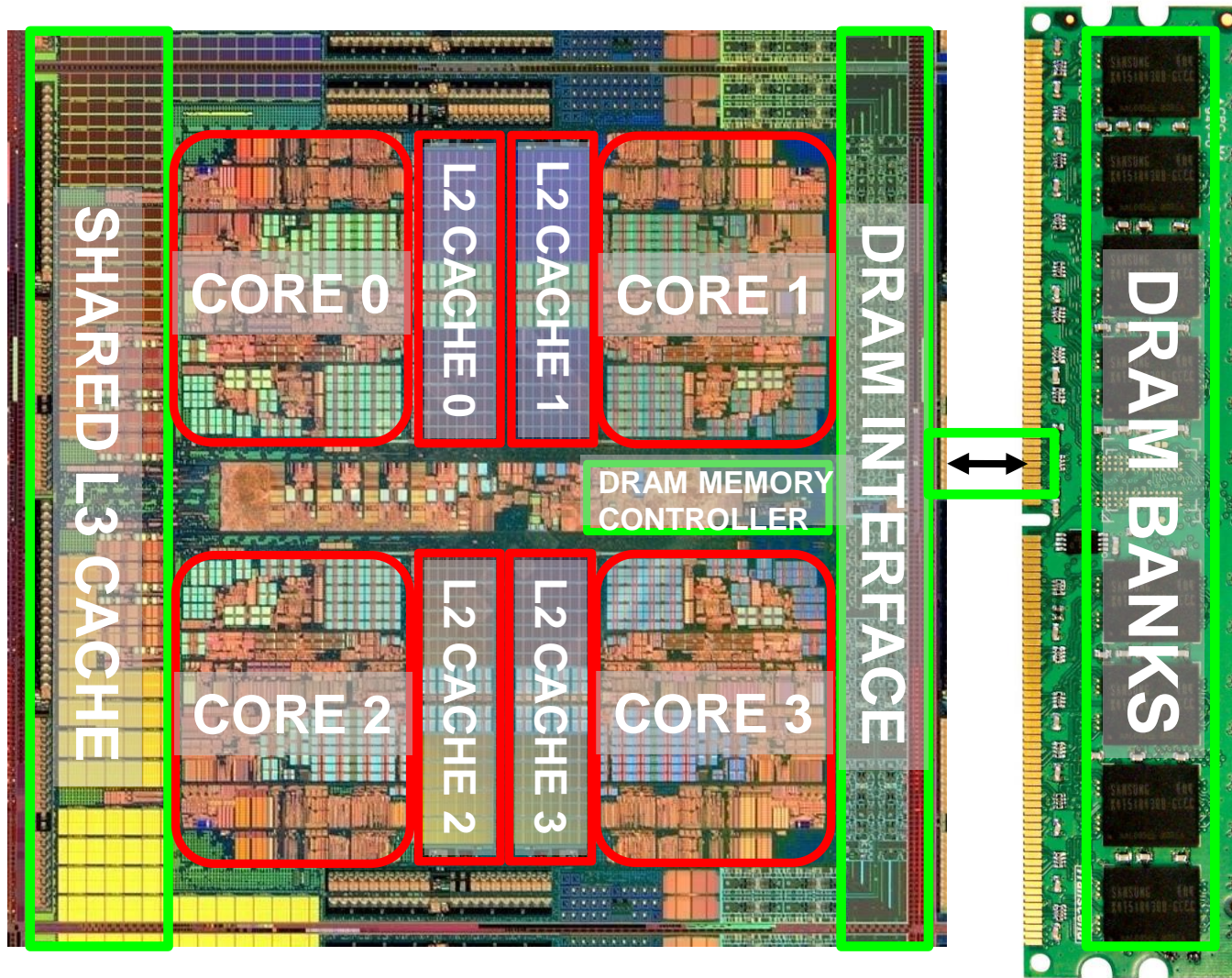
How to allocate resources to heterogeneous agents to mitigate interference and provide predictable performance?

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.

Main Memory Fundamentals

Main Memory in the System



Ideal Memory

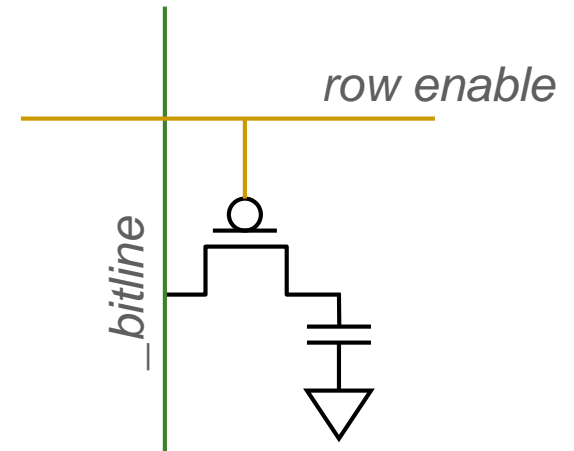
- Zero access time (latency)
- Infinite capacity
- Zero cost
- Infinite bandwidth (to support multiple accesses in parallel)

The Problem

- Ideal memory's requirements oppose each other
- Bigger is slower
 - Bigger → Takes longer to determine the location
- Faster is more expensive
 - Memory technology: SRAM vs. DRAM
- Higher bandwidth is more expensive
 - Need more banks, more ports, higher frequency, or faster technology

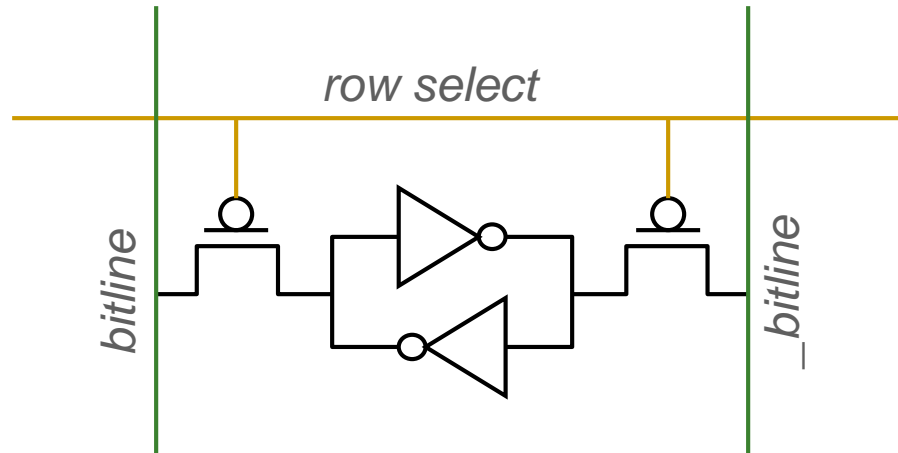
Memory Technology: DRAM

- Dynamic random access memory
- Capacitor charge state indicates stored value
 - Whether the capacitor is charged or discharged indicates storage of 1 or 0
 - 1 capacitor
 - 1 access transistor
- Capacitor leaks through the RC path
 - DRAM cell loses charge over time
 - DRAM cell needs to be refreshed
- Read Liu et al., “[RAIDR: Retention-aware Intelligent DRAM Refresh](#),” ISCA 2012.



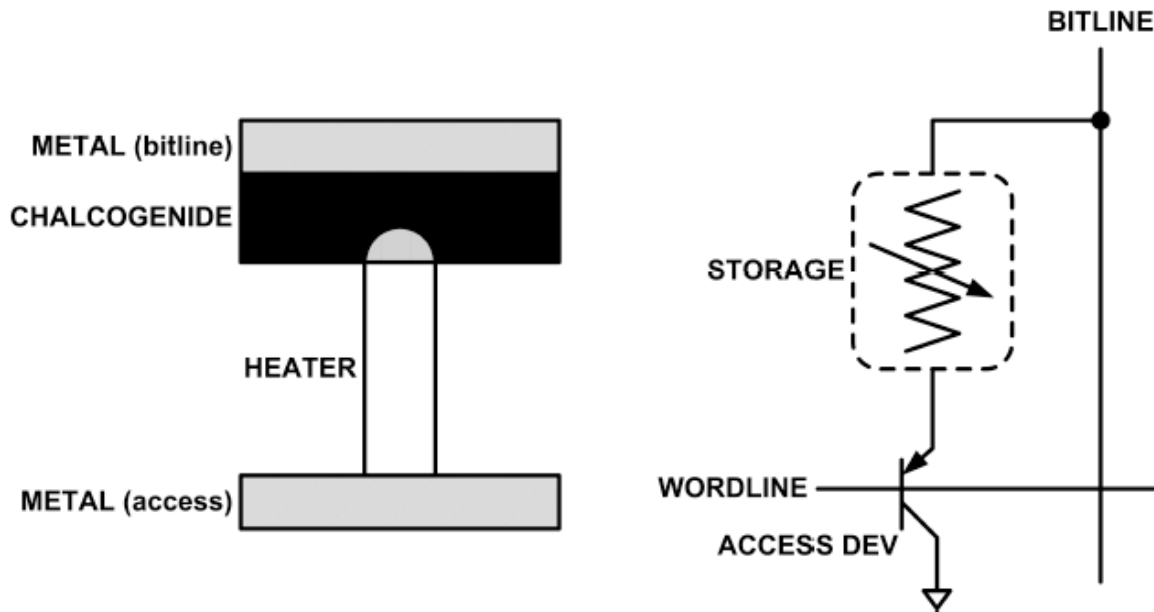
Memory Technology: SRAM

- Static random access memory
- Two cross coupled inverters store a single bit
 - ❑ Feedback path enables the stored value to persist in the “cell”
 - ❑ 4 transistors for storage
 - ❑ 2 transistors for access



An Aside: Phase Change Memory

- Phase change material (chalcogenide glass) exists in two states:
 - Amorphous: Low optical reflexivity and high electrical resistivity
 - Crystalline: High optical reflexivity and low electrical resistivity



PCM is resistive memory: High resistance (0), Low resistance (1)

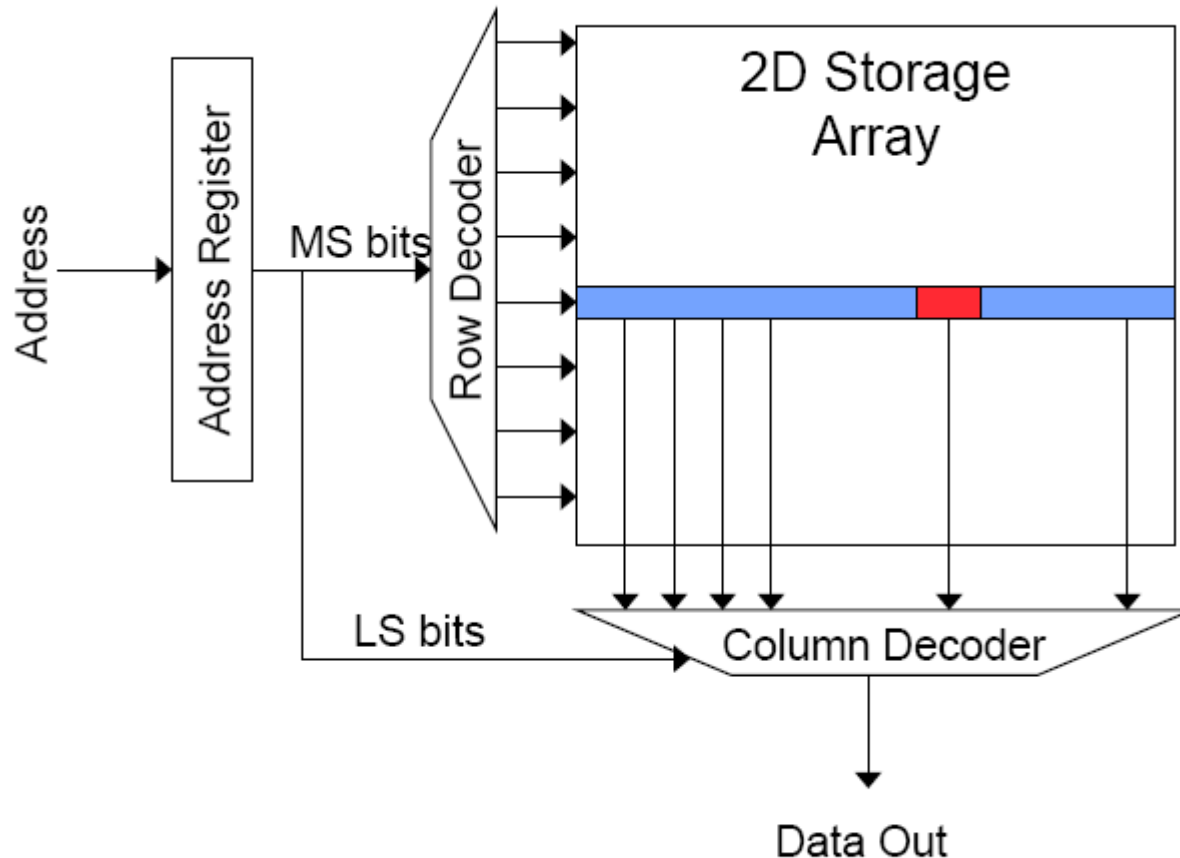
Lee, Ipek, Mutlu, Burger, “[Architecting Phase Change Memory as a Scalable DRAM Alternative](#),” ISCA 2009.

Memory Bank: A Fundamental Concept

■ 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
- ❑ **An issue:** How do you map data to different banks? (i.e., how do you interleave data across banks?)

Memory Bank Organization and Operation



■ Read access sequence:

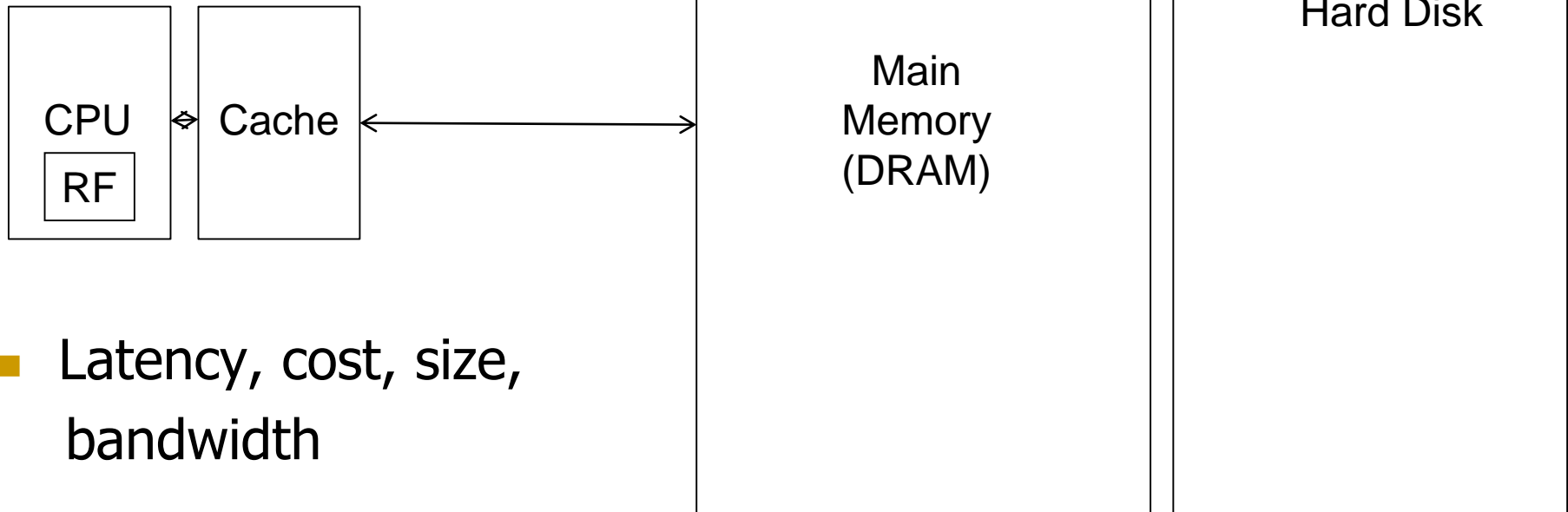
1. Decode row address & drive word-lines
2. Selected bits drive bit-lines
 - Entire row read
3. Amplify row data
4. Decode column address & select subset of row
 - Send to output
5. Precharge bit-lines
 - For next access

Why Memory Hierarchy?

- We want both fast and large
- But we cannot achieve both with a single level of memory
- Idea: Have multiple levels of storage (progressively bigger and slower as the levels are farther from the processor) and ensure most of the data the processor needs is kept in the fast(er) level(s)

Memory Hierarchy

- Fundamental tradeoff
 - Fast memory: small
 - Large memory: slow
- Idea: **Memory hierarchy**



- Latency, cost, size, bandwidth

Caching Basics: Exploit Temporal Locality

- Idea: Store recently accessed data in automatically managed fast memory (called cache)
- Anticipation: the data will be accessed again soon
- Temporal locality principle
 - Recently accessed data will be again accessed in the near future
 - This is what Maurice Wilkes had in mind:
 - Wilkes, “Slave Memories and Dynamic Storage Allocation,” IEEE Trans. On Electronic Computers, 1965.
 - “The use is discussed of a fast core memory of, say 32000 words as a slave to a slower core memory of, say, one million words in such a way that in practical cases the effective access time is nearer that of the fast memory than that of the slow memory.”

Caching Basics: Exploit Spatial Locality

- Idea: Store addresses adjacent to the recently accessed one in automatically managed fast memory
 - Logically divide memory into equal size blocks
 - Fetch to cache the accessed block in its entirety
- Anticipation: nearby data will be accessed soon
- Spatial locality principle
 - Nearby data in memory will be accessed in the near future
 - E.g., sequential instruction access, array traversal
 - This is what IBM 360/85 implemented
 - 16 Kbyte cache with 64 byte blocks
 - Liptay, “Structural aspects of the System/360 Model 85 II: the cache,” IBM Systems Journal, 1968.

A Note on Manual vs. Automatic Management

- **Manual:** Programmer manages data movement across levels
 - too painful for programmers on substantial programs
 - “core” vs “drum” memory in the 50’s
 - still done in some embedded processors (on-chip scratch pad SRAM in lieu of a cache)

- **Automatic:** Hardware manages data movement across levels, transparently to the programmer
 - ++ programmer’s life is easier
 - simple heuristic: keep most recently used items in cache
 - the average programmer doesn’t need to know about it
 - You don’t need to know how big the cache is and how it works to write a “correct” program! (What if you want a “fast” program?)

Automatic Management in Memory Hierarchy

- Wilkes, “**Slave Memories and Dynamic Storage Allocation**,” IEEE Trans. On Electronic Computers, 1965.

Slave Memories and Dynamic Storage Allocation

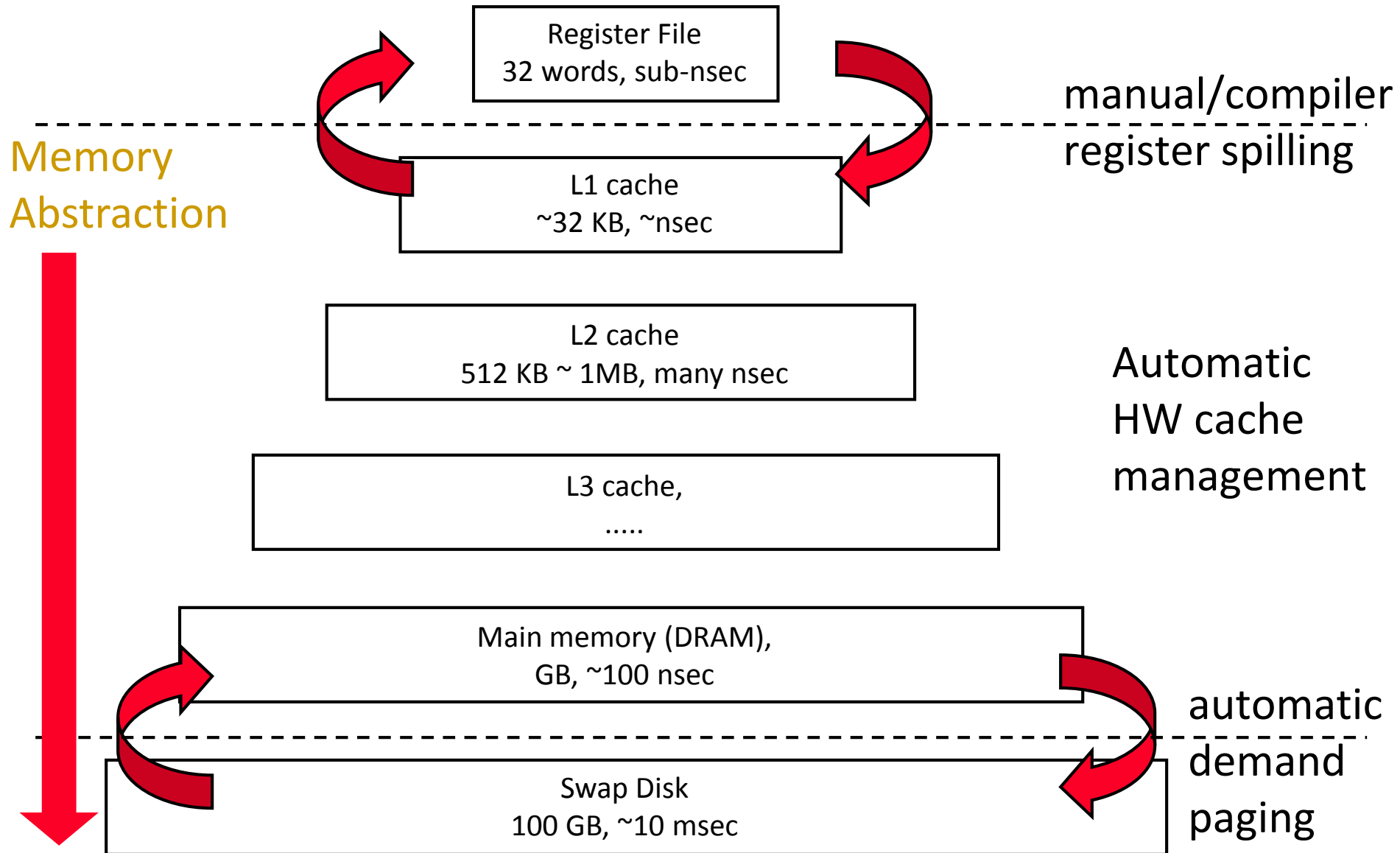
M. V. WILKES

SUMMARY

The use is discussed of a fast core memory of, say, 32 000 words as a slave to a slower core memory of, say, one million words in such a way that in practical cases the effective access time is nearer that of the fast memory than that of the slow memory.

- “By a slave memory I mean one which **automatically accumulates to itself words** that come from a slower main memory, and keeps them available for subsequent use without it being necessary for the penalty of main memory access to be incurred again.”

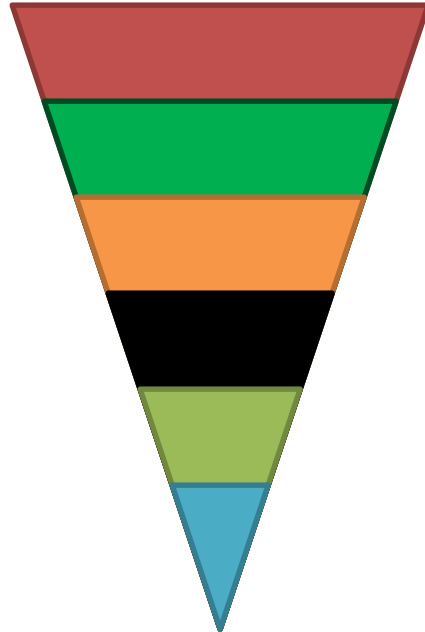
A Modern Memory Hierarchy



The DRAM Subsystem

DRAM Subsystem Organization

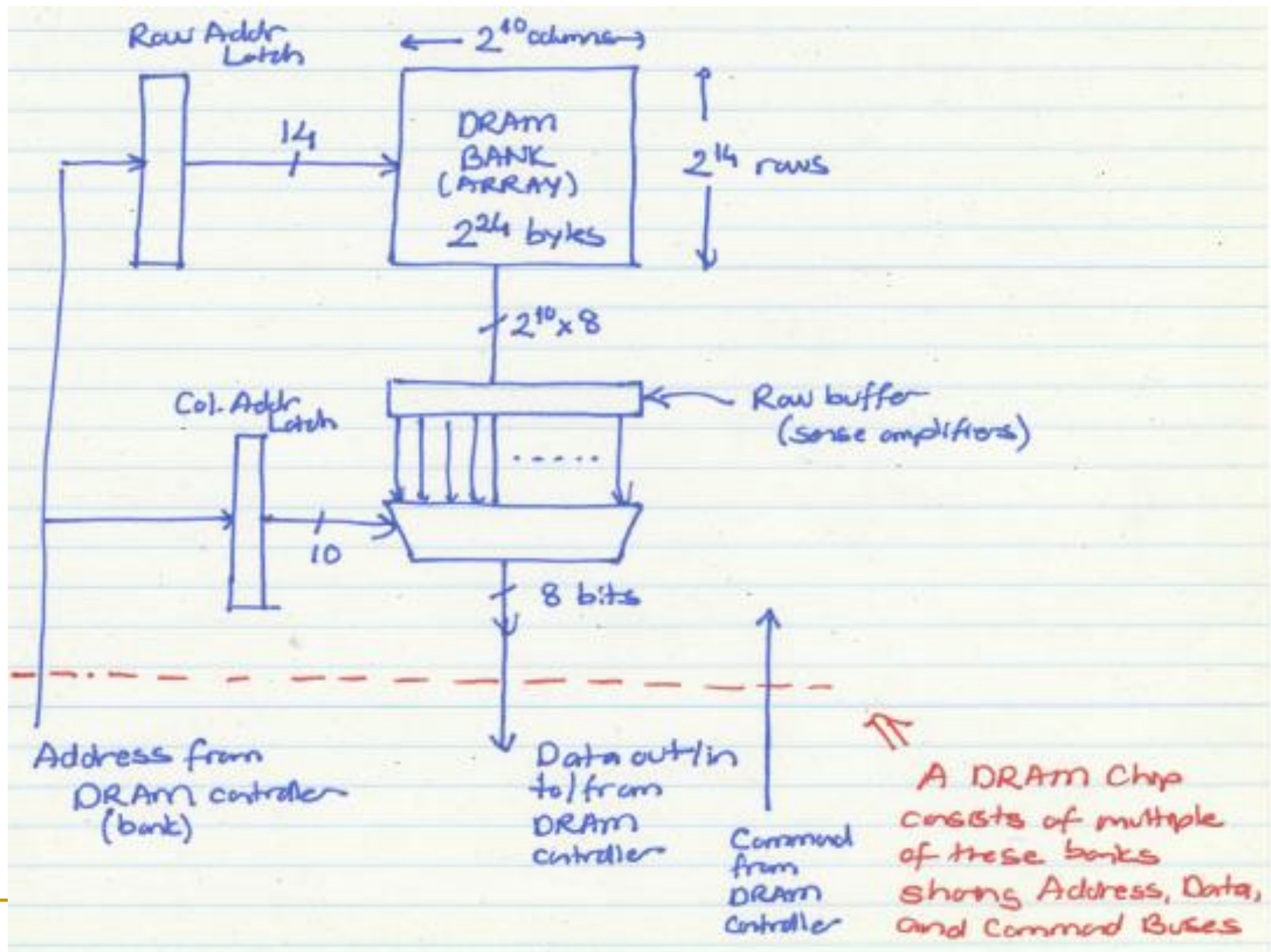
- Channel
- DIMM
- Rank
- Chip
- Bank
- Row/Column



Page Mode DRAM

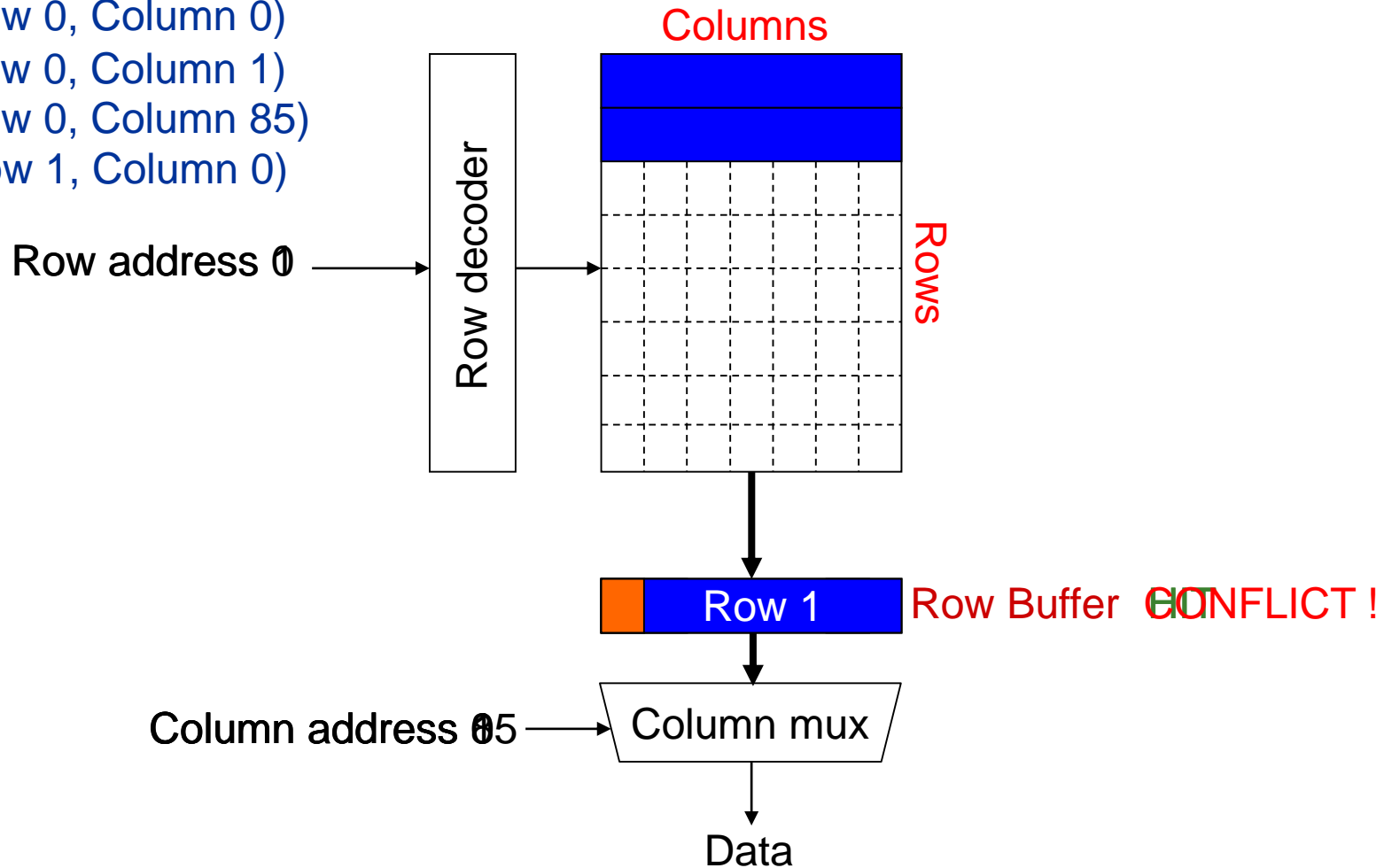
- A DRAM bank is a 2D array of cells: rows x columns
- A “DRAM row” is also called a “DRAM page”
- “Sense amplifiers” also called “row buffer”
- Each address is a <row,column> pair
- Access to a “closed row”
 - **Activate** command opens row (placed into row buffer)
 - **Read/write** command reads/writes 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

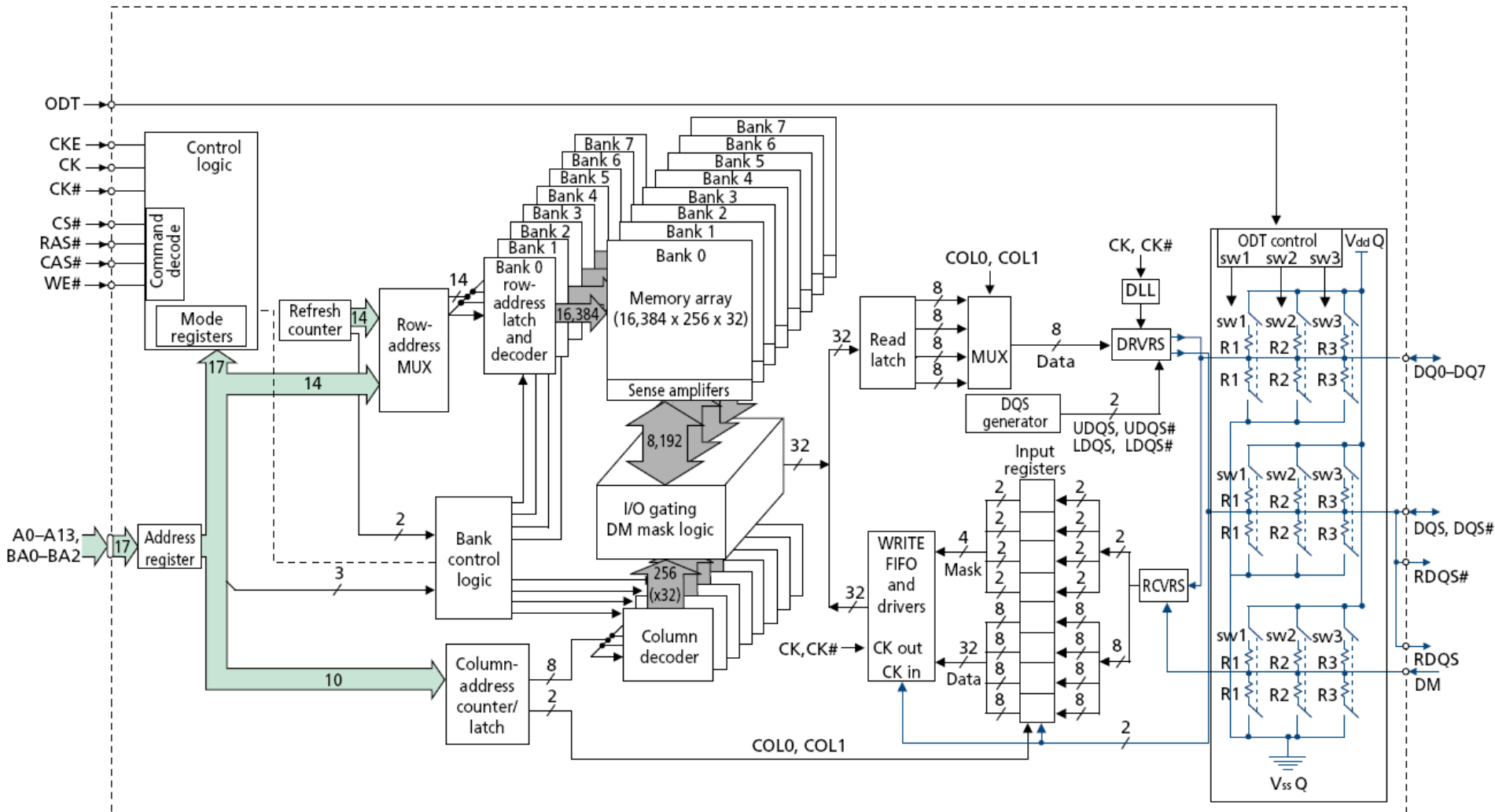
Access Address:
(Row 0, Column 0)
(Row 0, Column 1)
(Row 0, Column 85)
(Row 1, Column 0)



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

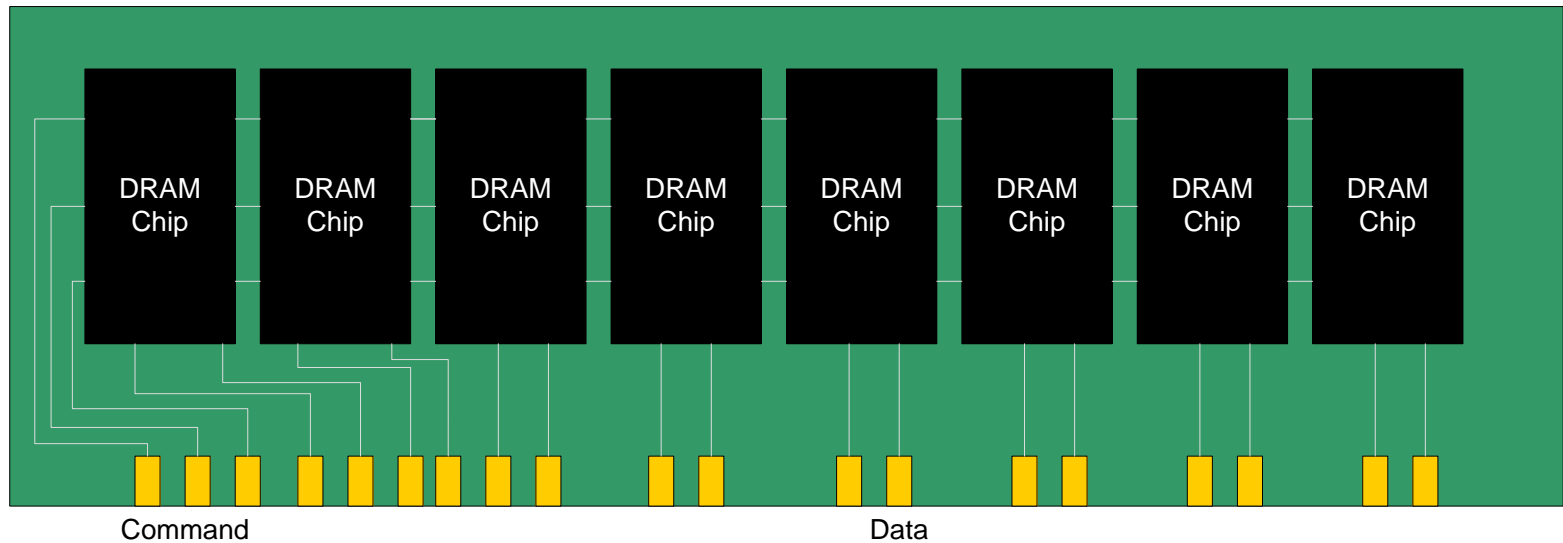
128M x 8-bit DRAM Chip



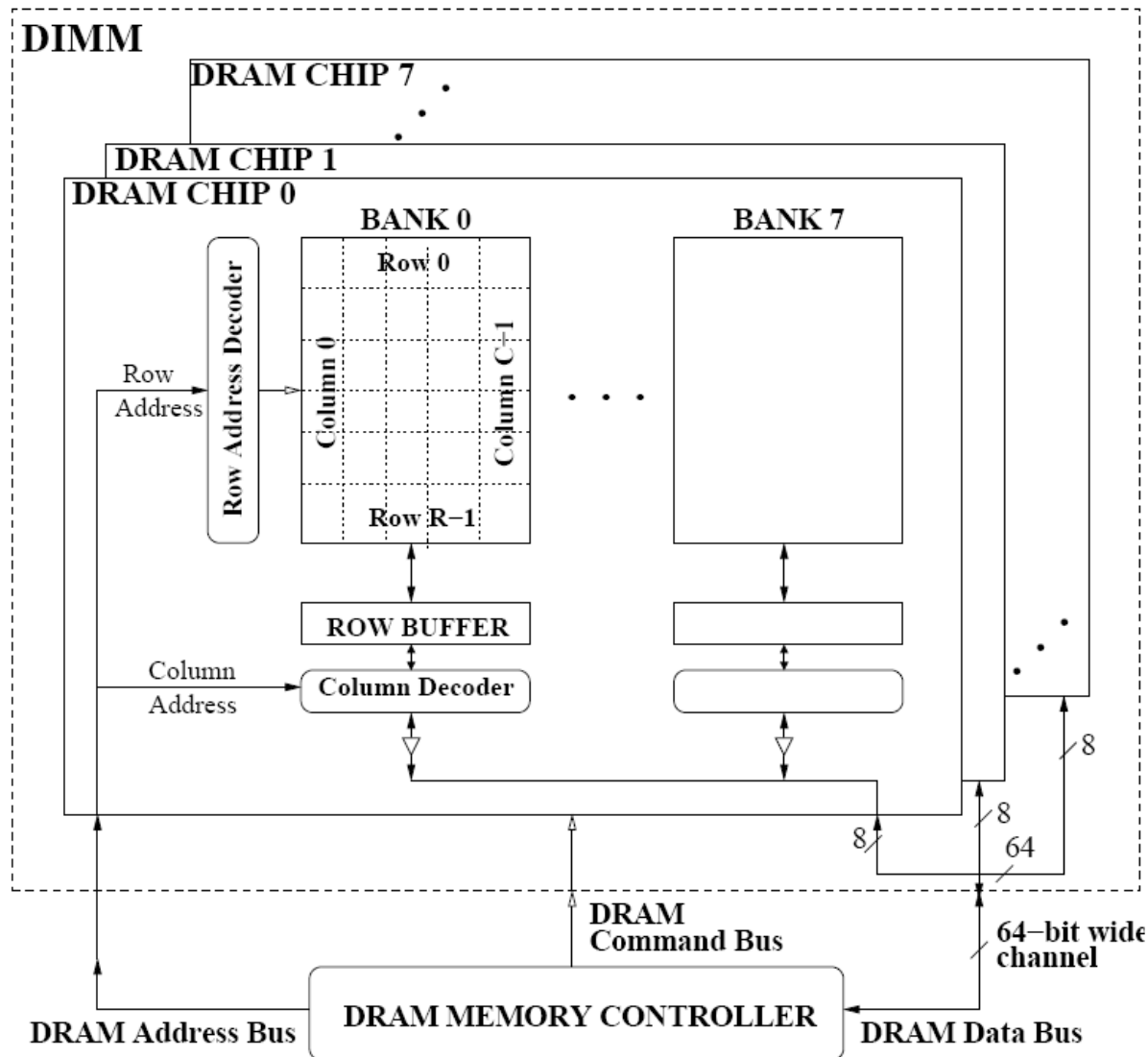
DRAM Rank and Module

- 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)

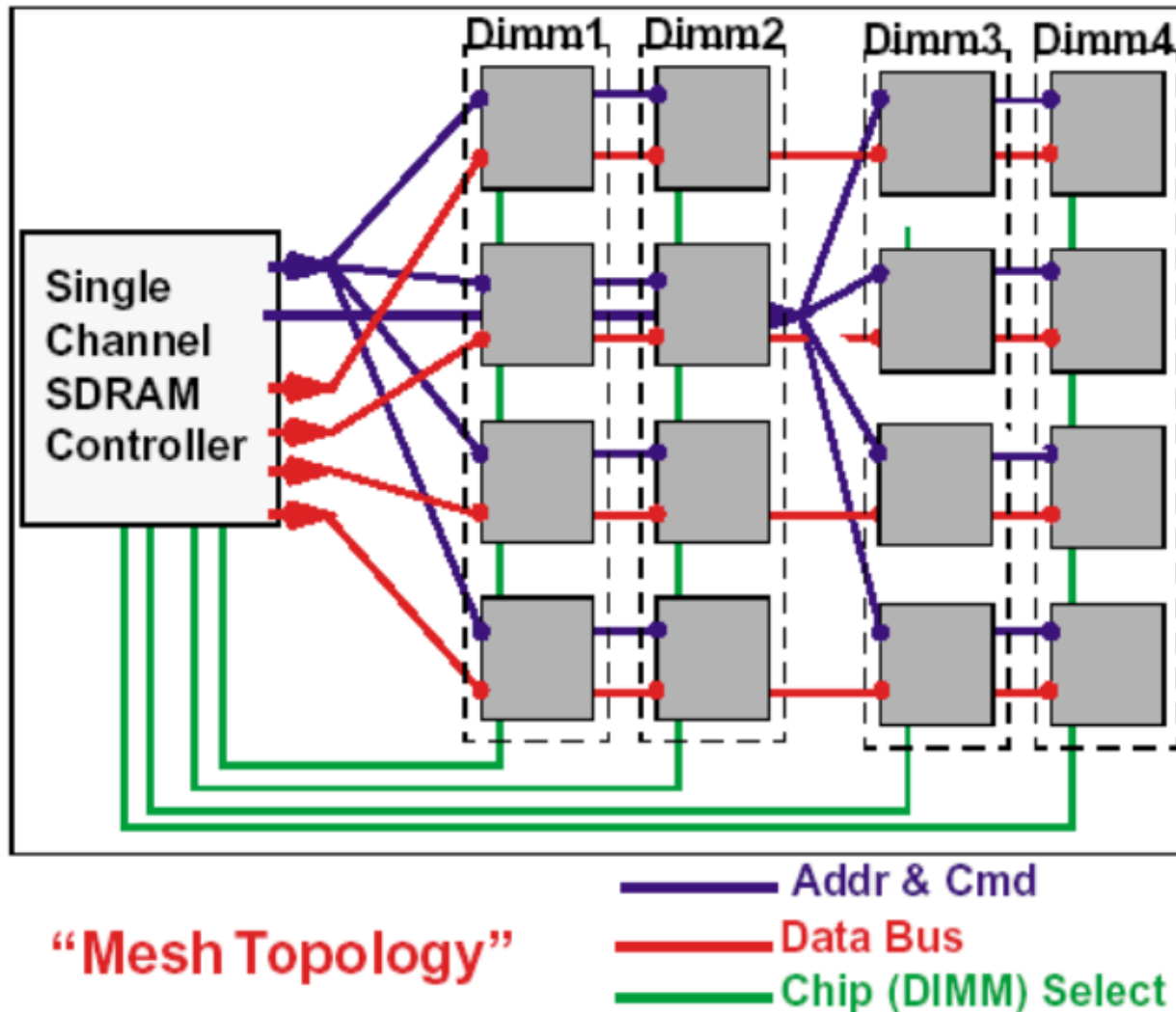


A 64-bit Wide DIMM (One Rank)



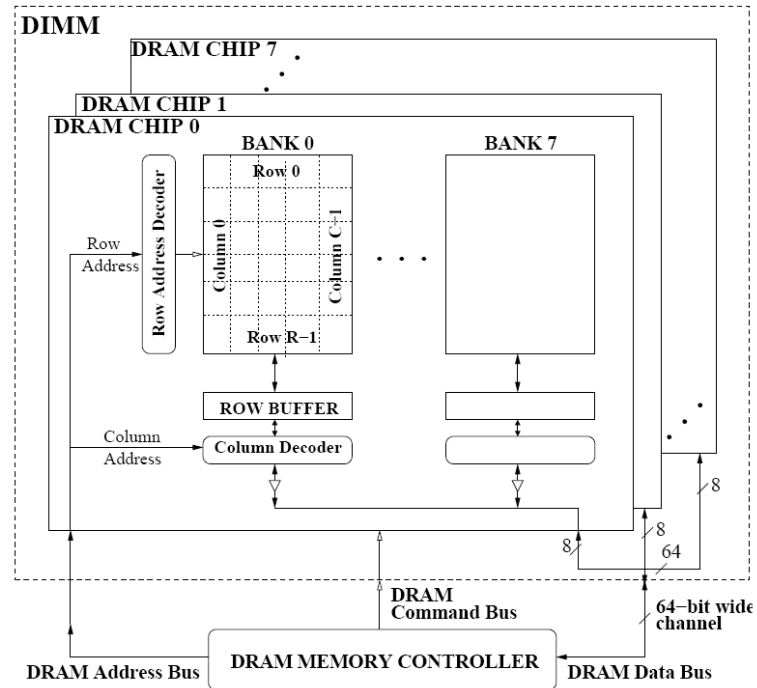
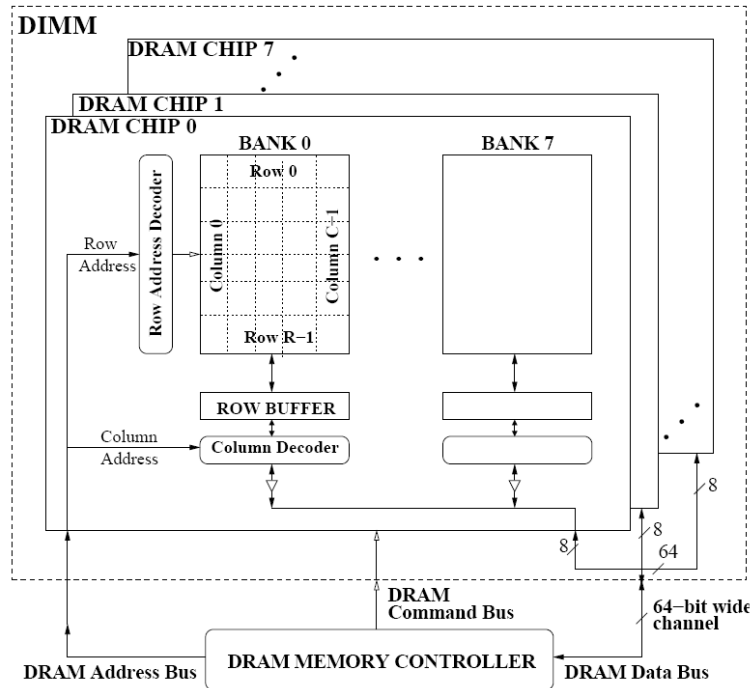
- **Advantages:**
 - Acts like a **high-capacity DRAM chip** with a **wide interface**
 - **Flexibility:** memory controller does not need to deal with individual chips
- **Disadvantages:**
 - **Granularity:** Accesses cannot be smaller than the interface width

Multiple DIMMs



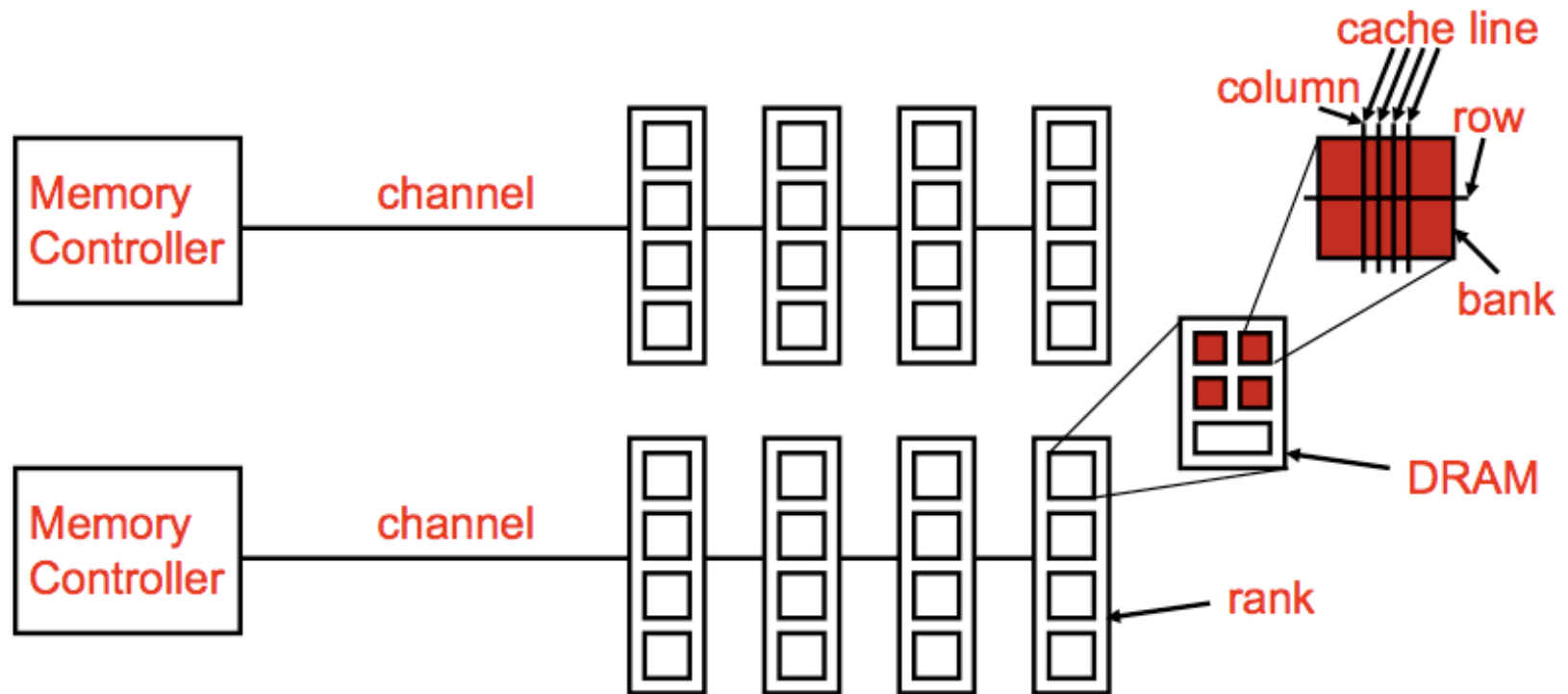
- Advantages:
 - Enables even higher capacity
- Disadvantages:
 - Interconnect complexity and energy consumption can be high
→ Scalability is limited by this

DRAM Channels

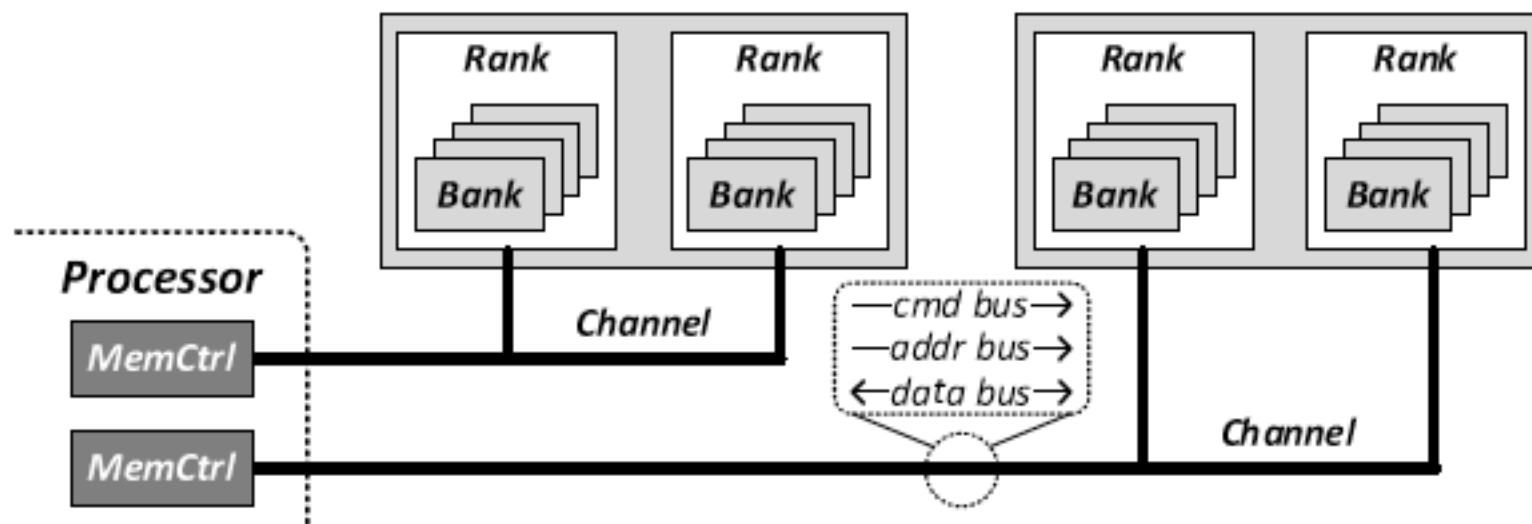


- 2 Independent Channels: 2 Memory Controllers (Above)
- 2 Dependent/Lockstep Channels: 1 Memory Controller with wide interface (Not shown above)

Generalized Memory Structure



Generalized Memory Structure



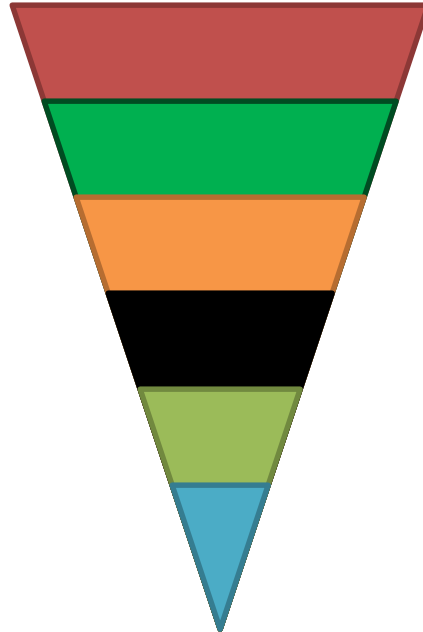
Kim+, "A Case for Exploiting Subarray-Level Parallelism in DRAM," ISCA 2012.

The DRAM Subsystem

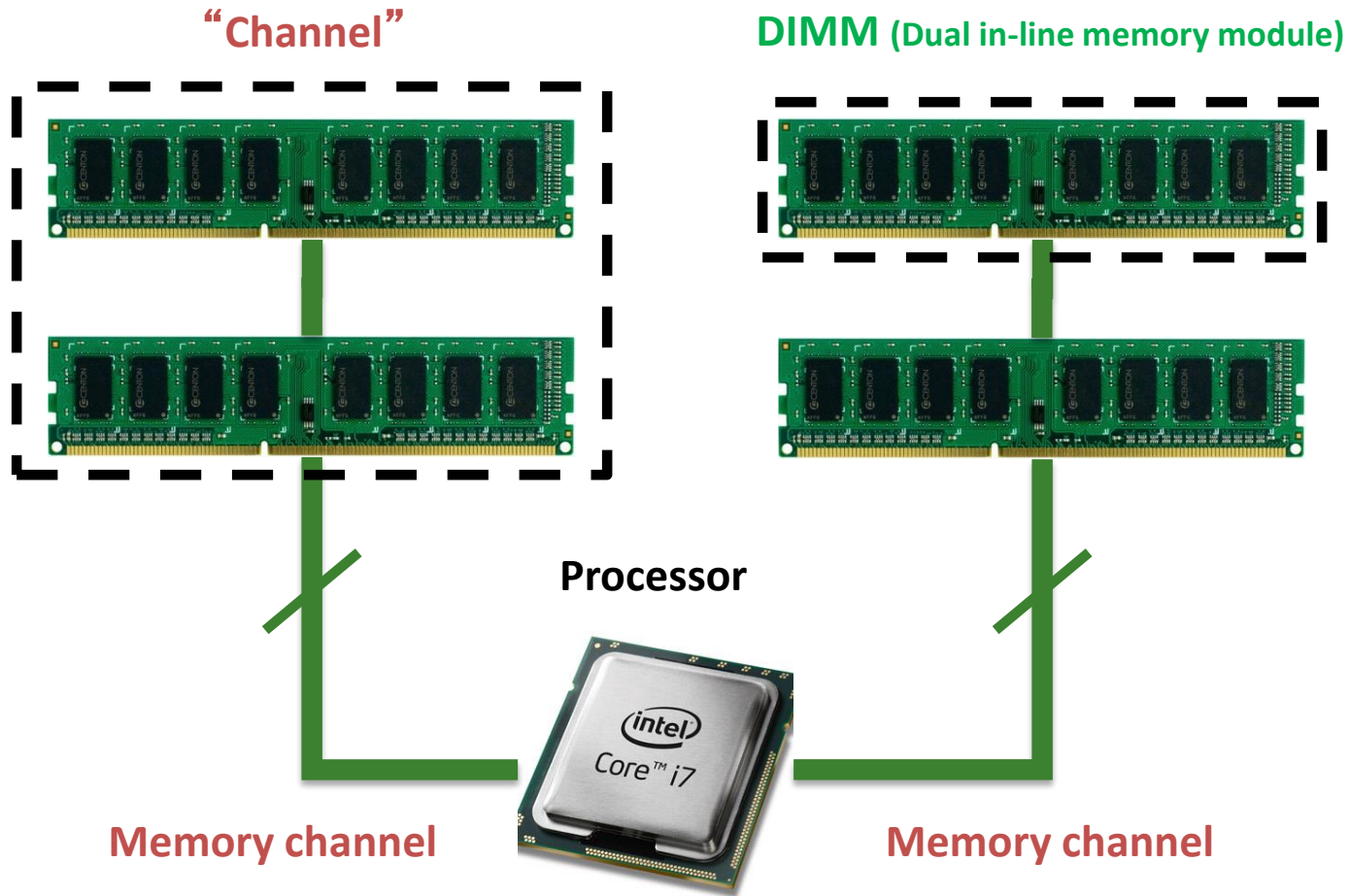
The Top Down View

DRAM Subsystem Organization

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



The DRAM subsystem



Breaking down a DIMM

DIMM (Dual in-line memory module)



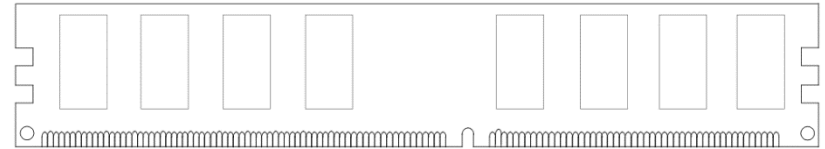
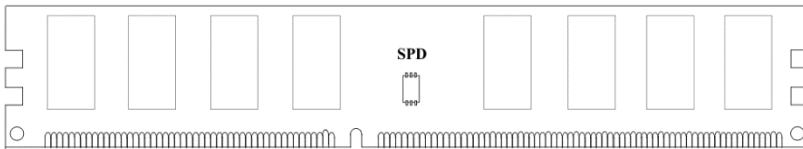
Side view

SIDE

4.00

Front of DIMM

Back of DIMM



Breaking down a DIMM

DIMM (Dual in-line memory module)



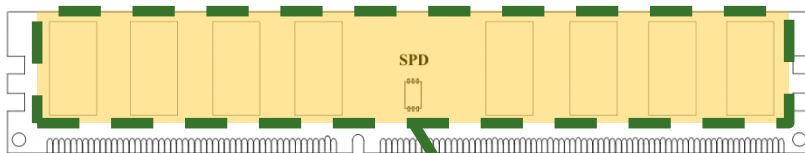
Side view

SIDE

4.00

Front of DIMM

Back of DIMM

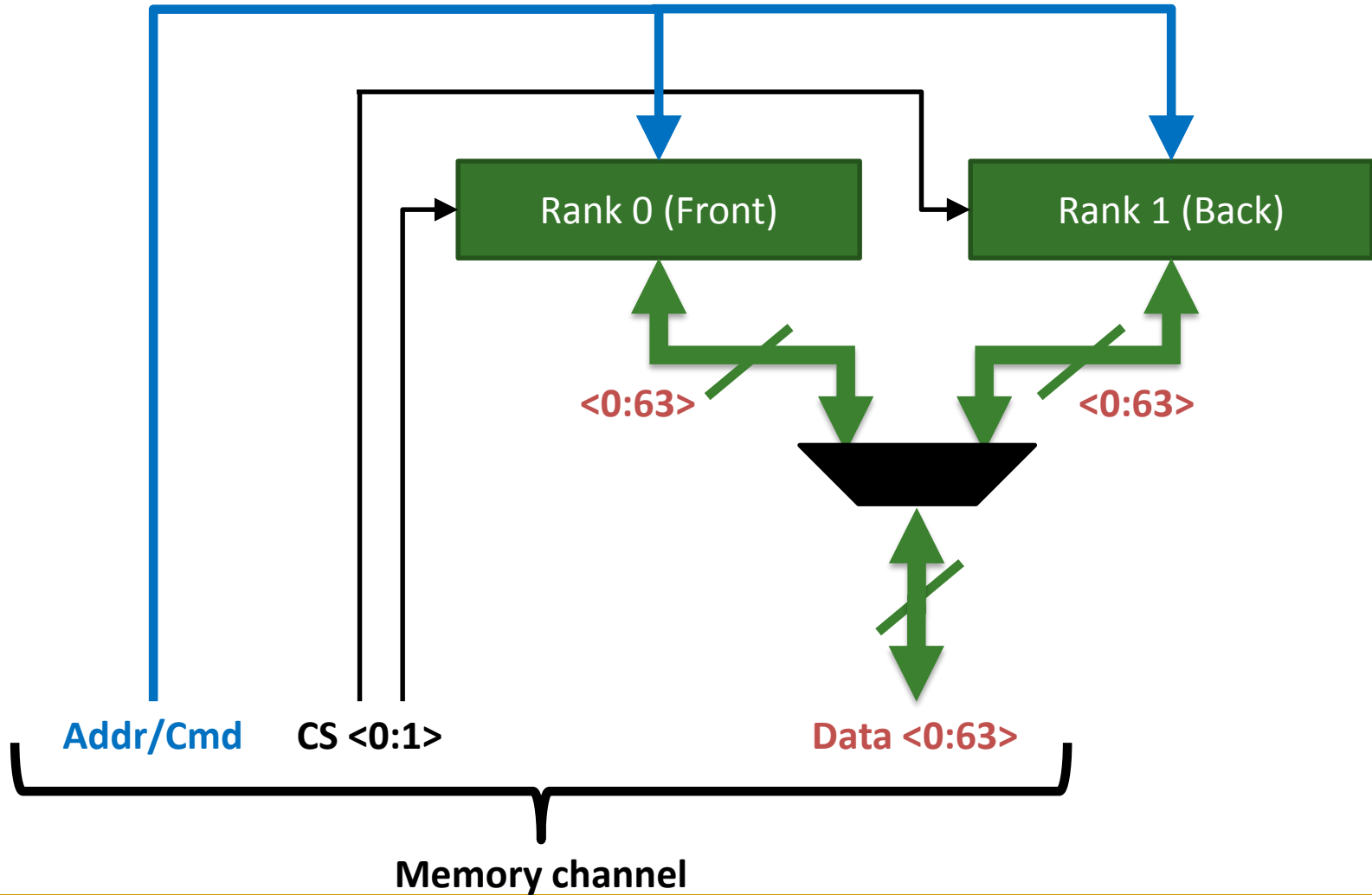


Rank 0: collection of 8 chips

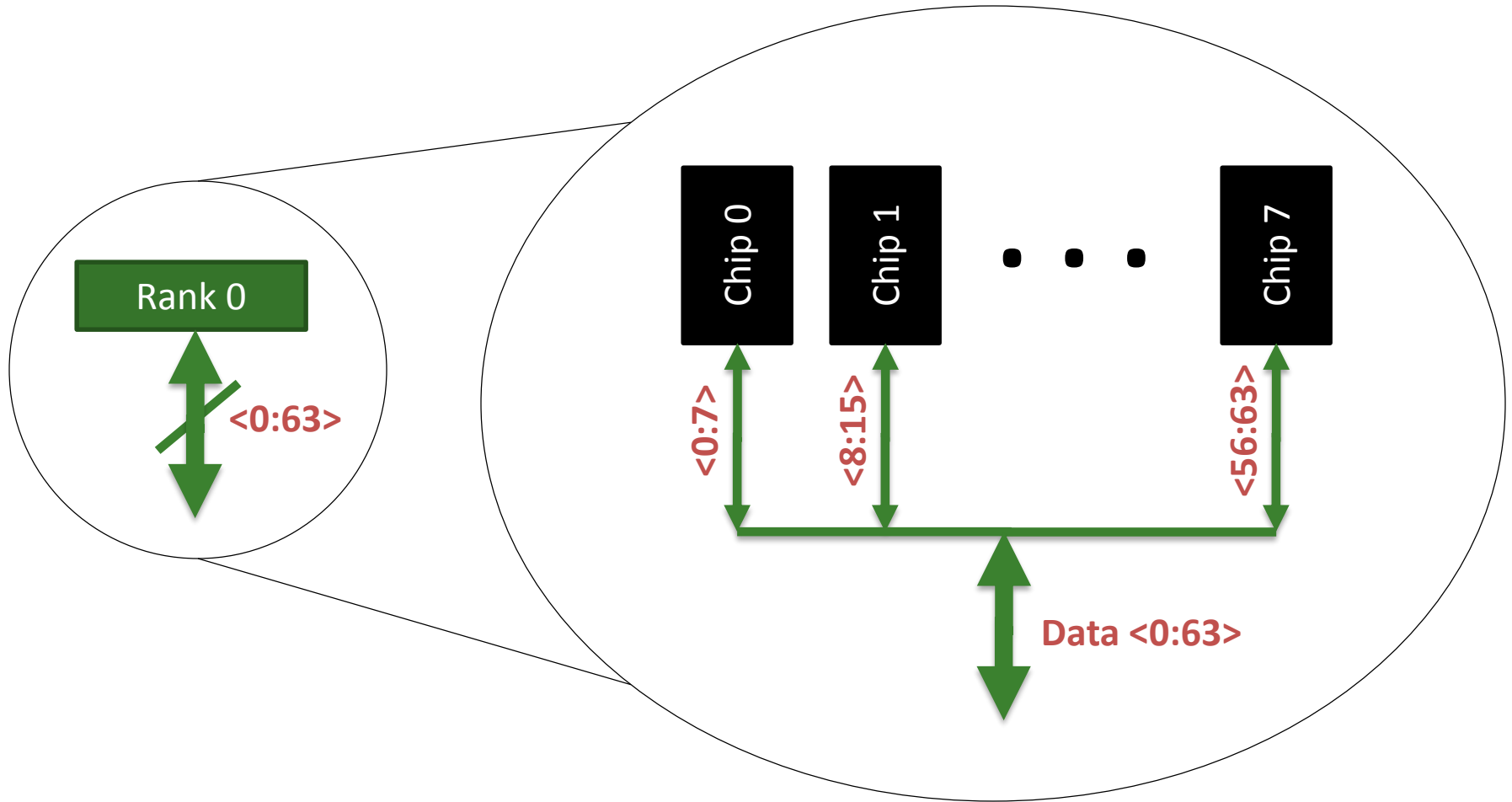


Rank 1

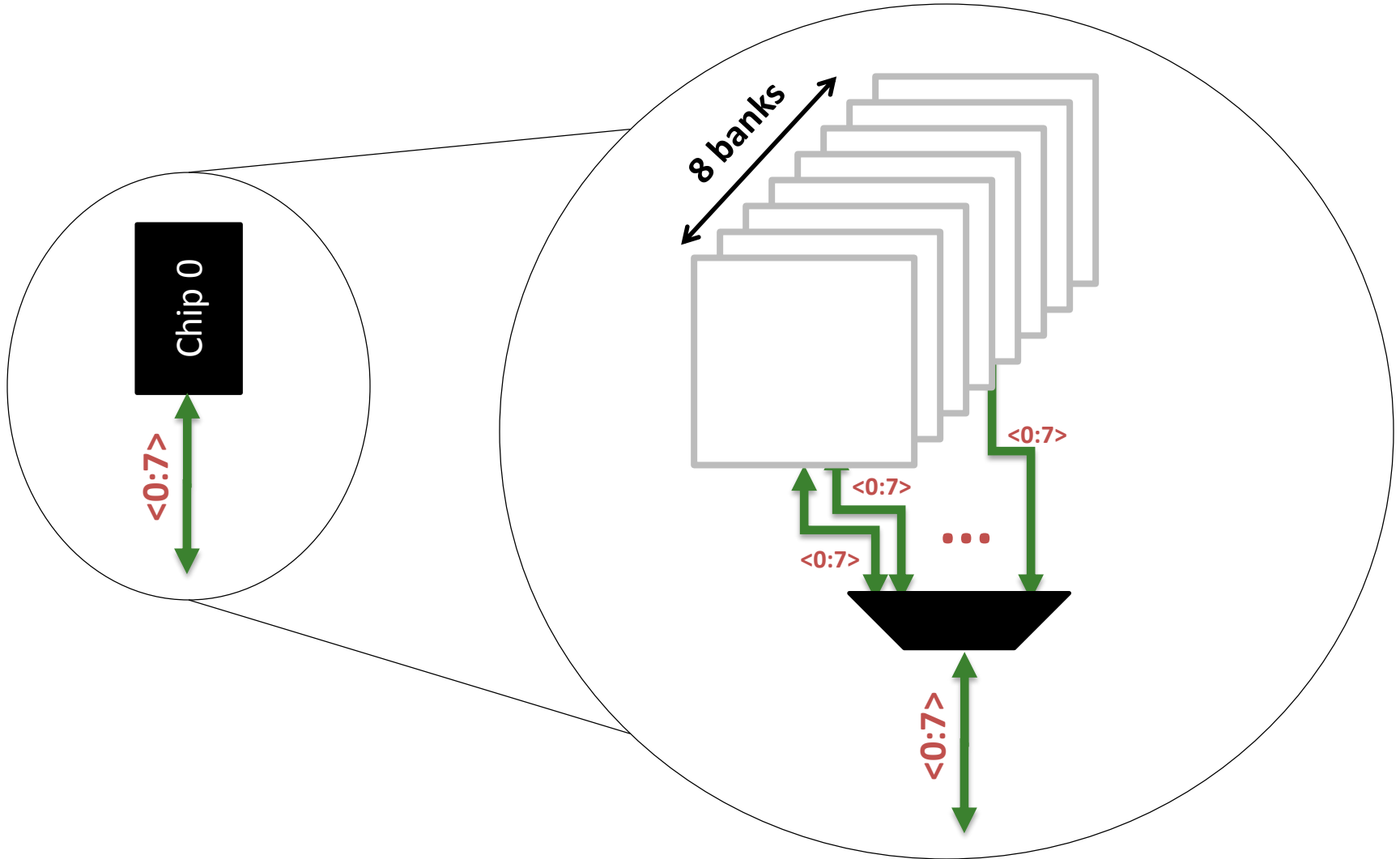
Rank



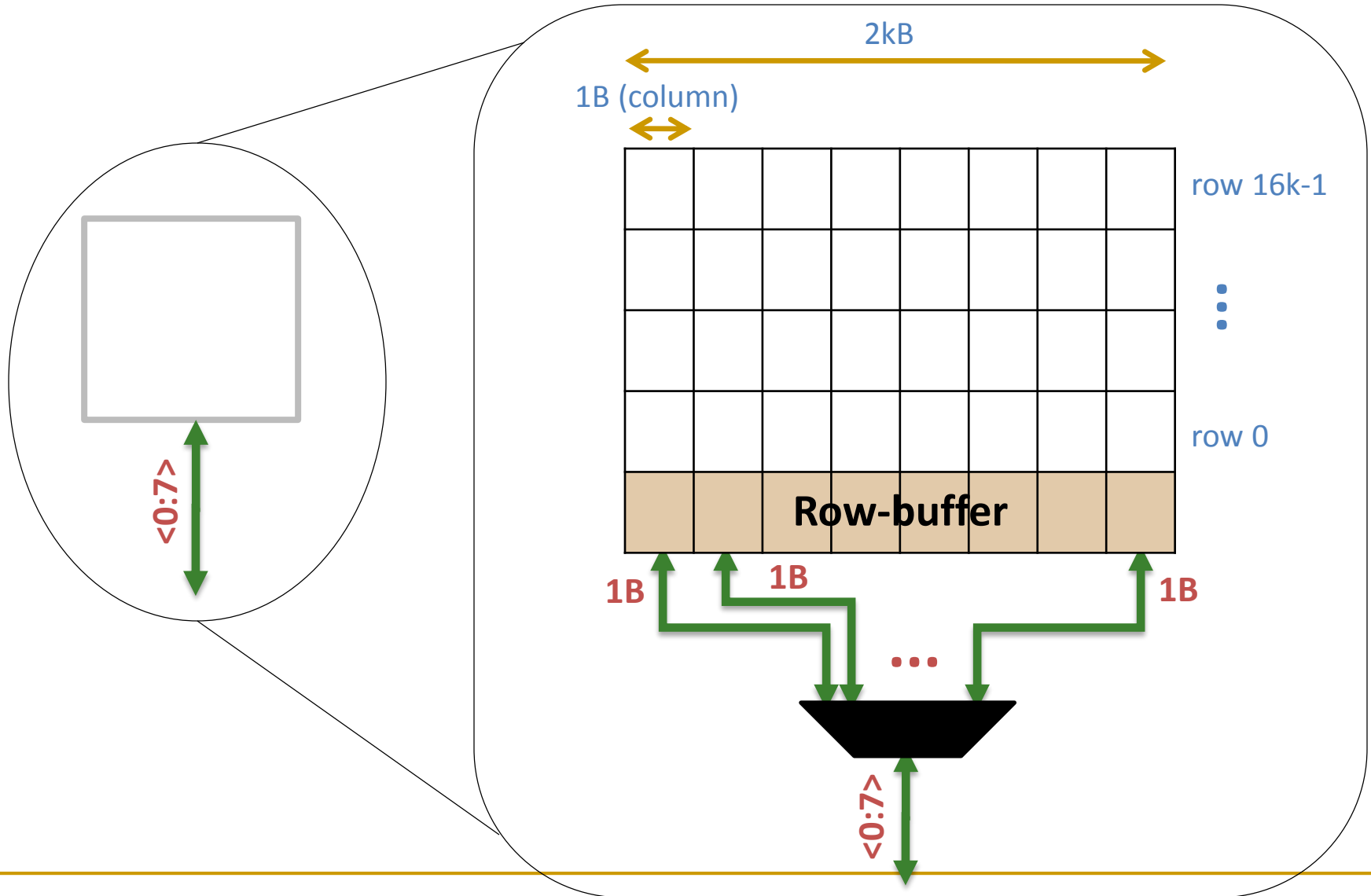
Breaking down a Rank



Breaking down a Chip

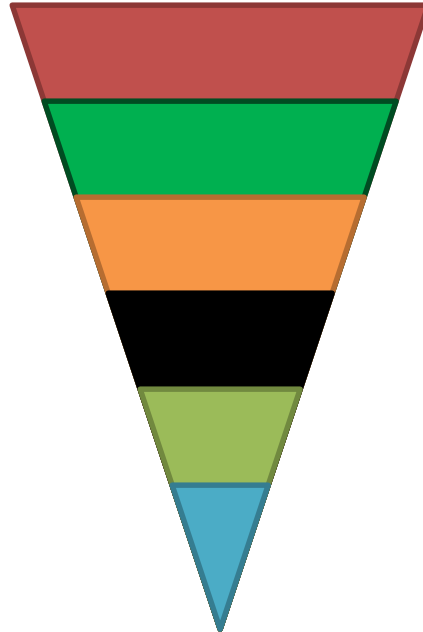


Breaking down a Bank



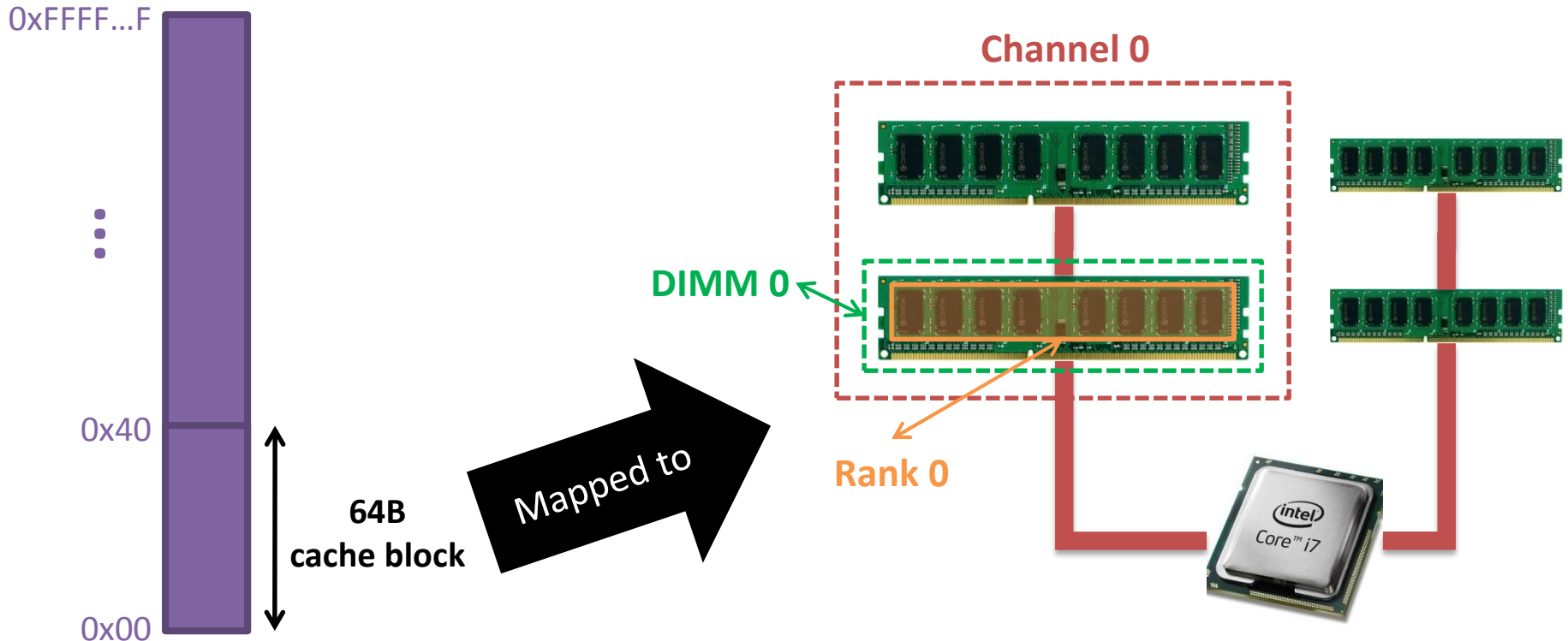
DRAM Subsystem Organization

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



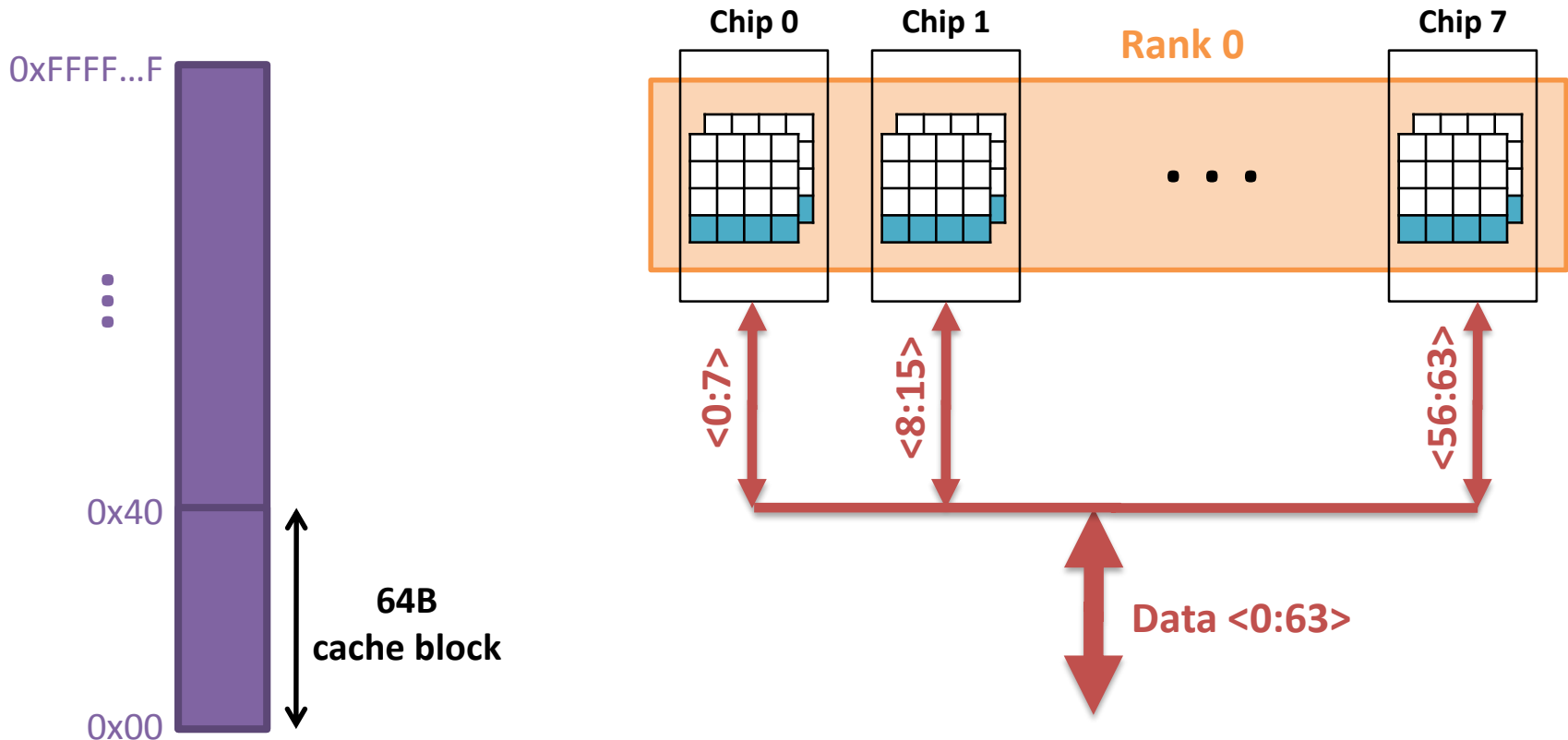
Example: Transferring a cache block

Physical memory space



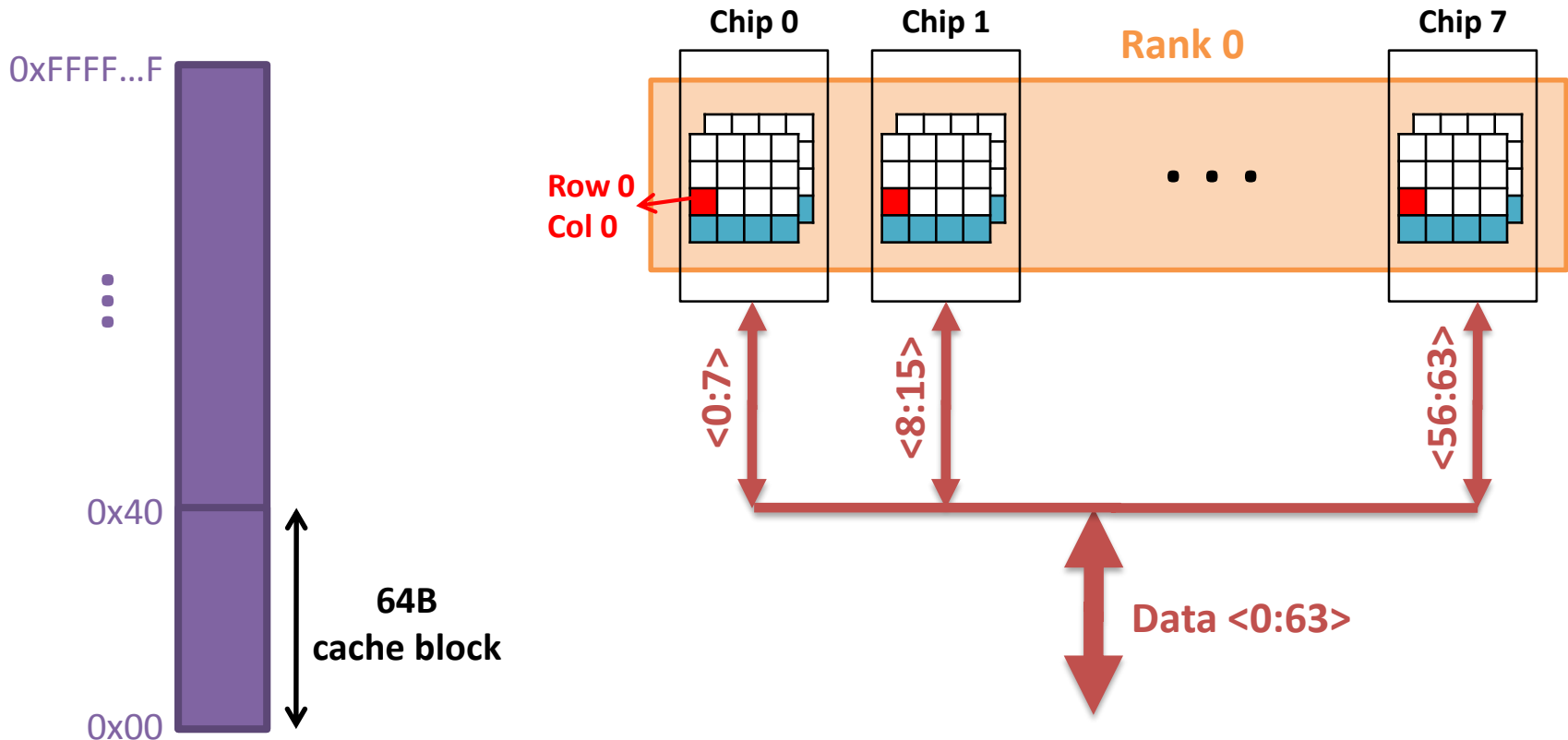
Example: Transferring a cache block

Physical memory space



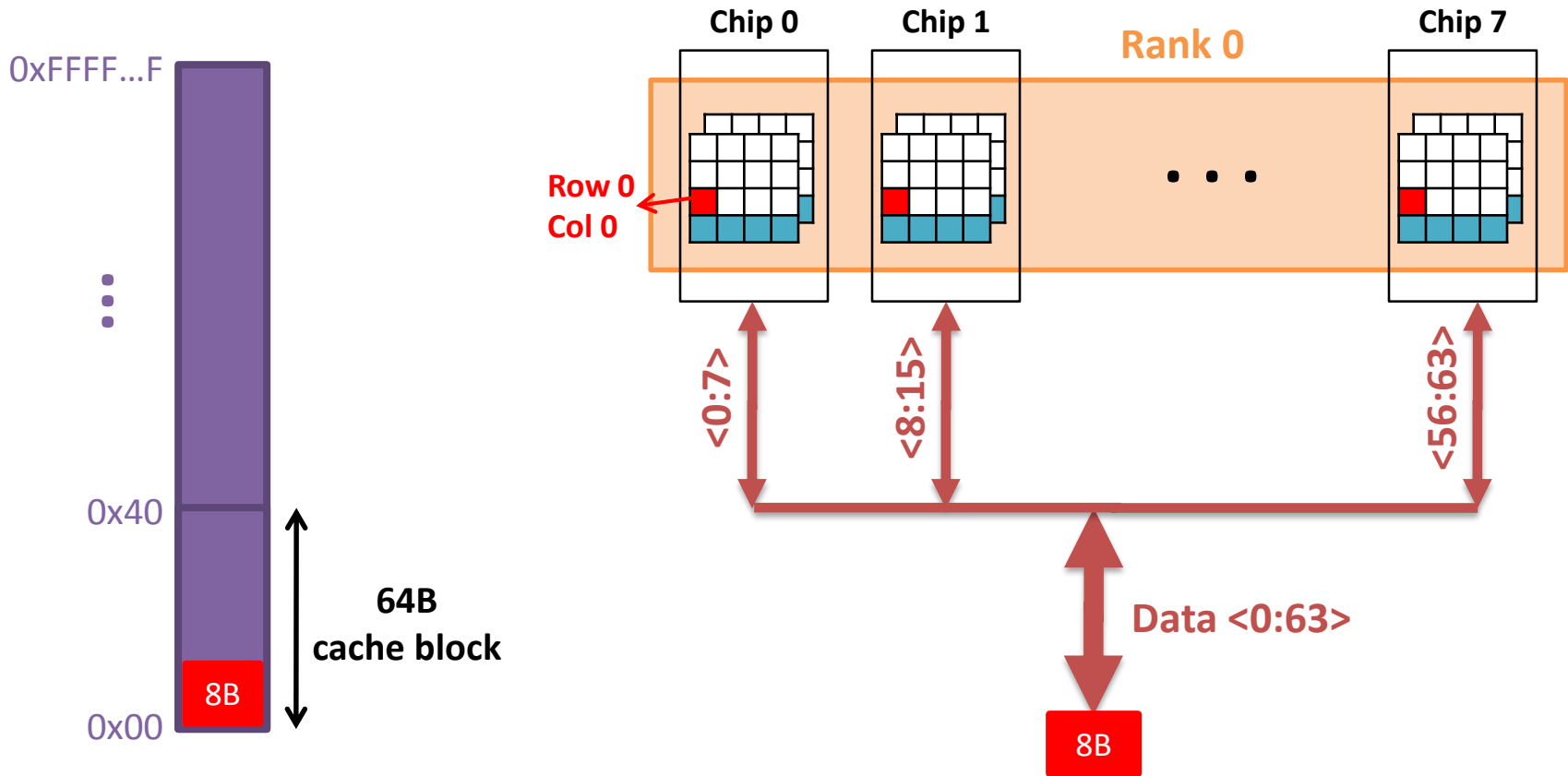
Example: Transferring a cache block

Physical memory space



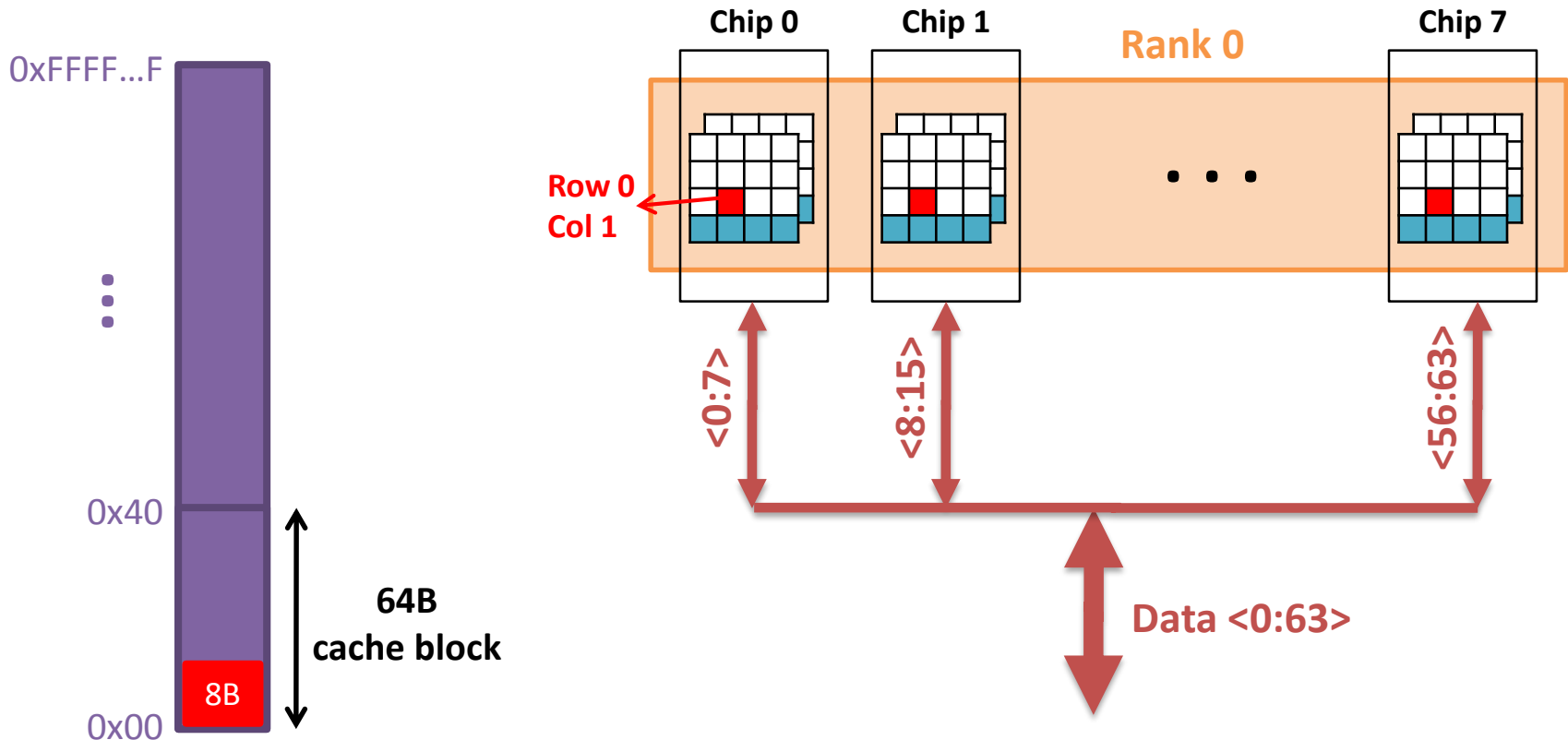
Example: Transferring a cache block

Physical memory space



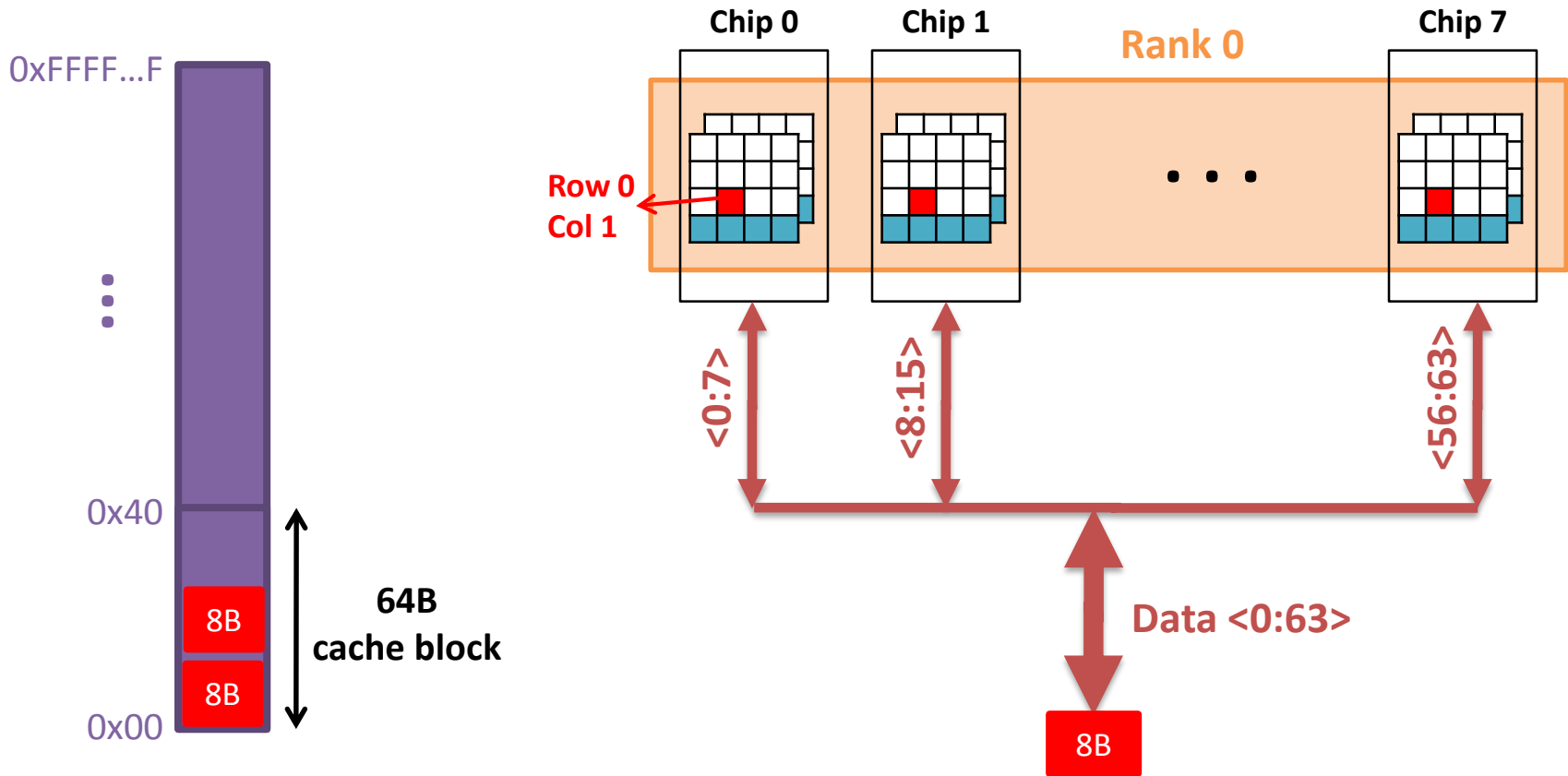
Example: Transferring a cache block

Physical memory space

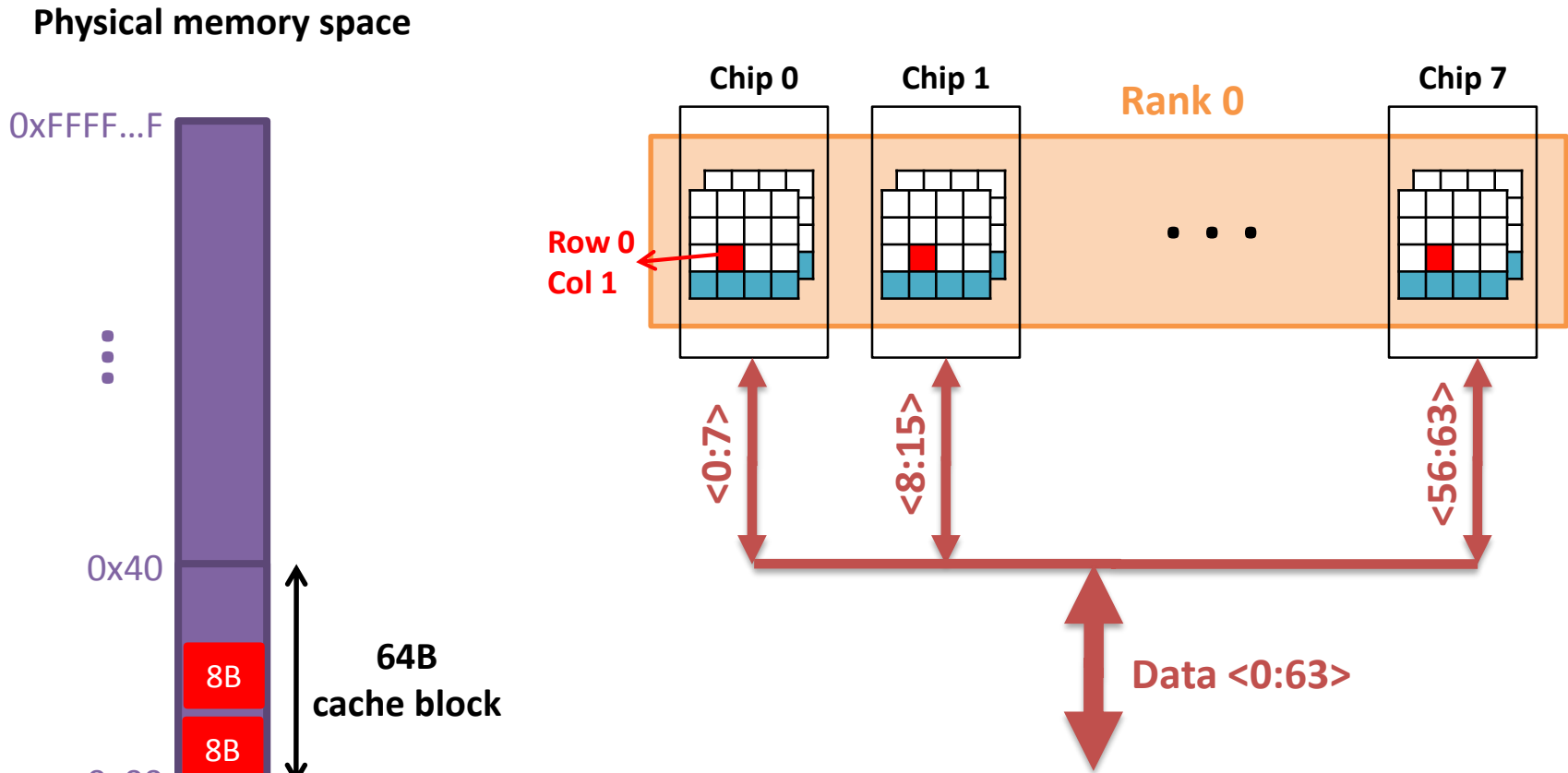


Example: Transferring a cache block

Physical memory space



Example: Transferring a cache block



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

During the process, 8 columns are read sequentially.

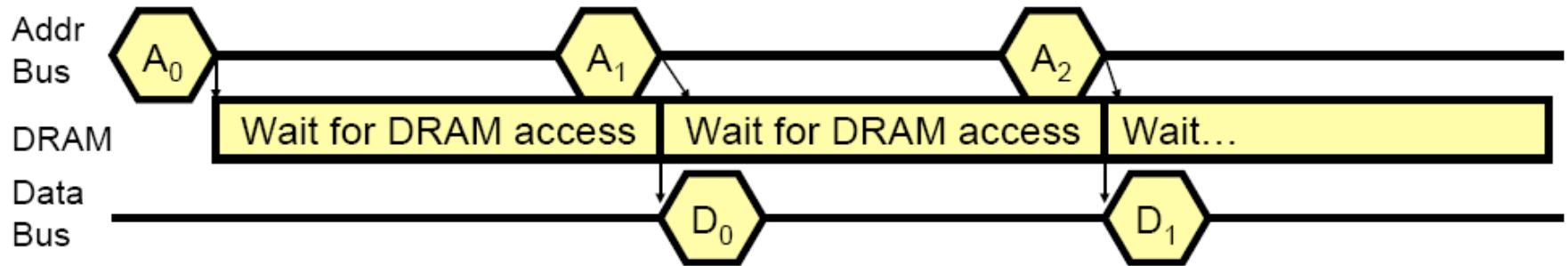
Latency Components: Basic DRAM Operation

- 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

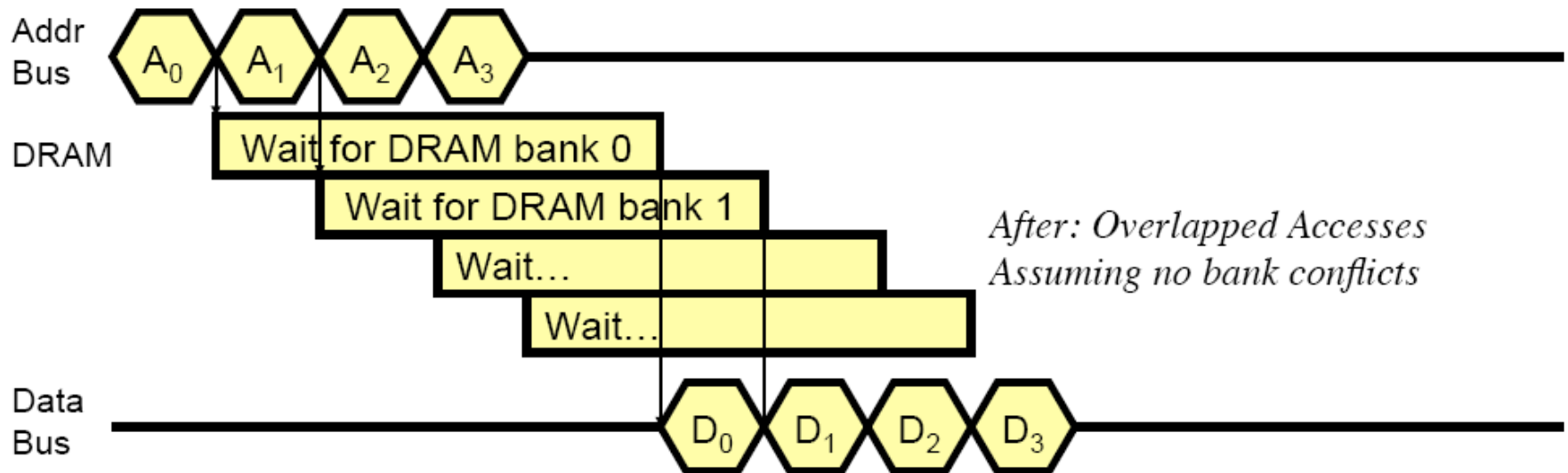
Multiple Banks (Interleaving) and Channels

- 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

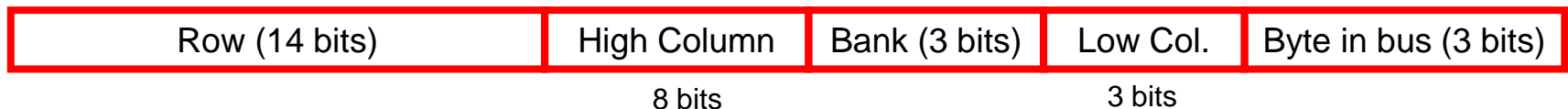
- Consecutive rows of memory in consecutive banks



- Accesses to consecutive cache blocks serviced in a pipelined manner

- Cache block interleaving

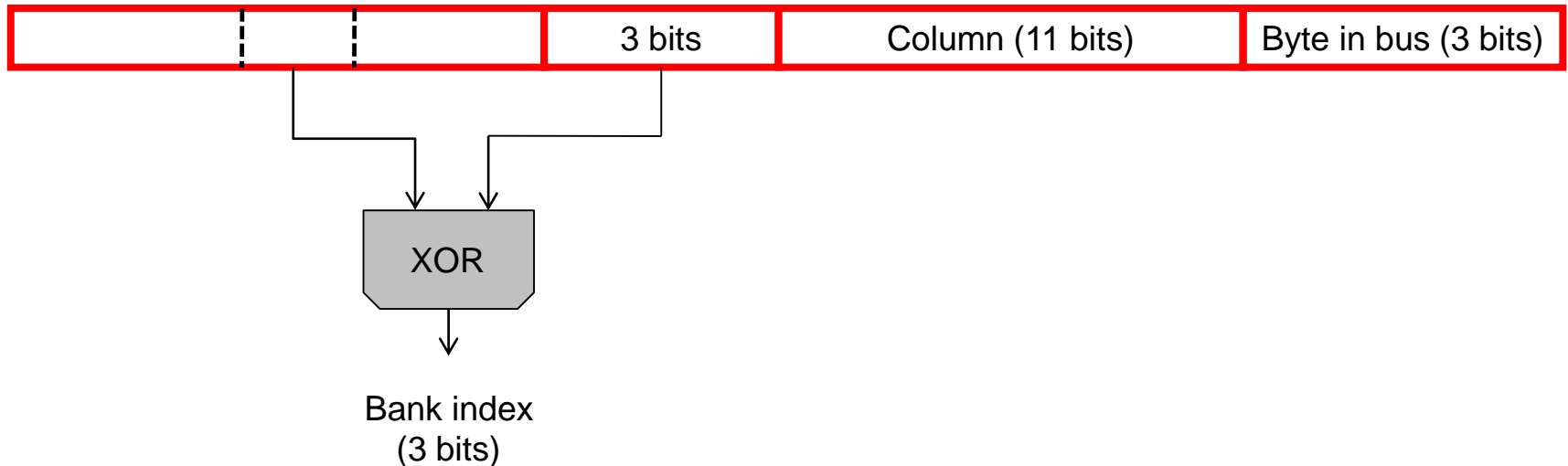
- Consecutive cache block addresses in consecutive banks
 - 64 byte cache blocks



- Accesses to consecutive cache blocks can be serviced in parallel

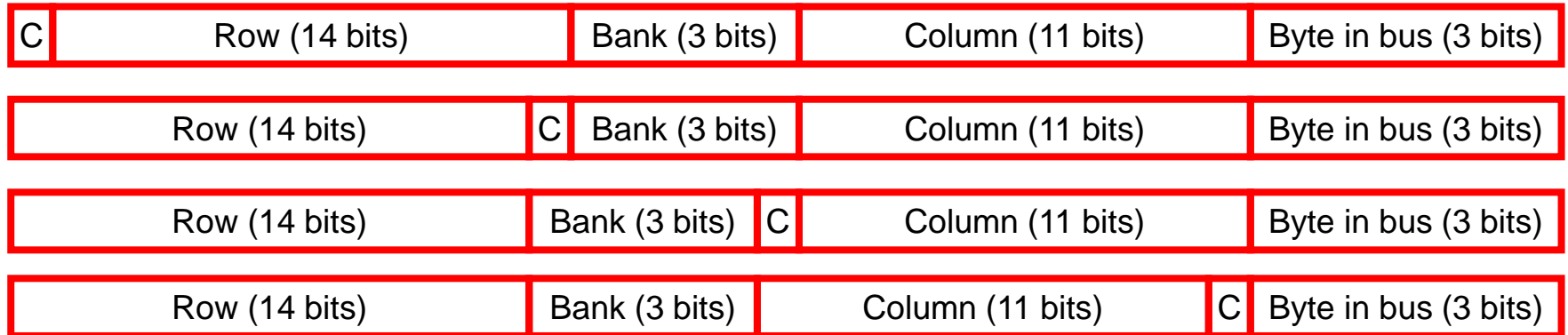
Bank Mapping Randomization

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

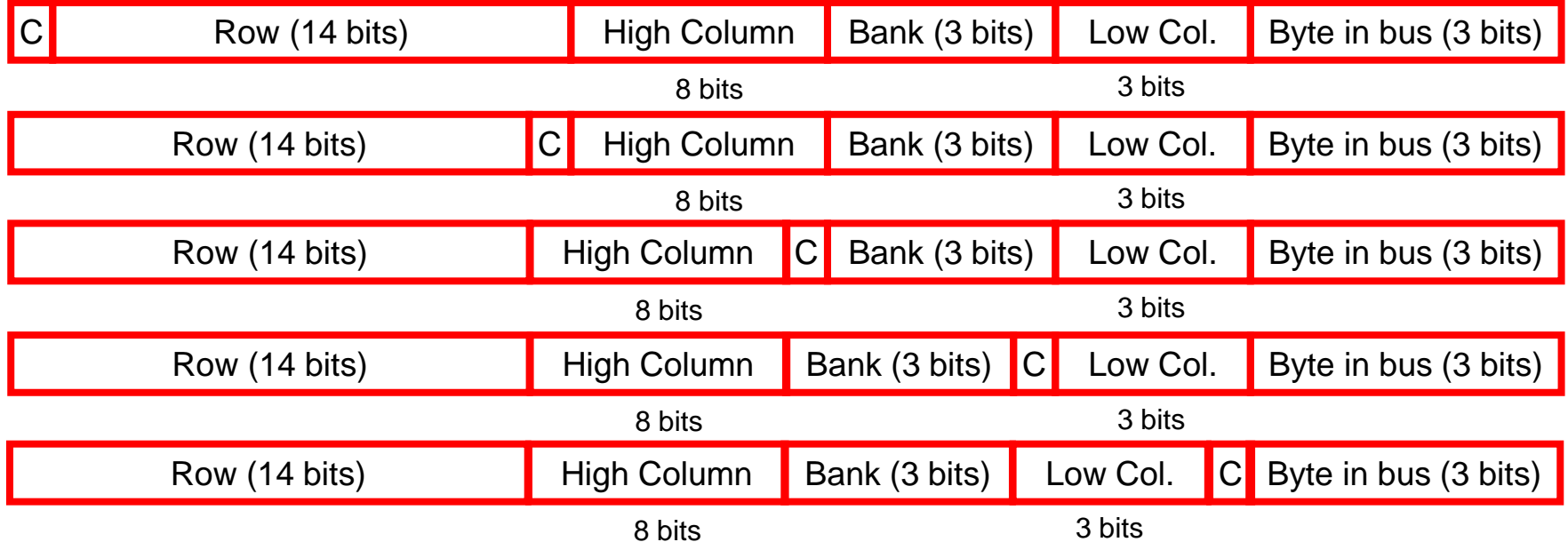


- Reading:
 - Rau, "Pseudo-randomly Interleaved Memory," ISCA 1991.

Address Mapping (Multiple Channels)

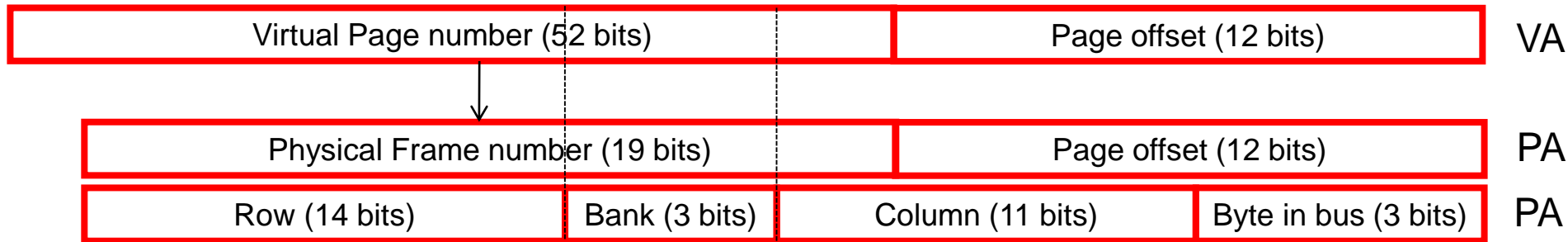


■ Where are consecutive cache blocks?



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**]

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.

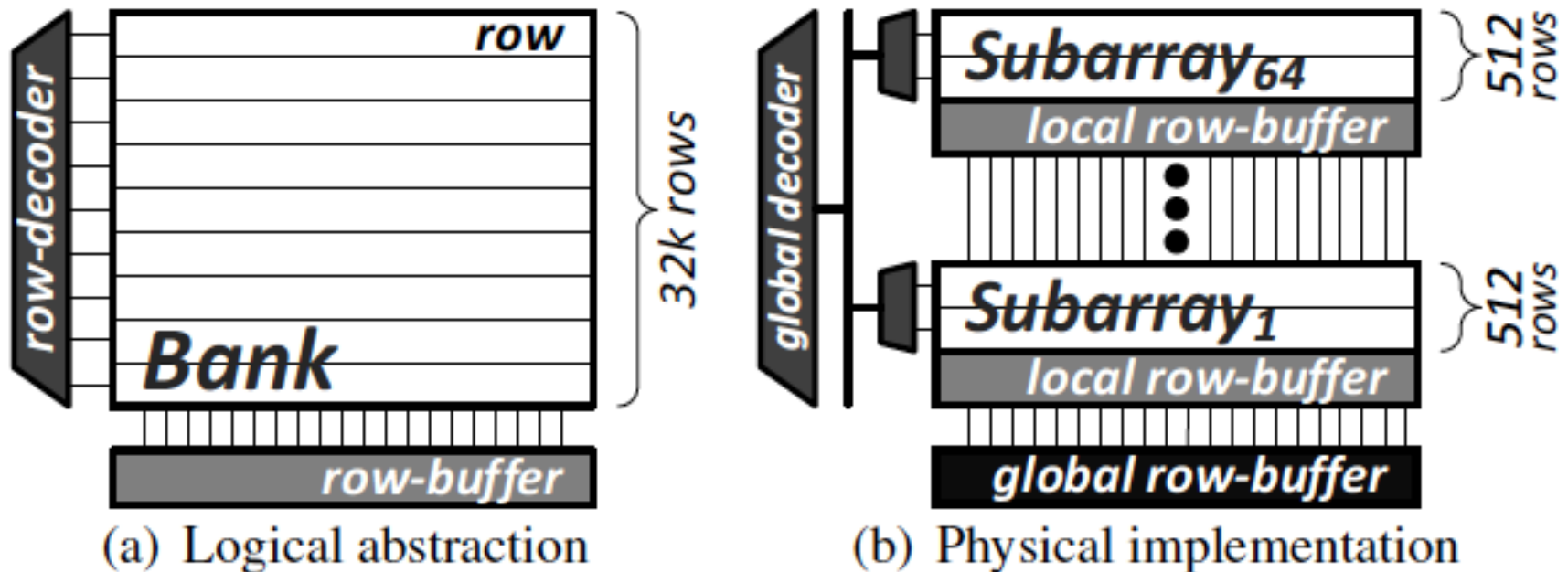
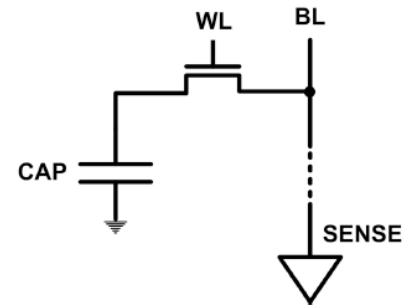


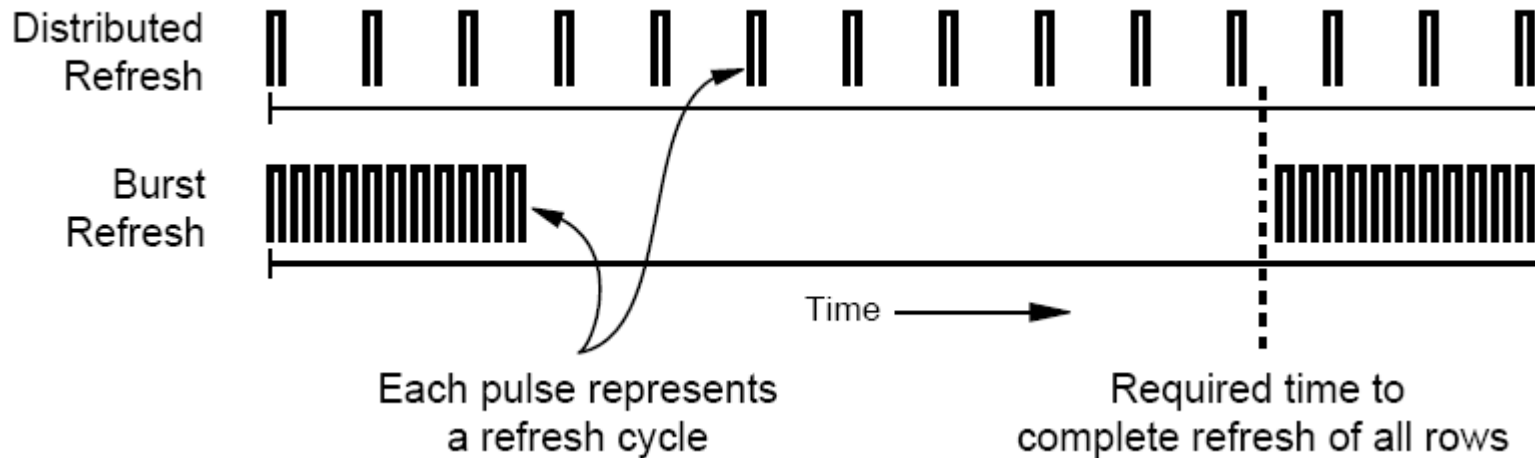
Figure 1. DRAM bank organization

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



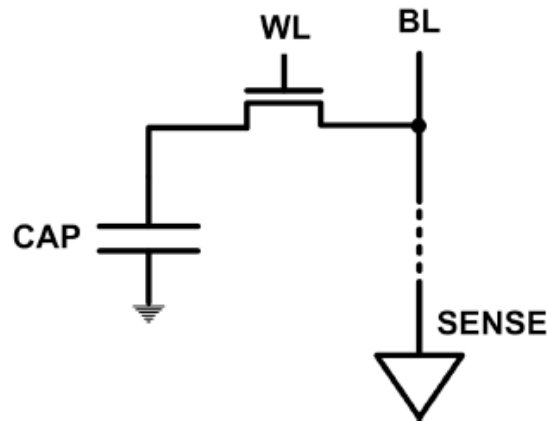
DRAM Refresh (II)



- 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.

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

DRAM Types

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

DRAM Types (II)

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

Table 1. Landscape of DRAM-based memory

Kim et al., “[Ramulator: A Fast and Extensible DRAM Simulator](#),” IEEE Comp Arch Letters 2015.

DRAM Controller: Functions

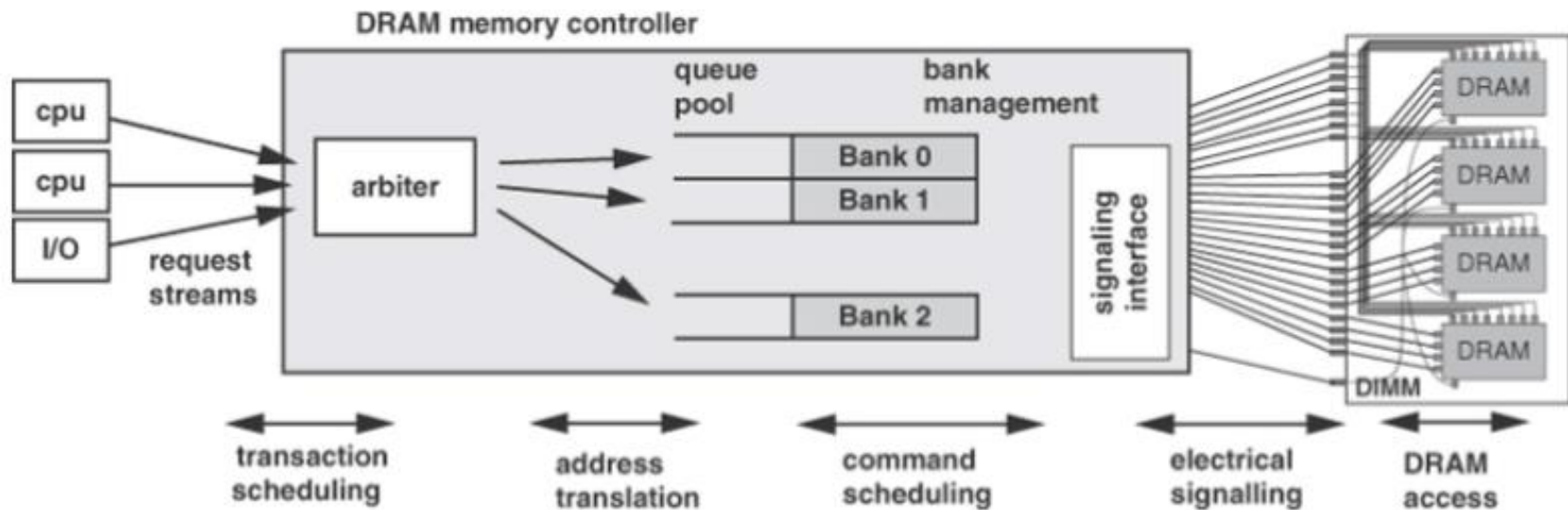
- Ensure correct operation of DRAM (refresh and timing)
- Service DRAM requests while obeying timing constraints of DRAM chips
 - Constraints: resource conflicts (bank, bus, channel), minimum write-to-read delays
 - Translate requests to DRAM command sequences
- Buffer and schedule requests to for high 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

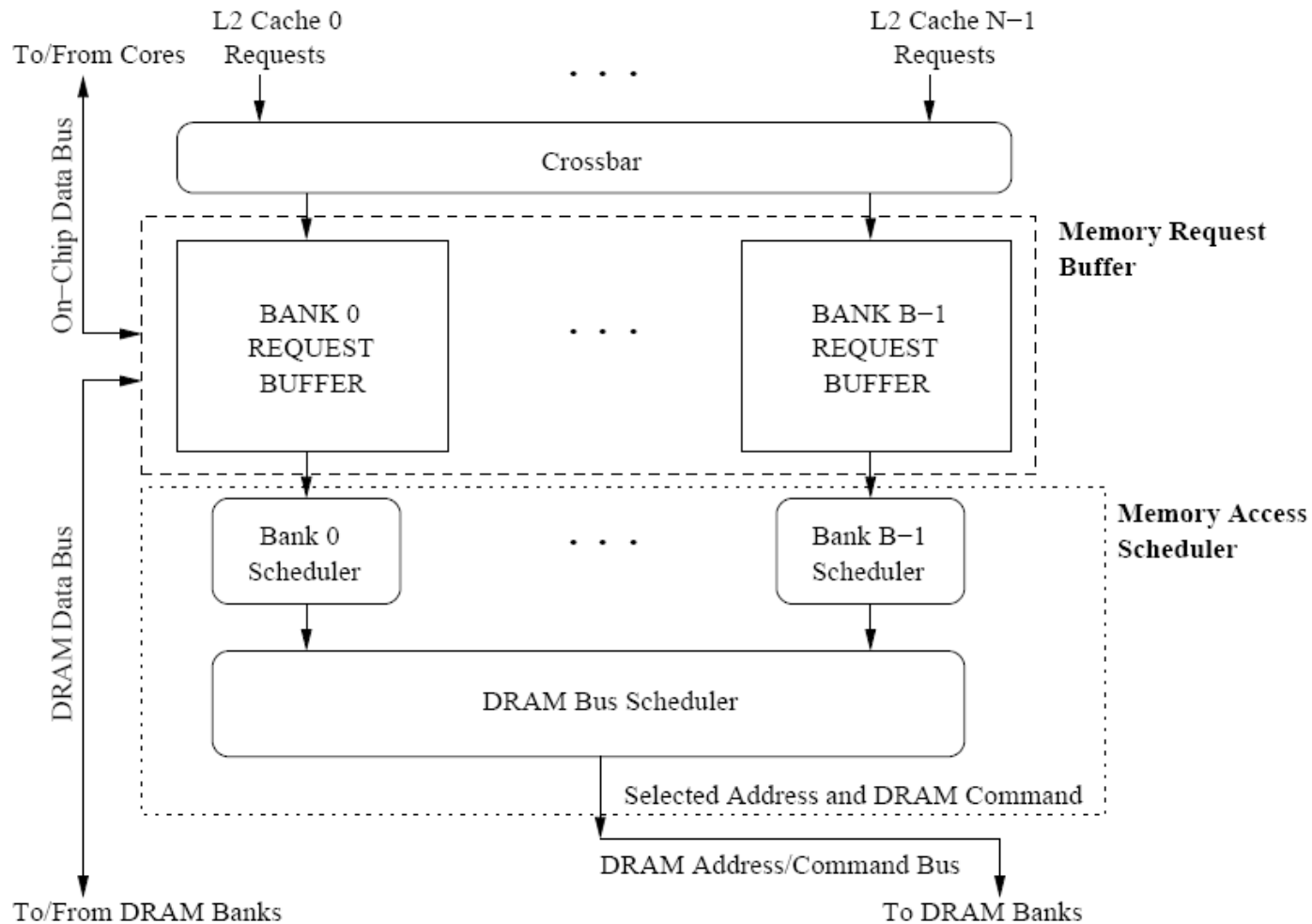
- In chipset
 - + More flexibility to plug different DRAM types into the system
 - + Less power density in the CPU chip

- 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

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.

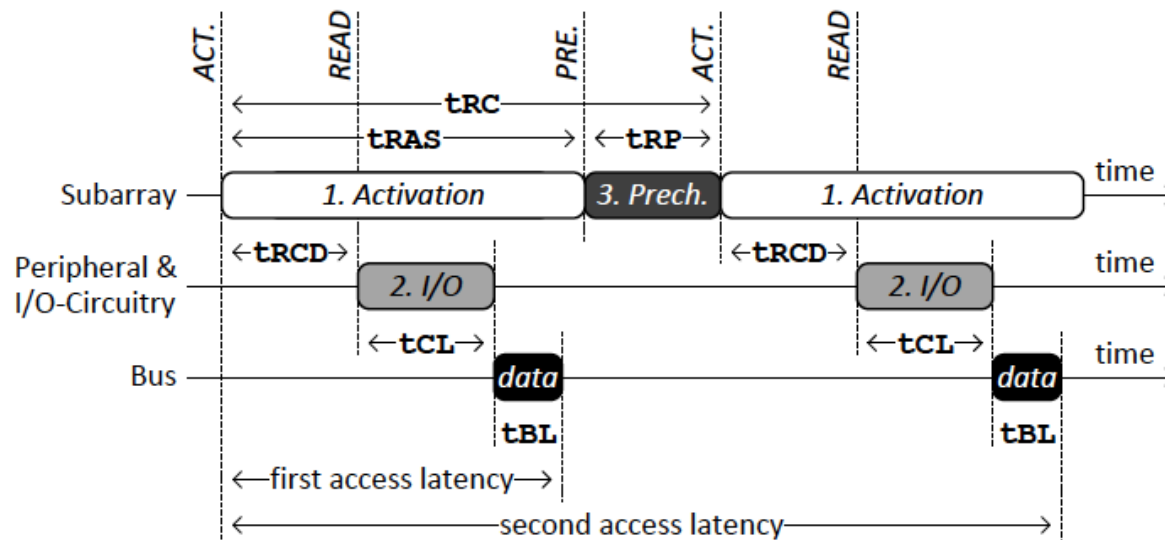
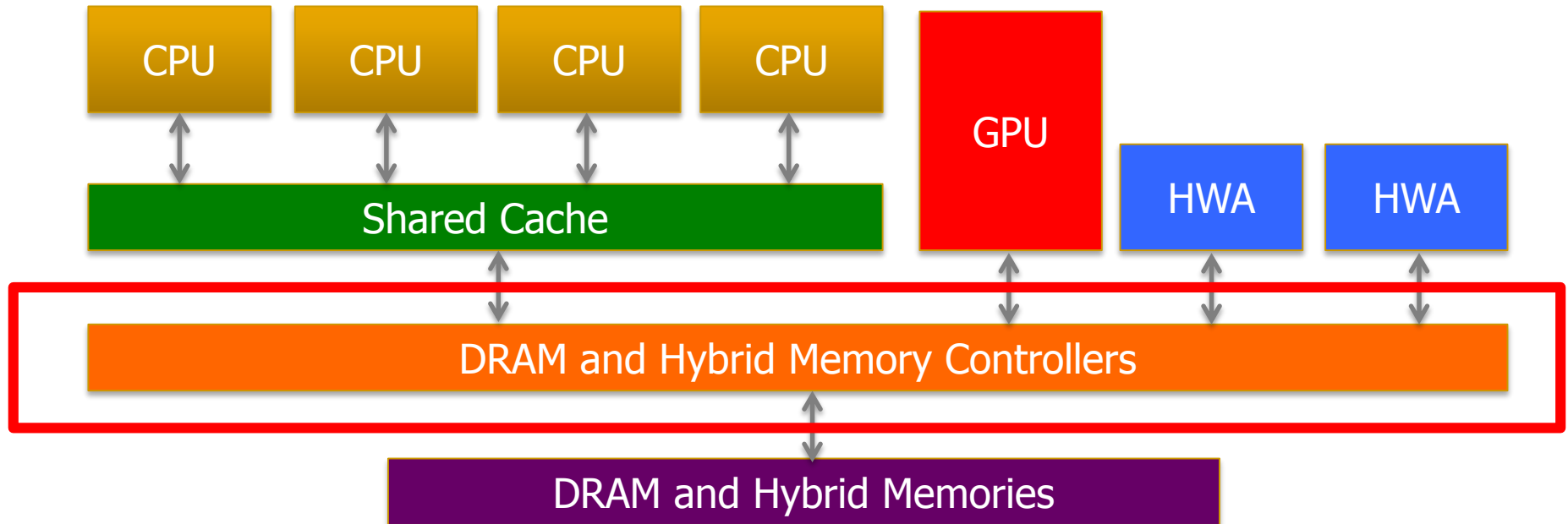


Figure 5. Three Phases of DRAM Access

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

Phase	Commands	Name	Value
1	ACT → READ ACT → WRITE	tRCD	15ns
	ACT → PRE	tRAS	37.5ns
	READ → data WRITE → data	tCL tCWL	15ns 11.25ns
2	data burst	tBL	7.5ns
	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.

Self-Optimizing DRAM Controllers



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

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

Next Lecture (9/30): Required Readings

➤ Required Reading Assignment:

- **Lee et al., "Phase Change Technology and the Future of Main Memory," IEEE Micro, Jan/Feb 2010.**

➤ Recommended References:

- M. Qureshi et al., "Scalable high performance main memory system using phase-change memory technology," ISCA 2009.
- H. Yoon et al., "Row buffer locality aware caching policies for hybrid memories," ICCD 2012.
- J. Zhao et al., "FIRM: Fair and High-Performance Memory Control for Persistent Memory Systems," MICRO 2014.

18-740/640

Computer Architecture

Lecture 8: Main Memory System

Prof. Onur Mutlu

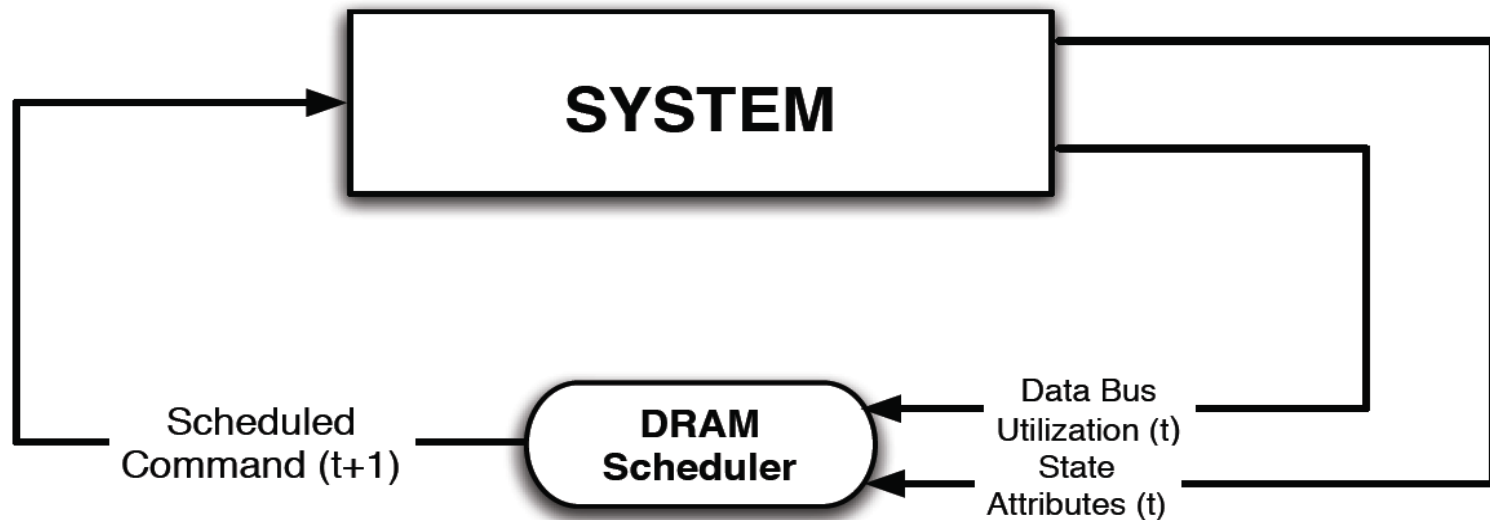
Carnegie Mellon University

Fall 2015, 9/28/2015

We did not cover the remaining slides in lecture. They are for your benefit.

Self-Optimizing DRAM Controllers

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



Self-Optimizing DRAM Controllers

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

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

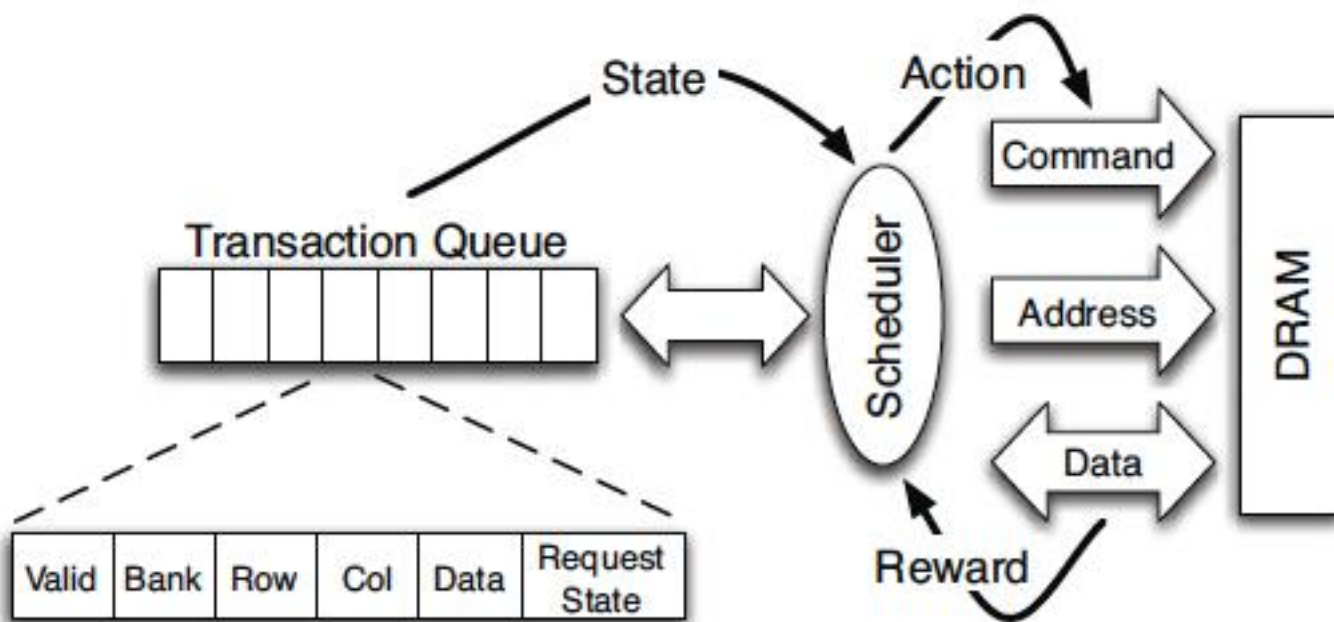


Figure 4: High-level overview of an RL-based scheduler.

States, Actions, Rewards

❖ Reward function

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

Goal is to maximize 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

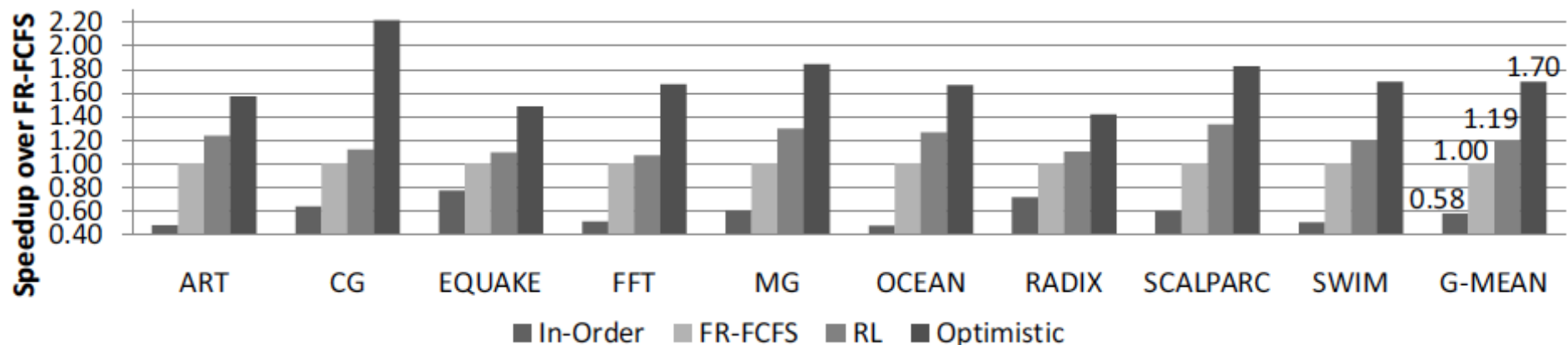


Figure 7: Performance comparison of in-order, FR-FCFS, RL-based, and optimistic memory controllers

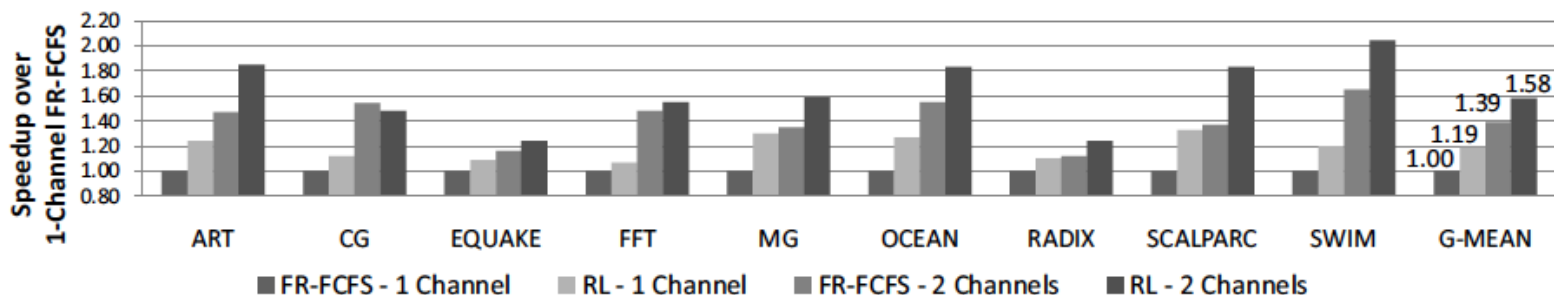


Figure 15: Performance comparison of FR-FCFS and RL-based memory controllers on systems with 6.4GB/s and 12.8GB/s peak DRAM bandwidth

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

¹Cornell University, Ithaca, NY 14850 USA

²Microsoft Research, Redmond, WA 98052 USA

Evaluating New Ideas for New (Memory) Architectures

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)
- 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

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)
 - Speed and flexibility reduce
 - 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

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

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

Table 1. Landscape of DRAM-based memory

Ramulator

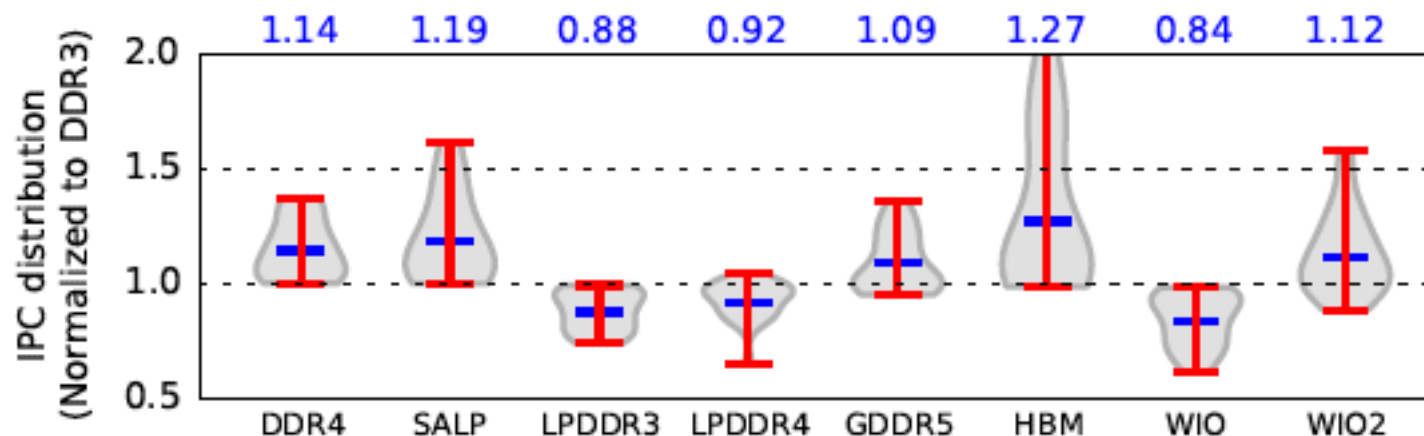
- Provides out-of-the box support for many DRAM standards:
 - DDR3/4, LPDDR3/4, GDDR5, WIO1/2, HBM, plus new proposals (SALP, AL-DRAM, TLDRAM, RowClone, and SARP)
- ~2.5X faster than fastest open-source simulator
- Modular and extensible to different standards

<i>Simulator</i> (clang -O3)	<i>Cycles (10⁶)</i>		<i>Runtime (sec.)</i>		<i>Req/sec (10³)</i>		<i>Memory</i> (MB)
	<i>Random</i>	<i>Stream</i>	<i>Random</i>	<i>Stream</i>	<i>Random</i>	<i>Stream</i>	
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

<i>Standard</i>	<i>Rate (MT/s)</i>	<i>Timing (CL-RCD-RP)</i>	<i>Data-Bus (Width×Chan.)</i>	<i>Rank-per-Chan</i>	<i>BW (GB/s)</i>
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



Across 22 workloads, simple CPU model

Figure 2. Performance comparison of DRAM standards

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 Kim¹ Weikun Yang^{1,2} Onur Mutlu¹
¹Carnegie Mellon University ²Peking University

Extra Credit Assignment

- Review the Ramulator paper
 - Send your reviews to me (omutlu@gmail.com)

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