Memory Systems in the Multi-Core Era Lecture 1: DRAM Basics and DRAM Scaling

Prof. Onur Mutlu
http://www.ece.cmu.edu/~omutlu
onur@cmu.edu

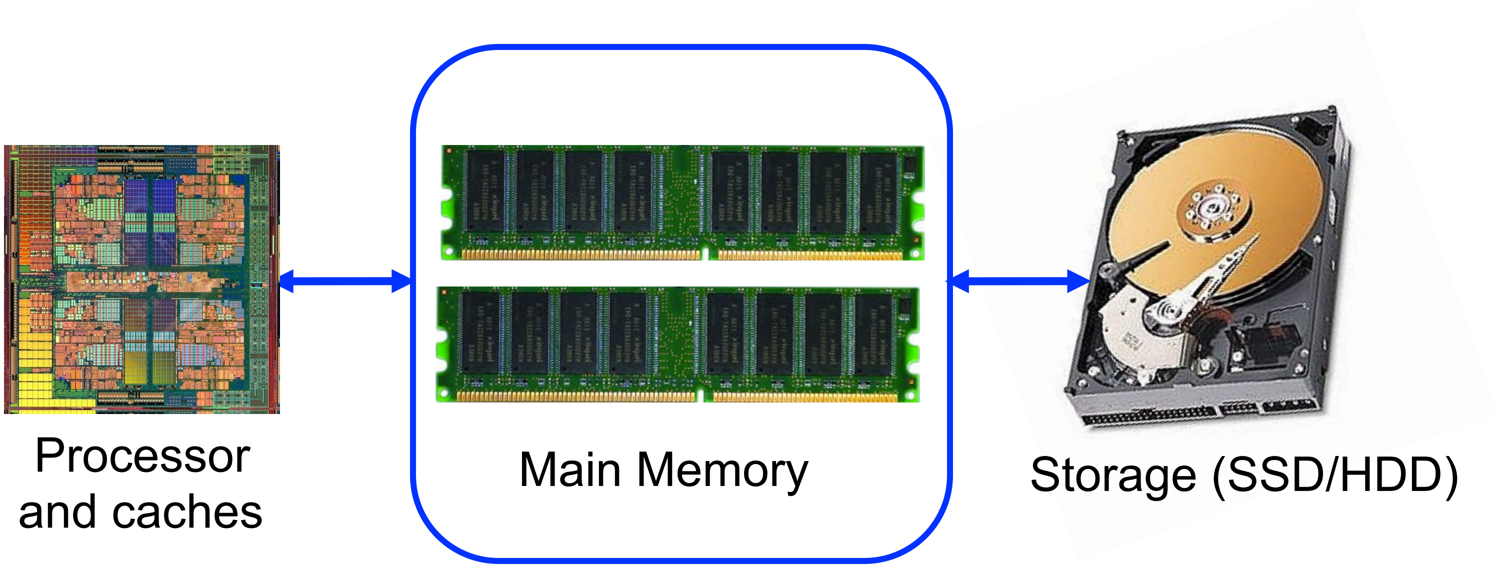
Bogazici University

June 13, 2013

Carnegie Mellon

SAFARI

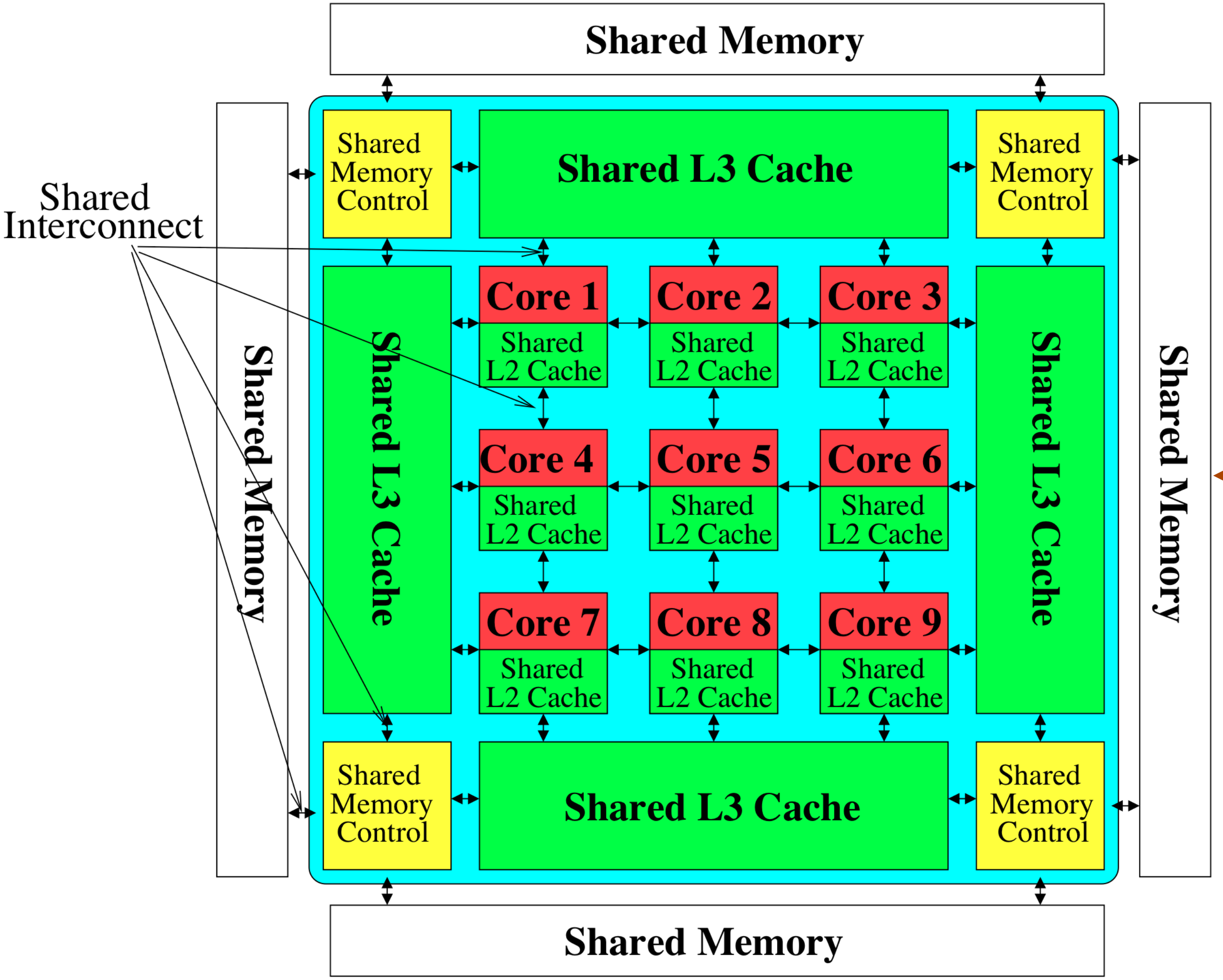
The Main Memory System

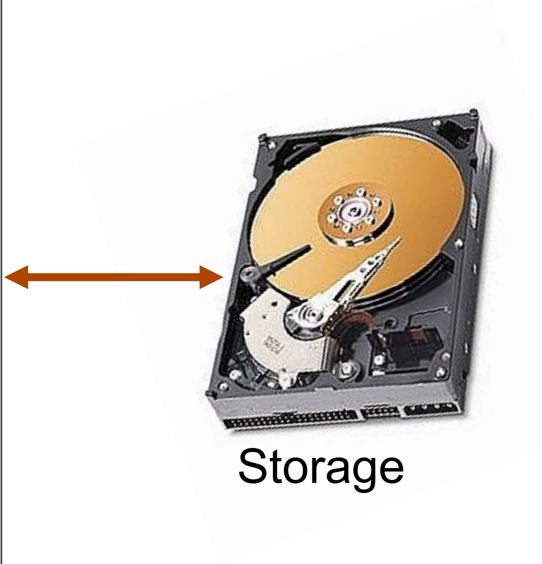


- Main memory is a critical component of all computing systems: server, mobile, embedded, desktop, sensor

- 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)

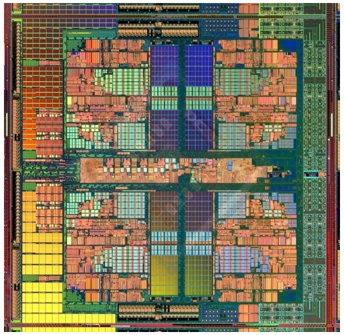
- Need for main memory capacity, bandwidth, QoS increasing
  - Multi-core: increasing number of cores
  - Data-intensive applications: increasing demand/hunger for data
  - Consolidation: cloud computing, GPUs, mobile

- Main memory energy/power is a key system design concern

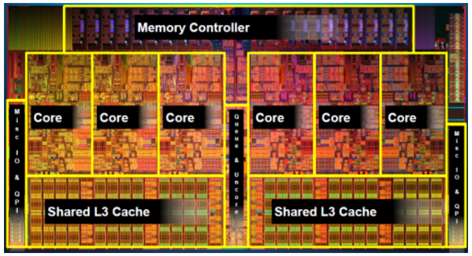
- DRAM technology scaling is ending

Example Trend: Many Cores on Chip

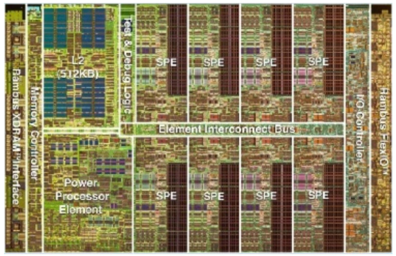
- Simpler and lower power than a single large core
- Large scale parallelism on chip



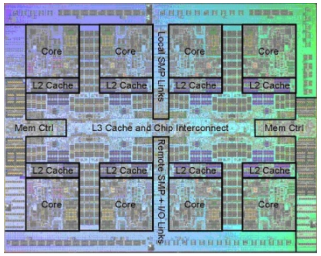
AMD Barcelona
4 cores



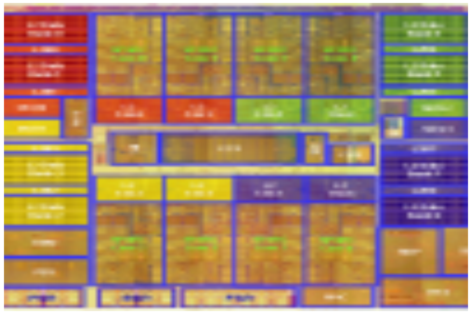
Intel Core i7
8 cores



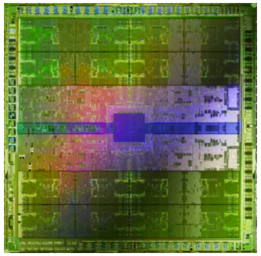
IBM Cell BE
8+1 cores



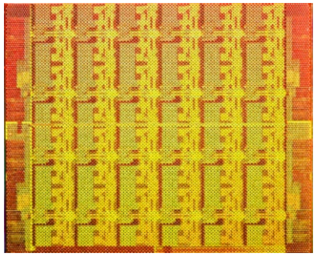
IBM POWER7
8 cores



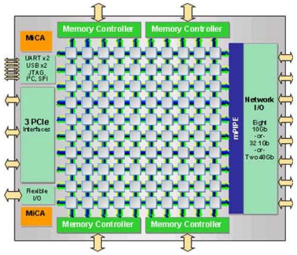
Sun Niagara II
8 cores



Nvidia Fermi
448 “cores”



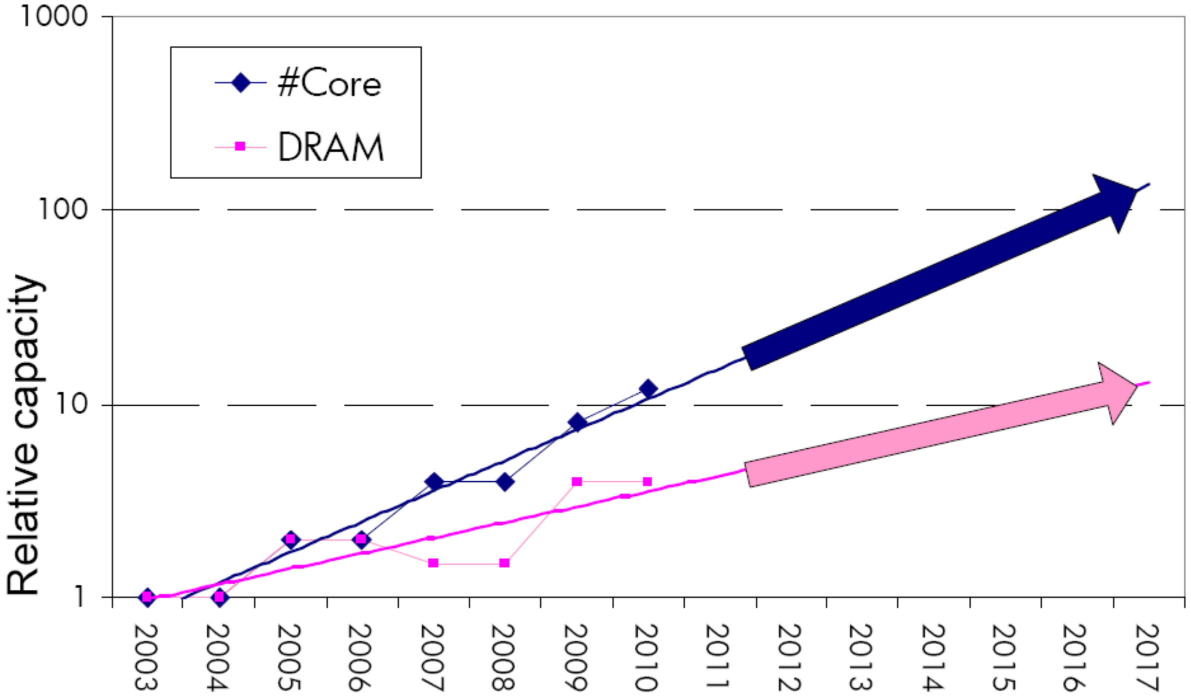
Intel SCC
48 cores, networked



Tilera TILE Gx
100 cores, networked

Consequence: The Memory Capacity Gap

Core count doubling ~ every 2 years
DRAM DIMM capacity doubling ~ every 3 years



Source: Lim et al., ISCA 2009.

- *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)

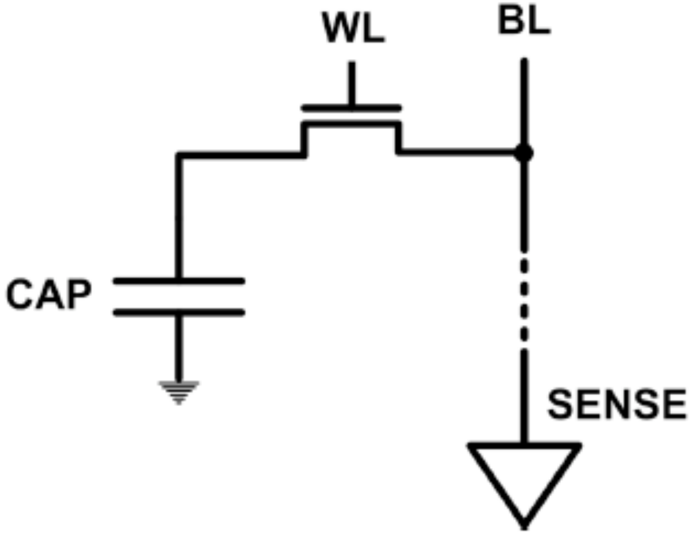
- Need for main memory capacity, bandwidth, QoS increasing

- 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

Solutions to the DRAM Scaling Problem

- Two potential solutions
  - Tolerate DRAM (by taking a fresh look at it)
  - Enable emerging memory technologies to eliminate/minimize DRAM

- Do both
  - Hybrid memory systems

Solution 1: Tolerate DRAM

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

- Key issues to tackle
  - Reduce refresh energy
  - Improve bandwidth and latency
  - Reduce waste
  - Enable reliability at low cost

- Liu, Jaiyen, Veras, Mutlu, "RAIDR: Retention-Aware Intelligent DRAM Refresh," ISCA 2012.
- Kim, Seshadri, Lee+, "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'13.

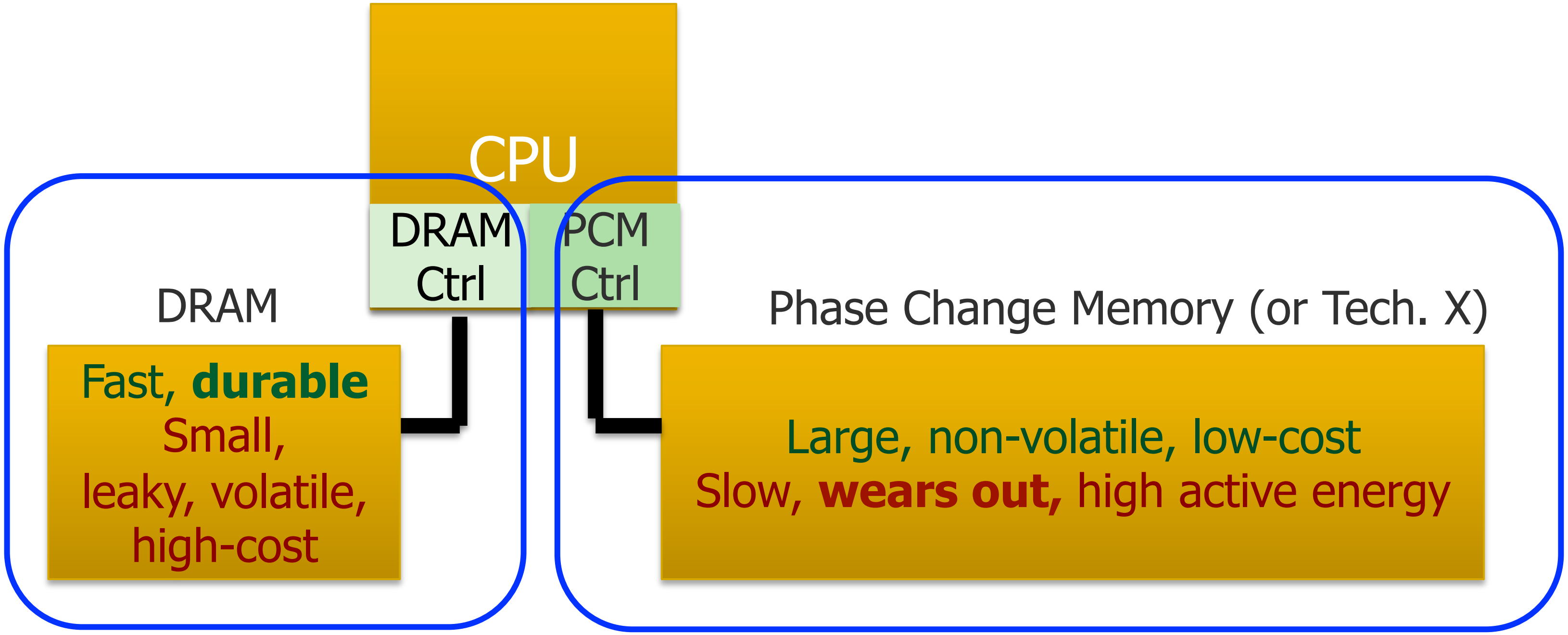
Solution 2: Emerging Memory Technologies

- 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, Ipek, Mutlu, Burger, "Architecting Phase Change Memory as a Scalable DRAM Alternative," ISCA 2009, CACM 2010, Top Picks 2010.
- Meza, Chang, Yoon, Mutlu, Ranganathan, "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.

Hybrid Memory Systems


CPU
DRAM Ctrl
PCM Ctrl
DRAM
Fast, durable
Small, leaky, volatile, high-cost
Phase Change Memory (or Tech. X)
Large, non-volatile, low-cost
Slow, wears out, high active energy

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.

An Orthogonal Issue: Memory Interference

- Problem: Memory interference is uncontrolled → uncontrollable, unpredictable, vulnerable system

- Goal: We need to control it → Design a QoS-aware system

- Solution: Hardware/software cooperative memory QoS
  - 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

  - E.g., fair, programmable memory controllers and on-chip networks provide QoS and predictable performance
    [2007-2012, Top Picks'09,'11a,'11b,'12]

Agenda for Today

- What Will You Learn in This Mini-Lecture Series
- Main Memory Basics (with a Focus on DRAM)
- Major Trends Affecting Main Memory
- DRAM Scaling Problem and Solution Directions
- Solution Direction 1: System-DRAM Co-Design
- Ongoing Research
- Summary

What Will You Learn in Mini Course 2?

- Memory Systems in the Multi-Core Era
  - June 13, 14, 17 (1-4pm)

- Lecture 1: Main memory basics, DRAM scaling
- Lecture 2: Emerging memory technologies and hybrid memories
- Lecture 3: Main memory interference and QoS

- Major Overview Reading:
  - Mutlu, "Memory Scaling: A Systems Architecture Perspective," IMW 2013.

Attendance Sheet

- If you are not on the email list, please sign the attendance sheet with your name and email address.

This Course

- Will cover many problems and potential solutions related to the design of memory systems in the many core era

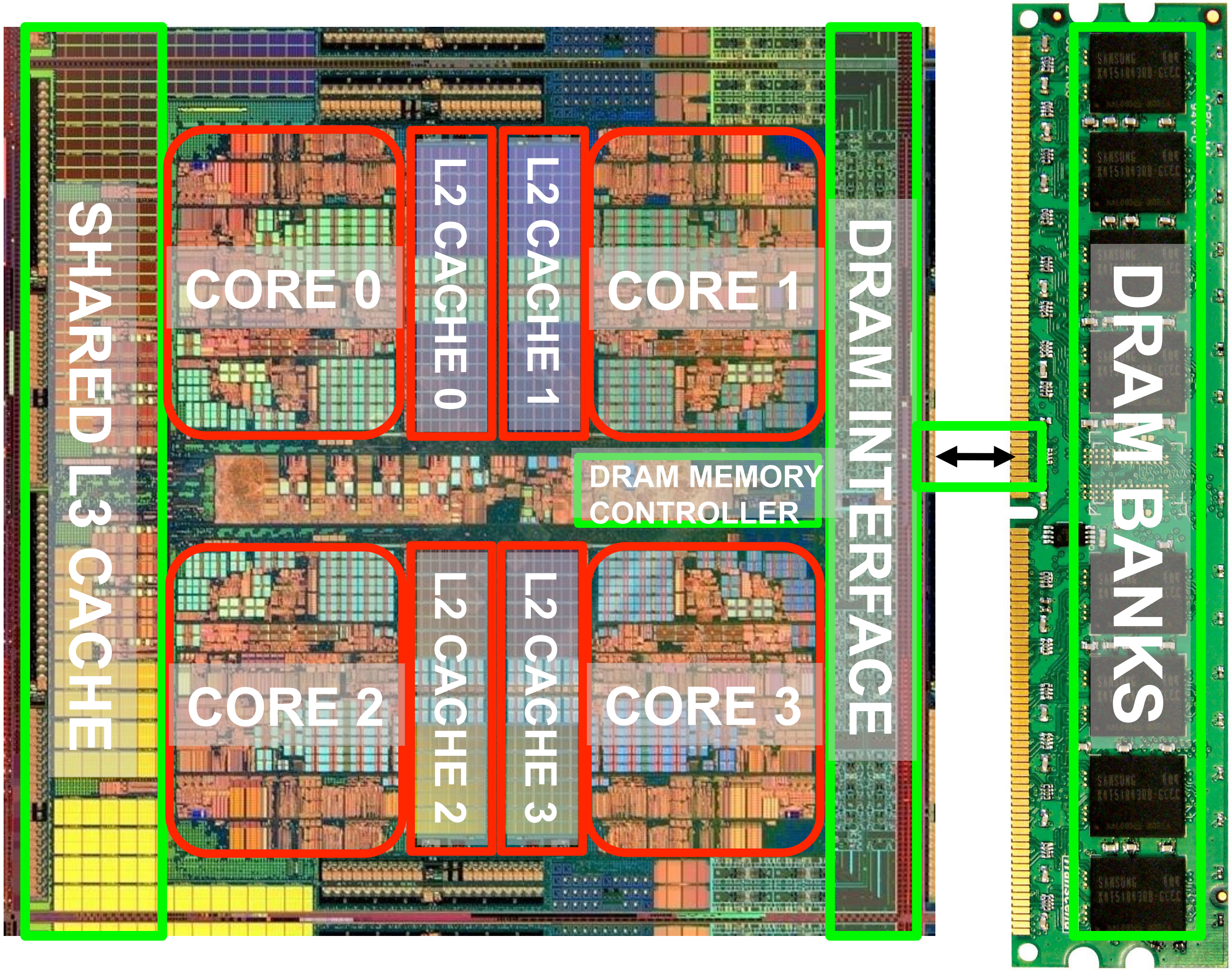
- The design of the memory system poses many
    - Difficult research and engineering problems
    - Important fundamental problems
    - Industry-relevant problems

- Many creative and insightful solutions are needed to solve these problems

- Goal: Acquire the basics to develop such solutions (by covering fundamentals and cutting edge research)

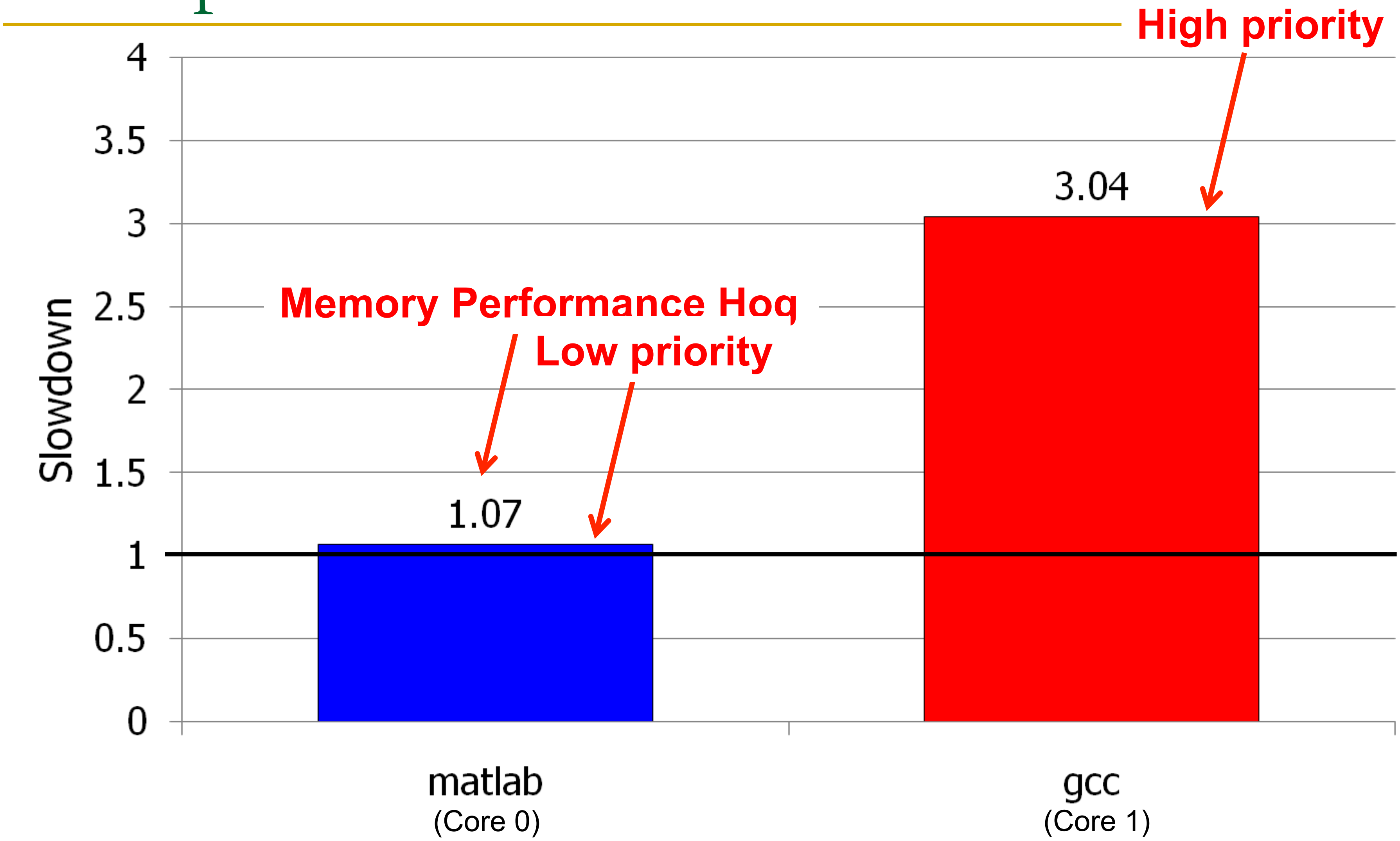
An Example Problem: Shared Main Memory

Multi-Core Chip



*Die photo credit: AMD Barcelona

Unexpected Slowdowns in Multi-Core



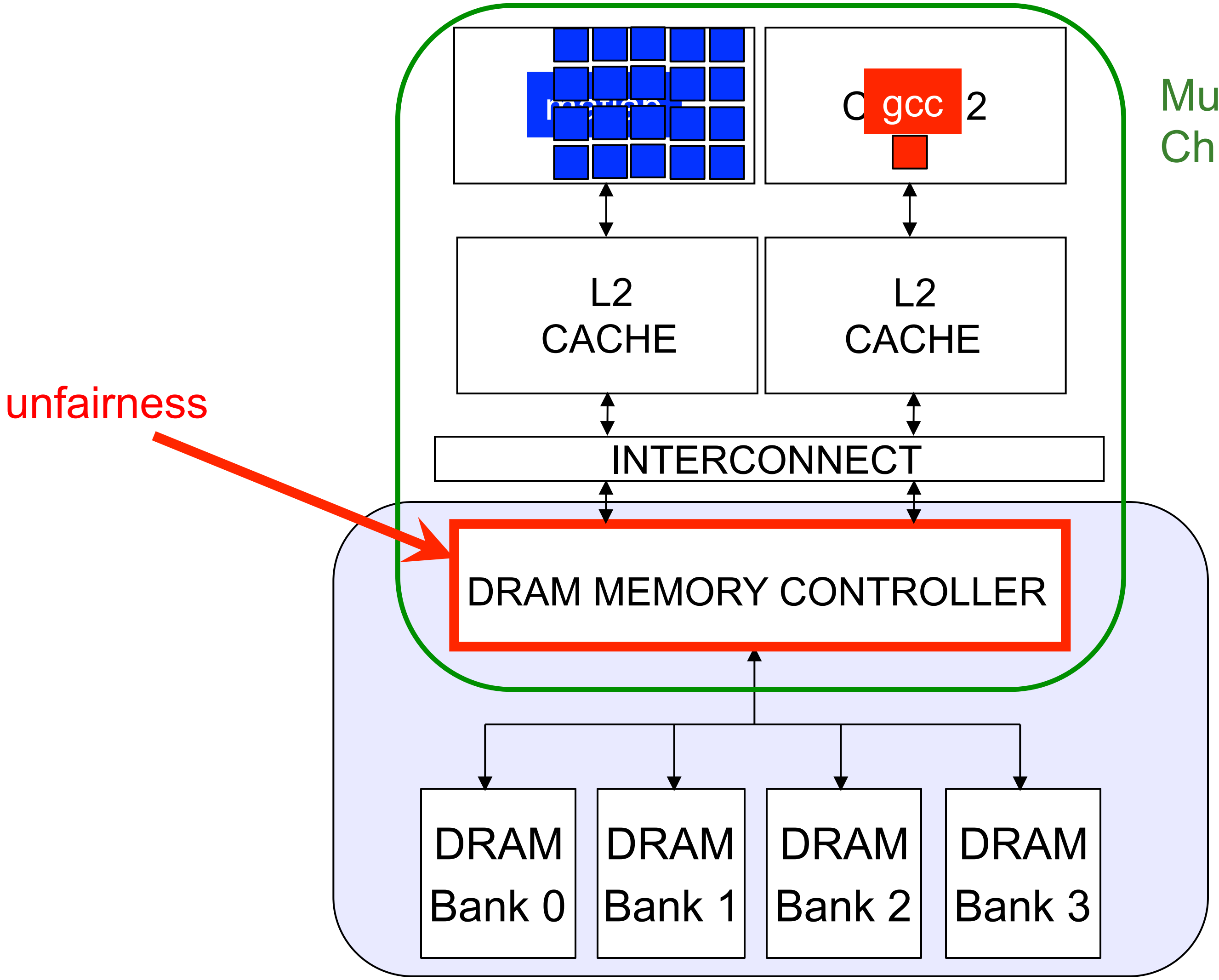
Moscibroda and Mutlu, "Memory performance attacks: Denial of memory service in multi-core systems," USENIX Security 2007.

A Question or Two

- Can you figure out why there is a disparity in slowdowns if you do not know how the processor executes the programs?

- Can you fix the problem without knowing what is happening “underneath”?

Why the Disparity in Slowdowns?



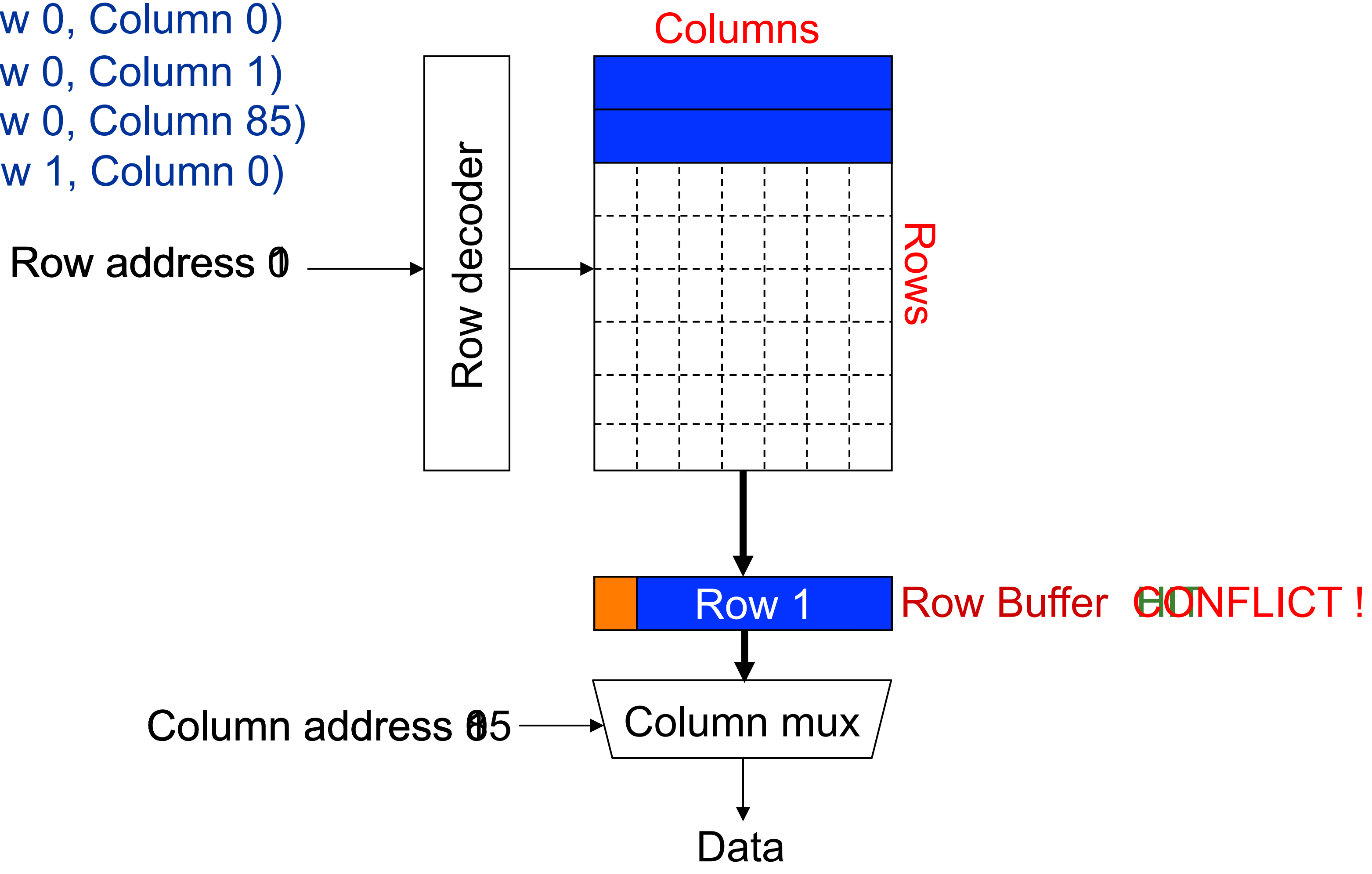
unfairness

Multi-Core Chip

Shared DRAM Memory System

DRAM Bank Operation

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



DRAM Controllers

- A row-conflict memory access takes significantly longer than a row-hit access

- Current controllers take advantage of the row buffer

- Commonly used scheduling policy (FR-FCFS) [Rixner 2000]*

  (1) Row-hit first: Service row-hit memory accesses first

  (2) Oldest-first: Then service older accesses first

- This scheduling policy aims to maximize DRAM throughput

*Rixner et al., “Memory Access Scheduling,” ISCA 2000.
*Zuravleff and Robinson, “Controller for a synchronous DRAM …,” US Patent 5,630,096, May 1997.

The Problem

- Multiple threads share the DRAM controller
- DRAM controllers designed to maximize DRAM throughput

- DRAM scheduling policies are thread-unfair
  - Row-hit first: unfairly prioritizes threads with high row buffer locality
    - Threads that keep on accessing the same row
  - Oldest-first: unfairly prioritizes memory-intensive threads

- DRAM controller vulnerable to denial of service attacks
  - Can write programs to exploit unfairness

Now That We Know What Happens Underneath

- How would you solve the problem?

Some Solution Examples (To Be Covered)

- We will cover some solutions later in this accelerated course
- Example recent solutions (part of your reading list)
  - Yoongu Kim, Michael Papamichael, Onur Mutlu, and Mor Harchol-Balter,
    "Thread Cluster Memory Scheduling: Exploiting Differences in Memory Access Behavior"
    Proceedings of the 43rd International Symposium on Microarchitecture (***MICRO***), pages 65-76, Atlanta, GA, December 2010.

  - 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)

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

Readings and Videos

Overview Reading

- Mutlu, "Memory Scaling: A Systems Architecture Perspective," IMW 2013.

- Onur Mutlu,
  "Memory Scaling: A Systems Architecture Perspective"
  *Proceedings of the 5th International Memory Workshop (**IMW**)*, Monterey, CA, May 2013. Slides (pptx) (pdf)

Memory Lecture Videos

- Memory Hierarchy (and Introduction to Caches)
  - http://www.youtube.com/watch?v=JBdfZ5i21cs&list=PL5PHm2jkkXmidJOd59REog9jDnPDTG6IJ&index=22
- Main Memory
  - http://www.youtube.com/watch?v=ZLCy3pG7Rc0&list=PL5PHm2jkkXmidJOd59REog9jDnPDTG6IJ&index=25
- Memory Controllers, Memory Scheduling, Memory QoS
  - http://www.youtube.com/watch?v=ZSotvL3WXmA&list=PL5PHm2jkkXmidJOd59REog9jDnPDTG6IJ&index=26
  - http://www.youtube.com/watch?v=1xe2w3_NzmI&list=PL5PHm2jkkXmidJOd59REog9jDnPDTG6IJ&index=27
- Emerging Memory Technologies
  - http://www.youtube.com/watch?v=LzfOghMKyA0&list=PL5PHm2jkkXmidJOd59REog9jDnPDTG6IJ&index=35
- Multiprocessor Correctness and Cache Coherence
  - http://www.youtube.com/watch?v=U-VZKMgItDM&list=PL5PHm2jkkXmidJOd59REog9jDnPDTG6IJ&index=32

Readings for Lecture 2.1 (DRAM Scaling)

- Lee et al., "Tiered-Latency DRAM: A Low Latency and Low Cost DRAM Architecture," HPCA 2013.
- Liu et al., "RAIDR: Retention-Aware Intelligent DRAM Refresh," ISCA 2012.
- Kim et al., "A Case for Exploiting Subarray-Level Parallelism in DRAM," ISCA 2012.
- Liu et al., "An Experimental Study of Data Retention Behavior in Modern DRAM Devices," ISCA 2013.
- Seshadri et al., "RowClone: Fast and Efficient In-DRAM Copy and Initialization of Bulk Data," CMU CS Tech Report 2013.
- David et al., "Memory Power Management via Dynamic Voltage/ Frequency Scaling," ICAC 2011.
- Ipek et al., "Self Optimizing Memory Controllers: A Reinforcement Learning Approach," ISCA 2008.

Readings for Lecture 2.2 (Emerging Technologies)

- Lee, Ipek, Mutlu, Burger, “Architecting Phase Change Memory as a Scalable DRAM Alternative,” ISCA 2009, CACM 2010, Top Picks 2010.
- Qureshi et al., “Scalable high performance main memory system using phase-change memory technology,” ISCA 2009.
- Meza et al., “Enabling Efficient and Scalable Hybrid Memories,” IEEE Comp. Arch. Letters 2012.
- Yoon et al., “Row Buffer Locality Aware Caching Policies for Hybrid Memories,” ICCD 2012 Best Paper Award.
- Meza et al., “A Case for Efficient Hardware-Software Cooperative Management of Storage and Memory,” WEED 2013.
- Kultursay et al., “Evaluating STT-RAM as an Energy-Efficient Main Memory Alternative,” ISPASS 2013.

- More to come in next lecture…

Readings for Lecture 2.3 (Memory QoS)

- Moscibroda and Mutlu, "Memory Performance Attacks," USENIX Security 2007.
- Mutlu and Moscibroda, "Stall-Time Fair Memory Access Scheduling," MICRO 2007.
- Mutlu and Moscibroda, "Parallelism-Aware Batch Scheduling," ISCA 2008, IEEE Micro 2009.
- Kim et al., "ATLAS: A Scalable and High-Performance Scheduling Algorithm for Multiple Memory Controllers," HPCA 2010.
- Kim et al., "Thread Cluster Memory Scheduling," MICRO 2010, IEEE Micro 2011.
- Muralidhara et al., "Memory Channel Partitioning," MICRO 2011.
- Ausavarungnirun et al., "Staged Memory Scheduling," ISCA 2012.
- Subramanian et al., "MISE: Providing Performance Predictability and Improving Fairness in Shared Main Memory Systems," HPCA 2013.
- Das et al., "Application-to-Core Mapping Policies to Reduce Memory System Interference in Multi-Core Systems," HPCA 2013.

Readings for Lecture 2.3 (Memory QoS)

- Ebrahimi et al., "Fairness via Source Throttling," ASPLOS 2010, ACM TOCS 2012.
- Lee et al., "Prefetch-Aware DRAM Controllers," MICRO 2008, IEEE TC 2011.
- Ebrahimi et al., "Parallel Application Memory Scheduling," MICRO 2011.
- Ebrahimi et al., "Prefetch-Aware Shared Resource Management for Multi-Core Systems," ISCA 2011.

- More to come in next lecture…

Readings in Flash Memory

- Yu Cai, Gulay Yalcin, Onur Mutlu, Erich F. Haratsch, Adrian Cristal, Osman Unsal, and Ken Mai,
  "Error Analysis and Retention-Aware Error Management for NAND Flash Memory"
  *Intel Technology Journal (**ITJ**) Special Issue on Memory Resiliency*, Vol. 17, No. 1, May 2013.

- Yu Cai, Erich F. Haratsch, Onur Mutlu, and Ken Mai,
  "Threshold Voltage Distribution in MLC NAND Flash Memory: Characterization, Analysis and Modeling"
  *Proceedings of the Design, Automation, and Test in Europe Conference (**DATE**)*, Grenoble, France, March 2013. Slides (ppt)

- Yu Cai, Gulay Yalcin, Onur Mutlu, Erich F. Haratsch, Adrian Cristal, Osman Unsal, and Ken Mai,
  "Flash Correct-and-Refresh: Retention-Aware Error Management for Increased Flash Memory Lifetime"
  *Proceedings of the 30th IEEE International Conference on Computer Design (**ICCD**)*, Montreal, Quebec, Canada, September 2012. Slides (ppt) (pdf)

- Yu Cai, Erich F. Haratsch, Onur Mutlu, and Ken Mai,
  "Error Patterns in MLC NAND Flash Memory: Measurement, Characterization, and Analysis"
  *Proceedings of the Design, Automation, and Test in Europe Conference (**DATE**)*, Dresden, Germany, March 2012. Slides (ppt)

Online Lectures and More Information

- Online Computer Architecture Lectures
  - http://www.youtube.com/playlist?list=PL5PHm2jkkXmidJOd59REog9jDnPDTG6IJ

- Online Computer Architecture Courses
  - Intro: http://www.ece.cmu.edu/~ece447/s13/doku.php
  - Advanced: http://www.ece.cmu.edu/~ece740/f11/doku.php
  - Advanced: http://www.ece.cmu.edu/~ece742/doku.php

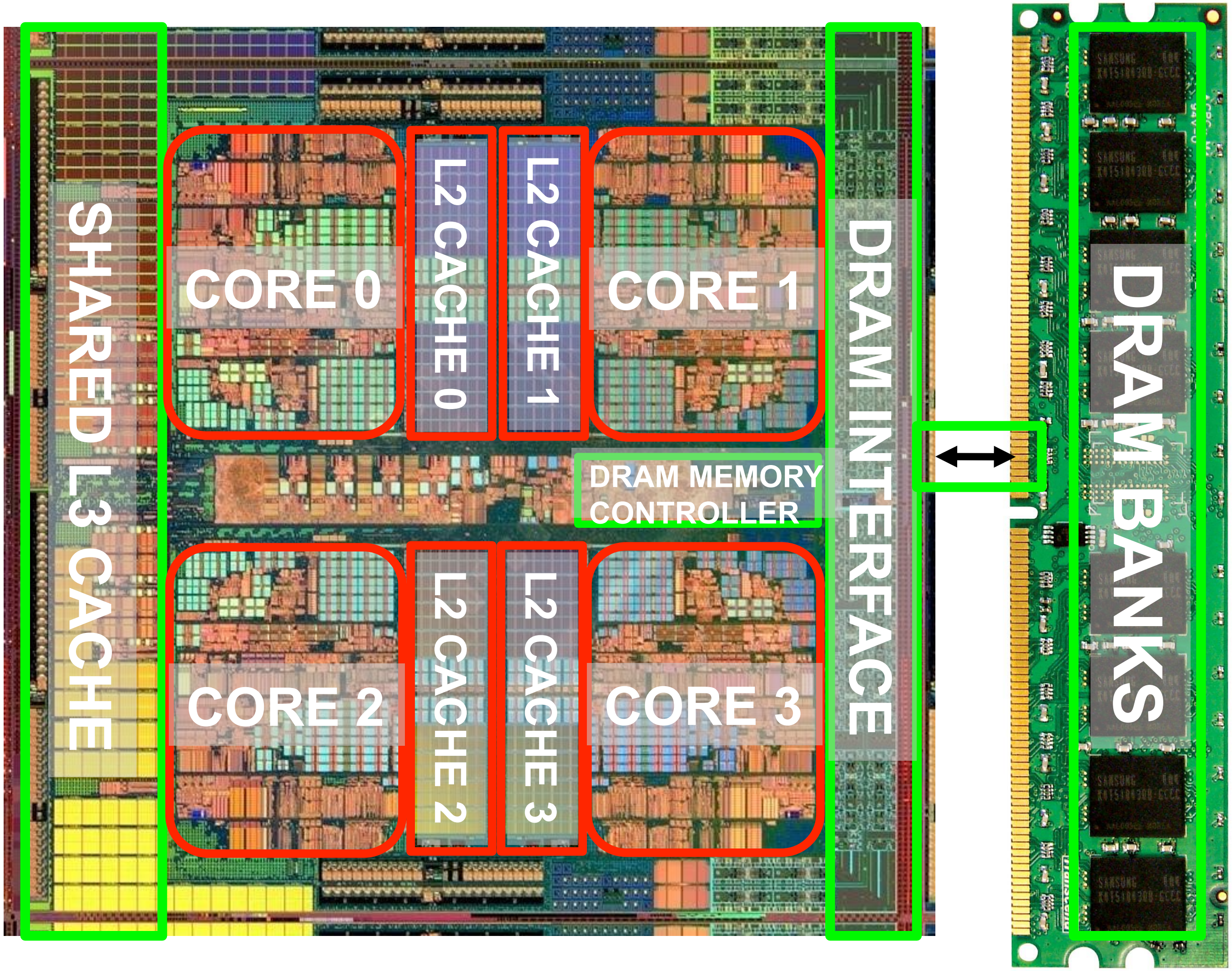
- Recent Research Papers
  - http://users.ece.cmu.edu/~omutlu/projects.htm
  - http://scholar.google.com/citations?user=7XyGUGkAAAAJ&hl=en

Agenda for Today

- What Will You Learn in This Mini-Lecture Series
- Main Memory Basics (with a Focus on DRAM)
- Major Trends Affecting Main Memory
- DRAM Scaling Problem and Solution Directions
- Solution Direction 1: System-DRAM Co-Design
- Ongoing Research
- Summary

Main Memory

Main Memory in the System



Ideal Memory

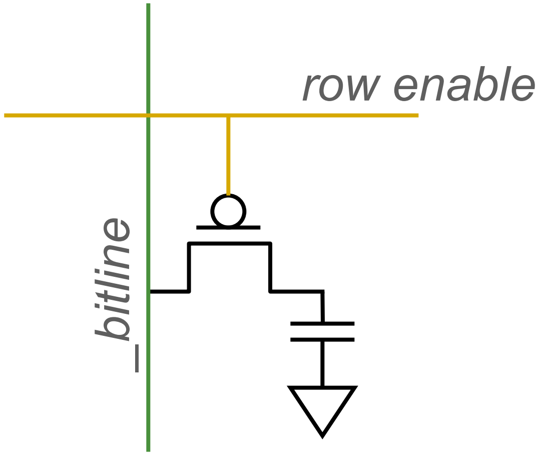
- Zero access time (latency)
- Infinite capacity
- Zero cost
- Infinite bandwidth (to support multiple accesses in parallel)

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

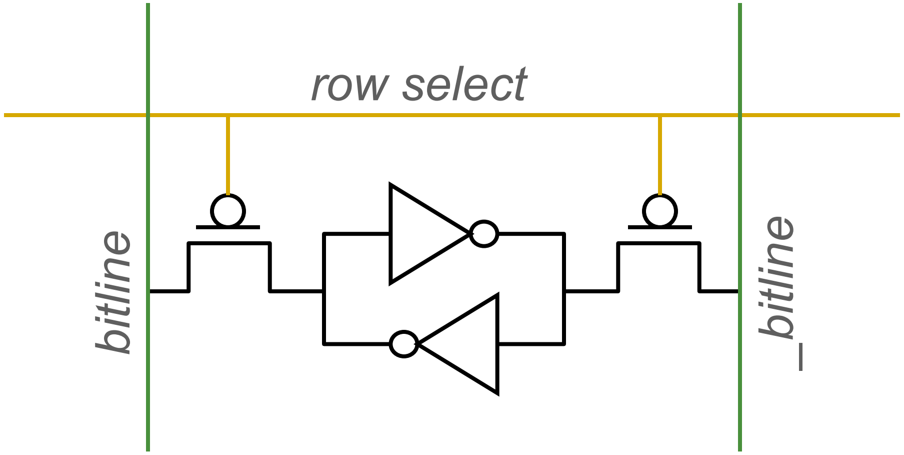


- Capacitor leaks through the RC path
  - DRAM cell loses charge over time
  - DRAM cell needs to be refreshed

  - 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



Memory Bank: A Fundamental Concept

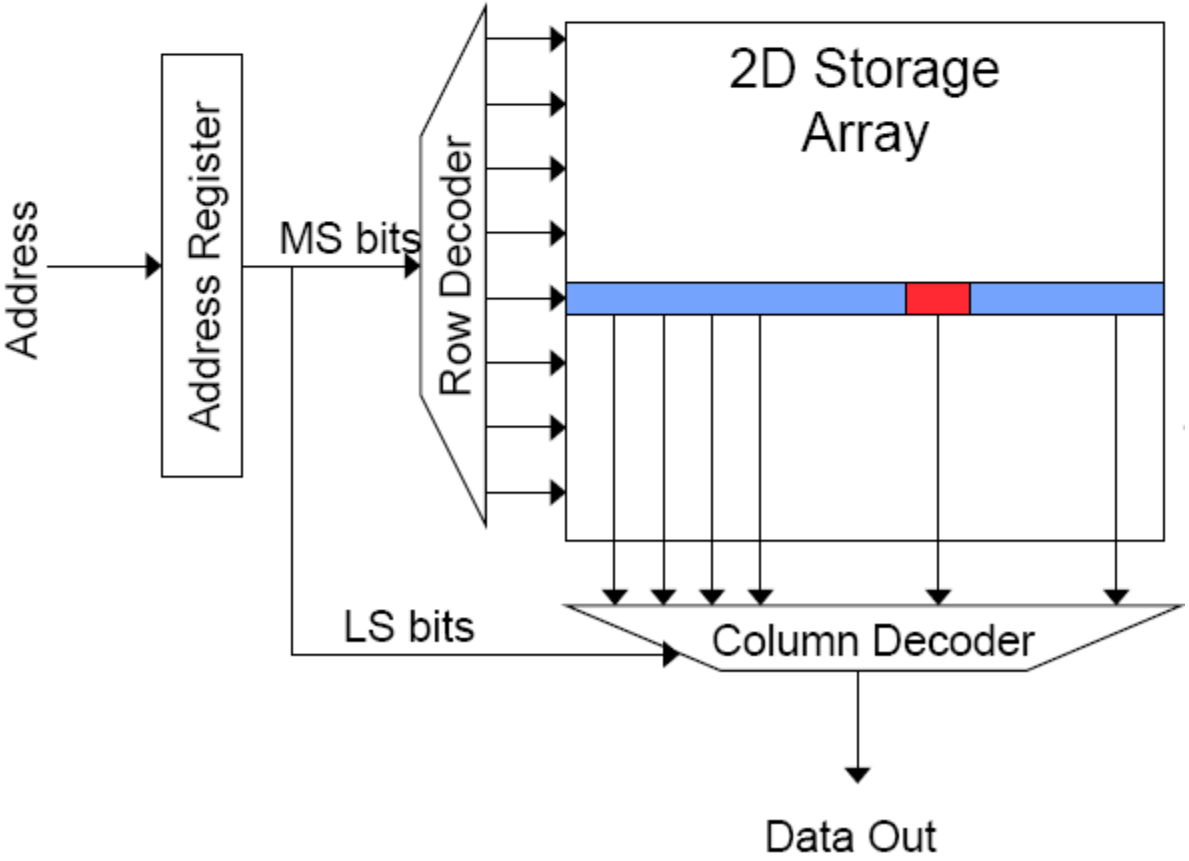
- Interleaving (banking)
  - Problem: a single monolithic memory array takes long to access and does not enable multiple accesses in parallel

  - 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

  - 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

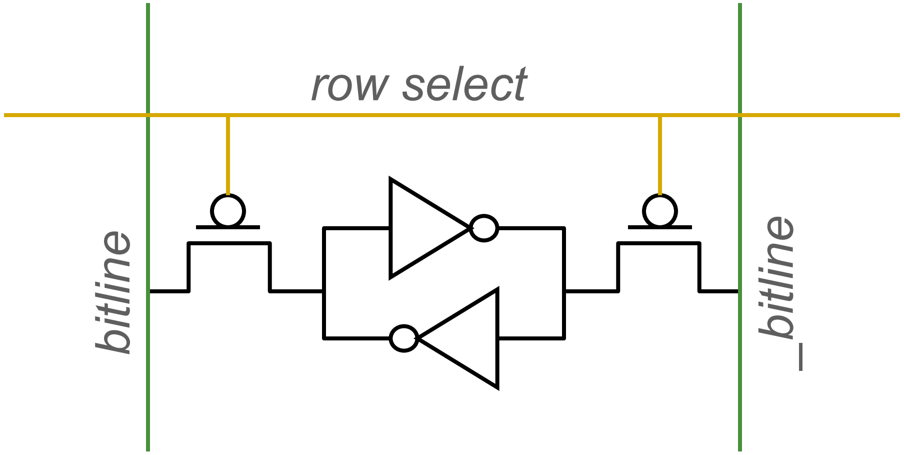
  2. Selected bits drive bit-lines
     - Entire row read

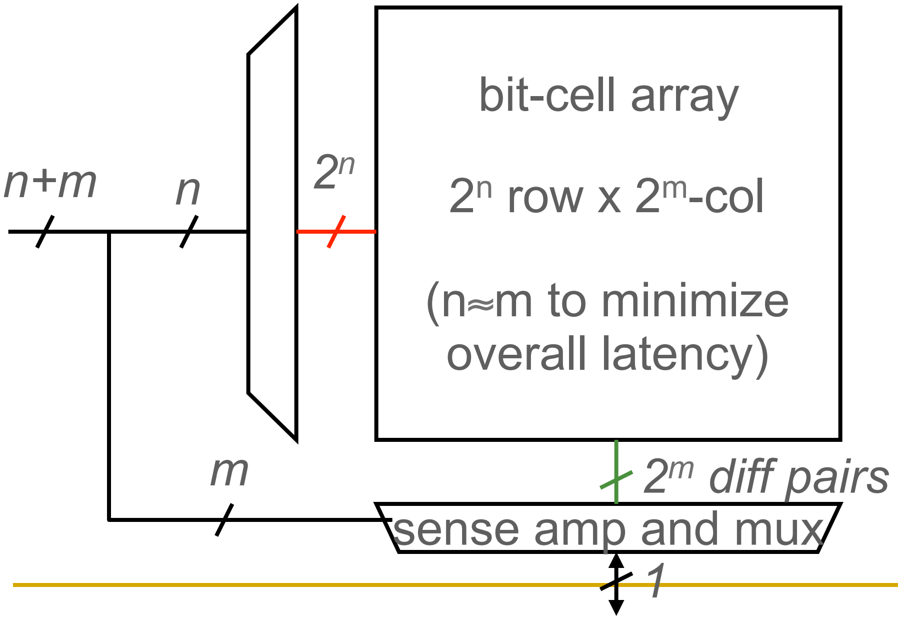
  3. Amplify row data

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

  5. Precharge bit-lines
     - For next access

SRAM (Static Random Access Memory)





Read Sequence

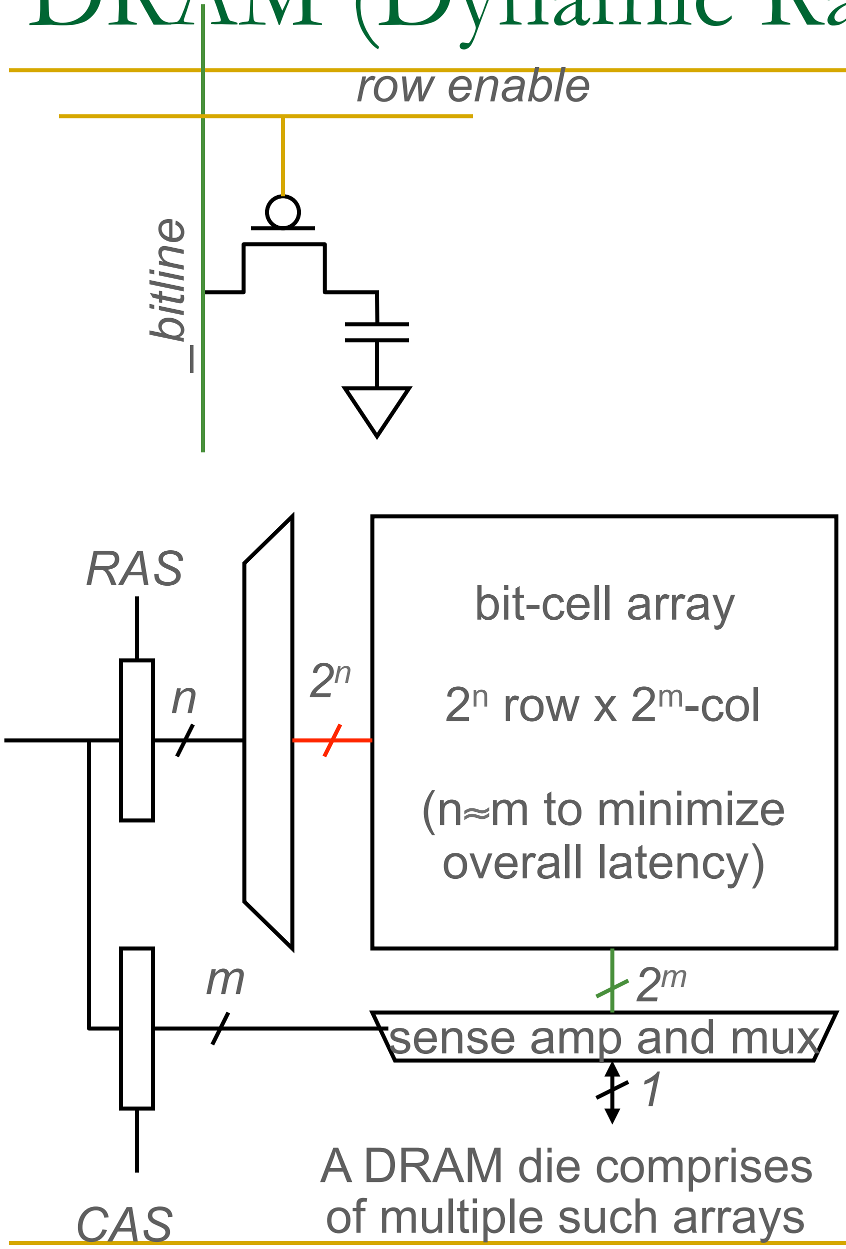
1. address decode
2. drive row select
3. selected bit-cells drive bitlines
   (entire row is read together)
4. differential sensing and column select
   (data is ready)
5. precharge all bitlines
   (for next read or write)

Access latency dominated by steps 2 and 3

Cycling time dominated by steps 2, 3 and 5

- step 2 proportional to 2^m
- step 3 and 5 proportional to 2^n

DRAM (Dynamic Random Access Memory)



A DRAM die comprises of multiple such arrays

Bits stored as charges on node capacitance (non-restorative)

- bit cell loses charge when read
- bit cell loses charge over time

Read Sequence

1~3 same as SRAM

4. a “flip-flopping” sense amp amplifies and regenerates the bitline, data bit is mux’ed out
5. precharge all bitlines

Refresh: A DRAM controller must periodically read all rows within the allowed refresh time (10s of ms) such that charge is restored in cells

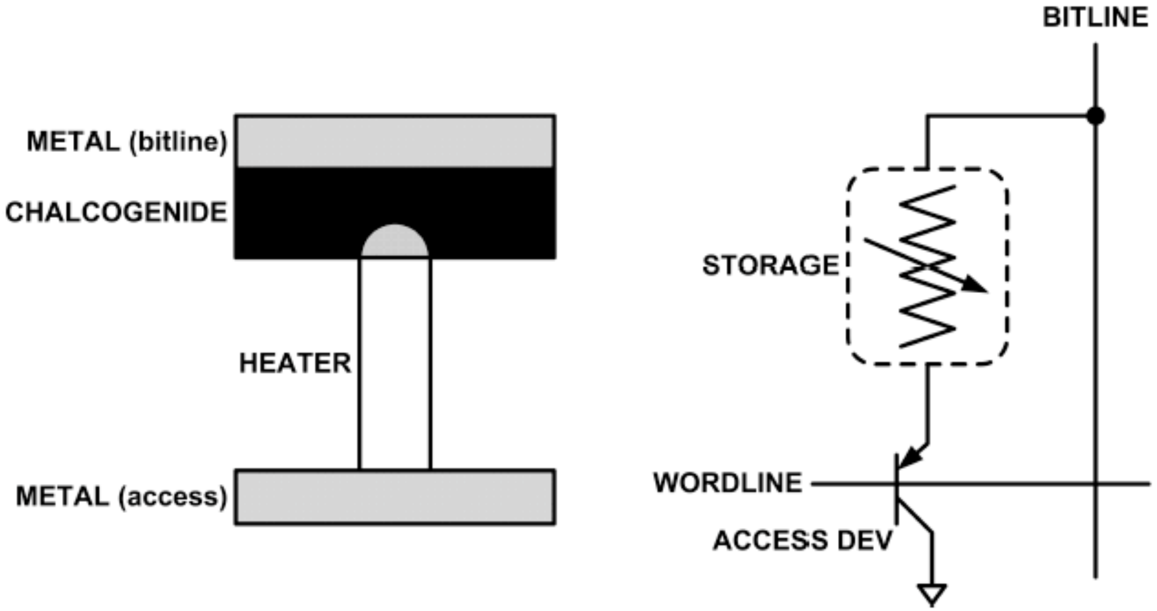
DRAM vs. SRAM

- DRAM
  - Slower access (capacitor)
  - Higher density (1T 1C cell)
  - Lower cost
  - Requires refresh (power, performance, circuitry)
  - Manufacturing requires putting capacitor and logic together

- SRAM
  - Faster access (no capacitor)
  - Lower density (6T cell)
  - Higher cost
  - No need for refresh
  - Manufacturing compatible with logic process (no capacitor)

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

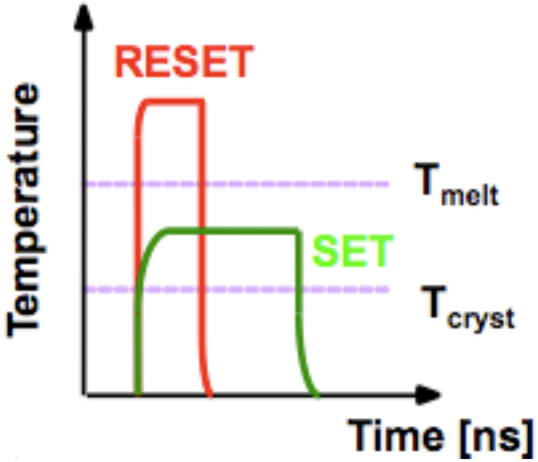


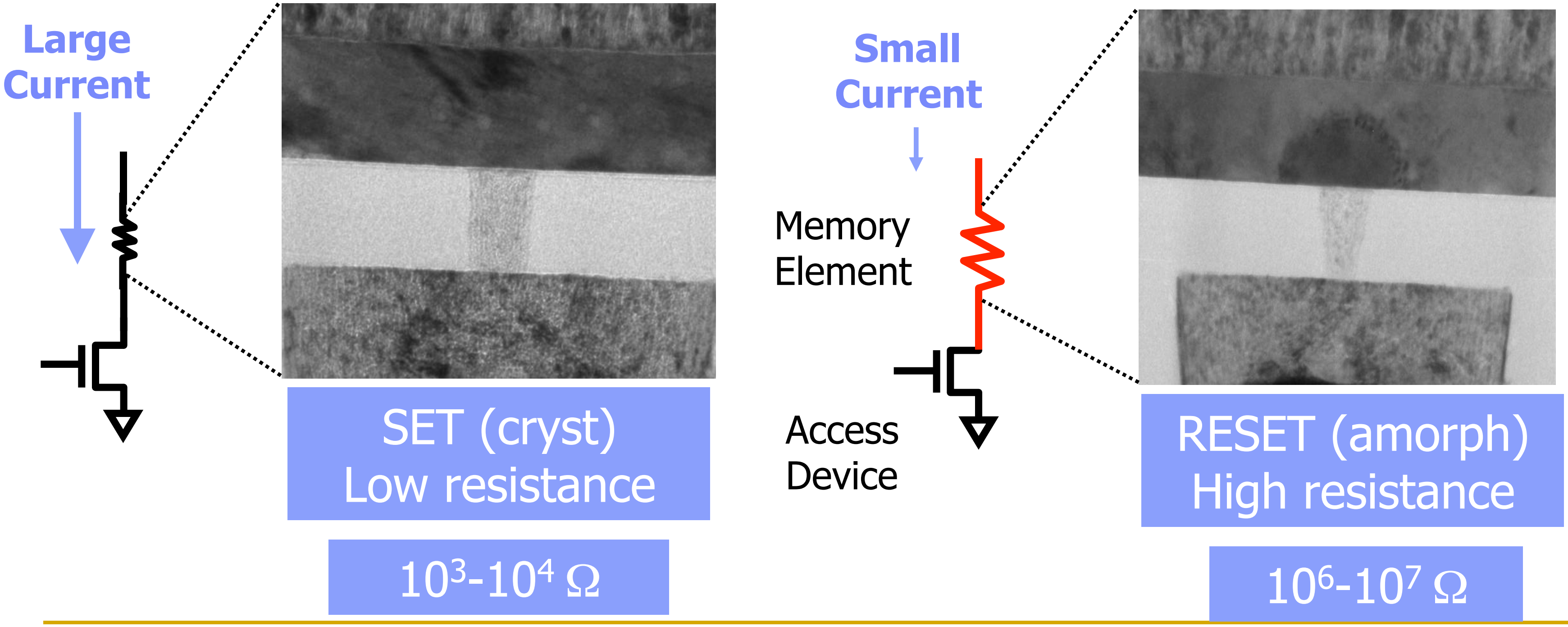
PCM is resistive memory: High resistance (0), Low resistance (1)

Lee, Ipek, Mutlu, Burger, "Architecting Phase Change Memory as a Scalable DRAM Alternative," ISCA 2009.

An Aside: How Does PCM Work?

- Write: change phase via current injection
  - SET: sustained current to heat cell above T*cryst*
  - RESET: cell heated above T*melt* and quenched
- Read: detect phase via material resistance
  - amorphous/crystalline




Large Current

SET (cryst)
Low resistance

10^3-10^4 Ω

Small Current

Memory Element

Access Device

RESET (amorph)
High resistance

10^6-10^7 Ω

Photo Courtesy: Bipin Rajendran, IBM   Slide Courtesy: Moinuddin Qureshi, IBM

The Problem

- Bigger is slower
  - SRAM, 512 Bytes, sub-nanosec
  - SRAM, KByte~MByte, ~nanosec
  - DRAM, Gigabyte, ~50 nanosec
  - Hard Disk, Terabyte, ~10 millisec

- Faster is more expensive (dollars and chip area)
  - SRAM, < 10$ per Megabyte
  - DRAM, < 1$ per Megabyte
  - Hard Disk < 1$ per Gigabyte
  - These sample values scale with time

- Other technologies have their place as well
  - Flash memory, Phase-change memory (not mature yet)

Why Memory Hierarchy?

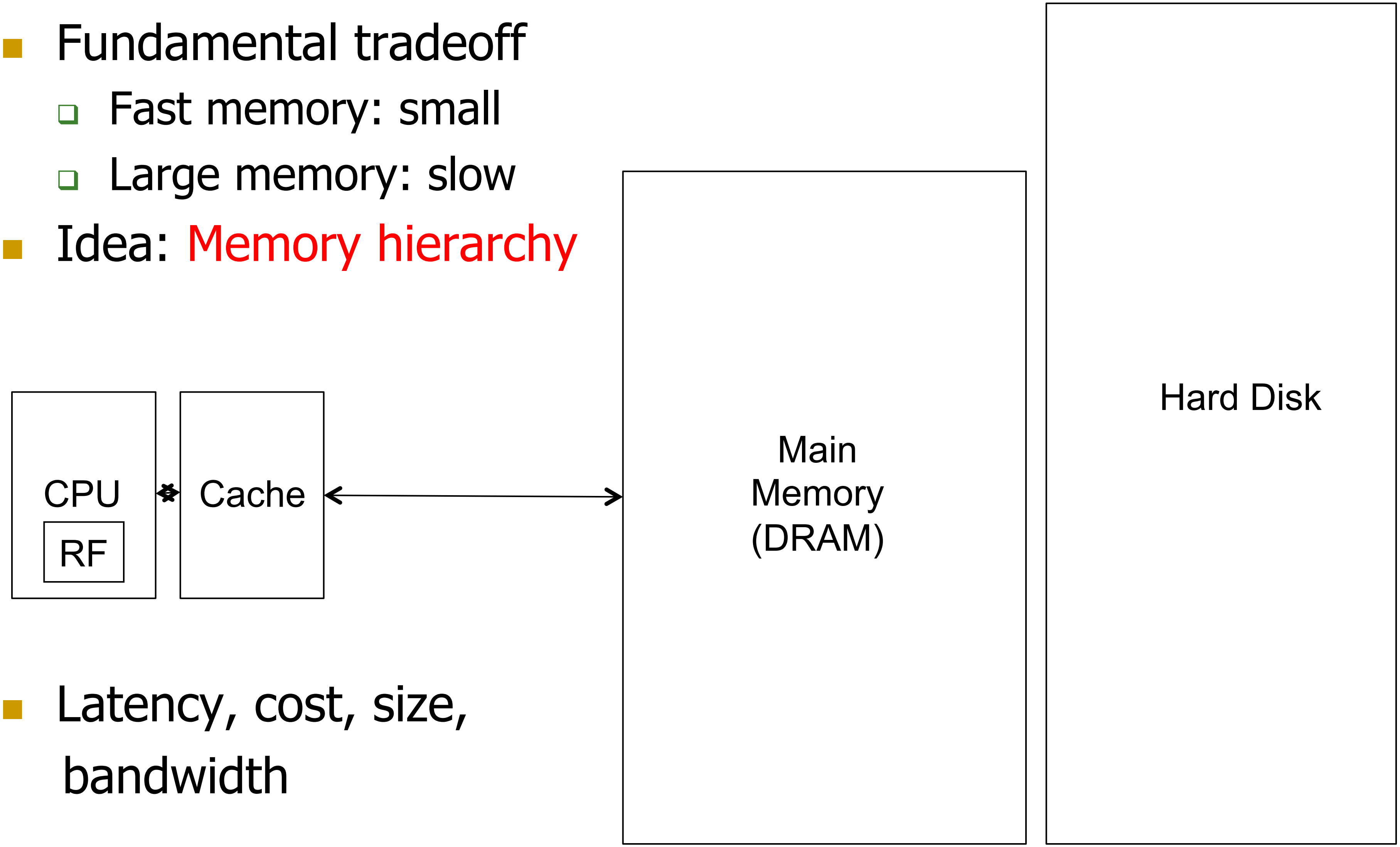
- We want both fast and large

- 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

Locality

- One’s recent past is a very good predictor of his/her near future.

- Temporal Locality: If you just did something, it is very likely that you will do the same thing again soon

- Spatial Locality: If you just did something, it is very likely you will do something similar/related

Memory Locality

- A "typical" program has a lot of locality in memory references
  - typical programs are composed of "loops"

- Temporal: A program tends to reference the same memory location many times and all within a small window of time

- Spatial: A program tends to reference a cluster of memory locations at a time
  - most notable examples:
    - 1. instruction memory references
    - 2. array/data structure references

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."

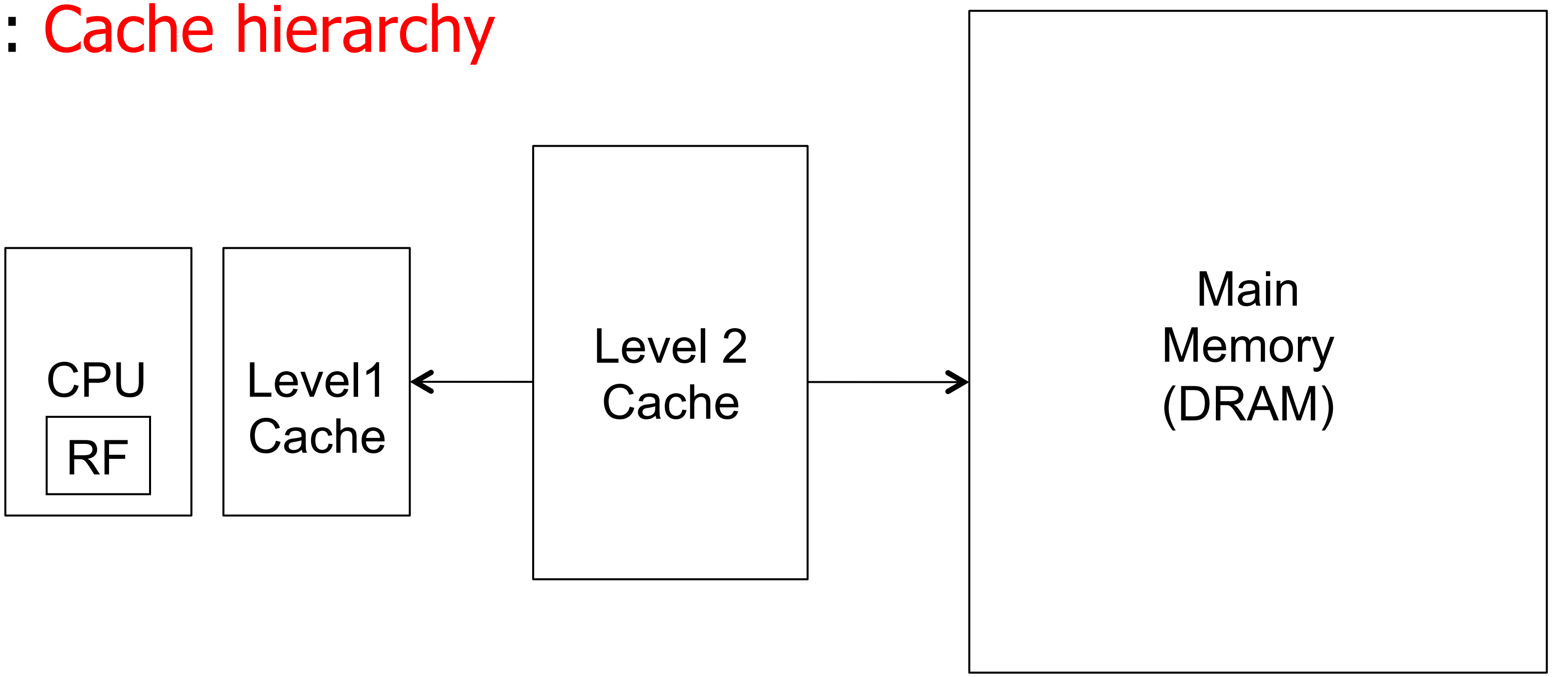
Caching Basics: Exploit Spatial Locality

- 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.

Caching in a Pipelined Design

- The cache needs to be tightly integrated into the pipeline
  - Ideally, access in 1-cycle so that dependent operations do not stall
- High frequency pipeline → Cannot make the cache large
  - But, we want a large cache AND a pipelined design
- Idea: Cache hierarchy



A Note on Manual vs. Automatic Management

- Manual: Programmer manages data movement across levels

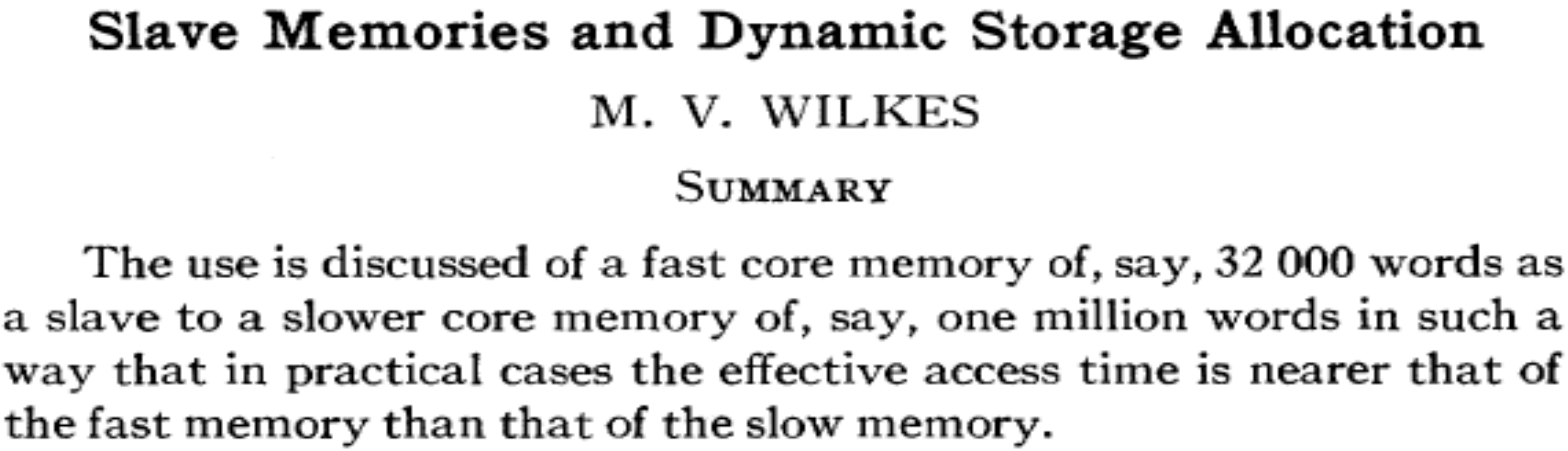
  -- too painful for programmers on substantial programs
  - "core" vs "drum" memory in the 50's
  - still done in some embedded processors (on-chip scratch pad SRAM in lieu of a cache)

- 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

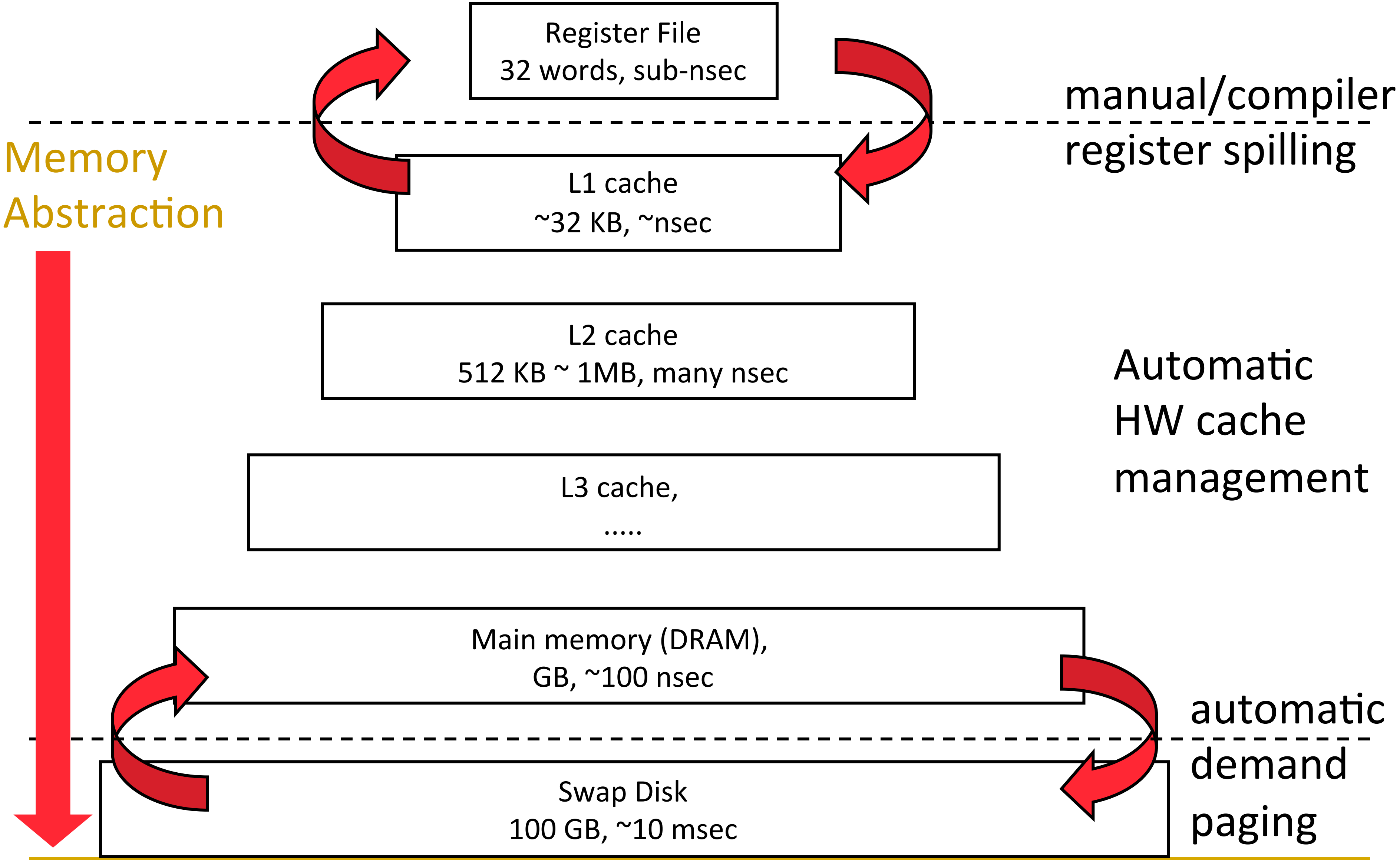
M. V. WILKES

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



Register File
32 words, sub-nsec

manual/compiler
register spilling

Memory
Abstraction

L1 cache
~32 KB, ~nsec

L2 cache
512 KB ~ 1MB, many nsec

Automatic
HW cache
management

L3 cache,
.....

Main memory (DRAM),
GB, ~100 nsec

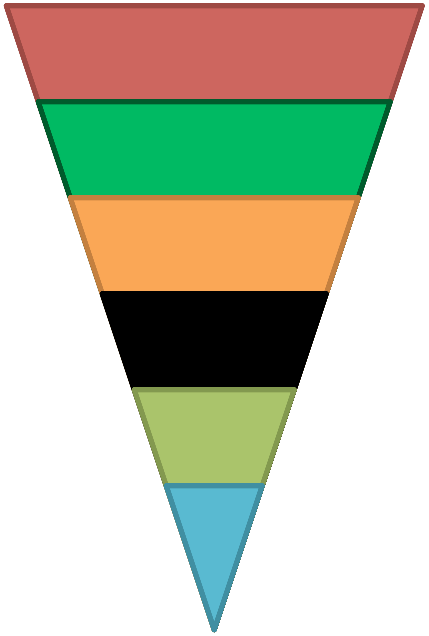
automatic
demand
paging

Swap Disk
100 GB, ~10 msec

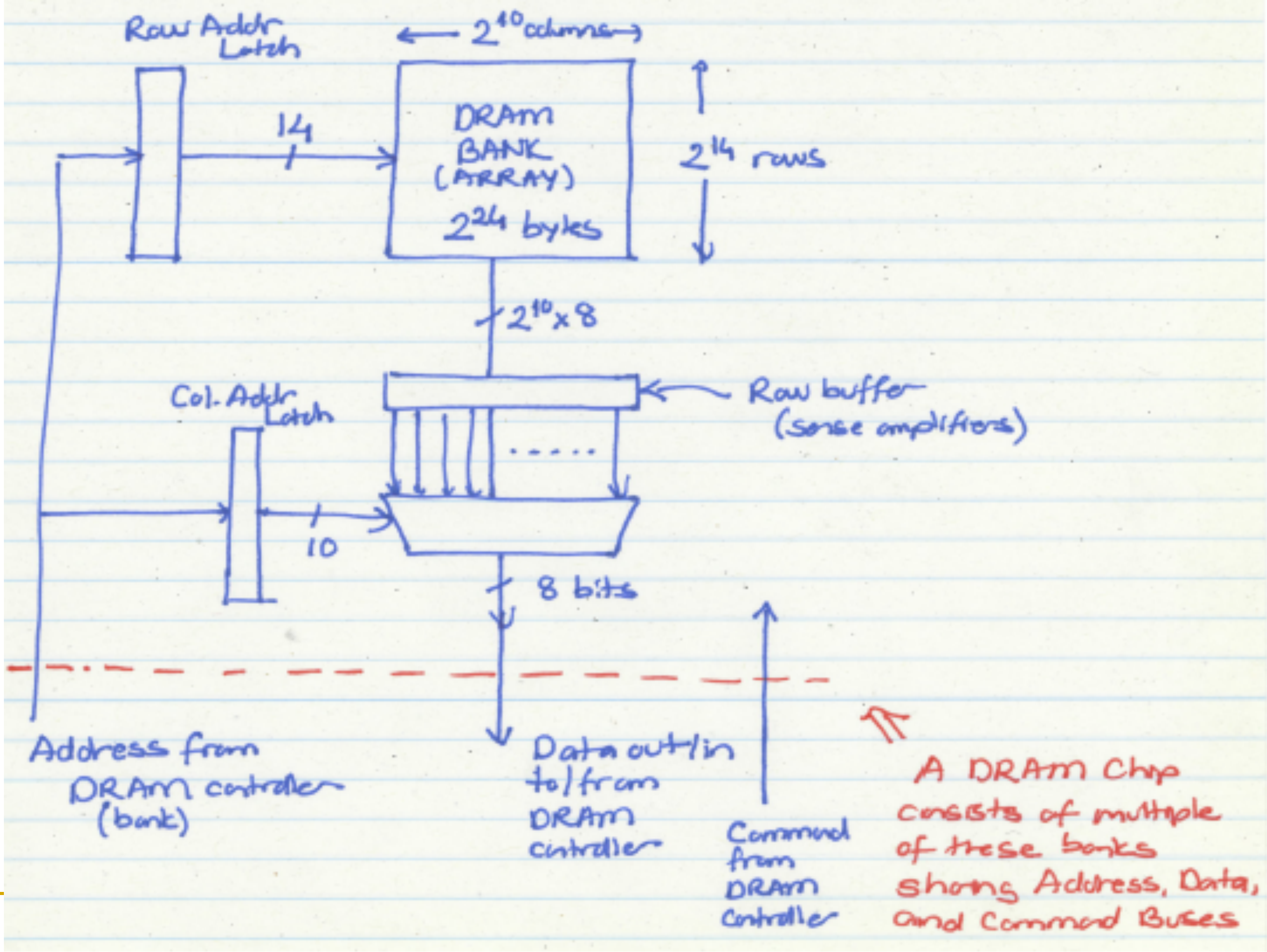
The DRAM Subsystem

DRAM Subsystem Organization

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



The DRAM Bank Structure


Row Addr Latch
← 2^{10} columns →
14
DRAM BANK (ARRAY)
2^{24} bytes
2^{14} rows
$2^{10} \times 8$
Col. Addr Latch
Row buffer (sense amplifiers)
.....
10
8 bits
Address from DRAM controller (bank)
Data out/in to/from DRAM controller
Command from DRAM Controller

A DRAM Chip consists of multiple of these banks showing Address, Data, and Command Buses

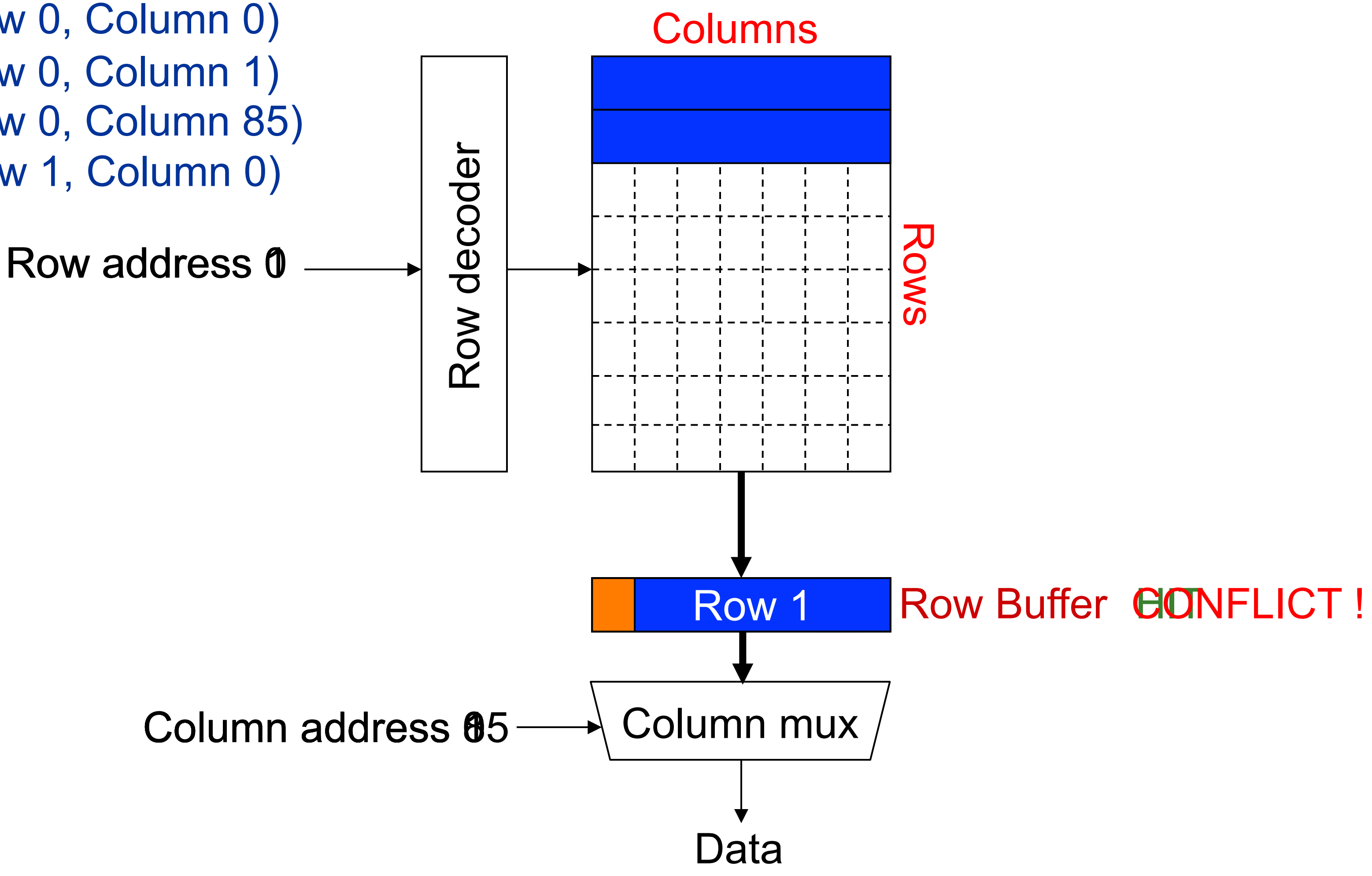
Page Mode DRAM

- 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

DRAM Bank Operation

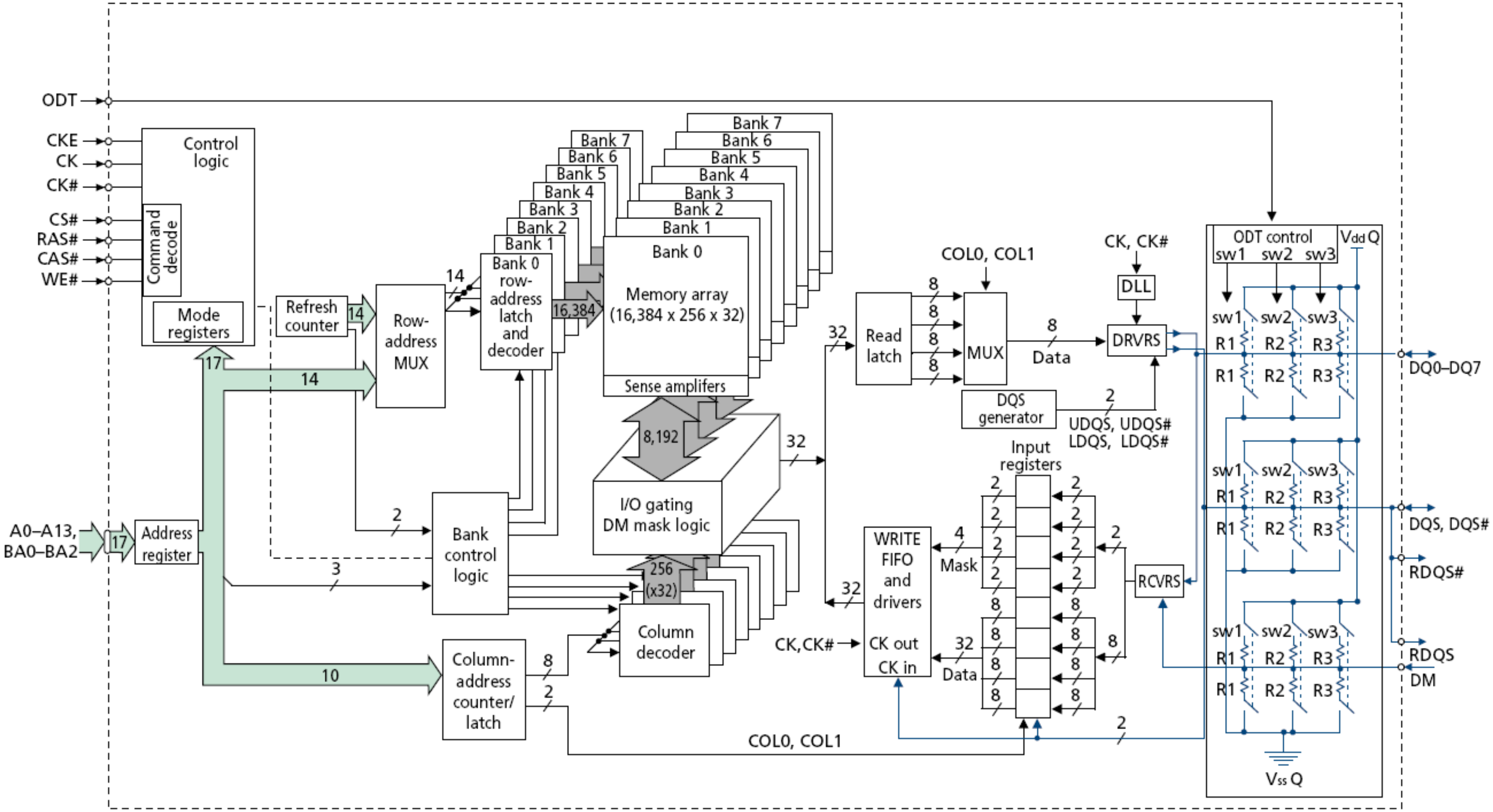
Access Address:
(Row 0, Column 0)
(Row 0, Column 1)
(Row 0, Column 85)
(Row 1, Column 0)



The DRAM Chip

- Consists of multiple banks (2-16 in Synchronous DRAM)
- Banks share command/address/data buses
- The chip itself has a narrow interface (4-16 bits per read)

128M x 8-bit DRAM Chip



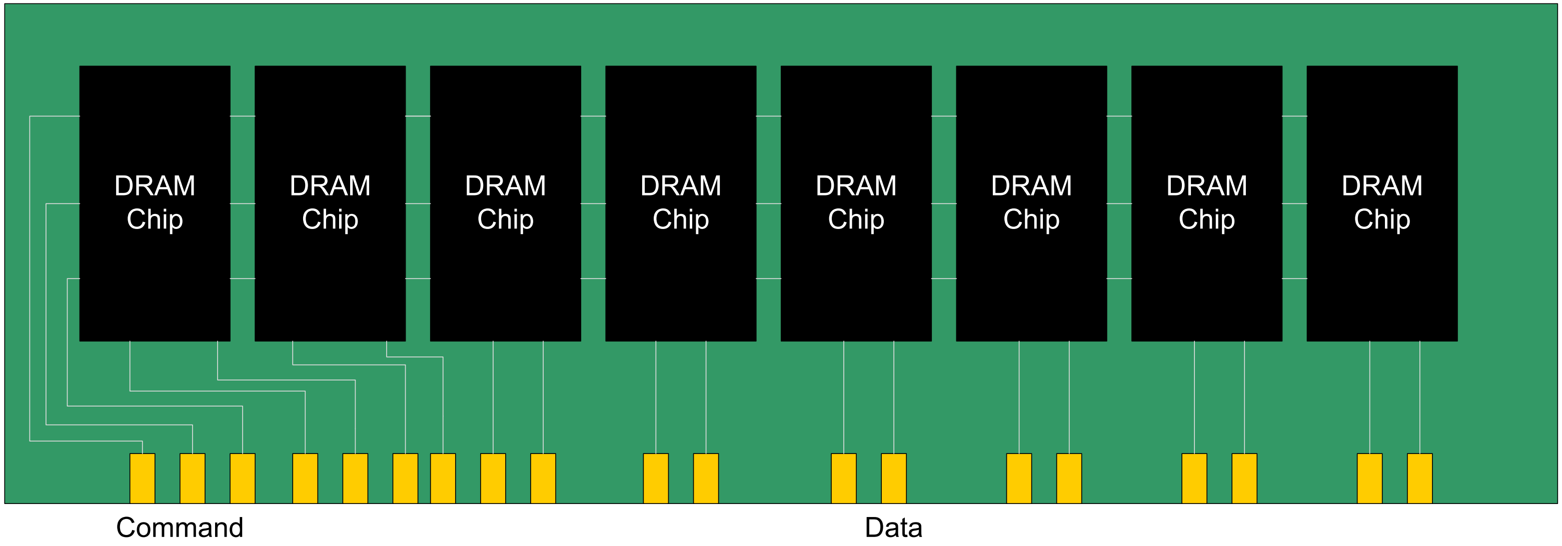
DRAM Rank and Module

- 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

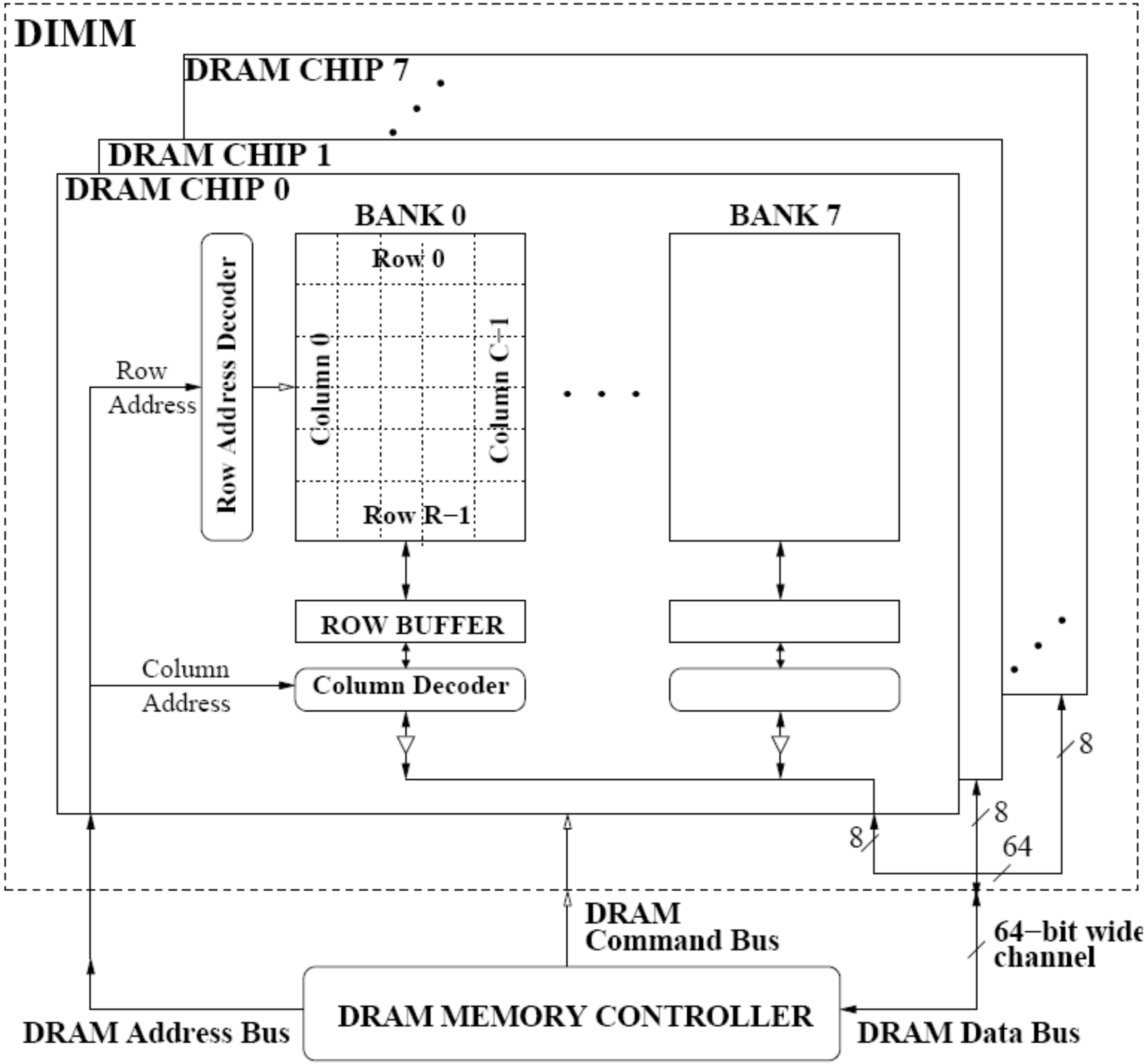
- A DRAM module consists of one or more ranks
  - E.g., DIMM (dual inline memory module)
  - This is what you plug into your motherboard

- 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)

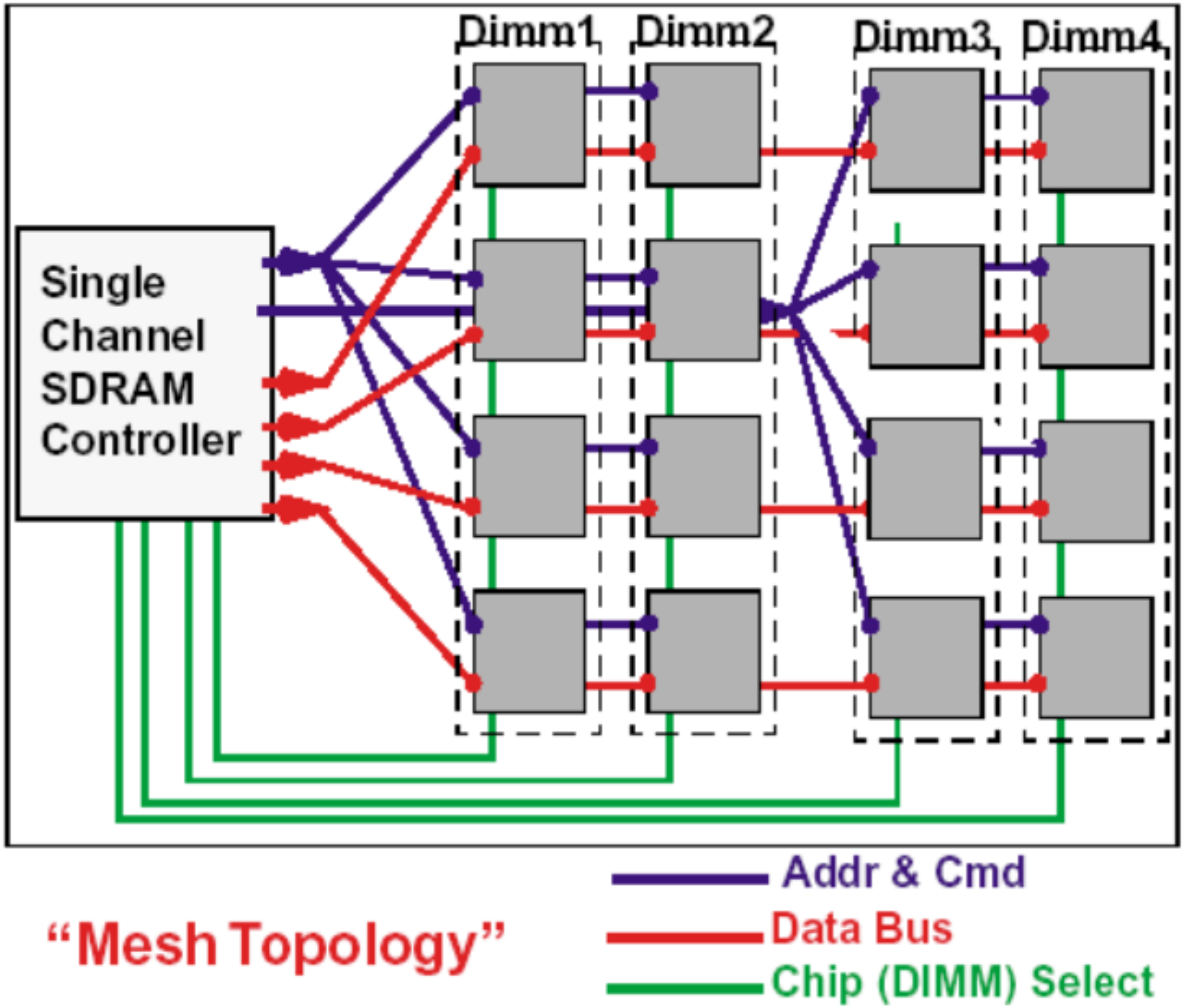


A 64-bit Wide DIMM (One Rank)



- 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

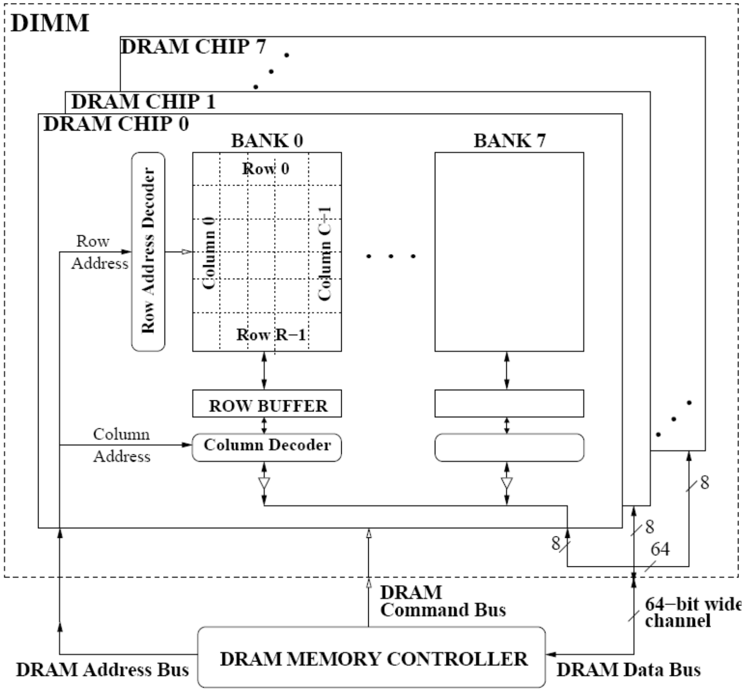
Multiple DIMMs

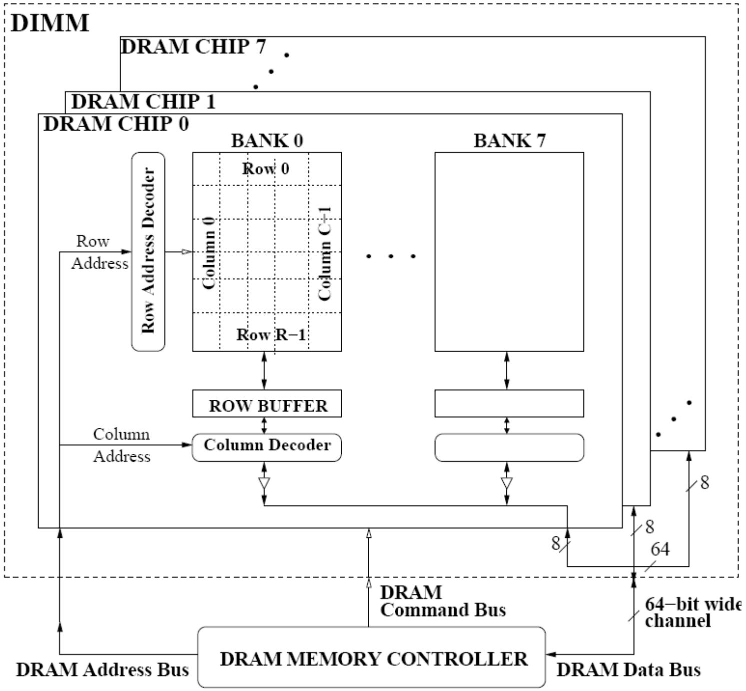


"Mesh Topology"

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

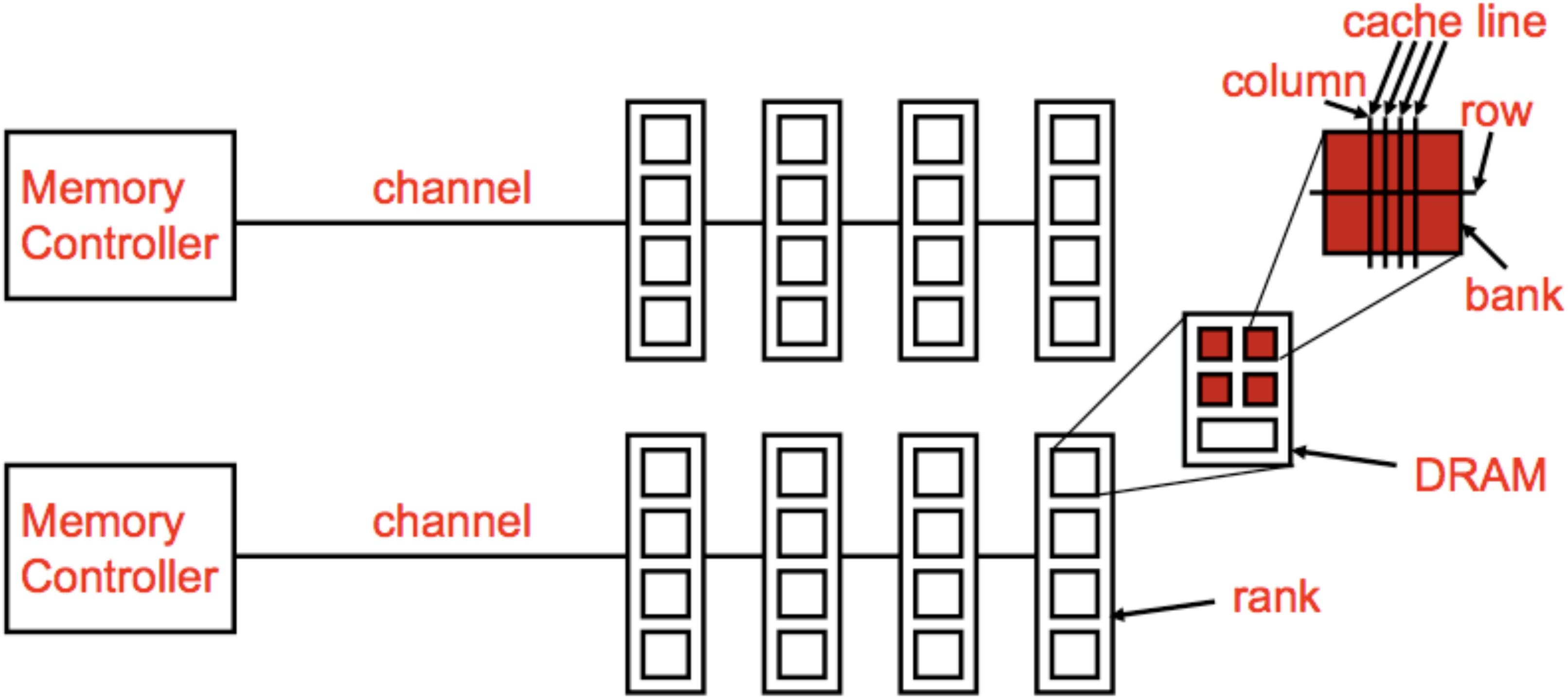
DRAM Channels



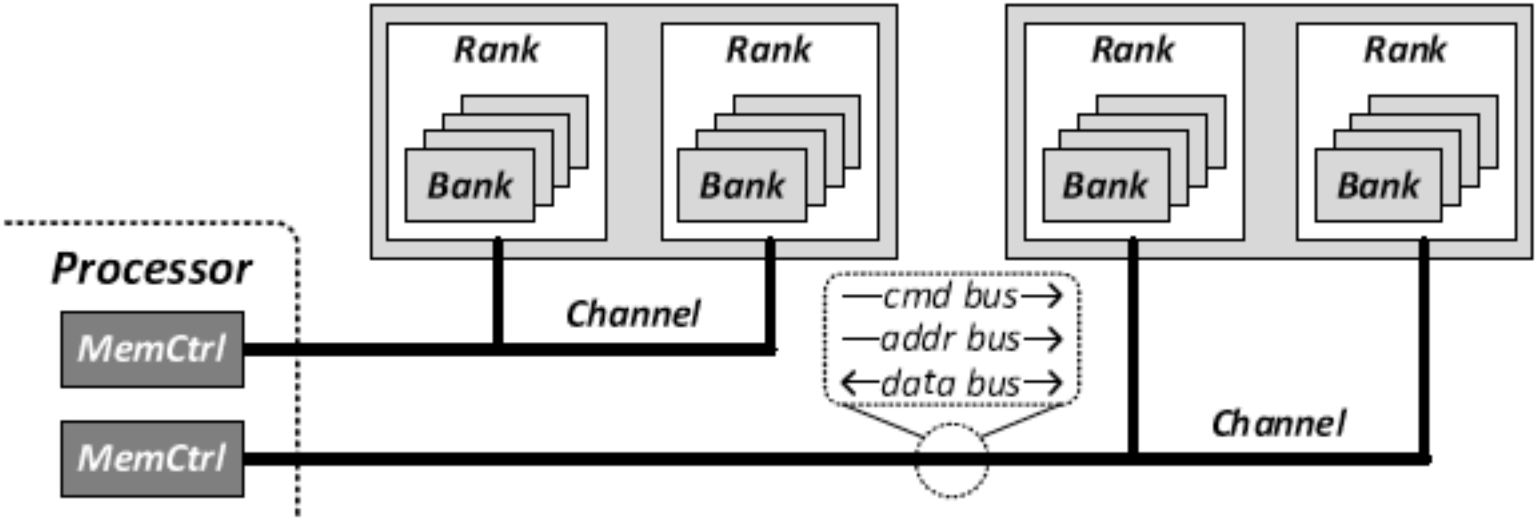


- 2 Independent Channels: 2 Memory Controllers (Above)
- 2 Dependent/Lockstep Channels: 1 Memory Controller with wide interface (Not Shown above)

Generalized Memory Structure



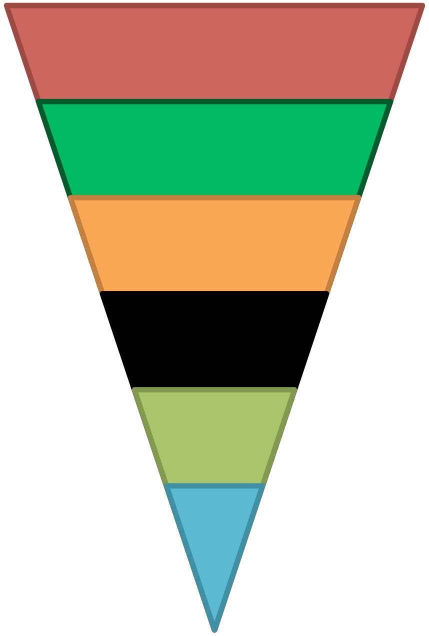
Generalized Memory Structure



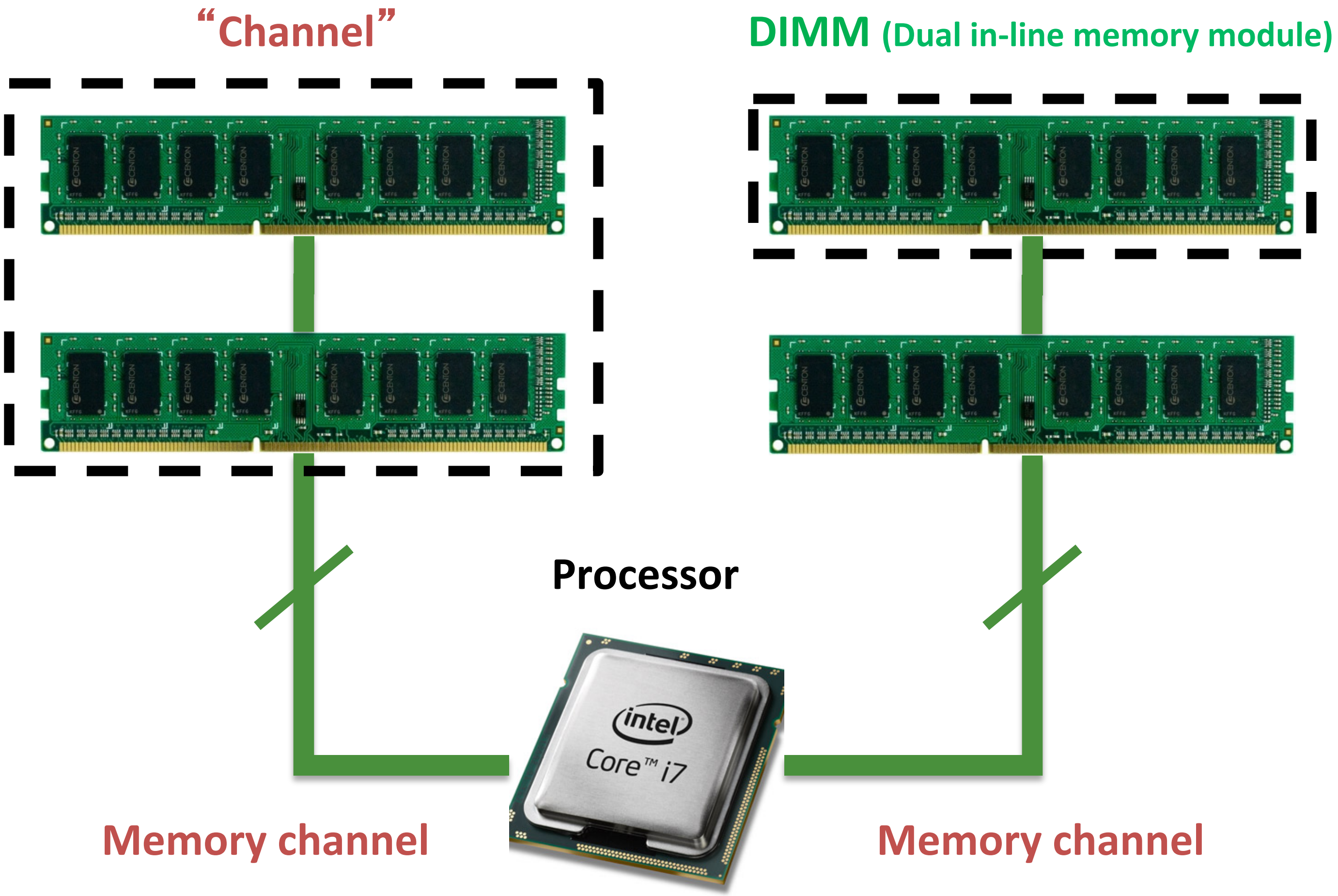
The DRAM Subsystem
The Top Down View

DRAM Subsystem Organization

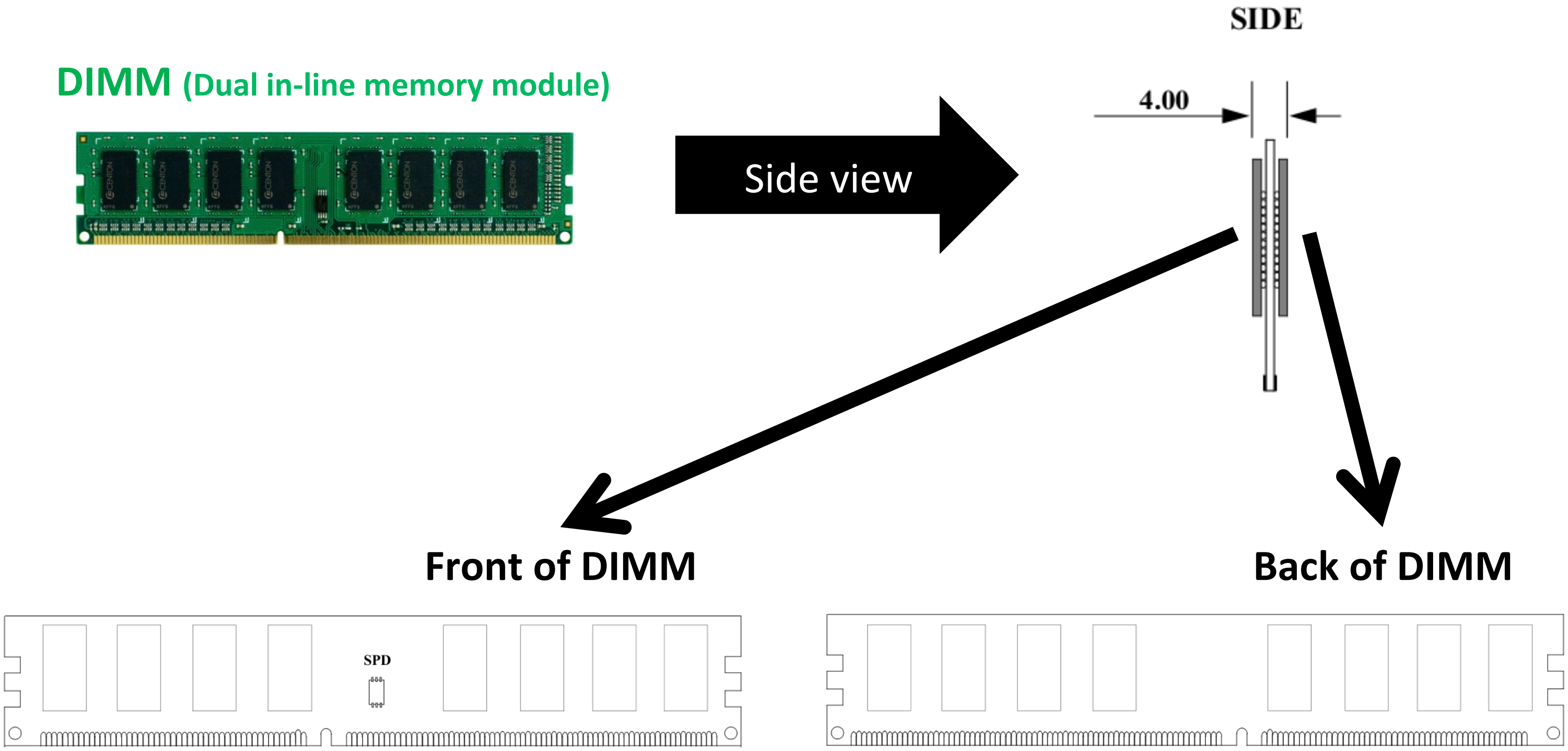
- Channel
- DIMM
- Rank
- Chip
- Bank
- Row/Column



The DRAM subsystem



Breaking down a DIMM



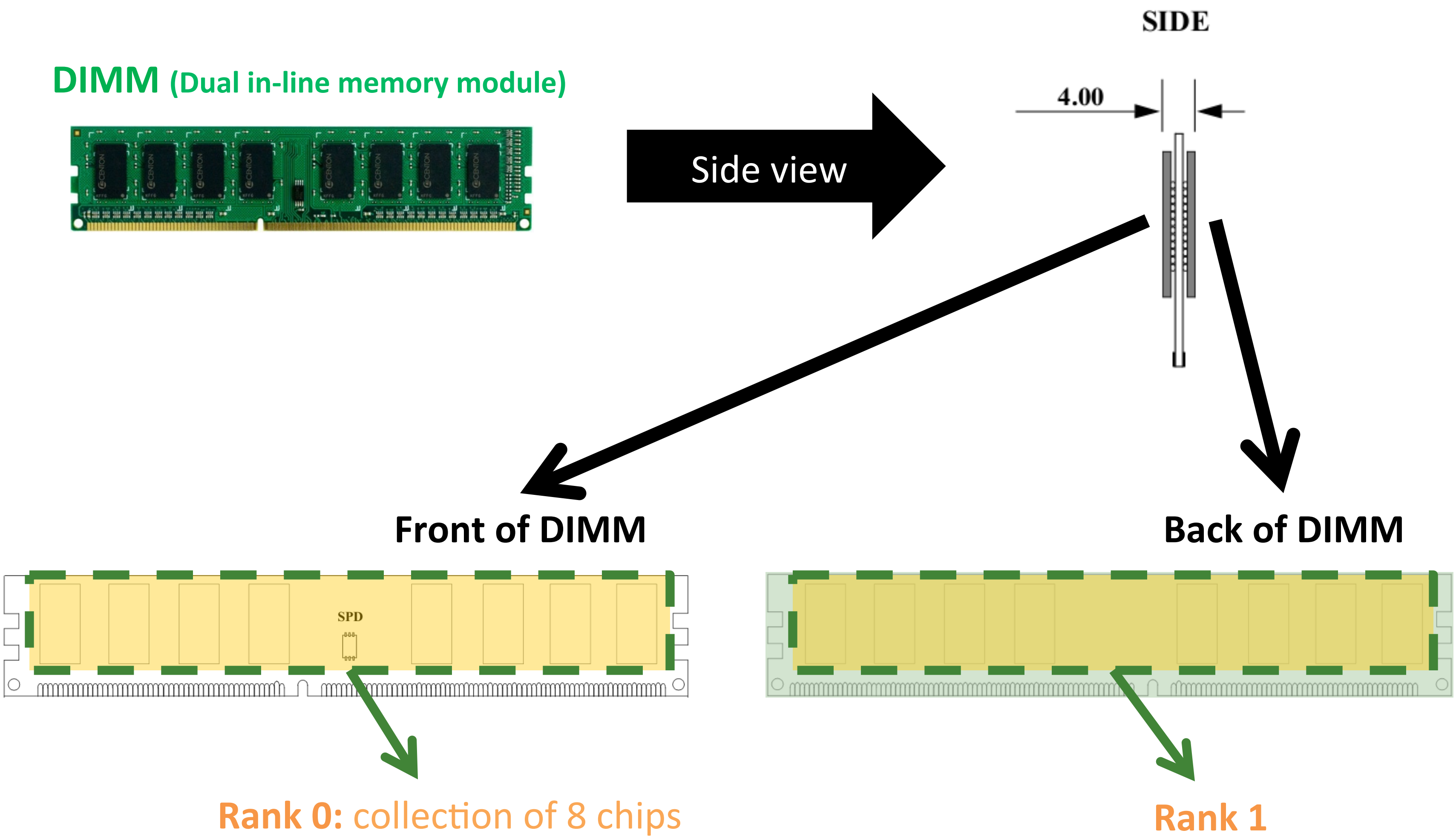
DIMM (Dual in-line memory module)

Side view

Front of DIMM

Back of DIMM

Breaking down a DIMM



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

Side view

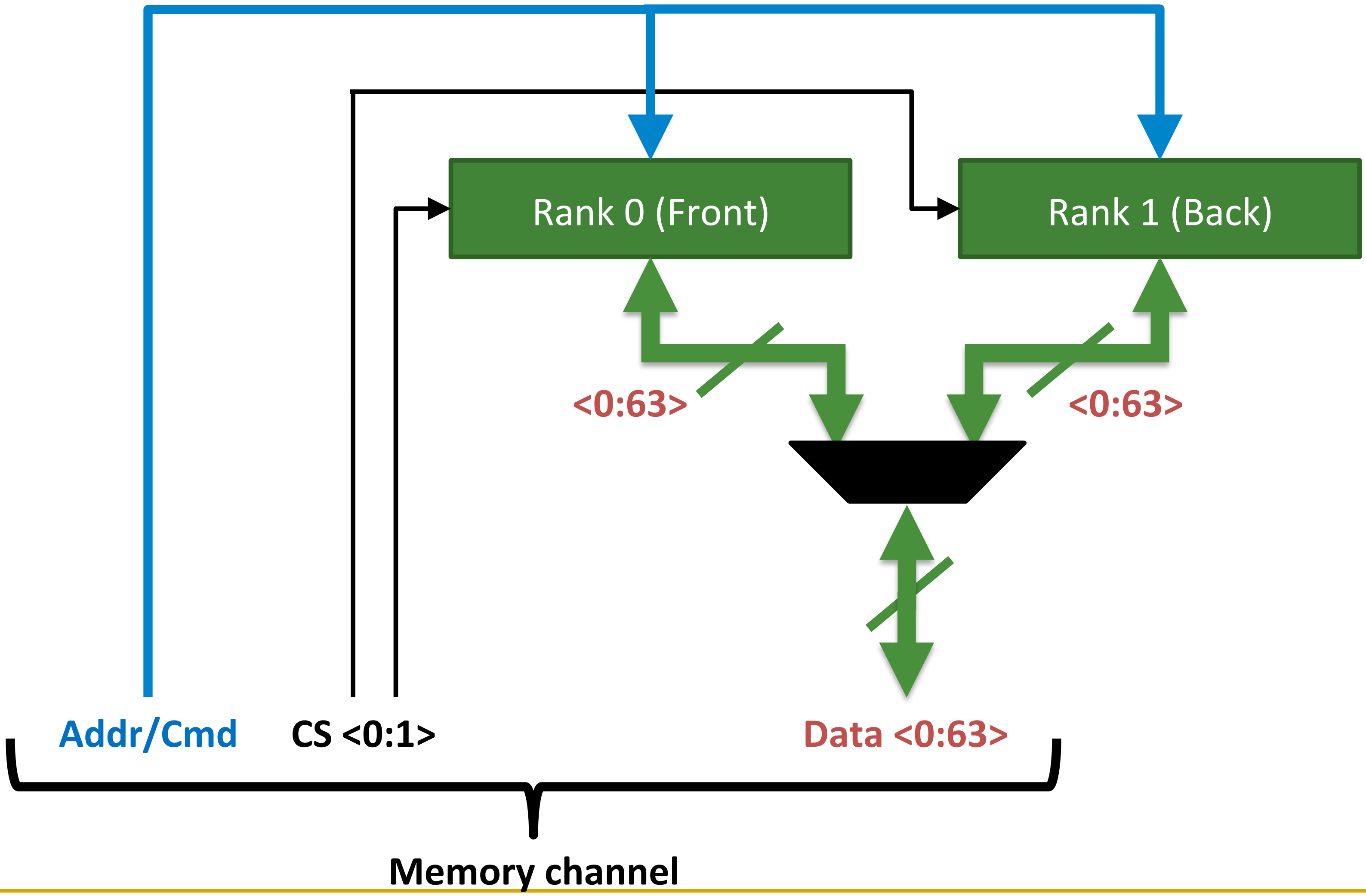
Front of DIMM

Back of DIMM

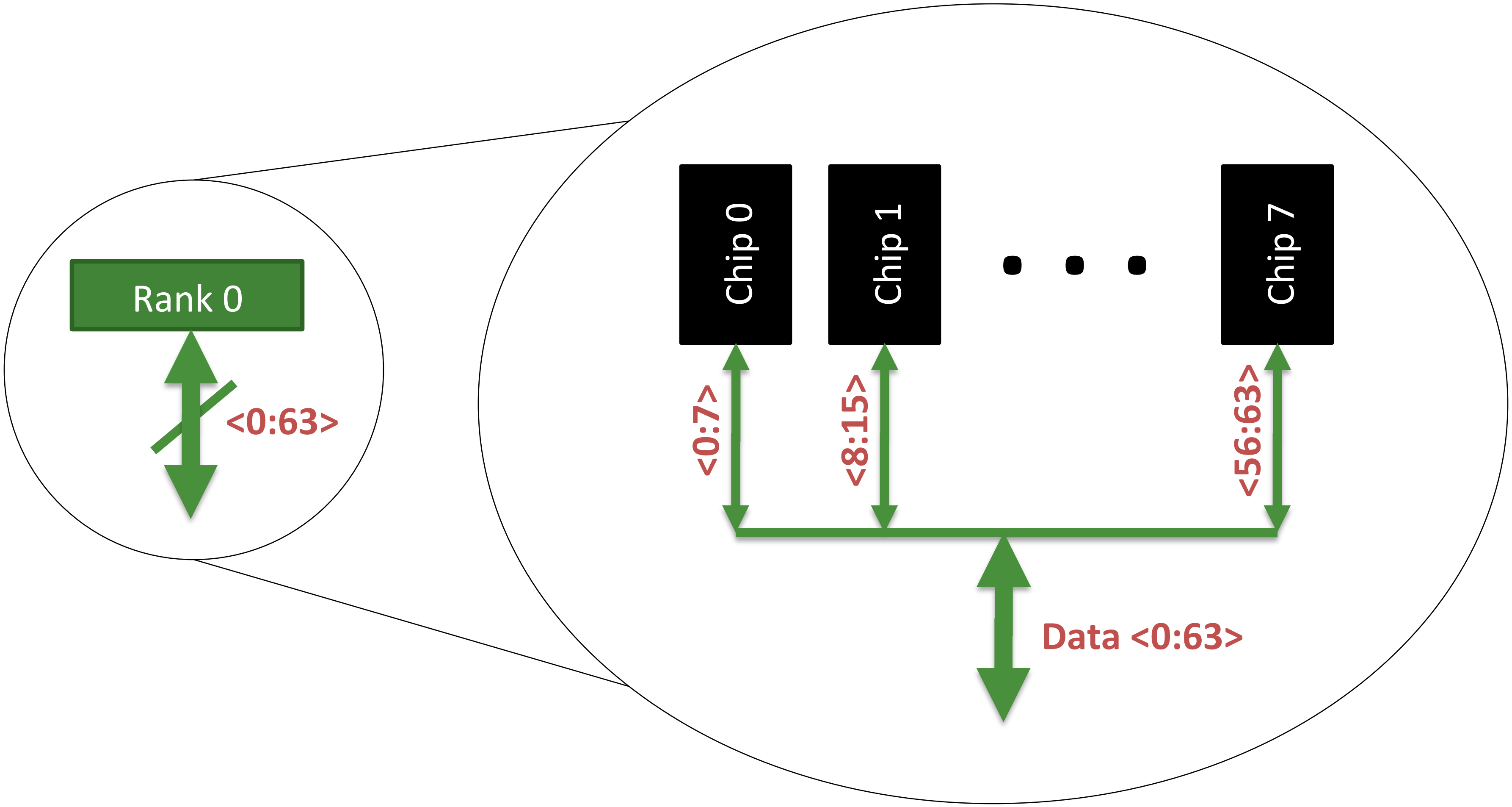
Rank 0: collection of 8 chips

Rank 1

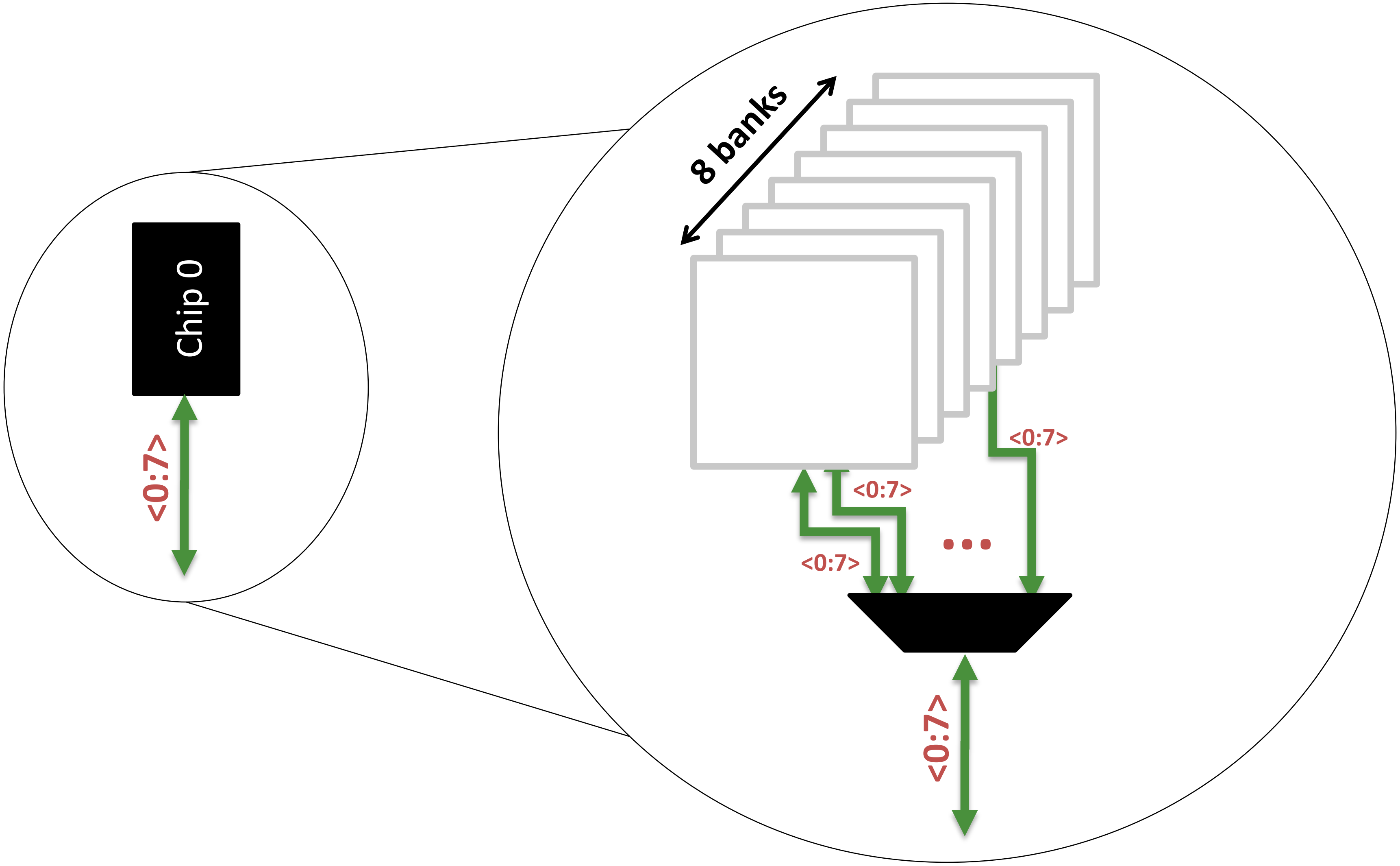
Rank


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

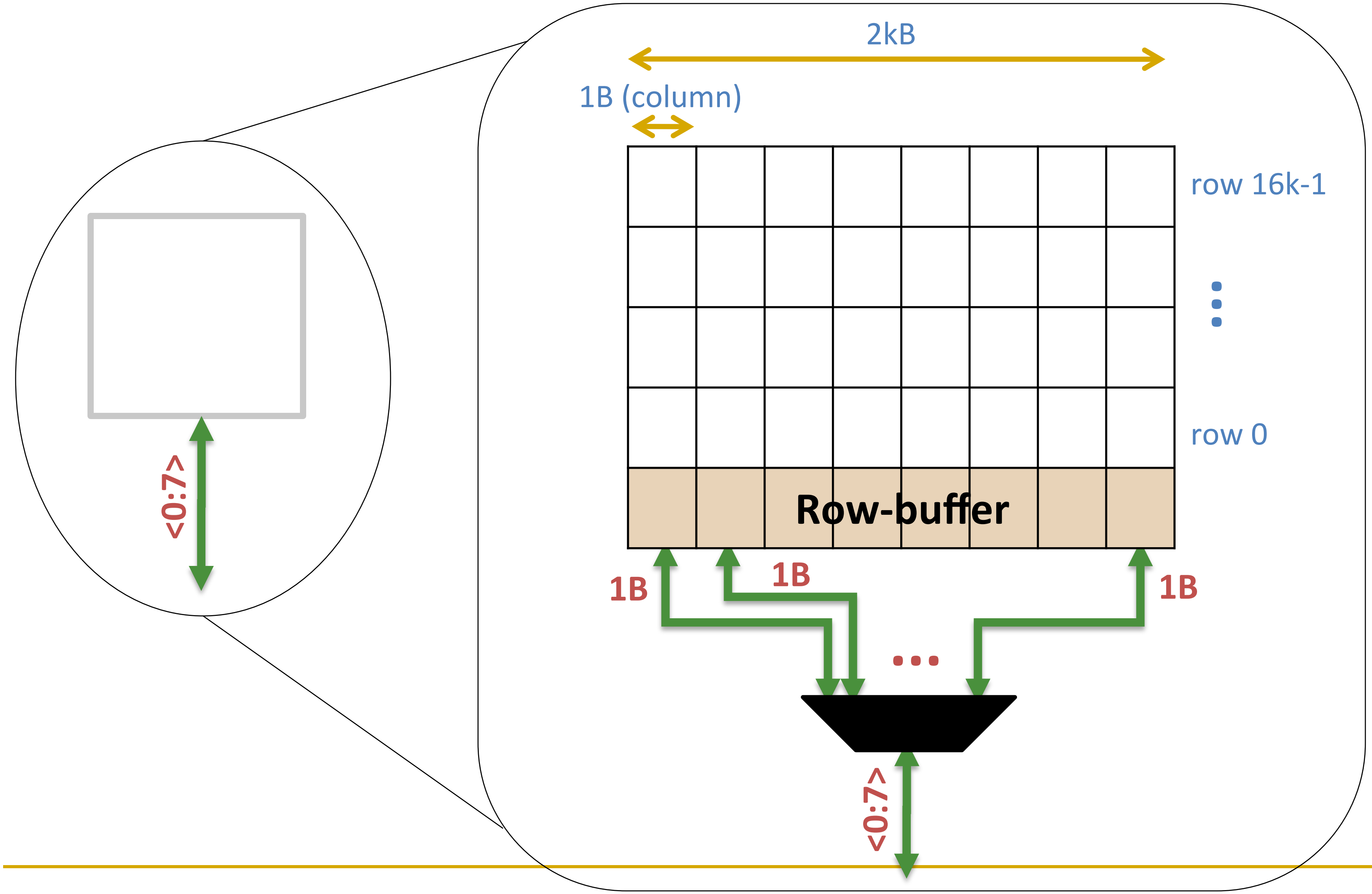
Breaking down a Rank



Breaking down a Chip

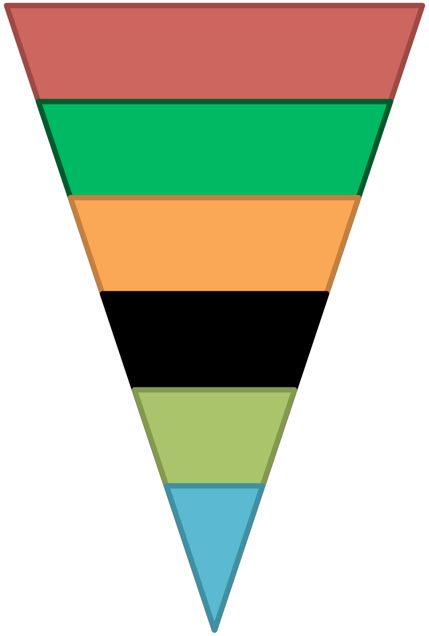


Breaking down a Bank

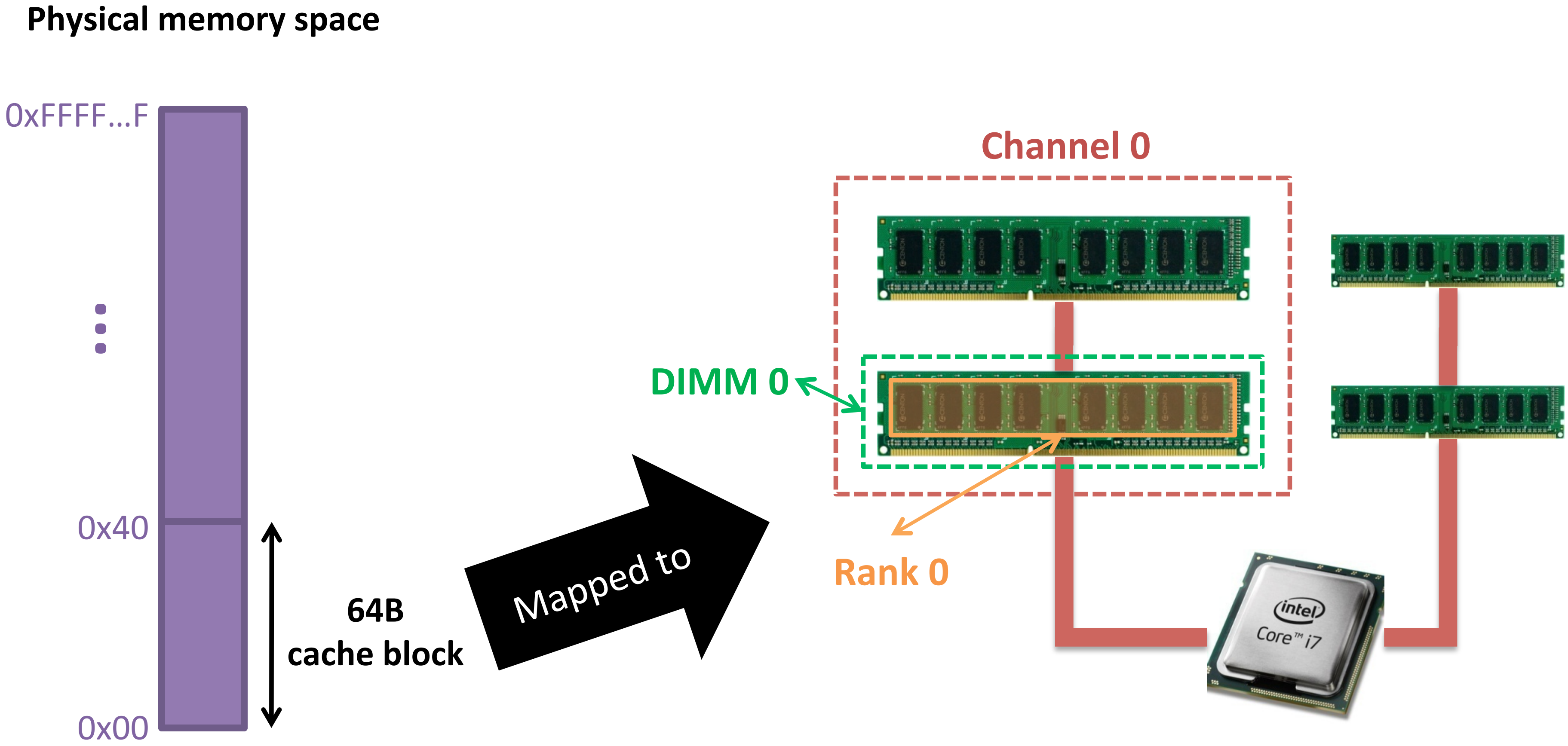


DRAM Subsystem Organization

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

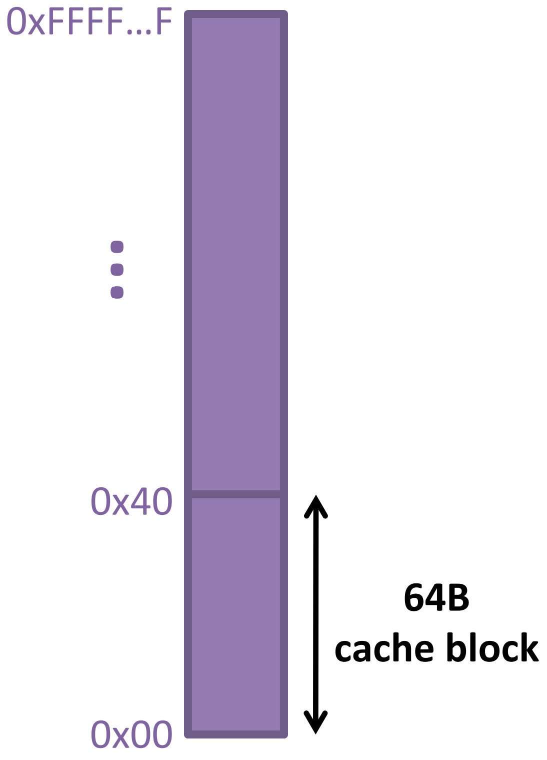


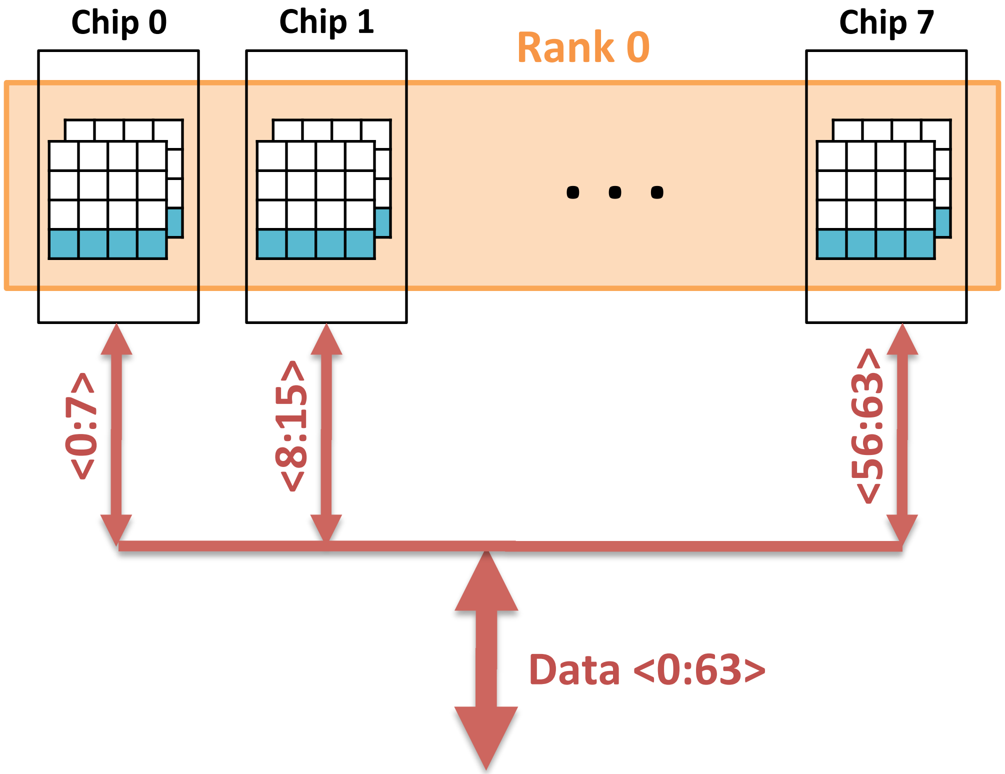
Example: Transferring a cache block



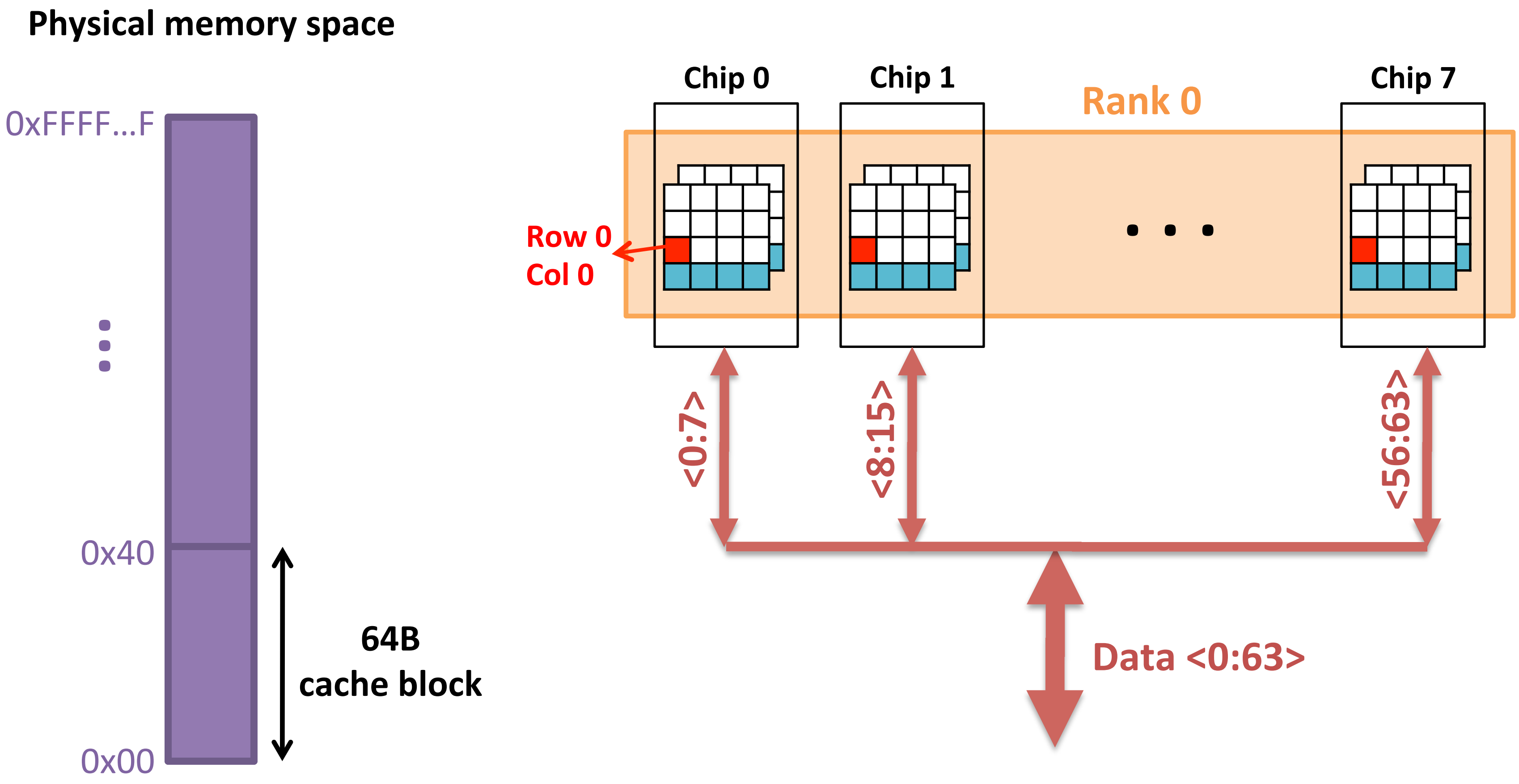
Example: Transferring a cache block

Physical memory space

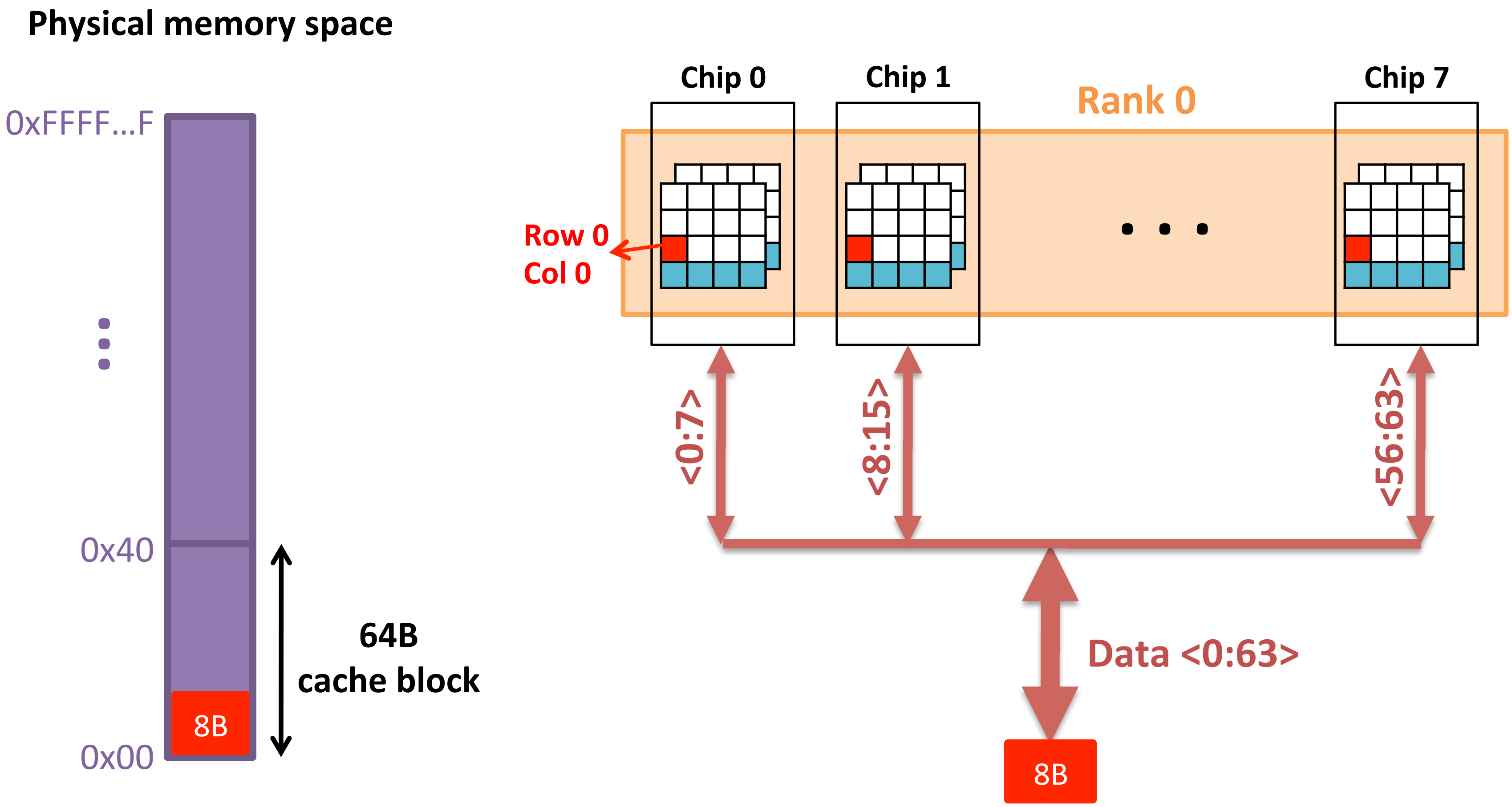




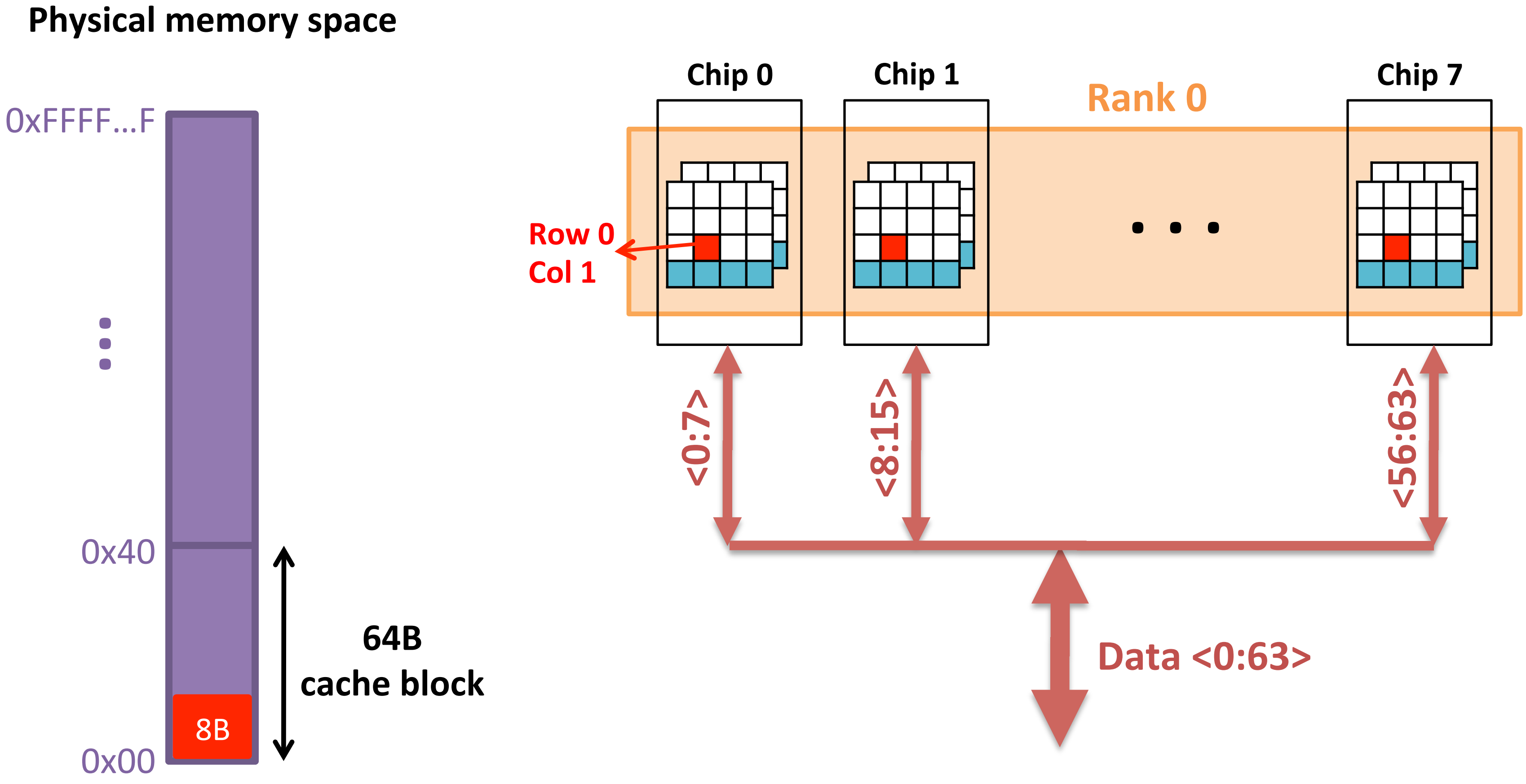
Example: Transferring a cache block



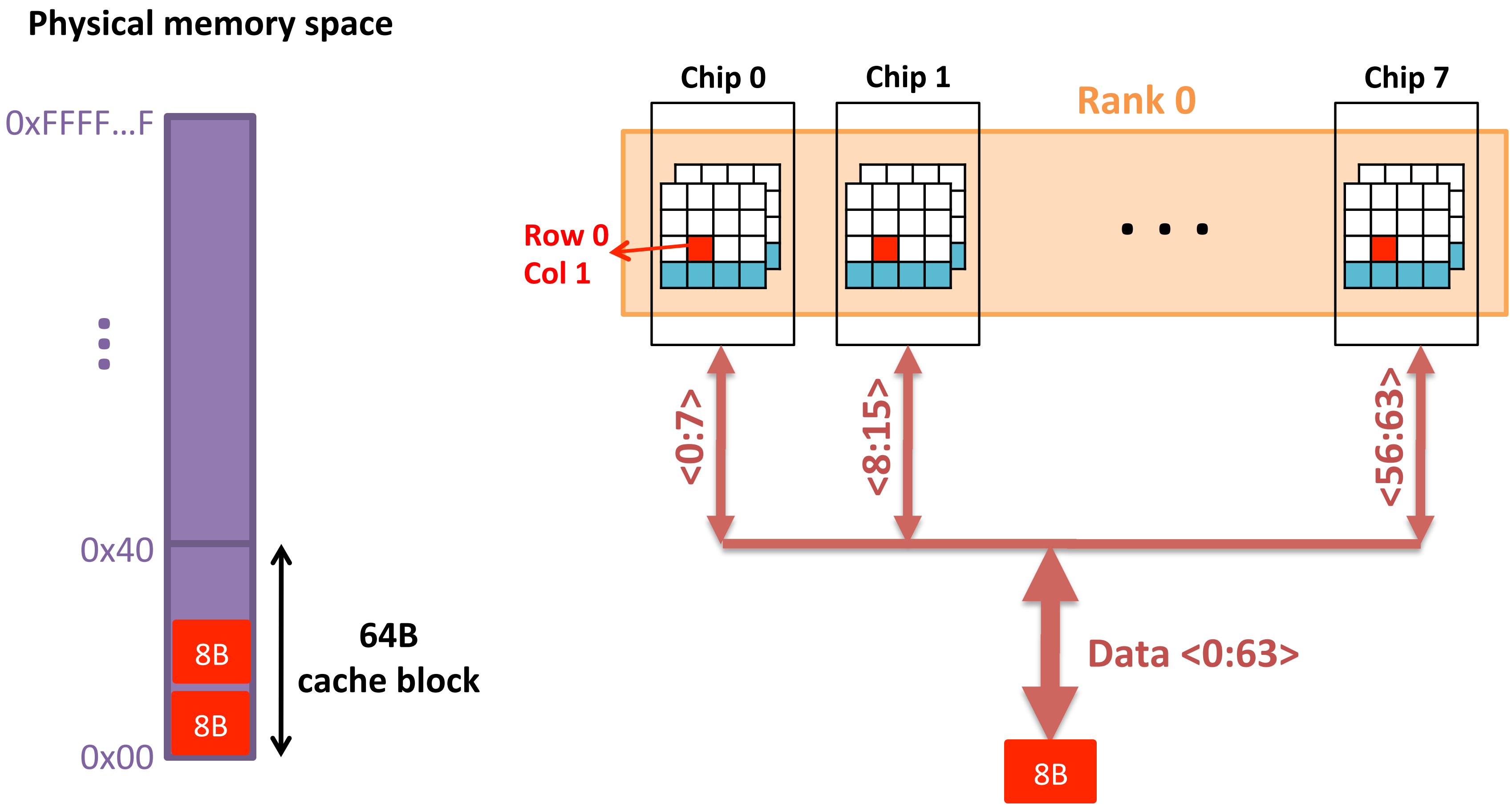
Example: Transferring a cache block



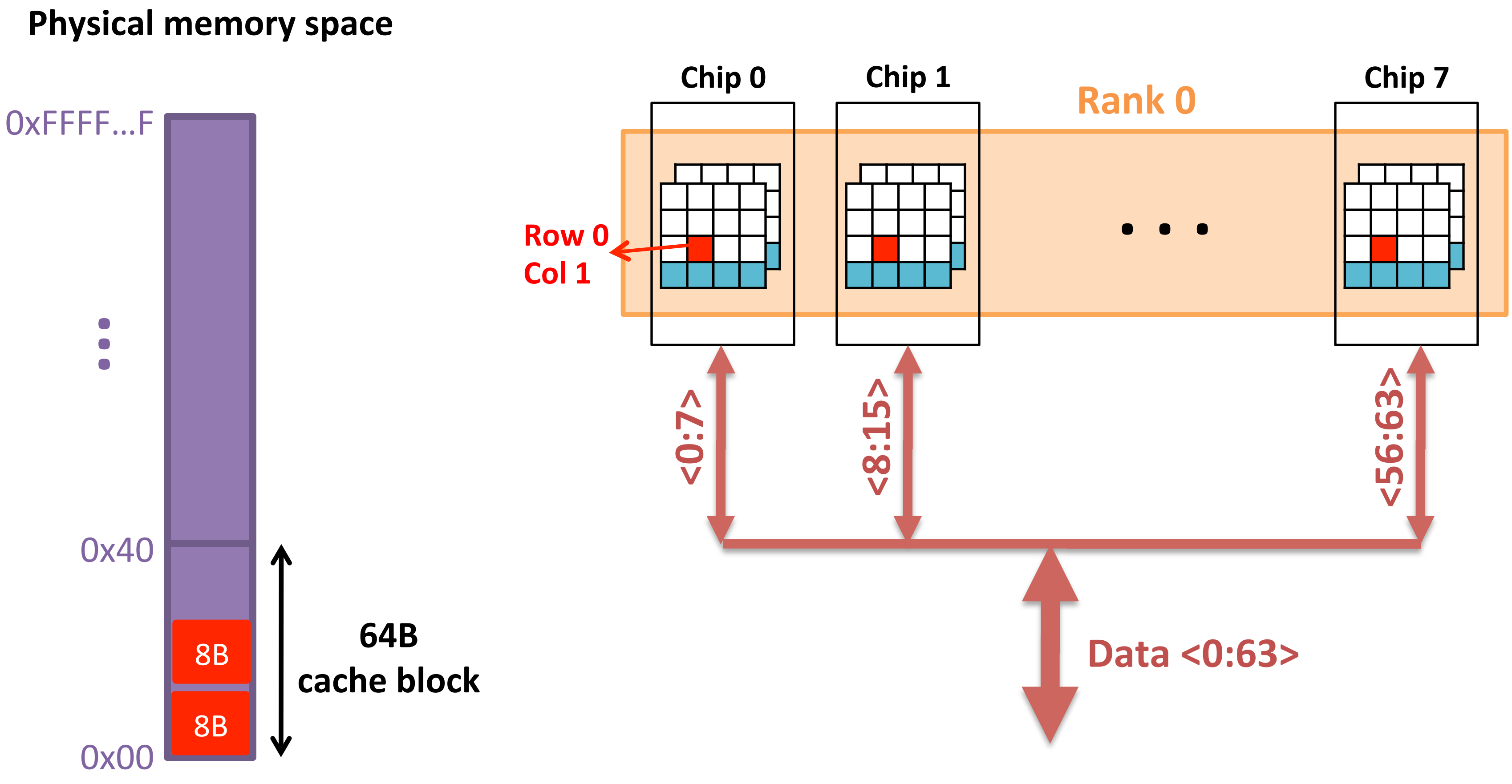
Example: Transferring a cache block



Example: Transferring a cache block



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

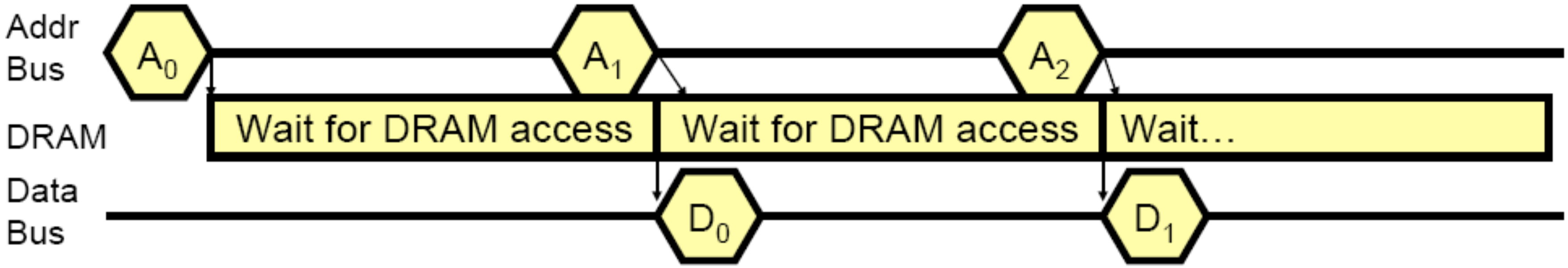
- CPU → controller transfer time
- Controller latency
  - Queuing & scheduling delay at the controller
  - Access converted to basic commands
- Controller → DRAM transfer time
- DRAM bank latency
  - Simple CAS if row is "open" OR
  - RAS + CAS if array precharged OR
  - PRE + RAS + CAS (worst case)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

Multiple Banks (Interleaving) and Channels

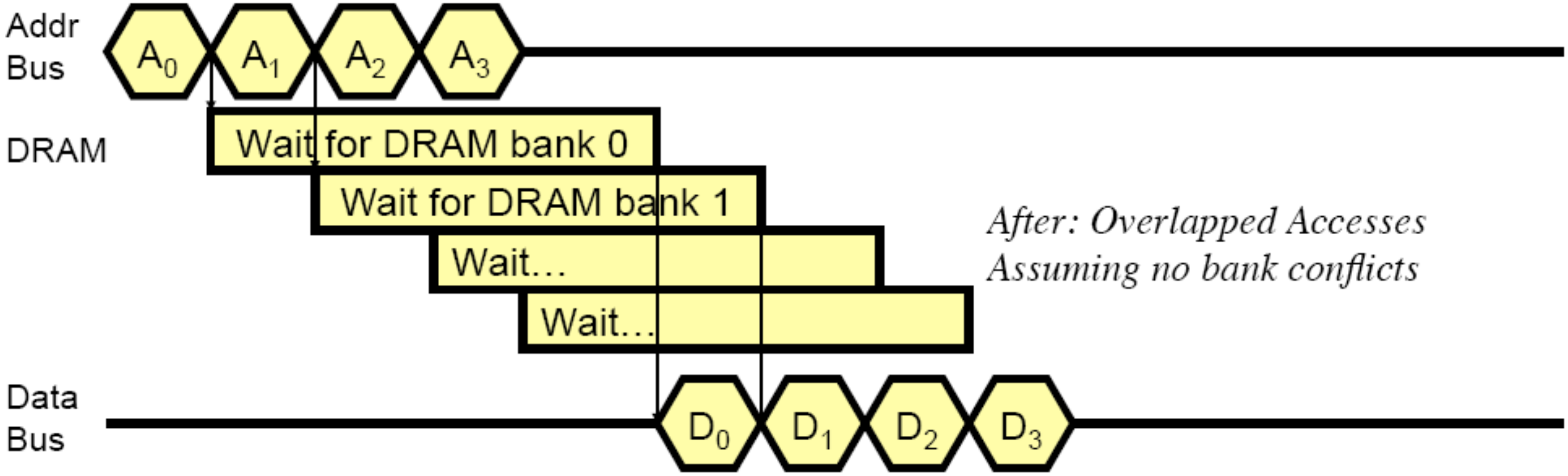
- Multiple banks
  - Enable concurrent DRAM accesses
  - Bits in address determine which bank an address resides in
- Multiple independent channels serve the same purpose
  - But they are even better because they have separate data buses
  - Increased bus bandwidth

- 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/Channels Help



Before: No Overlapping
Assuming accesses to different DRAM rows



After: Overlapped Accesses
Assuming no bank conflicts

Multiple Channels

- Advantages
  - Increased bandwidth
  - Multiple concurrent accesses (if independent channels)

- Disadvantages
  - Higher cost than a single channel
    - More board wires
    - More pins (if on-chip memory controller)

Address Mapping (Single Channel)

- Single-channel system with 8-byte memory bus
  - 2GB memory, 8 banks, 16K rows & 2K columns per bank
- Row interleaving
  - Consecutive rows of memory in consecutive banks

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

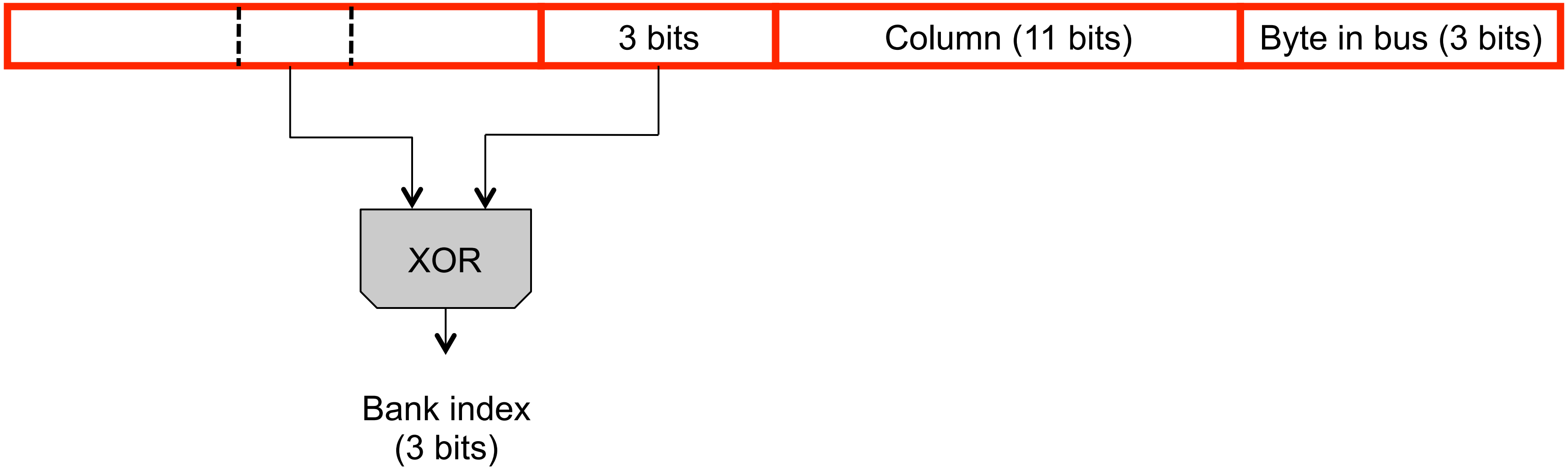
- Cache block interleaving
  - Consecutive cache block addresses in consecutive banks
  - 64 byte cache blocks

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

  - Accesses to consecutive cache blocks can be serviced in parallel
  - How about random accesses? Strided accesses?

Bank Mapping Randomization

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



Address Mapping (Multiple Channels)

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

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

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

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

- Where are consecutive cache blocks?

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

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

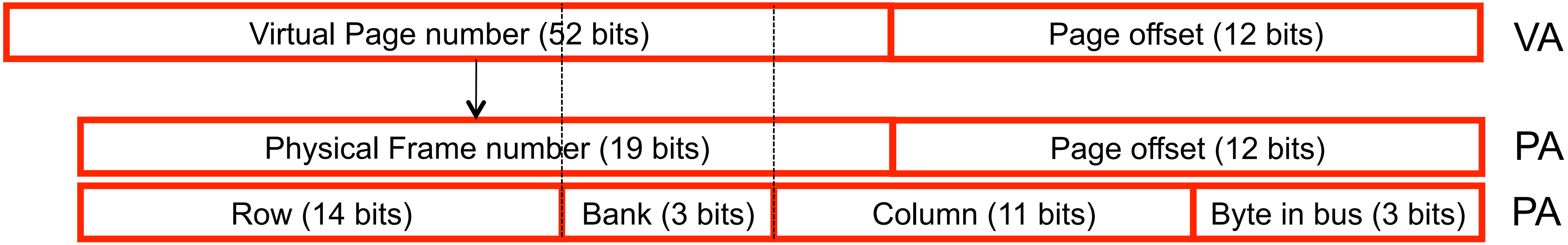
Row (14 bits)	High Column	C	Bank (3 bits)	Low Col.	Byte in bus (3 bits)
	8 bits			3 bits	

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

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

Interaction with Virtual→Physical Mapping

- Operating System influences where an address maps to in DRAM



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

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

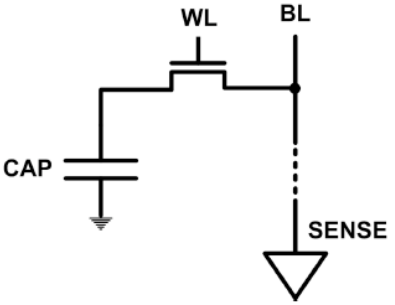
Row (14 bits) | Bank (3 bits) | Column (11 bits) | Byte in bus (3 bits) | PA

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

- It can perform page coloring to minimize bank conflicts
- Or to minimize inter-application interference

DRAM Refresh (I)

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

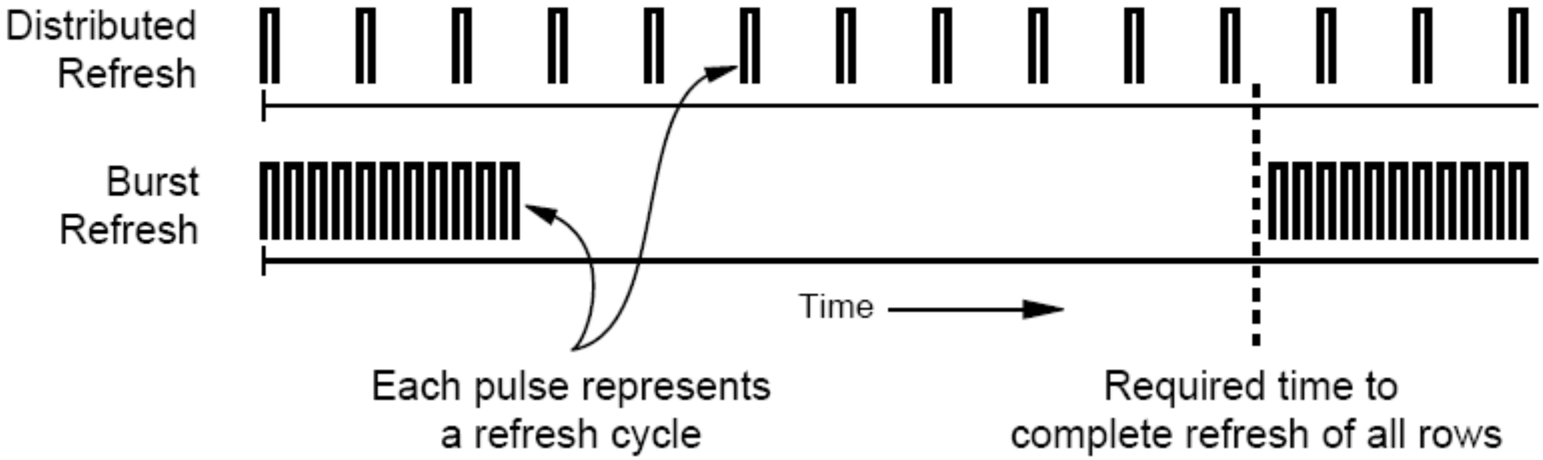


- Implications on performance?

  -- DRAM bank unavailable while refreshed

  -- Long pause times: If we refresh all rows in burst, every 64ms the DRAM will be unavailable until refresh ends
- Burst refresh: All rows refreshed immediately after one another
- Distributed refresh: Each row refreshed at a different time, at regular intervals

DRAM Refresh (II)


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

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

Downsides of DRAM Refresh

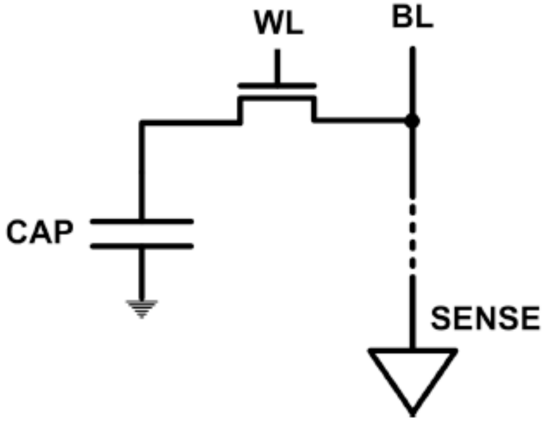
- Downsides of refresh

  -- 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



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 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

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 improve performance
  - 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

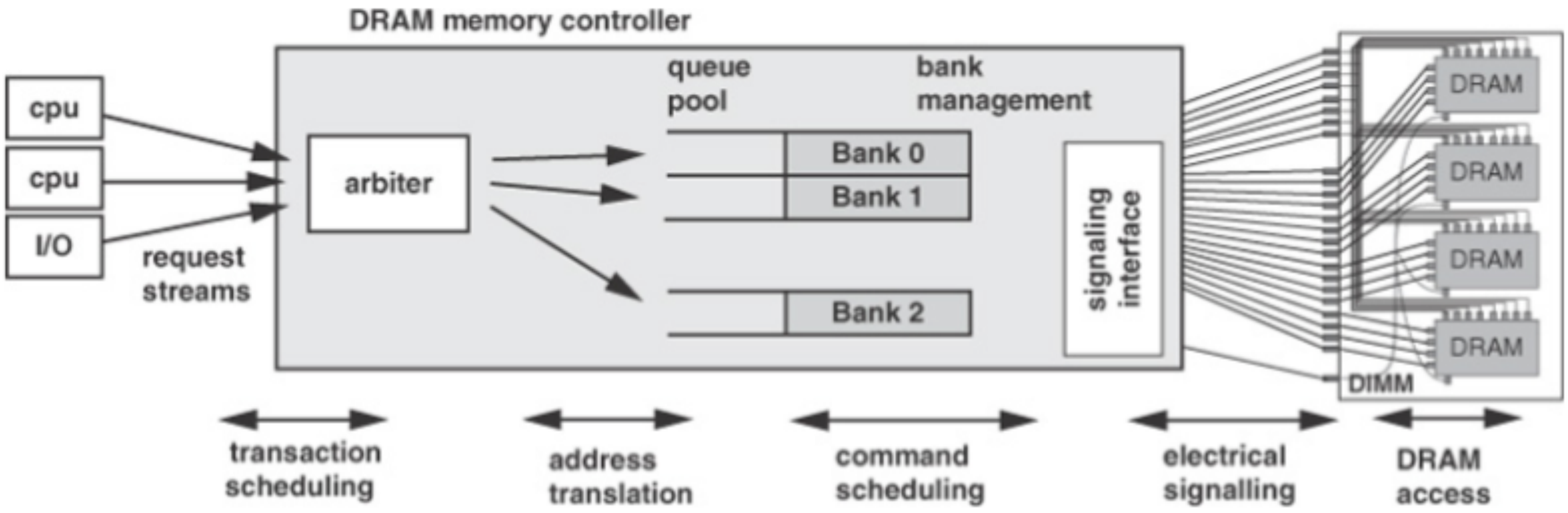
- On CPU chip

  + Reduced latency for main memory access

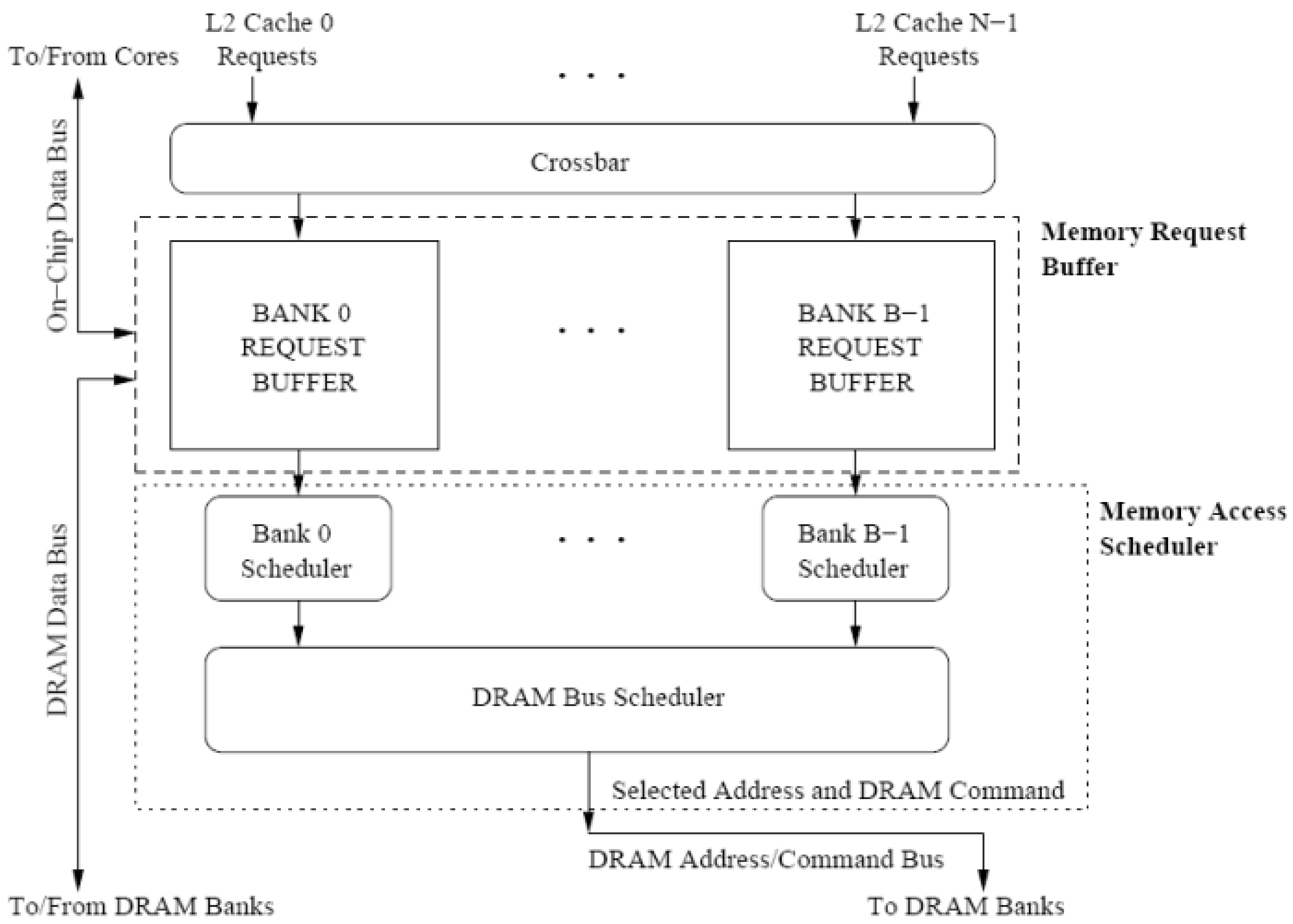
  + Higher bandwidth between cores and controller

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

DRAM Controller (II)



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 essentially a 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?

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

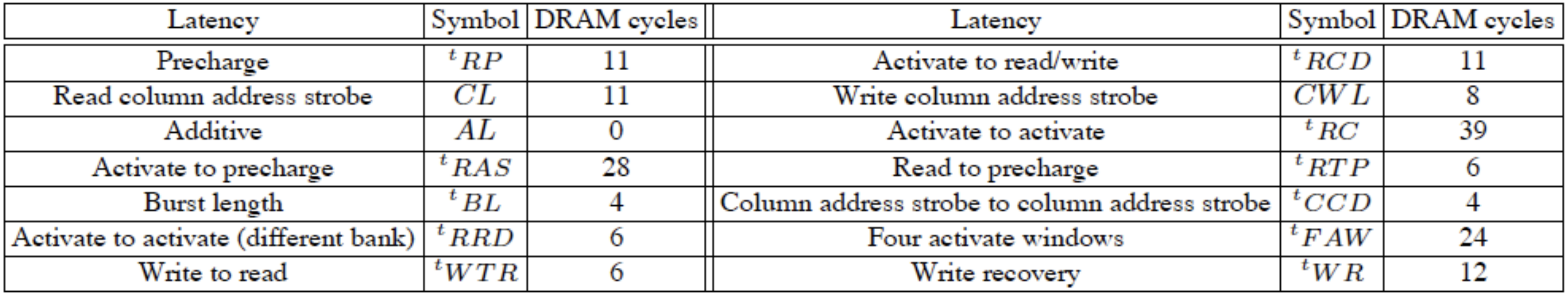
Open vs. Closed Row Policies

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

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 optimize for performance (in the presence of constraints)
  - Reordering is not simple
  - Predicting the future?

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.

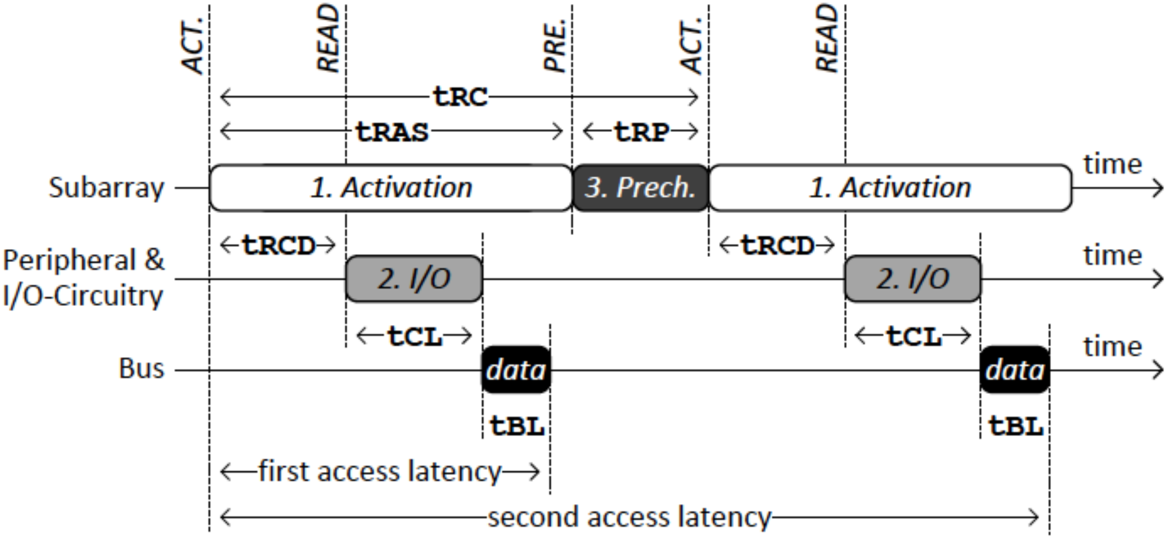


Figure 5. Three Phases of DRAM Access

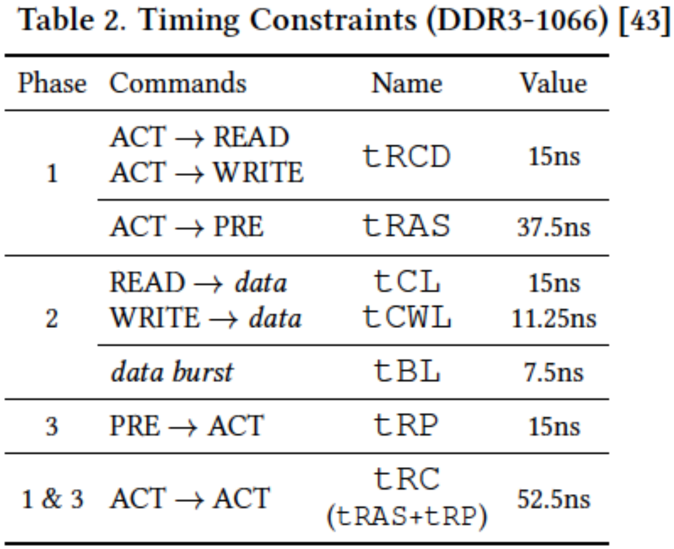


Table 2. Timing Constraints (DDR3-1066) [43]

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

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

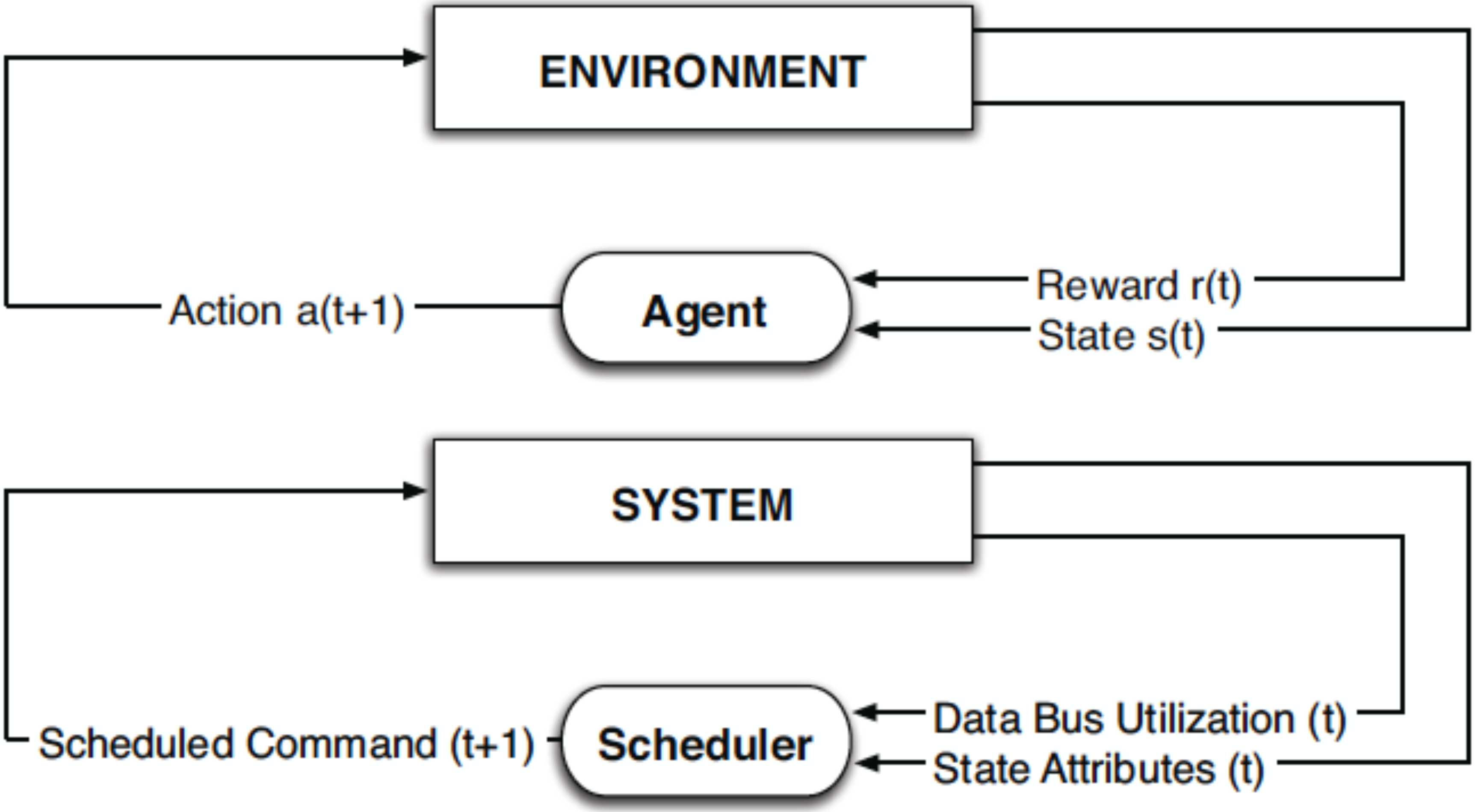


Figure 2: (a) Intelligent agent based on reinforcement learning principles; (b) DRAM scheduler as an RL-agent

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.

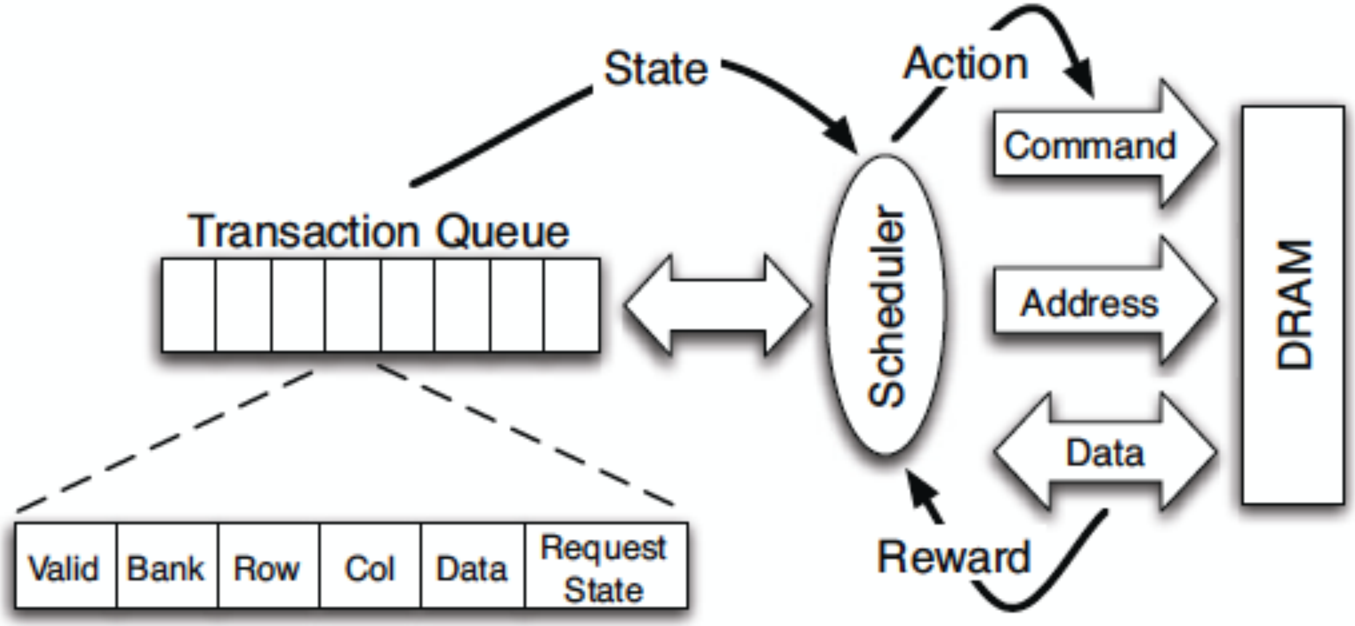


Figure 4: High-level overview of an RL-based scheduler.

Performance Results

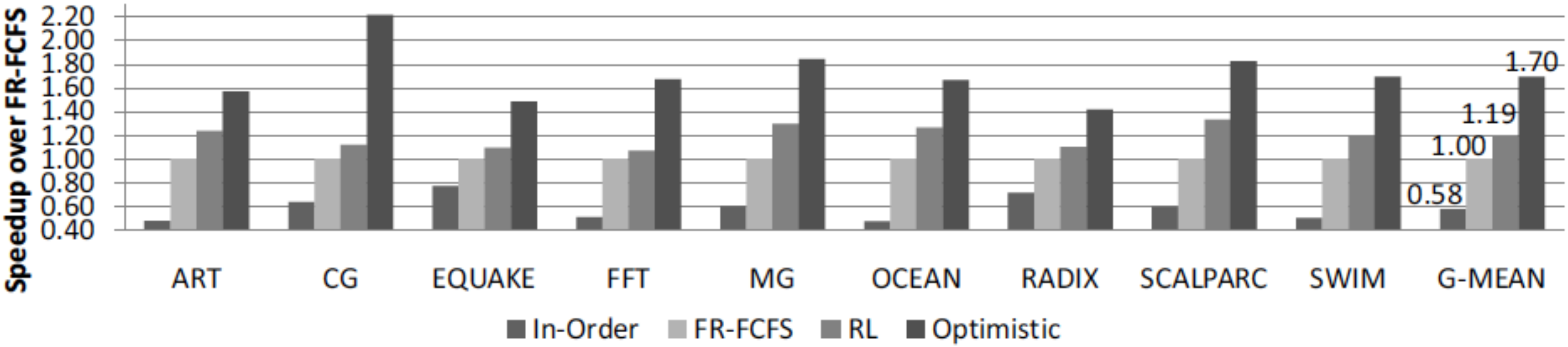


Figure 7: Performance comparison of in-order, FR-FCFS, RL-based, and optimistic memory controllers

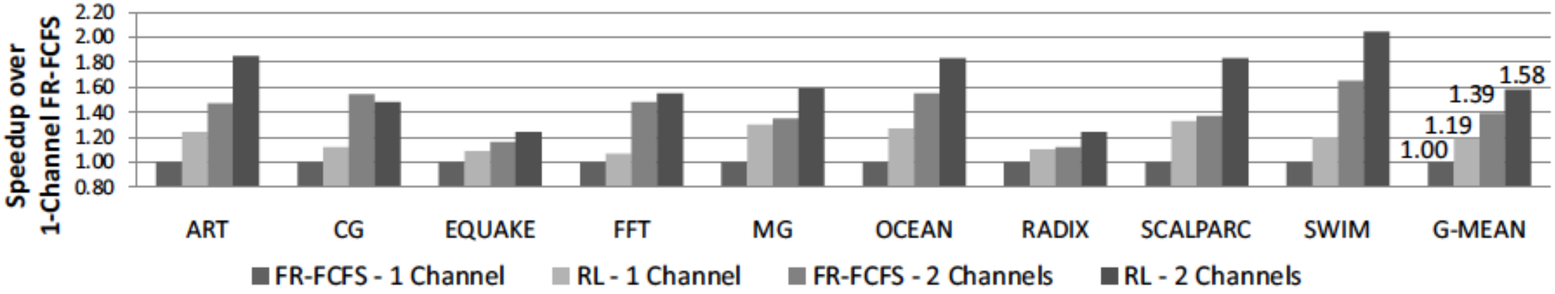


Figure 15: Performance comparison of FR-FCFS and RL-based memory controllers on systems with 6.4GB/s and 12.8GB/s peak DRAM bandwidth

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)

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

Trends Affecting Main Memory

Agenda for Today

- What Will You Learn in This Mini-Lecture Series
- Main Memory Basics (with a Focus on DRAM)
- Major Trends Affecting Main Memory
- DRAM Scaling Problem and Solution Directions
- Solution Direction 1: System-DRAM Co-Design
- Ongoing Research
- Summary

Technology Trends

- DRAM does not scale well beyond N nm [ITRS 2009, 2010]
  - Memory scaling benefits: density, capacity, cost

- Energy/power already key design limiters
  - Memory hierarchy responsible for a large fraction of power
    - IBM servers: ~50% energy spent in off-chip memory hierarchy [Lefurgy+, IEEE Computer 2003]
    - DRAM consumes power when idle and needs periodic refresh

- More transistors (cores) on chip
- Pin bandwidth not increasing as fast as number of transistors
  - Memory is the major shared resource among cores
  - More pressure on the memory hierarchy

Application Trends

- Many different threads/applications/virtual-machines (will) concurrently share the memory system
  - Cloud computing/servers: Many workloads consolidated on-chip to improve efficiency
  - GP-GPU, CPU+GPU, accelerators: Many threads from multiple applications
  - Mobile: Interactive + non-interactive consolidation

- Different applications with different requirements (SLAs)
  - Some applications/threads require performance guarantees
  - Modern hierarchies do not distinguish between applications

- Applications are increasingly data intensive
  - More demand for memory capacity and bandwidth

Architecture/System Trends

- Sharing of memory hierarchy

- More cores and components
  - More capacity and bandwidth demand from memory hierarchy
- Asymmetric cores: Performance asymmetry, CPU+GPUs, accelerators, …
  - Motivated by energy efficiency and Amdahl’s Law
- Different cores have different performance requirements
  - Memory hierarchies do not distinguish between cores

- Different goals for different systems/users
  - System throughput, fairness, per-application performance
  - Modern hierarchies are not flexible/configurable

Summary: Major Trends Affecting Memory

- Need for memory capacity and bandwidth increasing
- New need for handling inter-core interference; providing fairness, QoS, predictability
- Need for memory system flexibility increasing
- Memory energy/power is a key system design concern
- DRAM capacity, cost, energy are not scaling well

Requirements from an Ideal Memory System

- Traditional
  - High system performance
  - Enough capacity
  - Low cost

- New
  - Technology scalability
  - QoS and predictable performance
  - Energy (and power, bandwidth) efficiency

Requirements from an Ideal Memory System

- Traditional
  - High system performance: More parallelism, less interference
  - Enough capacity: New technologies and waste management
  - Low cost: New technologies and scaling DRAM

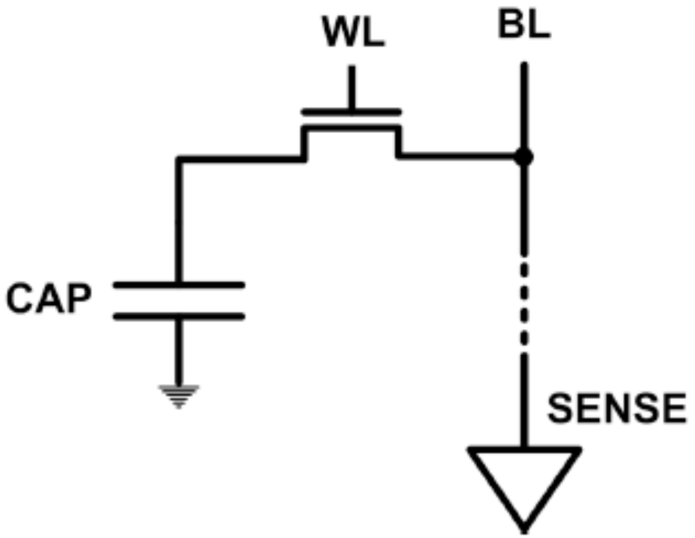
- New
  - Technology scalability
    - New memory technologies can help? DRAM can scale?
  - QoS and predictable performance
    - Hardware mechanisms to control interference and build QoS policies
  - Energy (and power, bandwidth) efficiency
    - Need to reduce waste and enable configurability

Agenda for Today

- What Will You Learn in This Mini-Lecture Series
- Main Memory Basics (with a Focus on DRAM)
- Major Trends Affecting Main Memory
- DRAM Scaling Problem and Solution Directions
- Solution Direction 1: System-DRAM Co-Design
- Ongoing Research
- Summary

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

Solutions to the DRAM Scaling Problem

- Two potential solutions
  - Tolerate DRAM (by taking a fresh look at it)
  - Enable emerging memory technologies to eliminate/minimize DRAM

- Do both
  - Hybrid memory systems

Solution 1: Tolerate DRAM

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

- Key issues to tackle
  - Reduce refresh energy
  - Improve bandwidth and latency
  - Reduce waste
  - Enable reliability at low cost

- Liu, Jaiyen, Veras, Mutlu, "RAIDR: Retention-Aware Intelligent DRAM Refresh," ISCA 2012.
- Kim, Seshadri, Lee+, "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'13.

Tolerating DRAM: System-DRAM Co-Design

New DRAM Architectures

- RAIDR: Reducing Refresh Impact
- TL-DRAM: Reducing DRAM Latency
- SALP: Reducing Bank Conflict Impact
- RowClone: Fast Bulk Data Copy and Initialization

RAIDR: Reducing DRAM Refresh Impact

DRAM Refresh

- DRAM capacitor charge leaks over time

- The memory controller needs to refresh each row periodically to restore charge
  - Activate + precharge each row every N ms
  - Typical N = 64 ms

- Downsides of refresh

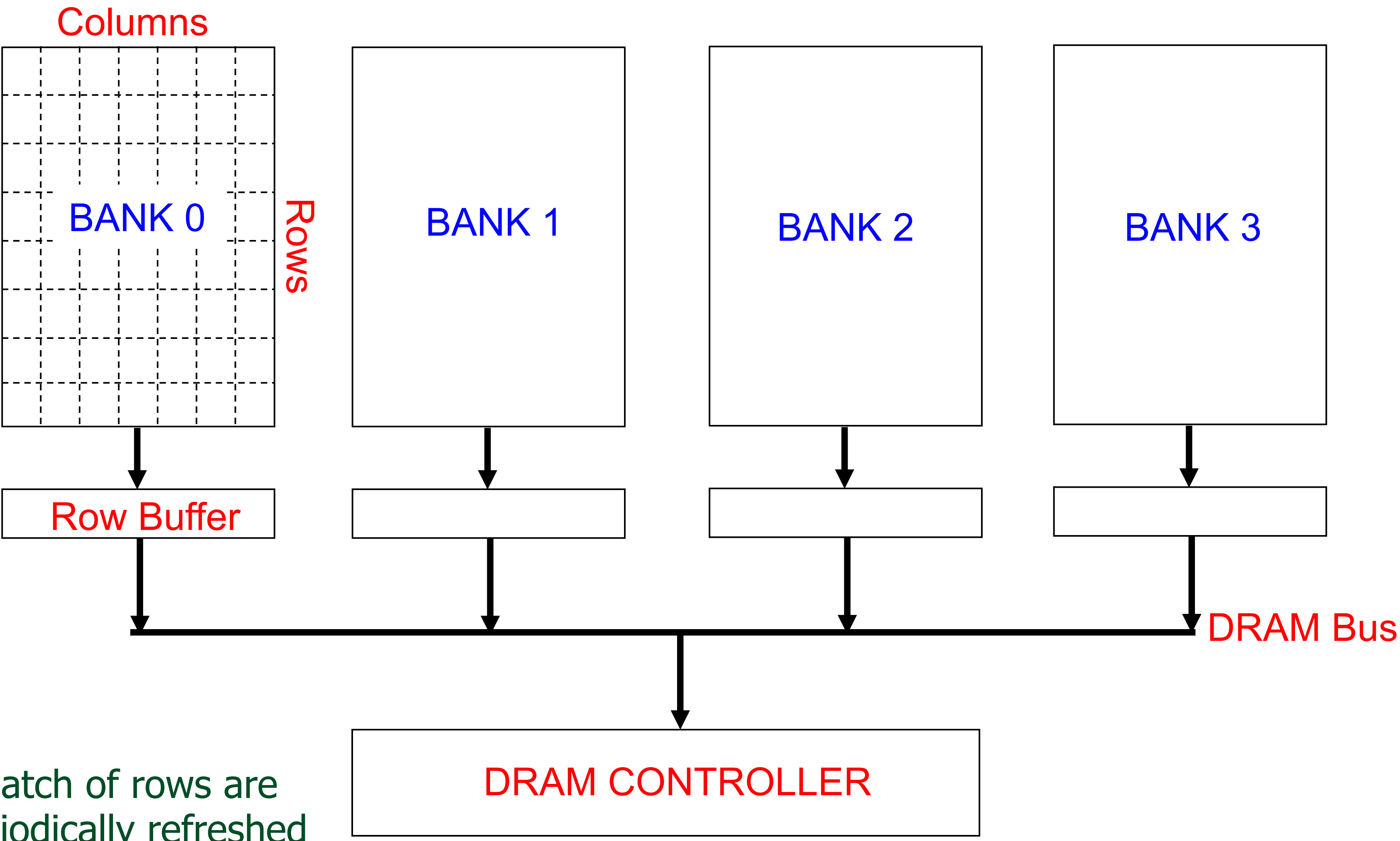
  -- Energy consumption: Each refresh consumes energy

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

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

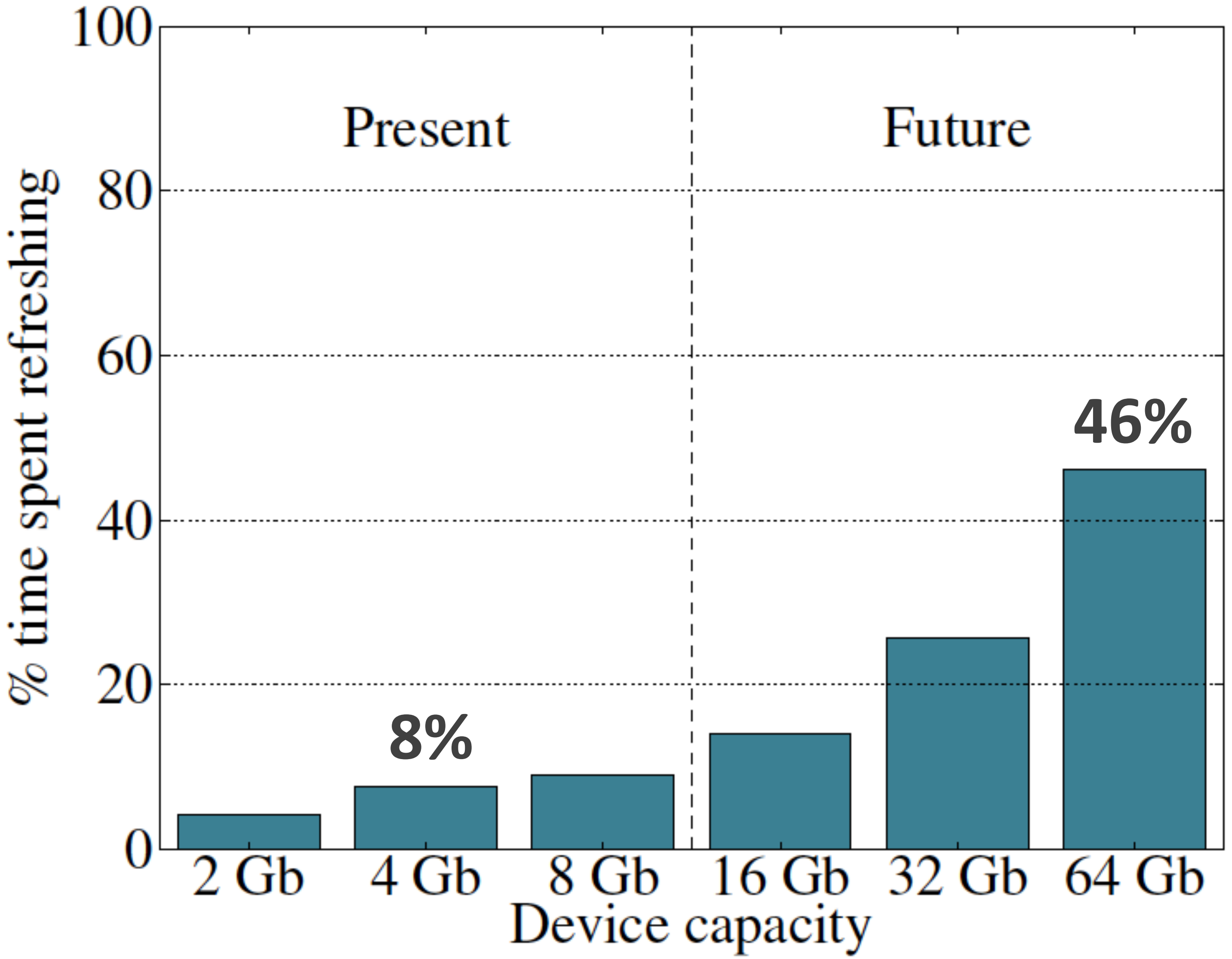
  -- Refresh rate limits DRAM density scaling

Refresh Today: Auto Refresh

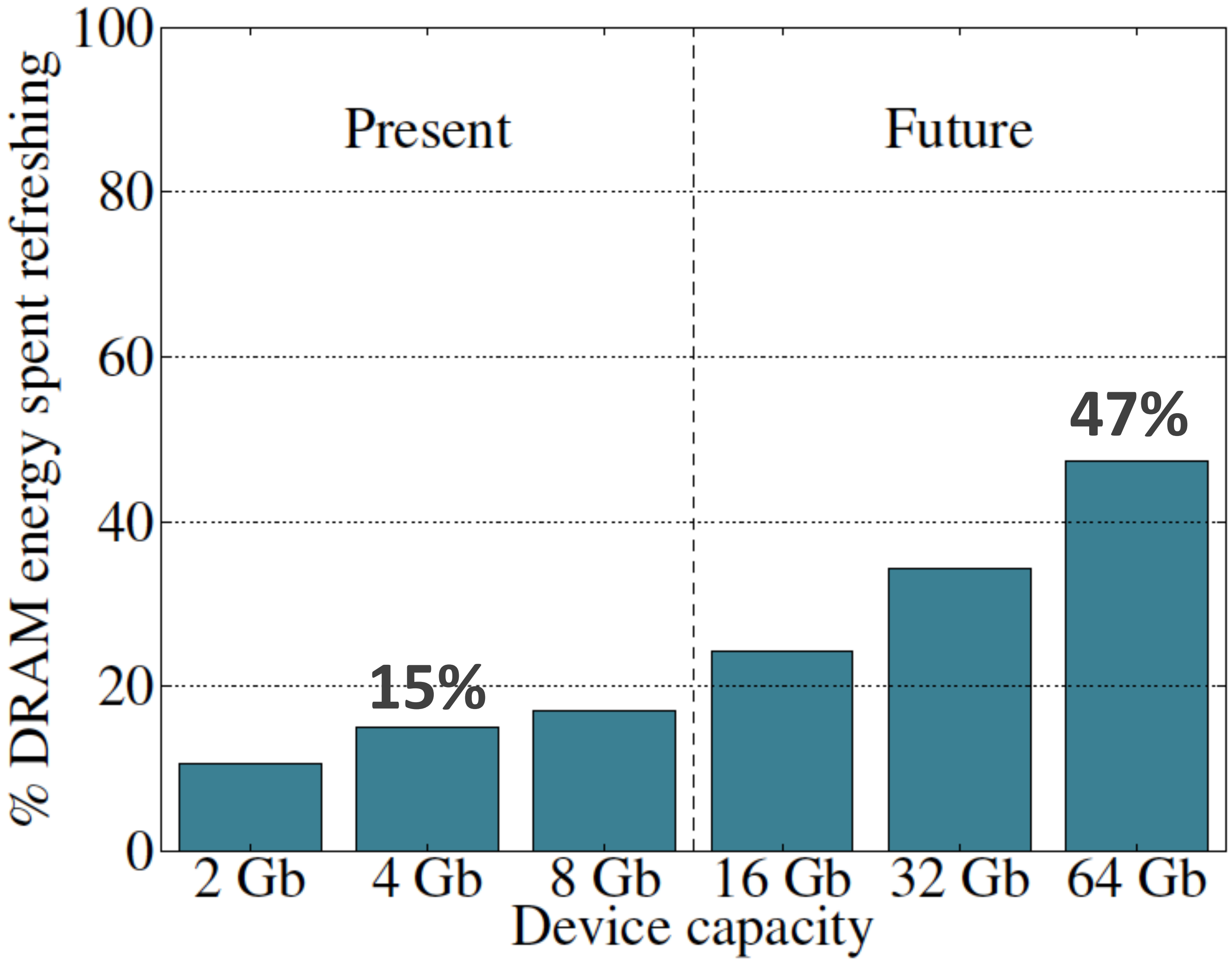


A batch of rows are periodically refreshed via the auto-refresh command

Refresh Overhead: Performance

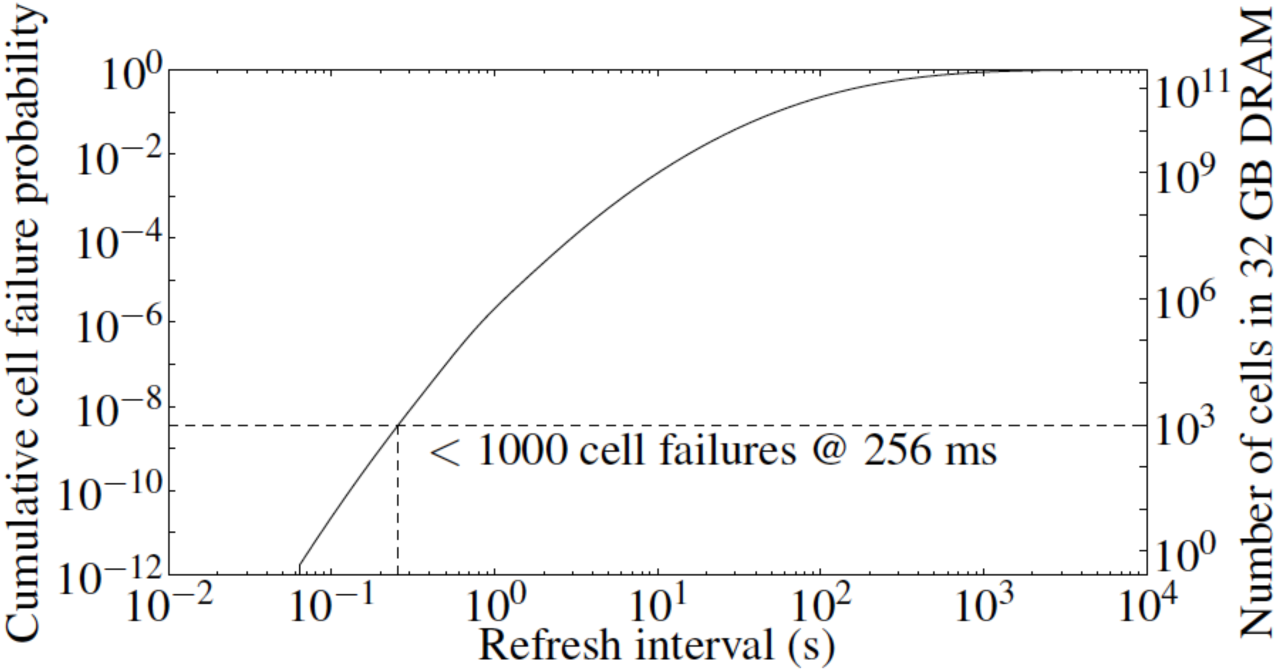


Refresh Overhead: Energy



Problem with Conventional Refresh

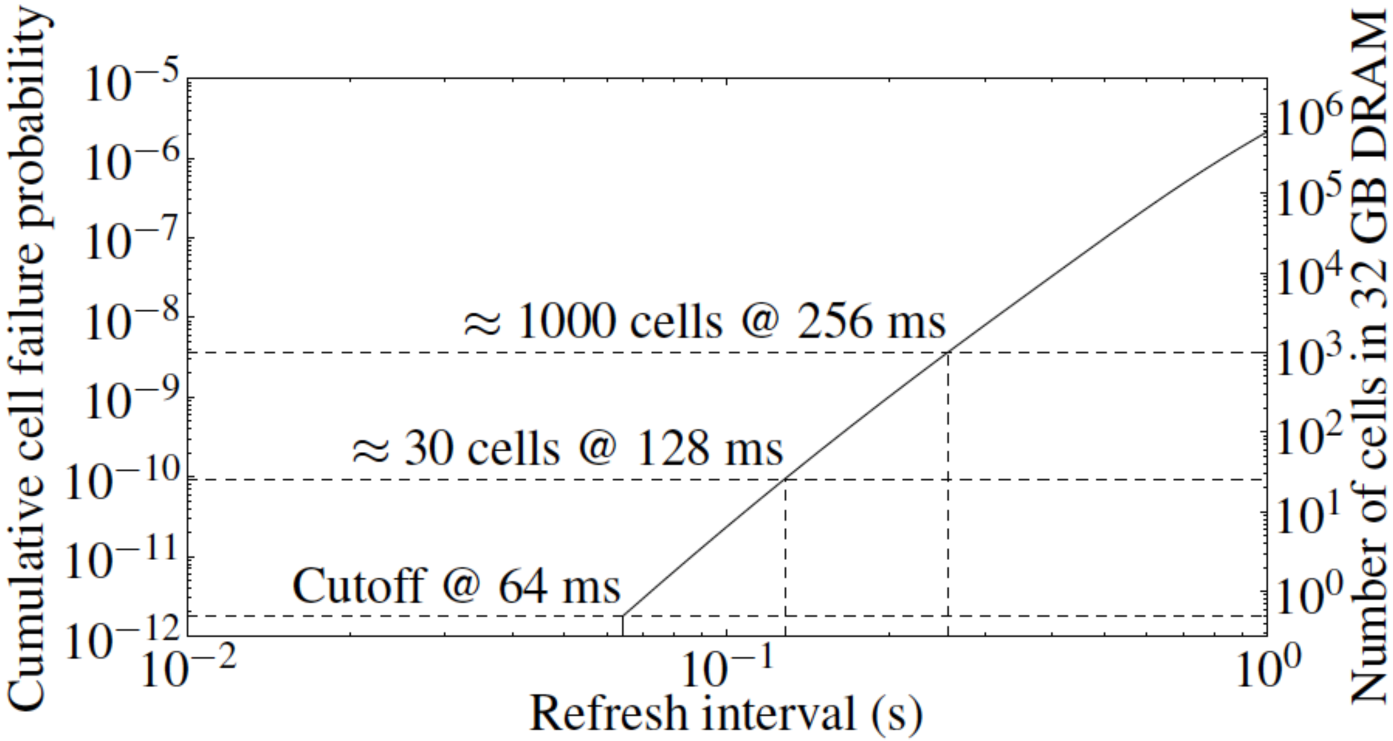
- Today: Every row is refreshed at the same rate



- Observation: Most rows can be refreshed much less often without losing data [Kim+, EDL'09]
- Problem: No support in DRAM for different refresh rates per row

Retention Time of DRAM Rows

- Observation: Only very few rows need to be refreshed at the worst-case rate



- Can we exploit this to reduce refresh operations at low cost?

Reducing DRAM Refresh Operations

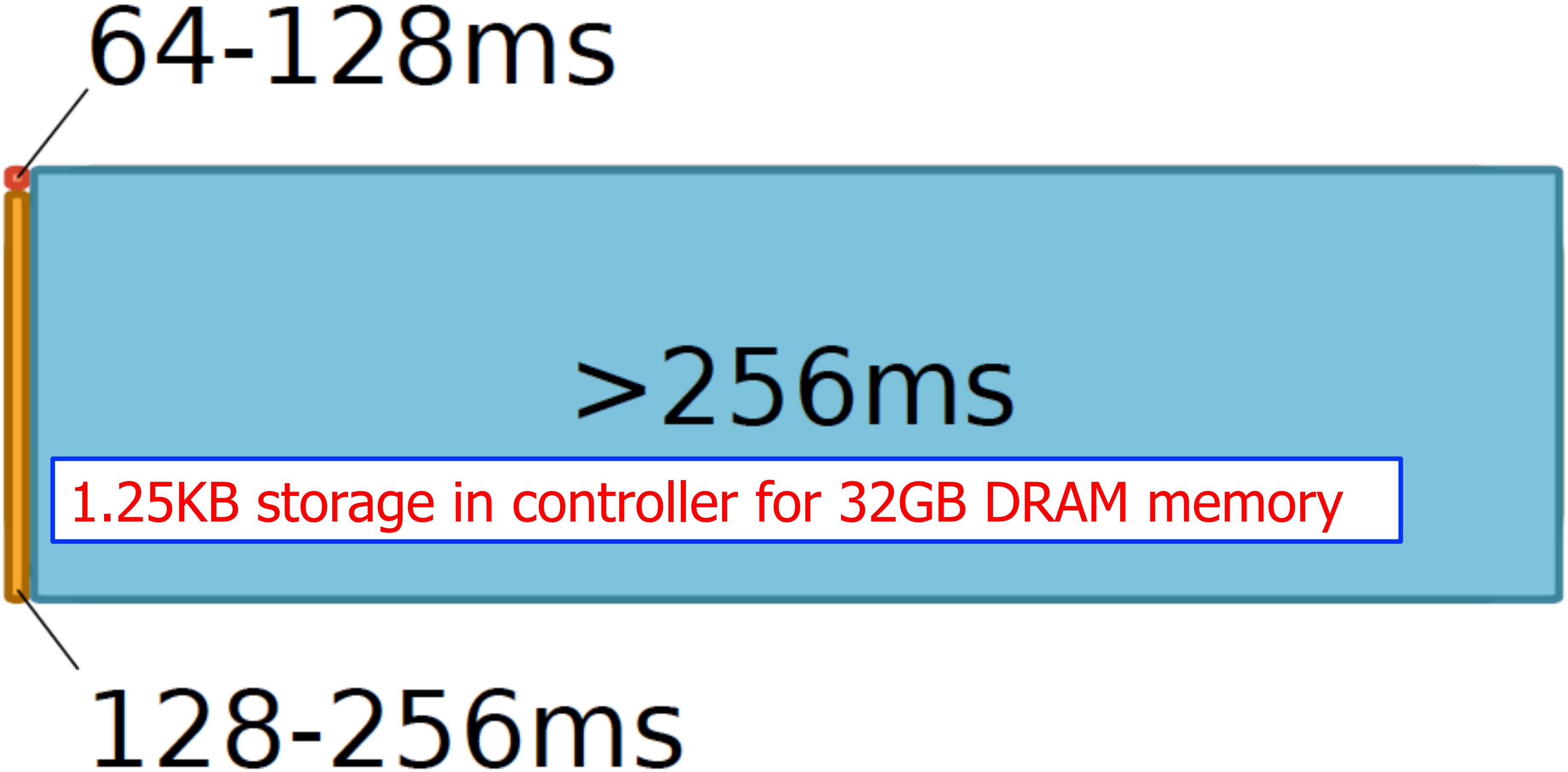
- Idea: Identify the retention time of different rows and refresh each row at the frequency it needs to be refreshed

- (Cost-conscious) Idea: Bin the rows according to their minimum retention times and refresh rows in each bin at the refresh rate specified for the bin
  - e.g., a bin for 64-128ms, another for 128-256ms, …

- Observation: Only very few rows need to be refreshed very frequently [64-128ms] → Have only a few bins → Low HW overhead to achieve large reductions in refresh operations

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

RAIDR: Mechanism



1.25KB storage in controller for 32GB DRAM memory

bins at different rates

→ probe Bloom Filters to determine refresh rate of a row

1. Profiling

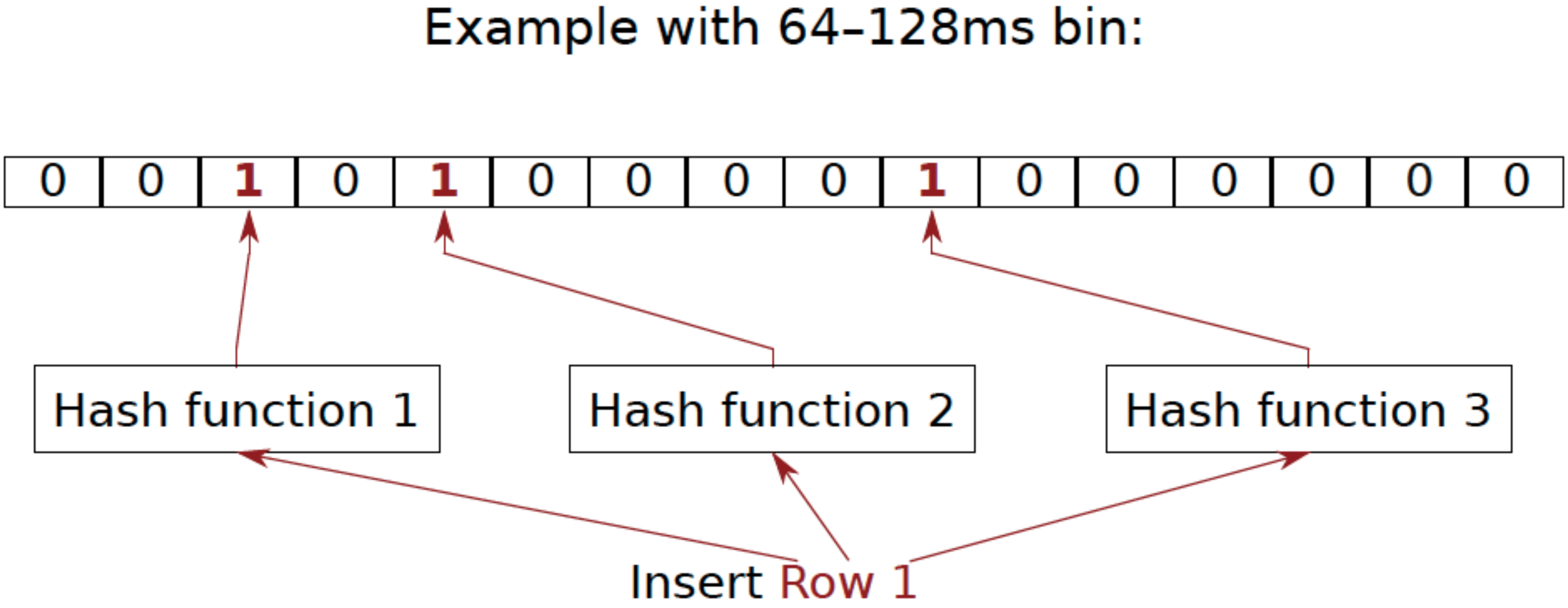
To profile a row:

1. Write data to the row
2. Prevent it from being refreshed
3. Measure time before data corruption

	Row 1	Row 2	Row 3
Initially	11111111...	11111111...	11111111...
After 64 ms	11111111...	11111111...	11111111...
After 128 ms	11011111... (64–128ms)	11111111...	11111111...
After 256 ms		11111011... (128–256ms)	11111111... (>256ms)

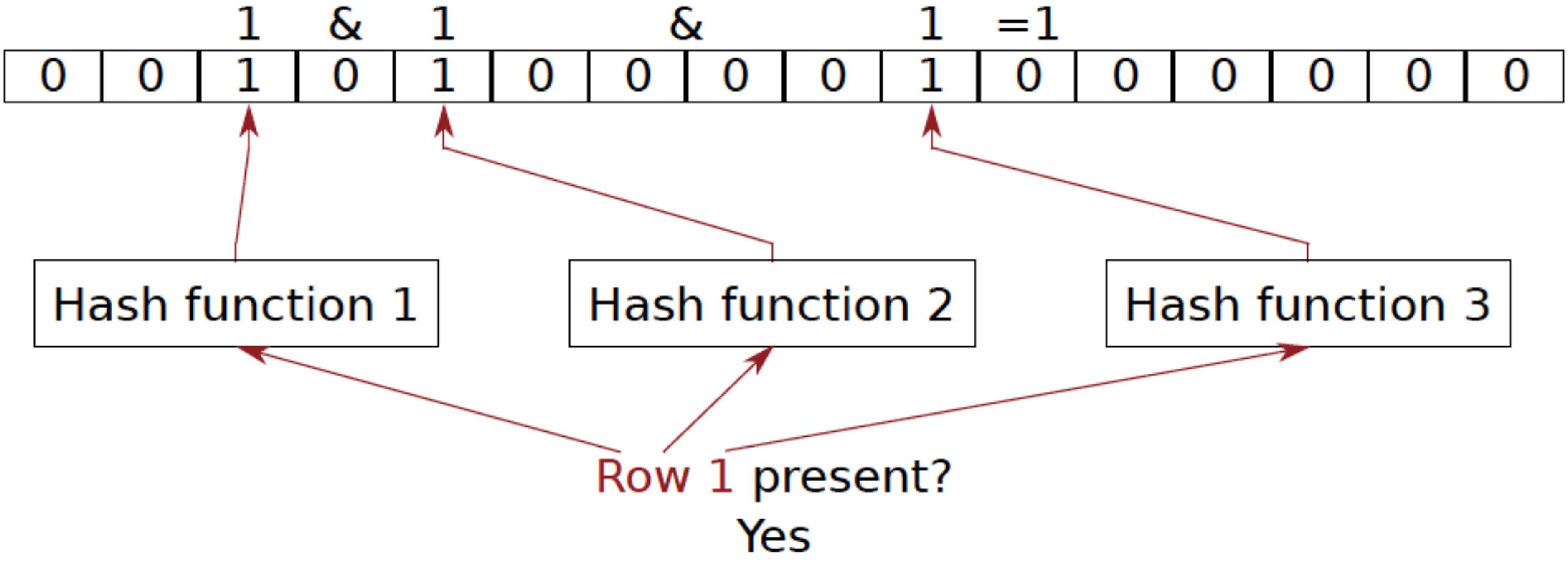
2. Binning

- How to efficiently and scalably store rows into retention time bins?
- Use Hardware Bloom Filters [Bloom, CACM 1970]



Bloom Filter Operation Example

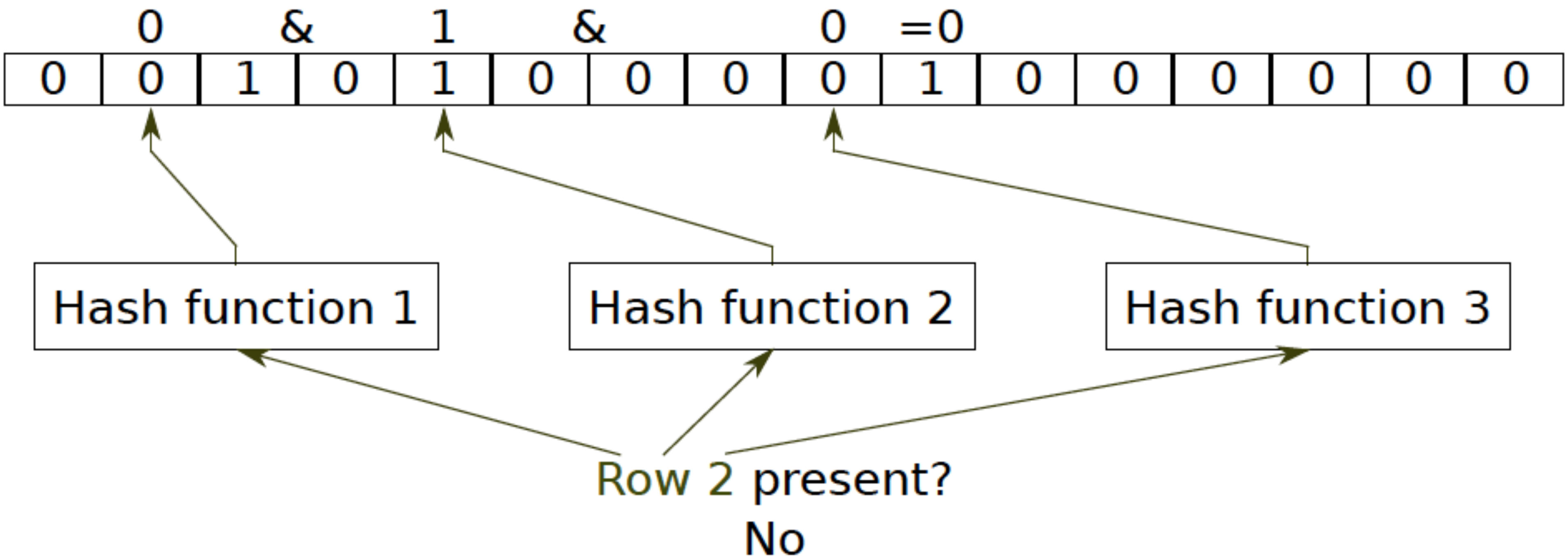
Example with 64–128ms bin:



0	0	1	0	1	0	0	0	0	1	0	0	0	0	0	0

Bloom Filter Operation Example

Example with 64–128ms bin:



0 & 1 & 0 =0

0	0	1	0	1	0	0	0	0	1	0	0	0	0	0	0

Hash function 1

Hash function 2

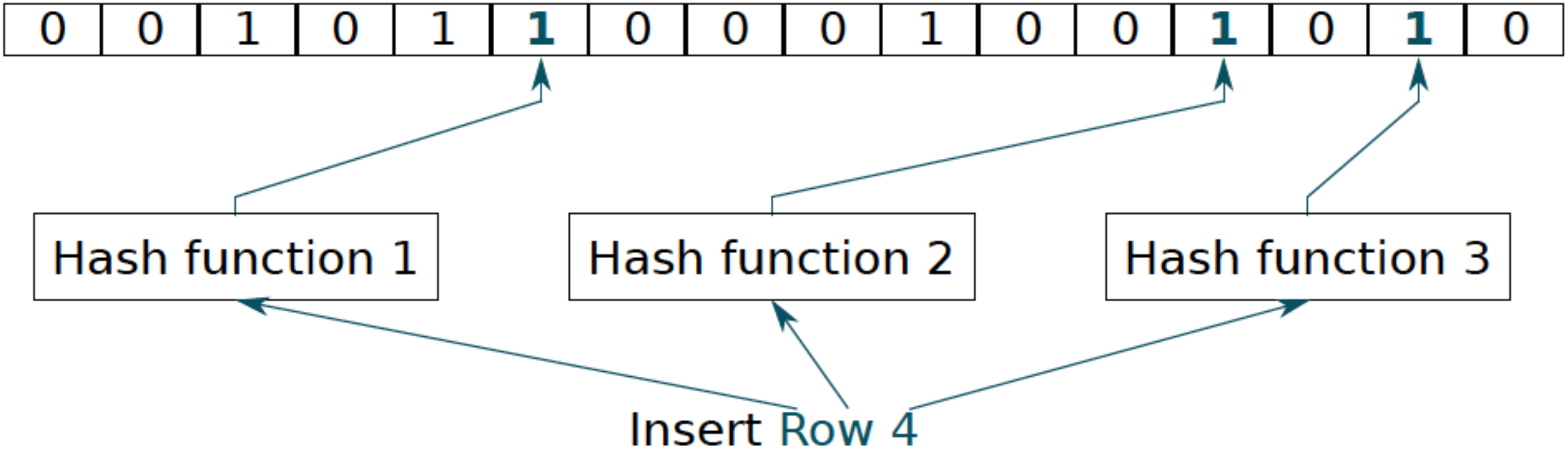
Hash function 3

Row 2 present?

No

Bloom Filter Operation Example

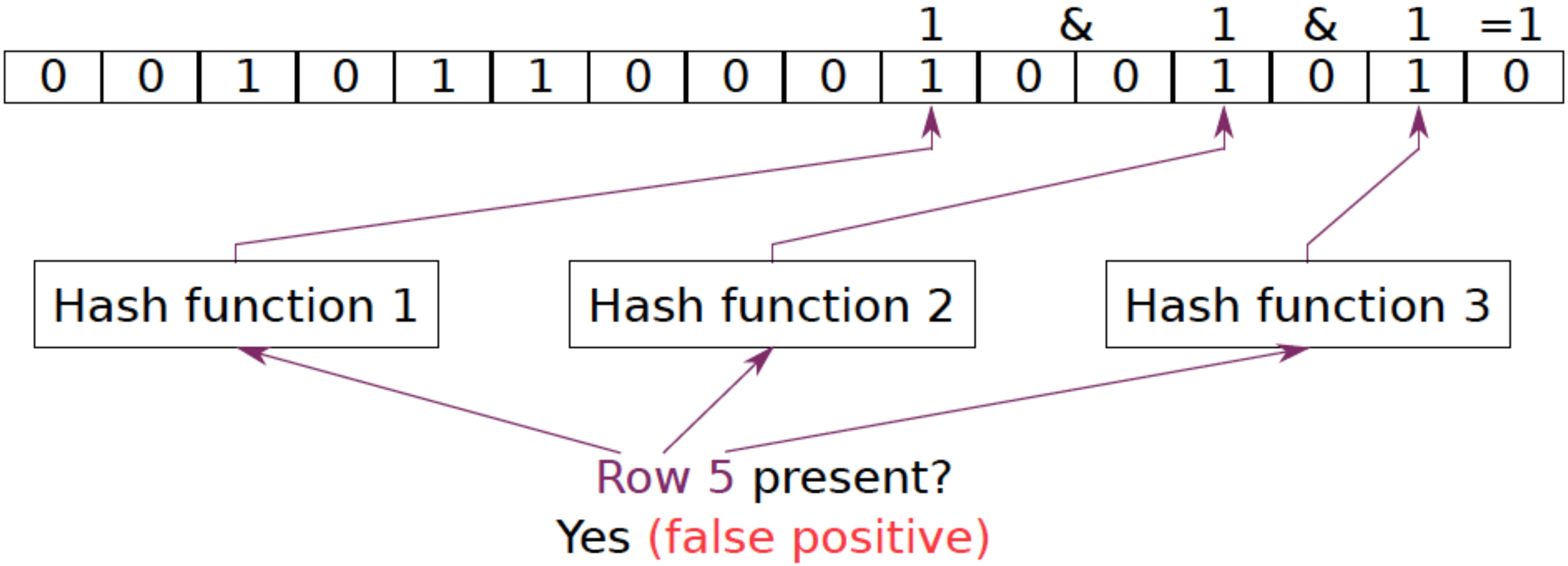
Example with 64–128ms bin:



0	0	1	0	1	**1**	0	0	0	1	0	0	**1**	0	**1**	0

Bloom Filter Operation Example

Example with 64–128ms bin:



Row 5 present?
Yes (false positive)

Benefits of Bloom Filters as Bins

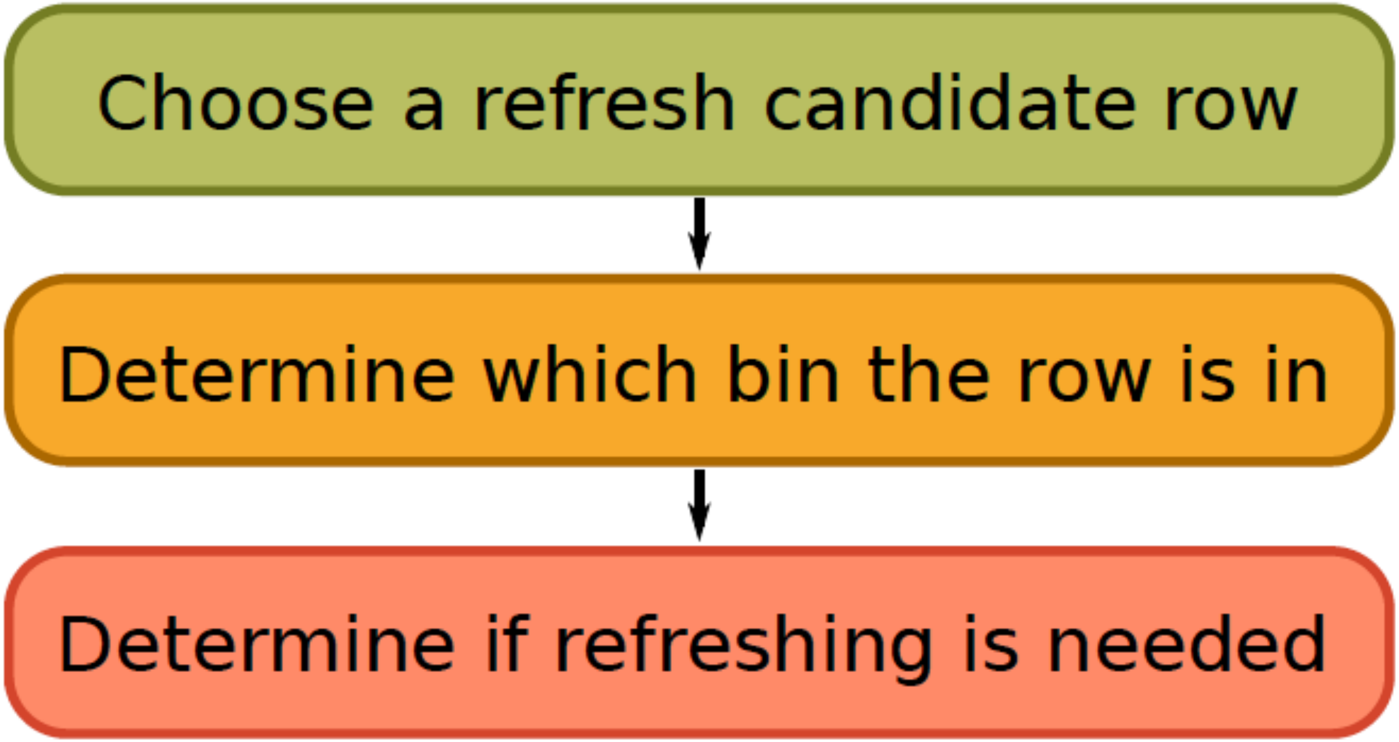
- False positives: a row may be declared present in the Bloom filter even if it was never inserted
  - Not a problem: Refresh some rows more frequently than needed

- No false negatives: rows are never refreshed less frequently than needed (no correctness problems)

- Scalable: a Bloom filter never overflows (unlike a fixed-size table)

- Efficient: No need to store info on a per-row basis; simple hardware → 1.25 KB for 2 filters for 32 GB DRAM system

3. Refreshing (RAIDR Refresh Controller)

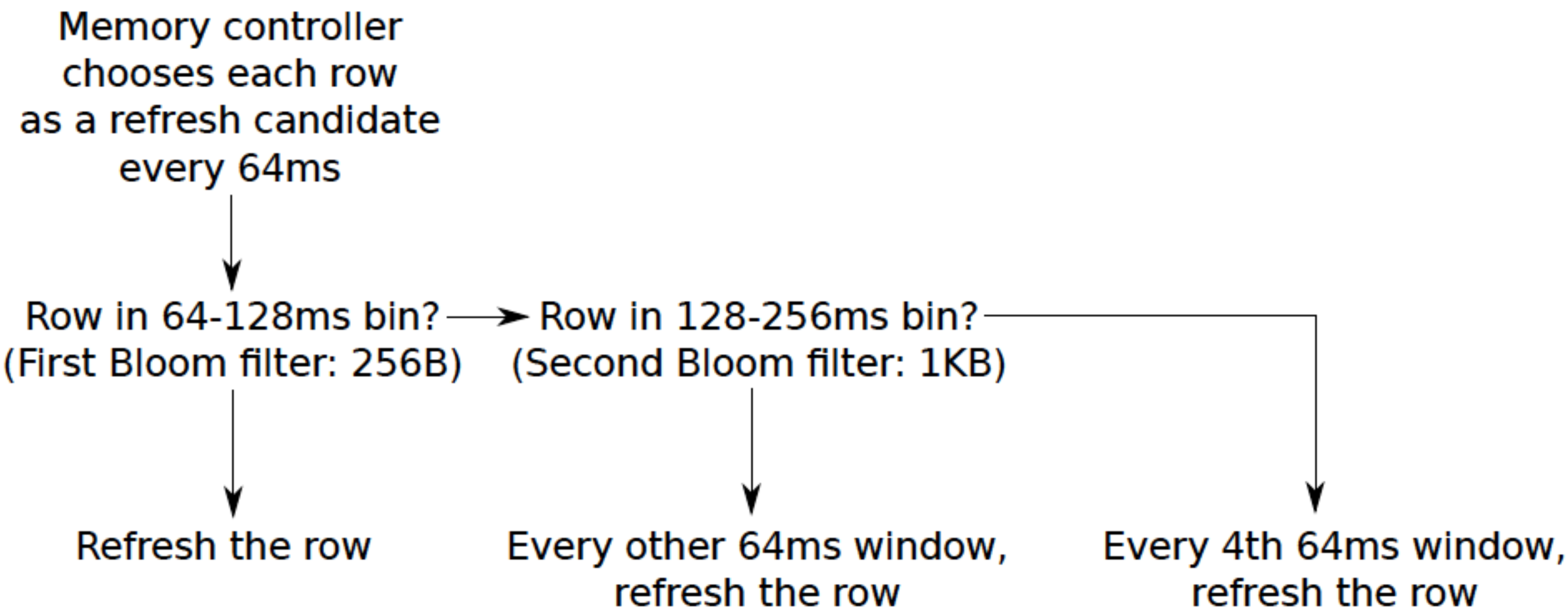


Choose a refresh candidate row

Determine which bin the row is in

Determine if refreshing is needed

3. Refreshing (RAIDR Refresh Controller)


Memory controller chooses each row as a refresh candidate every 64ms

Row in 64-128ms bin? (First Bloom filter: 256B)

Row in 128-256ms bin? (Second Bloom filter: 1KB)

Refresh the row

Every other 64ms window, refresh the row

Every 4th 64ms window, refresh the row

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

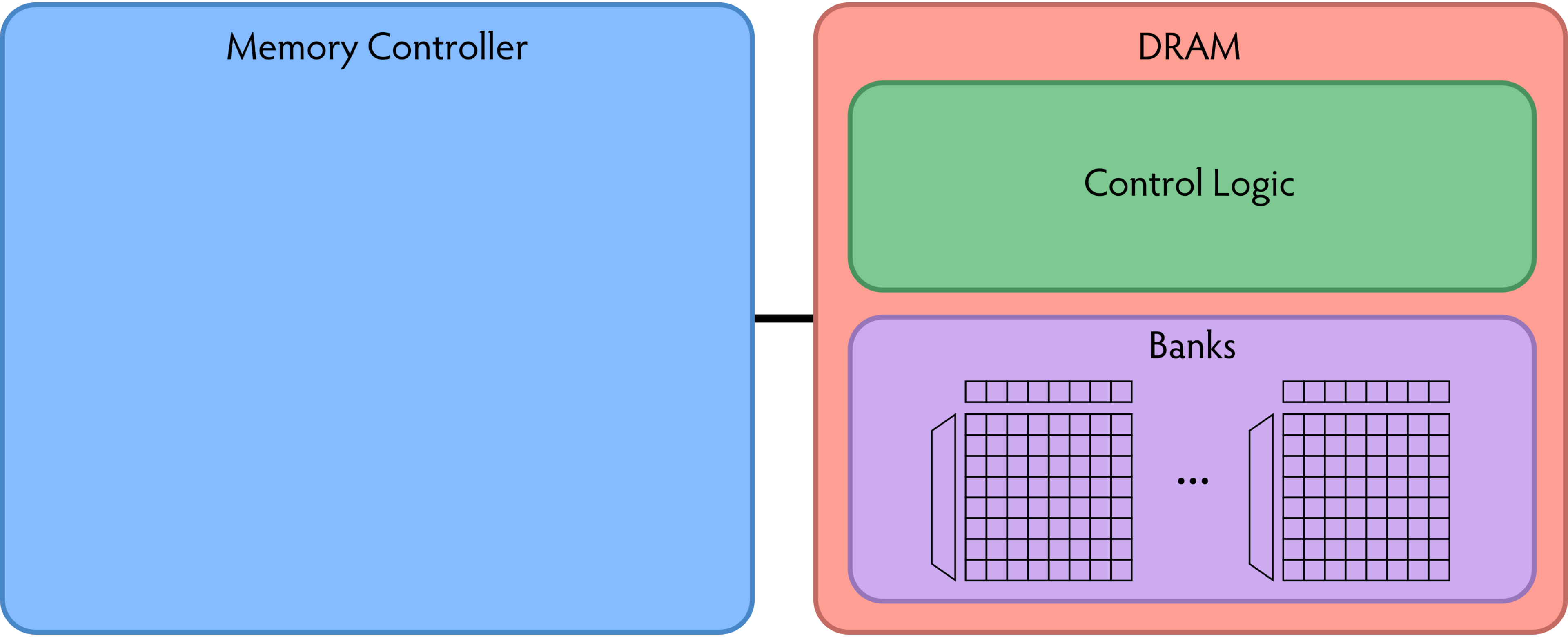
Tolerating Temperature Changes

- Change in temperature causes retention time of all cells to change by a uniform and predictable factor

- Refresh rate scaling: increase the refresh rate for all rows uniformly, depending on the temperature

- Implementation: counter with programmable period
  - Lower temperature ⇒ longer period ⇒ less frequent refreshes
  - Higher temperature ⇒ shorter period ⇒ more frequent refreshes

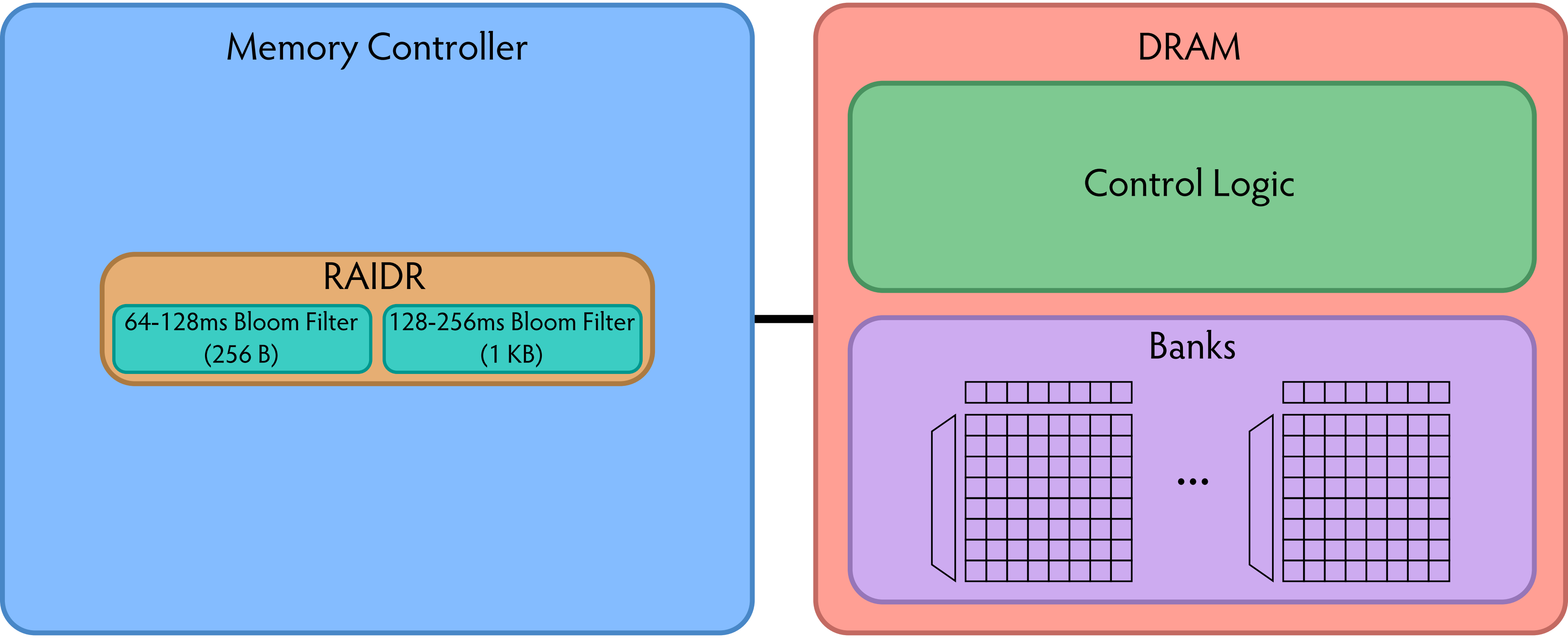
RAIDR: Baseline Design



Refresh control is in DRAM in today’s auto-refresh systems

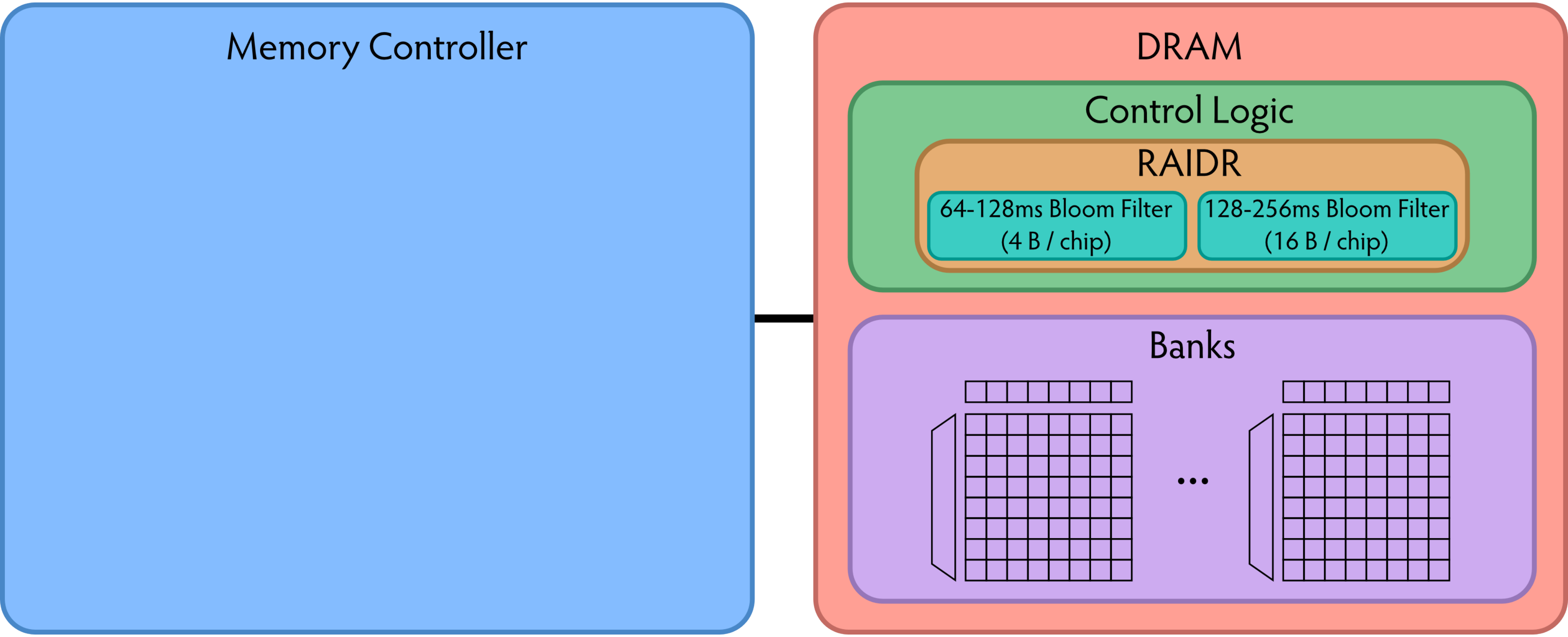
RAIDR can be implemented in either the controller or DRAM

RAIDR in Memory Controller: Option 1



Overhead of RAIDR in DRAM controller:
1.25 KB Bloom Filters, 3 counters, additional commands issued for per-row refresh (all accounted for in evaluations)

RAIDR in DRAM Chip: Option 2



Overhead of RAIDR in DRAM chip:
Per-chip overhead: 20B Bloom Filters, 1 counter (4 Gbit chip)
Total overhead: 1.25KB Bloom Filters, 64 counters (32 GB DRAM)

RAIDR Results

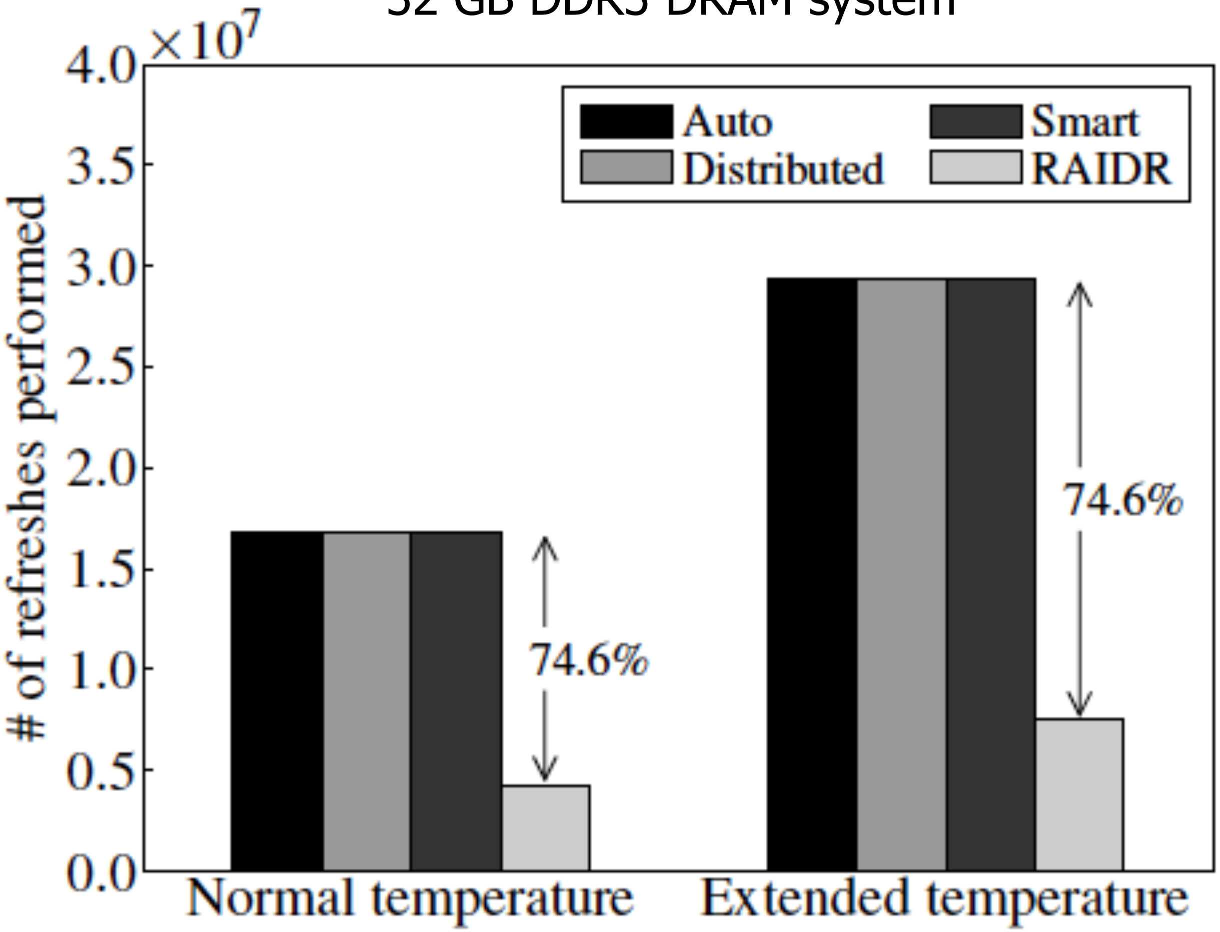
- Baseline:
  - 32 GB DDR3 DRAM system (8 cores, 512KB cache/core)
  - 64ms refresh interval for all rows

- RAIDR:
  - 64–128ms retention range: 256 B Bloom filter, 10 hash functions
  - 128–256ms retention range: 1 KB Bloom filter, 6 hash functions
  - Default refresh interval: 256 ms

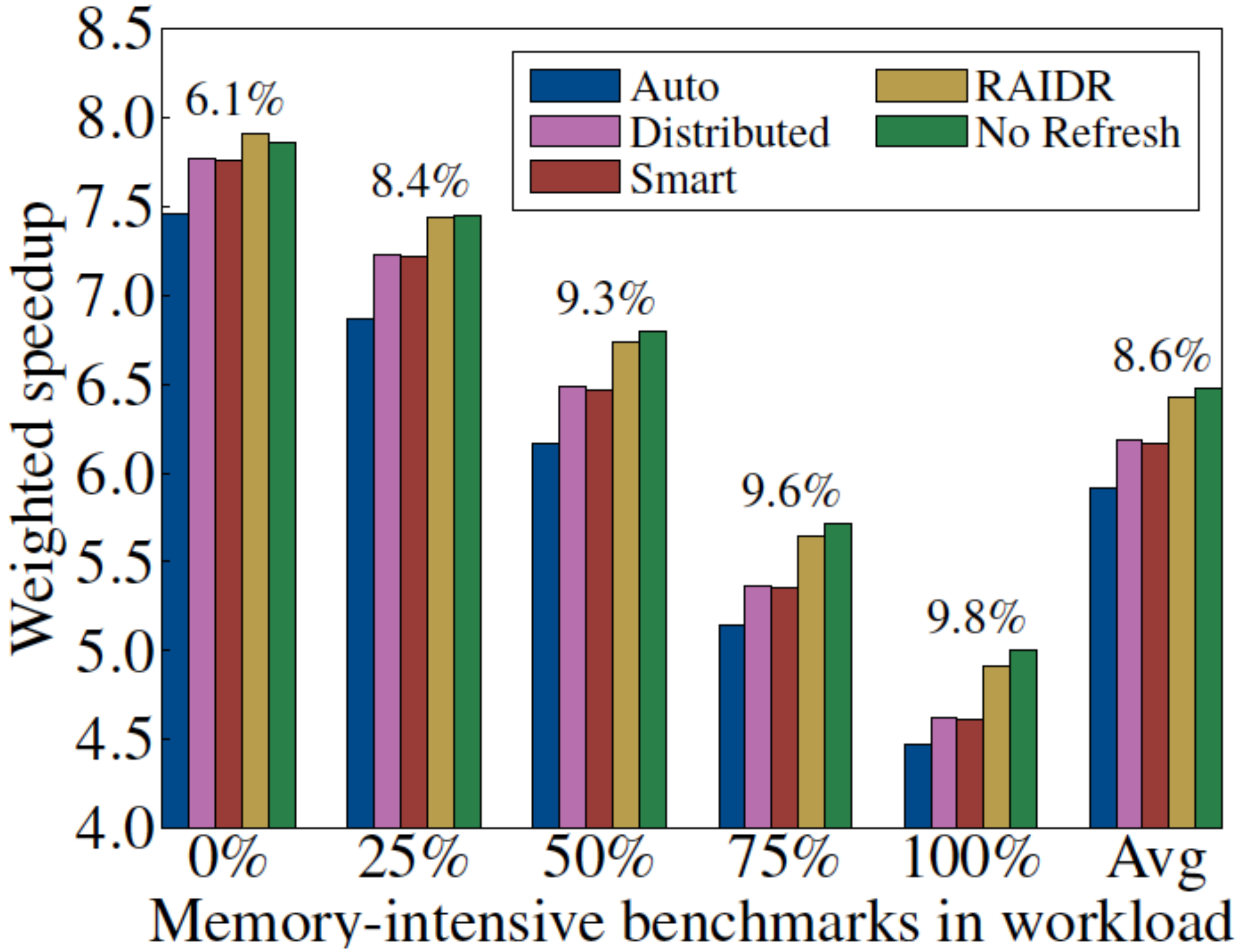
- Results on SPEC CPU2006, TPC-C, TPC-H benchmarks
  - 74.6% refresh reduction
  - ~16%/20% DRAM dynamic/idle power reduction
  - ~9% performance improvement

RAIDR Refresh Reduction

32 GB DDR3 DRAM system

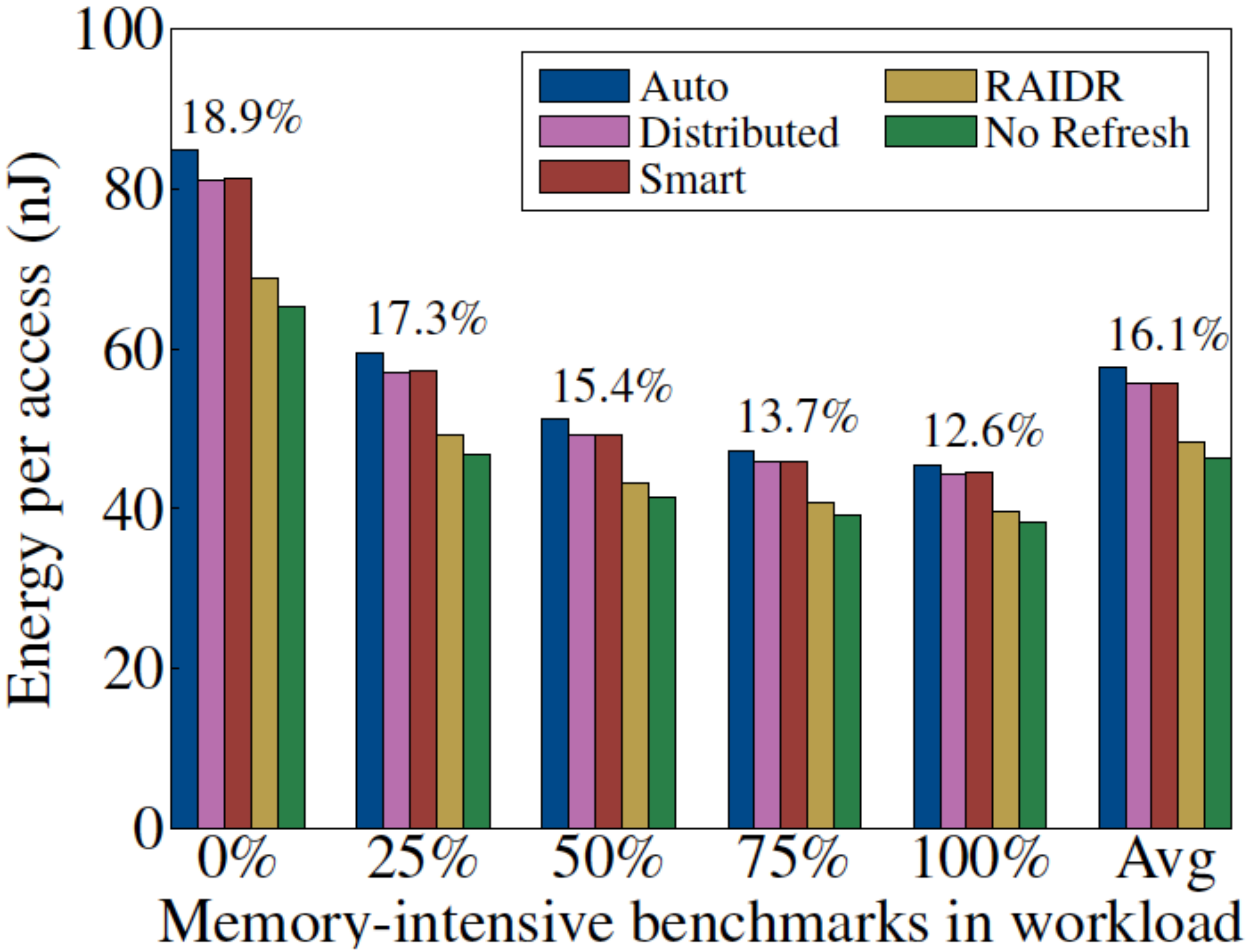


RAIDR: Performance



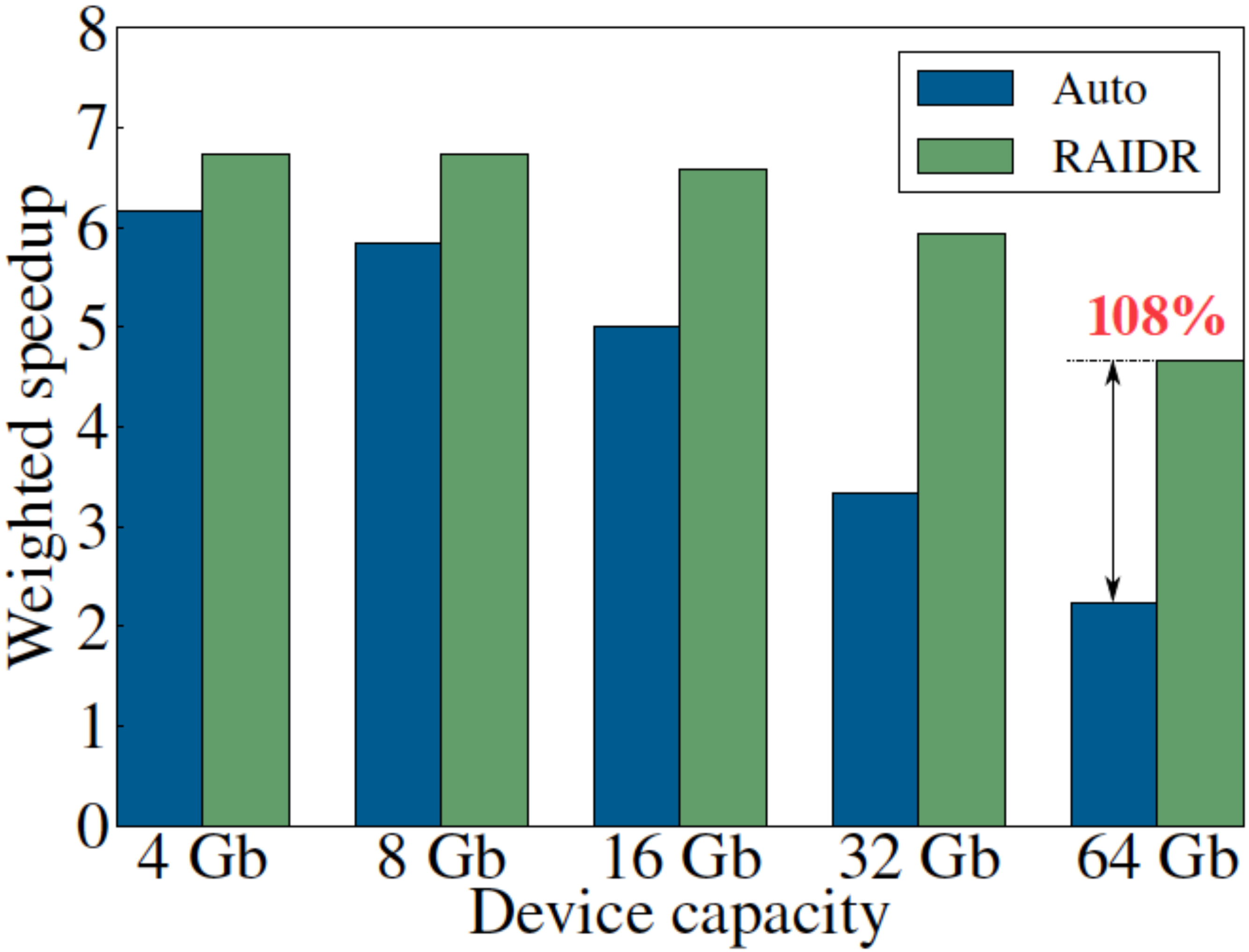
RAIDR performance benefits increase with workload's memory intensity

RAIDR: DRAM Energy Efficiency



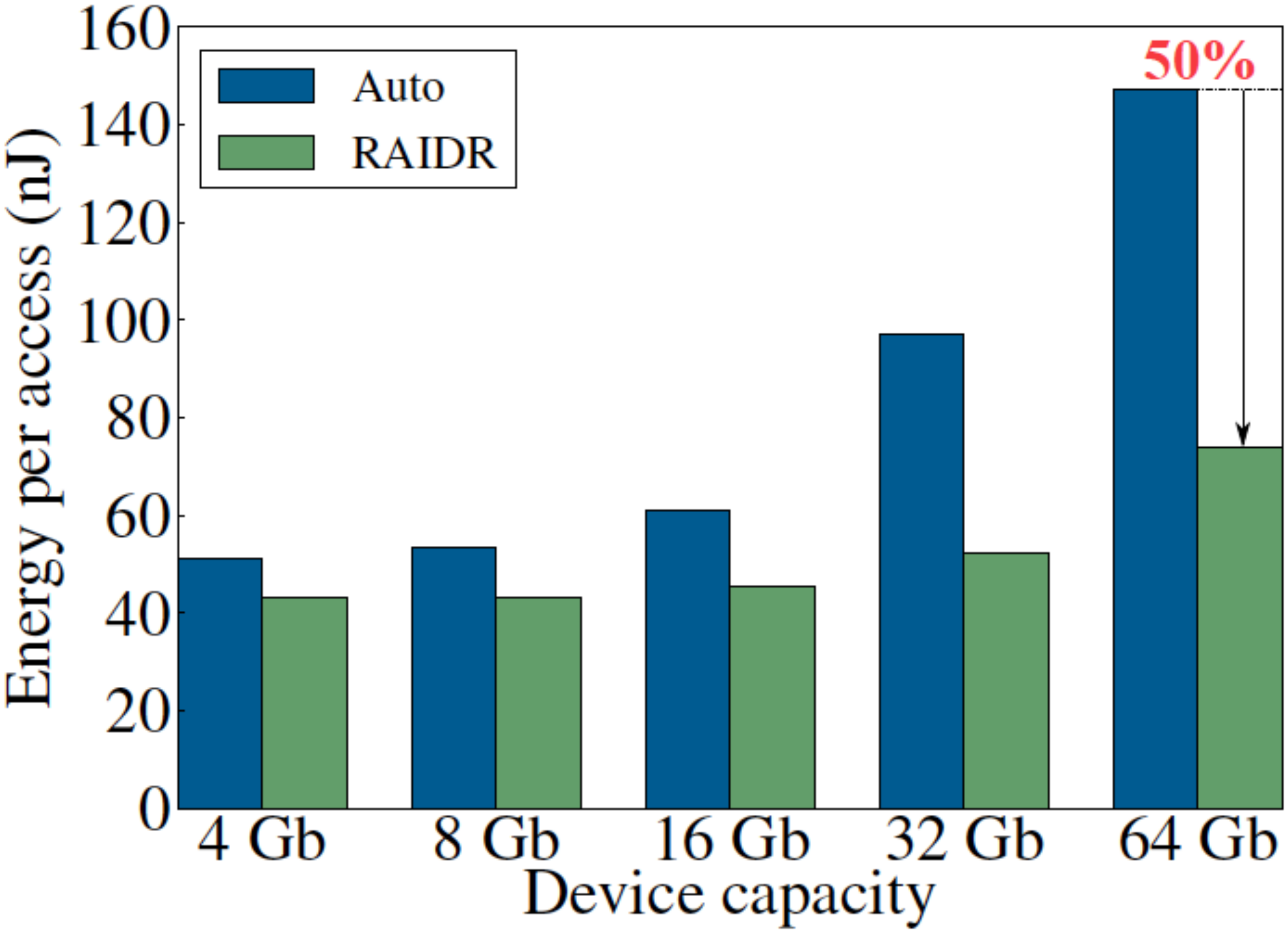
RAIDR energy benefits increase with memory idleness

DRAM Device Capacity Scaling: Performance



RAIDR performance benefits increase with DRAM chip capacity

DRAM Device Capacity Scaling: Energy



RAIDR energy benefits increase with DRAM chip capacity

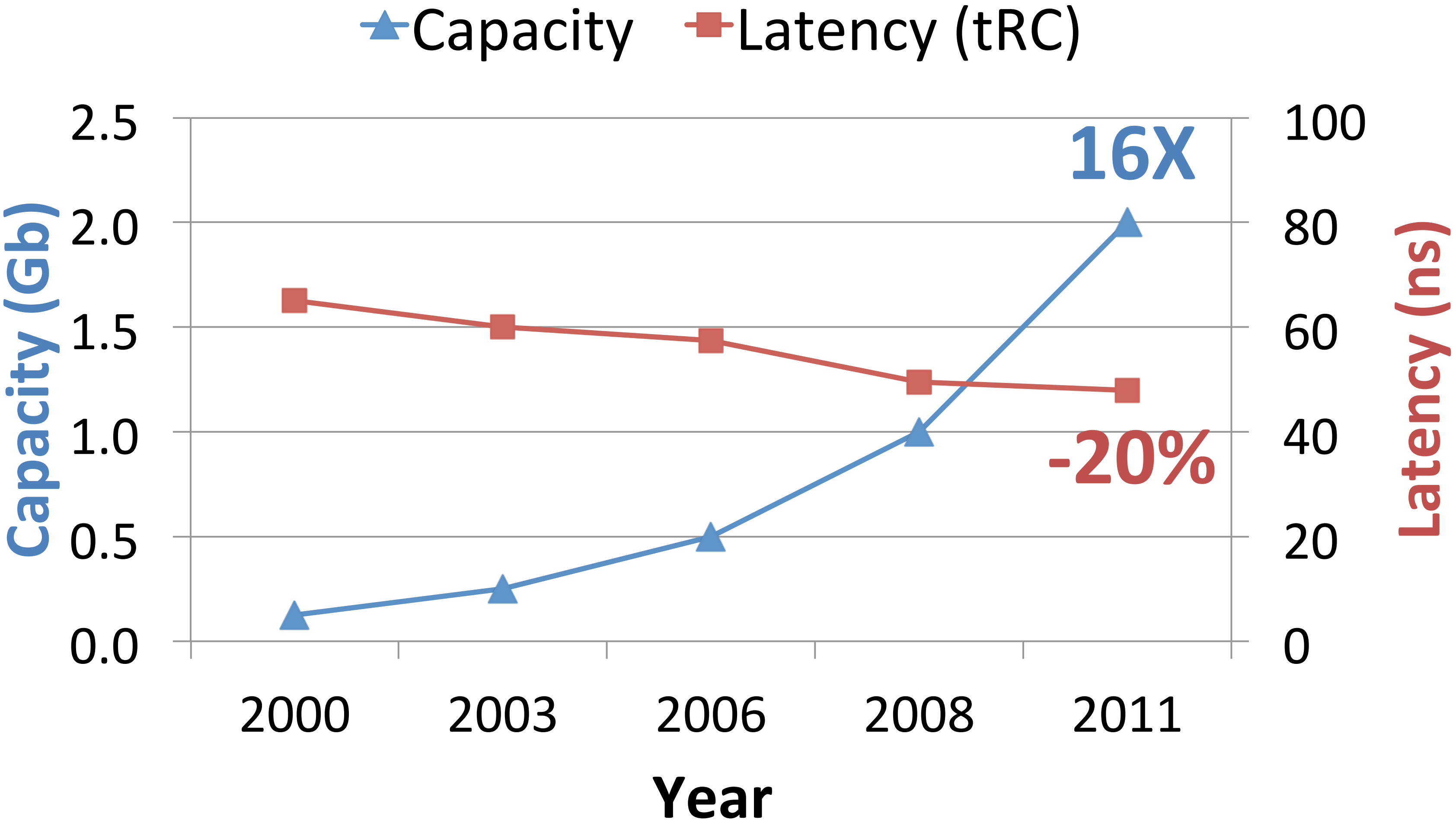
RAIDR slides

New DRAM Architectures

- RAIDR: Reducing Refresh Impact
- TL-DRAM: Reducing DRAM Latency
- SALP: Reducing Bank Conflict Impact
- RowClone: Fast Bulk Data Copy and Initialization

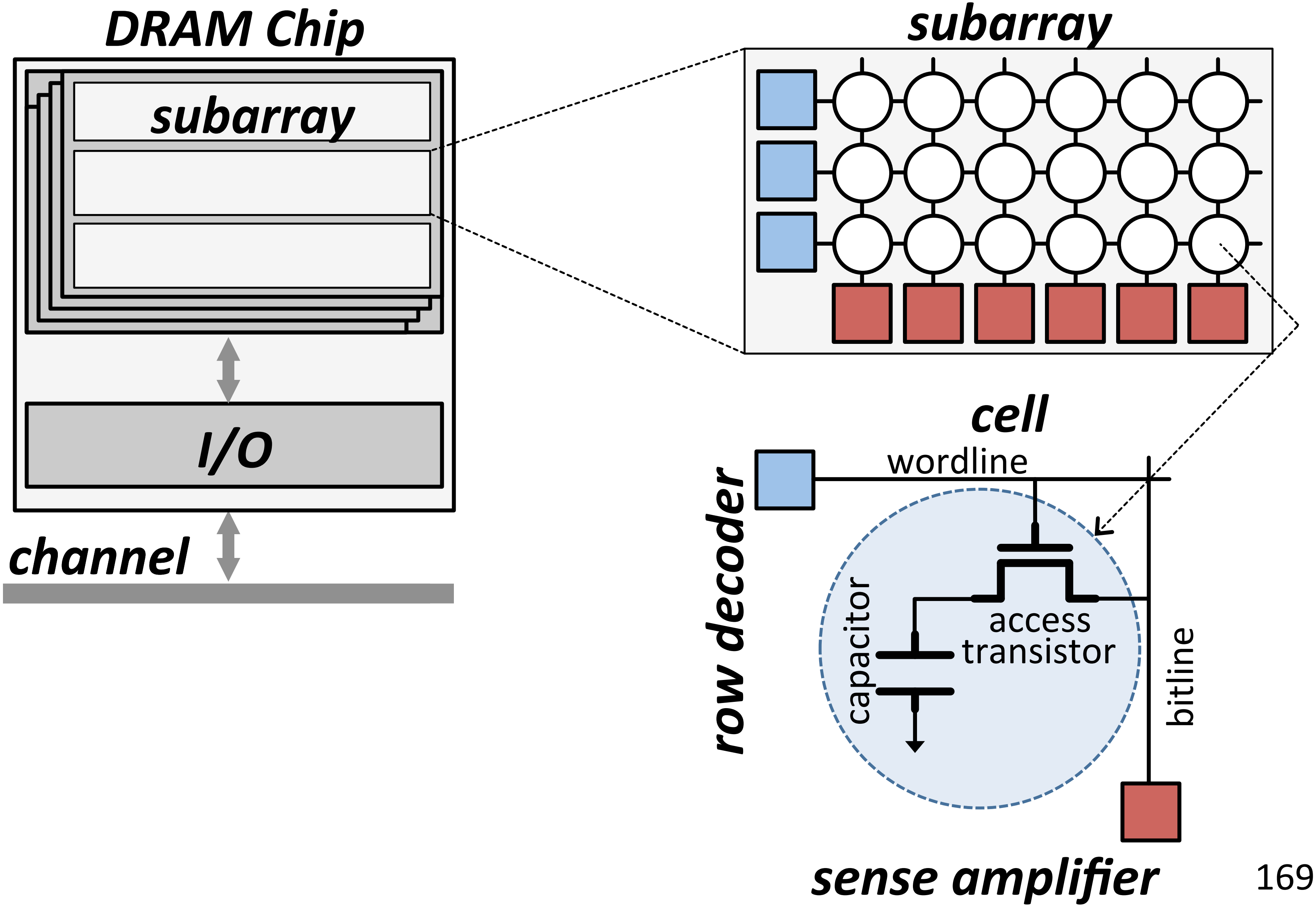
Tiered-Latency DRAM: Reducing DRAM Latency

Historical DRAM Latency-Capacity Trend

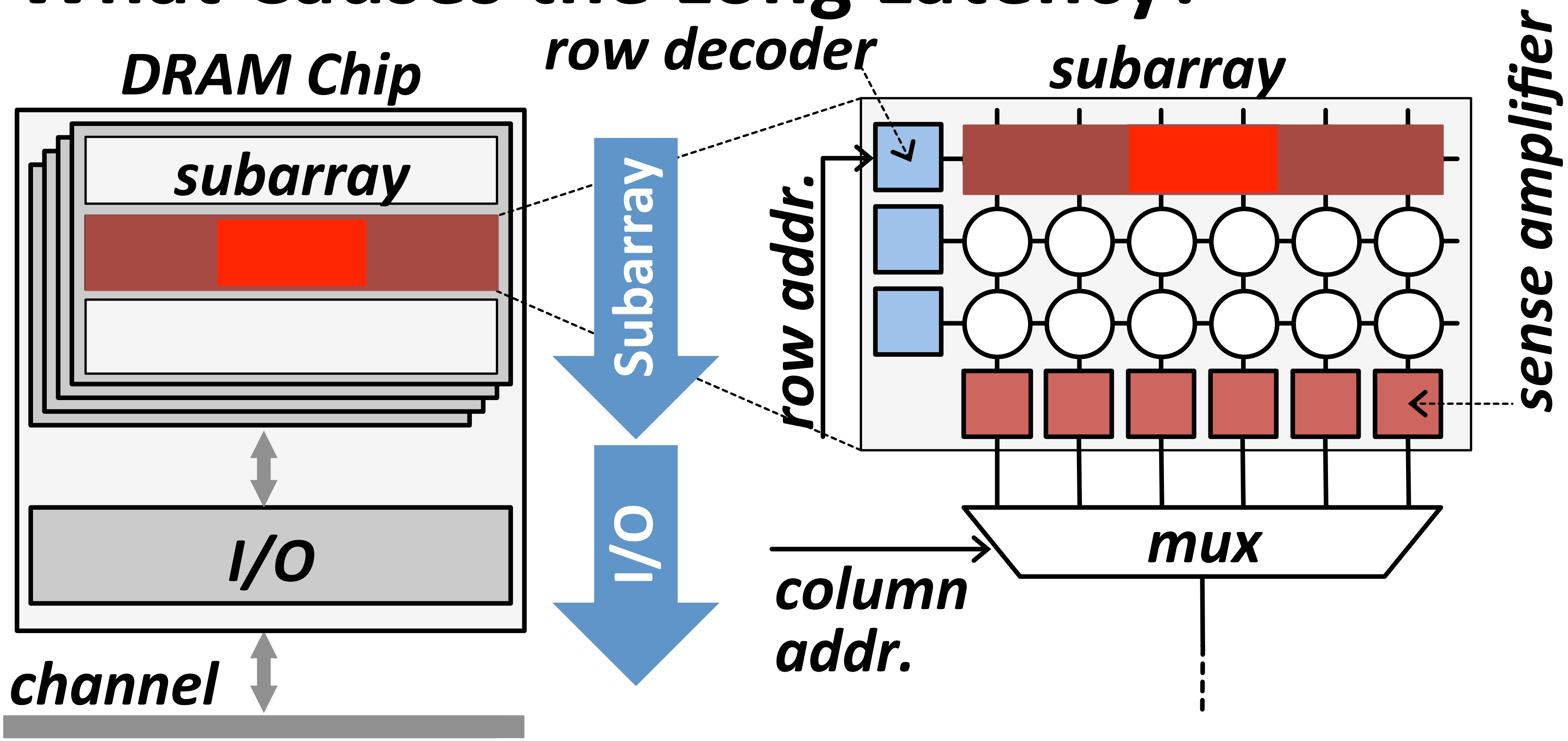


DRAM latency continues to be a critical bottleneck

What Causes the Long Latency?



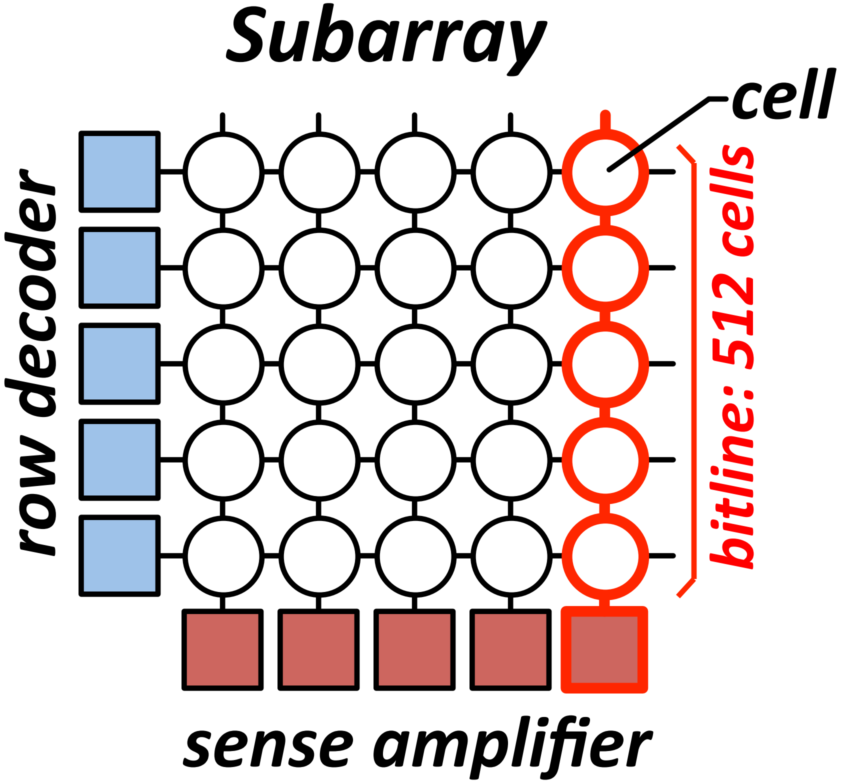
What Causes the Long Latency?

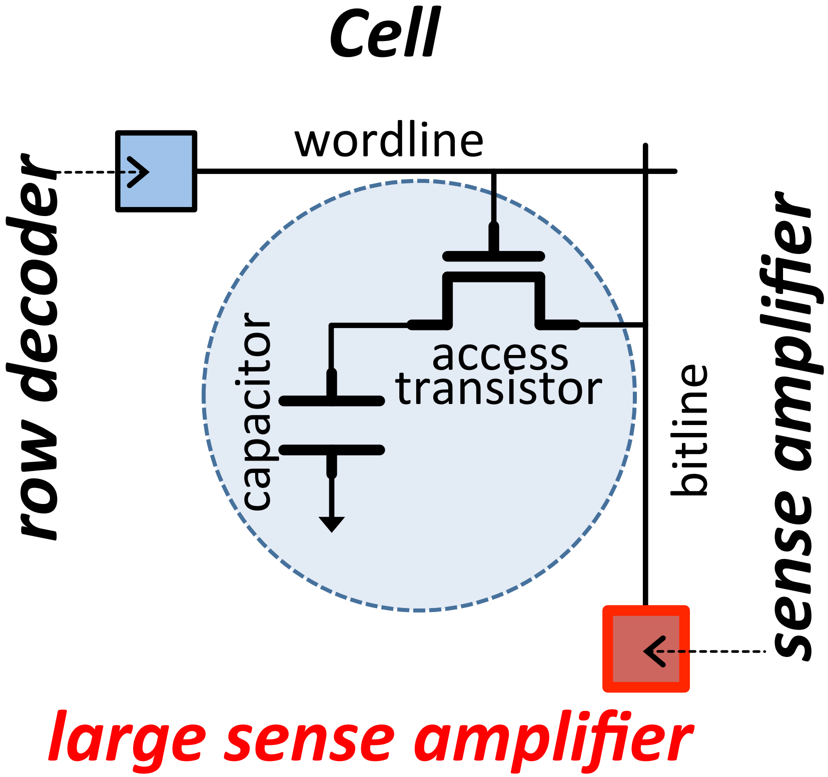


DRAM Latency = Subarray Latency + I/O Latency

Dominant

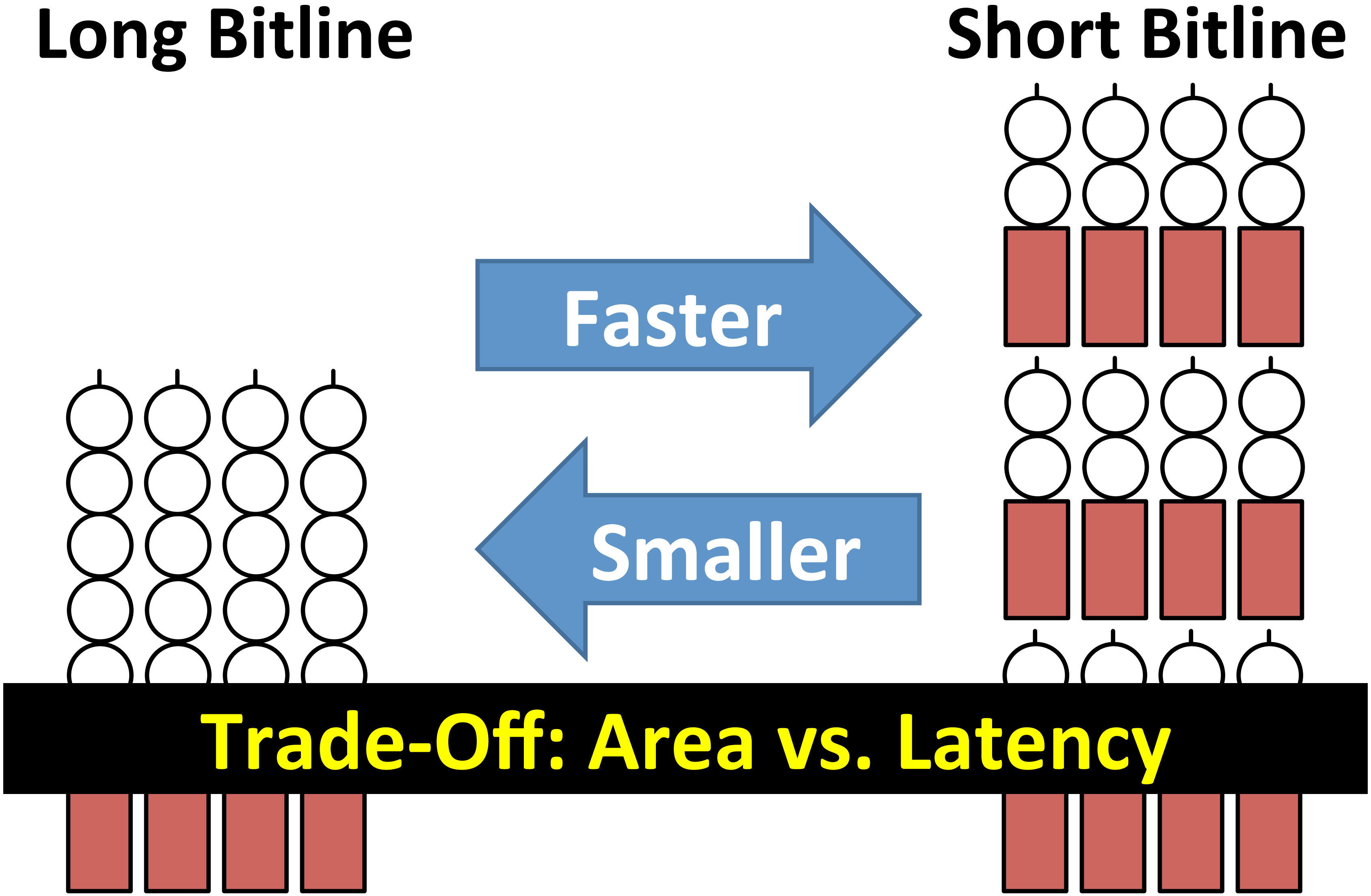
Why is the Subarray So Slow?





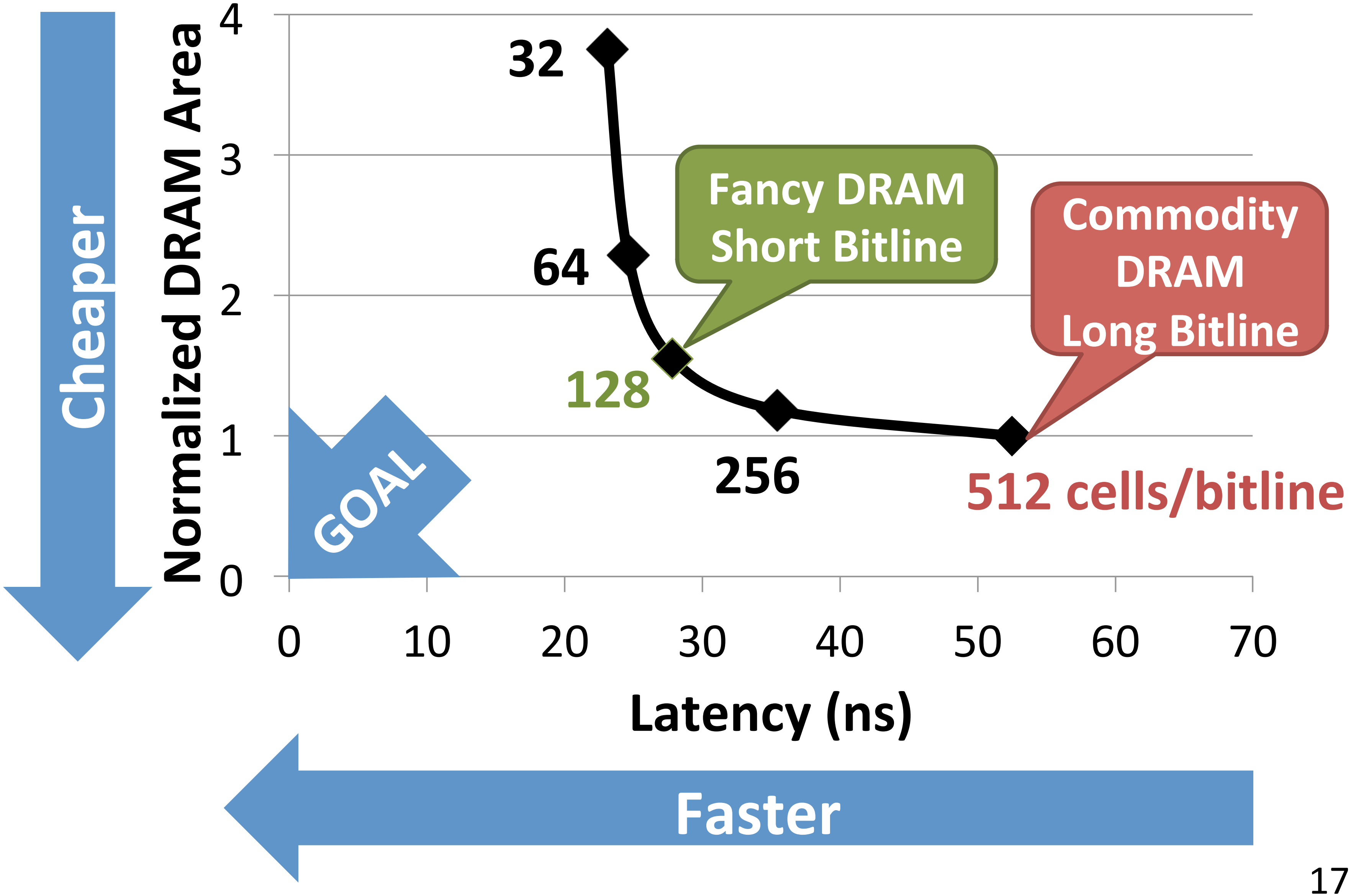
- **Long bitline**
  - **Amortizes sense amplifier cost → Small area**
  - **Large bitline capacitance → High latency & power**

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



Trade-Off: Area vs. Latency

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



Fancy DRAM
Short Bitline

Commodity
DRAM
Long Bitline

Approximating the Best of Both Worlds

Long Bitline

Small Area

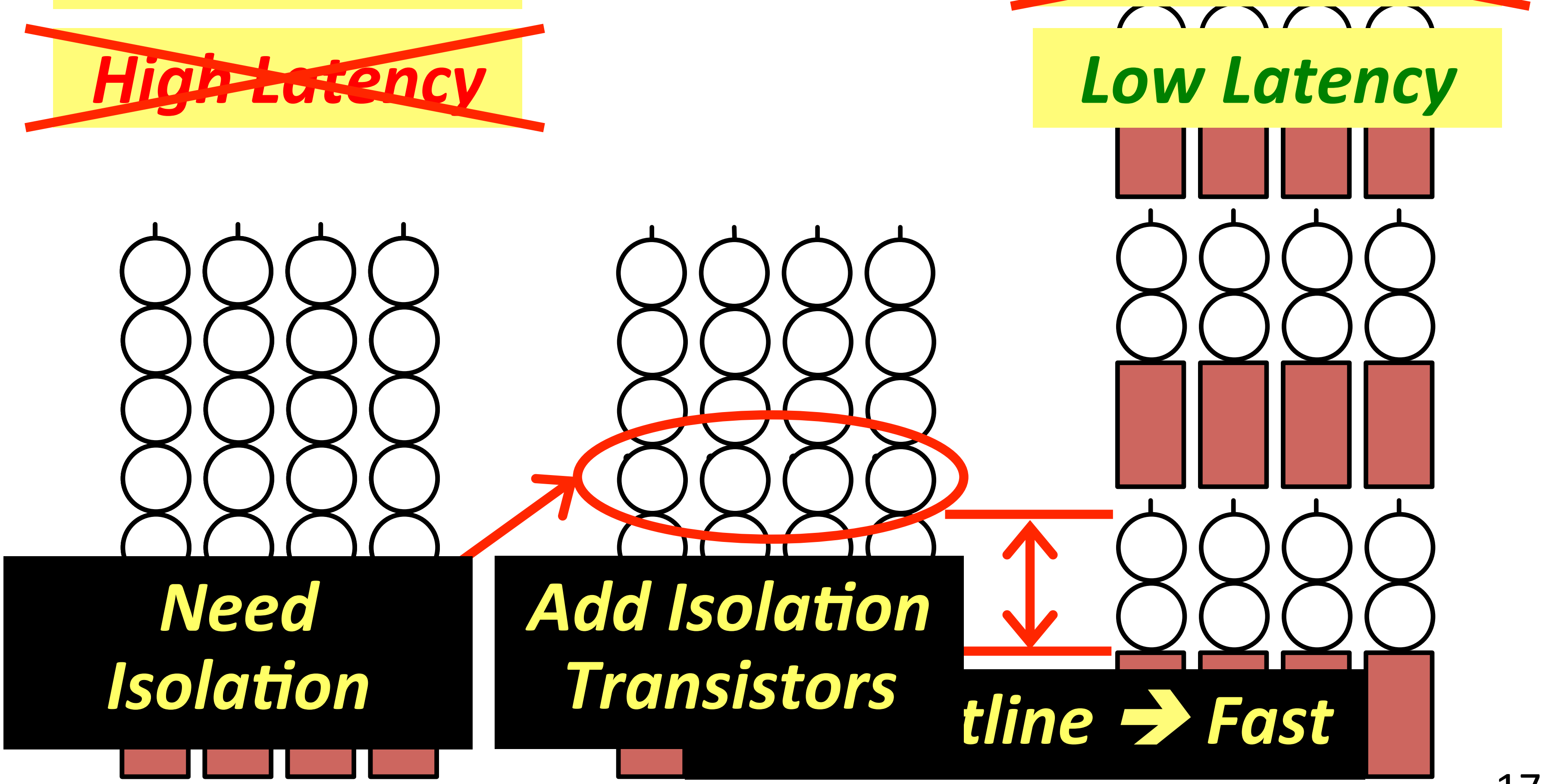
~~High Latency~~

Our Proposal

Short Bitline

~~Large Area~~

Low Latency



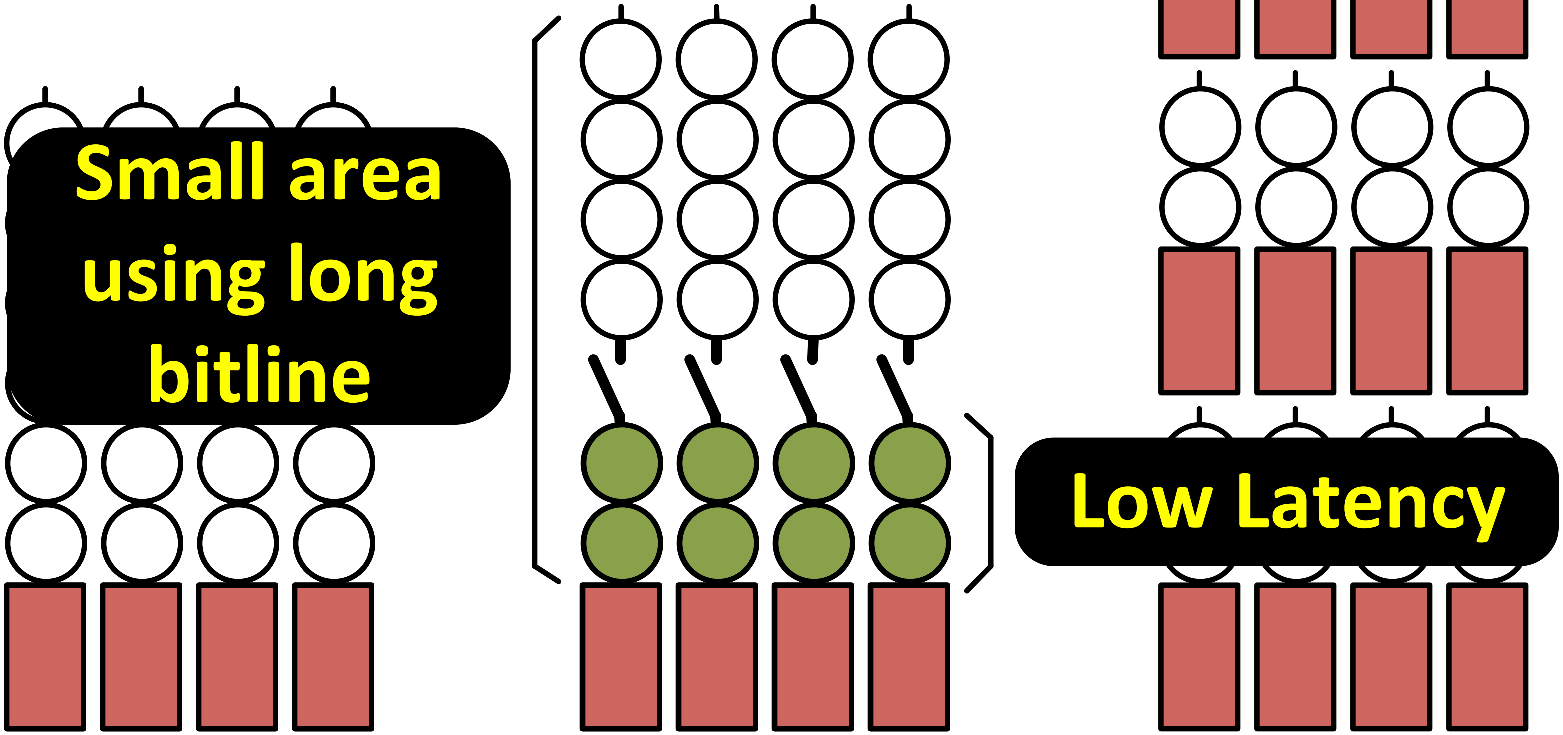
Need Isolation

Add Isolation Transistors

tline → Fast

Approximating the Best of Both Worlds

Long Bitlin	Tiered-Latency DRAM	hort Bitline
Small Area	*Small Area*	~~*Large Area*~~
~~*High Latency*~~	*Low Latency*	*Low Latency*

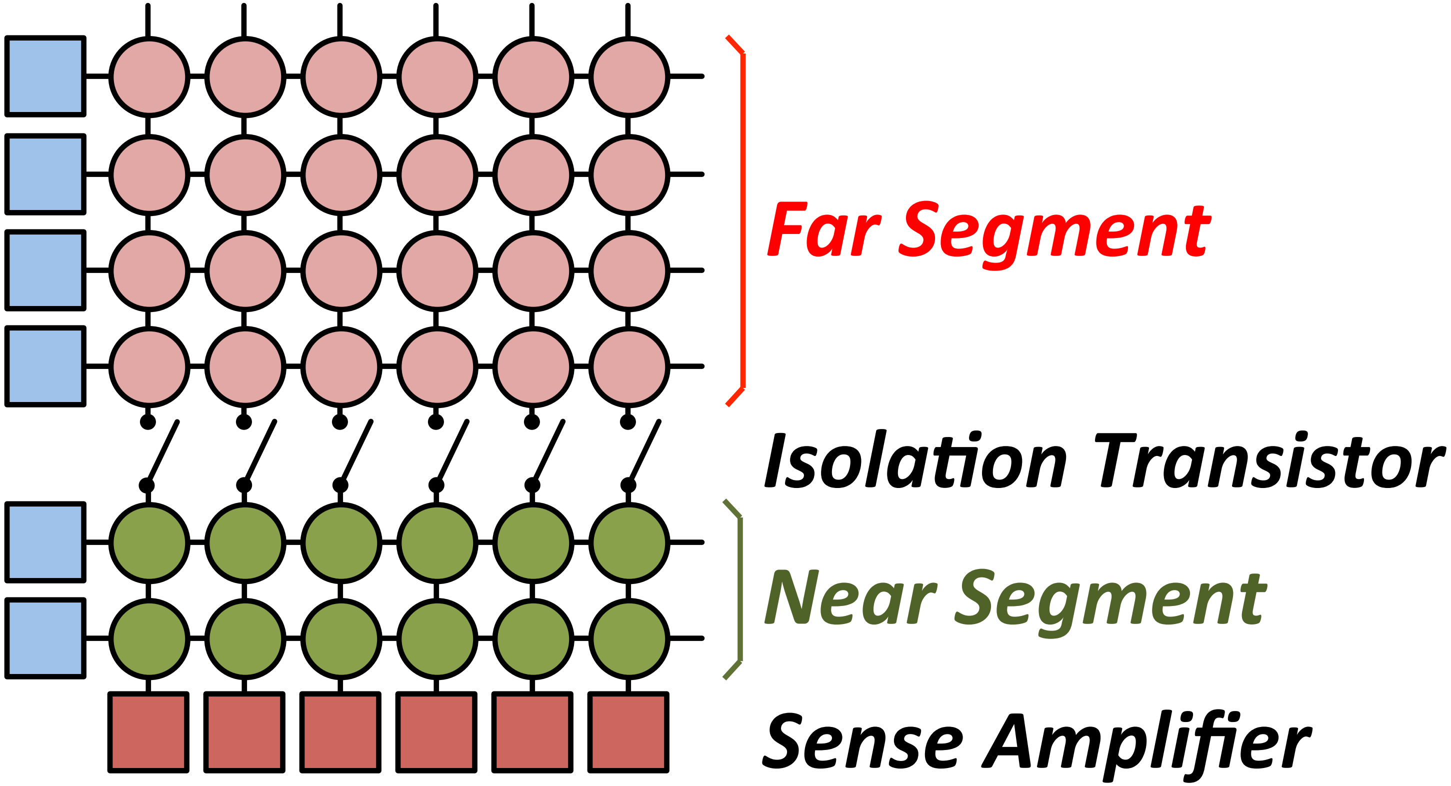


Small area using long bitline

Low Latency

Tiered-Latency DRAM

- Divide a bitline into two segments with an **isolation transistor**



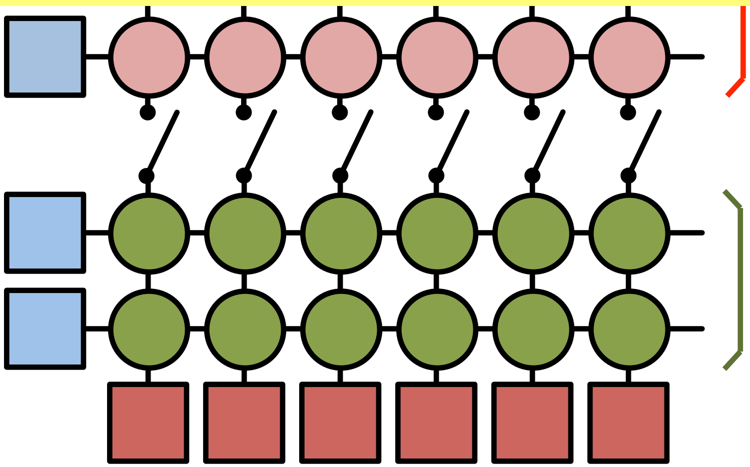
Near Segment Access

- Turn *off* the isolation transistor

Reduced bitline length

Reduced bitline capacitance

➔ Low latency & low power



*Isolation Transistor (**off**)*

Near Segment

Sense Amplifier

Far Segment Access

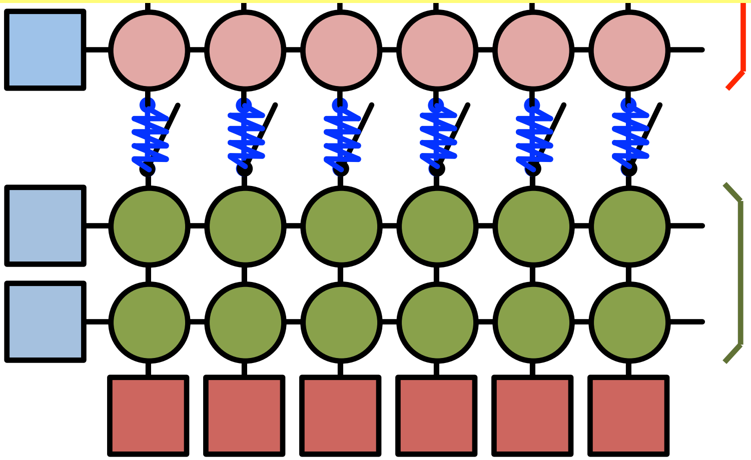
- Turn *on* the isolation transistor

Long bitline length

Large bitline capacitance

Additional resistance of isolation transistor

➔ High latency & high power



Isolation Transistor (on)

Near Segment

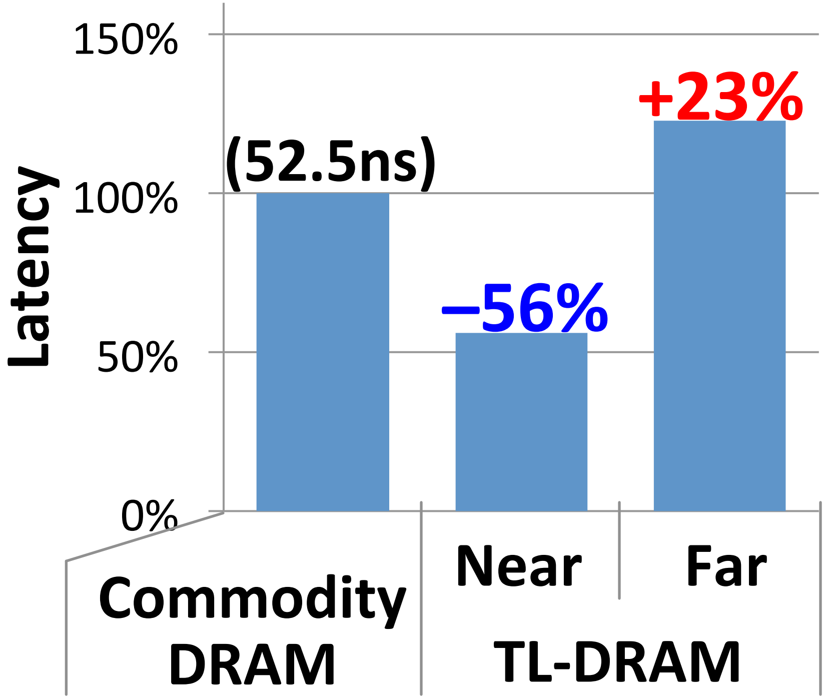
Sense Amplifier

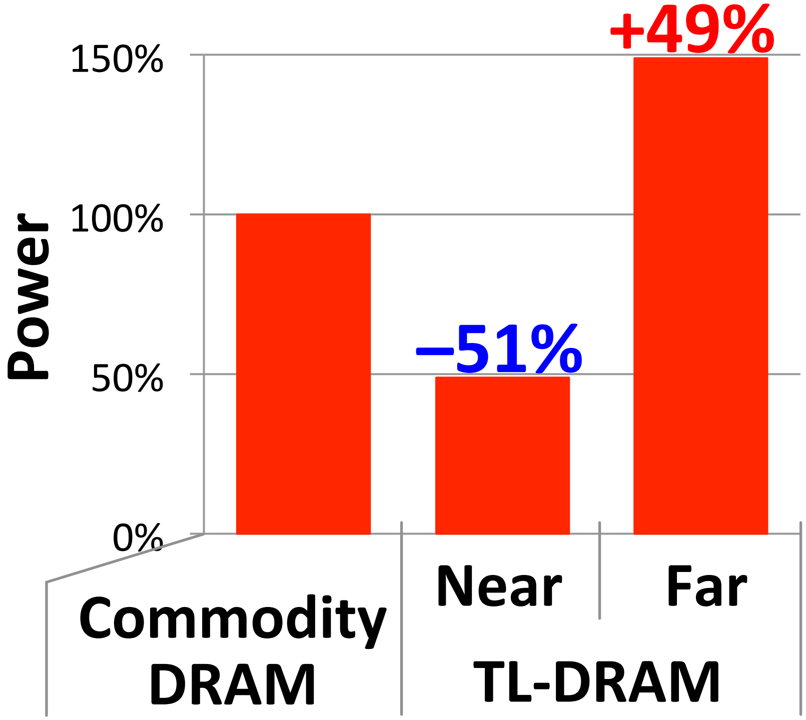
Latency, Power, and Area Evaluation

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

Commodity DRAM vs. TL-DRAM

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

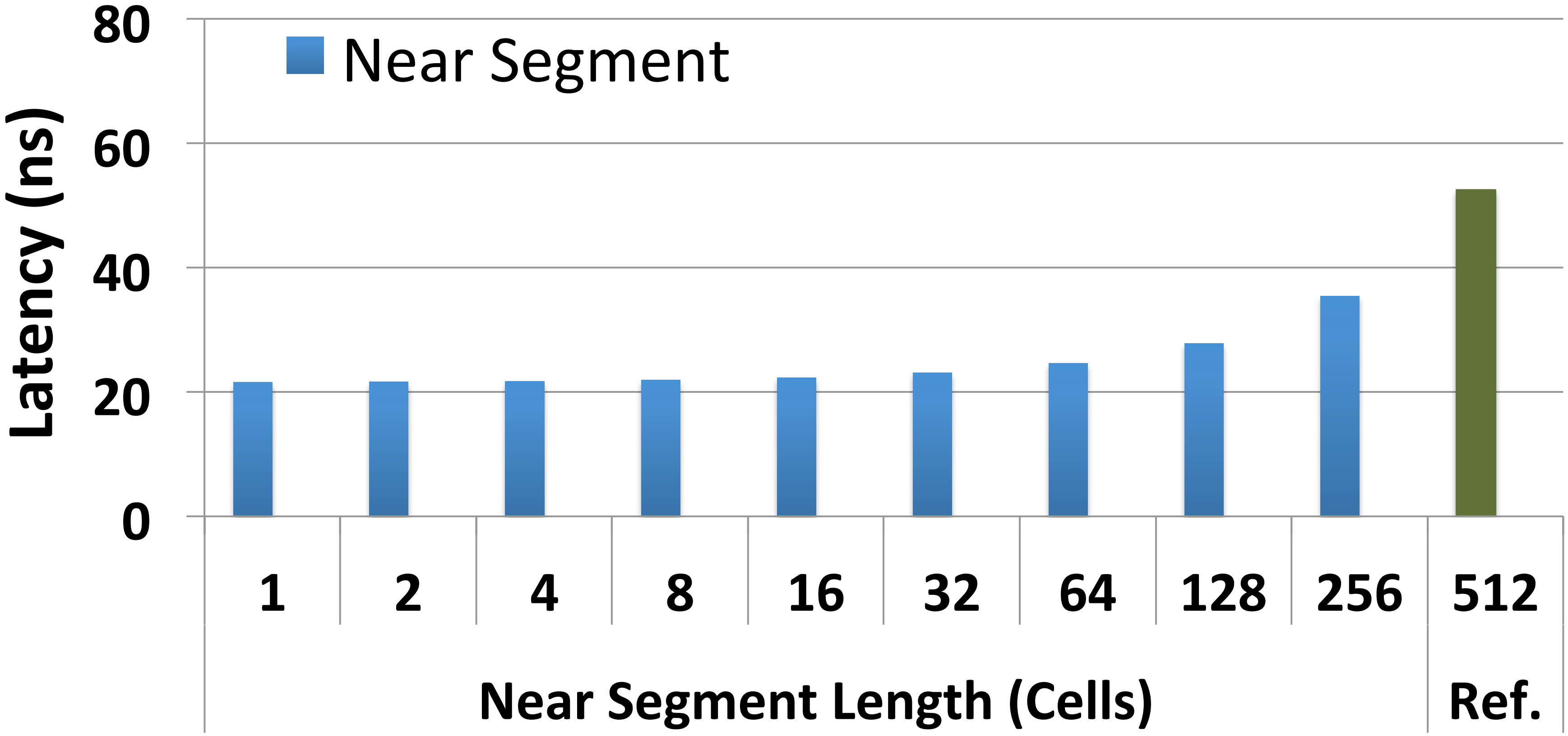




- **DRAM Area Overhead**

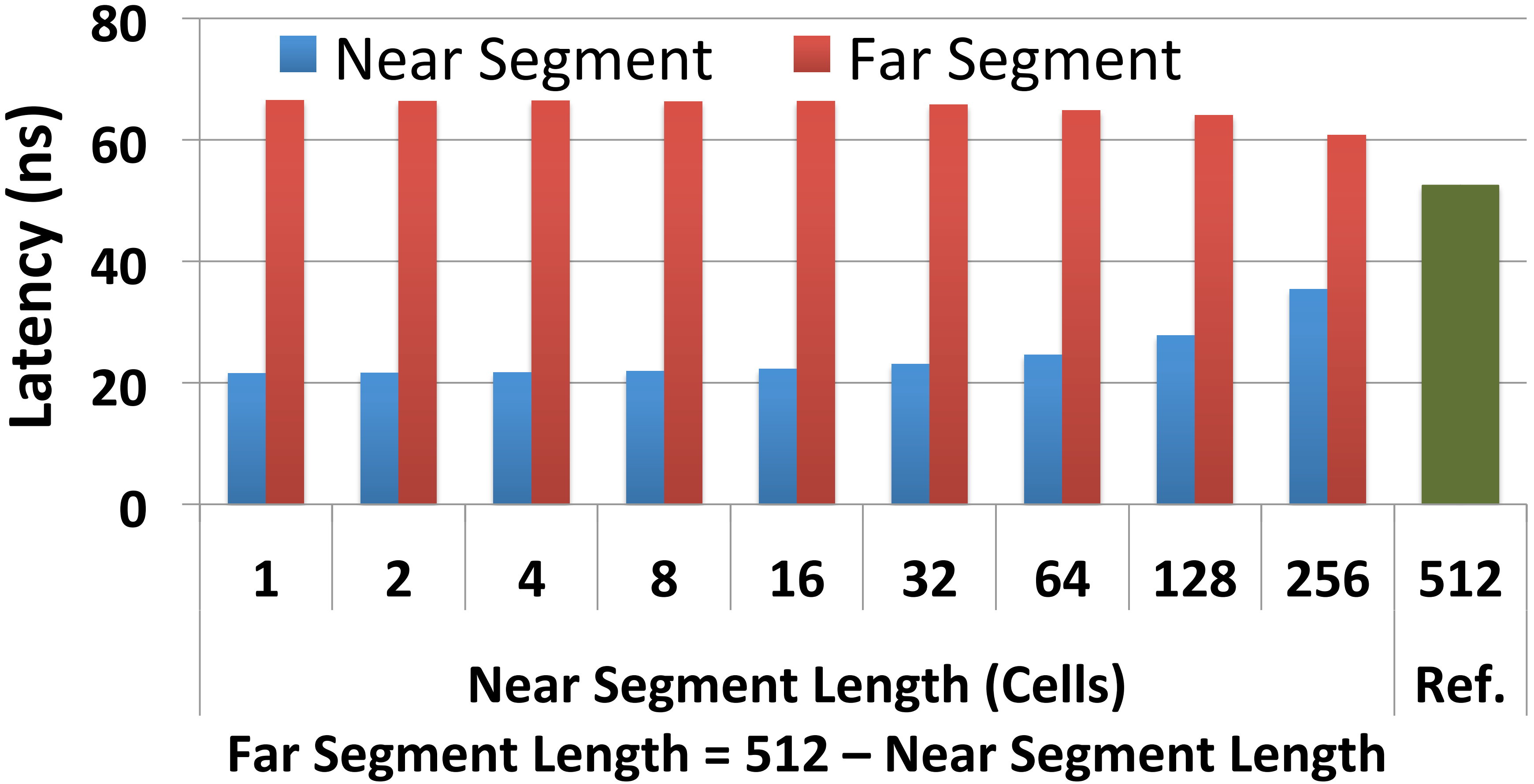
  ~3%: mainly due to the isolation transistors

Latency vs. Near Segment Length



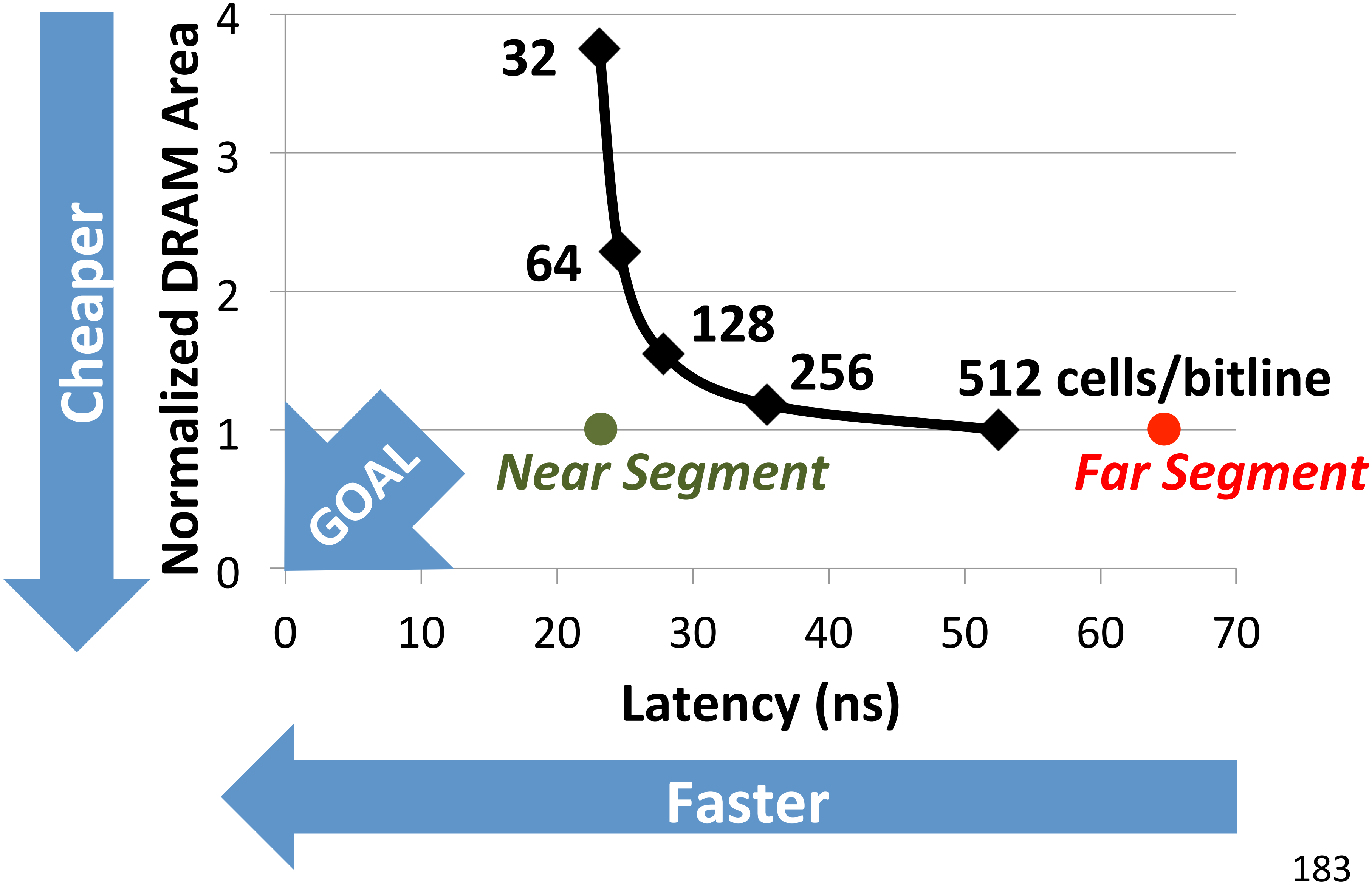
Longer near segment length leads to higher near segment latency

Latency vs. Near Segment Length



Far segment latency is higher than commodity DRAM latency

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

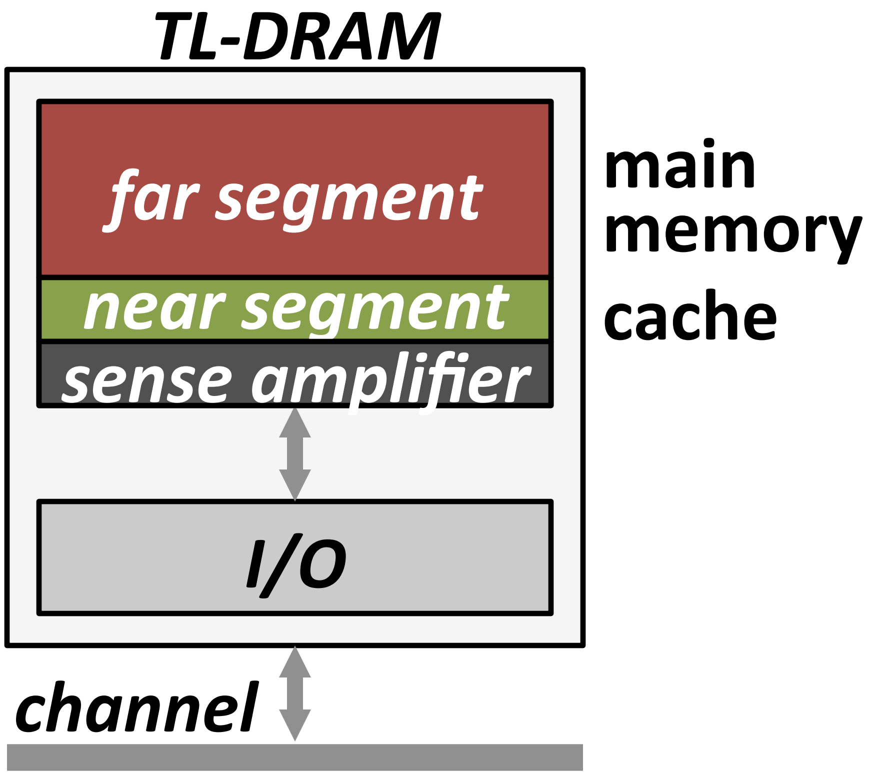


Leveraging Tiered-Latency DRAM

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

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

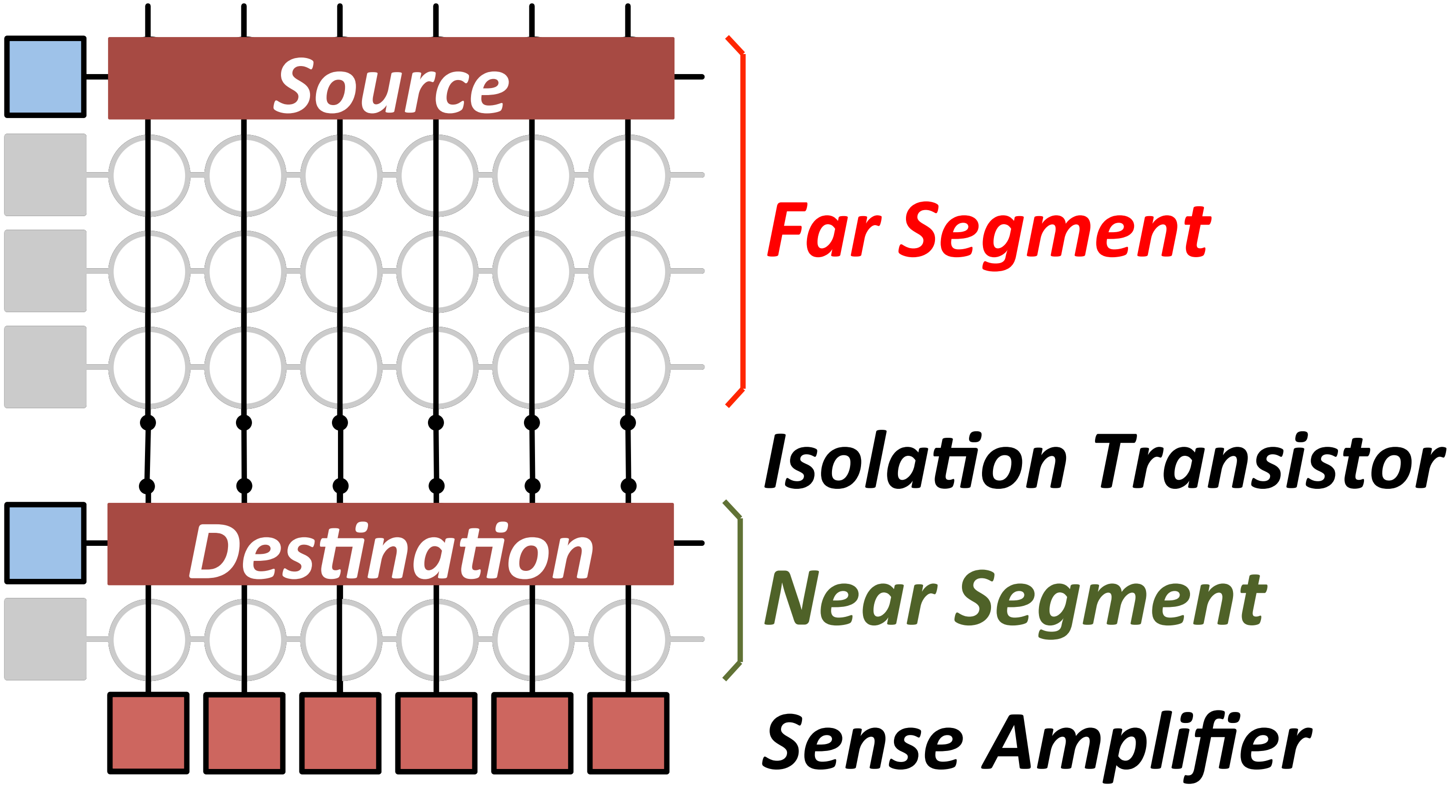
Near Segment as Hardware-Managed Cache



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

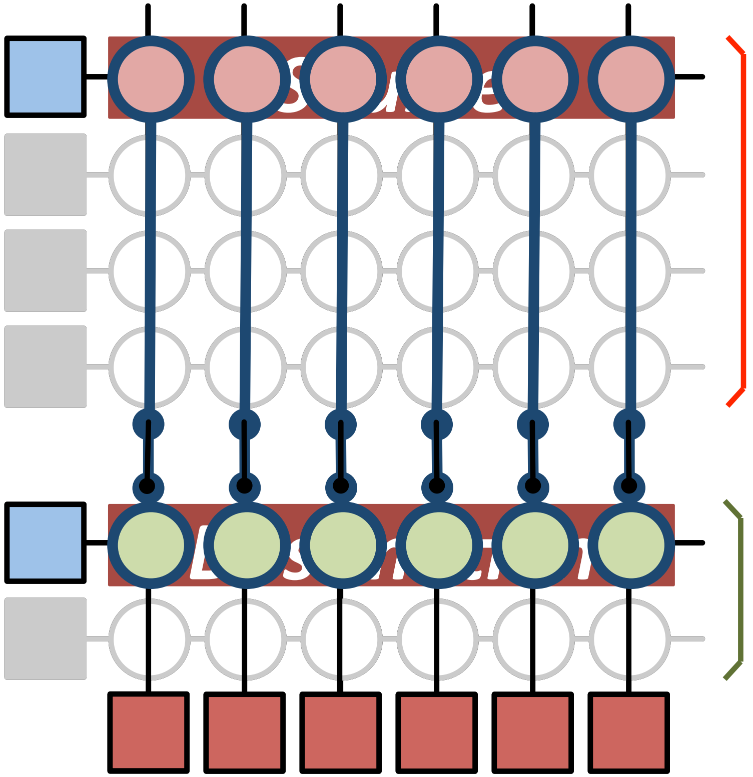
Inter-Segment Migration

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



Inter-Segment Migration

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



Far Segment

Isolation Transistor

Near Segment

Sense Amplifier

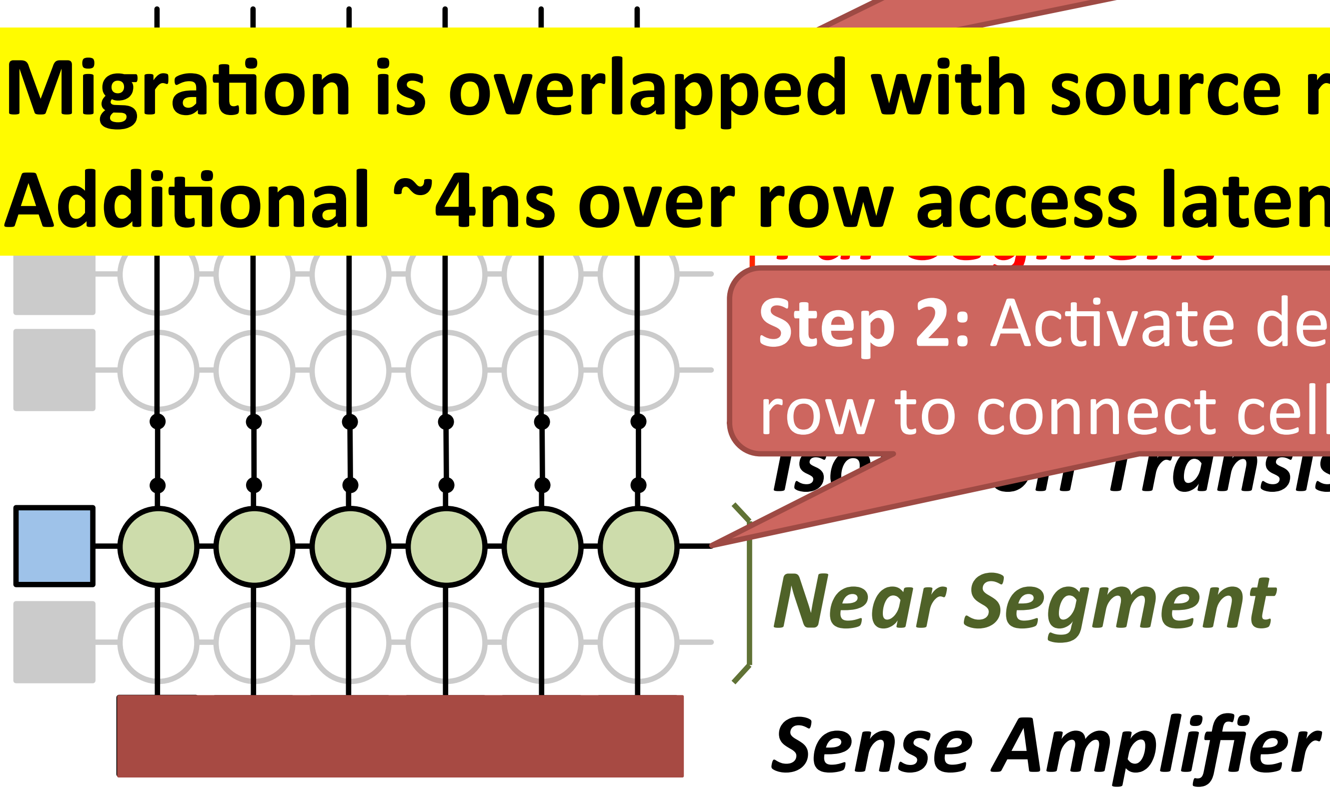
Inter-Segment Migration

- **Our way:**
  - Source and destination cells ***share bitlines***
  - Transfer data from so
    shared bitlines concur

Step 1: Activate source row

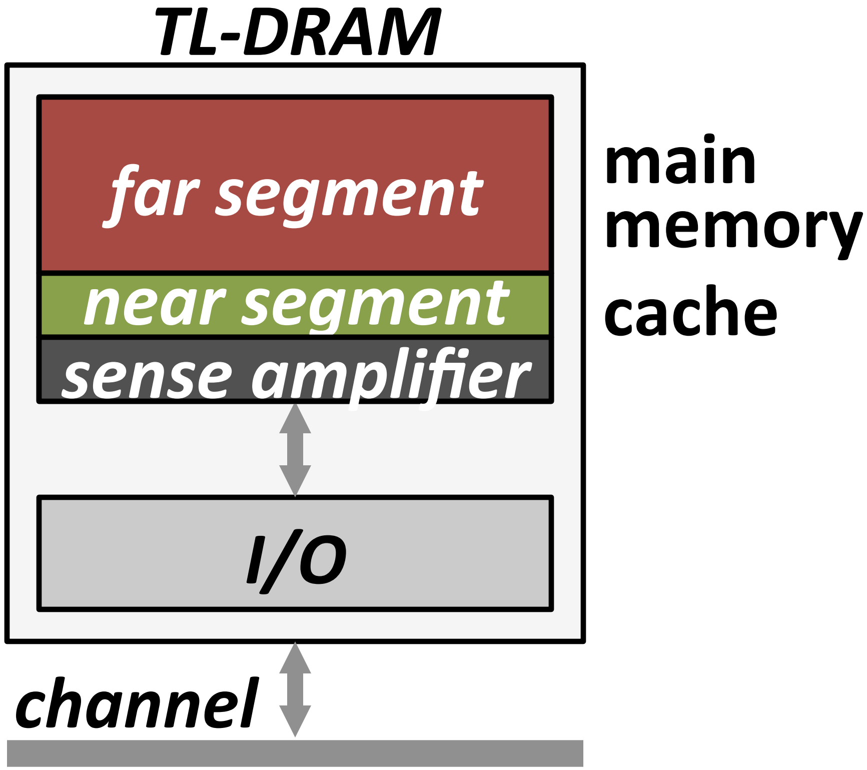
Migration is overlapped with source row access

Additional ~4ns over row access latency



Step 2: Activate destination row to connect cell and bitline

Near Segment as Hardware-Managed Cache



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

Evaluation Methodology

- **System simulator**
  - CPU: Instruction-trace-based x86 simulator
  - Memory: Cycle-accurate DDR3 DRAM simulator

- **Workloads**
  - 32 Benchmarks from TPC, STREAM, SPEC CPU2006

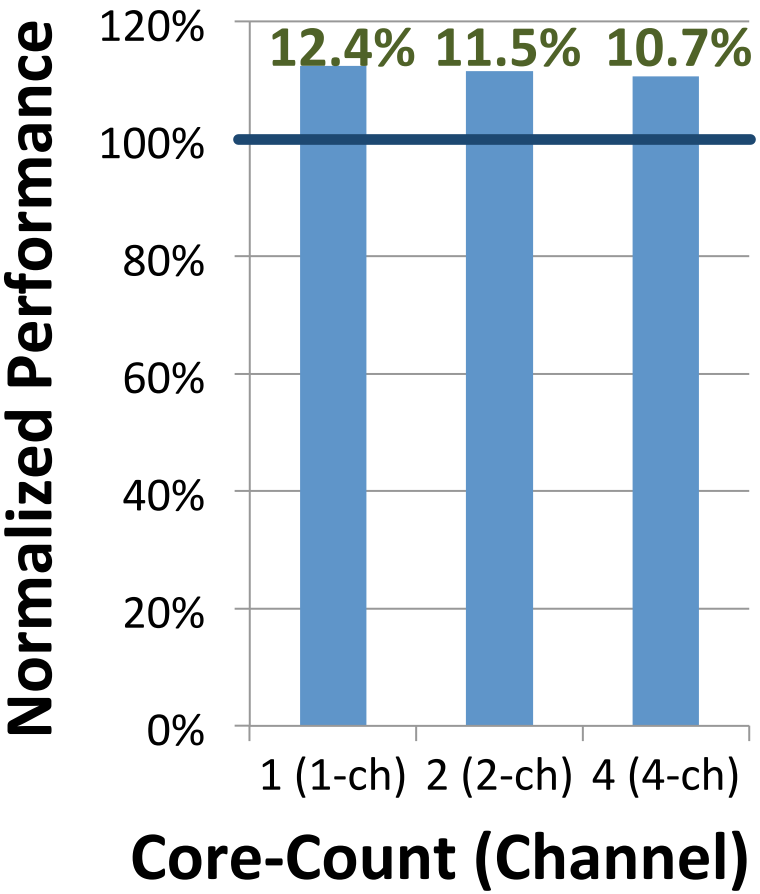
- **Performance Metrics**
  - Single-core: Instructions-Per-Cycle
  - Multi-core: Weighted speedup

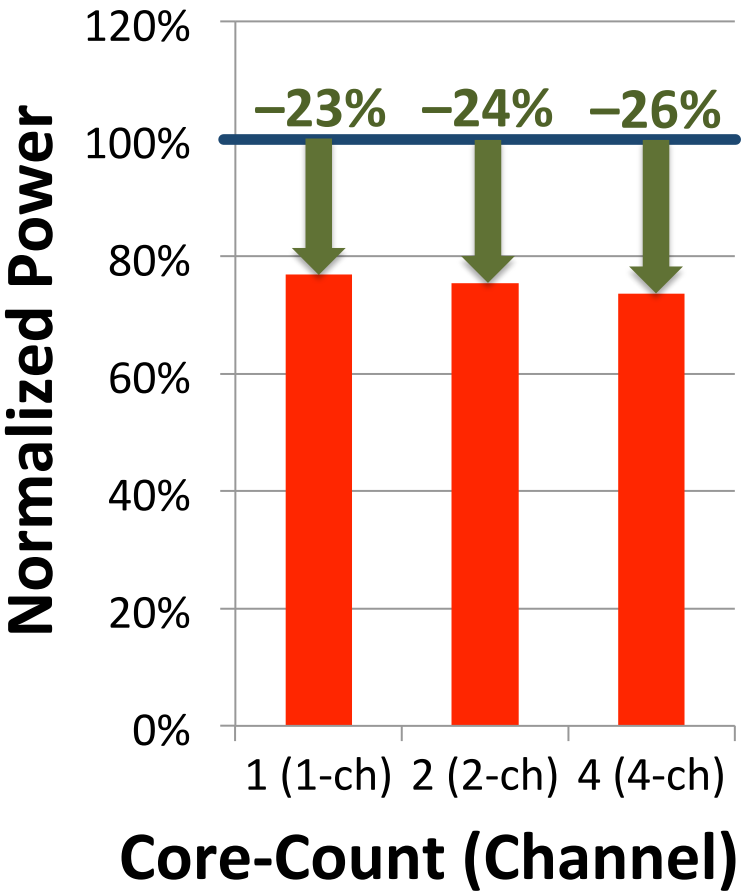
Configurations

- **System configuration**
  - CPU: 5.3GHz
  - LLC: 512kB private per core
  - **Memory: DDR3-1066**
    - 1-2 channel, 1 rank/channel
    - 8 banks, 32 subarrays/bank, **512 cells/bitline**
    - Row-interleaved mapping & closed-row policy

- **TL-DRAM configuration**
  - Total bitline length: **512 cells/bitline**
  - Near segment length: 1-256 cells
  - Hardware-managed inclusive cache: near segment

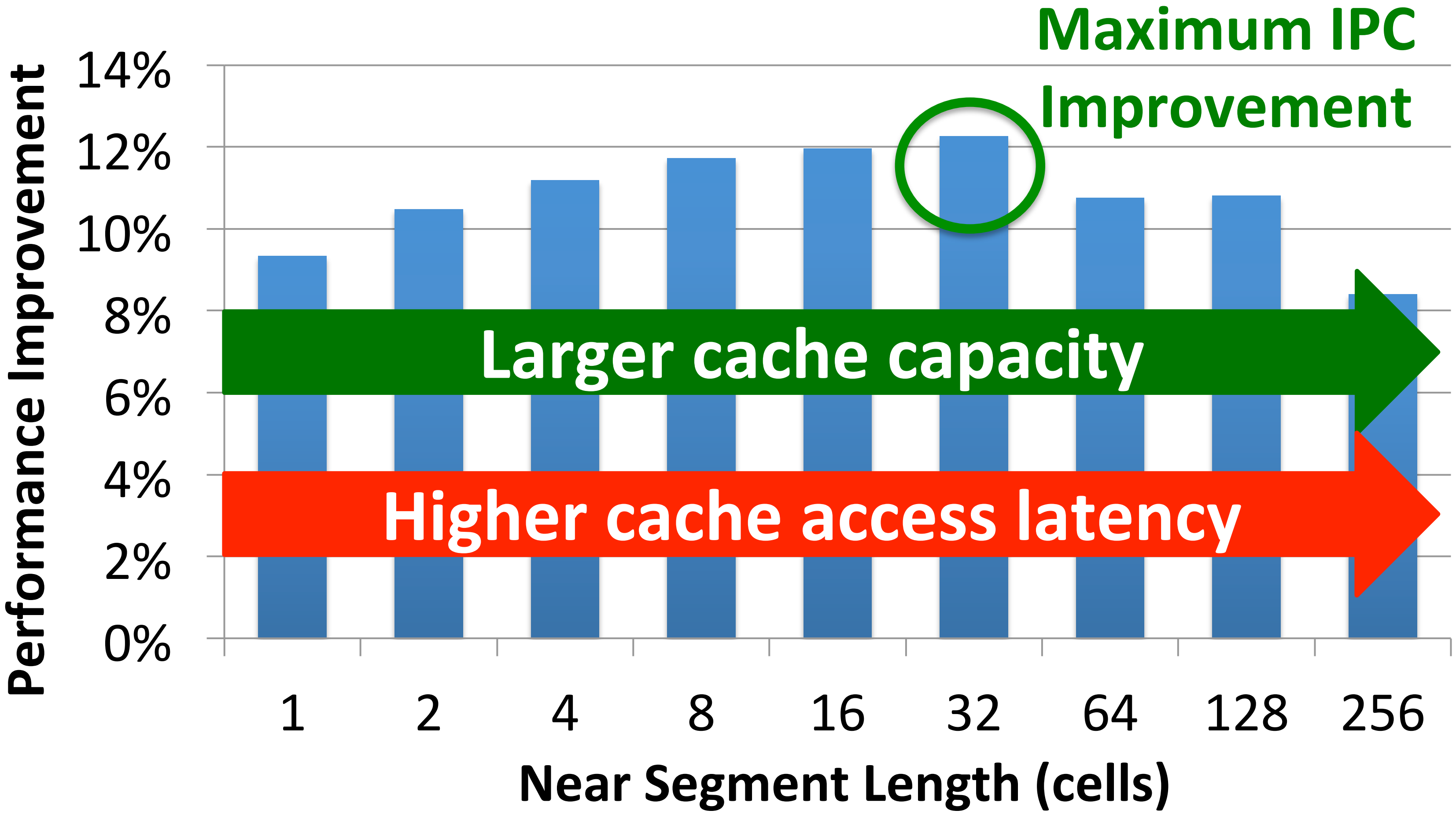
Performance & Power Consumption





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

Single-Core: Varying Near Segment Length



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

Other Mechanisms & Results

- **More mechanisms** for leveraging TL-DRAM
  - Hardware-managed ***exclusive*** caching mechanism
  - Profile-based page mapping to near segment
  - TL-DRAM improves performance and reduces power consumption with other mechanisms
- **More than two tiers**
  - Latency evaluation for three-tier TL-DRAM
- **Detailed circuit evaluation** for DRAM latency and power consumption
  - Examination of tRC and tRCD
- **Implementation details** and **storage cost analysis** in memory controller

Summary of TL-DRAM

- **Problem: DRAM latency is a critical performance bottleneck**
- **Our Goal**: Reduce DRAM latency with low area cost
- **Observation**: Long bitlines in DRAM are the dominant source of DRAM latency
- **Key Idea: Divide long bitlines into two shorter segments**
  - **Fast and slow segments**
- **Tiered-latency DRAM: Enables latency heterogeneity in DRAM**
  - **Can leverage this in many ways to improve performance and reduce power consumption**
- **Results**: When the fast segment is used as a cache to the slow segment → Significant performance improvement (>12%) and power reduction (>23%) at low area cost (3%)

New DRAM Architectures

- RAIDR: Reducing Refresh Impact
- TL-DRAM: Reducing DRAM Latency
- SALP: Reducing Bank Conflict Impact
- RowClone: Fast Bulk Data Copy and Initialization

Subarray-Level Parallelism: Reducing Bank Conflict Impact

The Memory Bank Conflict Problem

- Two requests to the same bank are serviced serially
- Problem: Costly in terms of performance and power
- Goal: We would like to reduce bank conflicts without increasing the number of banks (at low cost)

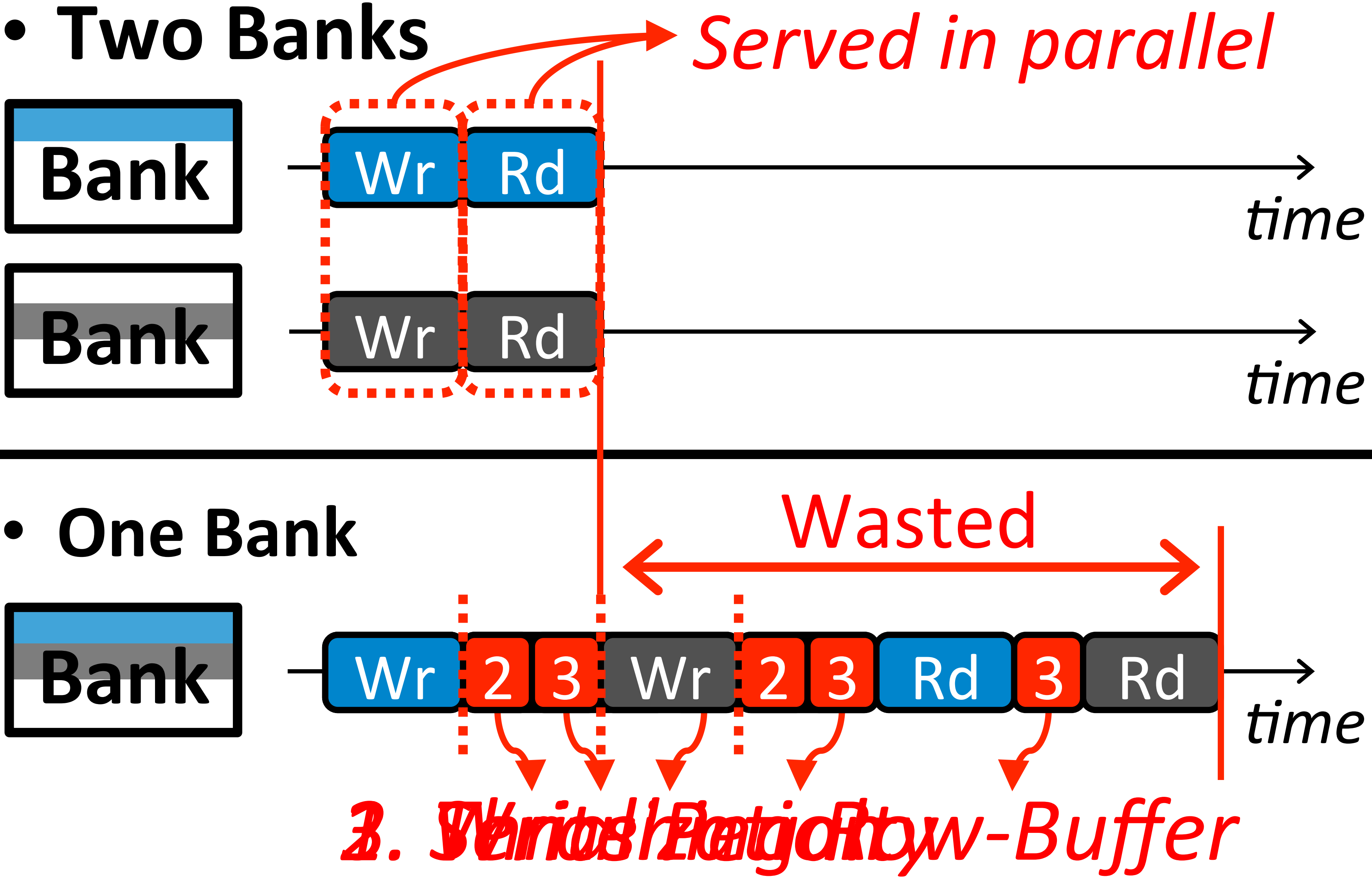
- Idea: Exploit the internal sub-array structure of a DRAM bank to parallelize bank conflicts
  - By reducing global sharing of hardware between sub-arrays

- Kim, Seshadri, Lee, Liu, Mutlu, "A Case for Exploiting Subarray-Level Parallelism in DRAM," ISCA 2012.

The Problem with Memory Bank Conflicts

- Two Banks

- One Bank



Goal

- **Goal:** *Mitigate the detrimental effects of **bank conflicts** in a cost-effective manner*

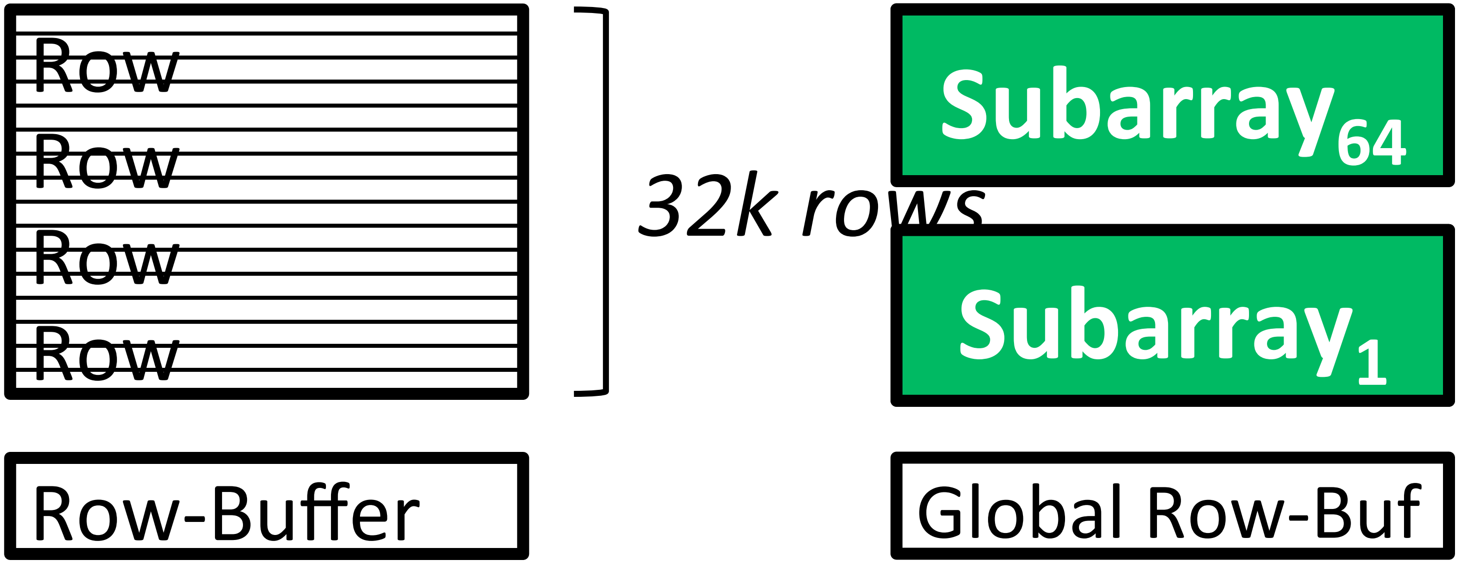
- **Naïve solution:** Add more banks
  - Very expensive

- **Cost-effective solution:** Approximate the benefits of more banks without adding more banks

Key Observation #1

A DRAM bank is divided into ***subarrays***

Logical Bank ***Physical Bank***



Row

Row

Row

Row

32k rows

Subarray$_{64}$

Subarray$_{1}$

Row-Buffer

Global Row-Buf

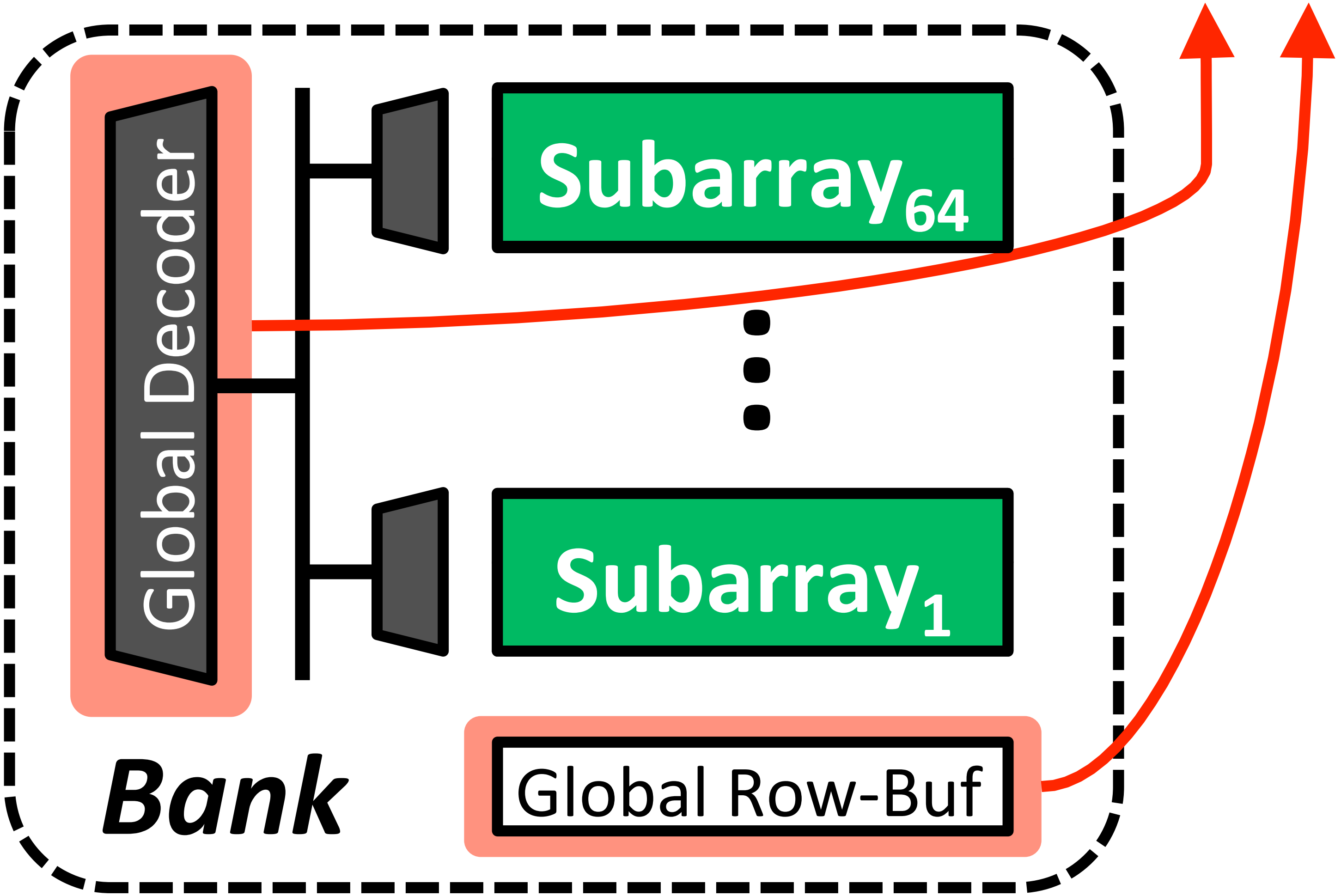
A *single* row-buffer cannot drive *all* rows

Many ***local row-buffers***, one at each ***subarray***

Key Observation #2

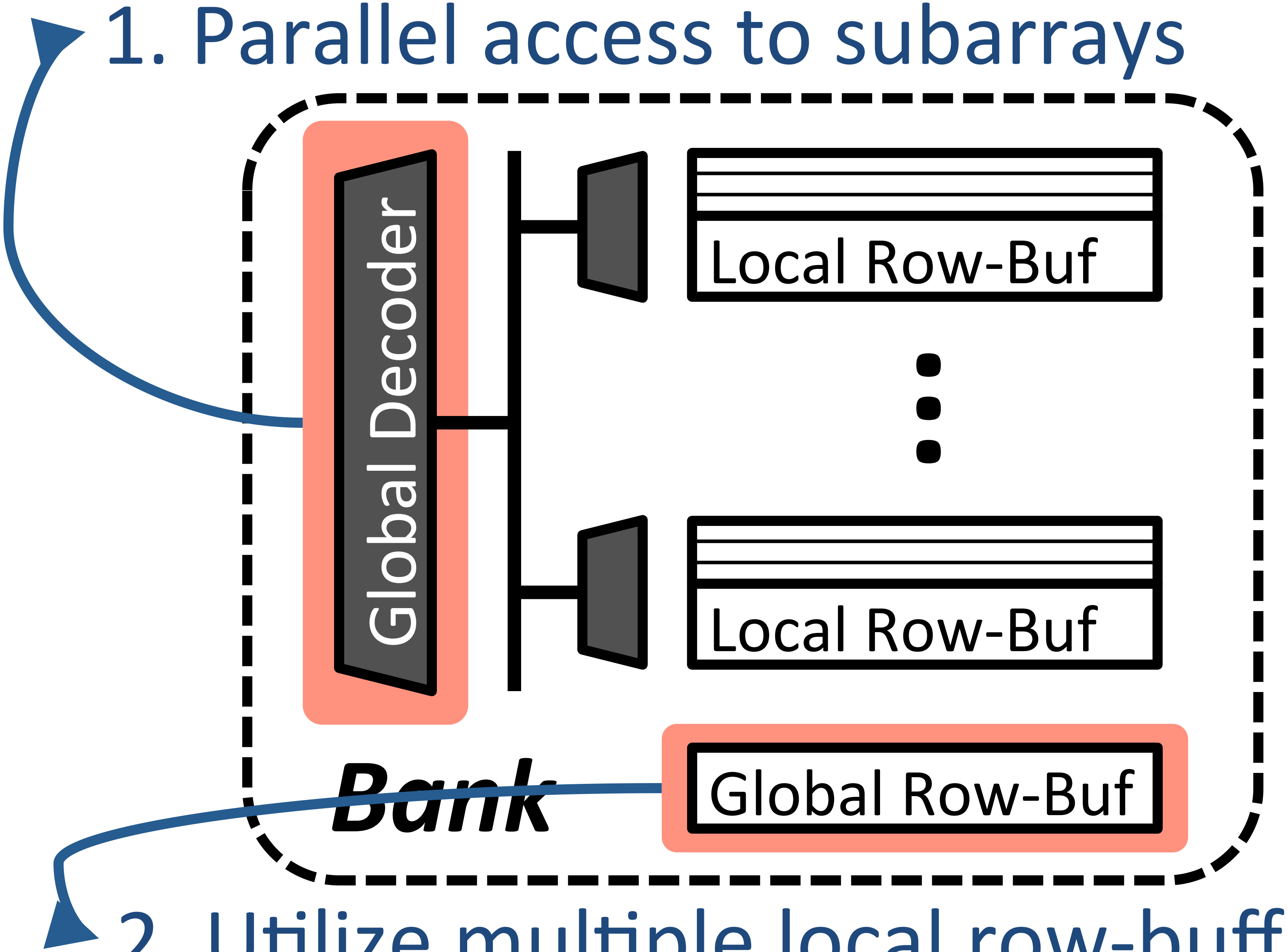
Each subarray is mostly independent…

- except occasionally sharing ***global structures***



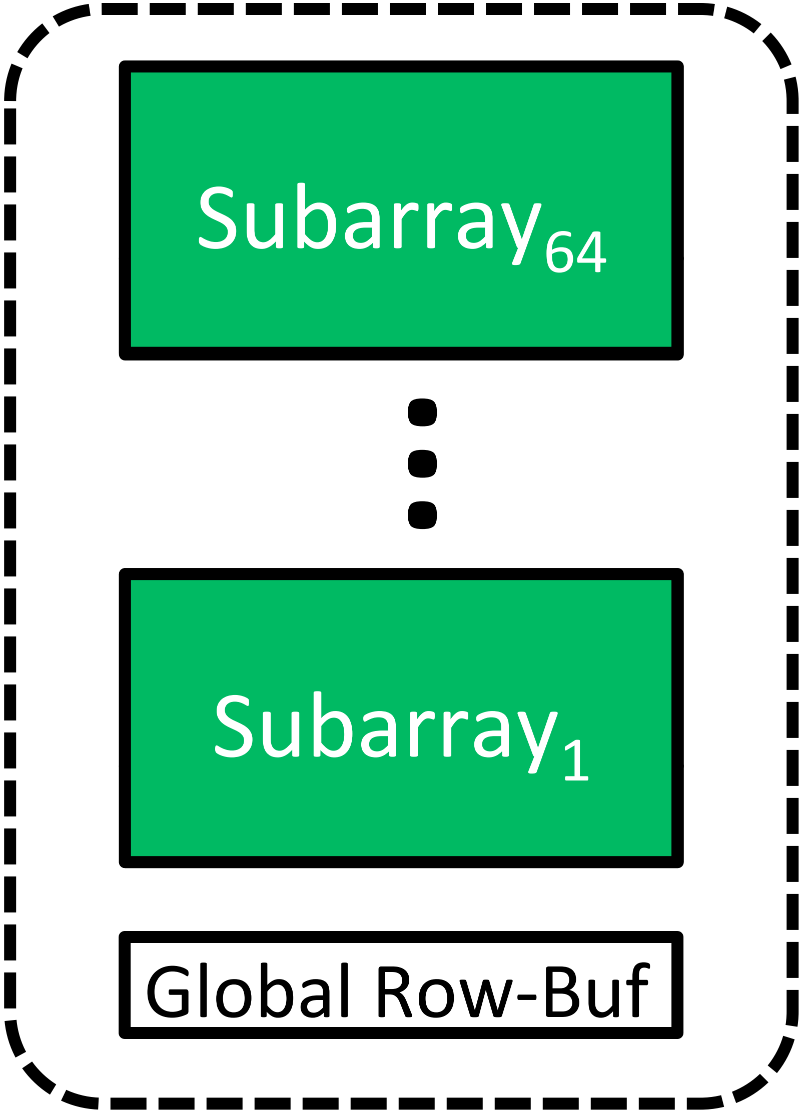
Key Idea: Reduce Sharing of Globals

1. Parallel access to subarrays



2. Utilize multiple local row-buffers

Overview of Our Mechanism



1. Parallelize

2. Utilize multiple local row-buffers

To same bank...

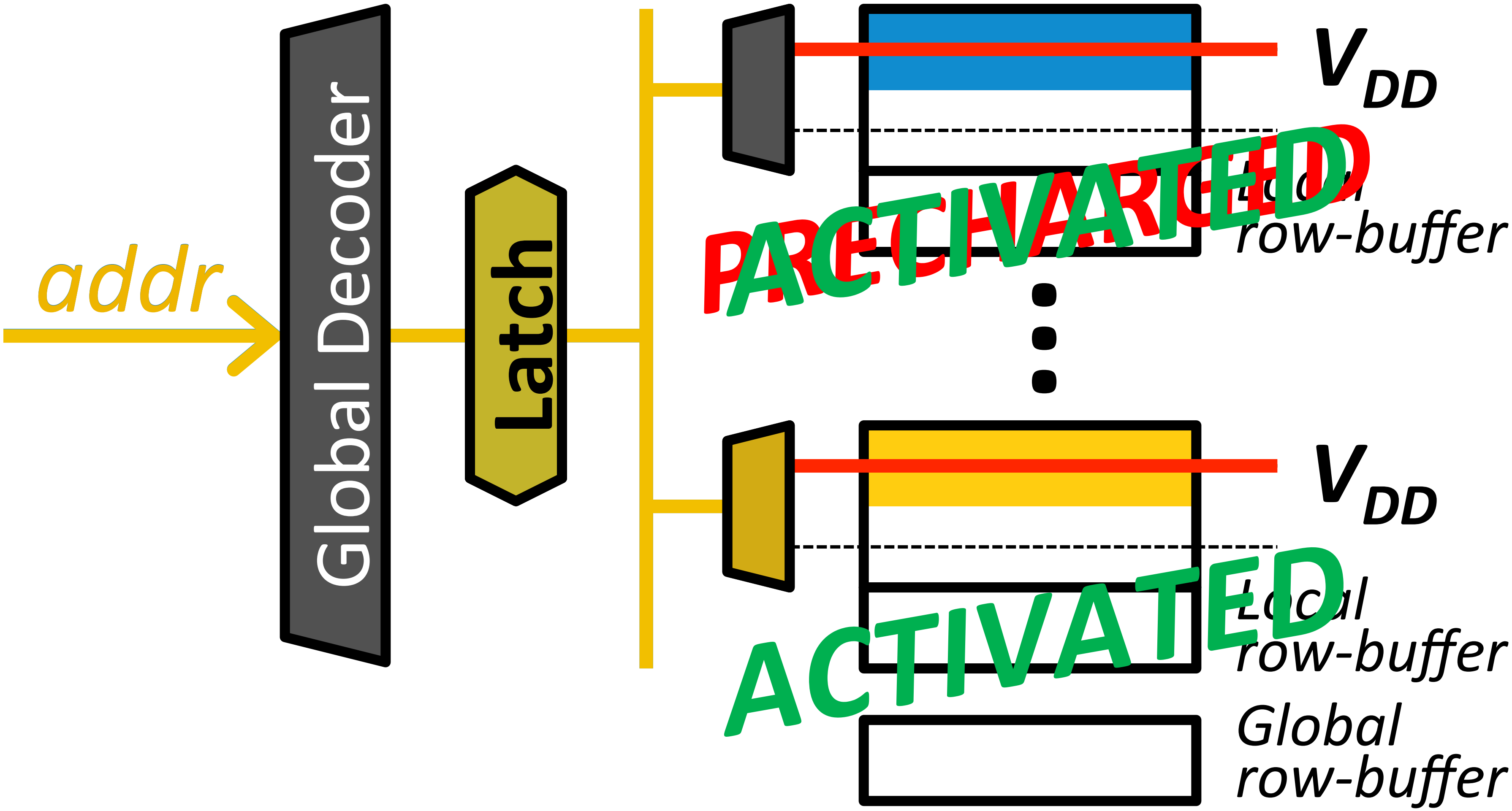
but diff. subarrays

Challenges: Global Structures

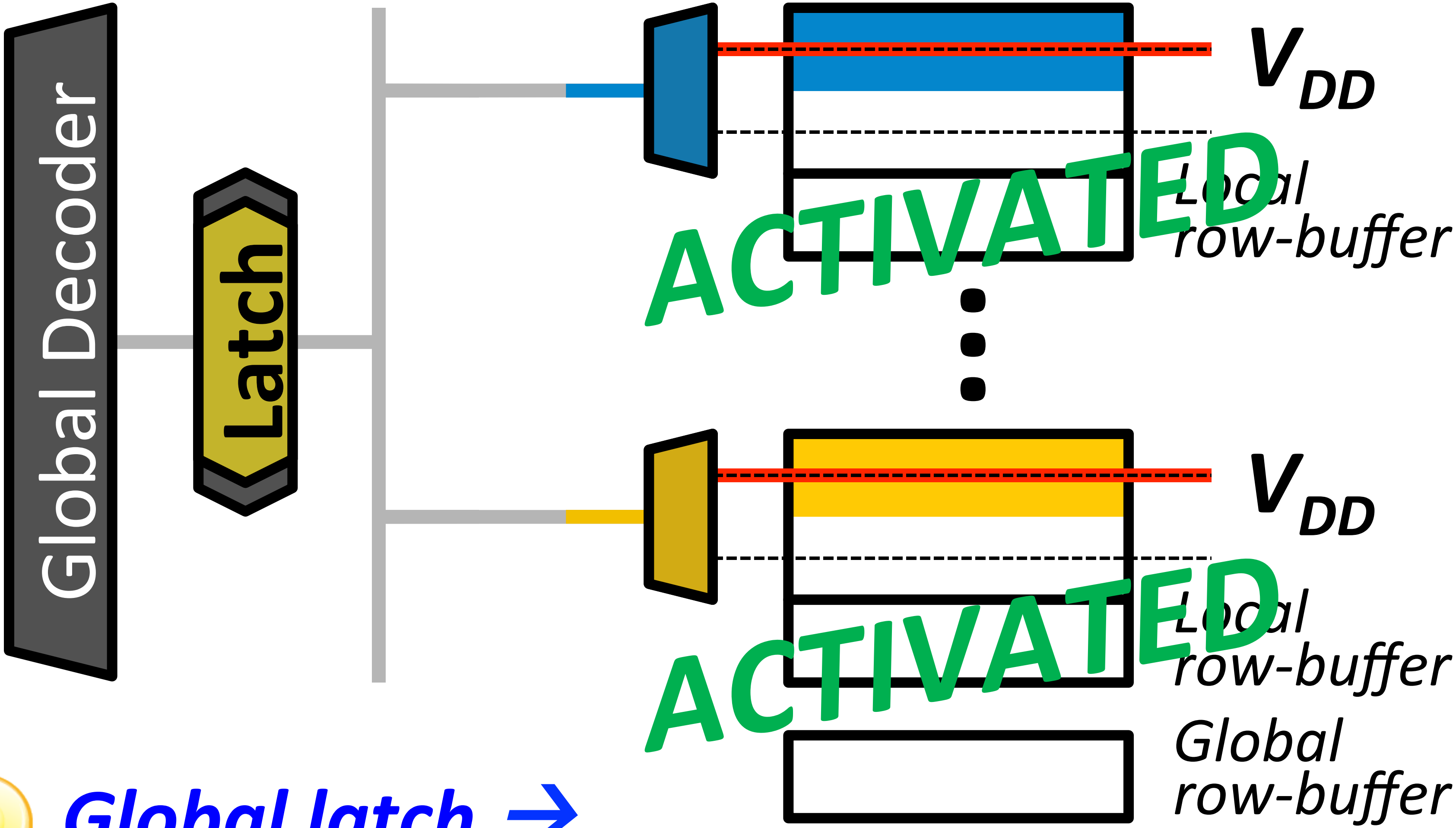
1. Global Address Latch

2. Global Bitlines

Challenge #1. Global Address Latch



Solution #1. Subarray Address Latch

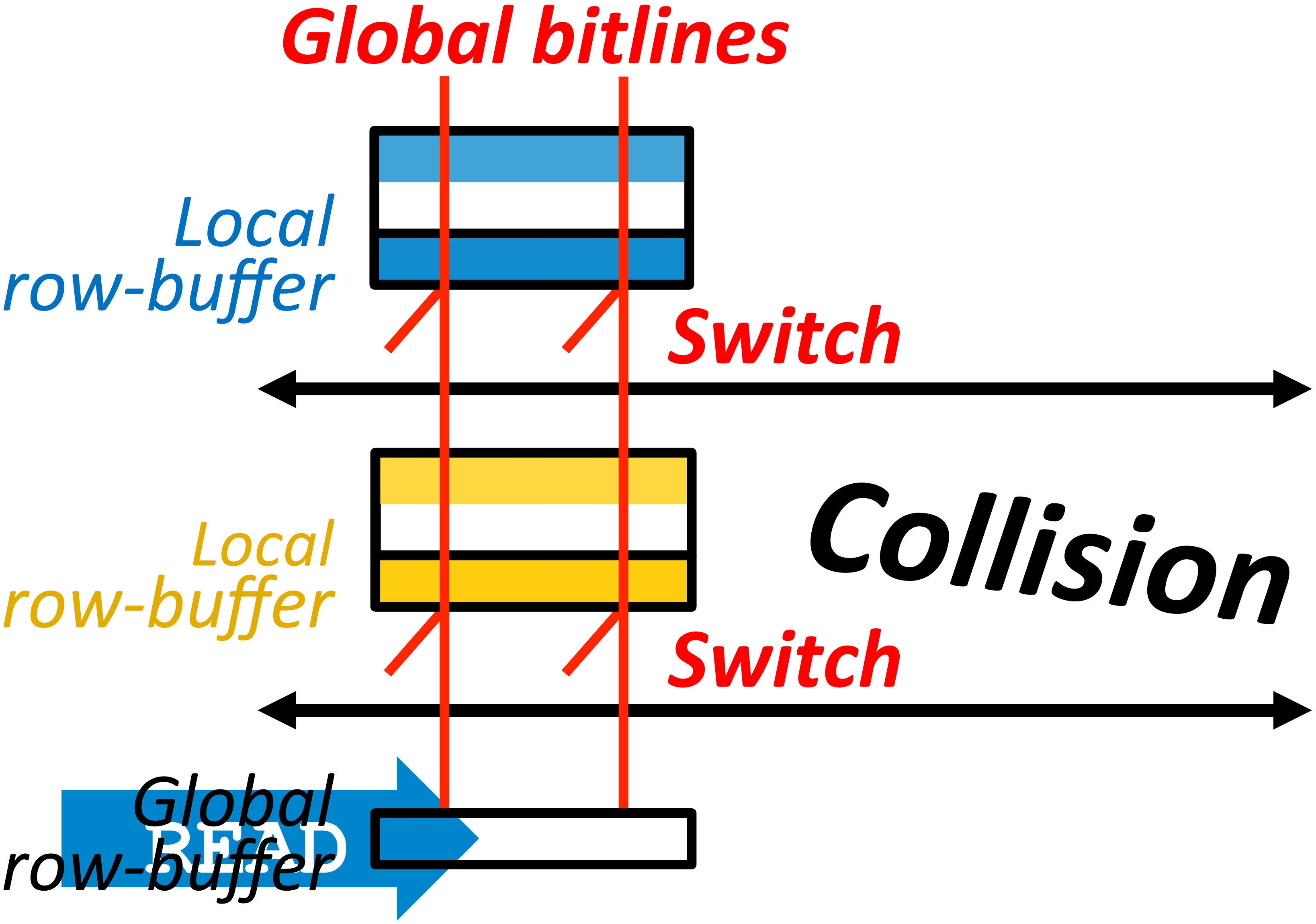


Global latch → local latches

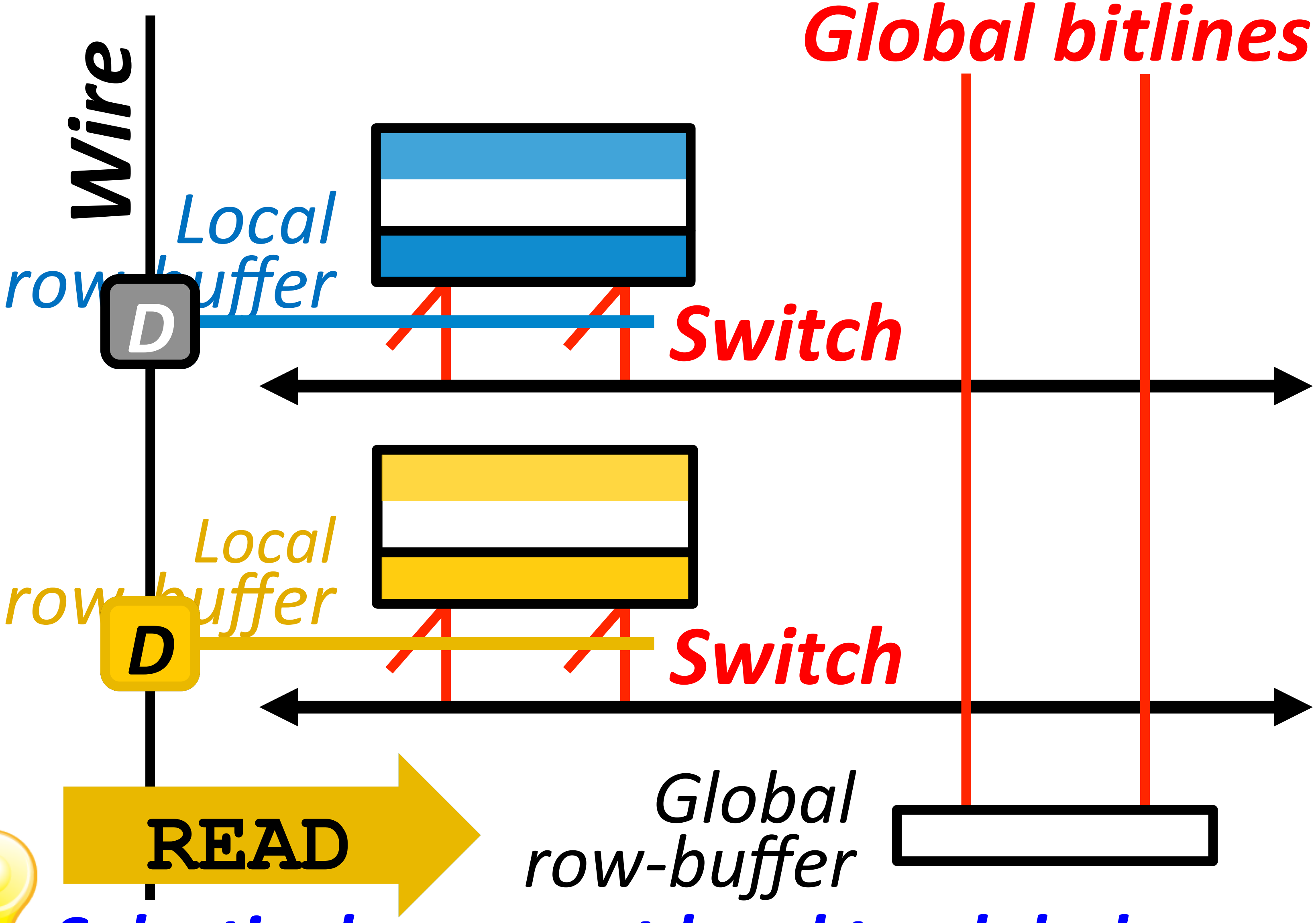
Challenges: Global Structures

1. **Global Address Latch**
   - Problem: Only *one* raised wordline
   - Solution: **Subarray Address Latch**
2. **Global Bitlines**

Challenge #2. Global Bitlines



Solution #2. Designated-Bit Latch



Selectively connect local to global

Challenges: Global Structures

1. Global Address Latch

- Problem: Only *one* raised wordline
- Solution: **Subarray Address Latch**

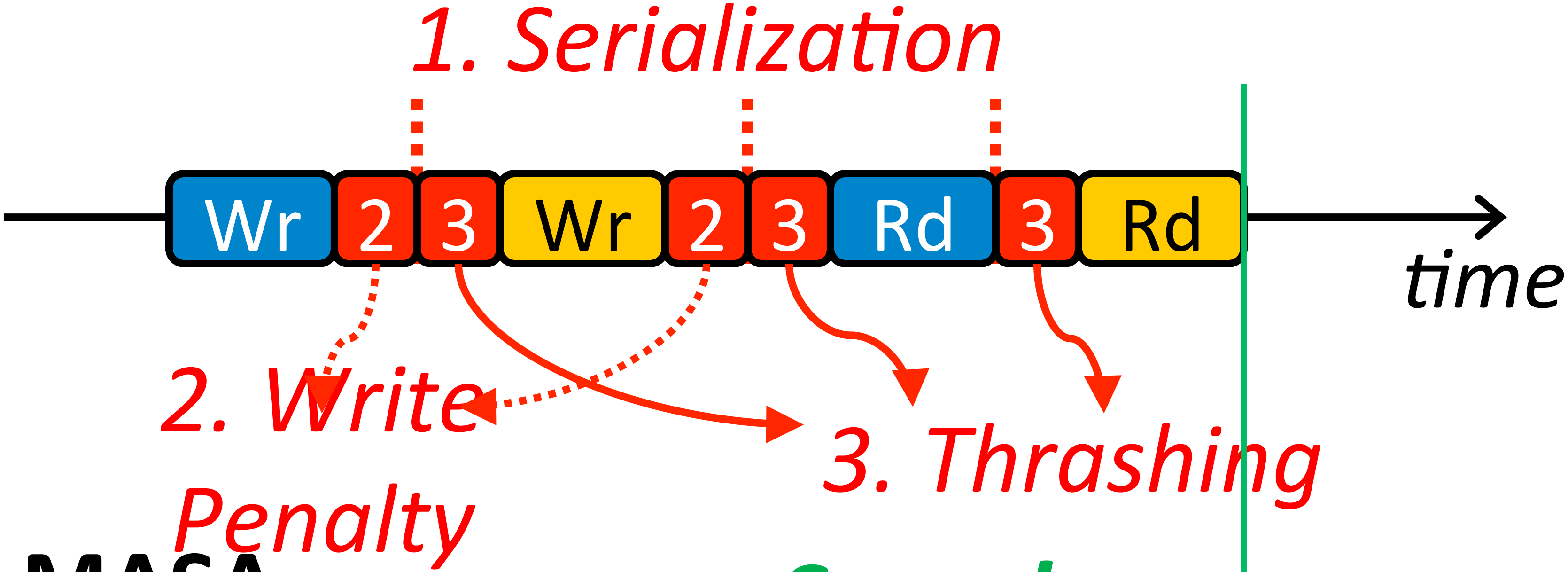
2. Global Bitlines

- Problem: Collision during access
- Solution: **Designated-Bit Latch**

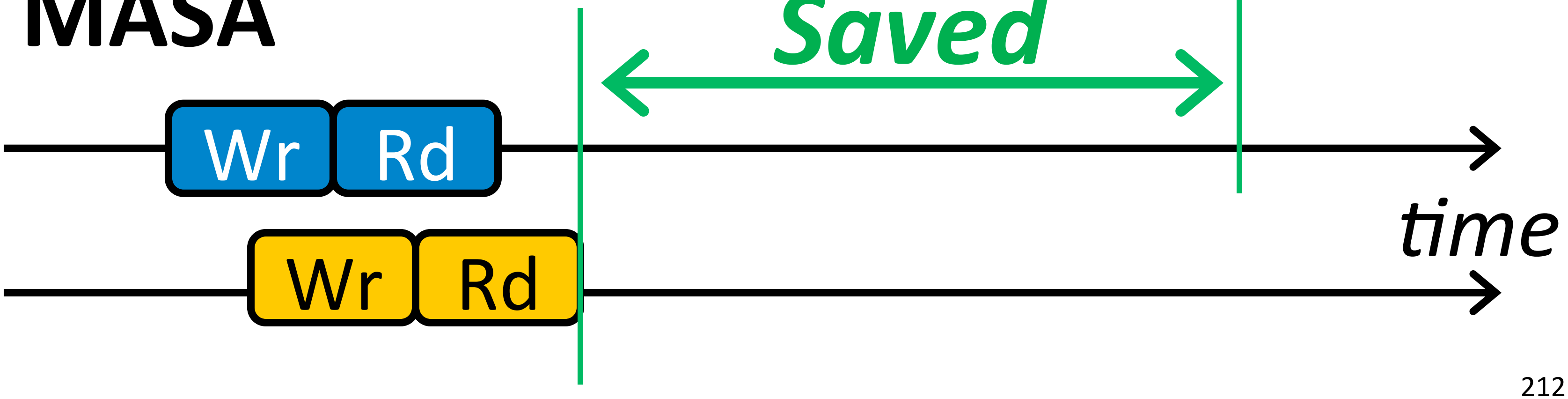
MASA (Multitude of Activated Subarrays)

MASA: Advantages

- Baseline (Subarray-Oblivious)



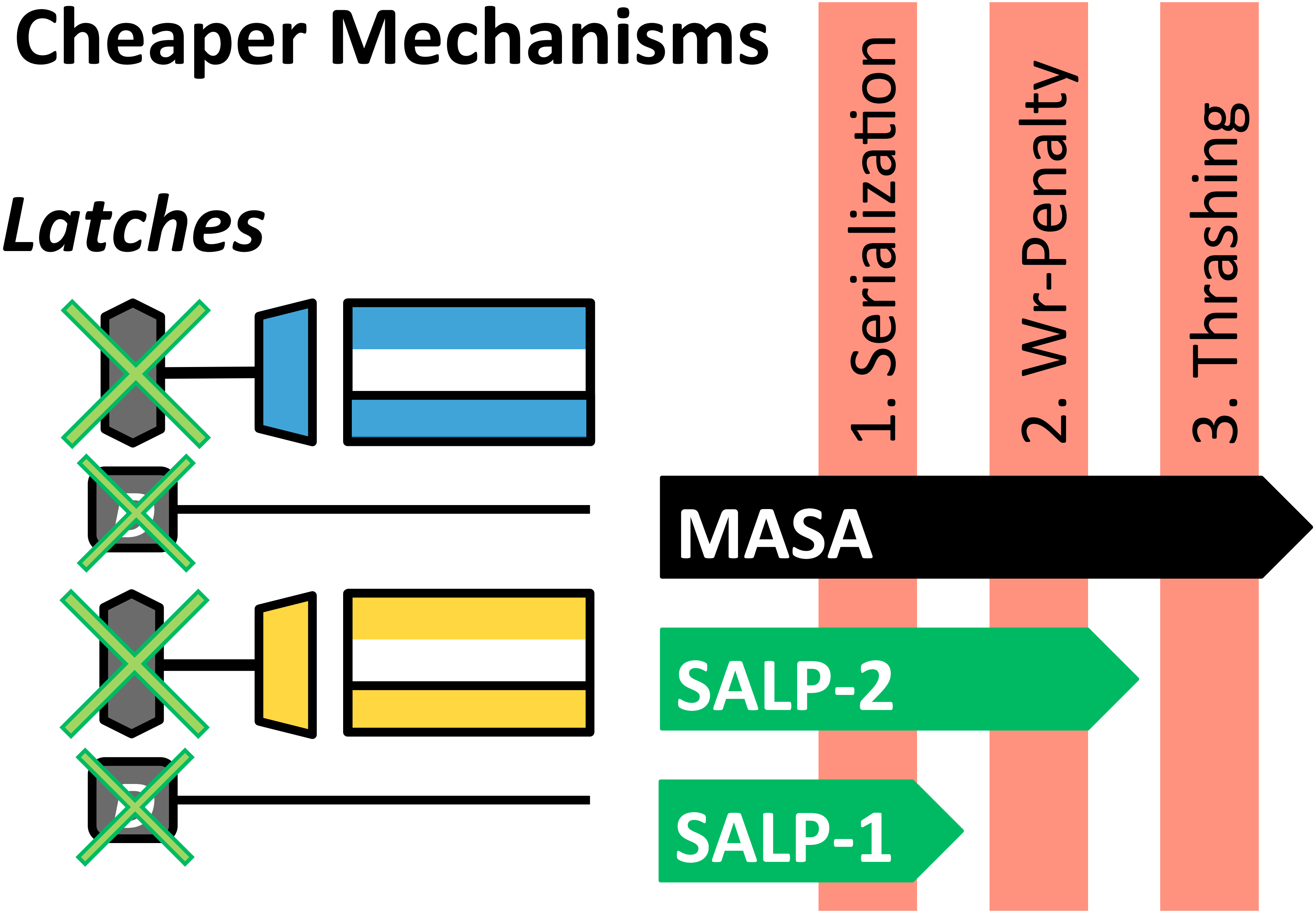
- **MASA**



MASA: Overhead

- **DRAM Die Size**: Only **0.15%** increase
  - Subarray Address Latches
  - Designated-Bit Latches & Wire
- **DRAM Static Energy:** Small increase
  - **0.56mW** for each activated subarray
  - *But saves dynamic energy*
- **Controller**: Small additional storage
  - Keep track of subarray status (< **256B**)
  - Keep track of new timing constraints

Cheaper Mechanisms



Latches

1. Serialization

2. Wr-Penalty

3. Thrashing

MASA

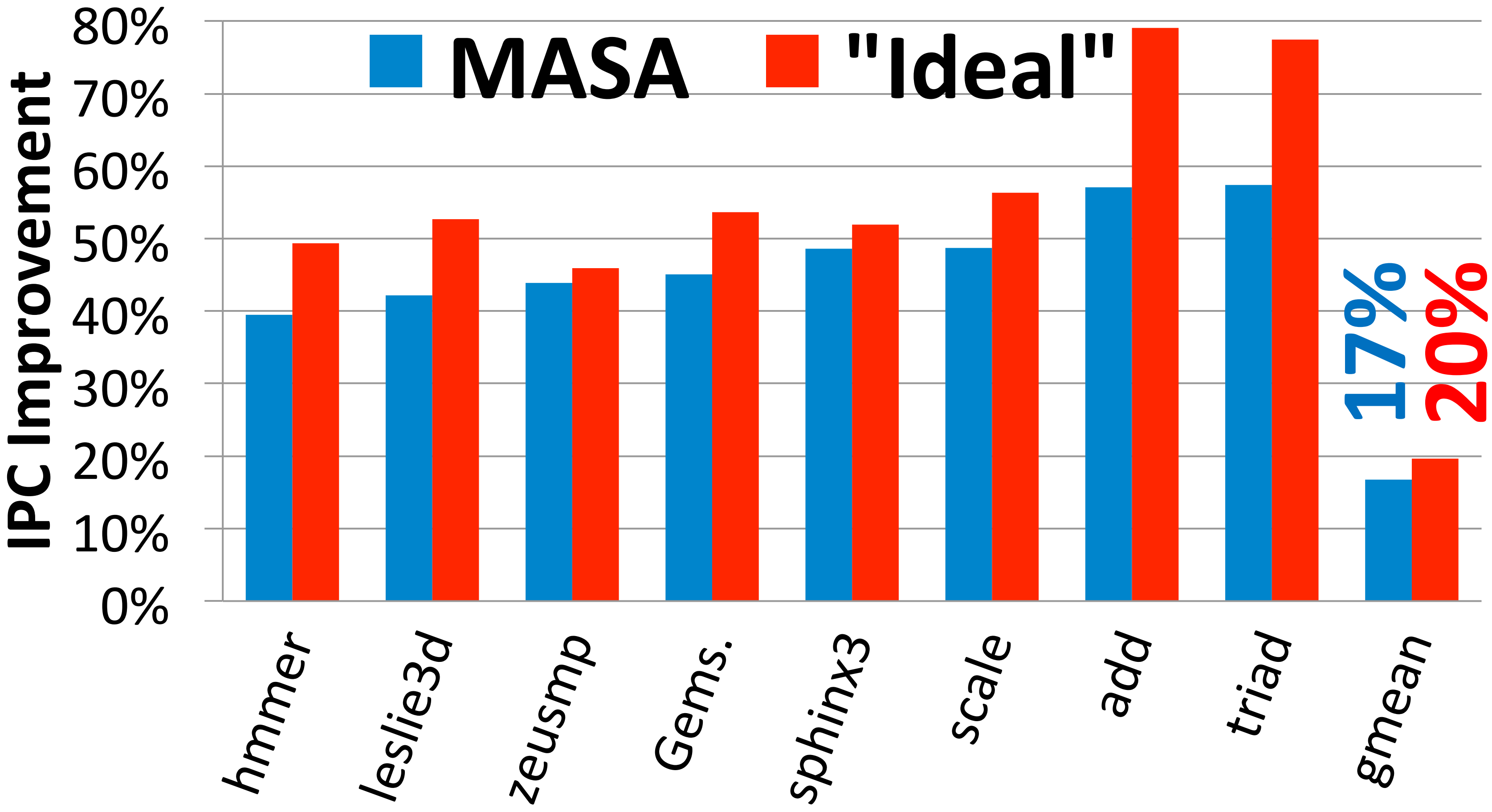
SALP-2

SALP-1

System Configuration

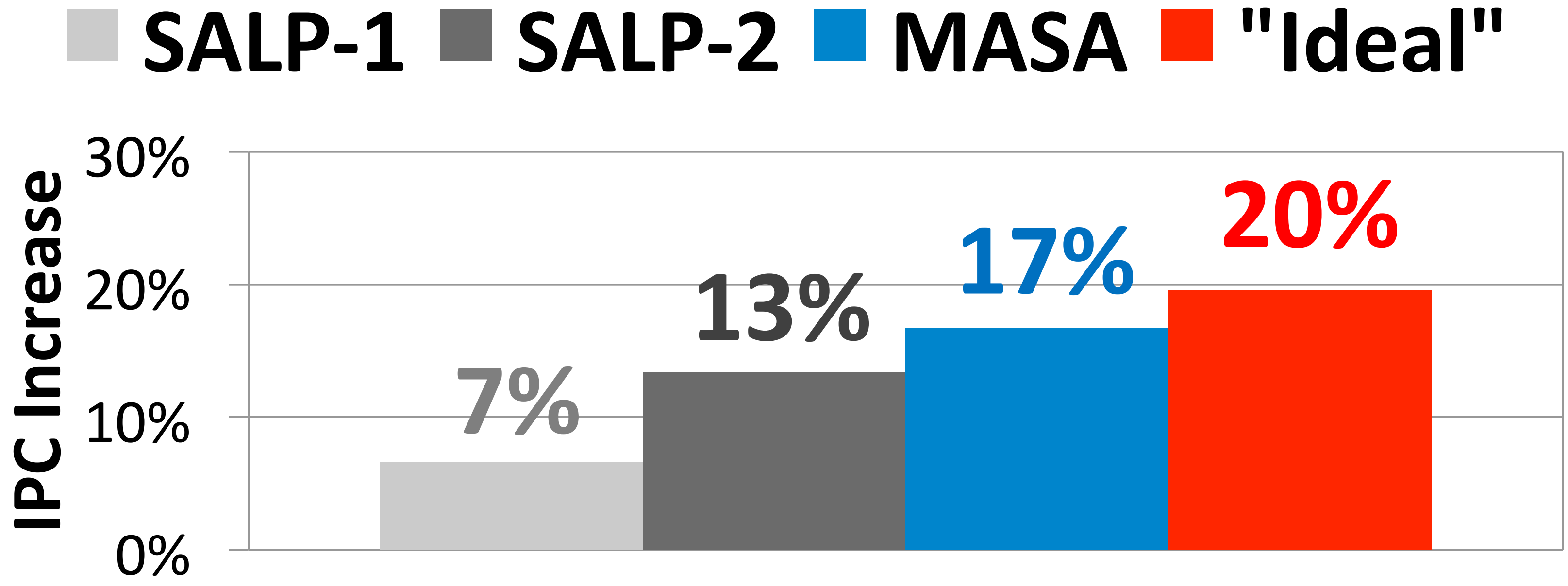
- **System Configuration**
  - CPU: 5.3GHz, 128 ROB, 8 MSHR
  - LLC: 512kB per-core slice
- **Memory Configuration**
  - DDR3-1066
  - ***(default)*** **1 channel, 1 rank, 8 banks, 8 subarrays-per-bank**
  - *(sensitivity)* 1-8 chans, 1-8 ranks, 8-64 banks, 1-128 subarrays
- **Mapping & Row-Policy**
  - ***(default)*** **Line-interleaved & Closed-row**
  - *(sensitivity)* Row-interleaved & Open-row
- **DRAM Controller Configuration**
  - 64-/64-entry read/write queues per-channel
  - FR-FCFS, batch scheduling for writes

SALP: Single-core Results



***MASA** achieves most of the benefit of having more banks (“Ideal”)*

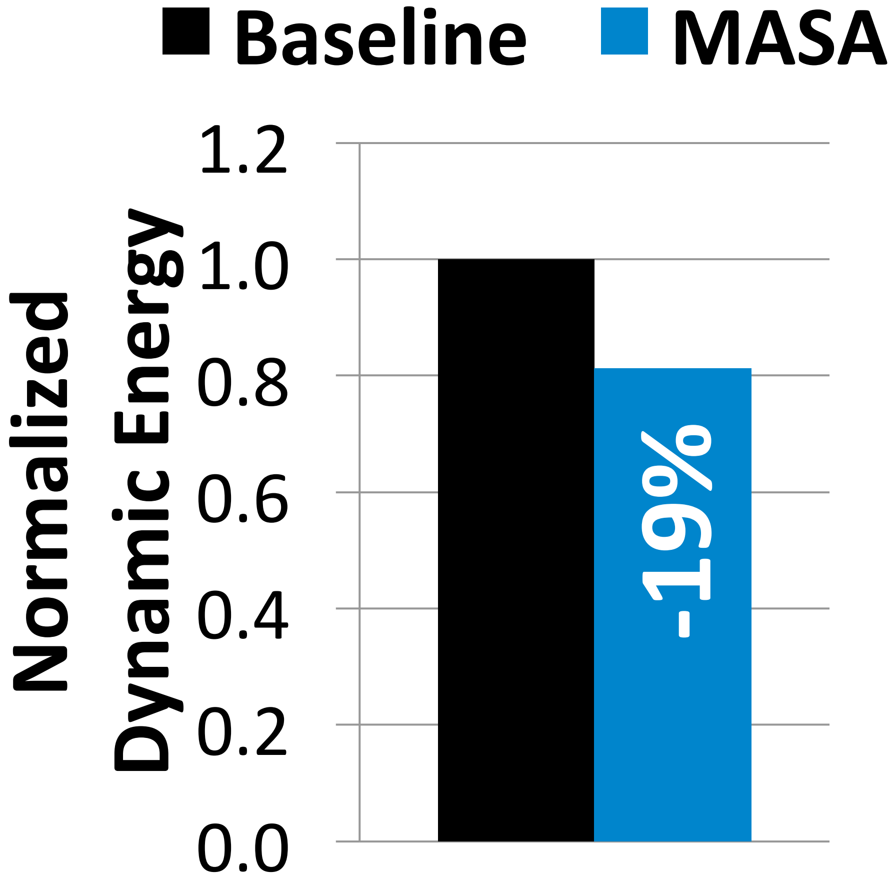
SALP: Single-Core Results

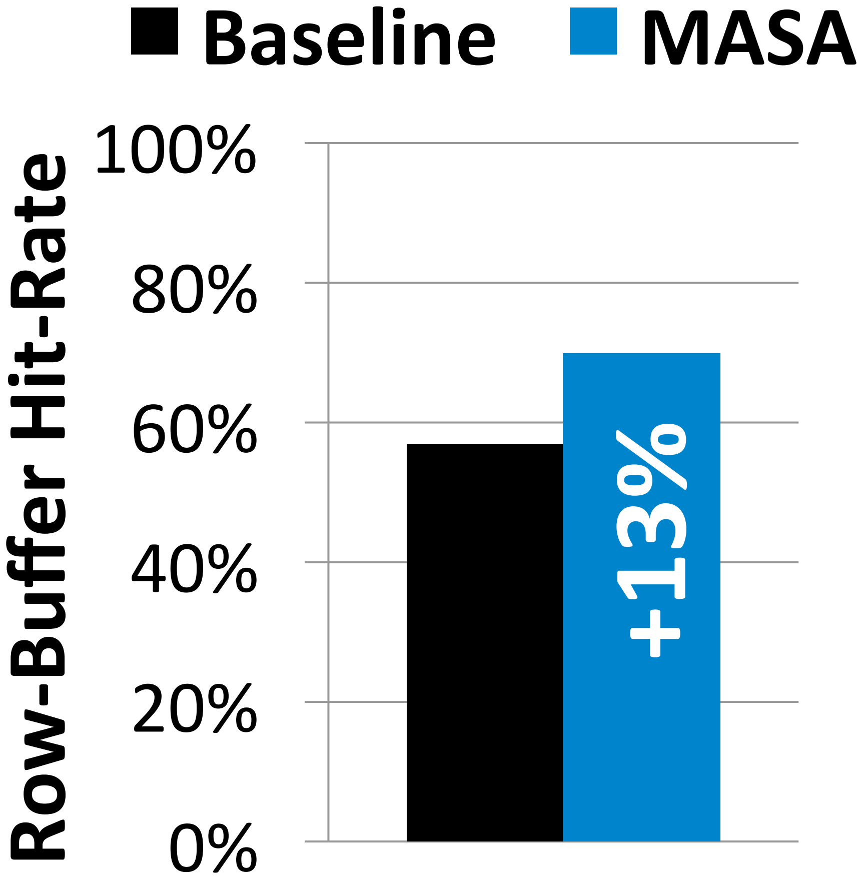


DRAM Die Area	< 0.15%	0.15%	36.3%

SALP-1, SALP-2, MASA *improve performance at low cost*

Subarray-Level Parallelism: Results





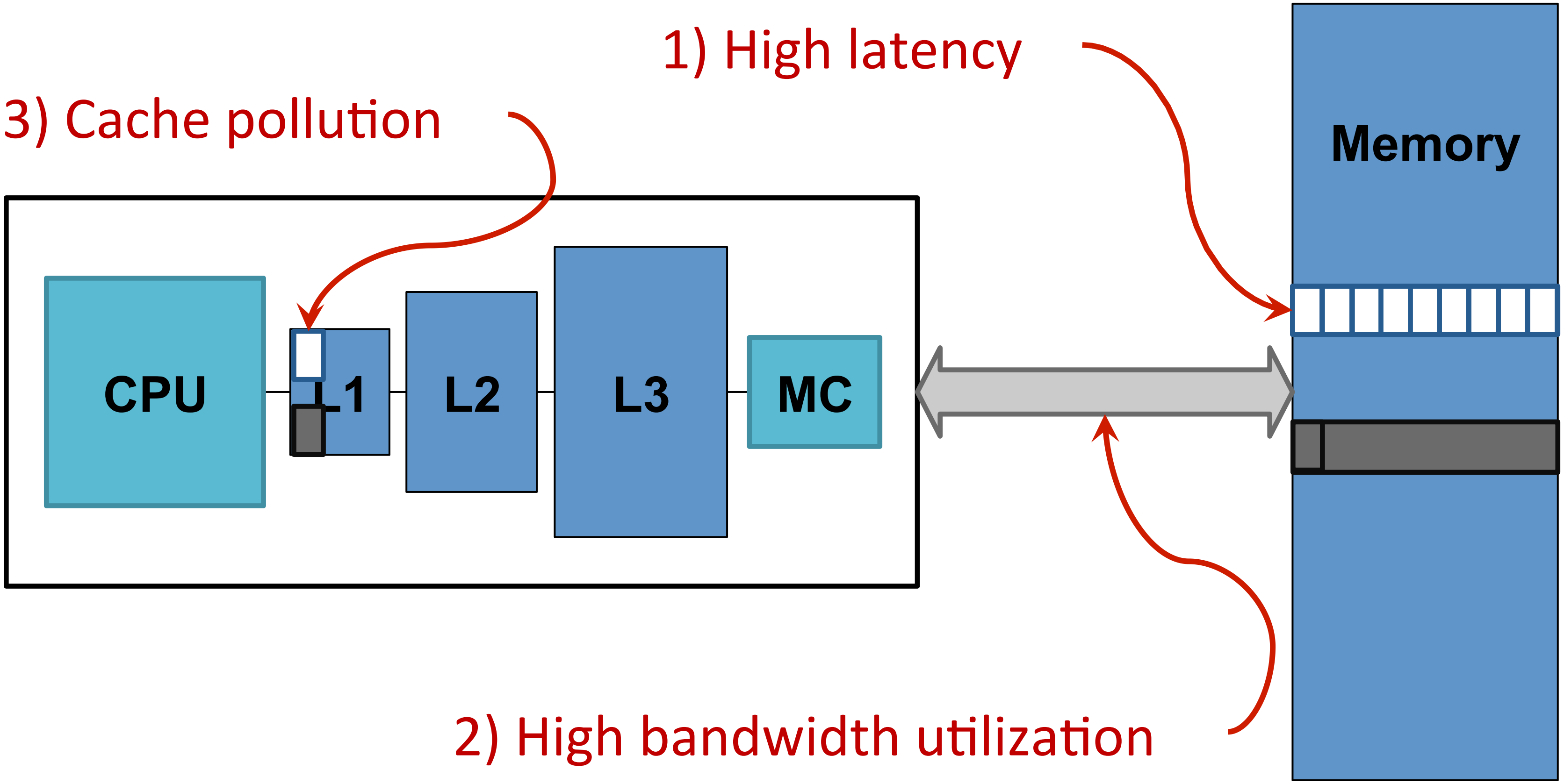
***MASA** increases energy-efficiency*

New DRAM Architectures

- RAIDR: Reducing Refresh Impact
- TL-DRAM: Reducing DRAM Latency
- SALP: Reducing Bank Conflict Impact
- RowClone: Fast Bulk Data Copy and Initialization

RowClone: Fast Bulk Data Copy and Initialization

Today's Memory: Bulk Data Copy



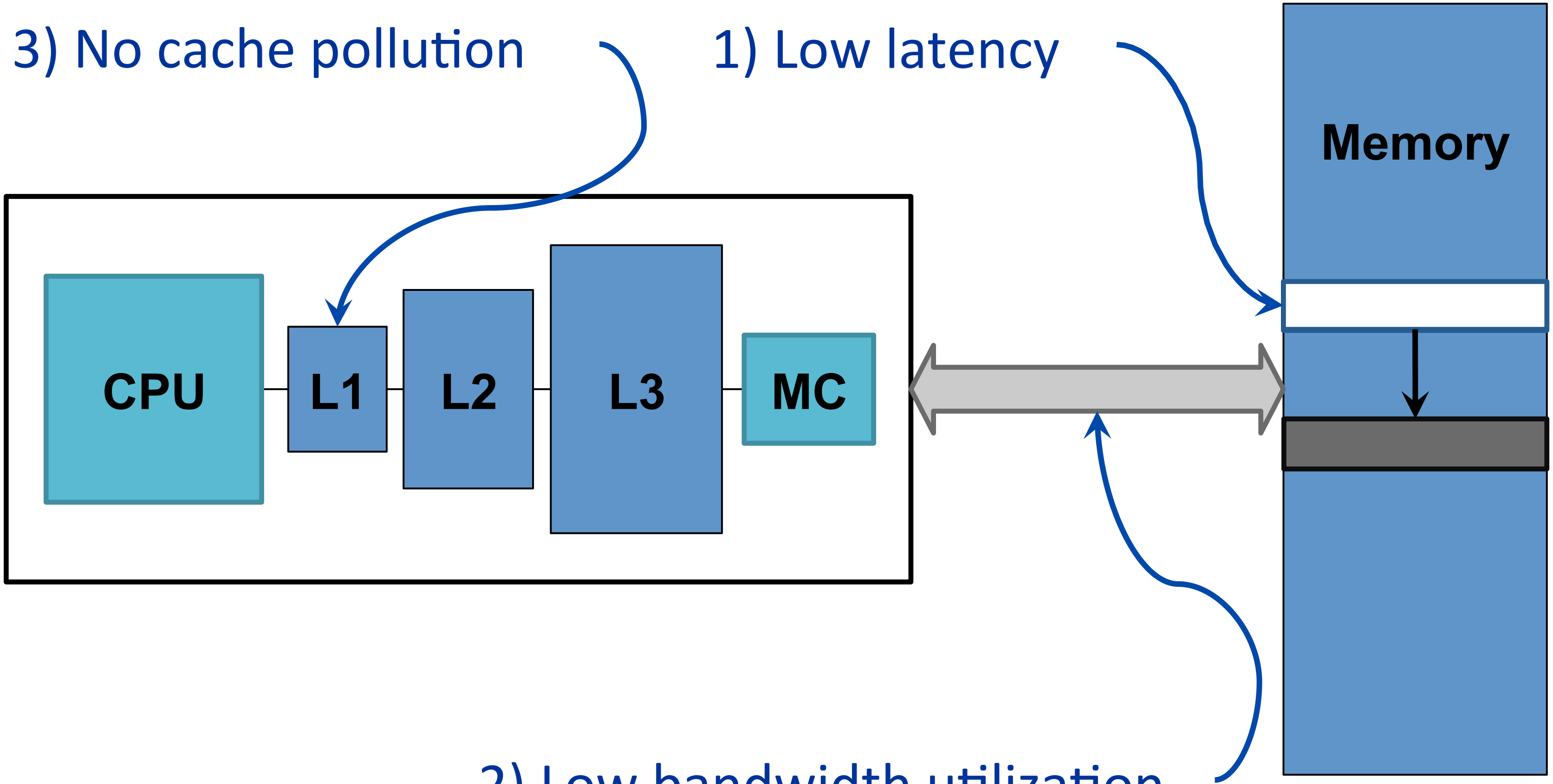
1) High latency

3) Cache pollution

2) High bandwidth utilization

4) Unwanted data movement

Future: RowClone (In-Memory Copy)



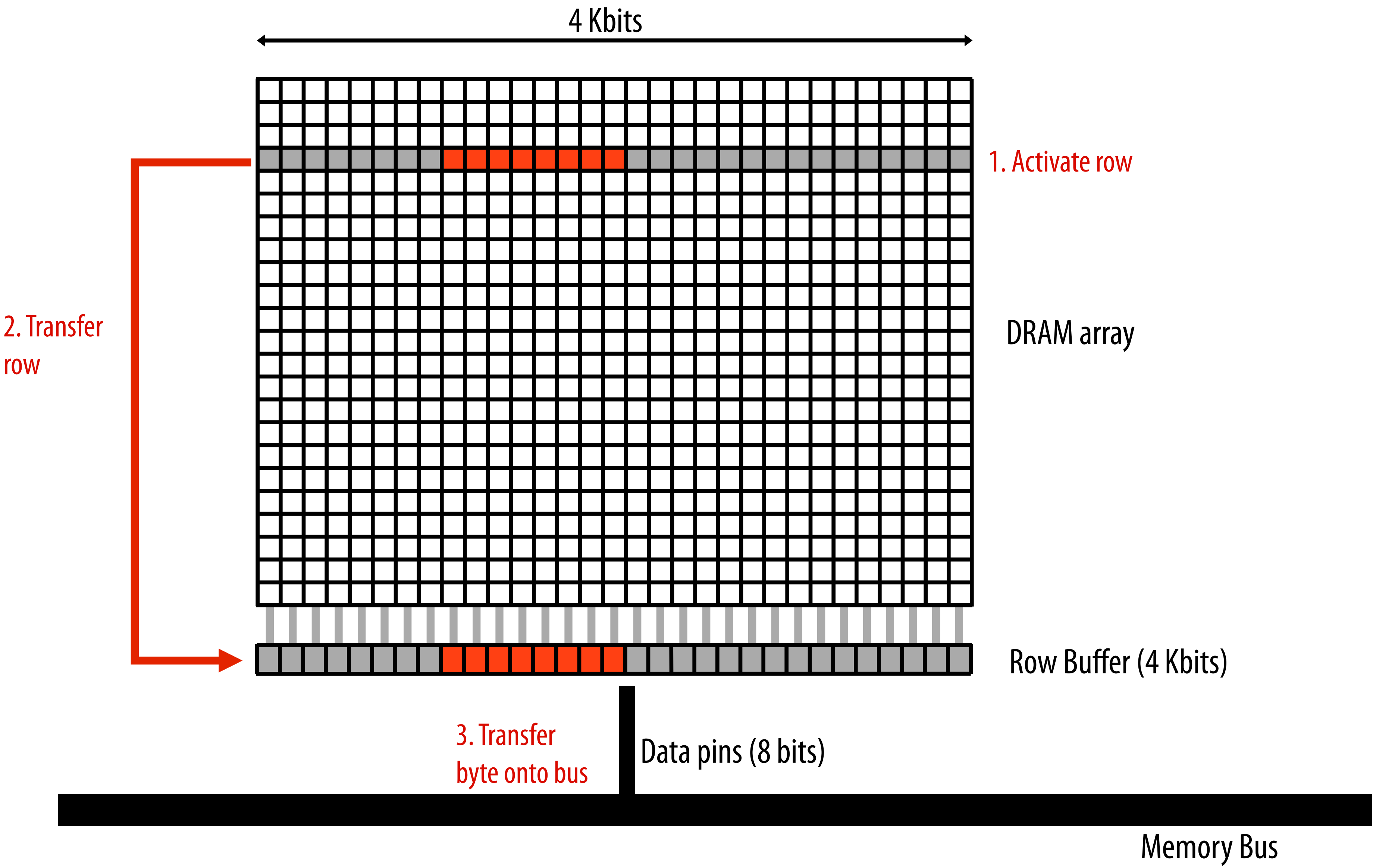
3) No cache pollution

1) Low latency

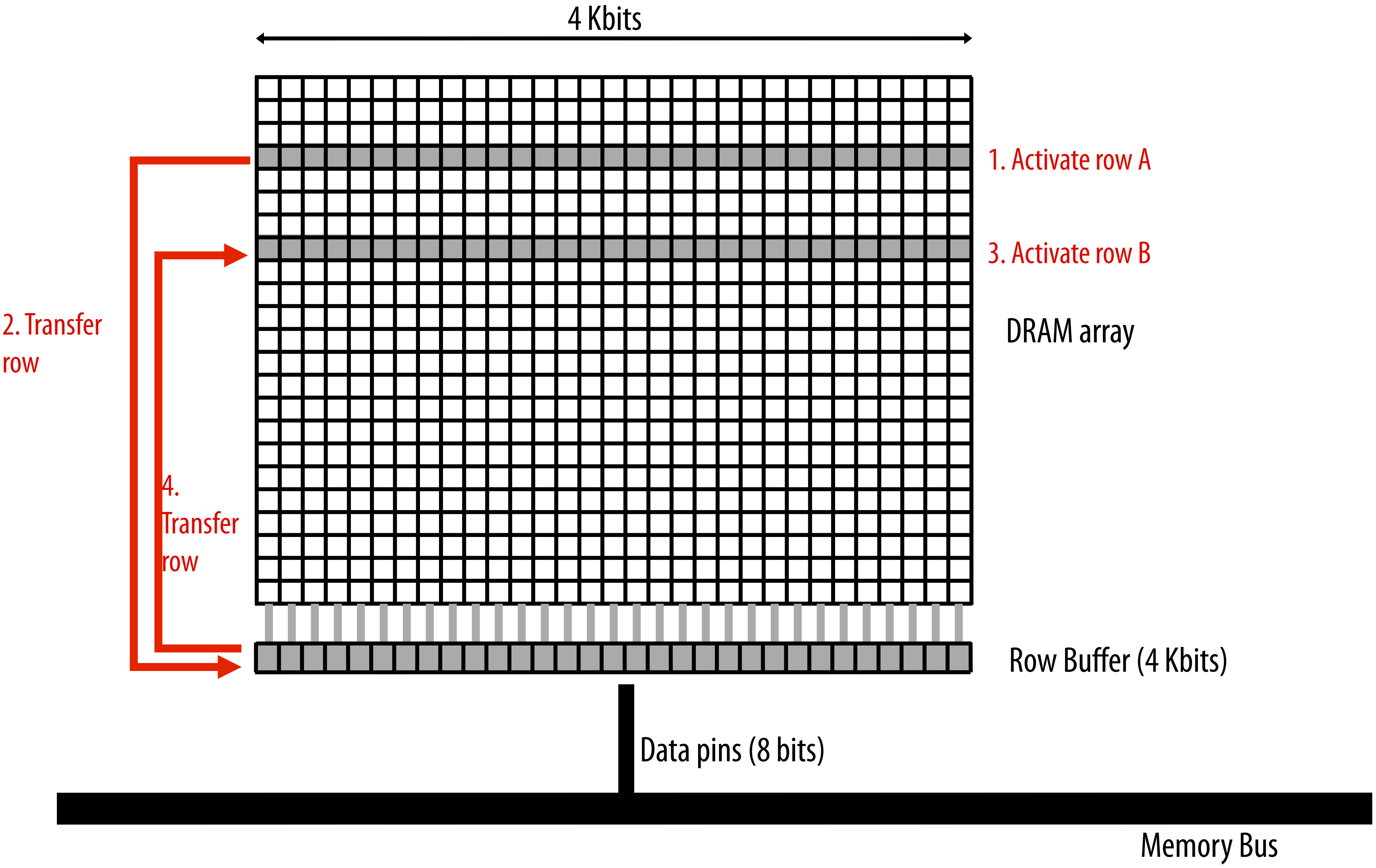
2) Low bandwidth utilization

4) No unwanted data movement

DRAM operation (load one byte)



RowClone: in-DRAM Row Copy (and Initialization)

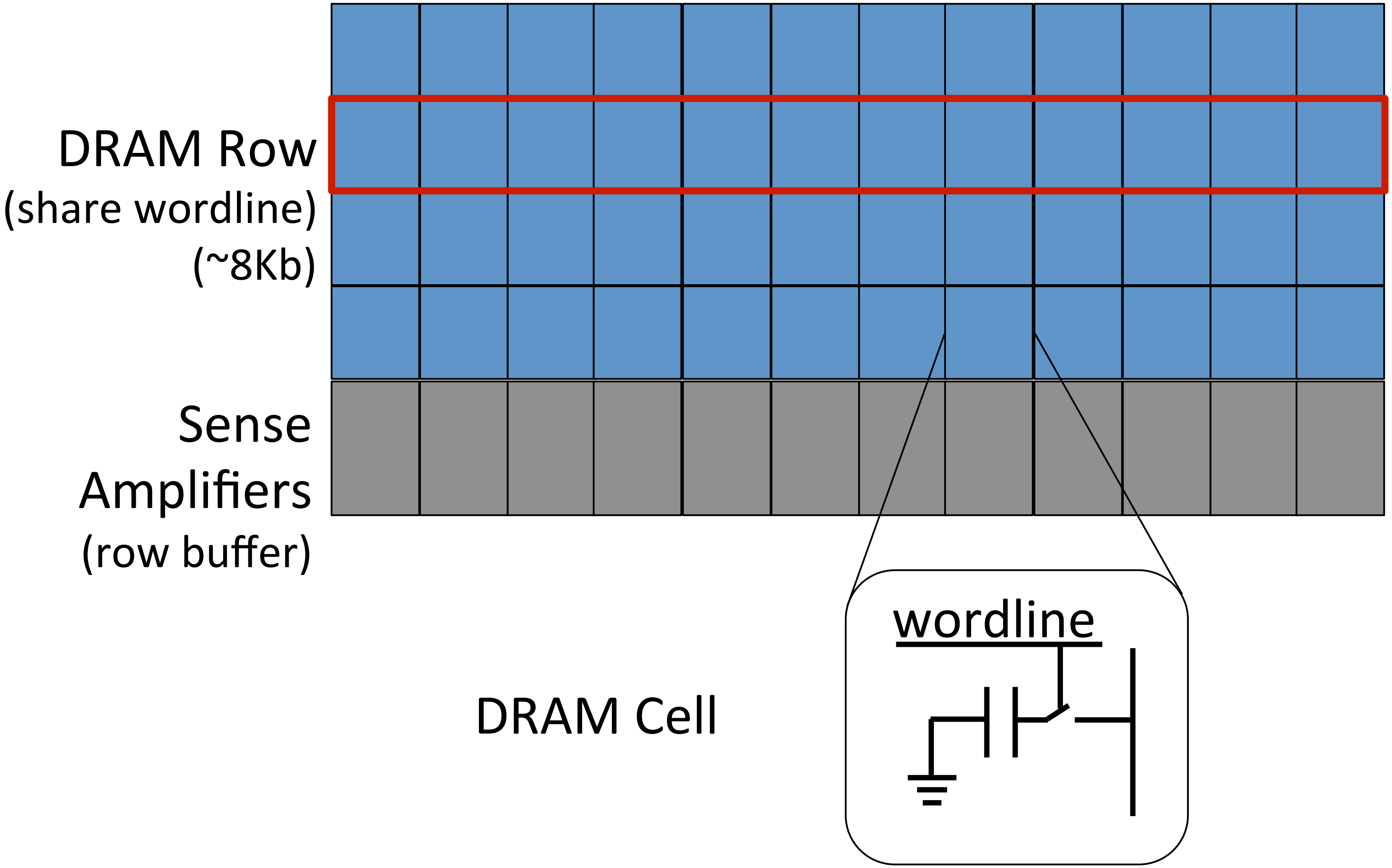


Our Approach: Key Idea

- DRAM banks contain
  1. Mutiple rows of DRAM cells – row = 8KB
  2. A row buffer shared by the DRAM rows

- Large scale copy
  1. Copy data from source row to row buffer
  2. Copy data from row buffer to destination row

DRAM Subarray Microarchitecture

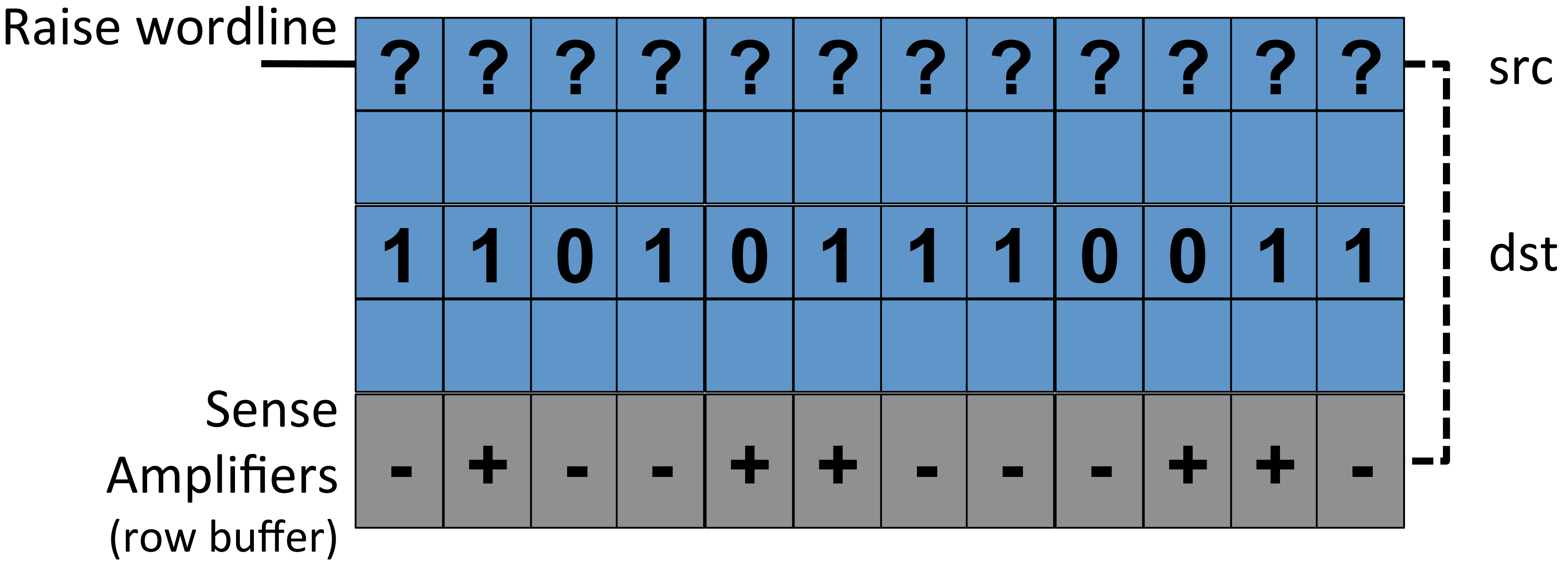


DRAM Row
(share wordline)
(~8Kb)

Sense
Amplifiers
(row buffer)

DRAM Cell

DRAM Operation



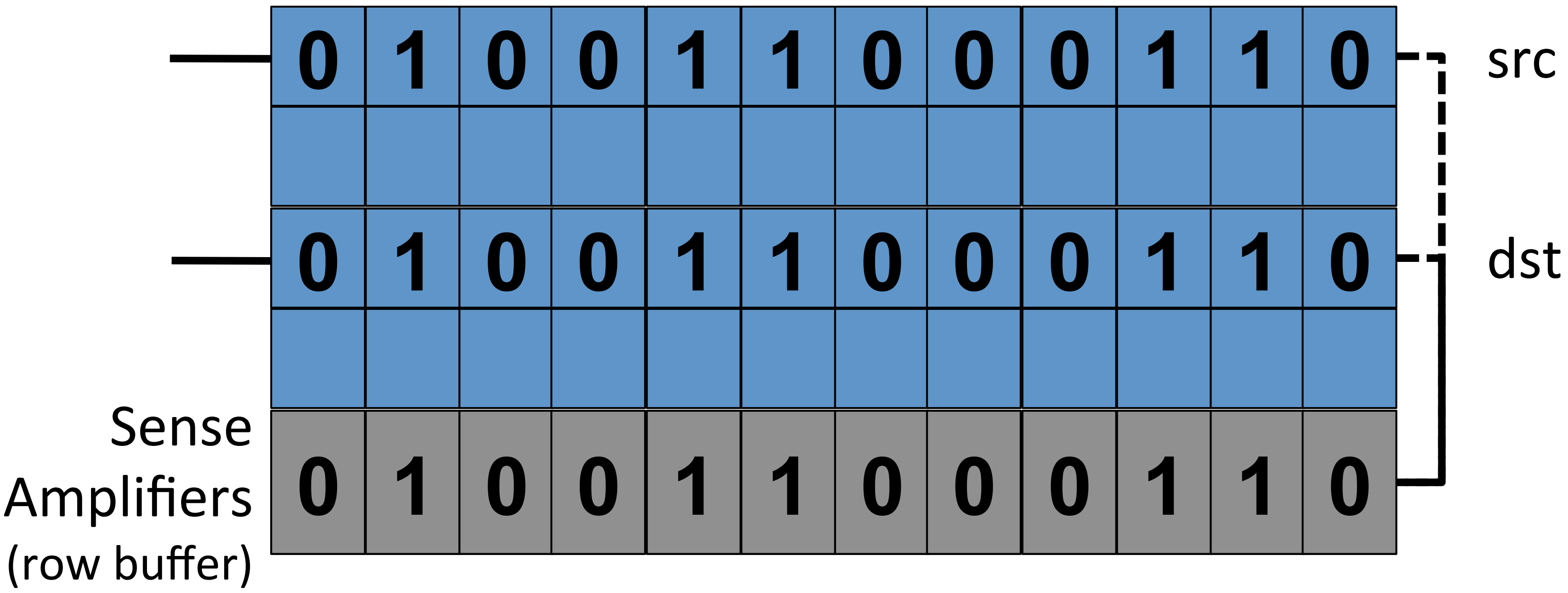
? ? ? ? ? ? ? ? ? ? ? ?

1 1 0 1 0 1 1 1 0 0 1 1

- + - - + + - - - + + -

Activate (src) → Precharge

RowClone: Intra-subarray Copy



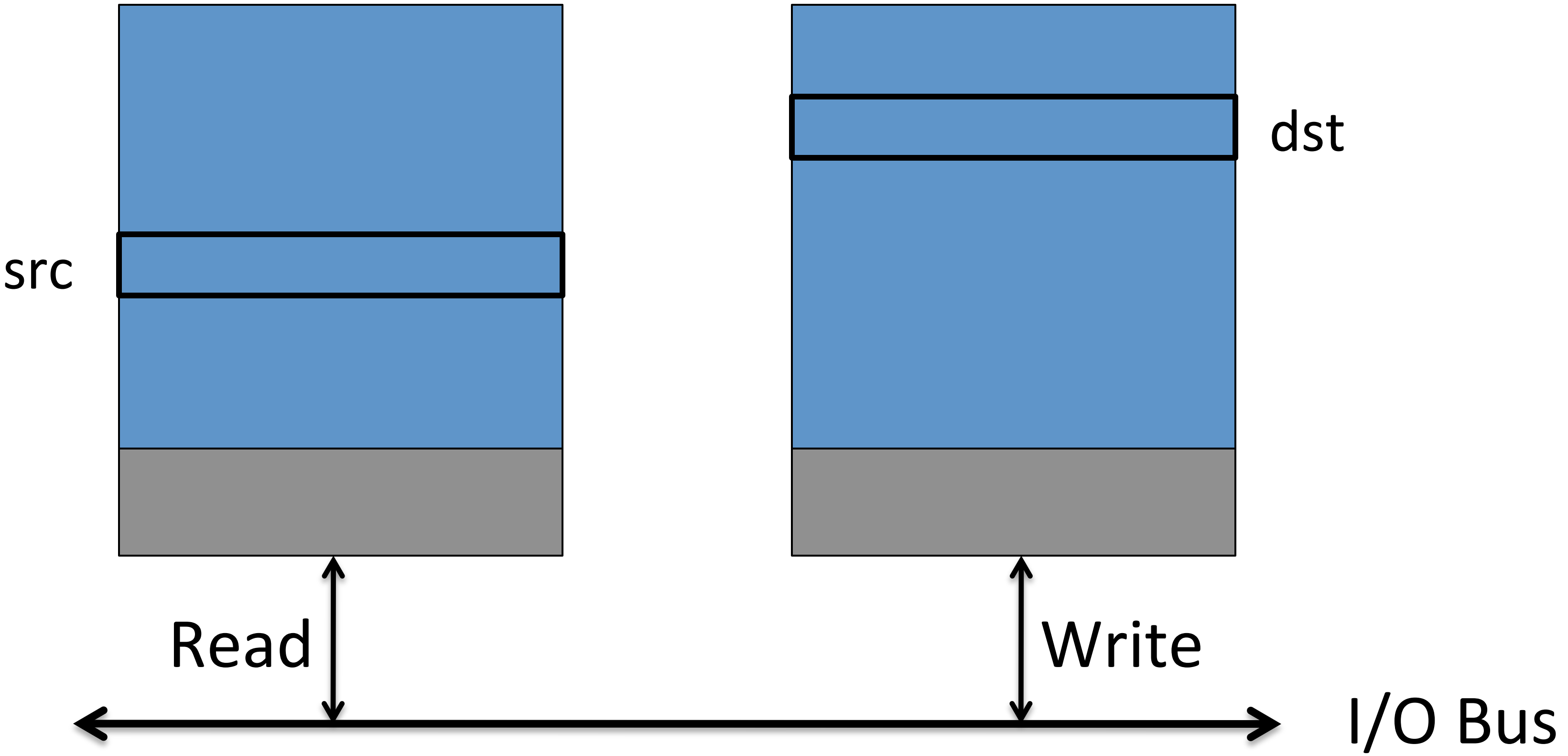
0 1 0 0 1 1 0 0 0 1 1 0

0 1 0 0 1 1 0 0 0 1 1 0

0 1 0 0 1 1 0 0 0 1 1 0

Activate (src) → Deactivate (our proposal) → Activate (dst)

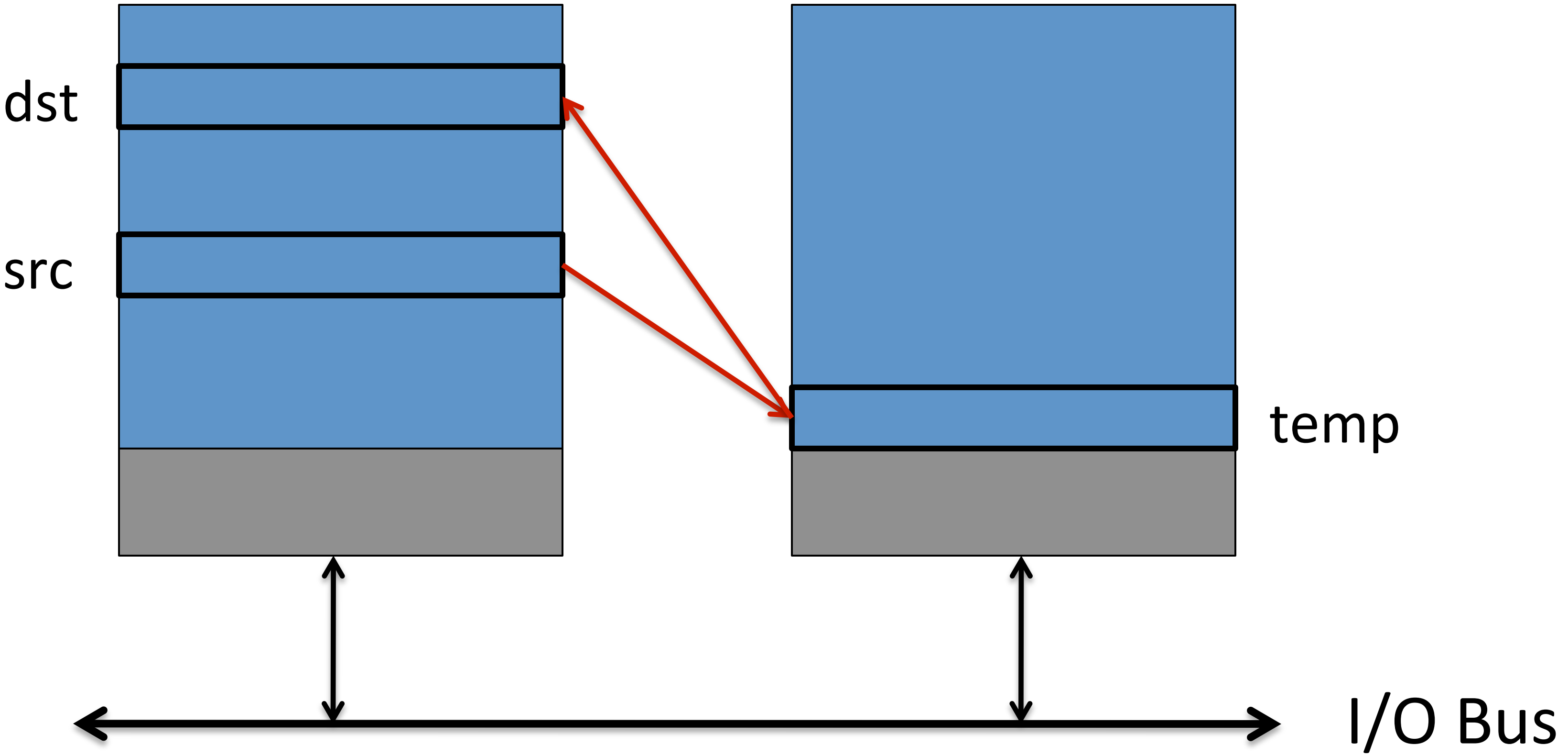
RowClone: Inter-bank Copy



Transfer

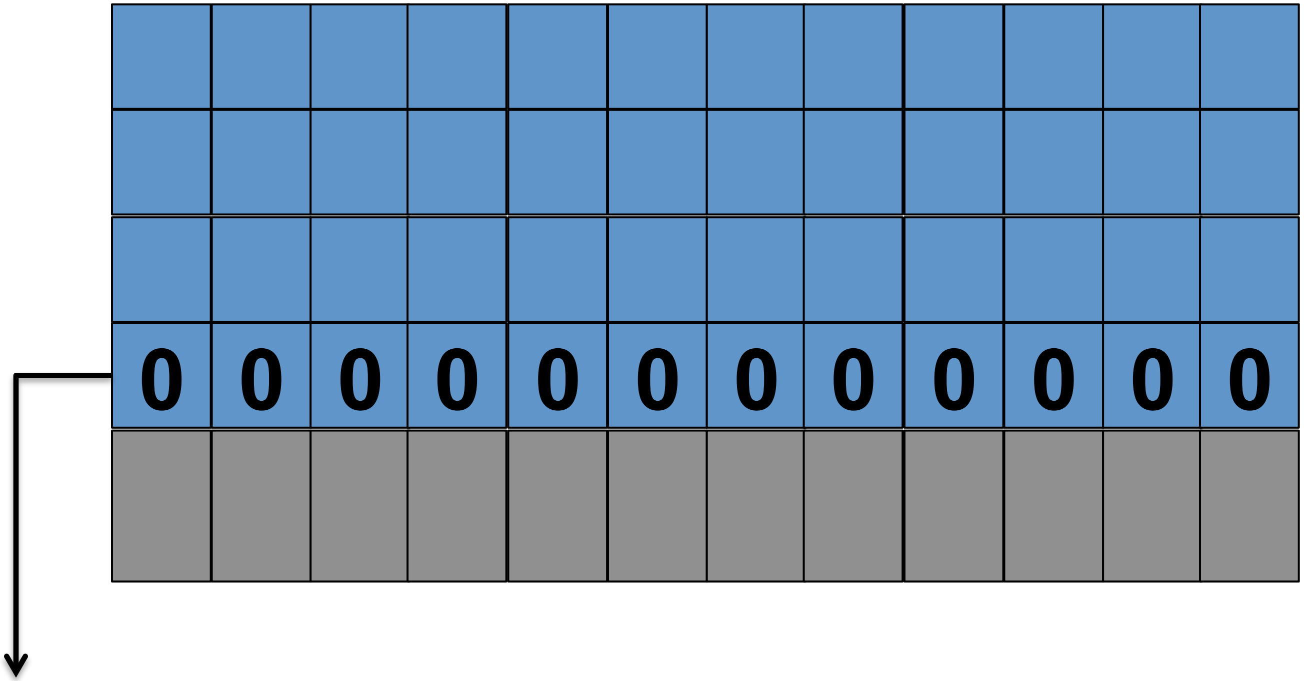
(our proposal)

RowClone: Inter-subarray Copy



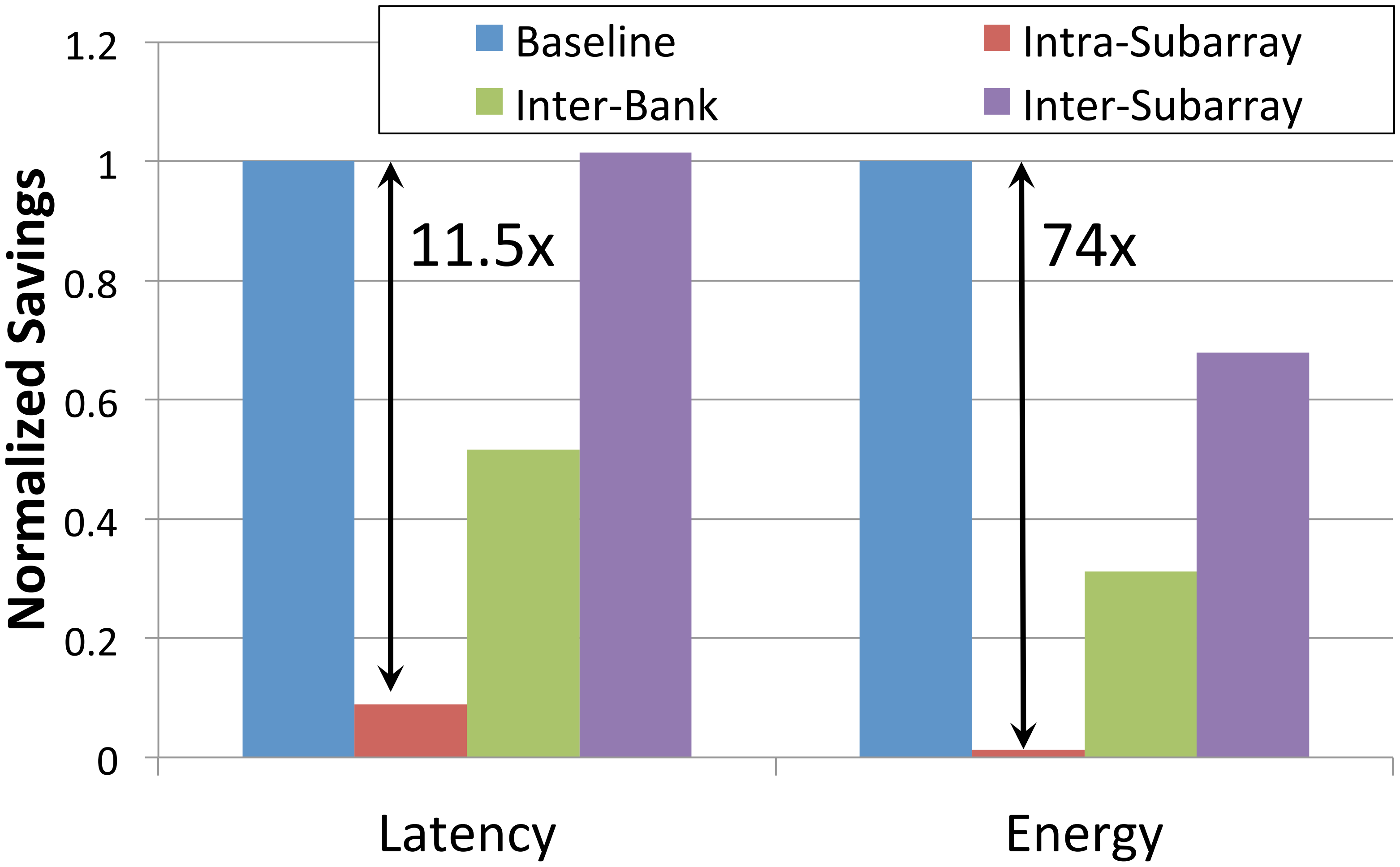
1. Transfer (src to temp)
2. Transfer (temp to dst)

Fast Row Initialization



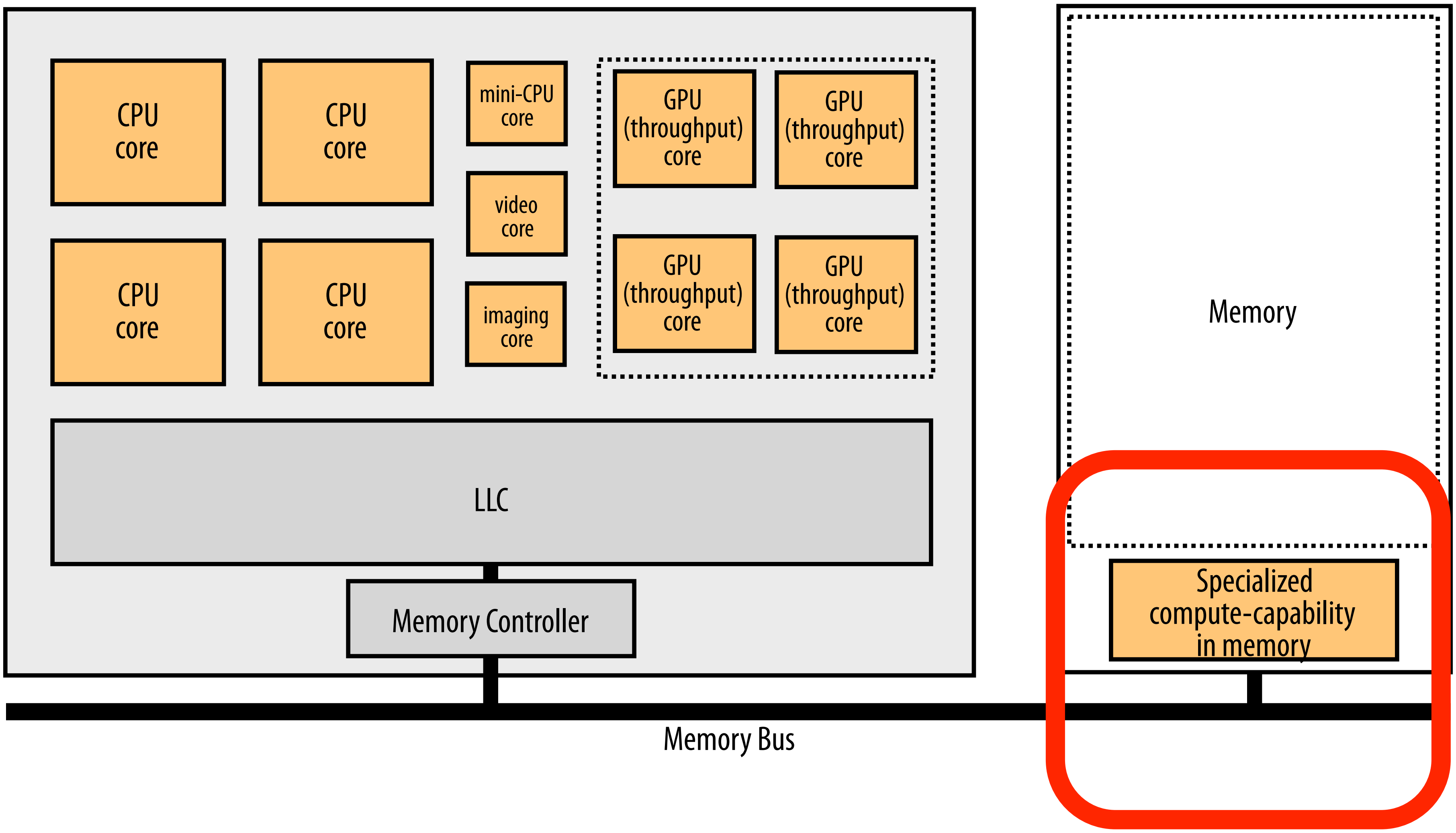
Fix a row at Zero
(0.5% loss in capacity)

RowClone: Latency and Energy Savings



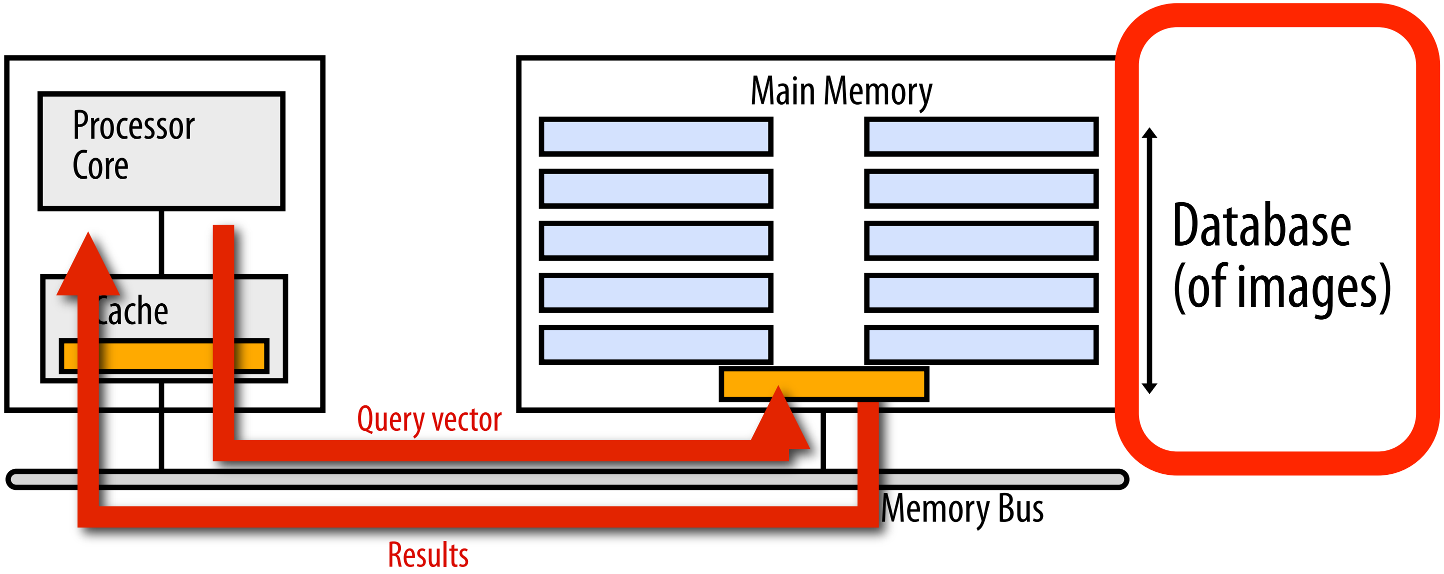
Seshadri et al., "RowClone: Fast and Efficient In-DRAM Copy and Initialization of Bulk Data," CMU Tech Report 2013.

Goal: Ultra-efficient heterogeneous architectures



Slide credit: Prof. Kayvon Fatahalian, CMU

Enabling Ultra-efficient (Visual) Search



- What is the right partitioning of computation capability?
- What is the right low-cost memory substrate?
- What memory technologies are the best enablers?
- How do we rethink/ease (visual) search algorithms/applications?

Picture credit: Prof. Kayvon Fatahalian, CMU

Agenda for Today

- What Will You Learn in This Mini-Lecture Series
- Main Memory Basics (with a Focus on DRAM)
- Major Trends Affecting Main Memory
- DRAM Scaling Problem and Solution Directions
- Solution Direction 1: System-DRAM Co-Design
- Ongoing Research
- Summary

Sampling of Ongoing Research

- Online retention time profiling
- Refresh/demand parallelization
- More computation in memory and controllers
- Efficient use of 3D stacked memory

Summary

- Major problems with DRAM scaling and design: high refresh rate, high latency, low parallelism, bulk data movement
- Four new DRAM designs
  - RAIDR: Reduces refresh impact
  - TL-DRAM: Reduces DRAM latency at low cost
  - SALP: Improves DRAM parallelism
  - RowClone: Reduces energy and performance impact of bulk data copy
- All four designs
  - Improve both performance and energy consumption
  - Are low cost (low DRAM area overhead)
  - Enable new degrees of freedom to software & controllers
- Rethinking DRAM interface and design essential for scaling
  - Co-design DRAM with the rest of the system

Thank you.

Memory Systems in the Multi-Core Era
Lecture 1: DRAM Basics and DRAM Scaling

Prof. Onur Mutlu
http://www.ece.cmu.edu/~omutlu
onur@cmu.edu
Bogazici University

June 13, 2013

Carnegie Mellon

SAFARI

Aside: Scaling Flash Memory [Cai+, ICCD'12]

- NAND flash memory has low endurance: a flash cell dies after 3k P/E cycles vs. 50k desired → Major scaling challenge for flash memory
- Flash error rate increases exponentially over flash lifetime
- Problem: Stronger error correction codes (ECC) are ineffective and undesirable for improving flash lifetime due to
  - diminishing returns on lifetime with increased correction strength
  - prohibitively high power, area, latency overheads
- Our Goal: Develop techniques to tolerate high error rates w/o strong ECC
- Observation: Retention errors are the dominant errors in MLC NAND flash
  - flash cell loses charge over time; retention errors increase as cell gets worn out
- Solution: Flash Correct-and-Refresh (FCR)
  - Periodically read, correct, and reprogram (in place) or remap each flash page before it accumulates more errors than can be corrected by simple ECC
  - Adapt "refresh" rate to the severity of retention errors (i.e., # of P/E cycles)
- Results: FCR improves flash memory lifetime by 46X with no hardware changes and low energy overhead; outperforms strong ECCs

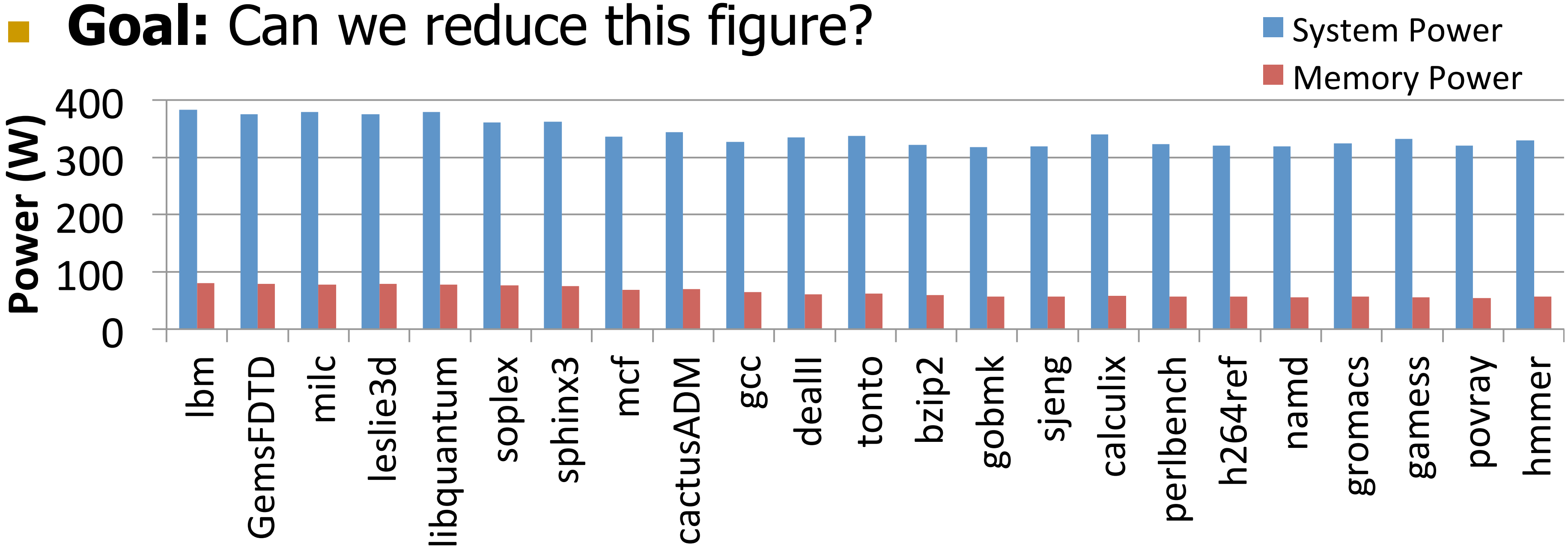
Memory Power Management via Dynamic Voltage/Frequency Scaling

Howard David (Intel)
Eugene Gorbatov (Intel)
Ulf R. Hanebutte (Intel)

Chris Fallin (CMU)
Onur Mutlu (CMU)

Memory Power is Significant

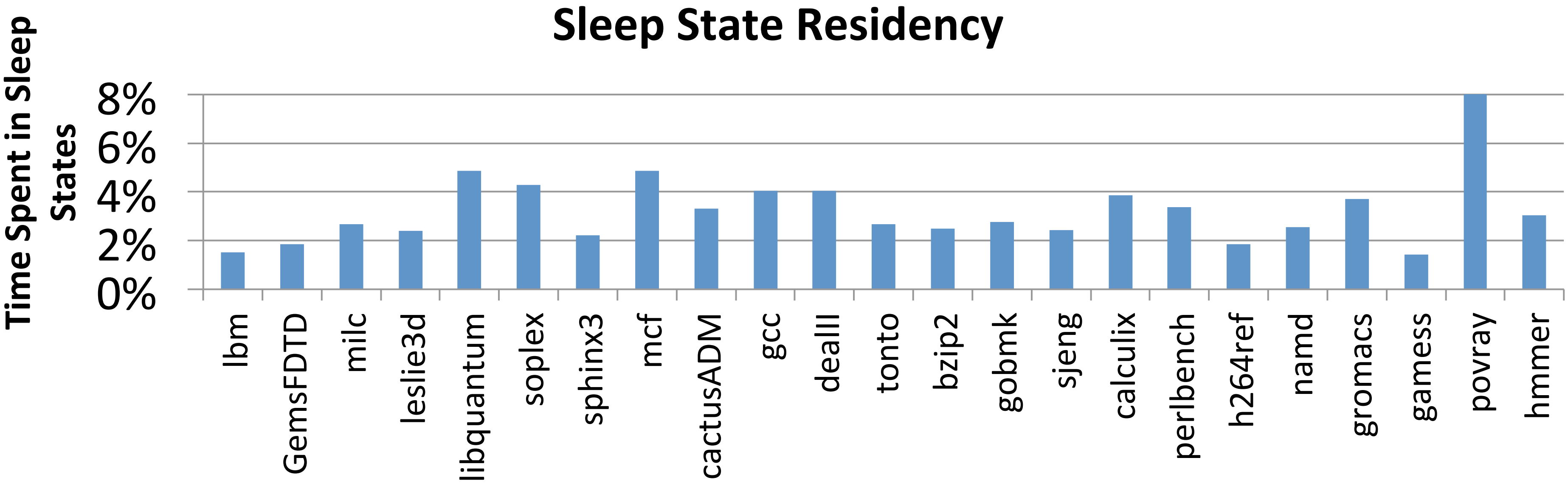
- Power consumption is a primary concern in modern servers
- Many works: CPU, whole-system or cluster-level approach
- But memory power is largely unaddressed
- Our server system*: memory is 19% of system power (avg)
  - Some work notes up to 40% of total system power
- **Goal:** Can we reduce this figure?



*Dual 4-core Intel Xeon®, 48GB DDR3 (12 DIMMs), SPEC CPU2006, all cores active.
Measured AC power, analytically modeled memory power.

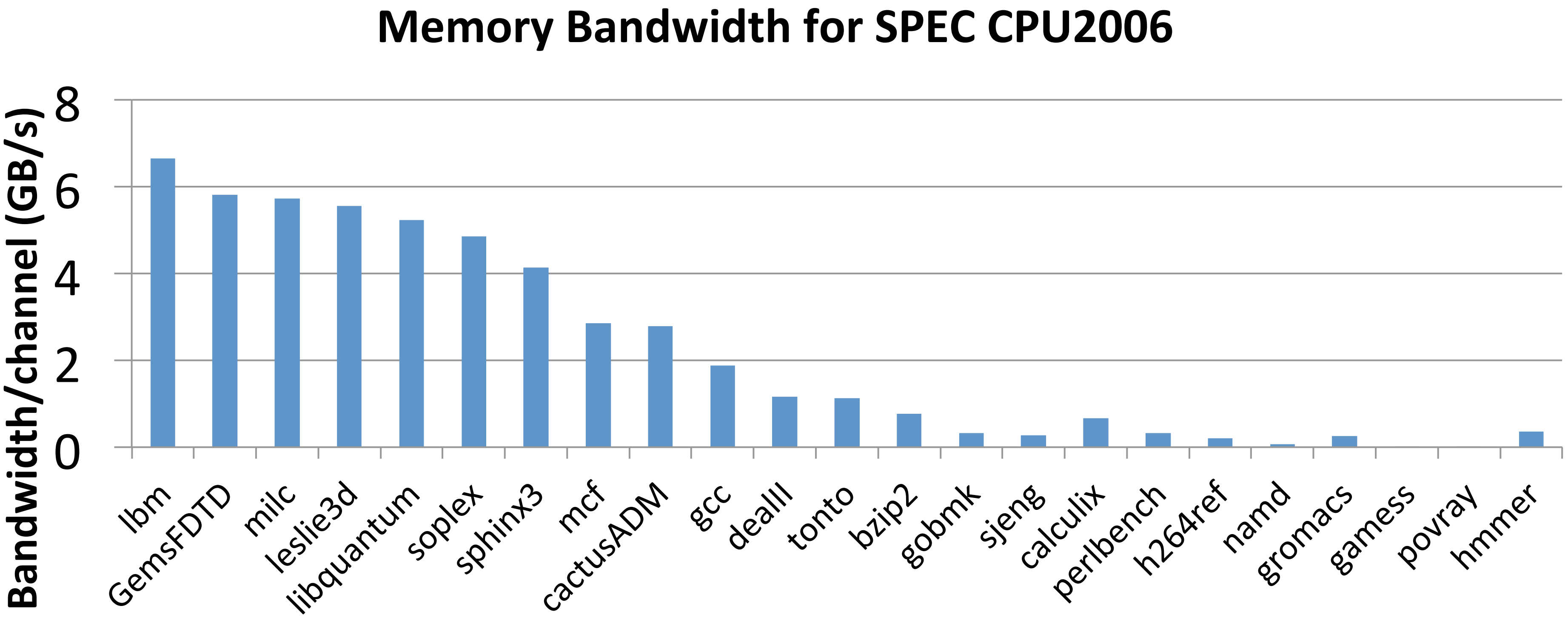
Existing Solution: Memory Sleep States?

- Most memory energy-efficiency work uses sleep states
  - Shut down DRAM devices when no memory requests active
- But, even low-memory-bandwidth workloads keep memory awake
  - Idle periods between requests diminish in multicore workloads
  - CPU-bound workloads/phases rarely completely cache-resident



Memory Bandwidth Varies Widely

- Workload memory bandwidth requirements vary widely



- Memory system is provisioned for peak capacity

  → often underutilized

Memory Power can be Scaled Down

- DDR can operate at multiple frequencies → reduce power
  - Lower frequency directly reduces switching power
  - Lower frequency allows for lower voltage
  - Comparable to CPU DVFS

CPU Voltage/ Freq.	System Power
↓ 15%	↓ 9.9%

Memory Freq.	System Power
↓ 40%	↓ 7.6%

- Frequency scaling increases latency → reduce performance
  - Memory storage array is asynchronous
  - But, bus transfer depends on frequency
  - When bus bandwidth is bottleneck, performance suffers

Observations So Far

- Memory power is a significant portion of total power
  - 19% (avg) in our system, up to 40% noted in other works

- Sleep state residency is low in many workloads
  - Multicore workloads reduce idle periods
  - CPU-bound applications send requests frequently enough to keep memory devices awake

- Memory bandwidth demand is very low in some workloads

- Memory power is reduced by frequency scaling
  - And voltage scaling can give further reductions

DVFS for Memory

- **Key Idea:** observe memory bandwidth utilization, then adjust memory frequency/voltage, to reduce power with minimal performance loss

  → **Dynamic Voltage/Frequency Scaling (DVFS) for memory**

- **Goal in this work**:
  - Implement DVFS in the memory system, by:
  - Developing a simple control algorithm to exploit opportunity for reduced memory frequency/voltage by observing behavior
  - Evaluating the proposed algorithm on a real system

Outline

- Motivation

- Background and Characterization
  - DRAM Operation
  - DRAM Power
  - Frequency and Voltage Scaling

- Performance Effects of Frequency Scaling

- Frequency Control Algorithm

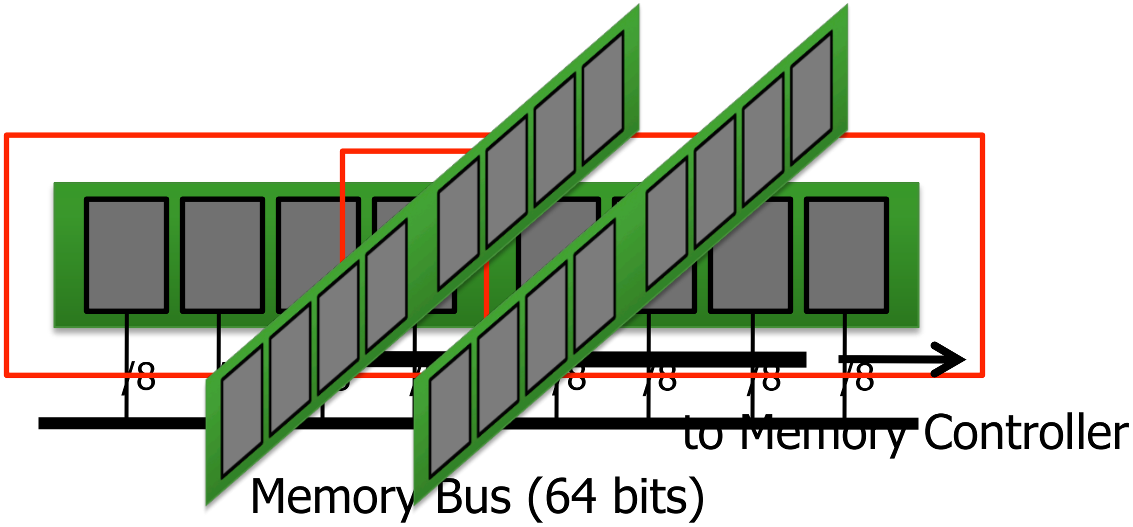
- Evaluation and Conclusions

Outline

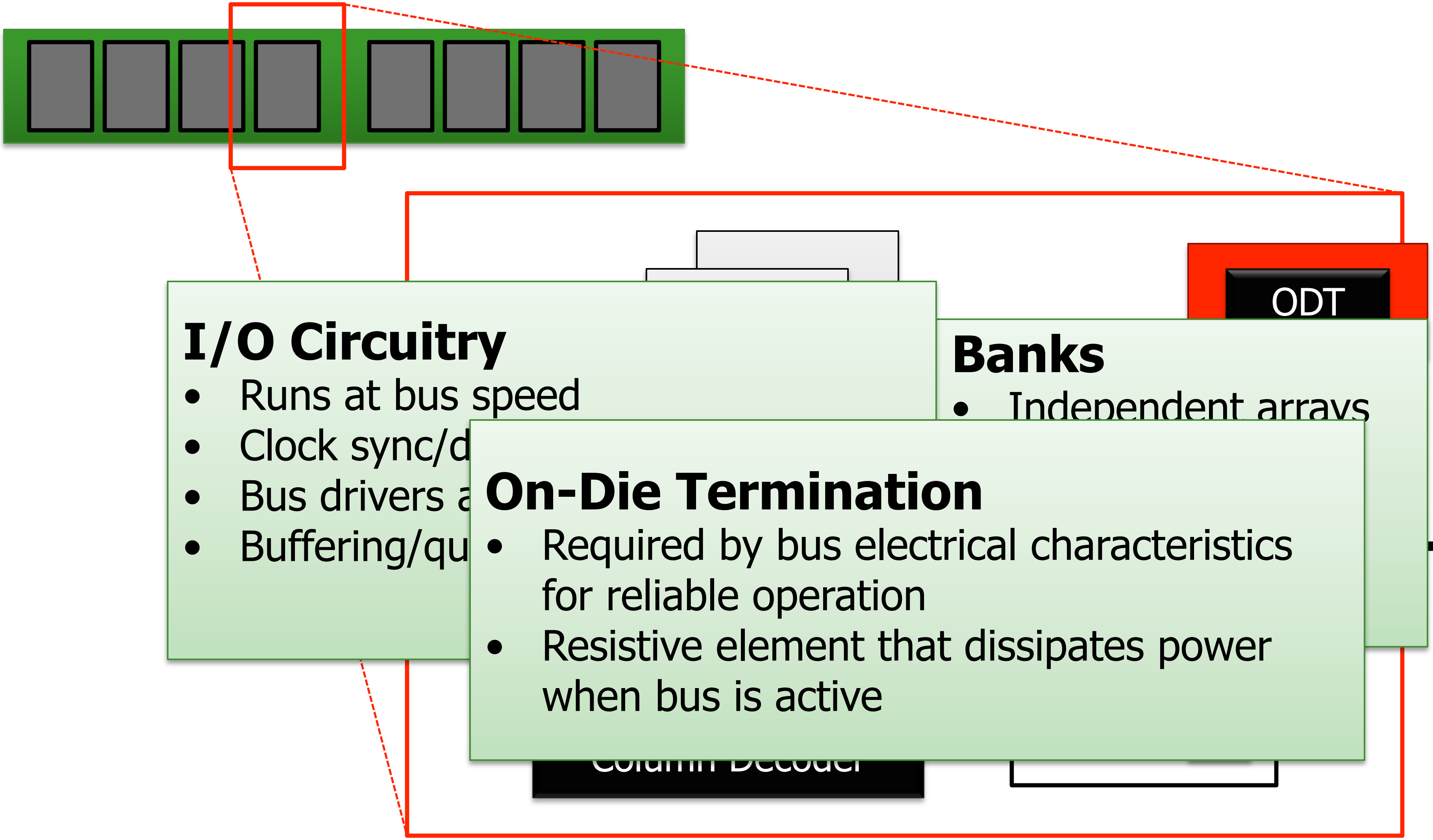
- Motivation
- Background and Characterization
  - DRAM Operation
  - DRAM Power
  - Frequency and Voltage Scaling
- Performance Effects of Frequency Scaling
- Frequency Control Algorithm
- Evaluation and Conclusions

DRAM Operation

- Main memory consists of DIMMs of DRAM devices
- Each DIMM is attached to a memory bus (channel)
- Multiple DIMMs can connect to one channel



Inside a DRAM Device



I/O Circuitry

- Runs at bus speed
- Clock sync/d
- Bus drivers a
- Buffering/qu

Banks

- Independent arrays

On-Die Termination

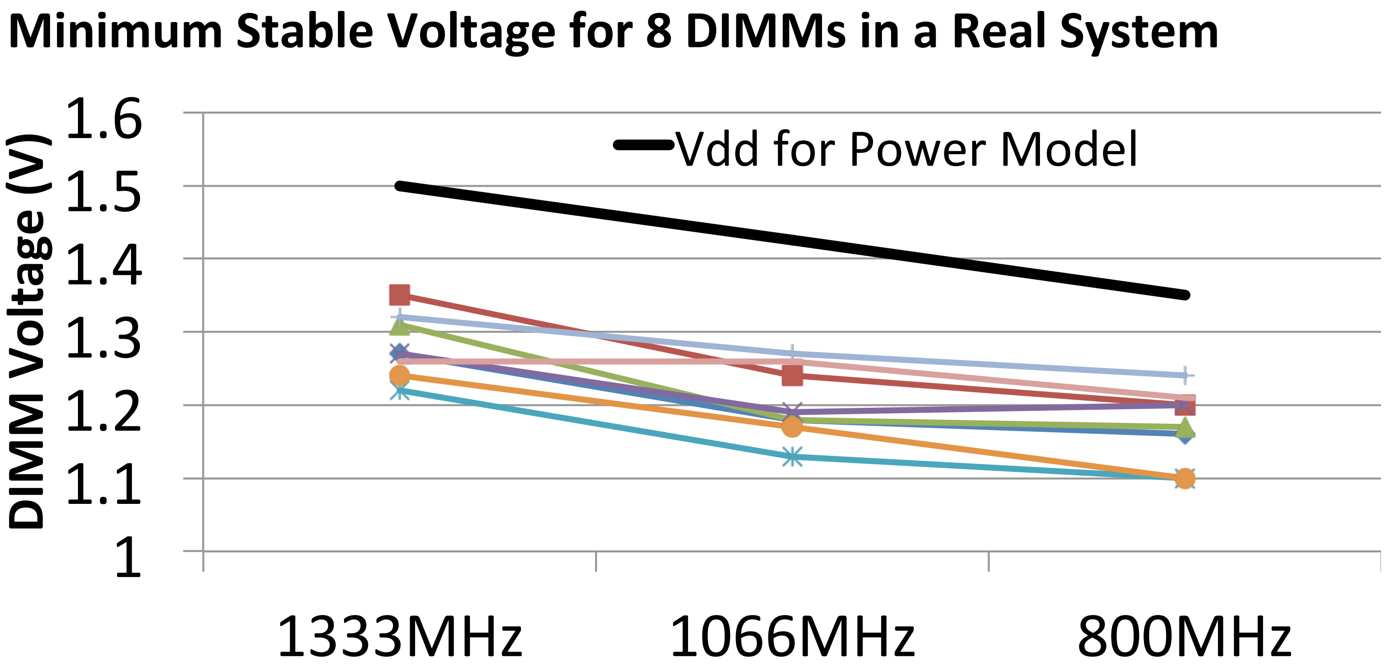
- Required by bus electrical characteristics for reliable operation
- Resistive element that dissipates power when bus is active

Effect of Frequency Scaling on Power

- **Reduced memory bus frequency:**
- Does not affect bank power:
  - Constant energy per operation
  - Depends only on utilized memory bandwidth
- Decreases I/O power:
  - Dynamic power in bus interface and clock circuitry reduces due to less frequent switching
- Increases termination power:
  - Same data takes longer to transfer
  - Hence, bus utilization increases
- Tradeoff between I/O and termination results in a net power reduction at lower frequencies

Effects of Voltage Scaling on Power

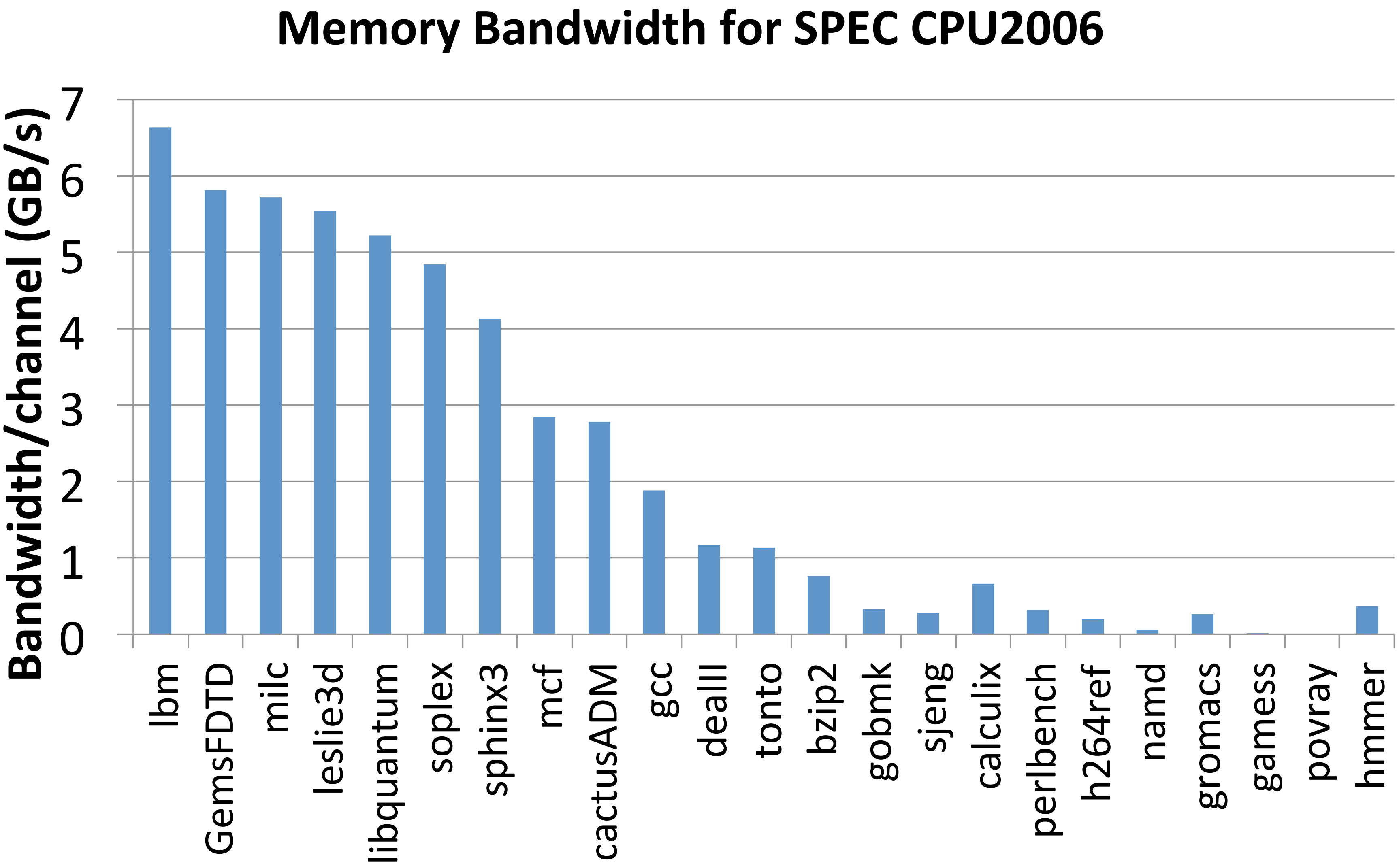
- Voltage scaling further reduces power because all parts of memory devices will draw less current (at less voltage)
- Voltage reduction is possible because stable operation requires lower voltage at lower frequency:



Outline

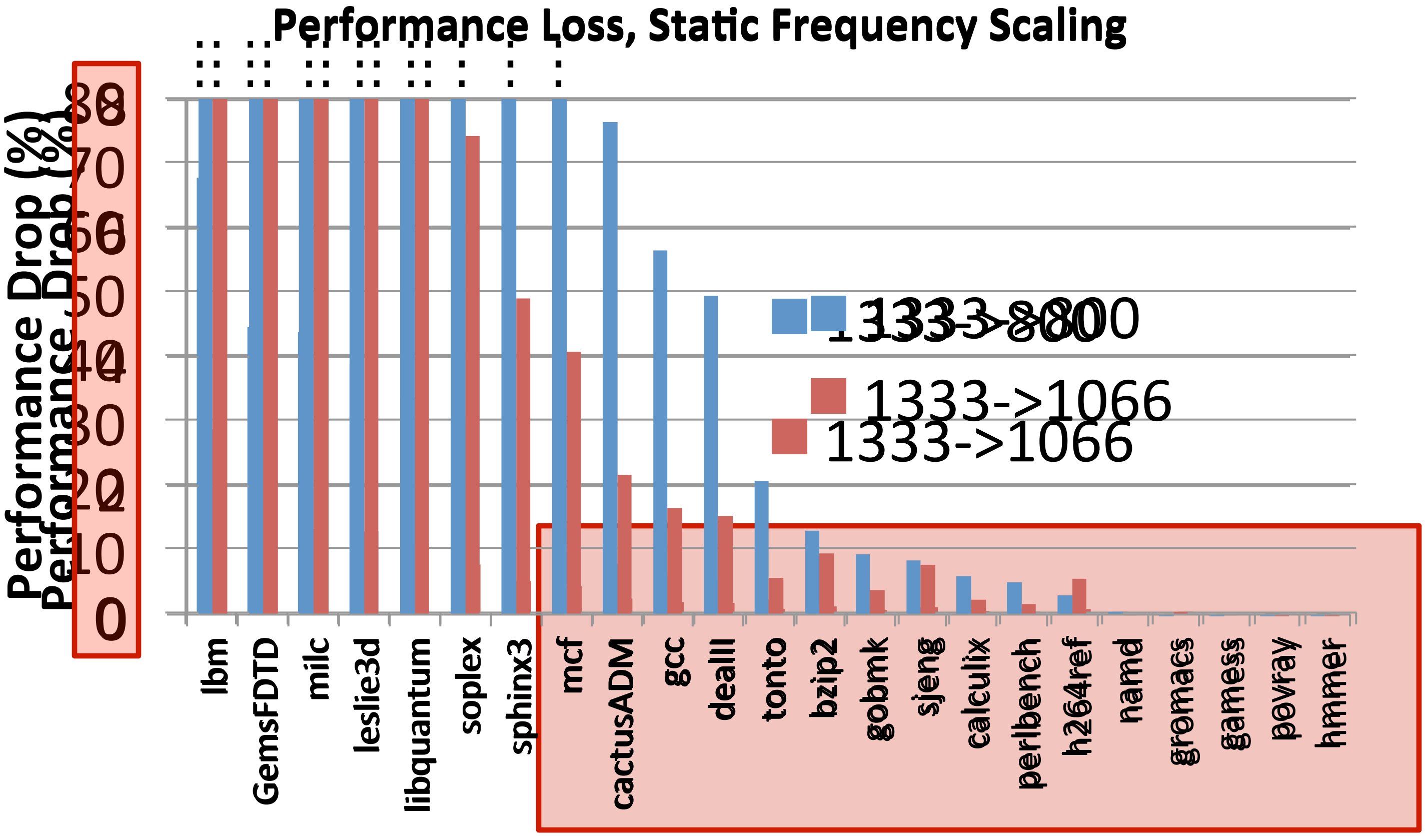
- Motivation
- Background and Characterization
  - DRAM Operation
  - DRAM Power
  - Frequency and Voltage Scaling
- Performance Effects of Frequency Scaling
- Frequency Control Algorithm
- Evaluation and Conclusions

How Much Memory Bandwidth is Needed?



Performance Impact of Static Frequency Scaling

- Performance impact is proportional to bandwidth demand
- Many workloads tolerate lower frequency with minimal performance drop



Outline

- Motivation

- Background and Characterization
  - DRAM Operation
  - DRAM Power
  - Frequency and Voltage Scaling

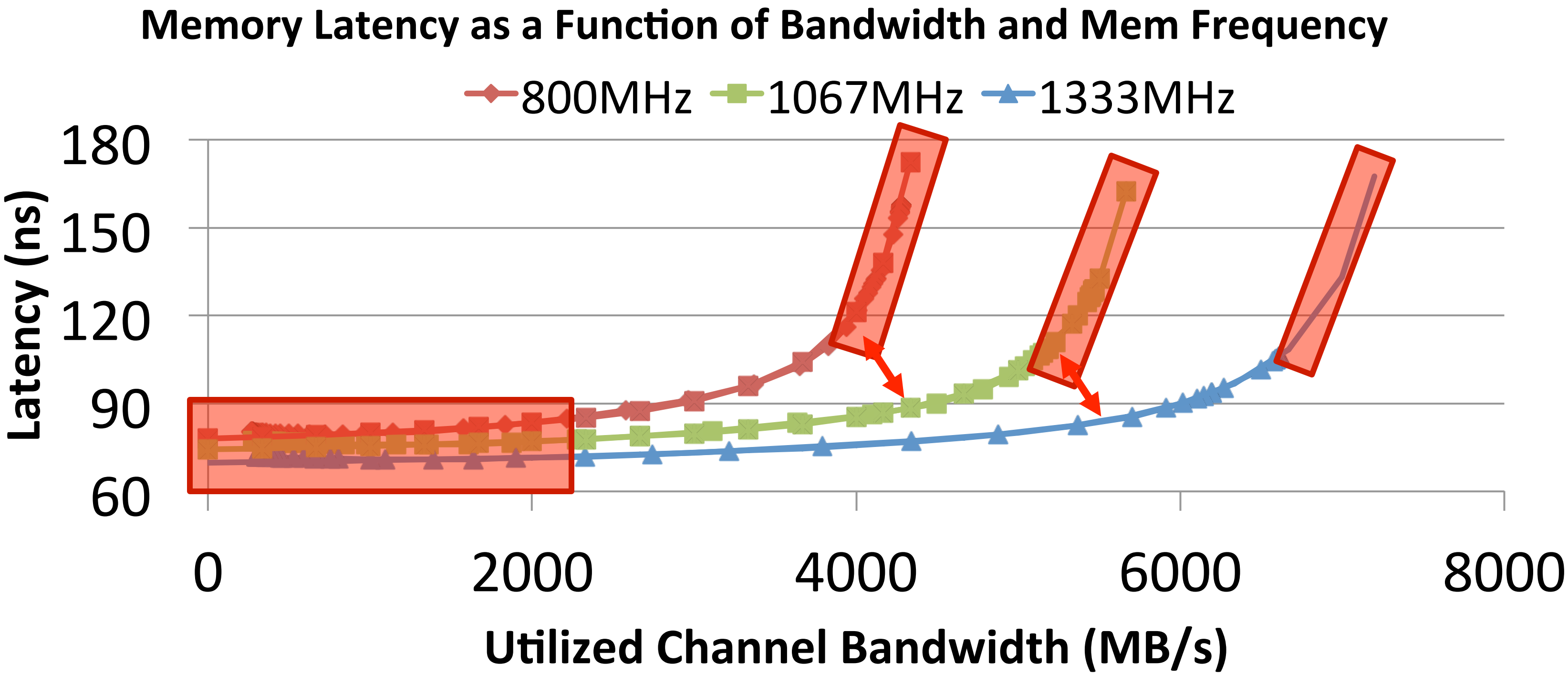
- Performance Effects of Frequency Scaling

- Frequency Control Algorithm

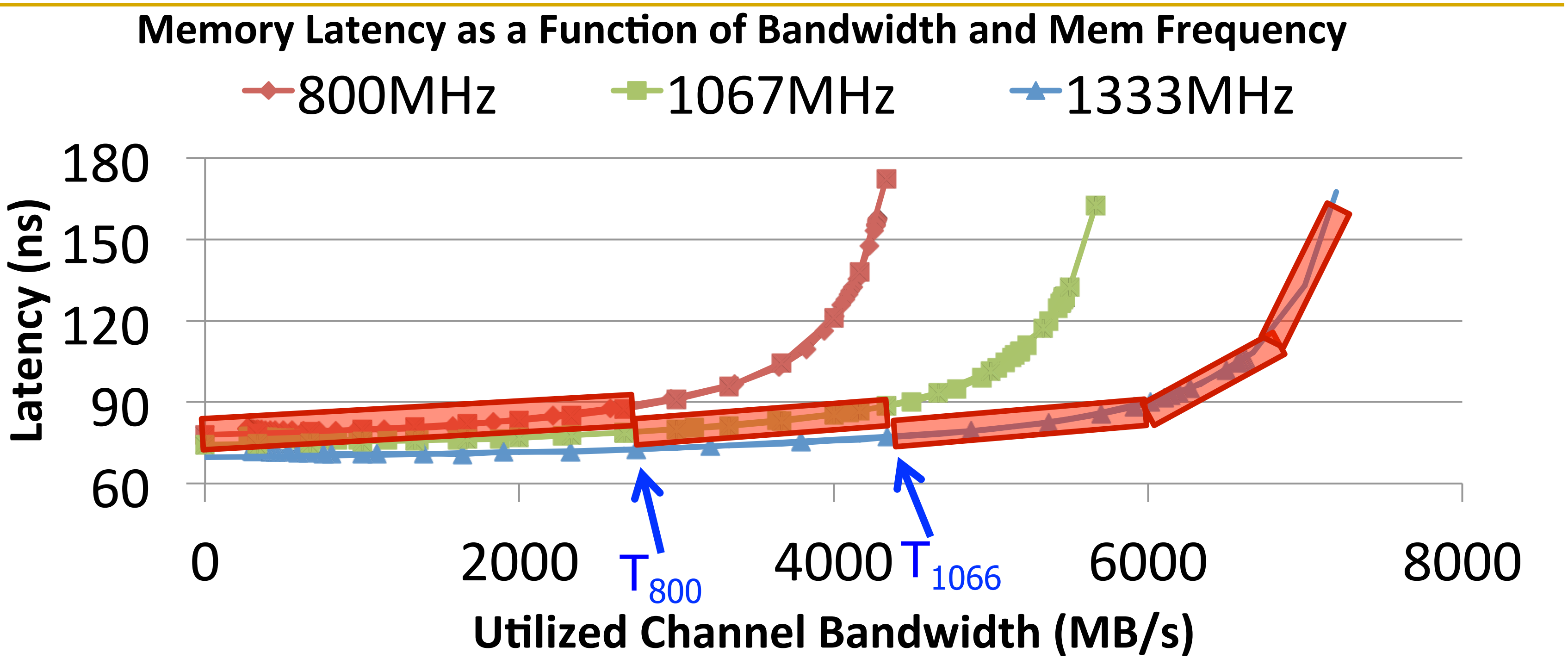
- Evaluation and Conclusions

Memory Latency Under Load

- At low load, most time is in array access and bus transfer
  → small constant offset between bus-frequency latency curves
- As load increases, queueing delay begins to dominate
  → bus frequency significantly affects latency



Control Algorithm: Demand-Based Switching



```
After each epoch of length $T_{epoch}$:
    Measure per-channel bandwidth BW
    if          BW < $T_{800}$     : switch to  800MHz
    else if BW < $T_{1066}$    : switch to 1066MHz
    else                              : switch to 1333MHz
```

Implementing V/F Switching

- **Halt Memory Operations**
  - Pause requests
  - Put DRAM in Self-Refresh
  - Stop the DIMM clock
- **Transition Voltage/Frequency**
  - Begin voltage ramp

- 👍 Memory frequency already adjustable statically
- 👍 Voltage regulators for CPU DVFS can work for memory DVFS
- 👍 Full transition takes ~20μs

Outline

- Motivation
- Background and Characterization
  - DRAM Operation
  - DRAM Power
  - Frequency and Voltage Scaling
- Performance Effects of Frequency Scaling
- Frequency Control Algorithm
- Evaluation and Conclusions

Evaluation Methodology

- **Real-system evaluation**
  - Dual 4-core Intel Xeon®, 3 memory channels/socket
  - 48 GB of DDR3 (12 DIMMs, 4GB dual-rank, 1333MHz)

- **Emulating memory frequency for performance**
  - Altered memory controller timing registers (tRC, tB2BCAS)
  - Gives performance equivalent to slower memory frequencies

- **Modeling power reduction**
  - Measure baseline system (AC power meter, 1s samples)
  - Compute reductions with an analytical model (see paper)

Evaluation Methodology

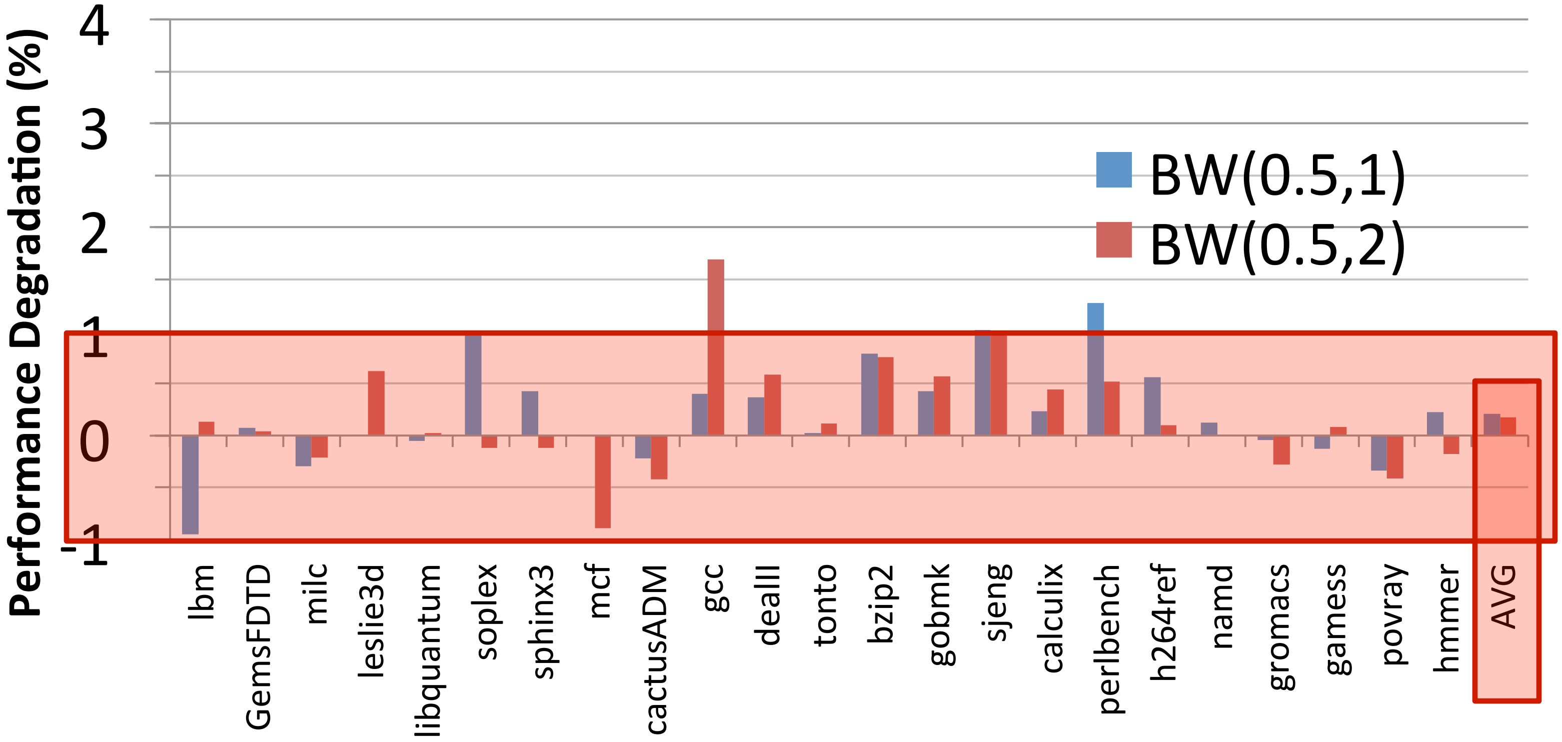
- **Workloads**
  - SPEC CPU2006: CPU-intensive workloads
  - All cores run a copy of the benchmark

- **Parameters**
  - T_{epoch} = 10ms
  - Two variants of algorithm with different switching thresholds:
  - BW(0.5, 1): T_{800} = 0.5GB/s, T_{1066} = 1GB/s
  - BW(0.5, 2): T_{800} = 0.5GB/s, T_{1066} = 2GB/s

    → More aggressive frequency/voltage scaling

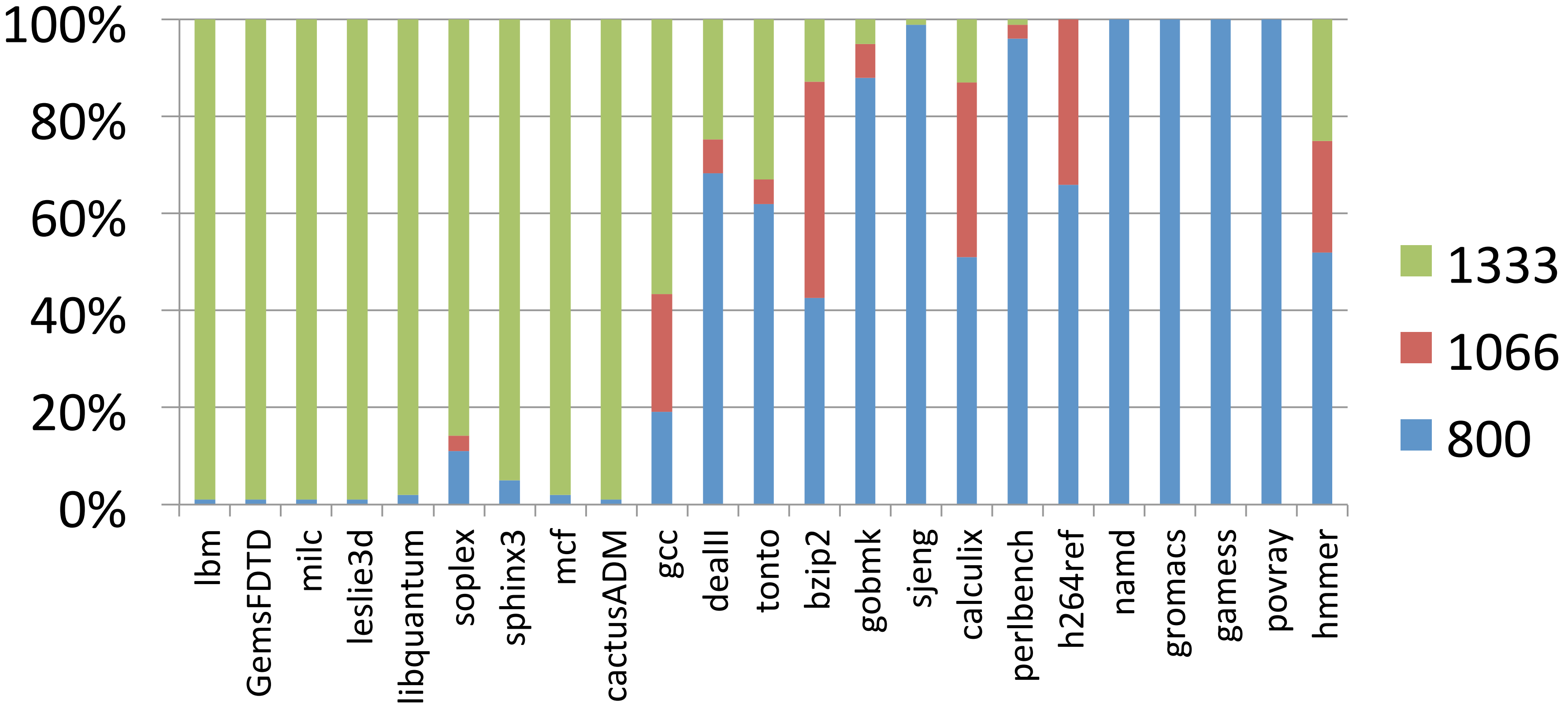
Performance Impact of Memory DVFS

- Minimal performance degradation: 0.2% (avg), 1.7% (max)
- Experimental error ~1%



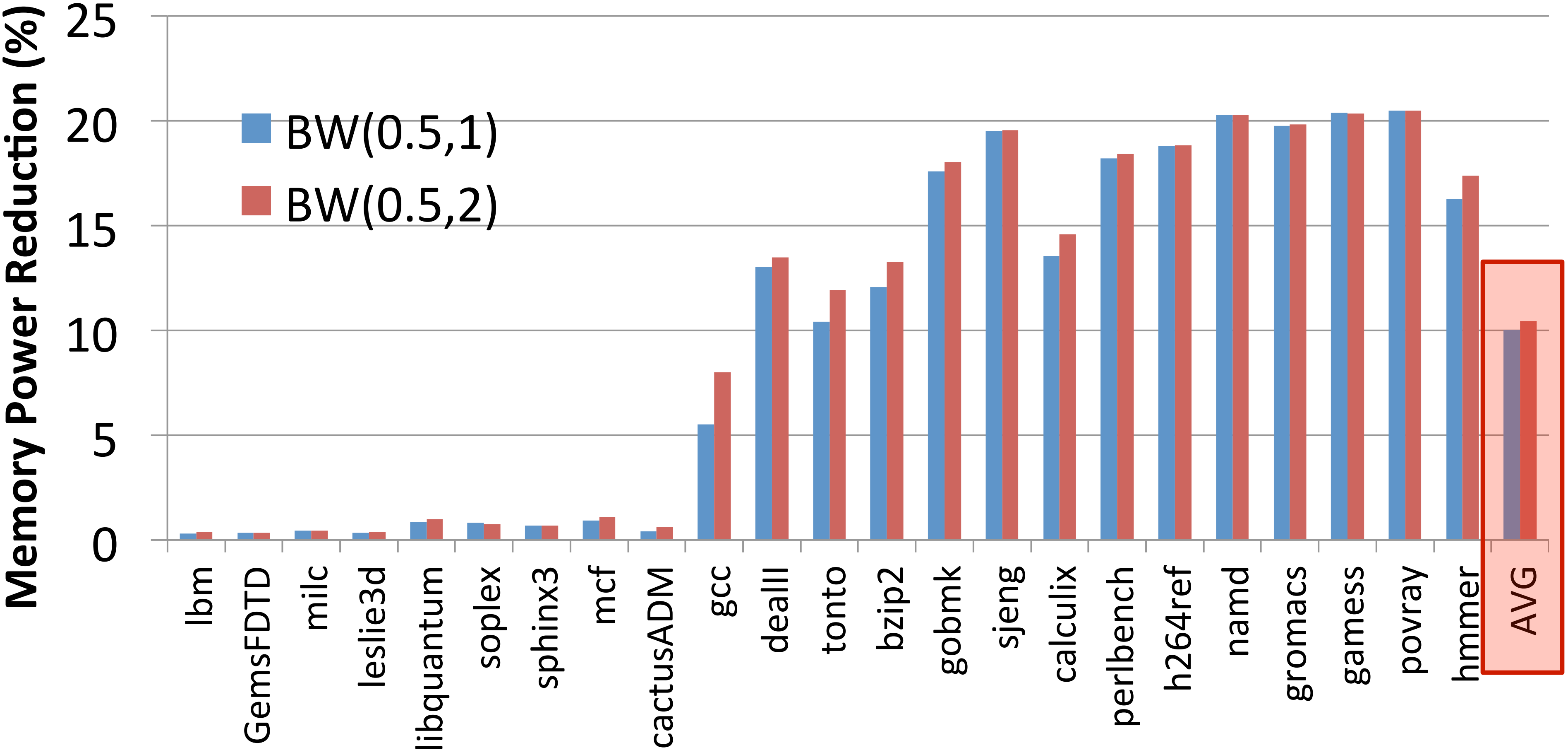
Memory Frequency Distribution

- Frequency distribution shifts toward higher memory frequencies with more memory-intensive benchmarks



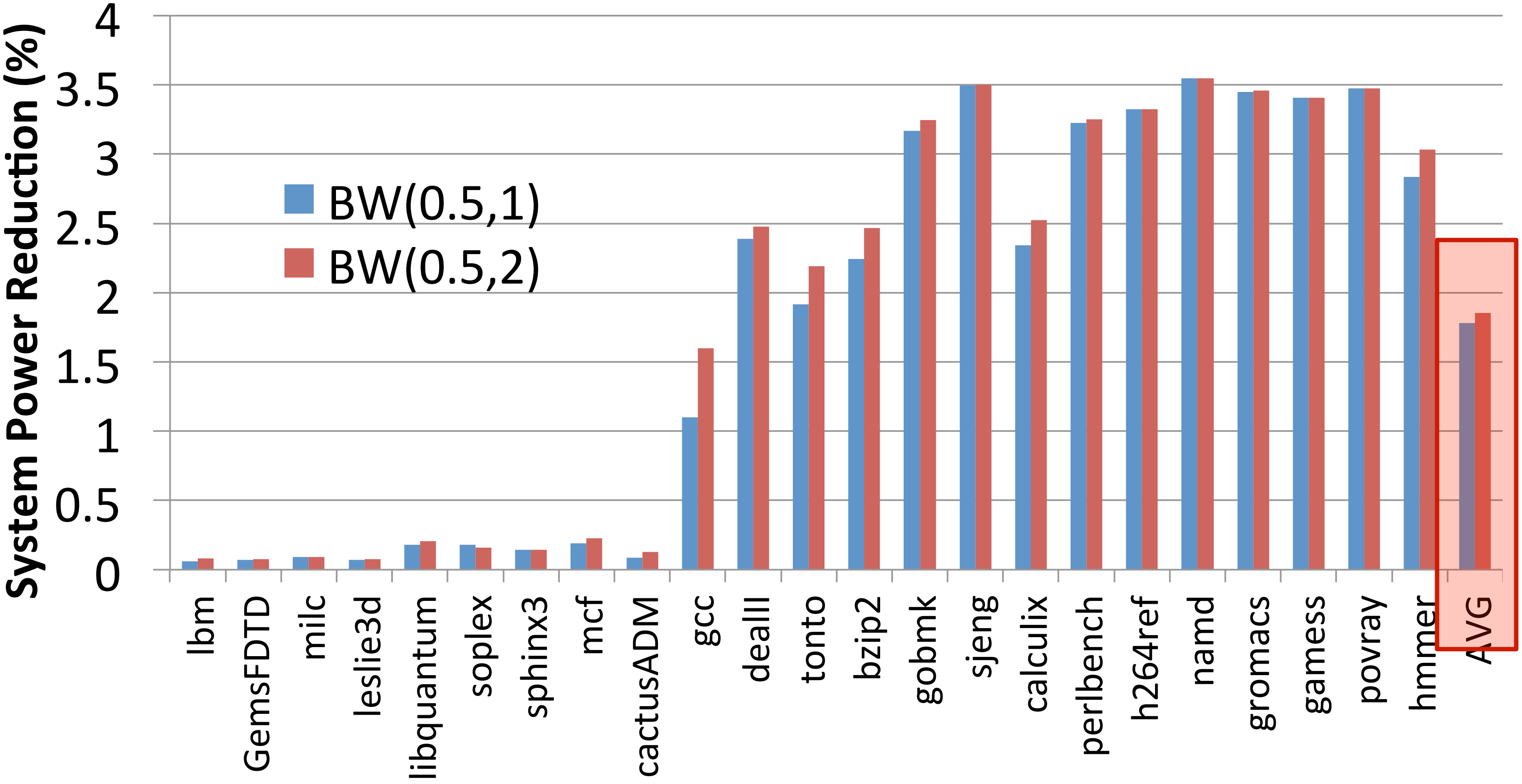
Memory Power Reduction

- Memory power reduces by 10.4% (avg), 20.5% (max)



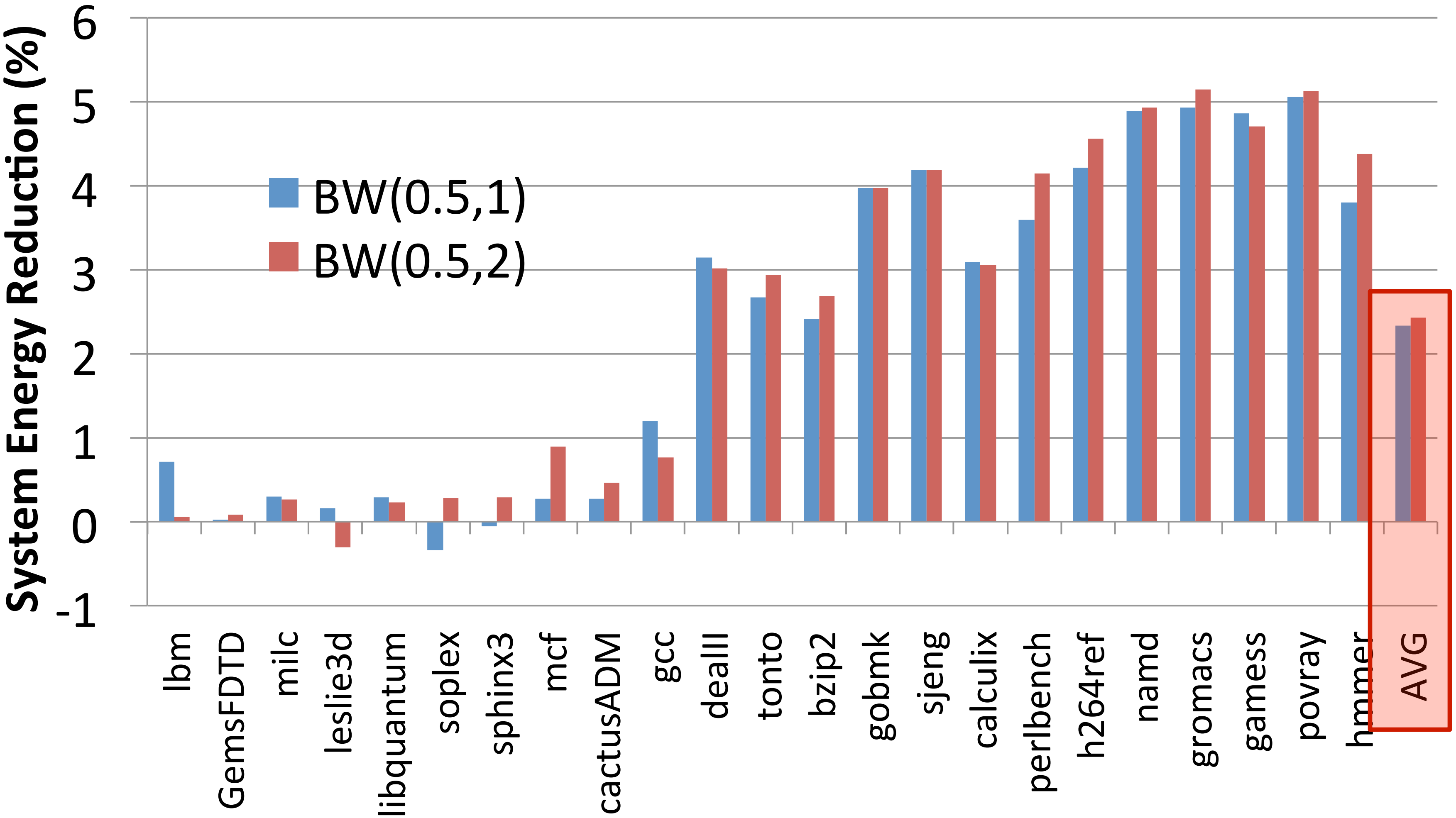
System Power Reduction

- As a result, system power reduces by 1.9% (avg), 3.5% (max)



System Energy Reduction

- System energy reduces by 2.4% (avg), 5.1% (max)



Related Work

- **MemScale** [Deng11], concurrent work (ASPLOS 2011)
  - Also proposes Memory DVFS
  - Application performance impact model to decide voltage and frequency: requires specific modeling for a given system; our bandwidth-based approach avoids this complexity
  - Simulation-based evaluation; our work is a real-system proof of concept

- **Memory Sleep States** (Creating opportunity with data placement [Lebeck00,Pandey06], OS scheduling [Delaluz02], VM subsystem [Huang05]; Making better decisions with better models [Hur08,Fan01])
- **Power Limiting/Shifting** (RAPL [David10] uses memory throttling for thermal limits; CPU throttling for memory traffic [Lin07,08]; Power shifting across system [Felter05])

Conclusions

- Memory power is a significant component of system power
  - 19% average in our evaluation system, 40% in other work

- Workloads often keep memory active but underutilized
  - Channel bandwidth demands are highly variable
  - Use of memory sleep states is often limited

- Scaling memory frequency/voltage can reduce memory power with minimal system performance impact
  - 10.4% average memory power reduction
  - Yields 2.4% average system energy reduction

- Greater reductions are possible with wider frequency/ voltage range and better control algorithms

Memory Power Management via Dynamic Voltage/Frequency Scaling

Howard David (Intel)
Eugene Gorbatov (Intel)
Ulf R. Hanebutte (Intel)

Chris Fallin (CMU)
Onur Mutlu (CMU)

SAFARI

Carnegie Mellon