Scalable Many-Core Memory Systems Topic 1: DRAM Basics and DRAM Scaling

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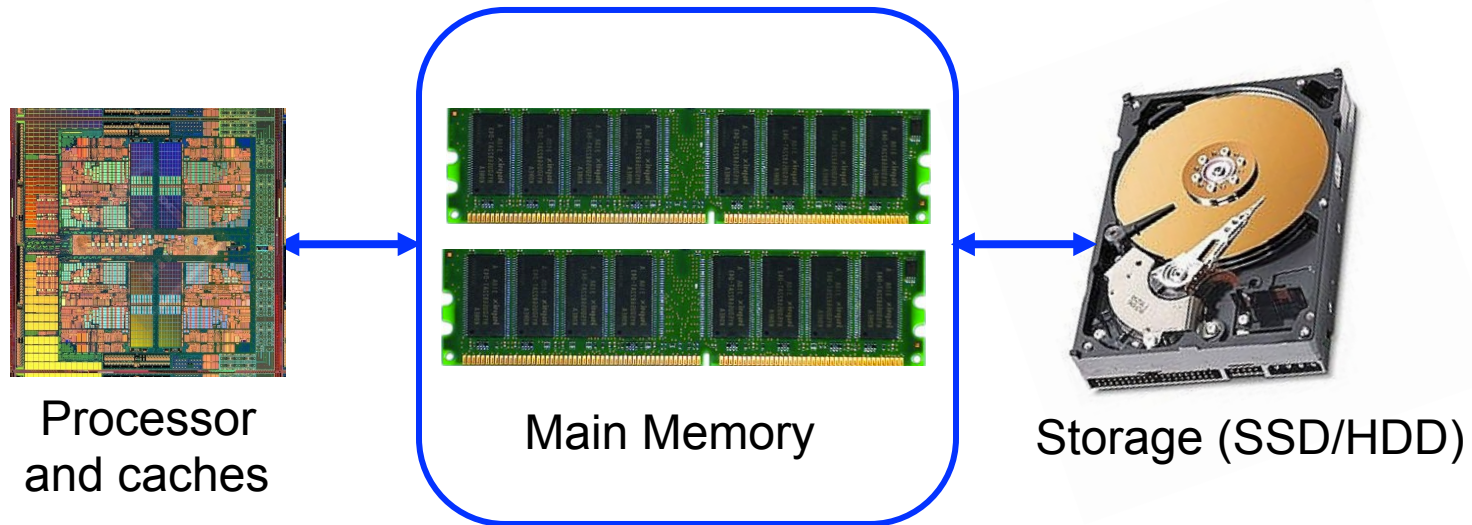
http://www.ece.cmu.edu/~omutlu

onur@cmu.edu

HiPEAC ACACES Summer School 2013

July 15-19, 2013

Carnegie Mellon

SAFARI

The Main Memory System

Processor and caches — Main Memory — Storage (SSD/HDD)

- 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
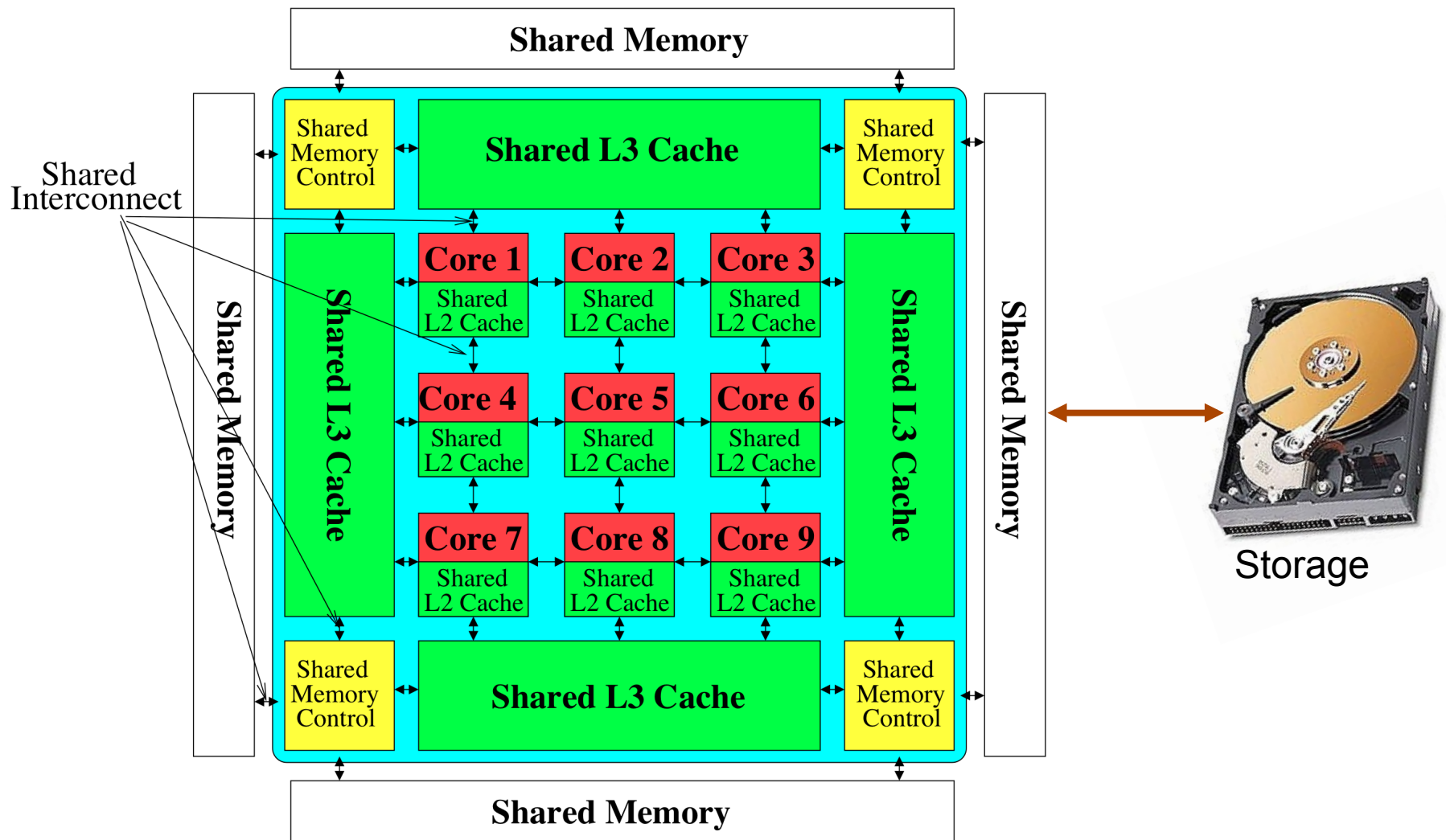
State of the Main Memory System

- 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

SAFARI

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
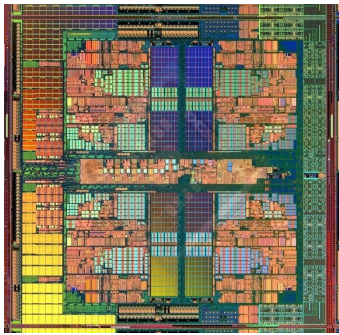
Major Trends Affecting Main Memory (II)

- **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

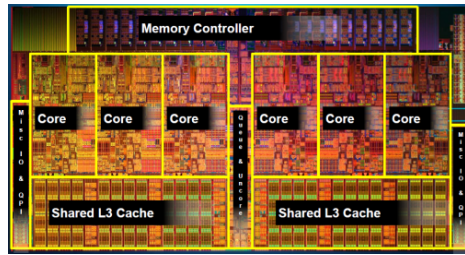- Main memory energy/power is a key system design concern

- DRAM technology scaling is ending

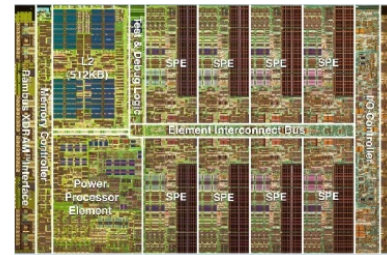**SAFARI**

Example Trend: Many Cores on Chip

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

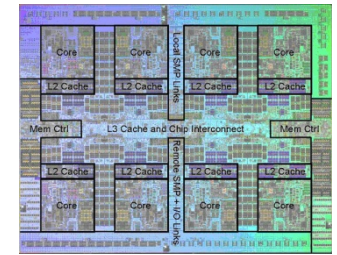AMD Barcelona
4 cores

Intel Core i7
8 cores

IBM Cell BE
8+1 cores

IBM POWER7
8 cores
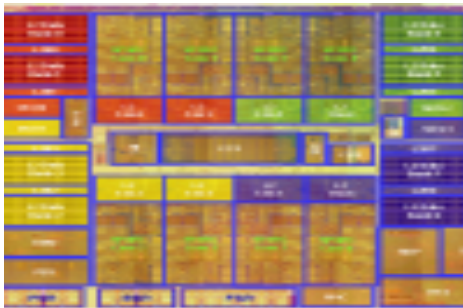
Sun Niagara II
8 cores
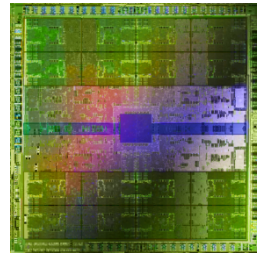
Nvidia Fermi
448 "cores"
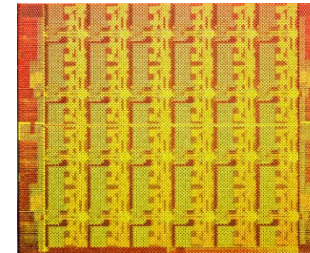
Intel SCC
48 cores, networked
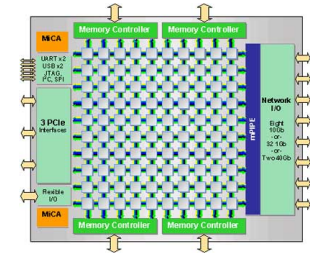
Tilera TILE Gx
100 cores, networked

SAFARI

Consequence: The Memory Capacity Gap

Core count doubling ~ every 2 years
DRAM DIMM capacity doubling ~ every 3 years
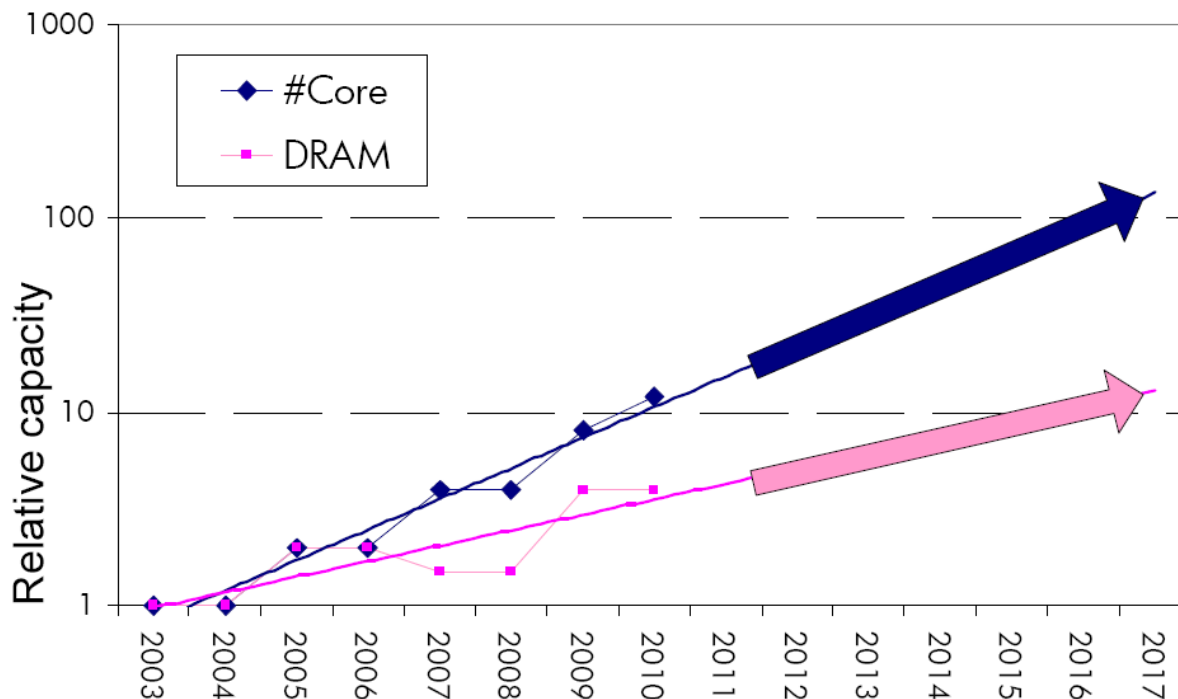
Source: Lim et al., ISCA 2009.

- *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

SAFARI

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

SAFARI

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.
- Seshadri+, "RowClone: Fast and Efficient In-DRAM Copy and Initialization of Bulk Data," 2013.

SAFARI

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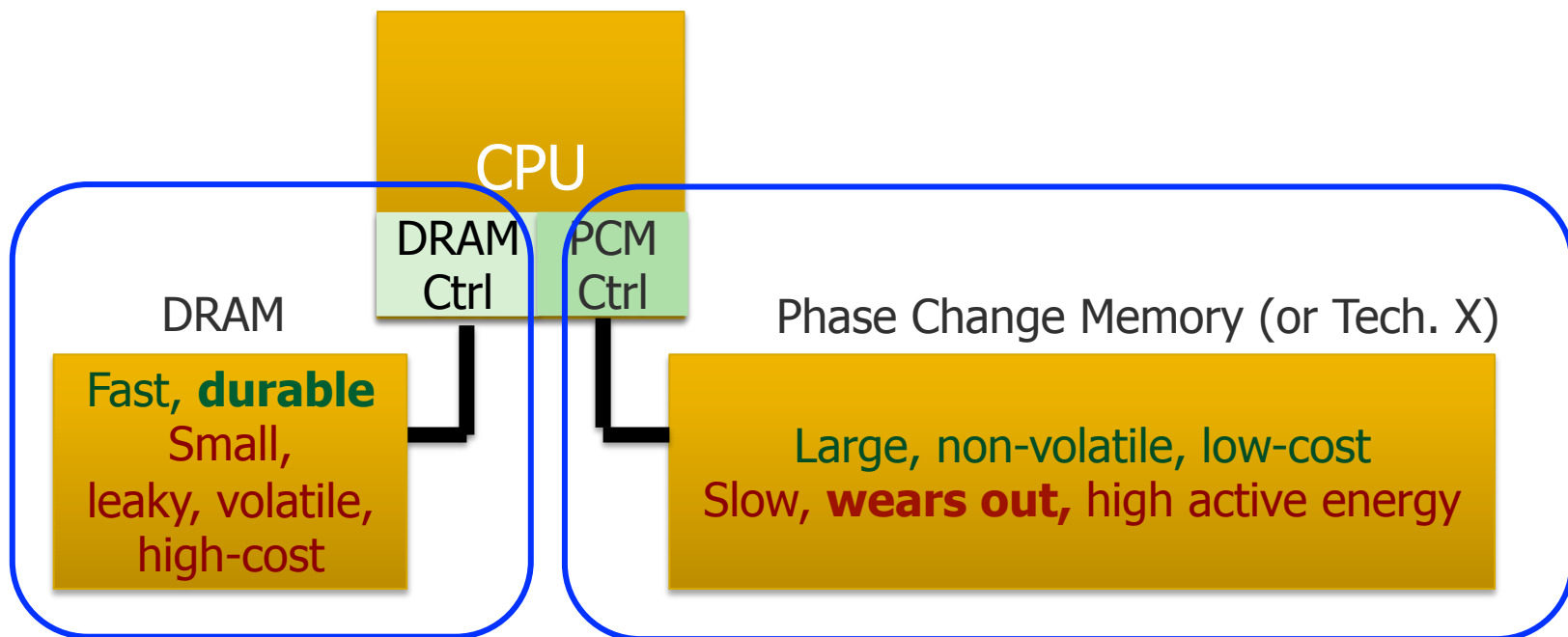
Hardware/software manage data allocation and movement
to achieve the best of multiple technologies

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.

SAFARI

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 Topic 1 (DRAM Scaling)

- **What Will You Learn in This Course**
- 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

SAFARI

What Will You Learn in This Course?

- Scalable Many-Core Memory Systems
  - July 15-19, 2013

- Topic 1: Main memory basics, DRAM scaling
- Topic 2: Emerging memory technologies and hybrid memories
- Topic 3: Main memory interference and QoS
- Topic 4 (unlikely): Cache management
- Topic 5 (unlikely): Interconnects

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

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)

SAFARI

Course Information

- My Contact Information
  - Onur Mutlu
  - onur@cmu.edu
  - http://users.ece.cmu.edu/~omutlu
  - +1-512-658-0891 (my cell phone)
  - Find me during breaks and/or email any time.

- Website for Course Slides and Papers
  - http://users.ece.cmu.edu/~omutlu/acaces2013-memory.html
  - http://users.ece.cmu.edu/~omutlu

SAFARI

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)

Online Slides (Longer Versions)

- Topic 1: DRAM Basics and DRAM Scaling
  - http://users.ece.cmu.edu/~omutlu/pub/onur-ACACES2013-Topic1-dram-basics-and-scaling.pptx
  - http://users.ece.cmu.edu/~omutlu/pub/onur-ACACES2013-Topic1-dram-basics-and-scaling.pdf

- Topic 2: Emerging Technologies and Hybrid Memories
  - http://users.ece.cmu.edu/~omutlu/pub/onur-ACACES2013-Topic2-emerging-and-hybrid-memory-technologies.pptx
  - http://users.ece.cmu.edu/~omutlu/pub/onur-ACACES2013-Topic2-emerging-and-hybrid-memory-technologies.pdf

- Topic 3: Memory Interference and QoS-Aware Memory Systems
  - http://users.ece.cmu.edu/~omutlu/pub/onur-ACACES2013-Topic3-memory-qos.pptx
  - http://users.ece.cmu.edu/~omutlu/pub/onur-ACACES2013-Topic3-memory-qos.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 Topic 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 Topic 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.

- Cai et al., "Error Analysis and Retention-Aware Error Management for NAND Flash Memory," ITJ 2013.

Readings for Topic 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 Topic 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.

Readings in Flash Memory

- Yu Cai, Gulay Yalcin, <u>Onur Mutlu</u>, 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, <u>Onur Mutlu</u>, 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, <u>Onur Mutlu</u>, 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, <u>Onur Mutlu</u>, 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 Topic 1 (DRAM Scaling)

- 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
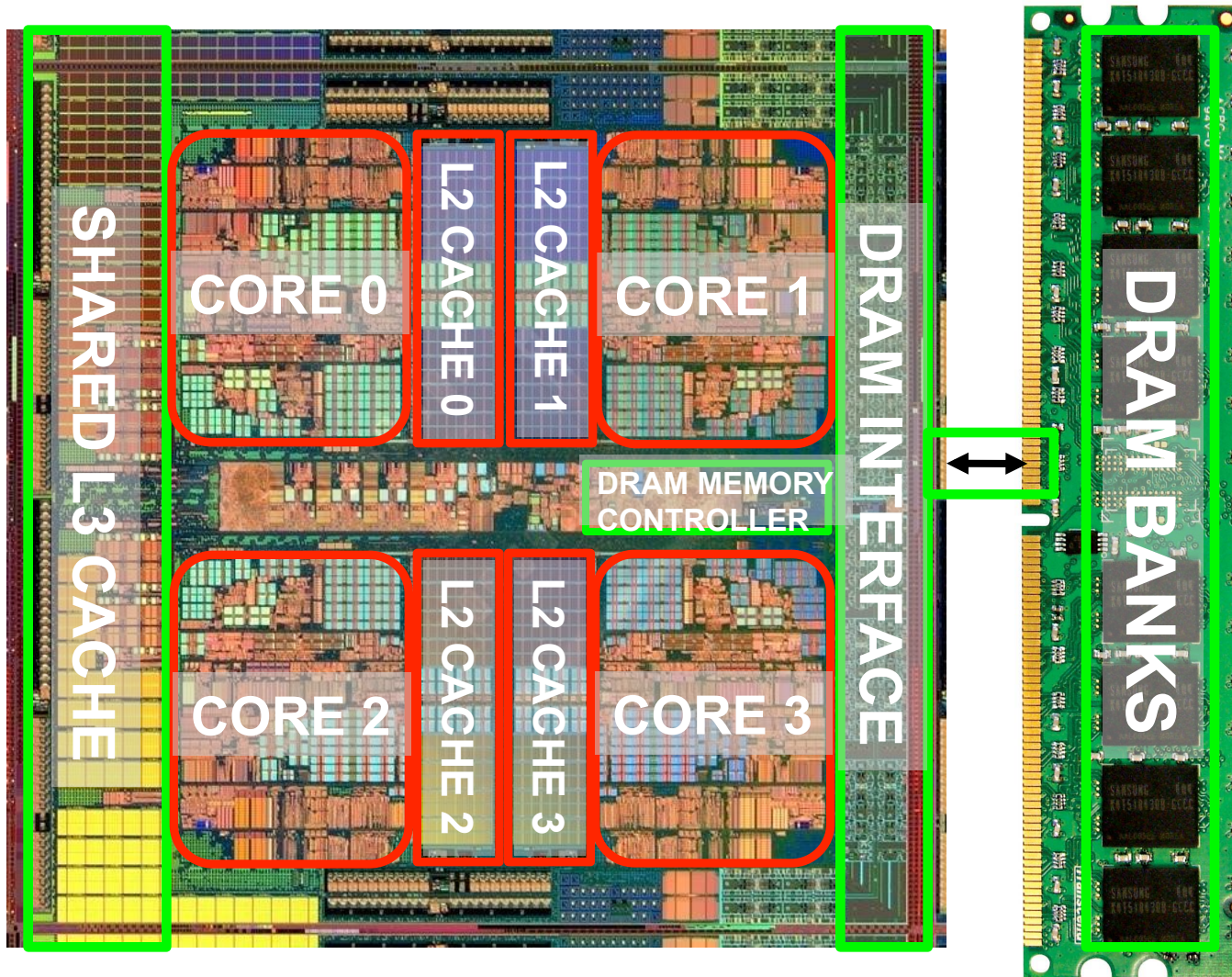
Ideal Memory

- 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.
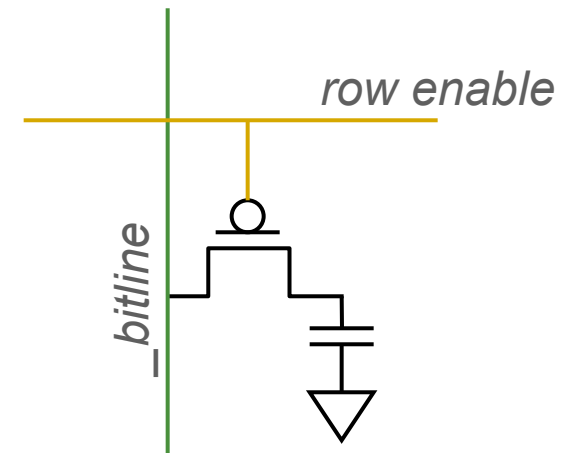
Memory Technology: SRAM

- 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
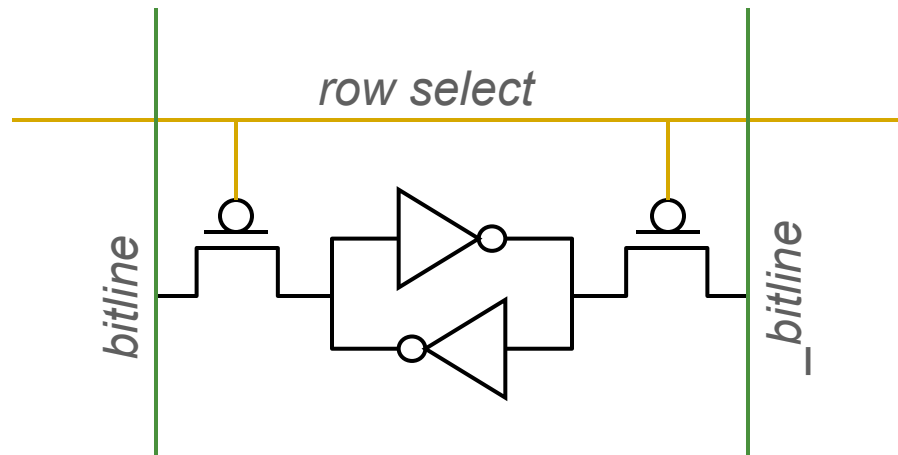
row select

bitline

_bitline

An Aside: Phase Change Memory

■ Phase change material (chalcogenide glass) exists in two states:

❑ Amorphous: Low optical reflexivity and high electrical resistivity

❑ Crystalline: High optical reflexivity and low electrical resistivity
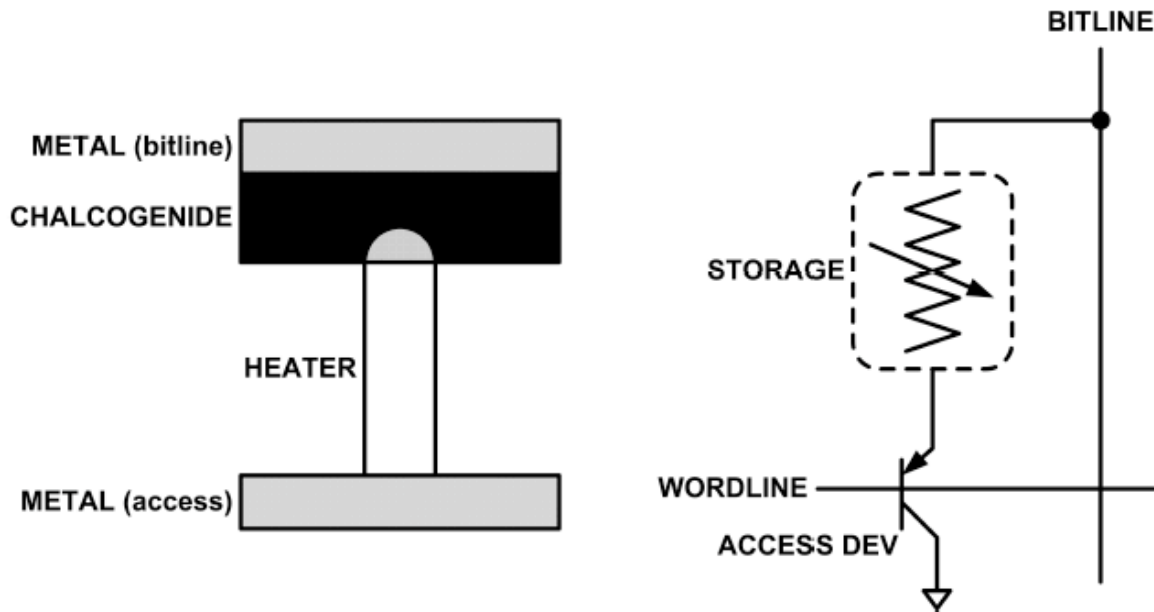
PCM is resistive memory: High resistance (0), Low resistance (1)

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

Memory Bank: A Fundamental Concept

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

  - Goal: Reduce the latency of memory array access and enable multiple accesses in parallel

  - Idea: Divide the array into multiple banks that can be accessed independently (in the same cycle or in consecutive cycles)
    - Each bank is smaller than the entire memory storage
    - Accesses to different banks can be overlapped

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

Memory Bank Organization and Operation
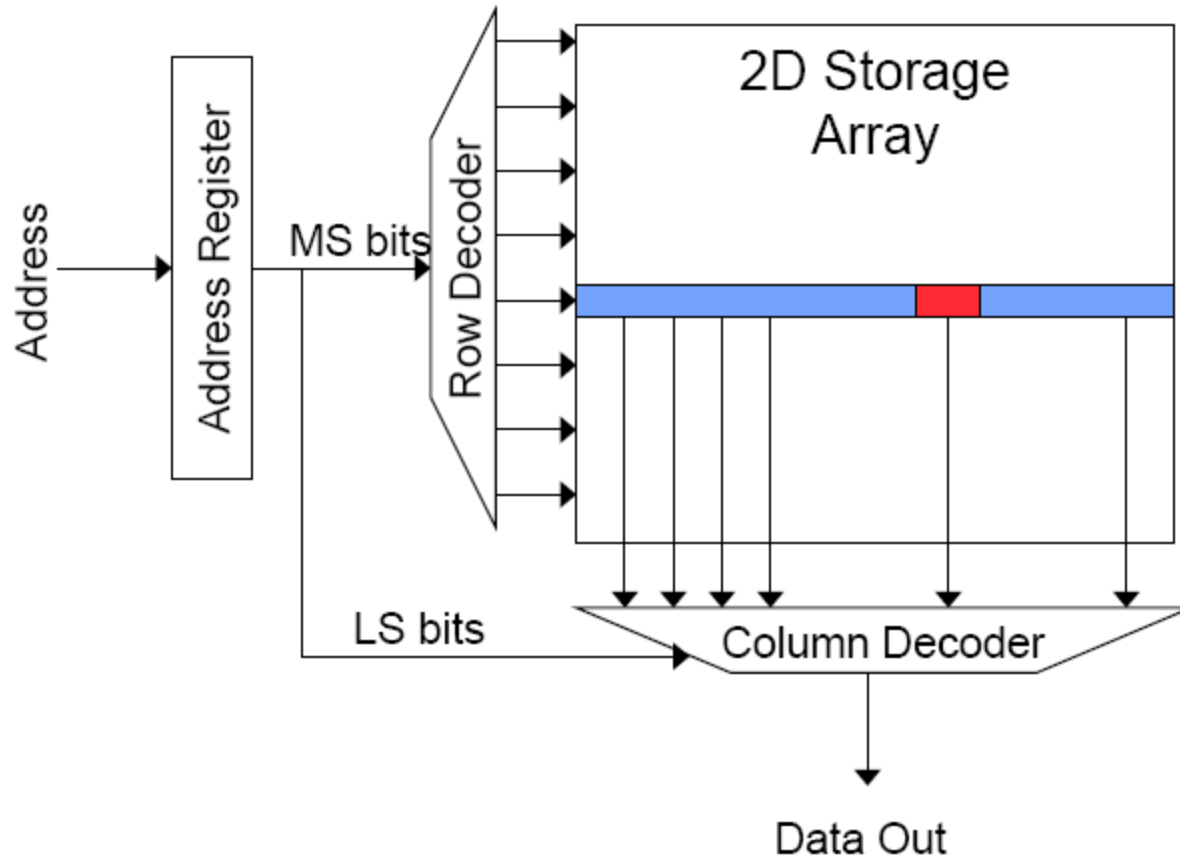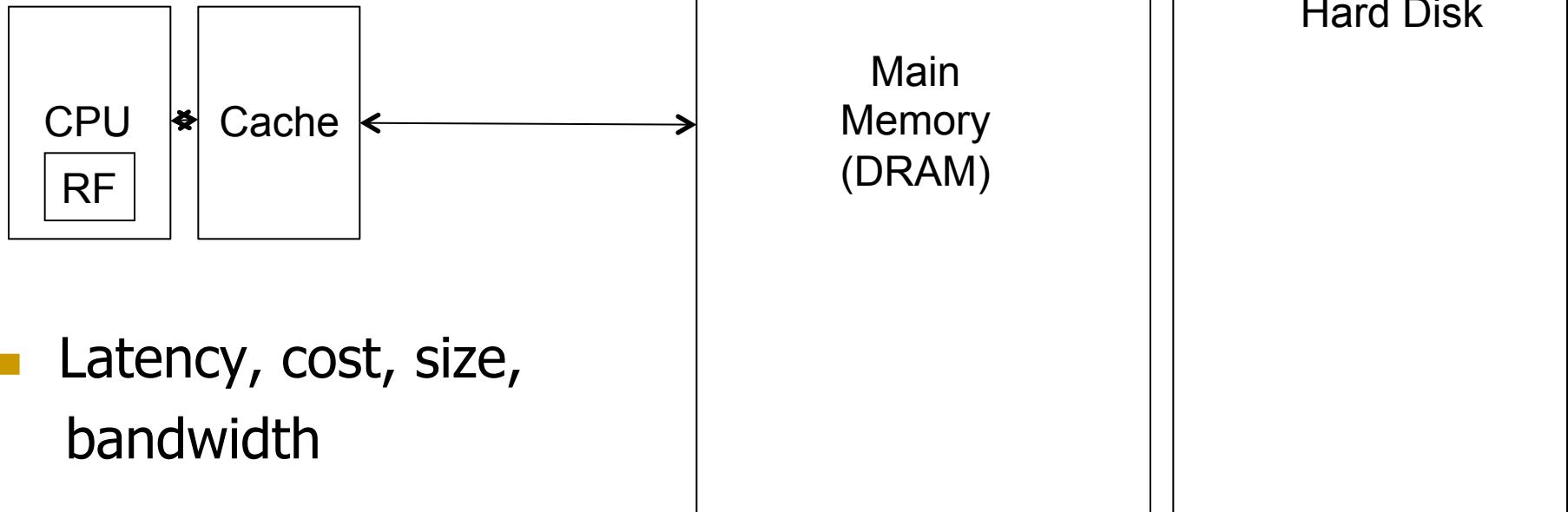

- Read access sequence:

1. Decode row address & drive word-lines

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

3. Amplify row data

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

5. Precharge bit-lines
   - For next access

Why Memory Hierarchy?

- We want both fast and large

- But we cannot achieve both with a single level of memory

- Idea: Have multiple levels of storage (progressively bigger and slower as the levels are farther from the processor) and ensure most of the data the processor needs is kept in the fast(er) level(s)

Memory Hierarchy

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

```
┌──────────────┐  ┌──────────┐        ┌───────────┐  ┌───────────┐
│ CPU          │  │          │        │           │  │           │
│       ┌────┐ │◄►│  Cache   │◄──────►│  Main     │  │ Hard Disk │
│  RF   │    │ │  │          │        │  Memory   │  │           │
│       └────┘ │  │          │        │  (DRAM)   │  │           │
└──────────────┘  └──────────┘        └───────────┘  └───────────┘
```

- **Latency, cost, size, bandwidth**

Caching Basics: Exploit Temporal Locality

- Idea: Store recently accessed data in automatically managed fast memory (called cache)
- Anticipation: the data will be accessed again soon

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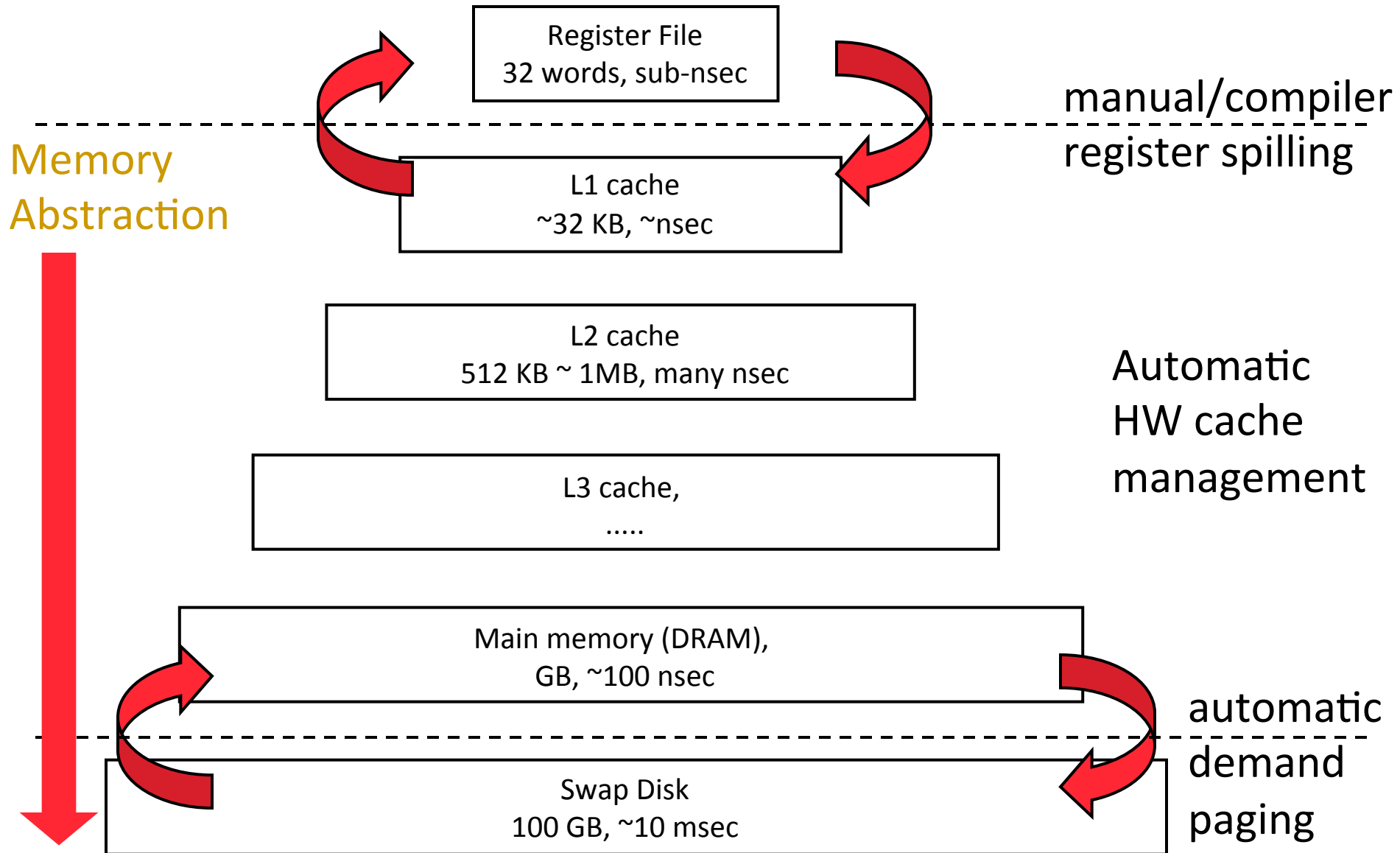   - Wilkes, "Slave Memories and Dynamic Storage Allocation," IEEE Trans. On Electronic Computers, 1965.
    - "The use is discussed of a fast core memory of, say 32000 words as a slave to a slower core memory of, say, one million words in such a way that in practical cases the effective access time is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

Caching Basics: Exploit Spatial Locality

- Idea: Store addresses adjacent to the recently accessed one in automatically managed fast memory
    - Logically divide memory into equal size blocks
    - Fetch to cache the accessed block in its entirety
- Anticipation: nearby data will be accessed soon

- Spatial locality principle
    - Nearby data in memory will be accessed in the near future
        - E.g., sequential instruction access, array traversal
    - This is what IBM 360/85 implemented
        - 16 Kbyte cache with 64 byte blocks
        - Liptay, "Structural aspects of the System/360 Model 85 II: the cache," IBM Systems Journal, 1968.

A Note on Manual vs. Automatic Management

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

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

Automatic Management in Memory Hierarchy

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

Slave Memories and Dynamic Storage Allocation

M. V. WILKES

SUMMARY

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

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

A Modern Memory Hierarchy

Memory Abstraction

Register File
32 words, sub-nsec

manual/compiler
register spilling

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
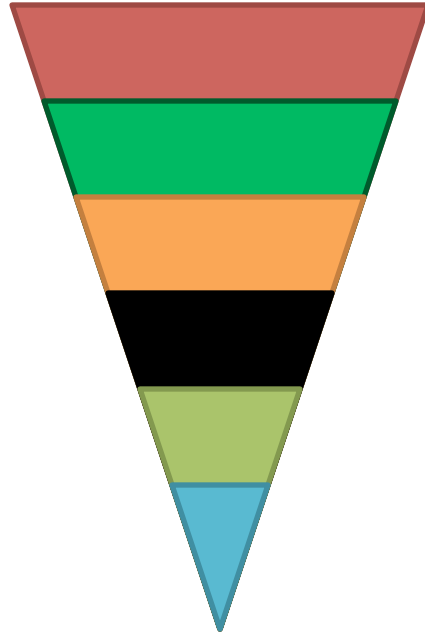
Page Mode DRAM

- 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)

Columns

Row decoder

Rows

Row address 0 1

Row Buffer   HIT CONFLICT !

Row 1

Column address 0 85

Column mux
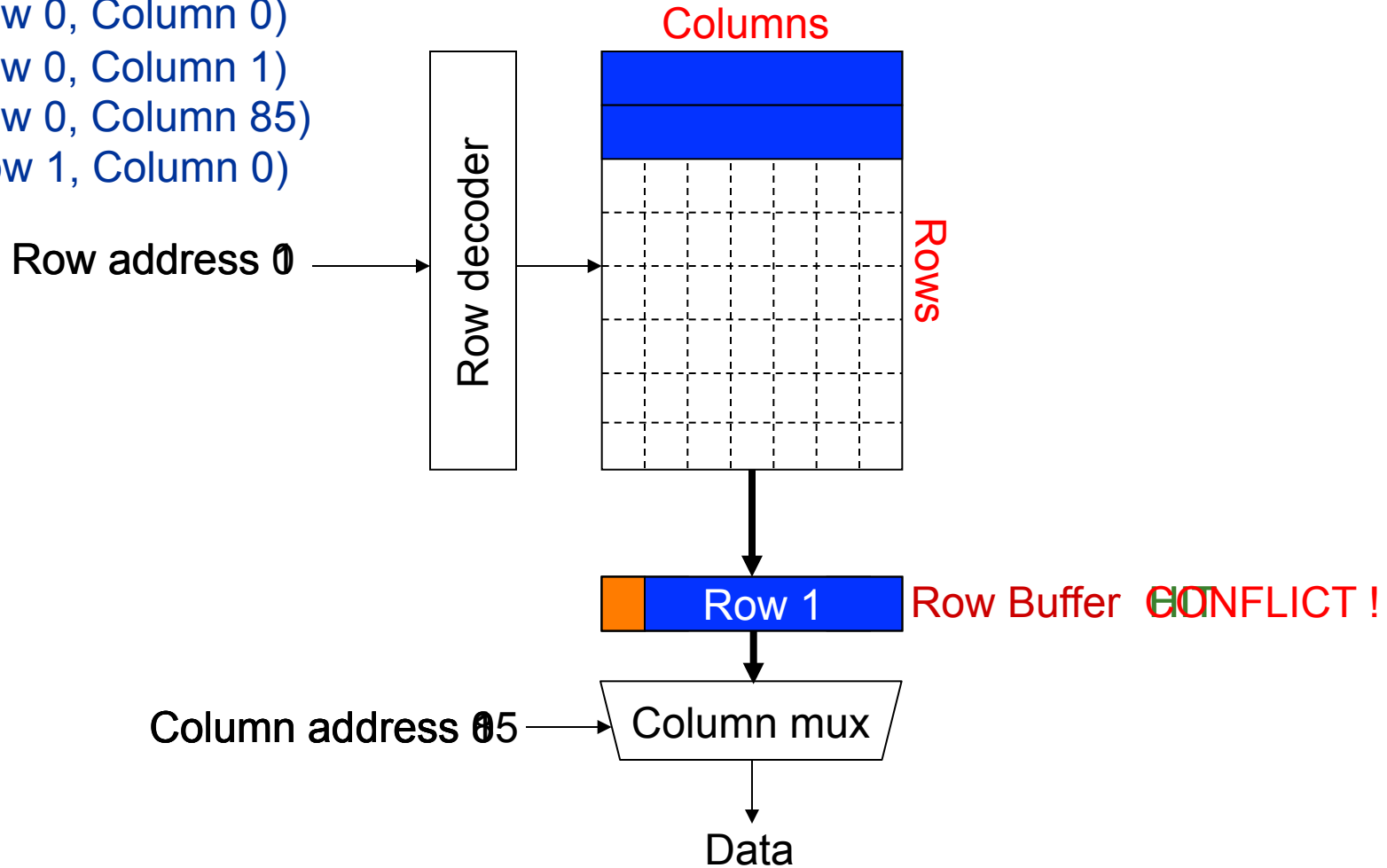
Data

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
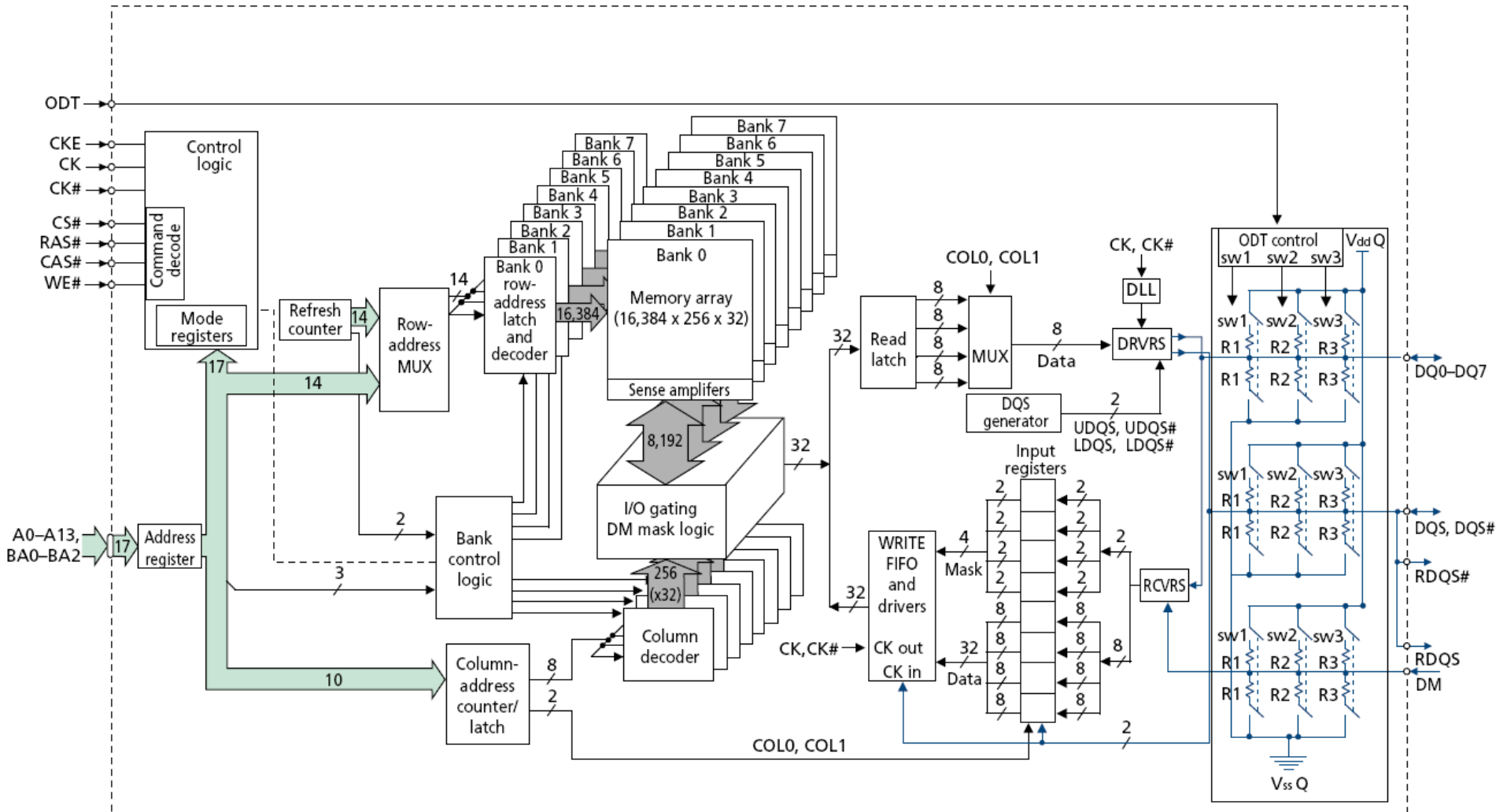
DRAM Rank and Module

- **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)
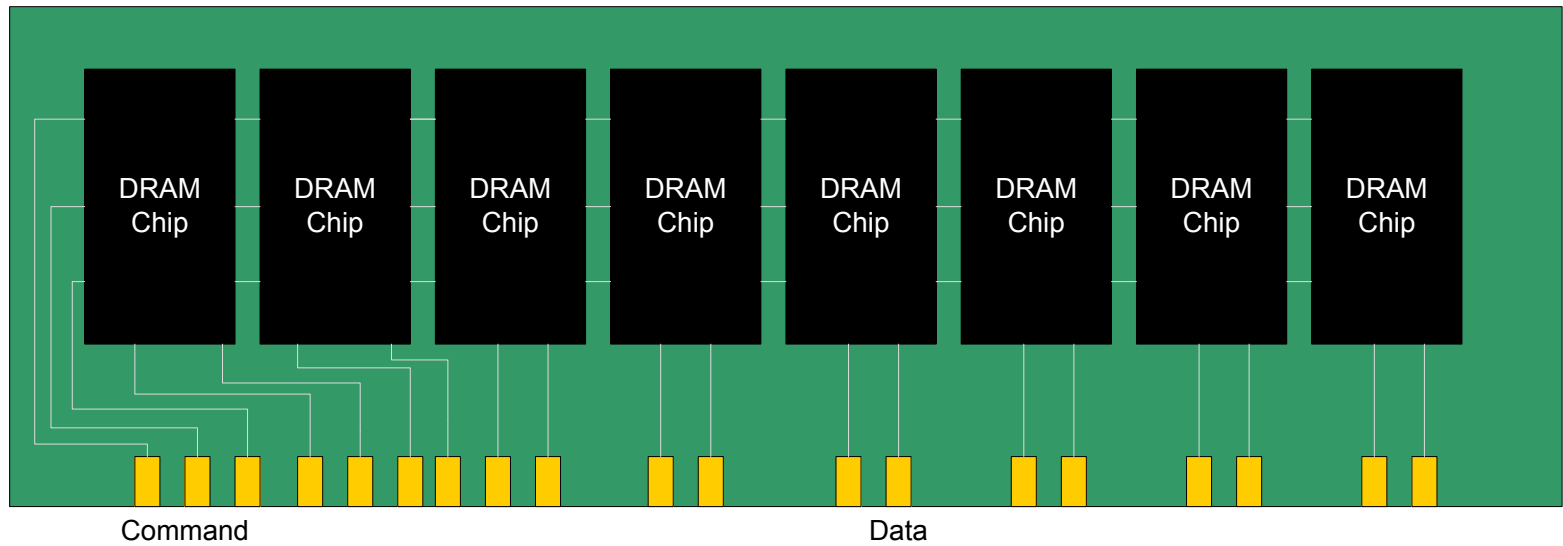
A 64-bit Wide DIMM (One Rank)
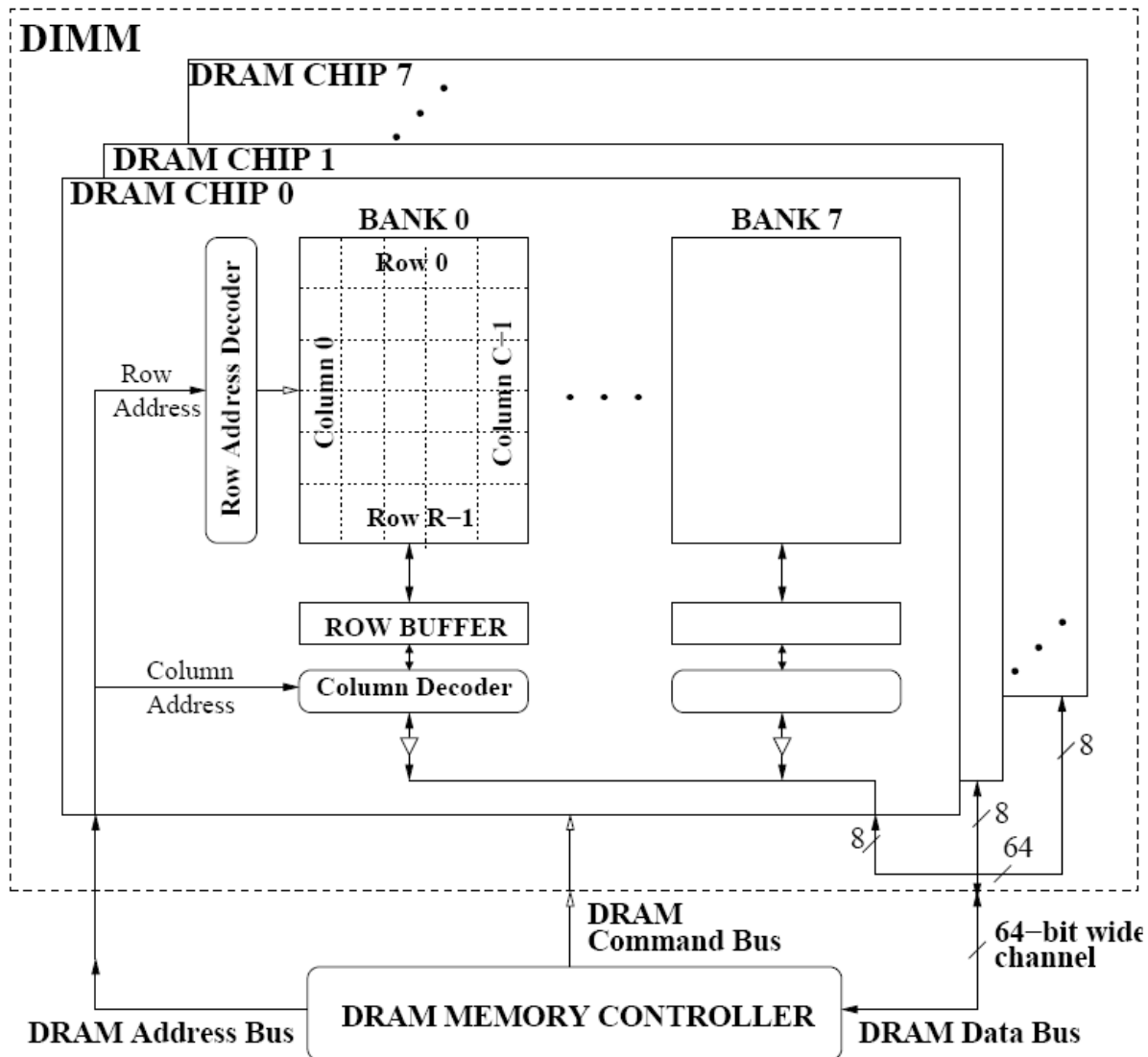
- **Advantages:**
  - Acts like a high-capacity DRAM chip with a wide interface
  - Flexibility: memory controller does not need to deal with individual chips

- **Disadvantages:**
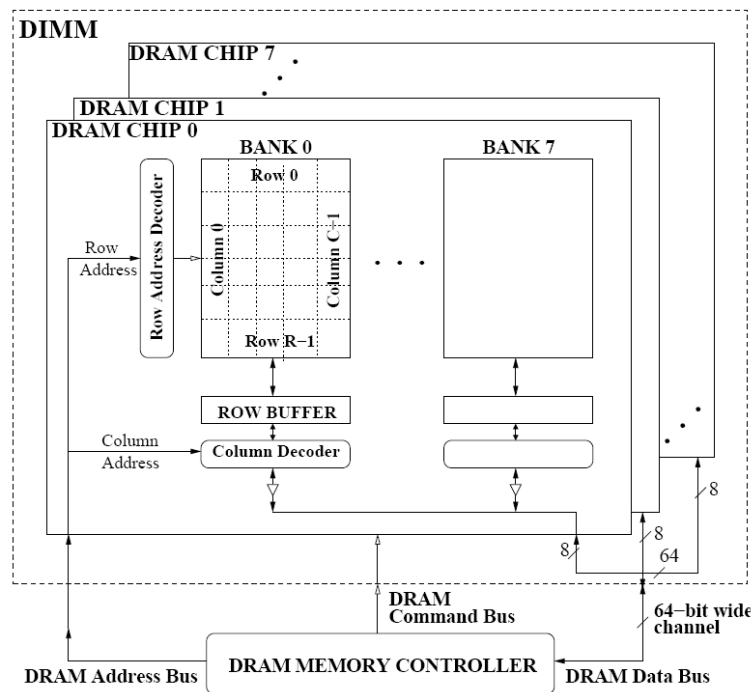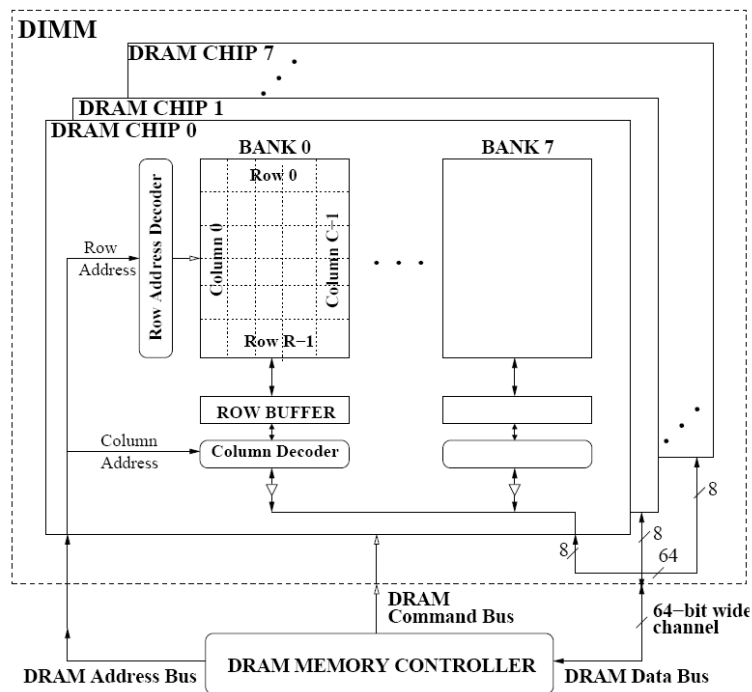 - Granularity: Accesses cannot be smaller than the interface width

Multiple DIMMs

"Mesh Topology"

Legend:
- **Addr & Cmd**
- **Data Bus**
- **Chip (DIMM) Select**
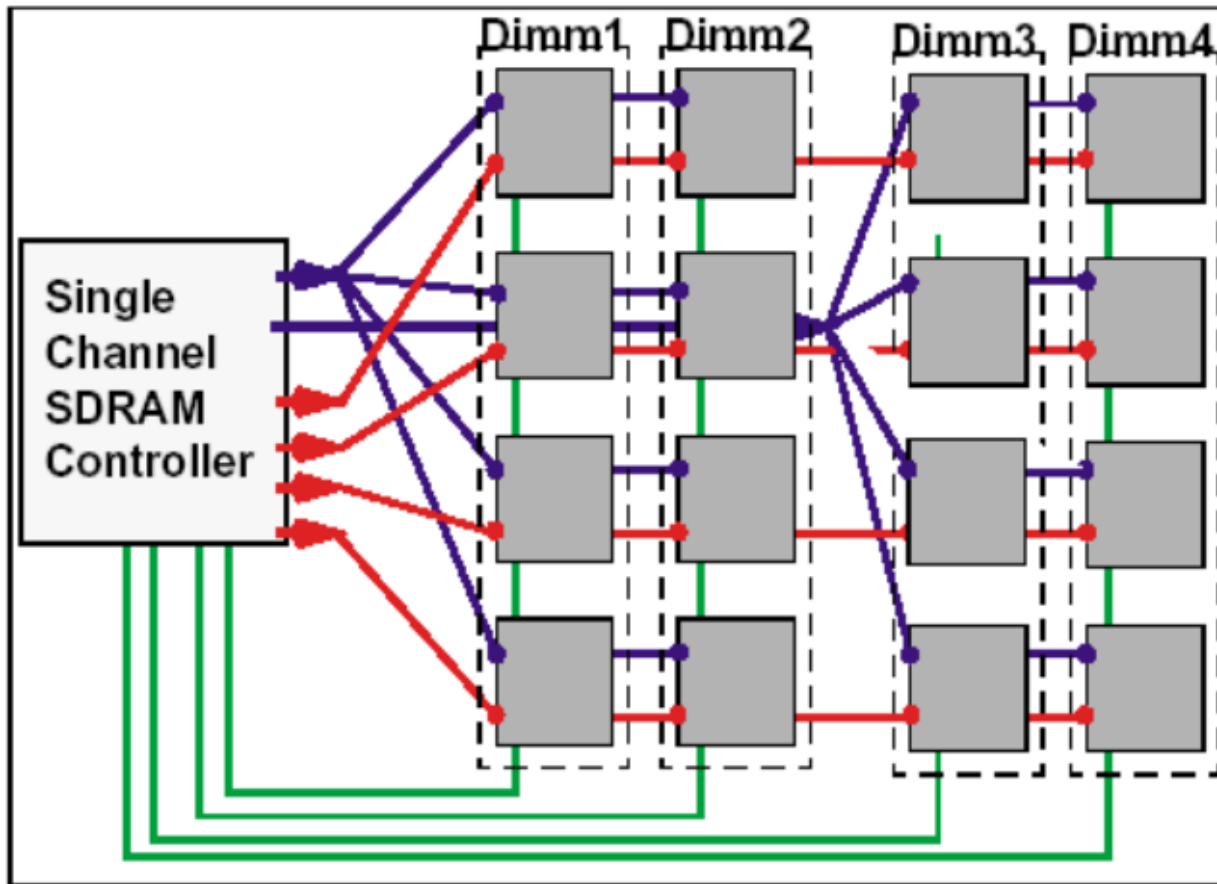
- Advantages:
  - Enables even higher capacity

- Disadvantages:
  - Interconnect complexity and energy consumption can be high
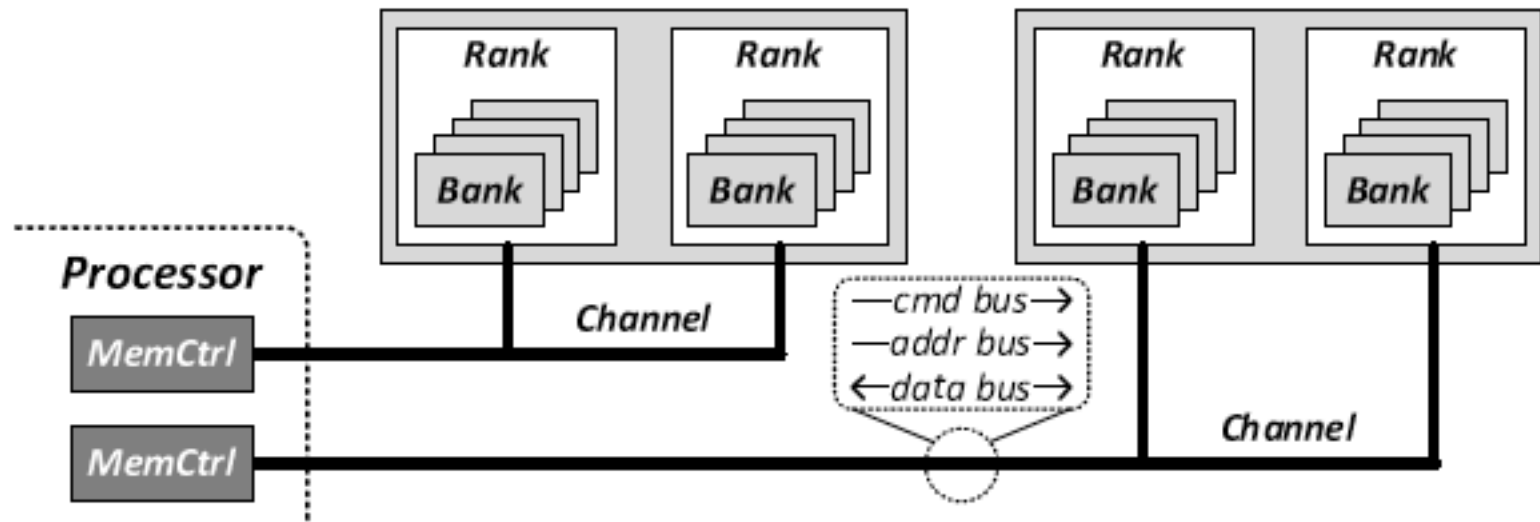
DRAM Channels

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

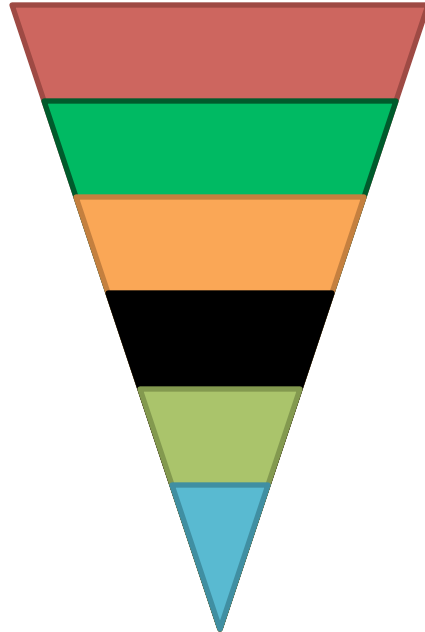# Generalized Memory Structure
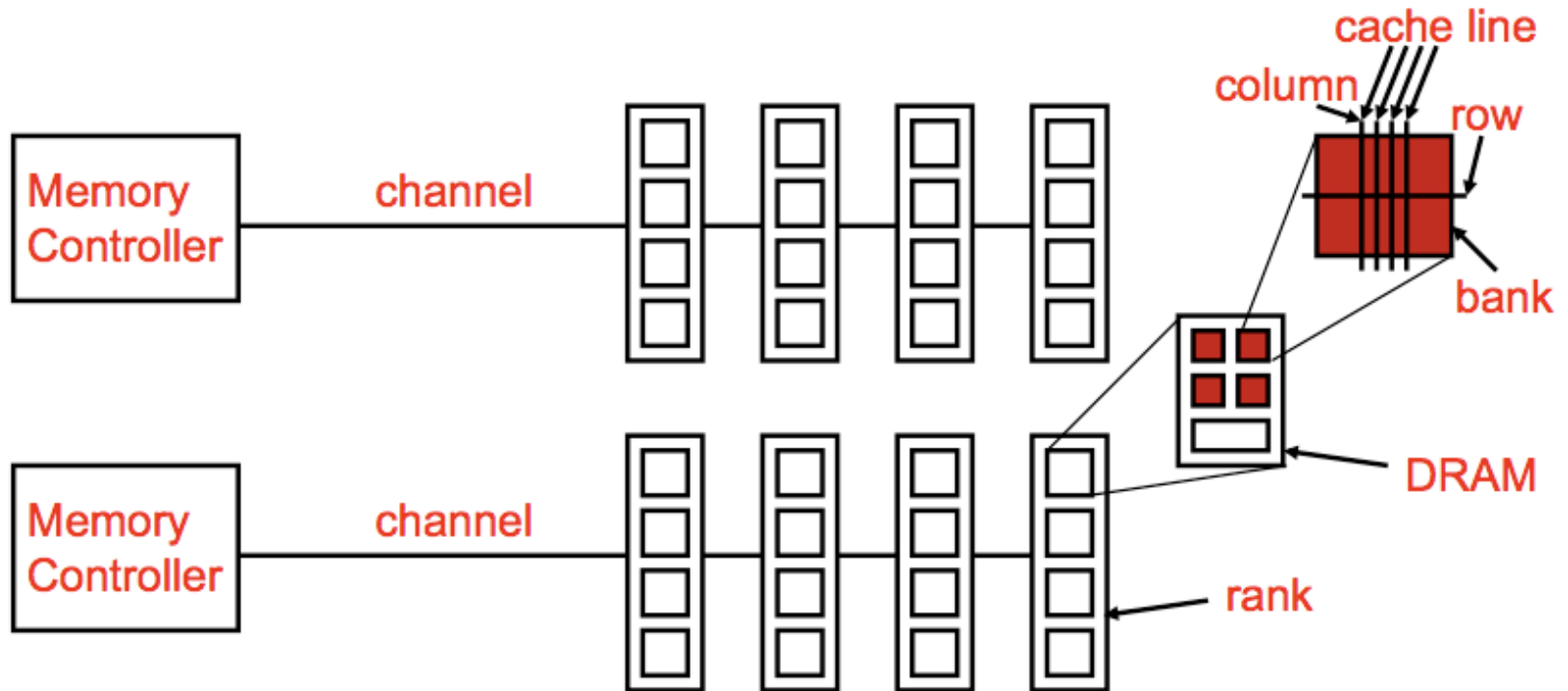
Generalized Memory Structure

Kim+, "A Case for Exploiting Subarray-Level Parallelism in DRAM," ISCA 2012.
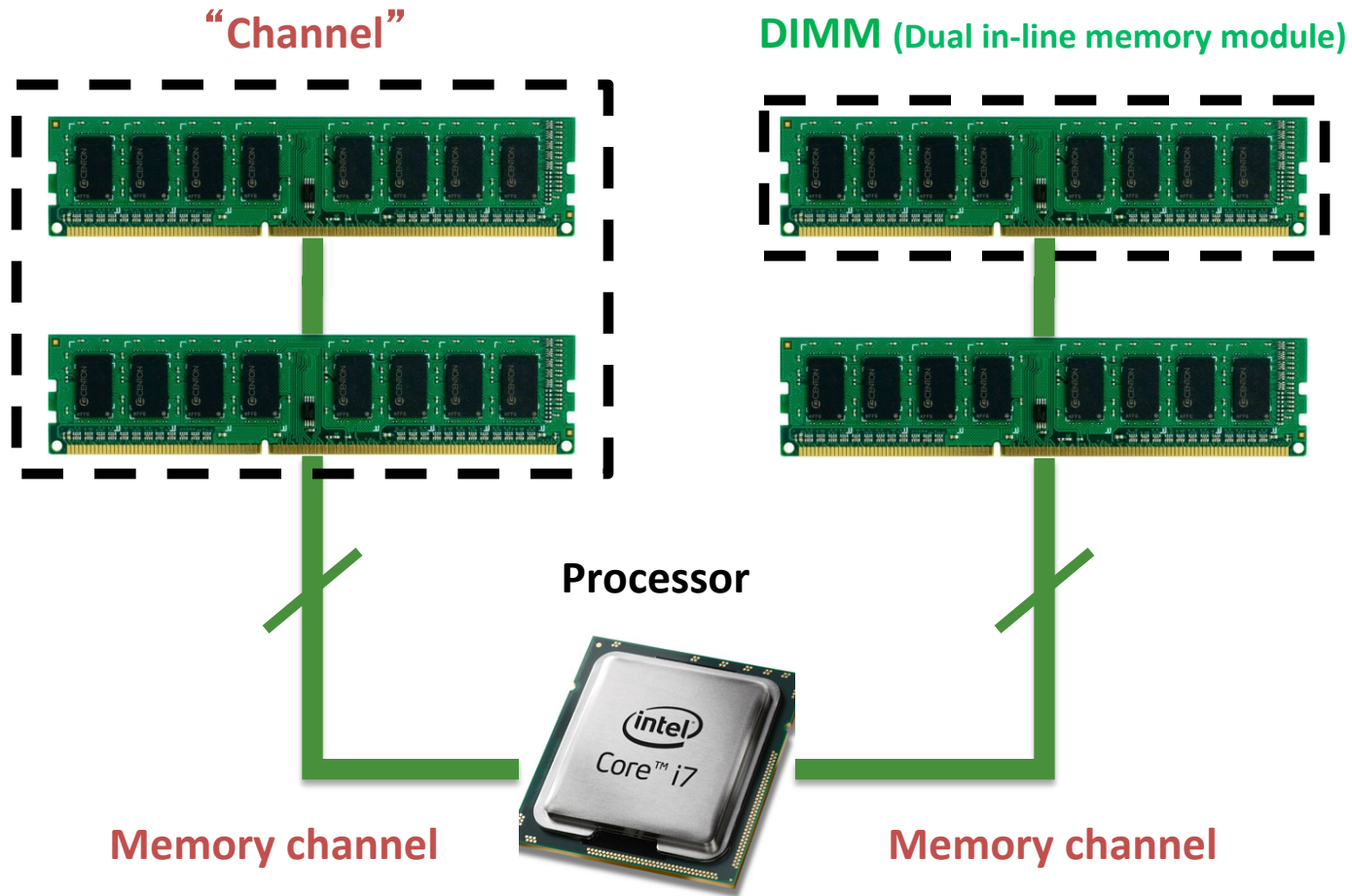
The DRAM Subsystem
The Top Down View

DRAM Subsystem Organization

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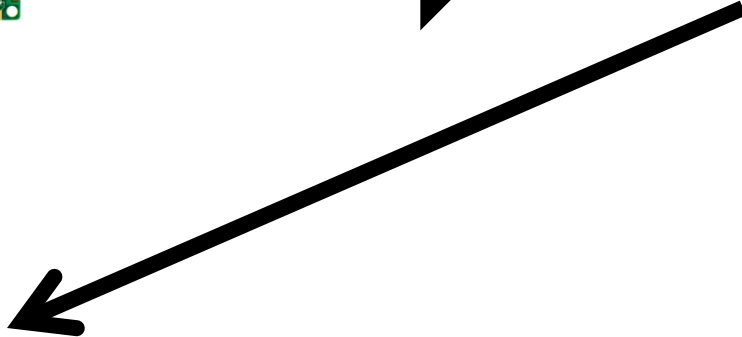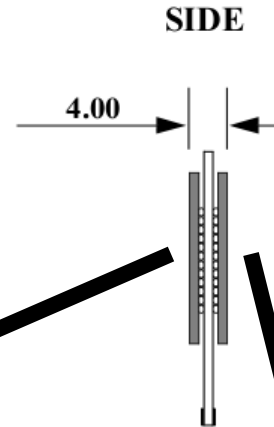- DIMM
- Rank
- Chip
- Bank
- Row/Column

The DRAM subsystem

"Channel"

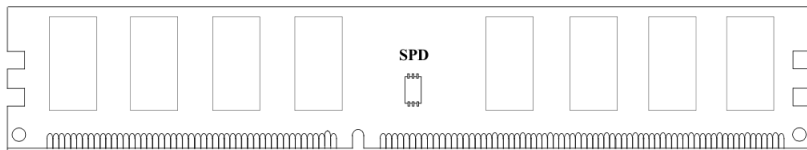DIMM (Dual in-line memory module)

Processor

Memory channel

Memory channel

Breaking down a DIMM

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

Side view

SIDE

4.00

Front of DIMM

SPD

Back of DIMM

Breaking down a DIMM

DIMM (Dual in-line memory module)

Side view

SIDE

4.00

Front of DIMM

Back of DIMM

SPD
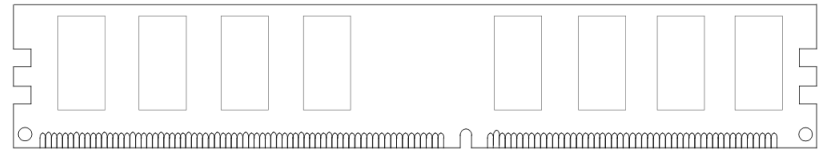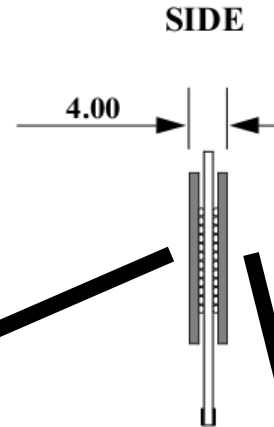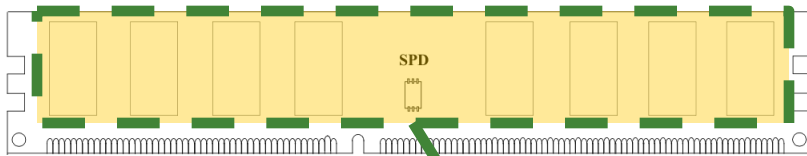
Rank 0: collection of 8 chips

Rank 1

Rank

Rank 0 (Front)   Rank 1 (Back)

<0:63>   <0:63>

Addr/Cmd   CS <0:1>   Data <0:63>
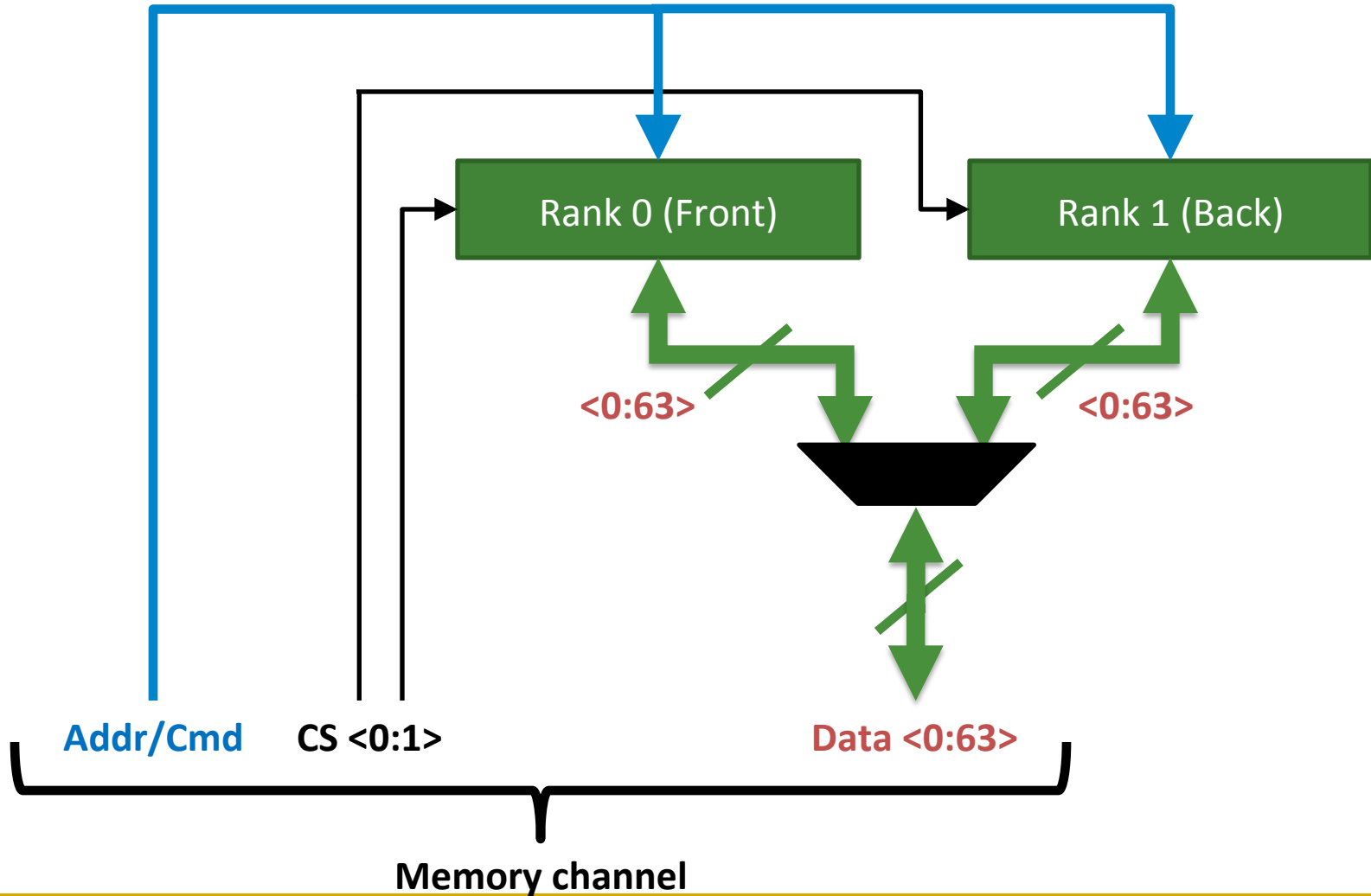
Memory channel

Breaking down a Rank
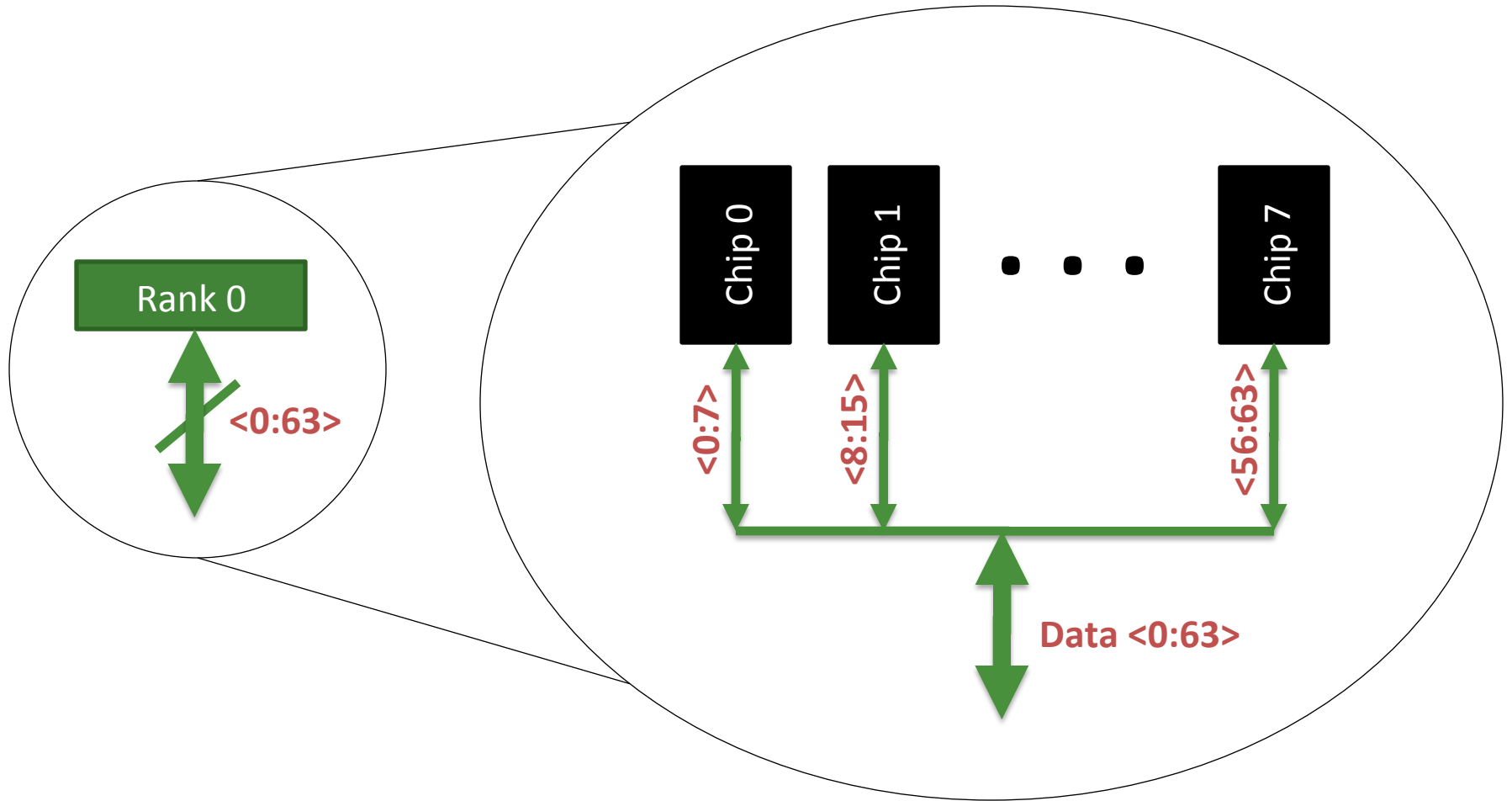
Breaking down a Chip
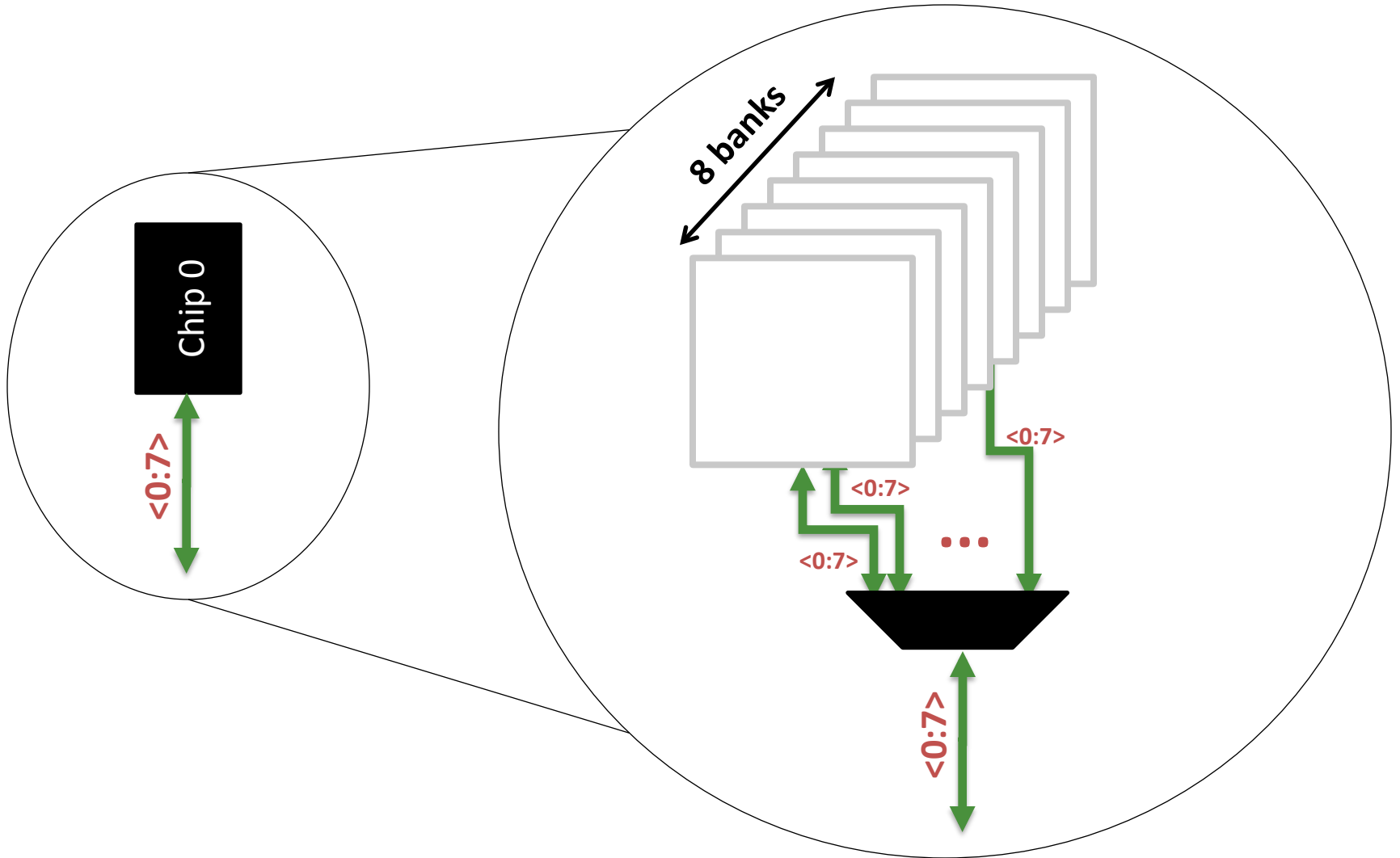
Breaking down a Bank
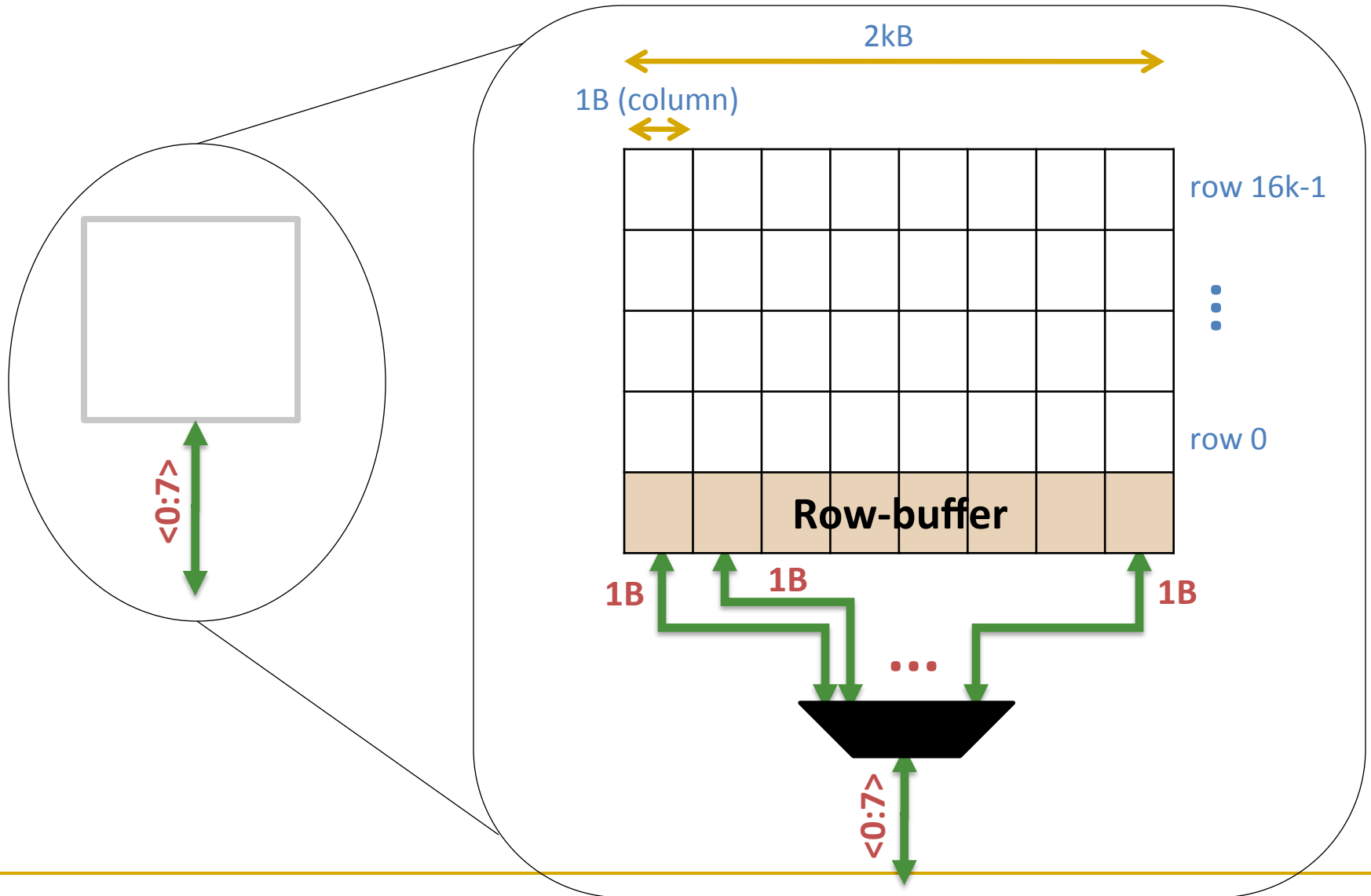
DRAM Subsystem Organization

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

Example: Transferring a cache block
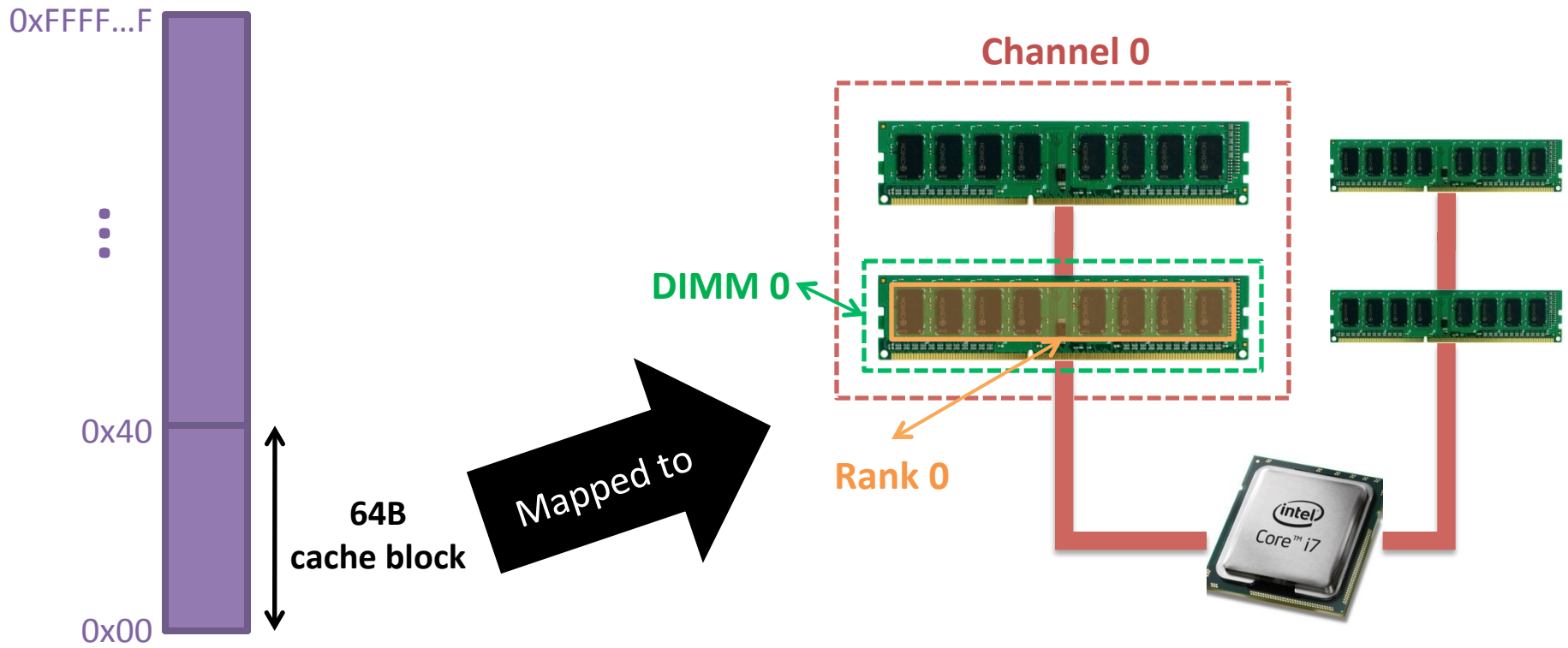
Physical memory space

0xFFFF...F

0x40

0x00

64B cache block

Mapped to

Channel 0

DIMM 0

Rank 0

Example: Transferring a cache block

Physical memory space

0xFFFF...F

⋮

0x40

64B cache block

0x00

Chip 0   Chip 1   **Rank 0**   Chip 7

. . .

<0:7>   <8:15>   <56:63>

Data <0:63>

Example: Transferring a cache block

Physical memory space

0xFFFF...F

0x40

64B
cache block

0x00

Chip 0   Chip 1   Rank 0   Chip 7

Row 0
Col 0

. . .

<0:7>   <8:15>   <56:63>

Data <0:63>
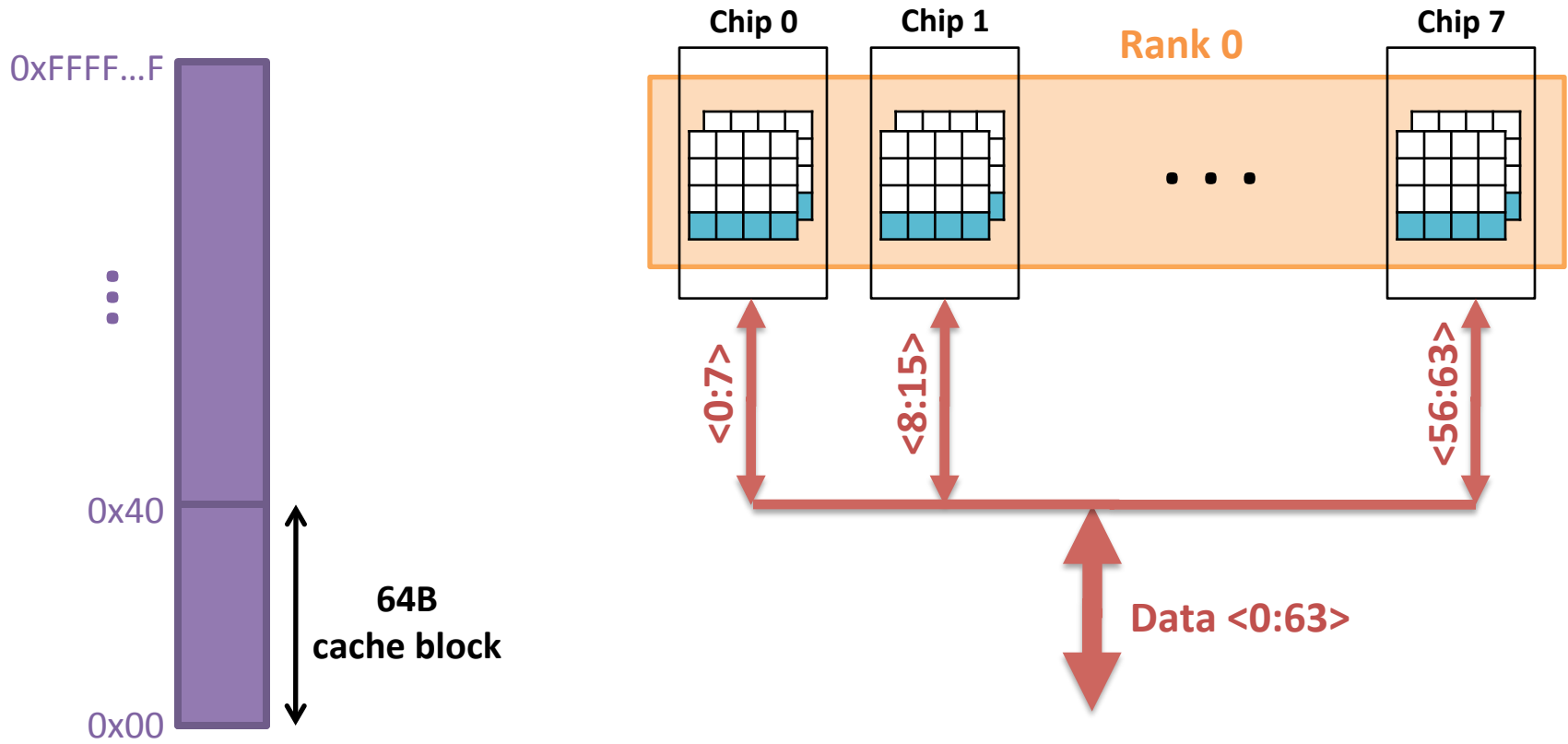
Example: Transferring a cache block
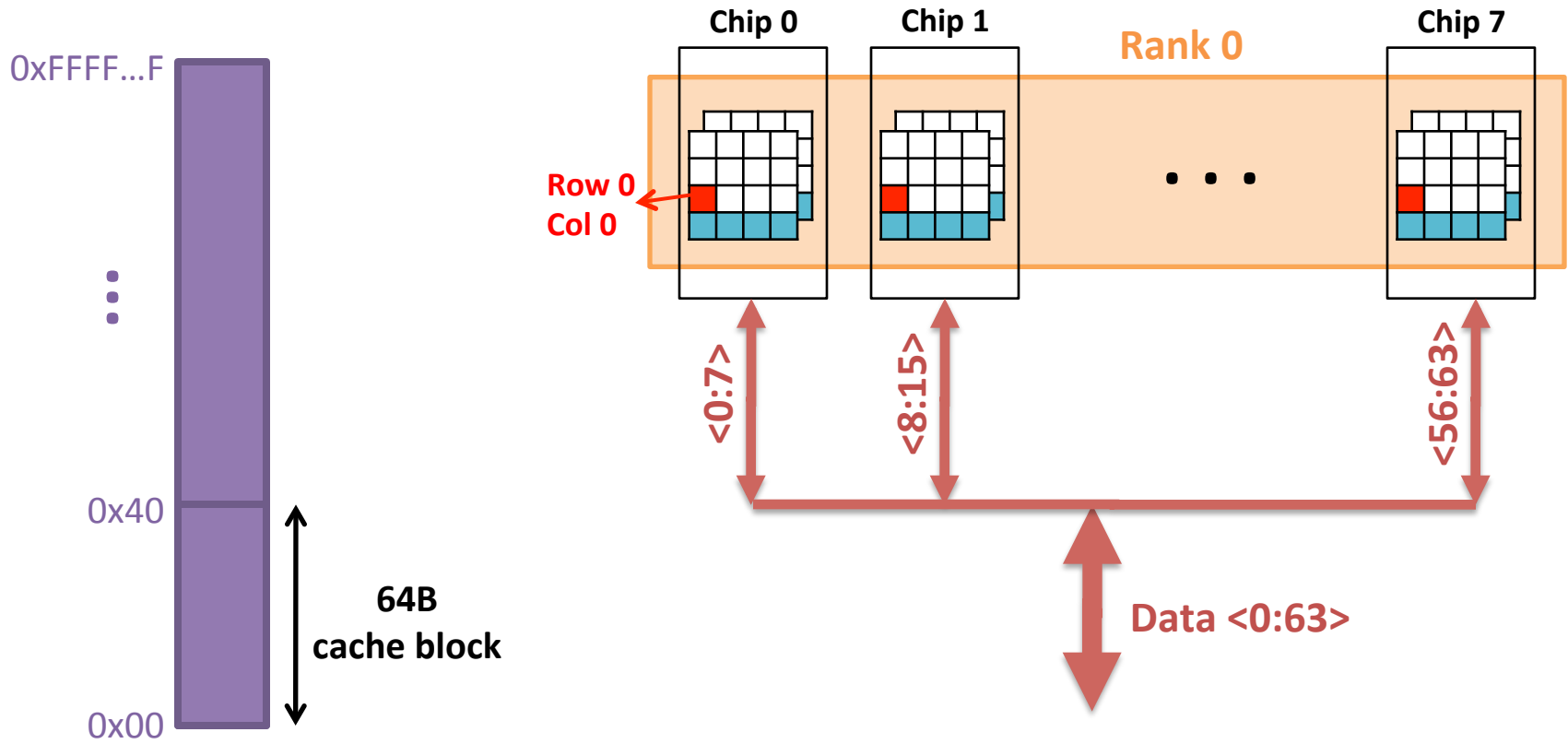
Physical memory space

Example: Transferring a cache block
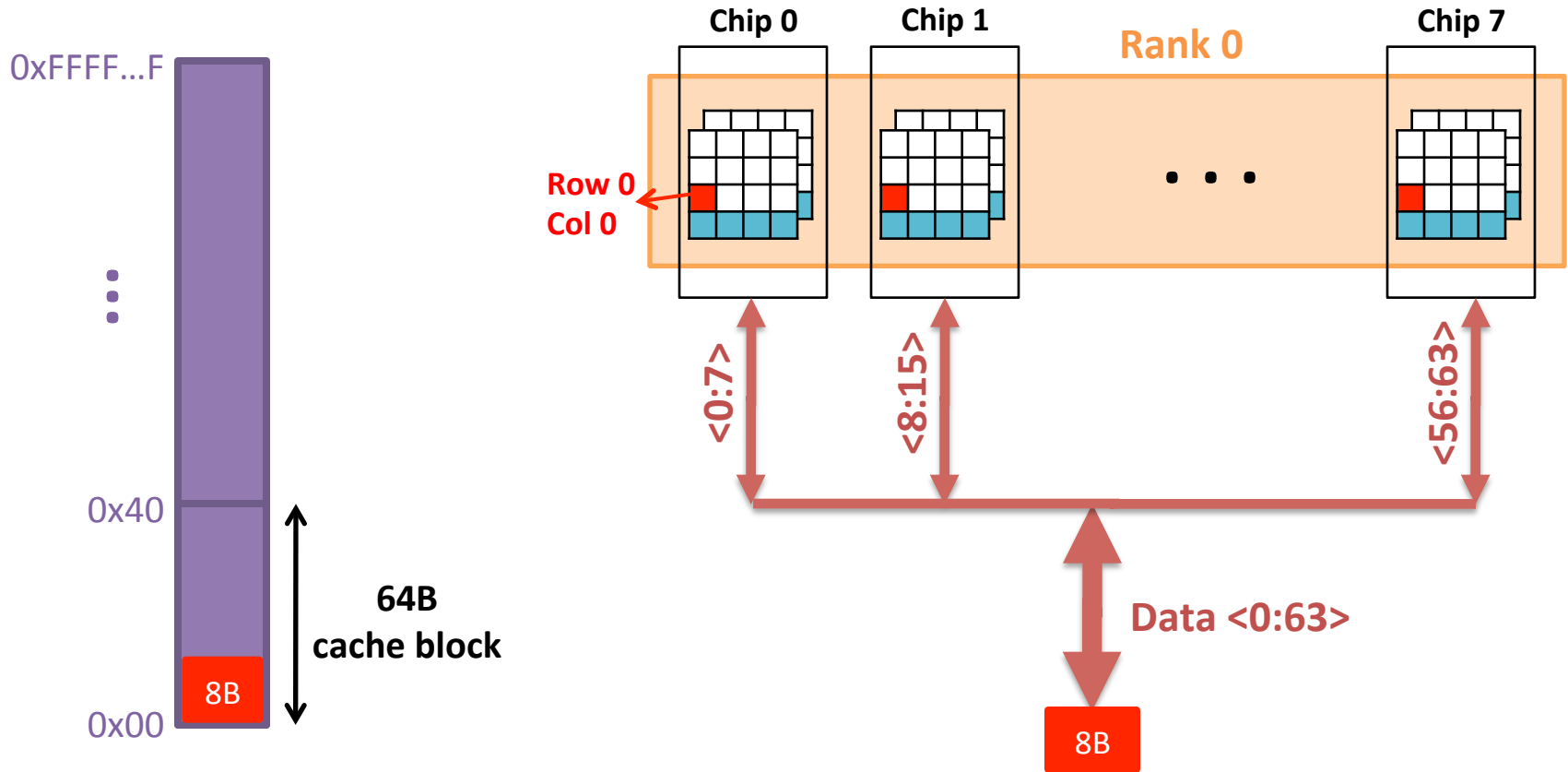
Physical memory space

Example: Transferring a cache block
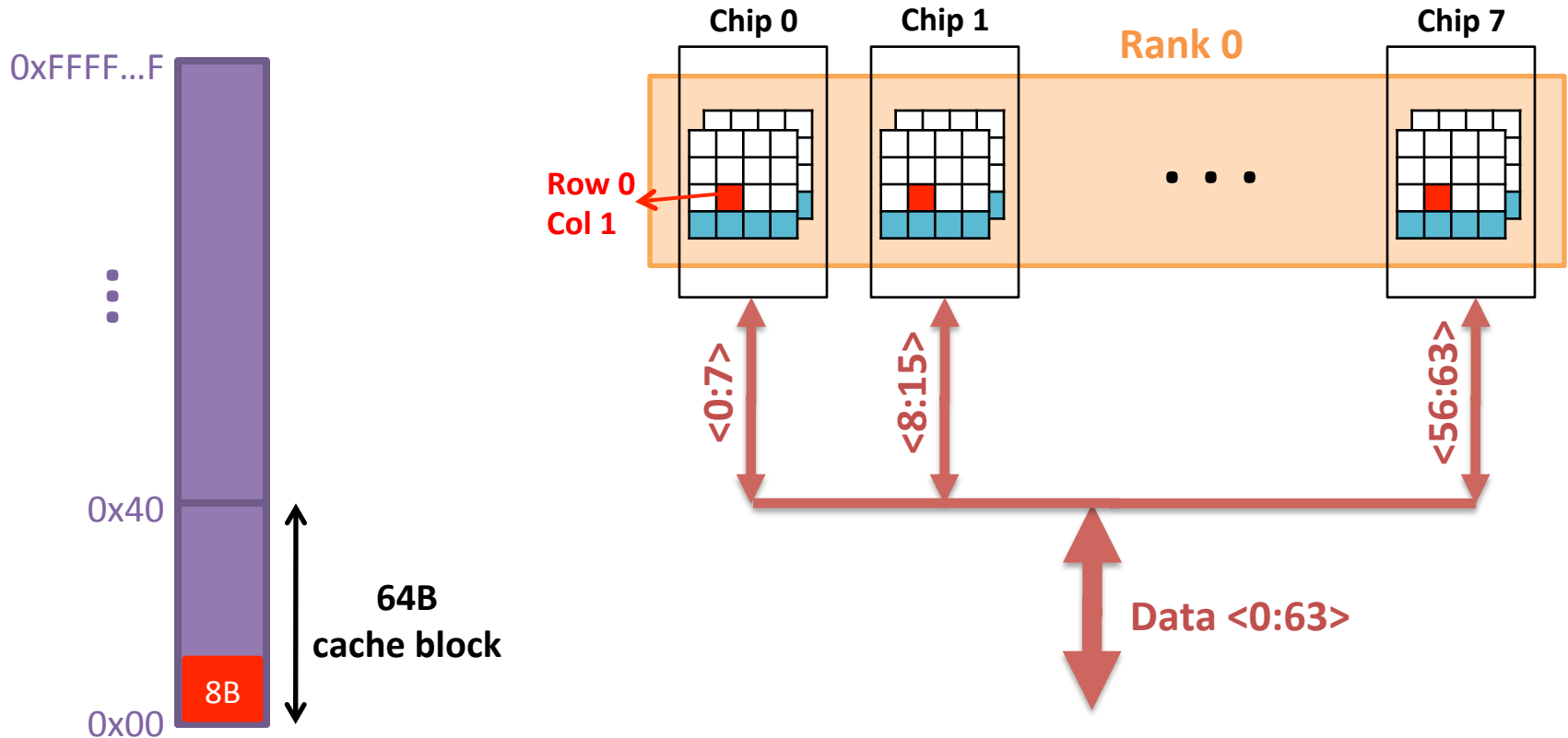
Physical memory space

Example: Transferring a cache block

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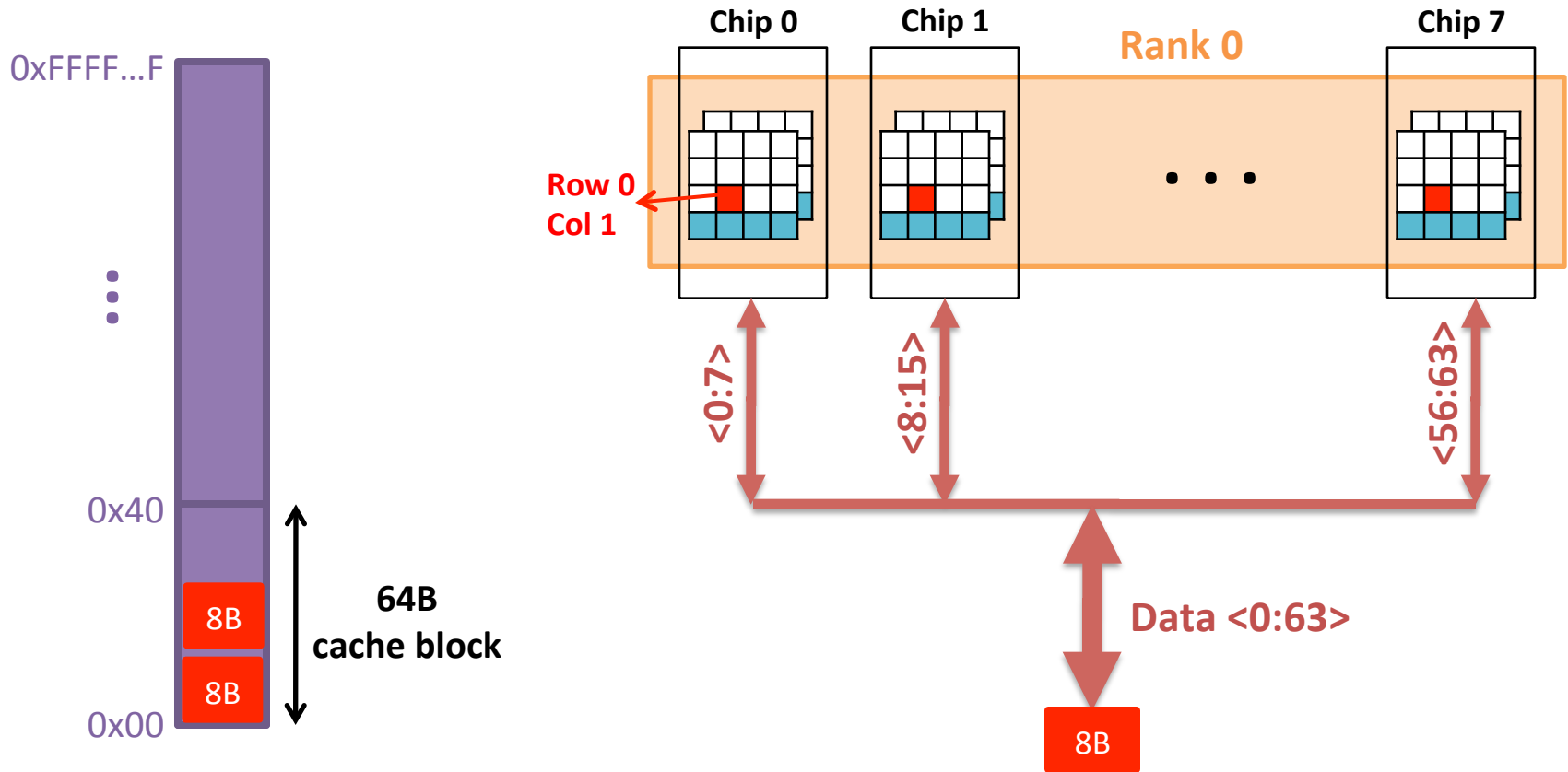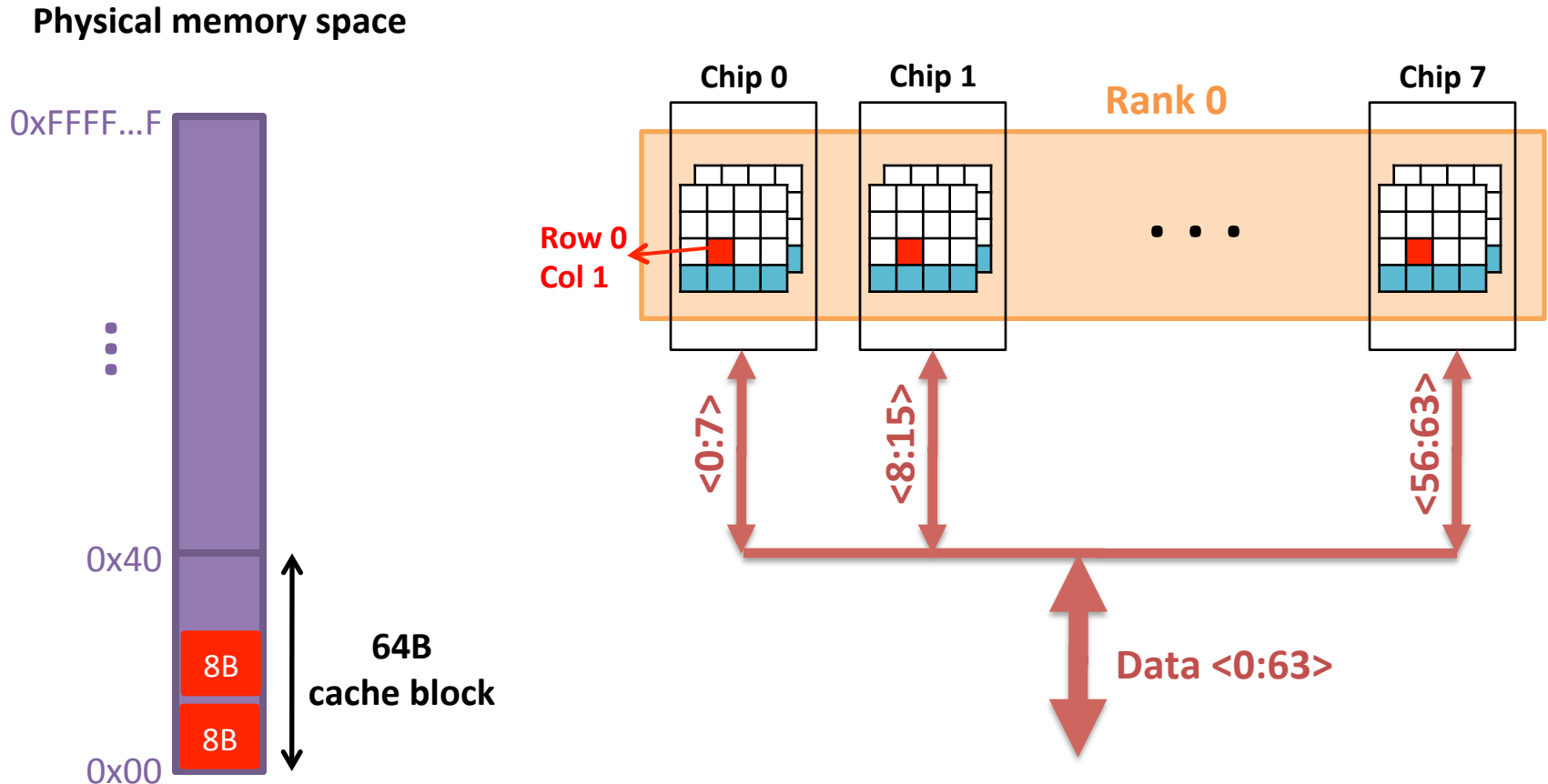
A 64B cache block takes 8 I/O cycles to transfer.

During the process, 8 columns are read sequentially.

Latency Components: Basic DRAM Operation

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

Multiple Banks (Interleaving) and Channels

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

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

How Multiple Banks Help
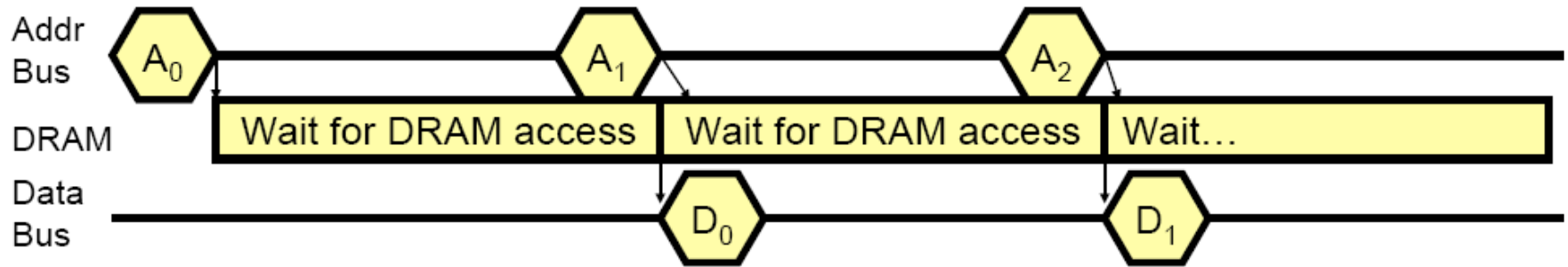
Before: No Overlapping
Assuming accesses to different DRAM rows
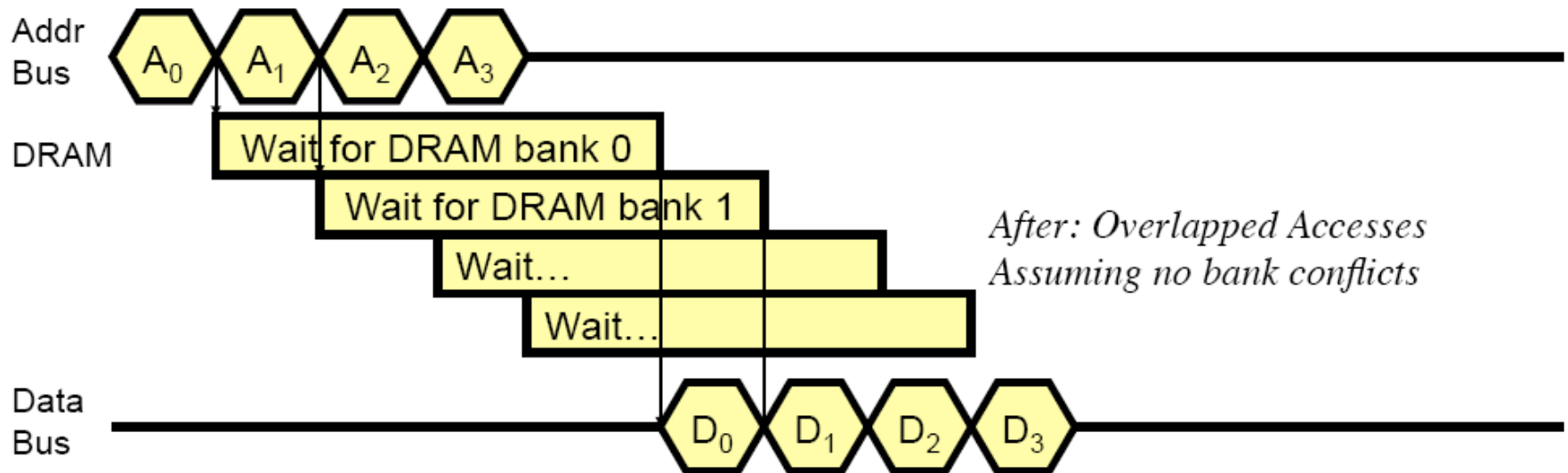
After: Overlapped Accesses
Assuming no bank conflicts

Address Mapping (Single Channel)

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

- Row interleaving
  - Consecutive rows of memory in consecutive banks

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

  - Accesses to consecutive cache blocks serviced in a pipelined manner

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

Row (14 bits)	High Column	Bank (3 bits)	Low Col.	Byte in bus (3 bits)
	8 bits		3 bits	
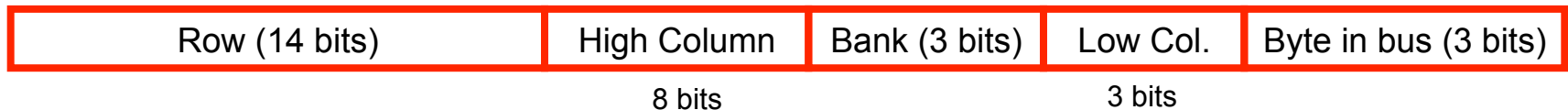
  - Accesses to consecutive cache blocks can be serviced in parallel

Bank Mapping Randomization

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

		3 bits	Column (11 bits)	Byte in bus (3 bits)

XOR

Bank index
(3 bits)
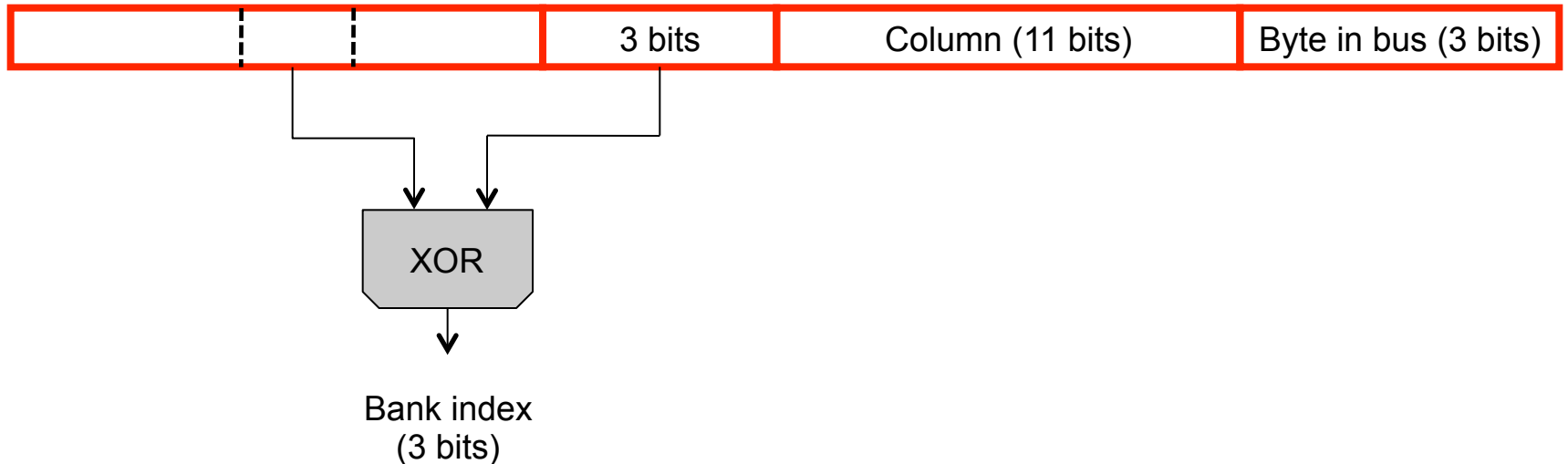
Address Mapping (Multiple Channels)

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

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

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

Row (14 bits)	Bank (3 bits)	Column (11 bits)	C	Byte in bus (3 bits)
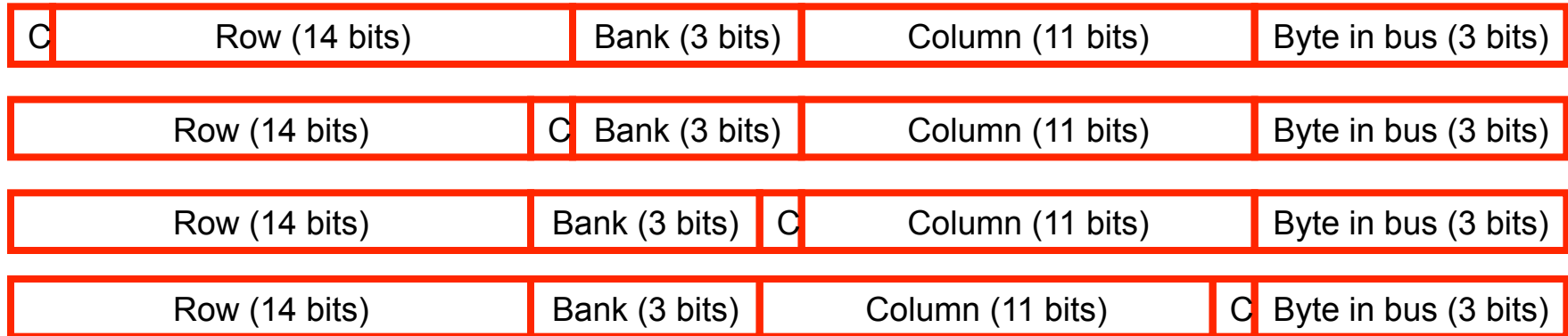
- ## Where are consecutive cache blocks?

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

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

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

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

Row (14 bits)	High Column	Bank (3 bits)	Low Col.	C	Byte in bus (3 bits)
	8 bits		3 bits		
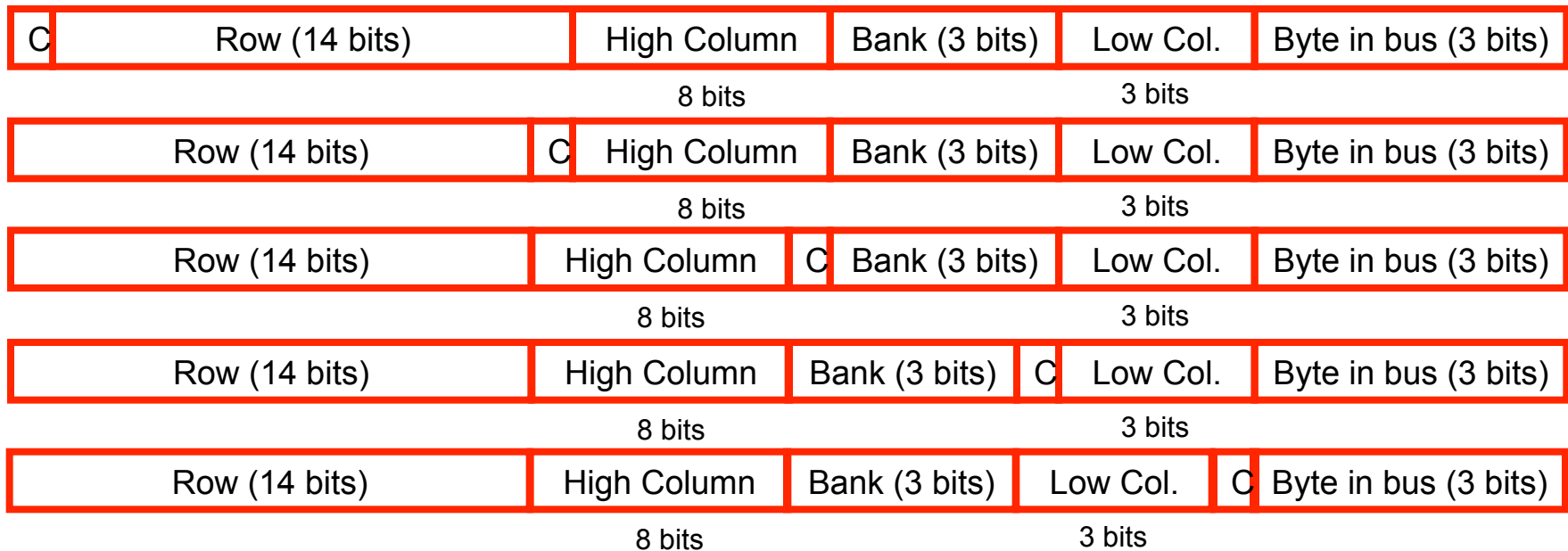
Interaction with Virtual→Physical Mapping

- Operating System influences where an address maps to in DRAM

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

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

Row (14 bits)	Bank (3 bits)	Column (11 bits)	Byte in bus (3 bits)	PA
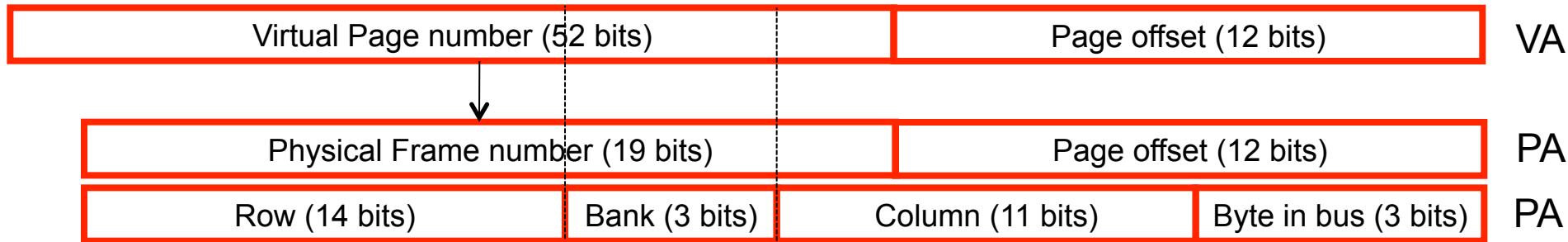
- Operating system can influence which bank/channel/rank a virtual page is mapped to.

- It can perform page coloring to
  - Minimize bank conflicts
  - Minimize inter-application interference **[Muralidhara+ MICRO'11]**

DRAM Refresh (I)

- DRAM capacitor charge leaks over time
- The memory controller needs to read each row periodically to restore the charge
  - Activate + precharge each row every N ms
  - Typical N = 64 ms
- Implications on performance?
- -- DRAM bank unavailable while refreshed
- -- Long pause times: If we refresh all rows in burst, every 64ms the DRAM will be unavailable until refresh ends
- Burst refresh: All rows refreshed immediately after one another
- Distributed refresh: Each row refreshed at a different time, at regular intervals
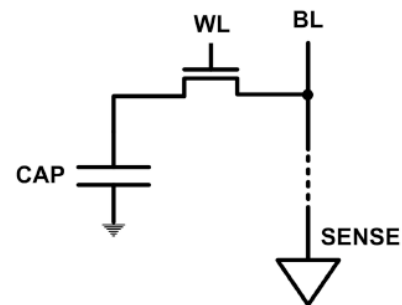
DRAM Refresh (II)

Distributed Refresh

Burst Refresh

Time →

Each pulse represents a refresh cycle

Required time to complete refresh of all rows
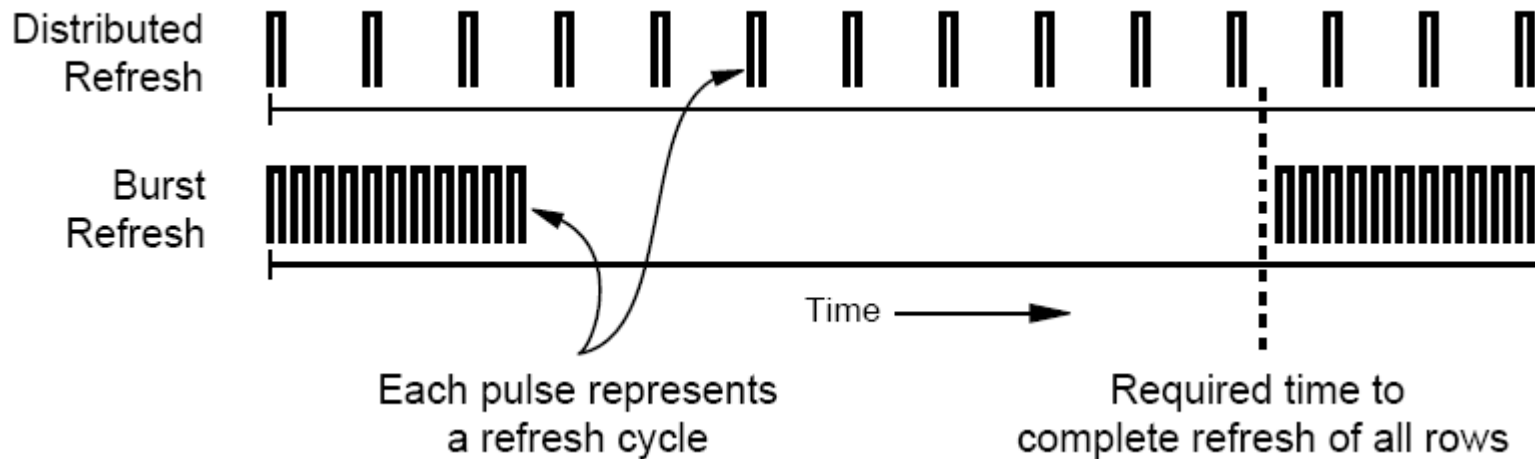
- **Distributed refresh eliminates long pause times**
- How else we can reduce the effect of refresh on performance?
  - Can we reduce the number of refreshes?

Downsides of DRAM Refresh

-- Energy consumption: Each refresh consumes energy

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

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

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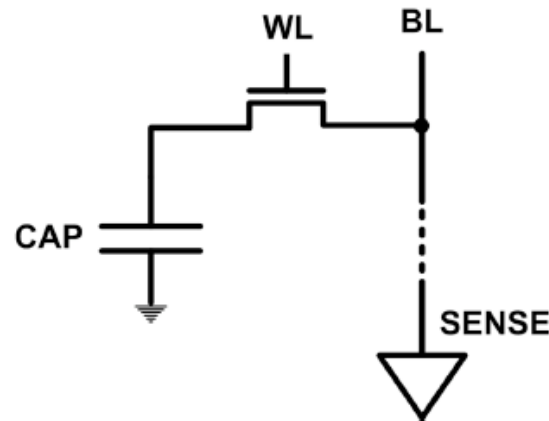
Liu et al., "RAIDR: Retention-aware Intelligent DRAM Refresh," ISCA 2012.

Memory Controllers

DRAM versus Other Types of Memories

- Long latency memories have similar characteristics that need to be controlled.

- The following discussion will use DRAM as an example, but many 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)

A Modern DRAM Controller
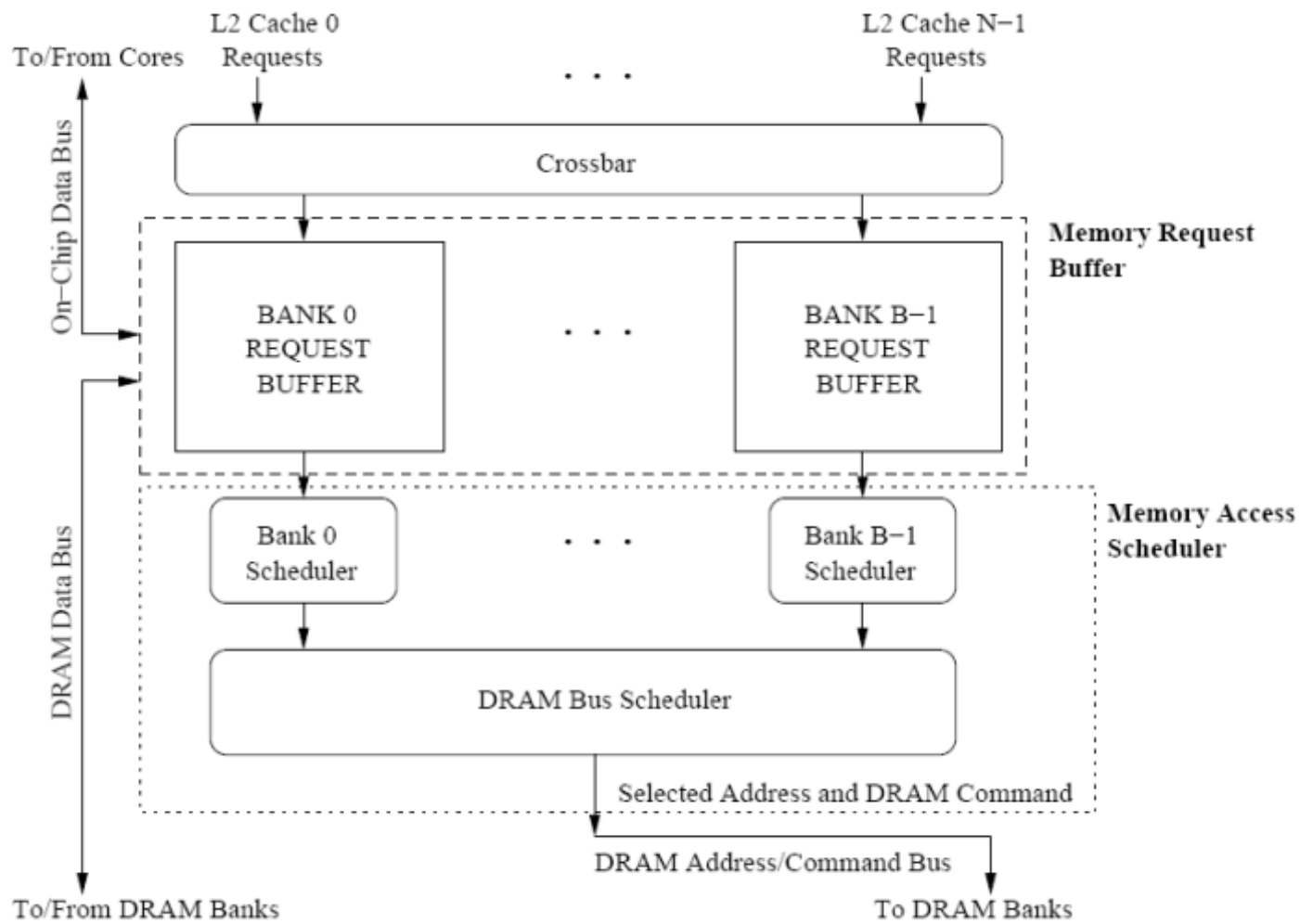
DRAM Scheduling Policies (I)

- **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

DRAM Power Management

- DRAM chips have power modes
- Idea: When not accessing a chip power it down

- Power states
    - Active (highest power)
    - All banks idle
    - Power-down
    - Self-refresh (lowest power)

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

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.
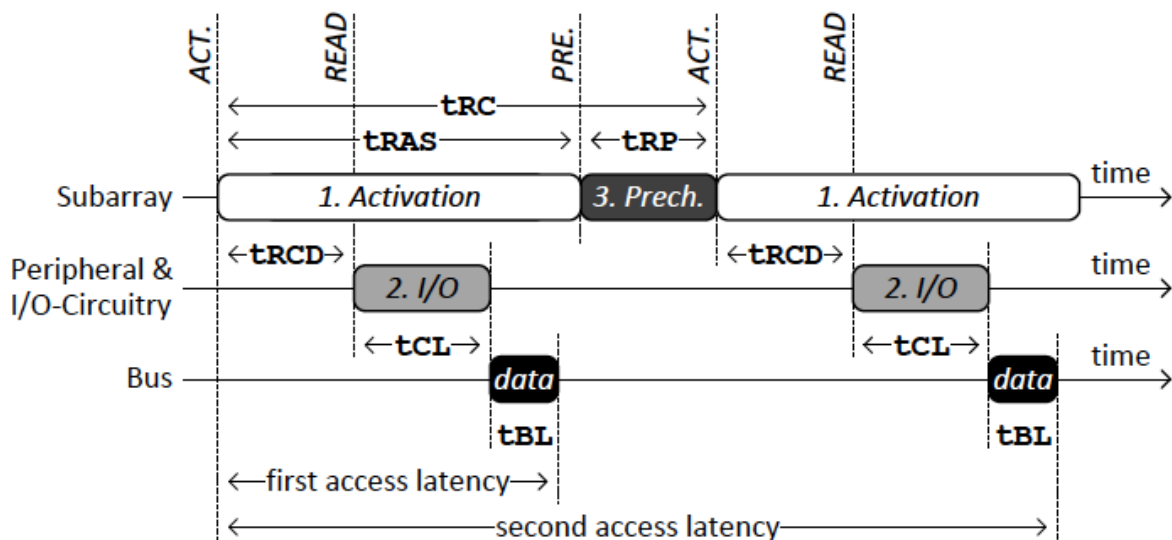

Figure 5. Three Phases of DRAM Access

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.

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

Self-Optimizing DRAM Controllers

Goal: Learn to choose actions to maximize $r_0 + \gamma r_1 + \gamma^2 r_2 + \dots$ ($0 \leq \gamma < 1$)
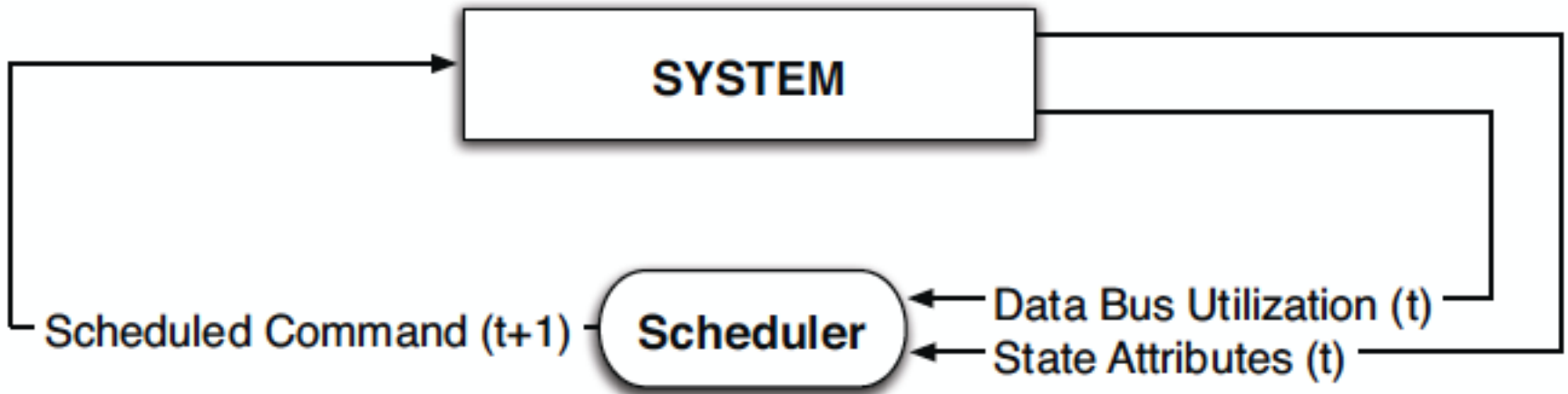
Figure 2: (a) Intelligent agent based on reinforcement learning principles; (b) DRAM scheduler as an RL-agent

Self-Optimizing DRAM Controllers

- Dynamically adapt the memory scheduling policy via interaction with the system at runtime

  - Associate system states and actions (commands) with long term reward values

  - Schedule command with highest estimated long-term value in each state

  - Continuously update state-action values based on feedback from system
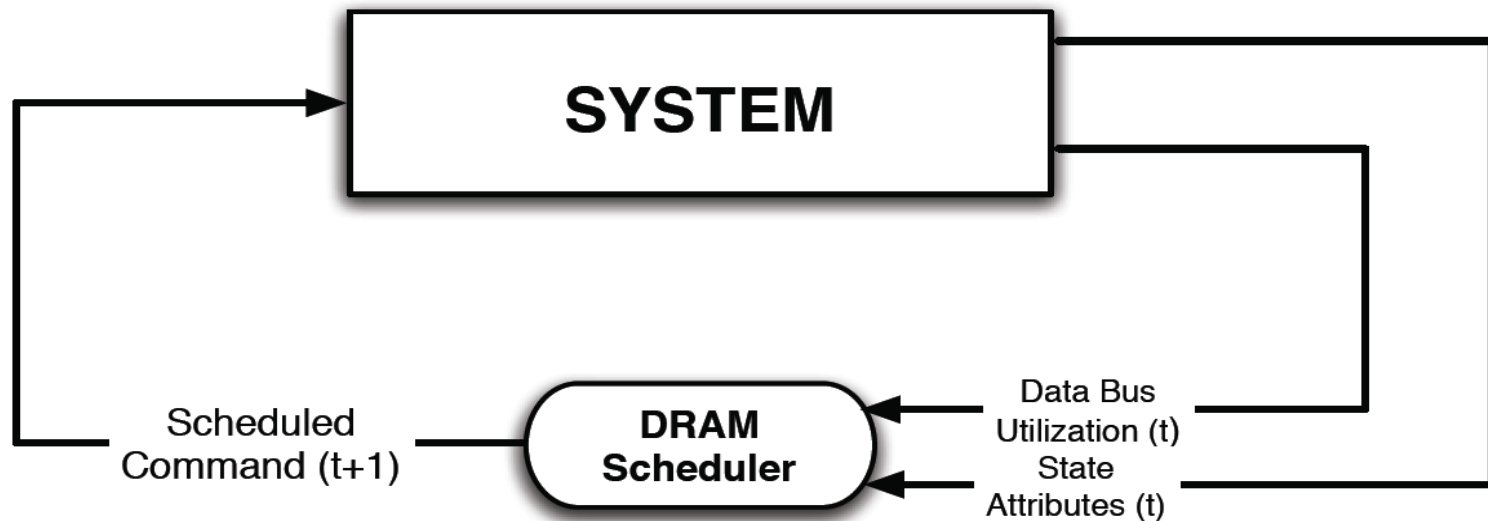
Self-Optimizing DRAM Controllers

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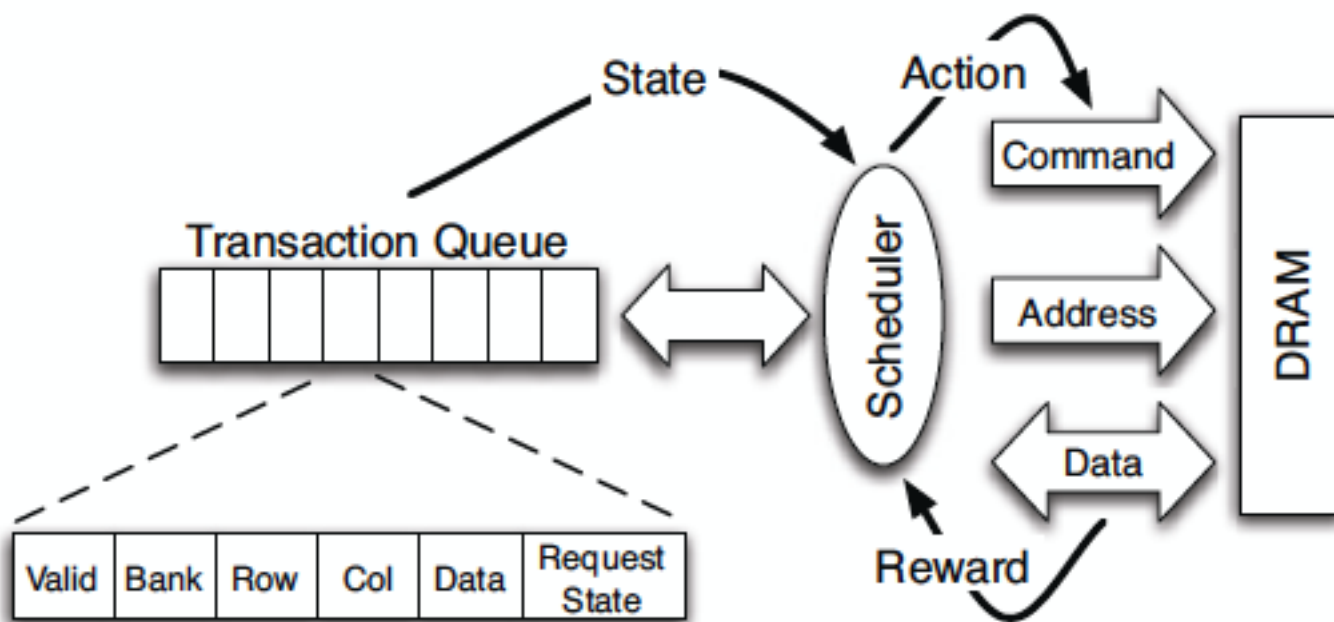

Figure 4: High-level overview of an RL-based scheduler.

States, Actions, Rewards

❖ **Reward function**

- +1 for scheduling Read and Write commands

- 0 at all other times

❖ **State attributes**

- Number of reads, writes, and load misses in transaction queue

- Number of pending writes and ROB heads waiting for referenced row

- Request's relative ROB order

❖ **Actions**

- Activate

- Write

- Read - load miss

- Read - store miss

- Precharge - pending

- Precharge - preemptive
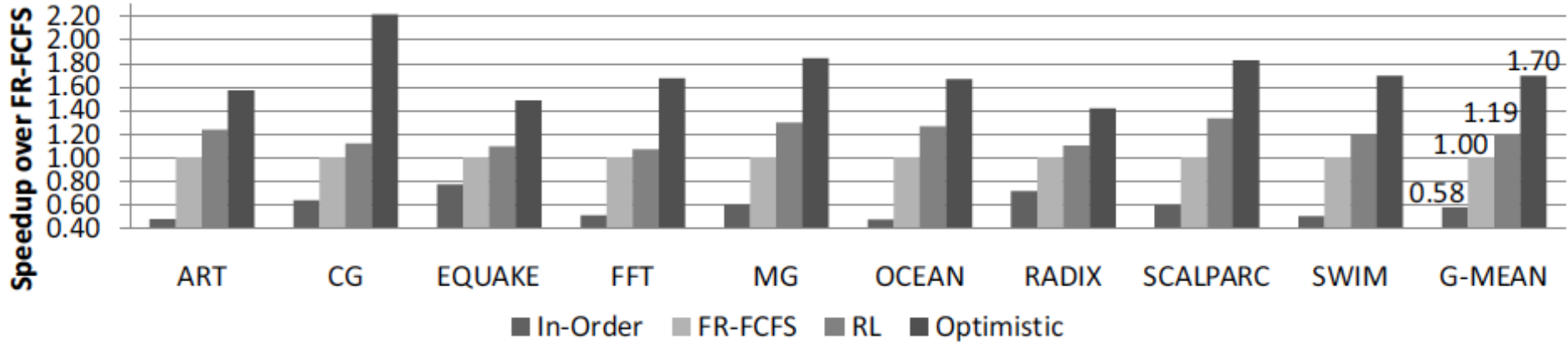
- NOP

Performance Results

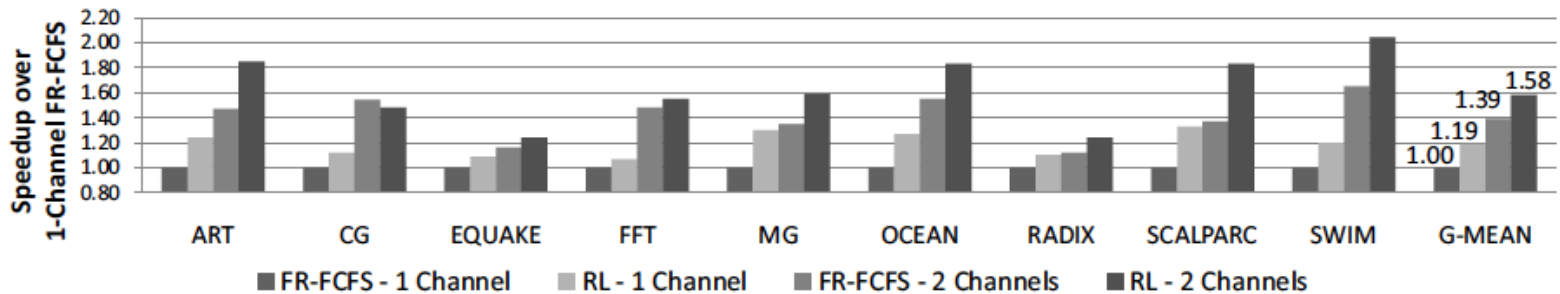Figure 7: Performance comparison of in-order, FR-FCFS, RL-based, and optimistic memory controllers

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

Self Optimizing DRAM Controllers

- Advantages

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

  \+ Reduces the designer's burden in finding a good scheduling policy. Designer specifies:

    1) What system variables might be useful

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

- Disadvantages

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

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

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

SAFARI

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

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

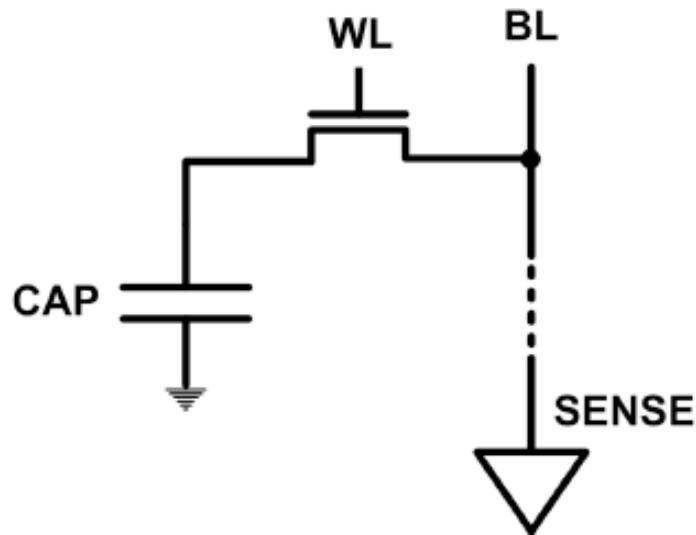**SAFARI**

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

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.
- Seshadri+, "RowClone: Fast and Efficient In-DRAM Copy and Initialization of Bulk Data," 2013.

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

Columns

BANK 0 BANK 1 BANK 2 BANK 3

Rows

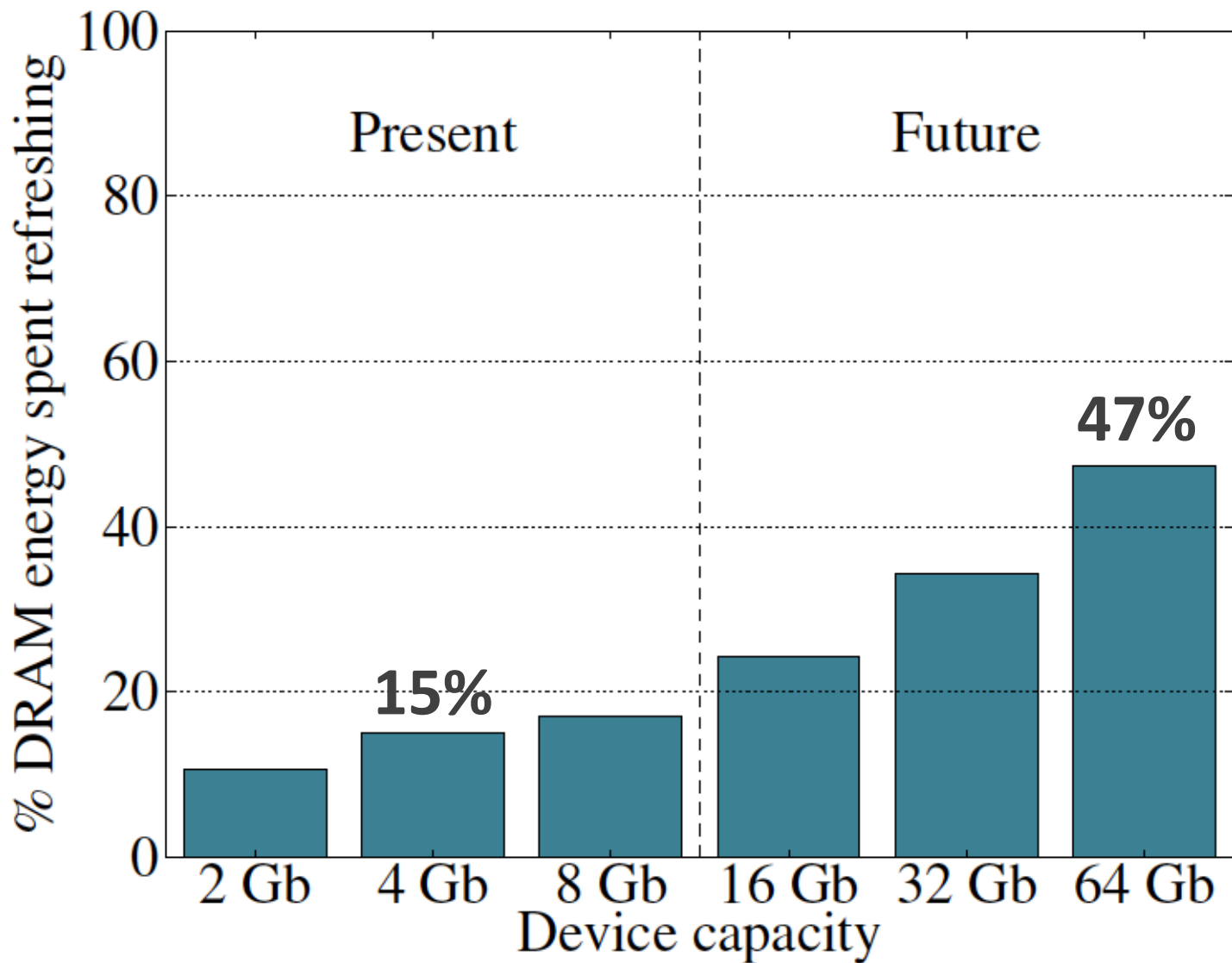Row Buffer

DRAM Bus

DRAM CONTROLLER
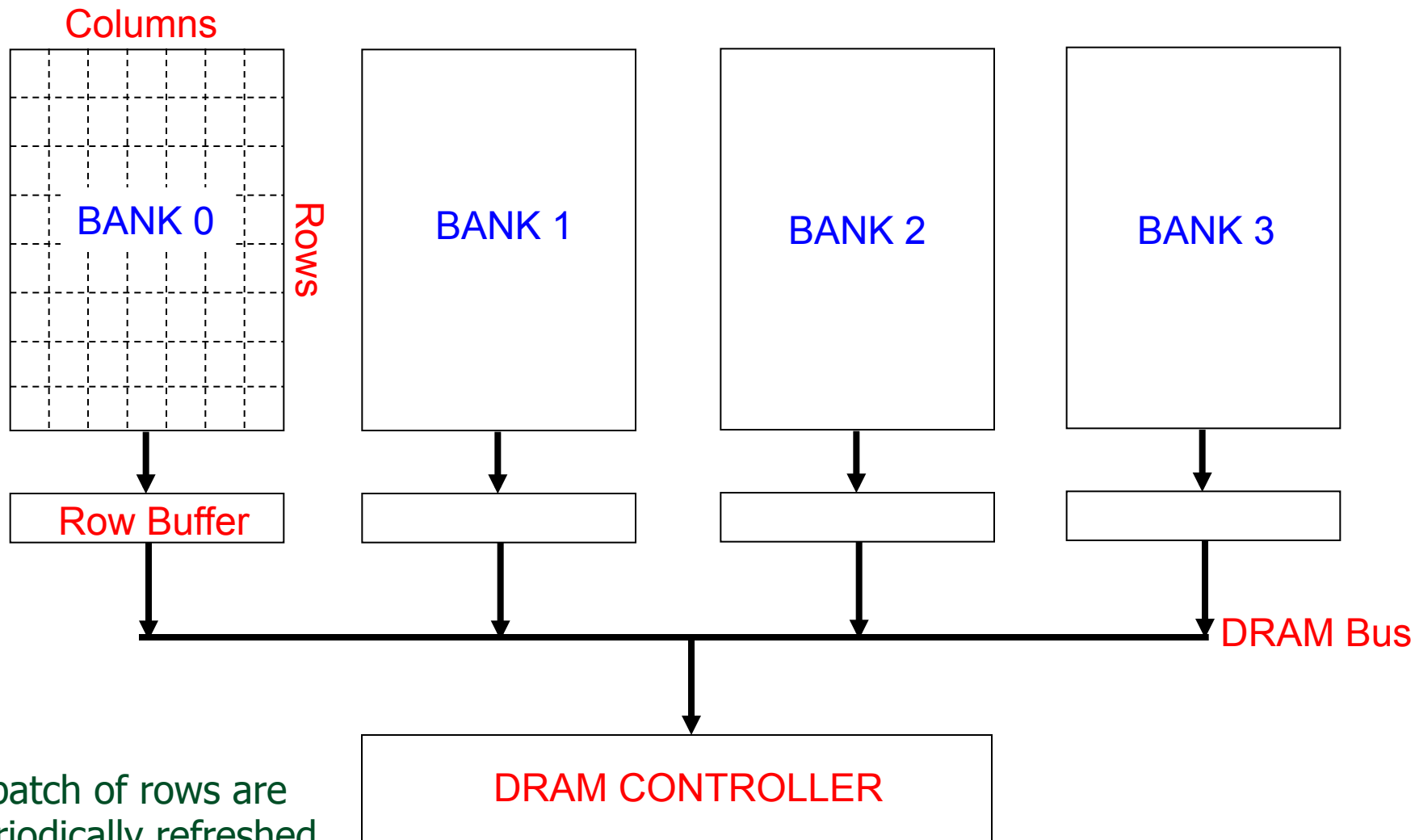
A batch of rows are periodically refreshed via the auto-refresh command
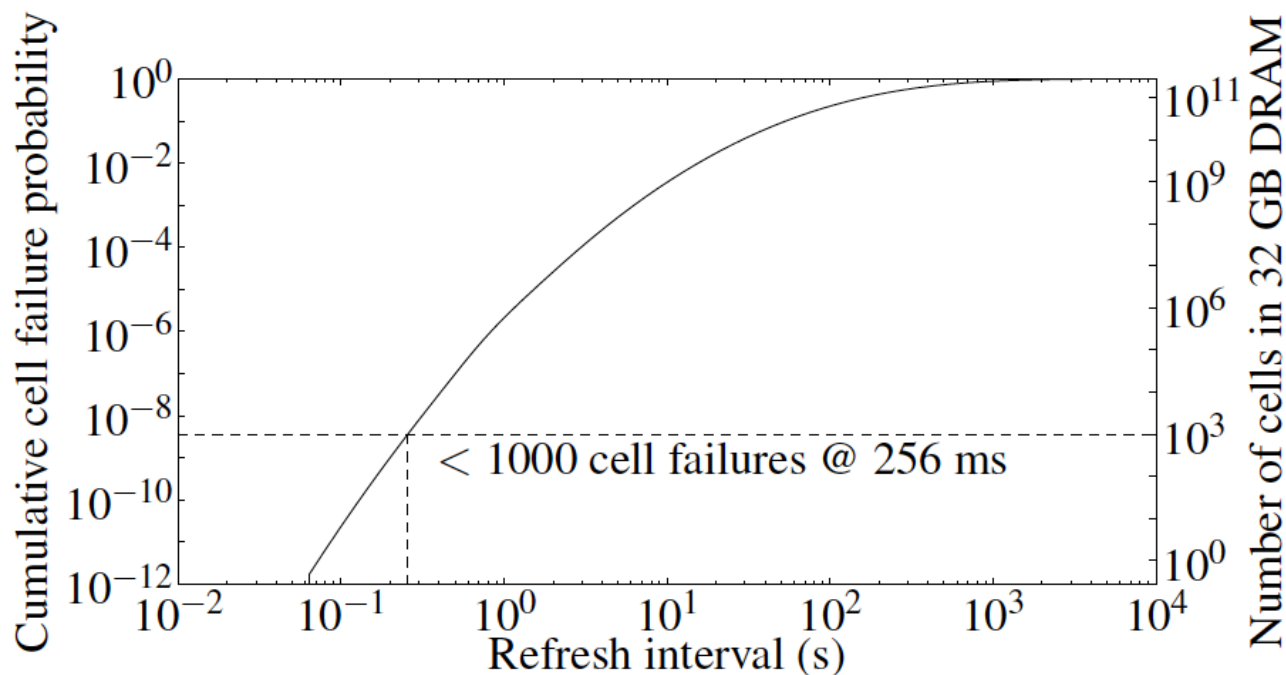
Refresh Overhead: Performance
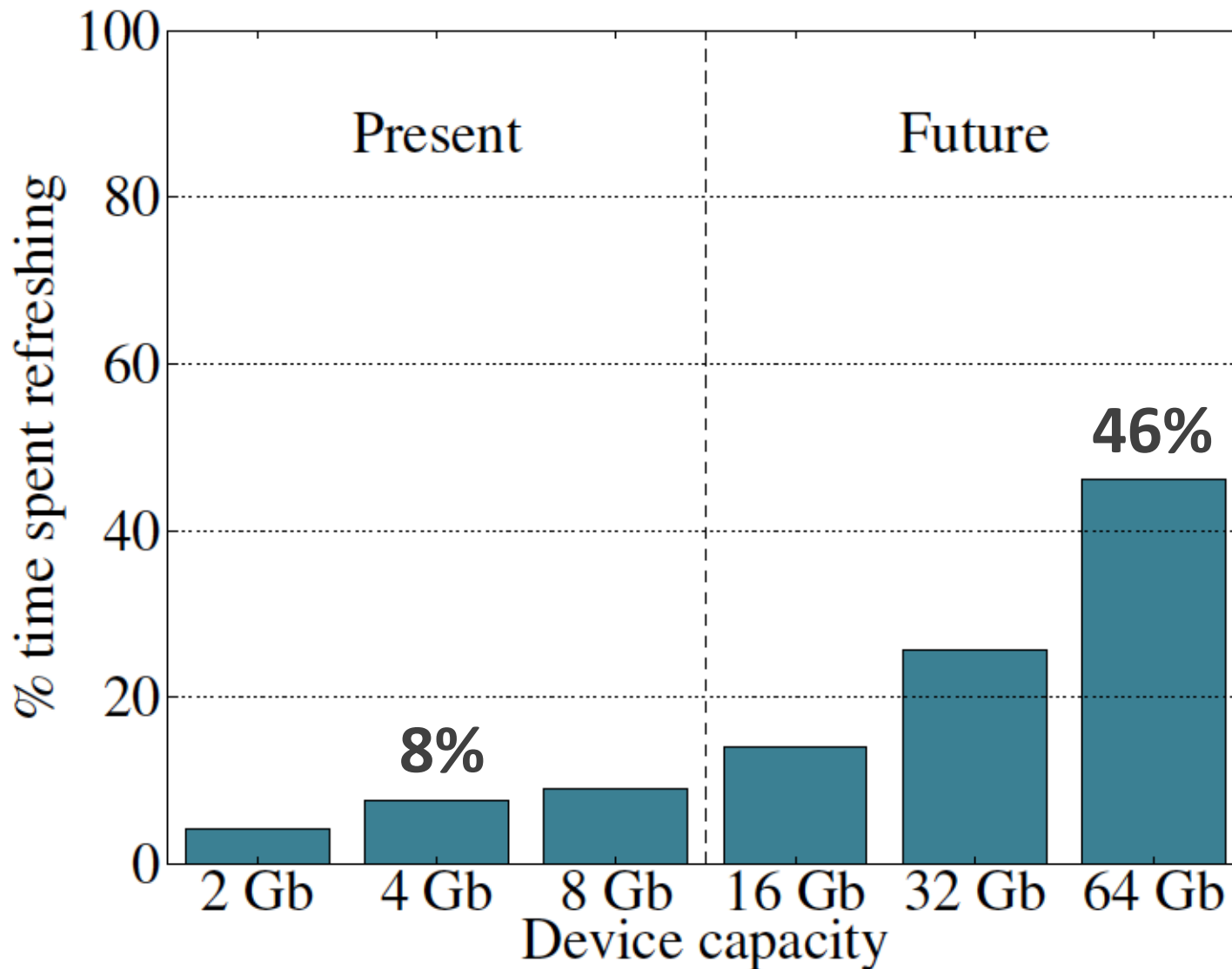
# Refresh Overhead: Energy

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

SAFARI

Retention Time of DRAM Rows

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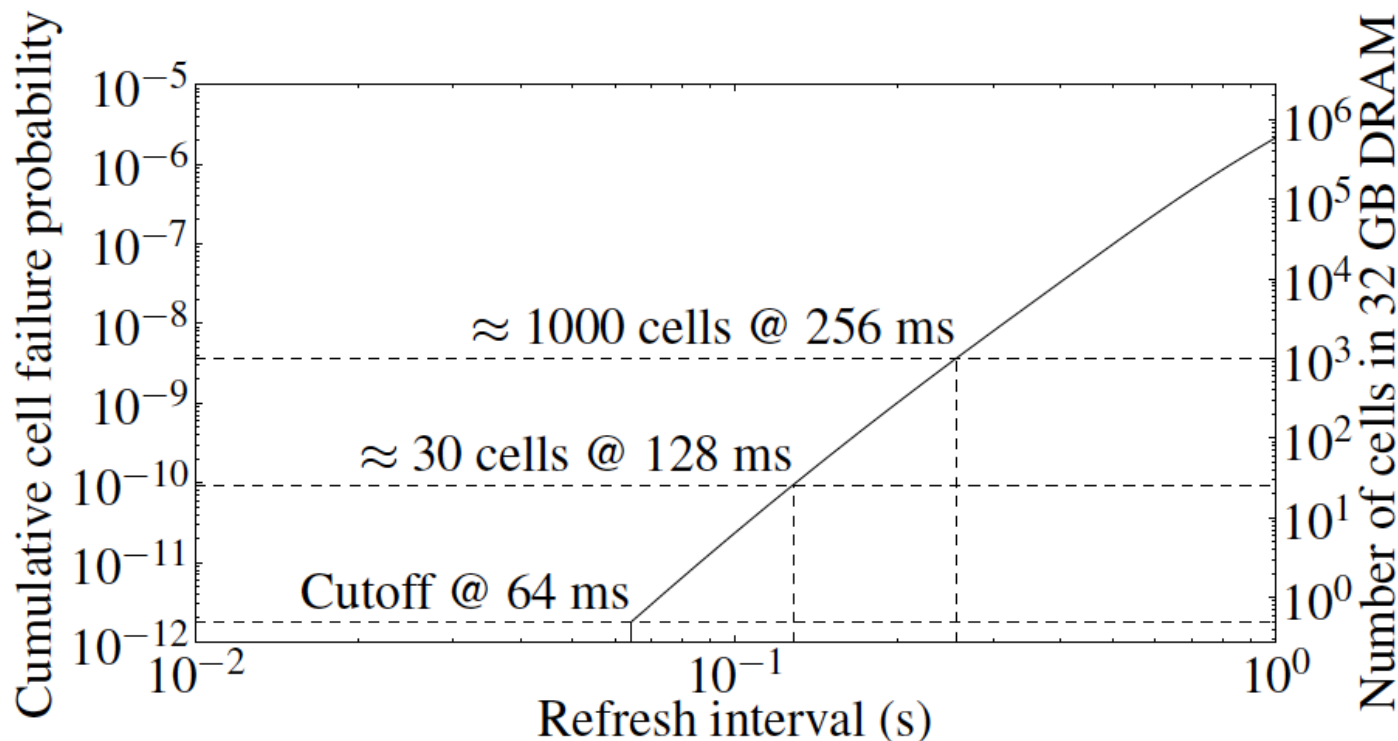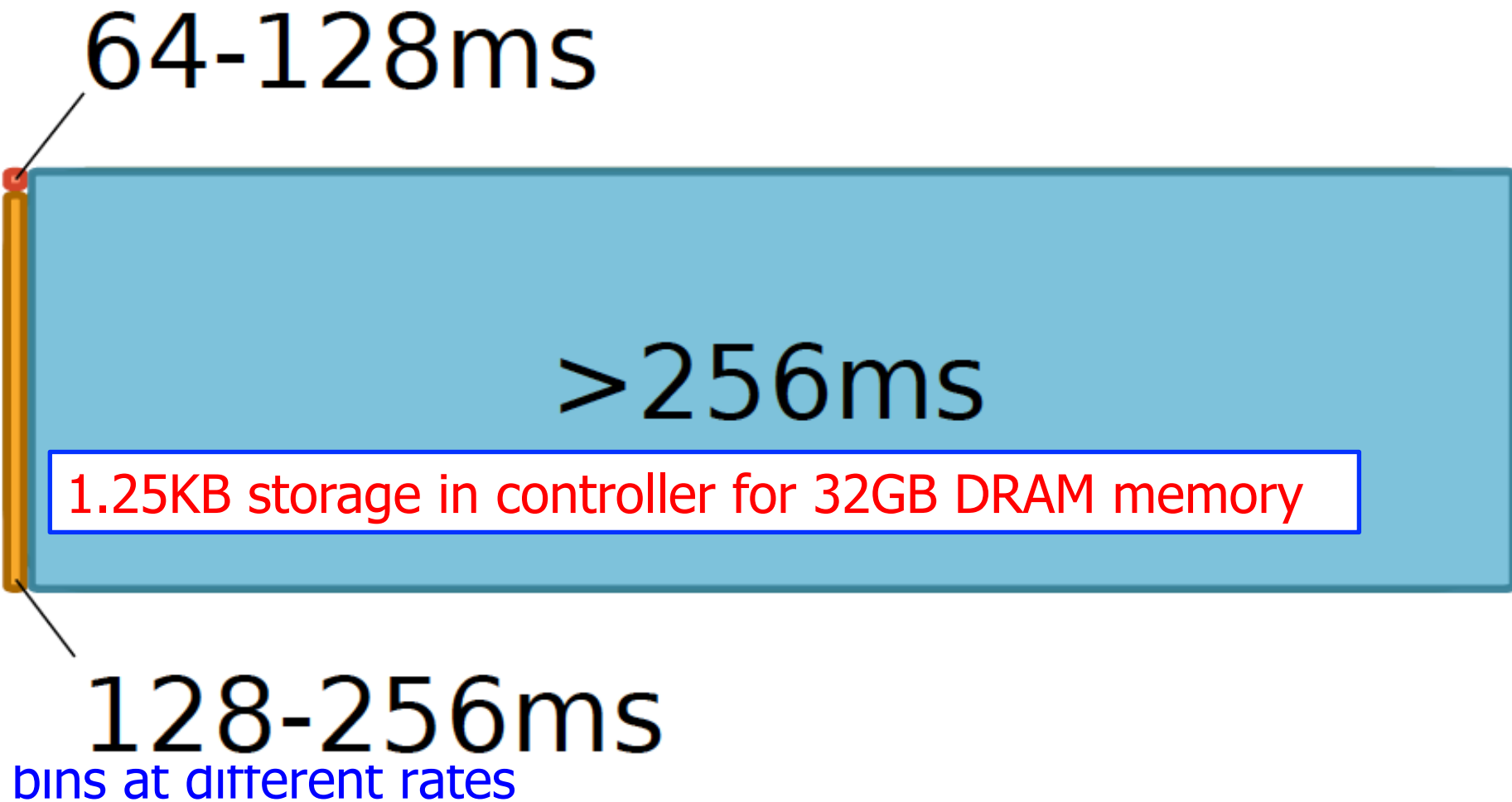

- Can we exploit this to reduce refresh operations at low cost?

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.

SAFARI

RAIDR: Mechanism

64-128ms

> 256ms

1.25KB storage in controller for 32GB DRAM memory

128-256ms

bins at different rates

→ probe Bloom Filters to determine refresh rate of a row
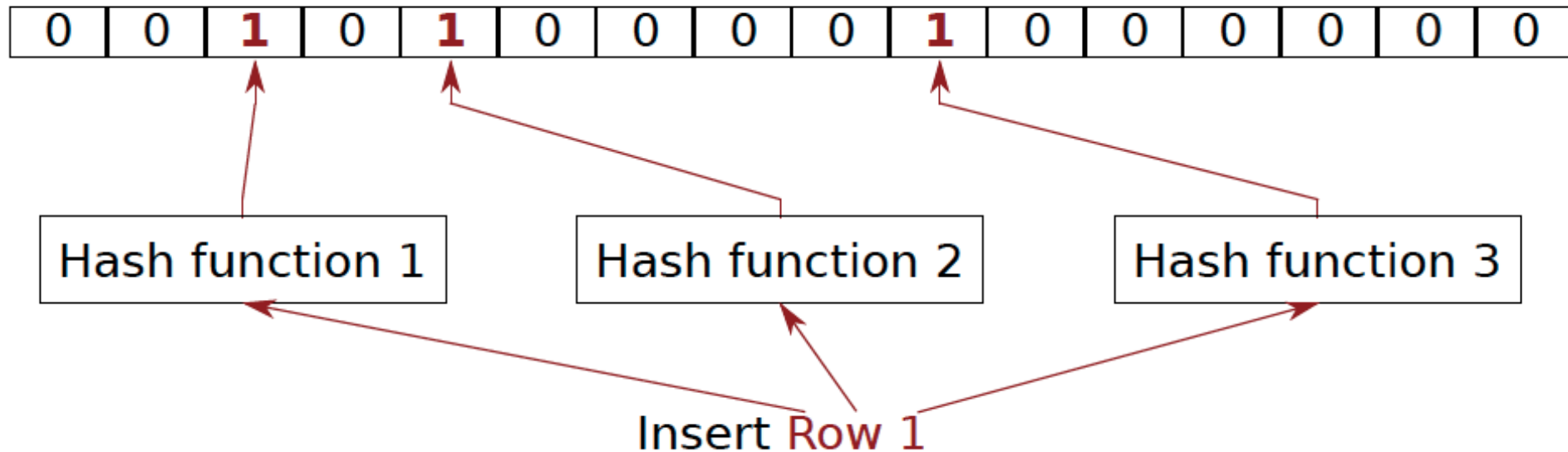
1. Profiling

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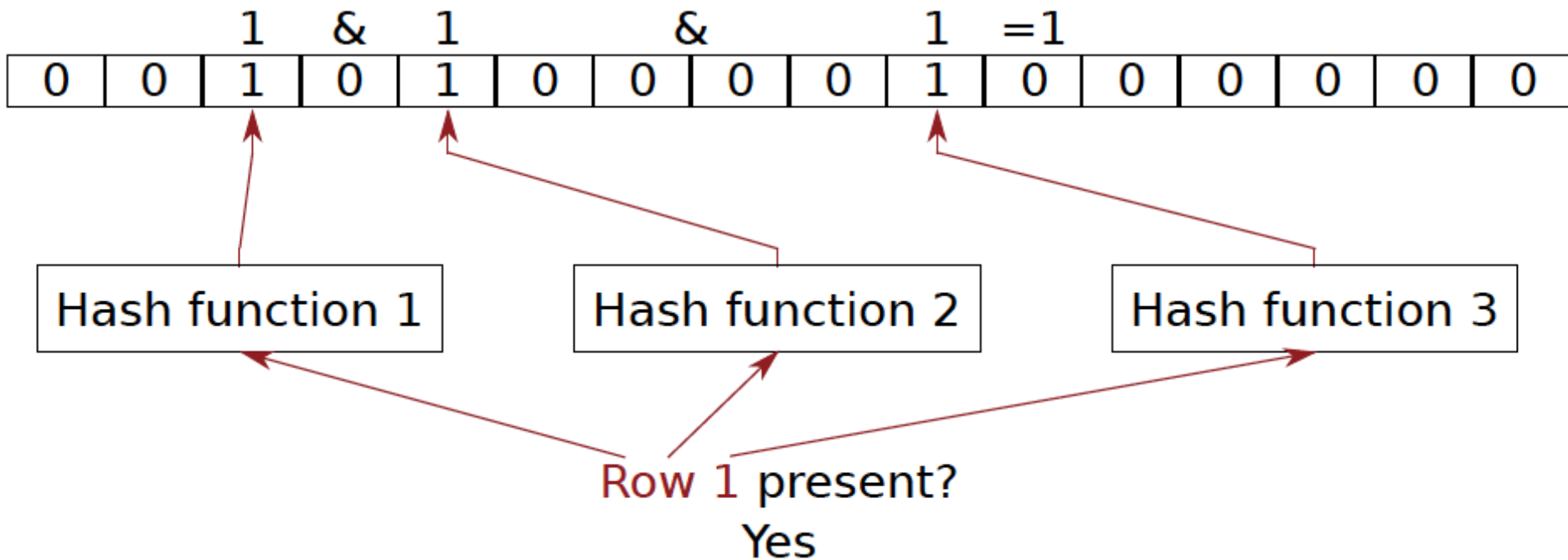
- How to efficiently and scalably store rows into retention time bins?
- Use Hardware Bloom Filters [Bloom, CACM 1970]

Example with 64–128ms bin:

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

Insert Row 1

SAFARI

Bloom Filter Operation Example

Example with 64–128ms bin:

1 & 1 & 1 $=1$

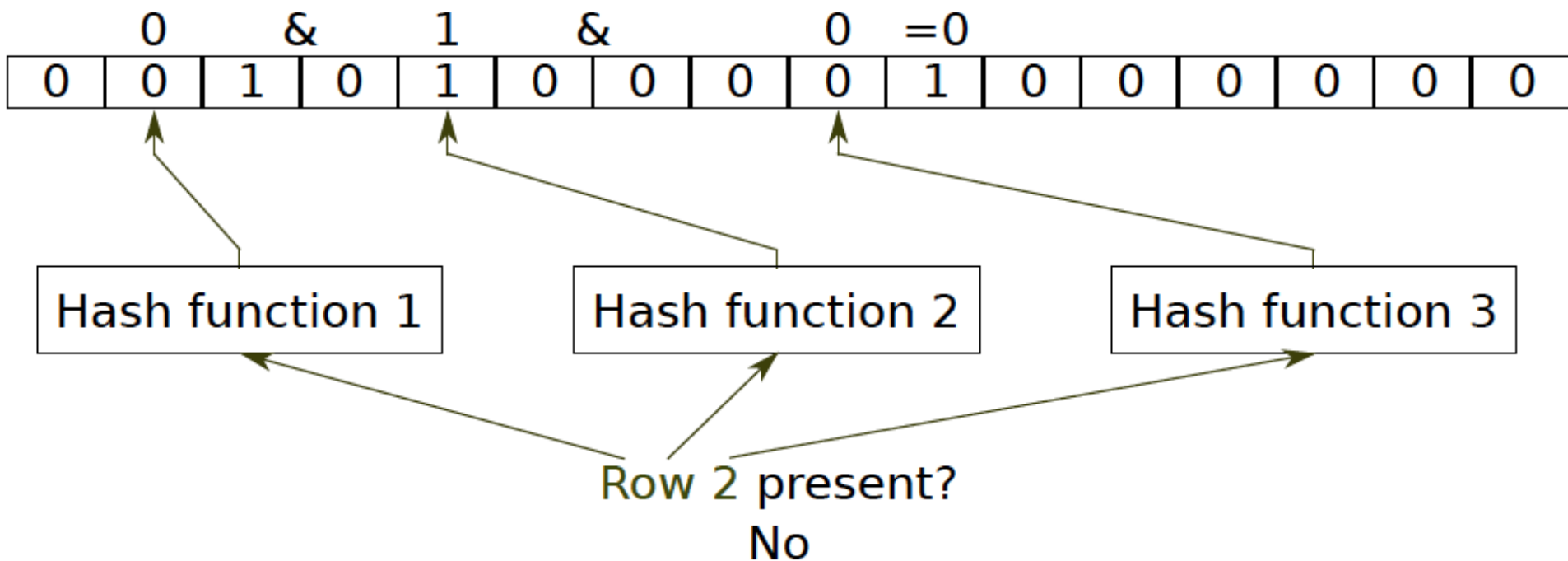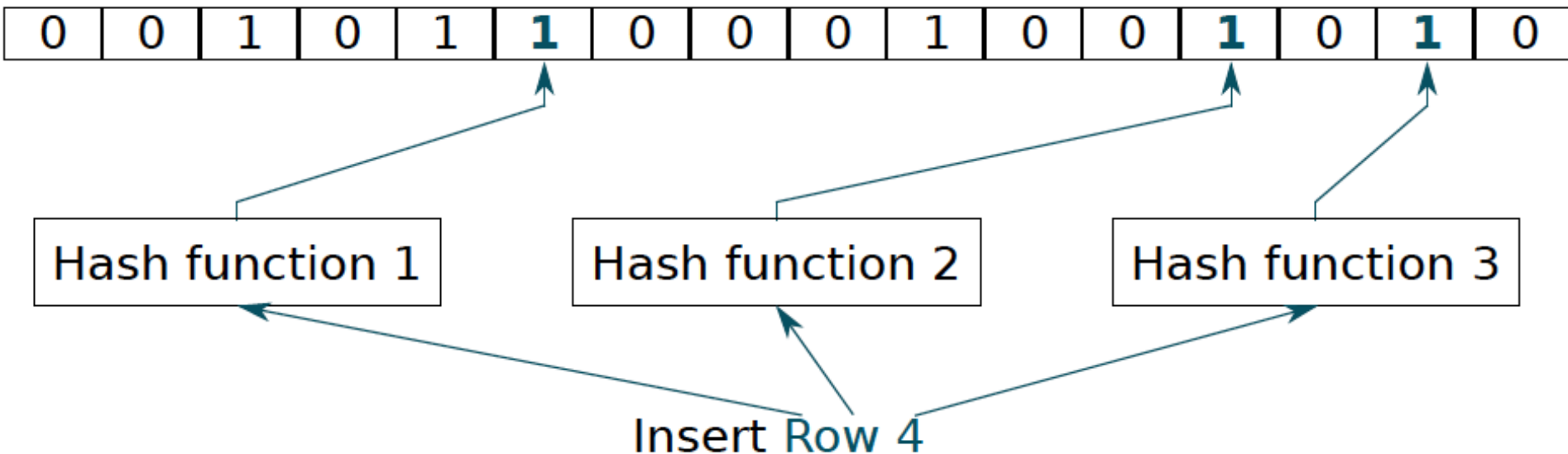| 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |

Hash function 1   Hash function 2   Hash function 3

Row 1 present?
Yes

Bloom Filter Operation Example

Example with 64–128ms bin:

Bloom Filter Operation Example

Example with 64–128ms bin:

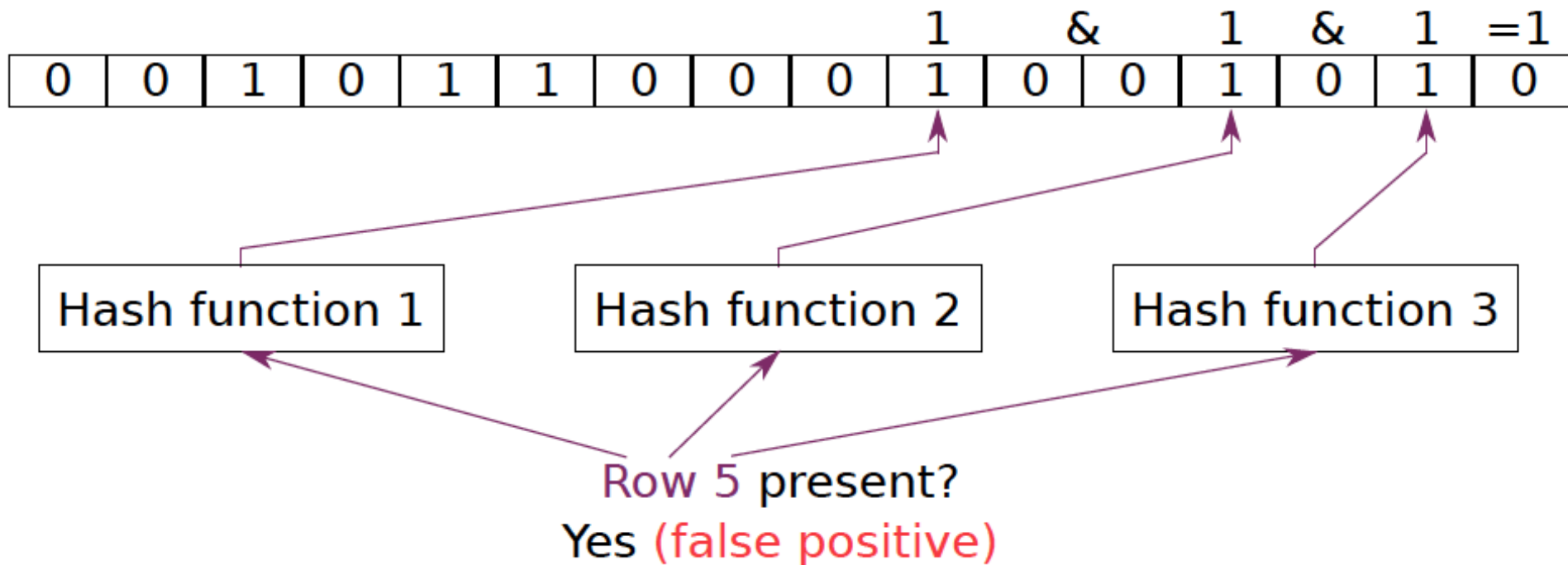| 0	0	1	0	1	**1**	0	0	0	1	0	0	**1**	0	**1**	0

Hash function 1   Hash function 2   Hash function 3

Insert Row 4

Bloom Filter Operation Example

Example with 64–128ms bin:

$$1 \quad \& \quad 1 \quad \& \quad 1 \quad =1$$

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

Hash function 1	Hash function 2	Hash function 3

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
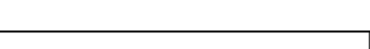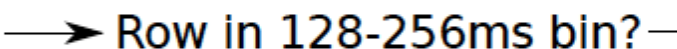
↓

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

↓   ↓   ↓

Refresh the row   Every other 64ms window,   Every 4th 64ms window,
   refresh the row   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 $\Rightarrow$ longer period $\Rightarrow$ less frequent refreshes
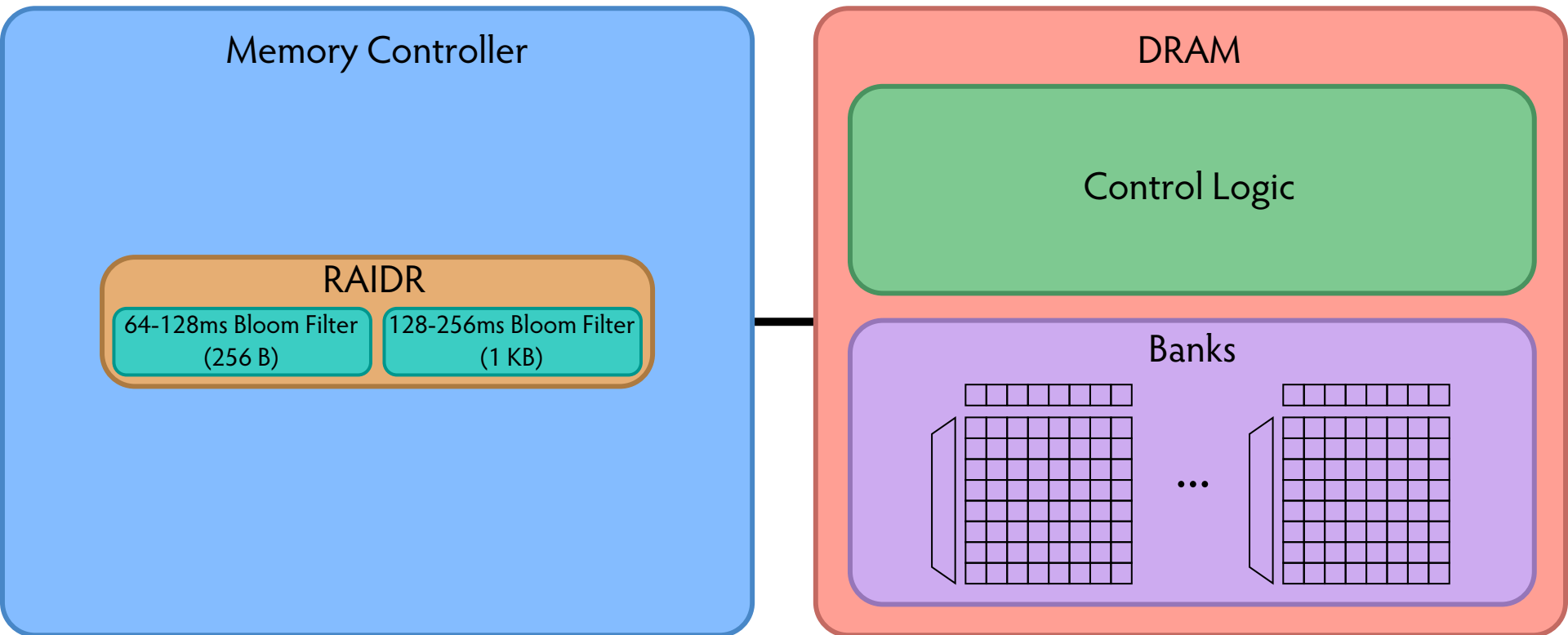 ► Higher temperature $\Rightarrow$ shorter period $\Rightarrow$ more frequent refreshes

SAFARI

RAIDR: Baseline Design

Memory Controller

DRAM

Control Logic

Banks

...
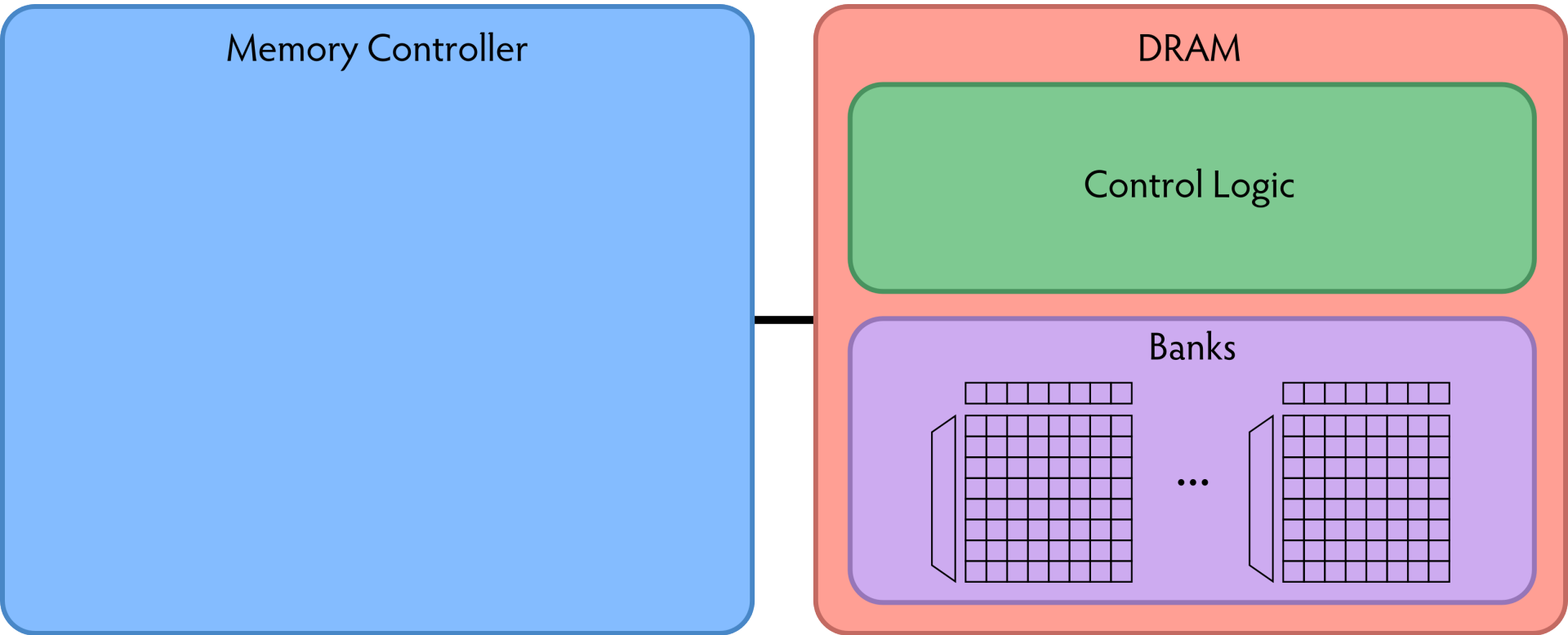
Refresh control is in DRAM in today's auto-refresh systems

RAIDR can be implemented in either the controller or DRAM

SAFARI

RAIDR in Memory Controller: Option 1

Memory Controller

RAIDR

64-128ms Bloom Filter (256 B)

128-256ms Bloom Filter (1 KB)

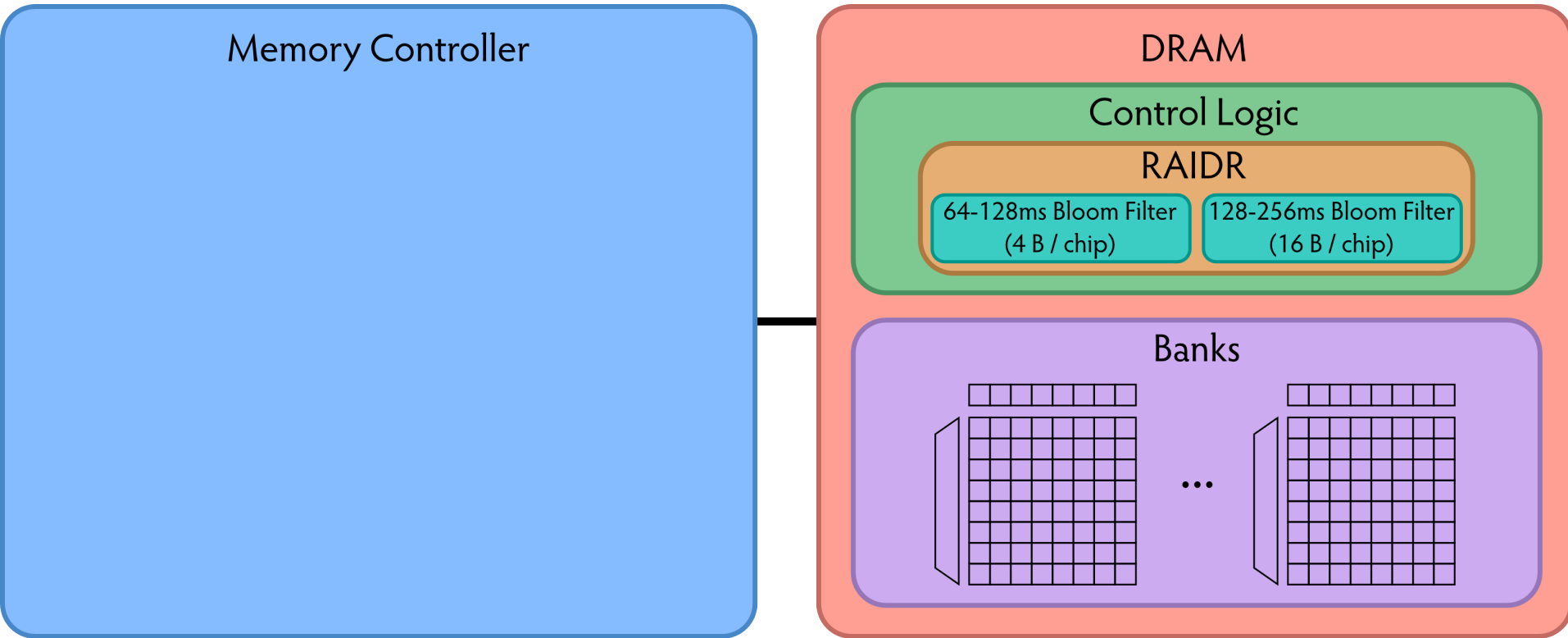**DRAM**

Control Logic

Banks

...

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

Memory Controller

DRAM

Control Logic

RAIDR

64-128ms Bloom Filter (4 B / chip) | 128-256ms Bloom Filter (16 B / chip)

Banks

...

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)

SAFARI

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

SAFARI

RAIDR Refresh Reduction
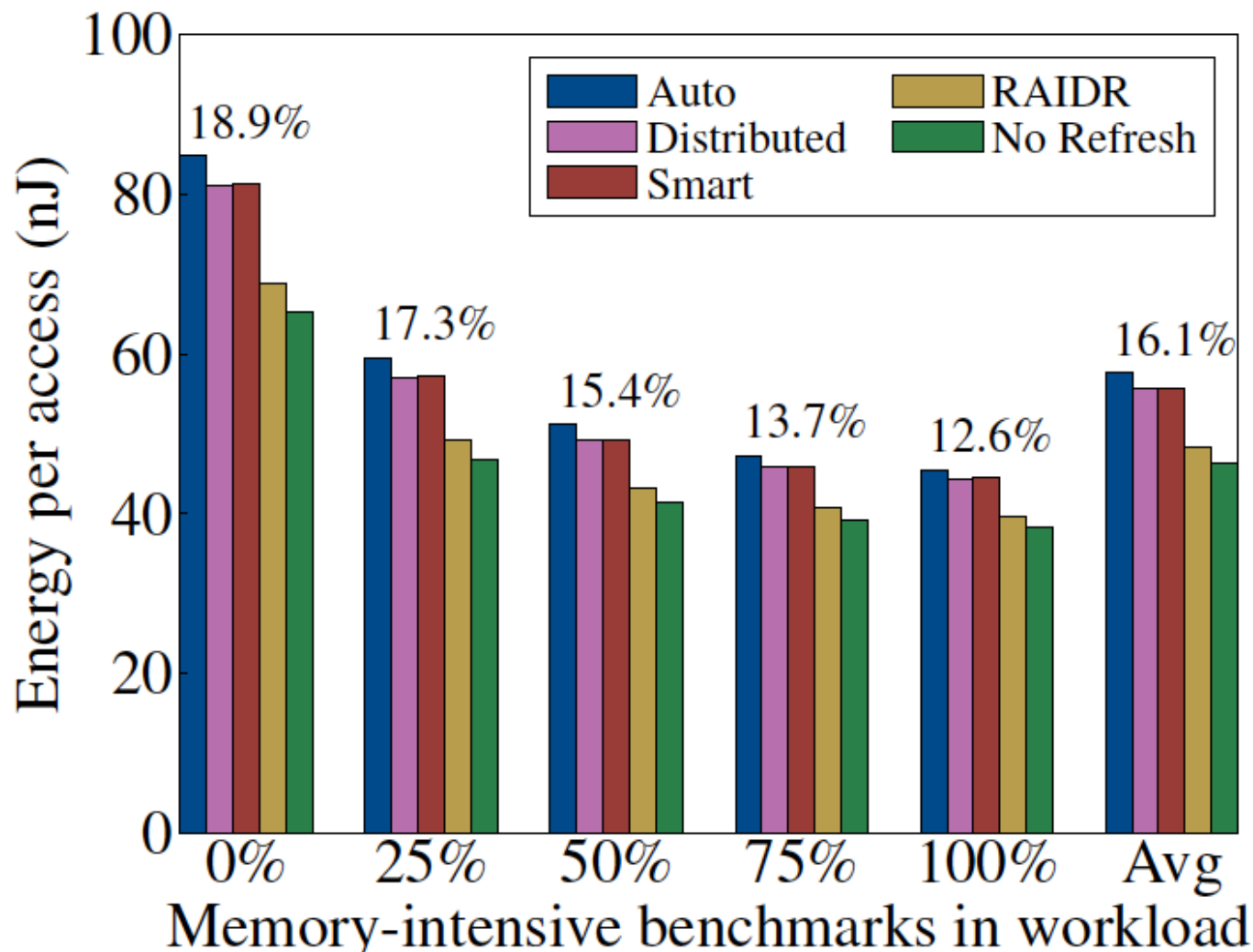
32 GB DDR3 DRAM system
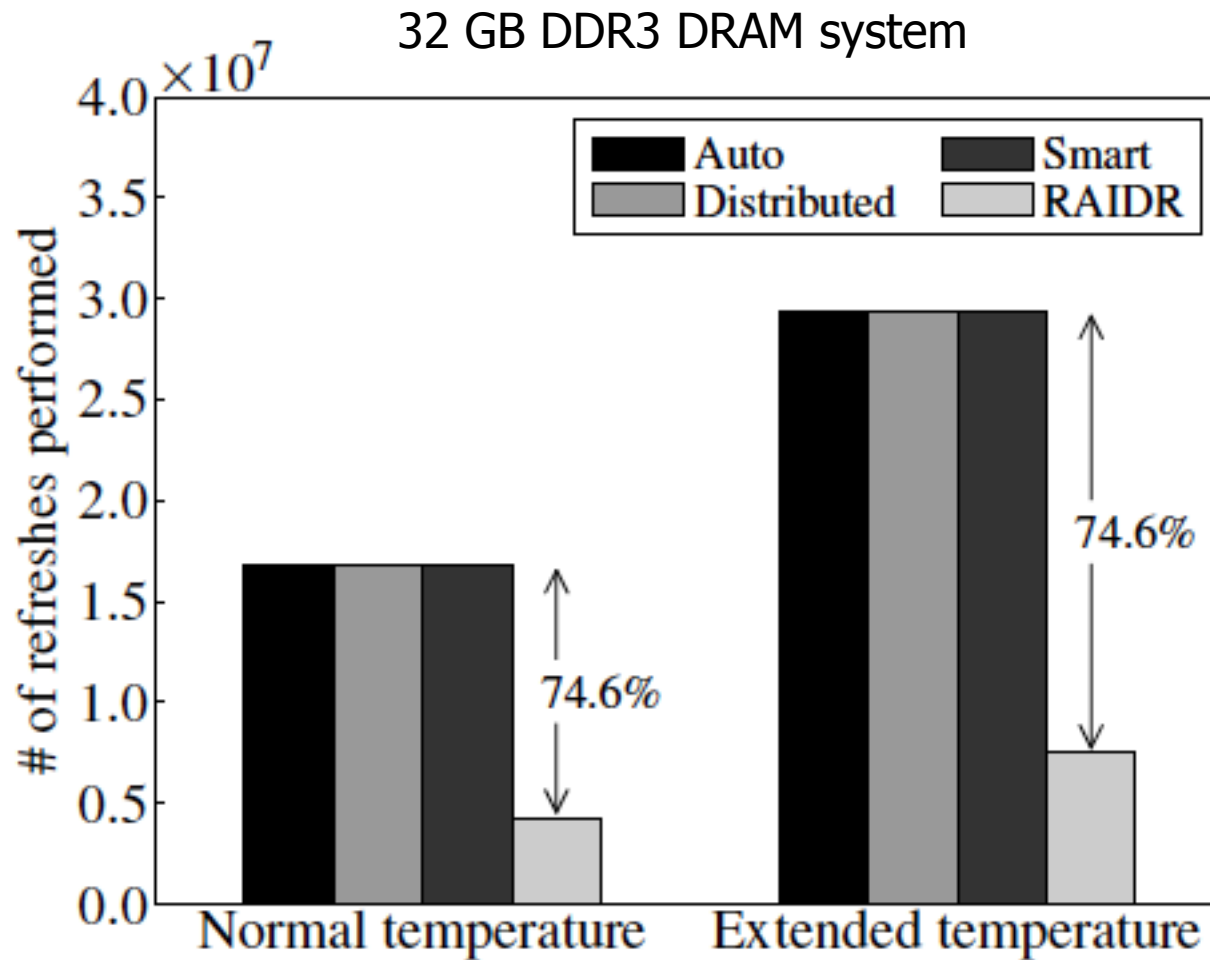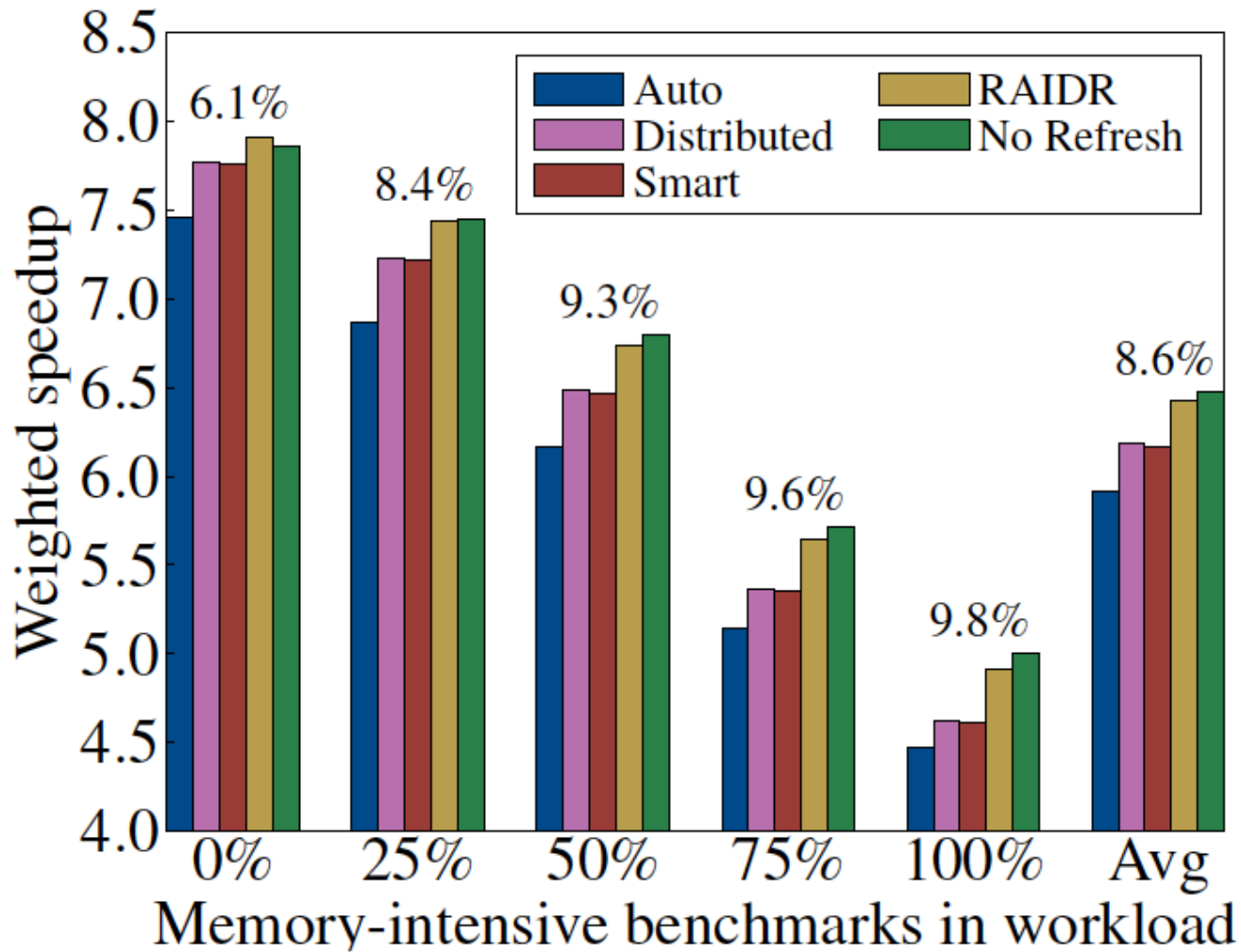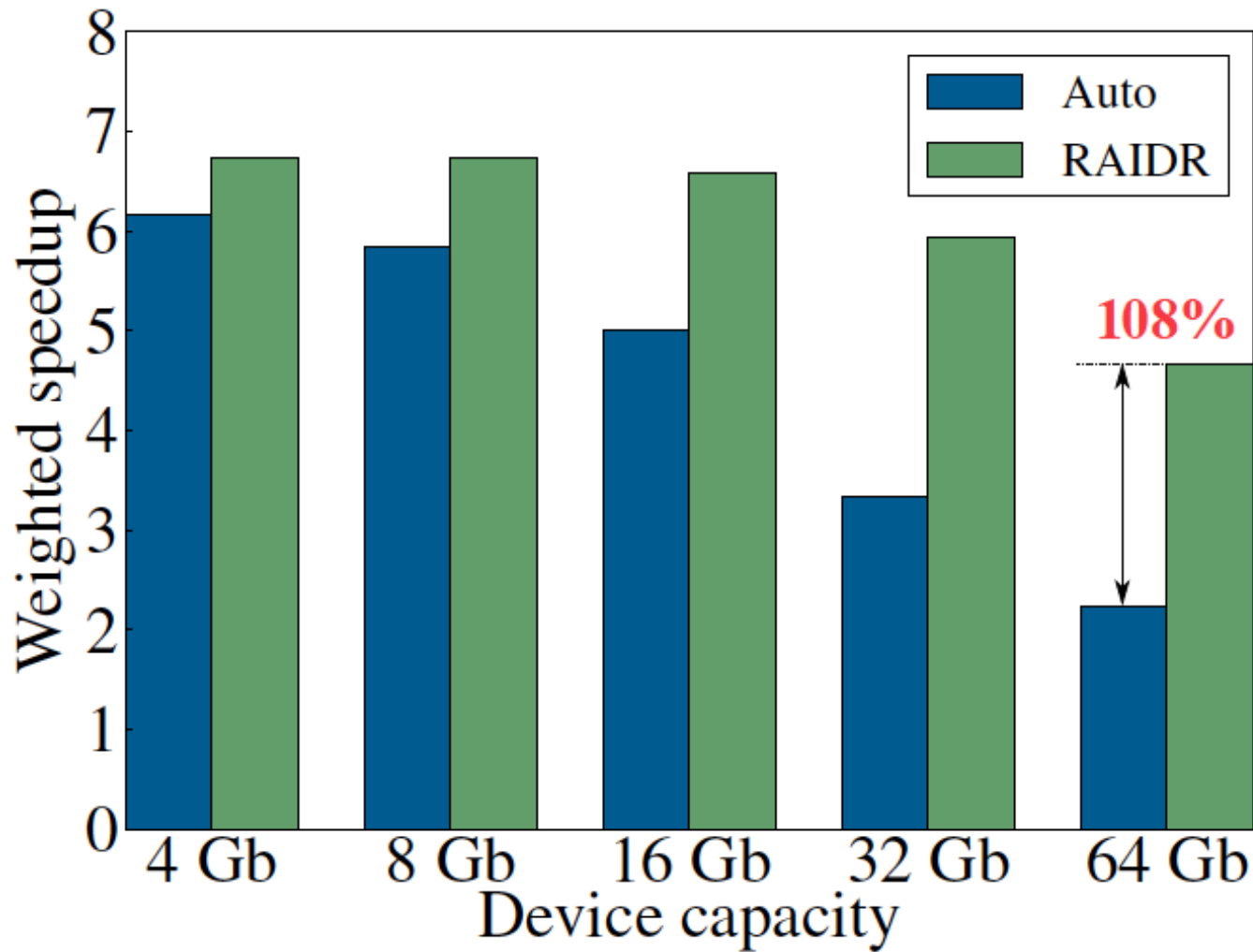
RAIDR: Performance

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
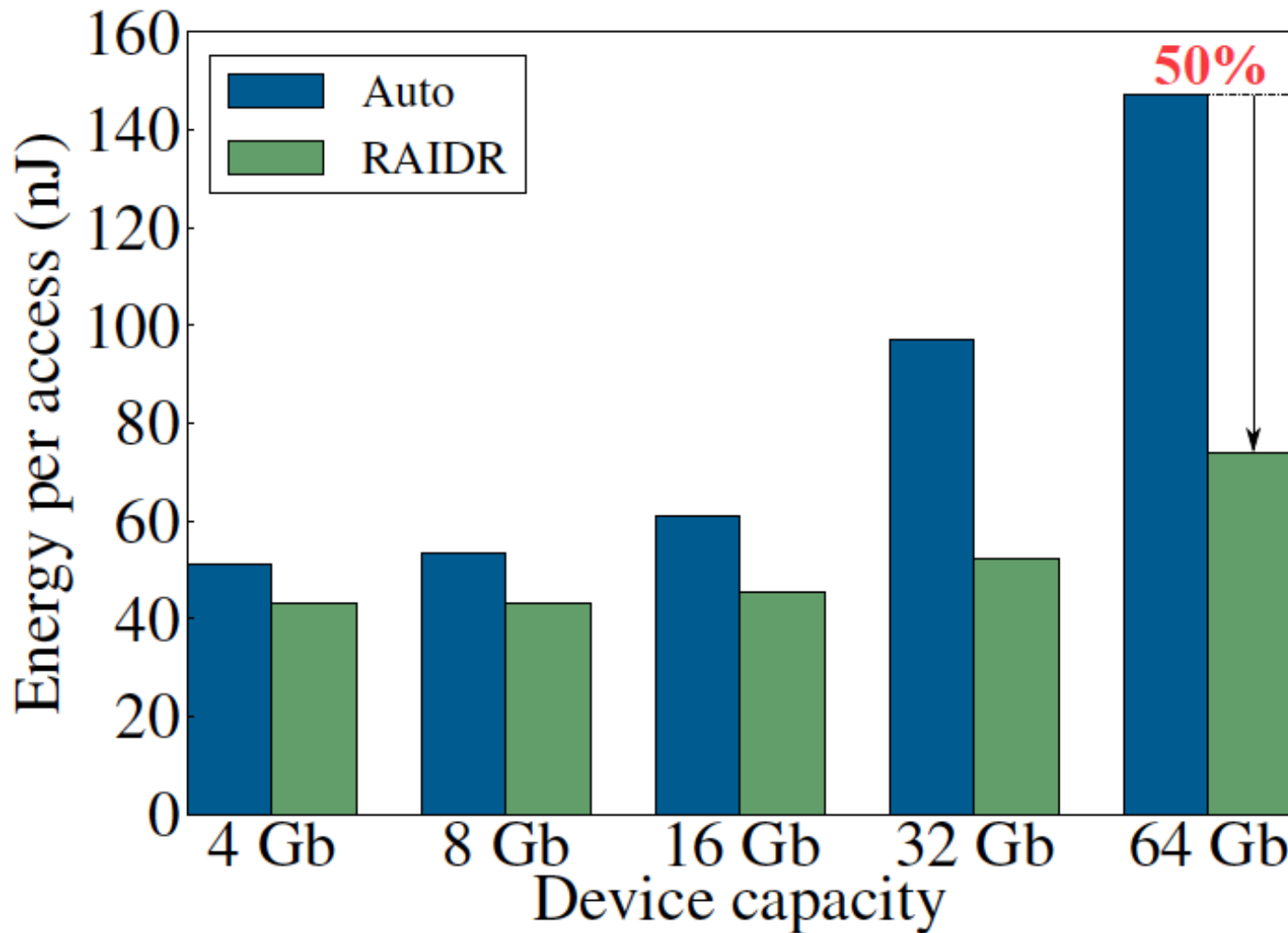
RAIDR energy benefits increase with DRAM chip capacity

SAFARI

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

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

Historical DRAM Latency-Capacity Trend
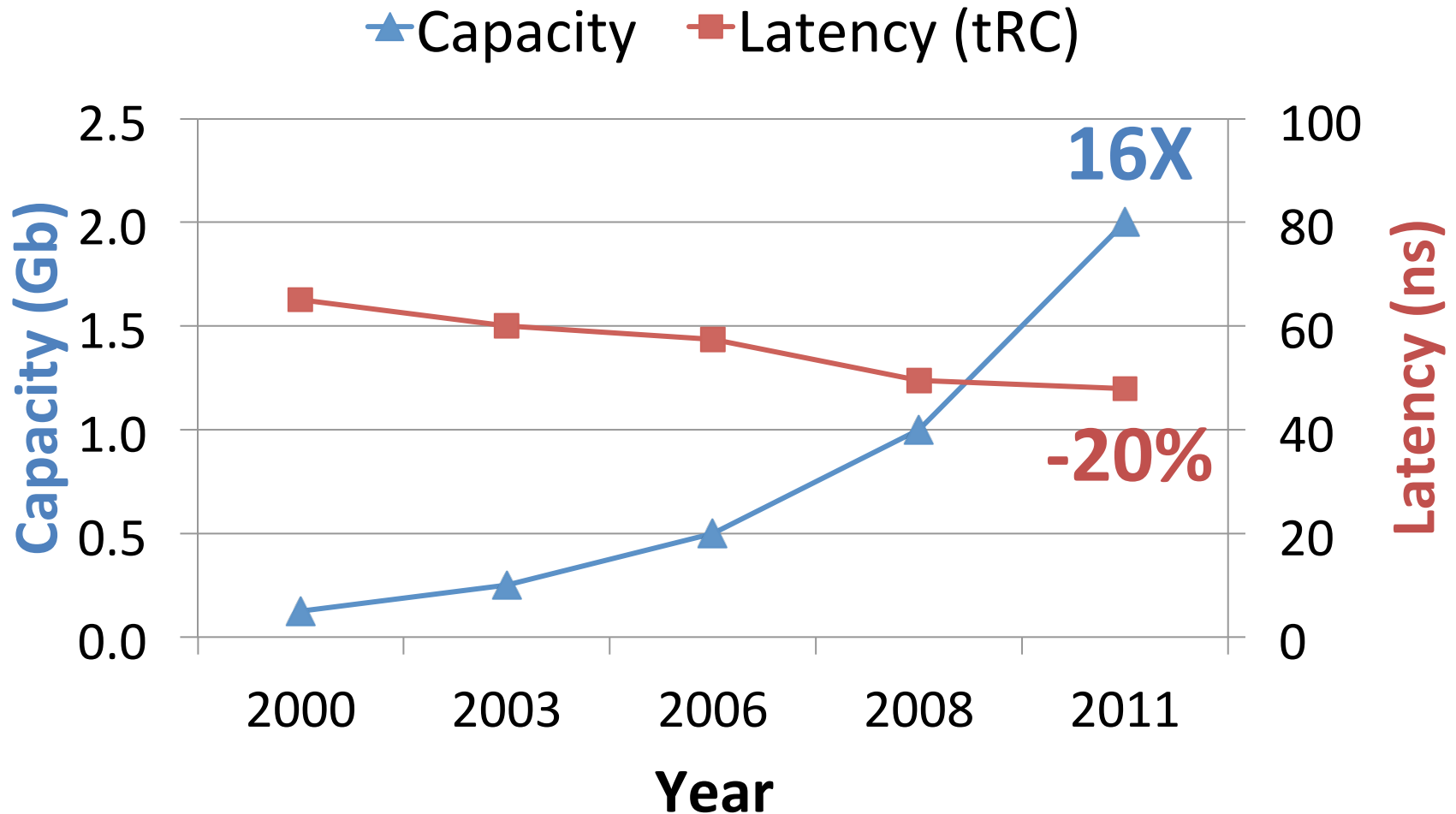
DRAM latency continues to be a critical bottleneck

What Causes the Long Latency?

DRAM Chip

subarray

subarray

I/O

channel

cell

row decoder

wordline

capacitor

access transistor

bitline

sense amplifier

151

What Causes the Long Latency?

DRAM Chip

row decoder

subarray

sense amplifier

subarray

row addr.

Subarray

I/O

I/O

column addr.

mux

channel
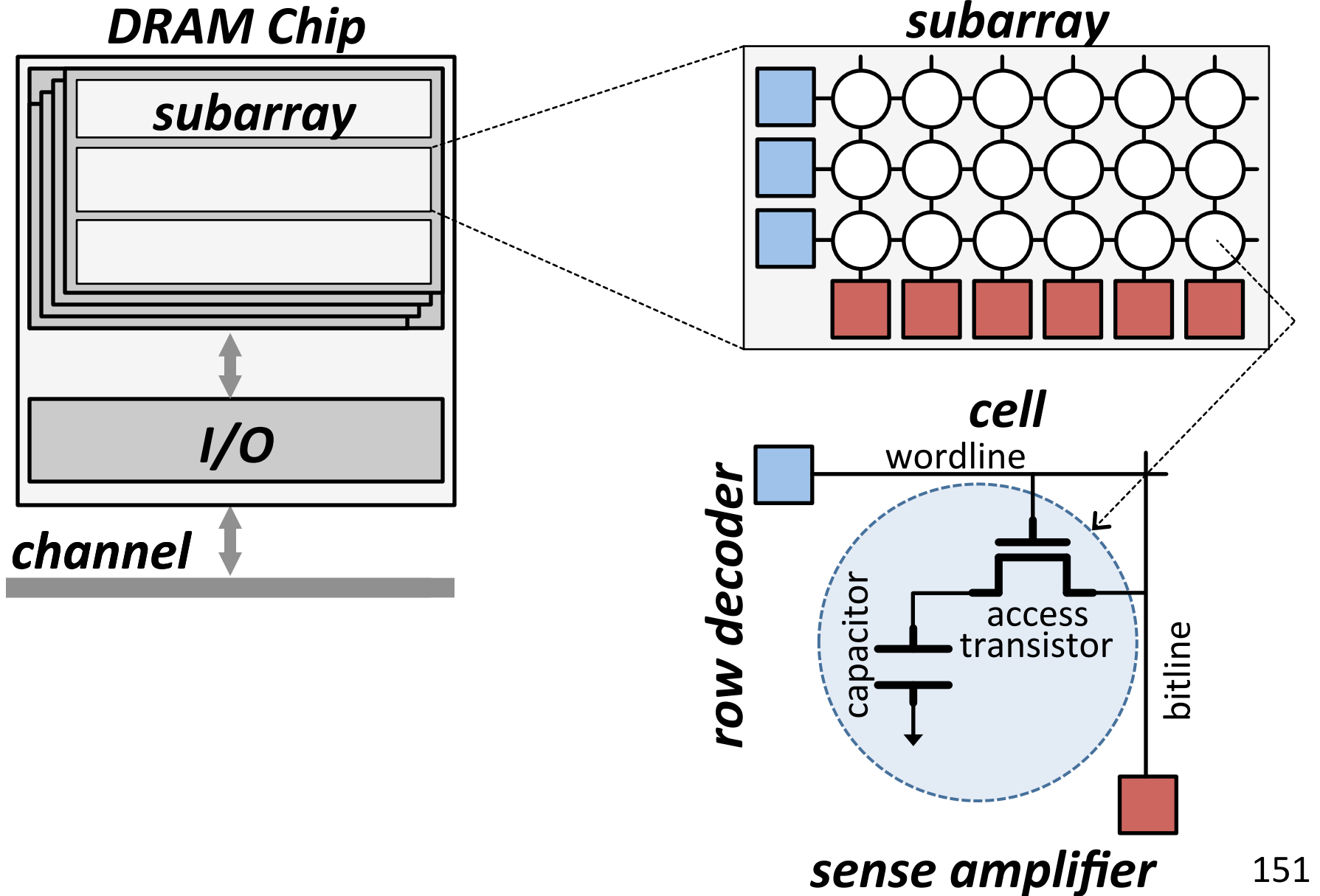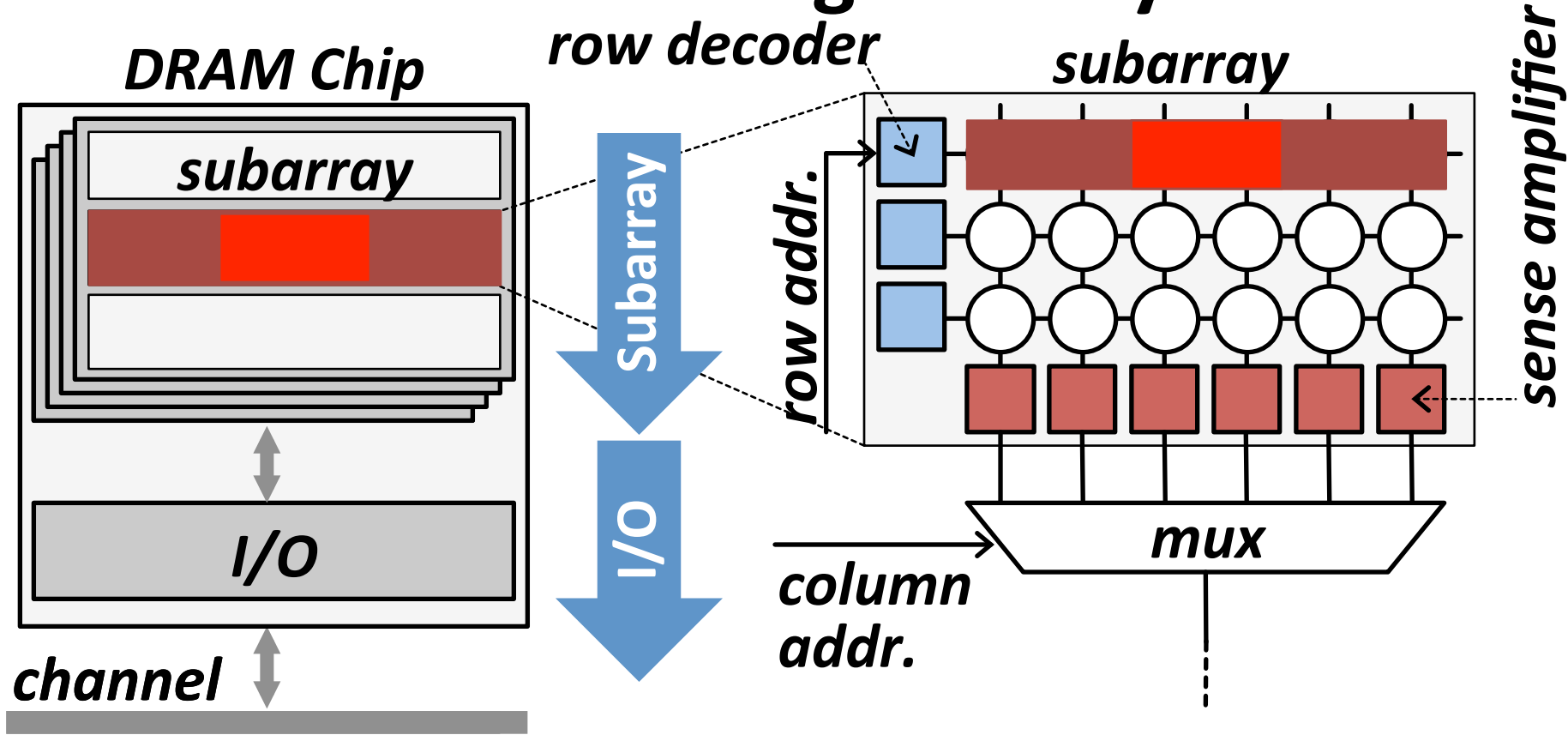
DRAM Latency = ~~Subarray Latency~~ Subarray Latency + ~~I/O Latency~~ I/O Latency

Dominant

Why is the Subarray So Slow?

Subarray

Cell

row decoder — cell — *bitline: 512 cells*

sense amplifier

wordline
access transistor
capacitor
bitline
row decoder
sense amplifier

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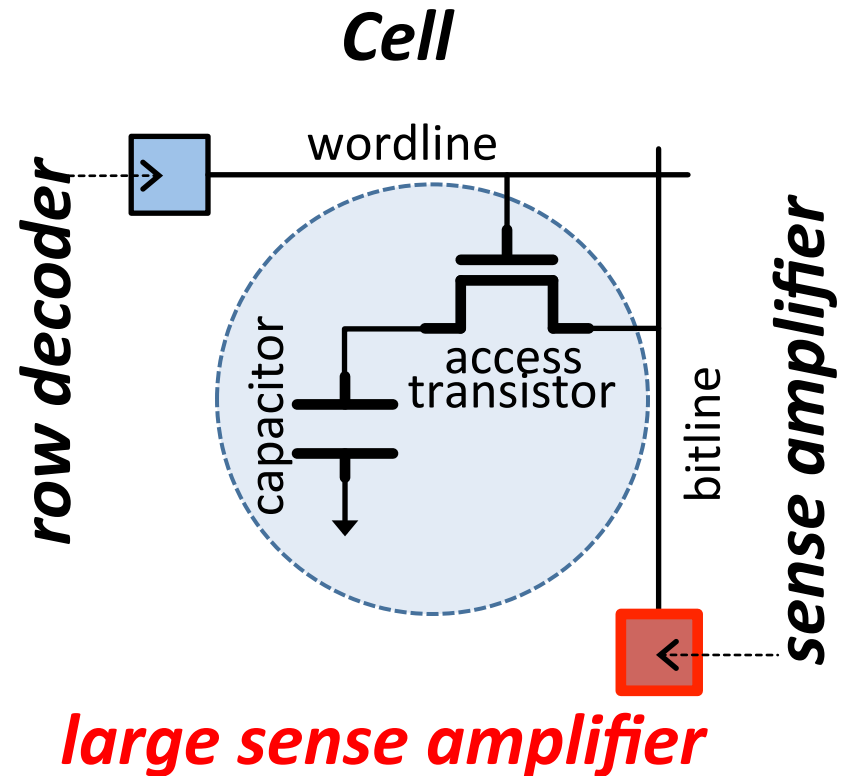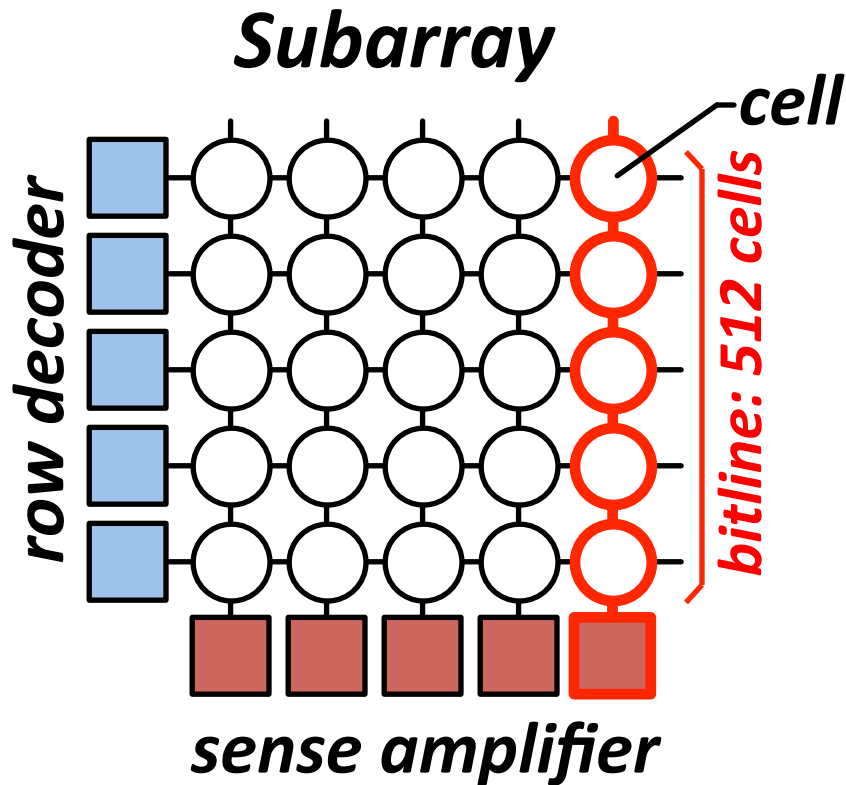
- **Long bitline**
  - **Amortizes sense amplifier cost → Small area**
  - **Large bitline capacitance → High latency & power**

153

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

Long Bitline   **Short Bitline**

Faster

Smaller
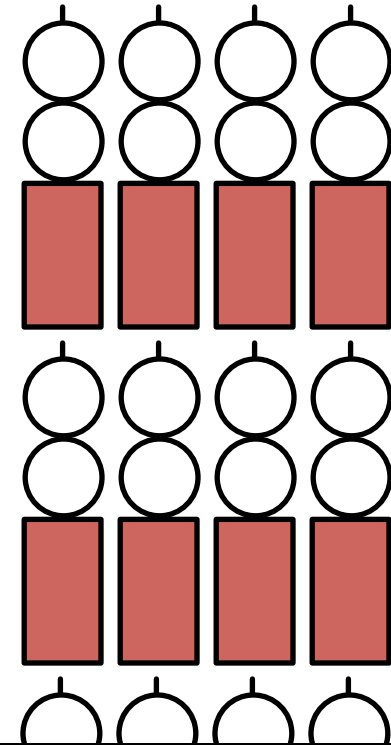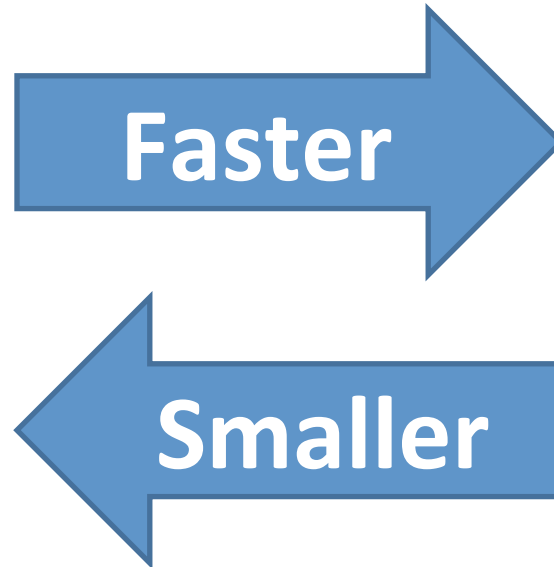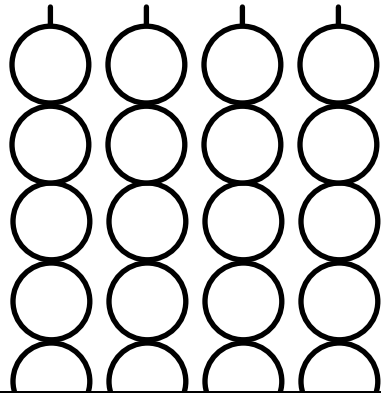
Trade-Off: Area vs. Latency

154
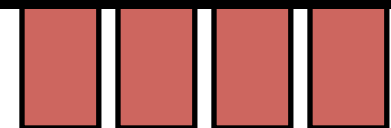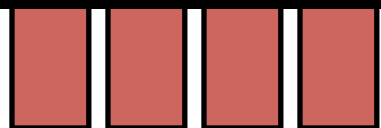
Trade-Off: Area (Die Size) vs. Latency

Approximating the Best of Both Worlds
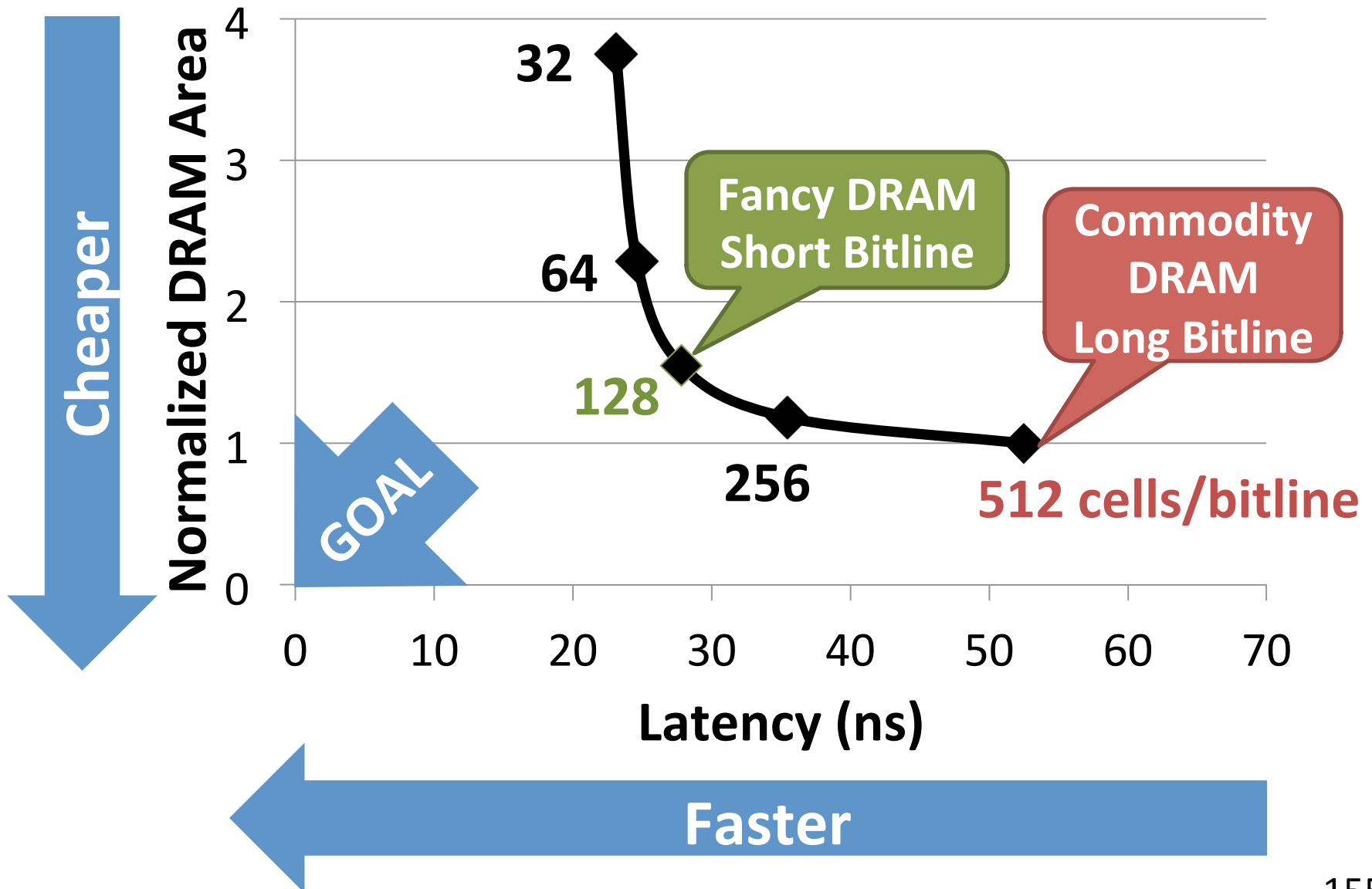
Long Bitline **Our Proposal** **Short Bitline**

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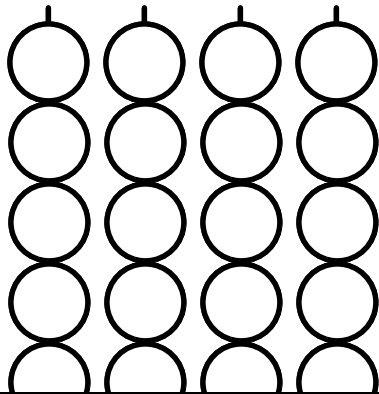
~~*High Latency*~~ *Low Latency*

Need Isolation **Add Isolation Transistors** *tline* ➔ *Fast*
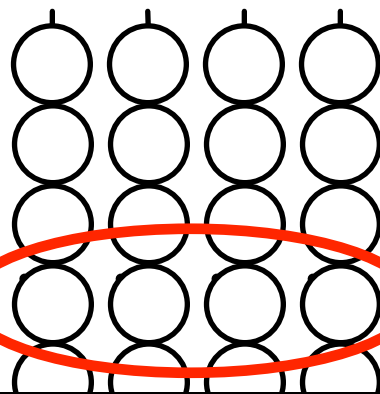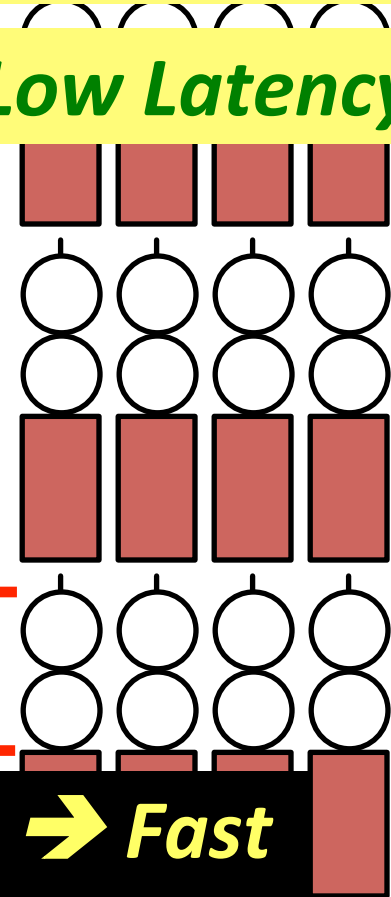
Approximating the Best of Both Worlds

Long Bitline **Tiered-Latency DRAM** **Short Bitline**

Small Area *Small Area* ~~*Large Area*~~

~~*High Latency*~~ *Low Latency* *Low Latency*

Small area using long bitline

Low Latency

157
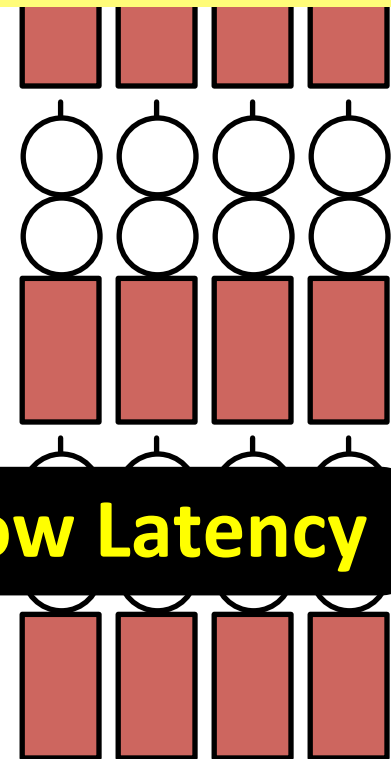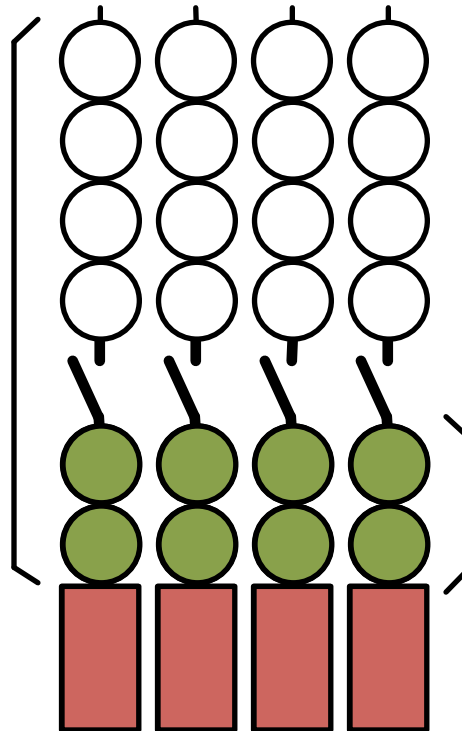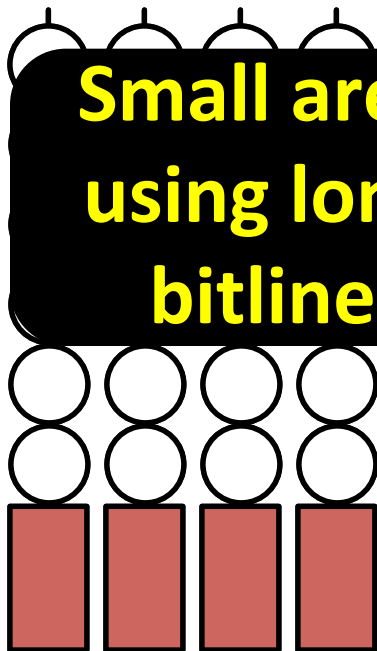
Tiered-Latency DRAM

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

Far Segment

Isolation Transistor

Near Segment

Sense Amplifier
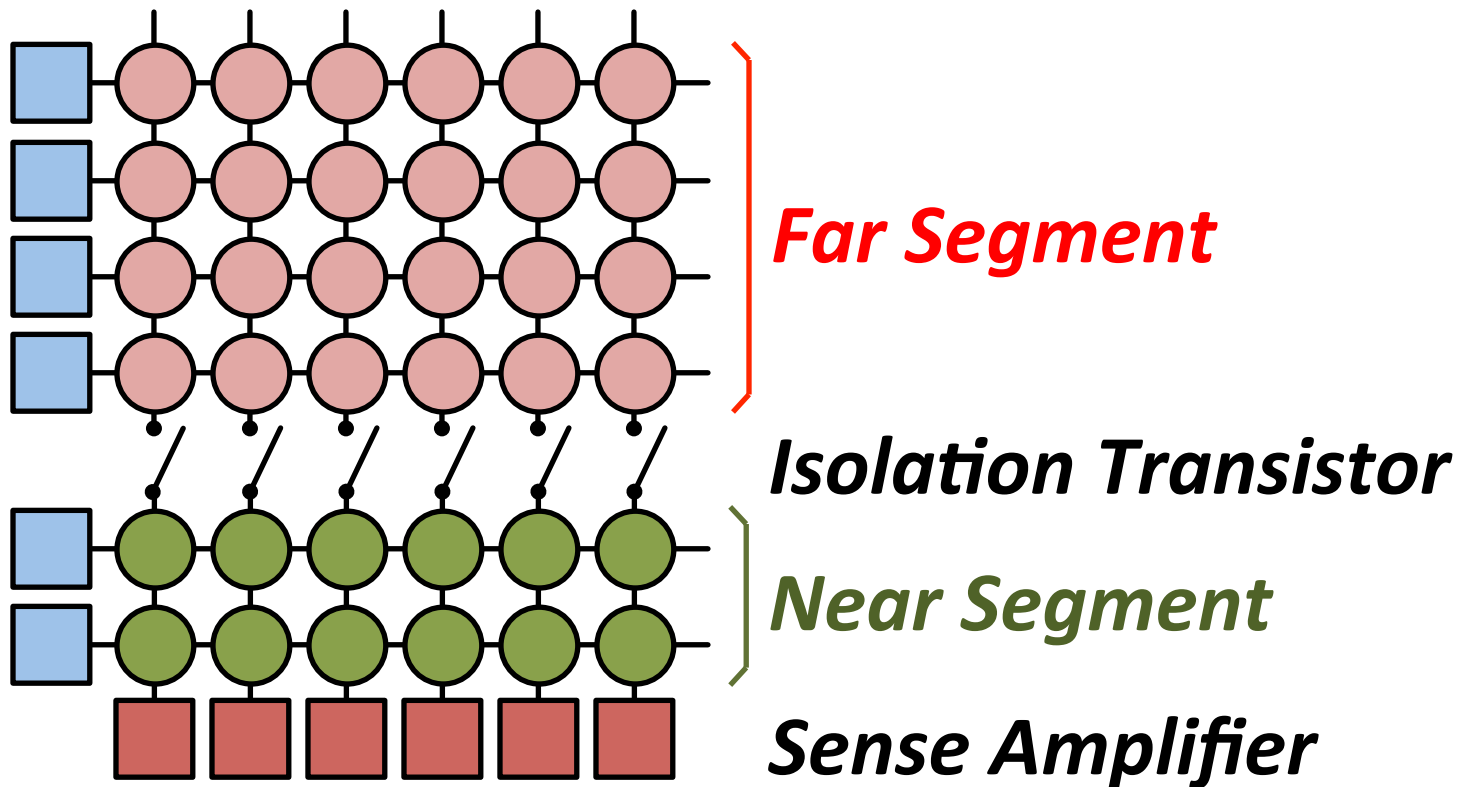
Near Segment Access

- **Turn *off* the isolation transistor**

Reduced bitline length

Reduced bitline capacitance

➔ **Low latency & low power**

*Isolation Transistor (**off**)*

Near Segment

Sense Amplifier
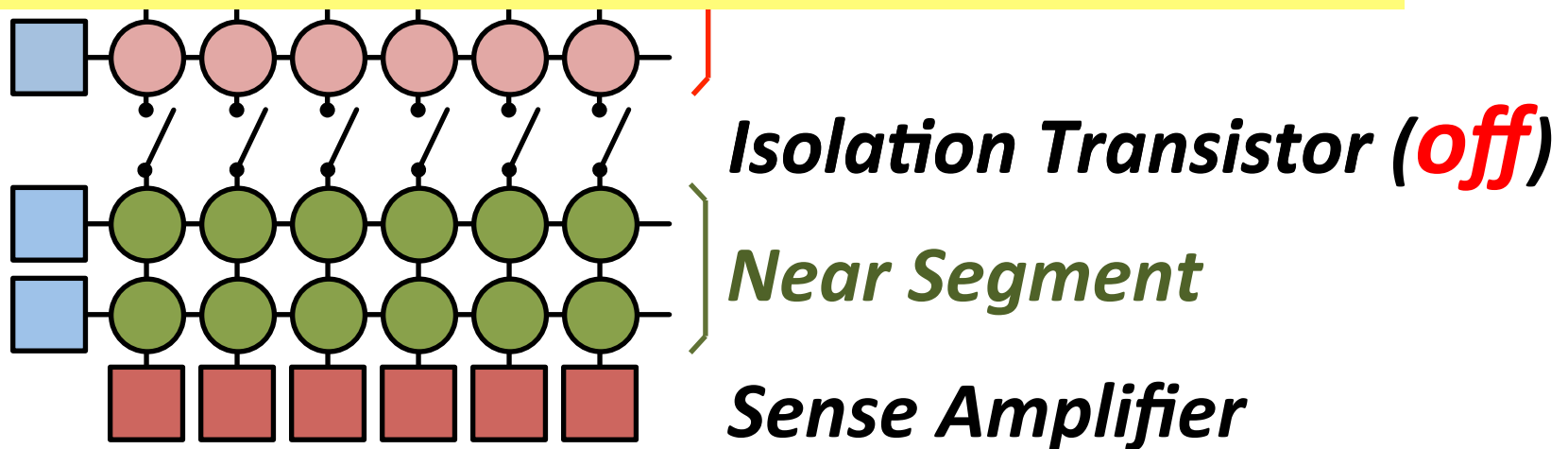
Far Segment Access

- 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
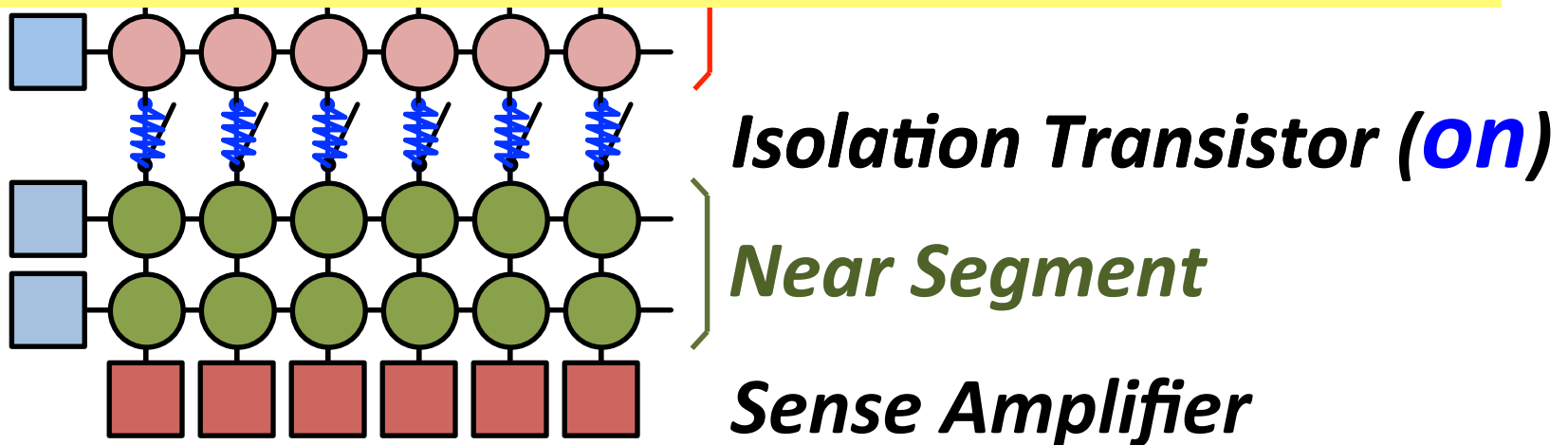
Latency, Power, and Area Evaluation

- **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**
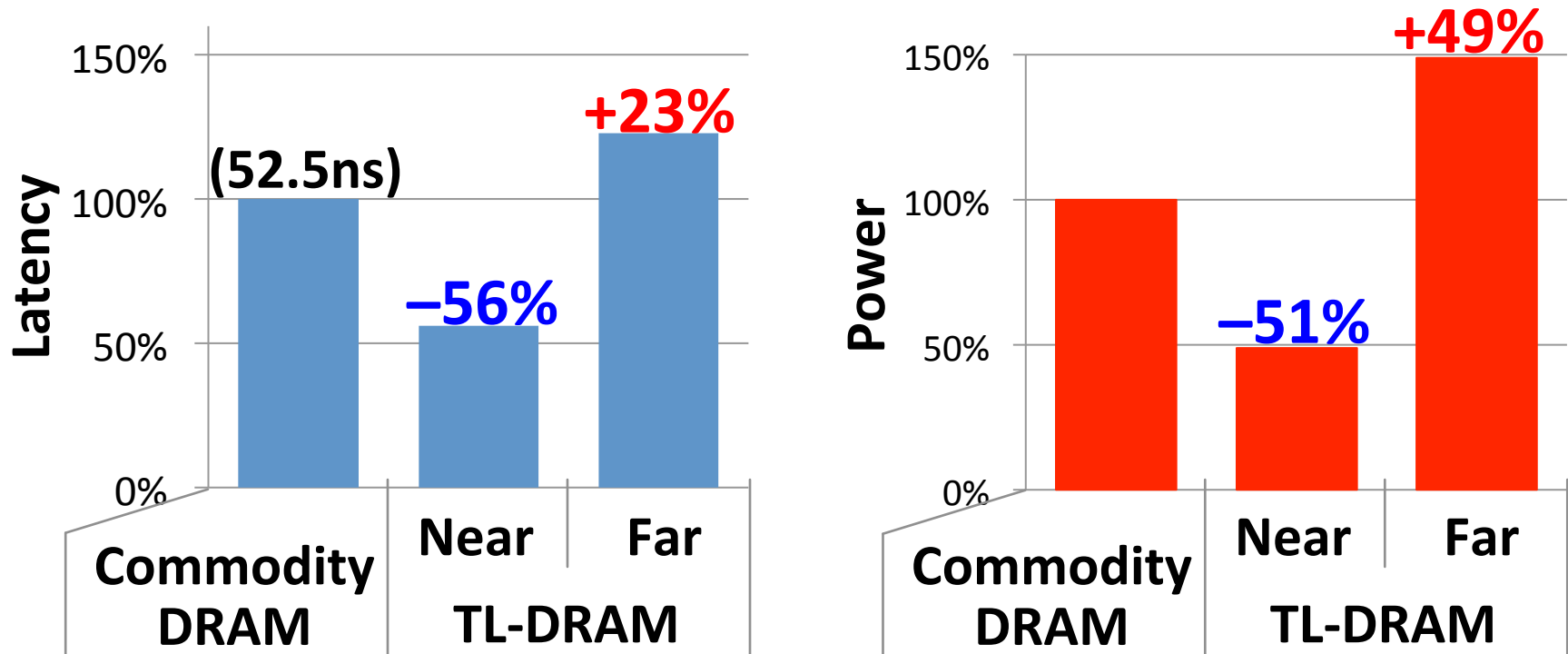
- **DRAM Area Overhead**

  ~3%: mainly due to the isolation transistors

Latency vs. Near Segment Length
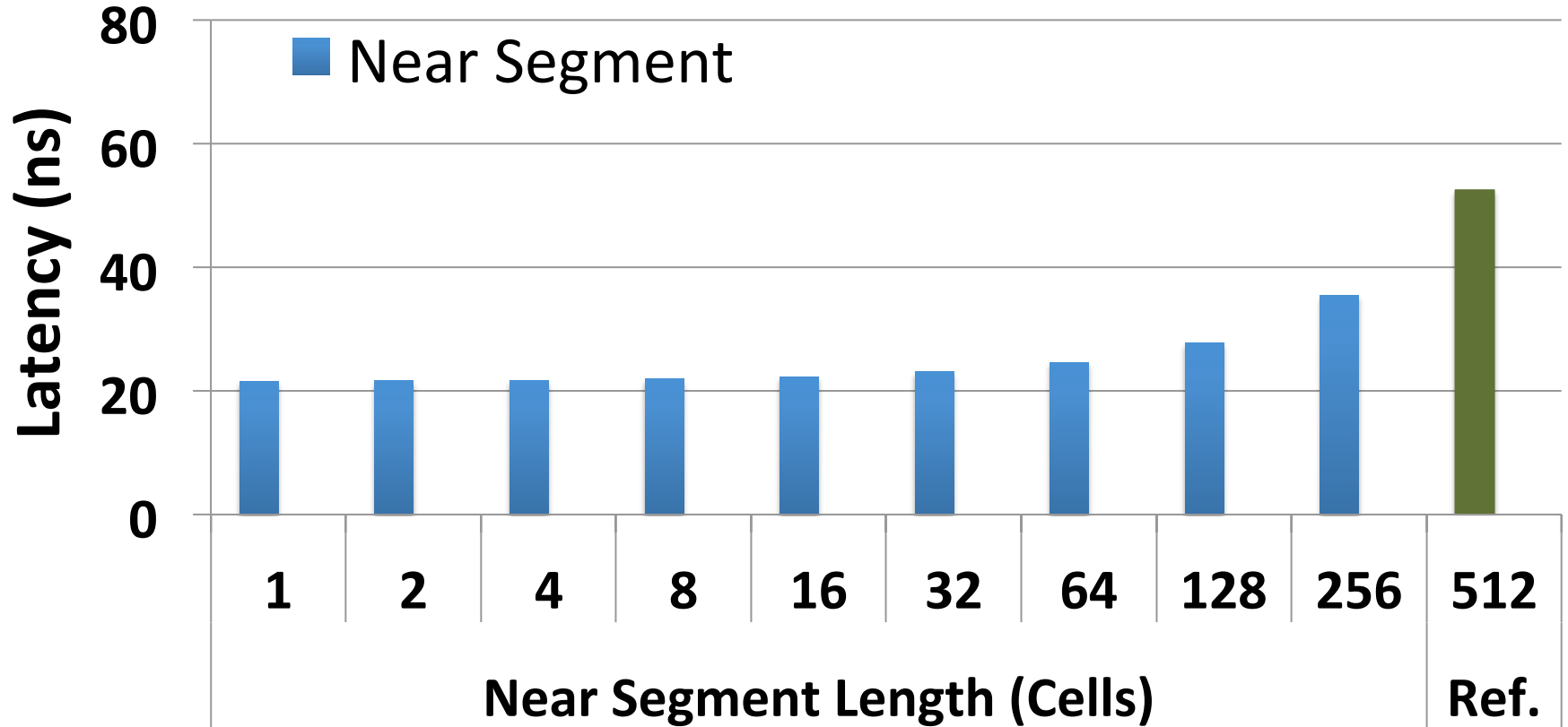
Longer near segment length leads to higher near segment latency

163
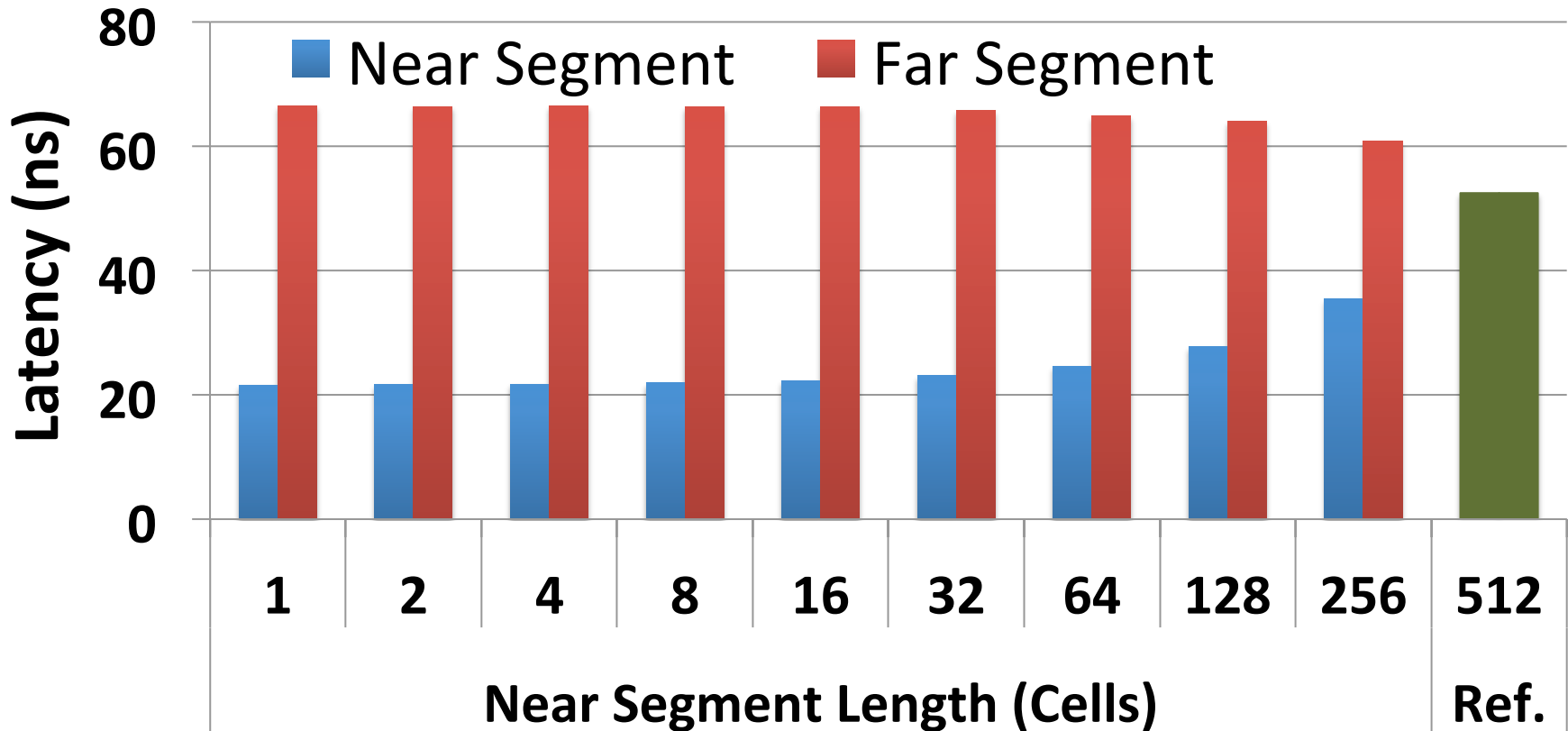
Latency vs. Near Segment Length

Far segment latency is higher than commodity DRAM latency

164

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

Cheaper ↓

Normalized DRAM Area (y-axis: 0, 1, 2, 3, 4)

32
64
128
256
512 cells/bitline

● *Near Segment*
● *Far Segment*

GOAL

Latency (ns) (x-axis: 0, 10, 20, 30, 40, 50, 60, 70)

← **Faster**
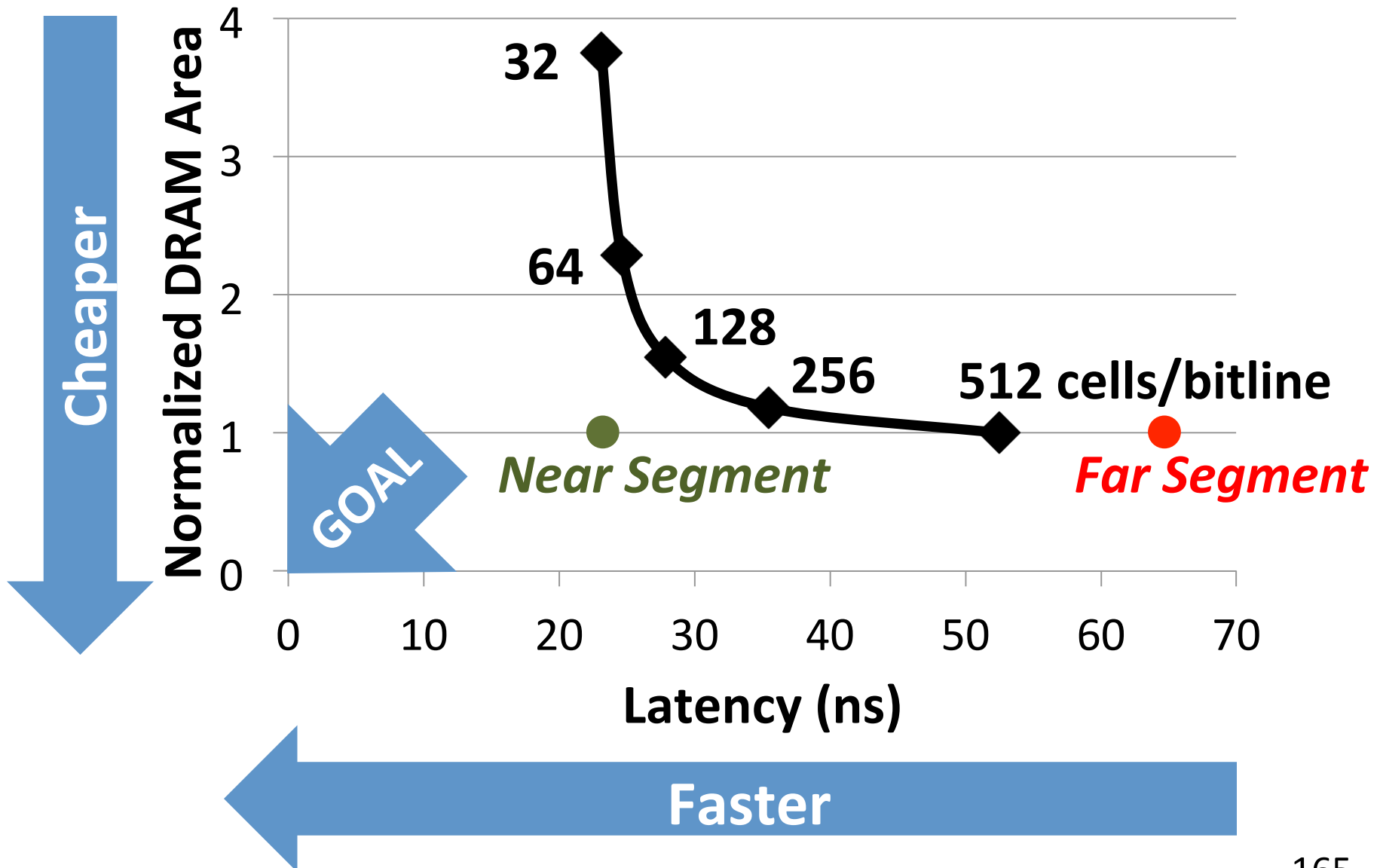
Leveraging Tiered-Latency DRAM

- 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
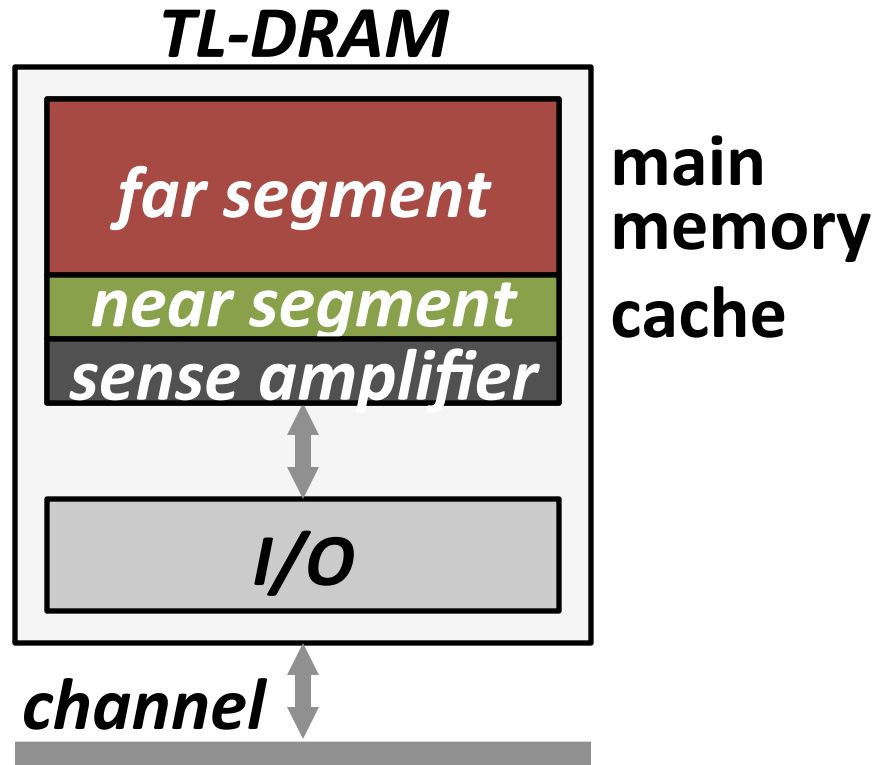
TL-DRAM

- **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*
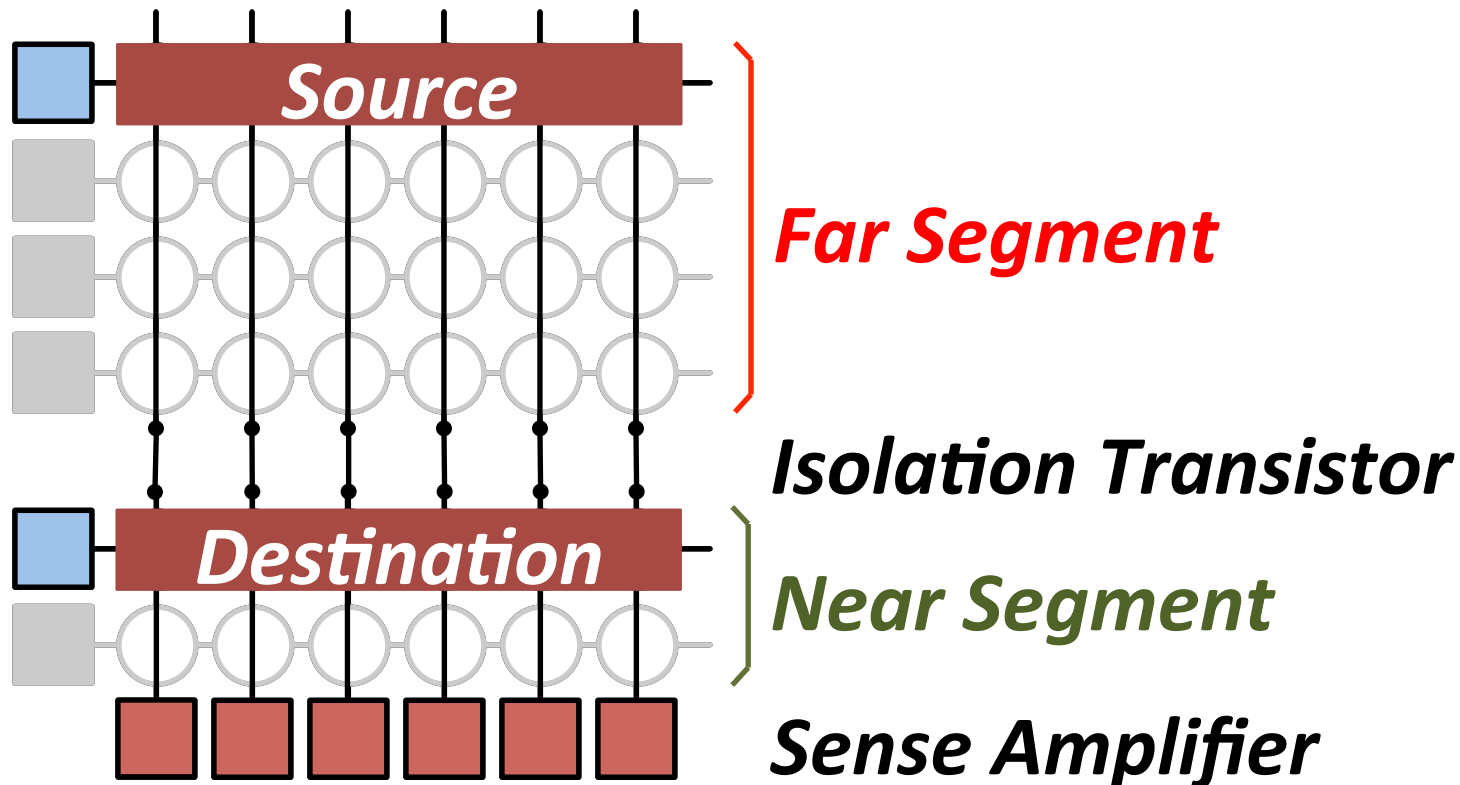
Source

Far Segment

Isolation Transistor

Destination

Near Segment

Sense Amplifier

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
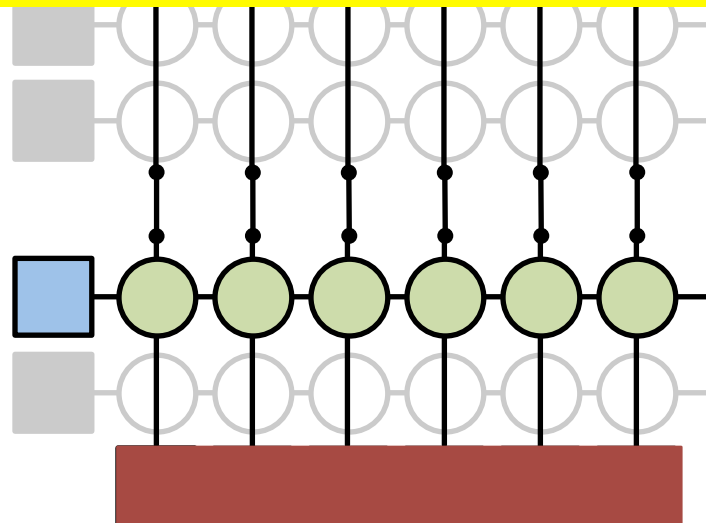
> **Step 2:** Activate destination row to connect cell and bitline

Isc... ...ll Transistor

Near Segment

Sense Amplifier

Near Segment as Hardware-Managed Cache
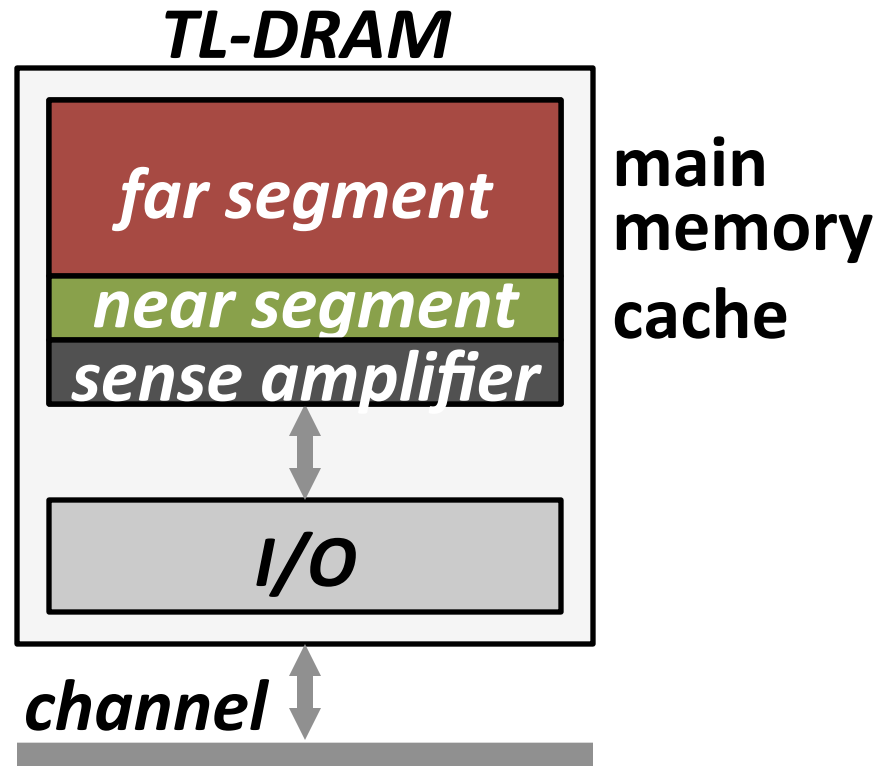
TL-DRAM

far segment

near segment

sense amplifier

main
memory

cache

I/O

channel

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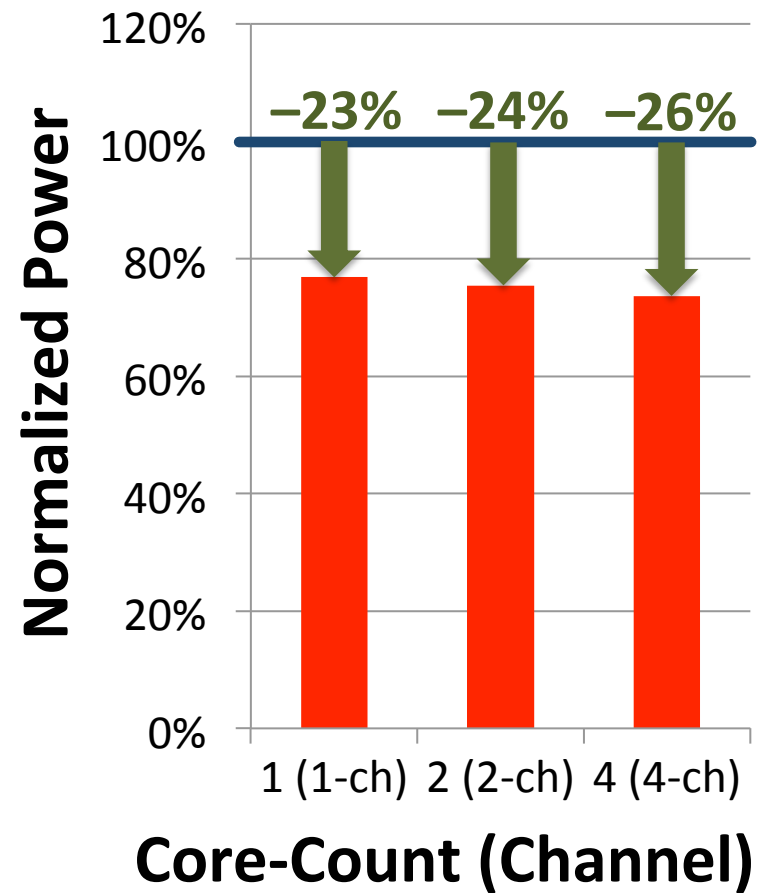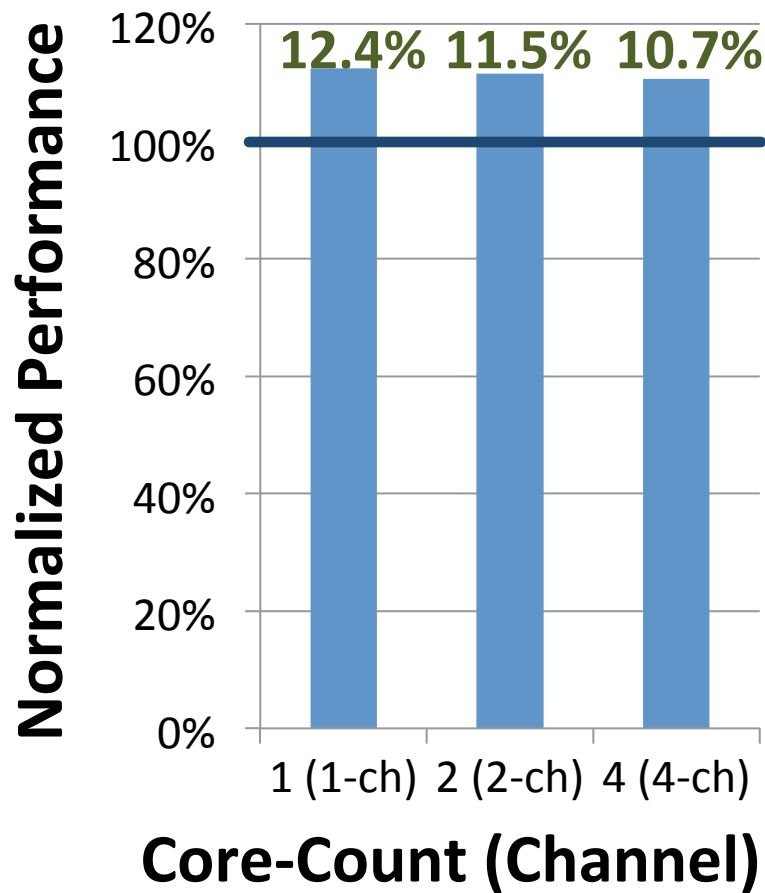
Using near segment as a cache improves performance and reduces power consumption

174

Single-Core: Varying Near Segment Length
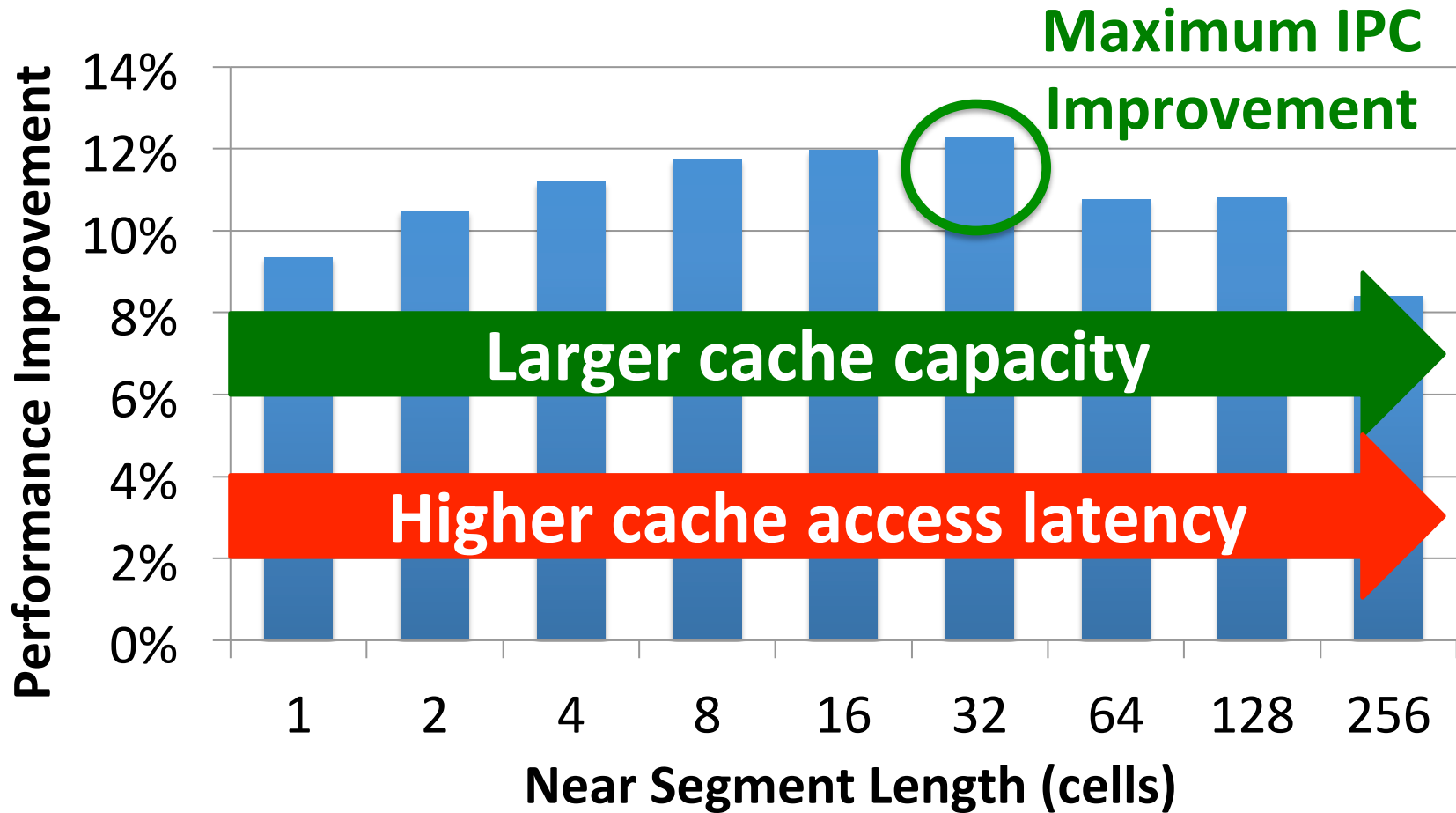
By adjusting the near segment length, we can trade off cache capacity for cache latency

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

SAFARI

Subarray-Level Parallelism: Reducing Bank Conflict Impact

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

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.

SAFARI

The Problem with Memory Bank Conflicts

- **Two Banks**

Served in parallel

Bank

| Wr | Rd |

time

Bank

| Wr | Rd |

time

- **One Bank**

Wasted

Bank

| Wr | 2 | 3 | Wr | 2 | 3 | Rd | 3 | Rd |

time

2. Serialization
3. Thrashed Row-Buffer
1. Write Penalty
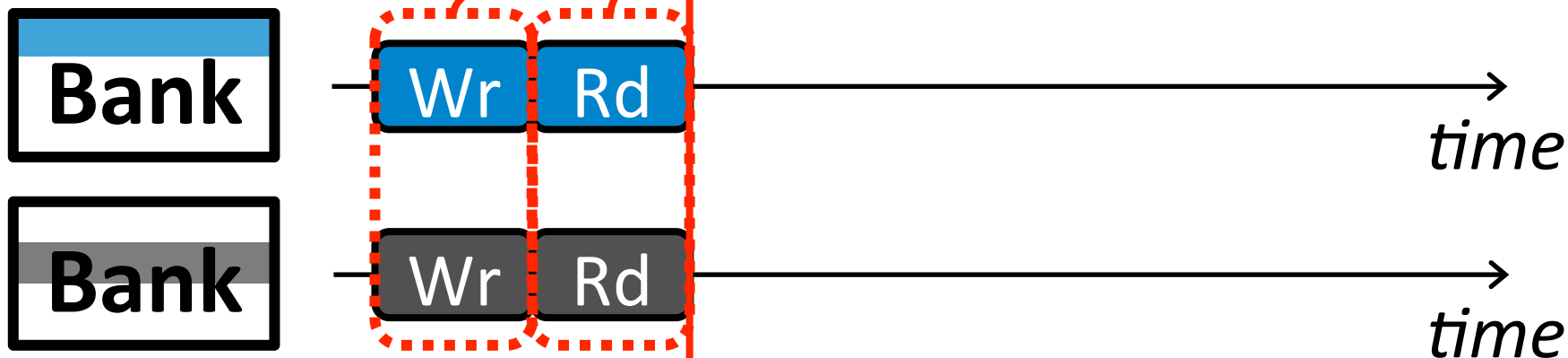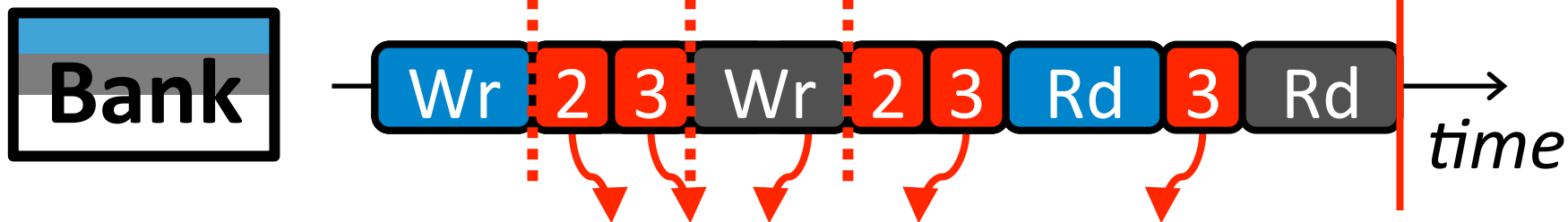
Goal

- **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$_1$
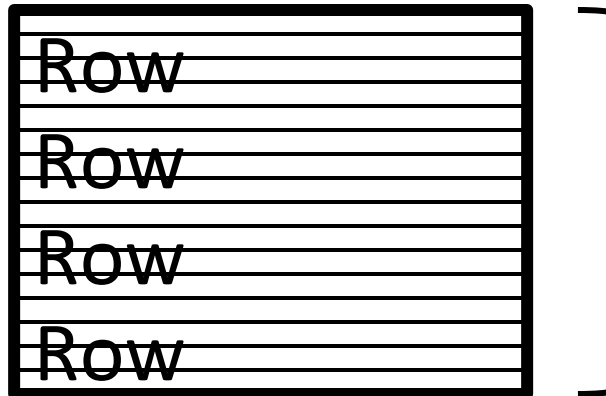
Row-Buffer
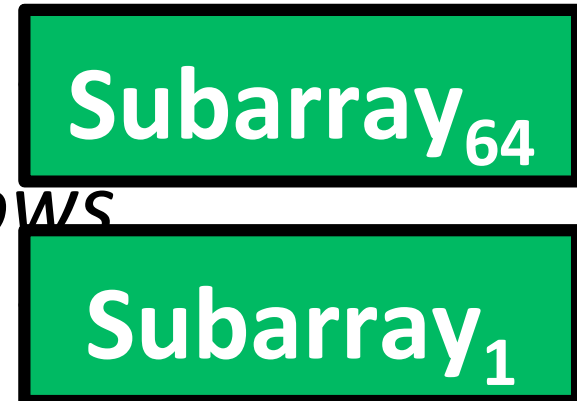
Global Row-Buf

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*
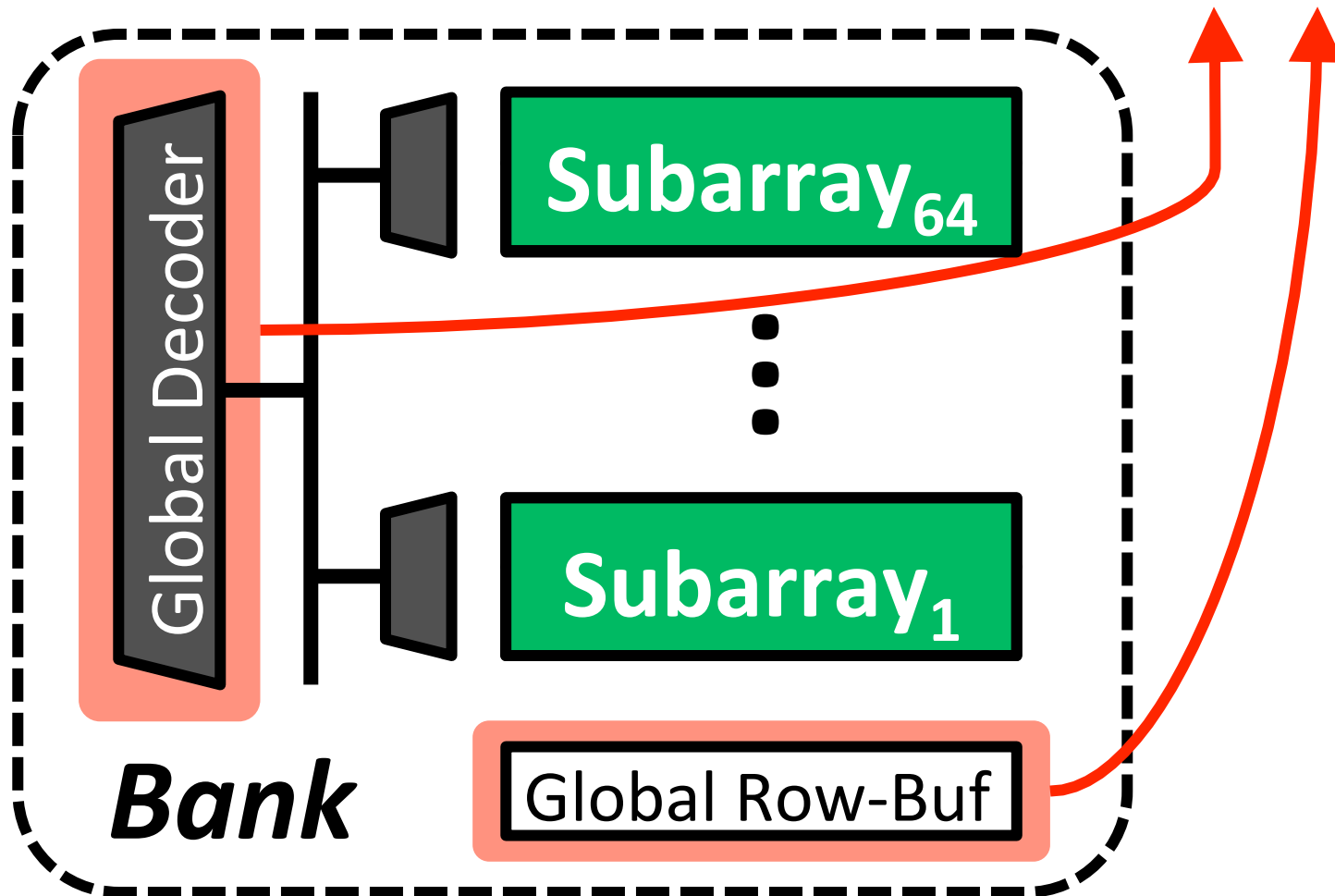
Key Idea: Reduce Sharing of Globals

1. Parallel access to subarrays

Global Decoder

Local Row-Buf

Local Row-Buf

Bank
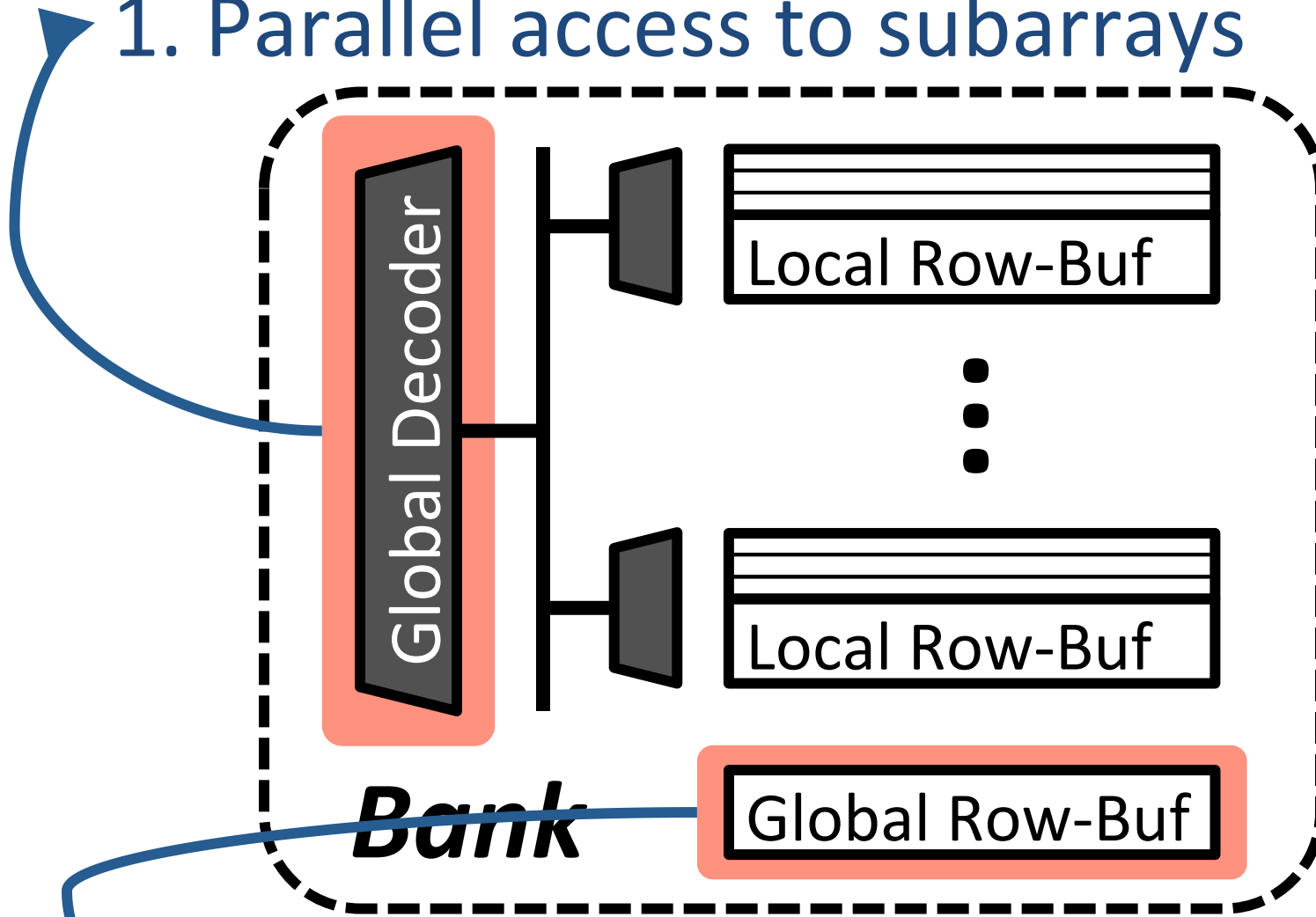
Global Row-Buf

2. Utilize multiple local row-buffers

Overview of Our Mechanism

Subarray$_{64}$

⋮

Subarray$_1$

Global Row-Buf

1. Parallelize

Reg Reg Reg Reg
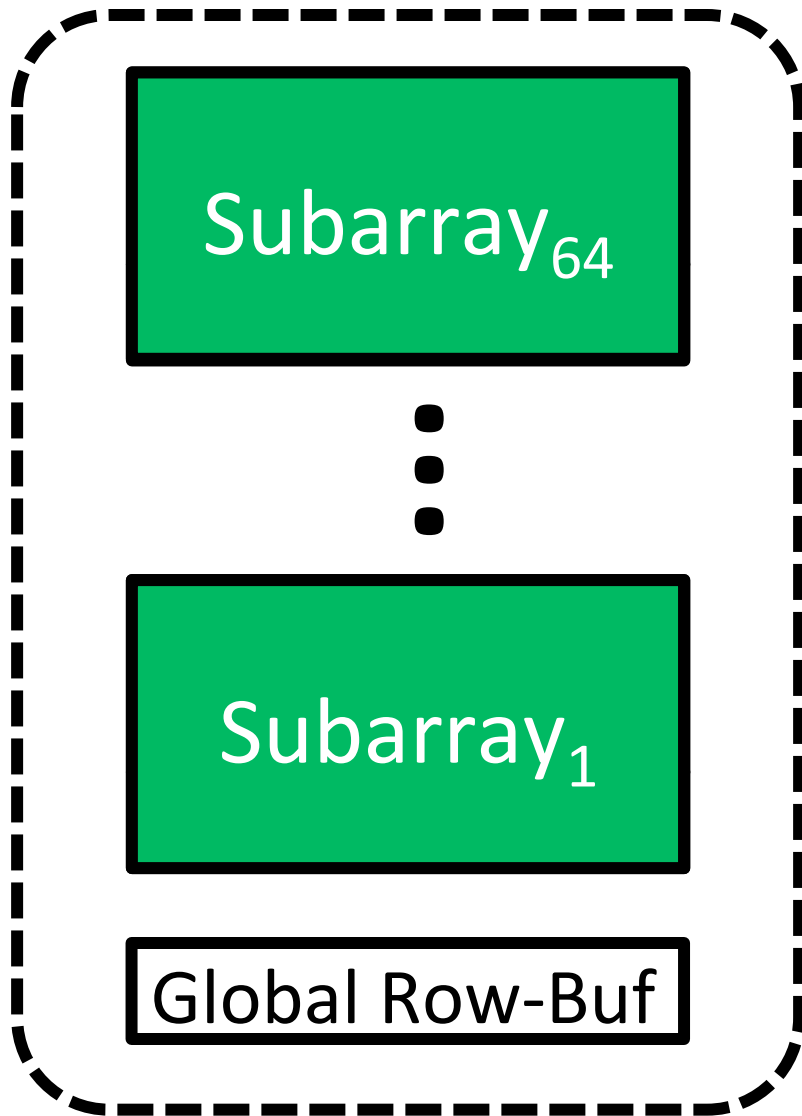
2. Utilize multiple row-buffers

*To same bank...
but diff. subarrays*

Challenges: Global Structures

1. Global Address Latch

2. Global Bitlines

Challenge #1. Global Address Latch
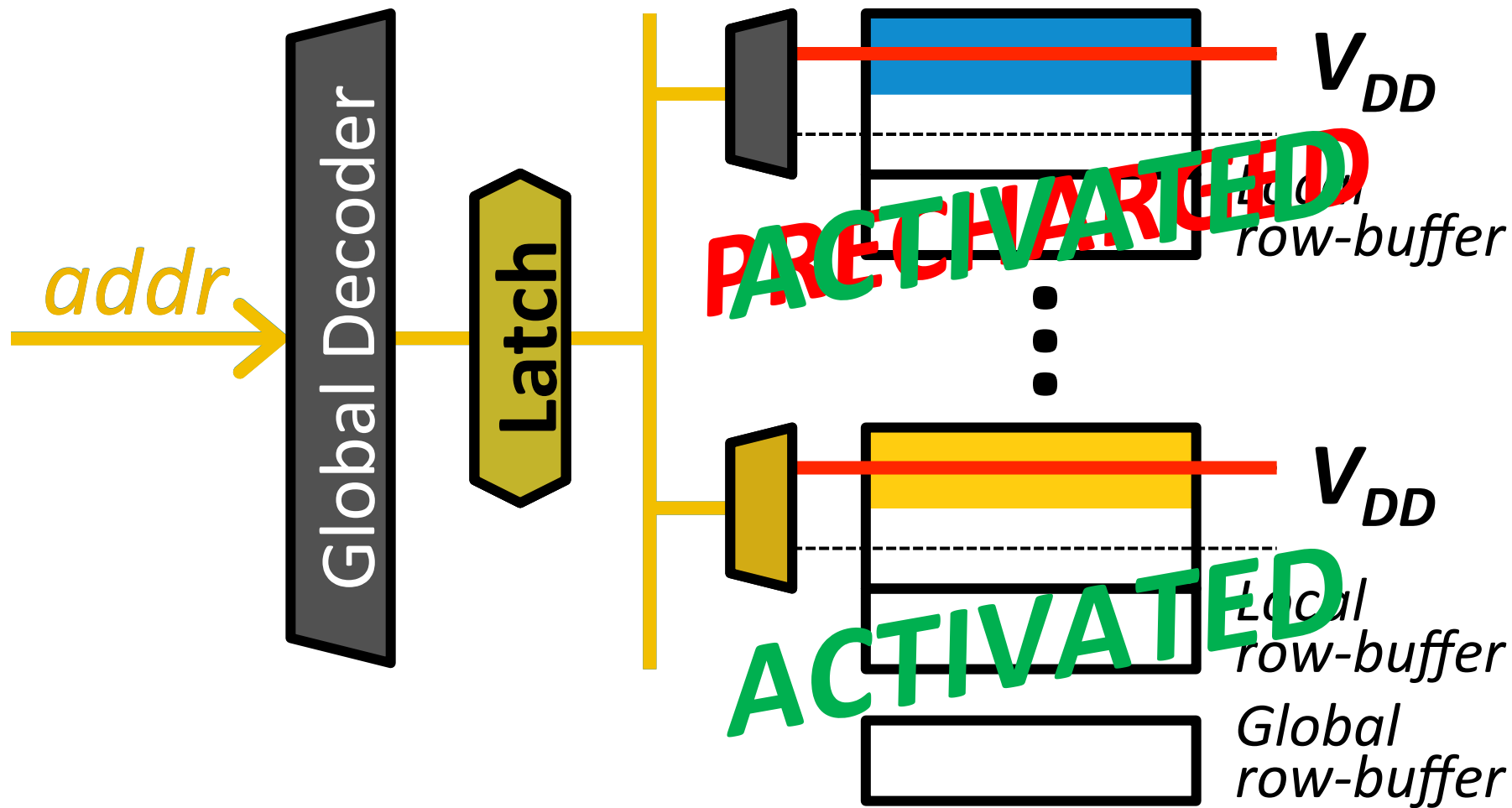
Solution #1. Subarray Address Latch

V_{DD}

V_{DD}

ACTIVATED

ACTIVATED

Local row-buffer

Local row-buffer

Global row-buffer
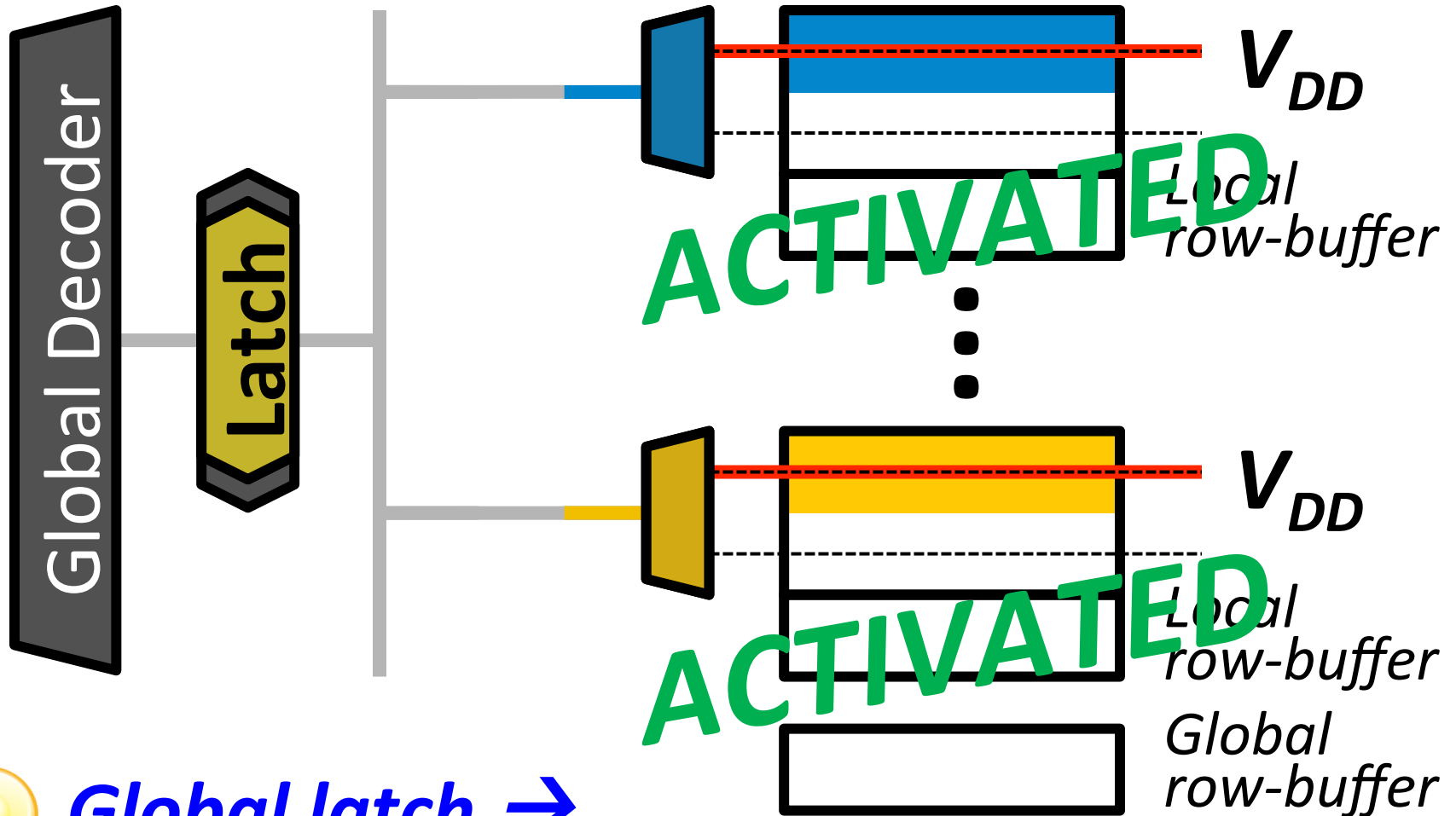
💡 *Global latch → local latches*

189

Challenges: Global Structures

1. Global Address Latch

- Problem: Only *one* raised wordline

- Solution: **Subarray Address Latch**

2. Global Bitlines

Challenge #2. Global Bitlines
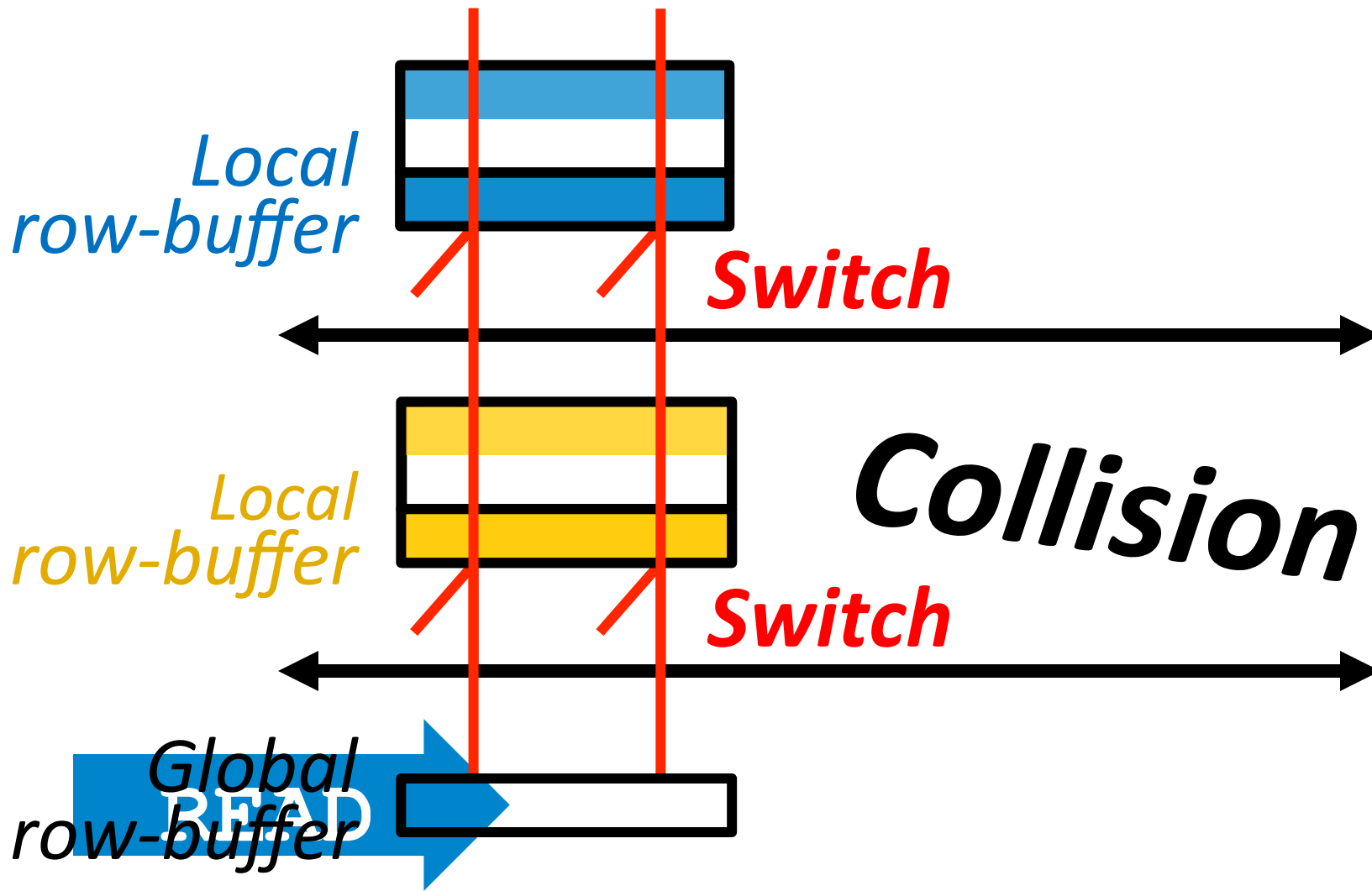
Solution #2. Designated-Bit Latch

Wire

Local row-buffer

D

Global bitlines

Switch

Local row-buffer

D

Switch

READ

Global row-buffer
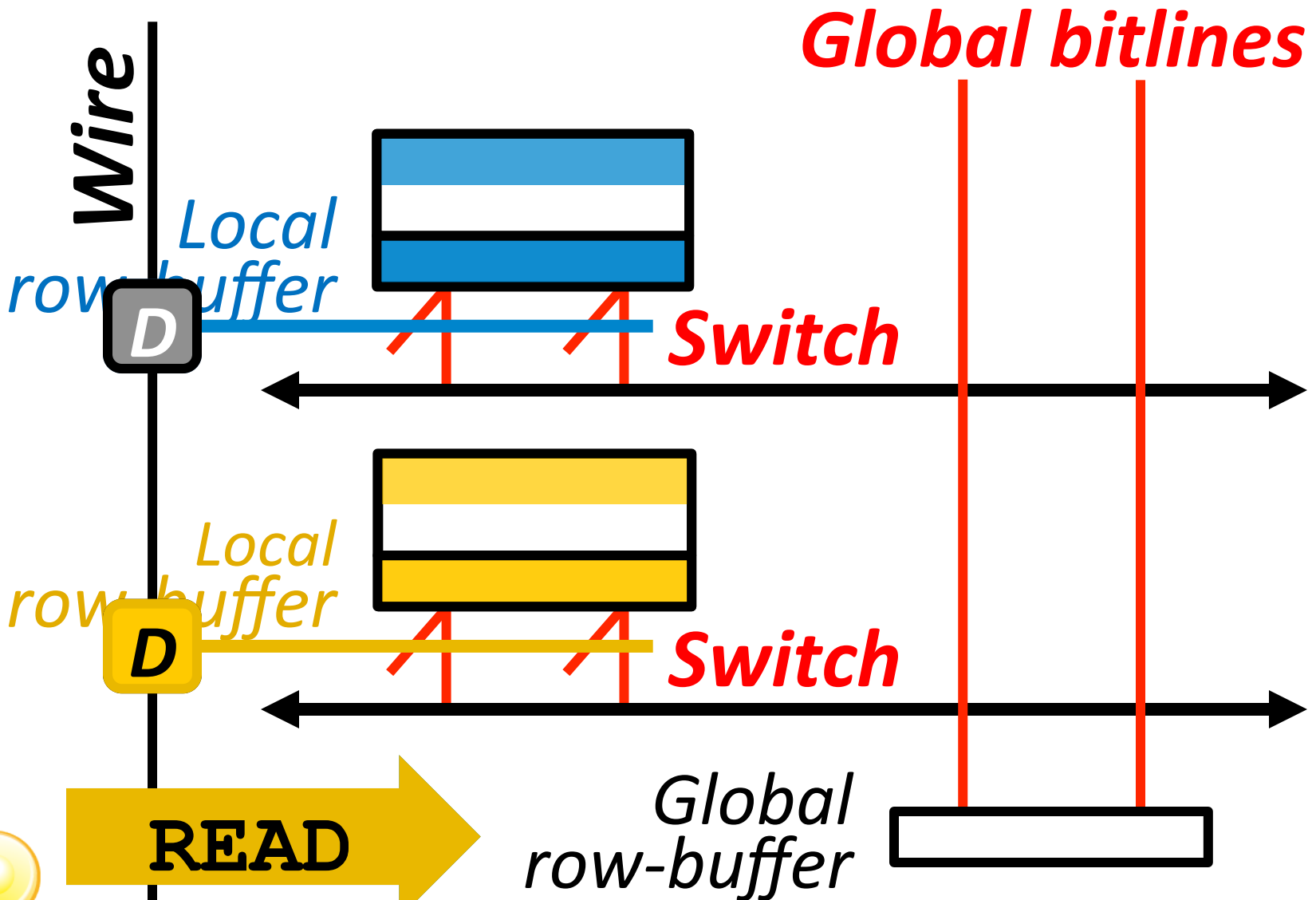
Selectively connect local to global

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)
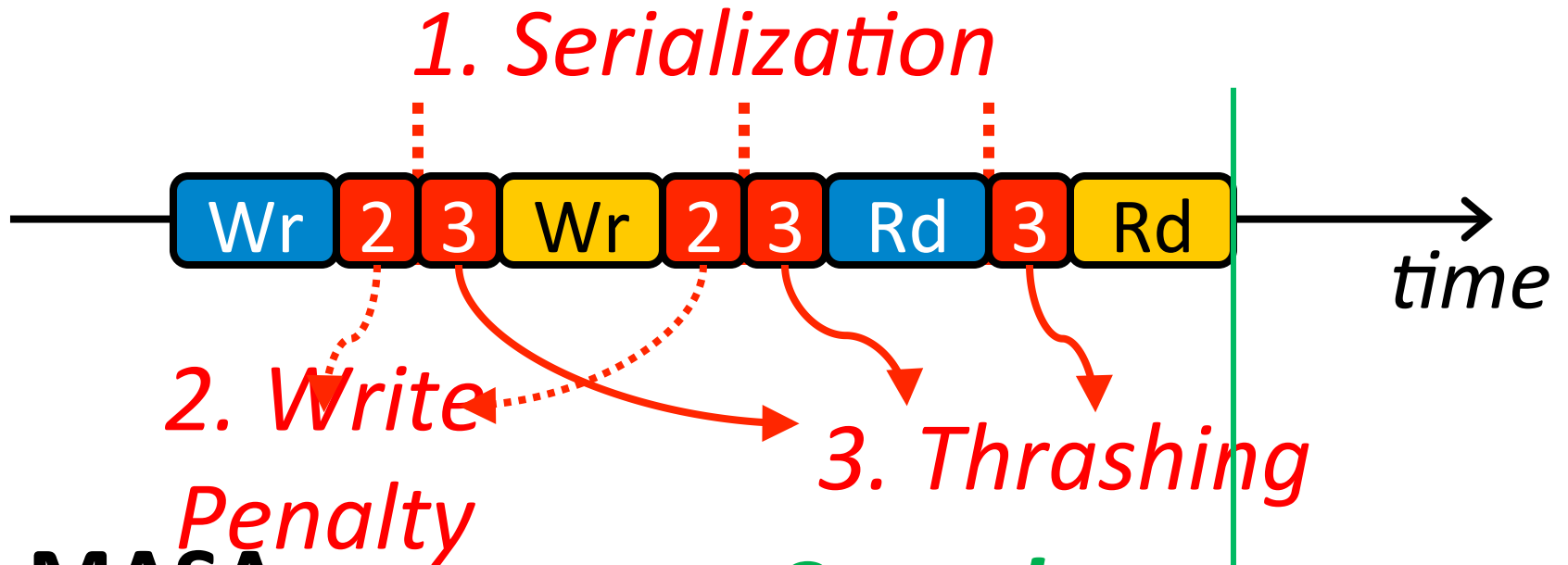
- **MASA**
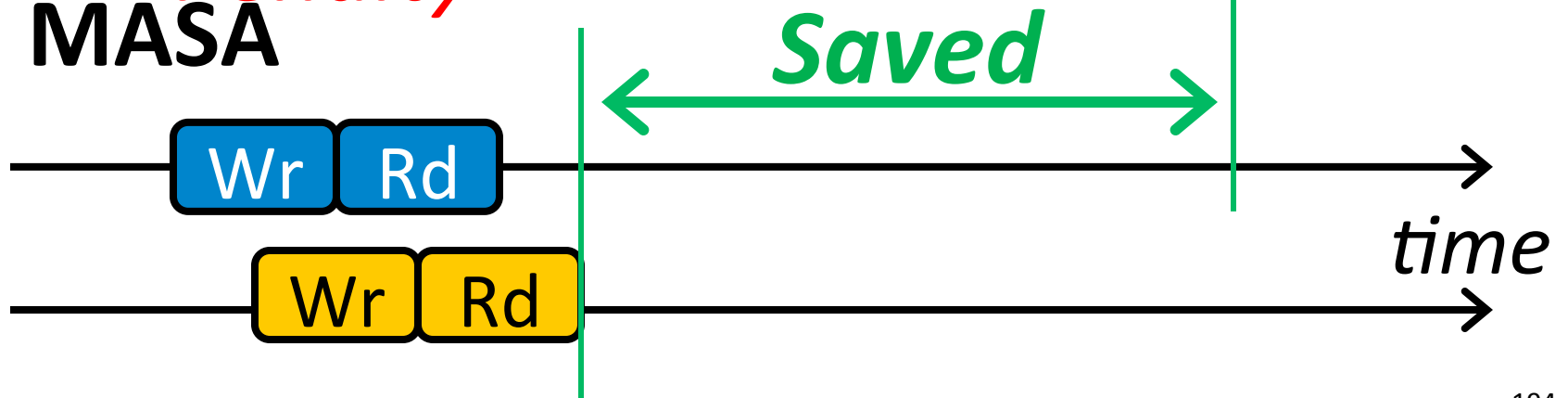
MASA: Overhead

- **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
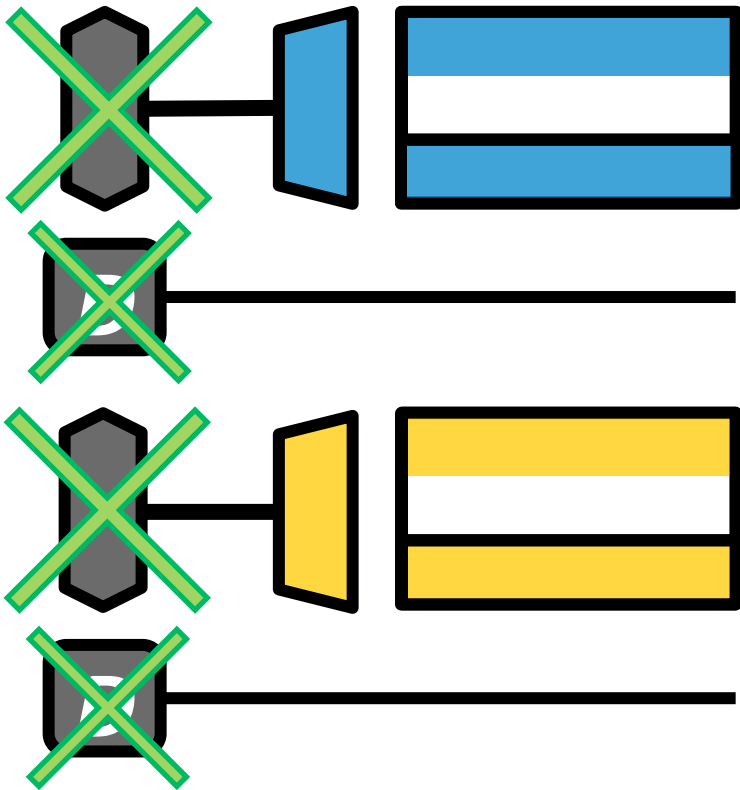
System Configuration

- **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
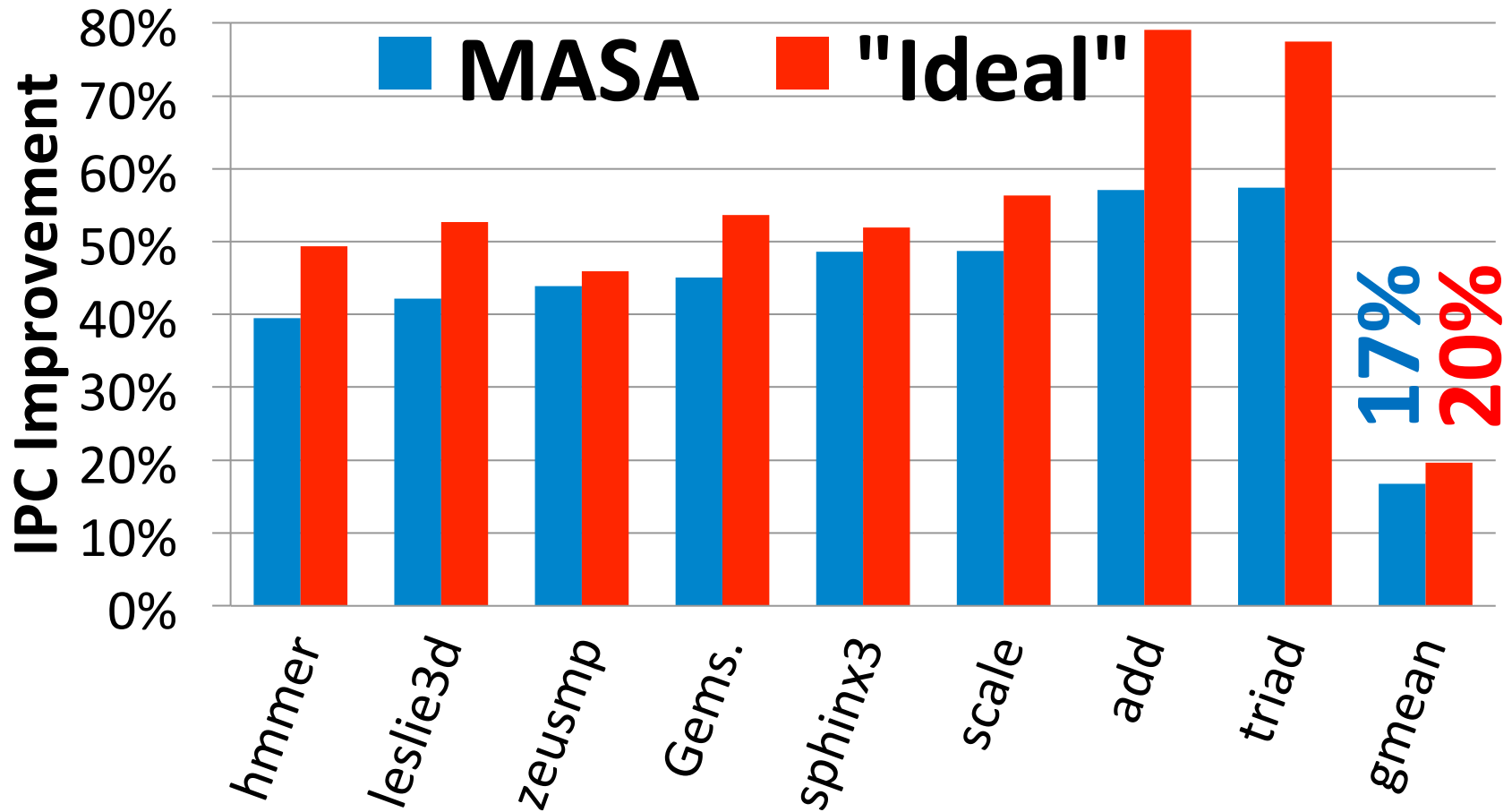
MASA *achieves most of the benefit of having more banks ("Ideal")*

198

SALP: Single-Core Results

SALP-1 ■ SALP-2 ■ MASA ■ "Ideal"

	IPC Increase		
7%	13%	17%	20%
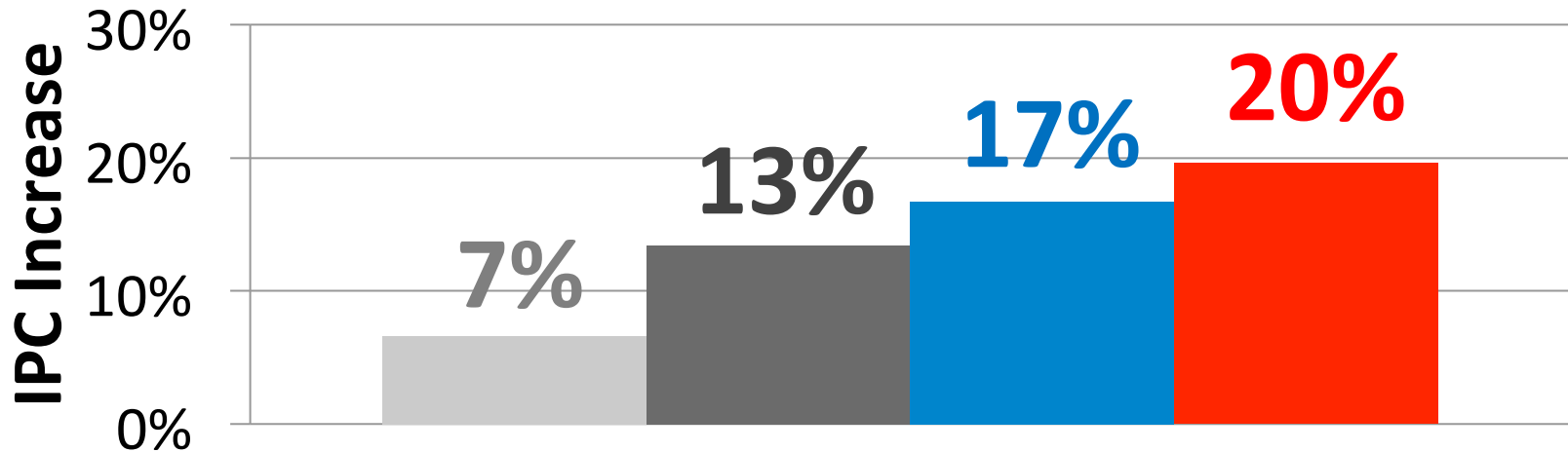
| DRAM Die Area | < 0.15% | 0.15% | 36.3% |

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

Subarray-Level Parallelism: Results
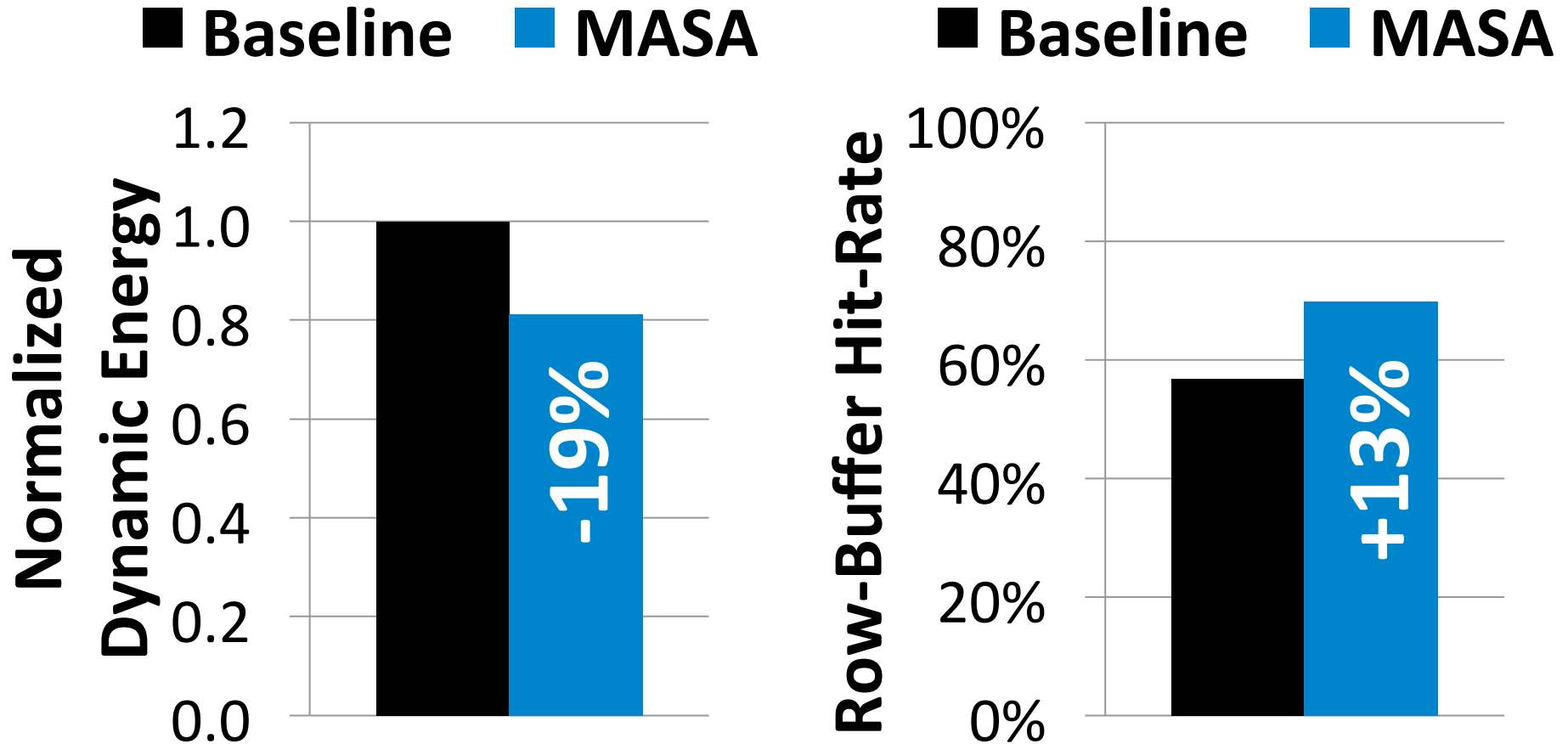
MASA increases energy-efficiency

New DRAM Architectures

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

SAFARI

RowClone: Fast Bulk Data Copy and Initialization

Vivek Seshadri, Yoongu Kim, Chris Fallin, Donghyuk Lee, Rachata Ausavarungnirun,
Gennady Pekhimenko, Yixin Luo, Onur Mutlu, Phillip B. Gibbons, Michael A. Kozuch, Todd C. Mowry,
"RowClone: Fast and Efficient In-DRAM Copy and Initialization of Bulk Data"
CMU Computer Science Technical Report, CMU-CS-13-108, Carnegie Mellon University, April 2013.

Today's Memory: Bulk Data Copy

3) Cache pollution

1) High latency

2) High bandwidth utilization

4) Unwanted data movement
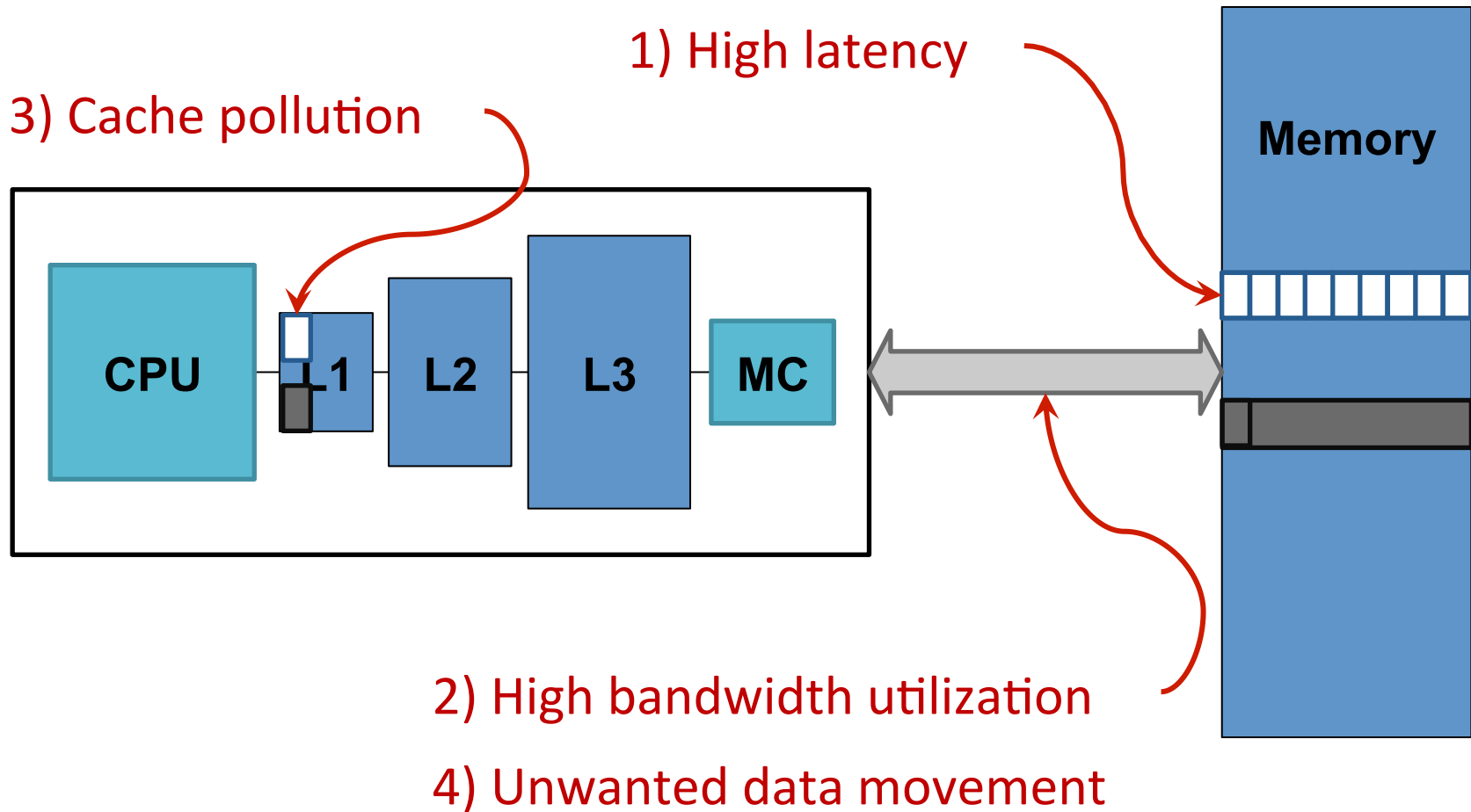
Future: RowClone (In-Memory Copy)
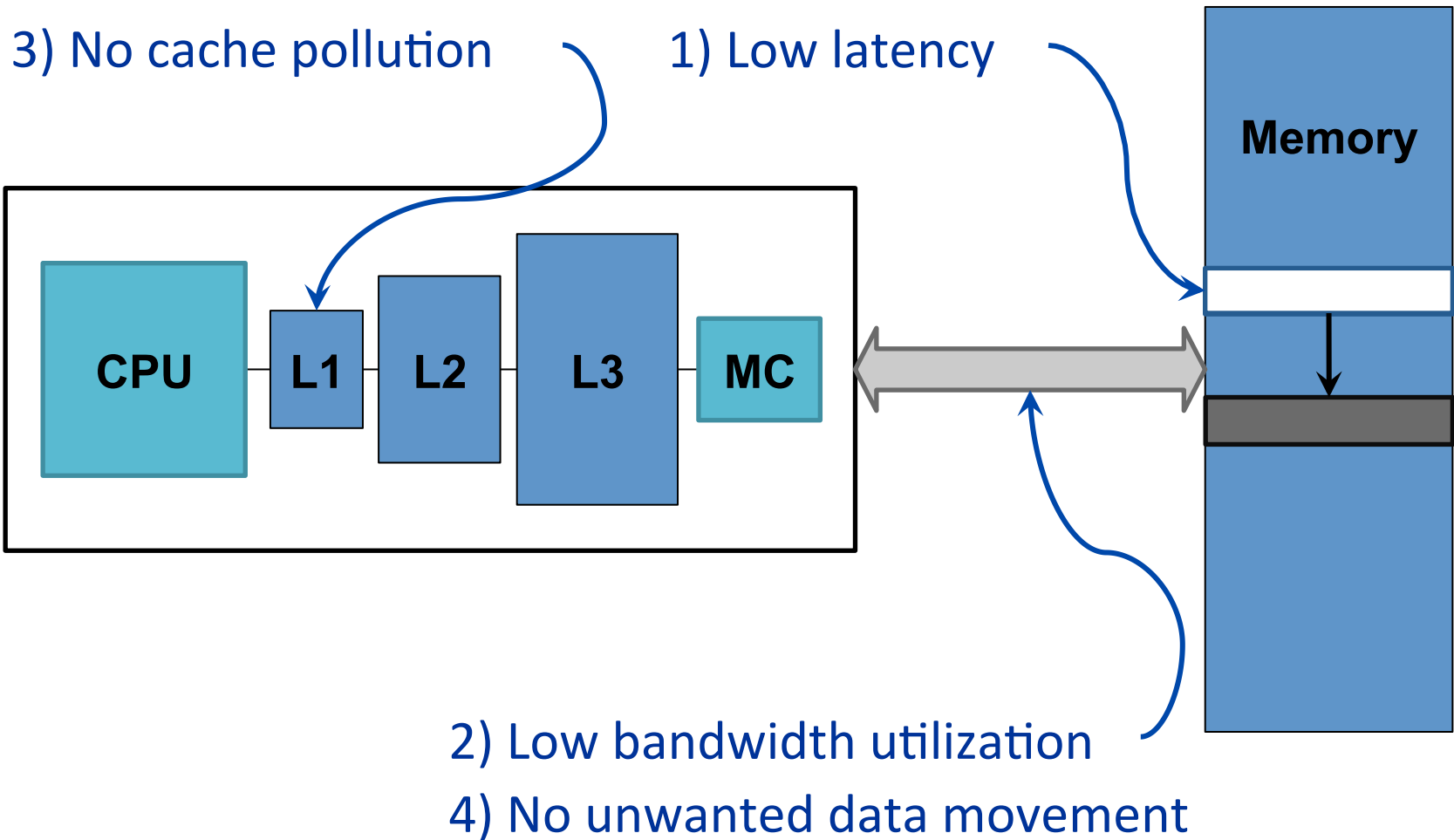
3) No cache pollution

1) Low latency

Memory

CPU | **L1** | **L2** | **L3** | **MC**

2) Low bandwidth utilization

4) No unwanted data movement

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

DRAM operation (load one byte)

4 Kbits

1. Activate row

2. Transfer row

DRAM array

Row Buffer (4 Kbits)

3. Transfer byte onto bus

Data pins (8 bits)

Memory Bus
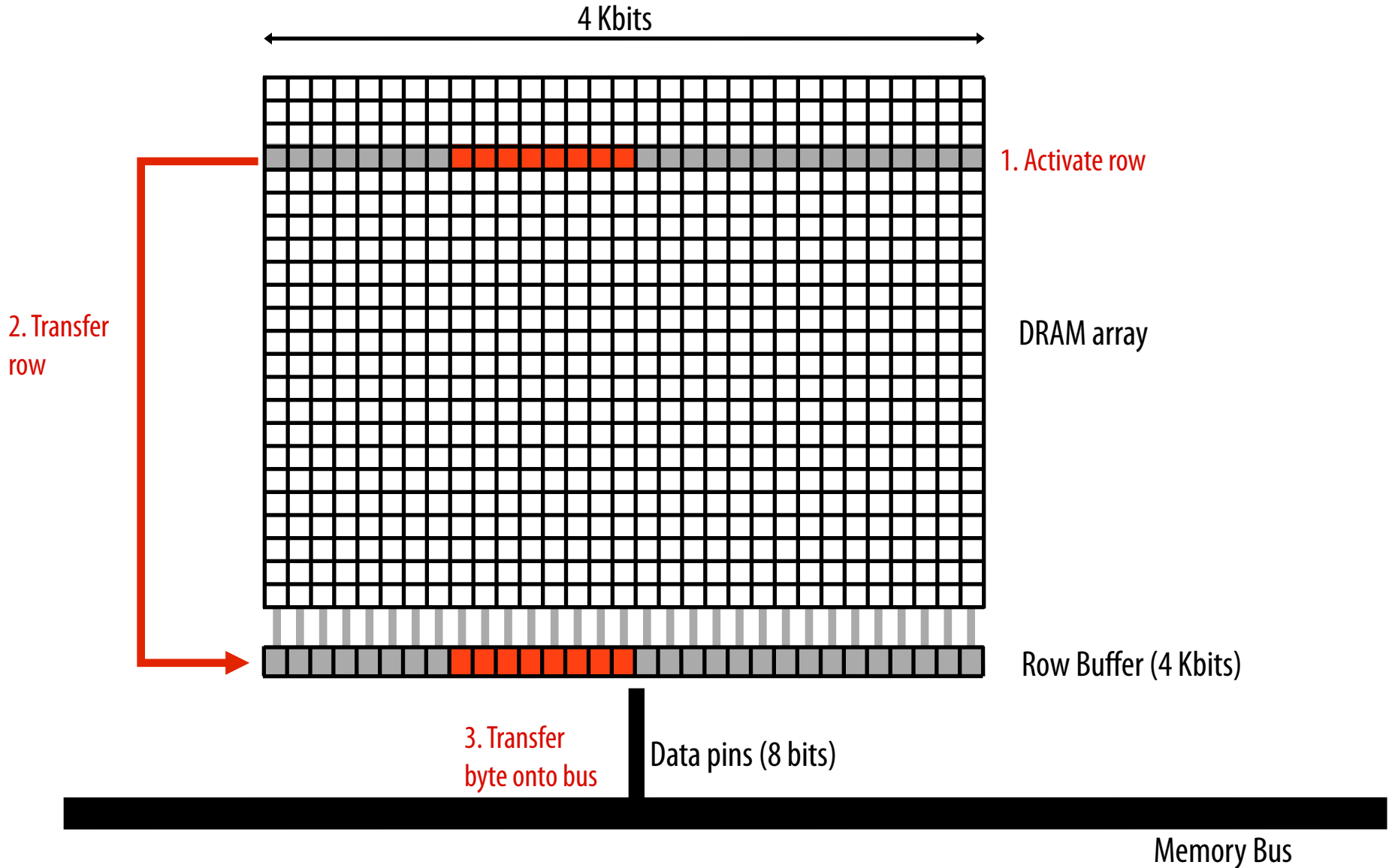
RowClone: in-DRAM Row Copy (and Initialization)

4 Kbits

1. Activate row A

3. Activate row B

DRAM array

2. Transfer row

4. Transfer row

Row Buffer (4 Kbits)

Data pins (8 bits)

Memory Bus
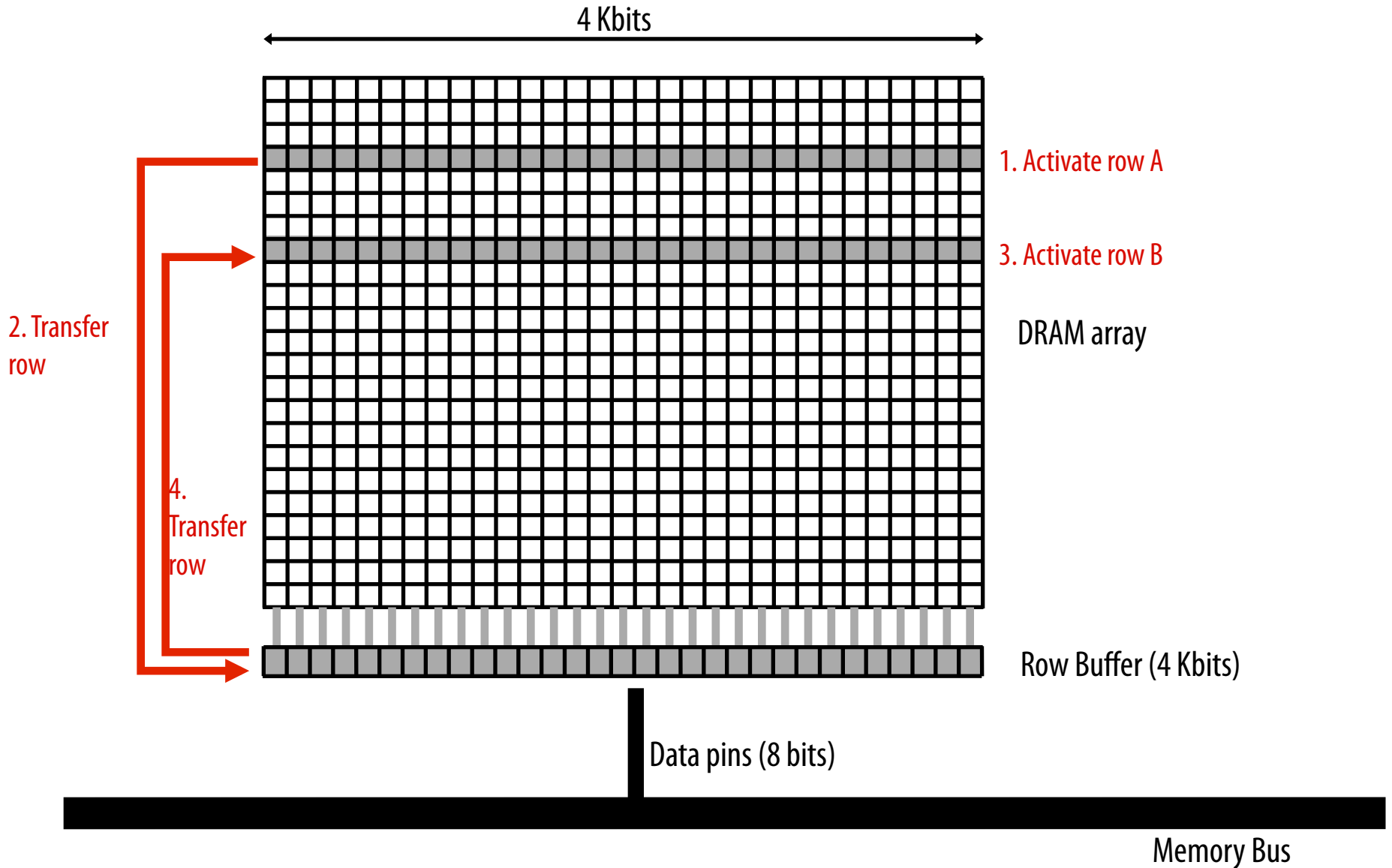
Our Approach: Key Idea

- 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

wordline

DRAM Operation

Raise wordline

?	?	?	?	?	?	?	?	?	?	?	?	src
1	1	0	1	0	1	1	1	0	0	1	1	dst

Sense
Amplifiers
(row buffer)

| - | + | - | - | + | + | - | - | - | + | + | - |
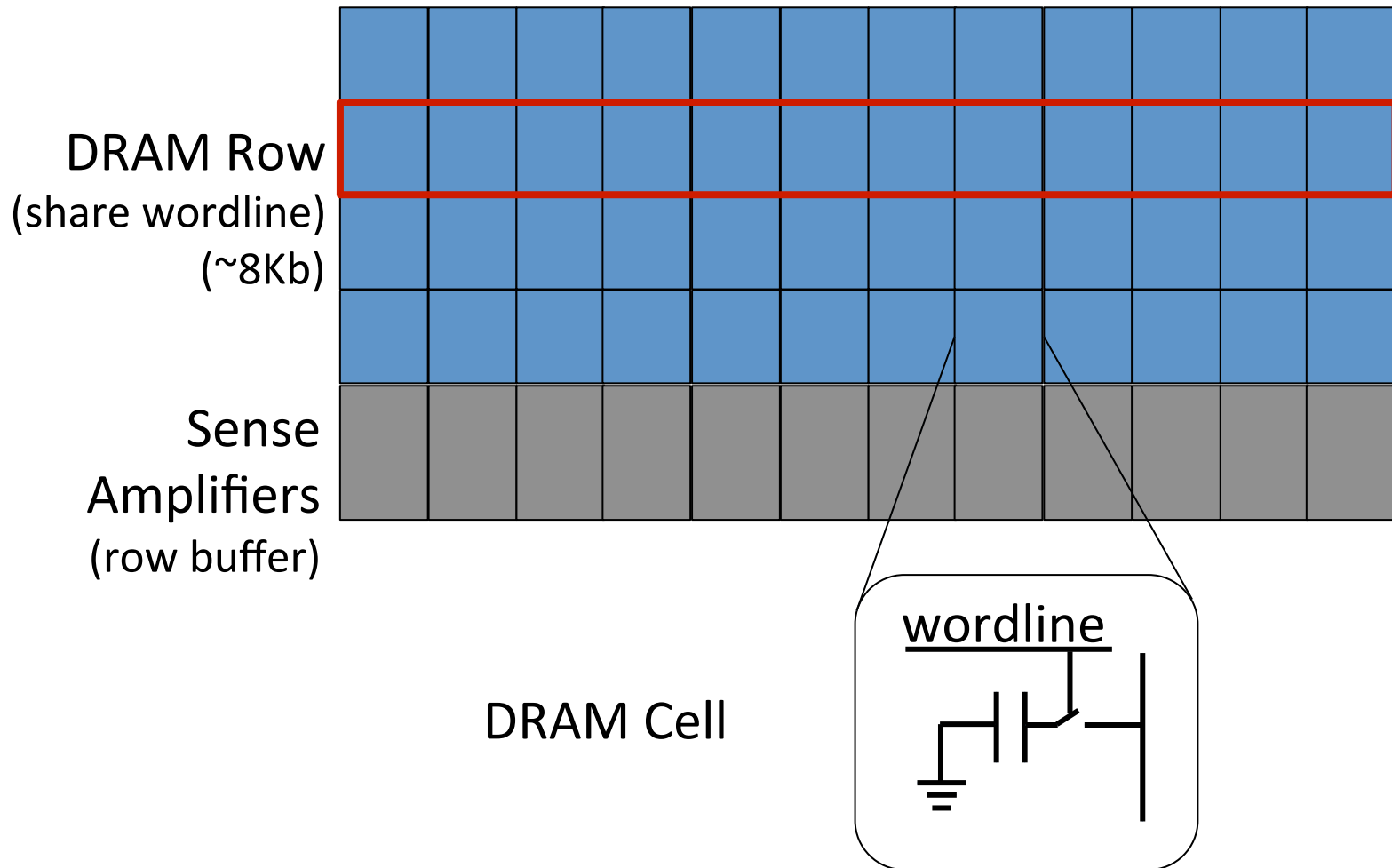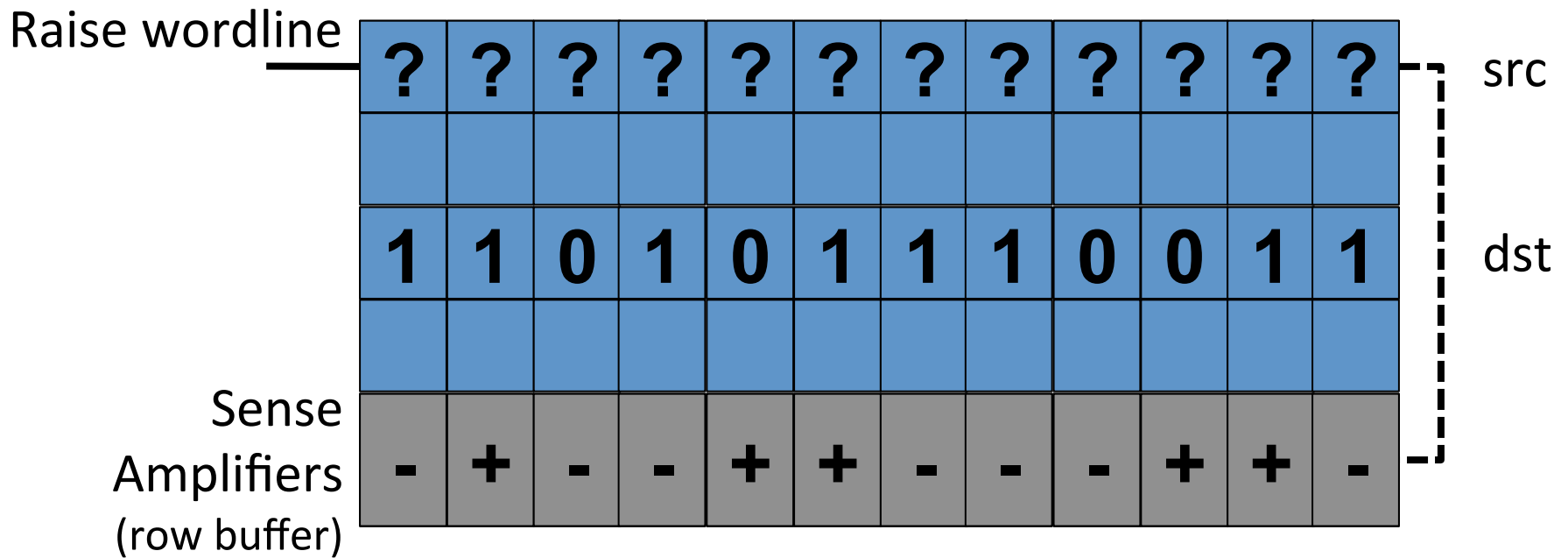
Activate (src) ⟶ Precharge

RowClone: Intra-subarray Copy

0	**1**	**0**	**0**	**1**	**1**	**0**	**0**	**0**	**1**	**1**	**0**	src
0	**1**	**0**	**0**	**1**	**1**	**0**	**0**	**0**	**1**	**1**	**0**	dst
0	**1**	**0**	**0**	**1**	**1**	**0**	**0**	**0**	**1**	**1**	**0**	

Sense
Amplifiers
(row buffer)
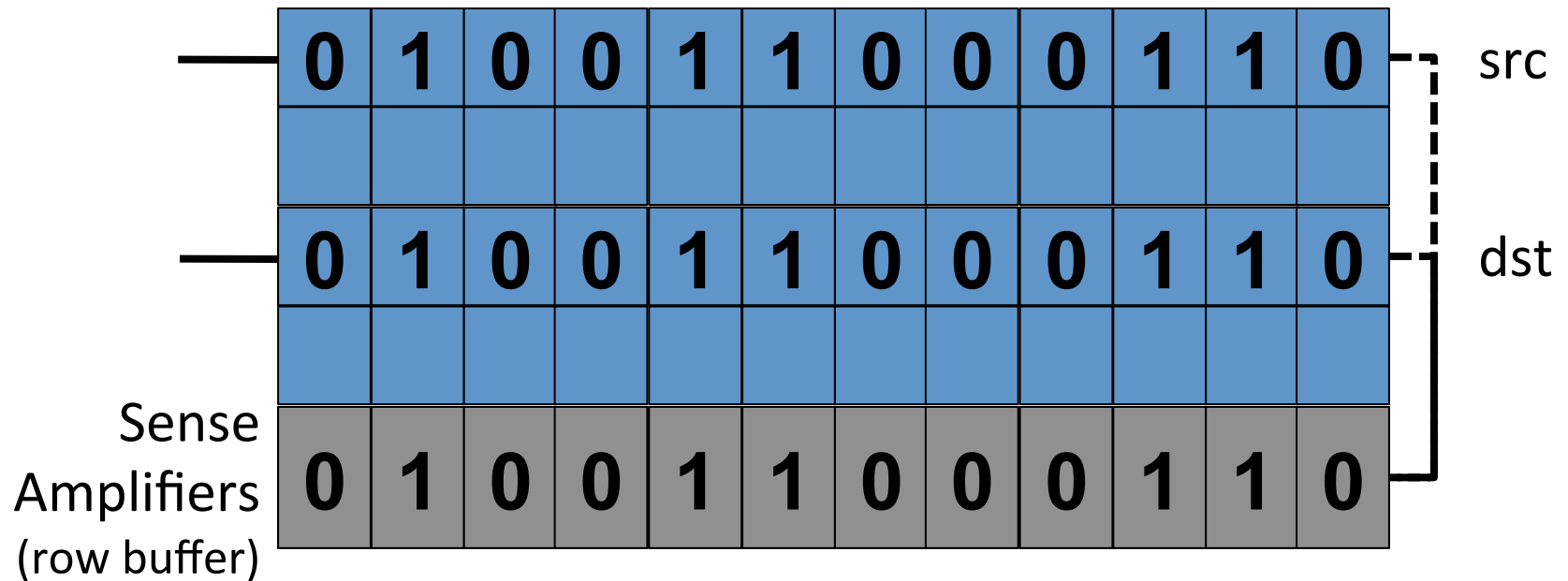
Activate (src) ⟶ Deactivate (our proposal) ⟶ Activate (dst)

RowClone: Inter-bank Copy

src

dst

Read

Write

I/O Bus

Transfer
(our proposal)
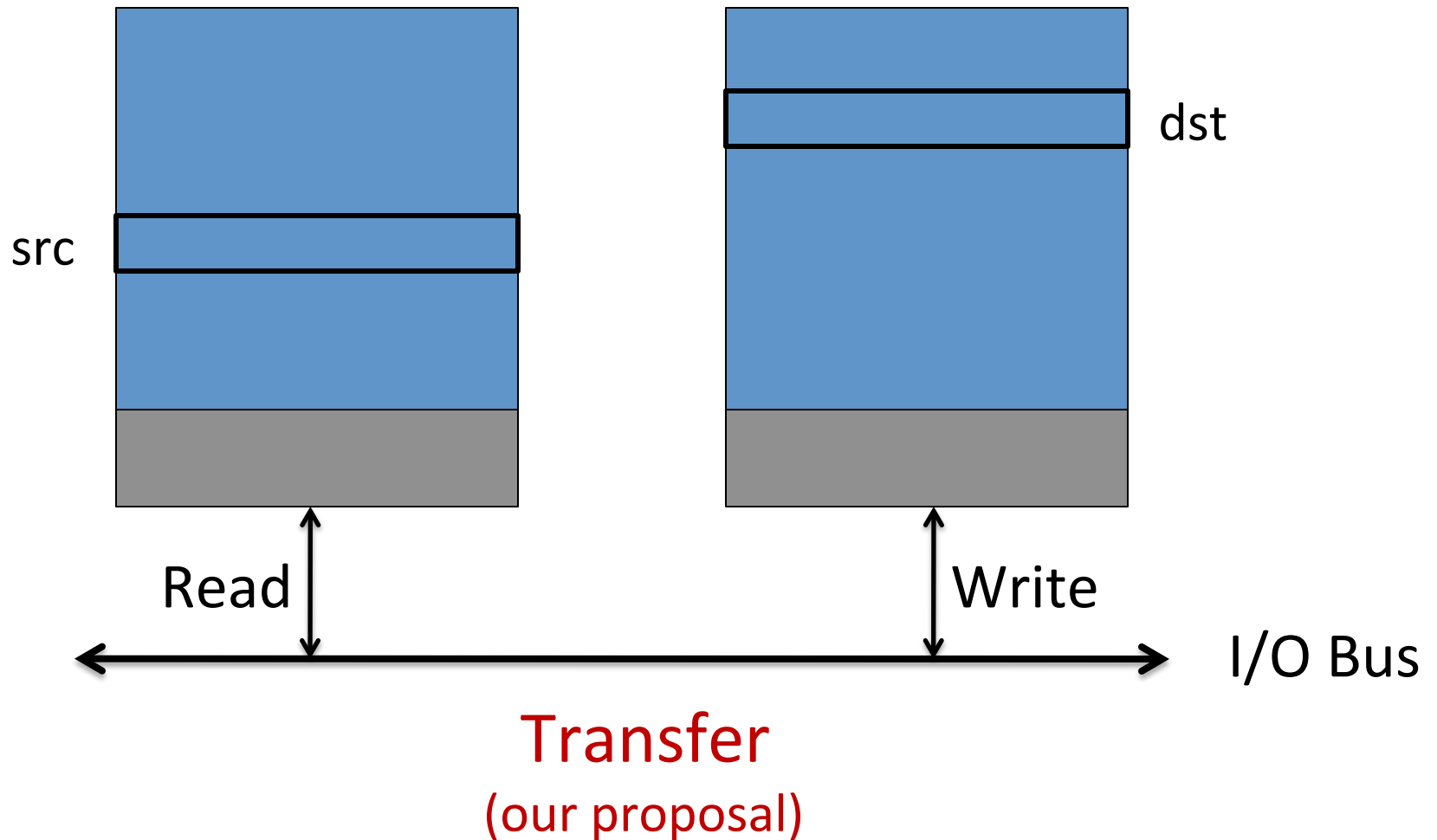
RowClone: Inter-subarray Copy

dst

src

temp

I/O Bus
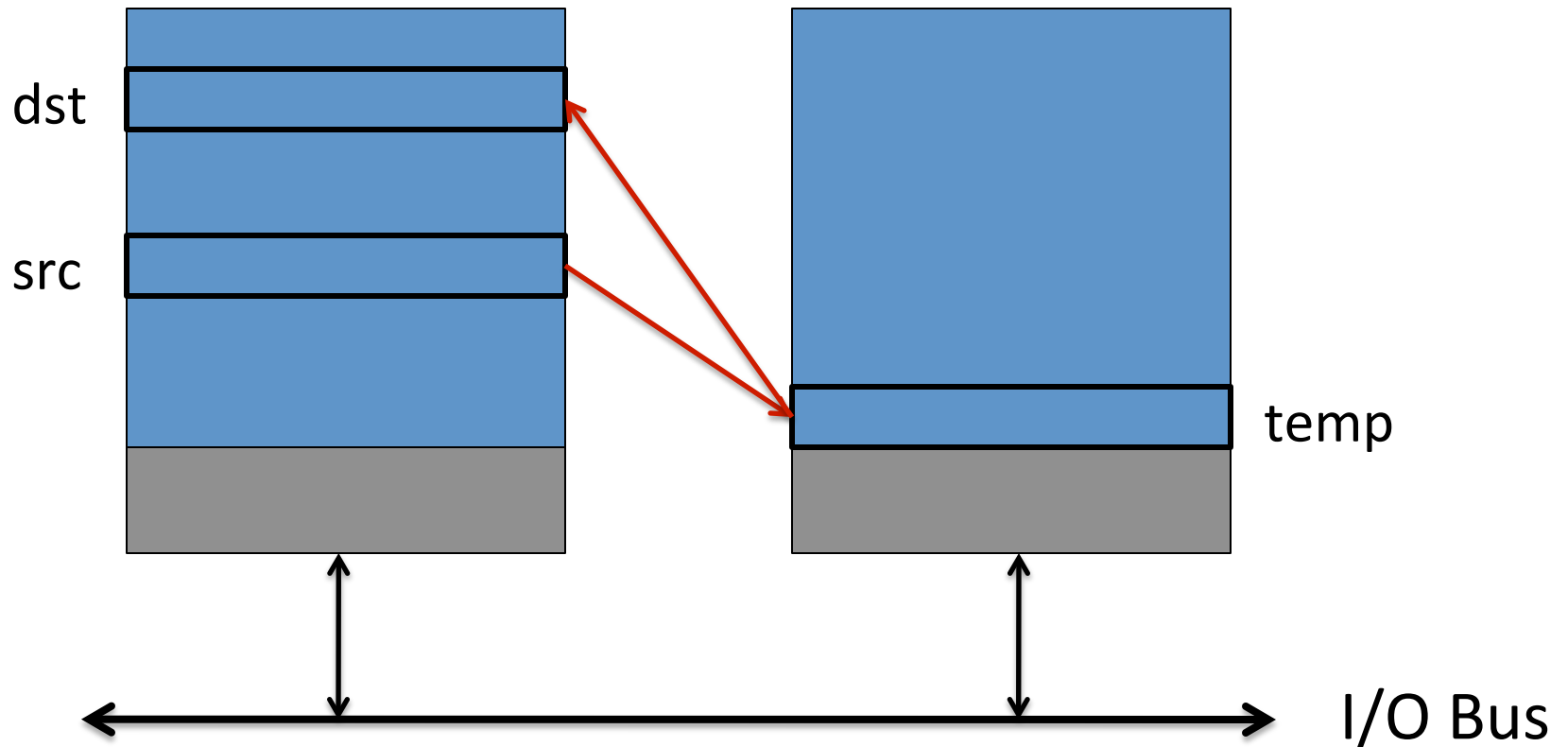
1. Transfer (src to temp)
2. Transfer (temp to dst)

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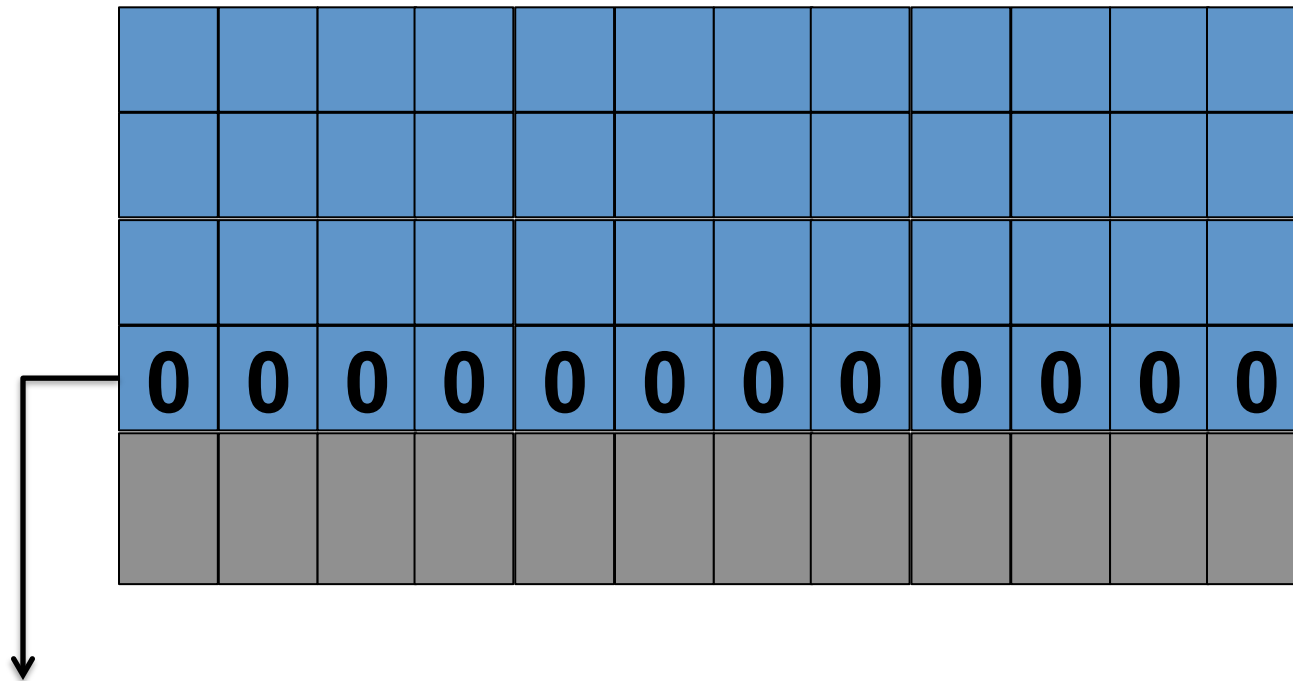
Fix a row at Zero
(0.5% loss in capacity)

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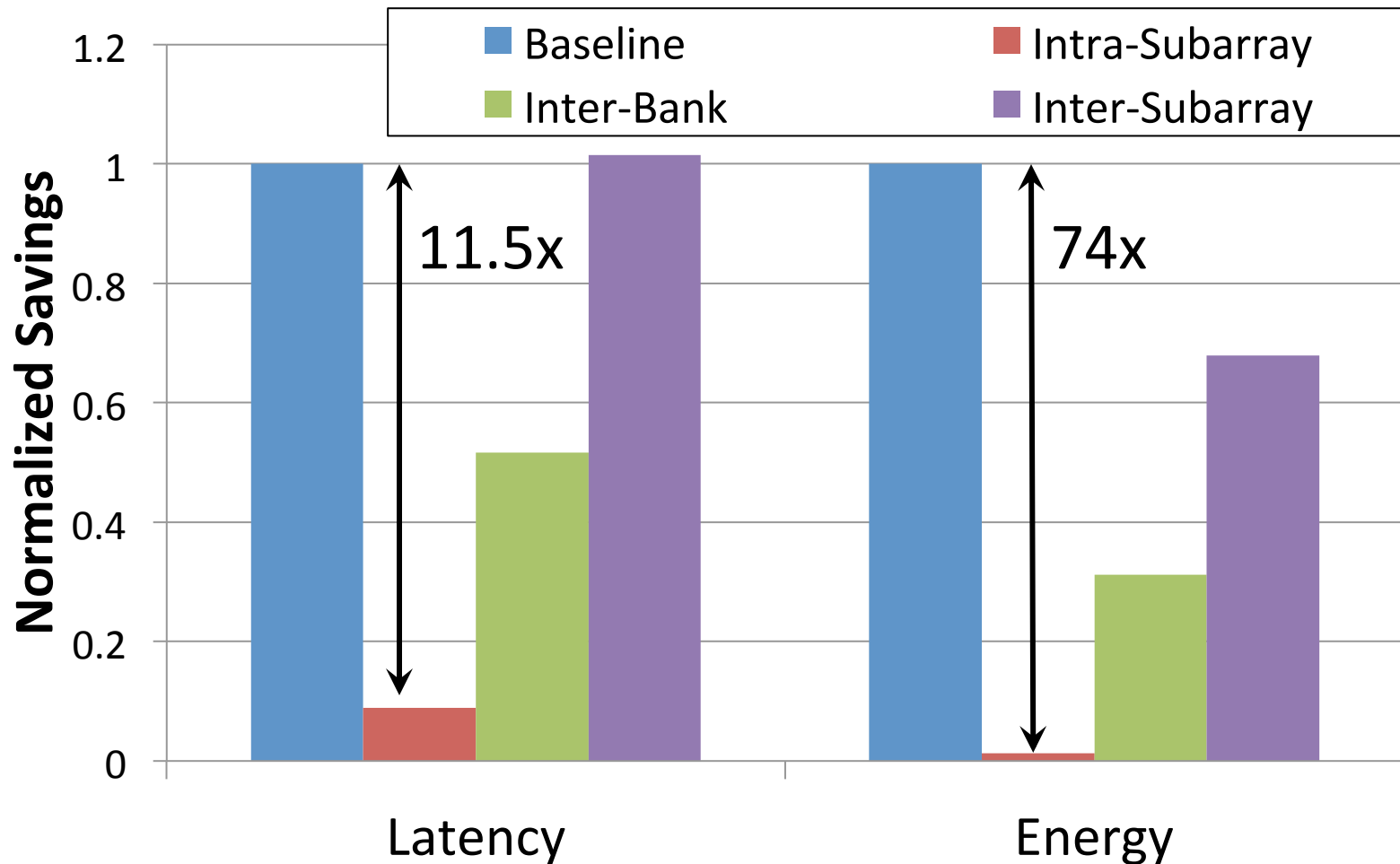
Seshadri et al., "RowClone: Fast and Efficient In-DRAM Copy and Initialization of Bulk Data," CMU Tech Report 2013.

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

SAFARI

Sampling of Ongoing Research

- ## Online retention time profiling

  - Preliminary work in ISCA 2013

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

- ## More computation in memory and controllers

- ## Refresh/demand parallelization

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

SAFARI

Thank you.

Scalable Many-Core Memory Systems Topic 1: DRAM Basics and DRAM Scaling

Prof. Onur Mutlu

http://www.ece.cmu.edu/~omutlu

onur@cmu.edu

HiPEAC ACACES Summer School 2013

July 15-19, 2013

Carnegie Mellon

Additional Material

Three Papers

- Howard David, Chris Fallin, Eugene Gorbatov, Ulf R. Hanebutte, and <u>Onur Mutlu</u>,
"Memory Power Management via Dynamic Voltage/Frequency Scaling"
Proceedings of the <u>8th International Conference on Autonomic Computing</u>
(**ICAC**), Karlsruhe, Germany, June 2011. <u>Slides (pptx)</u> <u>(pdf)</u>

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

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

Memory Power Management via Dynamic Voltage/Frequency Scaling

Howard David (Intel)
Eugene Gorbatov (Intel)
Ulf R. Hanebutte (Intel)
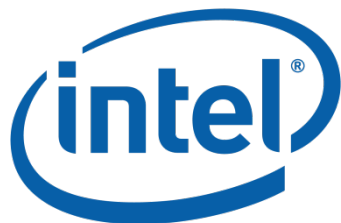
Chris Fallin (CMU)
Onur Mutlu (CMU)

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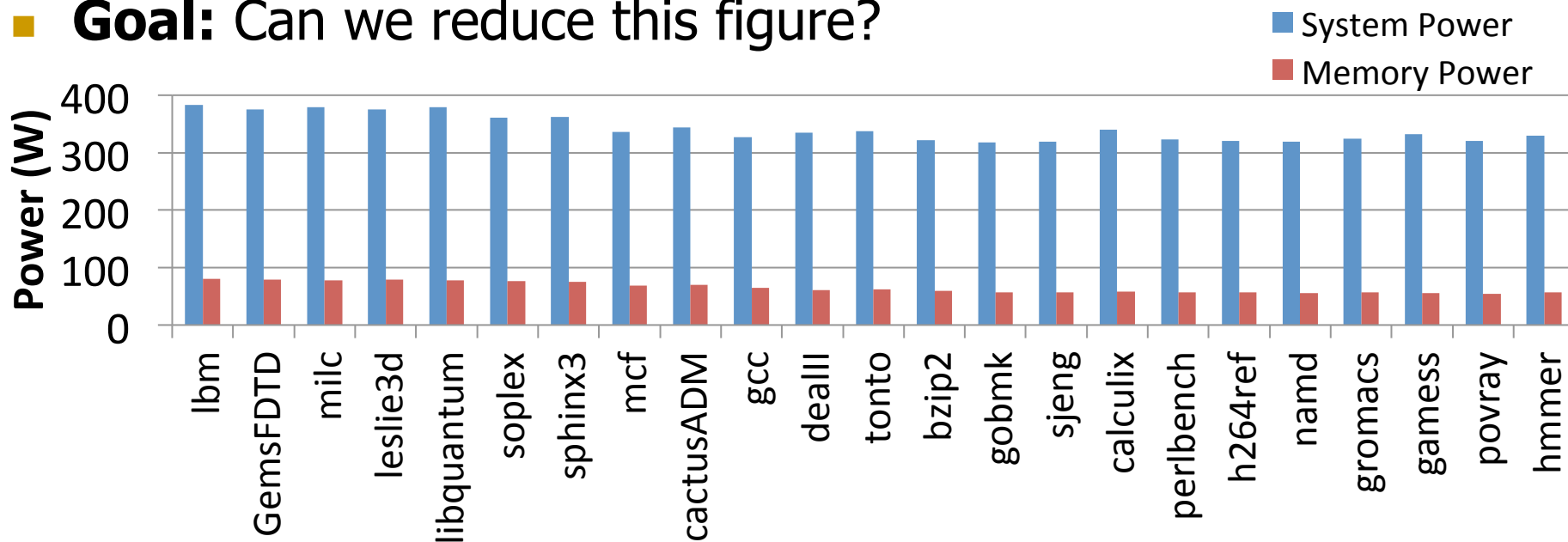
*Dual 4-core Intel Xeon®, 48GB DDR3 (12 DIMMs), SPEC CPU2006, all cores active.
Measured AC power, analytically modeled memory power.

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

Sleep State Residency
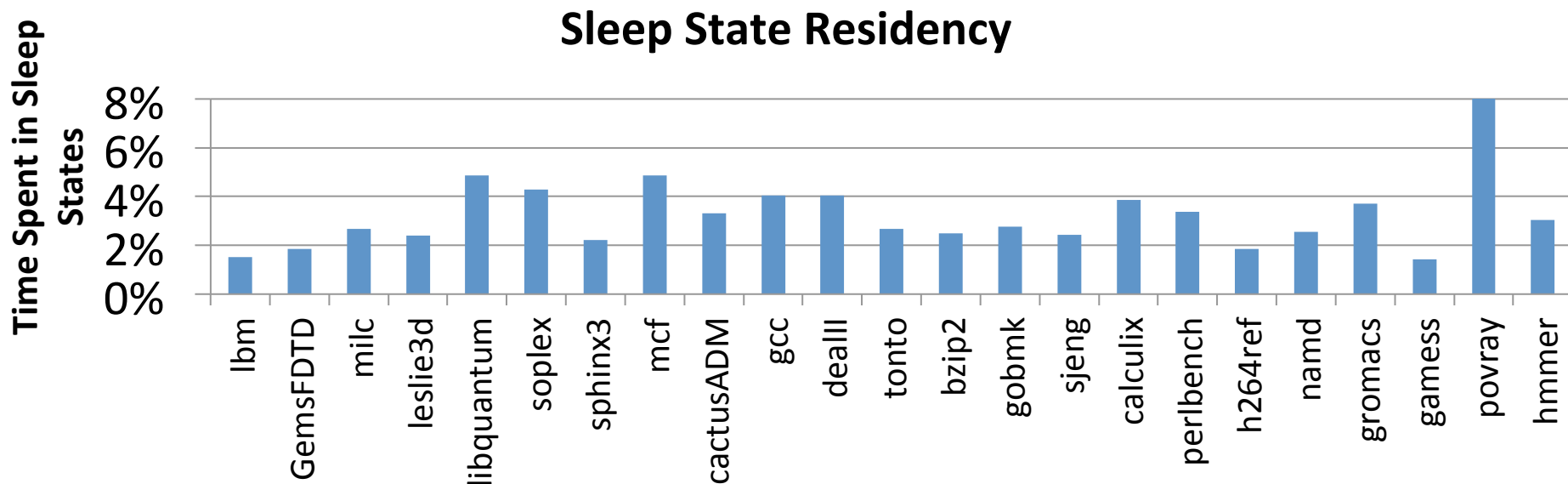
Memory Bandwidth Varies Widely

- Workload memory bandwidth requirements vary widely

Memory Bandwidth for SPEC CPU2006
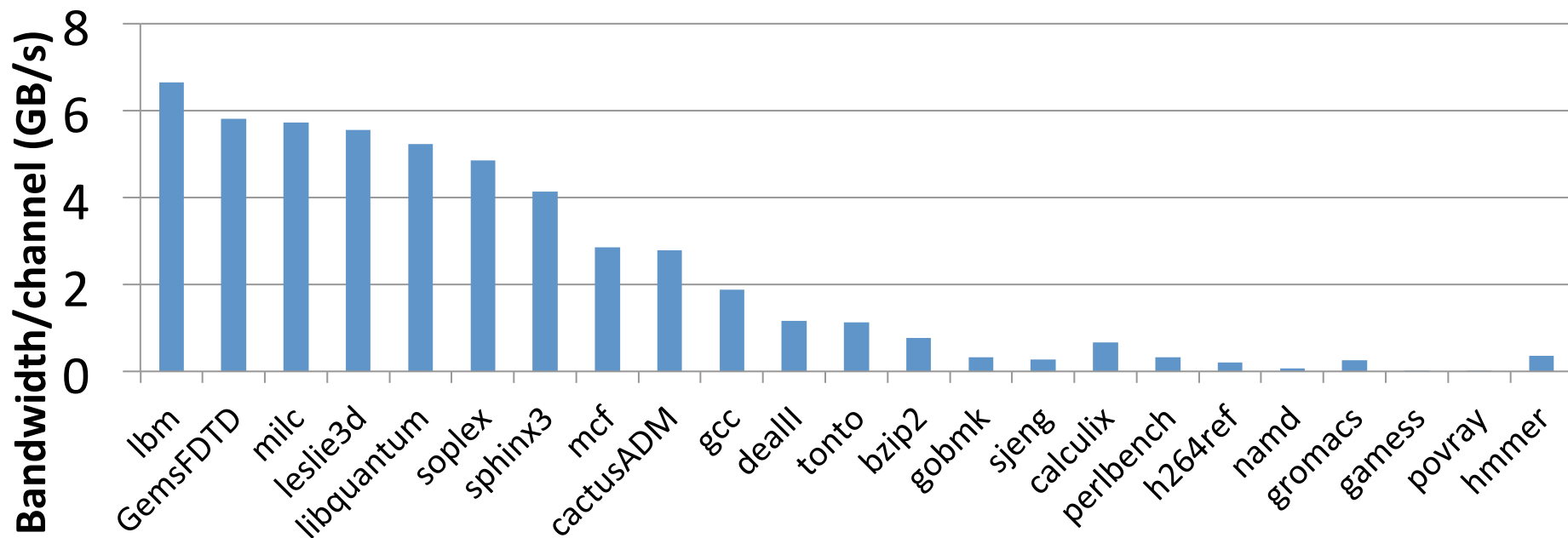
- Memory system is provisioned for peak capacity
  - → often underutilized

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

Memory Bus (64 bits)

to Memory Controller

Inside a DRAM Device

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

Banks
- Independent arrays

ODT

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

Column Decoder

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:

Minimum Stable Voltage for 8 DIMMs in a Real System
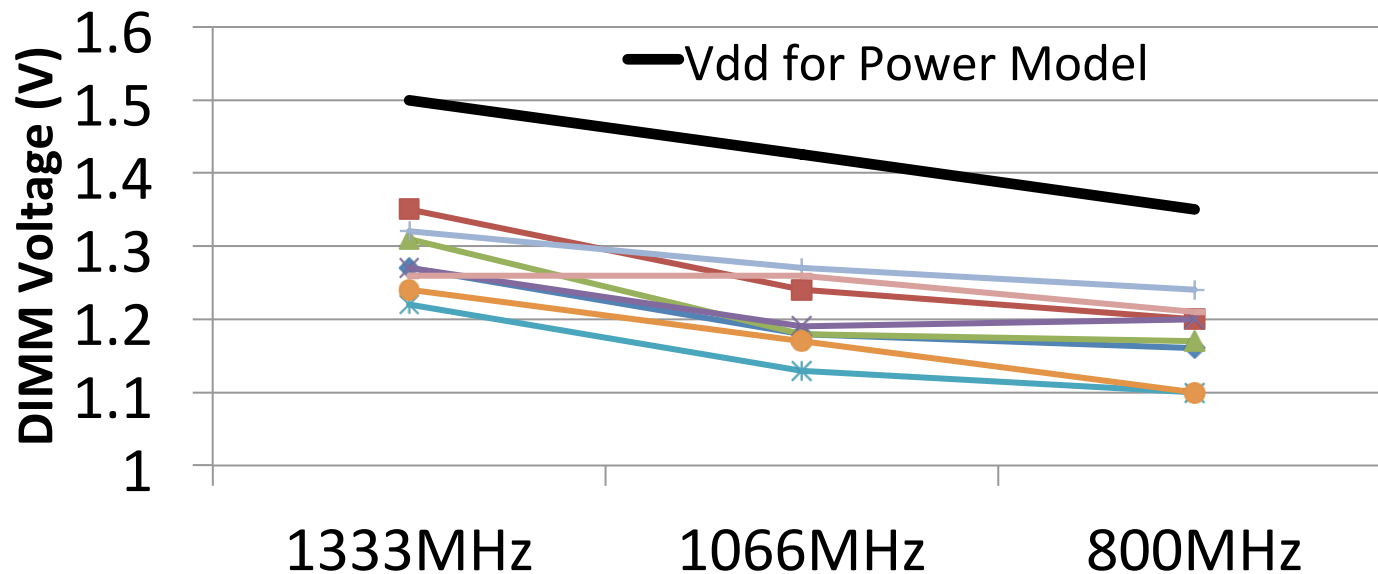
Outline

- **Motivation**

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

- **Performance Effects of Frequency Scaling**

- **Frequency Control Algorithm**

- **Evaluation and Conclusions**

How Much Memory Bandwidth is Needed?

Memory Bandwidth for SPEC CPU2006
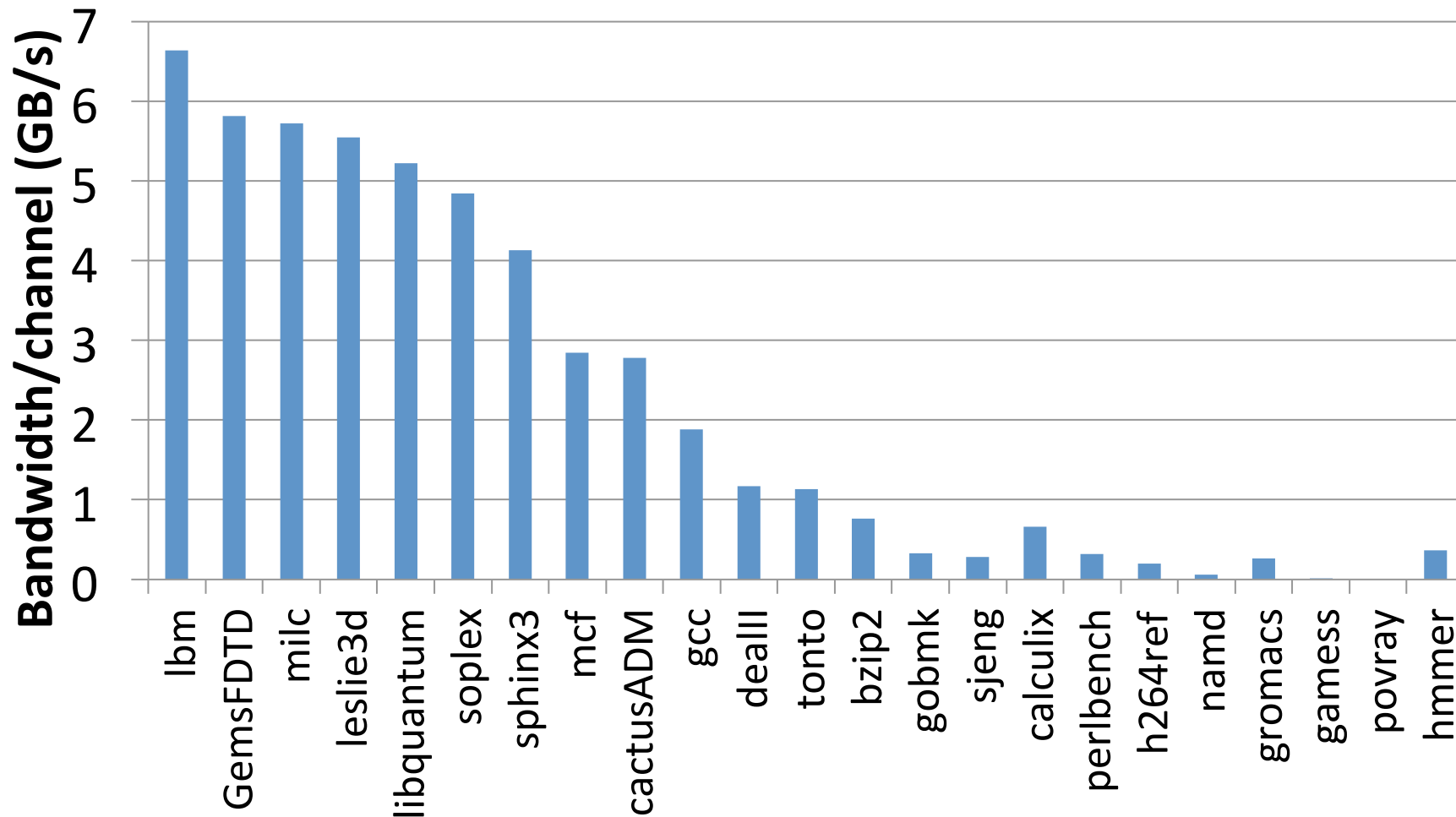
Performance Impact of Static Frequency Scaling

- Performance impact is proportional to bandwidth demand
- Many workloads tolerate lower frequency with minimal performance drop
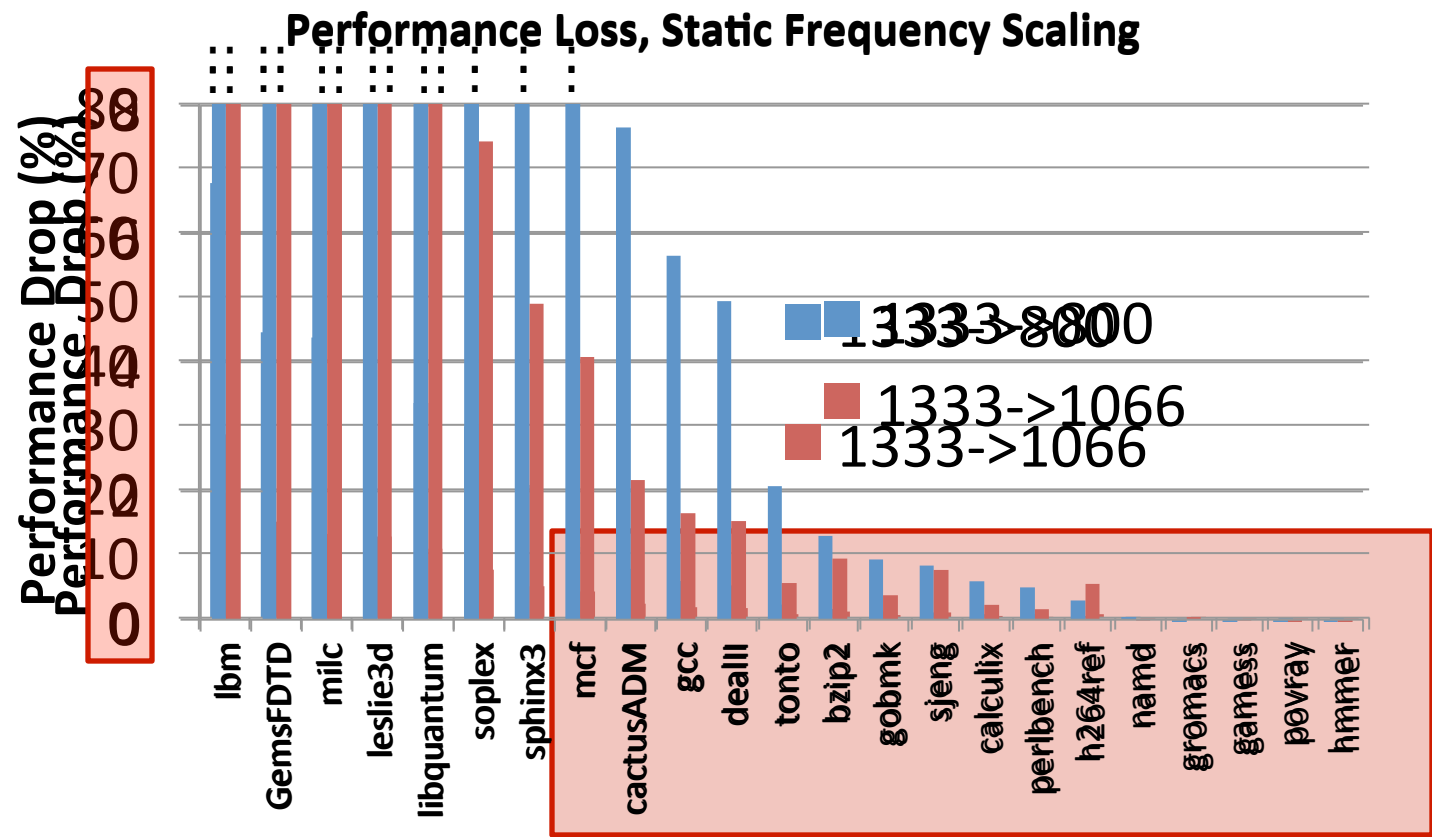
Performance Loss, Static Frequency Scaling

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

Memory Latency as a Function of Bandwidth and Mem Frequency
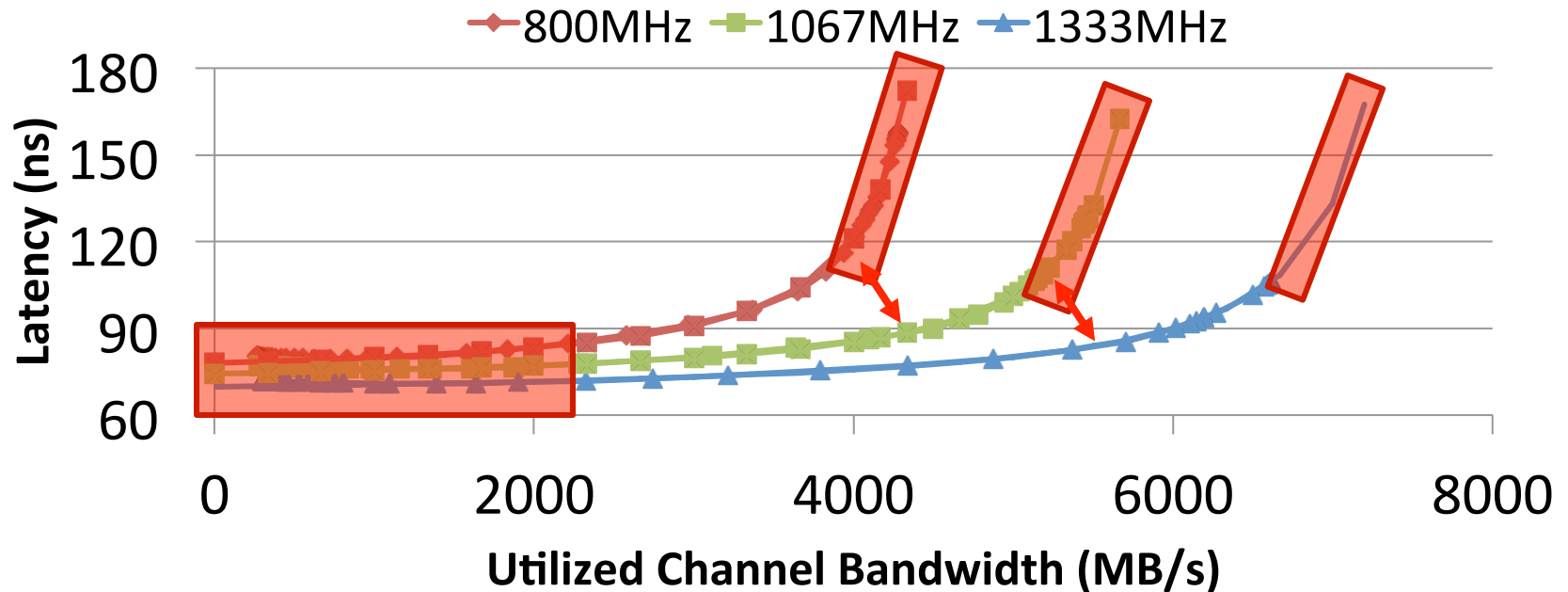
Control Algorithm: Demand-Based Switching

Memory Latency as a Function of Bandwidth and Mem Frequency
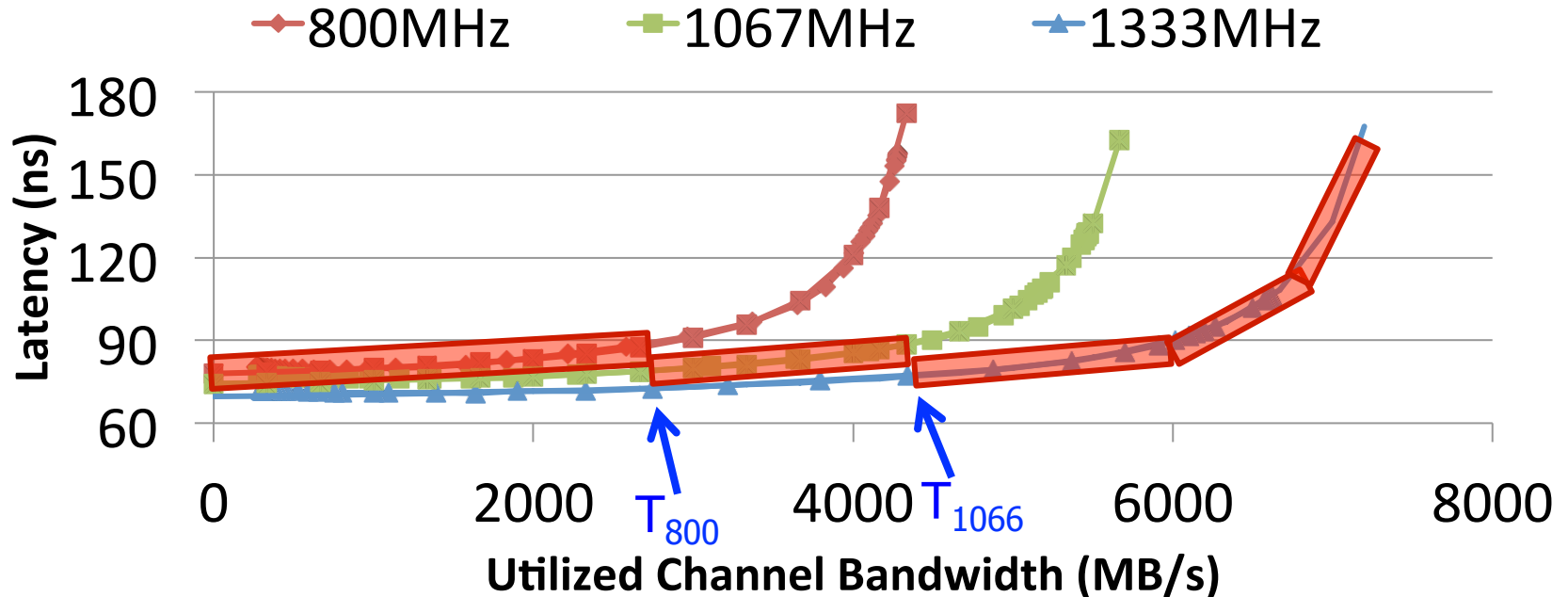
After each epoch of length T_{epoch}:

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

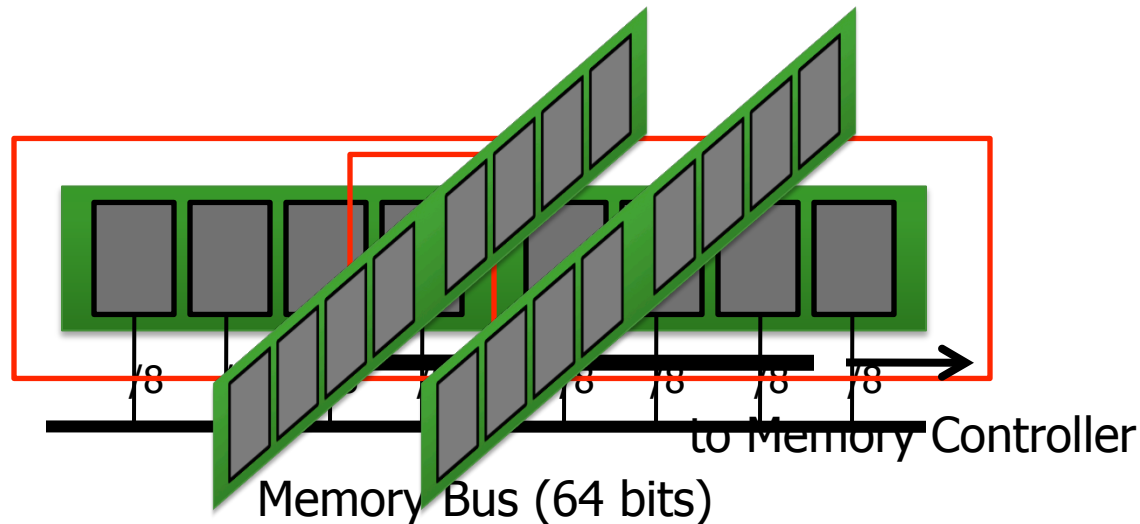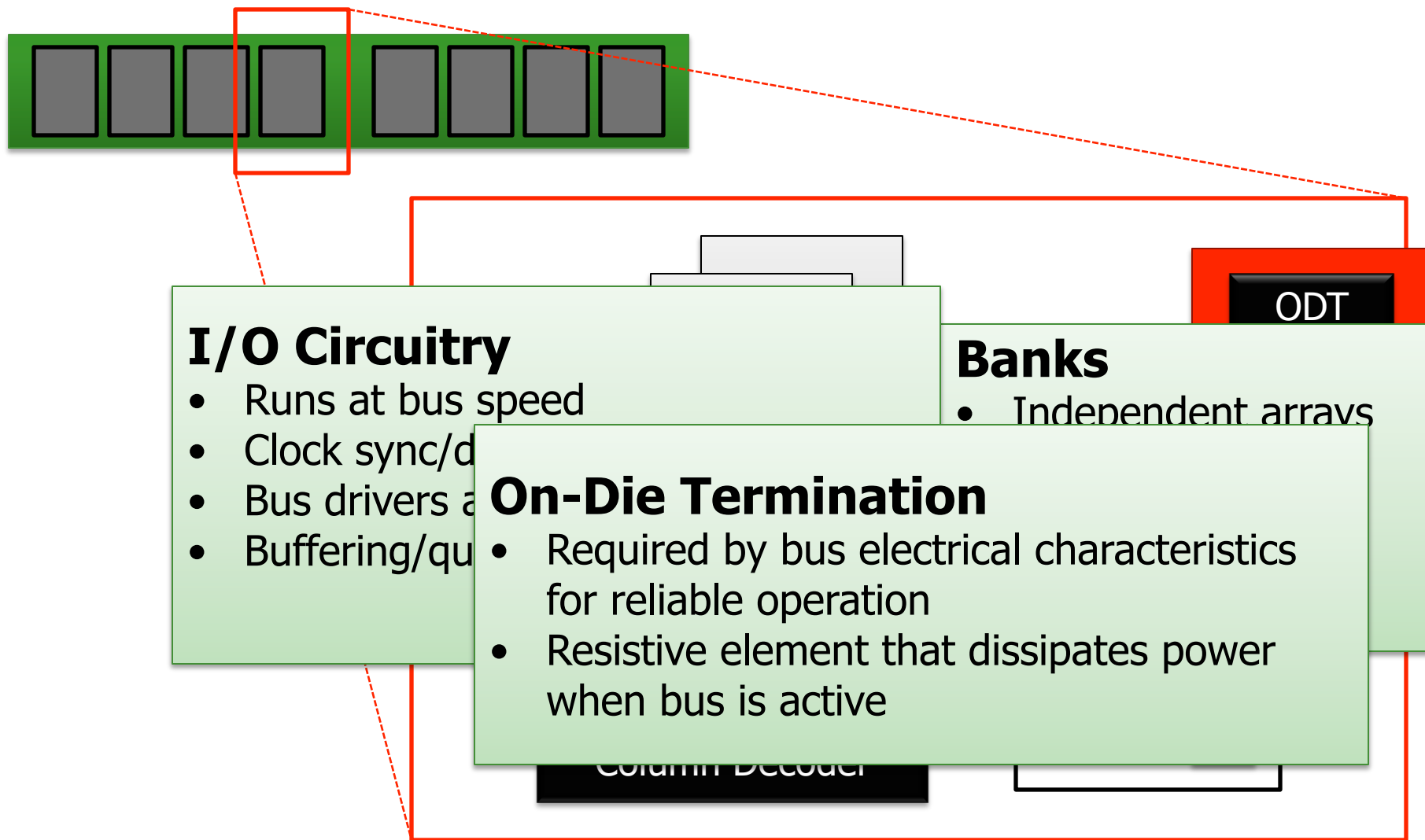👍 Full transition takes ~20μs

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%
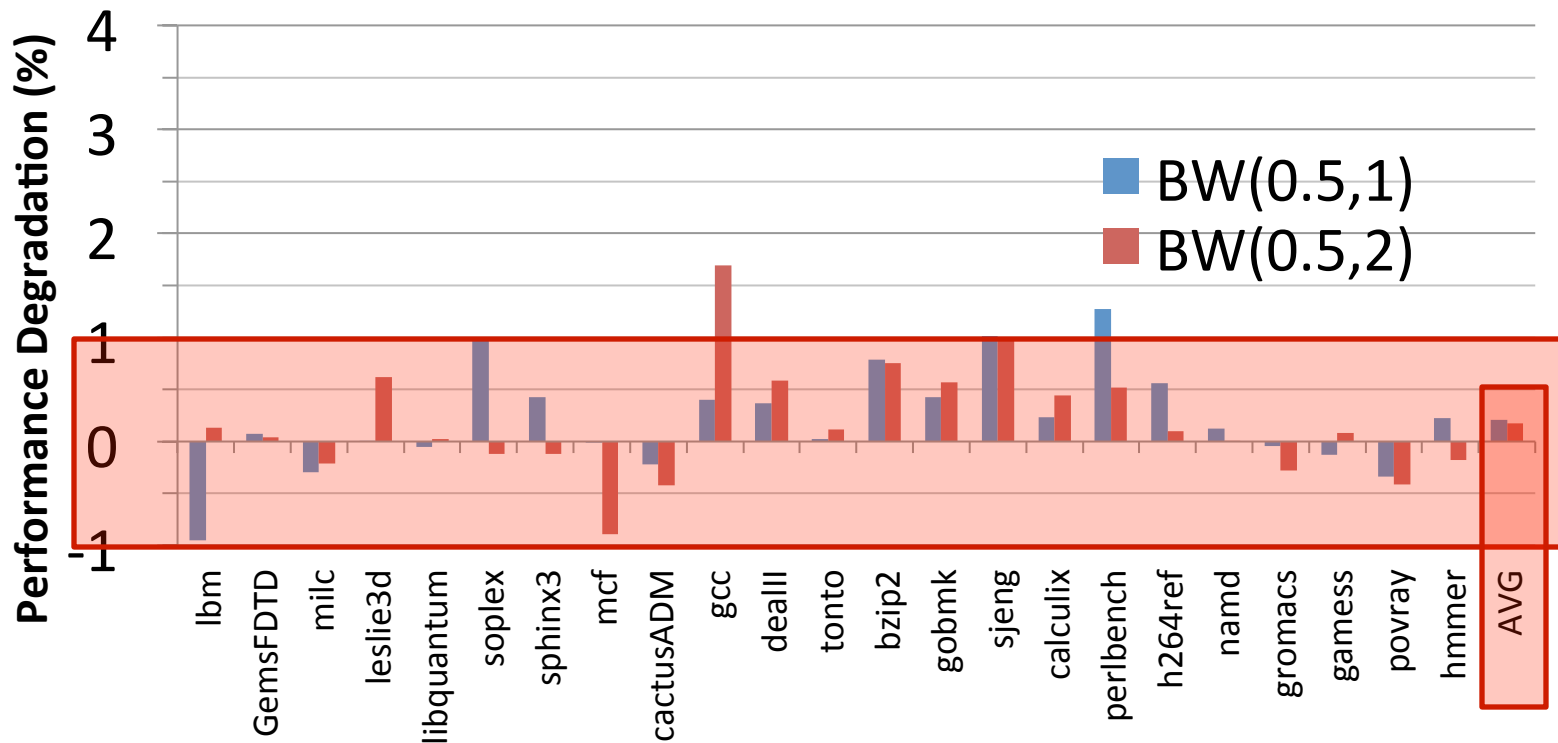
Memory Frequency Distribution

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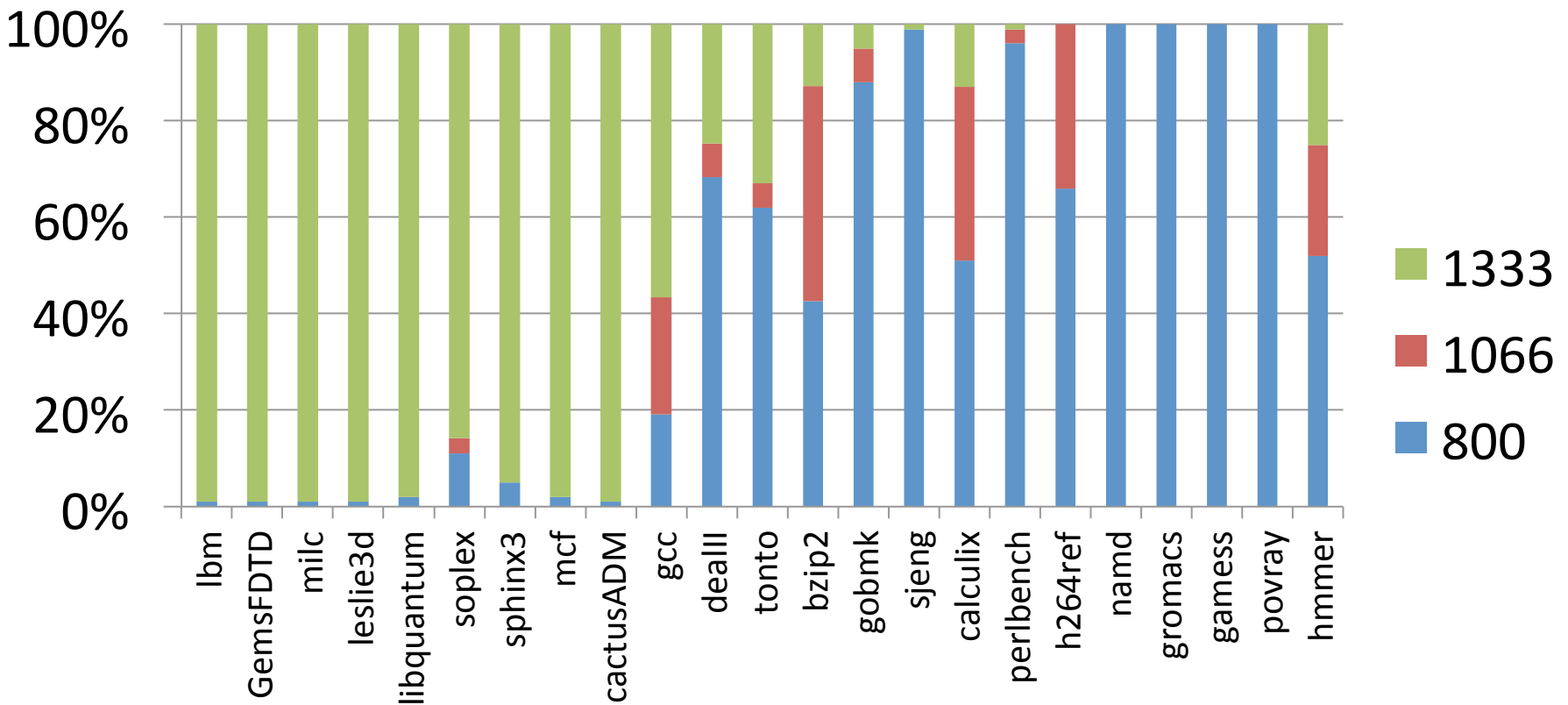
Memory Power Reduction

■ Memory power reduces by 10.4% (avg), 20.5% (max)
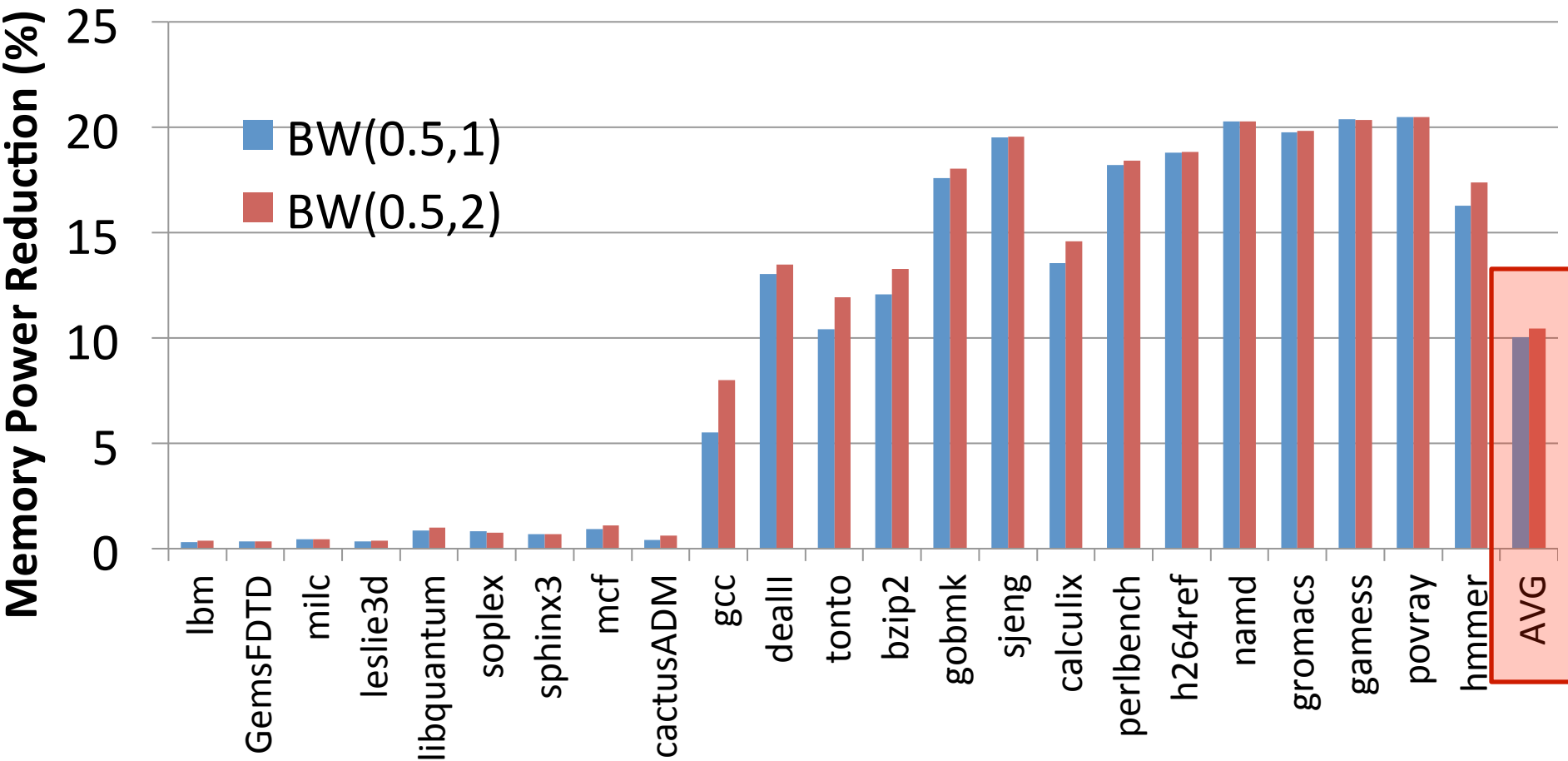
System Power Reduction

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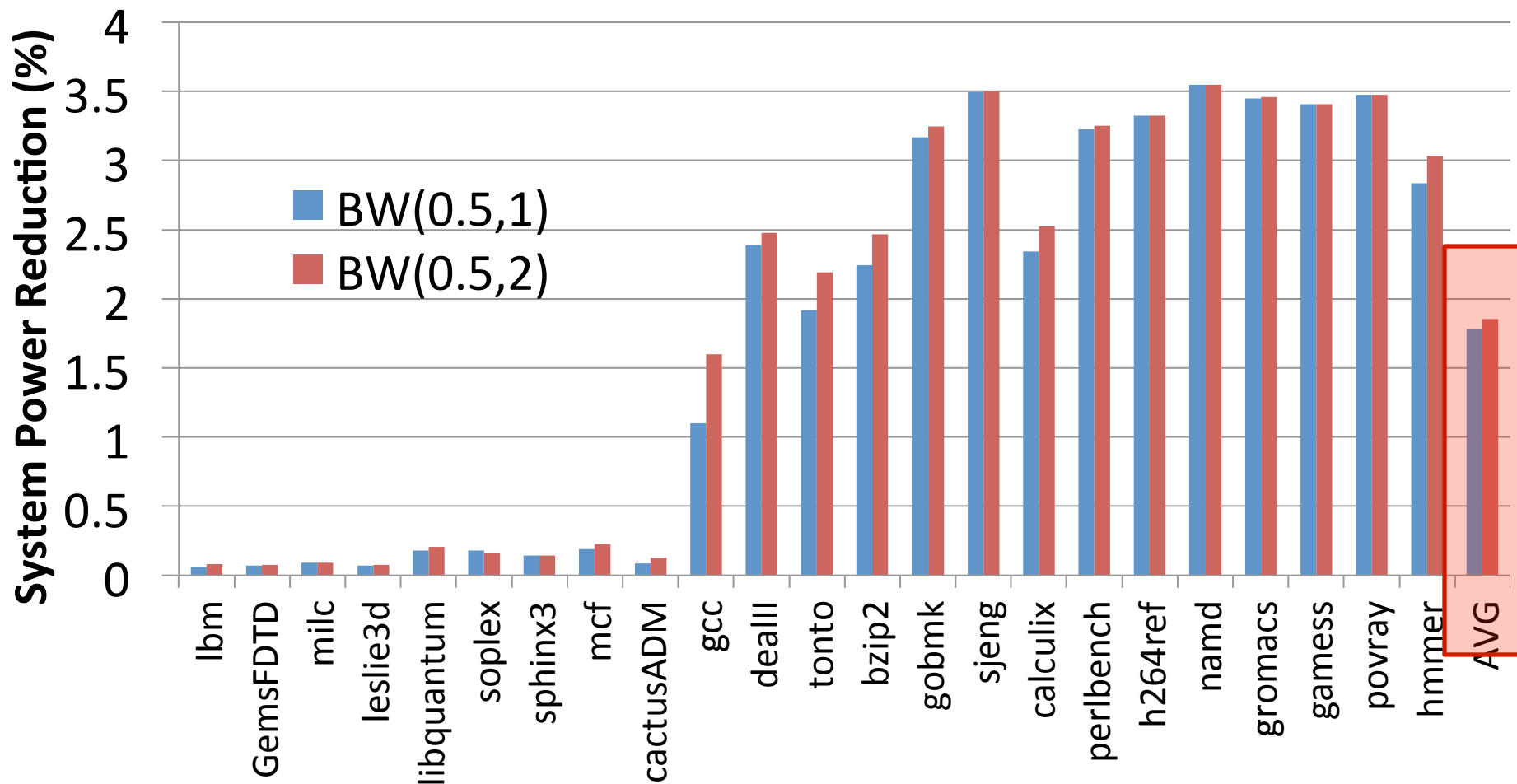
System Energy Reduction

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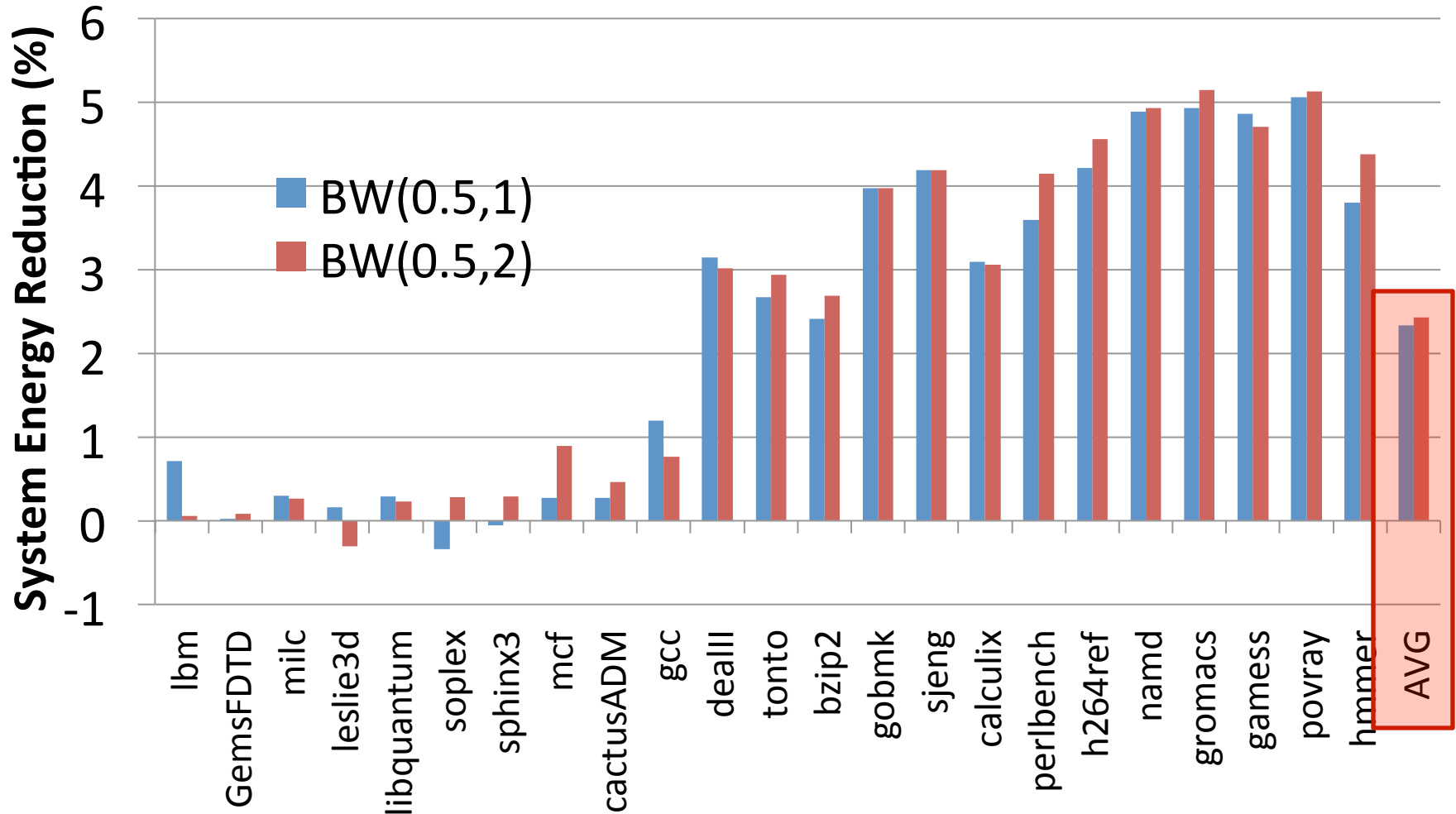
Related Work

- **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)

An Experimental Study of Data Retention Behavior in Modern DRAM Devices

Implications for Retention Time Profiling Mechanisms

Jamie Liu[1]   Ben Jaiyen[1]   Yoongu Kim[1]

Chris Wilkerson[2]   Onur Mutlu[1]

[1] Carnegie Mellon University
[2] Intel Corporation
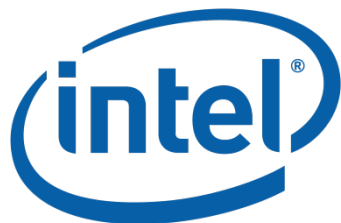
SAFARI

Carnegie Mellon

Summary (I)

- DRAM requires periodic refresh to avoid data loss
  - Refresh wastes energy, reduces performance, limits DRAM density scaling
- Many past works observed that different DRAM cells can retain data for different times without being refreshed; proposed reducing refresh rate for strong DRAM cells
  - Problem: These techniques require an accurate profile of the retention time of all DRAM cells
- Our goal: To analyze the retention time behavior of DRAM cells in modern DRAM devices to aid the collection of accurate profile information
- Our experiments: We characterize 248 modern commodity DDR3 DRAM chips from 5 manufacturers using an FPGA based testing platform
- Two Key Issues:

  1. Data Pattern Dependence: A cell's retention time is heavily dependent on data values stored in itself and nearby cells, which cannot easily be controlled.

  2. Variable Retention Time: Retention time of some cells change unpredictably from high to low at large timescales.

SAFARI

Summary (II)

- **Key findings on Data Pattern Dependence**
  - There is no observed single data pattern that elicits the lowest retention times for a DRAM device → very hard to find this pattern
  - DPD varies between devices due to variation in DRAM array circuit design between manufacturers
  - DPD of retention time gets worse as DRAM scales to smaller feature sizes

- **Key findings on Variable Retention Time**
  - VRT is common in modern DRAM cells that are weak
  - The timescale at which VRT occurs is very large (e.g., a cell can stay in high retention time state for a day or longer) → finding minimum retention time can take very long

- Future work on retention time profiling must address these issues

Talk Agenda

- **DRAM Refresh: Background and Motivation**
- Challenges and Our Goal
- DRAM Characterization Methodology
- Foundational Results
  - Temperature Dependence
  - Retention Time Distribution
- Data Pattern Dependence: Analysis and Implications
- Variable Retention Time: Analysis and Implications
- Conclusions

A DRAM Cell

wordline (row enable)

bitline   bitline   bitline   bitline
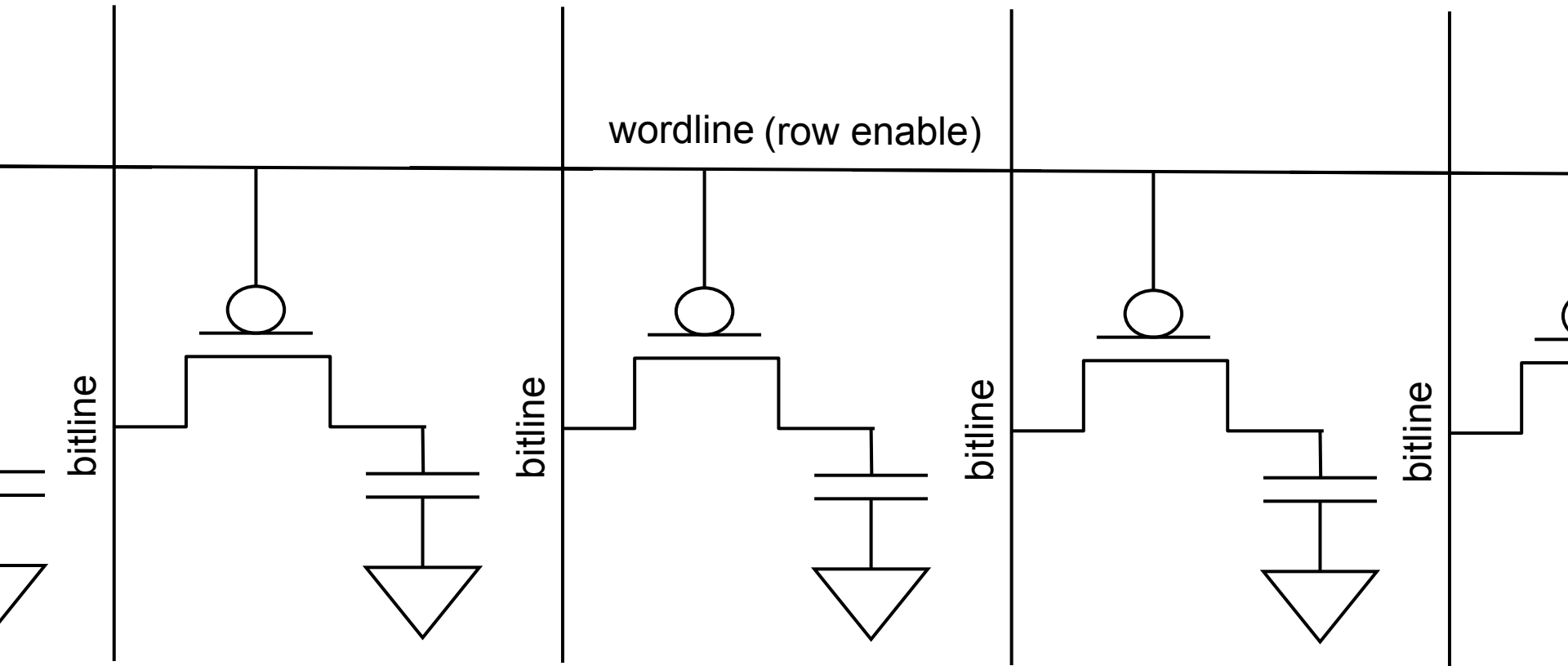
- A DRAM cell consists of a capacitor and an access transistor
- It stores data in terms of charge in the capacitor
- A DRAM chip consists of (10s of 1000s of) rows of such cells

DRAM Refresh

- DRAM capacitor charge leaks over time

- Each DRAM row is periodically refreshed to restore charge
    - Activate 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 capacity scaling

Refresh Overhead: Performance

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

Refresh Overhead: Energy

Liu et al., "RAIDR: Retention-Aware Intelligent DRAM Refresh," ISCA 2012.
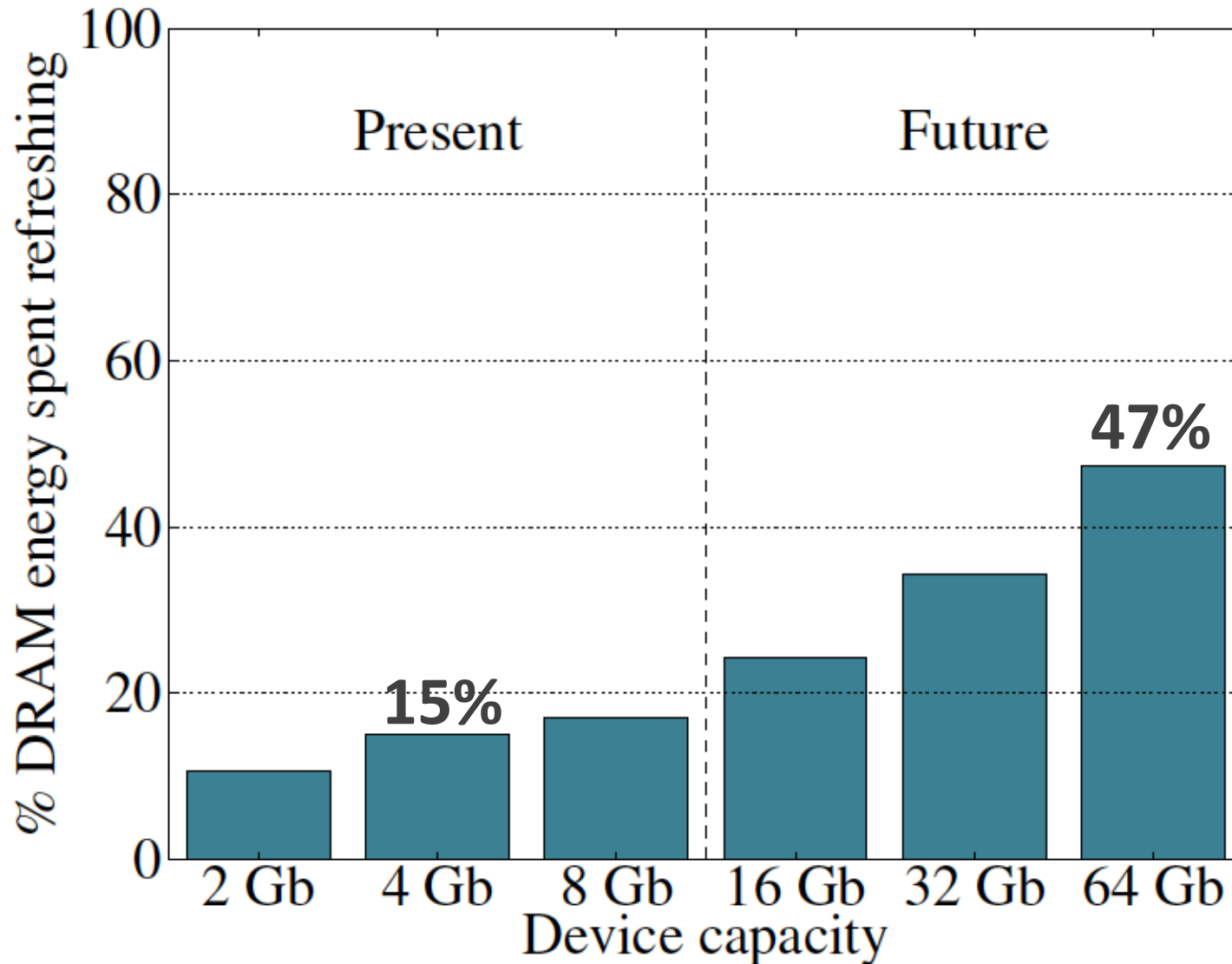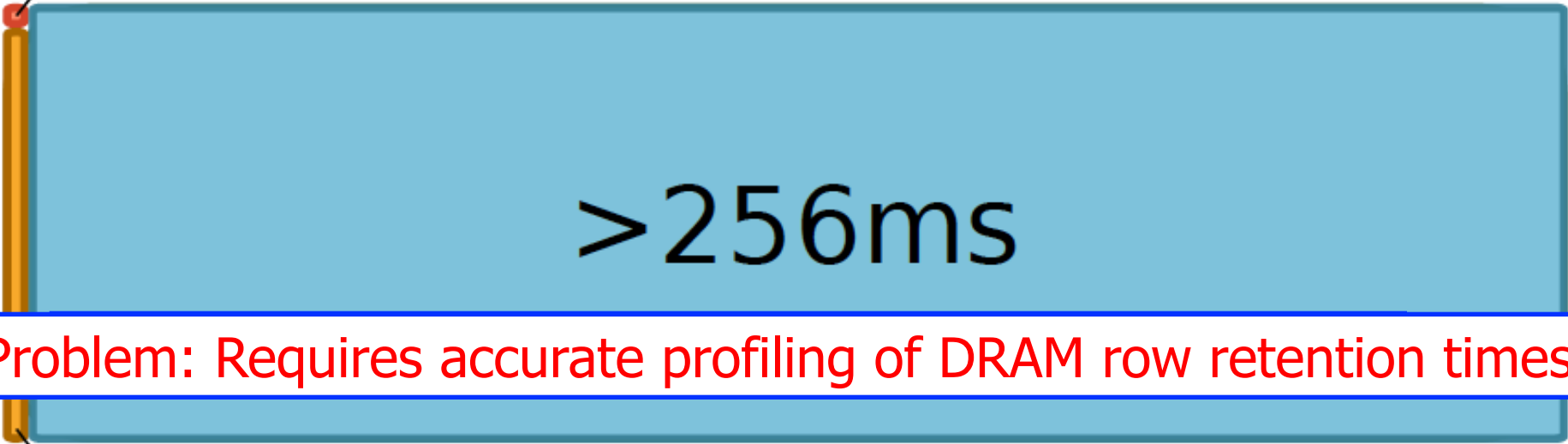
Previous Work on Reducing Refreshes

- **Observed significant variation in data retention times of DRAM cells** (due to manufacturing process variation)
  - **Retention time:** maximum time a cell can go without being refreshed while maintaining its stored data

- **Proposed methods to take advantage of widely varying retention times among DRAM rows**
  - Reduce refresh rate for rows that can retain data for longer than 64 ms, e.g., **[Liu+ ISCA 2012]**
  - Disable rows that have low retention times, e.g., **[Venkatesan+ HPCA 2006]**

- **Showed large benefits in energy and performance**

SAFARI

An Example: RAIDR [Liu+, ISCA 2012]

64-128ms

>256ms

Problem: Requires accurate profiling of DRAM row retention times

128-256ms

Can reduce refreshes by ~75%
→ reduces energy consumption and improves performance

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

Motivation

- Past works require accurate and reliable measurement of retention time of each DRAM row

  - To maintain data integrity while reducing refreshes

- Assumption: worst-case retention time of each row can be determined and stays the same at a given temperature

  - Some works propose writing all 1's and 0's to a row, and measuring the time before data corruption

- Question:

  - Can we reliably and accurately determine retention times of all DRAM rows?

Talk Agenda

- DRAM Refresh: Background and Motivation
- Challenges and Our Goal
- DRAM Characterization Methodology
- Foundational Results
    - Temperature Dependence
    - Retention Time Distribution
- Data Pattern Dependence: Analysis and Implications
- Variable Retention Time: Analysis and Implications
- Conclusions

Two Challenges to Retention Time Profiling

- **Data Pattern Dependence (DPD)** of retention time

- **Variable Retention Time (VRT)** phenomenon

Two Challenges to Retention Time Profiling

- **Challenge 1: Data Pattern Dependence (DPD)**
  - Retention time of a DRAM cell depends on its value and the values of cells nearby it

  - When a row is activated, all bitlines are perturbed simultaneously

SAFARI
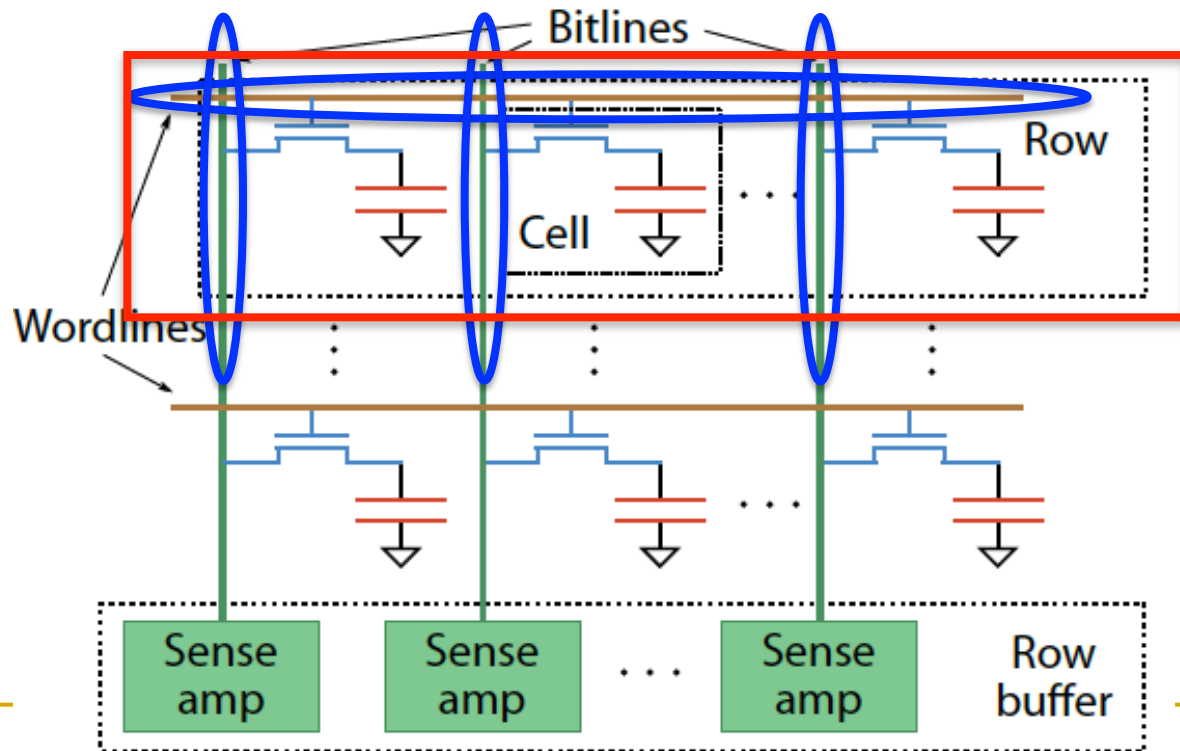
Data Pattern Dependence

- Electrical noise on the bitline affects reliable sensing of a DRAM cell
- The magnitude of this noise is affected by values of nearby cells via
    - Bitline-bitline coupling → electrical coupling between adjacent bitlines
    - Bitline-wordline coupling → electrical coupling between each bitline and the activated wordline
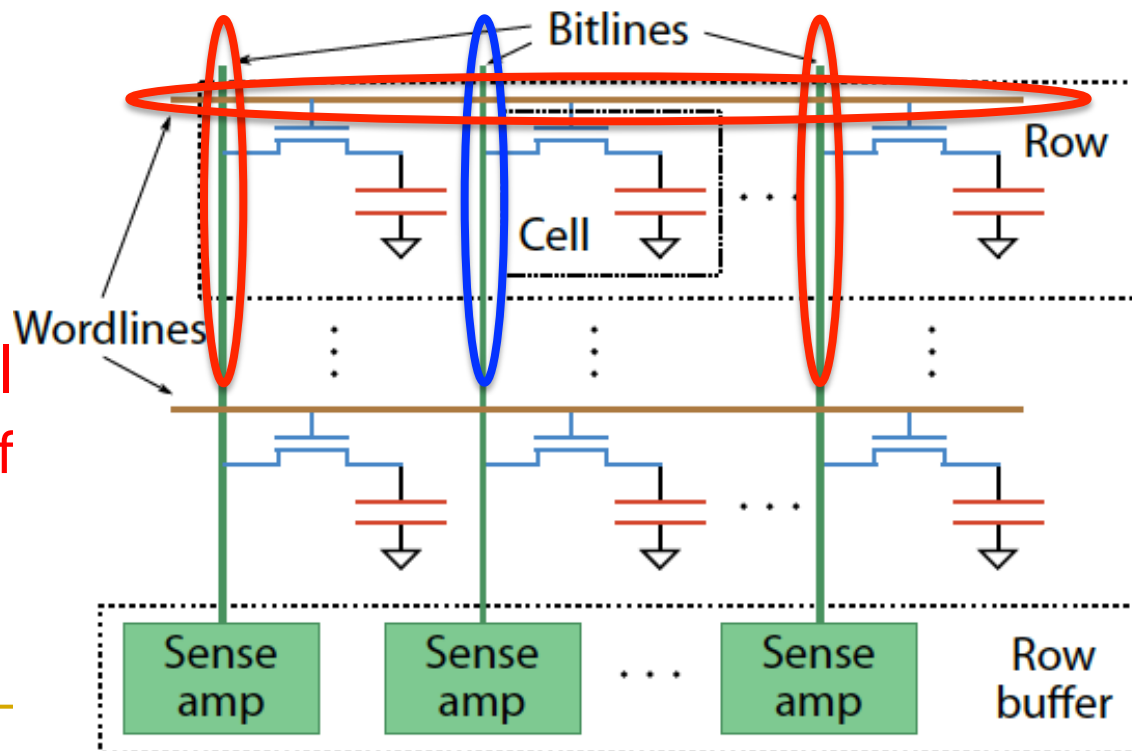
- Retention ... tored in nearby cel...
    - → need to f... etention time

SAFARI

Two Challenges to Retention Time Profiling

- **Challenge 2: Variable Retention Time (VRT)**
  - Retention time of a DRAM cell changes randomly over time
    - a cell alternates between multiple retention time states

  - Leakage current of a cell changes sporadically due to a charge trap in the gate oxide of the DRAM cell access transistor
  - When the trap becomes occupied, charge leaks more readily from the transistor's drain, leading to a short retention time
    - Called *Trap-Assisted Gate-Induced Drain Leakage*
  - This process appears to be a random process **[Kim+ IEEE TED'11]**

  - Worst-case retention time depends on a random process
    → need to find the worst case despite this
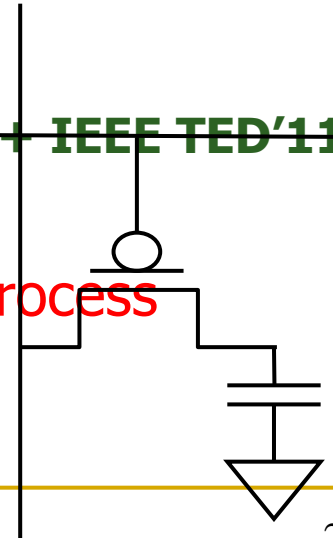
Our Goal

- Analyze the retention time behavior of DRAM cells in modern commodity DRAM devices
  - to aid the collection of accurate profile information

- Provide a comprehensive empirical investigation of two key challenges to retention time profiling
  - Data Pattern Dependence (DPD)
  - Variable Retention Time (VRT)

Talk Agenda

- DRAM Refresh: Background and Motivation
- Challenges and Our Goal
- DRAM Characterization Methodology
- Foundational Results
    - Temperature Dependence
    - Retention Time Distribution
- Data Pattern Dependence: Analysis and Implications
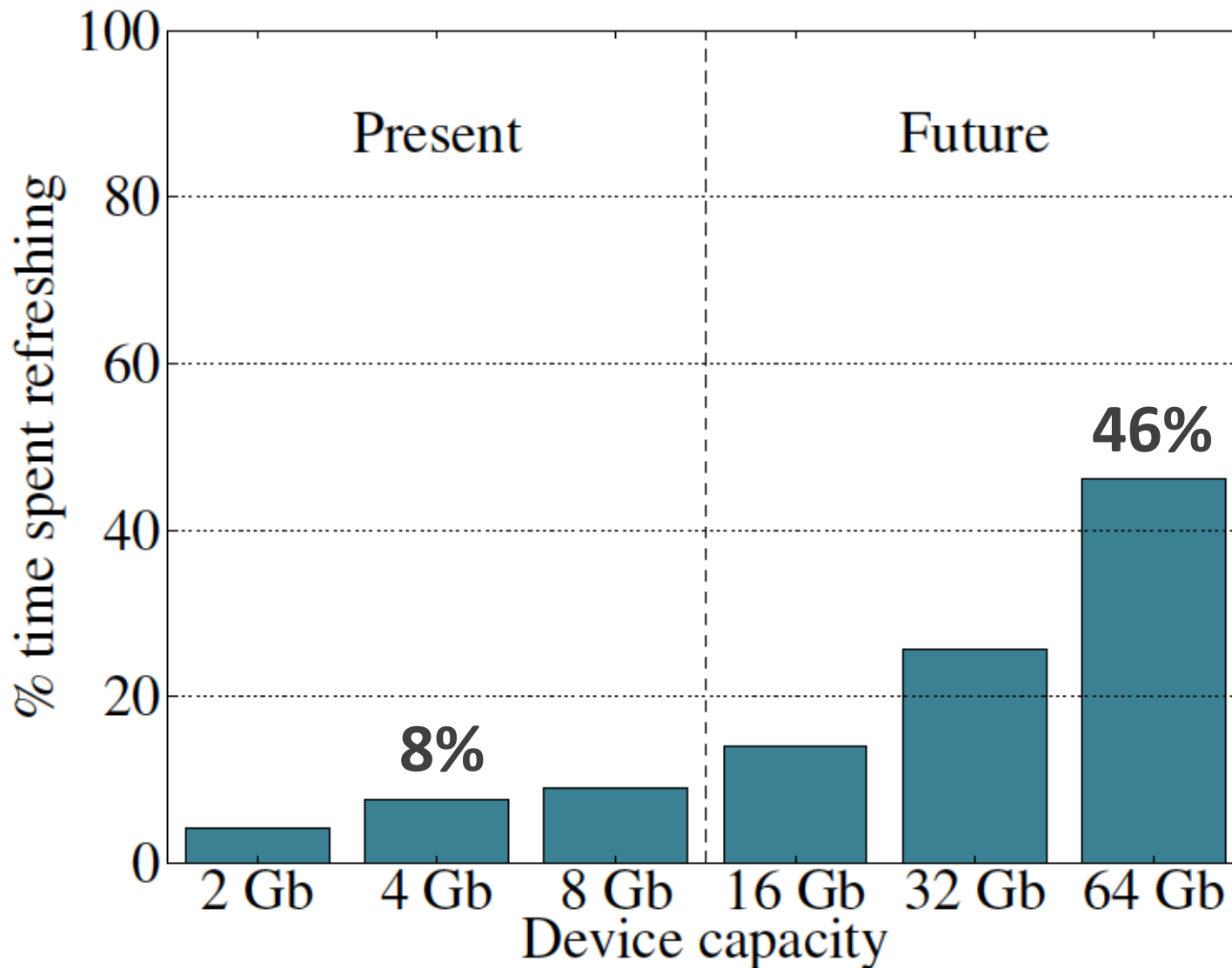- Variable Retention Time: Analysis and Implications
- Conclusions

SAFARI

DRAM Testing Platform and Method

- **Test platform:** Developed a DDR3 DRAM testing platform using the Xilinx ML605 FPGA development board
  - Temperature controlled

- **Tested DRAM chips:** 248 commodity DRAM chips from five manufacturers (A,B,C,D,E)
- Seven families based on equal capacity per device:
  - A 1Gb, A 2Gb
  - B 2Gb
  - C 2Gb
  - D 1Gb, D 2Gb
  - E 2Gb

Experiment Design

- Each module tested for multiple **rounds** of **tests**.

- Each test searches for the set of cells with a retention time less than a threshold value for a particular data pattern

- High-level structure of a test:
  - Write data pattern to rows in a DRAM bank
  - Prevent refresh for a period of time *tWAIT*, leave DRAM idle
  - Read stored data pattern, compare to written pattern and record corrupt cells as those with retention time < *tWAIT*

- Test details and important issues to pay attention to are discussed in paper

Experiment Structure

Test

Tests both the data pattern and its complement

Round

Round 1

Data Pattern X tWAIT = 1.5s	Data Pattern X tWAIT = 1.6s	...	Data Pattern X tWAIT = 6.0s
Data Pattern Y tWAIT = 1.5s	Data Pattern Y tWAIT = 1.6s		Data Pattern Y tWAIT = 6.0s
Data Pattern Z tWAIT = 1.5s	Data Pattern Z tWAIT = 1.6s		Data Pattern Z tWAIT = 6.0s

Round 2

Data Pattern X tWAIT = 1.5s
Data Pattern Y tWAIT = 1.5s
Data Pattern Z tWAIT = 1.5s
...

Experiment Parameters

- Most tests conducted at 45 degrees Celsius

- No cells observed to have a retention time less than 1.5 second at 45$^\text{o}$C

- Tested *tWAIT* in increments of 128ms from 1.5 to 6.1 seconds

Tested Data Patterns

- **All 0s/1s**: Value 0/1 is written to all bits   **Fixed patterns**
  - Previous work suggested this is sufficient

- **Checkerboard**: Consecutive bits alternate between 0 and 1
  - Coupling noise increases with voltage difference between the neighboring bitlines → May induce worst case data pattern (if adjacent bits mapped to adjacent cells)

- **Walk**: Attempts to ensure a single cell storing 1 is surrounded by cells storing 0
  - This may lead to even worse coupling noise and retention time due to coupling between *nearby* bitlines **[Li+ IEEE TCSI 2011]**
  - Walk pattern is permuted in each round to exercise different cells

- **Random**: Randomly generated data is written to each row
  - A new set of random data is generated for each round

Talk Agenda

- DRAM Refresh: Background and Motivation
- Challenges and Our Goal
- DRAM Characterization Methodology
- Foundational Results
  - Temperature Dependence
  - Retention Time Distribution
- Data Pattern Dependence: Analysis and Implications
- Variable Retention Time: Analysis and Implications
- Conclusions

Temperature Stability
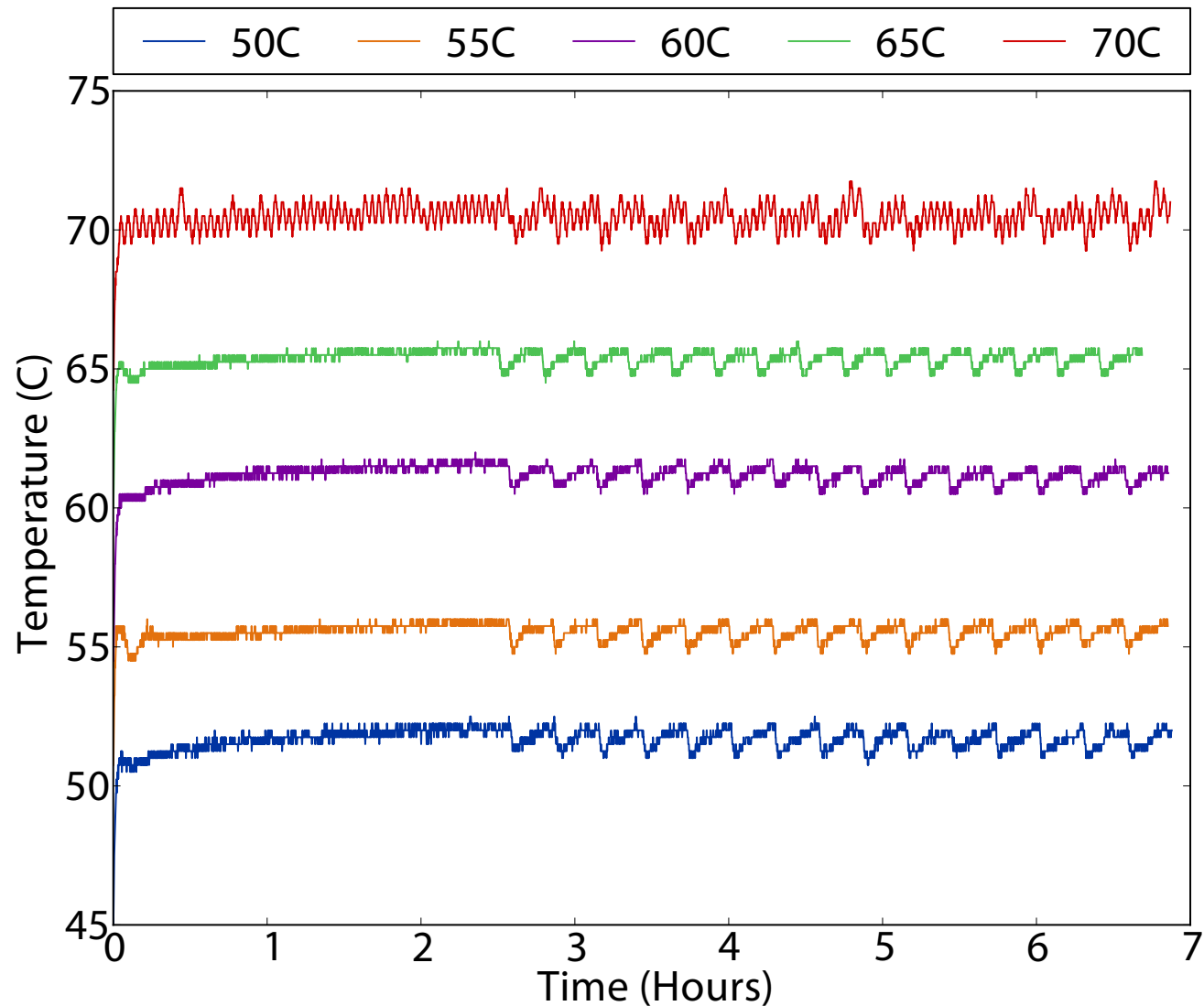
Tested chips at five different stable temperatures

Dependence of Retention Time on Temperature

Fraction of cells that exhibited retention time failure at any *tWAIT* for any data pattern at 50°C

Normalized retention times of the same cells at 55°C

Best-fit exponential curves for retention time change with temperature

Normalized retention times of the same cells At 70°C
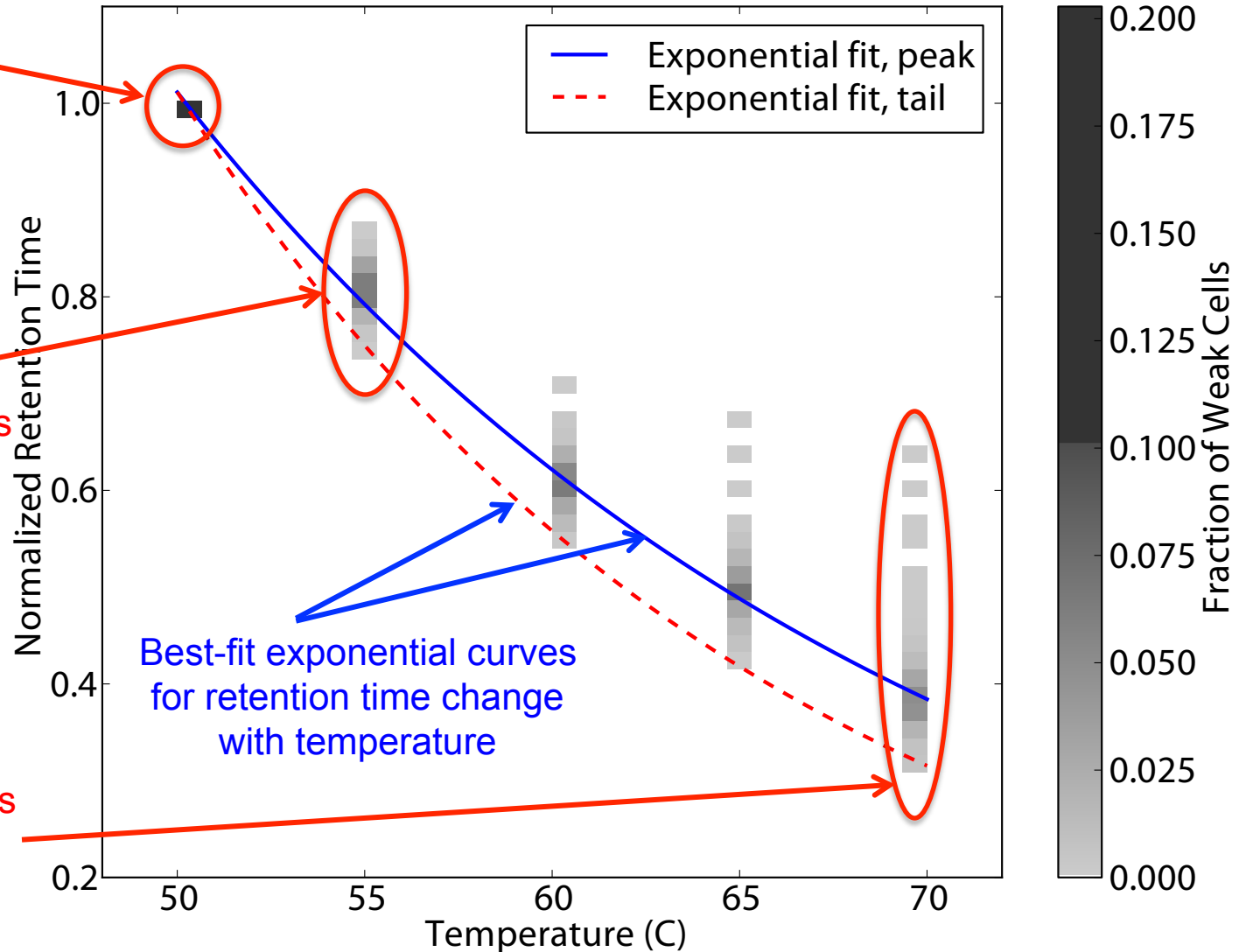
Dependence of Retention Time on Temperature
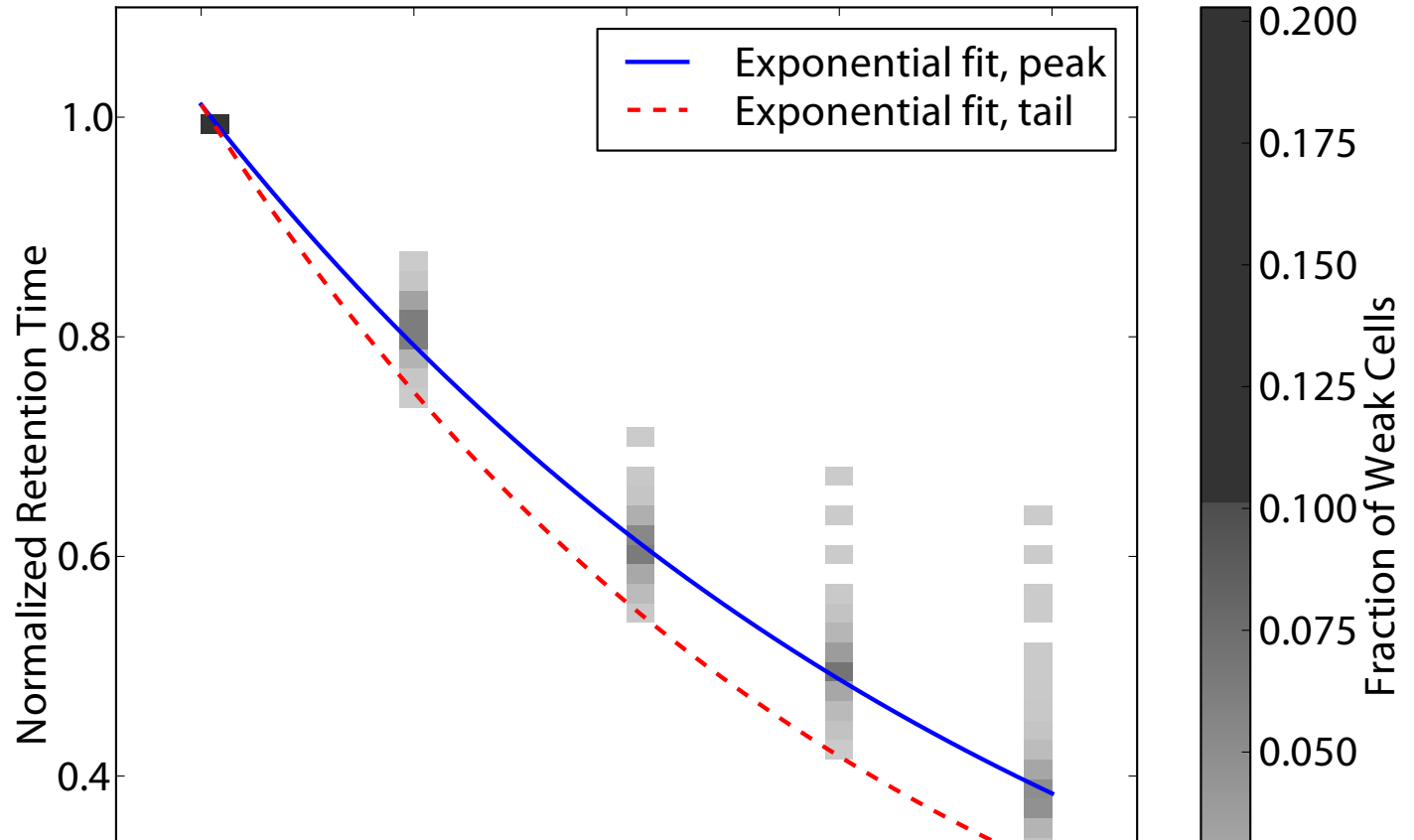
Relationship between retention time and temperature is consistently bounded (predictable) within a device

Every 10$^{\text{O}}$C temperature increase → 46.5% reduction in retention time in the worst case

Retention Time Distribution
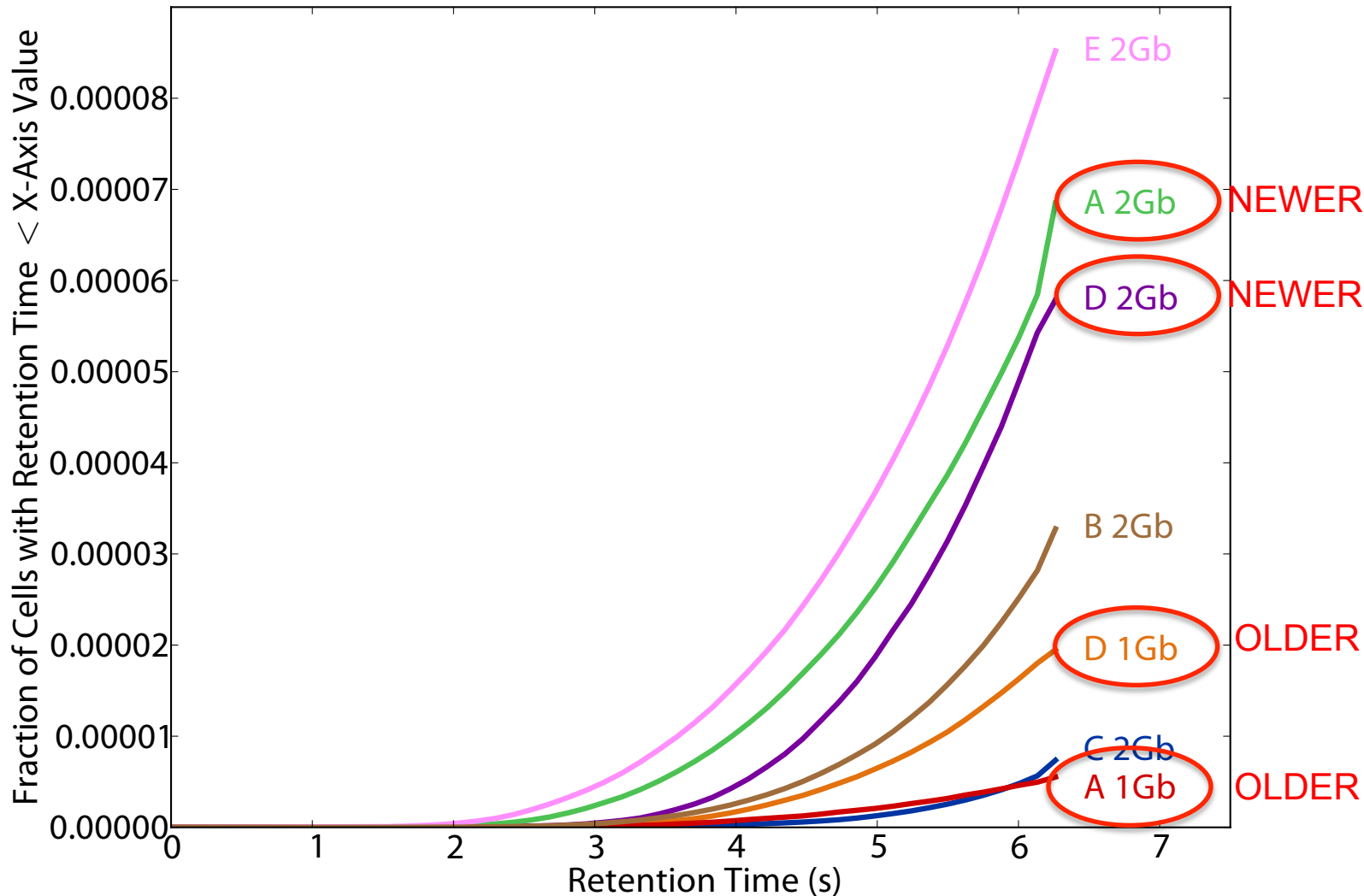
Newer device families have more weak cells than older ones
Likely a result of technology scaling

Talk Agenda

- DRAM Refresh: Background and Motivation
- Challenges and Our Goal
- DRAM Characterization Methodology
- Foundational Results
  - Temperature Dependence
  - Retention Time Distribution
- Data Pattern Dependence: Analysis and Implications
- Variable Retention Time: Analysis and Implications
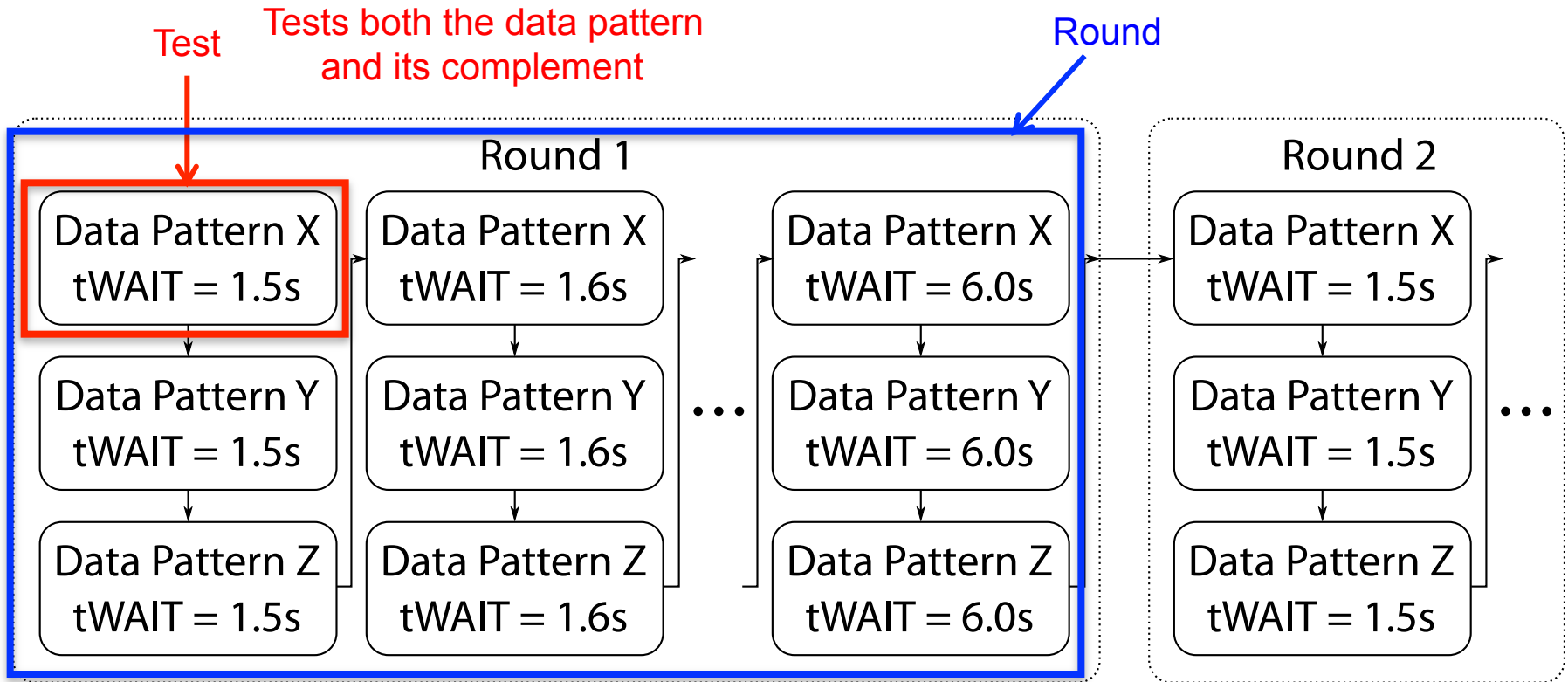- Conclusions

Some Terminology

- **Failure population of cells with Retention Time X:** The set of all cells that exhibit retention failure in any test with *any data pattern* at that retention time (*tWAIT*)

- **Retention Failure Coverage of a Data Pattern DP:** Fraction of cells with retention time X that exhibit retention failure with that *particular* data pattern DP

- If retention times are not dependent on data pattern stored in cells, we would expect
  - Coverage of any data pattern to be 100%
  - In other words, if one data pattern causes a retention failure, any other data pattern also would

Recall the Tested Data Patterns

- **All 0s/1s**: Value 0/1 is written to all bits   Fixed patterns

- **Checkerboard**: Consecutive bits alternate between 0 and 1

- **Walk**: Attempts to ensure a single cell storing 1 is surrounded by cells storing 0

- **Random**: Randomly generated data is written to each row

Retention Failure Coverage of Data Patterns
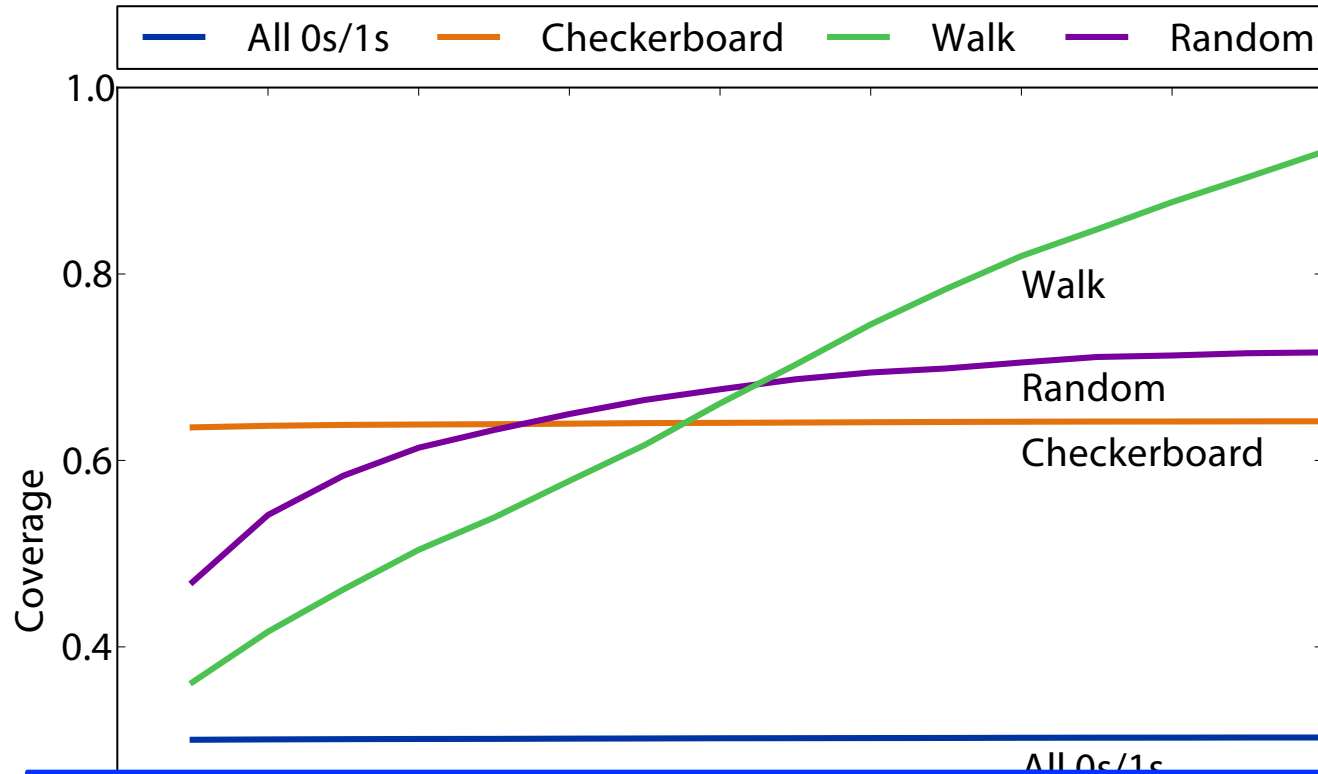
A 2Gb chip family

6.1s retention time

Different data patterns have widely different coverage:
Data pattern dependence exists and is severe

Coverage of fixed patterns is low: ~30% for *All 0s/1s*

***Walk* is the most effective data pattern for this device**

No data pattern achieves 100% coverage

Retention Failure Coverage of Data Patterns

Legend: All 0s/1s — Checkerboard — Walk — Random
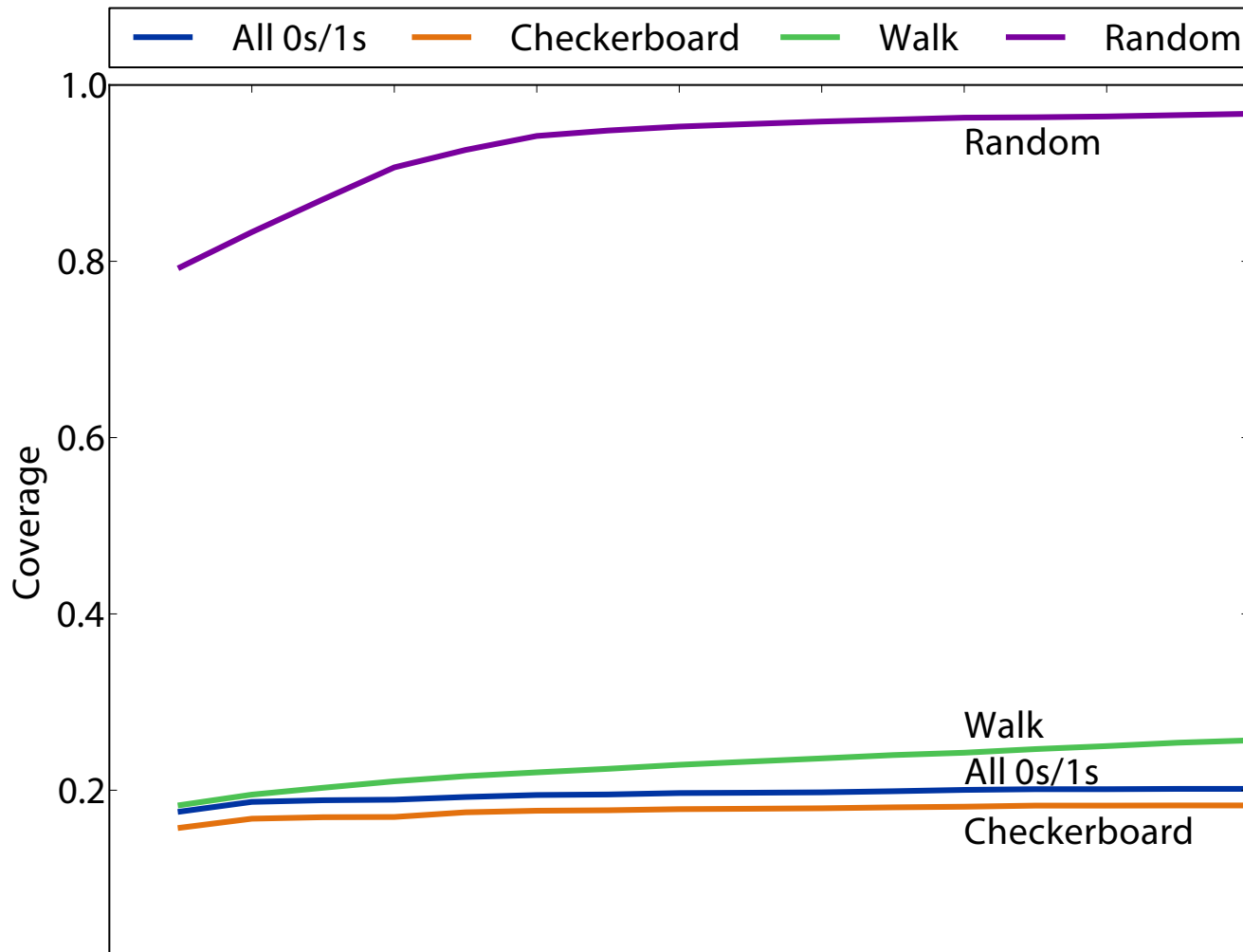
B 2Gb chip family

6.1s retention time

Random is the most effective data pattern for this device

No data pattern achieves 100% coverage

Retention Failure Coverage of Data Patterns
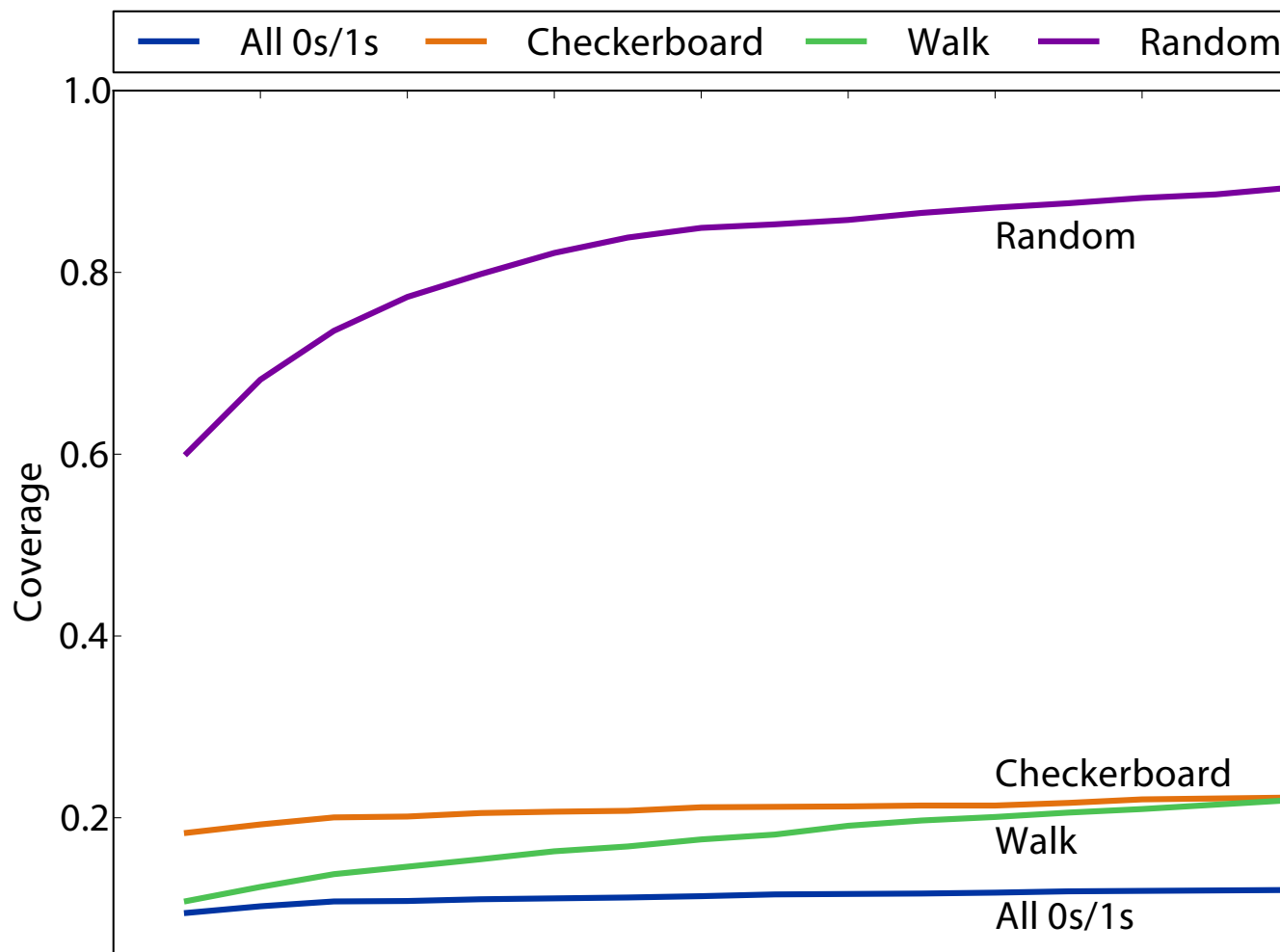
C 2Gb chip family

6.1s retention time

Random is the most effective data pattern for this device

No data pattern achieves 100% coverage

Data Pattern Dependence: Observations (I)

- **A cell's retention time is heavily influenced by data pattern stored in other cells**
  - Pattern affects the coupling noise, which affects cell leakage

- **No tested data pattern exercises the worst case retention time for all cells (no pattern has 100% coverage)**
  - No pattern is able to induce the worst-case coupling noise for every cell
  - Problem: Underlying DRAM circuit organization is *not* known to the memory controller → very hard to construct a pattern that exercises the worst-case cell leakage
    - → Opaque mapping of addresses to physical DRAM geometry
    - → Internal remapping of addresses within DRAM to tolerate faults
    - → Second order coupling effects are very hard to determine

Data Pattern Dependence: Observations (II)

- **Fixed, simple data patterns have low coverage**
  - They do not exercise the worst-case coupling noise

- **The effectiveness of each data pattern varies significantly between DRAM devices (of the same or different vendors)**
  - Underlying DRAM circuit organization likely differs between different devices → patterns leading to worst coupling are different in different devices

- **Technology scaling appears to increase the impact of data pattern dependence**
  - Scaling reduces the physical distance between circuit elements, increasing the magnitude of coupling effects

Effect of Technology Scaling on DPD
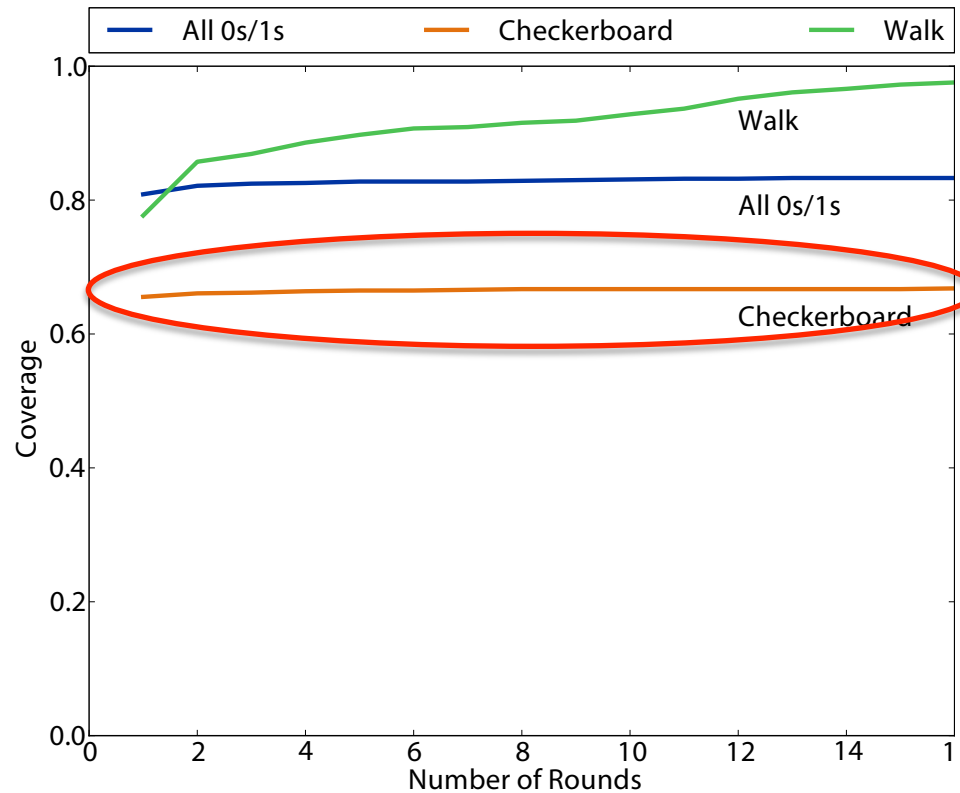
A 1Gb chip family
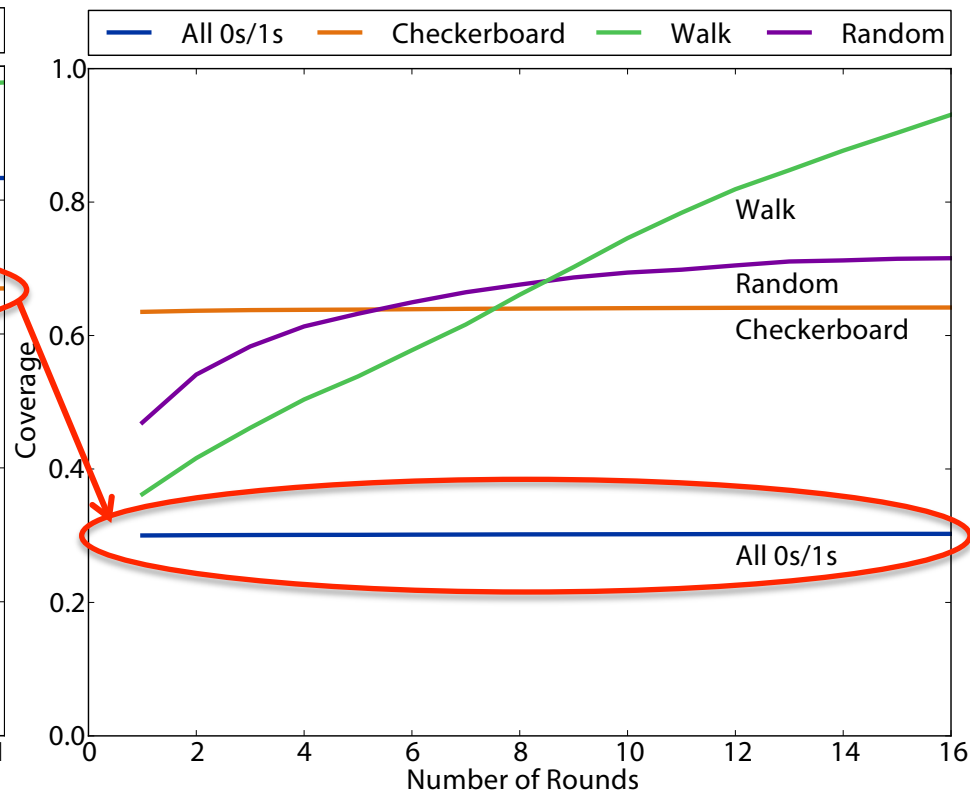
A 2Gb chip family

The lowest-coverage data pattern achieves much lower coverage for the smaller technology node

SAFARI

DPD: Implications on Profiling Mechanisms

- Any retention time profiling mechanism must handle data pattern dependence of retention time

- Intuitive approach: Identify the data pattern that induces the worst-case retention time for a particular cell or device

- Problem 1: Very hard to know at the memory controller which bits actually interfere with each other due to
  - Opaque mapping of addresses to physical DRAM geometry → logically consecutive bits may not be physically consecutive
  - Remapping of faulty bitlines/wordlines to redundant ones internally within DRAM

- Problem 2: Worst-case coupling noise is affected by non-obvious second order bitline coupling effects

DPD: Suggestions (for Future Work)

- A mechanism for identifying worst-case data pattern(s) likely requires support from DRAM device

  - DRAM manufacturers might be in a better position to do this

  - But, the ability of the manufacturer to identify and expose the entire retention time profile is limited due to VRT

- An alternative approach: Use random data patterns to increase coverage as much as possible; handle incorrect retention time estimates with ECC

  - Need to keep profiling time in check

  - Need to keep ECC overhead in check

Talk Agenda

- DRAM Refresh: Background and Motivation
- Challenges and Our Goal
- DRAM Characterization Methodology
- Foundational Results
  - Temperature Dependence
  - Retention Time Distribution
- Data Pattern Dependence: Analysis and Implications
- Variable Retention Time: Analysis and Implications
- Conclusions

SAFARI

Variable Retention Time

- Retention time of a cell can vary over time

- A cell can randomly switch between multiple leakage current states due to *Trap-Assisted Gate-Induced Drain Leakage,* which appears to be a random process

  [Yaney+ IEDM 1987, Restle+ IEDM 1992]

An Example VRT Cell

A cell from E 2Gb chip family
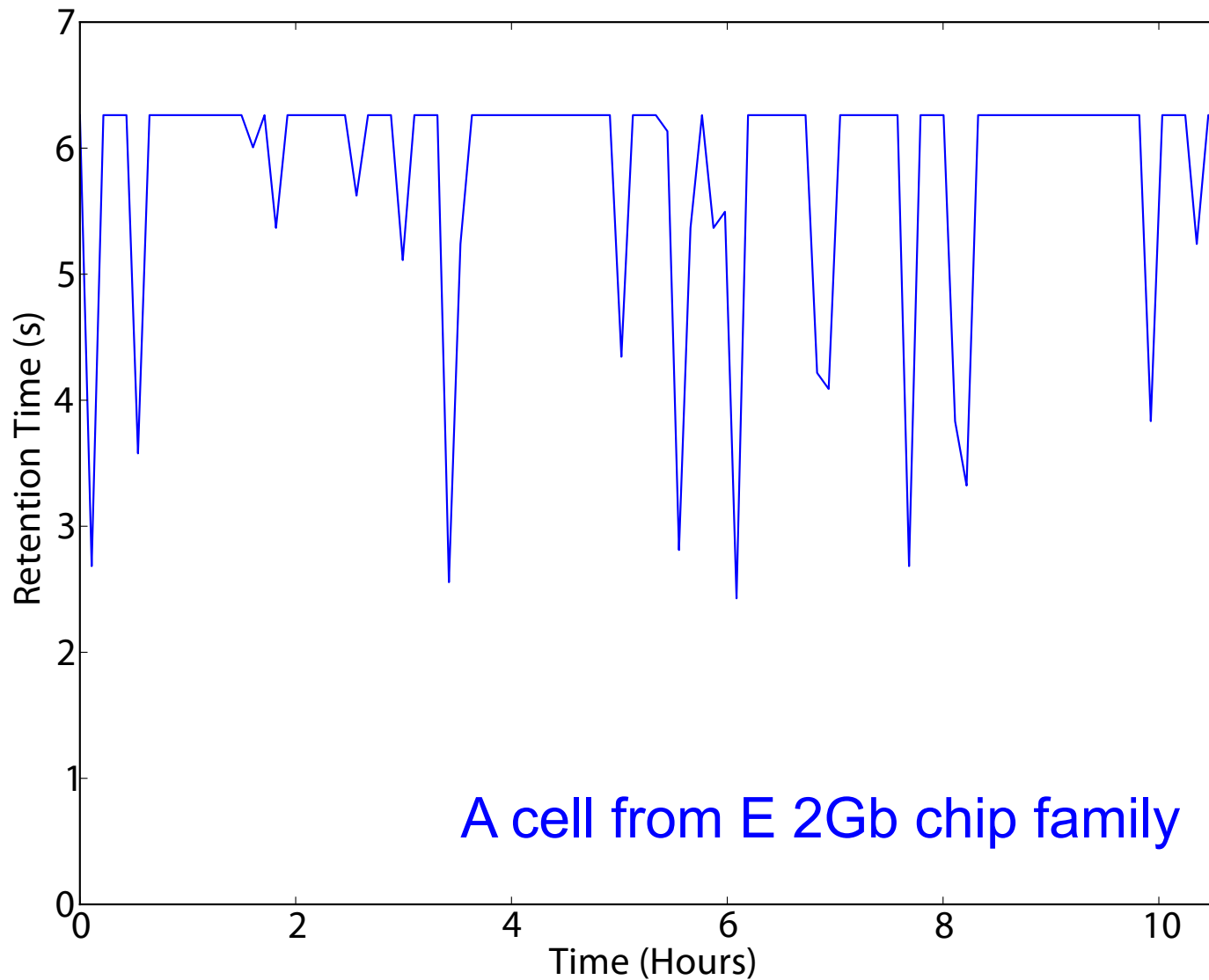
VRT: Questions and Methodology

- **Key Questions**
  - How prevalent is VRT in modern DRAM devices?
  - What is the timescale of observation of the lowest retention time state?
  - What are the implications on retention time profiling?

- **Test Methodology**
  - Each device was tested for at least 1024 rounds over 24 hours
  - Temperature fixed at 45°C
  - Data pattern used is the most effective data pattern for each device
  - For each cell that fails at any retention time, we record the minimum and the maximum retention time observed

Variable Retention Time

Many failing cells jump from very high retention time to very low

Most failing cells exhibit VRT

Min ret time = Max ret time
Expected if no VRT

A 2Gb chip family

Maximum Retention Time (s)

Minimum Retention Time (s)

log10(Fraction of Cells)

Variable Retention Time

B 2Gb chip family

Variable Retention Time

C 2Gb chip family
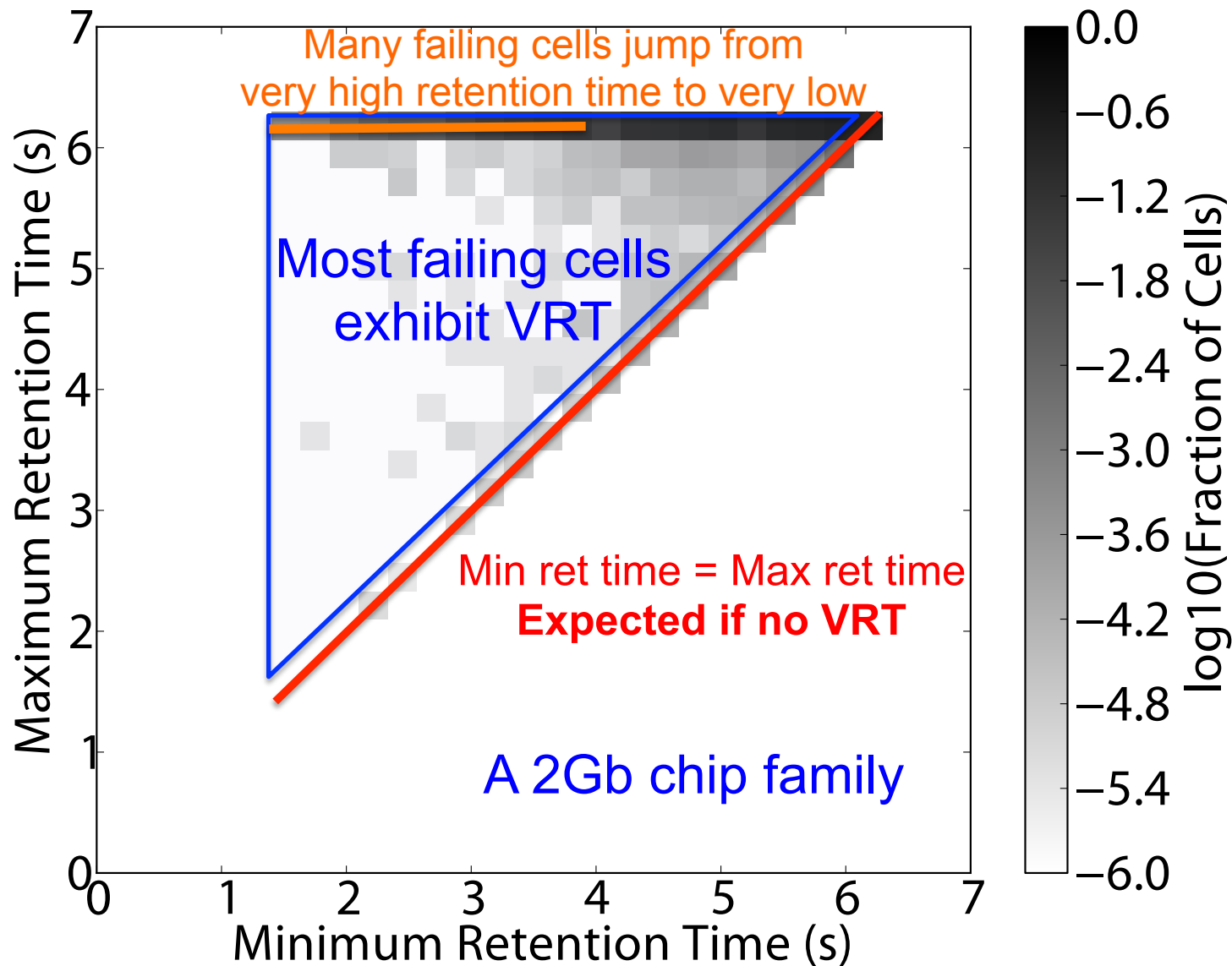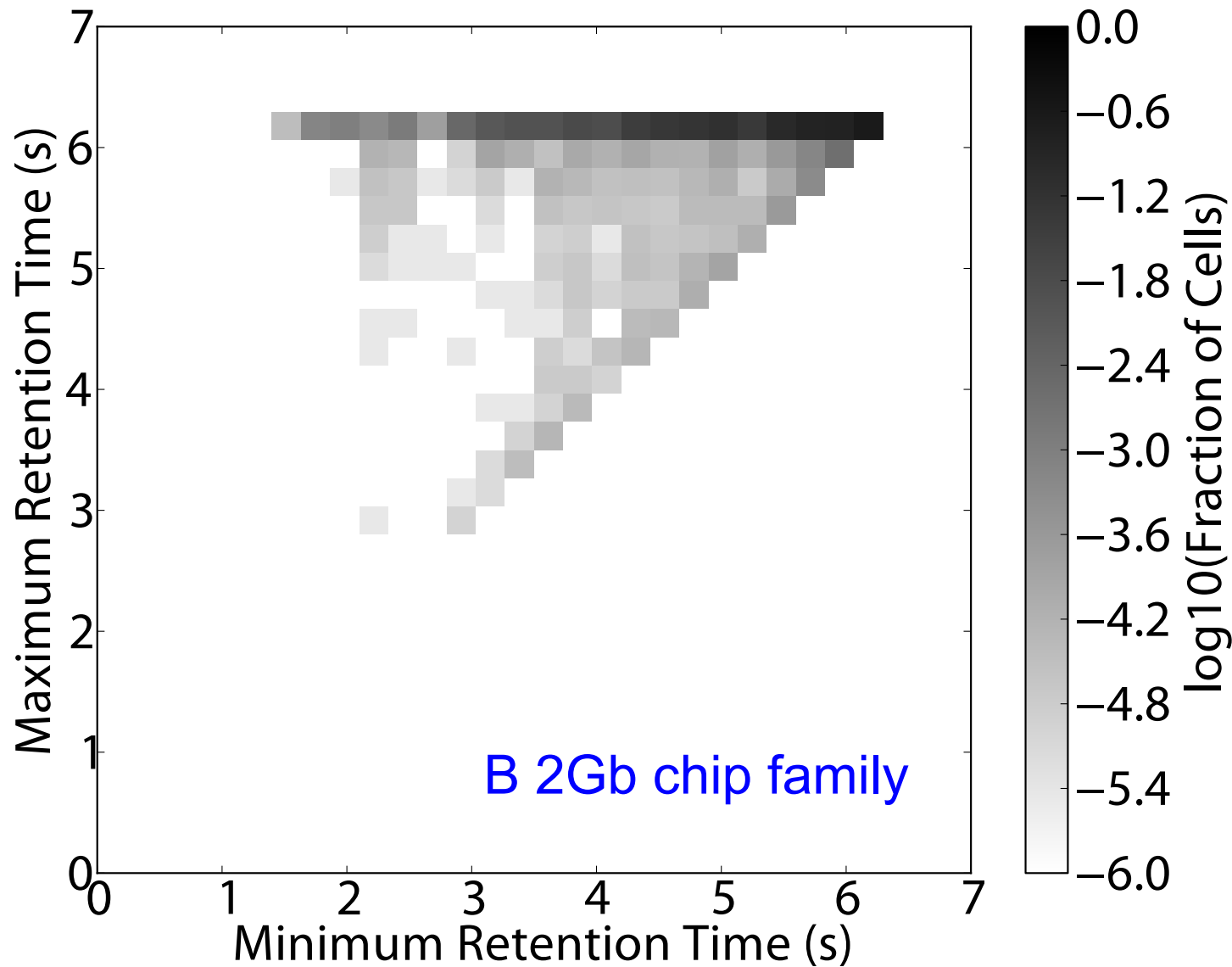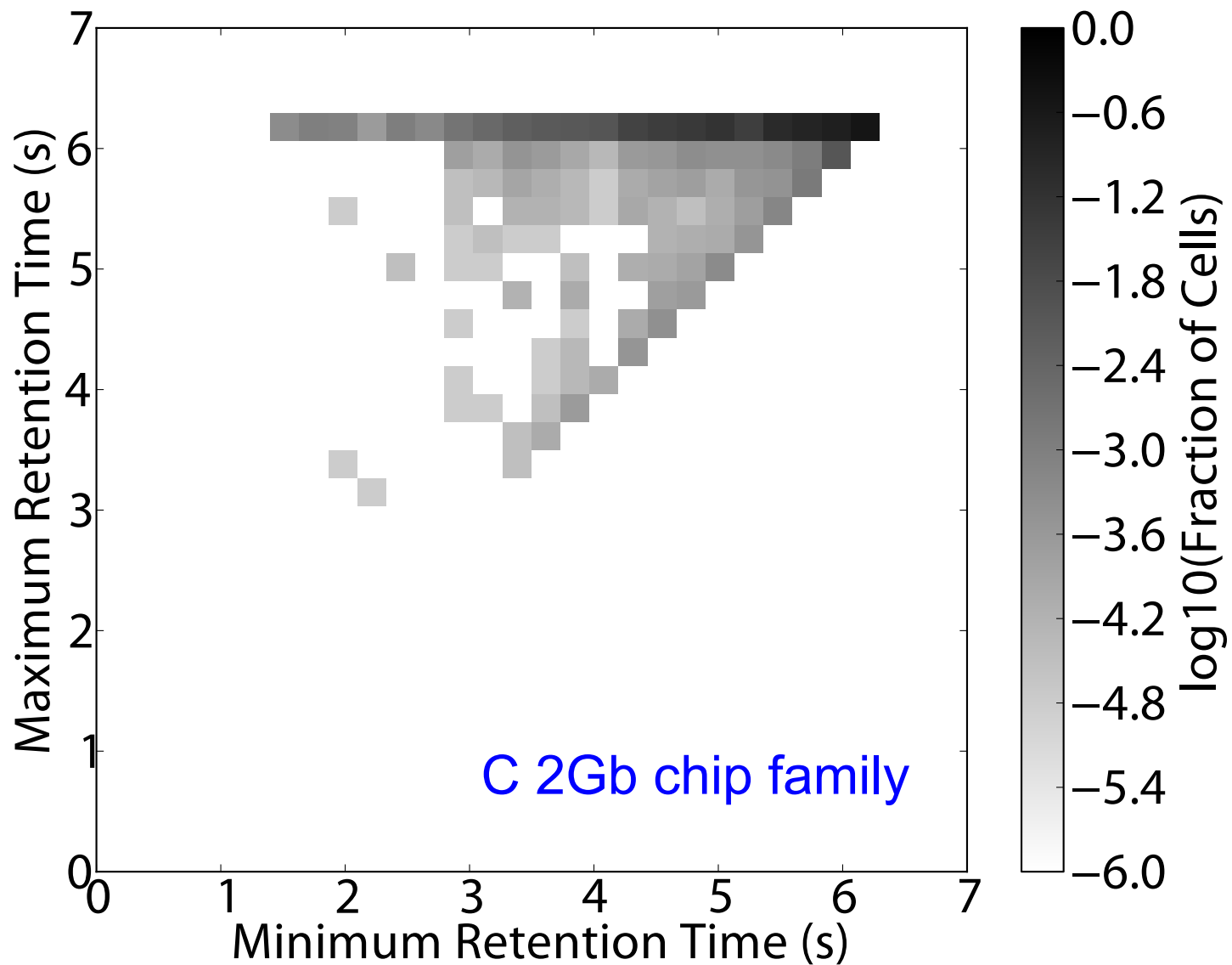
VRT: Observations So Far

- **VRT is common among weak cells** (i.e., those cells that experience low retention times)

- **VRT can result in significant retention time changes**
  - Difference between minimum and maximum retention times of a cell can be more than 4x, and may not be bounded
  - Implication: Finding *a* retention time for a cell and using a guardband to ensure minimum retention time is "covered" requires a large guardband or may not work

- **Retention time profiling mechanisms must identify lowest retention time in the presence of VRT**
  - Question: How long to profile a cell to find its lowest retention time state?

Time Between Retention Time State Changes

- **How much time does a cell spend in a high retention state** before switching to the minimum observed retention time state?

Time Spent in High Retention Time State

A 2Gb chip family

~4 hours

~1 day
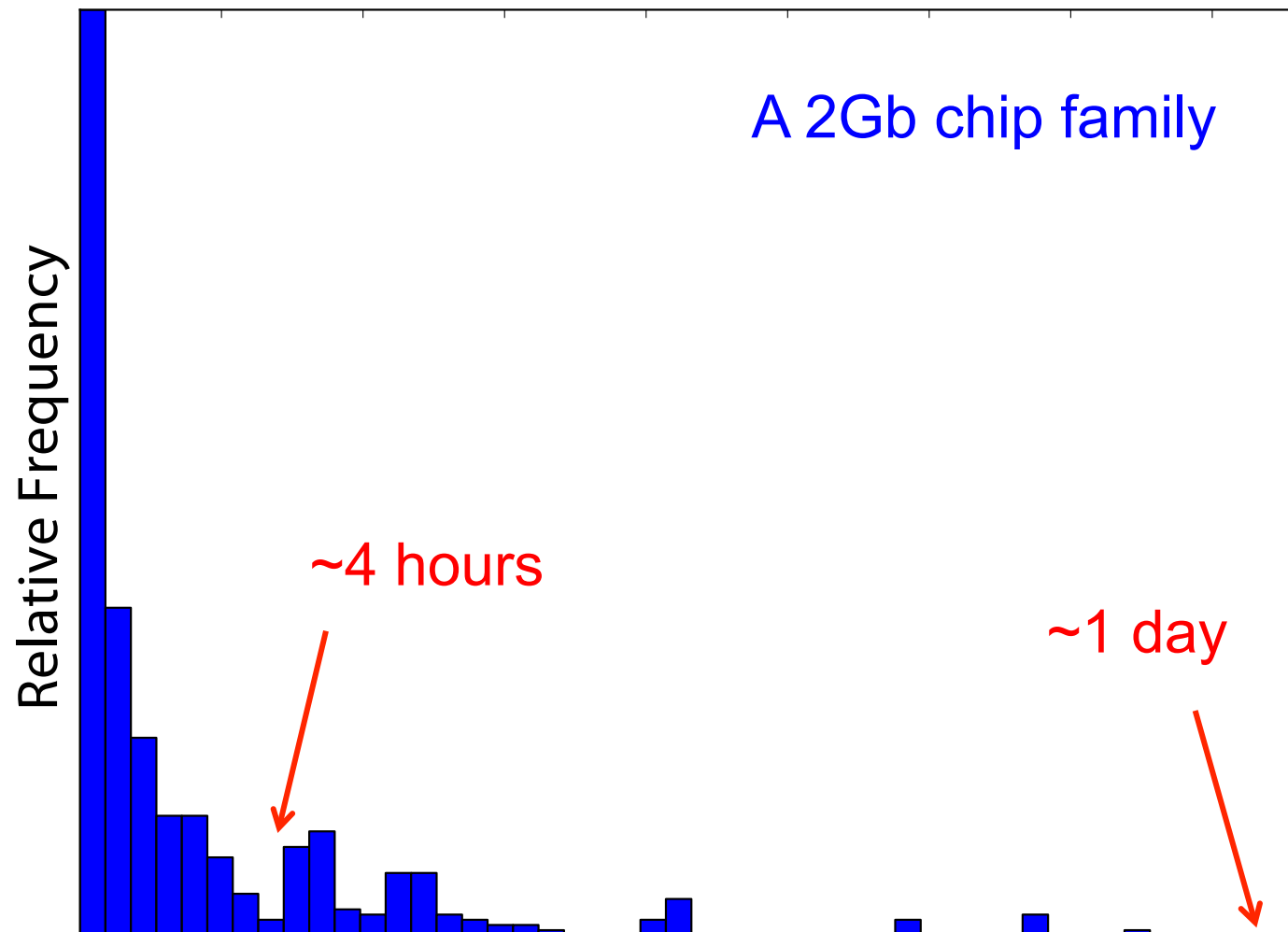
Need to profile for a long time to get to the minimum retention time state

Time Spent in High Retention Time State

Time Spent in High Retention Time State

C 2Gb chip family

Relative Frequency

Time Spent in High Retention Time State (s)

0   20000   40000   60000   80000
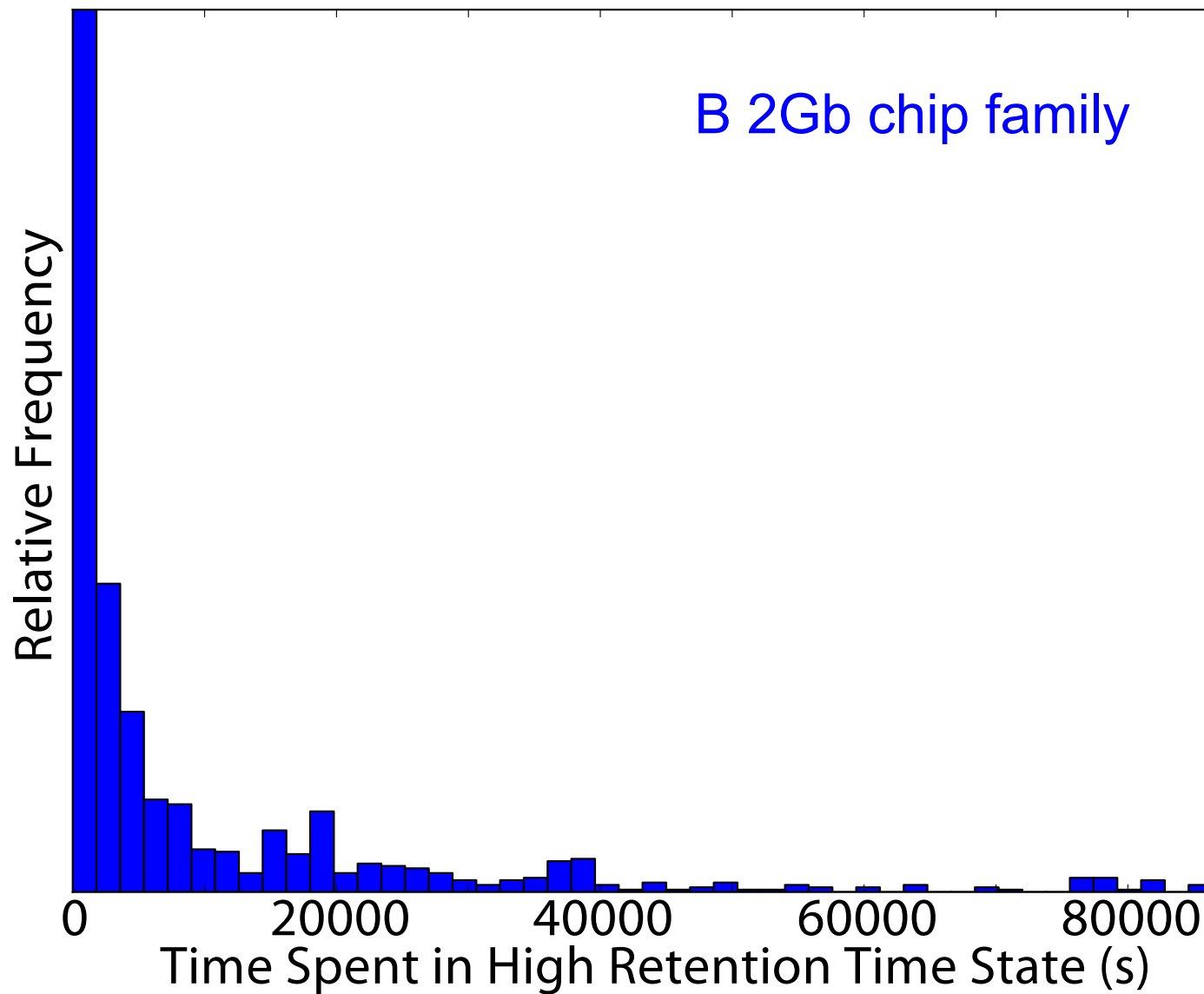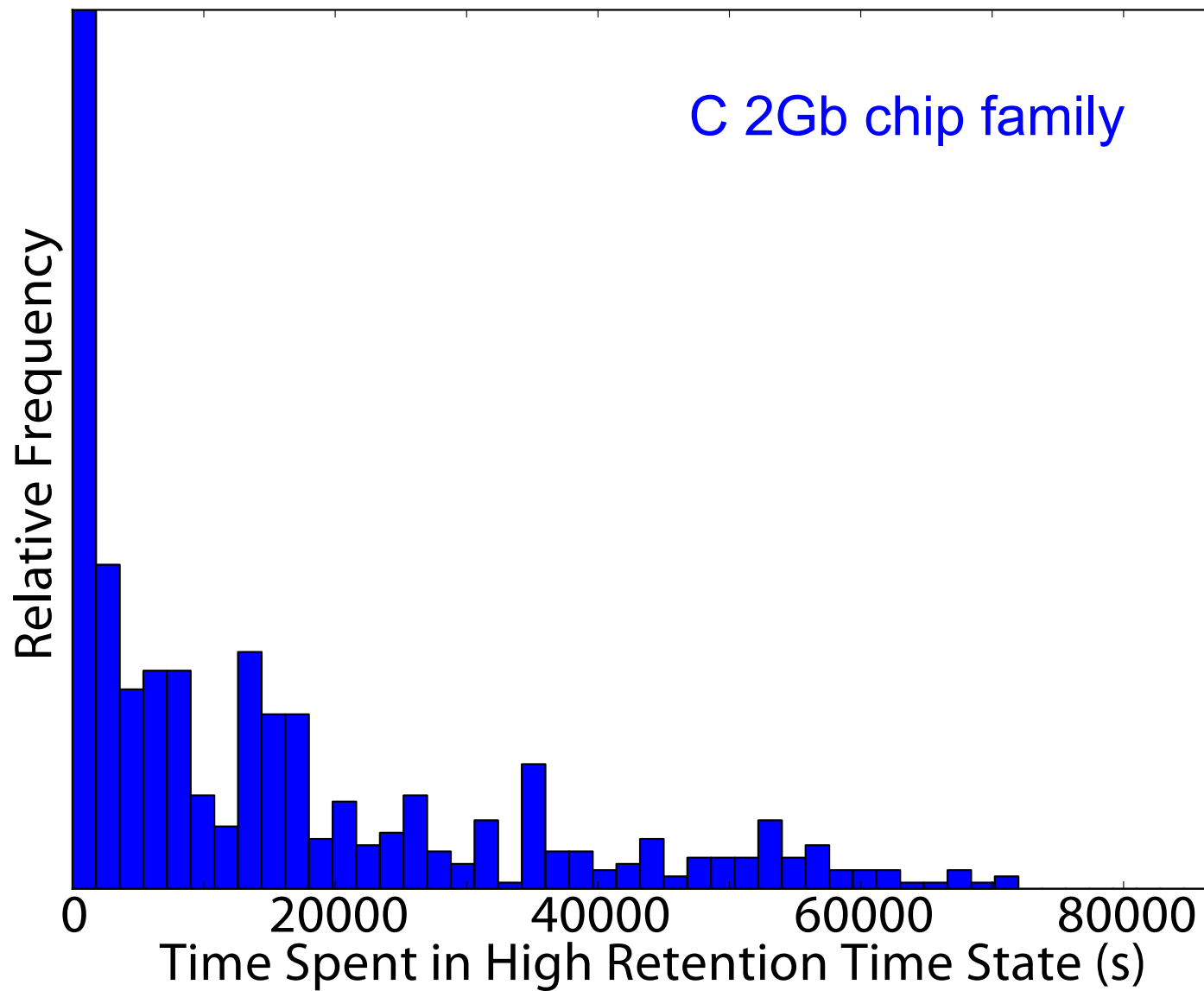
VRT: Implications on Profiling Mechanisms

- Problem 1: There does not seem to be a way of determining if a cell exhibits VRT without actually observing a cell exhibiting VRT
  - VRT is a memoryless random process **[Kim+ JJAP 2010]**

- Problem 2: VRT complicates retention time profiling by DRAM manufacturers
  - Exposure to very high temperatures can induce VRT in cells that were not previously susceptible
    - → can happen during soldering of DRAM chips
    - → manufacturer's retention time profile may not be accurate

- One option for future work: Use ECC to continuously profile DRAM online while aggressively reducing refresh rate
 - Need to keep ECC overhead in check

Talk Agenda

- DRAM Refresh: Background and Motivation
- Challenges and Our Goal
- DRAM Characterization Methodology
- Foundational Results
    - Temperature Dependence
    - Retention Time Distribution
- Data Pattern Dependence: Analysis and Implications
- Variable Retention Time: Analysis and Implications
- Conclusions

SAFARI

Summary and Conclusions

- DRAM refresh is a critical challenge in scaling DRAM technology efficiently to higher capacities and smaller feature sizes

- Understanding the retention time of modern DRAM devices can enable old or new methods to reduce the impact of refresh
  - Many mechanisms require accurate and reliable retention time profiles

- We presented the first work that comprehensively examines data retention behavior in modern commodity DRAM devices
  - Characterized 248 devices from five manufacturers

- Key findings: Retention time of a cell significantly depends on data pattern stored in other cells (data pattern dependence) and changes over time via a random process (variable retention time)
  - Discussed the underlying reasons and provided suggestions

- Future research on retention time profiling should solve the challenges posed by the DPD and VRT phenomena

An Experimental Study of Data Retention Behavior in Modern DRAM Devices

Implications for Retention Time Profiling Mechanisms

Jamie Liu[1]   Ben Jaiyen[1]   Yoongu Kim[1]

Chris Wilkerson[2]   Onur Mutlu[1]

[1] Carnegie Mellon University
[2] Intel Corporation

SAFARI

Carnegie Mellon

Flash Memory Scaling

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

Readings in Flash Memory

- Yu Cai, Gulay Yalcin, <u>Onur Mutlu</u>, 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, <u>Onur Mutlu</u>, 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, <u>Onur Mutlu</u>, 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, <u>Onur Mutlu</u>, 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)

Evolution of NAND Flash Memory
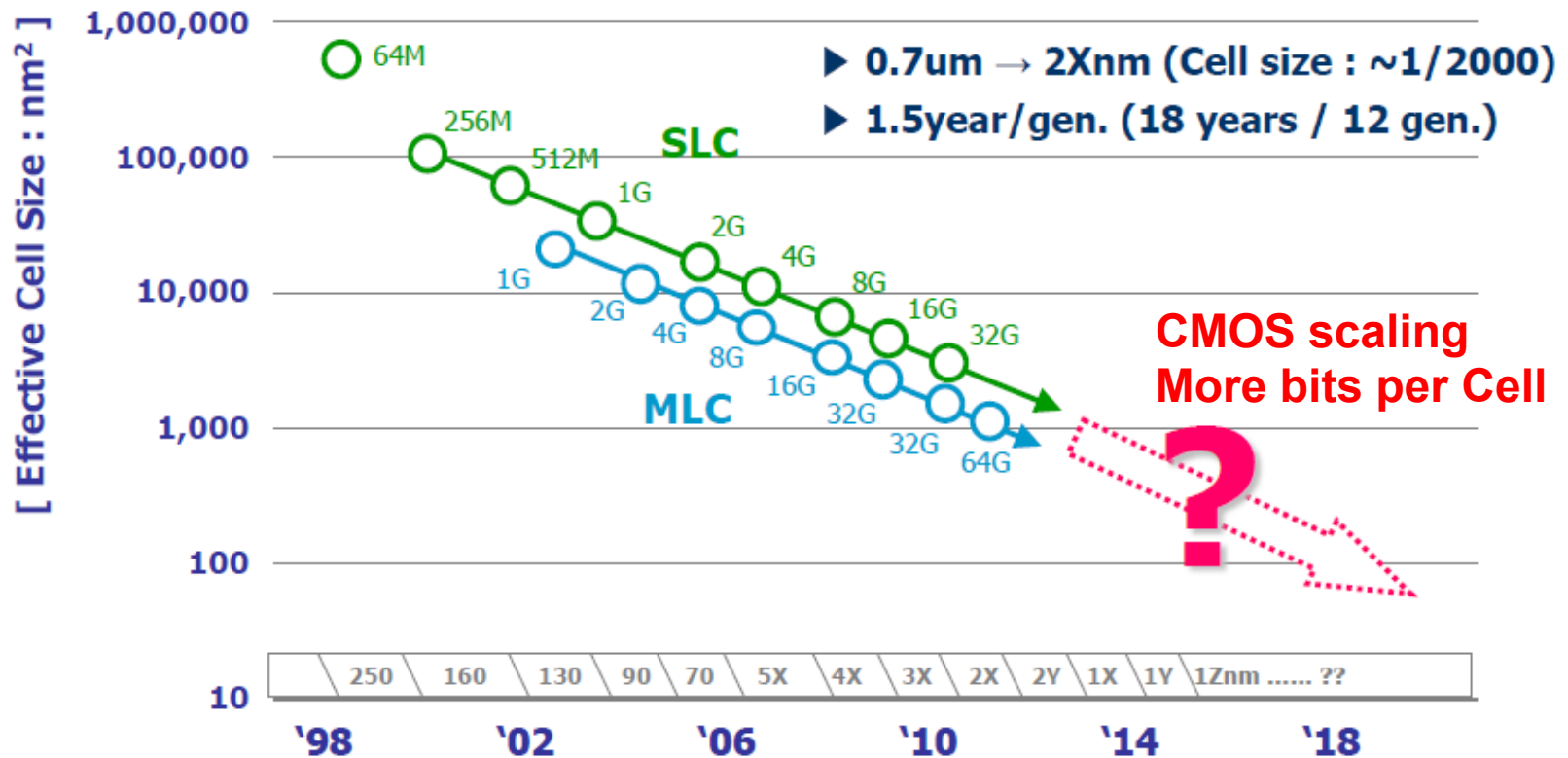
Seaung Suk Lee, "Emerging Challenges in NAND Flash Technology", Flash Summit 2011 (Hynix)

- Flash memory widening its range of applications
  - Portable consumer devices, laptop PCs and enterprise servers

Decreasing Endurance with Flash Scaling
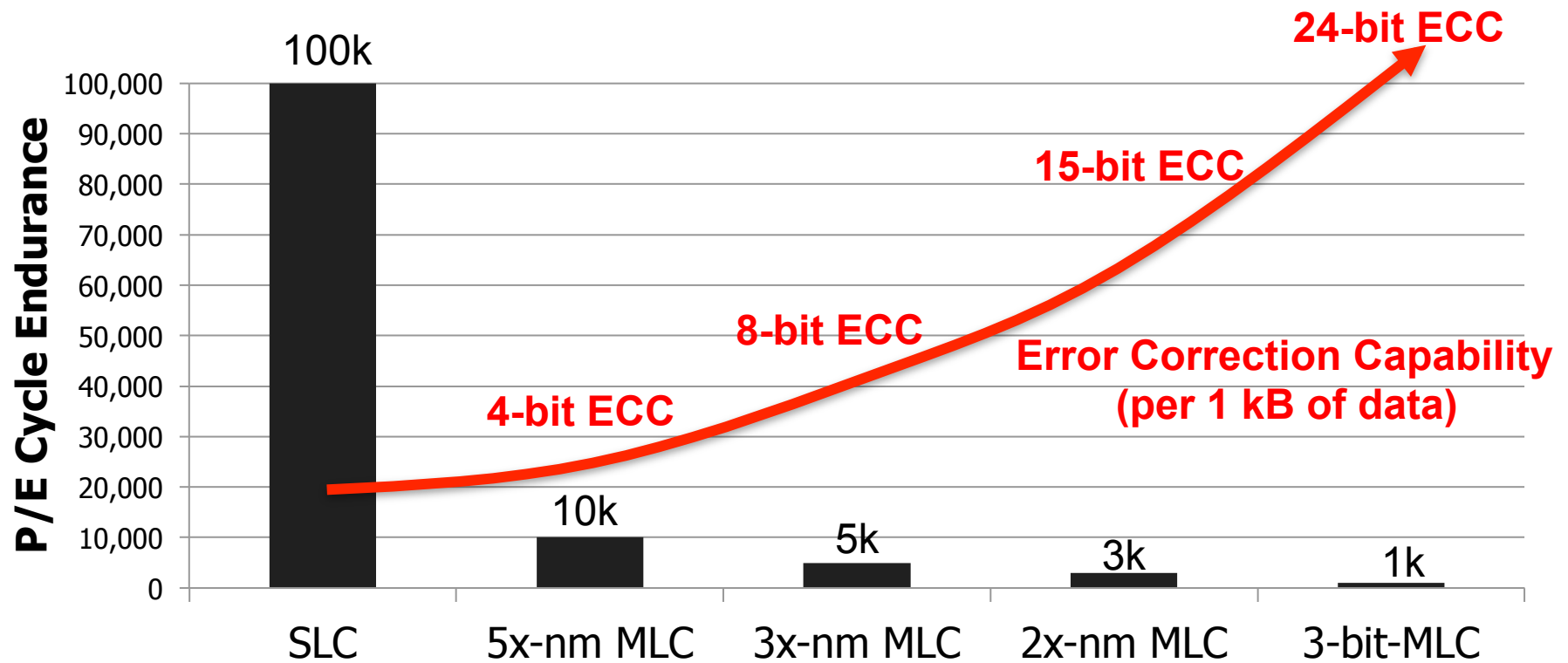
Ariel Maislos, "A New Era in Embedded Flash Memory", Flash Summit 2011 (Anobit)

- Endurance of flash memory decreasing with scaling and multi-level cells
- Error correction capability required to guarantee storage-class reliability (UBER < 10^{-15}) is increasing exponentially to reach *less* endurance

UBER: Uncorrectable bit error rate. Fraction of erroneous bits after error correction.

SAFARI

Carnegie Mellon

Future NAND Flash Storage Architecture

Noisy

Memory Signal Processing

Raw Bit Error Rate

Error Correction

BER $< 10^{-15}$

- Read voltage adjusting
- Data scrambler
- Data recovery
- Soft-information estimation

- Hamming codes
- BCH codes
- Reed-Solomon codes
- LDPC codes
- Other Flash friendly codes
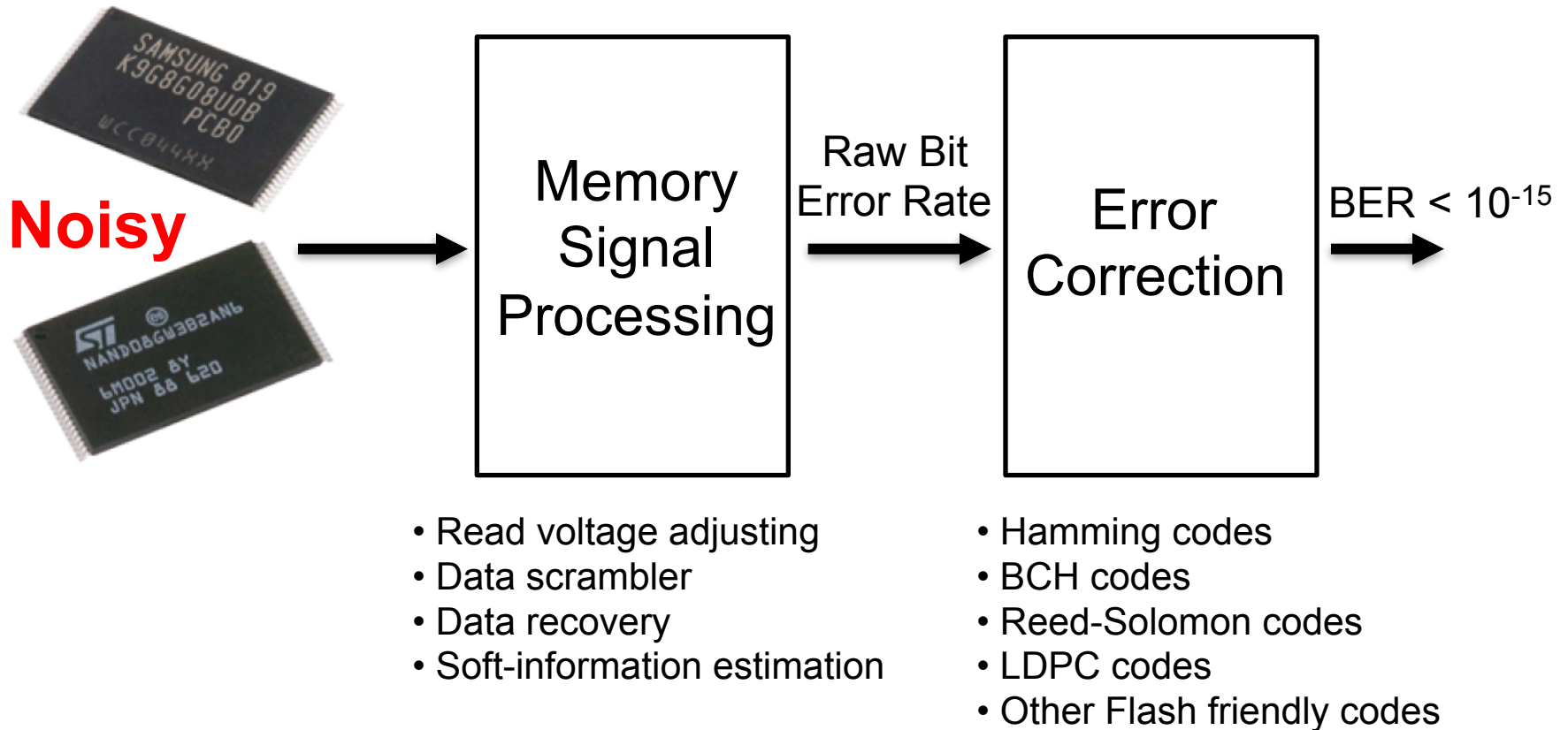
Need to understand NAND flash error patterns

Carnegie Mellon

Test System Infrastructure

Algorithms
- Wear Leveling
- Address Mapping
- Garbage Collection
- ECC (BCH, RS, LDPC)
- Signal Processing

Software Platform

USB Driver

Host USB PHY

Control Firmware

USB PHYChip

FPGA USB controller

NAND Controller

Flash Memories

1. Reset
2. Erase block
3. Program page
4. Read page

Host Computer   USB Daughter Board   Mother Board   Flash Board
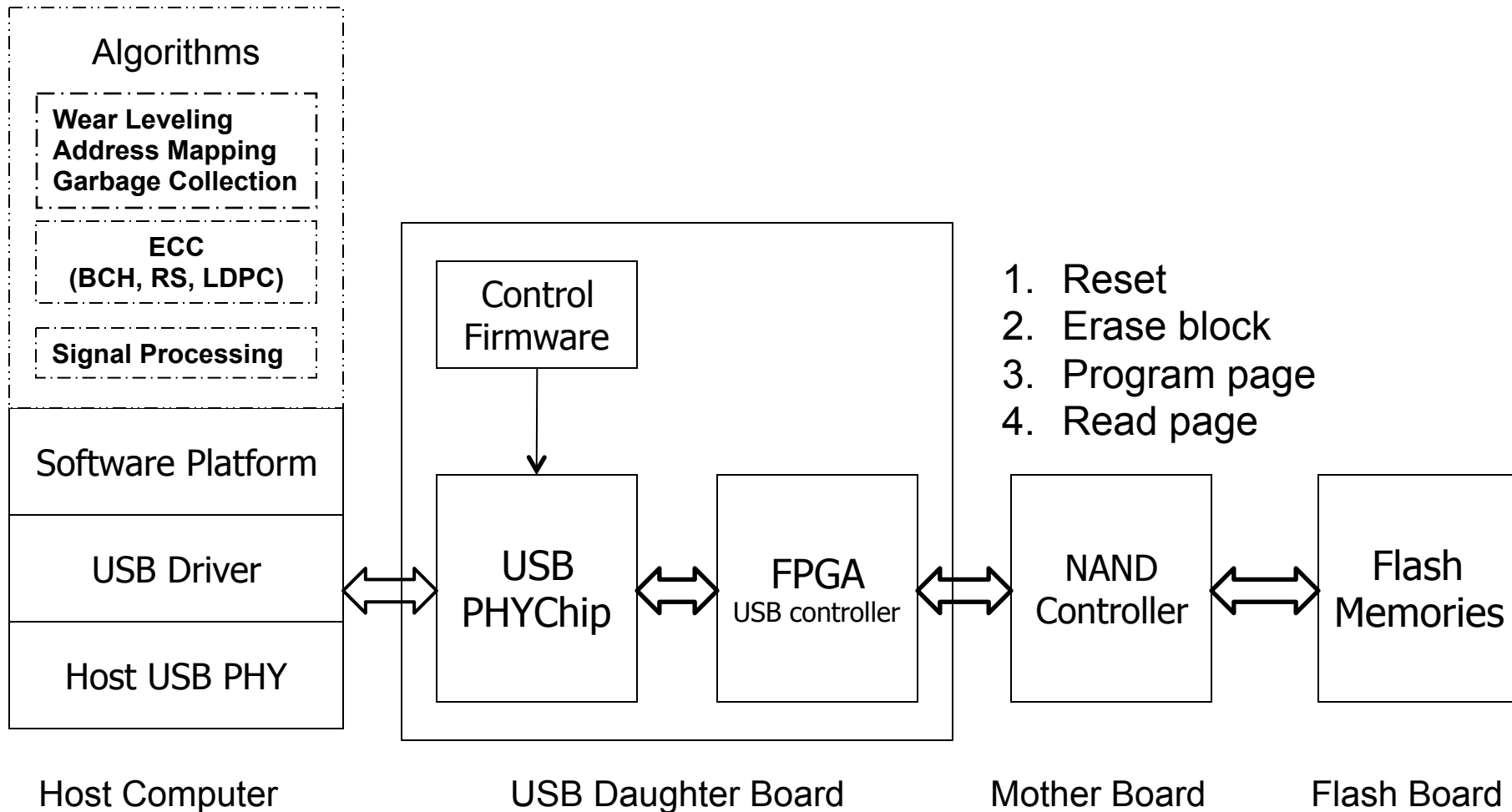
NAND Flash Testing Platform

USB Daughter Board

USB Jack

HAPS-52 Mother Board

Virtex-II Pro
(USB controller)

3x-nm
NAND Flash

Virtex-V FPGA
(NAND Controller)

NAND Daughter Board

SAFARI

Carnegie Mellon
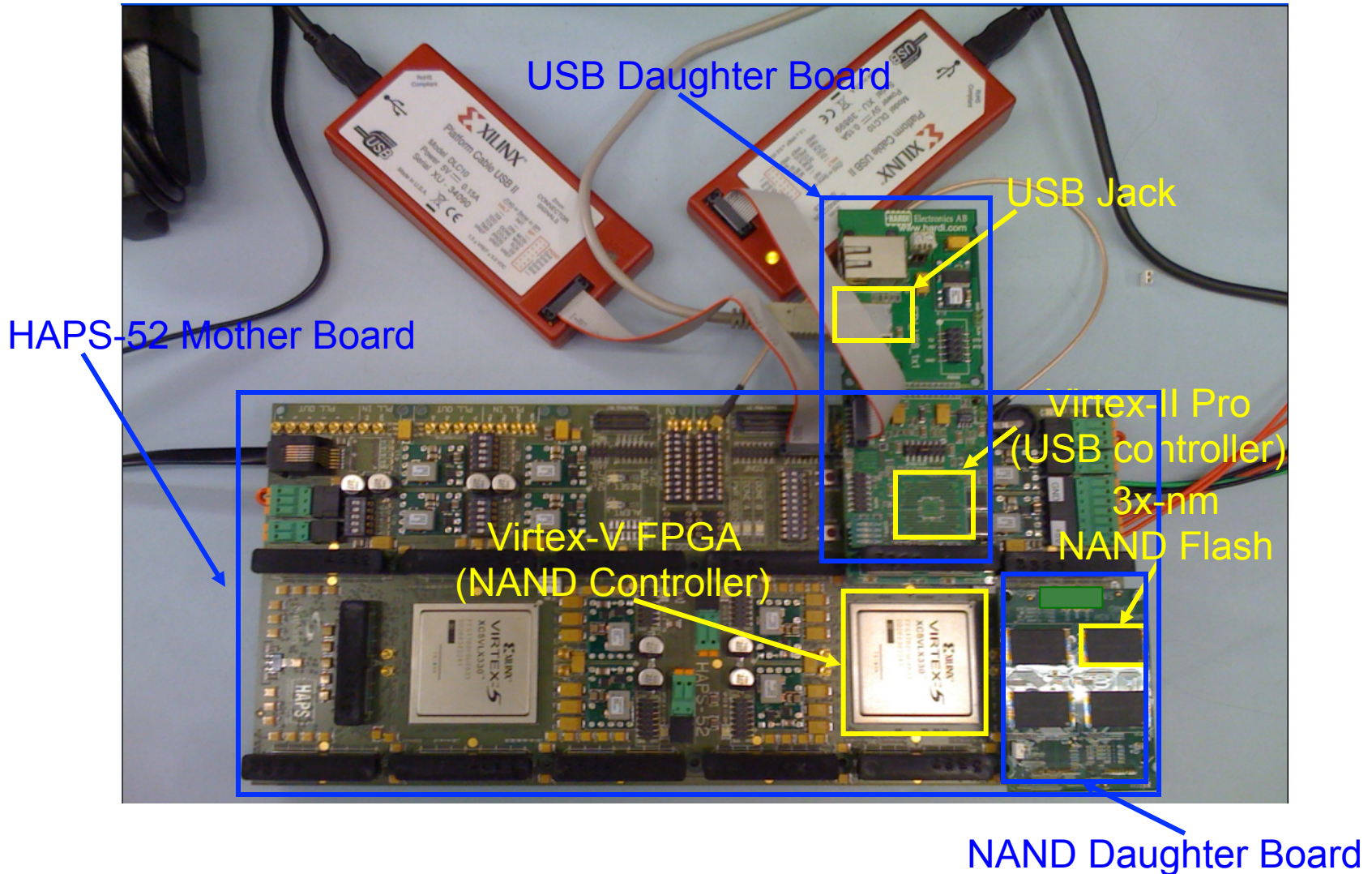
NAND Flash Usage and Error Model
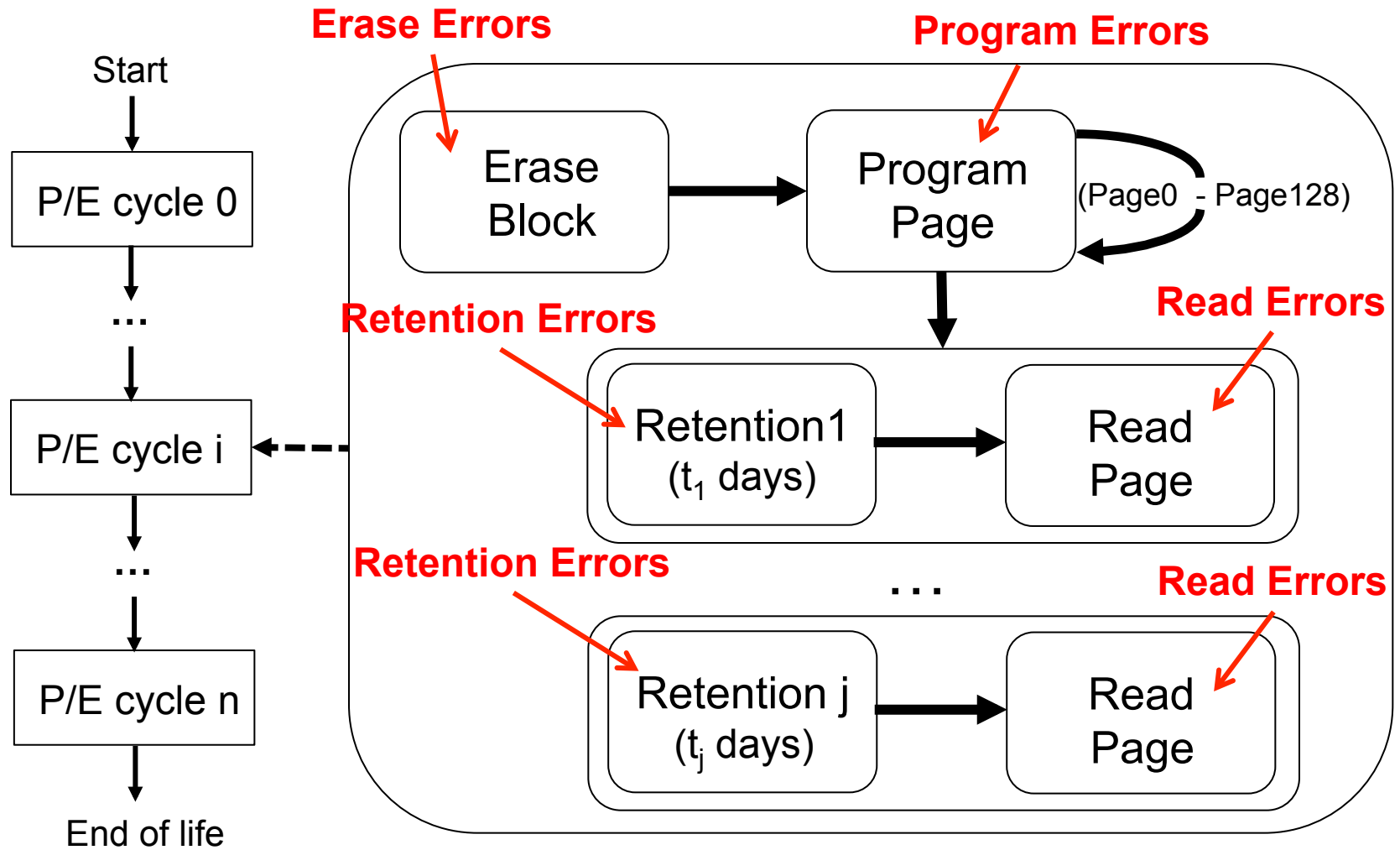
Error Types and Testing Methodology

- Erase errors
  - Count the number of cells that fail to be erased to "11" state

- Program interference errors
  - Compare the data immediately after page programming and the data after the whole block being programmed

- Read errors
  - Continuously read a given block and compare the data between consecutive read sequences

- Retention errors
  - Compare the data read after an amount of time to data written
    - Characterize short term retention errors under room temperature
    - Characterize long term retention errors by baking in the oven under 125℃

Carnegie Mellon
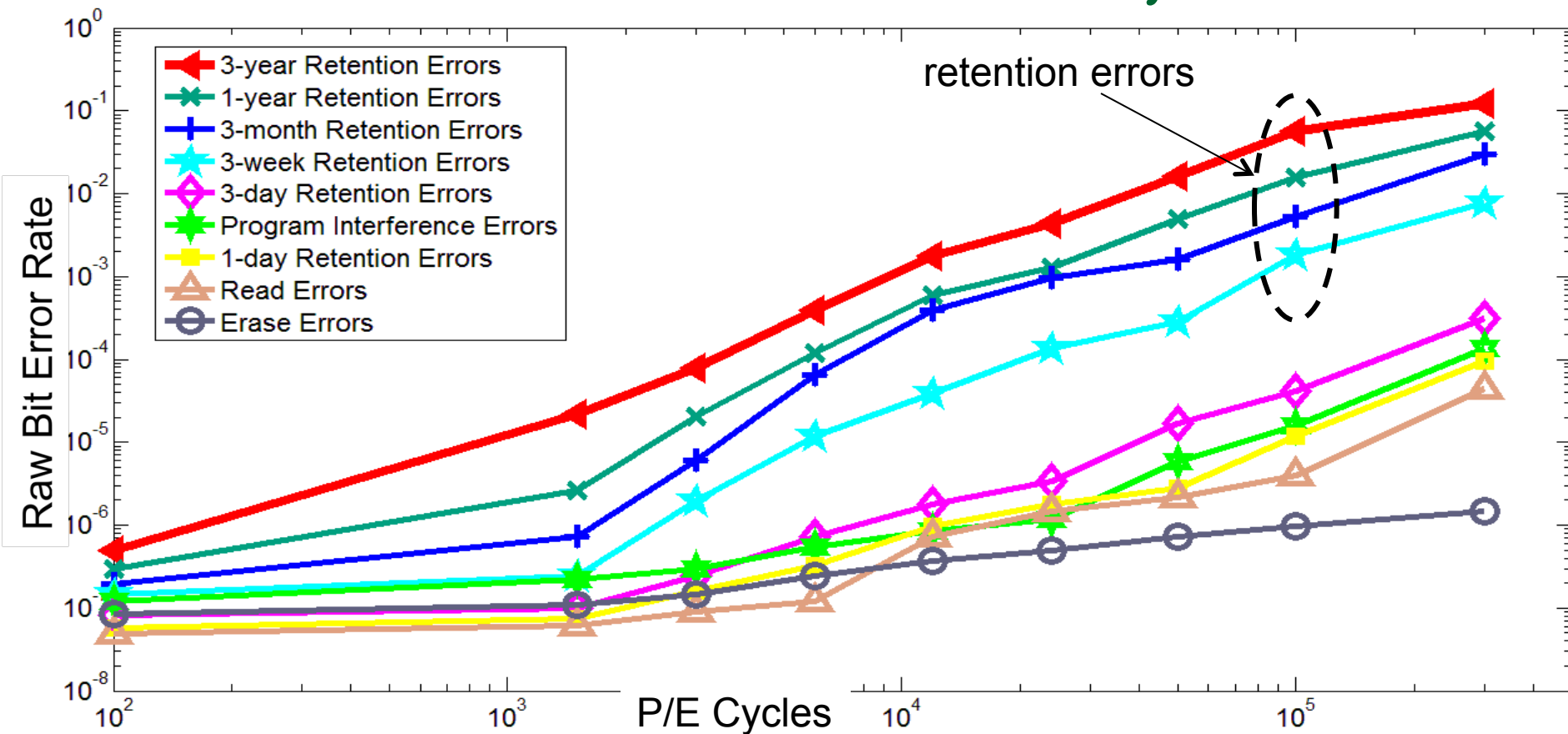
Observations: Flash Error Analysis

- Raw bit error rate increases exponentially with P/E cycles
- Retention errors are dominant (>99% for 1-year ret. time)
- Retention errors increase with retention time requirement

Retention Error Mechanism

LSB/MSB

Stress Induced Leakage Current (SILC)

Floating
Gate

REF1

REF2

REF3

11

10

01

00

V_{th}

Erased

Fully programmed
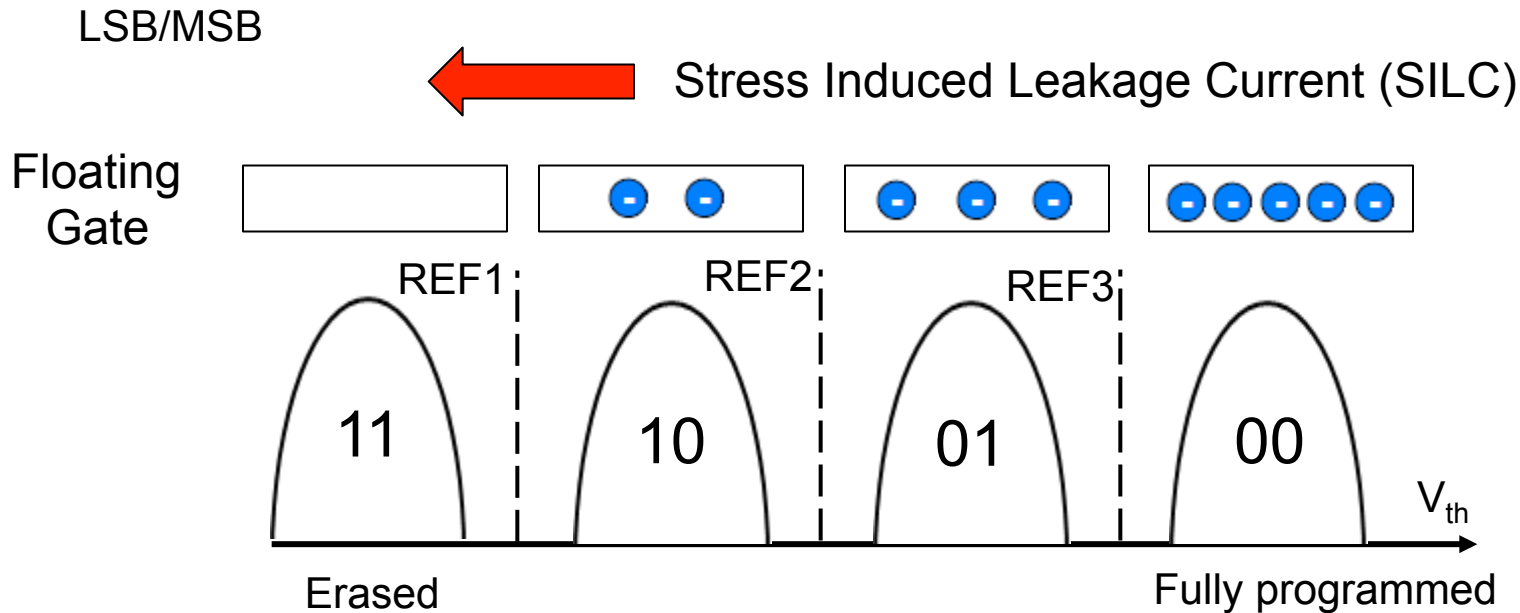
- **Electron loss from the floating gate causes retention errors**
  - ❑ Cells with more programmed electrons suffer more from retention errors
  - ❑ Threshold voltage is more likely to shift by one window than by multiple

Retention Error Value Dependency
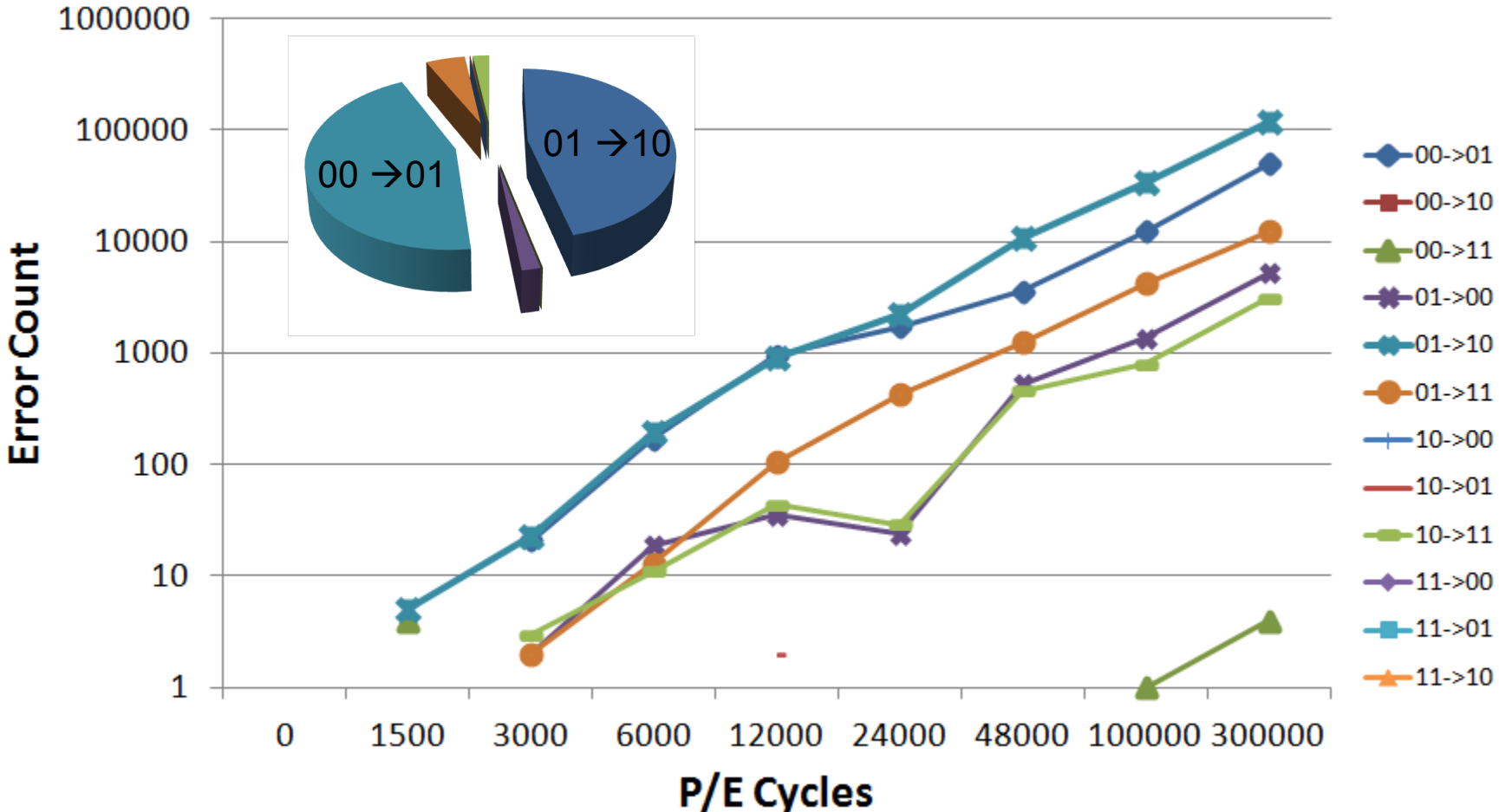
- Cells with more programmed electrons tend to suffer more from retention noise (i.e. 00 and 01)

Carnegie Mellon

More Details on Flash Error Analysis

- Yu Cai, Erich F. Haratsch, <u>Onur Mutlu</u>, 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)

Carnegie Mellon

Threshold Voltage Distribution Shifts
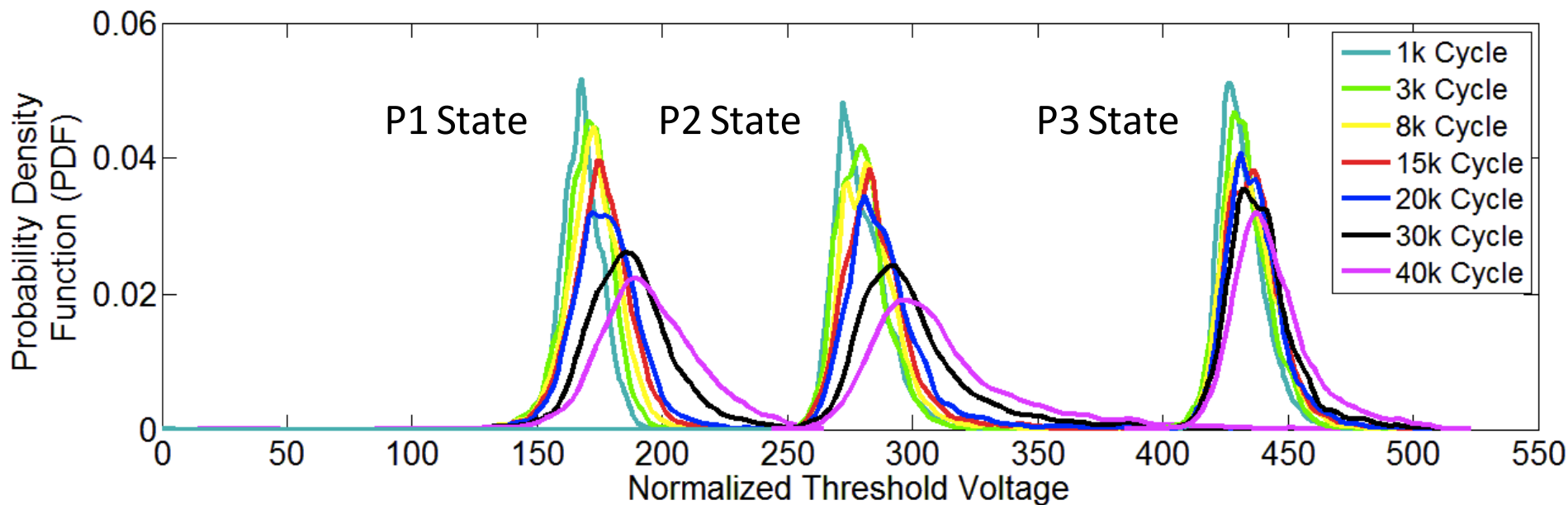
As P/E cycles increase ...

- Distribution shifts to the right
- Distribution becomes wider

More Detail

- Yu Cai, Erich F. Haratsch, <u>Onur Mutlu</u>, 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. <u>Slides (ppt)</u>

Carnegie Mellon

Flash Correct-and-Refresh

Retention-Aware Error Management for Increased Flash Memory Lifetime

Yu Cai[1]   Gulay Yalcin[2]   Onur Mutlu[1]   Erich F. Haratsch[3]

Adrian Cristal[2]   Osman S. Unsal[2]   Ken Mai[1]

[1] Carnegie Mellon University
[2] Barcelona Supercomputing Center
[3] LSI Corporation

SAFARI

Carnegie Mellon

Executive Summary

- 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

Outline

- Executive Summary
- The Problem: Limited Flash Memory Endurance/Lifetime
- Error and ECC Analysis for Flash Memory
- Flash Correct and Refresh Techniques (FCR)
- Evaluation
- Conclusions

SAFARI

Carnegie Mellon

Problem: Limited Endurance of Flash Memory

- **NAND flash has limited endurance**
  - A cell can tolerate a small number of Program/Erase (P/E) cycles
  - 3x-nm flash with 2 bits/cell → 3K P/E cycles

- Enterprise data storage requirements demand very high endurance
  - >50K P/E cycles (10 full disk writes per day for 3-5 years)

- **Continued process scaling and more bits per cell will reduce flash endurance**

- One potential solution: stronger error correction codes (ECC)
  - Stronger ECC not effective enough and inefficient

SAFARI

Carnegie Mellon

Decreasing Endurance with Flash Scaling

P/E Cycle Endurance (y-axis)

- SLC: 100k
- 5x-nm MLC: 10k
- 3x-nm MLC: 5k
- 2x-nm MLC: 3k
- 3-bit-MLC: 1k

4-bit ECC, **8-bit ECC**, **15-bit ECC**, **24-bit ECC**

Error Correction Capability (per 1 kB of data)

Ariel Maislos, "A New Era in Embedded Flash Memory", Flash Summit 2011 (Anobit)

- Endurance of flash memory decreasing with scaling and multi-level cells
- Error correction capability required to guarantee storage-class reliability (UBER < 10^{-15}) is increasing exponentially to reach *less* endurance

UBER: Uncorrectable bit error rate. Fraction of erroneous bits after error correction.

Carnegie Mellon

The Problem with Stronger Error Correction

- Stronger ECC detects and corrects more raw bit errors → increases P/E cycles endured

- Two shortcomings of stronger ECC:

1. High implementation complexity
    - → Power and area overheads increase super-linearly, but correction capability increases sub-linearly with ECC strength

2. Diminishing returns on flash lifetime improvement
    - → Raw bit error rate increases exponentially with P/E cycles, but correction capability increases sub-linearly with ECC strength
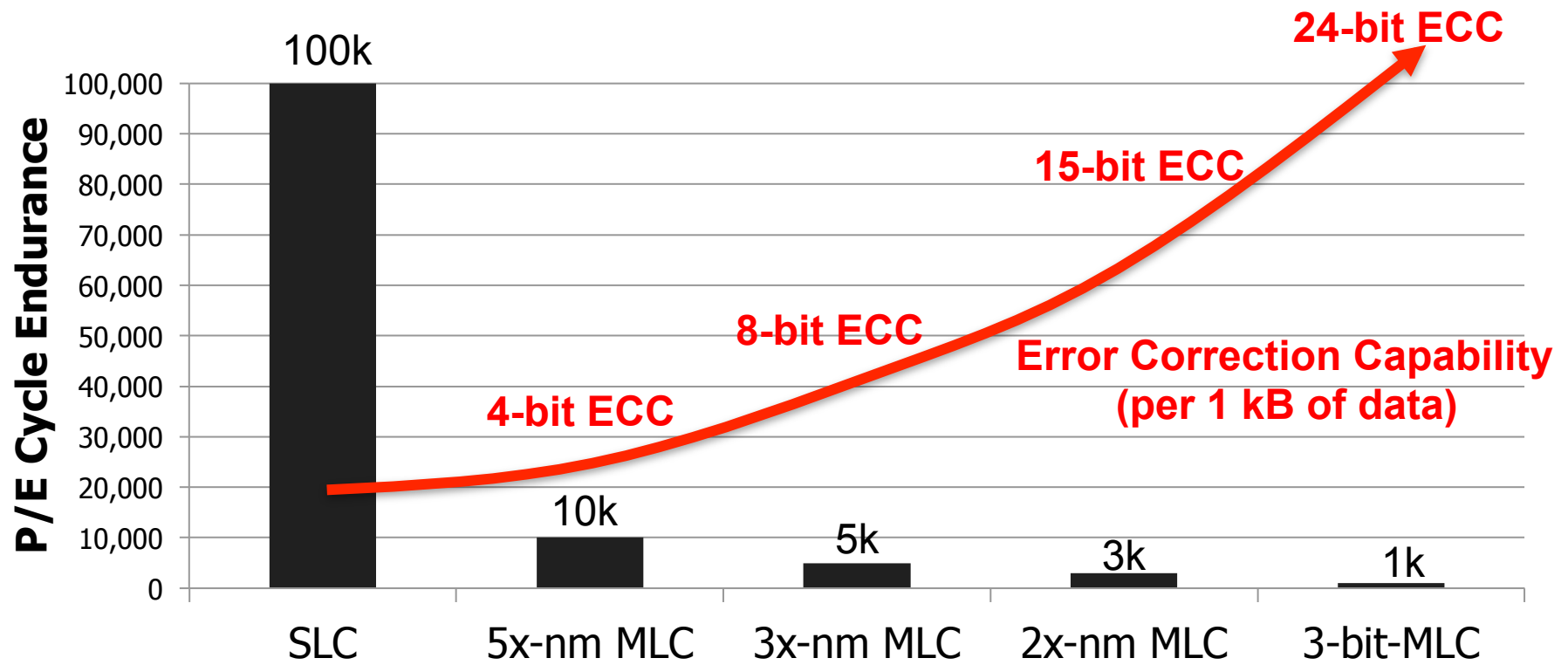
Carnegie Mellon

Outline

- Executive Summary
- The Problem: Limited Flash Memory Endurance/Lifetime
- Error and ECC Analysis for Flash Memory
- Flash Correct and Refresh Techniques (FCR)
- Evaluation
- Conclusions

SAFARI

Carnegie Mellon

Methodology: Error and ECC Analysis

- **Characterized errors and error rates** of 3x-nm MLC NAND flash using an experimental FPGA-based flash platform
  - ❑ Cai et al., "Error Patterns in MLC NAND Flash Memory: Measurement, Characterization, and Analysis," DATE 2012.

- **Quantified Raw Bit Error Rate (RBER) at a given P/E cycle**
  - ❑ Raw Bit Error Rate: Fraction of erroneous bits without any correction

- **Quantified error correction capability** (and area and power consumption) of various BCH-code implementations
  - ❑ Identified how much RBER each code can tolerate
    - → how many P/E cycles (flash lifetime) each code can sustain

SAFARI

Carnegie Mellon

NAND Flash Error Types

- Four types of errors [Cai+, DATE 2012]

- Caused by common flash operations
  - Read errors
  - Erase errors
  - Program (interference) errors

- Caused by flash cell losing charge over time
  - Retention errors
    - Whether an error happens depends on required retention time
    - Especially problematic in MLC flash because voltage threshold window to determine stored value is smaller

SAFARI

Carnegie Mellon

Observations: Flash Error Analysis

- Raw bit error rate increases exponentially with P/E cycles
- Retention errors are dominant (>99% for 1-year ret. time)
- Retention errors increase with retention time requirement

Carnegie Mellon

Methodology: Error and ECC Analysis

- **Characterized errors and error rates** of 3x-nm MLC NAND flash using an experimental FPGA-based flash platform
  - Cai et al., "Error Patterns in MLC NAND Flash Memory: Measurement, Characterization, and Analysis," DATE 2012.

- **Quantified Raw Bit Error Rate (RBER) at a given P/E cycle**
  - Raw Bit Error Rate: Fraction of erroneous bits without any correction

- **Quantified error correction capability** (and area and power consumption) of various BCH-code implementations
  - Identified how much RBER each code can tolerate
    - → how many P/E cycles (flash lifetime) each code can sustain

Carnegie Mellon

ECC Strength Analysis

Error correction capability increases sub-linearly

Power and area overheads increase super-linearly

Code length (n)	Correctable Errors (t)	Acceptable Raw BER	Norm. Power	Norm. Area
512	7	1.0×10^{-4} (1x)	1	1
1024	12	4.0×10^{-4} (4x)	2	2.1
2048	22	1.0×10^{-3} (10x)	4.1	3.9
4096	40	1.7×10^{-3} (17x)	8.6	10.3
8192	74	2.2×10^{-3} (22x)	17.8	21.3
32768	259	2.6×10^{-3} (26x)	71	85

Carnegie Mellon

Resulting Flash Lifetime with Strong ECC

- Lifetime improvement comparison of various BCH codes

4X Lifetime Improvement

71X Power Consumption
85X Area Consumption
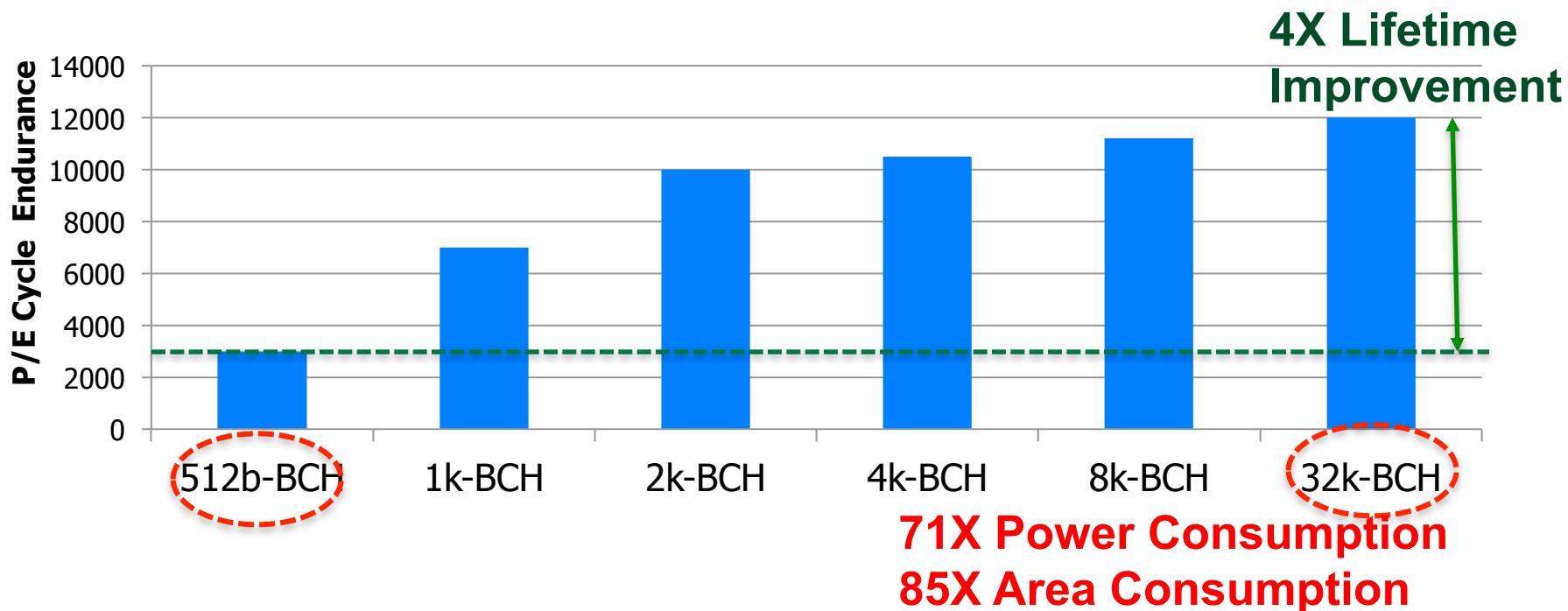
Strong ECC is very inefficient at improving lifetime

SAFARI

Carnegie Mellon

Our Goal

Develop new techniques

to improve flash lifetime

without relying on stronger ECC

SAFARI

Carnegie Mellon

Outline

- Executive Summary
- The Problem: Limited Flash Memory Endurance/Lifetime
- Error and ECC Analysis for Flash Memory
- Flash Correct and Refresh Techniques (FCR)
- Evaluation
- Conclusions

Carnegie Mellon

Flash Correct-and-Refresh (FCR)

- **Key Observations:**
  - ❑ Retention errors are the dominant source of errors in flash memory **[Cai+ DATE 2012][Tanakamaru+ ISSCC 2011]**
    - → limit flash lifetime as they increase over time
  - ❑ Retention errors can be corrected by "refreshing" each flash page periodically

- **Key Idea:**
  - ❑ Periodically read each flash page,
  - ❑ Correct its errors using "weak" ECC, and
  - ❑ Either remap it to a new physical page or reprogram it in-place,
  - ❑ Before the page accumulates more errors than ECC-correctable
  - ❑ Optimization: Adapt refresh rate to endured P/E cycles

Carnegie Mellon

FCR Intuition

Errors with No refresh — Errors with Periodic refresh

Program Page / After time T / After time 2T / After time 3T
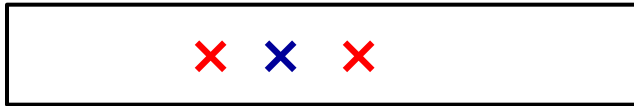
× Retention Error   × Program Error

SAFARI

Carnegie Mellon
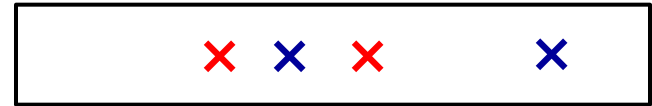
FCR: Two Key Questions

- How to refresh?
  - Remap a page to another one
  - Reprogram a page (in-place)
  - Hybrid of remap and reprogram

- When to refresh?
  - Fixed period
  - Adapt the period to retention error severity

Carnegie Mellon

Outline

- Executive Summary
- The Problem: Limited Flash Memory Endurance/Lifetime
- Error and ECC Analysis for Flash Memory
- Flash Correct and Refresh Techniques (FCR)
  1. Remapping based FCR
  2. Hybrid Reprogramming and Remapping based FCR
  3. Adaptive-Rate FCR
- Evaluation
- Conclusions

Carnegie Mellon

Outline

- Executive Summary
- The Problem: Limited Flash Memory Endurance/Lifetime
- Error and ECC Analysis for Flash Memory
- Flash Correct and Refresh Techniques (FCR)

  1. Remapping based FCR

  2. Hybrid Reprogramming and Remapping based FCR

  3. Adaptive-Rate FCR

- Evaluation
- Conclusions

Carnegie Mellon

Remapping Based FCR

- Idea: Periodically remap each page to a different physical page (after correcting errors)

  - Also **[Pan et al., HPCA 2012]**
  - FTL already has support for changing logical → physical flash block/page mappings
  - Deallocated block is erased by garbage collector
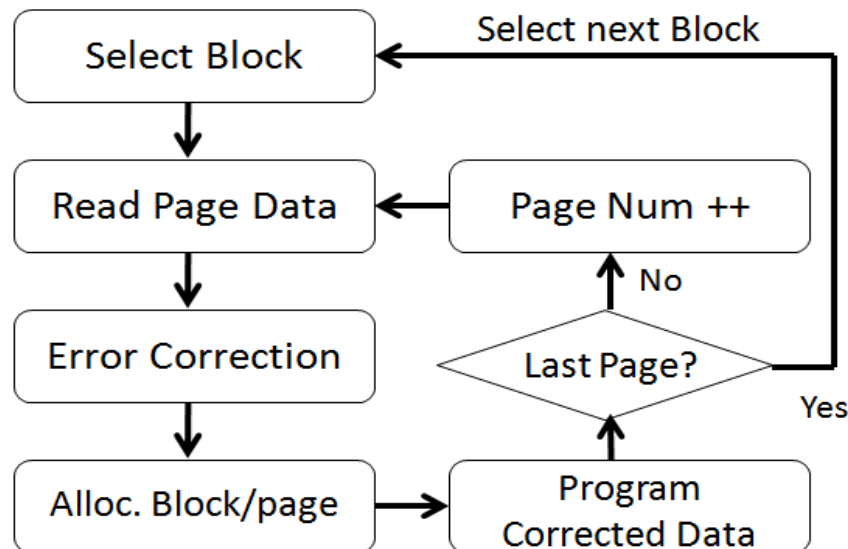
- Problem: Causes additional erase operations → more wearout
  - Bad for read-intensive workloads (few erases really needed)
  - Lifetime degrades for such workloads (see paper)

Carnegie Mellon

Outline

- **Executive Summary**
- **The Problem: Limited Flash Memory Endurance/Lifetime**
- **Error and ECC Analysis for Flash Memory**
- **Flash Correct and Refresh Techniques (FCR)**
  1. Remapping based FCR
  2. Hybrid Reprogramming and Remapping based FCR
  3. Adaptive-Rate FCR
- **Evaluation**
- **Conclusions**

SAFARI

Carnegie Mellon

In-Place Reprogramming Based FCR

- **Idea:** Periodically reprogram (in-place) each physical page (after correcting errors)

  - Flash programming techniques (ISPP) can correct retention errors in-place by recharging flash cells
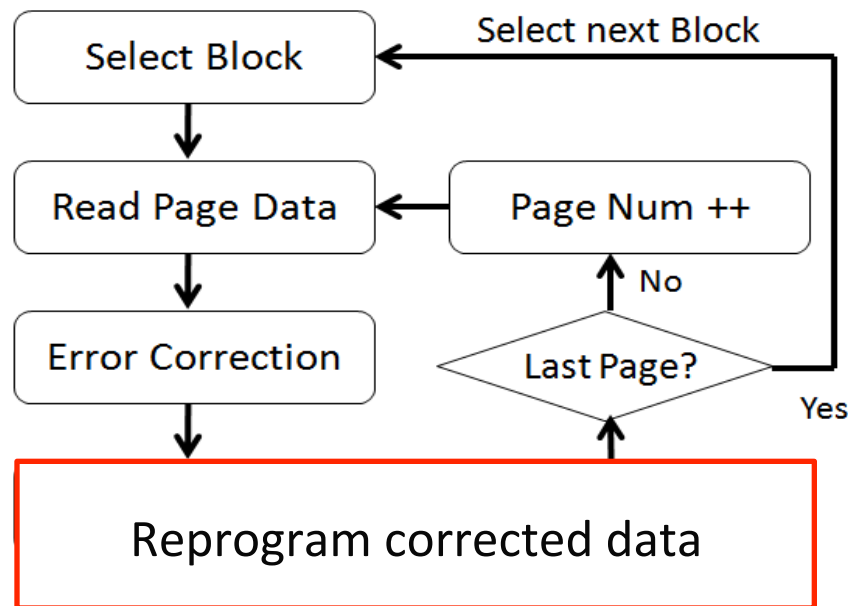
- **Problem:** Program errors accumulate on the same page → may not be correctable by ECC after some time

Carnegie Mellon

In-Place Reprogramming of Flash Cells

Floating Gate

Floating Gate
Voltage Distribution
for each Stored Value

REF1   REF2   REF3

11   10   01   00   VT
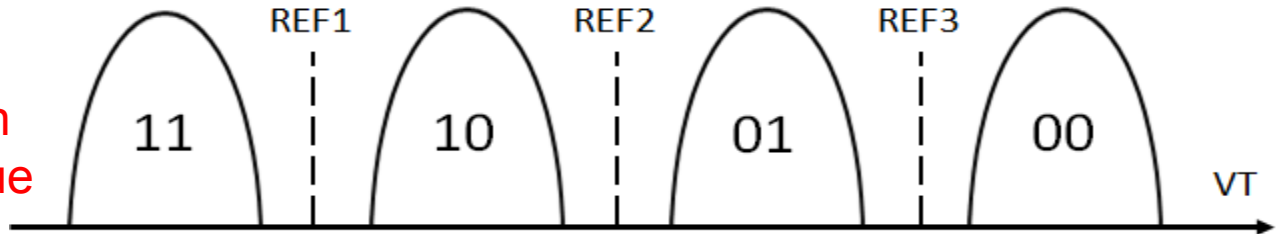
Retention errors are caused by cell voltage shifting to the left

REF1   REF2   REF3

11   10   01   00   VT
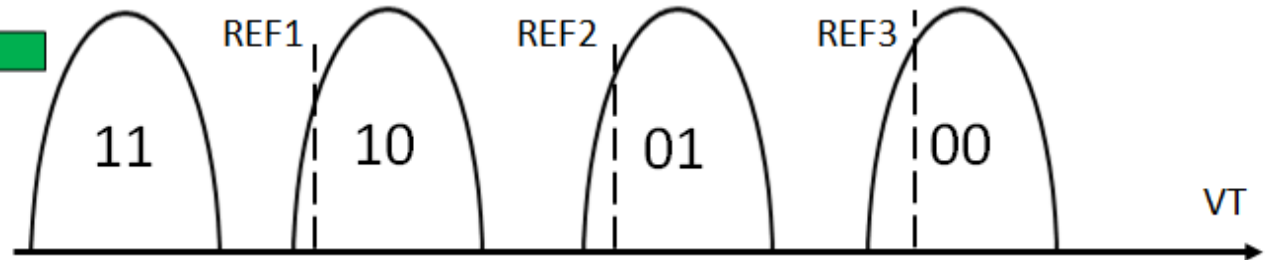
ISPP moves cell voltage to the right; fixes retention errors

11   10   01   00   VT
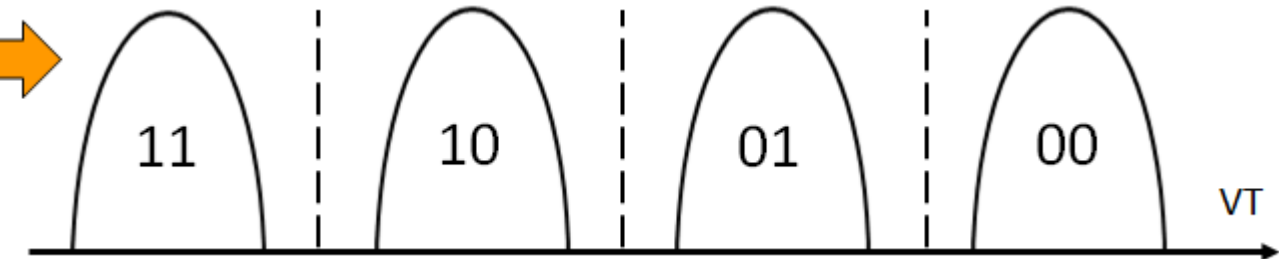
- **Pro: No remapping needed → no additional erase operations**
- **Con: Increases the occurrence of program errors**

Carnegie Mellon

Program Errors in Flash Memory

- When a cell is being programmed, voltage level of a neighboring cell changes (unintentionally) due to parasitic capacitance coupling

    → can change the data value stored

- Also called program interference error

- Program interference causes neighboring cell voltage to shift to the right

Problem with In-Place Reprogramming

Floating Gate

Floating Gate Voltage Distribution

REF1   REF2   REF3

11   10   01   00

Additional Electrons Injected

VT

Original data to be programmed
… | 00 | 11 | 01 | 00 | 10 | 11 | 00 | …

Program errors after initial programming
… | 00 | **10** | 01 | 00 | 10 | 11 | 00 | …

Retention errors after some time
… | **01** | **10** | **10** | 00 | **11** | 11 | **01** | …

Errors after in-place reprogramming
… | 00 | **10** | 01 | 00 | 10 | **10** | 00 | …

1. Read data
2. Correct errors
3. Reprogram back
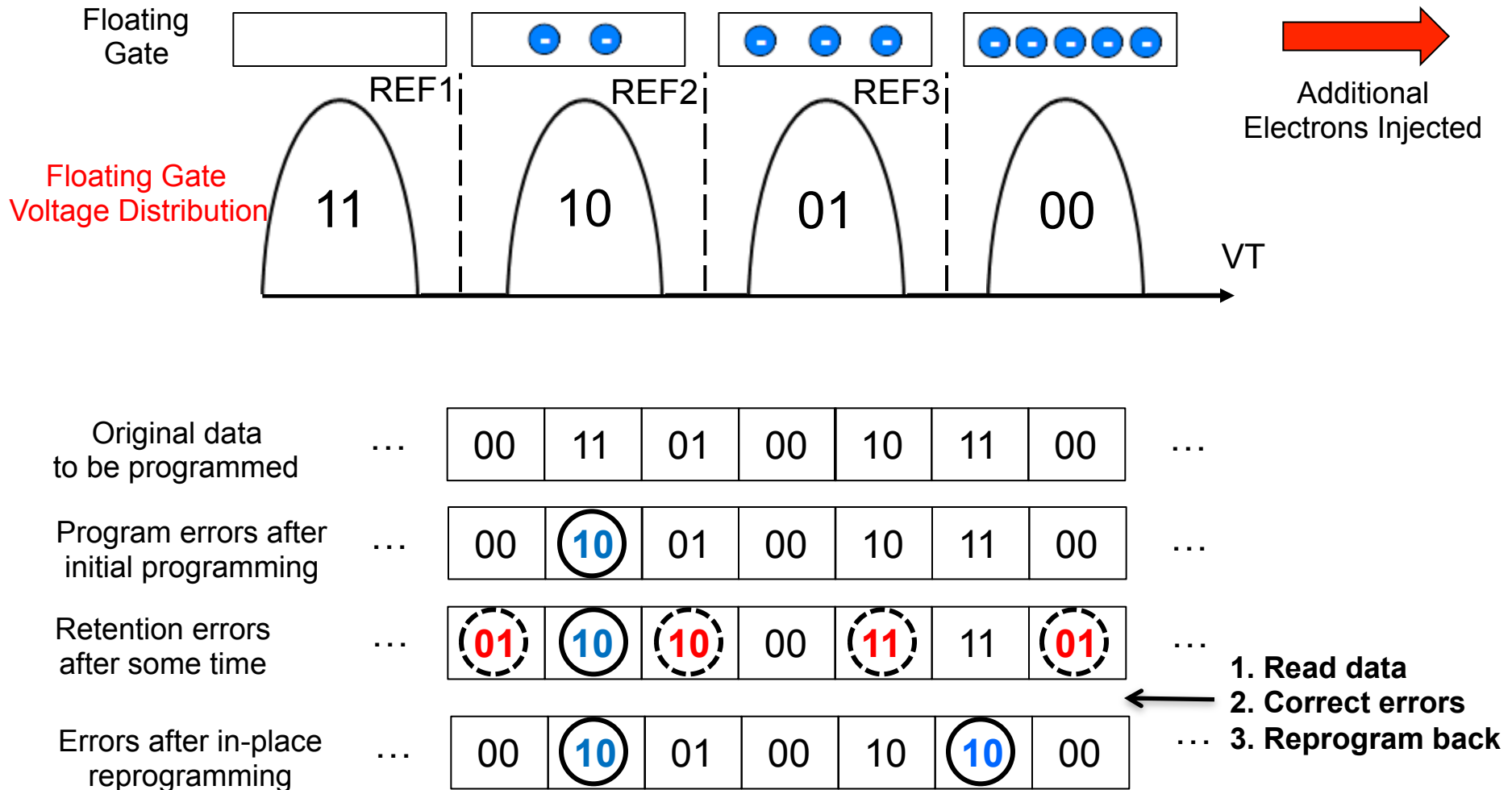
Problem: Program errors can accumulate over time

SAFARI

Carnegie Mellon

349

Hybrid Reprogramming/Remapping Based FCR

- Idea:
  - Monitor the count of right-shift errors (after error correction)
  - If count < threshold, in-place reprogram the page
  - Else, remap the page to a new page

- Observation:
  - Program errors much less frequent than retention errors → Remapping happens only infrequently

- Benefit:
  - Hybrid FCR greatly reduces erase operations due to remapping

Carnegie Mellon

Outline

- Executive Summary
- The Problem: Limited Flash Memory Endurance/Lifetime
- Error and ECC Analysis for Flash Memory
- Flash Correct and Refresh Techniques (FCR)
  1. Remapping based FCR
  2. Hybrid Reprogramming and Remapping based FCR
  3. Adaptive-Rate FCR
- Evaluation
- Conclusions

Carnegie Mellon

Adaptive-Rate FCR

- **Observation:**
  - ❏ Retention error rate strongly depends on the P/E cycles a flash page endured so far
  - ❏ No need to refresh frequently (at all) early in flash lifetime

- **Idea:**
  - ❏ Adapt the refresh rate to the P/E cycles endured by each page
  - ❏ Increase refresh rate gradually with increasing P/E cycles

- **Benefits:**
  - ❏ Reduces overhead of refresh operations
  - ❏ Can use existing FTL mechanisms that keep track of P/E cycles

SAFARI

CarnegieMellon

Adaptive-Rate FCR (Example)

Select refresh frequency such that error rate is below acceptable rate

Outline

- Executive Summary
- The Problem: Limited Flash Memory Endurance/Lifetime
- Error and ECC Analysis for Flash Memory
- Flash Correct and Refresh Techniques (FCR)

  1. Remapping based FCR

  2. Hybrid Reprogramming and Remapping based FCR

  3. Adaptive-Rate FCR

- Evaluation
- Conclusions

Carnegie Mellon

FCR: Other Considerations

- **Implementation cost**
  - No hardware changes
  - FTL software/firmware needs modification

- **Response time impact**
  - FCR not as frequent as DRAM refresh; low impact

- **Adaptation to variations in retention error rate**
  - Adapt refresh rate based on, e.g., temperature **[Liu+ ISCA 2012]**

- **FCR requires power**
  - Enterprise storage systems typically powered on

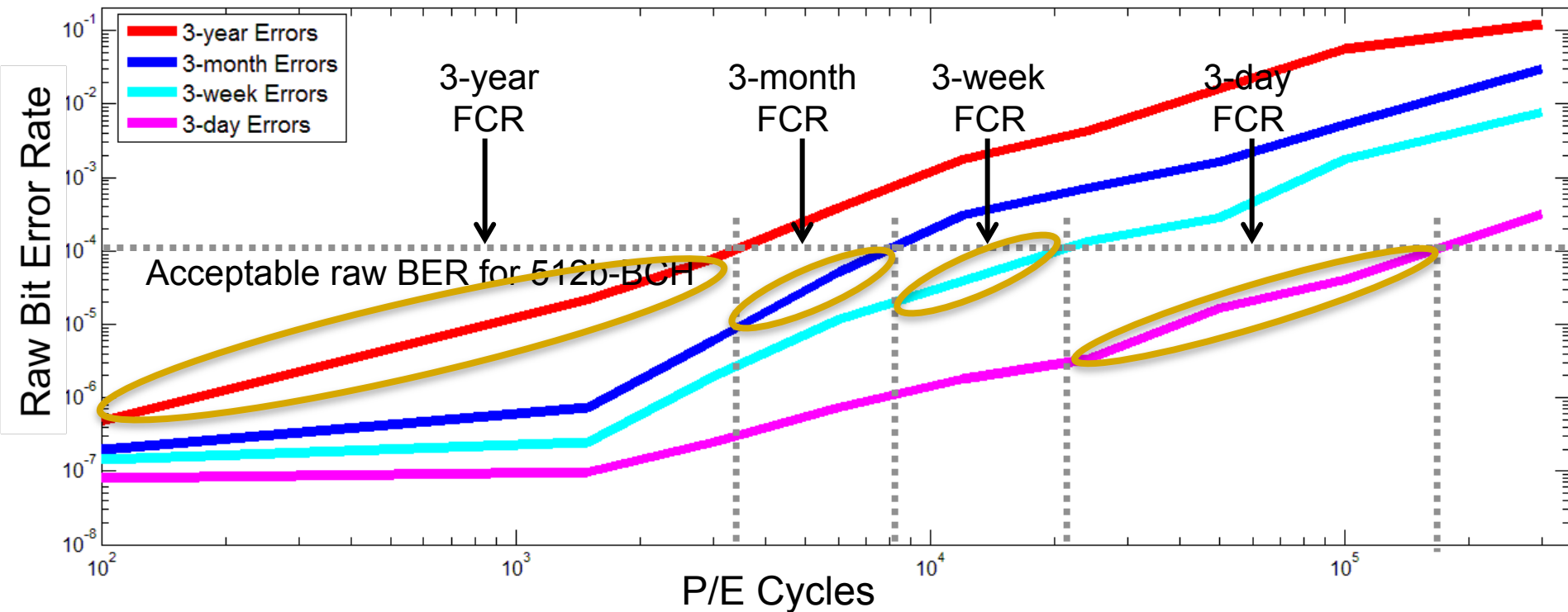*SAFARI*

Carnegie Mellon

Outline

- Executive Summary
- The Problem: Limited Flash Memory Endurance/Lifetime
- Error and ECC Analysis for Flash Memory
- Flash Correct and Refresh Techniques (FCR)
- Evaluation
- Conclusions

SAFARI

Carnegie Mellon

Evaluation Methodology

- **Experimental flash platform** to obtain error rates at different P/E cycles **[Cai+ DATE 2012]**

- **Simulation framework** to obtain P/E cycles of real workloads: DiskSim with SSD extensions

- **Simulated system:** 256GB flash, 4 channels, 8 chips/channel, 8K blocks/chip, 128 pages/block, 8KB pages

- **Workloads**
  - File system applications, databases, web search
  - Categories: Write-heavy, read-heavy, balanced

- **Evaluation metrics**
  - Lifetime (extrapolated)
  - Energy overhead, P/E cycle overhead

Carnegie Mellon

Extrapolated Lifetime

Obtained from Experimental Platform Data

$$\frac{\text{Maximum full disk P/E Cycles for a Technique}}{\text{Total full disk P/E Cycles for a Workload}} \times \text{\# of Days of Given Application}$$

Obtained from Workload Simulation

Real length (in time) of each workload trace

SAFARI

Carnegie Mellon
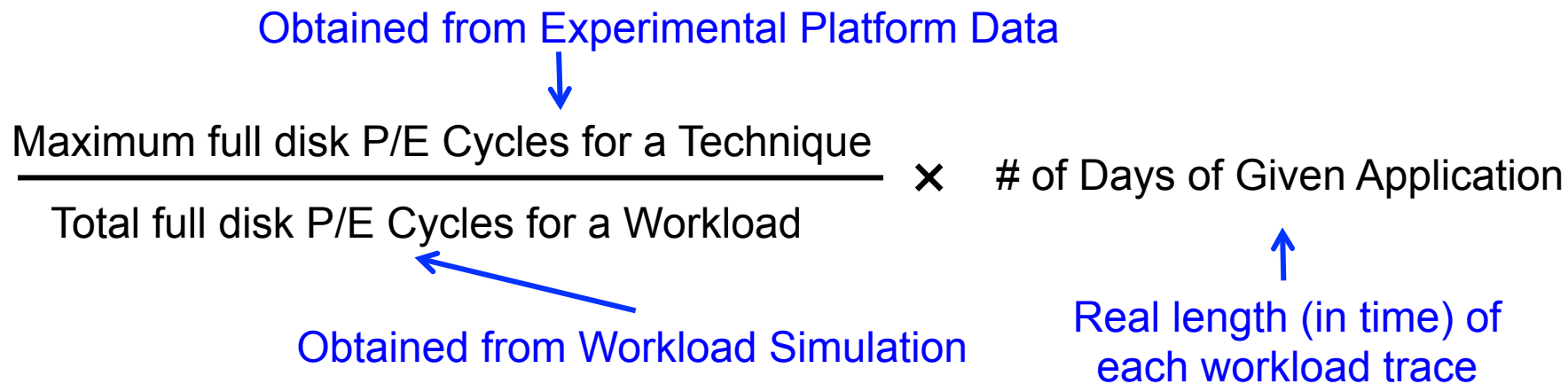
Normalized Flash Memory Lifetime
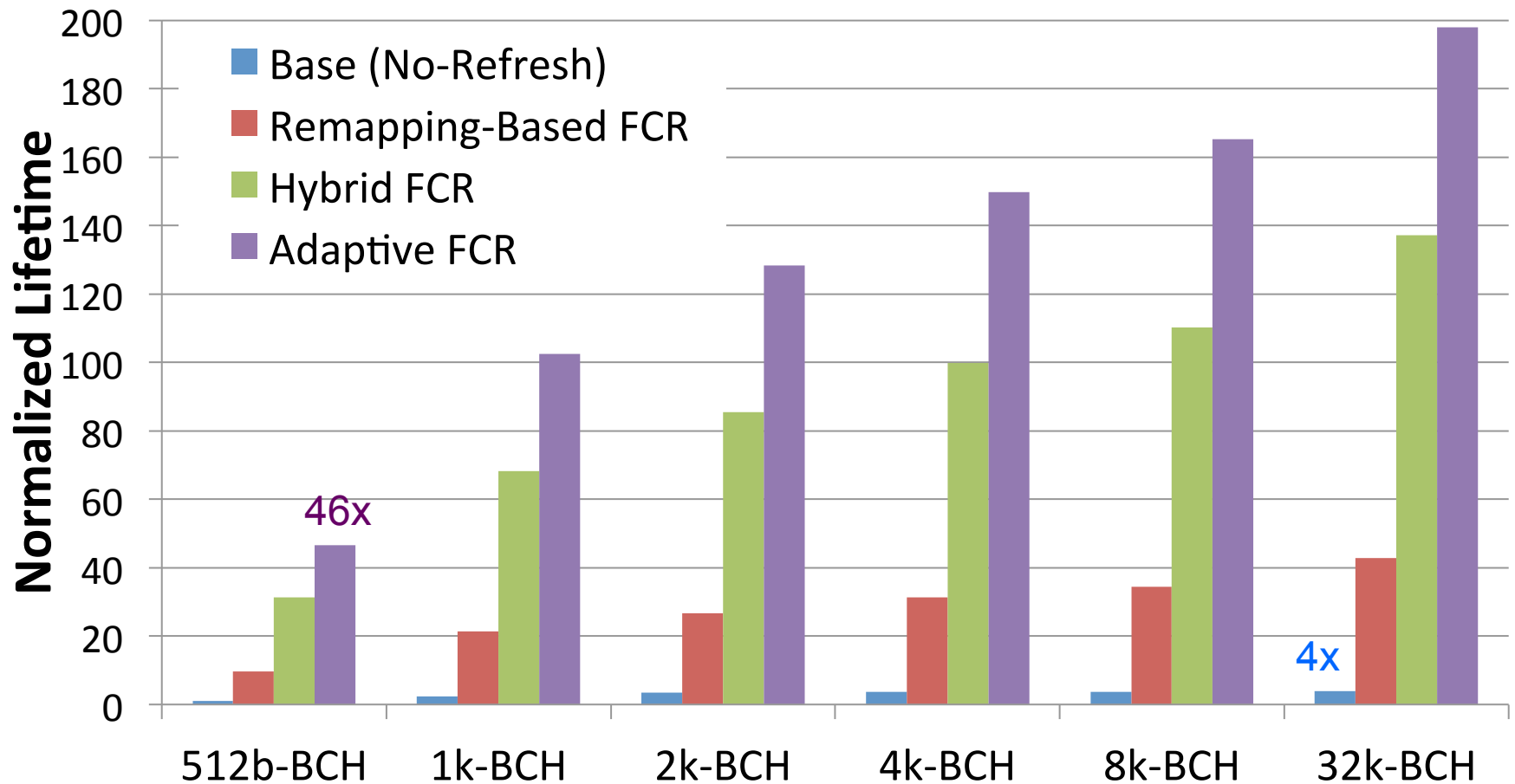
Lifetime of FCR much higher than lifetime of stronger ECC

SAFARI

Carnegie Mellon

Lifetime Evaluation Takeaways

- **Significant average lifetime improvement over no refresh**
  - Adaptive-rate FCR: 46X
  - Hybrid reprogramming/remapping based FCR: 31X
  - Remapping based FCR: 9X

- **FCR lifetime improvement larger than that of stronger ECC**
  - 46X vs. 4X with 32-kbit ECC (over 512-bit ECC)
  - FCR is less complex and less costly than stronger ECC

- **Lifetime on all workloads improves with Hybrid FCR**
  - Remapping based FCR can degrade lifetime on read-heavy WL
  - Lifetime improvement highest in write-heavy workloads

SAFARI

Carnegie Mellon
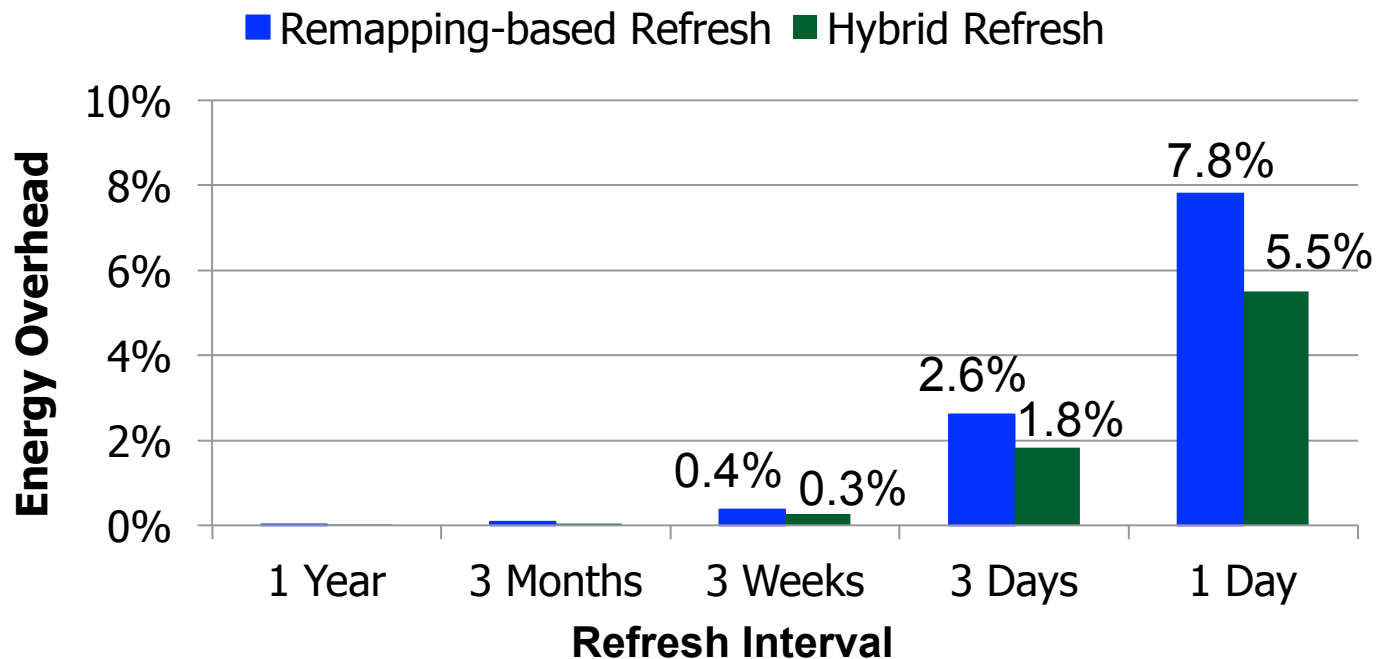
Energy Overhead

- **Adaptive-rate refresh:** <1.8% energy increase until daily refresh is triggered

SAFARI

Carnegie Mellon

Overhead of Additional Erases

- Additional erases happen due to remapping of pages

- Low (2%-20%) for write intensive workloads
- High (up to 10X) for read-intensive workloads

- Improved P/E cycle lifetime of all workloads largely outweighs the additional P/E cycles due to remapping

SAFARI

Carnegie Mellon

More Results in the Paper

- Detailed workload analysis

- Effect of refresh rate

Outline

- Executive Summary
- The Problem: Limited Flash Memory Endurance/Lifetime
- Error and ECC Analysis for Flash Memory
- Flash Correct and Refresh Techniques (FCR)
- Evaluation
- Conclusions

Conclusion

- NAND flash memory lifetime is limited due to uncorrectable errors, which increase over lifetime (P/E cycles)

- Observation: Dominant source of errors in flash memory is retention errors → retention error rate limits lifetime

- Flash Correct-and-Refresh (FCR) techniques reduce retention error rate to improve flash lifetime
  - Periodically read, correct, and remap or reprogram each page before it accumulates more errors than can be corrected
  - Adapt refresh period to the severity of errors

- FCR improves flash lifetime by 46X at no hardware cost
  - More effective and efficient than stronger ECC
  - Can enable better flash memory scaling

SAFARI

Carnegie Mellon

Flash Correct-and-Refresh

Retention-Aware Error Management for Increased Flash Memory Lifetime

Yu Cai[1]   Gulay Yalcin[2]   Onur Mutlu[1]   Erich F. Haratsch[3]
Adrian Cristal[2]   Osman S. Unsal[2]   Ken Mai[1]

[1] Carnegie Mellon University
[2] Barcelona Supercomputing Center
[3] LSI Corporation

SAFARI

Carnegie Mellon

Unused Slides
(from Longer Version)

An Example Problem: Shared Main Memory

Multi-Core Chip

SHARED L3 CACHE

CORE 0

L2 CACHE 0

L2 CACHE 1

CORE 1

DRAM MEMORY CONTROLLER

DRAM INTERFACE

CORE 2

L2 CACHE 2

L2 CACHE 3

CORE 3

DRAM BANKS
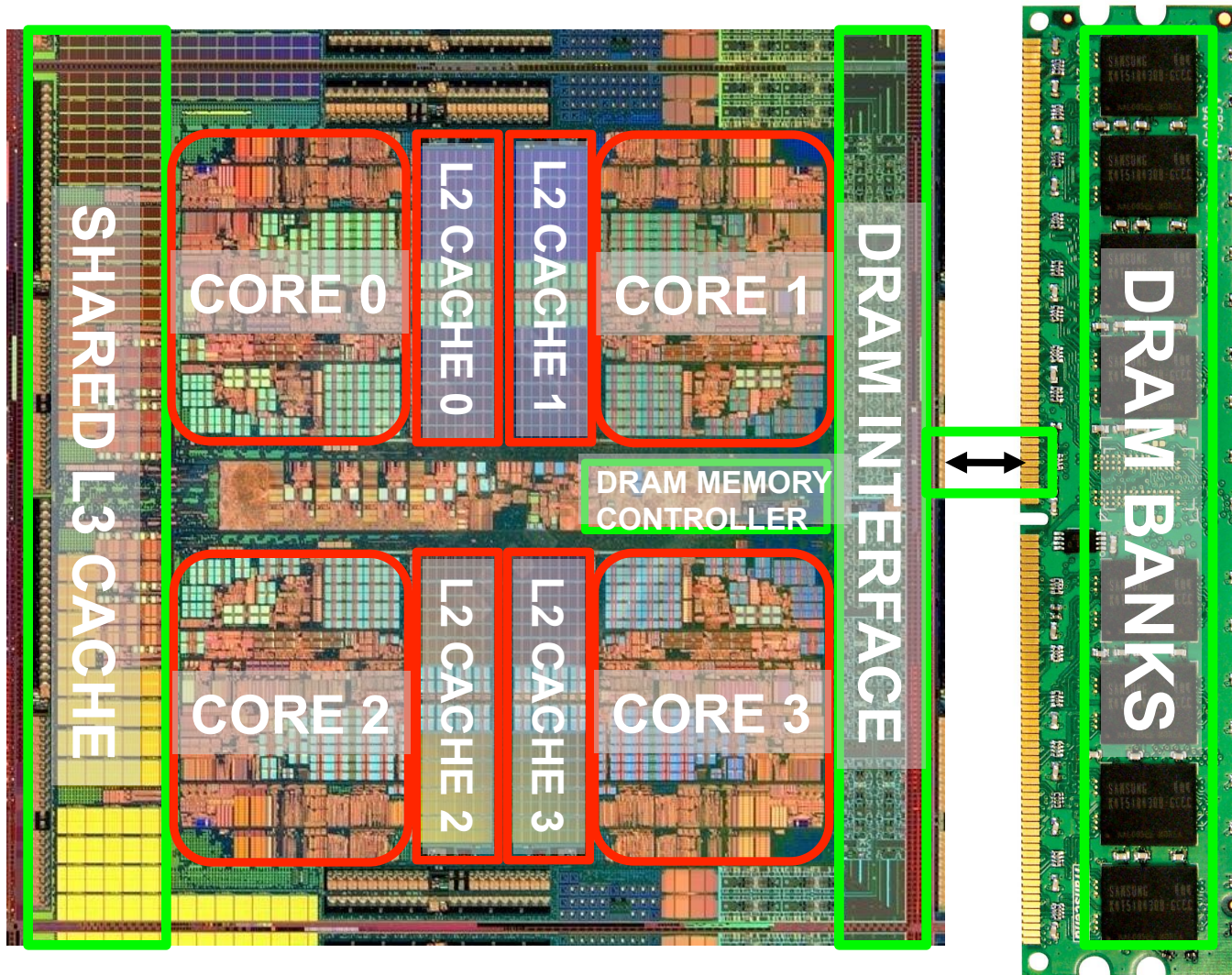
*Die photo credit: AMD Barcelona

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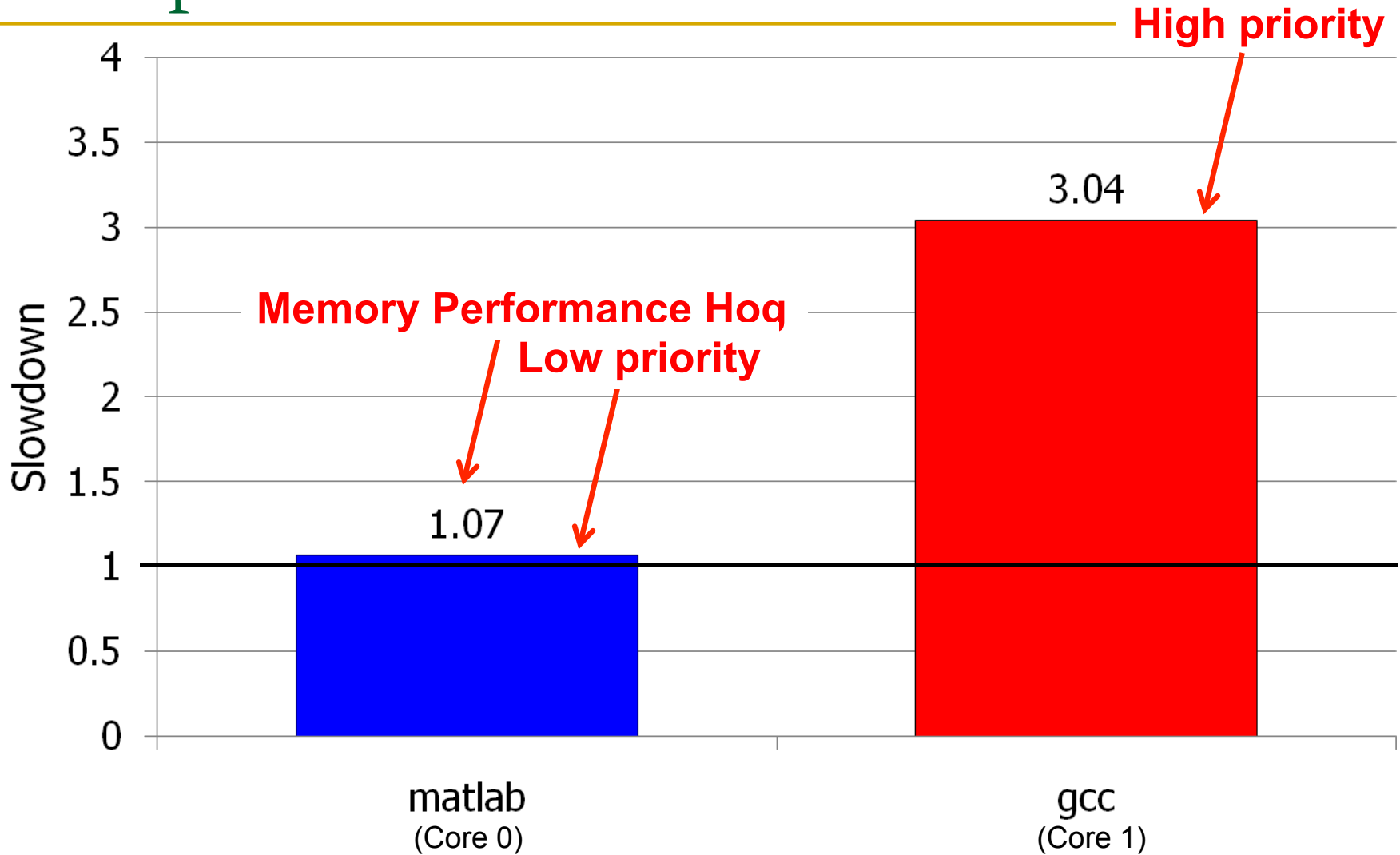
Moscibroda and Mutlu, "Memory performance attacks: Denial of memory service in multi-core systems," USENIX Security 2007.

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?
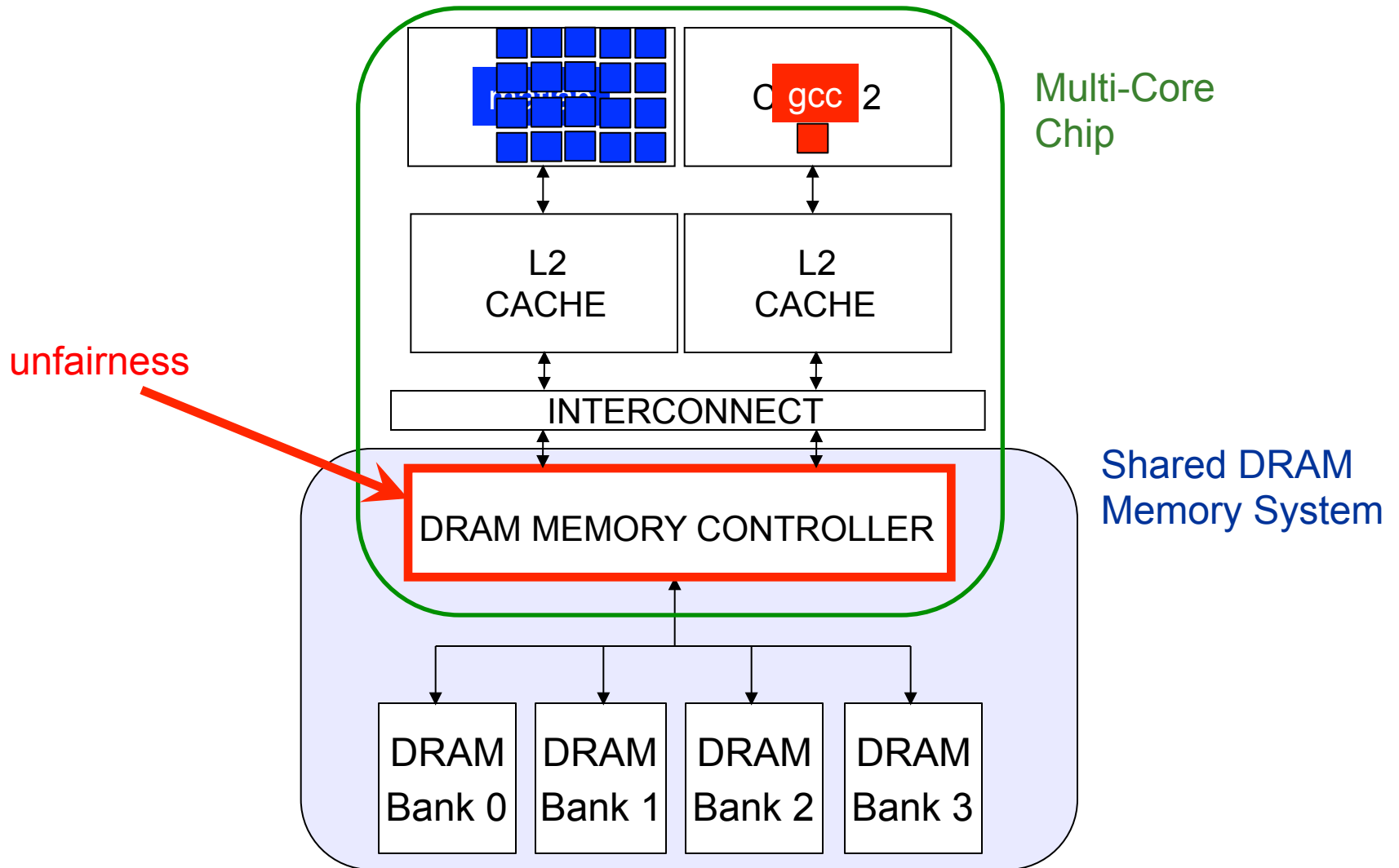
Multi-Core Chip

Shared DRAM Memory System

unfairness

DRAM MEMORY CONTROLLER

INTERCONNECT

L2 CACHE

L2 CACHE

DRAM Bank 0

DRAM Bank 1

DRAM Bank 2

DRAM Bank 3

DRAM Bank Operation

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

Columns

Rows

Row decoder

Row address 0 1

Row Buffer CONFLICT HIT !

Row 1

Column address 0 85

Column mux

Data
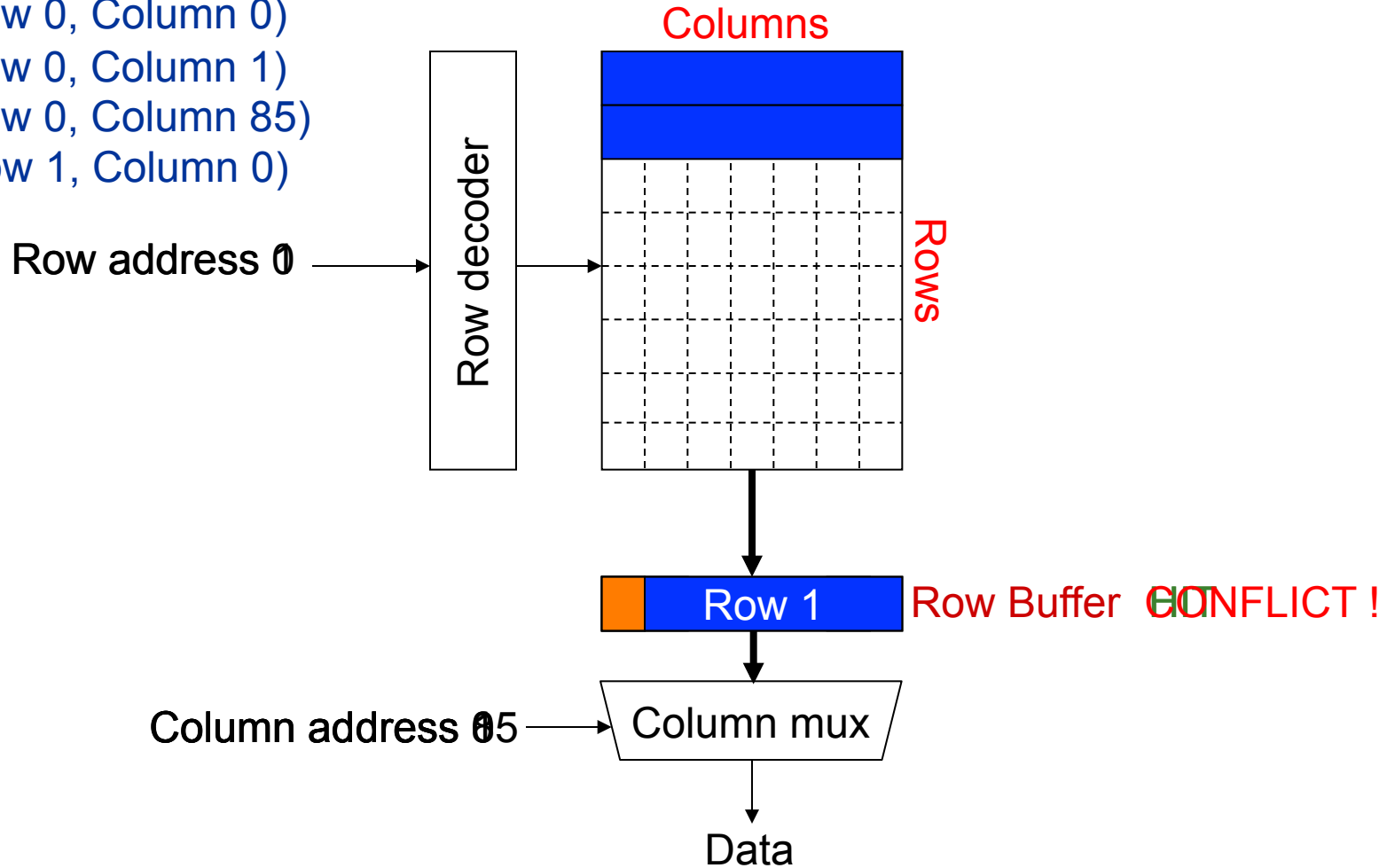
DRAM Controllers

- 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.

SRAM (Static Random Access Memory)
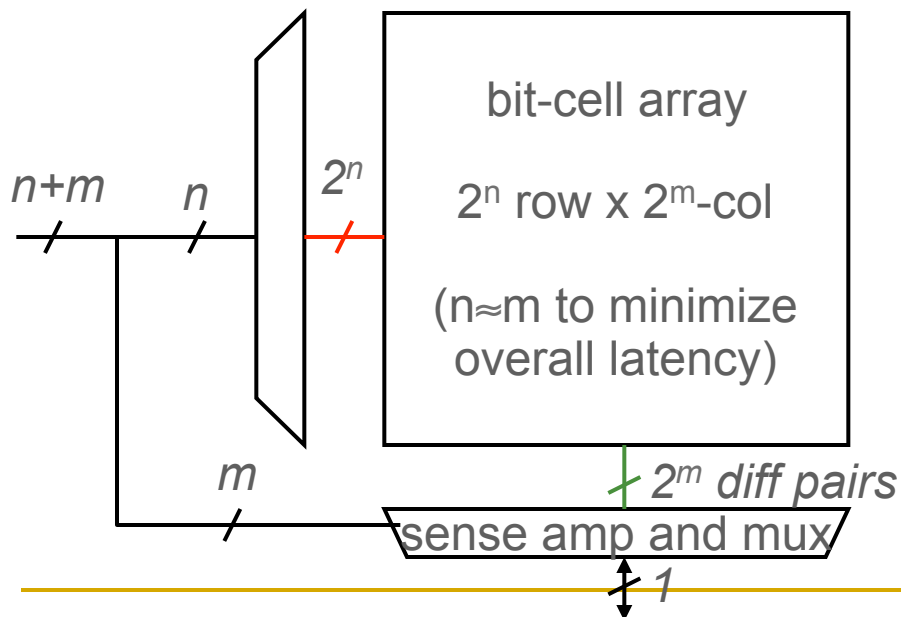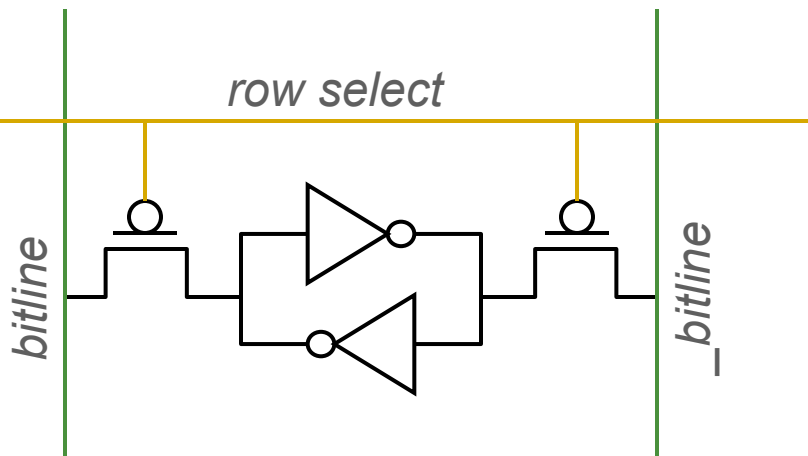
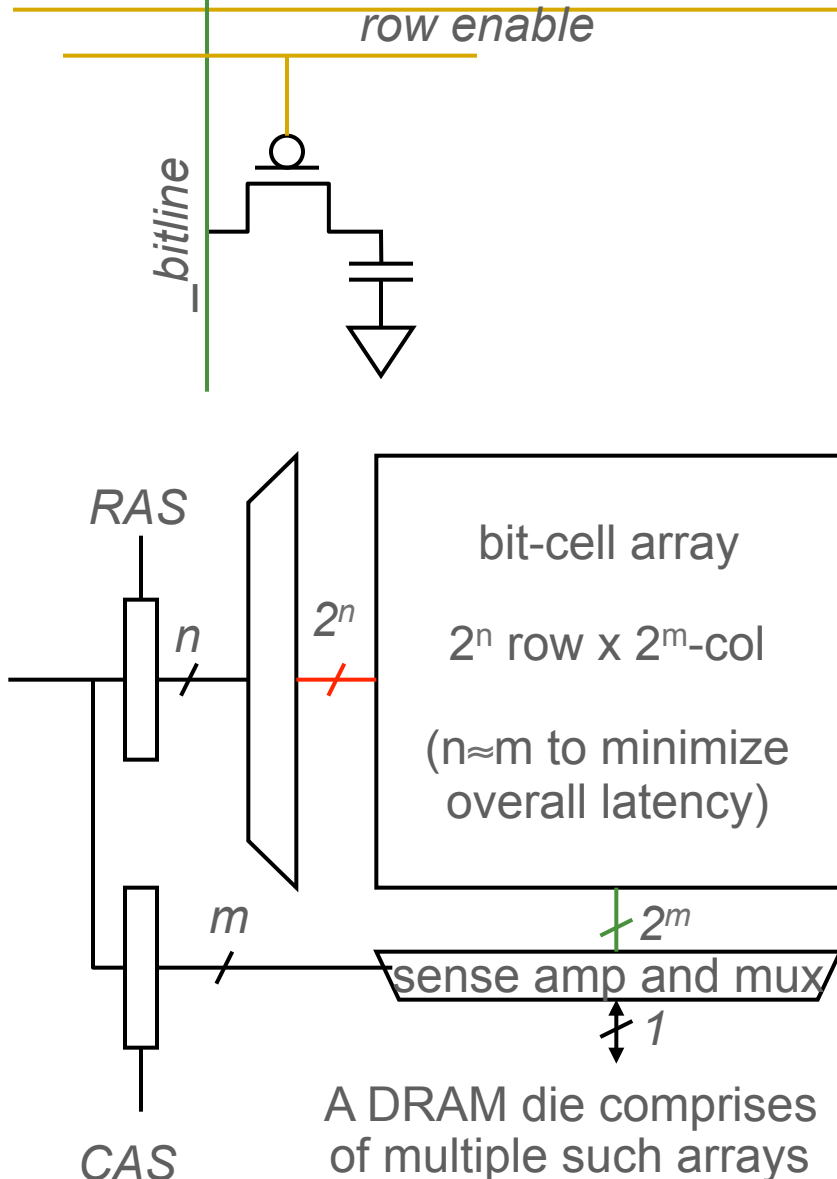Read Sequence

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)

row enable

bitline

RAS

n

2^n

bit-cell array

2^n row x 2^m-col

(n≈m to minimize overall latency)

m

2^m

sense amp and mux

1

CAS

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
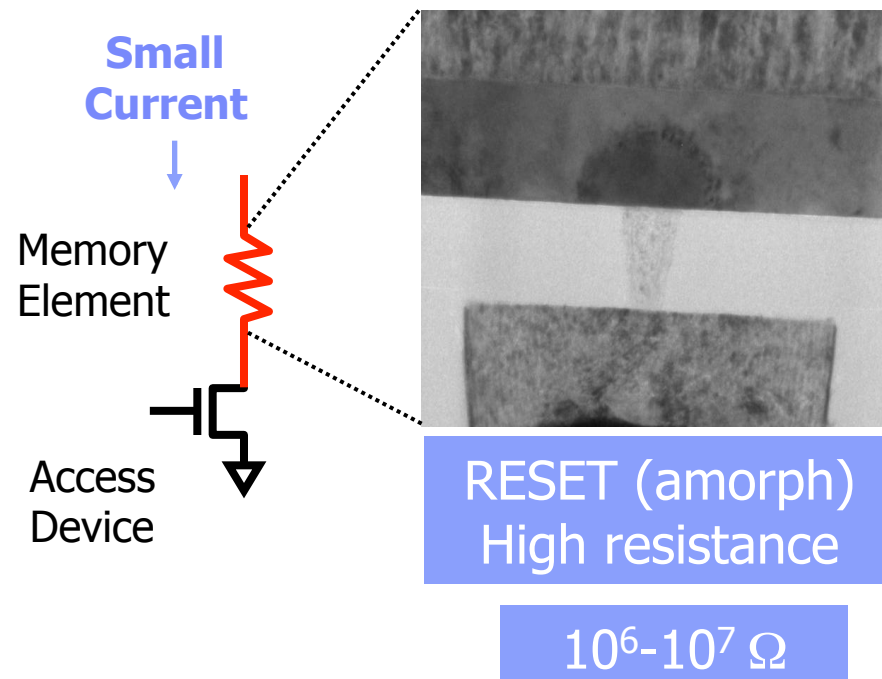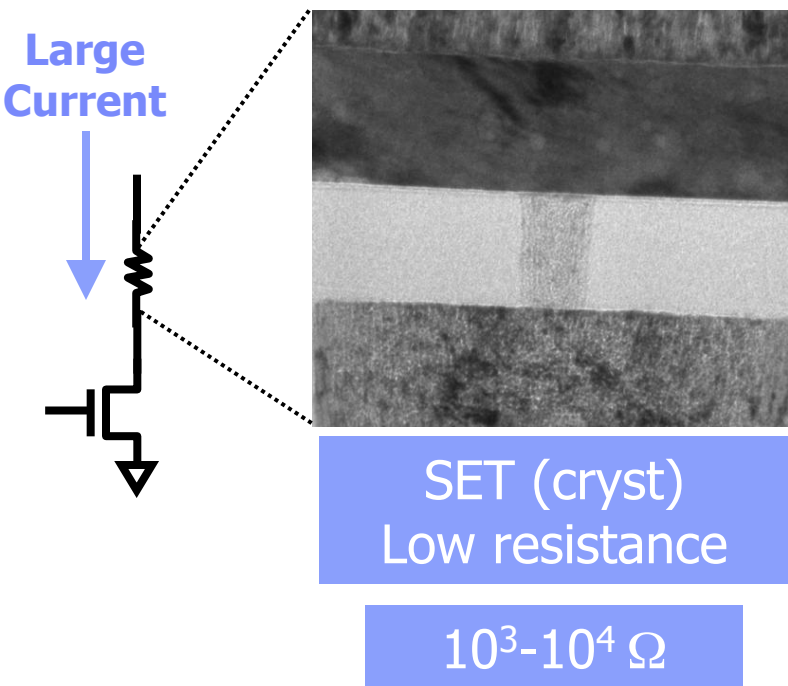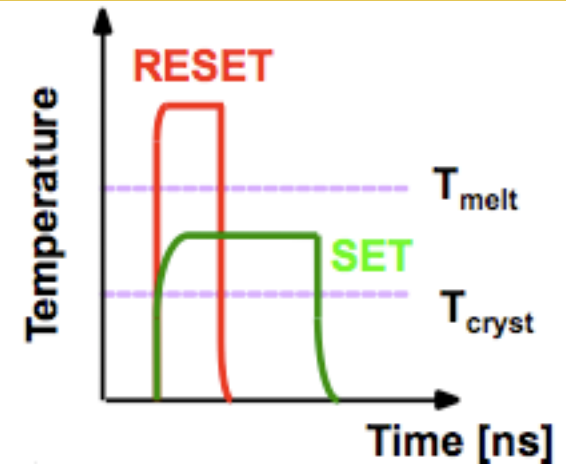 - Crystalline: High optical reflexivity and low electrical resistivity
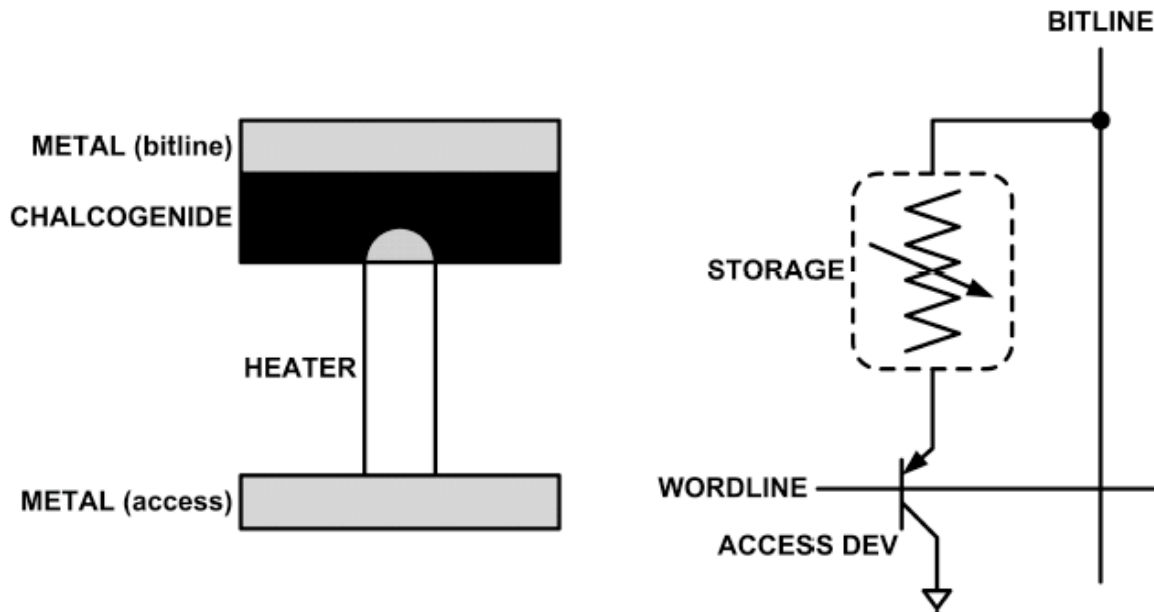
PCM is resistive memory: High resistance (0), Low resistance (1)

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

Small Current

Memory Element

Access Device

SET (cryst)
Low resistance

10^3-10^4 Ω

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)

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

SAFARI

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

SAFARI

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

SAFARI

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

SAFARI

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

SAFARI

Goal: Ultra-efficient heterogeneous architectures
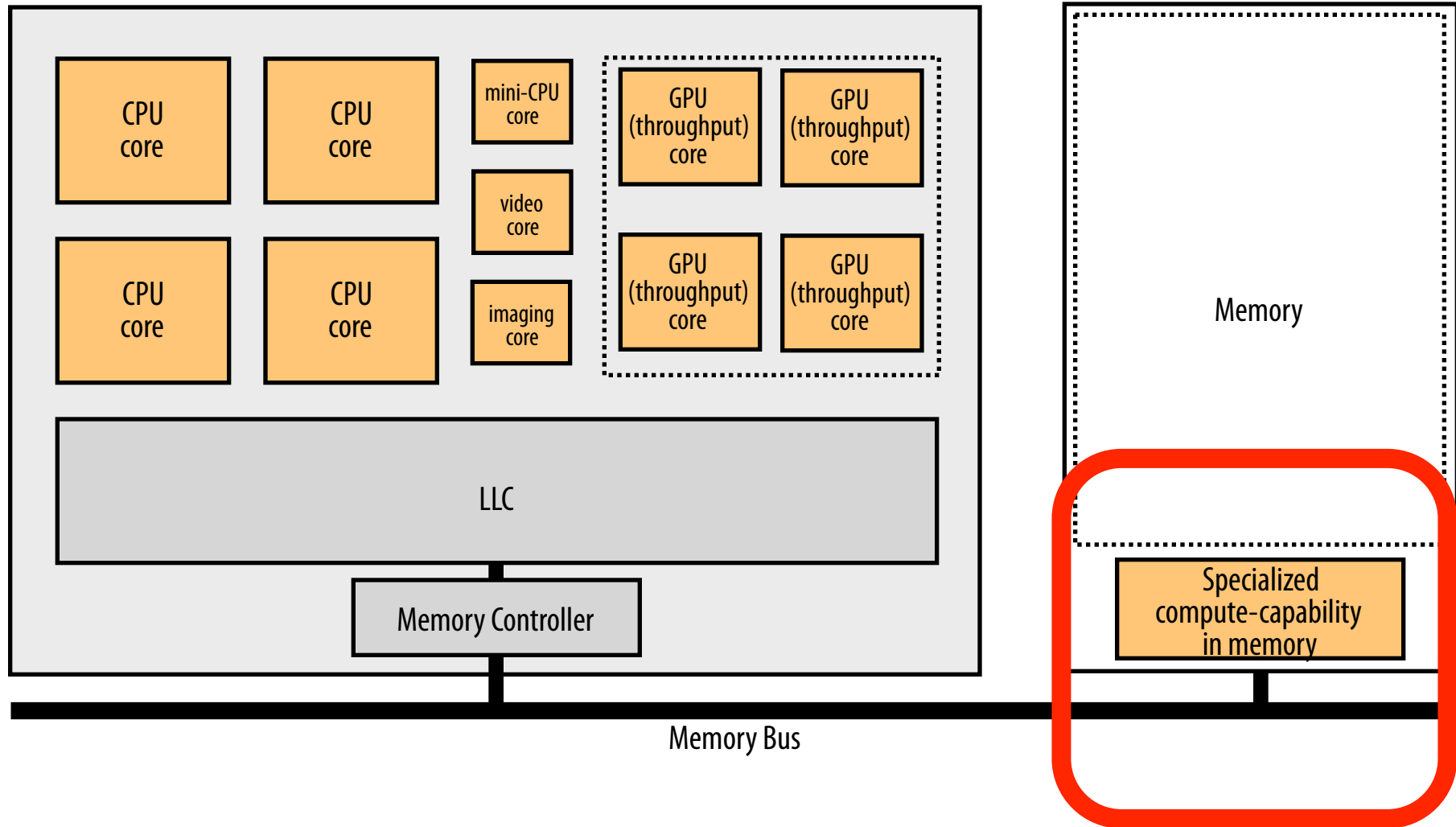
Enabling Ultra-efficient (Visual) Search
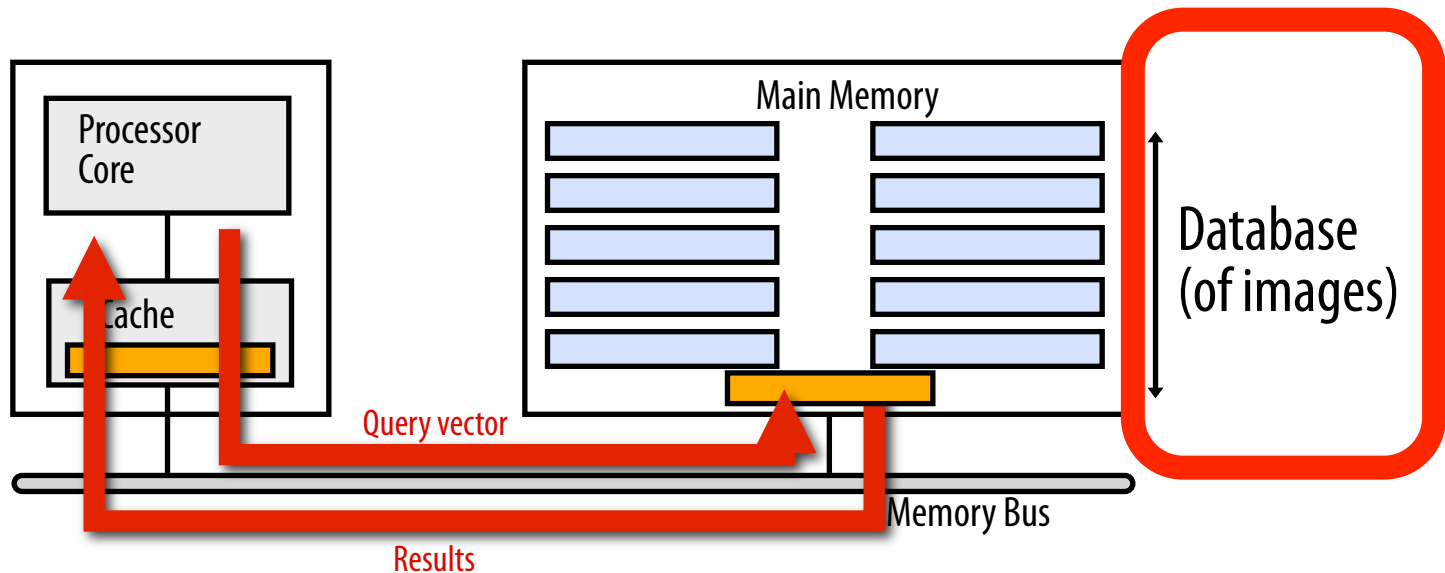
- What is the right partitioning of computation capability?

- 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