A Fresh Look at DRAM Architecture: New Techniques to Improve DRAM Latency, Parallelism, and Energy Efficiency

Onur Mutlu
onur@cmu.edu
July 4, 2013
INRIA

Carnegie Mellon

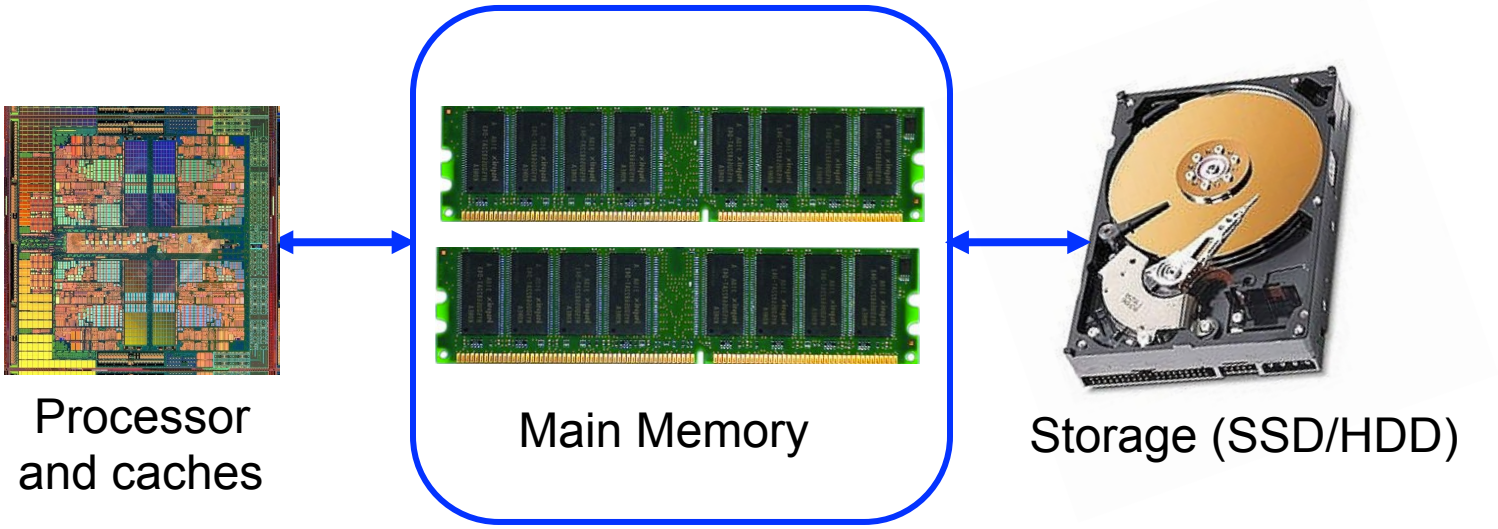
Video Lectures on Same Topics

- Videos from a similar series of lectures at Bogazici University (these are longer)

- http://www.youtube.com/playlist?list=PLVngZ7BemHHV6N0ejHhwOfLwTr8Q-UKXj

- DRAM Basics and DRAM Scaling Lectures
  - http://www.youtube.com/watch?v=jX6McDvAIn4&list=PLVngZ7BemHHV6N0ejHhwOfLwTr8Q-UKXj&index=6
  - http://www.youtube.com/watch?v=E0GuX12dnVo&list=PLVngZ7BemHHV6N0ejHhwOfLwTr8Q-UKXj&index=7
  - http://www.youtube.com/watch?v=ANskLp74Z2k&list=PLVngZ7BemHHV6N0ejHhwOfLwTr8Q-UKXj&index=8
  - http://www.youtube.com/watch?v=gzjaNUYxfFo&list=PLVngZ7BemHHV6N0ejHhwOfLwTr8Q-UKXj&index=9

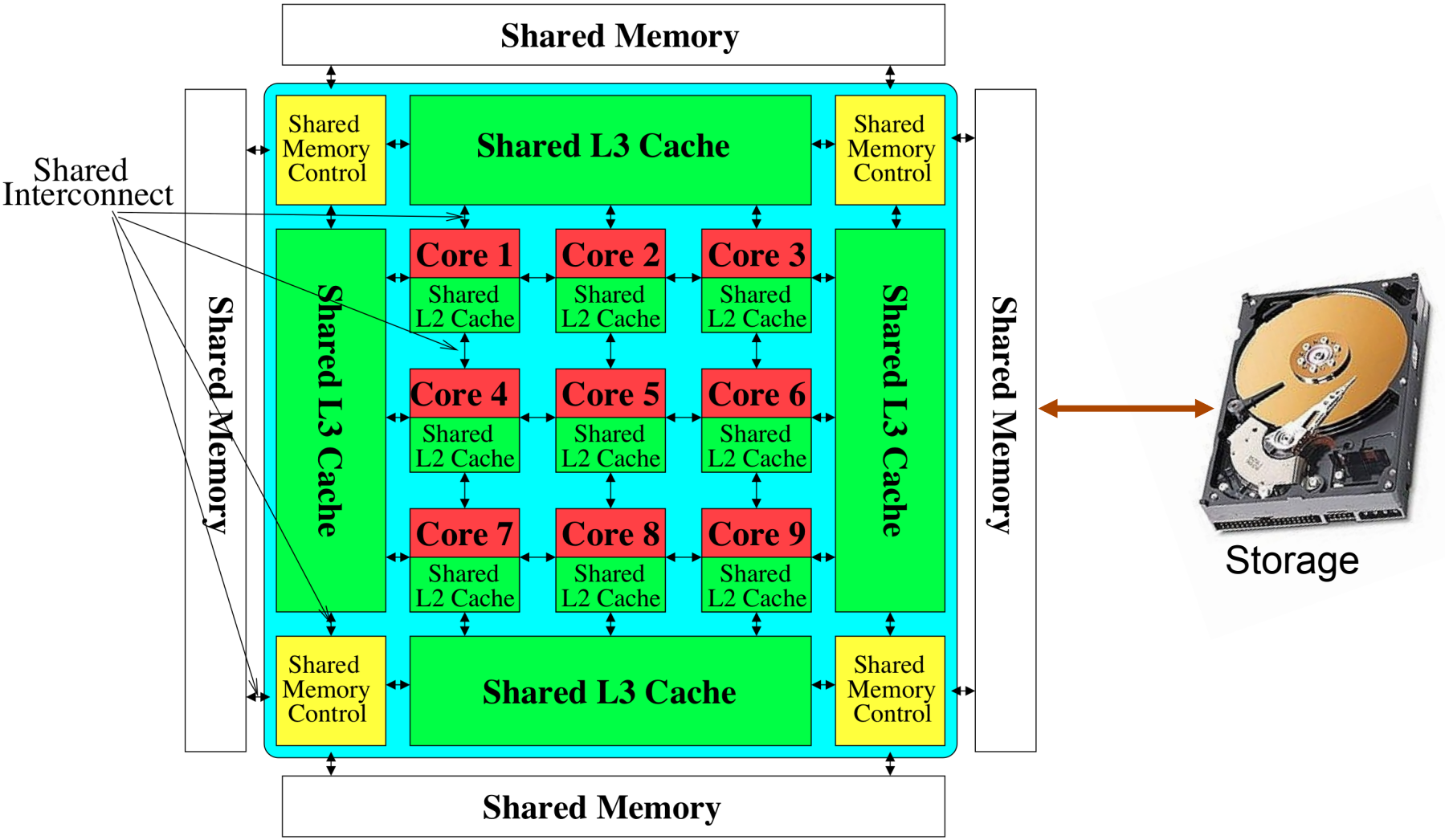
The Main Memory System



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

- Main memory system must scale (in *size*, *technology*, *efficiency*, *cost*, and *management algorithms*) to maintain performance growth and technology scaling benefits

Memory System: A *Shared Resource* View



State of the Main Memory System

- Recent technology, architecture, and application trends
  - lead to new requirements
  - exacerbate old requirements

- DRAM and memory controllers, as we know them today, are (will be) unlikely to satisfy all requirements

- Some emerging non-volatile memory technologies (e.g., PCM) enable new opportunities: memory+storage merging

- We need to rethink the main memory system
  - to fix DRAM issues and enable emerging technologies
  - to satisfy all requirements

Agenda

- Major Trends Affecting Main Memory
- DRAM Scaling Problem and Solution Directions
- Three New Techniques for DRAM
  - RAIDR: Reducing Refresh Impact
  - TL-DRAM: Reducing DRAM Latency
  - SALP: Reducing Bank Conflict Impact
- Ongoing Research
- Summary

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

Major Trends Affecting Main Memory (IV)

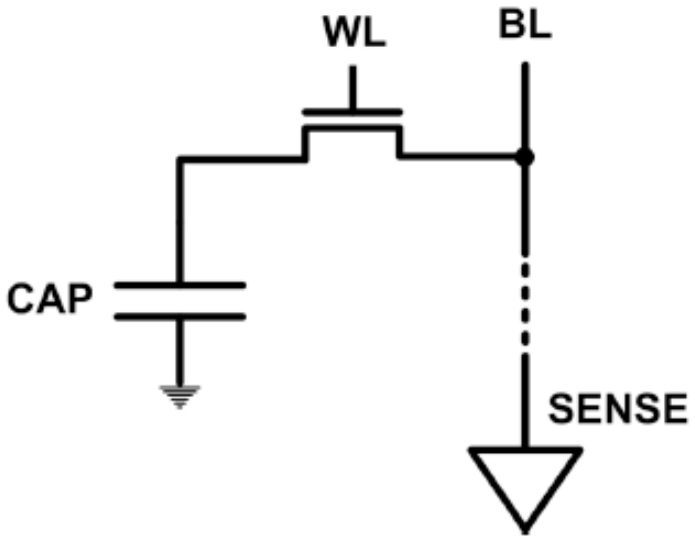
- Need for main memory capacity, bandwidth, QoS increasing

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

- DRAM technology scaling is ending
  - ITRS projects DRAM will not scale easily below X nm
  - Scaling has provided many benefits:
    - higher capacity (density), lower cost, lower energy

The DRAM Scaling Problem

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



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

Solutions to the DRAM Scaling Problem

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

Solution 1: Tolerate DRAM

- Overcome DRAM shortcomings with
  - System-DRAM co-design
  - Novel DRAM architectures, interface, functions
  - Better waste management (efficient utilization)
- Key issues to tackle
  - Reduce refresh energy
  - Improve bandwidth and latency
  - Reduce waste
  - Enable reliability at low cost

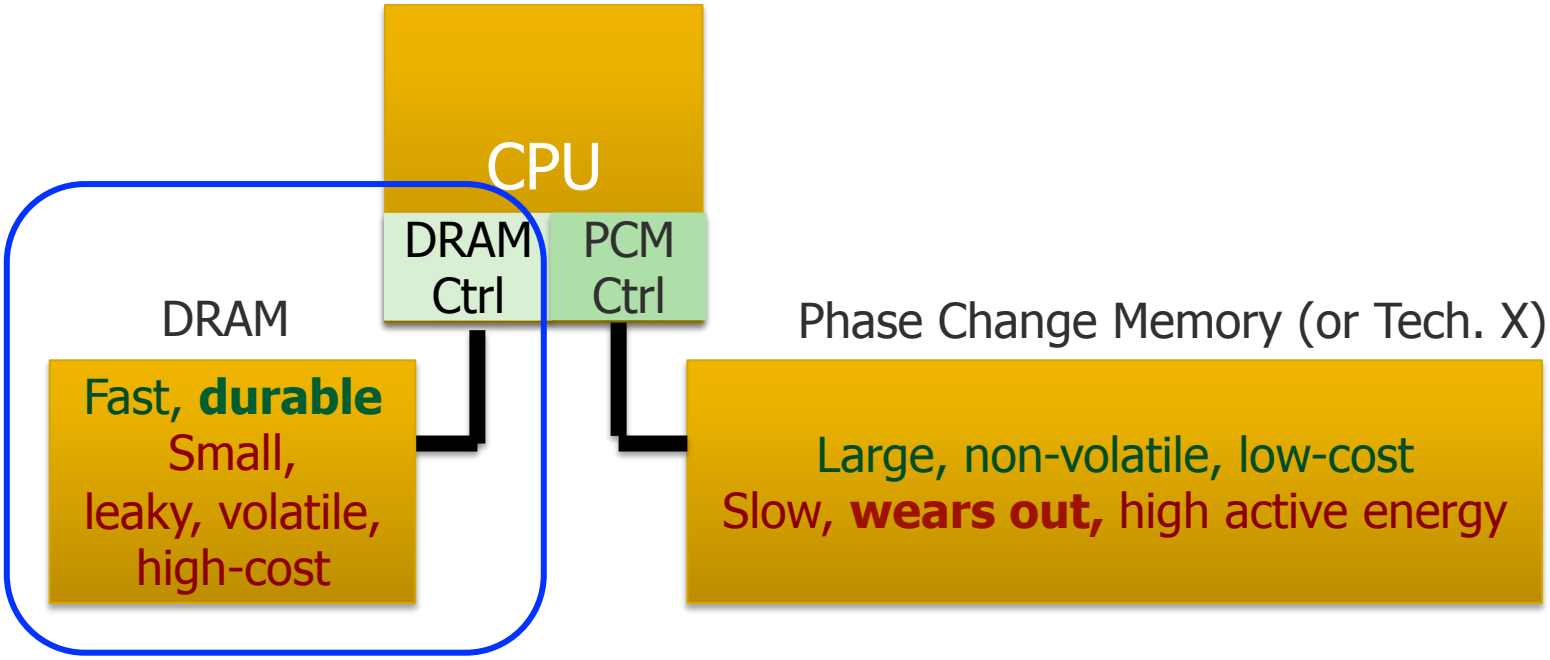
- Liu, Jaiyen, Veras, Mutlu, "RAIDR: Retention-Aware Intelligent DRAM Refresh," ISCA 2012.
- Kim, Seshadri, Lee+, "A Case for Exploiting Subarray-Level Parallelism in DRAM," ISCA 2012.
- Lee+, "Tiered-Latency DRAM: A Low Latency and Low Cost DRAM Architecture," HPCA 2013.
- Liu+, "An Experimental Study of Data Retention Behavior in Modern DRAM Devices" ISCA'13.
- Seshadri+, "RowClone: Fast and Efficient In-DRAM Copy and Initialization of Bulk Data," 2013.

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



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.

Agenda

- Major Trends Affecting Main Memory
- DRAM Scaling Problem and Solution Directions
- Three New Techniques for DRAM
  - RAIDR: Reducing Refresh Impact
  - TL-DRAM: Reducing DRAM Latency
  - SALP: Reducing Bank Conflict Impact
- Ongoing Research
- Summary

DRAM Refresh

- DRAM capacitor charge leaks over time

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

- Downsides of refresh

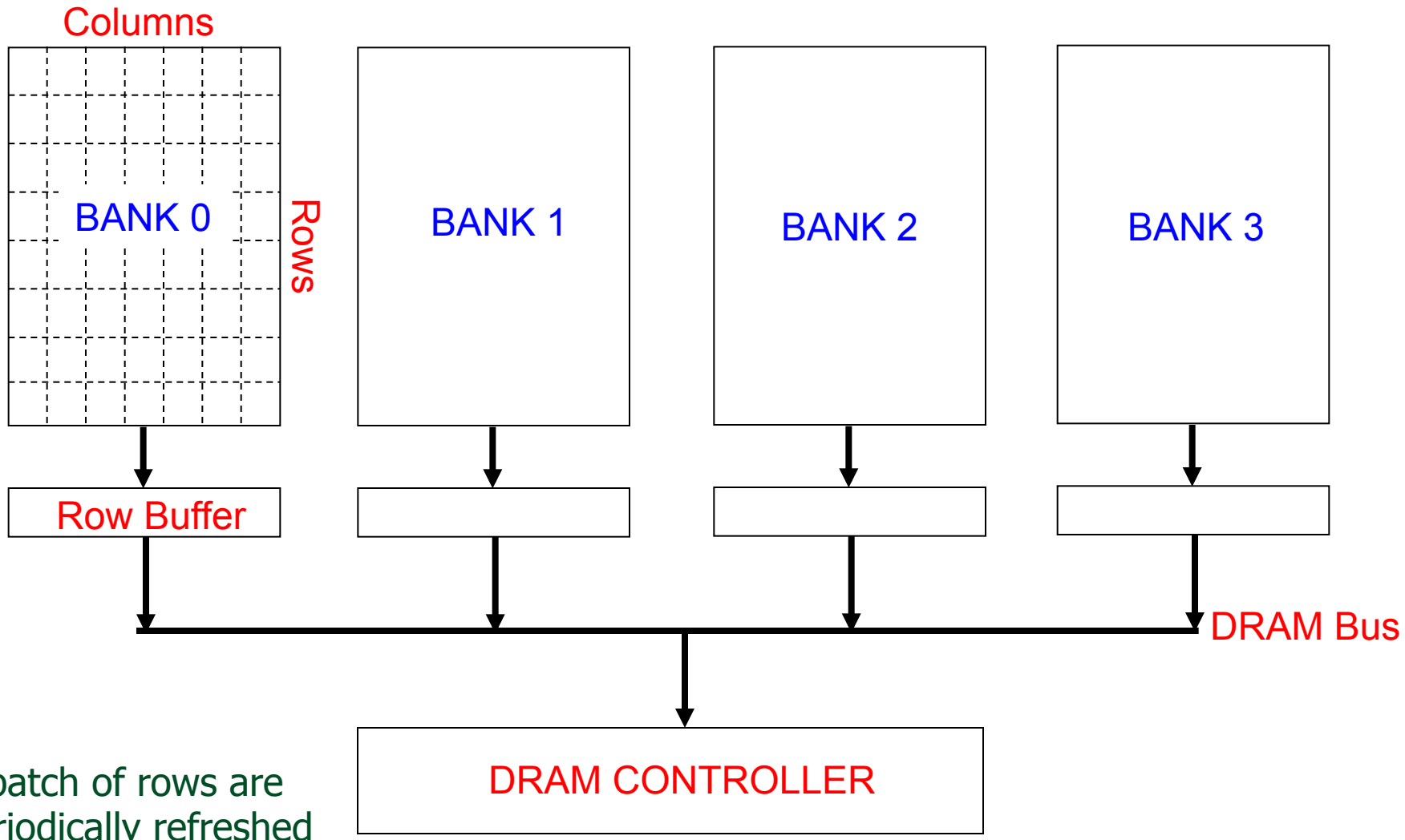
  -- Energy consumption: Each refresh consumes energy

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

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

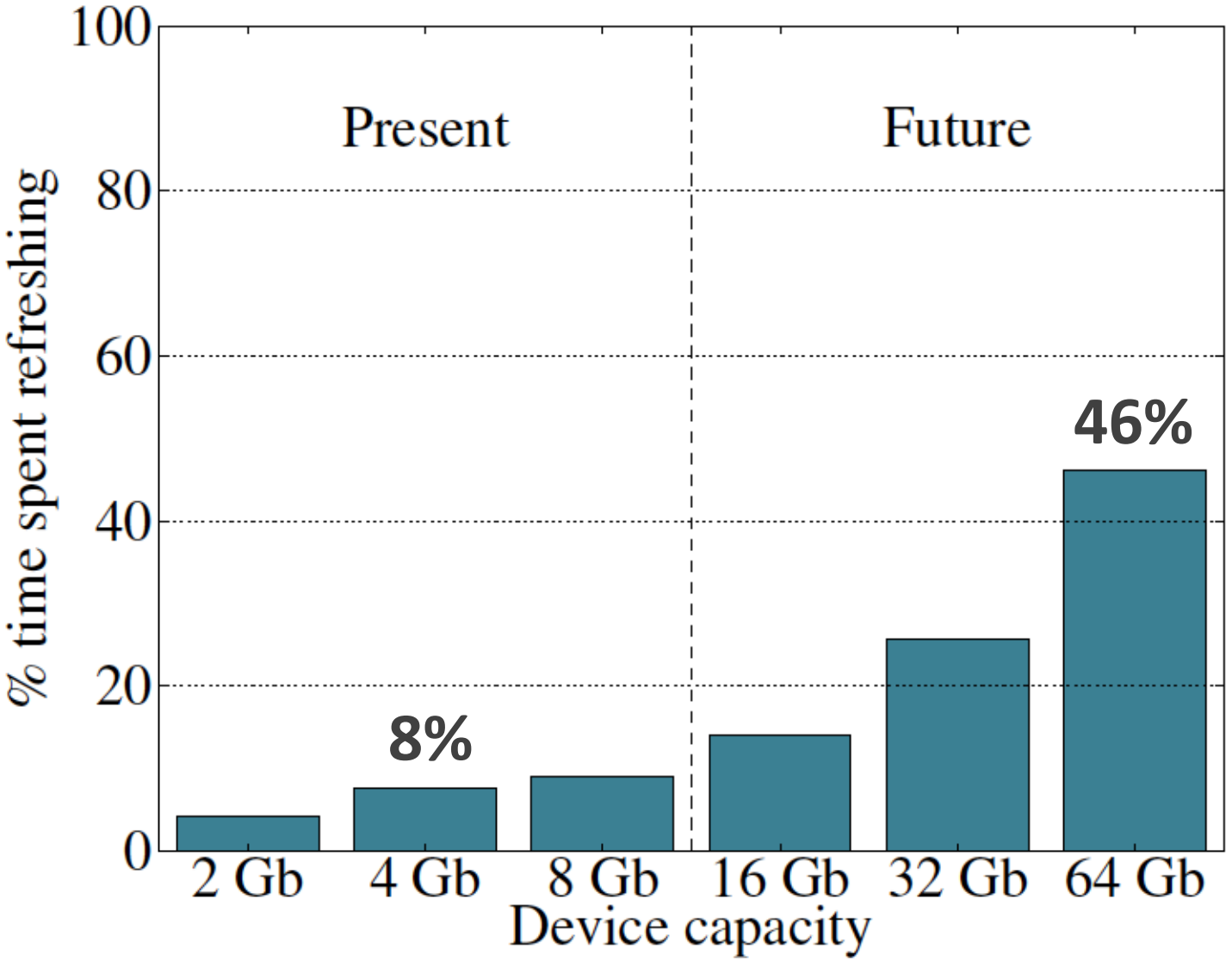
  -- Refresh rate limits DRAM density scaling

Refresh Today: Auto Refresh

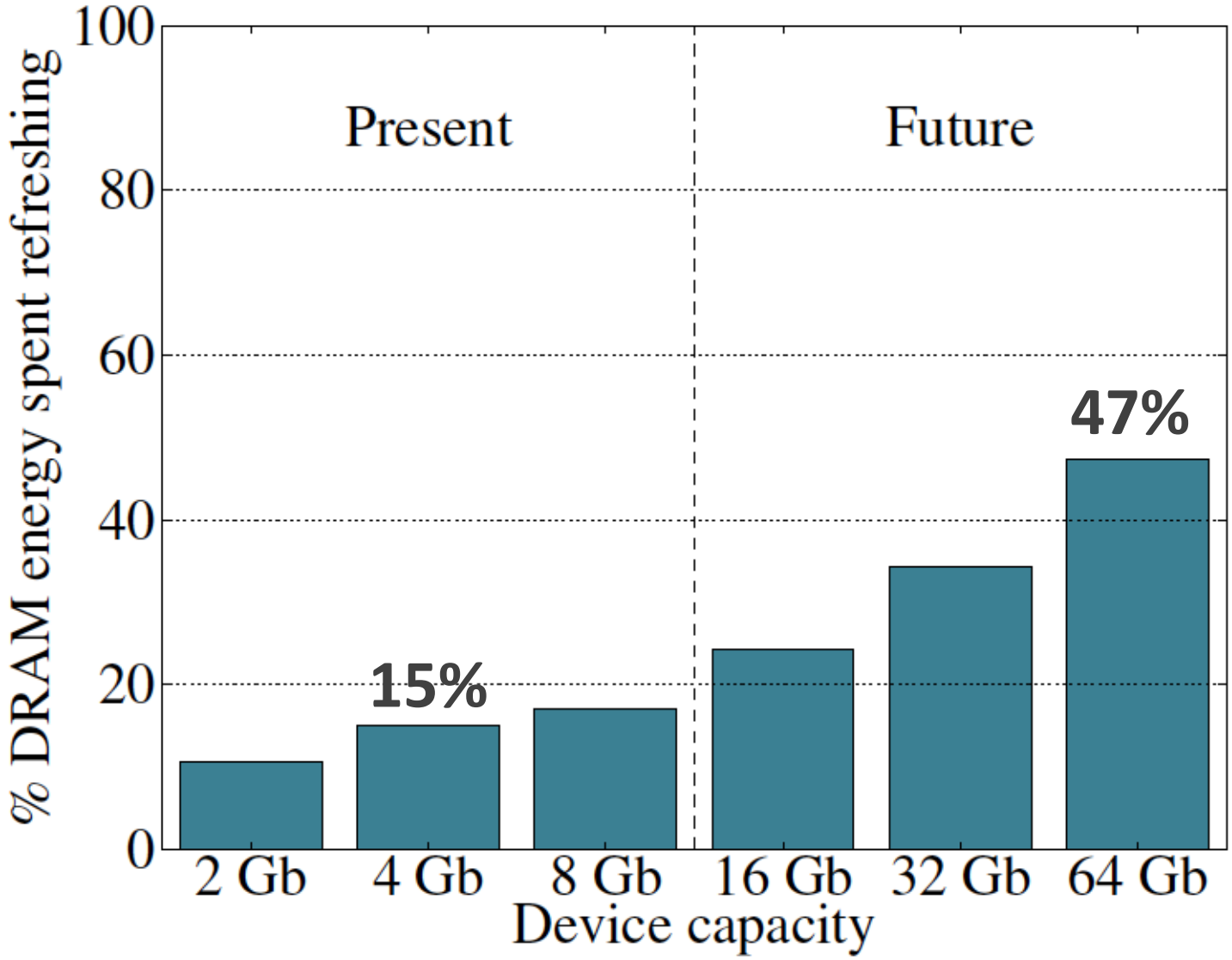


A batch of rows are periodically refreshed via the auto-refresh command

Refresh Overhead: Performance

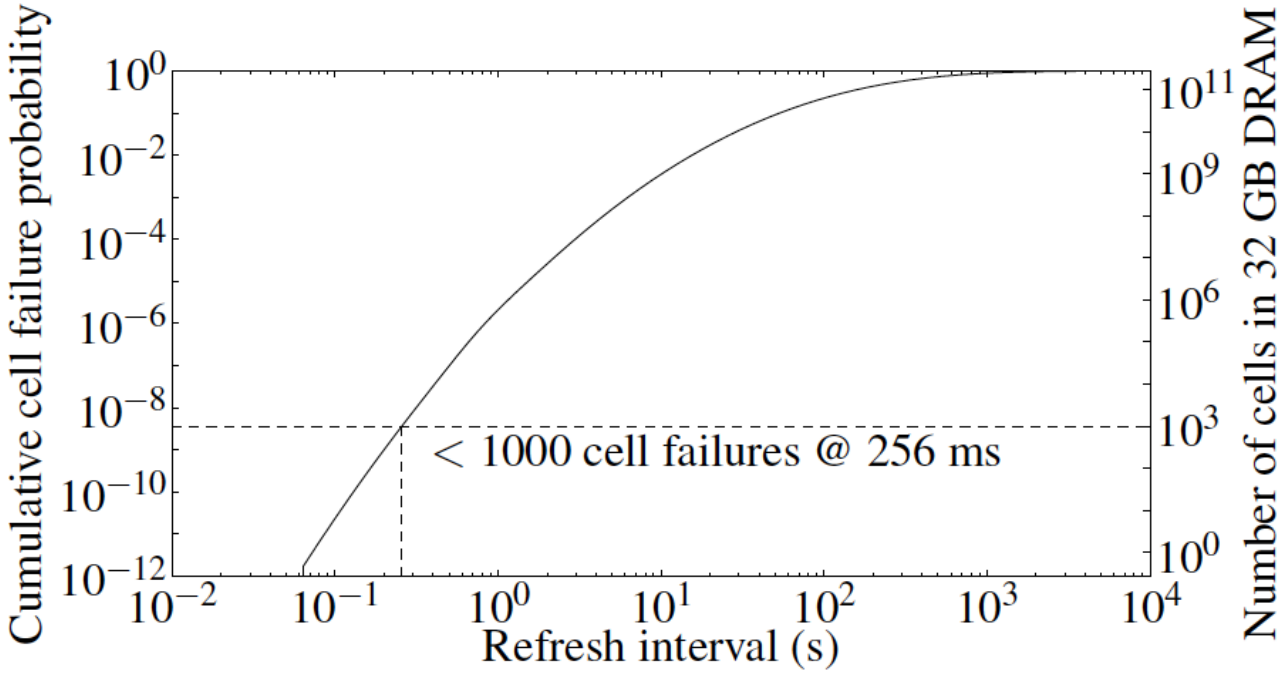


Refresh Overhead: Energy



Problem with Conventional Refresh

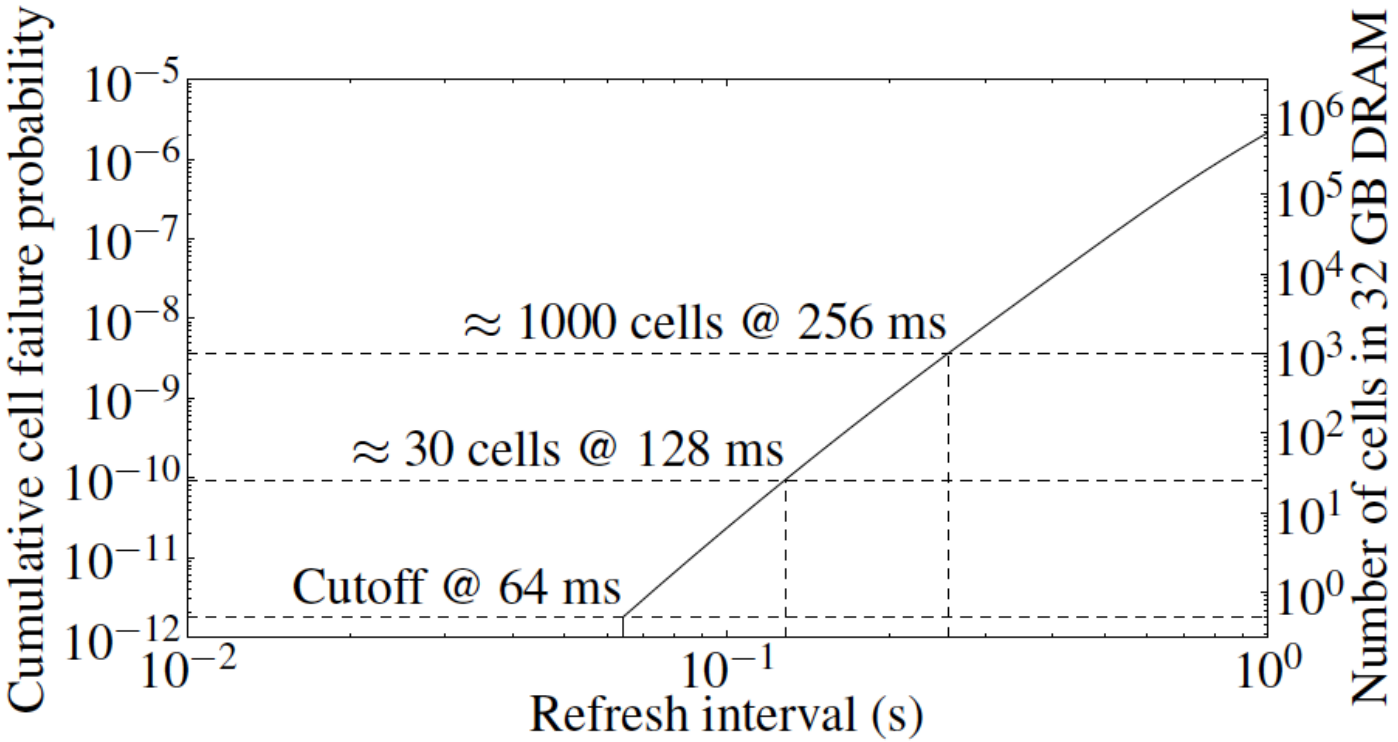
- Today: Every row is refreshed at the same rate



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

Retention Time of DRAM Rows

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

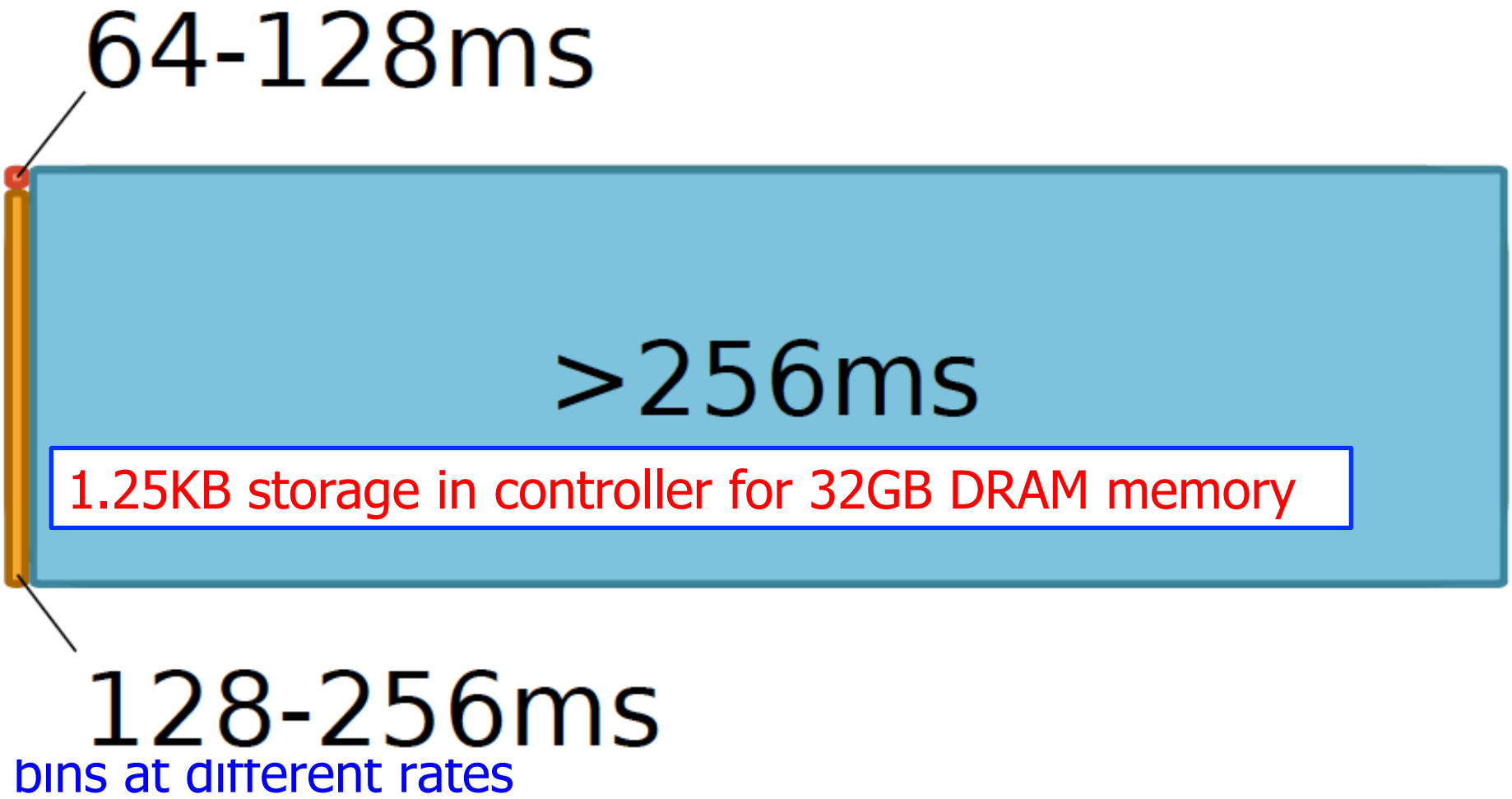


- Can we exploit this to reduce refresh operations at low cost?

Reducing DRAM Refresh Operations

- Idea: Identify the retention time of different rows and refresh each row at the frequency it needs to be refreshed
- (Cost-conscious) Idea: Bin the rows according to their minimum retention times and refresh rows in each bin at the refresh rate specified for the bin
  - e.g., a bin for 64-128ms, another for 128-256ms, …
- Observation: Only very few rows need to be refreshed very frequently [64-128ms] → Have only a few bins → Low HW overhead to achieve large reductions in refresh operations
- Liu et al., "RAIDR: Retention-Aware Intelligent DRAM Refresh," ISCA 2012.

RAIDR: Mechanism



→ probe Bloom Filters to determine refresh rate of a row

1. Profiling

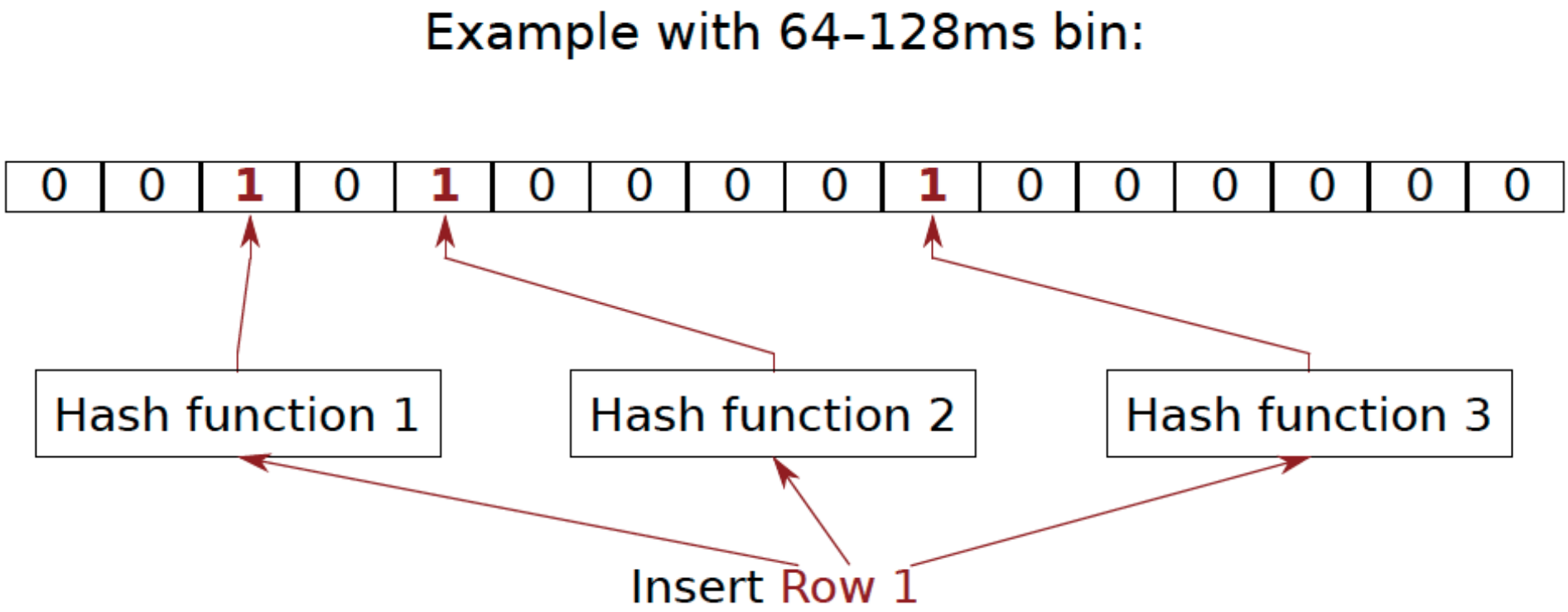
To profile a row:

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

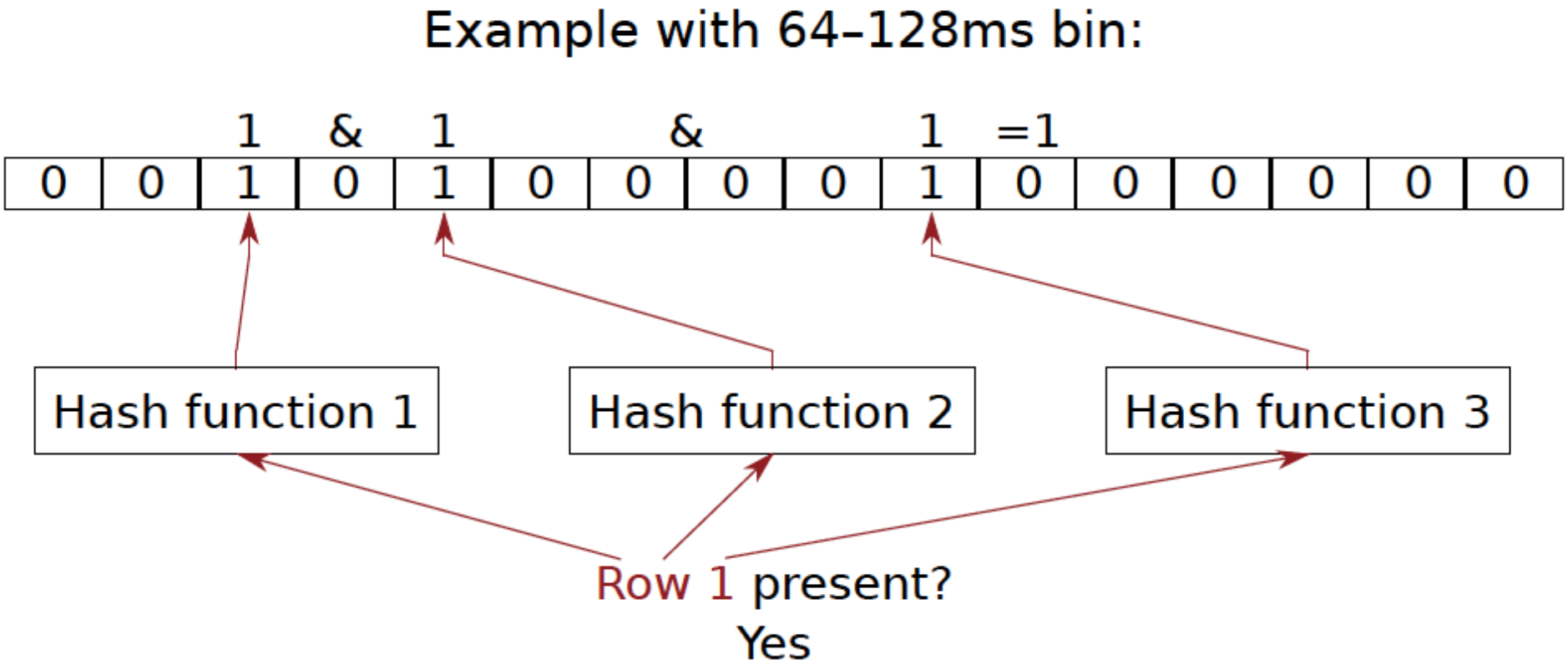
	Row 1	Row 2	Row 3
Initially	11111111...	11111111...	11111111...
After 64 ms	11111111...	11111111...	11111111...
After 128 ms	11011111... (64–128ms)	11111111...	11111111...
After 256 ms		11111011... (128–256ms)	11111111... (>256ms)

2. Binning

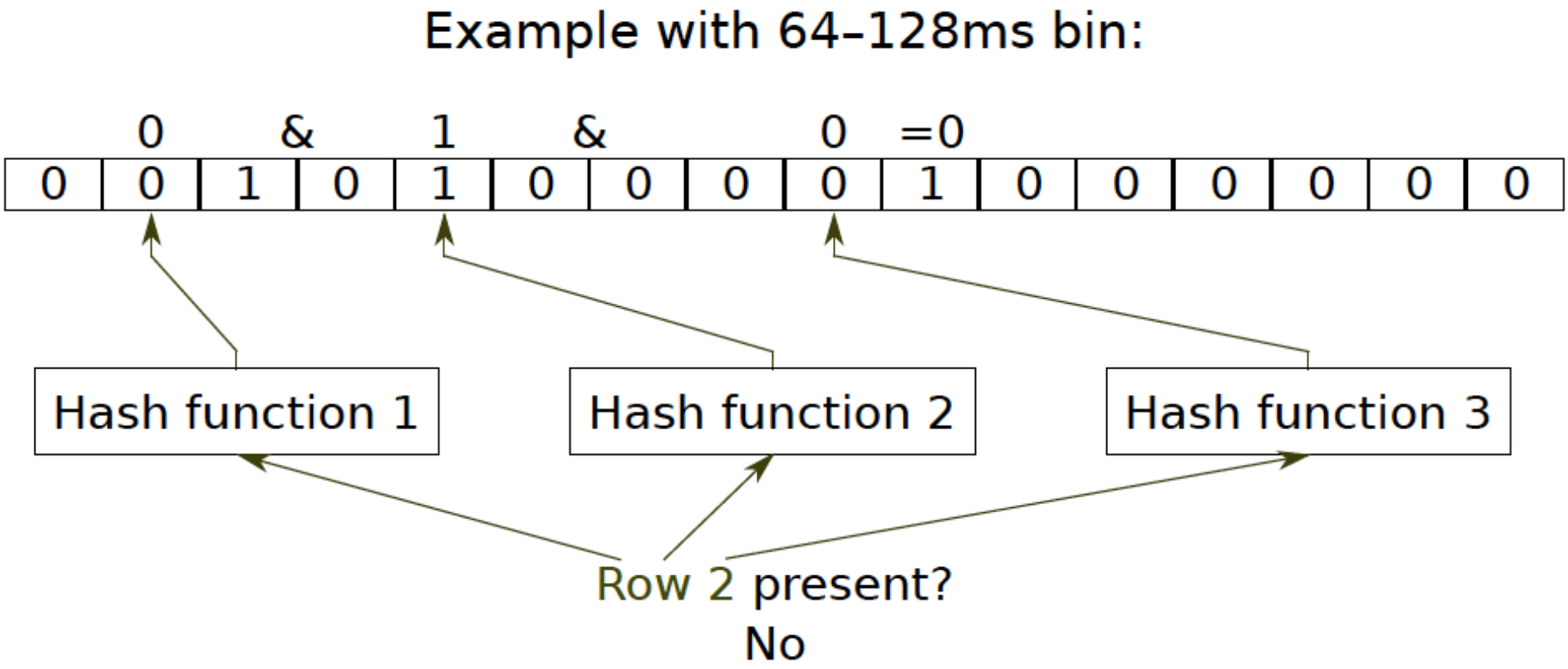
- How to efficiently and scalably store rows into retention time bins?
- Use Hardware Bloom Filters [Bloom, CACM 1970]



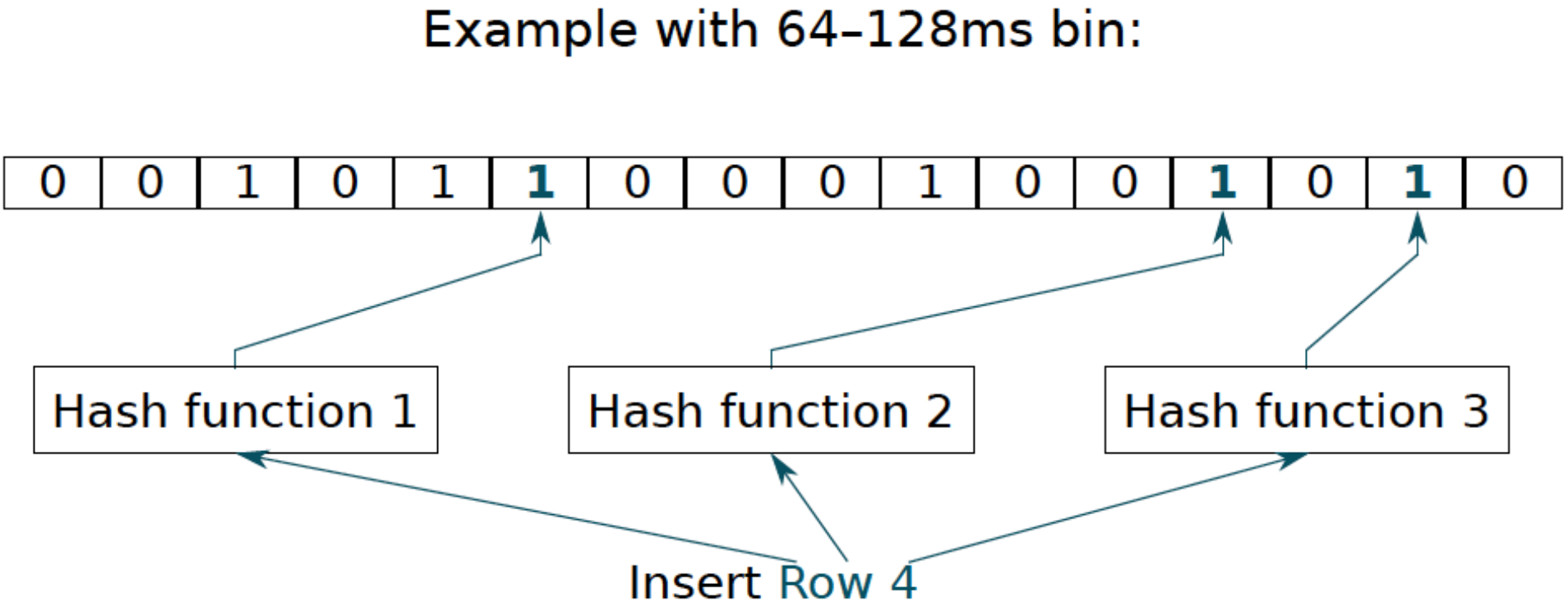
Bloom Filter Operation Example



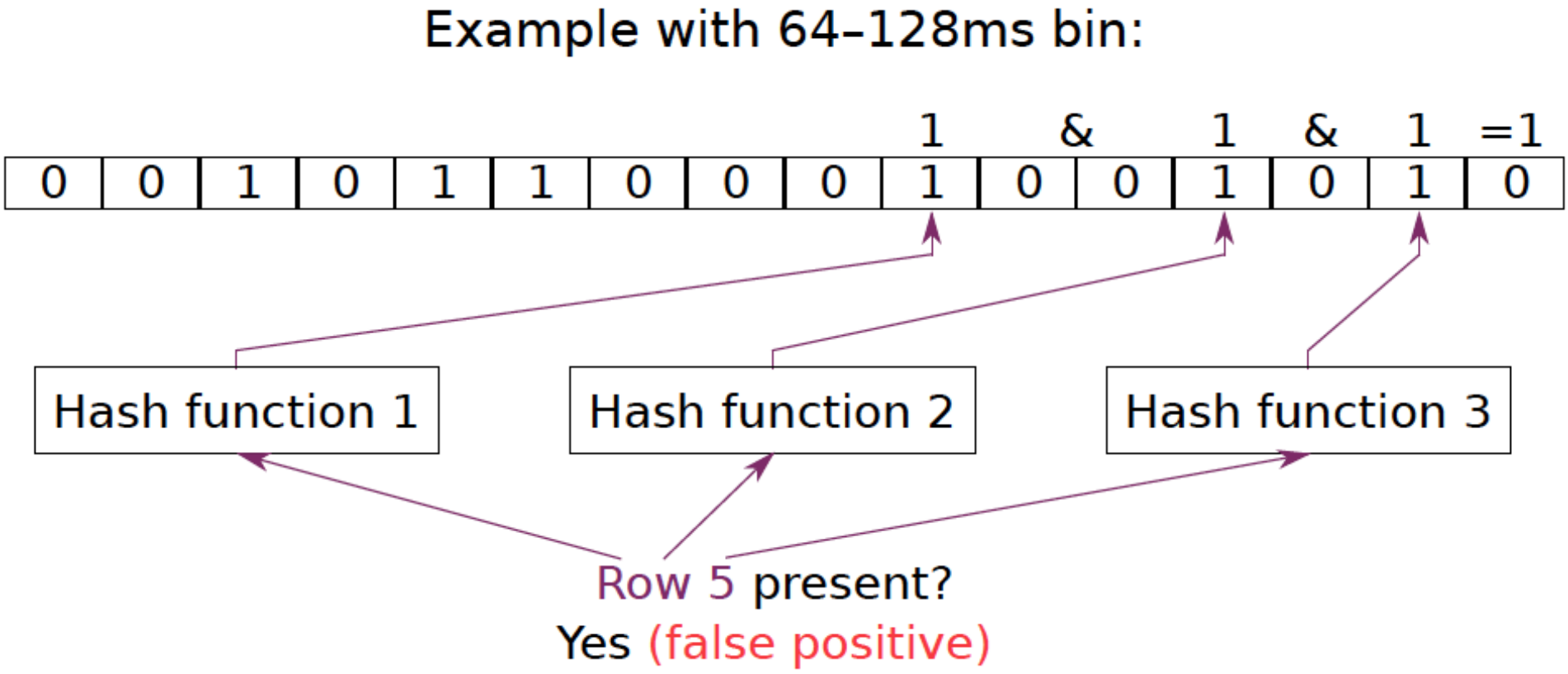
Bloom Filter Operation Example



Bloom Filter Operation Example



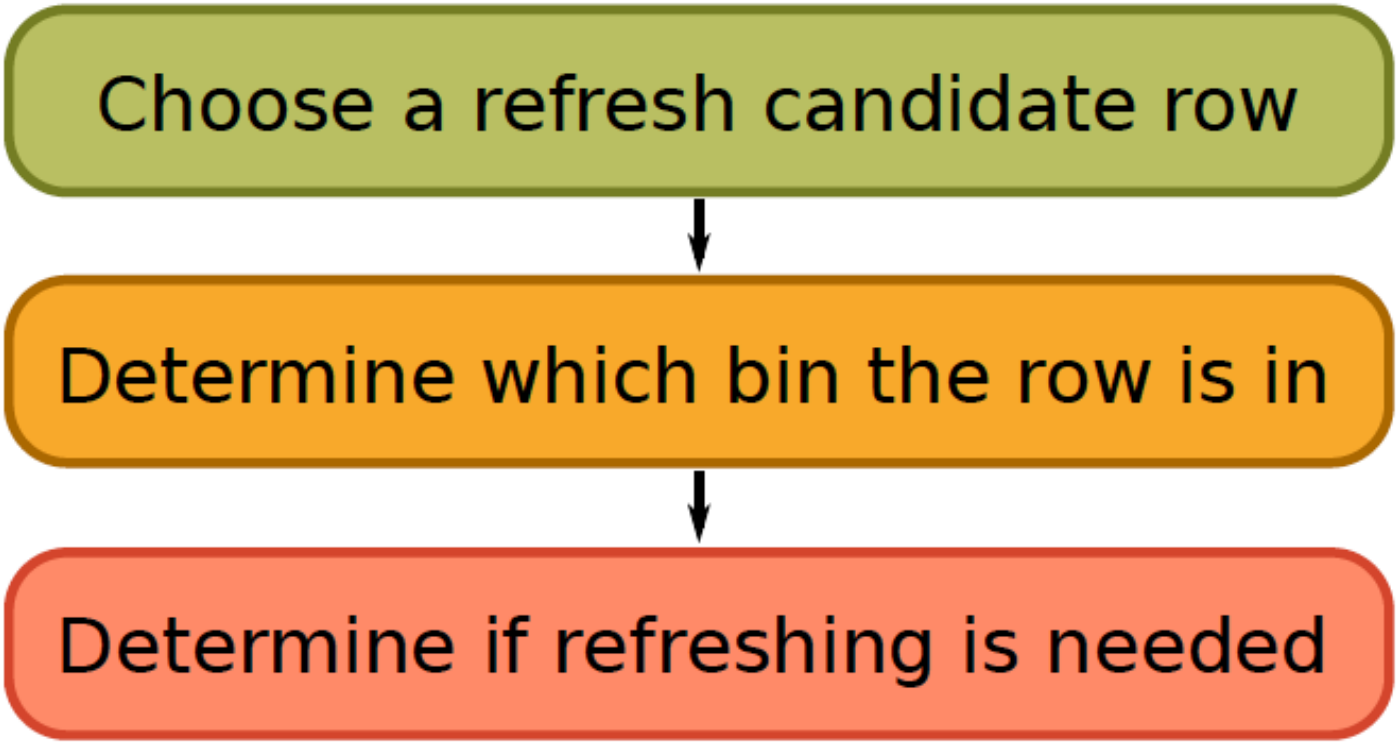
Bloom Filter Operation Example



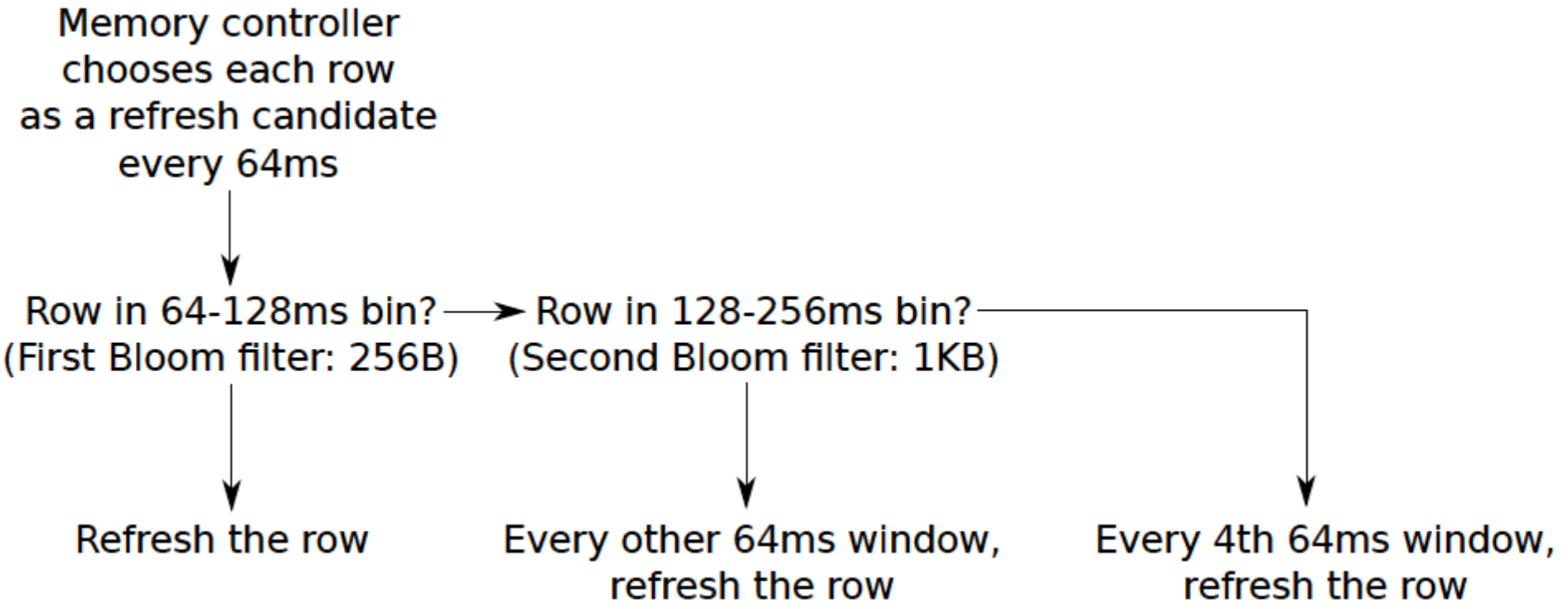
Benefits of Bloom Filters as Bins

- 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)



3. Refreshing (RAIDR Refresh Controller)

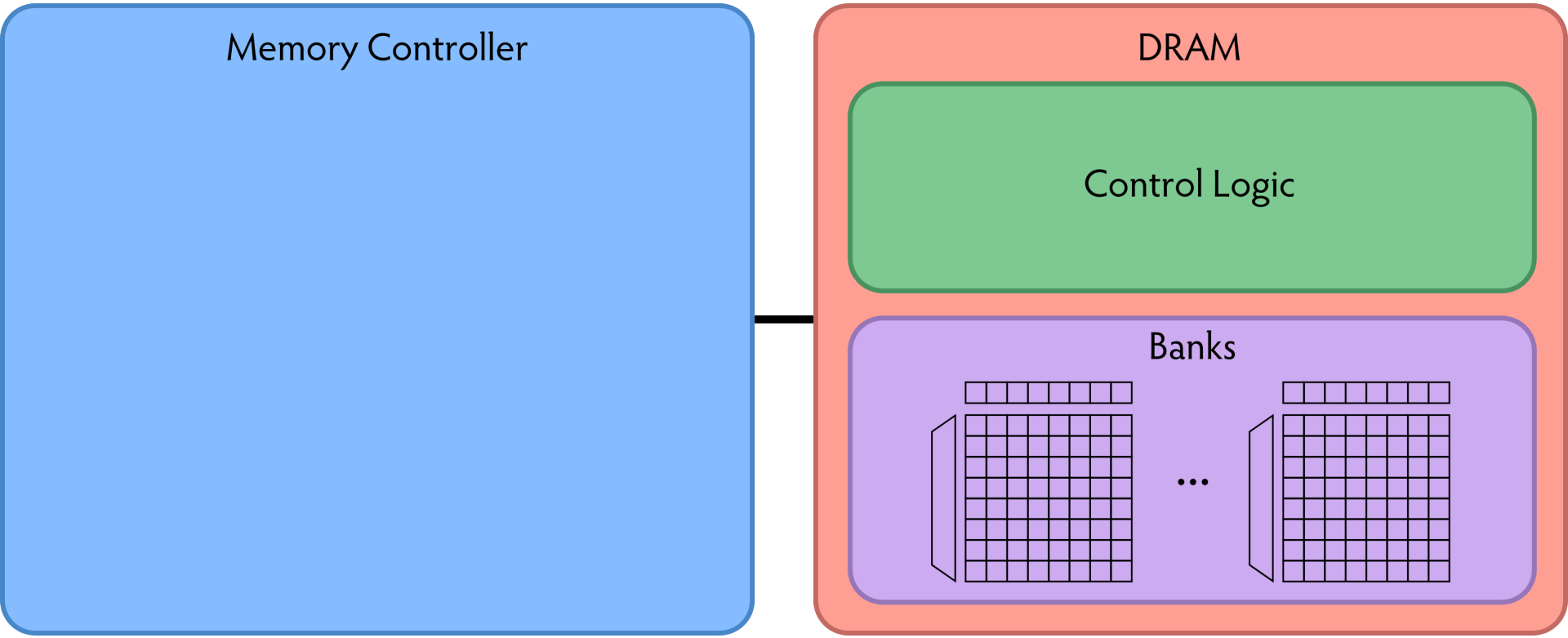


Liu et al., "RAIDR: Retention-Aware Intelligent DRAM Refresh," ISCA 2012.

Tolerating Temperature Changes

- Change in temperature causes retention time of all cells to change by a uniform and predictable factor
- Refresh rate scaling: increase the refresh rate for all rows uniformly, depending on the temperature
- Implementation: counter with programmable period
  - Lower temperature ⇒ longer period ⇒ less frequent refreshes
  - Higher temperature ⇒ shorter period ⇒ more frequent refreshes

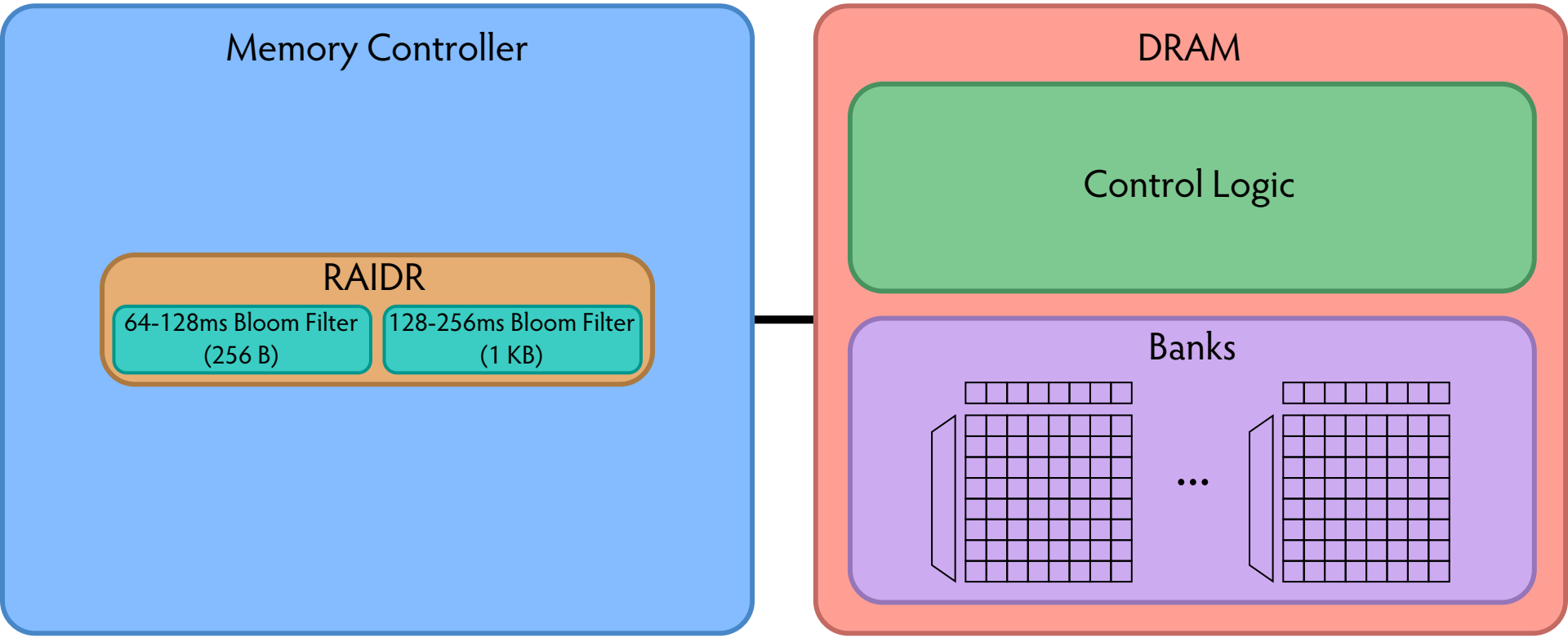
RAIDR: Baseline Design



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

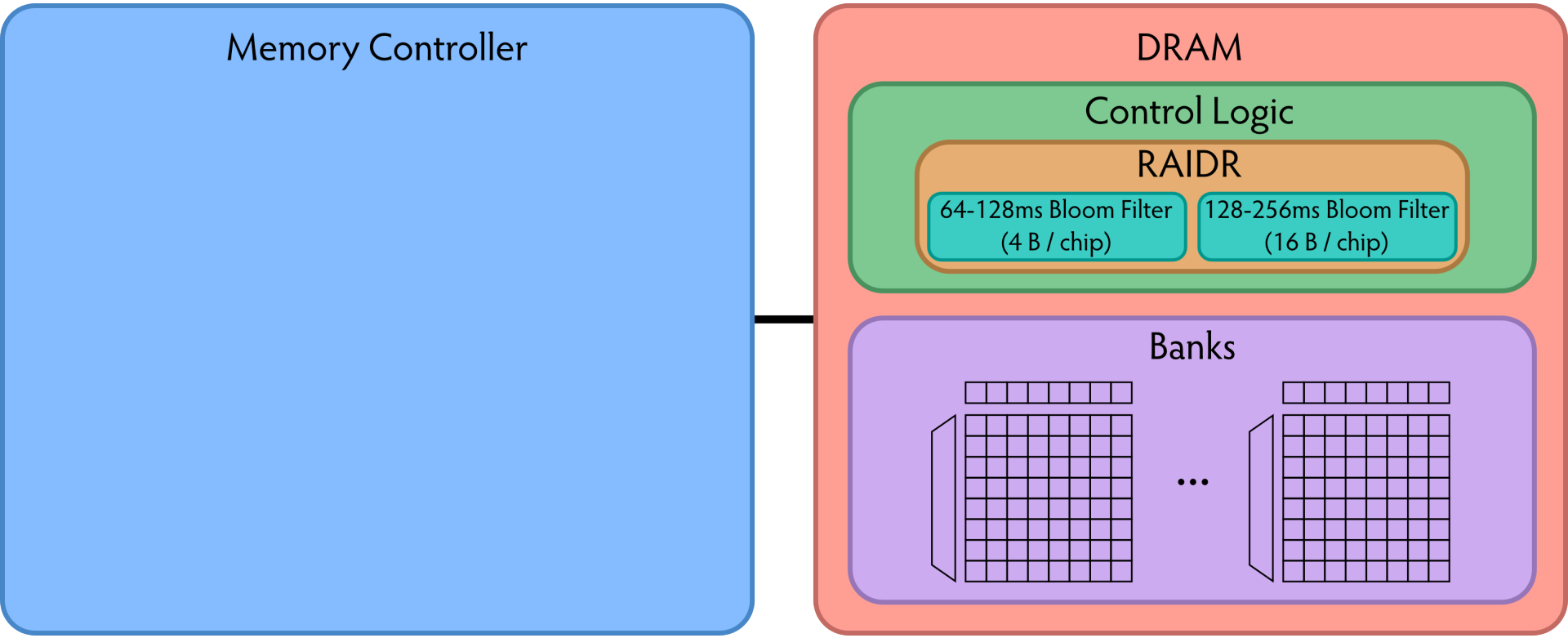
RAIDR can be implemented in either the controller or DRAM

RAIDR in Memory Controller: Option 1



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

RAIDR in DRAM Chip: Option 2



Overhead of RAIDR in DRAM chip:

Per-chip overhead: 20B Bloom Filters, 1 counter (4 Gbit chip)

Total overhead: 1.25KB Bloom Filters, 64 counters (32 GB DRAM)

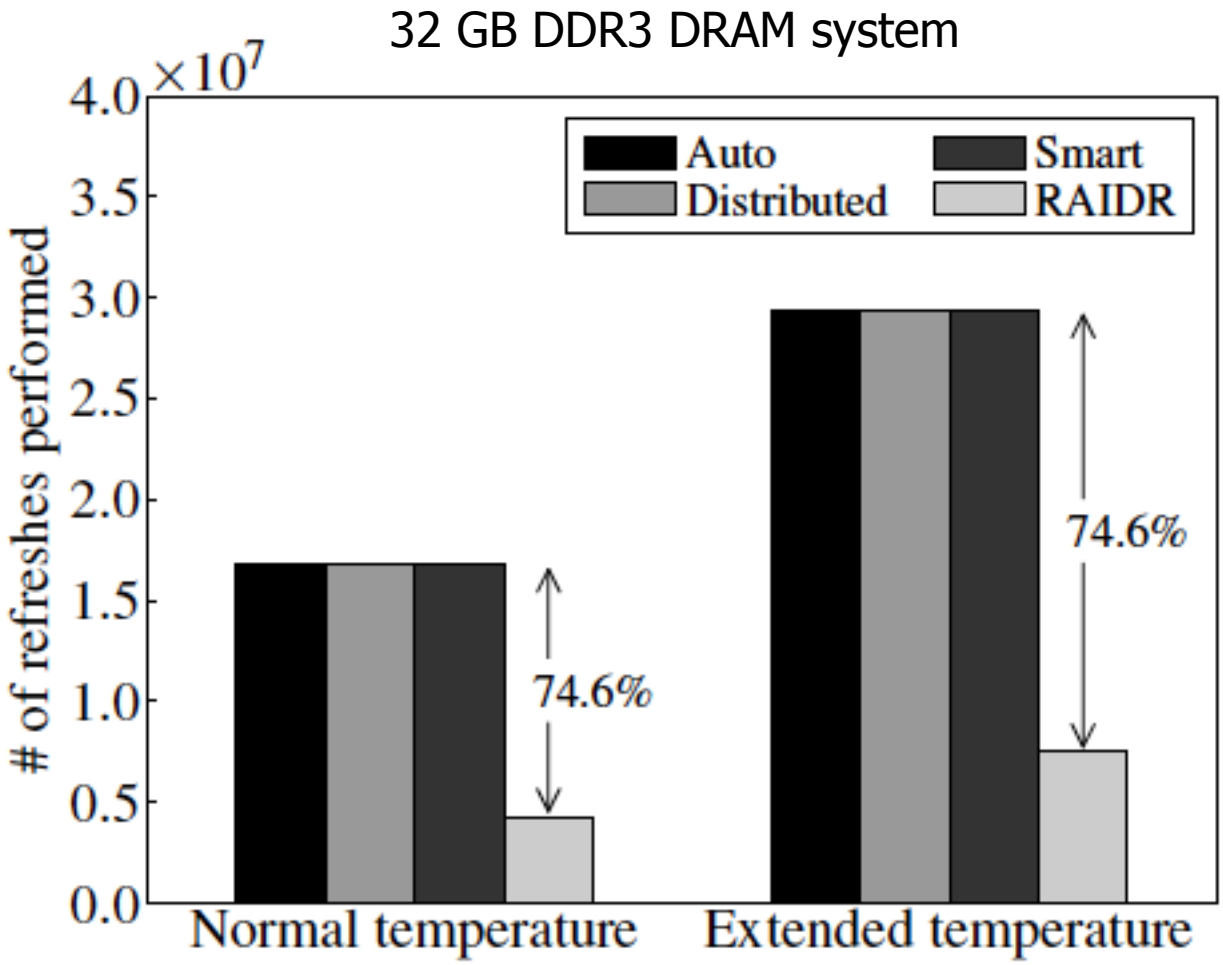
RAIDR Results

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

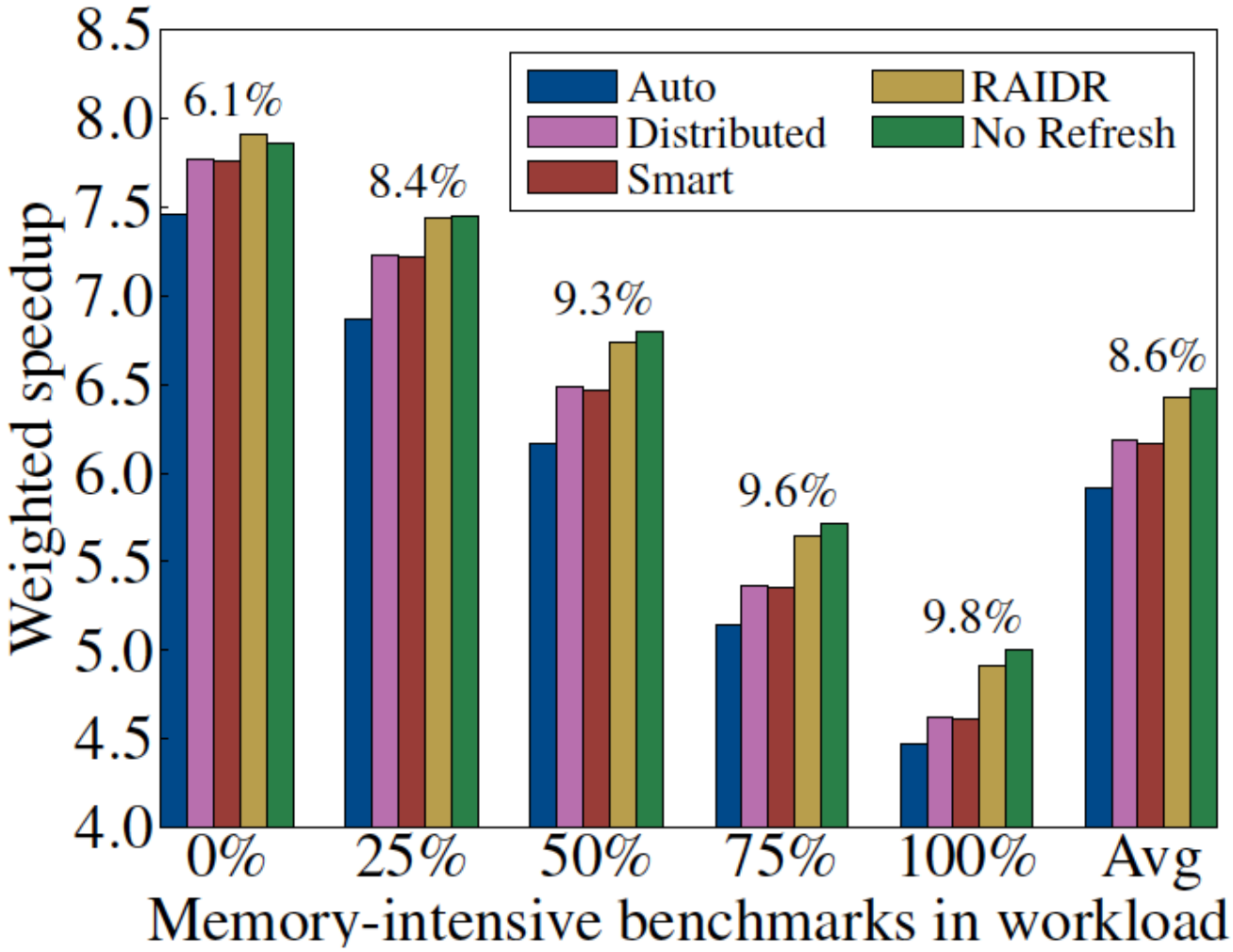
- RAIDR:
  - 64–128ms retention range: 256 B Bloom filter, 10 hash functions
  - 128–256ms retention range: 1 KB Bloom filter, 6 hash functions
  - Default refresh interval: 256 ms

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

RAIDR Refresh Reduction

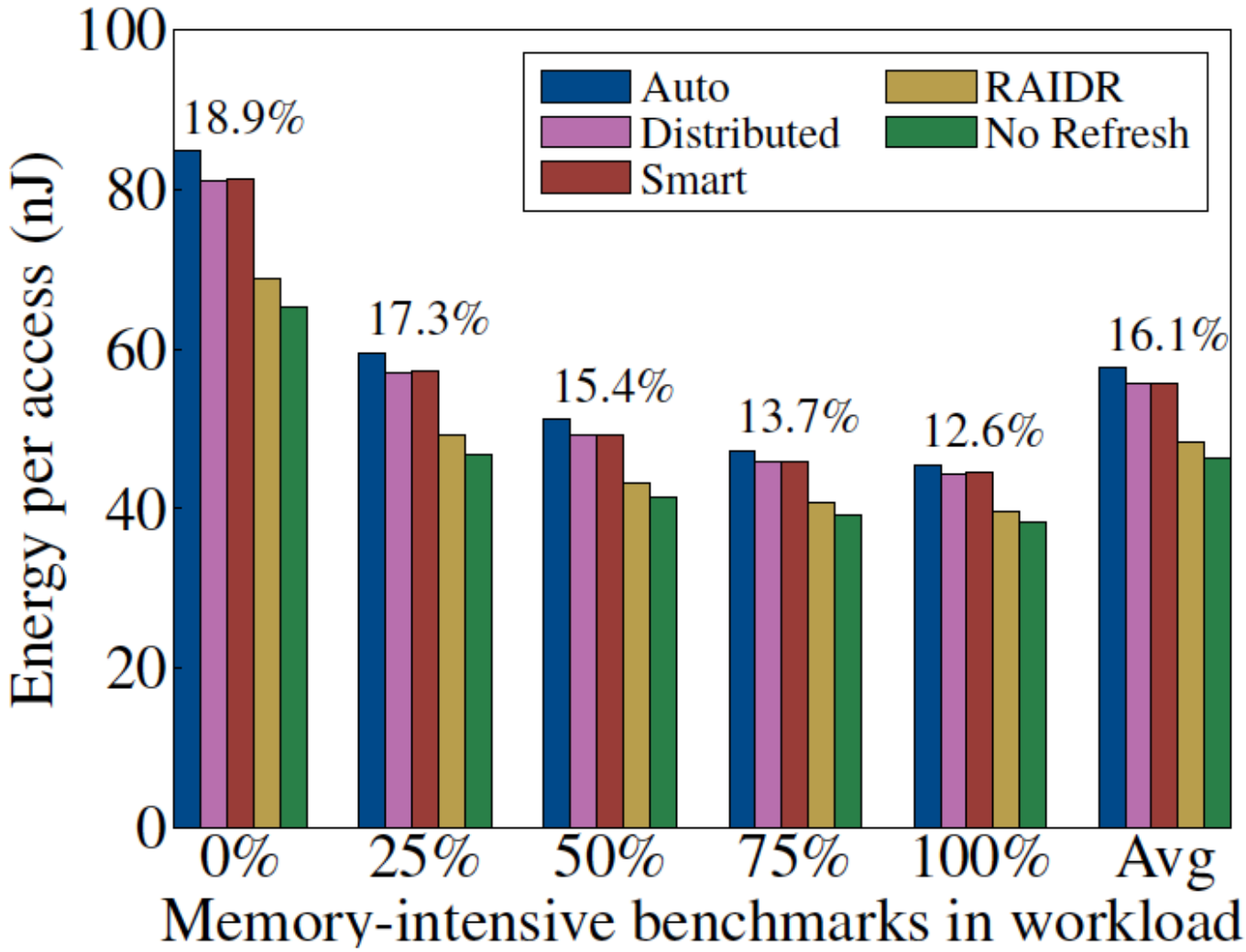


RAIDR: Performance



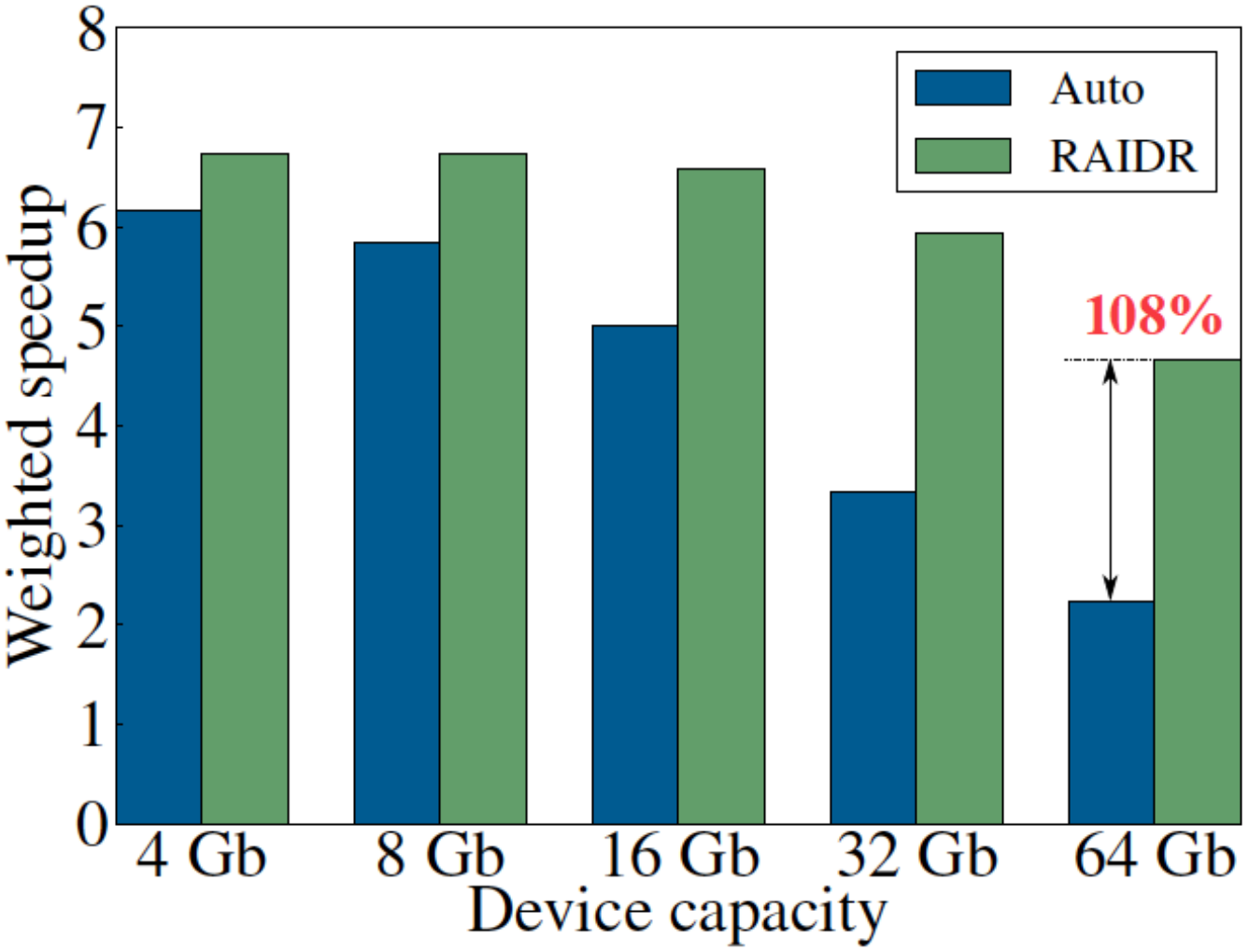
RAIDR performance benefits increase with workload's memory intensity

RAIDR: DRAM Energy Efficiency



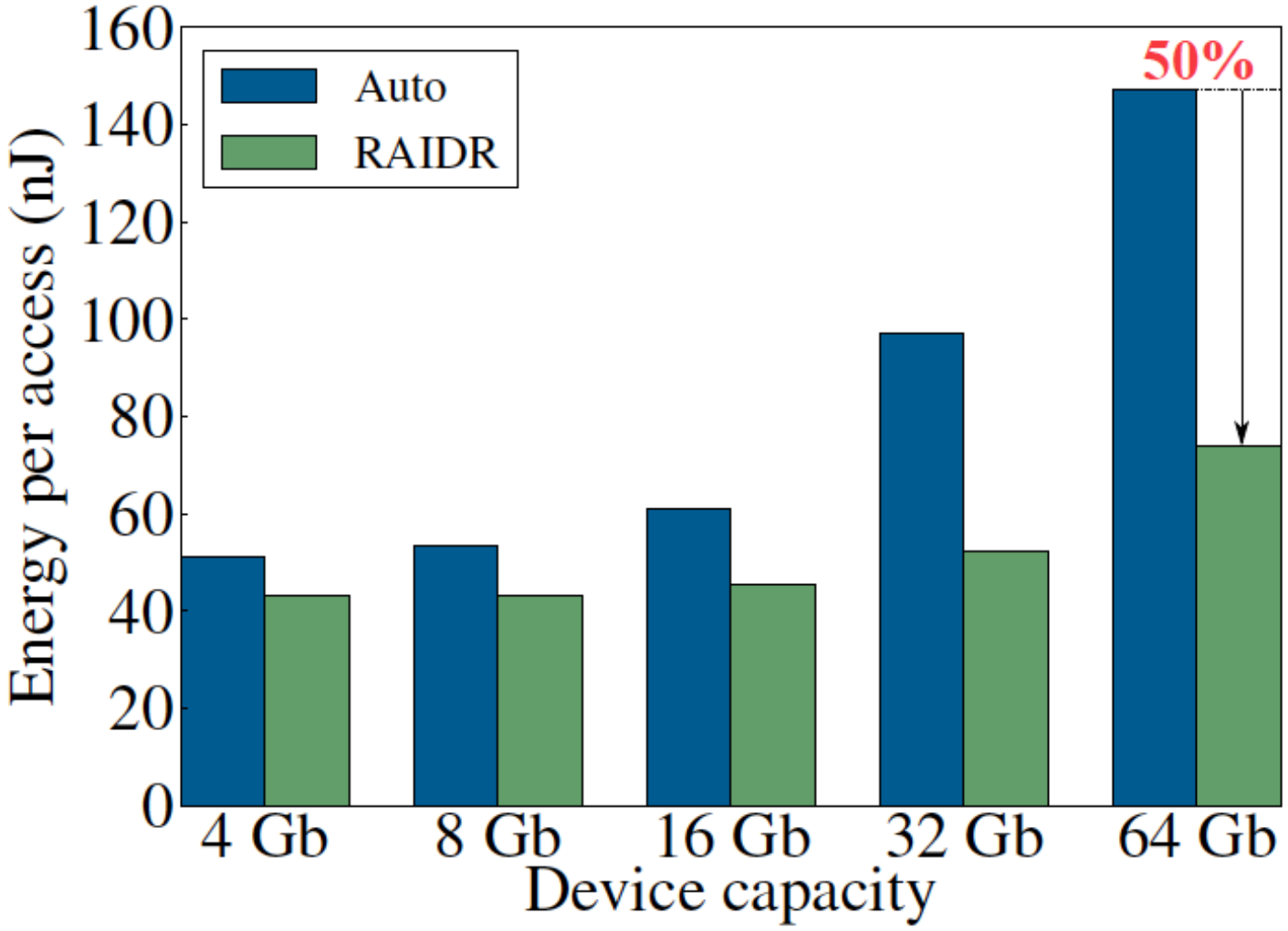
RAIDR energy benefits increase with memory idleness

DRAM Device Capacity Scaling: Performance



RAIDR performance benefits increase with DRAM chip capacity

DRAM Device Capacity Scaling: Energy



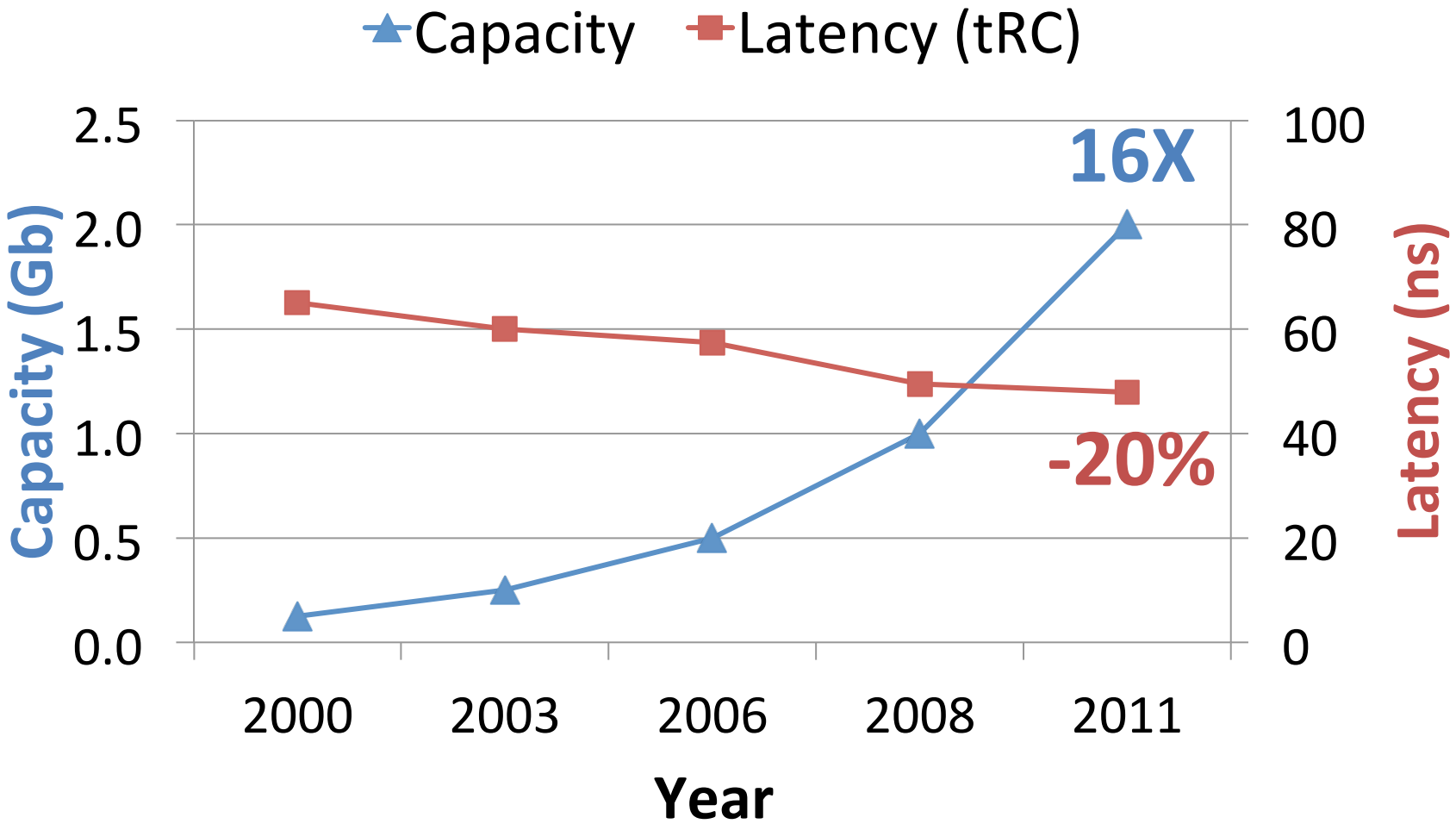
RAIDR energy benefits increase with DRAM chip capacity

RAIDR slides

Agenda

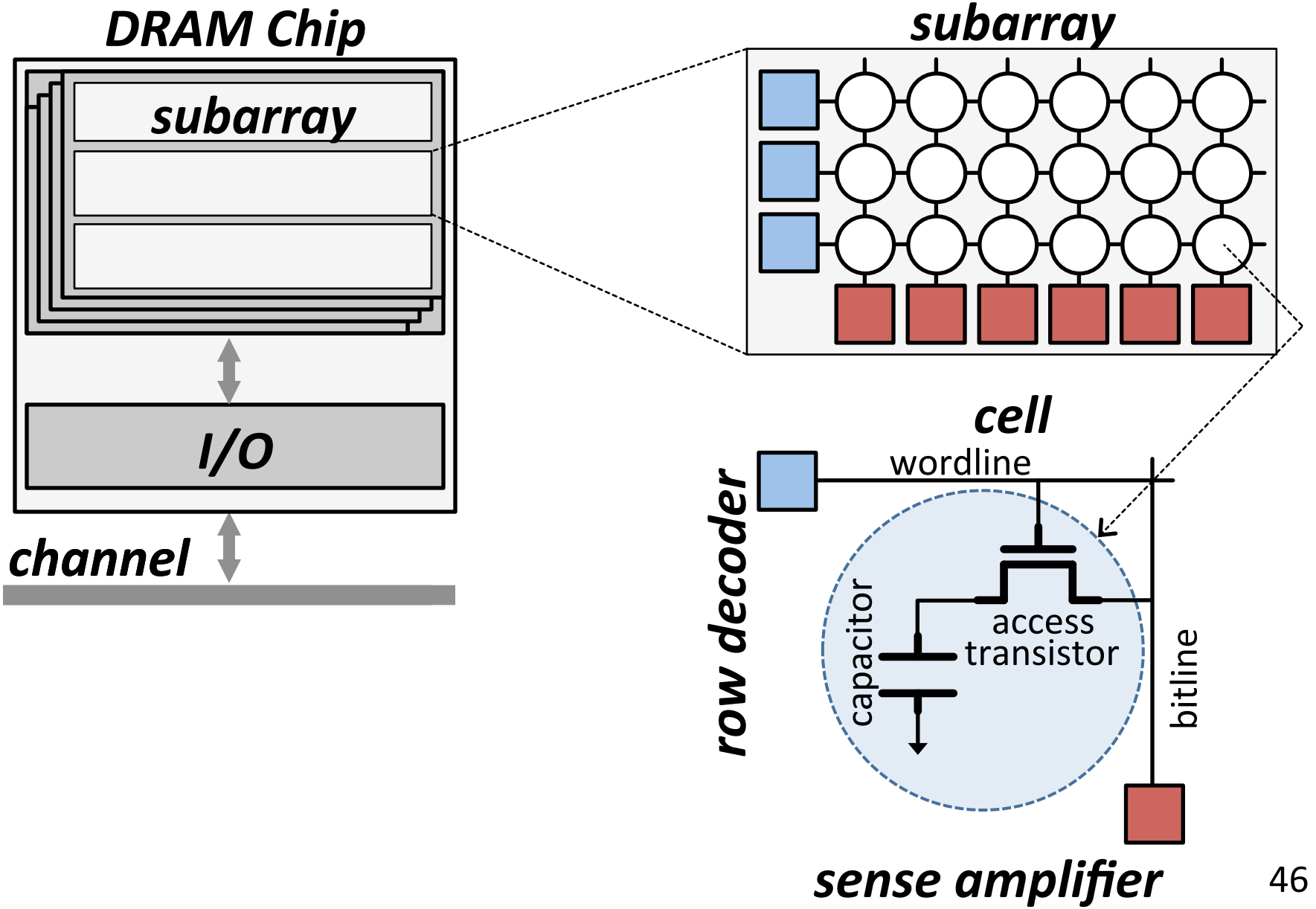
- Major Trends Affecting Main Memory
- DRAM Scaling Problem and Solution Directions
- Three New Techniques for DRAM
  - RAIDR: Reducing Refresh Impact
  - TL-DRAM: Reducing DRAM Latency
  - SALP: Reducing Bank Conflict Impact
- Ongoing Research
- Summary

Historical DRAM Latency-Capacity Trend

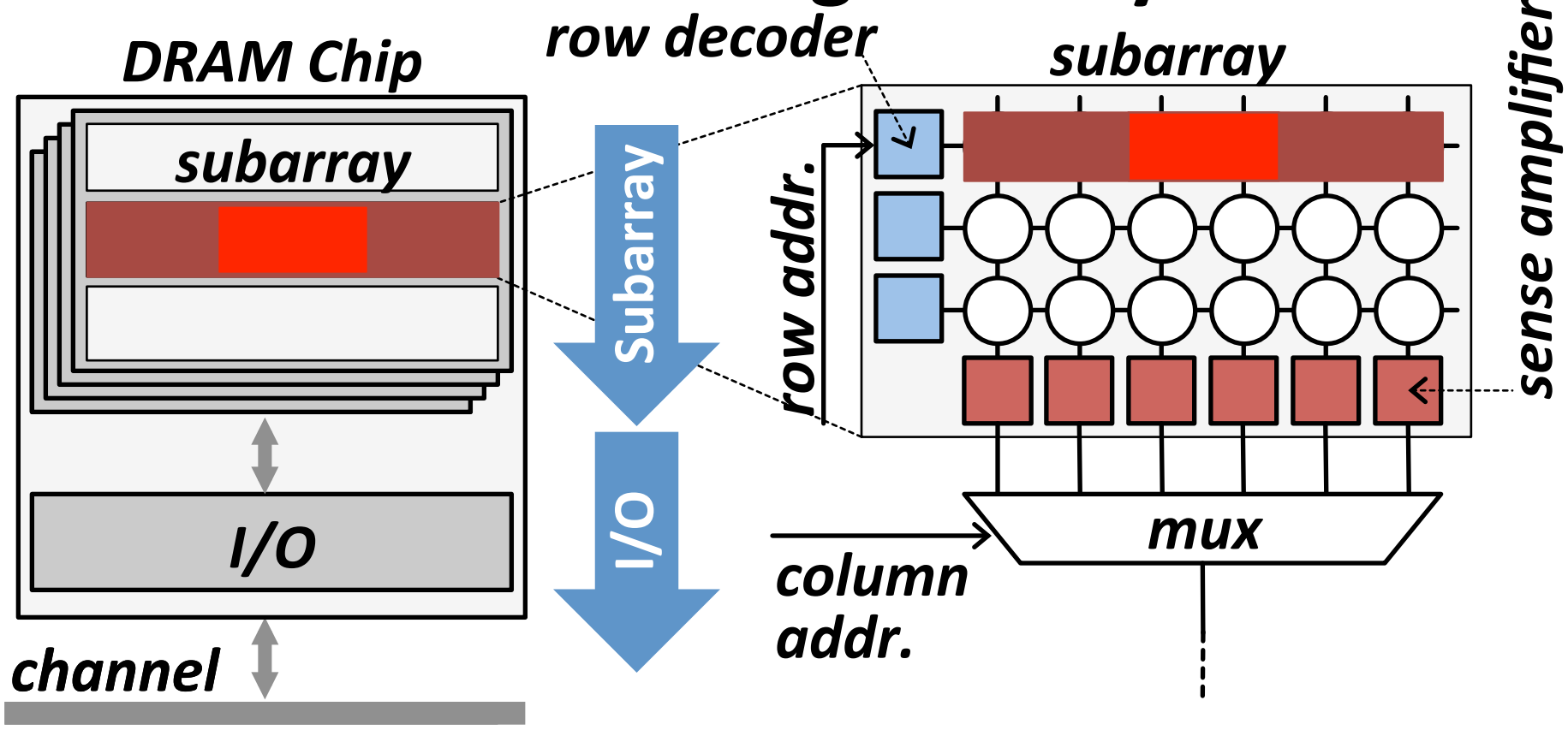


DRAM latency continues to be a critical bottleneck

What Causes the Long Latency?



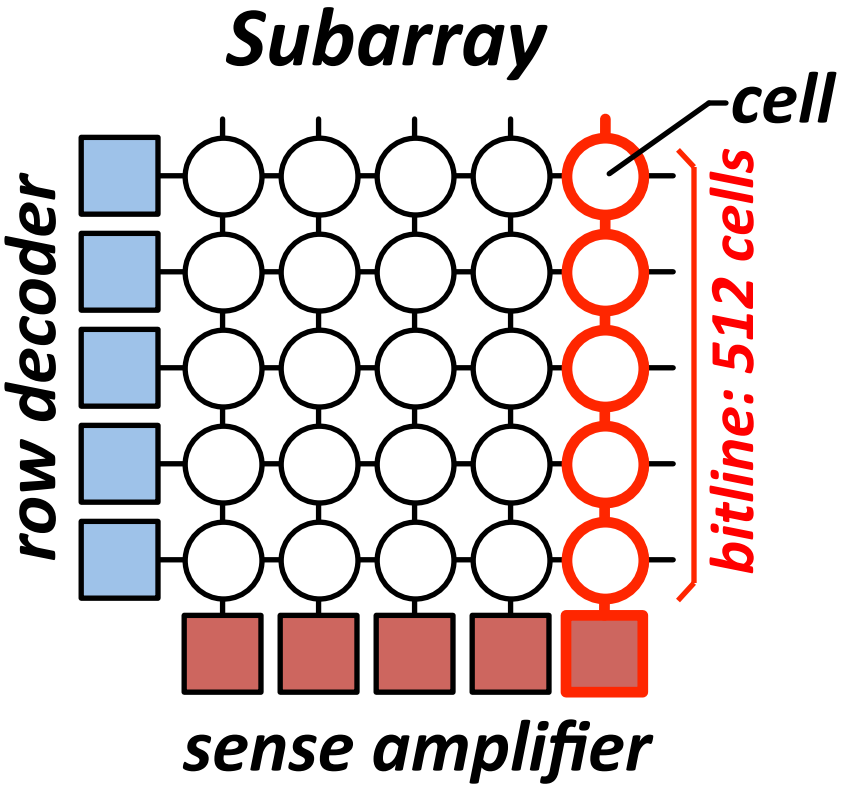
What Causes the Long Latency?

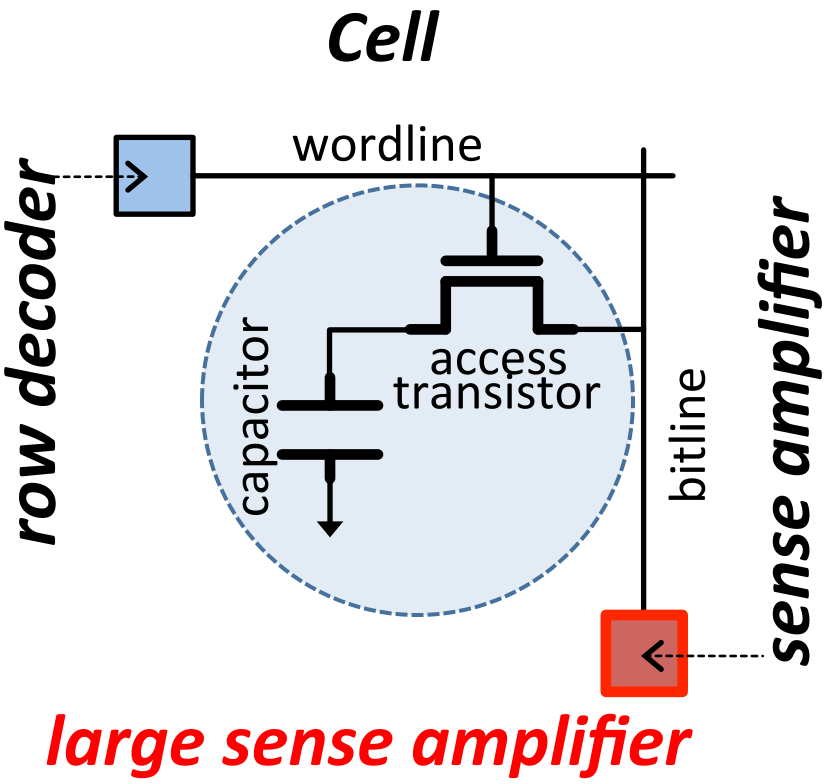


DRAM Latency = Subarray Latency + I/O Latency

Dominant

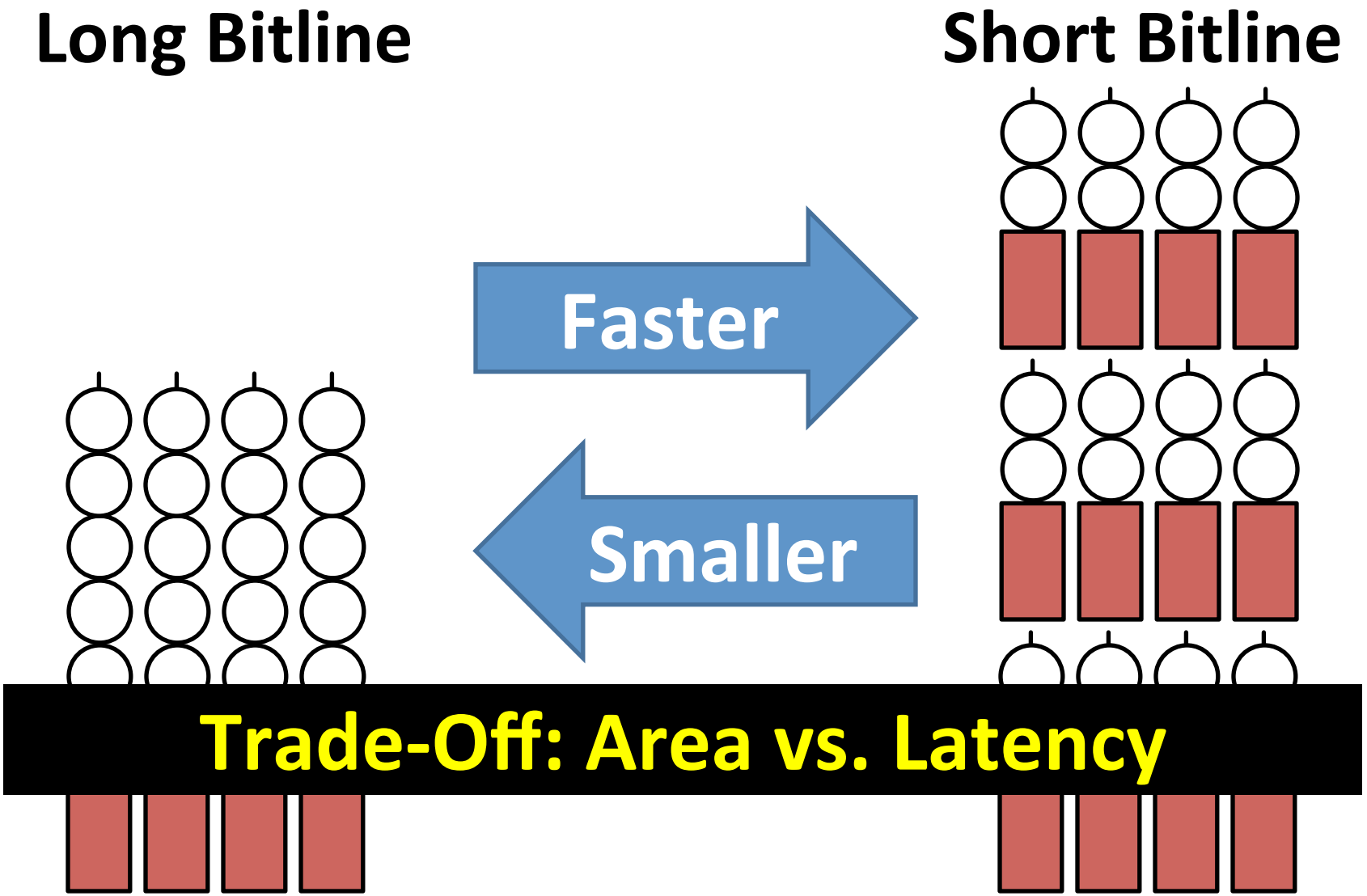
Why is the Subarray So Slow?



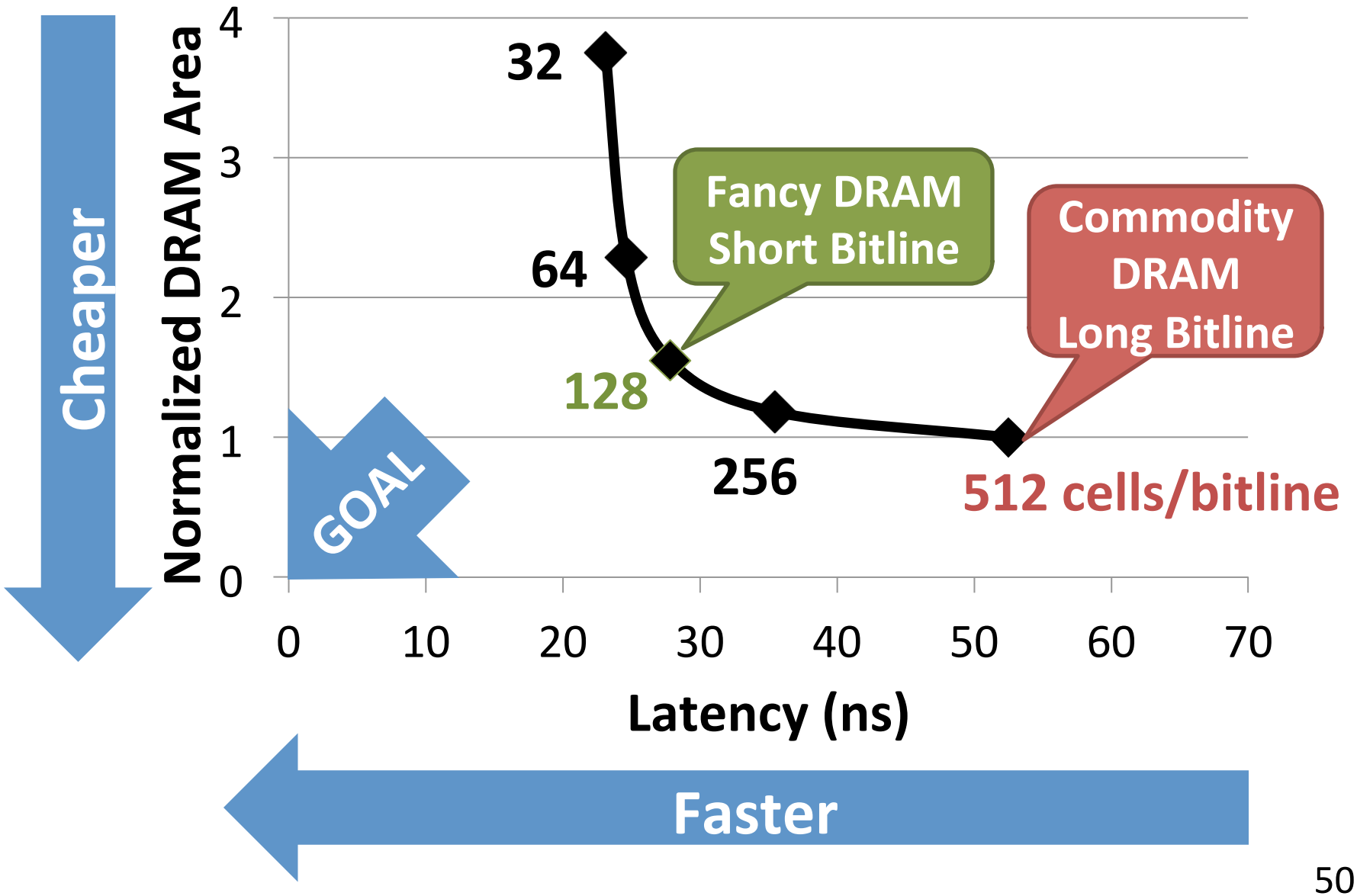


- **Long bitline**
  - **Amortizes sense amplifier cost → Small area**
  - **Large bitline capacitance → High latency & power**

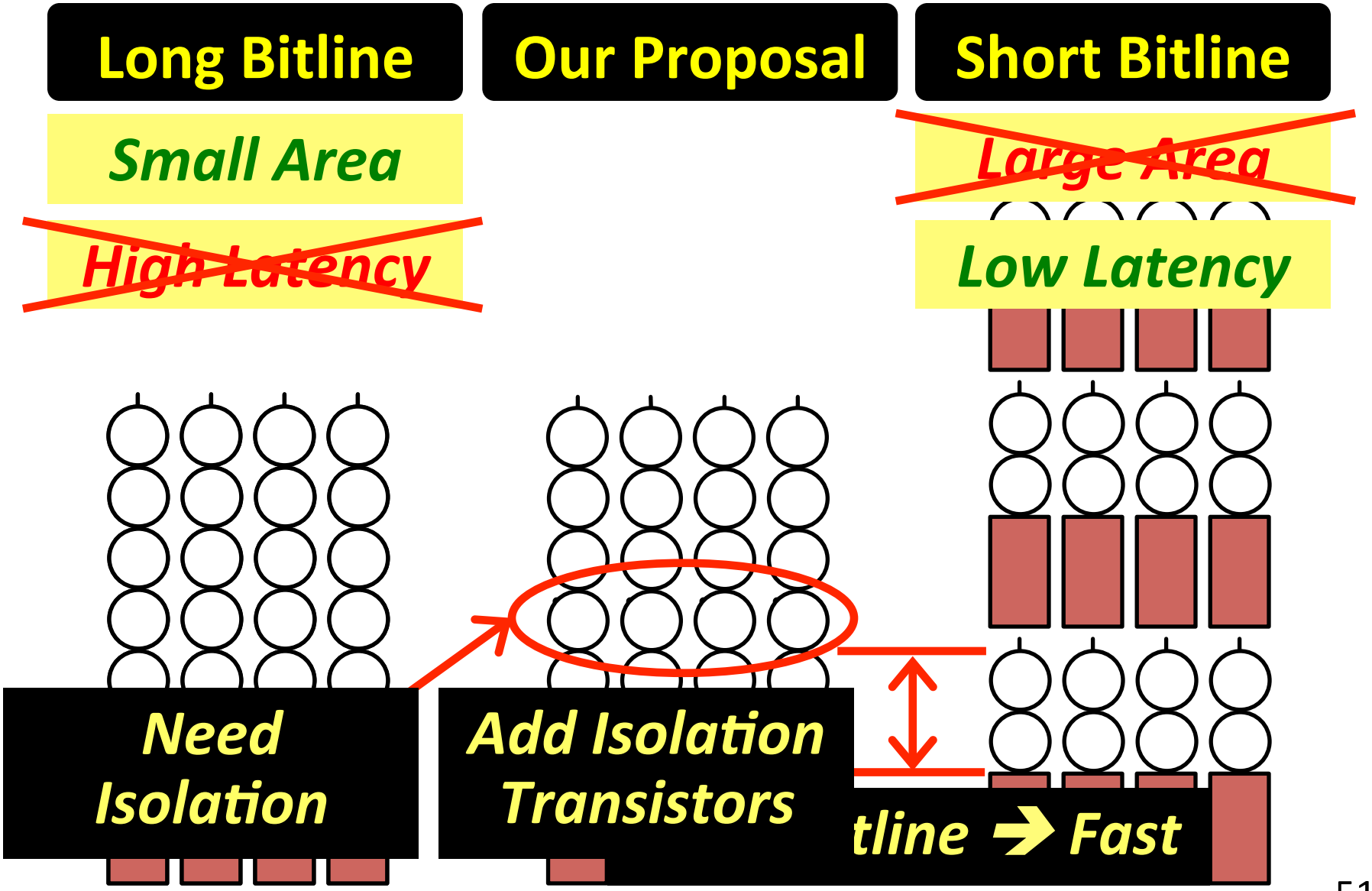
Trade-Off: Area (Die Size) vs. Latency



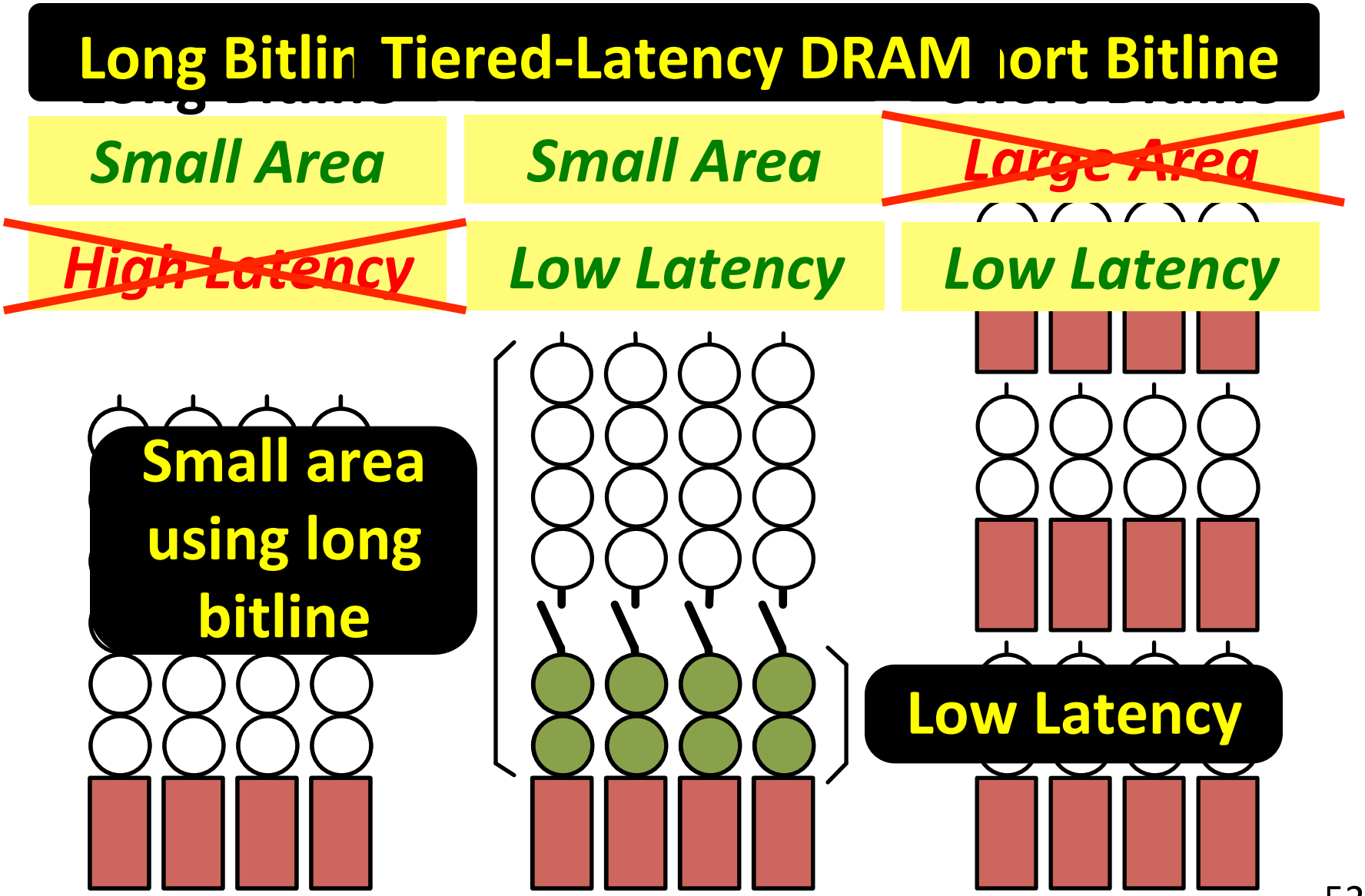
Trade-Off: Area (Die Size) vs. Latency



Approximating the Best of Both Worlds

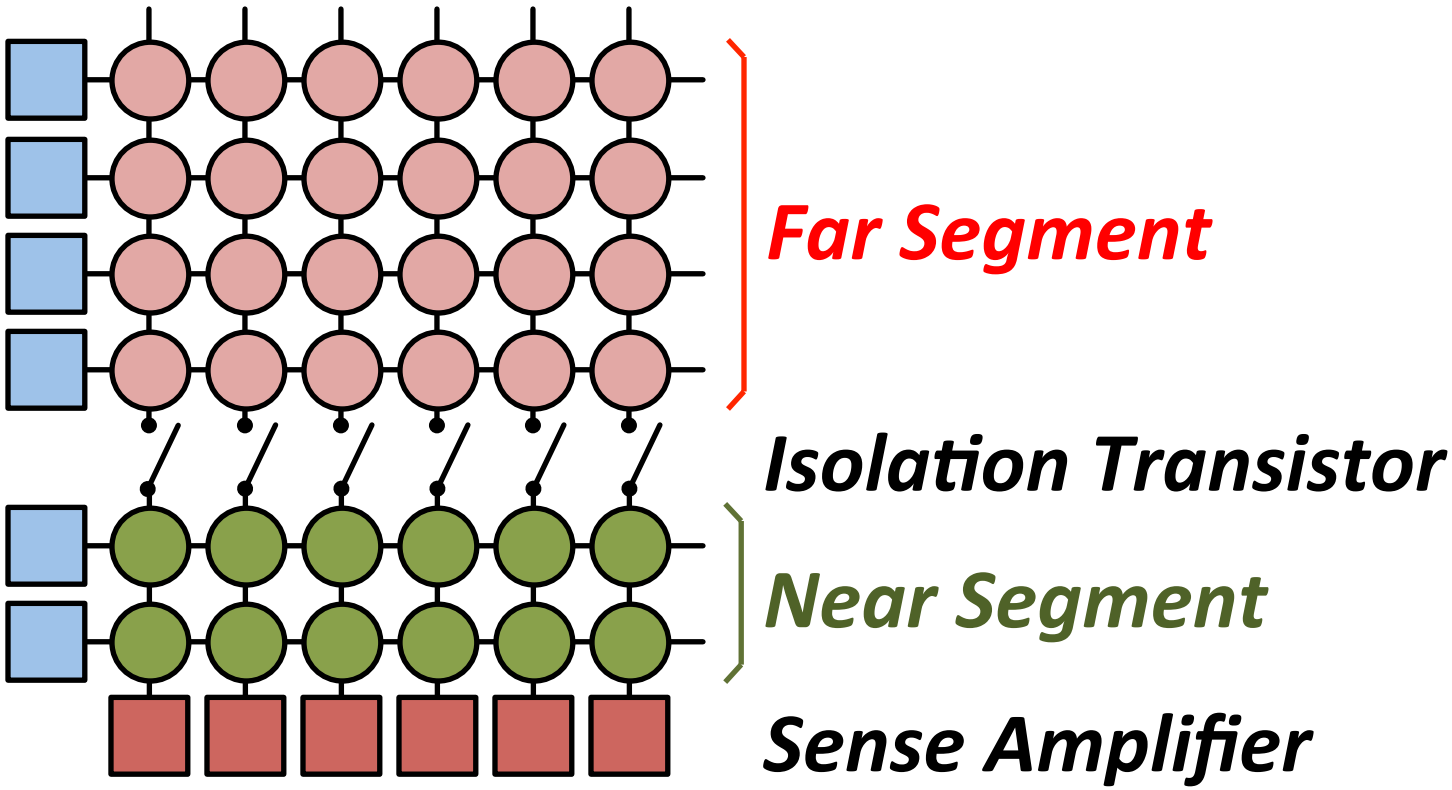


Approximating the Best of Both Worlds



Tiered-Latency DRAM

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



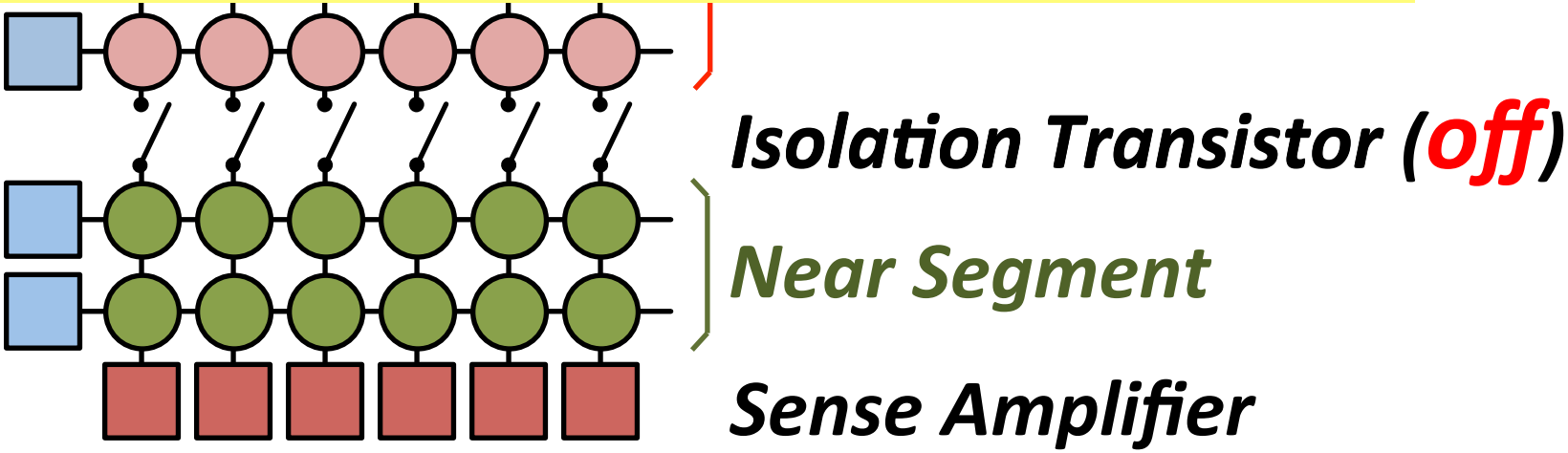
Near Segment Access

- Turn *off* the isolation transistor

Reduced bitline length

Reduced bitline capacitance

➔ Low latency & low power



Far Segment Access

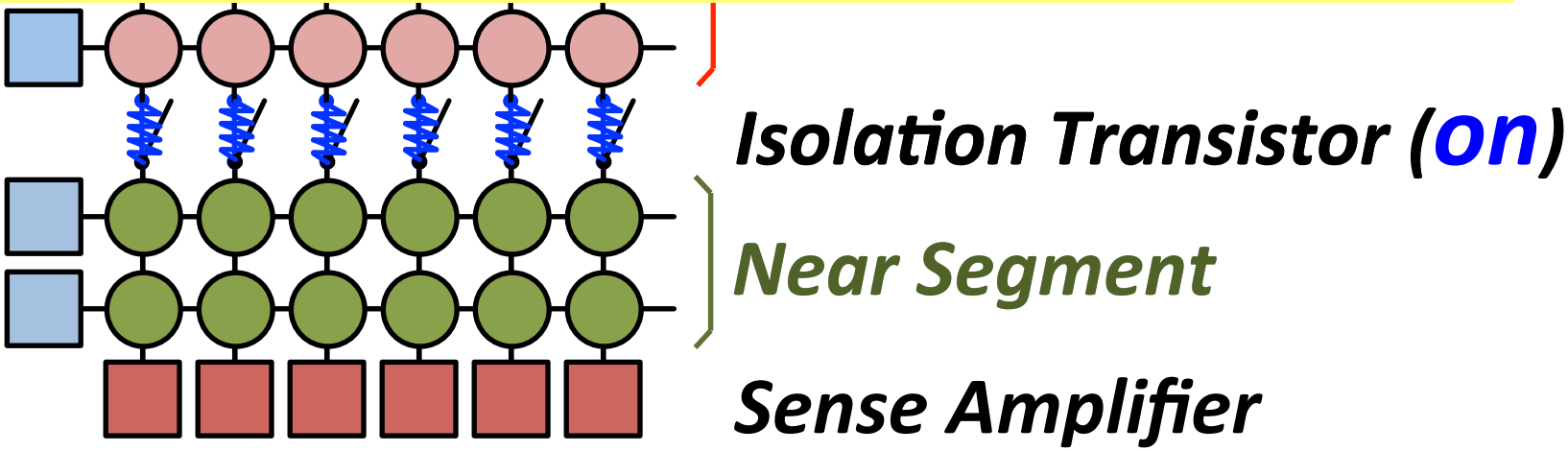
- Turn ***on*** the isolation transistor

Long bitline length

Large bitline capacitance

Additional resistance of isolation transistor

➔ **High latency & high power**

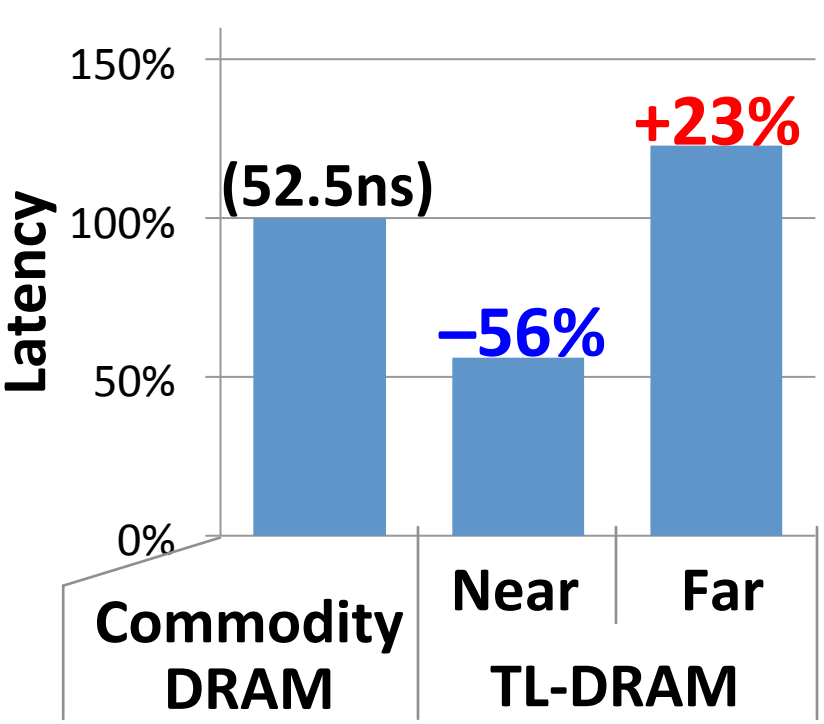


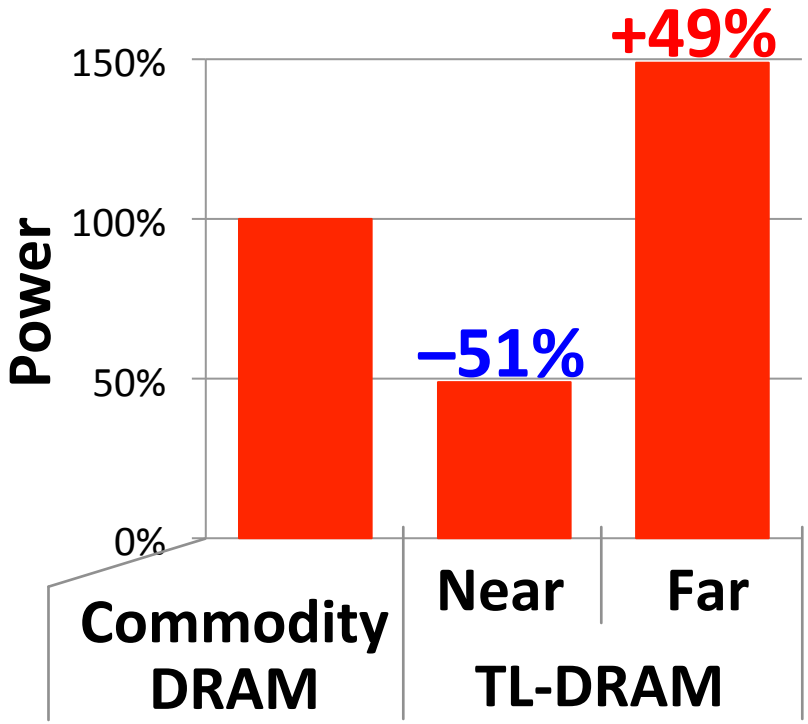
Latency, Power, and Area Evaluation

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

Commodity DRAM vs. TL-DRAM

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

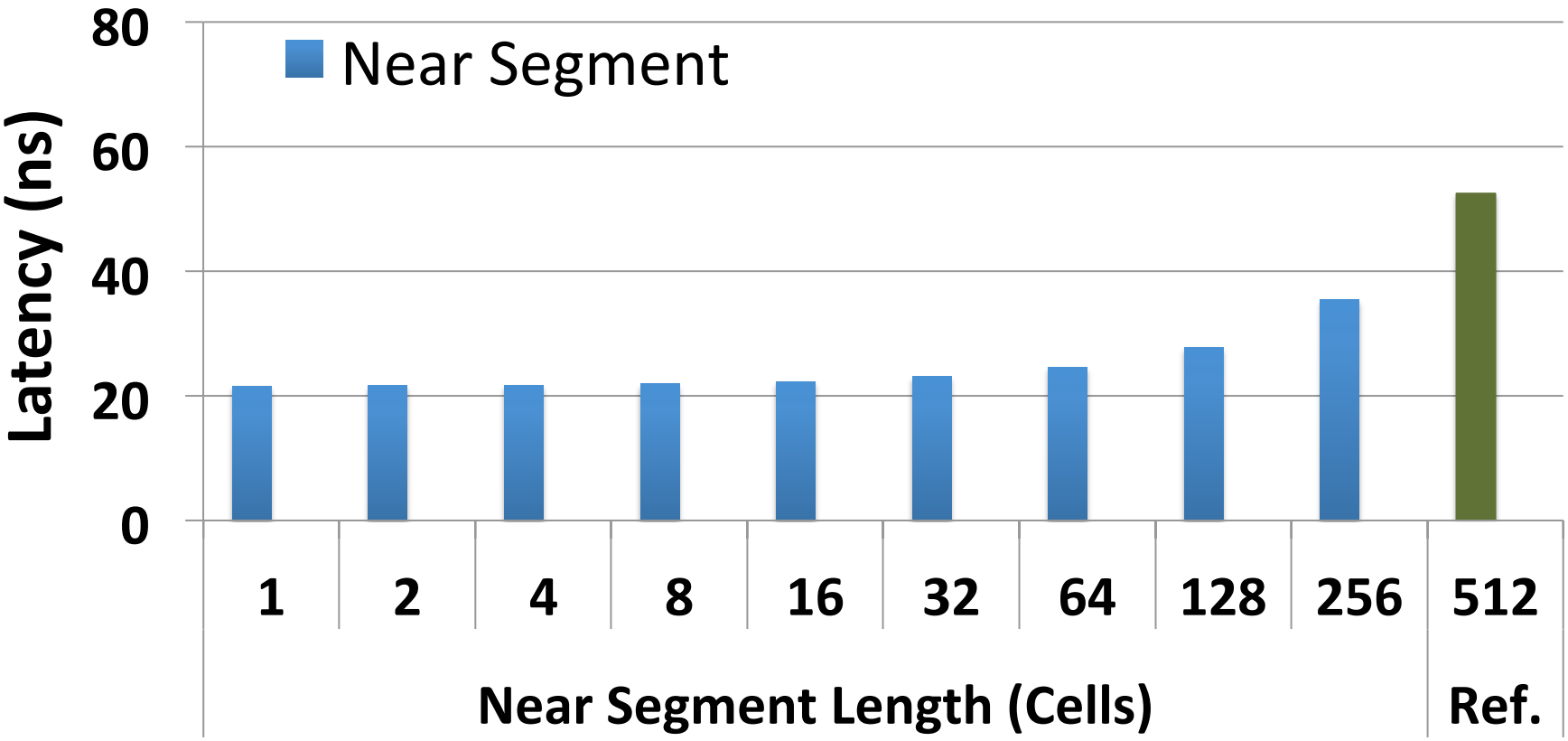




- **DRAM Area Overhead**

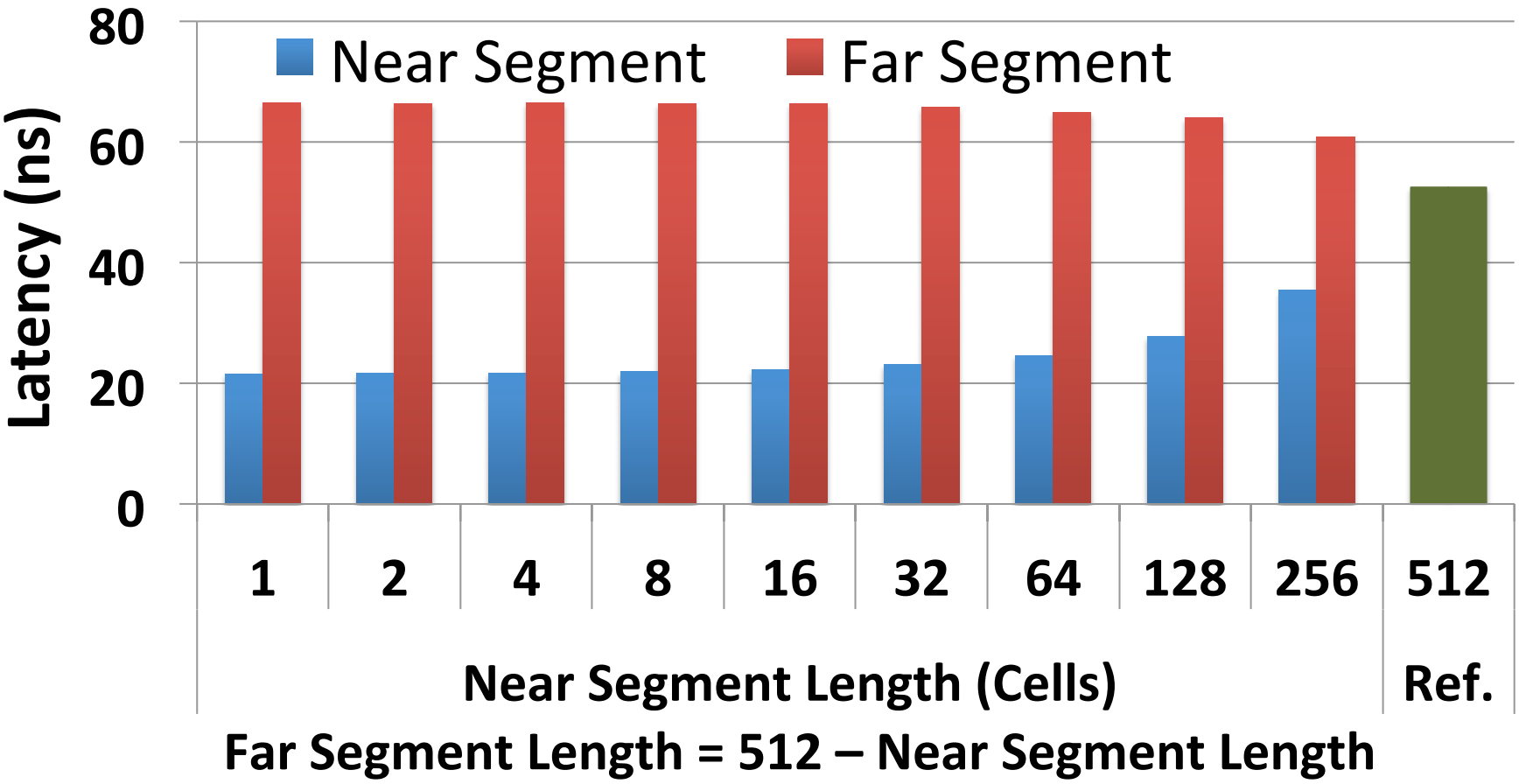
  ~3%: mainly due to the isolation transistors

Latency vs. Near Segment Length



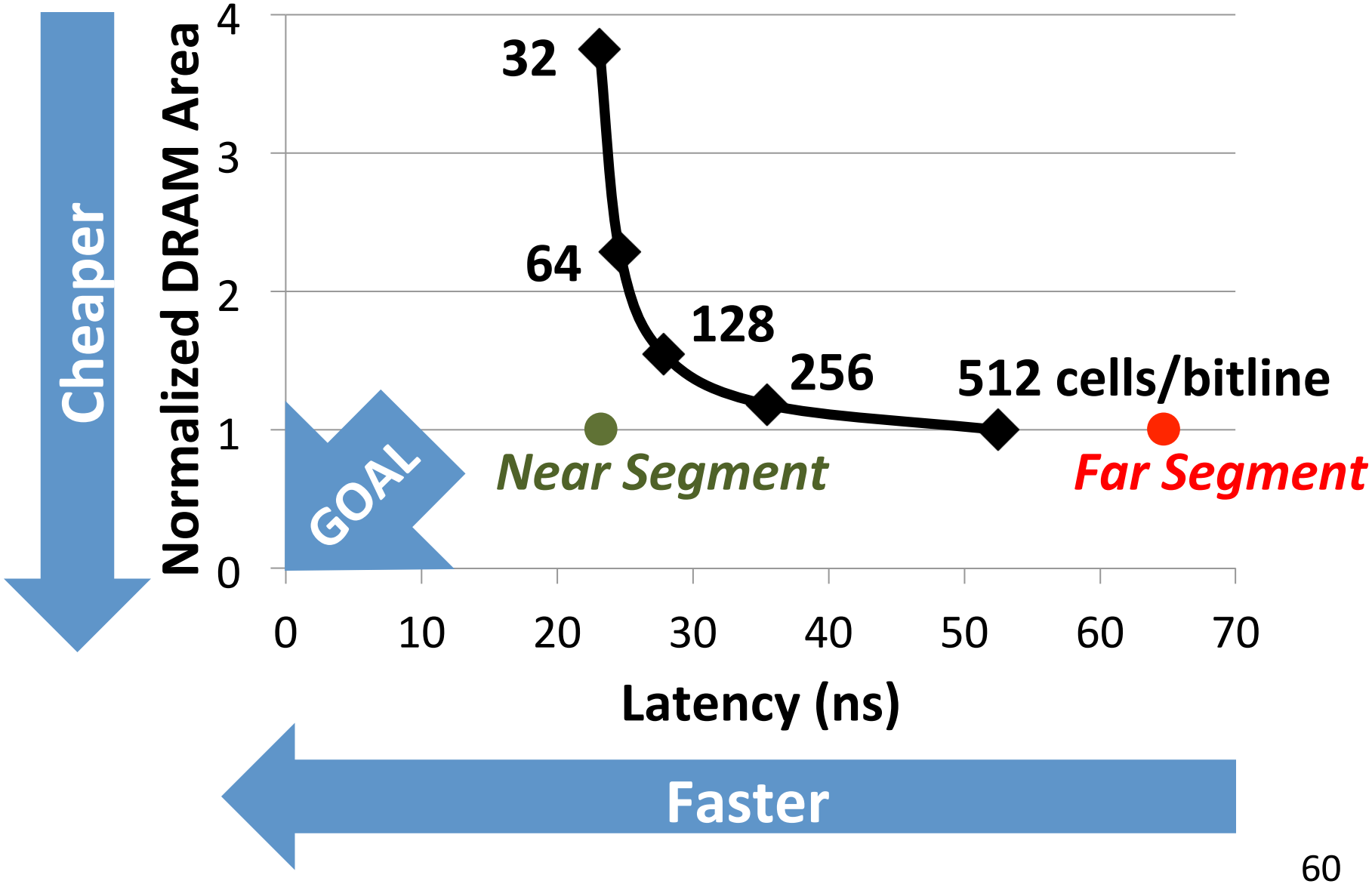
Longer near segment length leads to higher near segment latency

Latency vs. Near Segment Length



Far segment latency is higher than commodity DRAM latency

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



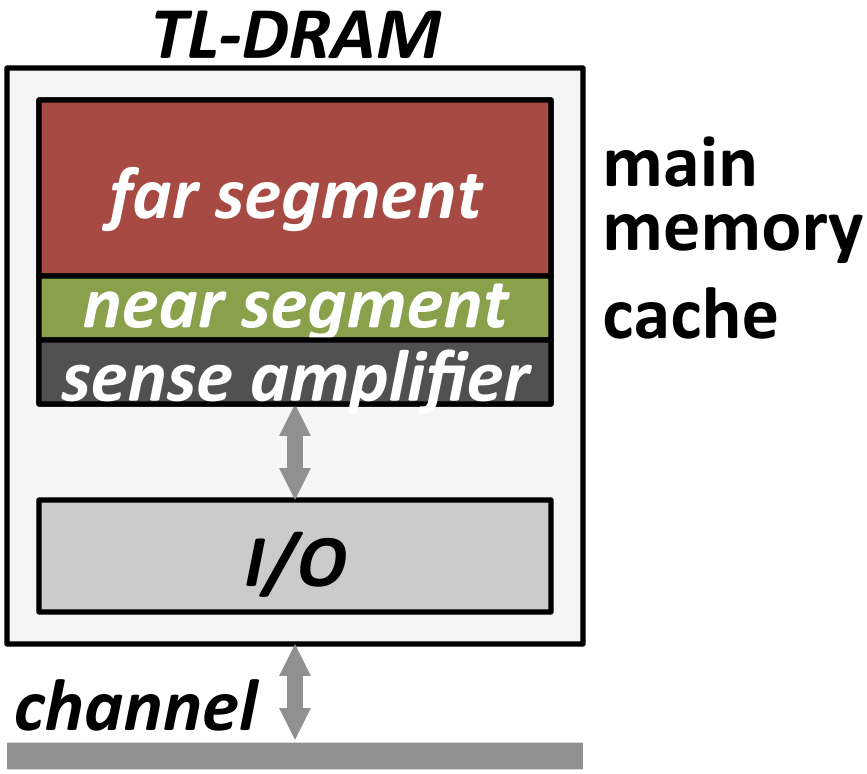
Leveraging Tiered-Latency DRAM

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

- Many potential uses

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

Near Segment as Hardware-Managed Cache

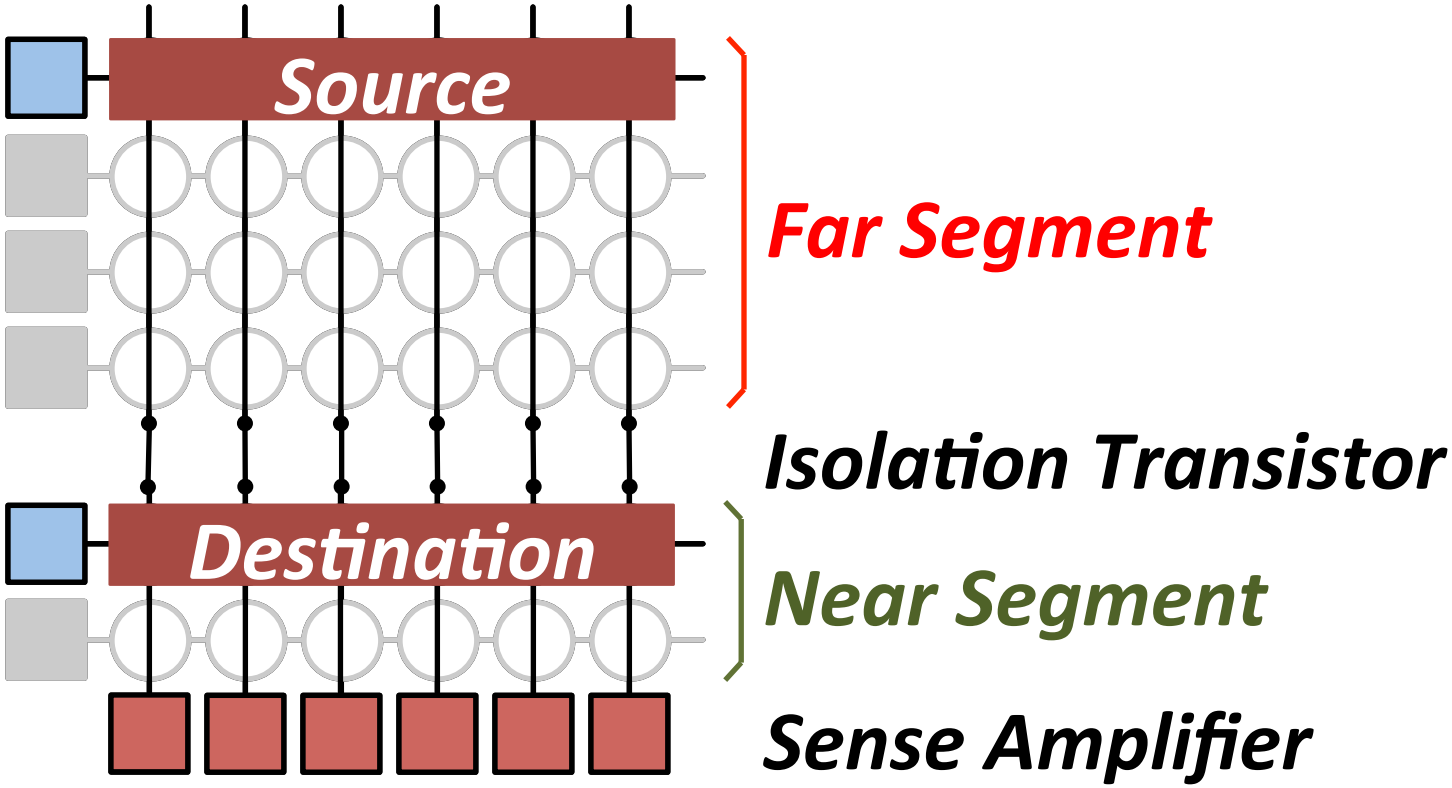


- **Challenge 1:** How to efficiently migrate a row between segments?
- **Challenge 2:** How to efficiently manage the cache?

Inter-Segment Migration

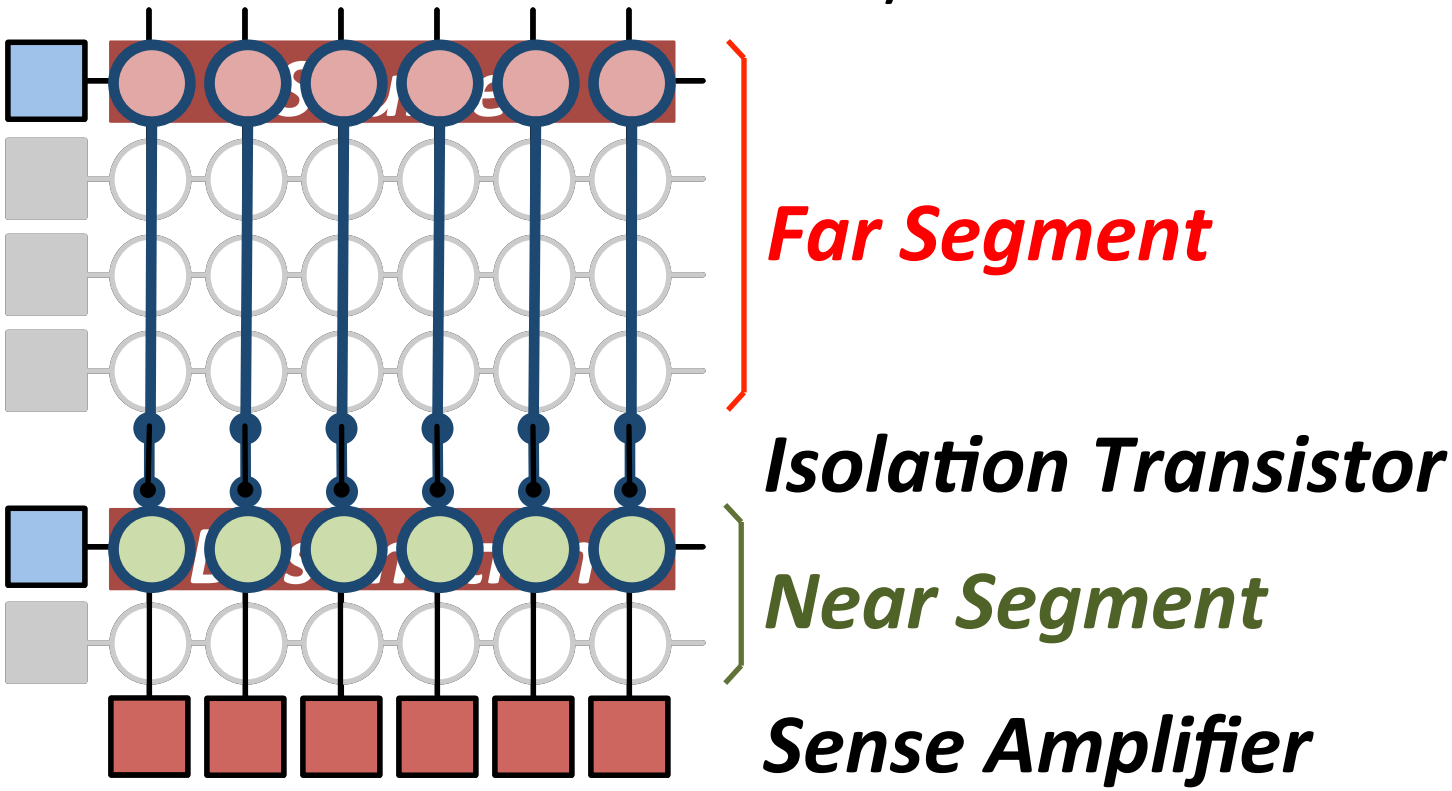
- **Goal:** Migrate source row into destination row
- **Naïve way:** Memory controller reads the source row *byte by byte* and writes to destination row *byte by byte*

→ High latency



Inter-Segment Migration

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



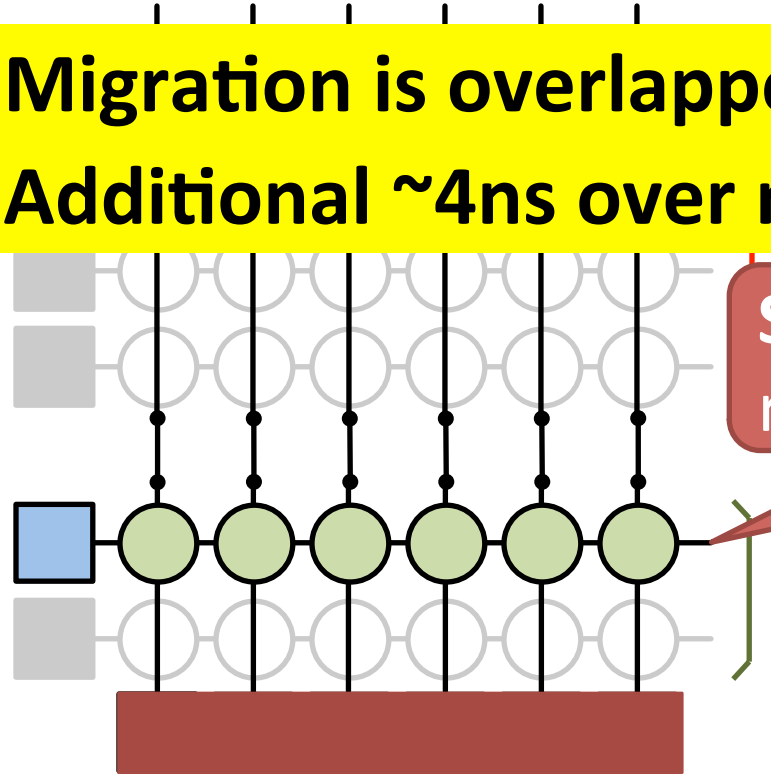
Inter-Segment Migration

- **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



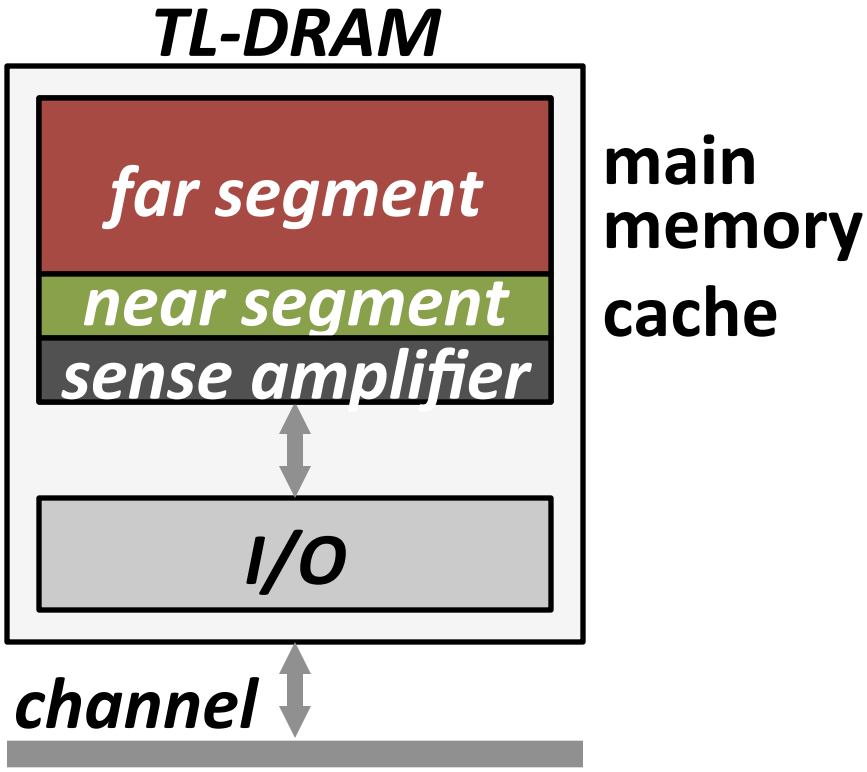
Step 2: Activate destination row to connect cell and bitline

Isolation Transistor

Near Segment

Sense Amplifier

Near Segment as Hardware-Managed Cache



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

Evaluation Methodology

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

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

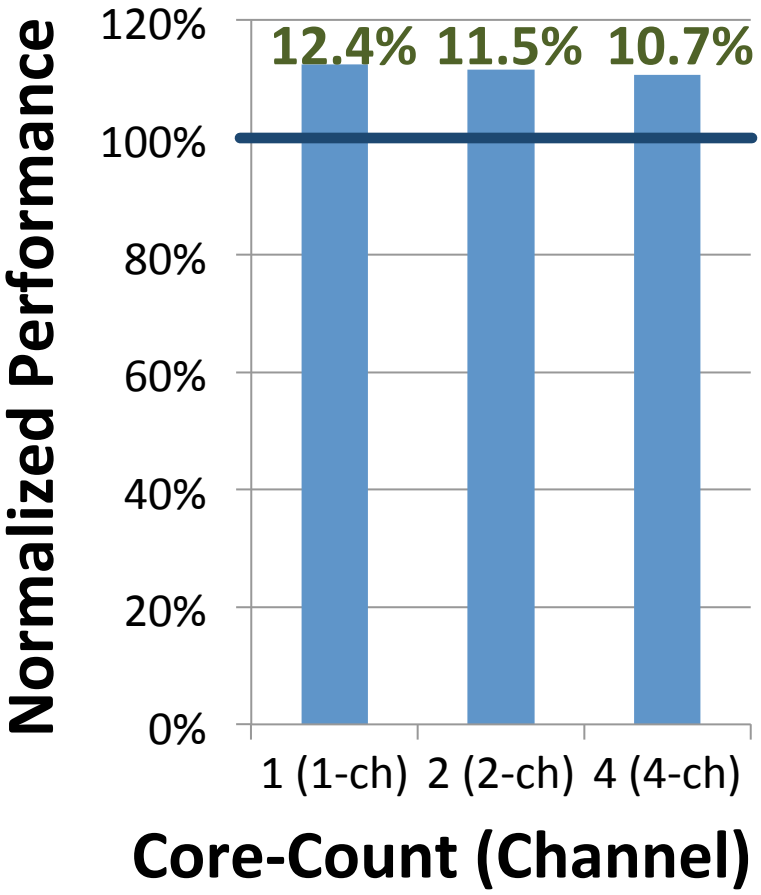
- **Performance Metrics**
  - Single-core: Instructions-Per-Cycle
  - Multi-core: Weighted speedup

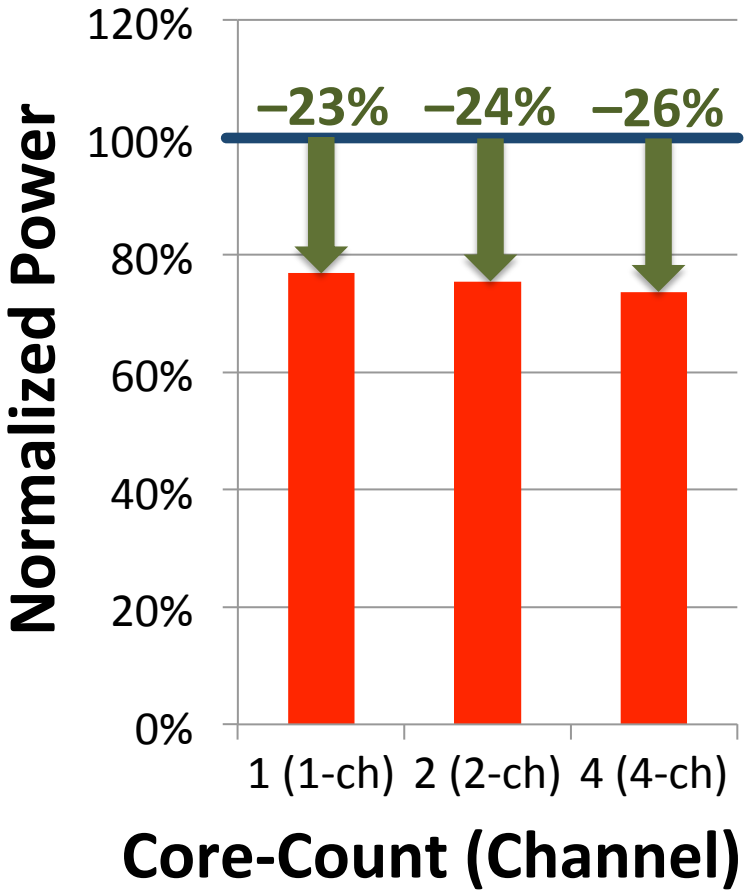
Configurations

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

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

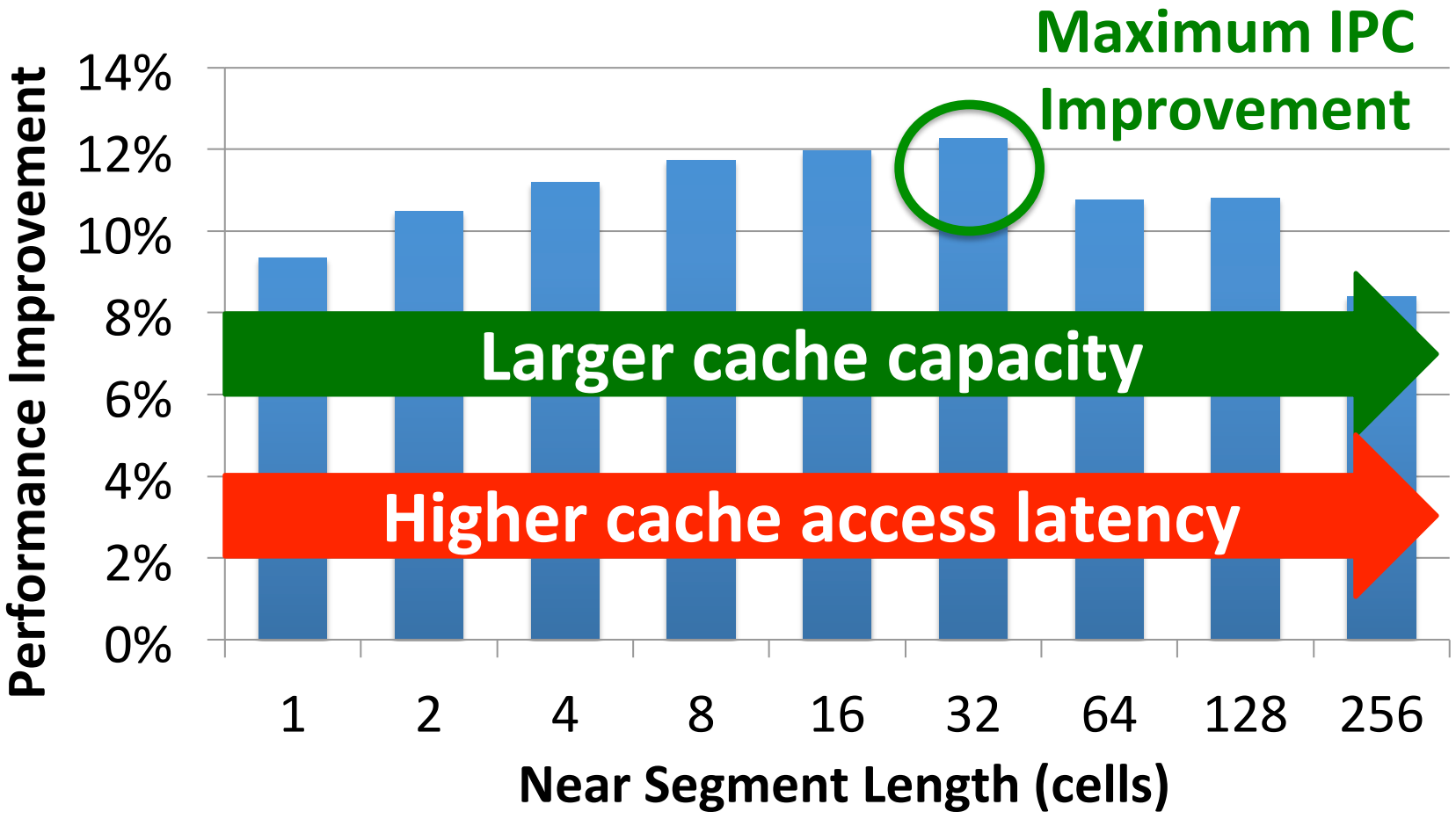
Performance & Power Consumption





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

Single-Core: Varying Near Segment Length



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

Other Mechanisms & Results

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

Summary of TL-DRAM

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

Agenda

- Major Trends Affecting Main Memory
- DRAM Scaling Problem and Solution Directions
- Three New Techniques for DRAM
  - RAIDR: Reducing Refresh Impact
  - TL-DRAM: Reducing DRAM Latency
  - SALP: Reducing Bank Conflict Impact
- Ongoing Research
- Summary

The Memory Bank Conflict Problem

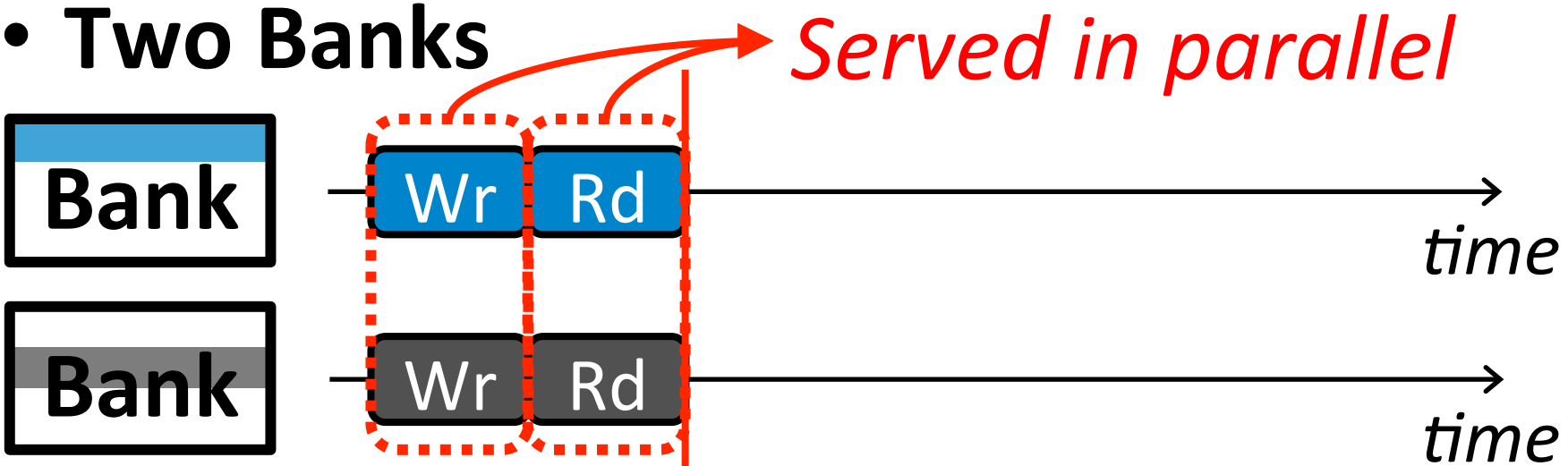
- Two requests to the same bank are serviced serially
- Problem: Costly in terms of performance and power
- Goal: We would like to reduce bank conflicts without increasing the number of banks (at low cost)

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

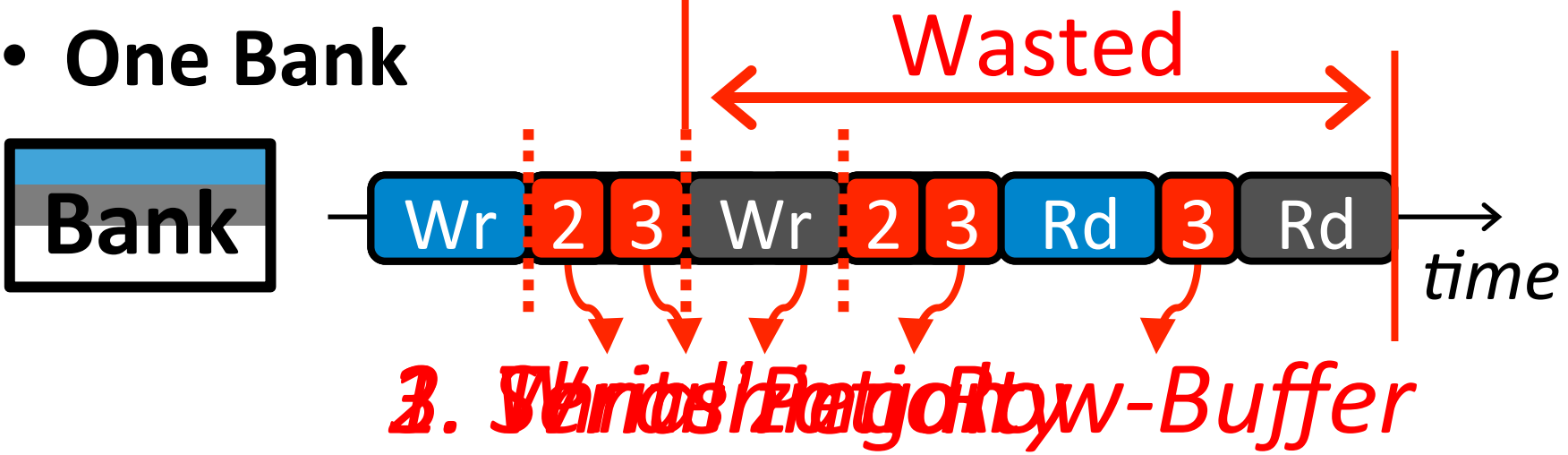
- Kim, Seshadri, Lee, Liu, Mutlu, "A Case for Exploiting Subarray-Level Parallelism in DRAM," ISCA 2012.

The Problem with Memory Bank Conflicts

- **Two Banks**



- **One Bank**



Goal

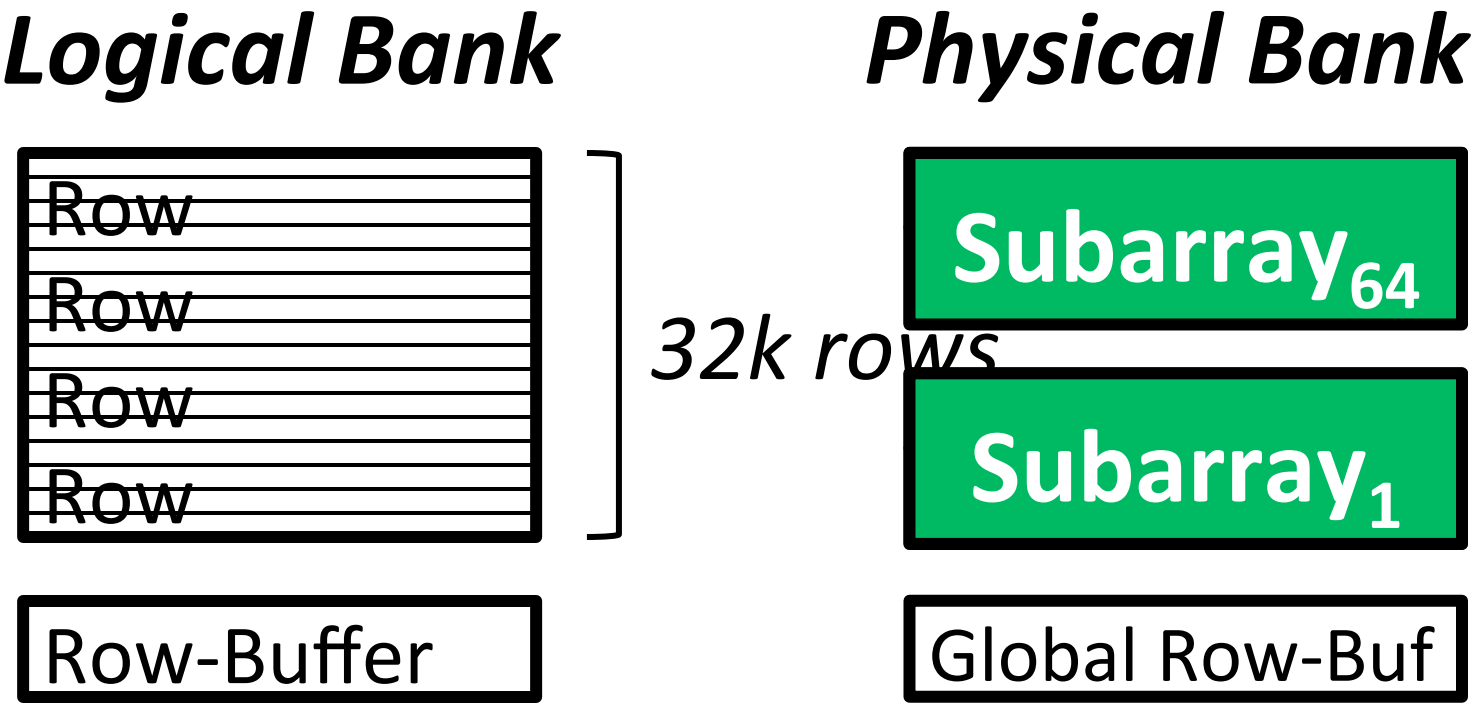
- **Goal:** *Mitigate the detrimental effects of* ***bank conflicts*** *in a cost-effective manner*

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

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

Key Observation #1

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



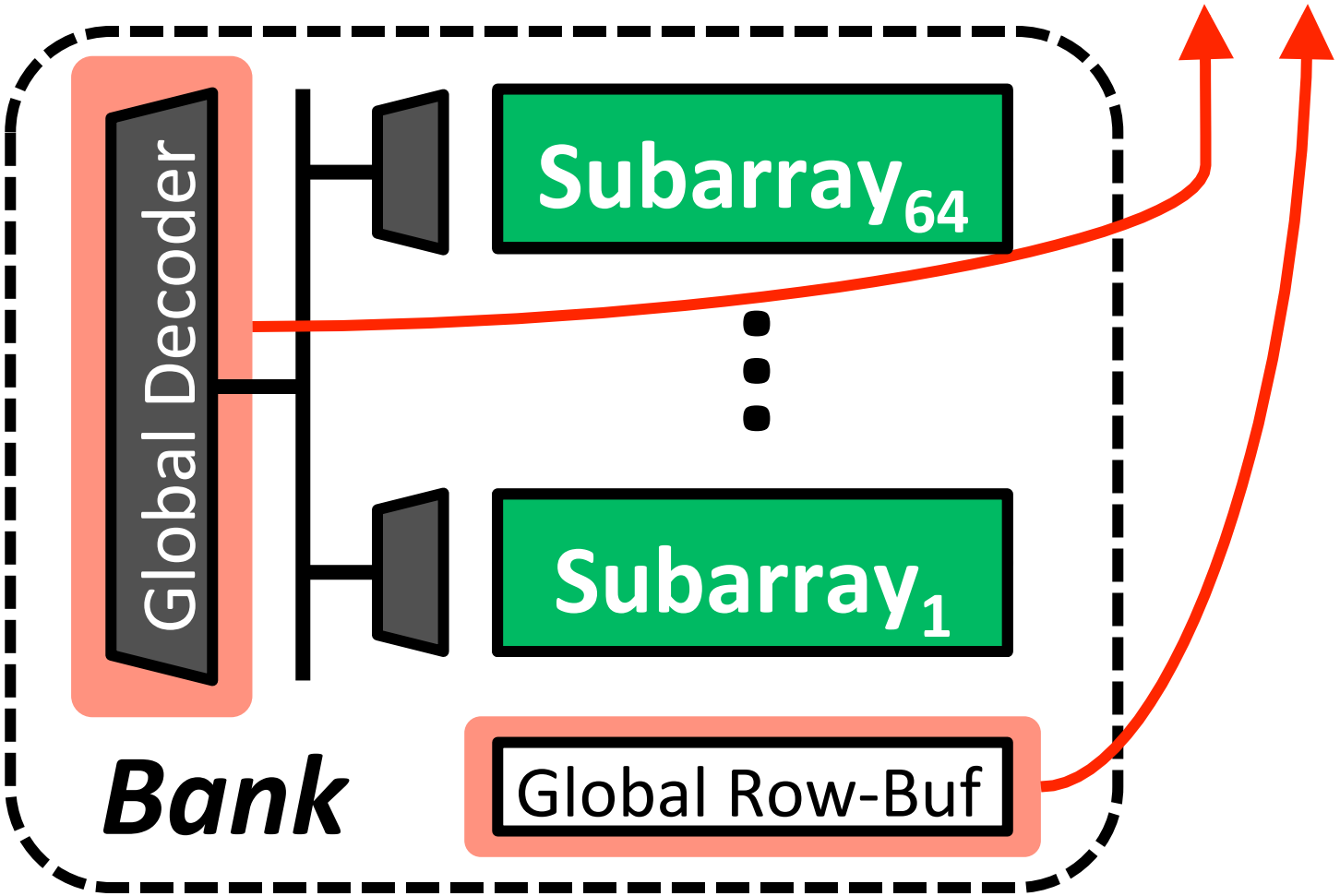
A *single* row-buffer cannot drive *all* rows

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

Key Observation #2

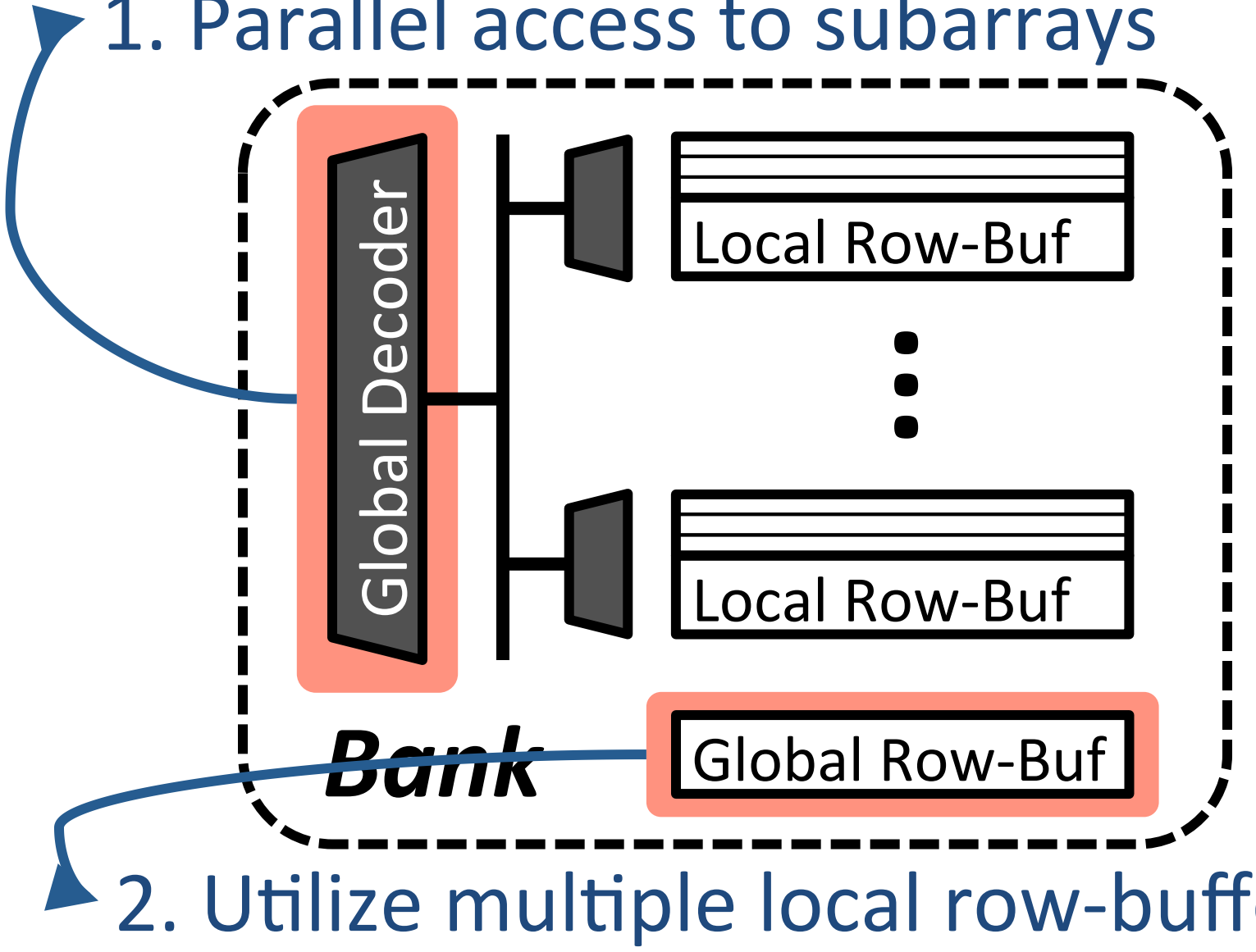
Each subarray is mostly independent…

– except occasionally sharing ***global structures***



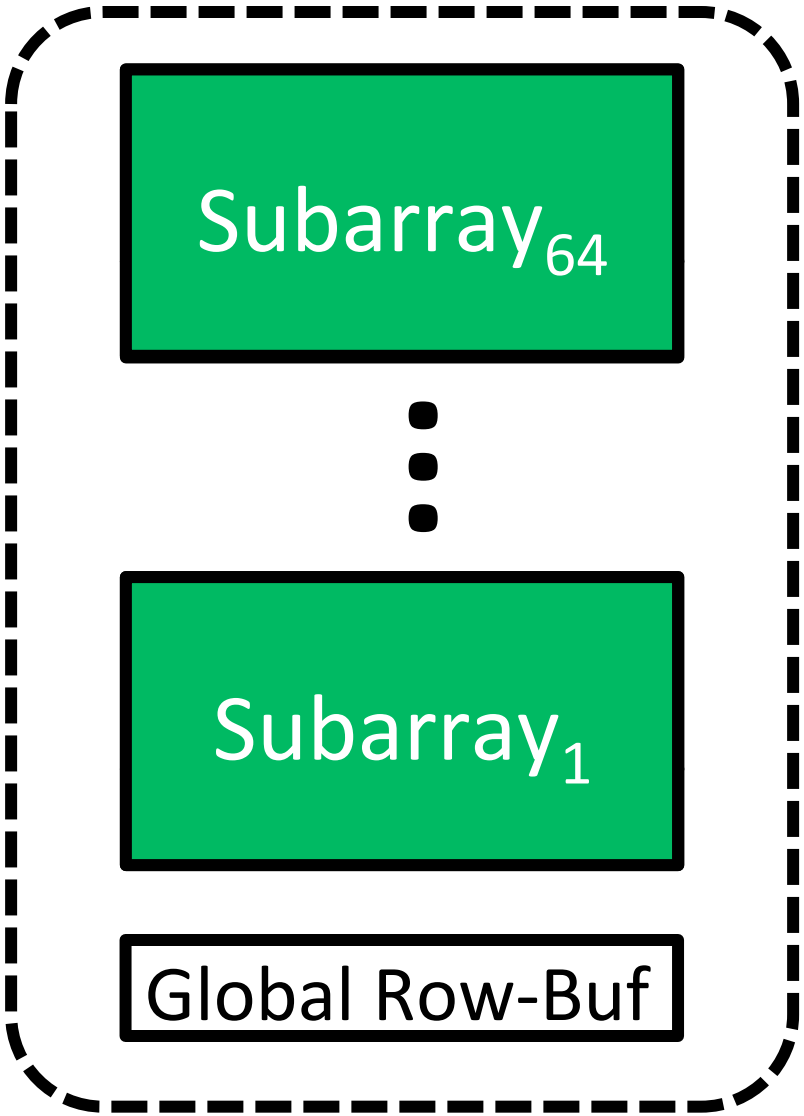
Key Idea: Reduce Sharing of Globals

1. Parallel access to subarrays



2. Utilize multiple local row-buffers

Overview of Our Mechanism



1. Parallelize

2. Utilize multiple local row-buffers

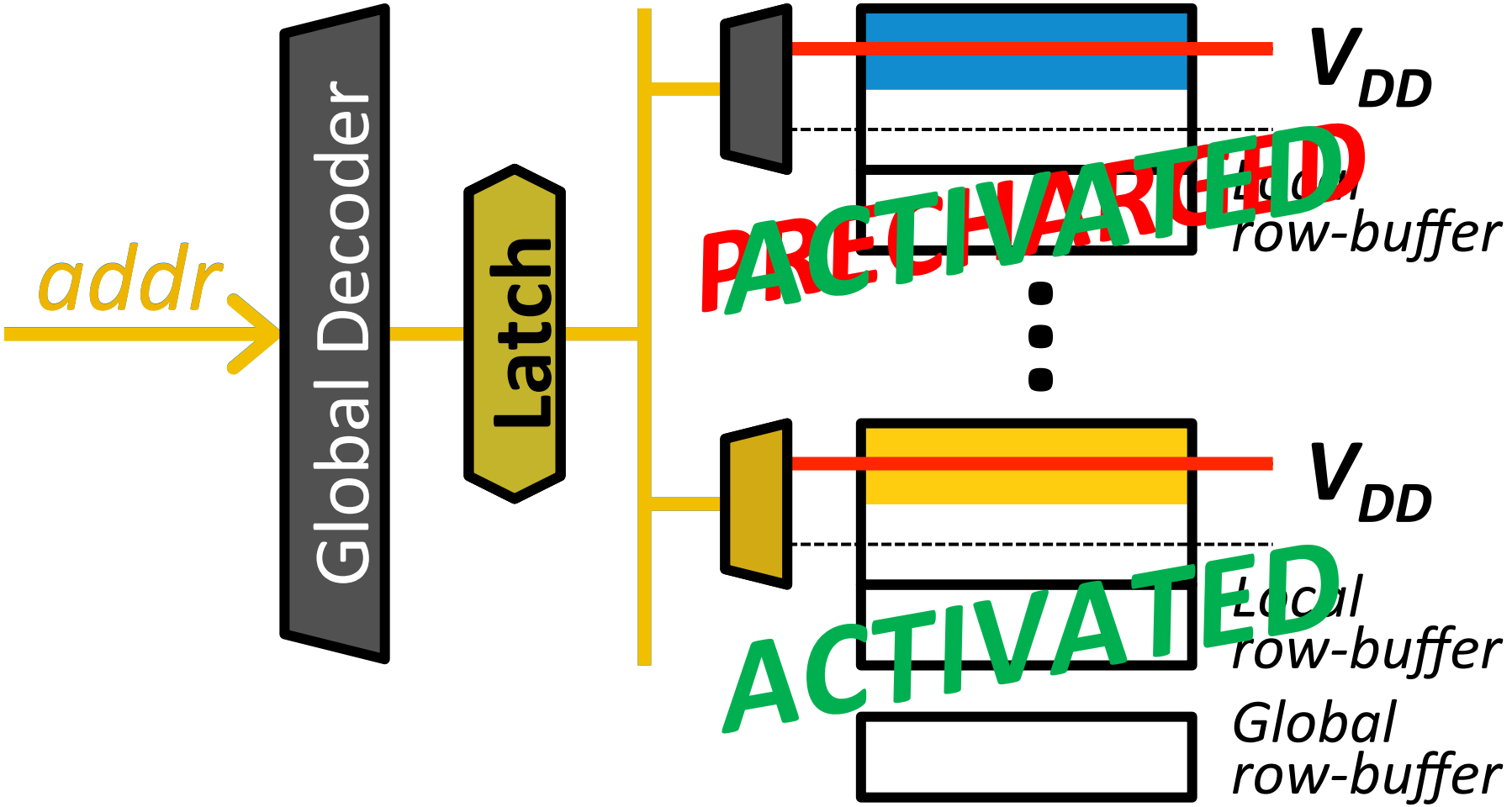
To same bank... but diff. subarrays

Challenges: Global Structures

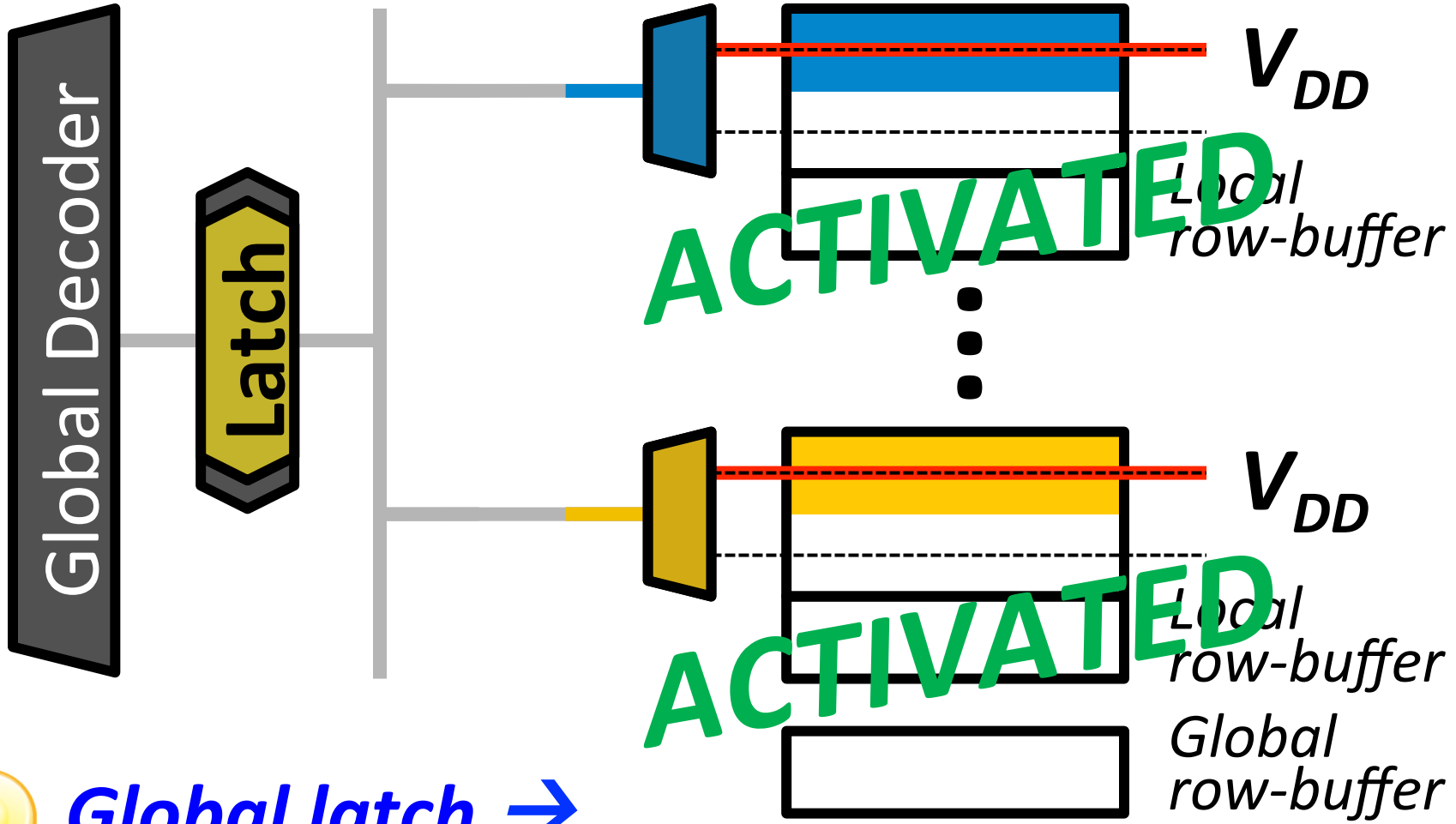
1. Global Address Latch

2. Global Bitlines

Challenge #1. Global Address Latch



Solution #1. Subarray Address Latch



Global latch → local latches

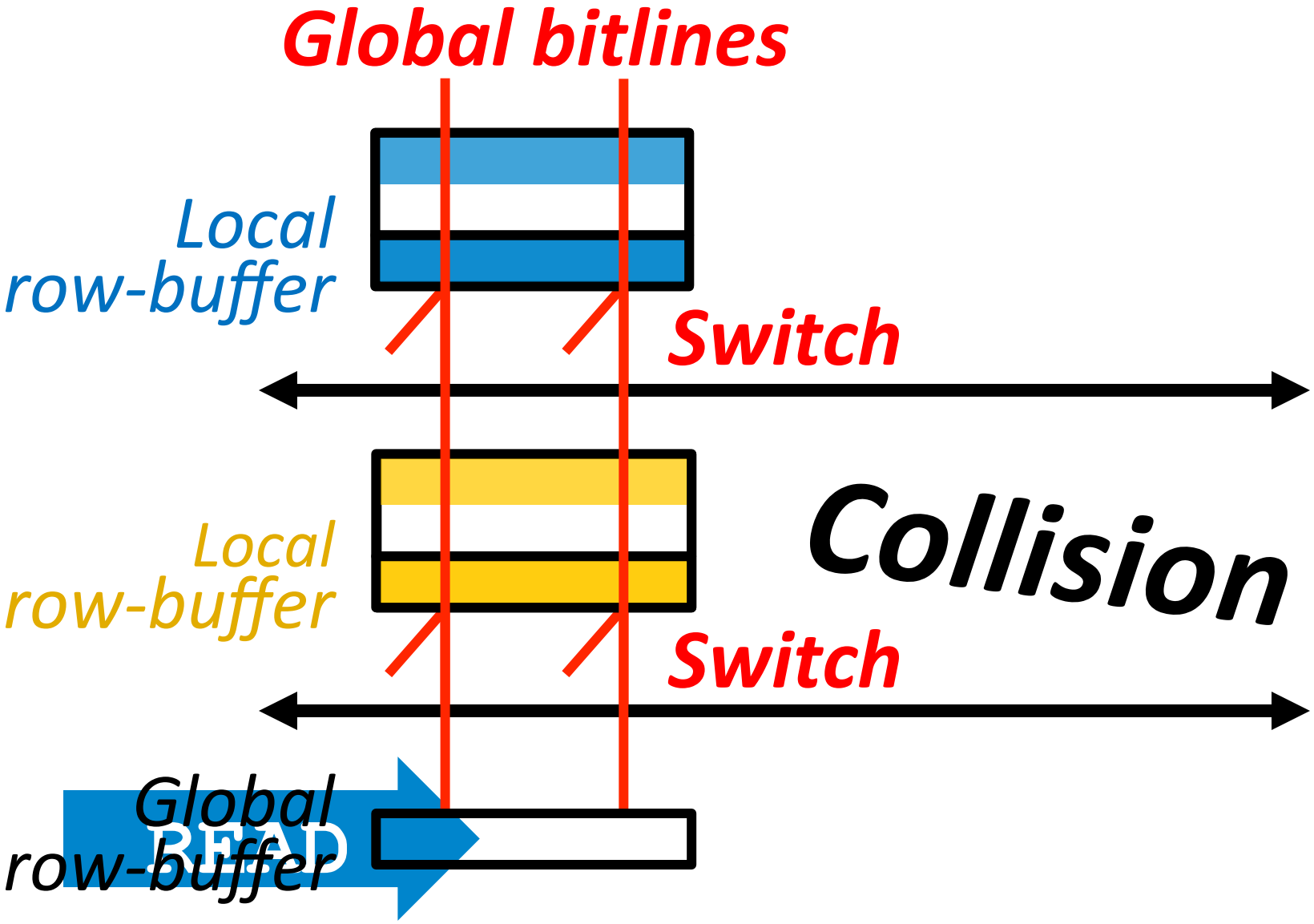
Challenges: Global Structures

1. Global Address Latch

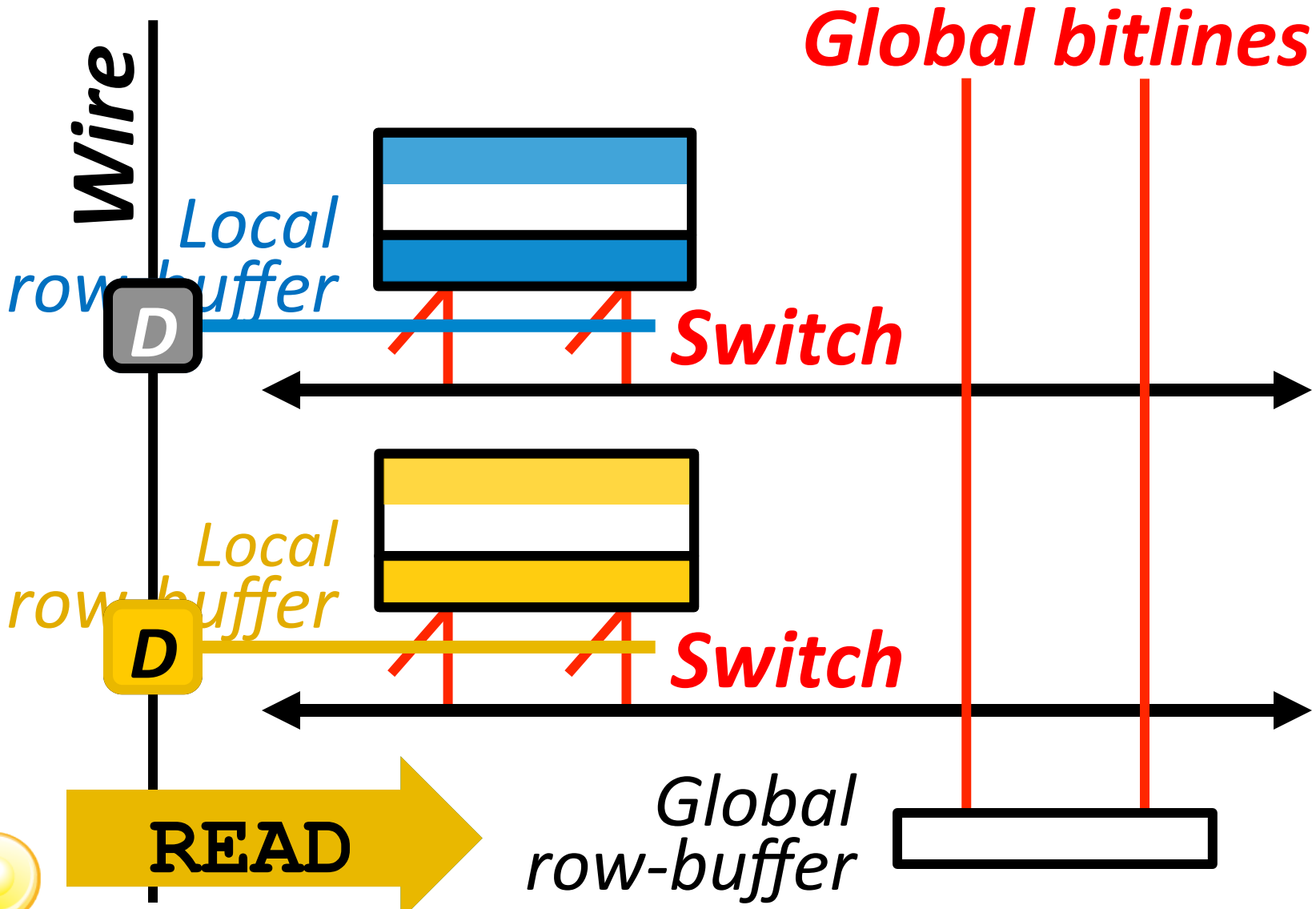
- Problem: Only *one* raised wordline
- Solution: **Subarray Address Latch**

2. Global Bitlines

Challenge #2. Global Bitlines



Solution #2. Designated-Bit Latch



Selectively connect local to global

Challenges: Global Structures

1. Global Address Latch

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

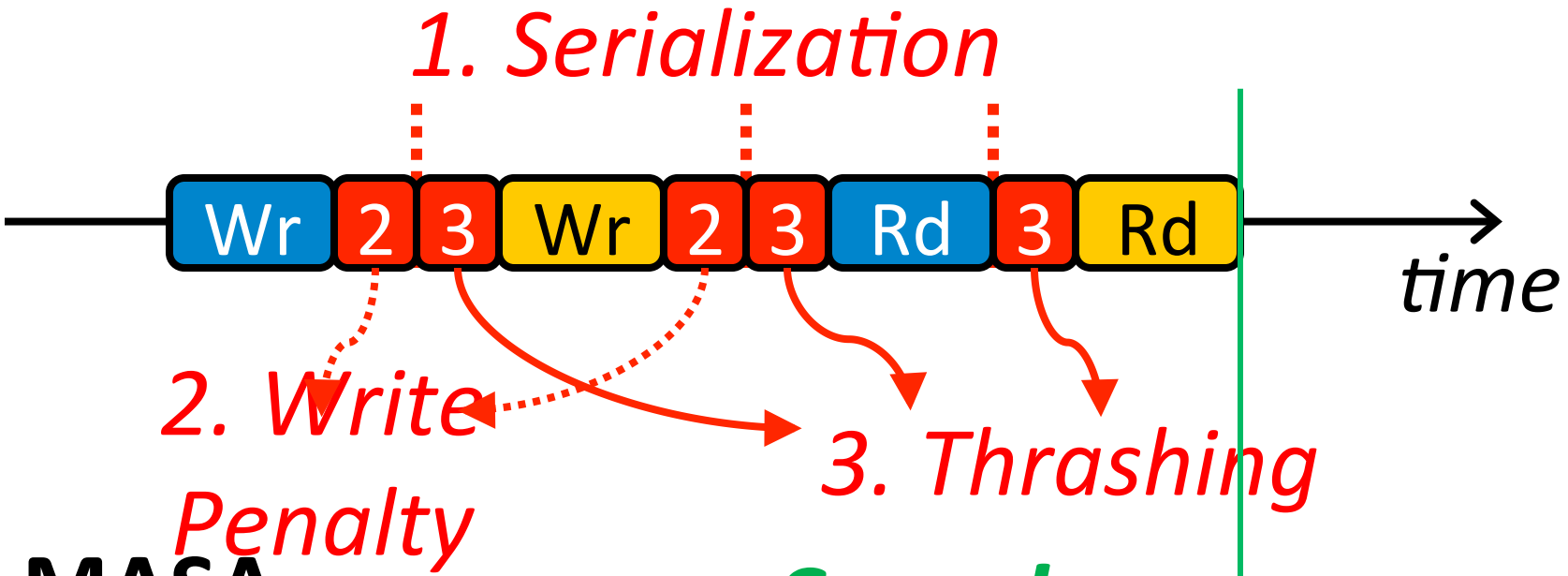
2. Global Bitlines

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

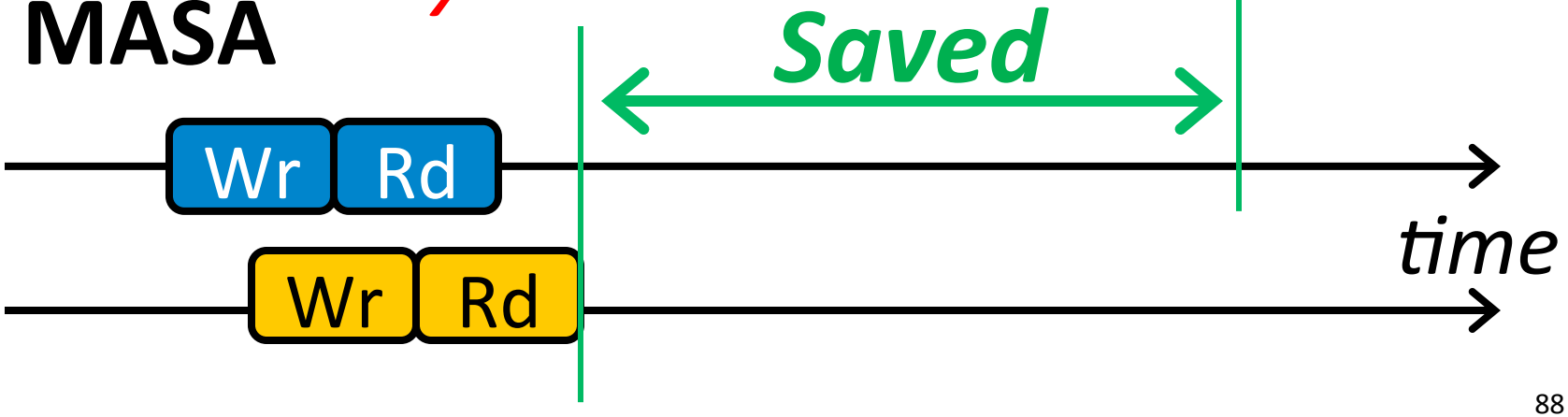
MASA (Multitude of Activated Subarrays)

MASA: Advantages

- Baseline (Subarray-Oblivious)



- **MASA**

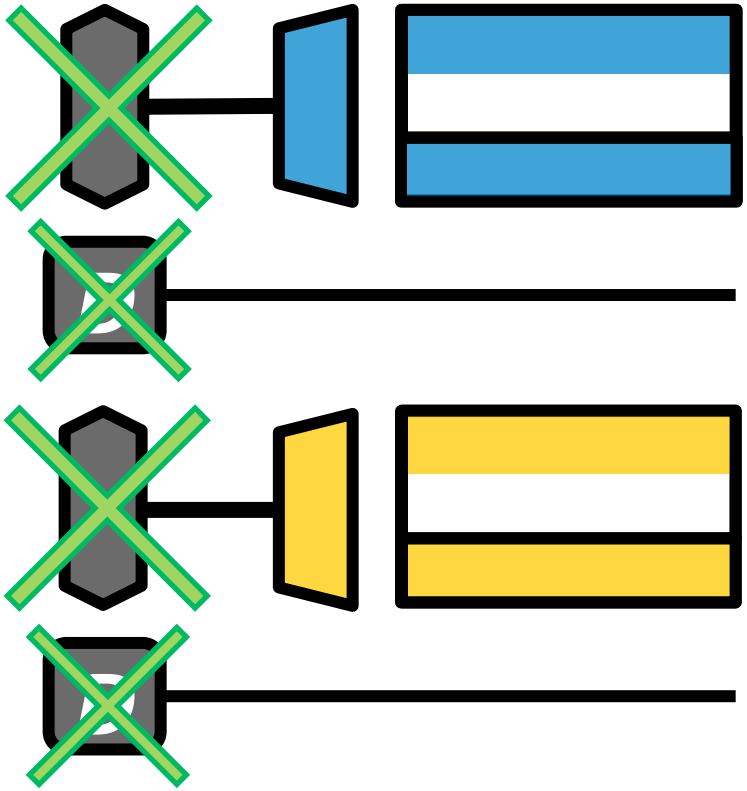


MASA: Overhead

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

Cheaper Mechanisms

Latches



1. Serialization
2. Wr-Penalty
3. Thrashing

MASA

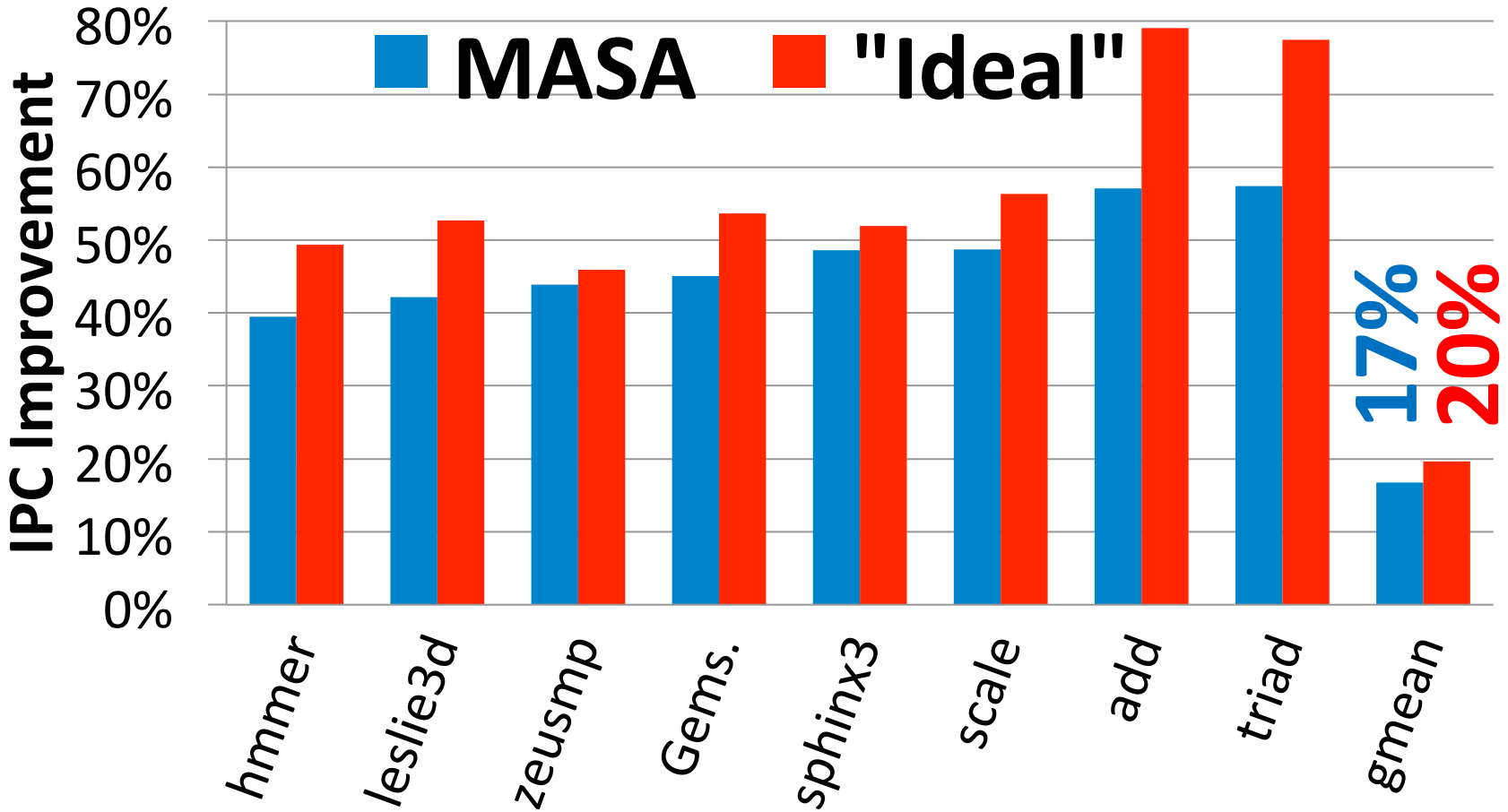
SALP-2

SALP-1

System Configuration

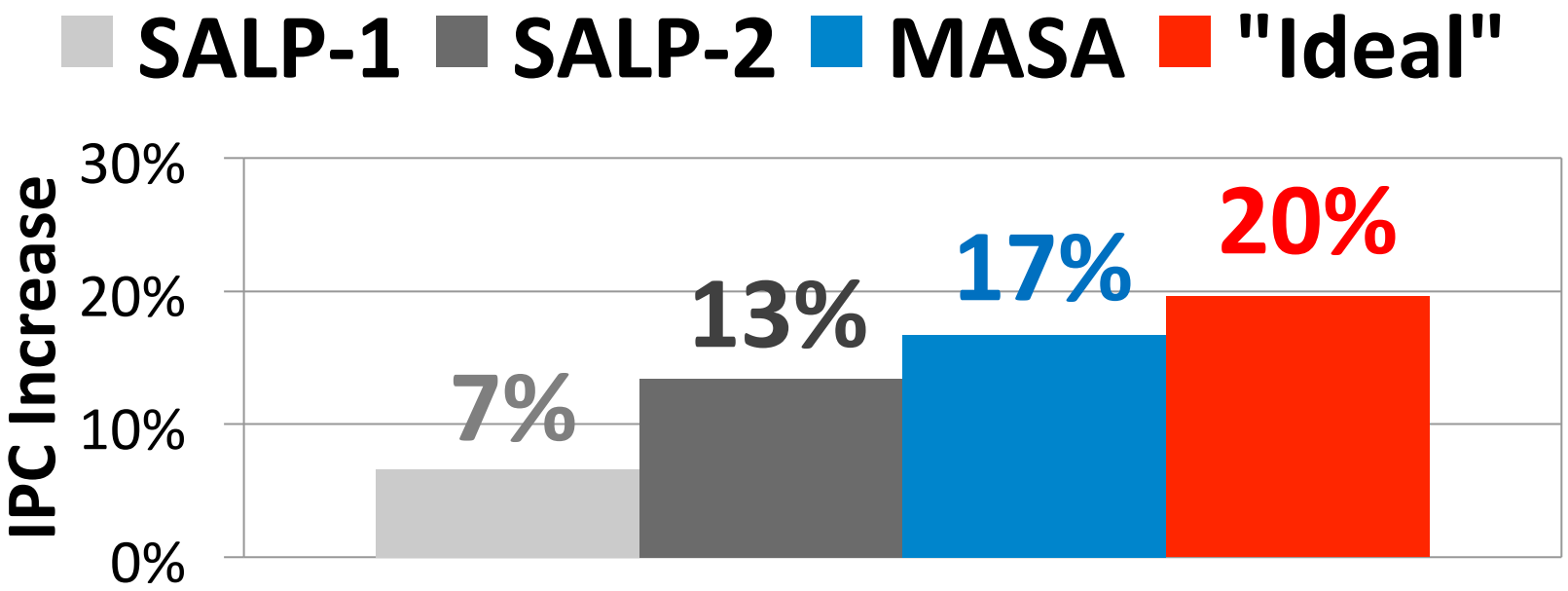
- **System Configuration**
  - CPU: 5.3GHz, 128 ROB, 8 MSHR
  - LLC: 512kB per-core slice
- **Memory Configuration**
  - DDR3-1066
  - ***(default)* 1 channel, 1 rank, 8 banks, 8 subarrays-per-bank**
  - *(sensitivity)* 1-8 chans, 1-8 ranks, 8-64 banks, 1-128 subarrays
- **Mapping & Row-Policy**
  -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

SALP: Single-core Results



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

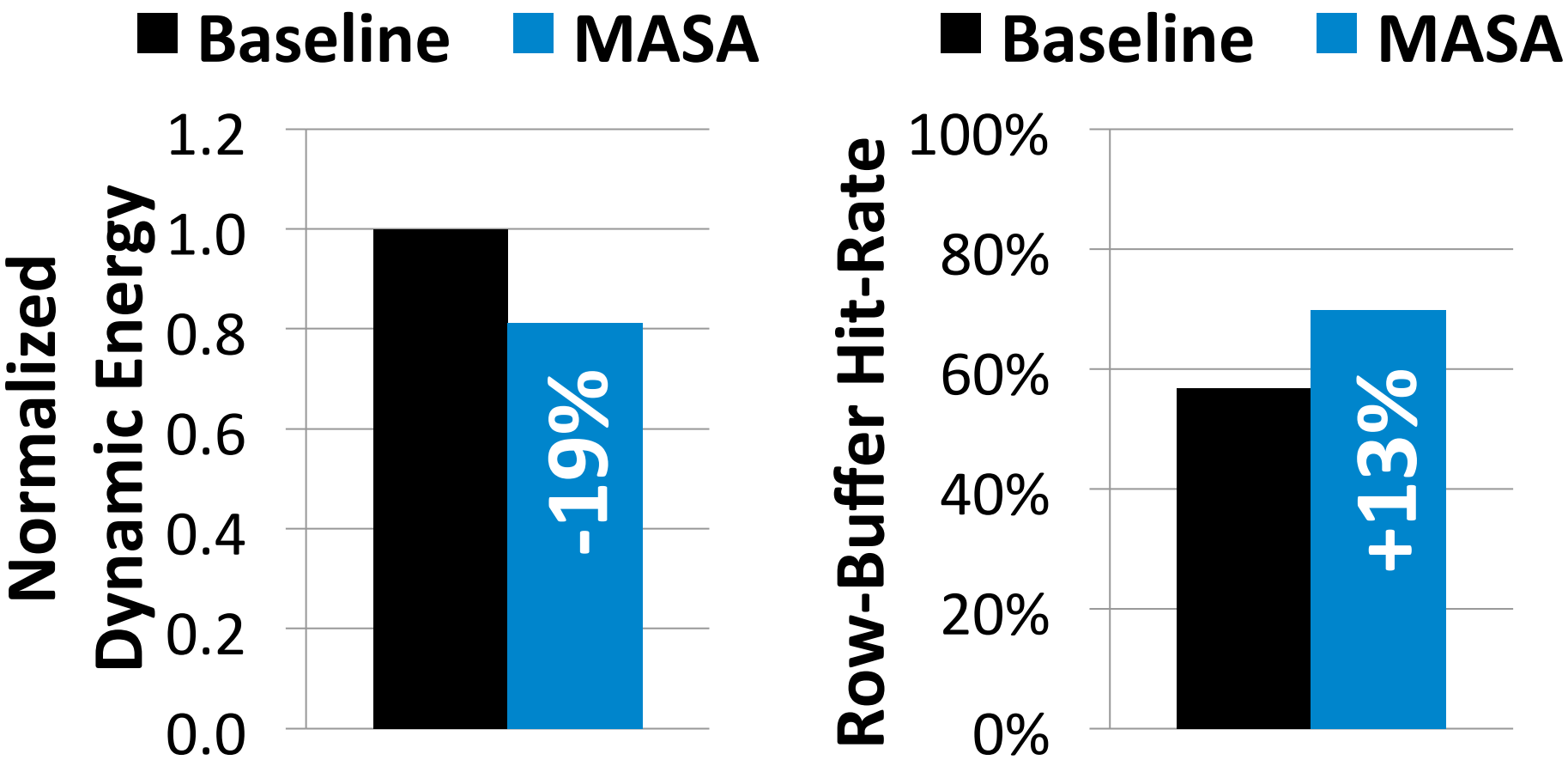
SALP: Single-Core Results



DRAM Die Area	< 0.15%	0.15%	36.3%

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

Subarray-Level Parallelism: Results



***MASA** increases energy-efficiency*

Agenda

- Major Trends Affecting Main Memory
- DRAM Scaling Problem and Solution Directions
- Three New Techniques for DRAM
  - RAIDR: Reducing Refresh Impact
  - TL-DRAM: Reducing DRAM Latency
  - SALP: Reducing Bank Conflict Impact
- Ongoing Research
- Summary

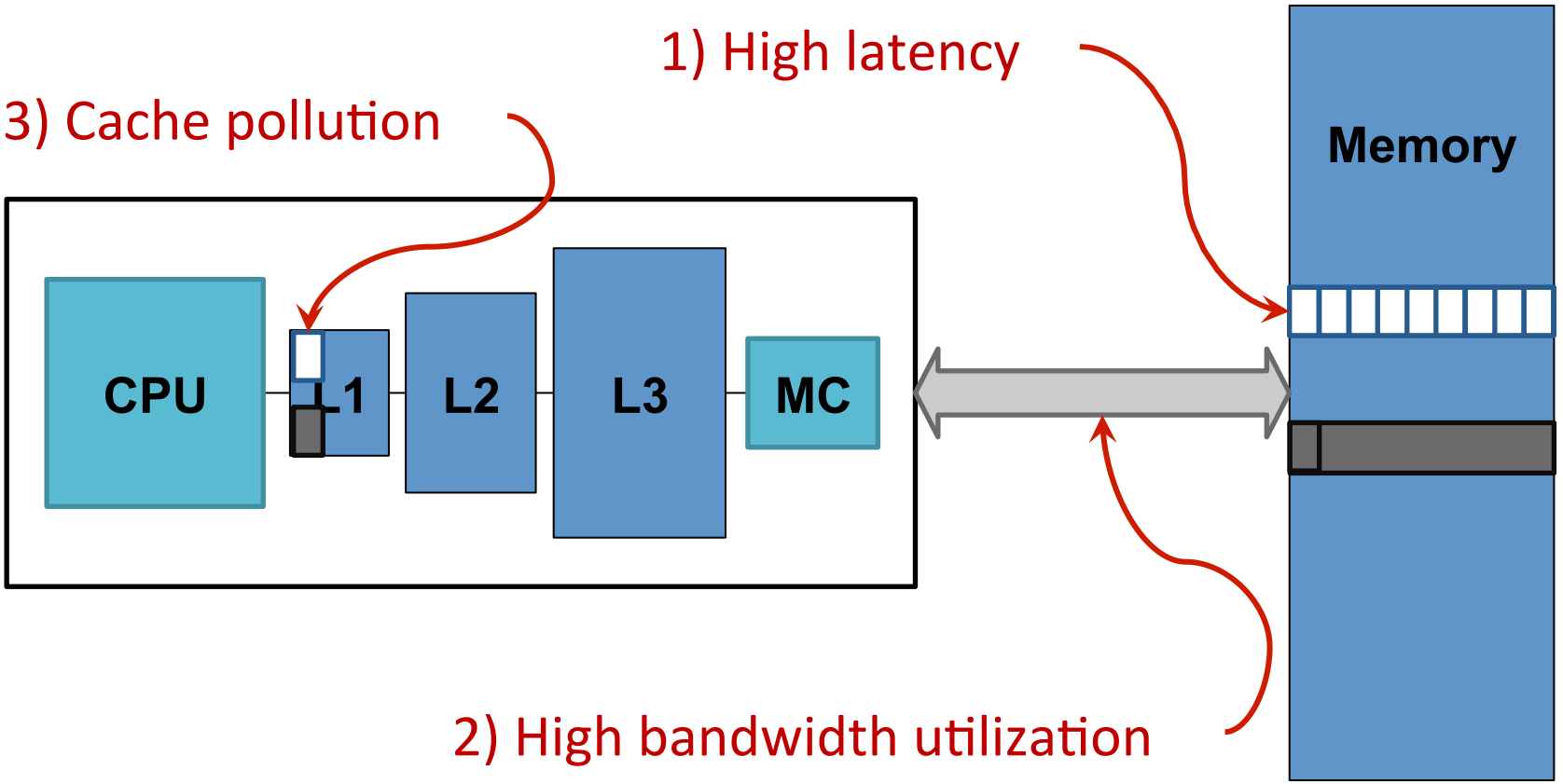
Sampling of Ongoing Research

- 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)
- Fast bulk data copy and initialization: RowClone
- Refresh/demand parallelization

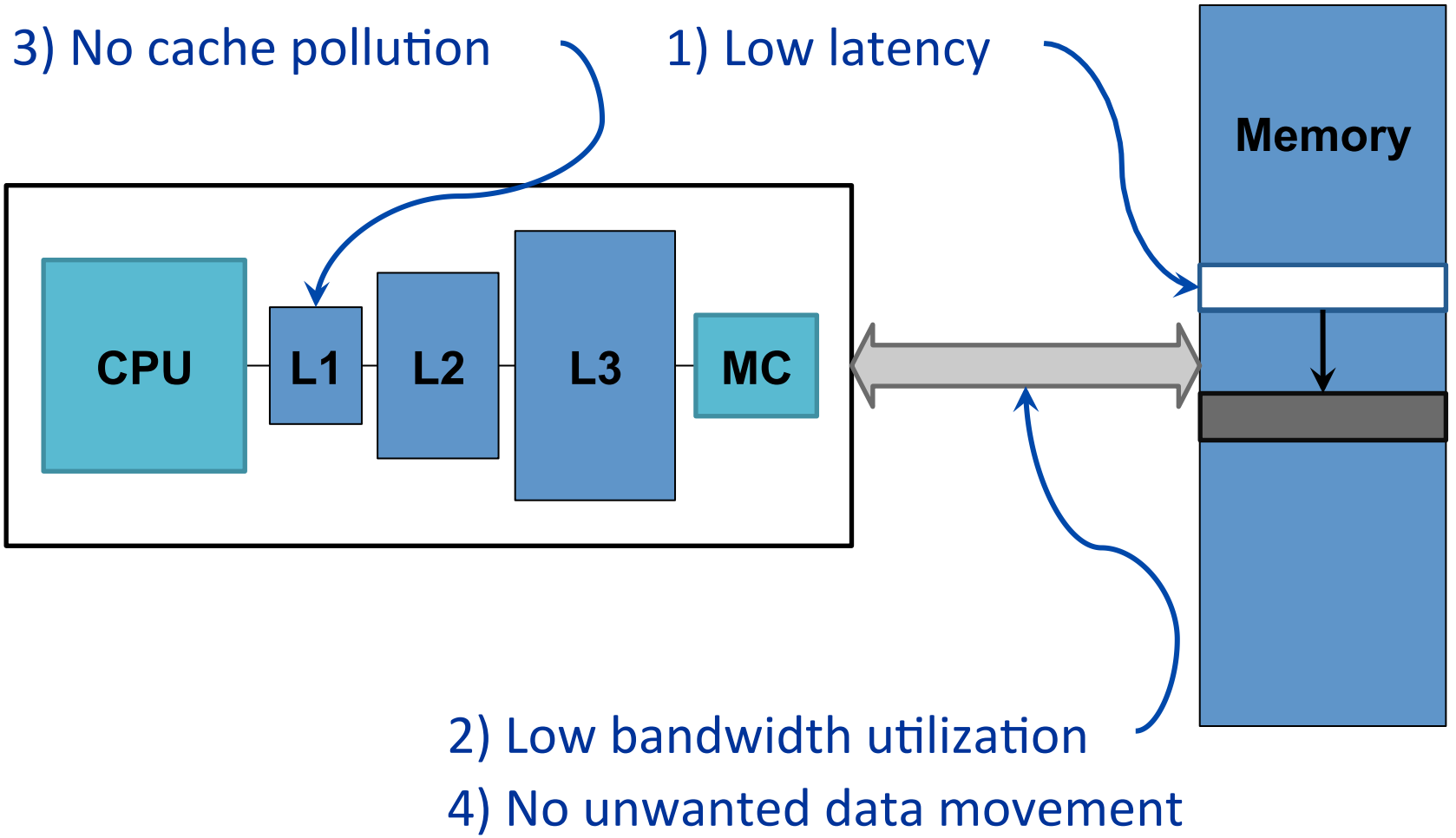
RowClone: Fast Bulk Data Copy and Initialization

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

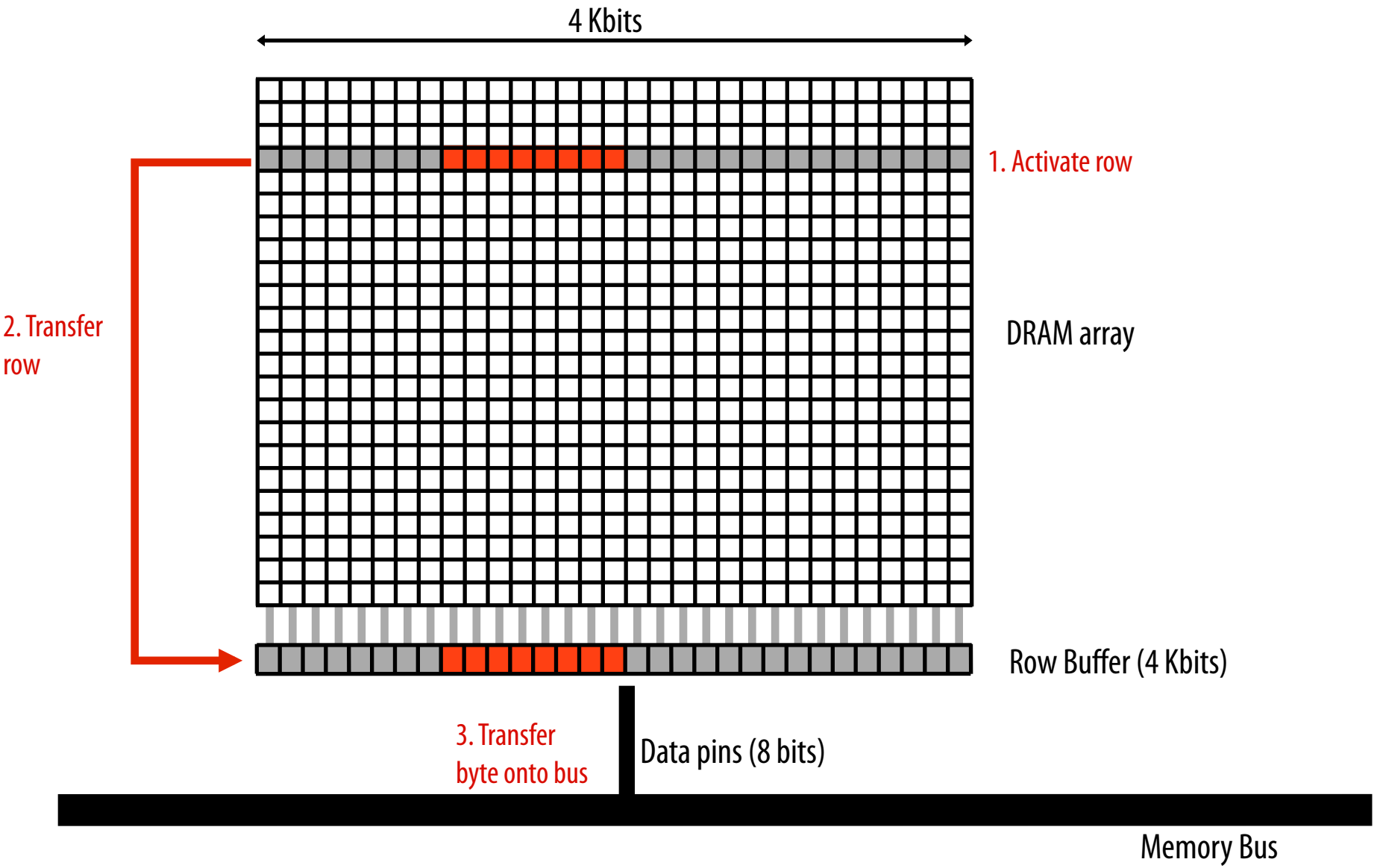


Future: RowClone (In-Memory Copy)

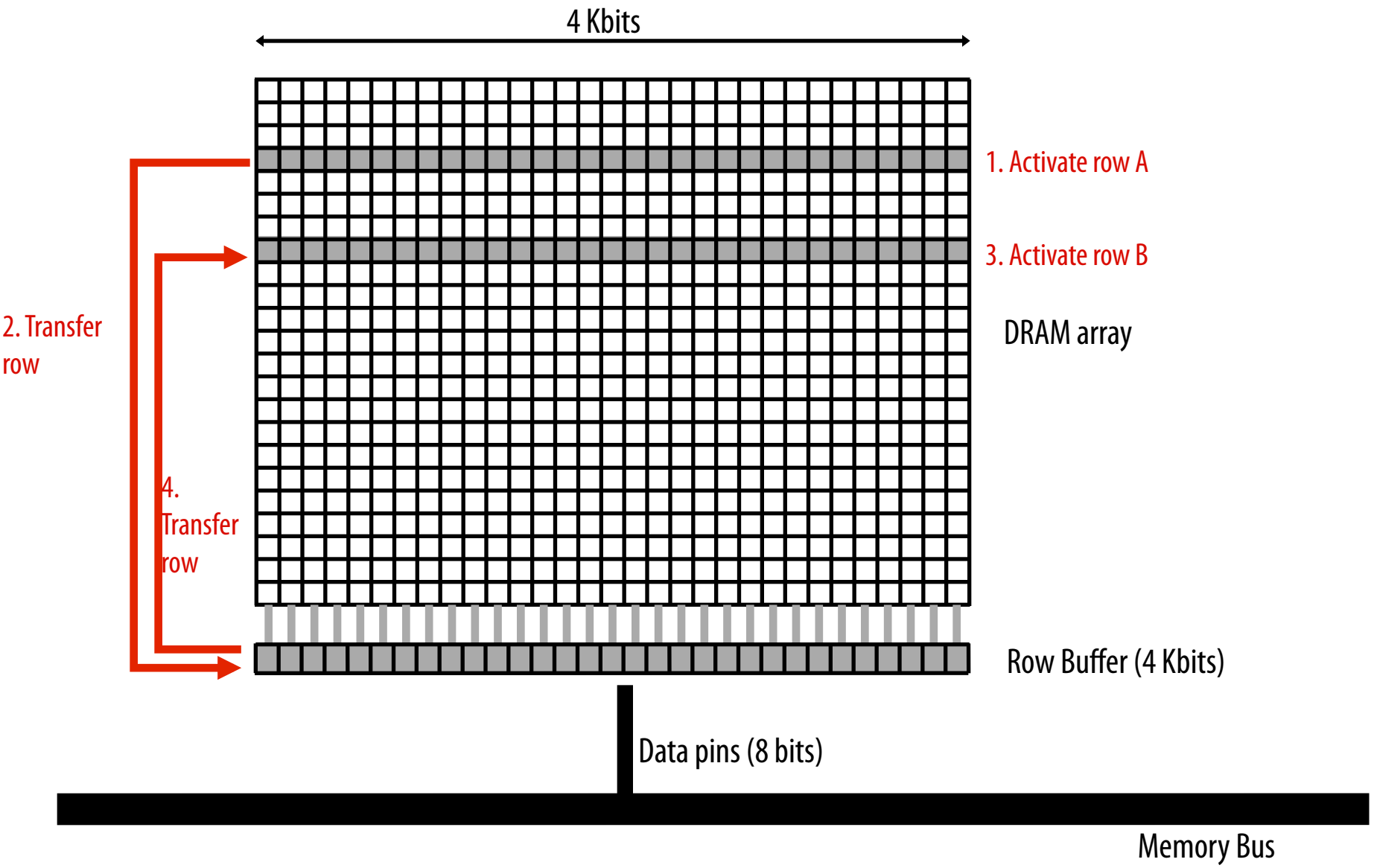


Seshadri et al., "RowClone: Fast and Efficient In-DRAM Copy and Initialization of Bulk Data," CMU Tech Report 2013.

DRAM operation (load one byte)



RowClone: in-DRAM Row Copy (and Initialization)

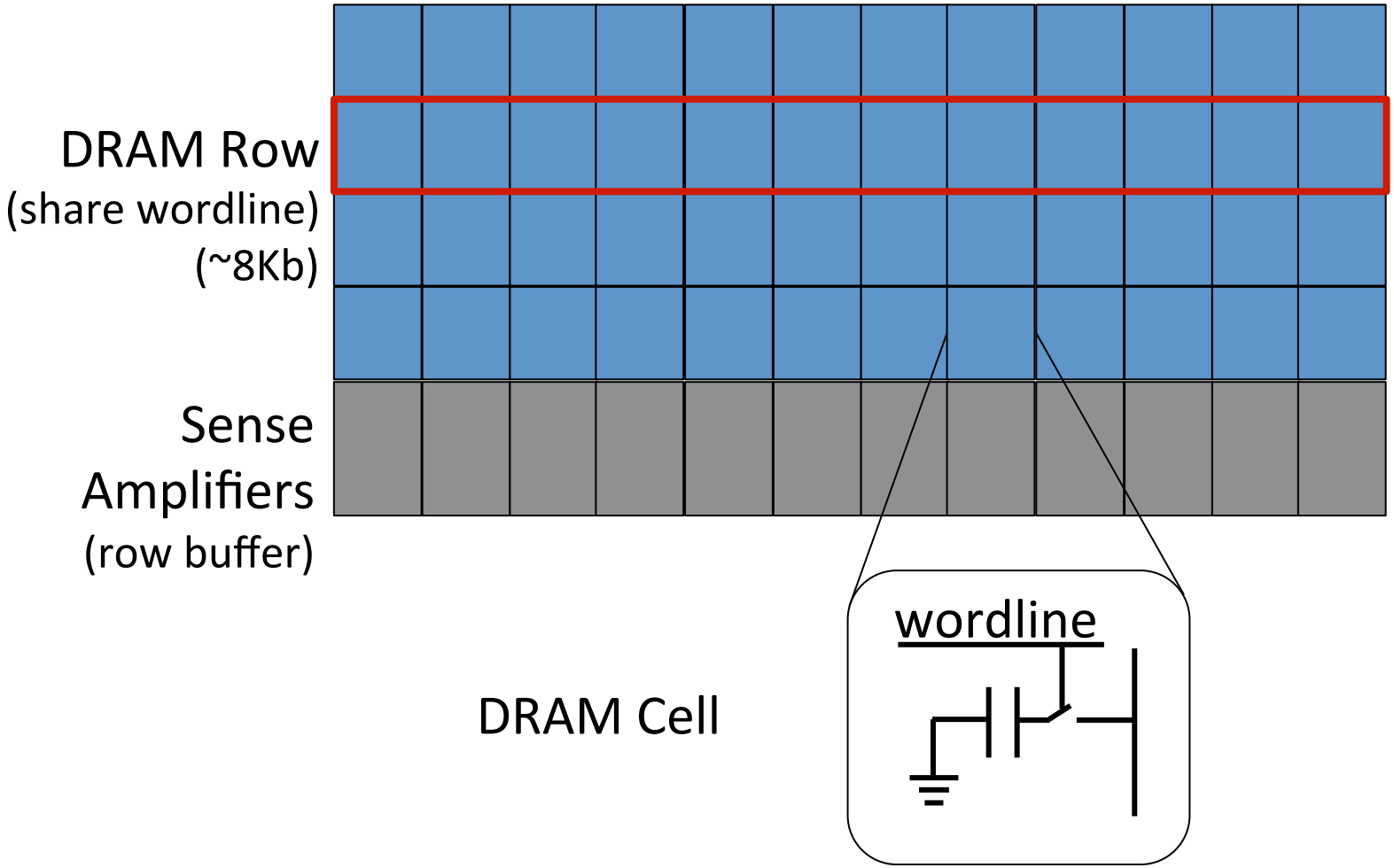


Our Approach: Key Idea

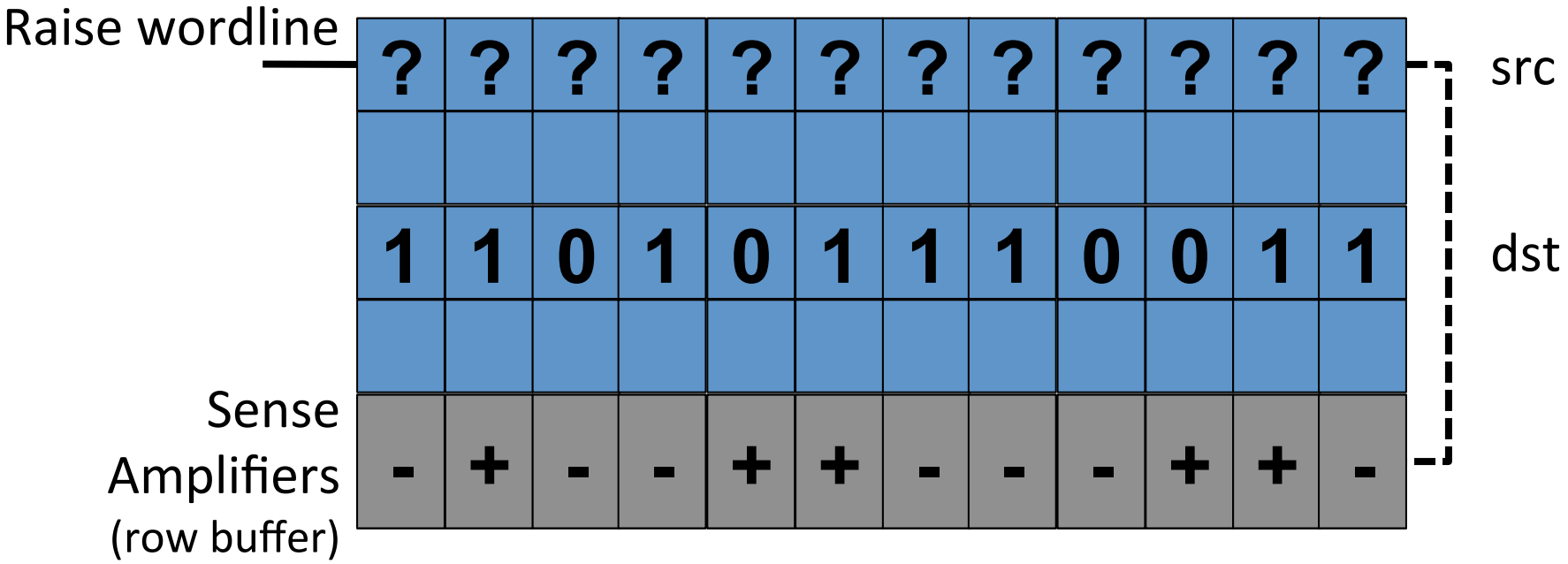
- DRAM banks contain
  1. Mutiple rows of DRAM cells – row = 8KB
  2. A row buffer shared by the DRAM rows

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

DRAM Subarray Microarchitecture

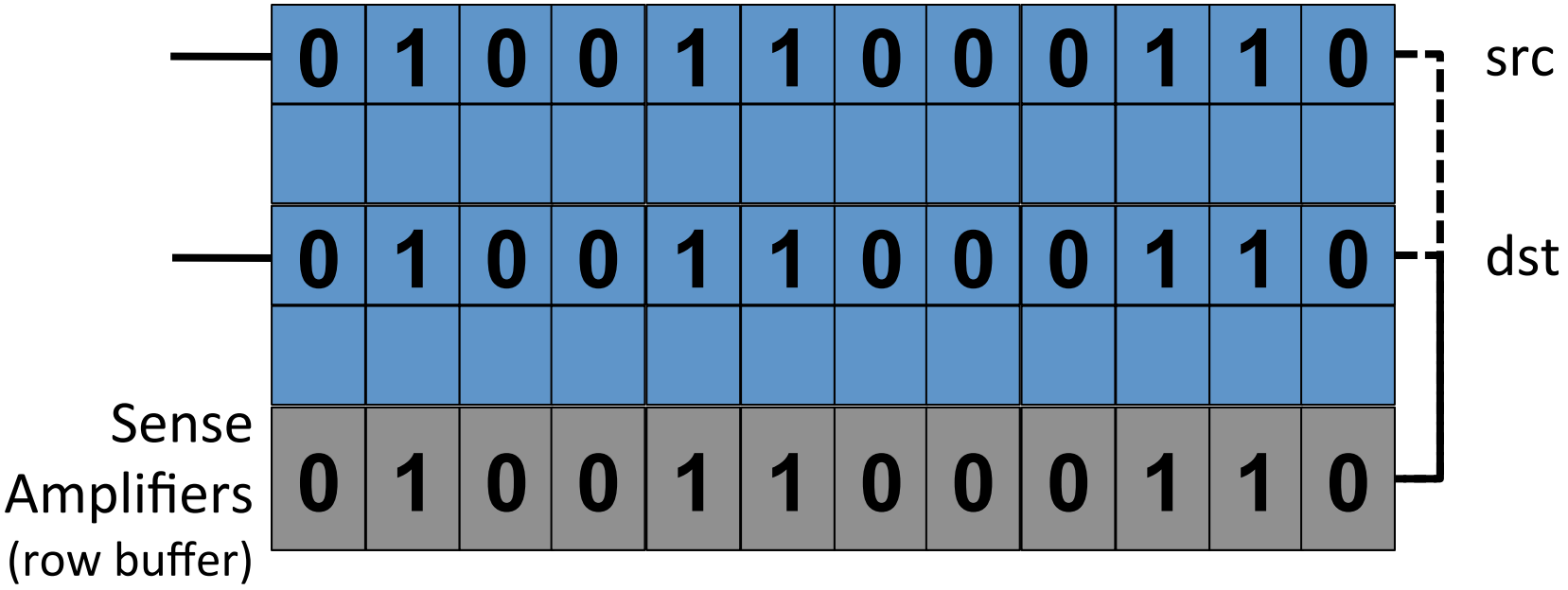


DRAM Operation



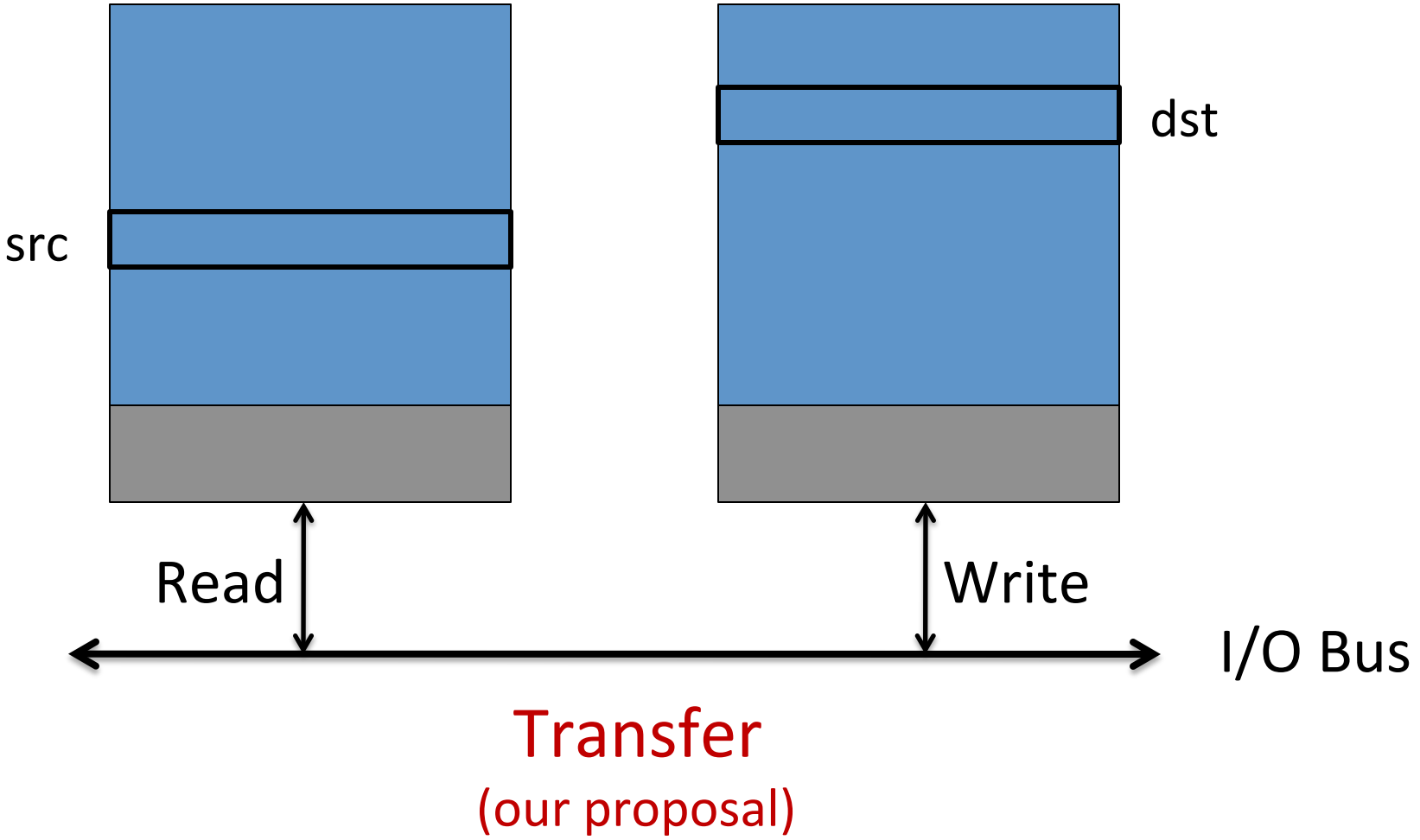
Activate (src) ⟶ Precharge

RowClone: Intra-subarray Copy

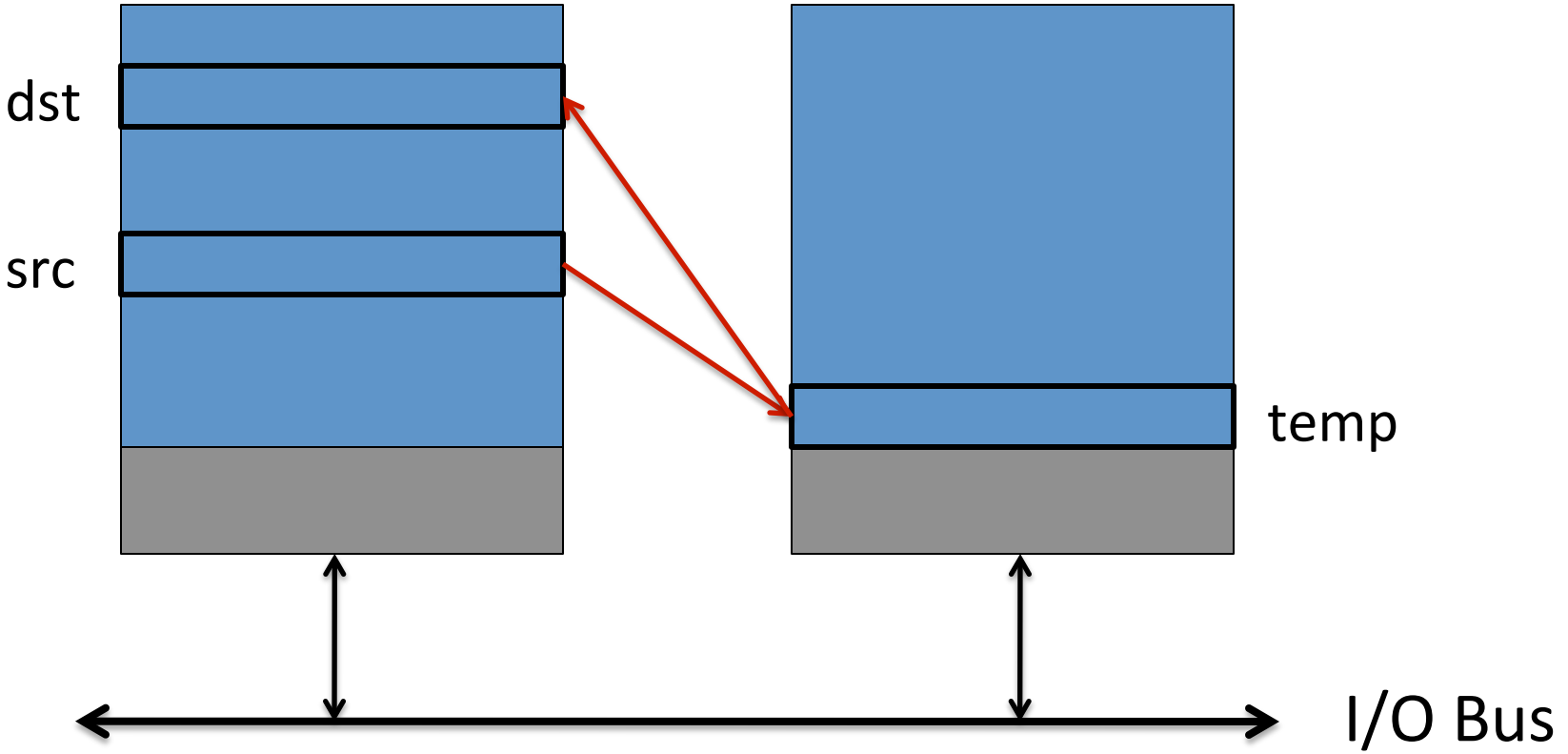


Activate (src) → Deactivate (our proposal) → Activate (dst)

RowClone: Inter-bank Copy

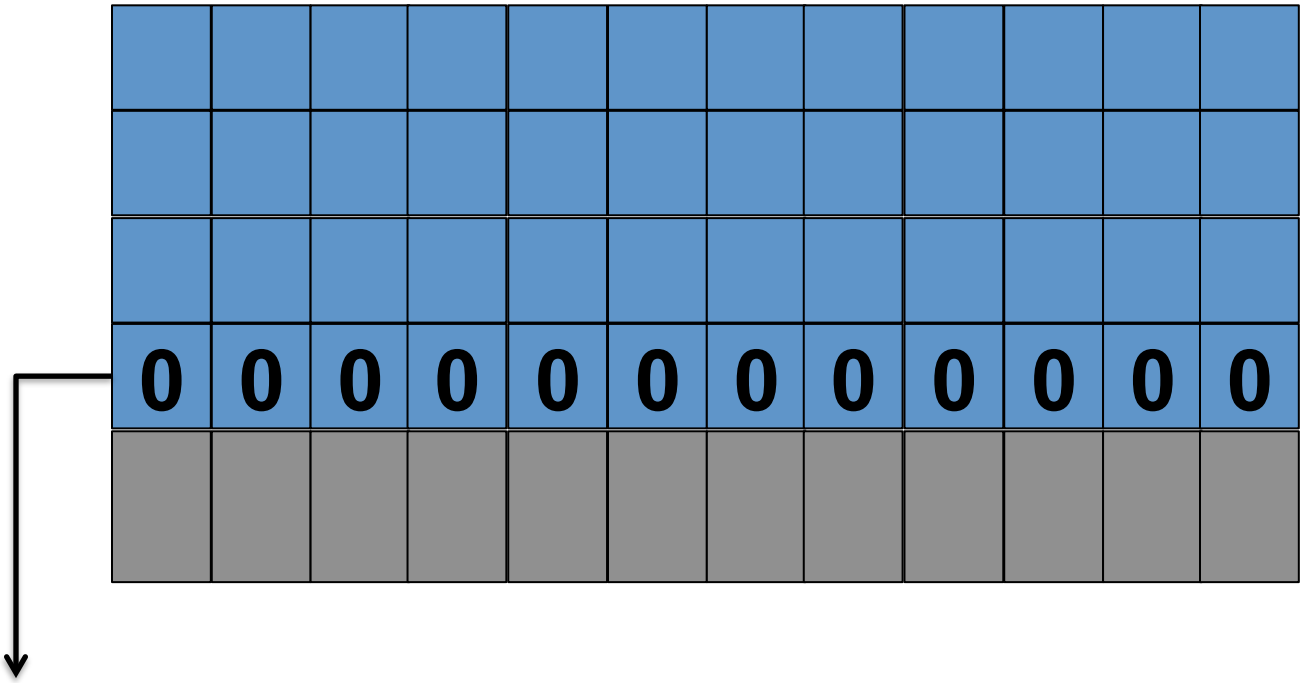


RowClone: Inter-subarray Copy



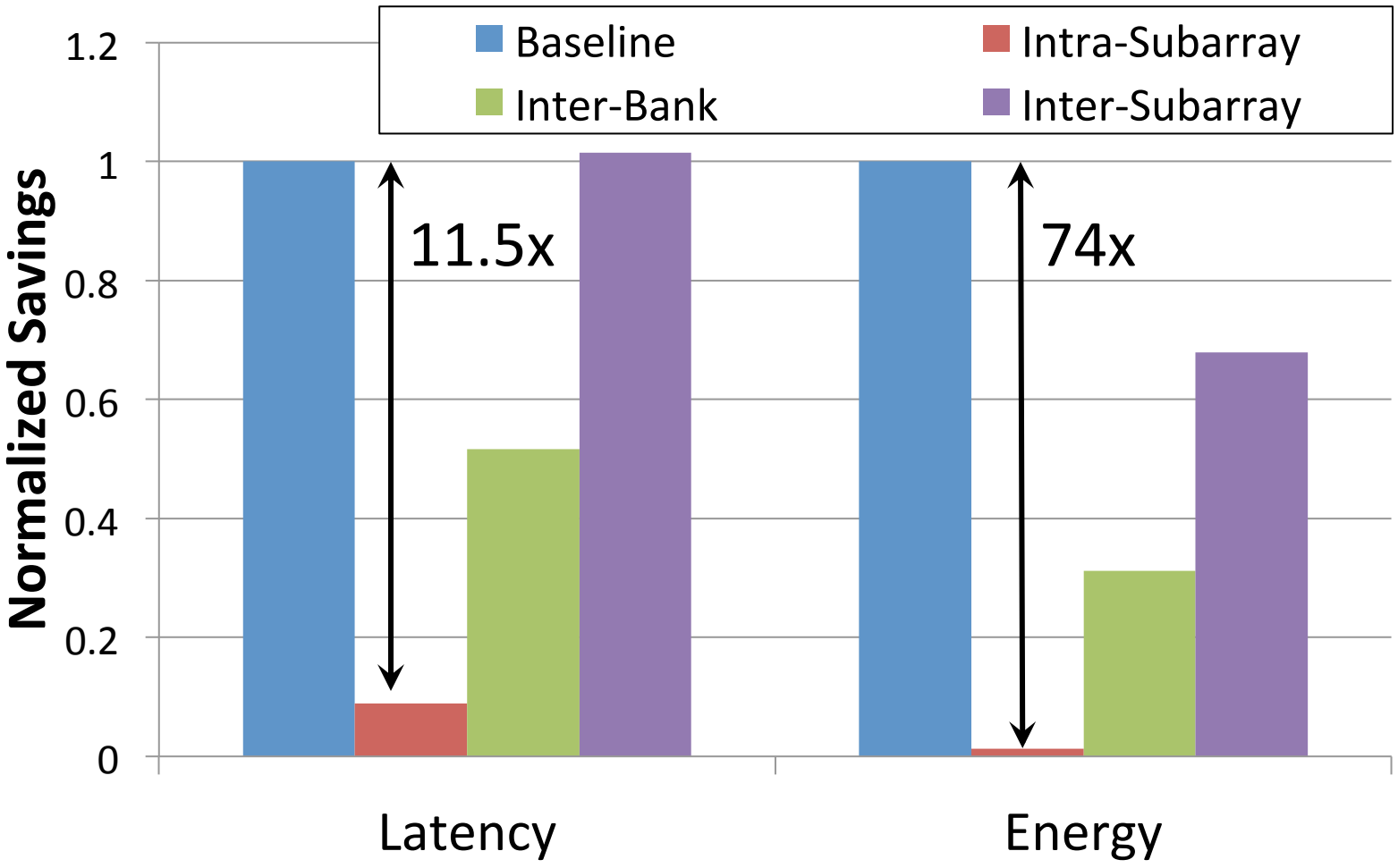
1. Transfer (src to temp)
2. Transfer (temp to dst)

Fast Row Initialization



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

RowClone: Latency and Energy Savings



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

Agenda

- Major Trends Affecting Main Memory
- DRAM Scaling Problem and Solution Directions
- Three New Techniques for DRAM
  - RAIDR: Reducing Refresh Impact
  - TL-DRAM: Reducing DRAM Latency
  - SALP: Reducing Bank Conflict Impact
- Ongoing Research
- Summary

Summary

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

Thank you.

A Fresh Look at DRAM Architecture: New Techniques to Improve DRAM Latency, Parallelism, and Energy Efficiency

Onur Mutlu
onur@cmu.edu
July 4, 2013
INRIA

Carnegie Mellon

Additional Material

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.

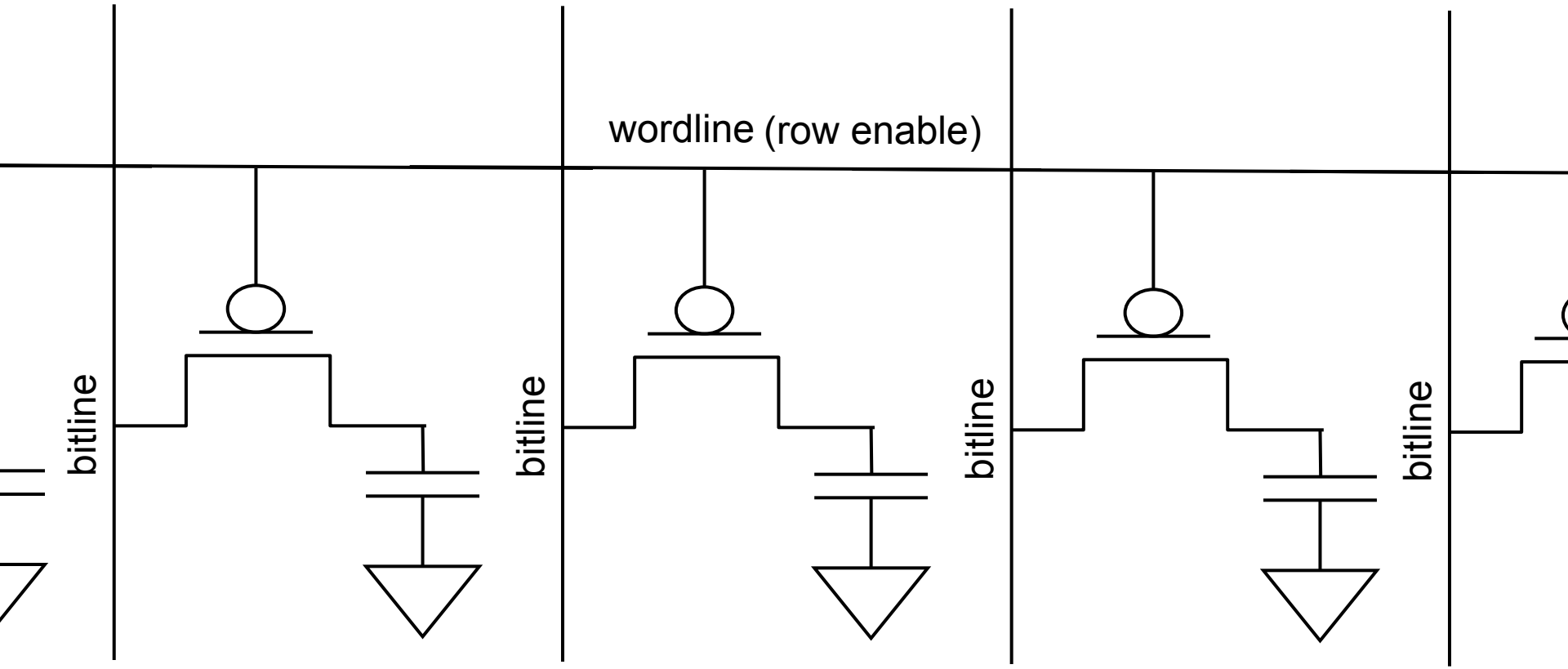
Summary (II)

- 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



- 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

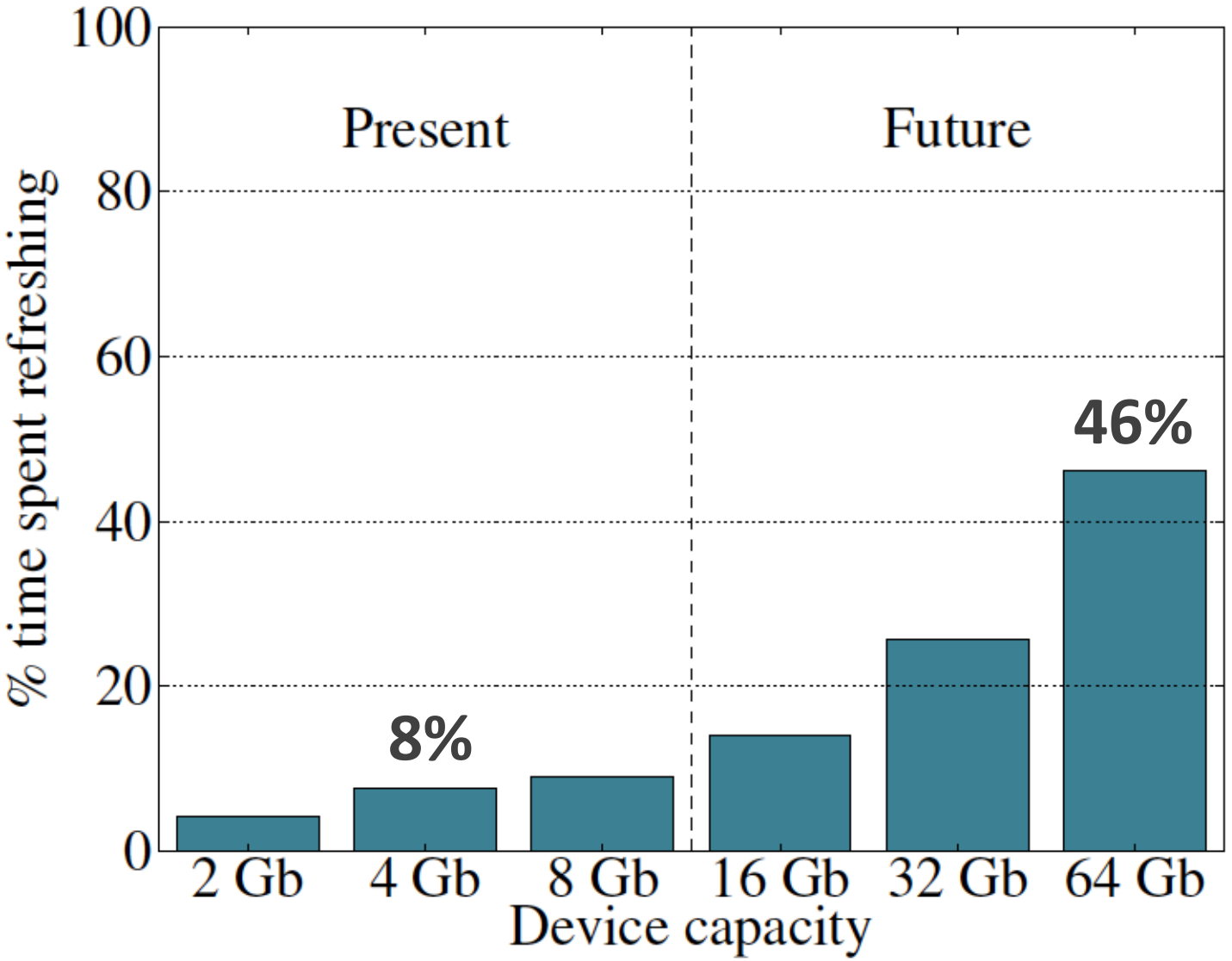
  -- Energy consumption: Each refresh consumes energy

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

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

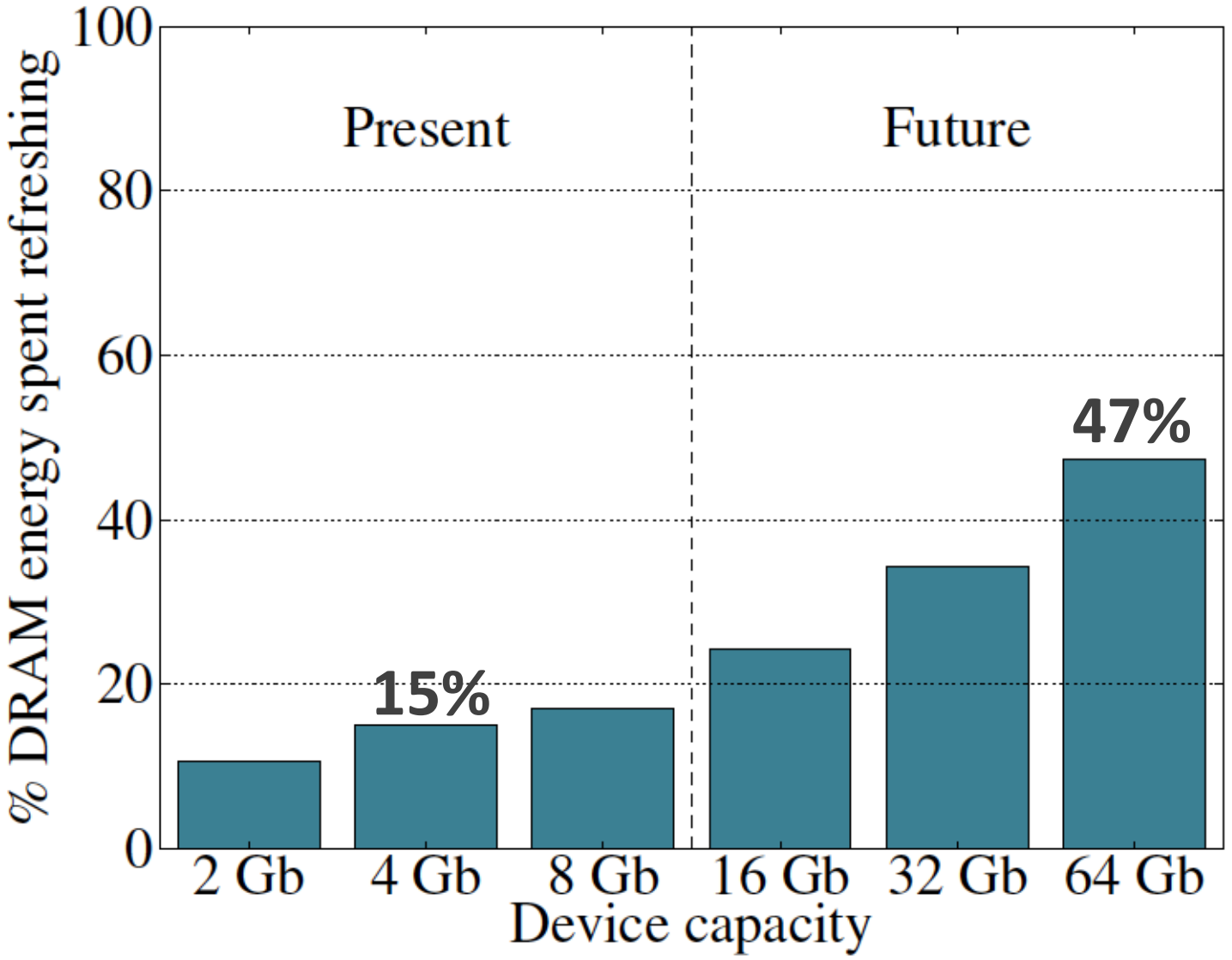
  -- Refresh rate limits DRAM capacity scaling

Refresh Overhead: Performance



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

Refresh Overhead: Energy

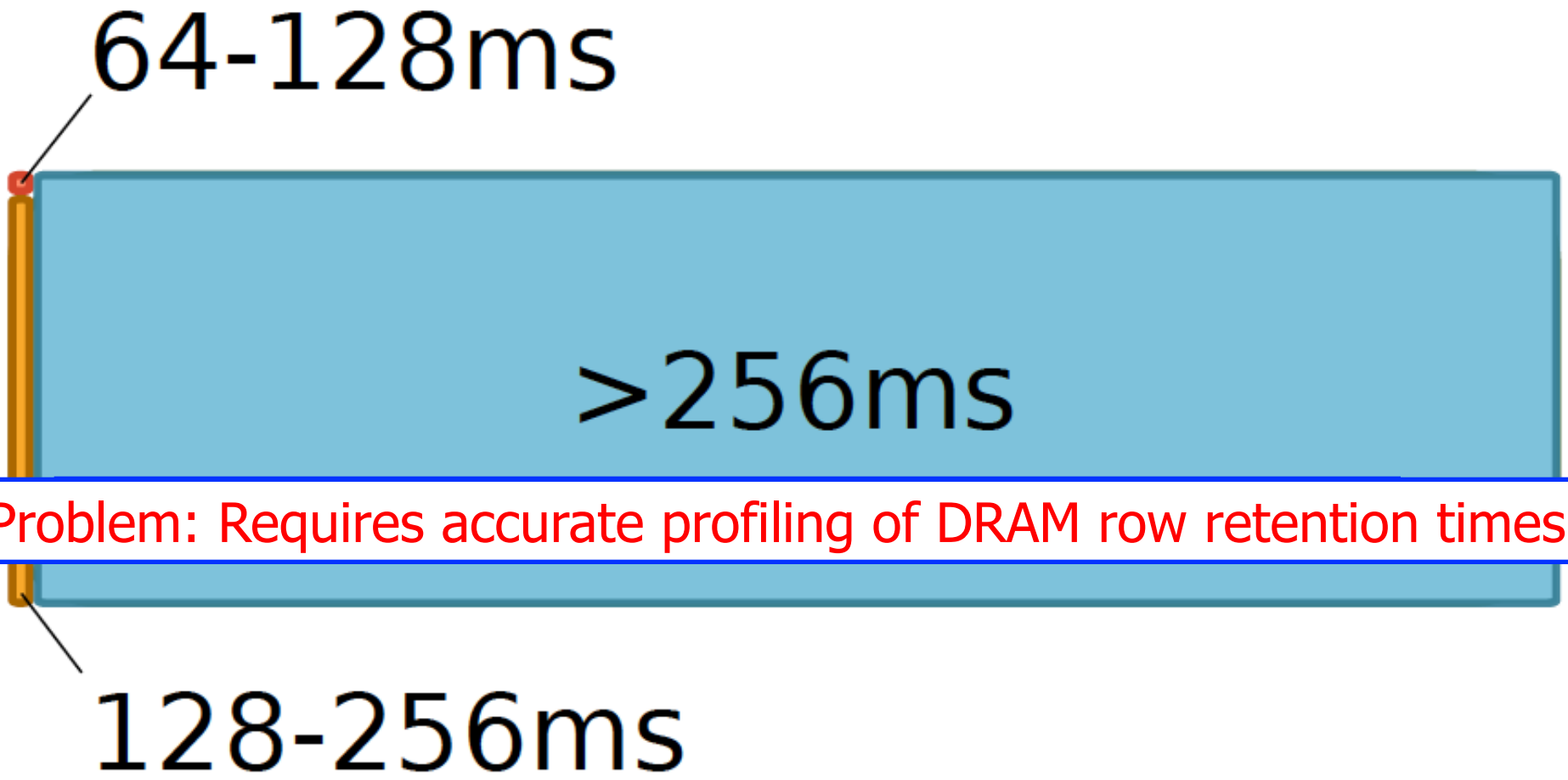


Liu et al., "RAIDR: Retention-Aware Intelligent DRAM Refresh," ISCA 2012.

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

An Example: RAIDR [Liu+, ISCA 2012]



Problem: Requires accurate profiling of DRAM row retention times

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

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

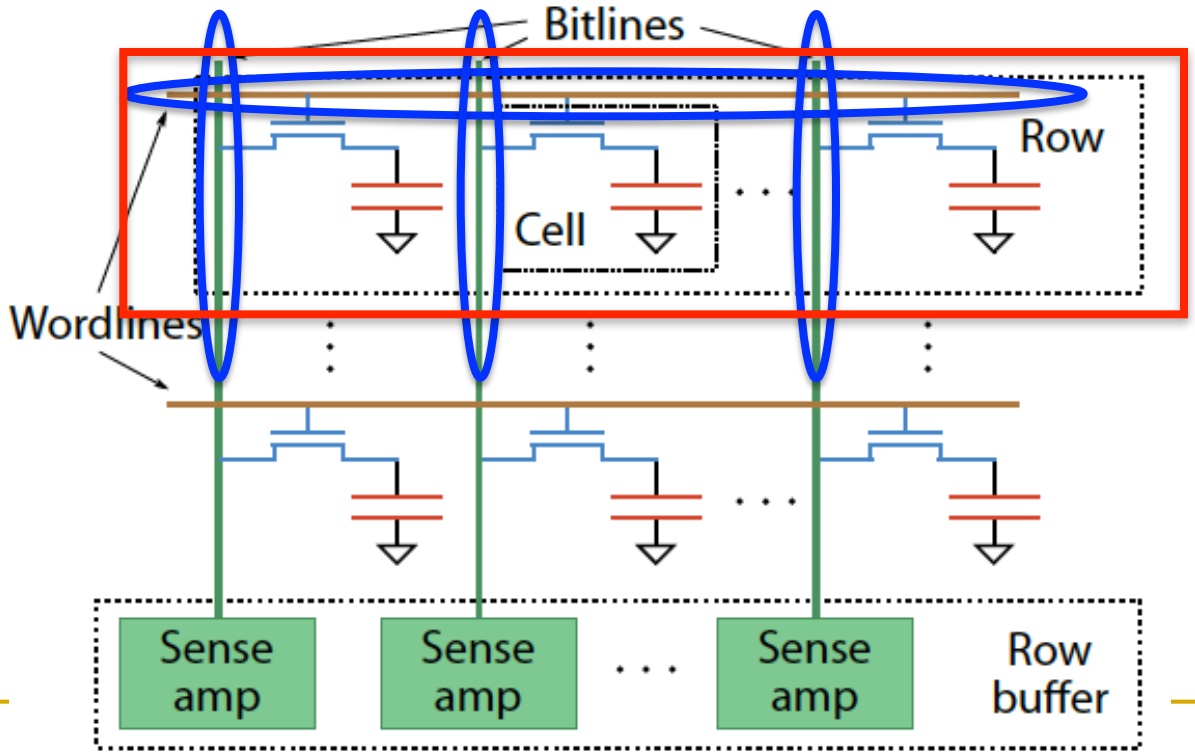
Two Challenges to Retention Time Profiling

- Data Pattern Dependence (DPD) of retention time

- Variable Retention Time (VRT) phenomenon

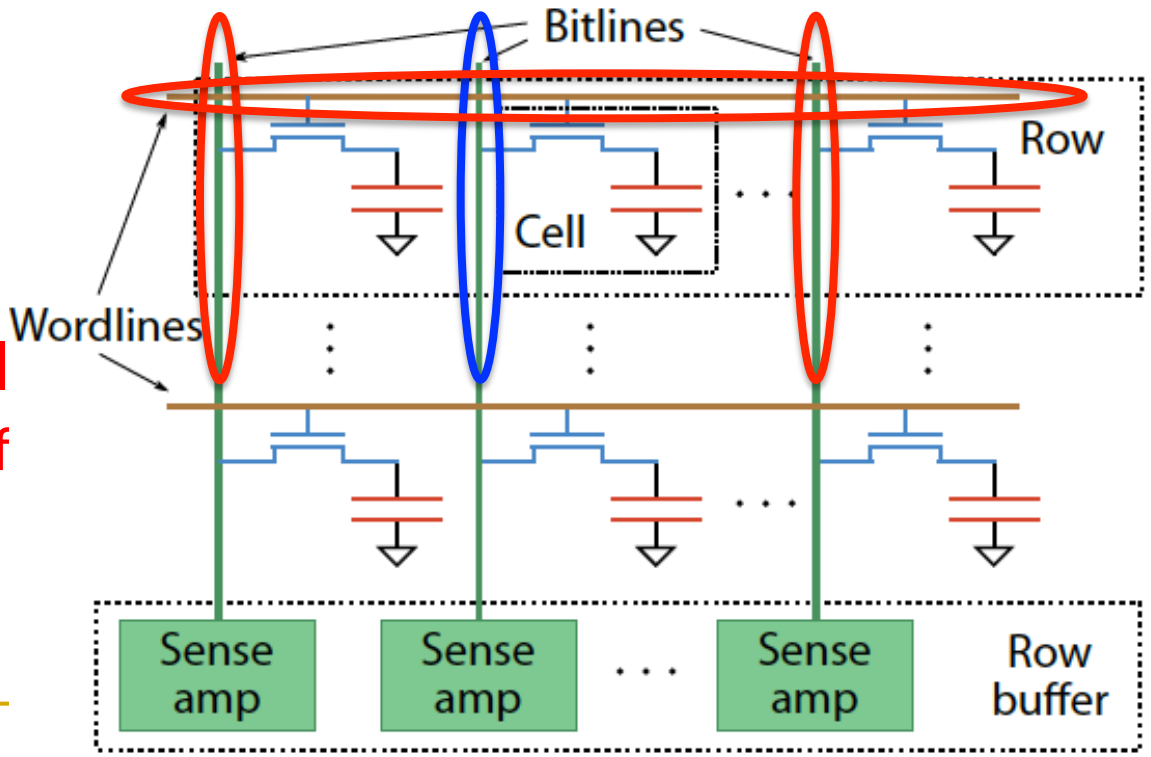
Two Challenges to Retention Time Profiling

- 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



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



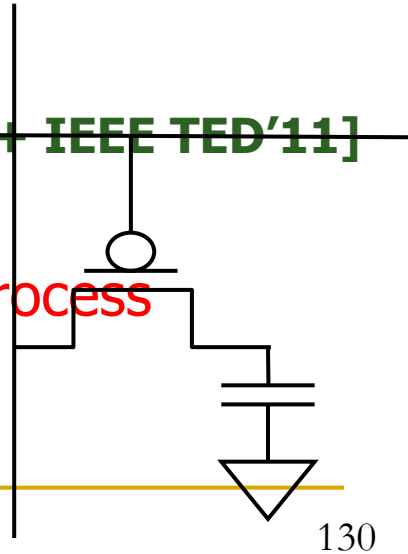
- Retention
  nearby cel
  → need to f

tored in

etention time

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

DRAM Testing Platform and Method

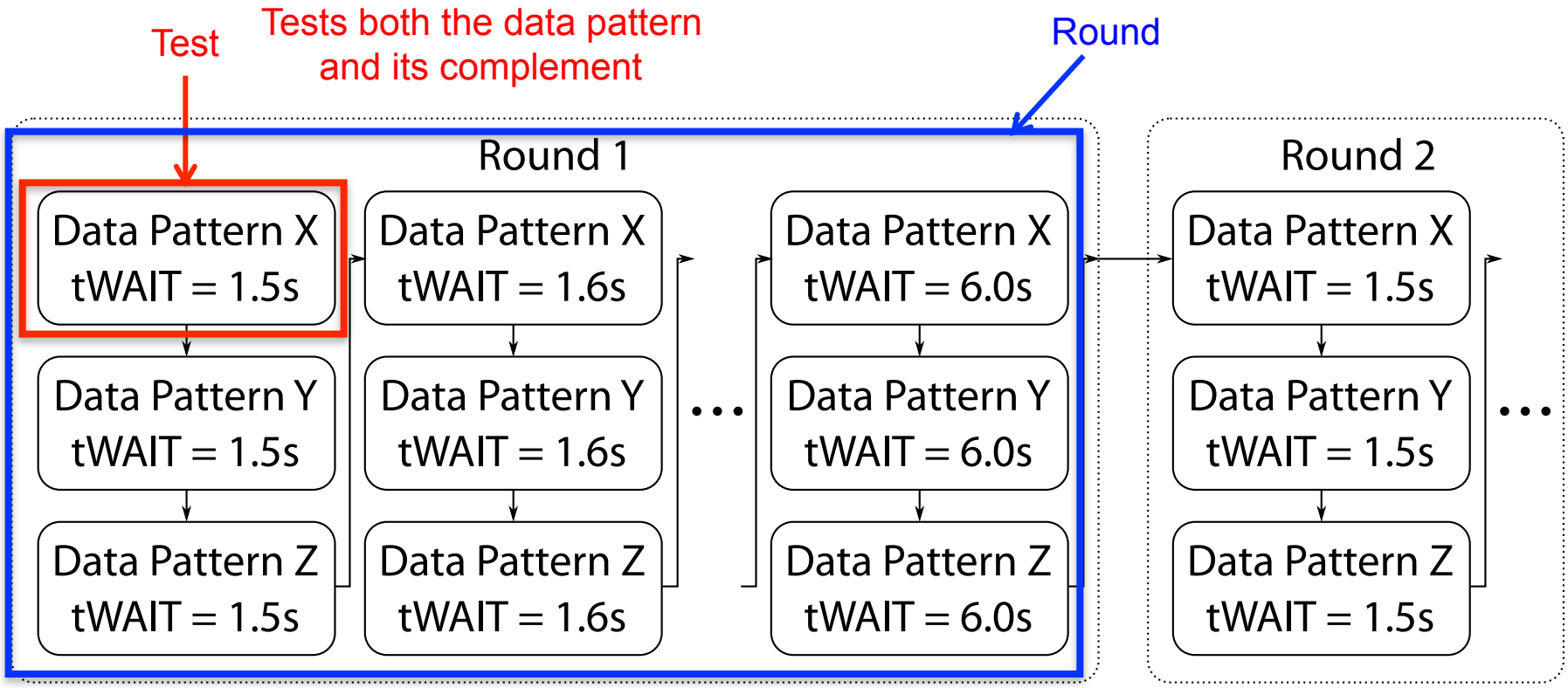
- Test platform: Developed a DDR3 DRAM testing platform using the Xilinx ML605 FPGA development board
  - Temperature controlled

- 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



Experiment Parameters

- Most tests conducted at 45 degrees Celsius
- No cells observed to have a retention time less than 1.5 second at 45°C
- Tested *tWAIT* in increments of 128ms from 1.5 to 6.1 seconds

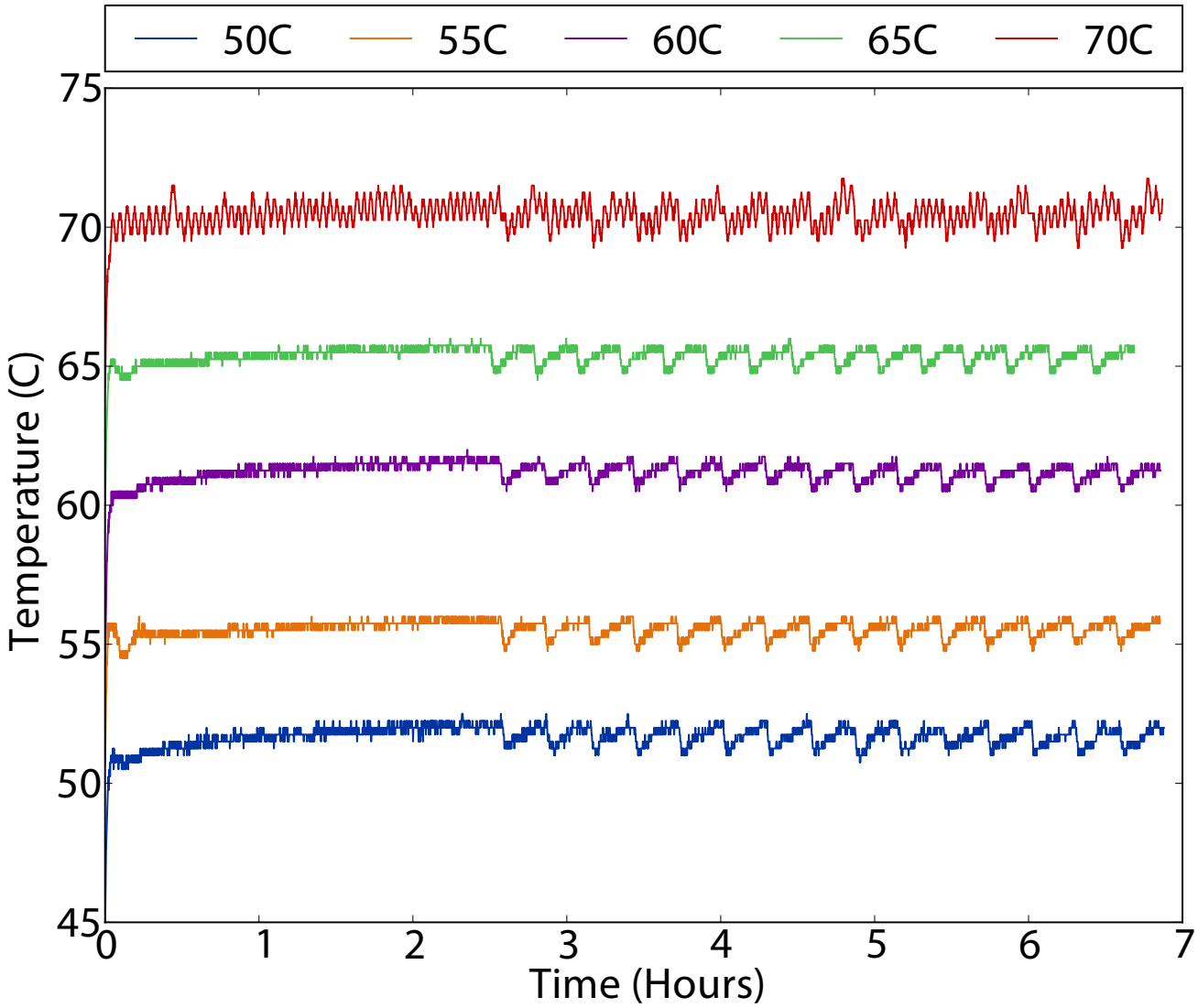
Tested Data Patterns

- 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

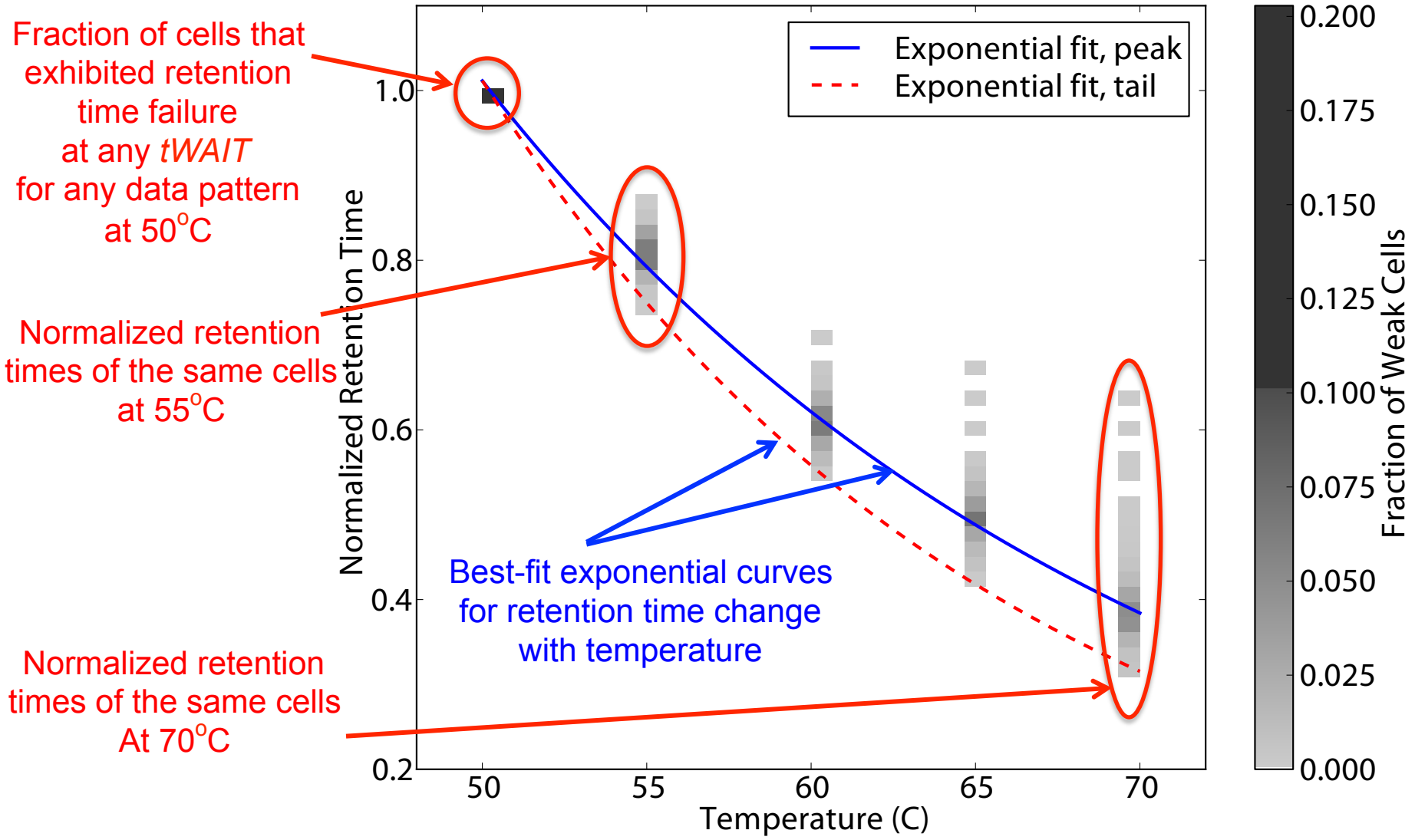
- DRAM Refresh: Background and Motivation
- Challenges and Our Goal
- DRAM Characterization Methodology
- Foundational Results
  - Temperature Dependence
  - Retention Time Distribution
- Data Pattern Dependence: Analysis and Implications
- Variable Retention Time: Analysis and Implications
- Conclusions

Temperature Stability

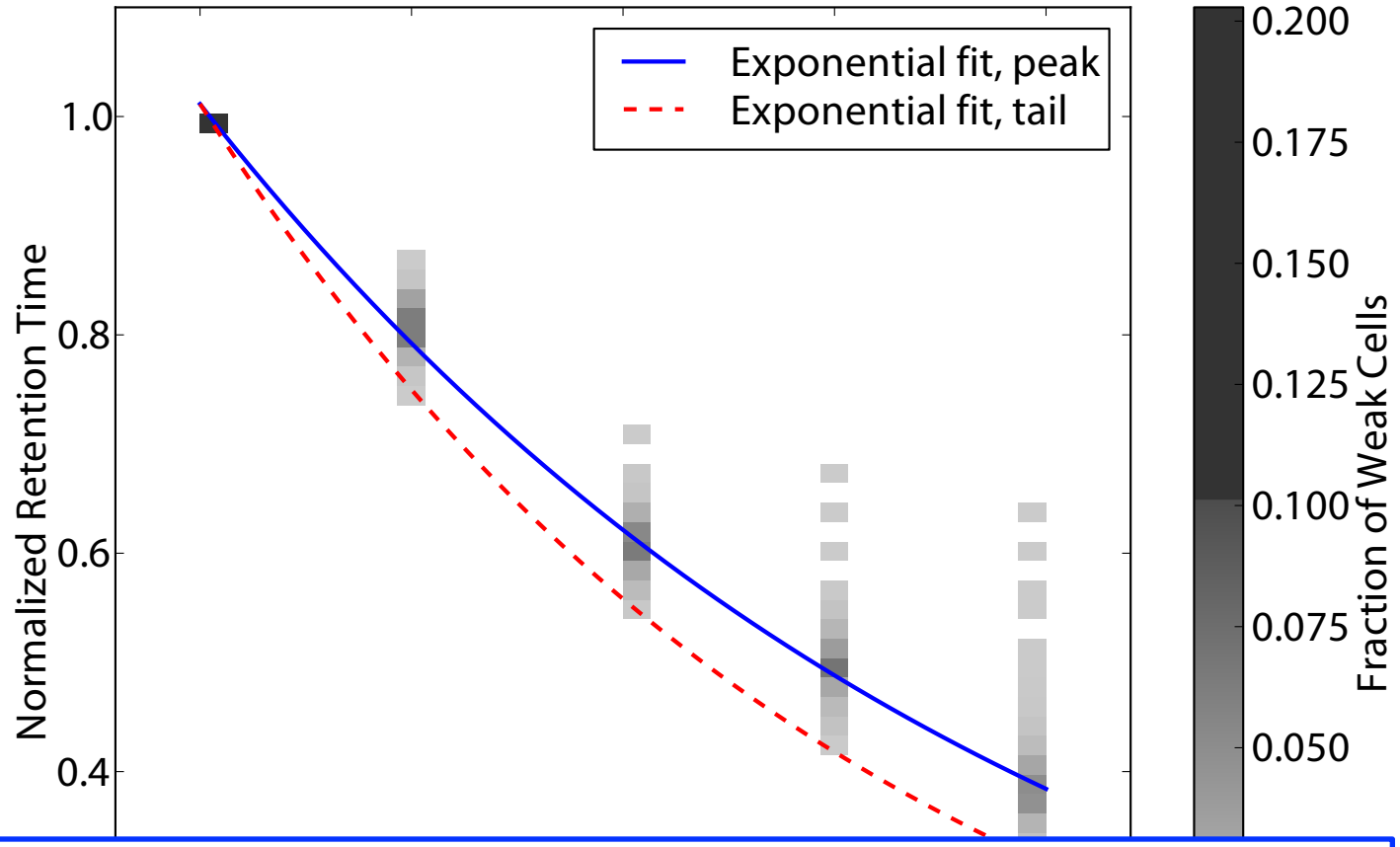


Tested chips at five different stable temperatures

Dependence of Retention Time on Temperature



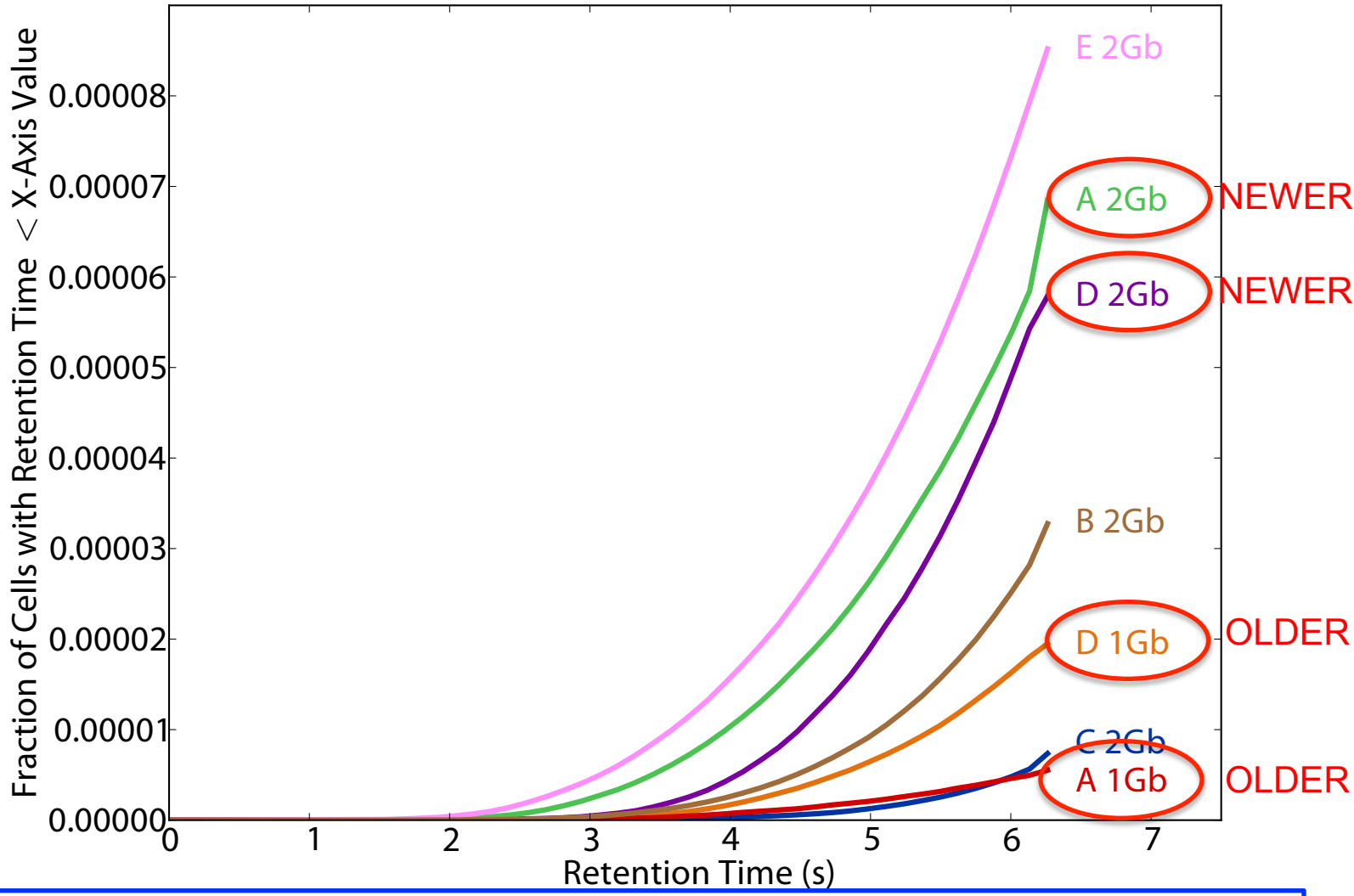
Dependence of Retention Time on Temperature



Relationship between retention time and temperature is consistently bounded (predictable) within a device

Every 10°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

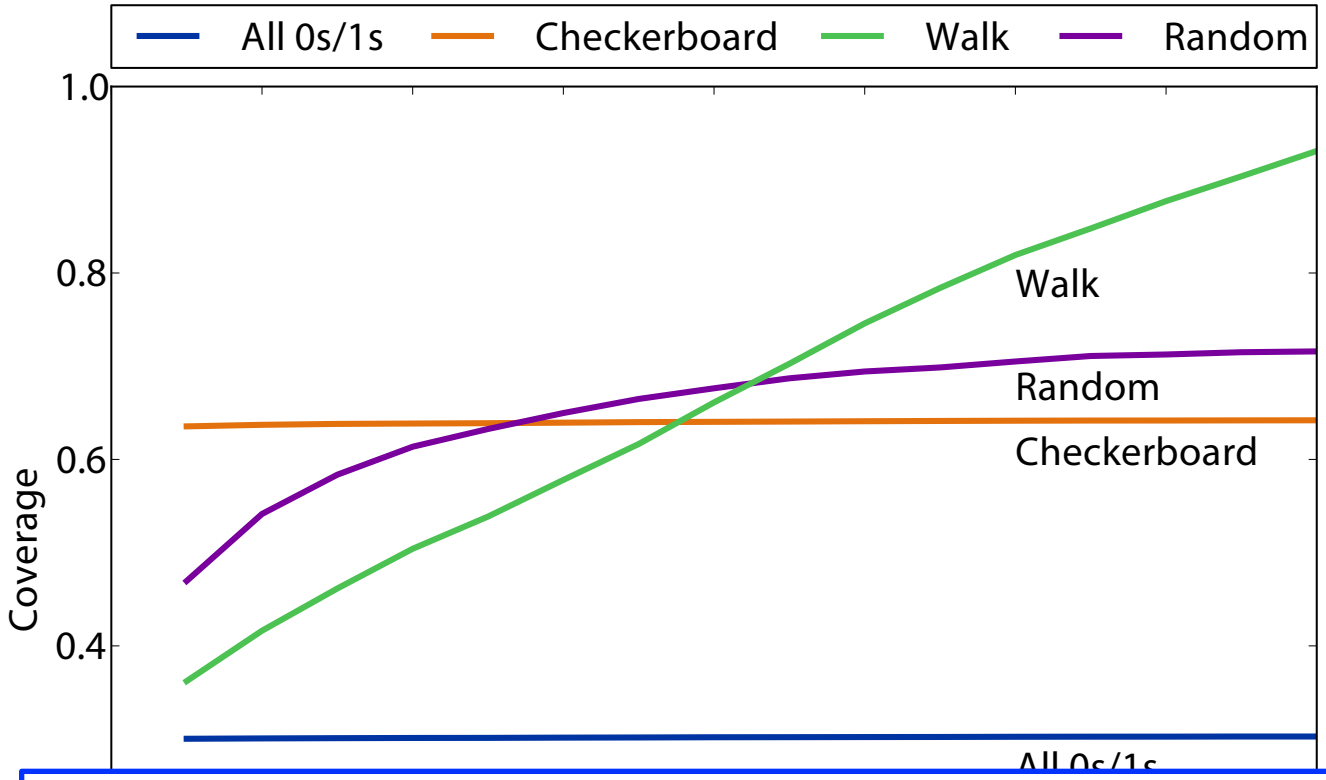
Fixed patterns

- 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

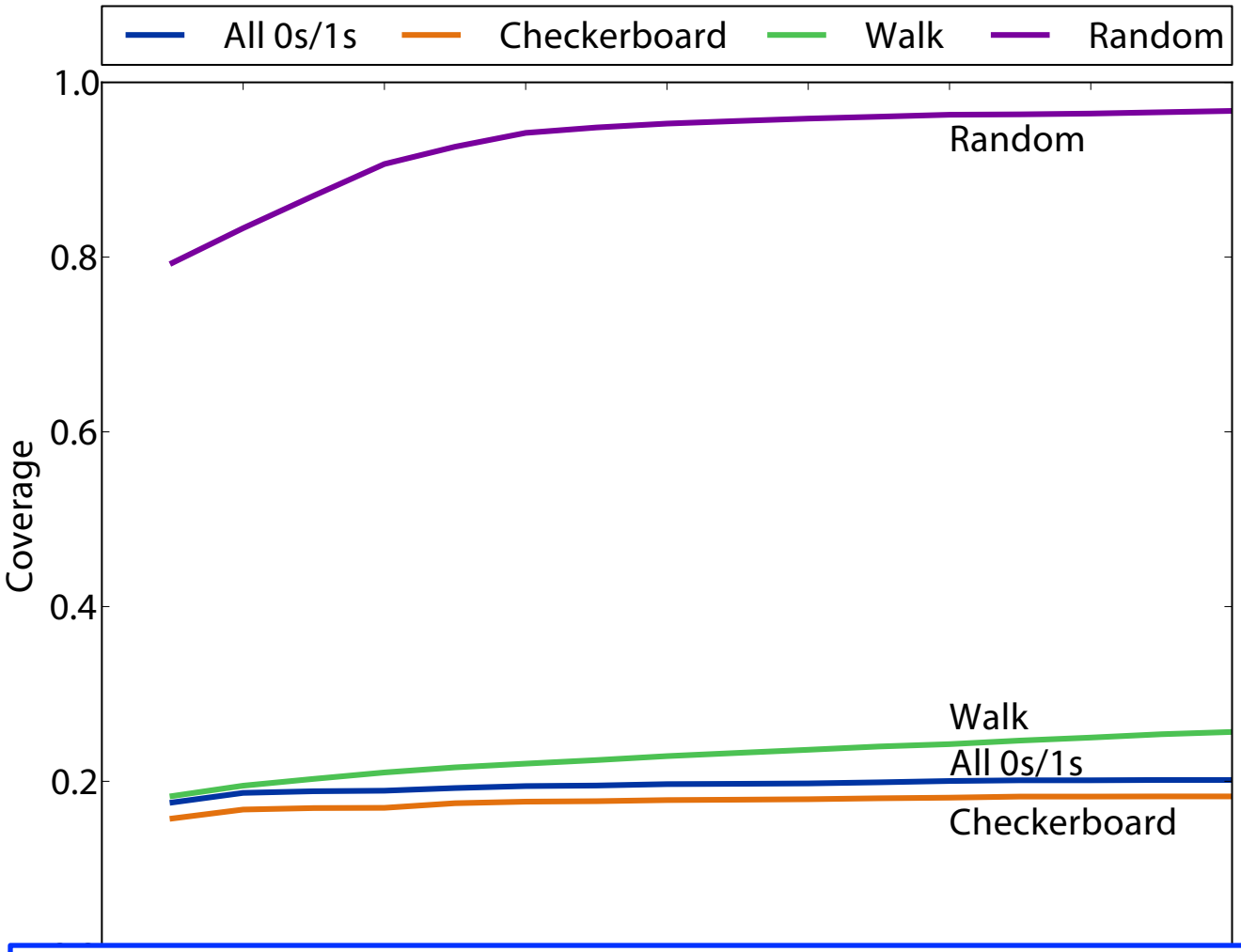
Different data patterns have widely different coverage: Data pattern dependence exists and is severe

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



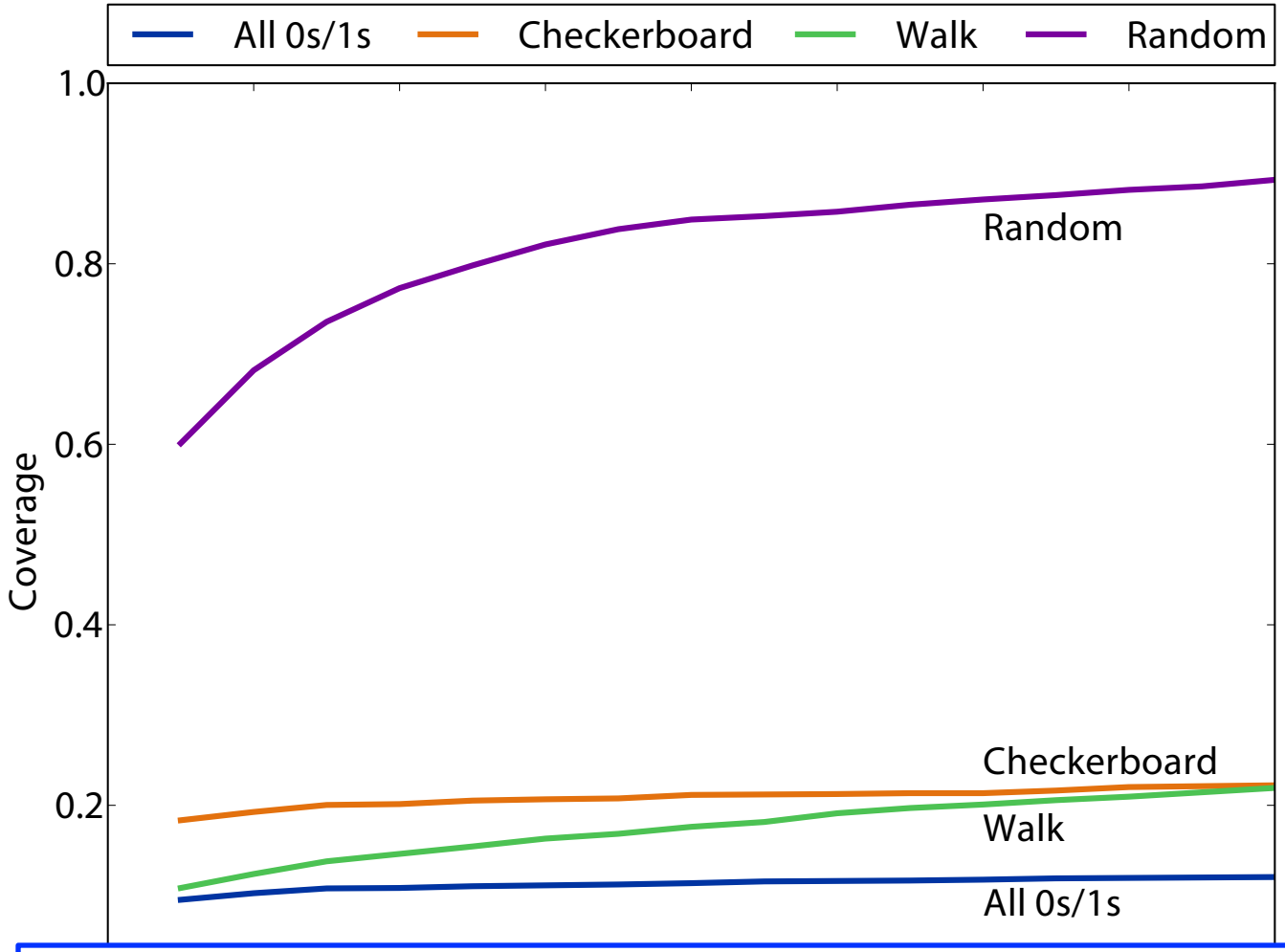
B 2Gb chip family

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)

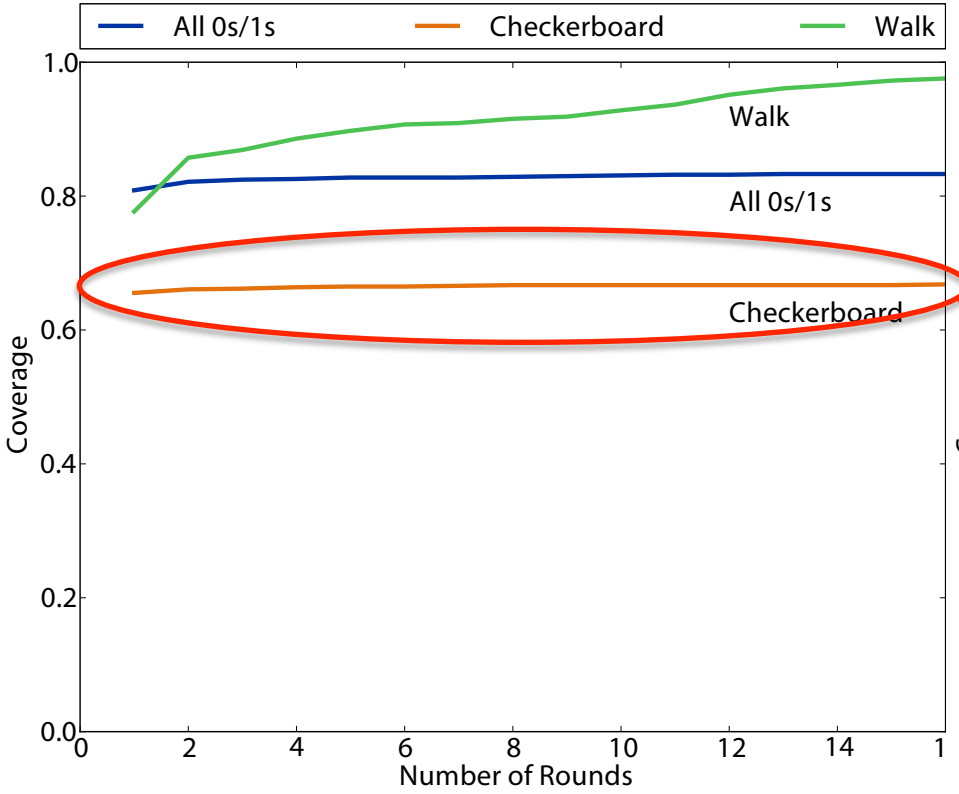
- A cell's retention time is heavily influenced by data pattern stored in other cells
  - Pattern affects the coupling noise, which affects cell leakage

- 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

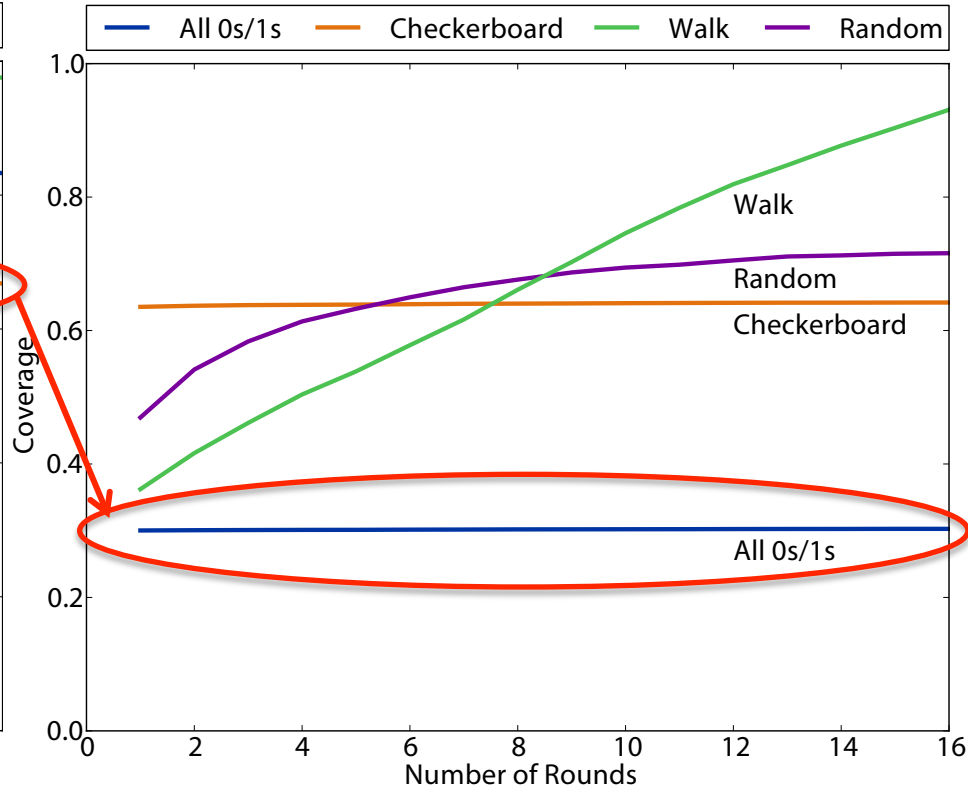
Data Pattern Dependence: Observations (II)

- 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

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

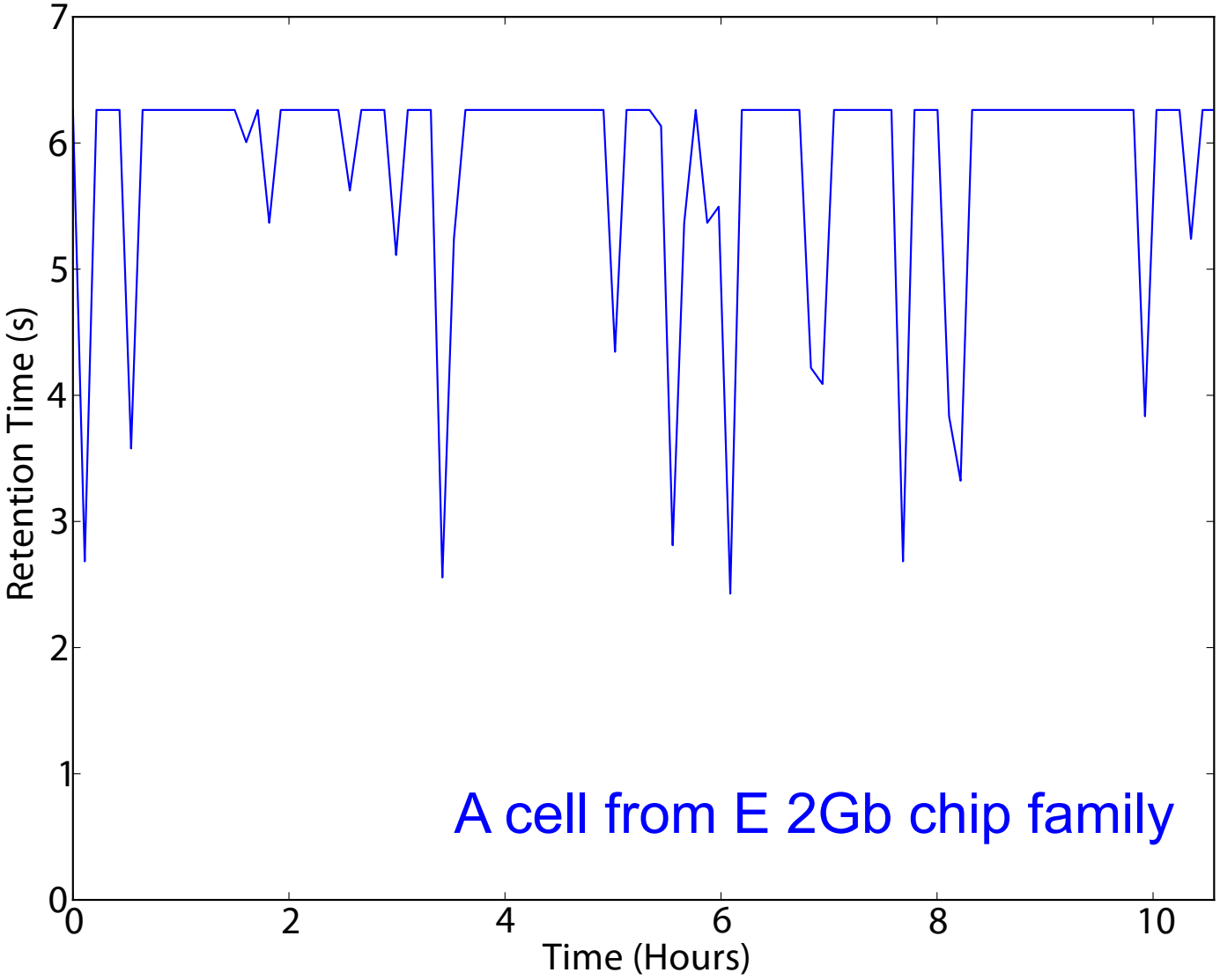
Variable Retention Time

- 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

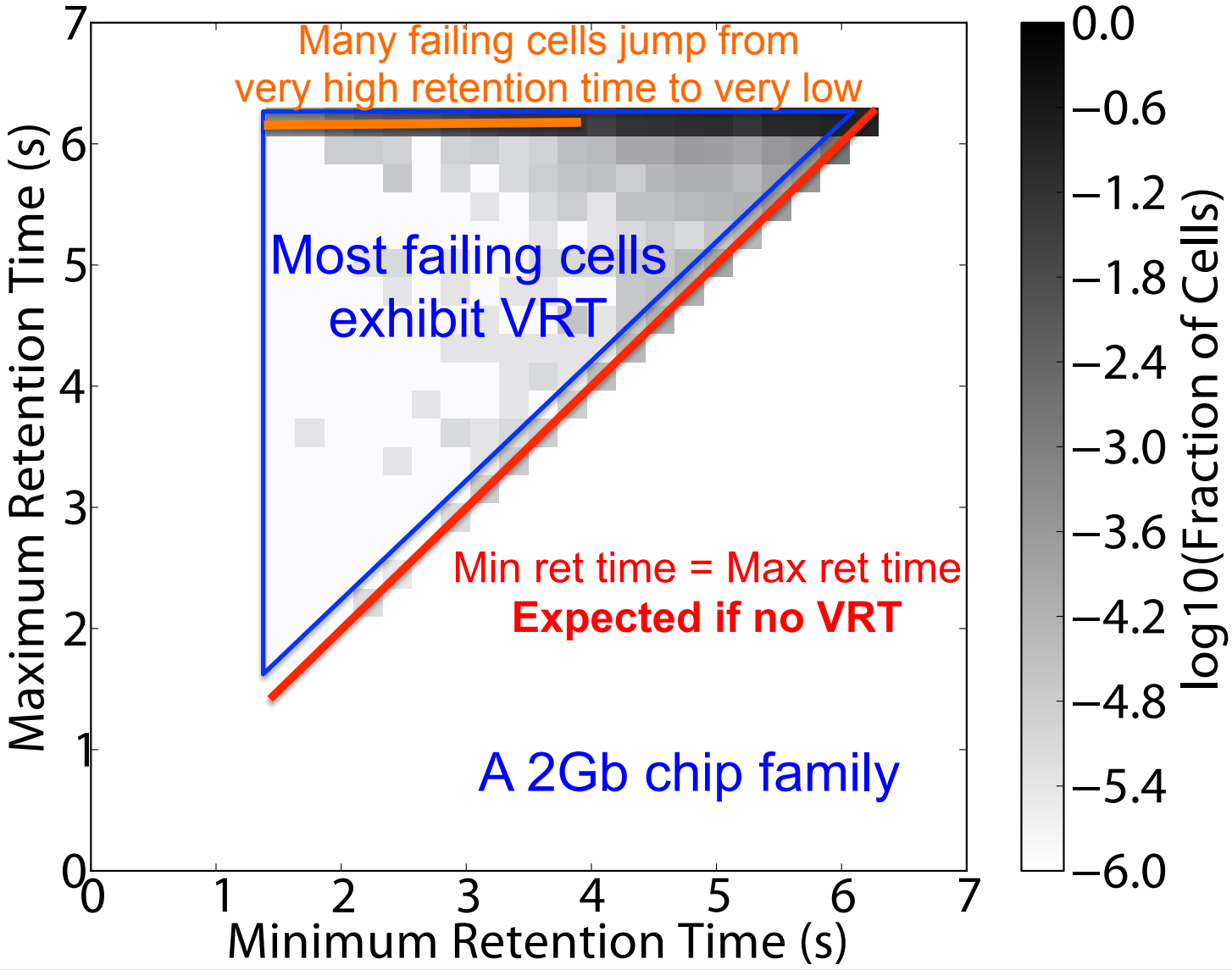


VRT: Questions and Methodology

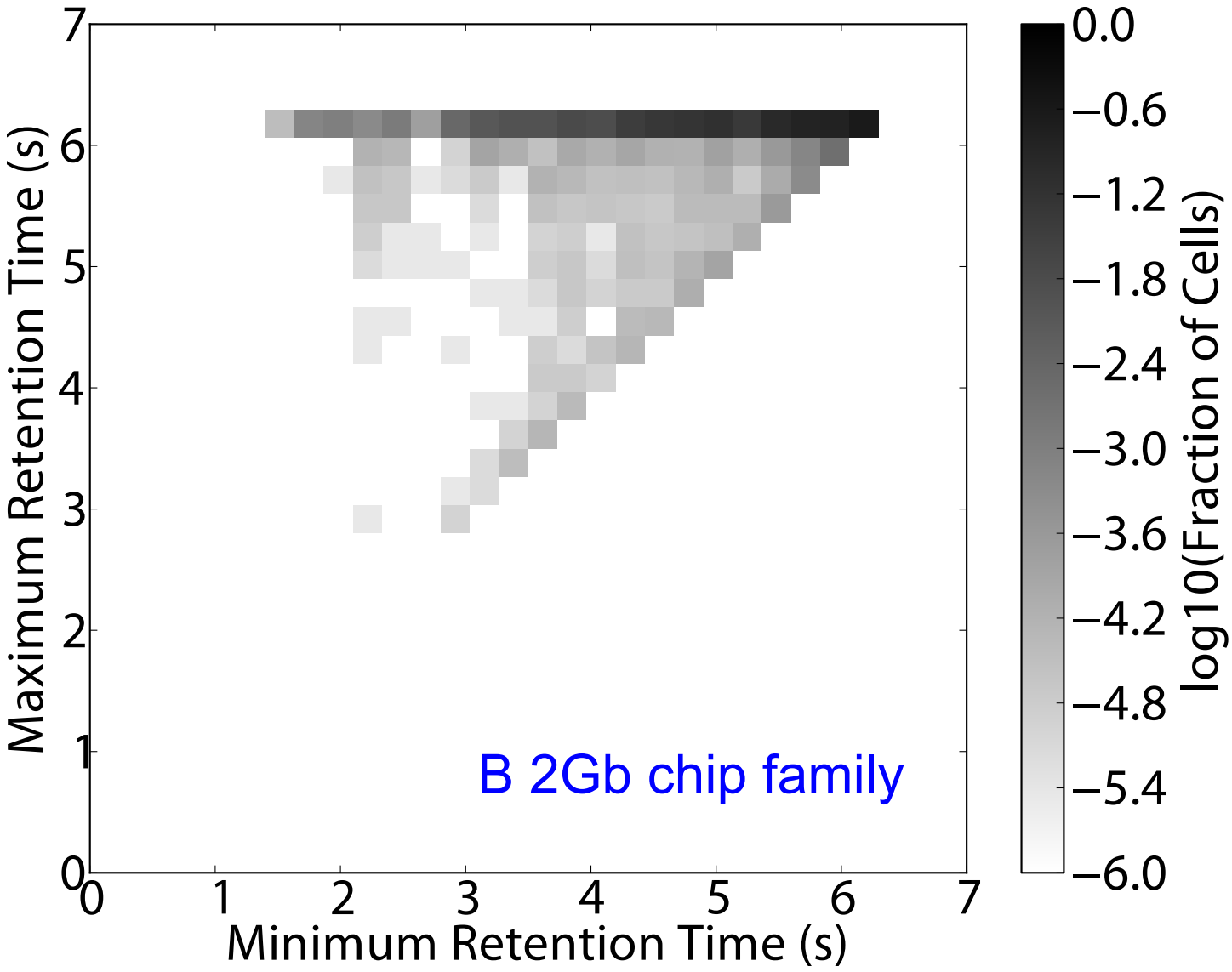
- Key Questions
  - How prevalent is VRT in modern DRAM devices?
  - What is the timescale of observation of the lowest retention time state?
  - What are the implications on retention time profiling?

- Test Methodology
  - Each device was tested for at least 1024 rounds over 24 hours
  - Temperature fixed at 45^{o}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

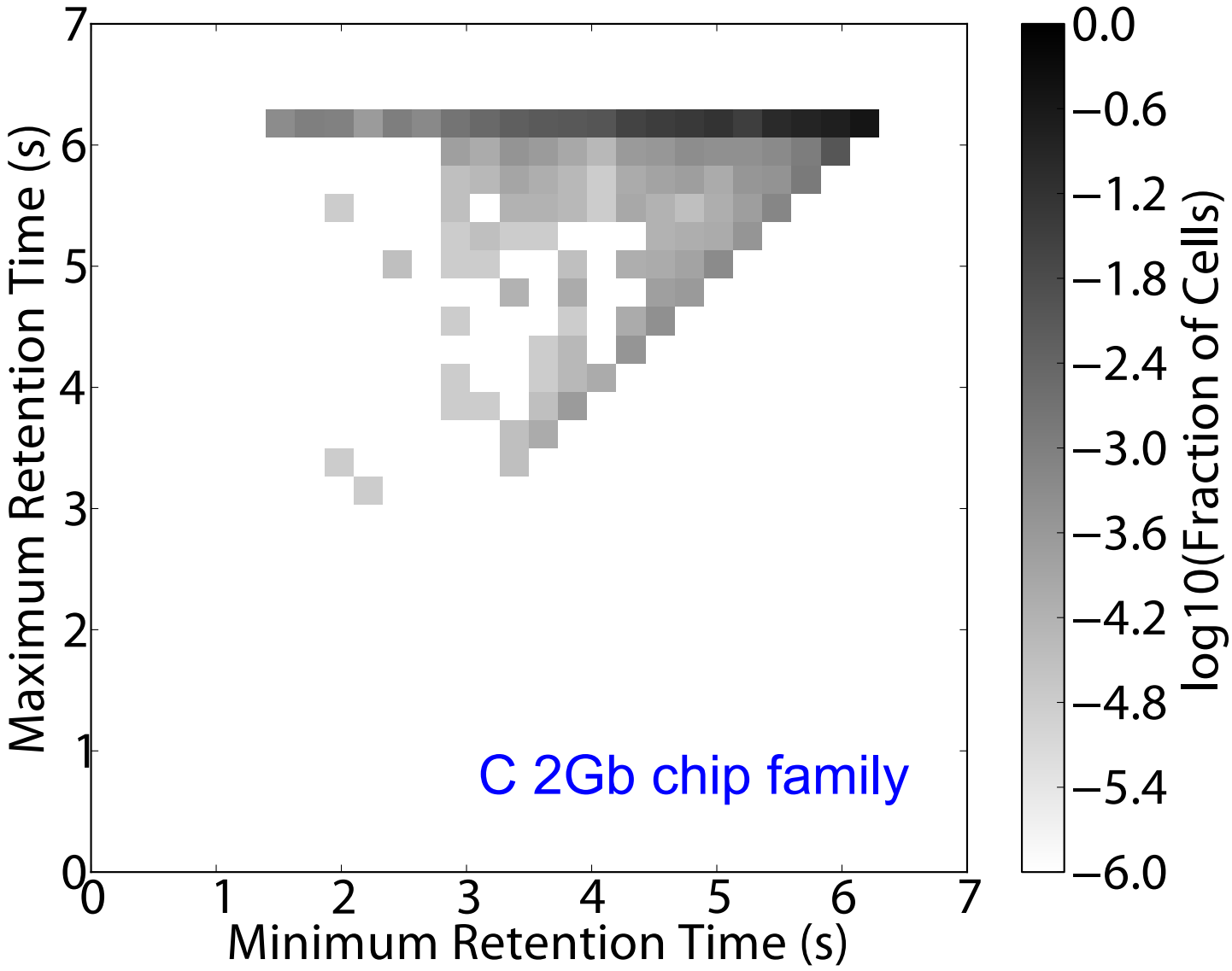
Variable Retention Time



Variable Retention Time



Variable Retention Time



VRT: Observations So Far

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

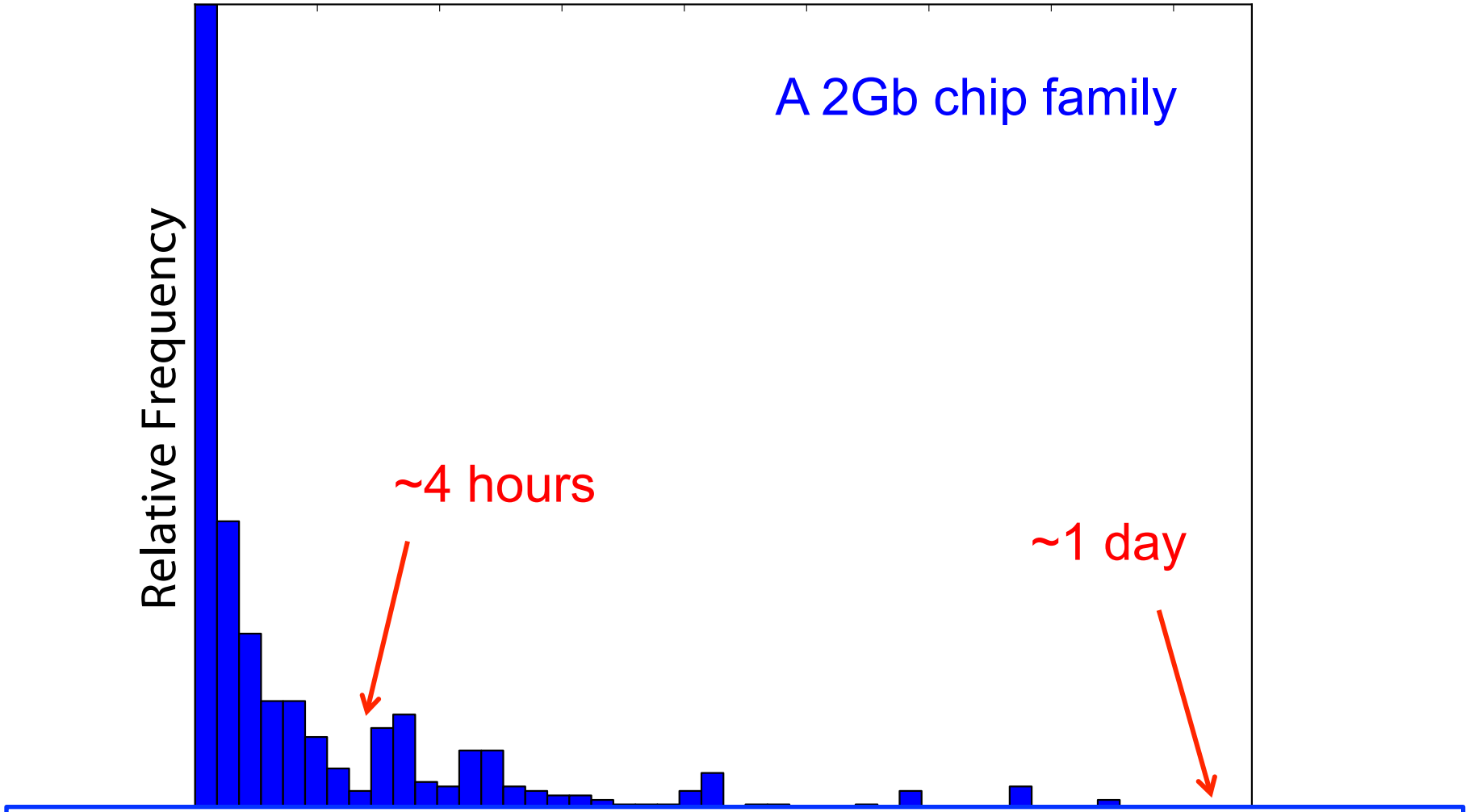
- VRT can result in significant retention time changes
  - Difference between minimum and maximum retention times of a cell can be more than 4x, and may not be bounded
  - Implication: Finding *a* retention time for a cell and using a guardband to ensure minimum retention time is "covered" requires a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

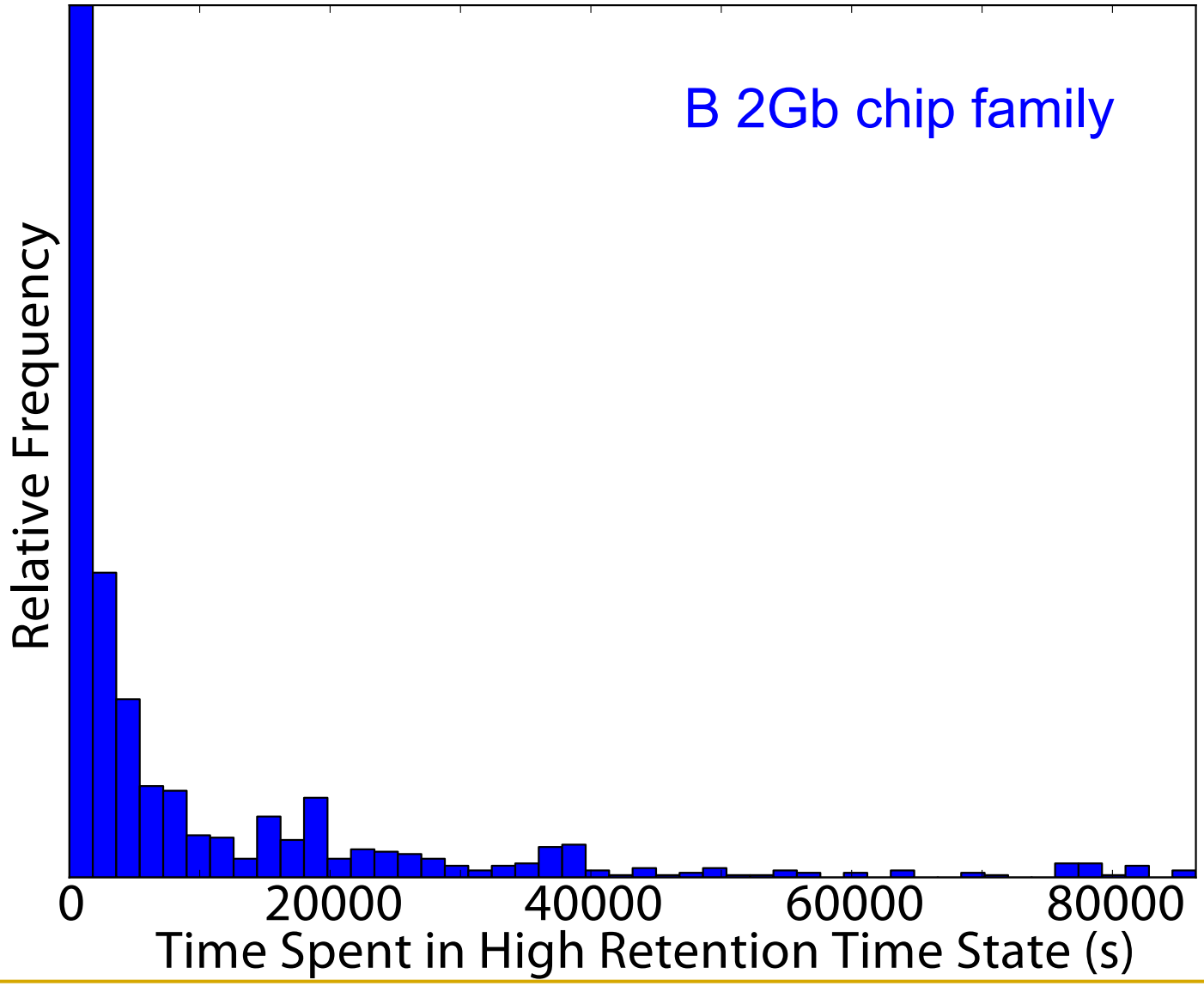
- How much time does a cell spend in a high retention state before switching to the minimum observed retention time state?

Time Spent in High Retention Time State

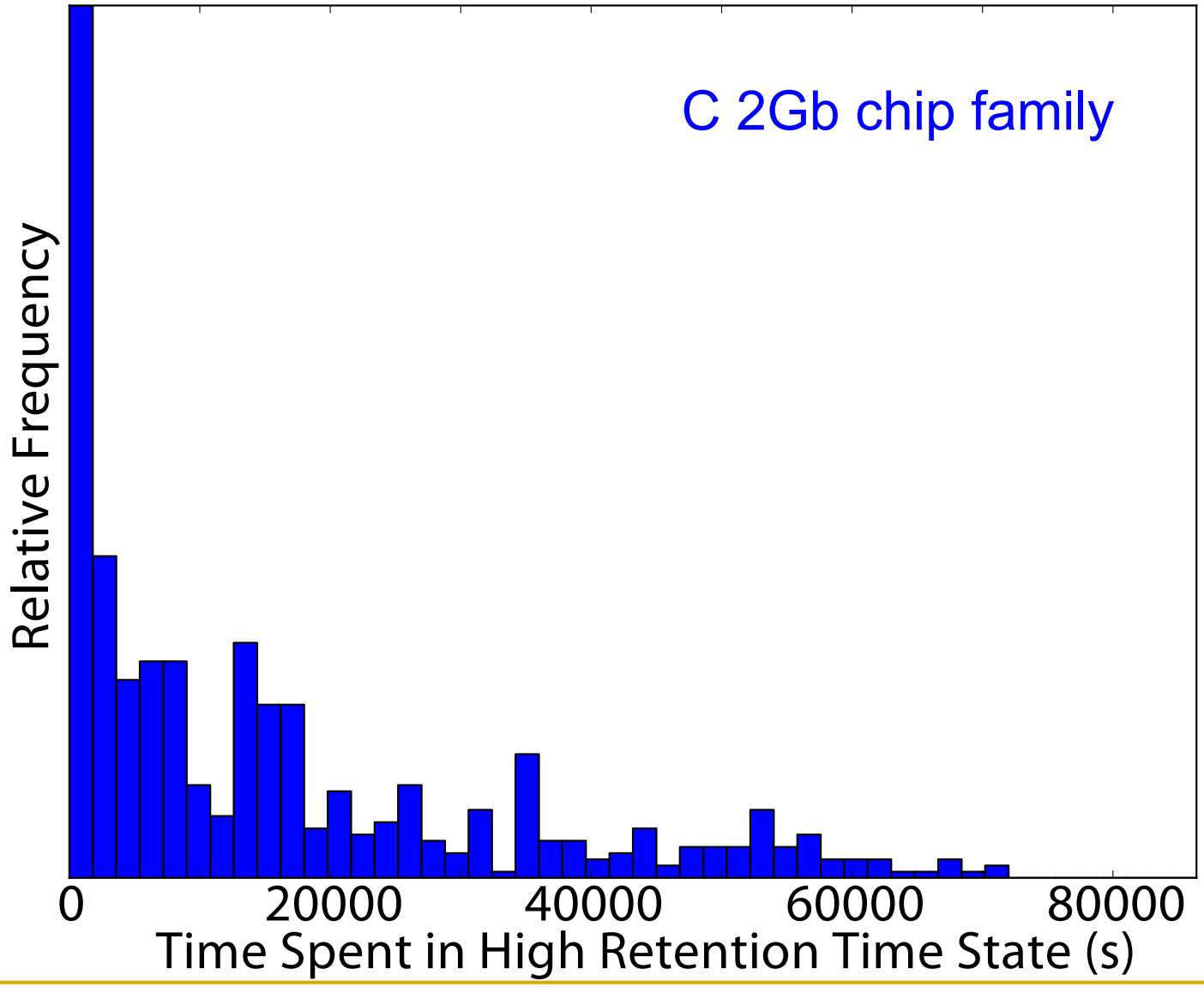


Need to profile for a long time to get to the minimum retention time state

Time Spent in High Retention Time State



Time Spent in High Retention Time State



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

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

Enabling Emerging Memory Technologies

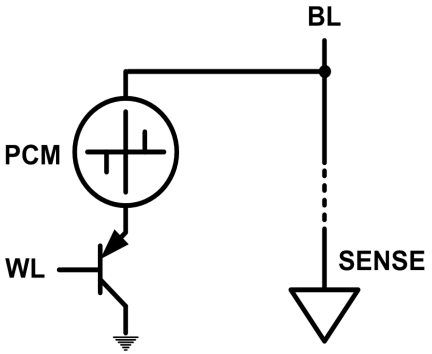
Aside: Scaling Flash Memory [Cai+, ICCD'12]

- 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

Solution 2: Emerging Memory Technologies

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

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



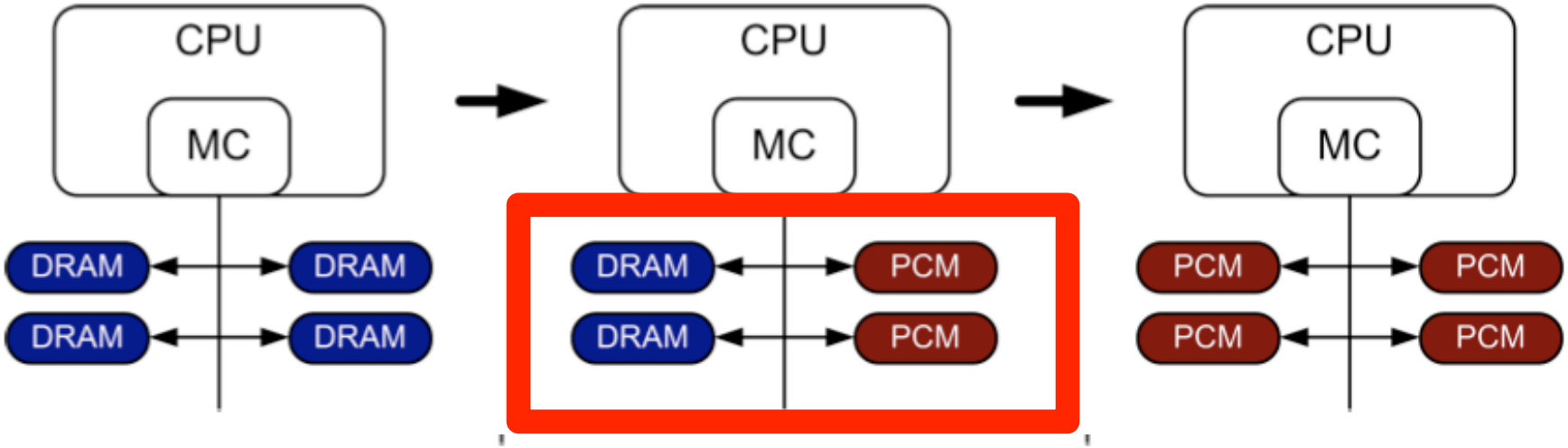
- But, emerging technologies have (many) shortcomings
  - Can they be enabled to replace/augment/surpass DRAM?

Phase Change Memory: Pros and Cons

- Pros over DRAM
  - Better technology scaling (capacity and cost)
  - Non volatility
  - Low idle power (no refresh)
- Cons
  - Higher latencies: ~4-15x DRAM (especially write)
  - Higher active energy: ~2-50x DRAM (especially write)
  - Lower endurance (a cell dies after ~10^8 writes)
- Challenges in enabling PCM as DRAM replacement/helper:
  - Mitigate PCM shortcomings
  - Find the right way to place PCM in the system

PCM-based Main Memory (I)

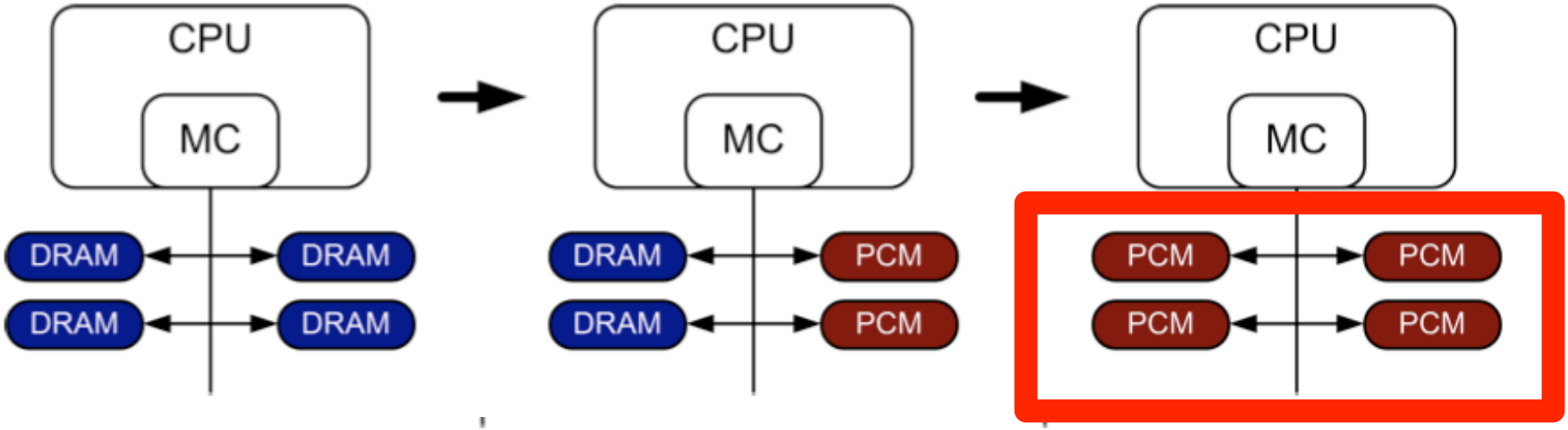
- How should PCM-based (main) memory be organized?



- Hybrid PCM+DRAM [Qureshi+ ISCA'09, Dhiman+ DAC'09]:
  - How to partition/migrate data between PCM and DRAM

PCM-based Main Memory (II)

- How should PCM-based (main) memory be organized?



- Pure PCM main memory [Lee et al., ISCA'09, Top Picks'10]:
  - How to redesign entire hierarchy (and cores) to overcome PCM shortcomings

PCM-Based Memory Systems: Research Challenges

- Partitioning
  - Should DRAM be a cache or main memory, or configurable?
  - What fraction? How many controllers?
- Data allocation/movement (energy, performance, lifetime)
  - Who manages allocation/movement?
  - What are good control algorithms?
  - How do we prevent degradation of service due to wearout?
- Design of cache hierarchy, memory controllers, OS
  - Mitigate PCM shortcomings, exploit PCM advantages
- Design of PCM/DRAM chips and modules
  - Rethink the design of PCM/DRAM with new requirements

An Initial Study: Replace DRAM with PCM

- Lee, Ipek, Mutlu, Burger, "Architecting Phase Change Memory as a Scalable DRAM Alternative," ISCA 2009.
  - Surveyed prototypes from 2003-2008 (e.g. IEDM, VLSI, ISSCC)
  - Derived "average" PCM parameters for F=90nm

Density

- 9 - 12F^2 using BJT
- 1.5× DRAM

Latency

- 50ns Rd, 150ns Wr
- 4×, 12× DRAM

Endurance

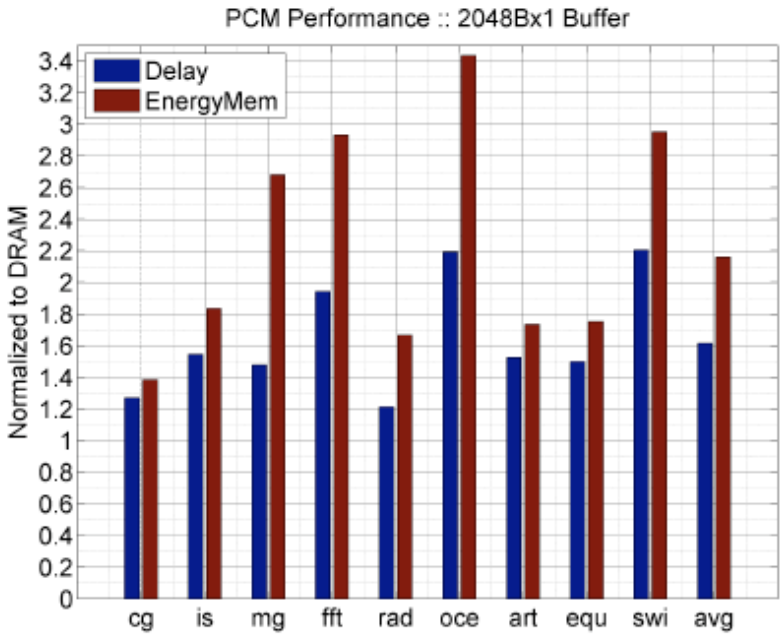
- 1E+08 writes
- 1E-08× DRAM

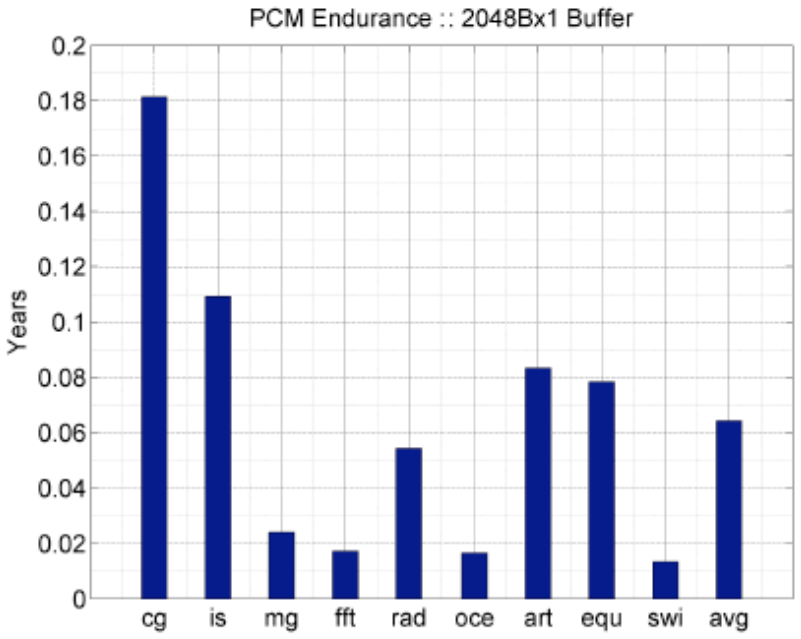
Energy

- 40μA Rd, 150μA Wr
- 2×, 43× DRAM

Results: Naïve Replacement of DRAM with PCM

- Replace DRAM with PCM in a 4-core, 4MB L2 system
- PCM organized the same as DRAM: row buffers, banks, peripherals
- 1.6x delay, 2.2x energy, 500-hour average lifetime



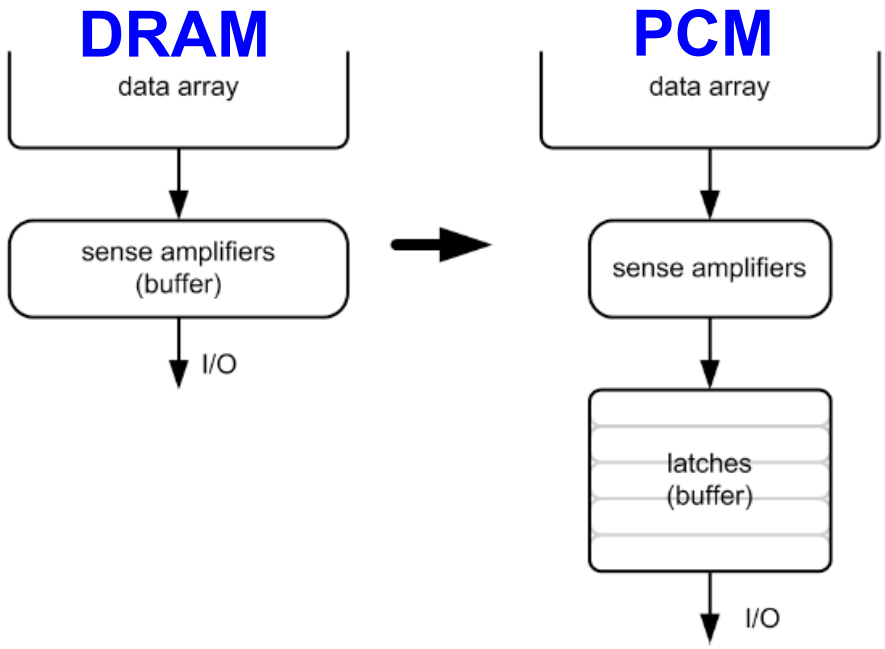


- Lee, Ipek, Mutlu, Burger, "Architecting Phase Change Memory as a Scalable DRAM Alternative," ISCA 2009.

Architecting PCM to Mitigate Shortcomings

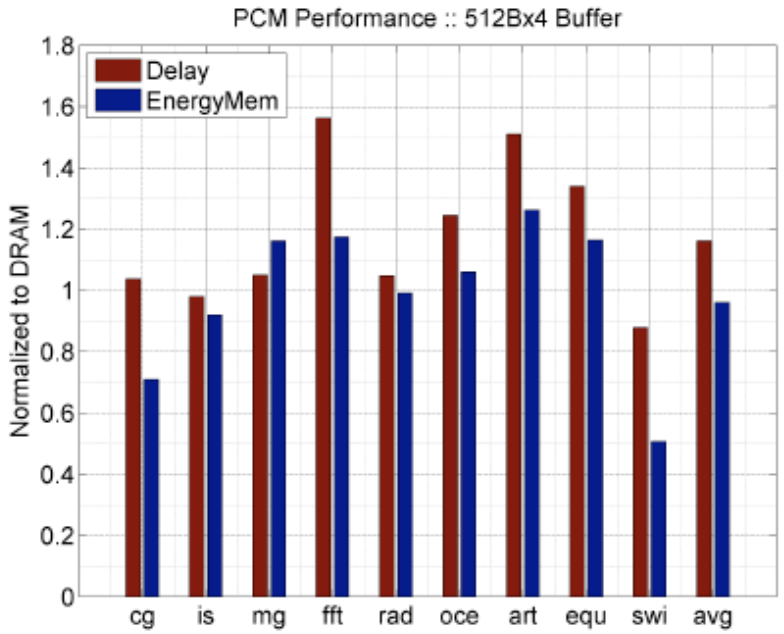
- Idea 1: Use multiple narrow row buffers in each PCM chip
  → Reduces array reads/writes → better endurance, latency, energy

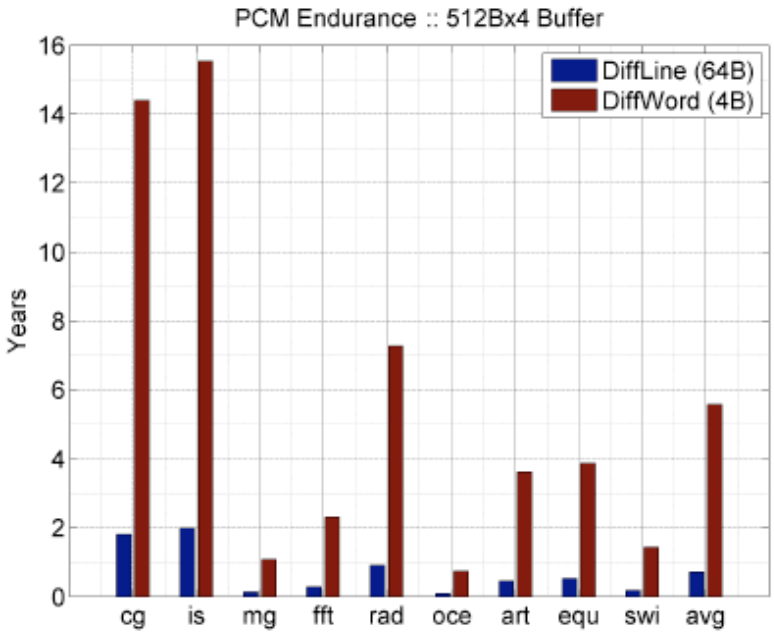
- Idea 2: Write into array at cache block or word granularity
  → Reduces unnecessary wear



Results: Architected PCM as Main Memory

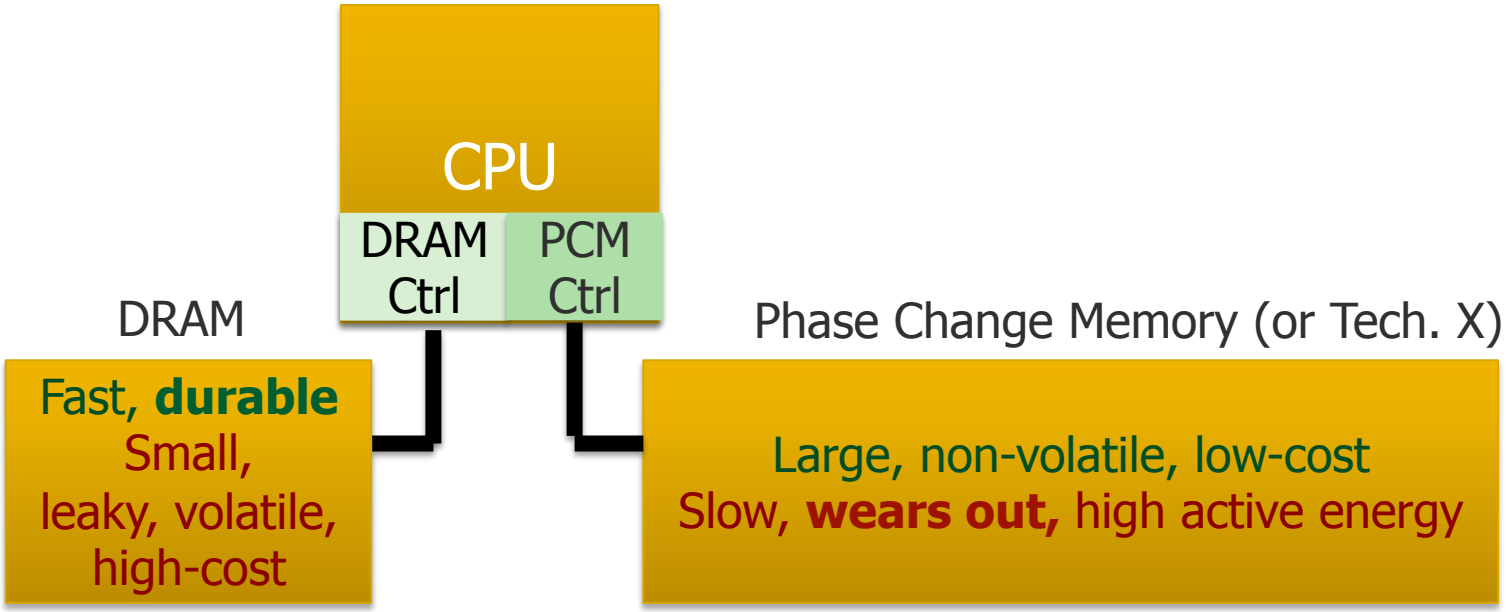
- 1.2x delay, 1.0x energy, 5.6-year average lifetime
- Scaling improves energy, endurance, density





- Caveat 1: Worst-case lifetime is much shorter (no guarantees)
- Caveat 2: Intensive applications see large performance and energy hits
- Caveat 3: Optimistic PCM parameters?

Hybrid Memory Systems



Hardware/software manage data allocation and movement
to achieve the best of multiple technologies
(5-9 years of average lifetime)

Meza, Chang, Yoon, Mutlu, Ranganathan, "Enabling Efficient and Scalable Hybrid Memories," IEEE Comp. Arch. Letters, 2012.

One Option: DRAM as a Cache for PCM

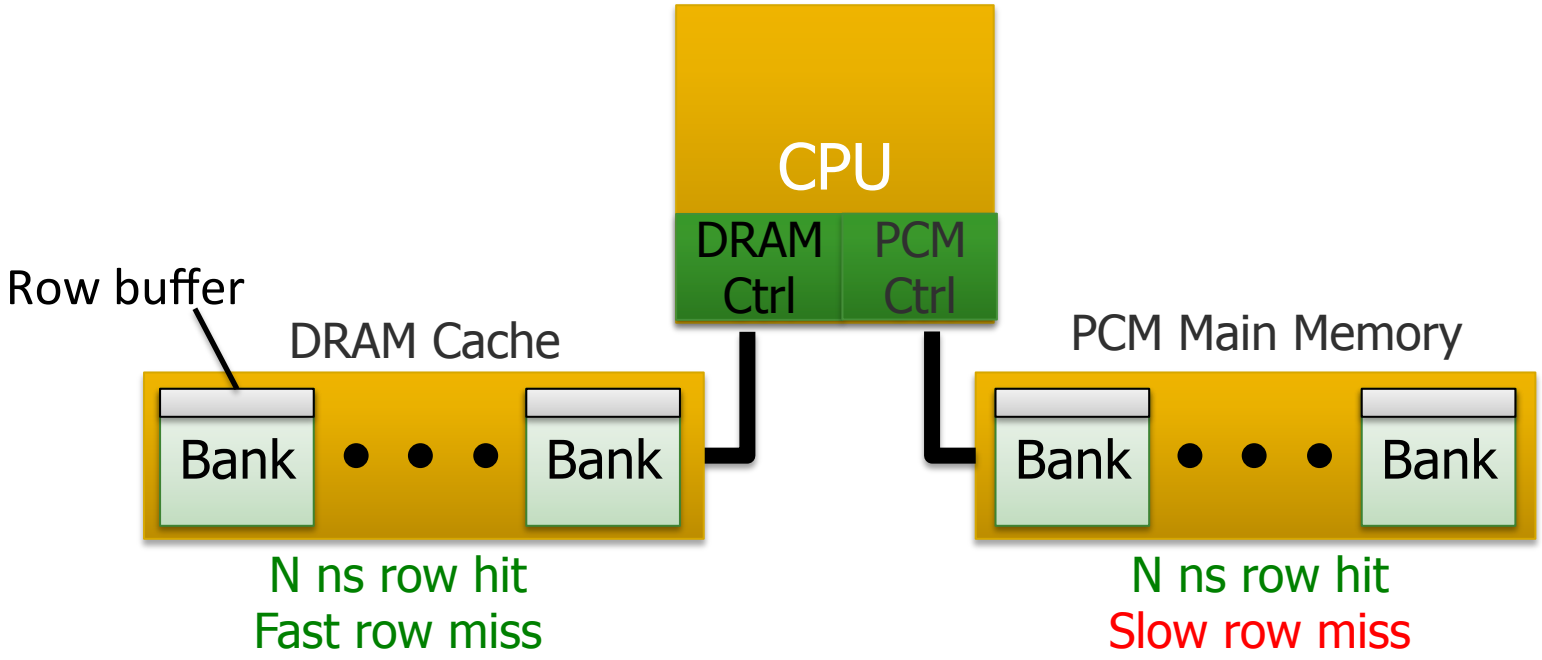
- PCM is main memory; DRAM caches memory rows/blocks
  - Benefits: Reduced latency on DRAM cache hit; write filtering
- Memory controller hardware manages the DRAM cache
  - Benefit: Eliminates system software overhead

- Three issues:
  - What data should be placed in DRAM versus kept in PCM?
  - What is the granularity of data movement?
  - How to design a low-cost hardware-managed DRAM cache?

- Two idea directions:
  - Locality-aware data placement **[Yoon+ , ICCD 2012]**
  - Cheap tag stores and dynamic granularity **[Meza+, IEEE CAL 2012]**

DRAM vs. PCM: An Observation

- Row buffers are the same in DRAM and PCM
- Row buffer hit latency **same** in DRAM and PCM
- Row buffer miss latency **small** in DRAM, **large** in PCM



- Accessing the row buffer in PCM is fast
- What incurs high latency is the PCM array access → avoid this

Row-Locality-Aware Data Placement

- Idea: Cache in DRAM only those rows that
  - Frequently cause row buffer conflicts → because row-conflict latency is smaller in DRAM
  - Are reused many times → to reduce cache pollution and bandwidth waste

- Simplified rule of thumb:
  - Streaming accesses: Better to place in PCM
  - Other accesses (with some reuse): Better to place in DRAM

- Bridges half of the performance gap between all-DRAM and all-PCM memory on memory-intensive workloads

- Yoon et al., "Row Buffer Locality-Aware Caching Policies for Hybrid Memories," ICCD 2012.

Row-Locality-Aware Data Placement: Mechanism

- For a subset of rows in PCM, memory controller:
  - Tracks **row conflicts** as a predictor of future locality
  - Tracks **accesses** as a predictor of future reuse

- Cache a row in DRAM if its row conflict and access counts are greater than certain thresholds

- Determine thresholds dynamically to adjust to application/ workload characteristics
  - Simple cost/benefit analysis every fixed interval

Implementation: "Statistics Store"

- Goal: To keep count of row buffer misses to recently used rows in PCM

- Hardware structure in memory controller
  - Operation is similar to a cache
    - Input: row address
    - Output: row buffer miss count
  - 128-set 16-way statistics store (9.25KB) achieves system performance within 0.3% of an unlimited-sized statistics store

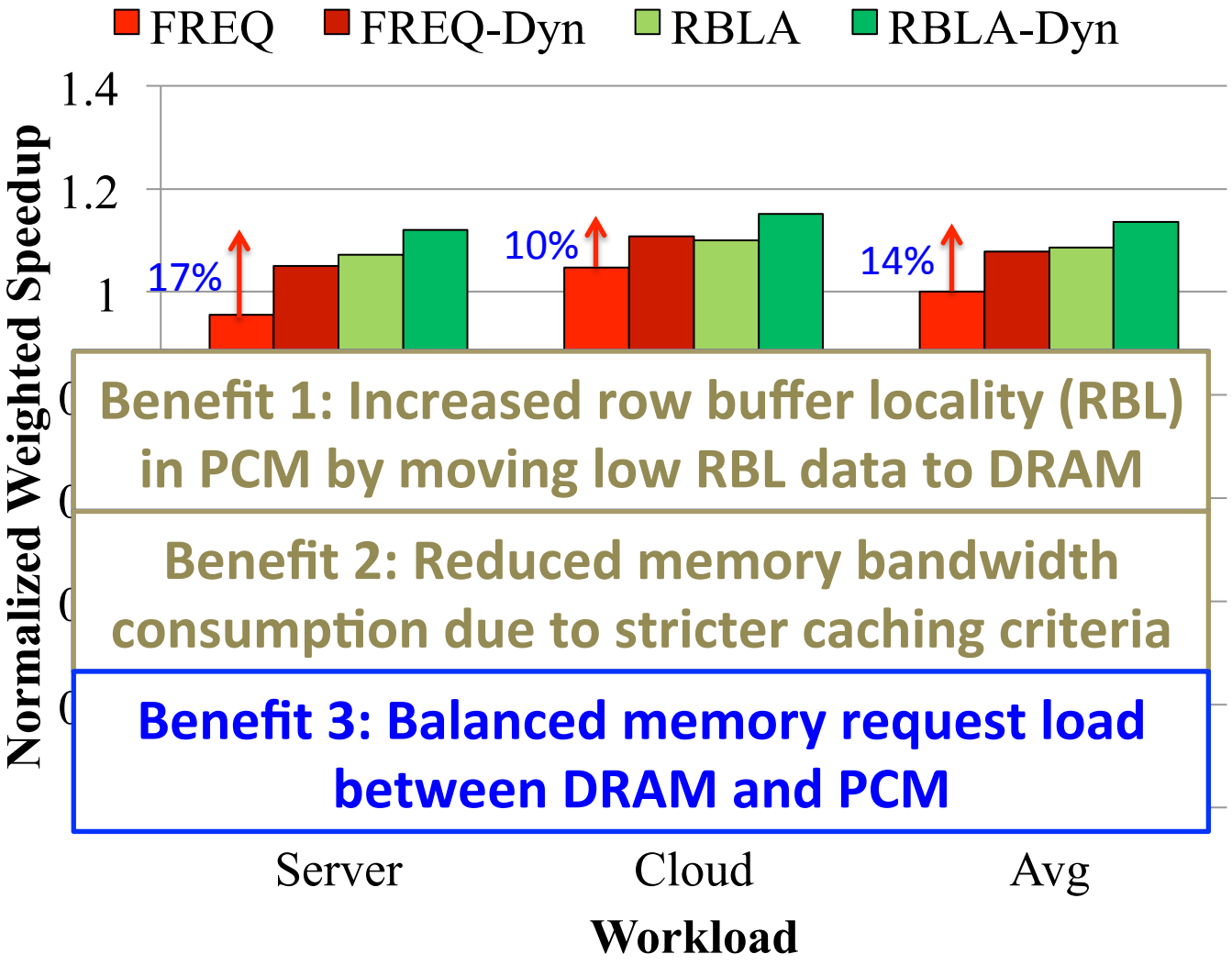
Evaluation Methodology

- Cycle-level x86 CPU-memory simulator
  - **CPU**: 16 out-of-order cores, 32KB private L1 per core, 512KB shared L2 per core
  - **Memory**: 1GB DRAM (8 banks), 16GB PCM (8 banks), 4KB migration granularity
- 36 multi-programmed server, cloud workloads
  - Server: TPC-C (OLTP), TPC-H (Decision Support)
  - Cloud: Apache (Webserv.), H.264 (Video), TPC-C/H
- Metrics: Weighted speedup (perf.), perf./Watt (energy eff.), Maximum slowdown (fairness)

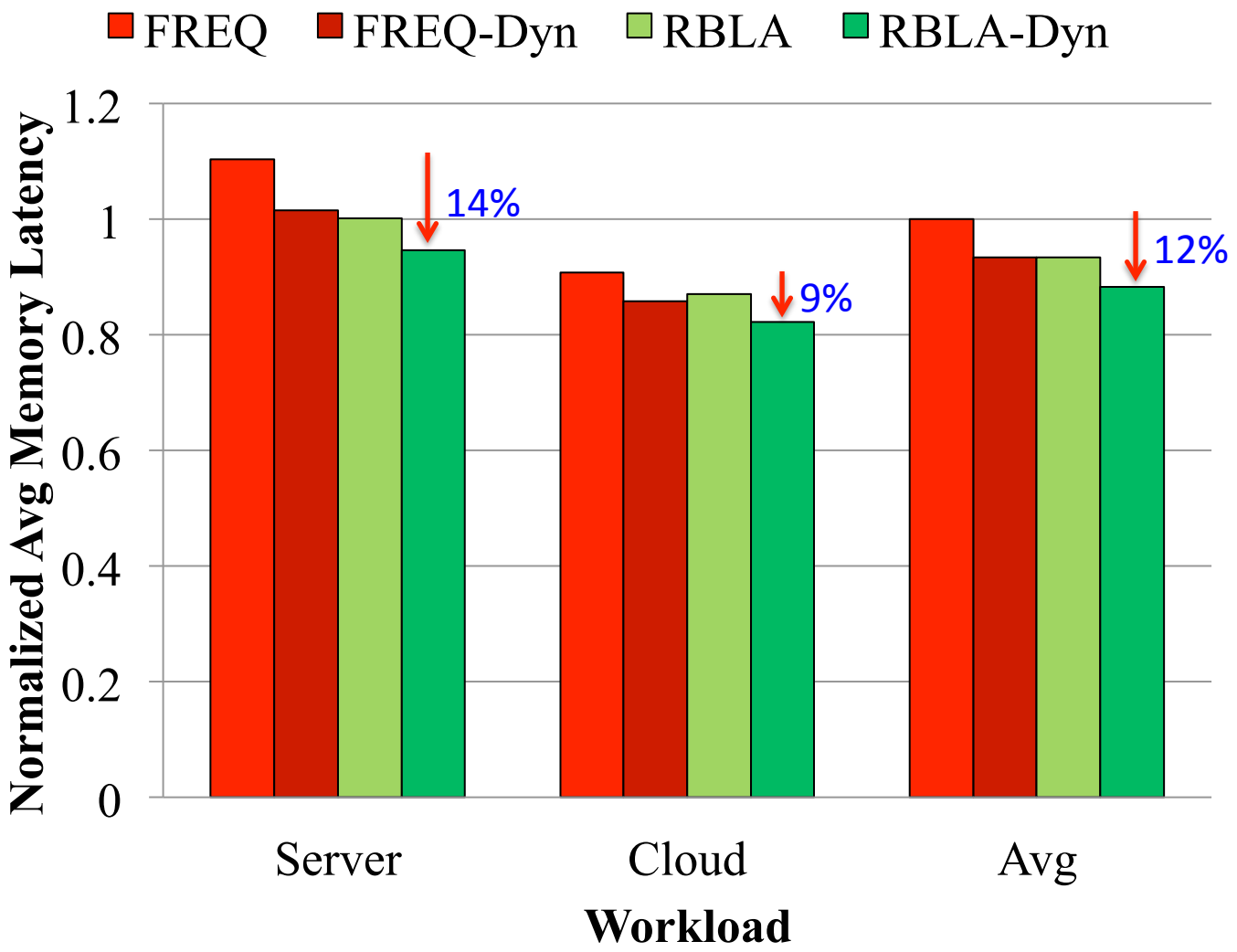
Comparison Points

- **Conventional LRU Caching**
- **FREQ**: Access-frequency-based caching
  - Places "hot data" in cache [Jiang+ HPCA'10]
  - Cache to DRAM rows with accesses ≥ threshold
  - Row buffer locality-*unaware*
- **FREQ-Dyn**: Adaptive Freq.-based caching
  - **FREQ** + our dynamic threshold adjustment
  - Row buffer locality-*unaware*
- **RBLA**: Row buffer locality-aware caching
- **RBLA-Dyn**: Adaptive RBL-aware caching

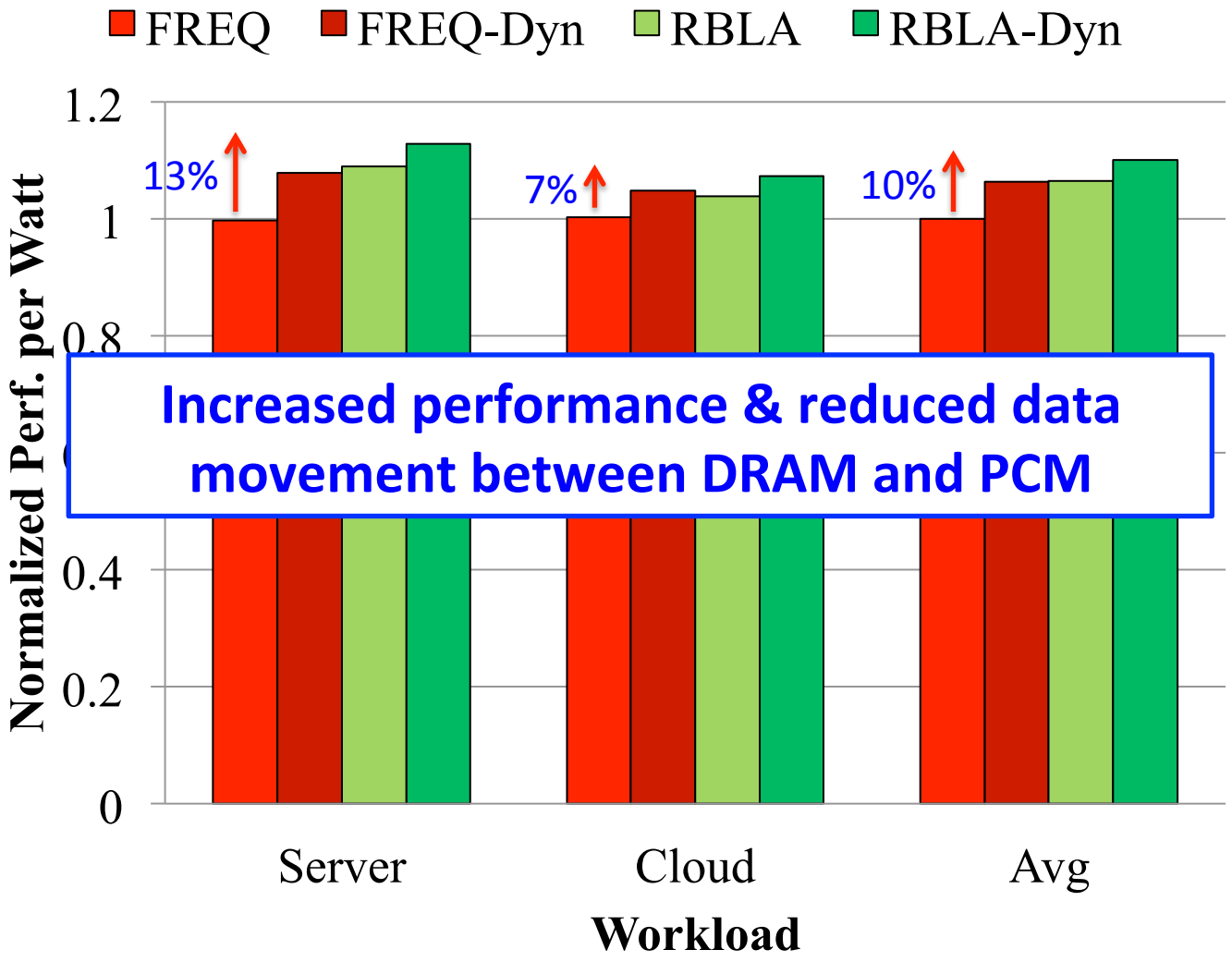
System Performance



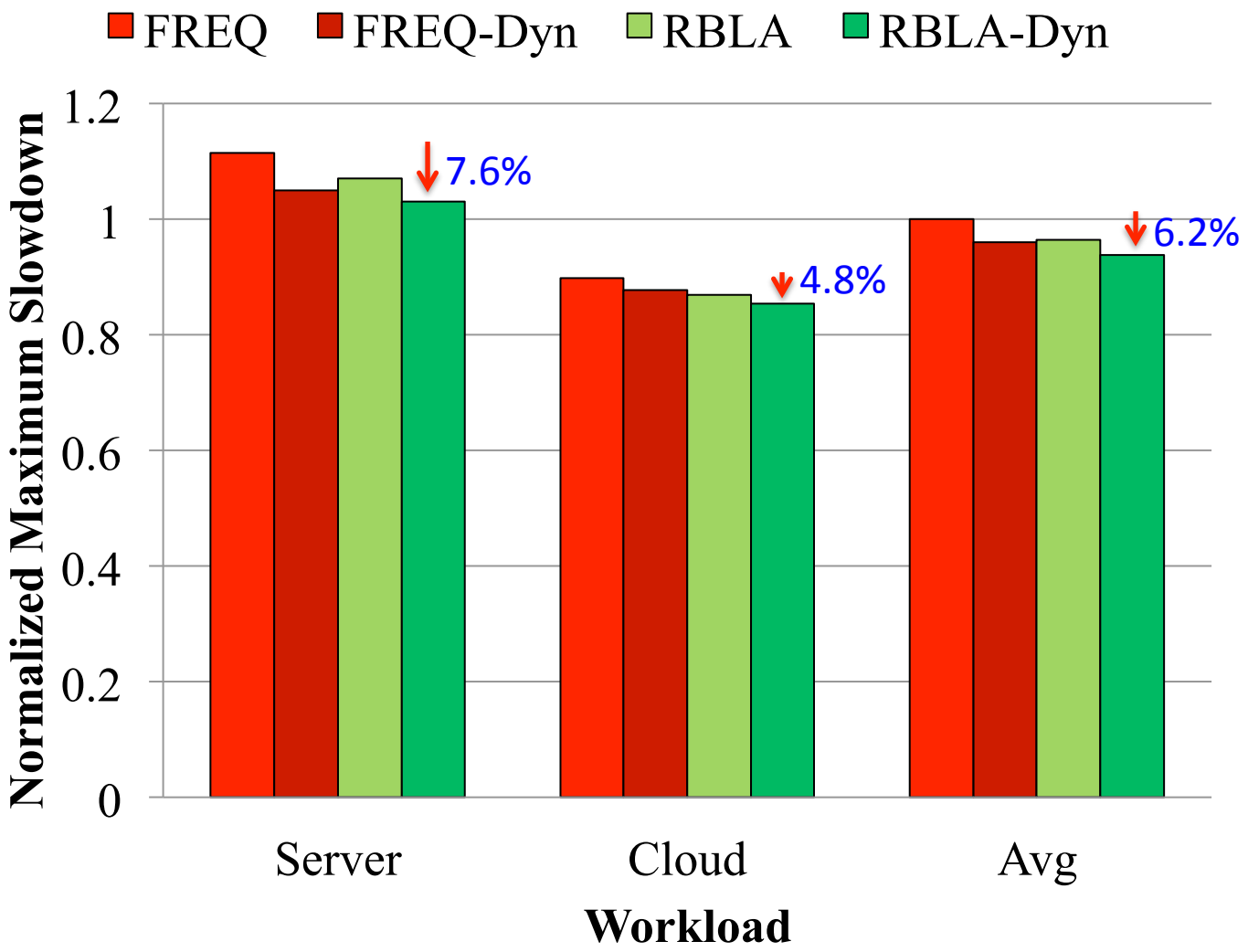
Average Memory Latency



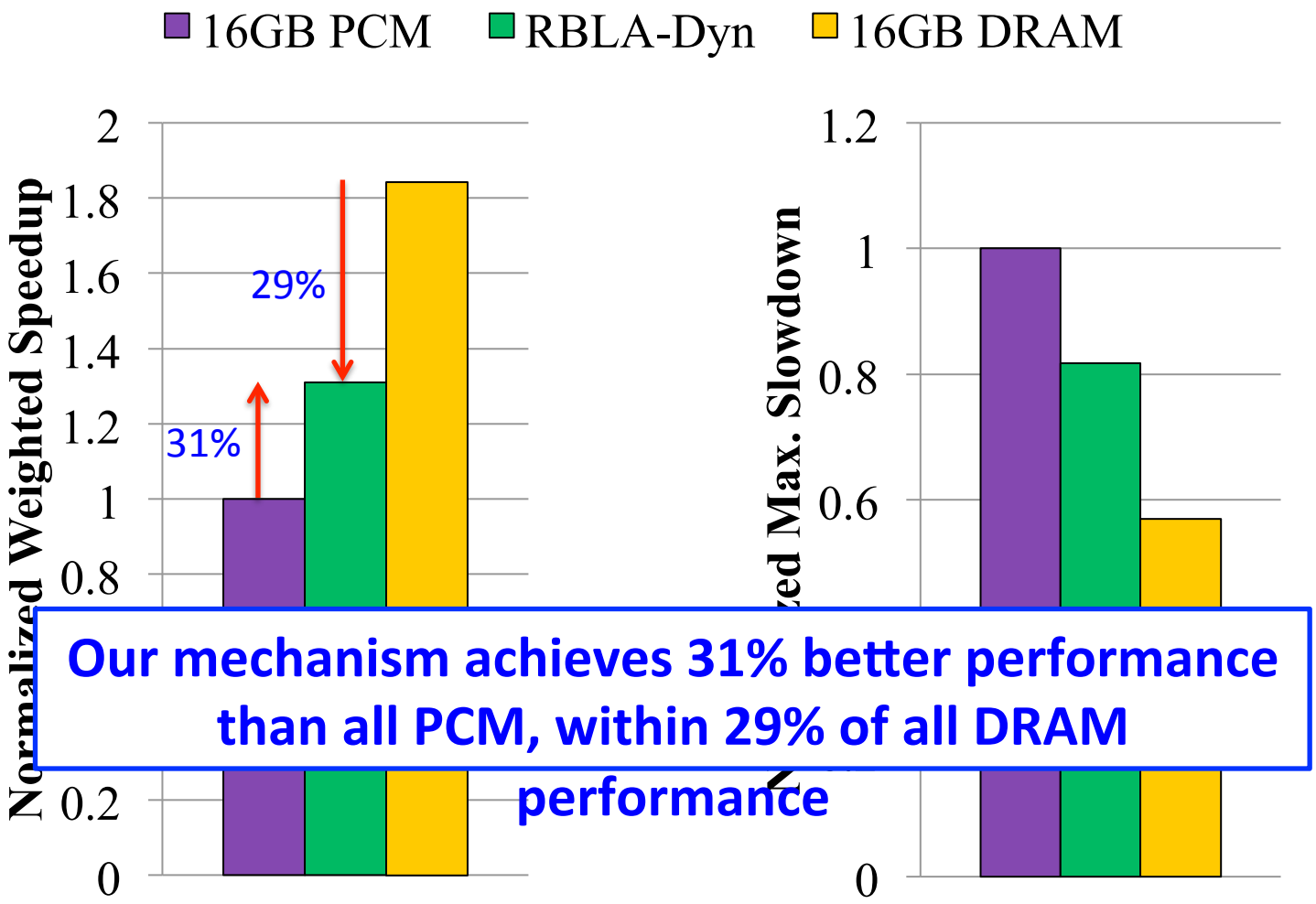
Memory Energy Efficiency



Thread Fairness

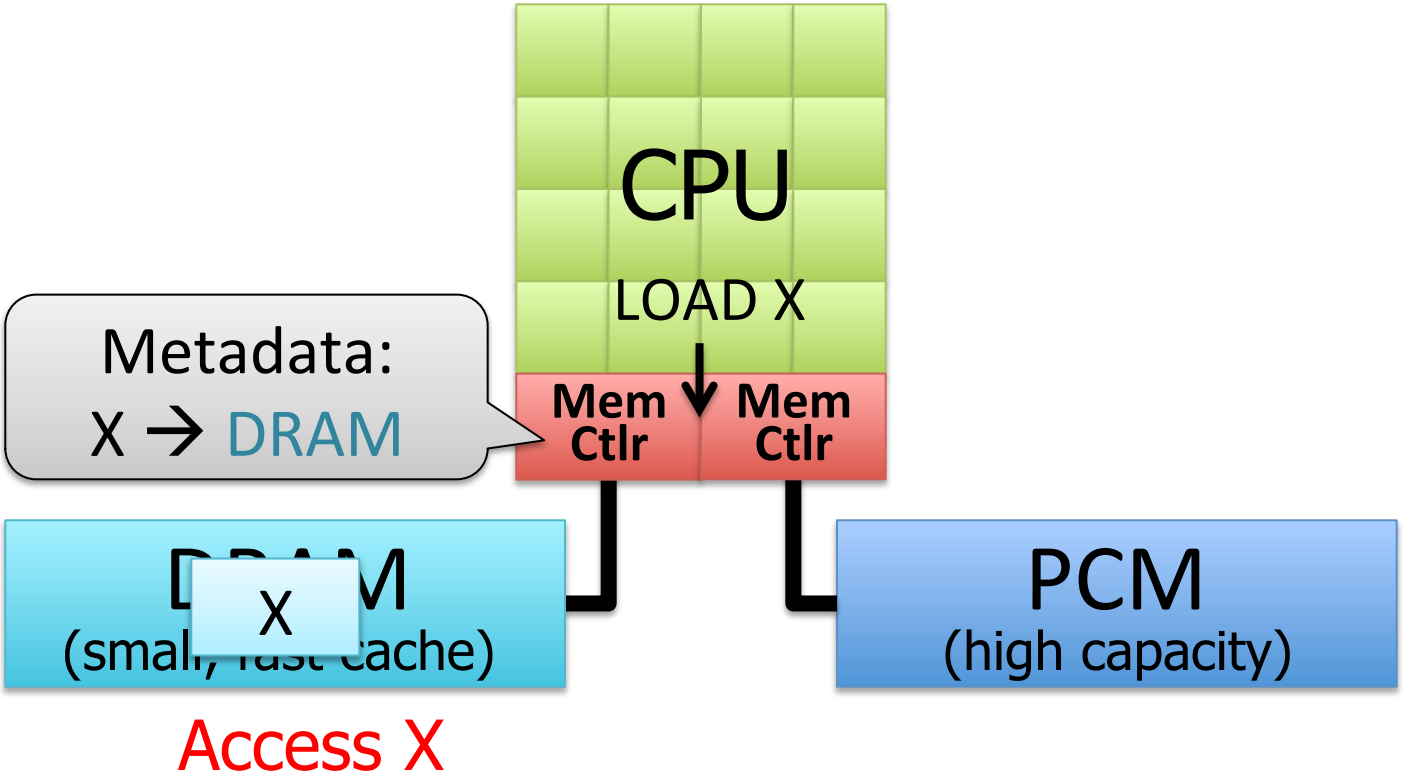


Compared to All-PCM/DRAM



The Problem with Large DRAM Caches

- A large DRAM cache requires a large metadata (tag + block-based information) store
- How do we design an efficient DRAM cache?



Idea 1: Tags in Memory

- Store tags in the same row as data in DRAM
  - Store metadata in same row as their data
  - Data and metadata can be accessed together

- Benefit: No on-chip tag storage overhead
- Downsides:
  - Cache hit determined only after a DRAM access
  - Cache hit requires two DRAM accesses

Idea 2: Cache Tags in SRAM

- Recall Idea 1: Store all metadata in DRAM
  - To reduce metadata storage overhead

- Idea 2: Cache in on-chip SRAM frequently-accessed metadata
  - Cache only a small amount to keep SRAM size small

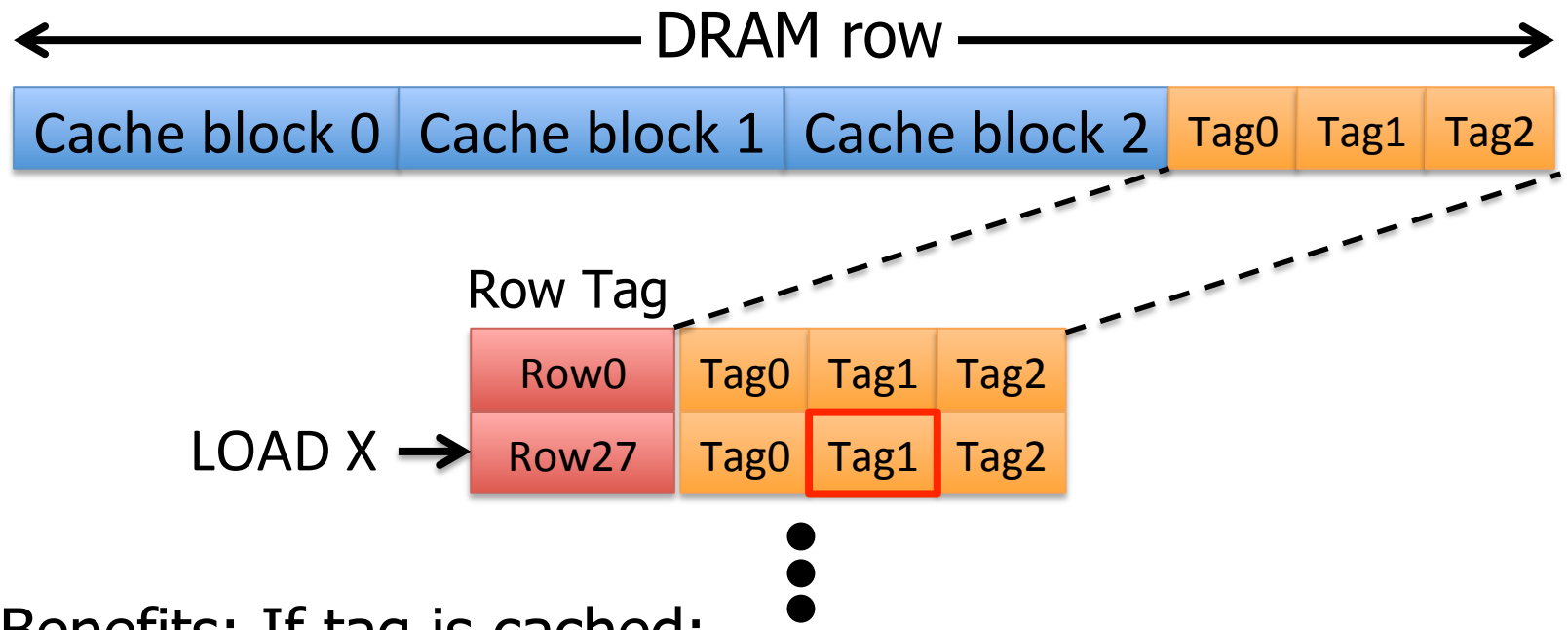
Idea 3: Dynamic Data Transfer Granularity

- Some applications benefit from caching more data
  - They have good spatial locality
- Others do not
  - Large granularity wastes bandwidth and reduces cache utilization

- Idea 3: Simple dynamic caching granularity policy
  - Cost-benefit analysis to determine best DRAM cache block size
  - Group main memory into sets of rows
  - Some row sets follow a fixed caching granularity
  - The rest of main memory follows the best granularity
    - Cost–benefit analysis: access latency versus number of cachings
    - Performed every quantum

TIMBER Tag Management

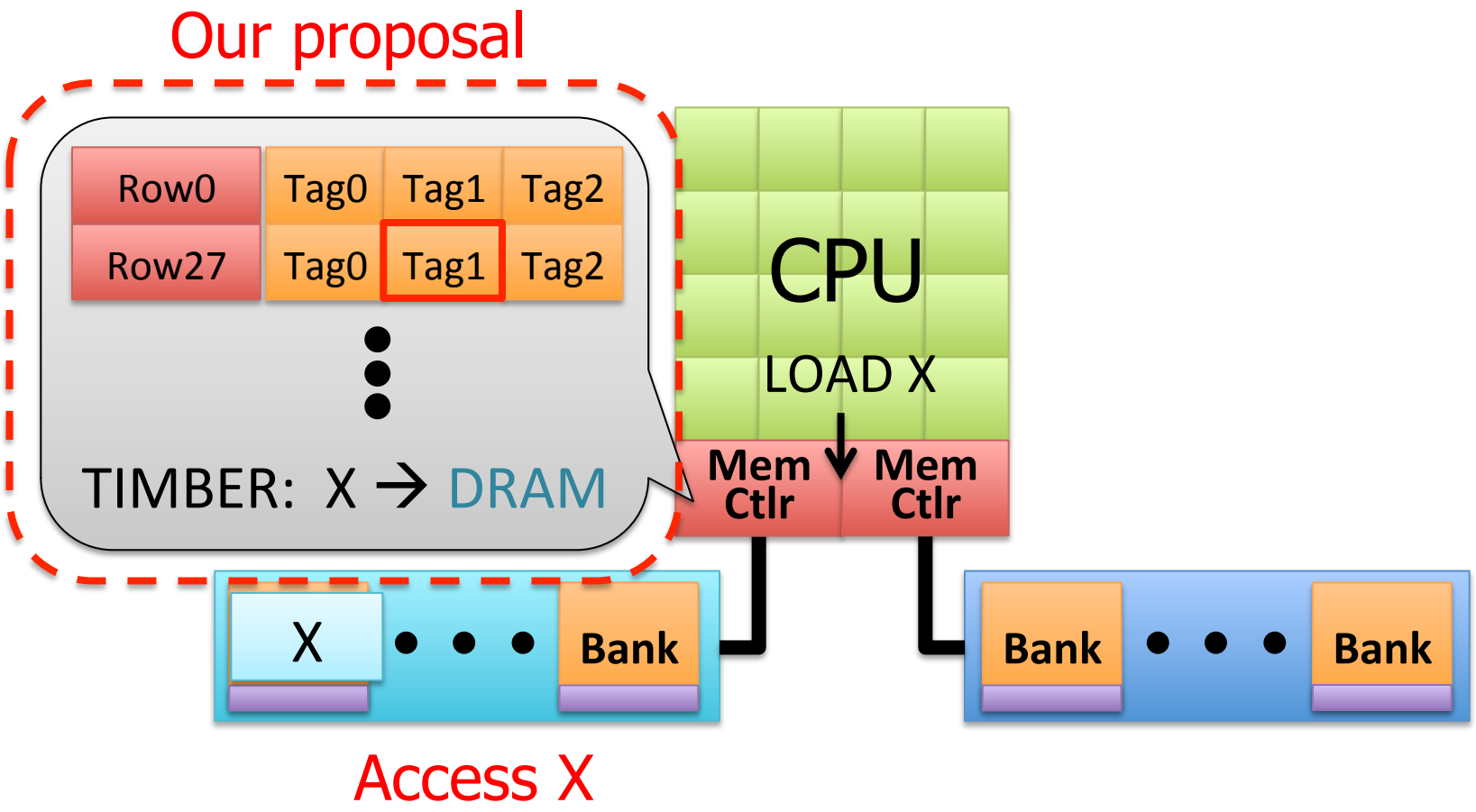
- A Tag-In-Memory BuffER (TIMBER)
  - Stores recently-used tags in a small amount of SRAM



- Benefits: If tag is cached:
  - no need to access DRAM twice
  - cache hit determined quickly

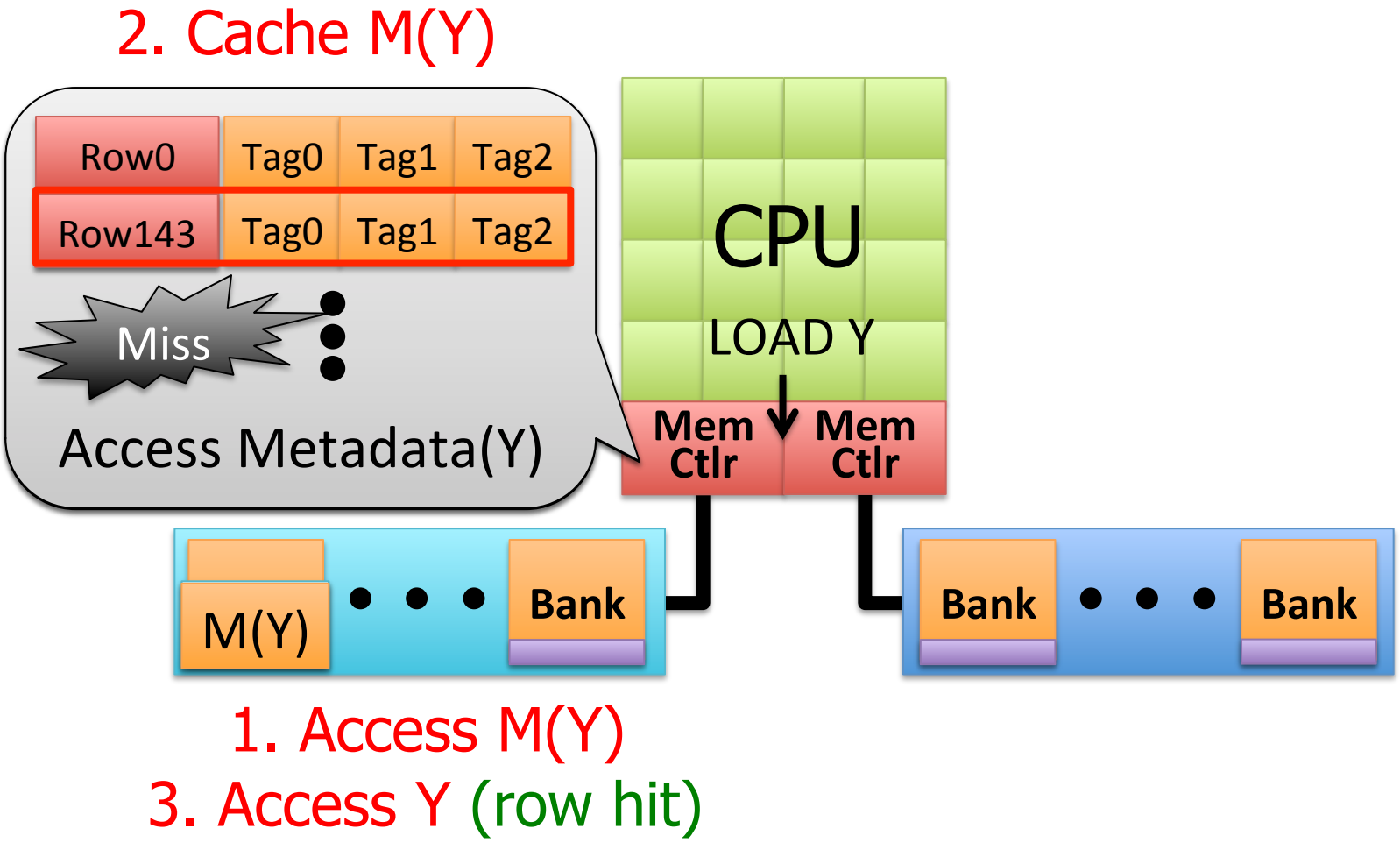
TIMBER Tag Management Example (I)

- Case 1: TIMBER hit



TIMBER Tag Management Example (II)

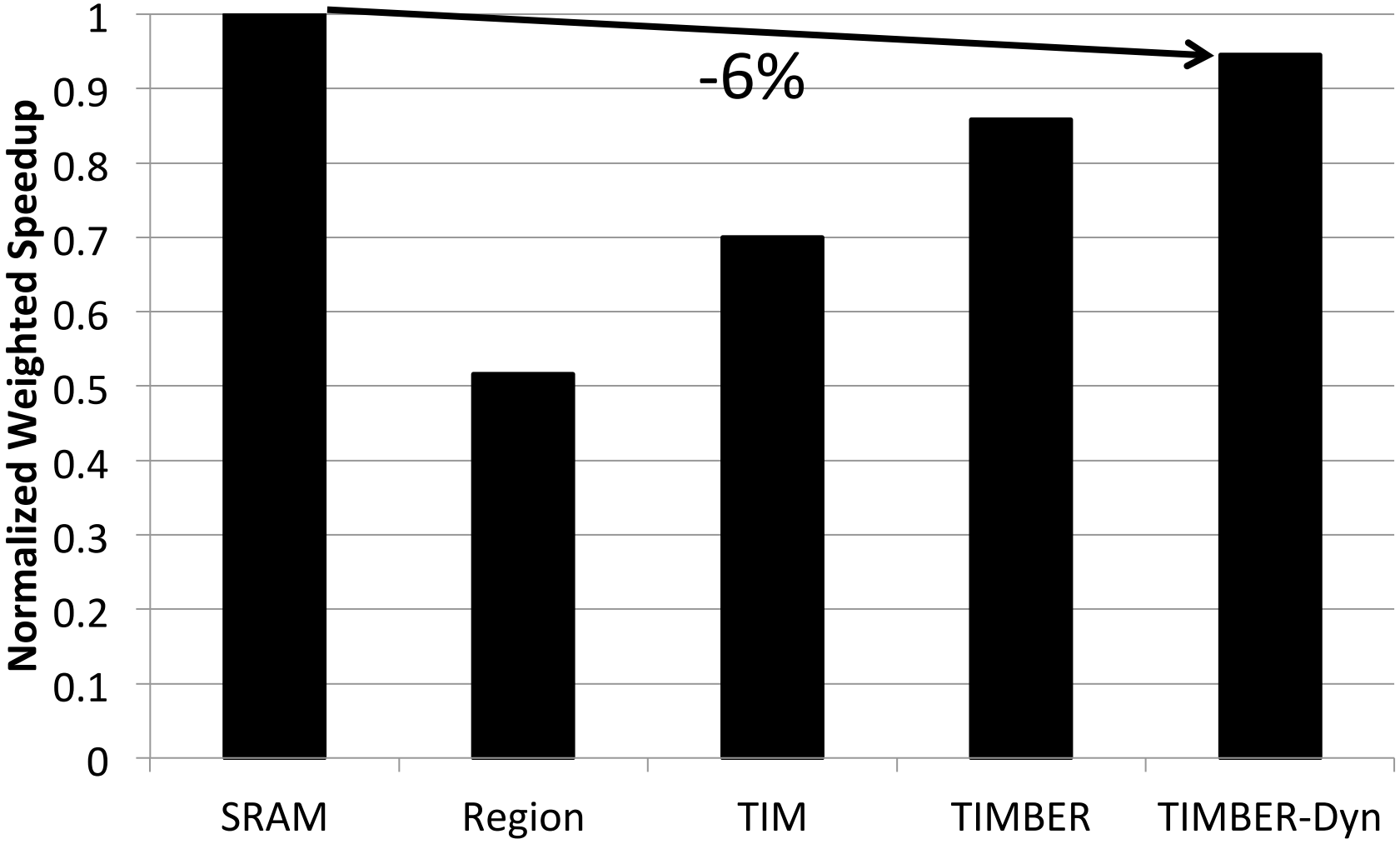
- Case 2: TIMBER miss



Methodology

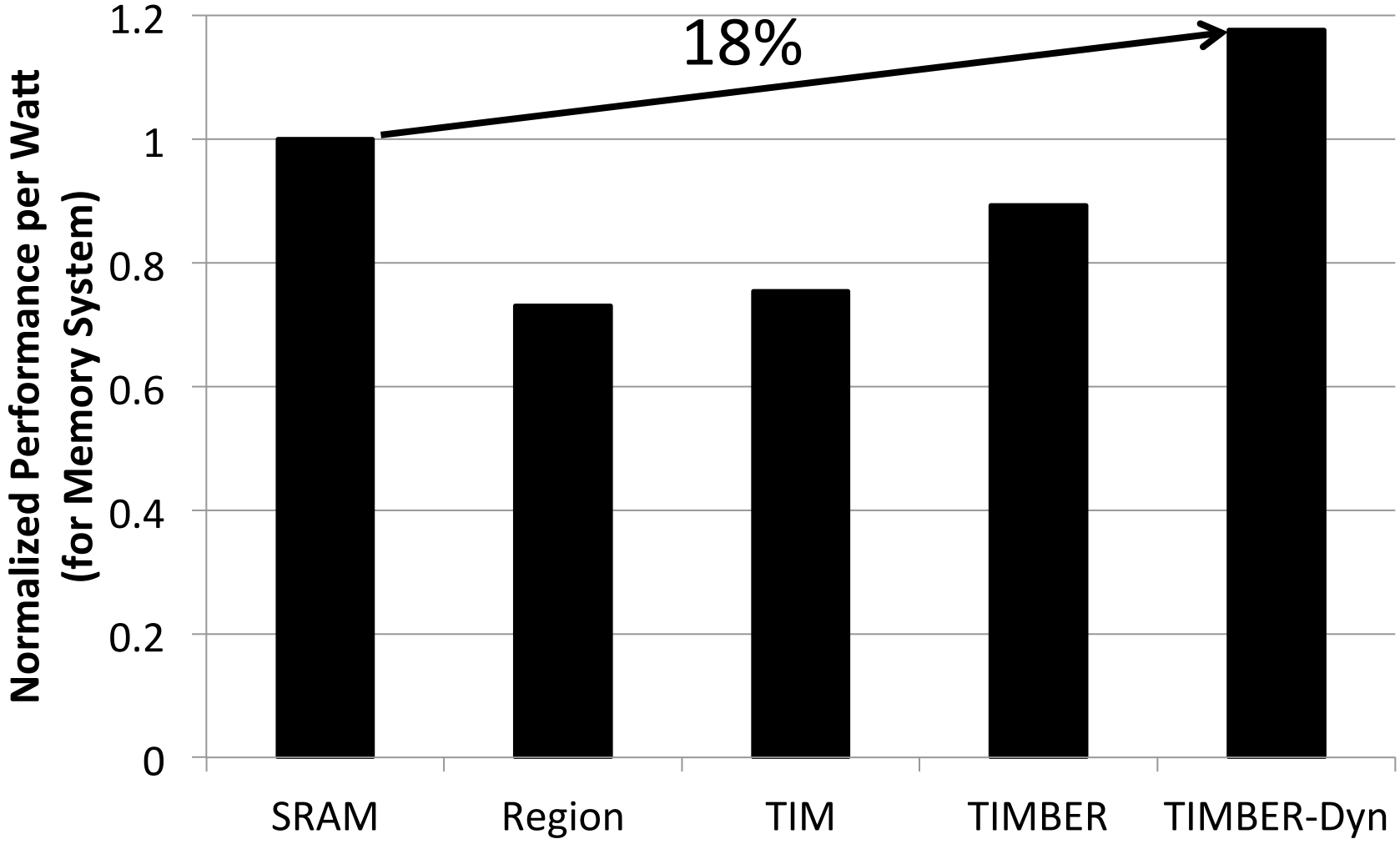
- System: 8 out-of-order cores at 4 GHz
- Memory: 512 MB direct-mapped DRAM, 8 GB PCM
  - 128B caching granularity
  - DRAM row hit (miss): 200 cycles (400 cycles)
  - PCM row hit (clean / dirty miss): 200 cycles (640 / 1840 cycles)
- Evaluated metadata storage techniques
  - All SRAM system (8MB of SRAM)
  - Region metadata storage
  - TIM metadata storage (same row as data)
  - TIMBER, 64-entry direct-mapped (8KB of SRAM)

TIMBER Performance



Meza, Chang, Yoon, Mutlu, Ranganathan, "Enabling Efficient and Scalable Hybrid Memories," IEEE Comp. Arch. Letters, 2012.

TIMBER Energy Efficiency



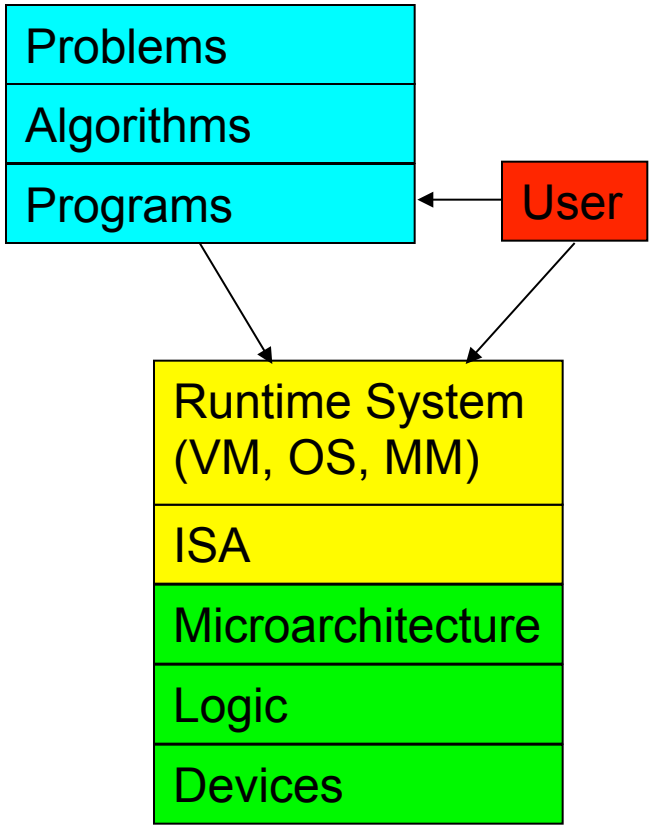
Meza, Chang, Yoon, Mutlu, Ranganathan, "Enabling Efficient and Scalable Hybrid Memories," IEEE Comp. Arch. Letters, 2012.

Hybrid Main Memory: Research Topics

- Many research ideas from technology layer to algorithms layer

- Enabling NVM and hybrid memory
  - How to maximize performance?
  - How to maximize lifetime?
  - How to prevent denial of service?

- Exploiting emerging tecnologies
  - How to exploit non-volatility?
  - How to minimize energy consumption?
  - How to minimize cost?
  - How to exploit NVM on chip?



Security Challenges of Emerging Technologies

1. Limited endurance → Wearout attacks

2. Non-volatility → Data persists in memory after powerdown
   → Easy retrieval of privileged or private information

3. Multiple bits per cell → Information leakage (via side channel)

Securing Emerging Memory Technologies

1. Limited endurance → Wearout attacks
   - Better architecting of memory chips to absorb writes
   - Hybrid memory system management
   - Online wearout attack detection

2. Non-volatility → Data persists in memory after powerdown
   → Easy retrieval of privileged or private information
   - Efficient encryption/decryption of whole main memory
   - Hybrid memory system management

3. Multiple bits per cell → Information leakage (via side channel)
   - System design to hide side channel information