Computer Architecture: Main Memory (Part II)

Prof. Onur Mutlu

Carnegie Mellon University

Main Memory Lectures

- These slides are from the Scalable Memory Systems course taught at ACACES 2013 (July 15-19, 2013)

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

- This is the second lecture:
  - Lecture 2 (July 16, 2013): DRAM Basics and DRAM Scaling: New DRAM Architectures I (pptx) (pdf)

Scalable Many-Core Memory Systems Lecture 2, 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 16, 2013

Carnegie Mellon

SAFARI

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

Review: 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 Power Management

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

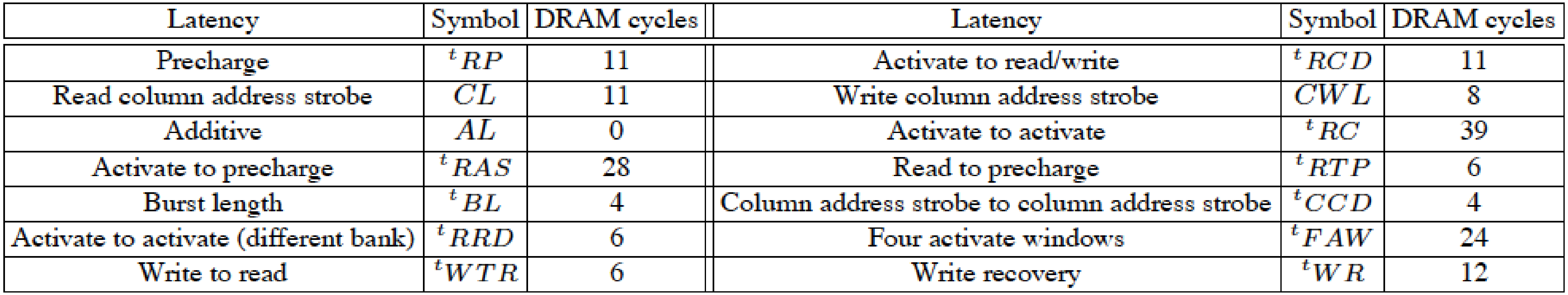
- Power states
  - Active (highest power)
  - All banks idle
  - Power-down
  - Self-refresh (lowest power)

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

Review: 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?

Review: 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.

Review: 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.

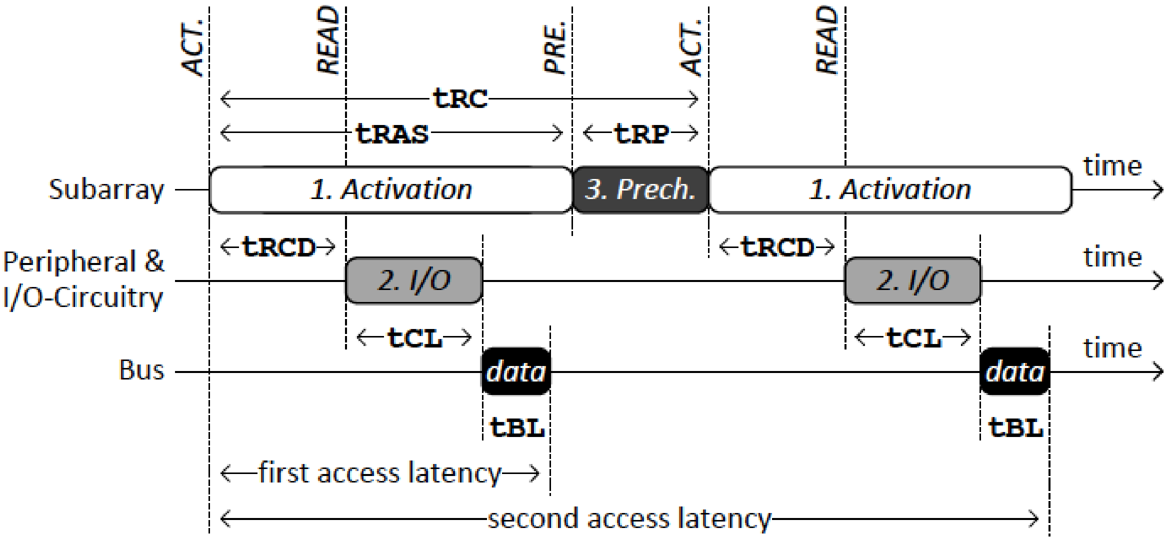


Figure 5. Three Phases of DRAM Access

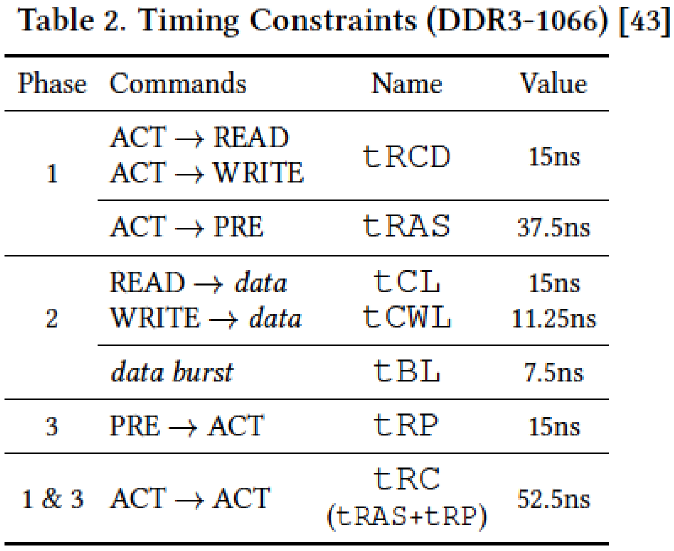


Table 2. Timing Constraints (DDR3-1066) [43]

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

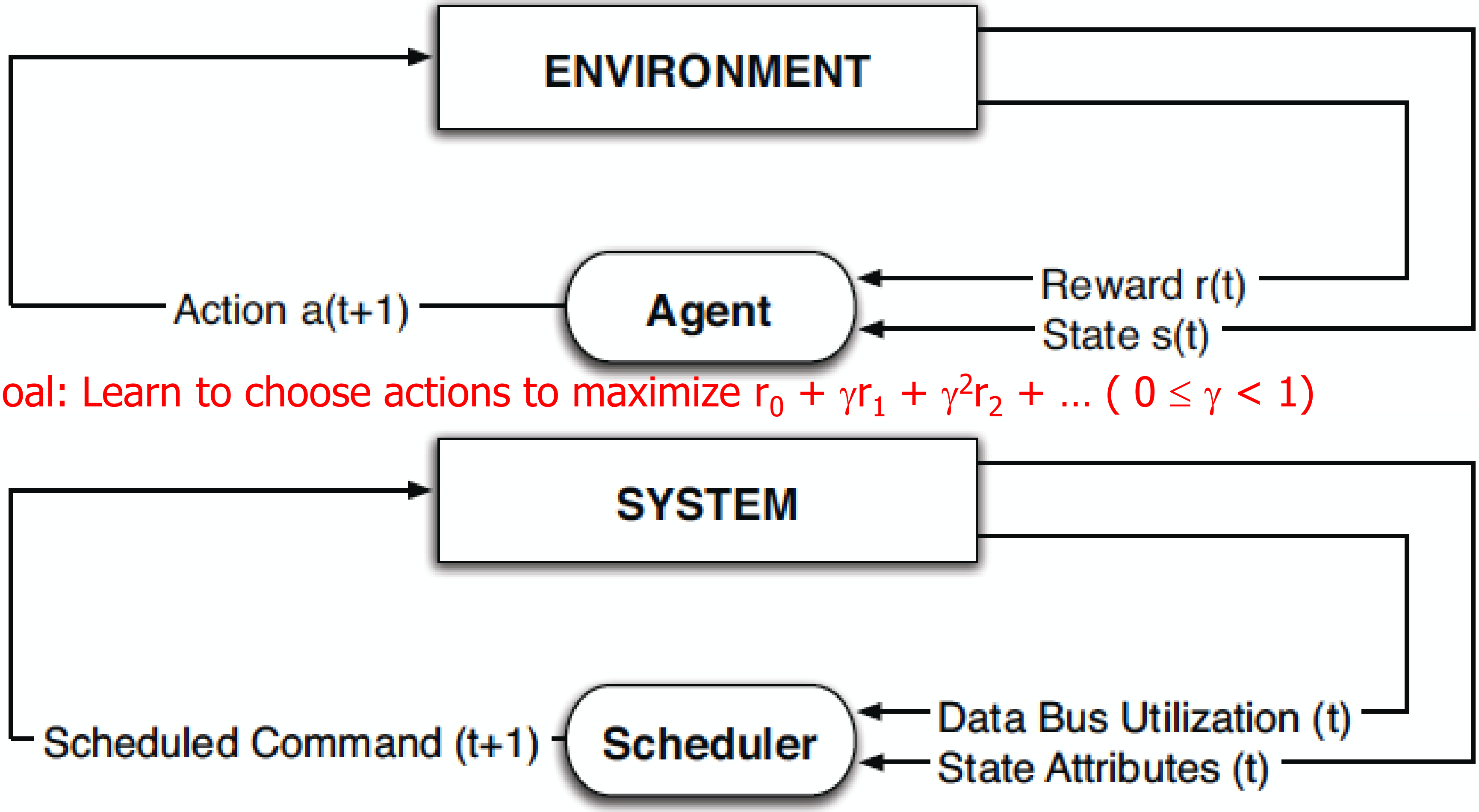
- Idea: Design a memory controller that adapts its scheduling policy decisions to workload behavior and system conditions using machine learning.

- 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

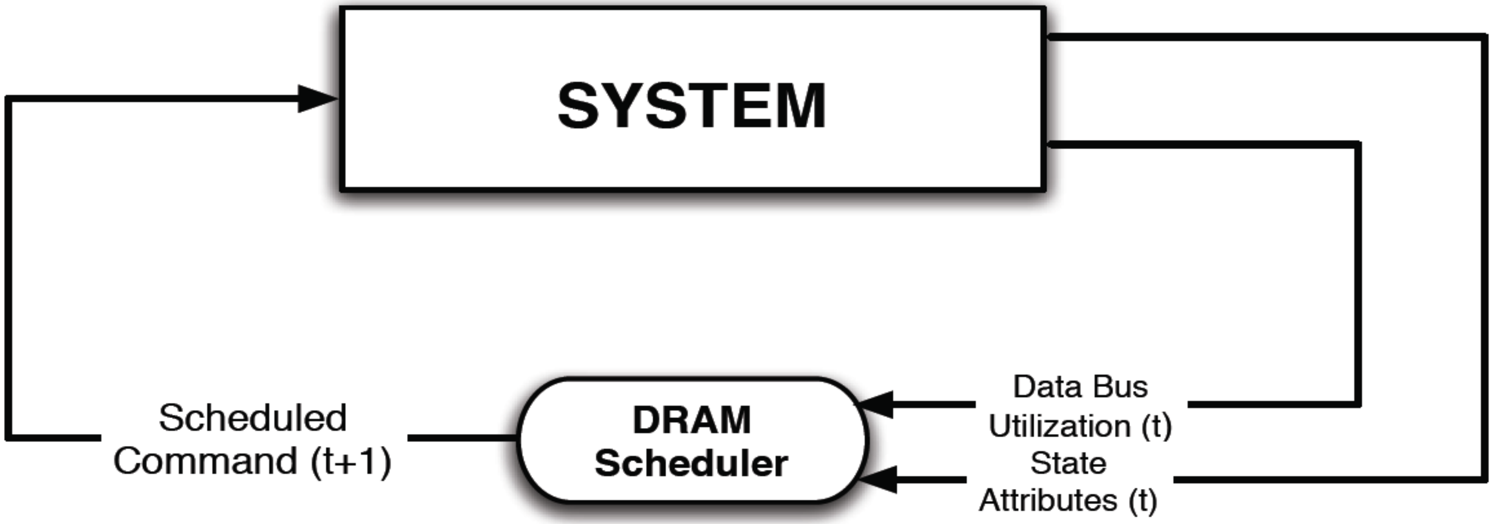


Goal: Learn to choose actions to maximize $r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots$ ($0 \leq \gamma < 1$)

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

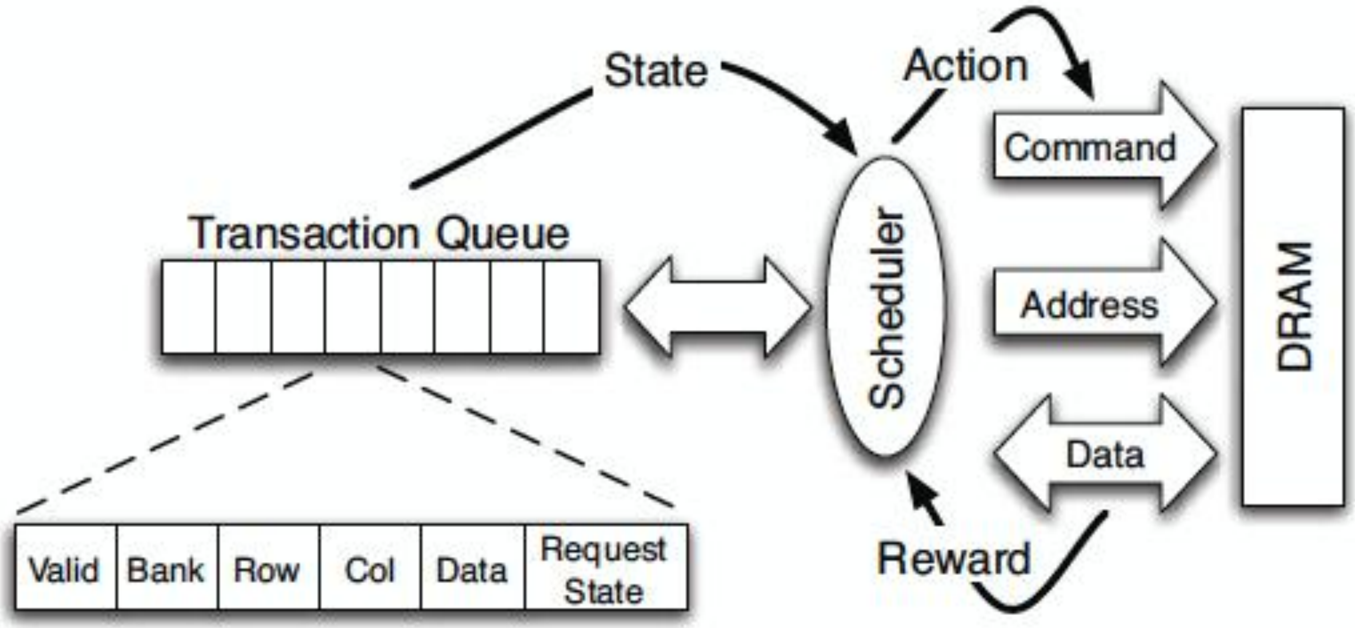


Figure 4: High-level overview of an RL-based scheduler.

States, Actions, Rewards

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

- 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

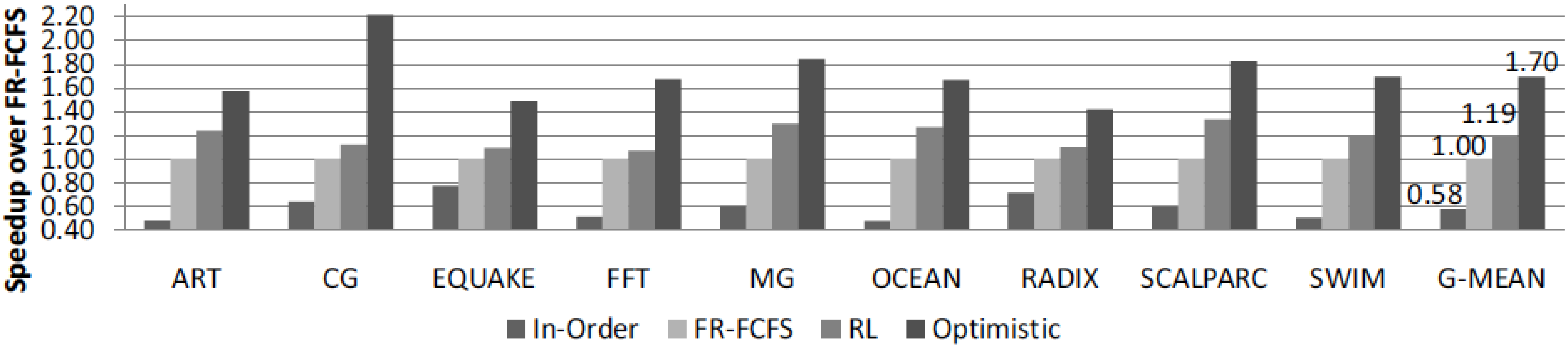


Figure 7: Performance comparison of in-order, FR-FCFS, RL-based, and optimistic memory controllers

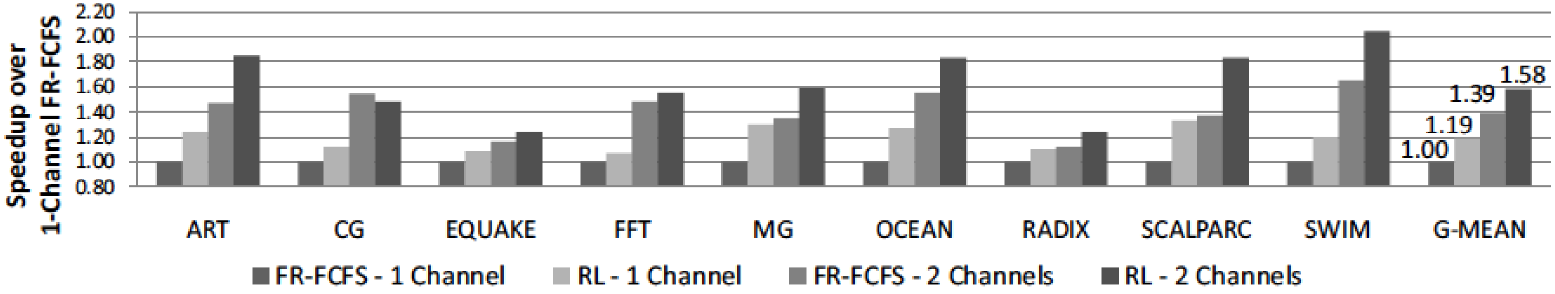


Figure 15: Performance comparison of FR-FCFS and RL-based memory controllers on systems with 6.4GB/s and 12.8GB/s peak DRAM bandwidth

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 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

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

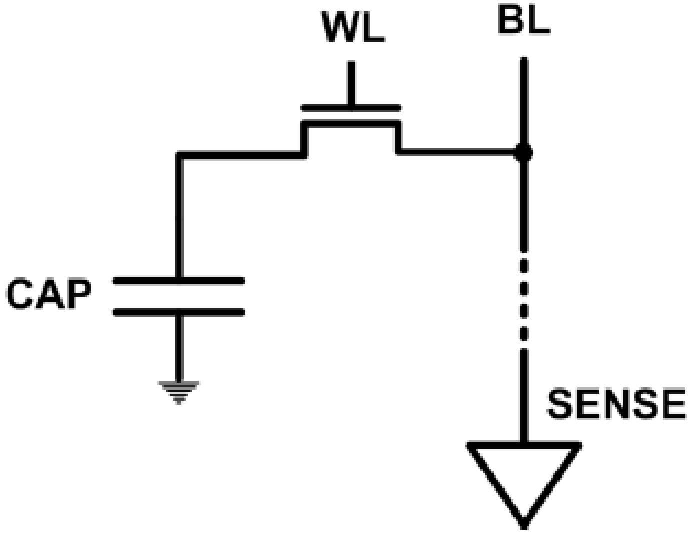
- DRAM technology scaling is ending
  - ITRS projects DRAM will not scale easily below X nm
  - Scaling has provided many benefits:
    - higher capacity (density), lower cost, lower energy

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

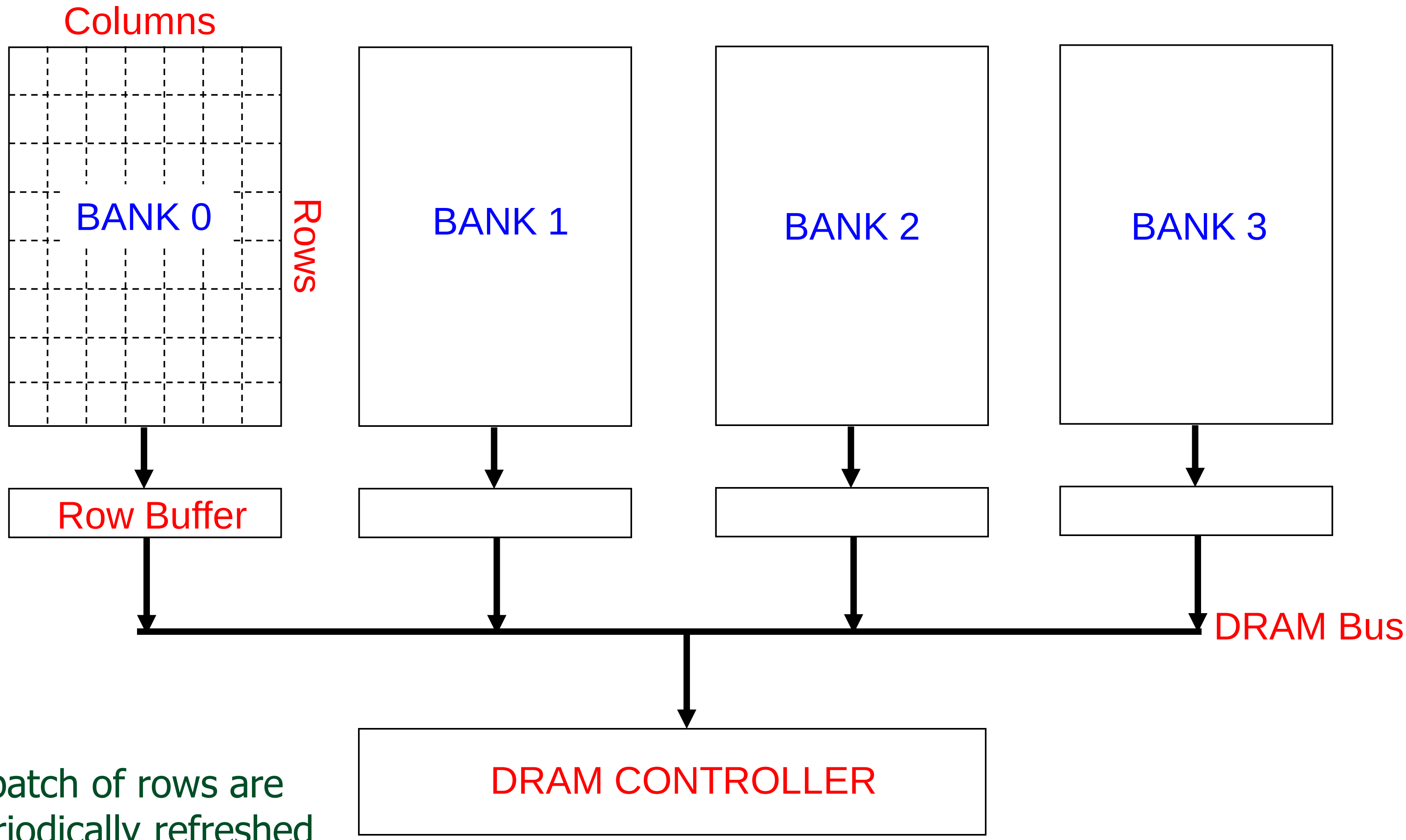
  -- Energy consumption: Each refresh consumes energy

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

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

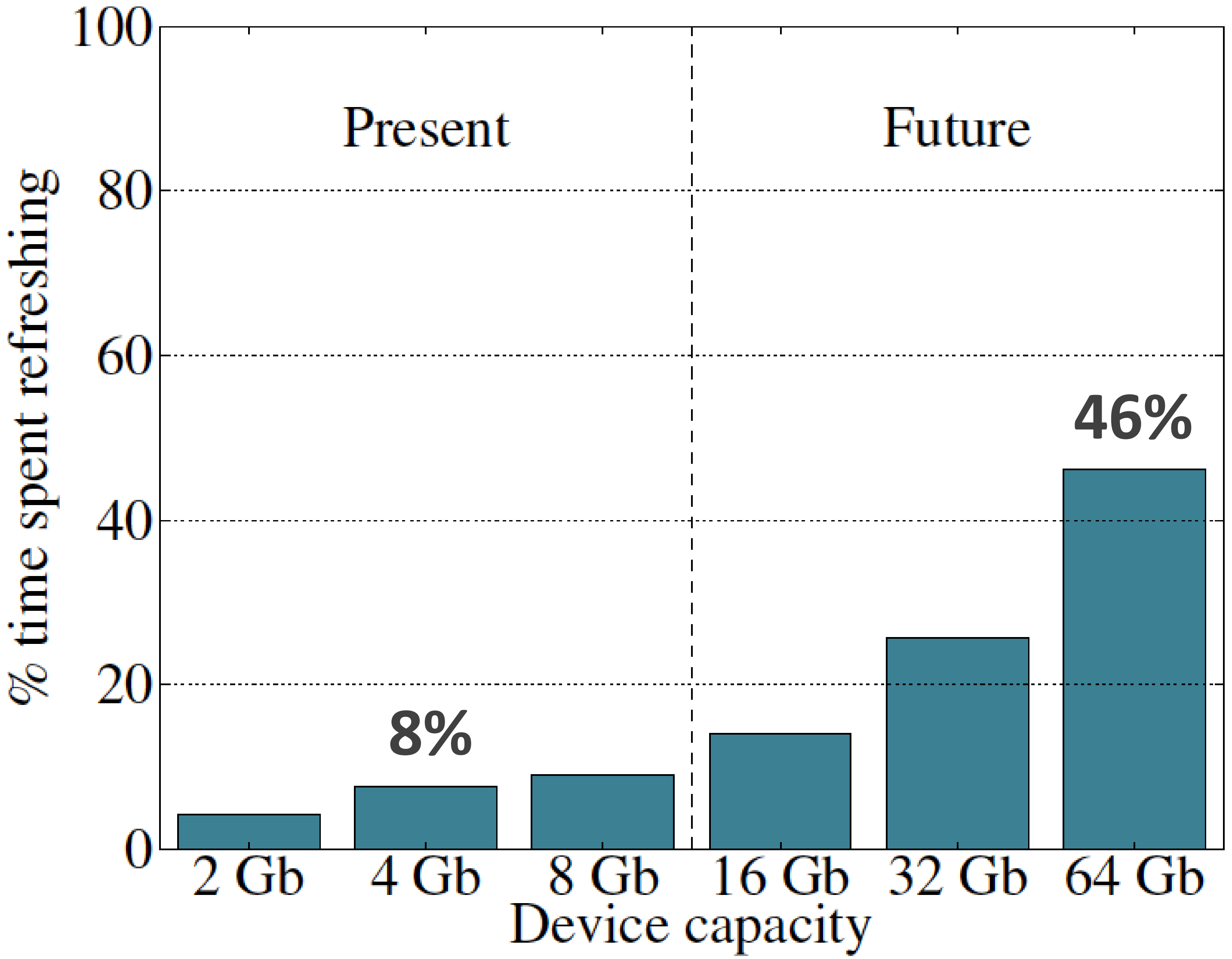
  -- Refresh rate limits DRAM density scaling

Refresh Today: Auto Refresh

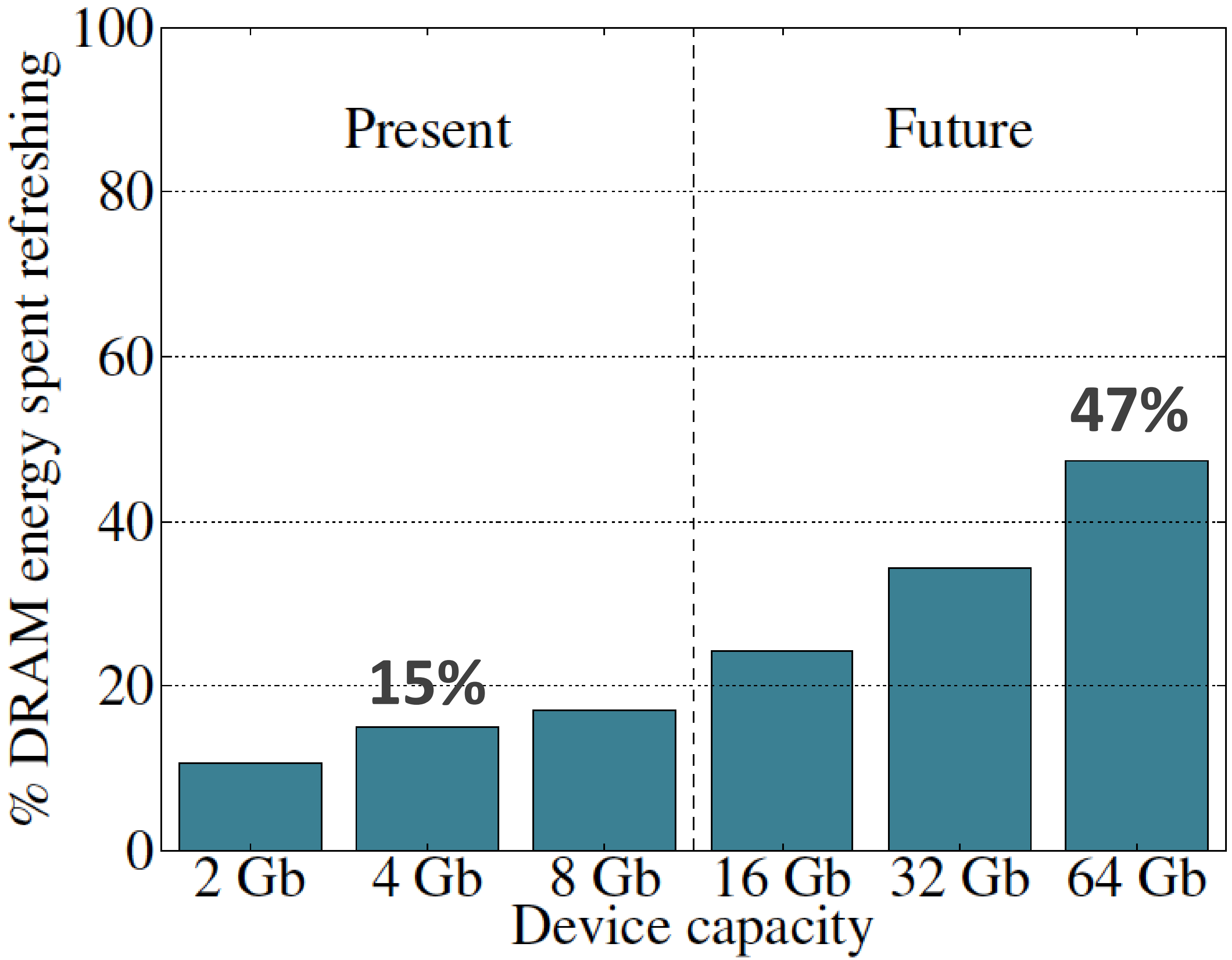


A batch of rows are periodically refreshed via the auto-refresh command

Refresh Overhead: Performance

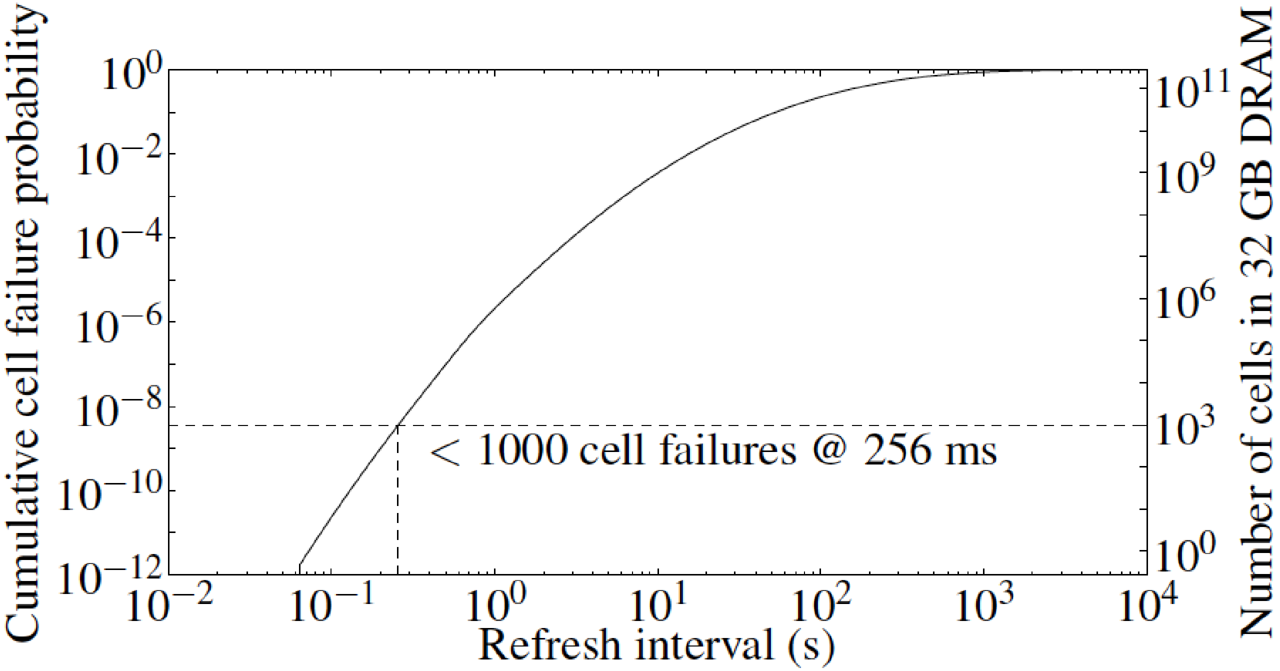


Refresh Overhead: Energy



Problem with Conventional Refresh

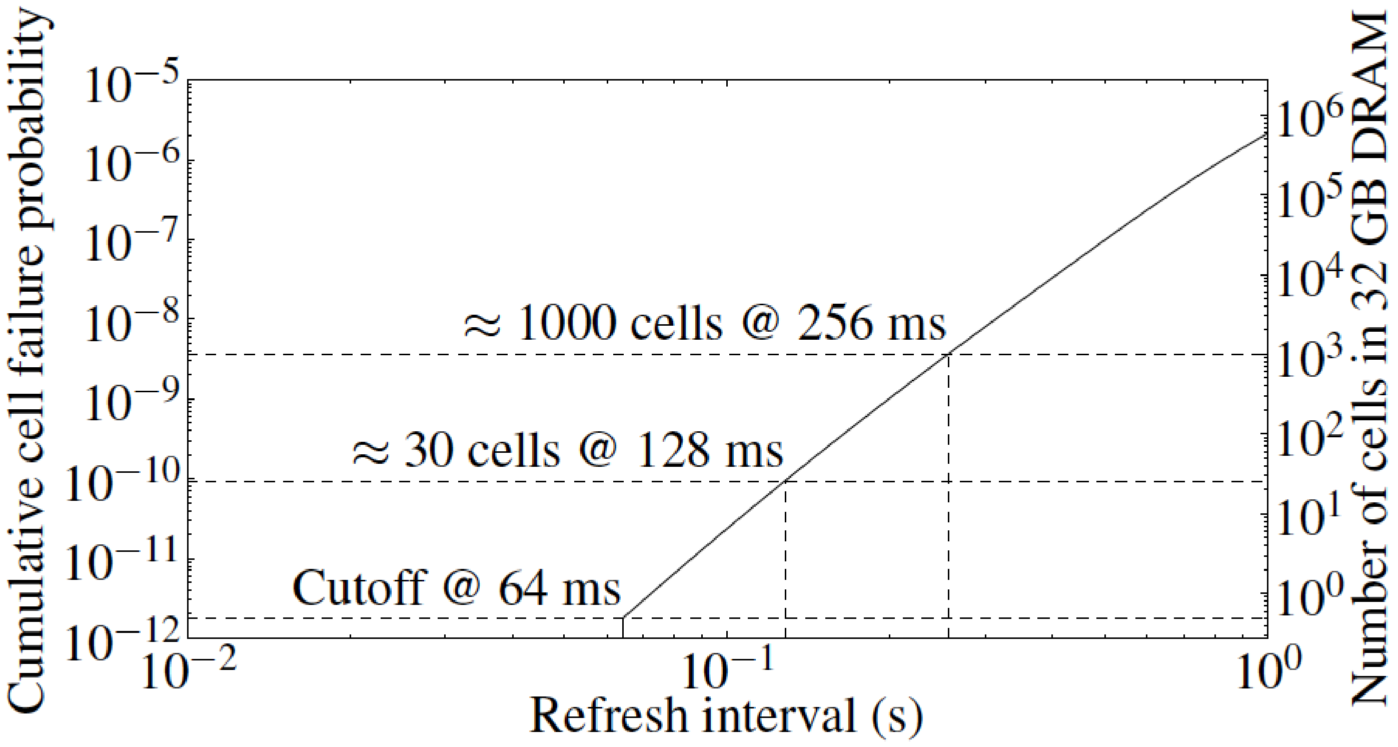
- Today: Every row is refreshed at the same rate



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

Retention Time of DRAM Rows

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



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

Reducing DRAM Refresh Operations

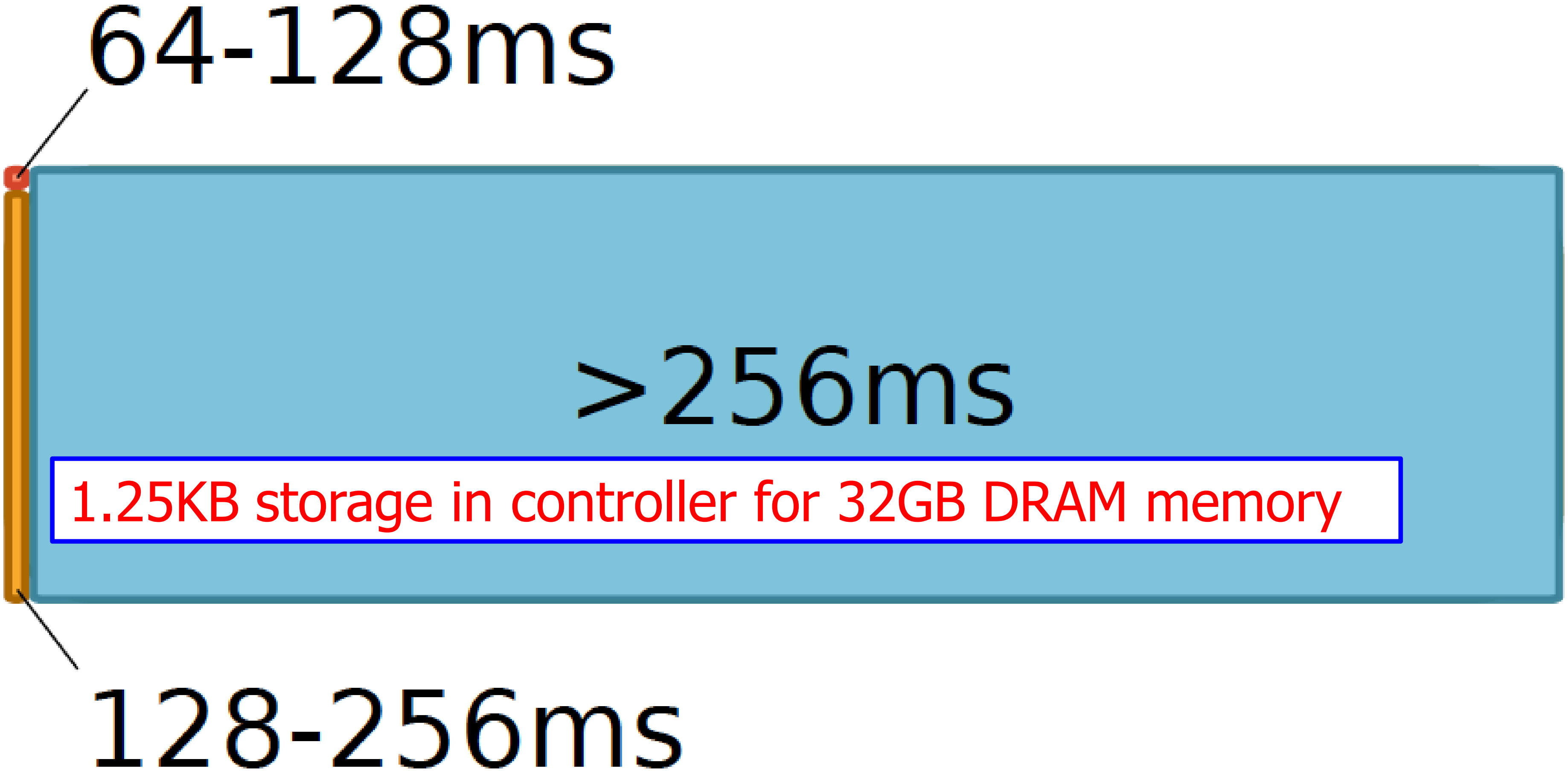
- Idea: Identify the retention time of different rows and refresh each row at the frequency it needs to be refreshed

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

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

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

RAIDR: Mechanism



1.25KB storage in controller for 32GB DRAM memory

bins at different rates

→ probe Bloom Filters to determine refresh rate of a row

1. Profiling

To profile a row:

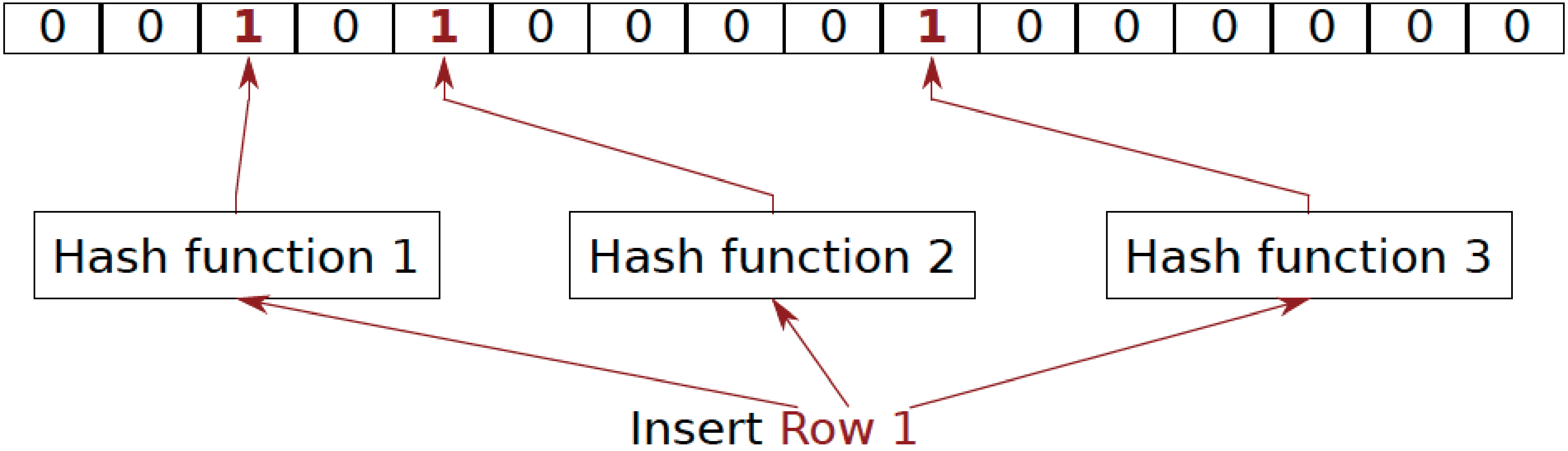
1. Write data to the row
2. Prevent it from being refreshed
3. Measure time before data corruption

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

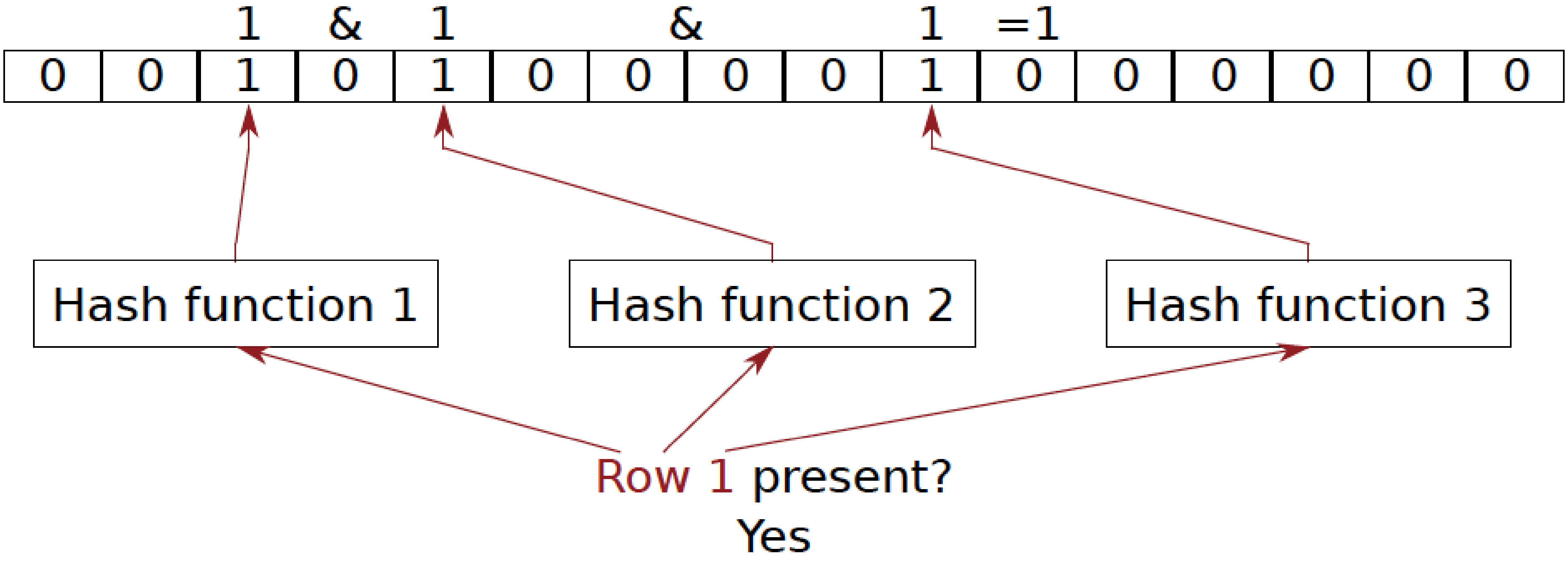
Hash function 2

Hash function 3

Insert Row 1

Bloom Filter Operation Example

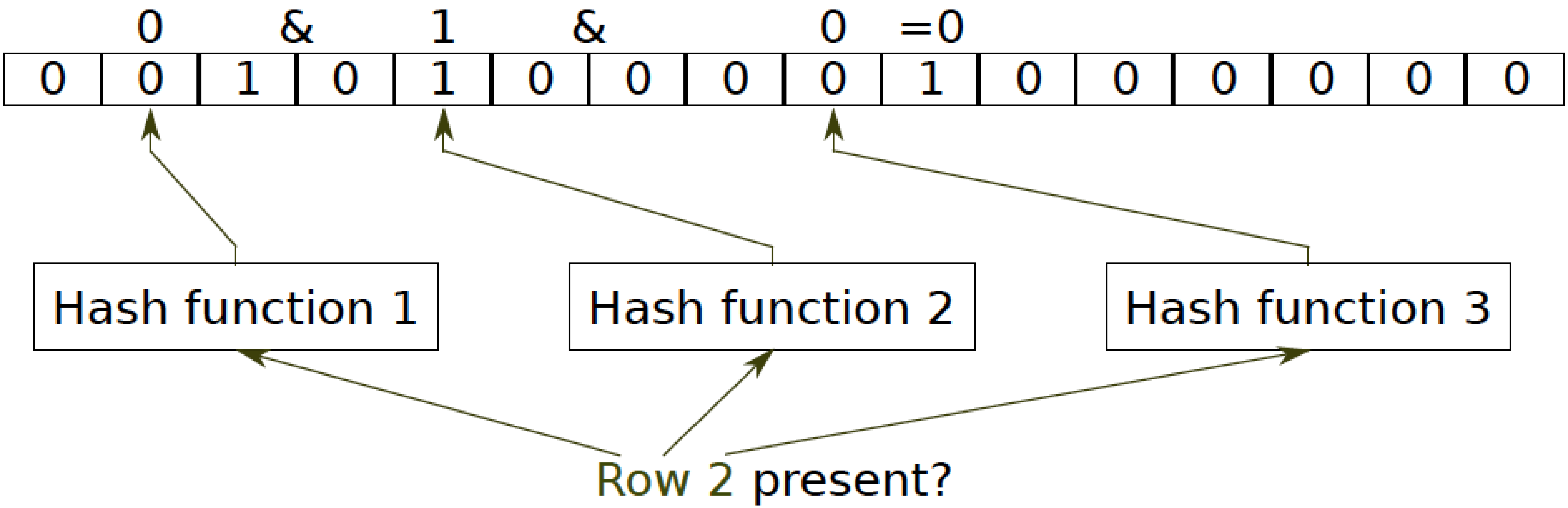
Example with 64-128ms bin:



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

Bloom Filter Operation Example

Example with 64–128ms bin:



0 & 1 & 0 =0

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

Hash function 1

Hash function 2

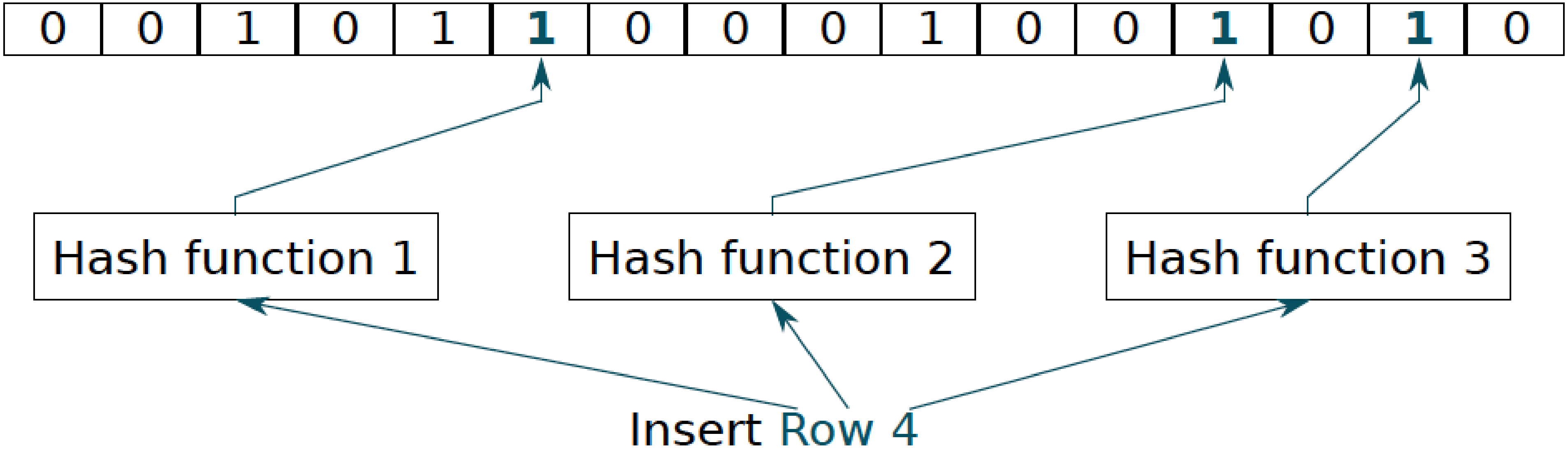
Hash function 3

Row 2 present?

No

Bloom Filter Operation Example

Example with 64–128ms bin:



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

Hash function 1

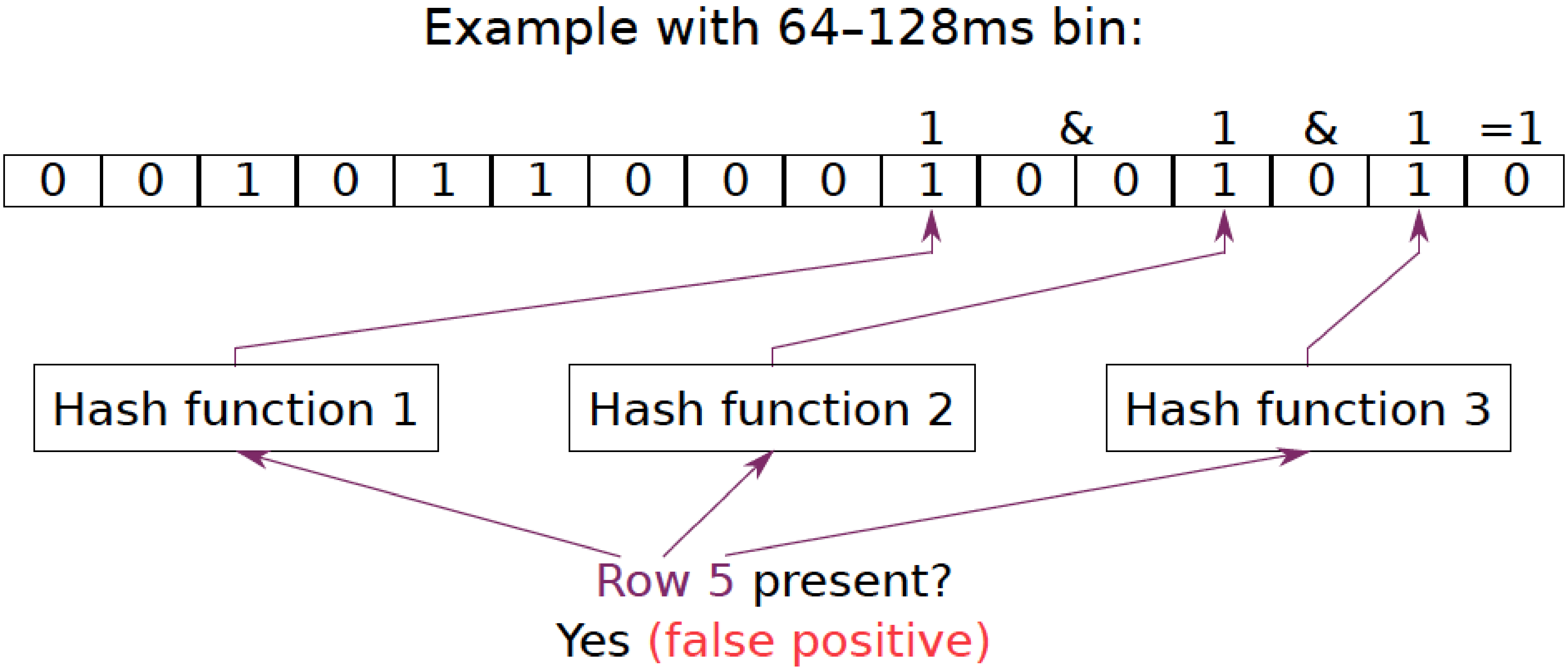
Hash function 2

Hash function 3

Insert Row 4

SAFARI

Bloom Filter Operation Example



Example with 64–128ms bin:

1 & 1 & 1 =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

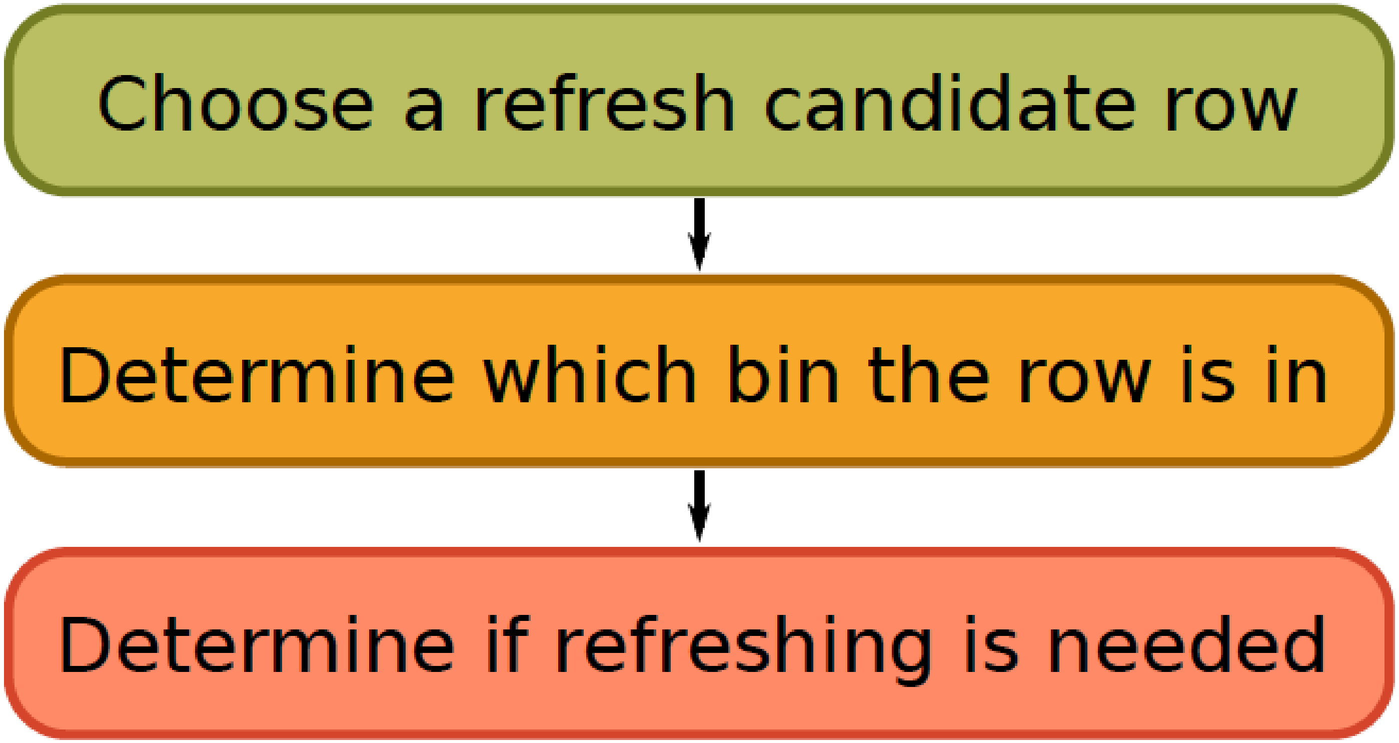
- False positives: a row may be declared present in the Bloom filter even if it was never inserted
  - Not a problem: Refresh some rows more frequently than needed

- 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)

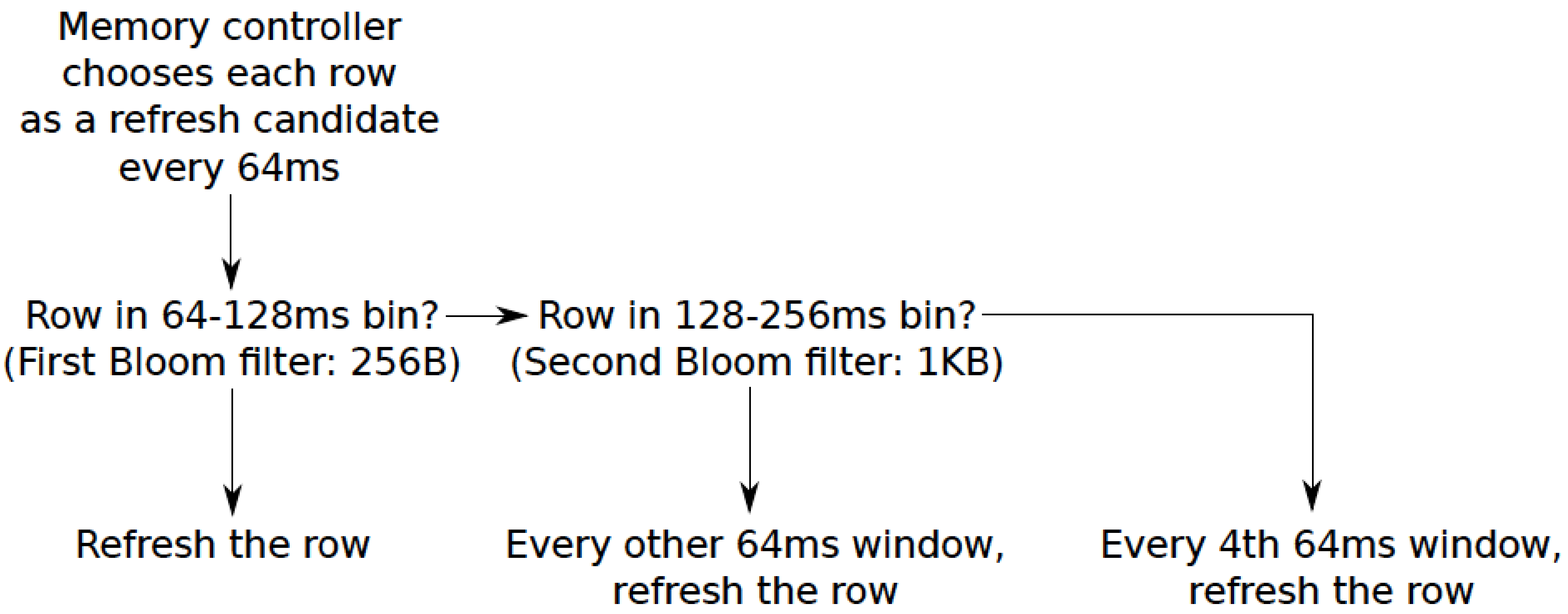


Choose a refresh candidate row

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? (First Bloom filter: 256B)
Row in 128-256ms bin? (Second Bloom filter: 1KB)
Refresh the row
Every other 64ms window, refresh the row
Every 4th 64ms window, refresh the row

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

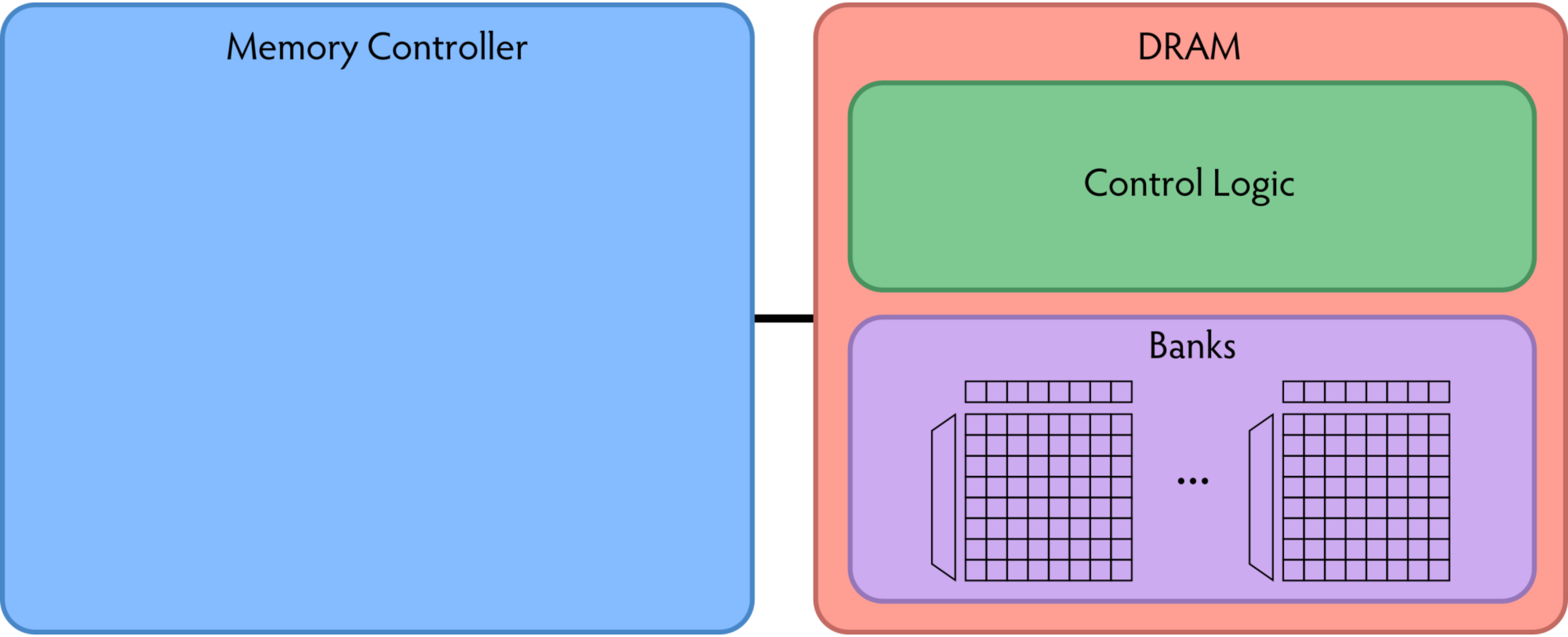
Tolerating Temperature Changes

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

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

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

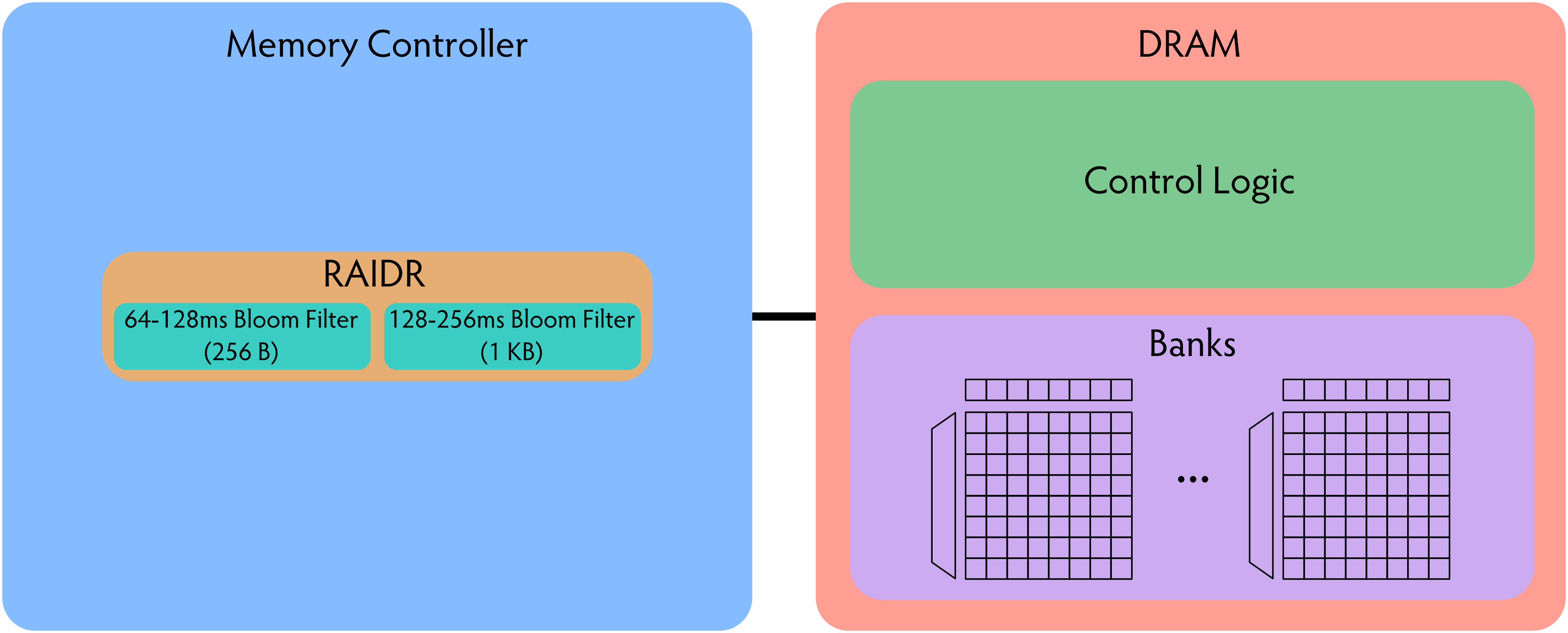
RAIDR: Baseline Design



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

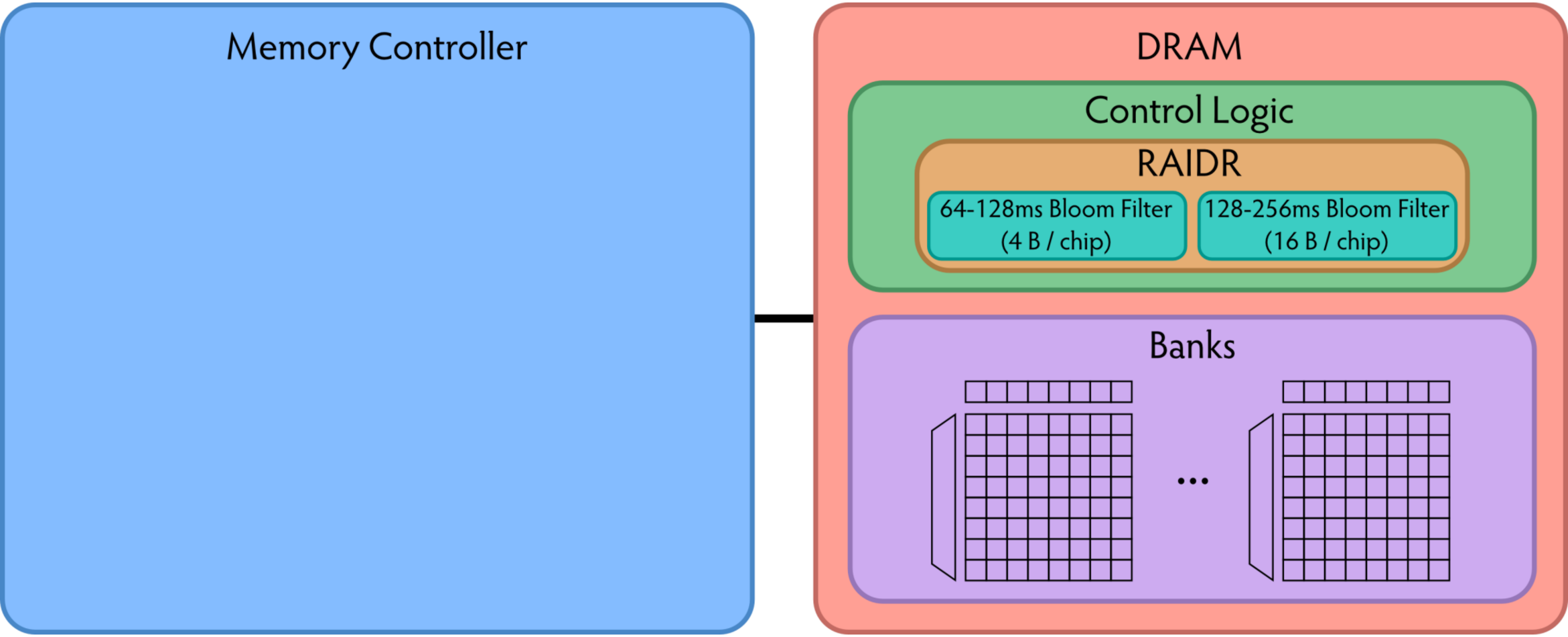
RAIDR can be implemented in either the controller or DRAM

RAIDR in Memory Controller: Option 1



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

RAIDR in DRAM Chip: Option 2



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

RAIDR Results

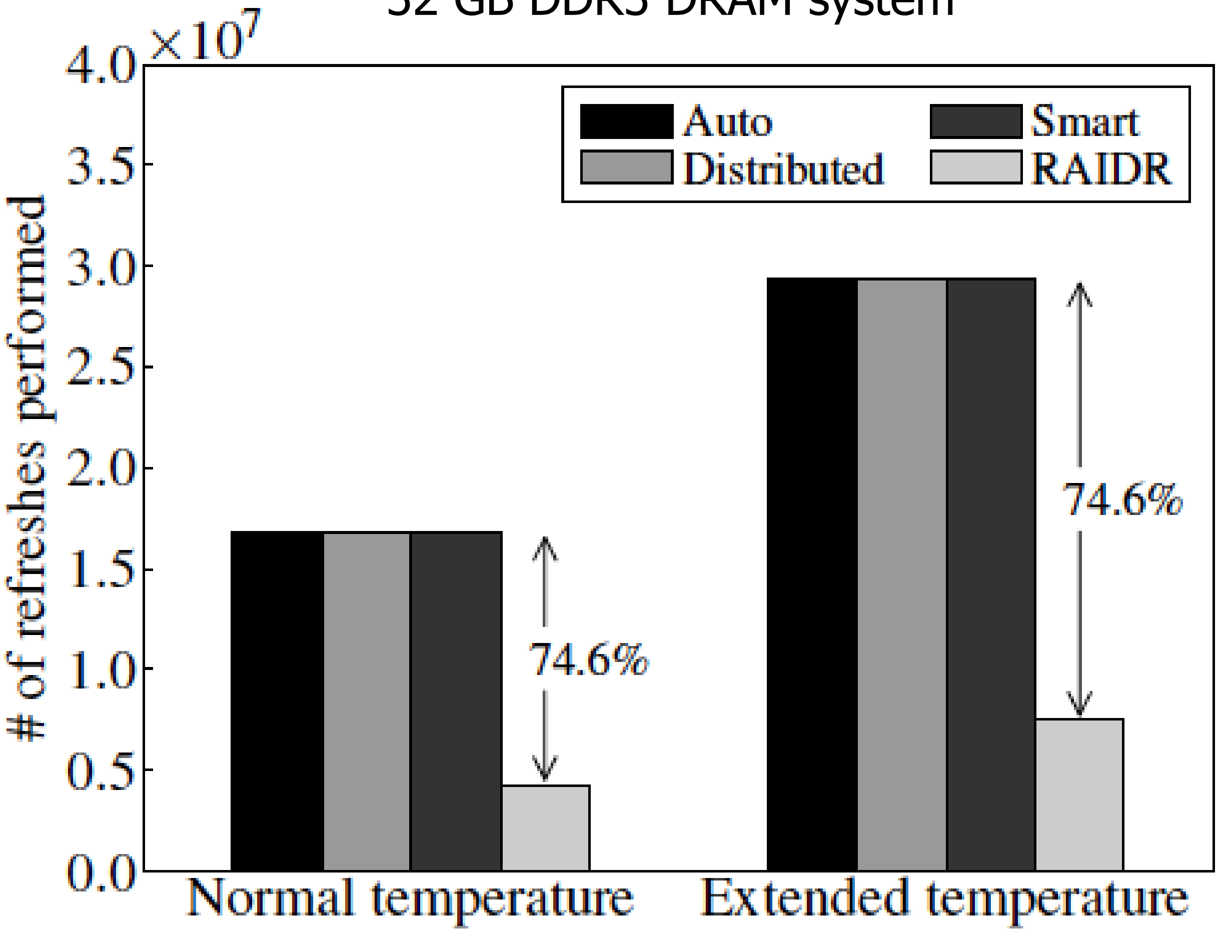
- Baseline:
  - 32 GB DDR3 DRAM system (8 cores, 512KB cache/core)
  - 64ms refresh interval for all rows

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

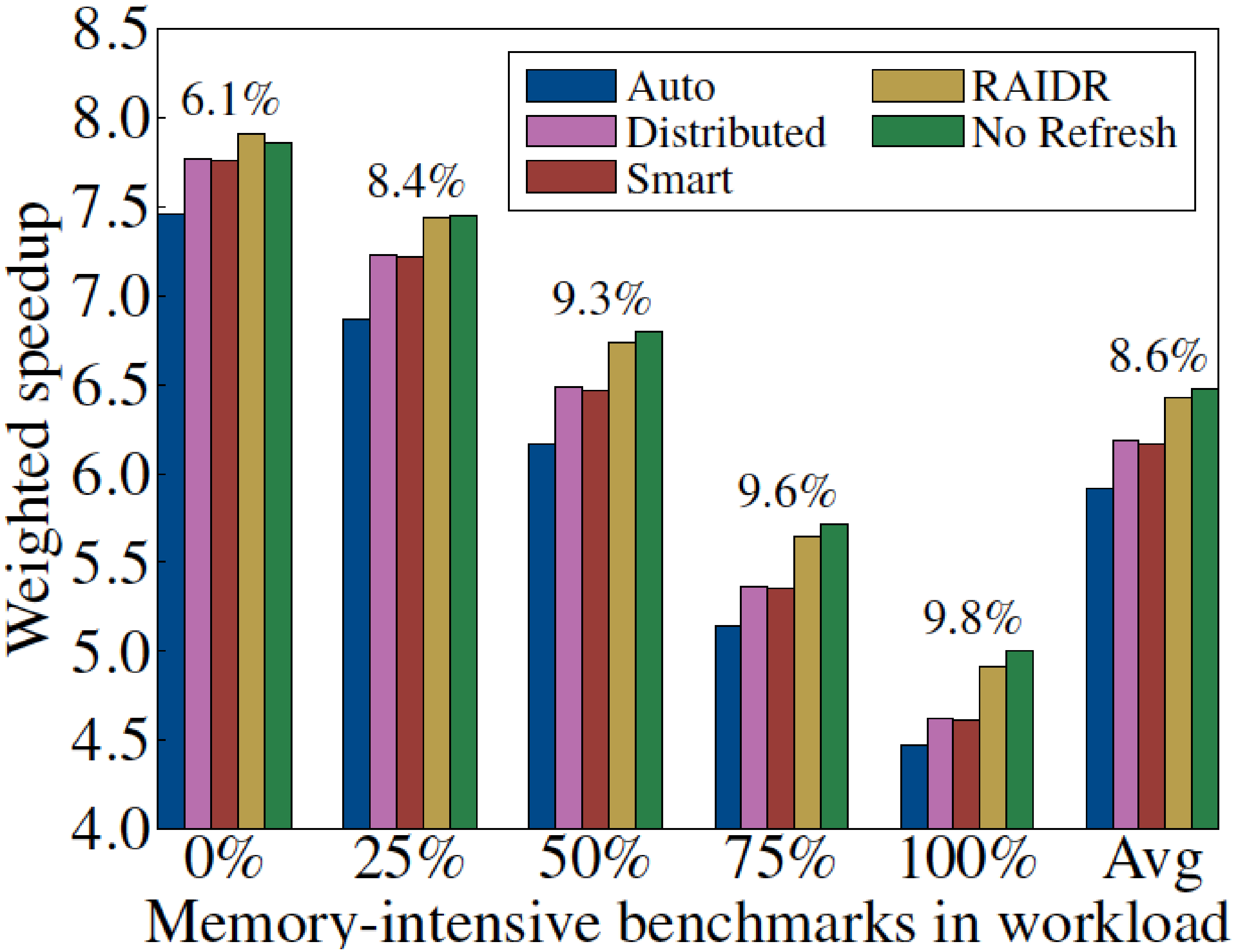
- Results on SPEC CPU2006, TPC-C, TPC-H benchmarks
  - 74.6% refresh reduction
  - ~16%/20% DRAM dynamic/idle power reduction
  - ~9% performance improvement

RAIDR Refresh Reduction

32 GB DDR3 DRAM system

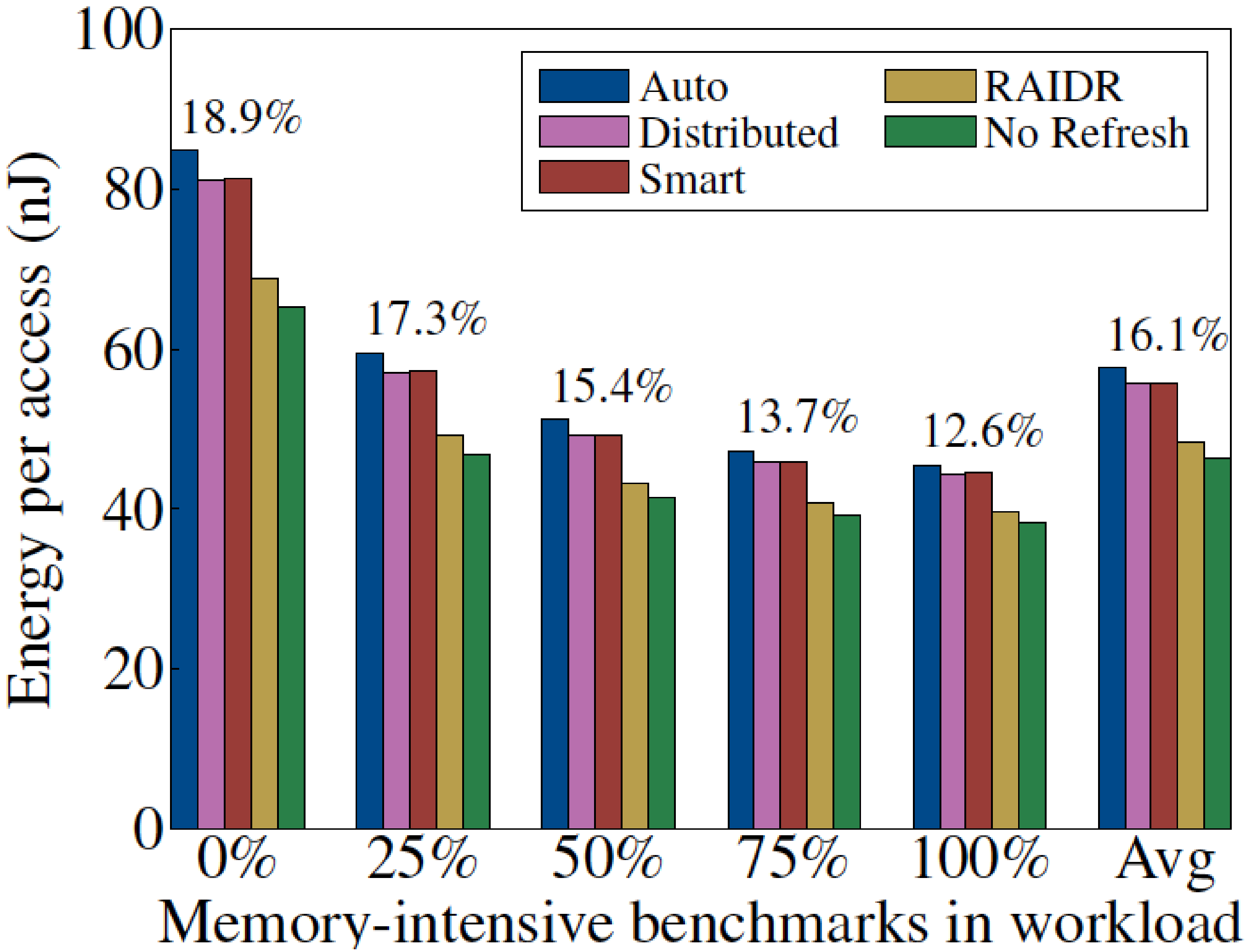


RAIDR: Performance



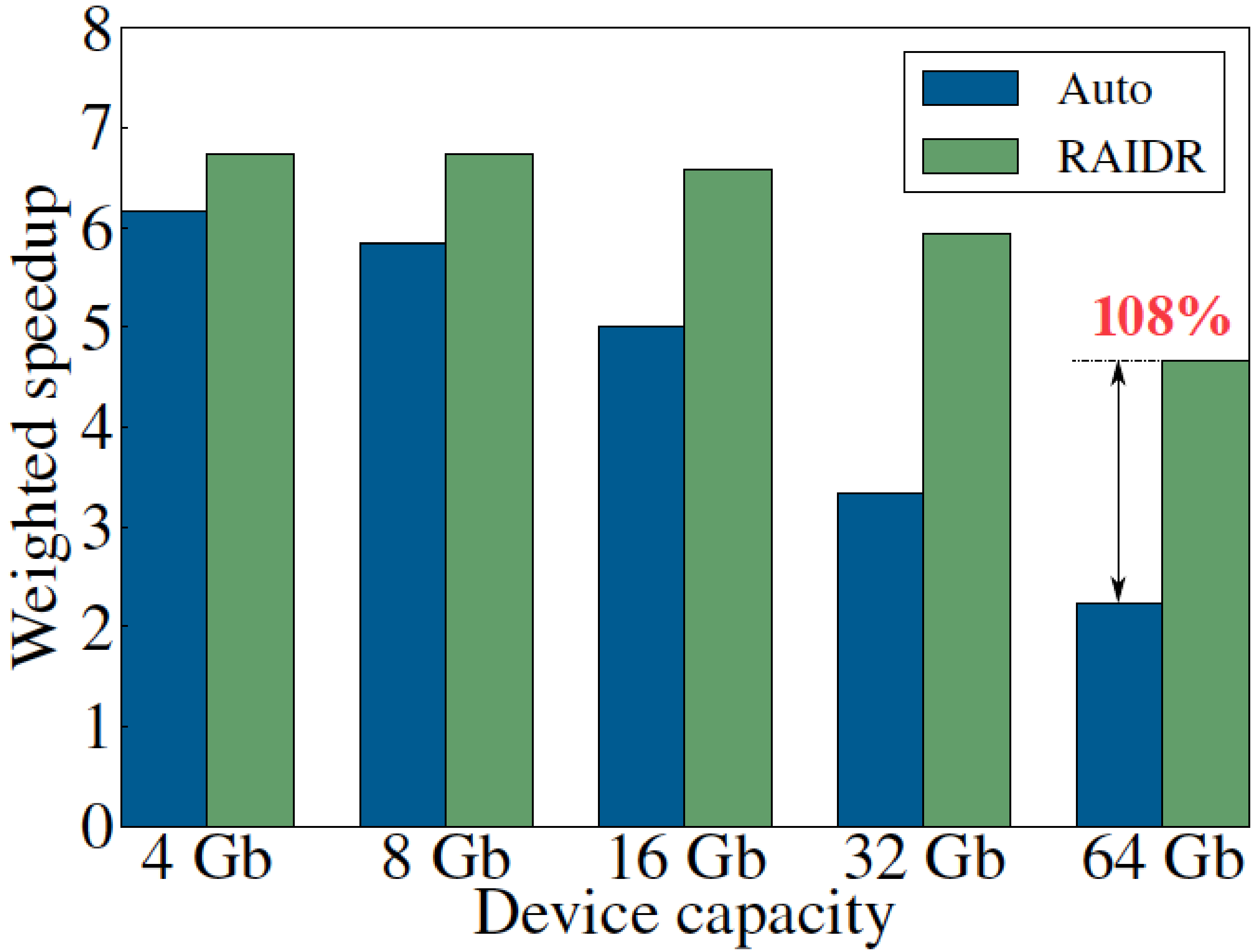
RAIDR performance benefits increase with workload's memory intensity

RAIDR: DRAM Energy Efficiency



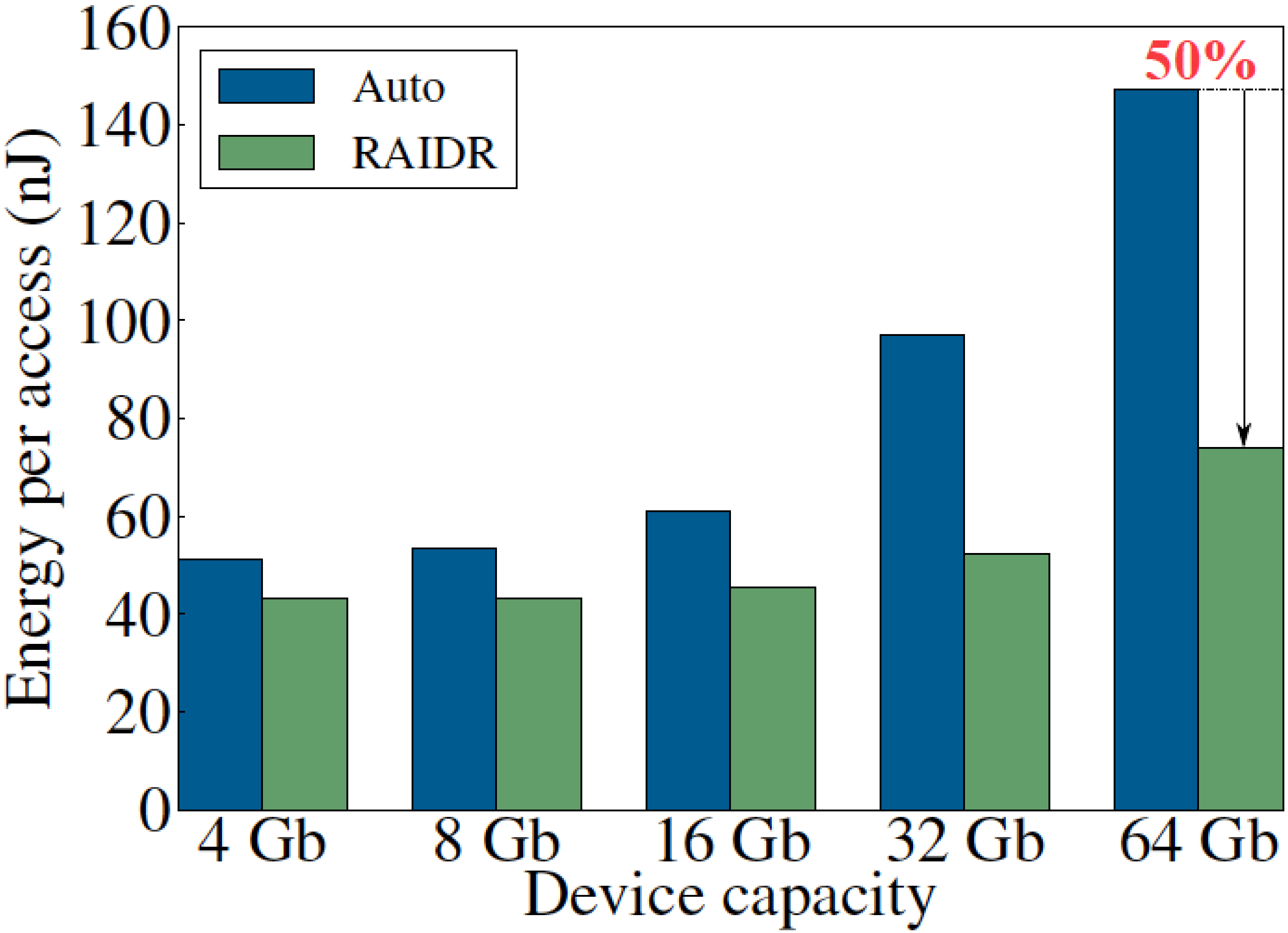
RAIDR energy benefits increase with memory idleness

DRAM Device Capacity Scaling: Performance



RAIDR performance benefits increase with DRAM chip capacity

DRAM Device Capacity Scaling: Energy



RAIDR energy benefits increase with DRAM chip capacity

RAIDR slides

SAFARI

More Readings Related to RAIDR

- 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)

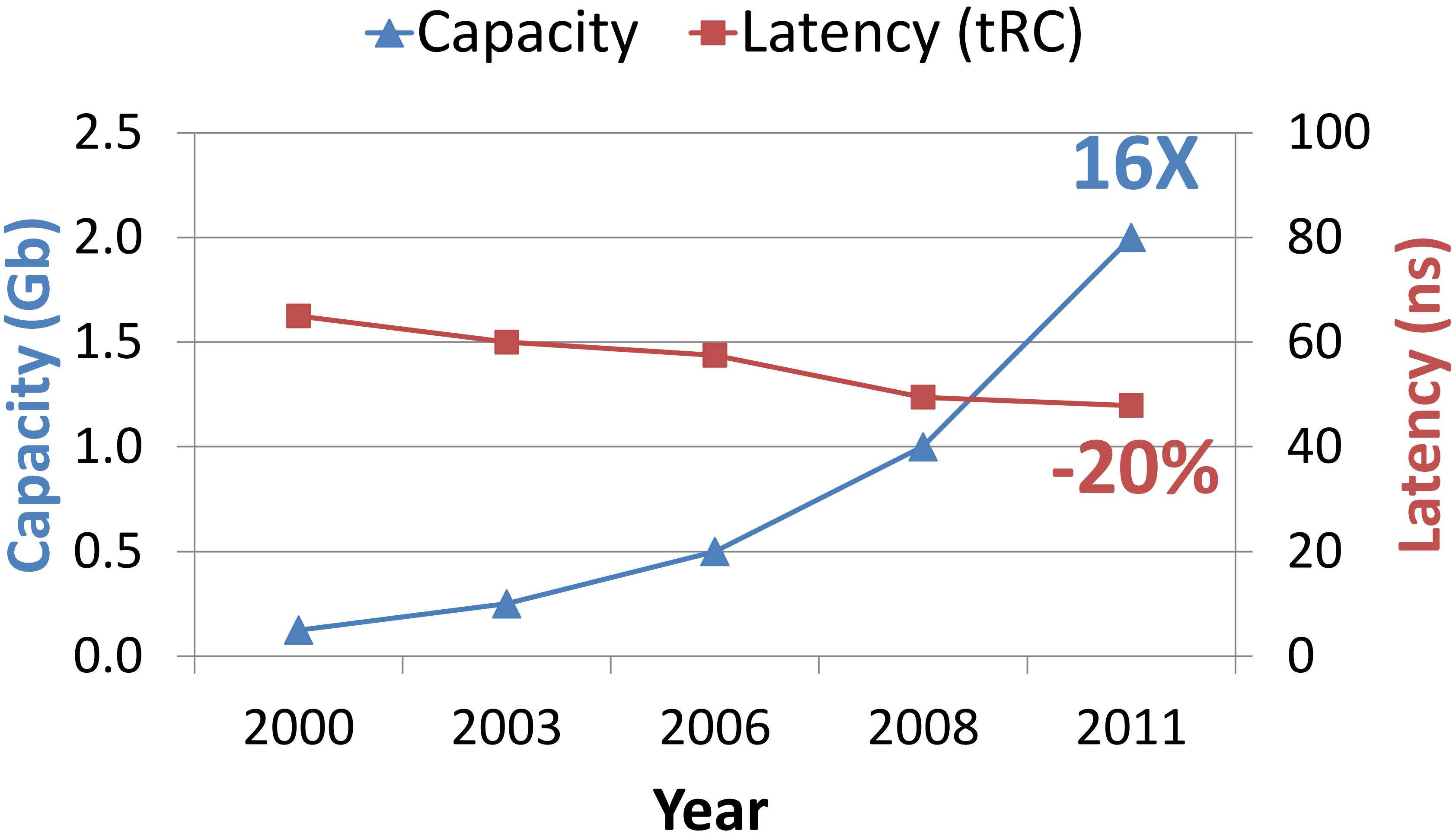
New DRAM Architectures

- 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

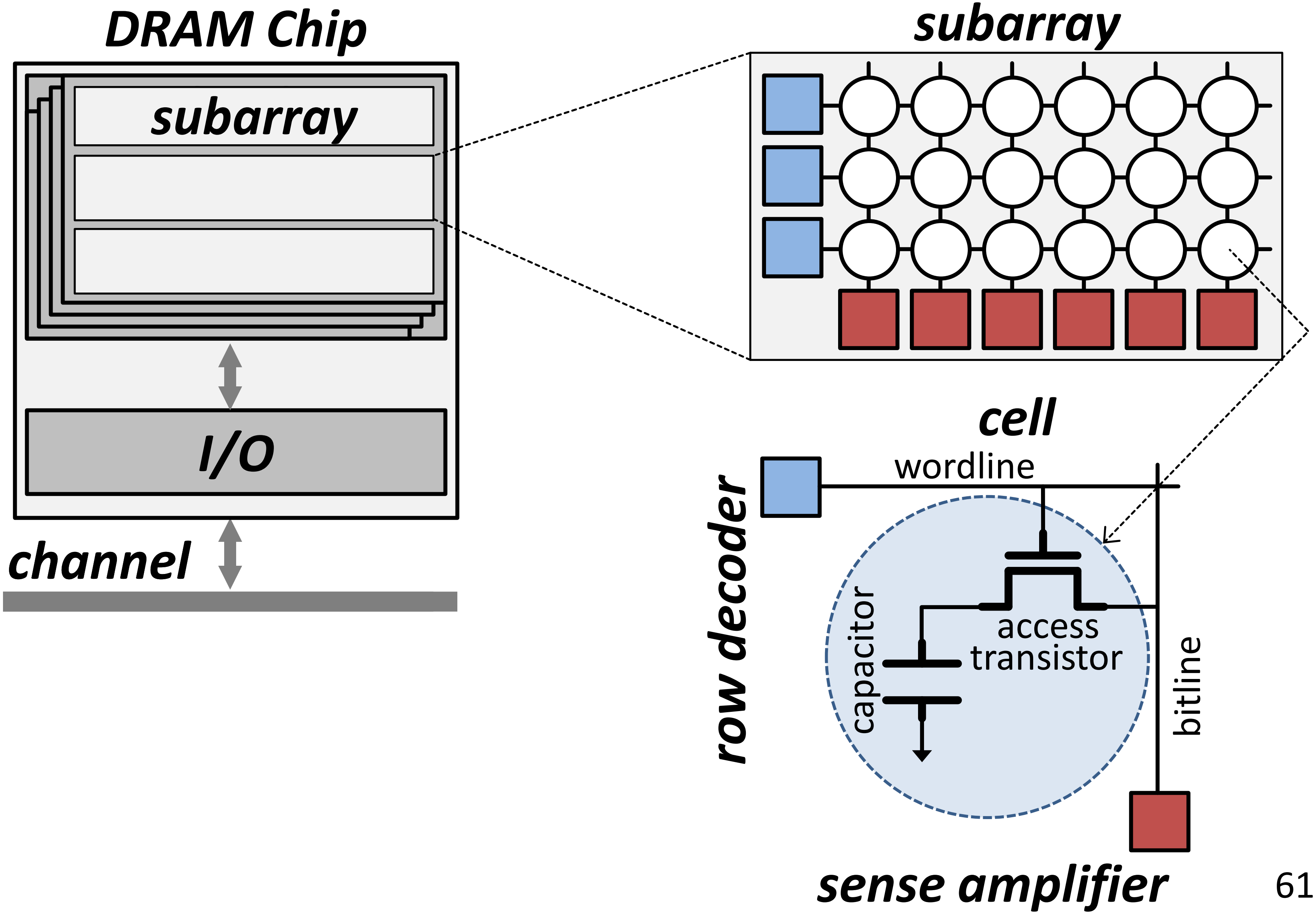
Donghyuk Lee, Yoongu Kim, Vivek Seshadri, Jamie Liu, Lavanya Subramanian, and Onur Mutlu,
"Tiered-Latency DRAM: A Low Latency and Low Cost DRAM Architecture"
19th International Symposium on High-Performance Computer Architecture (***HPCA***),
Shenzhen, China, February 2013. Slides (pptx)

Historical DRAM Latency-Capacity Trend

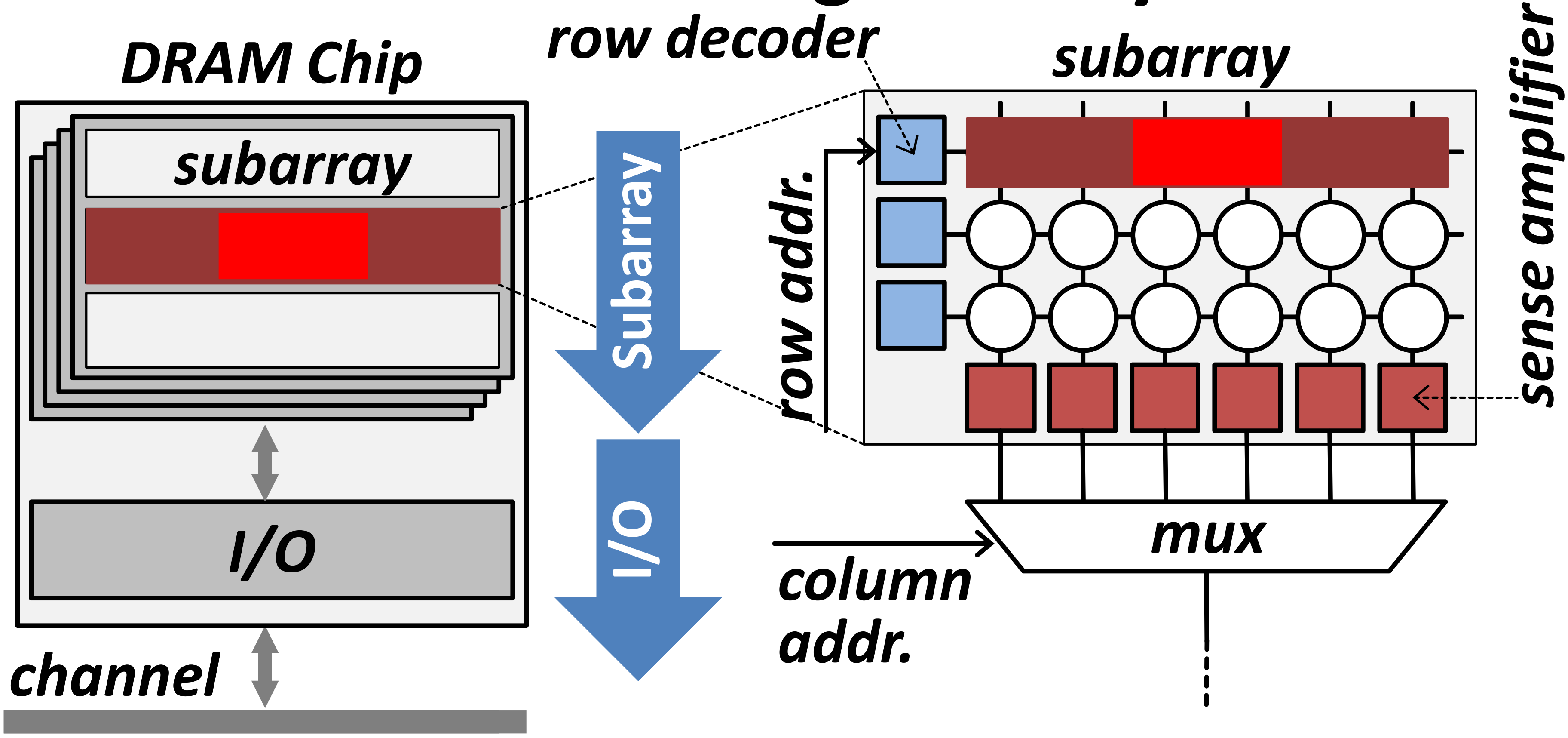


DRAM latency continues to be a critical bottleneck

What Causes the Long Latency?



What Causes the Long Latency?

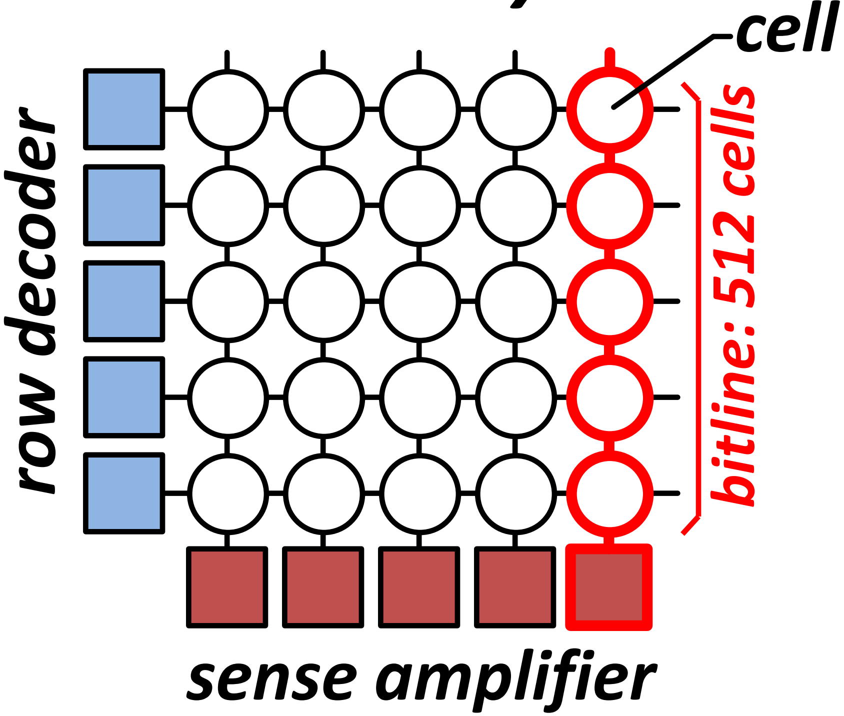


DRAM Latency = Subarray Latency + I/O Latency

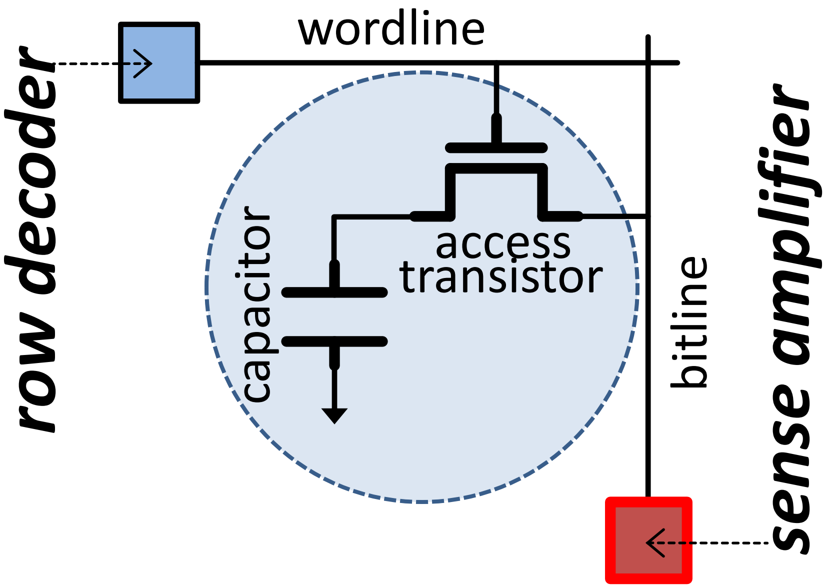
Dominant

Why is the Subarray So Slow?

Subarray



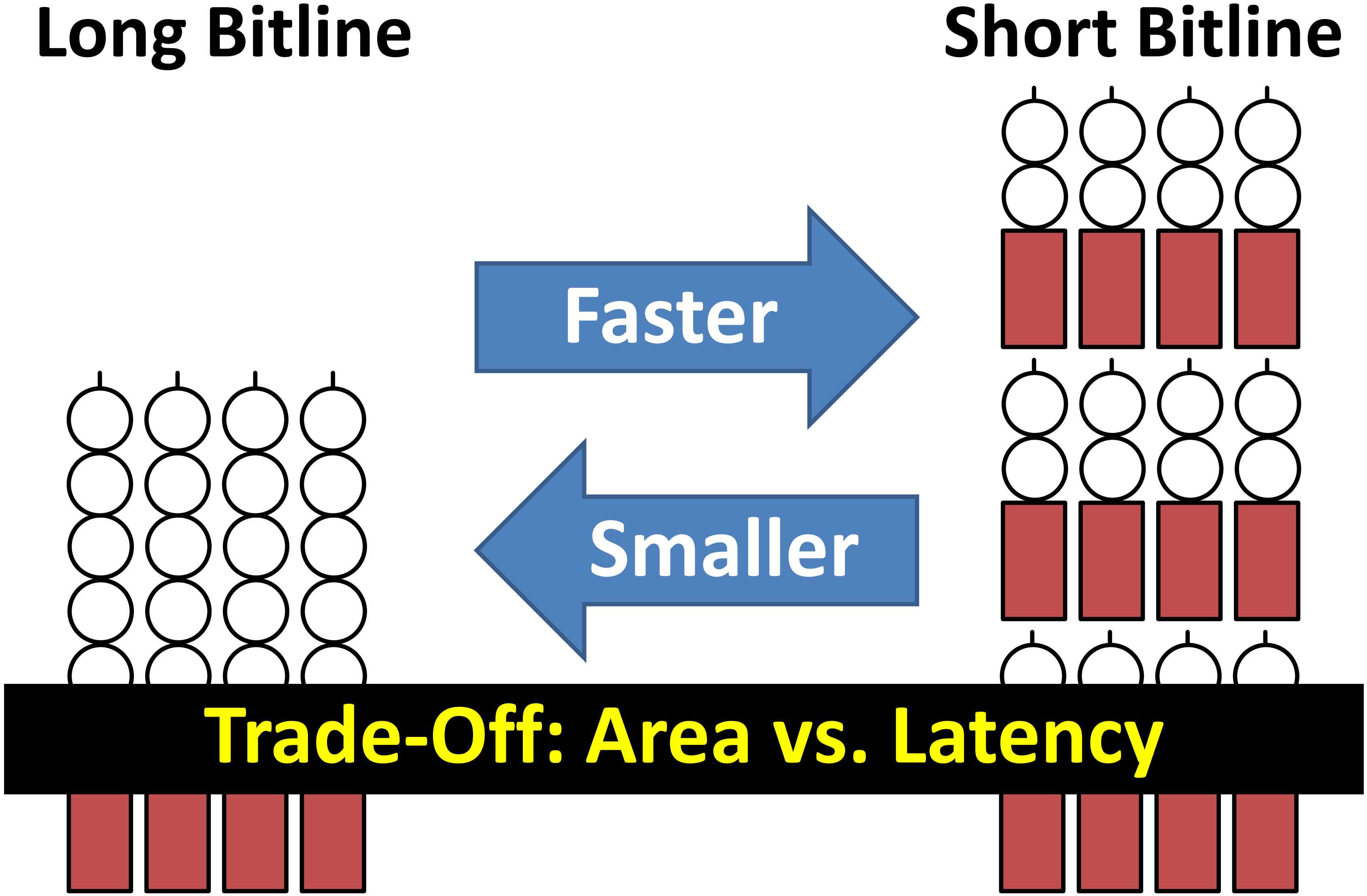
Cell



large sense amplifier

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

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



Long Bitline

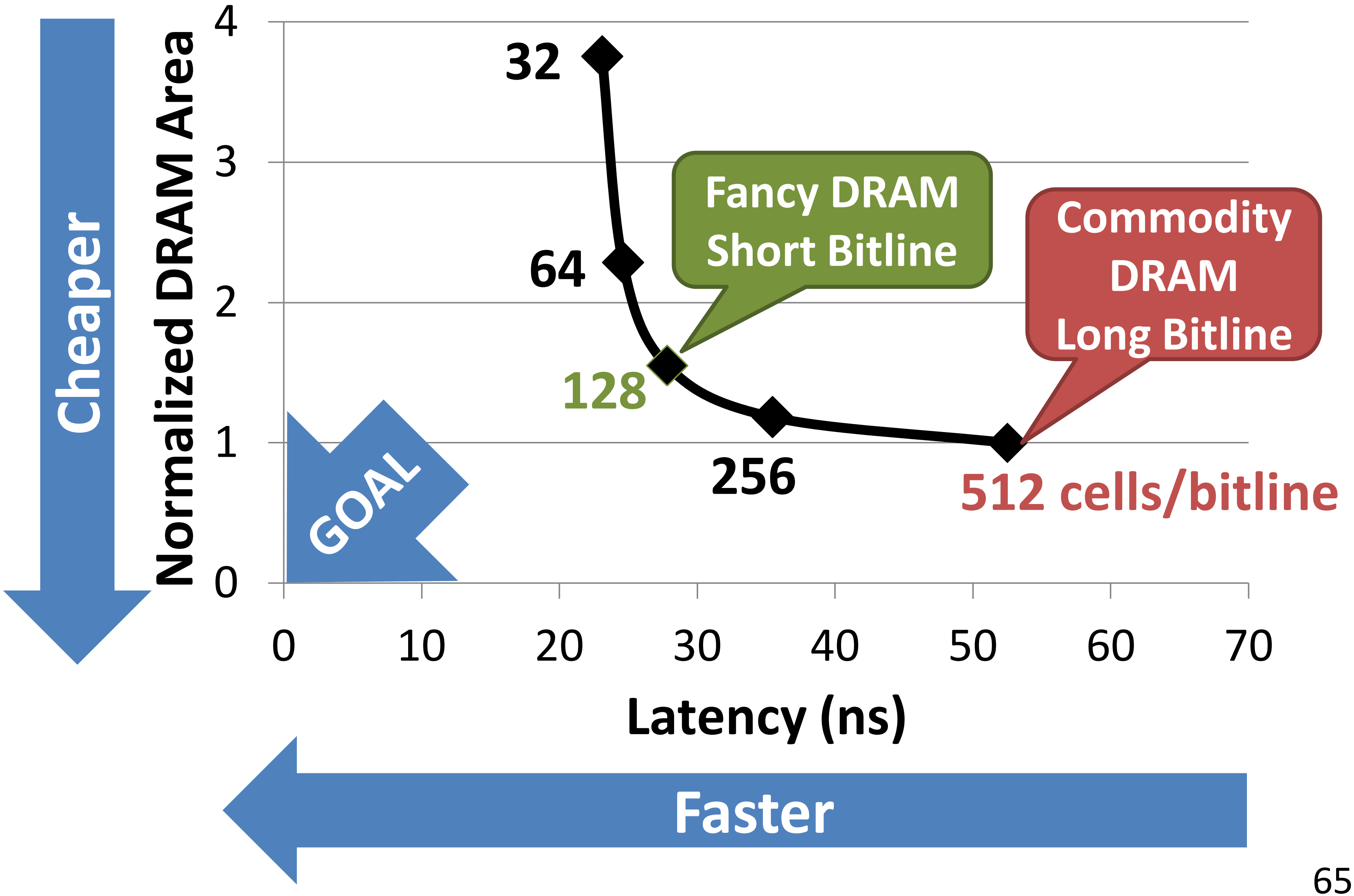
Short Bitline

Faster

Smaller

Trade-Off: Area vs. Latency

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



Approximating the Best of Both Worlds

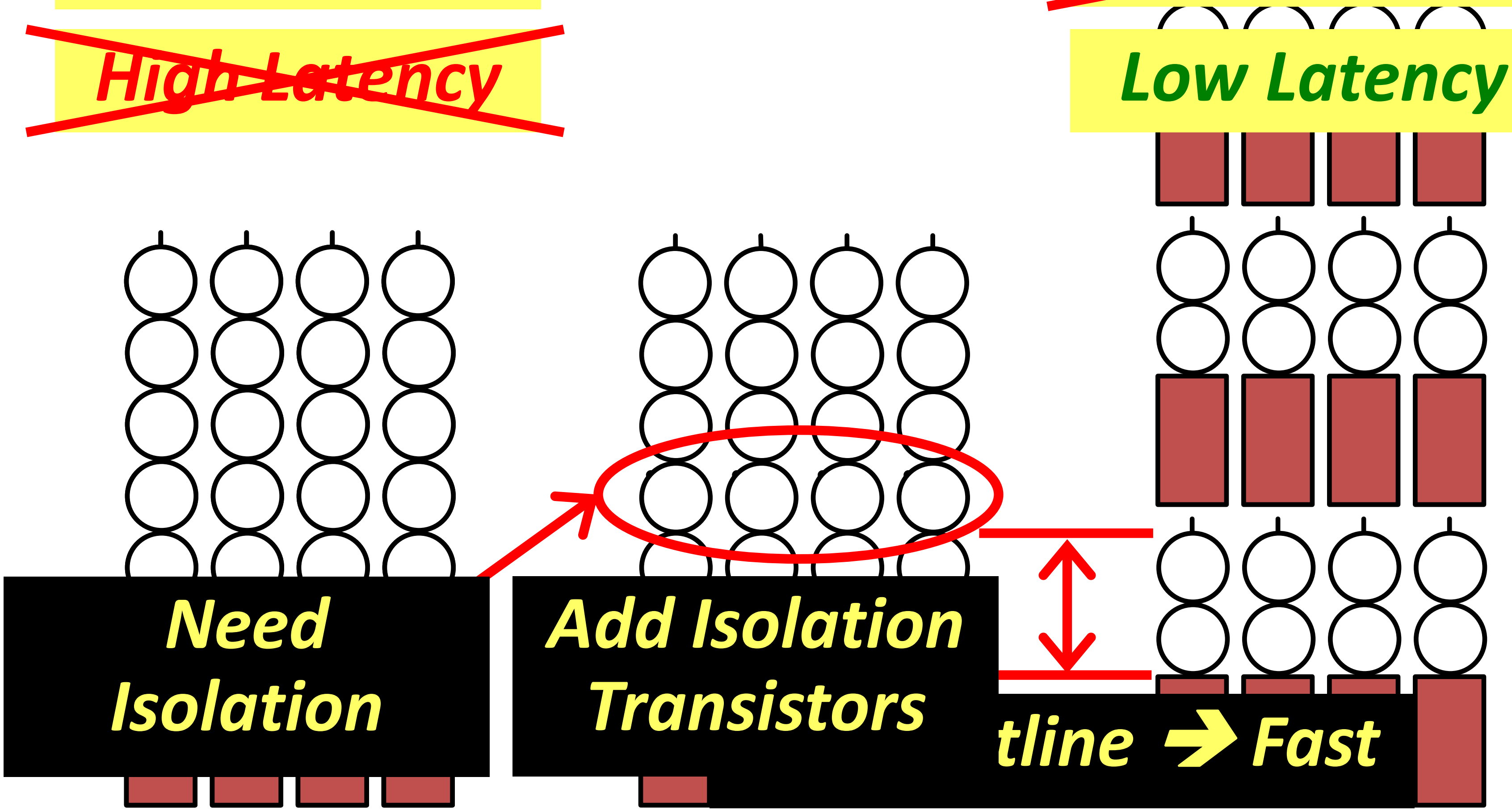
Long Bitline | **Our Proposal** | **Short Bitline**

Small Area

~~*Large Area*~~

~~*High Latency*~~

Low Latency



Need Isolation

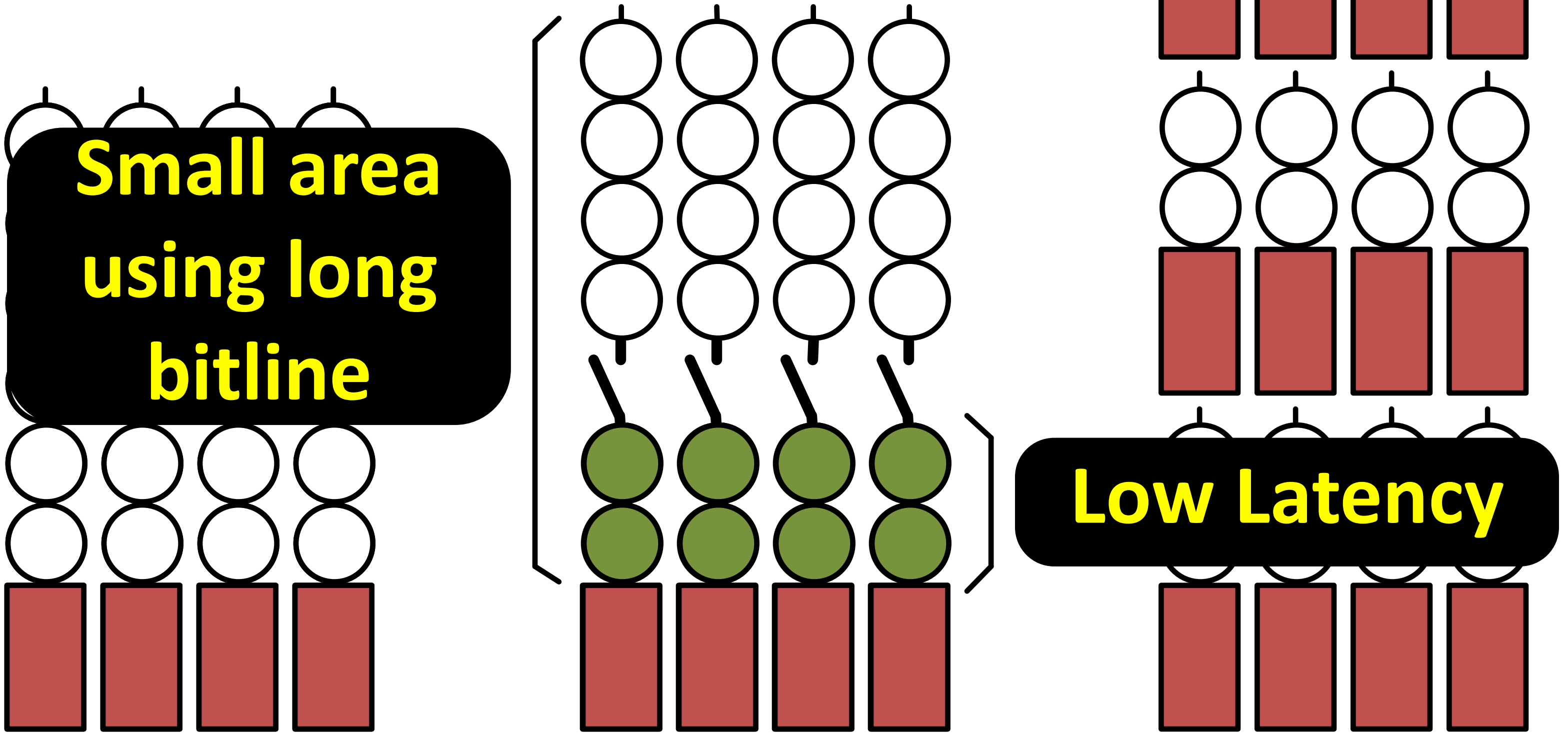
Add Isolation Transistors

tline → Fast

Approximating the Best of Both Worlds

Long Bitlin Tiered-Latency DRAM ıort Bitline

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

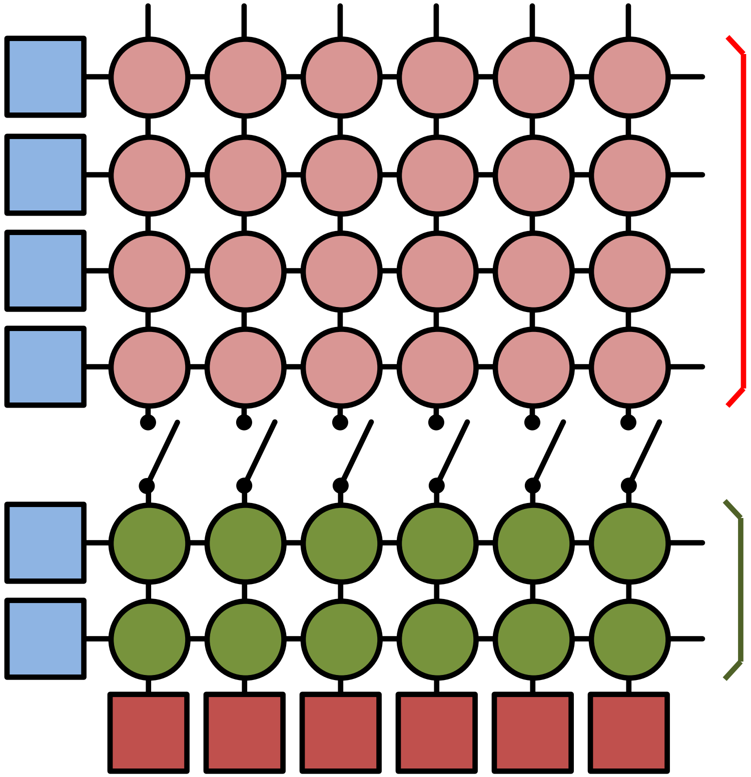


Small area using long bitline

Low Latency

Tiered-Latency DRAM

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



Far Segment

Isolation Transistor

Near Segment

Sense Amplifier

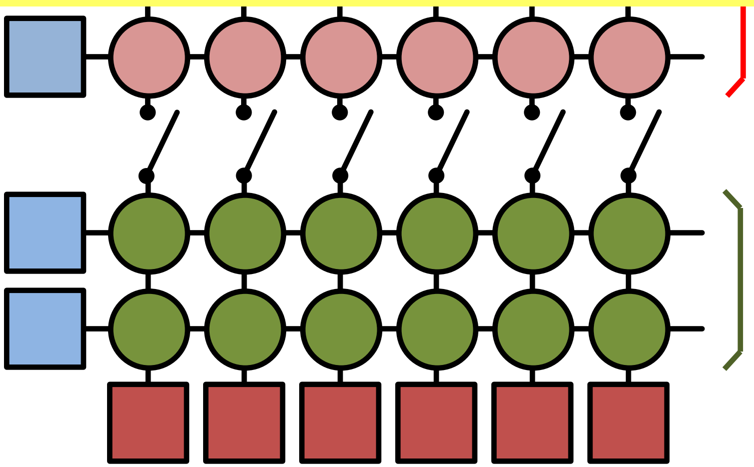
Near Segment Access

- 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

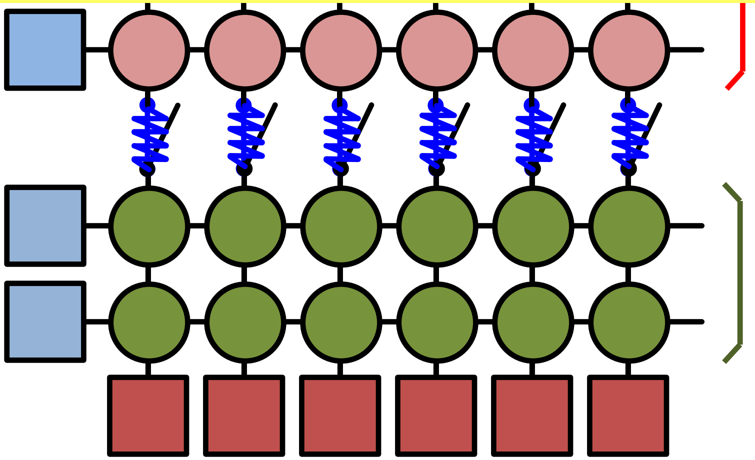
- Turn *on* the isolation transistor

Long bitline length

Large bitline capacitance

Additional resistance of isolation transistor

➔ High latency & high power



Isolation Transistor (on)

Near Segment

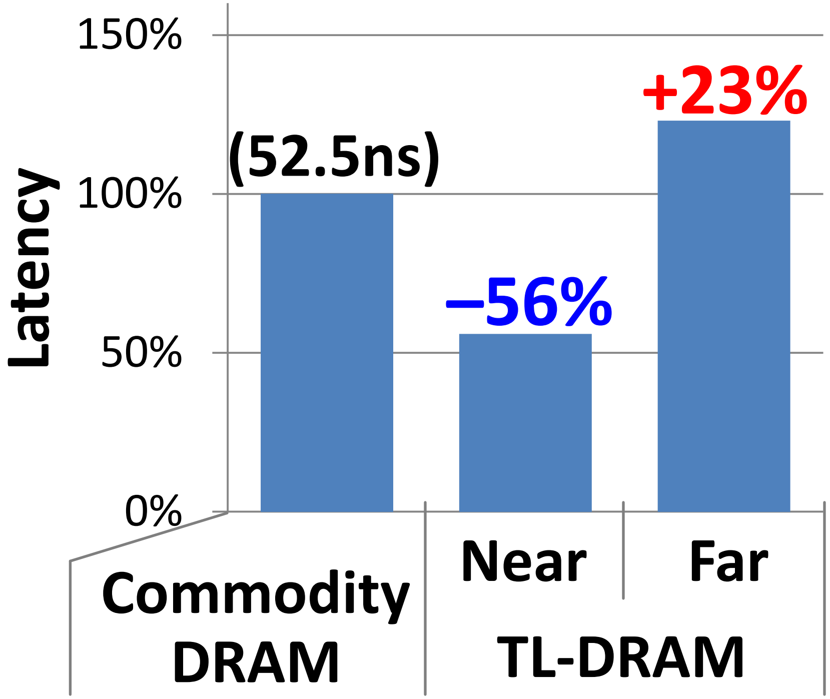
Sense Amplifier

Latency, Power, and Area Evaluation

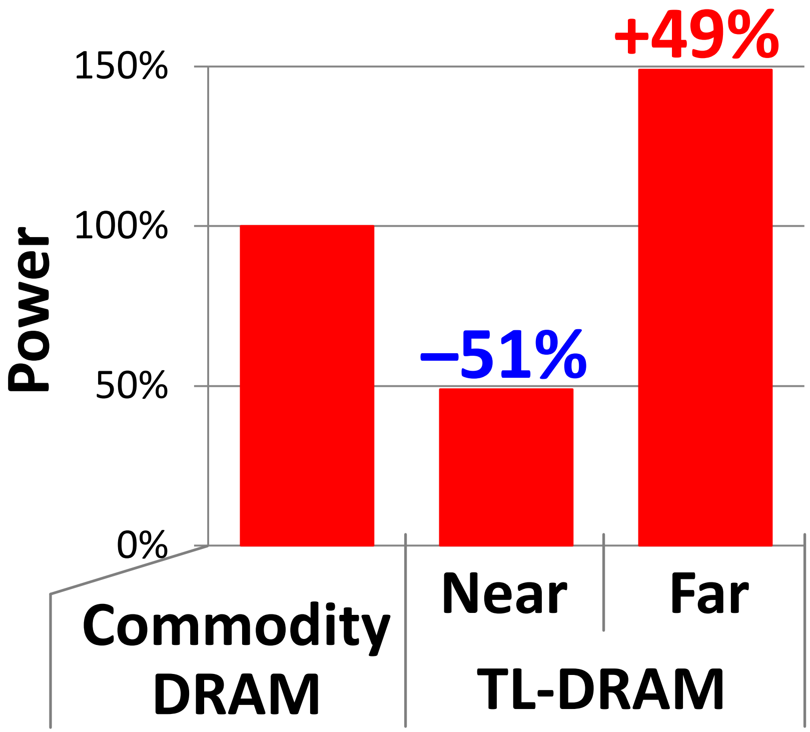
- **Commodity DRAM:** 512 cells/bitline
- **TL-DRAM:** 512 cells/bitline
  - Near segment: 32 cells
  - Far segment: 480 cells
- **Latency Evaluation**
  - SPICE simulation using circuit-level DRAM model
- **Power and Area Evaluation**
  - DRAM area/power simulator from Rambus
  - DDR3 energy calculator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

Commodity DRAM vs. TL-DRAM

- **DRAM Latency** (tRC)



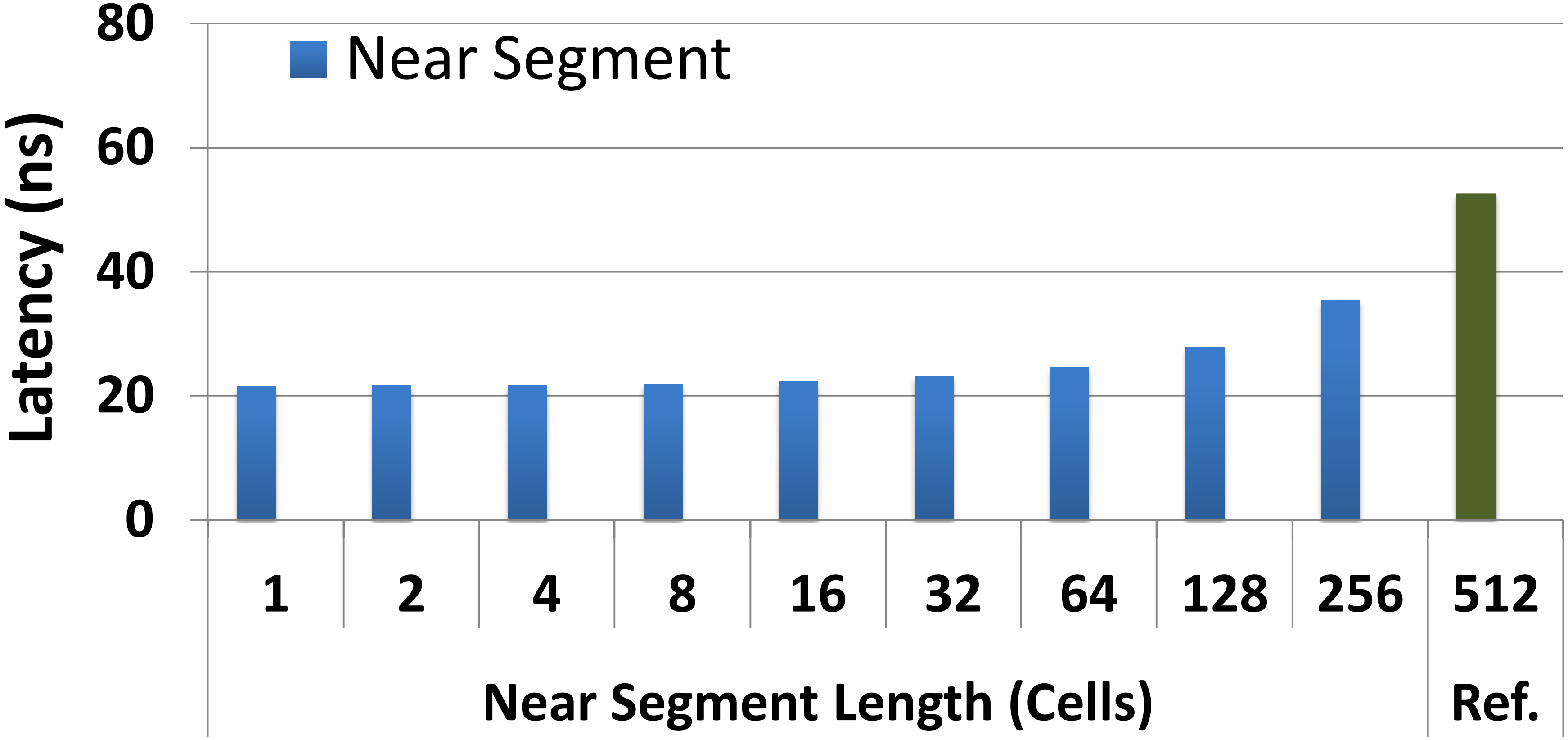
- **DRAM Power**



- **DRAM Area Overhead**

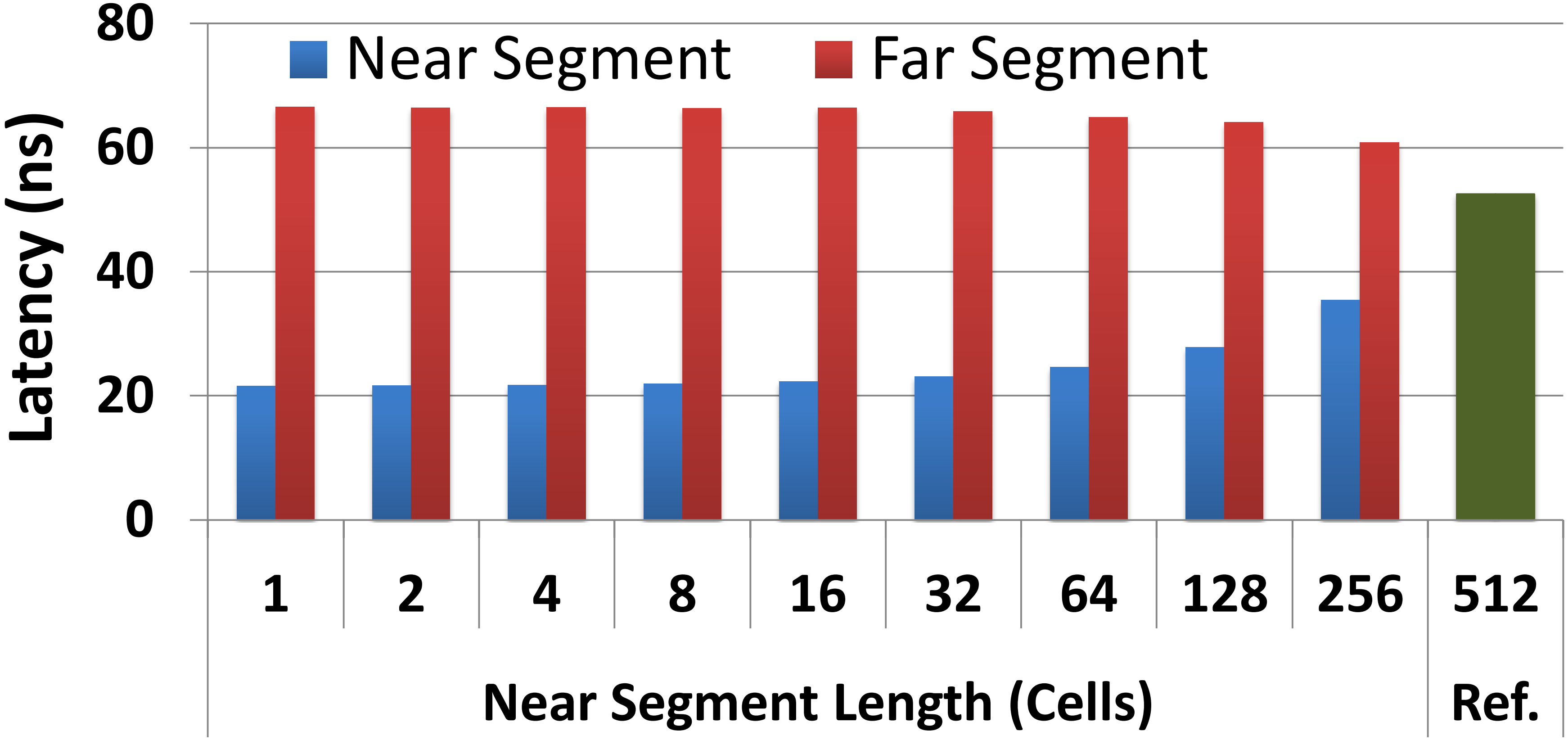
  ~3%: mainly due to the isolation transistors

Latency vs. Near Segment Length



Longer near segment length leads to higher near segment latency

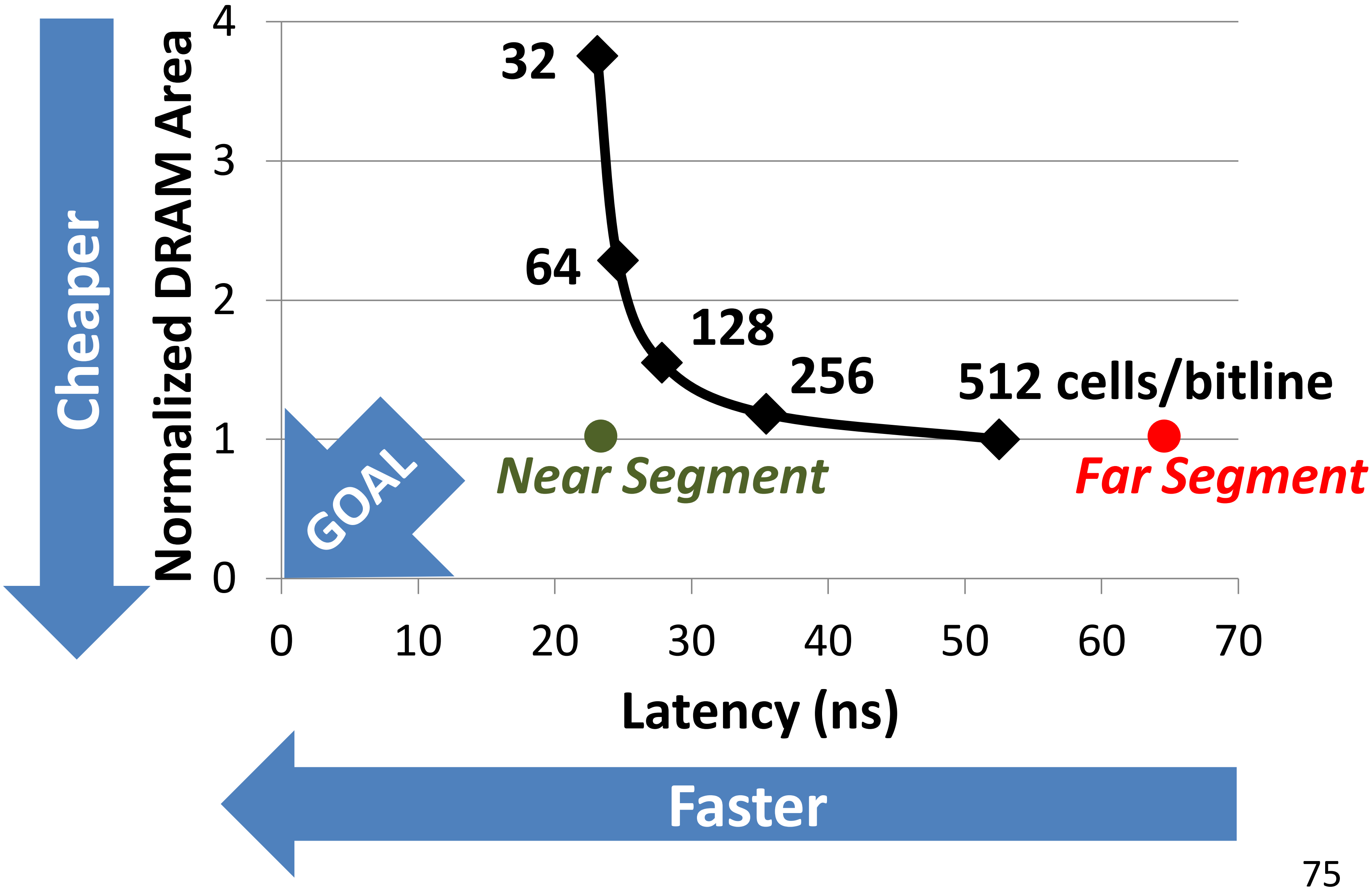
Latency vs. Near Segment Length



Far Segment Length = 512 – Near Segment Length

Far segment latency is higher than commodity DRAM latency

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



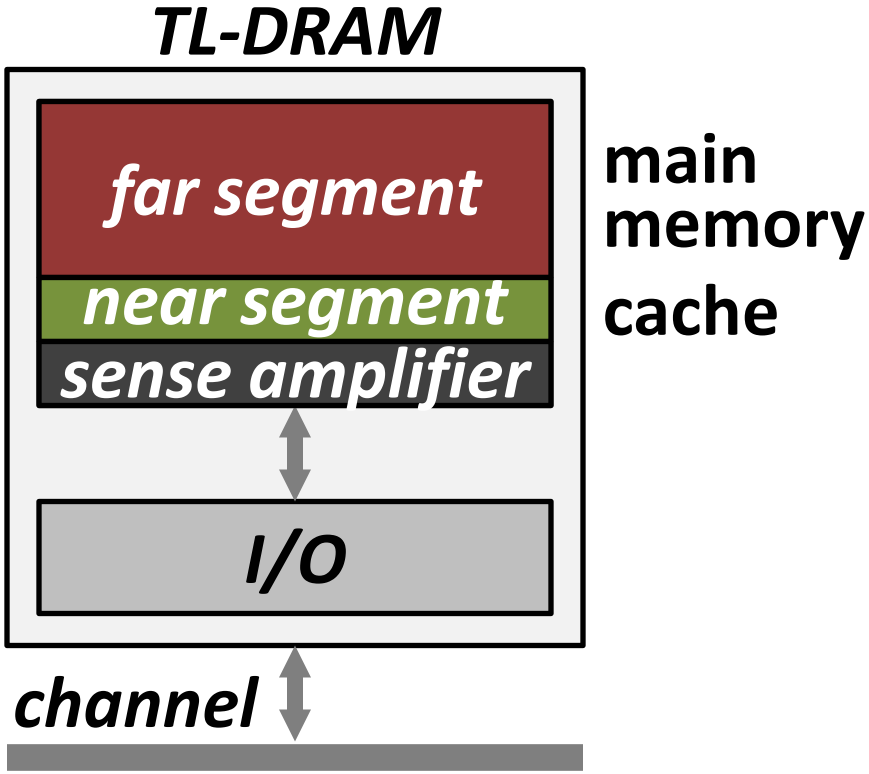
Leveraging Tiered-Latency DRAM

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

- Many potential uses

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

Near Segment as Hardware-Managed Cache

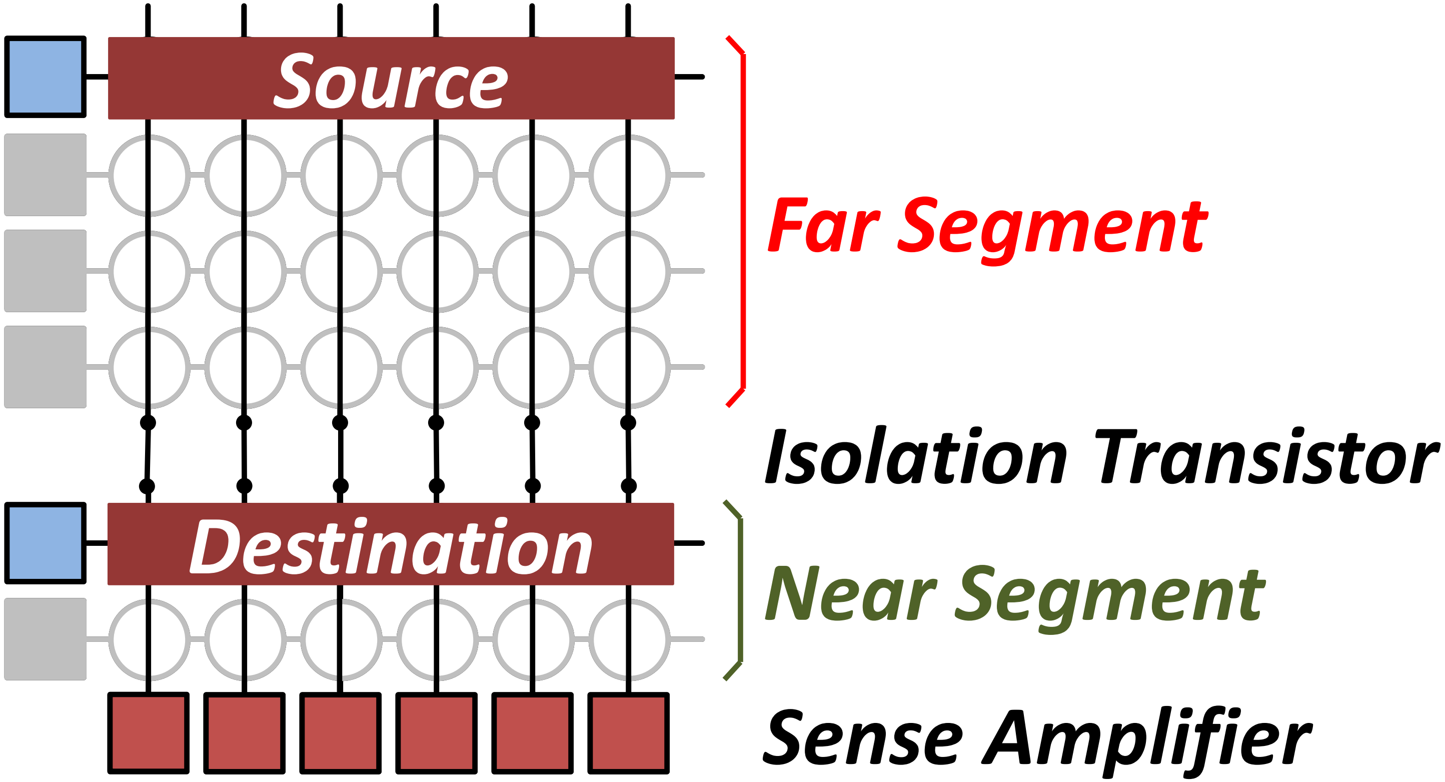


- **Challenge 1:** How to efficiently migrate a row between segments?
- **Challenge 2:** How to efficiently manage the cache?

Inter-Segment Migration

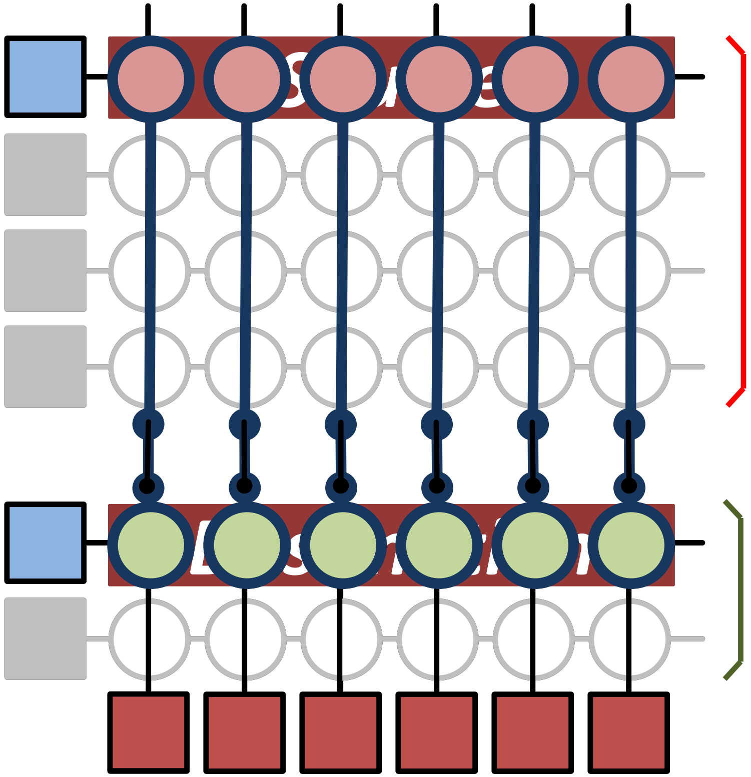
- **Goal:** Migrate source row into destination row
- **Naïve way:** Memory controller reads the source row *byte by byte* and writes to destination row *byte by byte*

→ ***High latency***



Inter-Segment Migration

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



Far Segment

Isolation Transistor

Near Segment

Sense Amplifier

Inter-Segment Migration

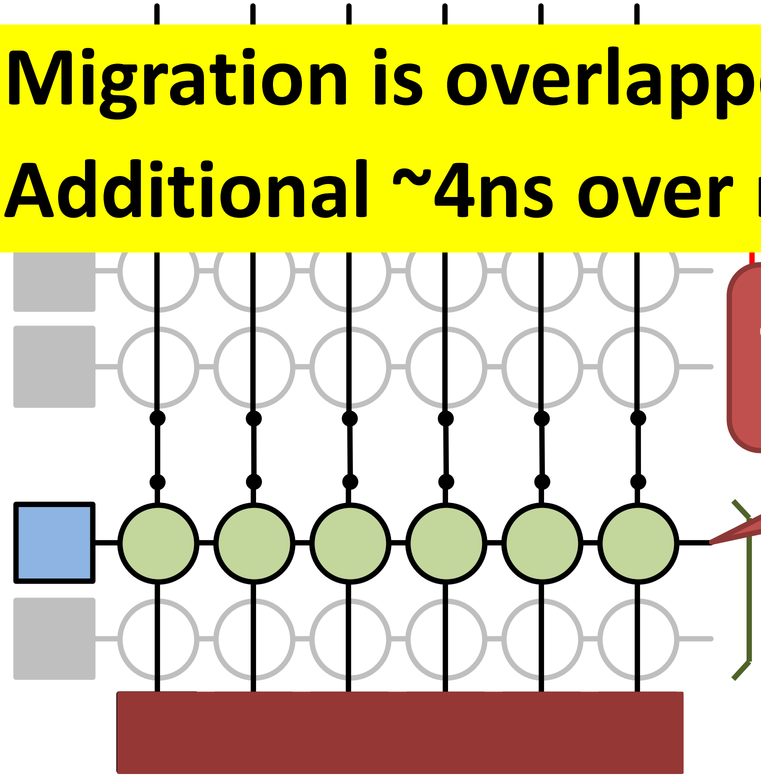
- **Our way:**
  - Source and destination cells ***share bitlines***
  - Transfer data from sou
    shared bitlines concur

Step 1: Activate source row

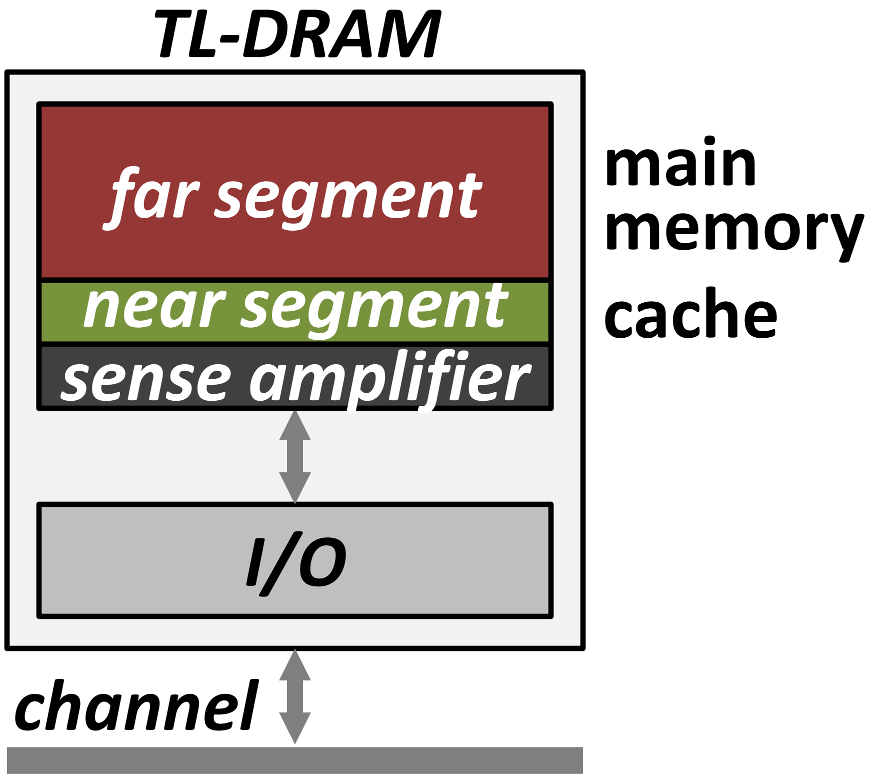
Migration is overlapped with source row access

Additional ~4ns over row access latency

Step 2: Activate destination row to connect cell and bitline



Near Segment as Hardware-Managed Cache



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

Evaluation Methodology

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

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

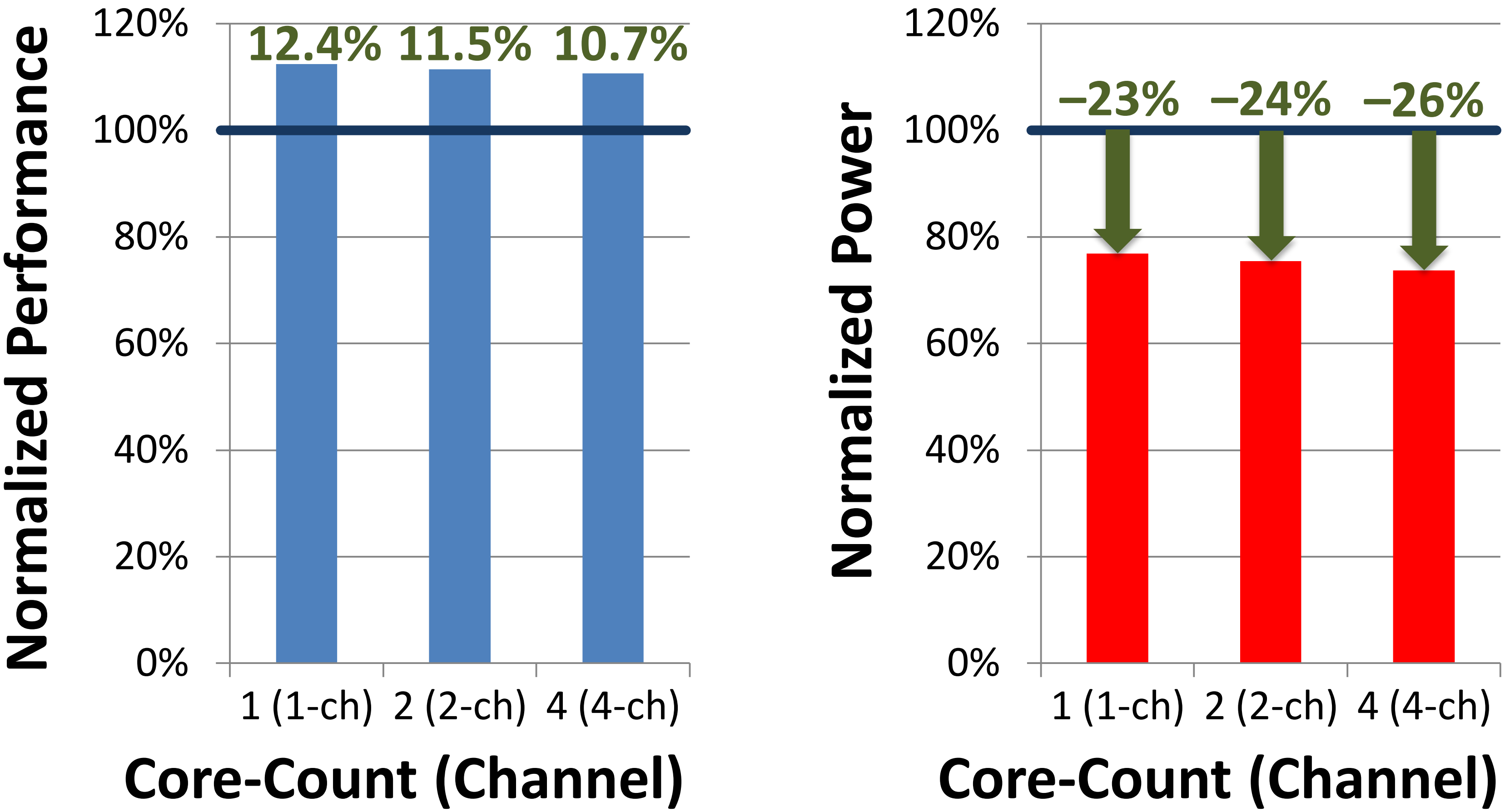
- **Performance Metrics**
  - Single-core: Instructions-Per-Cycle
  - Multi-core: Weighted speedup

Configurations

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

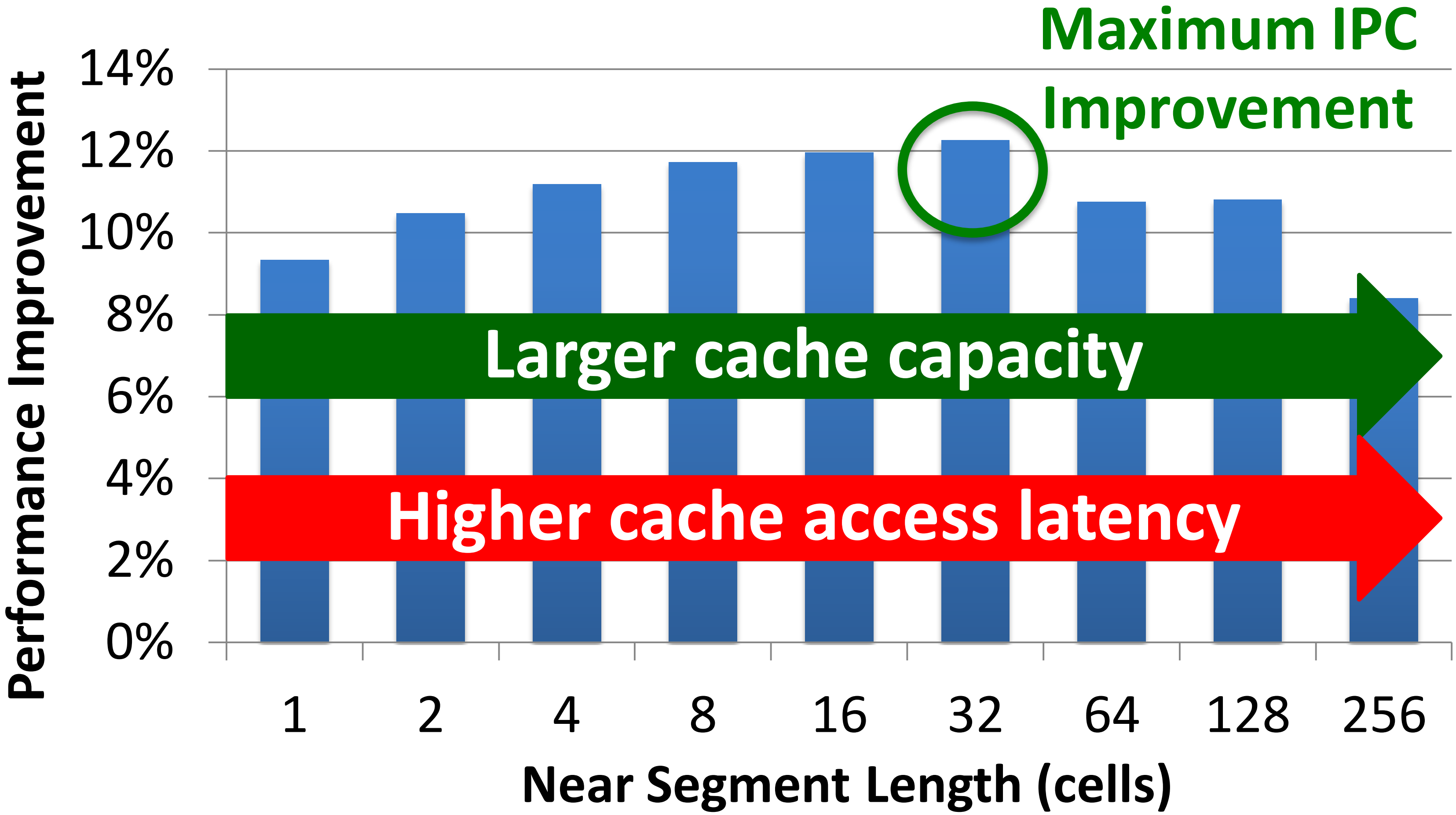
- **TL-DRAM configuration**
  - Total bitline length: **512 cells/bitline**
  - Near segment length: 1-256 cells
  - Hardware-managed inclusive cache: near segment

Performance & Power Consumption



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

Single-Core: Varying Near Segment Length



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

Other Mechanisms & Results

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

Summary of TL-DRAM

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

New DRAM Architectures

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

To Be Covered in Lecture 3

- 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)

- 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.

Scalable Many-Core Memory Systems Lecture 2, 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 16, 2013

Carnegie Mellon

SAFARI

Computer Architecture: Main Memory (Part II)

Prof. Onur Mutlu
Carnegie Mellon University