Scaling the Memory System in the Many-Core Era

Onur Mutlu
onur@cmu.edu

July 26, 2013
BSC/UPC

Carnegie Mellon

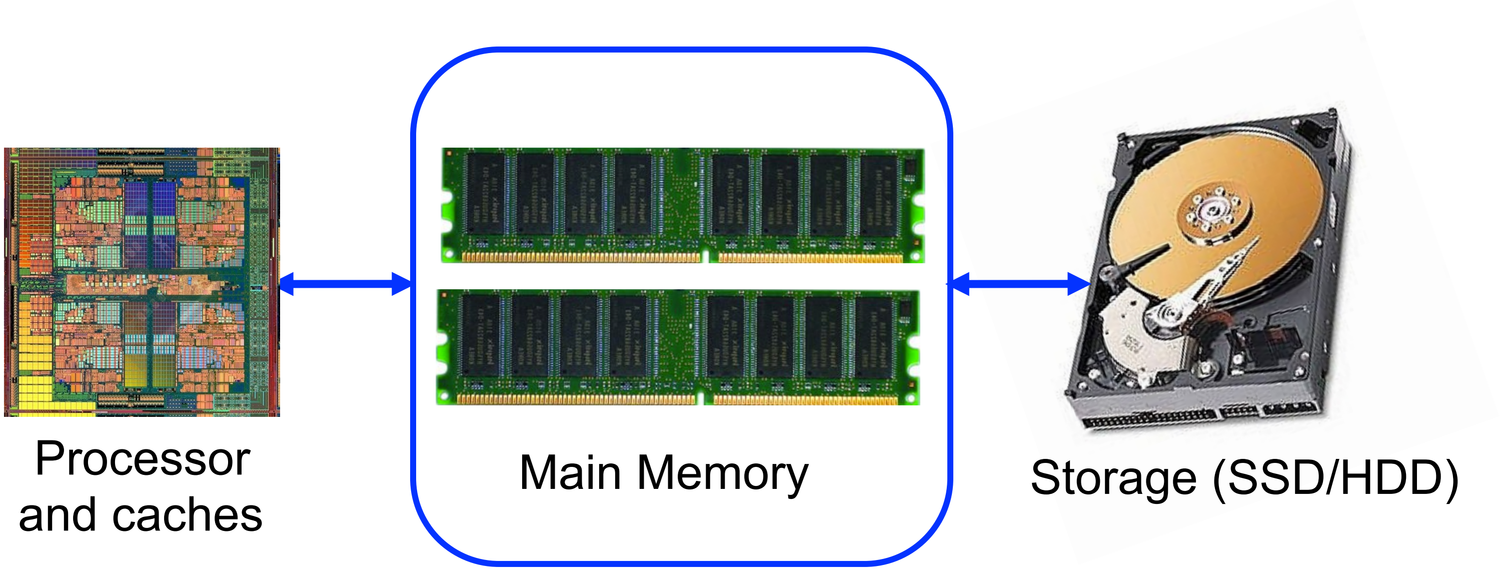
Course Materials and Beyond

- These slides are a shortened version of the Scalable Memory Systems course at ACACES 2013

- Website for Course Slides, Papers, and Videos
  - http://users.ece.cmu.edu/~omutlu/acaces2013-memory.html
  - http://users.ece.cmu.edu/~omutlu
  - Includes extended lecture notes and readings

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

The Main Memory System



Processor and caches

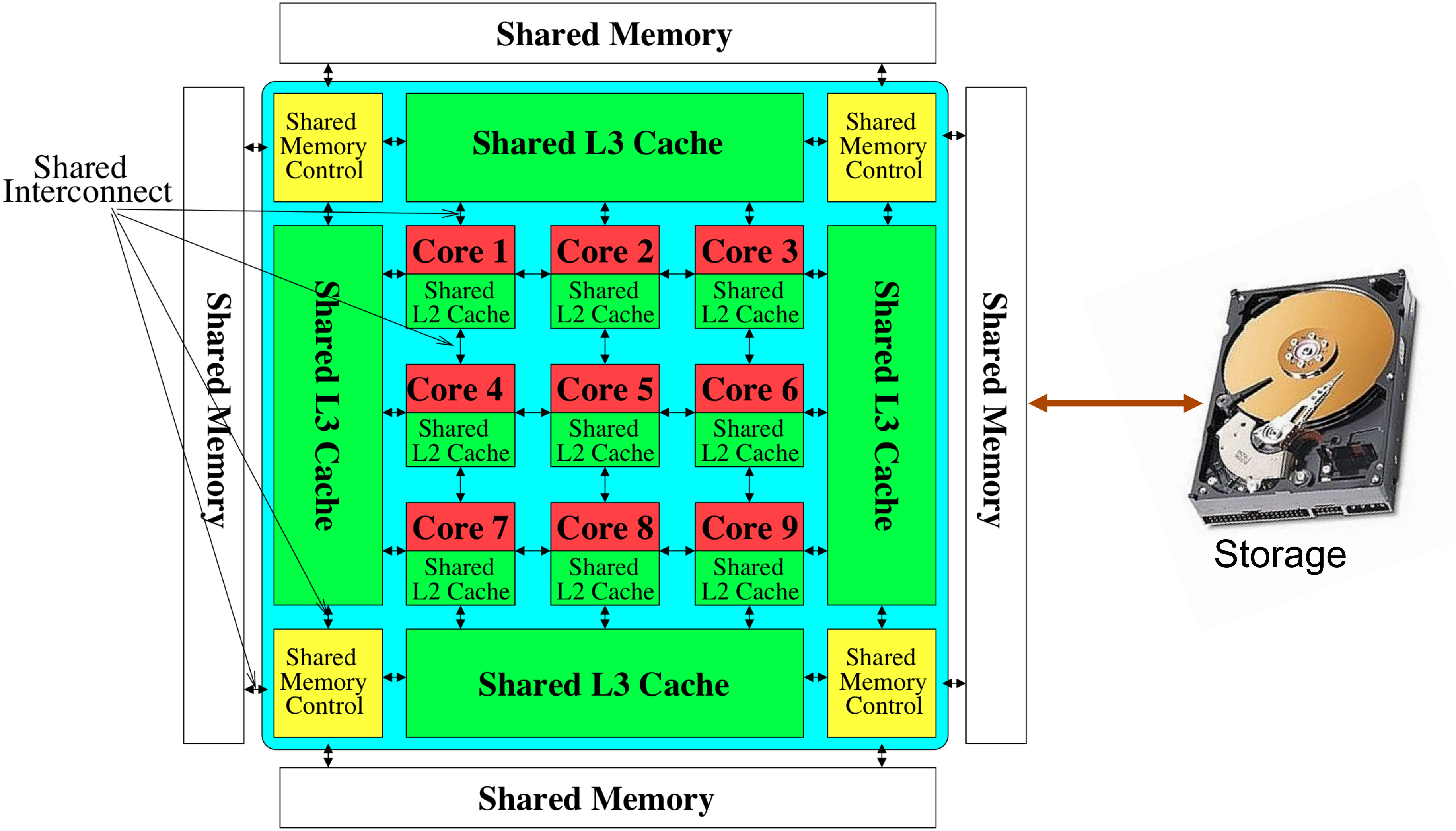
Main Memory

Storage (SSD/HDD)

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

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

Memory System: A *Shared Resource* View



State of the Main Memory System

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

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

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

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

Agenda

- Major Trends Affecting Main Memory
- The DRAM Scaling Problem and Solution Directions
  - Tolerating DRAM: New DRAM Architectures
  - Enabling Emerging Technologies: Hybrid Memory Systems
- The Memory Interference/QoS Problem and Solutions
  - QoS-Aware Memory Systems
- How Can We Do Better?
- 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)

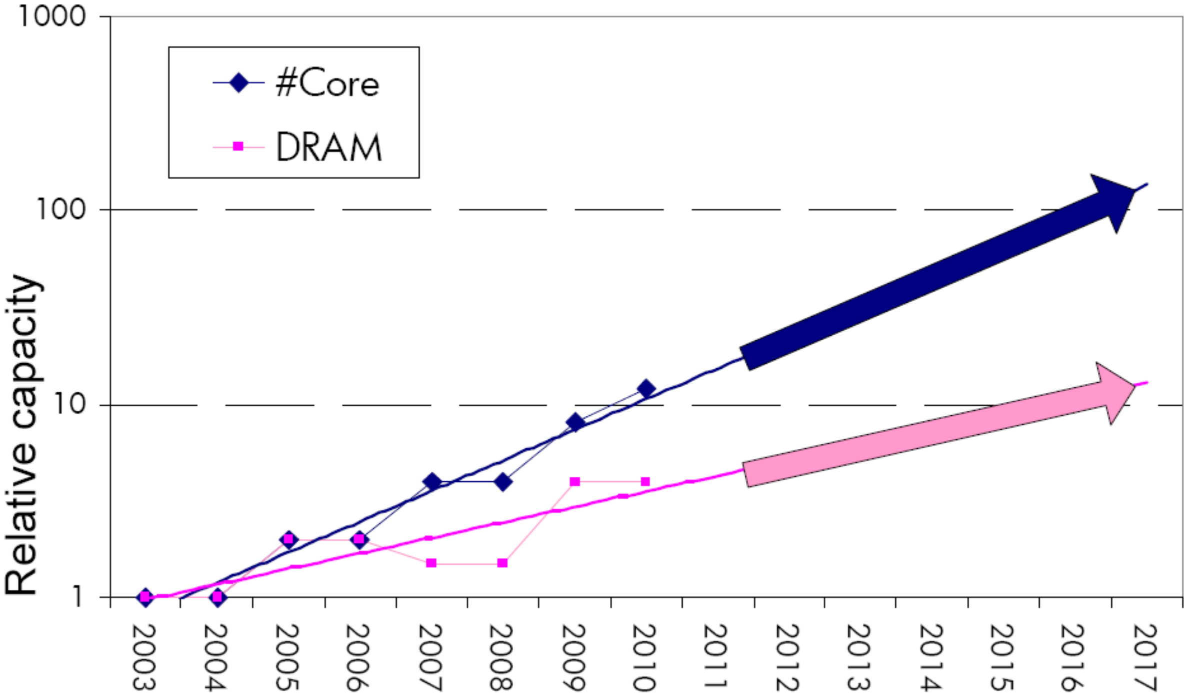
- Need for main memory capacity, bandwidth, QoS increasing
  - Multi-core: increasing number of cores/agents
  - Data-intensive applications: increasing demand/hunger for data
  - Consolidation: cloud computing, GPUs, mobile, heterogeneity

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

- DRAM technology scaling is ending

Example: The Memory Capacity Gap

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



Source: Lim et al., ISCA 2009.

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

Major Trends Affecting Main Memory (III)

- Need for main memory capacity, bandwidth, QoS increasing

- Main memory energy/power is a key system design concern
  - ~40-50% energy spent in off-chip memory hierarchy [Lefurgy, IEEE Computer 2003]
  - DRAM consumes power even when not used (periodic refresh)

- DRAM technology scaling is ending

Major Trends Affecting Main Memory (IV)

- Need for main memory capacity, bandwidth, QoS increasing

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

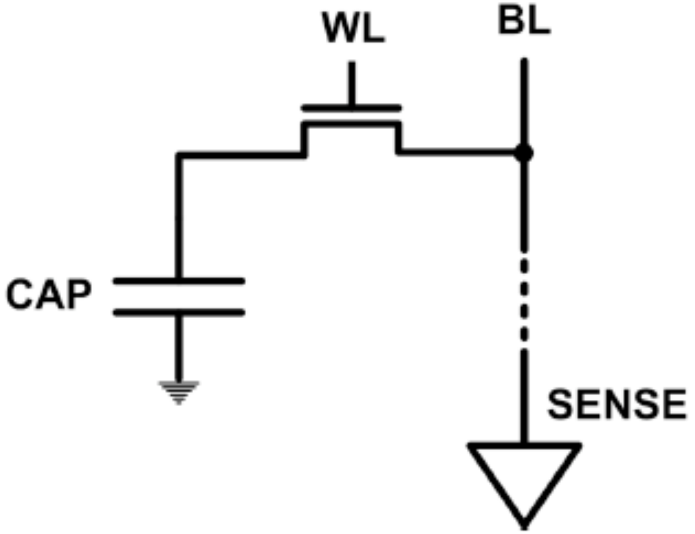
- DRAM technology scaling is ending
  - ITRS projects DRAM will not scale easily below X nm
  - Scaling has provided many benefits:
    - higher capacity (density), lower cost, lower energy

Agenda

- Major Trends Affecting Main Memory
- The DRAM Scaling Problem and Solution Directions
  - Tolerating DRAM: New DRAM Architectures
  - Enabling Emerging Technologies: Hybrid Memory Systems
- The Memory Interference/QoS Problem and Solutions
  - QoS-Aware Memory Systems
- How Can We Do Better?
- 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

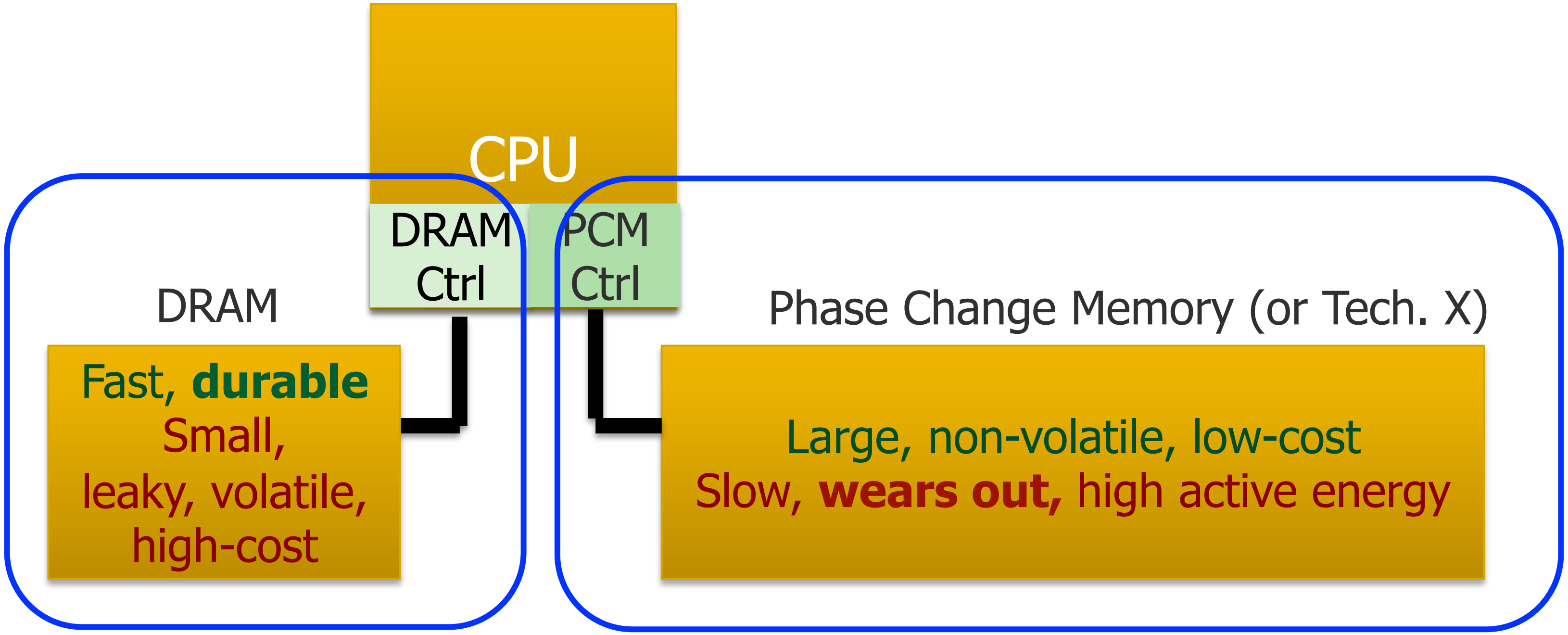
- Liu, Jaiyen, Veras, Mutlu, "RAIDR: Retention-Aware Intelligent DRAM Refresh," ISCA 2012.
- Kim, Seshadri, Lee+, "A Case for Exploiting Subarray-Level Parallelism in DRAM," ISCA 2012.
- Lee+, "Tiered-Latency DRAM: A Low Latency and Low Cost DRAM Architecture," HPCA 2013.
- Liu+, "An Experimental Study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

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.
- Kultursay+, "Evaluating STT-RAM as an Energy-Efficient Main Memory Alternative," ISPASS 2013.

Hybrid Memory Systems


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

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

Meza+, "Enabling Efficient and Scalable Hybrid Memories," IEEE Comp. Arch. Letters, 2012.

Yoon, Meza et al., "Row Buffer Locality Aware Caching Policies for Hybrid Memories," ICCD 2012 Best Paper Award.

An Orthogonal Issue: Memory Interference

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

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

- Solution: Hardware/software cooperative memory QoS
  - Hardware designed to provide a configurable fairness substrate
    - Application-aware memory scheduling, partitioning, throttling
  - Software designed to configure the resources to satisfy different QoS goals

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

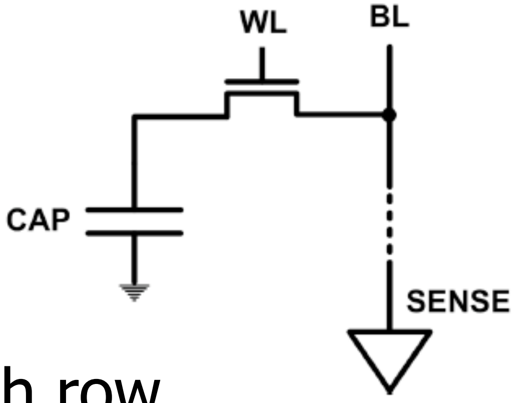
Agenda

- Major Trends Affecting Main Memory
- The DRAM Scaling Problem and Solution Directions
  - Tolerating DRAM: New DRAM Architectures
  - Enabling Emerging Technologies: Hybrid Memory Systems
- The Memory Interference/QoS Problem and Solutions
  - QoS-Aware Memory Systems
- How Can We Do Better?
- Summary

Tolerating DRAM: Example Techniques

- Retention-Aware DRAM Refresh: Reducing Refresh Impact
- Tiered-Latency DRAM: Reducing DRAM Latency
- RowClone: In-Memory Page Copy and Initialization
- Subarray-Level Parallelism: Reducing Bank Conflict Impact

DRAM Refresh



- DRAM capacitor charge leaks over time

- The memory controller needs to refresh each row periodically to restore charge
  - Read and close each row every N ms
  - Typical N = 64 ms

- Downsides of refresh

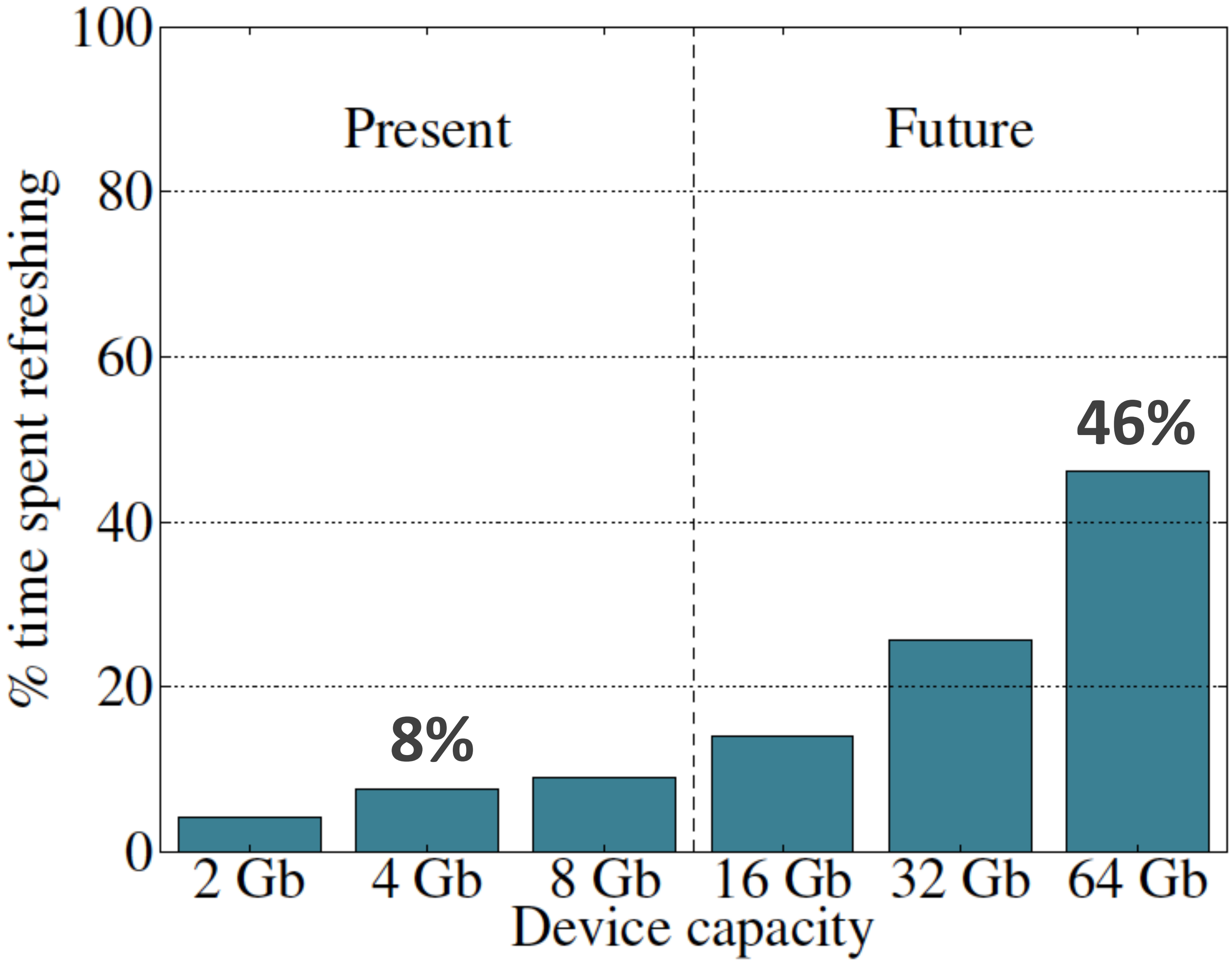
  -- Energy consumption: Each refresh consumes energy

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

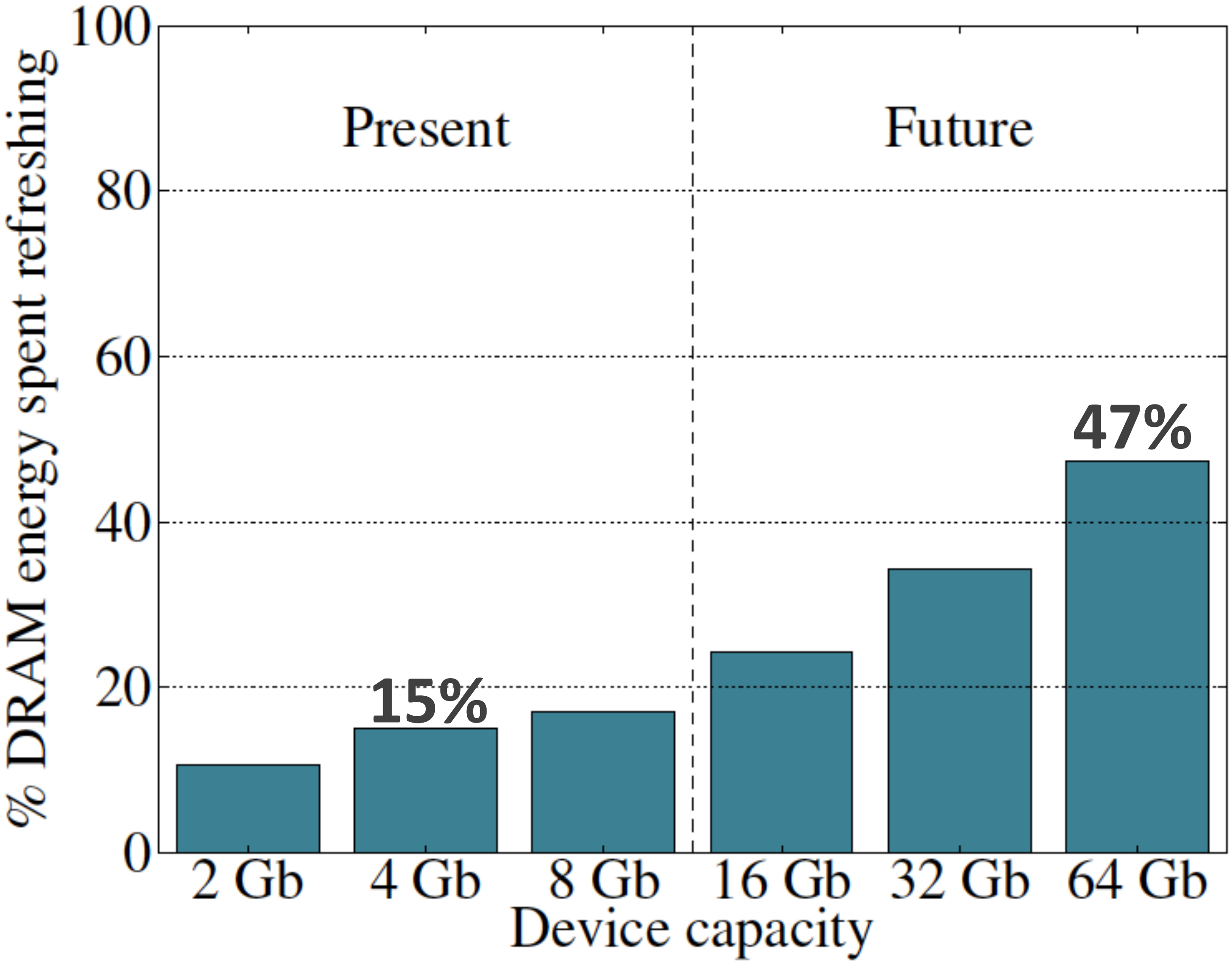
  -- QoS/predictability impact: (Long) pause times during refresh

  -- Refresh rate limits DRAM capacity scaling

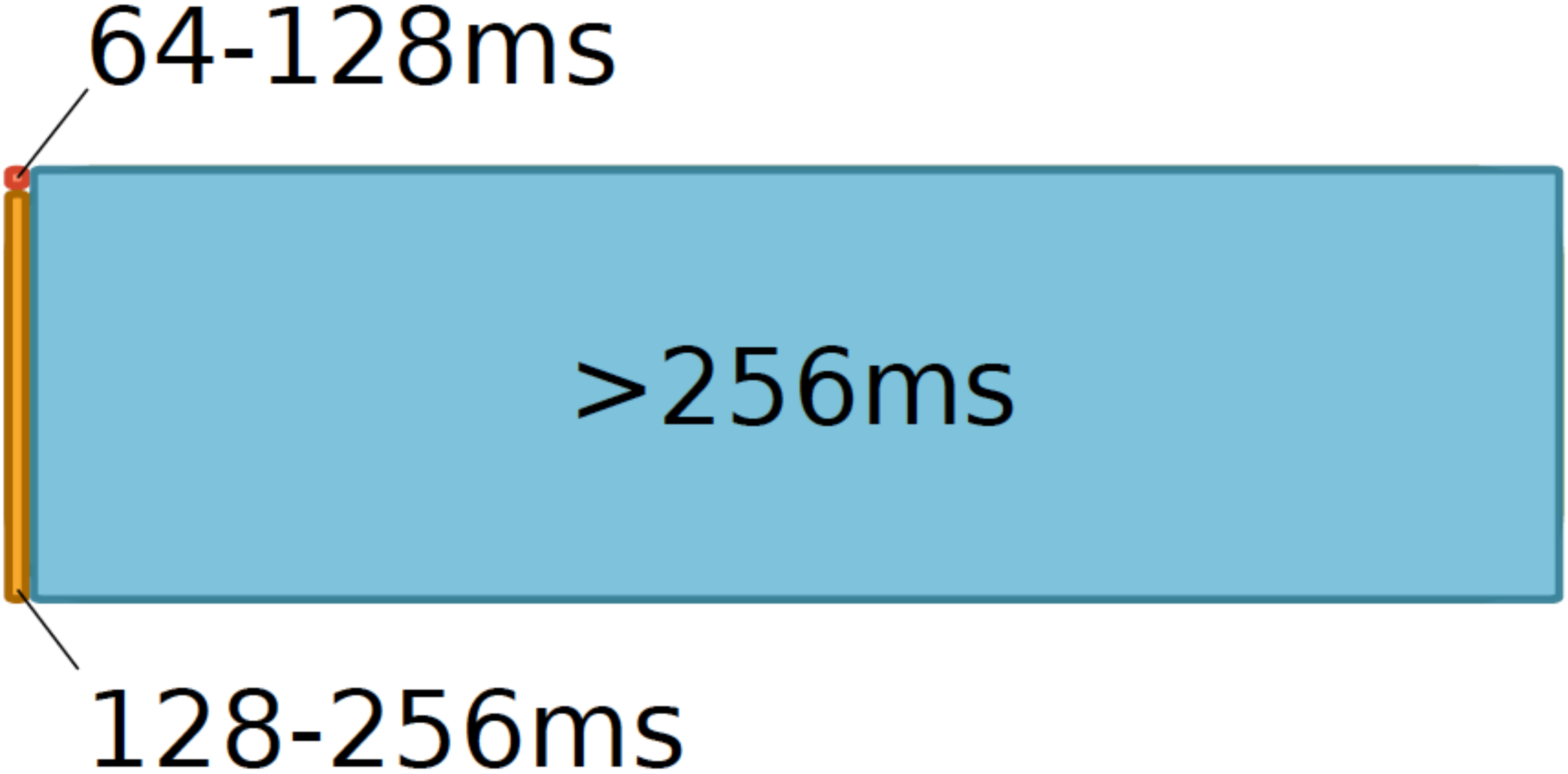
Refresh Overhead: Performance



Refresh Overhead: Energy

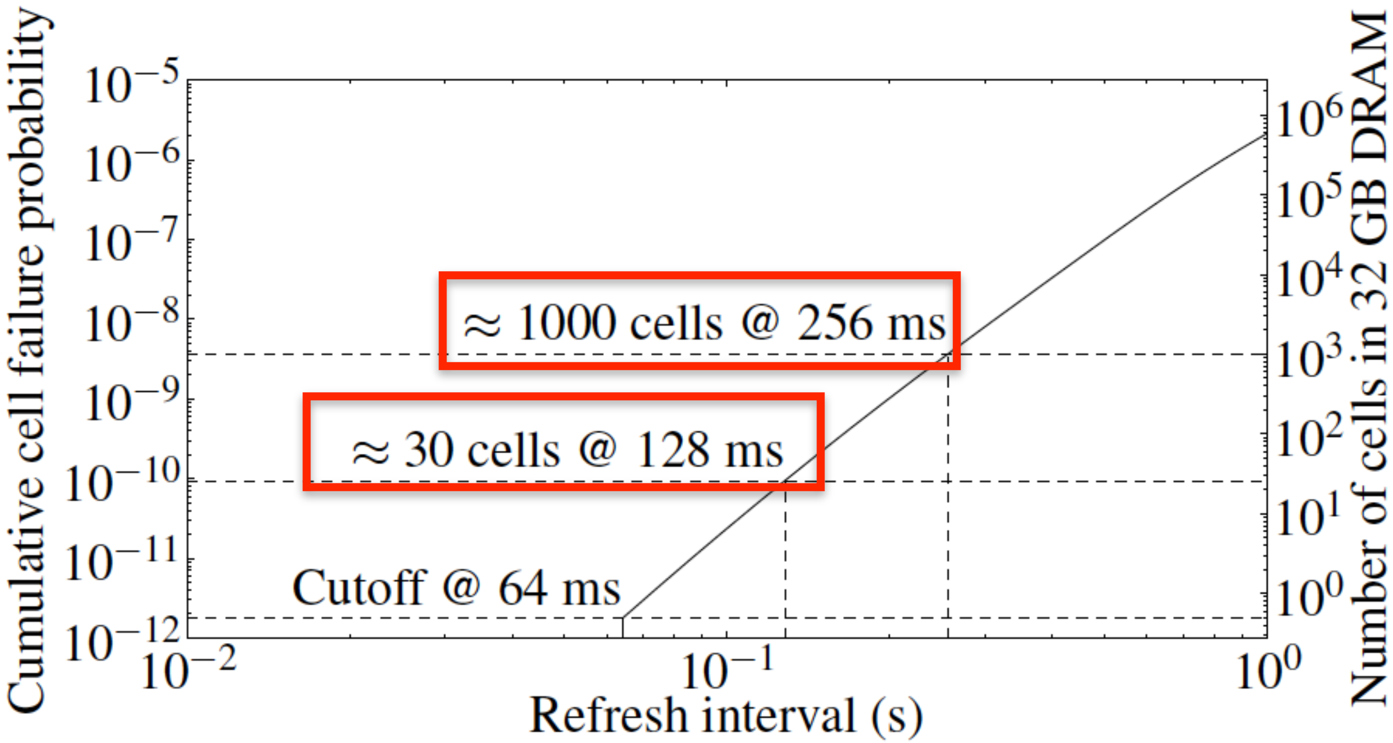


Retention Time Profile of DRAM



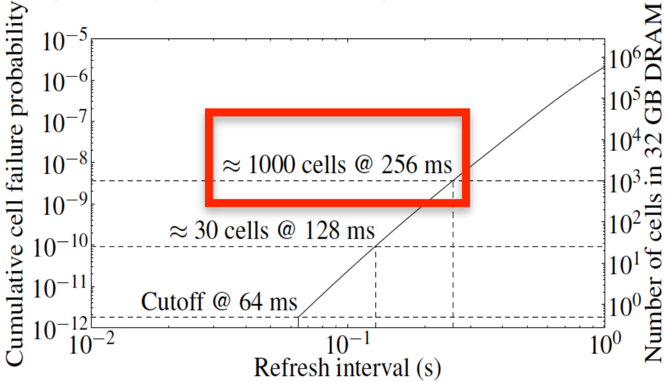
Retention Time Profile of DRAM

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



RAIDR: Eliminating Unnecessary Refreshes

- Observation: Most DRAM rows can be refreshed much less often without losing data [Kim+, EDL'09]

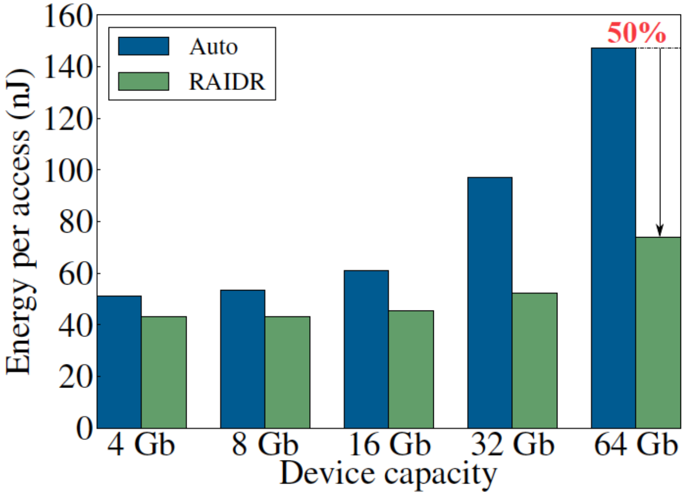


- Key idea: Refresh rows containing weak cells more frequently, other rows less frequently

  1. Profiling: Profile retention time of all rows
  2. Binning: Store rows into bins by retention time in memory controller

     Efficient storage with Bloom Filters (only 1.25KB for 32GB memory)
  3. Refreshing: Memory controller refreshes rows in different bins at different rates

- Results: 8-core, 32GB, SPEC, TPC-C, TPC-H
  - 74.6% refresh reduction @ 1.25KB storage
  - ~16%/20% DRAM dynamic/idle power reduction
  - ~9% performance improvement
  - Energy benefits increase with DRAM capacity



SAFARI

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

1. Profiling

To profile a row:

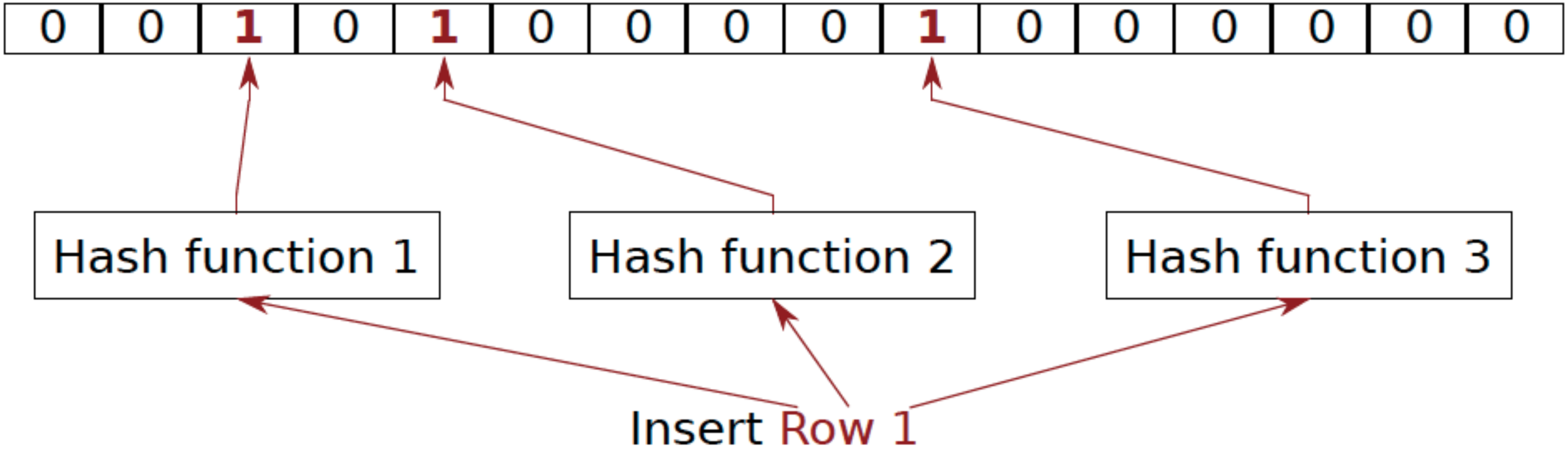
1. Write data to the row
2. Prevent it from being refreshed
3. Measure time before data corruption

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

2. Binning

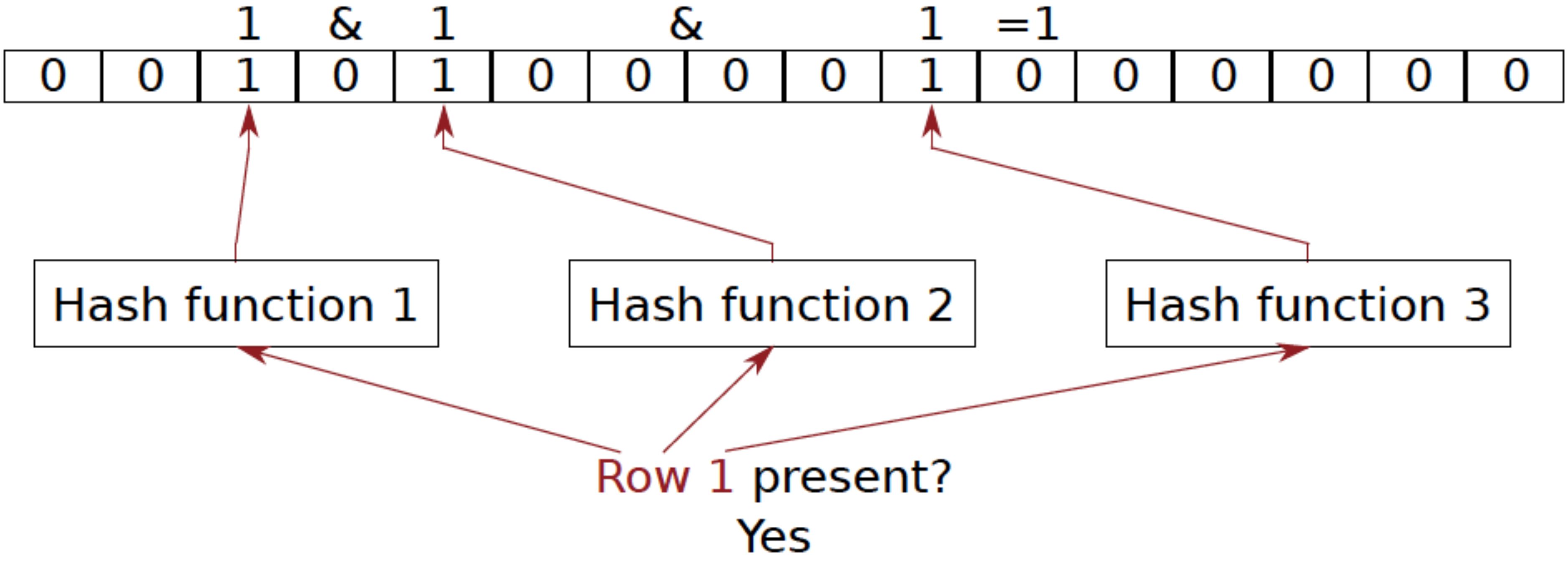
- How to efficiently and scalably store rows into retention time bins?
- Use Hardware Bloom Filters [Bloom, CACM 1970]

Example with 64-128ms bin:



Bloom Filter Operation Example

Example with 64–128ms bin:



1 & 1 & 1 =1

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

Hash function 1

Hash function 2

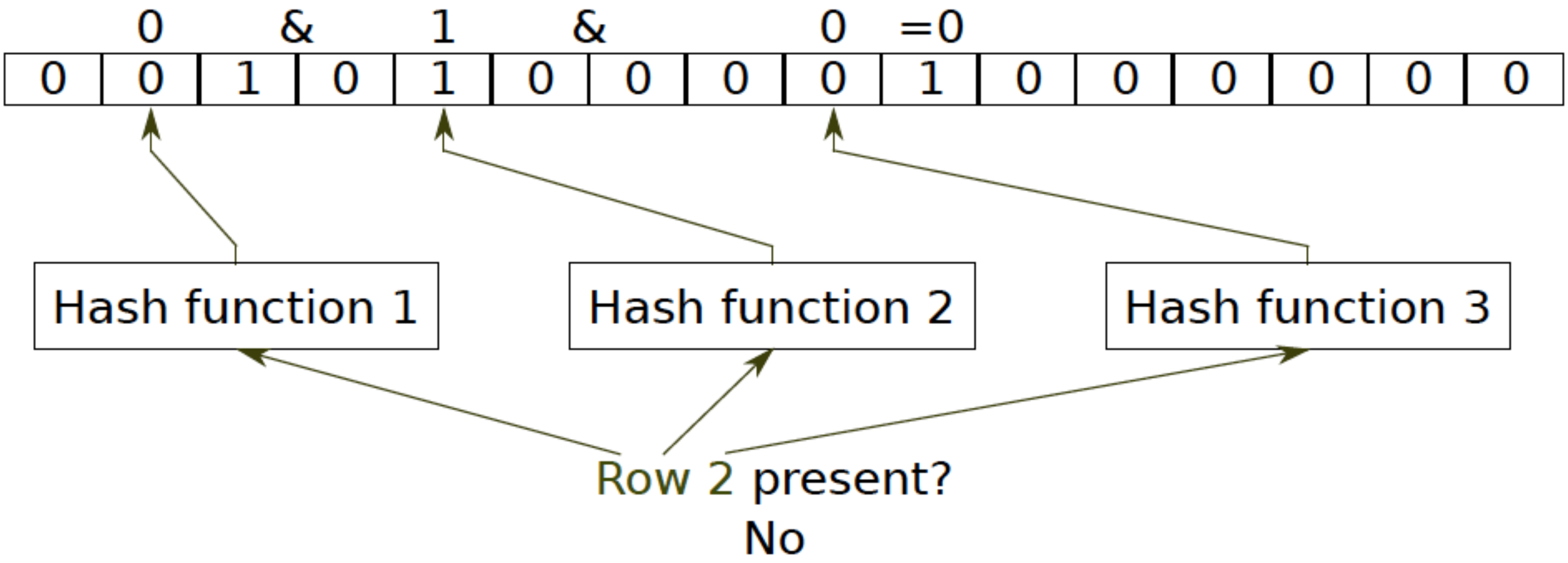
Hash function 3

Row 1 present?

Yes

Bloom Filter Operation Example

Example with 64–128ms bin:



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

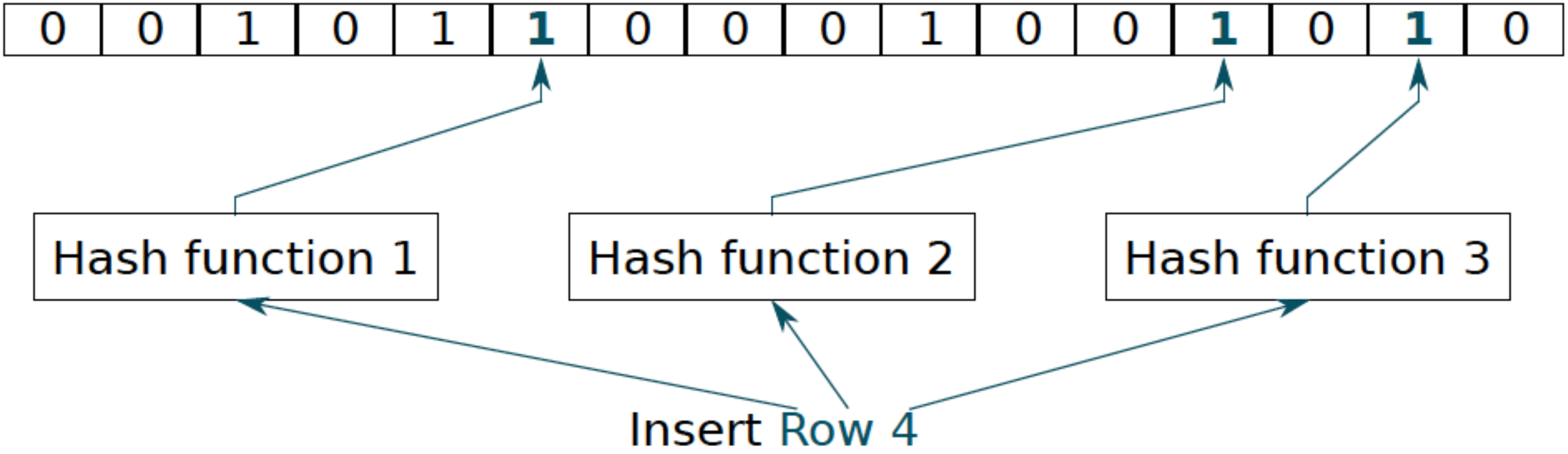
Hash function 3

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

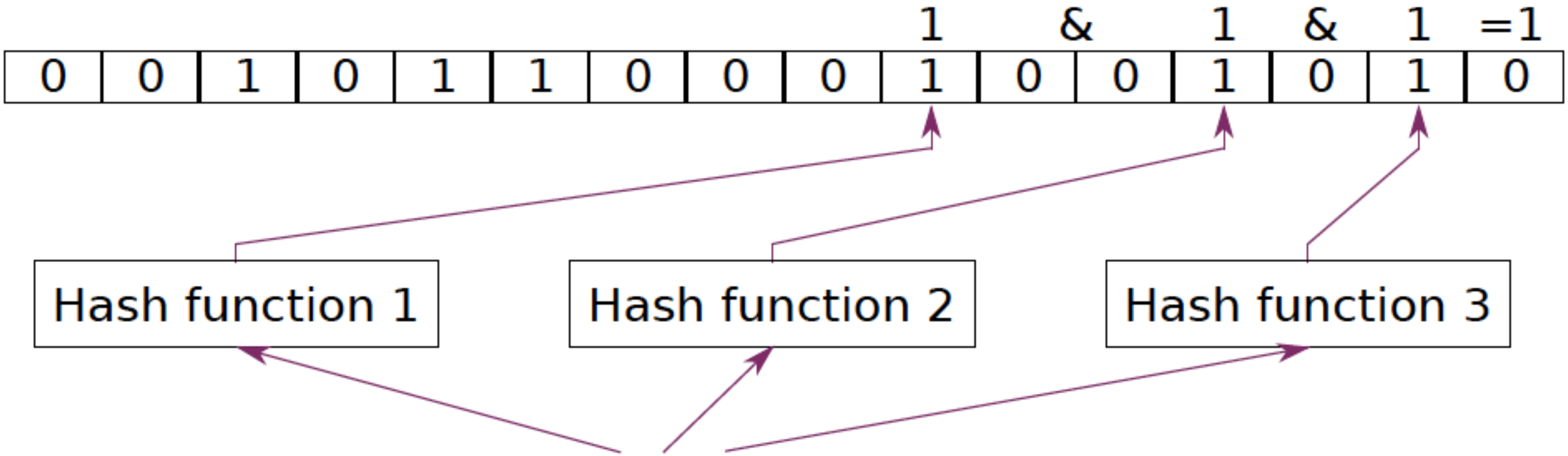
Hash function 2

Hash function 3

Insert Row 4

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

Row 5 present?

Yes (false positive)

Benefits of Bloom Filters as Bins

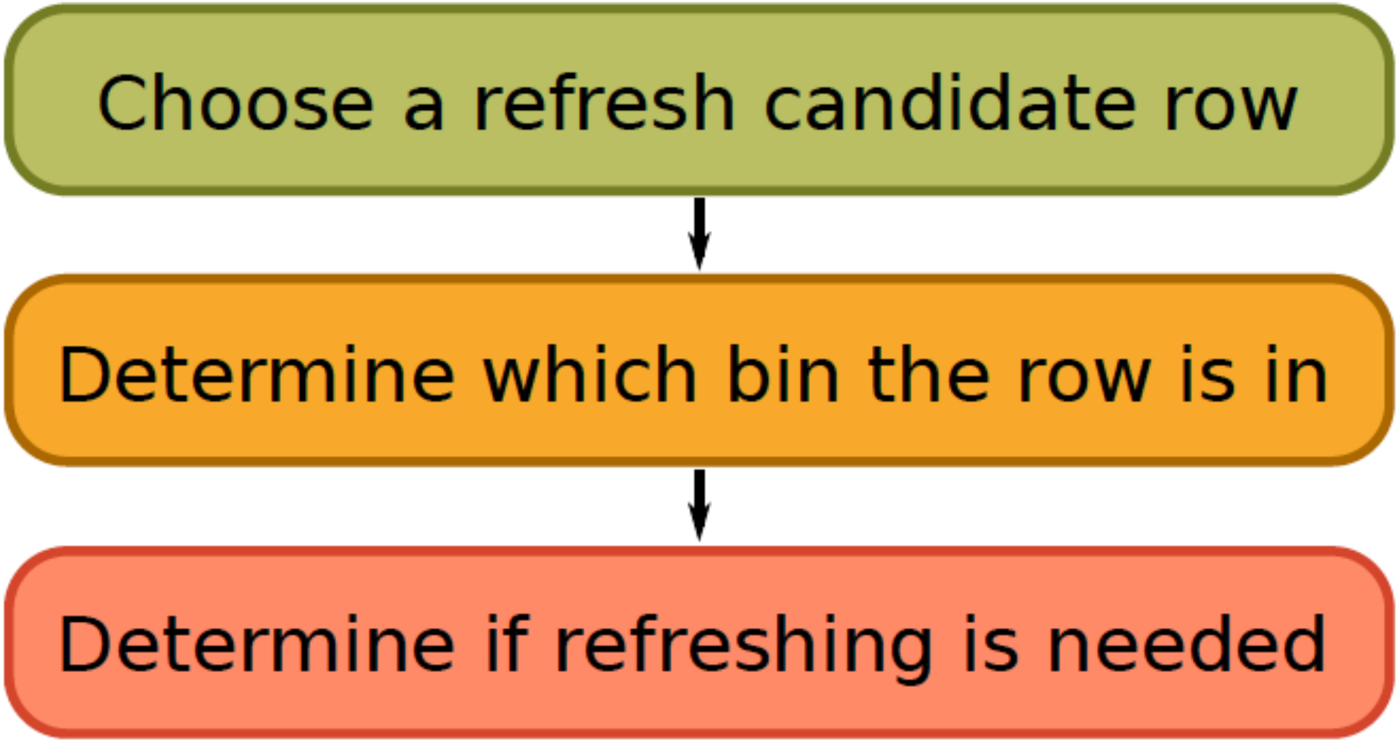
- False positives: a row may be declared present in the Bloom filter even if it was never inserted
  - Not a problem: Refresh some rows more frequently than needed

- 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)

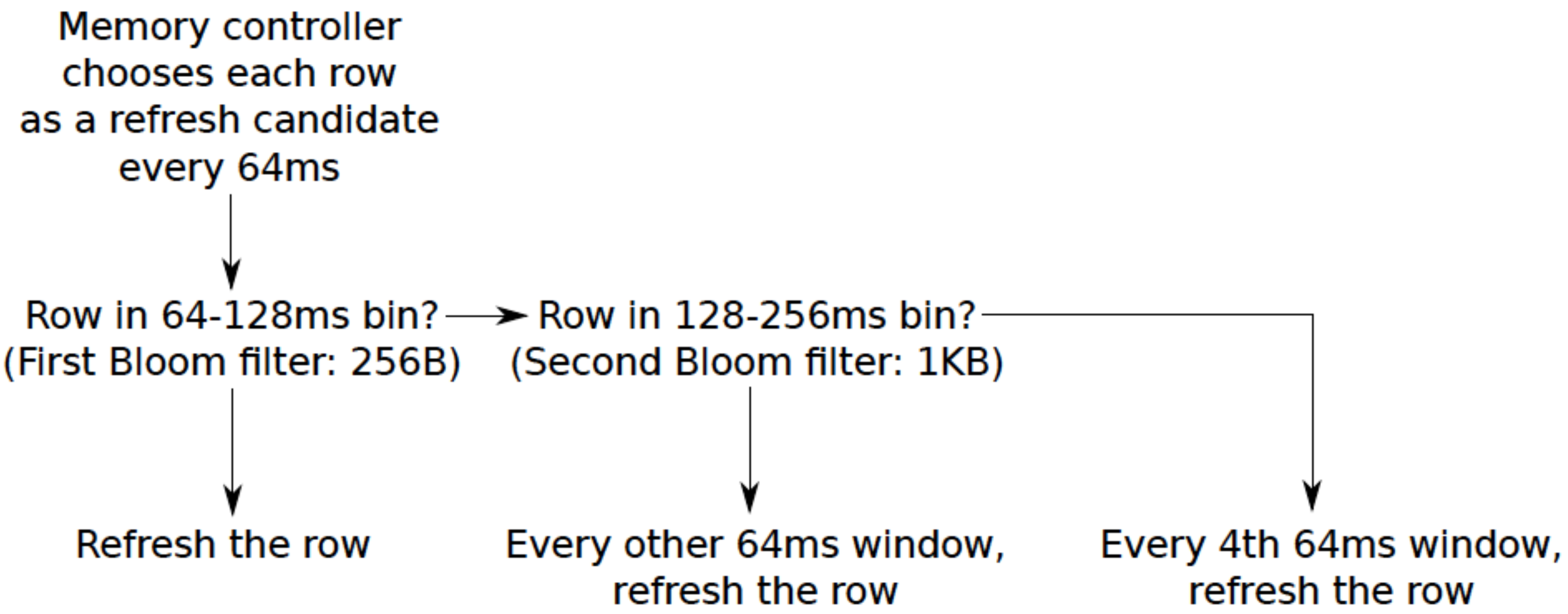


Choose a refresh candidate row

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.

Going Forward

- How to find out and expose weak memory cells/rows
  - Retention time profiling
  - Early analysis of modern DRAM chips:
    - Liu+, "An Experimental Study of Data Retention Behavior in Modern DRAM Devices: Implications for Retention Time Profiling Mechanisms", ISCA 2013.

- Tolerating cell-to-cell interference at the system level
  - Flash and DRAM

More on RAIDR and Refresh

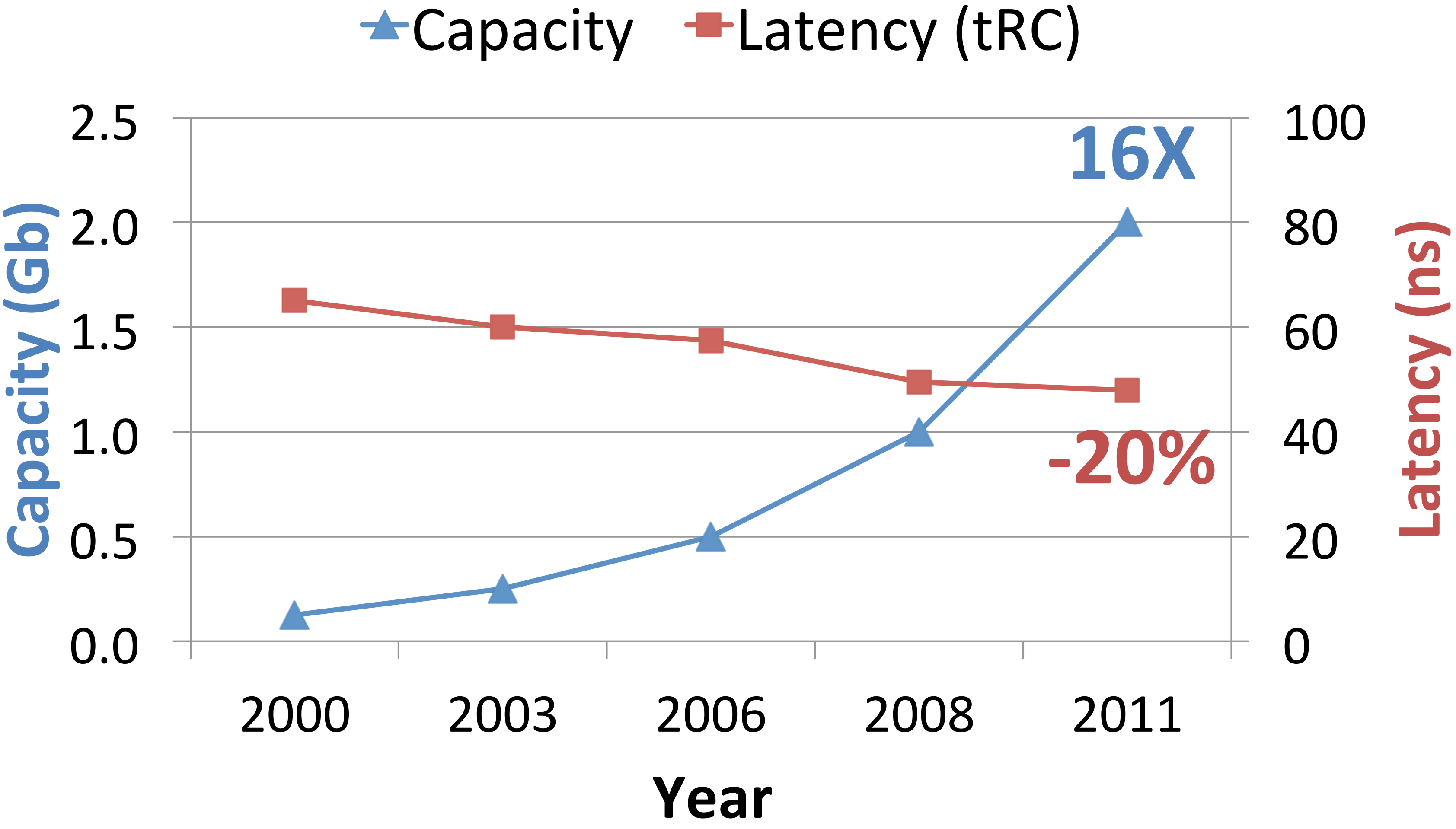
- Jamie Liu, Ben Jaiyen, Richard Veras, and Onur Mutlu, **"RAIDR: Retention-Aware Intelligent DRAM Refresh"** *Proceedings of the 39th International Symposium on Computer Architecture* (***ISCA***), Portland, OR, June 2012. Slides (pdf)

- 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)

Tolerating DRAM: Example Techniques

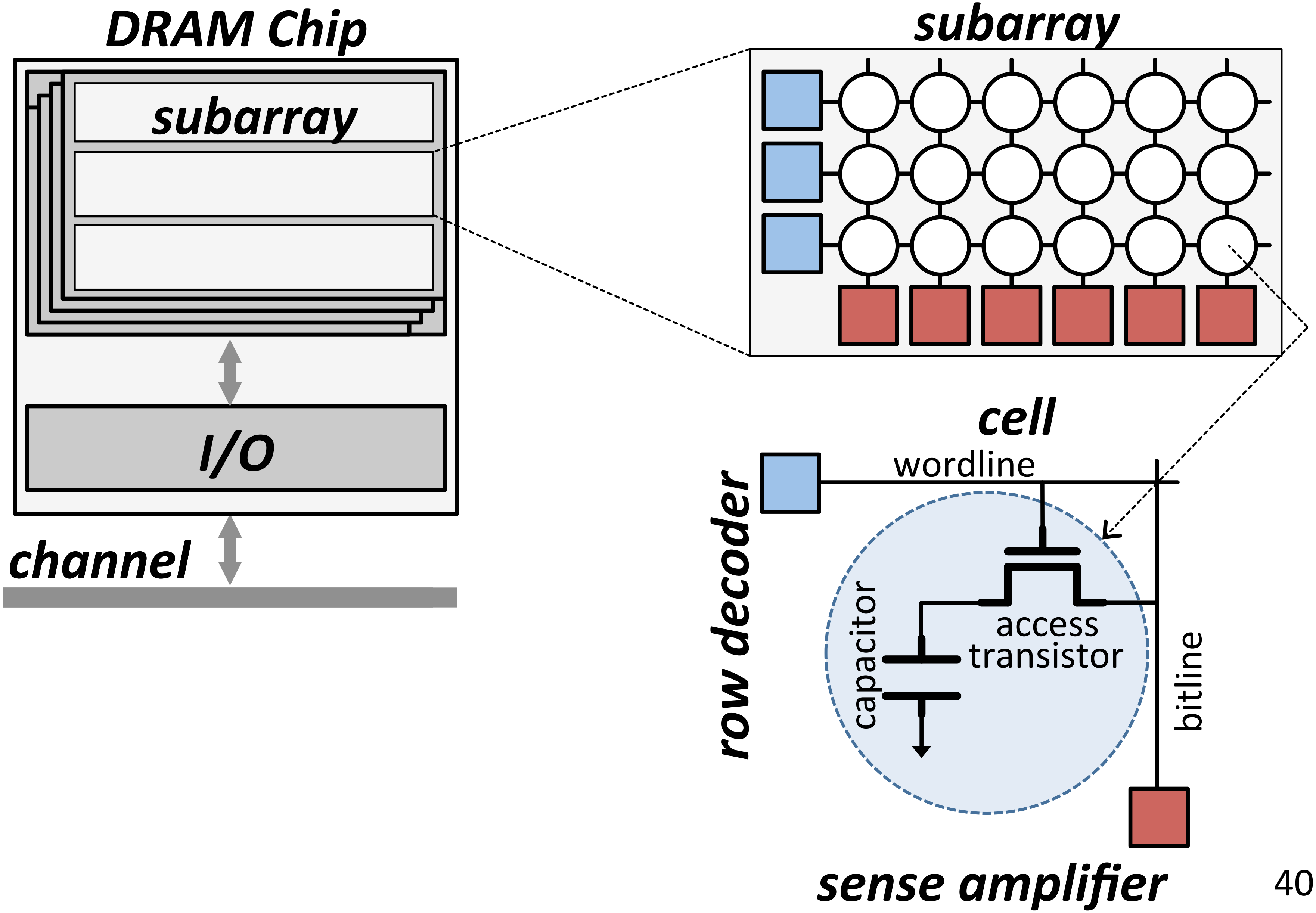
- Retention-Aware DRAM Refresh: Reducing Refresh Impact
- Tiered-Latency DRAM: Reducing DRAM Latency
- RowClone: In-Memory Page Copy and Initialization
- Subarray-Level Parallelism: Reducing Bank Conflict Impact

DRAM Latency-Capacity Trend

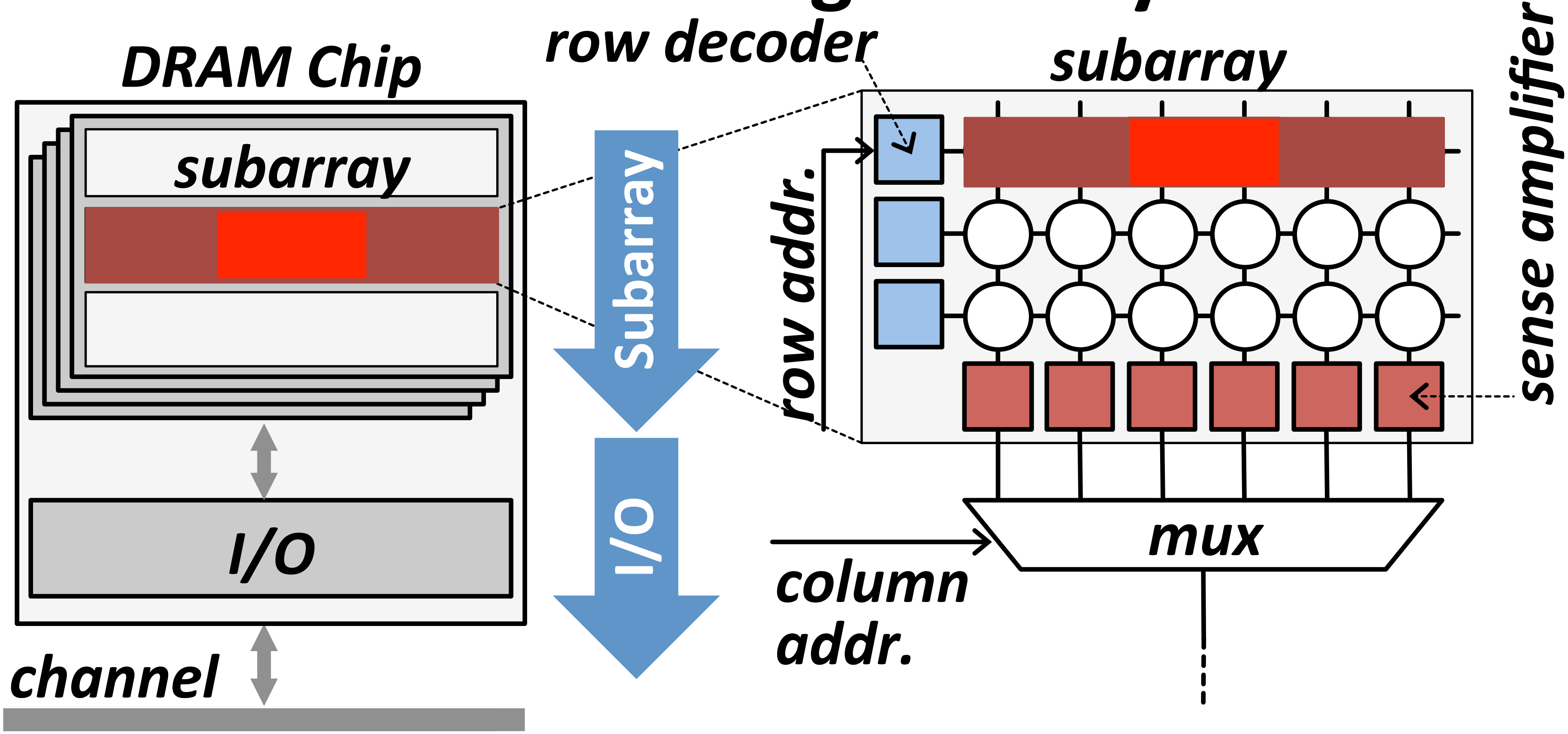


DRAM latency continues to be a critical bottleneck

What Causes the Long Latency?



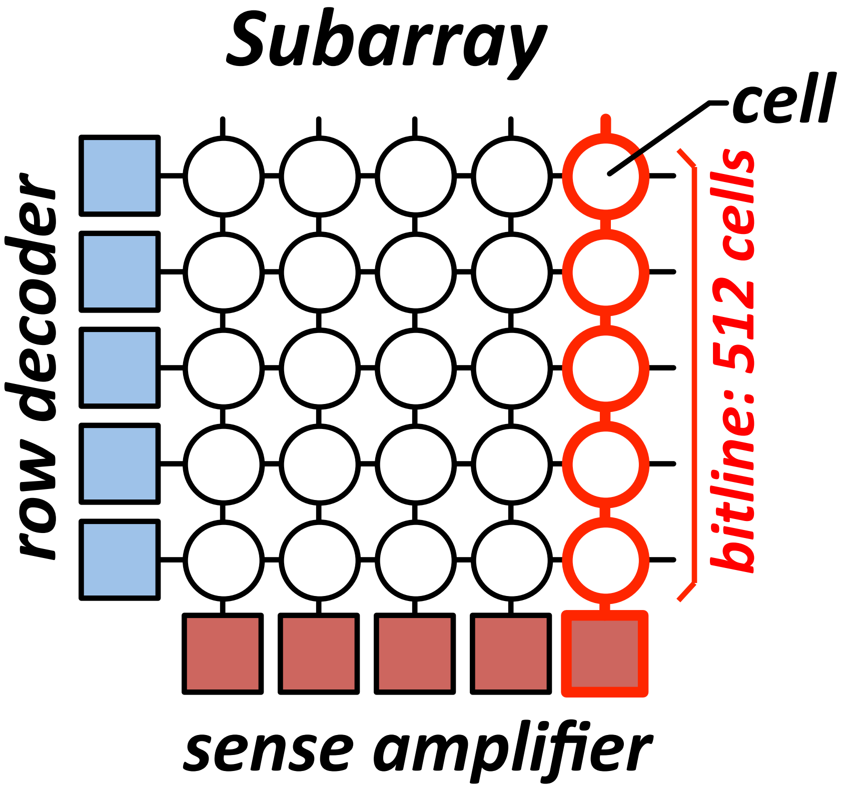
What Causes the Long Latency?

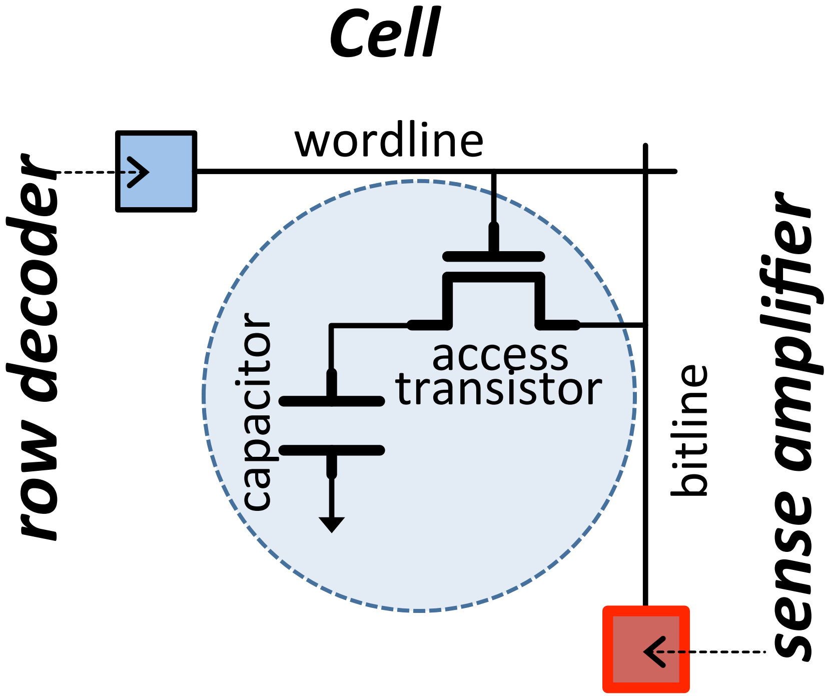


DRAM Latency = Subarray Latency + I/O Latency

Dominant

Why is the Subarray So Slow?

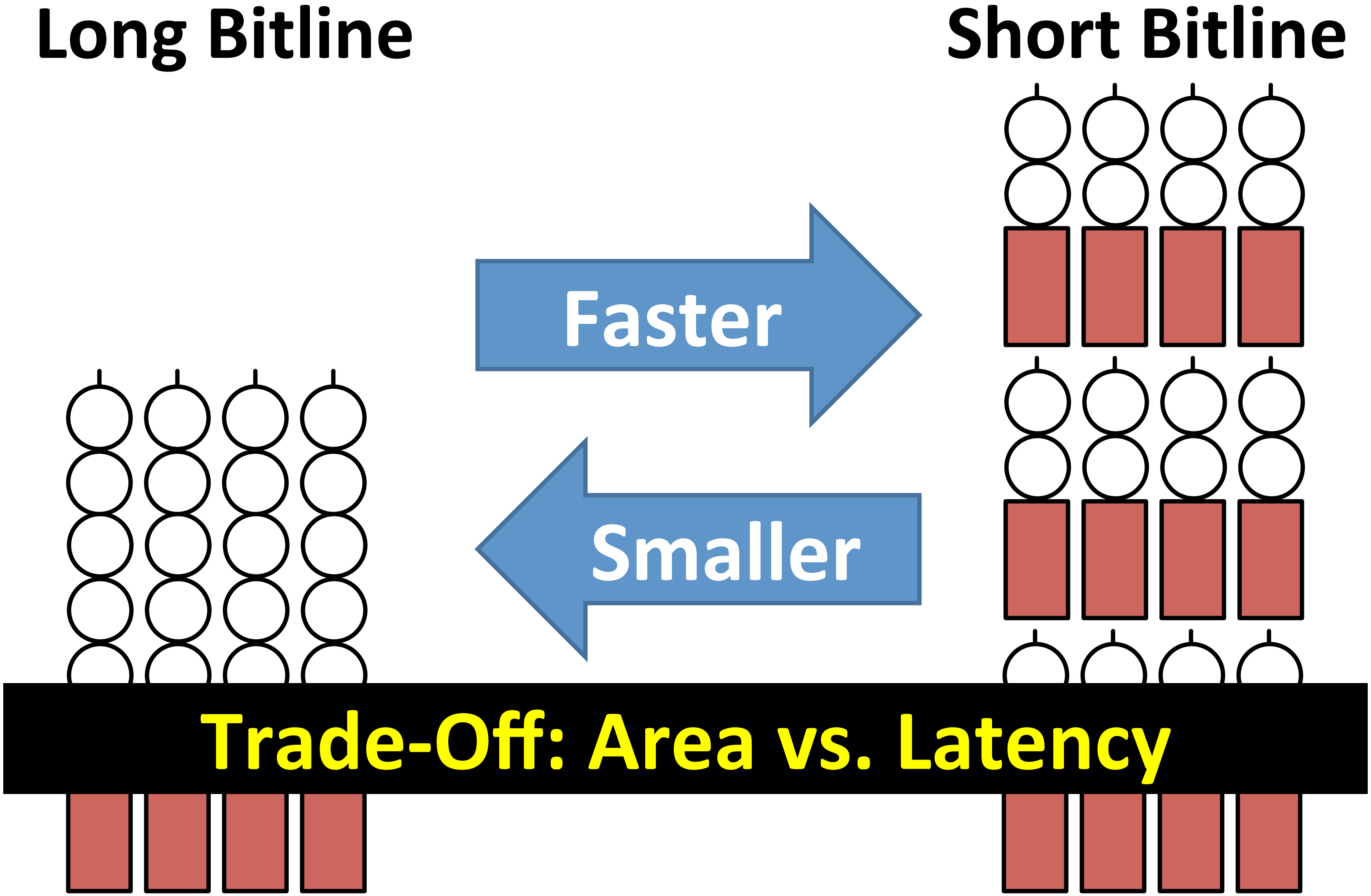




large sense amplifier

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

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



Long Bitline

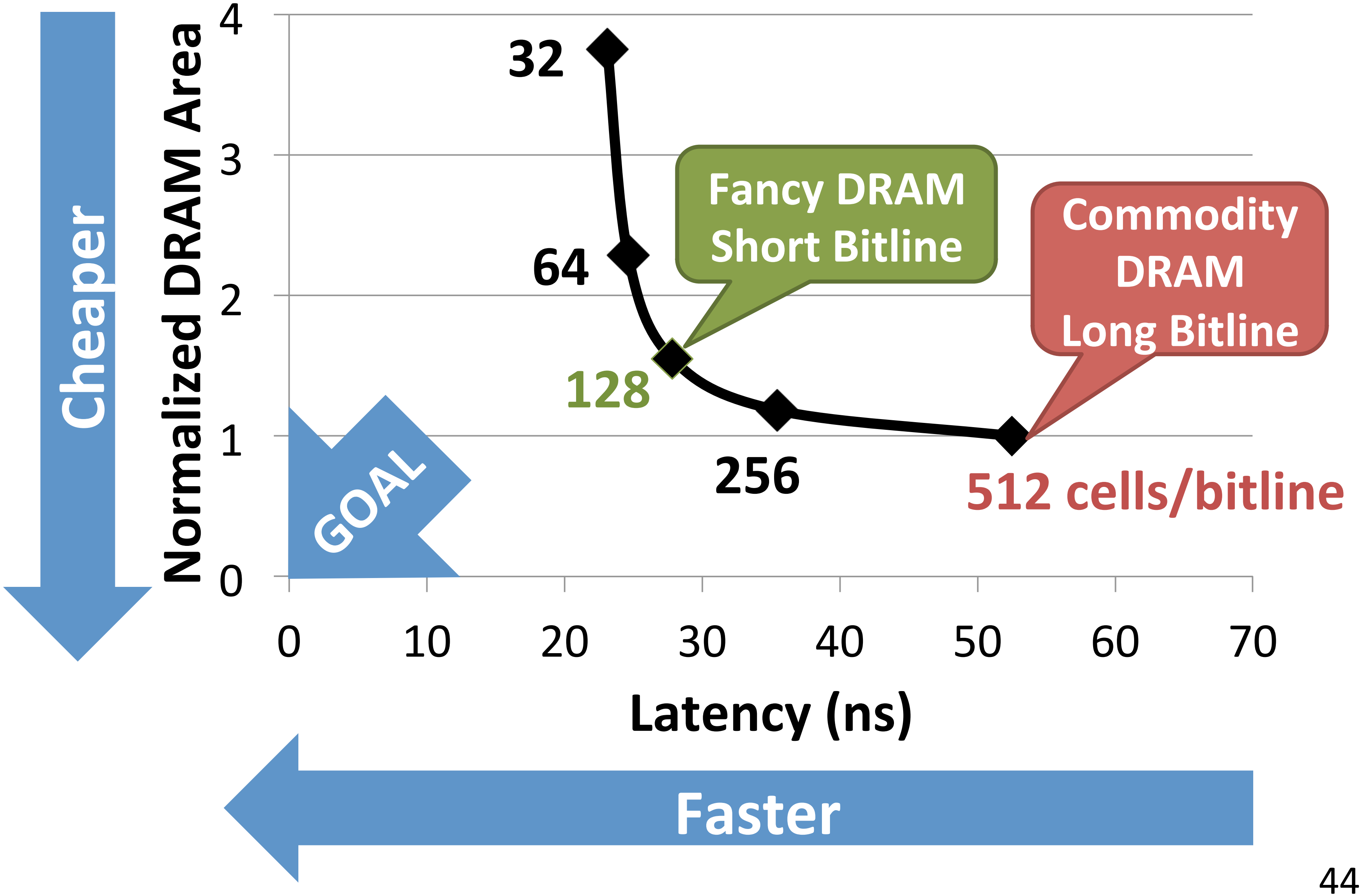
Short Bitline

Faster

Smaller

Trade-Off: Area vs. Latency

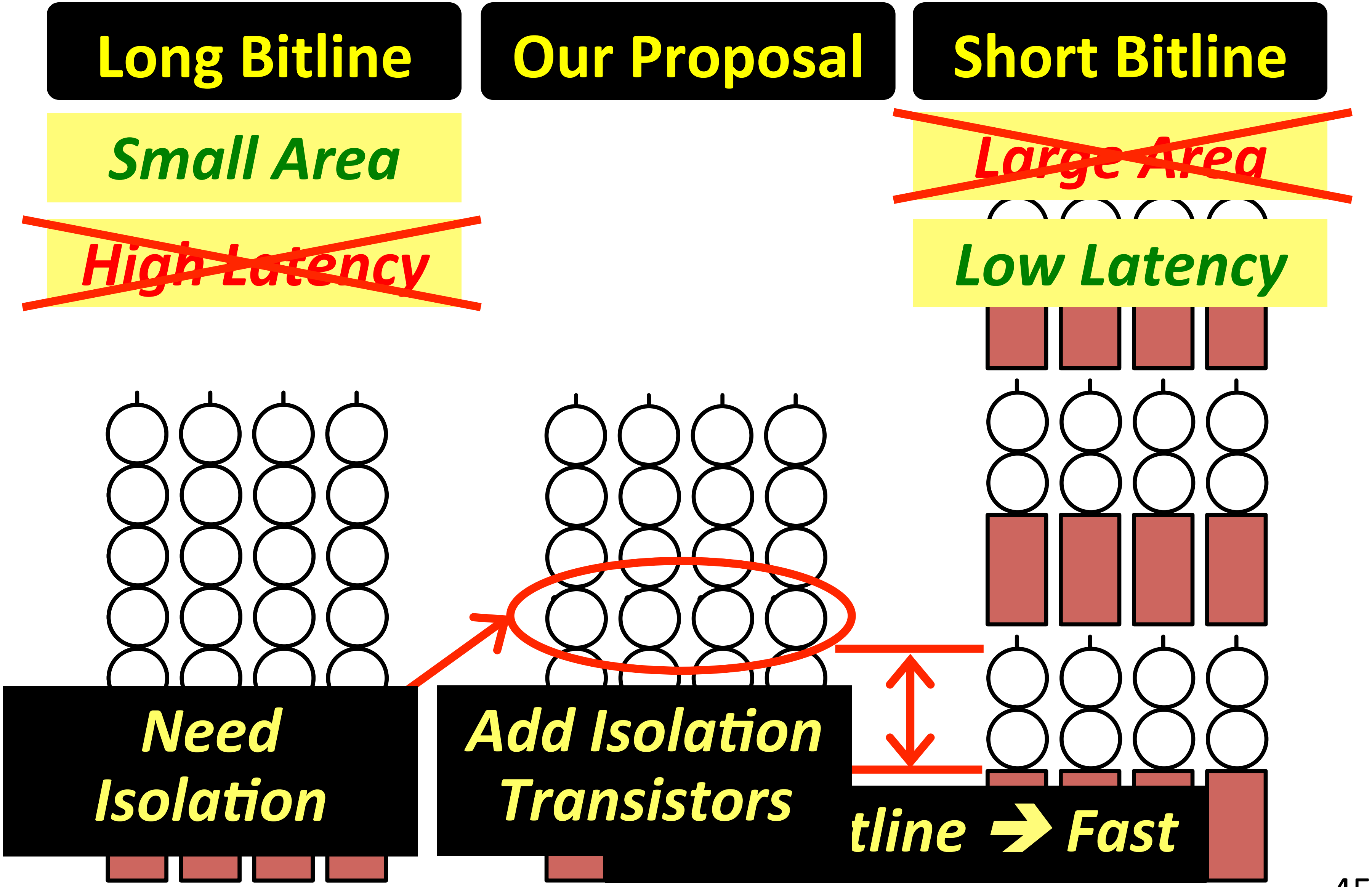
Trade-Off: Area (Die Size) vs. Latency



Fancy DRAM Short Bitline

Commodity DRAM Long Bitline

Approximating the Best of Both Worlds



Long Bitline

Small Area

~~High Latency~~

Our Proposal

Short Bitline

~~Large Area~~

Low Latency

Need Isolation

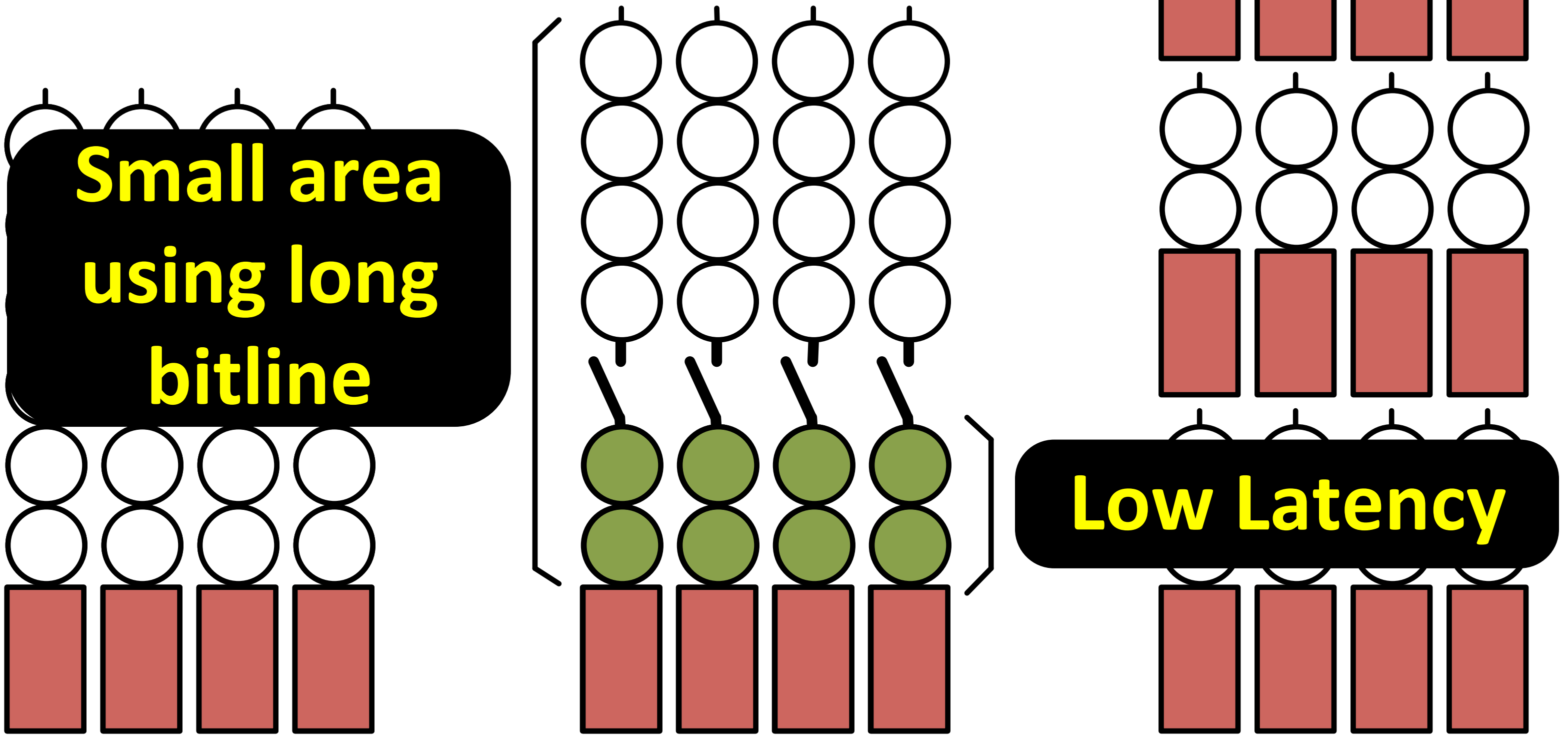
Add Isolation Transistors

tline → Fast

Approximating the Best of Both Worlds

Long Bitlin Tiered-Latency DRAM hort Bitline

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

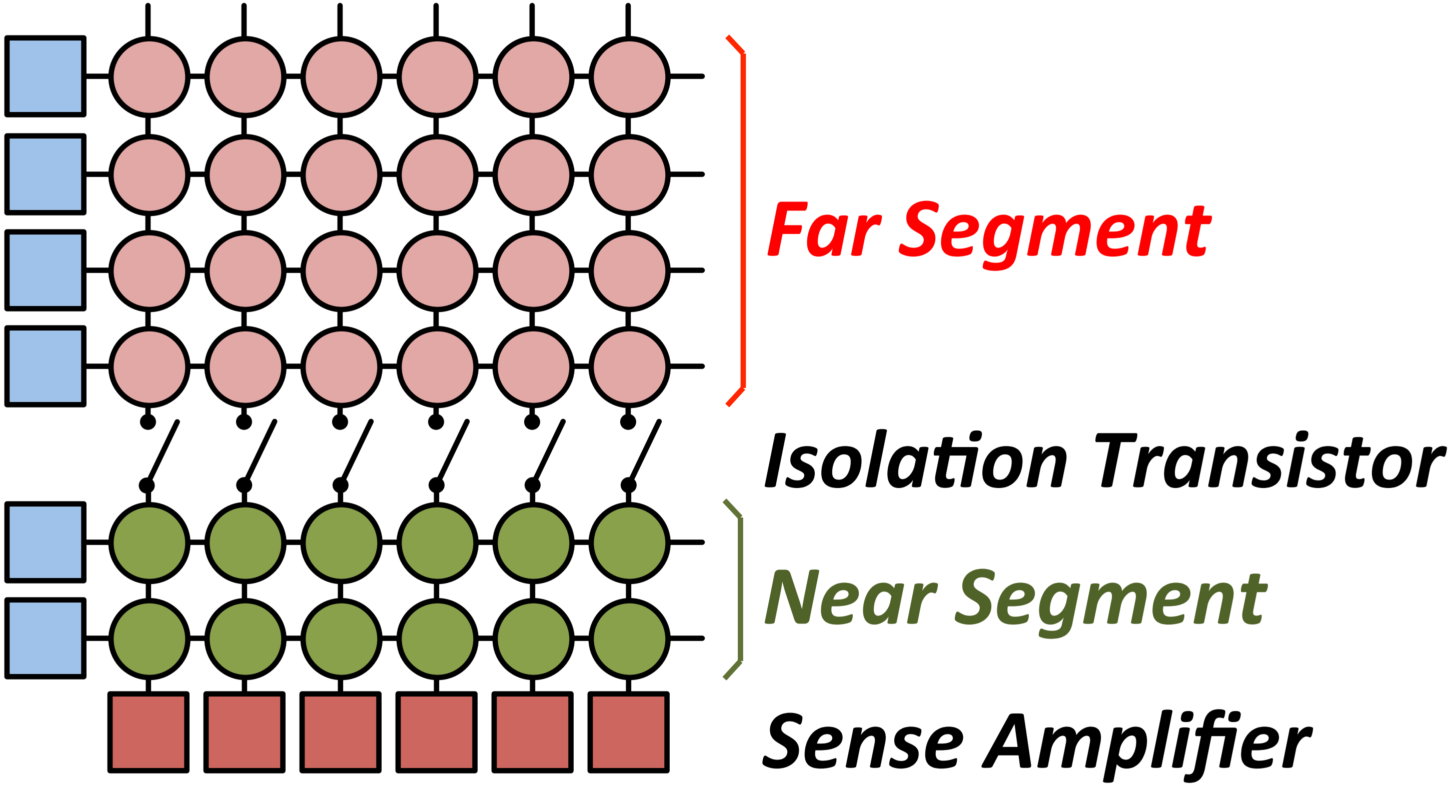


Small area using long bitline

Low Latency

Tiered-Latency DRAM

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



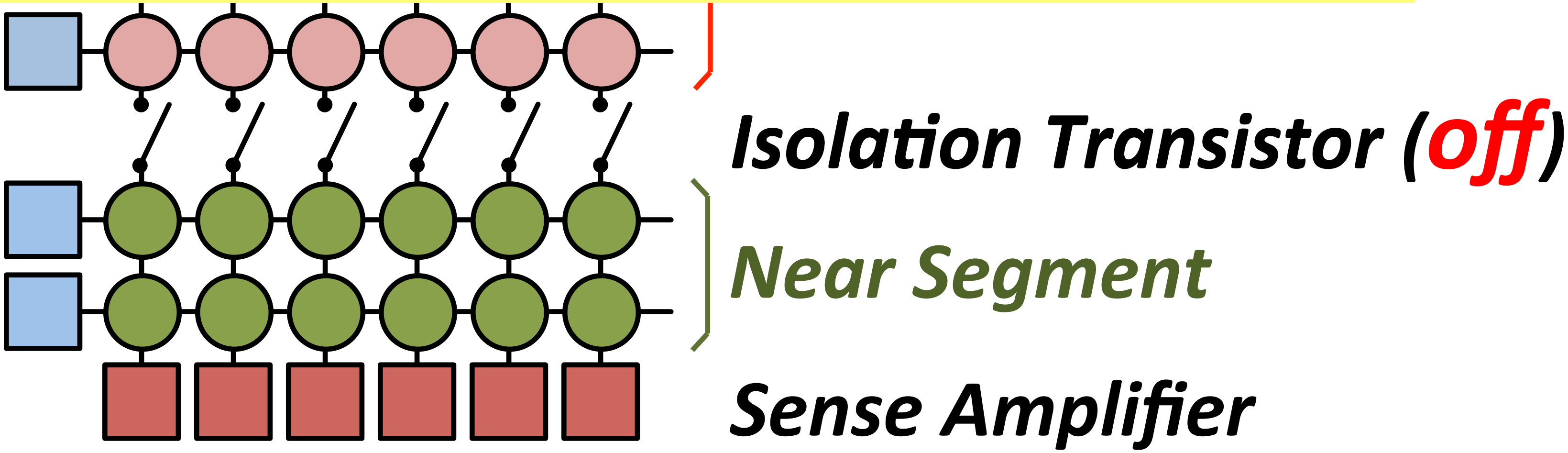
Near Segment Access

- Turn *off* the isolation transistor

Reduced bitline length

Reduced bitline capacitance

➔ Low latency & low power



Far Segment Access

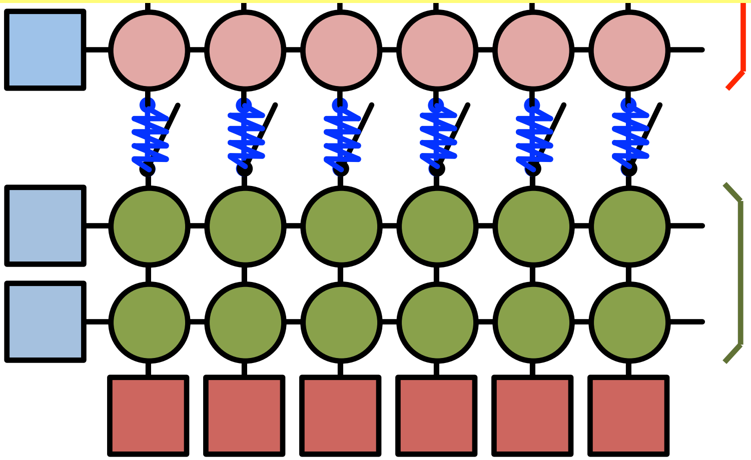
- Turn *on* the isolation transistor

Long bitline length

Large bitline capacitance

Additional resistance of isolation transistor

→ High latency & high power



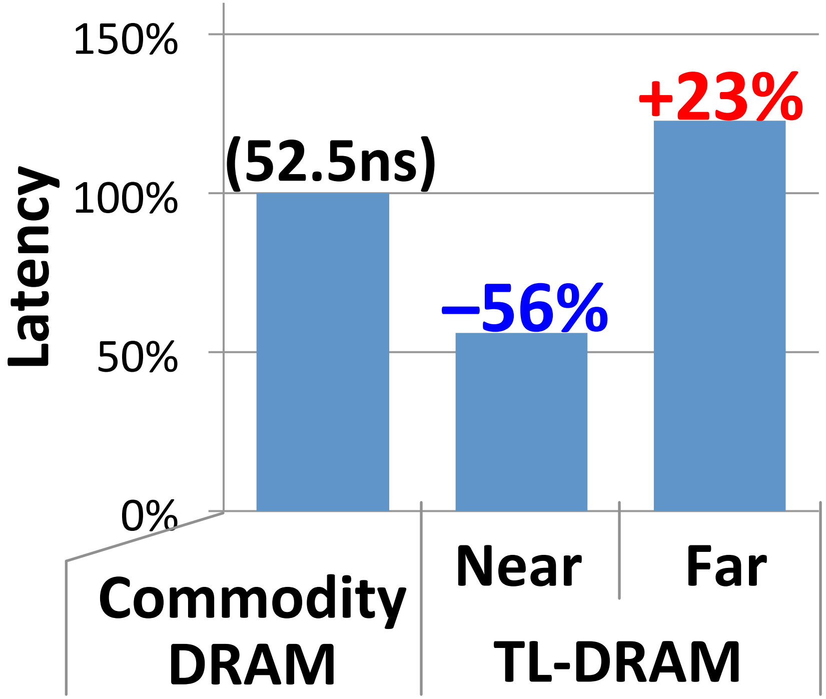
Isolation Transistor (on)

Near Segment

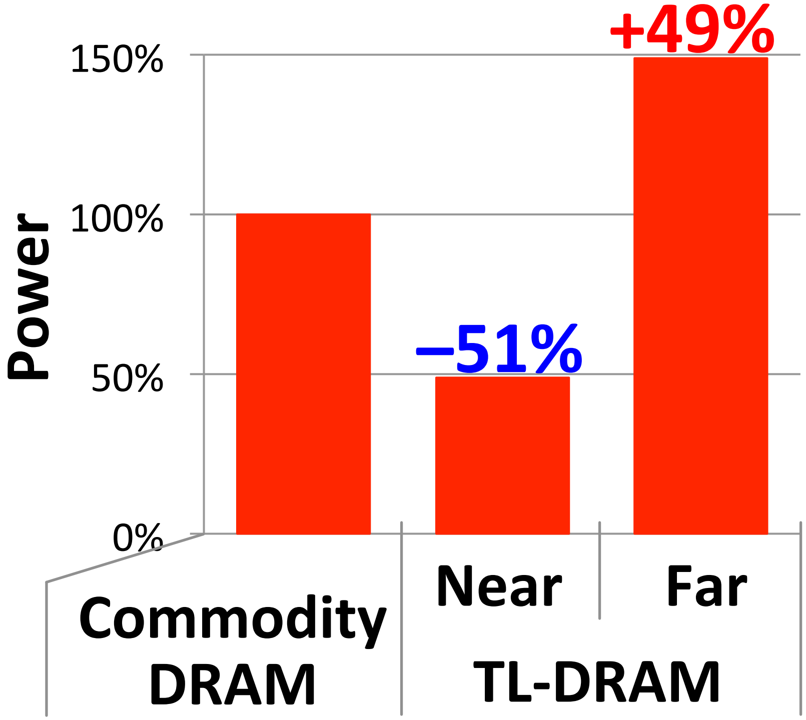
Sense Amplifier

Commodity DRAM vs. TL-DRAM

- **DRAM Latency** (tRC)



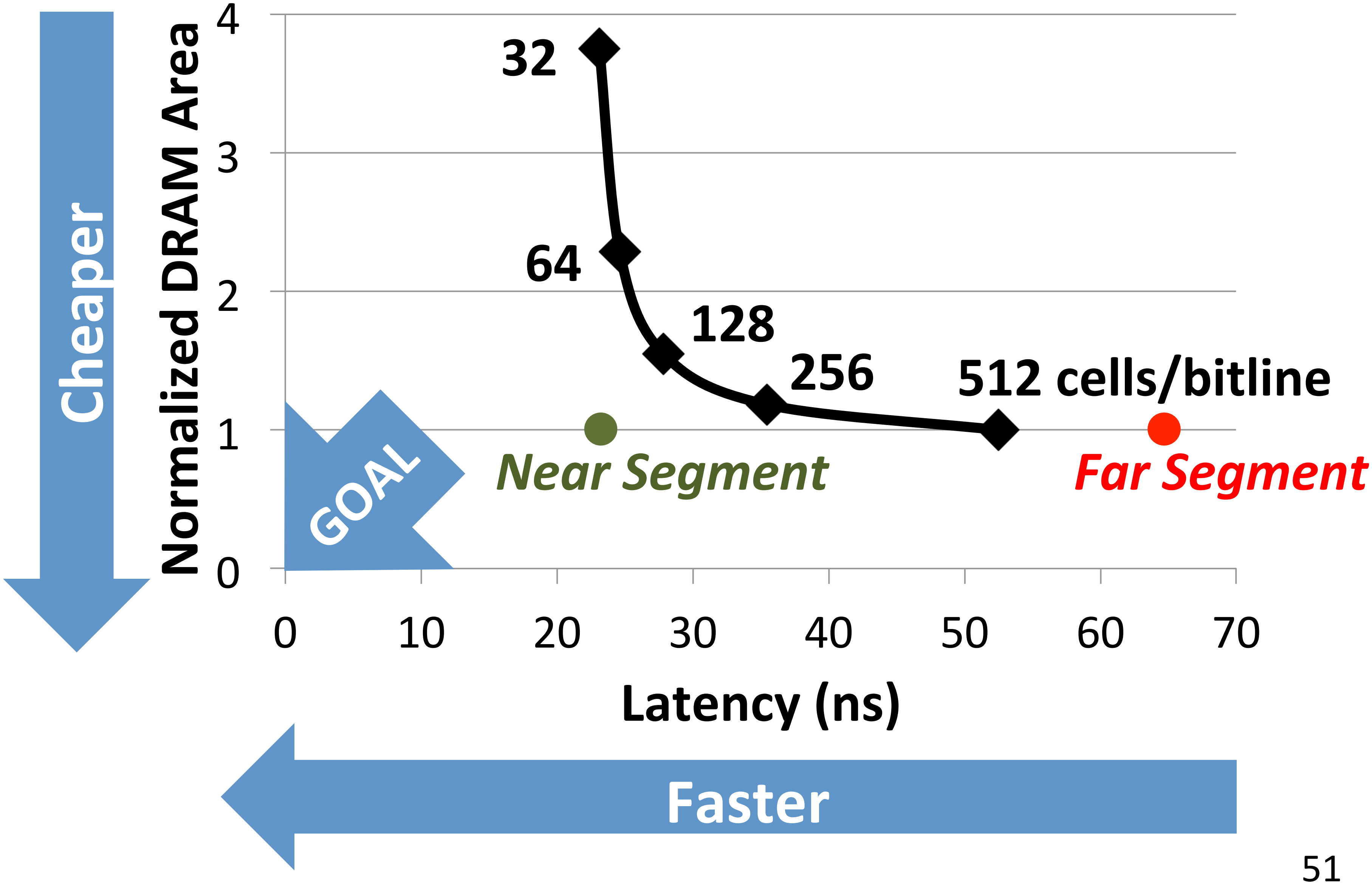
- **DRAM Power**



- **DRAM Area Overhead**

  ~3%: mainly due to the isolation transistors

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



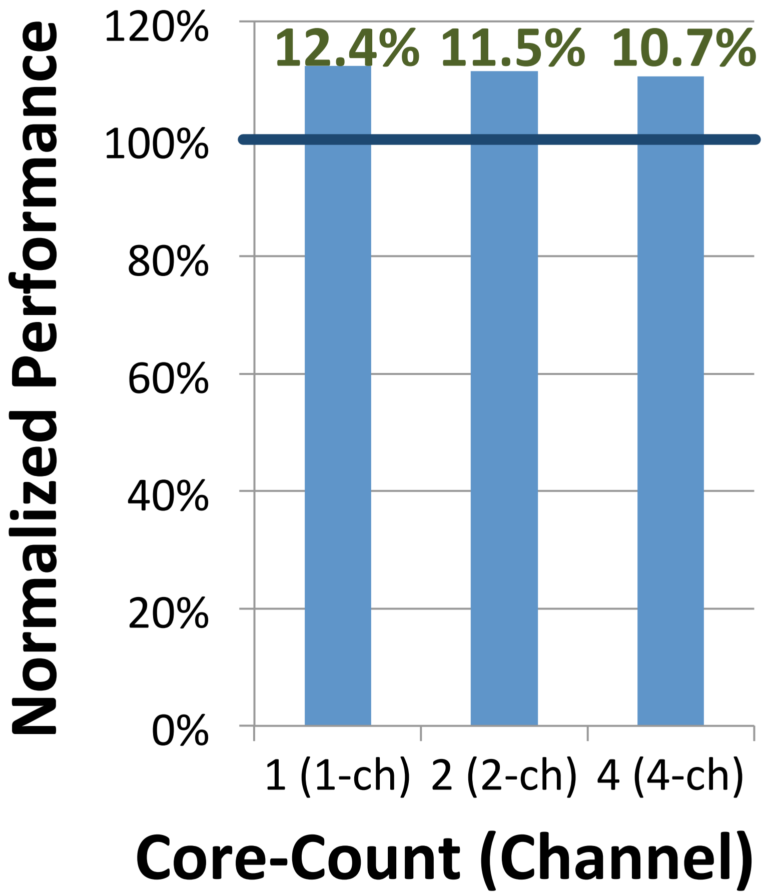
Leveraging Tiered-Latency DRAM

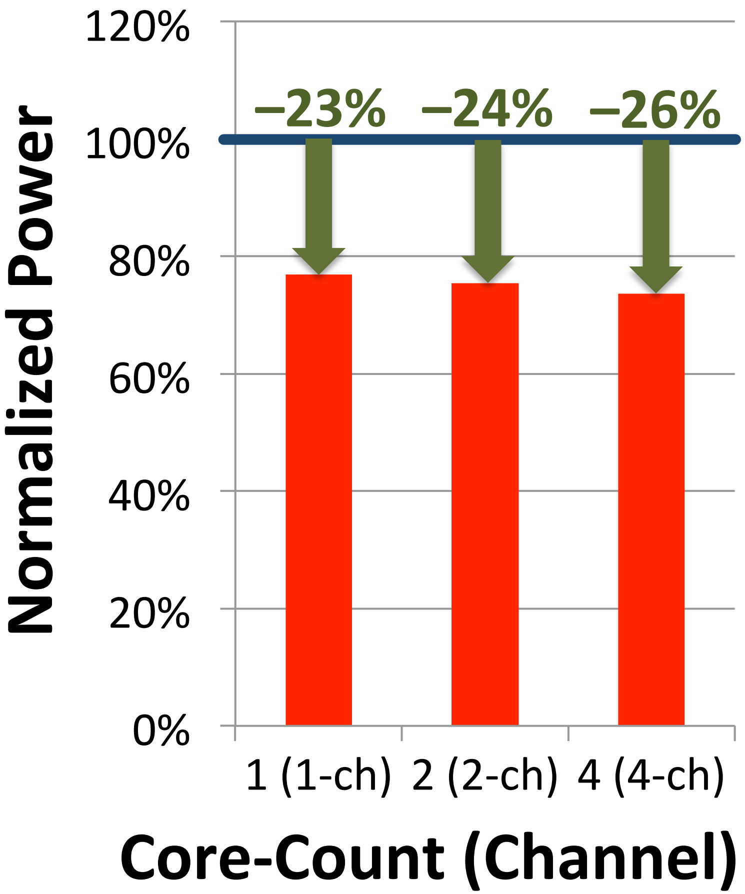
- TL-DRAM is a ***substrate*** that can be leveraged by the hardware and/or software

- 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

Performance & Power Consumption





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

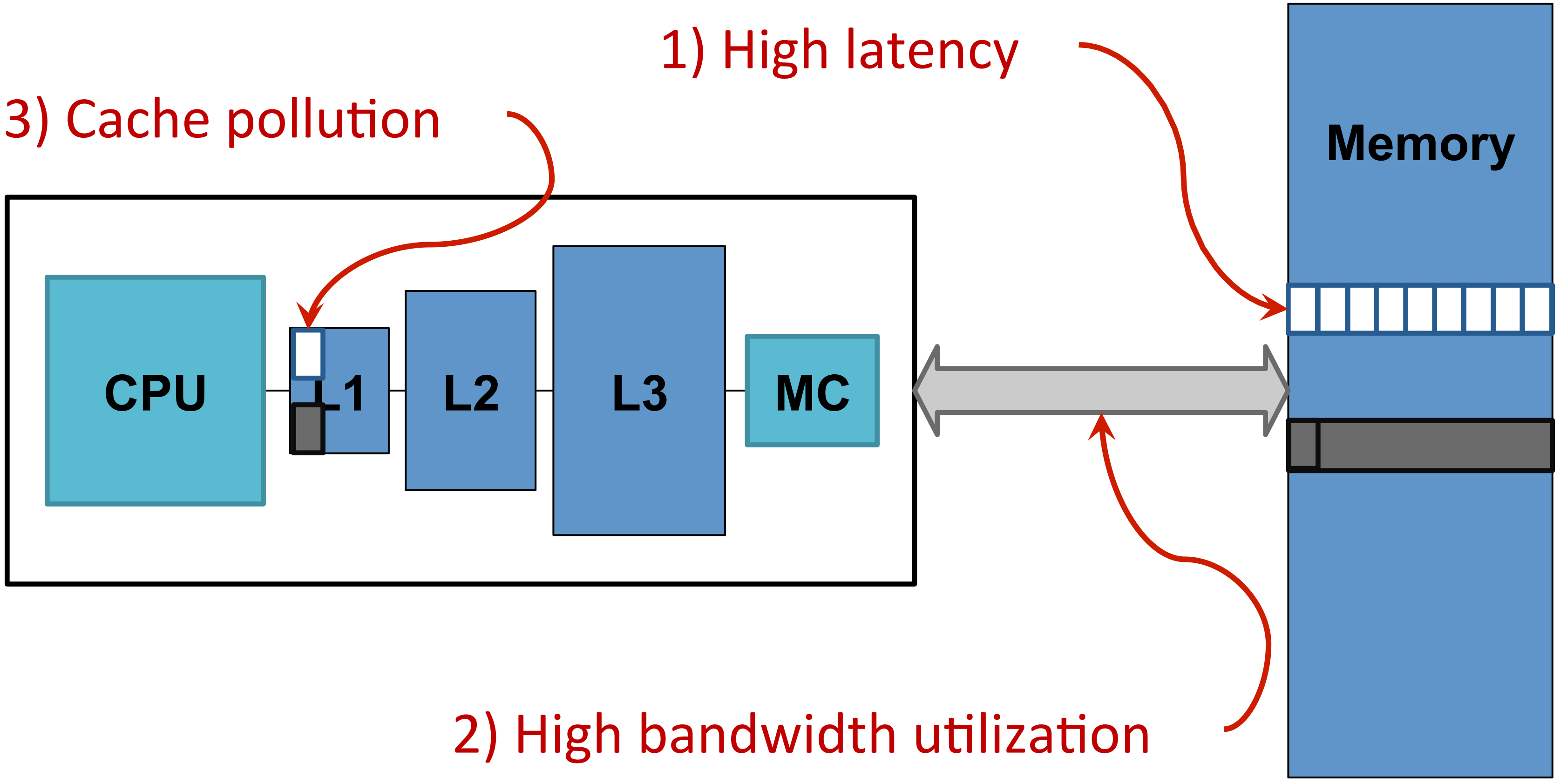
More on TL-DRAM

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

Tolerating DRAM: Example Techniques

- Retention-Aware DRAM Refresh: Reducing Refresh Impact
- Tiered-Latency DRAM: Reducing DRAM Latency
- RowClone: In-Memory Page Copy and Initialization
- Subarray-Level Parallelism: Reducing Bank Conflict Impact

Today's Memory: Bulk Data Copy



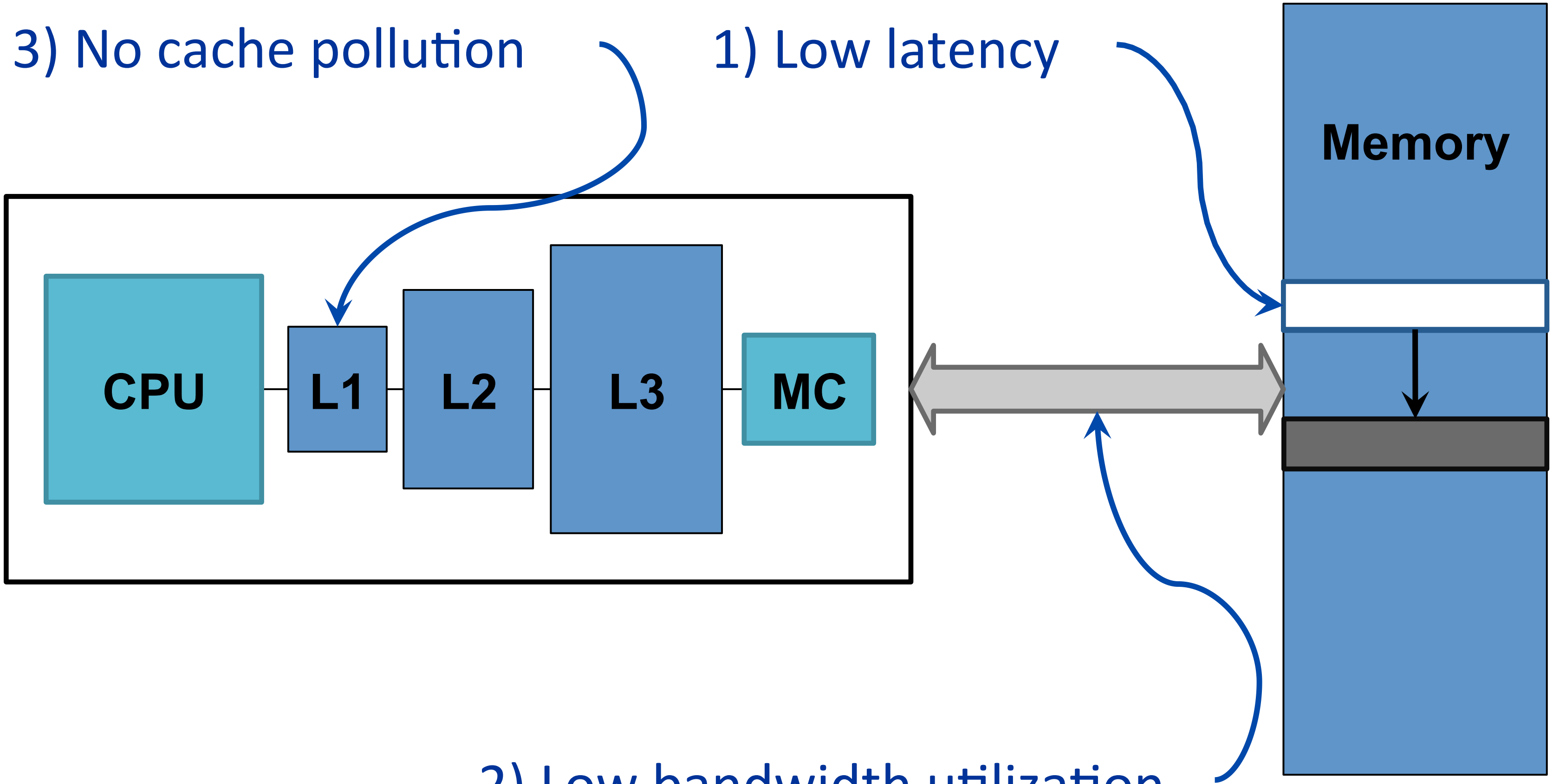
1) High latency

3) Cache pollution

2) High bandwidth utilization

4) Unwanted data movement

Future: RowClone (In-Memory Copy)



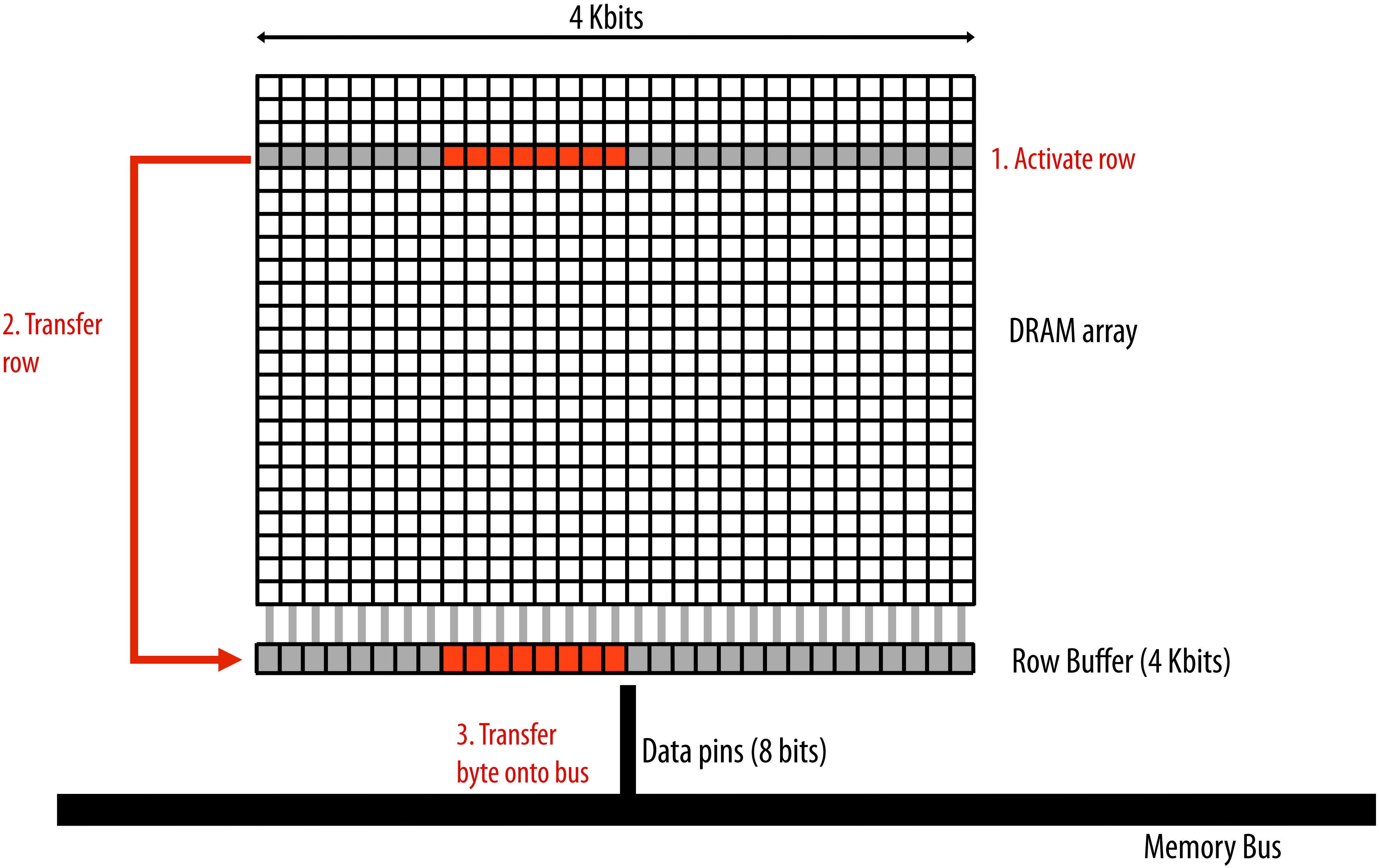
3) No cache pollution

1) Low latency

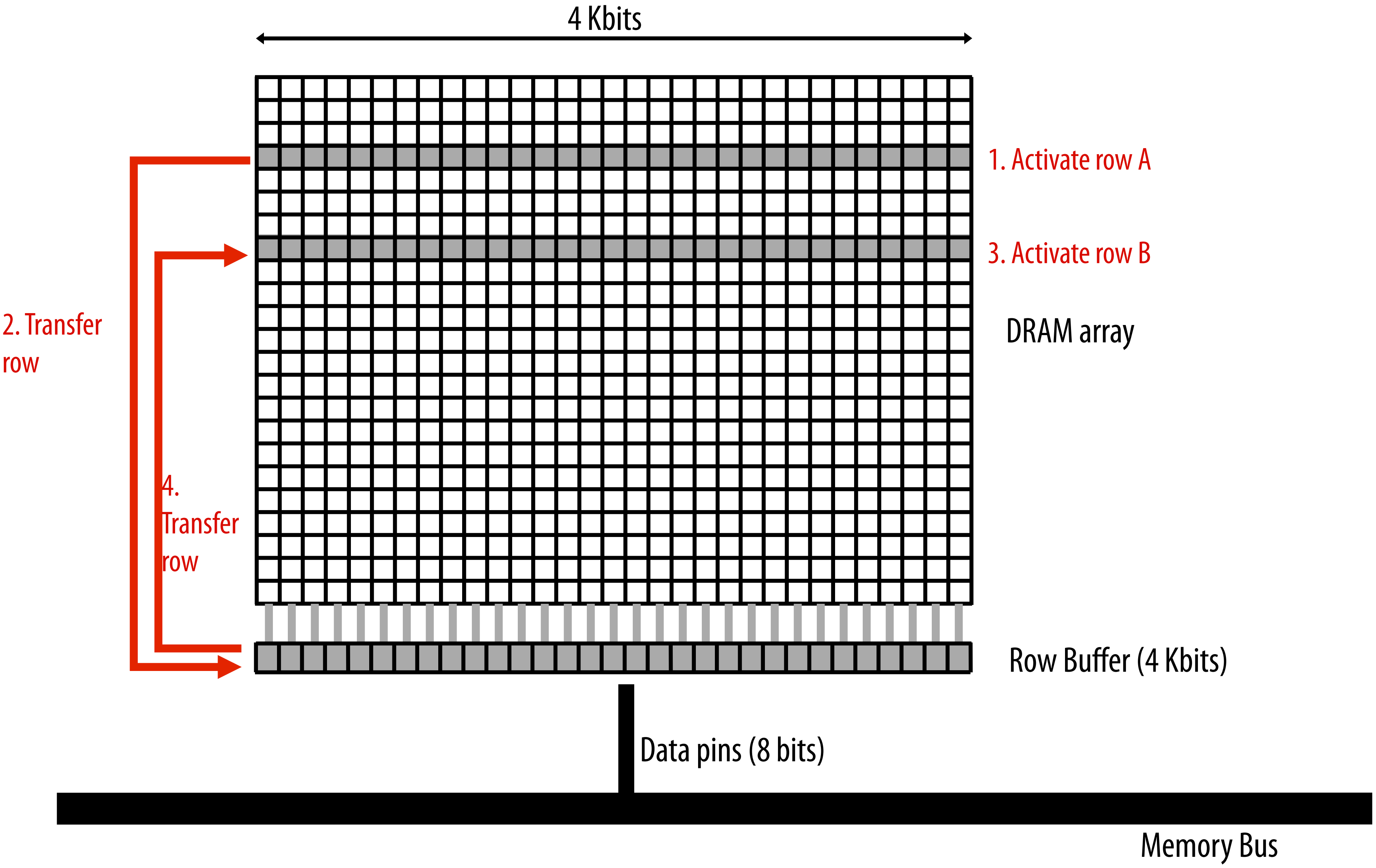
2) Low bandwidth utilization

4) No unwanted data movement

DRAM operation (load one byte)



RowClone: in-DRAM Row Copy (and Initialization)



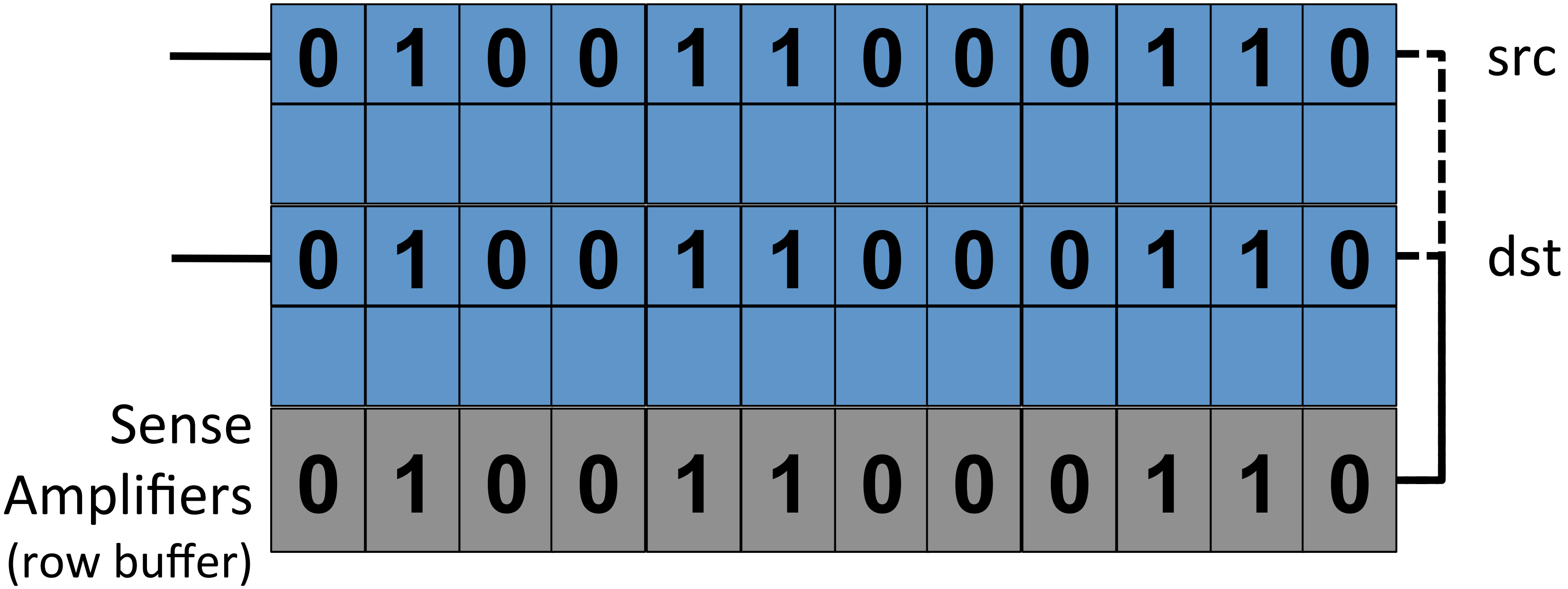
RowClone: Key Idea

- 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

Can be accomplished by two consecutive ACTIVATEs
(if source and destination rows are in the same subarray)

RowClone: Intra-subarray Copy



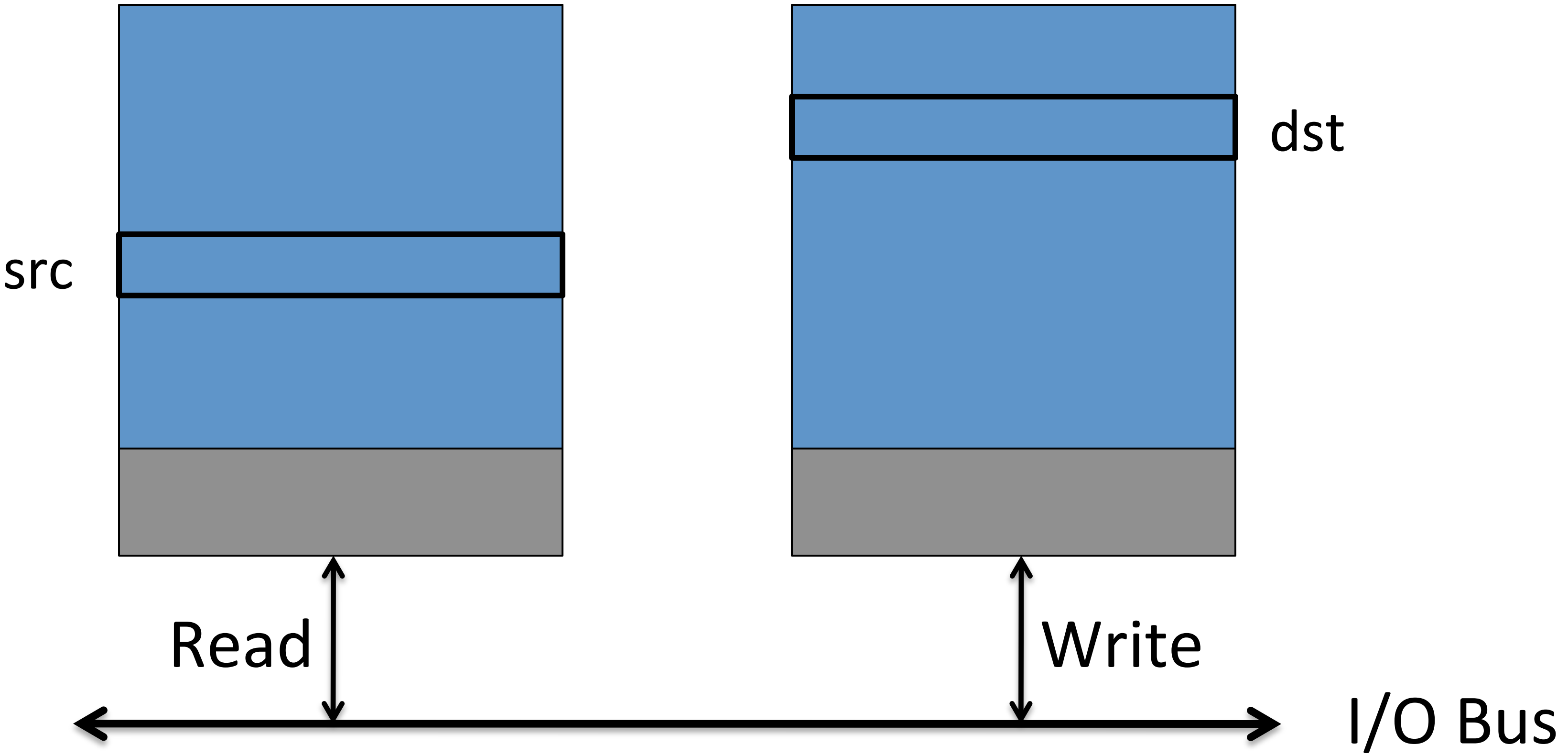
0 1 0 0 1 1 0 0 0 1 1 0

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

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

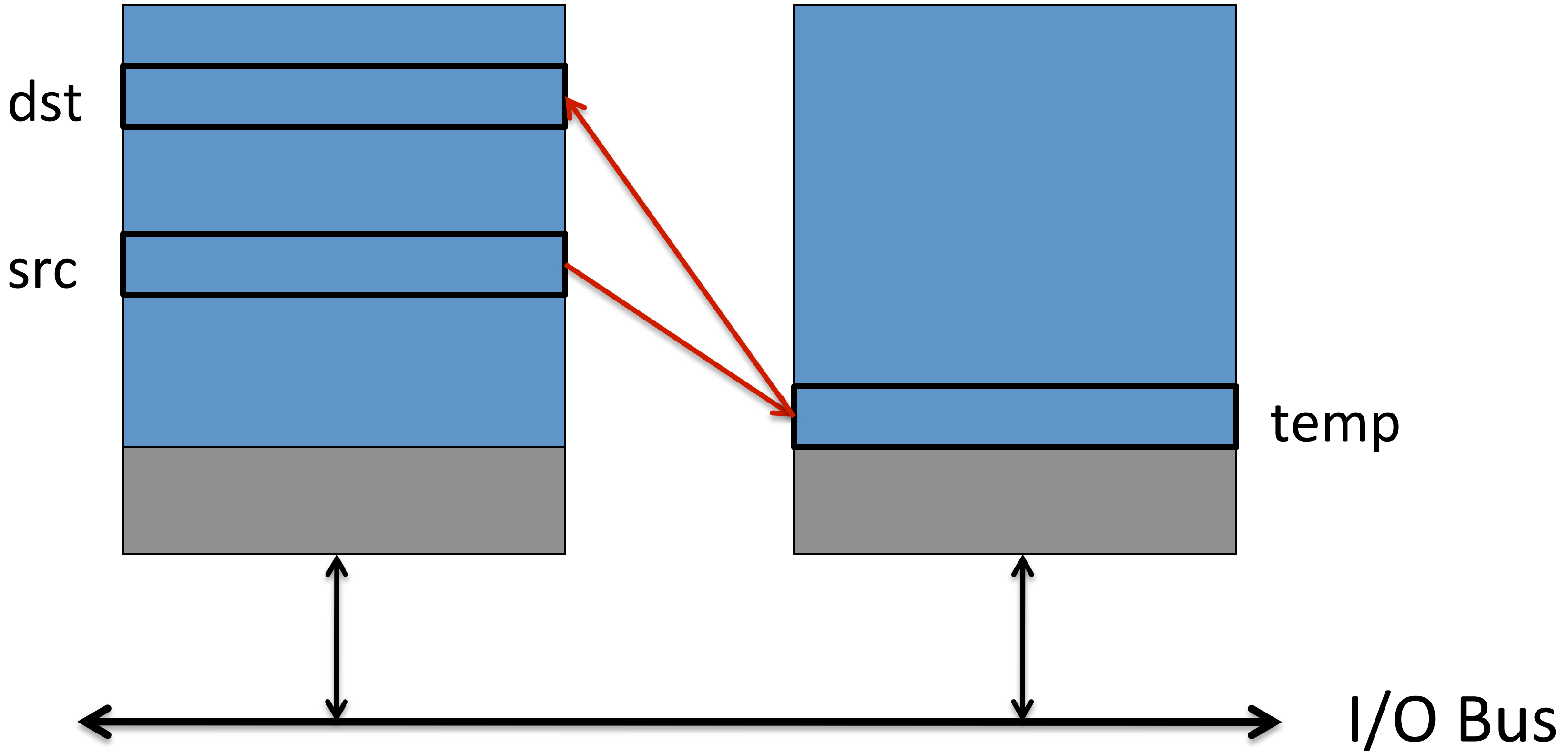
Activate (src) → Deactivate (our proposal) → Activate (dst)

RowClone: Inter-bank Copy



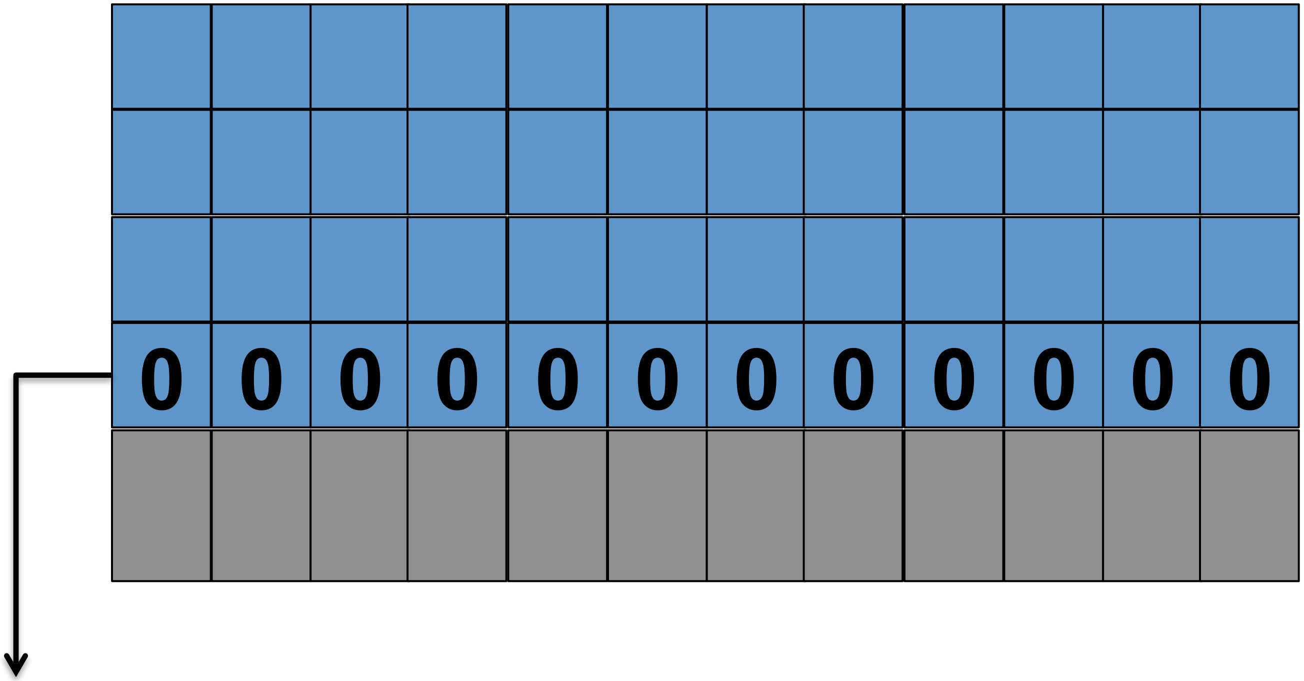
Transfer
(our proposal)

RowClone: Inter-subarray Copy



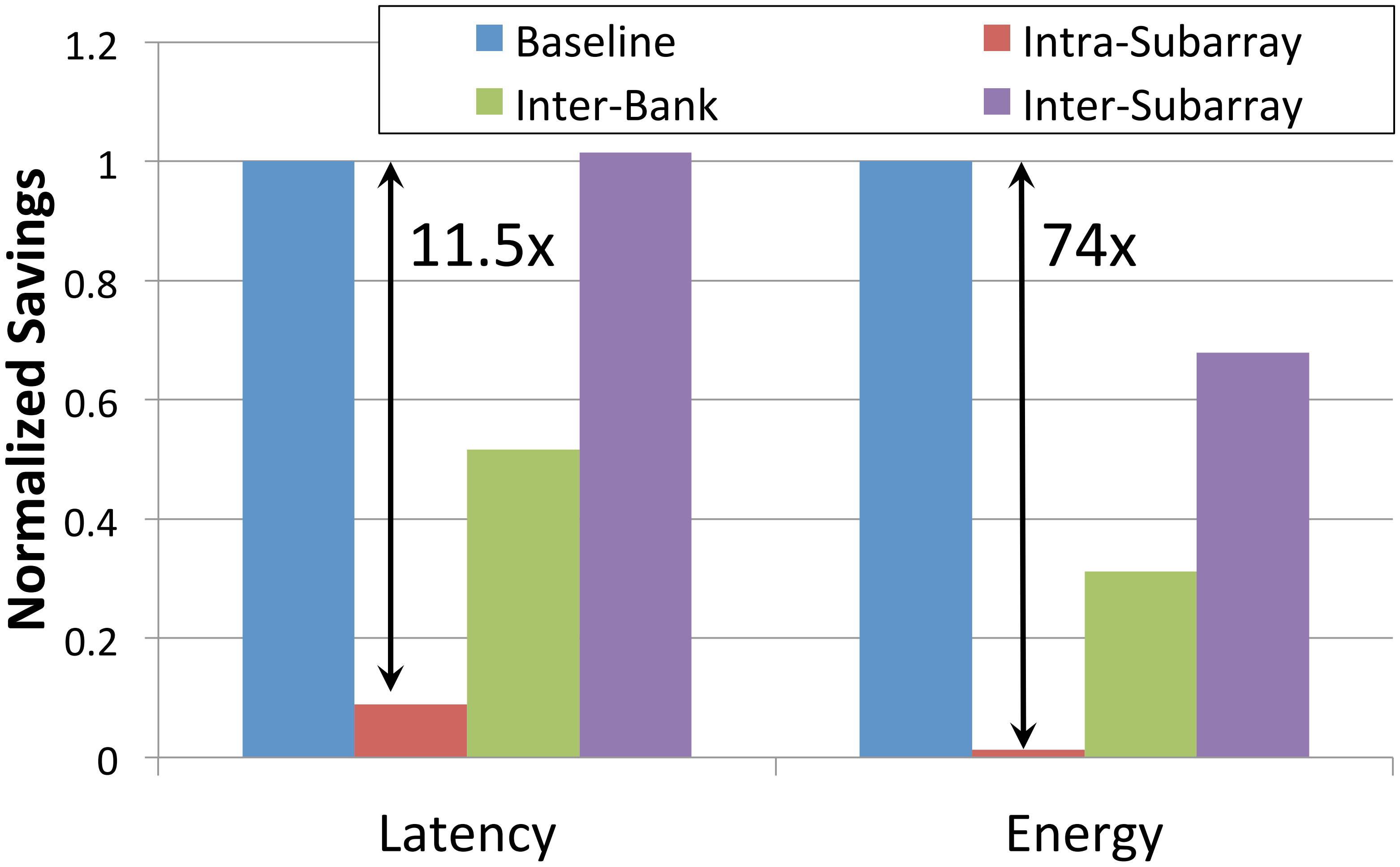
1. Transfer (src to temp)
2. Transfer (temp to dst)

Fast Row Initialization



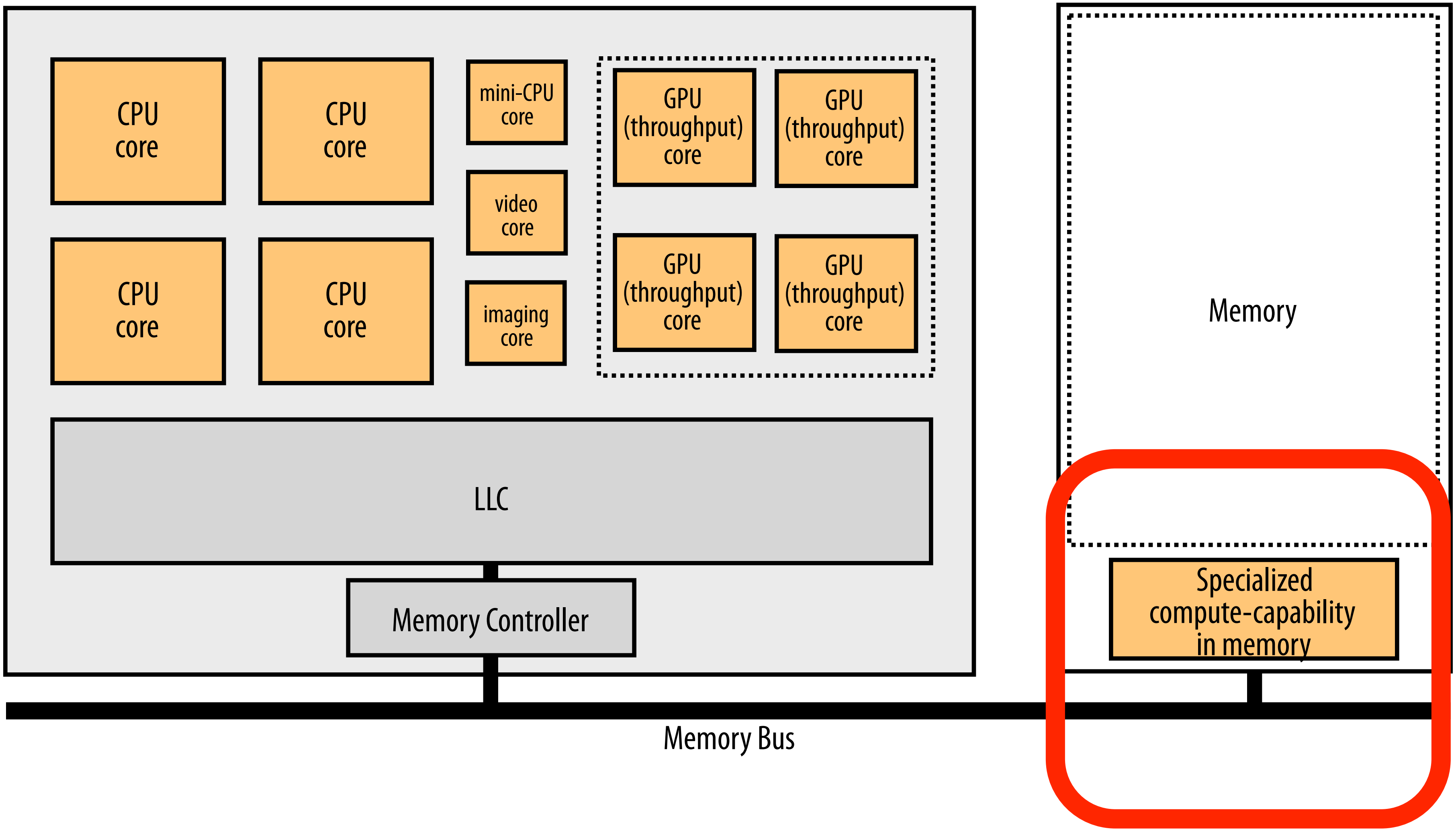
Fix a row at Zero
(0.5% loss in capacity)

RowClone: Latency and Energy Savings



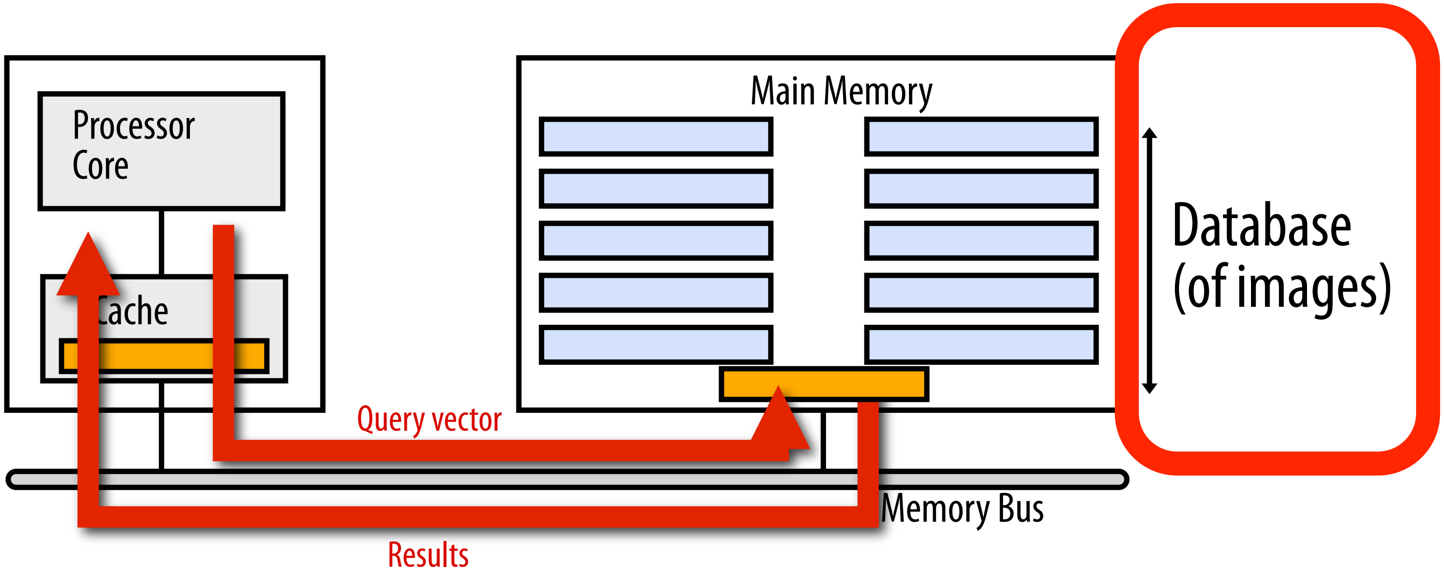
Seshadri et al., "RowClone: Fast and Efficient In-DRAM Copy and Initialization of Bulk Data," CMU Tech Report 2013.

Goal: Ultra-efficient heterogeneous architectures



Slide credit: Prof. Kayvon Fatahalian, CMU

Enabling Ultra-efficient (Visual) Search



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

Picture credit: Prof. Kayvon Fatahalian, CMU

More on RowClone

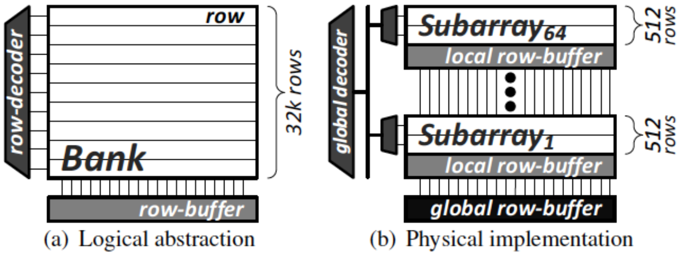
- Vivek Seshadri, Yoongu Kim, Chris Fallin, Donghyuk Lee, Rachata Ausavarungnirun, Gennady Pekhimenko, Yixin Luo, Onur Mutlu, Phillip B. Gibbons, Michael A. Kozuch, Todd C. Mowry,
"RowClone: Fast and Efficient In-DRAM Copy and Initialization of Bulk Data"
CMU Computer Science Technical Report, CMU-CS-13-108,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

Tolerating DRAM: Example Techniques

- Retention-Aware DRAM Refresh: Reducing Refresh Impact
- Tiered-Latency DRAM: Reducing DRAM Latency
- RowClone: In-Memory Page Copy and Initialization
- Subarray-Level Parallelism: Reducing Bank Conflict Impact

SALP: Reducing DRAM Bank Conflicts

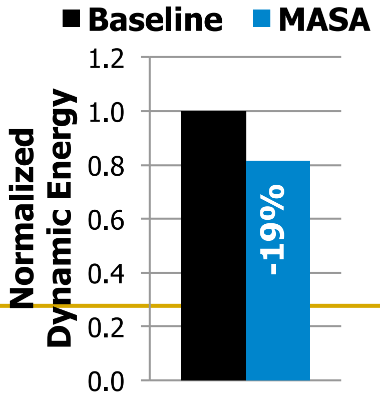
- Problem: Bank conflicts are costly for performance and energy
  - serialized requests, wasted energy (thrashing of row buffer, busy wait)
- Goal: Reduce bank conflicts without adding more banks (low cost)
- Key idea: Exploit the internal subarray structure of a DRAM bank to parallelize bank conflicts to different subarrays
  - Slightly modify DRAM bank to reduce subarray-level hardware sharing

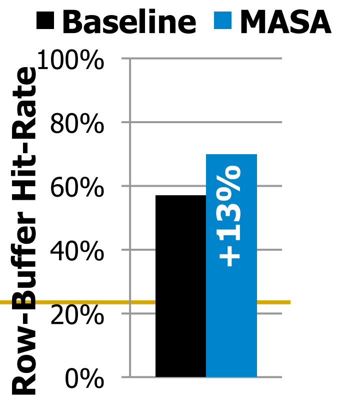


(a) Logical abstraction (b) Physical implementation

Figure 1. DRAM bank organization

- Results on Server, Stream/Random, SPEC
  - 19% reduction in dynamic DRAM energy
  - 13% improvement in row hit rate
  - 17% performance improvement
  - 0.15% DRAM area overhead





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

More on SALP

- 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)

Tolerating DRAM: Summary

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

Further Reading: Data Retention and Power

- Characterization of Commodity DRAM Chips
  - 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)

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

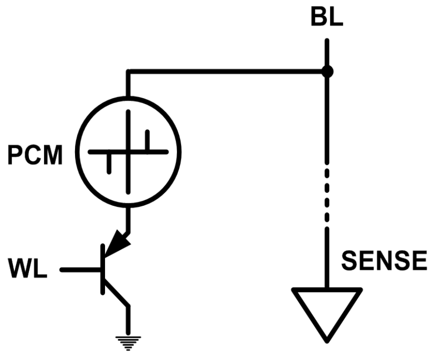
Agenda

- Major Trends Affecting Main Memory
- The DRAM Scaling Problem and Solution Directions
  - Tolerating DRAM: New DRAM Architectures
  - Enabling Emerging Technologies: Hybrid Memory Systems
- The Memory Interference/QoS Problem and Solutions
  - QoS-Aware Memory Systems
- How Can We Do Better?
- Summary

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

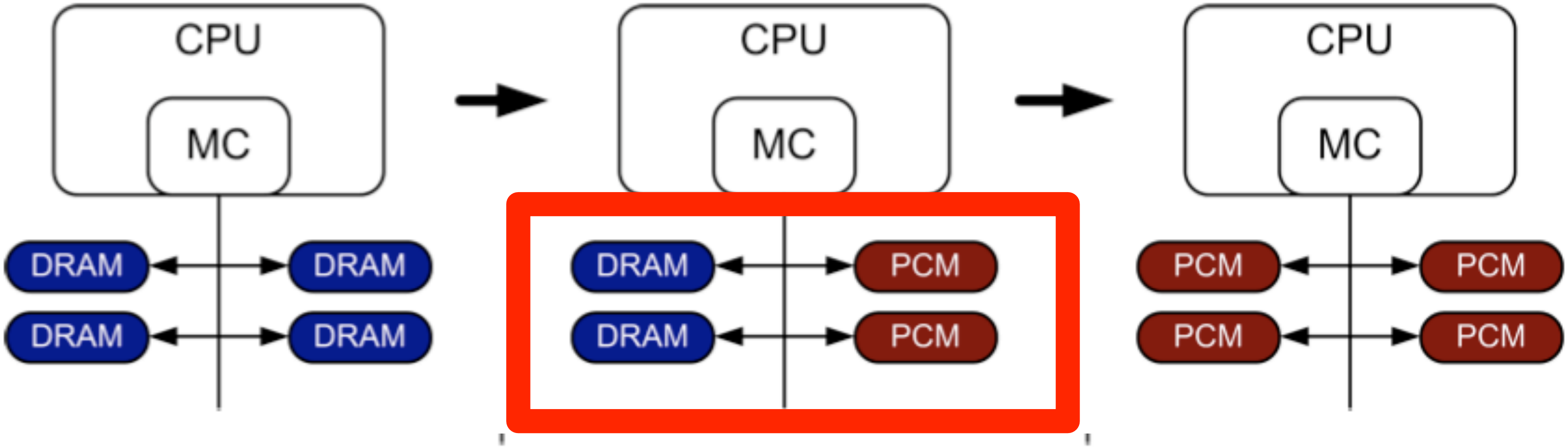
- Pros over DRAM
  - Better technology scaling (capacity and cost)
  - Non volatility
  - Low idle power (no refresh)

- 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)

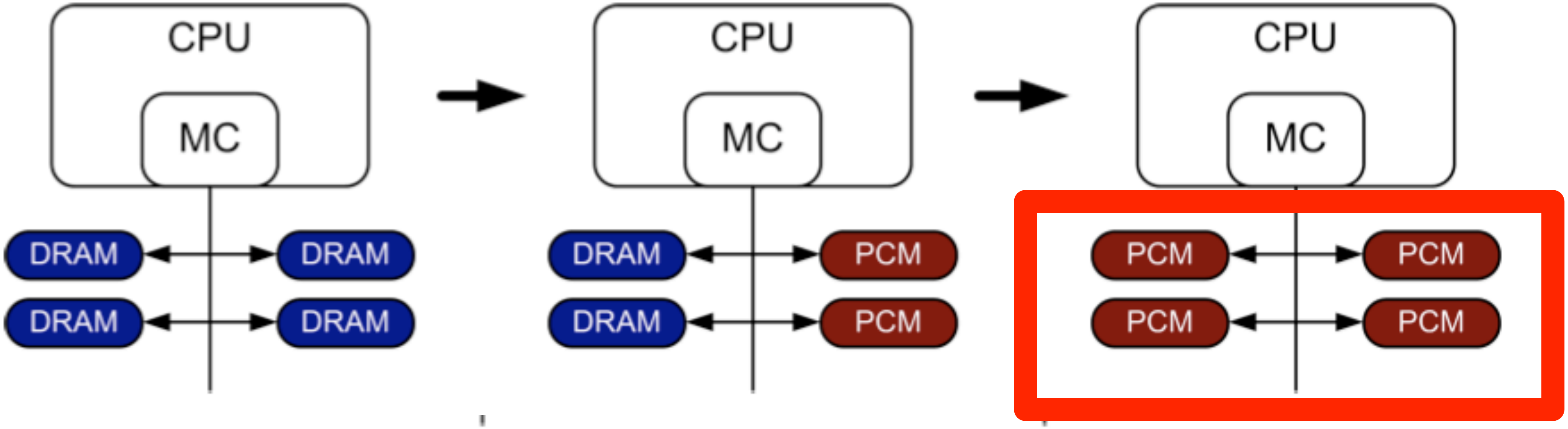
- How should PCM-based (main) memory be organized?



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

PCM-based Main Memory (II)

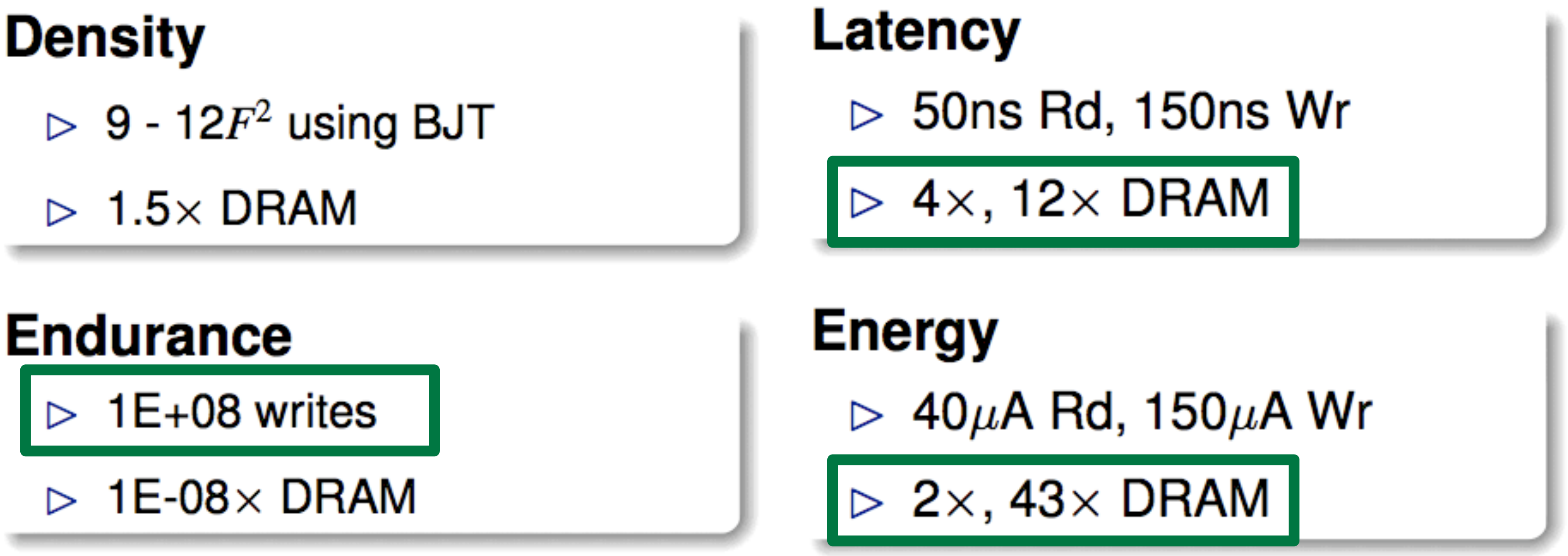
- How should PCM-based (main) memory be organized?



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

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

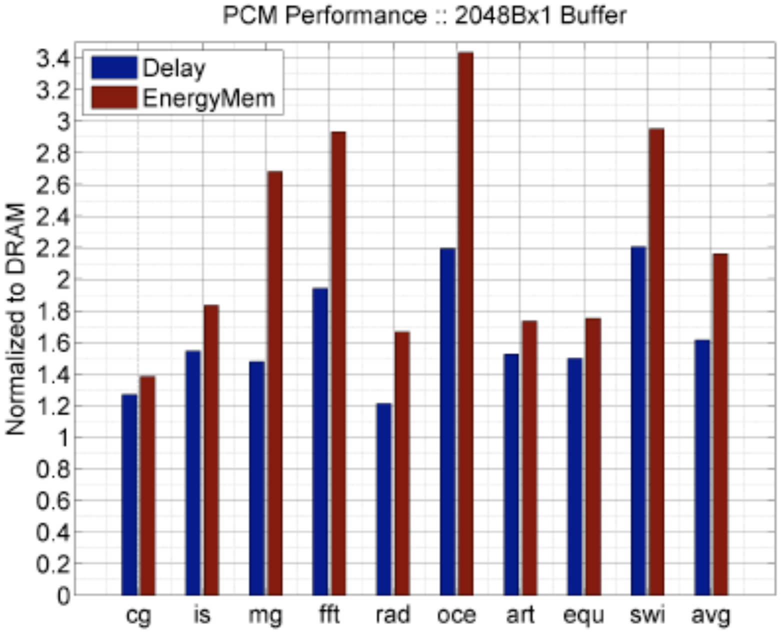
Latency
- ▷ 50ns Rd, 150ns Wr
- ▷ 4×, 12× DRAM

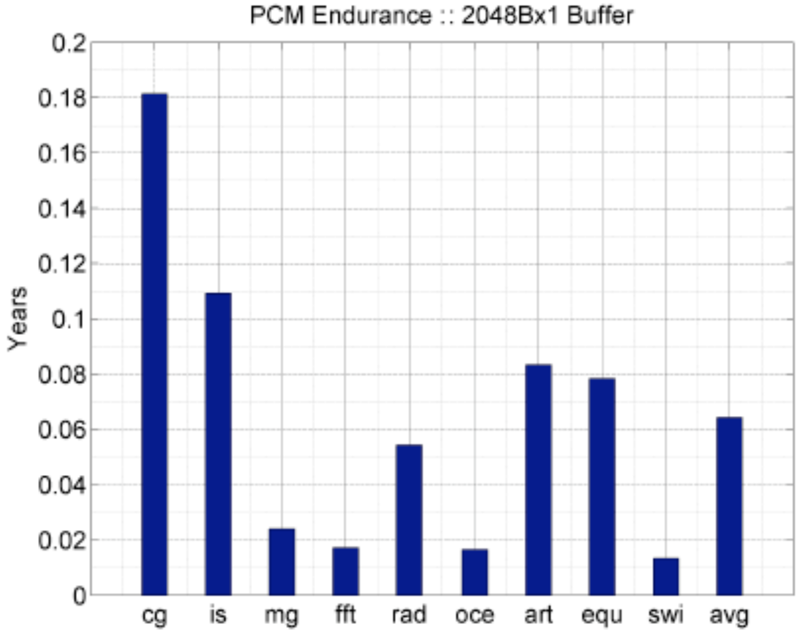
Endurance
- ▷ 1E+08 writes
- ▷ 1E-08× DRAM

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



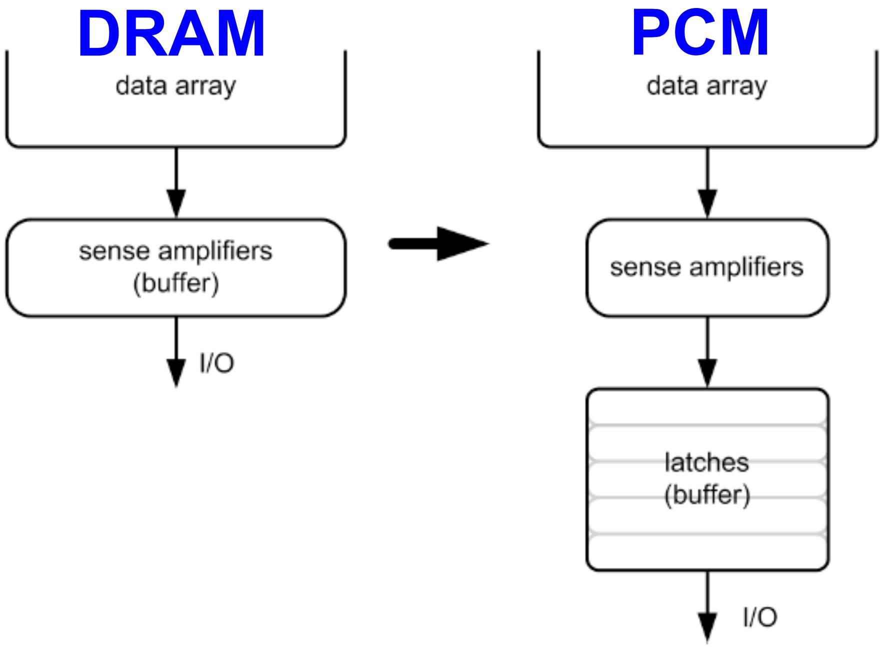


- Lee, Ipek, Mutlu, Burger, "Architecting Phase Change Memory as a Scalable DRAM Alternative," ISCA 2009.

Architecting PCM to Mitigate Shortcomings

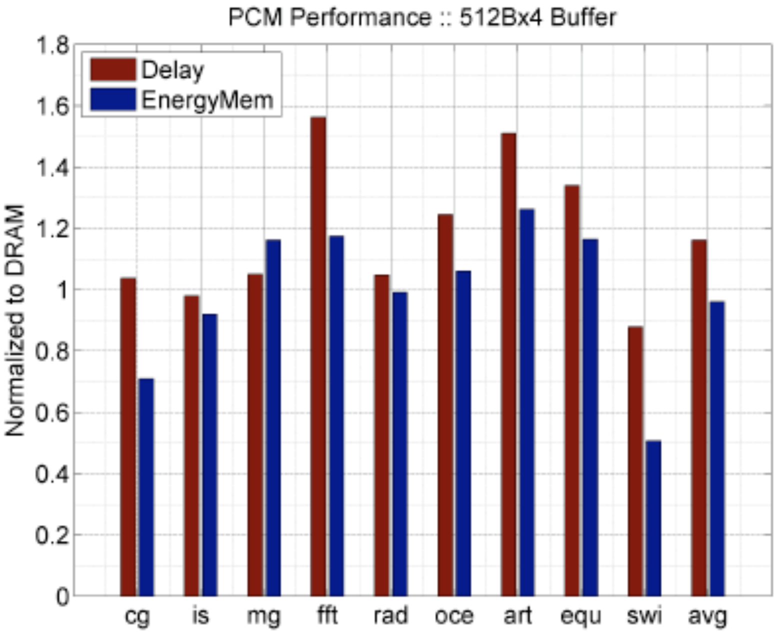
- Idea 1: Use multiple narrow row buffers in each PCM chip
  → Reduces array reads/writes → better endurance, latency, energy

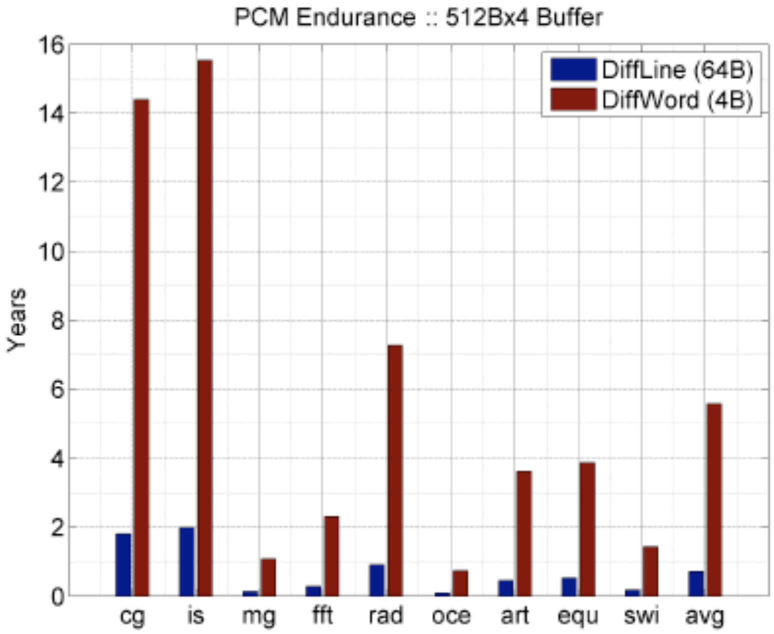
- Idea 2: Write into array at cache block or word granularity
  → Reduces unnecessary wear



Results: Architected PCM as Main Memory

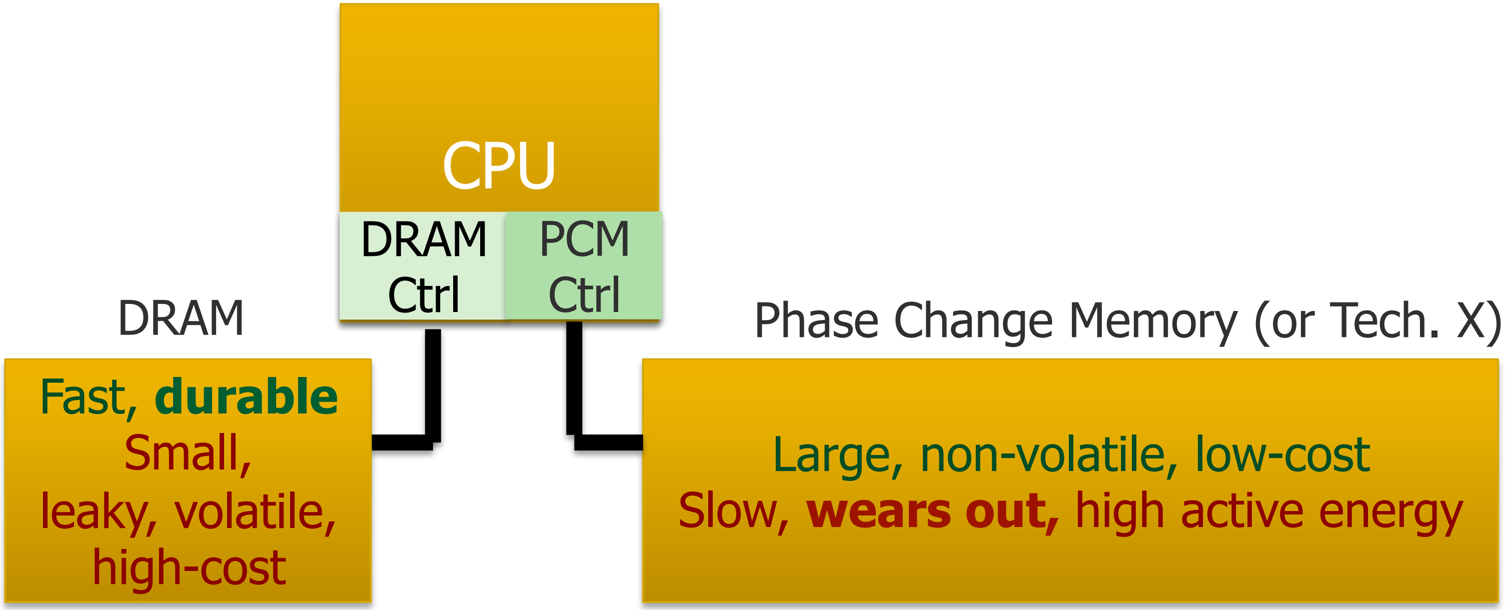
- 1.2x delay, 1.0x energy, 5.6-year average lifetime
- Scaling improves energy, endurance, density





- 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


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

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

Meza+, "Enabling Efficient and Scalable Hybrid Memories," IEEE Comp. Arch. Letters, 2012.

Yoon, Meza et al., "Row Buffer Locality Aware Caching Policies for Hybrid Memories," ICCD 2012 Best Paper Award.

One Option: DRAM as a Cache for PCM

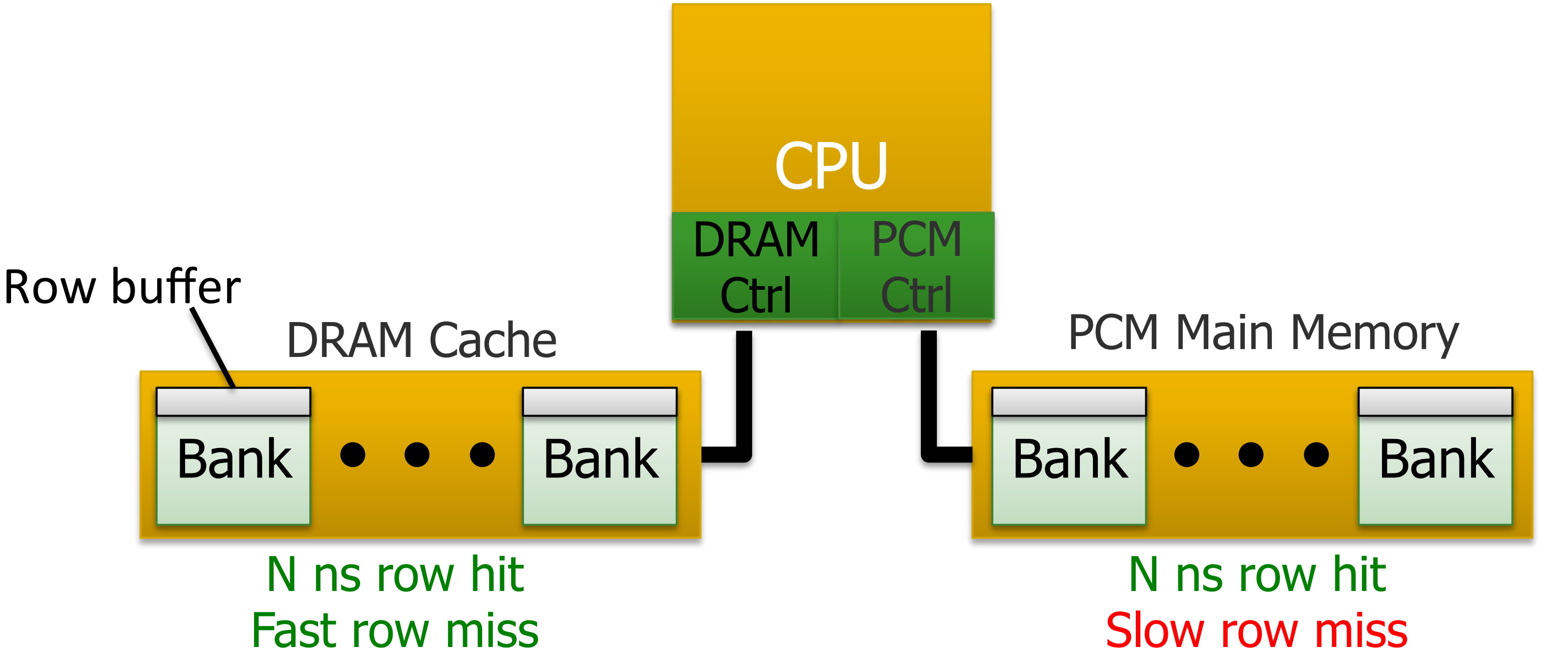
- PCM is main memory; DRAM caches memory rows/blocks
  - Benefits: Reduced latency on DRAM cache hit; write filtering
- Memory controller hardware manages the DRAM cache
  - Benefit: Eliminates system software overhead

- 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 solutions:
  - 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

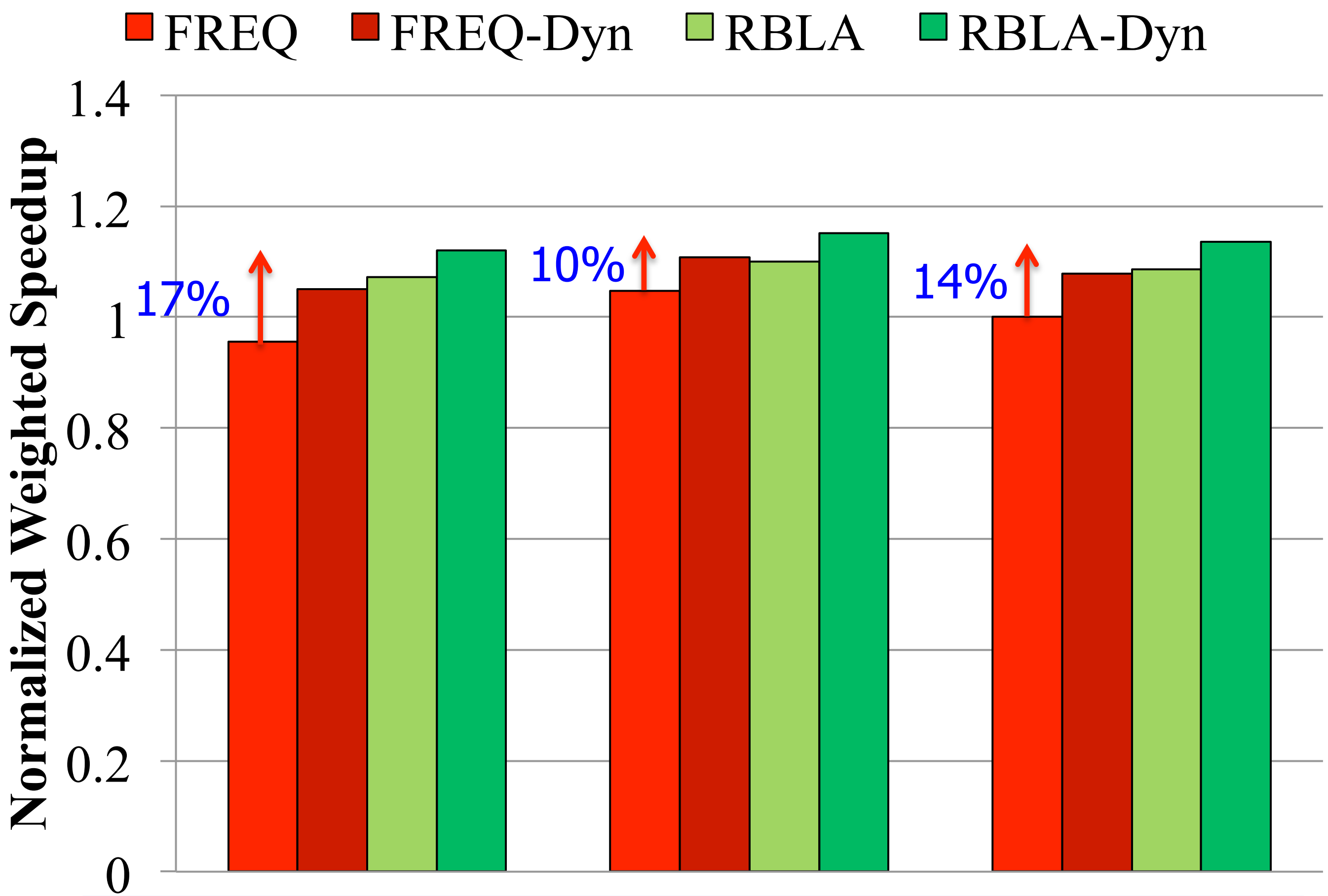
Row-Locality-Aware Data Placement

- 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

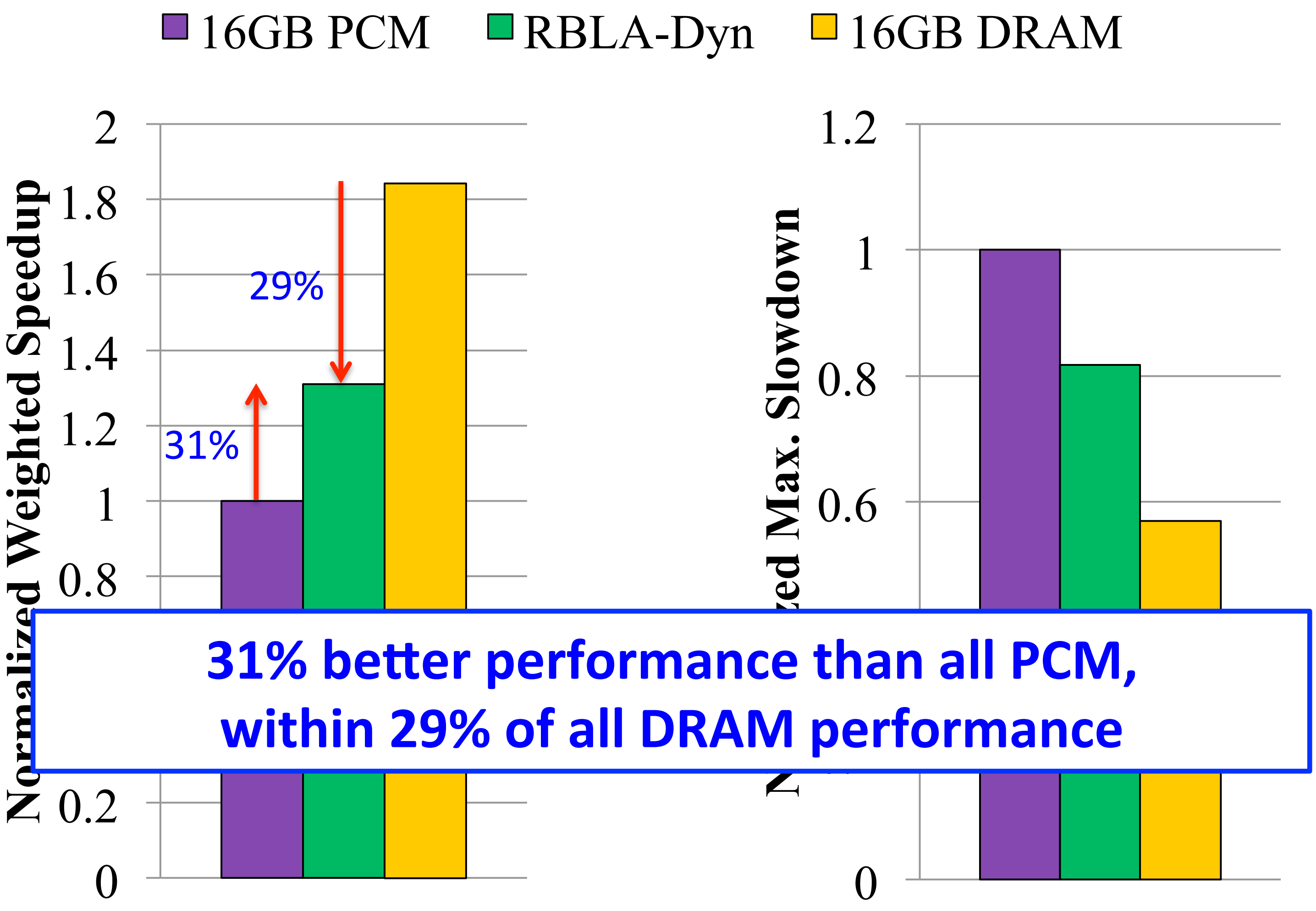
- Yoon et al., "Row Buffer Locality-Aware Data Placement in Hybrid Memories," ICCD 2012.

Row-Locality-Aware Data Placement: Results



Memory energy-efficiency and fairness also improve correspondingly

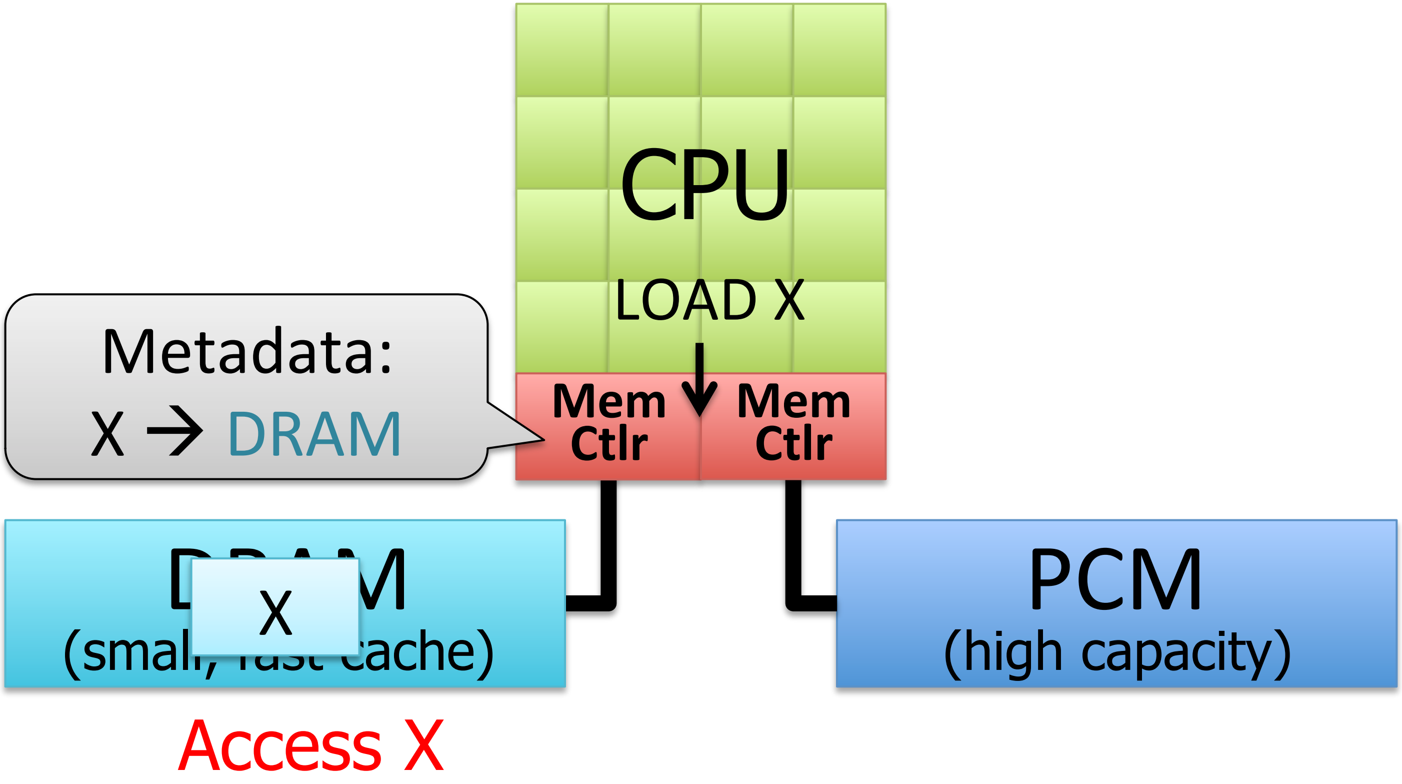
Hybrid vs. All-PCM/DRAM



31% better performance than all PCM, within 29% of all DRAM performance

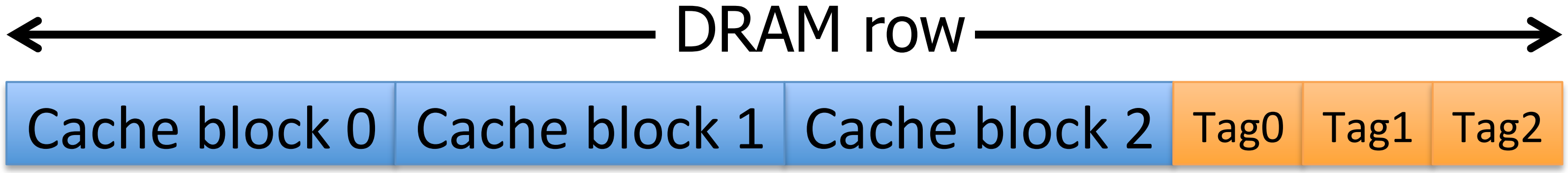
The Problem with Large DRAM Caches

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



Idea 1: Store Tags in Main Memory

- Store tags in the same row as data in DRAM
  - 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 On-Chip SRAM

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

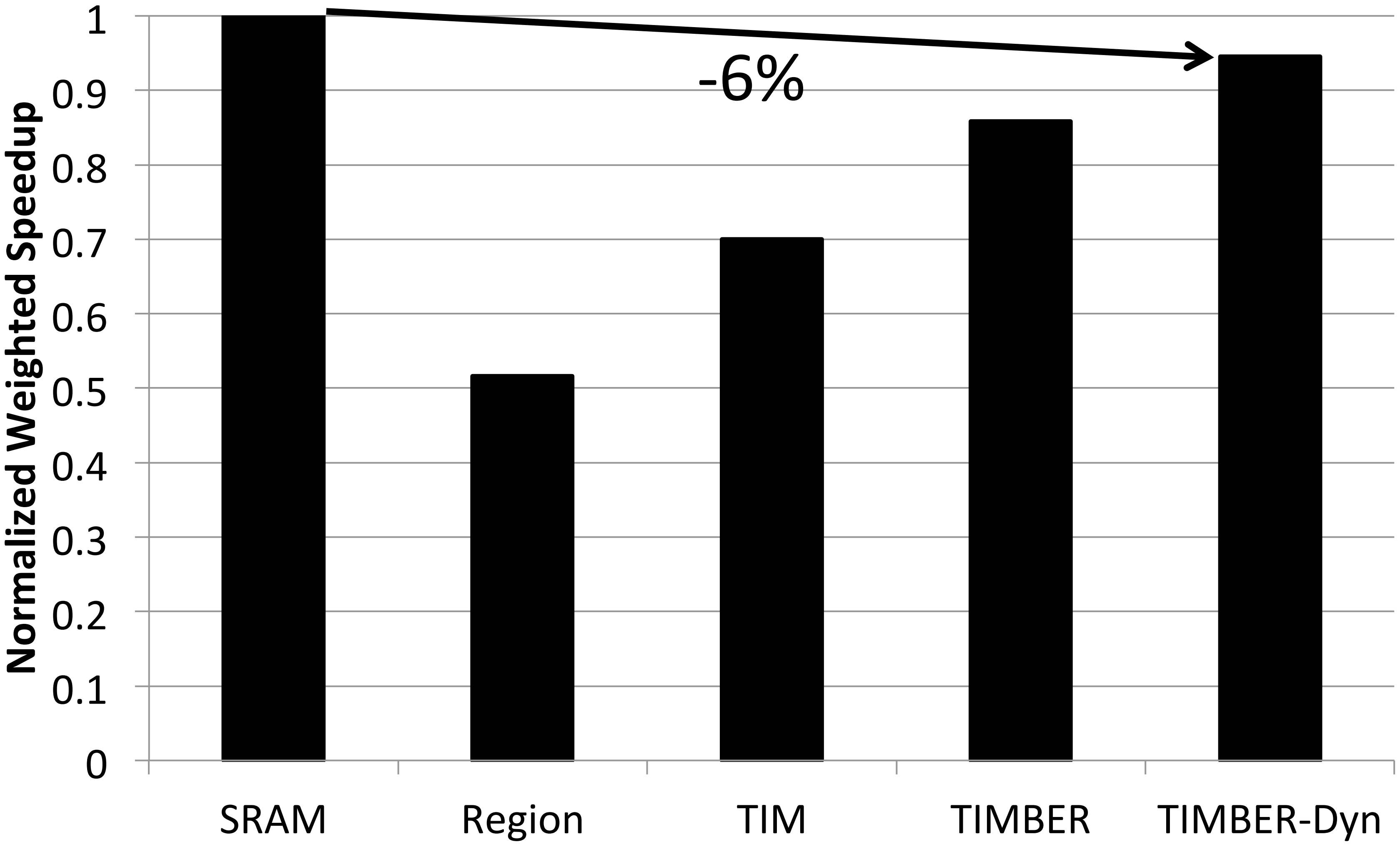
- Idea 2: Cache in on-chip SRAM frequently-accessed metadata
  - Cache only a small amount to keep SRAM size small

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

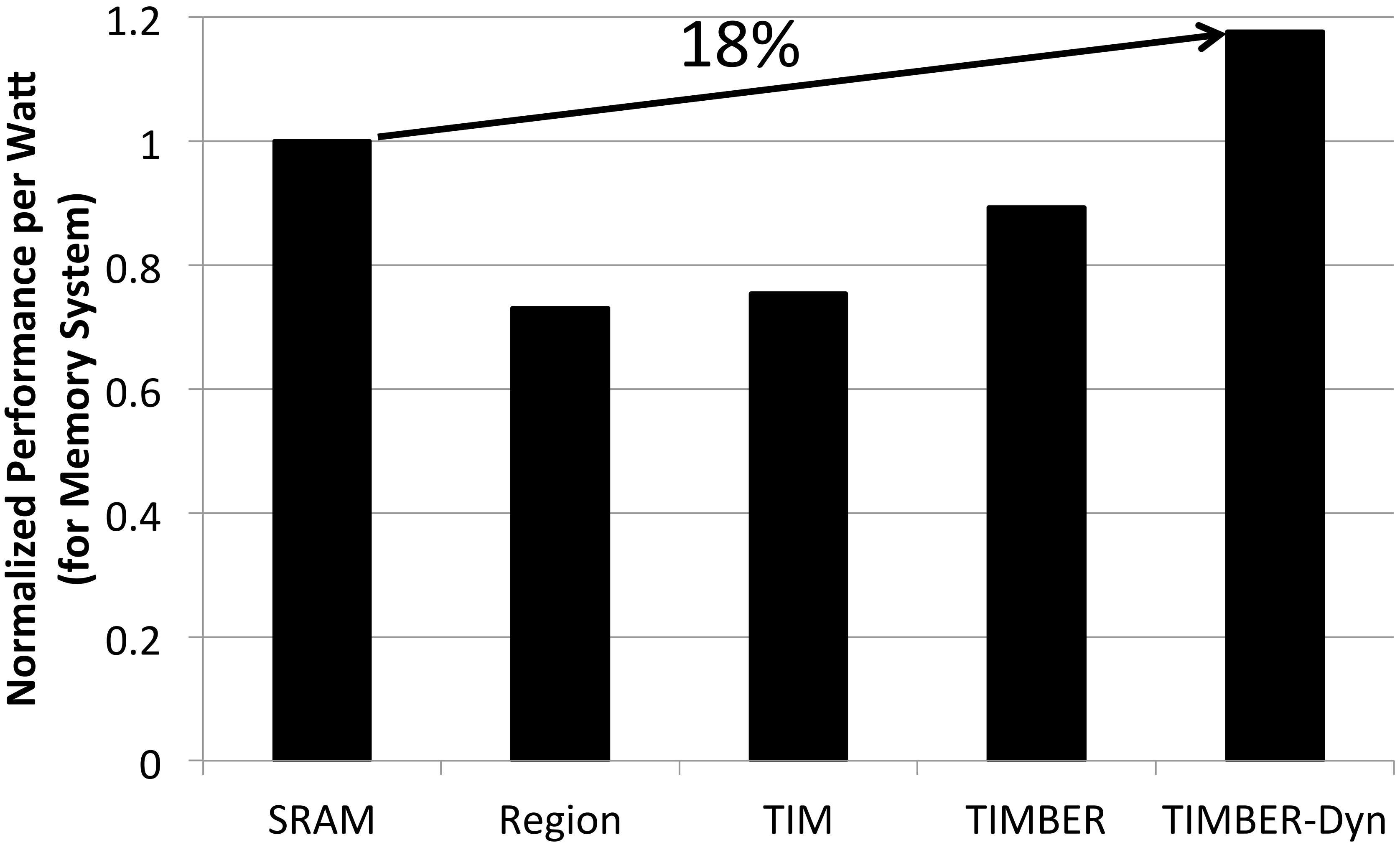
- Idea 3: Simple dynamic caching granularity policy
  - Cost-benefit analysis to determine best DRAM cache block size
  - Group main memory into sets of rows
  - Some row sets follow a fixed caching granularity
  - The rest of main memory follows the best granularity
    - Cost–benefit analysis: access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

TIMBER 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.

More on Hybrid Memories

- Justin Meza, Jichuan Chang, HanBin Yoon, Onur Mutlu, and Parthasarathy Ranganathan,
  "Enabling Efficient and Scalable Hybrid Memories Using Fine-Granularity DRAM Cache Management"
  IEEE Computer Architecture Letters (***CAL***), February 2012.

- HanBin Yoon, Justin Meza, Rachata Ausavarungnirun, Rachael Harding, and Onur Mutlu,
  "Row Buffer Locality Aware Caching Policies for Hybrid Memories"
  Proceedings of the 30th IEEE International Conference on Computer Design (***ICCD***), Montreal, Quebec, Canada, September 2012. Slides (pptx) (pdf)

- Benjamin C. Lee, Engin Ipek, Onur Mutlu, and Doug Burger,
  "Architecting Phase Change Memory as a Scalable DRAM Alternative"
  Proceedings of the 36th International Symposium on Computer Architecture (***ISCA***), pages 2-13, Austin, TX, June 2009. Slides (pdf)

Further: Overview Papers on Two Topics

- Merging of Memory and Storage
  - Justin Meza, Yixin Luo, Samira Khan, Jishen Zhao, Yuan Xie, and Onur Mutlu,
    "A Case for Efficient Hardware-Software Cooperative Management of Storage and Memory"
    *Proceedings of the 5th Workshop on Energy-Efficient Design (**WEED**)*, Tel-Aviv, Israel, June 2013. Slides (pptx) Slides (pdf)

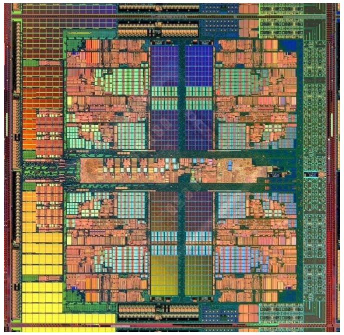
- Flash Memory Scaling
  - Yu Cai, Gulay Yalcin, Onur Mutlu, Erich F. Haratsch, Adrian Cristal, Osman Unsal, and Ken Mai,
    "Error Analysis and Retention-Aware Error Management for NAND Flash Memory"
    *Intel Technology Journal (**ITJ**) Special Issue on Memory Resiliency*, Vol. 17, No. 1, May 2013.

Agenda

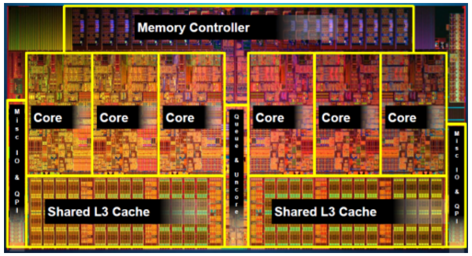
- Major Trends Affecting Main Memory
- The DRAM Scaling Problem and Solution Directions
  - Tolerating DRAM: New DRAM Architectures
  - Enabling Emerging Technologies: Hybrid Memory Systems
- The Memory Interference/QoS Problem and Solutions
  - QoS-Aware Memory Systems
- How Can We Do Better?
- Summary

Trend: Many Cores on Chip

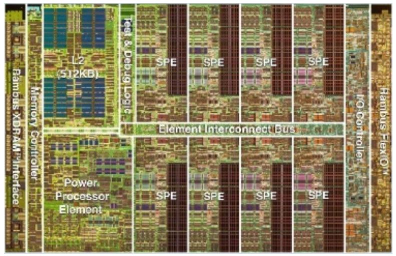
- Simpler and lower power than a single large core
- Large scale parallelism on chip



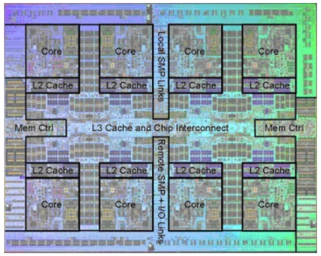
AMD Barcelona
4 cores



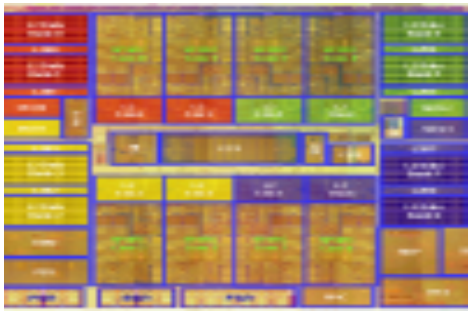
Intel Core i7
8 cores



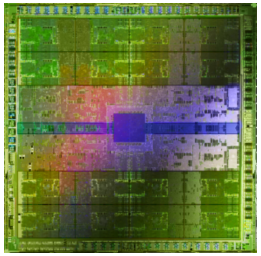
IBM Cell BE
8+1 cores



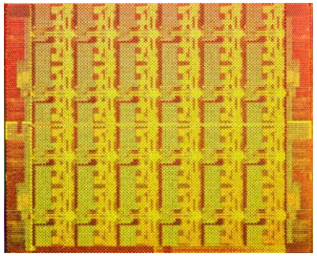
IBM POWER7
8 cores



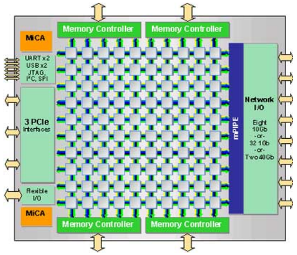
Sun Niagara II
8 cores



Nvidia Fermi
448 “cores”



Intel SCC
48 cores, networked



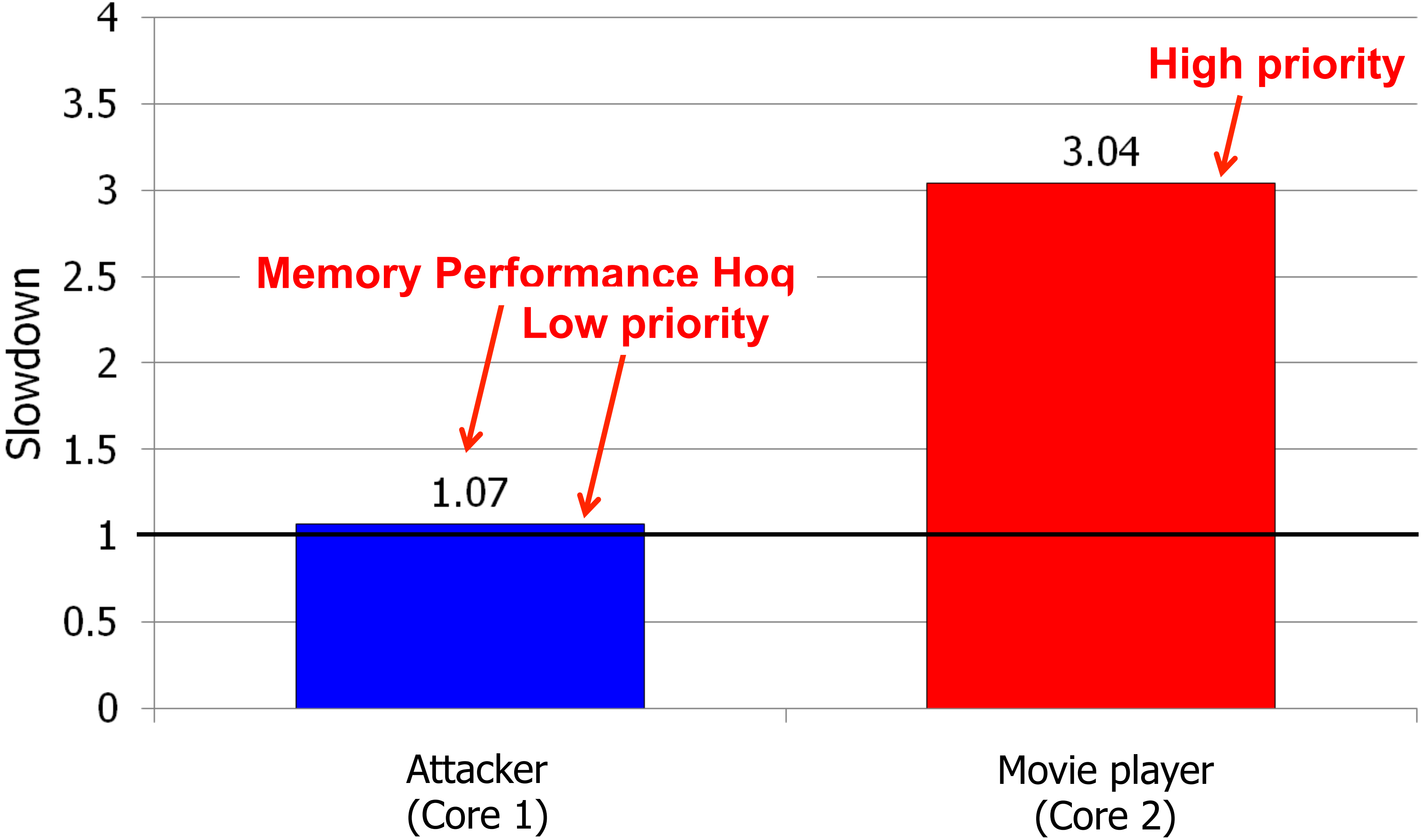
Tilera TILE Gx
100 cores, networked

Many Cores on Chip

- What we want:
  - N times the system performance with N times the cores

- What do we get today?

(Un)expected Slowdowns



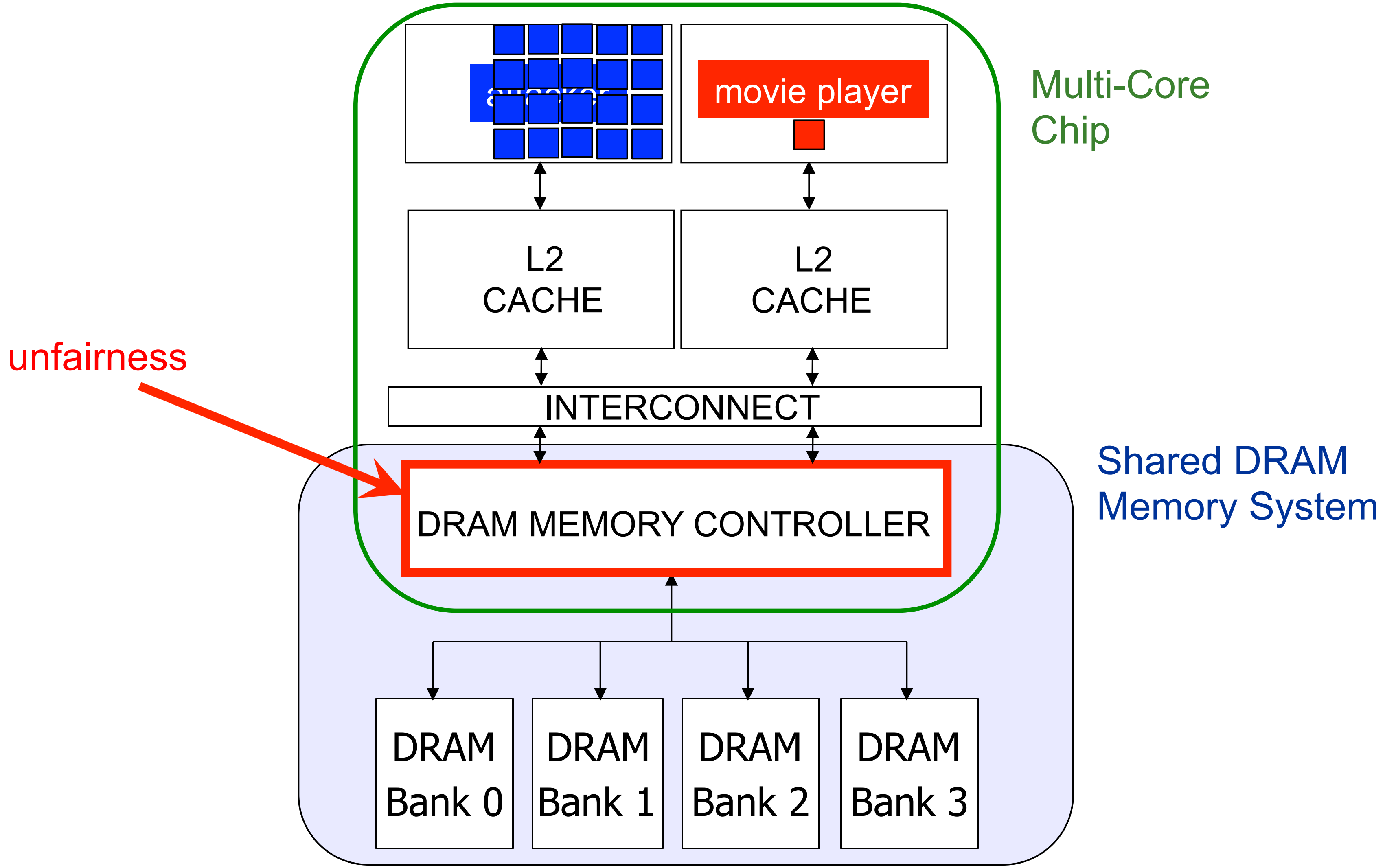
High priority

Memory Performance Hoq

Low priority

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

Why? Uncontrolled Memory Interference



attacker

movie player

L2 CACHE

L2 CACHE

INTERCONNECT

DRAM MEMORY CONTROLLER

DRAM Bank 0

DRAM Bank 1

DRAM Bank 2

DRAM Bank 3

unfairness

Multi-Core Chip

Shared DRAM Memory System

A Memory Performance Hog

```
// initialize large arrays A, B

for (j=0; j<N; j++) {
    index = j*linesize;  streaming
    A[index] = B[index];
    …
}
```

```
// initialize large arrays A, B

for (j=0; j<N; j++) {
    index = rand();  random
    A[index] = B[index];
    …
}
```

STREAM

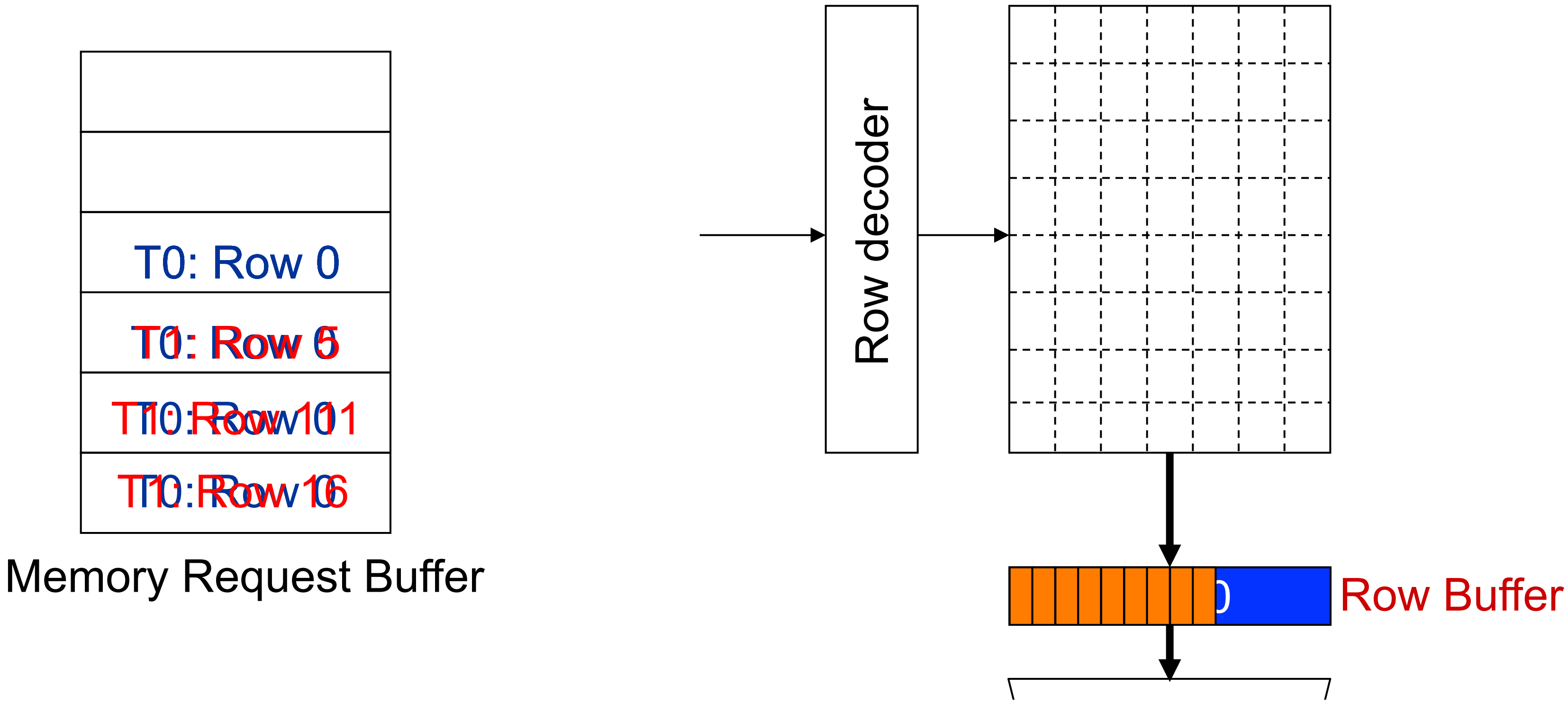
- Sequential memory access
- Very high row buffer locality (96% hit rate)
- Memory intensive

RANDOM

- Random memory access
- Very low row buffer locality (3% hit rate)
- Similarly memory intensive

Moscibroda and Mutlu, "Memory Performance Attacks," USENIX Security 2007.

What Does the Memory Hog Do?

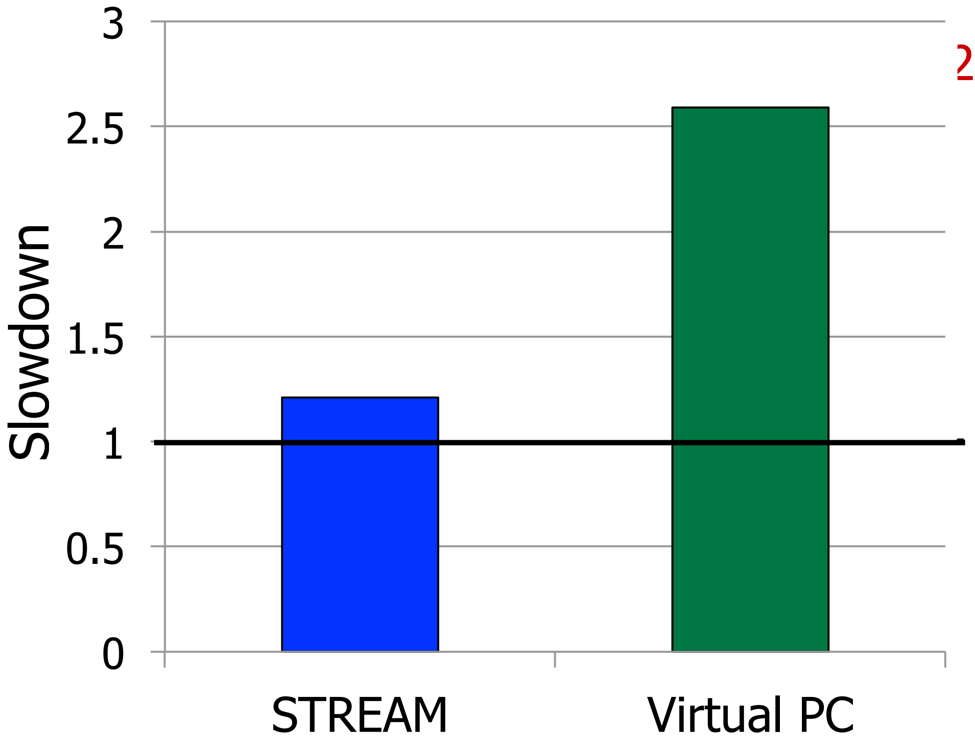


Row size: 8KB, cache block size: 64B

128 (8KB/64B) requests of T0 serviced before T1

Moscibroda and Mutlu, "Memory Performance Attacks," USENIX Security 2007.

Effect of the Memory Performance Hog

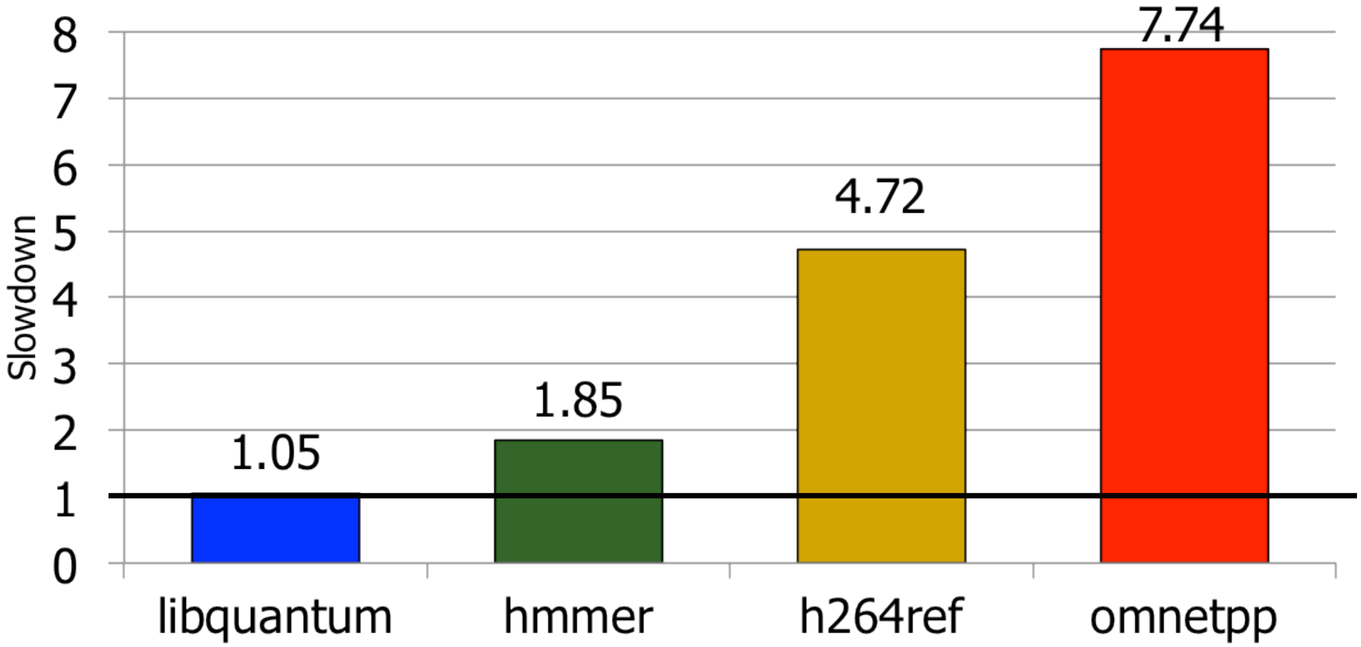


2X slowdown

Results on Intel Pentium D running Windows XP
(Similar results for Intel Core Duo and AMD Turion, and on Fedora Linux)

Moscibroda and Mutlu, “Memory Performance Attacks,” USENIX Security 2007.

Greater Problem with More Cores



- Vulnerable to denial of service (DoS) [Usenix Security'07]
- Unable to enforce priorities or SLAs [MICRO'07,'10,'11, ISCA'08'11'12, ASPLOS'10]
- Low system performance [IEEE Micro Top Picks '09,'11a,'11b,'12]

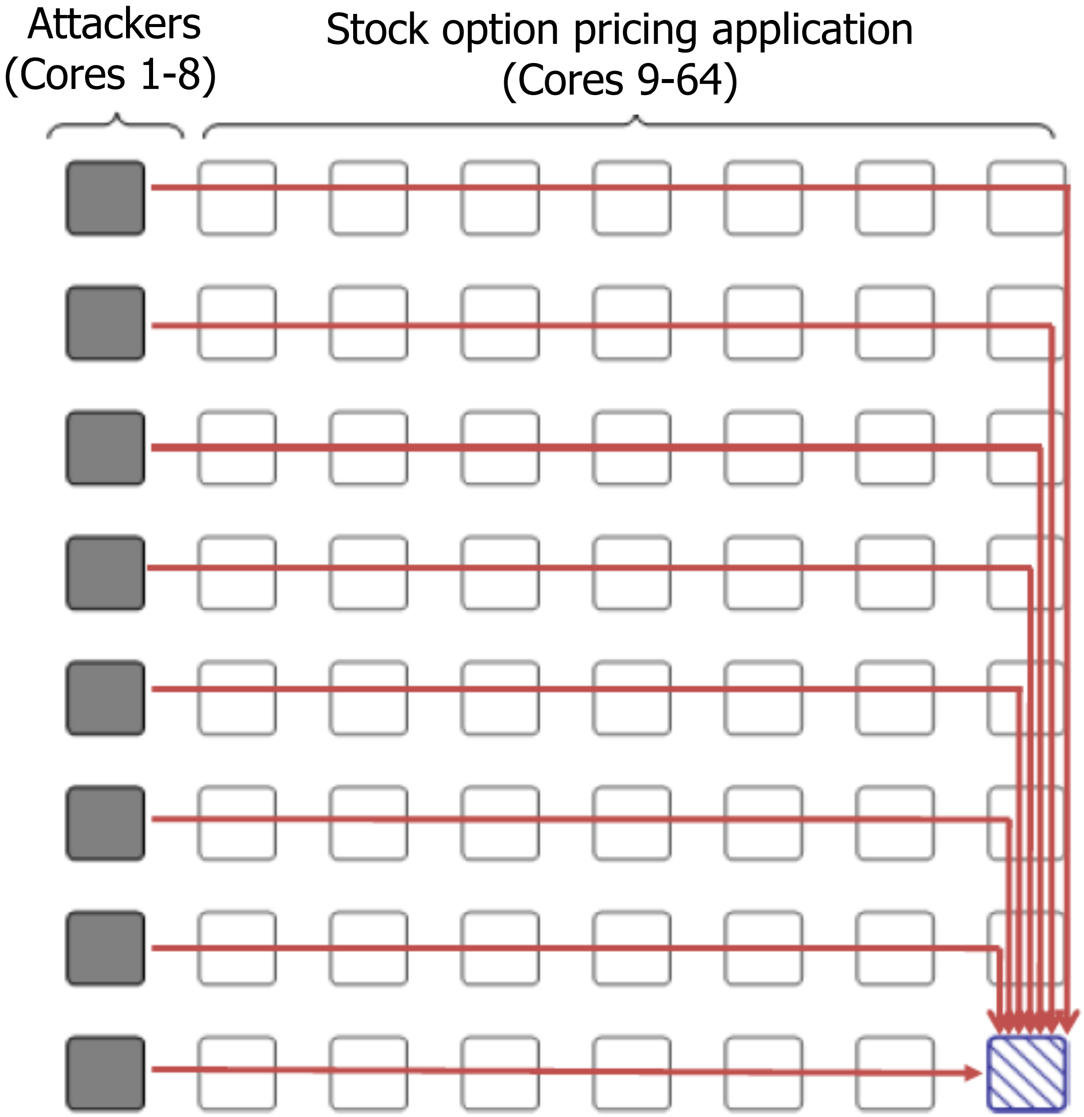
Uncontrollable, unpredictable system

Distributed DoS in Networked Multi-Core Systems

Cores connected via packet-switched routers on chip

~5000X slowdown

Grot, Hestness, Keckler, Mutlu, "Preemptive virtual clock: A Flexible, Efficient, and Cost-effective QOS Scheme for Networks-on-Chip," MICRO 2009.



Solution: QoS-Aware, Predictable Memory

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

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

- Solution: Hardware/software cooperative memory QoS
  - Hardware designed to provide a configurable fairness substrate
    - Application-aware memory scheduling, partitioning, throttling
  - Software designed to configure the resources to satisfy different QoS goals

  - E.g., fair, programmable memory controllers and on-chip networks provide QoS and predictable performance

    [2007-2012, Top Picks'09,'11a,'11b,'12]

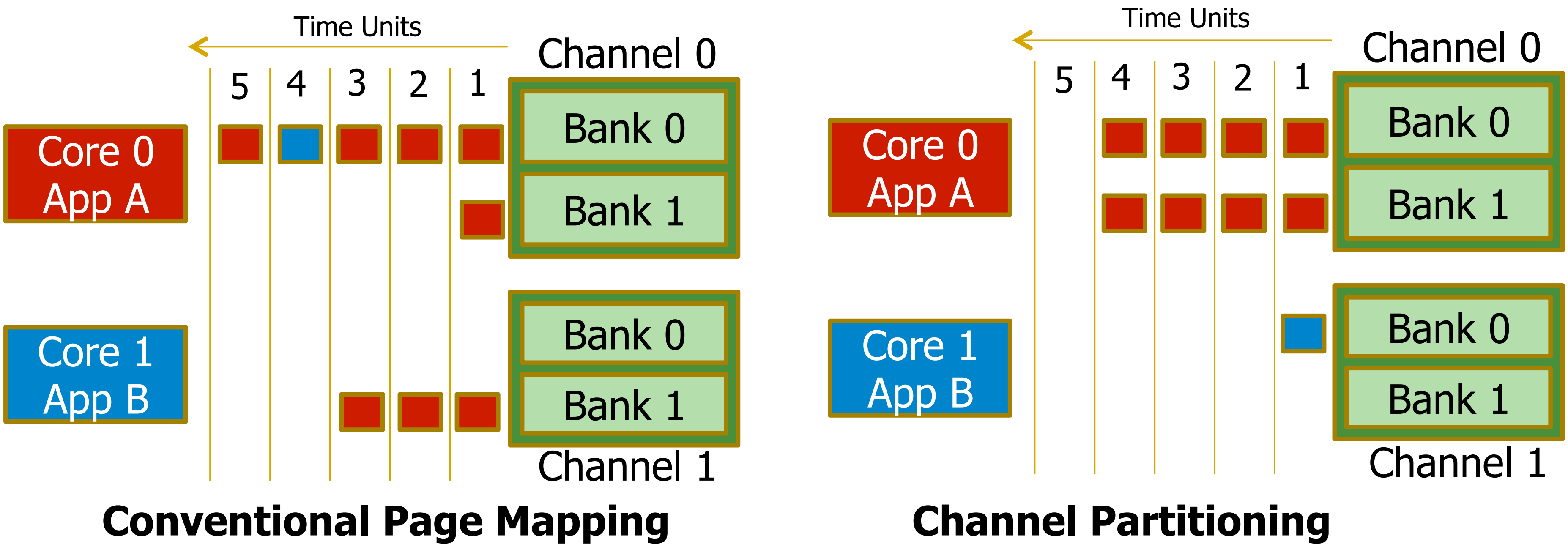
Designing QoS-Aware Memory Systems: Approaches

- Smart resources: Design each shared resource to have a configurable interference control/reduction mechanism
  - QoS-aware memory controllers [Mutlu+ MICRO'07] [Moscibroda+, Usenix Security'07] [Mutlu+ ISCA'08, Top Picks'09] [Kim+ HPCA'10] [Kim+ MICRO'10, Top Picks'11] [Ebrahimi+ ISCA'11, MICRO'11] [Ausavarungnirun+, ISCA'12][Subramanian+, HPCA'13]
  - QoS-aware interconnects [Das+ MICRO'09, ISCA'10, Top Picks '11] [Grot+ MICRO'09, ISCA'11, Top Picks '12]
  - QoS-aware caches

- Dumb resources: Keep each resource free-for-all, but reduce/control interference by injection control or data mapping
  - Source throttling to control access to memory system [Ebrahimi+ ASPLOS'10, ISCA'11, TOCS'12] [Ebrahimi+ MICRO'09] [Nychis+ HotNets'10] [Nychis+ SIGCOMM'12]
  - QoS-aware data mapping to memory controllers [Muralidhara+ MICRO'11]
  - QoS-aware thread scheduling to cores [Das+ HPCA'13]

A Mechanism to Reduce Memory Interference

- Memory Channel Partitioning
  - Idea: System software maps badly-interfering applications' pages to different channels [Muralidhara+, MICRO'11]



- Separate data of low/high intensity and low/high row-locality applications
- Especially effective in reducing interference of threads with "medium" and "heavy" memory intensity
  - 11% higher performance over existing systems (200 workloads)

More on Memory Channel Partitioning

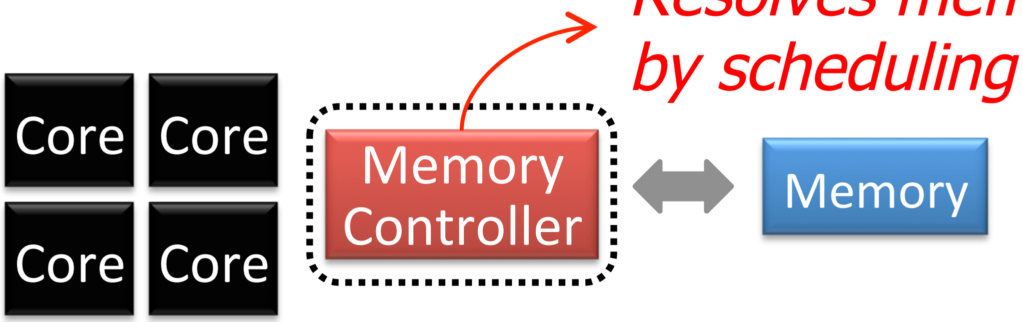
- Sai Prashanth Muralidhara, Lavanya Subramanian, Onur Mutlu, Mahmut Kandemir, and Thomas Moscibroda,
"Reducing Memory Interference in Multicore Systems via Application-Aware Memory Channel Partitioning"
*Proceedings of the 44th International Symposium on Microarchitecture (**MICRO**)*, Porto Alegre, Brazil, December 2011. Slides (pptx)

Designing QoS-Aware Memory Systems: Approaches

- Smart resources: Design each shared resource to have a configurable interference control/reduction mechanism
  - QoS-aware memory controllers [Mutlu+ MICRO'07] [Moscibroda+, Usenix Security'07] [Mutlu+ ISCA'08, Top Picks'09] [Kim+ HPCA'10] [Kim+ MICRO'10, Top Picks'11] [Ebrahimi+ ISCA'11, MICRO'11] [Ausavarungnirun+, ISCA'12][Subramanian+, HPCA'13]
  - QoS-aware interconnects [Das+ MICRO'09, ISCA'10, Top Picks '11] [Grot+ MICRO'09, ISCA'11, Top Picks '12]
  - QoS-aware caches

- Dumb resources: Keep each resource free-for-all, but reduce/control interference by injection control or data mapping
  - Source throttling to control access to memory system [Ebrahimi+ ASPLOS'10, ISCA'11, TOCS'12] [Ebrahimi+ MICRO'09] [Nychis+ HotNets'10] [Nychis+ SIGCOMM'12]
  - QoS-aware data mapping to memory controllers [Muralidhara+ MICRO'11]
  - QoS-aware thread scheduling to cores [Das+ HPCA'13]

QoS-Aware Memory Scheduling



Resolves memory contention by scheduling requests

- How to schedule requests to provide
  - High system performance
  - High fairness to applications
  - Configurability to system software

- Memory controller needs to be aware of threads

QoS-Aware Memory Scheduling: Evolution

- Stall-time fair memory scheduling [Mutlu+ MICRO'07]
  - Idea: Estimate and balance thread slowdowns
  - Takeaway: Proportional thread progress improves performance, especially when threads are "heavy" (memory intensive)
- Parallelism-aware batch scheduling [Mutlu+ ISCA'08, Top Picks'09]
  - Idea: Rank threads and service in rank order (to preserve bank parallelism); batch requests to prevent starvation
  - Takeaway: Preserving within-thread bank-parallelism improves performance; request batching improves fairness
- ATLAS memory scheduler [Kim+ HPCA'10]
  - Idea: Prioritize threads that have attained the least service from the memory scheduler
  - Takeaway: Prioritizing "light" threads improves performance

Throughput vs. Fairness

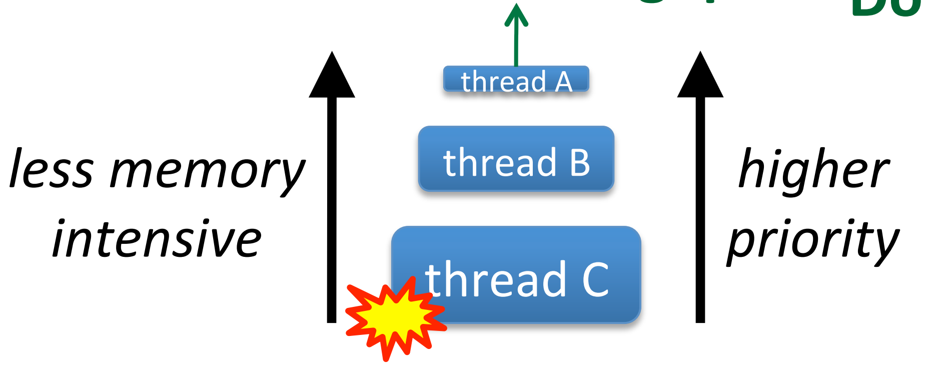
***Throughput biased** approach*

Prioritize less memory-intensive threads

***Fairness biased** approach*

Take turns accessing memory

Good for throughput



less memory intensive

thread A

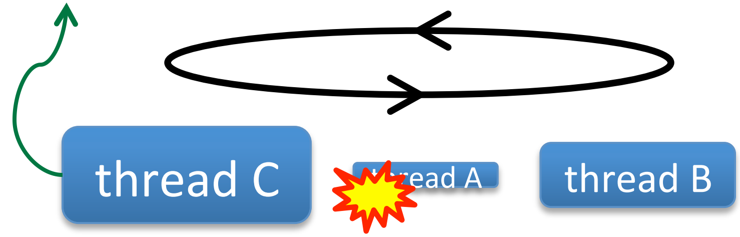
thread B

thread C

higher priority

starvation → unfairness

Does not starve



thread C

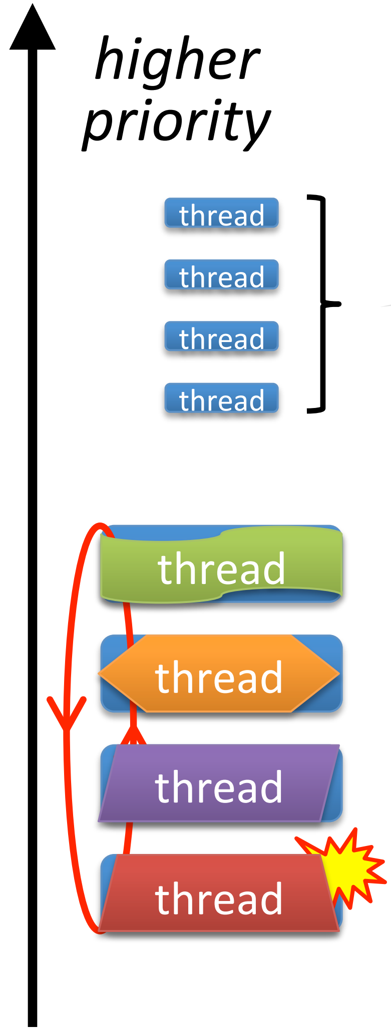
thread A

thread B

not prioritized → reduced throughput

Single policy for all threads is insufficient

Achieving the Best of Both Worlds



For Throughput

Prioritize memory-non-intensive threads

For Fairness

Unfairness caused by memory-intensive being prioritized over each other

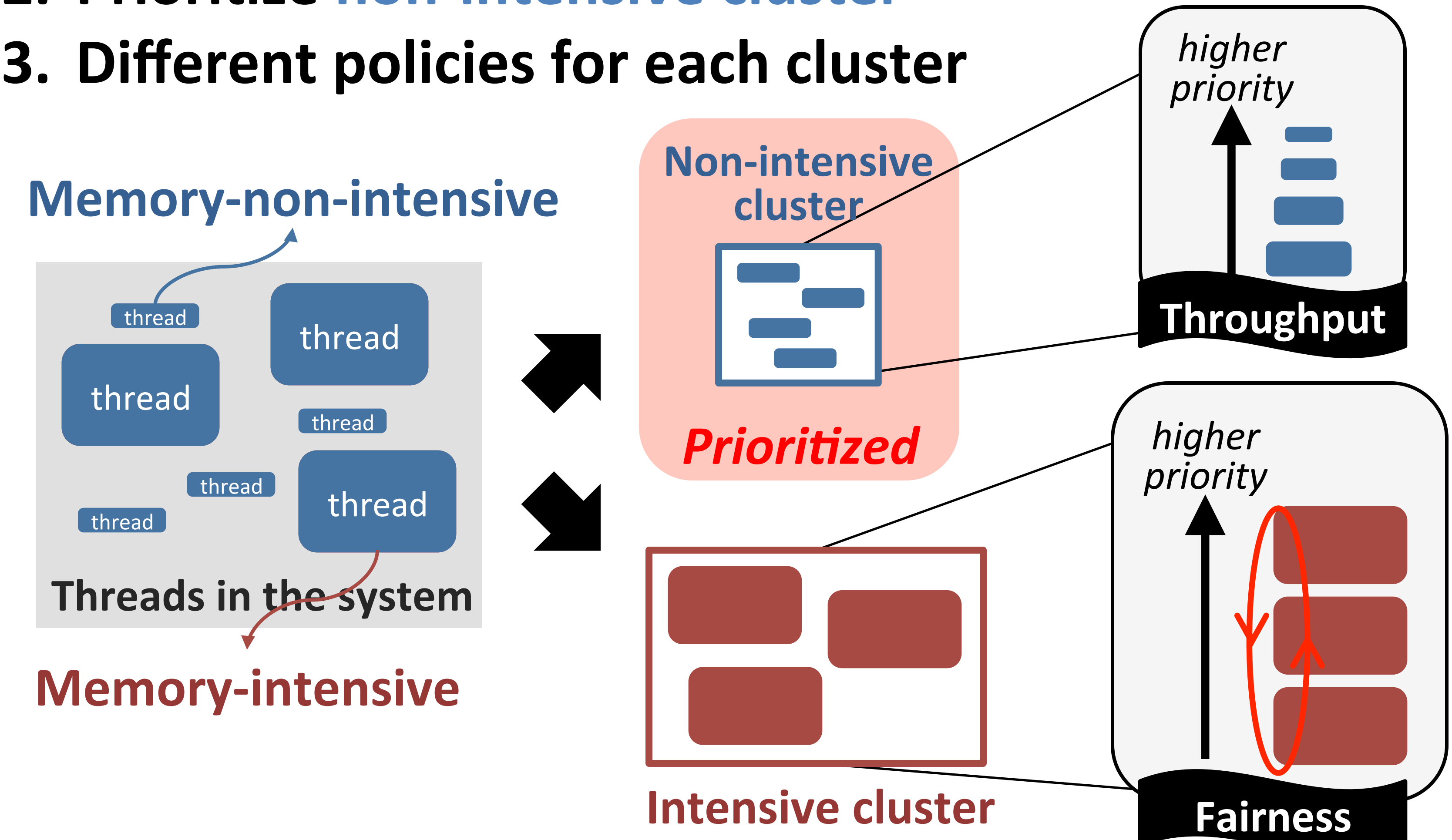
- Shuffle thread ranking

Memory-intensive threads have different vulnerability to interference

- Shuffle asymmetrically

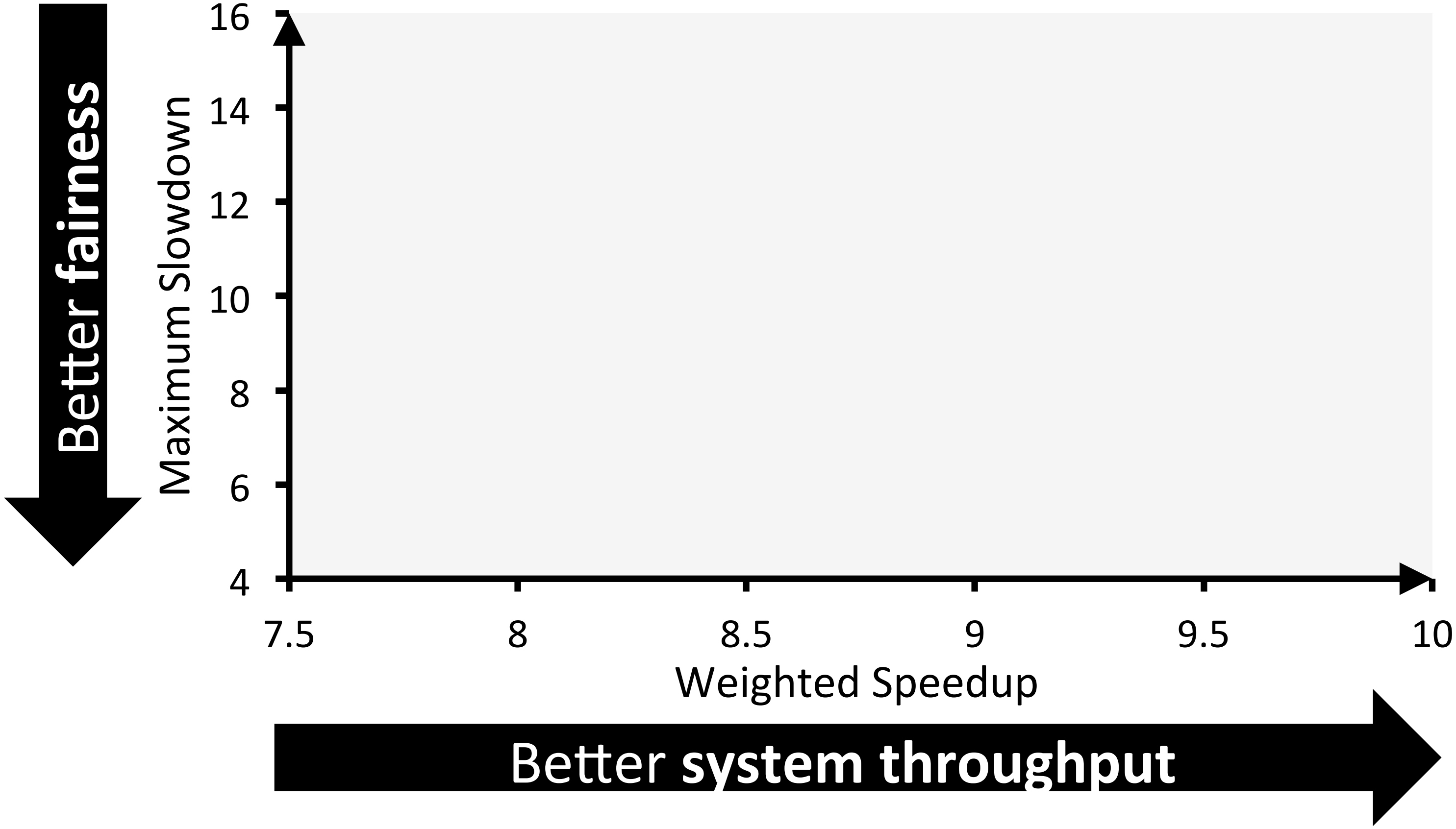
Thread Cluster Memory Scheduling [Kim+ MICRO'10]

1. Group threads into two *clusters*
2. Prioritize non-intensive cluster
3. Different policies for each cluster



TCM: Throughput and Fairness

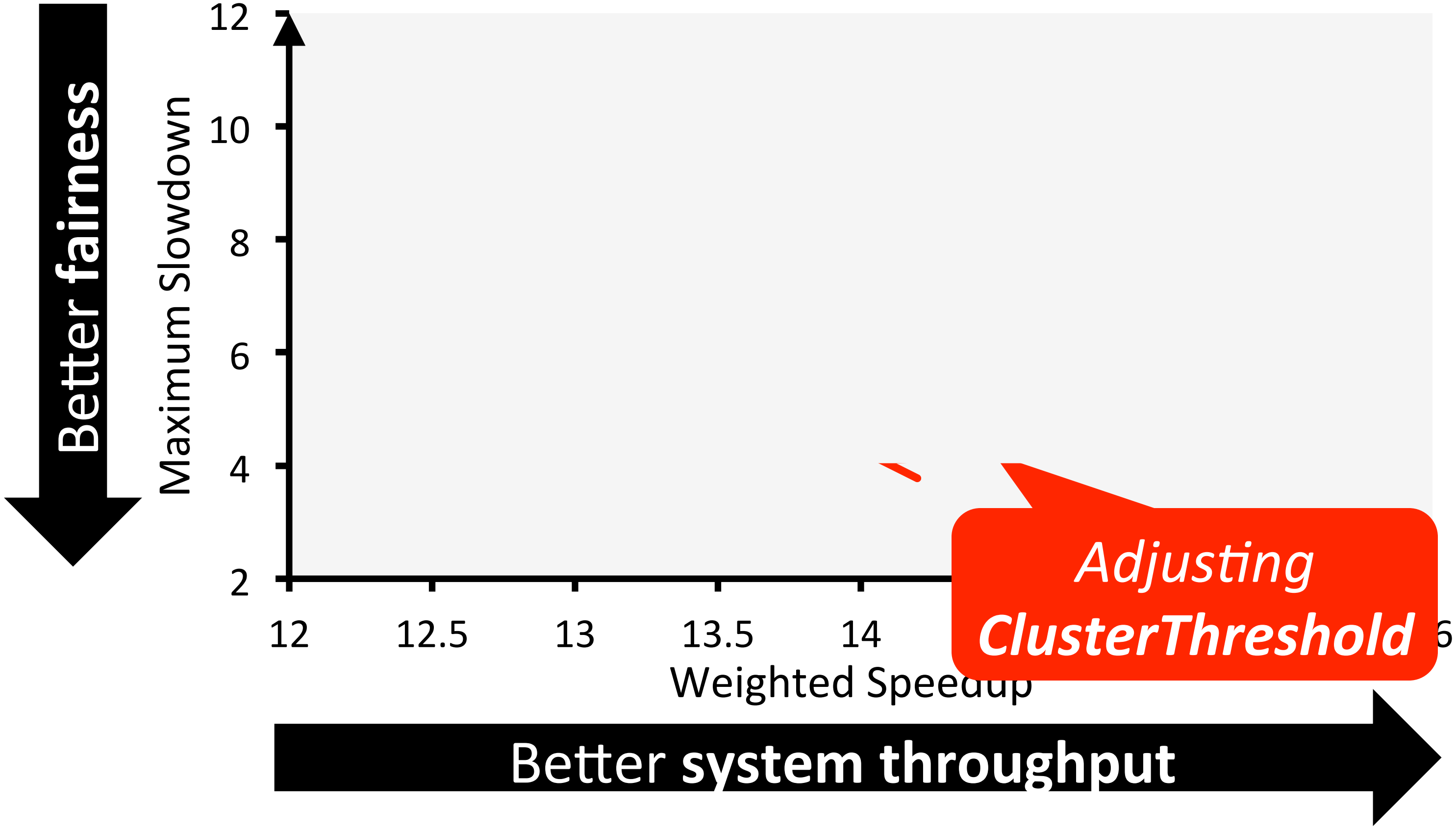
24 cores, 4 memory controllers, 96 workloads



TCM, a heterogeneous scheduling policy, provides best fairness and system throughput

TCM: Fairness-Throughput Tradeoff

When configuration parameter is varied…



Adjusting ***ClusterThreshold***

TCM allows robust fairness-throughput tradeoff

Memory QoS in a Parallel Application

- Threads in a multithreaded application are inter-dependent
- Some threads can be on the critical path of execution due to synchronization; some threads are not
- How do we schedule requests of inter-dependent threads to maximize multithreaded application performance?

- Idea: Estimate limiter threads likely to be on the critical path and prioritize their requests; shuffle priorities of non-limiter threads to reduce memory interference among them [Ebrahimi+, MICRO'11]

- Hardware/software cooperative limiter thread estimation:
  - Thread executing the most contended critical section
  - Thread that is falling behind the most in a *parallel for* loop

Summary: Memory QoS Approaches and Techniques

- Approaches: Smart vs. dumb resources
  - Smart resources: QoS-aware memory scheduling
  - Dumb resources: Source throttling; channel partitioning
  - Both approaches are effective in reducing interference
  - No single best approach for all workloads

- Techniques: Request/thread scheduling, source throttling, memory partitioning
  - All approaches are effective in reducing interference
  - Can be applied at different levels: hardware vs. software
  - No single best technique for all workloads

- Combined approaches and techniques are the most powerful
  - Integrated Memory Channel Partitioning and Scheduling [MICRO'11]

Summary: Memory Interference and QoS

- QoS-unaware memory → uncontrollable and unpredictable system

- Providing QoS awareness improves performance, predictability, fairness, and utilization of the memory system

- Designed new techniques to:
  - Minimize memory interference
  - Provide predictable performance

- Many new research ideas needed for integrated techniques and closing the interaction with software

Readings on Memory QoS (I)

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

Readings on Memory QoS (II)

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

Agenda

- Major Trends Affecting Main Memory
- The DRAM Scaling Problem and Solution Directions
  - Tolerating DRAM: New DRAM Architectures
  - Enabling Emerging Technologies: Hybrid Memory Systems
- The Memory Interference/QoS Problem and Solutions
  - QoS-Aware Memory Systems
- How Can We Do Better?
- Summary

Principles (So Far)

- Better cooperation between devices/circuits, hardware and the system
  - Expose more information about devices and controllers to upper layers

- Better-than-worst-case design
  - Do not optimize for worst case
  - Worst case should not determine the common case

- Heterogeneity in parameters/design
  - Enables a more efficient design (No one size fits all)

Other Opportunities with Emerging Technologies

- Merging of memory and storage
  - e.g., a single interface to manage all data [Meza+, WEED'13]

- New applications
  - e.g., ultra-fast checkpoint and restore

- More robust system design
  - e.g., reducing data loss

- Memory as an accelerator (or computing element)
  - e.g., enabling efficient search and filtering

Agenda

- Major Trends Affecting Main Memory
- The DRAM Scaling Problem and Solution Directions
  - Tolerating DRAM: New DRAM Architectures
  - Enabling Emerging Technologies: Hybrid Memory Systems
- How Can We Do Better?
- Summary

Summary: Scalable Memory Systems

- Main memory scaling problems are a critical bottleneck for system performance, efficiency, and usability
- Solution 1: Tolerate DRAM with novel architectures
  - RAIDR: Retention-aware refresh
  - TL-DRAM: Tiered-Latency DRAM
  - RowClone: Fast Page Copy and Initialization
  - SALP: Subarray-Level Parallelism
- Solution 2: Enable emerging memory technologies
  - Replace DRAM with NVM by architecting NVM chips well
  - Hybrid memory systems with automatic data management
- QoS-aware Memory Systems
  - Required to reduce and control memory interference
- Software/hardware/device cooperation essential for effective scaling of main memory

Thank you.

Scaling the Memory System in the Many-Core Era

Onur Mutlu
onur@cmu.edu

July 26, 2013
BSC/UPC

Carnegie Mellon

SAFARI

Backup Slides

Backup Slides Agenda

- RAIDR: Retention-Aware DRAM Refresh
- Building Large DRAM Caches for Hybrid Memories
- Memory QoS and Predictable Performance
- Subarray-Level Parallelism (SALP) in DRAM

RAIDR: Reducing DRAM Refresh Impact

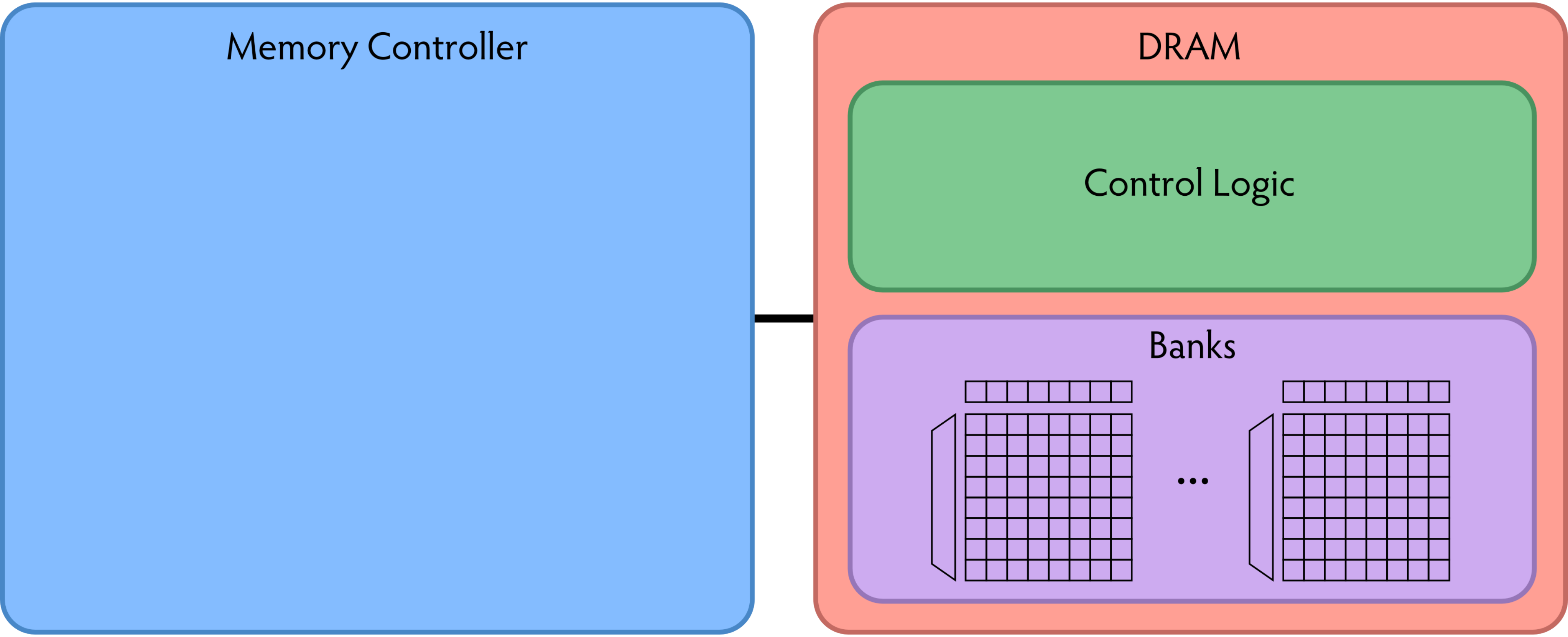
Tolerating Temperature Changes

- 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

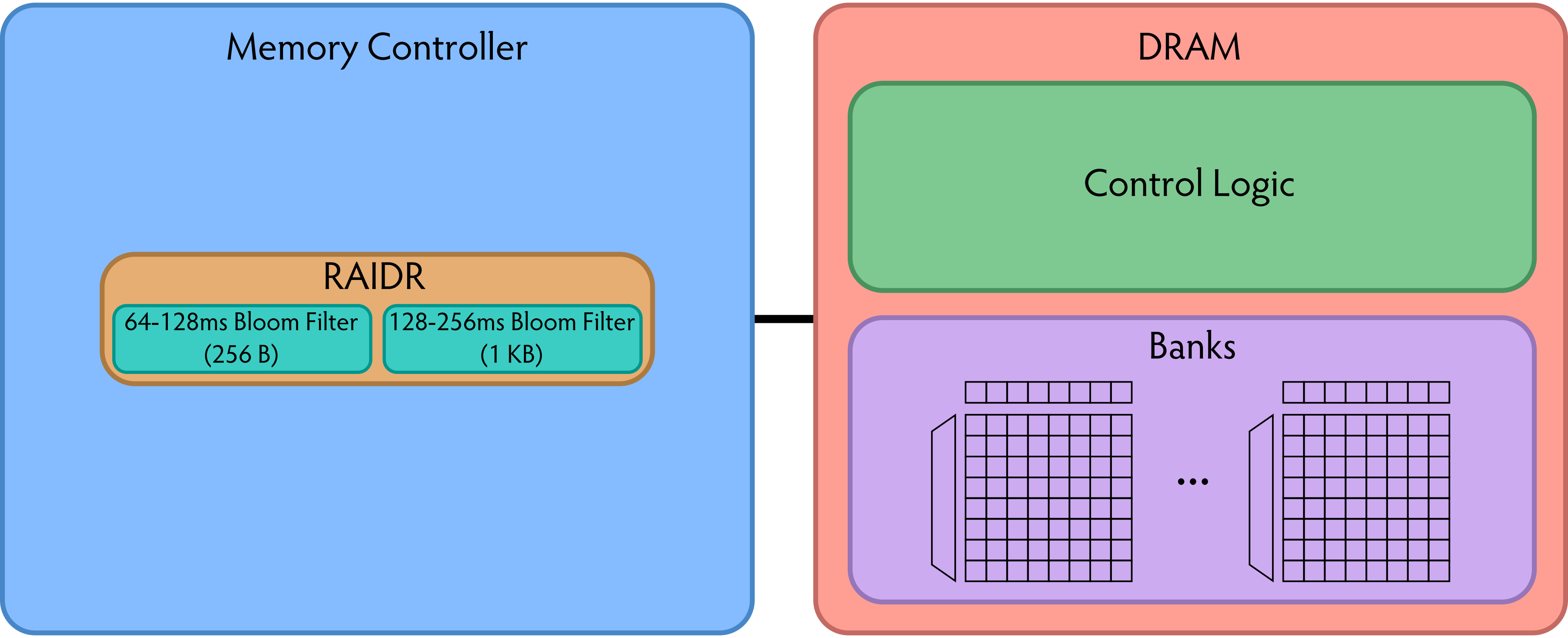
RAIDR: Baseline Design



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

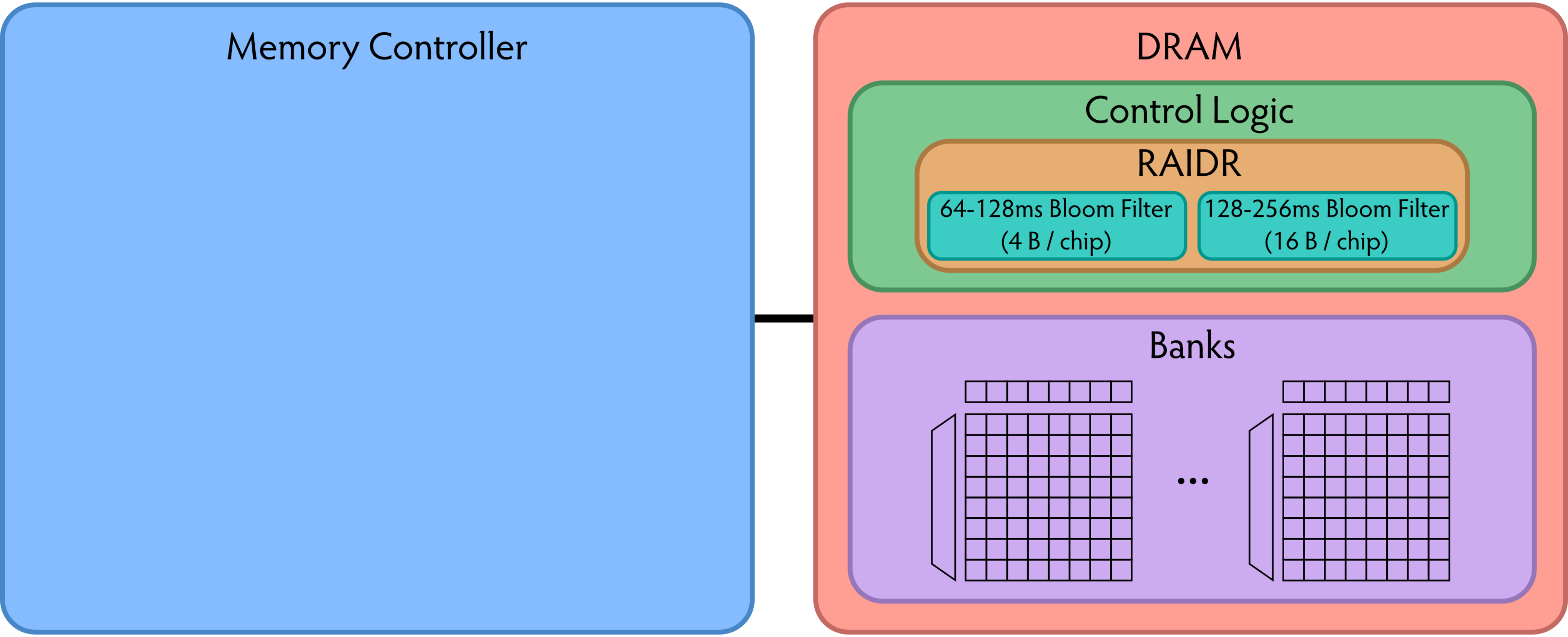
RAIDR can be implemented in either the controller or DRAM

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

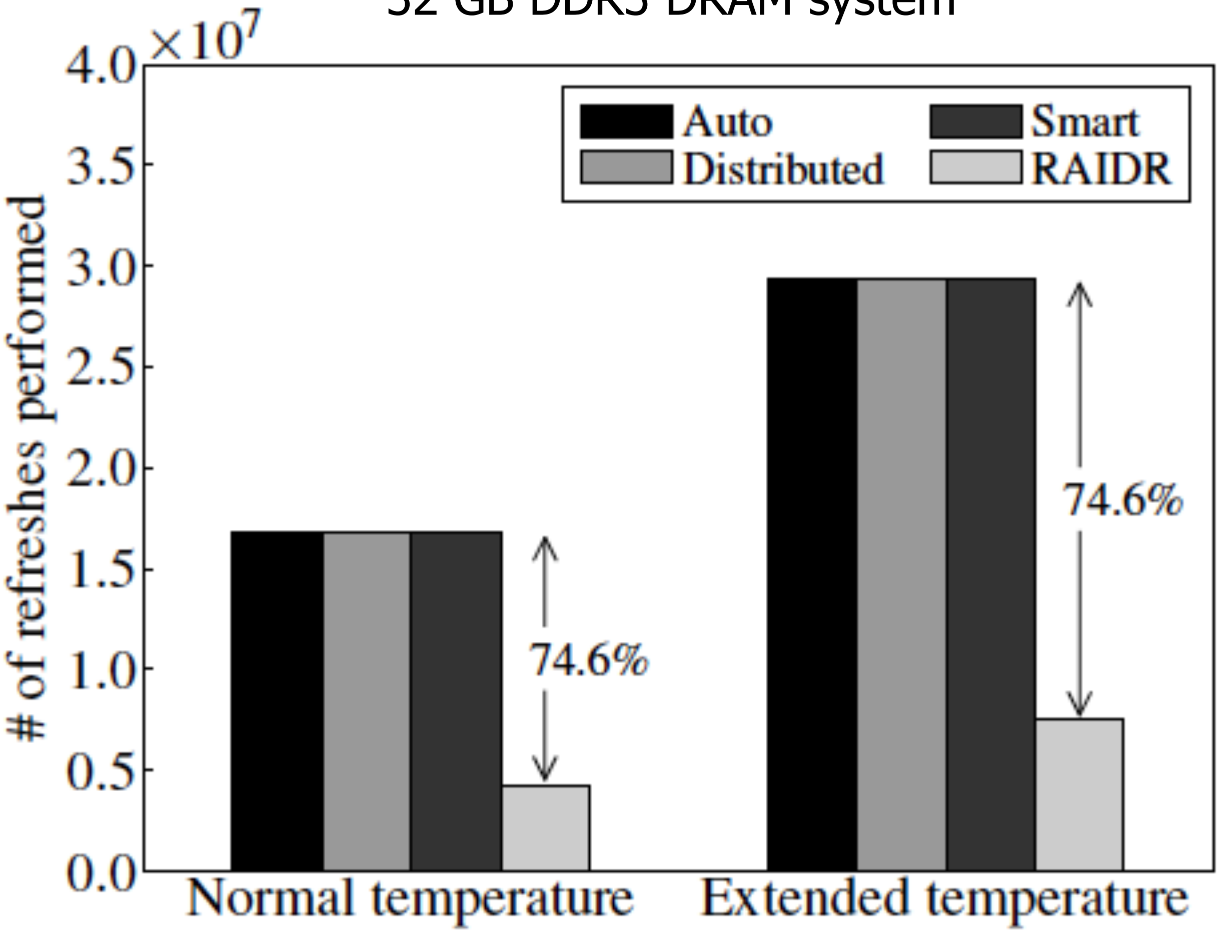
- Baseline:
  - 32 GB DDR3 DRAM system (8 cores, 512KB cache/core)
  - 64ms refresh interval for all rows

- 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

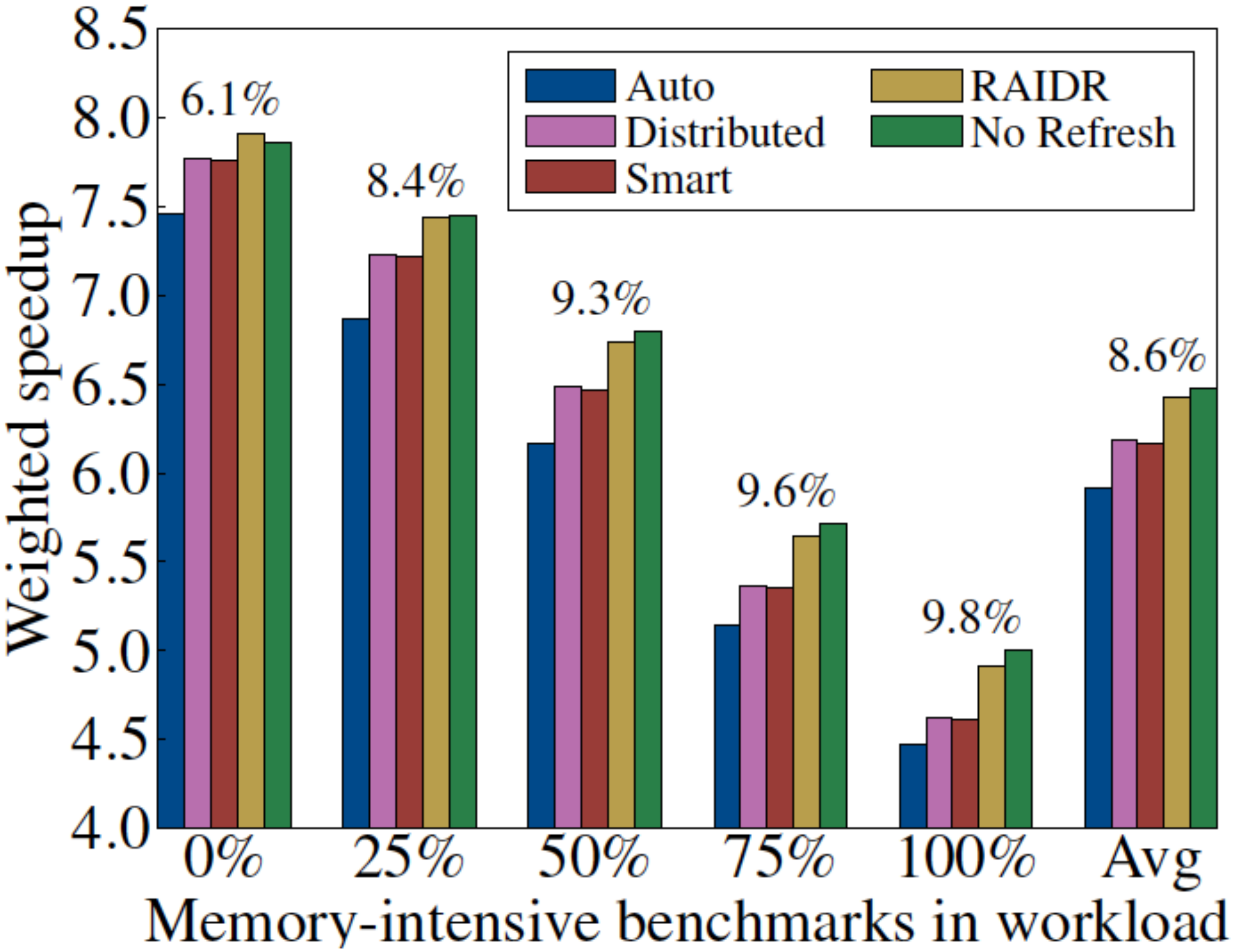
- Results on SPEC CPU2006, TPC-C, TPC-H benchmarks
  - 74.6% refresh reduction
  - ~16%/20% DRAM dynamic/idle power reduction
  - ~9% performance improvement

RAIDR Refresh Reduction

32 GB DDR3 DRAM system

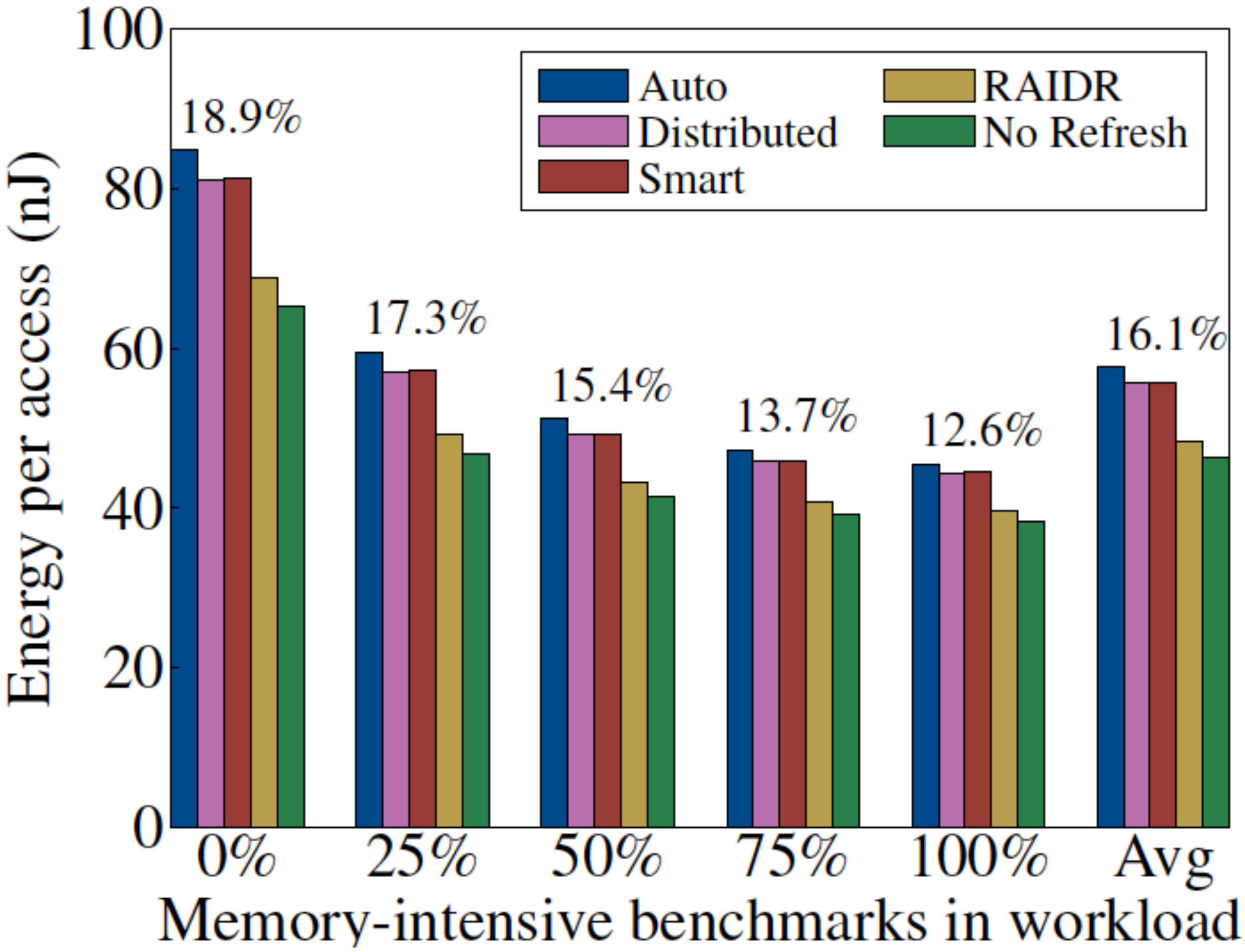


RAIDR: Performance



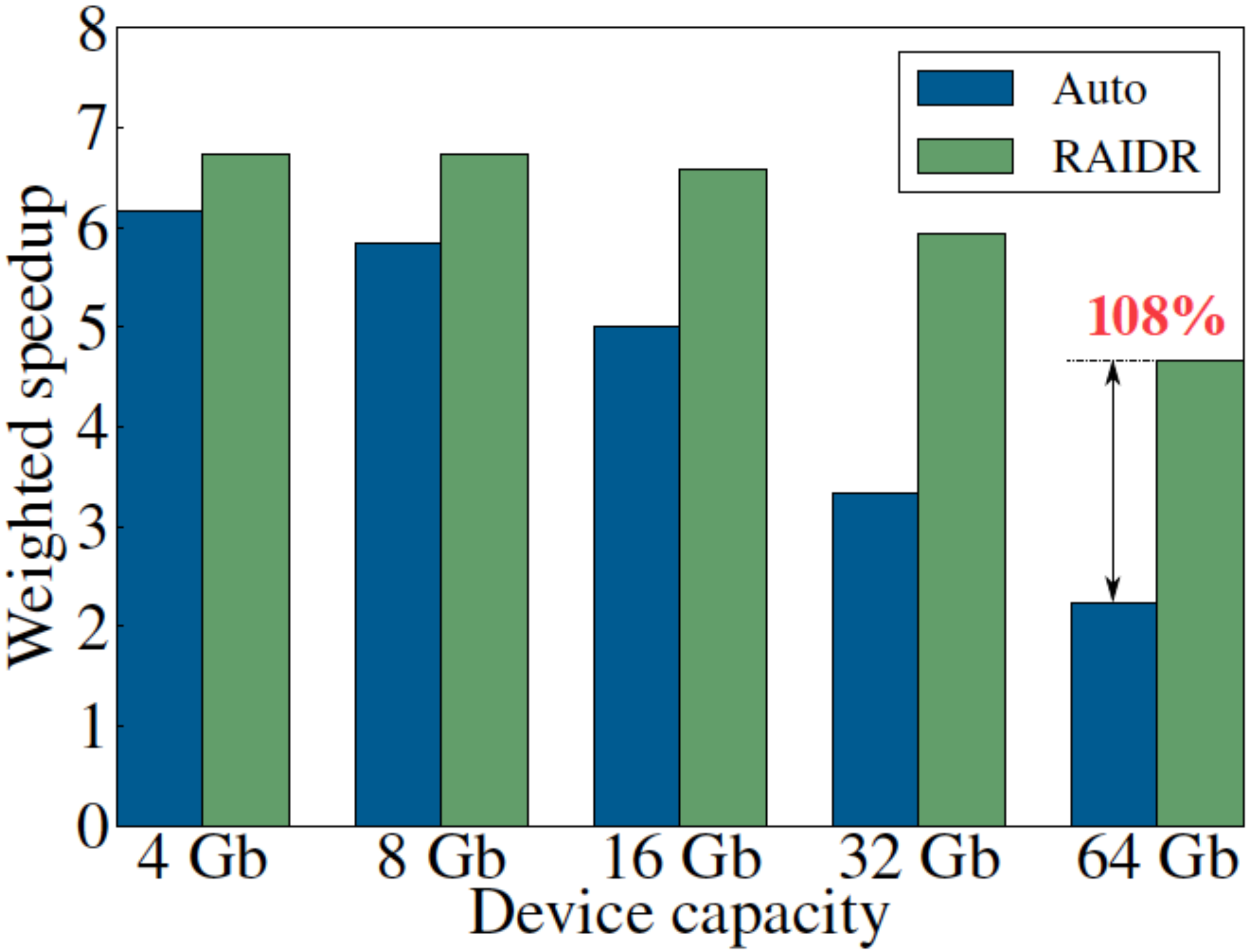
RAIDR performance benefits increase with workload's memory intensity

RAIDR: DRAM Energy Efficiency



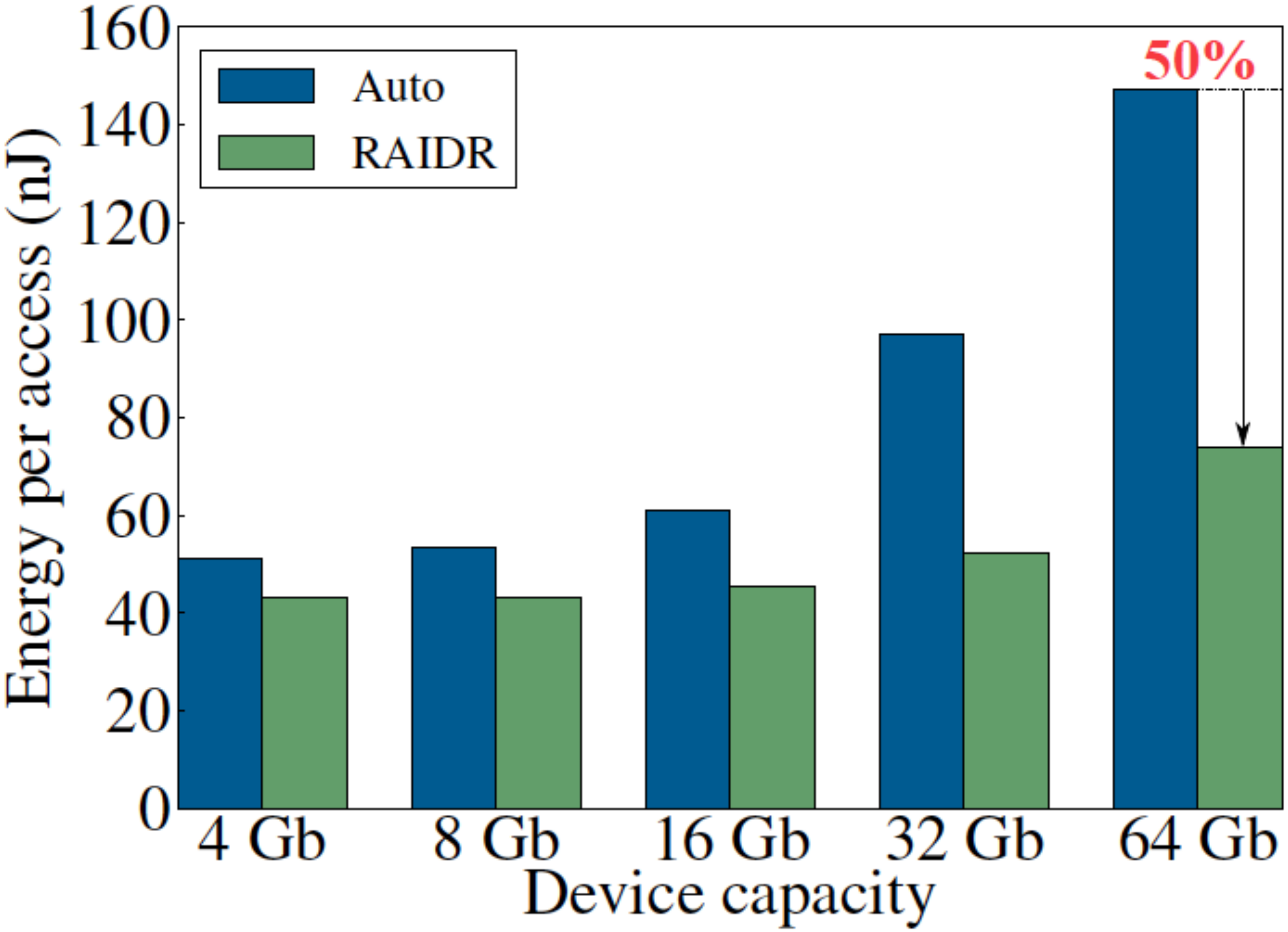
RAIDR energy benefits increase with memory idleness

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

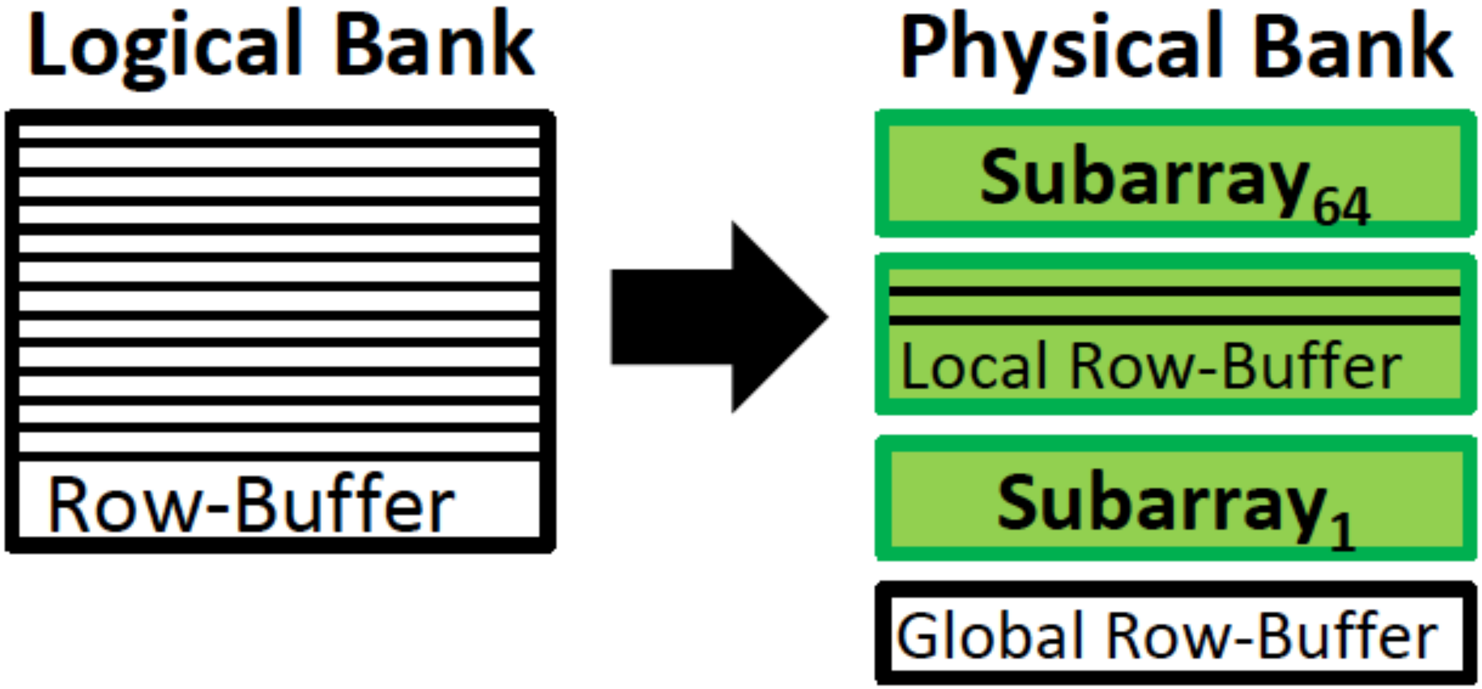
RAIDR slides

SALP: Reducing DRAM Bank Conflict Impact

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

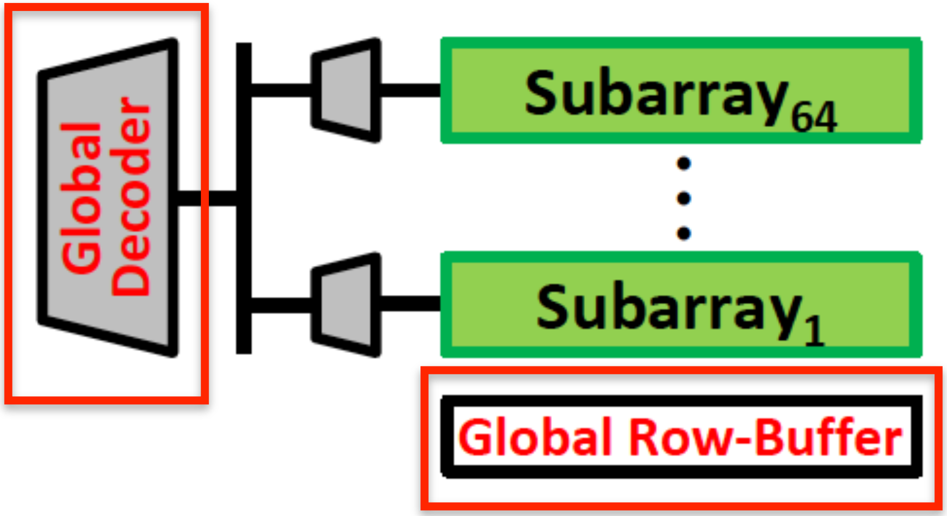
SALP: Problem, Goal, Observations

- Problem: Bank conflicts are costly for performance and energy
  - serialized requests, wasted energy (thrashing of row buffer, busy wait)
- Goal: Reduce bank conflicts without adding more banks (low cost)
- Observation 1: A DRAM bank is divided into subarrays and each subarray has its own local row buffer



SALP: Key Ideas

- Observation 2: Subarrays are mostly independent
  - Except when sharing global structures to reduce cost

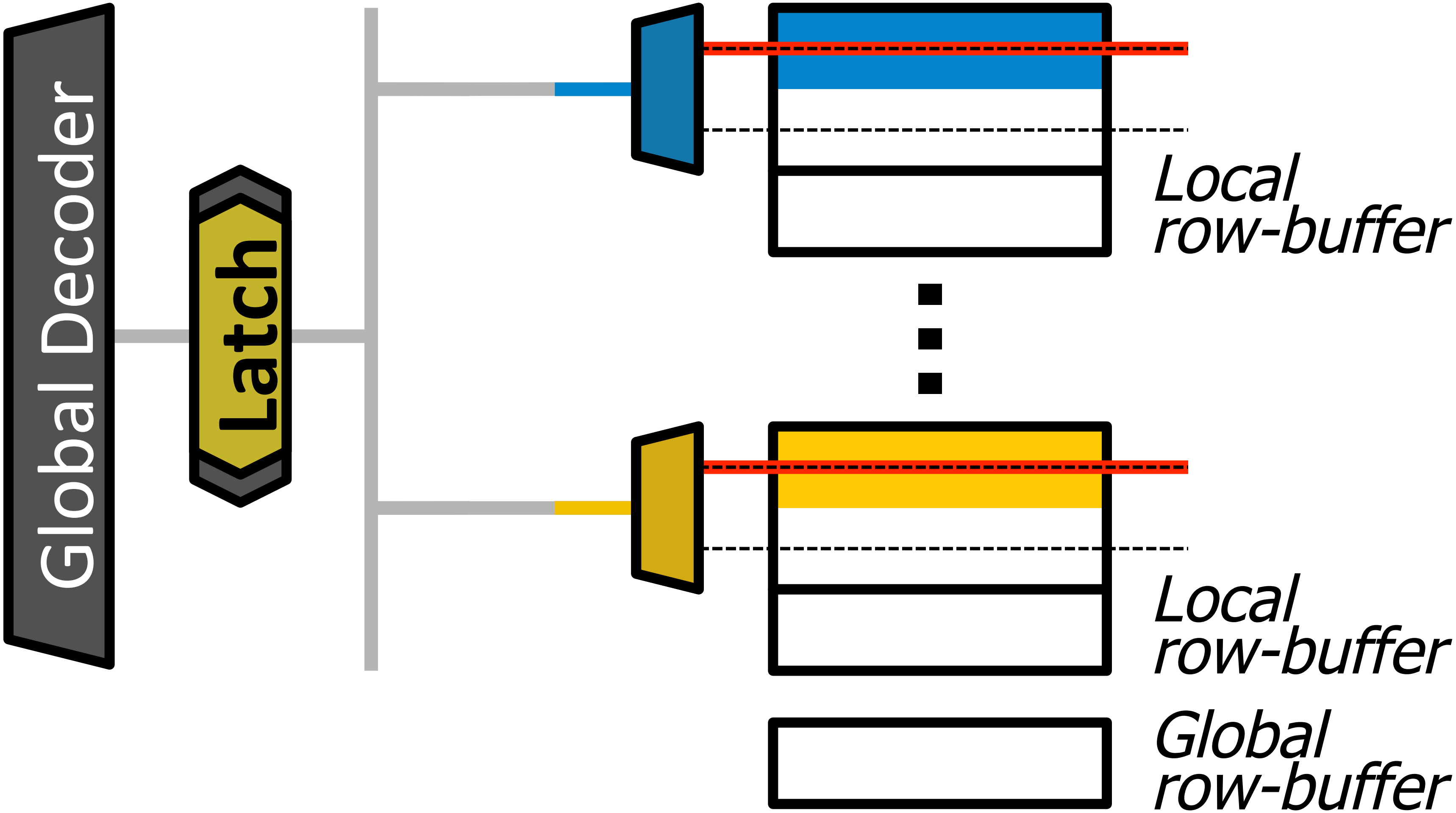


Key Idea of SALP: Minimally reduce sharing of global structures

Reduce the sharing of …
Global decoder → Enables almost parallel access to subarrays
Global row buffer → Utilizes multiple local row buffers

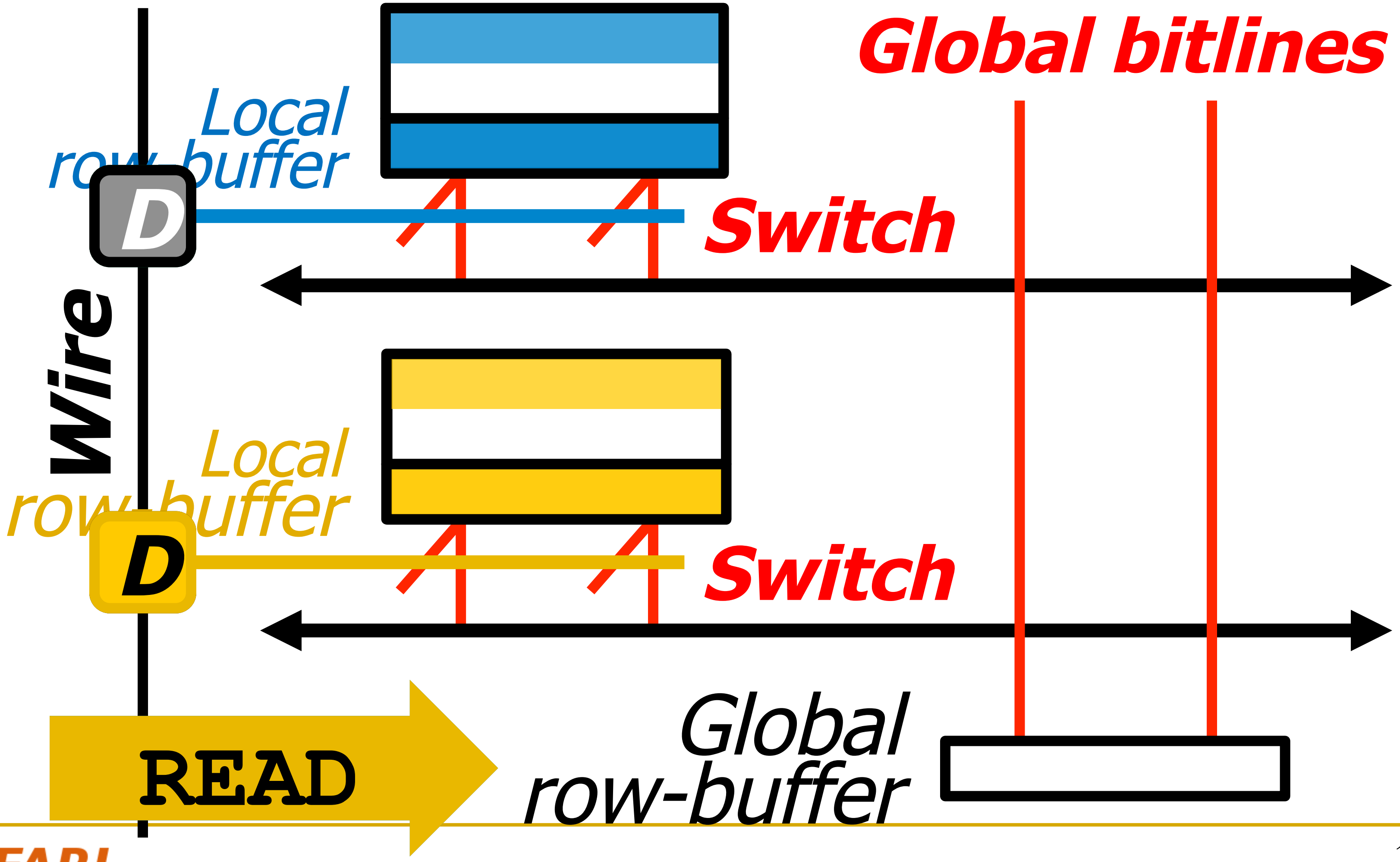
SALP: Reduce Sharing of Global Decoder

Instead of a global latch, have ***per-subarray latches***

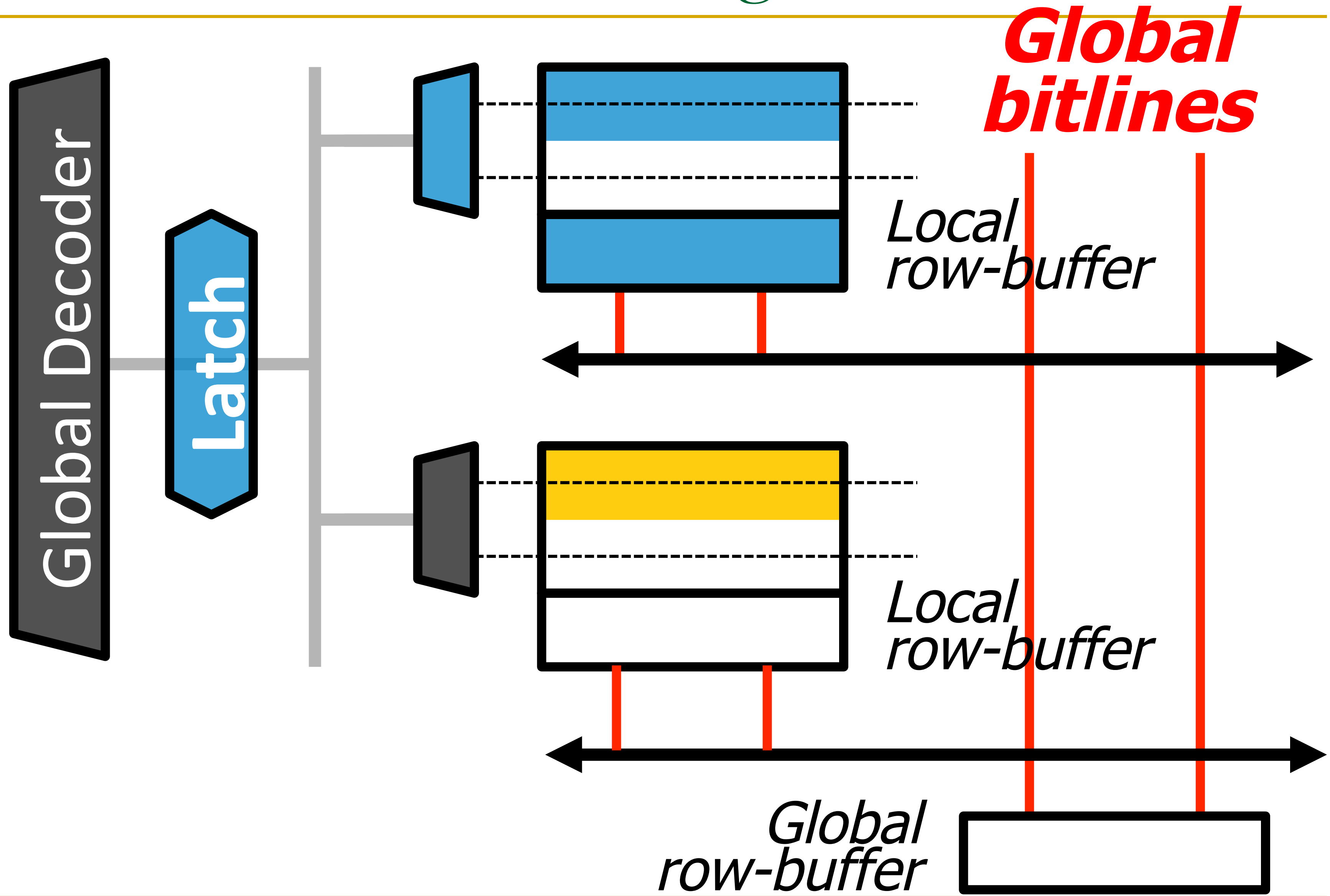


SALP: Reduce Sharing of Global Row-Buffer

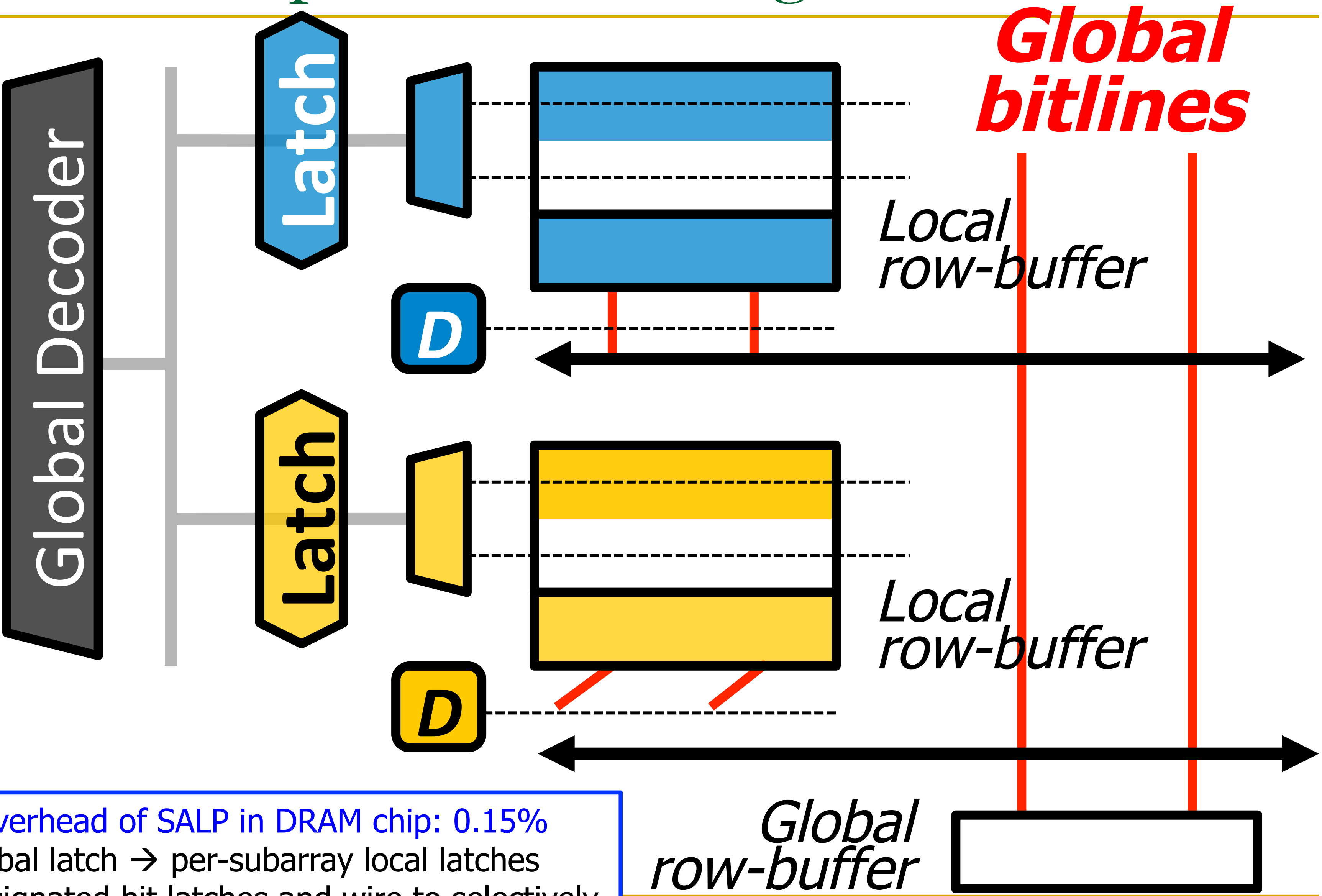
Selectively connect local row-buffers to global row-buffer using a ***Designated*** single-bit latch



SALP: Baseline Bank Organization



SALP: Proposed Bank Organization



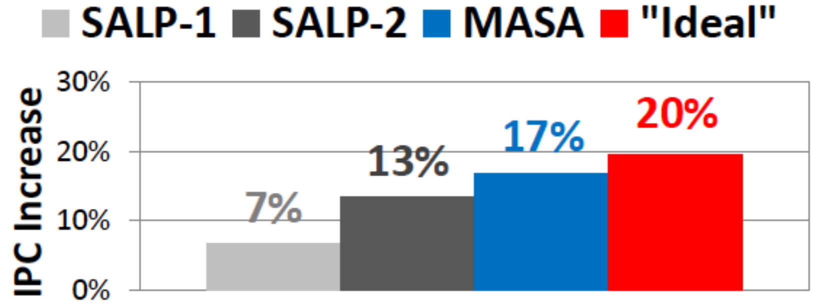
Overhead of SALP in DRAM chip: 0.15%

1. Global latch → per-subarray local latches
2. Designated bit latches and wire to selectively enable a subarray

SALP: Results

- Wide variety of systems with different #channels, banks, ranks, subarrays
- Server, streaming, random-access, SPEC workloads

- Dynamic DRAM energy reduction: 19%
  - DRAM row hit rate improvement: 13%
- System performance improvement: 17%
  - Within 3% of ideal (all independent banks)
- DRAM die area overhead: 0.15%
  - vs. 36% overhead of independent banks



Die-Size	< 0.15%	0.15%	36.3%