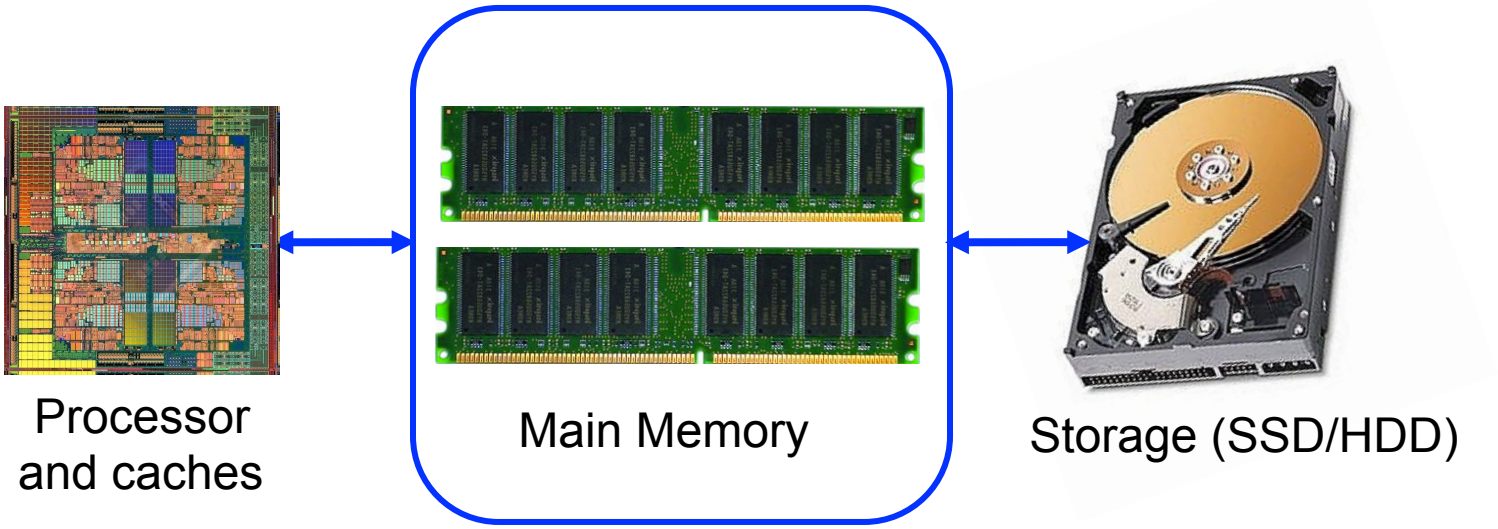
Memory Scaling: A Systems Architecture Perspective

Onur Mutlu
onur@cmu.edu
May 27, 2013
IMW 2013

Carnegie Mellon

SAFARI

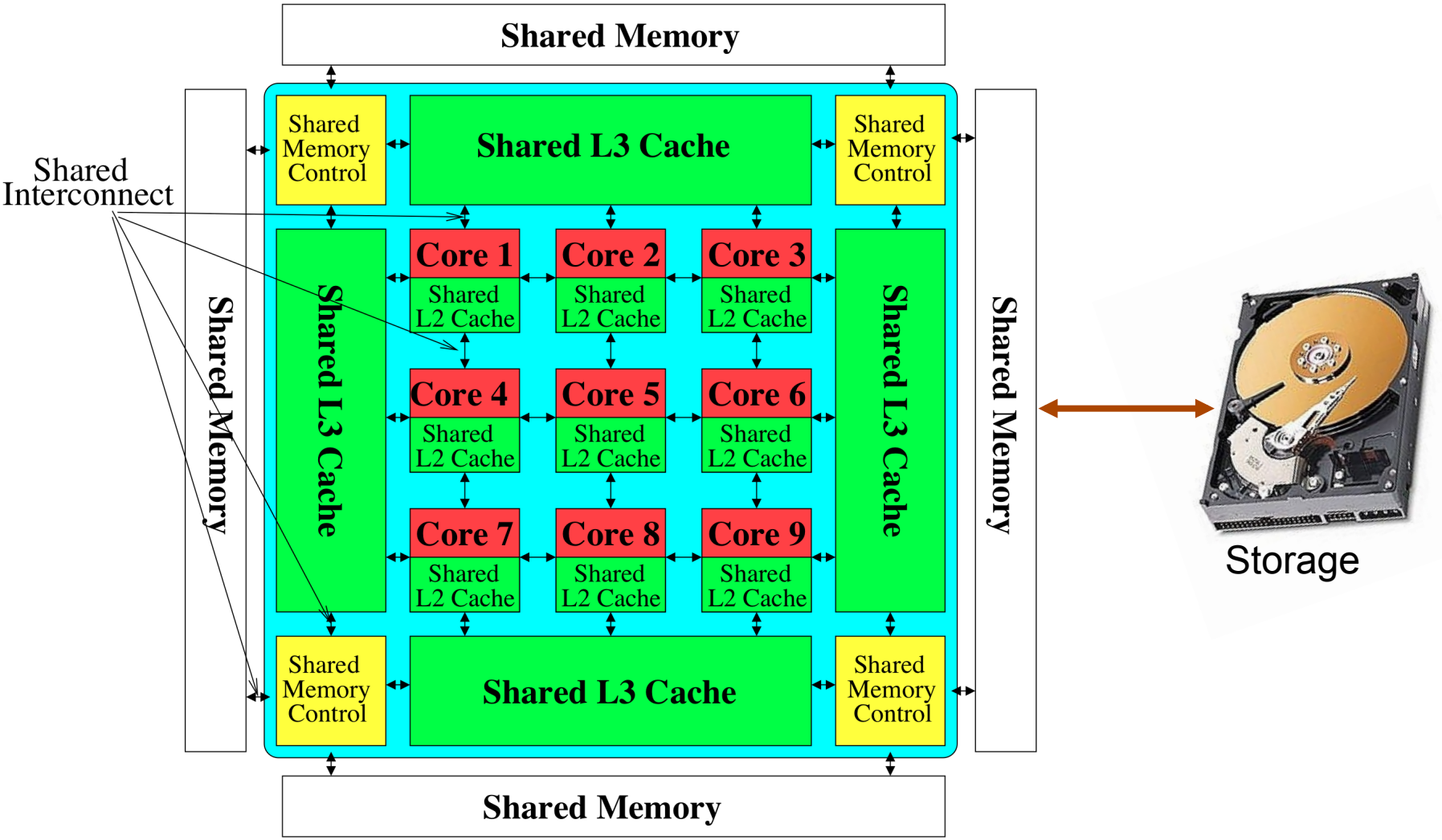
The Main Memory System



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

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

Memory System: A *Shared Resource* View



State of the Main Memory System

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

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

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

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

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

Major Trends Affecting Main Memory (I)

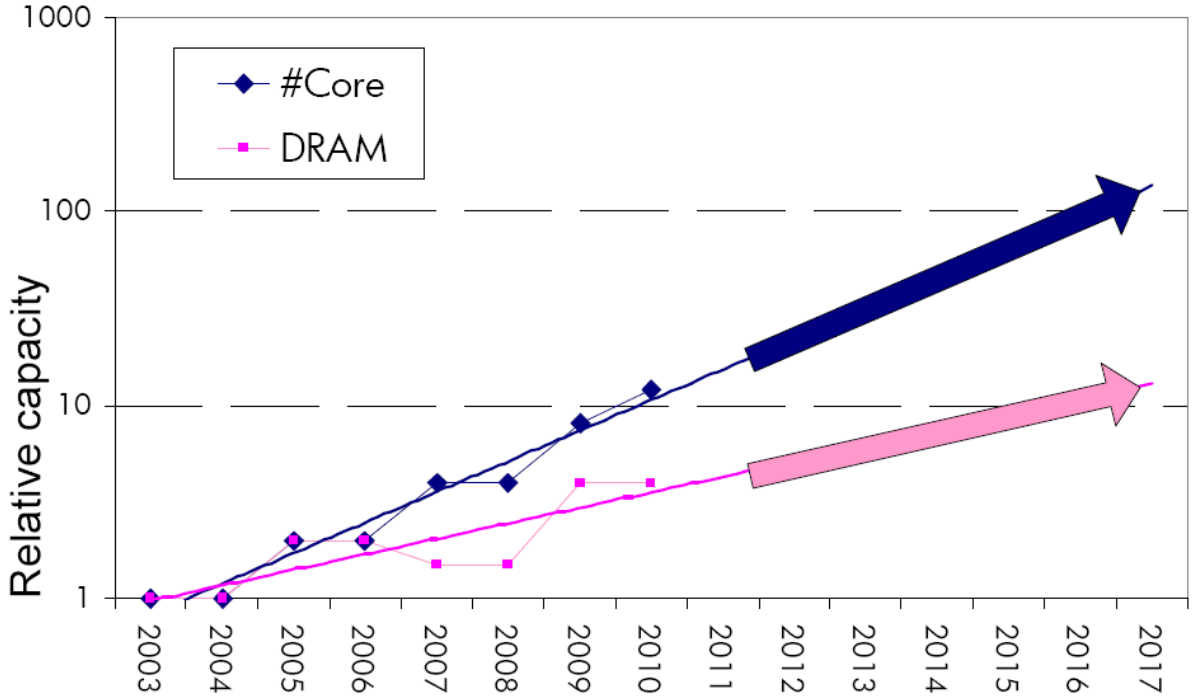
- Need for main memory capacity, bandwidth, QoS increasing
- Main memory energy/power is a key system design concern
- DRAM technology scaling is ending

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

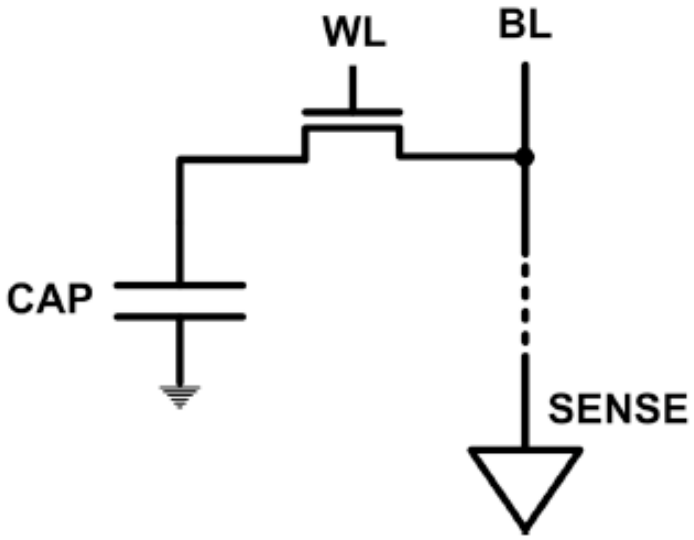
- DRAM technology scaling is ending
  - ITRS projects DRAM will not scale easily below X nm
  - Scaling has provided many benefits:
    - higher capacity (density), lower cost, lower energy

Agenda

- Major Trends Affecting Main Memory
- The DRAM Scaling Problem and Solution Directions
  - Tolerating DRAM: New DRAM Architectures
  - Enabling Emerging Technologies: Hybrid Memory Systems
- 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

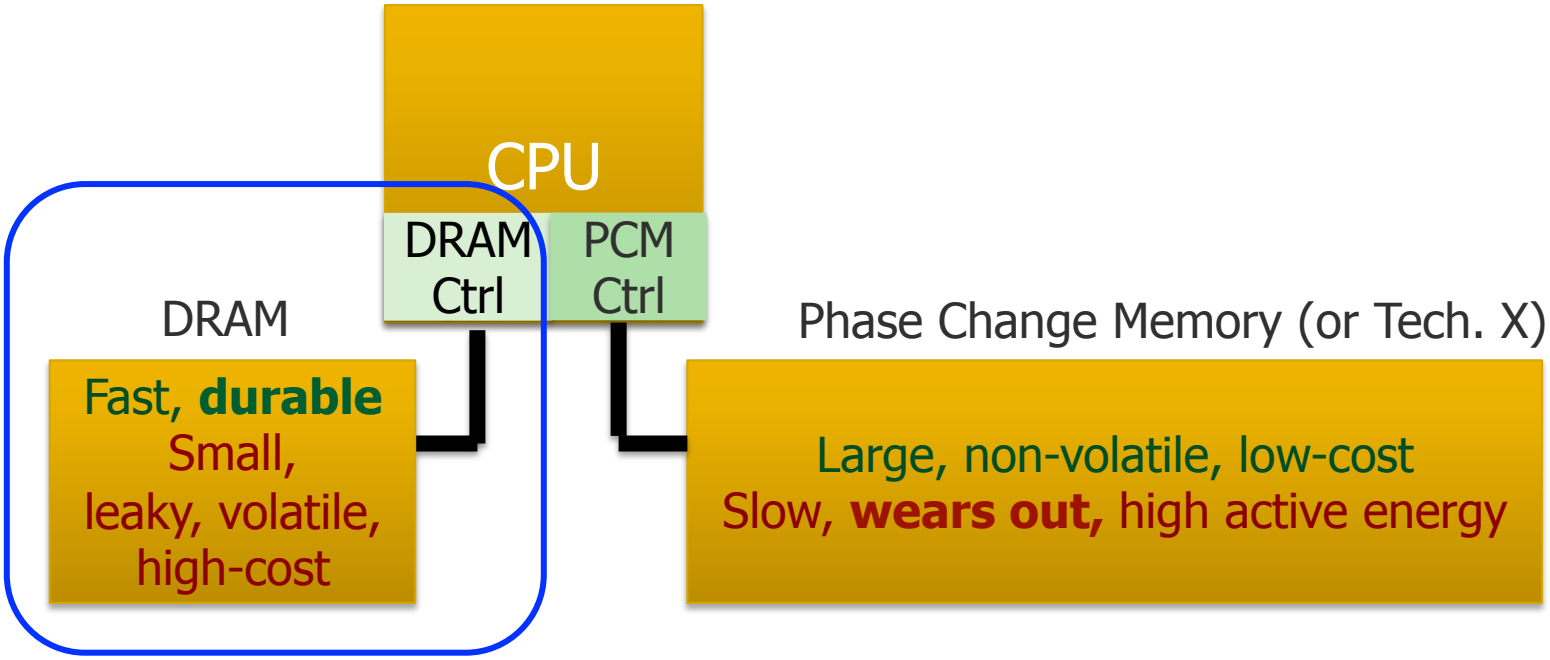
- Liu, Jaiyen, Veras, Mutlu, "RAIDR: Retention-Aware Intelligent DRAM Refresh," ISCA 2012.
- Kim, Seshadri, Lee+, "A Case for Exploiting Subarray-Level Parallelism in DRAM," ISCA 2012.
- Lee+, "Tiered-Latency DRAM: A Low Latency and Low Cost DRAM Architecture," HPCA 2013.
- Liu+, "An Experimental Study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

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



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

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

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

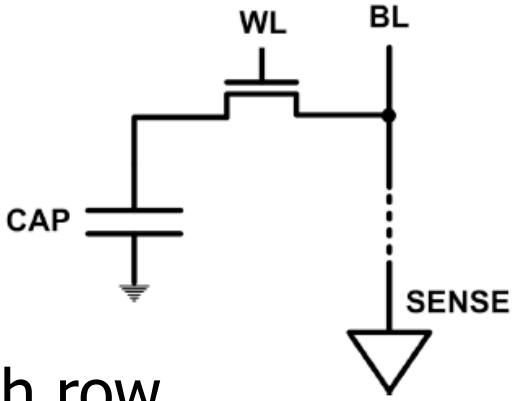
Agenda

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

Tolerating DRAM: Example Techniques

- Retention-Aware DRAM Refresh
- Tiered-Latency DRAM
- In-Memory Page Copy and Initialization
- Subarray-Level Parallelism

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

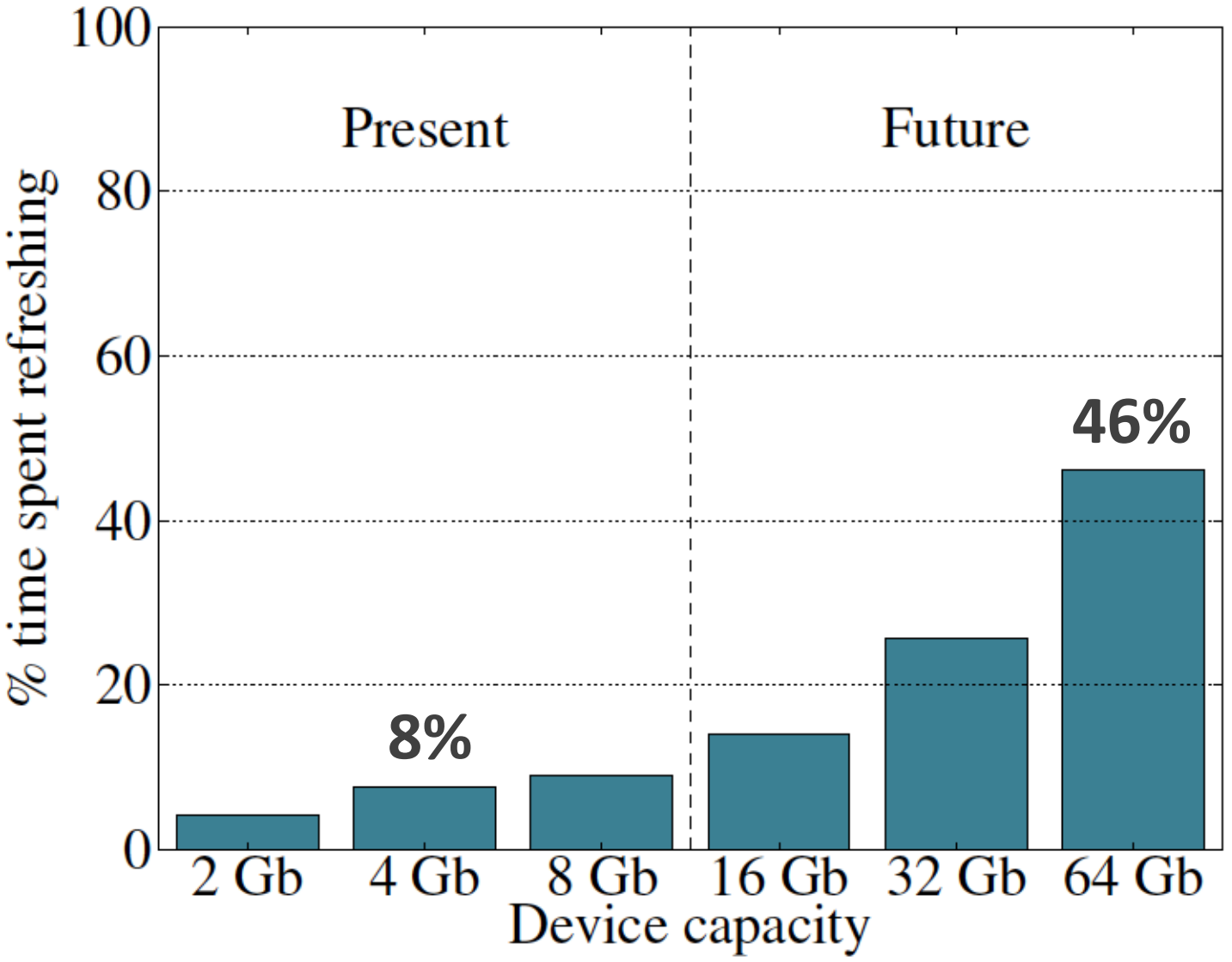
  -- Energy consumption: Each refresh consumes energy

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

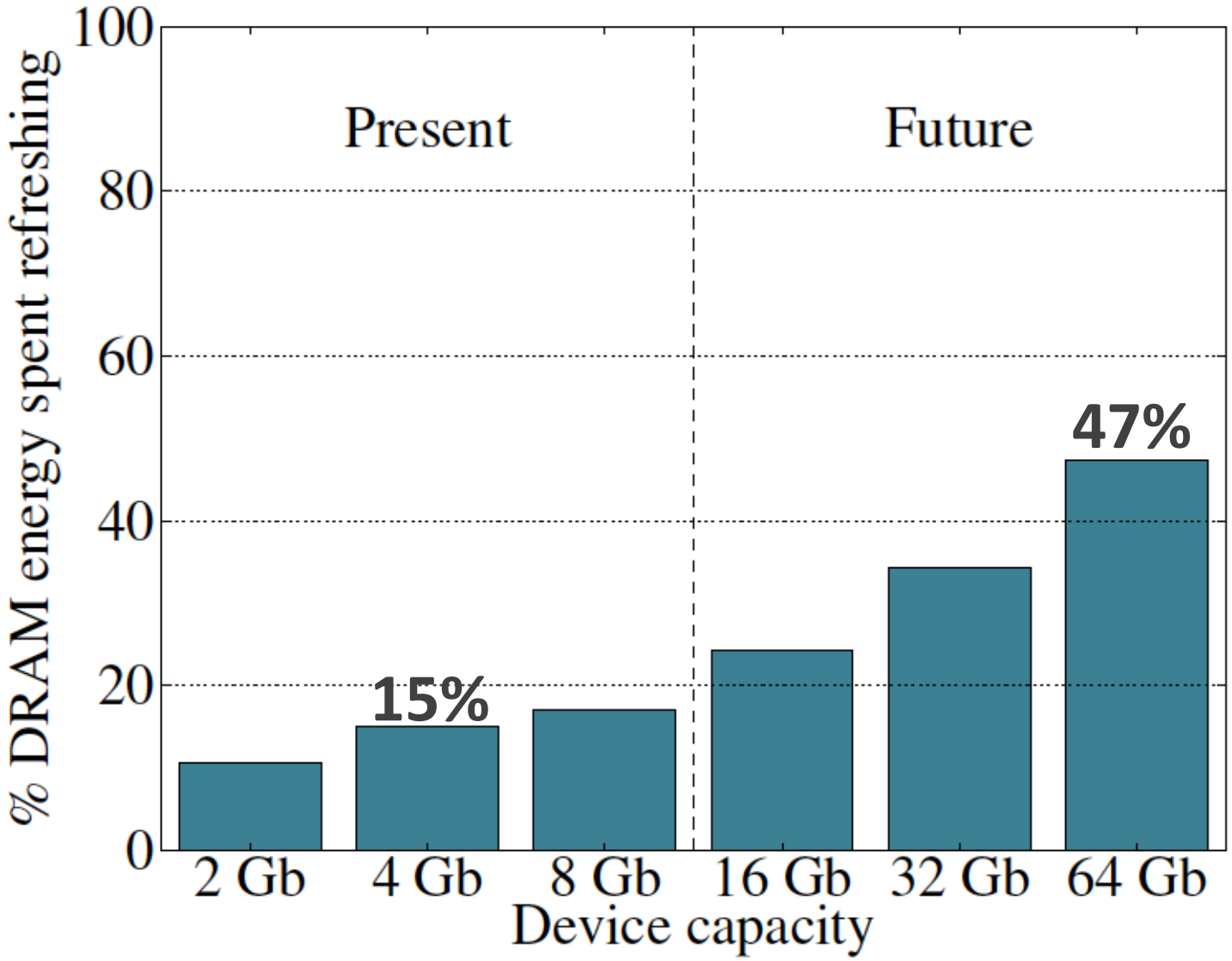
  -- QoS/predictability impact: (Long) pause times during refresh

  -- Refresh rate limits DRAM capacity scaling

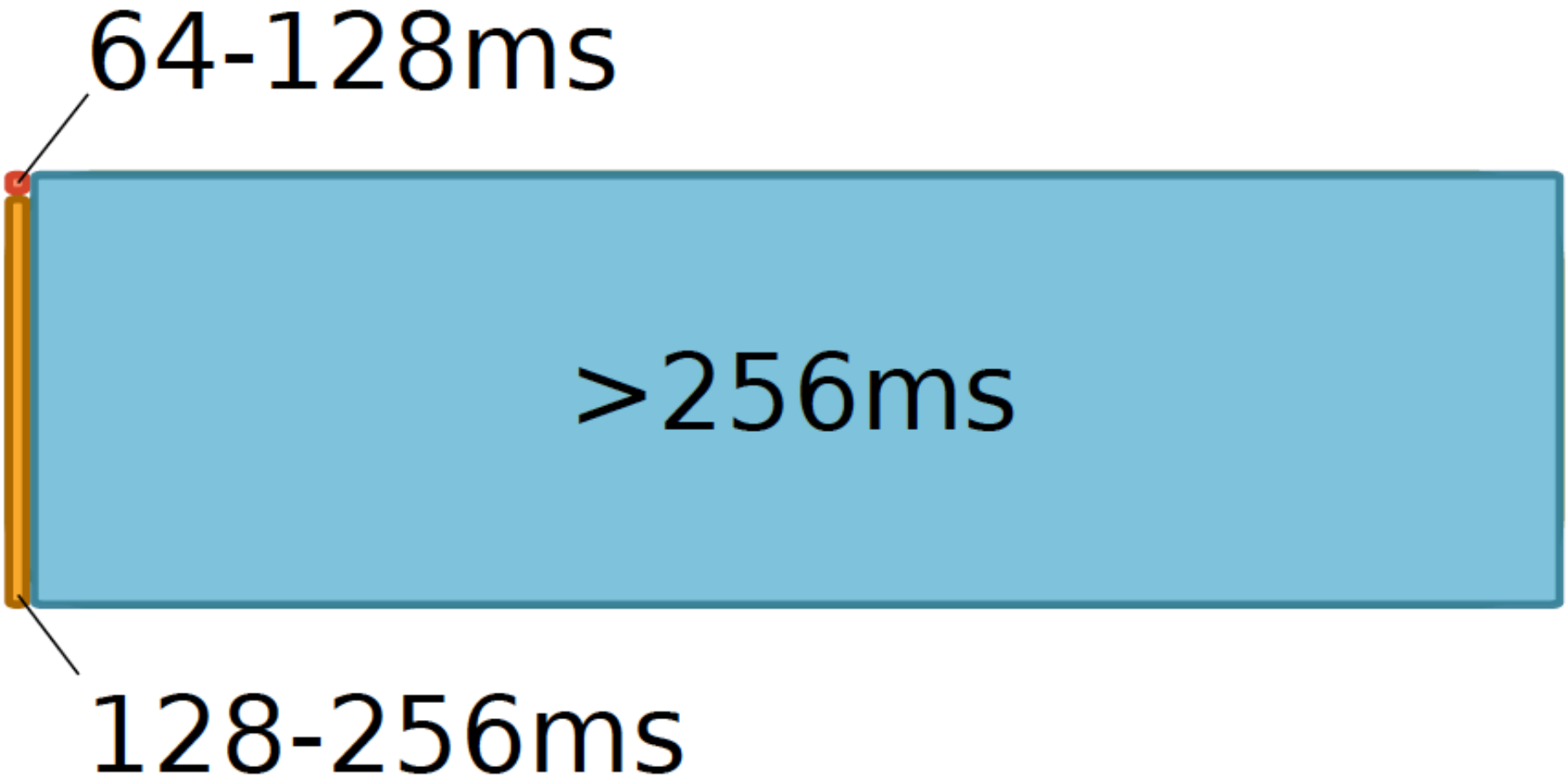
Refresh Overhead: Performance



Refresh Overhead: Energy

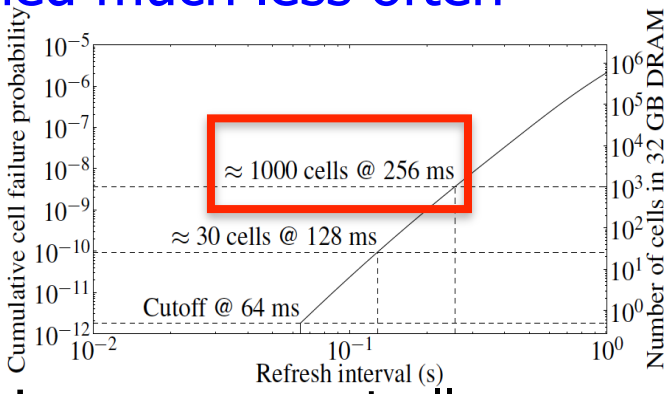


Retention Time Profile of DRAM



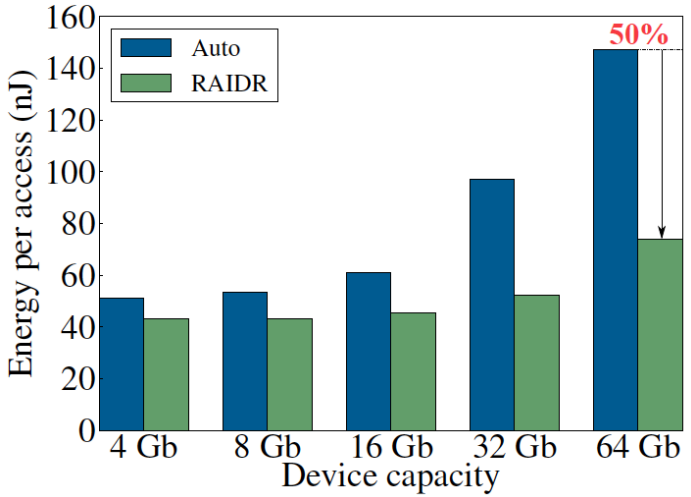
RAIDR: Eliminating Unnecessary Refreshes

- 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



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

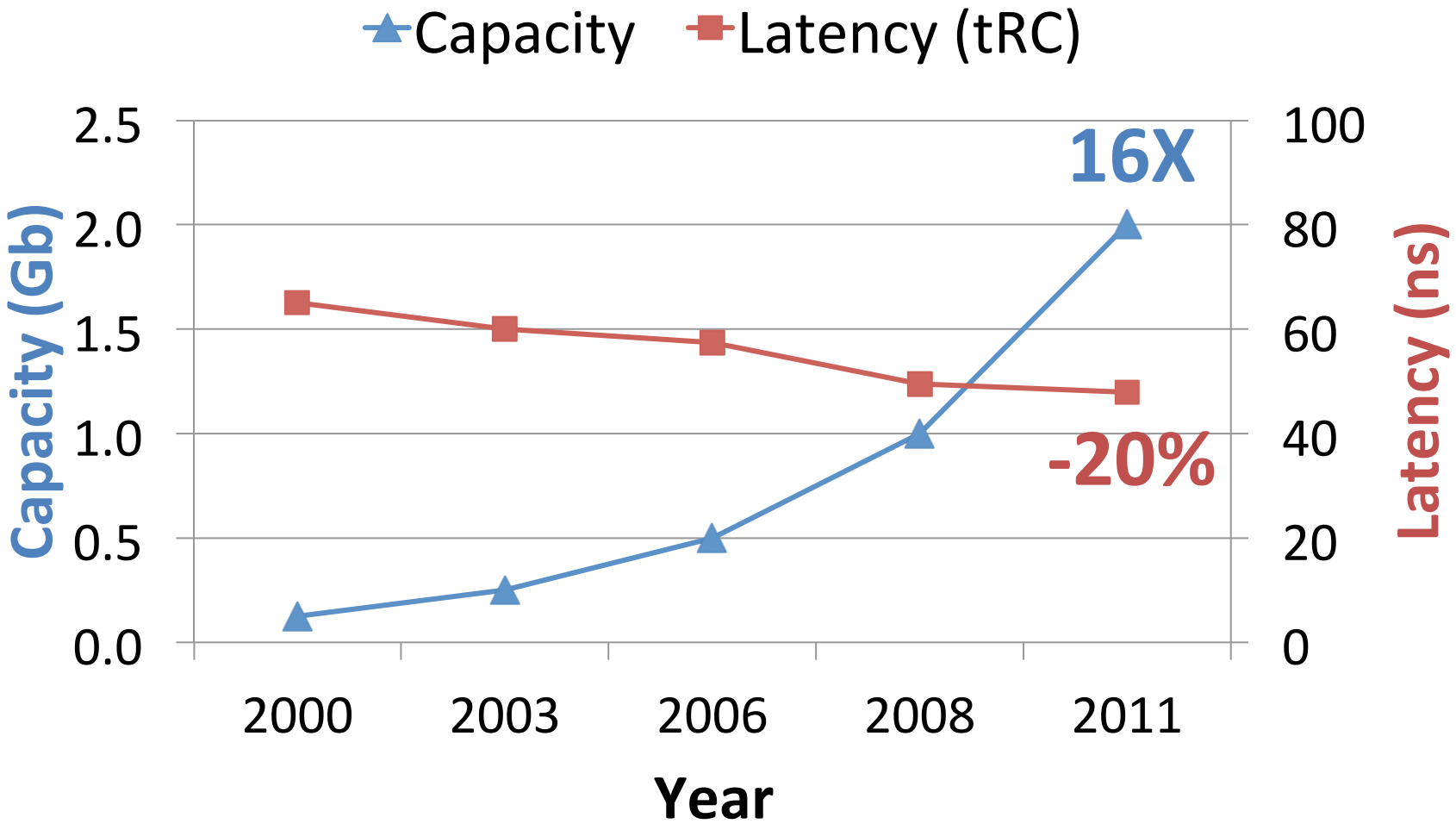
Going Forward

- How to find out and expose weak memory cells/rows
- Tolerating cell-to-cell interference at the system level
  - Flash and DRAM

Tolerating DRAM: Example Techniques

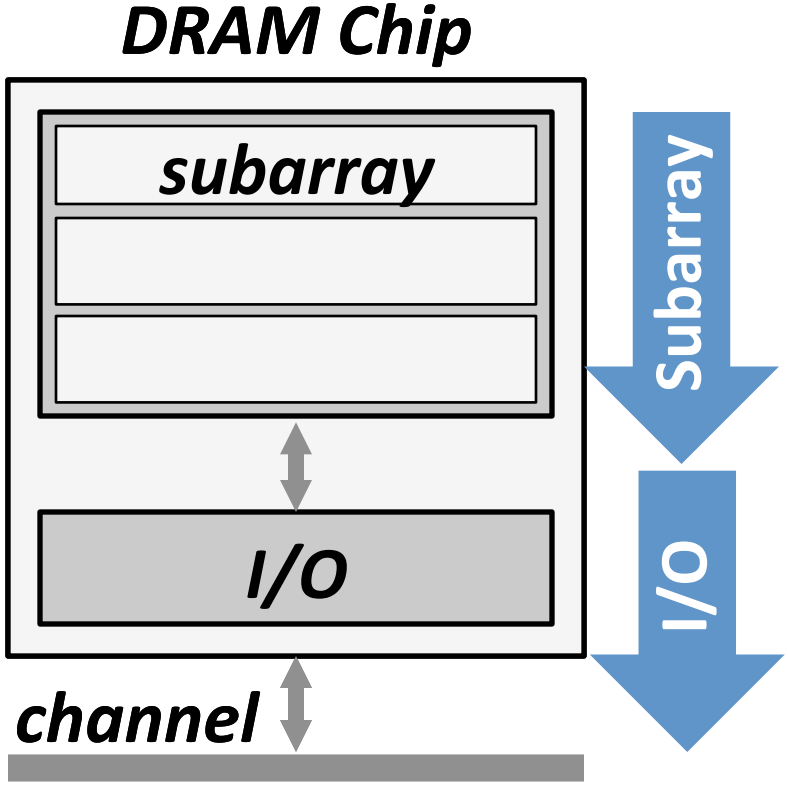
- Retention-Aware DRAM Refresh
- Tiered-Latency DRAM
- In-Memory Page Copy and Initialization
- Subarray-Level Parallelism

DRAM Latency-Capacity Trend



DRAM latency continues to be a critical bottleneck

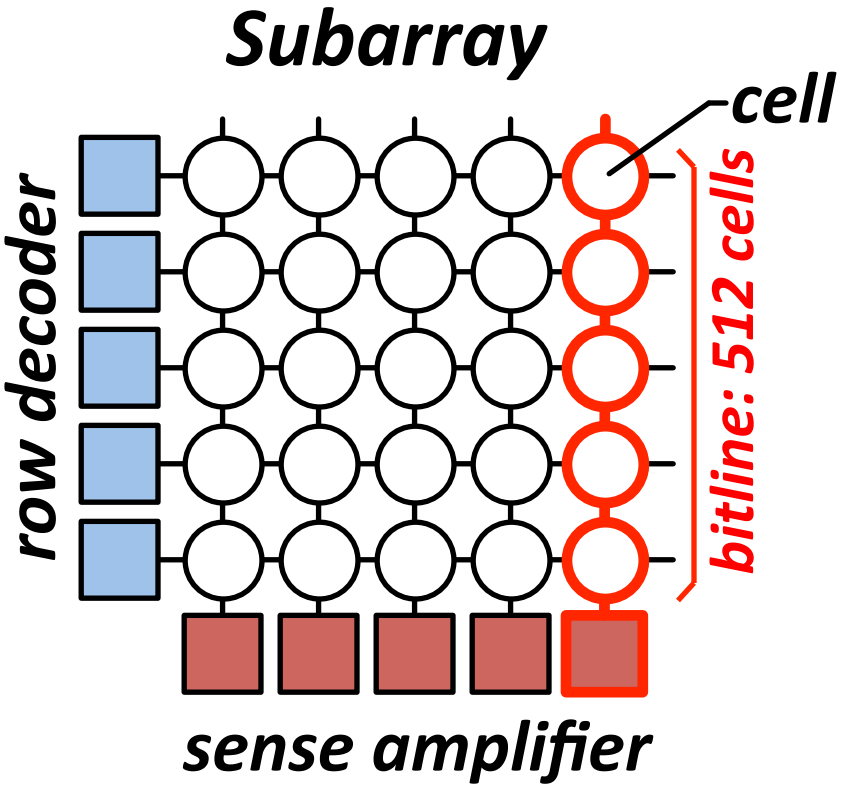
What Causes the Long Latency?

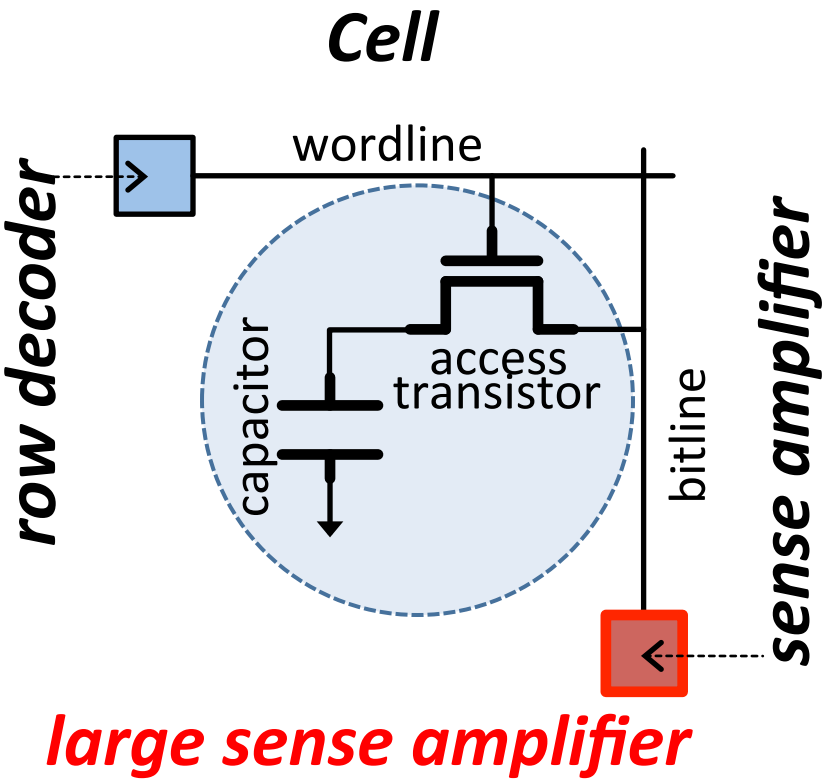


DRAM Latency = Subarray Latency + I/O Latency

Dominant

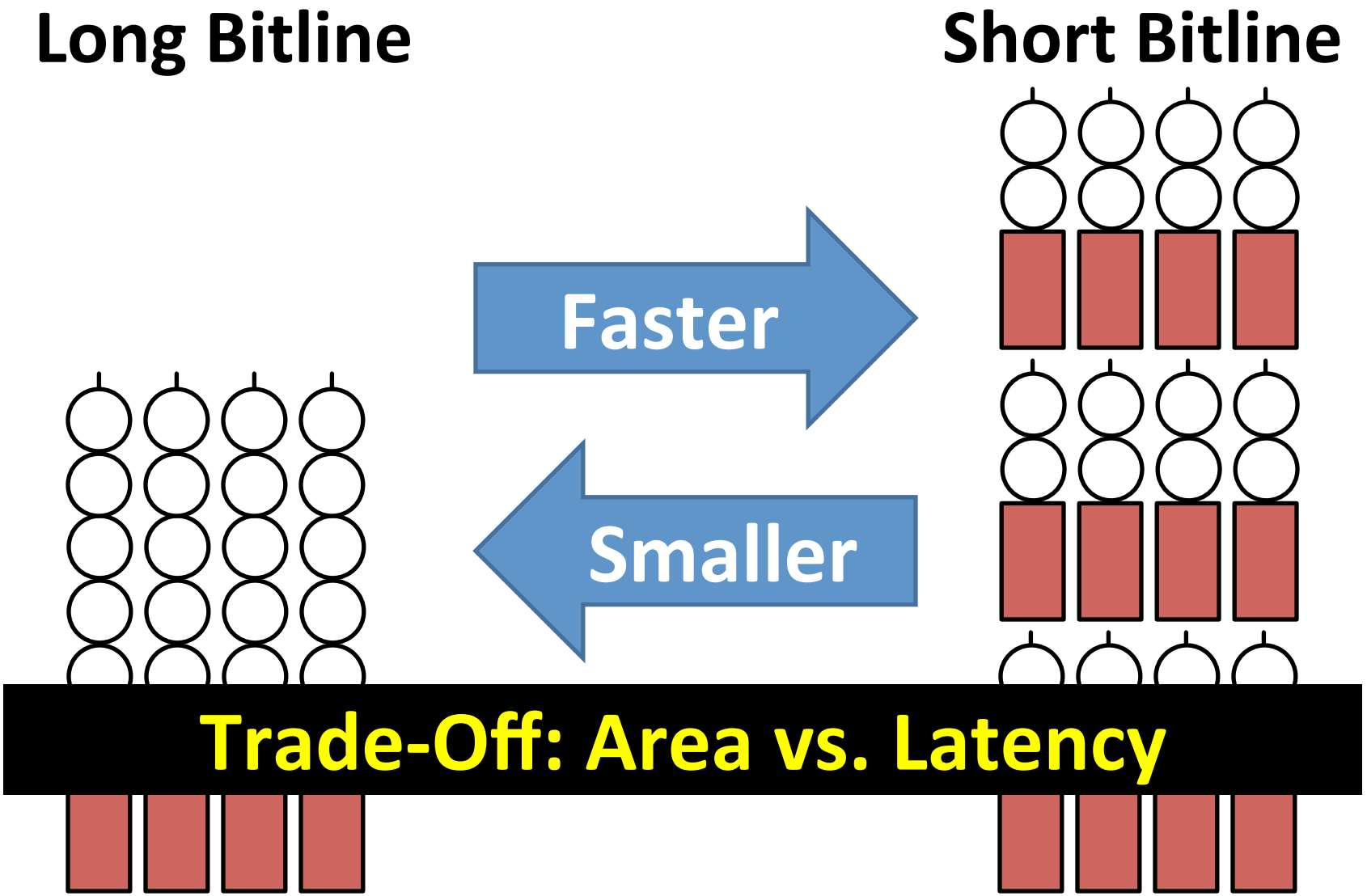
Why is the Subarray So Slow?



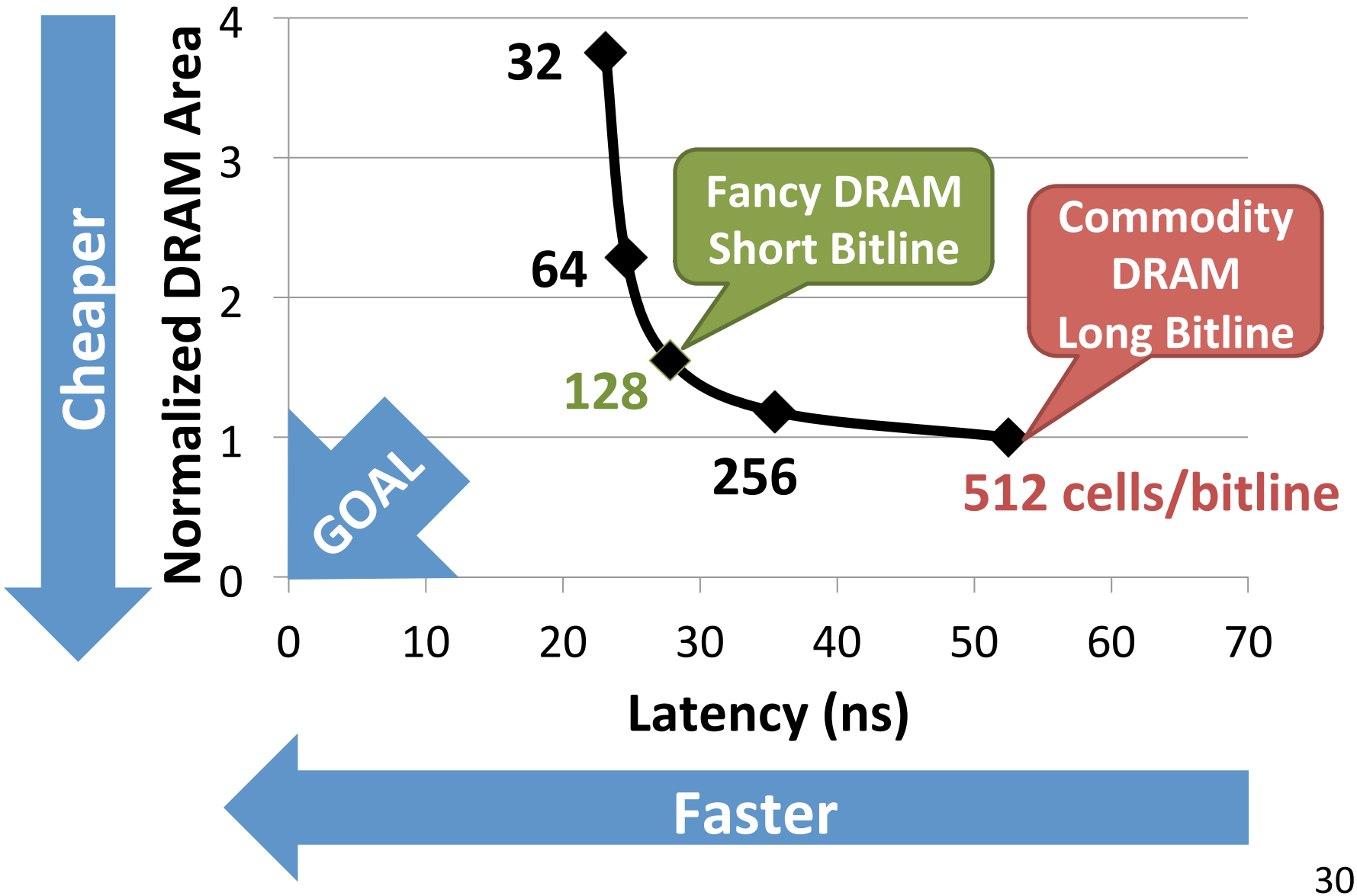


- **Long bitline**
  - **Amortizes sense amplifier cost → Small area**
  - **Large bitline capacitance → High latency & power**

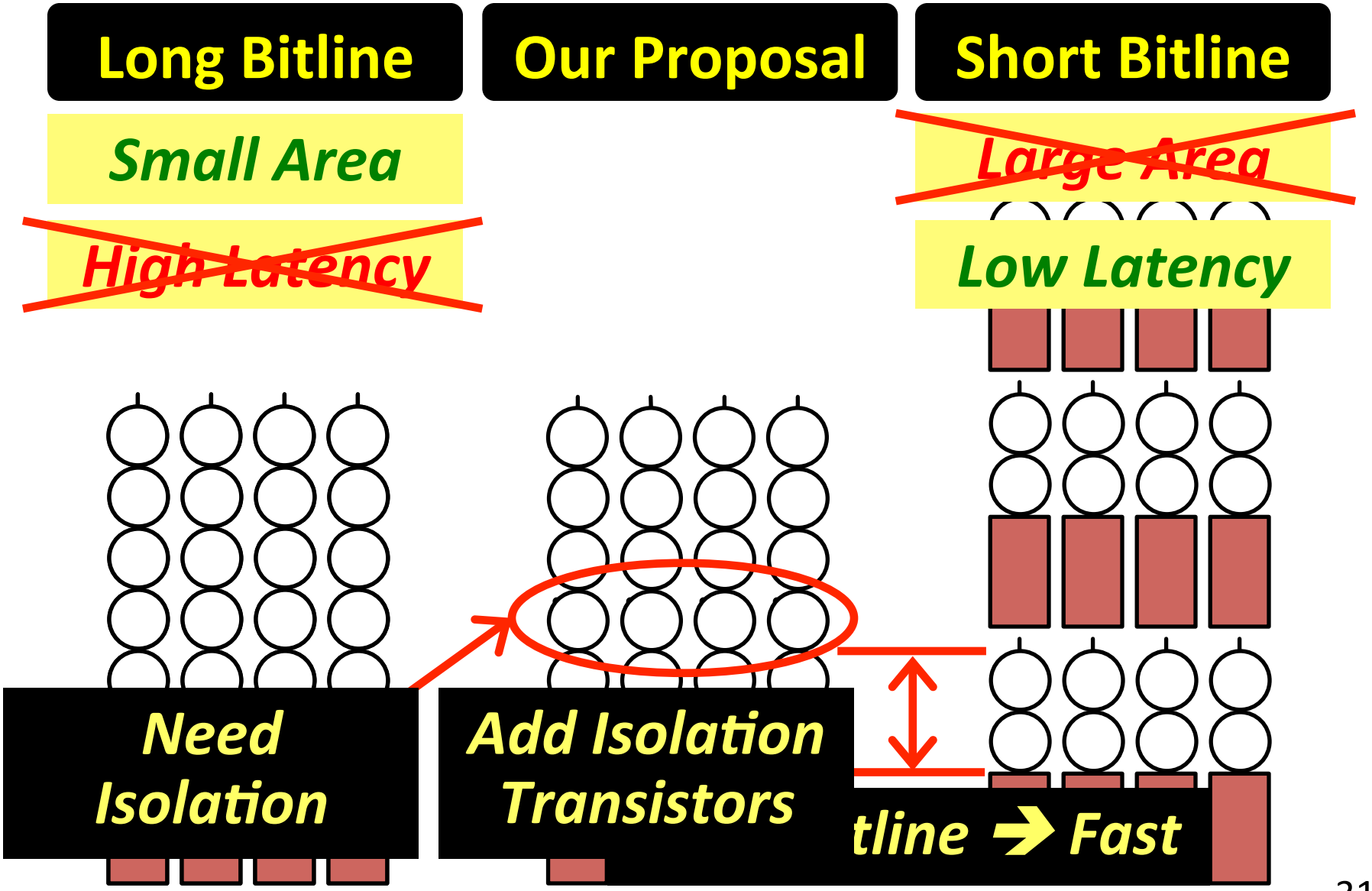
Trade-Off: Area (Die Size) vs. Latency



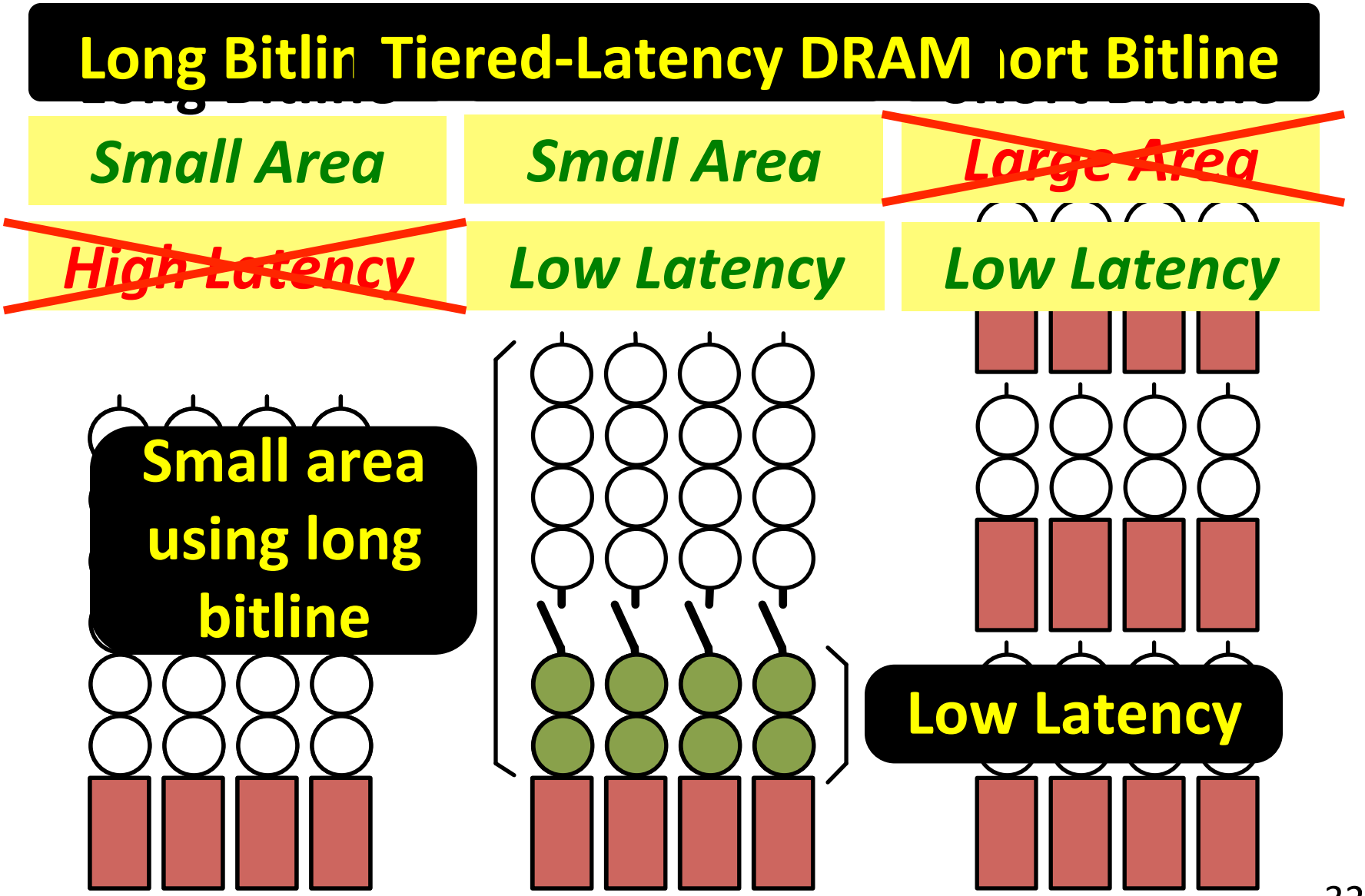
Trade-Off: Area (Die Size) vs. Latency



Approximating the Best of Both Worlds

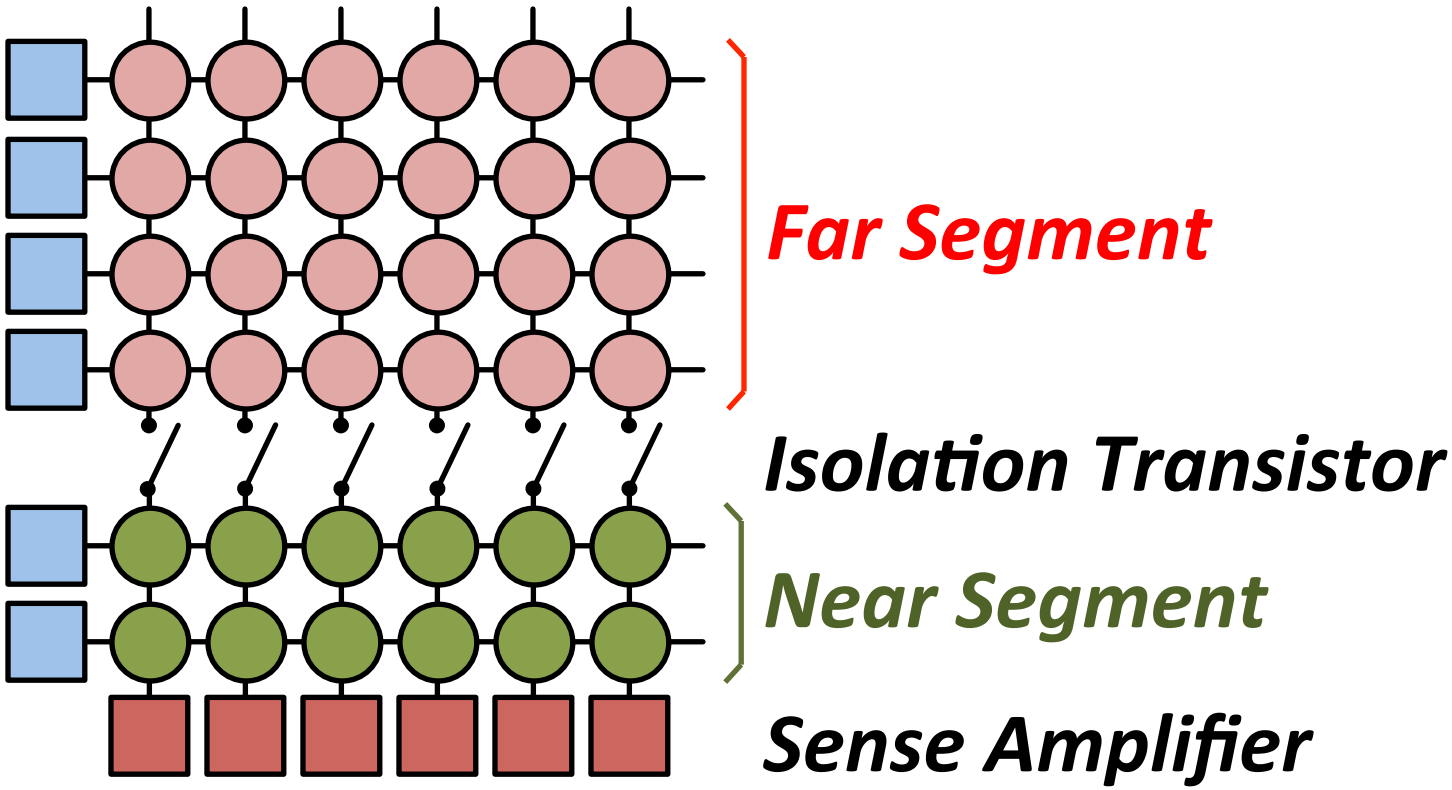


Approximating the Best of Both Worlds



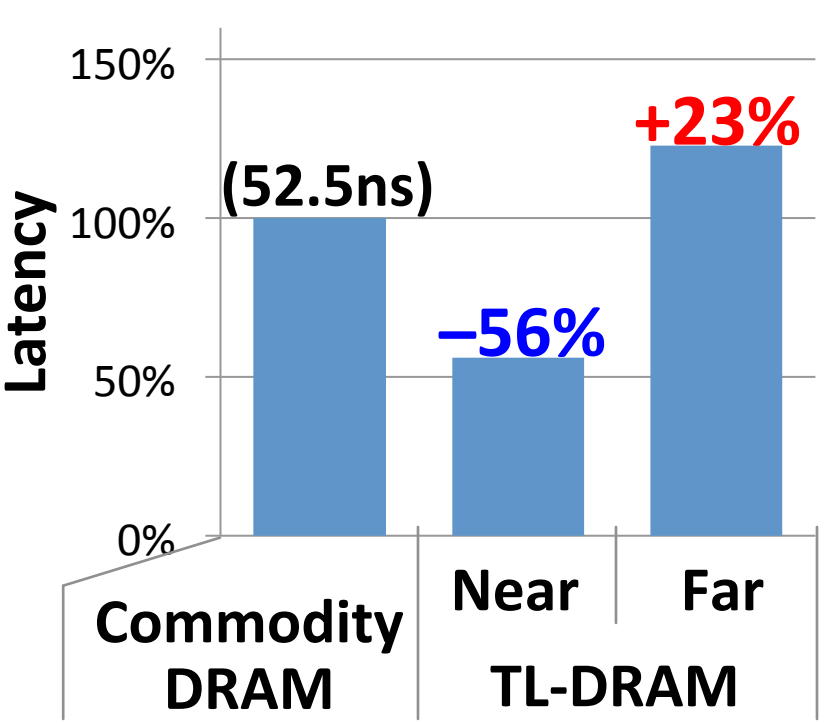
Tiered-Latency DRAM

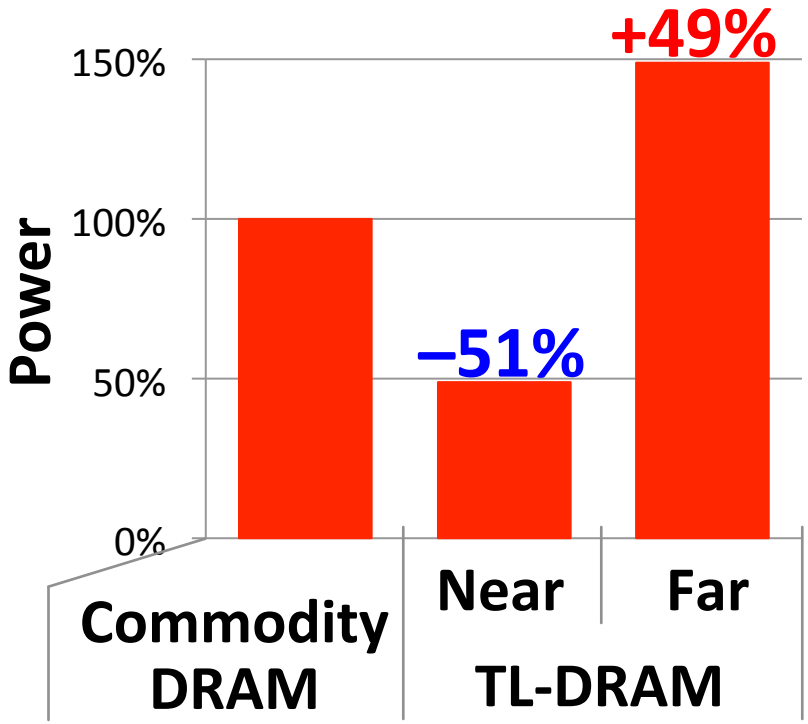
- Divide a bitline into two segments with an **isolation transistor**



Commodity DRAM vs. TL-DRAM

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

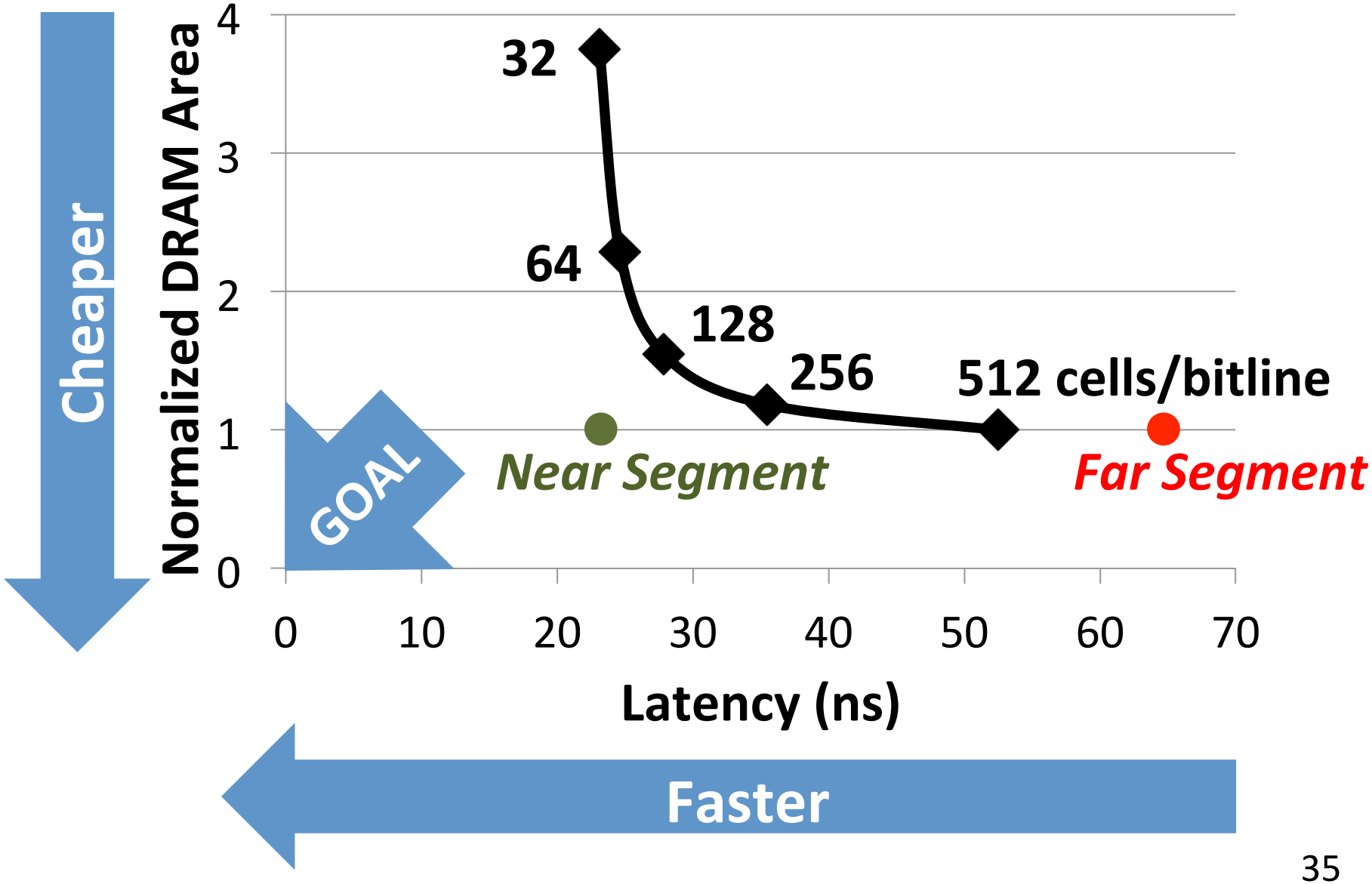




- **DRAM Area Overhead**

  ~3%: mainly due to the isolation transistors

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



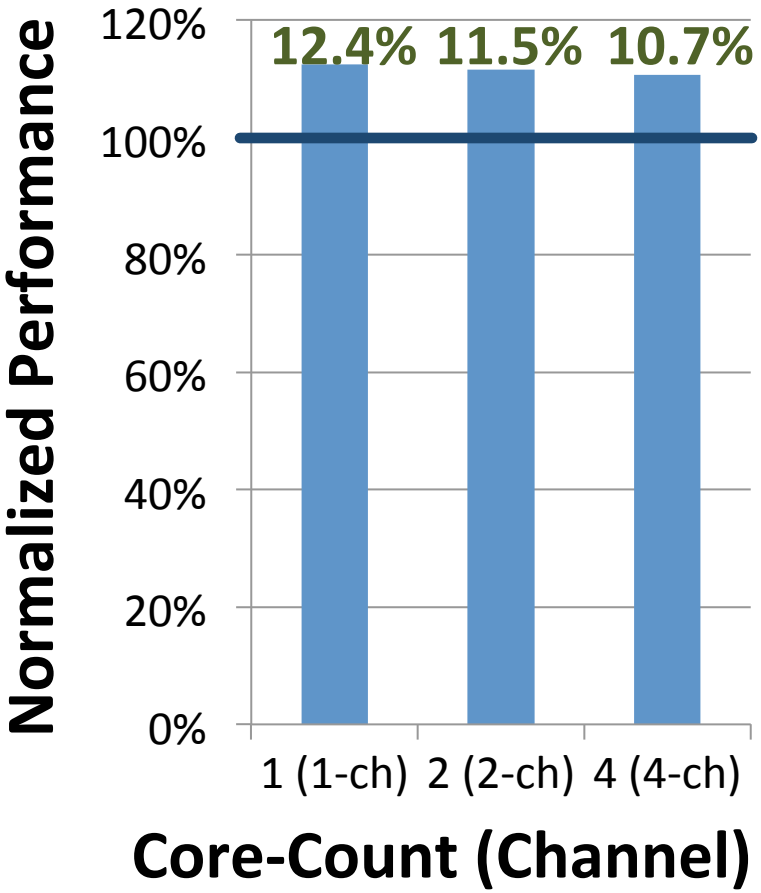
Leveraging Tiered-Latency DRAM

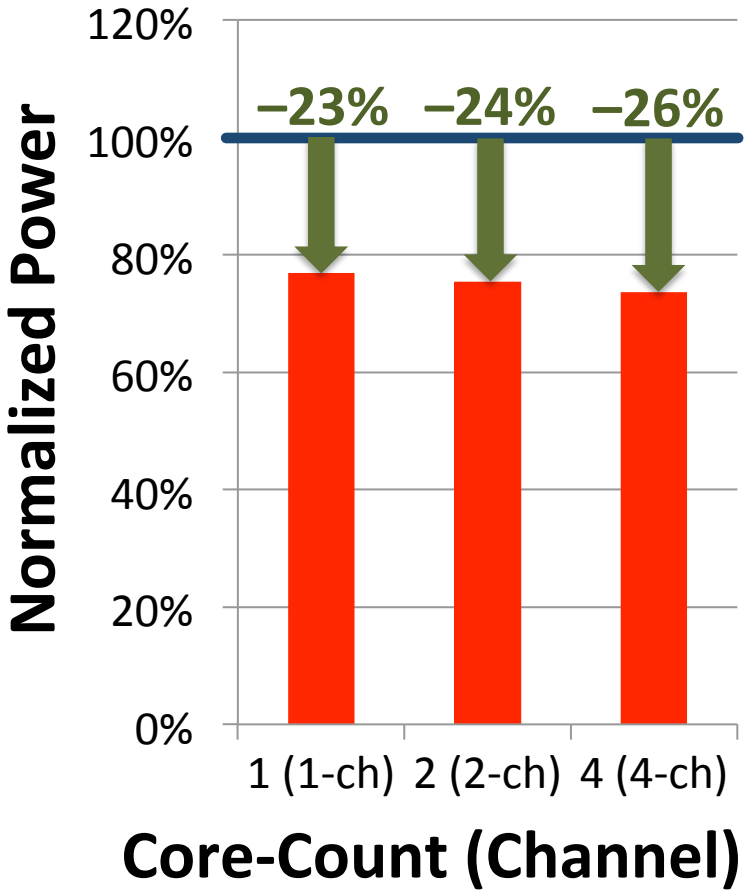
- TL-DRAM is a ***substrate*** that can be leveraged by the hardware and/or software

- 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



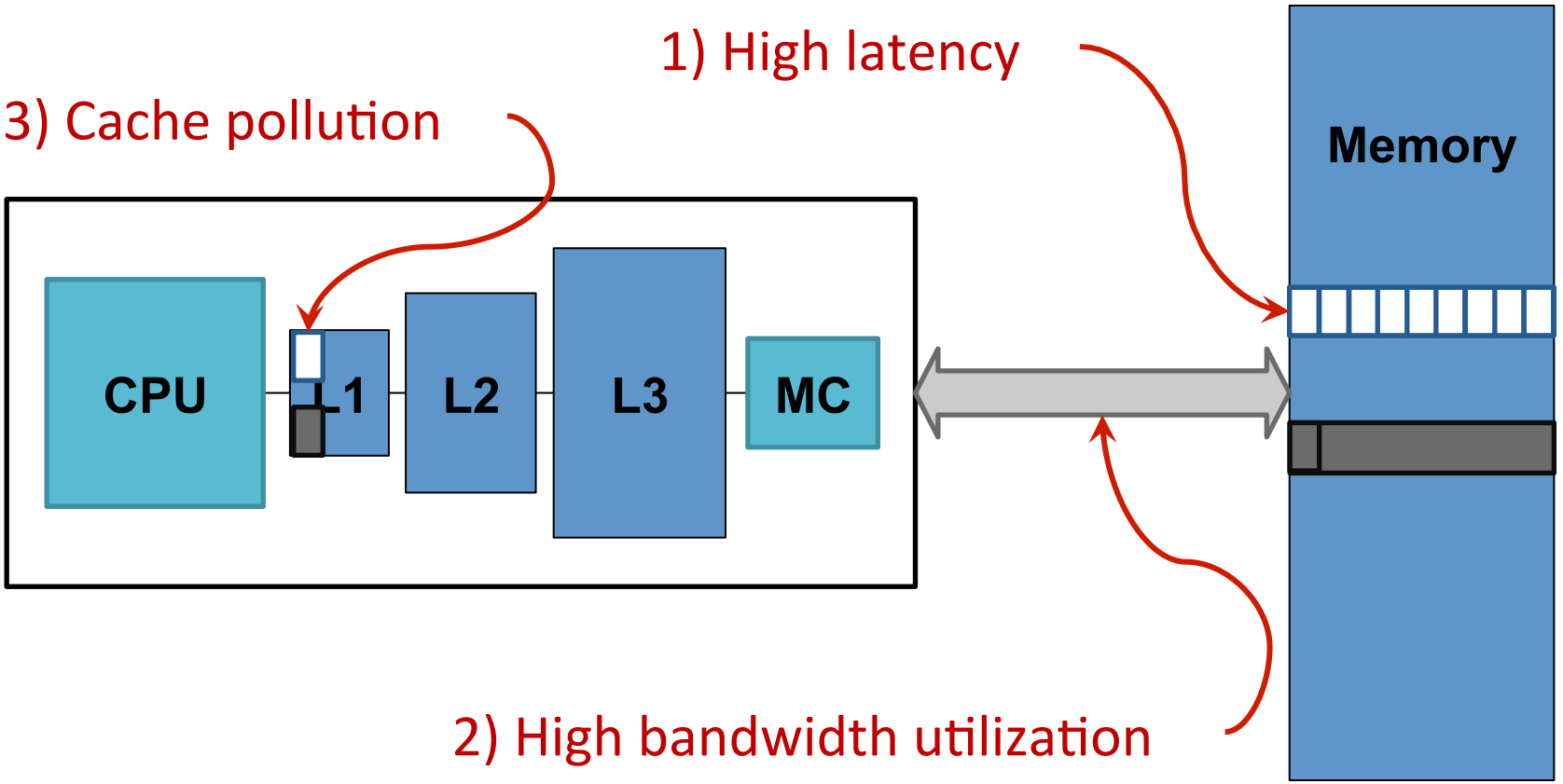


Using near segment as a cache improves performance and reduces power consumption

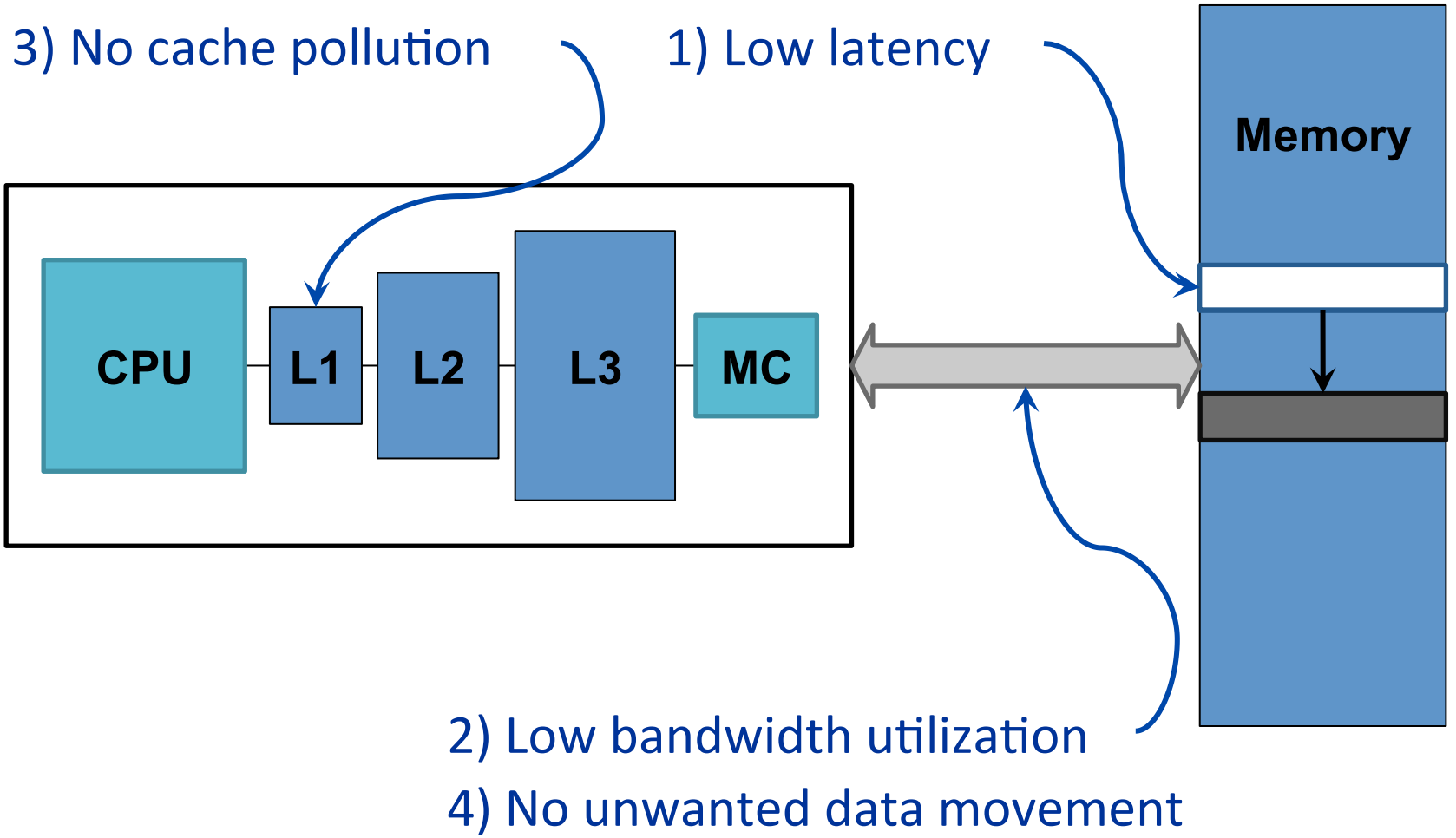
Tolerating DRAM: Example Techniques

- Retention-Aware DRAM Refresh
- Tiered-Latency DRAM
- In-Memory Page Copy and Initialization
- Subarray-Level Parallelism

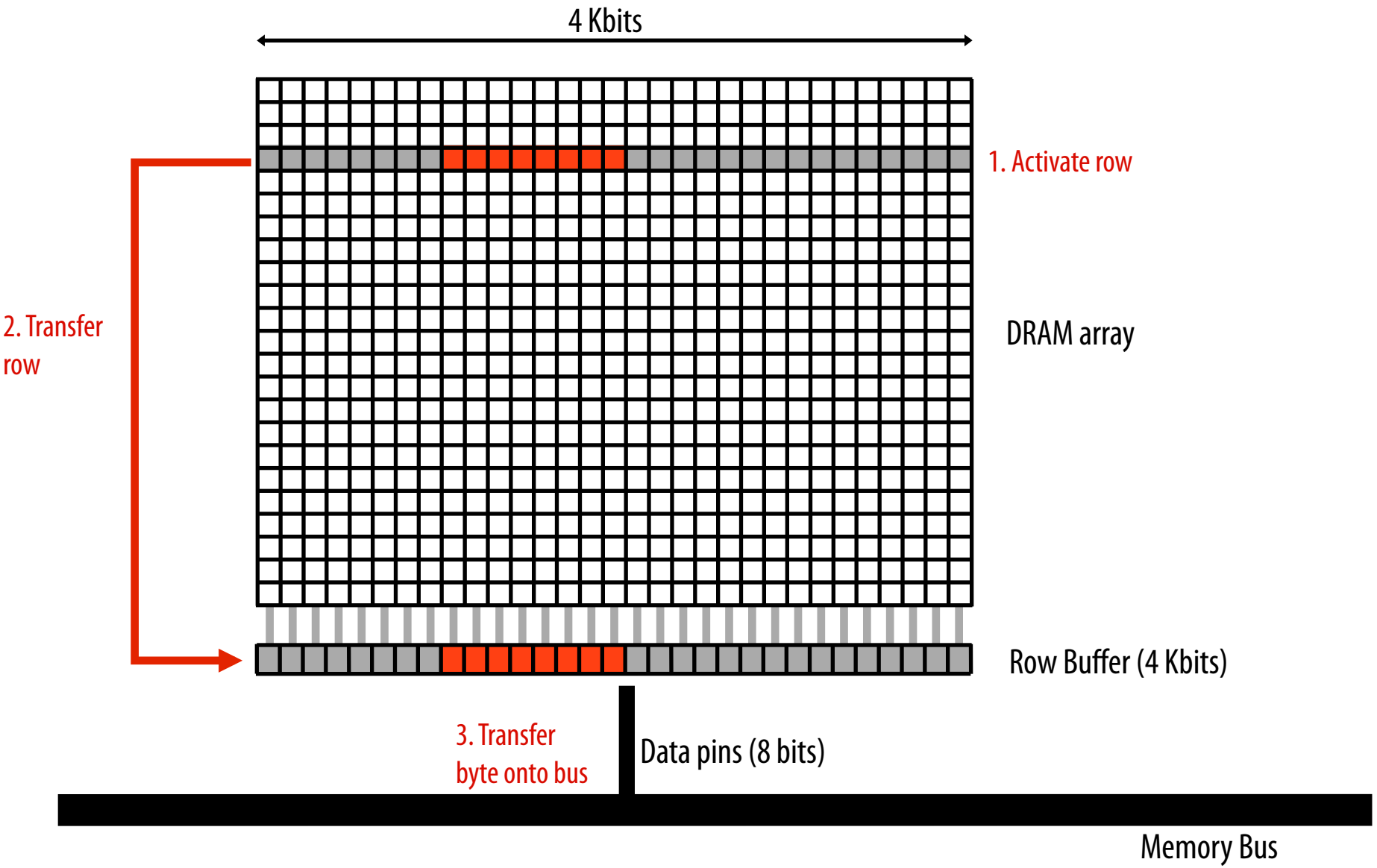
Today's Memory: Bulk Data Copy



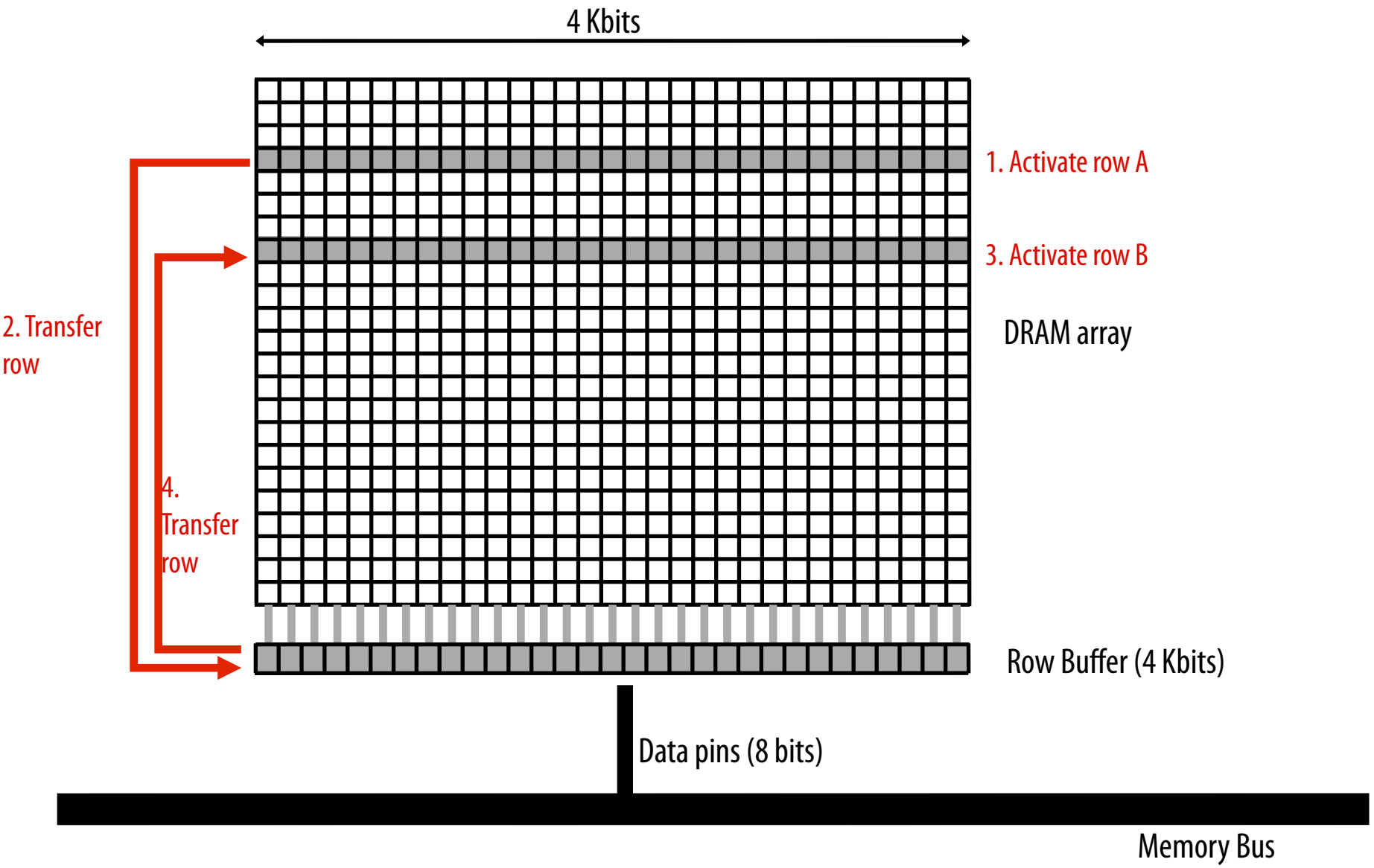
Future: RowClone (In-Memory Copy)



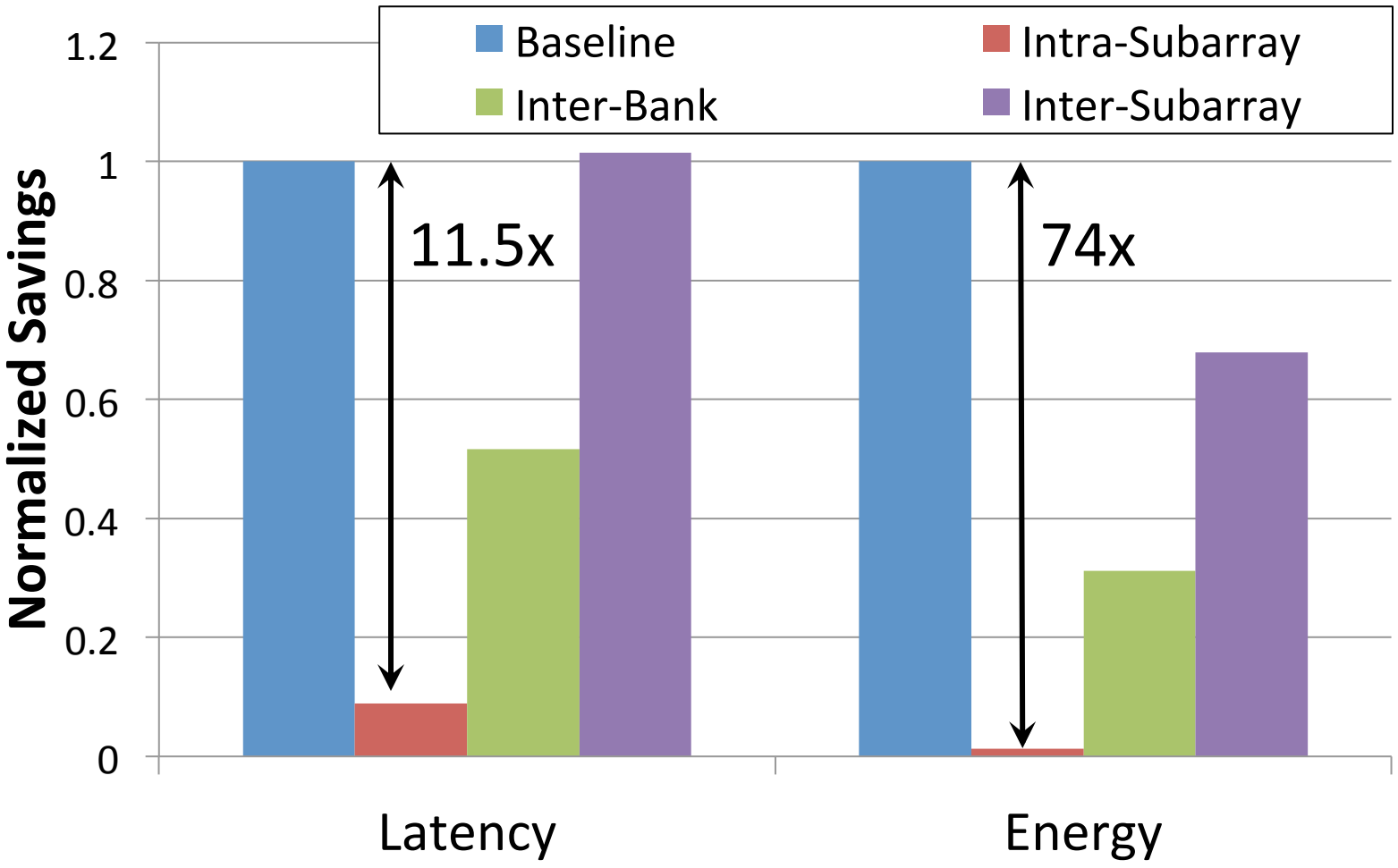
DRAM operation (load one byte)



RowClone: in-DRAM Row Copy (and Initialization)

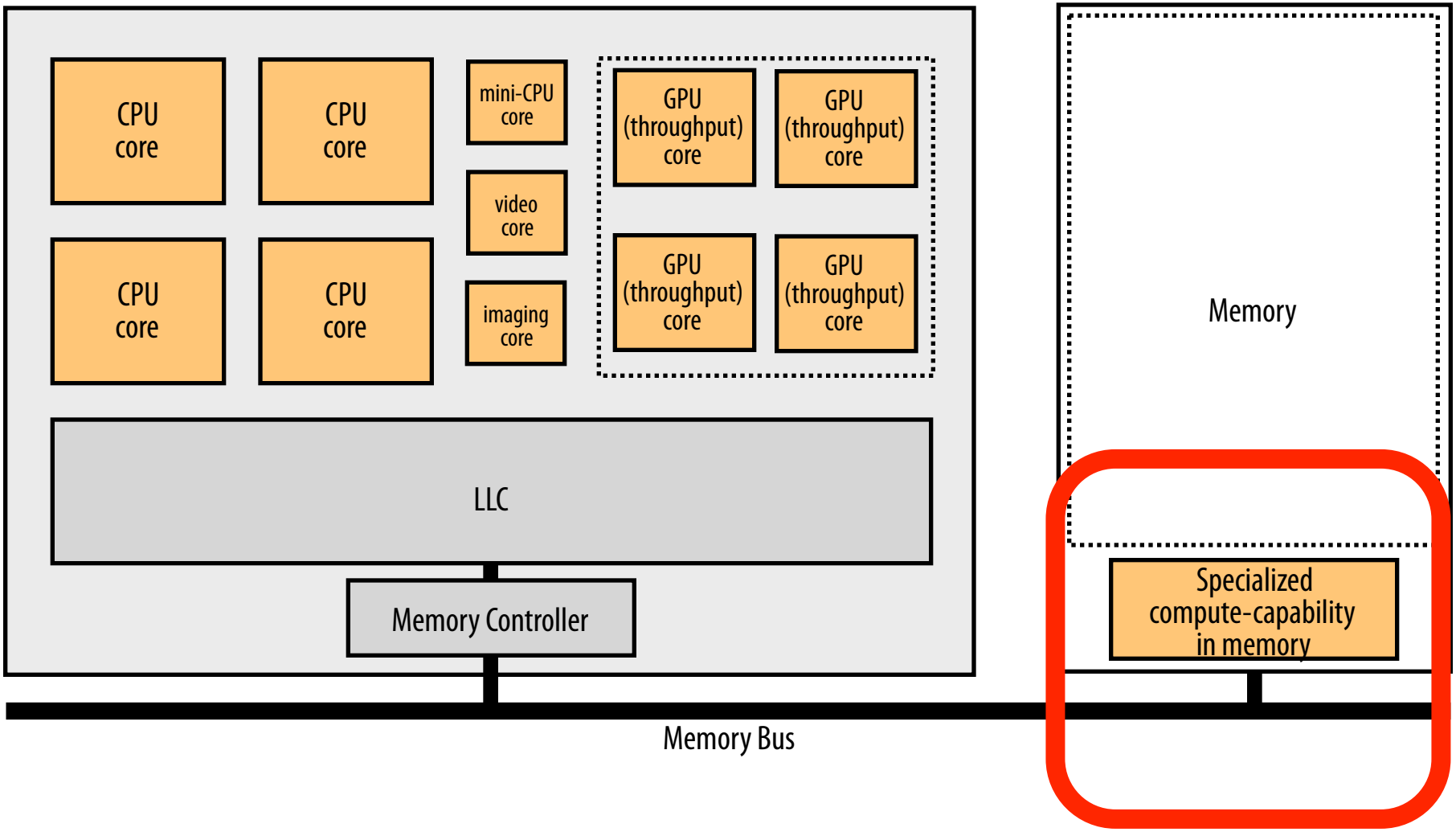


RowClone: Latency and Energy Savings



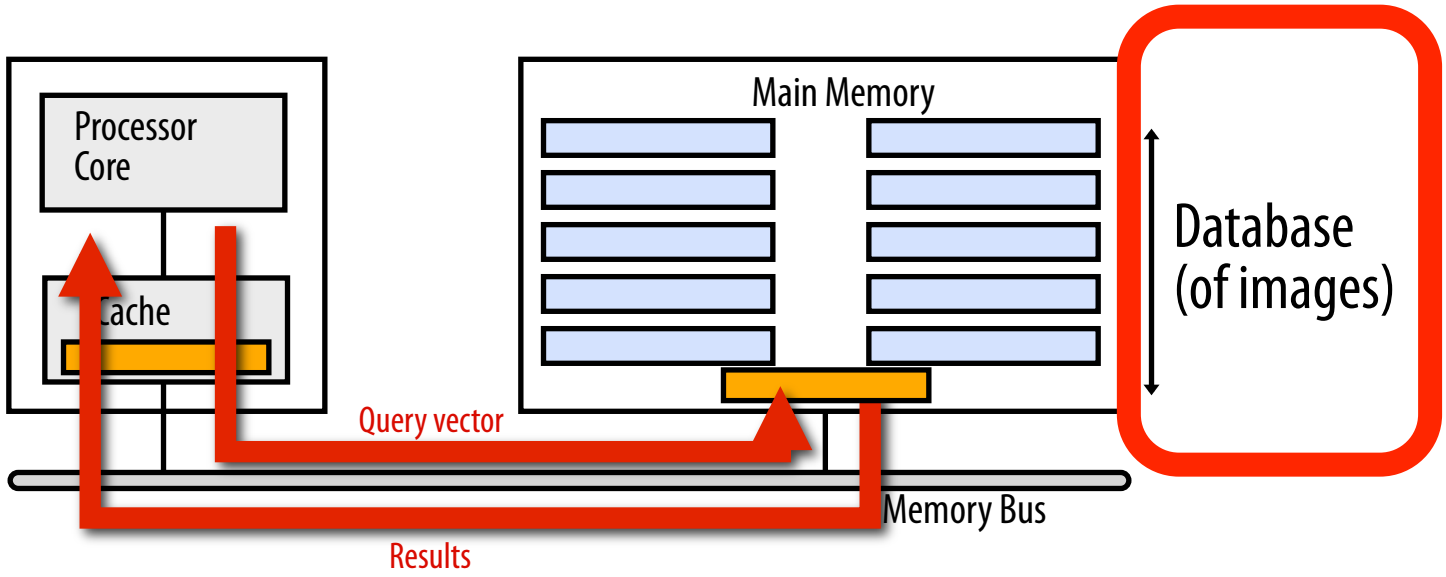
Seshadri et al., "RowClone: Fast and Efficient In-DRAM Copy and Initialization of Bulk Data," CMU Tech Report 2013.

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

Tolerating DRAM: Example Techniques

- Retention-Aware DRAM Refresh
- Tiered-Latency DRAM
- In-Memory Page Copy and Initialization
- Subarray-Level Parallelism

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

- 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

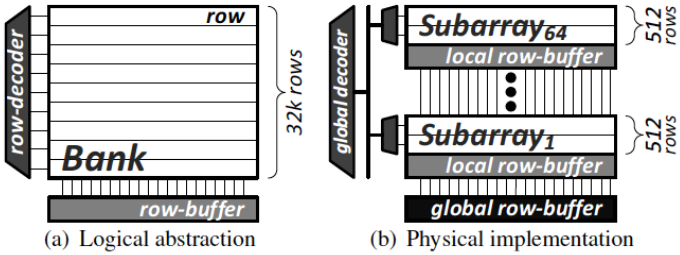
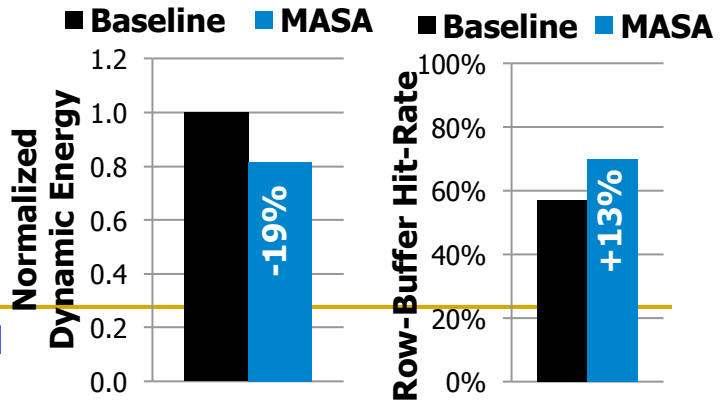


Figure 1. DRAM bank organization



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

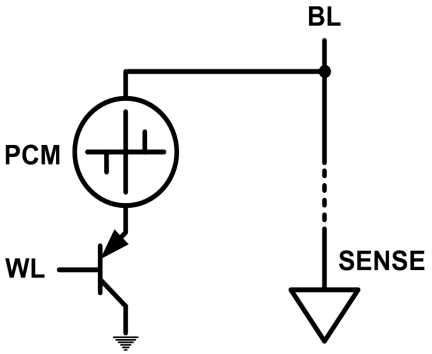
Agenda

- 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

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



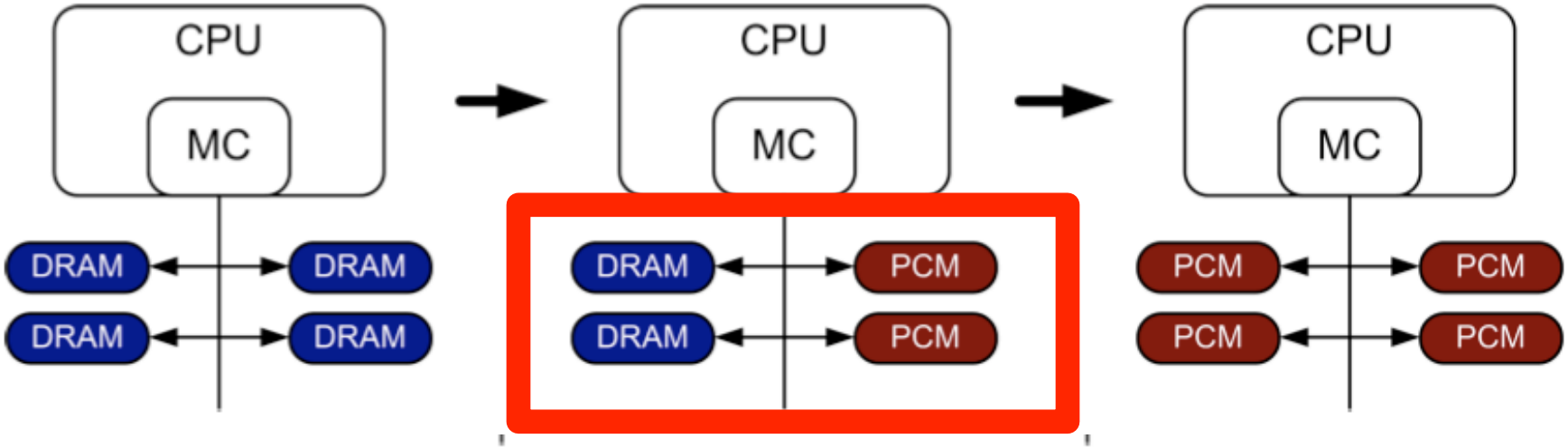
- But, emerging technologies have (many) shortcomings
  - Can they be enabled to replace/augment/surpass DRAM?

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)

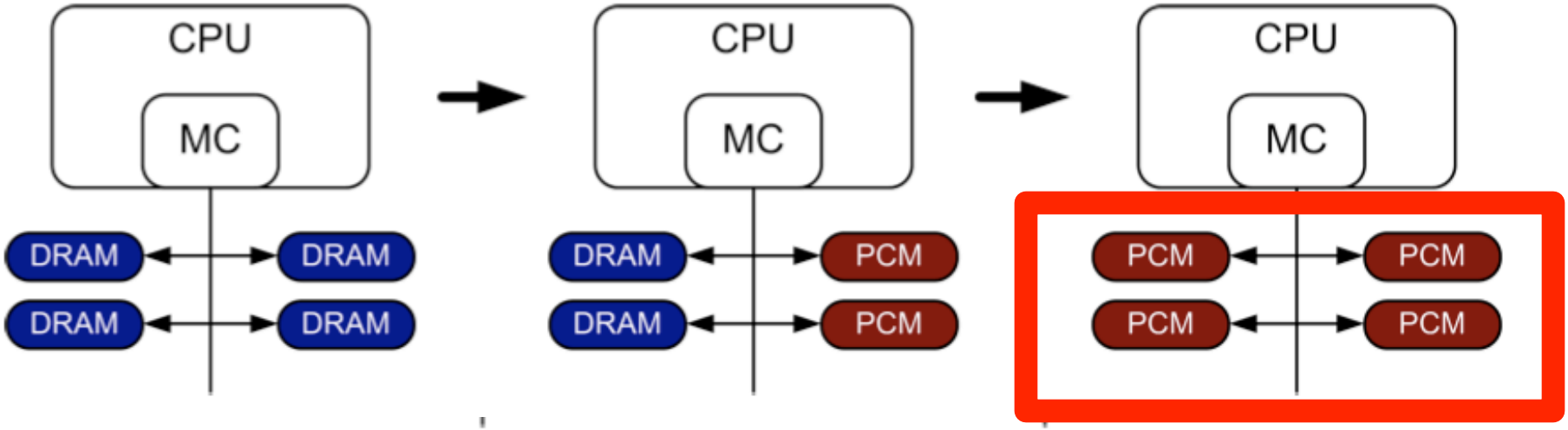
- How should PCM-based (main) memory be organized?



- 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

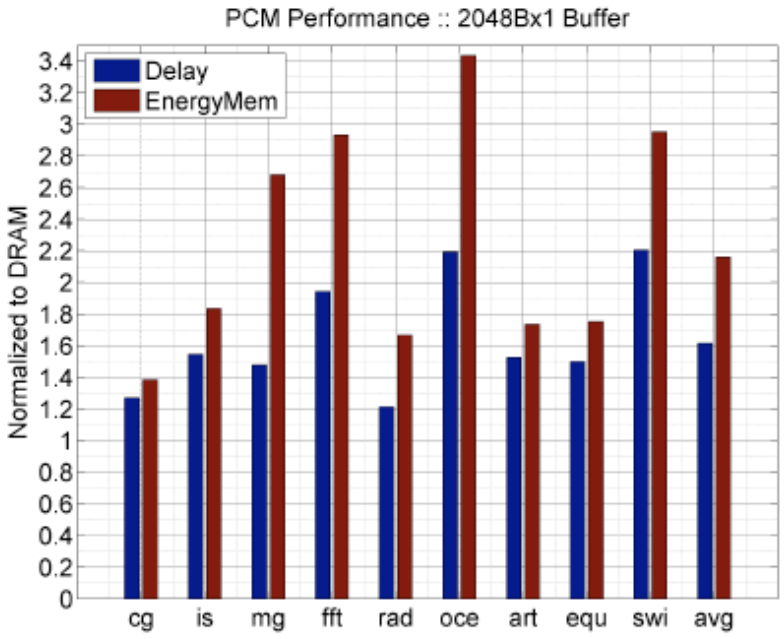
- 1E+08 writes
- 1E-08× DRAM

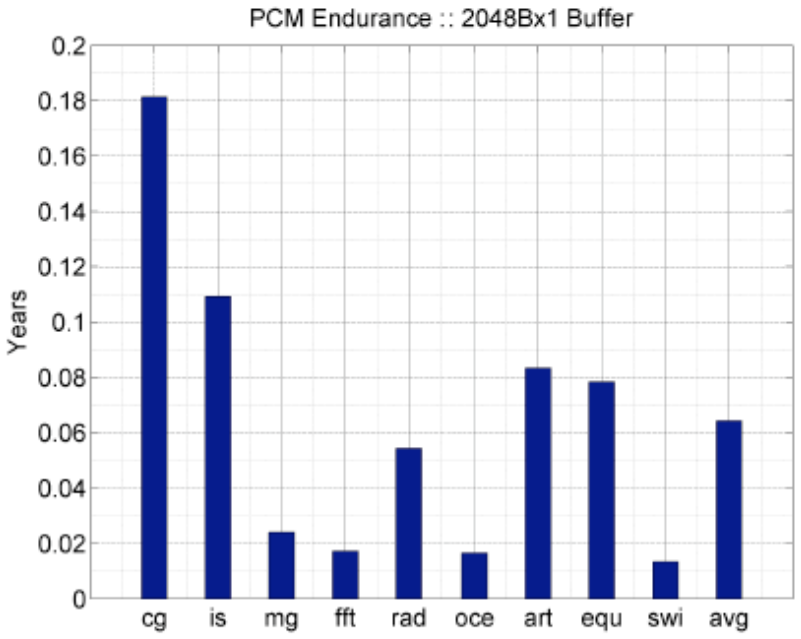
Energy

- 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



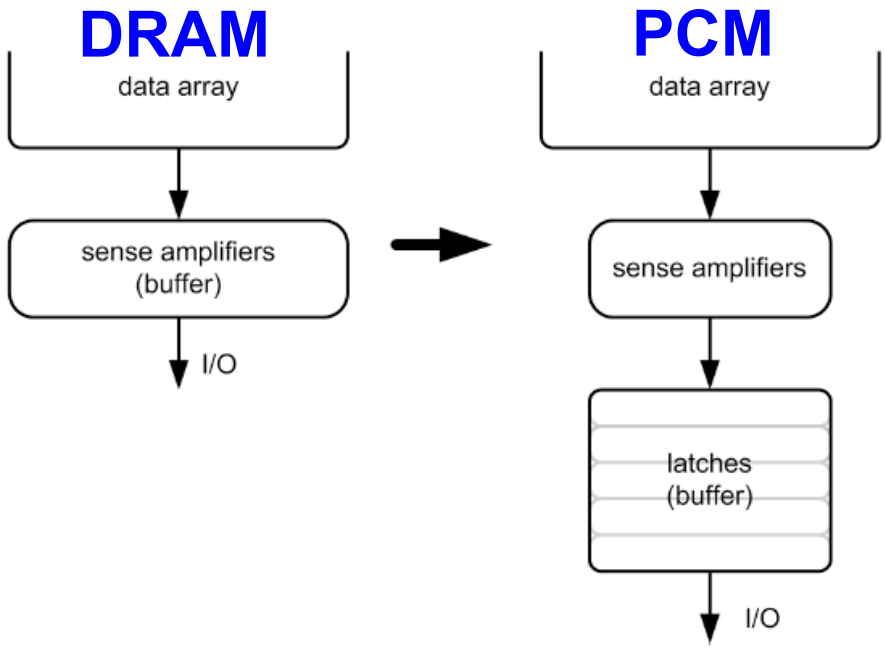


- Lee, Ipek, Mutlu, Burger, "Architecting Phase Change Memory as a Scalable DRAM Alternative," ISCA 2009.

Architecting PCM to Mitigate Shortcomings

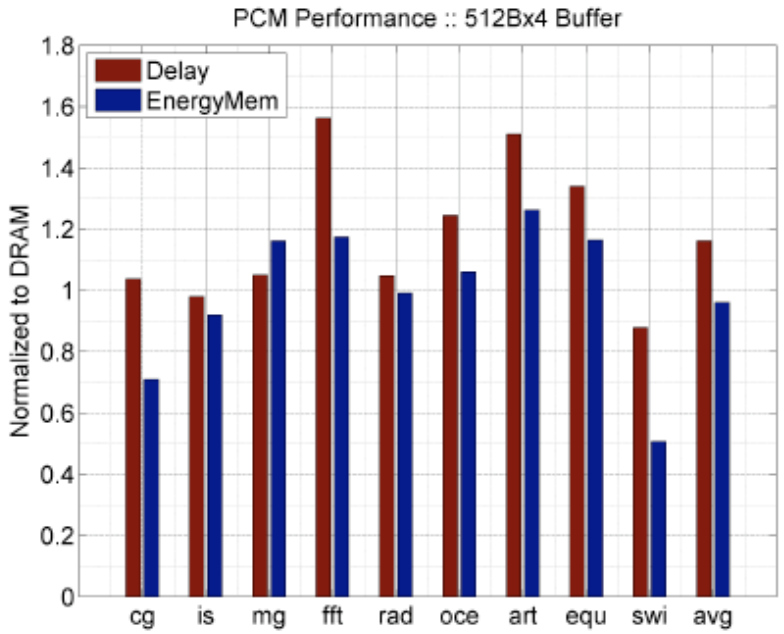
- Idea 1: Use multiple narrow row buffers in each PCM chip
  → Reduces array reads/writes → better endurance, latency, energy

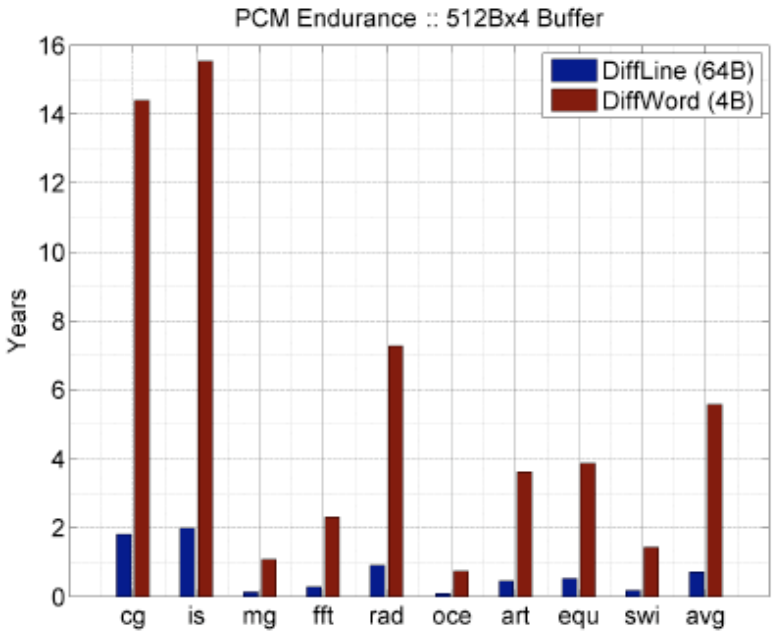
- Idea 2: Write into array at cache block or word granularity
  → Reduces unnecessary wear



Results: Architected PCM as Main Memory

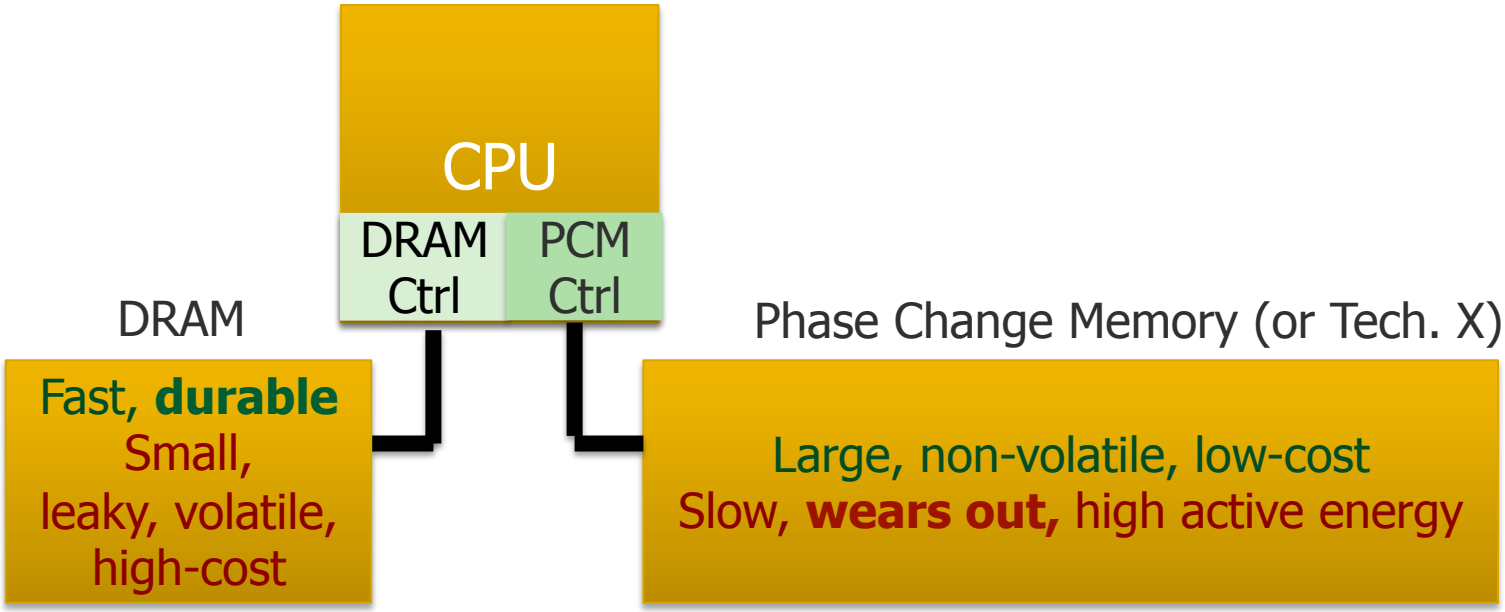
- 1.2x delay, 1.0x energy, 5.6-year average lifetime
- Scaling improves energy, endurance, density





- 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



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

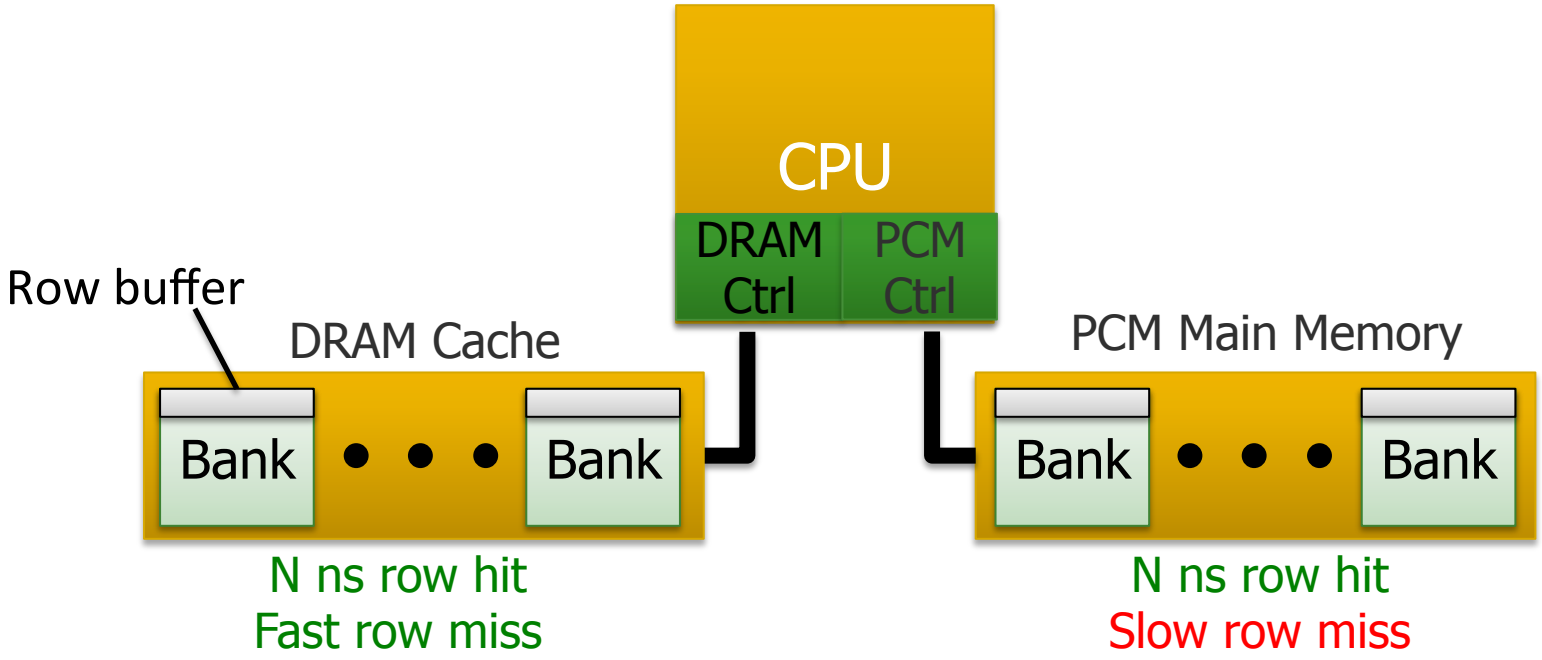
- PCM is main memory; DRAM caches memory rows/blocks
  - Benefits: Reduced latency on DRAM cache hit; write filtering
- Memory controller hardware manages the DRAM cache
  - Benefit: Eliminates system software overhead

- 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

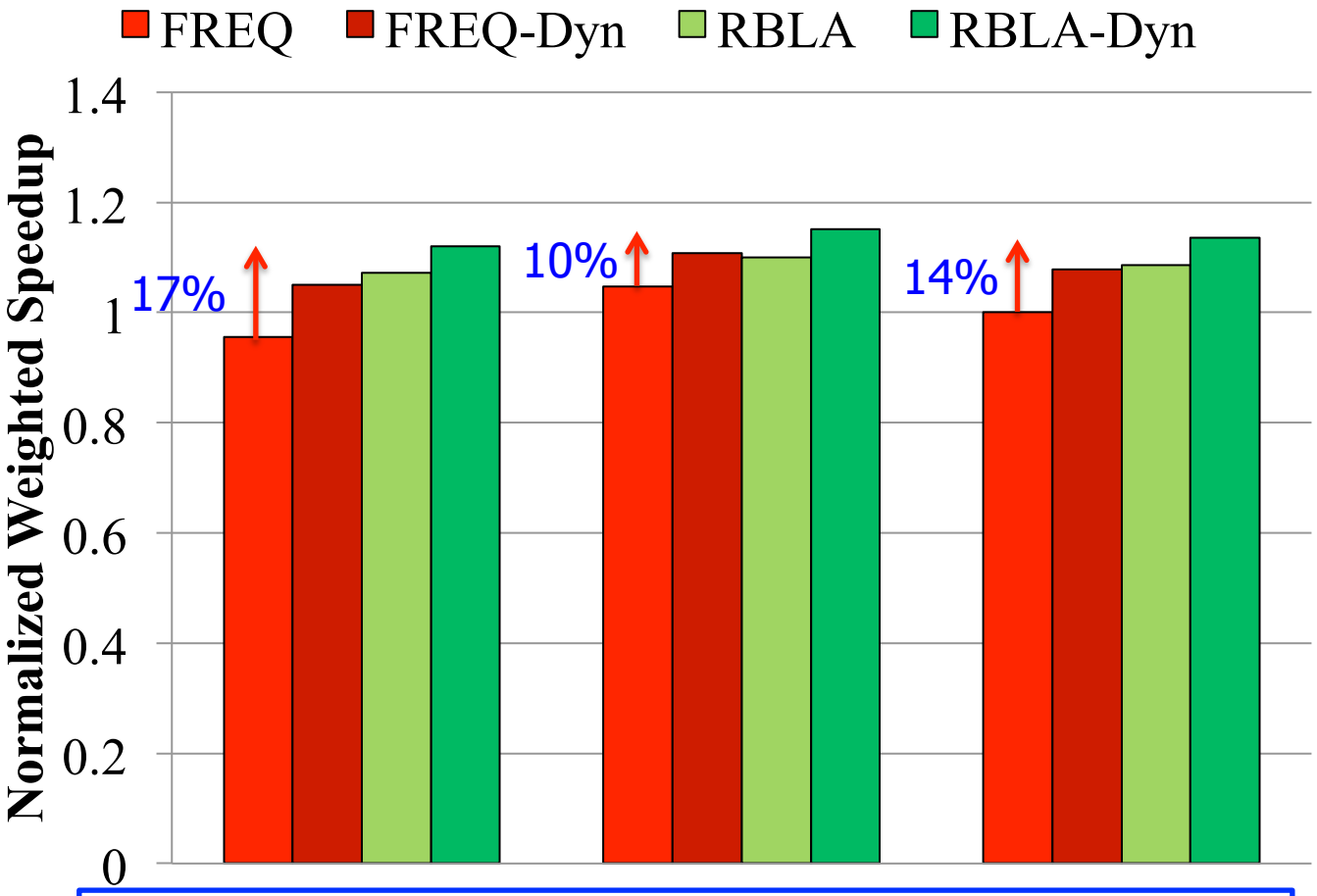
Row-Locality-Aware Data Placement

- 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

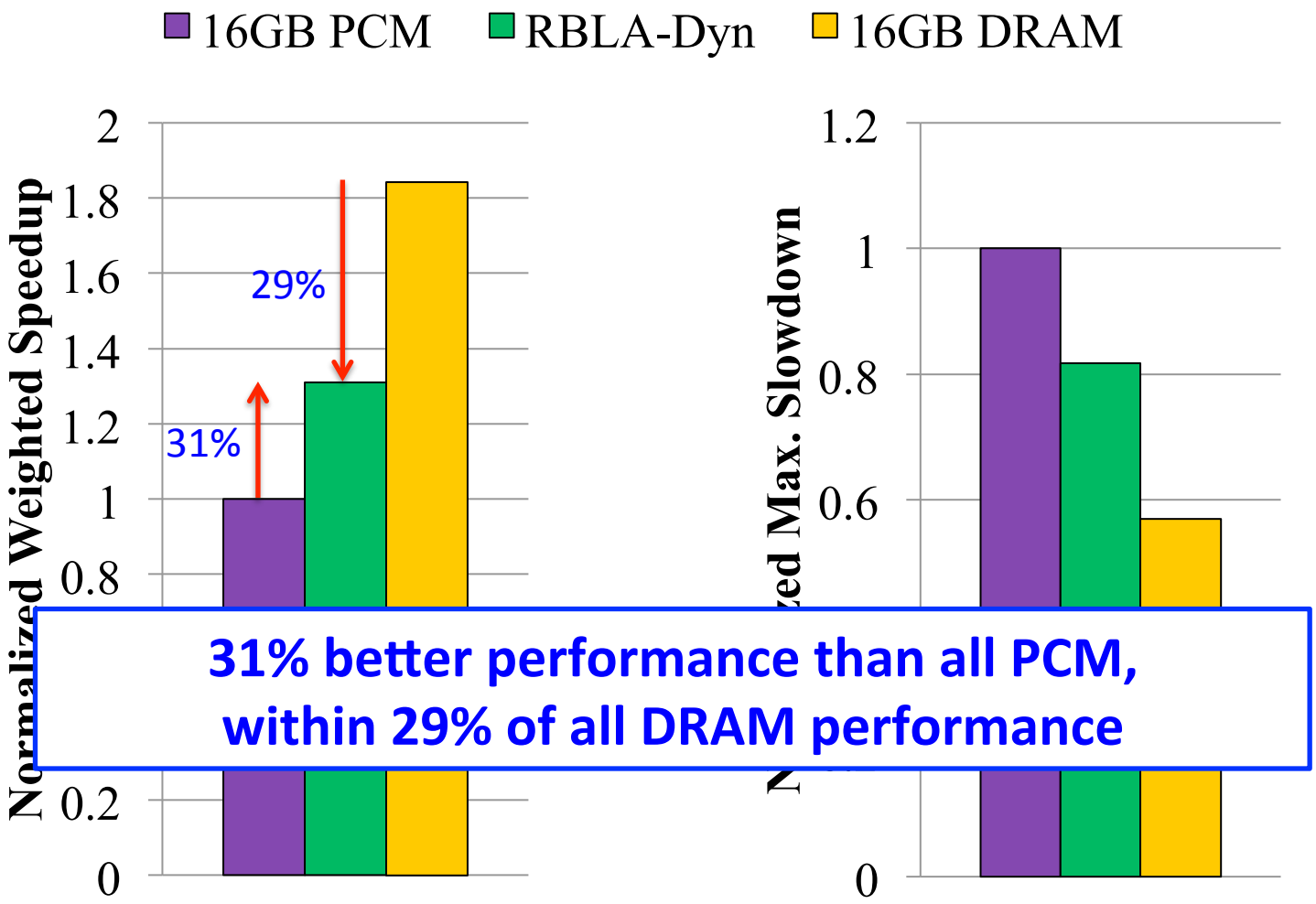
- Yoon et al., "Row Buffer Locality-Aware Data Placement in Hybrid Memories," ICCD 2012.

Row-Locality-Aware Data Placement: Results



Memory energy-efficiency and fairness also improve correspondingly

Hybrid vs. All-PCM/DRAM



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

Principles (So Far)

- Better cooperation between devices/circuits and the system
  - Expose more information about devices 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)
- Need sample chips with *main memory* interface/speeds
  - Can enable new designs and applications

Other Opportunities with Emerging Technologies

- Merging of memory and storage
  - e.g., a single interface to manage all data
- New applications
  - e.g., ultra-fast checkpoint and restore
- More robust system design
  - e.g., reducing data loss
- Logic in memory?
  - 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: Main Memory Scaling

- 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

- Solution 2: Enable emerging memory technologies
  - Replace DRAM with NVM by architecting NVM chips well
  - Hybrid memory systems with automatic data management

- Software/hardware/device cooperation essential for effective scaling of main memory

Thank you.

Memory Scaling: A Systems Architecture Perspective

Onur Mutlu
onur@cmu.edu
May 27, 2013
IMW 2013

Carnegie Mellon

SAFARI

Backup Slides

Backup Slides Agenda

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

Building Large Caches for Hybrid Memories

One Option: DRAM as a Cache for PCM

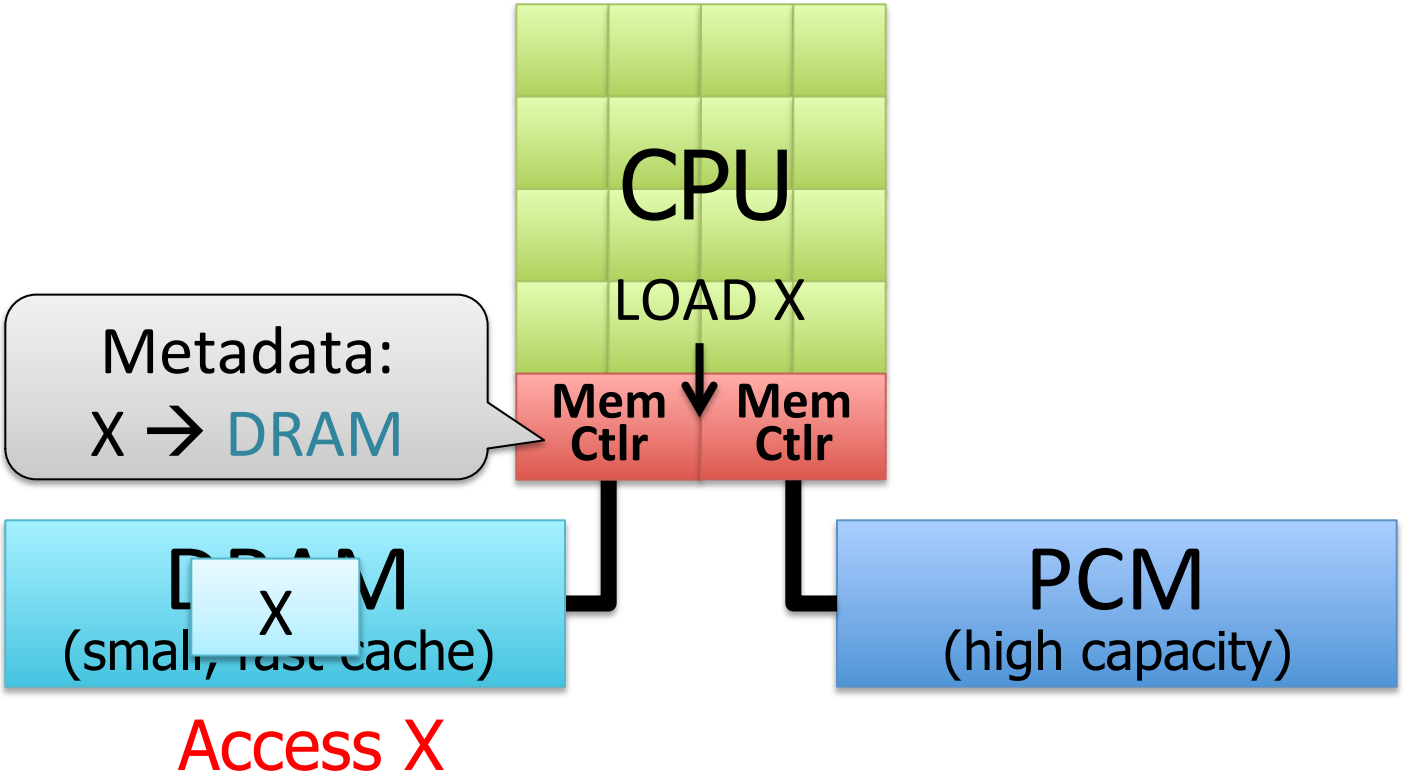
- PCM is main memory; DRAM caches memory rows/blocks
  - Benefits: Reduced latency on DRAM cache hit; write filtering
- Memory controller hardware manages the DRAM cache
  - Benefit: Eliminates system software overhead

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

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

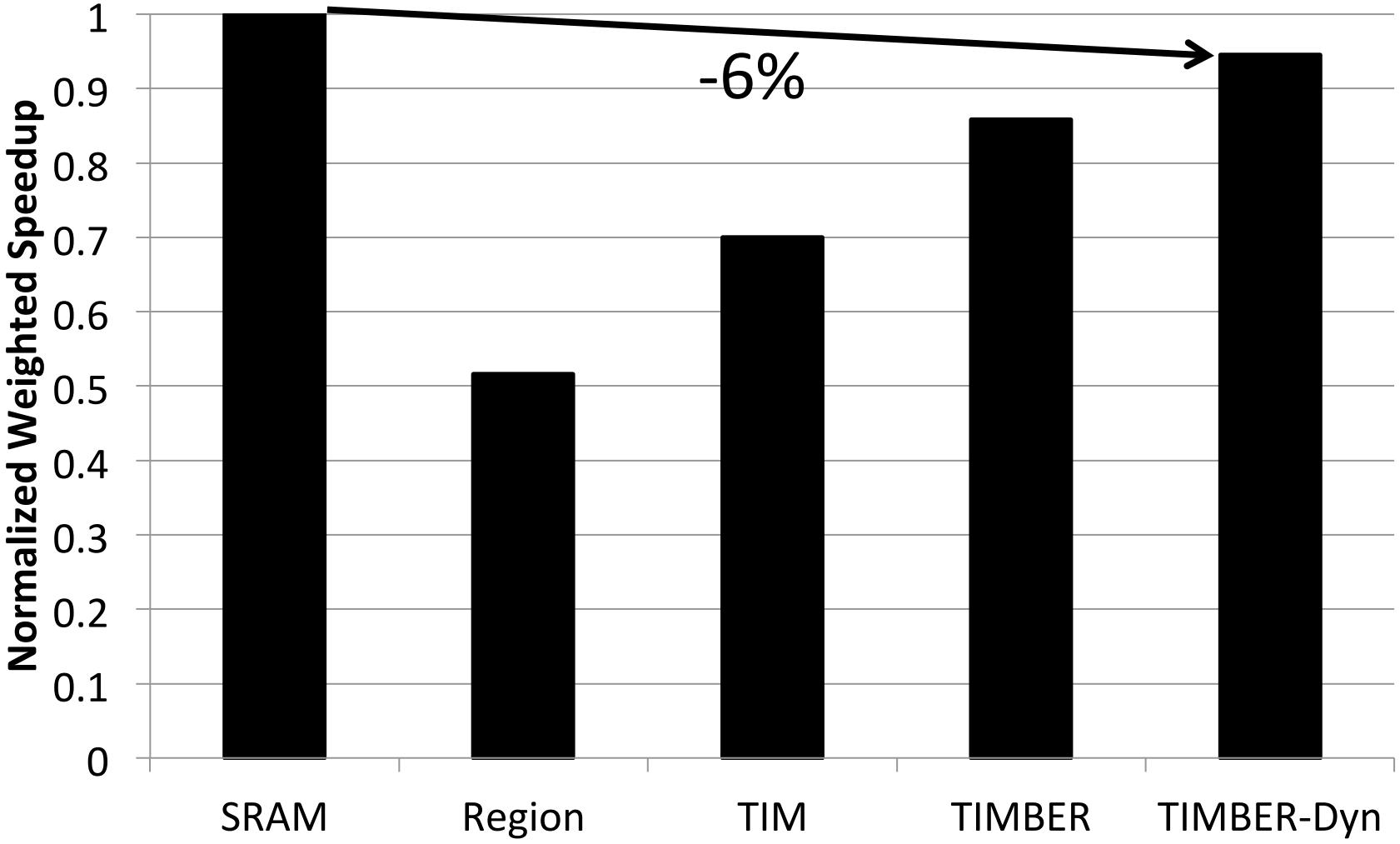
Idea 3: Dynamic Data Transfer Granularity

- 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

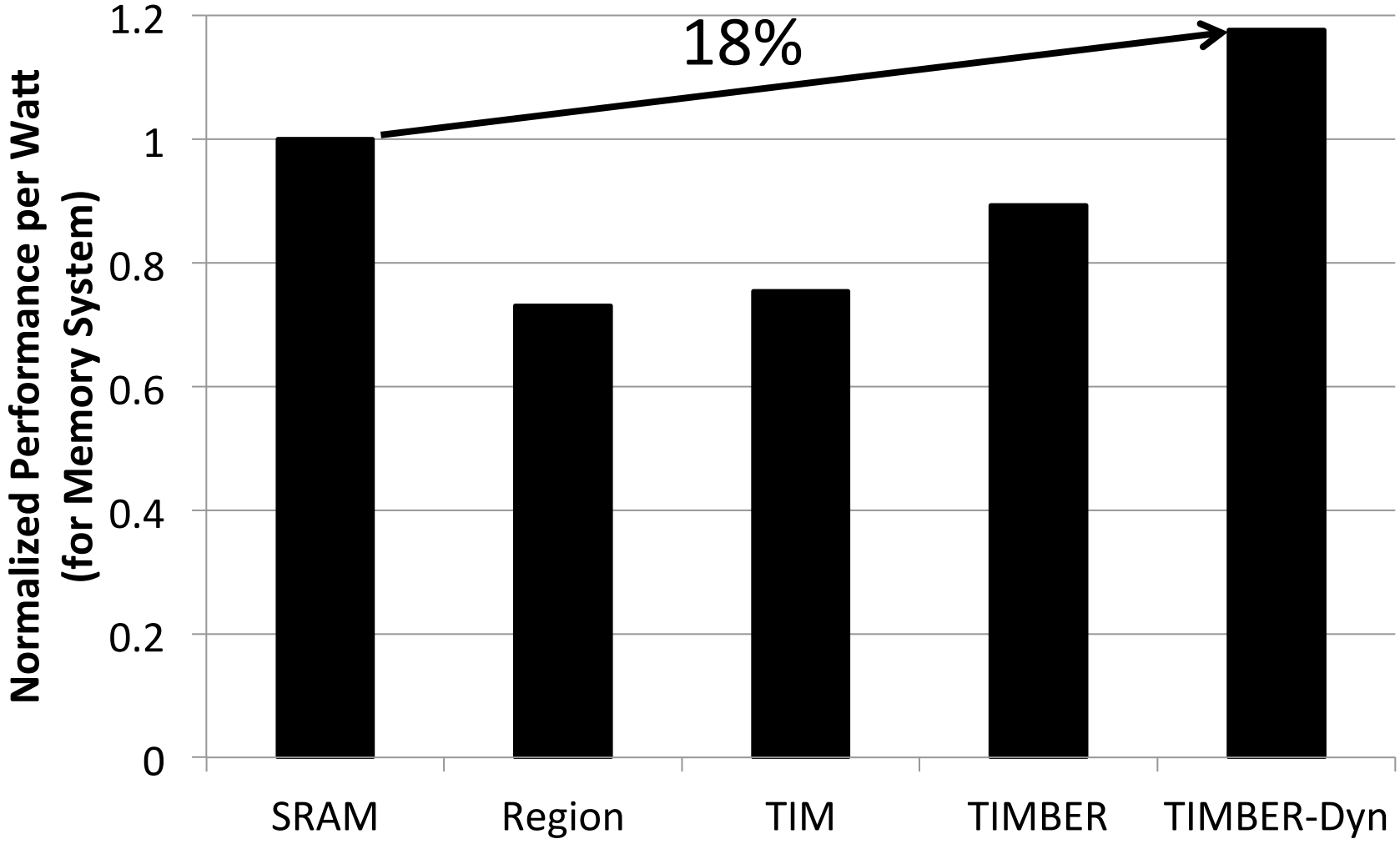
- Meza, Chang, Yoon, Mutlu, Ranganathan, "Enabling Efficient and Scalable Hybrid Memories," IEEE Comp. Arch. Letters, 2012.

TIMBER Performance



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

TIMBER Energy Efficiency



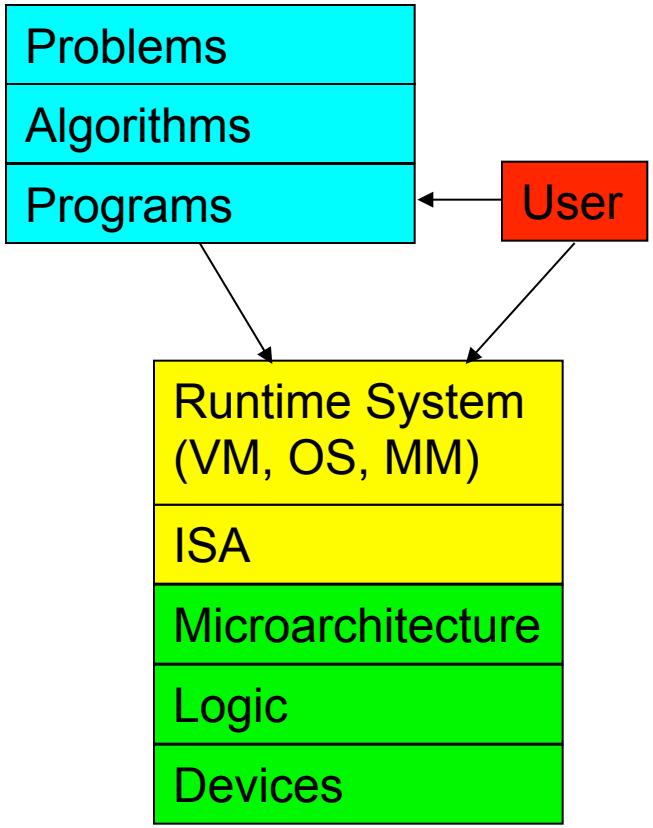
Meza, Chang, Yoon, Mutlu, Ranganathan, "Enabling Efficient and Scalable Hybrid Memories," IEEE Comp. Arch. Letters, 2012.

Hybrid Main Memory: Research Topics

- Many research topics from technology layer to algorithms layer

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

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



Security Challenges of Emerging Technologies

1. Limited endurance → Wearout attacks

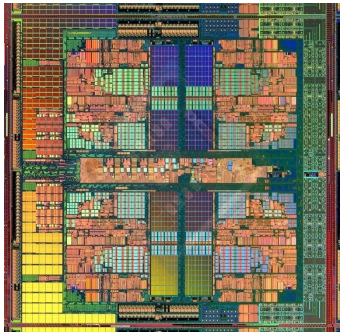
2. Non-volatility → Data persists in memory after powerdown
   → Easy retrieval of privileged or private information

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

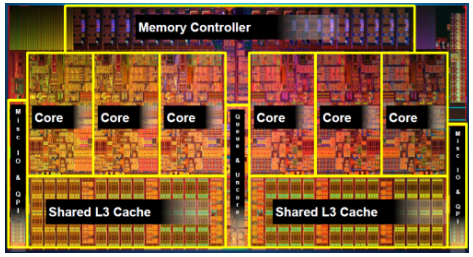
Memory QoS

Trend: Many Cores on Chip

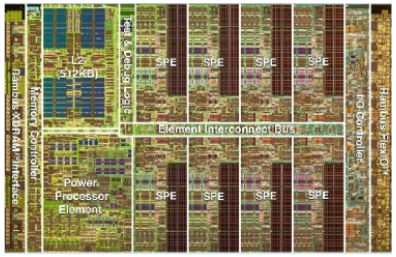
- Simpler and lower power than a single large core
- Large scale parallelism on chip



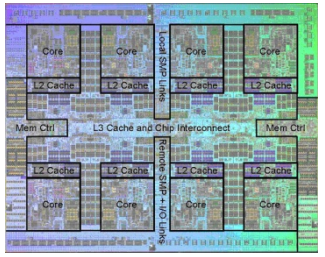
AMD Barcelona
4 cores



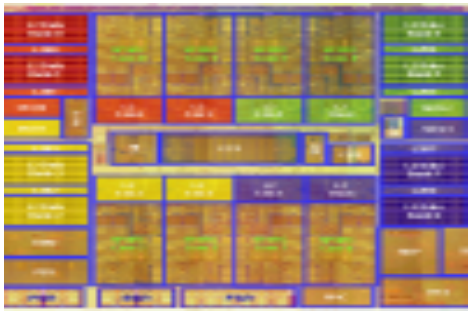
Intel Core i7
8 cores



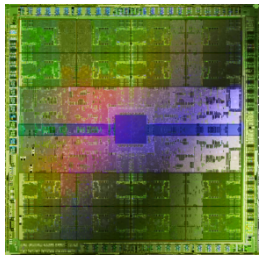
IBM Cell BE
8+1 cores



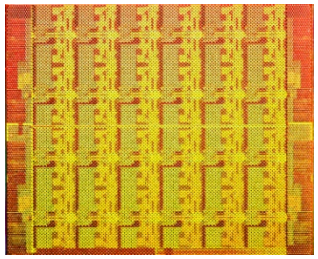
IBM POWER7
8 cores



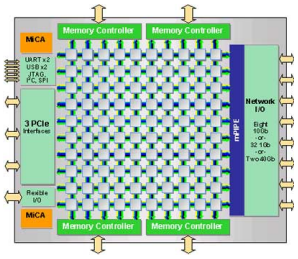
Sun Niagara II
8 cores



Nvidia Fermi
448 “cores”



Intel SCC
48 cores, networked

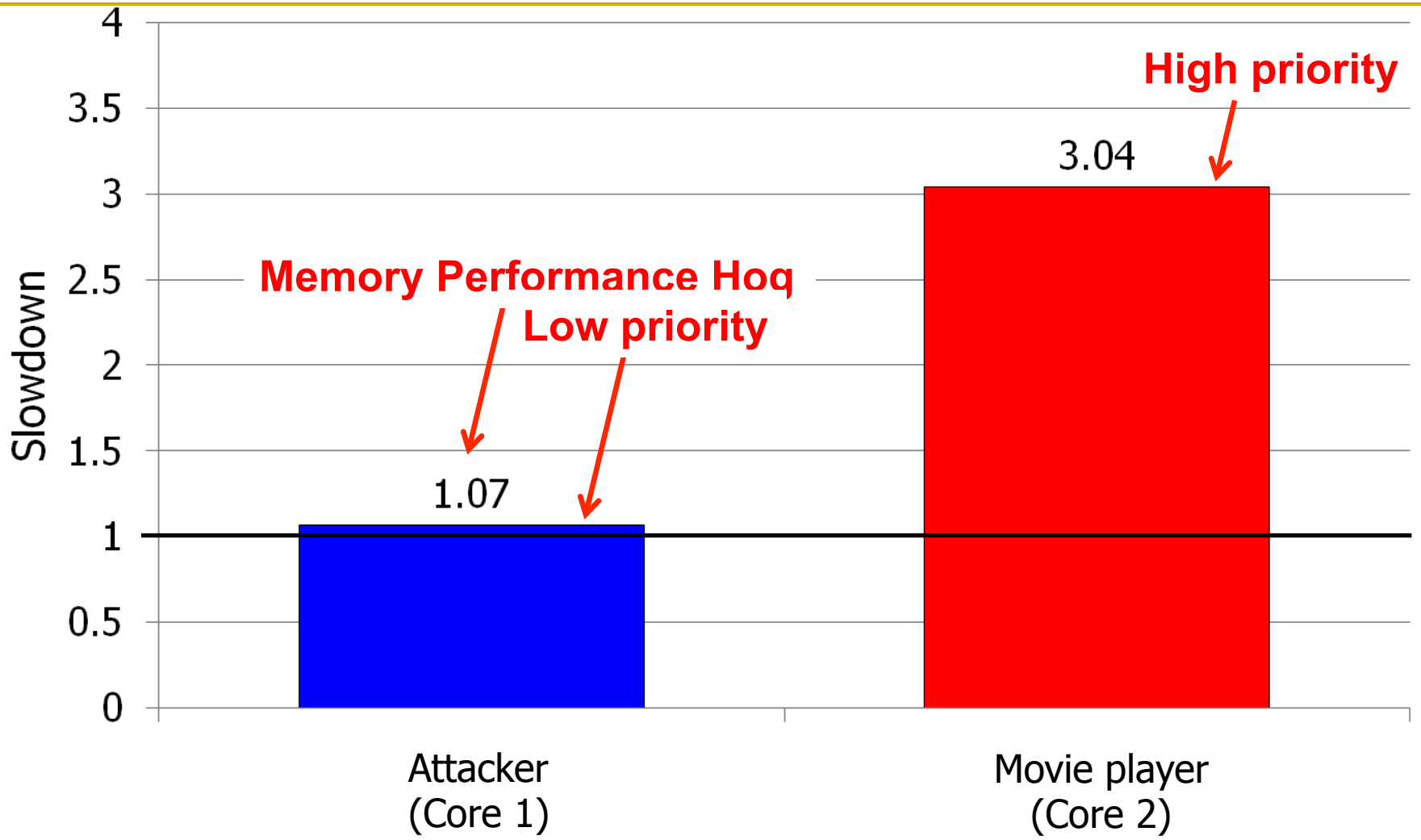


Tilera TILE Gx
100 cores, networked

Many Cores on Chip

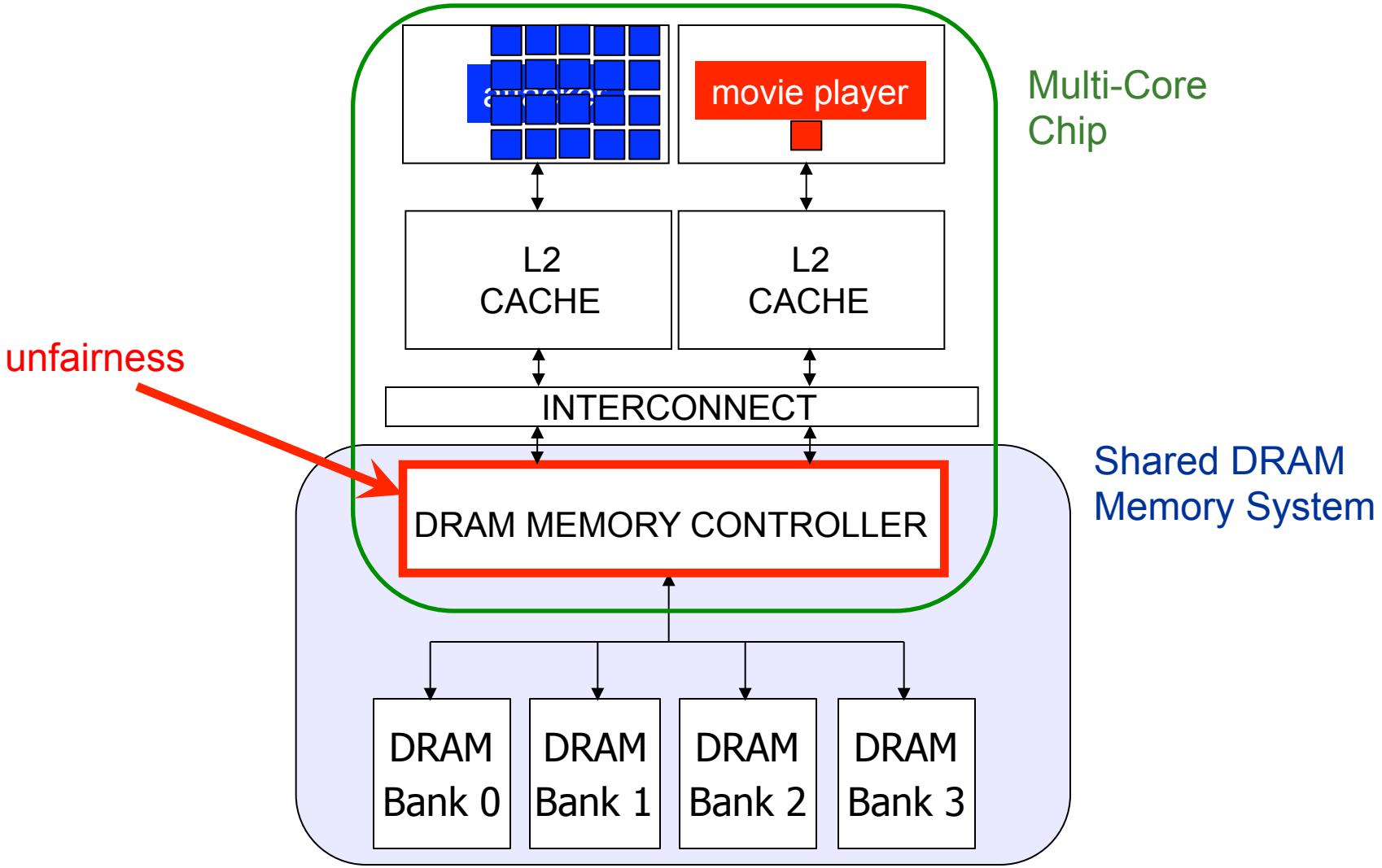
- What we want:
  - N times the system performance with N times the cores
- What do we get today?

(Un)expected Slowdowns



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

Why? Uncontrolled Memory Interference



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

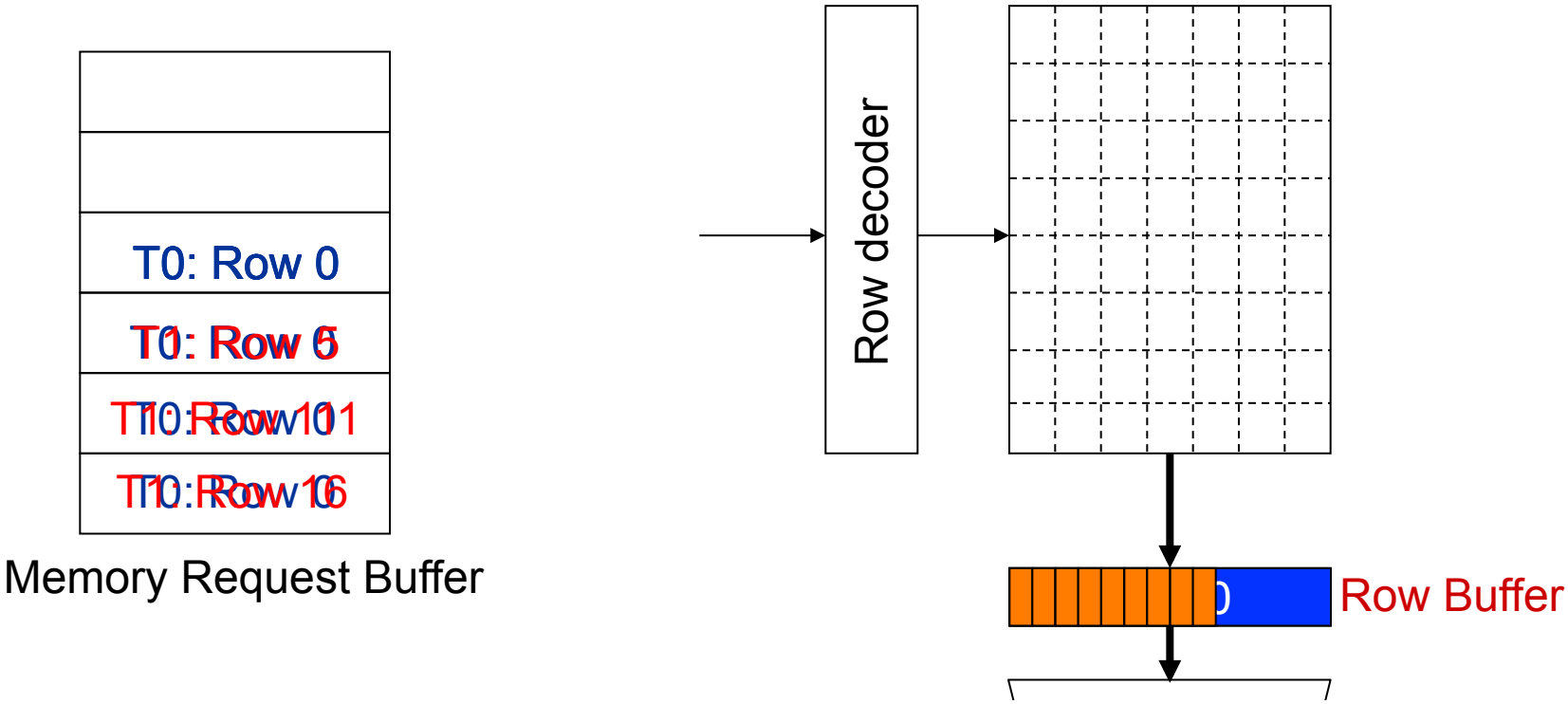
- Sequential memory access
- Very high row buffer locality (96% hit rate)
- Memory intensive

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?

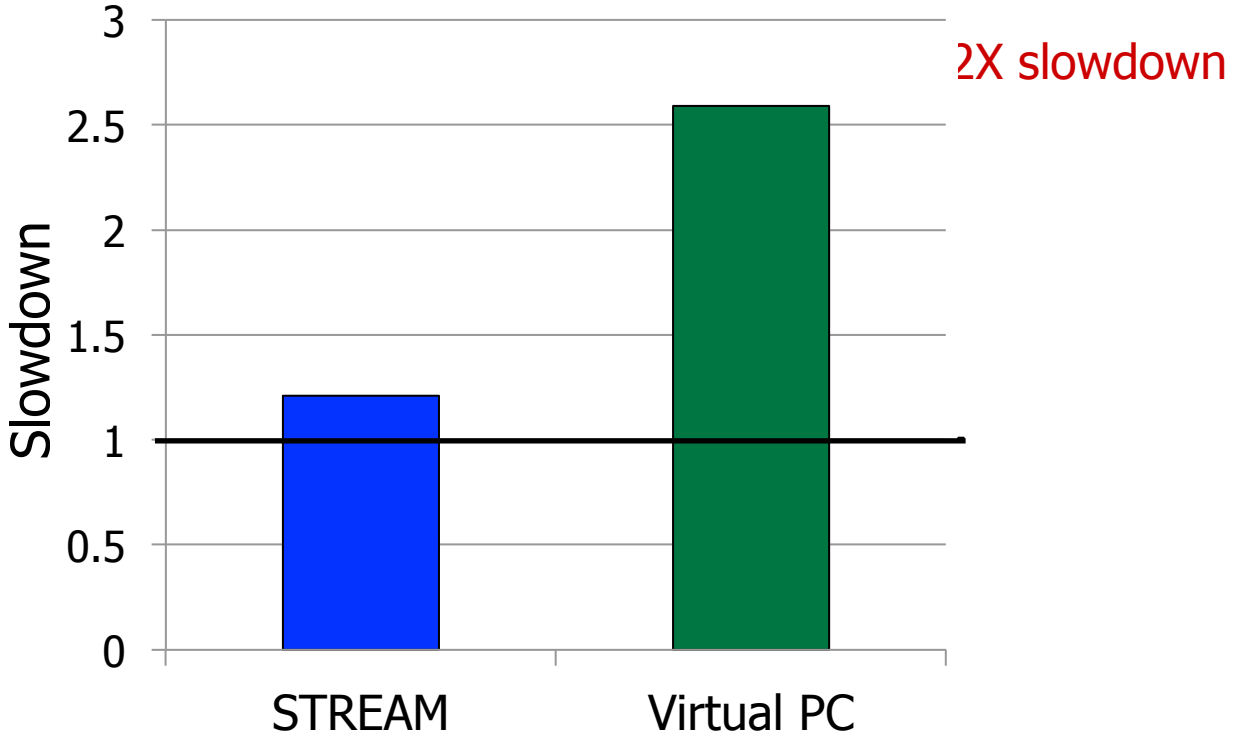


Row size: 8KB, cache block size: 64B

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

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

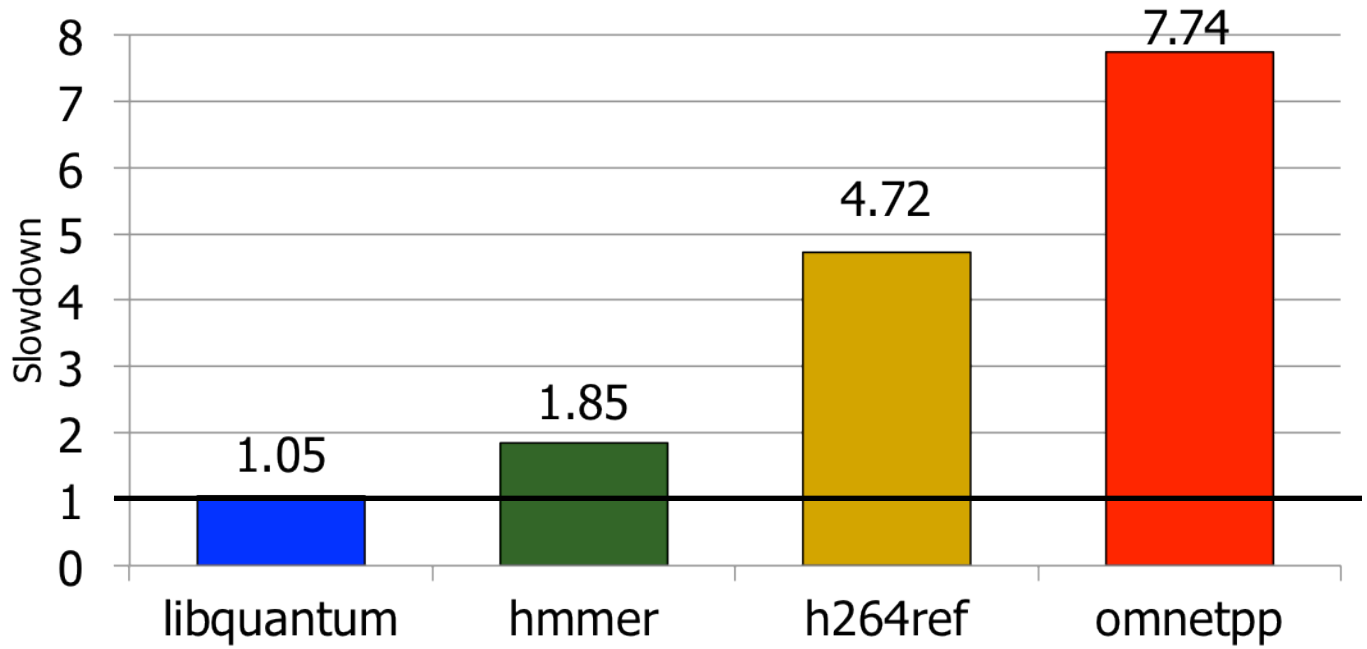
Effect of the Memory Performance Hog



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



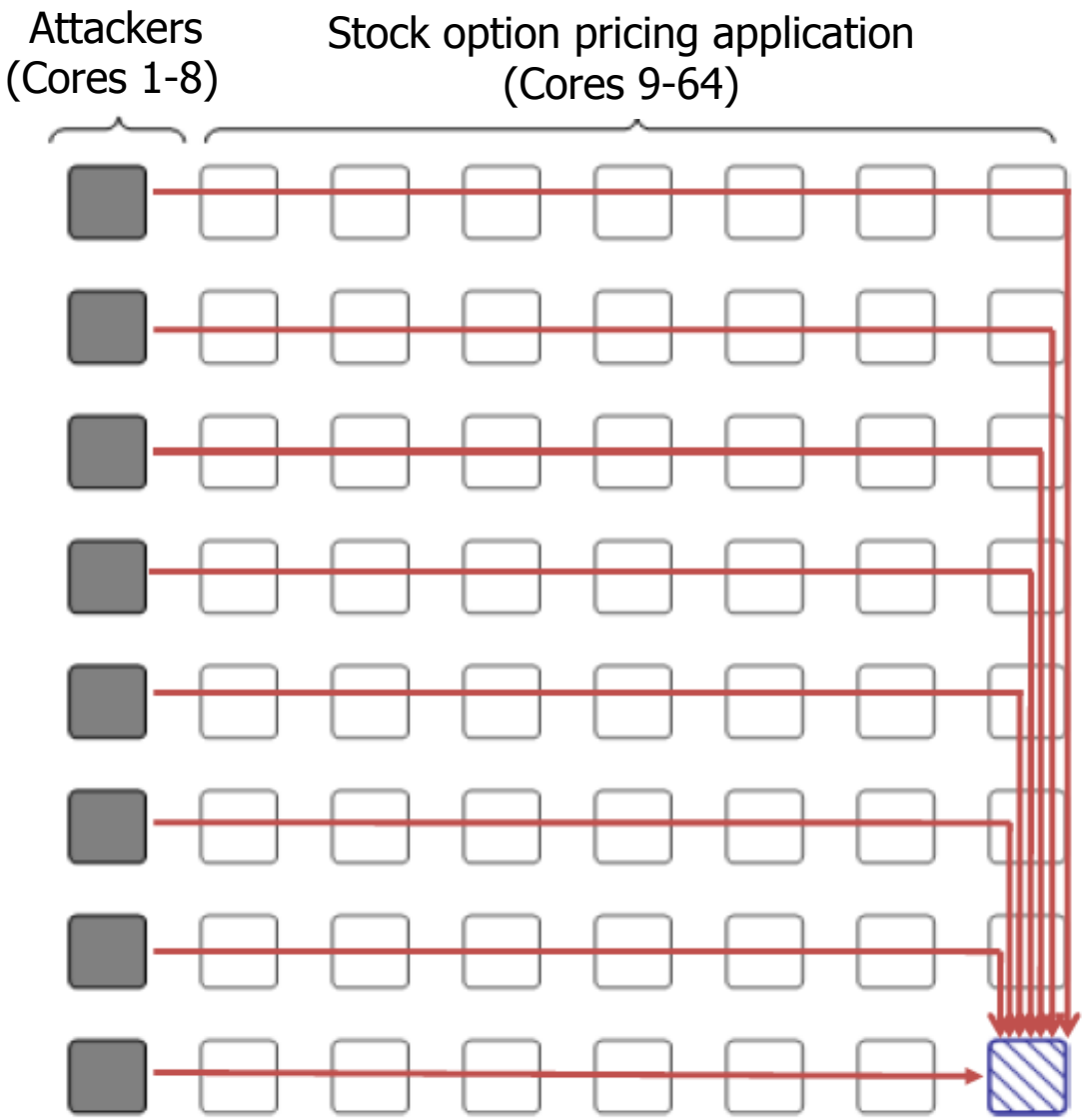
- Vulnerable to denial of service (DoS) [Usenix Security'07]
- Unable to enforce priorities or SLAs [MICRO'07,'10,'11, ISCA'08'11'12, ASPLOS'10]
- Low system performance [IEEE Micro Top Picks '09,'11a,'11b,'12]

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

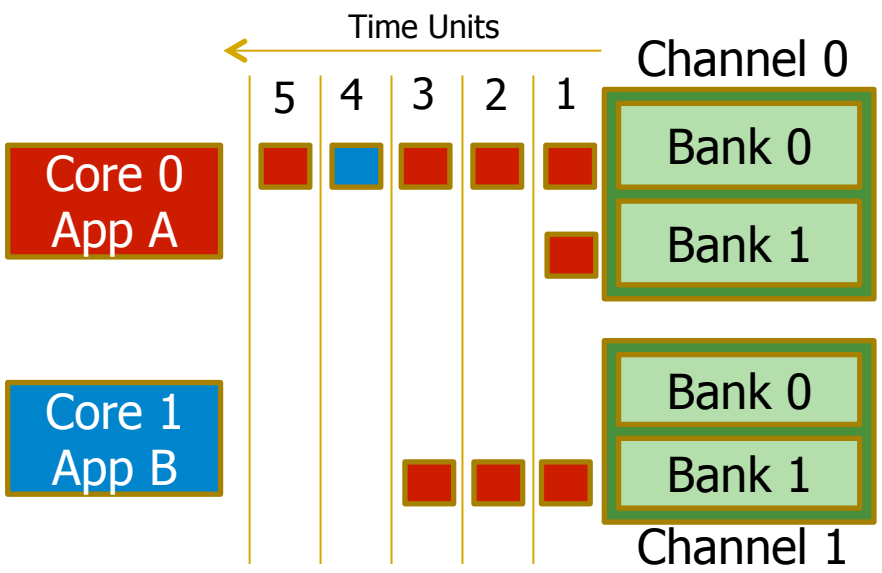
- Problem: Memory interference is uncontrolled → uncontrollable, unpredictable, vulnerable system
- Goal: We need to control it → Design a QoS-aware system
- Solution: Hardware/software cooperative memory QoS
  - Hardware designed to provide a configurable fairness substrate
    - Application-aware memory 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

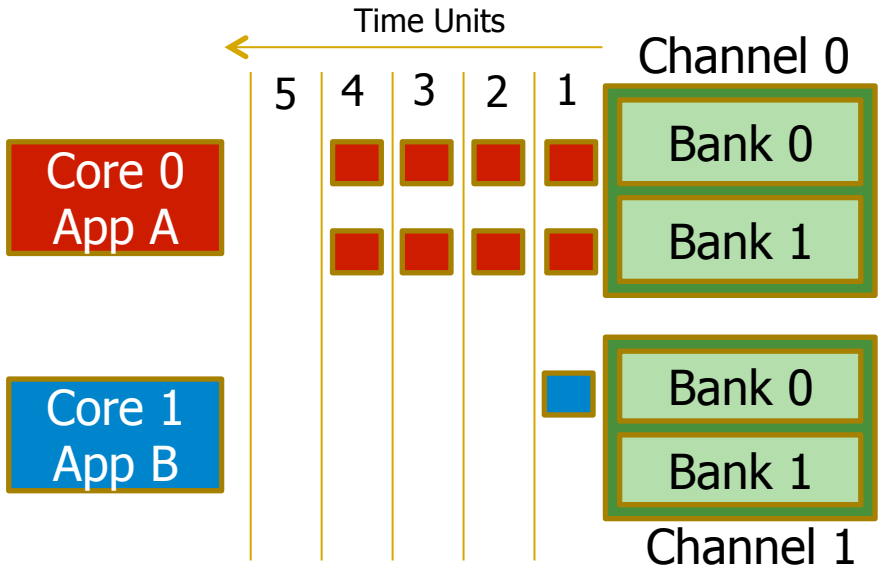
- Smart resources: Design each shared resource to have a configurable interference control/reduction mechanism
  - QoS-aware memory controllers [Mutlu+ MICRO'07] [Moscibroda+, Usenix Security'07] [Mutlu+ ISCA'08, Top Picks'09] [Kim+ HPCA'10] [Kim+ MICRO'10, Top Picks'11] [Ebrahimi+ ISCA'11, MICRO'11] [Ausavarungnirun+, ISCA'12]
  - 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]
  - QoS-aware data mapping to memory controllers [Muralidhara+ MICRO'11]
  - QoS-aware thread scheduling to cores

A Mechanism to Reduce Memory Interference

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



Conventional Page Mapping



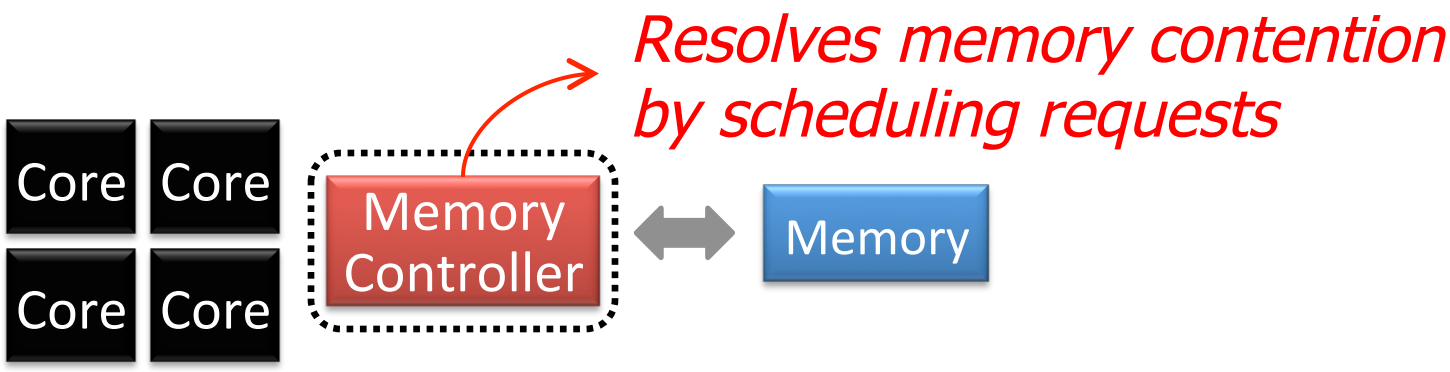
Channel Partitioning

- 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)

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]
  - 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]
  - QoS-aware data mapping to memory controllers [Muralidhara+ MICRO'11]
  - QoS-aware thread scheduling to cores

QoS-Aware Memory Scheduling



- 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

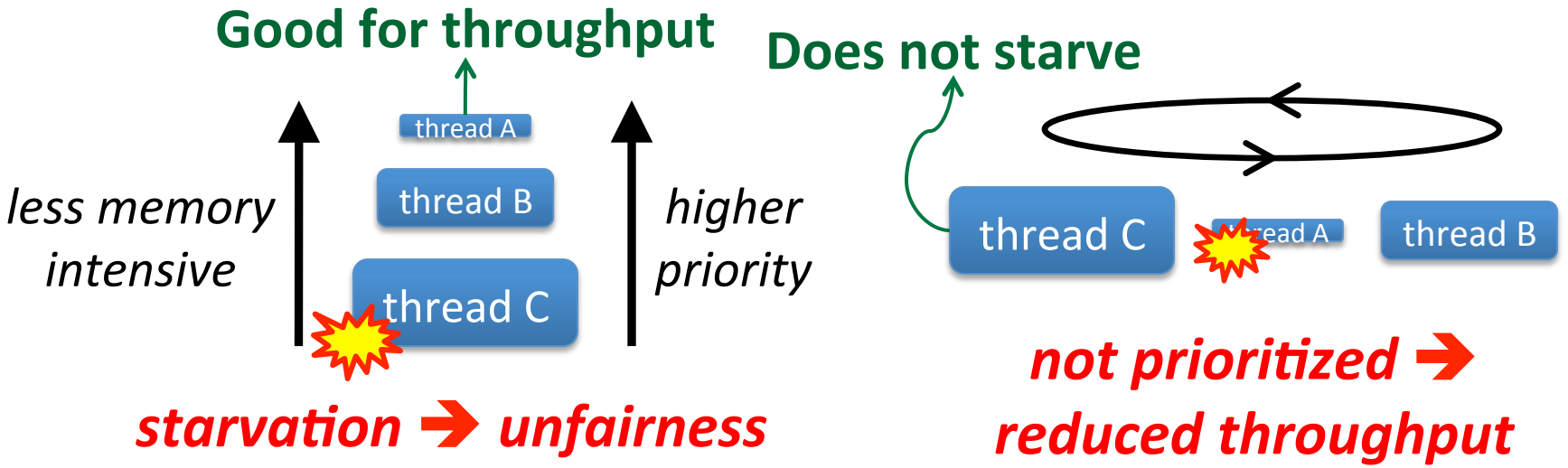
Throughput vs. Fairness

***Throughput biased** approach*

Prioritize less memory-intensive threads

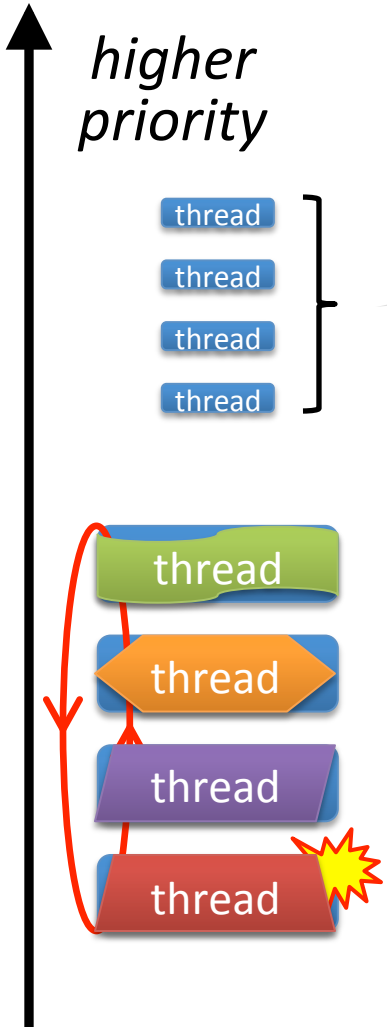
***Fairness biased** approach*

Take turns accessing memory



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

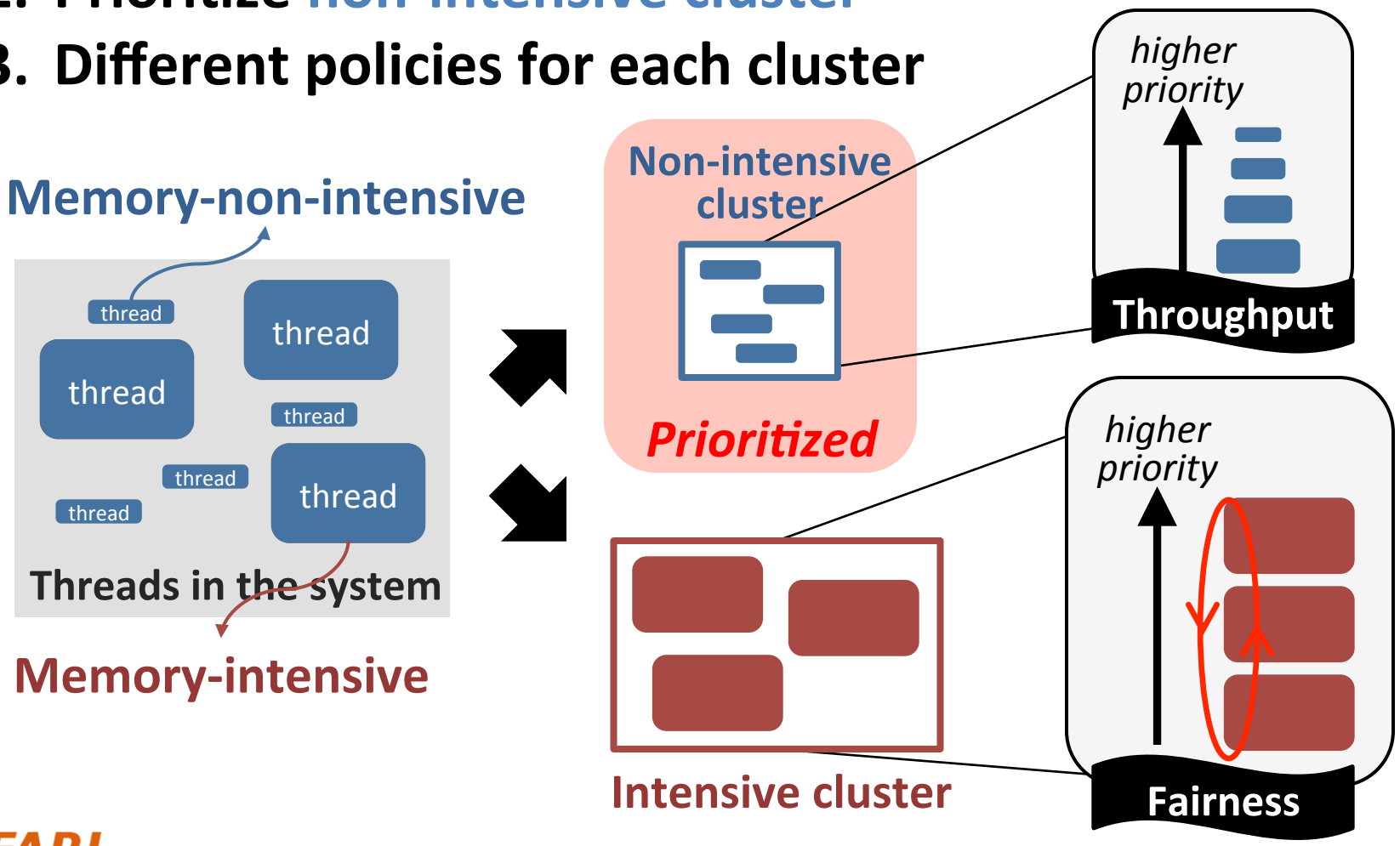
- Shuffle thread ranking

Memory-intensive threads have different vulnerability to interference

- Shuffle asymmetrically

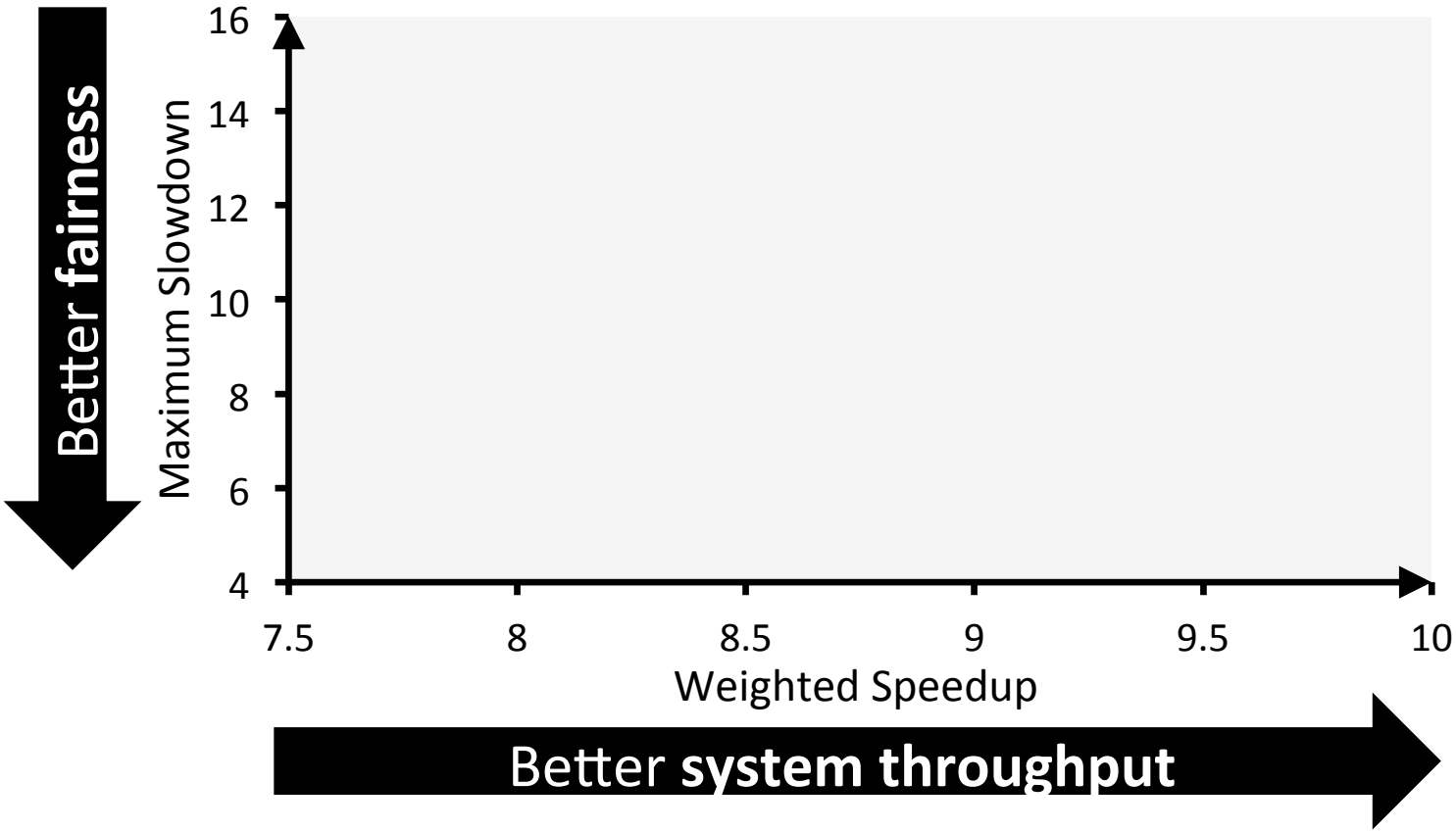
Thread Cluster Memory Scheduling [Kim+ MICRO'10]

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



TCM: Throughput and Fairness

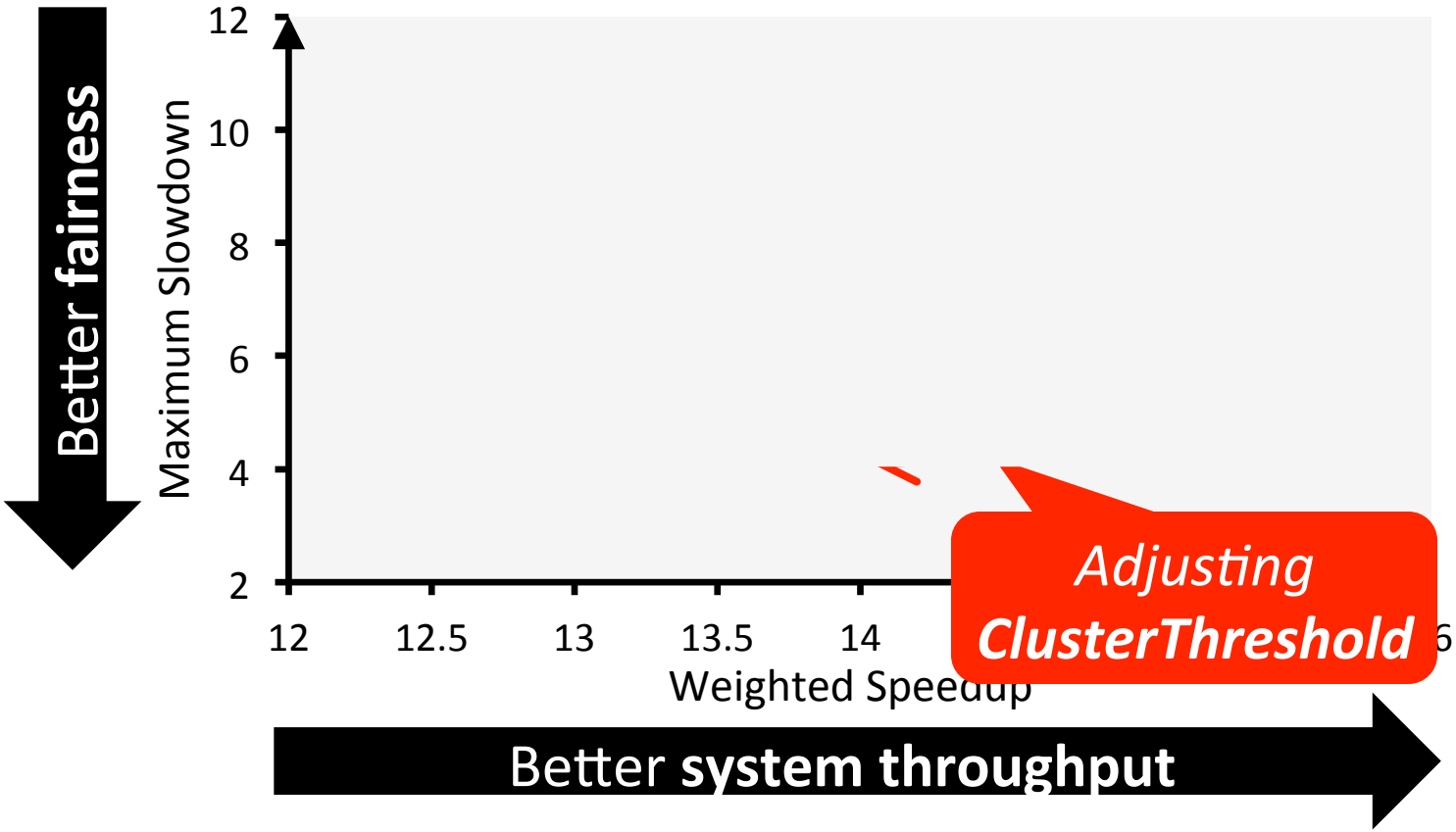
24 cores, 4 memory controllers, 96 workloads



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

TCM: Fairness-Throughput Tradeoff

When configuration parameter is varied…



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

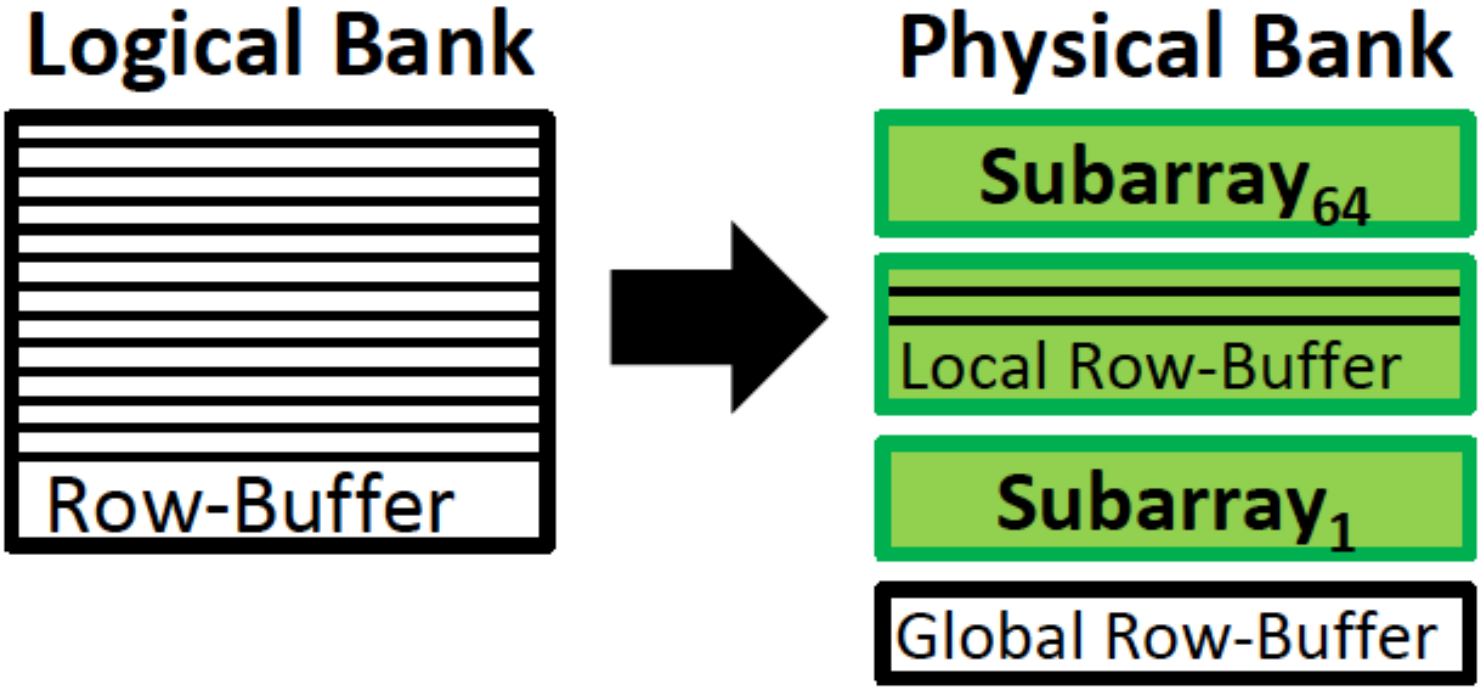
- Approaches: Smart vs. dumb resources
  - Smart resources: QoS-aware memory scheduling
  - Dumb resources: Source throttling; channel partitioning
  - Both approaches are effective in reducing interference
  - No single best approach for all workloads
- Techniques: Request scheduling, source throttling,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

SALP: Reducing DRAM Bank Conflict Impact

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

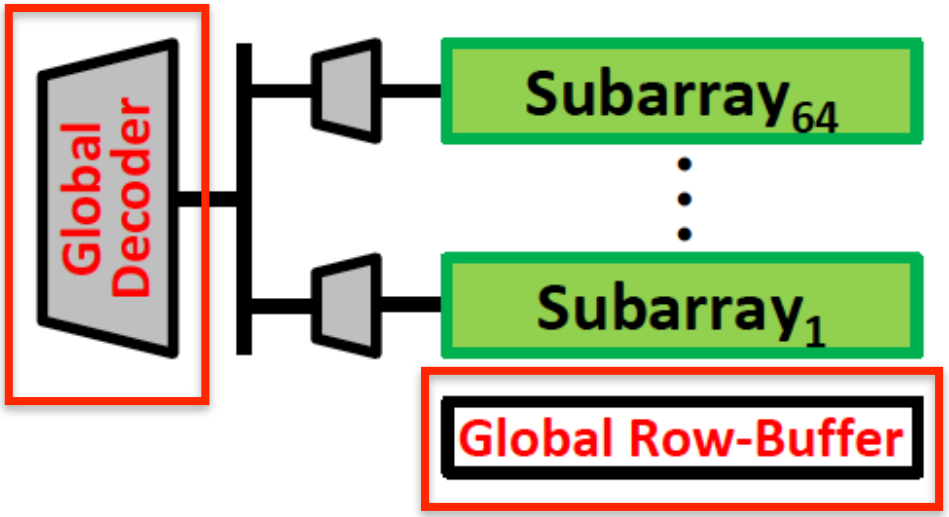
SALP: Problem, Goal, Observations

- 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

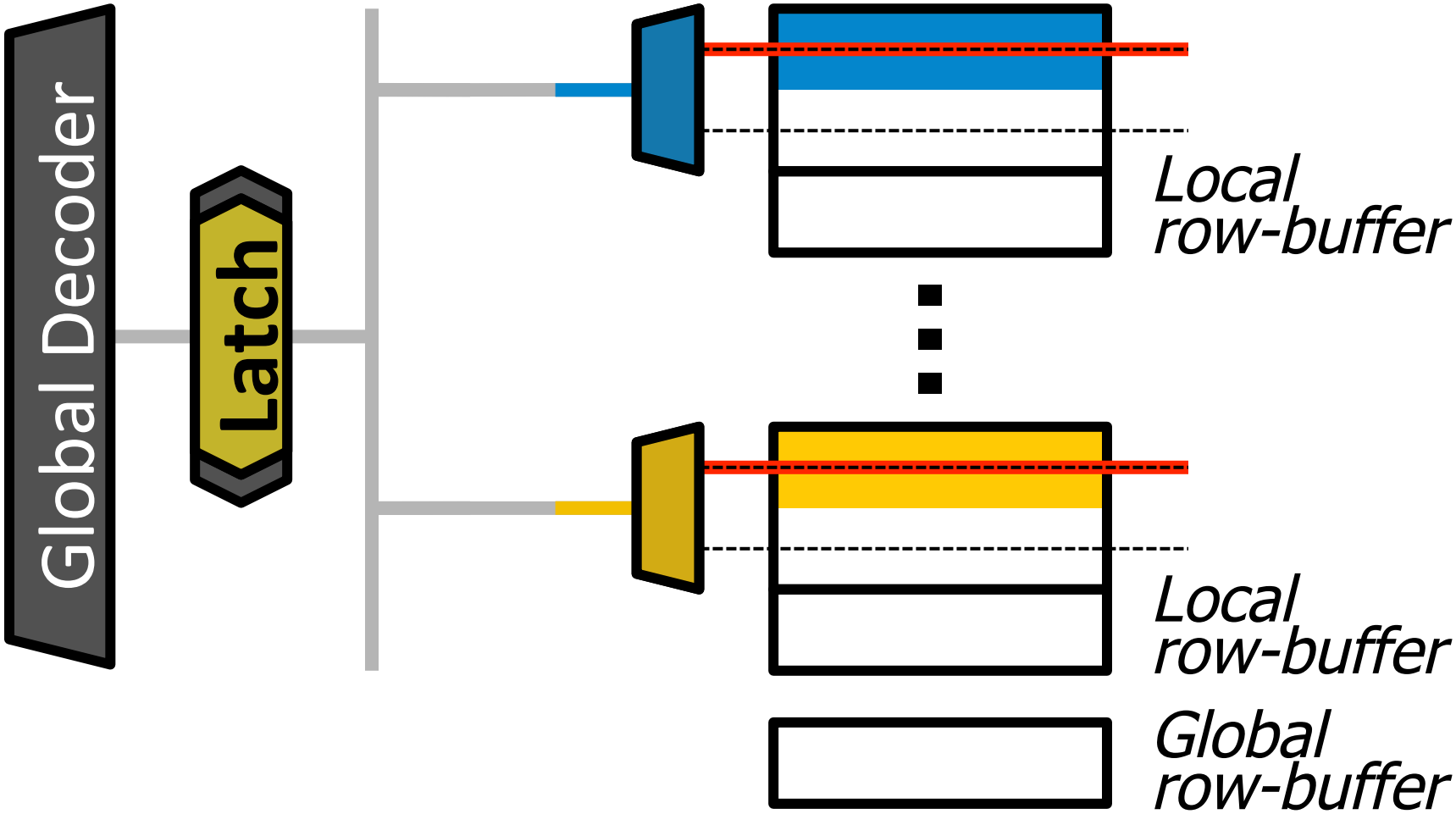


Key Idea of SALP: Minimally reduce sharing of global structures

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

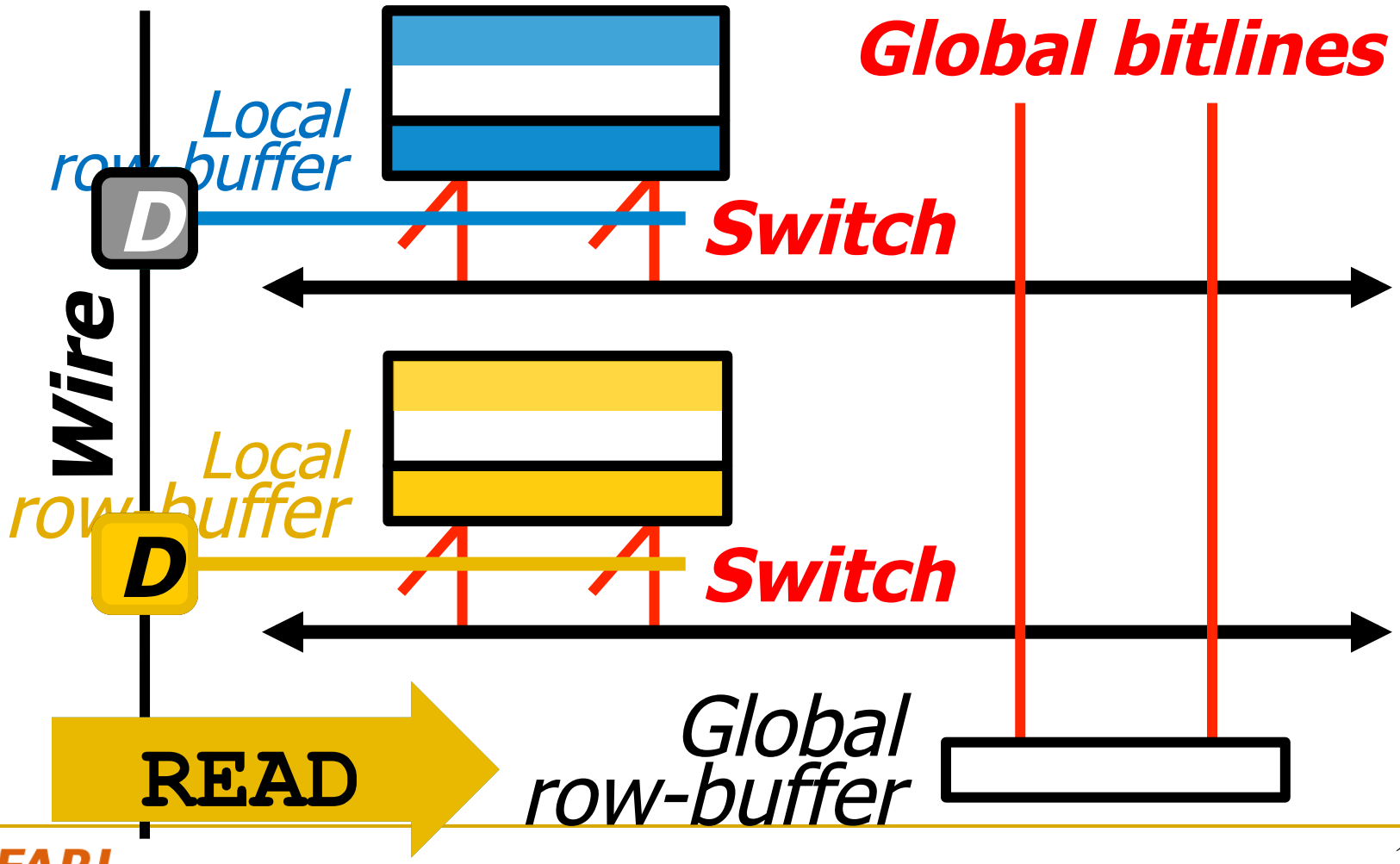
SALP: Reduce Sharing of Global Decoder

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

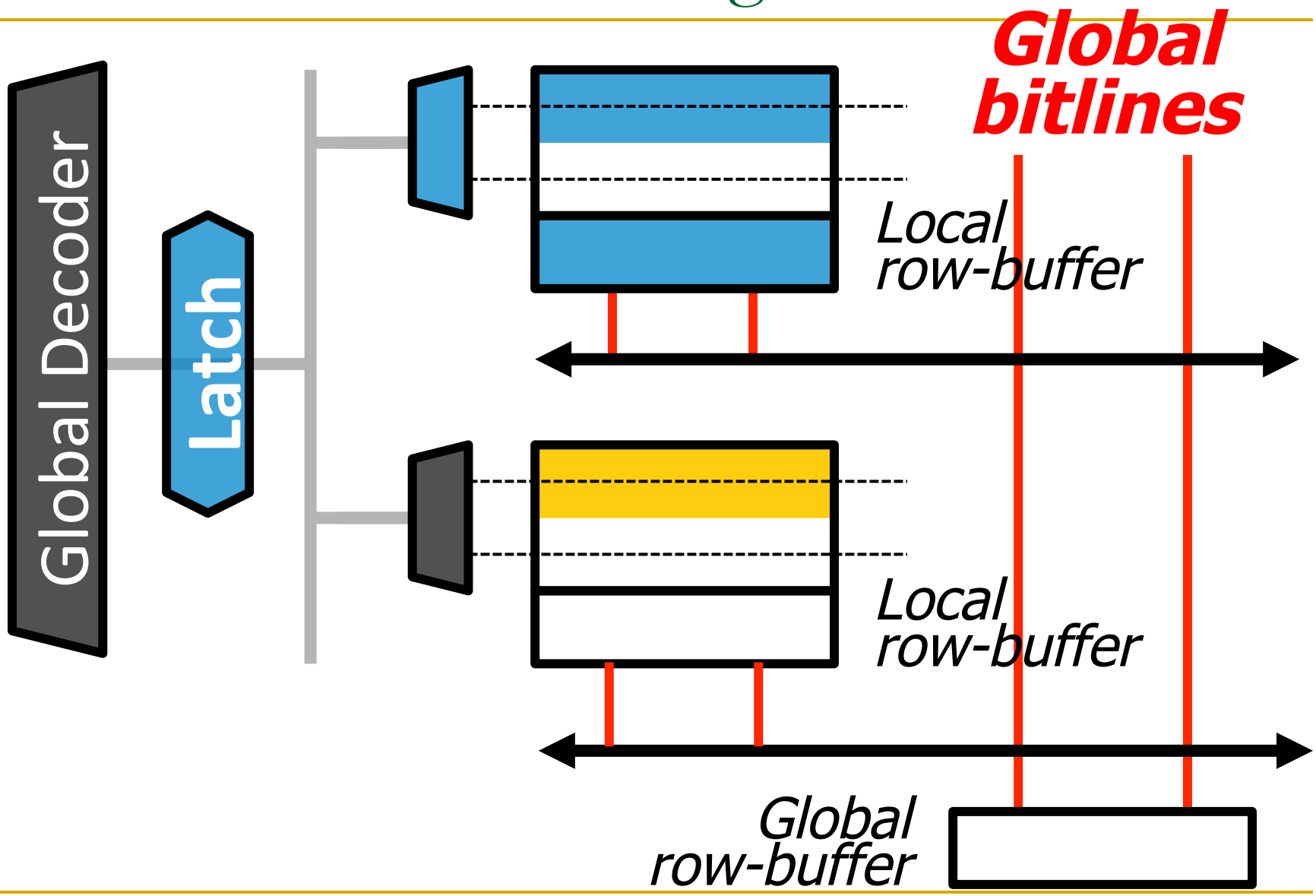


SALP: Reduce Sharing of Global Row-Buffer

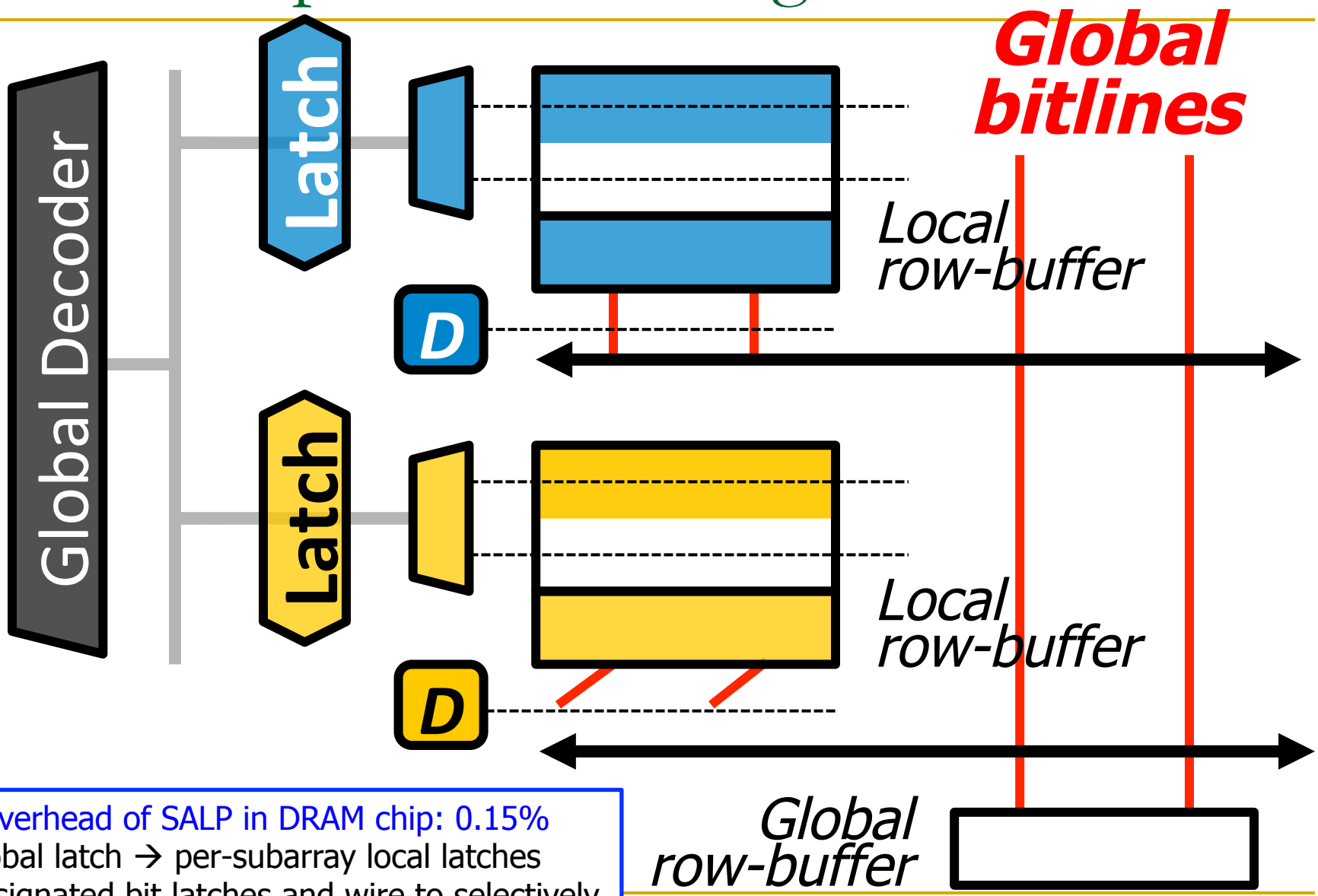
Selectively connect local row-buffers to global row-buffer using a ***Designated*** single-bit latch



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