18-742: Research in Parallel Computer Architecture Memory Systems Research

Prof. Onur Mutlu
Carnegie Mellon University

Fall 2014

September 3, 2014

Reminders

- Homework 0's and Main Memory Scaling reviews
- Please send me and Yixin your 3 papers
  - Your paper reviews are due September 4
- Hamming talk review due September 6
- Think about your projects
  - Project handout will be online soon
  - Proposal will be due ~September 30

Exciting Reading & Project Topic Areas

- ***Rethinking Memory System Design for Data-Intensive Computing***
  - ***All aspects of DRAM, Flash Memory, Emerging Technologies***
- ***Single-Level Stores: Merging Memory and Storage with Fast NVM***
- ***GPUs as First-Class Computing Engines***
- ***In-memory Computing: Enabling Near-Data Processing***
- ***Predictable Systems: QoS Everywhere in the System***
- ***Secure and Easy-to-Program/Manage Memories: DRAM, Flash, NVM***
- ***Heterogeneous Systems: Architecting and Exploiting Asymmetry***
- ***Efficient and Scalable Interconnects***
- ***Genome Sequence Analysis & Assembly: Algorithms and Architectures***

Sample Past Projects from 740/742

- **"ATLAS: A Scalable and High-Performance Scheduling Algorithm for Multiple Memory Controllers"** , HPCA 2010 Best Paper Session.
- **"Next Generation On-Chip Networks: What Kind of Congestion Control Do We Need?"** , HotNets 2010.
- **"Thread Cluster Memory Scheduling: Exploiting Differences in Memory Access Behavior"** , MICRO 2010, IEEE Micro Top Picks 2011.
- **"Reducing Memory Interference in Multicore Systems via Application-Aware Memory Channel Partitioning"**, MICRO 2011.
- **"RAIDR: Retention-Aware Intelligent DRAM Refresh"**, ISCA 2012.
- **"On-Chip Networks from a Networking Perspective: Congestion and Scalability in Many-core Interconnects"**, SIGCOMM 2012.
- **"Row Buffer Locality Aware Caching Policies for Hybrid Memories"**, ICCD 2012 Best Paper Award.
- **"HAT: Heterogeneous Adaptive Throttling for On-Chip Networks"**, SBAC-PAD 2012.
- **"Asymmetry-Aware Execution Placement on Manycore Chips"**, SFMA 2013.
- **"Exploiting Compressed Block Size as an Indicator of Future Reuse"**, SAFARI Technical Report 2013.

Next Week

- Want two presenters for next week (Tuesday and Thursday)
- Pick a set of papers to present so that we can have a discussion
  - We will decide this at the end of this meeting
- Signup sheet for later weeks will be posted

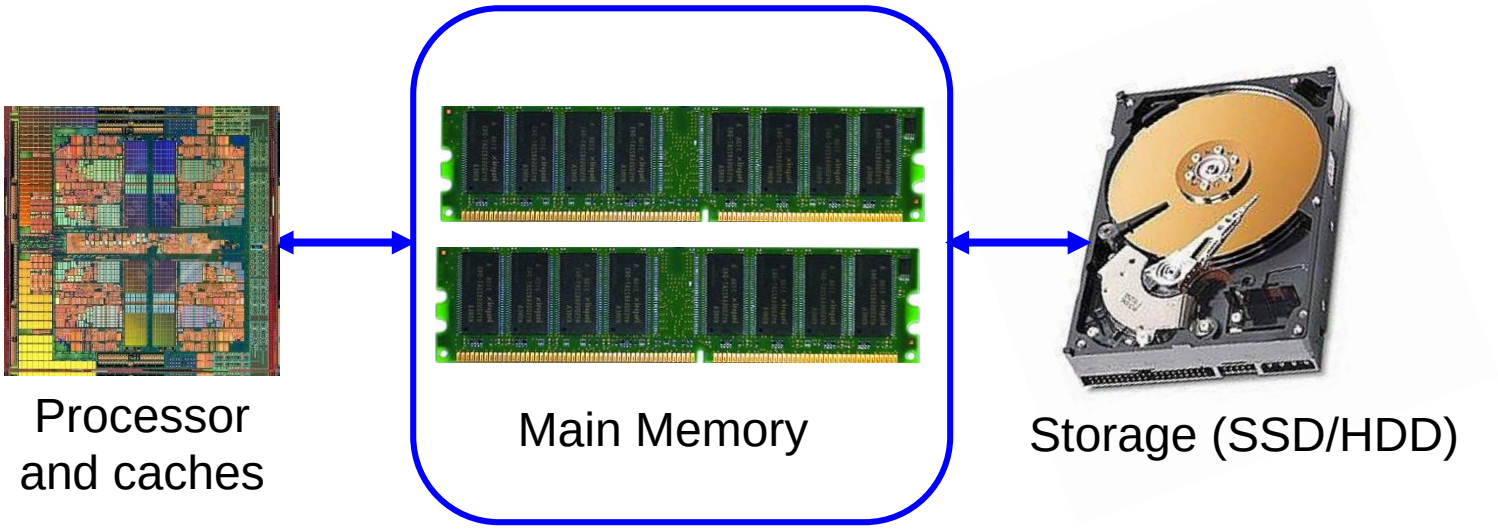
Rethinking Memory/Storage System Design

Onur Mutlu
onur@cmu.edu
http://users.ece.cmu.edu/~omutlu/

Carnegie Mellon

SAFARI

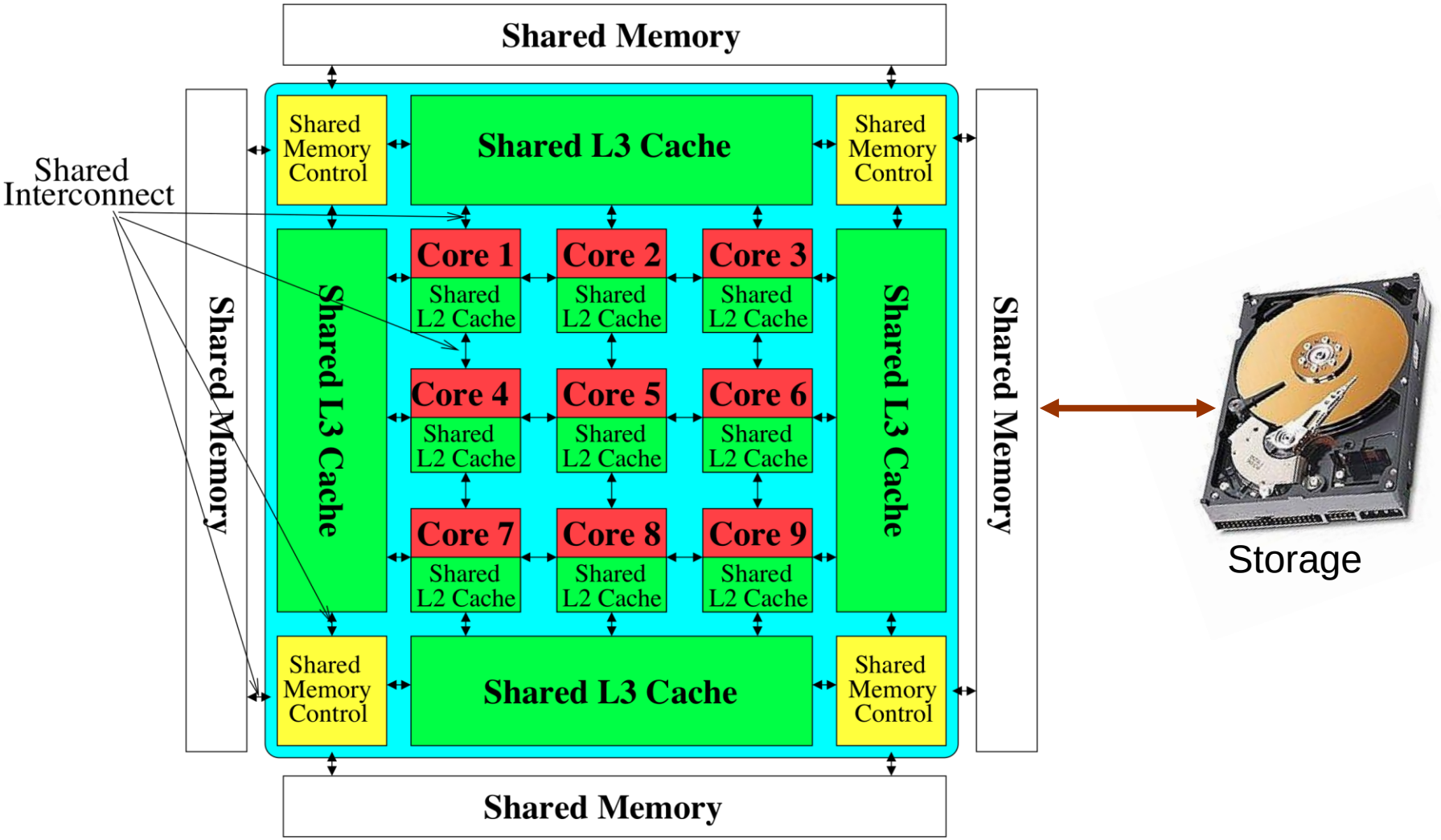
The Main Memory System



- 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 Memory Scaling Problem and Solution Directions
  - New Memory Architectures
  - Enabling Emerging Technologies: Hybrid Memory Systems
- How Can We Do Better?
- Summary

Major Trends Affecting Main Memory (I)

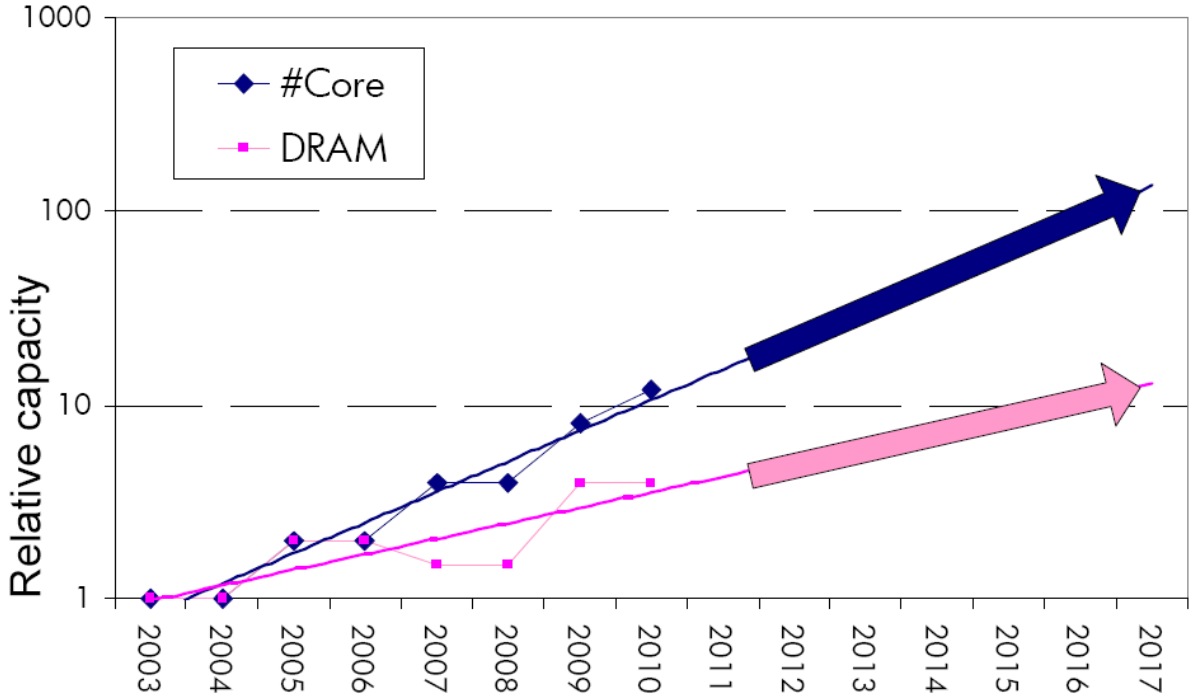
- Need for main memory capacity, bandwidth, QoS increasing
- Main memory energy/power is a key system design concern
- DRAM technology scaling is ending

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

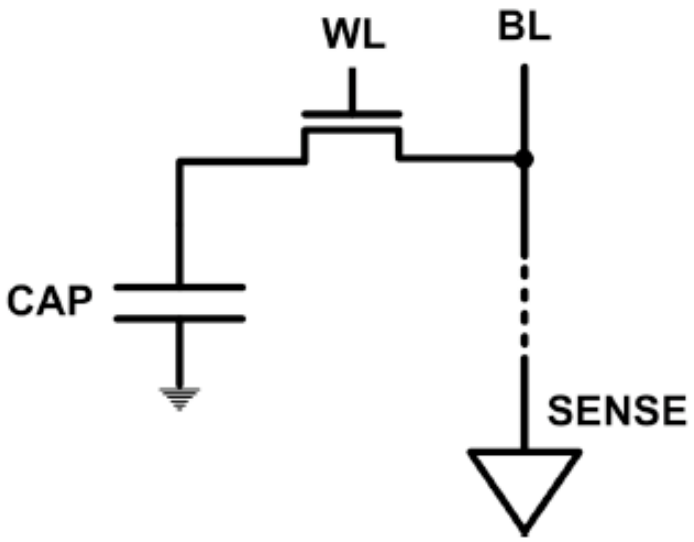
- DRAM technology scaling is ending
  - ITRS projects DRAM will not scale easily below X nm
  - Scaling has provided many benefits:
    - higher capacity (density), lower cost, lower energy

Agenda

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

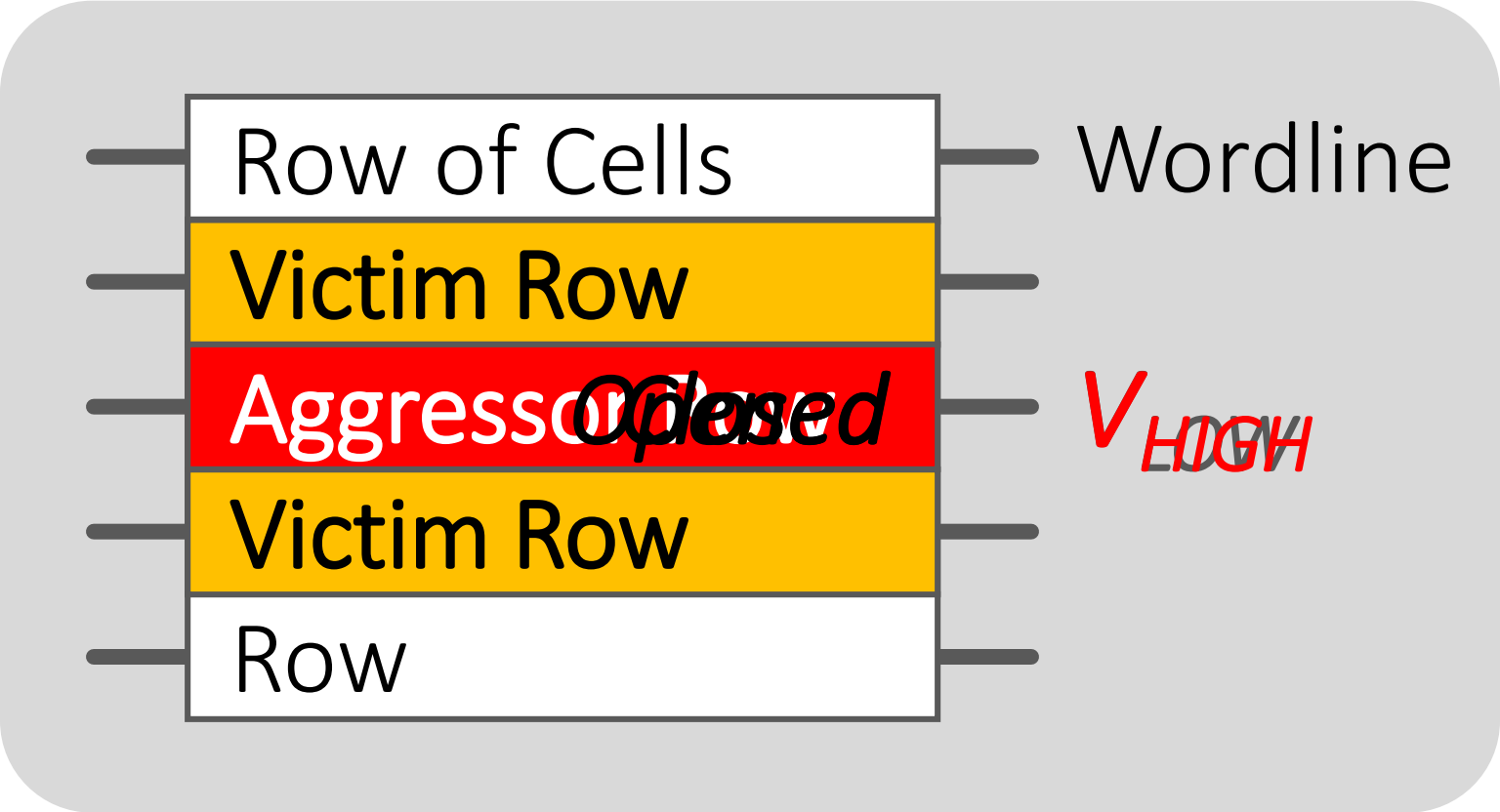
The DRAM Scaling Problem

- 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]



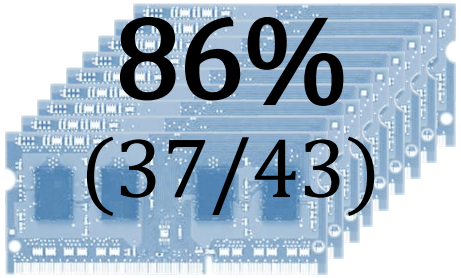
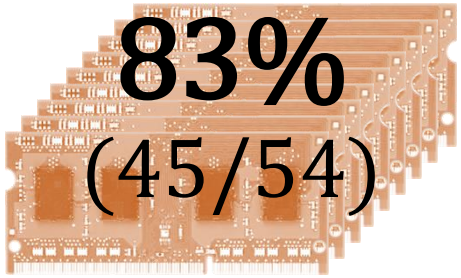
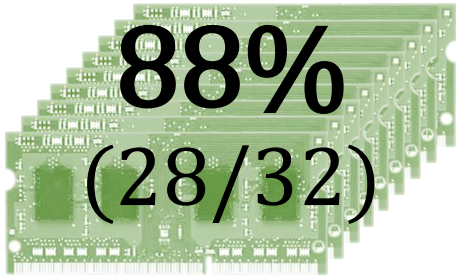
- DRAM capacity, cost, and energy/power hard to scale

An Example of The Scaling Problem



*Repeatedly opening and closing a row induces **disturbance errors** in adjacent rows in **most real DRAM chips** [Kim+ ISCA 2014]*

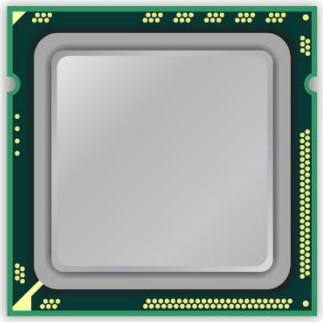
Most DRAM Modules Are at Risk

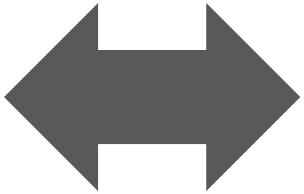
A company	B company	C company
		
86% (37/43)	**83%** (45/54)	**88%** (28/32)
Up to **1.0×10^{7}** errors	Up to **2.7×10^{6}** errors	Up to **3.3×10^{5}** errors

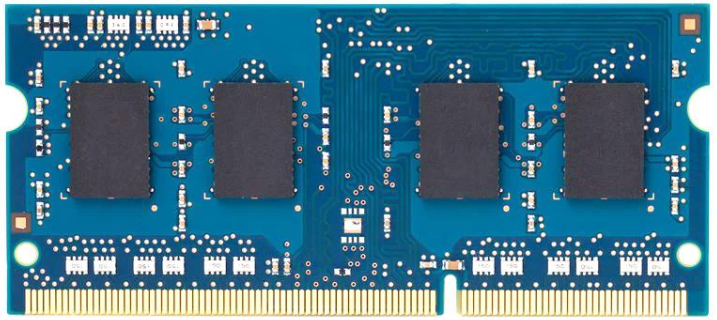
Kim+, "Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors," ISCA 2014.

x86 CPU

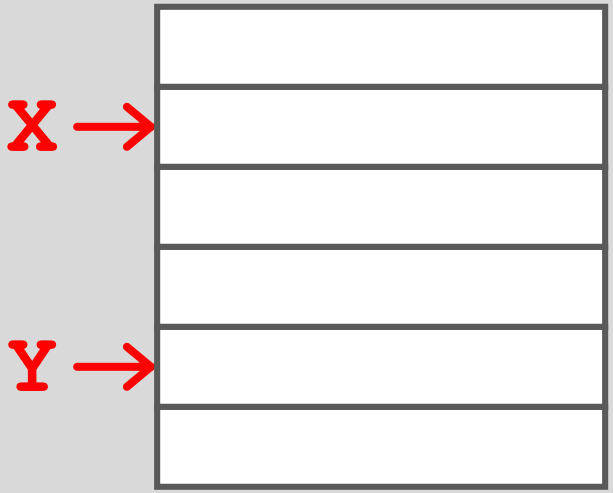
DRAM Module





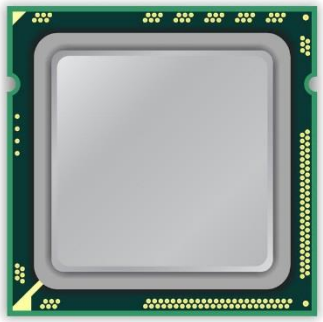


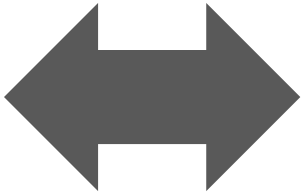
```
loop:
   mov (X), %eax
   mov (Y), %ebx
   clflush (X)
   clflush (Y)
   mfence
   jmp loop
```

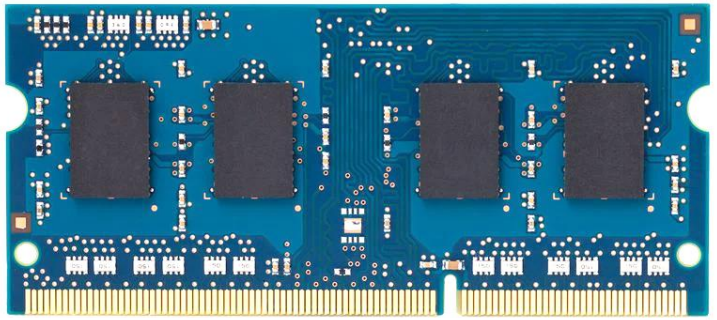


x86 CPU

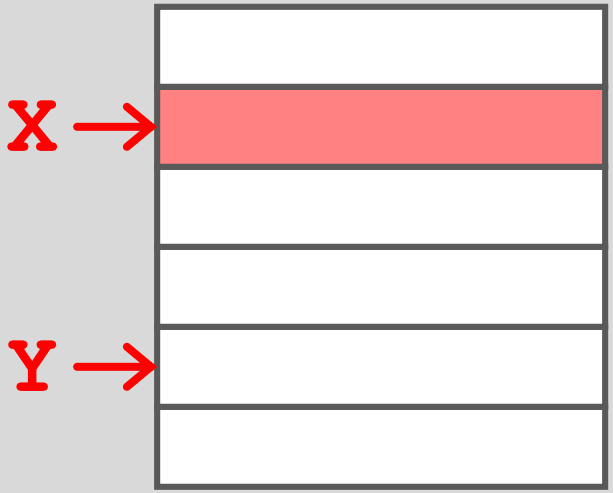
DRAM Module





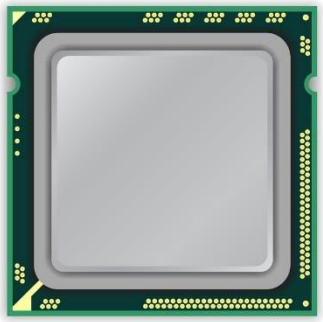


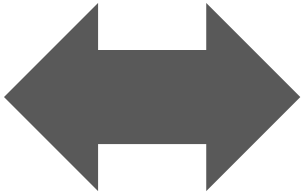
```
loop:
   mov (X), %eax
   mov (Y), %ebx
   clflush (X)
   clflush (Y)
   mfence
   jmp loop
```

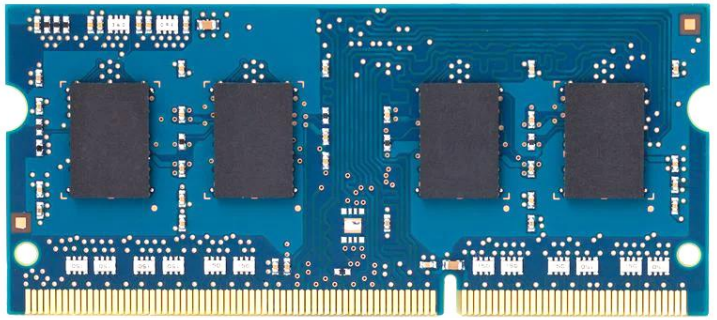


x86 CPU

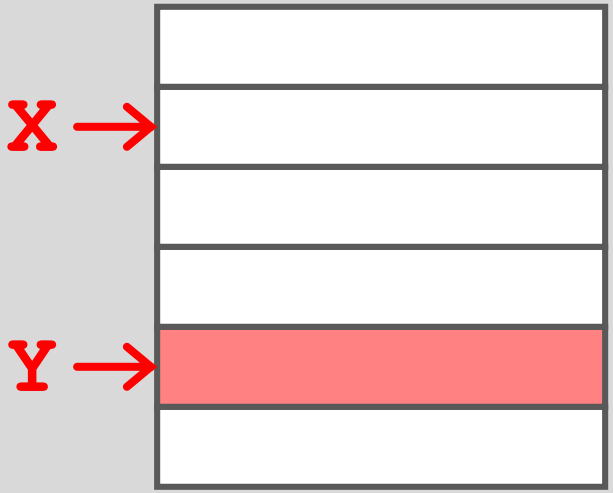
DRAM Module





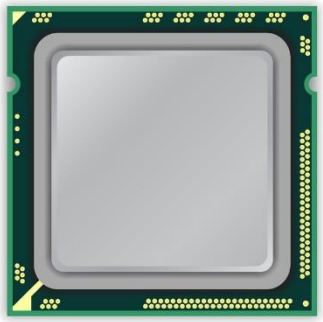


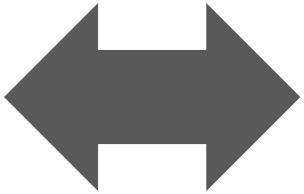
```
loop:
   mov (X), %eax
   mov (Y), %ebx
   clflush (X)
   clflush (Y)
   mfence
   jmp loop
```

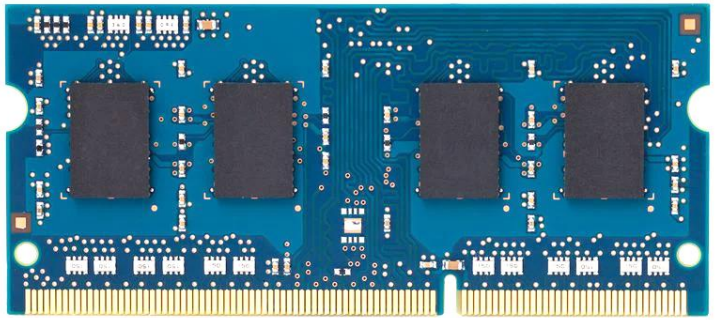


x86 CPU

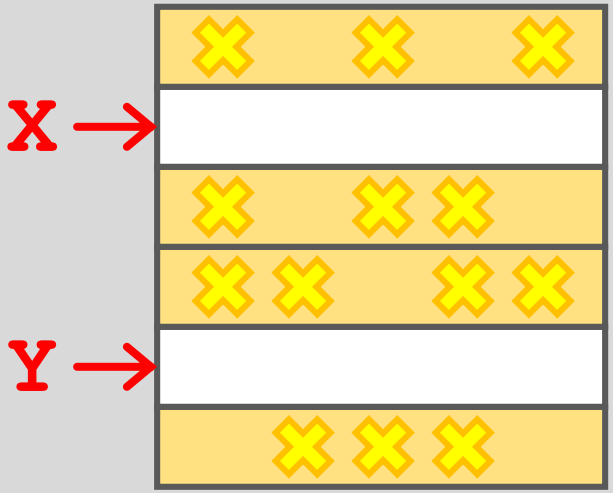
DRAM Module







```
loop:
   mov (X), %eax
   mov (Y), %ebx
   clflush (X)
   clflush (Y)
   mfence
   jmp loop
```



Observed Errors in Real Systems

CPU Architecture	Errors	Access-Rate
Intel Haswell (2013)	22.9K	12.3M/sec
Intel Ivy Bridge (2012)	20.7K	11.7M/sec
Intel Sandy Bridge (2011)	16.1K	11.6M/sec
AMD Piledriver (2012)	59	6.1M/sec

- *In a more controlled environment, we can induce as many as ten million disturbance errors*
- *Disturbance errors are a serious reliability issue*

Kim+, "Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors," ISCA 2014.

The DRAM Scaling Problem

DRAM Process Scaling Challenges

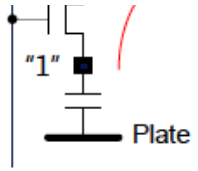
- **Refresh**
  - Difficult to build high-aspect ratio cell capacitors decreasing cell capacitance

THE MEMORY FORUM 2014

Co-Architecting Controllers and DRAM to Enhance DRAM Process Scaling

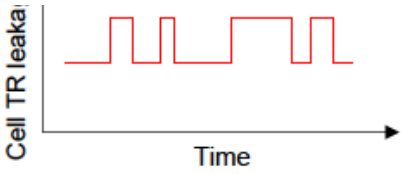
Uksong Kang, Hak-soo Yu, Churoo Park, *Hongzhong Zheng,
**John Halbert, **Kuljit Bains, SeongJin Jang, and Joo Sun Choi

*Samsung Electronics, Hwasung, Korea / *Samsung Electronics, San Jose / **Intel*



Refresh

tWR



VRT

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 energy
  - Enable reliability at low cost
  - Improve bandwidth and latency
  - Reduce waste
  - Enable computation close to data

Solution 1: Tolerate DRAM

- Liu+, "RAIDR: Retention-Aware Intelligent DRAM Refresh," ISCA 2012.
- Kim+, "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 2013.
- Seshadri+, "RowClone: Fast and Efficient In-DRAM Copy and Initialization of Bulk Data," MICRO 2013.
- Pekhimenko+, "Linearly Compressed Pages: A Main Memory Compression Framework," MICRO 2013.
- Chang+, "Improving DRAM Performance by Parallelizing Refreshes with Accesses," HPCA 2014.
- Khan+, "The Efficacy of Error Mitigation Techniques for DRAM Retention Failures: A Comparative Experimental Study," SIGMETRICS 2014.
- Luo+, "Characterizing Application Memory Error Vulnerability to Optimize Data Center Cost," DSN 2014.
- Kim+, "Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors," ISCA 2014.

Avoid DRAM:

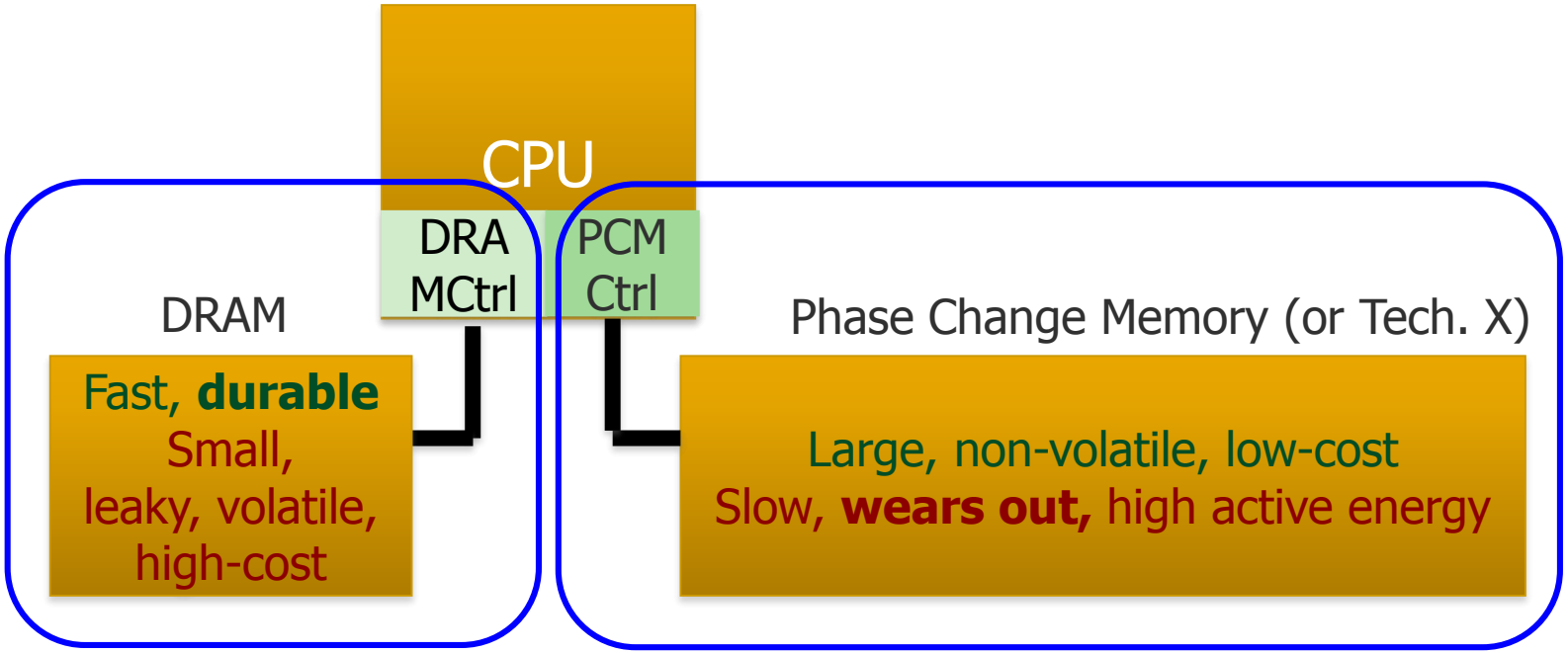
- Seshadri+, "The Evicted-Address Filter: A Unified Mechanism to Address Both Cache Pollution and Thrashing," PACT 2012.
- Pekhimenko+, "Base-Delta-Immediate Compression: Practical Data Compression for On-Chip Caches," PACT 2012.
- Seshadri+, "The Dirty-Block Index," ISCA 2014.

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.
- Meza+, "A Case for Efficient Hardware-Software Cooperative Management of Storage and Memory," WEED 2013.

Hybrid Memory Systems

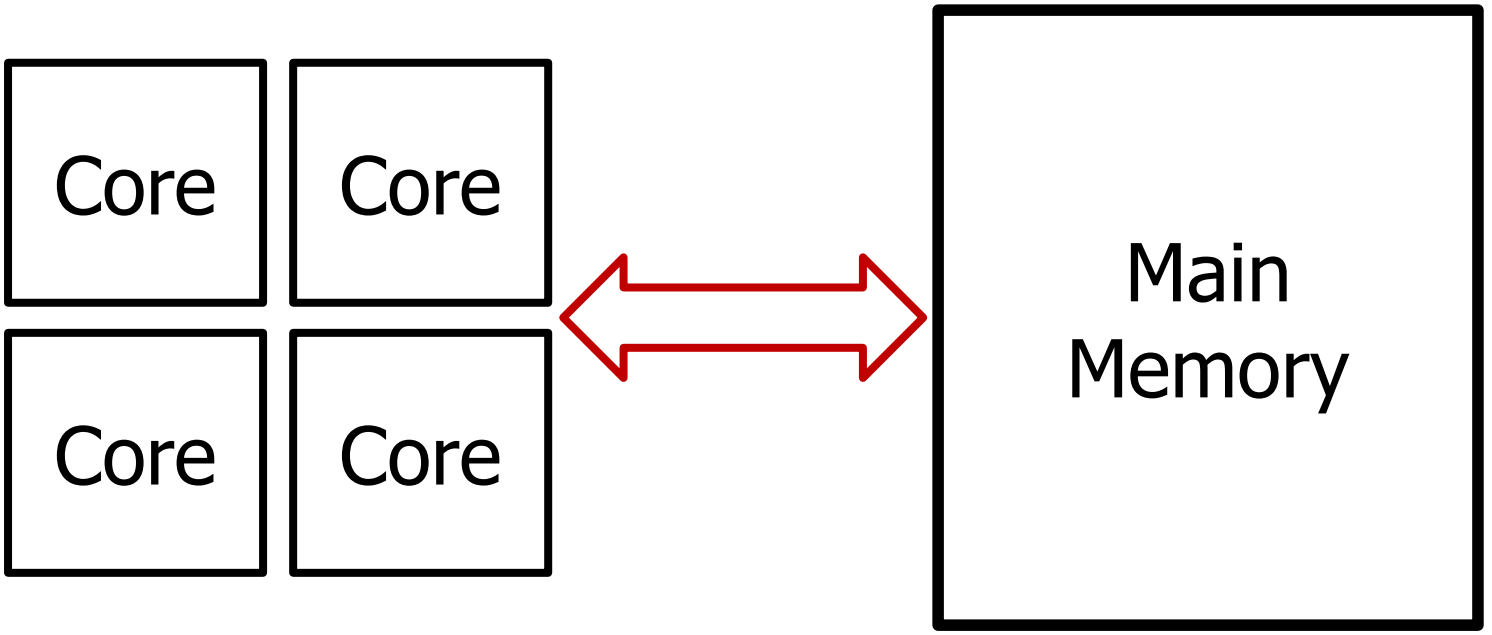


Hardware/software manage data allocation and movement
to achieve the best of multiple technologies

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.

An Orthogonal Issue: Memory Interference



Cores' interfere with each other when accessing shared main memory

An Orthogonal Issue: Memory Interference

- Problem: Memory interference between cores is uncontrolled
  → unfairness, starvation, low performance
  → uncontrollable, unpredictable, vulnerable system

- Solution: QoS-Aware Memory Systems
  - 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

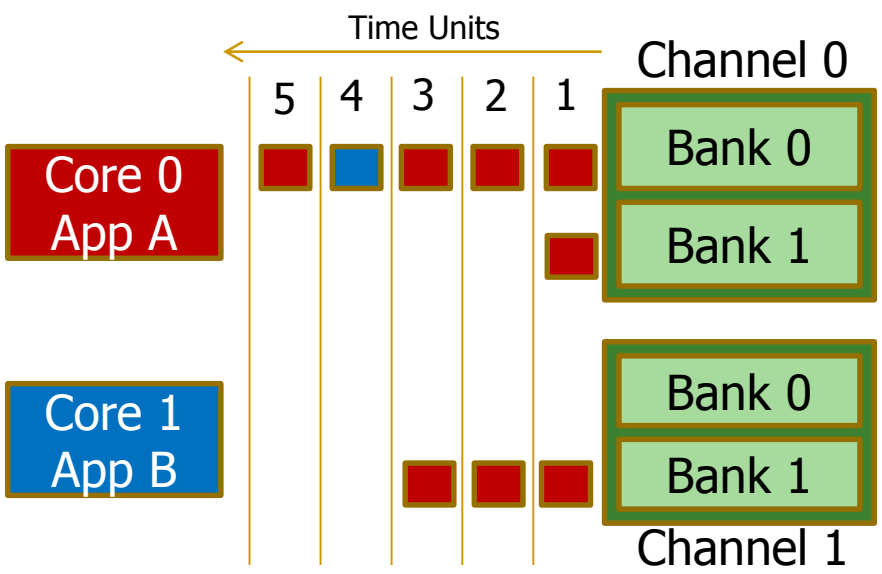
- QoS-aware memory controllers and interconnects can provide predictable performance and higher efficiency

Designing QoS-Aware Memory Systems: Approaches

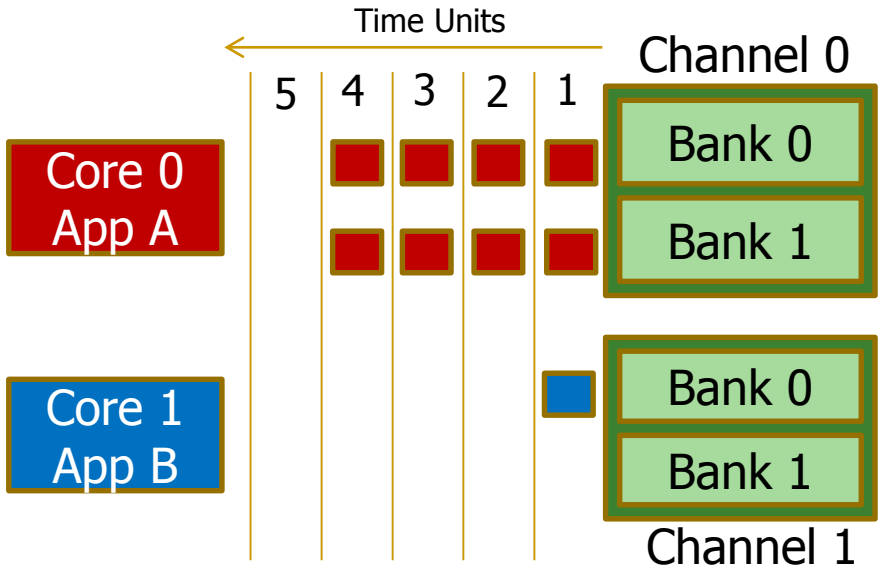
- Smart resources: Design each shared resource to have a configurable interference control/reduction mechanism
  - QoS-aware memory controllers [Mutlu+ MICRO'07] [Moscibroda+, Usenix Security'07] [Mutlu+ ISCA'08, Top Picks'09] [Kim+ HPCA'10] [Kim+ MICRO'10, Top Picks'11] [Ebrahimi+ ISCA'11, MICRO'11]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



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)

More on Memory Channel Partitioning

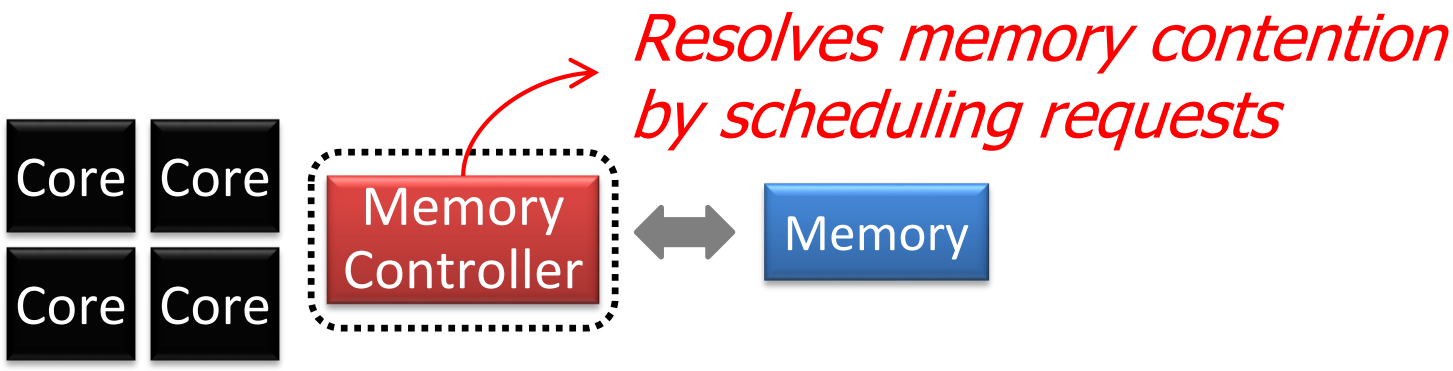
- Sai Prashanth Muralidhara, Lavanya Subramanian, Onur Mutlu, Mahmut Kandemir, and Thomas Moscibroda,
  "Reducing Memory Interference in Multicore Systems via Application-Aware Memory Channel Partitioning"
  *Proceedings of the 44th International Symposium on Microarchitecture (**MICRO**)*, Porto Alegre,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

In class meeting on September 3, we discussed until here.

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

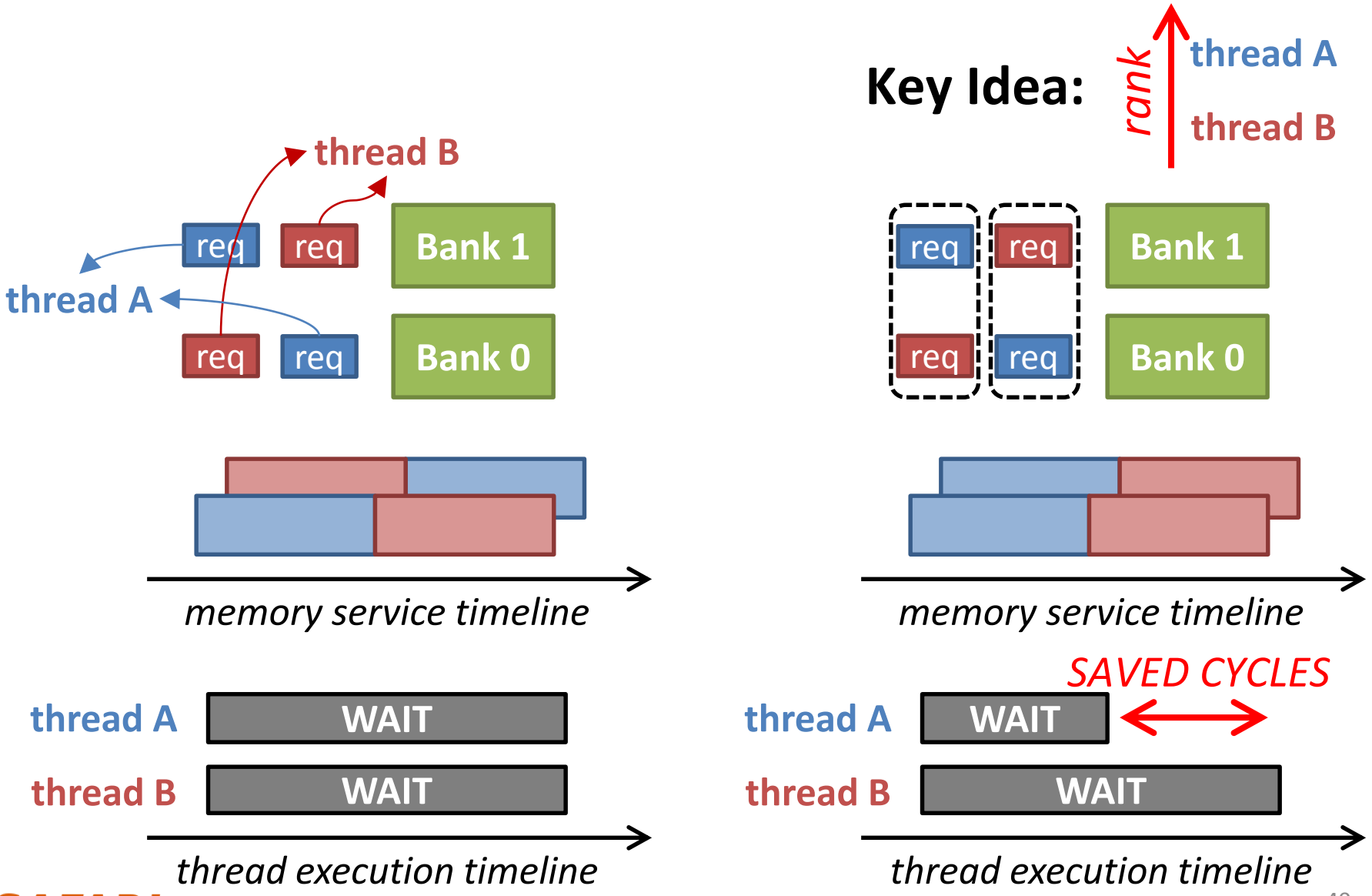
QoS-Aware Memory Scheduling: Evolution

- 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

- ATLAS memory scheduler [Kim+ HPCA'10]

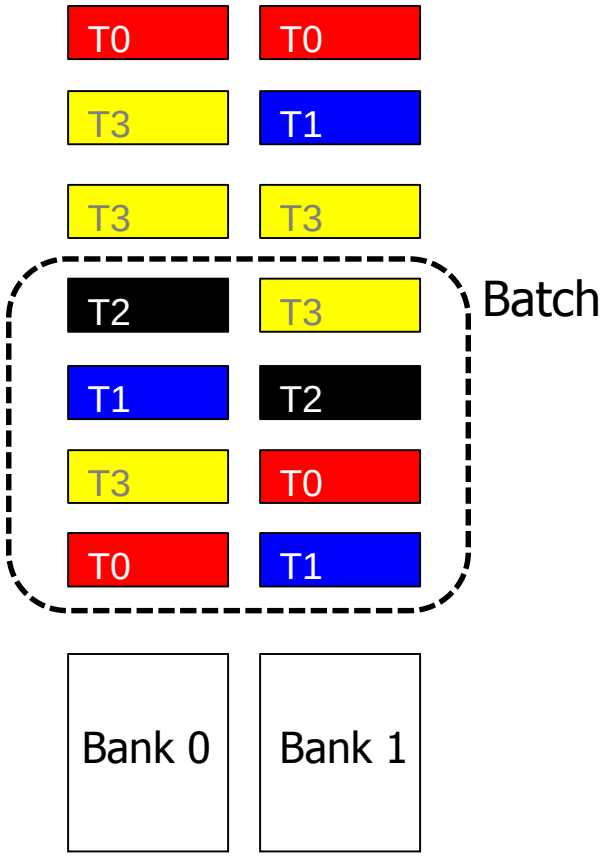
Within-Thread Bank Parallelism



Parallelism-Aware Batch Scheduling [ISCA'08]

- Principle 1: Schedule requests from a thread back to back
  - Preserves each thread's bank parallelism
  - But, this can cause starvation…

- Principle 2: Group a fixed number of oldest requests from each thread into a "batch"
  - Service the batch before all other requests
  - Form a new batch when the current batch is done
  - Eliminates starvation, provides fairness



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

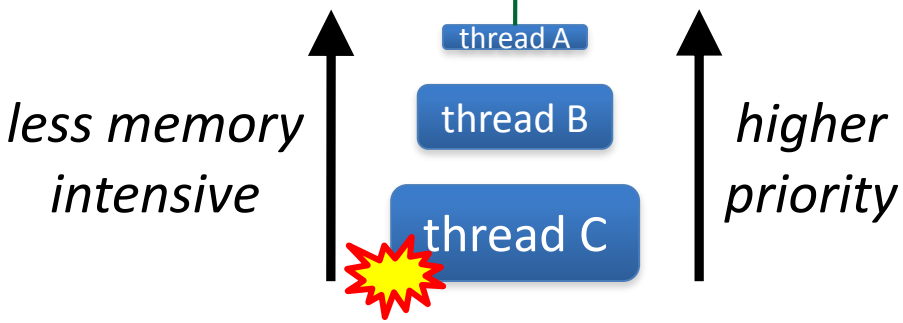
***Throughput biased** approach*

Prioritize less memory-intensive threads

***Fairness biased** approach*

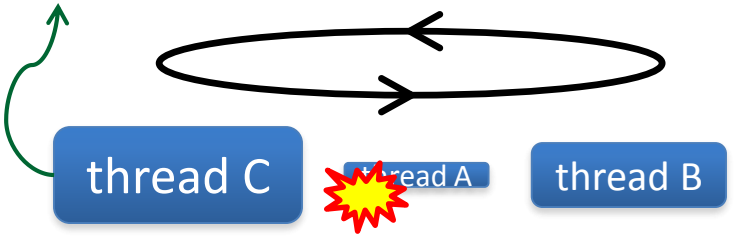
Take turns accessing memory

Good for throughput



starvation → unfairness

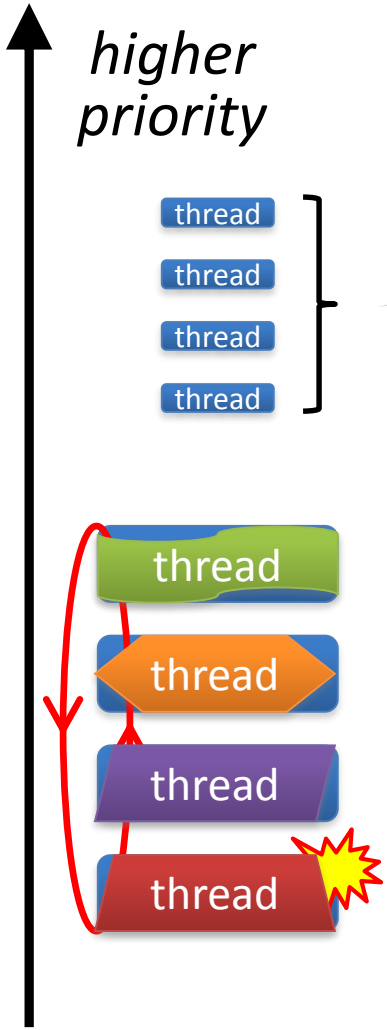
Does not starve



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

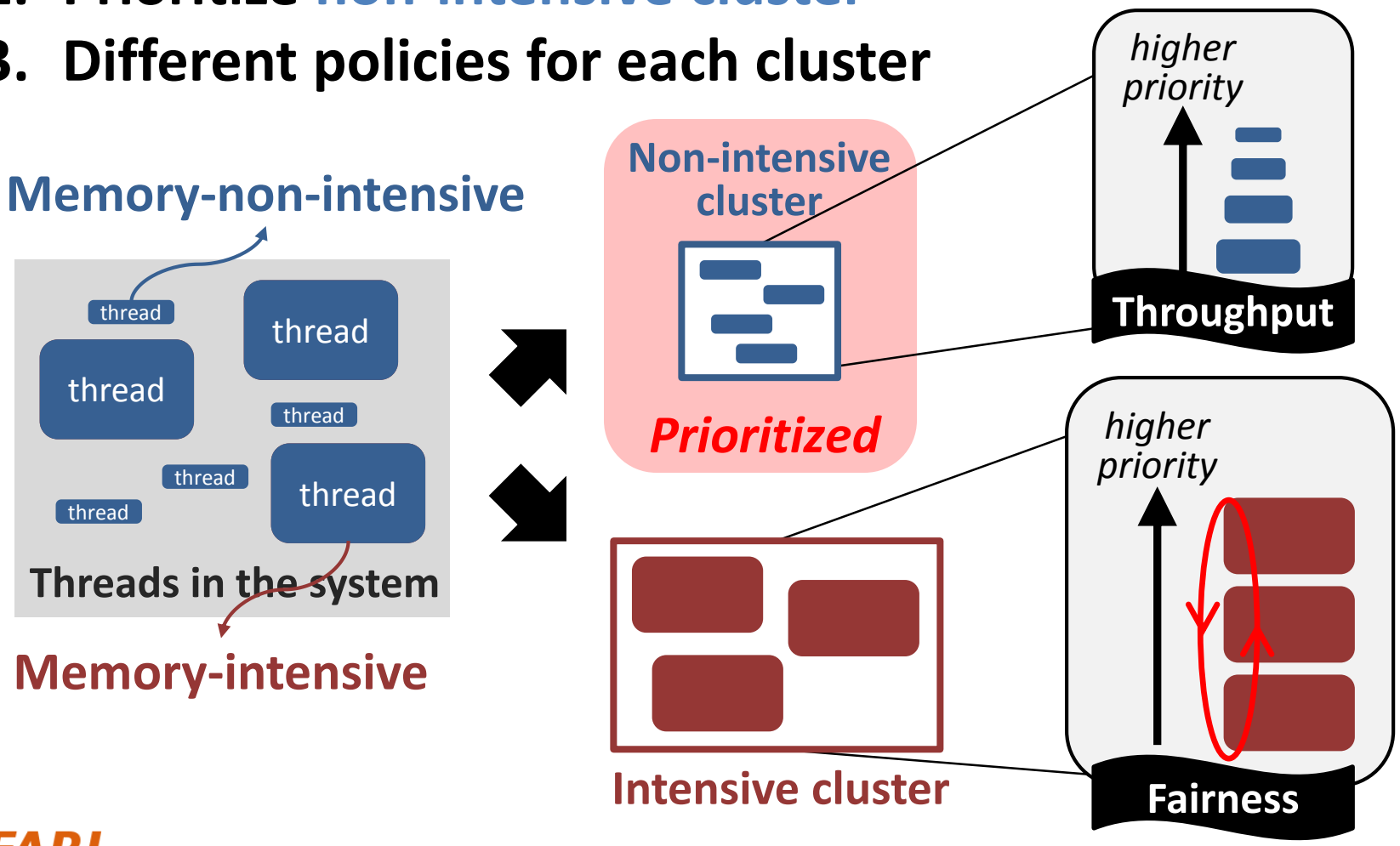
- Shuffle thread ranking

Memory-intensive threads have different vulnerability to interference

- Shuffle asymmetrically

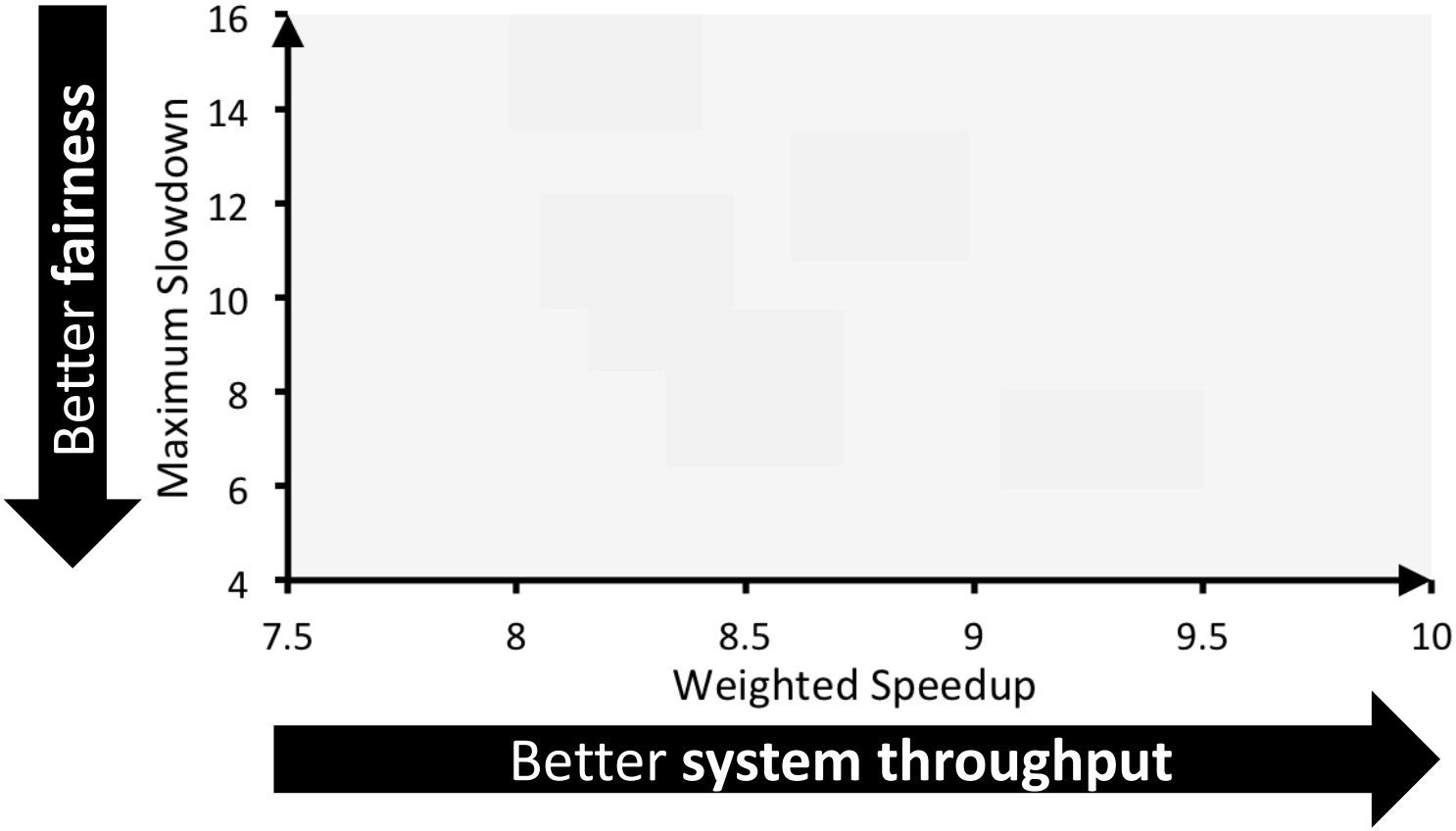
Thread Cluster Memory Scheduling [Kim+ MICRO'10]

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



TCM: Throughput and Fairness

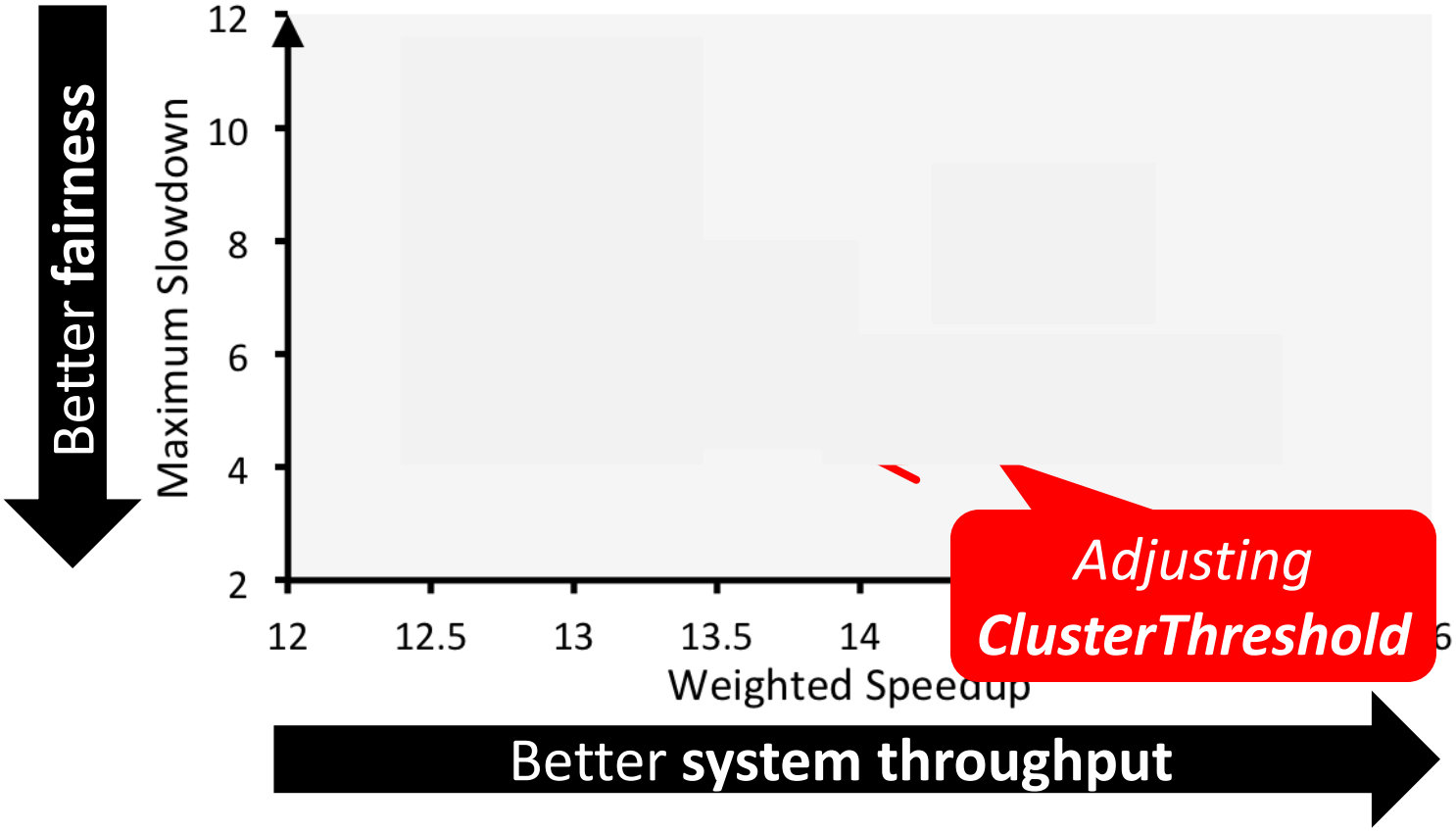
24 cores, 4 memory controllers, 96 workloads



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

TCM: Fairness-Throughput Tradeoff

When configuration parameter is varied…

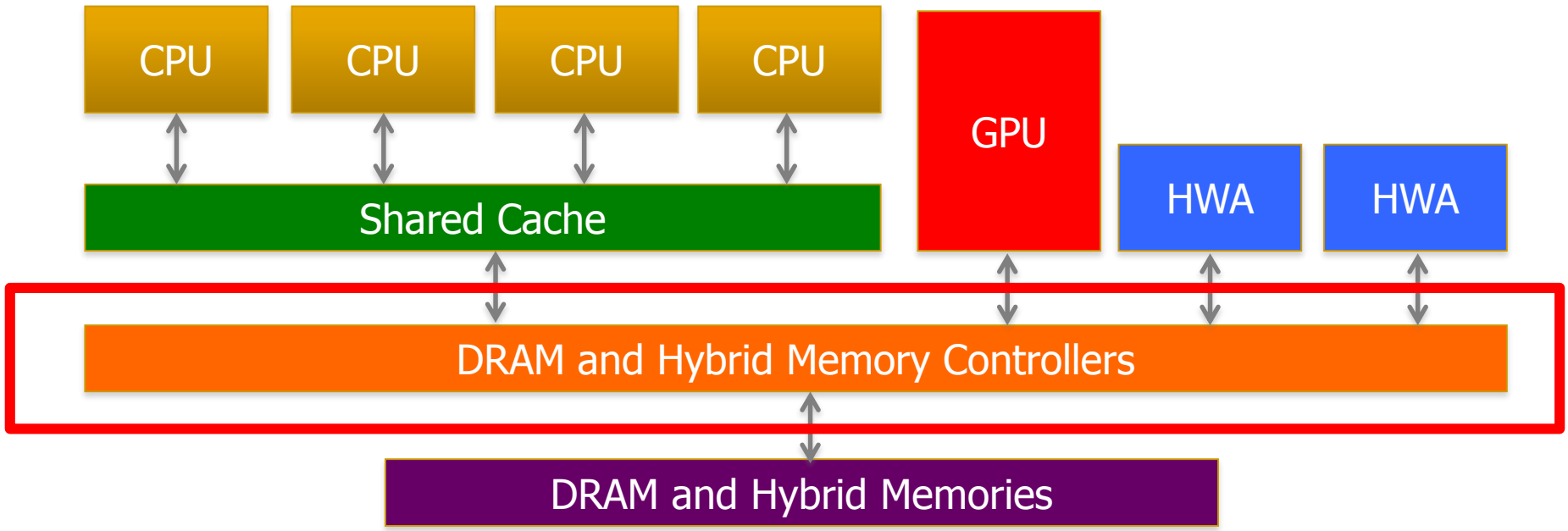


TCM allows robust fairness-throughput tradeoff

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] [Kim+, RTAS'14]
  - 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]

Predictable Performance in Complex Systems



- Heterogeneous agents: CPUs, GPUs, and HWAs
- Main memory interference between CPUs, GPUs, HWAs

How to allocate resources to heterogeneous agents to mitigate interference and provide predictable performance?

Strong Memory Service Guarantees

- Goal: Satisfy performance/SLA requirements in the presence of shared main memory, prefetchers, heterogeneous agents, and hybrid memory/storage

- Approach:
  - Develop techniques/models to accurately estimate the performance of an application/agent in the presence of resource sharing
  - Develop mechanisms (hardware and software) to enable the resource partitioning/prioritization needed to achieve the required performance levels for all applications
  - All the while providing high system performance

- Example work: Subramanian et al., "MISE: Providing Performance Predictability and Improving Fairness in Shared Main Memory Systems," HPCA 2013.

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.

Some Current Directions

- New memory/storage + compute architectures
  - Rethinking DRAM and flash memory
  - Processing close to data; accelerating bulk operations
  - Ensuring memory/storage reliability and robustness
- Enabling emerging NVM technologies
  - Hybrid memory systems with automatic data management
  - Coordinated management of memory and storage with NVM
- System-level memory/storage QoS
  - QoS-aware controller and system design
  - Coordinated memory + storage QoS

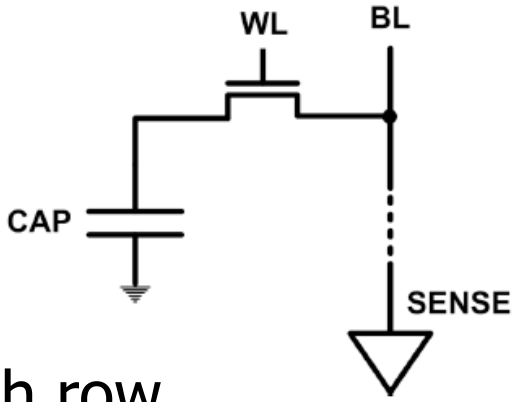
Agenda

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

Tolerating DRAM: Example Techniques

- Retention-Aware DRAM Refresh: Reducing Refresh Impact
- Refresh Access Parallelization: Reducing Refresh Impact
- Tiered-Latency DRAM: Reducing DRAM Latency
- RowClone: Accelerating Page Copy and Initialization
- Subarray-Level Parallelism: Reducing Bank Conflict Impact
- Base-Delta-Immediate Compression and Linearly Compressed Pages: Efficient Cache & Memory Compression

DRAM Refresh



- DRAM capacitor charge leaks over time

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

- Downsides of refresh

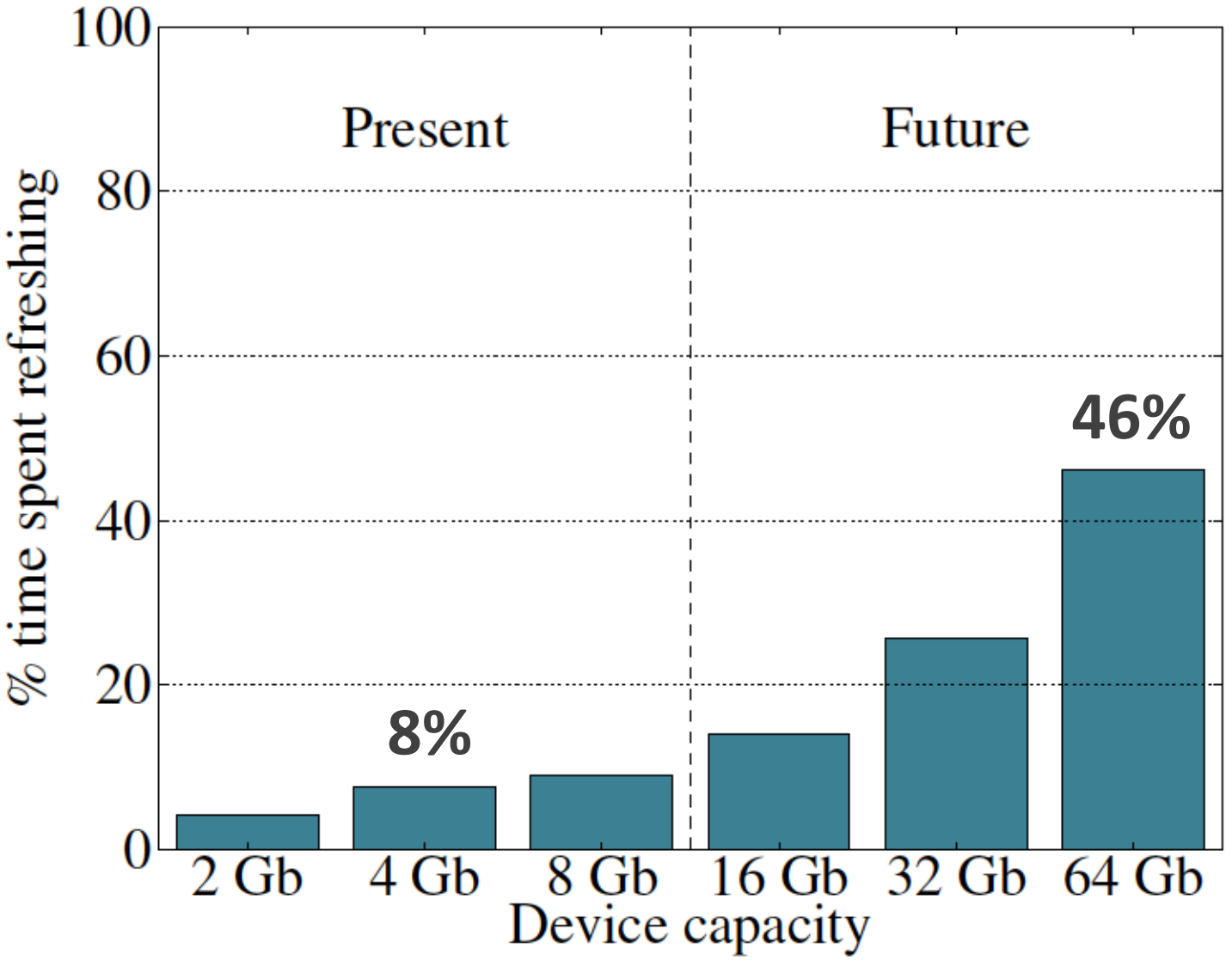
  -- Energy consumption: Each refresh consumes energy

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

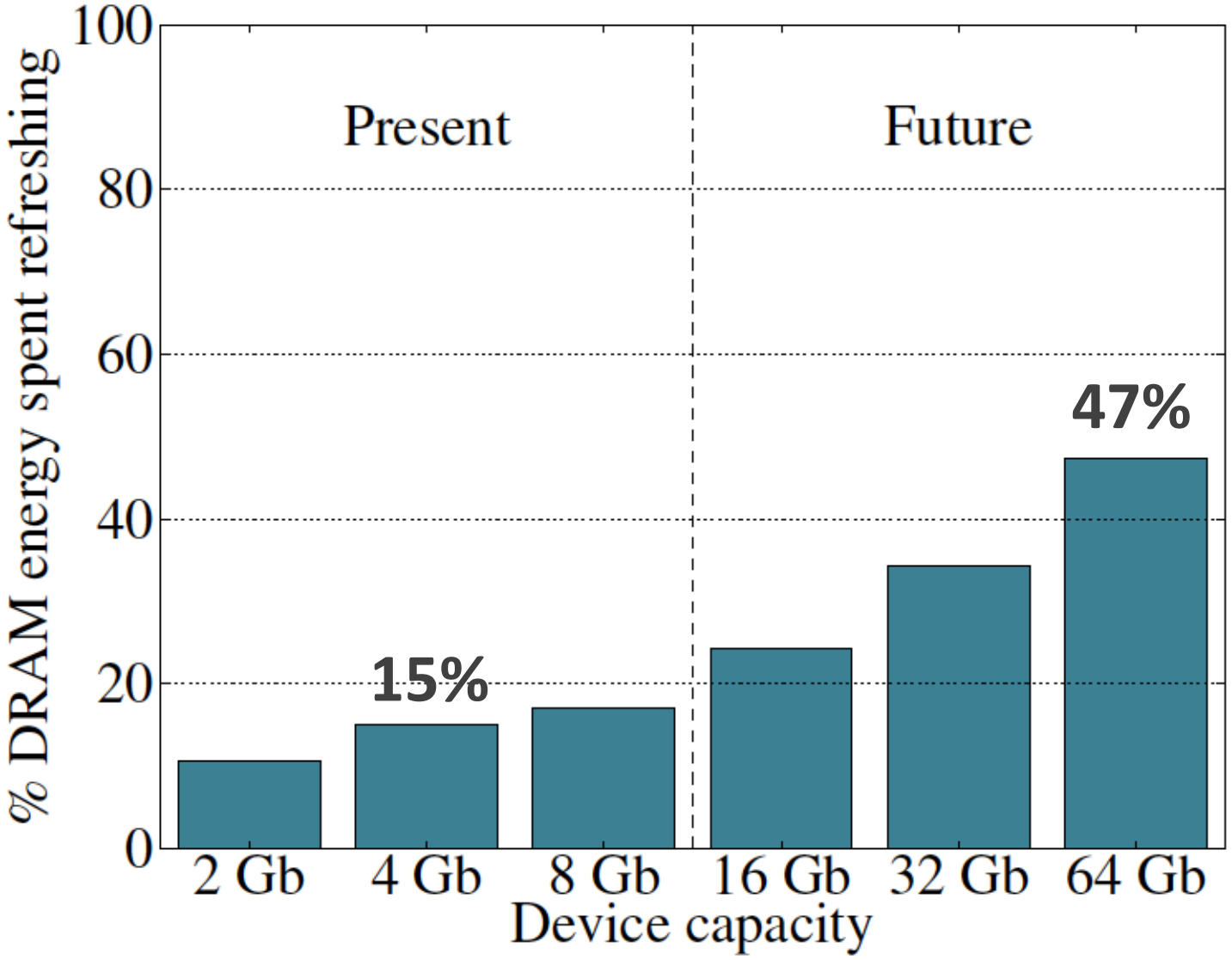
  -- QoS/predictability impact: (Long) pause times during refresh

  -- Refresh rate limits DRAM capacity scaling

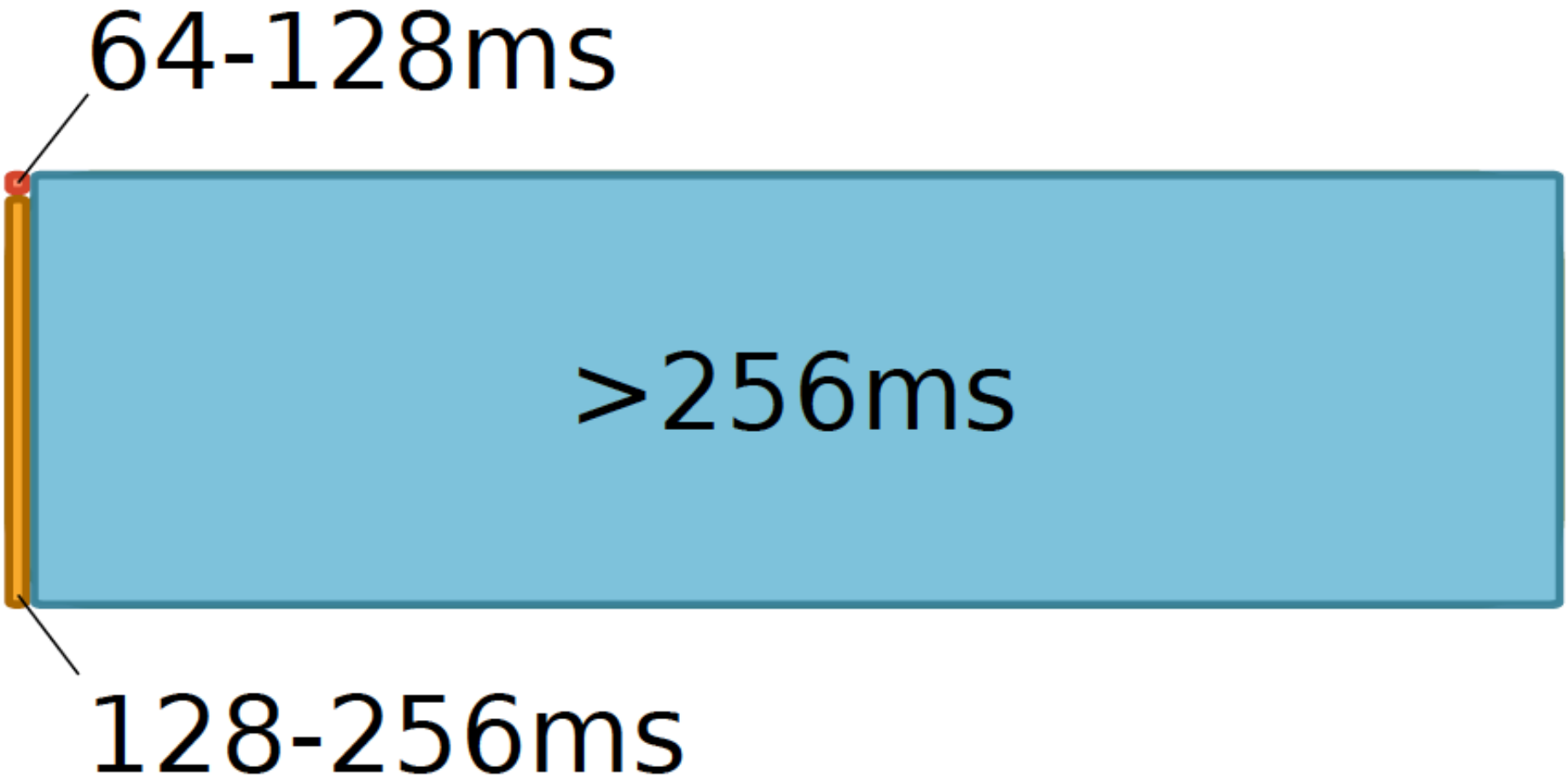
Refresh Overhead: Performance



Refresh Overhead: Energy

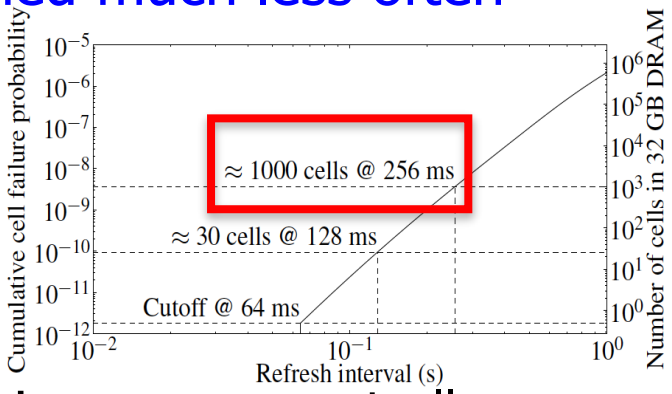


Retention Time Profile of DRAM



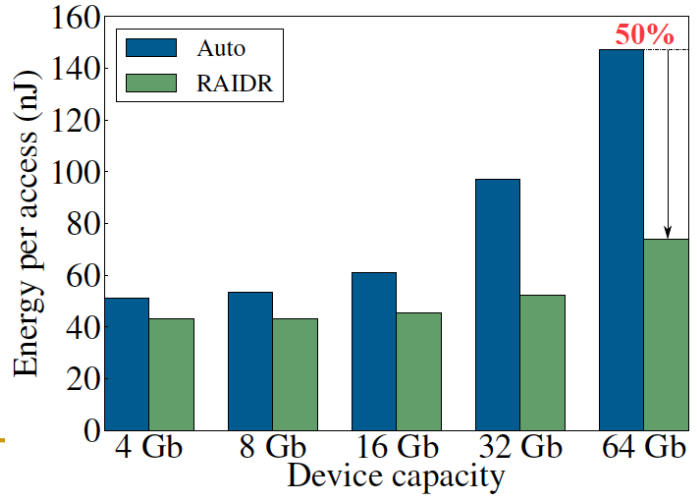
RAIDR: Eliminating Unnecessary Refreshes

- Observation: Most DRAM rows can be refreshed much less often without losing data **[Kim+, EDL'09][Liu+ ISCA'13]**



- 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
  - Benefits increase with DRAM capacity



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

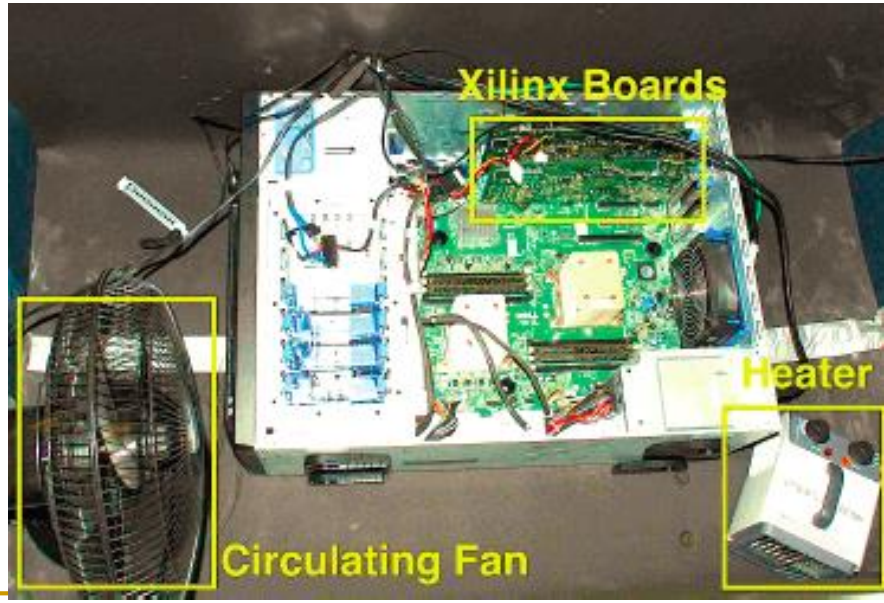
Going Forward (for DRAM and Flash)

- How to find out and expose weak memory cells/rows
  - Liu+, "An Experimental Study of Data Retention Behavior in Modern DRAM Devices: Implications for Retention Time Profiling Mechanisms", ISCA 2013.
  - Khan+, "The Efficacy of Error Mitigation Techniques for DRAM Retention Failures: A Comparative Experimental Study," SIGMETRICS 2014.
- Low-cost system-level tolerance of memory errors
  - Luo+, "Characterizing Application Memory Error Vulnerability to Optimize Data Center Cost," DSN 2014.
  - Cai+, "Error Analysis and Retention-Aware Error Management for NAND Flash Memory," Intel Technology Journal 2013.
  - Cai+, "Neighbor-Cell Assisted Error Correction for MLC NAND Flash Memories," SIGMETRICS 2014.
- Tolerating cell-to-cell interference at the system level
  - Kim+, "Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors," ISCA 2014.
  - Cai+, "Program Interference in MLC NAND Flash Memory: Characterization, Modeling, and Mitigation," ICCD 2013.

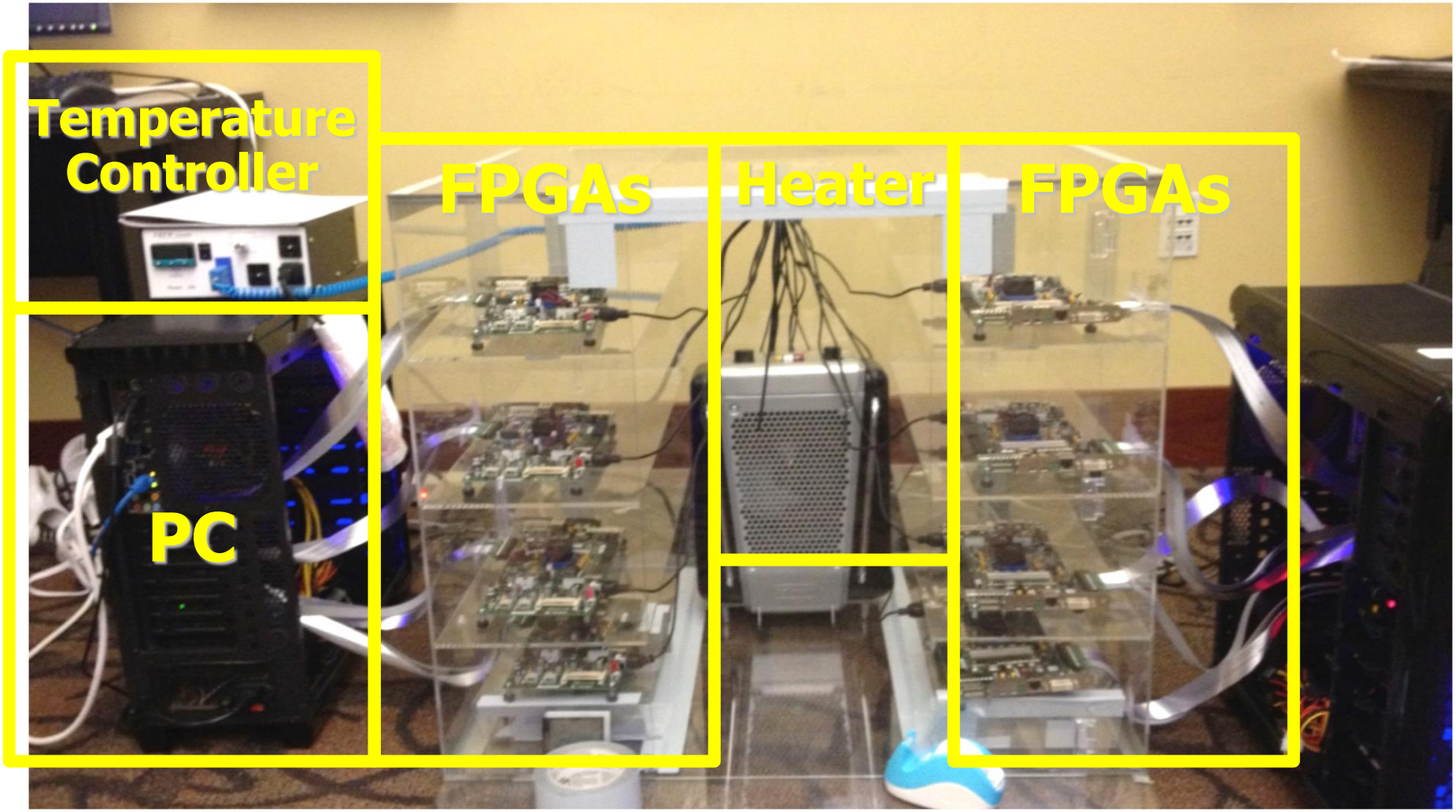
Experimental Infrastructure (DRAM)

Liu+, "An Experimental Study of Data Retention Behavior in Modern DRAM Devices: Implications for Retention Time Profiling Mechanisms", ISCA 2013.

Khan+, "The Efficacy of Error Mitigation Techniques for DRAM Retention Failures: A Comparative Experimental Study," SIGMETRICS 2014.

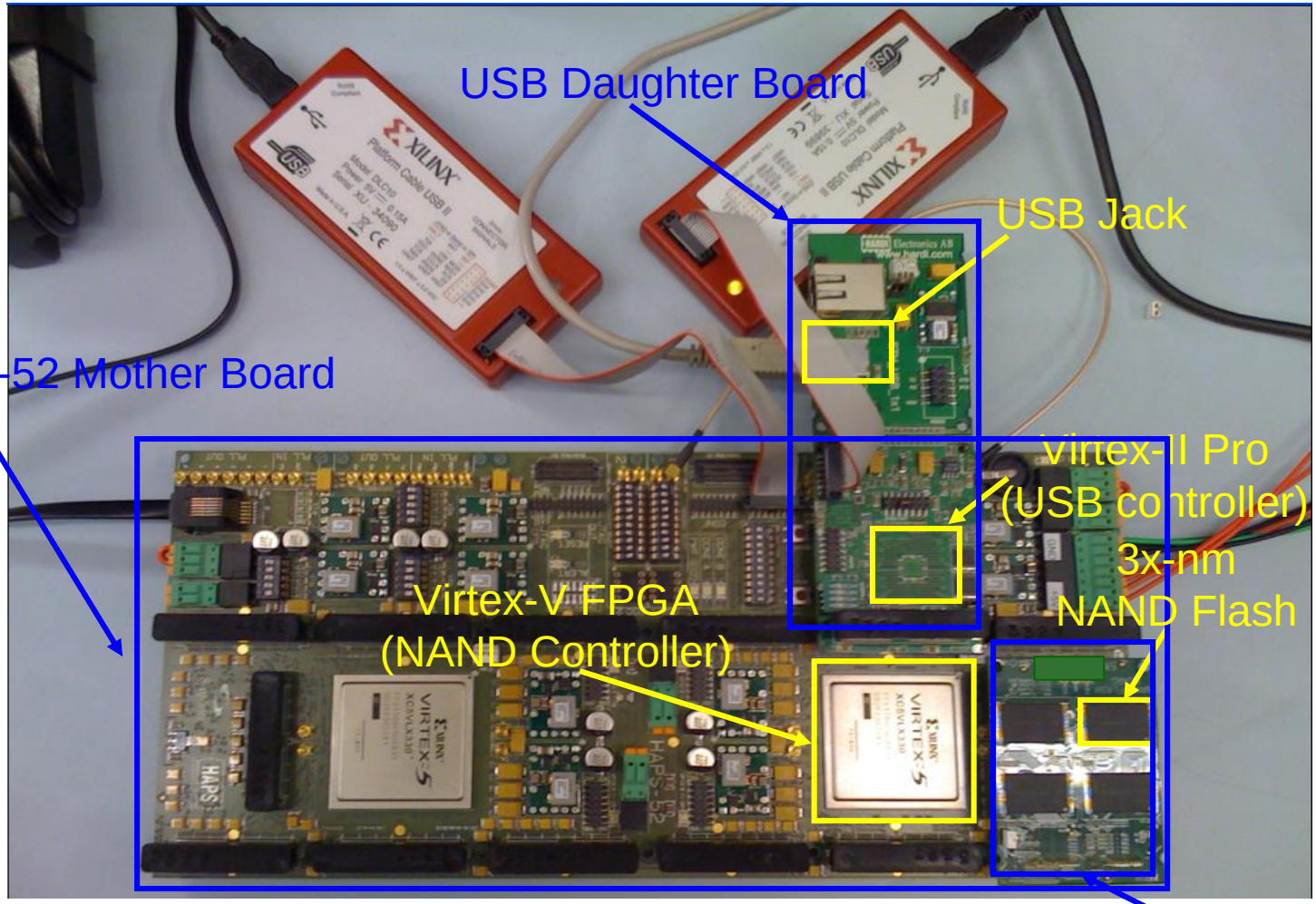


Experimental Infrastructure (DRAM)



Kim+, "Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors," ISCA 2014.

Experimental Infrastructure (Flash)



[Cai+, DATE 2012, ICCD 2012, DATE 2013, ITJ 2013, ICCD 2013, SIGMETRICS 2014]

Another Talk: NAND Flash Scaling Challenges

- Cai+, "Error Patterns in MLC NAND Flash Memory: Measurement, Characterization, and Analysis," DATE 2012.
- Cai+, "Flash Correct-and-Refresh: Retention-Aware Error Management for Increased Flash Memory Lifetime," ICCD 2012.
- Cai+, "Threshold Voltage Distribution in MLC NAND Flash Memory: Characterization, Analysis and Modeling," DATE 2013.
- Cai+, "Error Analysis and Retention-Aware Error Management for NAND Flash Memory," Intel Tech Journal 2013.
- Cai+, "Program Interference in MLC NAND Flash Memory: Characterization, Modeling, and Mitigation," ICCD 2013.
- Cai+, "Neighbor-Cell Assisted Error Correction for MLC NAND Flash Memories," SIGMETRICS 2014.

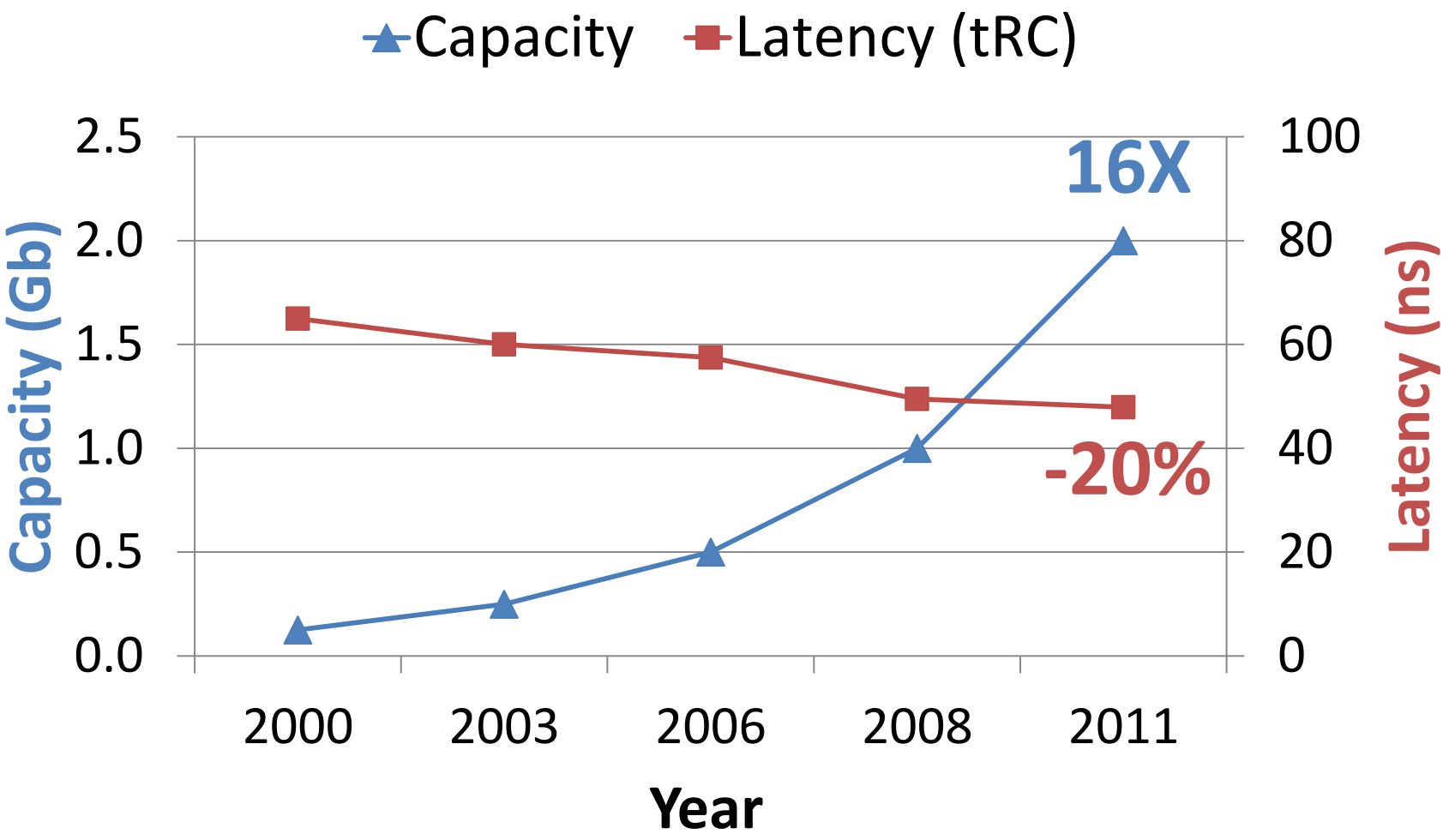
Error Management in MLC NAND Flash

- Problem: MLC NAND flash memory reliability/endurance is a key challenge for satisfying future storage systems' requirements
- Our Goals: (1) Build reliable error models for NAND flash memory via experimental characterization, (2) Develop efficient techniques to improve reliability and endurance
- This talk provides a "flash" summary of our recent results published in the past 3 years:
  - Experimental error and threshold voltage characterization **[DATE'12&13]**
  - Retention-aware error management **[ICCD'12]**
  - Program interference analysis and read reference V prediction **[ICCD'13]**
  - Neighbor-assisted error correction **[SIGMETRICS'14]**

Tolerating DRAM: Example Techniques

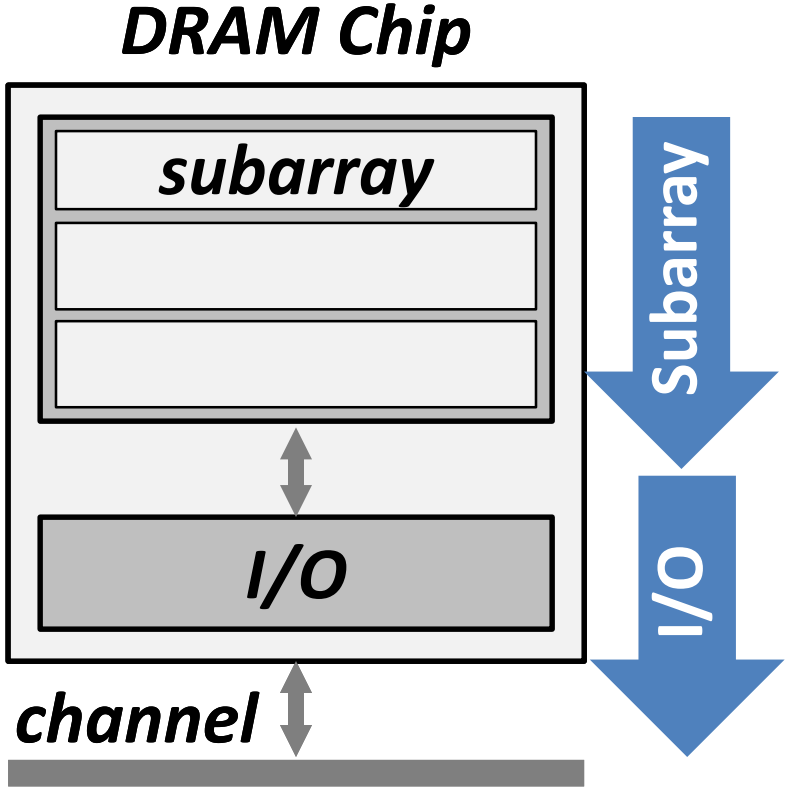
- Retention-Aware DRAM Refresh: Reducing Refresh Impact
- Refresh Access Parallelization: Reducing Refresh Impact
- Tiered-Latency DRAM: Reducing DRAM Latency
- RowClone: Accelerating Page Copy and Initialization
- Subarray-Level Parallelism: Reducing Bank Conflict Impact
- Base-Delta-Immediate Compression and Linearly Compressed Pages: Efficient Cache & Memory Compression

DRAM Latency-Capacity Trend



DRAM latency continues to be a critical bottleneck, especially for response time-sensitive

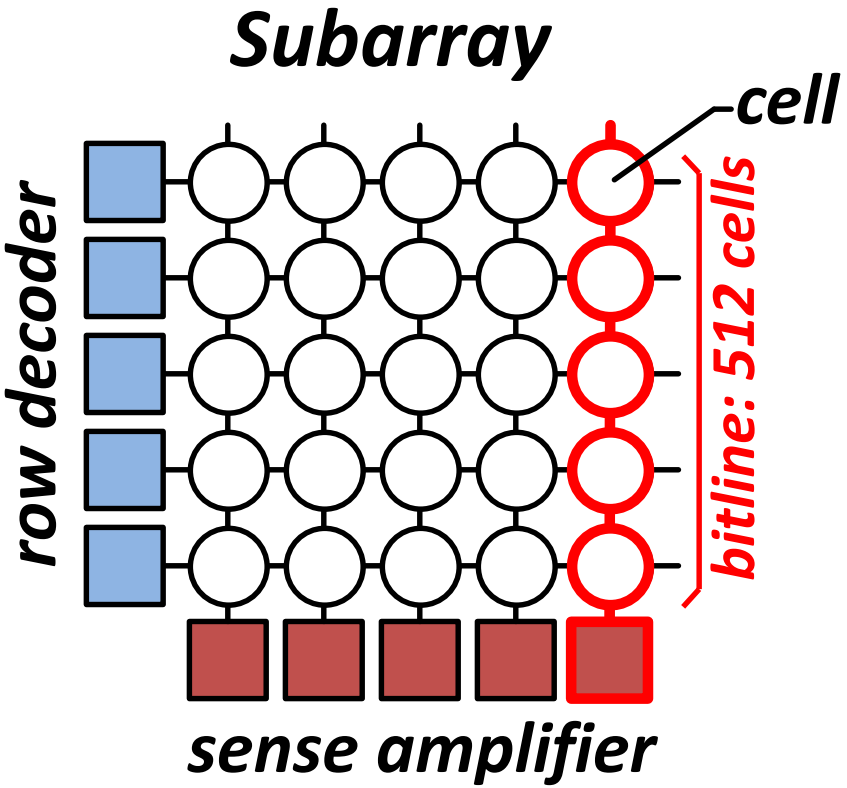
What Causes the Long Latency?

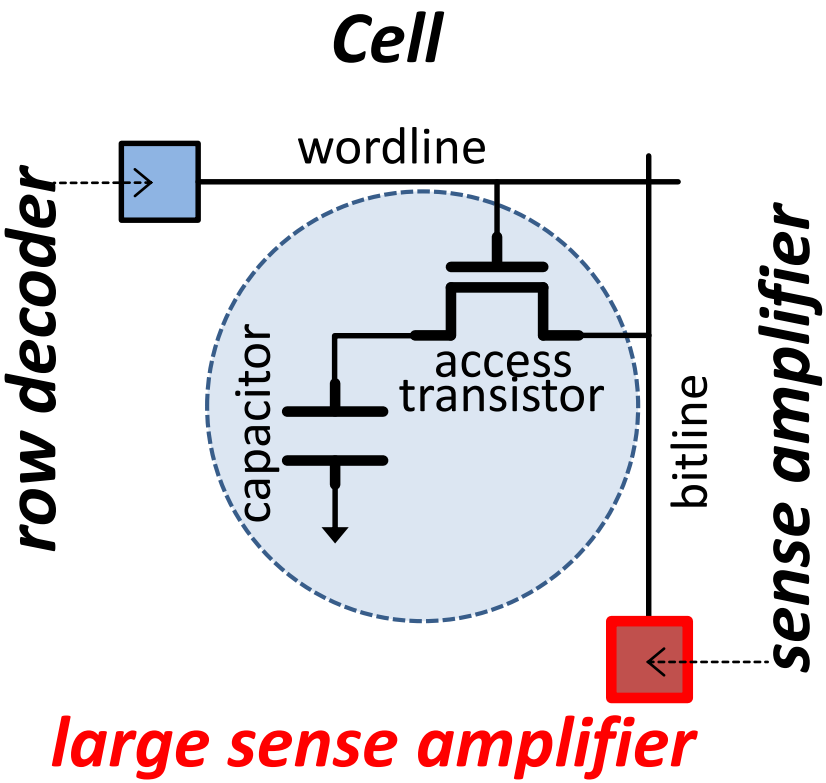


DRAM Latency = Subarray Latency + I/O Latency

Dominant

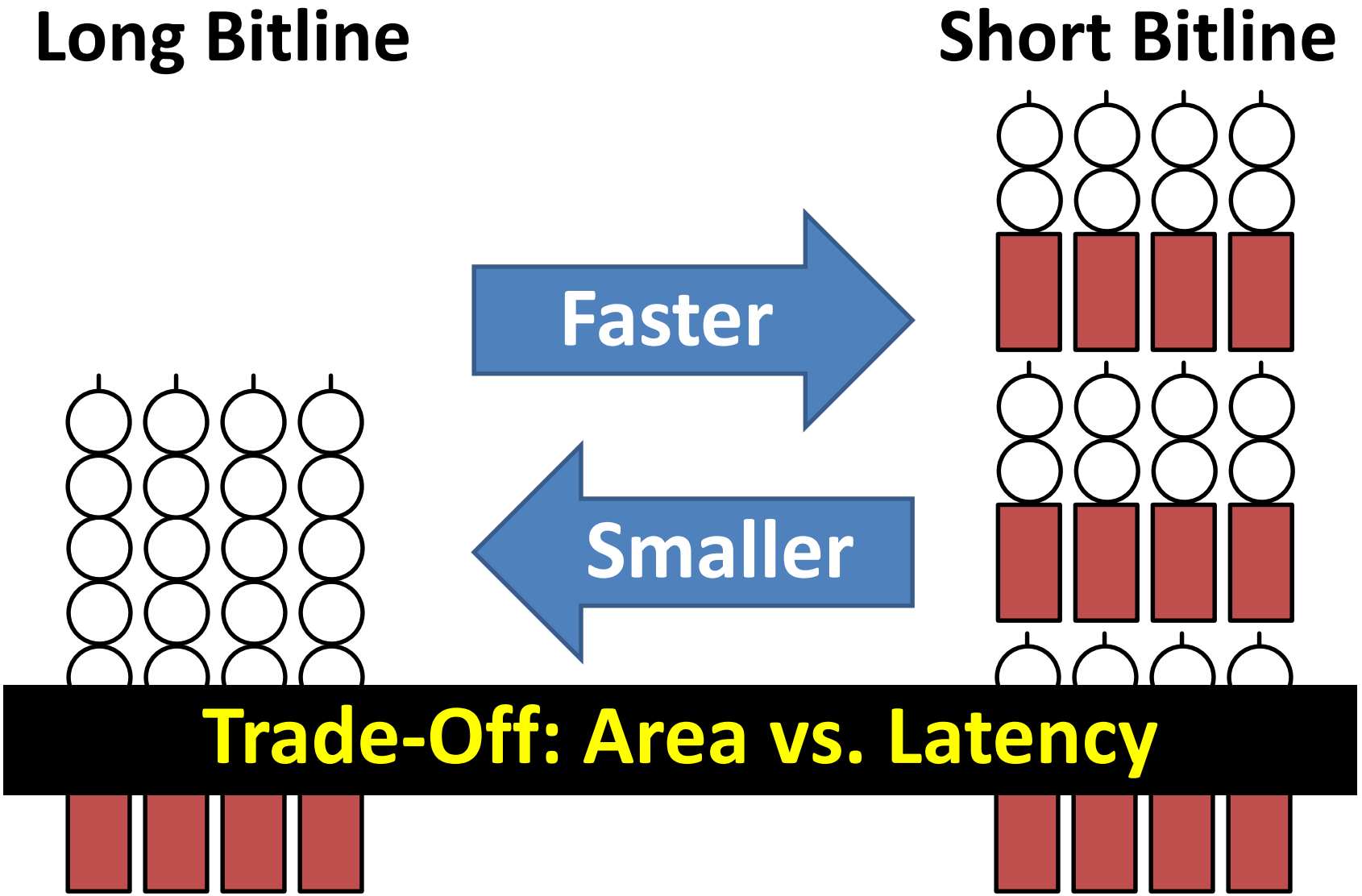
Why is the Subarray So Slow?



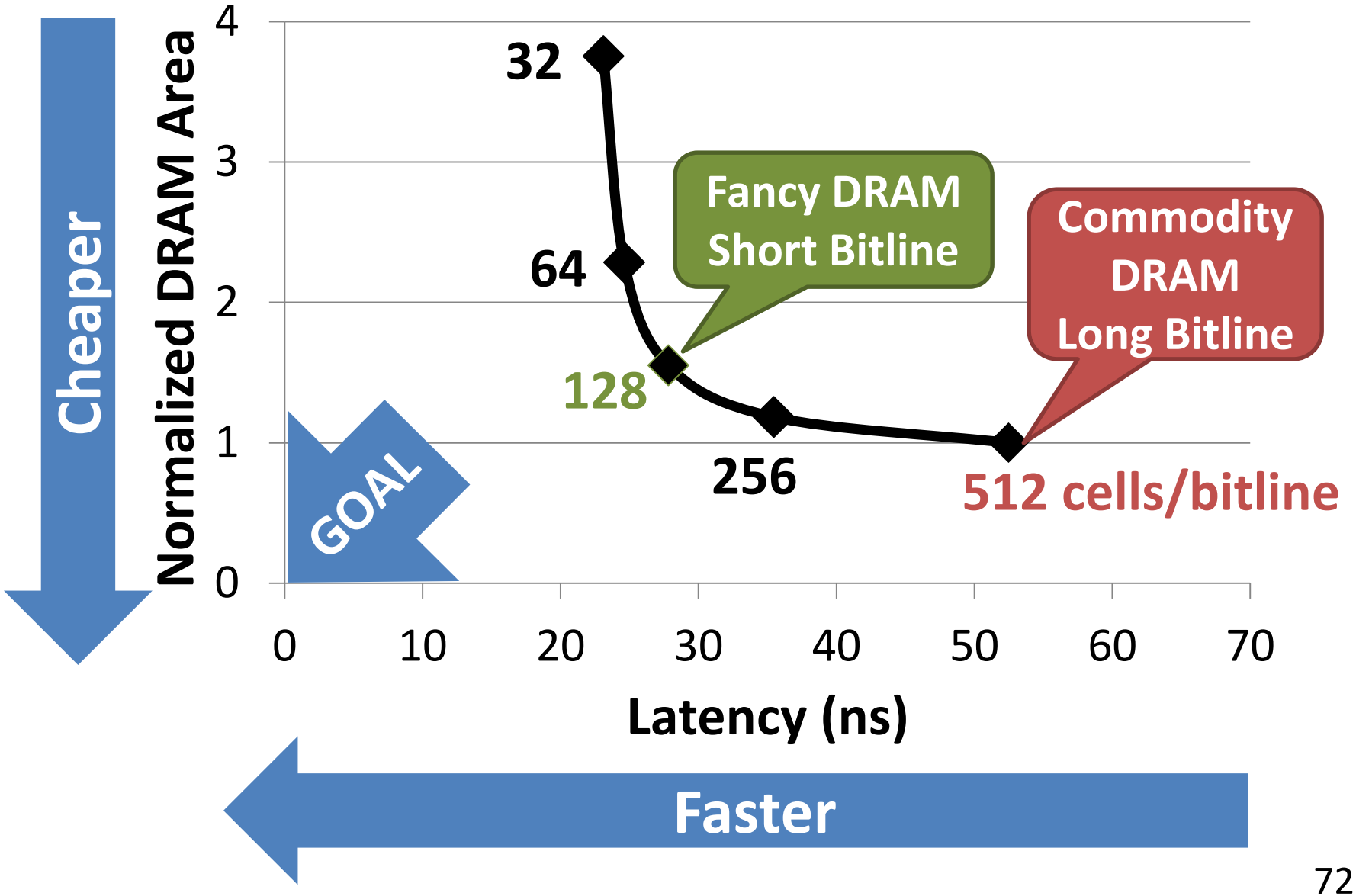


- **Long bitline**
  - **Amortizes sense amplifier cost → Small area**
  - **Large bitline capacitance → High latency & power**

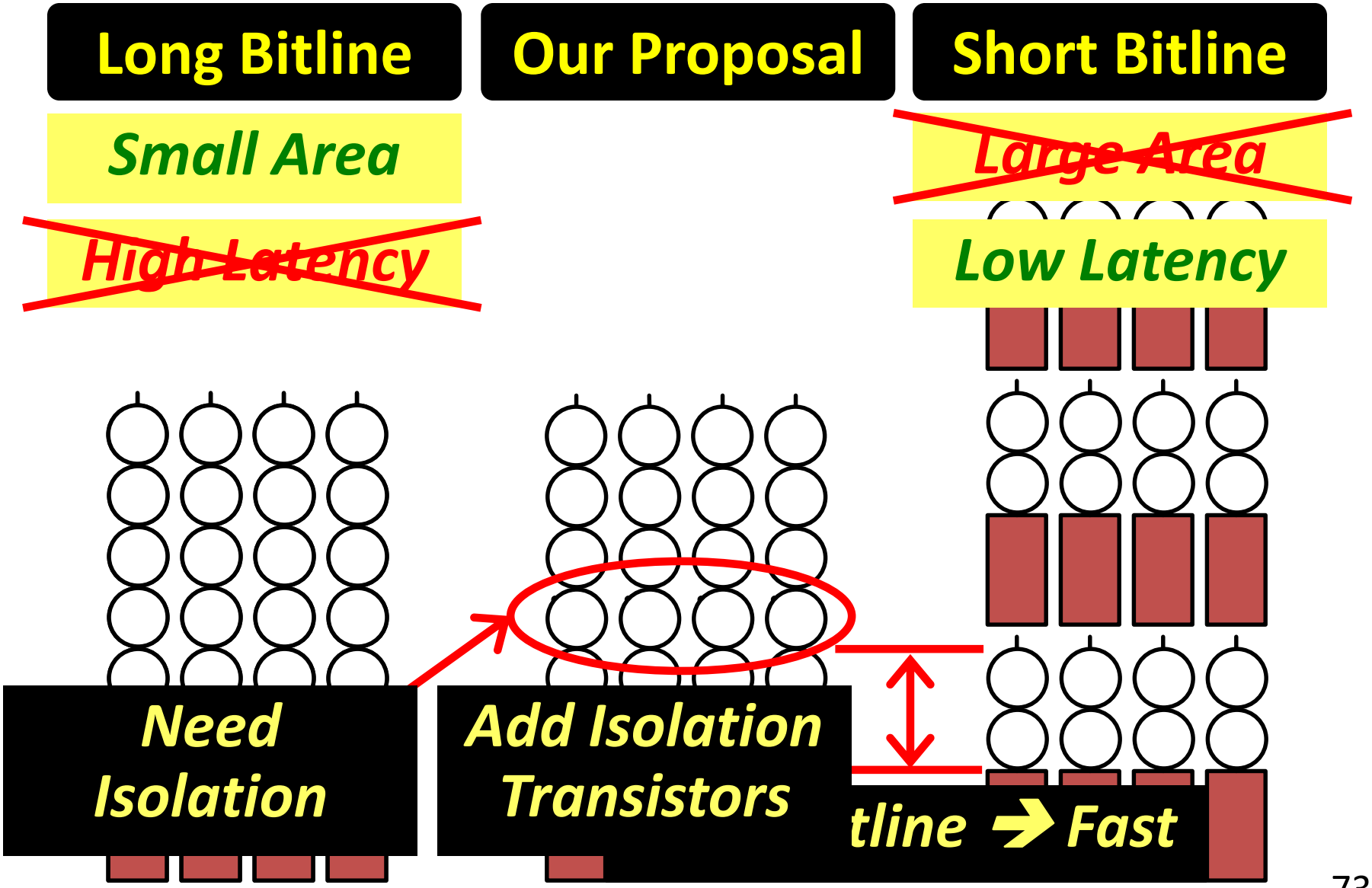
Trade-Off: Area (Die Size) vs. Latency



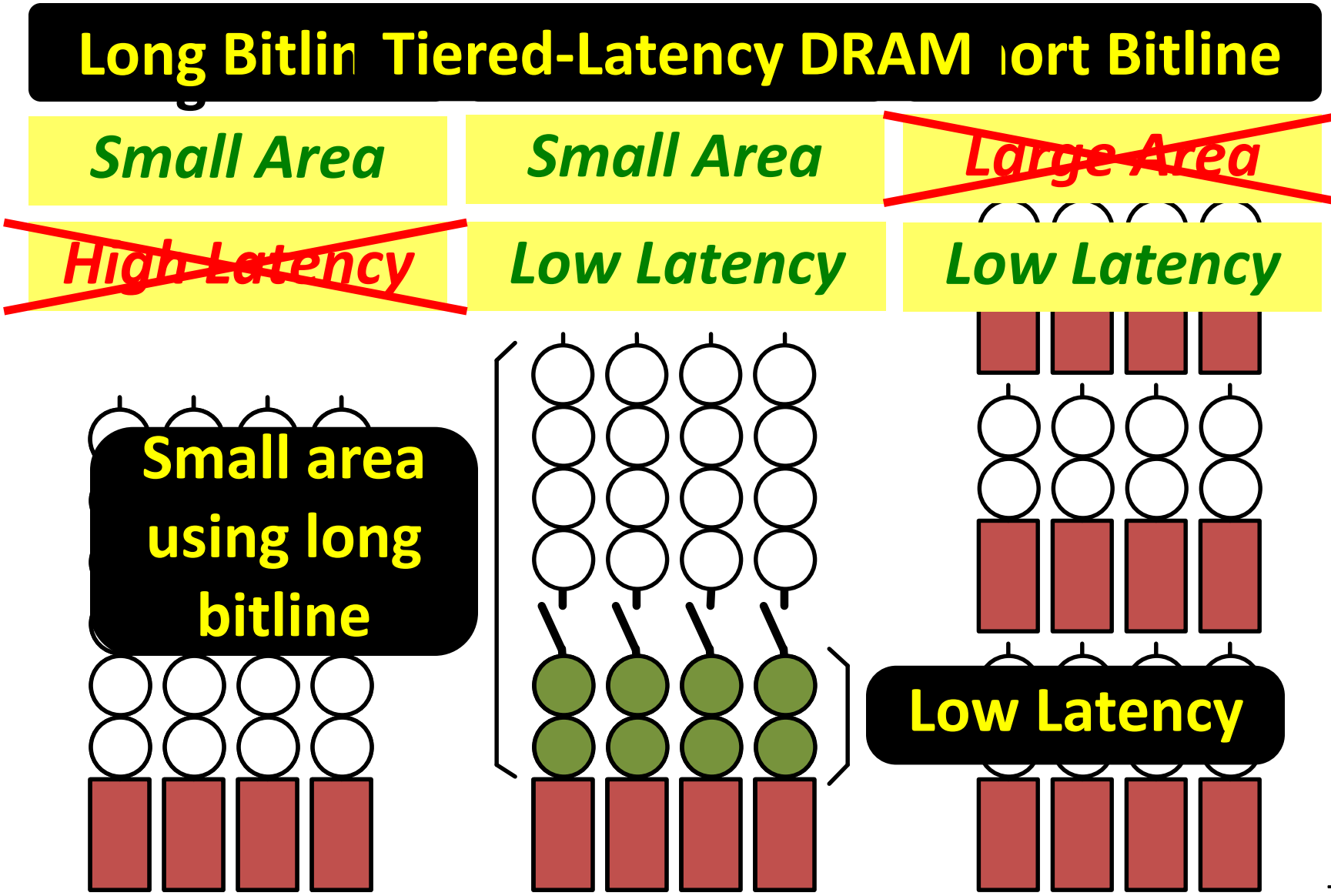
Trade-Off: Area (Die Size) vs. Latency



Approximating the Best of Both Worlds

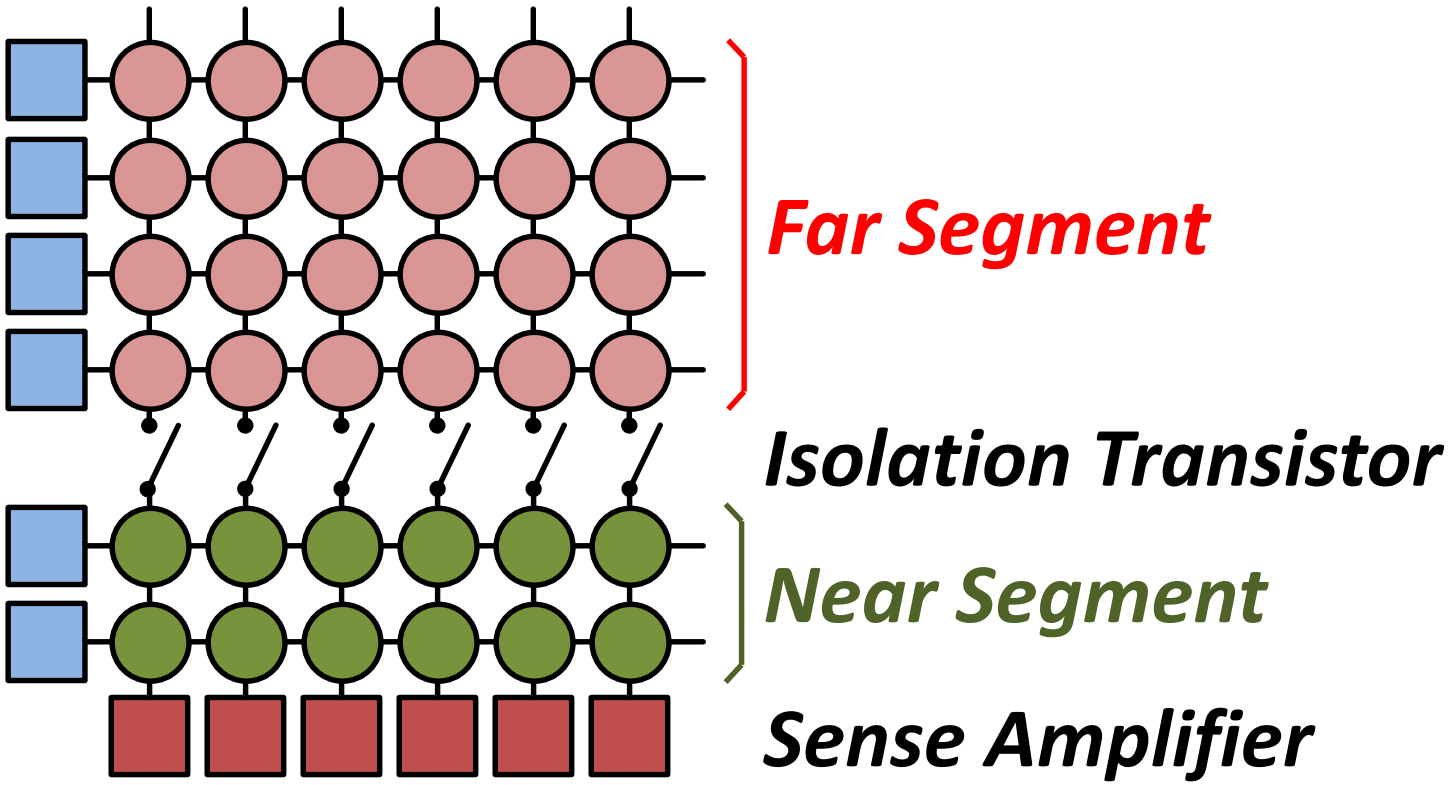


Approximating the Best of Both Worlds



Tiered-Latency DRAM

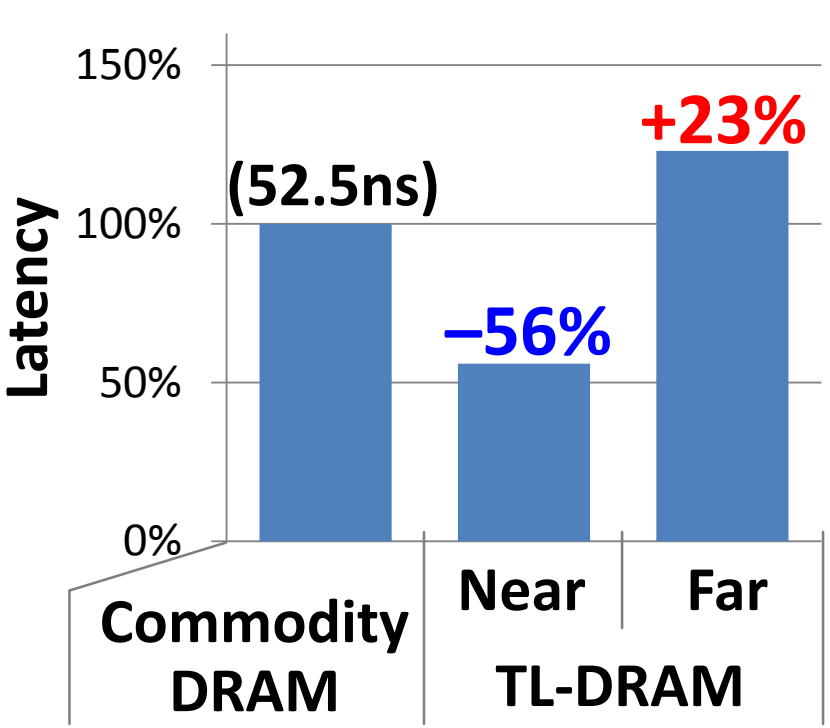
- Divide a bitline into two segments with an **isolation transistor**

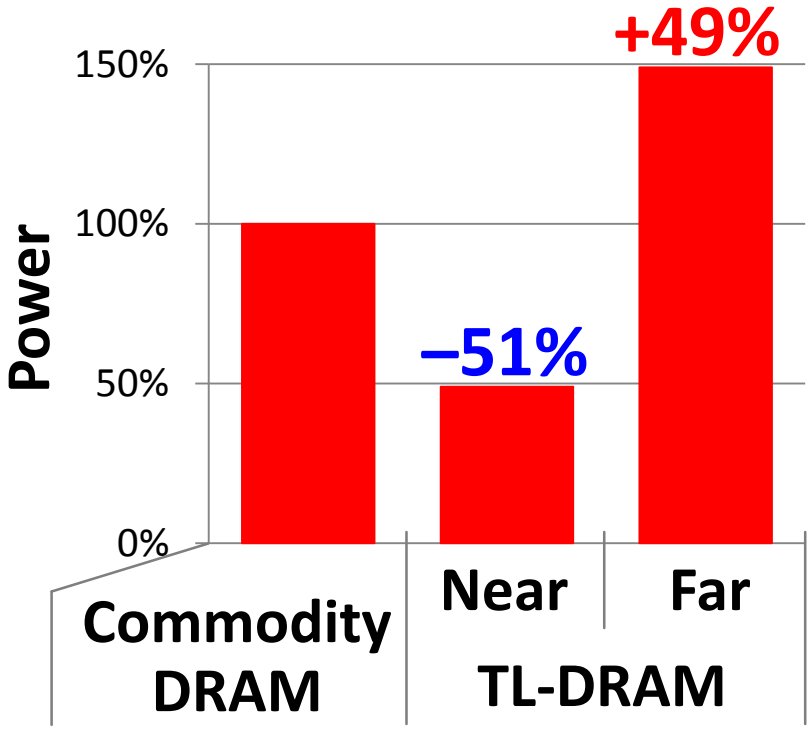


Lee+, "Tiered-Latency DRAM: A Low Latency and Low Cost DRAM Architecture," HPCA 2013.

Commodity DRAM vs. TL-DRAM

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

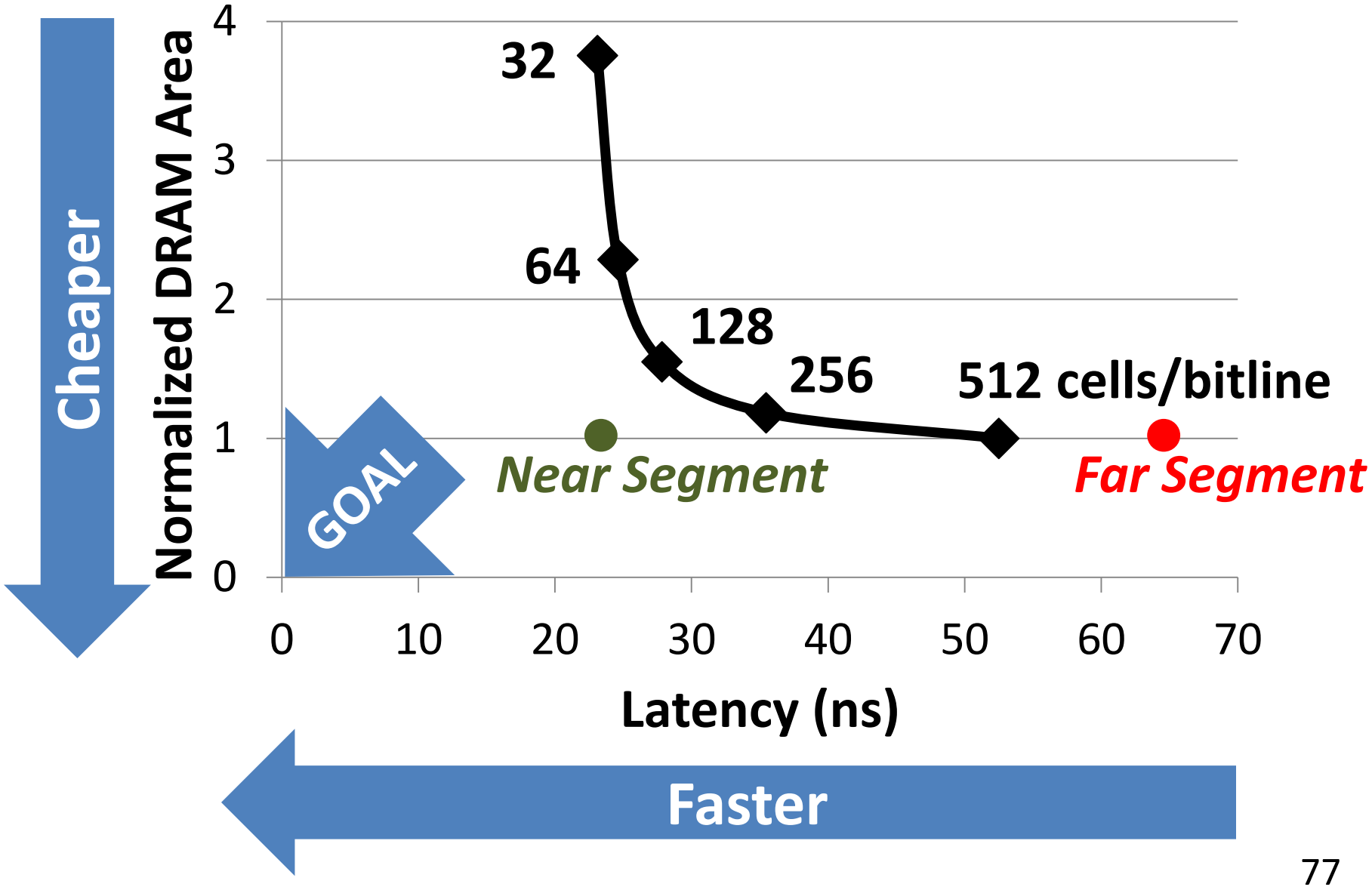




- **DRAM Area Overhead**

  ~3%: mainly due to the isolation transistors

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



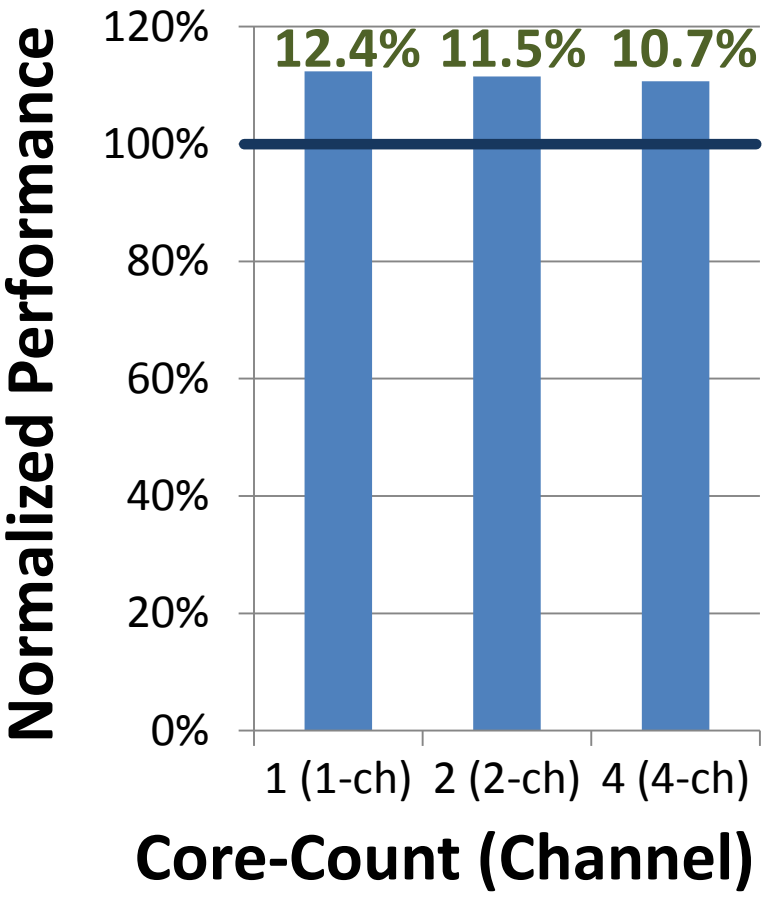
Leveraging Tiered-Latency DRAM

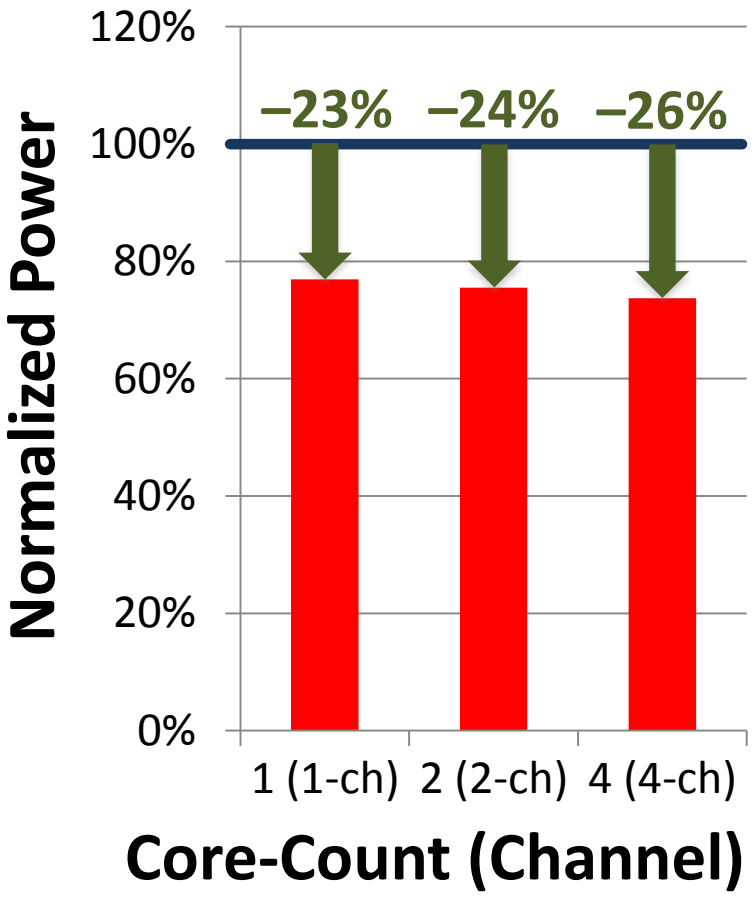
- TL-DRAM is a ***substrate*** that can be leveraged by the hardware and/or software

- 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



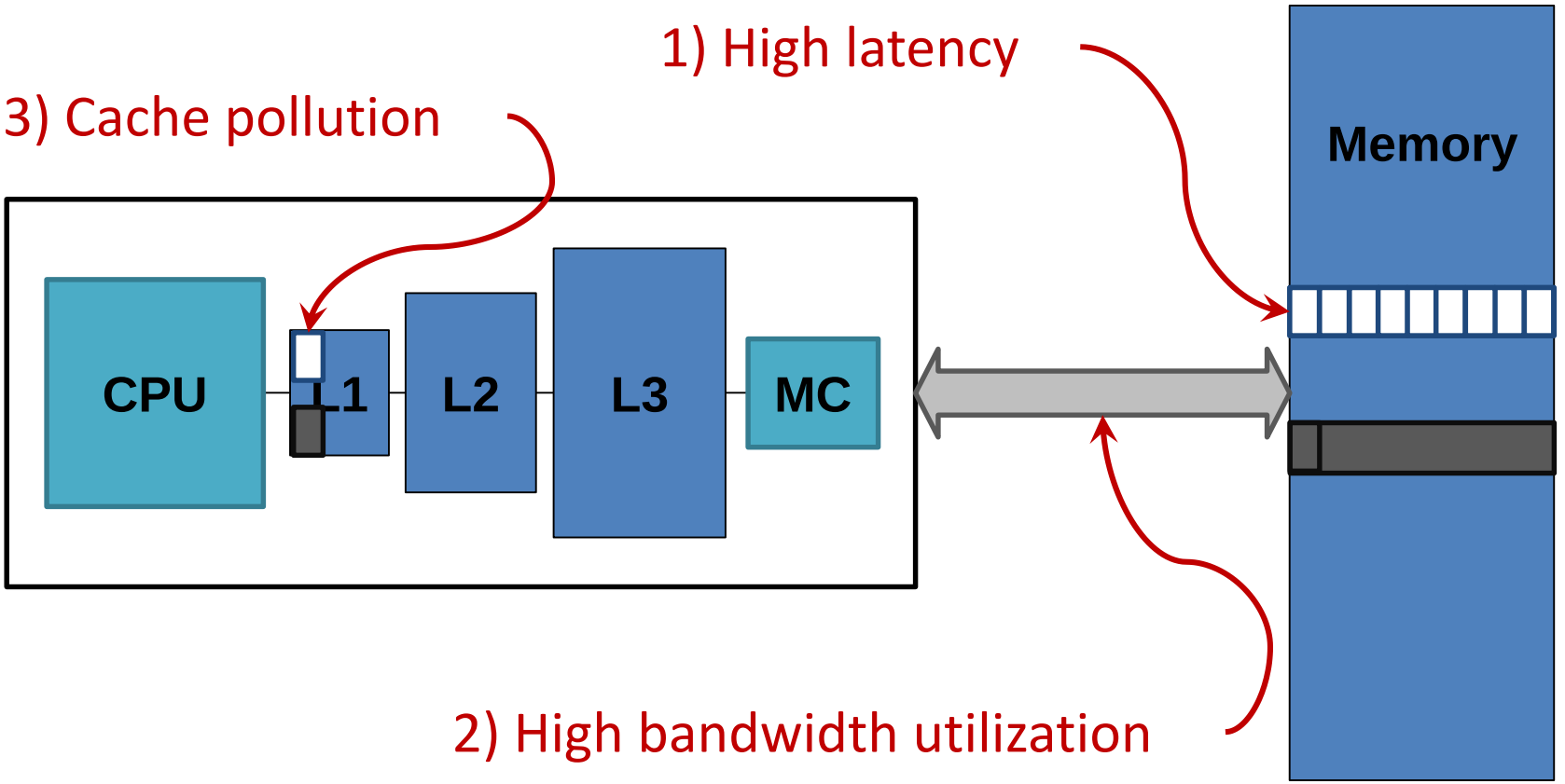


Using near segment as a cache improves performance and reduces power consumption

Tolerating DRAM: Example Techniques

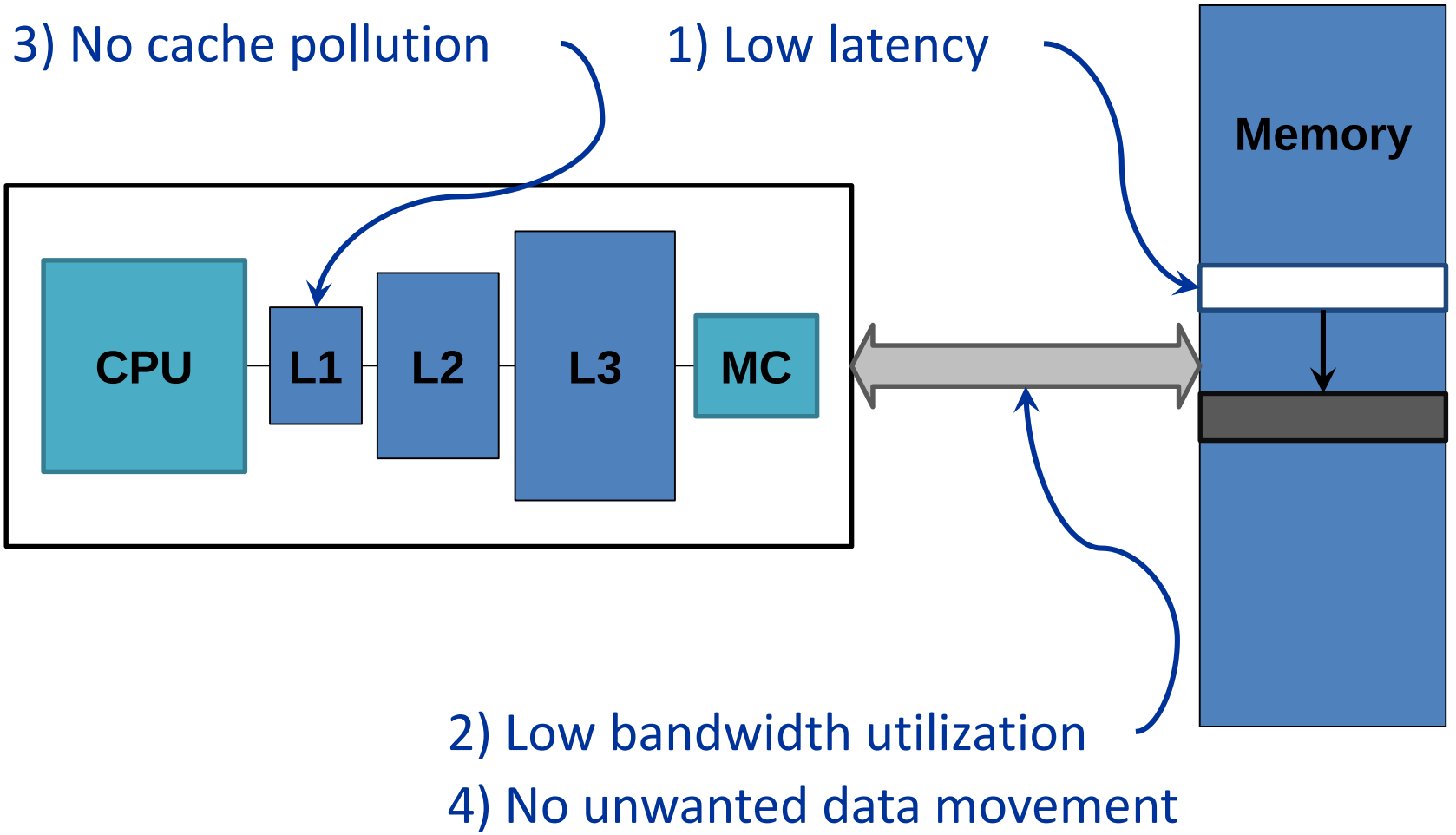
- Retention-Aware DRAM Refresh: Reducing Refresh Impact
- Refresh Access Parallelization: Reducing Refresh Impact
- Tiered-Latency DRAM: Reducing DRAM Latency
- RowClone: Accelerating Page Copy and Initialization
- Subarray-Level Parallelism: Reducing Bank Conflict Impact
- Base-Delta-Immediate Compression and Linearly Compressed Pages: Efficient Cache & Memory Compression

Today's Memory: Bulk Data Copy

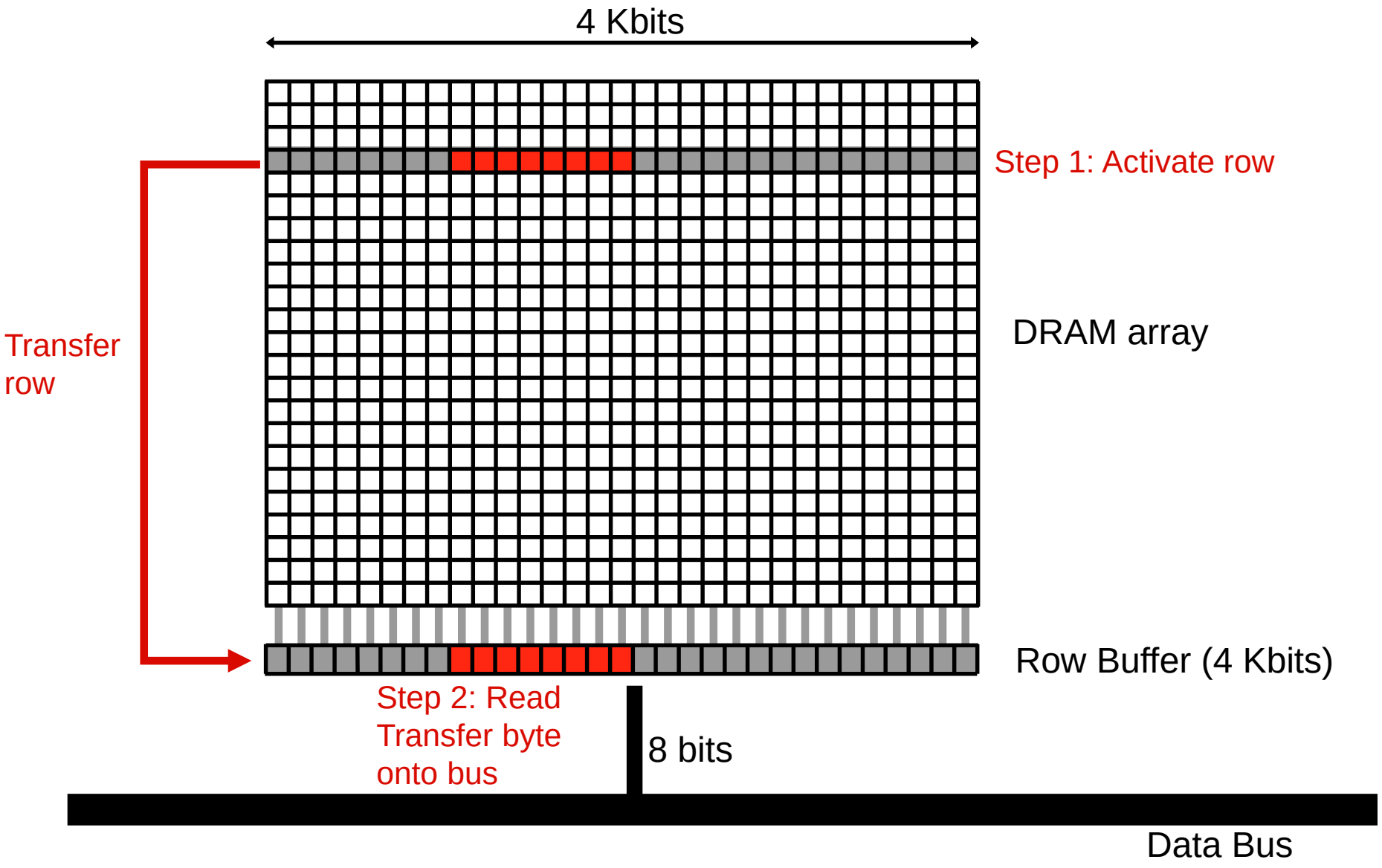


1046ns, 3.6uJ

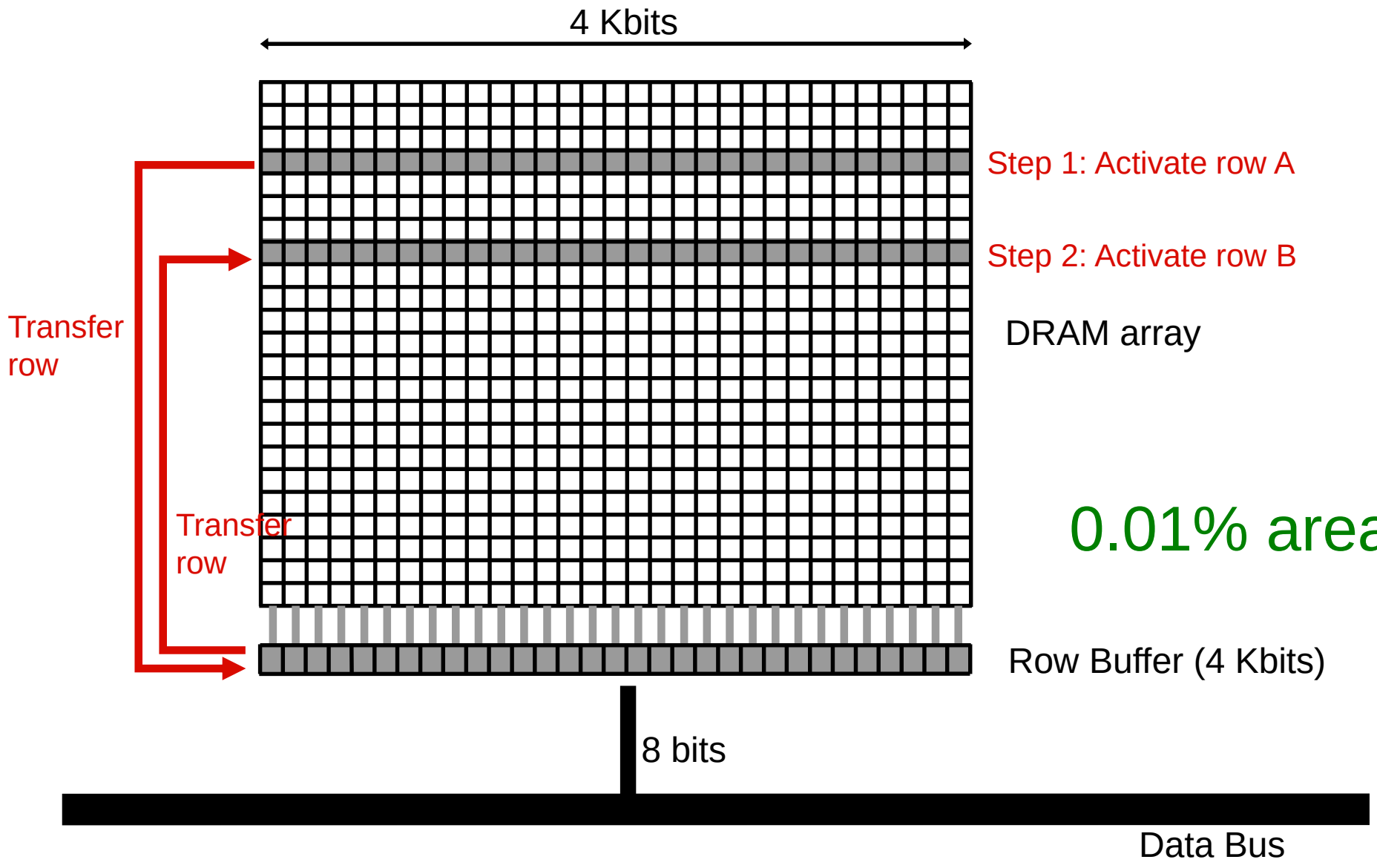
Future: RowClone (In-Memory Copy)



DRAM Subarray Operation (load one byte)

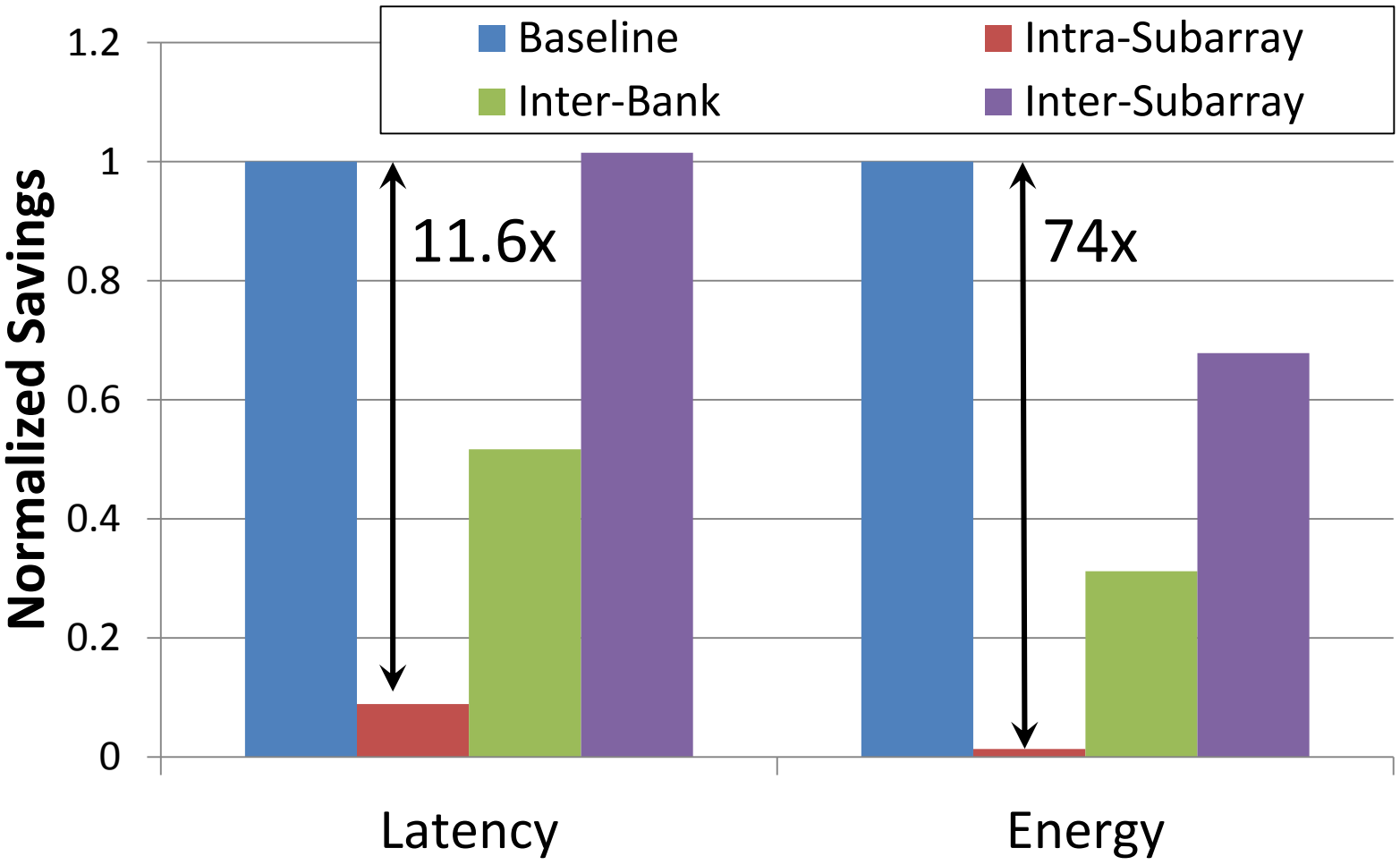


RowClone: In-DRAM Row Copy



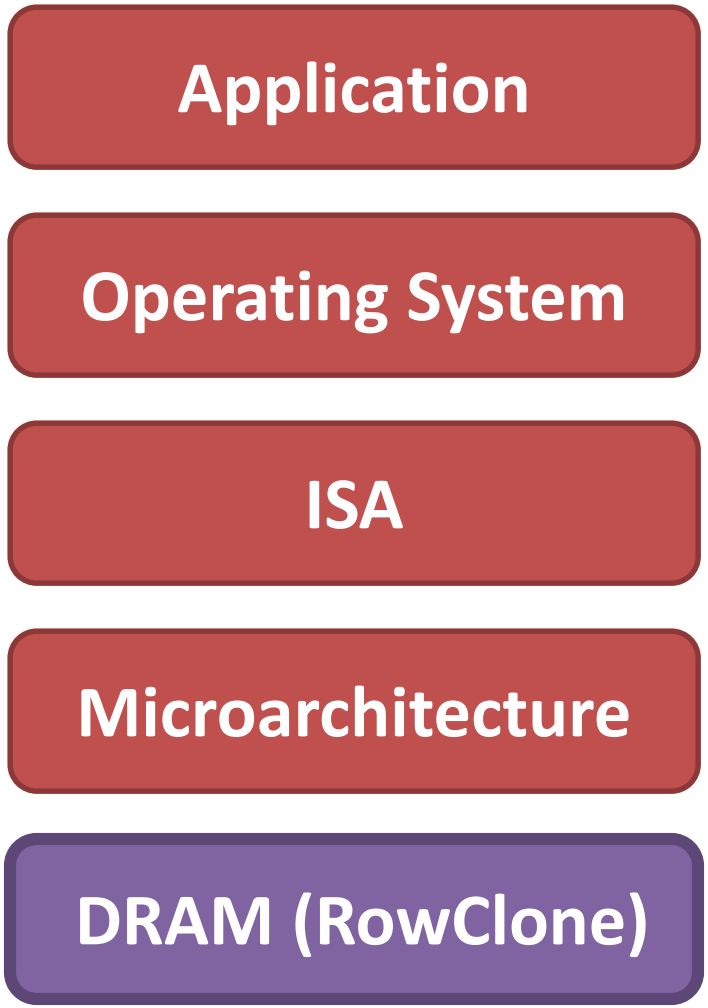
0.01% area

RowClone: Latency and Energy Savings



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

End-to-End System Design



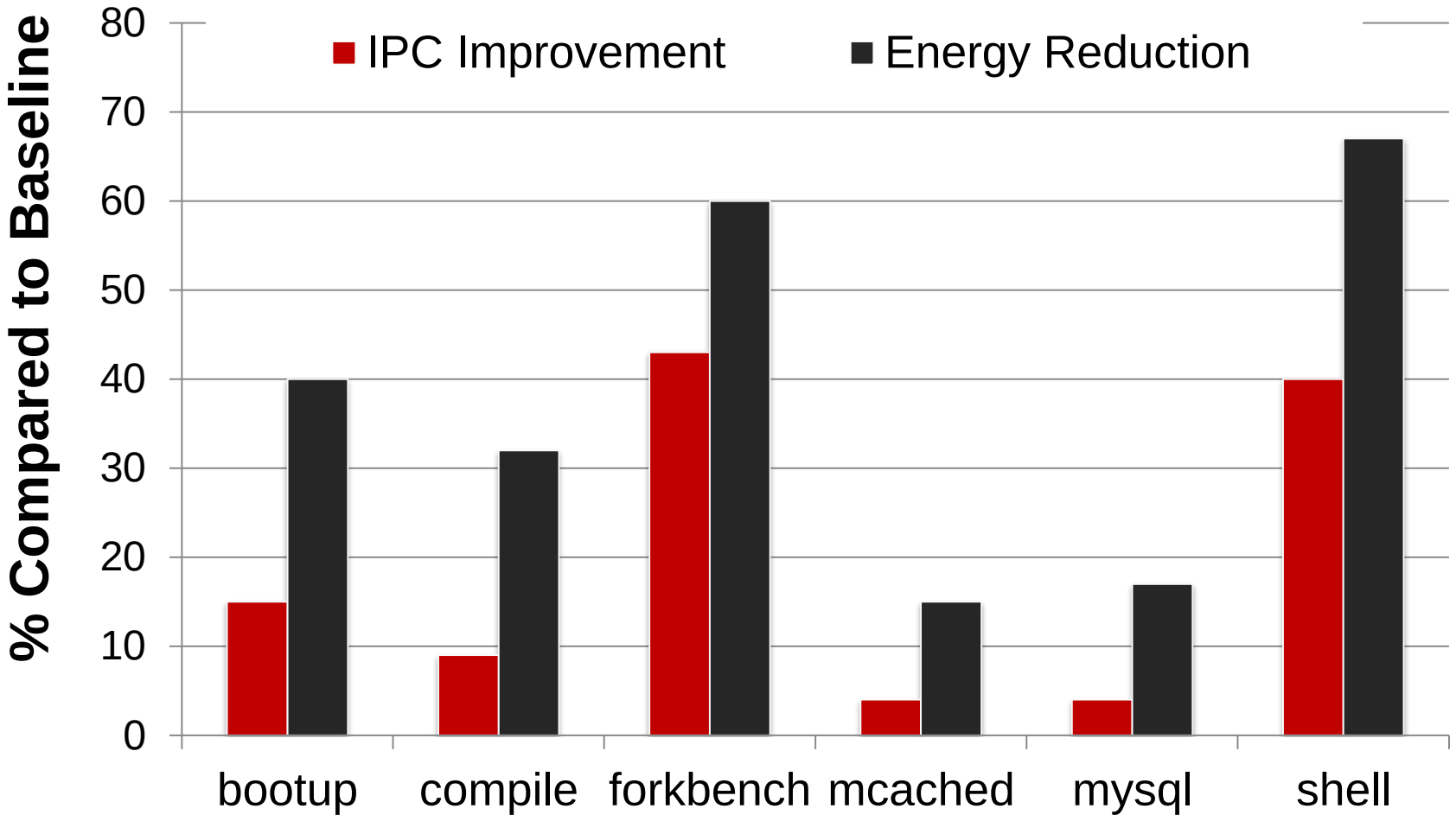
How does the software communicate occurrences of bulk copy/initialization to hardware?

How to ensure cache coherence?

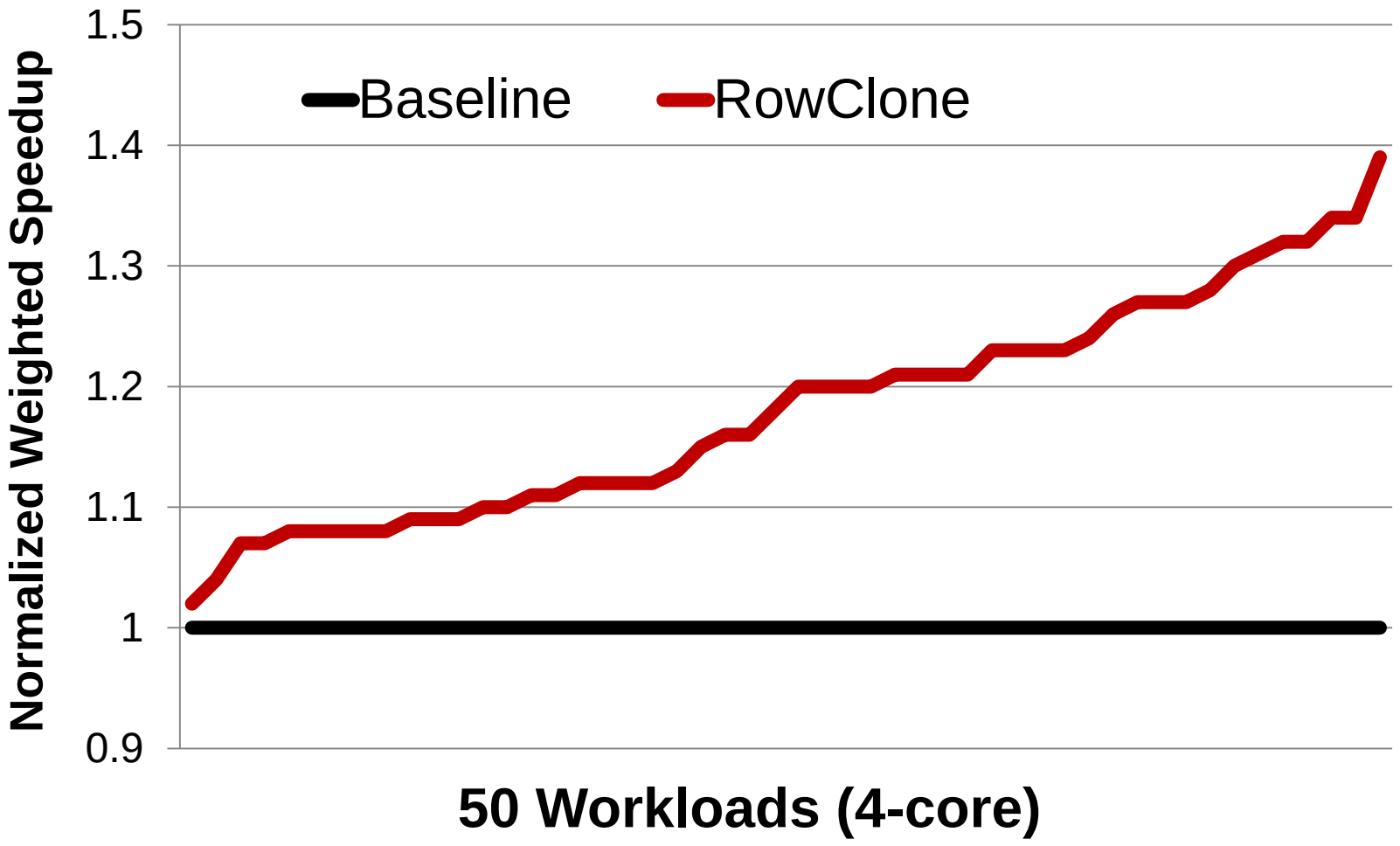
How to maximize latency and energy savings?

How to handle data reuse?

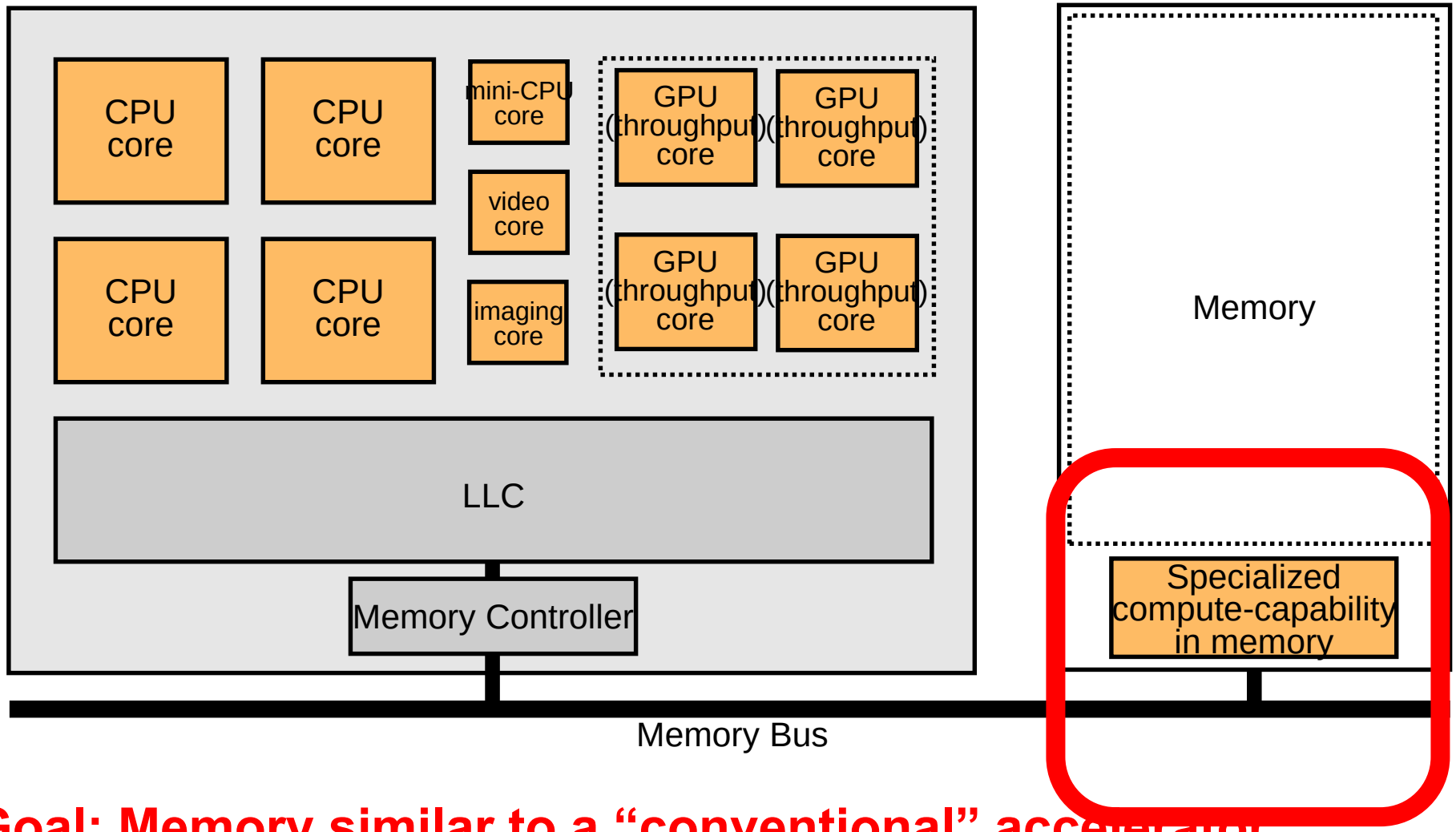
RowClone: Overall Performance



RowClone: Multi-Core Performance

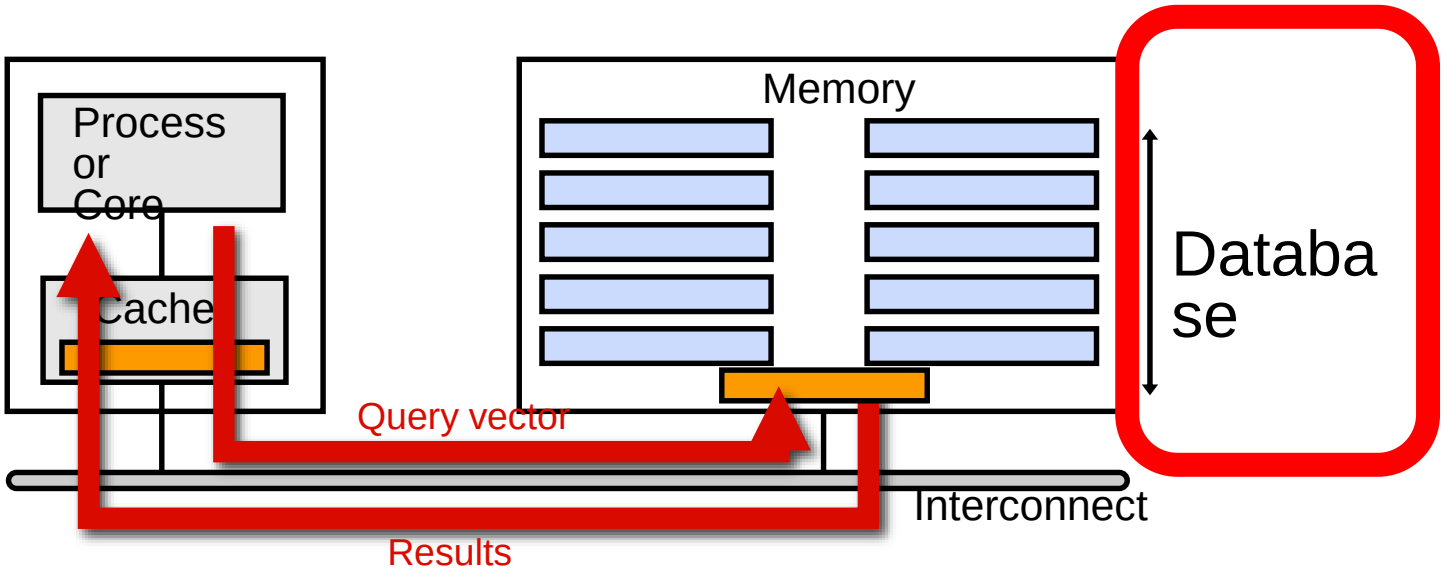


Goal: Ultra-Efficient Processing By Data



Goal: Memory similar to a "conventional" accelerator

Enabling Ultra-Efficient 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

Tolerating DRAM: Example Techniques

- Retention-Aware DRAM Refresh: Reducing Refresh Impact
- Refresh Access Parallelization: Reducing Refresh Impact
- Tiered-Latency DRAM: Reducing DRAM Latency
- RowClone: Accelerating Page Copy and Initialization
- Subarray-Level Parallelism: Reducing Bank Conflict Impact
- Base-Delta-Immediate Compression and Linearly Compressed Pages: Efficient Cache & Memory Compression

More Efficient Cache Utilization

- Compressing redundant data
  - Gennady Pekhimenko, Vivek Seshadri, Onur Mutlu, Philip B. Gibbons, Michael A. Kozuch, and Todd C. Mowry,
    "Base-Delta-Immediate Compression: Practical Data Compression for On-Chip Caches"
    Proceedings of the 21st ACM International Conference on Parallel Architectures and Compilation Techniques (***PACT***), Minneapolis, MN, September 2012. Slides (pptx)

  - Gennady Pekhimenko, Vivek Seshadri, Yoongu Kim, Hongyi Xin, Onur Mutlu, Michael A. Kozuch, Phillip B. Gibbons, and Todd C. Mowry,
    "Linearly Compressed Pages: A Low-Complexity, Low-Latency Main Memory Compression Framework"
    Proceedings of the 46th International Symposium on Microarchitecture (***MICRO***), Davis, CA, December 2013. Slides (pptx) (pdf) Lightning Session Slides (pptx) (pdf)

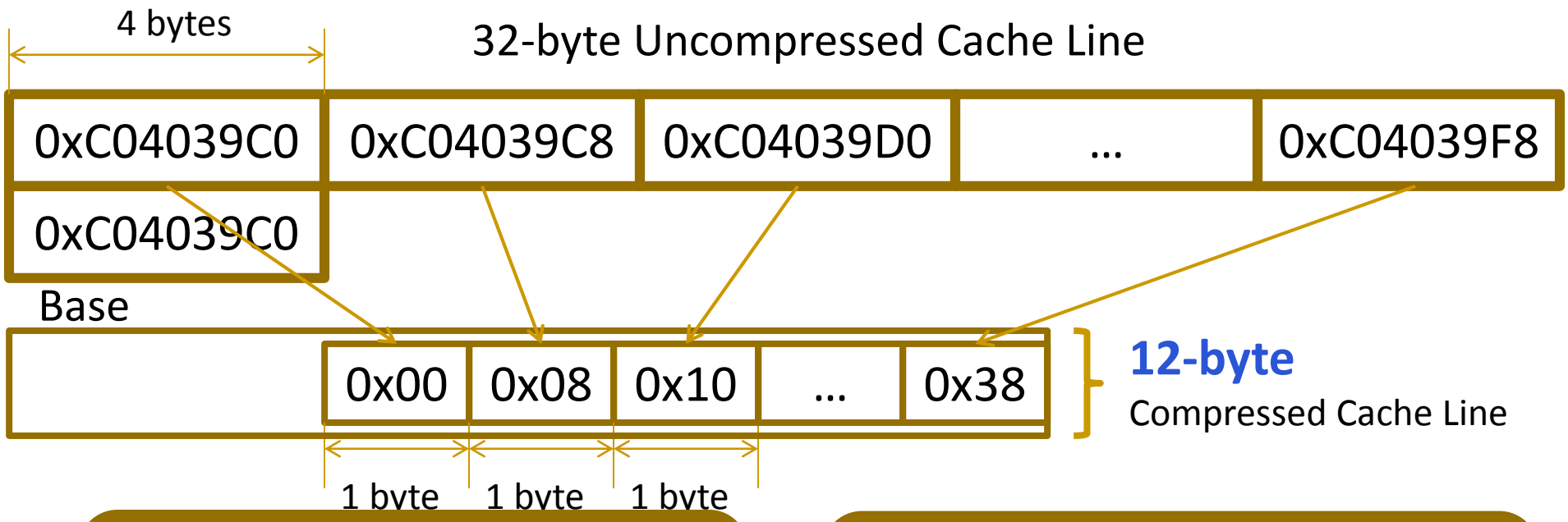
- Reducing pollution and thrashing
  - Vivek Seshadri, Onur Mutlu, Michael A. Kozuch, and Todd C. Mowry,
    "The Evicted-Address Filter: A Unified Mechanism to Address Both Cache Pollution and Thrashing"
    Proceedings of the 21st ACM International Conference on Parallel Architectures and Compilation Techniques (***PACT***), Minneapolis, MN, September 2012. Slides (pptx)

Key Data Patterns in Real Applications

Low Dynamic Range:

Differences between values are significantly smaller than the values themselves

Key Idea: Base+Delta (B+Δ) Encoding



✓ **Fast Decompression:** vector addition

✓ **Simple Hardware:** arithmetic and comparison

✓ **Effective:** good compression ratio

Can We Do Better?

- Uncompressible cache line (with a single base):

0x00000000	**0x09A4**0178	**0x000000**0B	**0x09A4**A838	…

- **Key idea:**
  Use more bases, e.g., two instead of one

- Pro:
  - More cache lines can be compressed

- Cons:
  - Unclear how to find these bases efficiently
  - Higher overhead (due to additional bases)

How to Find Two Bases Efficiently?

1. **First base - first element** in the cache line

   ✓ **Base+Delta part**

2. **Second base -** implicit base of **0**

   ✓ **Immediate part**

Advantages over 2 arbitrary bases:

- Better compression ratio
- Simpler compression logic

Base-Delta-Immediate (BΔI) Compression

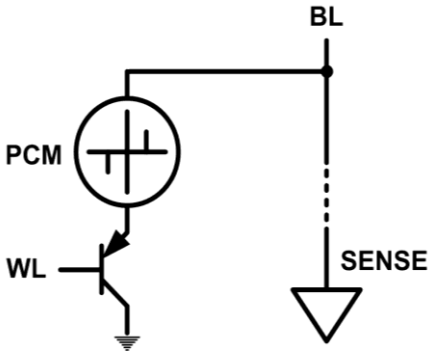
Agenda

- Major Trends Affecting Main Memory
- The Memory Scaling Problem and Solution Directions
  - New Memory 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?

Charge vs. Resistive Memories

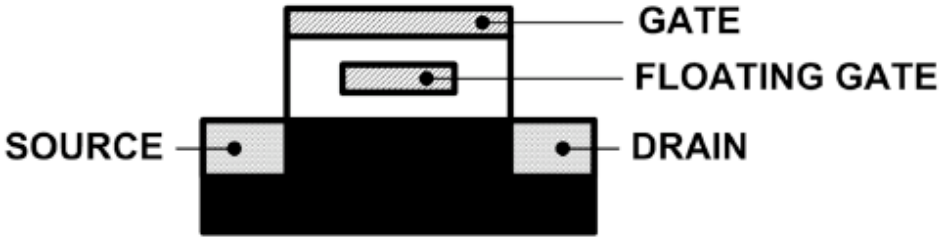
- Charge Memory (e.g., DRAM, Flash)
  - Write data by capturing charge Q
  - Read data by detecting voltage V

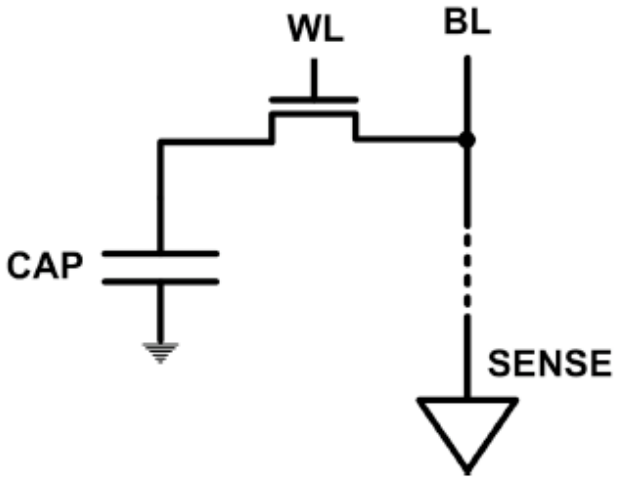
- Resistive Memory (e.g., PCM, STT-MRAM, memristors)
  - Write data by pulsing current dQ/dt
  - Read data by detecting resistance R

Limits of Charge Memory

- Difficult charge placement and control
  - Flash: floating gate charge
  - DRAM: capacitor charge, transistor leakage

- Reliable sensing becomes difficult as charge storage unit size reduces





Promising Resistive Memory Technologies

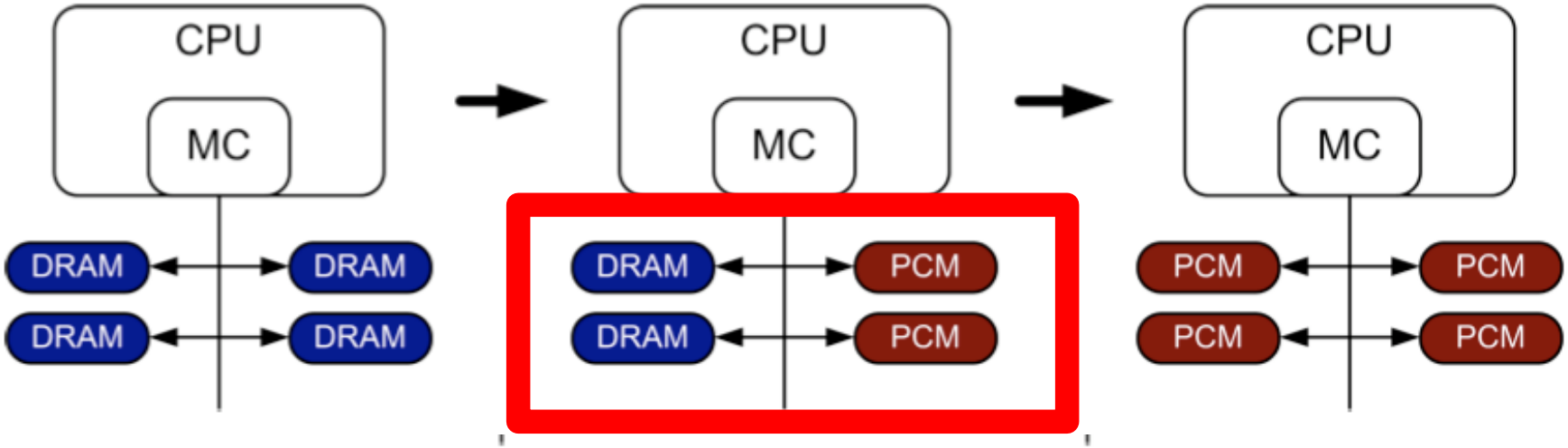
- PCM
  - Inject current to change material phase
  - Resistance determined by phase
- STT-MRAM
  - Inject current to change magnet polarity
  - Resistance determined by polarity
- Memristors/RRAM/ReRAM
  - Inject current to change atomic structure
  - Resistance determined by atom distance

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)
  - Reliability issues (resistance drift)
- 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)

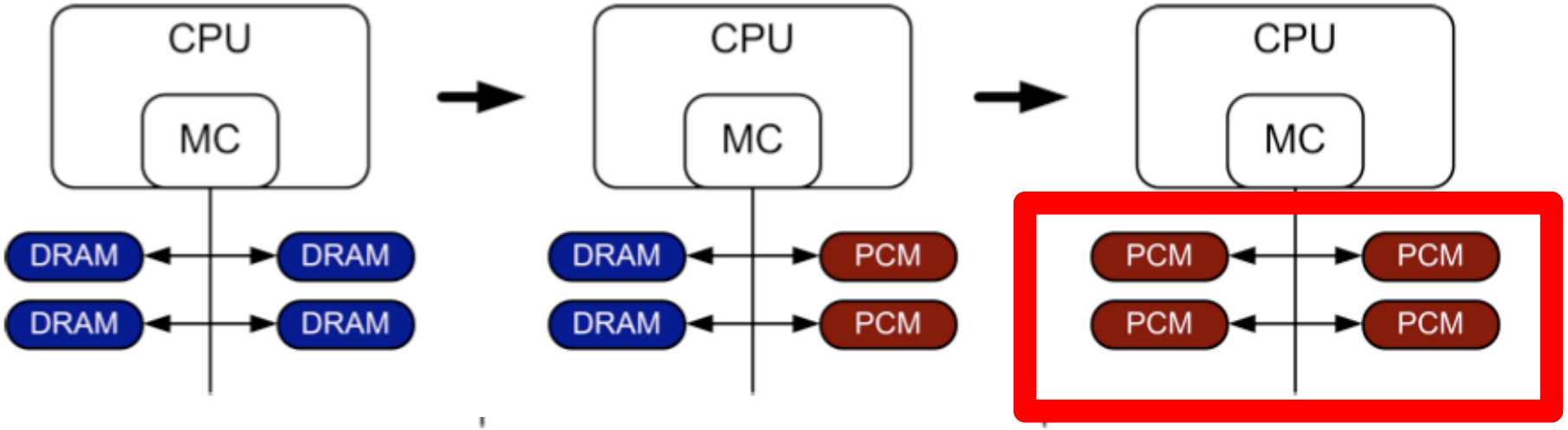
- How should PCM-based (main) memory be organized?



- 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

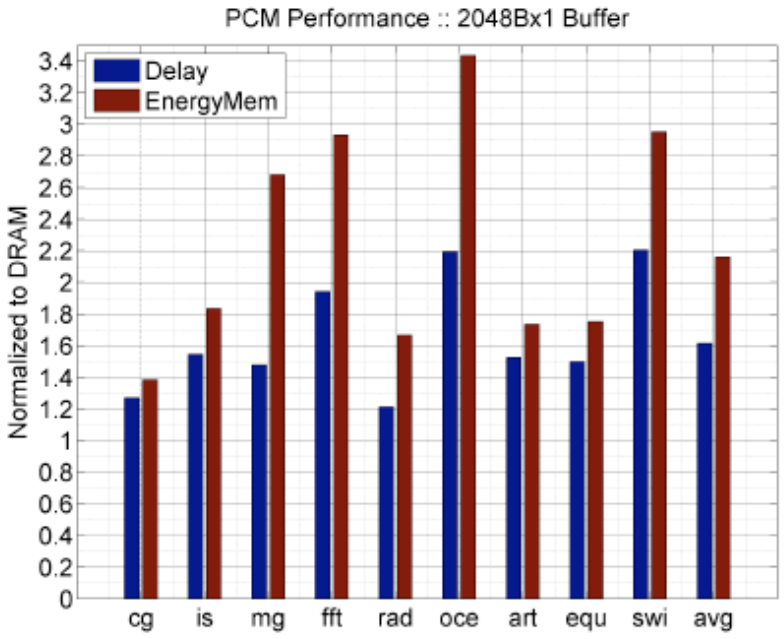
Table 1. Technology survey.

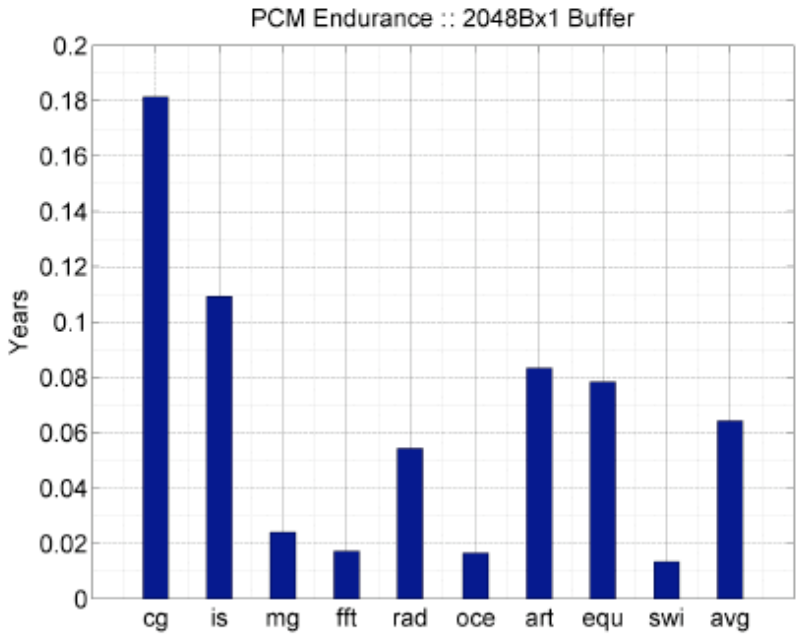
Parameter*	Published prototype									
	Horri[6]	Ahn[12]	Bedeschi[13]	Oh[14]	Pellizer[15]	Chen[5]	Kang[16]	Bedeschi[9]	Lee[10]	Lee[2]
Year	2003	2004	2004	2005	2006	2006	2006	2008	2008	**
Process, F (nm)	**	120	180	120	90	**	100	90	90	90
Array size (Mbytes)	**	64	8	64	**	**	256	256	512	**
Material	GST, N-d	GST, N-d	GST	GST	GST	GS, N-d	GST	GST	GST	GST, N-d
Cell size (μm^2)	**	0.290	0.290	**	0.097	60 nm^2	0.166	0.097	0.047	0.065 to 0.097
Cell size, F^2	**	20.1	9.0	**	12.0	**	16.6	12.0	5.8	9.0 to 12.0
Access device	**	**	BJT	FET	BJT	**	FET	BJT	Diode	BJT
Read time (ns)	**	70	48	68	**	**	62	**	55	48
Read current (μA)	**	**	40	**	**	**	**	**	**	40
Read voltage (V)	**	3.0	1.0	1.8	1.6	**	1.8	**	1.8	1.0
Read power (μW)	**	**	40	**	**	**	**	**	**	40
Read energy (pJ)	**	**	2.0	**	**	**	**	**	**	2.0
Set time (ns)	100	150	150	180	**	80	300	**	400	150
Set current (μA)	200	**	300	200	**	55	**	**	**	150
Set voltage (V)	**	**	2.0	**	**	1.25	**	**	**	1.2
Set power (μW)	**	**	300	**	**	34.4	**	**	**	90
Set energy (pJ)	**	**	45	**	**	2.8	**	**	**	13.5
Reset time (ns)	50	10	40	10	**	60	50	**	50	40
Reset current (μA)	600	600	600	600	400	90	600	300	600	300
Reset voltage (V)	**	**	2.7	**	1.8	1.6	**	1.6	**	1.6
Reset power (μW)	**	**	1620	**	**	80.4	**	**	**	480
Reset energy (pJ)	**	**	64.8	**	**	4.8	**	**	**	19.2
Write endurance (MLC)	10^7	10^9	10^6	**	10^8	10^4	**	10^5	10^5	10^8

* BJT: bipolar junction transistor; FET: field-effect transistor; GST: $Ge_2Sb_2Te_5$; MLC: multilevel cells; N-d: nitrogen doped.
** This information is not available in the publication cited.

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



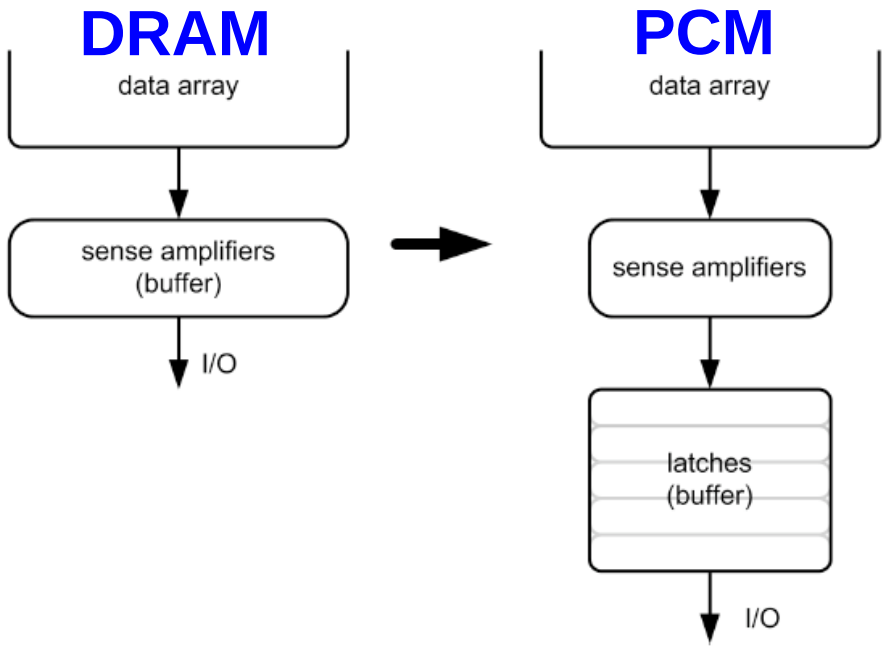


- Lee, Ipek, Mutlu, Burger, "Architecting Phase Change Memory as a Scalable DRAM Alternative," ISCA 2009.

Architecting PCM to Mitigate Shortcomings

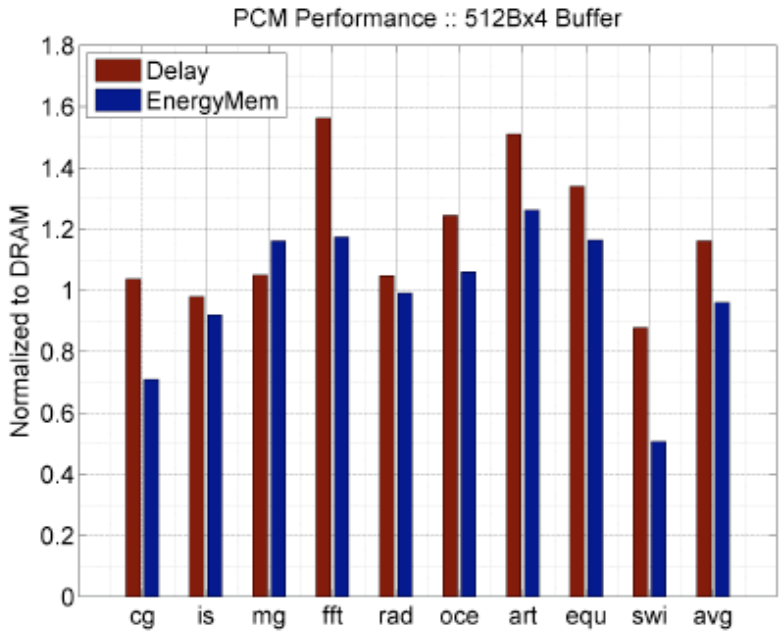
- Idea 1: Use multiple narrow row buffers in each PCM chip
  → Reduces array reads/writes → better endurance, latency, energy

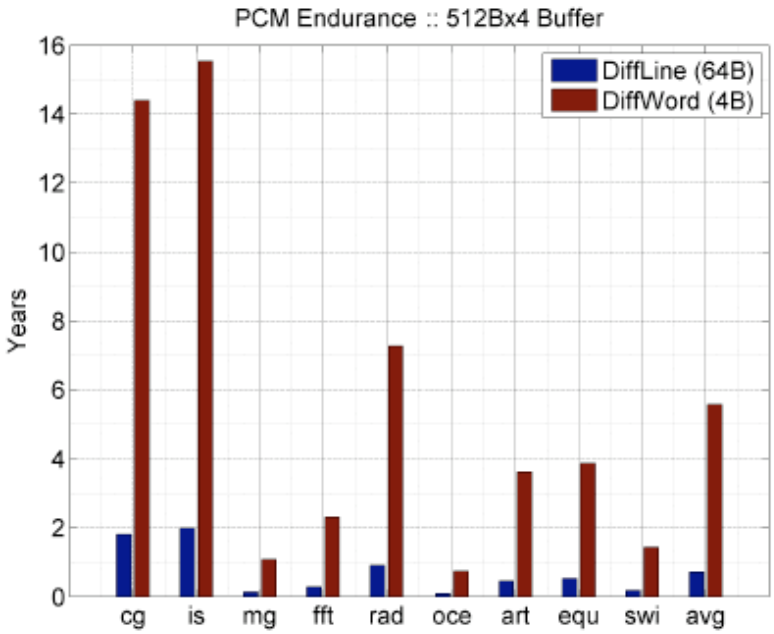
- Idea 2: Write into array at cache block or word granularity
  → Reduces unnecessary wear



Results: Architected PCM as Main Memory

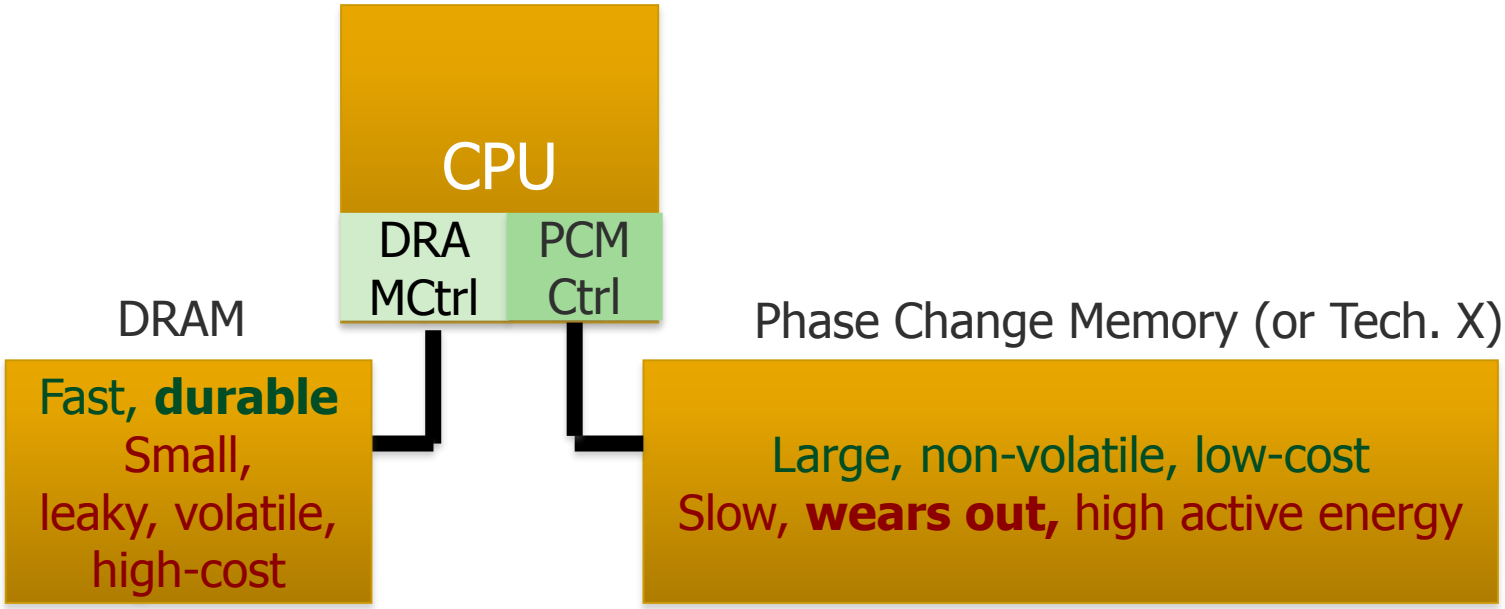
- 1.2x delay, 1.0x energy, 5.6-year average lifetime
- Scaling improves energy, endurance, density





- 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

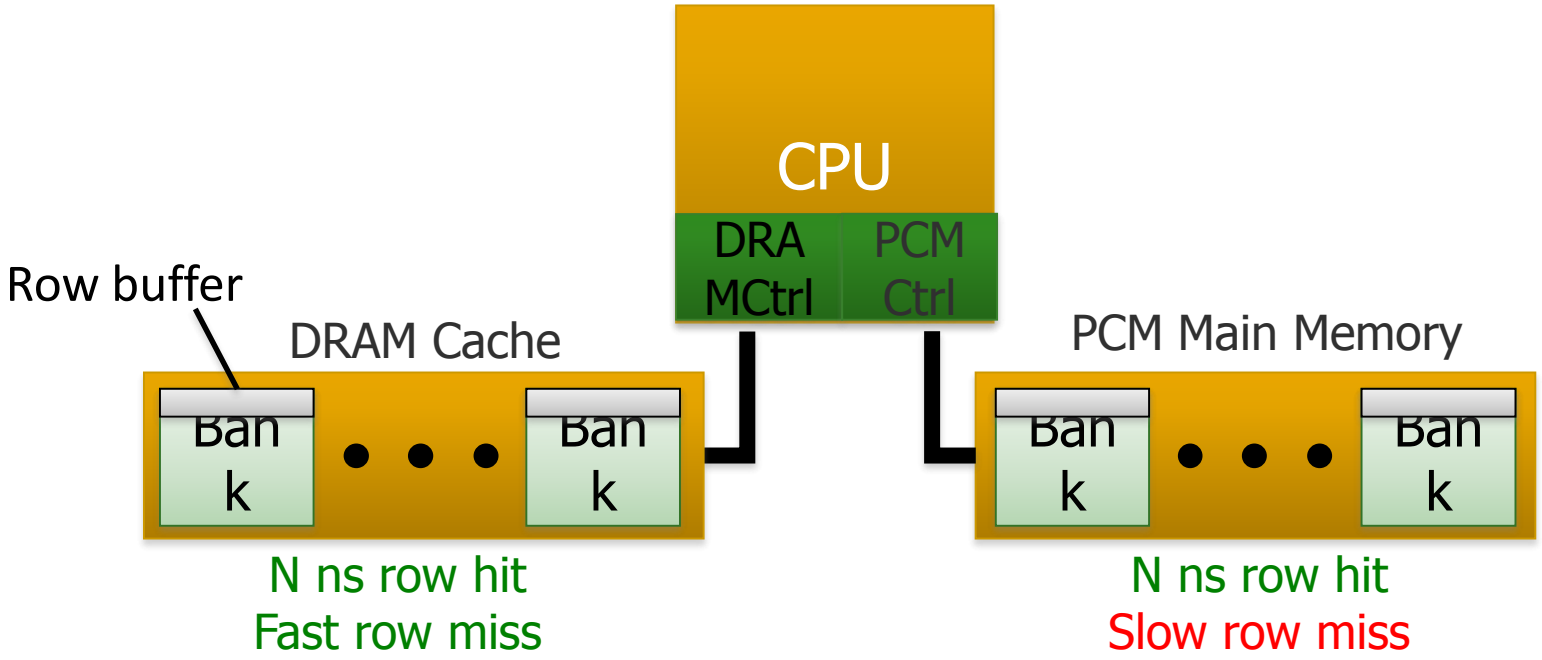
- PCM is main memory; DRAM caches memory rows/blocks
  - Benefits: Reduced latency on DRAM cache hit; write filtering
- Memory controller hardware manages the DRAM cache
  - Benefit: Eliminates system software overhead

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

- 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

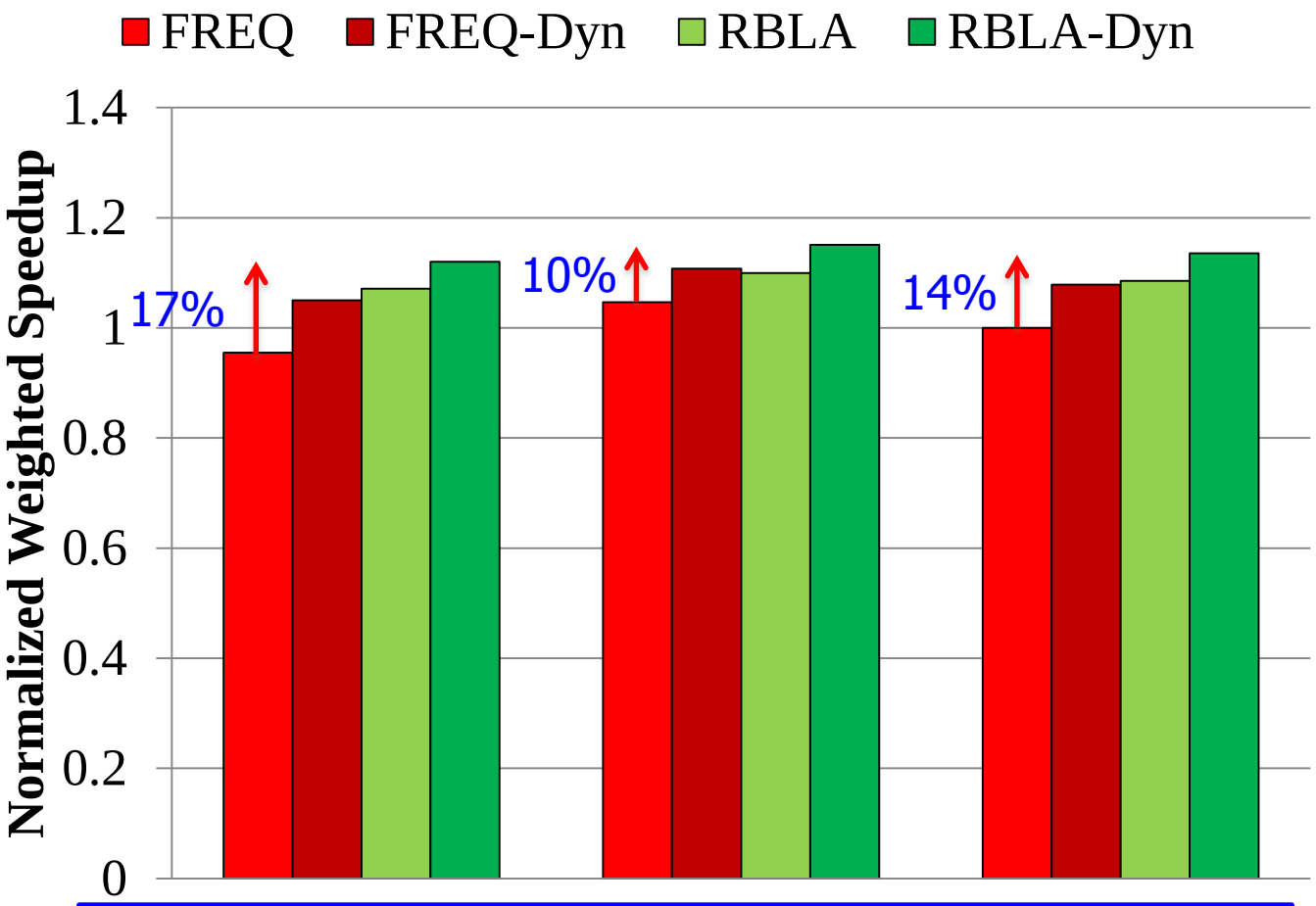
Row-Locality-Aware Data Placement

- 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

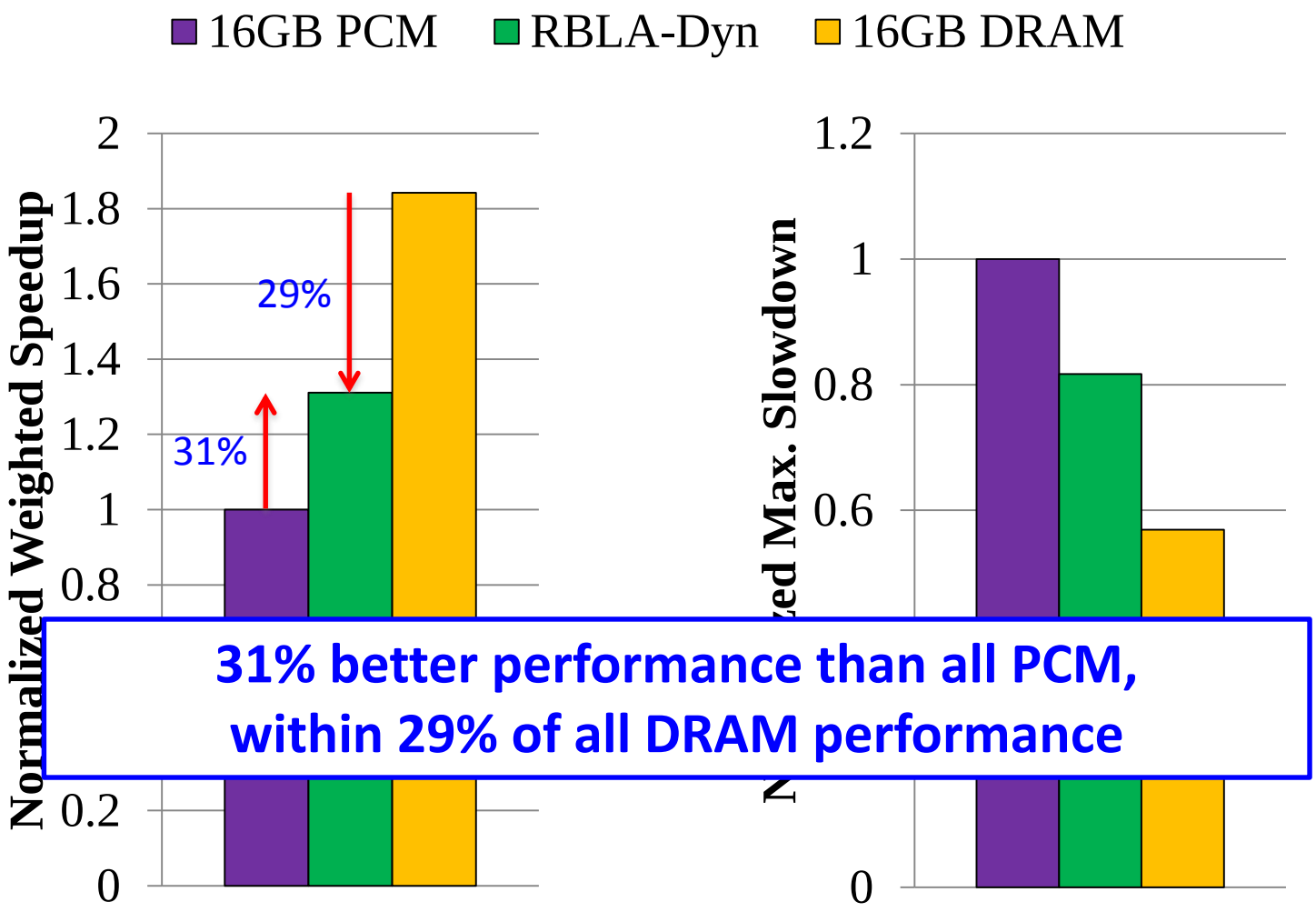
- Yoon et al., "Row Buffer Locality-Aware Data Placement in Hybrid Memories," ICCD 2012 Best Paper Award.

Row-Locality-Aware Data Placement: Results



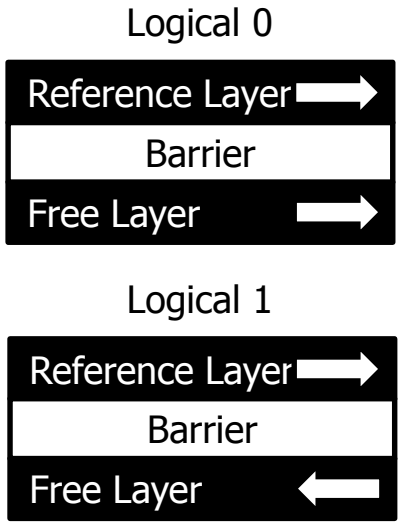
Memory energy-efficiency and fairness also improve correspondingly

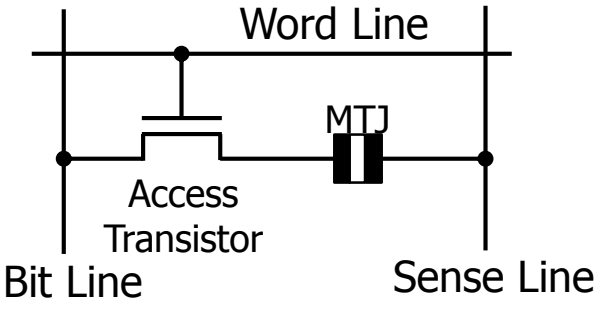
Hybrid vs. All-PCM/DRAM



Aside: STT-MRAM as Main Memory

- Magnetic Tunnel Junction (MTJ)
  - Reference layer: Fixed
  - Free layer: Parallel or anti-parallel
- Cell
  - Access transistor, bit/sense lines
- Read and Write
  - Read: Apply a small voltage across bitline and senseline; read the current.
  - Write: Push large current through MTJ. Direction of current determines new orientation of the free layer.





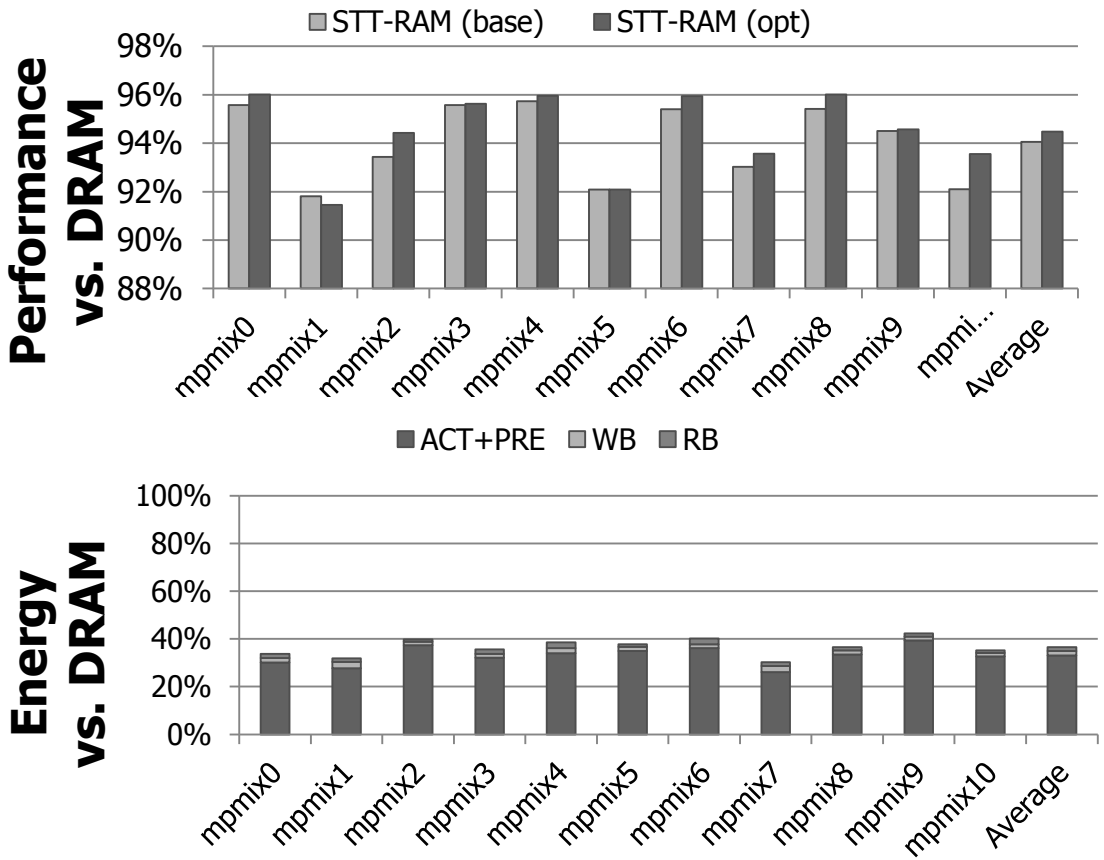
- Kultursay et al., "Evaluating STT-RAM as an Energy-Efficient Main Memory Alternative," ISPASS 2013.

Aside: STT-MRAM: Pros and Cons

- Pros over DRAM
  - Better technology scaling
  - Non volatility
  - Low idle power (no refresh)
- Cons
  - Higher write latency
  - Higher write energy
  - Reliability?
- Another level of freedom
  - Can trade off non-volatility for lower write latency/energy (by reducing the size of the MTJ)

Architected STT-MRAM as Main Memory

- 4-core, 4GB main memory, multiprogrammed workloads
- ~6% performance loss, ~60% energy savings vs. DRAM



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

Agenda

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

Principles (So Far)

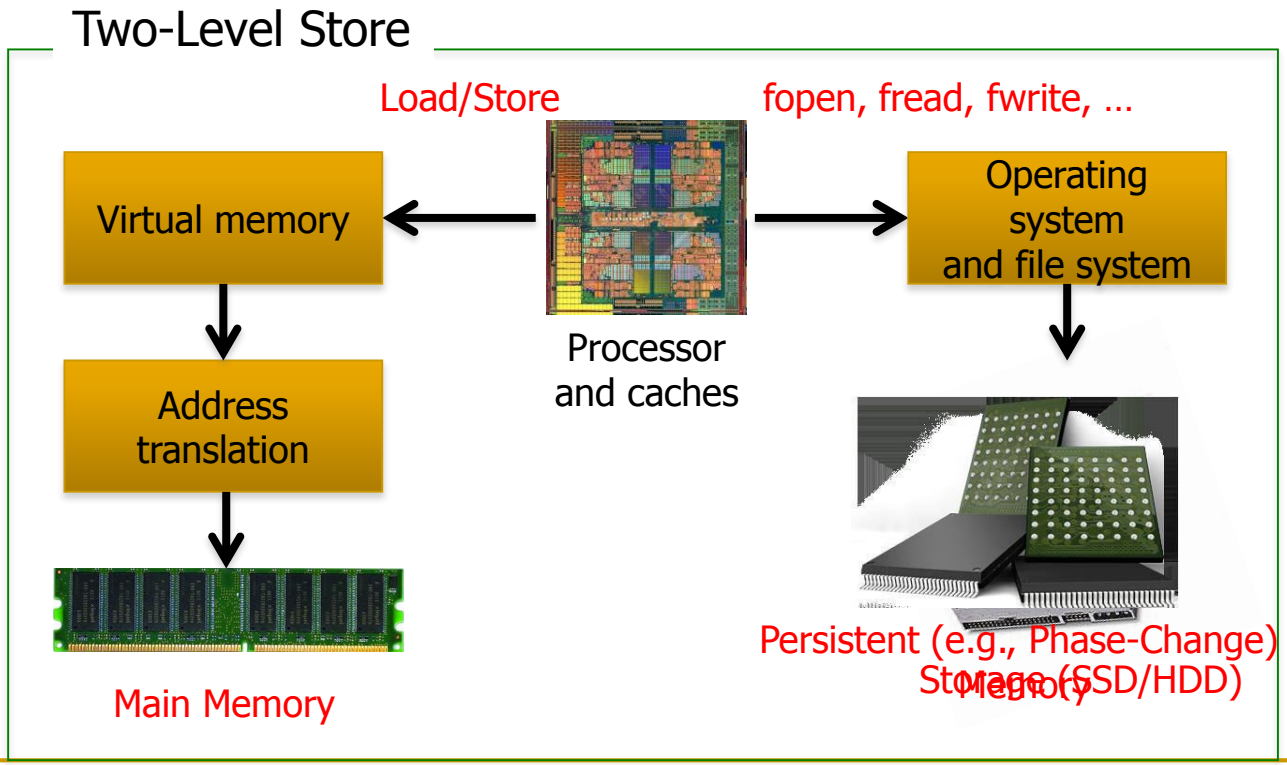
- Better cooperation between devices and the system
  - Expose more information about devices to upper layers
  - More flexible interfaces
- Better-than-worst-case design
  - Do not optimize for the worst case
  - Worst case should not determine the common case
- Heterogeneity in design (specialization, asymmetry)
  - 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
- New applications
  - e.g., ultra-fast checkpoint and restore
- More robust system design
  - e.g., reducing data loss
- Processing tightly-coupled with memory
  - e.g., enabling efficient search and filtering

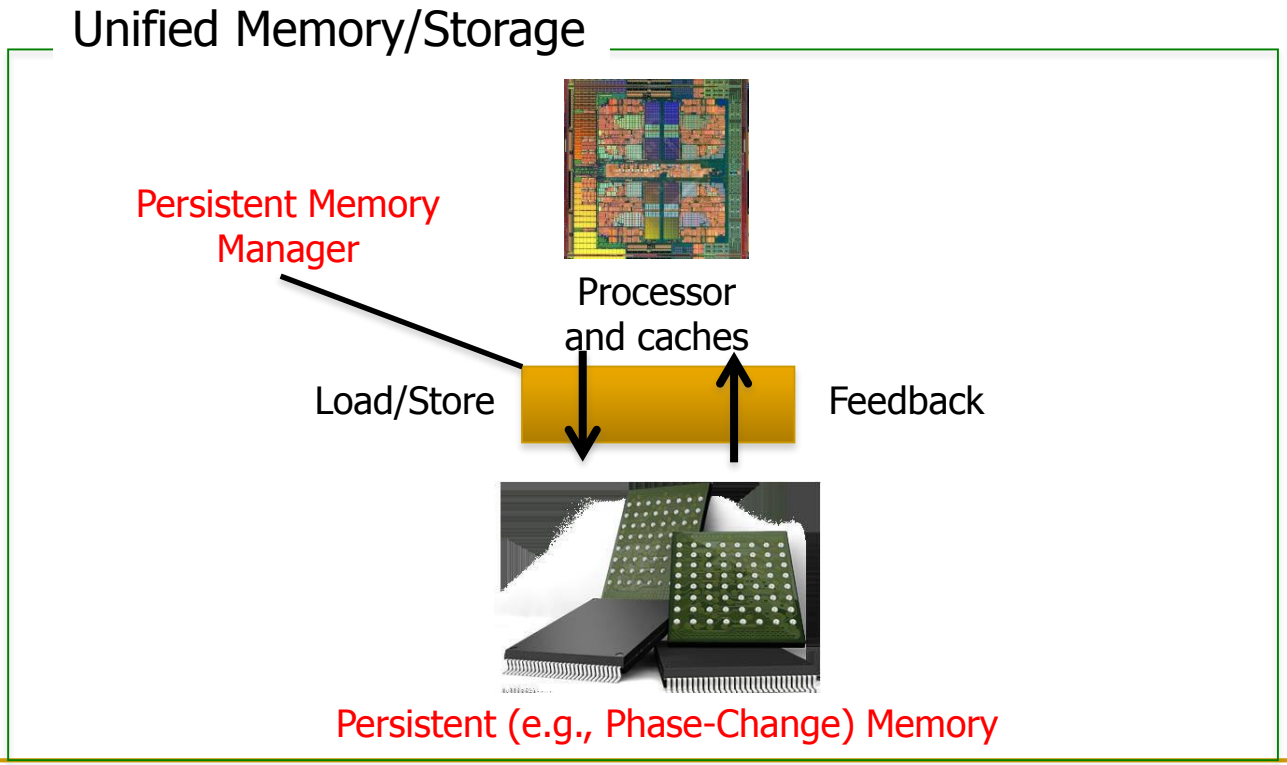
Coordinated Memory and Storage with NVM (I)

- The traditional two-level storage model is a bottleneck with NVM
  - **Volatile** data in memory → a **load/store** interface
  - **Persistent** data in storage → a **file system** interface
  - Problem: Operating system (OS) and file system (FS) code to locate, translate, buffer data become performance and energy bottlenecks with fast NVM stores



Coordinated Memory and Storage with NVM (II)

- Goal: Unify memory and storage management in a single unit to eliminate wasted work to locate, transfer, and translate data
  - Improves both energy and performance
  - Simplifies programming model as well



Meza+, "A Case for Efficient Hardware-Software Cooperative Management of Storage and Memory," WEED 2013.

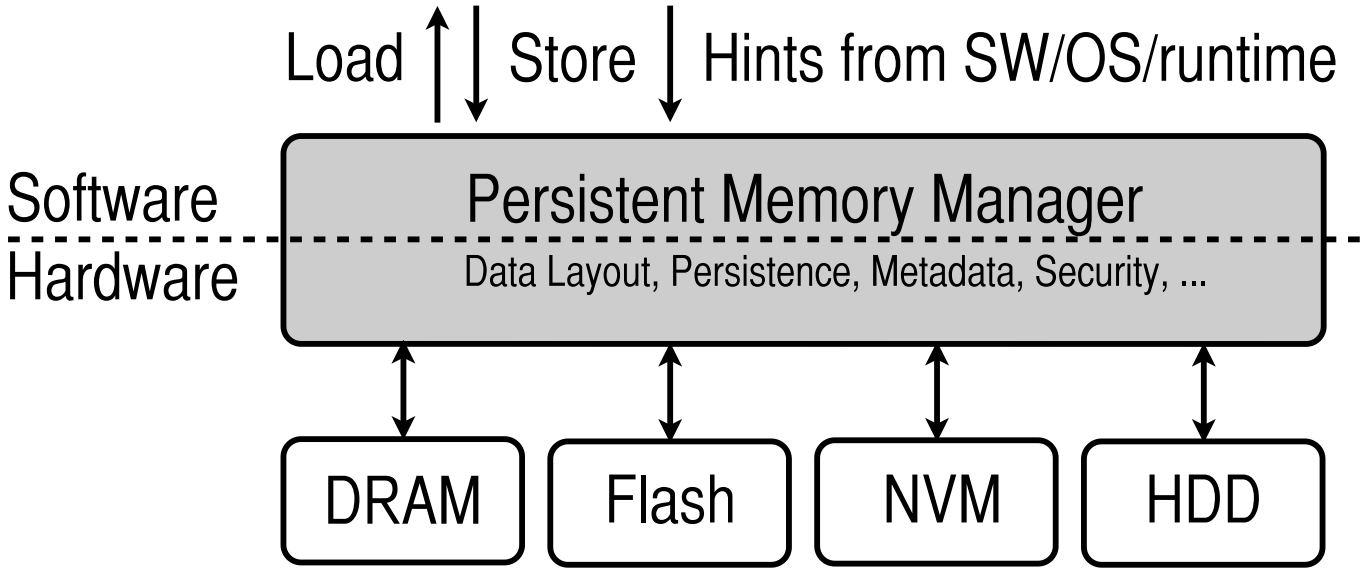
The Persistent Memory Manager (PMM)

- Exposes a load/store interface to access persistent data
  - Applications can directly access persistent memory → no conversion, translation, location overhead for persistent data
- Manages data placement, location, persistence, security
  - To get the best of multiple forms of storage
- Manages metadata storage and retrieval
  - This can lead to overheads that need to be managed
- Exposes hooks and interfaces for system software
  - To enable better data placement and management decisions
- Meza+, "A Case for Efficient Hardware-Software Cooperative Management of Storage and Memory," WEED 2013.

The Persistent Memory Manager (PMM)

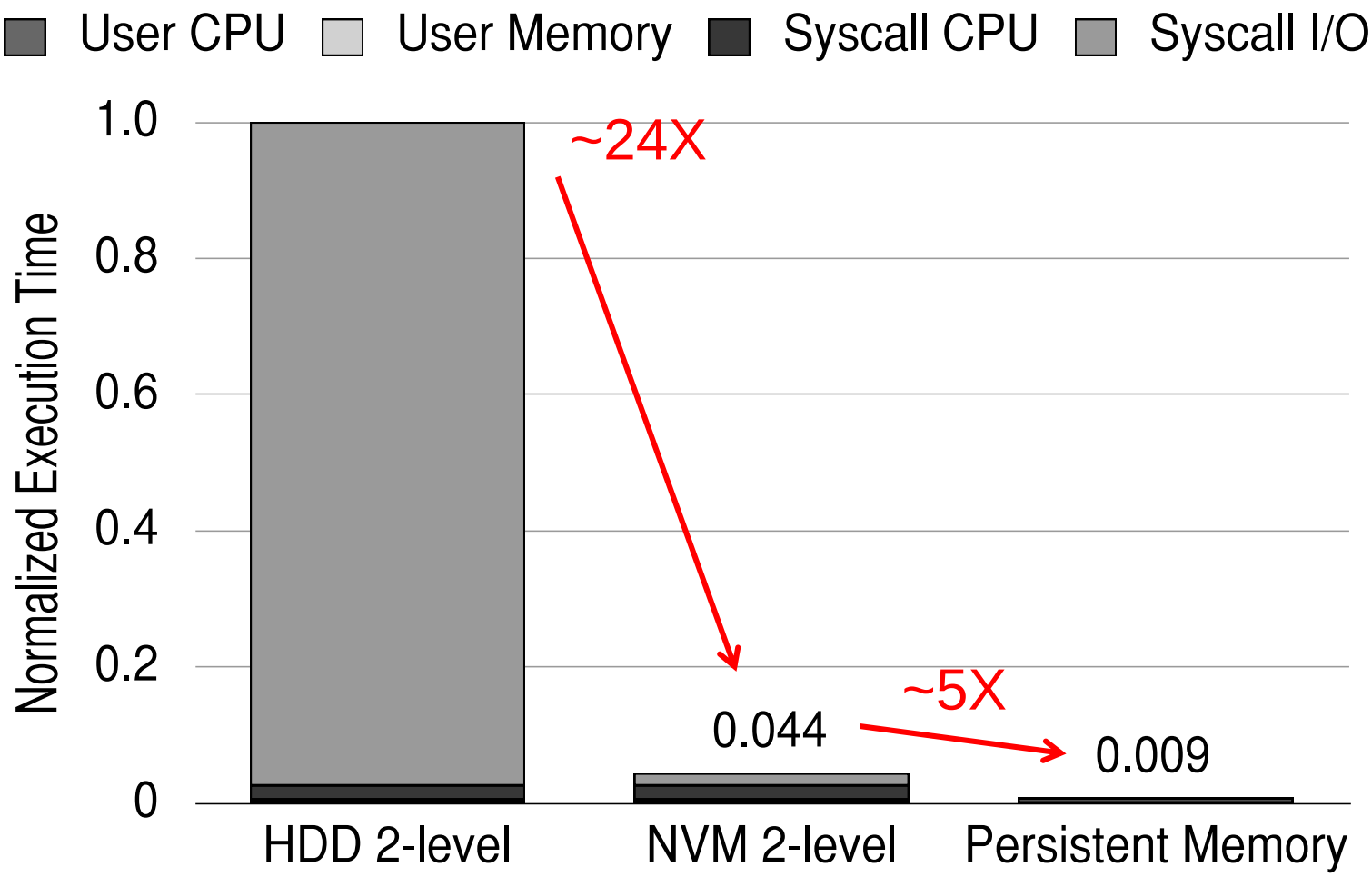
```
1  int main(void) {
2    // data in file.dat is persistent
3    FILE myData = "file.dat";
4    myData = new int[64];
5  }
6  void updateValue(int n, int value) {
7    FILE myData = "file.dat";
8    myData[n] = value; // value is persistent
9  }
```

Persistent objects

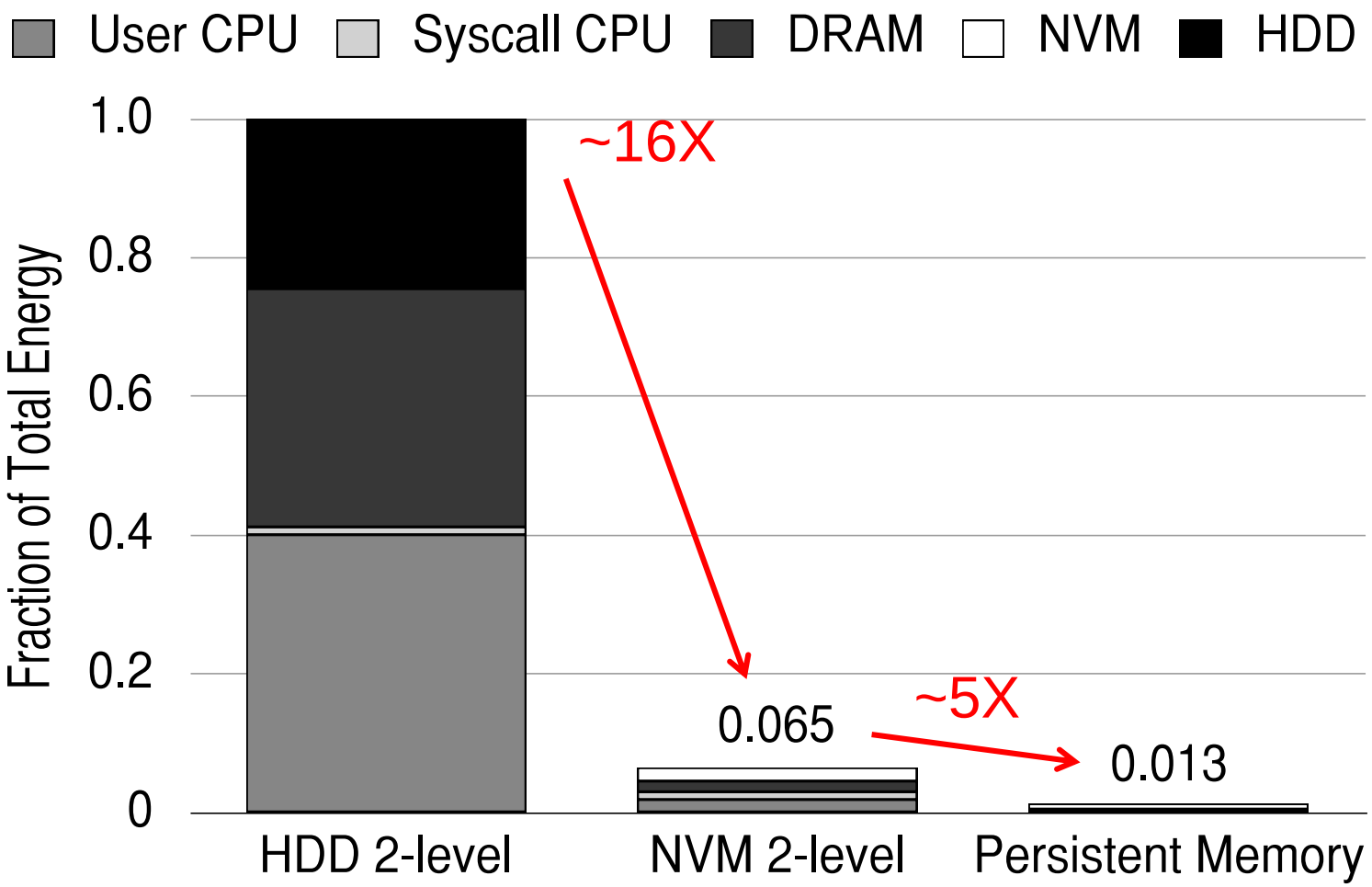


PMM uses access and hint information to allocate, locate, migrate and access data in the heterogeneous array of devices

Performance Benefits of a Single-Level Store

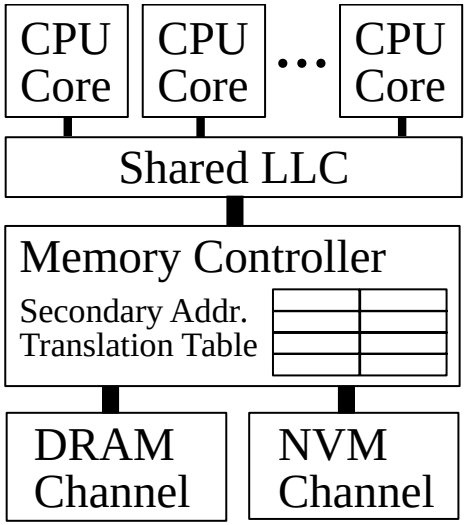


Energy Benefits of a Single-Level Store



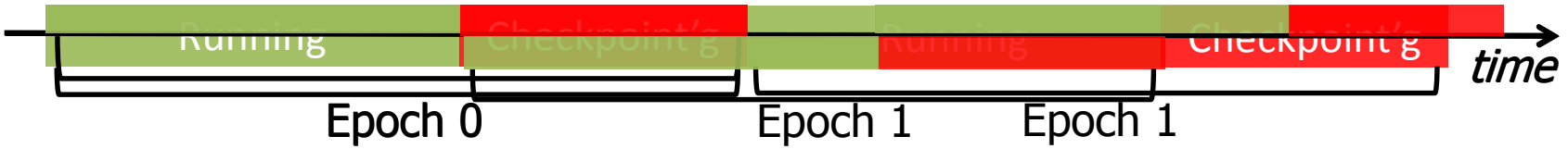
Transparent Hybrid Non-Volatile Memory

- **Problem**: How do you provide consistency and prevent data corruption in NVM upon a system crash?
- **Goal**: Provide efficient programmer-transparent consistency in hybrid NVM
  - **Efficiency**: use hybrid DRAM/NVM for high performance
    - DRAM is not (only) a transparent cache
  - **Transparency**: no library APIs or explicit interfaces to access NVM; just loads and stores
    - Makes life easier for the programmer
    - Easier to support legacy code and hypervisors
- **Challenges to Solve**
  - How to guarantee consistency
  - How to maximize performance



THNVM: Solution Overview

- Idea 1: Transparent checkpointing



  - Need to overlap checkpointing and execution

- Idea 2: Differentiated consistency schemes for DRAM and NVM
  - Writeback: buffer sequential writes in DRAM
  - Address Remapping: handle random writes in NVM

- Idea 3: Dynamic migration of data for performance
  - High write-locality data placed in DRAM

Agenda

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

Summary: Memory/Storage Scaling

- Memory scaling problems are a critical bottleneck for system performance, efficiency, and usability
- New memory/storage + compute architectures
  - Rethinking DRAM; processing close to data; accelerating bulk operations
- Enabling emerging NVM technologies
  - Hybrid memory systems with automatic data management
  - Coordinated management of memory and storage with NVM
- System-level memory/storage QoS
- Three principles are essential for scaling
  - Software/hardware/device cooperation
  - Better-than-worst-case design
  - Heterogeneity (specialization, asymmetry)

Related Videos and Course Materials

- Computer Architecture Lecture Videos on Youtube
  - https://www.youtube.com/playlist?list=PL5PHm2jkkXmidJOd59REog9jDnPDTG6IJ
- Computer Architecture Course Materials
  - http://www.ece.cmu.edu/~ece447/s13/doku.php?id=schedule
- Advanced Computer Architecture Course Materials
  - http://www.ece.cmu.edu/~ece740/f13/doku.php?id=schedule
- Advanced Computer Architecture Lecture Videos on Youtube
  - https://www.youtube.com/playlist?list=PL5PHm2jkkXmgDN1PLwOY_tGtUlynnyV6D

Referenced Papers

- All are available at
  http://users.ece.cmu.edu/~omutlu/projects.htm
  http://scholar.google.com/citations?user=7XyGUGkAAAAJ&hl=en

Rethinking Memory/Storage System Design

Onur Mutlu
onur@cmu.edu
http://users.ece.cmu.edu/~omutlu/

Carnegie Mellon

SAFARI

Aside: Self-Optimizing Memory Controllers

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