

Rethinking Memory System Design Business as Usual in the Next Decade?

Onur Mutlu

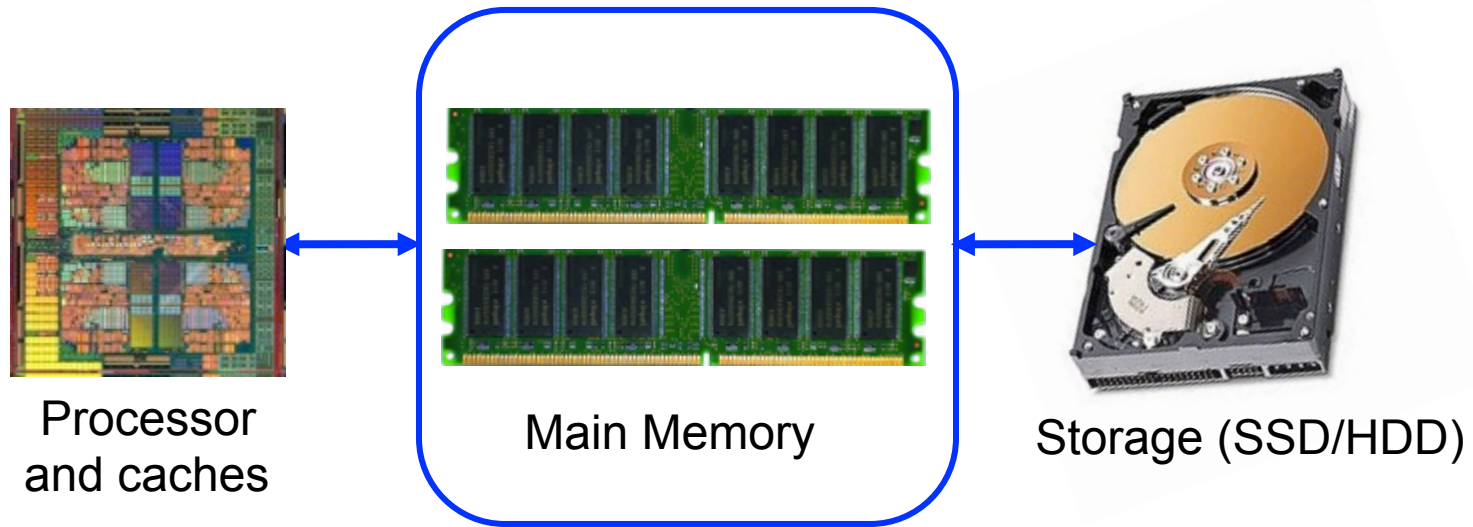
onur.mutlu@inf.ethz.ch

<http://users.ece.cmu.edu/~omutlu/>

October 6, 2016

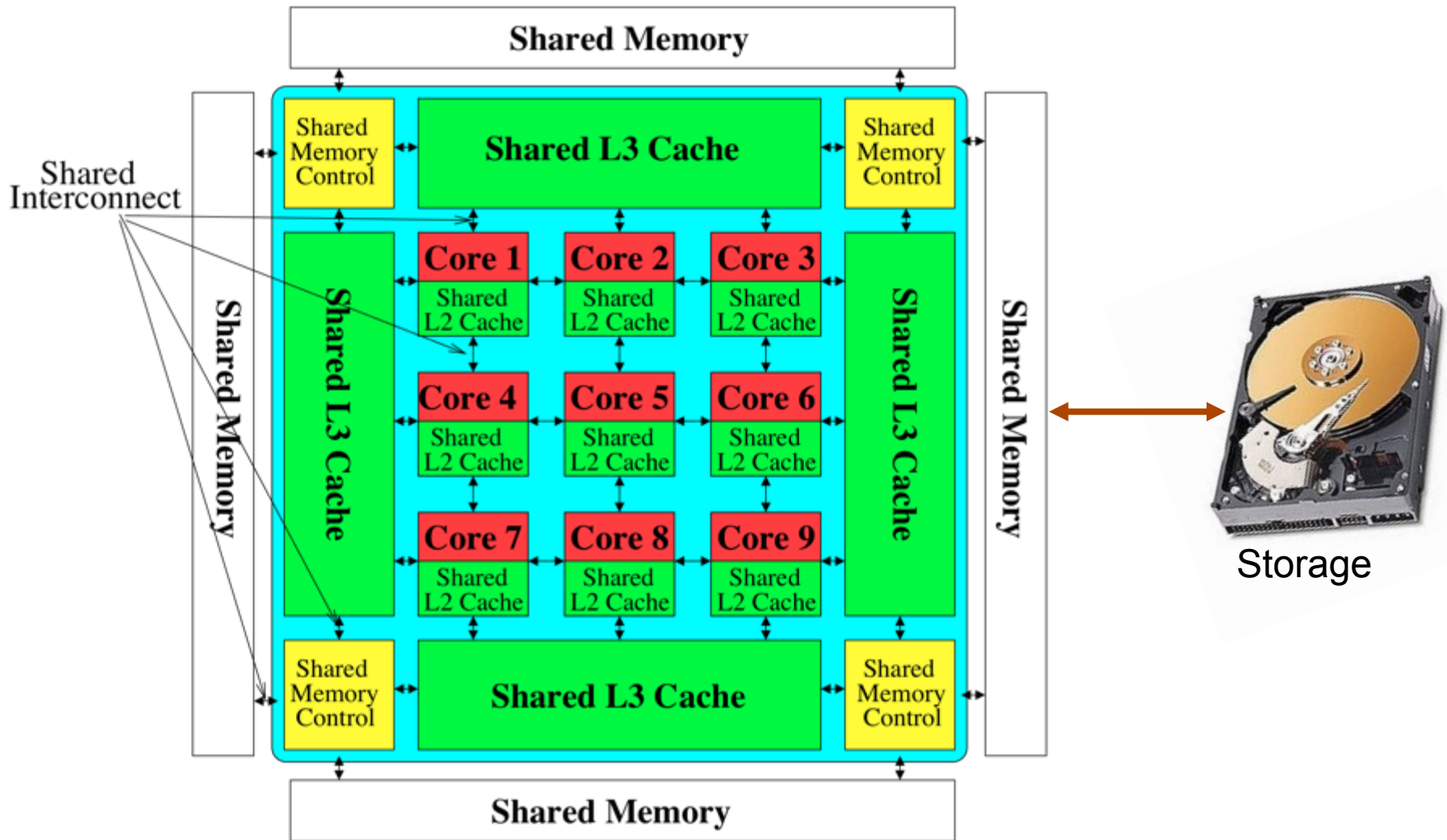
RSP 2016 Keynote (Pittsburgh)

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
- Cross-Cutting Principles
- 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

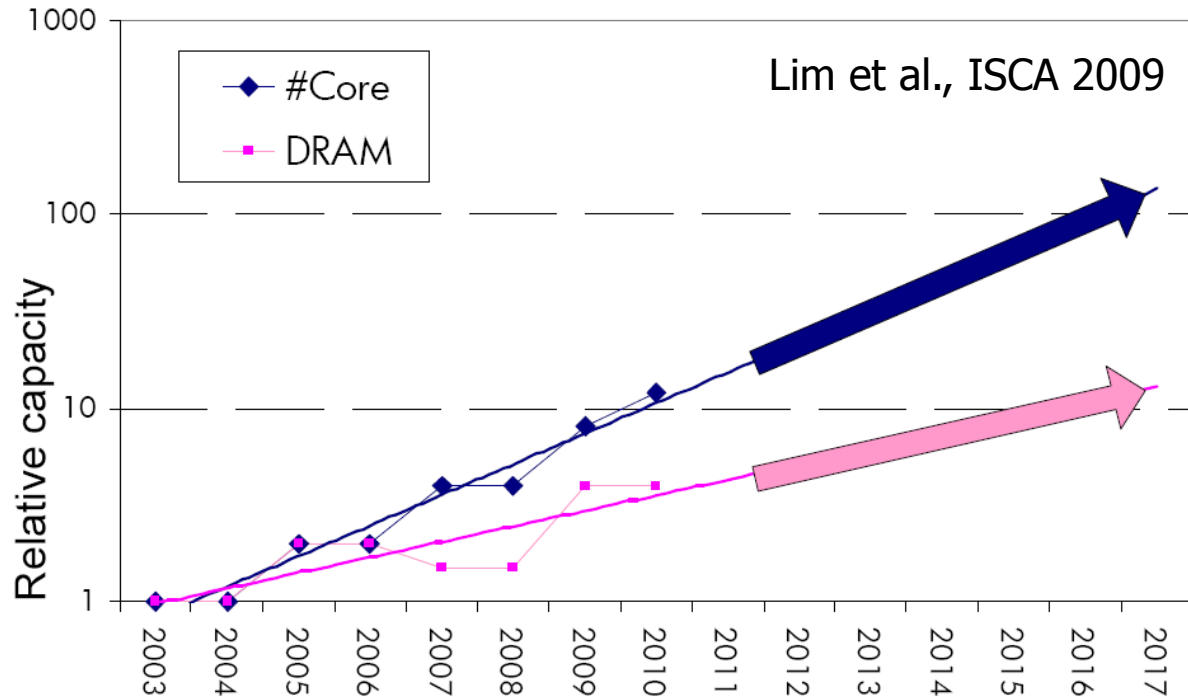
Major Trends Affecting Main Memory (II)

- 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



- *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)

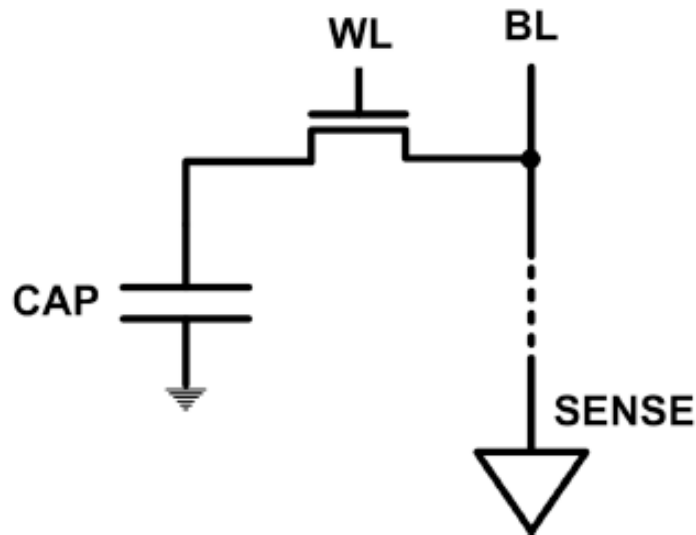
- Need for main memory capacity, bandwidth, QoS increasing
- Main memory energy/power is a key system design concern
- DRAM technology scaling is ending
 - ITRS projects DRAM will not scale easily below X nm
 - Scaling has provided many benefits:
 - higher capacity (density), lower cost, lower energy

Agenda

- Major Trends Affecting Main Memory
- The Memory Scaling Problem and Solution Directions
 - New Memory Architectures
 - Enabling Emerging Technologies
- Cross-Cutting Principles
- Summary

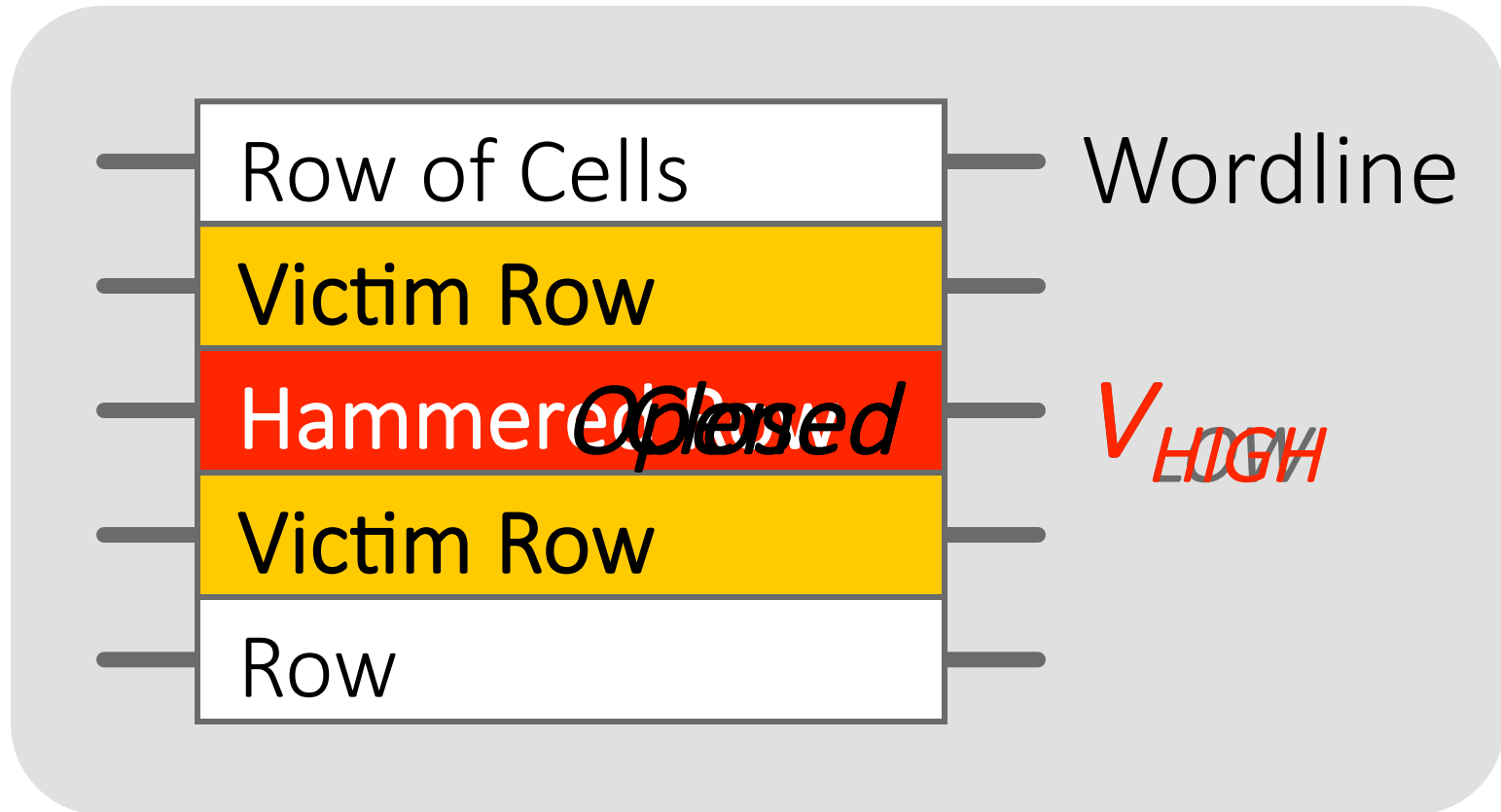
The DRAM Scaling Problem

- 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

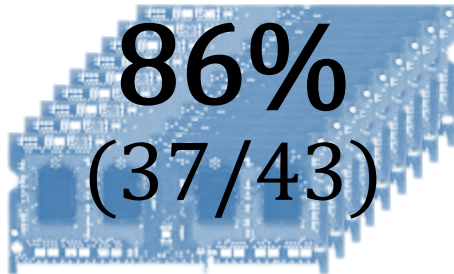
An Example of the DRAM Scaling Problem



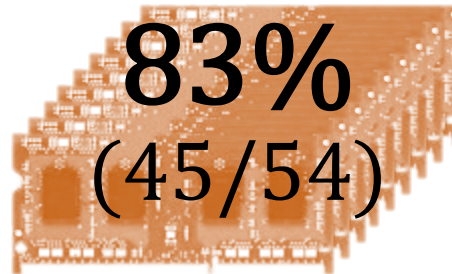
Repeatedly opening and closing a row enough times within a refresh interval induces **disturbance errors** in adjacent rows in **most real DRAM chips you can buy today**

Most DRAM Modules Are at Risk

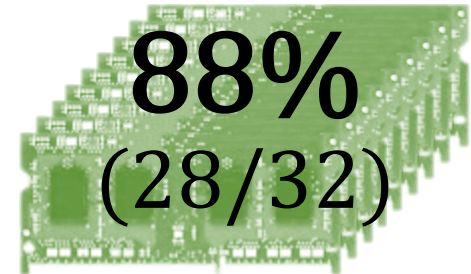
A company



B company



C company

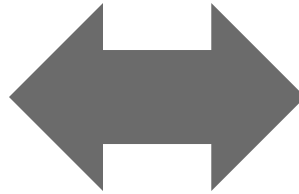


Up to
 1.0×10^7
errors

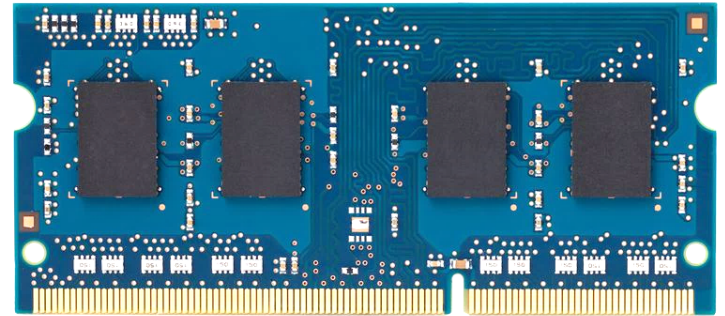
Up to
 2.7×10^6
errors

Up to
 3.3×10^5
errors

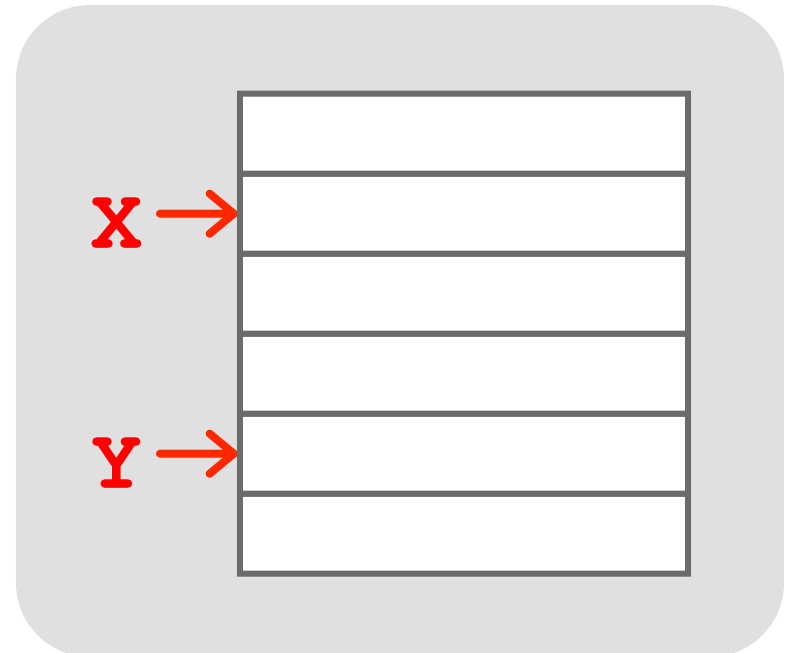
x86 CPU



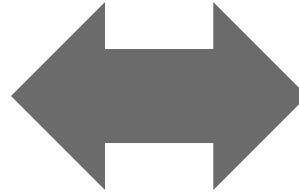
DRAM Module



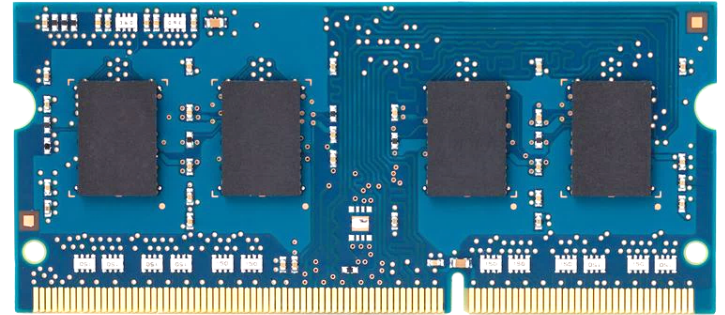
```
loop:  
  mov  (X), %eax  
  mov  (Y), %ebx  
  clflush (X)  
  clflush (Y)  
  mfence  
  jmp  loop
```



x86 CPU



DRAM Module

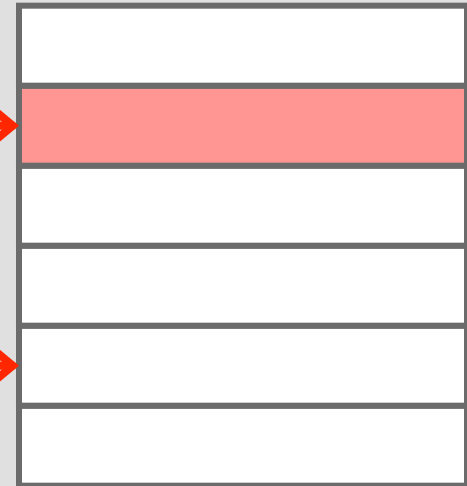


loop:

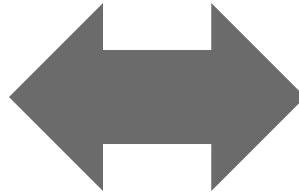
```
mov  (X), %eax  
mov  (Y), %ebx  
clflush (X)  
clflush (Y)  
mfence  
jmp  loop
```

X →

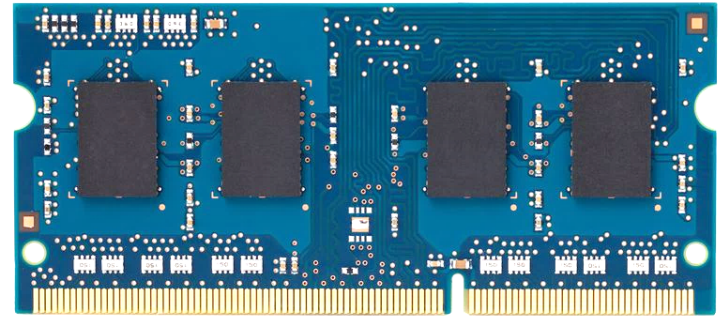
Y →



x86 CPU

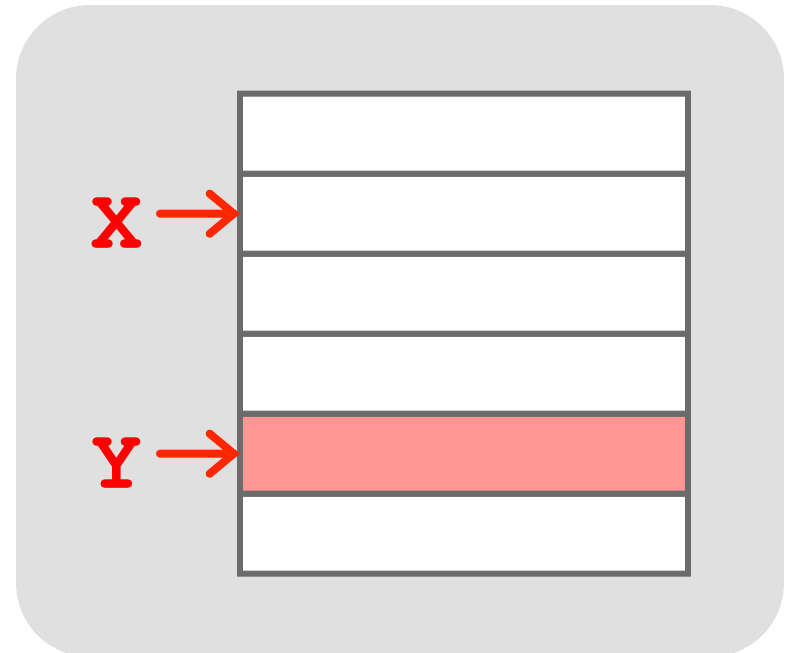


DRAM Module

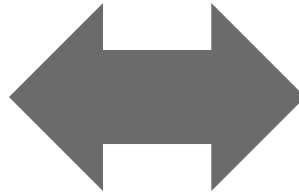


loop:

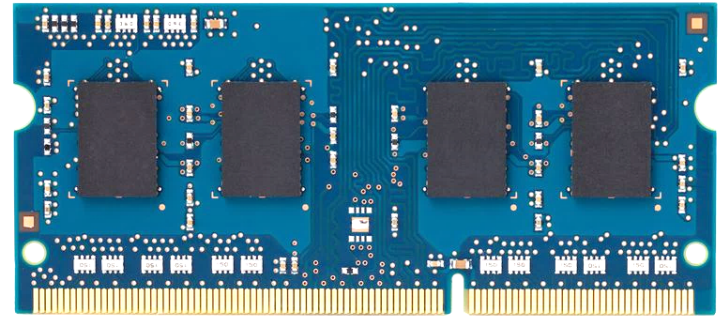
```
mov  (X), %eax  
mov  (Y), %ebx  
clflush (X)  
clflush (Y)  
mfence  
jmp  loop
```



x86 CPU

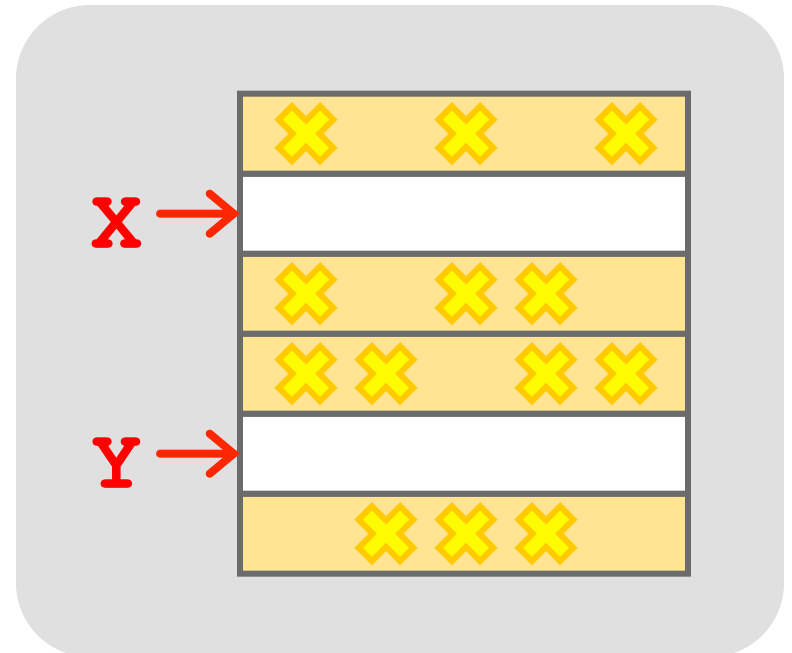


DRAM Module



loop:

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

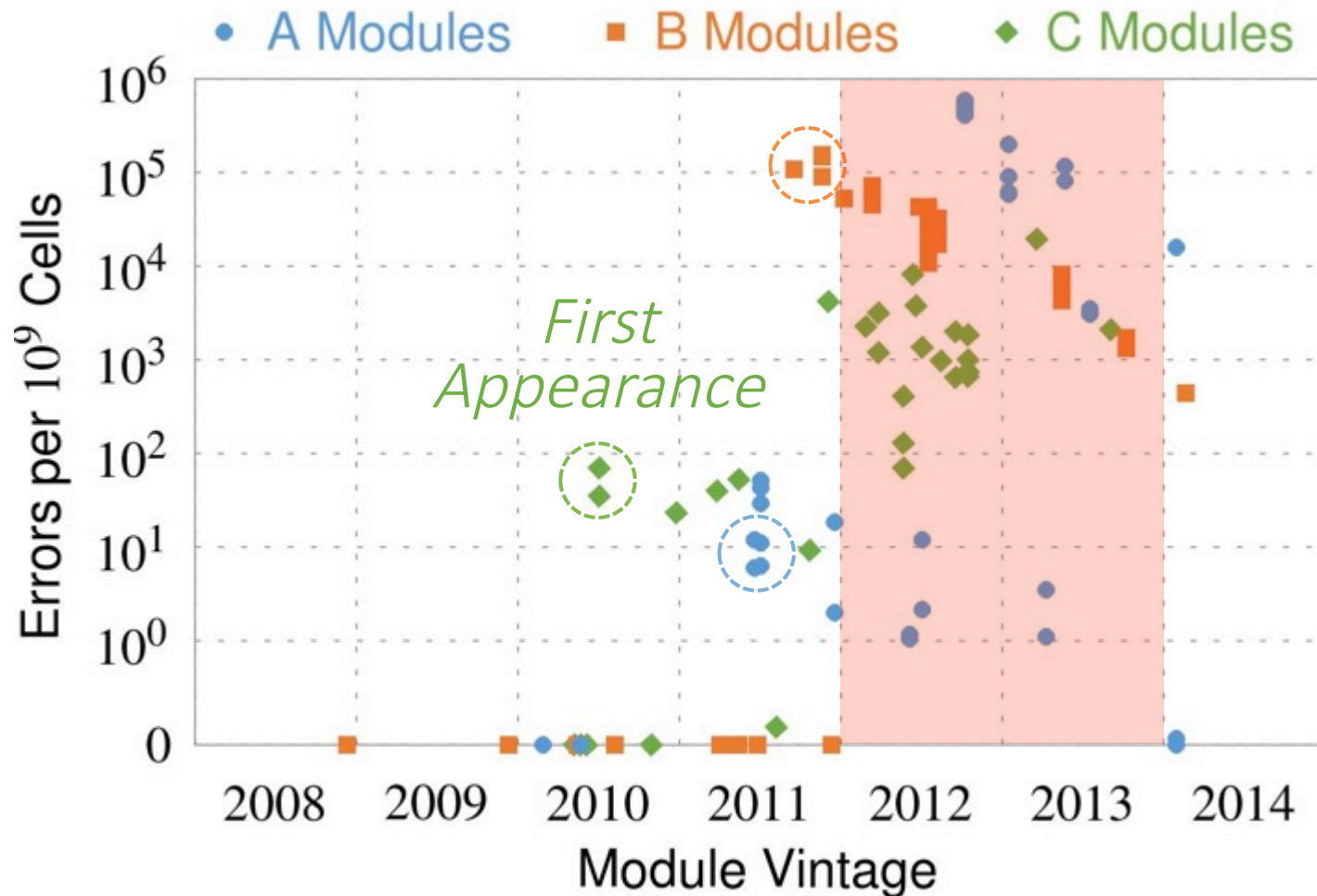


Observed Errors in Real Systems

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

- *A real reliability & security issue*
- *In a more controlled environment, we can induce as many as **ten million** disturbance errors*

Errors *vs.* Vintage



All modules from 2012–2013 are vulnerable

Experimental DRAM Testing Infrastructure



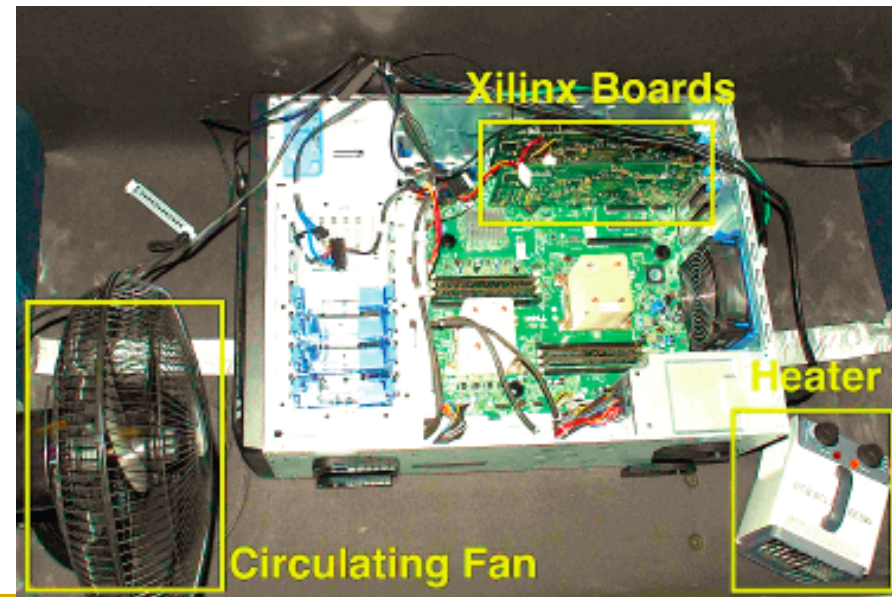
An Experimental Study of Data Retention Behavior in Modern DRAM Devices: Implications for Retention Time Profiling Mechanisms (Liu et al., ISCA 2013)

The Efficacy of Error Mitigation Techniques for DRAM Retention Failures: A Comparative Experimental Study (Khan et al., SIGMETRICS 2014)

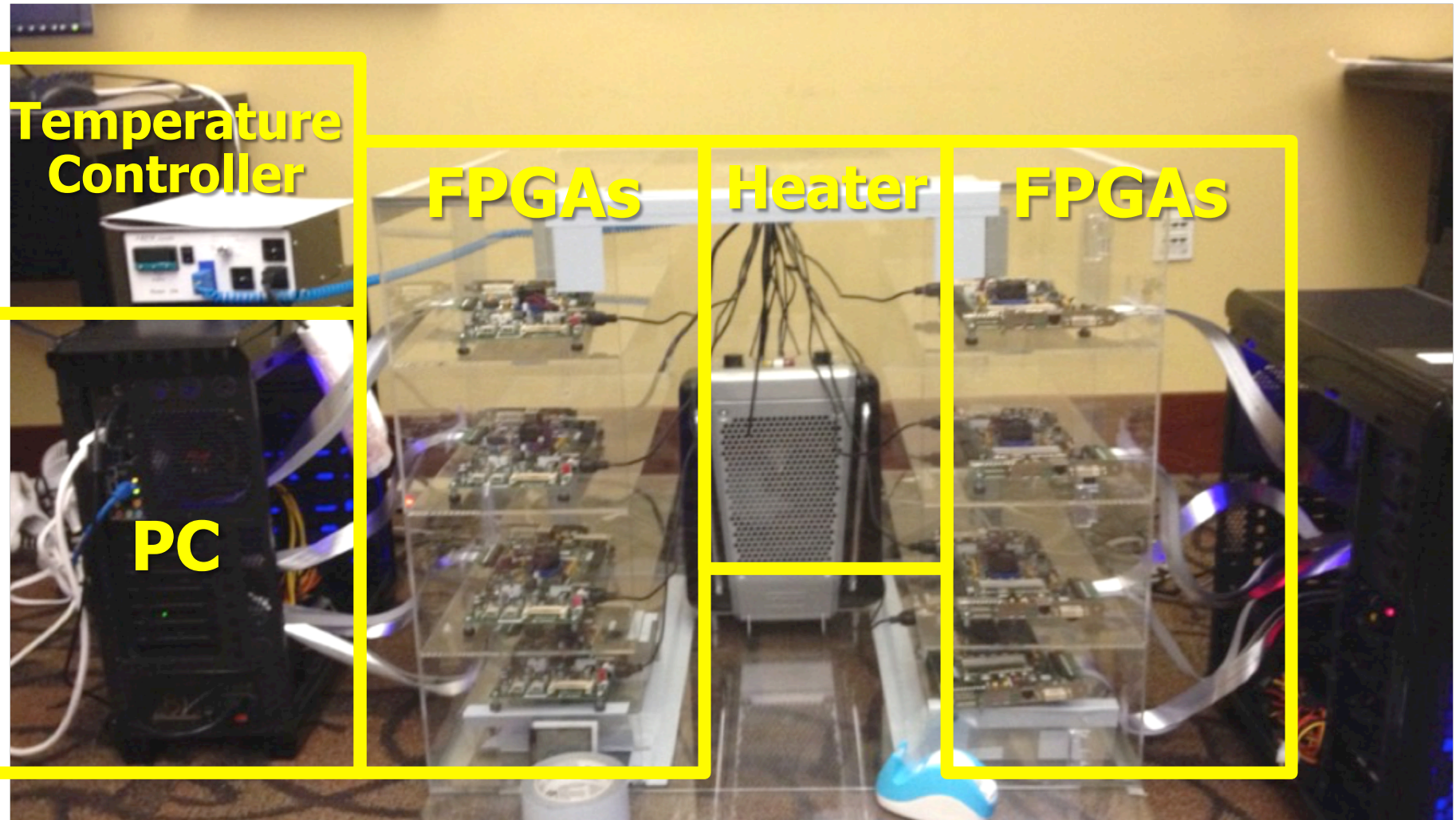
Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors (Kim et al., ISCA 2014)

Adaptive-Latency DRAM: Optimizing DRAM Timing for the Common-Case (Lee et al., HPCA 2015)

AVATAR: A Variable-Retention-Time (VRT) Aware Refresh for DRAM Systems (Qureshi et al., DSN 2015)



Experimental Infrastructure (DRAM)



One Can Take Over an Otherwise-Secure System

Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors

Abstract. Memory isolation is a key property of a reliable and secure computing system — an access to one memory address should not have unintended side effects on data stored in other addresses. However, as DRAM process technology

Project Zero

Flipping Bits in Memory Without Accessing Them:
An Experimental Study of DRAM Disturbance Errors
(Kim et al., ISCA 2014)

News and updates from the Project Zero team at Google

Exploiting the DRAM rowhammer bug to
gain kernel privileges (Seaborn, 2015)

Monday, March 9, 2015

Exploiting the DRAM rowhammer bug to gain kernel privileges

RowHammer Security Attack Example

- “Rowhammer” is a problem with some recent DRAM devices in which repeatedly accessing a row of memory can cause bit flips in adjacent rows (Kim et al., ISCA 2014).
 - Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors (Kim et al., ISCA 2014)
- We tested a selection of laptops and found that a subset of them exhibited the problem.
- We built two working privilege escalation exploits that use this effect.
 - Exploiting the DRAM rowhammer bug to gain kernel privileges (Seaborn, 2015)
- One exploit uses rowhammer-induced bit flips to gain kernel privileges on x86-64 Linux when run as an unprivileged userland process.
- When run on a machine vulnerable to the rowhammer problem, the process was able to induce bit flips in page table entries (PTEs).
- It was able to use this to gain write access to its own page table, and hence gain read-write access to all of physical memory.

Security Implications



It's like breaking into an apartment by repeatedly slamming a neighbor's door until the vibrations open the door you were after

Apple's Patch for RowHammer

- <https://support.apple.com/en-gb/HT204934>

Available for: OS X Mountain Lion v10.8.5, OS X Mavericks v10.9.5

Impact: A malicious application may induce memory corruption to escalate privileges

Description: A disturbance error, also known as Rowhammer, exists with some DDR3 RAM that could have led to memory corruption. This issue was mitigated by increasing memory refresh rates.

CVE-ID

CVE-2015-3693 : Mark Seaborn and Thomas Dullien of Google, working from original research by Yoongu Kim et al (2014)

HP and Lenovo released similar patches

Challenge and Opportunity

**Reliability
(and Security)**

Departing From “Business as Usual”

More Intelligent Memory Controllers

Online Detection and Fixing of DRAM Errors

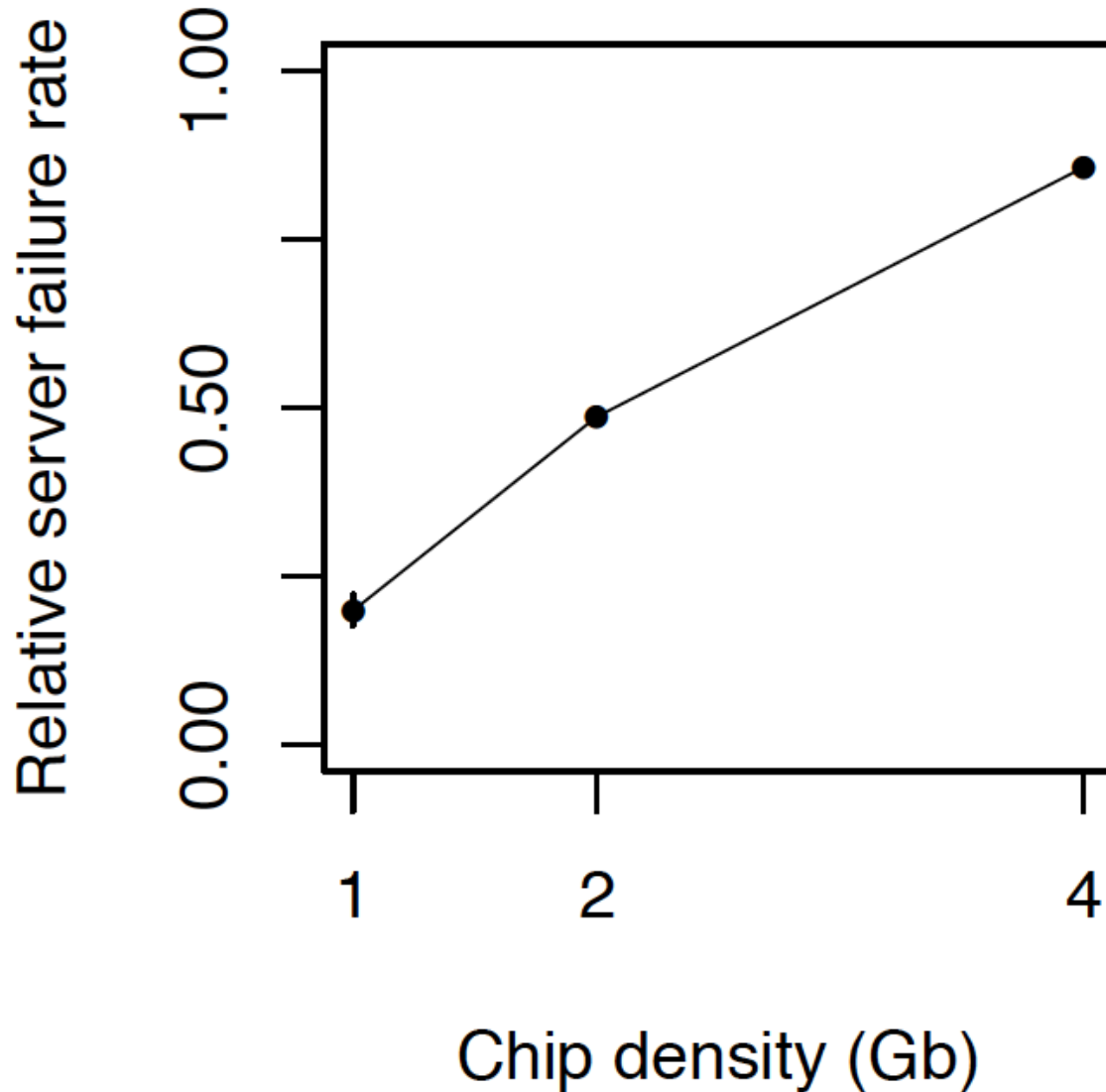
Large-Scale Failure Analysis of DRAM Chips

- Analysis and modeling of memory errors found in all of Facebook's server fleet
- Justin Meza, Qiang Wu, Sanjeev Kumar, and Onur Mutlu,
"Revisiting Memory Errors in Large-Scale Production Data Centers: Analysis and Modeling of New Trends from the Field"
Proceedings of the
45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), Rio de Janeiro, Brazil, June 2015.
[[Slides \(pptx\)](#)] [[pdf](#)] [[DRAM Error Model](#)]

Revisiting Memory Errors in Large-Scale Production Data Centers: Analysis and Modeling of New Trends from the Field

Justin Meza Qiang Wu* Sanjeev Kumar* Onur Mutlu
Carnegie Mellon University * Facebook, Inc.

DRAM Reliability Reducing



*Intuition:
quadratic
increase in
capacity*

Aside: Flash Error Analysis in the Field

- First large-scale field study of flash memory errors
- Justin Meza, Qiang Wu, Sanjeev Kumar, and Onur Mutlu,
"A Large-Scale Study of Flash Memory Errors in the Field"
*Proceedings of the
ACM International Conference on Measurement and Modeling of
Computer Systems (SIGMETRICS)*, Portland, OR, June 2015.
[[Slides \(pptx\)](#)] [[pdf](#)] [[Coverage at ZDNet](#)]

A Large-Scale Study of Flash Memory Failures in the Field

Justin Meza
Carnegie Mellon University
meza@cmu.edu

Qiang Wu
Facebook, Inc.
qwu@fb.com

Sanjeev Kumar
Facebook, Inc.
skumar@fb.com

Onur Mutlu
Carnegie Mellon University
onur@cmu.edu

Recap: The DRAM Scaling Problem

DRAM Process Scaling Challenges

❖ Refresh

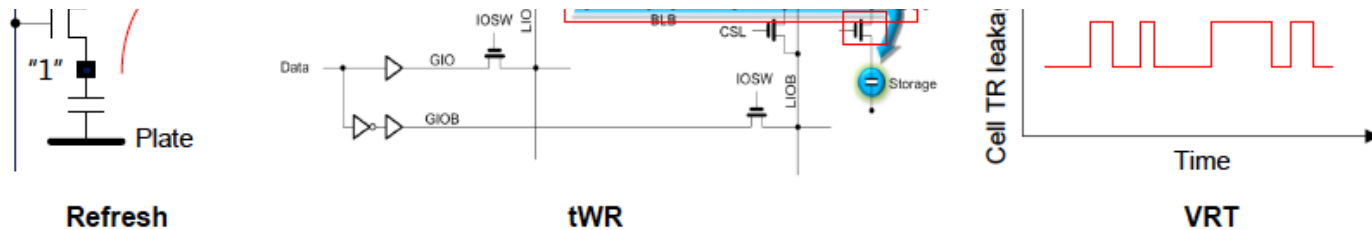
- Difficult to build high-aspect ratio cell capacitors decreasing cell capacitance

THE MEMORY FORUM 2014

Co-Architecting Controllers and DRAM to Enhance DRAM Process Scaling

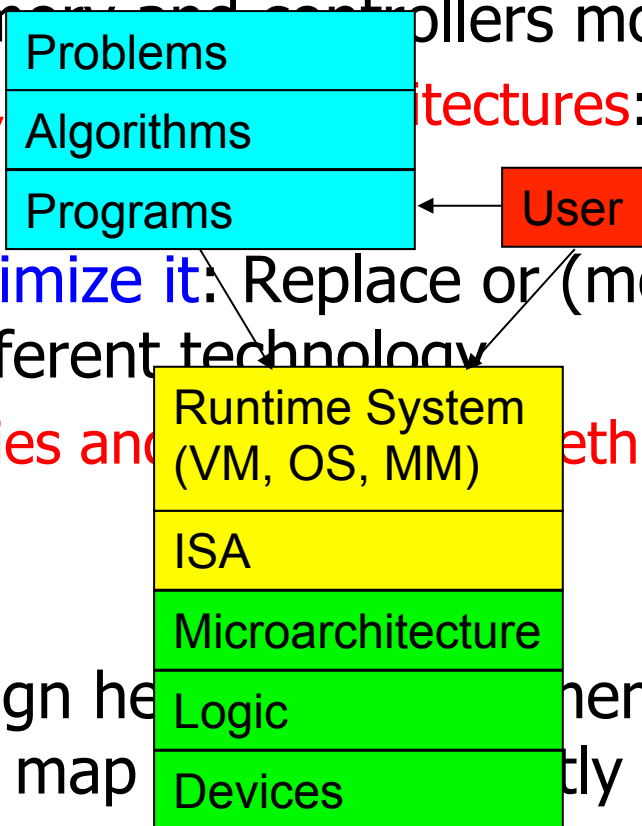
Uksong Kang, Hak-soo Yu, Churoo Park, *Hongzhong Zheng,
**John Halbert, **Kuljit Bains, SeongJin Jang, and Joo Sun Choi

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



How Do We Solve The Problem?

- **Fix it:** Make memory and controllers more intelligent
 - **New interfaces, architectures:** system-mem codesign
- **Eliminate or minimize it:** Replace or (more likely) augment DRAM with a different technology
 - **New technologies and storage**
- **Embrace it:** Design heterogeneous memories (none of which are perfect) and map applications across them
 - **New models for data management and maybe usage**



Solutions (to memory scaling) require software/hardware/device cooperation

Solution 1: New Memory Architectures

- Overcome memory shortcomings with
 - ❑ Memory-centric system design
 - ❑ Novel memory architectures, interfaces, functions
 - ❑ Better waste management (efficient utilization)
- Key issues to tackle
 - ❑ Enable reliability at low cost
 - ❑ Reduce energy
 - ❑ Improve latency and bandwidth
 - ❑ Reduce waste (capacity, bandwidth, latency)
 - ❑ Enable computation close to data

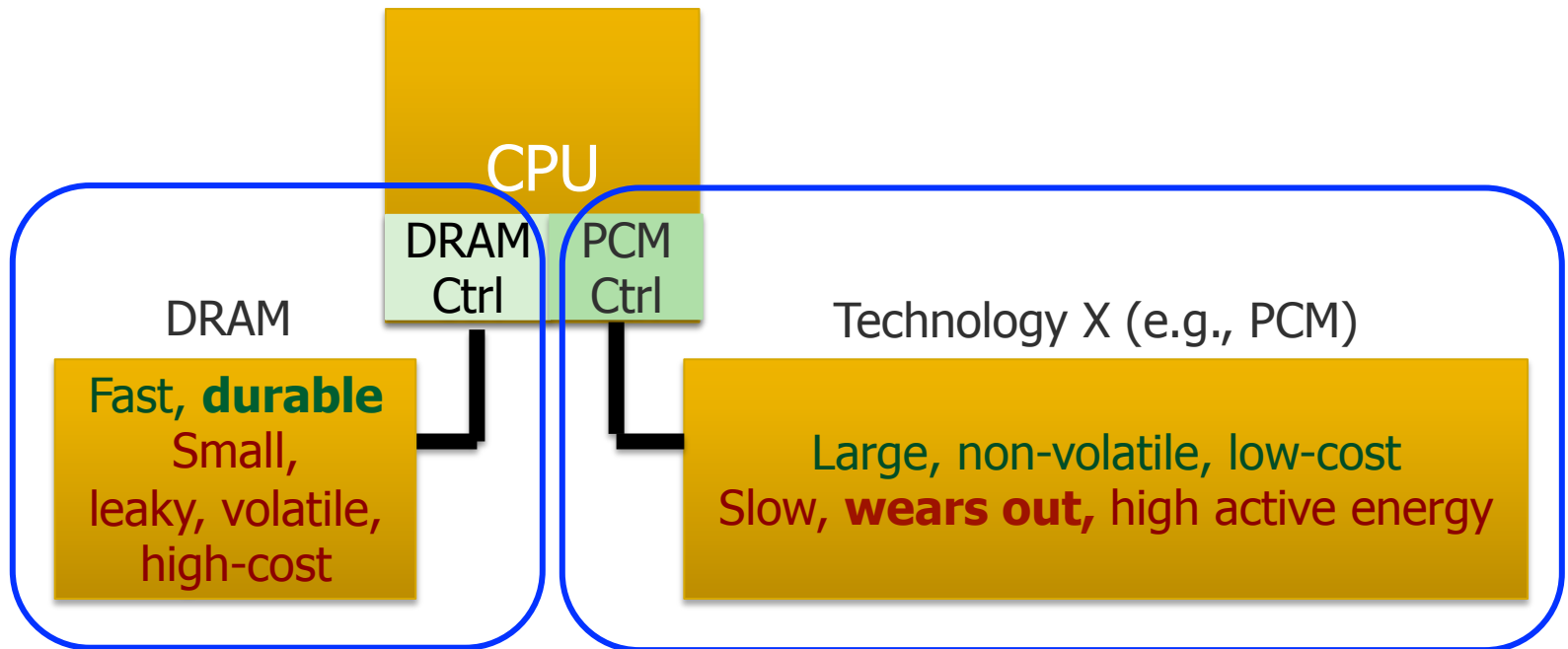
Solution 1: New Memory Architectures

- 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.
- Lee+, "Adaptive-Latency DRAM: Optimizing DRAM Timing for the Common-Case," HPCA 2015.
- Qureshi+, "AVATAR: A Variable-Retention-Time (VRT) Aware Refresh for DRAM Systems," DSN 2015.
- Meza+, "Revisiting Memory Errors in Large-Scale Production Data Centers: Analysis and Modeling of New Trends from the Field," DSN 2015.
- Kim+, "Ramulator: A Fast and Extensible DRAM Simulator," IEEE CAL 2015.
- Seshadri+, "Fast Bulk Bitwise AND and OR in DRAM," IEEE CAL 2015.
- Ahn+, "A Scalable Processing-in-Memory Accelerator for Parallel Graph Processing," ISCA 2015.
- Ahn+, "PIM-Enabled Instructions: A Low-Overhead, Locality-Aware Processing-in-Memory Architecture," ISCA 2015.
- Lee+, "Decoupled Direct Memory Access: Isolating CPU and IO Traffic by Leveraging a Dual-Data-Port DRAM," PACT 2015.
- Seshadri+, "Gather-Scatter DRAM: In-DRAM Address Translation to Improve the Spatial Locality of Non-unit Strided Accesses," MICRO 2015.
- Lee+, "Simultaneous Multi-Layer Access: Improving 3D-Stacked Memory Bandwidth at Low Cost," TACO 2016.
- Hasan+, "ChargeCache: Reducing DRAM Latency by Exploiting Row Access Locality," HPCA 2016.
- Chang+, "Low-Cost Inter-Linked Subarrays (LISA): Enabling Fast Inter-Subarray Data Migration in DRAM," HPCA 2016.
- Chang+, "Understanding Latency Variation in Modern DRAM Chips Experimental Characterization, Analysis, and Optimization," SIGMETRICS 2016.
- Khan+, "PARBOR: An Efficient System-Level Technique to Detect Data Dependent Failures in DRAM," DSN 2016.
- Hsieh+, "Transparent Offloading and Mapping (TOM): Enabling Programmer-Transparent Near-Data Processing in GPU Systems," ISCA 2016.
- Hashemi+, "Accelerating Dependent Cache Misses with an Enhanced Memory Controller," ISCA 2016.
- Boroumand+, "LazyPIM: An Efficient Cache Coherence Mechanism for Processing-in-Memory," IEEE CAL 2016.
- Pattnaik+, "Scheduling Techniques for GPU Architectures with Processing-In-Memory Capabilities," PACT 2016.
- Hsieh+, "Accelerating Pointer Chasing in 3D-Stacked Memory: Challenges, Mechanisms, Evaluation," ICCD 2016.
- Hashemi+, "Continuous Runahead: Transparent Hardware Acceleration for Memory Intensive Workloads," MICRO 2016.
- 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.
 - Pekhimenko+, "Exploiting Compressed Block Size as an Indicator of Future Reuse," HPCA 2015.
 - Vijaykumar+, "A Case for Core-Assisted Bottleneck Acceleration in GPUs: Enabling Flexible Data Compression with Assist Warps," ISCA 2015.
 - Pekhimenko+, "Toggle-Aware Bandwidth Compression for GPUs," HPCA 2016.

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+, “Architecting Phase Change Memory as a Scalable DRAM Alternative,” ISCA’09, CACM’10, Micro’10.
- Meza+, “Enabling Efficient and Scalable Hybrid Memories,” IEEE Comp. Arch. Letters 2012.
- Yoon, Meza+, “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.
- Lu+, “Loose Ordering Consistency for Persistent Memory,” ICCD 2014.
- Zhao+, “FIRM: Fair and High-Performance Memory Control for Persistent Memory Systems,” MICRO 2014.
- Yoon, Meza+, “Efficient Data Mapping and Buffering Techniques for Multi-Level Cell Phase-Change Memories,” ACM TACO 2014.
- Ren+, “ThyNVM: Enabling Software-Transparent Crash Consistency in Persistent Memory Systems,” MICRO 2015.

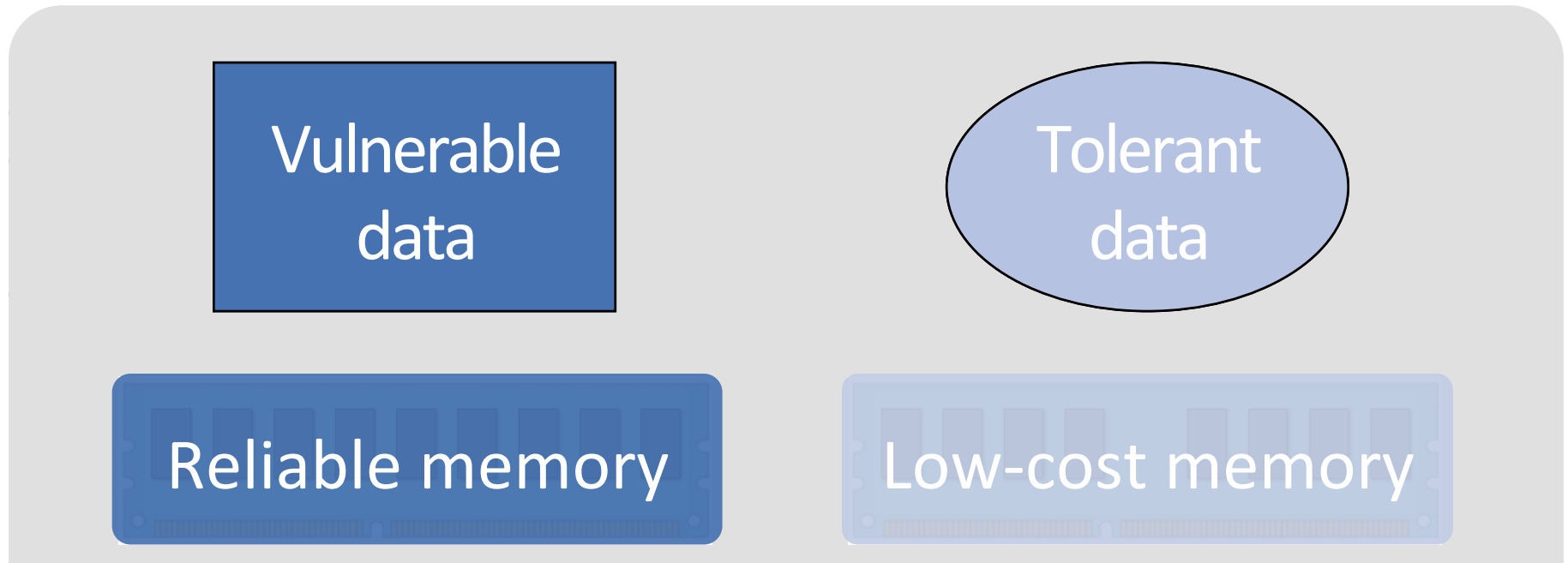
Solution 3: Hybrid Memory Systems



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.

Exploiting Memory Error Tolerance with Hybrid Memory Systems



On Microsoft's Web Search workload

Reduces server hardware **cost** by **4.7 %**

Achieves single server **availability** target of **99.90 %**

Heterogeneous-Reliability Memory [DSN 2014]

Challenge and Opportunity

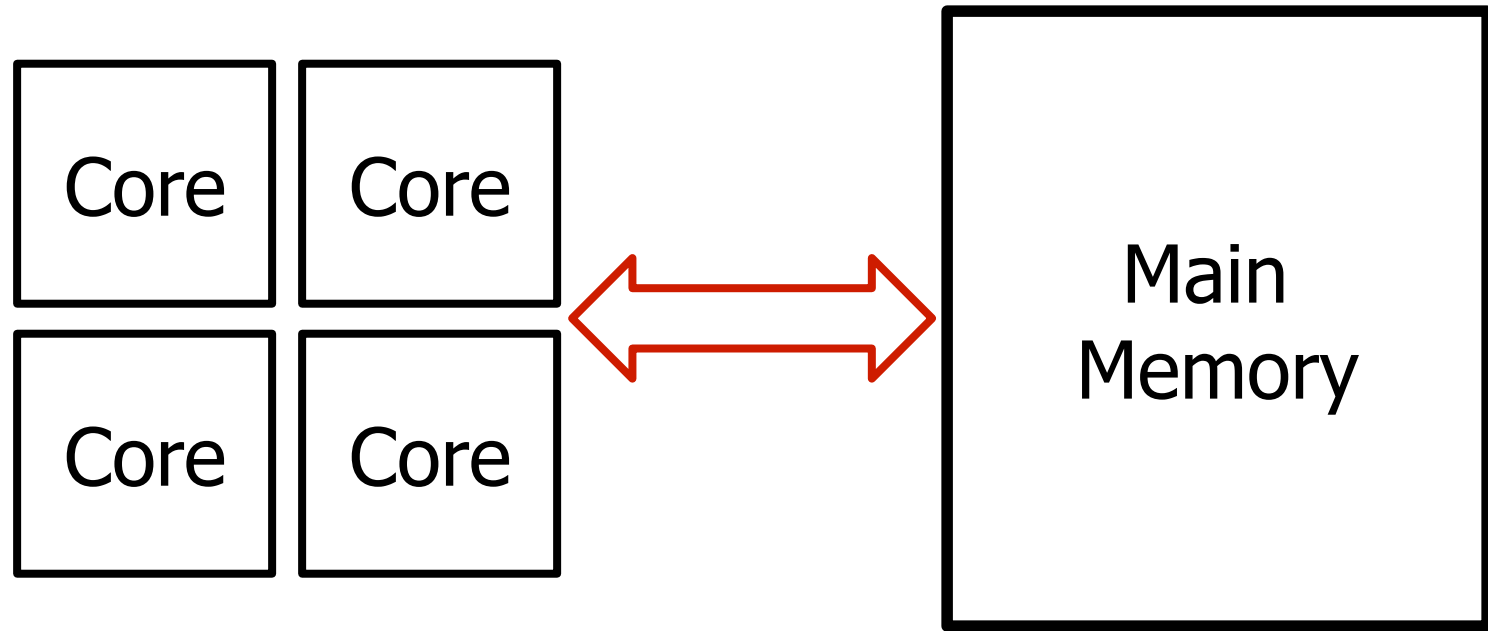
Providing the Best of
Multiple Metrics

Departing From “Business as Usual”

Heterogeneous Memory Systems

Configurable Memory Systems

An Orthogonal Issue: Memory Interference

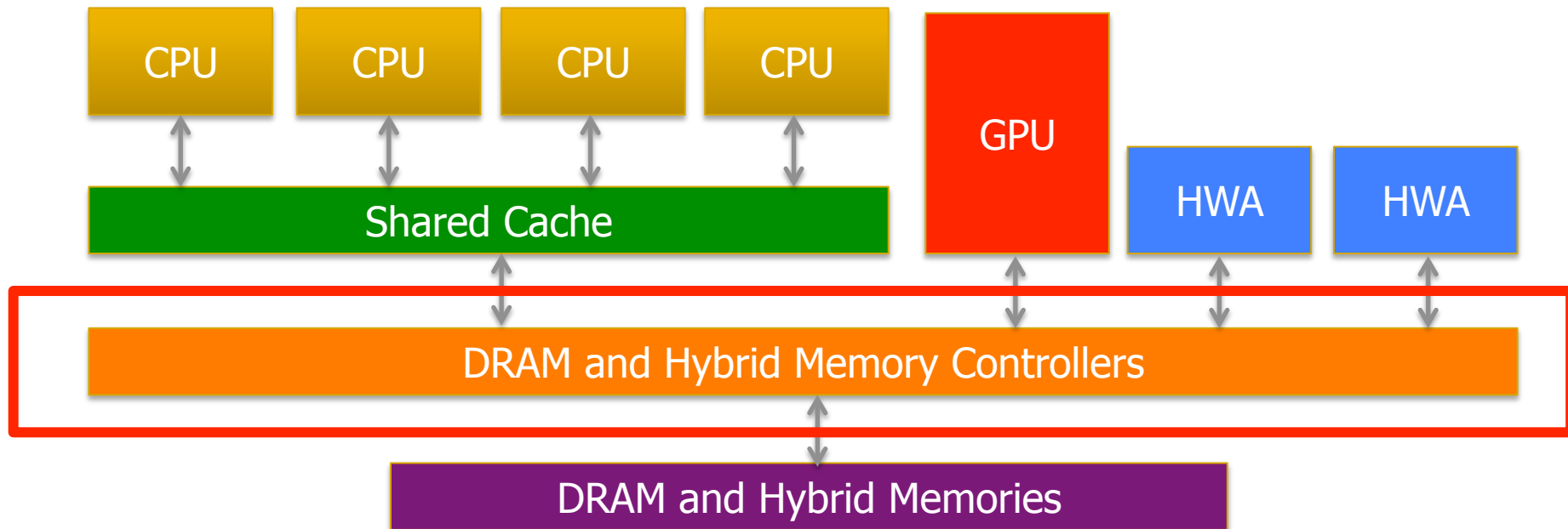


Cores' interfere with each other when accessing shared main memory

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 systems can provide predictable performance and higher efficiency

Goal: 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, heterogeneous agents, and hybrid memory/storage
- Approach:
 - Develop techniques/models to accurately estimate the performance loss 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
- Subramanian et al., "MISE: Providing Performance Predictability and Improving Fairness in Shared Main Memory Systems," HPCA 2013.
- Subramanian et al., "The Application Slowdown Model," MICRO 2015.

Challenge and Opportunity

Strong
Memory Service
Guarantees

Departing From “Business as Usual”

Predictable Memory Management

Programmable Memory Systems

Some Promising Directions

- New memory architectures
 - Rethinking memory's role and functions
 - Memory-centric system design
- Enabling and exploiting emerging NVM technologies
 - Hybrid memory systems
 - Single-level memory and storage
- System-level memory/storage QoS
 - Predictable systems with configurable QoS

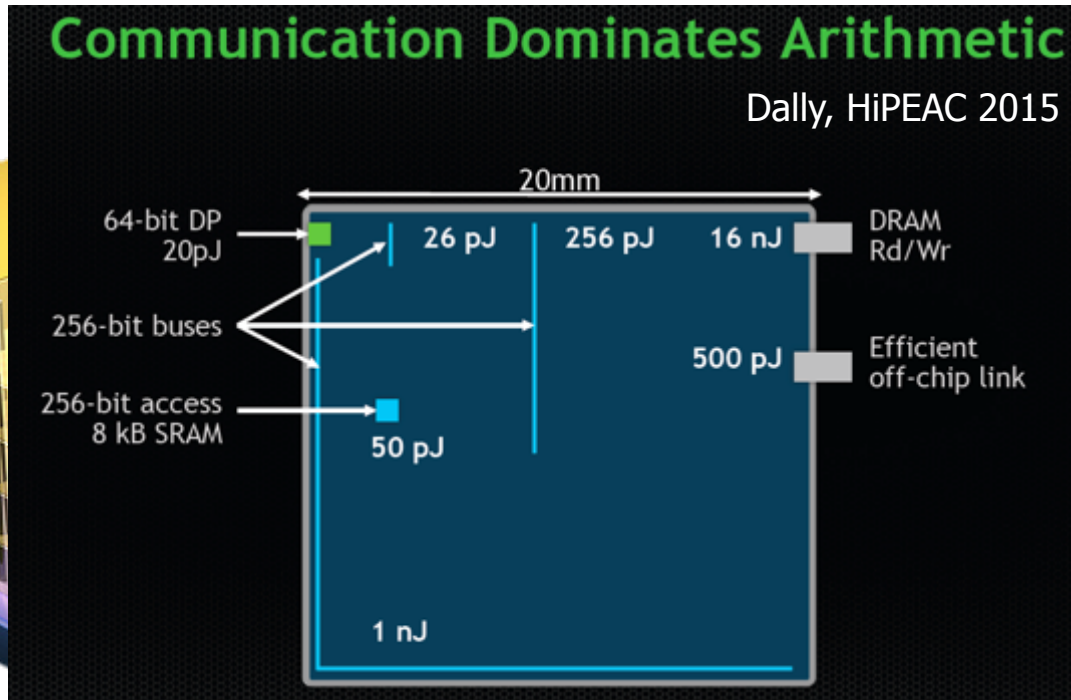
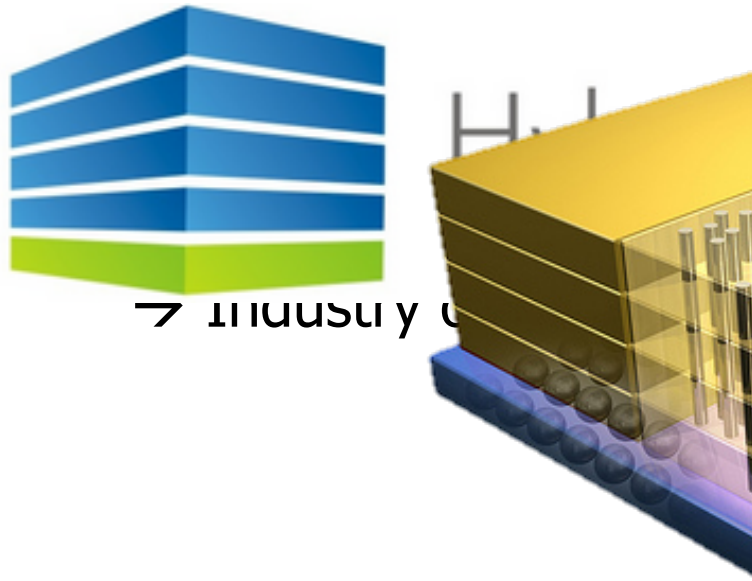
Agenda

- Major Trends Affecting Main Memory
- The Memory Scaling Problem and Solution Directions
 - New Memory Architectures
 - Enabling Emerging Technologies
- Cross-Cutting Principles
- Summary

Rethinking Memory Architecture

- Compute-capable memory
- Refresh
- Reliability
- Latency
- Bandwidth
- Energy
- Memory Compression

Why In-Memory Computation Today?



■ Pull from Systems and Applications

- ❑ Data access is a major system and application bottleneck
- ❑ Systems are energy limited
- ❑ Data movement much more energy-hungry than computation

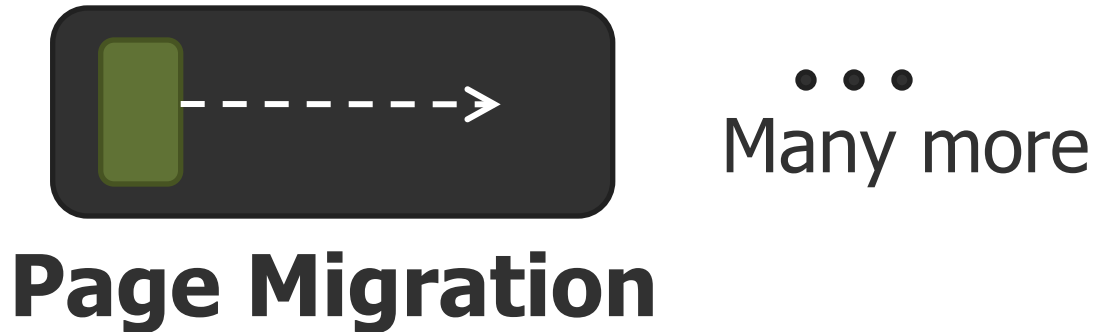
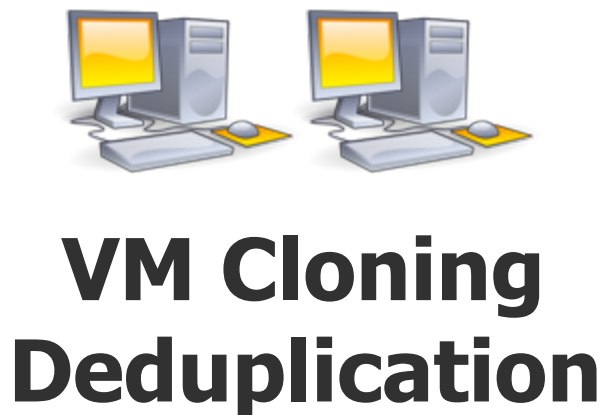
Two Approaches to In-Memory Processing

- 1. Minimally change DRAM to enable simple yet powerful computation primitives
 - RowClone: Fast and Efficient In-DRAM Copy and Initialization of Bulk Data (Seshadri et al., MICRO 2013)
 - Fast Bulk Bitwise AND and OR in DRAM (Seshadri et al., IEEE CAL 2015)

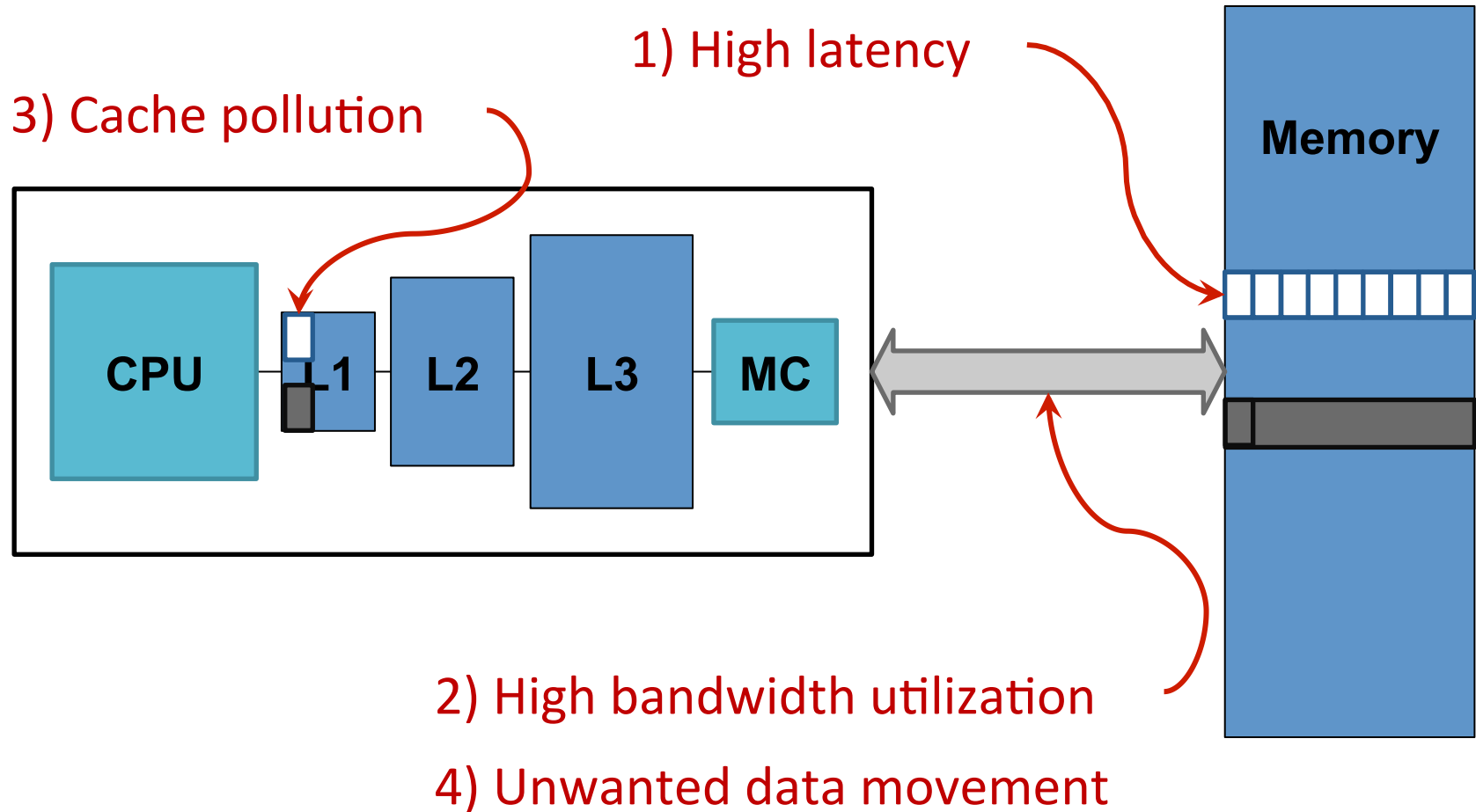
- 2. Exploit the control logic in 3D-stacked memory to enable more comprehensive computation near memory
 - PIM-Enabled Instructions: A Low-Overhead, Locality-Aware Processing-in-Memory Architecture (Ahn et al., ISCA 2015)
 - A Scalable Processing-in-Memory Accelerator for Parallel Graph Processing (Ahn et al., ISCA 2015)

Bulk Copy and Initialization

memmove & memcpy: 5% cycles in Google's datacenter [Kanev+ ISCA'15]

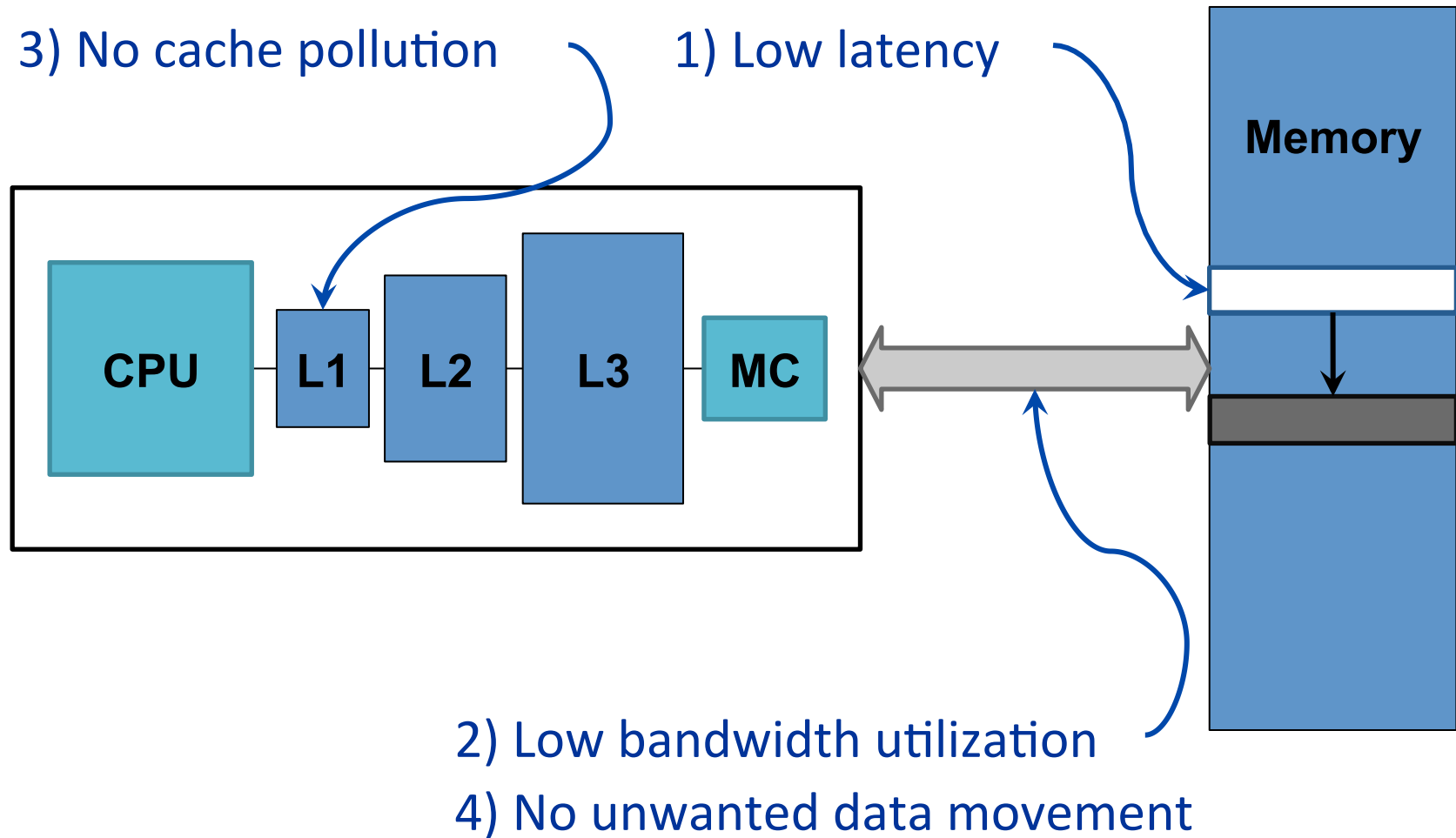


Today's Memory: Bulk Data Copy



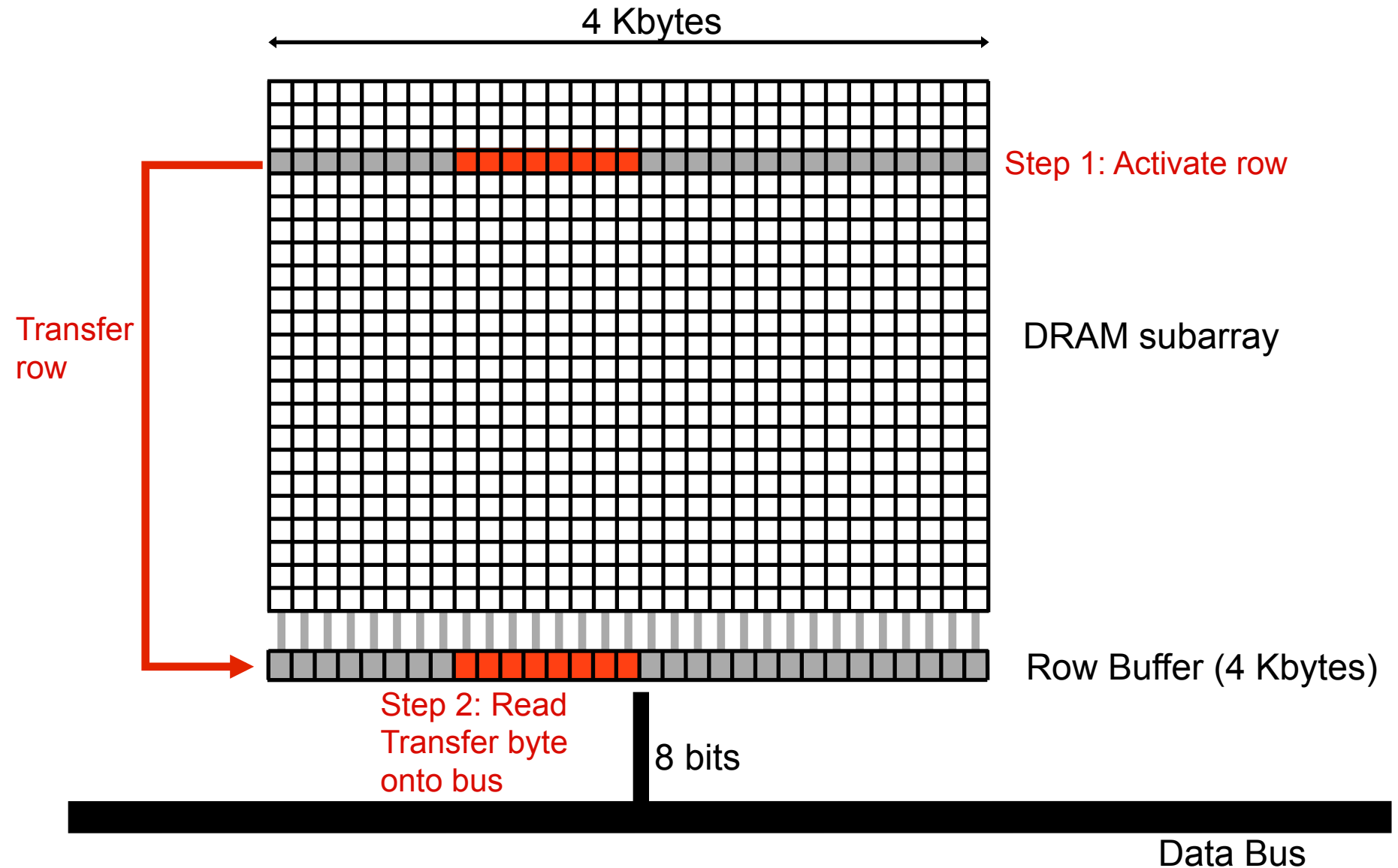
1046ns, 3.6uJ (for 4KB page copy via DMA)

Future: RowClone (In-Memory Copy)

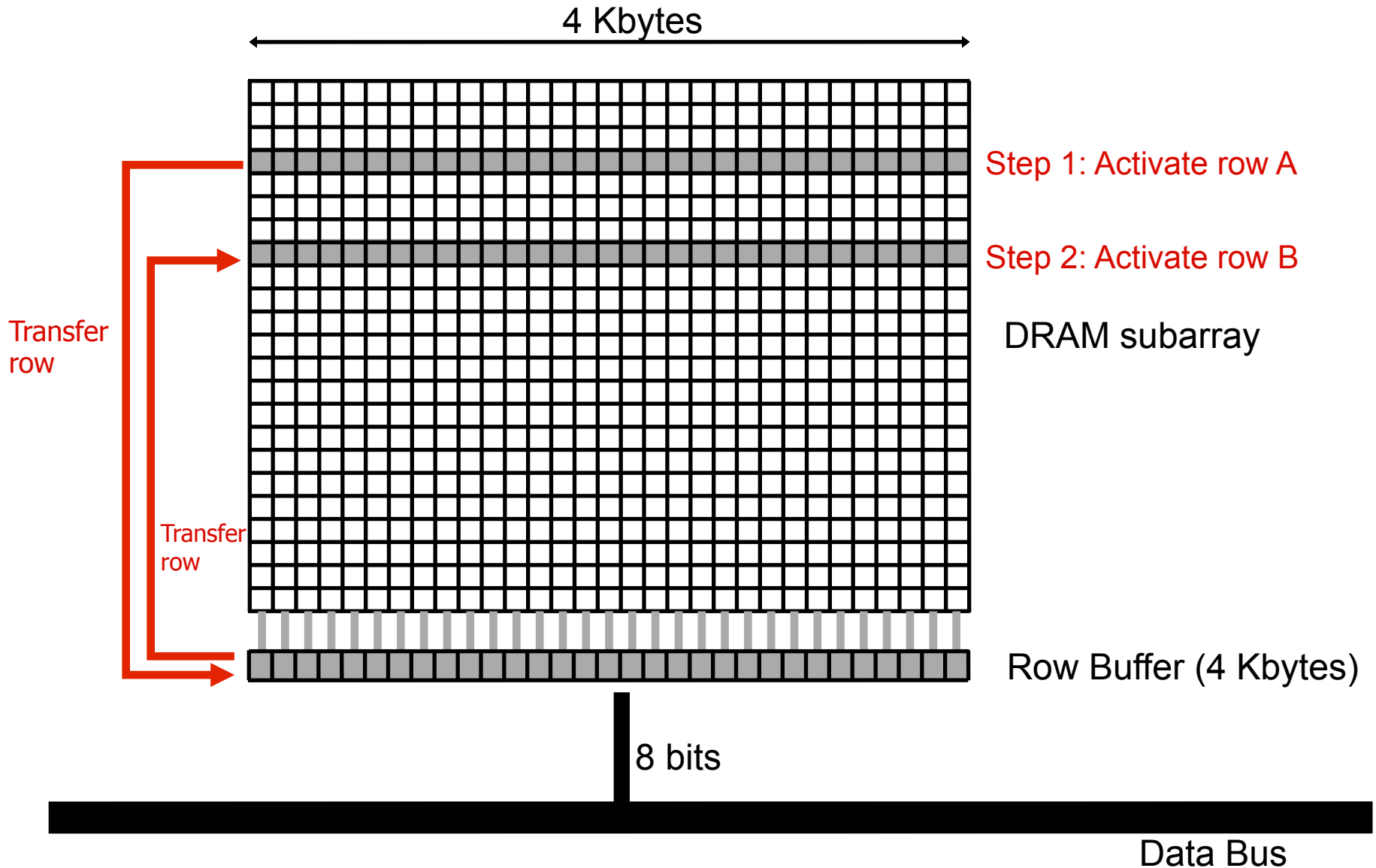


194ns, 304uJ

DRAM Subarray Operation (load one byte)

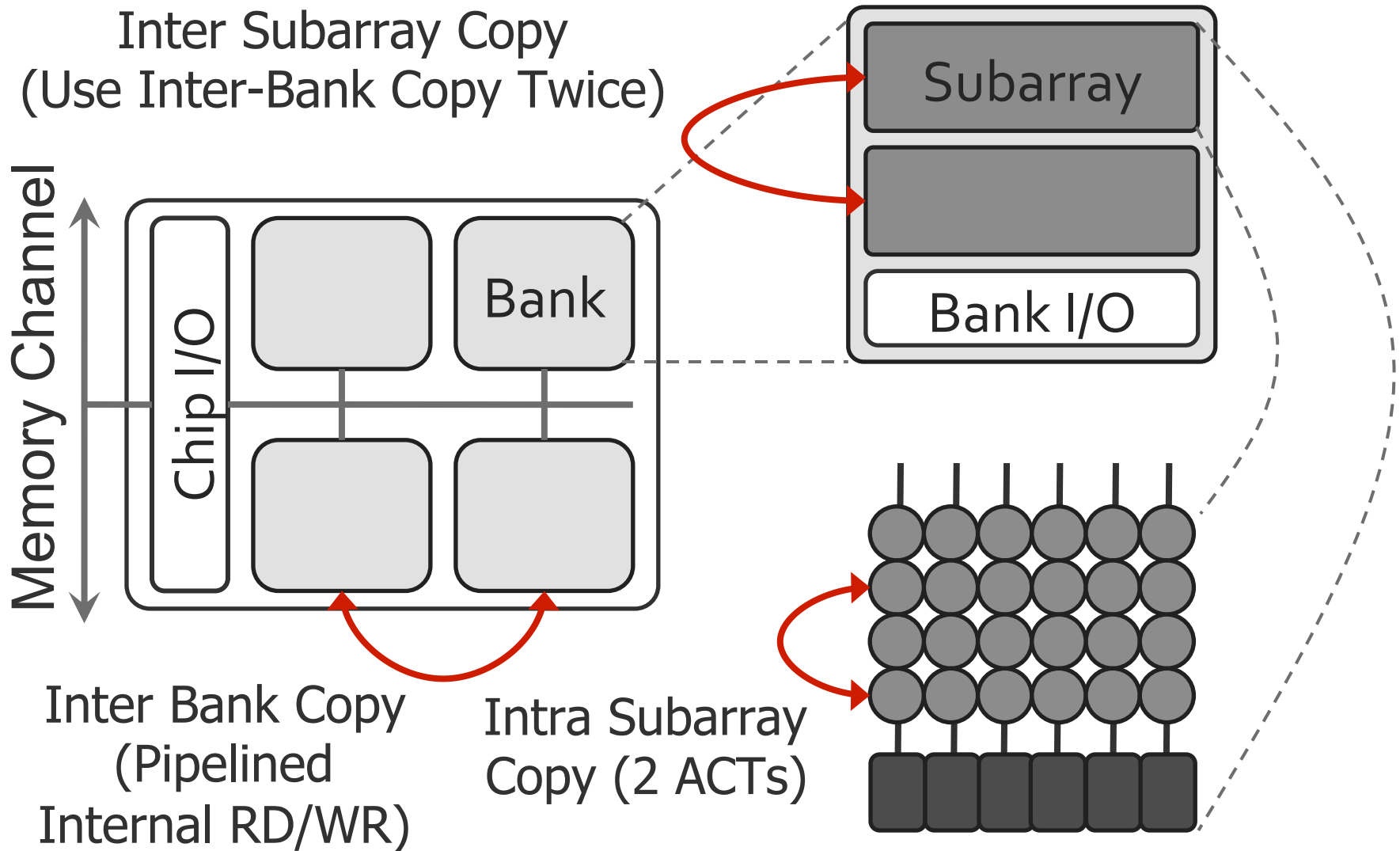


RowClone: In-DRAM Row Copy

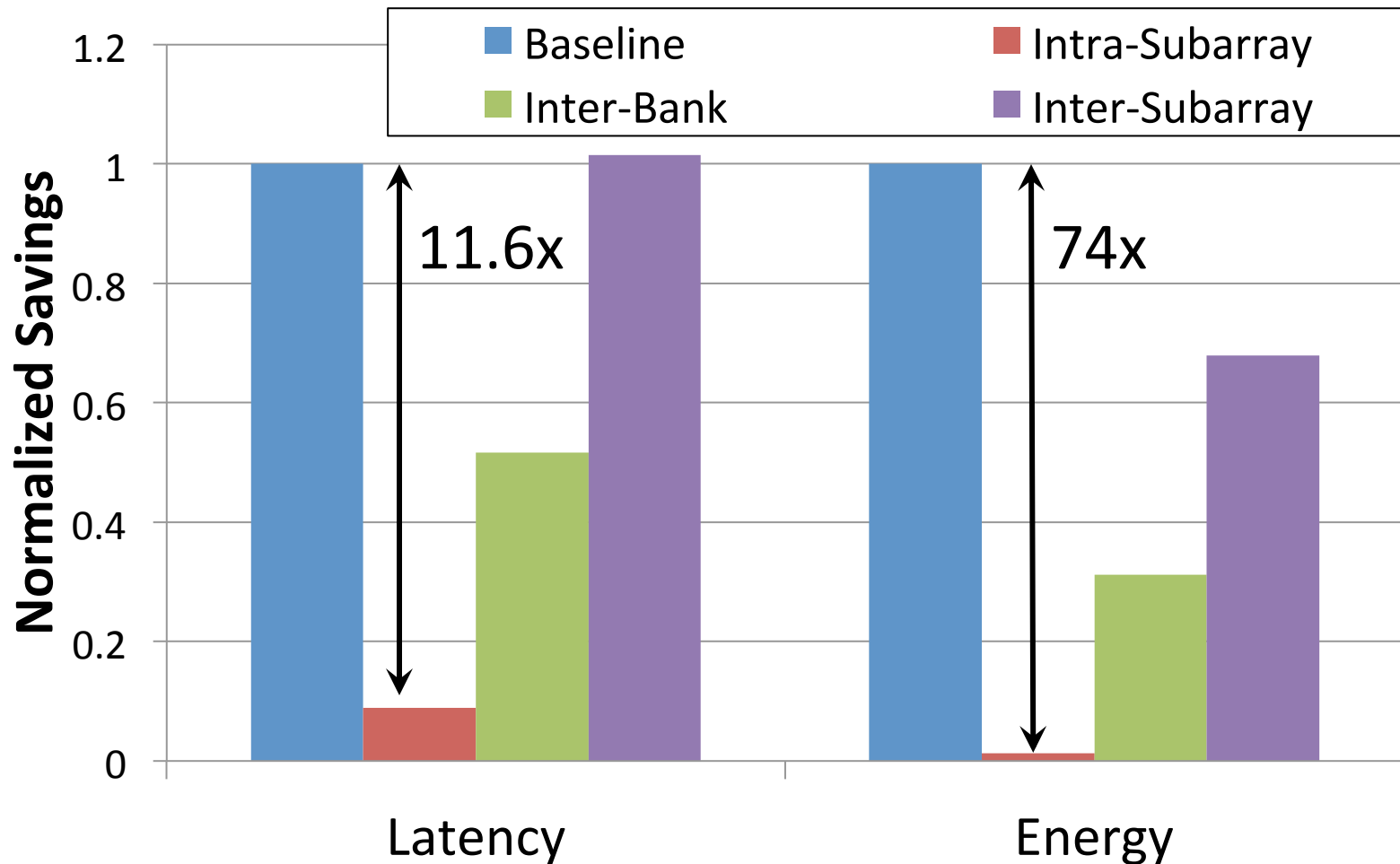


Generalized RowClone

0.01% area cost

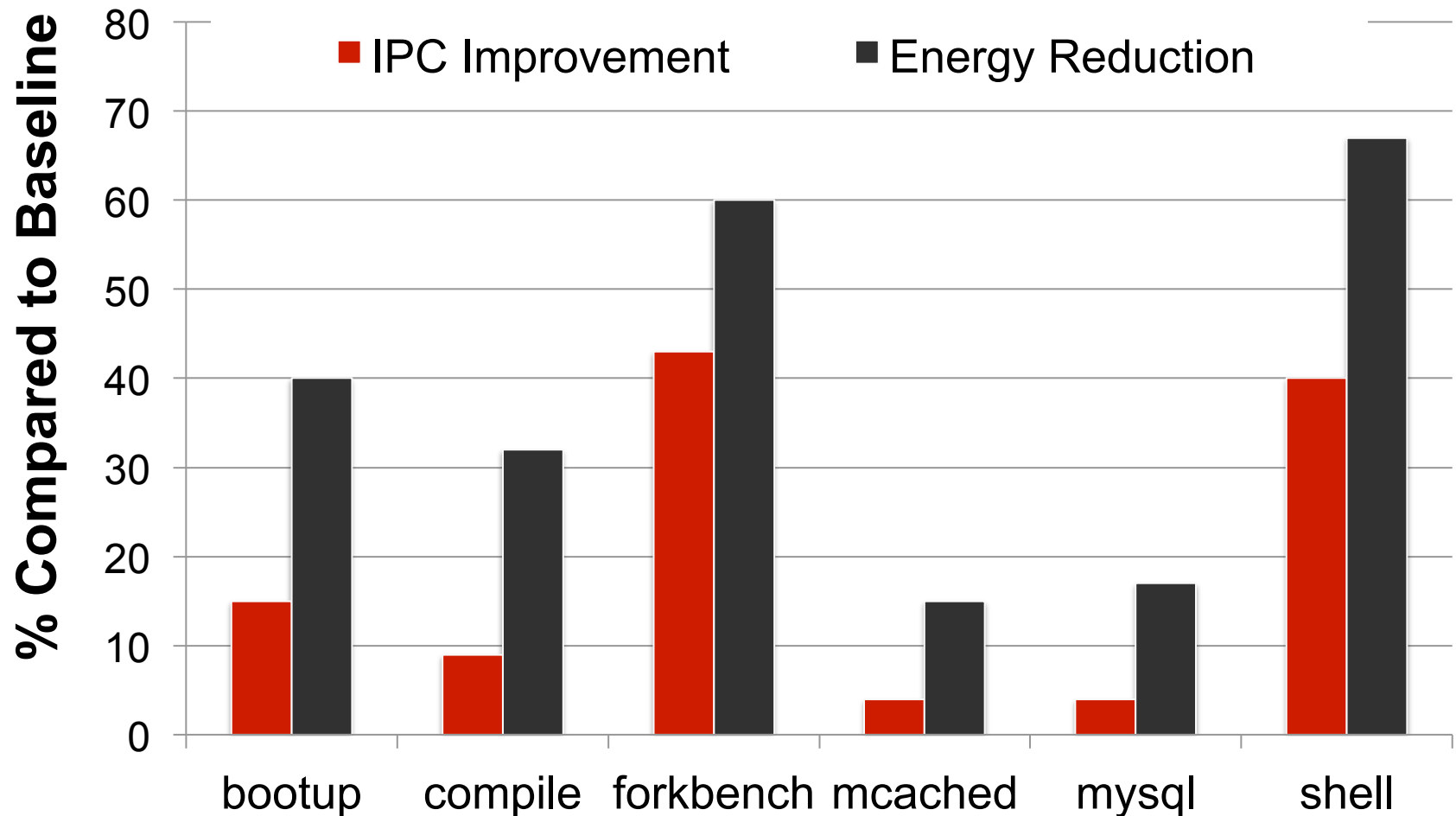


RowClone: Latency and Energy Savings

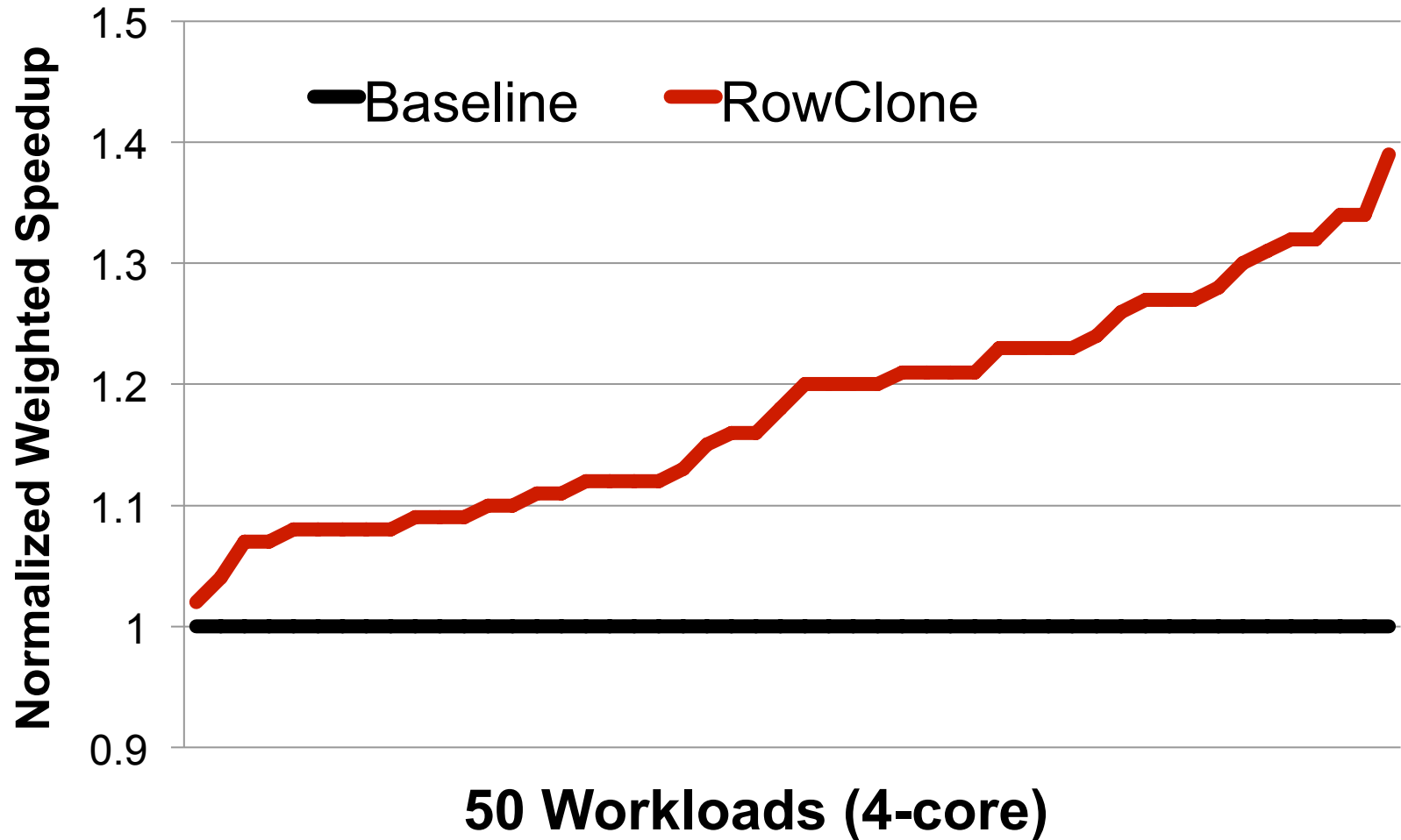


Seshadri et al., "RowClone: Fast and Efficient In-DRAM Copy and Initialization of Bulk Data," MICRO 2013.

RowClone: Application Performance



RowClone: Multi-Core Performance



End-to-End System Design

Application

How to communicate occurrences of bulk copy/ initialization across layers?

Operating System

ISA

How to ensure cache coherence?

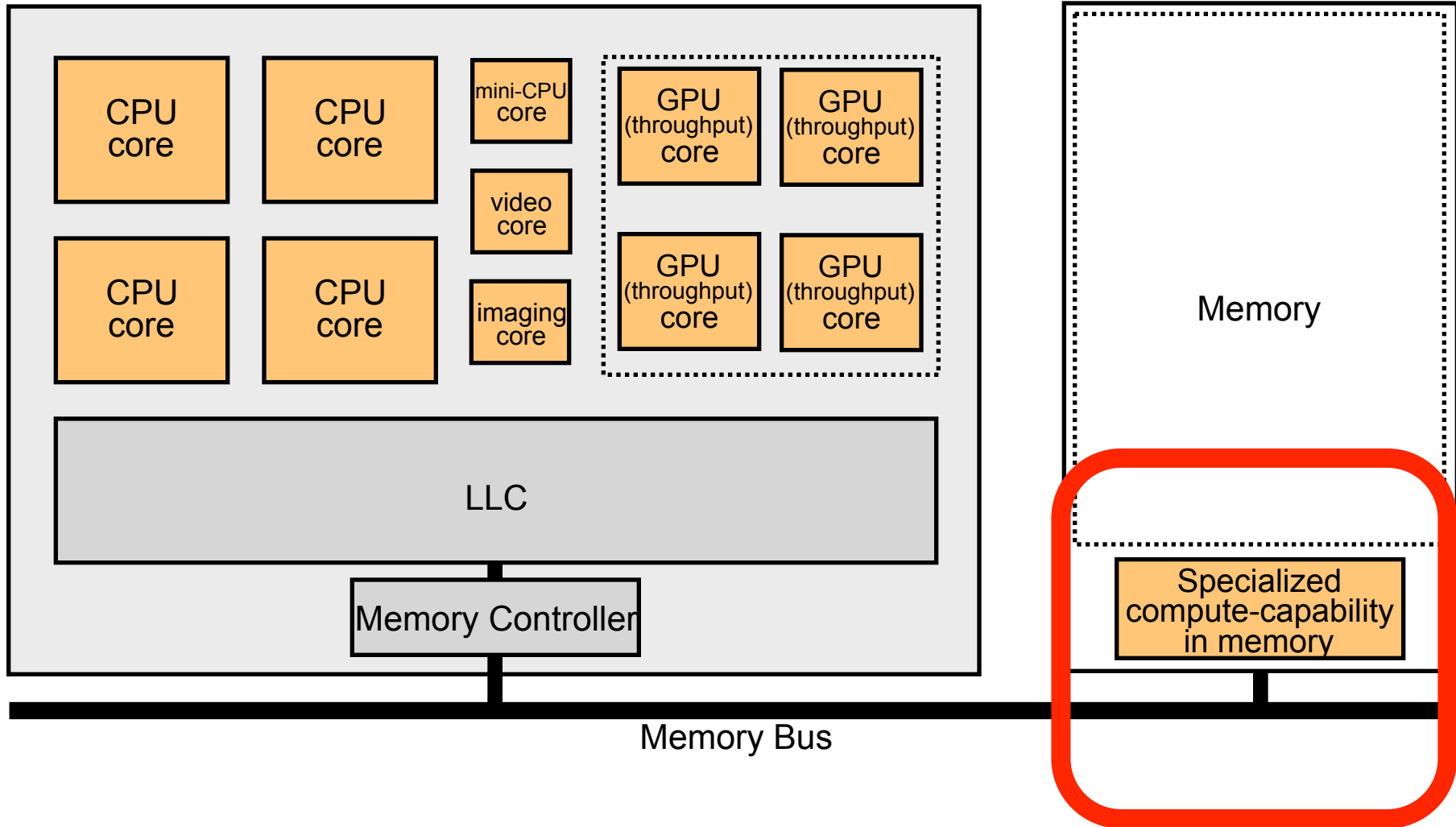
Microarchitecture

How to maximize latency and energy savings?

DRAM (RowClone)

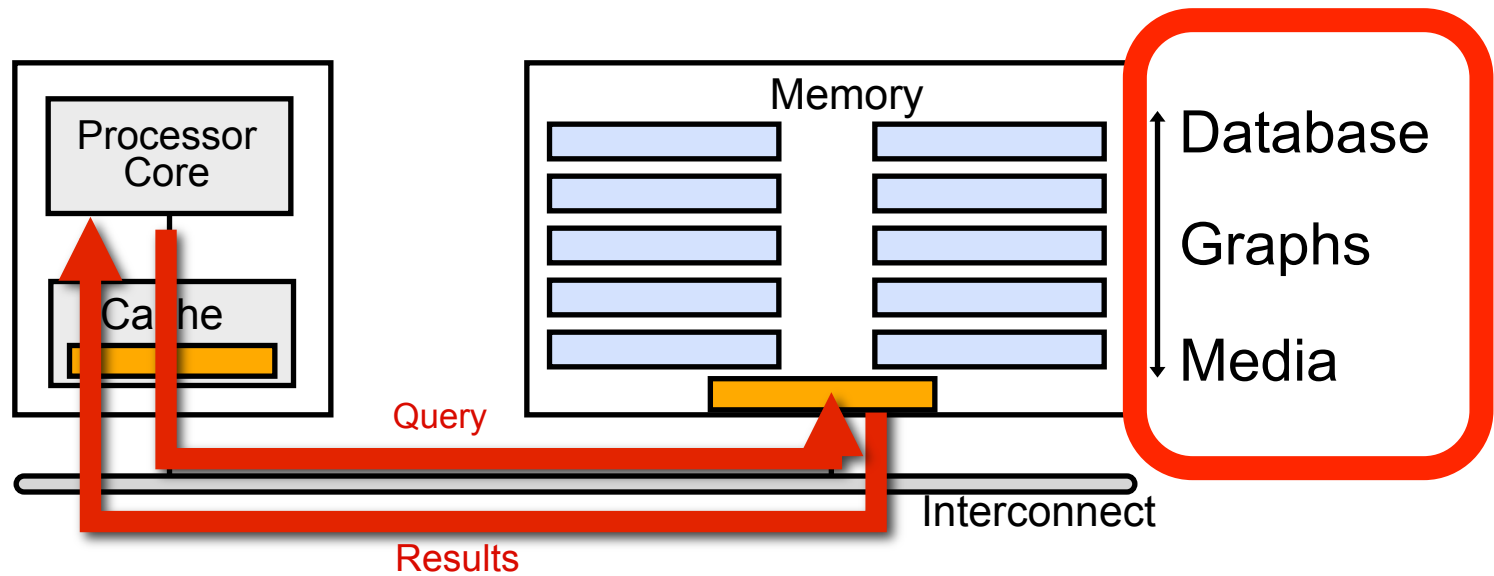
How to handle data reuse?

Goal: Ultra-Efficient Processing Near Data



Memory similar to a "conventional" accelerator

Enabling In-Memory X

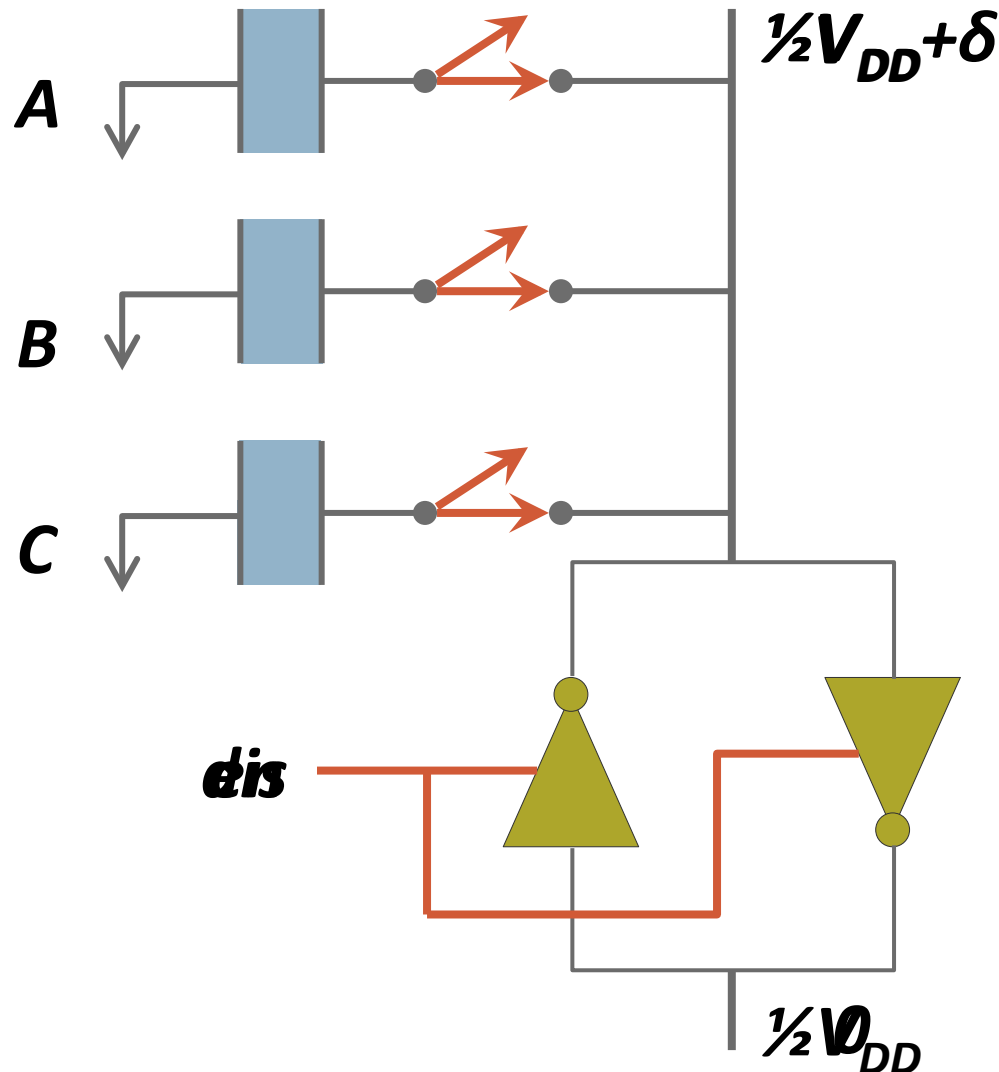


- What is a flexible and scalable memory interface?
- What is the right partitioning of computation capability?
- What is the right low-cost memory substrate?
- What memory technologies are the best enablers?
- How do we rethink/ease X algorithms/applications?

Enabling In-Memory Computation

DRAM Support	Cache Coherence	Virtual Memory Support
RowClone (MICRO 2013)	Dirty-Block Index (ISCA 2014)	Page Overlays (ISCA 2015)
In-DRAM Gather Scatter (MICRO 2015)	Non-contiguous Cache lines	Gathered Pages
In-DRAM Bitwise Operations (IEEE CAL 2015)	?	?

In-DRAM AND/OR: Triple Row Activation



Final State
 $AB + BC + AC$

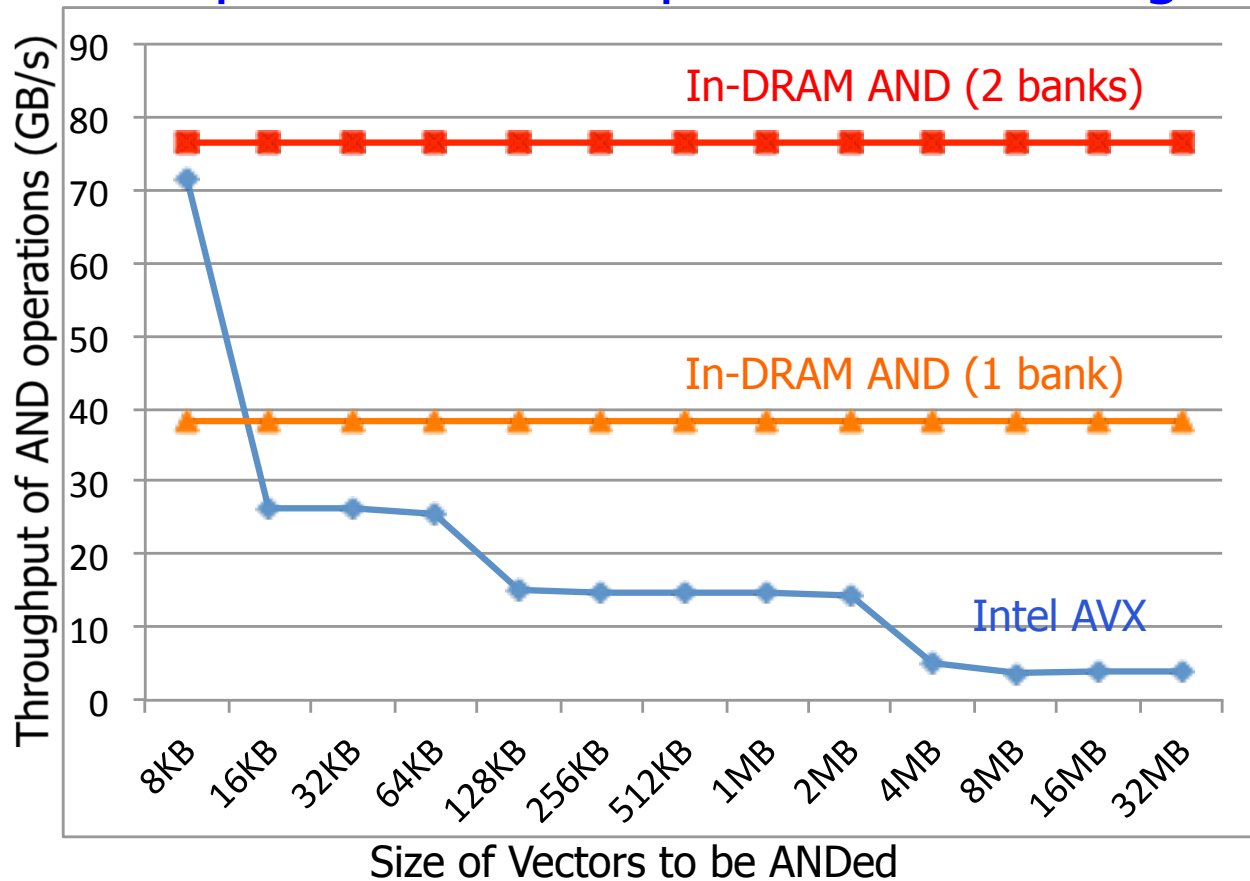
**$C(A + B) +$
 $\sim C(AB)$**

In-DRAM Bulk Bitwise AND/OR Operation

- **BULKAND A, B → C**
 - Semantics: Perform a bitwise AND of two rows A and B and store the result in row C
 - R0 – reserved zero row, R1 – reserved one row
 - D1, D2, D3 – Designated rows for triple activation
1. RowClone A into D1
 2. RowClone B into D2
 3. RowClone R0 into D3
 4. ACTIVATE D1,D2,D3
 5. RowClone Result into C

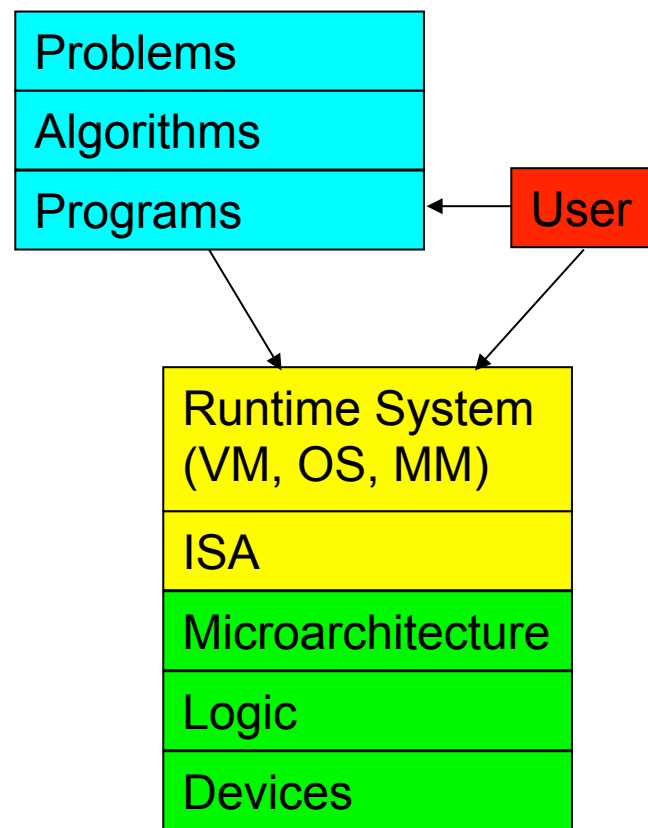
In-DRAM AND/OR Results

- 20X improvement in AND/OR throughput vs. Intel AVX
- 50.5X reduction in memory energy consumption
- At least 30% performance improvement in range queries



Going Forward

- A bulk computation model in memory
- New memory & software interfaces to enable bulk in-memory computation
- New programming models, algorithms, compilers, and system designs that can take advantage of the model



Challenge and Opportunity

Primitives and Interfaces for Computation in Memory

Departing From “Business as Usual”

Memory No Longer a Dumb Device

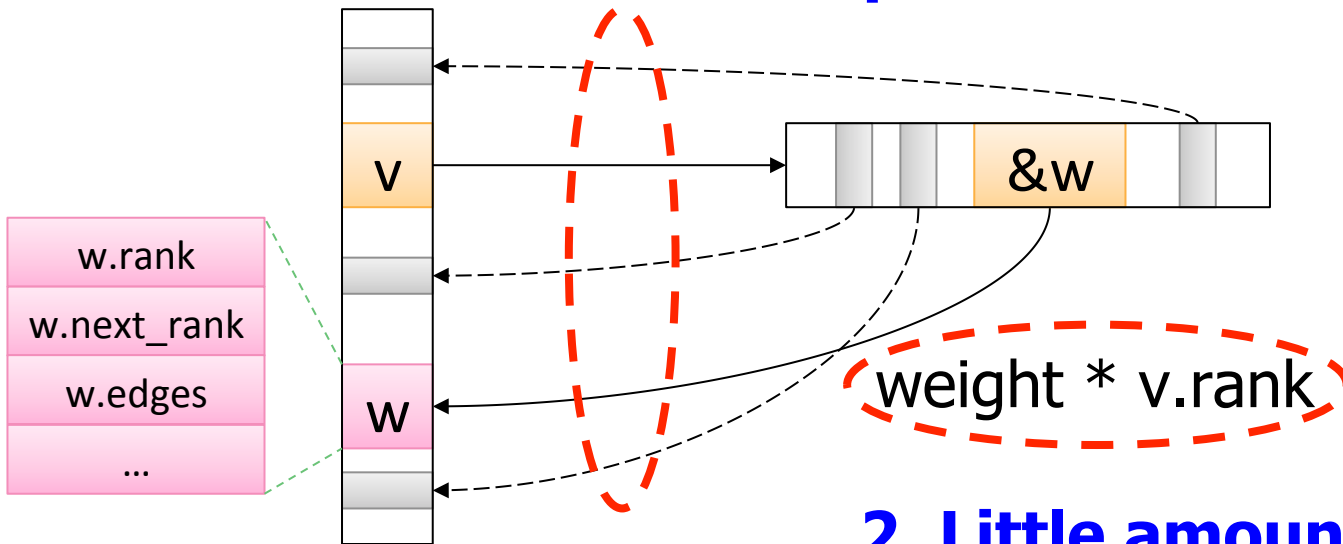
Two Approaches to In-Memory Processing

- 1. Minimally change DRAM to enable simple yet powerful computation primitives
 - RowClone: Fast and Efficient In-DRAM Copy and Initialization of Bulk Data (Seshadri et al., MICRO 2013)
 - Fast Bulk Bitwise AND and OR in DRAM (Seshadri et al., IEEE CAL 2015)
- 2. Exploit the control logic in 3D-stacked memory to enable more comprehensive computation near memory
 - PIM-Enabled Instructions: A Low-Overhead, Locality-Aware Processing-in-Memory Architecture (Ahn et al., ISCA 2015)
 - A Scalable Processing-in-Memory Accelerator for Parallel Graph Processing (Ahn et al., ISCA 2015)

Key Bottlenecks in Graph Processing

```
for (v: graph.vertices) {  
  for (w: v.successors) {  
    w.next_rank += weight * v.rank;  
  }  
}
```

1. Frequent random memory accesses

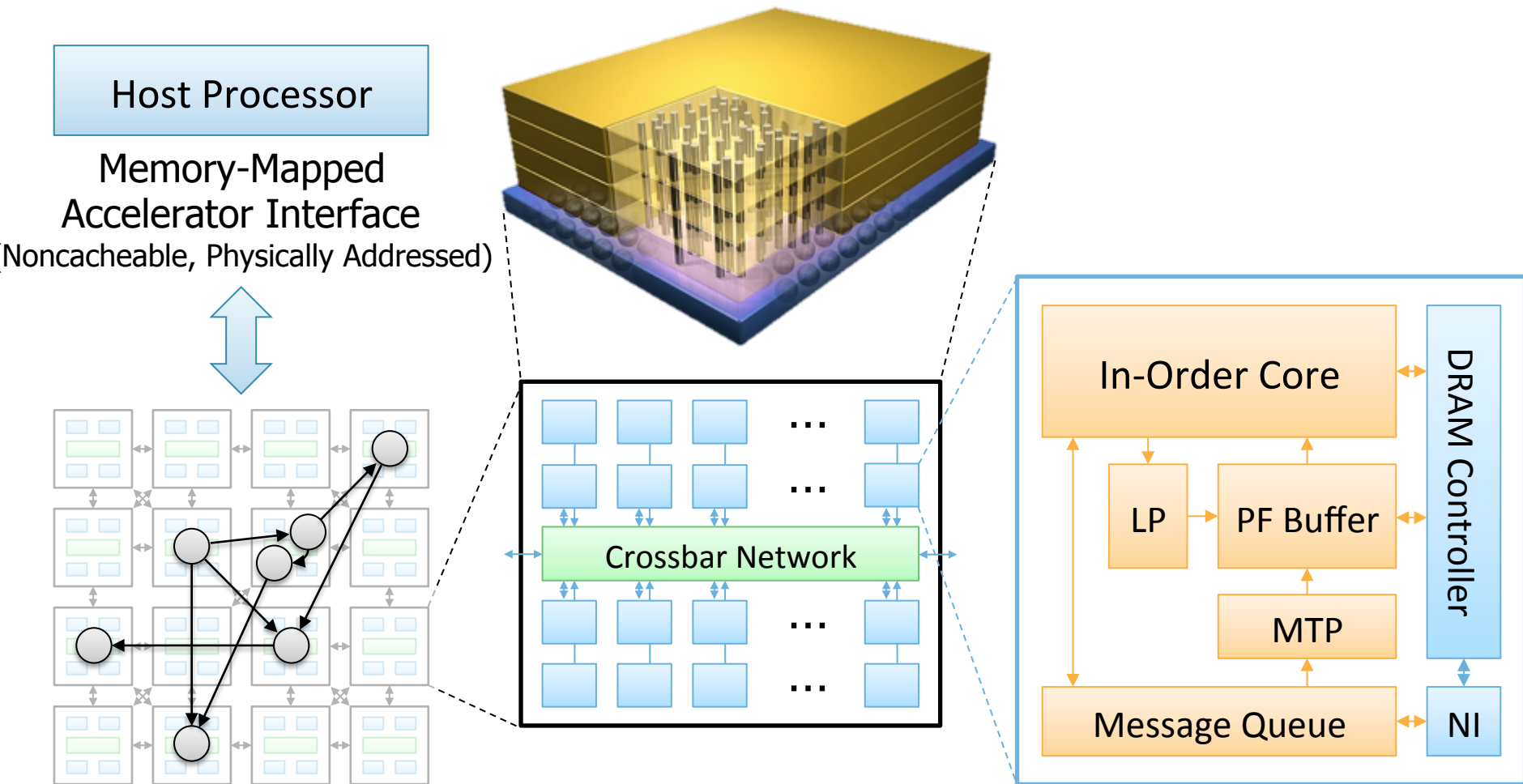


2. Little amount of computation

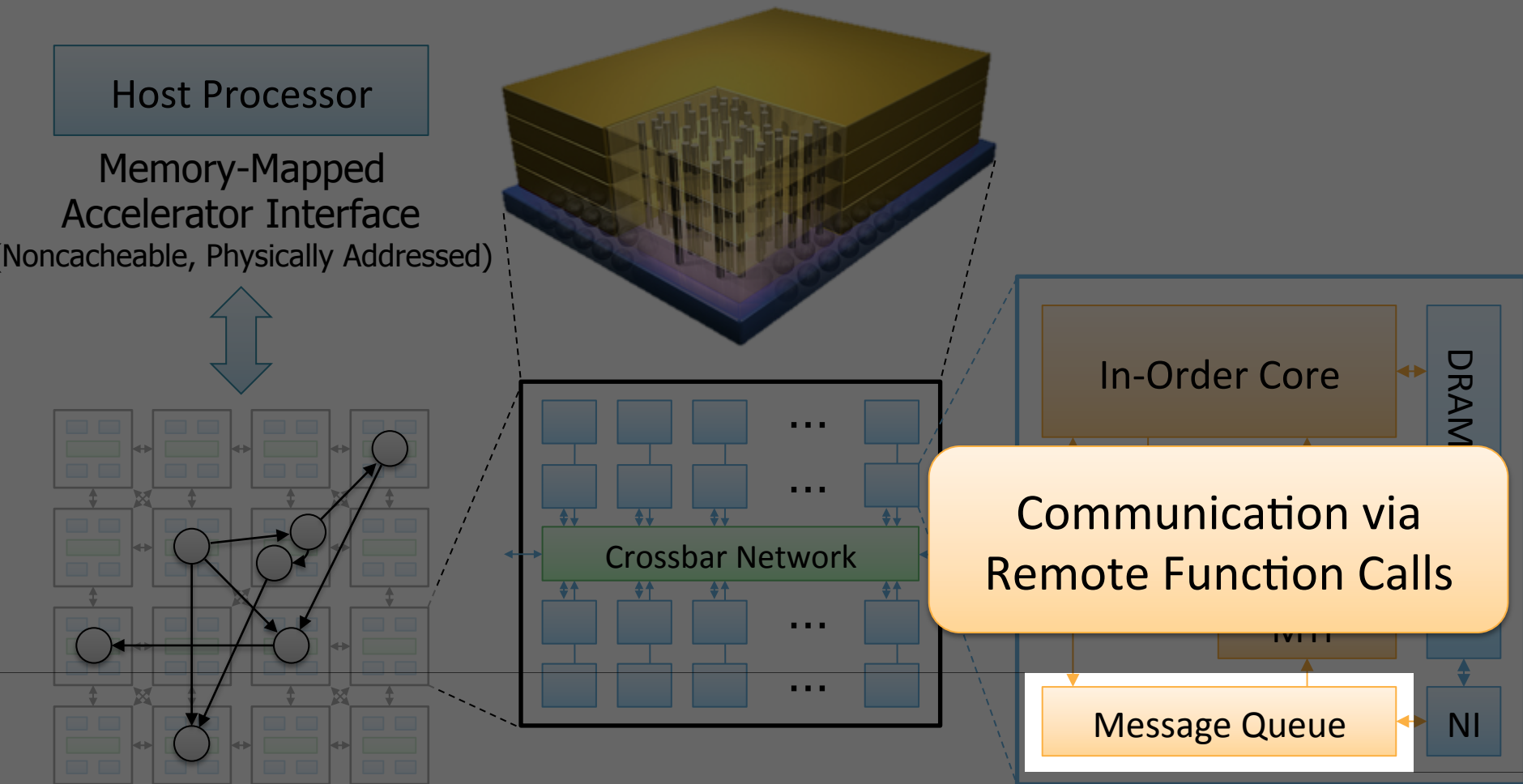
Challenges in Scalable Graph Processing

- **Challenge 1:** How to provide *high memory bandwidth* to computation units in a practical way?
 - Processing-in-memory based on 3D-stacked DRAM
- **Challenge 2:** How to design computation units that *efficiently exploit large memory bandwidth*?
 - Specialized in-order cores called *Tesseract* cores
 - Latency-tolerant programming model
 - Graph-processing-specific prefetching schemes

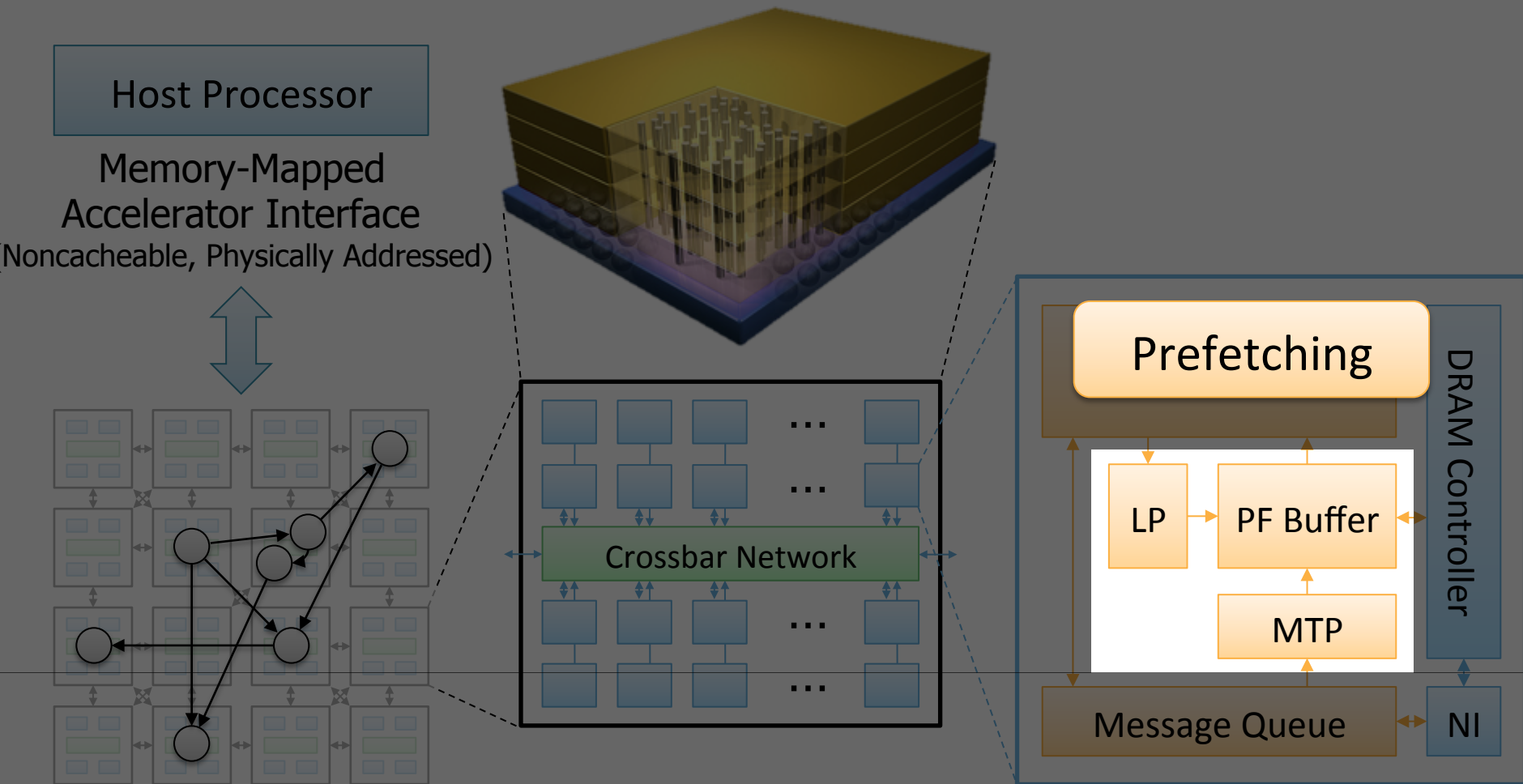
Tesseract System for Graph Processing



Tesseract System for Graph Processing

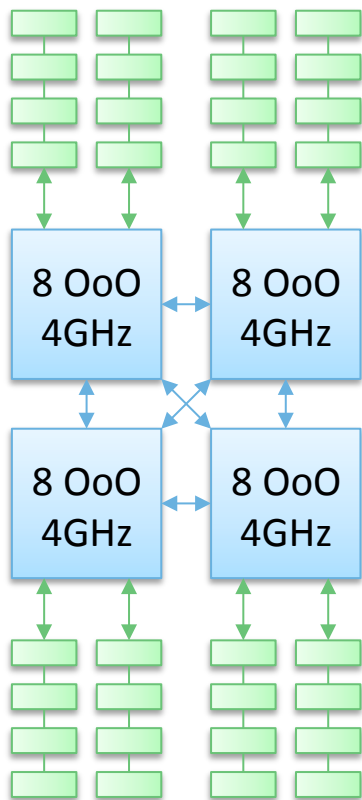


Tesseract System for Graph Processing



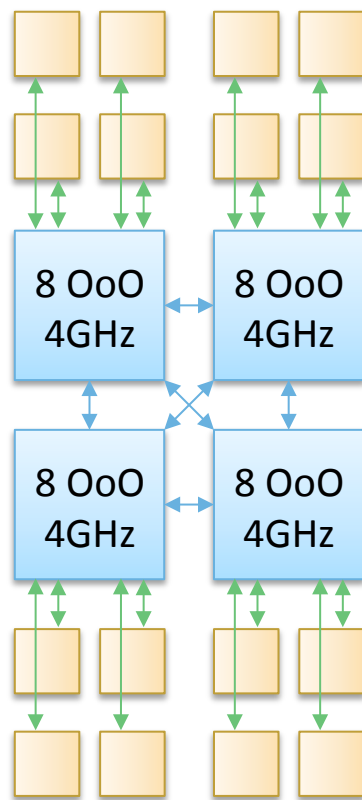
Evaluated Systems

DDR3-OoO
(with FDP)



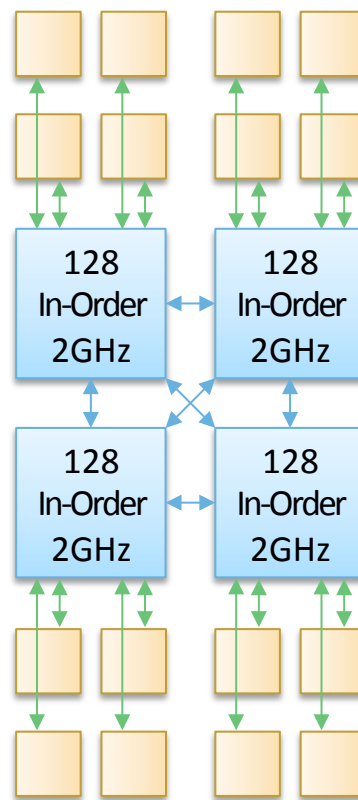
102.4GB/s

HMC-OoO
(with FDP)



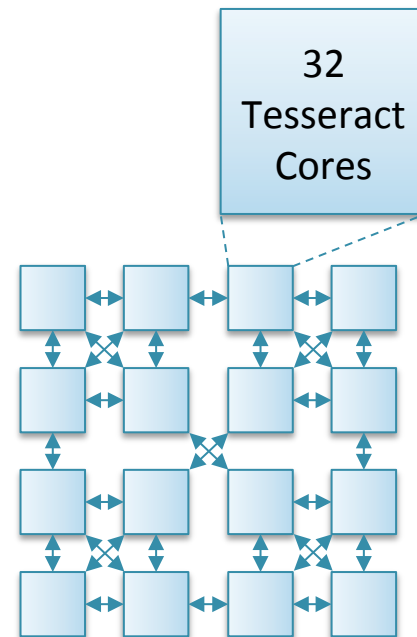
640GB/s

HMC-MC



640GB/s

Tesseract
(32-entry MQ, 4KB PF Buffer)



8TB/s

Workloads

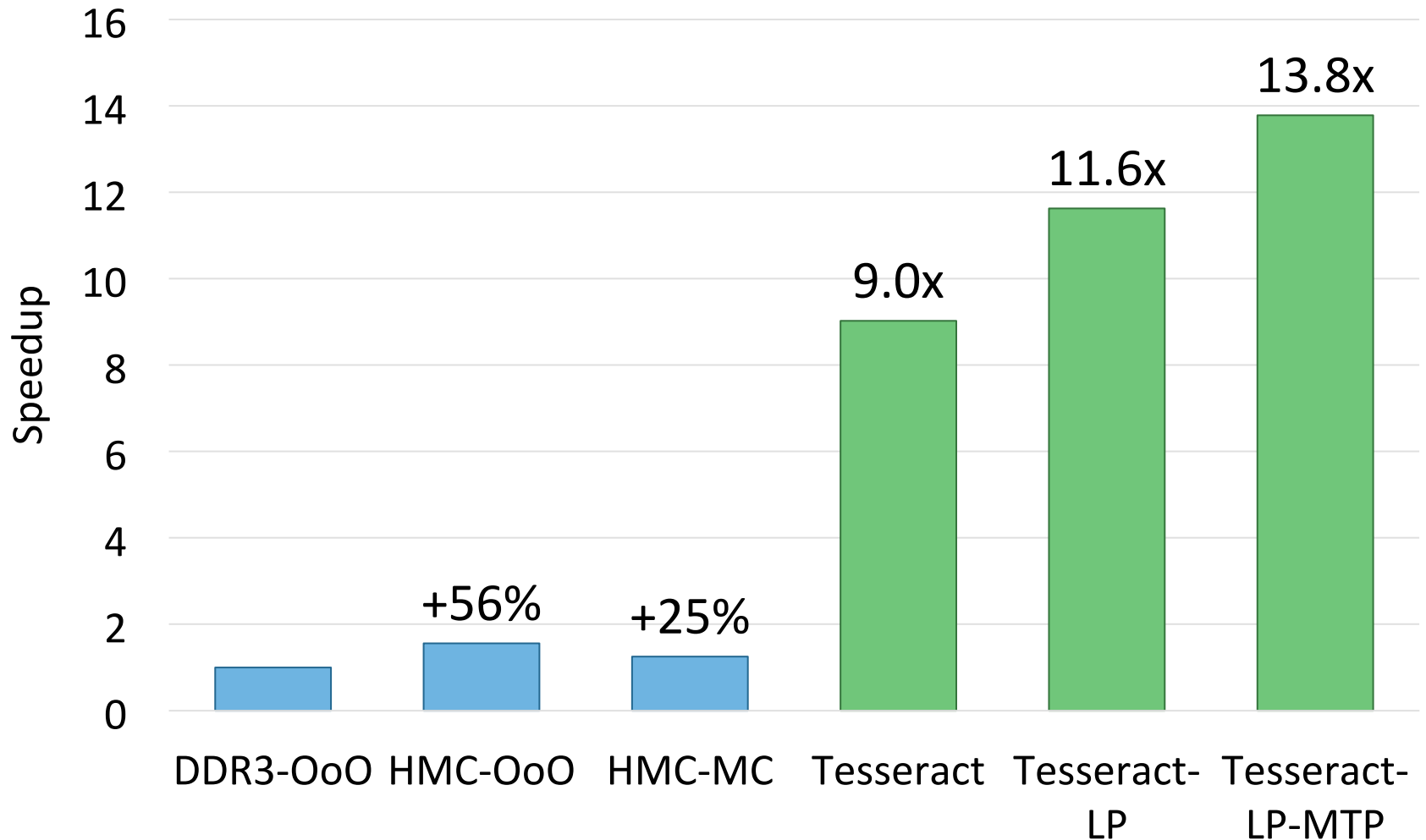
■ Five graph processing algorithms

- ❑ Average teenage follower
- ❑ Conductance
- ❑ PageRank
- ❑ Single-source shortest path
- ❑ Vertex cover

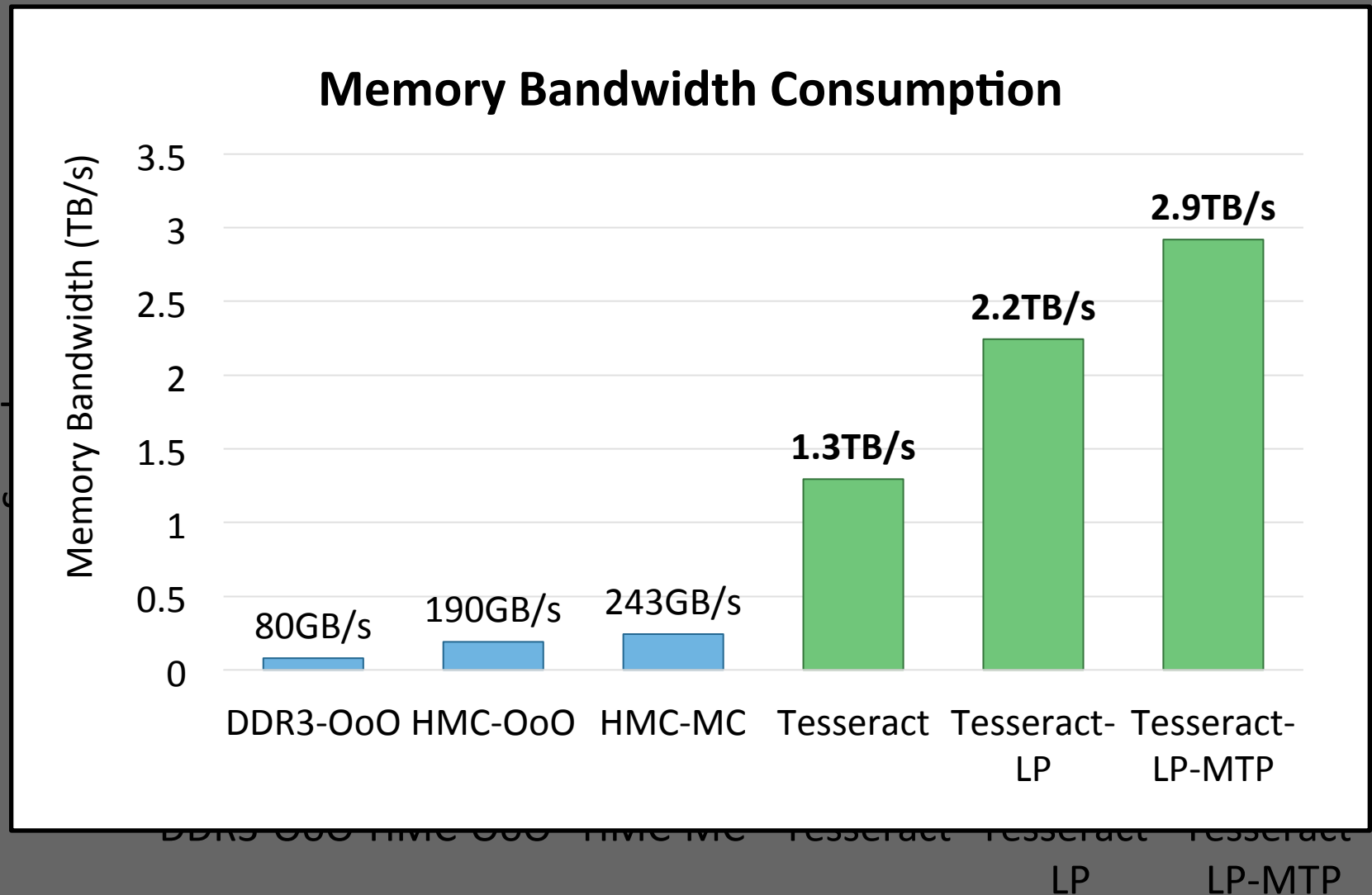
■ Three real-world large graphs

- ❑ ljournal-2008 (social network)
- ❑ enwiki-2003 (Wikipedia)
- ❑ indochina-0024 (web graph)
- ❑ 4~7M vertices, 79~194M edges

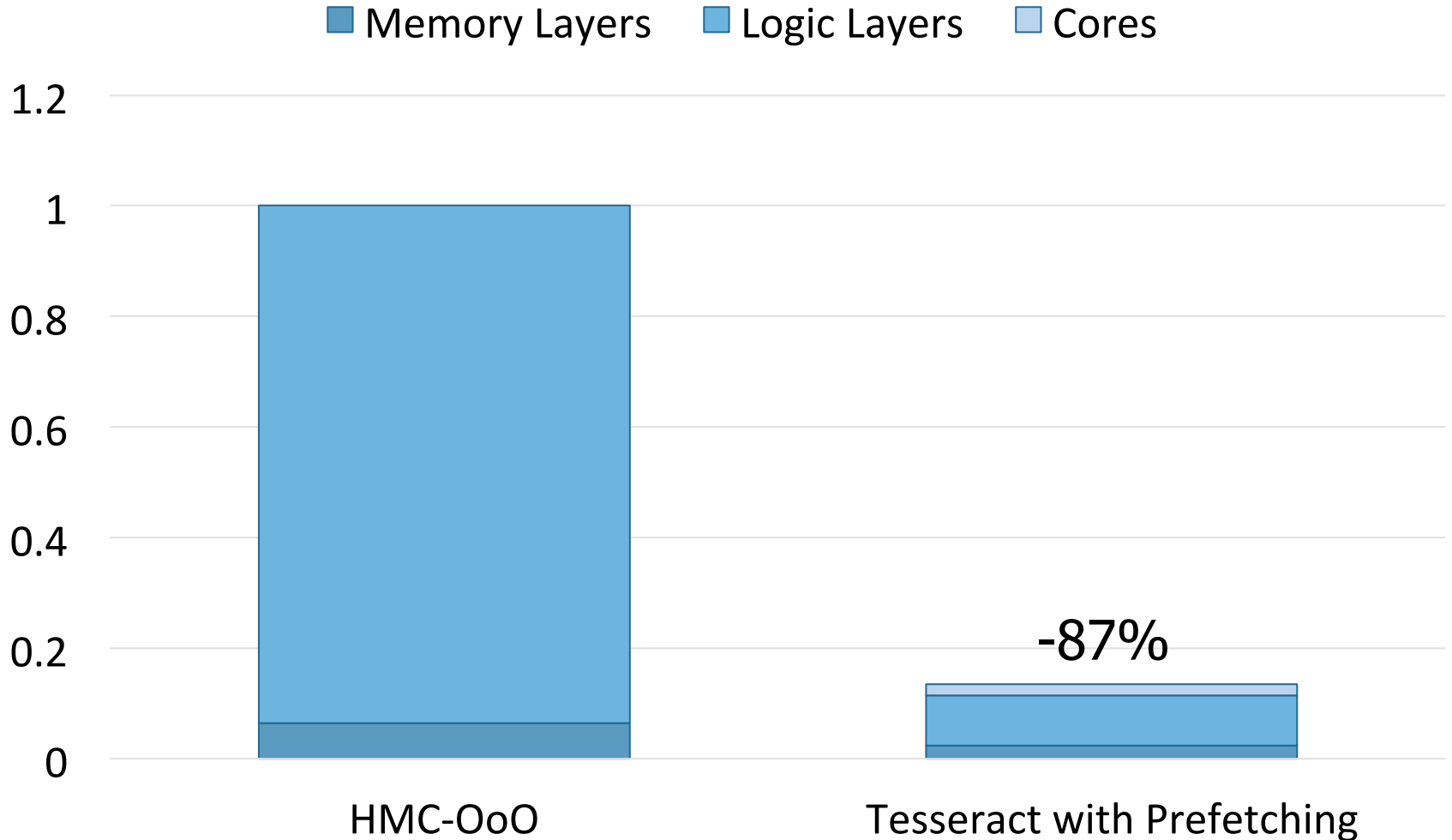
Tesseract Graph Processing Performance



Tesseract Graph Processing Performance



Memory Energy Consumption (Normalized)



Challenge and Opportunity

Memory
Bandwidth
and
Energy

Departing From “Business as Usual”

Memory No Longer a Dumb Device

Dynamic Management
of Margins in Memory

Rethinking Memory Architecture

- Compute Capable Memory

- Refresh

- Reliability

- Latency

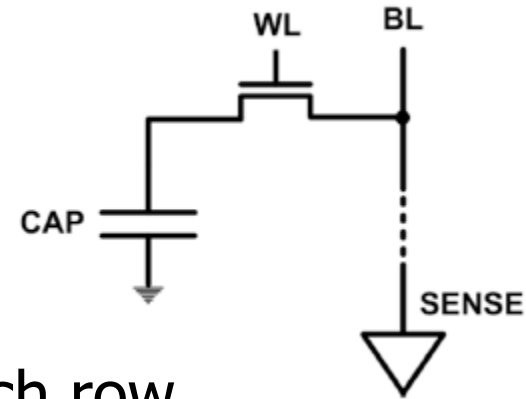
- Bandwidth

- Energy

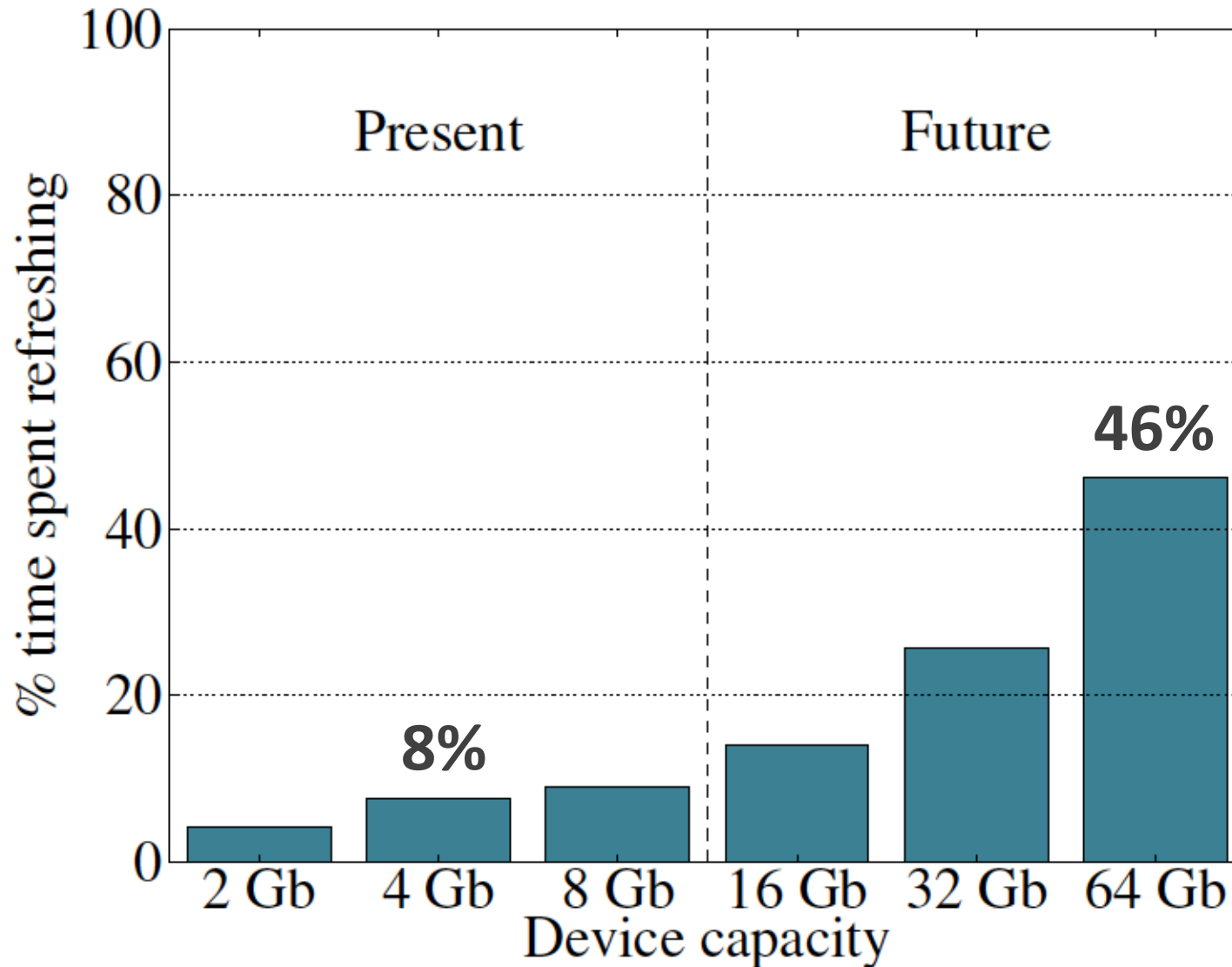
- Memory Compression

DRAM Refresh

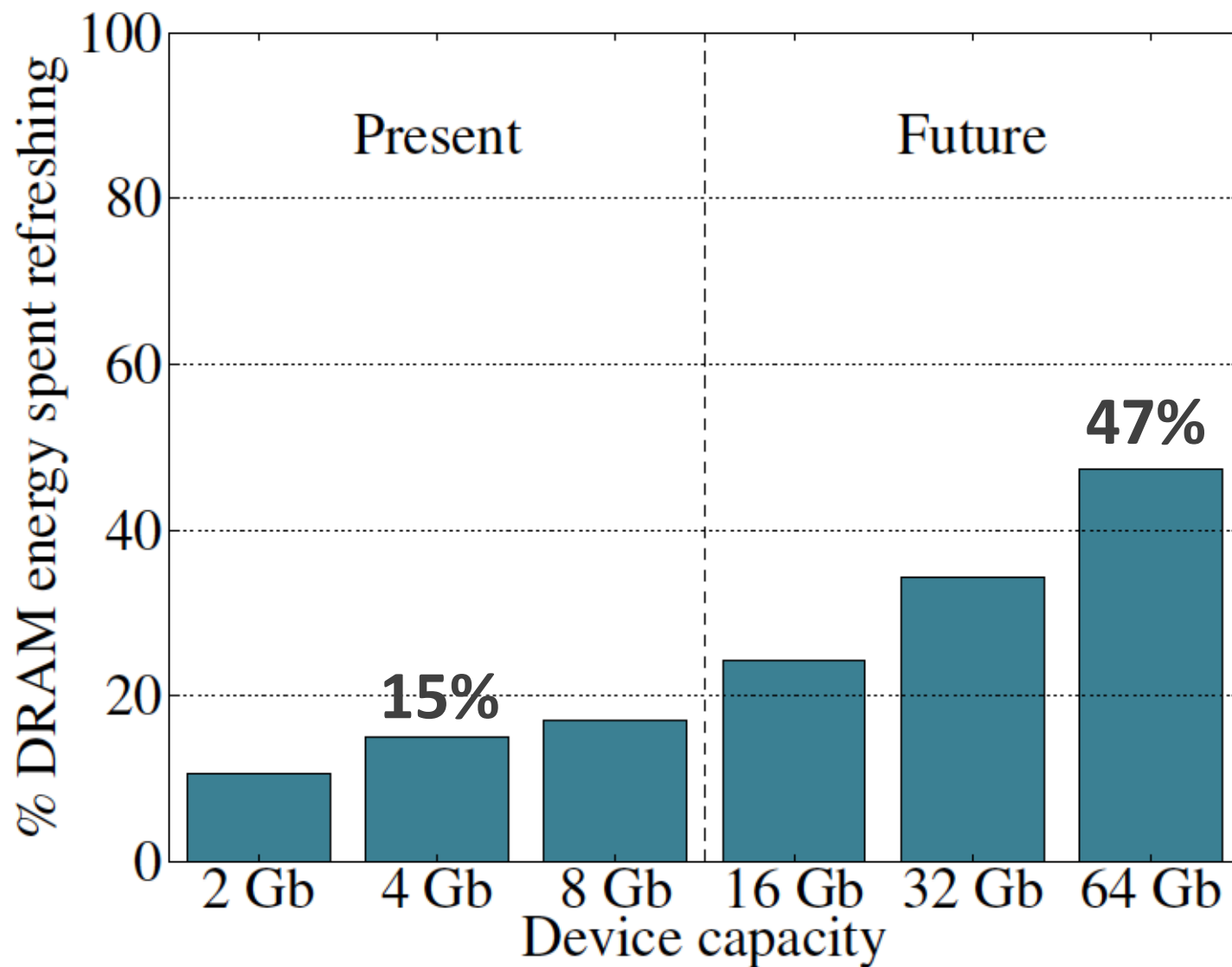
- DRAM capacitor charge leaks over time
- The memory controller needs to refresh each row periodically to restore charge
 - Activate each row every N ms
 - Typical N = 64 ms
- Downsides of refresh
 - **Energy consumption**: Each refresh consumes energy
 - **Performance degradation**: DRAM rank/bank unavailable while refreshed
 - **QoS/predictability impact**: (Long) pause times during refresh
 - **Refresh rate limits DRAM capacity scaling**



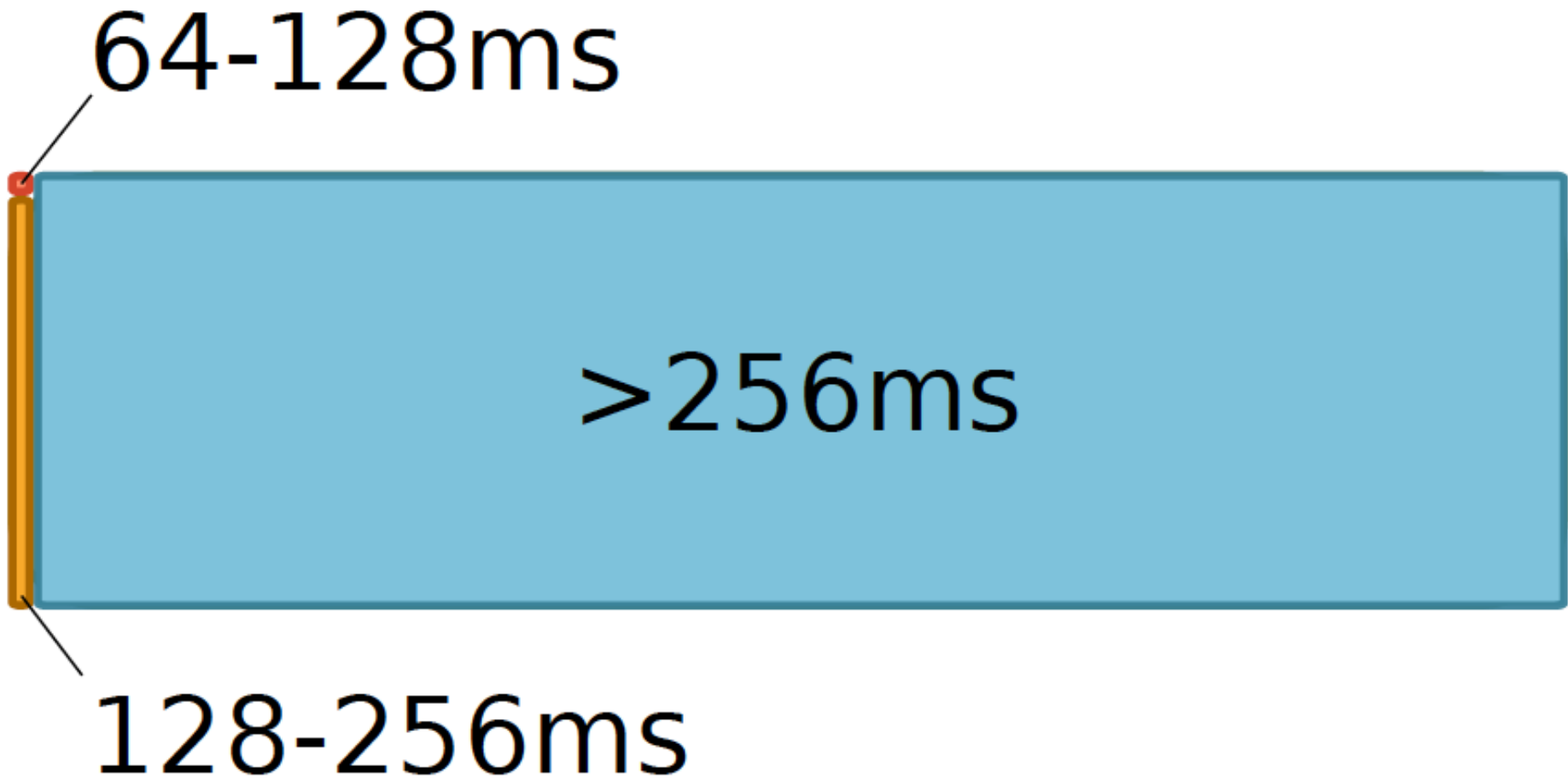
Refresh Overhead: Performance



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

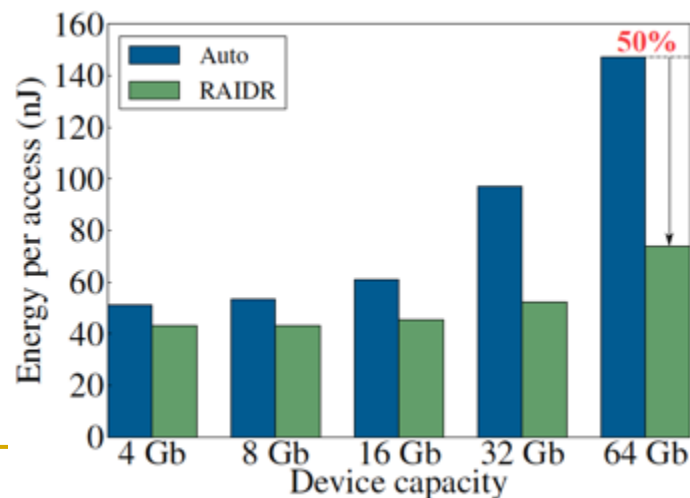
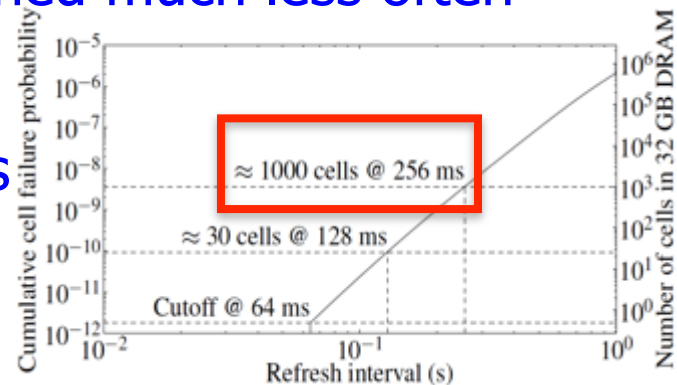
2. **Binning:** Store rows into bins by retention time in memory controller

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



Experimental DRAM Testing Infrastructure



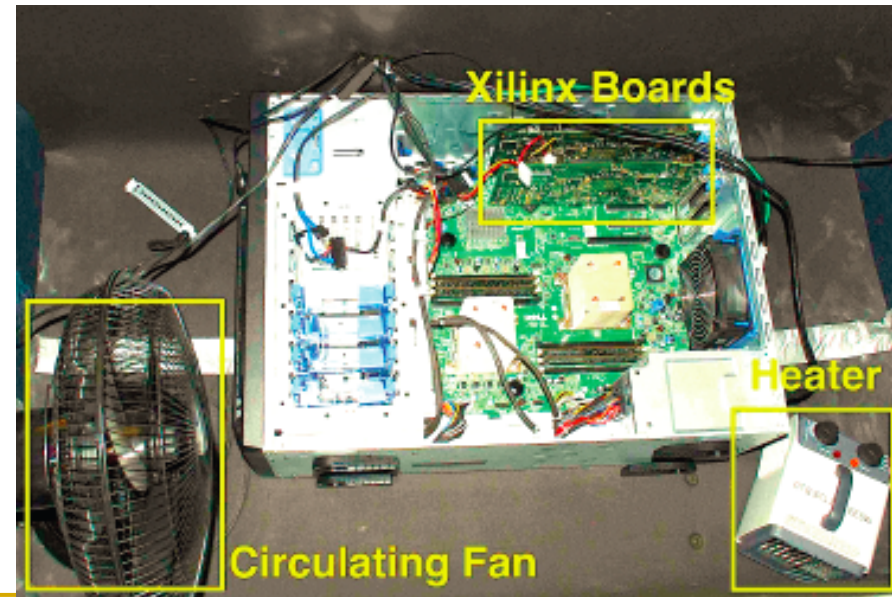
An Experimental Study of Data Retention Behavior in Modern DRAM Devices: Implications for Retention Time Profiling Mechanisms (Liu et al., ISCA 2013)

The Efficacy of Error Mitigation Techniques for DRAM Retention Failures: A Comparative Experimental Study (Khan et al., SIGMETRICS 2014)

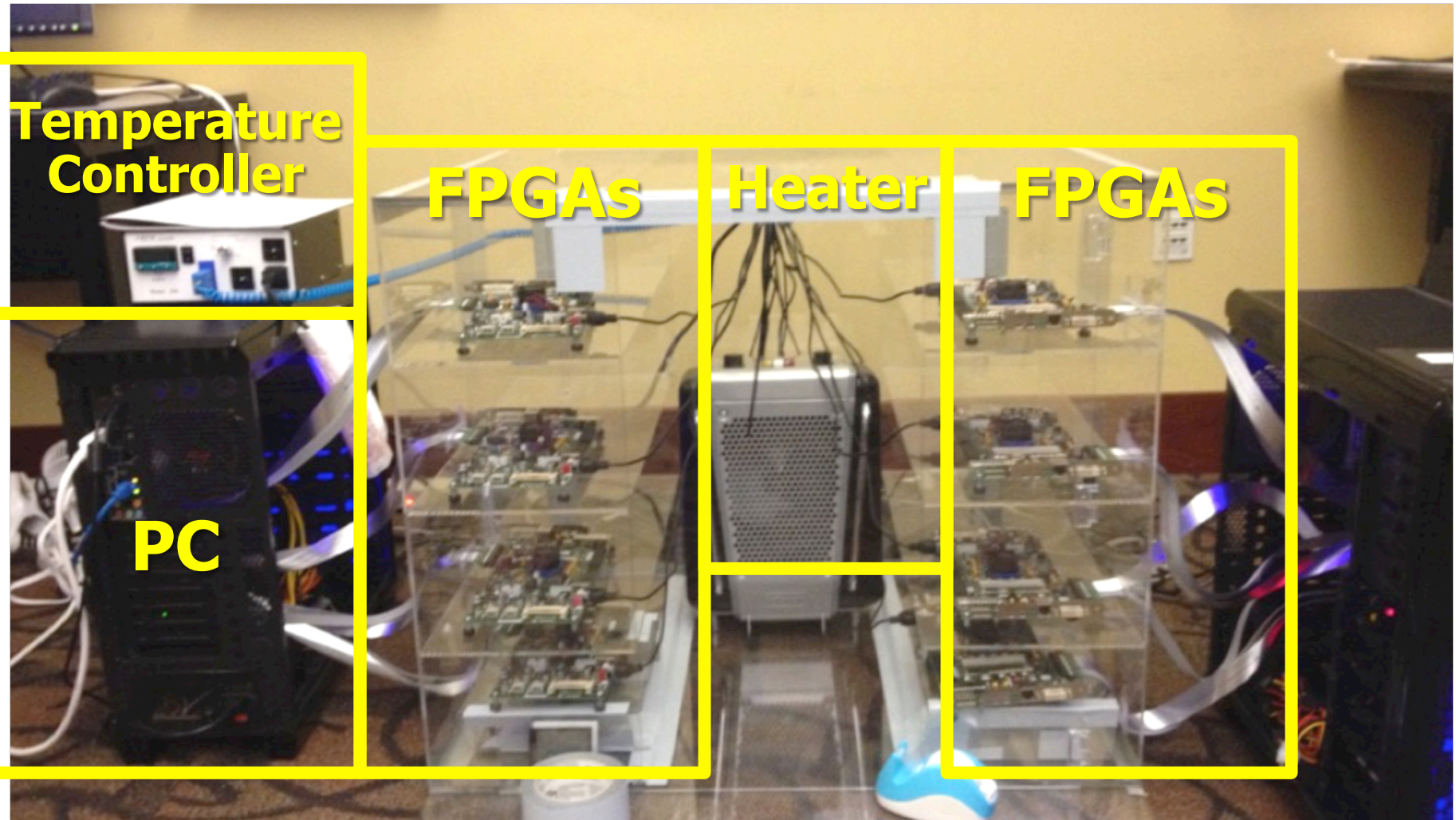
Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors (Kim et al., ISCA 2014)

Adaptive-Latency DRAM: Optimizing DRAM Timing for the Common-Case (Lee et al., HPCA 2015)

AVATAR: A Variable-Retention-Time (VRT) Aware Refresh for DRAM Systems (Qureshi et al., DSN 2015)



Experimental Infrastructure (DRAM)



More Information [ISCA'13, SIGMETRICS'14]

The Efficacy of Error Mitigation Techniques for DRAM Retention Failures: A Comparative Experimental Study

Samira Khan^{†*}
samirakhan@cmu.edu

Donghyuk Lee[†]
donghyuk1@cmu.edu

Yoongu Kim[†]
yoongukim@cmu.edu

Alaa R. Alameldeen^{*}
alaa.r.alameldeen@intel.com

Chris Wilkerson^{*}
chris.wilkerson@intel.com

Onur Mutlu[†]
onur@cmu.edu

[†]Carnegie Mellon University

^{*}Intel Labs

Online Profiling of DRAM In the Field

Initially protect DRAM
with ECC

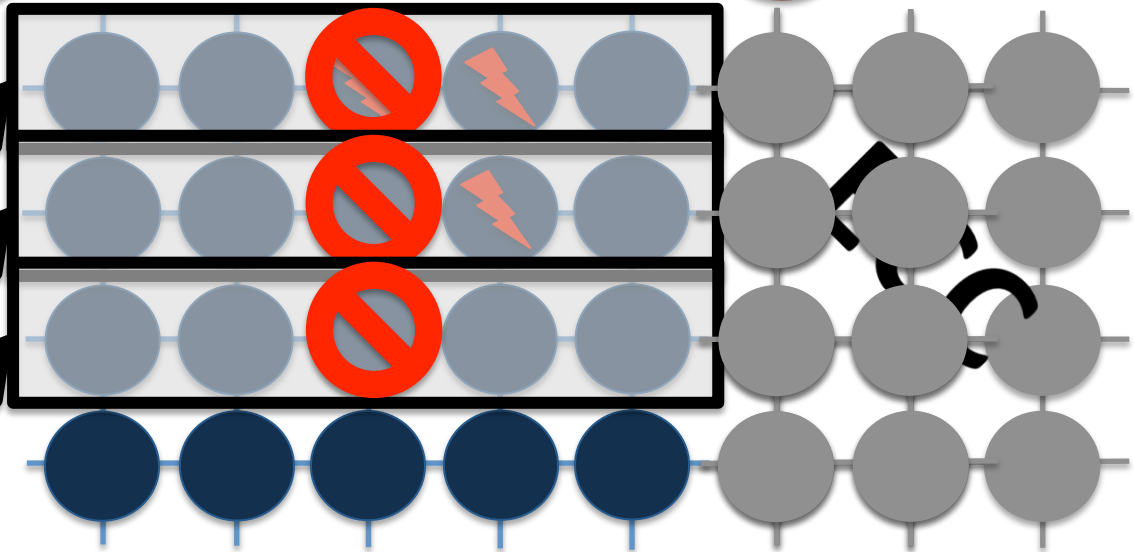
1



Periodically test
parts of DRAM

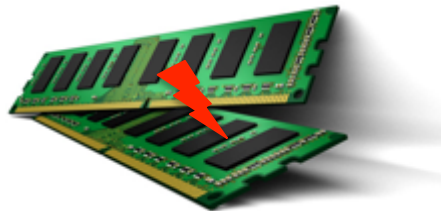
2

Test
Test
Test



Adjust refresh rate and
reduce ECC

3



**Optimize DRAM and mitigate errors online
without disturbing the system and applications**

Challenge and Opportunity

Minimizing Refresh (and Other Technology Taxes)

Departing From “Business as Usual”

Online Detection and Management of Memory Errors

(Online Avoidance of Technology Taxes)

Rethinking Memory Architecture

- In-Memory Computation
- Refresh
- Reliability
- Latency
- Bandwidth
- Energy
- Memory Compression

**Many More
Challenges and Opportunities**

Agenda

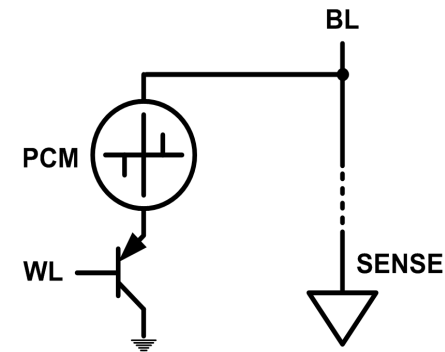
- Major Trends Affecting Main Memory
- The Memory Scaling Problem and Solution Directions
 - New Memory Architectures
 - Enabling Emerging Technologies
- Cross-Cutting Principles
- Summary

Emerging Memory Technologies

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

- Example: Phase Change Memory

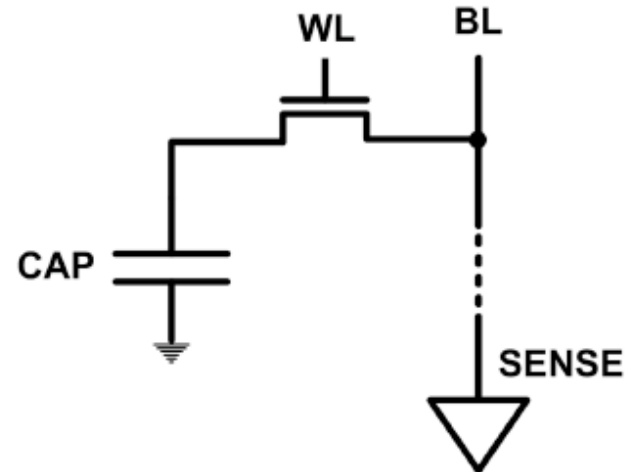
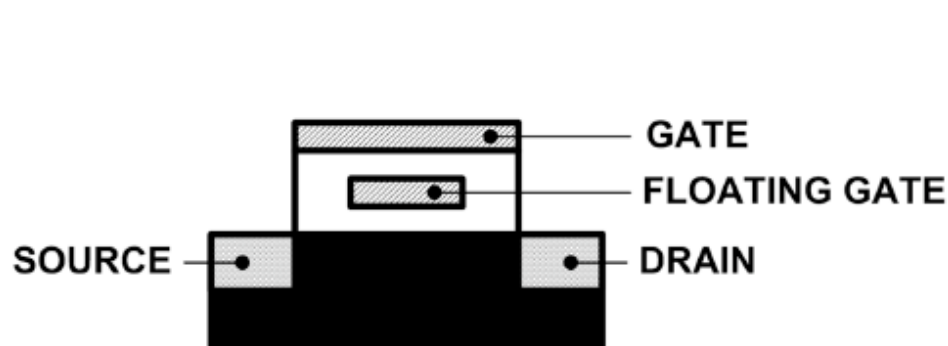
- Data stored by changing phase of material
- Data read by detecting material's resistance
- Expected to scale to 9nm (2022 [ITRS])
- Prototyped at 20nm (Raoux+, IBM JRD 2008)
- Expected to be denser than DRAM: can store multiple bits/cell



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

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

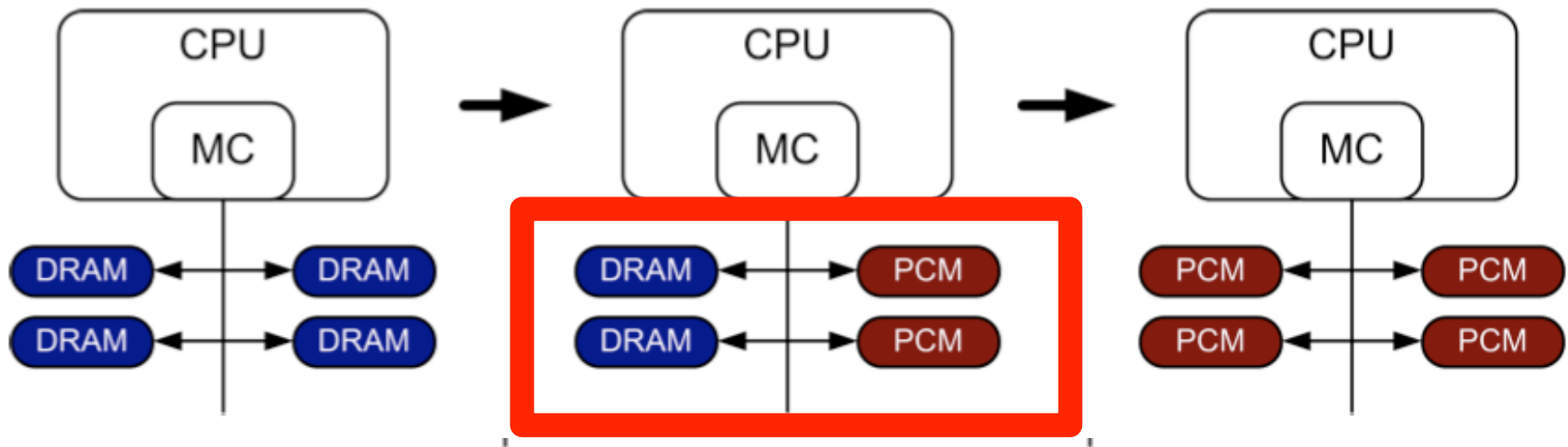
- ❑ Higher latencies: $\sim 4\text{-}15\times$ DRAM (especially write)
- ❑ Higher active energy: $\sim 2\text{-}50\times$ DRAM (especially write)
- ❑ Lower endurance (a cell dies after $\sim 10^8$ writes)
- ❑ Reliability issues (resistance drift)

■ 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)

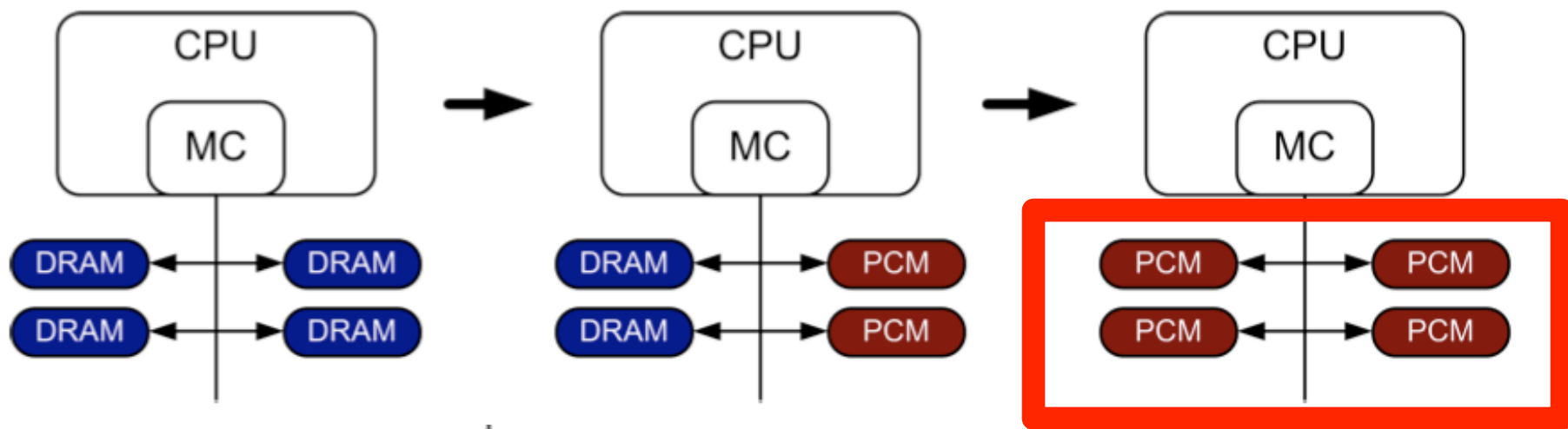
- How should PCM-based (main) memory be organized?



- **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
- ▷ $1.5\times$ DRAM

Latency

- ▷ 50ns Rd, 150ns Wr
- ▷ $4\times, 12\times$ DRAM

Endurance

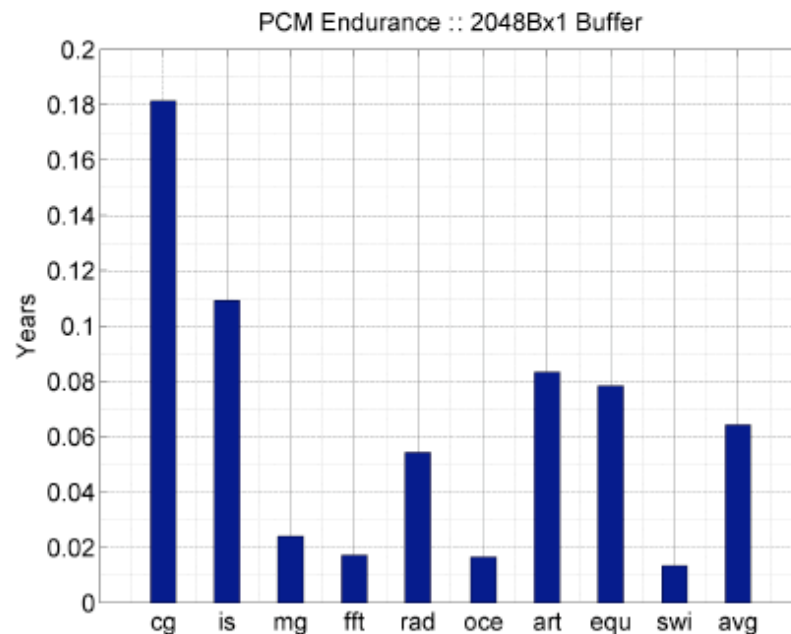
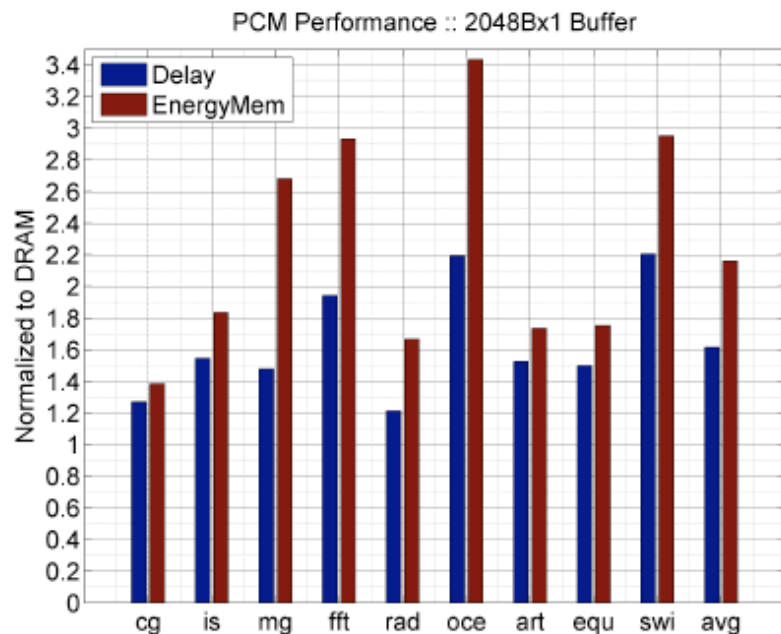
- ▷ $1E+08$ writes
- ▷ $1E-08\times$ DRAM

Energy

- ▷ $40\mu A$ Rd, $150\mu A$ Wr
- ▷ $2\times, 43\times$ DRAM

Results: Naïve Replacement of DRAM with PCM

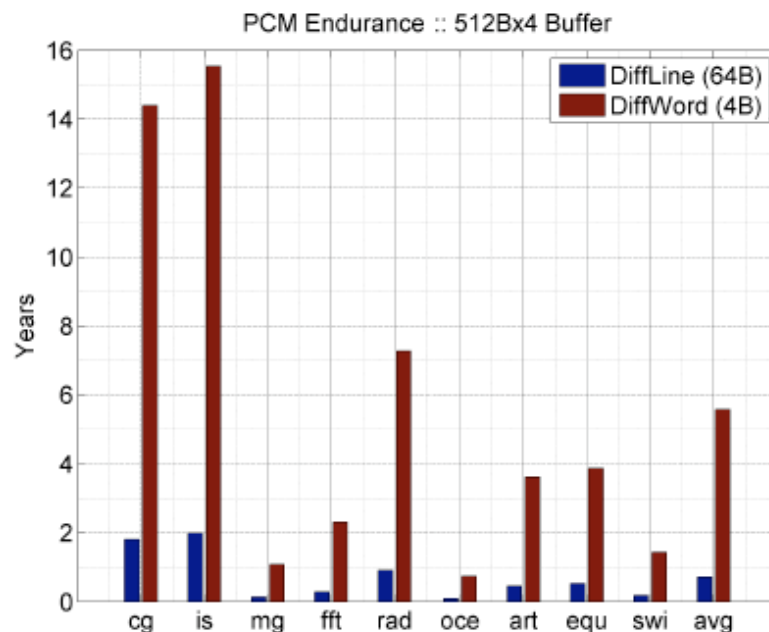
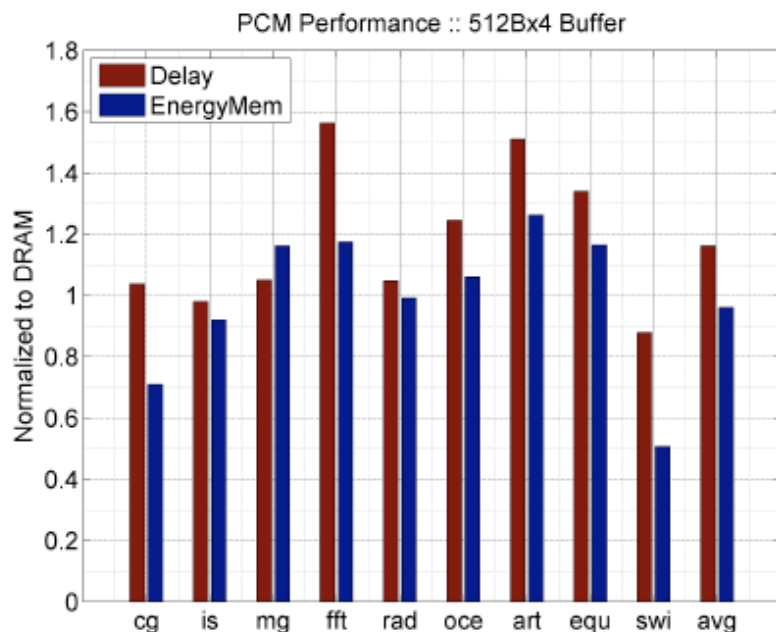
- Replace DRAM with PCM in a 4-core, 4MB L2 system
- PCM organized the same as DRAM: row buffers, banks, peripherals
- 1.6x delay, 2.2x energy, 500-hour average lifetime



- Lee, Ipek, Mutlu, Burger, “[Architecting Phase Change Memory as a Scalable DRAM Alternative](#),” ISCA 2009.

Results: Architected PCM as Main Memory

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

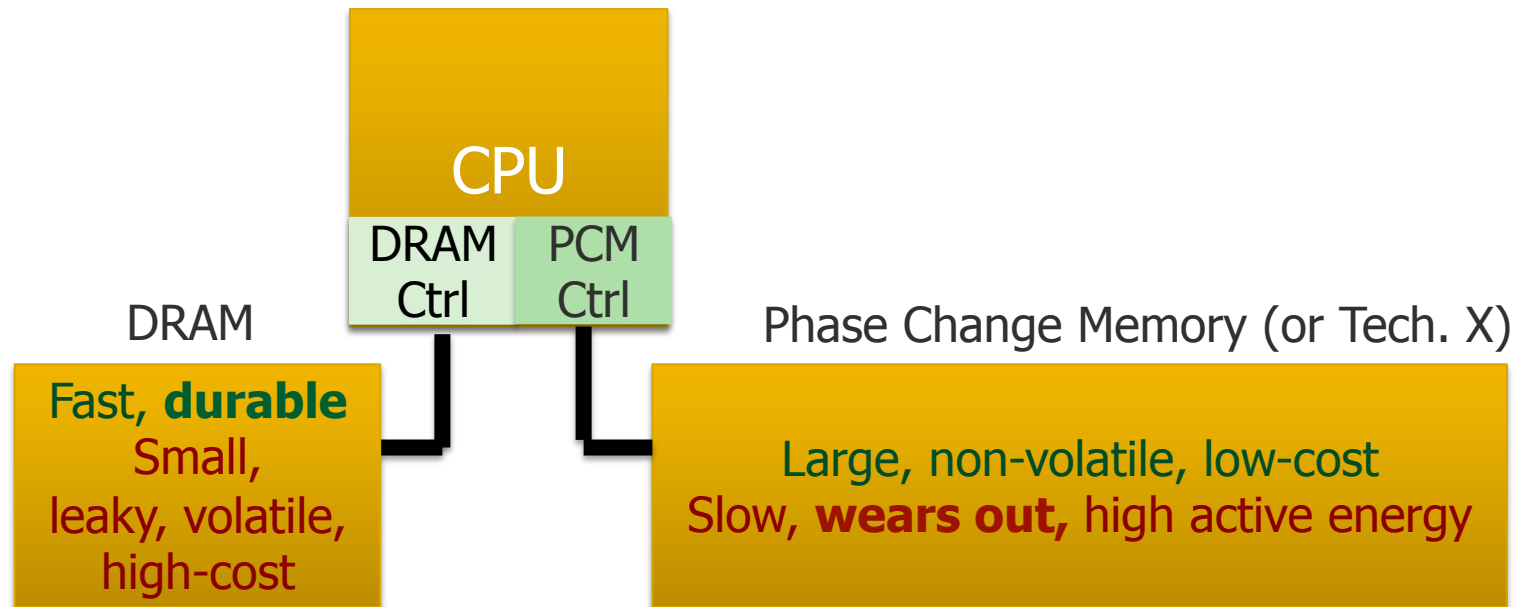


- Caveat 1: Worst-case lifetime is much shorter (no guarantees)
- Caveat 2: Intensive applications see large performance and energy hits
- Caveat 3: Optimistic PCM parameters?

Challenge and Opportunity

Enabling an Emerging Technology
to Replace DRAM

A More Viable Approach: Hybrid Memory Systems



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

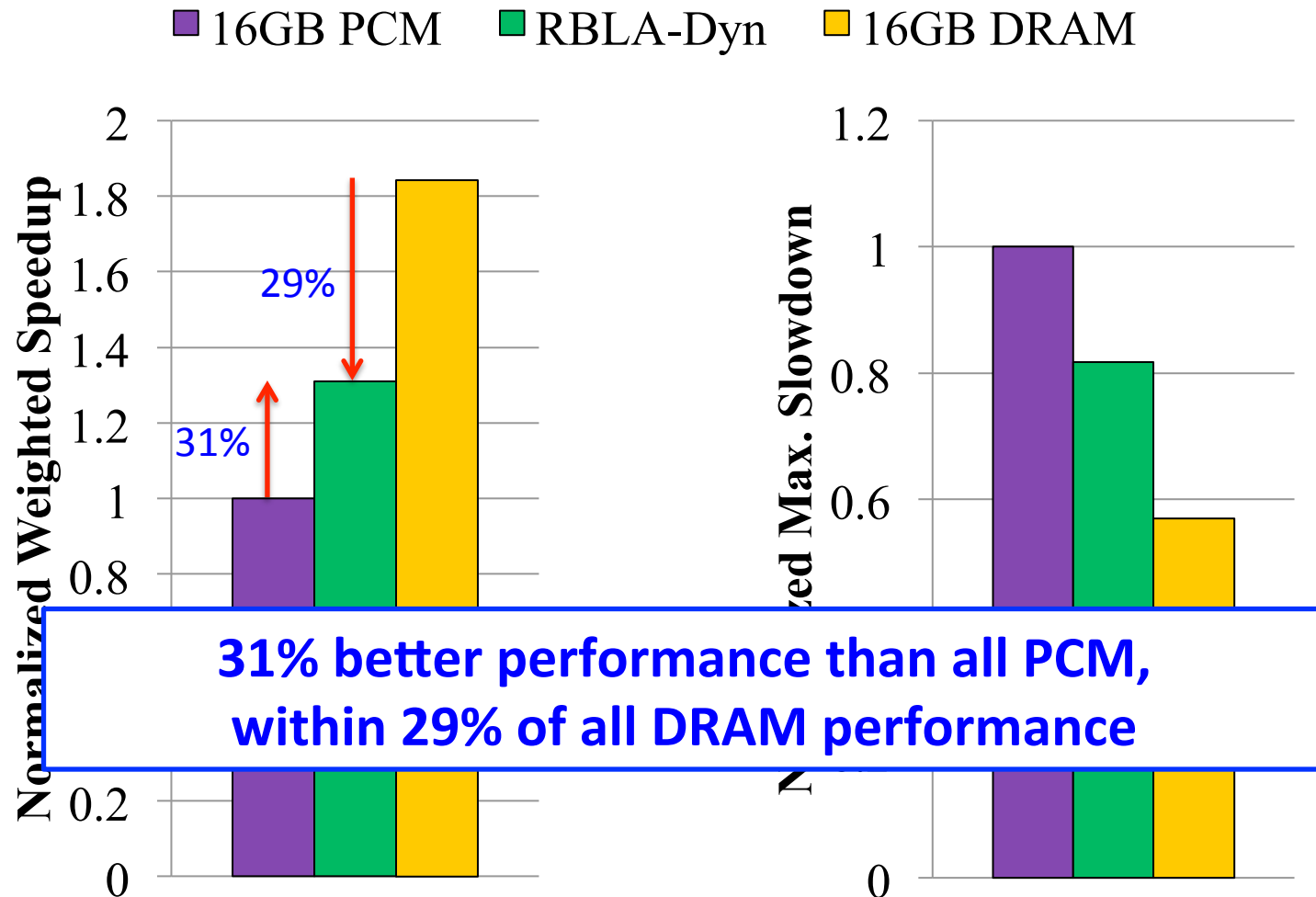
Meza+, "Enabling Efficient and Scalable Hybrid Memories," IEEE Comp. Arch. Letters, 2012.

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

Data Placement Between DRAM and PCM

- Idea: Characterize data access patterns and guide data placement in hybrid memory
- Streaming accesses: As fast in PCM as in DRAM
- Random accesses: Much faster in DRAM
- Idea: Place random access data with some reuse in DRAM; streaming data in PCM
- Yoon+, “Row Buffer Locality-Aware Data Placement in Hybrid Memories,” ICCD 2012 Best Paper Award.

Hybrid vs. All-PCM/DRAM [ICCD'12]



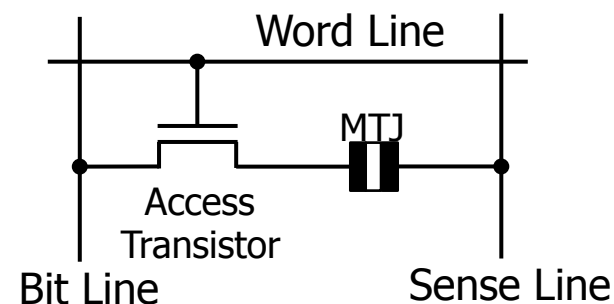
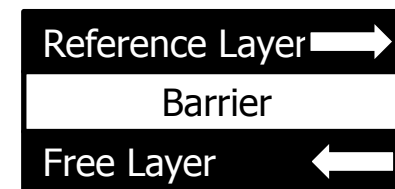
STT-MRAM as Main Memory

- Magnetic Tunnel Junction (MTJ) device
 - ❑ Reference layer: Fixed magnetic orientation
 - ❑ Free layer: Parallel or anti-parallel
- Magnetic orientation of the free layer determines logical state of device
 - ❑ High vs. low resistance
- Write: Push large current through MTJ to change orientation of free layer
- Read: Sense current flow
- Kultursay et al., "Evaluating STT-RAM as an Energy-Efficient Main Memory Alternative," ISPASS 2013.

Logical 0



Logical 1



STT-MRAM: Pros and Cons

■ Pros over DRAM

- Better technology scaling
- Non volatility
- Low idle power (no refresh)

■ Cons

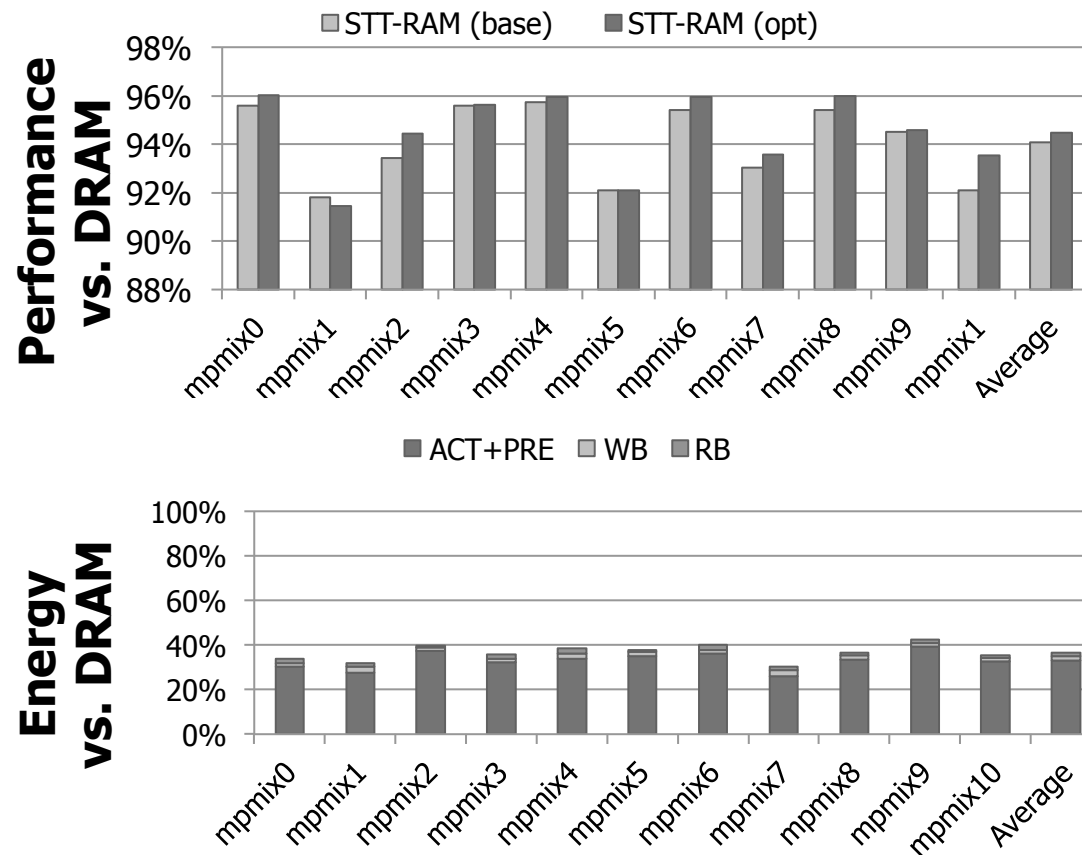
- Higher write latency
- Higher write energy
- Reliability?

■ 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



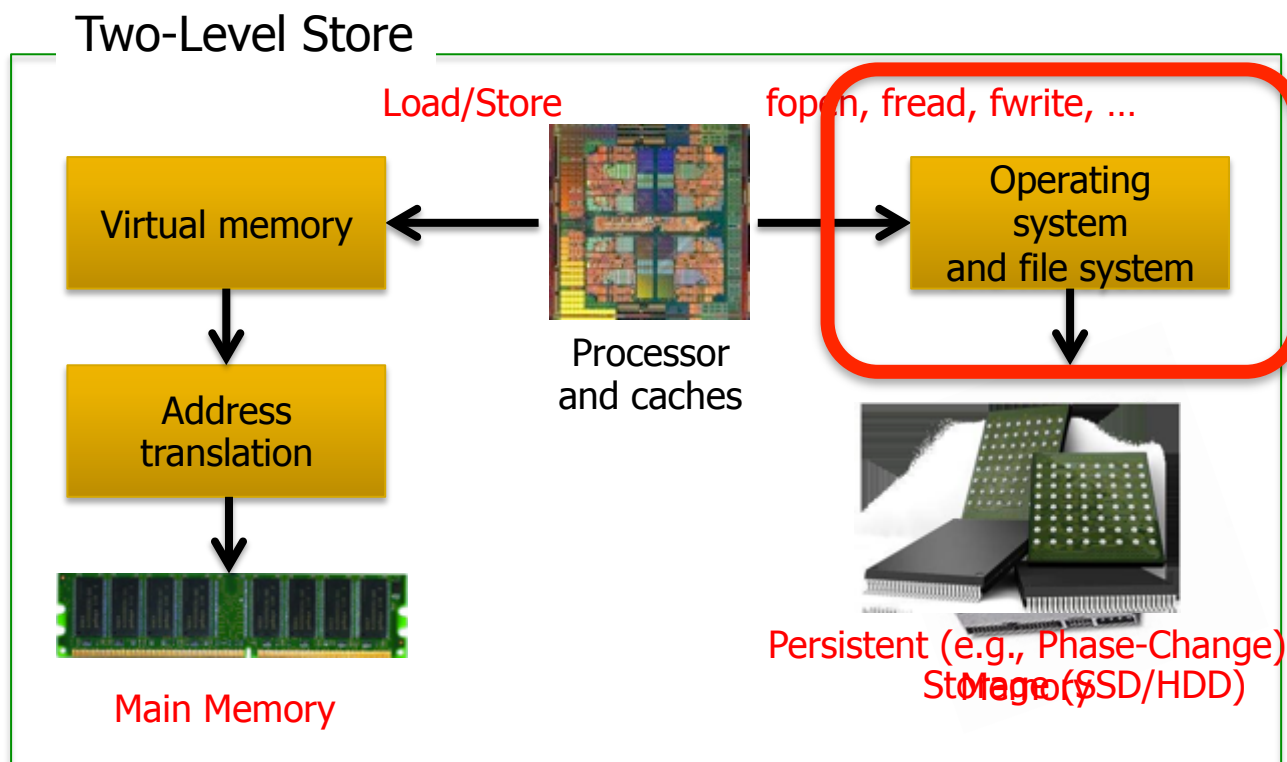
Kultursay+, "Evaluating STT-RAM as an Energy-Efficient Main Memory Alternative," ISPASS 2013.

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

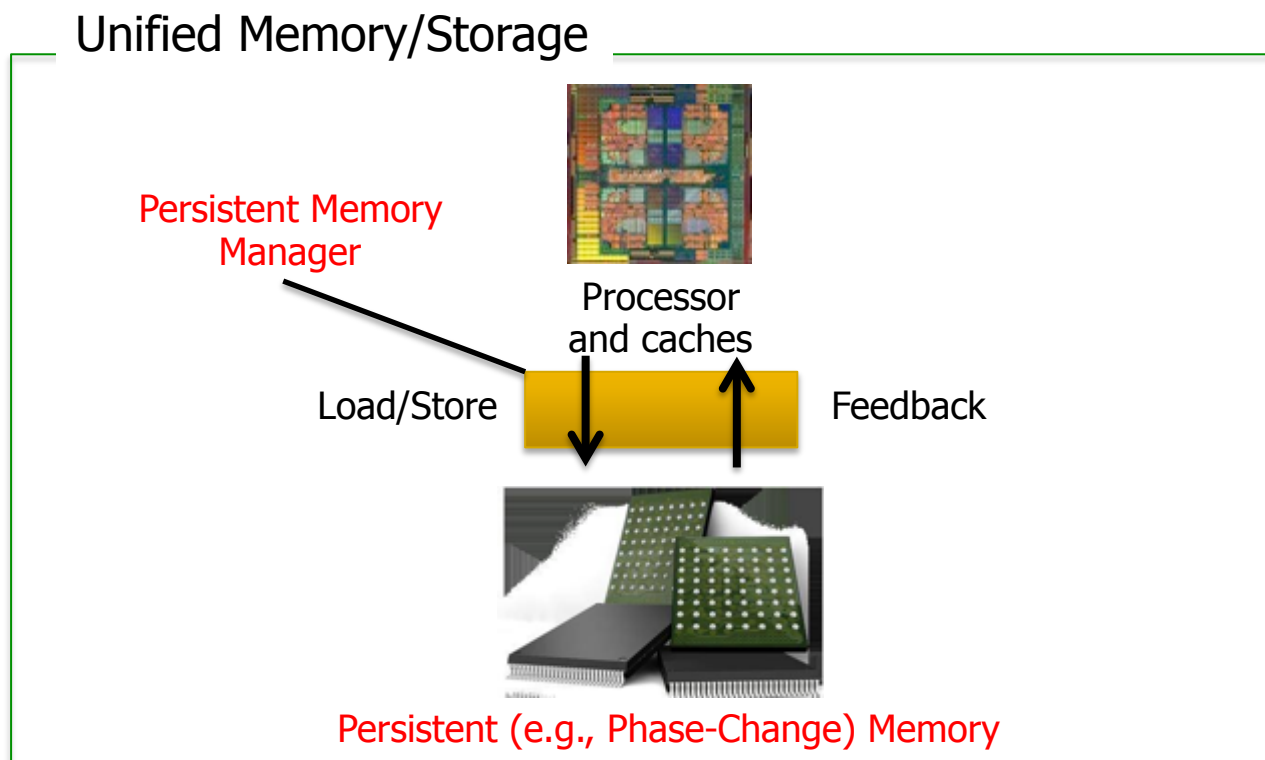
Coordinated Memory and Storage with NVM (I)

- 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



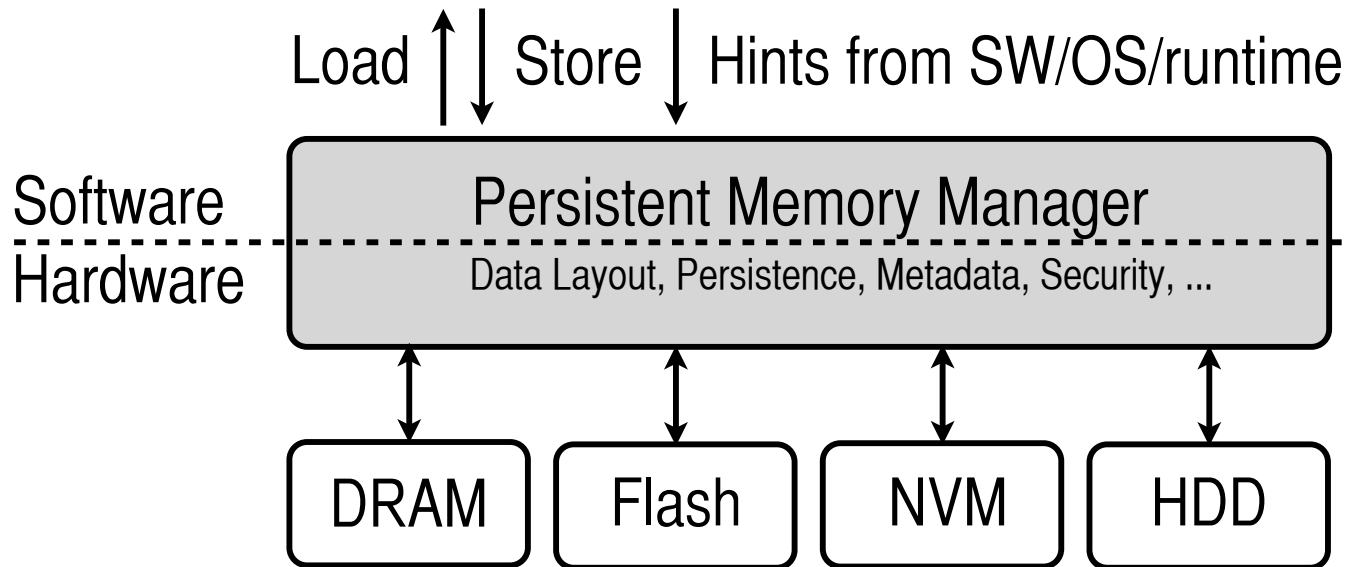
The Persistent Memory Manager (PMM)

- 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)

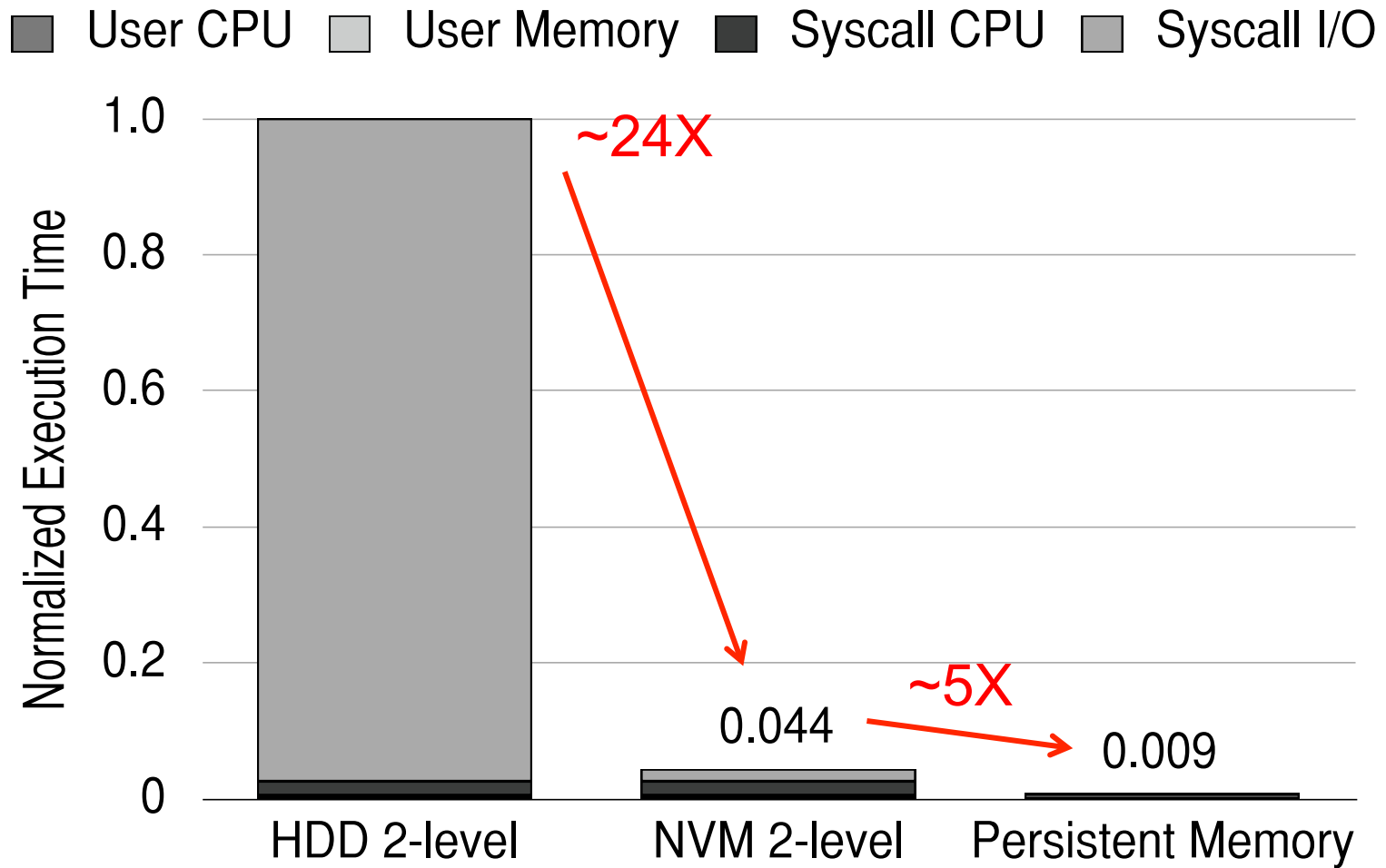
```
1 int main(void) {  
2     // data in file.dat is persistent  
3     FILE myData = "file.dat";  
4     myData = new int[64];  
5 }  
6 void updateValue(int n, int value) {  
7     FILE myData = "file.dat";  
8     myData[n] = value; // value is persistent  
9 }
```

Persistent objects

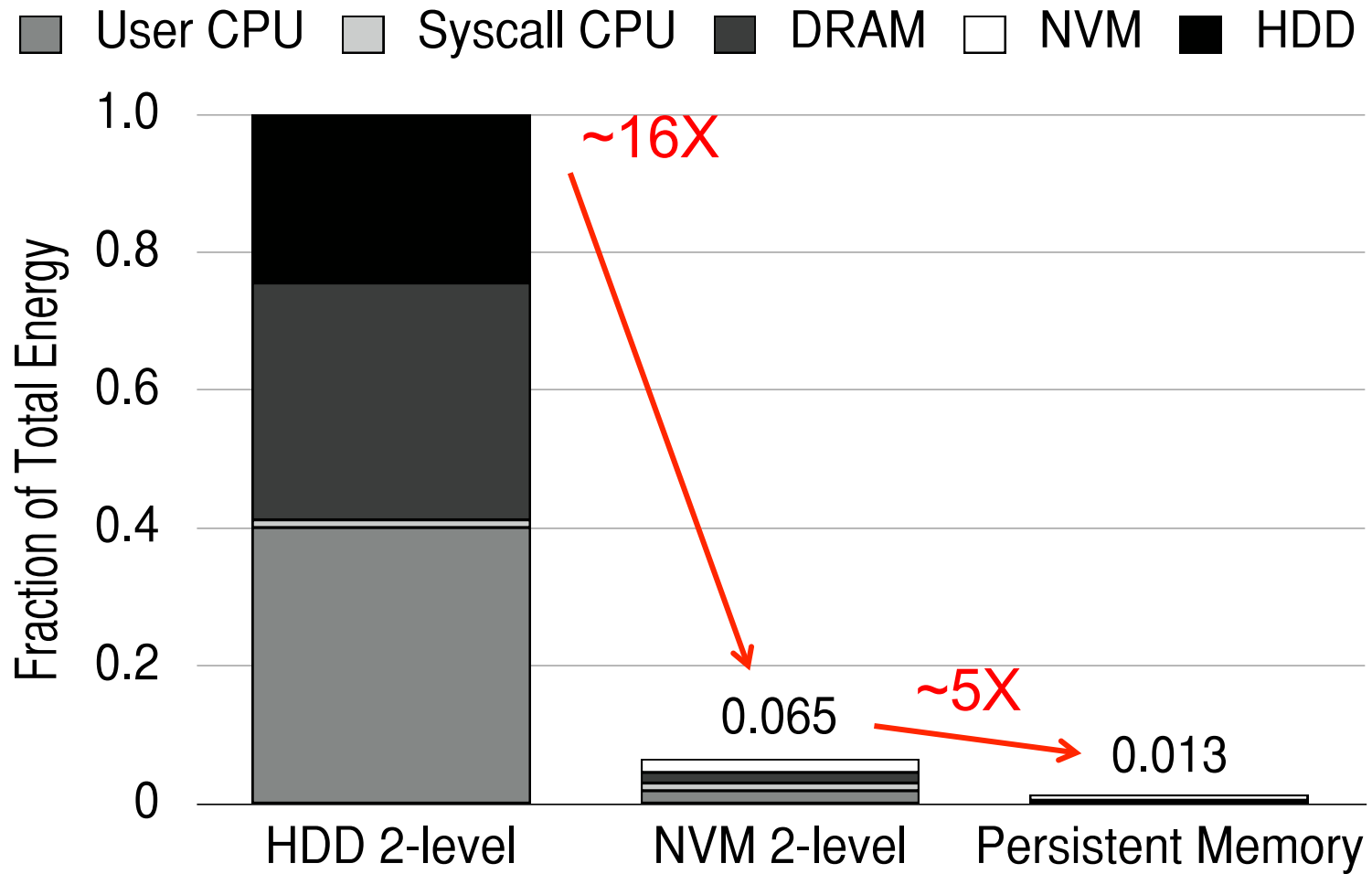


PMM uses access and hint information to allocate, locate, migrate and access data in the heterogeneous array of devices

Performance Benefits of a Single-Level Store



Energy Benefits of a Single-Level Store



Challenge and Opportunity

Combined Memory & Storage

Departing From “Business as Usual”

A Unified Interface to All Data

One Challenge

- How to ensure consistency of system/data if all memory is persistent?
- Two extremes
 - Programmer transparent: Let the system handle everything
 - Programmer only: Let the programmer handle everything
 - Many alternatives in-between...

CHALLENGE: CRASH CONSISTENCY



Persistent Memory System

System crash can result in permanent data corruption in NVM

CURRENT SOLUTIONS

Explicit interfaces to manage consistency

— NV-Heaps [ASPLOS'11], BPFS [SOSP'09], Mnemosyne [ASPLOS'11]

```
AtomicBegin {  
    Insert a new node;  
} AtomicEnd;
```

Limits adoption of NVM

Have to rewrite code with clear partition
between volatile and non-volatile data

Burden on the programmers

OUR APPROACH: ThyNVM

Goal:
**Software transparent consistency in
persistent memory systems**

ThyNVM: Summary

A new hardware-based checkpointing mechanism

- **Checkpoints** at *multiple granularities* to reduce both checkpointing latency and metadata overhead
- **Overlaps** *checkpointing* and *execution* to reduce checkpointing latency
- **Adapts** to *DRAM and NVM* characteristics

Performs within **4.9%** of an *idealized DRAM* with zero cost consistency

More About ThyNVM

- Ren+, “ThyNVM: Enabling Software-Transparent Crash Consistency in Persistent Memory Systems,” MICRO 2015.

ThyNVM: Enabling Software-Transparent Crash Consistency in Persistent Memory Systems

Jinglei Ren^{*†} Jishen Zhao[‡] Samira Khan^{†'} Jongmoo Choi^{+†} Yongwei Wu^{*} Onur Mutlu[†]

[†]Carnegie Mellon University ^{*}Tsinghua University

[‡]University of California, Santa Cruz [']University of Virginia ⁺Dankook University

Agenda

- Major Trends Affecting Main Memory
- The Memory Scaling Problem and Solution Directions
 - New Memory Architectures
 - Enabling Emerging Technologies
- Cross-Cutting Principles
- Summary

Principles (So Far)

- Better interfaces between layers of the system stack
 - Expose more information judiciously across the system stack
 - Design more flexible and efficient 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)
- These principles are coupled

Agenda

- Major Trends Affecting Main Memory
- The Memory Scaling Problem and Solution Directions
 - New Memory Architectures
 - Enabling Emerging Technologies
- Cross-Cutting Principles
- Summary

Summary

Business as Usual	Opportunity
RowHammer	Memory controller anticipates and fixes errors
Fixed, frequent refreshes	Heterogeneous refresh rate across memory
Fixed, high latency	Heterogeneous latency in time and space
Slow page copy & initialization	Exploit internal connectivity in memory to move data
Fixed reliability mechanisms	Heterogeneous reliability across time and space
Memory as a dumb device	Memory as an accelerator and autonomous agent
DRAM-only main memory	Emerging memory technologies and hybrid memories
Two-level data storage model	Unified interface to and management of all data
Large timing and error margins	Online adaptation of timing and error margins
Poor performance guarantees	Strong service guarantees and configurable QoS
Fixed policies in controllers	Configurable and programmable memory controllers
...	...

Summary

- Memory problems are a critical bottleneck for system performance, efficiency, and usability
- New memory architectures
 - Compute capable and autonomous memory
- Enabling emerging NVM technologies
 - Persistent and hybrid memory
- System-level memory/storage QoS
 - Predictable systems with configurable QoS
- **Many opportunities and challenges that will change the systems and software we design**

Acknowledgments

■ My current and past students and postdocs

- ❑ Rachata Ausavarungnirun, Abhishek Bhowmick, Amirali Boroumand, Rui Cai, Yu Cai, Kevin Chang, Saugata Ghose, Kevin Hsieh, Tyler Huberty, Ben Jaiyen, Samira Khan, Jeremie Kim, Yoongu Kim, Yang Li, Jamie Liu, Lavanya Subramanian, Donghyuk Lee, Yixin Luo, Justin Meza, Gennady Pekhimenko, Vivek Seshadri, Lavanya Subramanian, Nandita Vijaykumar, HanBin Yoon, Jishen Zhao, ...

■ My collaborators

- ❑ Can Alkan, Chita Das, Phil Gibbons, Sriram Govindan, Norm Jouppi, Mahmut Kandemir, Mike Kozuch, Konrad Lai, Ken Mai, Todd Mowry, Yale Patt, Moinuddin Qureshi, Partha Ranganathan, Bikash Sharma, Kushagra Vaid, Chris Wilkerson, ...

Funding Acknowledgments

- NSF
- GSRC
- SRC
- CyLab
- AMD, Google, Facebook, HP Labs, Huawei, IBM, Intel, Microsoft, Nvidia, Oracle, Qualcomm, Rambus, Samsung, Seagate, VMware

Open Source Tools

- Rowhammer

- <https://github.com/CMU-SAFARI/rowhammer>

- Ramulator – Fast and Extensible DRAM Simulator

- <https://github.com/CMU-SAFARI/ramulator>

- MemSim

- <https://github.com/CMU-SAFARI/memsim>

- NOCulator

- <https://github.com/CMU-SAFARI/NOCulator>

- DRAM Error Model

- <http://www.ece.cmu.edu/~safari/tools/memerr/index.html>

- Other open-source software from my group

- <https://github.com/CMU-SAFARI/>

- <http://www.ece.cmu.edu/~safari/tools.html>

Referenced Papers

- All are available at
<http://users.ece.cmu.edu/~omutlu/projects.htm>
<http://scholar.google.com/citations?user=7XyGUGkAAAAJ&hl=en>
- A detailed accompanying overview paper
 - Onur Mutlu and Lavanya Subramanian,
"Research Problems and Opportunities in Memory Systems"
Invited Article in Supercomputing Frontiers and Innovations (SUPERFRI), 2015.

Related Videos and Course Materials

- **Undergraduate Computer Architecture Course Lecture Videos (2013, 2014, 2015)**
- **Undergraduate Computer Architecture Course Materials (2013, 2014, 2015)**
- **Graduate Computer Architecture Lecture Videos (2013, 2015)**
- **Graduate Computer Architecture Course Materials (2013, 2015)**
- **Parallel Computer Architecture Course Materials (Lecture Videos)**
- **Memory Systems Short Course Materials (Lecture Video on Main Memory and DRAM Basics)**

Thank you.

onur.mutlu@inf.ethz.ch

<http://users.ece.cmu.edu/~omutlu/>

Rethinking Memory System Design Business as Usual in the Next Decade?

Onur Mutlu

onur.mutlu@inf.ethz.ch

<http://users.ece.cmu.edu/~omutlu/>

October 6, 2016

RSP 2016 Keynote (Pittsburgh)

Backup Slides

NAND Flash Memory Scaling

Another Talk: NAND Flash Scaling Challenges

- Onur Mutlu,
"Error Analysis and Management for MLC NAND Flash Memory"
Technical talk at Flash Memory Summit 2014 (FMS), Santa Clara, CA, August 2014. Slides (ppt) (pdf)

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 Technology 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.

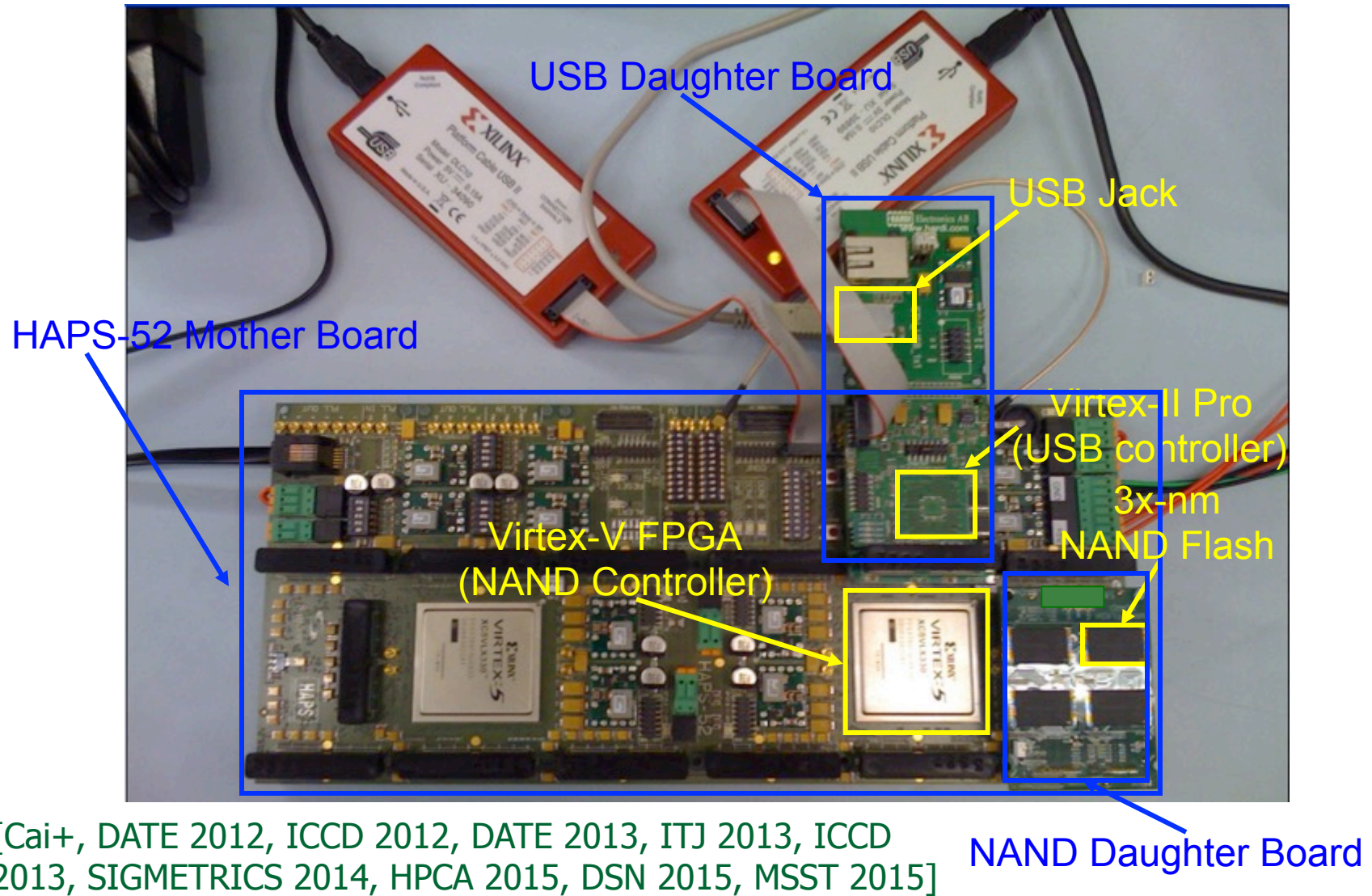
Cai+, "Data Retention in MLC NAND Flash Memory: Characterization, Optimization and Recovery," HPCA 2015.

Cai+, "Read Disturb Errors in MLC NAND Flash Memory: Characterization and Mitigation," DSN 2015.

Luo+, "WARM: Improving NAND Flash Memory Lifetime with Write-hotness Aware Retention Management," MSST 2015.

Meza+, "A Large-Scale Study of Flash Memory Errors in the Field," SIGMETRICS 2015.

Experimental Infrastructure (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]

Ramulator: A Fast and Extensible DRAM Simulator

[IEEE Comp Arch Letters'15]

Ramulator Motivation

- DRAM and Memory Controller landscape is changing
- Many new and upcoming standards
- Many new controller designs
- A fast and easy-to-extend simulator is very much needed

<i>Segment</i>	<i>DRAM Standards & Architectures</i>
Commodity	DDR3 (2007) [14]; DDR4 (2012) [18]
Low-Power	LPDDR3 (2012) [17]; LPDDR4 (2014) [20]
Graphics	GDDR5 (2009) [15]
Performance	eDRAM [28], [32]; RLDram3 (2011) [29]
3D-Stacked	WIO (2011) [16]; WIO2 (2014) [21]; MCDRAM (2015) [13]; HBM (2013) [19]; HMC1.0 (2013) [10]; HMC1.1 (2014) [11]
Academic	SBA/SSA (2010) [38]; Staged Reads (2012) [8]; RAIDR (2012) [27]; SALP (2012) [24]; TL-DRAM (2013) [26]; RowClone (2013) [37]; Half-DRAM (2014) [39]; Row-Buffer Decoupling (2014) [33]; SARP (2014) [6]; AL-DRAM (2015) [25]

Table 1. Landscape of DRAM-based memory

Ramulator

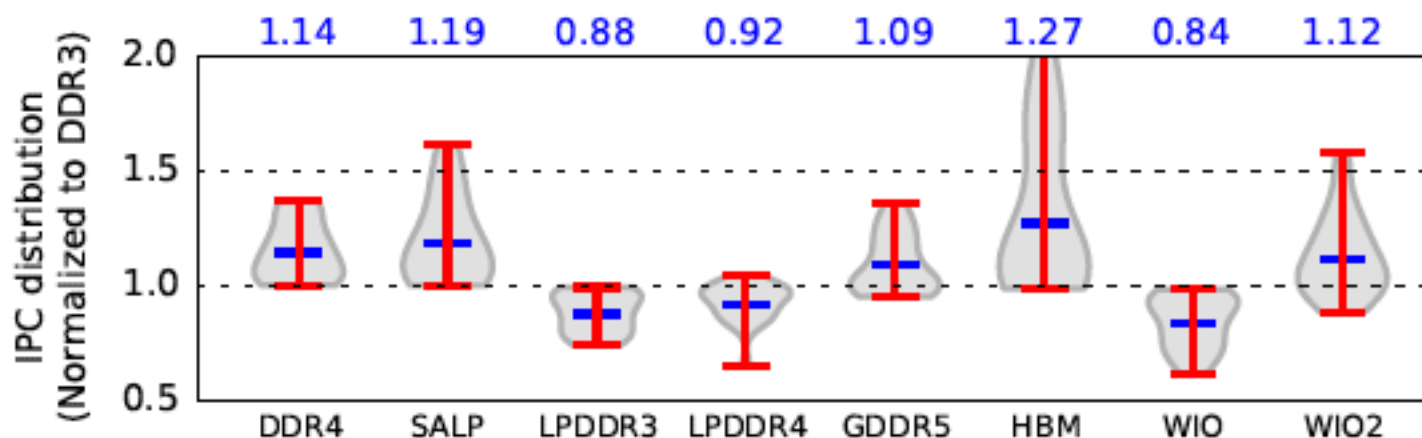
- Provides out-of-the box support for many DRAM standards:
 - DDR3/4, LPDDR3/4, GDDR5, WIO1/2, HBM, plus new proposals (SALP, AL-DRAM, TLDRAM, RowClone, and SARP)
- ~2.5X faster than fastest open-source simulator
- Modular and extensible to different standards

<i>Simulator</i> (clang -O3)	<i>Cycles (10⁶)</i>		<i>Runtime (sec.)</i>		<i>Req/sec (10³)</i>		<i>Memory</i> (MB)
	<i>Random</i>	<i>Stream</i>	<i>Random</i>	<i>Stream</i>	<i>Random</i>	<i>Stream</i>	
Ramulator	652	411	752	249	133	402	2.1
DRAMSim2	645	413	2,030	876	49	114	1.2
USIMM	661	409	1,880	750	53	133	4.5
DrSim	647	406	18,109	12,984	6	8	1.6
NVMain	666	413	6,881	5,023	15	20	4,230.0

Table 3. Comparison of five simulators using two traces

Case Study: Comparison of DRAM Standards

<i>Standard</i>	<i>Rate (MT/s)</i>	<i>Timing (CL-RCD-RP)</i>	<i>Data-Bus (Width×Chan.)</i>	<i>Rank-per-Chan</i>	<i>BW (GB/s)</i>
DDR3	1,600	11-11-11	64-bit × 1	1	11.9
DDR4	2,400	16-16-16	64-bit × 1	1	17.9
SALP [†]	1,600	11-11-11	64-bit × 1	1	11.9
LPDDR3	1,600	12-15-15	64-bit × 1	1	11.9
LPDDR4	2,400	22-22-22	32-bit × 2*	1	17.9
GDDR5 [12]	6,000	18-18-18	64-bit × 1	1	44.7
HBM	1,000	7-7-7	128-bit × 8*	1	119.2
WIO	266	7-7-7	128-bit × 4*	1	15.9
WIO2	1,066	9-10-10	128-bit × 8*	1	127.2



Across 22 workloads, simple CPU model

Figure 2. Performance comparison of DRAM standards

Ramulator Paper and Source Code

- Yoongu Kim, Weikun Yang, and Onur Mutlu,
"Ramulator: A Fast and Extensible DRAM Simulator"
IEEE Computer Architecture Letters (**CAL**), March 2015.
[Source Code]
- Source code is released under the liberal MIT License
 - <https://github.com/CMU-SAFARI/ramulator>

DRAM Infrastructure

Experimental DRAM Testing Infrastructure



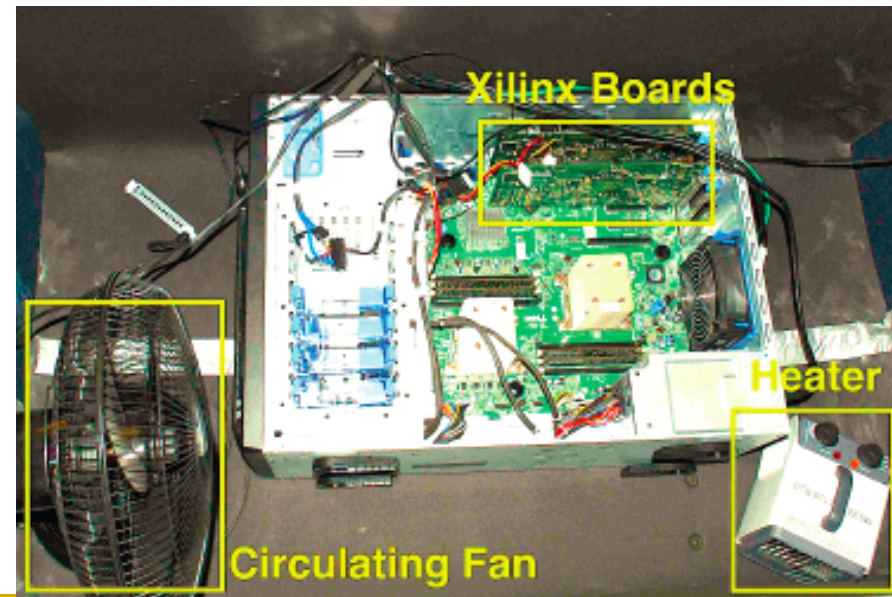
An Experimental Study of Data Retention Behavior in Modern DRAM Devices: Implications for Retention Time Profiling Mechanisms (Liu et al., ISCA 2013)

The Efficacy of Error Mitigation Techniques for DRAM Retention Failures: A Comparative Experimental Study (Khan et al., SIGMETRICS 2014)

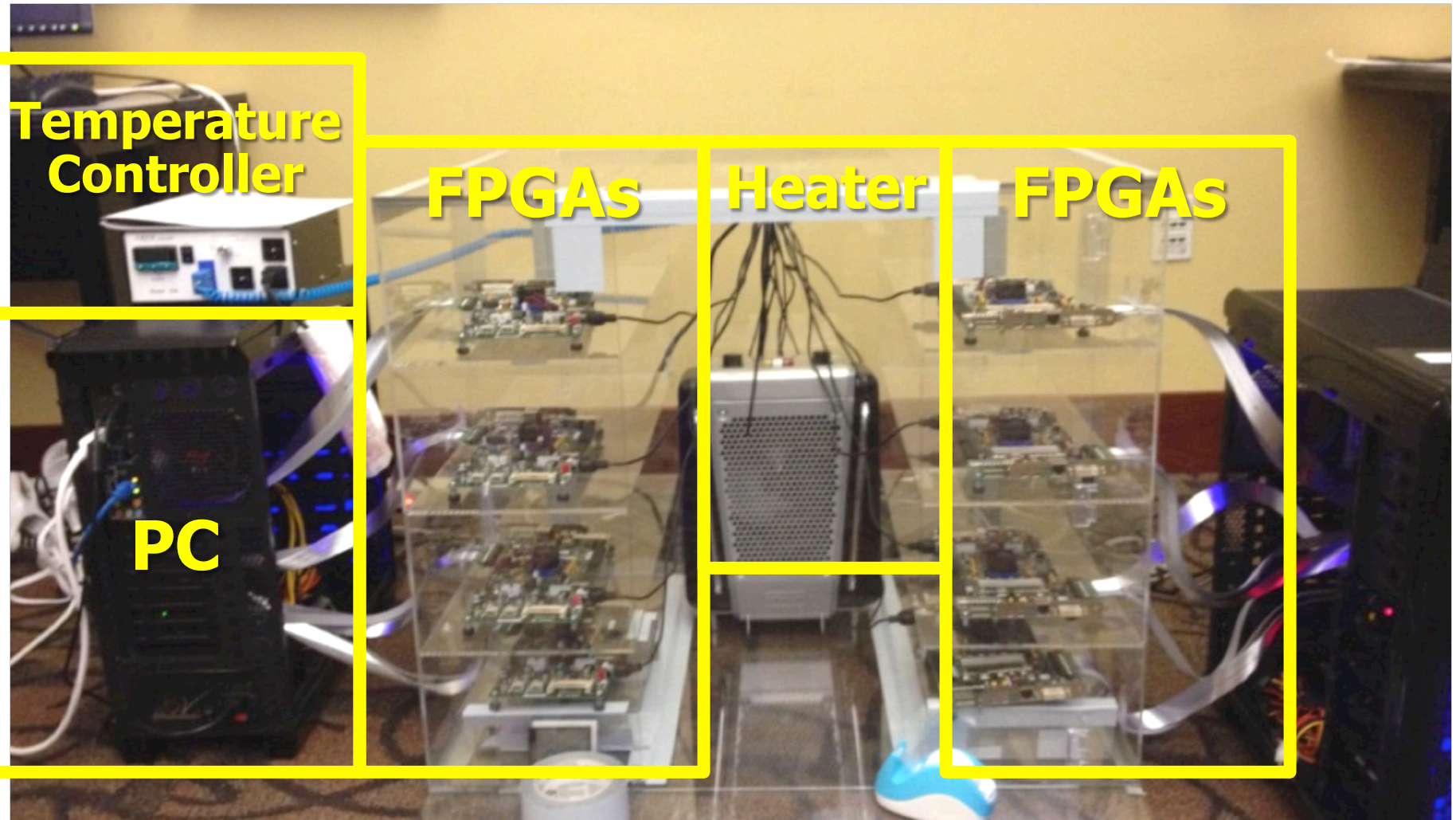
Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors (Kim et al., ISCA 2014)

Adaptive-Latency DRAM: Optimizing DRAM Timing for the Common-Case (Lee et al., HPCA 2015)

AVATAR: A Variable-Retention-Time (VRT) Aware Refresh for DRAM Systems (Qureshi et al., DSN 2015)



Experimental Infrastructure (DRAM)



ThyNVM

CHALLENGE: CRASH CONSISTENCY



Persistent Memory System

**System crash can result in
permanent data corruption in NVM**

CURRENT SOLUTIONS

Explicit interfaces to manage consistency

— NV-Heaps [ASPLOS'11], BPFS [SOSP'09], Mnemosyne [ASPLOS'11]

```
AtomicBegin {  
    Insert a new node;  
} AtomicEnd;
```

Limits adoption of NVM

Have to rewrite code with clear partition
between volatile and non-volatile data

Burden on the programmers

OUR APPROACH: ThyNVM

Goal:
**Software transparent consistency in
persistent memory systems**

ThyNVM: Summary

A new hardware-based checkpointing mechanism

- **Checkpoints** at *multiple granularities* to reduce both checkpointing latency and metadata overhead
- **Overlaps** *checkpointing* and *execution* to reduce checkpointing latency
- **Adapts** to *DRAM and NVM* characteristics

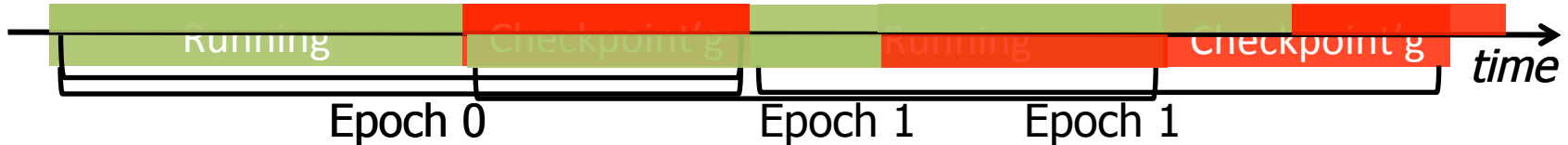
Performs within **4.9%** of an *idealized DRAM* with zero cost consistency

ThyNVM: Transparent Hybrid NVM

- **Problem:** How do you provide consistency and prevent data corruption in NVM upon a system crash?
- **Goal:** Provide efficient programmer-transparent crash consistency in hybrid NVM
 - **Transparency:** no library APIs or explicit interfaces to access NVM; just loads and stores
 - Easier to support legacy code and hypervisors
 - No programmer effort to adopt persistent memory
 - **Efficiency:** use hybrid DRAM/NVM for high performance

ThyNVM

- Idea 1: Transparent periodic checkpointing of data



- Need to overlap checkpointing and execution
- Idea 2: Differentiated checkpointing schemes for different types of updates
 - Page Writeback: for sequential accesses (use DRAM)
 - Address Remapping: for random accesses (use NVM/DRAM)
- Idea 3: Coordination/switching between checkpointing schemes for high performance

Checkpointing Tradeoffs in Hybrid Memory

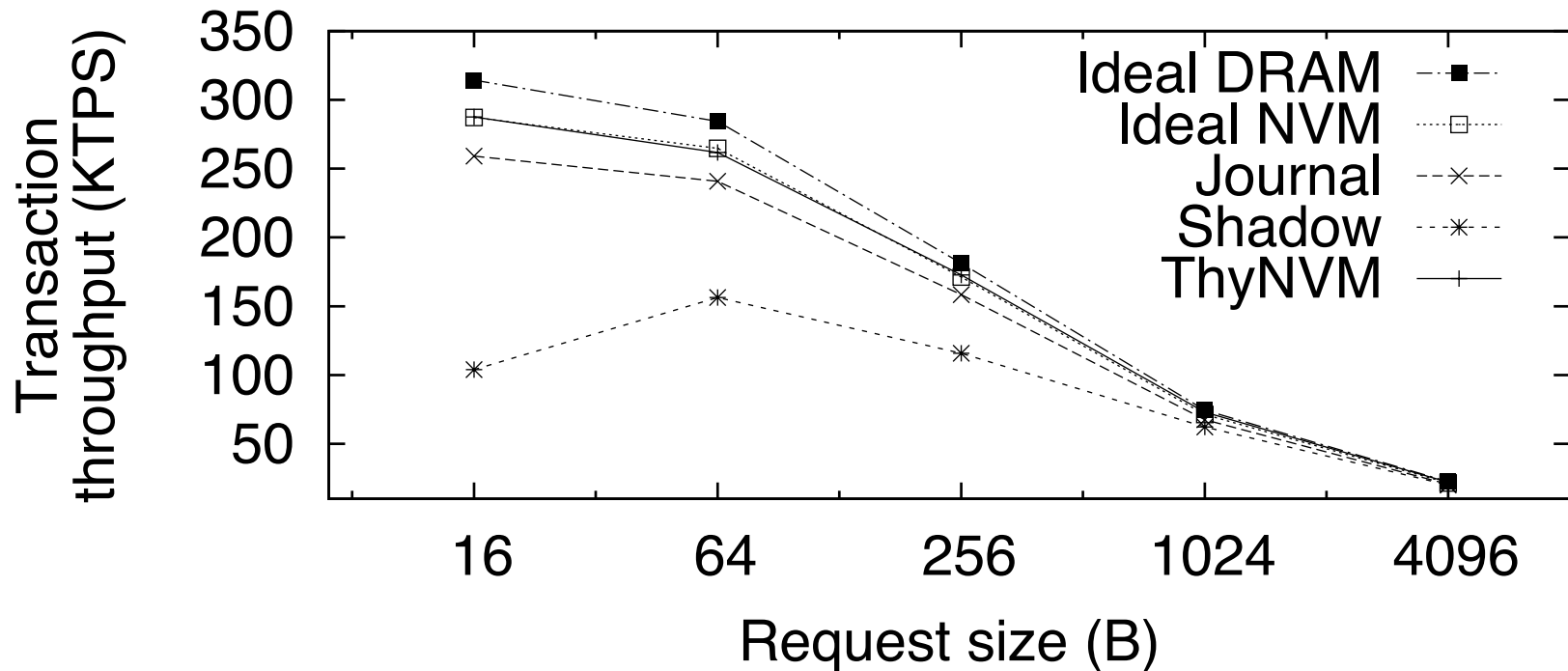
		Checkpointing granularity	
		Small (cache block)	Large (page)
Location of working copy	DRAM (based on writeback)	❶ Inefficient × Large metadata overhead × Long checkpointing latency	❷ Partially efficient ✓ Small metadata overhead × Long checkpointing latency
	NVM (based on remapping)	❸ Partially efficient × Large metadata overhead ✓ Short checkpointing latency ✓ Fast remapping	❹ Inefficient ✓ Small metadata overhead ✓ Short checkpointing latency × Slow remapping (on the critical path)

ThyNVM: Dual-Scheme Checkpointing

- Idea: Combine two types of checkpointing schemes to adapt to different types of access patterns
- Sparse updates with low spatial locality → address remapping
 - block granularity checkpointing
 - working copy stored in NVM (for short ckpt latency)
- Dense updates with high spatial locality → page writeback
 - page granularity checkpointing (small metadata)
 - working copy stored in DRAM for fast buffering; written back to NVM during ckpt.
- Can switch between schemes when one is on critical path

ThyNVM Performance (I)

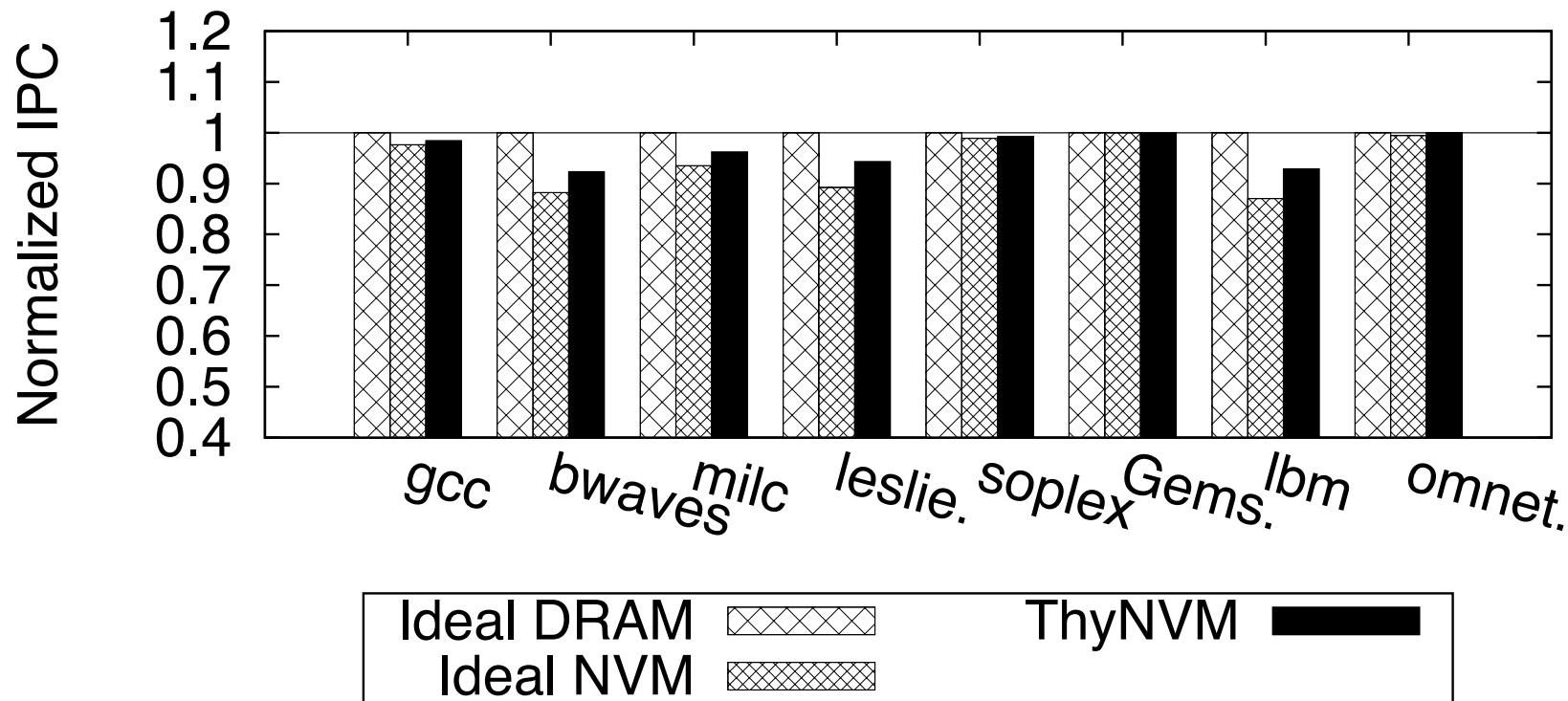
■ In-memory storage workloads



8.8%/29.9% higher throughput than journaling/shadow paging with a hash table based key-value store

ThyNVM Performance (II)

■ Legacy compute-intensive workloads



- Within 3.4% of Ideal DRAM,
- 2.7% higher performance than Ideal NVM.

New Memory Architectures

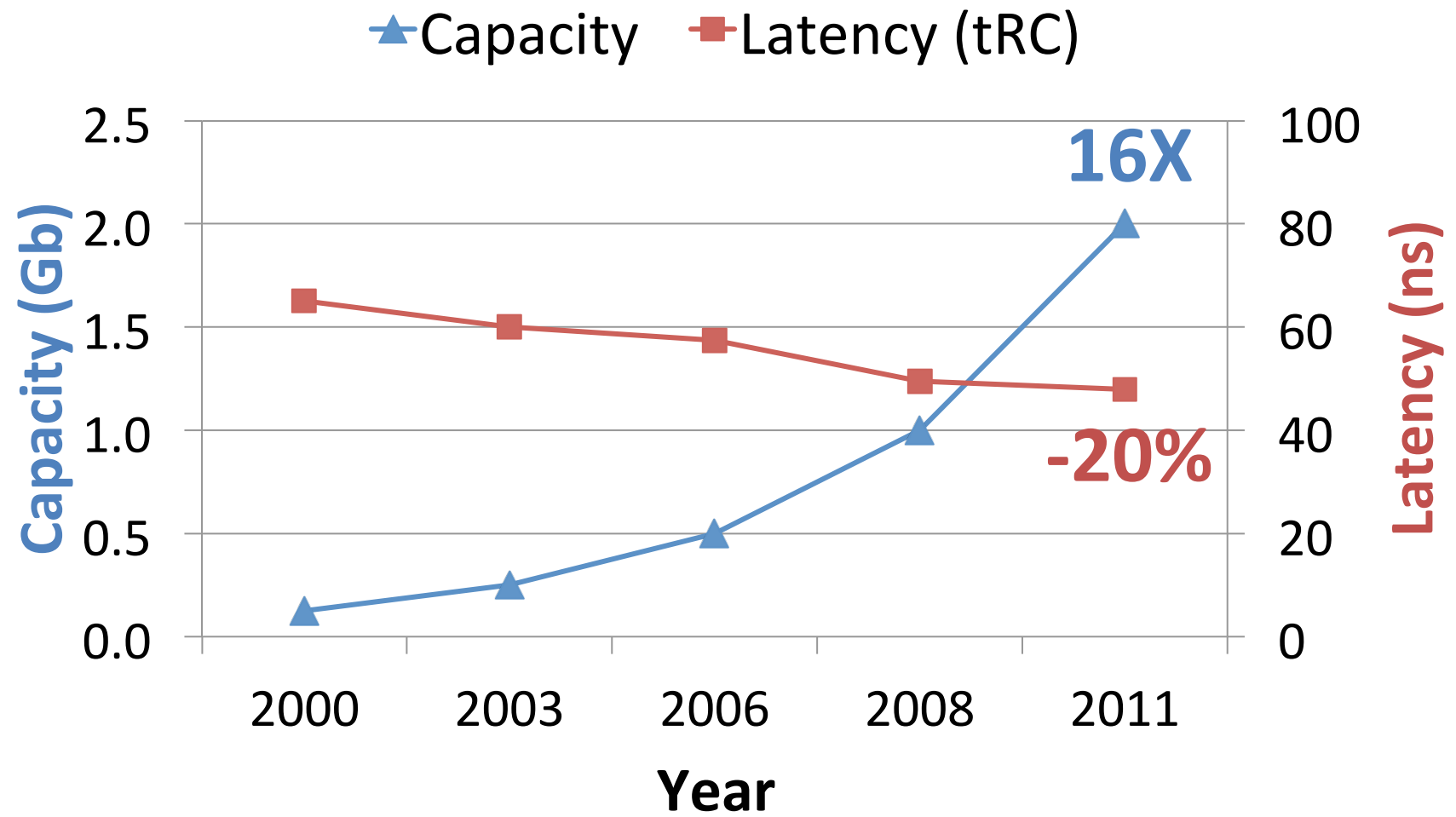
- Compute Capable Memory
- Refresh
- Reliability
- Latency
- Bandwidth
- Energy
- Memory Compression

DRAM Latency

New Memory Architectures

- Compute Capable Memory
- Refresh
- Reliability
- Latency
- Bandwidth
- Energy
- Memory Compression

DRAM Latency-Capacity Trend



DRAM latency continues to be a critical bottleneck, especially for response time-sensitive workloads 168

What Causes the Long Memory Latency?

- **Conservative timing margins!**
- DRAM timing parameters are set to cover the worst case
- **Worst-case temperatures**
 - 85 degrees vs. common-case
 - to enable a wide range of operating conditions
- **Worst-case devices**
 - DRAM cell with smallest charge across any acceptable device
 - to tolerate process variation at acceptable yield
- This leads to large timing margins for the common case

Adaptive-Latency DRAM [HPCA 2015]

- Idea: Optimize DRAM timing for the common case
 - Current temperature
 - Current DRAM module
- Why would this reduce latency?
 - A DRAM cell can store much more charge in the common case (low temperature, strong cell) than in the worst case
 - More charge in a DRAM cell
 - Faster sensing, charge restoration, precharging
 - Faster access (read, write, refresh, ...)

AL-DRAM

- *Key idea*
 - Optimize DRAM timing parameters online
- *Two components*
 - DRAM manufacturer provides multiple sets of reliable DRAM timing parameters at different temperatures for each DIMM
 - System monitors DRAM temperature & uses appropriate DRAM timing parameters

Experimental DRAM Testing Infrastructure



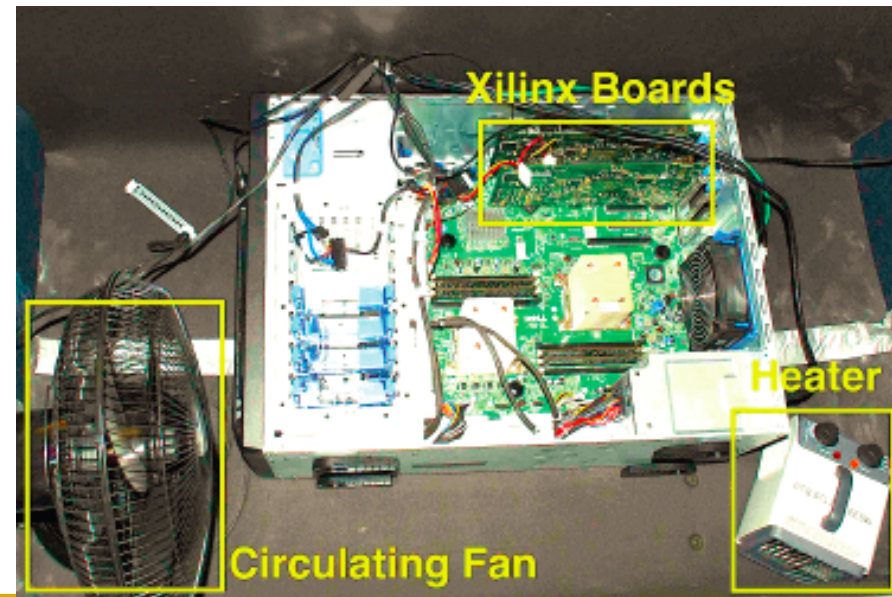
An Experimental Study of Data Retention Behavior in Modern DRAM Devices: Implications for Retention Time Profiling Mechanisms (Liu et al., ISCA 2013)

The Efficacy of Error Mitigation Techniques for DRAM Retention Failures: A Comparative Experimental Study (Khan et al., SIGMETRICS 2014)

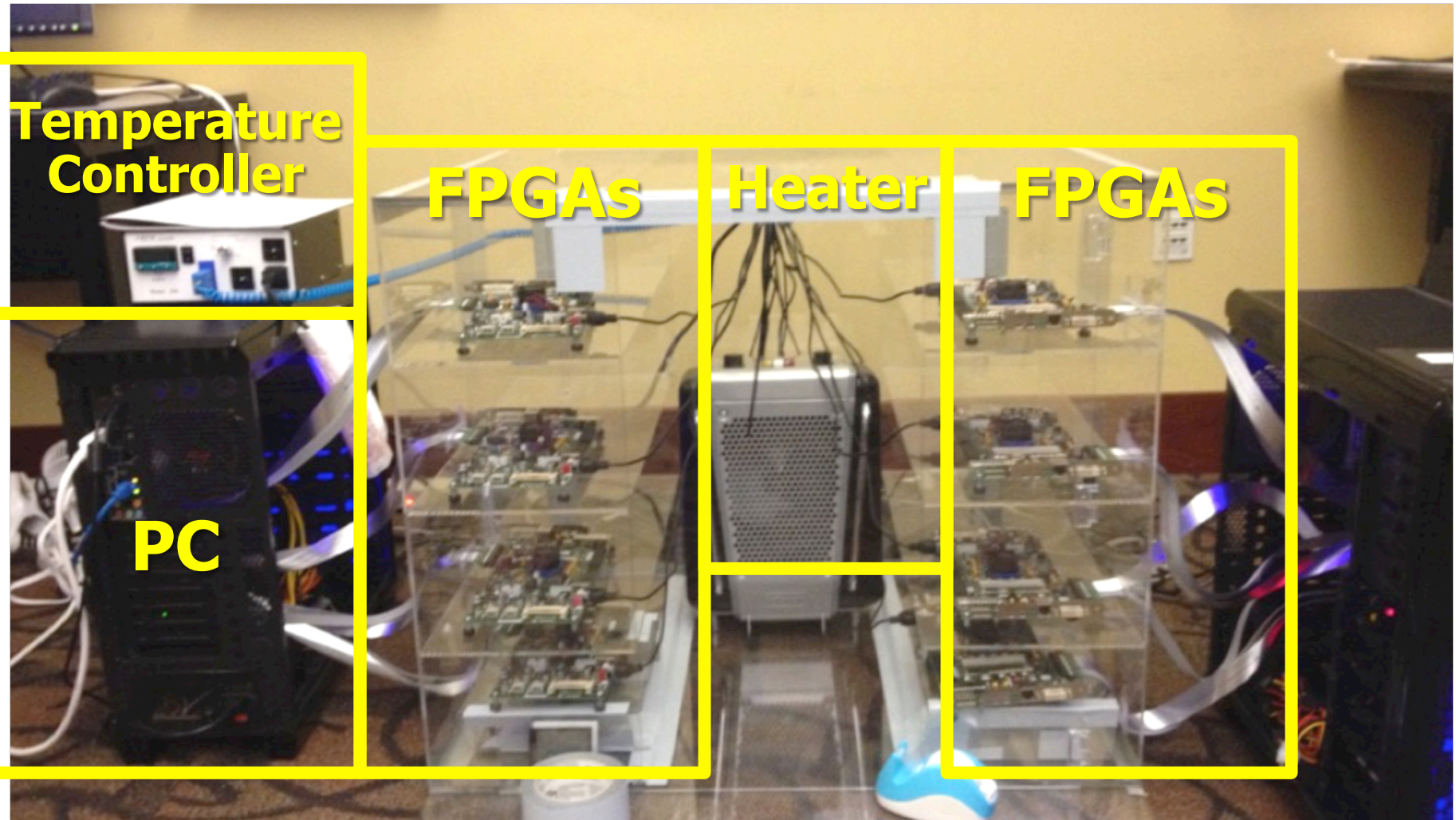
Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors (Kim et al., ISCA 2014)

Adaptive-Latency DRAM: Optimizing DRAM Timing for the Common-Case (Lee et al., HPCA 2015)

AVATAR: A Variable-Retention-Time (VRT) Aware Refresh for DRAM Systems (Qureshi et al., DSN 2015)



Experimental Infrastructure (DRAM)



Latency Reduction Summary of 115 DIMMs

- *Latency reduction for read & write (55°C)*
 - Read Latency: **32.7%**
 - Write Latency: **55.1%**
- *Latency reduction for each timing parameter (55°C)*
 - Sensing: **17.3%**
 - Restore: **37.3%** (read), **54.8%** (write)
 - Precharge: **35.2%**

AL-DRAM: Real System Evaluation

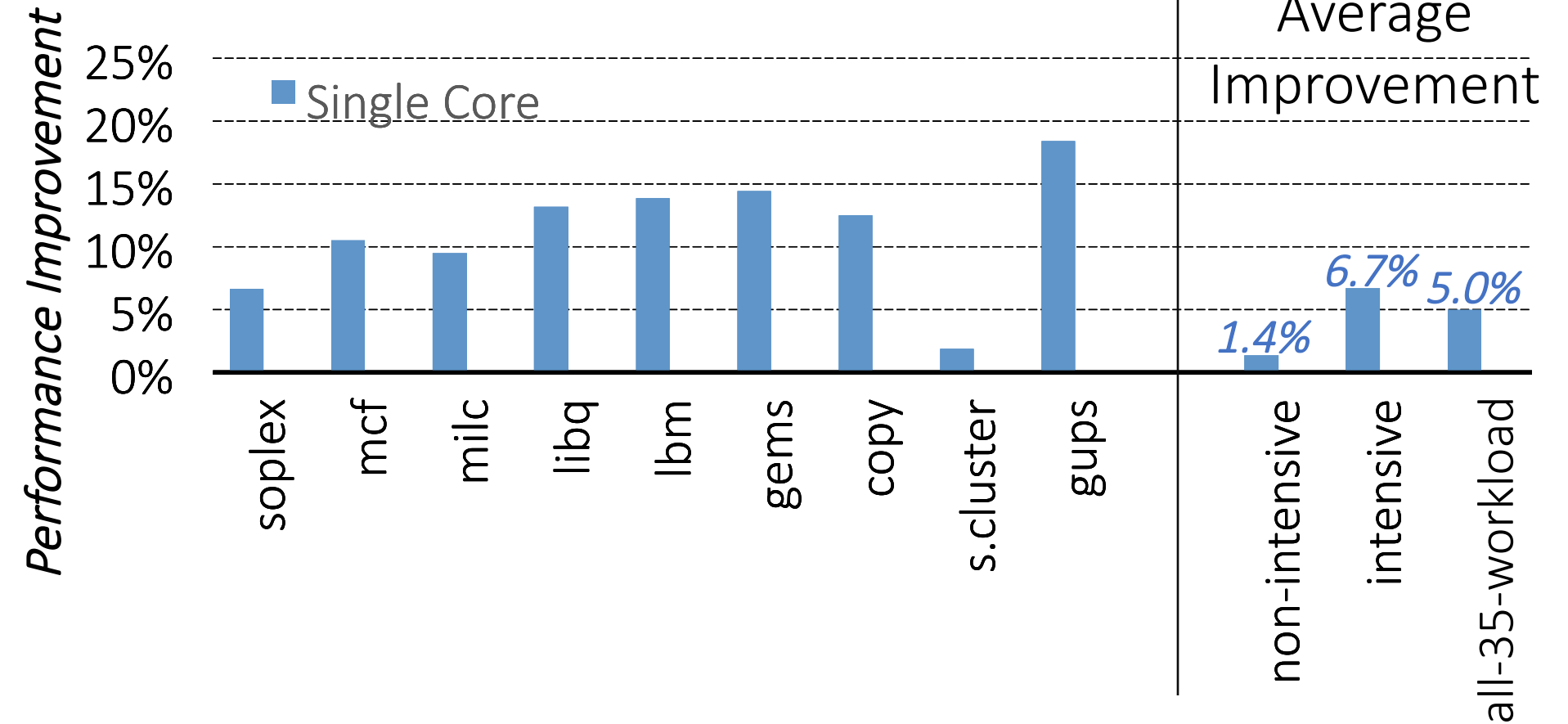
- *System*
 - *CPU: AMD 4386 (8 Cores, 3.1GHz, 8MB LLC)*

D18F2x200_dct[0]_mp[1:0] DDR3 DRAM Timing 0

Reset: 0F05_0505h. See [2.9.3 \[DCT Configuration Registers\]](#).

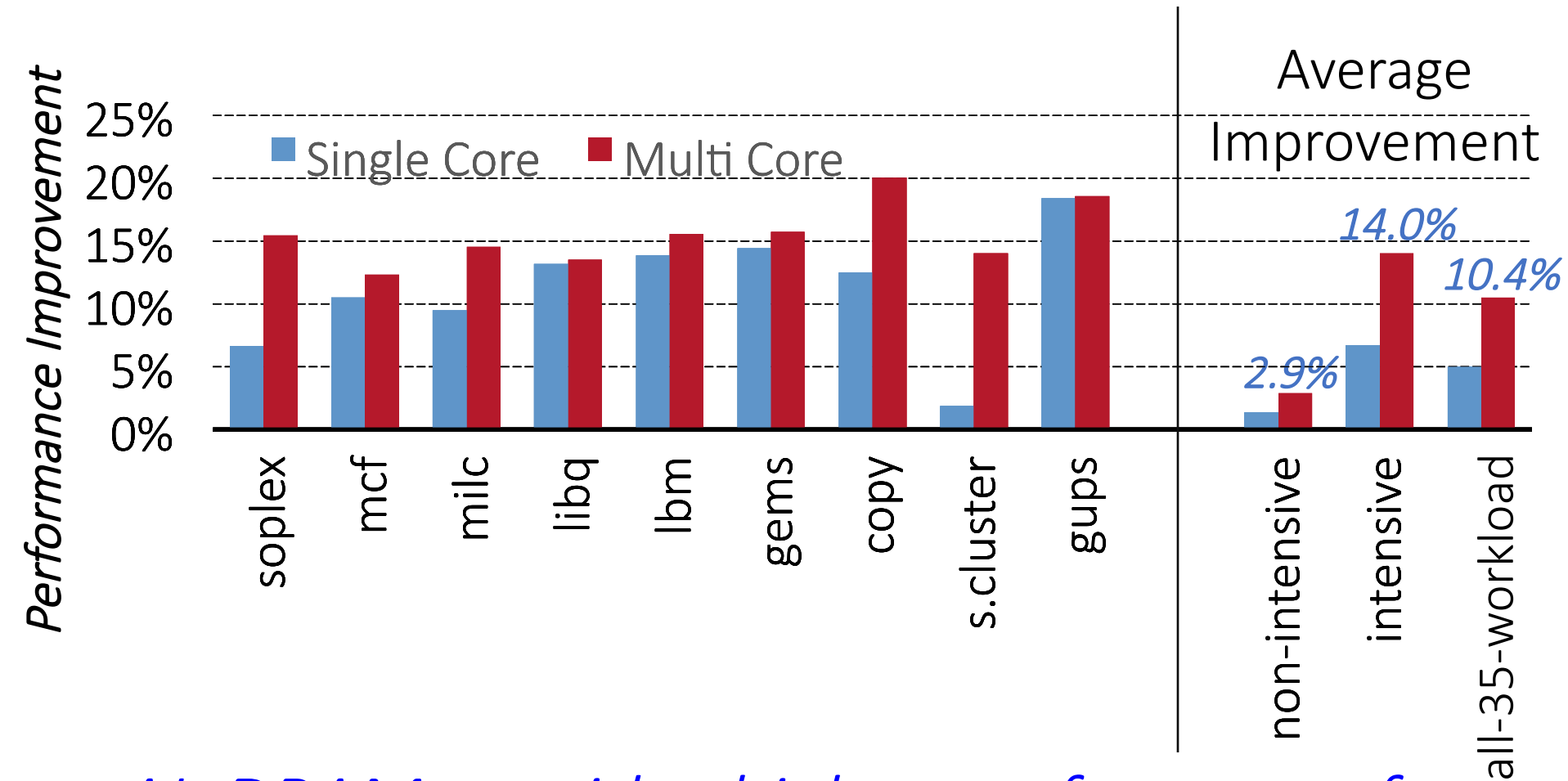
Bits	Description								
31:30	Reserved.								
29:24	Tras: row active strobe. Read-write. BIOS: See 2.9.7.5 [SPD ROM-Based Configuration] . Specifies the minimum time in memory clock cycles from an activate command to a precharge command, both to the same chip select bank. <table><tr><th>Bits</th><th>Description</th></tr><tr><td>07h-00h</td><td>Reserved</td></tr><tr><td>2Ah-08h</td><td><Tras> clocks</td></tr><tr><td>3Fh-2Bh</td><td>Reserved</td></tr></table>	Bits	Description	07h-00h	Reserved	2Ah-08h	<Tras> clocks	3Fh-2Bh	Reserved
Bits	Description								
07h-00h	Reserved								
2Ah-08h	<Tras> clocks								
3Fh-2Bh	Reserved								
23:21	Reserved.								
20:16	Trp: row precharge time. Read-write. BIOS: See 2.9.7.5 [SPD ROM-Based Configuration] . Specifies the minimum time in memory clock cycles from a precharge command to an activate command or auto refresh command, both to the same bank.								

AL-DRAM: Single-Core Evaluation



AL-DRAM improves performance on a real system

AL-DRAM: Multi-Core Evaluation



AL-DRAM provides higher performance for multi-programmed & multi-threaded workloads

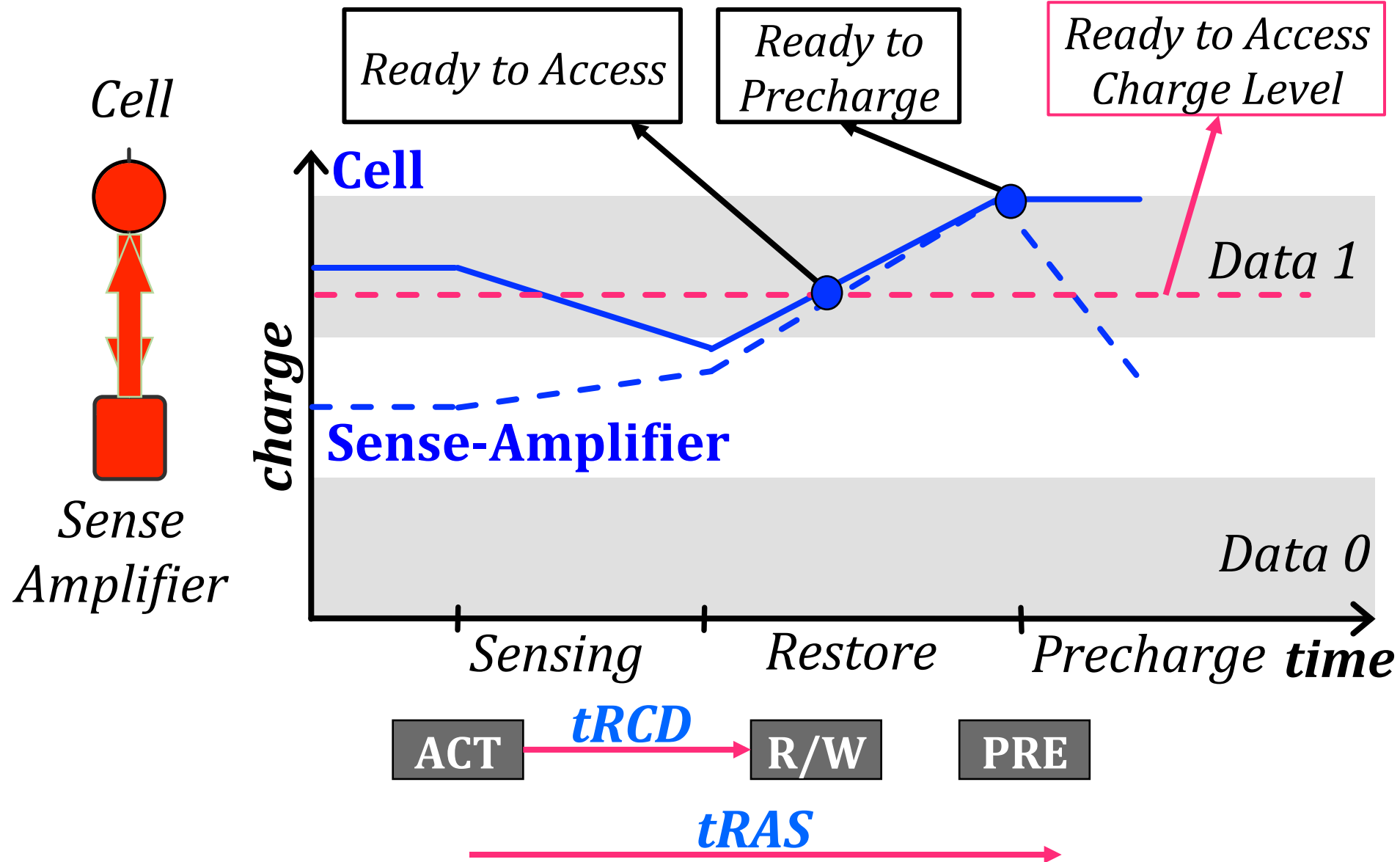
ChargeCache

ChargeCache: Executive Summary

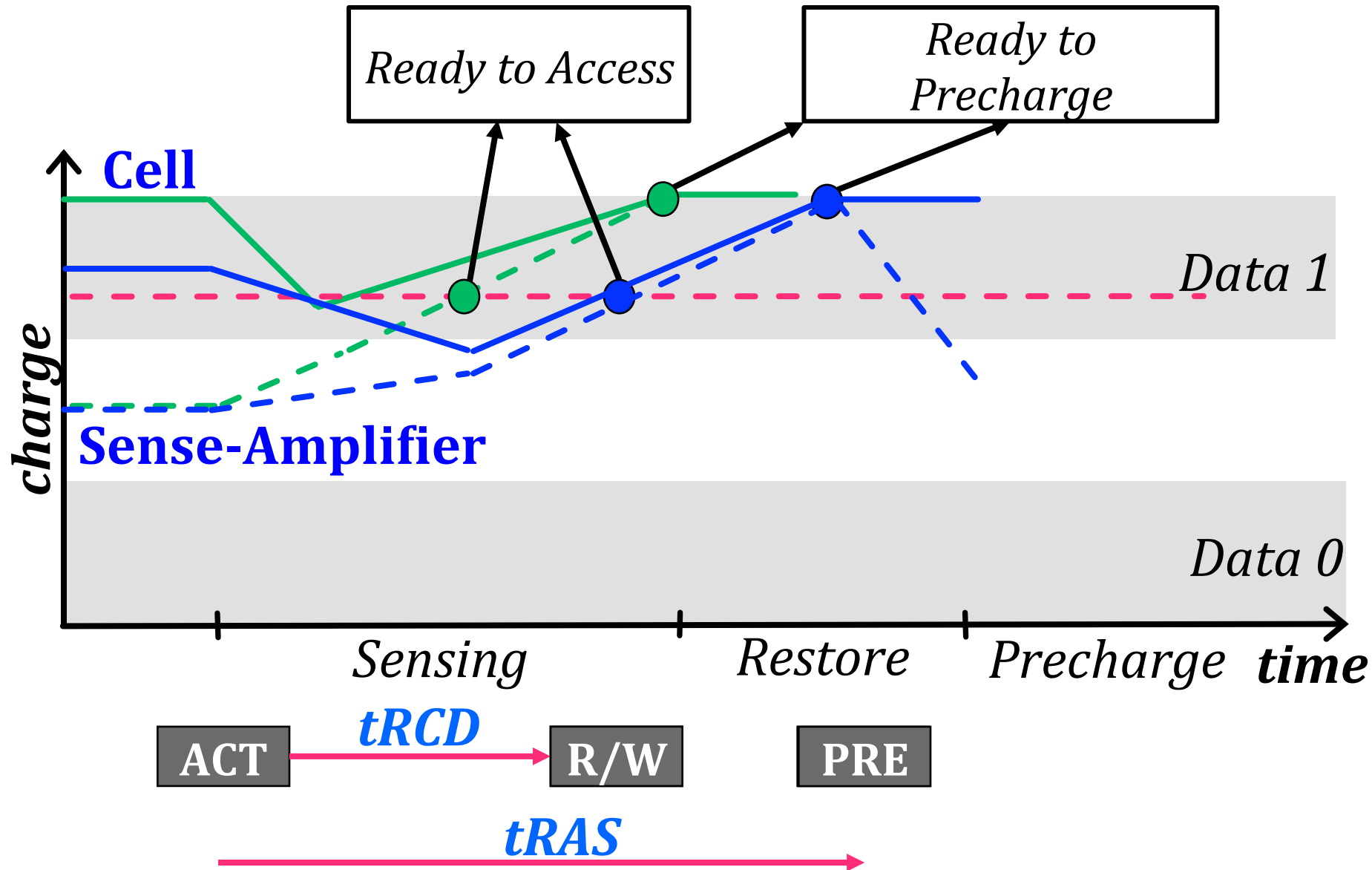
- **Goal**: Reduce average DRAM access latency with no modification to the existing DRAM chips
- **Observations**:
 - 1) A highly-charged DRAM row can be accessed with low latency
 - 2) A row's charge is restored when the row is accessed
 - 3) A recently-accessed row is likely to be accessed again:

Row Level Temporal Locality (RLTL)
- **Key Idea**: Track recently-accessed DRAM rows and use lower timing parameters if such rows are accessed again
- **ChargeCache**:
 - Low cost & no modifications to the DRAM
 - Higher performance (**8.6-10.6%** on average for 8-core)
 - Lower DRAM energy (**7.9%** on average)

DRAM Charge over Time



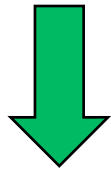
Accessing Highly-charged Rows



Observation 1

A **highly-charged** DRAM row can be accessed with **low latency**

- tRCD: 44%
- tRAS: 37%



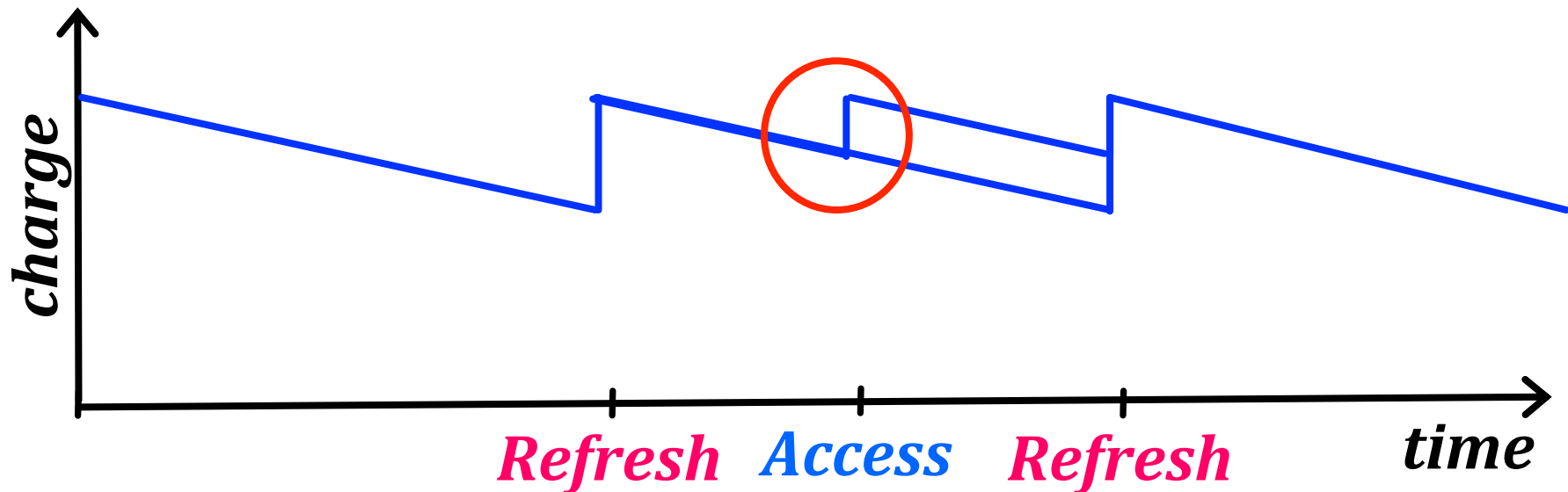
**How does a row become
highly-charged?**

How Does a Row Become Highly-Charged?

DRAM cells **lose charge** over time

Two ways of restoring a row's charge:

- Refresh Operation
- Access



Observation 2

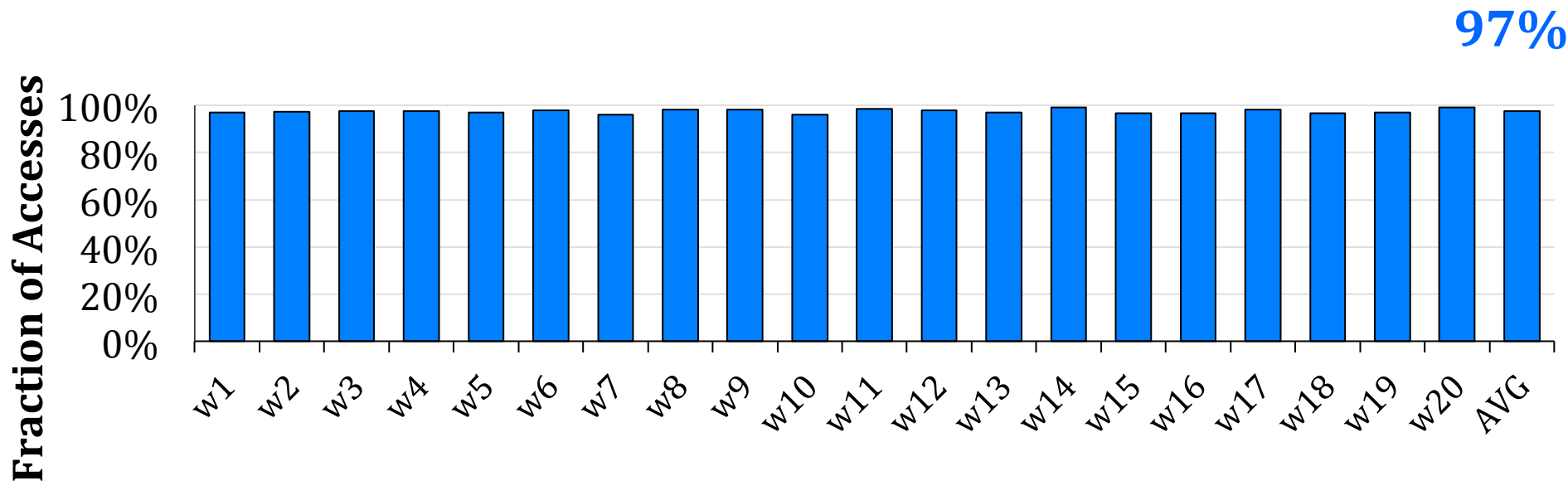
A row's charge is **restored** when the row is **accessed**

How likely is a **recently-accessed row to be accessed again?**

Row Level Temporal Locality (RLTL)

A **recently-accessed** DRAM row is likely to be accessed again.

- t -RLTL: Fraction of rows that are accessed within time t after their previous access

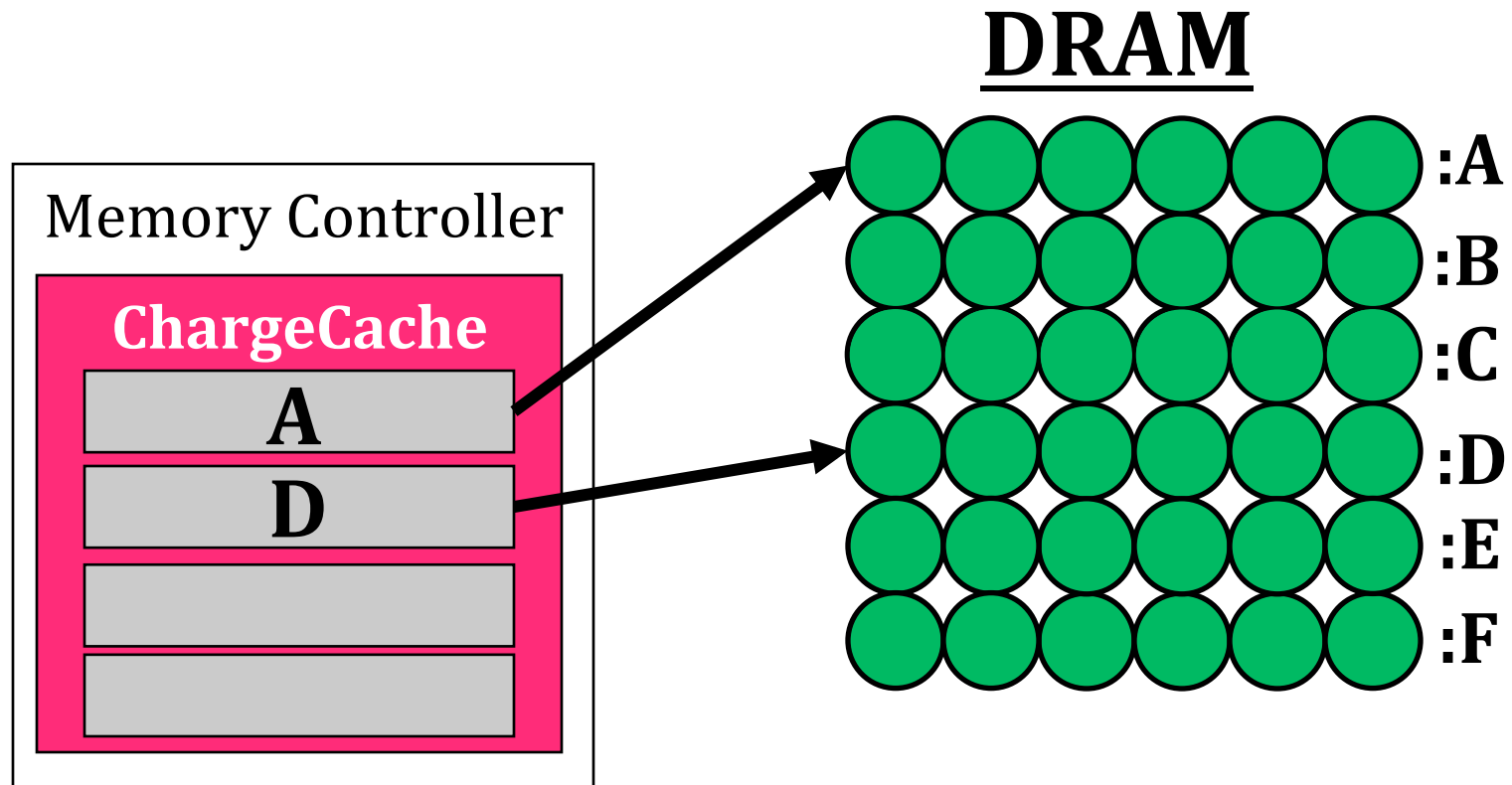


88ns — RLTL for eight-core workloads

Key Idea

Track **recently-accessed** DRAM rows and use **lower timing parameters** if such rows are accessed again

ChargeCache Overview



Requests: A D A 

ChargeCache Hits: Use Default Timings

Area and Power Overhead

- Modeled with CACTI
- Area
 - ~5KB for 128-entry ChargeCache
 - 0.24% of a 4MB Last Level Cache (LLC) area
- Power Consumption
 - 0.15 mW on average (static + dynamic)
 - 0.23% of the 4MB LLC power consumption

Methodology

- **Simulator**

- DRAM Simulator (Ramulator *[Kim+, CAL'15]*)
<https://github.com/CMU-SAFARI/ramulator>

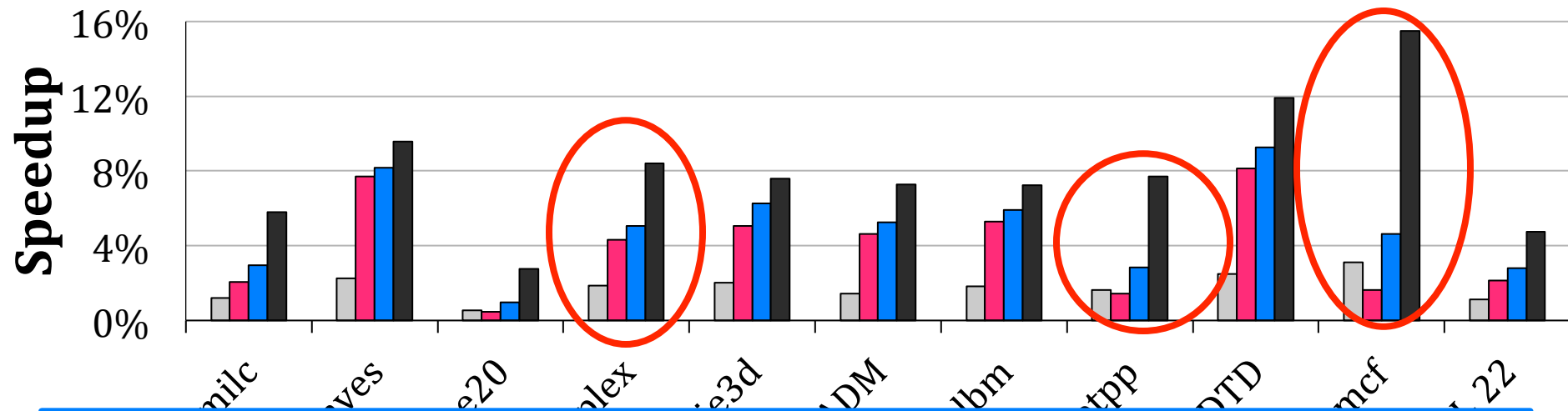
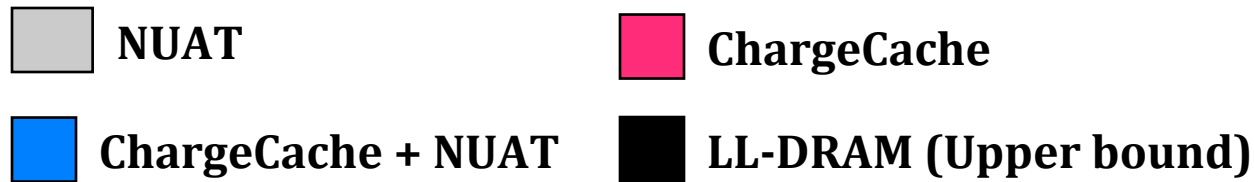
- **Workloads**

- 22 single-core workloads
 - SPEC CPU2006, TPC, STREAM
- 20 multi-programmed 8-core workloads
 - By randomly choosing from single-core workloads
- Execute at least 1 billion representative instructions per core (Pinpoints)

- **System Parameters**

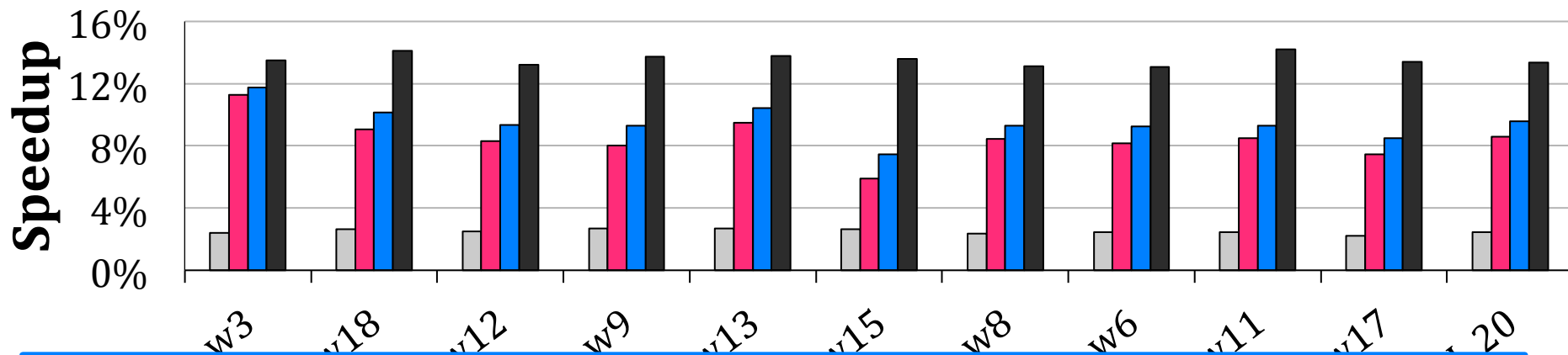
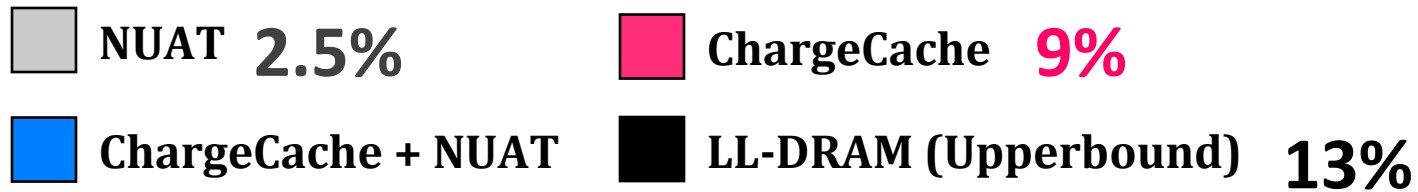
- 1/8 core system with 4MB LLC
- Default tRCD/tRAS of 11/28 cycles

Single-core Performance



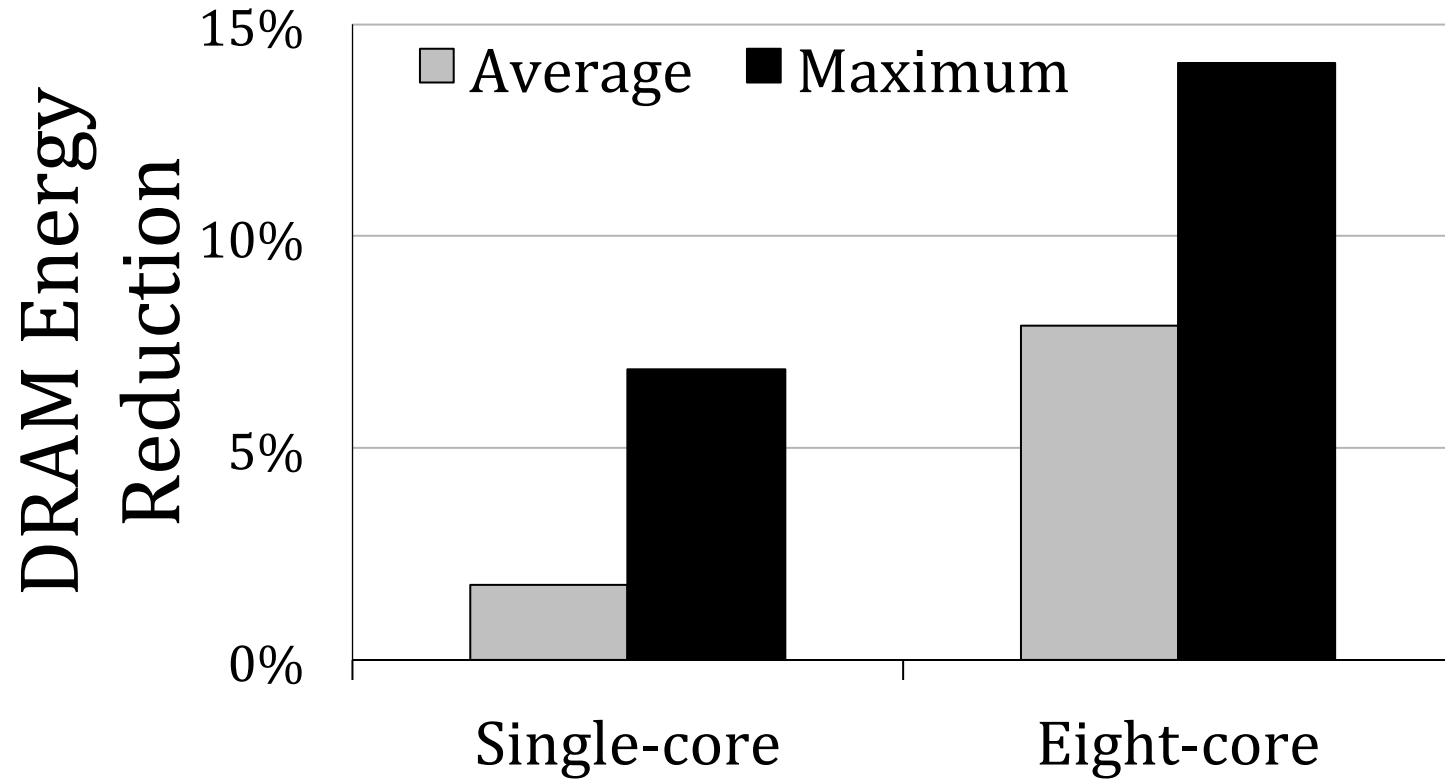
**ChargeCache improves
single-core performance**

Eight-core Performance



ChargeCache significantly improves multi-core performance

DRAM Energy Savings



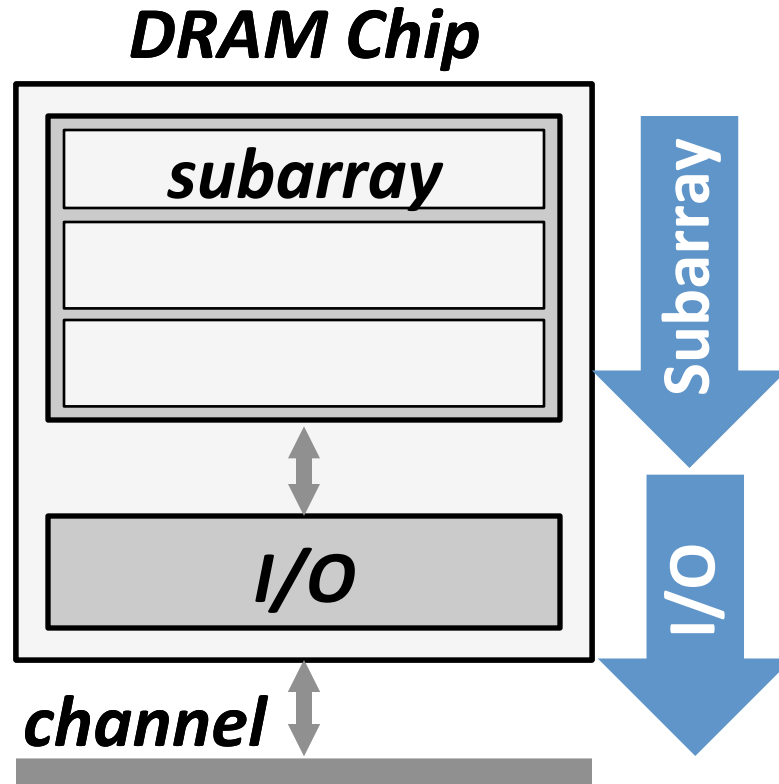
ChargeCache reduces DRAM energy

More on ChargeCache

- Hasan Hassan, Gennady Pekhimenko, Nandita Vijaykumar, Vivek Seshadri, Donghyuk Lee, Oguz Ergin, and Onur Mutlu, **"ChargeCache: Reducing DRAM Latency by Exploiting Row Access Locality"**
Proceedings of the 22nd International Symposium on High-Performance Computer Architecture (HPCA), Barcelona, Spain, March 2016.
[Slides (pptx)] [pdf]
- Source code will be released as part of Ramulator (May 2016)
 - <https://github.com/CMU-SAFARI/ramulator>

Tiered Latency DRAM

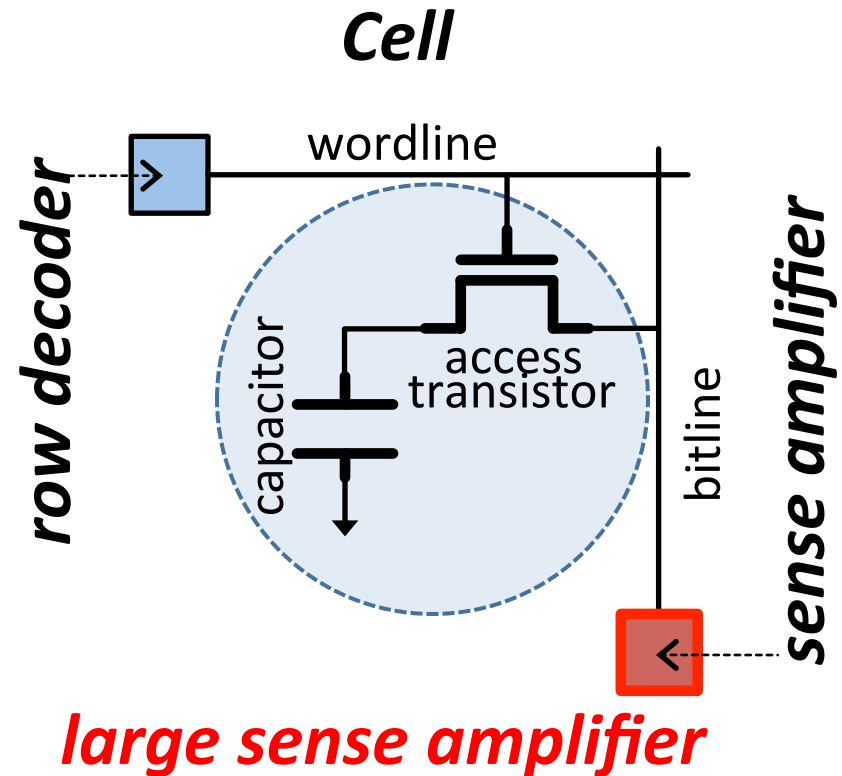
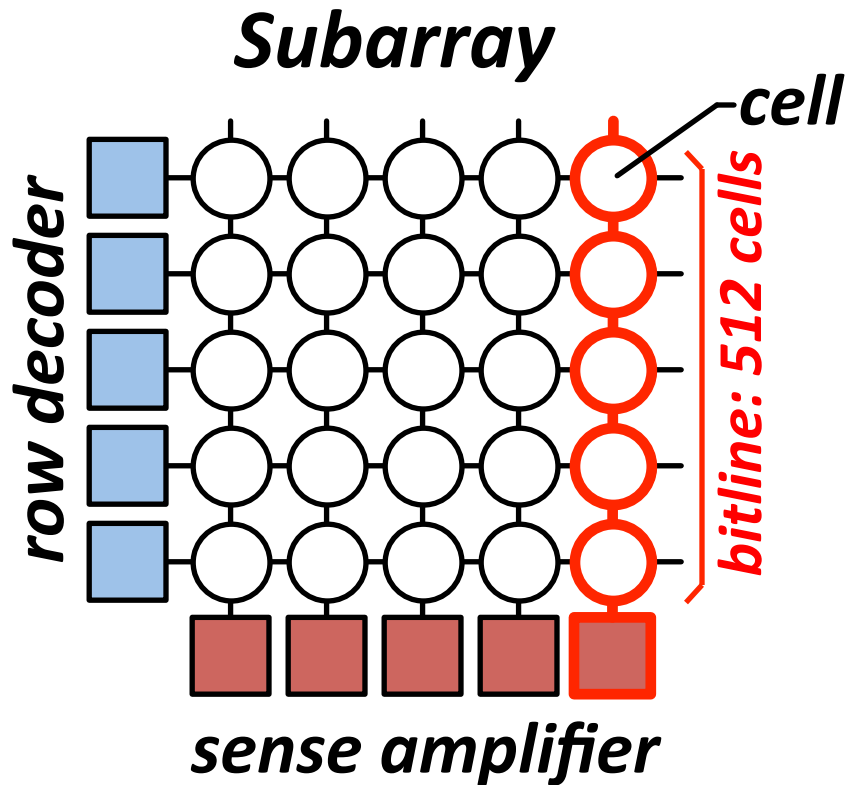
What Causes the Long Latency?



$DRAM\ Latency = \text{Subarray Latency} + I/O\ Latency$

Dominant

Why is the Subarray So Slow?

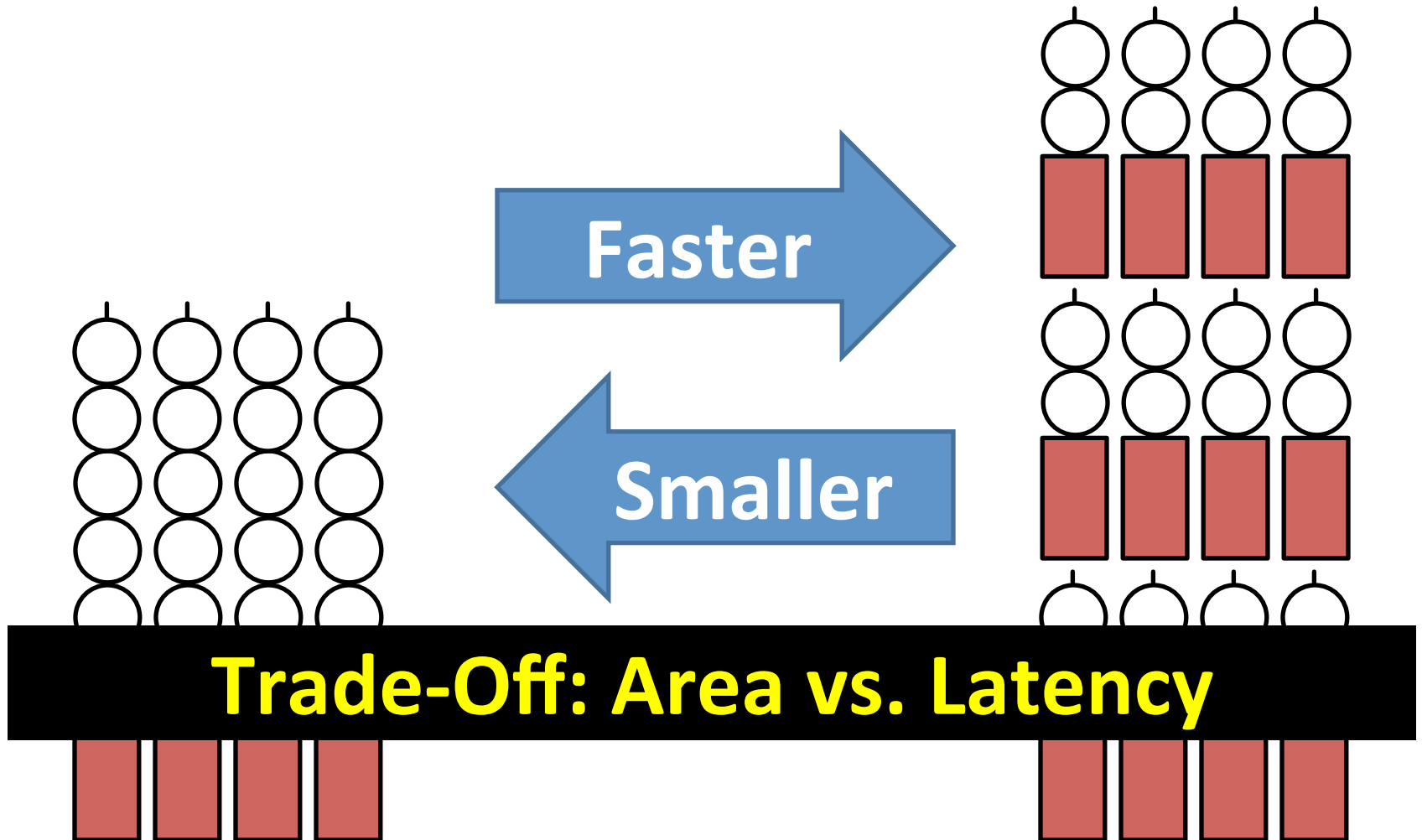


- Long bitline
 - Amortizes sense amplifier cost → Small area
 - Large bitline capacitance → High latency & power

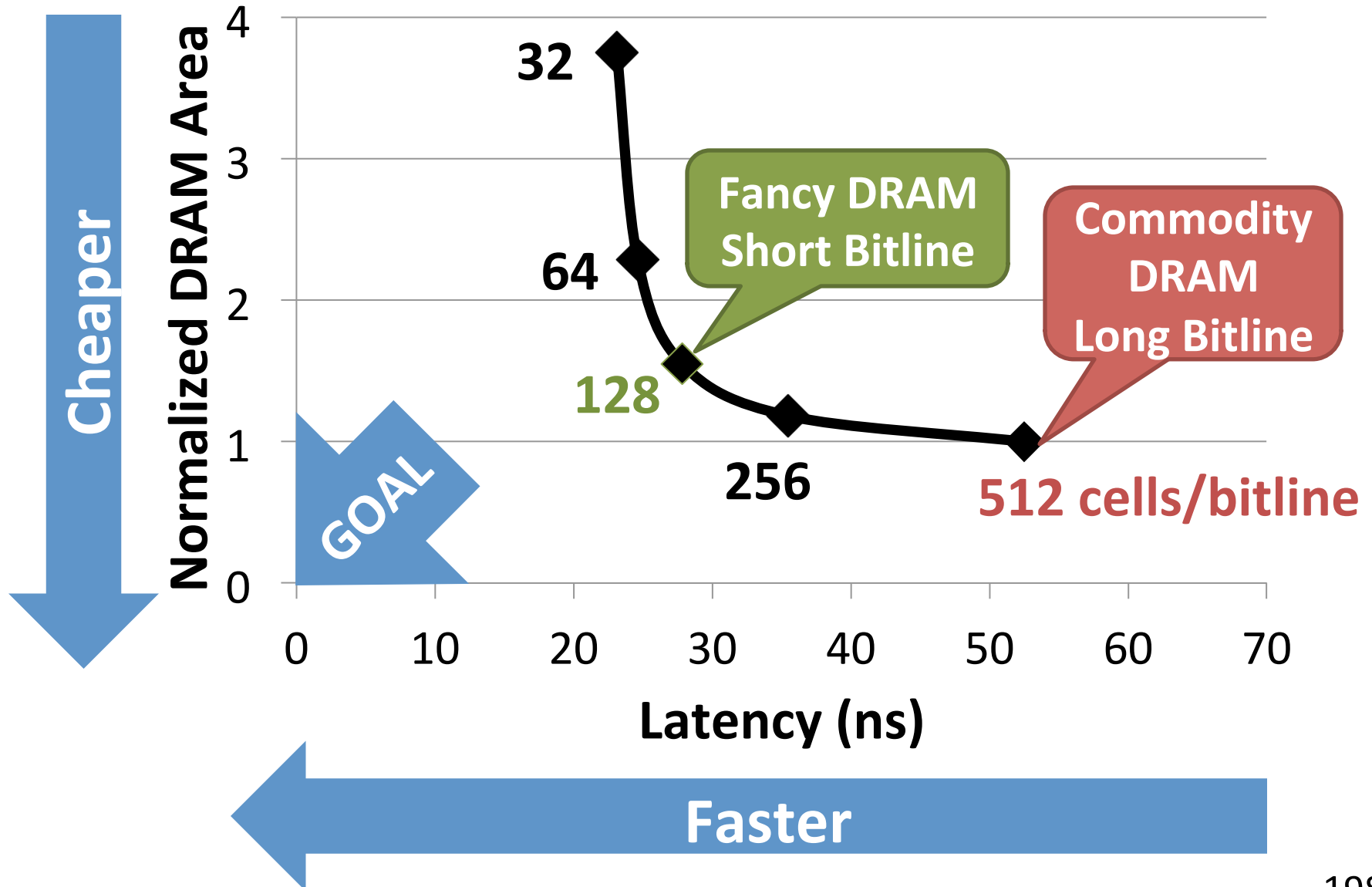
Trade-Off: Area (Die Size) vs. Latency

Long Bitline

Short Bitline



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

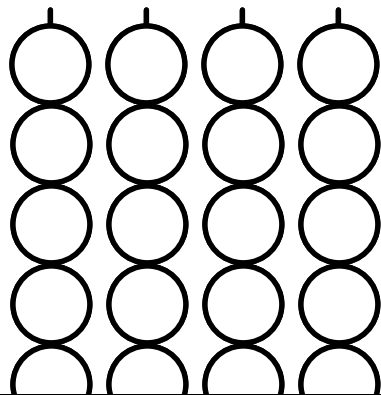


Approximating the Best of Both Worlds

Long Bitline

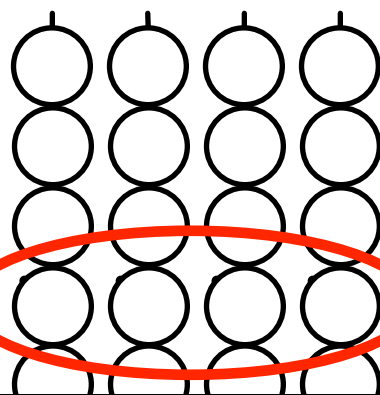
Small Area

~~High Latency~~



Need Isolation

Our Proposal

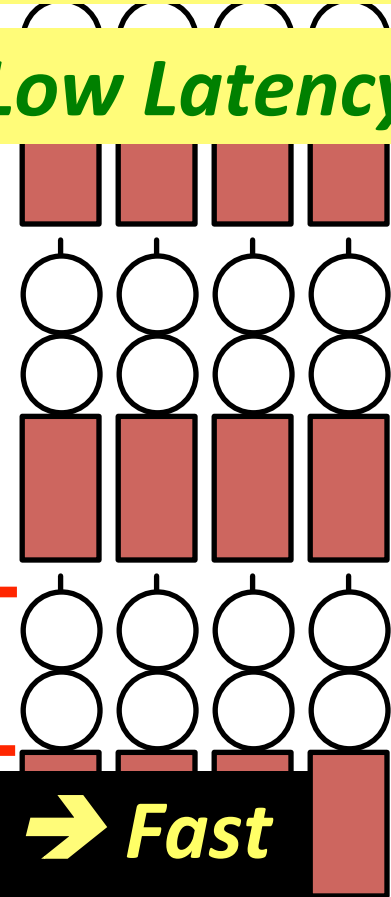


Add Isolation Transistors

Short Bitline

~~Large Area~~

Low Latency



tline → Fast

Approximating the Best of Both Worlds

Long Bitline Tiered-Latency DRAM **Short Bitline**

Small Area

Small Area

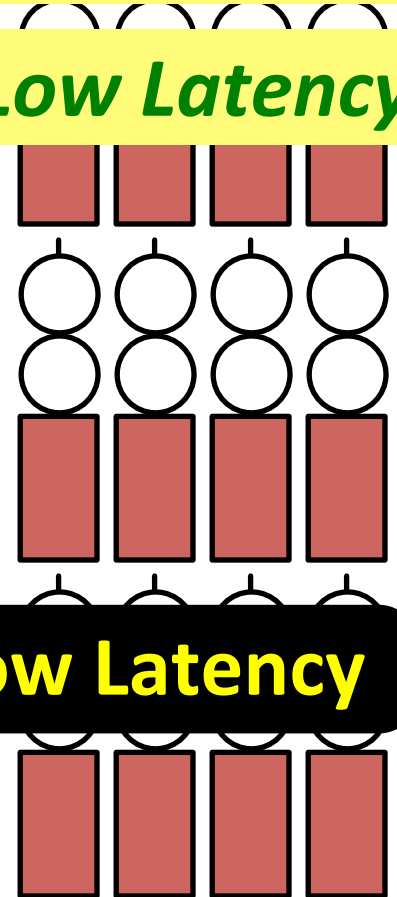
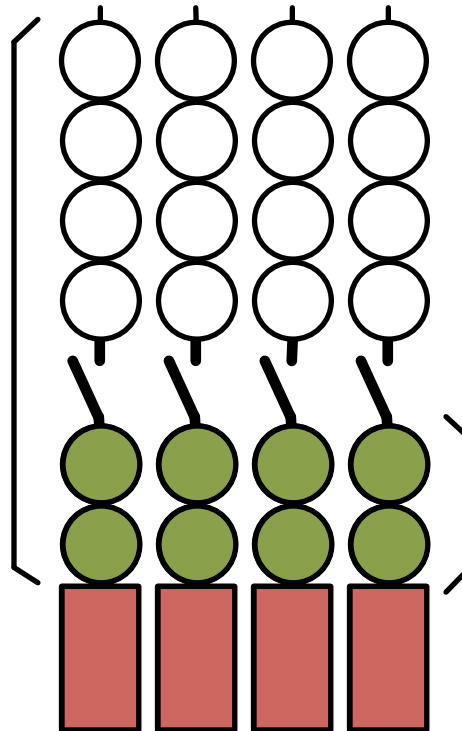
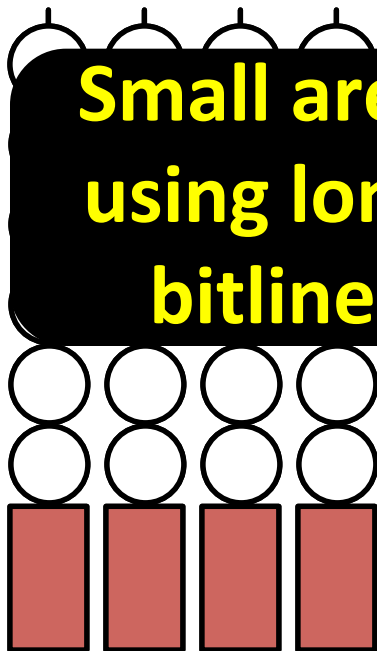
~~*Large Area*~~

~~*High Latency*~~

Low Latency

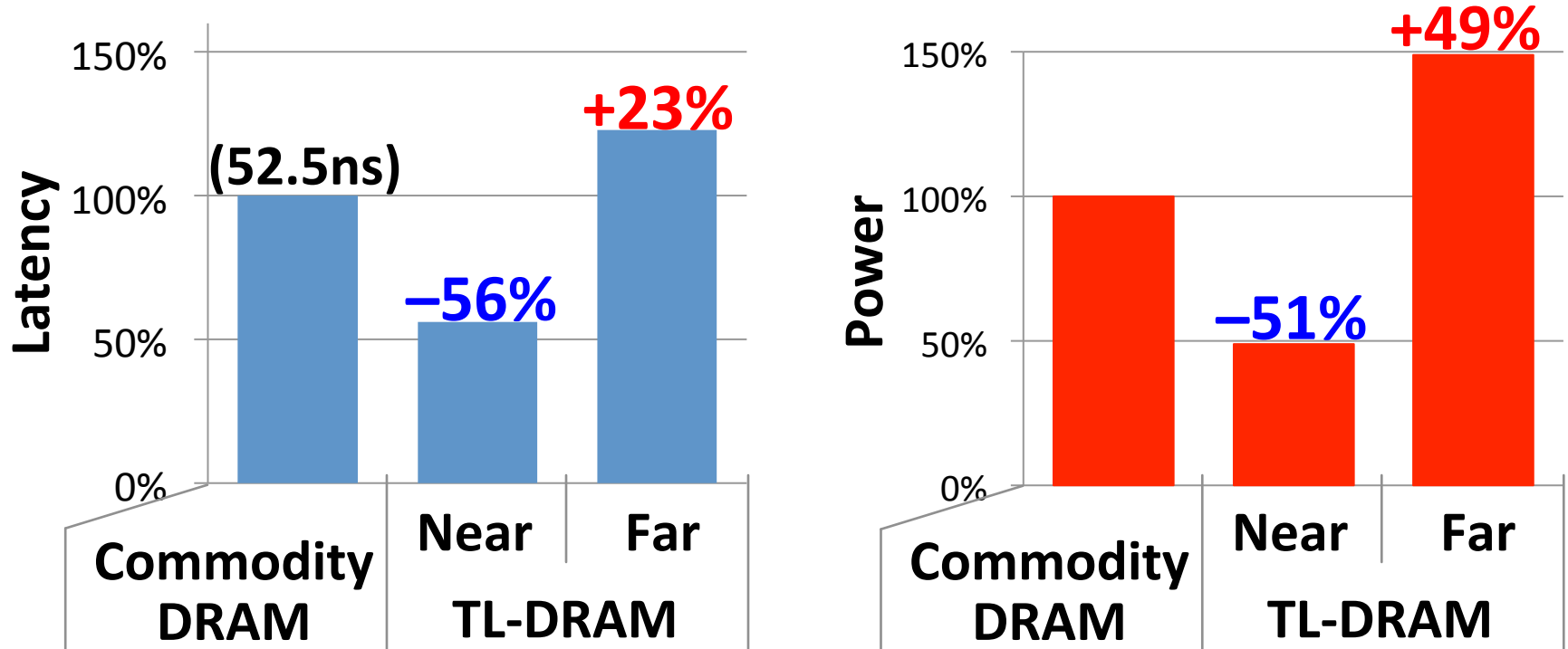
Low Latency

**Small area
using long
bitline**



Commodity DRAM vs. TL-DRAM [HPCA 2013]

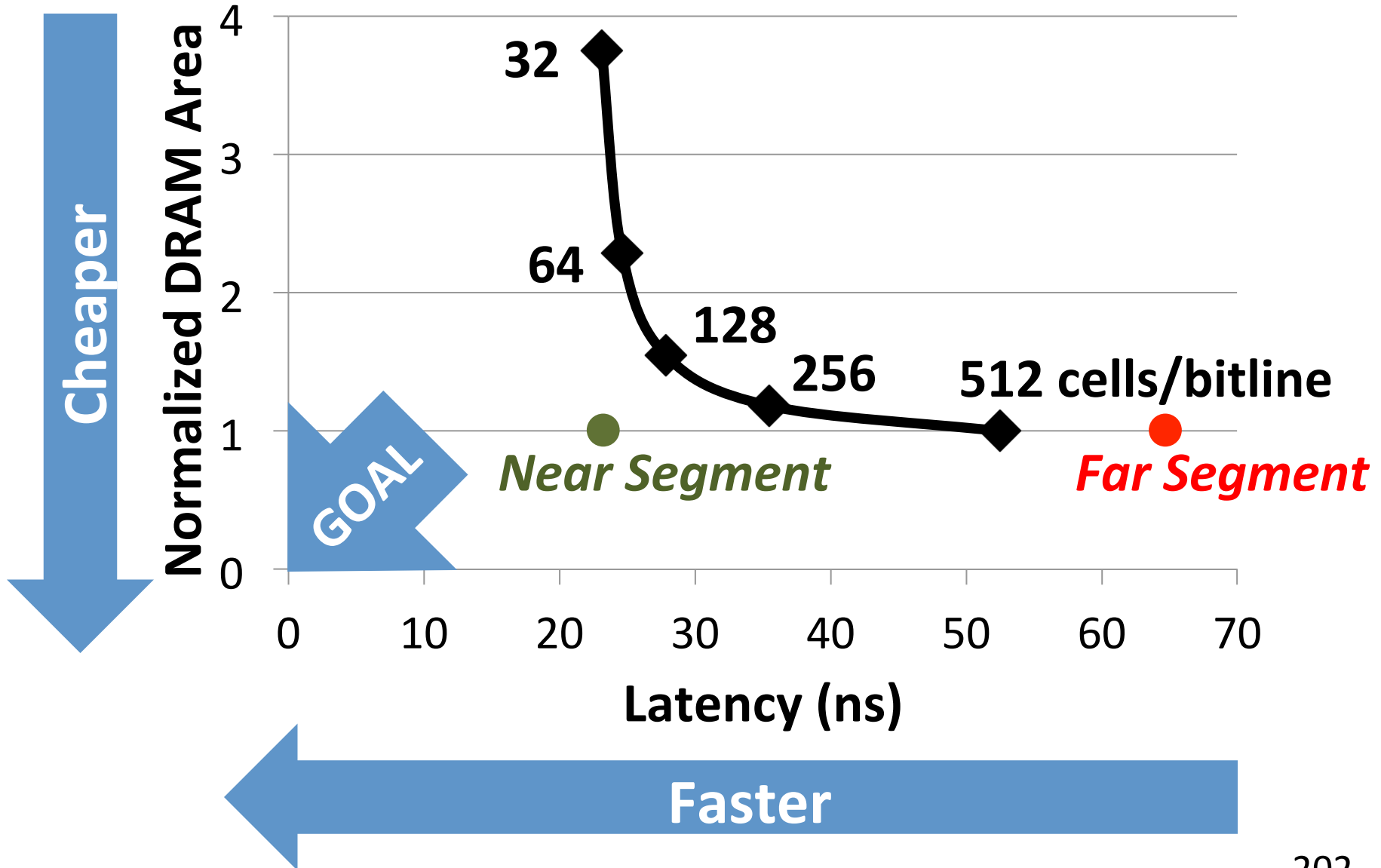
- DRAM Latency (tRC) • DRAM Power



- DRAM Area Overhead

~3%: mainly due to the isolation transistors

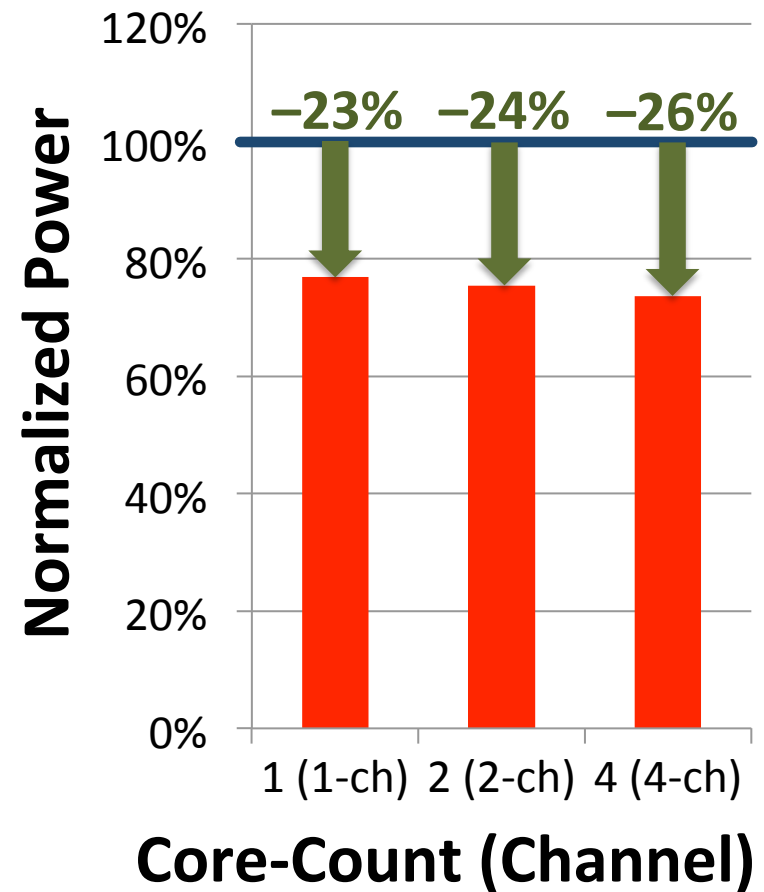
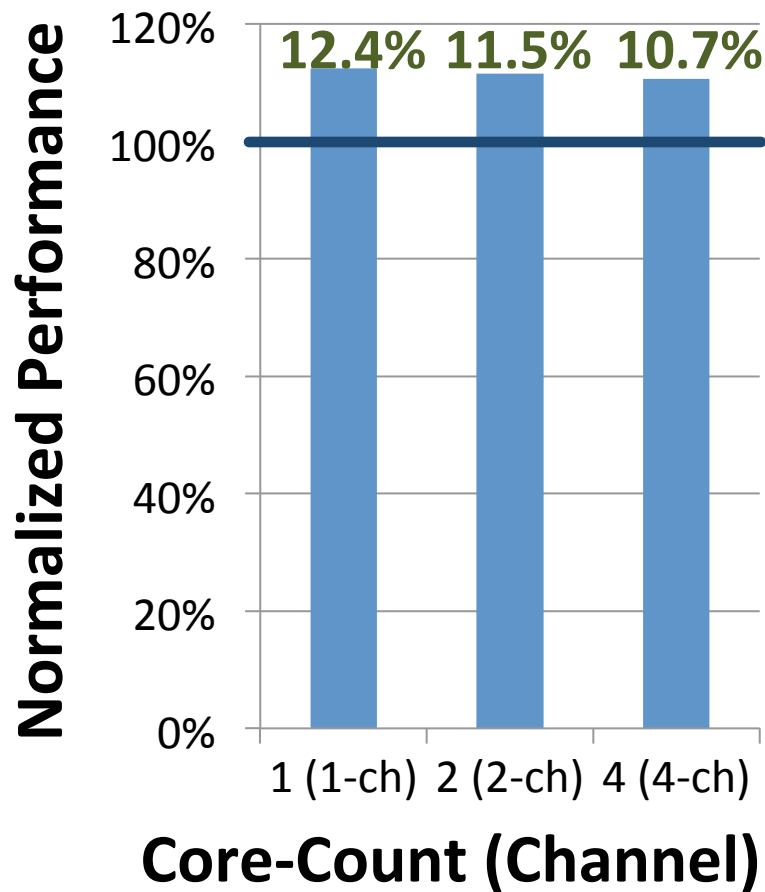
Trade-Off: Area (Die-Area) vs. Latency



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

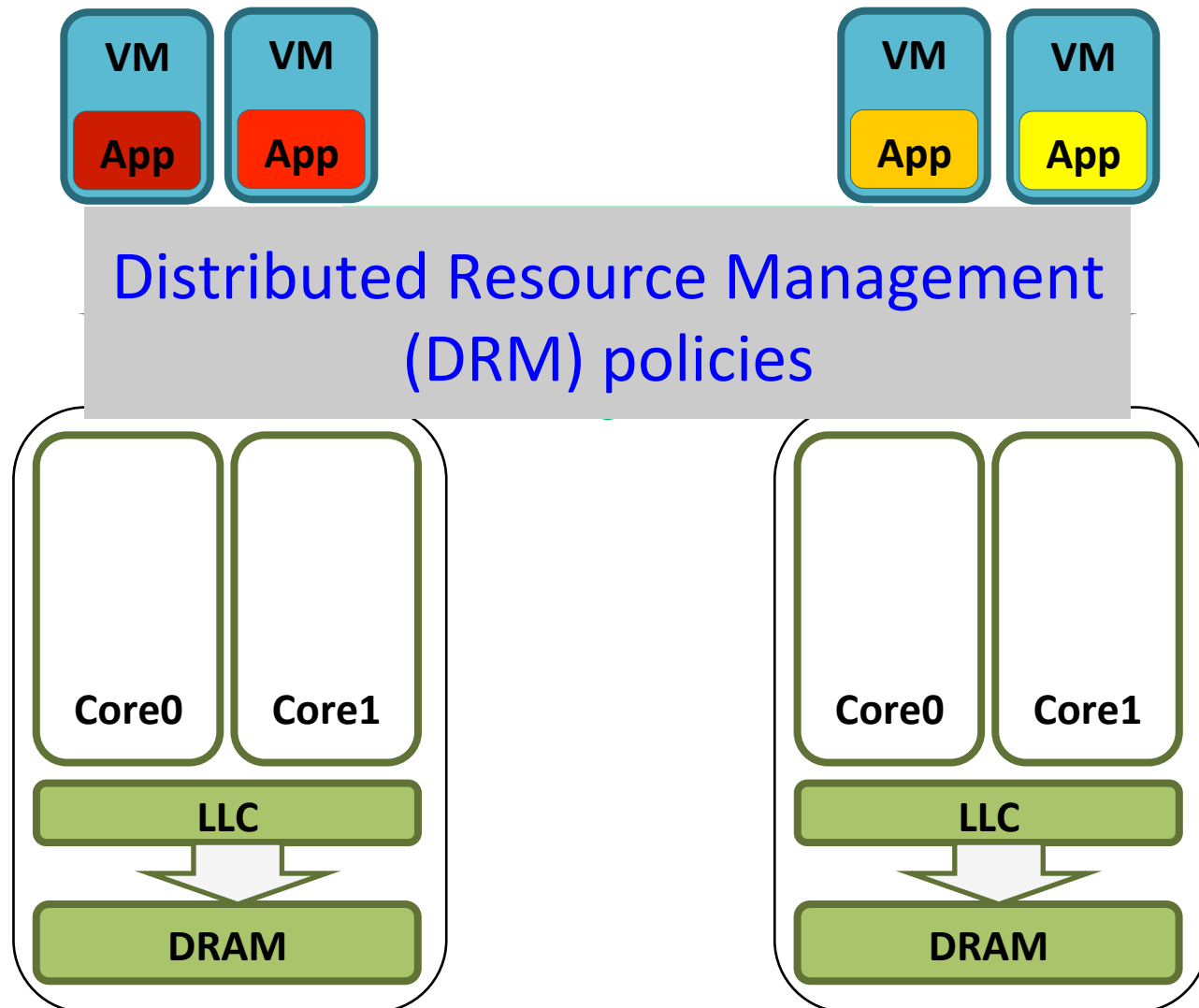
Performance & Power Consumption



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

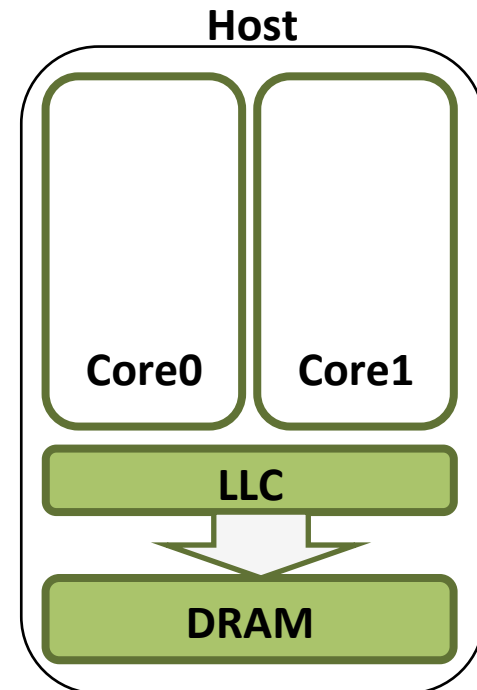
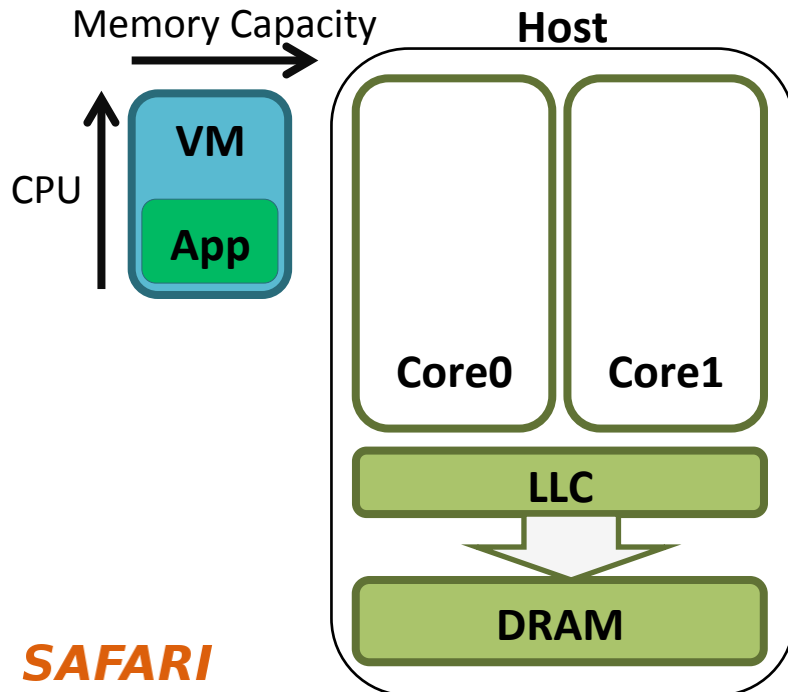
Architecture-Aware DRM

Virtualized Cluster



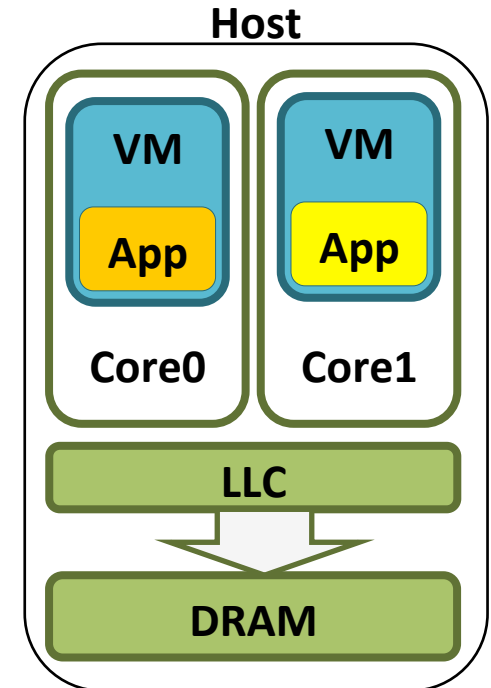
Conventional DRM Policies

Based on **operating-system-level metrics**
e.g., **CPU utilization**, **memory capacity demand**



Microarchitecture-level Interference

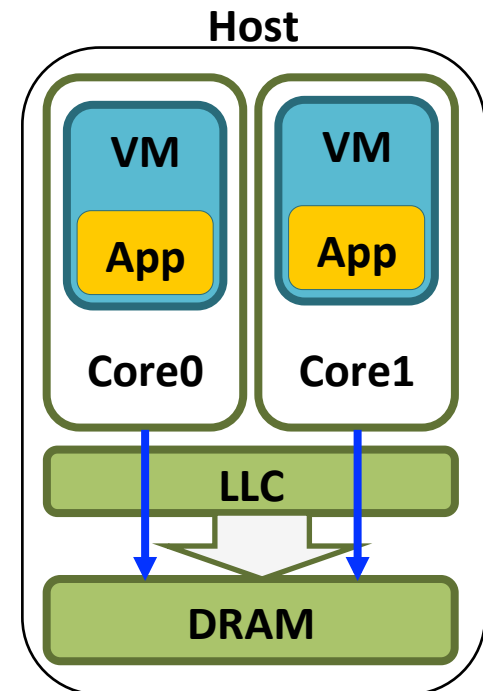
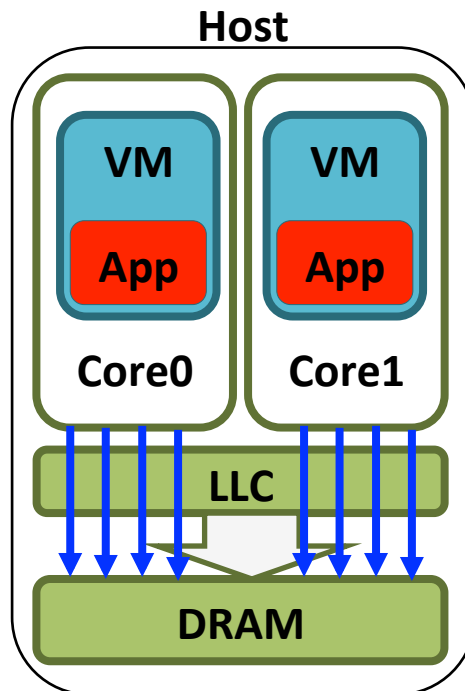
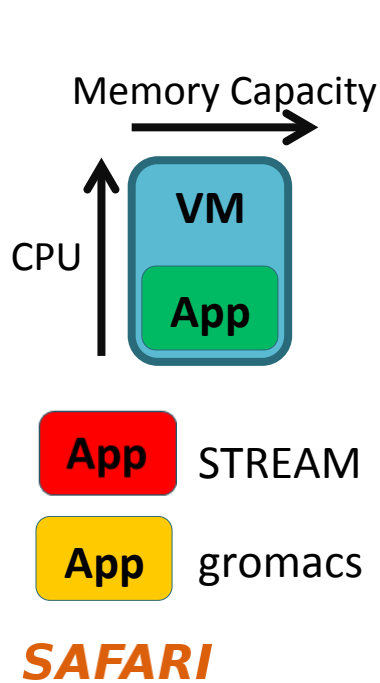
- VMs within a host compete for:
 - Shared cache capacity
 - Shared memory bandwidth



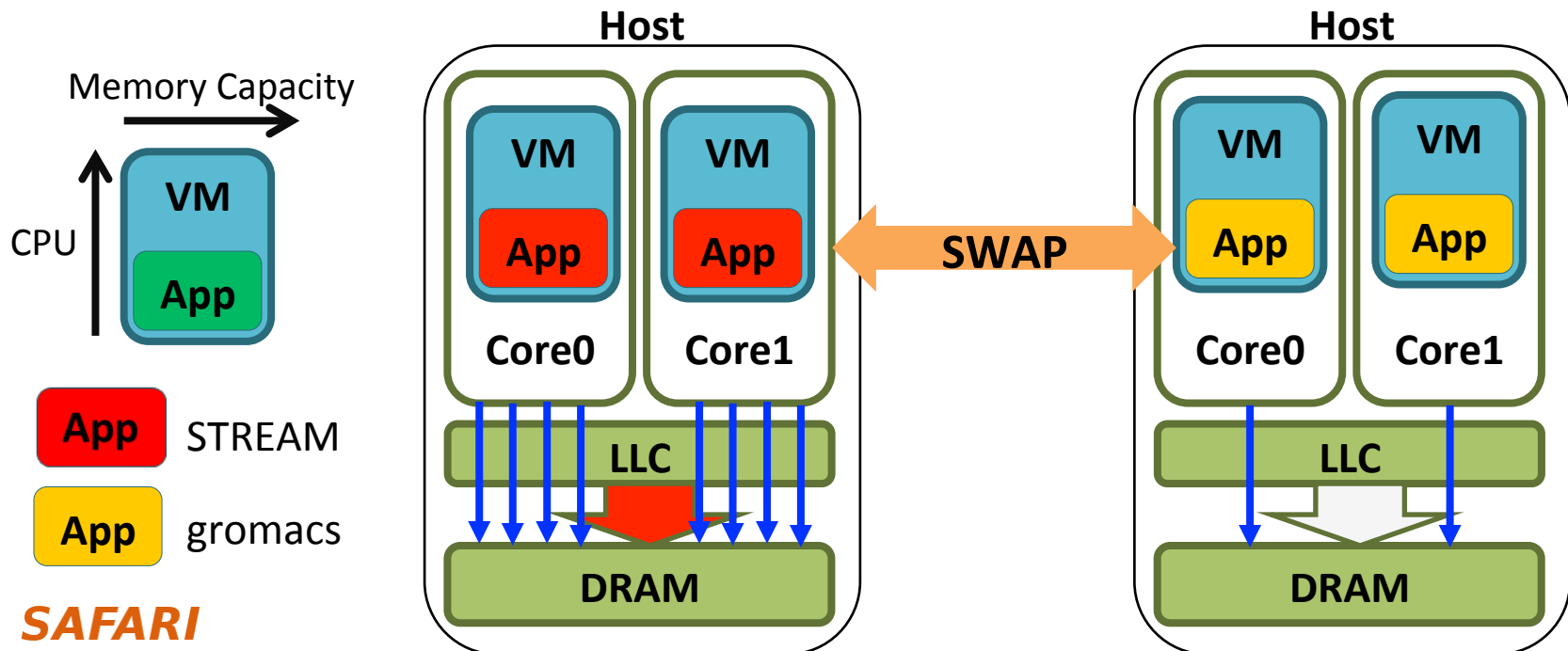
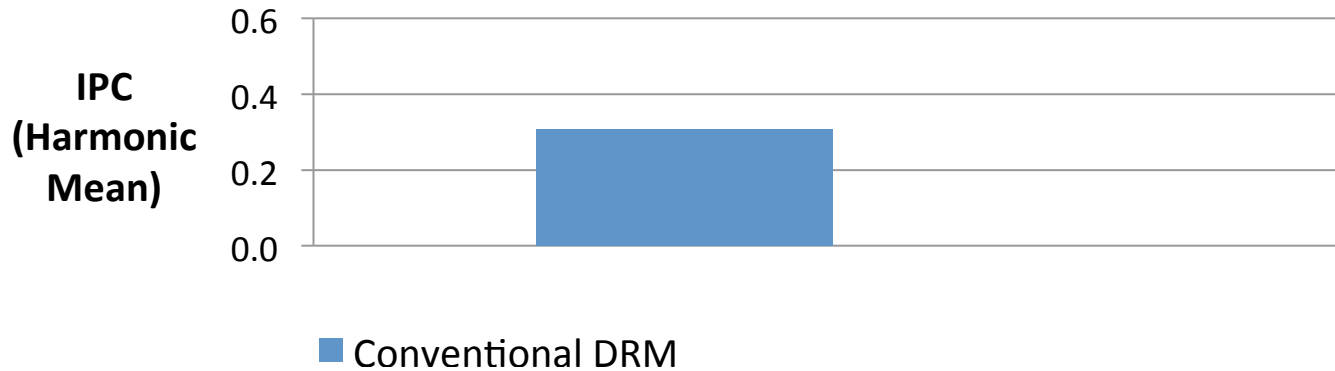
Can operating-system-level metrics capture the microarchitecture-level resource interference?

Microarchitecture Unawareness

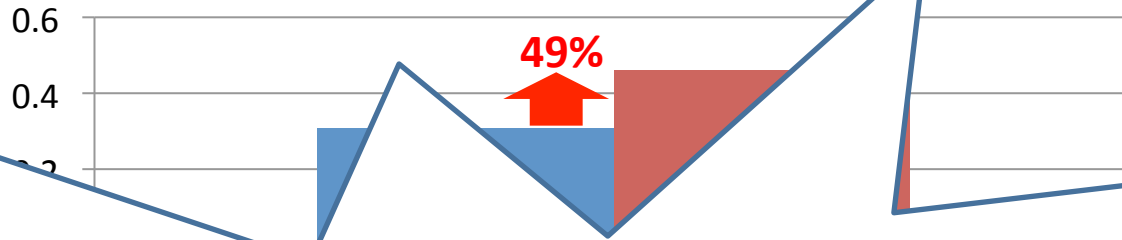
VM	Operating-system-level metrics		Microarchitecture-level metrics	
	CPU Utilization	Memory Capacity	LLC Hit Ratio	Memory Bandwidth
App	92%	369 MB	2%	2267 MB/s
App	93%	348 MB	98%	1 MB/s



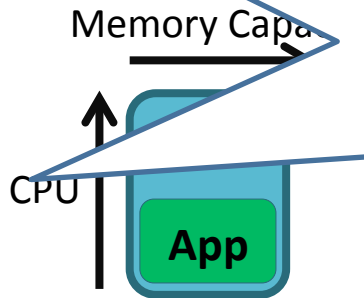
Impact on Performance



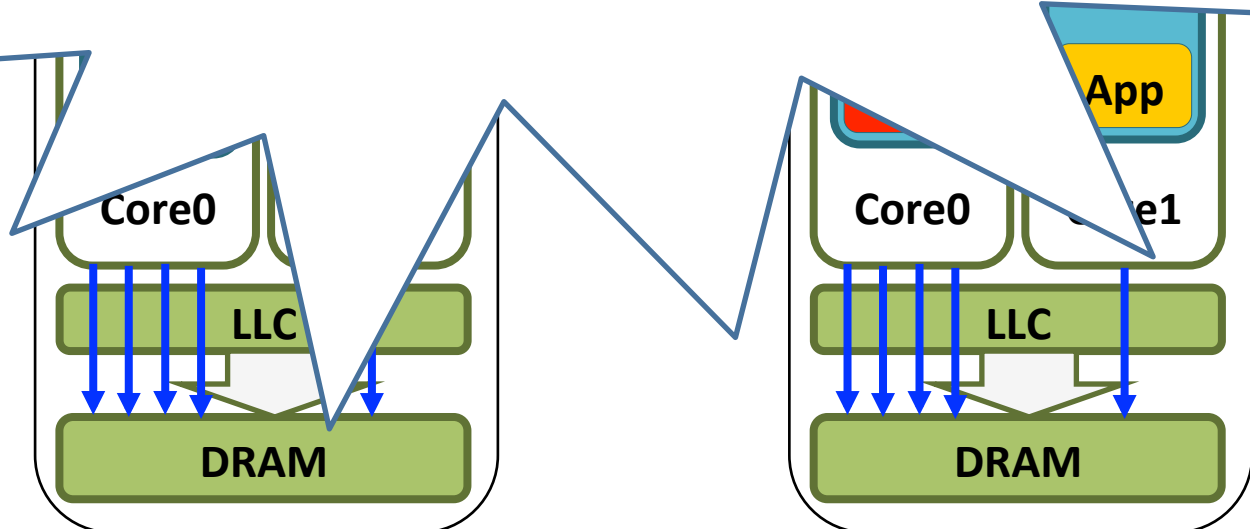
Impact on Performance



We need microarchitecture-level interference awareness in DRM!



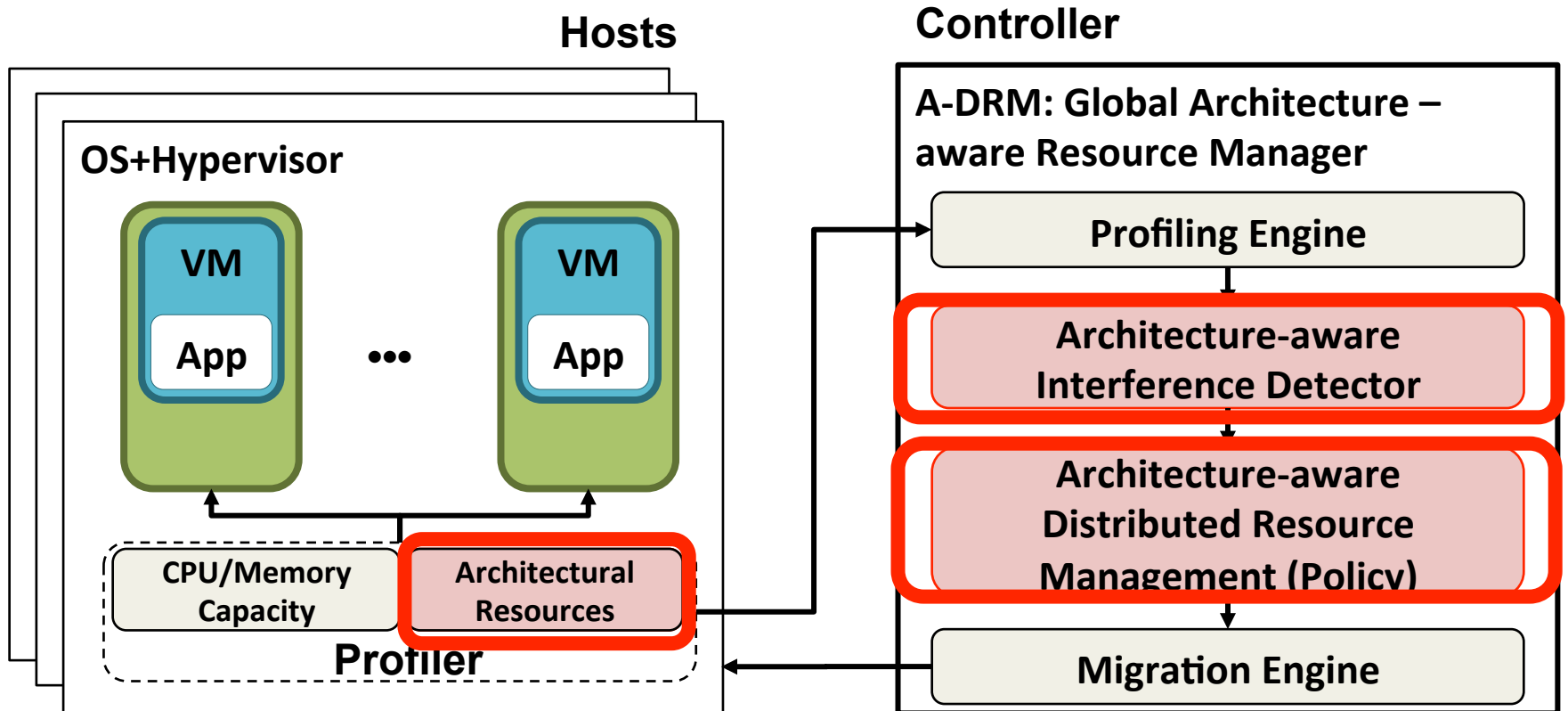
SAFARI



A-DRM: Architecture-aware DRM

- **Goal**: Take into account microarchitecture-level shared resource interference
 - Shared cache capacity
 - Shared memory bandwidth
- **Key Idea**:
 - Monitor and detect microarchitecture-level shared resource interference
 - Balance microarchitecture-level resource usage across cluster to minimize memory interference while maximizing system performance

A-DRM: Architecture-aware DRM



More on Architecture-Aware DRM

- Hui Wang, Canturk Isci, Lavanya Subramanian, Jongmoo Choi, Depei Qian, and Onur Mutlu,
"A-DRM: Architecture-aware Distributed Resource Management of Virtualized Clusters"
Proceedings of the
11th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE), Istanbul, Turkey, March 2015.
[[Slides \(pptx\)](#)] [[pdf](#)]

A-DRM: Architecture-aware Distributed Resource Management of Virtualized Clusters

Hui Wang^{†*}, Canturk Isci[‡], Lavanya Subramanian*, Jongmoo Choi^{‡*}, Depei Qian[†], Onur Mutlu*

[†]Beihang University, [‡]IBM Thomas J. Watson Research Center, *Carnegie Mellon University, [‡]Dankook University
{hui.wang, depei.qian}@buaa.edu.cn, canturk@us.ibm.com, {lsubrama, onur}@cmu.edu, choijm@dankook.ac.kr

Other Research

Current Research Focus Areas

Research Focus: *Computer architecture, HW/SW, bioinformatics*

- **Memory, memory, memory, storage, interconnects**
- *Parallel architectures, heterogeneous architectures, GP-GPUs*
- *System/architecture interaction, new execution models*
- *Energy efficiency, fault tolerance, hardware security*
- *Genome sequence analysis & assembly algorithms and architectures*

