

Some New Ideas in Memory System Design for Data-Intensive Computing

Onur Mutlu

onur@cmu.edu

September 4, 2014

ISTC-CC Retreat

Carnegie Mellon

Some New Ideas (This Year)

■ Specialization

- ❑ Heterogeneous Reliability Memory [DSN 2014]
- ❑ Heterogeneous Block Architecture [ICCD 2014]

■ Persistent Memory

- ❑ Loose Ordering Consistency for Persistent Memory [ICCD 2014]
- ❑ Transparent Consistency for Persistent/Hybrid Memory [in progress]

■ Memory Reliability/Security

- ❑ Row Hammer Problem in DRAM [ISCA 2014]
- ❑ Neighbor-Cell Assisted Error Correction in Flash [SIGMETRICS 2014]
- ❑ Error Mitigation for Intermittent DRAM Failures [SIGMETRICS 2014]

■ Memory Performance

- ❑ The Dirty-Block Index [ISCA 2014]
- ❑ DRAM Refresh-Access Parallelization [HPCA 2014]
- ❑ The Blacklisting Memory Scheduler [ICCD 2014]
- ❑ Exploiting Read-Write Disparity in Caches [HPCA 2014]

Memory Reliability Trends

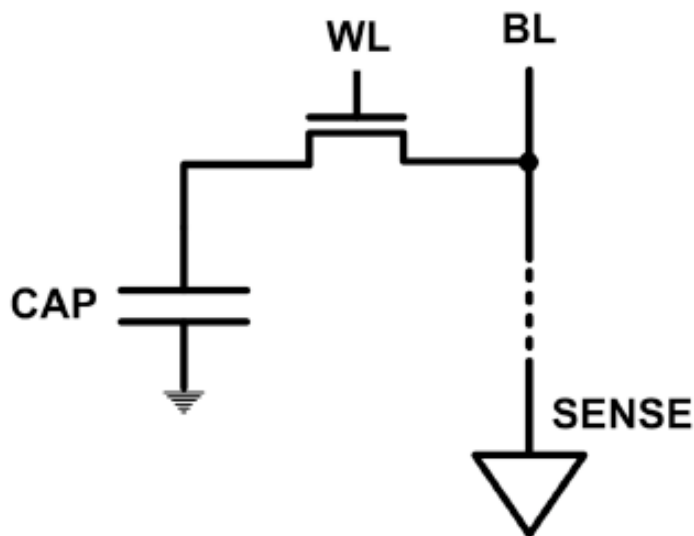
- **Memory is becoming less reliable** as its density increases with technology scaling
 - Reduced retention times
 - Increased vulnerability to disturbance
 - New error types (e.g., due to inter-cell interference)
 - ...
- **Maintaining reliability is expensive** in terms of
 - Energy
 - Performance
 - Cost (TCO)

DRAM Scaling

- DRAM technology scaling has provided many benefits
 - Higher capacity
 - Lower cost
 - Reasonable energy
- DRAM scaling is becoming difficult due to reduced reliability
 - ITRS projects **DRAM will not scale easily below X nm**

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

The DRAM Scaling Problem

- DRAM scaling has become a real problem the system should be concerned about
 - And, maybe embrace

Flipping Bits in Memory Without Accessing Them

DRAM Disturbance Errors

Yoongu Kim

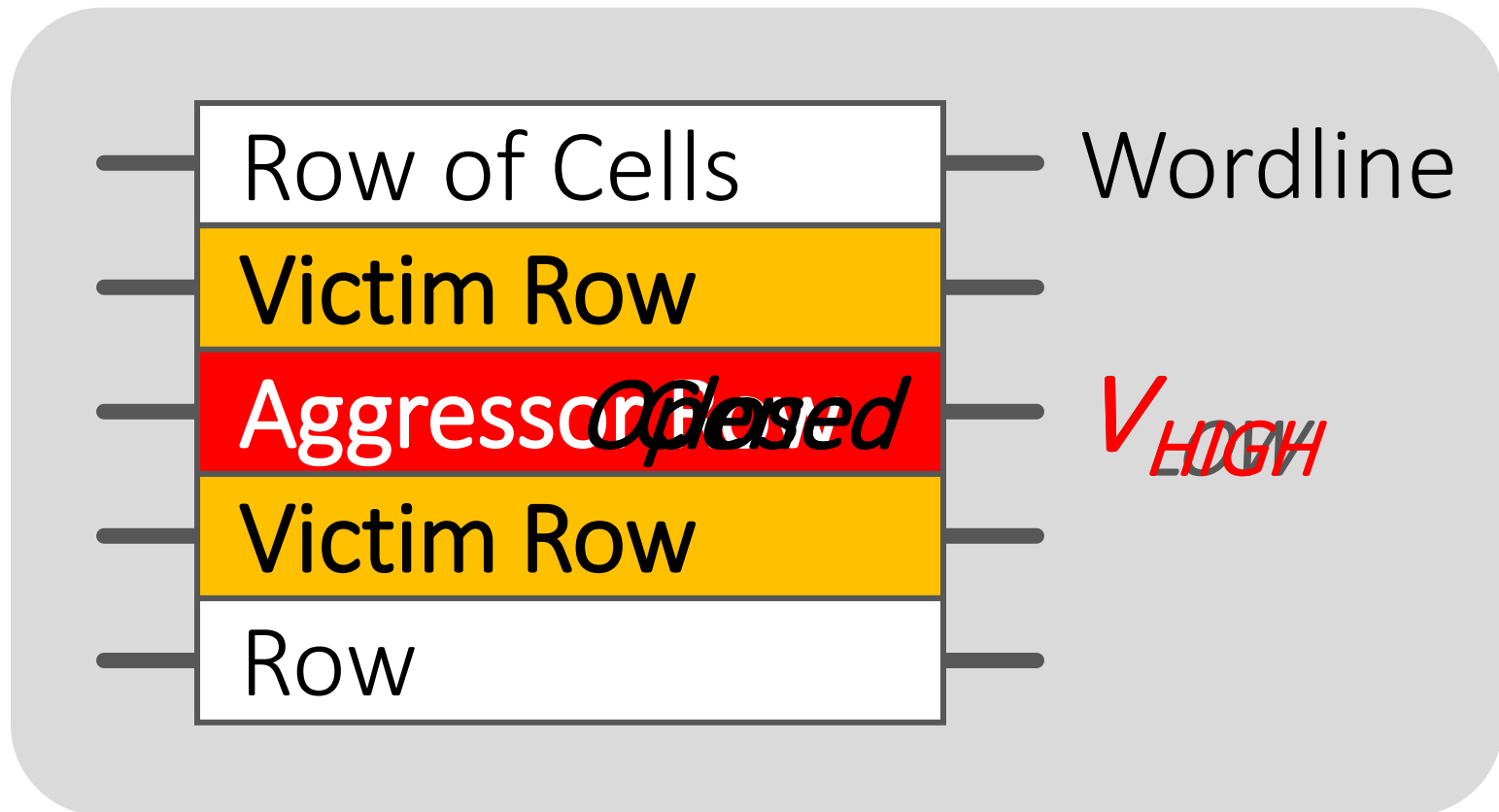
Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee,
Donghyuk Lee, Chris Wilkerson, Konrad Lai, Onur Mutlu

Carnegie Mellon

SAFARI



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]

Quick Summary

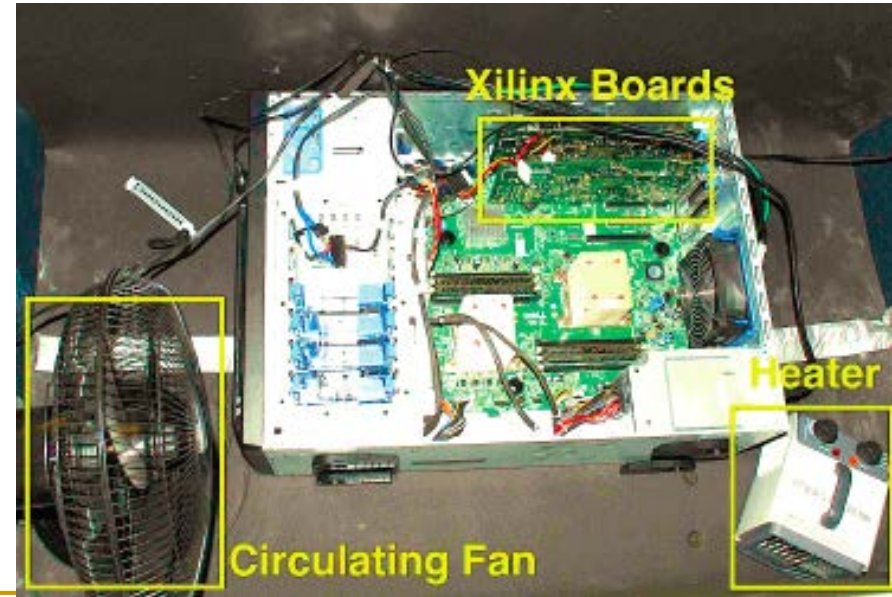
- We expose the *existence* and *prevalence* of disturbance errors in DRAM chips of today
 - 110 of 129 modules are vulnerable
 - Affects modules of 2010 vintage or later
- We characterize the *cause* and *symptoms*
 - Toggling a row accelerates charge leakage in adjacent rows: *row-to-row coupling*
- We prevent errors using a *system-level* approach
 - Each time a row is closed, we refresh the charge stored in its adjacent rows with a low probability

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)

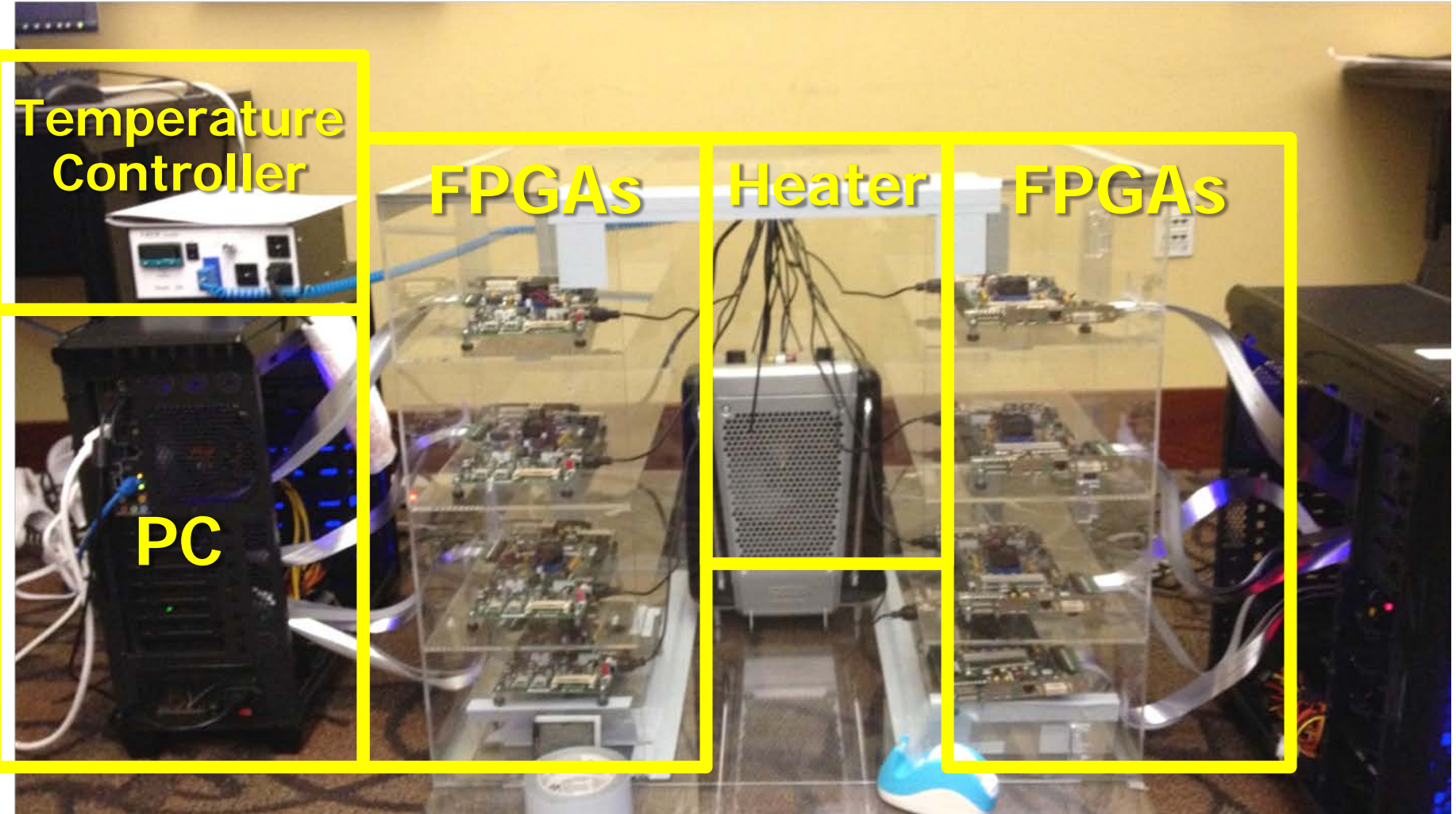
Temperature
Controller

FPGAs

Heater

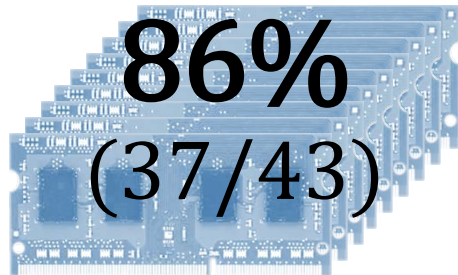
FPGAs

PC

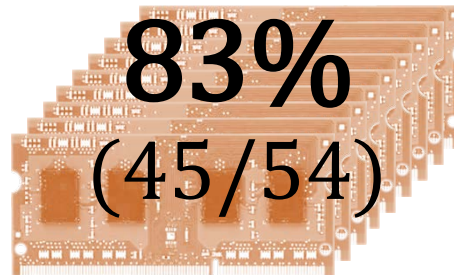


Most DRAM Modules Are at Risk

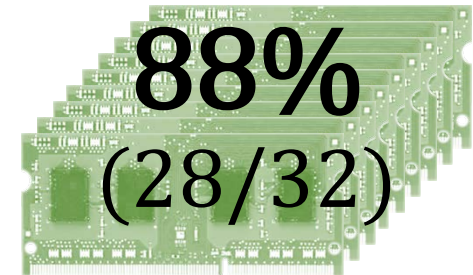
A company



B company



C company

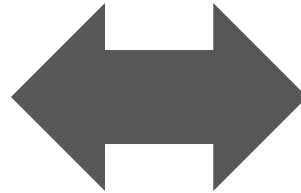
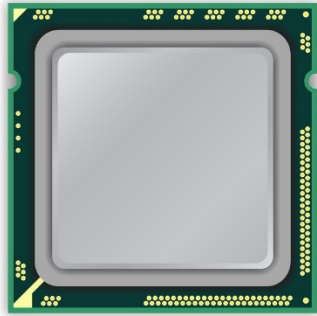


Up to
 1.0×10^7
errors

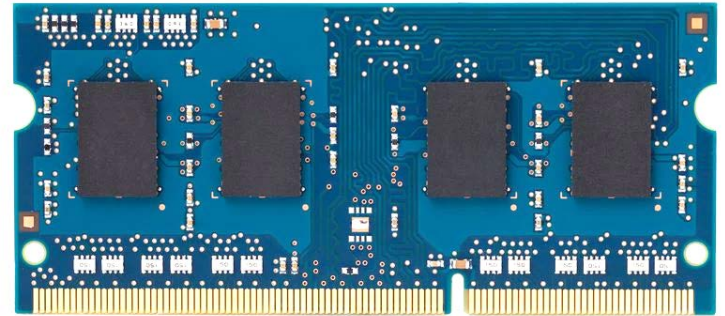
Up to
 2.7×10^6
errors

Up to
 3.3×10^5
errors

x86 CPU



DRAM Module

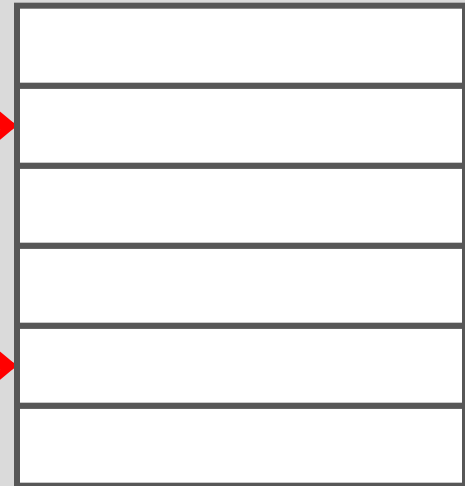


loop:

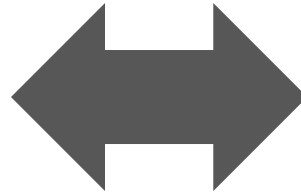
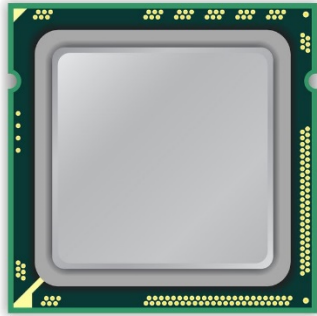
```
mov (X), %eax  
mov (Y), %ebx  
clflush (X)  
clflush (Y)  
mfence  
jmp loop
```

X →

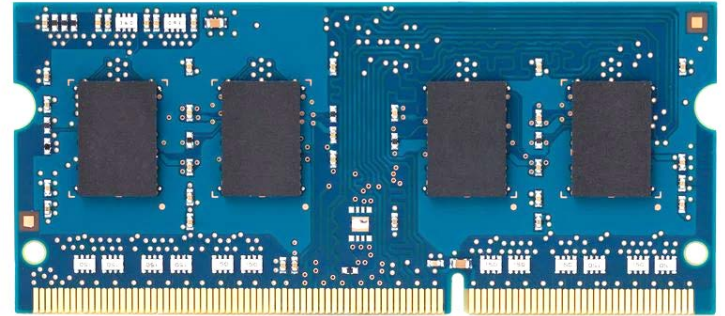
Y →



x86 CPU

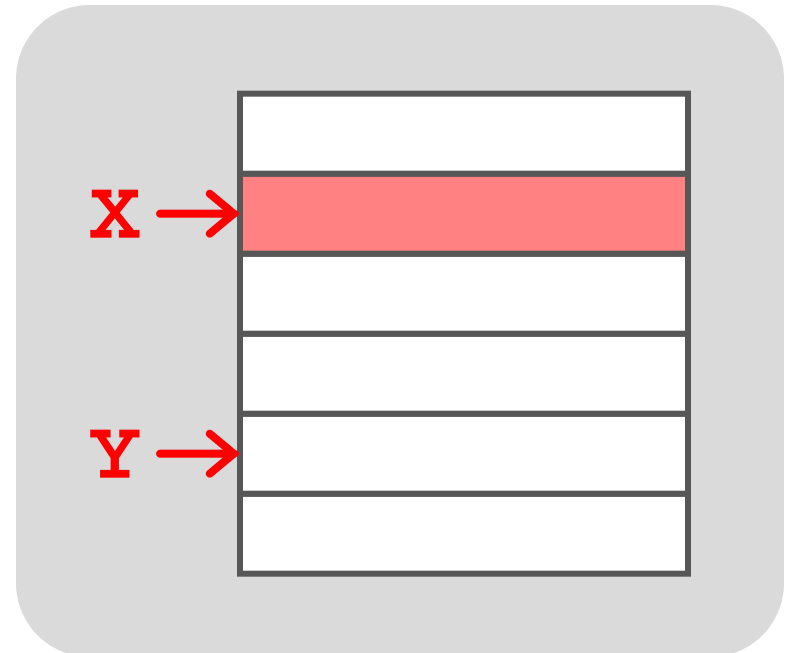


DRAM Module

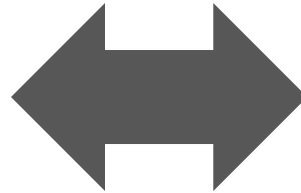
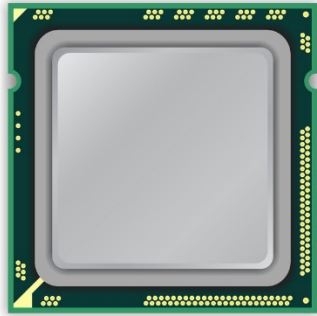


loop:

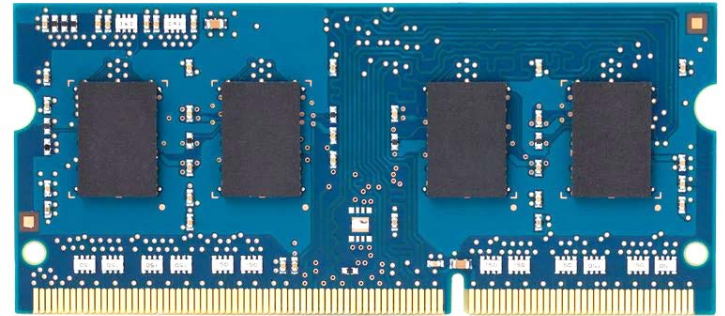
```
mov (X), %eax  
mov (Y), %ebx  
clflush (X)  
clflush (Y)  
mfence  
jmp loop
```



x86 CPU

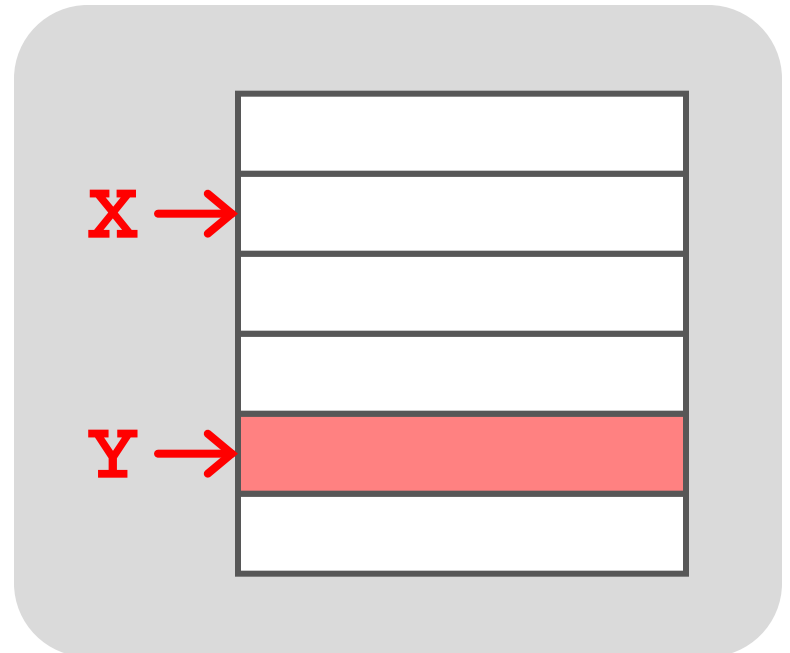


DRAM Module

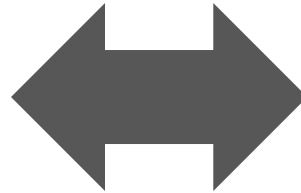
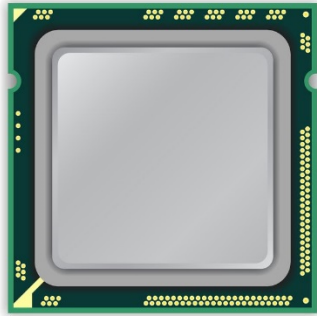


loop:

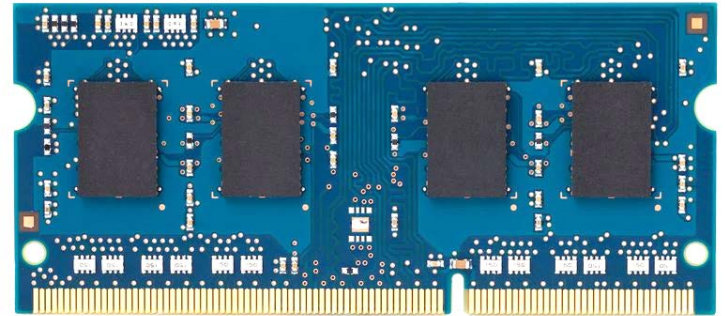
```
mov (X), %eax  
mov (Y), %ebx  
clflush (X)  
clflush (Y)  
mfence  
jmp loop
```



x86 CPU

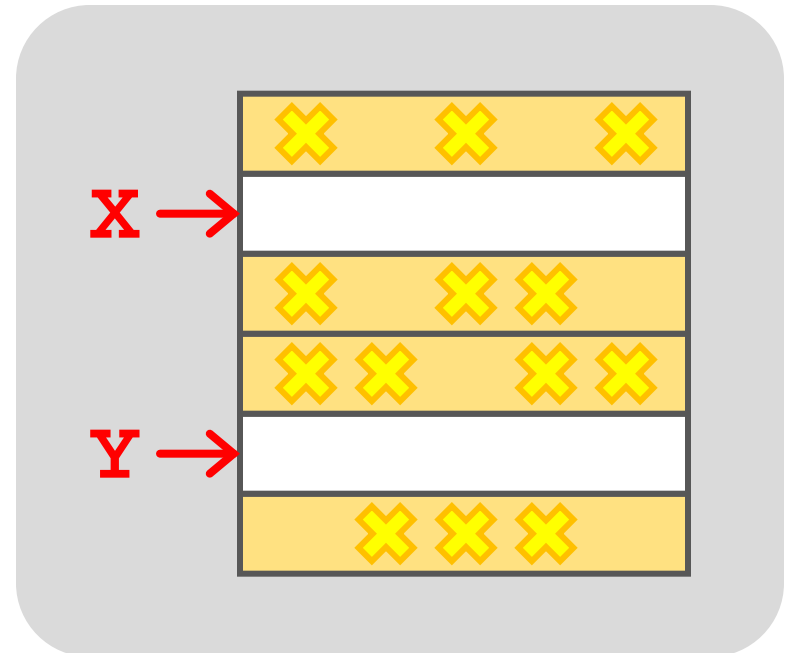


DRAM Module



loop:

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



Observed Errors in Real Systems

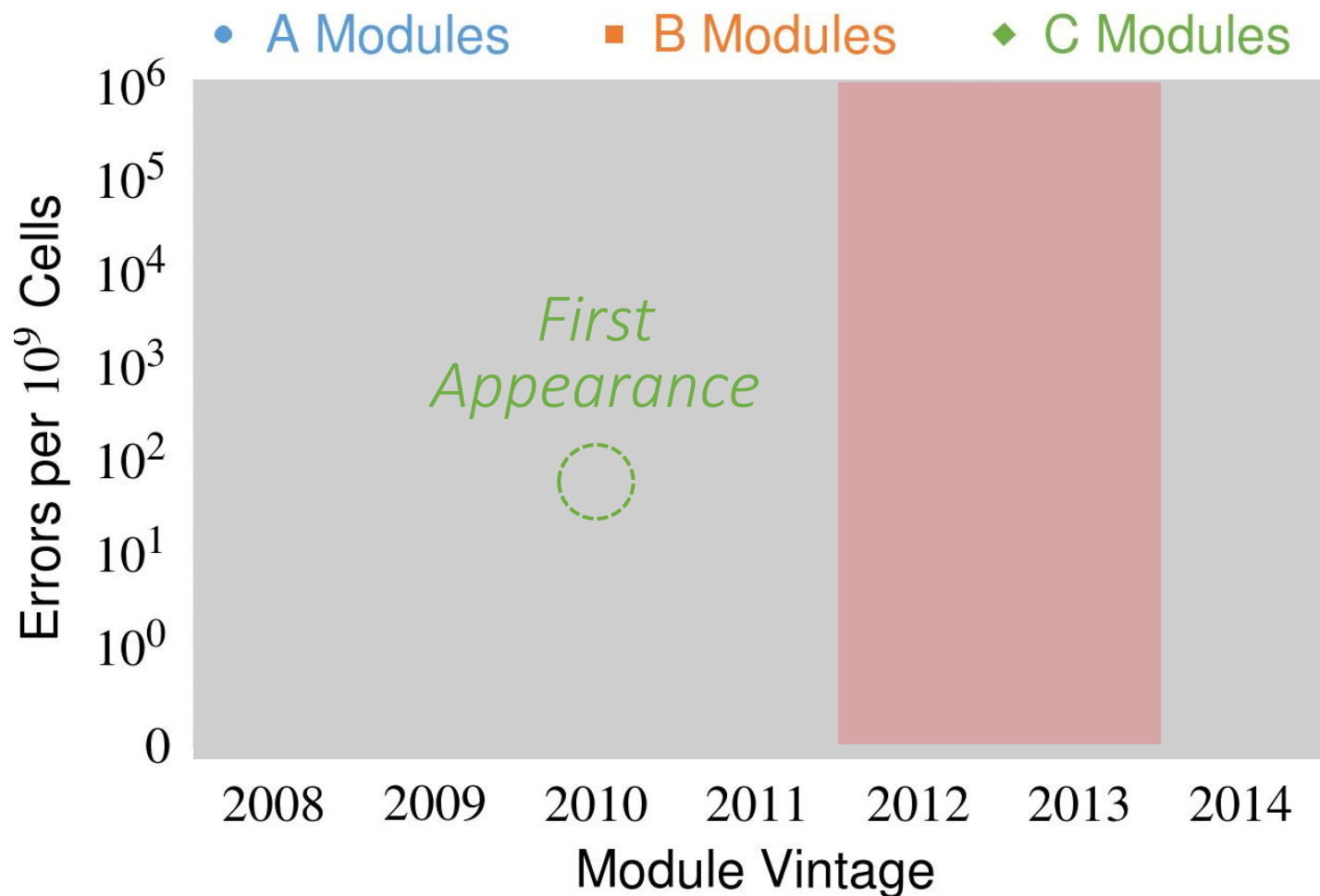
CPU Architecture	Errors	Access-Rate
Intel Haswell (2013)	22.9K	12.3M/sec
Intel Ivy Bridge (2012)	20.7K	11.7M/sec
Intel Sandy Bridge (2011)	16.1K	11.6M/sec
AMD Piledriver (2012)	59	6.1M/sec

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

Security Implications

- *Breach of memory protection*
 - OS page (4KB) fits inside DRAM row (8KB)
 - Adjacent DRAM row → Different OS page
- *Vulnerability: disturbance attack*
 - By accessing its own page, a program could corrupt pages belonging to another program
- *We constructed a proof-of-concept*
 - Using only user-level instructions

Errors vs. Vintage



All modules from 2012–2013 are vulnerable

Characterization Results

1. Most Modules Are at Risk
2. Errors vs. Vintage
3. Error = Charge Loss
4. Adjacency: Aggressor & Victim
5. Sensitivity Studies
6. Other Results in Paper

Several Potential Solutions

- Make better DRAM chips

Cost

- Refresh frequently

Power, Performance

- Sophisticated ECC

Cost, Power

- Access counters

Cost, Power, Complexity

Our Solution

- **PARA: *Probabilistic Adjacent Row Activation***
- **Key Idea**
 - After closing a row, we activate (i.e., refresh) one of its neighbors with a low probability: $p = 0.005$
- **Reliability Guarantee**
 - When $p=0.005$, errors in one year: 9.4×10^{-14}
 - By adjusting the value of p , we can provide an arbitrarily strong protection against errors

More Information ...

- Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and [Onur Mutlu](#),
"Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors"
Proceedings of the [41st International Symposium on Computer Architecture \(ISCA\)](#), Minneapolis, MN, June 2014. [Slides \(pptx\)](#) [\(pdf\)](#)
[Lightning Session Slides \(pptx\)](#) [\(pdf\)](#) [Source Code and Data](#)

Some New Ideas (This Year)

■ Specialization

- Heterogeneous Reliability Memory [DSN 2014]
- Heterogeneous Block Architecture [ICCD 2014]

■ Persistent Memory

- Loose Ordering Consistency for Persistent Memory [ICCD 2014]
- Transparent Consistency for Persistent/Hybrid Memory [in progress]

■ Memory Reliability/Security

- Row Hammer Problem in DRAM [ISCA 2014]
- Neighbor-Cell Assisted Error Correction in Flash [SIGMETRICS 2014]
- Error Mitigation for Intermittent DRAM Failures [SIGMETRICS 2014]

■ Memory Performance

- The Dirty-Block Index [ISCA 2014]
- DRAM Refresh-Access Parallelization [HPCA 2014]
- The Blacklisting Memory Scheduler [ICCD 2014]
- Exploiting Read-Write Disparity in Caches [HPCA 2014]

Characterizing Application Memory Error Vulnerability to Optimize Datacenter Cost via **Heterogeneous-Reliability Memory**

Yixin Luo, Sriram Govindan, Bikash Sharma,
Mark Santaniello, Justin Meza, Aman Kansal, Jie Liu,
Badriddine Khessib, Kushagra Vaid, Onur Mutlu

SAFARI

Carnegie Mellon

 **Microsoft**

Executive Summary

- Problem: *Reliable* memory hardware increases *cost*
- Our Goal: Reduce datacenter *cost*; meet *availability* target
- Observation: *Data-intensive applications' data exhibit a diverse spectrum of tolerance to memory errors*
 - Across applications and within an application
 - We characterized 3 modern data-intensive applications
- Our Proposal: *Heterogeneous-reliability memory (HRM)*
 - Store error-tolerant data in less-reliable lower-cost memory
 - Store error-vulnerable data in more-reliable memory
- Major results:
 - Reduce server hardware *cost* by **4.7 %**
 - Achieve single server *availability* target of **99.90 %**

Outline

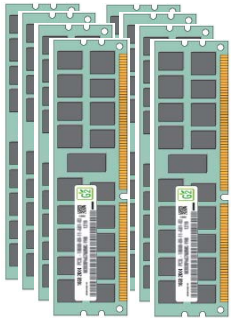
- Motivation
- Characterizing application memory error tolerance
- Key observations
 - Observation 1: Memory error tolerance varies across applications and within an application
 - Observation 2: Data can be recovered by software
- Heterogeneous-Reliability Memory (HRM)
- Evaluation

Outline

- Motivation
- Characterizing application memory error tolerance
- Key observations
 - Observation 1: Memory error tolerance varies across applications and within an application
 - Observation 2: Data can be recovered by software
- Heterogeneous-Reliability Memory (HRM)
- Evaluation

Server Memory Cost is High

- *Server hardware cost dominates datacenter Total Cost of Ownership (TCO) [Barroso '09]*
- *As server memory capacity grows, memory cost becomes the most important component of server hardware costs [Kozyrakis '10]*



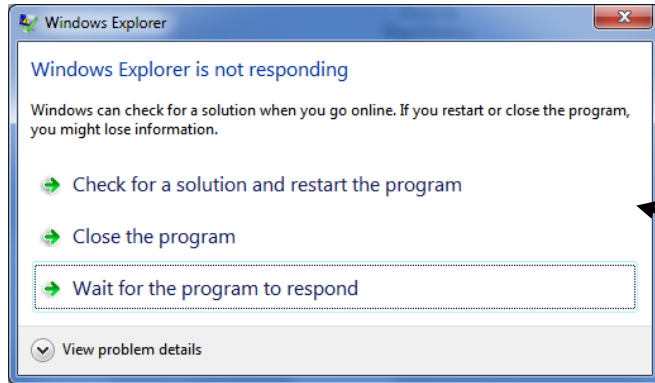
128GB Memory cost
~\$140(per 16GB)×8
= ~\$1120 *



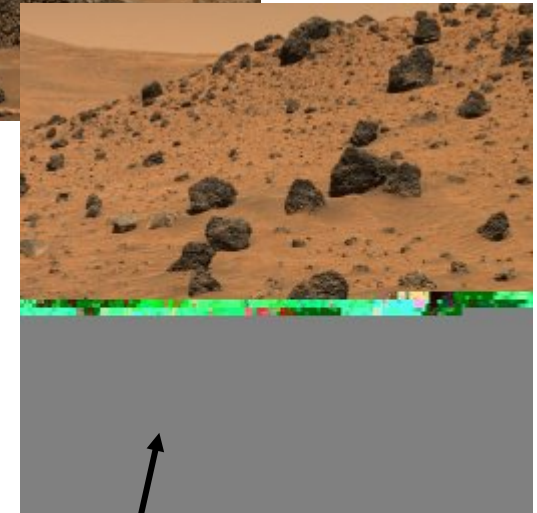
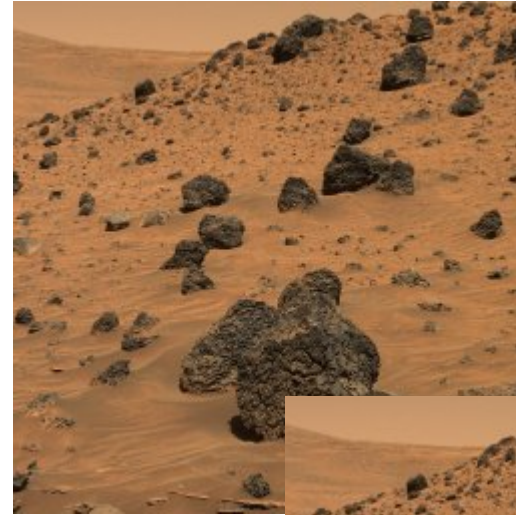
2 CPUs cost
~\$500(per CPU)×2
= ~\$1000 *

* Numbers in the year of 2014

Memory Reliability is Important

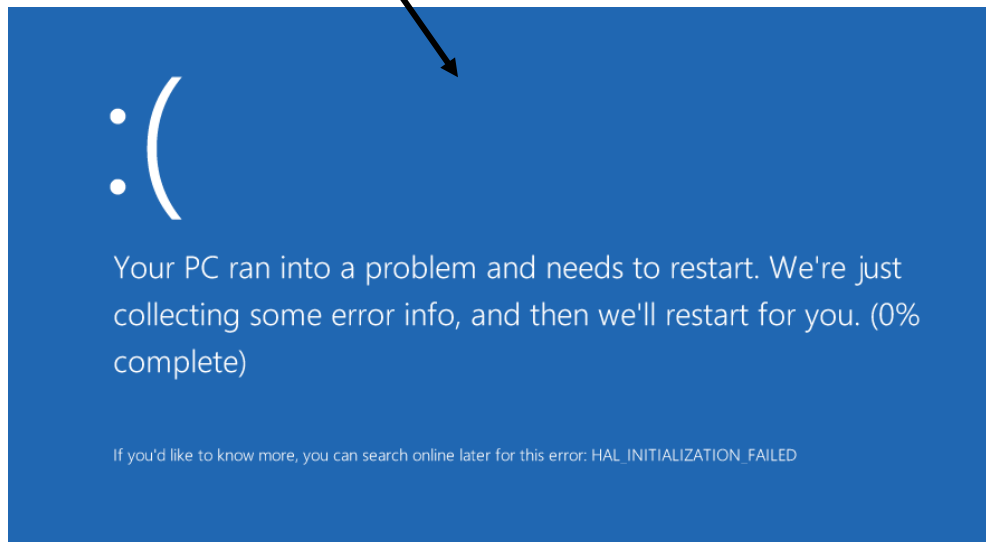


*System/app
hang or
slowdown*



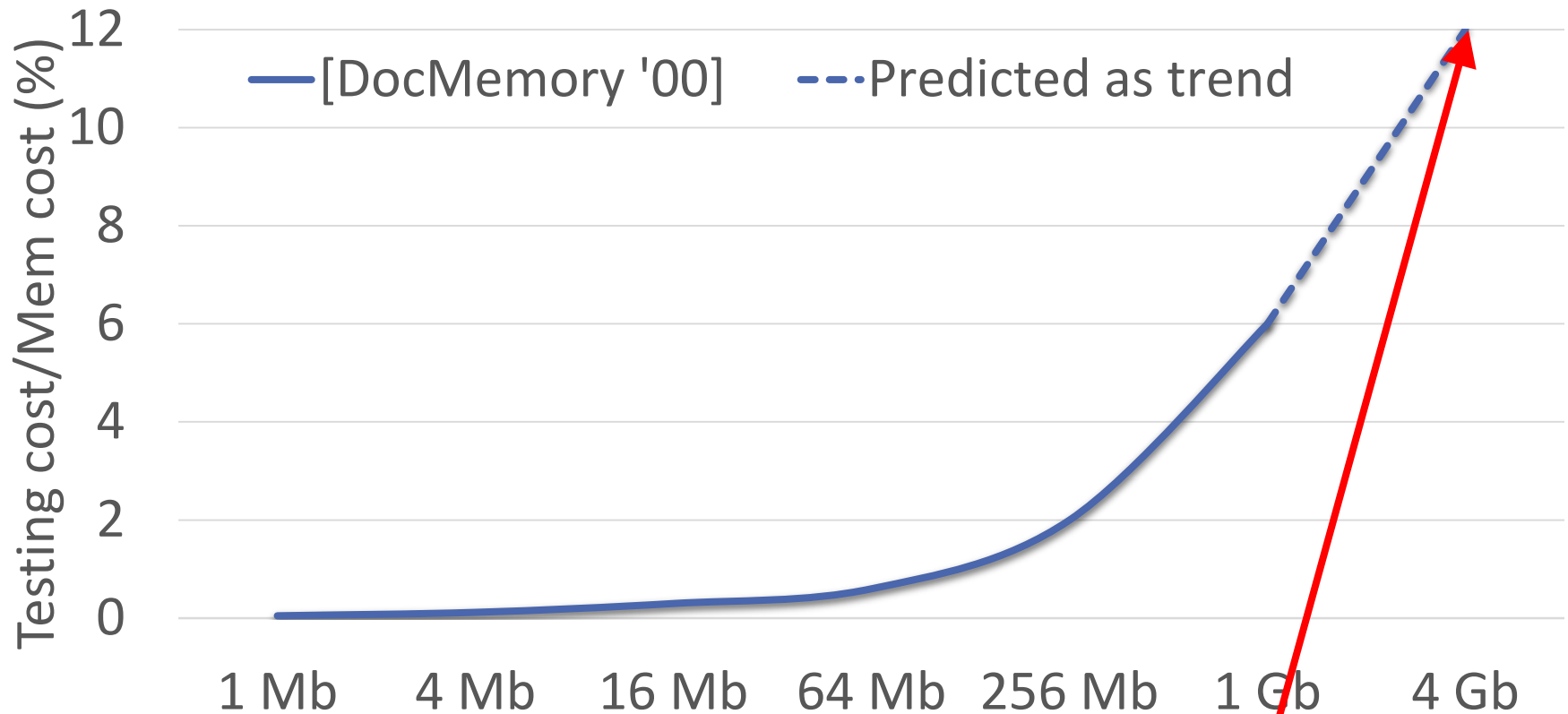
*Silent data corruption or
incorrect app output*

System/app crash



Existing Error Mitigation Techniques (I)

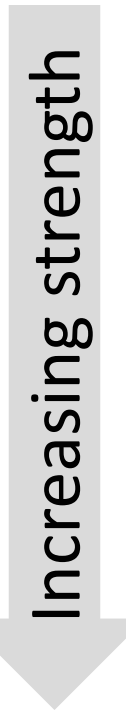
- *Quality assurance tests increase manufacturing cost*



Memory testing cost can be a significant fraction of memory cost as memory capacity grows

Existing Error Mitigation Techniques (II)

- *Error detection and correction increases system cost*



Technique	Detection	Correction	Added capacity	Added logic
NoECC	N/A	N/A	0.00%	No
Parity	1 bit	N/A	1.56%	Low
SEC-DED	2 bit	1 bit	12.5%	Low
Chipkill	2 chip	1 chip	12.5%	High

Stronger error protection techniques have higher cost

Shortcomings of Existing Approaches

- *Uniformly improve memory reliability*
 - Observation 1: Memory error tolerance varies across applications and with an application
- *Rely only on hardware-level techniques*
 - Observation 2: Once a memory error is detected, most corrupted data can be recovered by software

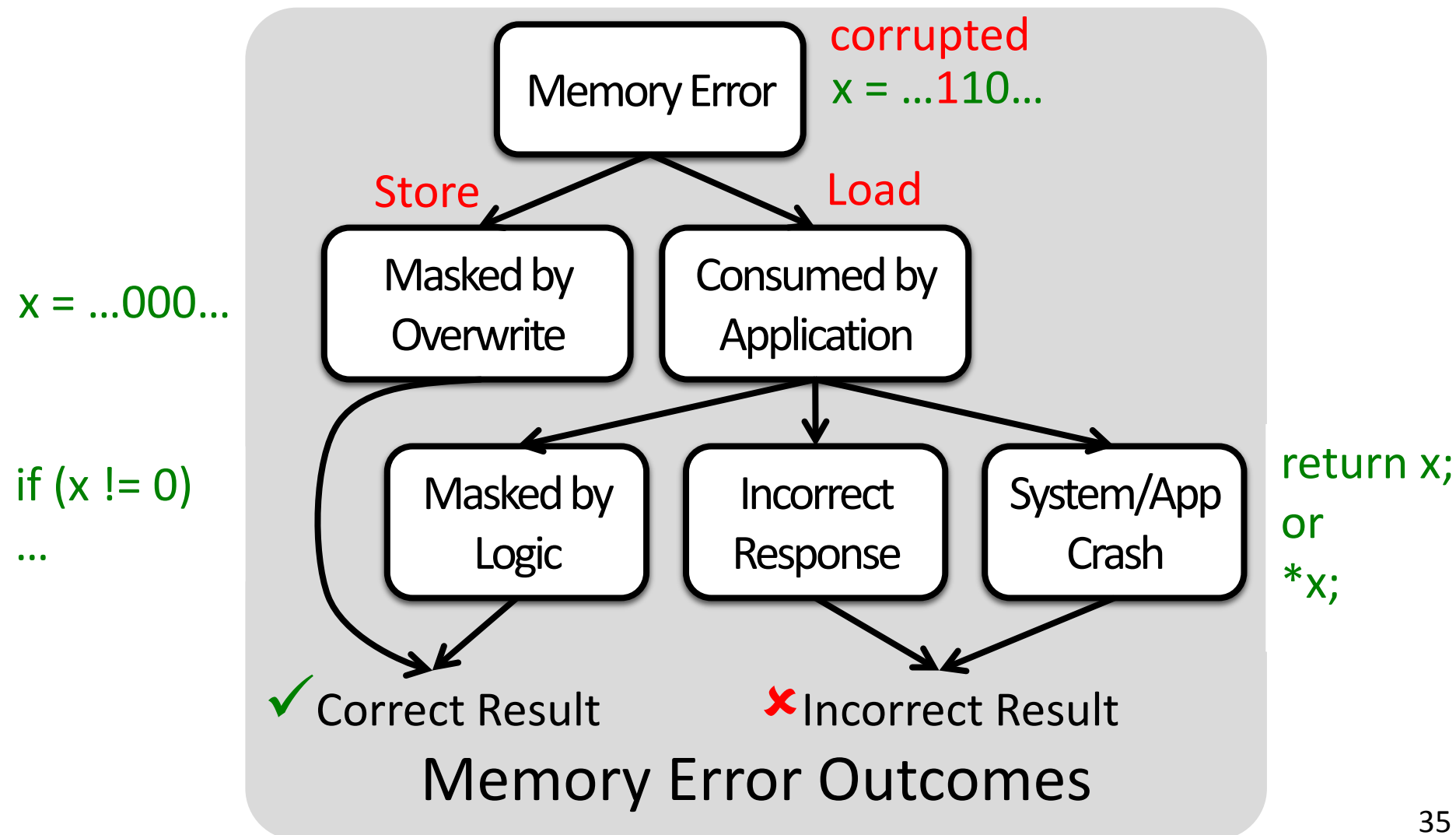
Goal: Design a new cost-efficient memory system that flexibly matches *memory reliability* with *application memory error tolerance*

Outline

- Motivation
- Characterizing application memory error tolerance
- Key observations
 - Observation 1: Memory error tolerance varies across applications and within an application
 - Observation 2: Data can be recovered by software
- Heterogeneous-Reliability Memory (HRM)
- Evaluation

Characterization Goal

Quantify application memory error tolerance



Characterization Methodology

- *3 modern data-intensive applications*

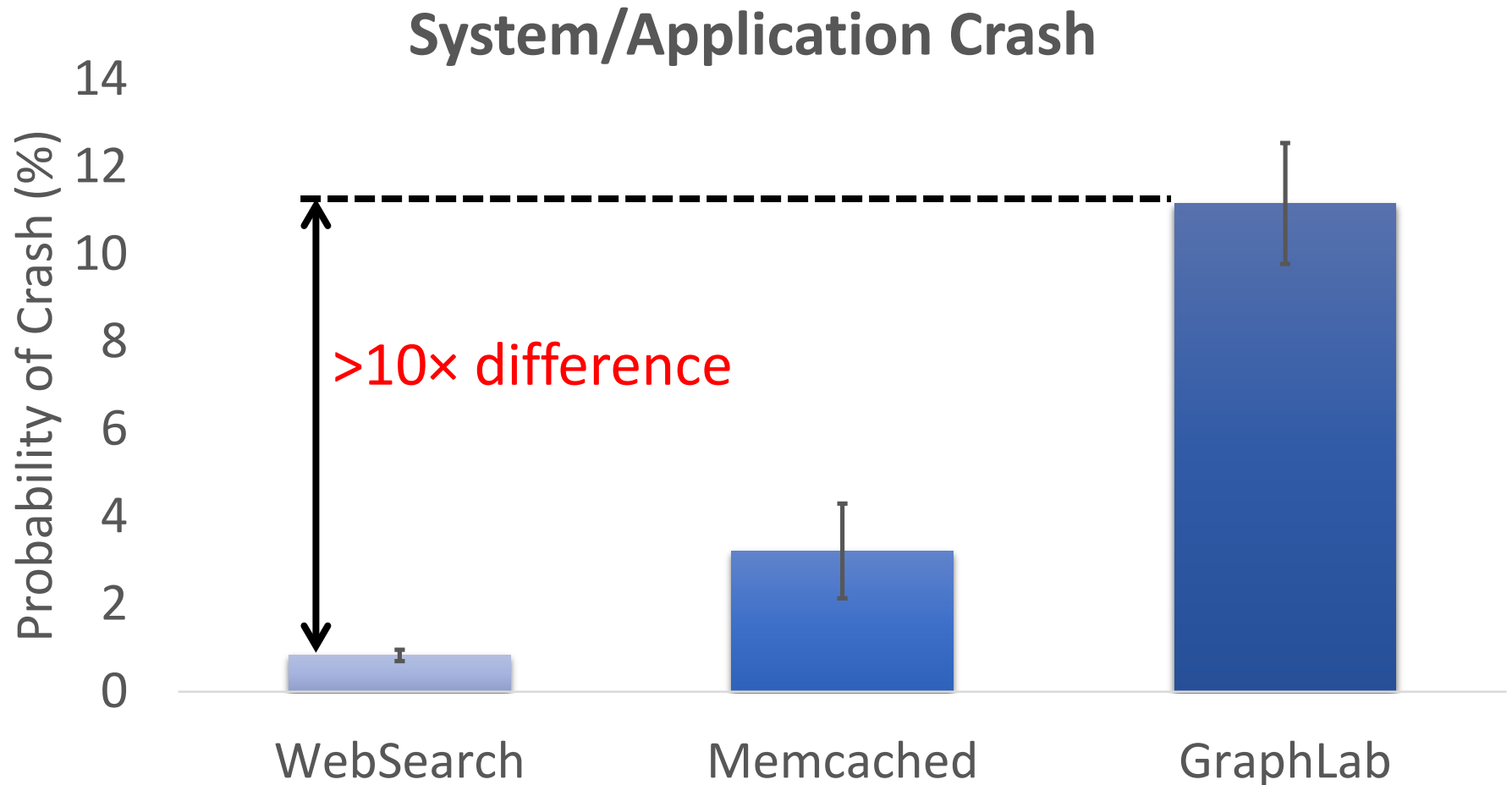
Application	WebSearch	Memcached	GraphLab
Memory footprint	46 GB	35 GB	4 GB

- *3 dominant memory regions*
 - Heap – dynamically allocated data
 - Stack – function parameters and local variables
 - Private – private heap managed by user
- *Injected a total of 23,718 memory errors using software debuggers (WinDbg and GDB)*
- *Examined correctness for over 4 billion queries*

Outline

- Motivation
- Characterizing application memory error tolerance
- Key observations
 - Observation 1: Memory error tolerance varies across applications and within an application
 - Observation 2: Data can be recovered by software
- Heterogeneous-Reliability Memory (HRM)
- Evaluation

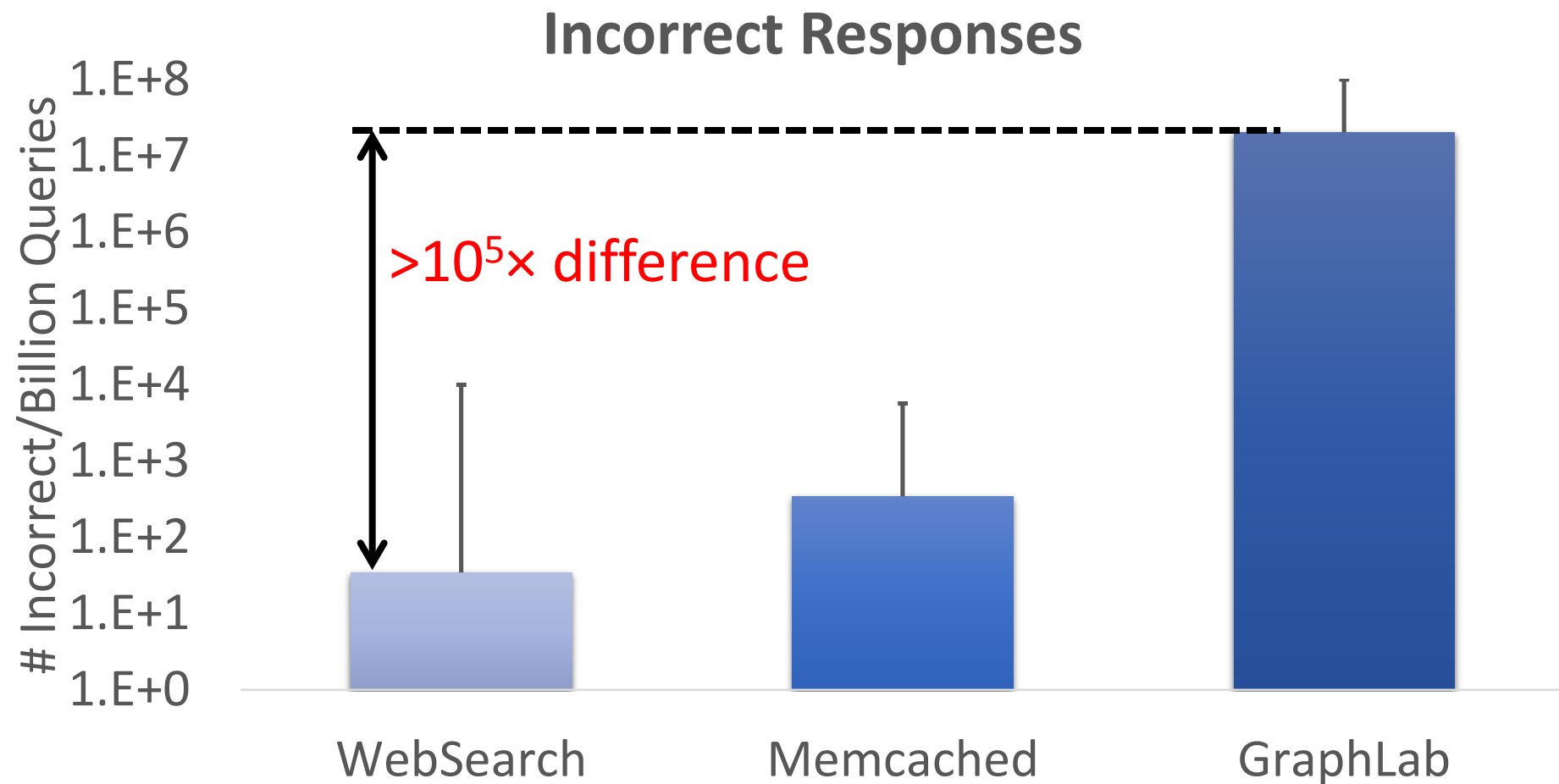
Observation 1: Memory Error Tolerance Varies Across Applications



Showing results for single-bit soft errors

Results for other memory error types can be found in the paper with similar conclusion

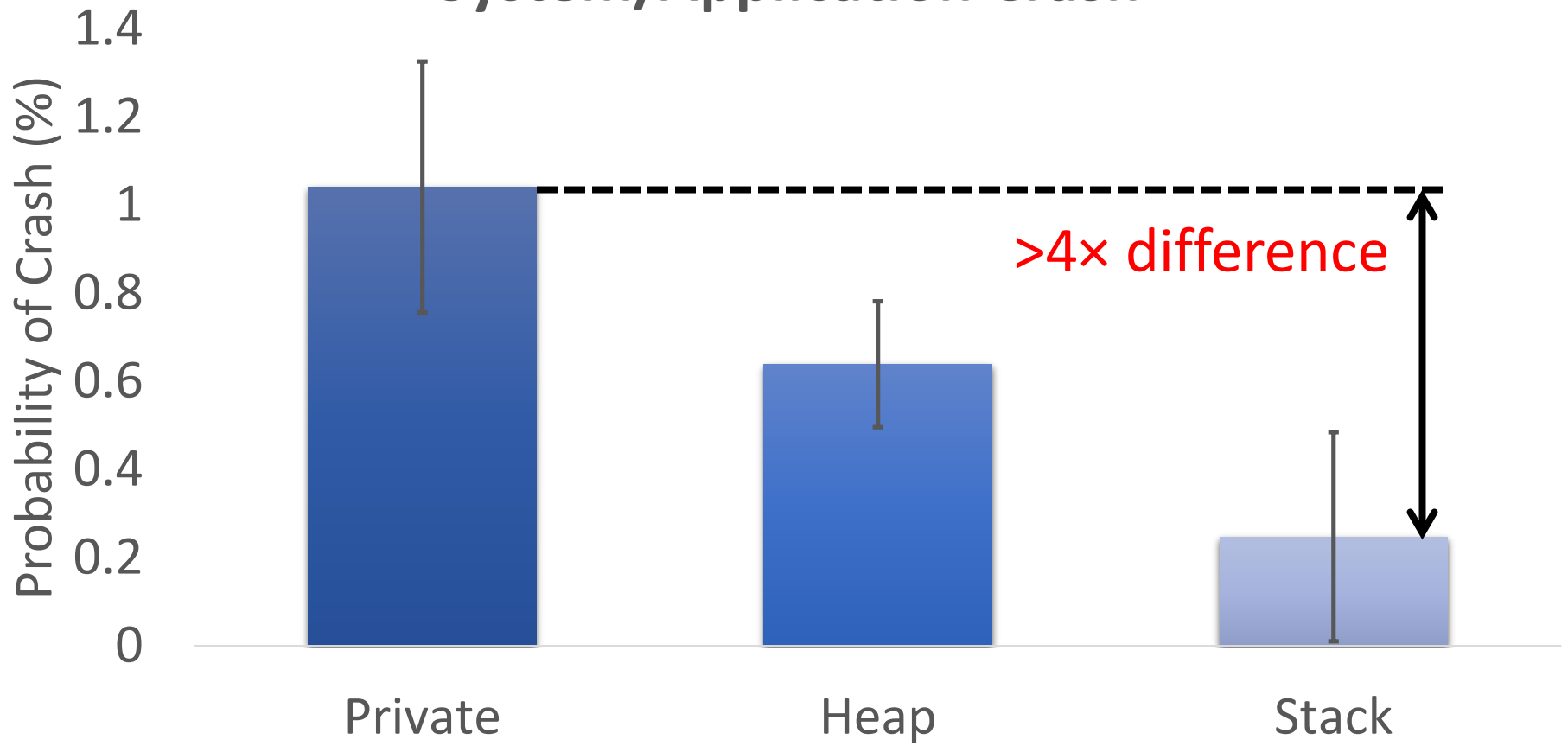
Observation 1: Memory Error Tolerance Varies Across Applications



Showing results for single-bit soft errors
Results for other memory error types can be found in the paper

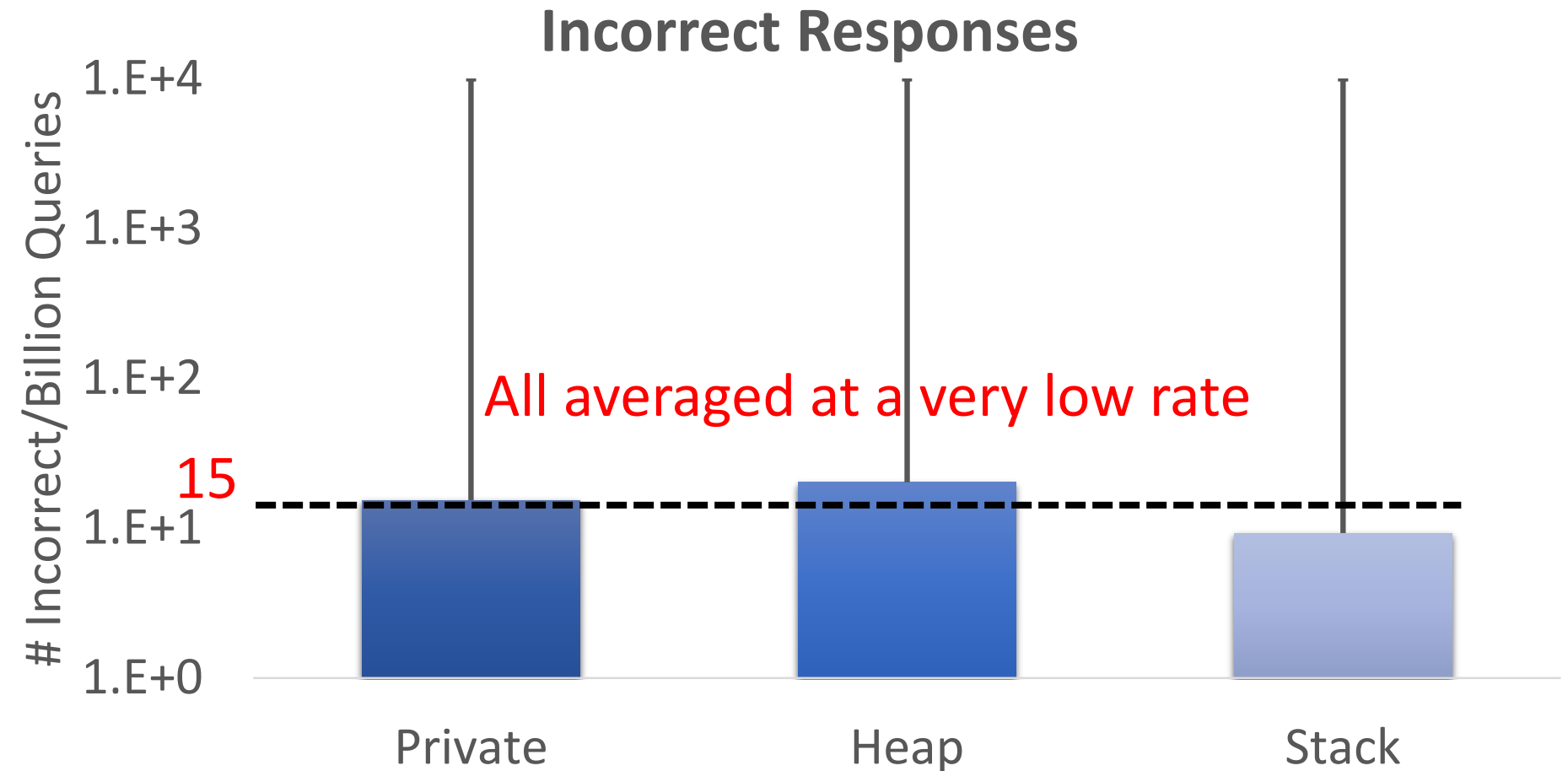
Observation 1: Memory Error Tolerance Varies Across Applications and **Within an Application**

System/Application Crash



Showing results for WebSearch
Results for other workloads can be found in the paper

Observation 1: Memory Error Tolerance Varies Across Applications and **Within an Application**



Showing results for WebSearch
Results for other workloads can be found in the paper

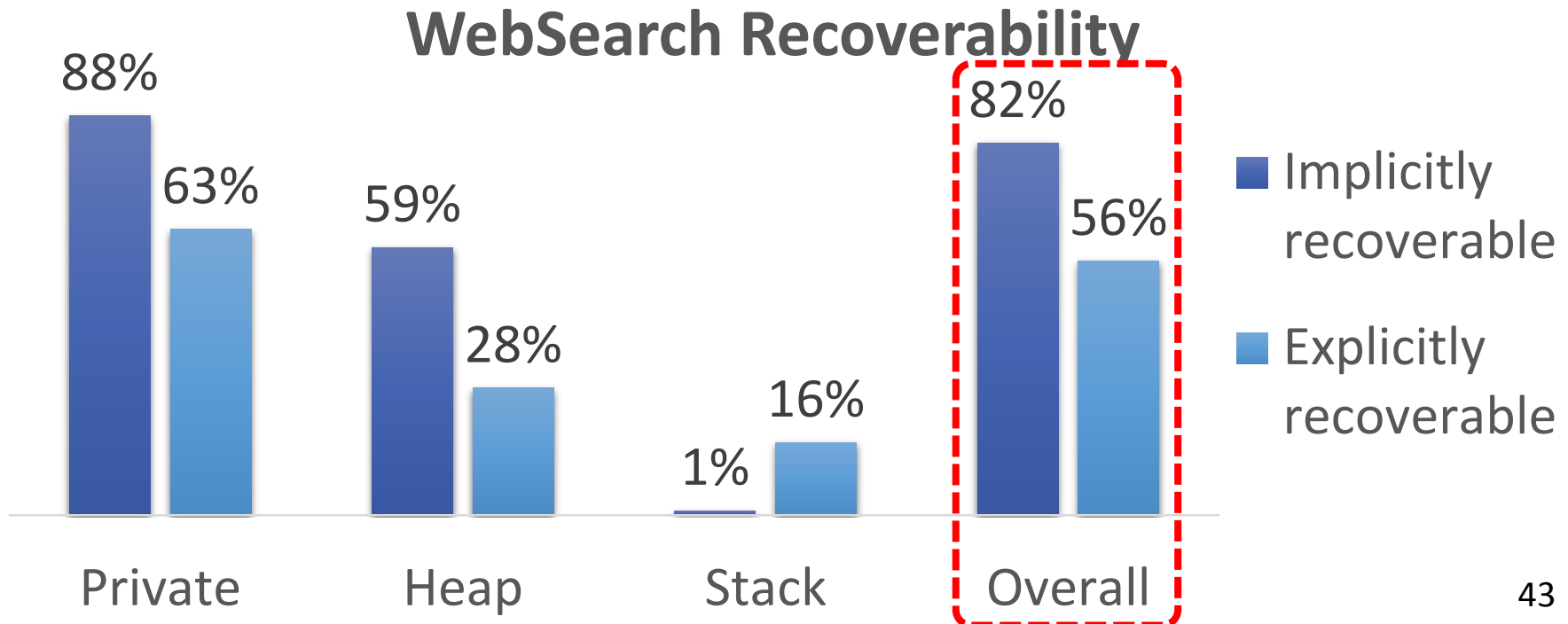
Outline

- Motivation
- Characterizing application memory error tolerance
- Key observations
 - Observation 1: Memory error tolerance varies across applications and within an application
 - Observation 2: Data can be recovered by software
- Heterogeneous-Reliability Memory (HRM)
- Evaluation

Observation 2: Data Can be Recovered by Software

Implicitly and Explicitly

- *Implicitly recoverable – application intrinsically has a clean copy of the data on disk*
- *Explicitly recoverable – application can create a copy of the data at a low cost (if it has very low write frequency)*



Outline

- Motivation
- Characterizing application memory error tolerance
- Key observations
 - Observation 1: Memory error tolerance varies across applications and within an application
 - Observation 2: Data can be recovered by software
- **Heterogeneous-Reliability Memory (HRM)**
- Evaluation

Exploiting Memory Error Tolerance

Vulnerable
data

Tolerant
data

Reliable memory

- ECC protected
- Well-tested chips

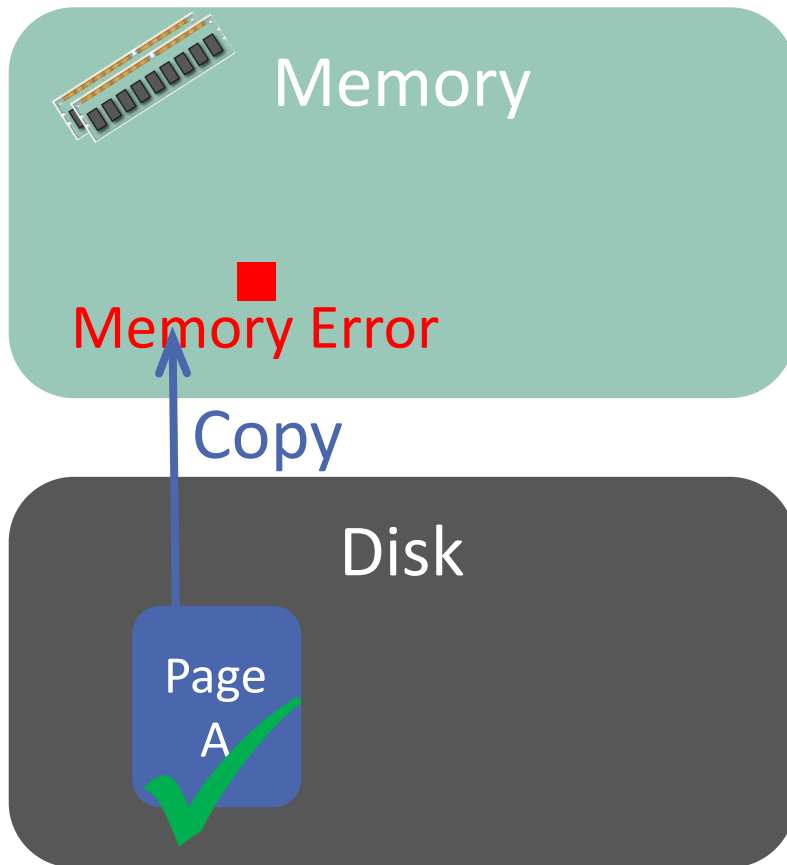
Low-cost memory

- NoECC or Parity
- Less-tested chips

Heterogeneous-Reliability Memory

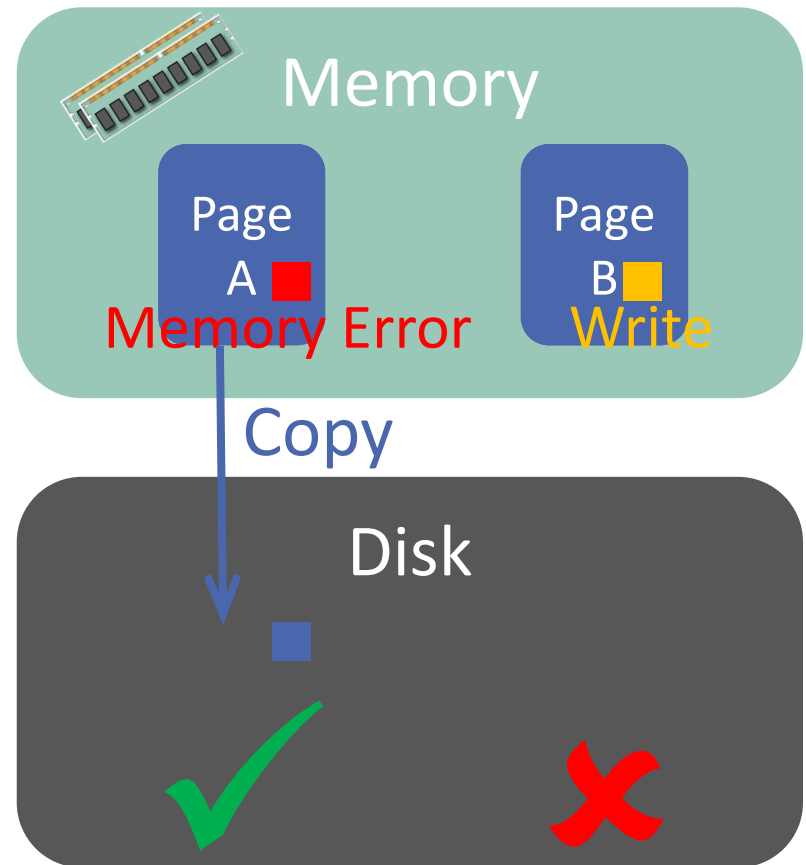
Par+R: Parity Detection + Software Recovery

Implicit Recovery



Intrinsic
copy

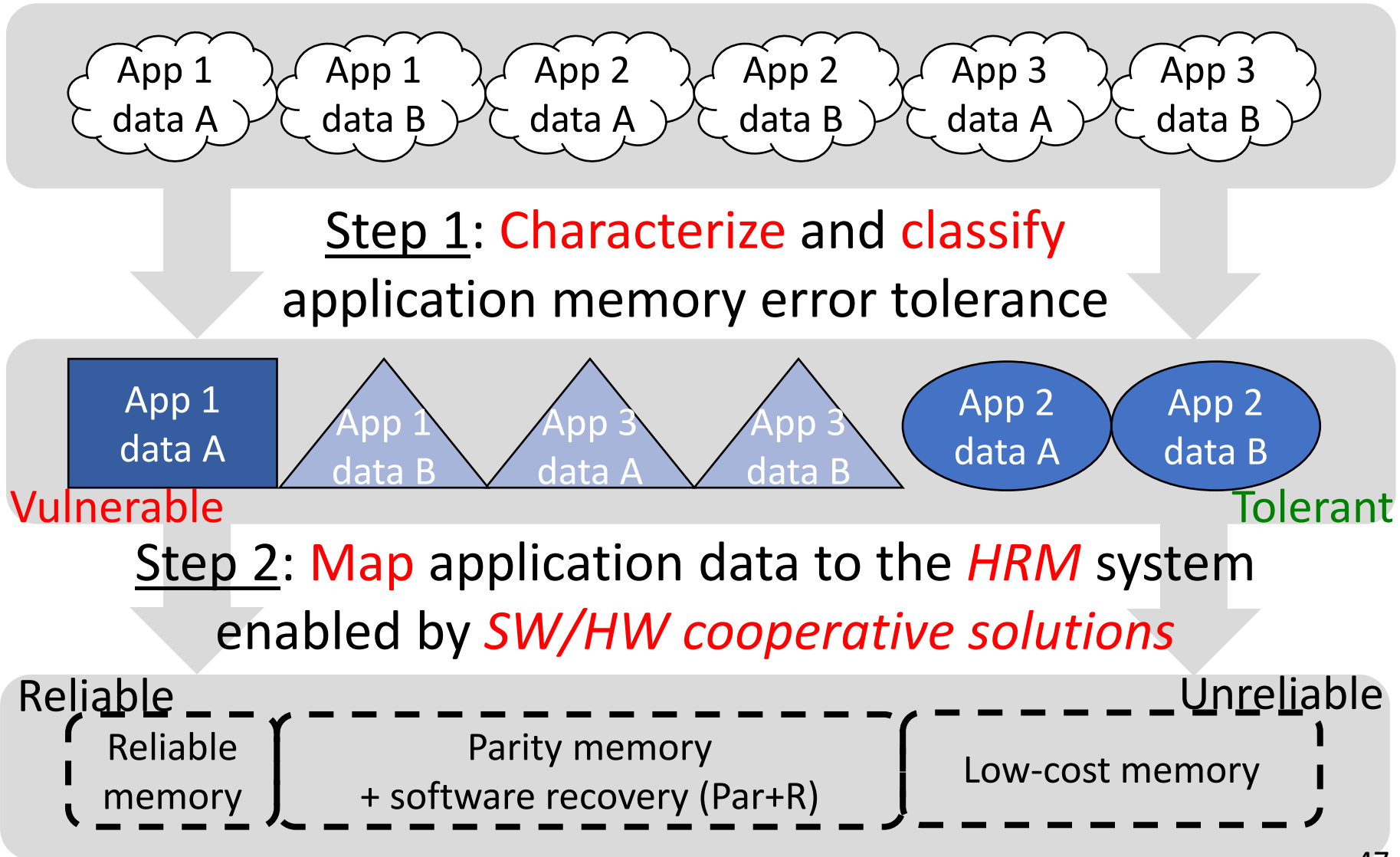
Explicit Recovery



Write non-
intensive

Write
intensive

Heterogeneous-Reliability Memory



Outline

- Motivation
- Characterizing application memory error tolerance
- Key observations
 - Observation 1: Memory error tolerance varies across applications and within an application
 - Observation 2: Data can be recovered by software
- Heterogeneous-Reliability Memory (HRM)
- Evaluation

Evaluated Systems

Configuration	Mapping			Pros and Cons
	Private (36 GB)	Heap (9 GB)	Stack (60 MB)	
<u>Typical Server</u>	ECC	ECC	ECC	Reliable but expensive
<u>Consumer PC</u>	NoECC	NoECC	NoECC	Low-cost but unreliable
<u>HRM</u>	Par+R	NoECC	NoECC	Parity only
<u>Less-Tested (L)</u>	NoECC	NoECC	NoECC	Least expensive and reliable
<u>HRM/L</u>	ECC	Par+R	NoECC	Low-cost and reliable HRM

 Baseline systems

 HRM systems

Design Parameters

DRAM/server HW cost [Kozyrakis '10]	30%
NoECC memory cost savings	11.1%
Parity memory cost savings	9.7%
Less-tested memory cost savings	18%±12%
Crash recovery time	10 mins
Par+R flush threshold	5 mins
Errors/server/month [Schroeder '09]	2000
Target single server availability	99.90%

Evaluation Metrics

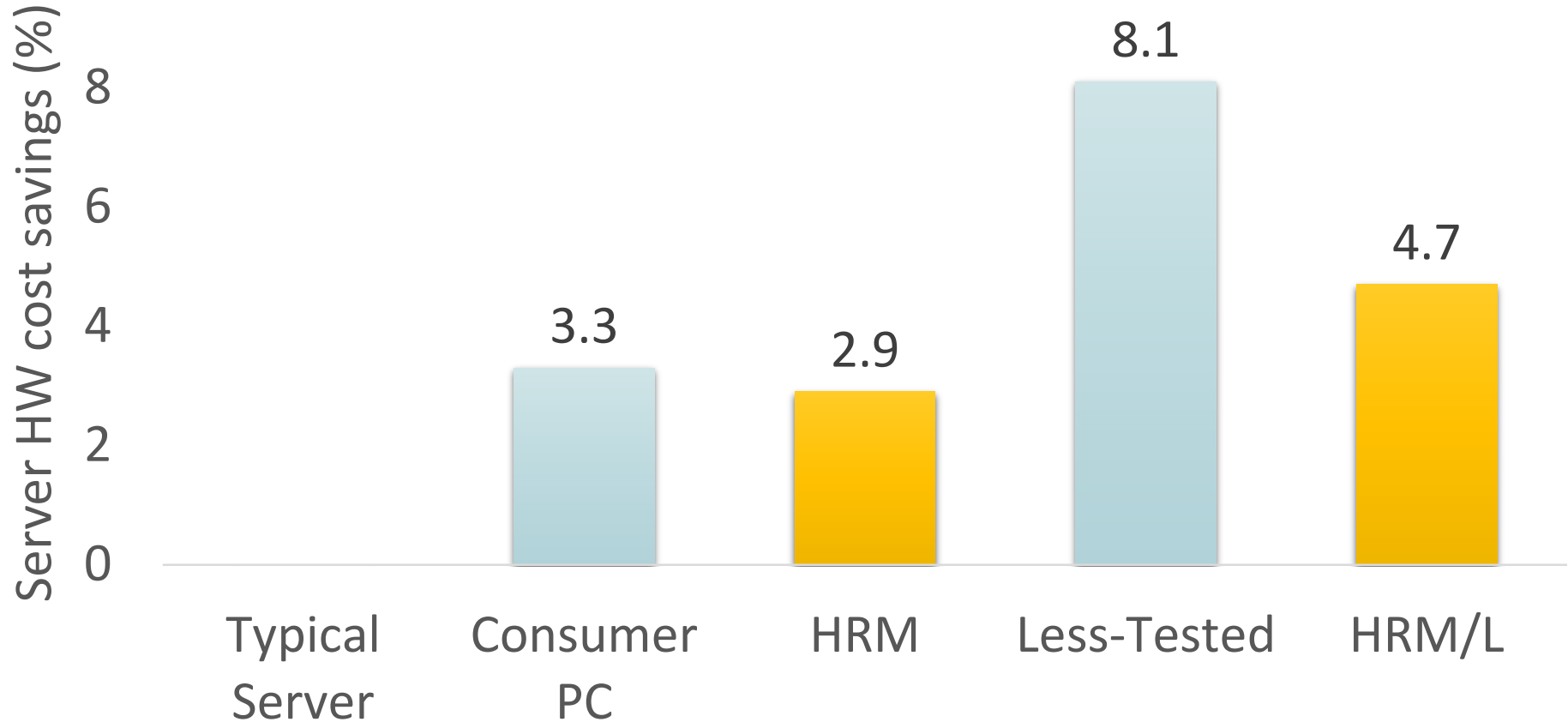
- *Cost*

- Memory cost savings
- Server HW cost savings
(both compared with *Typical Server*)

- *Reliability*

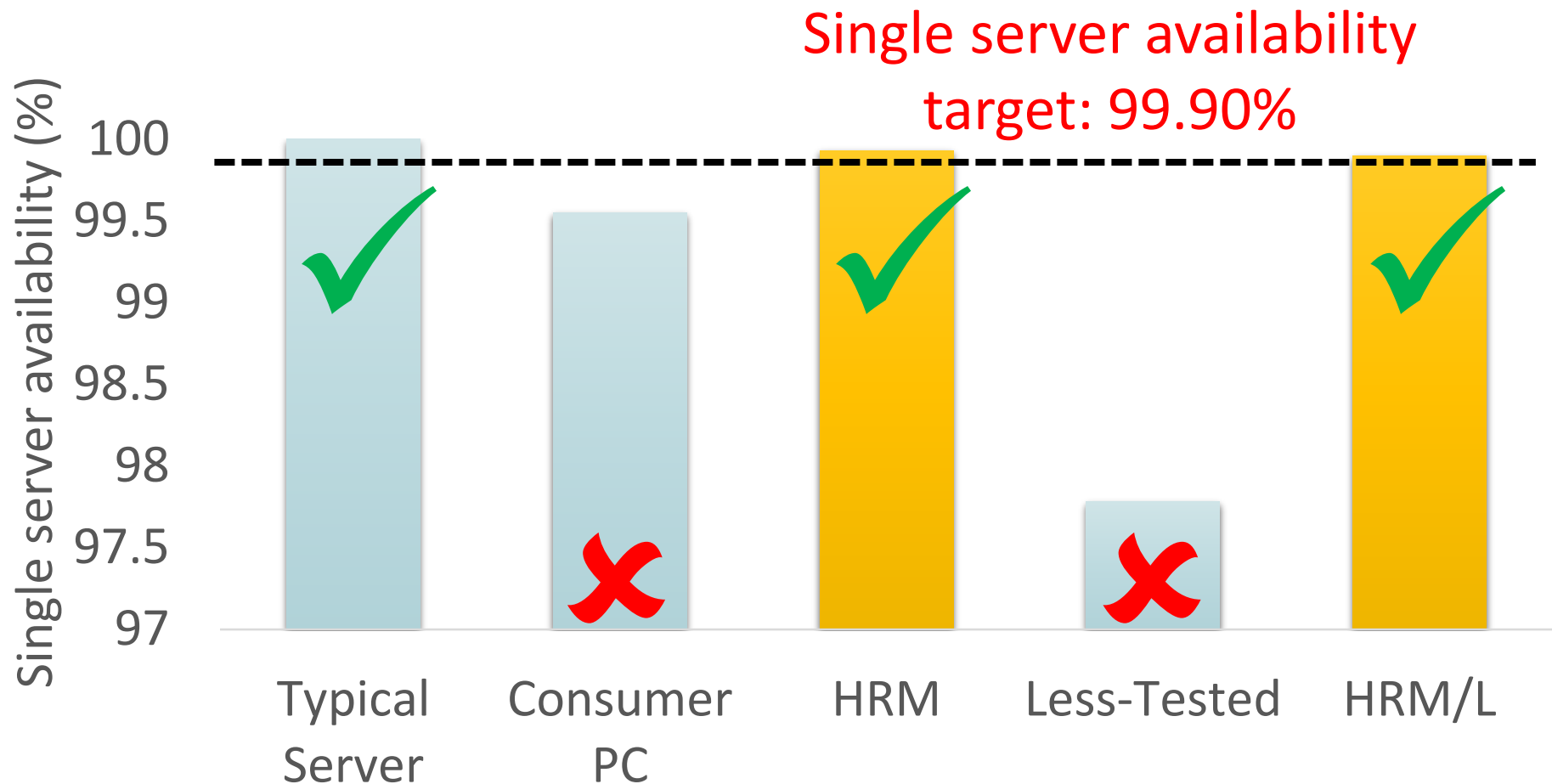
- Crashes/server/month
- Single server availability
- # incorrect/million queries

Improving Server HW Cost Savings



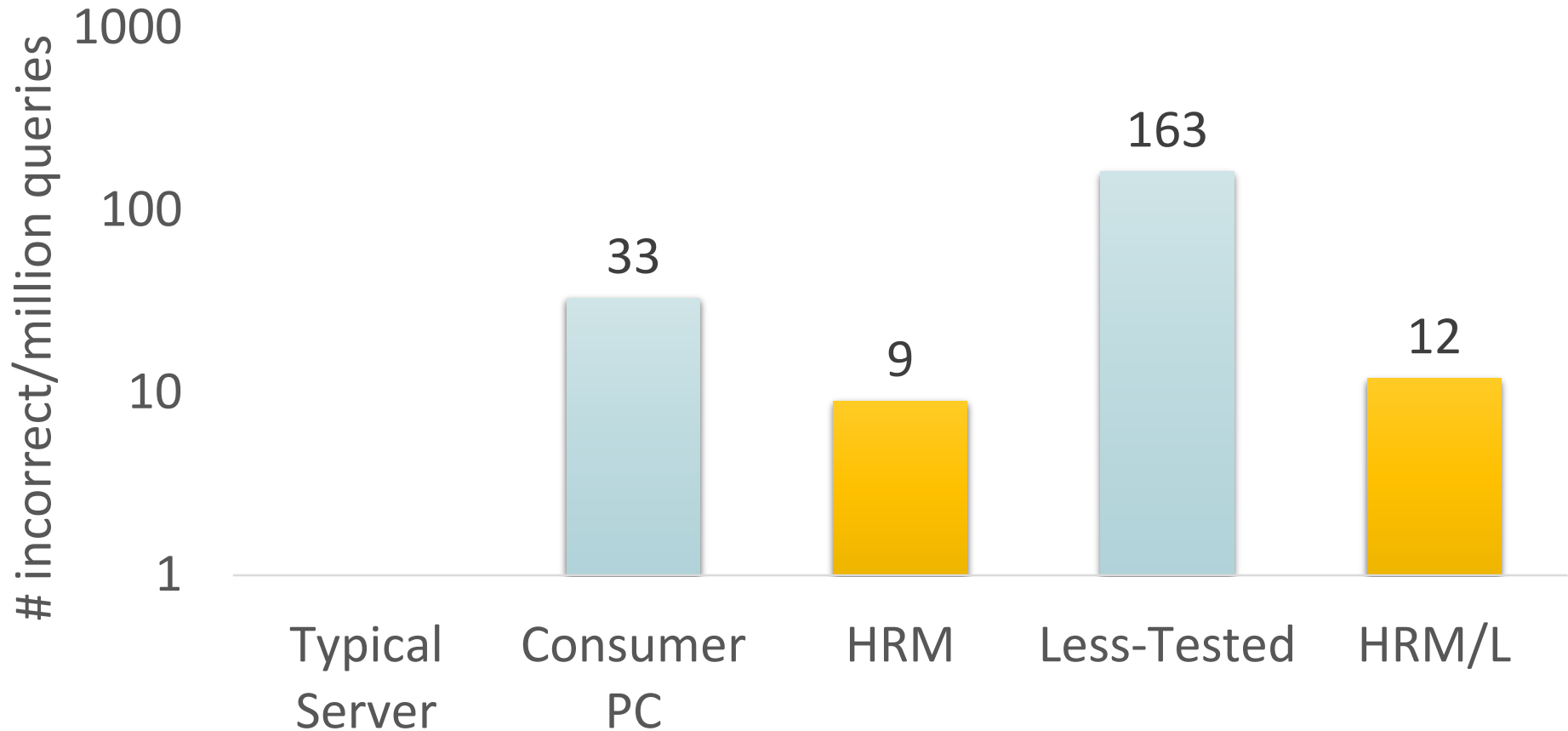
Reducing the use of memory error mitigation techniques in part of memory space can save noticeable amount of server HW cost

Achieving Target Availability



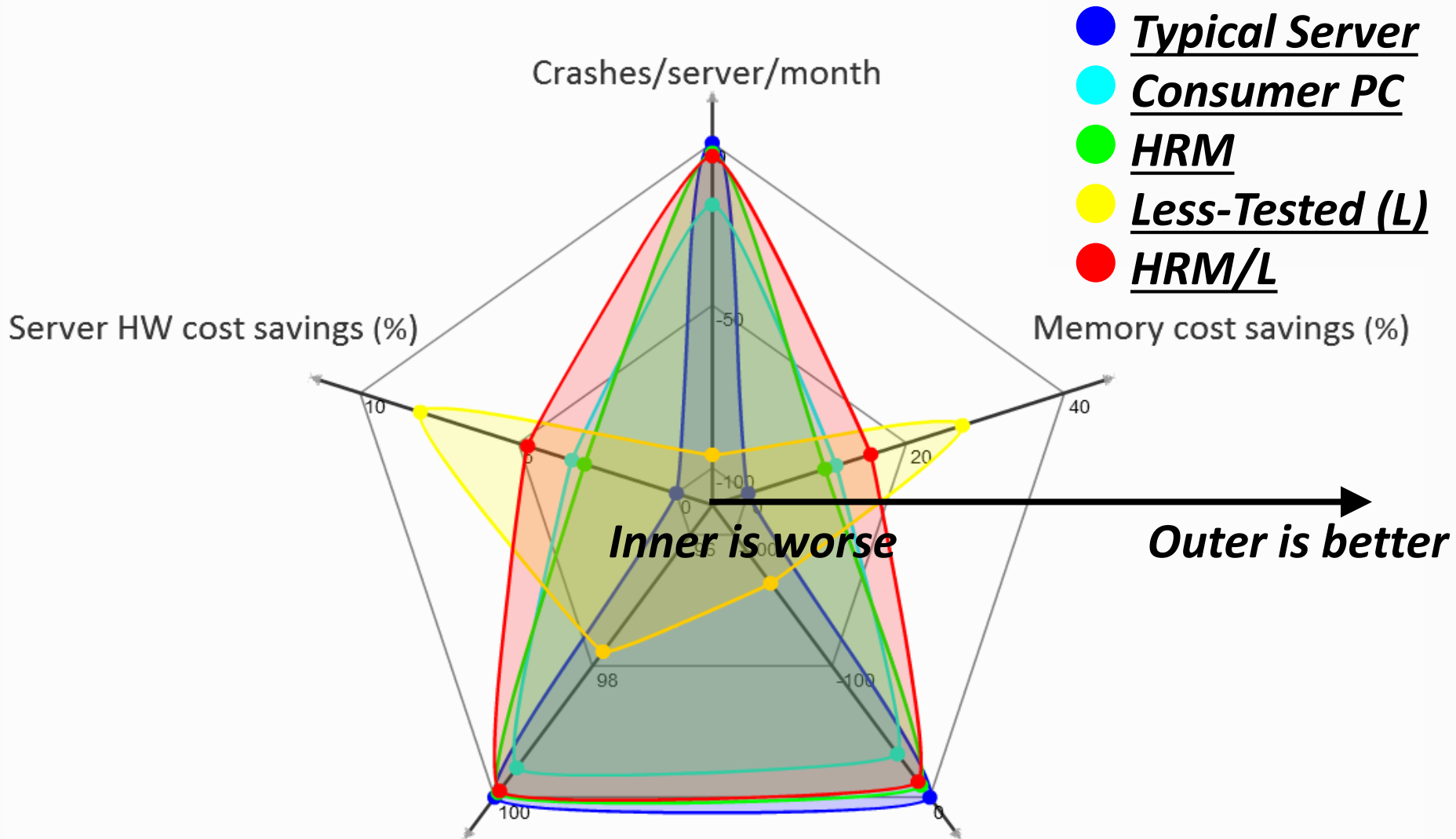
HRM systems are flexible to adjust and can achieve availability target

Achieving Acceptable Correctness



HRM systems can achieve acceptable correctness

Evaluation Results



● ● Bigger area means better tradeoff

Other Results and Findings

- *Characterization of applications' reactions to memory errors*
 - Finding: Quick-to-crash vs. periodically incorrect behavior
- *Characterization of most common types of memory errors including single-bit soft/hard errors, multi-bit hard errors*
 - Finding: More severe errors mainly decrease correctness
- *Characterization of how errors are masked*
 - Finding: Some memory regions are safer than others
- *Discussion about heterogeneous reliability design dimensions, techniques, and their benefits and tradeoffs*

Conclusion

- Our Goal: Reduce datacenter *cost*; meet *availability* target
- Characterized application-level memory error tolerance of 3 modern data-intensive workloads
- Proposed *Heterogeneous-Reliability Memory (HRM)*
 - Store error-tolerant data in less-reliable lower-cost memory
 - Store error-vulnerable data in more-reliable memory
- Evaluated example HRM systems
 - Reduce server hardware *cost* by 4.7 %
 - Achieve single-server *availability* target 99.90 %

More Information ...

- Yixin Luo, Sriram Govindan, Bikash Sharma, Mark Santaniello, Justin Meza, Aman Kansal, Jie Liu, Badriddine Khessib, Kushagra Vaid, and Onur Mutlu,

"Characterizing Application Memory Error Vulnerability to Optimize Data Center Cost via Heterogeneous-Reliability Memory"

Proceedings of the 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), Atlanta, GA, June 2014.

[Slides \(pptx\)](#) [\(pdf\)](#) [Coverage on ZDNet](#)

Some New Ideas (This Year)

■ Specialization

- Heterogeneous Reliability Memory [DSN 2014]
- Heterogeneous Block Architecture [ICCD 2014]

■ Persistent Memory

- Loose Ordering Consistency for Persistent Memory [ICCD 2014]
- Transparent Consistency for Persistent/Hybrid Memory [in progress]

■ Memory Reliability/Security

- Row Hammer Problem in DRAM [ISCA 2014]
- Neighbor-Cell Assisted Error Correction in Flash [SIGMETRICS 2014]
- Error Mitigation for Intermittent DRAM Failures [SIGMETRICS 2014]

■ Memory Performance

- The Dirty-Block Index [ISCA 2014]
- DRAM Refresh-Access Parallelization [HPCA 2014]
- The Blacklisting Memory Scheduler [ICCD 2014]
- Exploiting Read-Write Disparity in Caches [HPCA 2014]

Some New Ideas in Memory System Design for Data-Intensive Computing

Onur Mutlu

onur@cmu.edu

September 4, 2014

ISTC-CC Retreat

Carnegie Mellon

Backup Slides

The Dirty-Block Index

The Dirty-Block Index

ISCA 2014

Vivek Seshadri

Abhishek Bhowmick • Onur Mutlu

Phillip B. Gibbons • Michael A. Kozuch • Todd C. Mowry

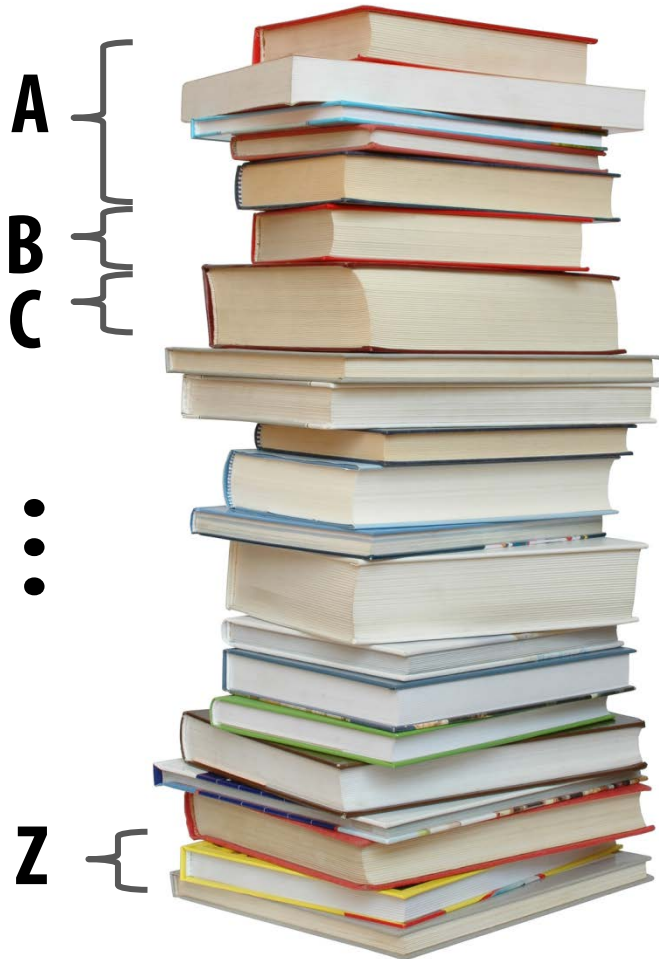
SAFARI

Carnegie Mellon



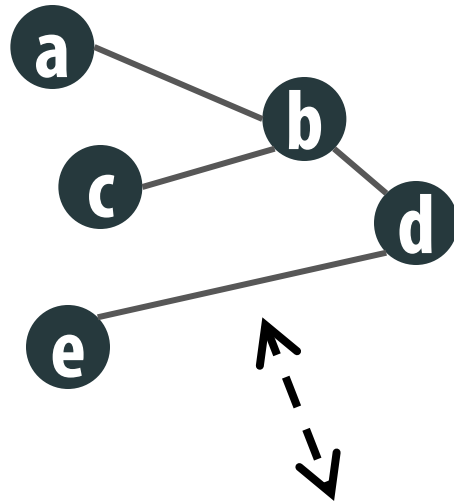
Mismatch: Representation and Query

Sorted by Title



**Get all the books
written by author X**

Mismatch: Representation and Query

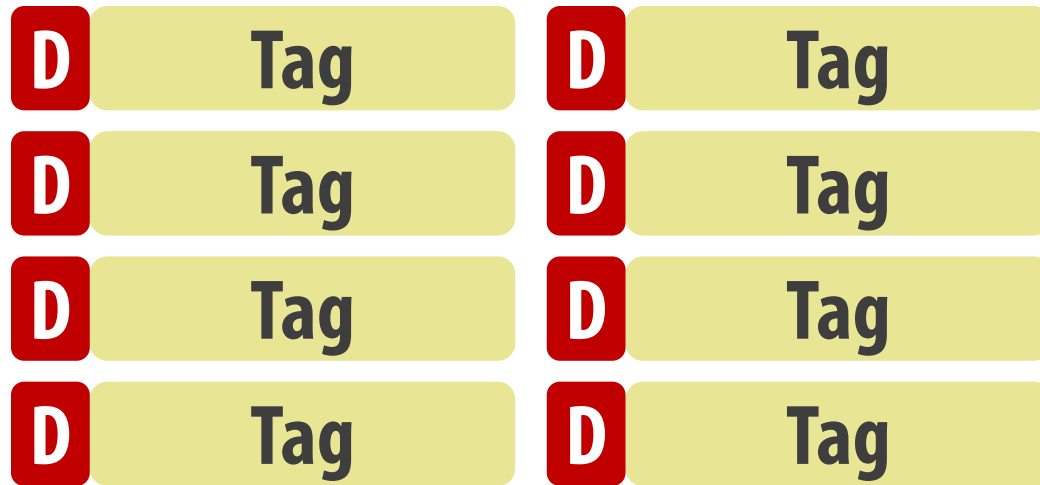

$$\left\{ \begin{array}{ccccc} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{array} \right\}$$

**Breadth First
Search**

**List all edges
adjacent to
vertex 'a'**

Mismatch: Representation and Query

Cache Tag Store



Dirty Bit



**List all dirty
blocks of
DRAM row R.**

**Is block X
dirty?**

Dirty-Block Index

Cache Tag Store

Tag	Tag
Tag	Tag
Tag	Tag
Tag	Tag

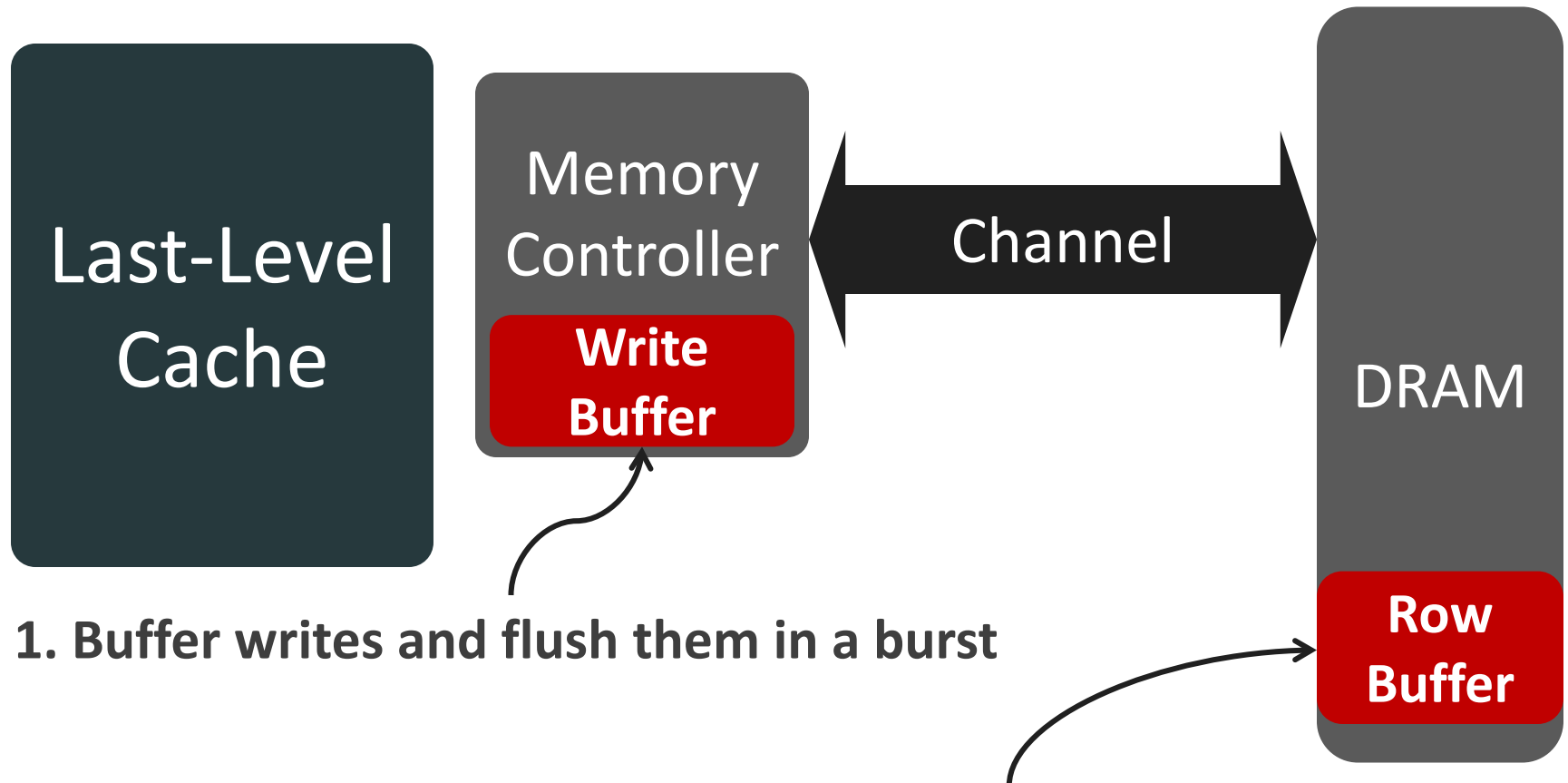
DBI

**List all dirty
blocks of
DRAM row R.**

**Is block X
dirty?**

Application: DRAM-Aware Writeback

Virtual Write Queue [ISCA 2010], DRAM-Aware Writeback [TR-HPS-2010-2]

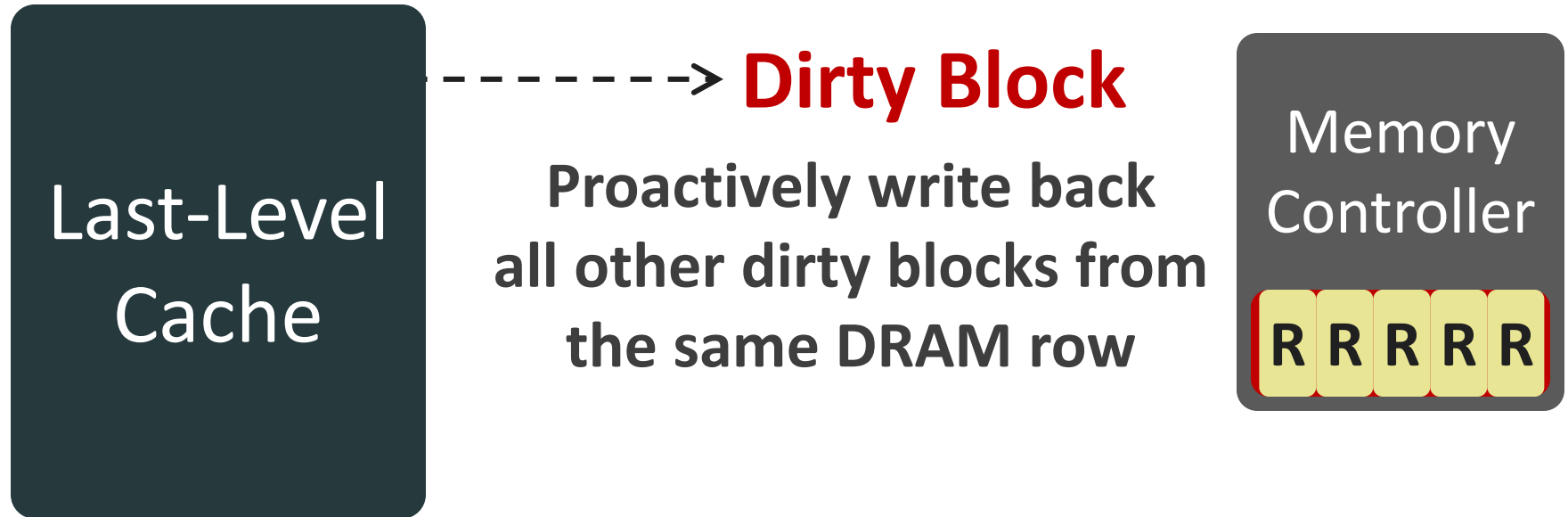


1. Buffer writes and flush them in a burst

2. Row buffer hits are faster and more efficient than row misses

Application: DRAM-Aware Writeback

Virtual Write Queue [ISCA 2010], DRAM-Aware Writeback [TR-HPS-2010-2]



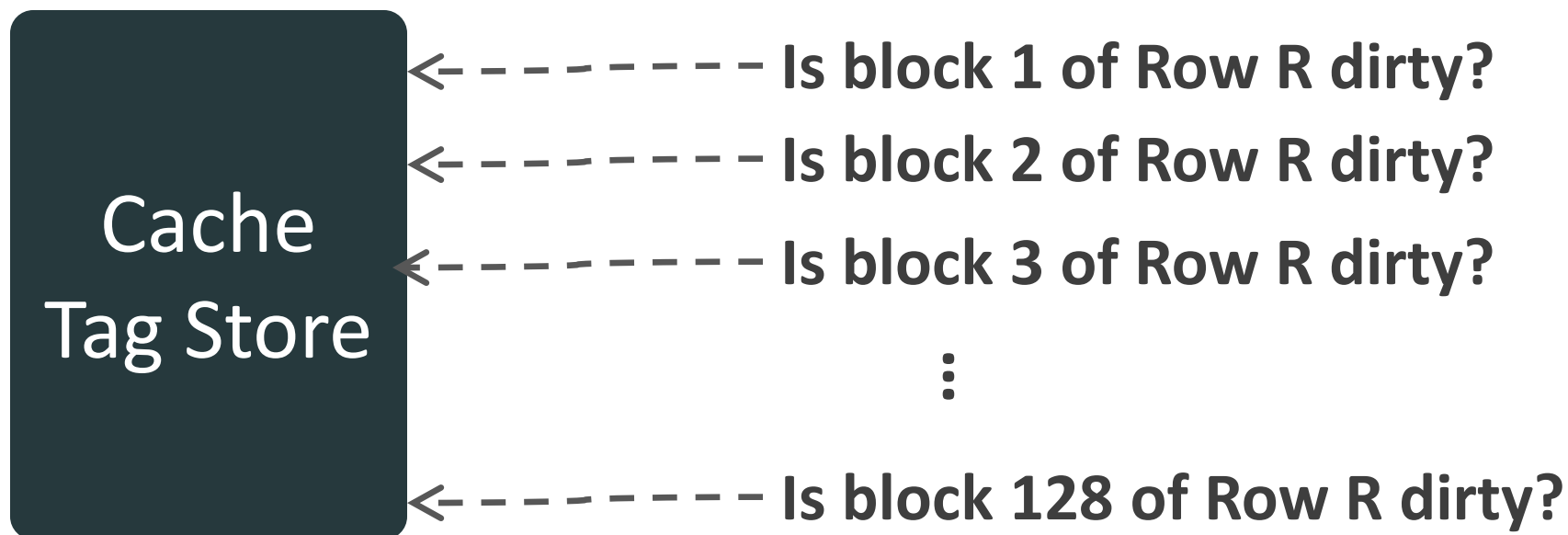
Significantly increases the DRAM write row hit rate

Get all dirty blocks of DRAM row 'R'

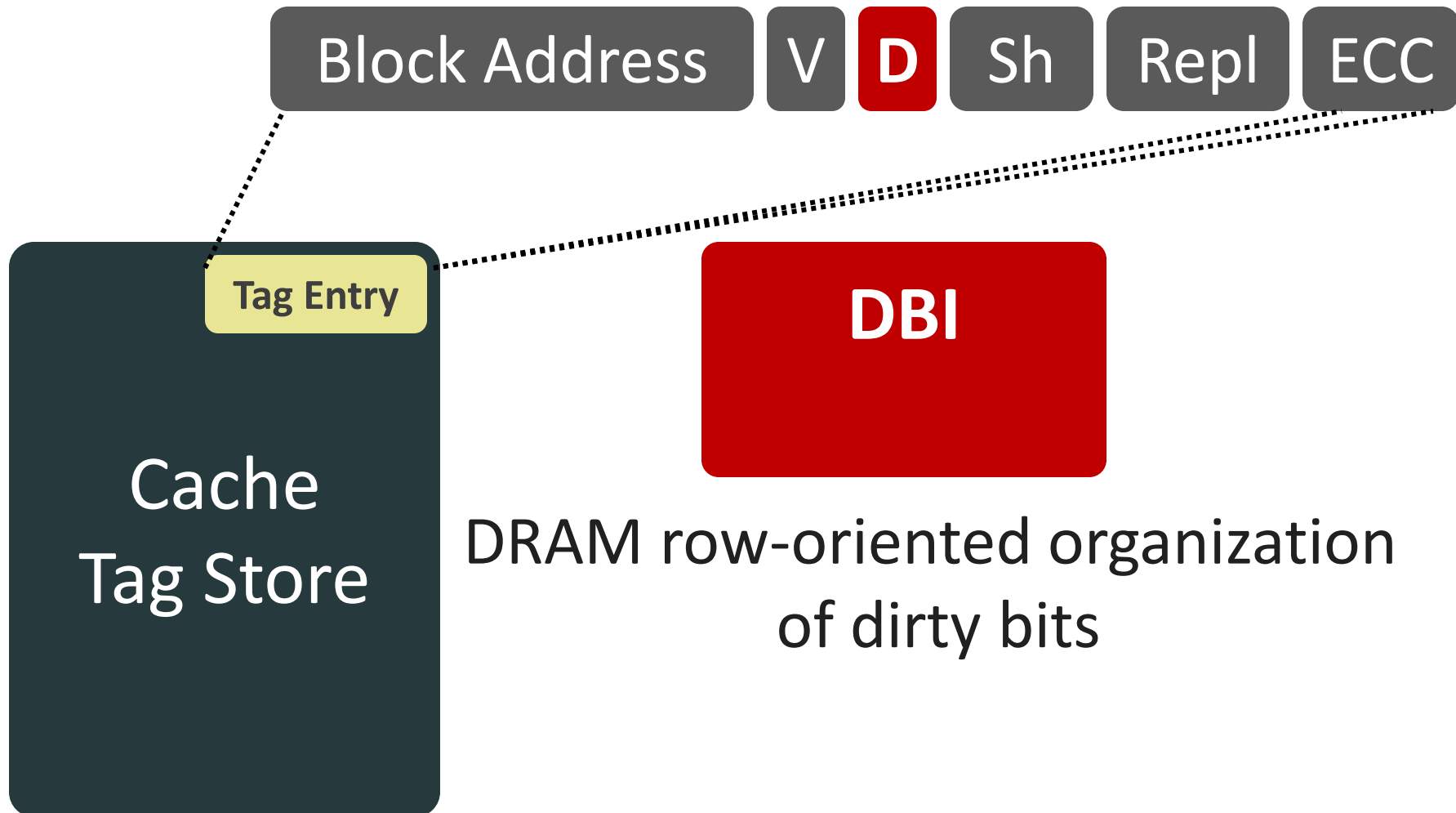
Shortcoming of Block-Oriented Organization

Get all dirty blocks of DRAM row 'R'

Set of blocks co-located in DRAM
~8KB = 128 cache blocks

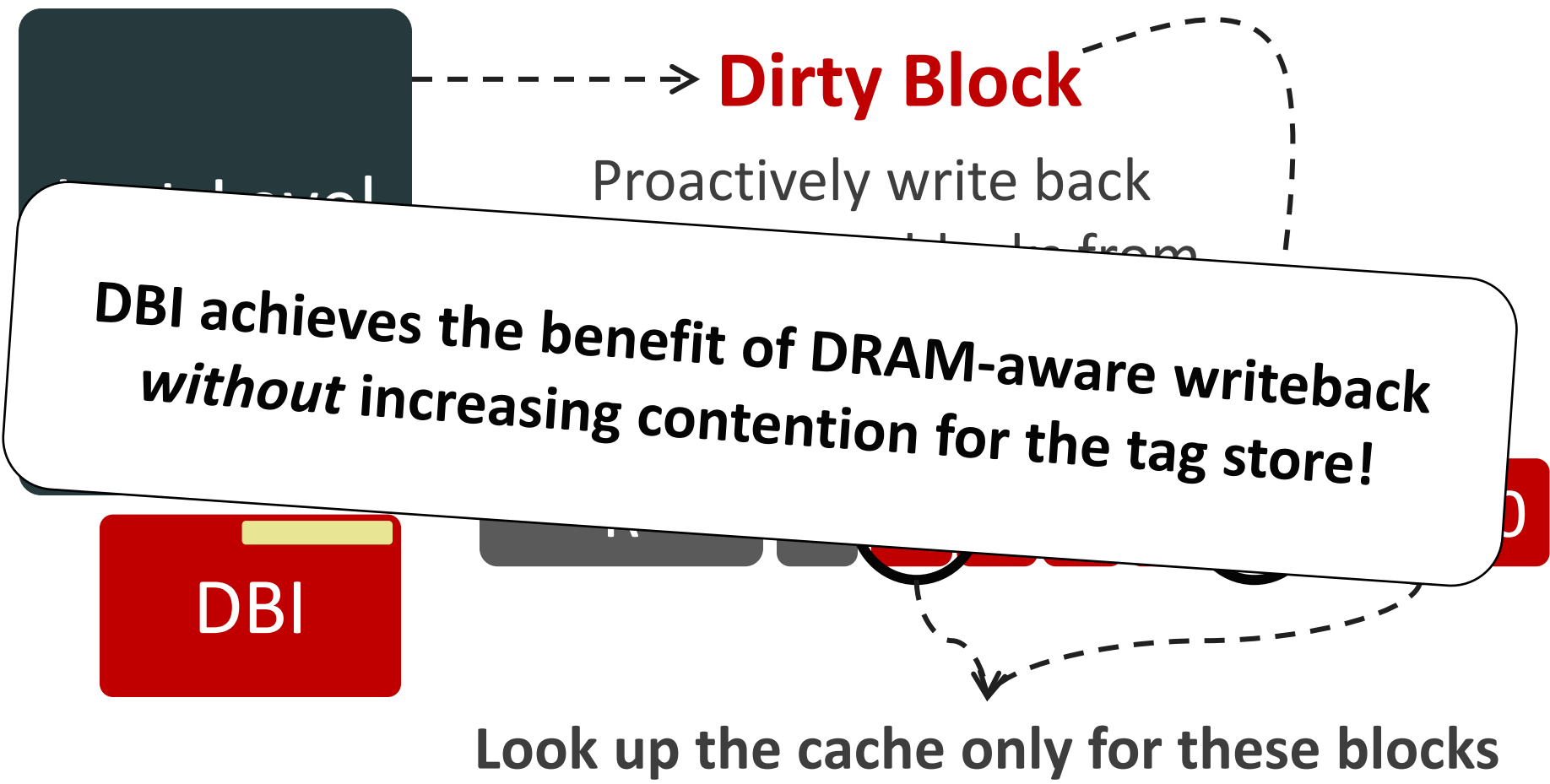


The Dirty-Block Index (DBI)



1 DRAM-Aware Writeback w/ DBI

Virtual Write Queue [ISCA 2010], DRAM-Aware Writeback [TR-HPS-2010-2]



Many Optimizations

1. **DRAM-aware writeback**
2. **Bypassing cache lookups**
3. **Reducing ECC overhead**
4. **Efficient cache flushing**
5. **Load balancing memory accesses**
6. **Bulk DMA**
7. **Efficient write scheduling**
- ...

DBI

Many Optimizations

1. DRAM-aware writeback
2. Bypassing cache lookups
3. Reducing ECC overhead

DBI

31% performance over baseline
6% over best previous mechanism
8% cache area reduction

4. Efficient cache flushing

5.
6.
7.

...

More Information ...

- Vivek Seshadri, Abhishek Bhowmick, [Onur Mutlu](#), Phillip B. Gibbons, Michael A. Kozuch, and Todd C. Mowry, **"The Dirty-Block Index"**
*Proceedings of the [41st International Symposium on Computer Architecture \(ISCA\)](#), Minneapolis, MN, June 2014. [Slides \(pptx\)](#) [\(pdf\)](#)
[Lightning Session Slides \(pptx\)](#) [\(pdf\)](#)*

Refresh-Access Parallelization

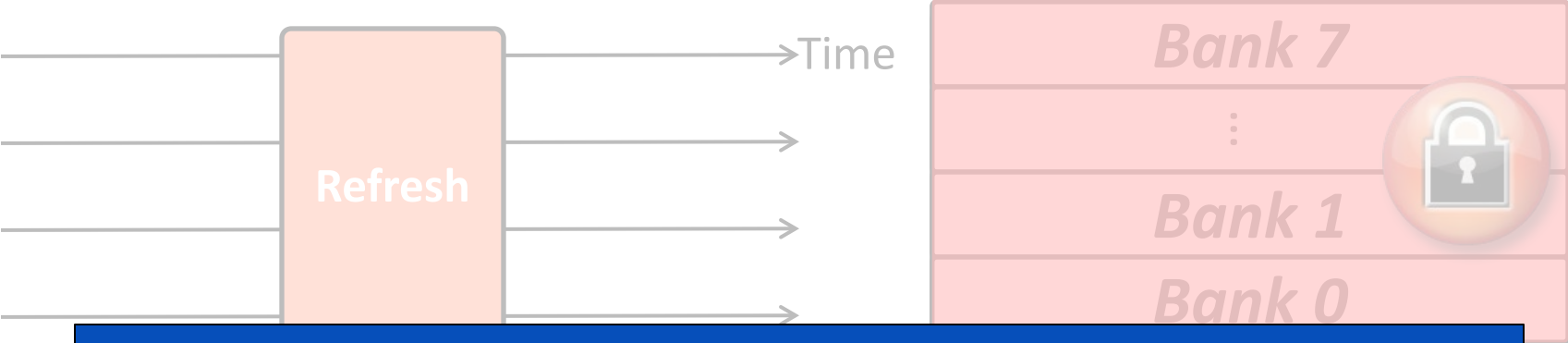
Refresh Penalty



Refresh delays requests by 100s of ns

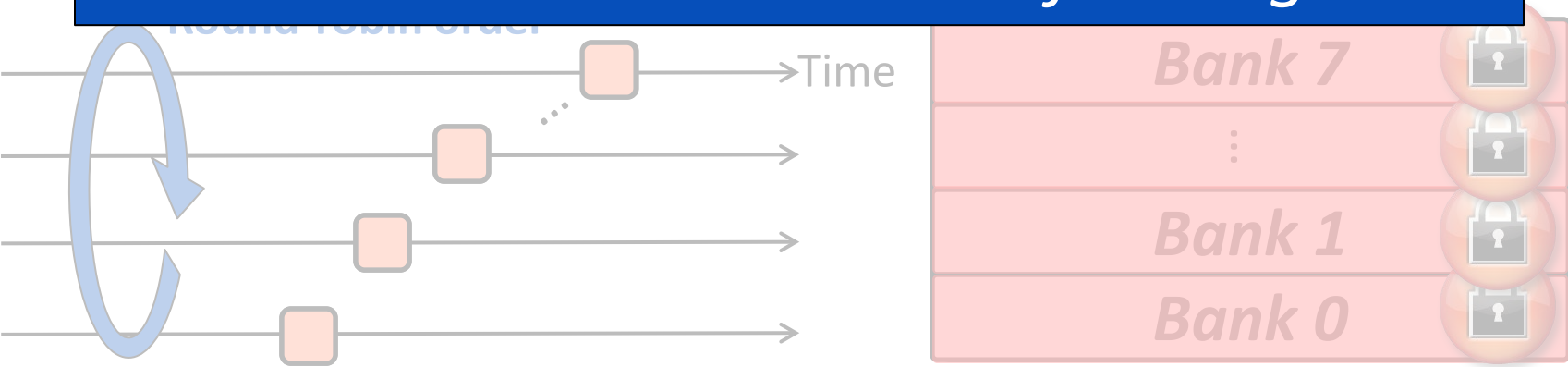
Existing Refresh Modes

All-bank refresh in commodity DRAM (DDR_x)



Per-bank refresh allows accesses to other banks while a bank is refreshing

Per



Shortcomings of Per-Bank Refresh

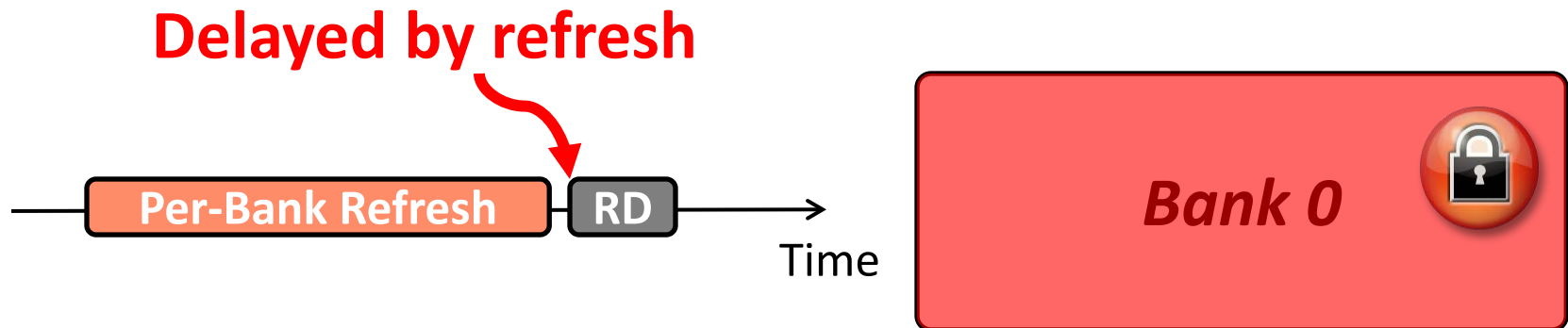
- Problem 1: Refreshes to different banks are scheduled in a **strict round-robin order**
 - The static ordering is hardwired into DRAM chips
 - **Refreshes busy banks with many queued requests when other banks are idle**
- Key idea: Schedule per-bank refreshes to idle banks opportunistically in a dynamic order

Our First Approach: DARP

- **Dynamic Access-Refresh Parallelization (DARP)**
 - An improved scheduling policy for **per-bank refreshes**
 - Exploits **refresh scheduling flexibility** in DDR DRAM
- Component 1: **Out-of-order per-bank refresh**
 - Avoids poor static scheduling decisions
 - Dynamically issues per-bank refreshes to idle banks
- Component 2: **Write-Refresh Parallelization**
 - Avoids refresh interference on latency-critical reads
 - Parallelizes refreshes with **a batch of writes**

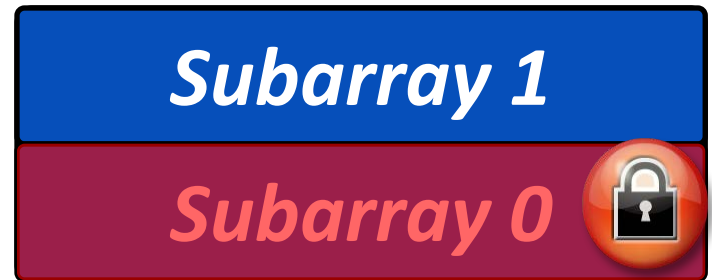
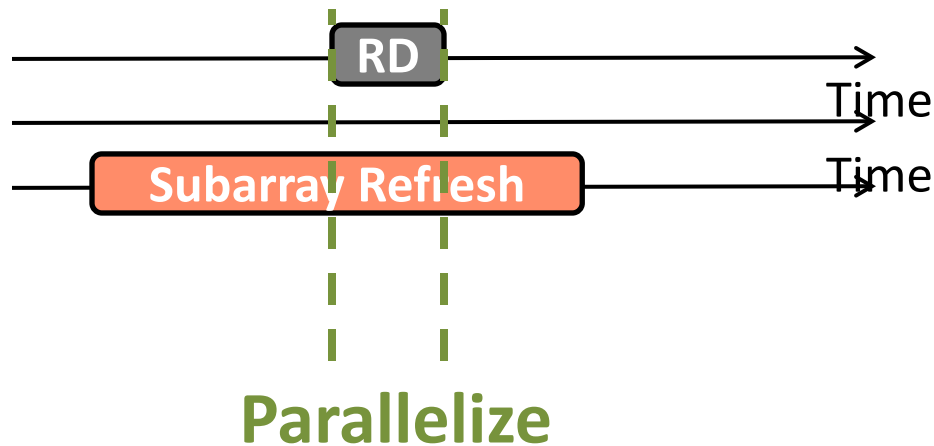
Shortcomings of Per-Bank Refresh

- Problem 2: Banks that are being refreshed cannot concurrently serve memory requests



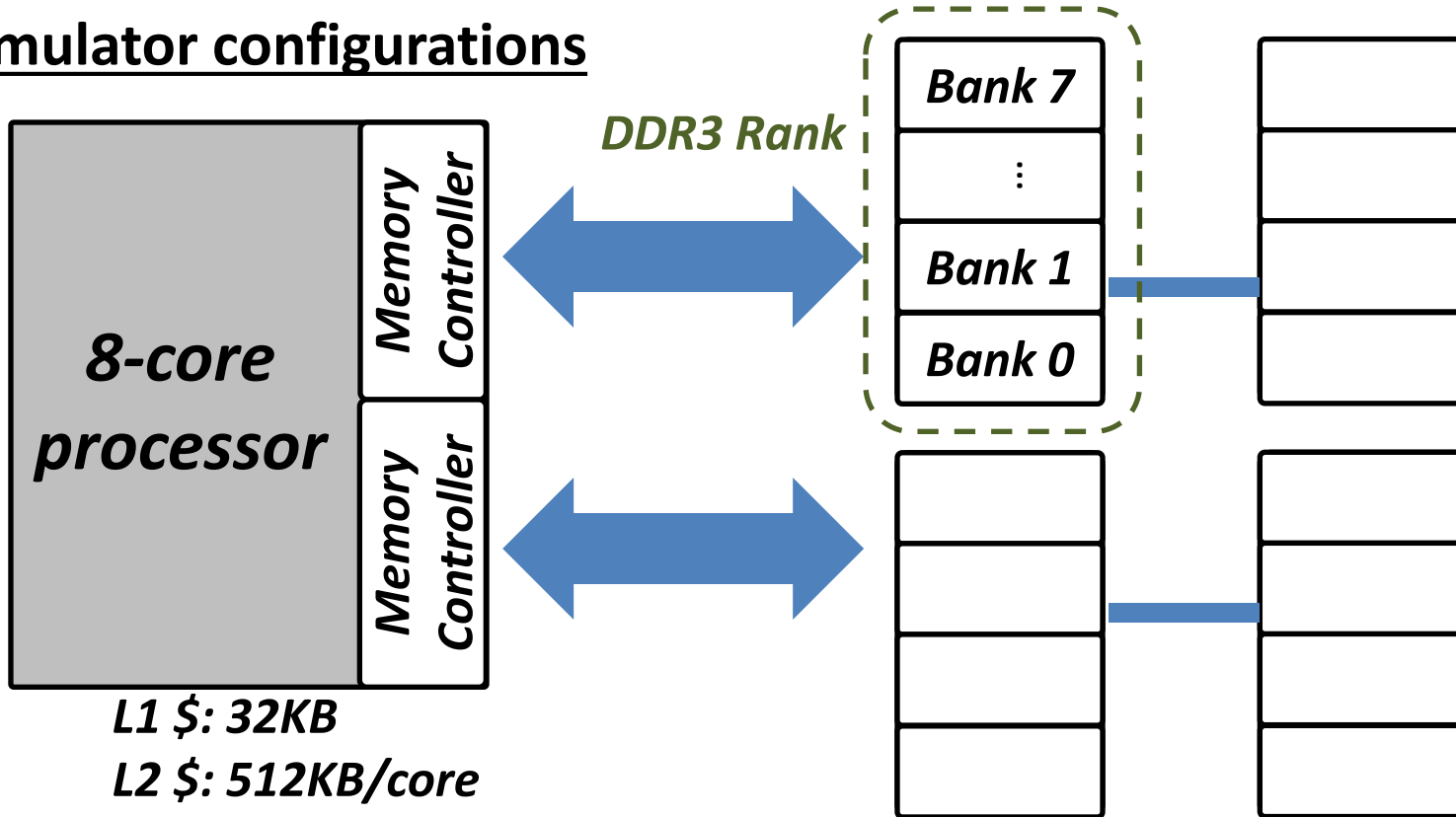
Shortcomings of Per-Bank Refresh

- Problem 2: Refreshing banks cannot concurrently serve memory requests
- Key idea: Exploit **subarrays** within a bank to parallelize refreshes and accesses across **subarrays**



Methodology

Simulator configurations

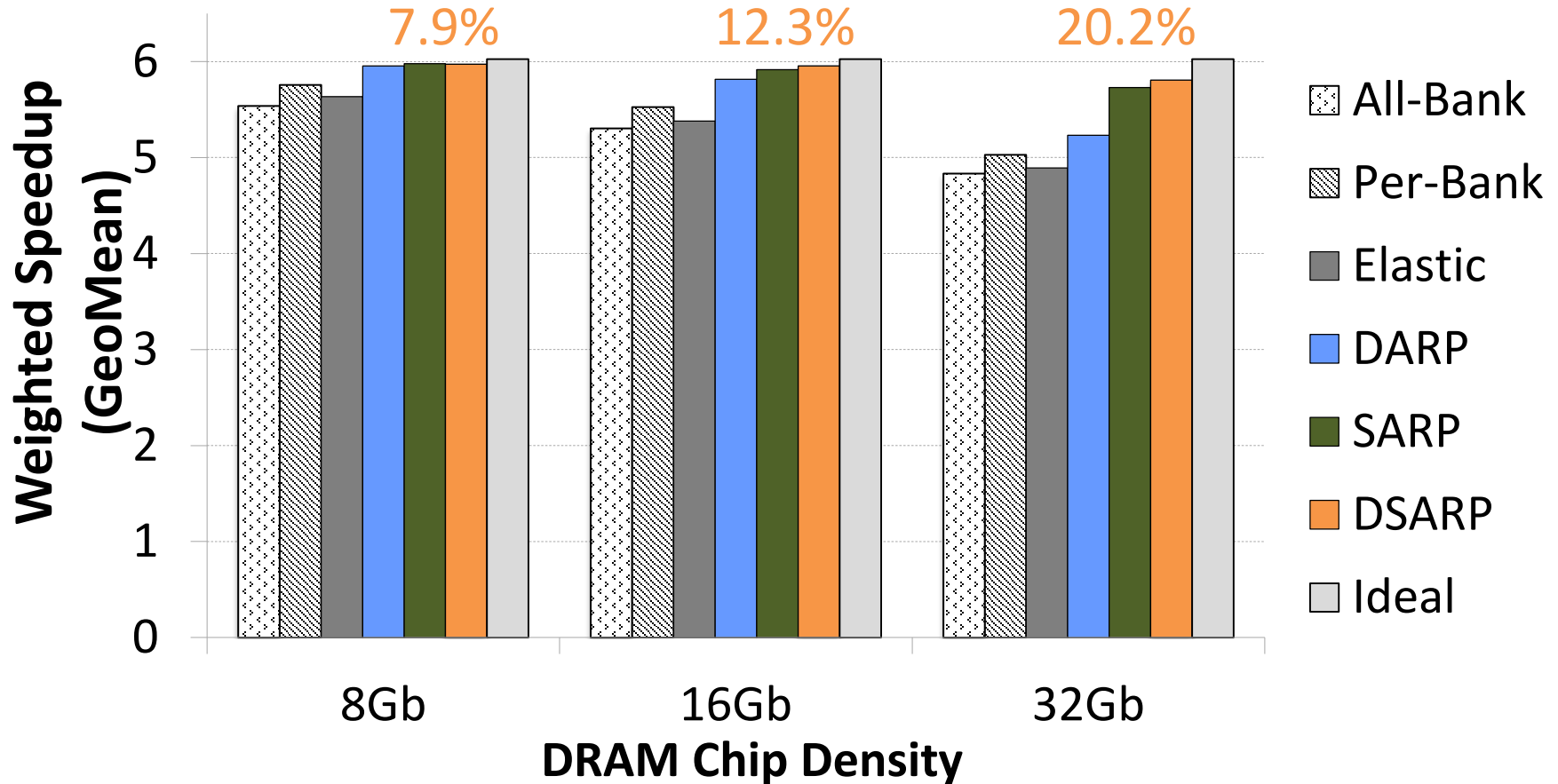


- **100 workloads**: SPEC CPU2006, STREAM, TPC-C/H, random access
- **System performance metric**: *Weighted speedup*

Comparison Points

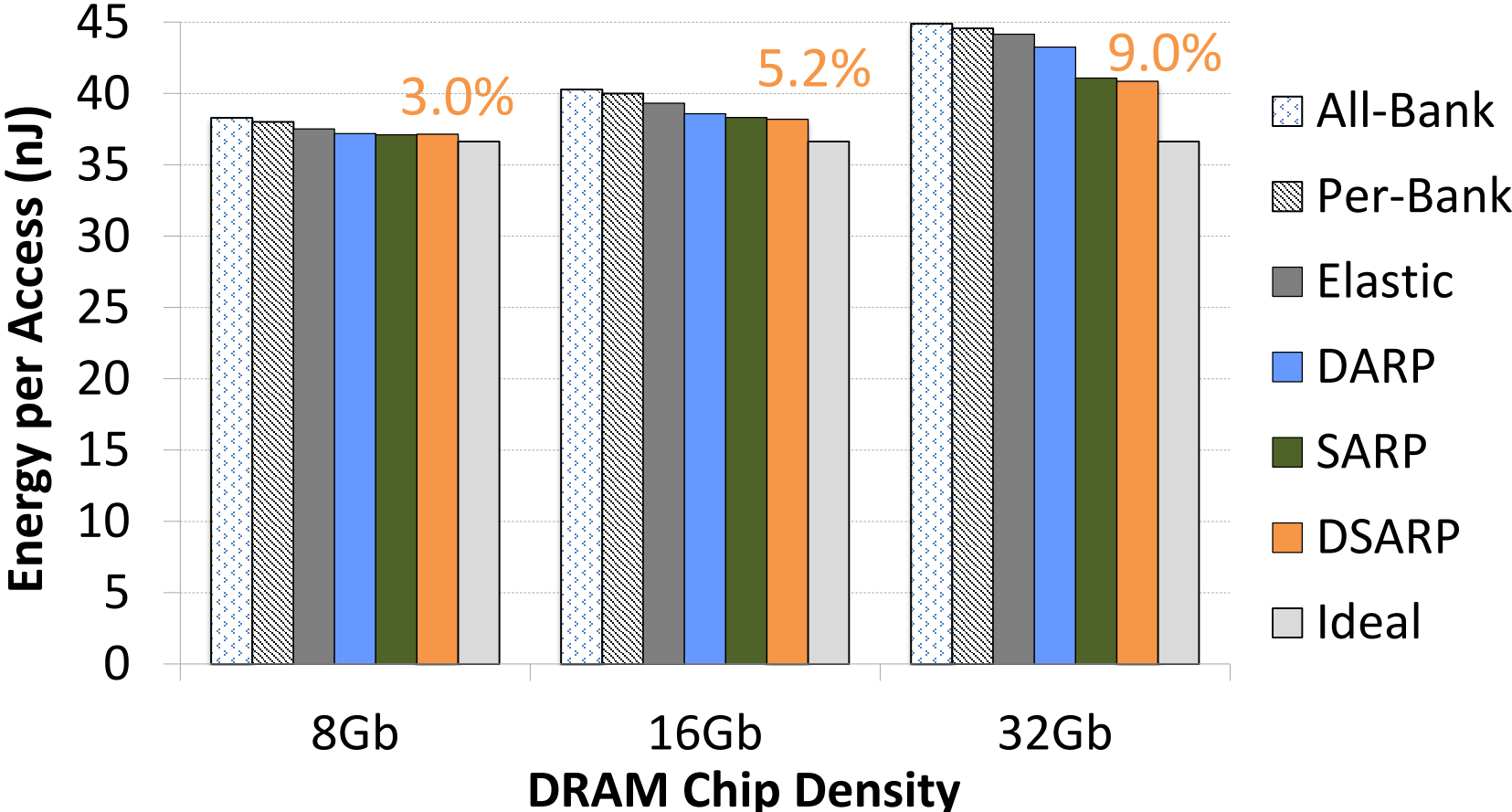
- **All-bank refresh** [DDR3, LPDDR3, ...]
- **Per-bank refresh** [LPDDR3]
- **Elastic refresh** [Stuecheli et al., MICRO '10]:
 - Postpones refreshes by a time delay based on the predicted rank idle time to avoid interference on memory requests
 - Proposed to schedule all-bank refreshes without exploiting per-bank refreshes
 - Cannot parallelize refreshes and accesses within a rank
- **Ideal (no refresh)**

System Performance



2. Consistent system performance improvement across DRAM densities (within 0.9%, 1.2%, and 3.8% of ideal)

Energy Efficiency



Consistent reduction on energy consumption