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

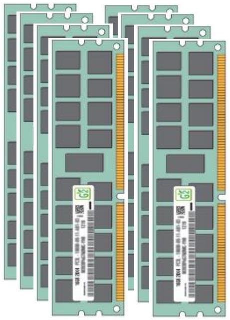
- Motivation
- Characterizing application memory error tolerance
- Key observations
  - Observation 1: Memory error tolerance varies across applications and within an application
  - Observation 2: Data can be recovered by software
- Heterogeneous-Reliability Memory (HRM)
- Evaluation

Outline

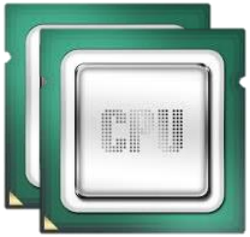
- Motivation
- Characterizing application memory error tolerance
- Key observations
  - Observation 1: Memory error tolerance varies across applications and within an application
  - Observation 2: Data can be recovered by software
- Heterogeneous-Reliability Memory (HRM)
- Evaluation

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]*



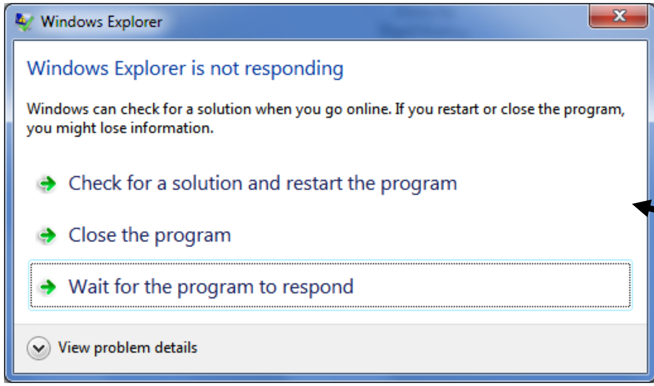
128GB Memory cost
~$140(per 16GB)×8
= ~$1120 *



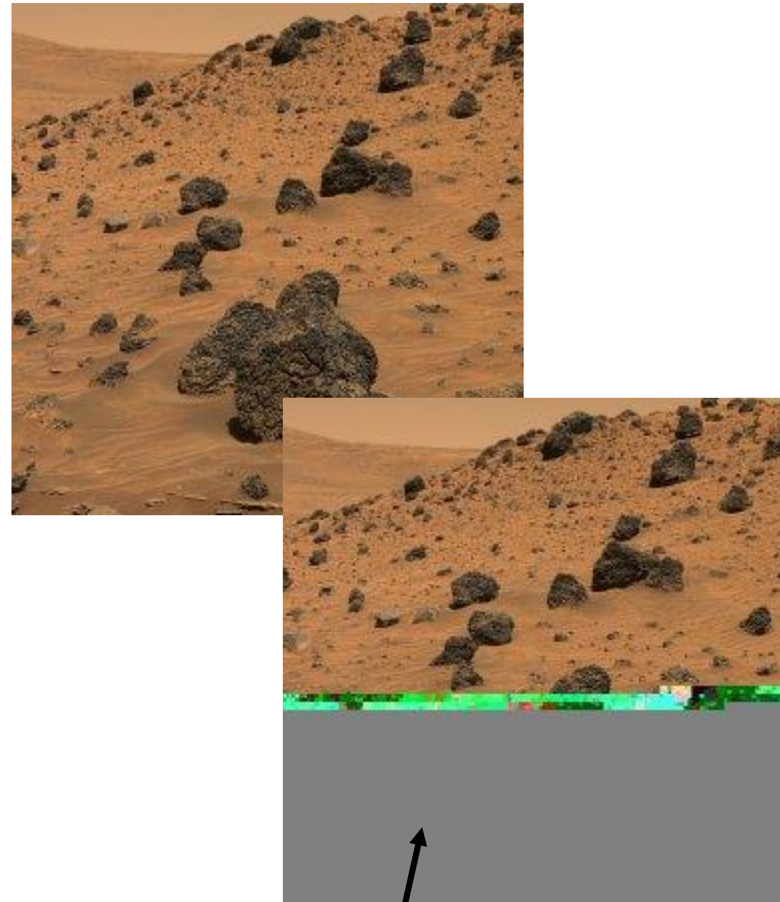
2 CPUs cost
~$500(per CPU)×2
= ~$1000 *

* Numbers in the year of 2014

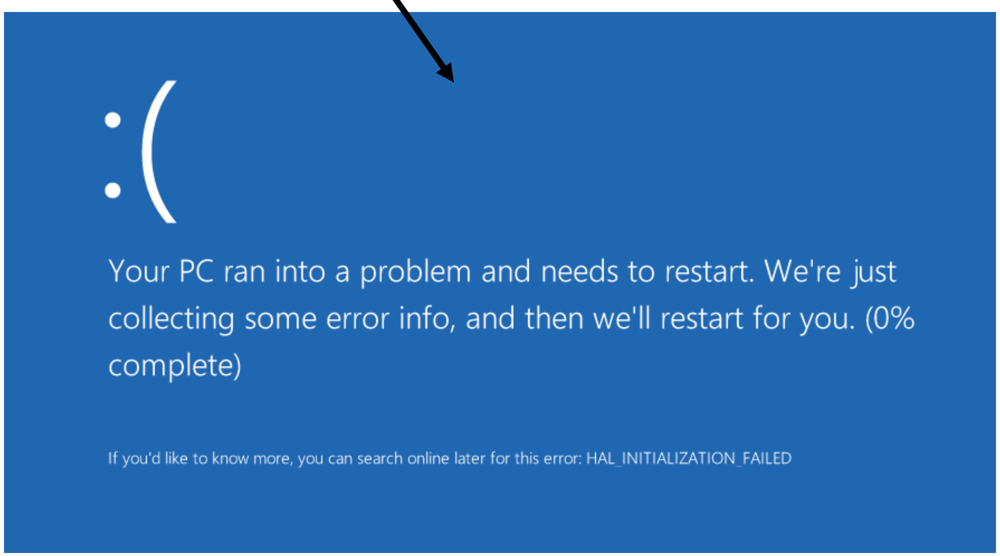
Memory Reliability is Important



System/app hang or slowdown



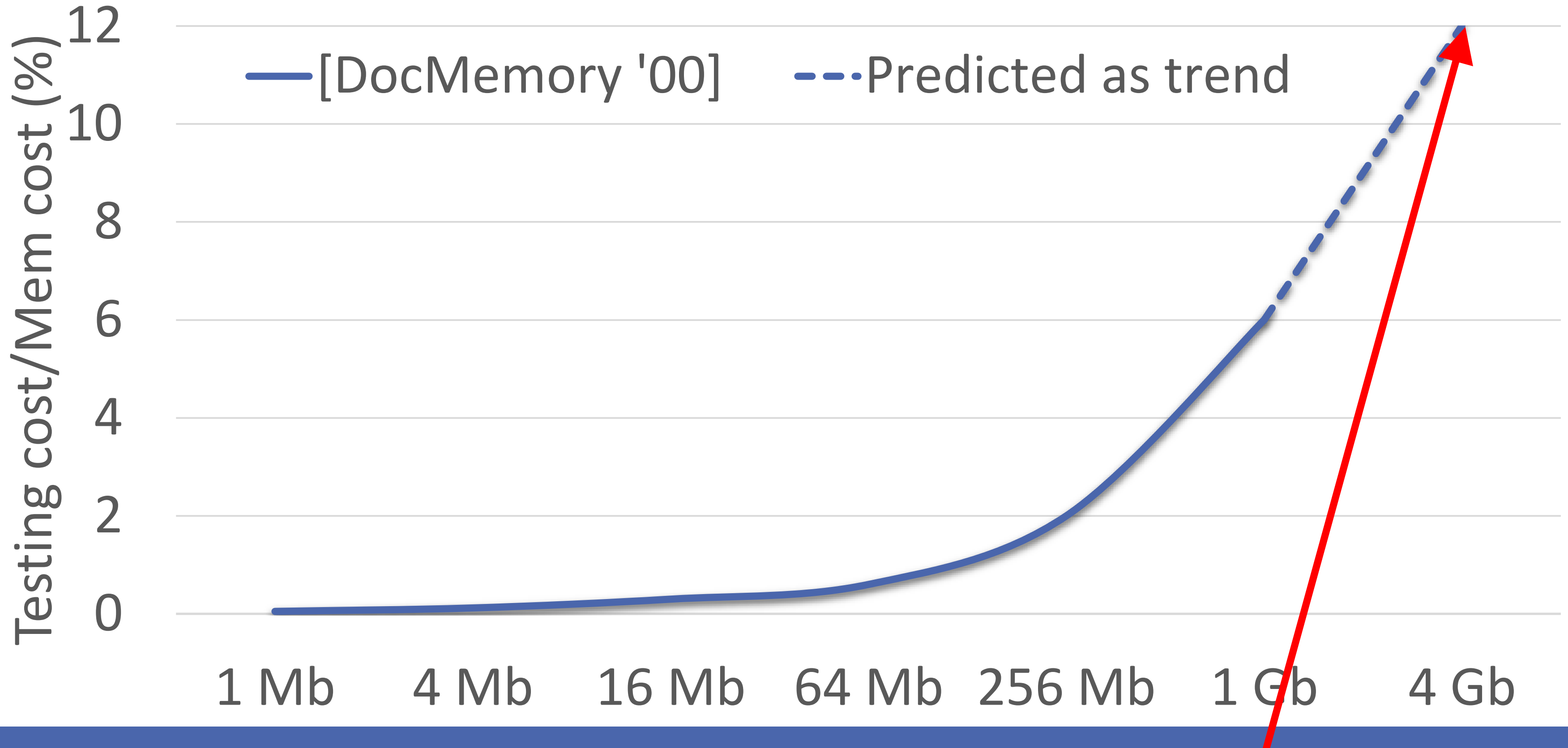
System/app crash



Silent data corruption or incorrect app output

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*

Increasing strength

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

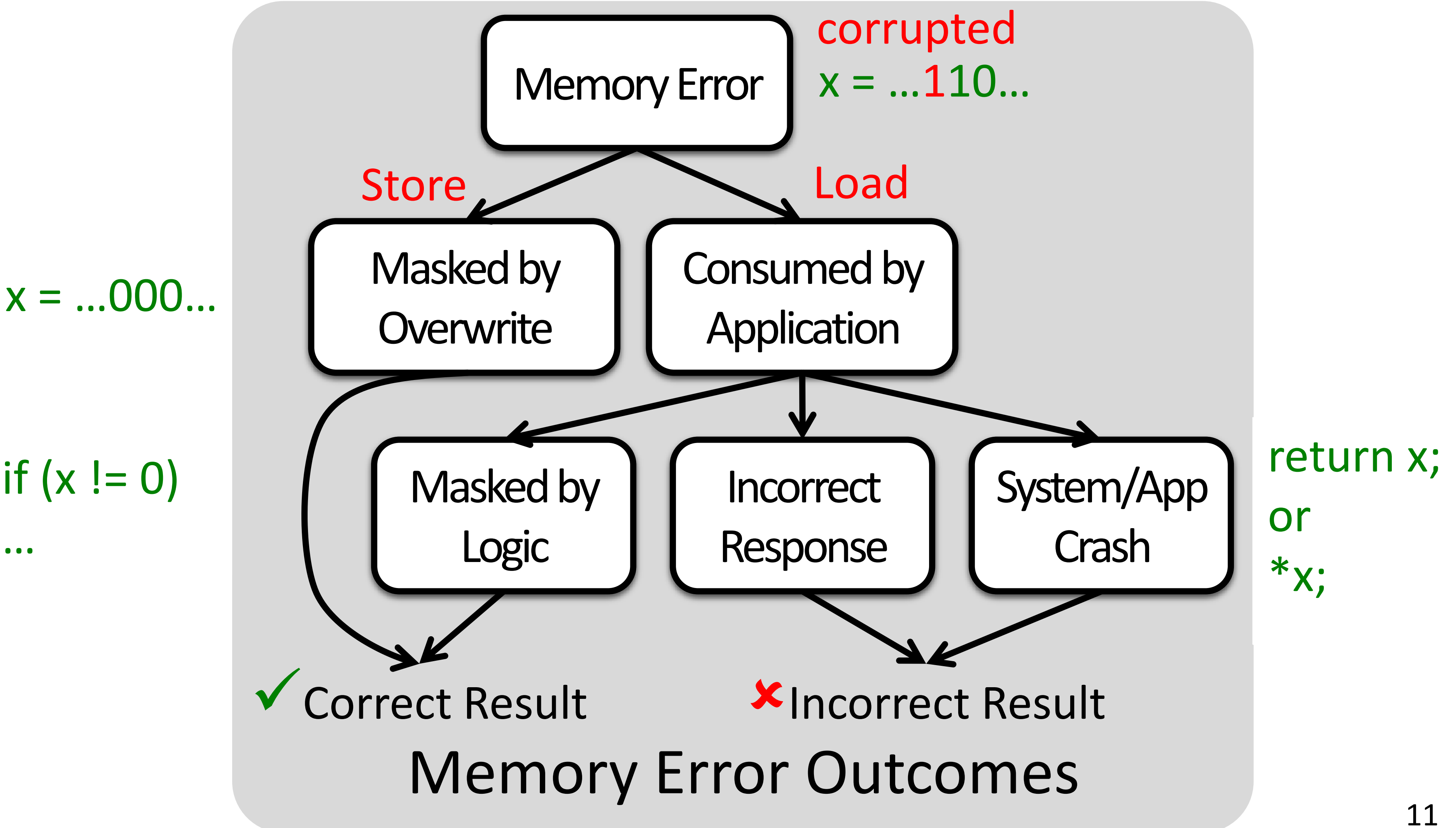
Goal: Design a new cost-efficient memory system that flexibly matches *memory reliability* with *application memory error tolerance*

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*

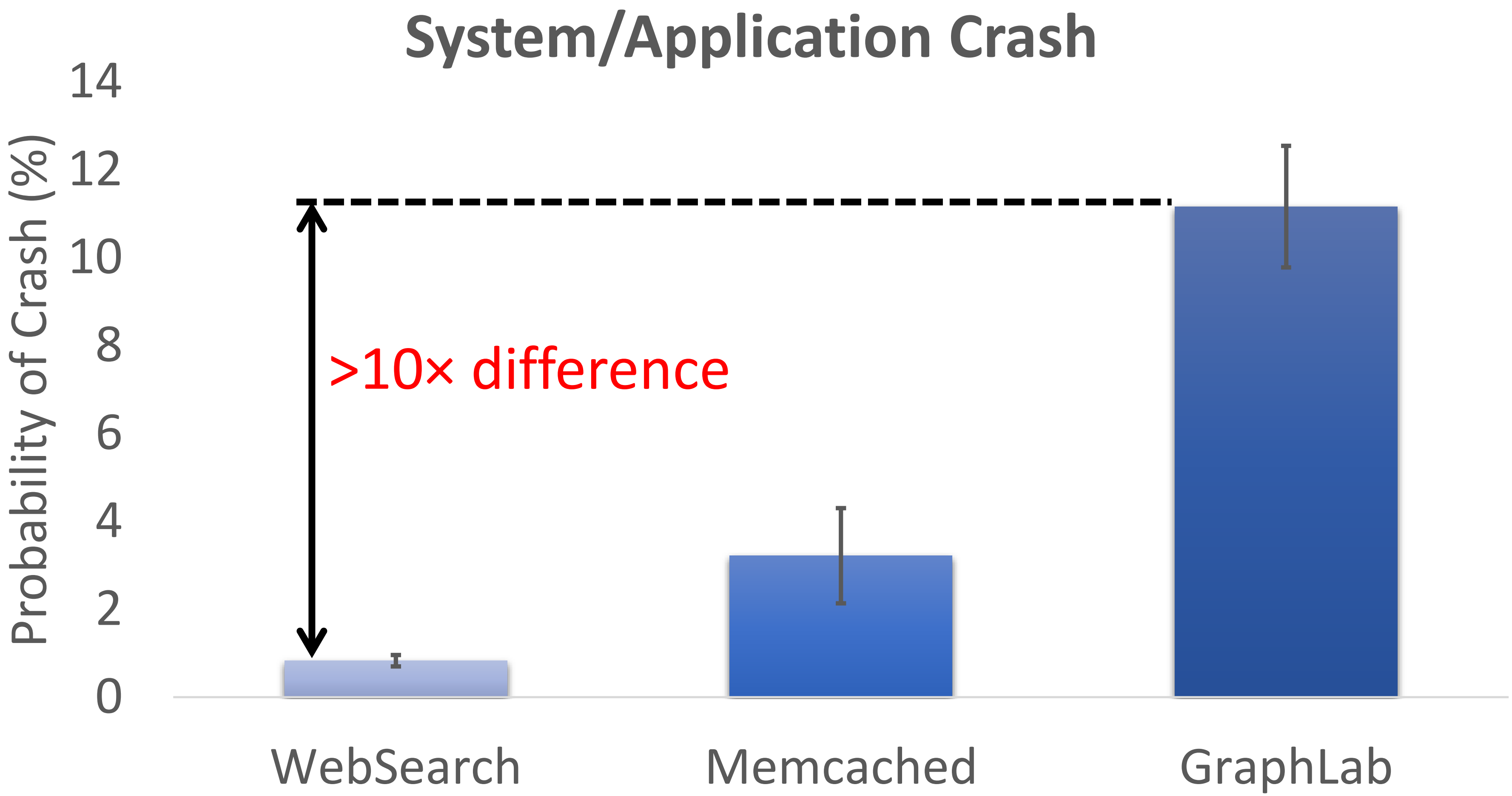
Application	WebSearch	Memcached	GraphLab
Memory footprint	46 GB	35 GB	4 GB

- *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

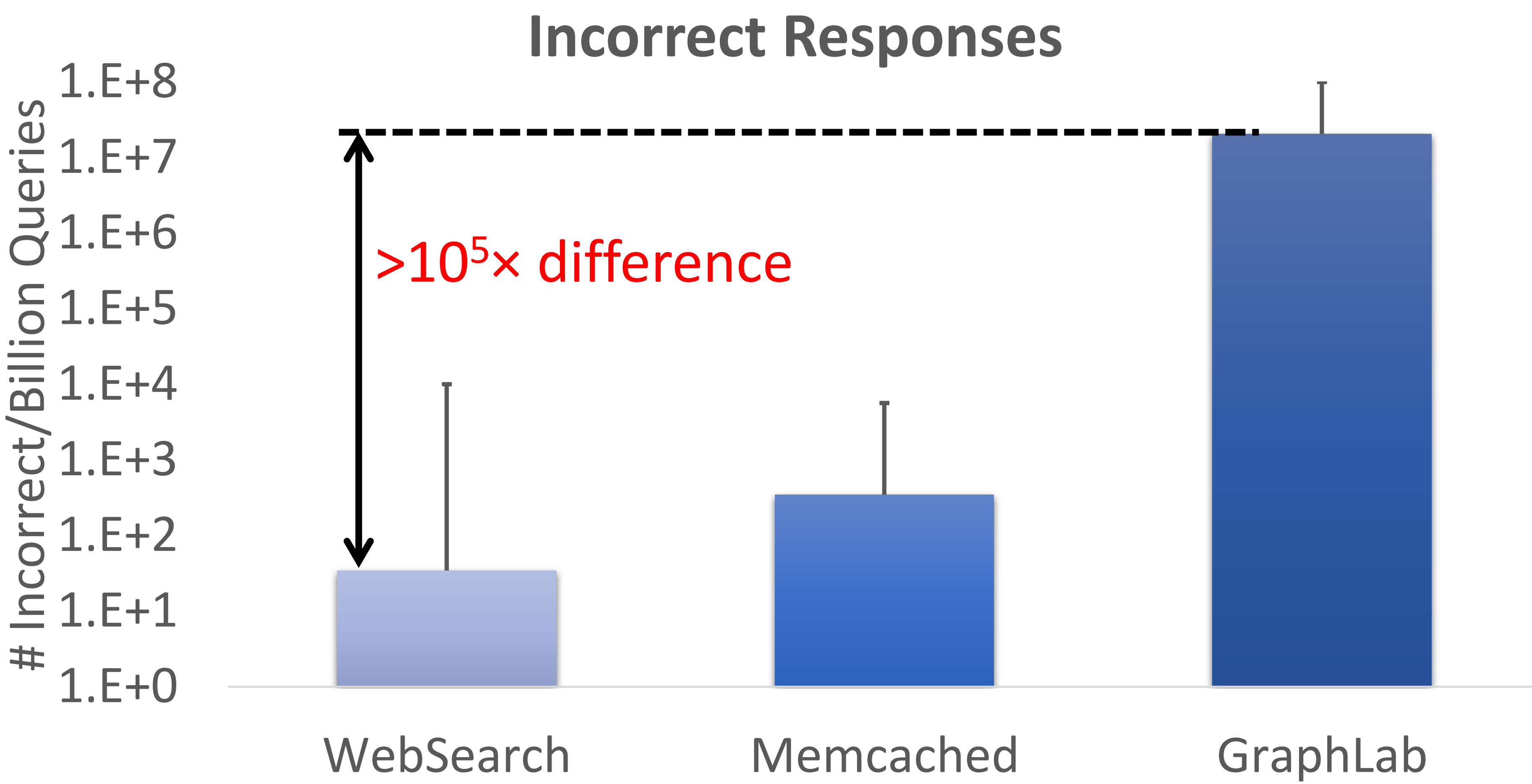
Observation 1: Memory Error Tolerance Varies Across Applications



Showing results for single-bit soft errors

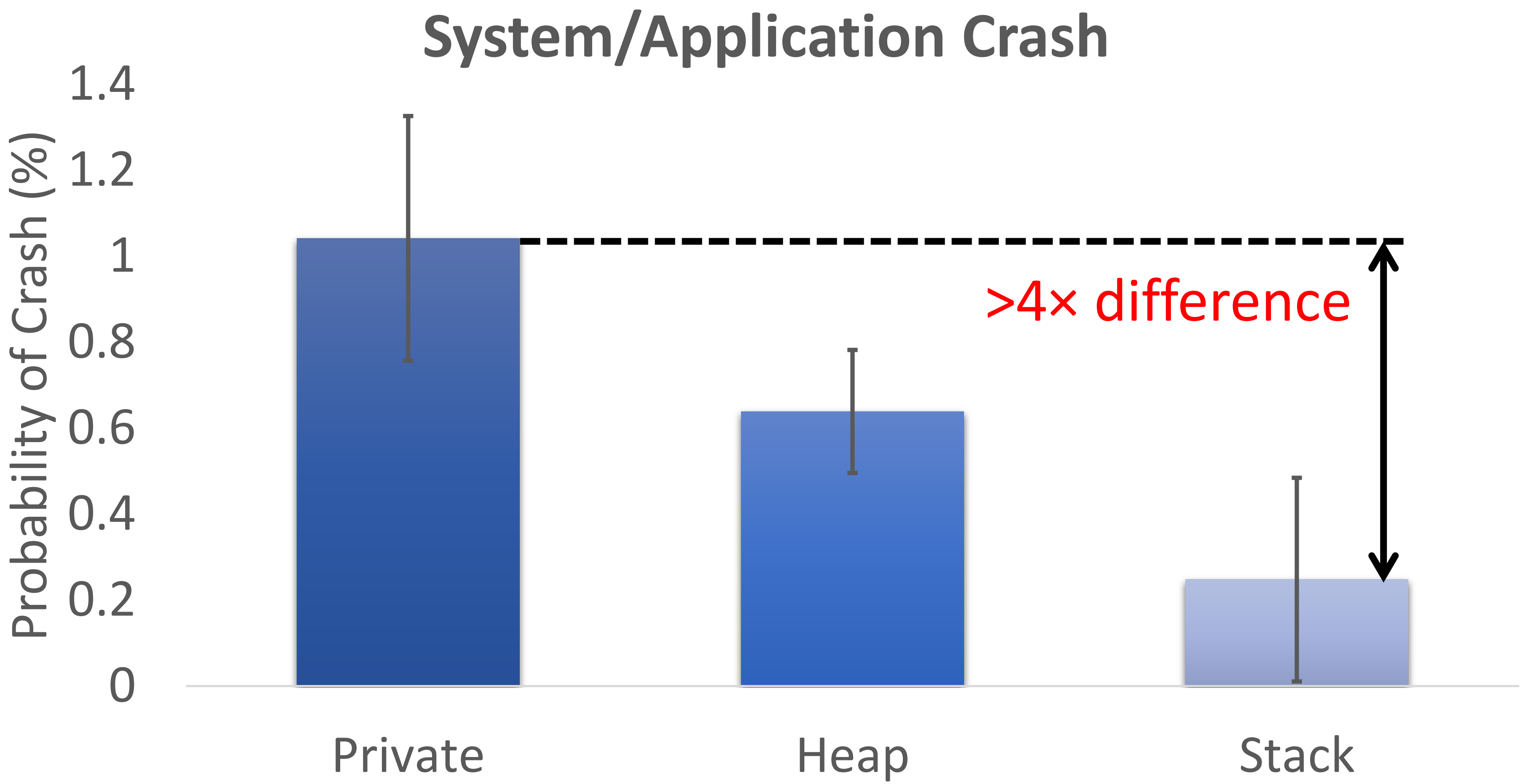
Results for other memory error types can be found in the paper with similar conclusion

Observation 1: Memory Error Tolerance Varies Across Applications



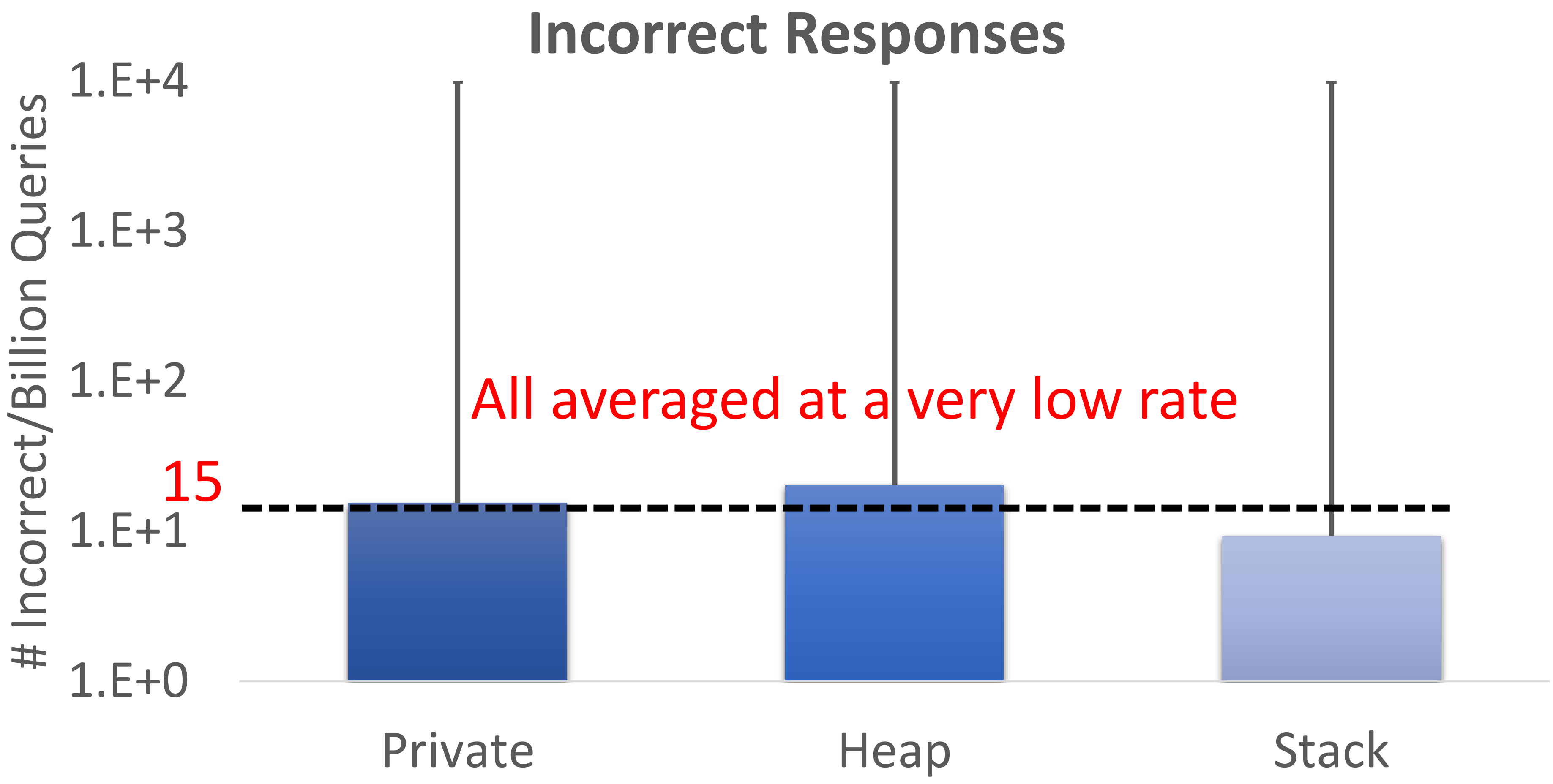
Showing results for single-bit soft errors
Results for other memory error types can be found in the paper

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



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



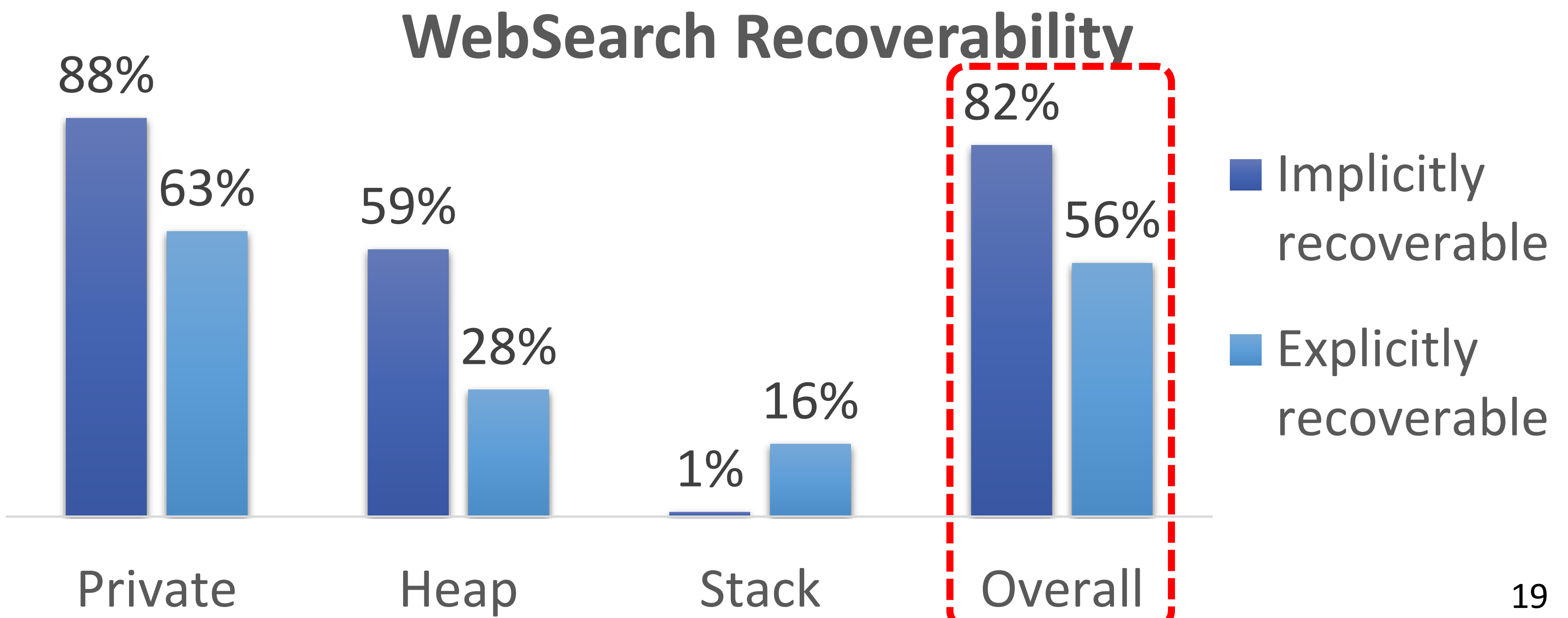
Showing results for WebSearch
Results for other workloads can be found in the paper

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

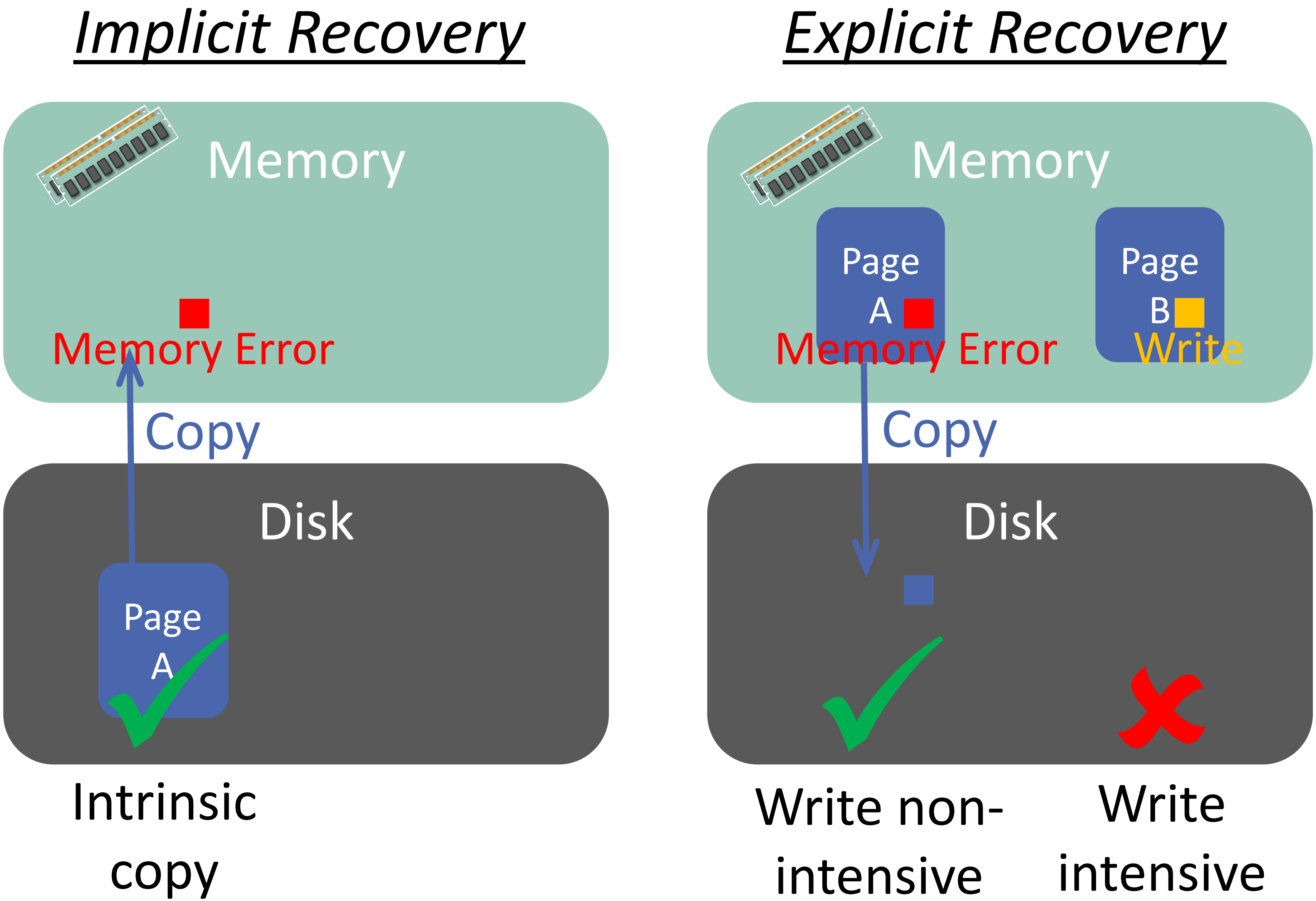
- ECC protected
- Well-tested chips

Low-cost memory

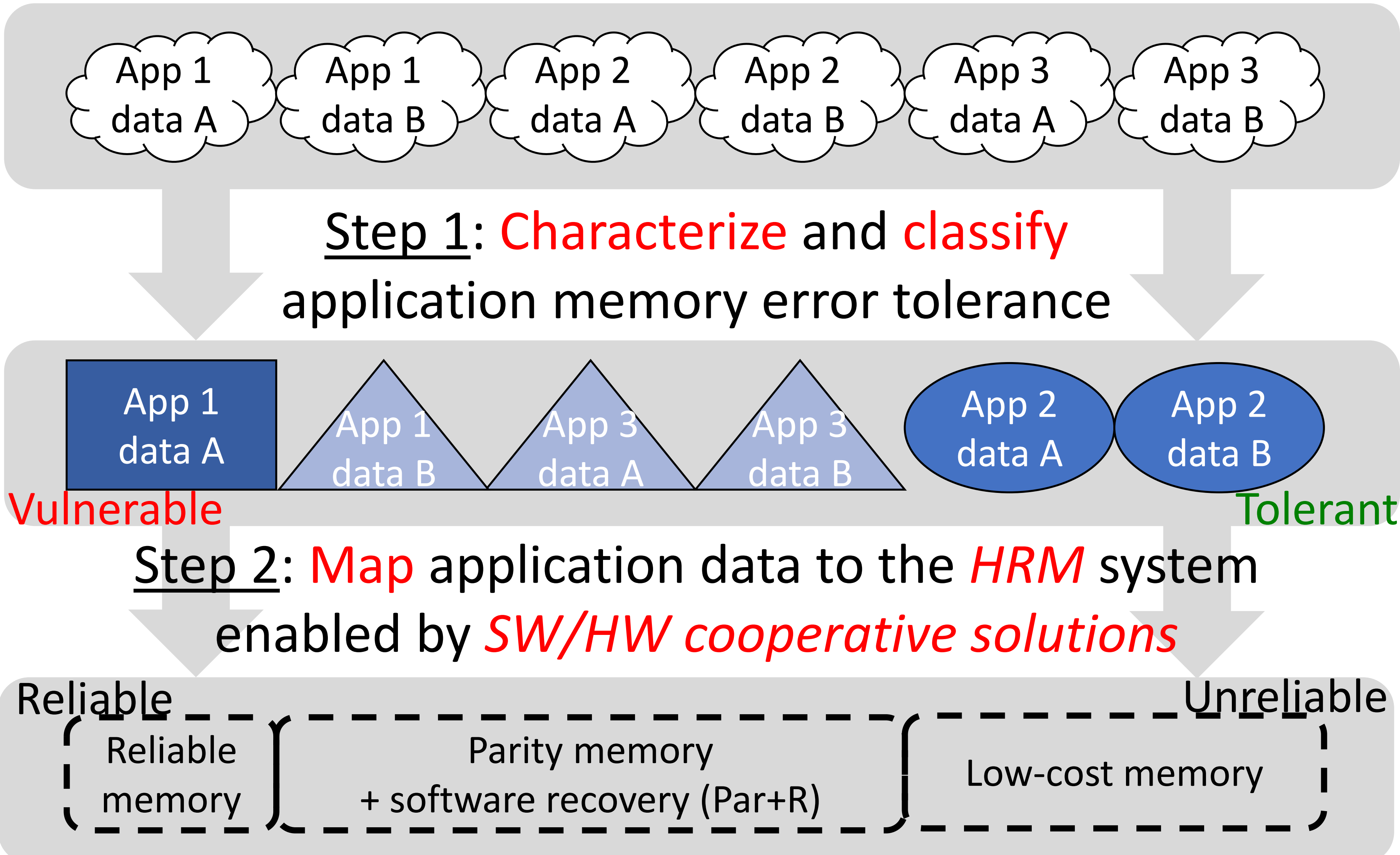
- NoECC or Parity
- Less-tested chips

Heterogeneous-**R**eliability **M**emory

Par+R: Parity Detection + Software Recovery



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

Baseline systems HRM systems

Design Parameters

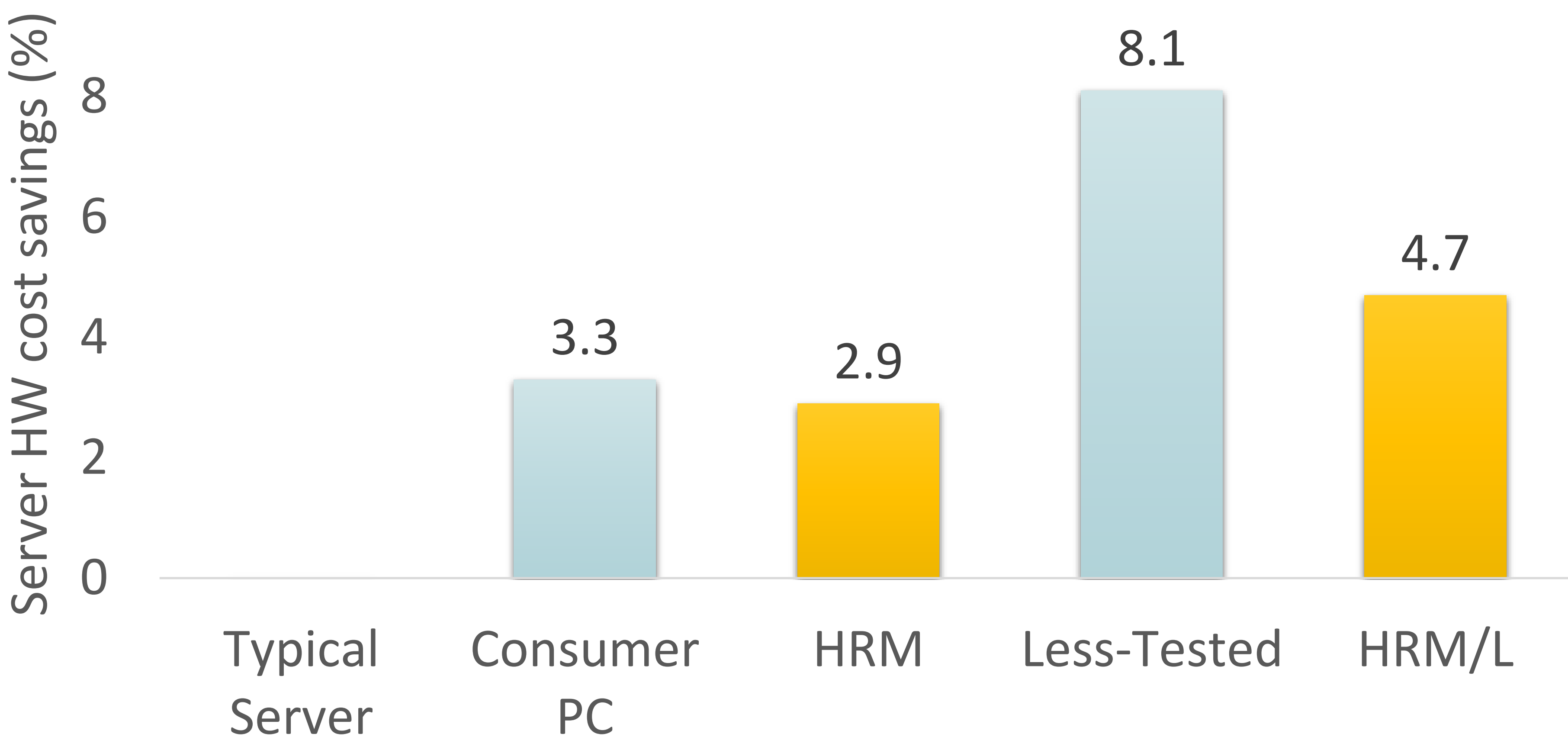
DRAM/server HW cost [Kozyrakis ‘10]	30%
NoECC memory cost savings	11.1%
Parity memory cost savings	9.7%
Less-tested memory cost savings	18%±12%
Crash recovery time	10 mins
Par+R flush threshold	5 mins
Errors/server/month [Schroeder ‘09]	2000
Target single server availability	99.90%

Evaluation Metrics

- *Cost*
  - Memory cost savings
  - Server HW cost savings
    (both compared with ***<u>Typical Server</u>***)

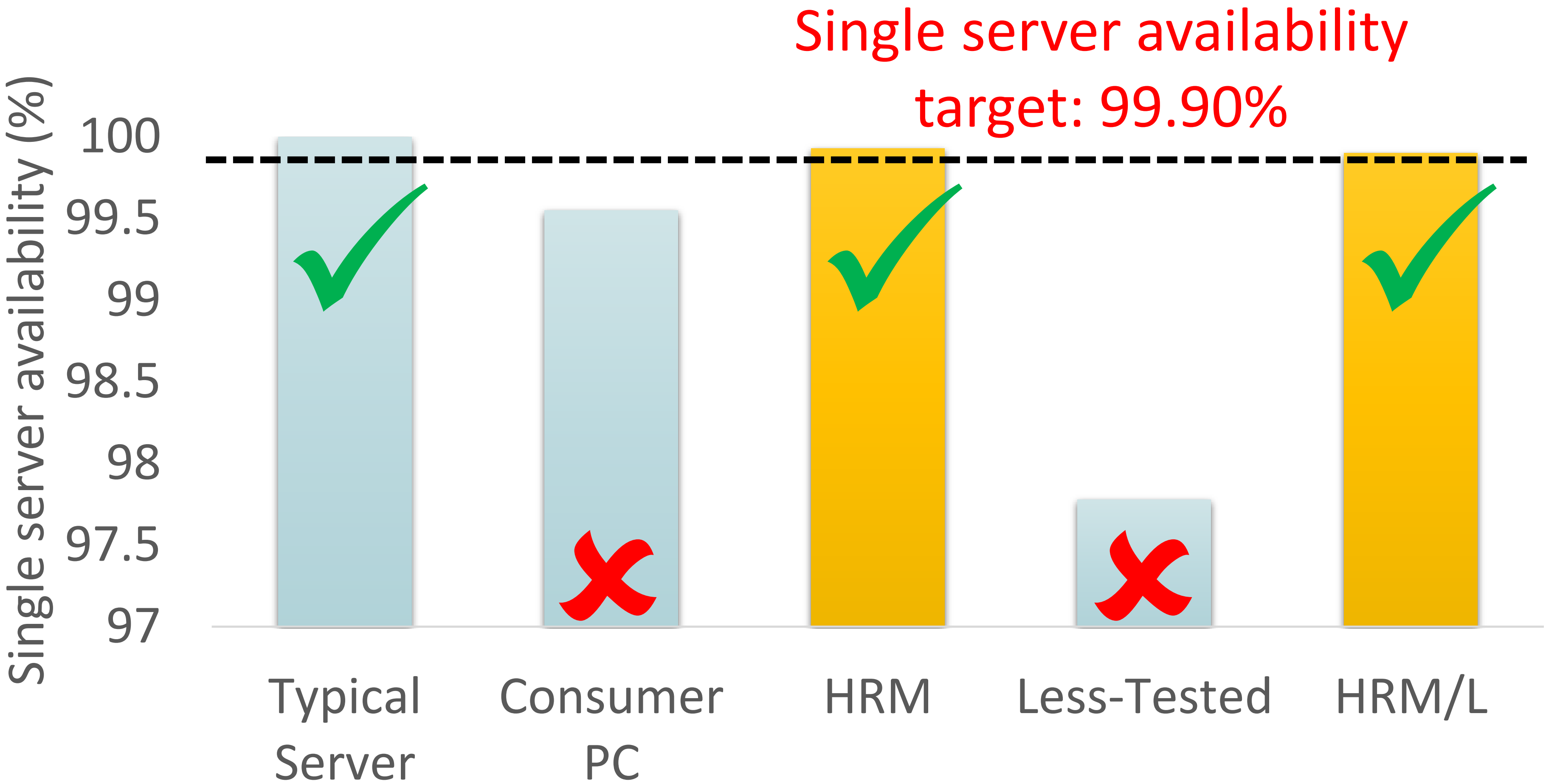
- *Reliability*
  - Crashes/server/month
  - Single server availability
  - # incorrect/million queries

Improving Server HW Cost Savings



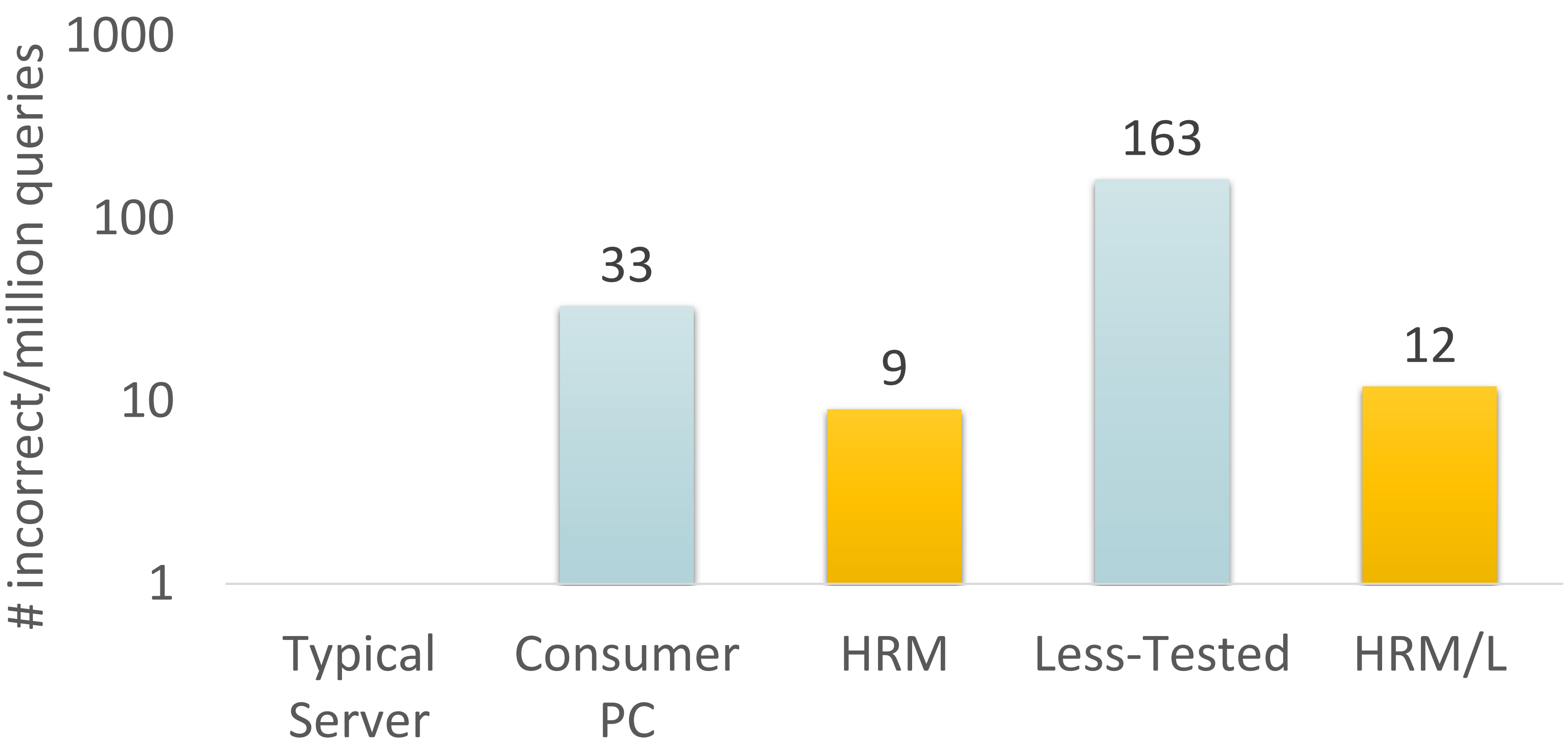
Reducing the use of memory error mitigation techniques in part of memory space can save noticeable amount of server HW cost

Achieving Target Availability



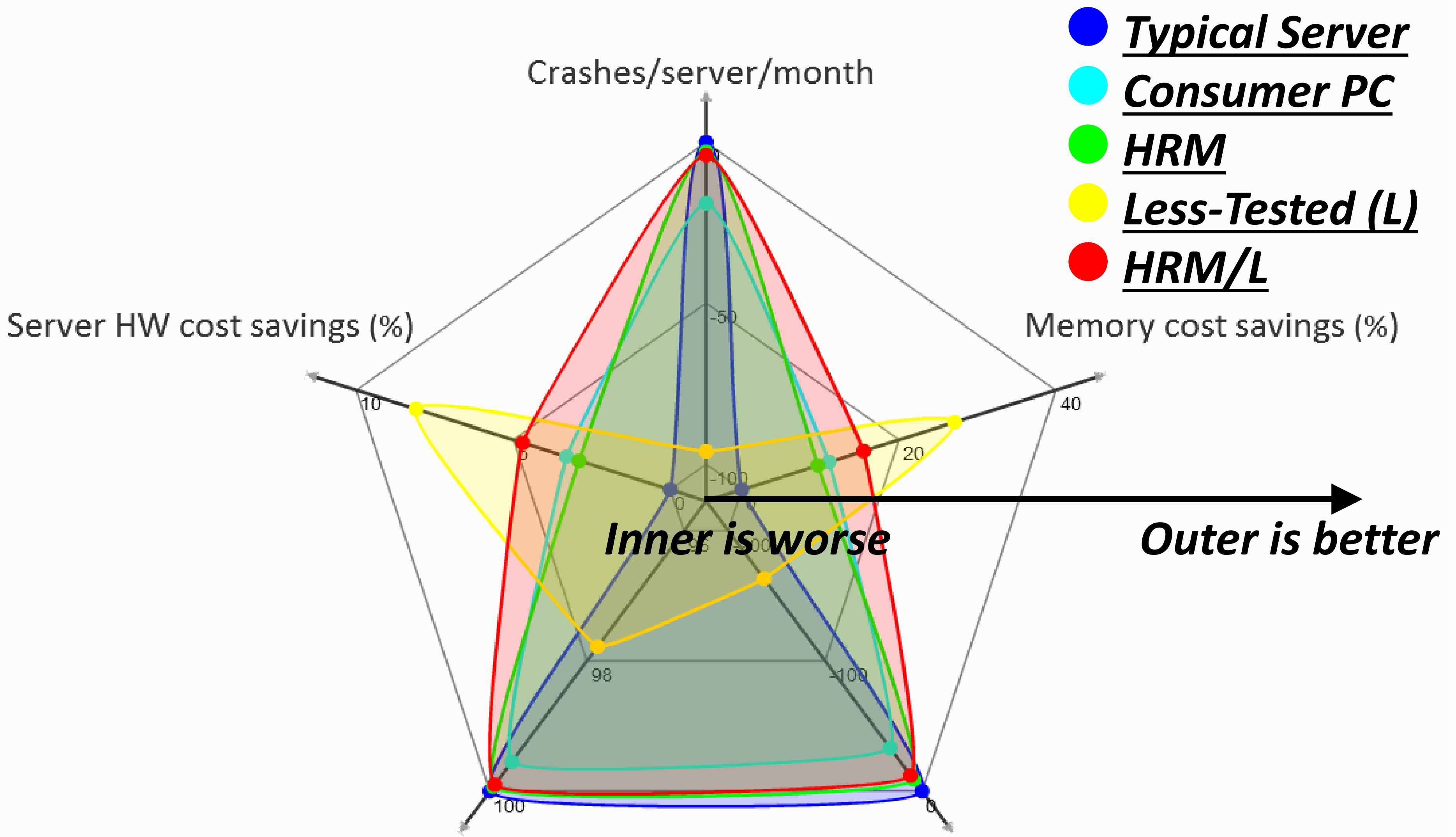
HRM systems are flexible to adjust and can achieve availability target

Achieving Acceptable Correctness



HRM systems can achieve acceptable correctness

Evaluation Results



Bigger area means better tradeoff

Other Results and Findings in the Paper

- *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

- *<u>Our Goal</u>: Reduce datacenter cost; meet availability target*

- *<u>Characterized</u> application-level memory error tolerance of 3 modern data-intensive workloads*

- *<u>Proposed</u> Heterogeneous-Reliability Memory (HRM)*
  - Store error-tolerant data in less-reliable lower-cost memory
  - Store error-vulnerable data in more-reliable memory

- *<u>Evaluated</u> example HRM systems*
  - Reduce server hardware cost by 4.7 %
  - Achieve single-server availability target 99.90 %

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

Why use a software debugger?

- *Speed*
  - Our workloads are relatively long running
    - *WebSearch – 30 minutes*
    - *Memcached – 10 minutes*
    - *GraphLab – 10 minutes*
  - Our workloads have large memory footprint
    - *WebSearch – 46 GB*
    - *Memcached – 35 GB*
    - *GraphLab – 4 GB*

What are the workload properties?

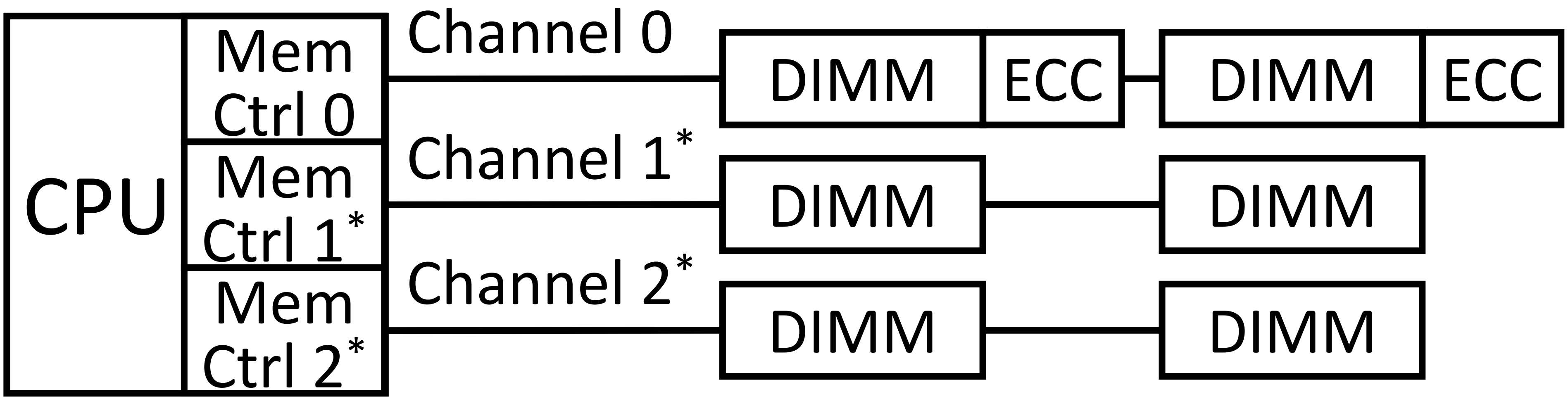
- *WebSearch*
  - Repeat a real-world trace of 200,000 queries, with 400 qps
  - Correctness: Top 4 most relevant documents
    - *Document id*
    - *Relevance and popularity*
- *Memcached*
  - 30 GB of twitter dataset
  - Synthetic client workload, at 5,000 rps
  - 90% read requests and 10% write requests
- *GraphLab*
  - 11 million twitter users' following relations, 1.3 GB dataset
  - TunkRank algorithm
  - Correctness: 100 most influential users and their scores

How many errors are injected to each application and each memory region?

- *WebSearch – 20,576*
- *Memcached – 983*
- *GraphLab – 2,159*
- *Errors injected to each memory region is proportional to their sizes*

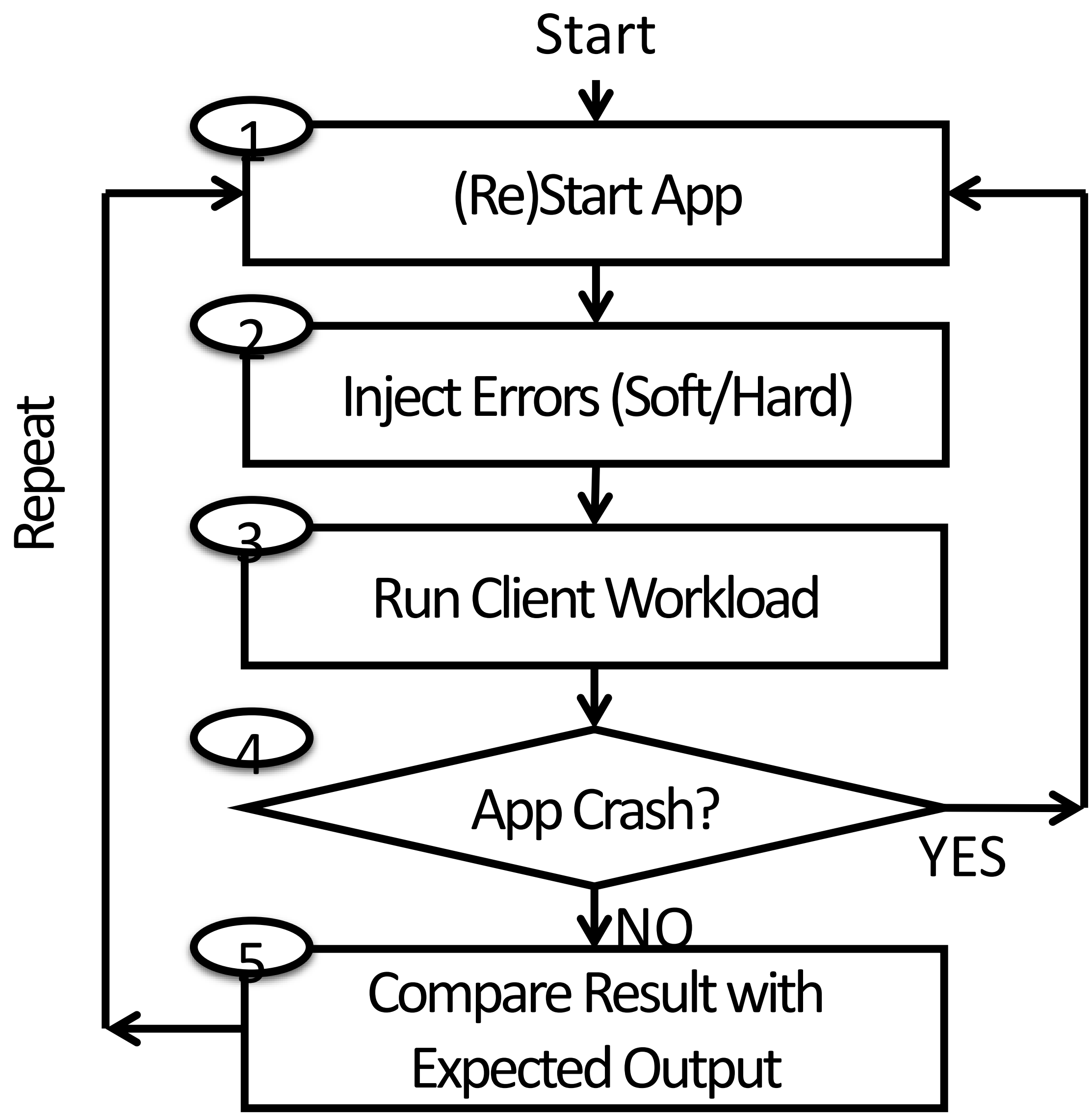
Application	Private	Heap	Stack	Total
WebSearch	36 GB	9 GB	60 MB	46 GB
Memcached	N/A	35 GB	132 KB	35 GB
GraphLab	N/A	4 GB	132 KB	4 GB

Does HRM require HW changes



* Memory controller/Channel without ECC support

What is the injection/monitoring process?



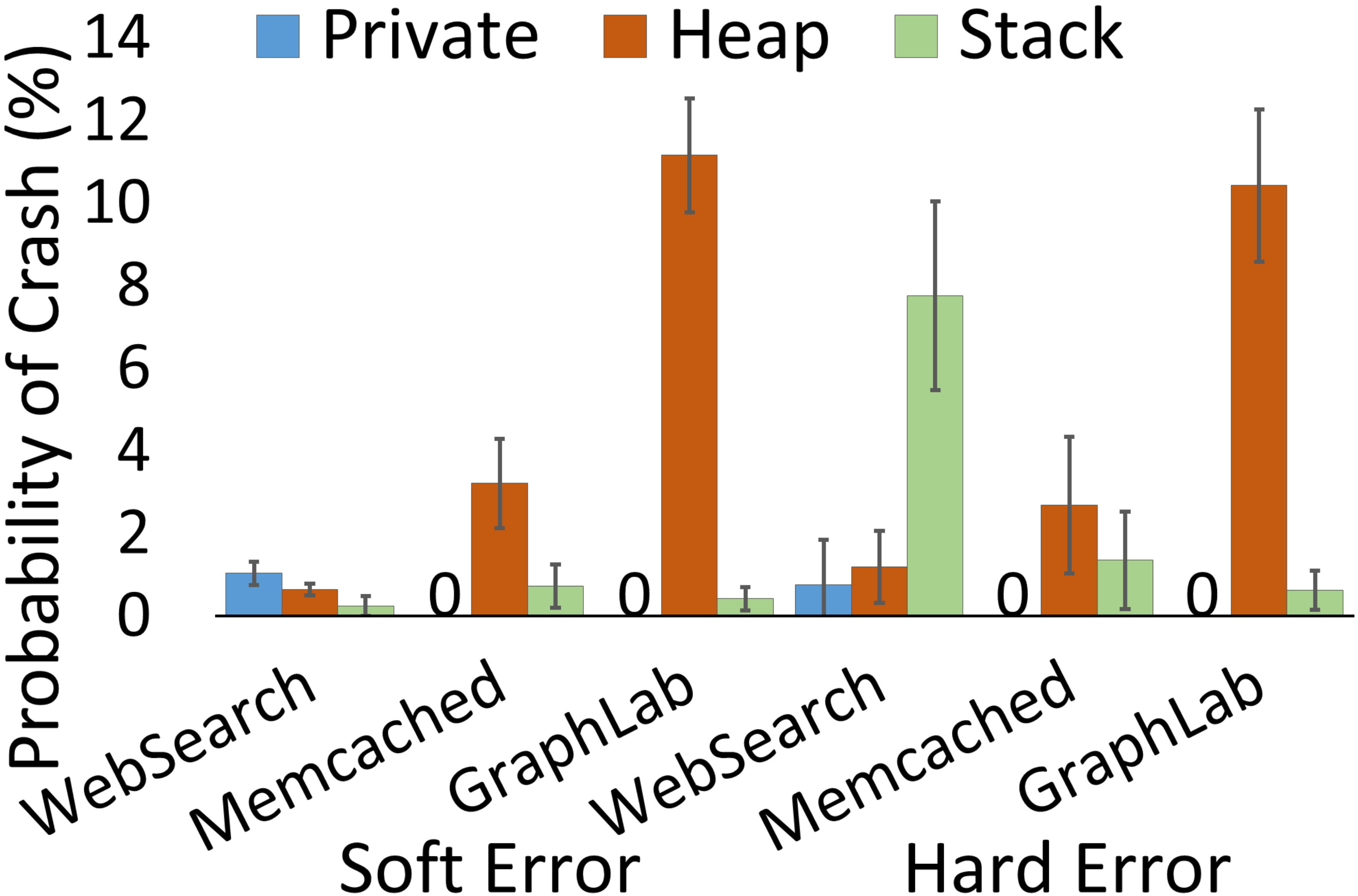
Comparison with previous works?

- *Virtualized and flexible ECC [Yoon '10]*
  - Requires changes to the MMU in the processor
  - Performance overhead ~10% over NoECC

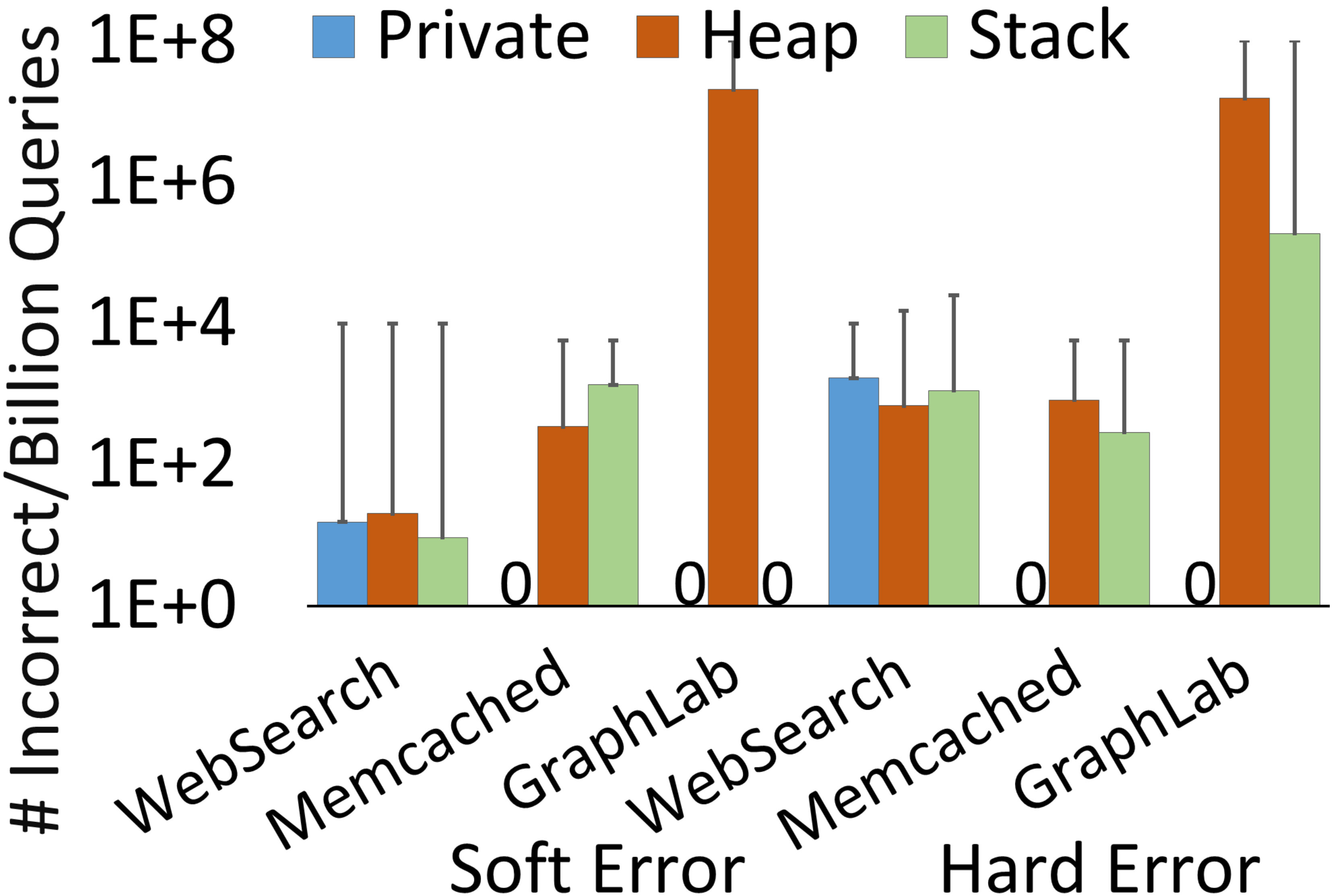
- *Our work: HRM*
  - Minimal changes to memory controller to enable different ECC on different channels
  - Low performance overhead
  - Enables the use of less-tested memory

Other Results

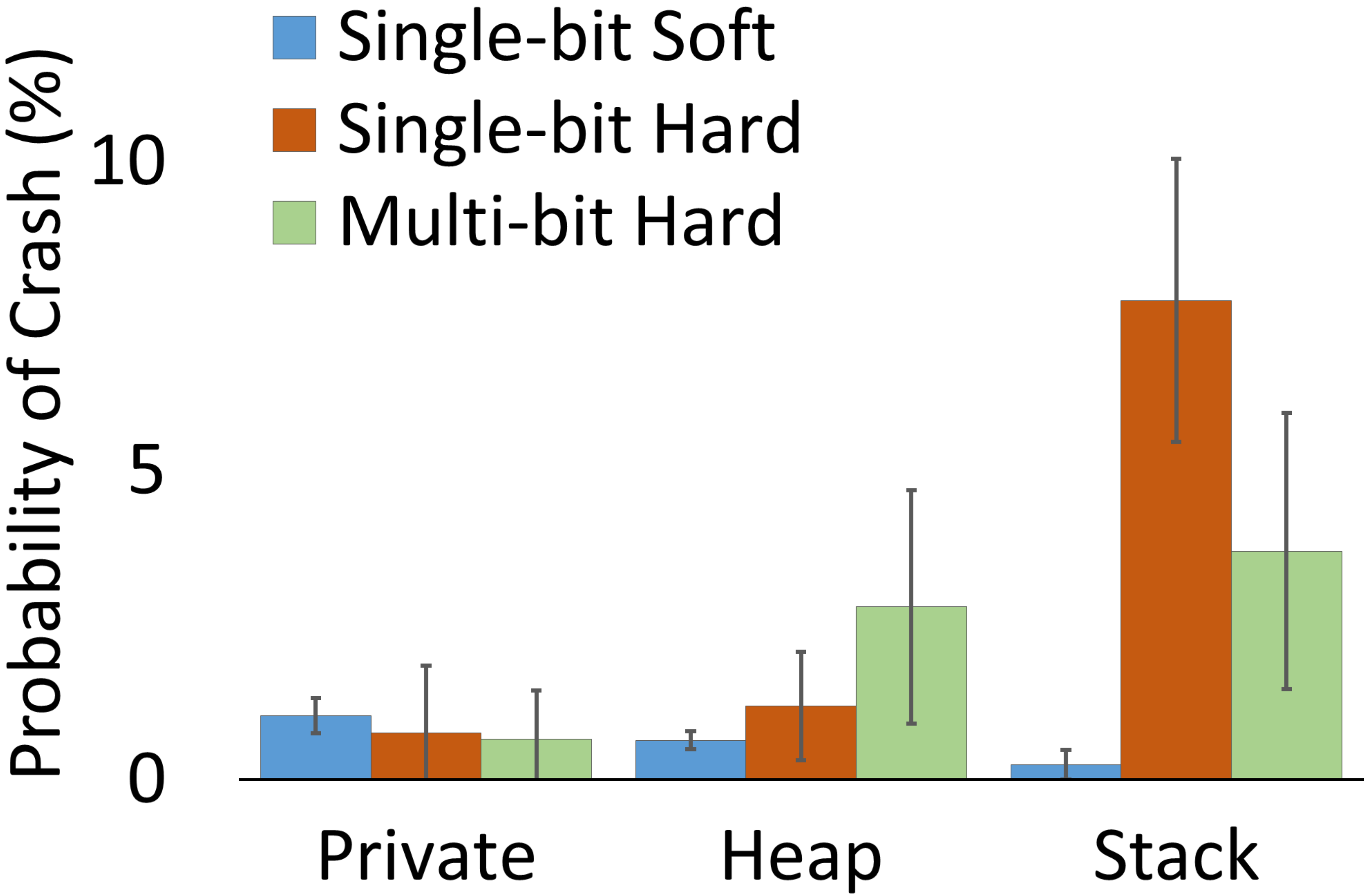
Variation within application



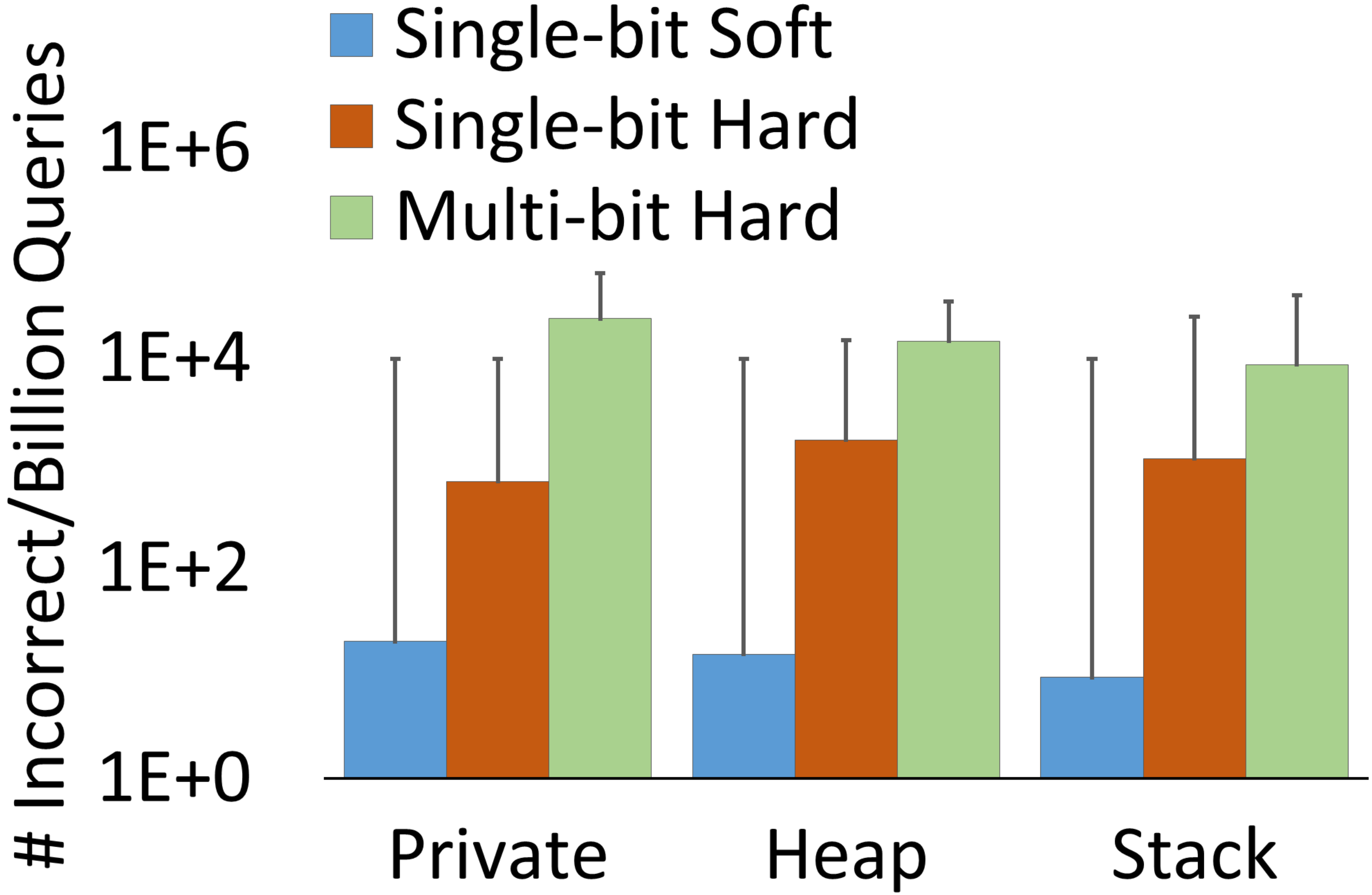
Variation within application



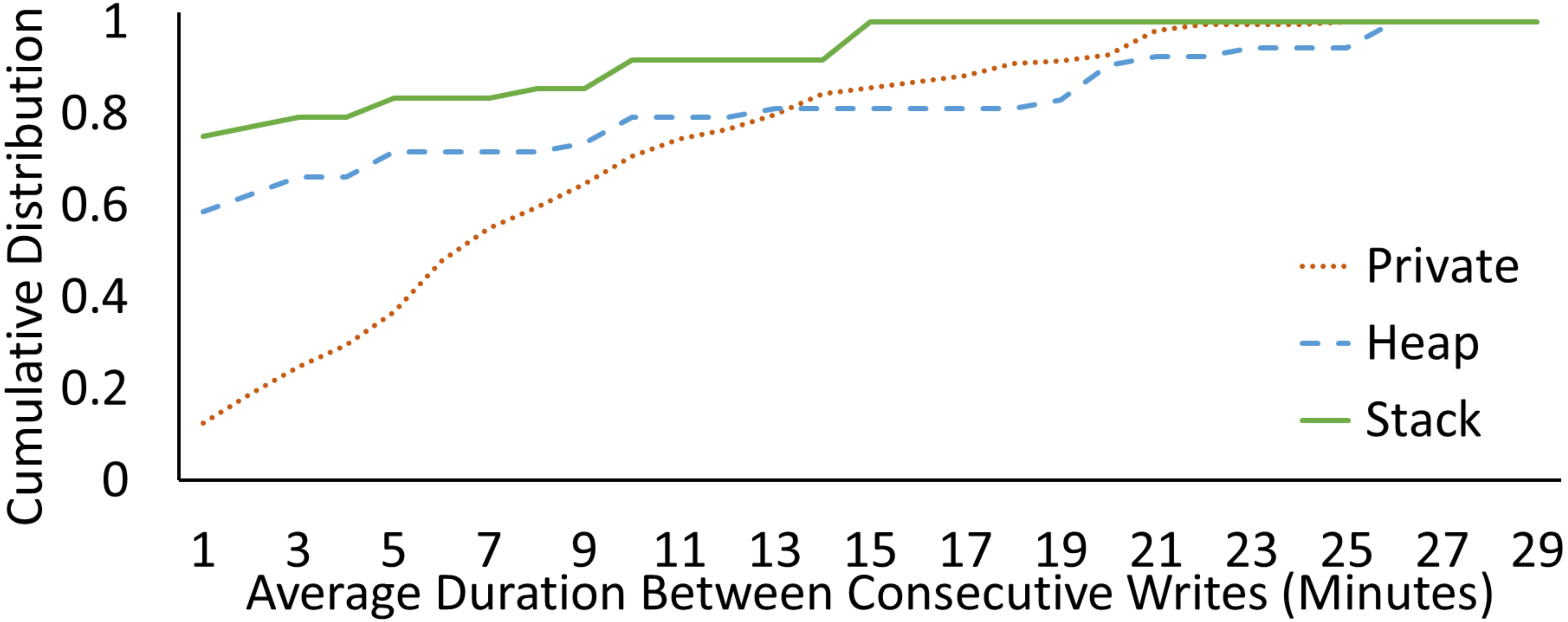
Other types of memory errors



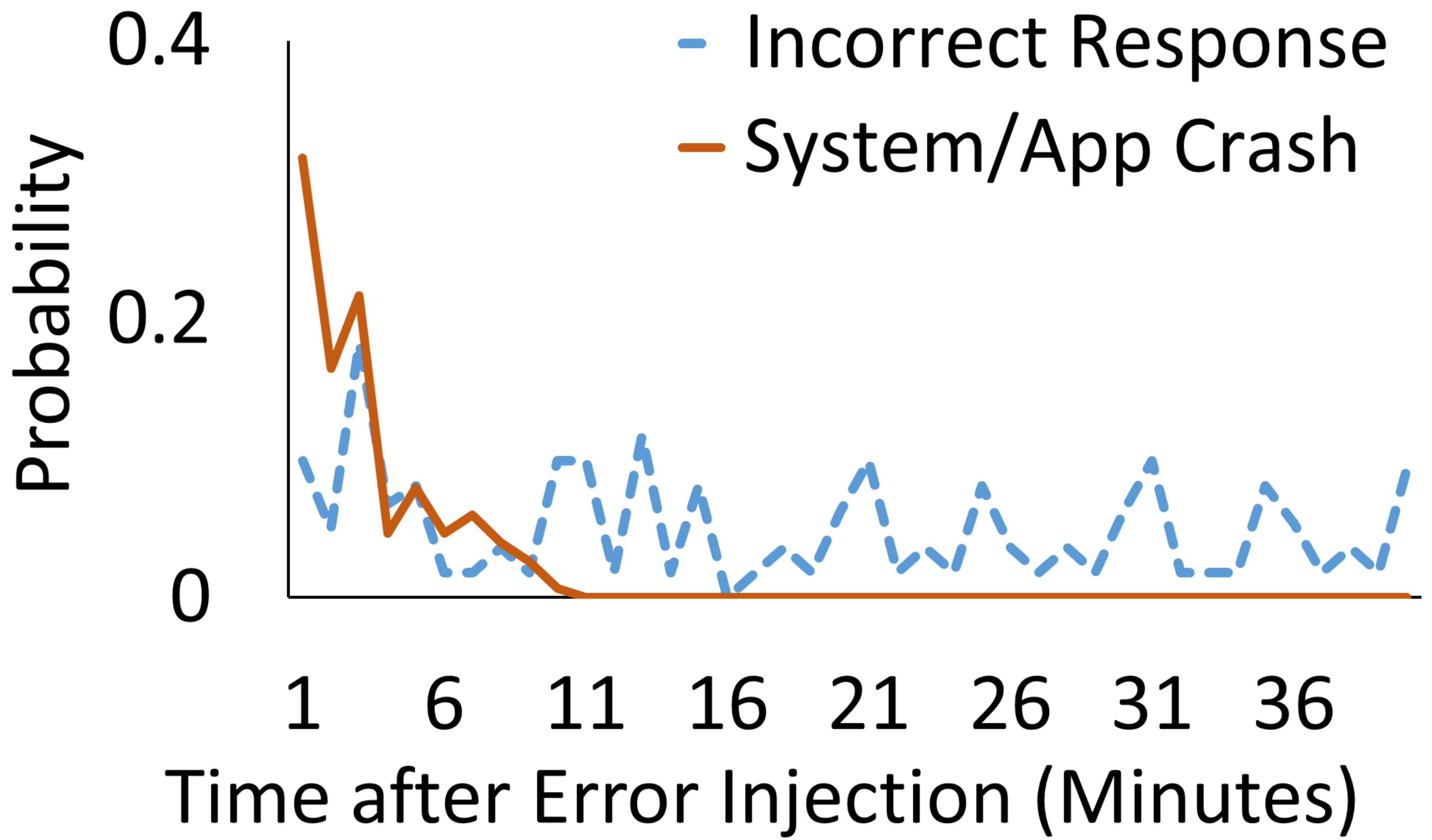
Other types of memory errors



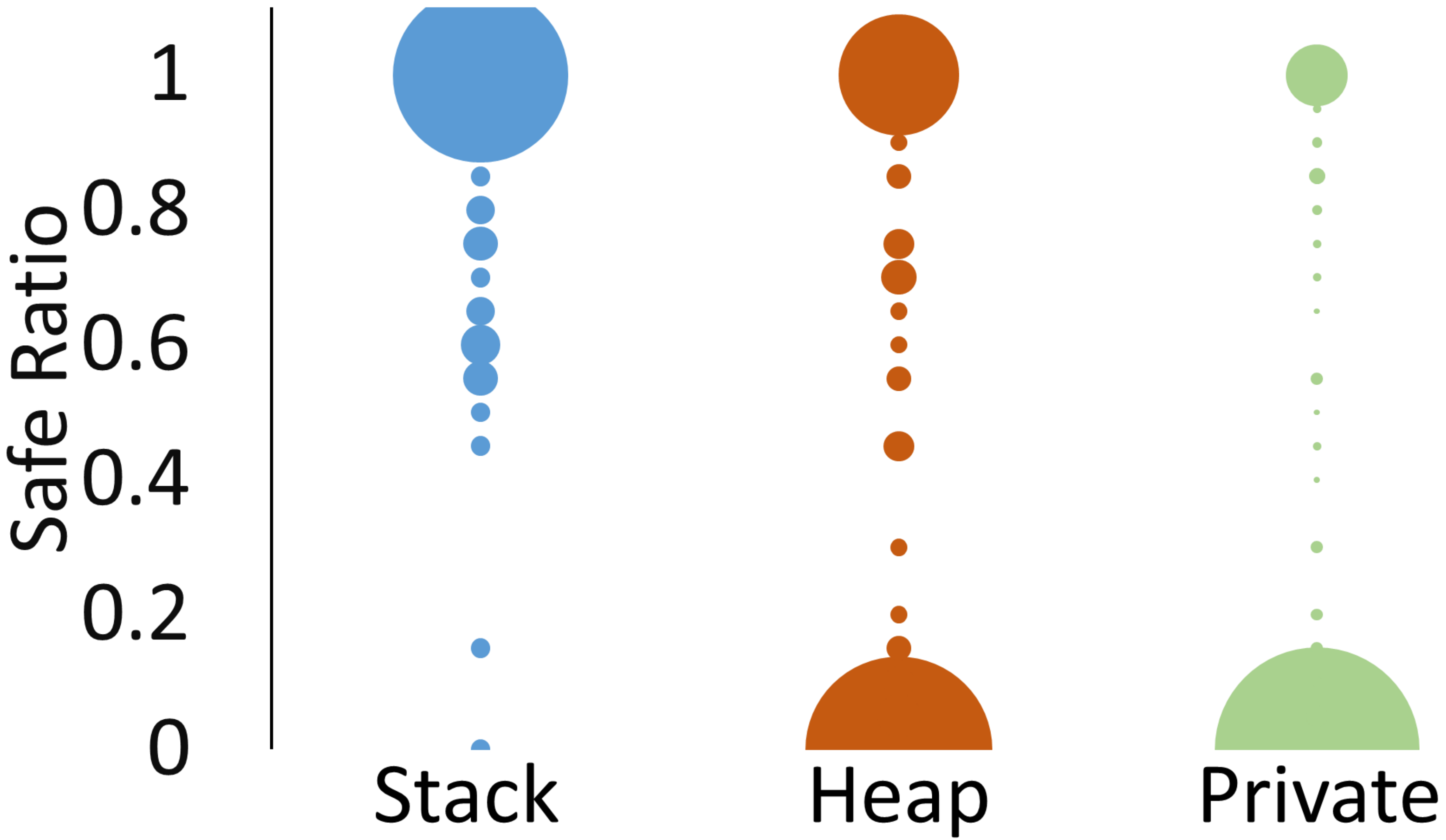
Explicit Recovery



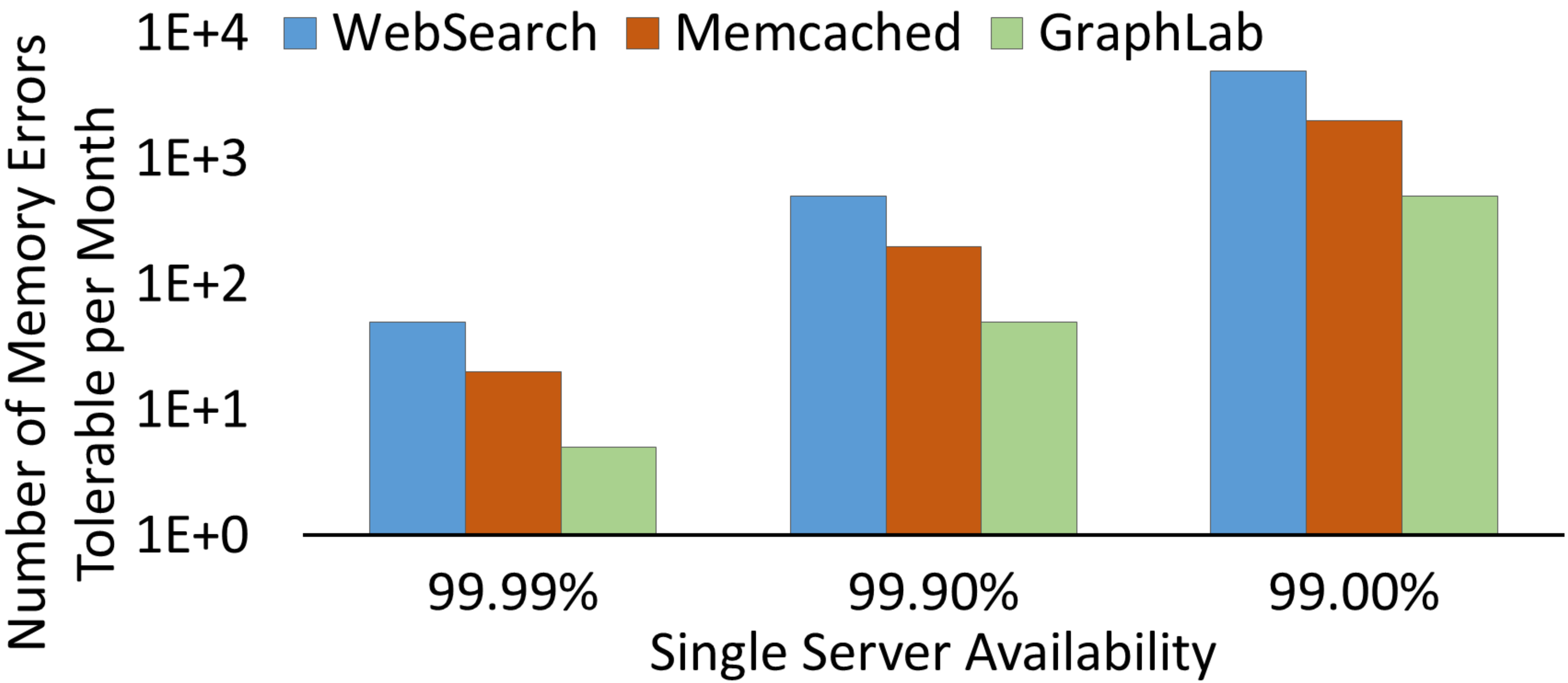
Quick to crash vs. periodic incorrect



Safe ratio: masked by overwrite



Potential to tolerate memory errors



Design dimension

Design dimension	Technique	Benefits	Trade-offs
Hardware techniques	No detection/correction	No associated overheads (low cost)	Unpredictable crashes and silent data corruption
	Parity	Relatively low cost with detection capability	No hardware correction capability
	SEC-DED/DEC-TED	Tolerate common single-/double-bit errors	Increased cost and memory access latency
	Chipkill [10]	Tolerate single-DRAM-chip errors	Increased cost and memory access latency
	Mirroring [12]	Tolerate memory module failure	100% capacity overhead
	Less-Tested DRAM	Saved testing cost during manufacturing	Increased error rates
Software responses	Consume errors in application	Simple, no performance overhead	Unpredictable crashes and data corruption
	Automatically restart application	Can prevent unpredictable application behavior	May make little progress if error is frequent
	Retire memory pages	Low overhead, effective for repeating errors	Reduces memory space (usually very little)
	Conditionally consume errors	Flexible, software vulnerability-aware	Memory management overhead to make decision
	Software correction	Tolerates detectable memory errors	Usually has performance overheads
Usage granularity	Physical machine	Simple, uniform usage across memory space	Costly depending on technique used
	Virtual machine	More fine-grained, flexible management	Host OS is still vulnerable to memory errors
	Application	Manageable by the OS	Does not leverage different region tolerance
	Memory region	Manageable by the OS	Does not leverage different page tolerance
	Memory page	Manageable by the OS	Does not leverage different data object tolerance
	Cache line	Most fine-grained management	Large management overhead; software changes

Table 4: Heterogeneous reliability design dimensions, techniques, and their potential benefits and trade-offs.

Design dimension

Design dimension	Technique
Hardware techniques	No detection/correction Parity SEC-DED/DEC-TED Chipkill [10] Mirroring [12] Less-Tested DRAM
Software responses	Consume errors in application Automatically restart application Retire memory pages Conditionally consume errors Software correction
Usage granularity	Physical machine Virtual machine Application Memory region Memory page Cache line