Scalable Many-Core Memory Systems Lecture 4, Topic 2: Emerging Technologies and Hybrid Memories

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

HiPEAC ACACES Summer School 2013

July 18, 2013

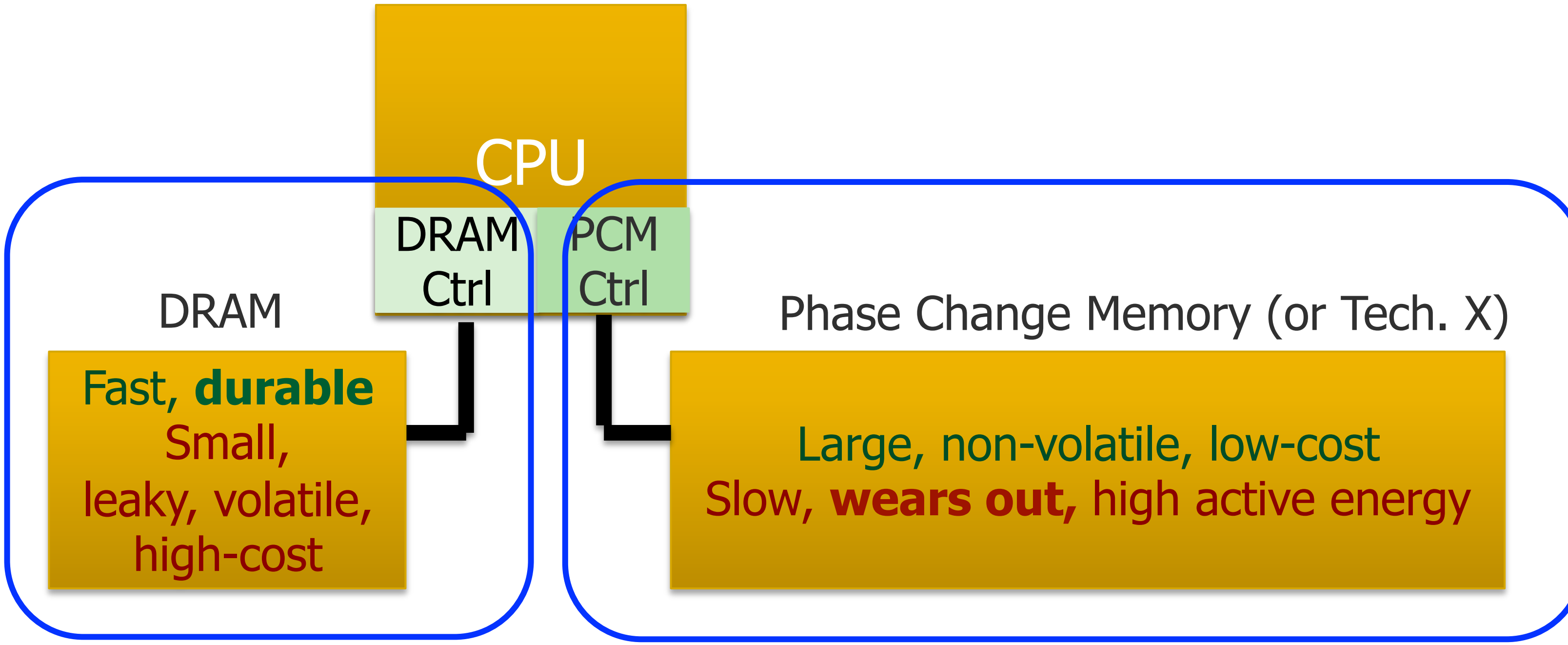
Carnegie Mellon

SAFARI

Agenda

- Major Trends Affecting Main Memory
- Requirements from an Ideal Main Memory System
- Opportunity: Emerging Memory Technologies
  - Background
  - PCM (or Technology X) as DRAM Replacement
  - Hybrid Memory Systems
- Conclusions
- Discussion

Hybrid Memory Systems



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

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

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

One Option: DRAM as a Cache for PCM

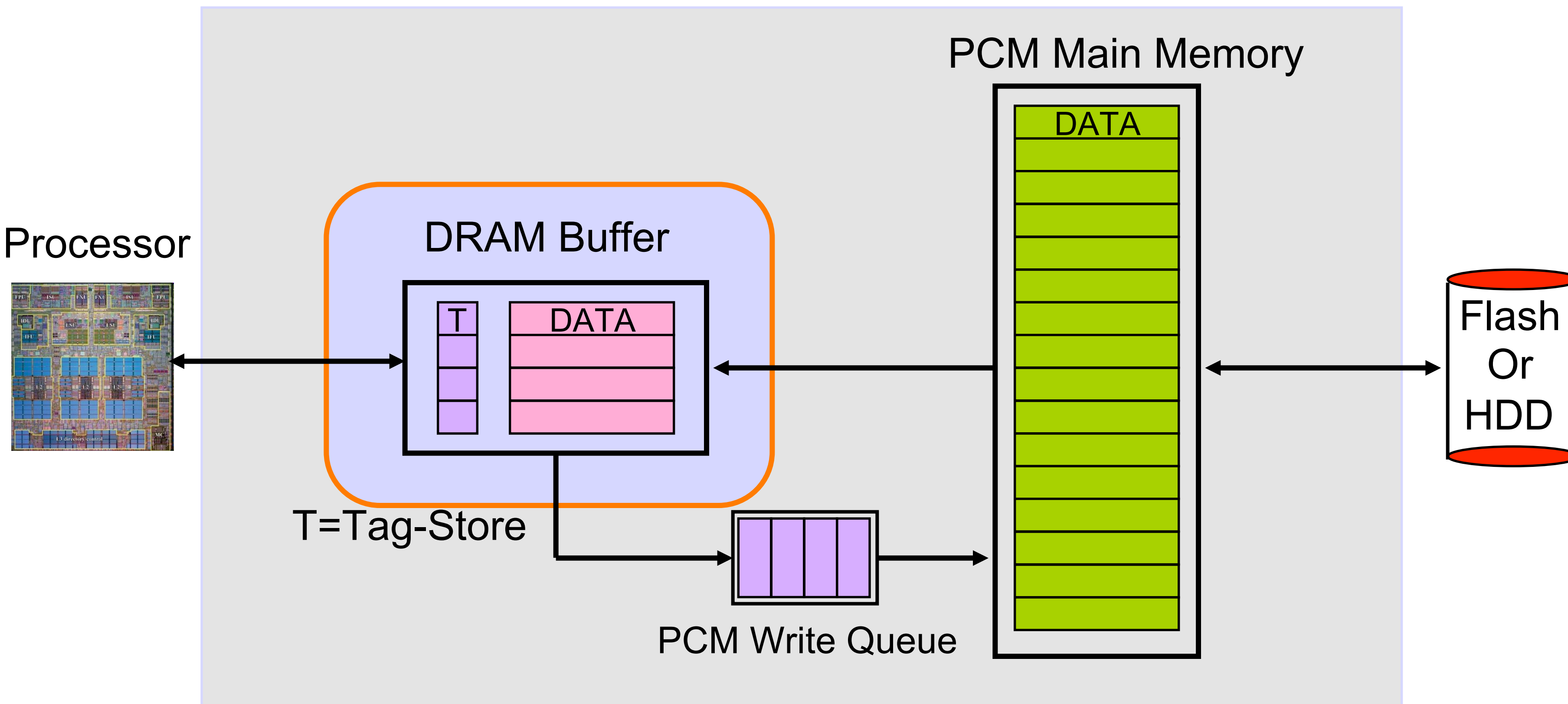
- PCM is main memory; DRAM caches memory rows/blocks
  - Benefits: Reduced latency on DRAM cache hit; write filtering
- Memory controller hardware manages the DRAM cache
  - Benefit: Eliminates system software overhead

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

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

DRAM as a Cache for PCM

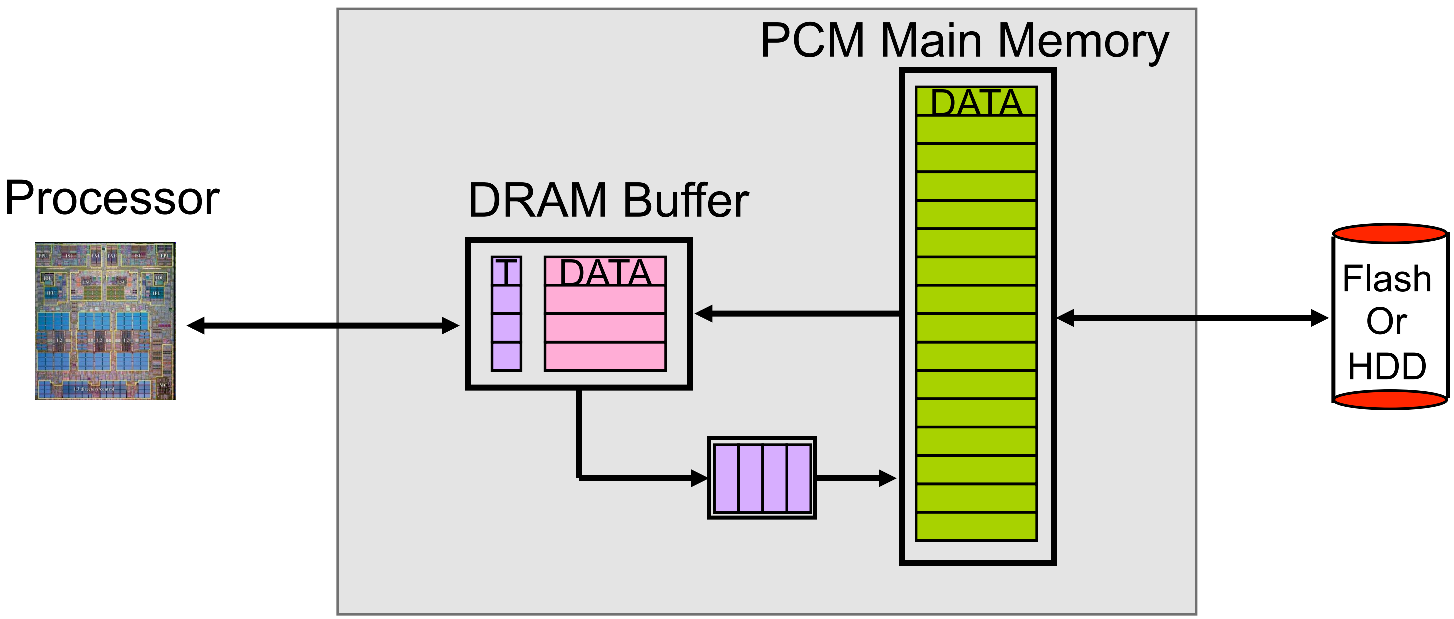
- Goal: Achieve the best of both DRAM and PCM/NVM
  - Minimize amount of DRAM w/o sacrificing performance, endurance
  - DRAM as cache to tolerate PCM latency and write bandwidth
  - PCM as main memory to provide large capacity at good cost and power



Qureshi+, "Scalable high performance main memory system using phase-change memory technology," ISCA 2009.

Write Filtering Techniques

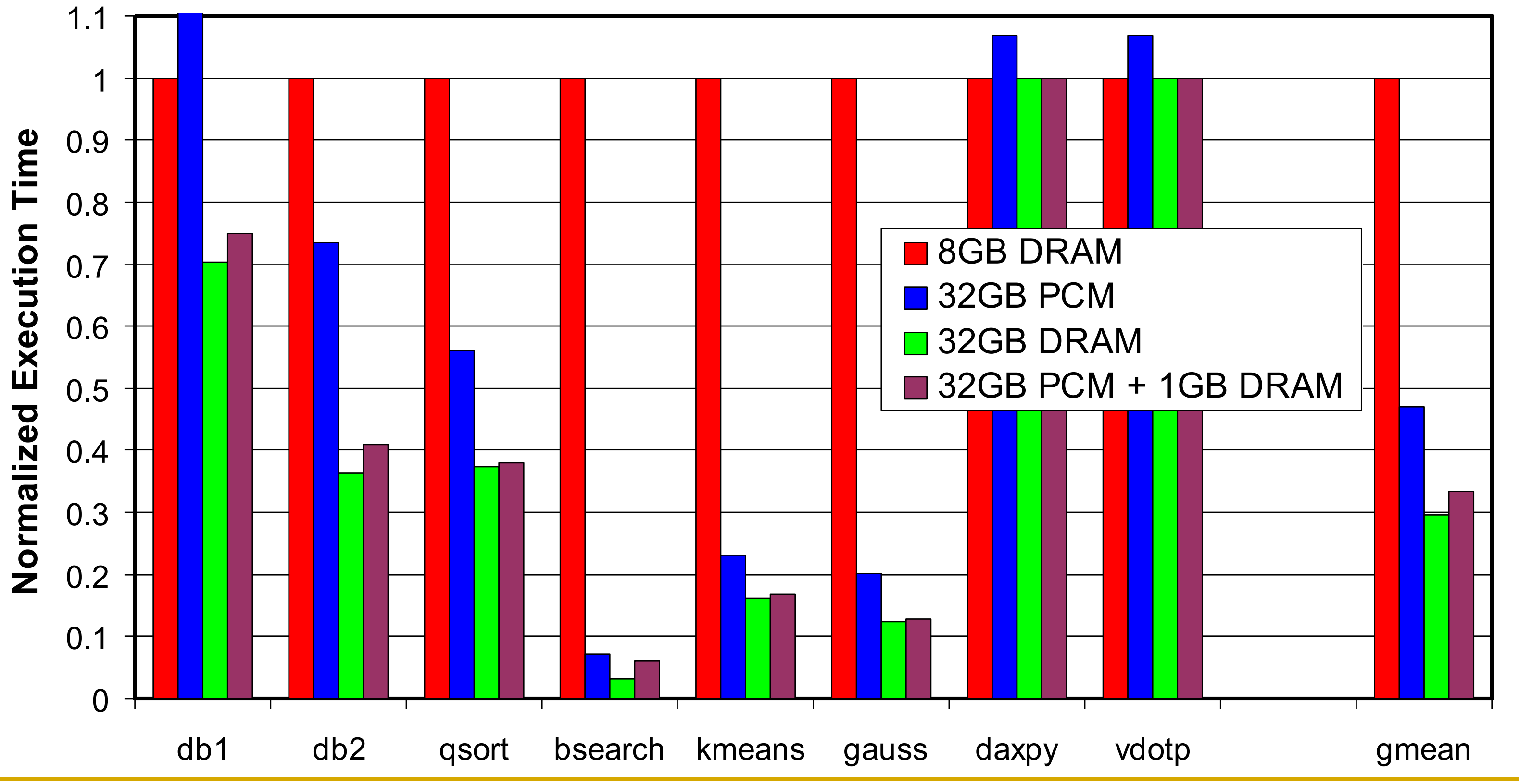
- Lazy Write: Pages from disk installed only in DRAM, not PCM
- Partial Writes: Only dirty lines from DRAM page written back
- Page Bypass: Discard pages with poor reuse on DRAM eviction



- Qureshi et al., "Scalable high performance main memory system using phase-change memory technology," ISCA 2009.

Results: DRAM as PCM Cache (I)

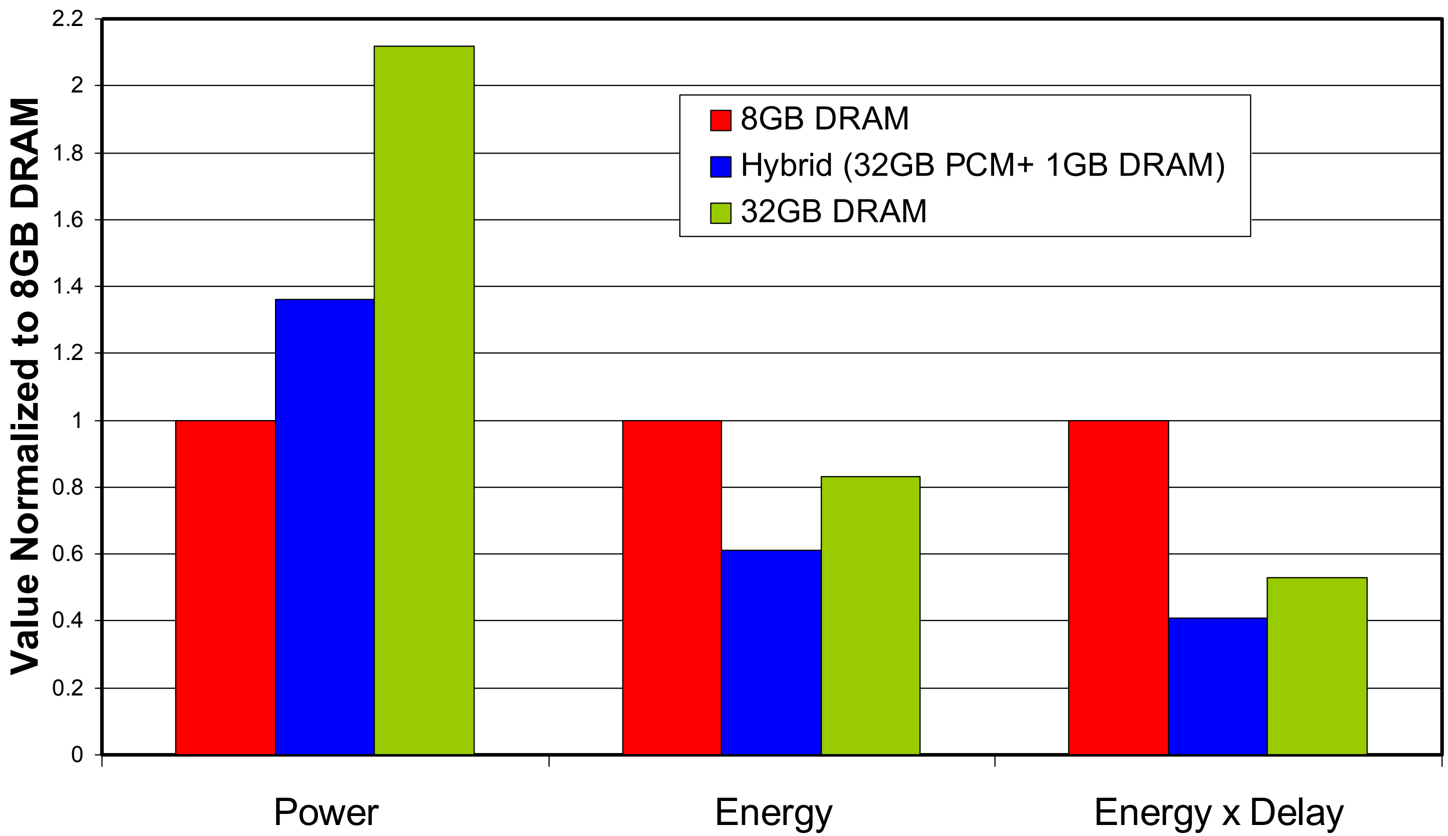
- Simulation of 16-core system, 8GB DRAM main-memory at 320 cycles, HDD (2 ms) with Flash (32 us) with Flash hit-rate of 99%
- Assumption: PCM 4x denser, 4x slower than DRAM
- DRAM block size = PCM page size (4kB)



Qureshi+, "Scalable high performance main memory system using phase-change memory technology," ISCA 2009.

Results: DRAM as PCM Cache (II)

- PCM-DRAM Hybrid performs similarly to similar-size DRAM
- Significant power and energy savings with PCM-DRAM Hybrid
- Average lifetime: 9.7 years (no guarantees)



Qureshi+, "Scalable high performance main memory system using phase-change memory technology," ISCA 2009.

Agenda

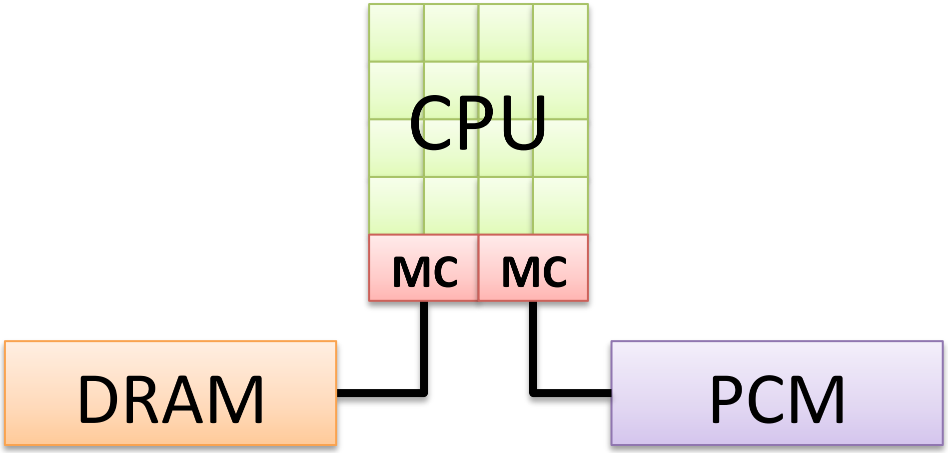
- Major Trends Affecting Main Memory
- Requirements from an Ideal Main Memory System
- Opportunity: Emerging Memory Technologies
  - Background
  - PCM (or Technology X) as DRAM Replacement
  - Hybrid Memory Systems
    - Row-Locality Aware Data Placement
    - Efficient DRAM (or Technology X) Caches
- Conclusions
- Discussion

Row Buffer Locality Aware Caching Policies for Hybrid Memories

HanBin Yoon, Justin Meza, Rachata Ausavarungnirun, Rachael Harding, and Onur Mutlu,
"Row Buffer Locality Aware Caching Policies for Hybrid Memories"
*Proceedings of the 30th IEEE International Conference on Computer Design (**ICCD**),*
Montreal, Quebec, Canada, September 2012. Slides (pptx) (pdf)

Hybrid Memory

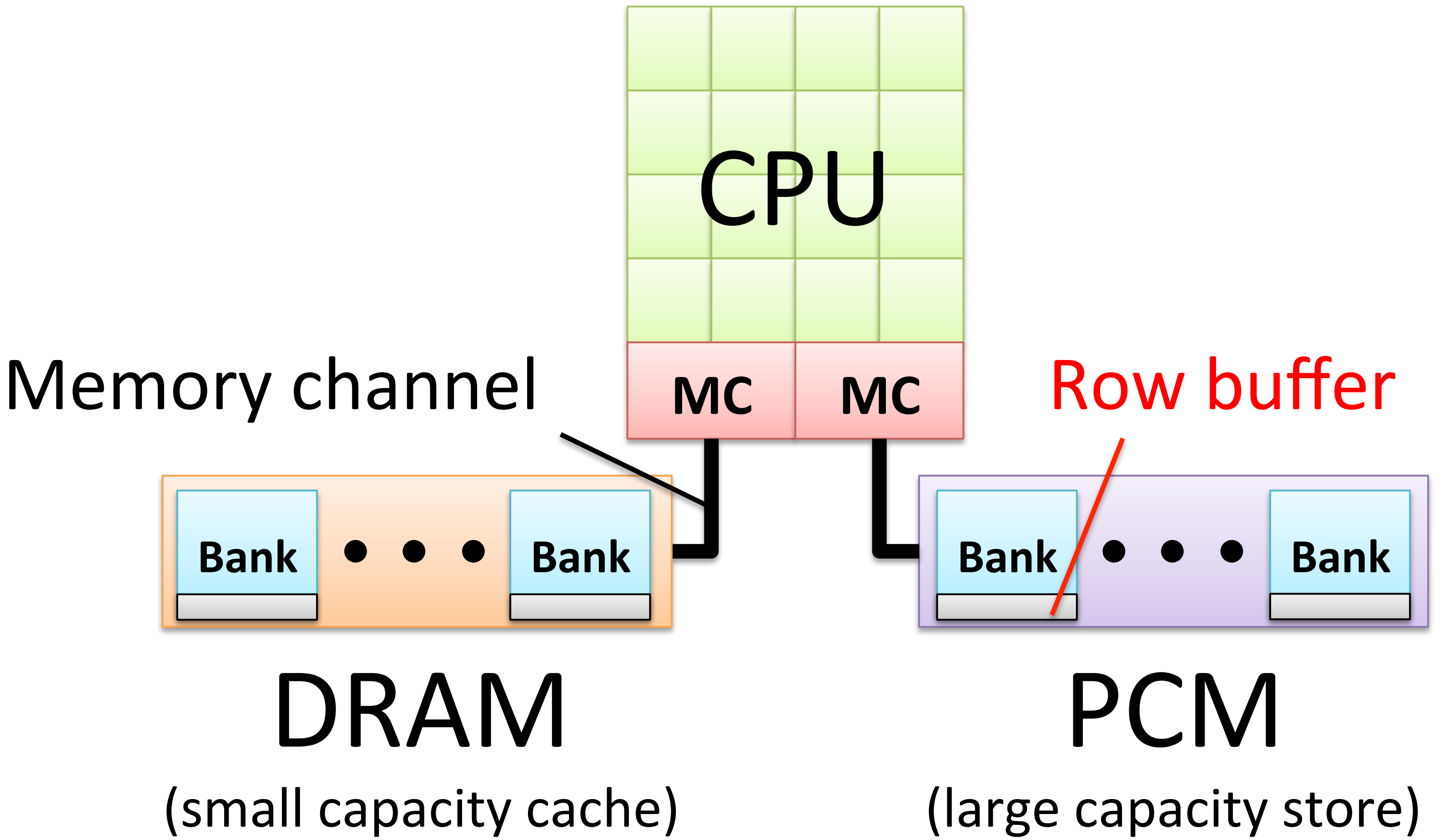
- Key question: How to place data between the heterogeneous memory devices?



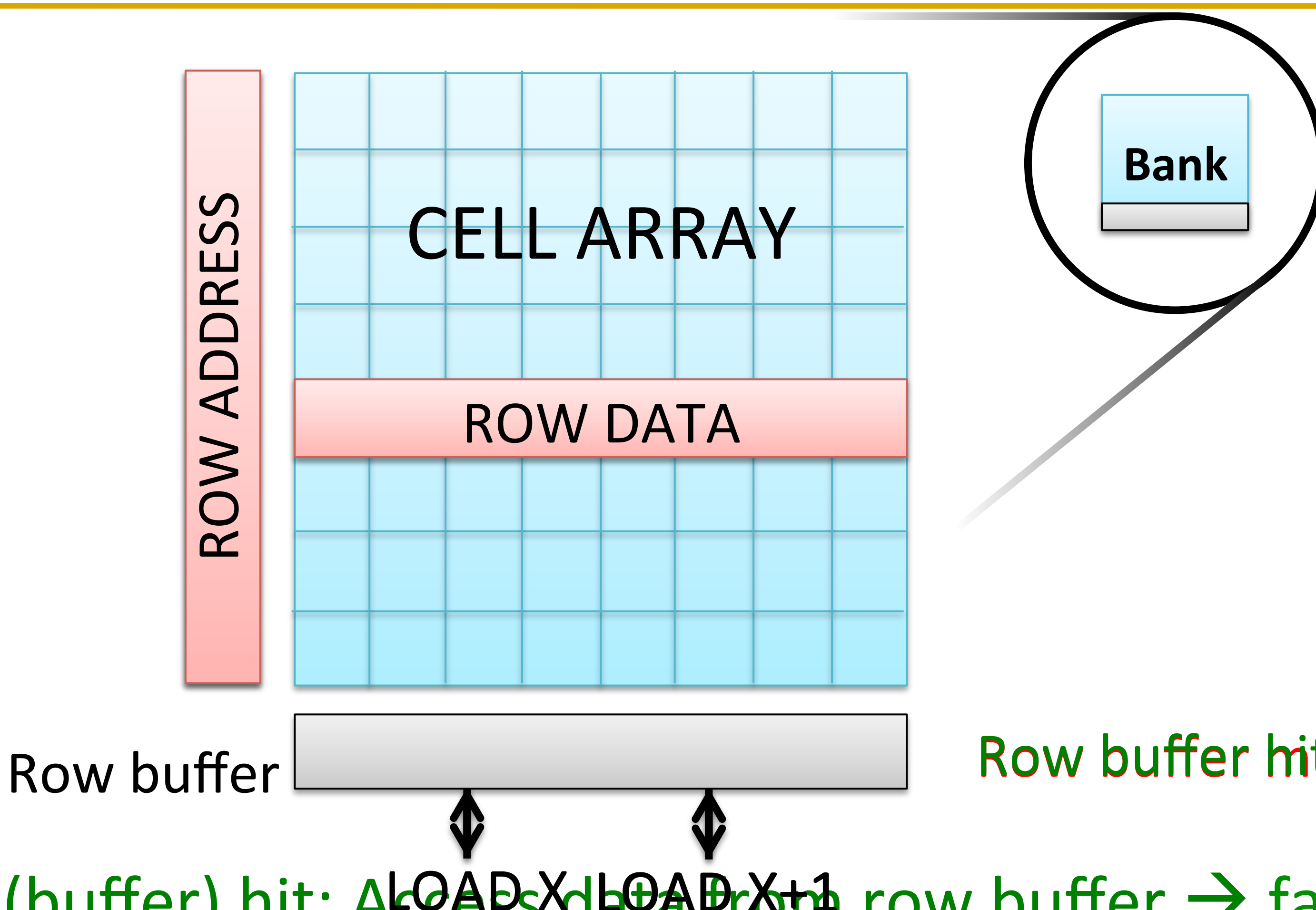
Outline

- Background: Hybrid Memory Systems
- Motivation: Row Buffers and Implications on Data Placement
- Mechanisms: Row Buffer Locality-Aware Caching Policies
- Evaluation and Results
- Conclusion

Hybrid Memory: A Closer Look



Row Buffers and Latency



Row buffer hits!
Row buffer miss!

Row (buffer) hit: Access data from row buffer → fast

Row (buffer) miss: Access data from cell array → slow

Key Observation

- Row buffers exist in both DRAM and PCM
  - Row hit latency **similar** in DRAM & PCM [Lee+ ISCA'09]
  - Row miss latency **small** in DRAM, **large** in PCM

- Place data in DRAM which
  - is likely to miss in the row buffer (low row buffer locality)→ miss penalty is smaller in DRAM

    AND
  - is reused many times → cache only the data worth the movement cost and DRAM space

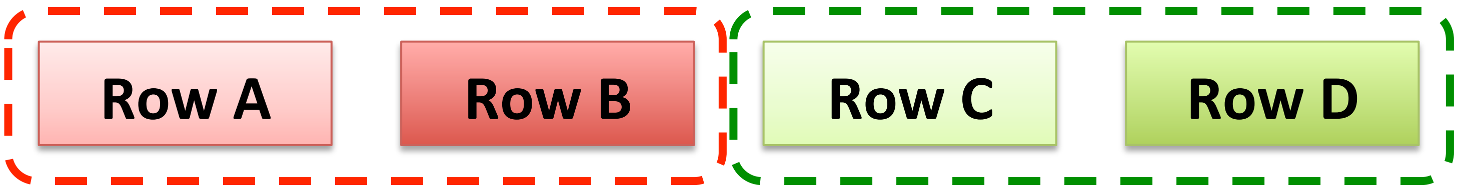
RBL-Awareness: An Example

Let’s say a processor accesses four rows

Row A **Row B** **Row C** **Row D**

RBL-Awareness: An Example

Let's say a processor accesses four rows with different row buffer localities (RBL)



Low RBL
(Frequently miss in row buffer)

High RBL
(Frequently hit in row buffer)

Case 1: RBL-*Unaware* Policy (state-of-the-art)
Case 2: RBL-Aware Policy (RBLA)

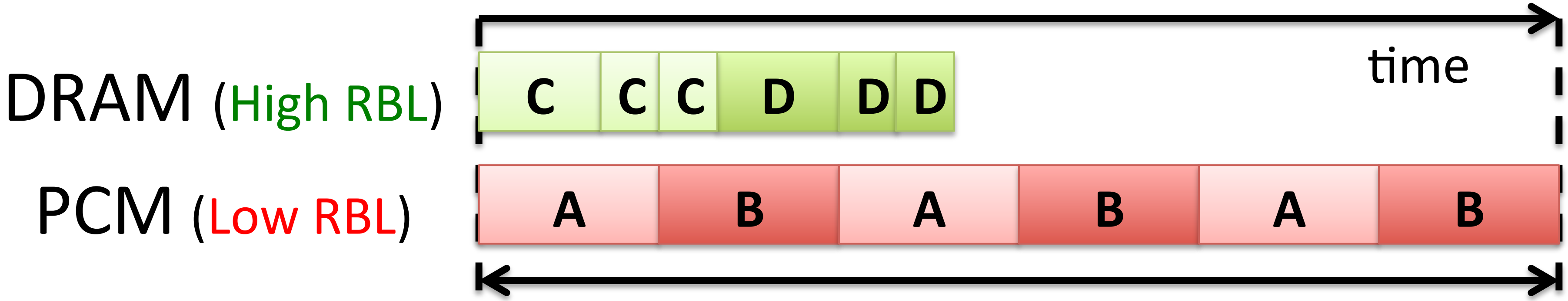
Case 1: RBL-*Unaware* Policy

A **row buffer locality-*unaware*** policy could place these rows in the following manner

Case 1: RBL-*Unaware* Policy

Access pattern to main memory:

A (oldest), B, C, C, C, A, B, D, D, D, A, B (youngest)



RBL-*Unaware*: Stall time is 6 PCM device accesses

Case 2: RBL-Aware Policy (RBLA)

A **row buffer locality-aware** policy would place these rows in the **opposite** manner

DRAM

(Low RBL)

→ Access data at lower row buffer miss latency of DRAM

PCM

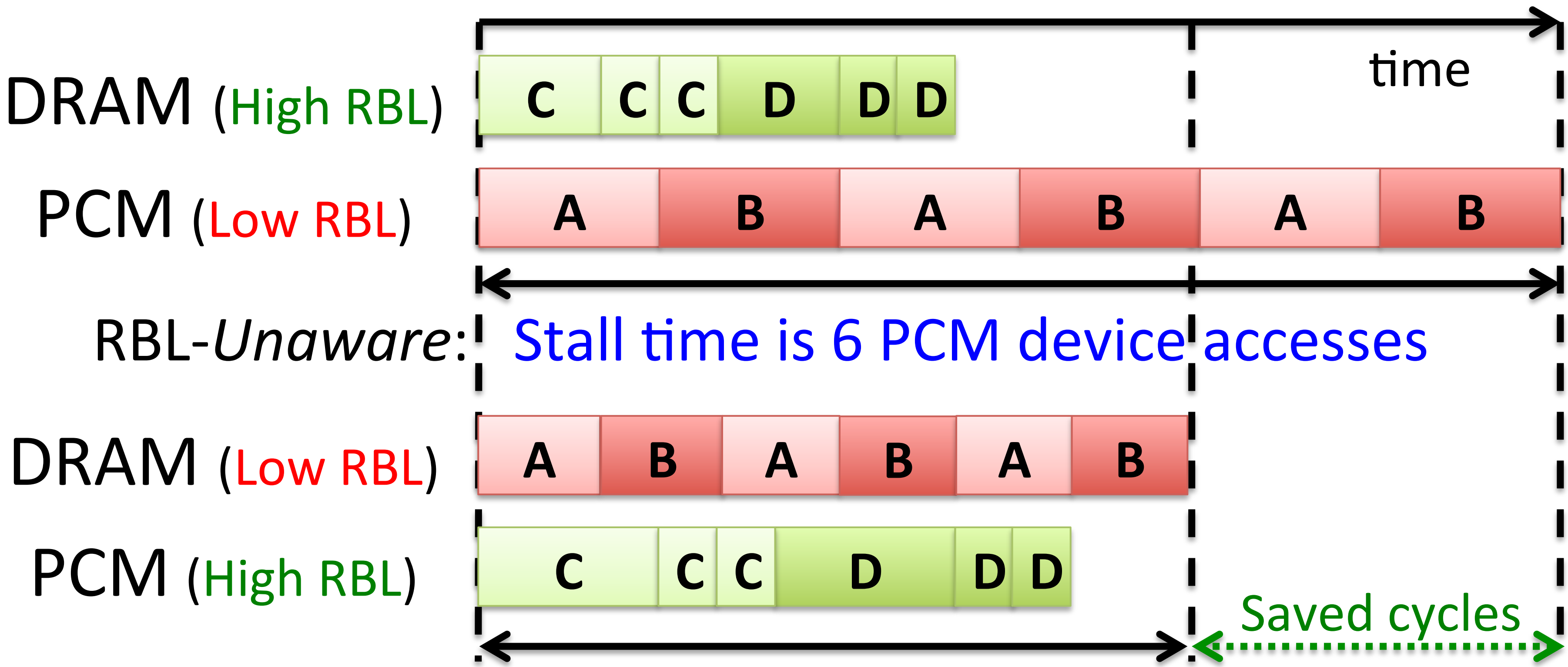
(High RBL)

→ Access data at low row buffer hit latency of PCM

Case 2: RBL-Aware Policy (RBLA)

Access pattern to main memory:

A (oldest), B, C, C, C, A, B, D, D, D, A, B (youngest)



RBL-Aware: Stall time is 6 **DRAM** device accesses

Outline

- Background: Hybrid Memory Systems
- Motivation: Row Buffers and Implications on Data Placement
- Mechanisms: Row Buffer Locality-Aware Caching Policies
- Evaluation and Results
- Conclusion

Our Mechanism: RBLA

1. For recently used rows in PCM:
   – Count row buffer misses as indicator of row buffer locality (RBL)

2. Cache to DRAM rows with misses ≥ threshold
   – Row buffer miss counts are periodically reset (only cache rows with high reuse)

Our Mechanism: RBLA-Dyn

1. For recently used rows in PCM:
   – Count row buffer misses as indicator of row buffer locality (RBL)

2. Cache to DRAM rows with misses ≥ threshold
   – Row buffer miss counts are periodically reset (only cache rows with high reuse)

3. Dynamically adjust threshold to adapt to workload/system characteristics
   – Interval-based cost-benefit analysis

Implementation: "Statistics Store"

- Goal: To keep count of row buffer misses to recently used rows in PCM

- Hardware structure in memory controller
  - Operation is similar to a cache
    - Input: row address
    - Output: row buffer miss count
  - 128-set 16-way statistics store (9.25KB) achieves system performance within 0.3% of an unlimited-sized statistics store

Outline

- Background: Hybrid Memory Systems
- Motivation: Row Buffers and Implications on Data Placement
- Mechanisms: Row Buffer Locality-Aware Caching Policies
- Evaluation and Results
- Conclusion

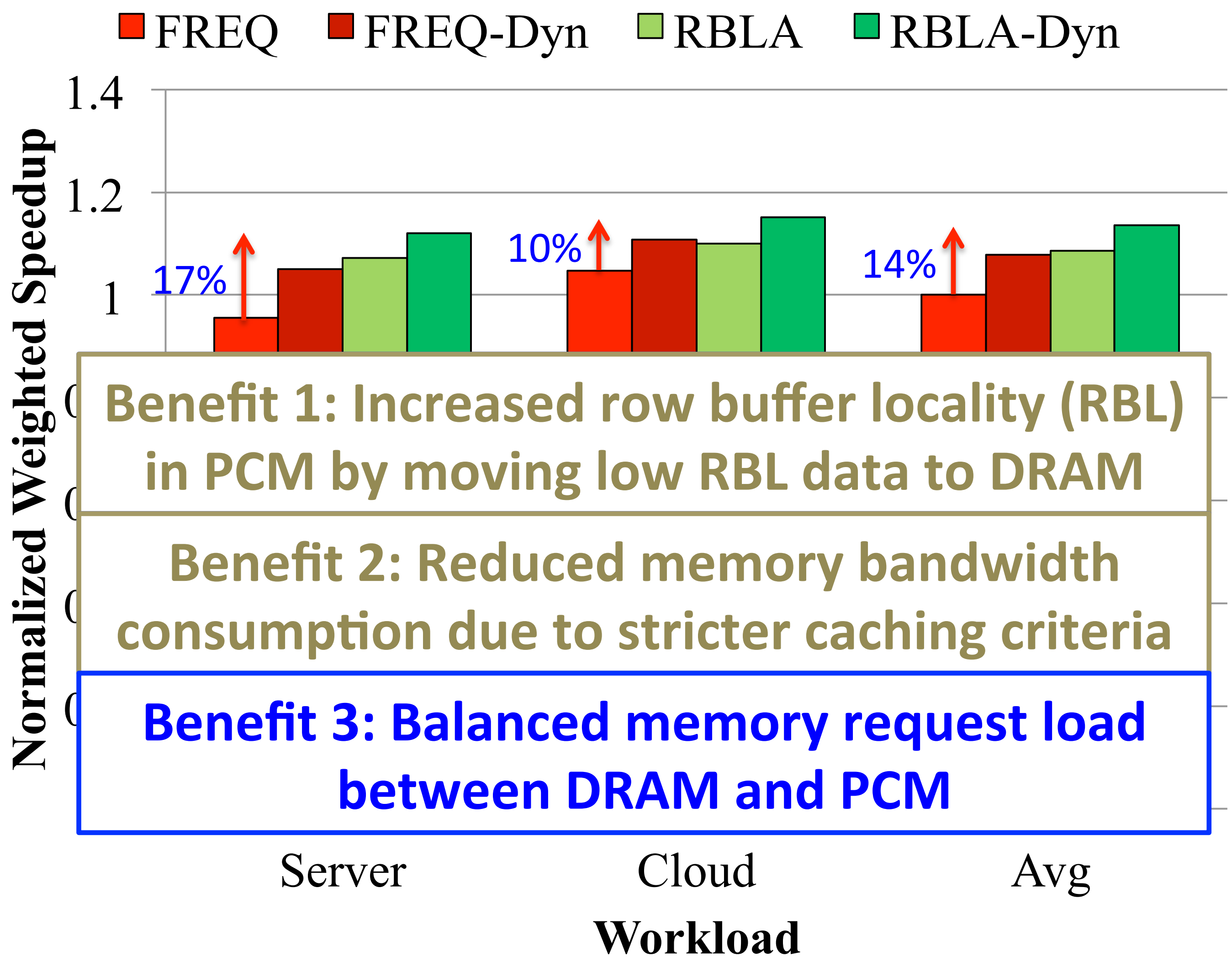
Evaluation Methodology

- Cycle-level x86 CPU-memory simulator
  - **CPU**: 16 out-of-order cores, 32KB private L1 per core, 512KB shared L2 per core
  - **Memory**: 1GB DRAM (8 banks), 16GB PCM (8 banks), 4KB migration granularity
- 36 multi-programmed server, cloud workloads
  - Server: TPC-C (OLTP), TPC-H (Decision Support)
  - Cloud: Apache (Webserv.), H.264 (Video), TPC-C/H
- Metrics: Weighted speedup (perf.), perf./Watt (energy eff.), Maximum slowdown (fairness)

Comparison Points

- **Conventional LRU Caching**
- **FREQ**: Access-frequency-based caching
  - Places "hot data" in cache [Jiang+ HPCA'10]
  - Cache to DRAM rows with accesses ≥ threshold
  - Row buffer locality-*unaware*
- **FREQ-Dyn**: Adaptive Freq.-based caching
  - **FREQ** + our dynamic threshold adjustment
  - Row buffer locality-*unaware*
- **RBLA**: Row buffer locality-aware caching
- **RBLA-Dyn**: Adaptive RBL-aware caching

System Performance

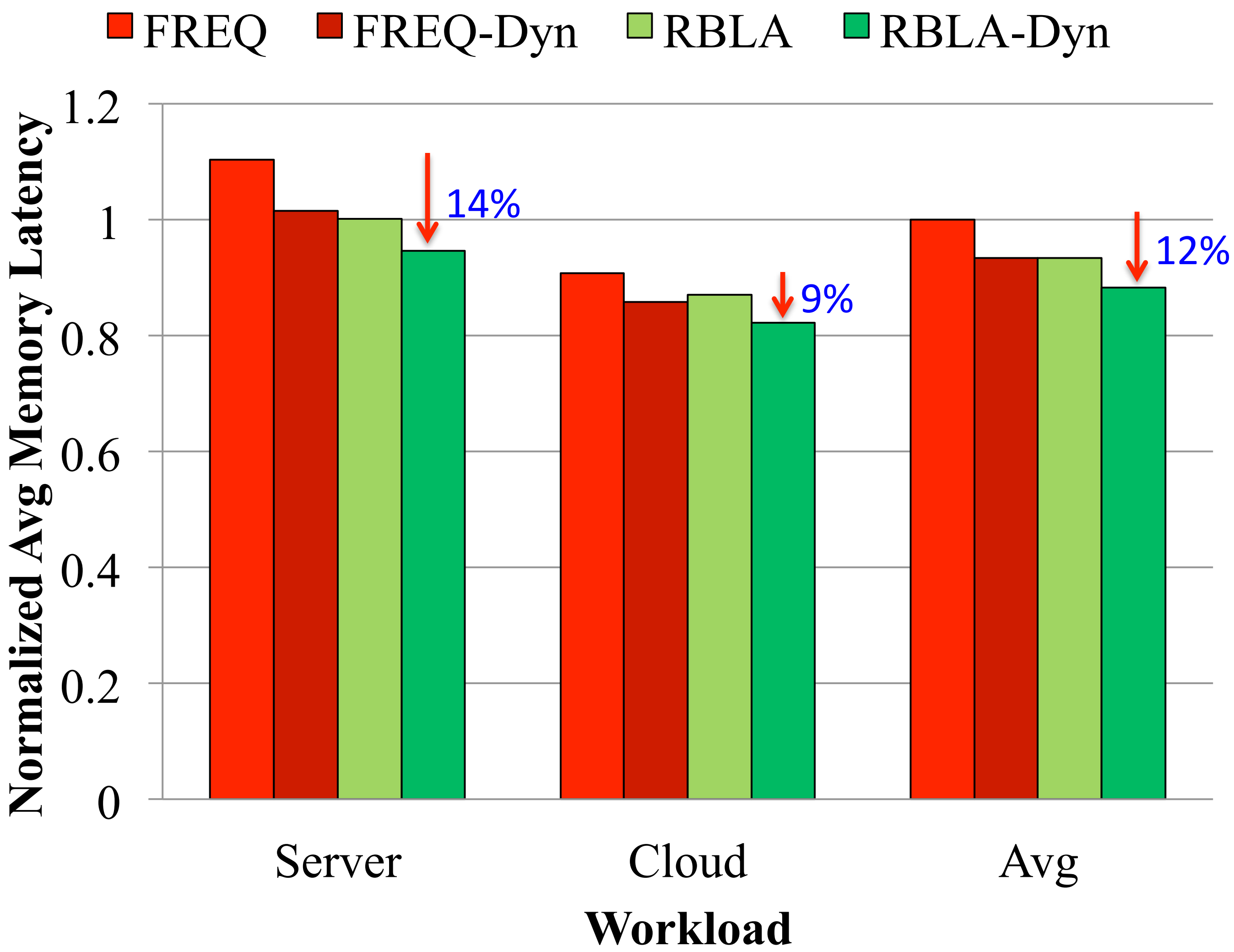


Benefit 1: Increased row buffer locality (RBL) in PCM by moving low RBL data to DRAM

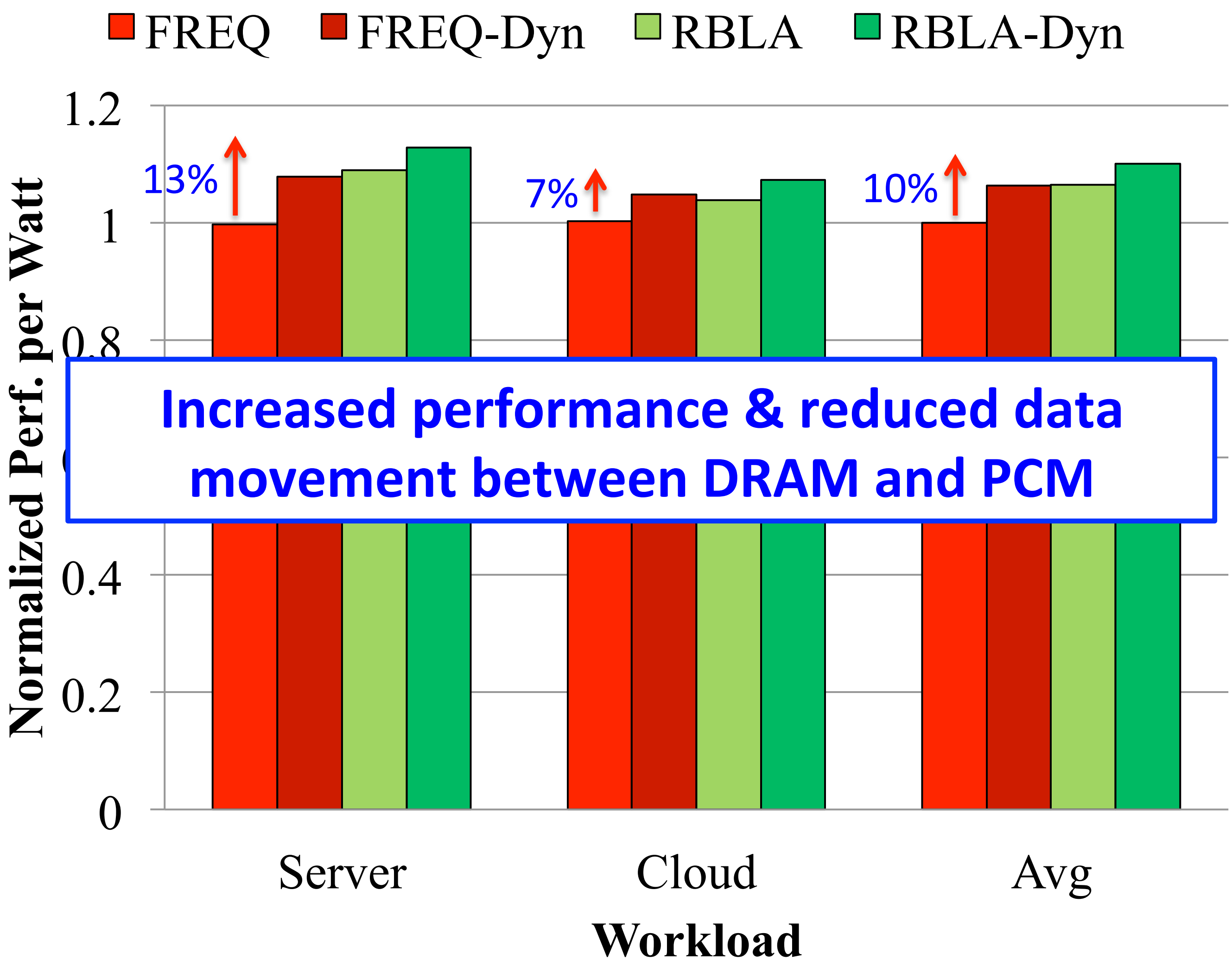
Benefit 2: Reduced memory bandwidth consumption due to stricter caching criteria

Benefit 3: Balanced memory request load between DRAM and PCM

Average Memory Latency

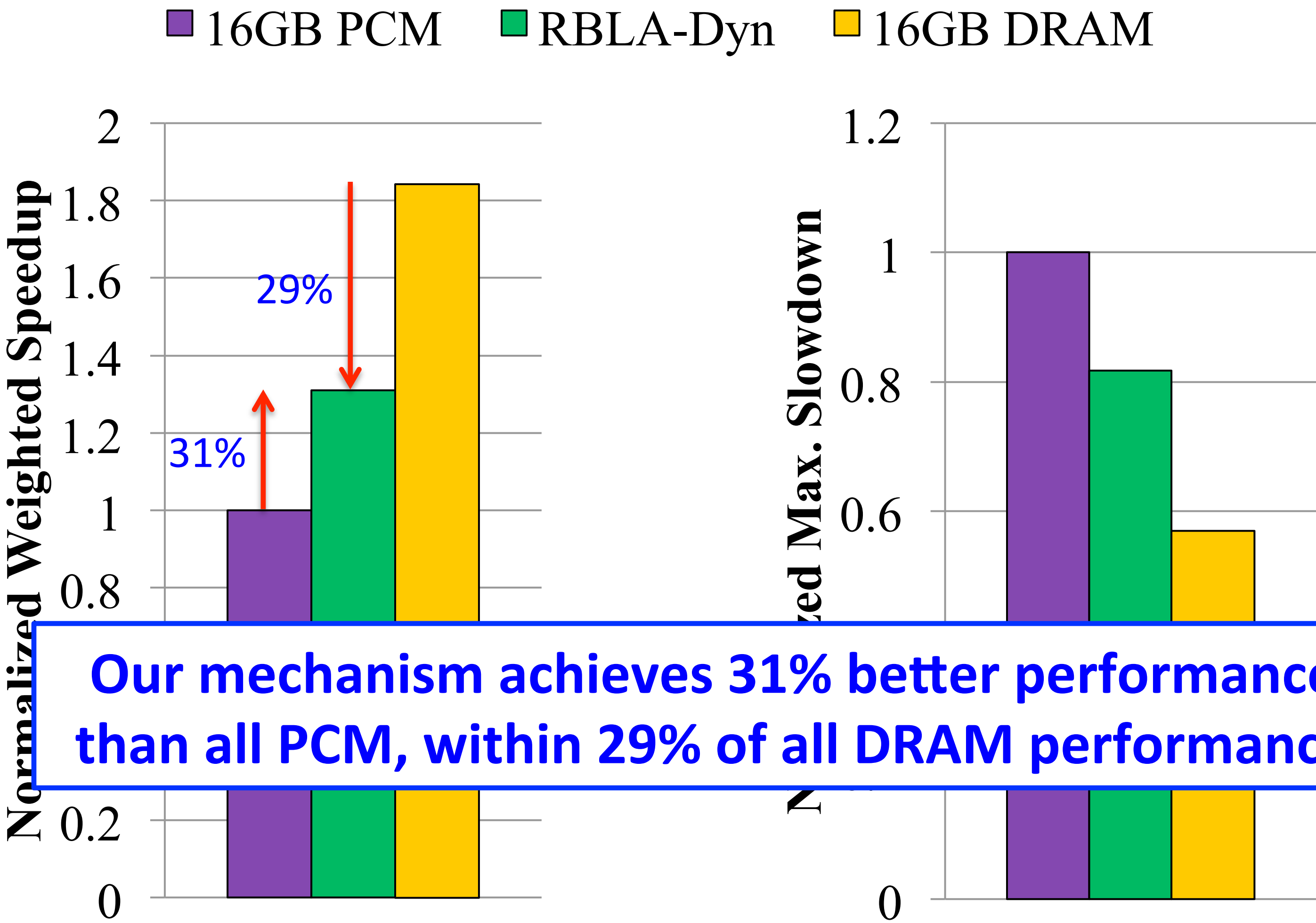


Memory Energy Efficiency



Increased performance & reduced data movement between DRAM and PCM

Compared to All-PCM/DRAM



Our mechanism achieves 31% better performance than all PCM, within 29% of all DRAM performance

Summary

- Different memory technologies have different strengths
- A hybrid memory system (DRAM-PCM) aims for best of both
- **Problem:** How to place data between these heterogeneous memory devices?
- **Observation:** PCM array access latency is higher than DRAM's – But peripheral circuit (row buffer) access latencies are similar
- **Key Idea:** Use row buffer locality (RBL) as a key criterion for data placement
- **Solution:** Cache to DRAM rows with low RBL and high reuse
- Improves both performance and energy efficiency over state-of-the-art caching policies

Row Buffer Locality Aware Caching Policies for Hybrid Memories

HanBin Yoon
Justin Meza
Rachata Ausavarungnirun
Rachael Harding
Onur Mutlu

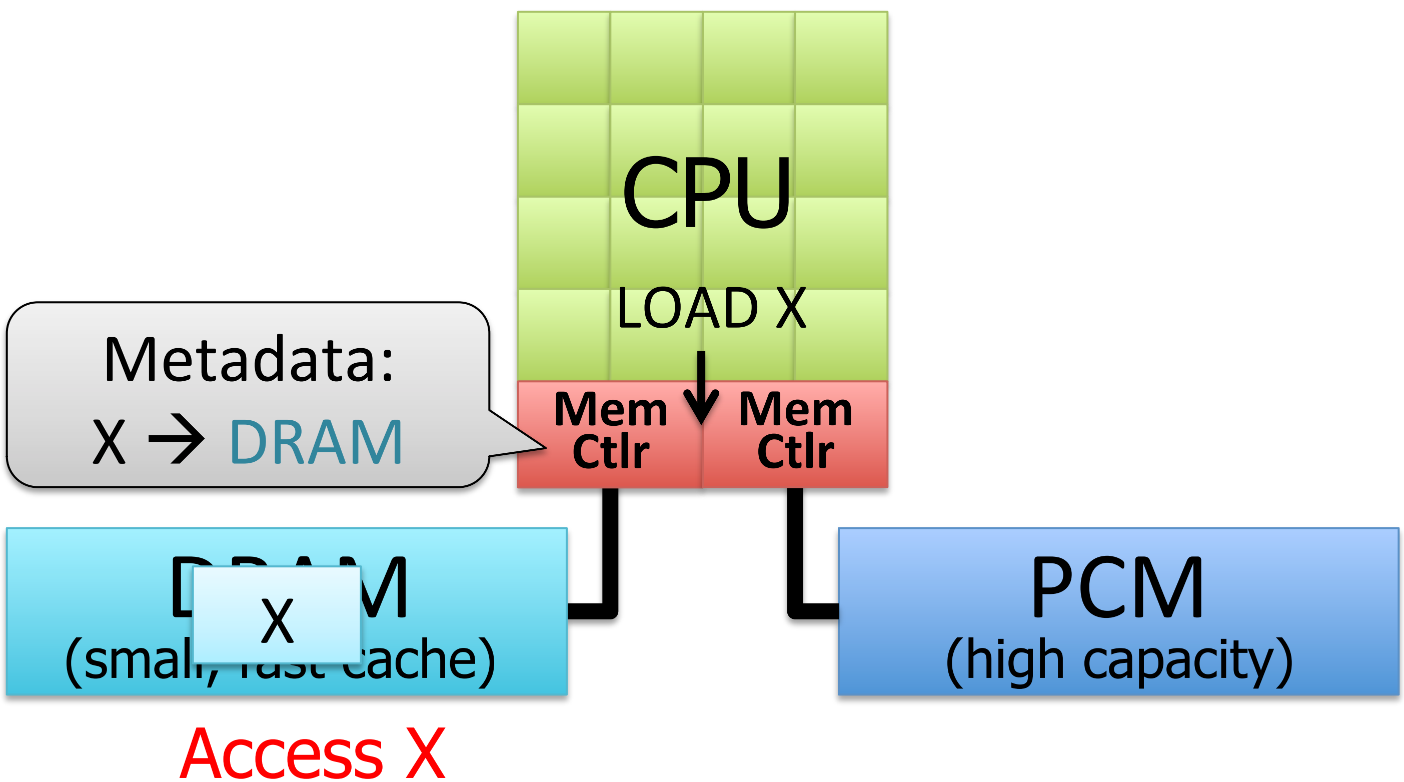
Carnegie Mellon University

Agenda

- Major Trends Affecting Main Memory
- Requirements from an Ideal Main Memory System
- Opportunity: Emerging Memory Technologies
  - Background
  - PCM (or Technology X) as DRAM Replacement
  - Hybrid Memory Systems
    - Row-Locality Aware Data Placement
    - Efficient DRAM (or Technology X) Caches
- Conclusions
- Discussion

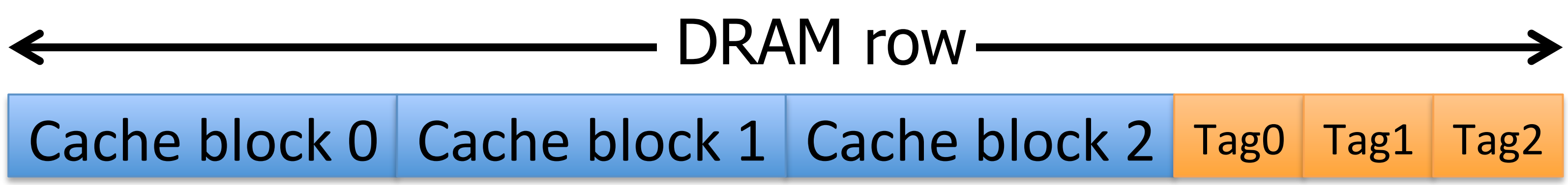
The Problem with Large DRAM Caches

- A large DRAM cache requires a large metadata (tag + block-based information) store
- How do we design an efficient DRAM cache?



Idea 1: Tags in Memory

- Store tags in the same row as data in DRAM
  - Store metadata in same row as their data
  - Data and metadata can be accessed together



- Benefit: No on-chip tag storage overhead
- Downsides:
  - Cache hit determined only after a DRAM access
  - Cache hit requires two DRAM accesses

Idea 2: Cache Tags in SRAM

- Recall Idea 1: Store all metadata in DRAM
  - To reduce metadata storage overhead

- Idea 2: Cache in on-chip SRAM frequently-accessed metadata
  - Cache only a small amount to keep SRAM size small

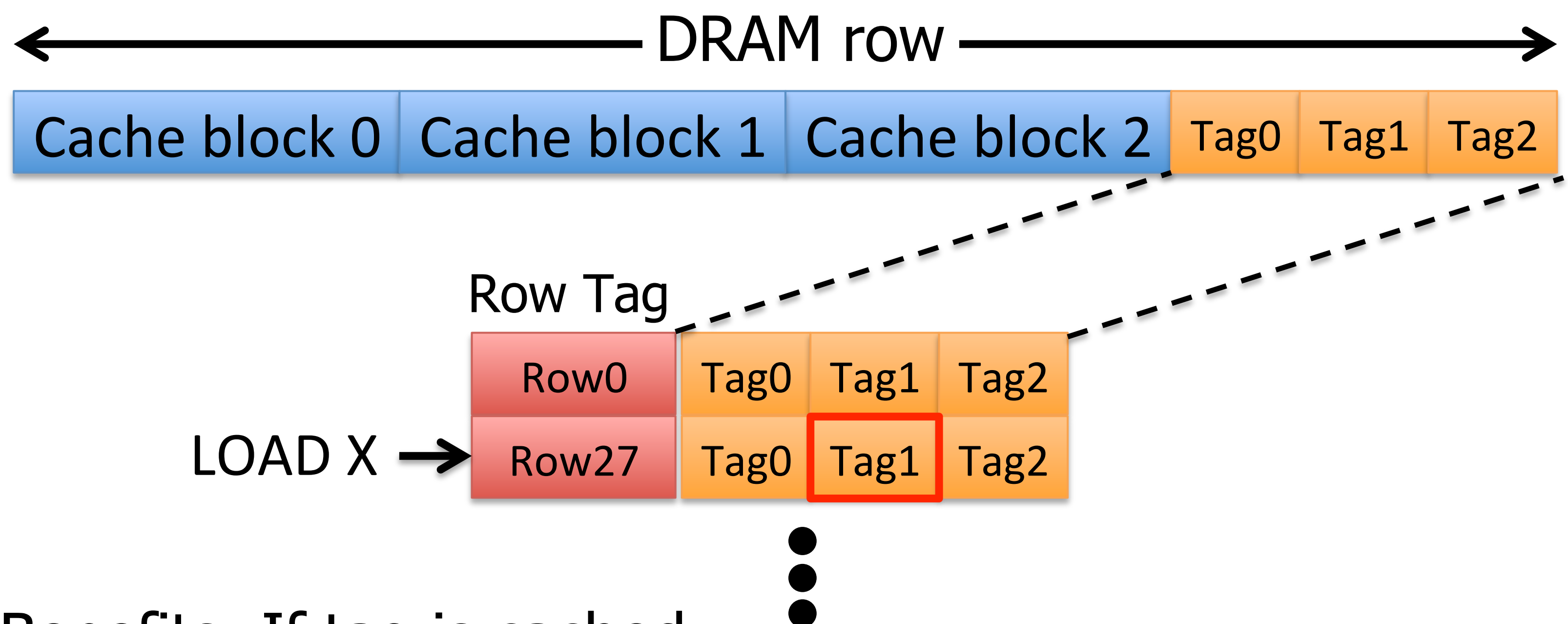
Idea 3: Dynamic Data Transfer Granularity

- Some applications benefit from caching more data
  - They have good spatial locality
- Others do not
  - Large granularity wastes bandwidth and reduces cache utilization

- Idea 3: Simple dynamic caching granularity policy
  - Cost-benefit analysis to determine best DRAM cache block size
  - Group main memory into sets of rows
  - Some row sets follow a fixed caching granularity
  - The rest of main memory follows the best granularity
    - Cost–benefit analysis: access latency versus number of cachings
    - Performed every quantum

TIMBER Tag Management

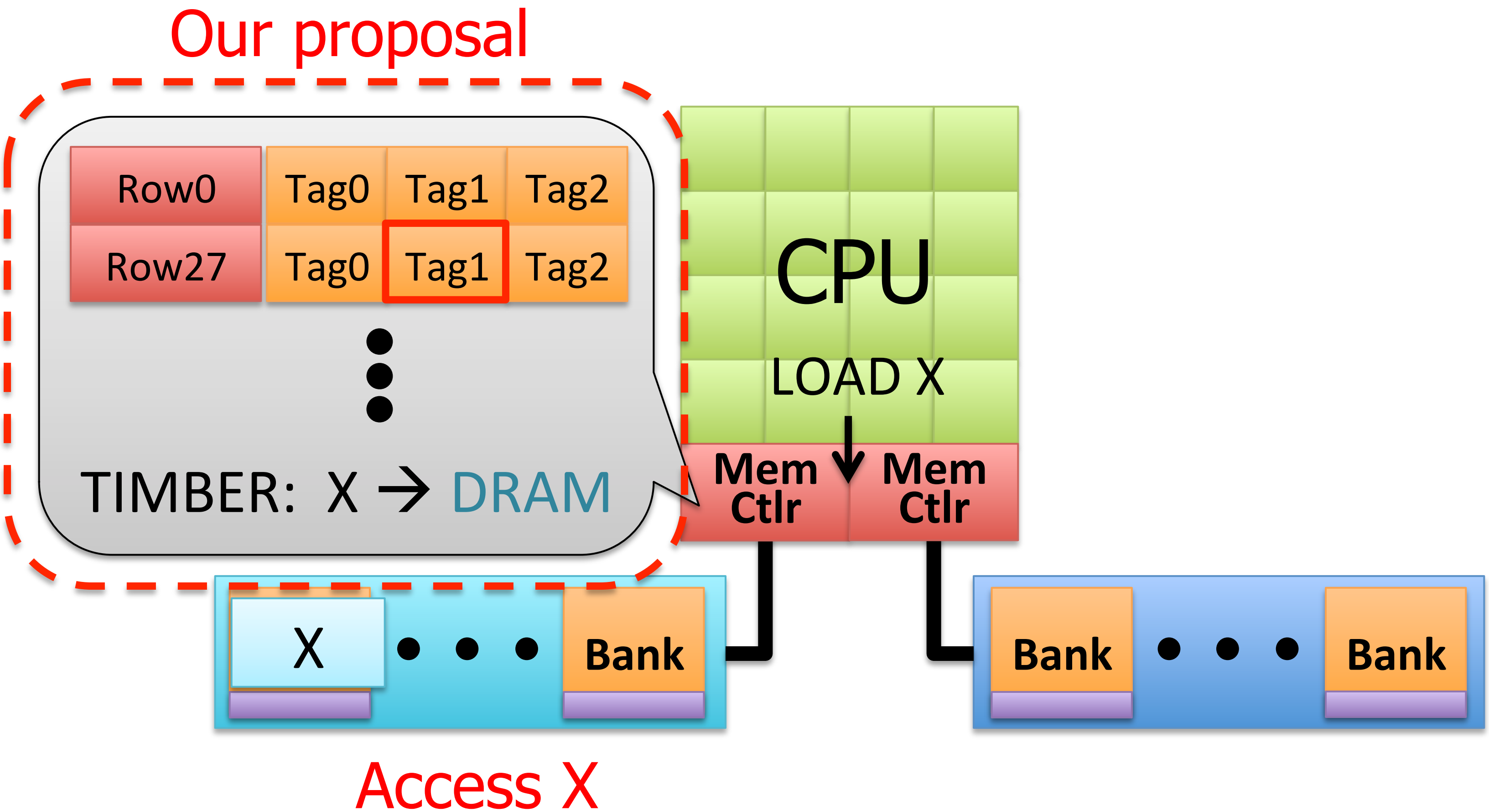
- A Tag-In-Memory BuffER (TIMBER)
  - Stores recently-used tags in a small amount of SRAM



- Benefits: If tag is cached:
  - no need to access DRAM twice
  - cache hit determined quickly

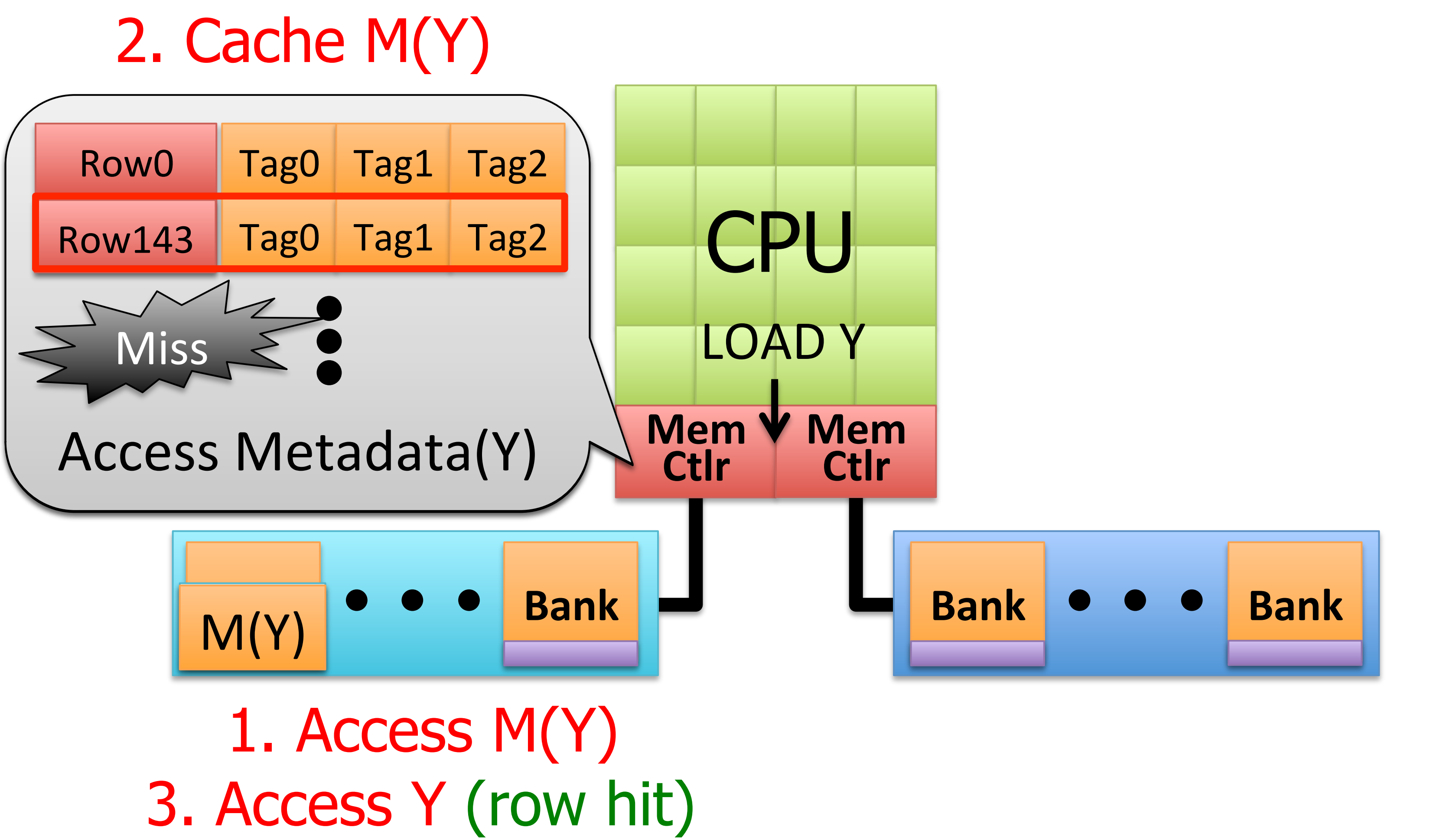
TIMBER Tag Management Example (I)

- Case 1: TIMBER hit



TIMBER Tag Management Example (II)

- Case 2: TIMBER miss



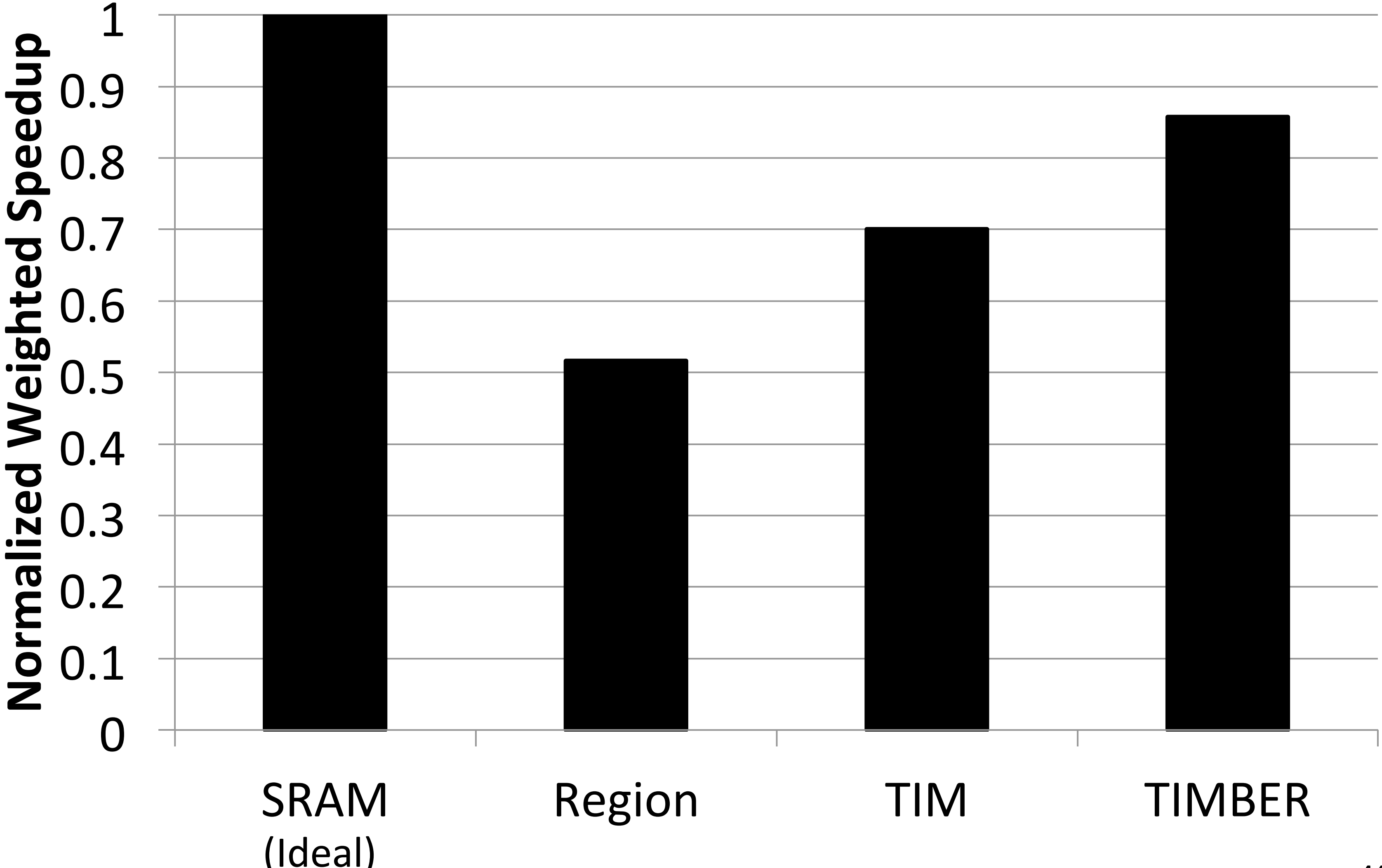
Methodology

- System: 8 out-of-order cores at 4 GHz

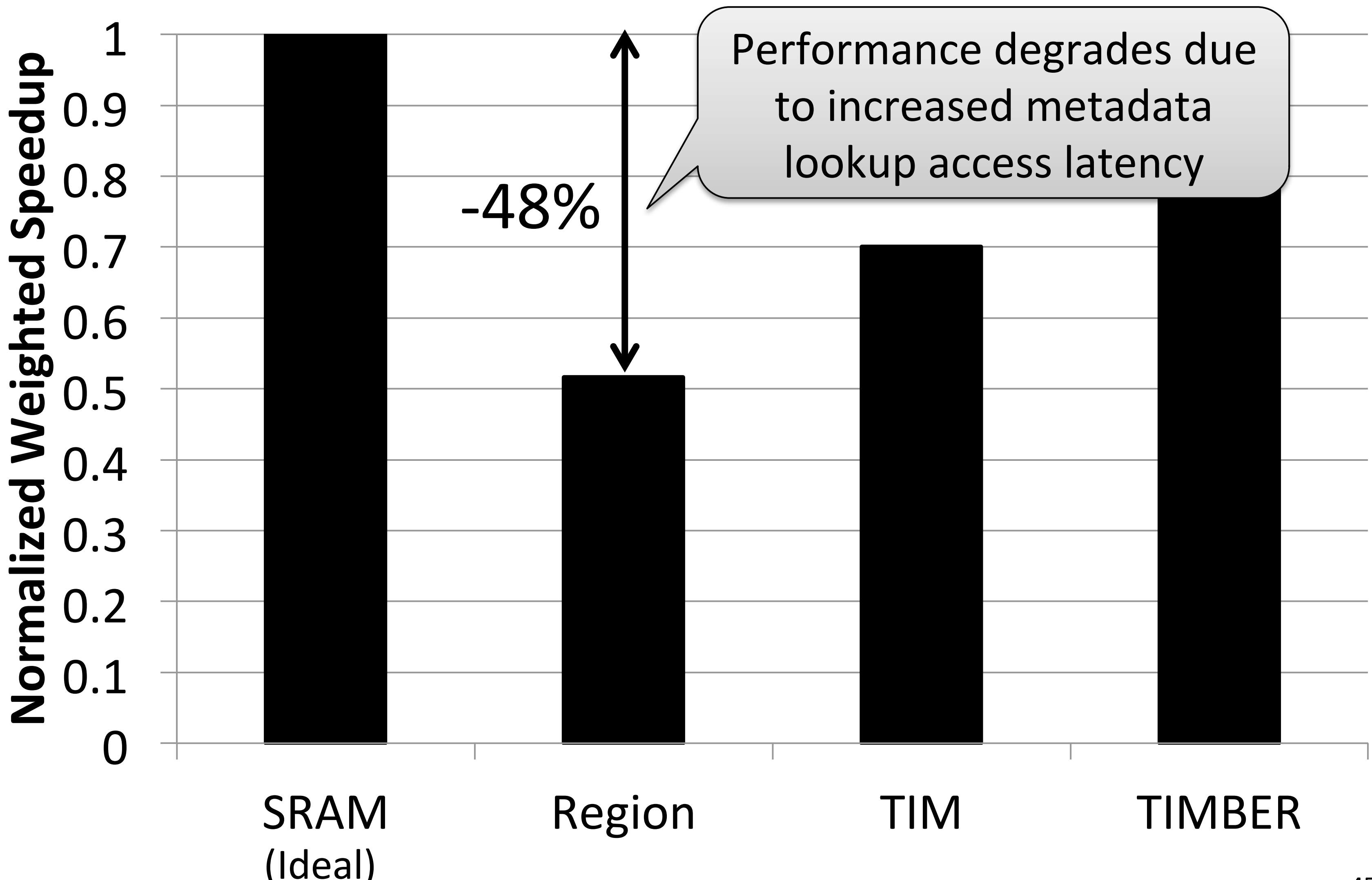
- Memory: 512 MB direct-mapped DRAM, 8 GB PCM
  - 128B caching granularity
  - DRAM row hit (miss): 200 cycles (400 cycles)
  - PCM row hit (clean / dirty miss): 200 cycles (640 / 1840 cycles)

- Evaluated metadata storage techniques
  - All SRAM system (8MB of SRAM)
  - Region metadata storage
  - TIM metadata storage (same row as data)
  - TIMBER, 64-entry direct-mapped (8KB of SRAM)

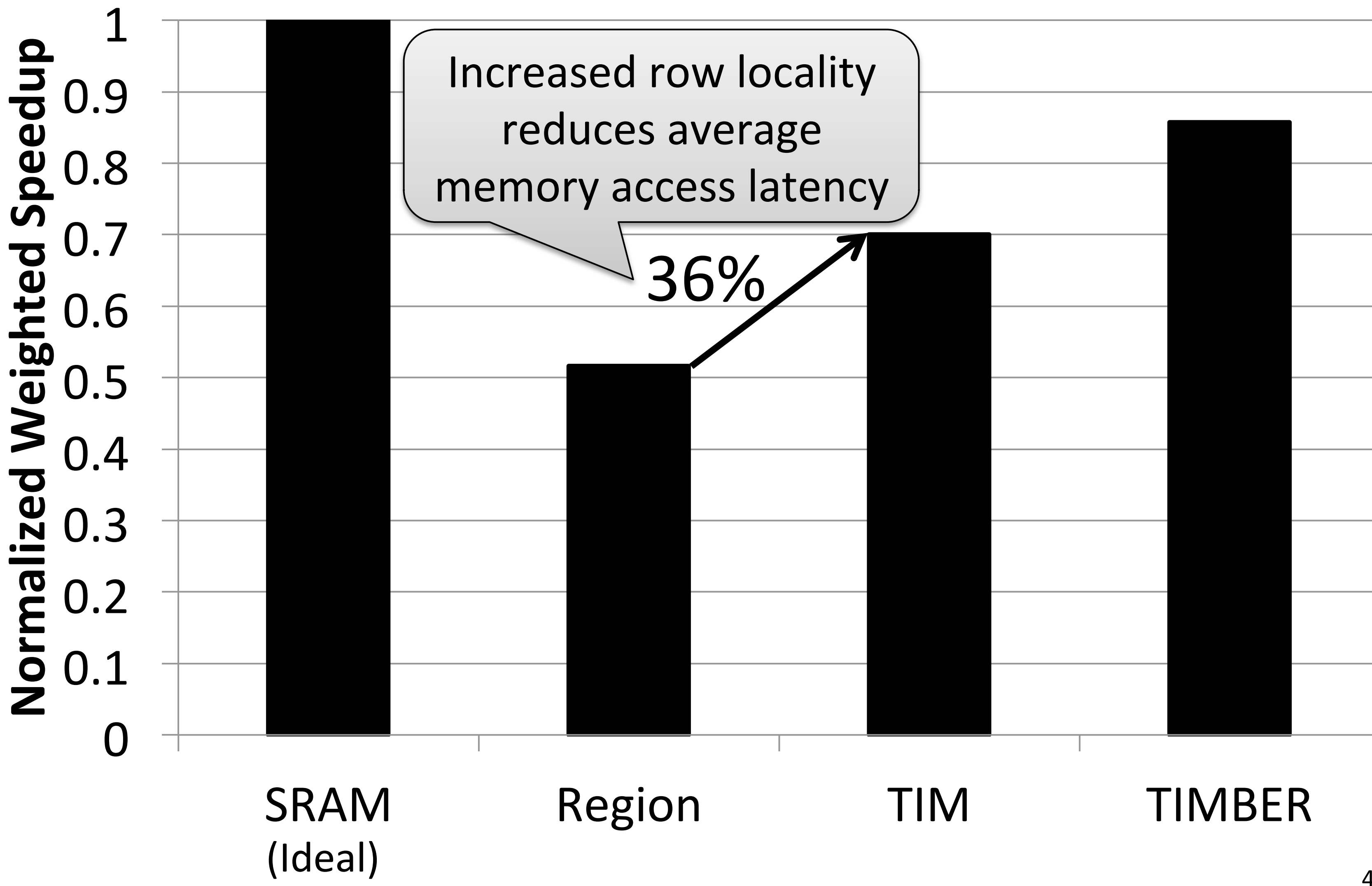
Metadata Storage Performance



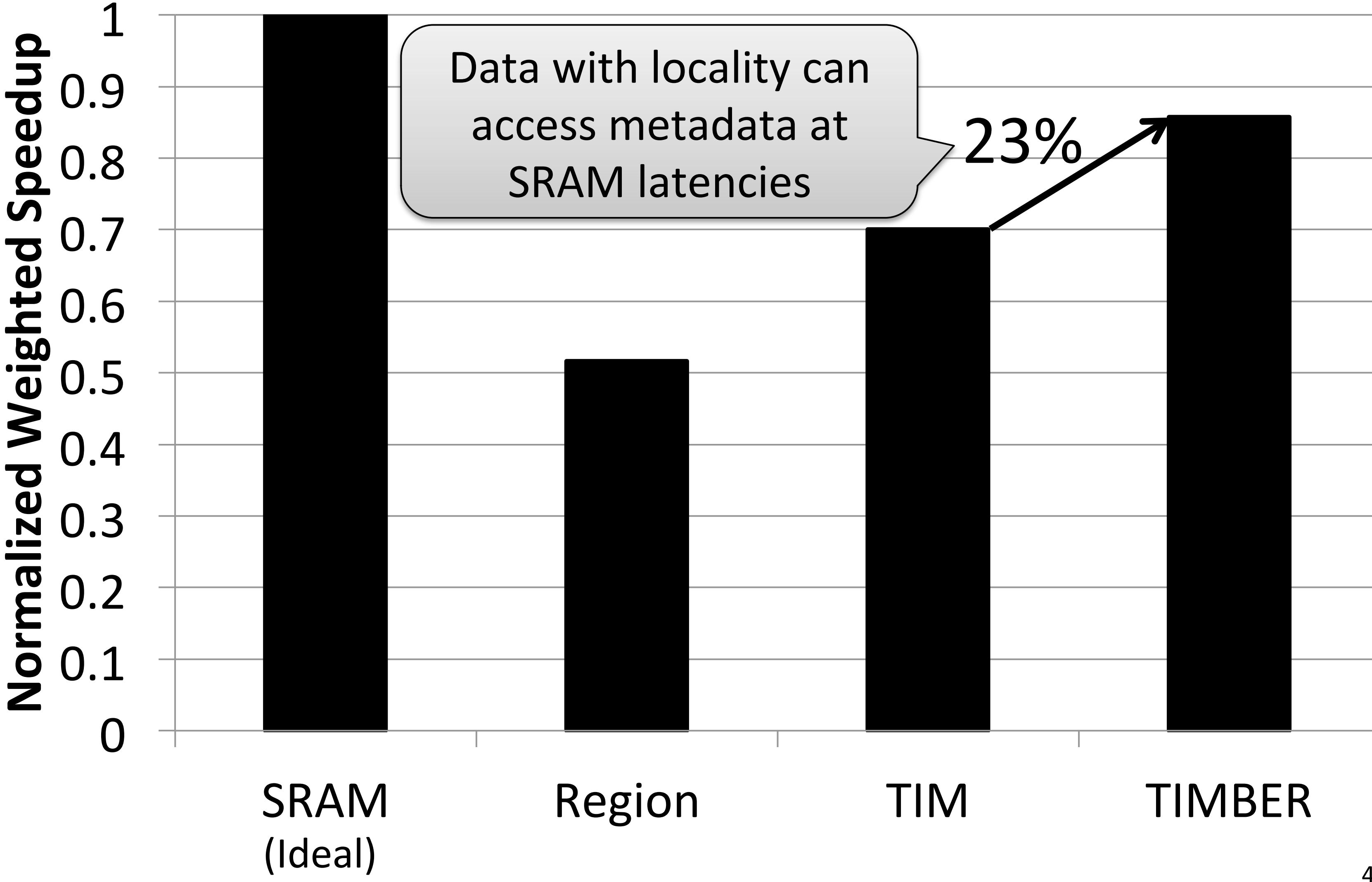
Metadata Storage Performance



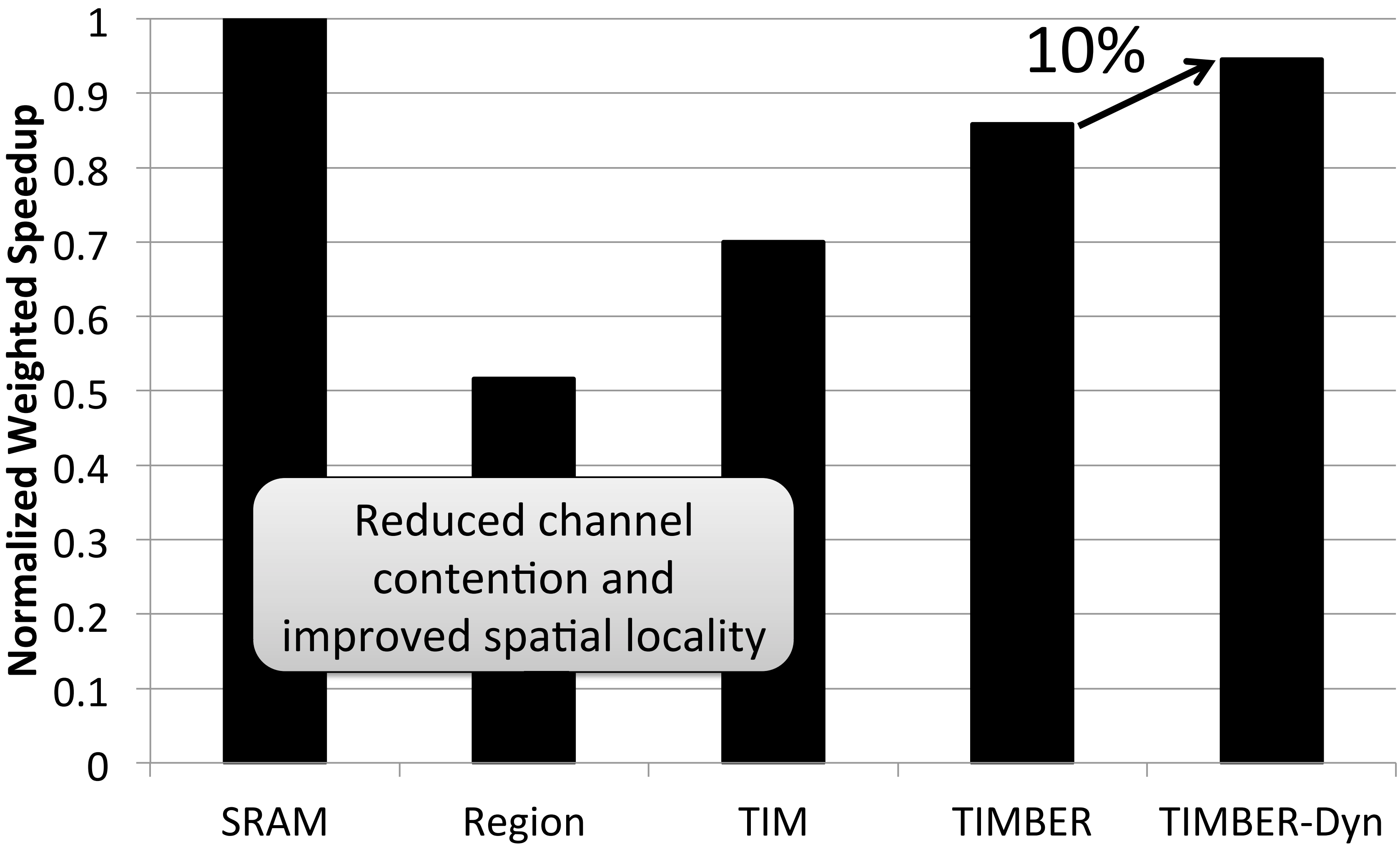
Metadata Storage Performance



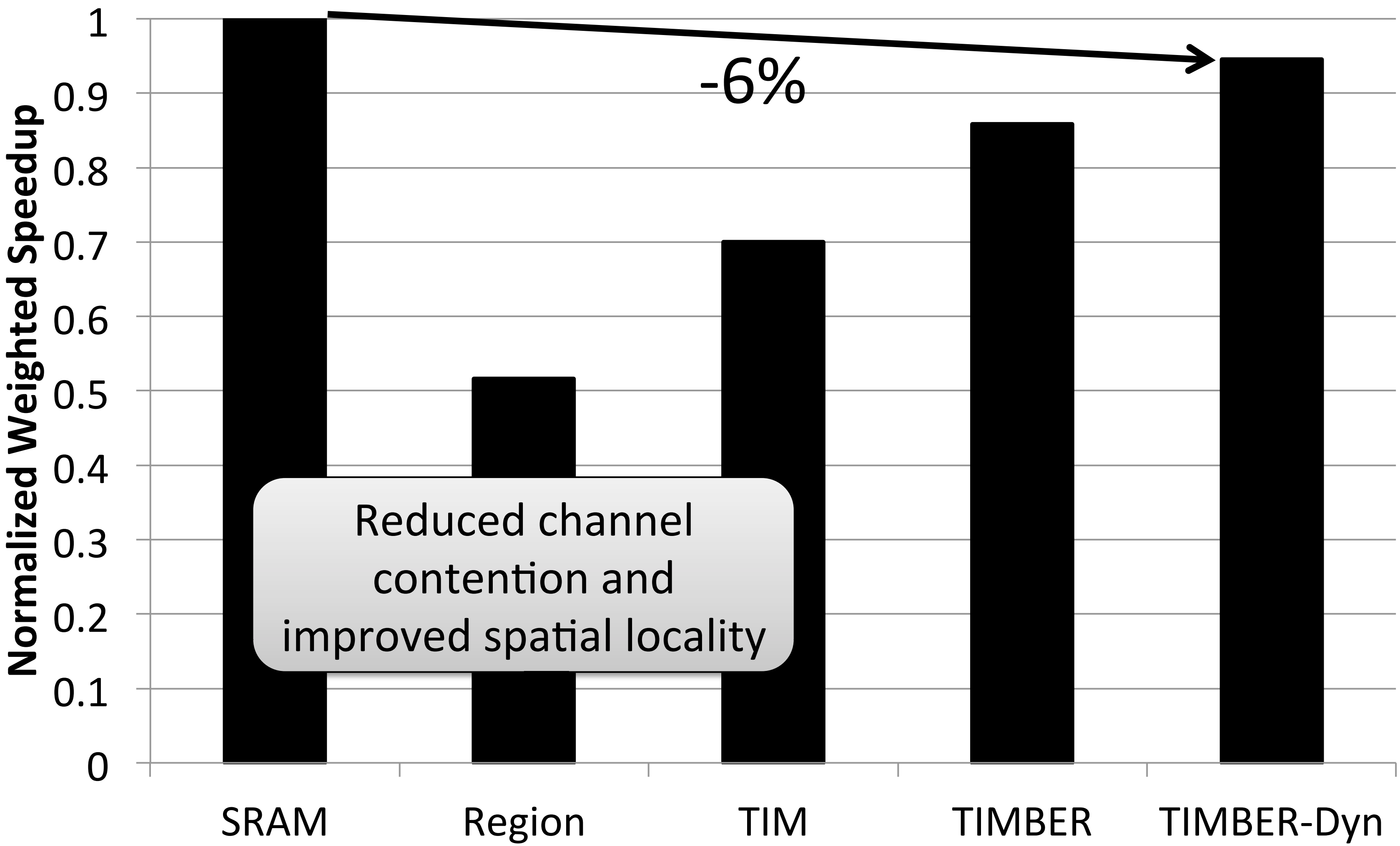
Metadata Storage Performance



Dynamic Granularity Performance

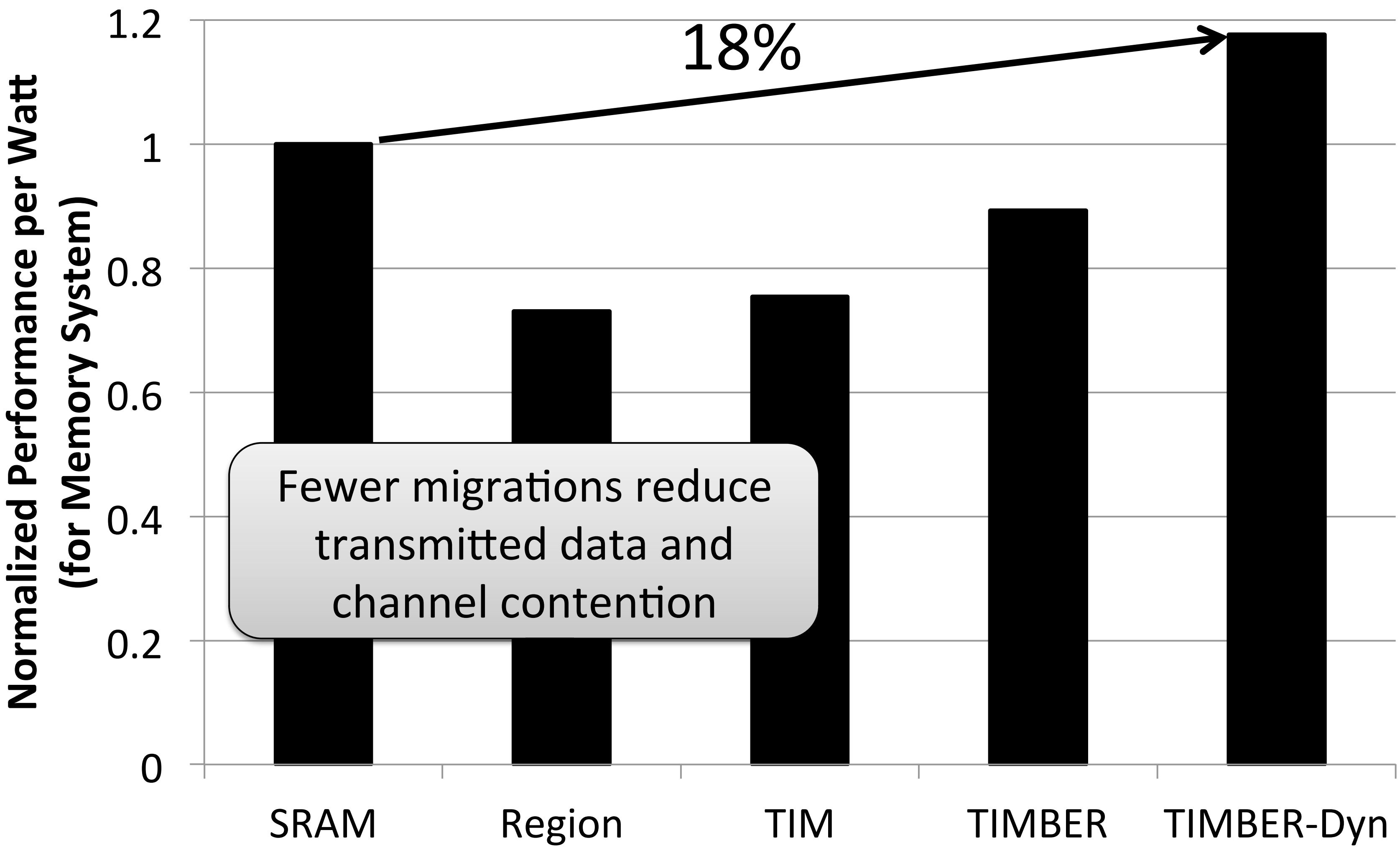


TIMBER Performance



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

TIMBER Energy Efficiency



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

More on Large DRAM Cache Design

- Justin Meza, Jichuan Chang, HanBin Yoon, Onur Mutlu, and Parthasarathy Ranganathan,
  "Enabling Efficient and Scalable Hybrid Memories Using Fine-Granularity DRAM Cache Management"
  IEEE Computer Architecture Letters (CAL), February 2012.

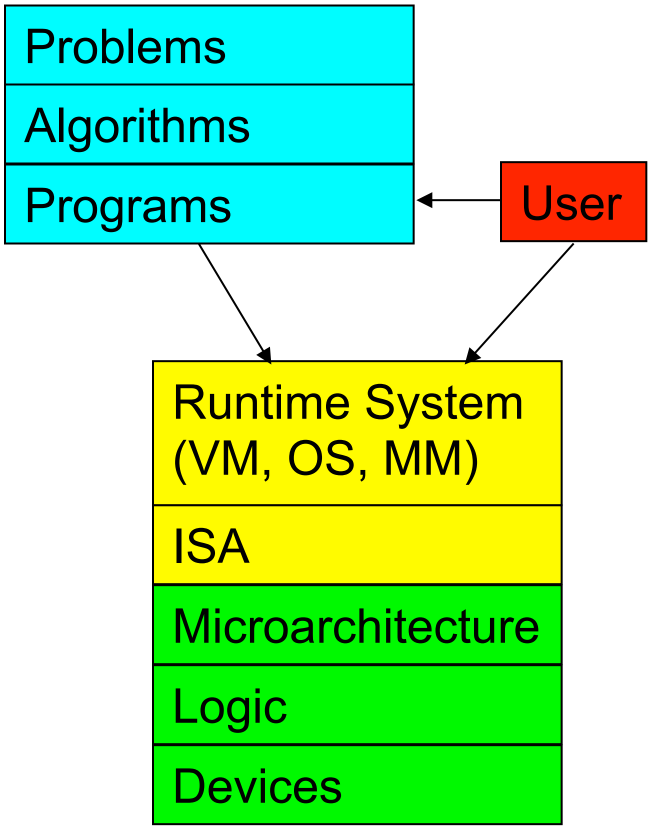
- **Fundamental Latency Trade-offs in Architecting DRAM Caches** (pdf, slides)
  Moinuddin K. Qureshi and Gabriel Loh
  Appears in the International Symposium on Microarchitecture (MICRO) 2012

Enabling and Exploiting NVM: Issues

- Many issues and ideas from technology layer to algorithms layer

- Enabling NVM and hybrid memory
  - How to tolerate errors?
  - How to enable secure operation?
  - How to tolerate performance and power shortcomings?
  - How to minimize cost?

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



Security Challenges of Emerging Technologies

1. Limited endurance → Wearout attacks

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

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

Securing Emerging Memory Technologies

1. Limited endurance → Wearout attacks
    - Better architecting of memory chips to absorb writes
    - Hybrid memory system management
    - Online wearout attack detection

2. Non-volatility → Data persists in memory after powerdown
    → Easy retrieval of privileged or private information
    - Efficient encryption/decryption of whole main memory
    - Hybrid memory system management

3. Multiple bits per cell → Information leakage (via side channel)
    - System design to hide side channel information

Agenda

- Major Trends Affecting Main Memory
- Requirements from an Ideal Main Memory System
- Opportunity: Emerging Memory Technologies
  - Background
  - PCM (or Technology X) as DRAM Replacement
  - Hybrid Memory Systems
- Conclusions
- Discussion

Summary: Memory Scaling (with NVM)

- Main memory scaling problems are a critical bottleneck for system performance, efficiency, and usability

- Solution 1: Tolerate DRAM
- Solution 2: Enable emerging memory technologies
  - Replace DRAM with NVM by architecting NVM chips well
  - Hybrid memory systems with automatic data management

- An exciting topic with many other solution directions & ideas
  - Hardware/software/device cooperation essential
  - Memory, storage, controller, software/app co-design needed
  - Coordinated management of persistent memory and storage
  - Application and hardware cooperative management of NVM

Further: Overview Papers on Two Topics

- Merging of Memory and Storage
  - Justin Meza, Yixin Luo, Samira Khan, Jishen Zhao, Yuan Xie, and Onur Mutlu,
    "A Case for Efficient Hardware-Software Cooperative Management of Storage and Memory"
    *Proceedings of the 5th Workshop on Energy-Efficient Design (**WEED**)*, Tel-Aviv, Israel, June 2013. Slides (pptx) Slides (pdf)

- Flash Memory Scaling
  - Yu Cai, Gulay Yalcin, Onur Mutlu, Erich F. Haratsch, Adrian Cristal, Osman Unsal, and Ken Mai,
    "Error Analysis and Retention-Aware Error Management for NAND Flash Memory"
    *Intel Technology Journal (**ITJ**) Special Issue on Memory Resiliency*, Vol. 17, No. 1, May 2013.

Scalable Many-Core Memory Systems Lecture 4, Topic 2: Emerging Technologies and Hybrid Memories

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

HiPEAC ACACES Summer School 2013

July 18, 2013

Carnegie Mellon

SAFARI

Additional Material

Overview Papers on Two Topics

- Merging of Memory and Storage
  - Justin Meza, Yixin Luo, Samira Khan, Jishen Zhao, Yuan Xie, and Onur Mutlu,
    "A Case for Efficient Hardware-Software Cooperative Management of Storage and Memory"
    *Proceedings of the 5th Workshop on Energy-Efficient Design (**WEED**)*, Tel-Aviv, Israel, June 2013. Slides (pptx) Slides (pdf)

- Flash Memory Scaling
  - Yu Cai, Gulay Yalcin, Onur Mutlu, Erich F. Haratsch, Adrian Cristal, Osman Unsal, and Ken Mai,
    "Error Analysis and Retention-Aware Error Management for NAND Flash Memory"
    *Intel Technology Journal (**ITJ**) Special Issue on Memory Resiliency*, Vol. 17, No. 1, May 2013.

Merging of Memory and Storage: Persistent Memory Managers

A Case for Efficient Hardware/Software Cooperative Management of Storage and Memory

Justin Meza*, Yixin Luo*, Samira Khan*†, Jishen Zhao§, Yuan Xie§‡, and Onur Mutlu*

*Carnegie Mellon University
§Pennsylvania State University
†Intel Labs ‡AMD Research

SAFARI

Carnegie Mellon

Overview

- Traditional systems have a two-level storage model
  - Access **volatile** data in memory with a **load/store** interface
  - Access **persistent** data in storage with a **file system** interface
  - Problem: Operating system (OS) and file system (FS) code and buffering for storage lead to energy and performance inefficiencies

- Opportunity: New non-volatile memory (NVM) technologies can help provide fast (similar to DRAM), persistent storage (similar to Flash)
  - Unfortunately, OS and FS code can easily become energy efficiency and performance bottlenecks if we keep the traditional storage model

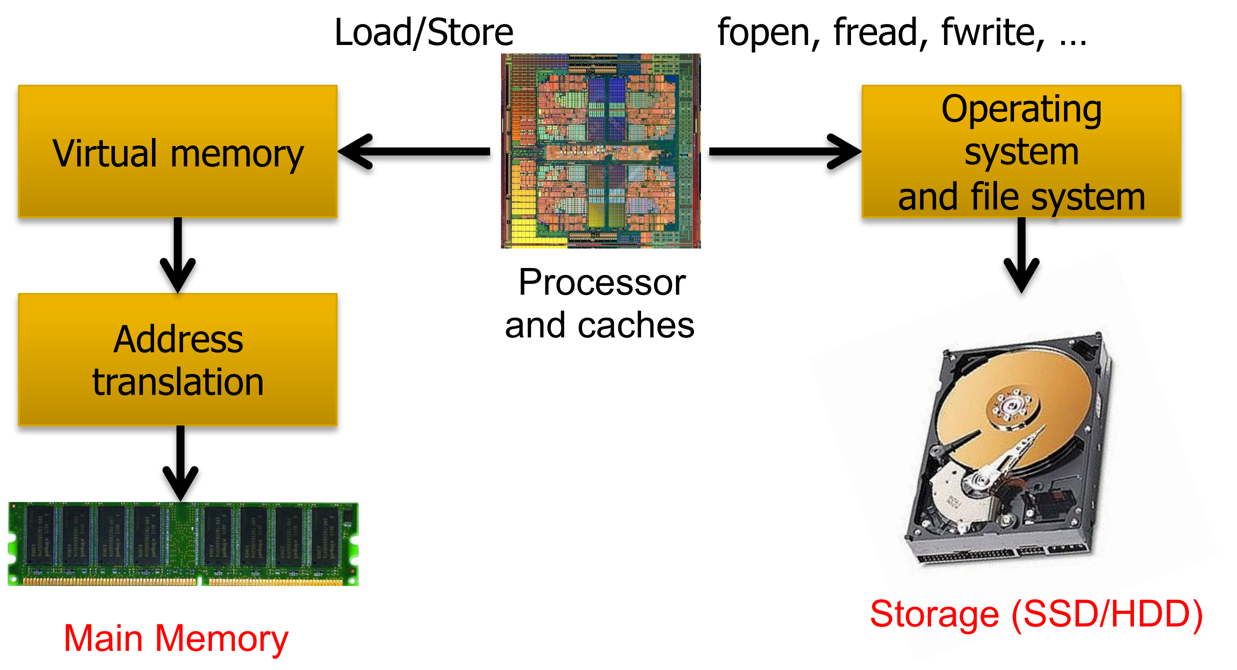
- This work: makes a case for hardware/software cooperative management of storage and memory within a single-level
  - We describe the idea of a Persistent Memory Manager (PMM) for efficiently coordinating storage and memory, and quantify its benefit
  - And, examine questions and challenges to address to realize PMM

Talk Outline

- Background: Storage and Memory Models
- Motivation: Eliminating Operating/File System Bottlenecks
- Our Proposal: Hardware/Software Coordinated Management of Storage and Memory
  - Opportunities and Benefits
- Evaluation Methodology
- Evaluation Results
- Related Work
- New Questions and Challenges
- Conclusions

A Tale of Two Storage Levels

- Traditional systems use a two-level storage model
  - Volatile data is stored in DRAM
  - Persistent data is stored in HDD and Flash
- Accessed through two vastly different interfaces



A Tale of Two Storage Levels

- Two-level storage arose in systems due to the widely different access latencies and methods of the commodity storage devices
  - Fast, low capacity, volatile DRAM → working storage
  - Slow, high capacity, non-volatile hard disk drives → persistent storage

- Data from slow storage media is buffered in fast DRAM
  - After that it can be manipulated by programs → programs cannot directly access persistent storage
  - It is the programmer's job to translate this data between the two formats of the two-level storage (files and data structures)

- Locating, transferring, and translating data and formats between the two levels of storage can waste significant energy and performance

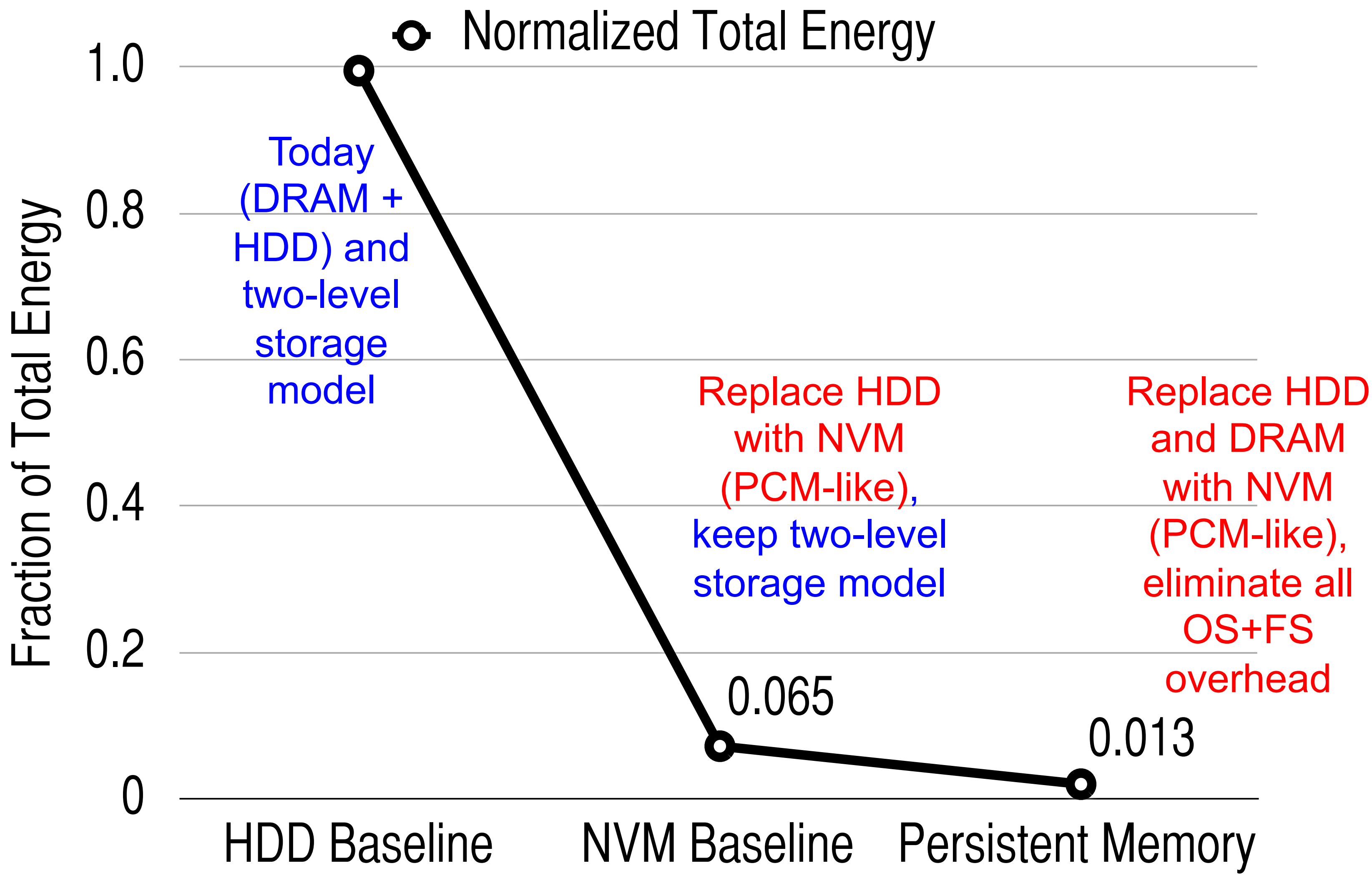
Opportunity: New Non-Volatile Memories

- Emerging memory technologies provide the potential for unifying storage and memory (e.g., Phase-Change, STT-RAM, RRAM)
  - Byte-addressable (can be accessed like DRAM)
  - Low latency (comparable to DRAM)
  - Low power (idle power better than DRAM)
  - High capacity (closer to Flash)
  - Non-volatile (can enable persistent storage)
  - May have limited endurance (but, better than Flash)

- Can provide fast access to ***both*** volatile data and persistent storage

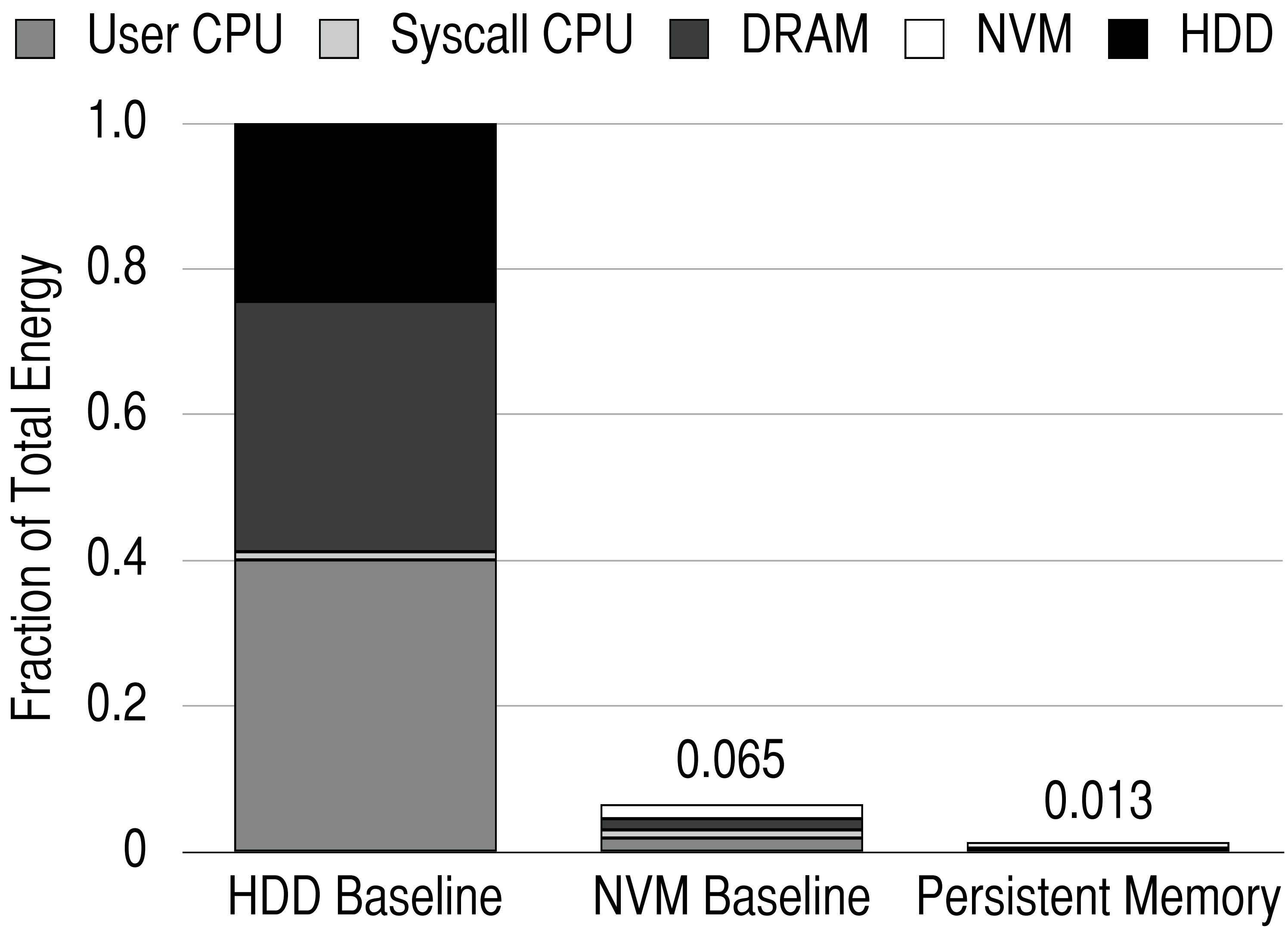
- Question: if such devices are used, is it efficient to keep a two-level storage model?

Eliminating Traditional Storage Bottlenecks



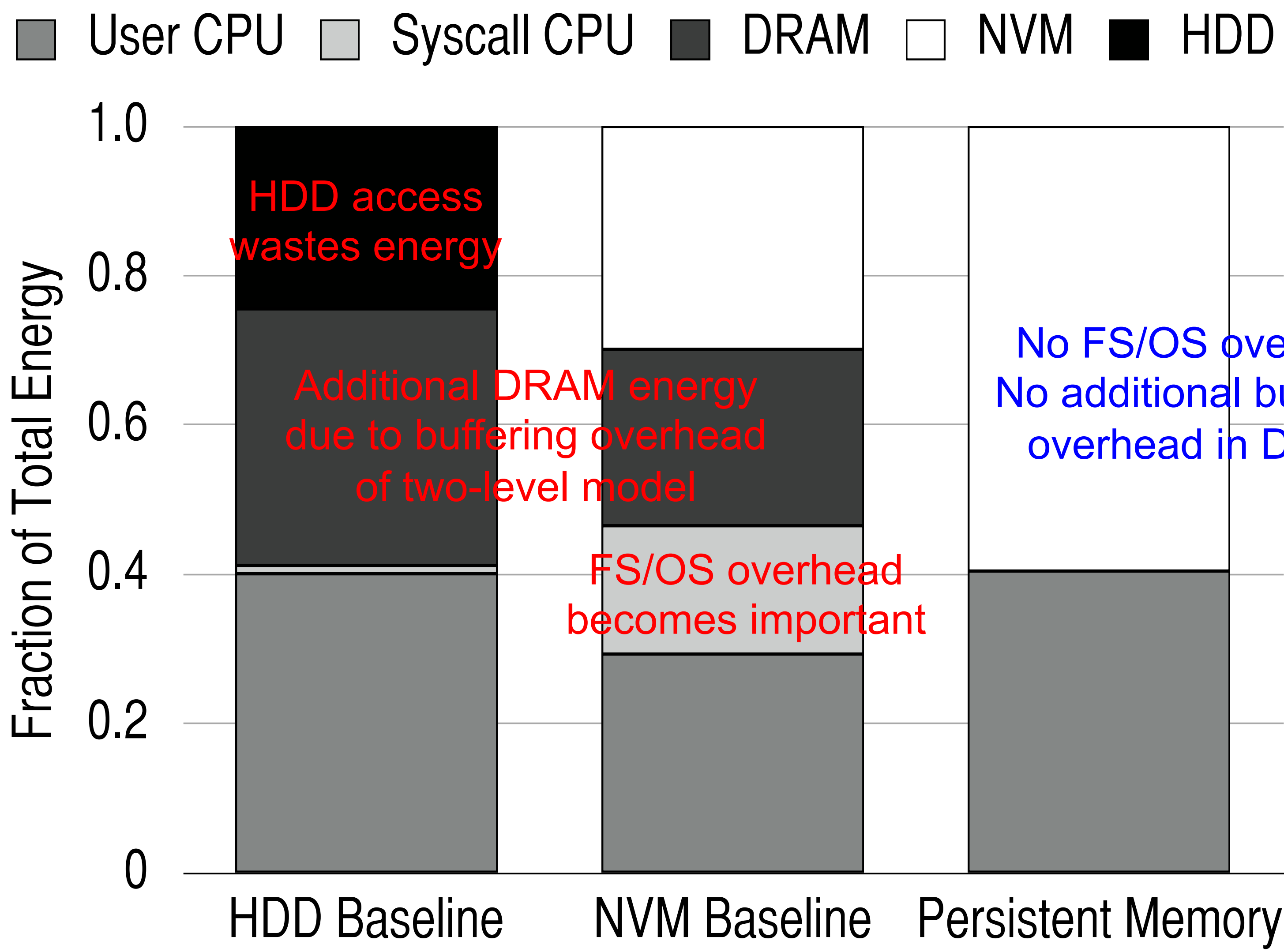
Results for PostMark

Eliminating Traditional Storage Bottlenecks



Results for PostMark

Where is Energy Spent in Each Model?



Results for PostMark

Outline

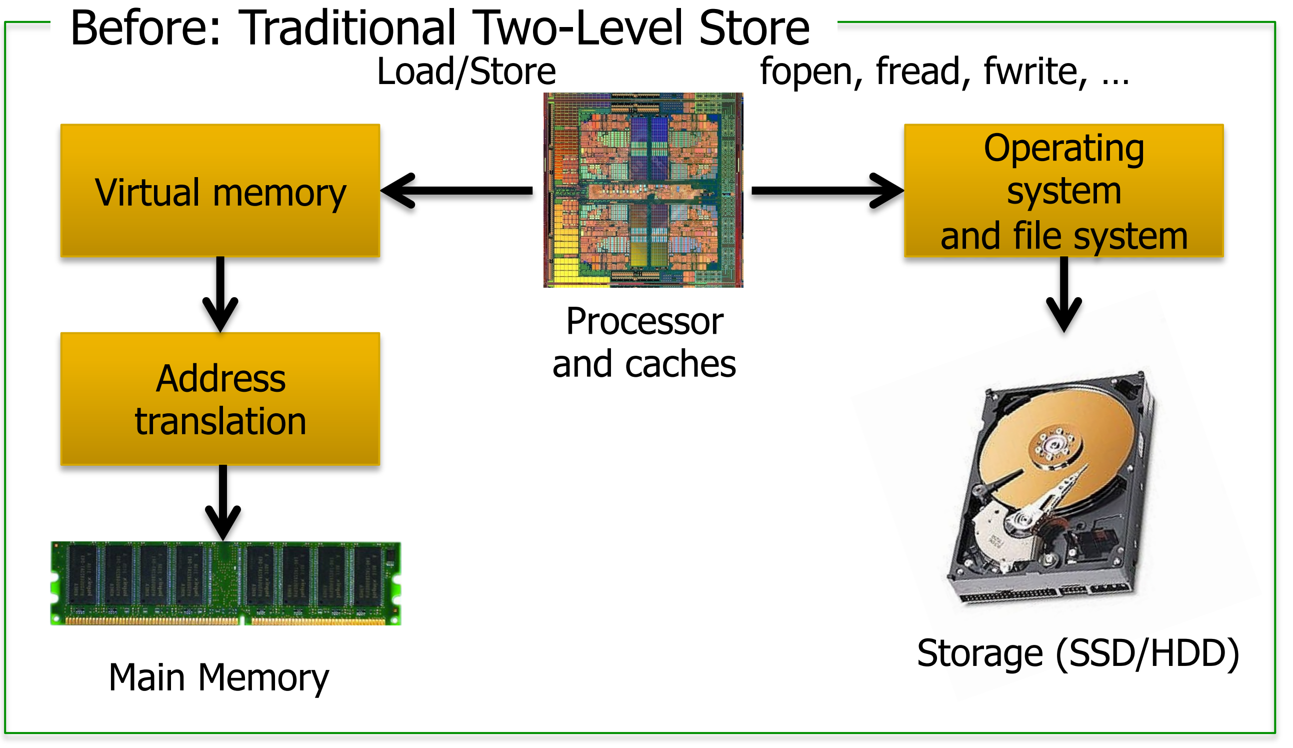
- Background: Storage and Memory Models
- Motivation: Eliminating Operating/File System Bottlenecks
- Our Proposal: Hardware/Software Coordinated Management of Storage and Memory
  - Opportunities and Benefits
- Evaluation Methodology
- Evaluation Results
- Related Work
- New Questions and Challenges
- Conclusions

Our Proposal: Coordinated HW/SW Memory and Storage Management

- Goal: Unify memory and storage to eliminate wasted work to locate, transfer, and translate data
  - Improve both energy and performance
  - Simplify programming model as well

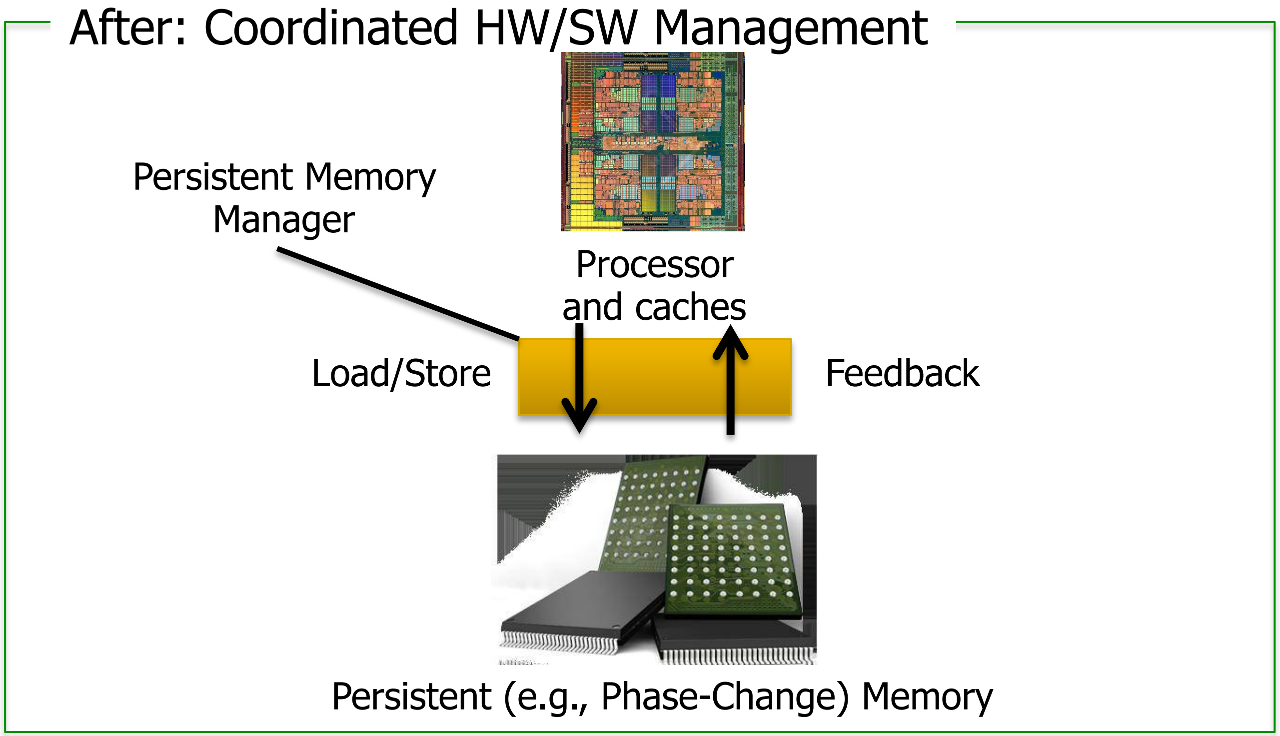
Our Proposal: Coordinated HW/SW Memory and Storage Management

- Goal: Unify memory and storage to eliminate wasted work to locate, transfer, and translate data
  - Improve both energy and performance
  - Simplify programming model as well



Our Proposal: Coordinated HW/SW Memory and Storage Management

- Goal: Unify memory and storage to eliminate wasted work to locate, transfer, and translate data
  - Improve both energy and performance
  - Simplify 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

The Persistent Memory Manager

- Persistent Memory Manager
  - Exposes a load/store interface to access persistent data
  - Manages data placement, location, persistence, security
  - Manages metadata storage and retrieval
  - Exposes hooks and interfaces for system software

- Example program manipulating a persistent object:

```
int main(void) {
  // data in file.dat is persistent
  FILE myData = "file.dat";
  myData = new int[64];
}
void updateValue(int n, int value) {
  FILE myData = "file.dat";
  myData[n] = value; // value is persistent
}
```

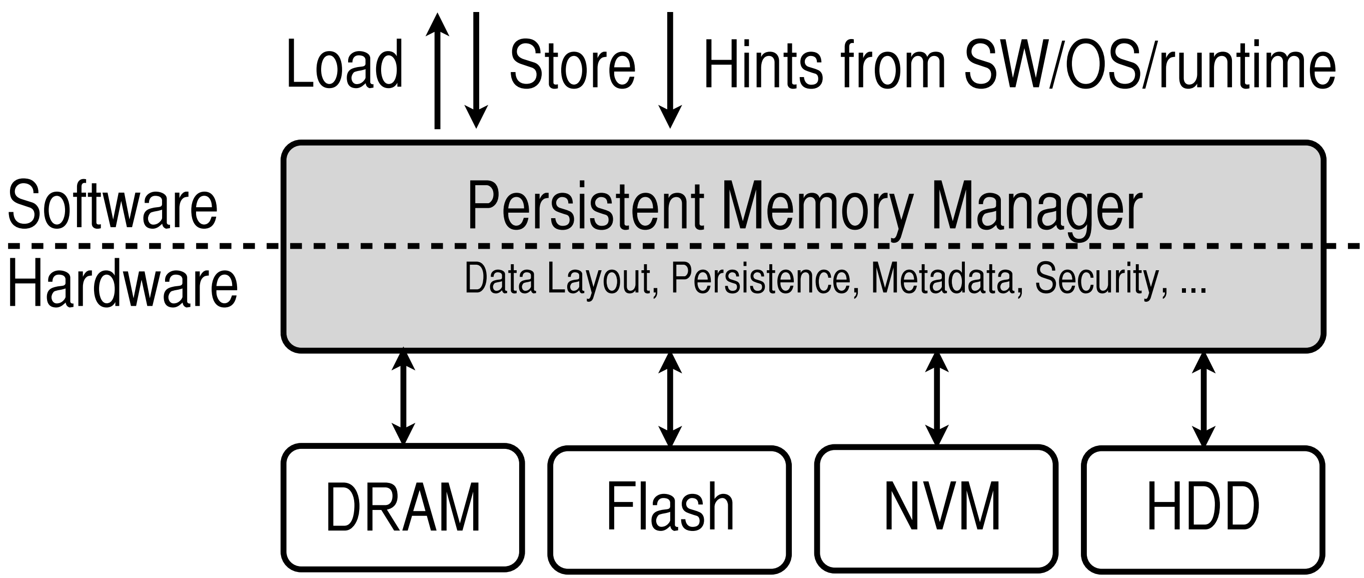
Create persistent object and its handle

Allocate a persistent array and assign

Load/store interface

Putting Everything Together

```
int main(void) {
  // data in file.dat is persistent
  FILE myData = "file.dat";
  myData = new int[64];
}
void updateValue(int n, int value) {
  FILE myData = "file.dat";
  myData[n] = value; // value is persistent
}
```



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

Outline

- Background: Storage and Memory Models
- Motivation: Eliminating Operating/File System Bottlenecks
- Our Proposal: Hardware/Software Coordinated Management of Storage and Memory
  - Opportunities and Benefits
- Evaluation Methodology
- Evaluation Results
- Related Work
- New Questions and Challenges
- Conclusions

Opportunities and Benefits

- We've identified at least five opportunities and benefits of a unified storage/memory system that gets rid of the two-level model:
  1. Eliminating system calls for file operations
  2. Eliminating file system operations
  3. Efficient data mapping/location among heterogeneous devices
  4. Providing security and reliability in persistent memories
  5. Hardware/software cooperative data management

Eliminating System Calls for File Operations

- A persistent memory can expose a large, linear, persistent address space
  - Persistent storage objects can be directly manipulated with load/ store operations

- This eliminates the need for layers of operating system code
  - Typically used for calls like `open`, `read`, and `write`

- Also eliminates OS file metadata
  - File descriptors, file buffers, and so on

Eliminating File System Operations

- Locating files is traditionally done using a *file system*
  - Runs code and traverses structures in software to locate files

- Existing hardware structures for locating data in virtual memory can be extended and adapted to meet the needs of persistent memories
  - Memory Management Units (MMUs), which map virtual addresses to physical addresses
  - Translation Lookaside Buffers (TLBs), which cache mappings of virtual-to-physical address translations

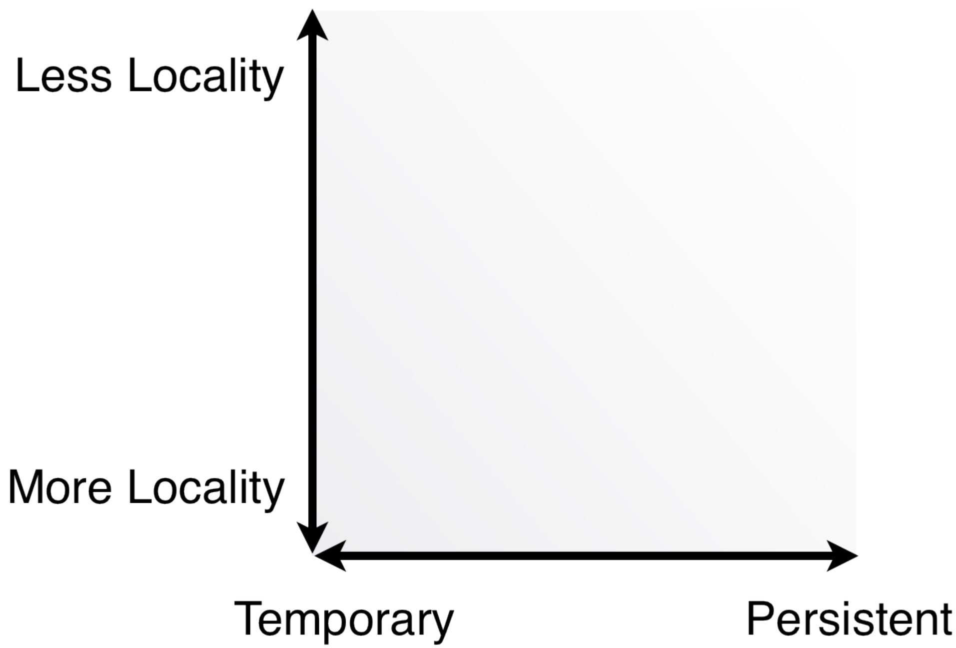
- Potential to eliminate file system code
- At the cost of additional hardware overhead to handle persistent data storage

Efficient Data Mapping among Heterogeneous Devices

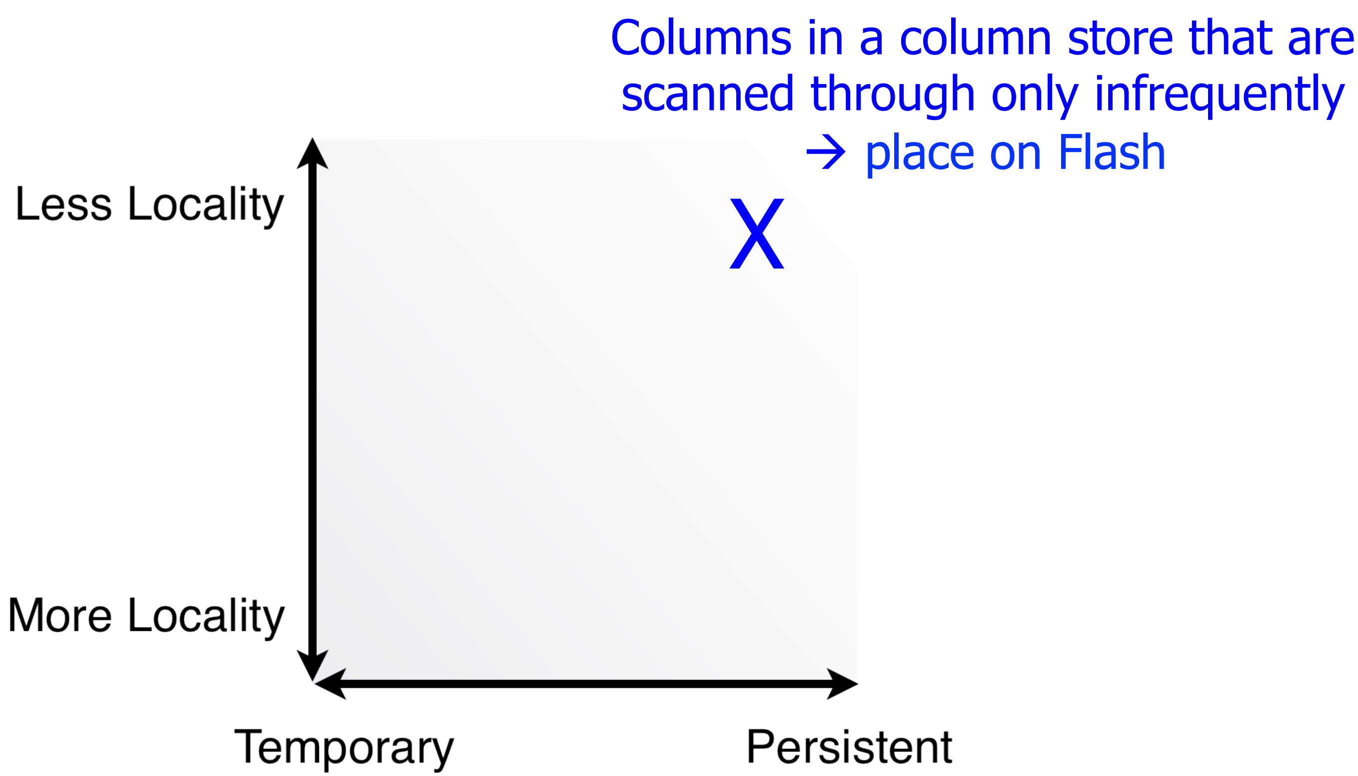
- A persistent memory exposes a large, persistent address space
  - But it may use many different devices to satisfy this goal
  - From fast, low-capacity volatile DRAM to slow, high-capacity non-volatile HDD or Flash
  - And other NVM devices in between

- Performance and energy can benefit from good placement of data among these devices
  - Utilizing the strengths of each device and avoiding their weaknesses, if possible
  - For example, consider two important application characteristics: locality and persistence

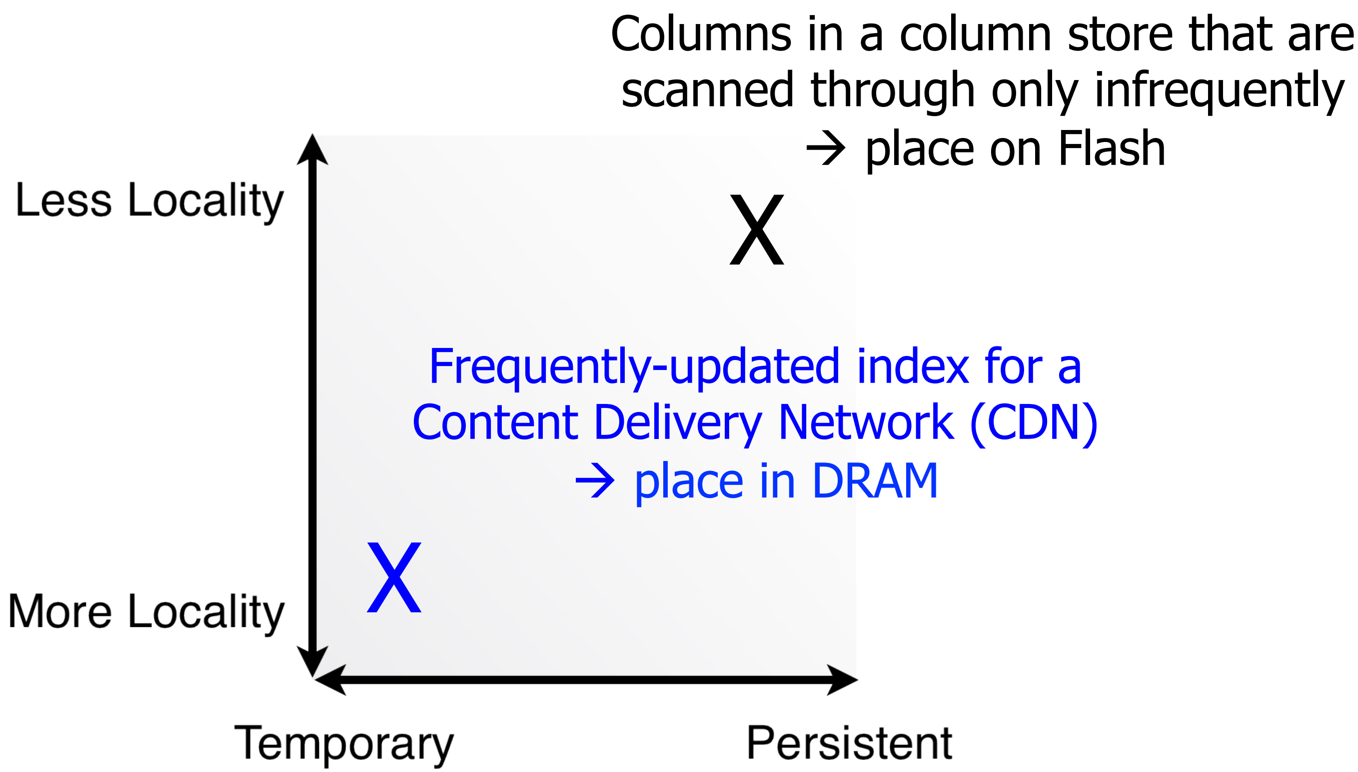
Efficient Data Mapping among Heterogeneous Devices



Efficient Data Mapping among Heterogeneous Devices



Efficient Data Mapping among Heterogeneous Devices



Applications or system software can provide hints for data placement

Providing Security and Reliability

- A persistent memory deals with data at the granularity of bytes and not necessarily files
  - Provides the opportunity for much finer-grained security and protection than traditional two-level storage models provide/afford
  - Need efficient techniques to avoid large metadata overheads

- A persistent memory can improve application reliability by ensuring updates to persistent data are less vulnerable to failures
  - Need to ensure that changes to copies of persistent data placed in volatile memories become persistent

HW/SW Cooperative Data Management

- Persistent memories can expose hooks and interfaces to applications, the OS, and runtimes
  - Have the potential to provide improved system robustness and efficiency than by managing persistent data with either software or hardware alone

- Can enable fast checkpointing and reboots, improve application reliability by ensuring persistence of data
  - How to redesign availability mechanisms to take advantage of these?

- Persistent locks and other persistent synchronization constructs can enable more robust programs and systems

Quantifying Persistent Memory Benefits

- We have identified several opportunities and benefits of using persistent memories without the traditional two-level store model

- We will next quantify:
  - How do persistent memories affect system performance?
  - How much energy reduction is possible?
  - Can persistent memories achieve these benefits despite additional access latencies to the persistent memory manager?

Outline

- Background: Storage and Memory Models
- Motivation: Eliminating Operating/File System Bottlenecks
- Our Proposal: Hardware/Software Coordinated Management of Storage and Memory
  - Opportunities and Benefits
- Evaluation Methodology
- Evaluation Results
- Related Work
- New Questions and Challenges
- Conclusions

Evaluation Methodology

- Hybrid real system / simulation-based approach
  - System calls are executed on host machine (functional correctness) and timed to accurately model their latency in the simulator
  - Rest of execution is simulated in Multi2Sim (enables hardware-level exploration)
- Power evaluated using McPAT and memory power models
- 16 cores, 4-wide issue, 128-entry instruction window, 1.6 GHz
- Volatile memory: 4GB DRAM, 4KB page size, 100-cycle latency
- Persistent memory
  - HDD (measured): 4ms seek latency, 6Gbps bus rate
  - NVM: (modeled after PCM) 4KB page size, 160-/480-cycle (read/write) latency

Evaluated Systems

- HDD Baseline (HB)
  - Traditional system with volatile DRAM memory and persistent HDD storage
  - Overheads of operating system and file system code and buffering
- HDD without OS/FS (HW)
  - Same as HDD Baseline, but with the ideal elimination of all OS/FS overheads
  - System calls take 0 cycles (but HDD access takes normal latency)
- NVM Baseline (NB)
  - Same as HDD Baseline, but HDD is replaced with NVM
  - Still has OS/FS overheads of the two-level storage model
- Persistent Memory (PM)
  - Uses only NVM (no DRAM) to ensure full-system persistence
  - All data accessed using loads and stores
  - Does not waste energy on system calls
  - Data is manipulated directly on the NVM device

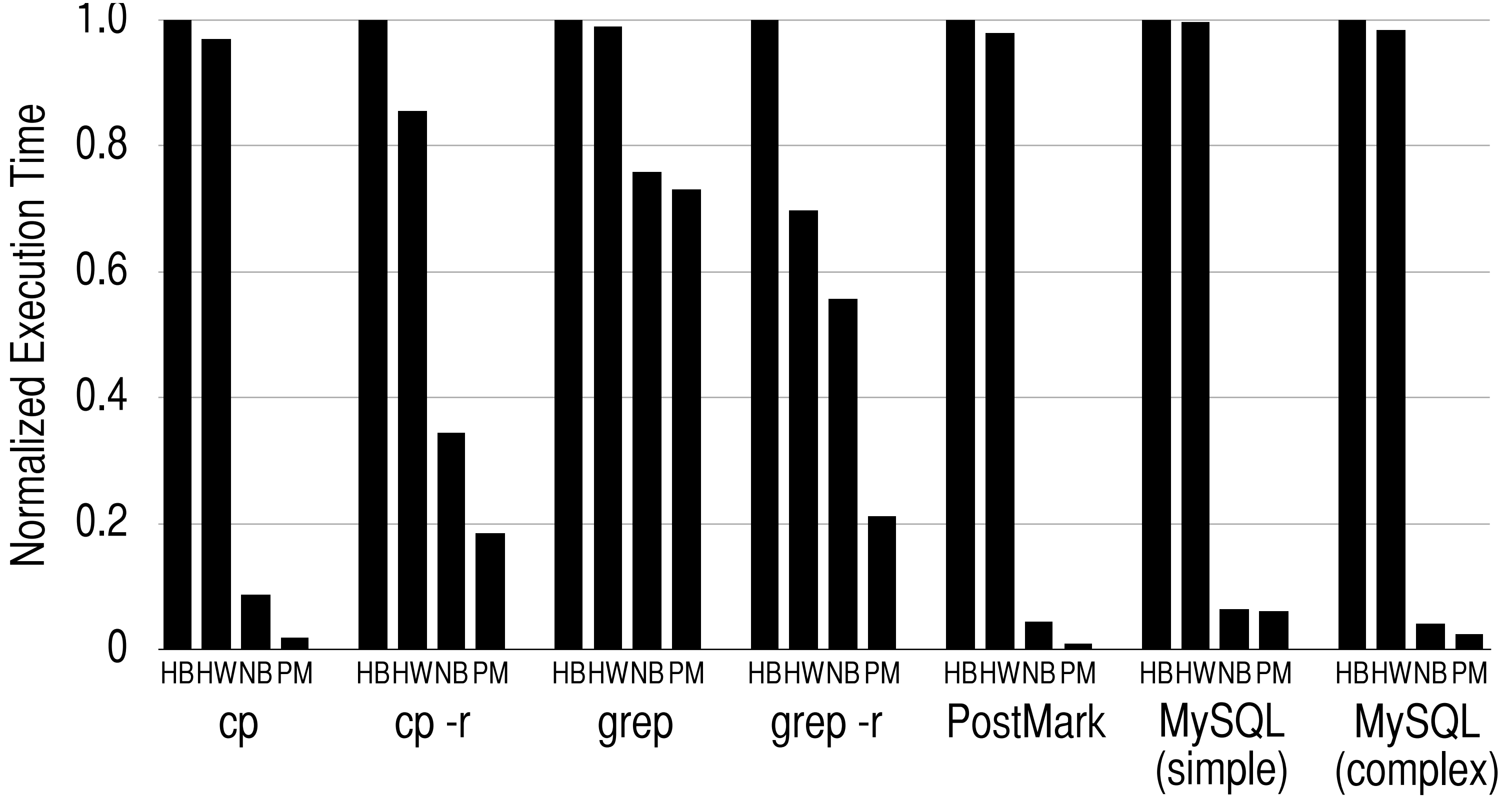
Evaluated Workloads

- Unix utilities that manipulate files
  - cp: copy a large file from one location to another
  - cp –r: copy files in a directory tree from one location to another
  - grep: search for a string in a large file
  - grep –r: search for a string recursively in a directory tree

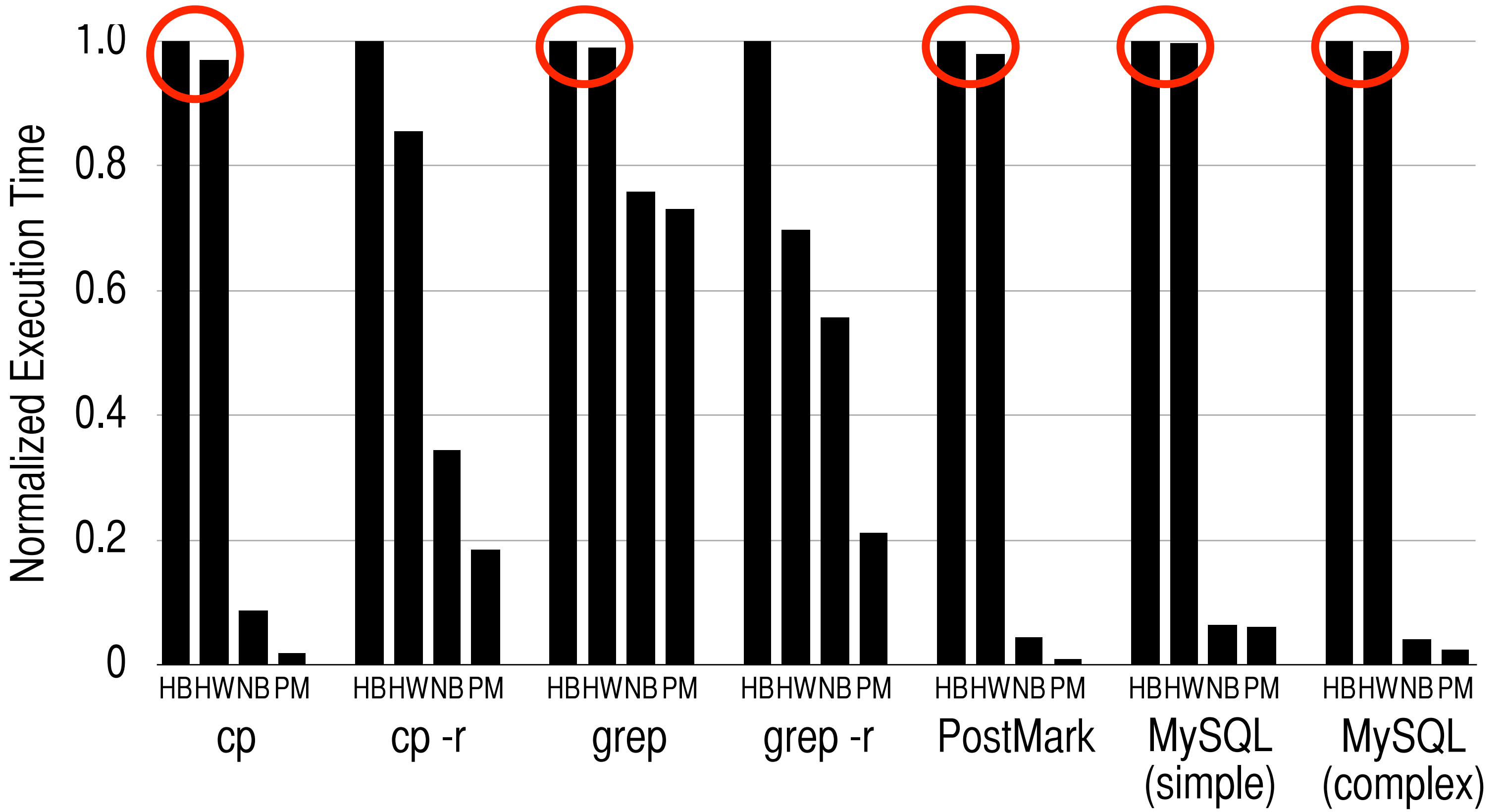
- PostMark: an I/O-intensive benchmark from NetApp
  - Emulates typical access patterns for email, news, web commerce

- MySQL Server: a popular database management system
  - OLTP-style queries generated by Sysbench
  - MySQL (simple): single, random read to an entry
  - MySQL (complex): reads/writes 1 to 100 entries per transaction

Performance Results

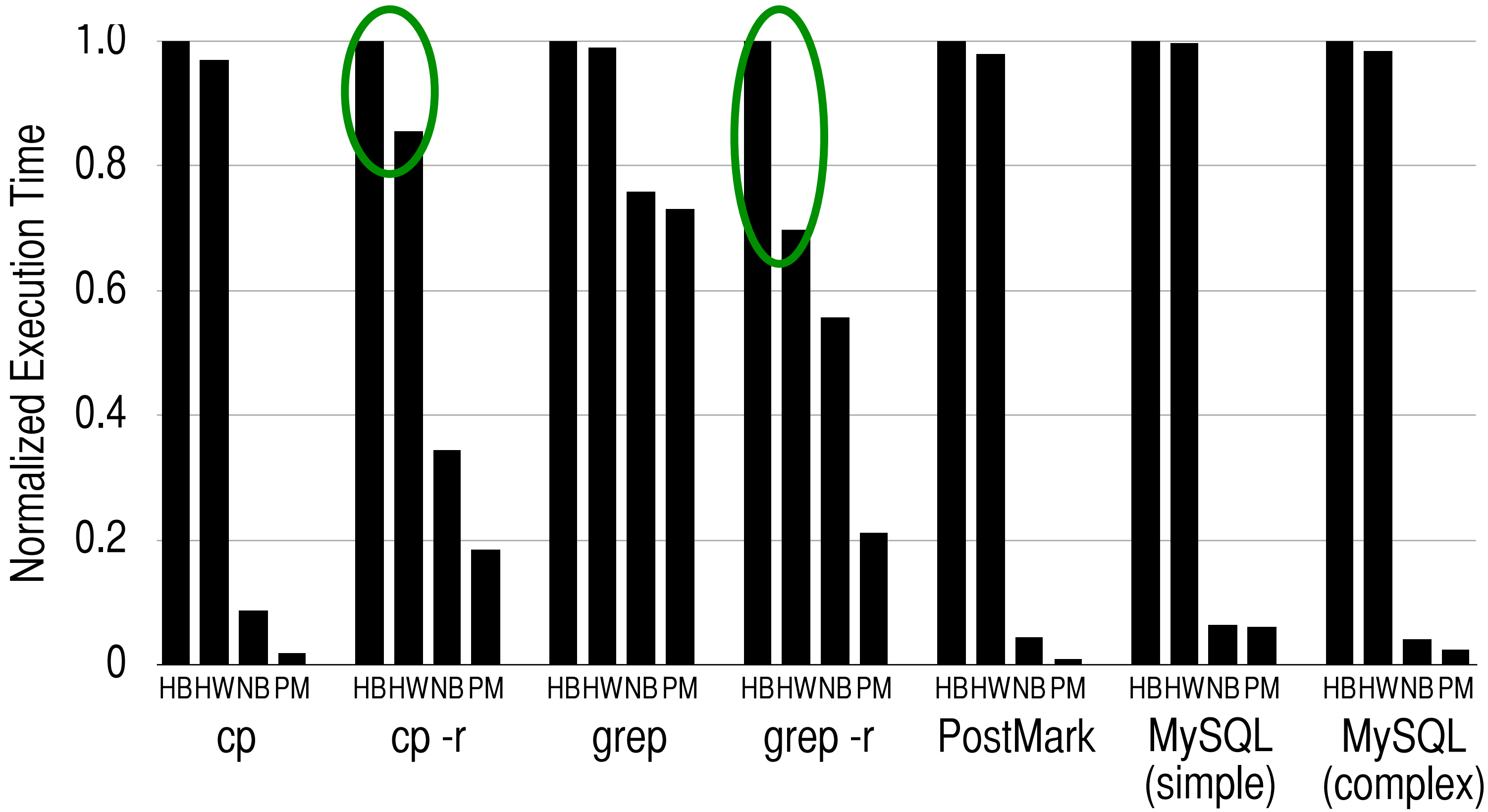


Performance Results: HDD w/o OS/FS



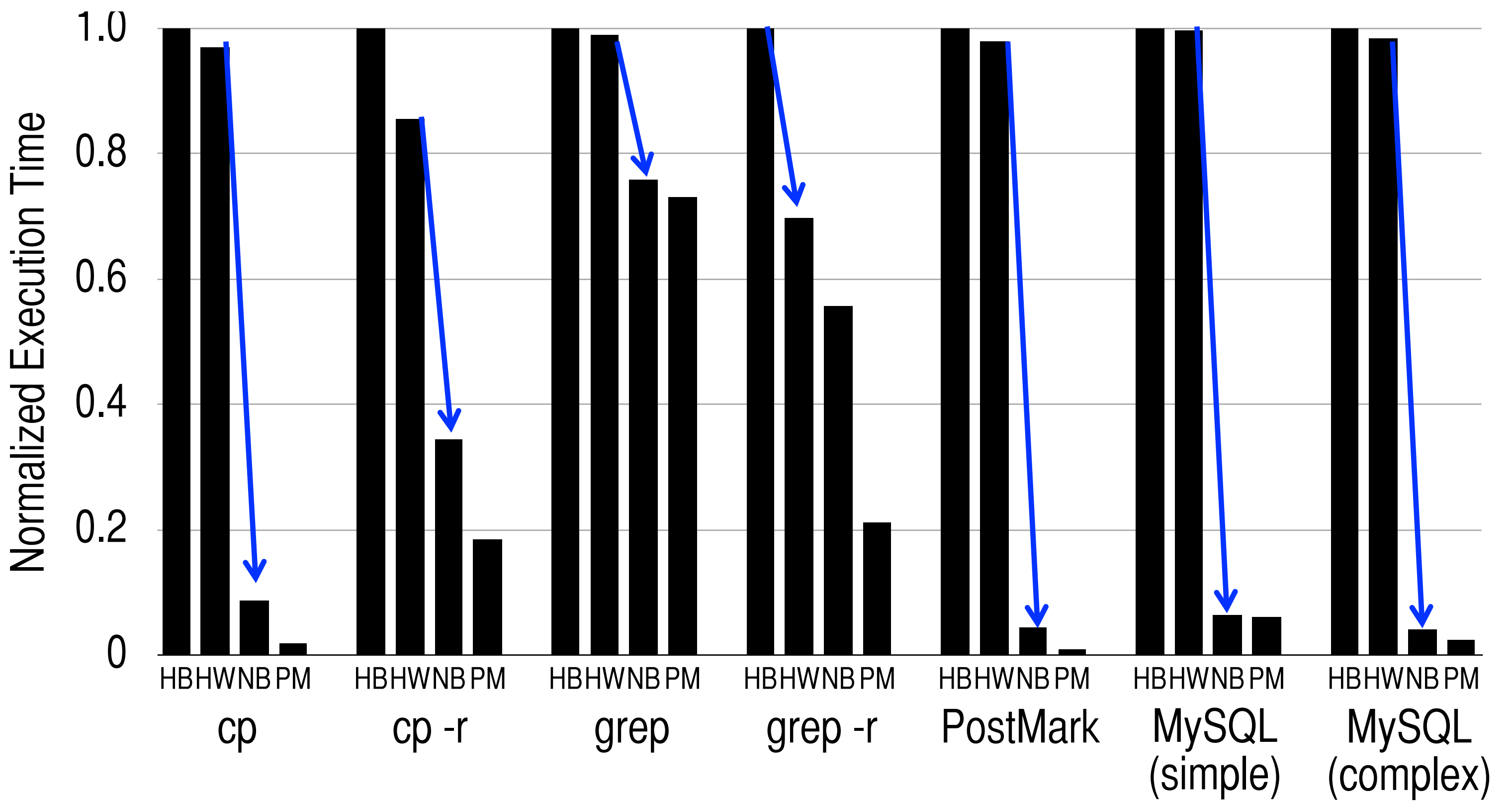
For HDD-based systems, eliminating OS/FS overheads typically leads to small performance improvements → execution time dominated by HDD access latency

Performance Results: HDD w/o OS/FS



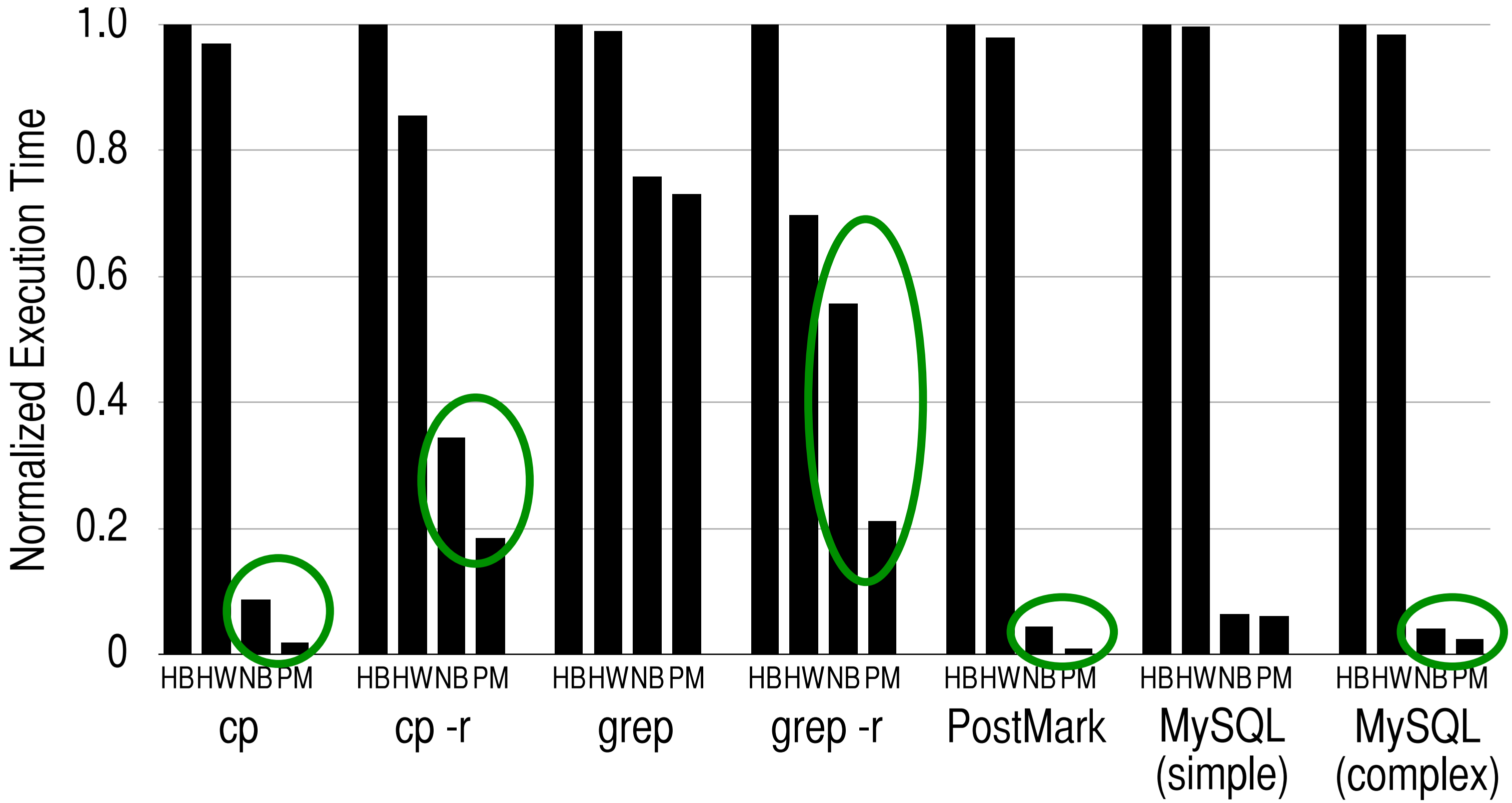
Though, for more complex file system operations like directory traversal (seen with cp -r and grep -r), eliminating the OS/FS overhead improves performance

Performance Results: HDD to NVM



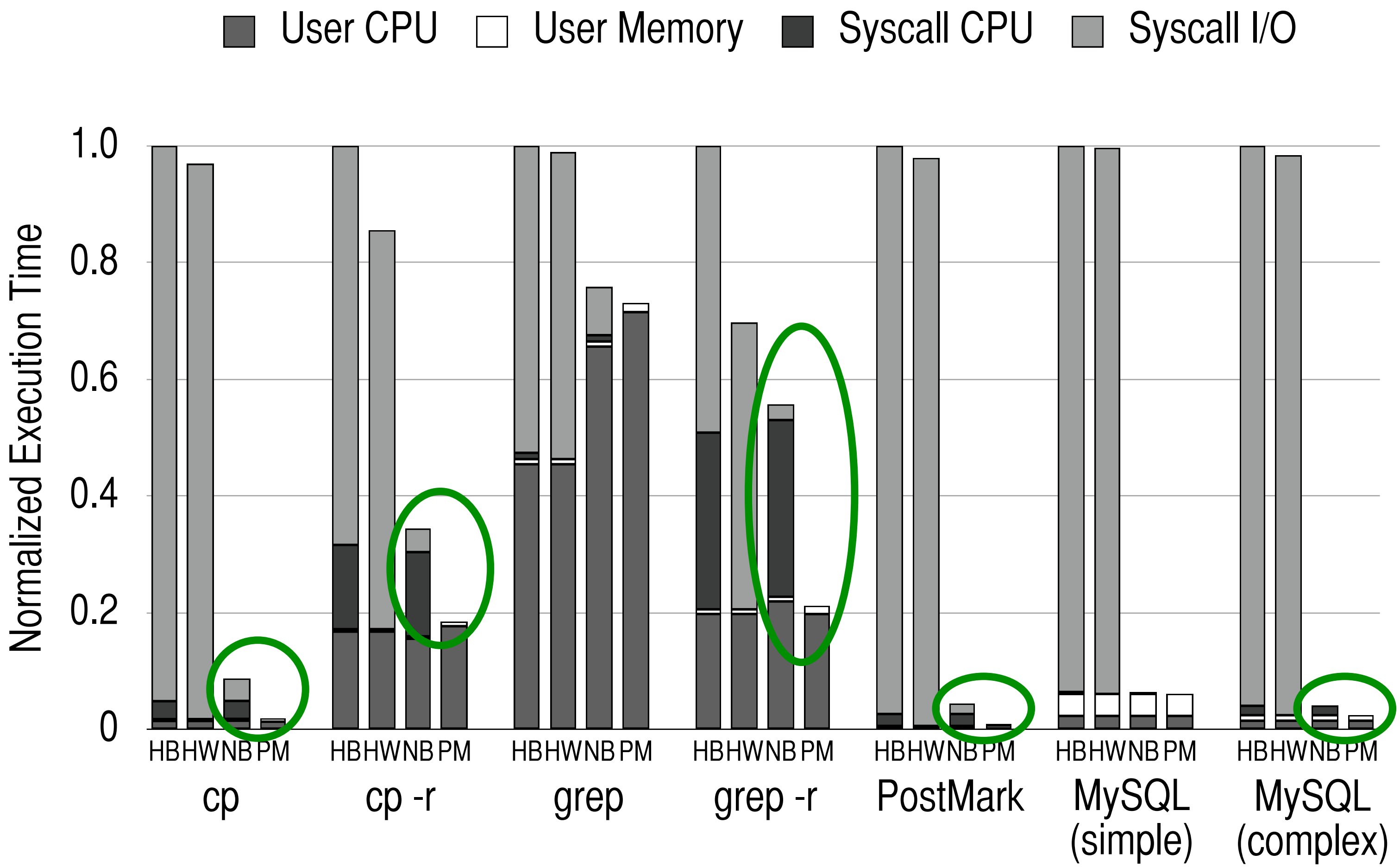
Switching from an HDD to NVM greatly reduces execution time due to NVM's much faster access latencies, especially for I/O-intensive workloads (cp, PostMark, MySQL)

Performance Results: NVM to PMM



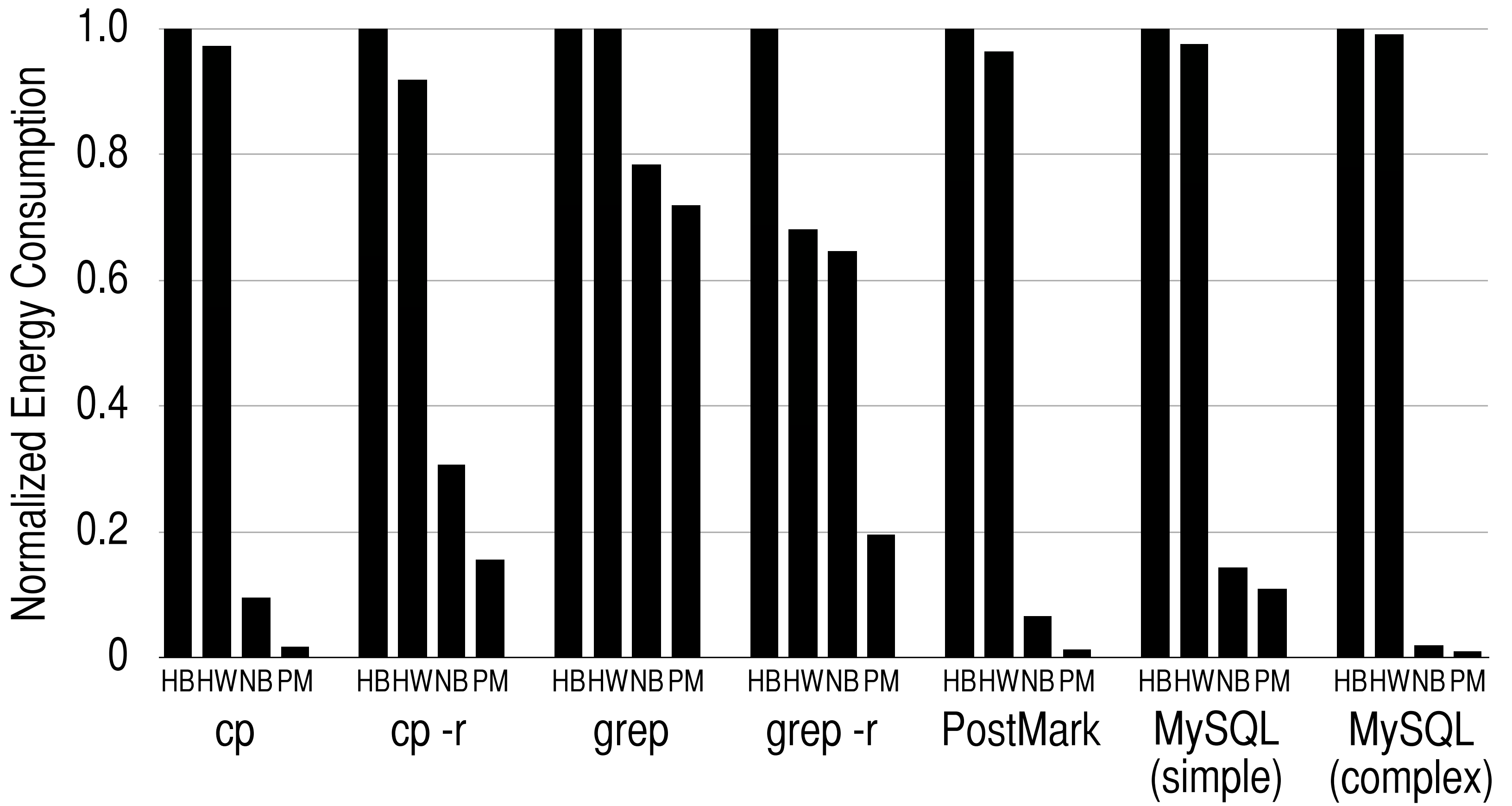
For most workloads, eliminating OS/FS code and buffering improves performance greatly on top of the NVM Baseline system
(even when DRAM is eliminated from the system)

Performance Results

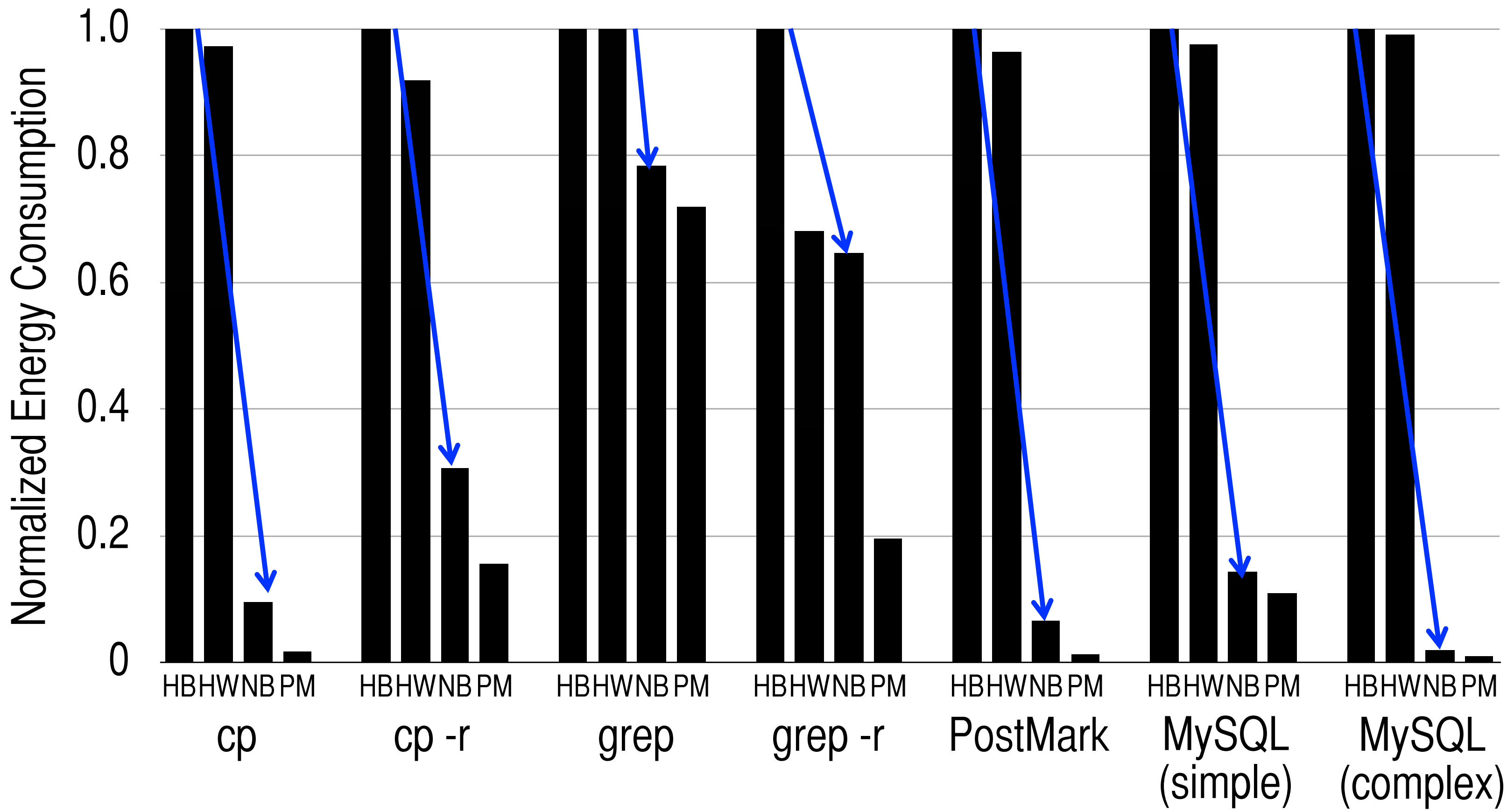


The workloads that see the greatest improvement from using a Persistent Memory are those that spend a large portion of their time executing system call code due to the two-level storage model

Energy Results

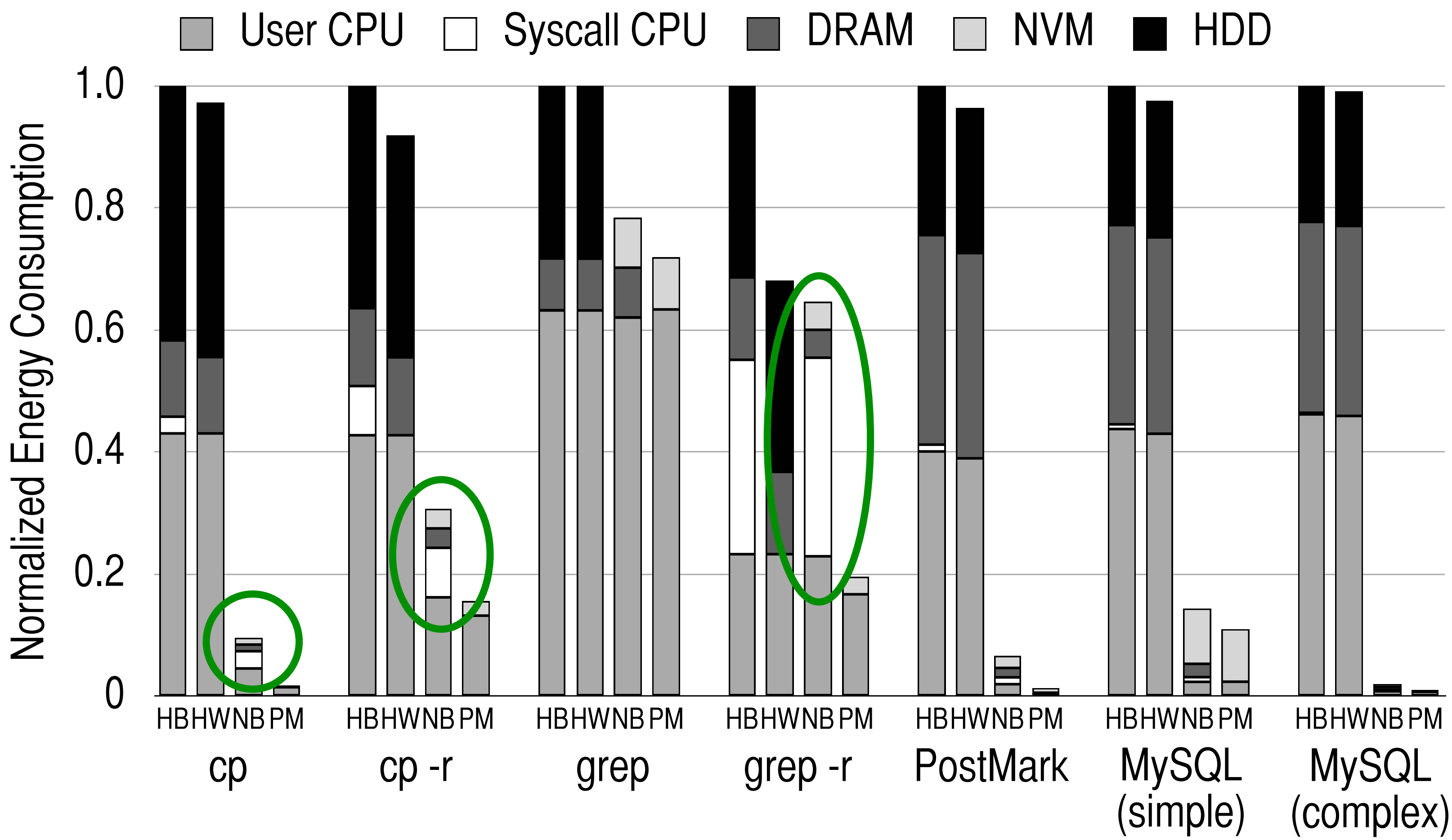


Energy Results: HDD to NVM



Between HDD-based and NVM-based systems, lower NVM energy leads to greatly reduced energy consumption

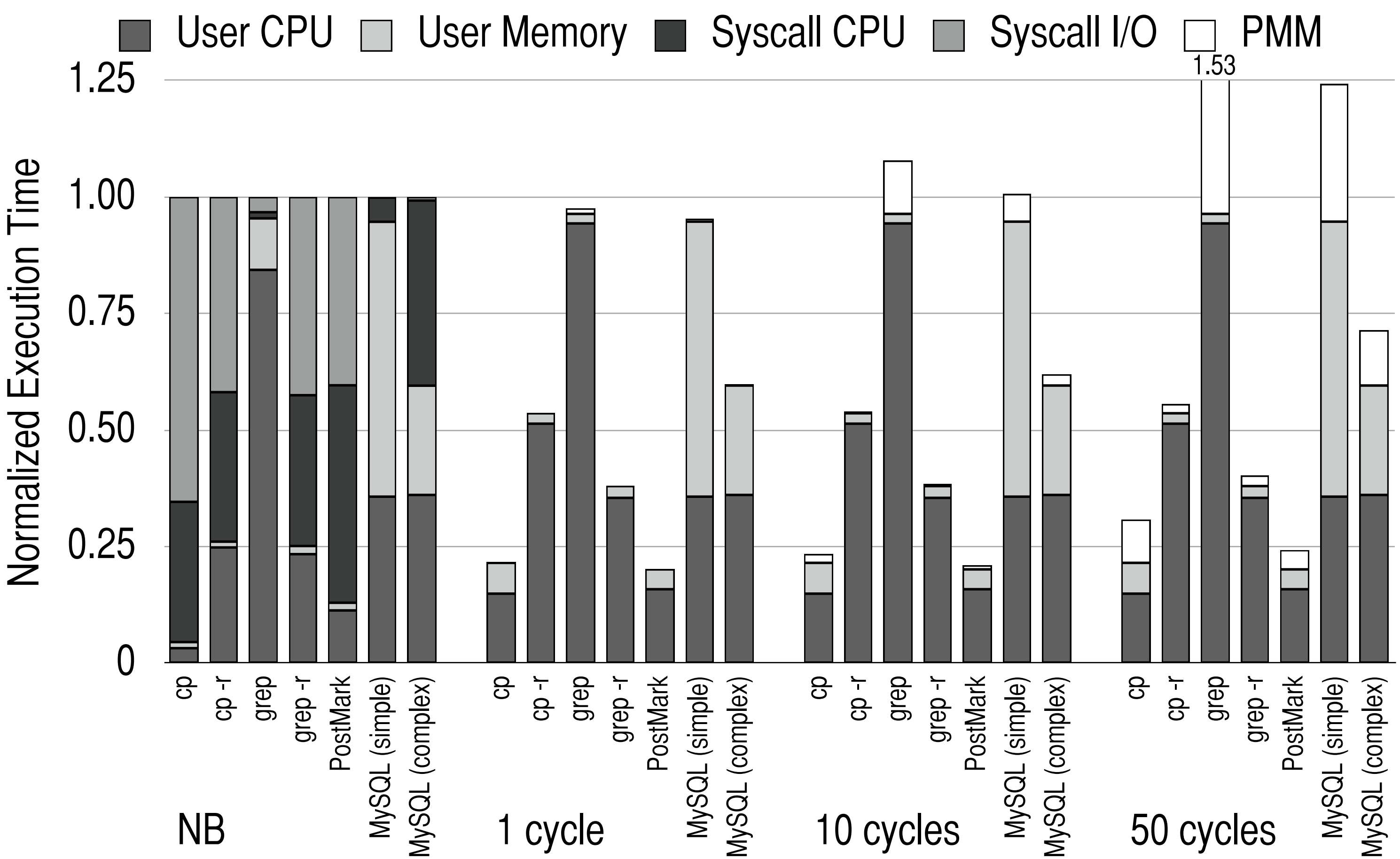
Energy Results: NVM to PMM



Between systems with and without OS/FS code, energy improvements come from:
1. reduced code footprint, 2. reduced data movement

Large energy reductions with a PMM over the NVM based system

Scalability Analysis: Effect of PMM Latency



Even if each PMM access takes a non-overlapped 50 cycles (conservative),
PMM still provides an overall improvement compared to the NVM baseline

Future research should target keeping PMM latencies in check

Outline

- Background: Storage and Memory Models
- Motivation: Eliminating Operating/File System Bottlenecks
- Our Proposal: Hardware/Software Coordinated Management of Storage and Memory
    - Opportunities and Benefits
- Evaluation Methodology
- Evaluation Results
- Related Work
- New Questions and Challenges
- Conclusions

Related Work

- We provide a comprehensive overview of past work related to single-level stores and persistent memory techniques

1. Integrating file systems with persistent memory
   - Need optimized hardware to fully take advantage of new technologies
2. Programming language support for persistent objects
   - Incurs the added latency of indirect data access through software
3. Load/store interfaces to persistent storage
   - Lack efficient and fast hardware support for address translation, efficient file indexing, fast reliability and protection guarantees
4. Analysis of OS overheads with Flash devices
   - Our study corroborates findings in this area and shows even larger consequences for systems with emerging NVM devices

- The goal of our work is to provide cheap and fast hardware support for memories to enable high energy efficiency and performance

Outline

- Background: Storage and Memory Models
- Motivation: Eliminating Operating/File System Bottlenecks
- Our Proposal: Hardware/Software Coordinated Management of Storage and Memory
  - Opportunities and Benefits
- Evaluation Methodology
- Evaluation Results
- Related Work
- New Questions and Challenges
- Conclusions

New Questions and Challenges

- We identify and discuss several open research questions
  - Q1. How to tailor applications for systems with persistent memory?
  - Q2. How can hardware and software cooperate to support a scalable, persistent single-level address space?
  - Q3. How to provide efficient backward compatibility (for two-level stores) on persistent memory systems?
  - Q4. How to mitigate potential hardware performance and energy overheads?

Outline

- Background: Storage and Memory Models
- Motivation: Eliminating Operating/File System Bottlenecks
- Our Proposal: Hardware/Software Coordinated Management of Storage and Memory
  - Opportunities and Benefits
- Evaluation Methodology
- Evaluation Results
- Related Work
- New Questions and Challenges
- Conclusions

Summary and Conclusions

- Traditional two-level storage model is inefficient in terms of performance and energy
  - Due to OS/FS code and buffering needed to manage two models
  - Especially so in future devices with NVM technologies, as we show
- New non-volatile memory based persistent memory designs that use a single-level storage model to unify memory and storage can alleviate this problem
- We quantified the performance and energy benefits of such a single-level persistent memory/storage design
  - Showed significant benefits from reduced code footprint, data movement, and system software overhead on a variety of workloads
- Such a design requires more research to answer the questions we have posed and enable efficient persistent memory managers

  → can lead to a fundamentally more efficient storage system

A Case for Efficient Hardware/Software Cooperative Management of Storage and Memory

Justin Meza[*], Yixin Luo[*], Samira Khan[*†], Jishen Zhao[§],
Yuan Xie[§‡], and Onur Mutlu[*]

[*]Carnegie Mellon University
[§]Pennsylvania State University
[†]Intel Labs [‡]AMD Research

SAFARI

Carnegie Mellon

Flash Memory Scaling

Readings in Flash Memory

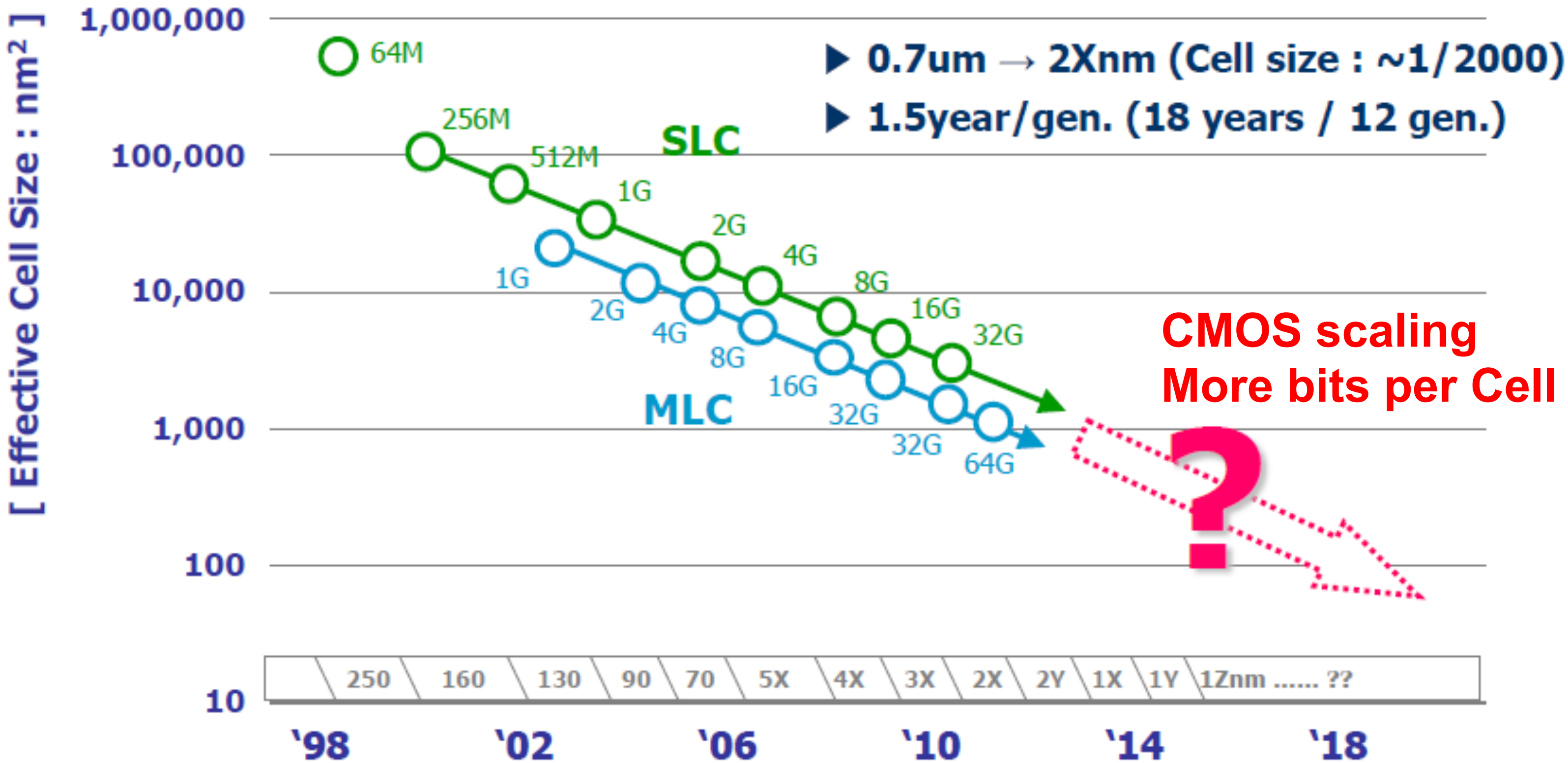
- Yu Cai, Gulay Yalcin, Onur Mutlu, Erich F. Haratsch, Adrian Cristal, Osman Unsal, and Ken Mai, **"Error Analysis and Retention-Aware Error Management for NAND Flash Memory"** *Intel Technology Journal (**ITJ**) Special Issue on Memory Resiliency*, Vol. 17, No. 1, May 2013.

- Yu Cai, Erich F. Haratsch, Onur Mutlu, and Ken Mai, **"Threshold Voltage Distribution in MLC NAND Flash Memory: Characterization, Analysis and Modeling"** *Proceedings of the Design, Automation, and Test in Europe Conference (**DATE**)*, Grenoble, France, March 2013. Slides (ppt)

- Yu Cai, Gulay Yalcin, Onur Mutlu, Erich F. Haratsch, Adrian Cristal, Osman Unsal, and Ken Mai, **"Flash Correct-and-Refresh: Retention-Aware Error Management for Increased Flash Memory Lifetime"** *Proceedings of the 30th IEEE International Conference on Computer Design (**ICCD**)*, Montreal, Quebec, Canada, September 2012. Slides (ppt) (pdf)

- Yu Cai, Erich F. Haratsch, Onur Mutlu, and Ken Mai, **"Error Patterns in MLC NAND Flash Memory: Measurement, Characterization, and Analysis"** *Proceedings of the Design, Automation, and Test in Europe Conference (**DATE**)*, Dresden, Germany, March 2012. Slides (ppt)

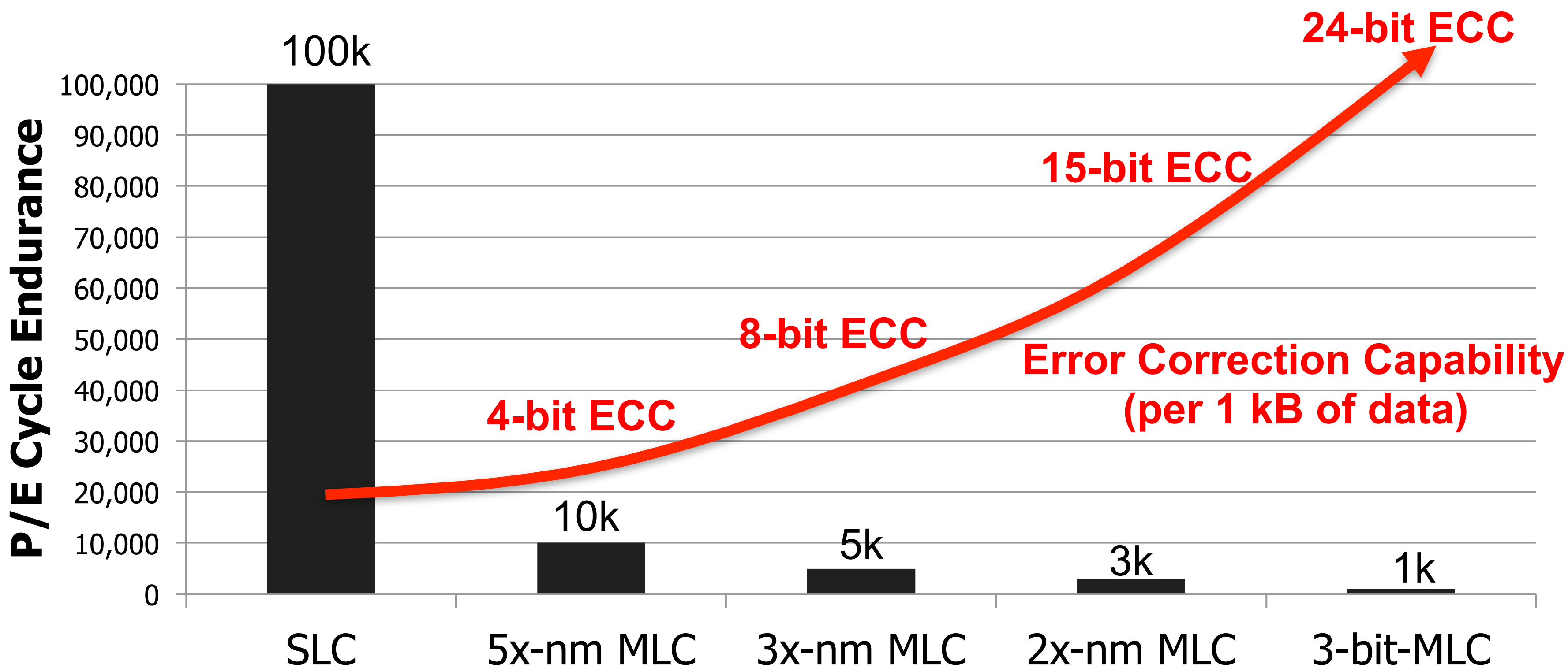
Evolution of NAND Flash Memory



Seaung Suk Lee, "Emerging Challenges in NAND Flash Technology", Flash Summit 2011 (Hynix)

- Flash memory widening its range of applications
  - Portable consumer devices, laptop PCs and enterprise servers

Decreasing Endurance with Flash Scaling

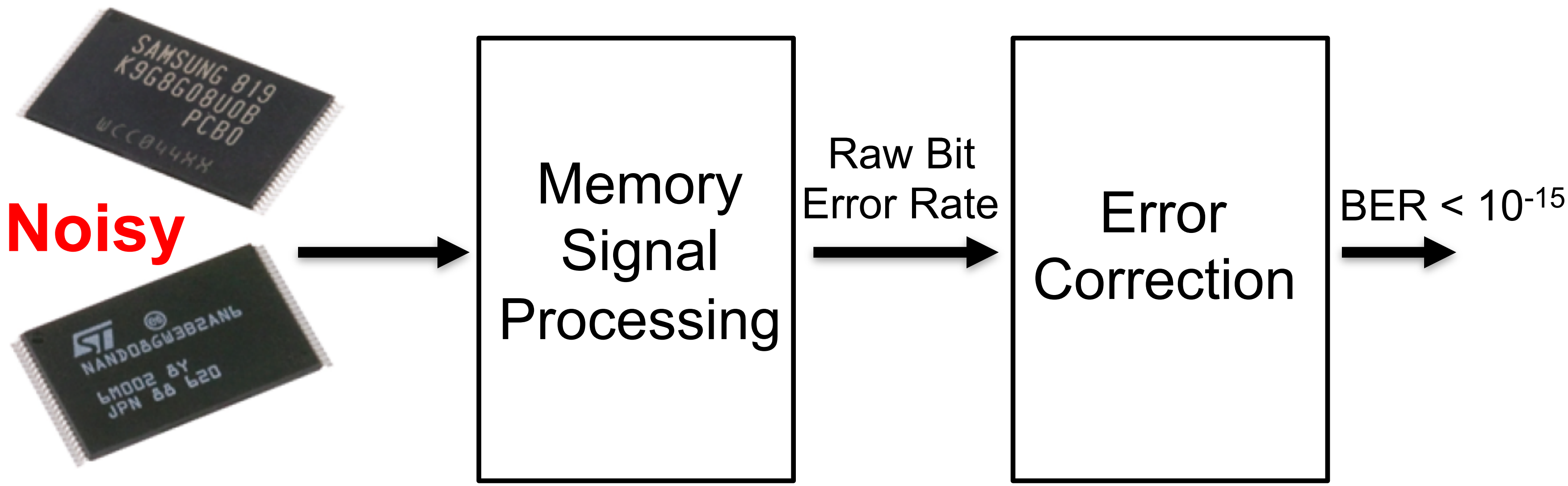


Ariel Maislos, "A New Era in Embedded Flash Memory", Flash Summit 2011 (Anobit)

- Endurance of flash memory decreasing with scaling and multi-level cells
- Error correction capability required to guarantee storage-class reliability (UBER < 10^{-15}) is increasing exponentially to reach *less* endurance

UBER: Uncorrectable bit error rate. Fraction of erroneous bits after error correction.

Future NAND Flash Storage Architecture

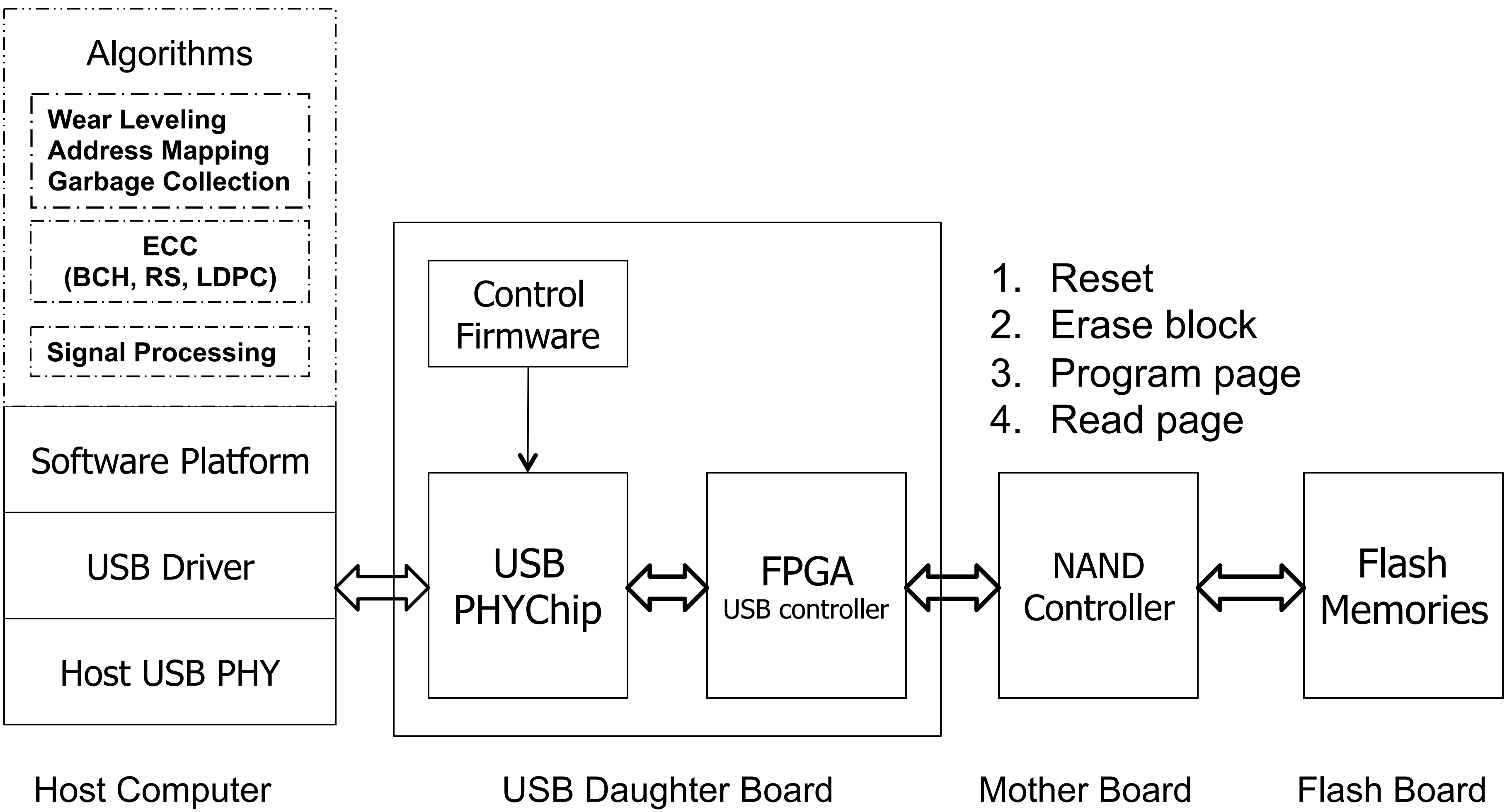


- Read voltage adjusting
- Data scrambler
- Data recovery
- Soft-information estimation

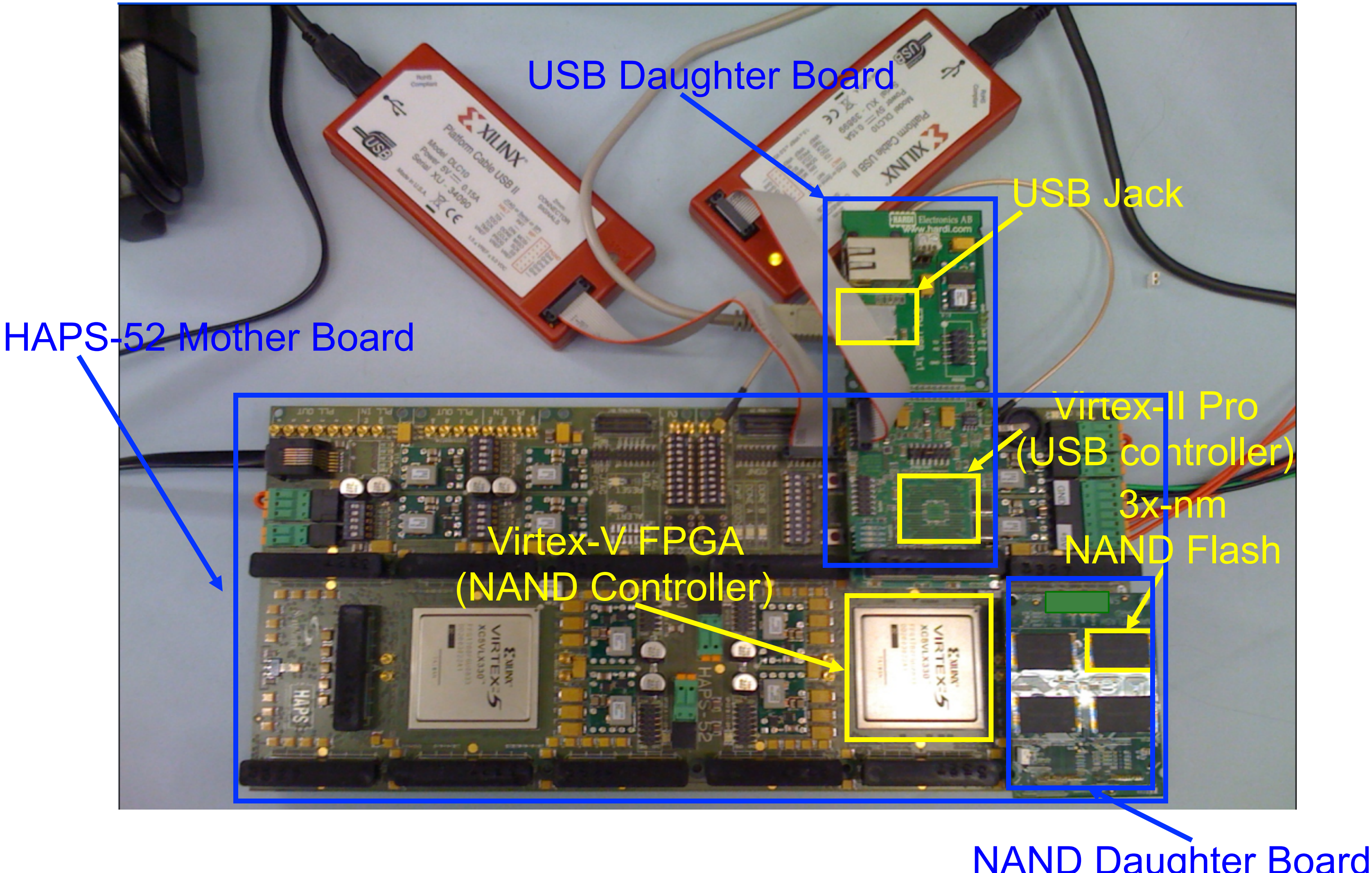
- Hamming codes
- BCH codes
- Reed-Solomon codes
- LDPC codes
- Other Flash friendly codes

Need to understand NAND flash error patterns

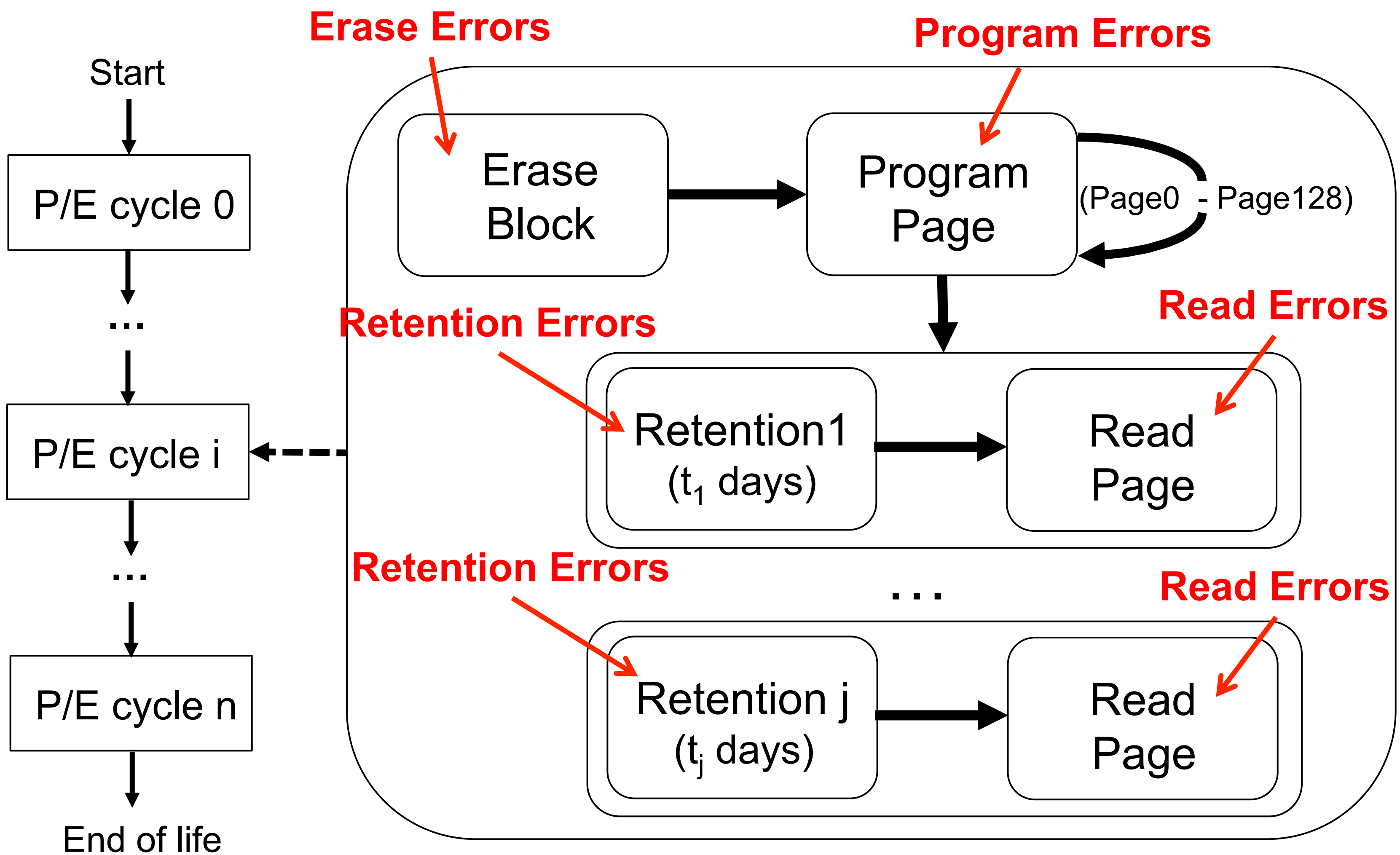
Test System Infrastructure



NAND Flash Testing Platform



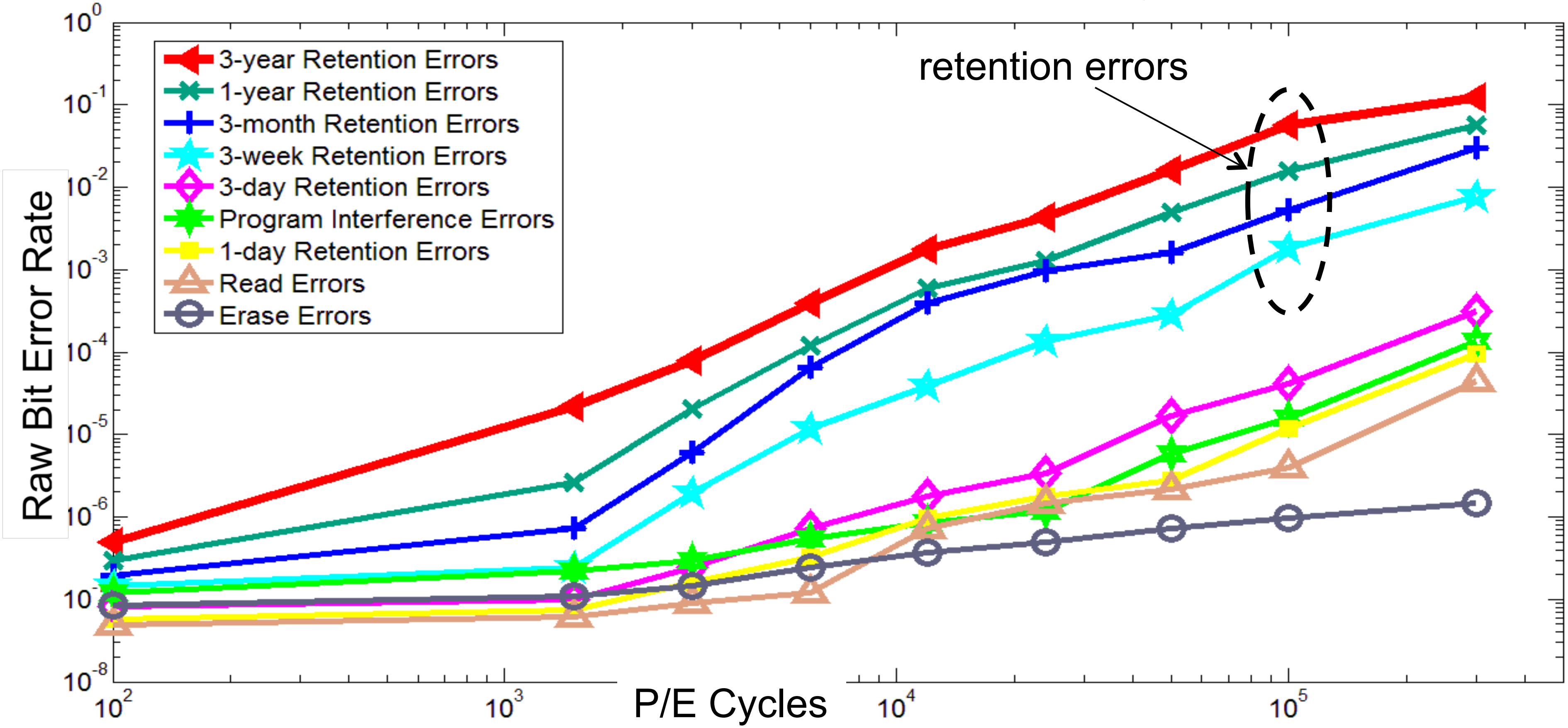
NAND Flash Usage and Error Model



Error Types and Testing Methodology

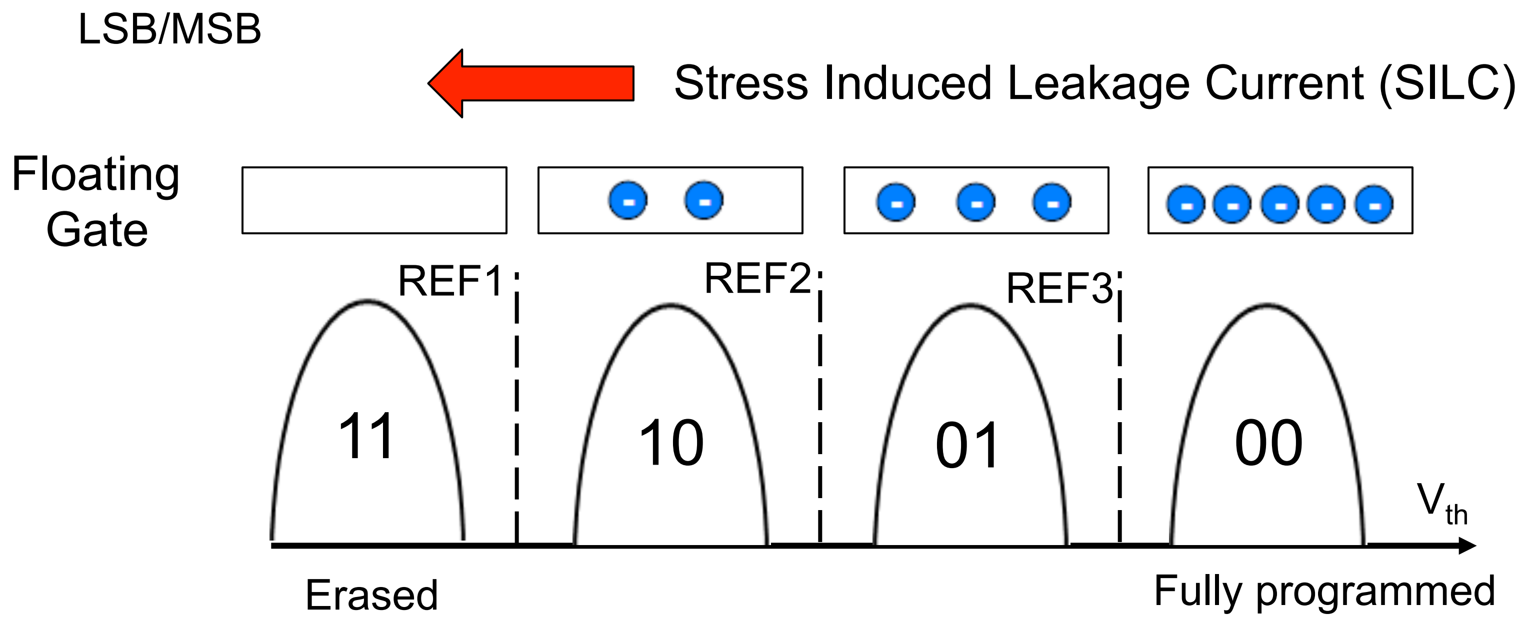
- Erase errors
  - Count the number of cells that fail to be erased to "11" state
- Program interference errors
  - Compare the data immediately after page programming and the data after the whole block being programmed
- Read errors
  - Continuously read a given block and compare the data between consecutive read sequences
- Retention errors
  - Compare the data read after an amount of time to data written
    - Characterize short term retention errors under room temperature
    - Characterize long term retention errors by baking in the oven under 125℃

Observations: Flash Error Analysis



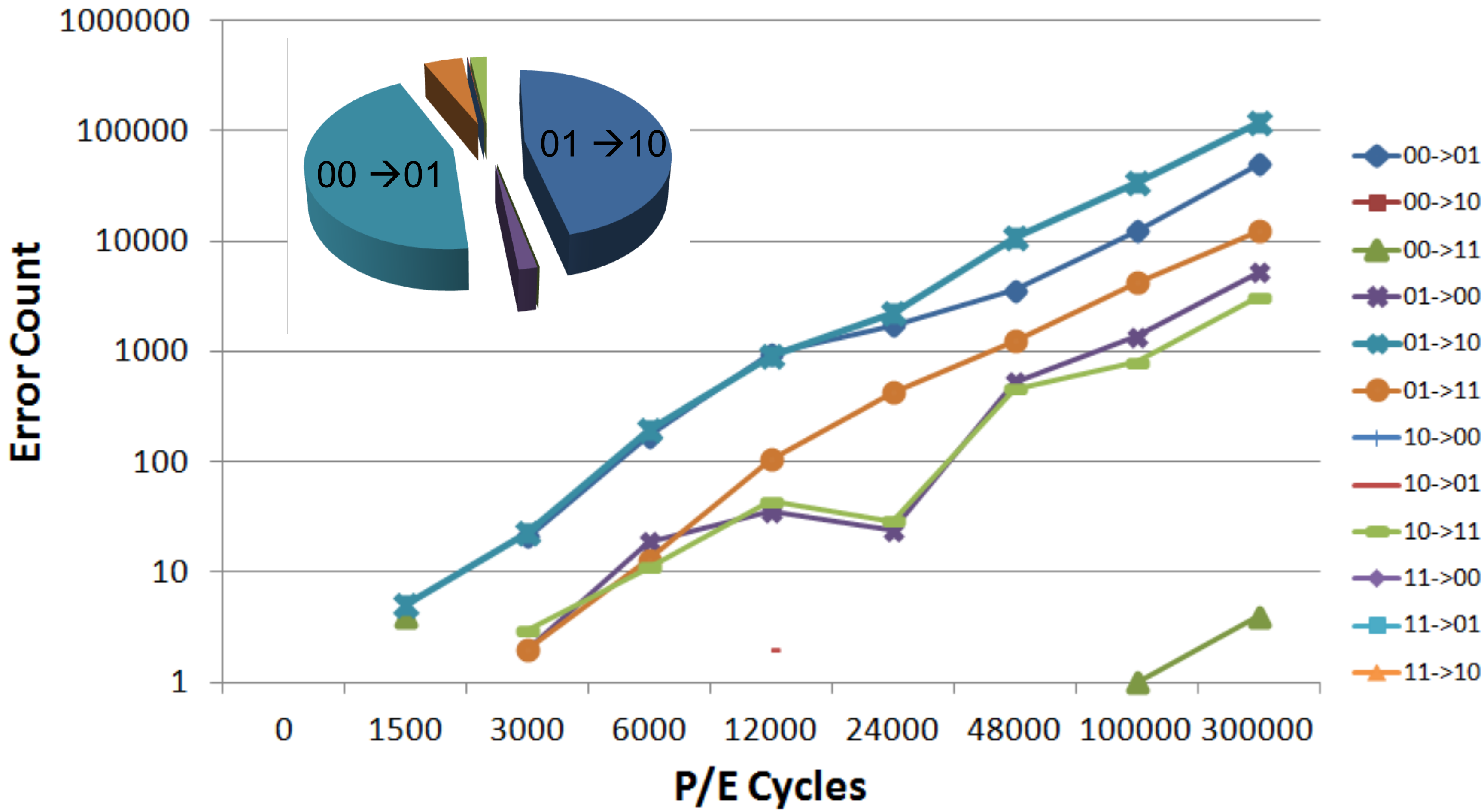
- Raw bit error rate increases exponentially with P/E cycles
- Retention errors are dominant (>99% for 1-year ret. time)
- Retention errors increase with retention time requirement

Retention Error Mechanism



- Electron loss from the floating gate causes retention errors
  - Cells with more programmed electrons suffer more from retention errors
  - Threshold voltage is more likely to shift by one window than by multiple

Retention Error Value Dependency

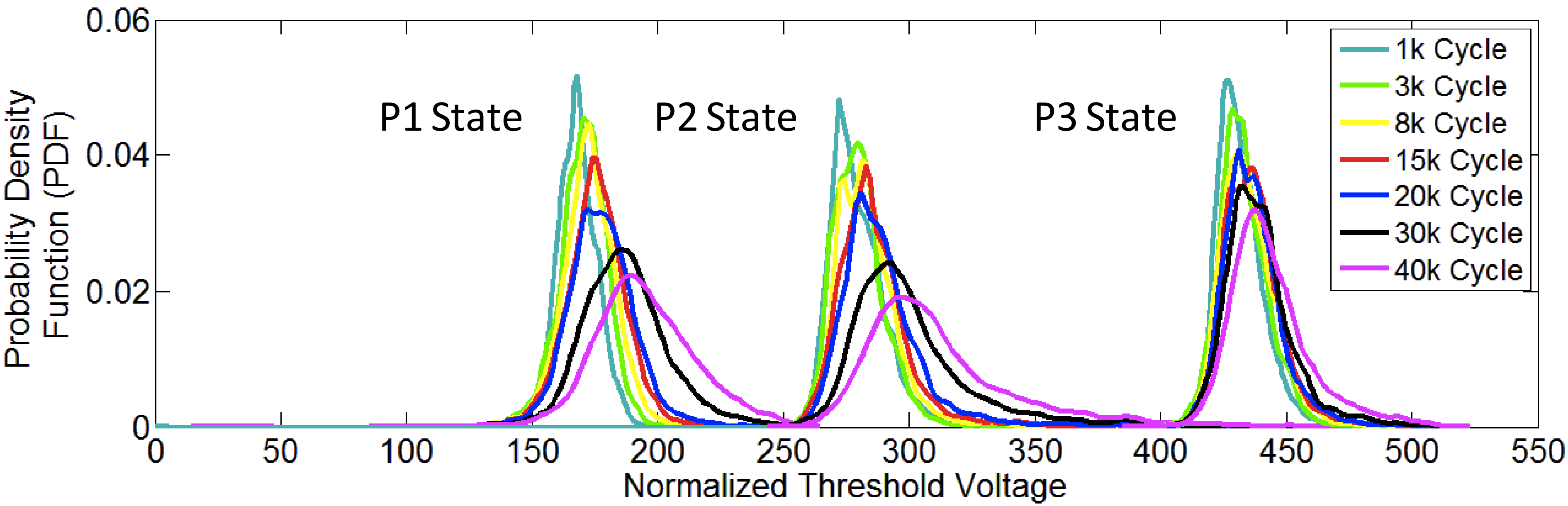


- Cells with more programmed electrons tend to suffer more from retention noise (i.e. 00 and 01)

More Details on Flash Error Analysis

- Yu Cai, Erich F. Haratsch, Onur Mutlu, and Ken Mai, **"Error Patterns in MLC NAND Flash Memory: Measurement, Characterization, and Analysis"** *Proceedings of the Design, Automation, and Test in Europe Conference (**DATE**)*, Dresden, Germany, March 2012. Slides (ppt)

Threshold Voltage Distribution Shifts



As P/E cycles increase ...

- Distribution shifts to the right
- Distribution becomes wider

More Detail

- Yu Cai, Erich F. Haratsch, Onur Mutlu, and Ken Mai, **"Threshold Voltage Distribution in MLC NAND Flash Memory: Characterization, Analysis and Modeling"** *Proceedings of the Design, Automation, and Test in Europe Conference* (***DATE***), Grenoble, France, March 2013. Slides (ppt)

Flash Correct-and-Refresh

Retention-Aware Error Management for Increased Flash Memory Lifetime

Yu Cai[1] Gulay Yalcin[2] Onur Mutlu[1] Erich F. Haratsch[3]
Adrian Cristal[2] Osman S. Unsal[2] Ken Mai[1]

[1] Carnegie Mellon University
[2] Barcelona Supercomputing Center
[3] LSI Corporation

SAFARI Carnegie Mellon

Executive Summary

- NAND flash memory has low endurance: a flash cell dies after 3k P/E cycles vs. 50k desired → Major scaling challenge for flash memory
- Flash error rate increases exponentially over flash lifetime
- Problem: Stronger error correction codes (ECC) are ineffective and undesirable for improving flash lifetime due to
  - diminishing returns on lifetime with increased correction strength
  - prohibitively high power, area, latency overheads
- Our Goal: Develop techniques to tolerate high error rates w/o strong ECC
- Observation: Retention errors are the dominant errors in MLC NAND flash
  - flash cell loses charge over time; retention errors increase as cell gets worn out
- Solution: Flash Correct-and-Refresh (FCR)
  - Periodically read, correct, and reprogram (in place) or remap each flash page before it accumulates more errors than can be corrected by simple ECC
  - Adapt "refresh" rate to the severity of retention errors (i.e., # of P/E cycles)
- Results: FCR improves flash memory lifetime by 46X with no hardware changes and low energy overhead; outperforms strong ECCs

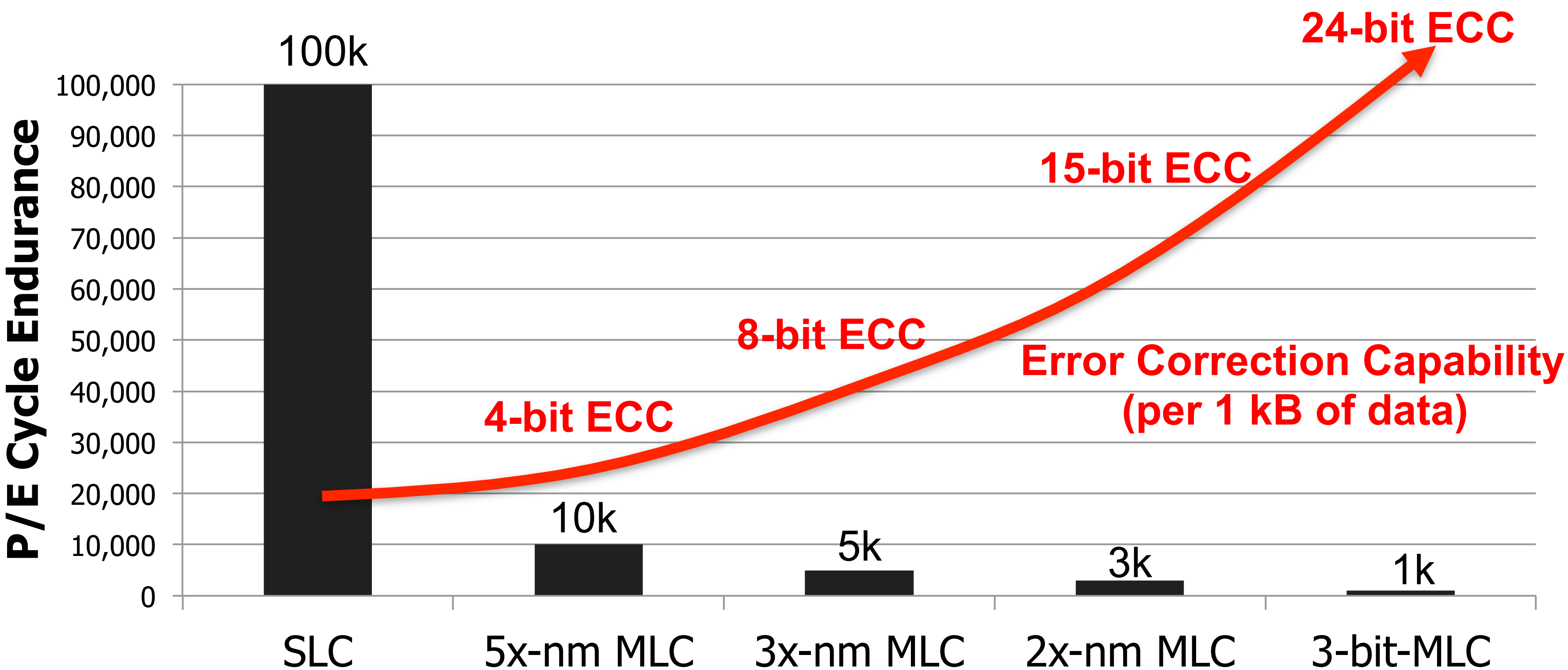
Outline

- Executive Summary
- The Problem: Limited Flash Memory Endurance/Lifetime
- Error and ECC Analysis for Flash Memory
- Flash Correct and Refresh Techniques (FCR)
- Evaluation
- Conclusions

Problem: Limited Endurance of Flash Memory

- NAND flash has limited endurance
  - A cell can tolerate a small number of Program/Erase (P/E) cycles
  - 3x-nm flash with 2 bits/cell → 3K P/E cycles
- Enterprise data storage requirements demand very high endurance
  - >50K P/E cycles (10 full disk writes per day for 3-5 years)
- Continued process scaling and more bits per cell will reduce flash endurance
- One potential solution: stronger error correction codes (ECC)
  - Stronger ECC not effective enough and inefficient

Decreasing Endurance with Flash Scaling



Ariel Maislos, "A New Era in Embedded Flash Memory", Flash Summit 2011 (Anobit)

- Endurance of flash memory decreasing with scaling and multi-level cells
- Error correction capability required to guarantee storage-class reliability (UBER < 10^{-15}) is increasing exponentially to reach *less* endurance

UBER: Uncorrectable bit error rate. Fraction of erroneous bits after error correction.

The Problem with Stronger Error Correction

- Stronger ECC detects and corrects more raw bit errors → increases P/E cycles endured

- Two shortcomings of stronger ECC:

  1. High implementation complexity

     → Power and area overheads increase super-linearly, but correction capability increases sub-linearly with ECC strength

  2. Diminishing returns on flash lifetime improvement

     → Raw bit error rate increases exponentially with P/E cycles, but correction capability increases sub-linearly with ECC strength

Outline

- Executive Summary
- The Problem: Limited Flash Memory Endurance/Lifetime
- Error and ECC Analysis for Flash Memory
- Flash Correct and Refresh Techniques (FCR)
- Evaluation
- Conclusions

Methodology: Error and ECC Analysis

- Characterized errors and error rates of 3x-nm MLC NAND flash using an experimental FPGA-based flash platform
  - Cai et al., "Error Patterns in MLC NAND Flash Memory: Measurement, Characterization, and Analysis," DATE 2012.
- Quantified Raw Bit Error Rate (RBER) at a given P/E cycle
  - Raw Bit Error Rate: Fraction of erroneous bits without any correction
- Quantified error correction capability (and area and power consumption) of various BCH-code implementations
  - Identified how much RBER each code can tolerate

    → how many P/E cycles (flash lifetime) each code can sustain

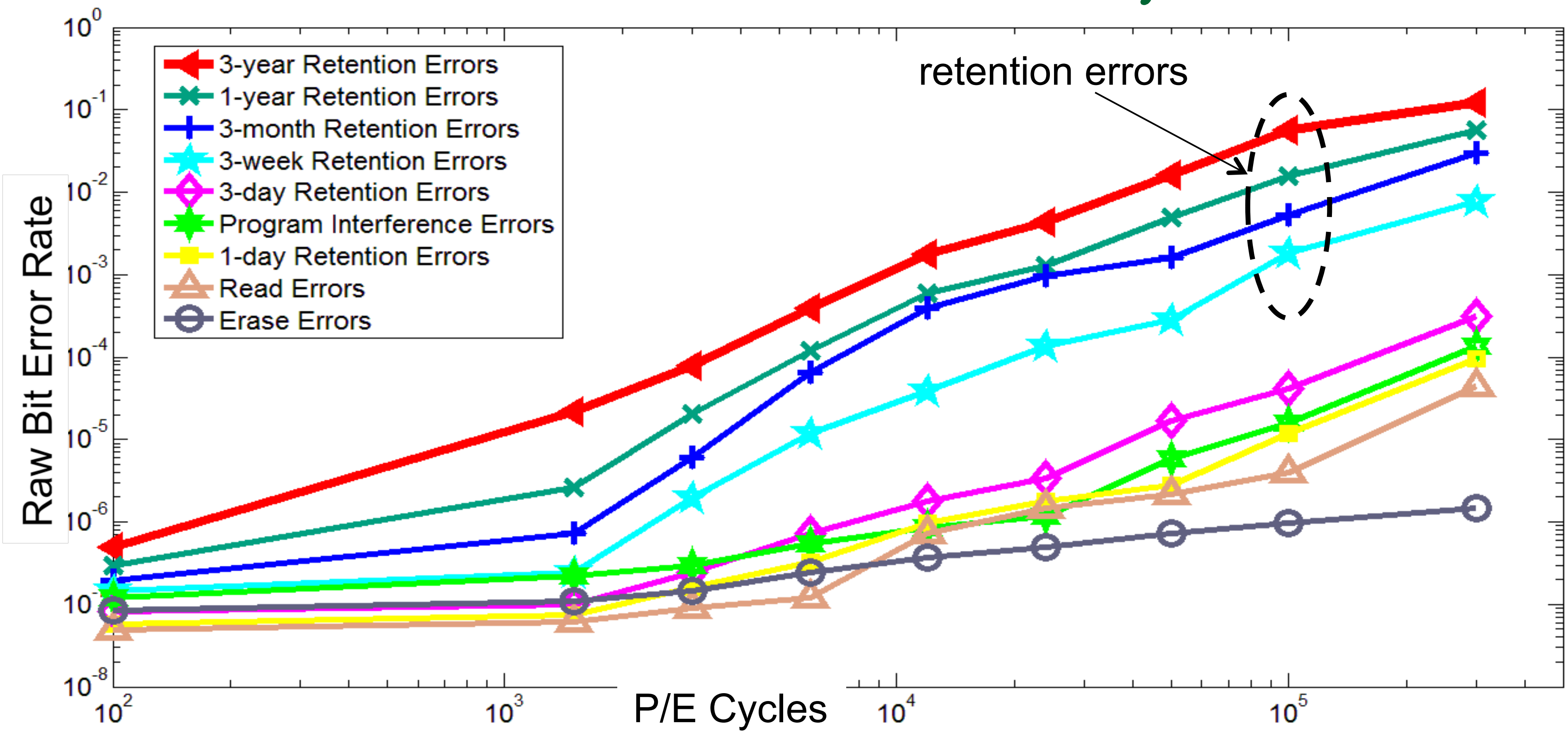
NAND Flash Error Types

- Four types of errors [Cai+, DATE 2012]

- Caused by common flash operations
  - Read errors
  - Erase errors
  - Program (interference) errors

- Caused by flash cell losing charge over time
  - Retention errors
    - Whether an error happens depends on required retention time
    - Especially problematic in MLC flash because voltage threshold window to determine stored value is smaller

Observations: Flash Error Analysis



- Raw bit error rate increases exponentially with P/E cycles
- Retention errors are dominant (>99% for 1-year ret. time)
- Retention errors increase with retention time requirement

Methodology: Error and ECC Analysis

- Characterized errors and error rates of 3x-nm MLC NAND flash using an experimental FPGA-based flash platform
  - Cai et al., "Error Patterns in MLC NAND Flash Memory: Measurement, Characterization, and Analysis," DATE 2012.
- Quantified Raw Bit Error Rate (RBER) at a given P/E cycle
  - Raw Bit Error Rate: Fraction of erroneous bits without any correction
- Quantified error correction capability (and area and power consumption) of various BCH-code implementations
  - Identified how much RBER each code can tolerate

    → how many P/E cycles (flash lifetime) each code can sustain

ECC Strength Analysis

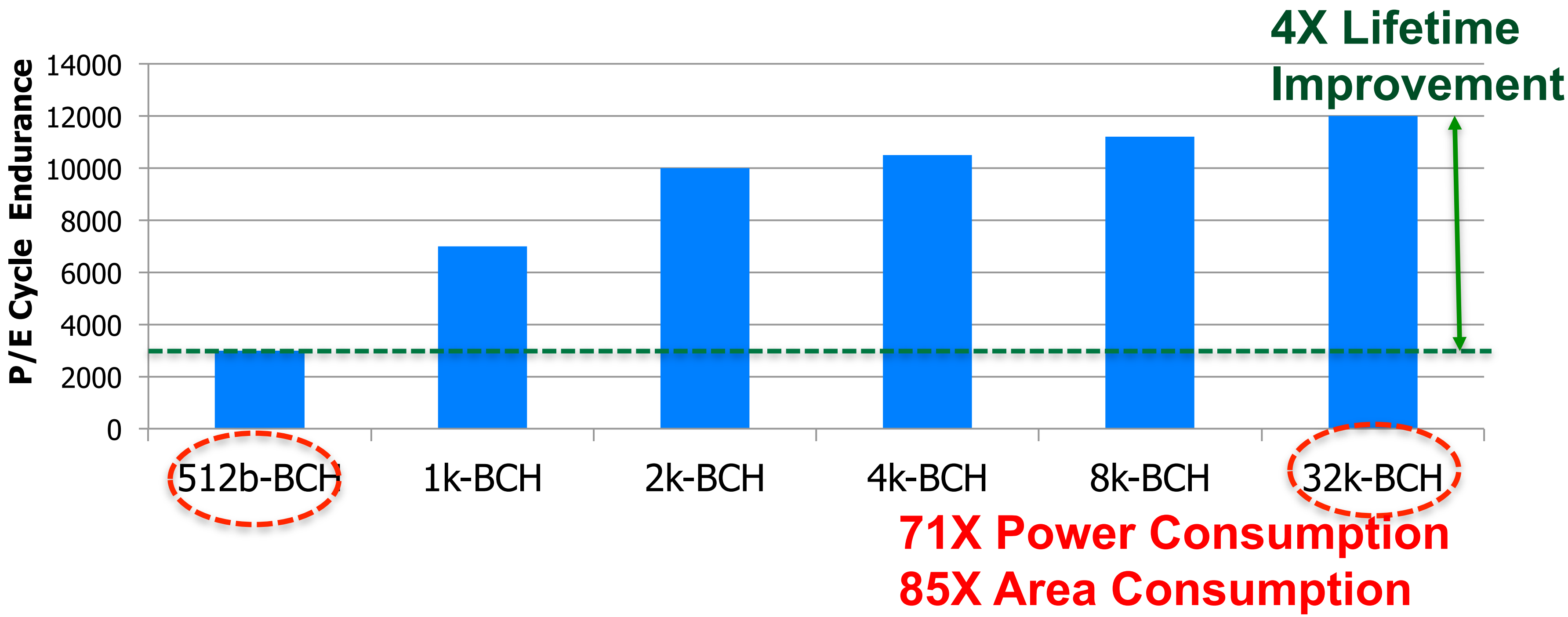
Error correction capability increases sub-linearly

Power and area overheads increase super-linearly

Code length (n)	Correctable Errors (t)	Acceptable Raw BER	Norm. Power	Norm. Area
512	7	1.0×10^{-4} (1x)	1	1
1024	12	4.0×10^{-4} (4x)	2	2.1
2048	22	1.0×10^{-3} (10x)	4.1	3.9
4096	40	1.7×10^{-3} (17x)	8.6	10.3
8192	74	2.2×10^{-3} (22x)	17.8	21.3
32768	259	2.6×10^{-3} (26x)	71	85

Resulting Flash Lifetime with Strong ECC

- Lifetime improvement comparison of various BCH codes



Strong ECC is very inefficient at improving lifetime

Our Goal

Develop new techniques

to improve flash lifetime

without relying on stronger ECC

Outline

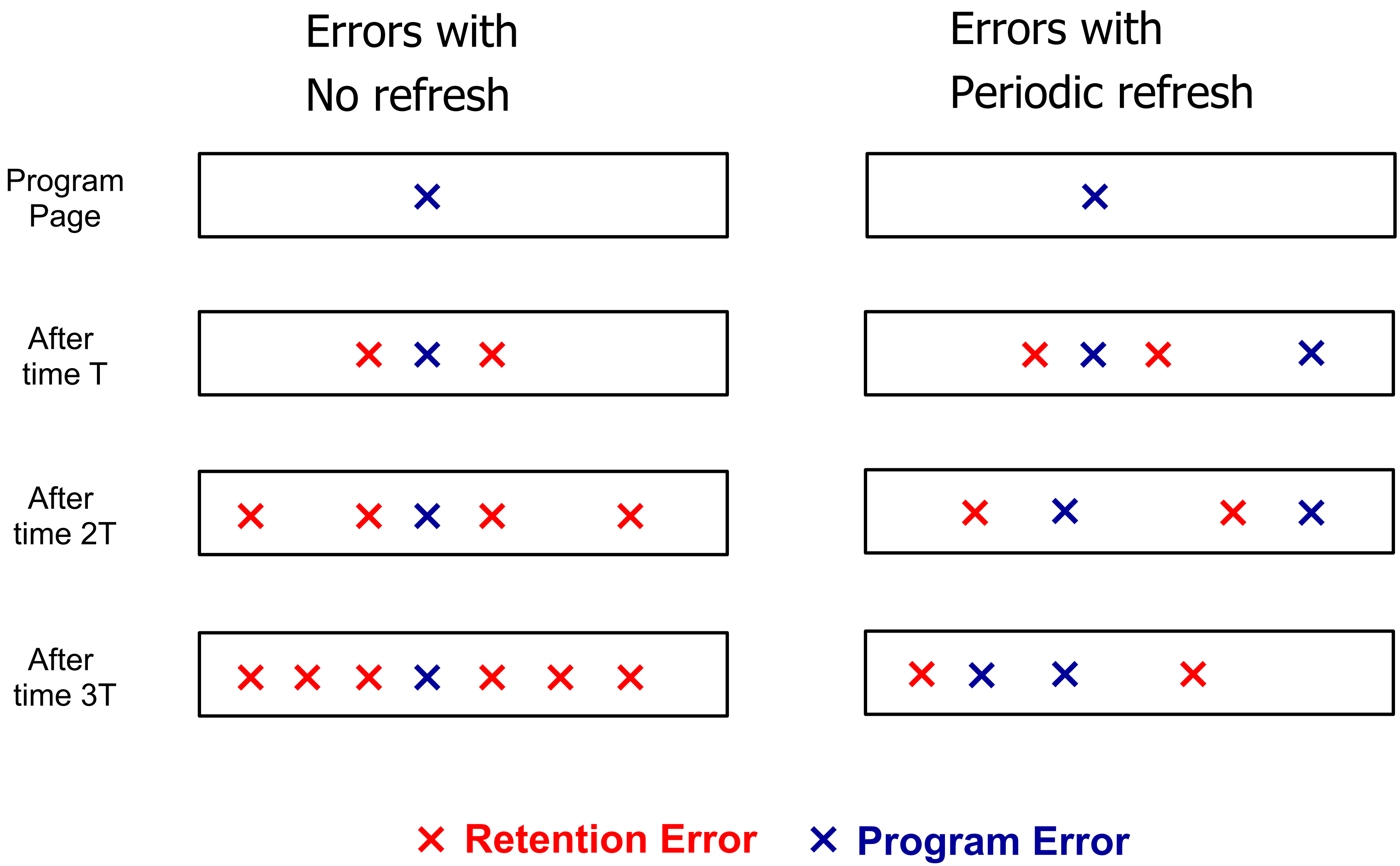
- Executive Summary
- The Problem: Limited Flash Memory Endurance/Lifetime
- Error and ECC Analysis for Flash Memory
- Flash Correct and Refresh Techniques (FCR)
- Evaluation
- Conclusions

Flash Correct-and-Refresh (FCR)

- Key Observations:
  - Retention errors are the dominant source of errors in flash memory **[Cai+ DATE 2012][Tanakamaru+ ISSCC 2011]**
    → limit flash lifetime as they increase over time
  - Retention errors can be corrected by "refreshing" each flash page periodically

- Key Idea:
  - Periodically read each flash page,
  - Correct its errors using "weak" ECC, and
  - Either remap it to a new physical page or reprogram it in-place,
  - Before the page accumulates more errors than ECC-correctable
  - Optimization: Adapt refresh rate to endured P/E cycles

FCR Intuition



FCR: Two Key Questions

- How to refresh?
  - Remap a page to another one
  - Reprogram a page (in-place)
  - Hybrid of remap and reprogram
- When to refresh?
  - Fixed period
  - Adapt the period to retention error severity

Outline

- Executive Summary
- The Problem: Limited Flash Memory Endurance/Lifetime
- Error and ECC Analysis for Flash Memory
- Flash Correct and Refresh Techniques (FCR)
  1. Remapping based FCR
  2. Hybrid Reprogramming and Remapping based FCR
  3. Adaptive-Rate FCR
- Evaluation
- Conclusions

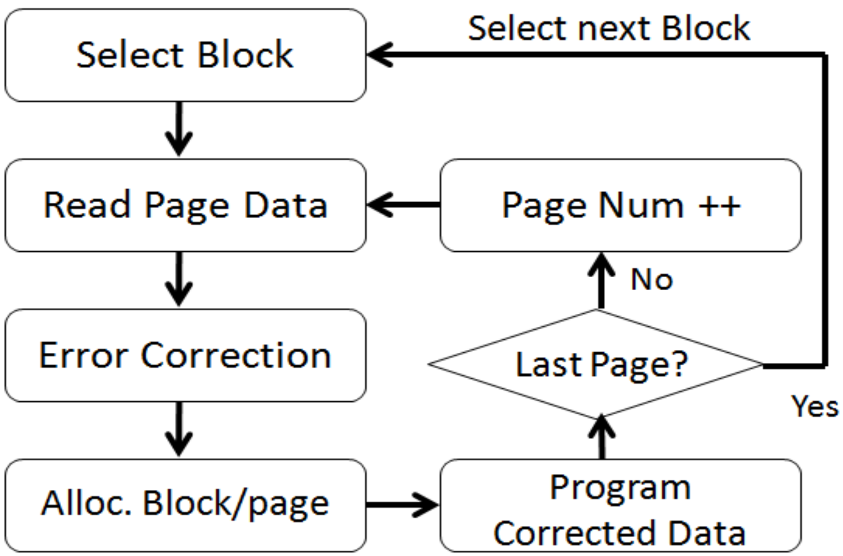
Outline

- Executive Summary
- The Problem: Limited Flash Memory Endurance/Lifetime
- Error and ECC Analysis for Flash Memory
- Flash Correct and Refresh Techniques (FCR)

  1. Remapping based FCR
  2. Hybrid Reprogramming and Remapping based FCR
  3. Adaptive-Rate FCR
- Evaluation
- Conclusions

Remapping Based FCR

- Idea: Periodically remap each page to a different physical page (after correcting errors)
  - Also [Pan et al., HPCA 2012]
  - FTL already has support for changing logical → physical flash block/page mappings
  - Deallocated block is erased by garbage collector



- Problem: Causes additional erase operations → more wearout
  - Bad for read-intensive workloads (few erases really needed)
  - Lifetime degrades for such workloads (see paper)

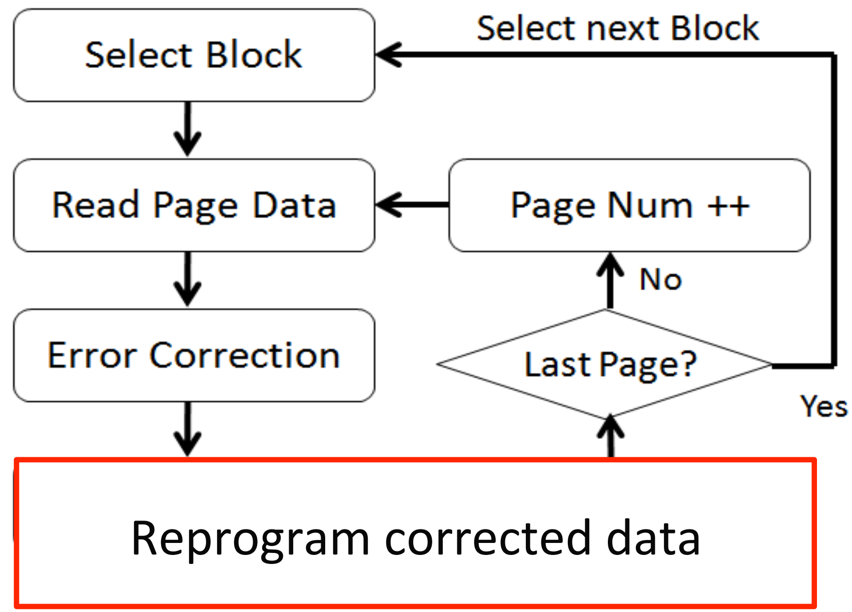
Outline

- Executive Summary
- The Problem: Limited Flash Memory Endurance/Lifetime
- Error and ECC Analysis for Flash Memory
- Flash Correct and Refresh Techniques (FCR)

  1. Remapping based FCR
  2. Hybrid Reprogramming and Remapping based FCR
  3. Adaptive-Rate FCR
- Evaluation
- Conclusions

In-Place Reprogramming Based FCR

- Idea: Periodically reprogram (in-place) each physical page (after correcting errors)
  - Flash programming techniques (ISPP) can correct retention errors in-place by recharging flash cells



- Problem: Program errors accumulate on the same page → may not be correctable by ECC after some time

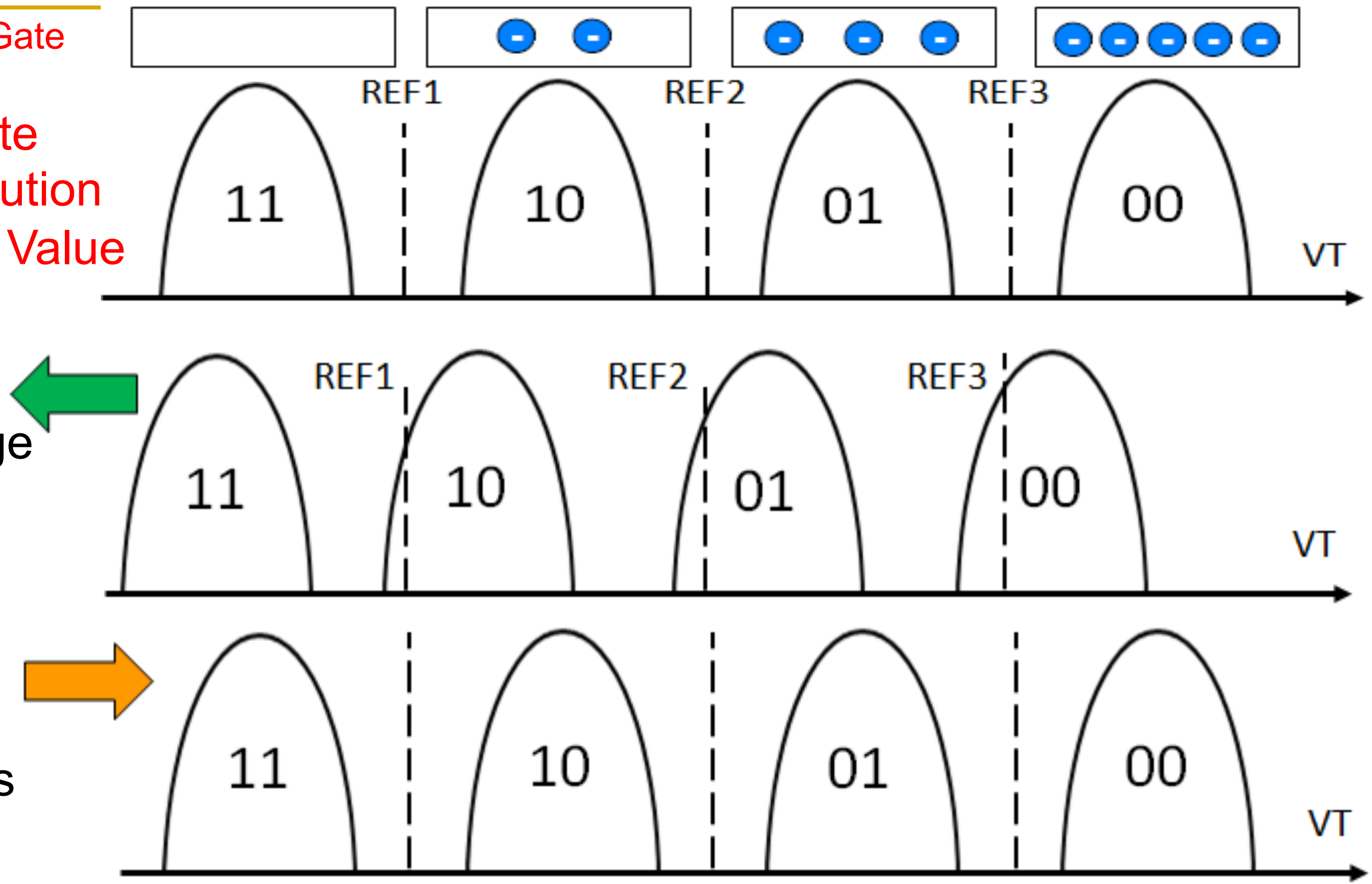
In-Place Reprogramming of Flash Cells

Floating Gate

Floating Gate Voltage Distribution for each Stored Value

Retention errors are caused by cell voltage shifting to the left

ISPP moves cell voltage to the right; fixes retention errors



- Pro: No remapping needed → no additional erase operations
- Con: Increases the occurrence of program errors

Program Errors in Flash Memory

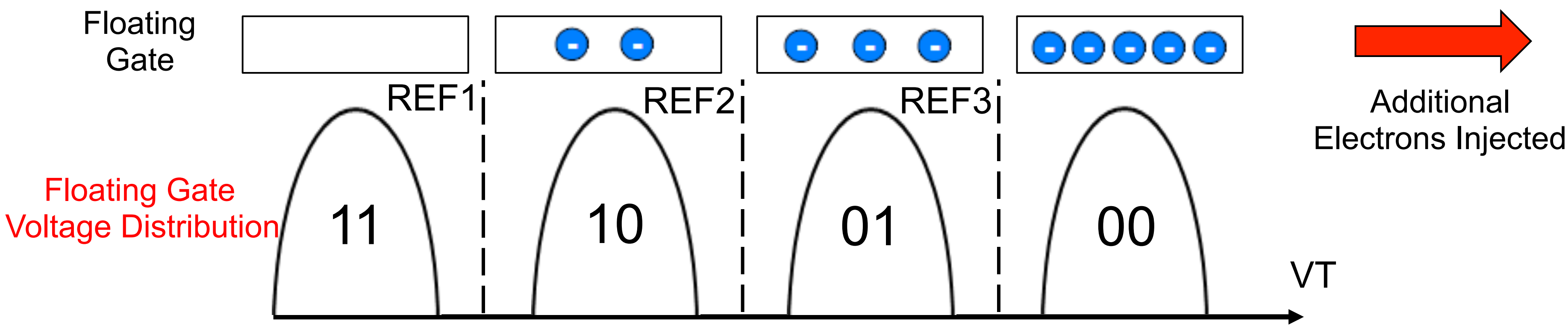
- When a cell is being programmed, voltage level of a neighboring cell changes (unintentionally) due to parasitic capacitance coupling

  → can change the data value stored

- Also called program interference error

- Program interference causes neighboring cell voltage to shift to the right

Problem with In-Place Reprogramming



Original data to be programmed	…	00	11	01	00	10	11	00	…
Program errors after initial programming	…	00	10	01	00	10	11	00	…
Retention errors after some time	…	01	10	10	00	11	11	01	…
Errors after in-place reprogramming	…	00	10	01	00	10	10	00	…

1. Read data
2. Correct errors
3. Reprogram back

Problem: Program errors can accumulate over time

Hybrid Reprogramming/Remapping Based FCR

- Idea:
  - Monitor the count of right-shift errors (after error correction)
  - If count < threshold, in-place reprogram the page
  - Else, remap the page to a new page

- Observation:
  - Program errors much less frequent than retention errors → Remapping happens only infrequently

- Benefit:
  - Hybrid FCR greatly reduces erase operations due to remapping

Outline

- Executive Summary
- The Problem: Limited Flash Memory Endurance/Lifetime
- Error and ECC Analysis for Flash Memory
- Flash Correct and Refresh Techniques (FCR)
  1. Remapping based FCR
  2. Hybrid Reprogramming and Remapping based FCR
  3. Adaptive-Rate FCR
- Evaluation
- Conclusions

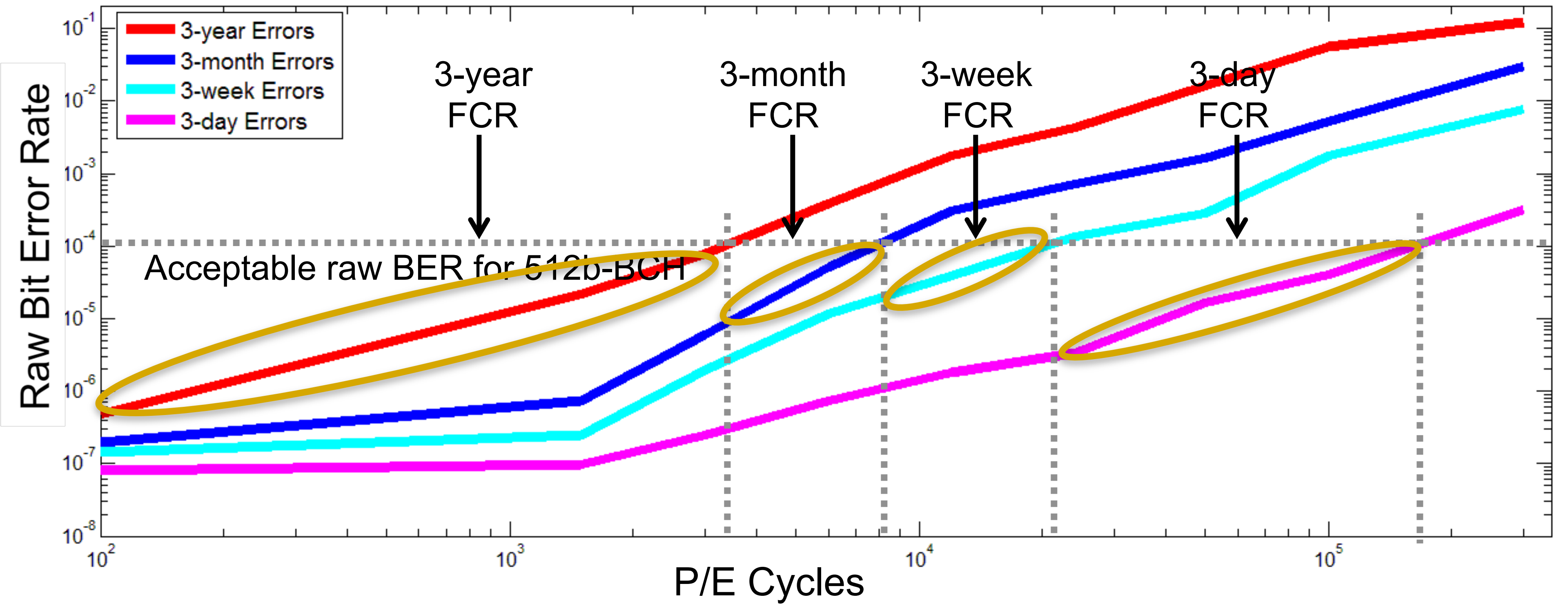
Adaptive-Rate FCR

- Observation:
  - Retention error rate strongly depends on the P/E cycles a flash page endured so far
  - No need to refresh frequently (at all) early in flash lifetime

- Idea:
  - Adapt the refresh rate to the P/E cycles endured by each page
  - Increase refresh rate gradually with increasing P/E cycles

- Benefits:
  - Reduces overhead of refresh operations
  - Can use existing FTL mechanisms that keep track of P/E cycles

Adaptive-Rate FCR (Example)



Select refresh frequency such that error rate is below acceptable rate

Outline

- Executive Summary
- The Problem: Limited Flash Memory Endurance/Lifetime
- Error and ECC Analysis for Flash Memory
- Flash Correct and Refresh Techniques (FCR)
  1. Remapping based FCR
  2. Hybrid Reprogramming and Remapping based FCR
  3. Adaptive-Rate FCR
- Evaluation
- Conclusions

FCR: Other Considerations

- Implementation cost
  - No hardware changes
  - FTL software/firmware needs modification
- Response time impact
  - FCR not as frequent as DRAM refresh; low impact
- Adaptation to variations in retention error rate
  - Adapt refresh rate based on, e.g., temperature **[Liu+ ISCA 2012]**
- FCR requires power
  - Enterprise storage systems typically powered on

Outline

- Executive Summary
- The Problem: Limited Flash Memory Endurance/Lifetime
- Error and ECC Analysis for Flash Memory
- Flash Correct and Refresh Techniques (FCR)
- Evaluation
- Conclusions

Evaluation Methodology

- Experimental flash platform to obtain error rates at different P/E cycles **[Cai+ DATE 2012]**
- Simulation framework to obtain P/E cycles of real workloads: DiskSim with SSD extensions
- Simulated system: 256GB flash, 4 channels, 8 chips/channel, 8K blocks/chip, 128 pages/block, 8KB pages
- Workloads
  - File system applications, databases, web search
  - Categories: Write-heavy, read-heavy, balanced
- Evaluation metrics
  - Lifetime (extrapolated)
  - Energy overhead, P/E cycle overhead

Extrapolated Lifetime

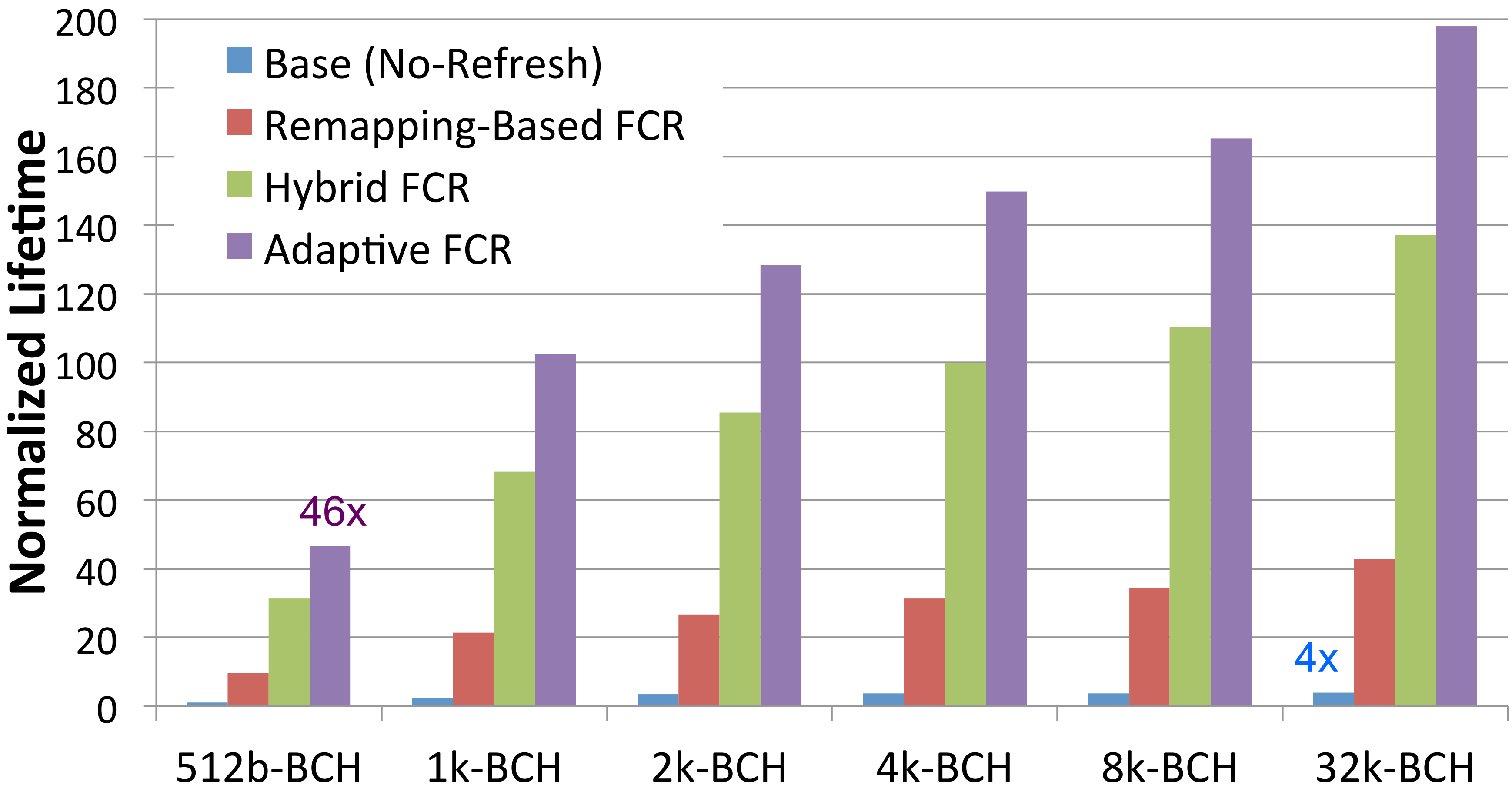
Obtained from Experimental Platform Data

$$\frac{\text{Maximum full disk P/E Cycles for a Technique}}{\text{Total full disk P/E Cycles for a Workload}} \times \text{\# of Days of Given Application}$$

Obtained from Workload Simulation

Real length (in time) of each workload trace

Normalized Flash Memory Lifetime

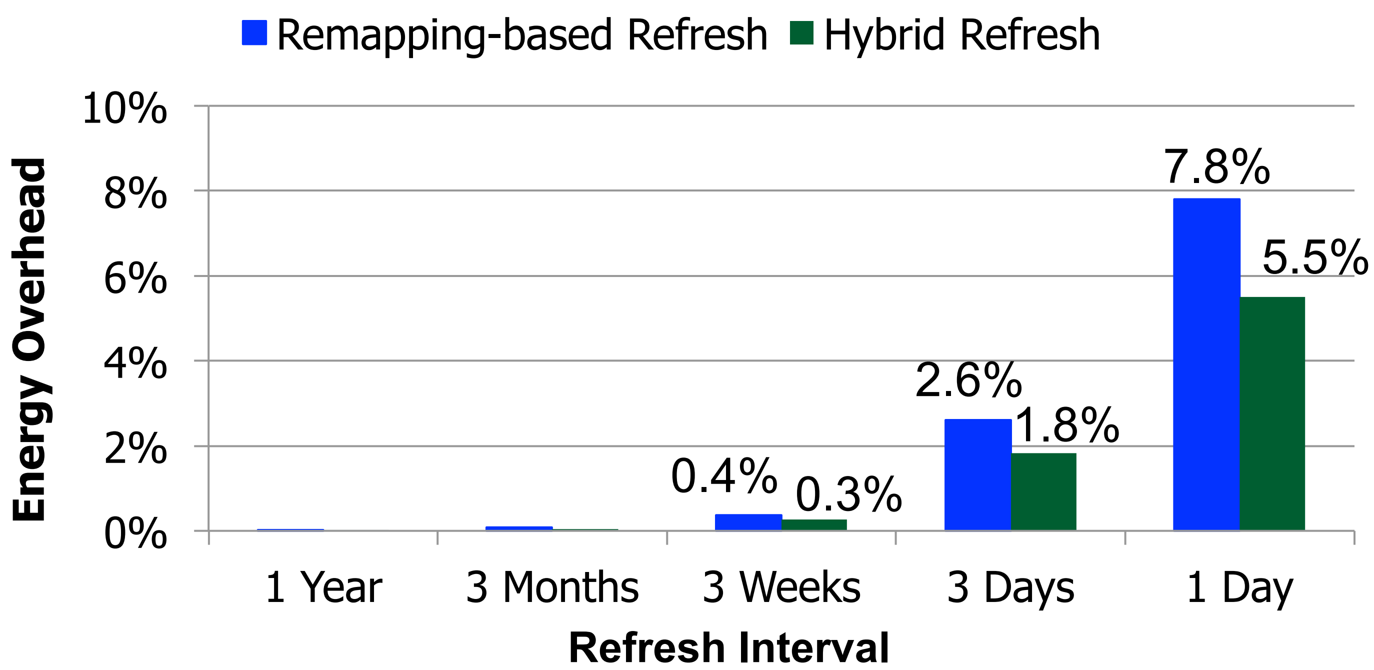


Lifetime of FCR much higher than lifetime of stronger ECC

Lifetime Evaluation Takeaways

- Significant average lifetime improvement over no refresh
  - Adaptive-rate FCR: 46X
  - Hybrid reprogramming/remapping based FCR: 31X
  - Remapping based FCR: 9X
- FCR lifetime improvement larger than that of stronger ECC
  - 46X vs. 4X with 32-kbit ECC (over 512-bit ECC)
  - FCR is less complex and less costly than stronger ECC
- Lifetime on all workloads improves with Hybrid FCR
  - Remapping based FCR can degrade lifetime on read-heavy WL
  - Lifetime improvement highest in write-heavy workloads

Energy Overhead



- Adaptive-rate refresh: <1.8% energy increase until daily refresh is triggered

Overhead of Additional Erases

- Additional erases happen due to remapping of pages

- Low (2%-20%) for write intensive workloads
- High (up to 10X) for read-intensive workloads

- Improved P/E cycle lifetime of all workloads largely outweighs the additional P/E cycles due to remapping

More Results in the Paper

- Detailed workload analysis
- Effect of refresh rate

Outline

- Executive Summary
- The Problem: Limited Flash Memory Endurance/Lifetime
- Error and ECC Analysis for Flash Memory
- Flash Correct and Refresh Techniques (FCR)
- Evaluation
- Conclusions

Conclusion

- NAND flash memory lifetime is limited due to uncorrectable errors, which increase over lifetime (P/E cycles)
- Observation: Dominant source of errors in flash memory is retention errors → retention error rate limits lifetime
- Flash Correct-and-Refresh (FCR) techniques reduce retention error rate to improve flash lifetime
    - Periodically read, correct, and remap or reprogram each page before it accumulates more errors than can be corrected
    - Adapt refresh period to the severity of errors
- FCR improves flash lifetime by 46X at no hardware cost
    - More effective and efficient than stronger ECC
    - Can enable better flash memory scaling

Flash Correct-and-Refresh

Retention-Aware Error Management for Increased Flash Memory Lifetime

Yu Cai[1] Gulay Yalcin[2] Onur Mutlu[1] Erich F. Haratsch[3]
Adrian Cristal[2] Osman S. Unsal[2] Ken Mai[1]

[1] Carnegie Mellon University
[2] Barcelona Supercomputing Center
[3] LSI Corporation

SAFARI Carnegie Mellon