ThyNVM

Software-Transparent Crash Consistency for Persistent Memory

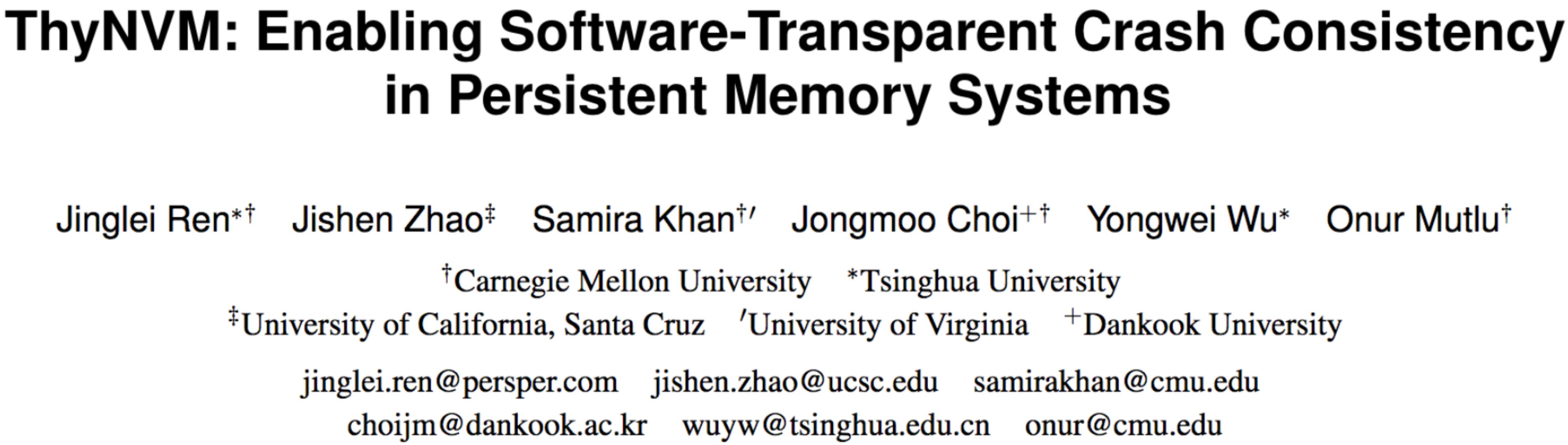
Onur Mutlu
omutlu@ethz.ch

(joint work with Jinglei Ren, Jishen Zhao, Samira Khan, Jongmoo Choi, Yongwei Wu)

August 8, 2016

Flash Memory Summit 2016, Santa Clara, CA

Original Paper (I)



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

Jinglei Ren[*†] Jishen Zhao[‡] Samira Khan[†′] Jongmoo Choi[+†] Yongwei Wu[*] Onur Mutlu[†]

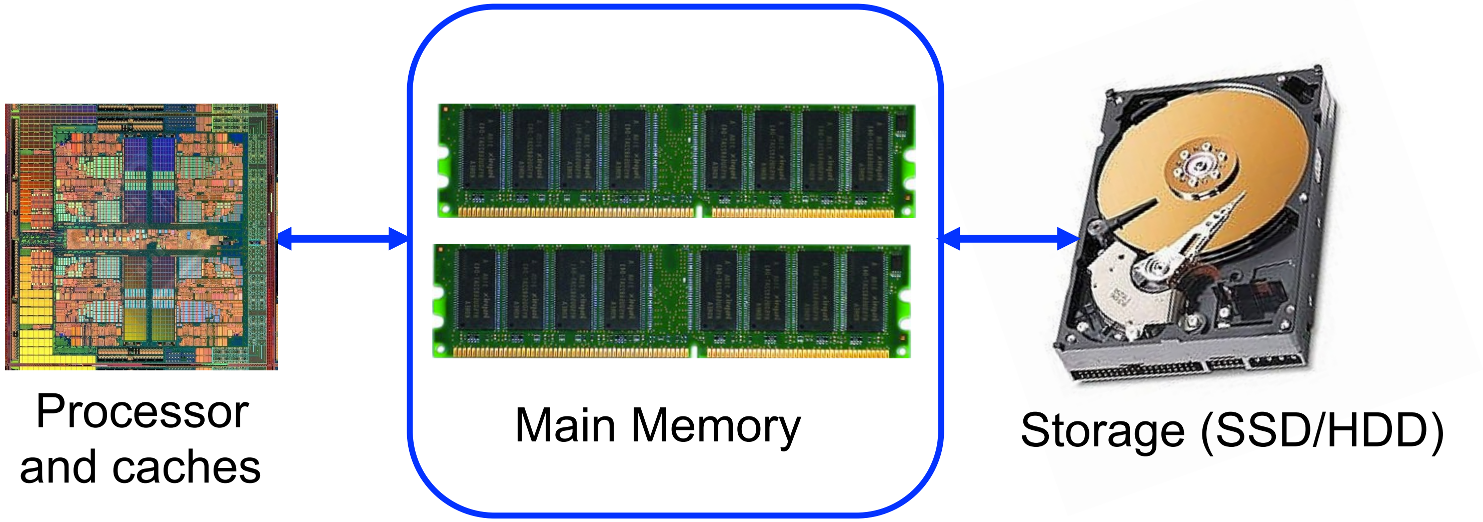
[†]Carnegie Mellon University [*]Tsinghua University
[‡]University of California, Santa Cruz [′]University of Virginia [+]Dankook University

jinglei.ren@persper.com jishen.zhao@ucsc.edu samirakhan@cmu.edu
choijm@dankook.ac.kr wuyw@tsinghua.edu.cn onur@cmu.edu

Original Paper (II)

- Presented at ACM/IEEE MICRO Conference in Dec 2015.
- Full paper for details:
  - Jinglei Ren, Jishen Zhao, Samira Khan, Jongmoo Choi, Yongwei Wu, and Onur Mutlu,
    "ThyNVM: Enabling Software-Transparent Crash Consistency in Persistent Memory Systems"
    Proceedings of the
    48th International Symposium on Microarchitecture (***MICRO***), Waikiki, Hawaii, USA, December 2015.
    [Slides (pptx) (pdf)] [Lightning Session Slides (pptx) (pdf)]
    [Poster (pptx) (pdf)]
    [Source Code]
  - https://users.ece.cmu.edu/~omutlu/pub/ThyNVM-transparent-crash-consistency-for-persistent-memory_micro15.pdf

The Main Memory System

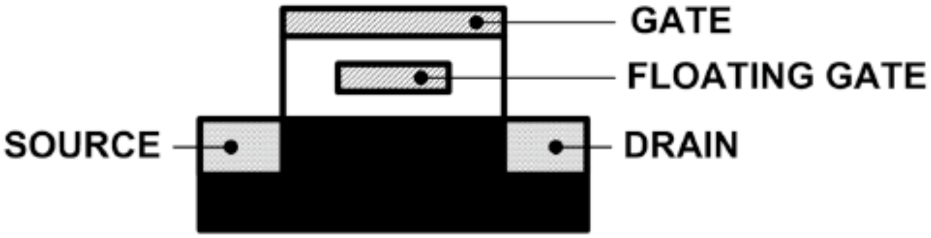


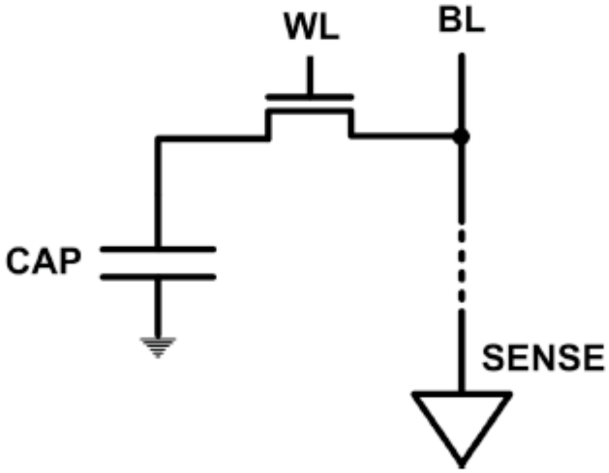
- Main memory is a critical component of all computing systems: server, mobile, embedded, desktop, sensor

- Main memory system must scale (in *size*, *technology*, *efficiency*, *cost*, and *management algorithms*) to maintain performance growth and technology scaling benefits

Limits of Charge Memory

- Difficult charge placement and control
  - Flash: floating gate charge
  - DRAM: capacitor charge, transistor leakage
- Reliable sensing becomes difficult as charge storage unit size reduces

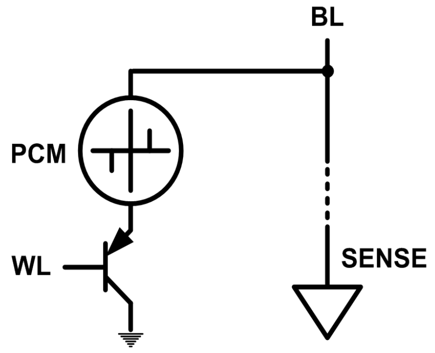




Emerging NVM Technologies

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

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



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

Promising NVM Technologies

- PCM
  - Inject current to change material phase
  - Resistance determined by phase

- STT-MRAM
  - Inject current to change magnet polarity
  - Resistance determined by polarity

- Memristors/RRAM/ReRAM
  - Inject current to change atomic structure
  - Resistance determined by atom distance

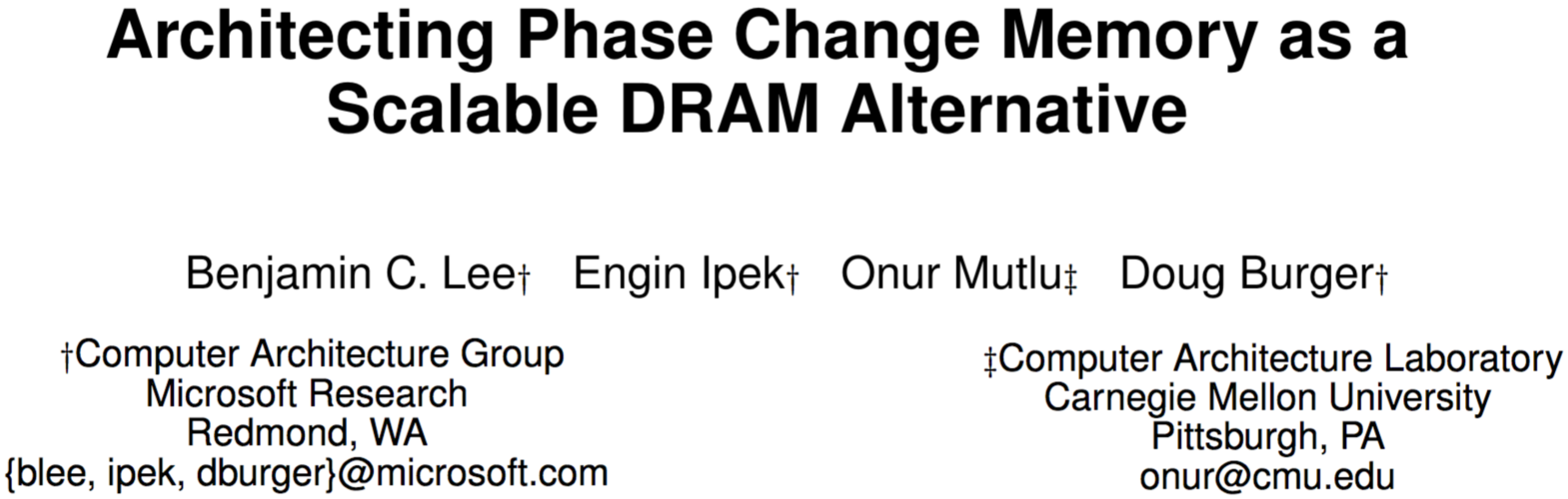
NVM as Main Memory Replacement

- **Very promising**
  - **persistence, high capacity, OK latency, low idle power**

- **Can enable merging of memory and storage**

- Two example works that show benefits
  - Lee, Ipek, Mutlu, Burger, "Architecting Phase Change Memory as a Scalable DRAM Alternative," ISCA 2009.
  - Kultursay, Kandemir, Sivasubramaniam, Mutlu, "Evaluating STT-RAM as an Energy-Efficient Main Memory Alternative," ISPASS 2013.

Two Example Works

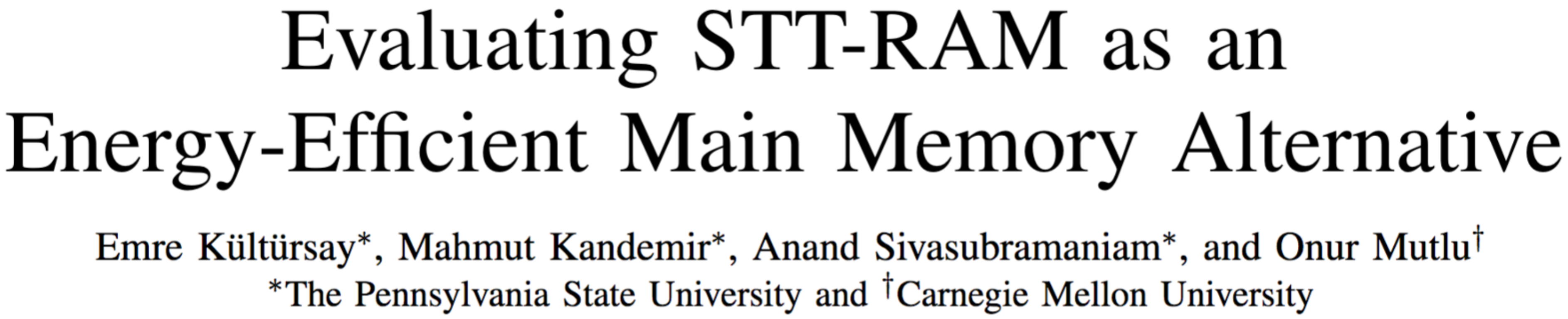


Architecting Phase Change Memory as a Scalable DRAM Alternative

Benjamin C. Lee† Engin Ipek† Onur Mutlu‡ Doug Burger†

†Computer Architecture Group
Microsoft Research
Redmond, WA
{blee, ipek, dburger}@microsoft.com

‡Computer Architecture Laboratory
Carnegie Mellon University
Pittsburgh, PA
onur@cmu.edu

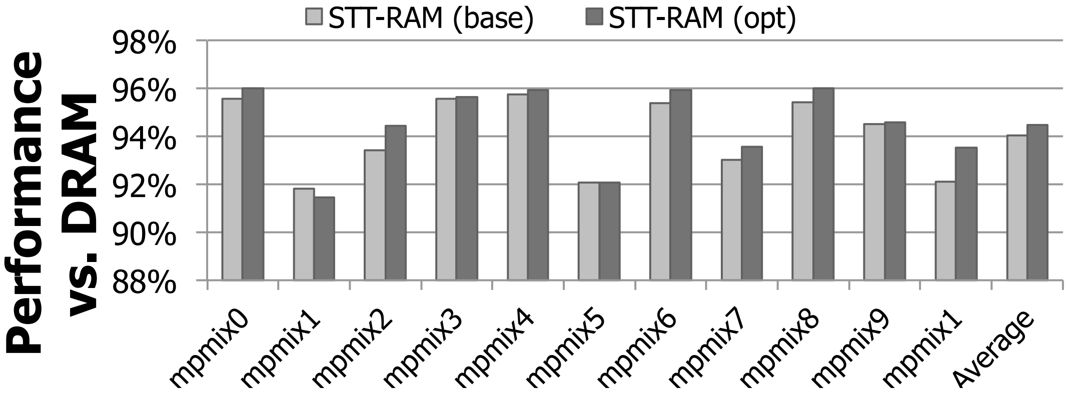


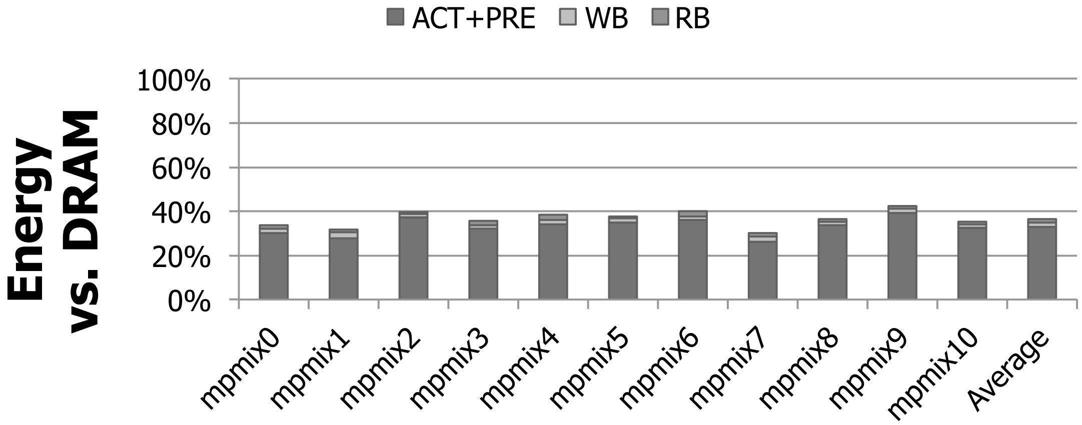
Evaluating STT-RAM as an Energy-Efficient Main Memory Alternative

Emre Kültürsay*, Mahmut Kandemir*, Anand Sivasubramaniam*, and Onur Mutlu†
*The Pennsylvania State University and †Carnegie Mellon University

Architected STT-MRAM as Main Memory

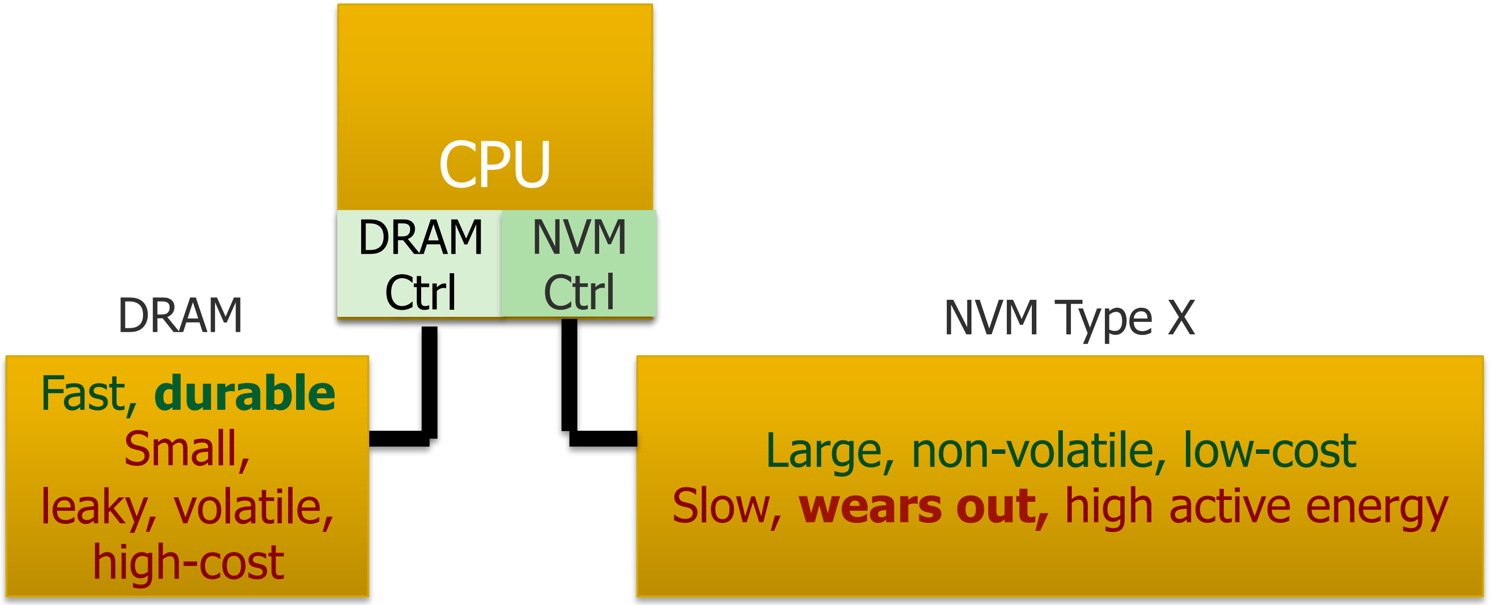
- 4-core, 4GB main memory, multiprogrammed workloads
- ~6% performance loss, ~60% energy savings vs. DRAM





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

A More Viable Approach: Hybrid Memory Systems



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

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

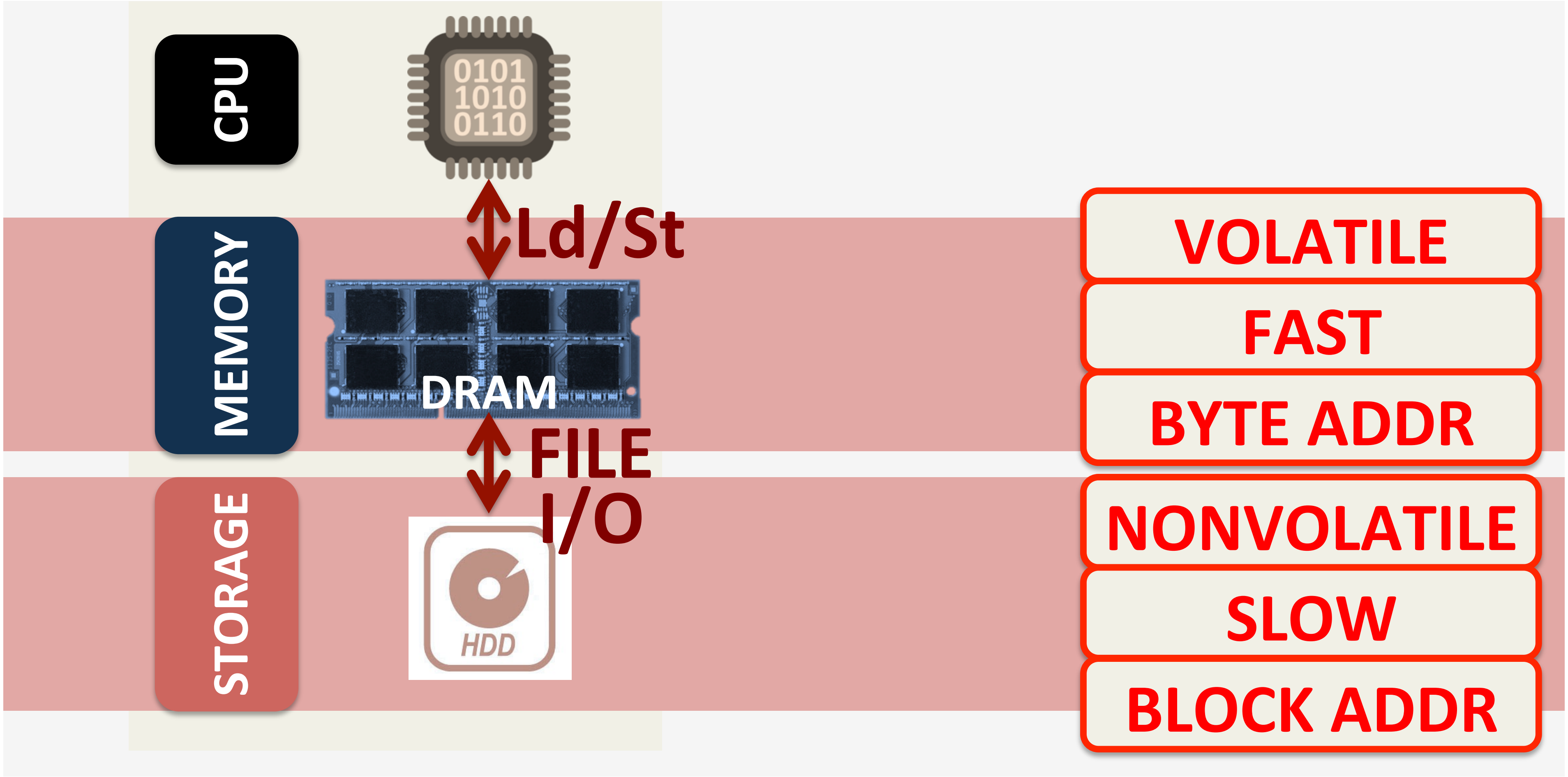
Yoon+, "Row Buffer Locality Aware Caching Policies for Hybrid Memories," ICCD 2012 Best Paper Award.

Some Opportunities with Emerging Technologies

- Merging of memory and storage
  - e.g., a single interface to manage all data
- New applications
  - e.g., ultra-fast checkpoint and restore
- More robust system design
  - e.g., reducing data loss
- Processing tightly-coupled with memory
  - e.g., enabling efficient search and filtering

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

TWO-LEVEL STORAGE MODEL



CPU

0101
1010
0110

Ld/St

MEMORY

DRAM

FILE I/O

STORAGE

HDD

VOLATILE

FAST

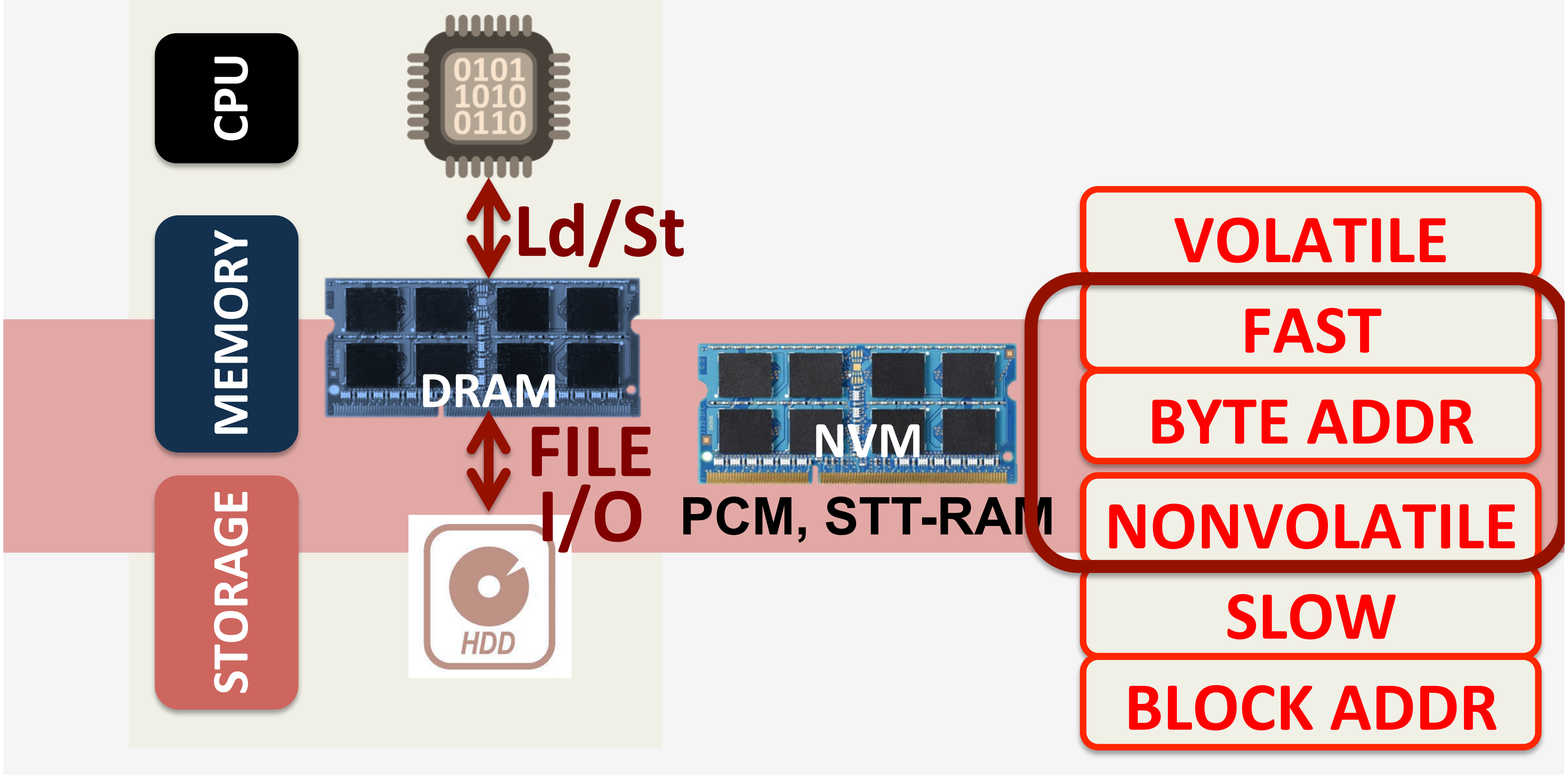
BYTE ADDR

NONVOLATILE

SLOW

BLOCK ADDR

TWO-LEVEL STORAGE MODEL



CPU

MEMORY

STORAGE

Ld/St

DRAM

FILE I/O

NVM

PCM, STT-RAM

VOLATILE

FAST

BYTE ADDR

NONVOLATILE

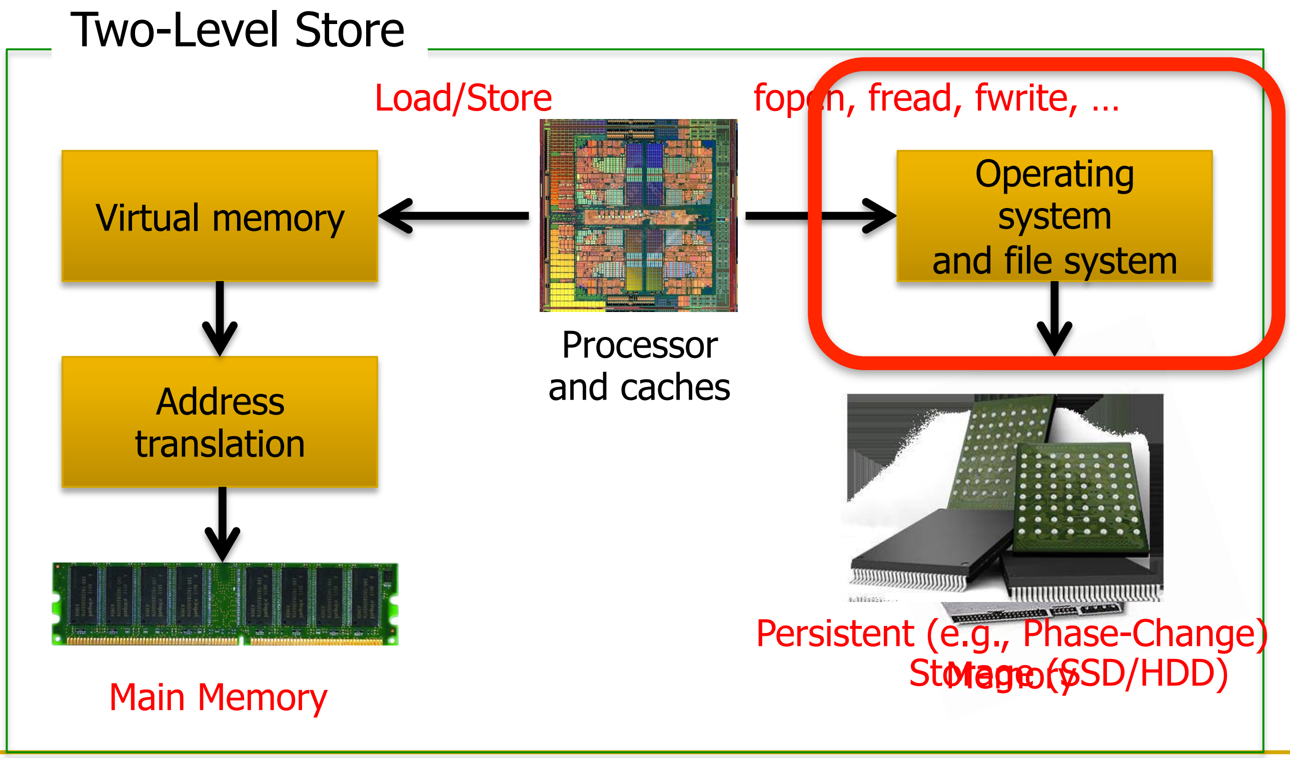
SLOW

BLOCK ADDR

Non-volatile memories combine characteristics of memory and storage

Coordinated Memory and Storage with NVM (I)

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



Two-Level Store

Load/Store

fopen, fread, fwrite, …

Virtual memory

Address translation

Main Memory

Processor and caches

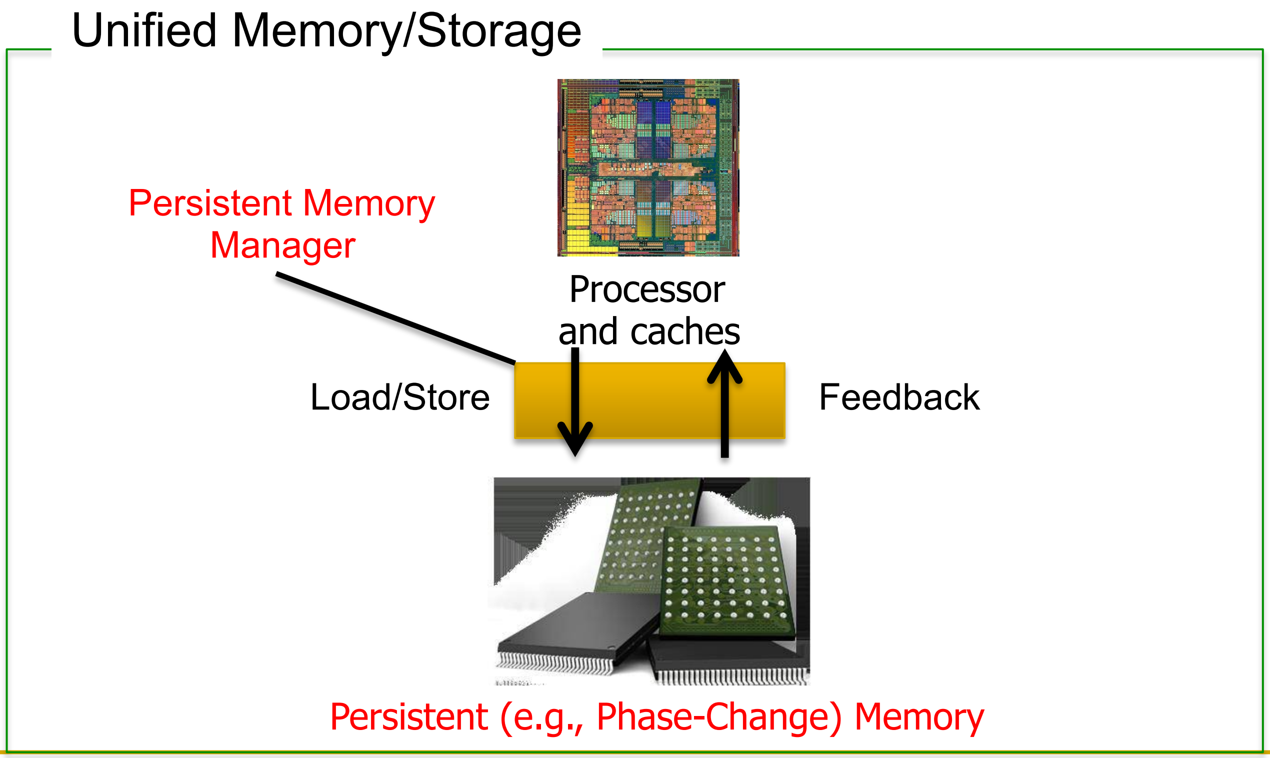
Operating system and file system

Persistent (e.g., Phase-Change) Memory

Storage (SSD/HDD)

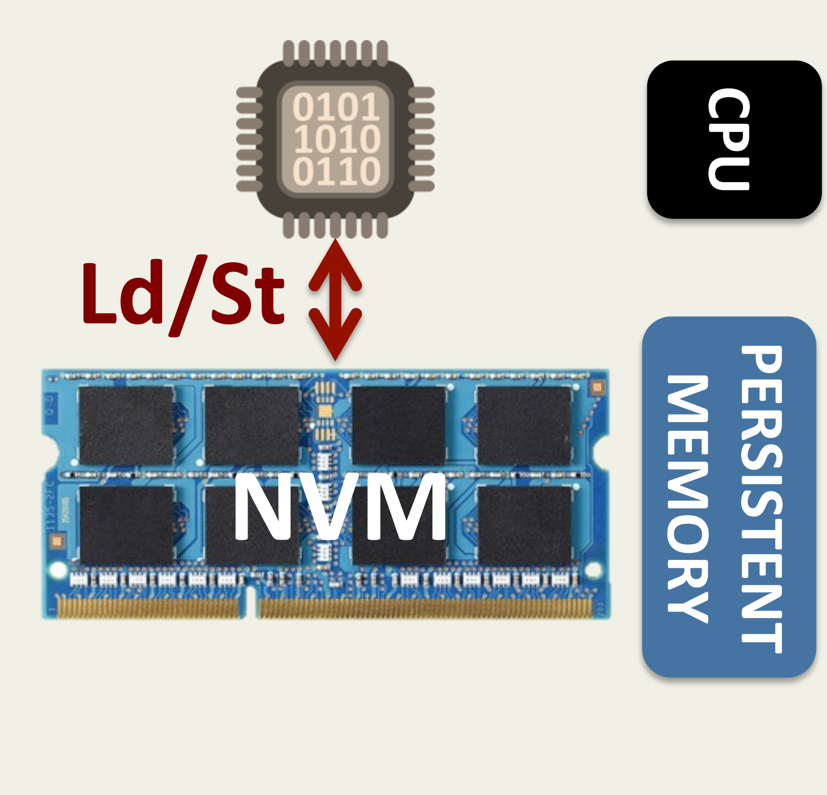
Coordinated Memory and Storage with NVM (II)

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


Unified Memory/Storage
Persistent Memory Manager
Processor and caches
Load/Store
Feedback
Persistent (e.g., Phase-Change) Memory

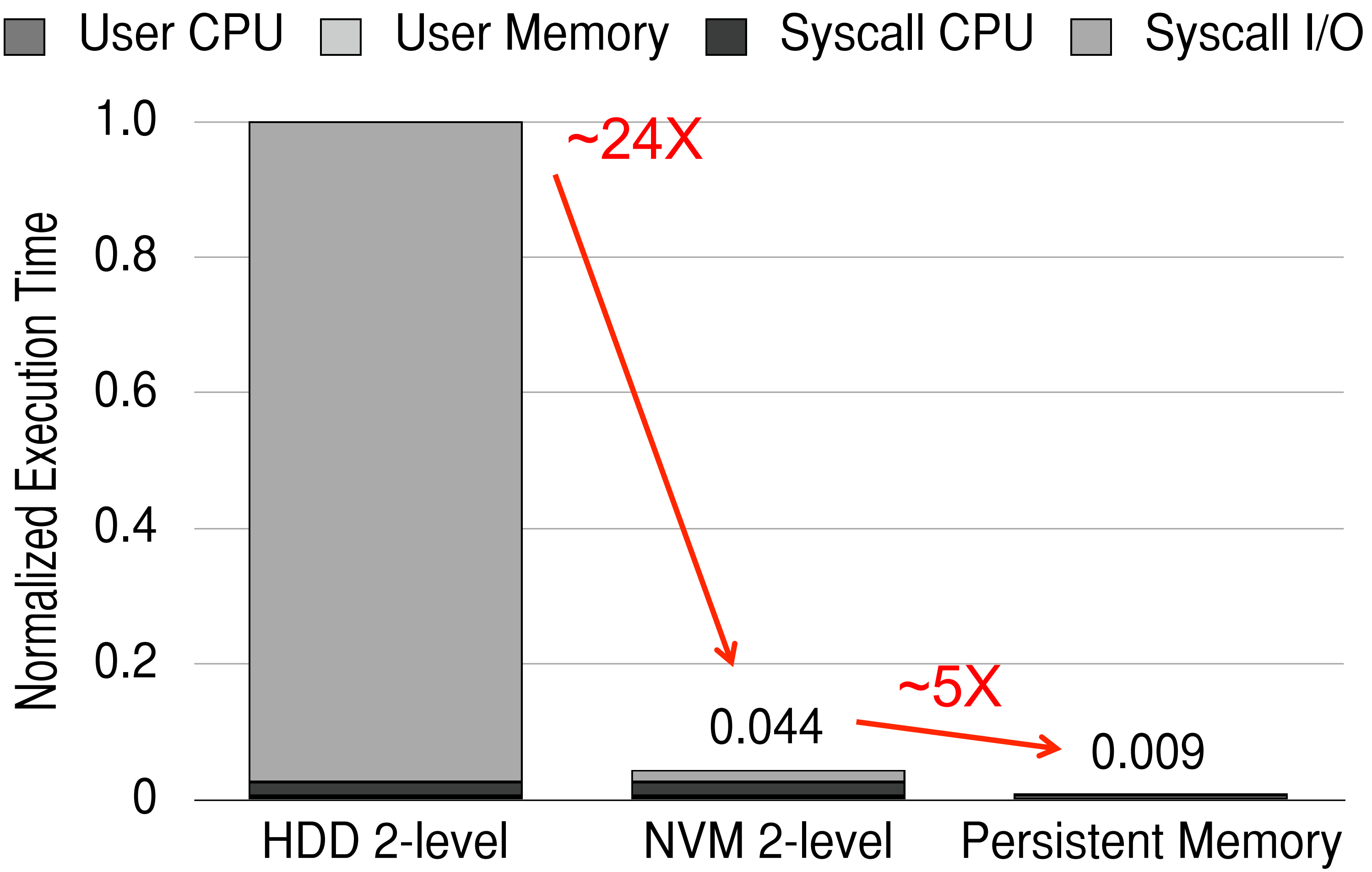
Meza+, "A Case for Efficient Hardware-Software Cooperative Management of Storage and Memory," WEED 2013.

PERSISTENT MEMORY



Provides an opportunity to manipulate persistent data directly

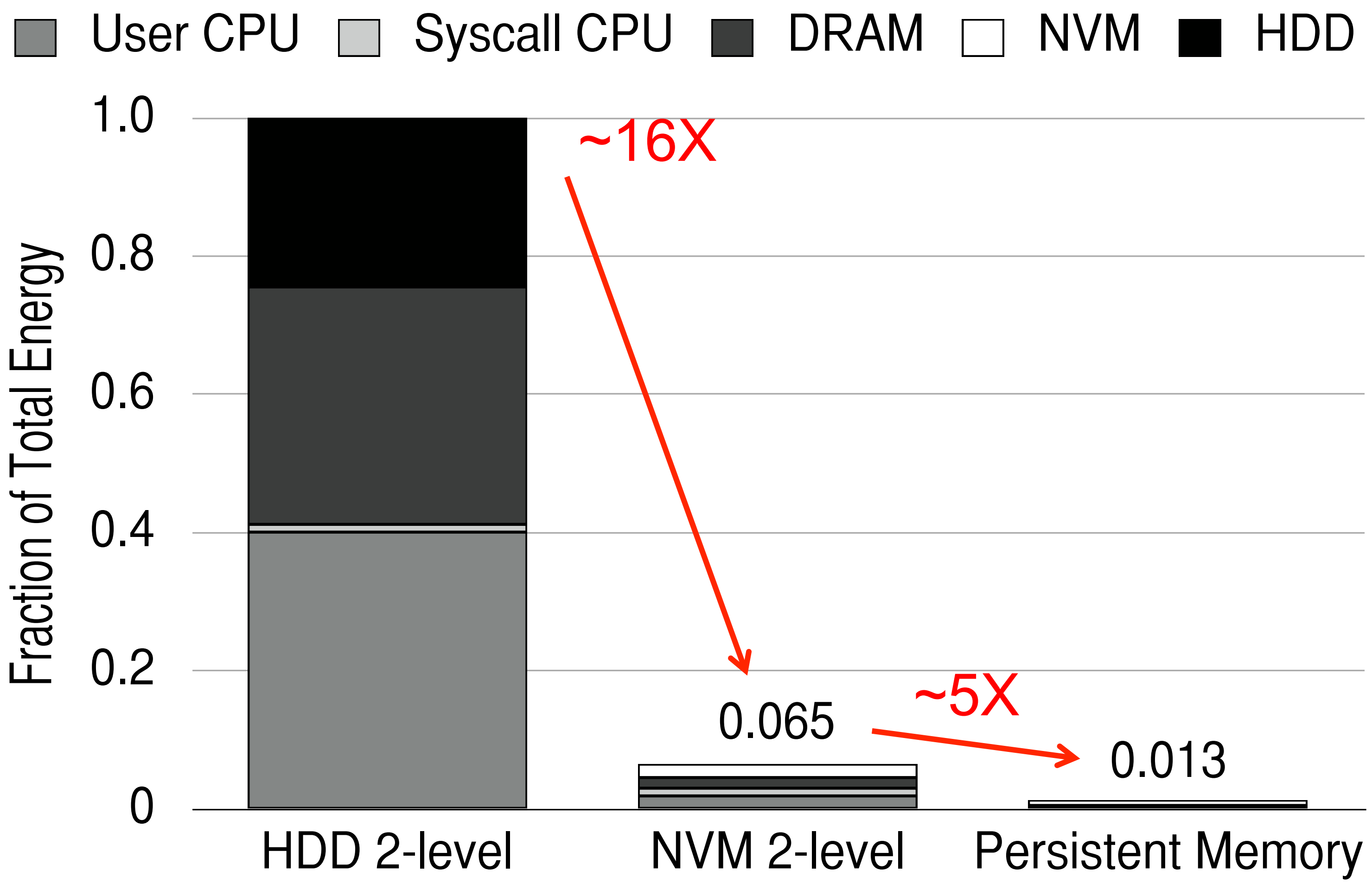
Performance Benefits of Persistent Memory



SAFARI

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

Energy Benefits of Persistent Memory

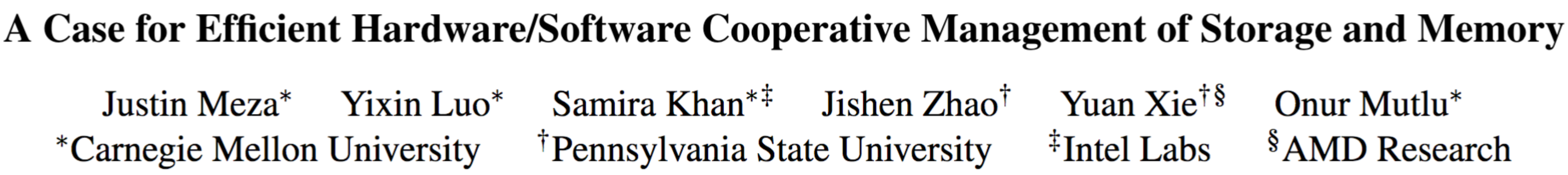


SAFARI

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

On Persistent Memory Benefits & Challenges

- 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)



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

Justin Meza* Yixin Luo* Samira Khan*‡ Jishen Zhao† Yuan Xie†§ Onur Mutlu*
*Carnegie Mellon University †Pennsylvania State University ‡Intel Labs §AMD Research

The Persistent Memory Manager (PMM)

- Exposes a load/store interface to access persistent data
  - Applications can directly access persistent memory → no conversion, translation, location overhead for persistent data

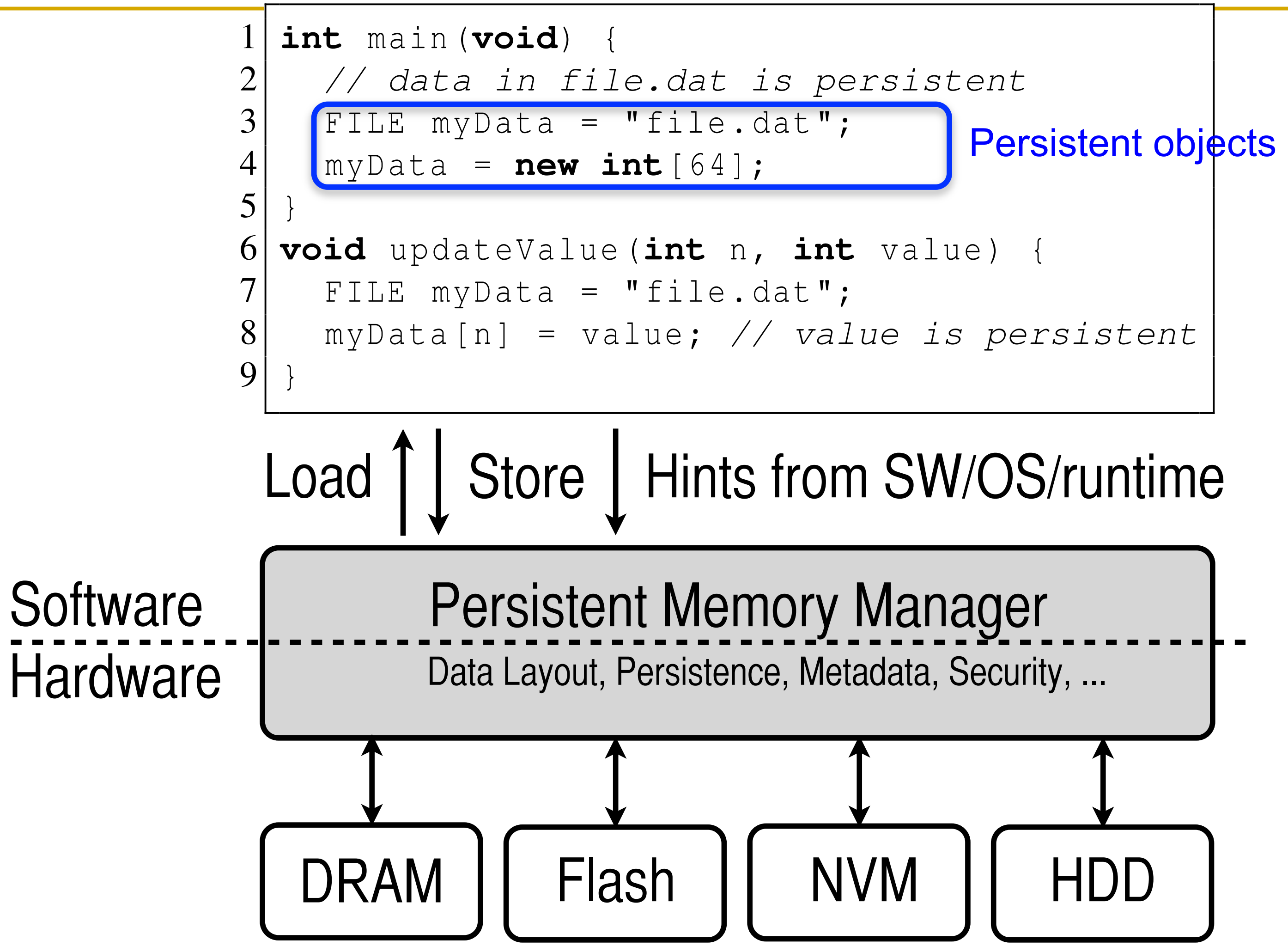
- Manages data placement, location, persistence, security
  - To get the best of multiple forms of storage

- Manages metadata storage and retrieval
  - This can lead to overheads that need to be managed

- Exposes hooks and interfaces for system software
  - To enable better data placement and management decisions

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

The Persistent Memory Manager (PMM)



```
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

One Key Challenge

- How to ensure consistency of system/data if all memory is persistent?

- Two extremes
  - Programmer transparent: Let the system handle it
  - Programmer only: Let the programmer handle it

- Many alternatives in-between…

CHALLENGE: CRASH CONSISTENCY

Persistent Memory System

System crash can result in permanent data corruption in NVM

CURRENT SOLUTIONS

Explicit interfaces to manage consistency

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

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

Limits adoption of NVM

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

Burden on the programmers

OUR APPROACH: ThyNVM

Goal:
Software-transparent crash consistency in persistent memory systems

ThyNVM: Summary

A new hardware-based checkpointing mechanism

- **Checkpoints** at *multiple granularities* to reduce both checkpointing latency and metadata overhead

- **Overlaps** *checkpointing* and *execution to* reduce checkpointing latency

- **Adapts** to *DRAM and NVM* characteristics

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

OUTLINE

Crash Consistency Problem

Current Solutions

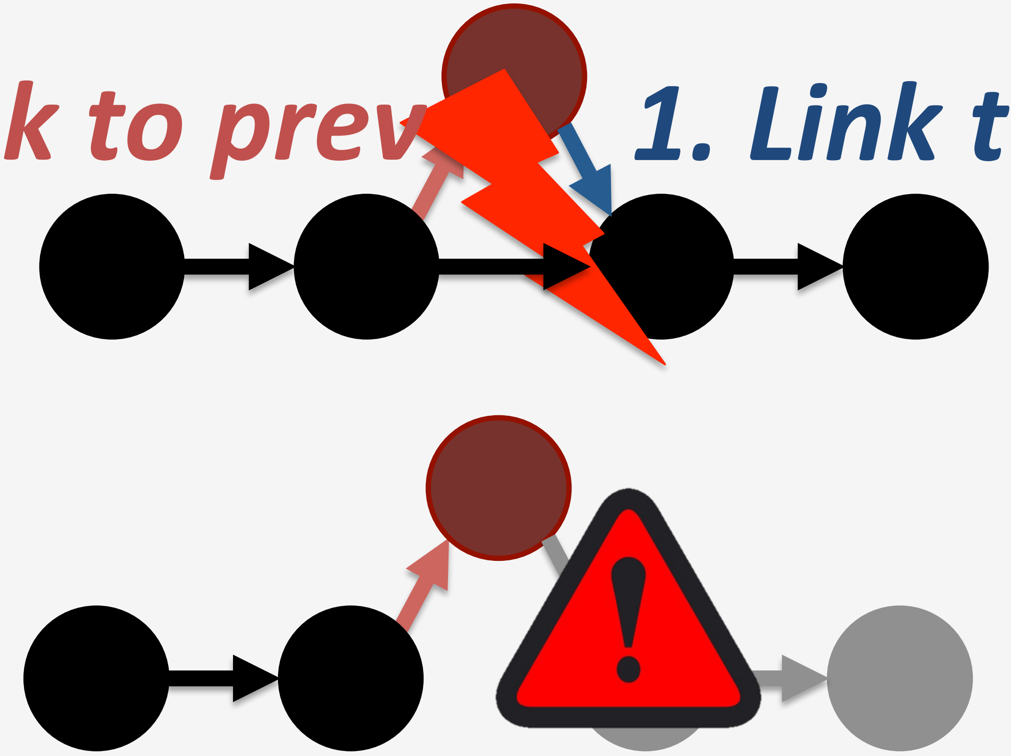
ThyNVM

Evaluation

Conclusion

CRASH CONSISTENCY PROBLEM

Add a node to a linked list



2. Link to prev

1. Link to next

System crash can result in inconsistent memory state

OUTLINE

Crash Consistency Problem

Current Solutions

ThyNVM

Evaluation

Conclusion

CURRENT SOLUTIONS

Explicit interfaces to manage consistency

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

Example Code

update a node in a persistent hash table

```
void hashtable_update(hashtable_t* ht,
                void *key, void *data)
{
   list_t* chain = get_chain(ht, key);
   pair_t* pair;
   pair_t updatePair;
   updatePair.first = key;
   pair = (pair_t*) list_find(chain,
                          &updatePair);
   pair->second = data;
}
```

CURRENT SOLUTIONS

```
void TMhashtable_update(TMARCGDECL
hashtable_t* ht, void *key, void*data)
{
  list_t* chain = get_chain(ht, key);
  pair_t* pair;
  pair_t updatePair;
  updatePair.first = key;
  pair = (pair_t*) TMLIST_FIND(chain,
                        &updatePair);
  pair->second = data;
}
```

CURRENT SOLUTIONS

Manual declaration of persistent components

```
void TMhashtable_update(TMARCGDECL
hashtable_t* ht, void *key, void*data)
{
  list_t* chain = get_chain(ht, key);
  pair_t* pair;
  pair_t updatePair;
  updatePair.first = key;
  pair = (pair_t*) TMLIST_FIND(chain,
                        &updatePair);
  pair->second = data;
}
```

CURRENT SOLUTIONS

Manual declaration of persistent components

```
void TMhashtable_update(TMARCGDECL
hashtable_t* ht, void *key, void*data)
{
    list_t* chain = get_chain(ht, key);
    pair_t* pair;
    pair_t updatePair;
    updatePair.first = key;
    pair = (pair_t*) TMLIST_FIND(chain,
                         &updatePair);
    pair->second = data;
}
```

Need a new implementation

CURRENT SOLUTIONS

Manual declaration of persistent components

```
void TMhashtable_update(TMARCGDECL
hashtable_t* ht, void *key, void*data)
{
  list_t* chain = get_chain(ht, key);
  pair_t* pair;
  pair_t updatePair;
  updatePair.first = key;
  pair = (pair_t*) TMLIST_FIND(chain,
                       &updatePair);

  pair->second = data;
}
```

Need a new implementation

Third party code can be inconsistent

CURRENT SOLUTIONS

Manual declaration of persistent components

```
void TMhashtable_update(TMARCGDECL
hashtable_t* ht, void *key, void*data)
{
   list_t* chain = get_chain(ht, key);
   pair_t* pair;
   pair_t updatePair;
   updatePair.first = key;
   pair = (pair_t*) TMLIST_FIND(chain,
                     &updatePair);
   pair->second = data;
}
```

Need a new implementation

Third party code can be inconsistent

Prohibited Operation =

Burden on the programmers

OUTLINE

Crash Consistency Problem

Current Solutions

ThyNVM

Evaluation

Conclusion

OUR GOAL

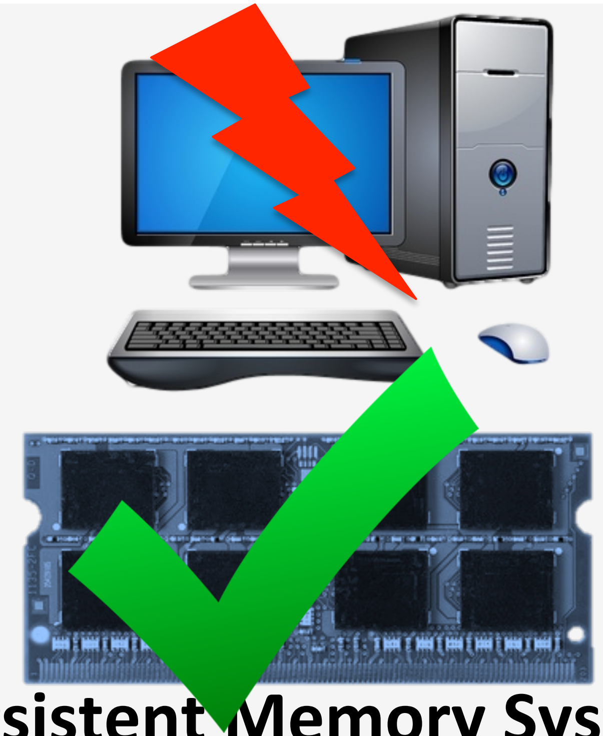
Software transparent consistency in persistent memory systems

- **Execute** *legacy applications*
- **Reduce burden** *on programmers*
- **Enable** *easier integration of NVM*

NO MODIFICATION IN THE CODE

```
void hashtable_update(hashtable_t* ht,
               void *key, void *data)
{
list_t* chain = get_chain(ht, key);
pair_t* pair;
pair_t updatePair;
updatePair.first = key;
pair = (pair_t*) list_find(chain,
                        &updatePair);
pair->second = data;
}
```

RUN THE EXACT SAME CODE…



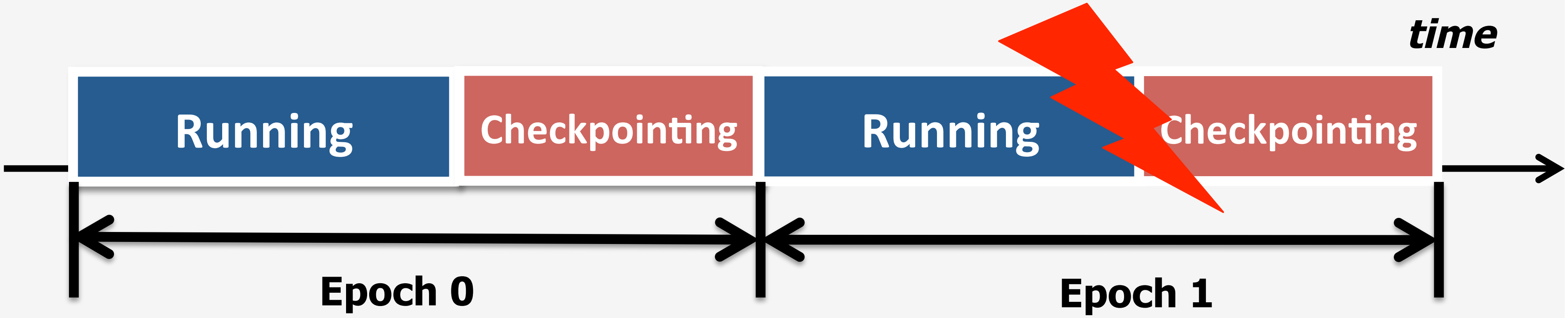
Persistent Memory System

```
void hashtable_update(hashtable_t* ht,
               void *key, void *data){
  list_t* chain = get_chain(ht, key);
  pair_t* pair;
  pair_t updatePair;
  updatePair.first = key;
  pair = (pair_t*) list_find(chain,
                         &updatePair);
  pair->second = data;
}
```

Software transparent memory crash consistency

ThyNVM APPROACH

Periodic checkpointing of data managed by hardware



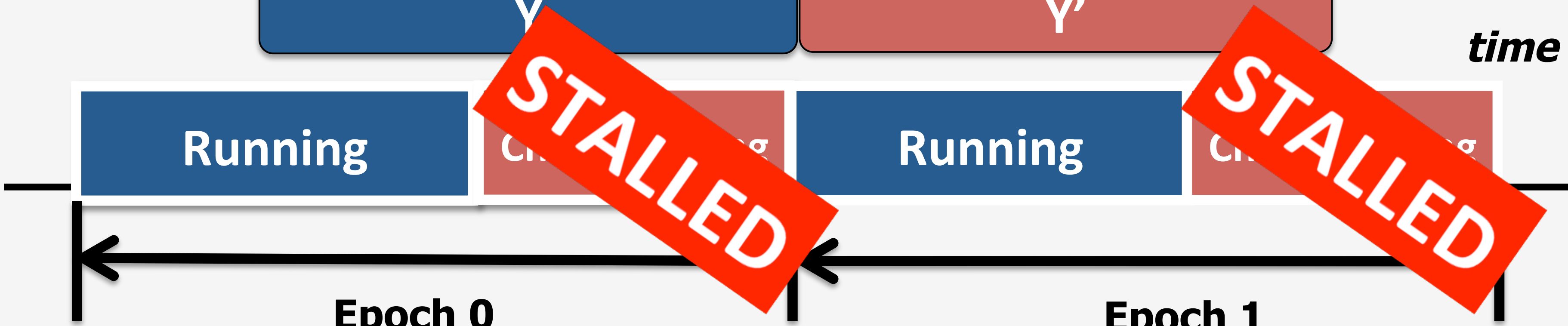
Transparent to application and system

CHECKPOINTING OVERHEAD

1. Metadata overhead

Metadata Table

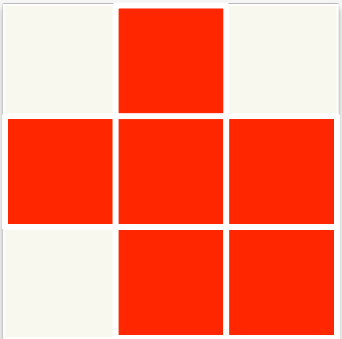
Working location	Checkpoint location
X	X’
Y	Y’



2. Checkpointing latency

1. METADATA AND CHECKPOINTING GRANULARITY

Working location	Checkpoint location
X	X’
Y	Y’



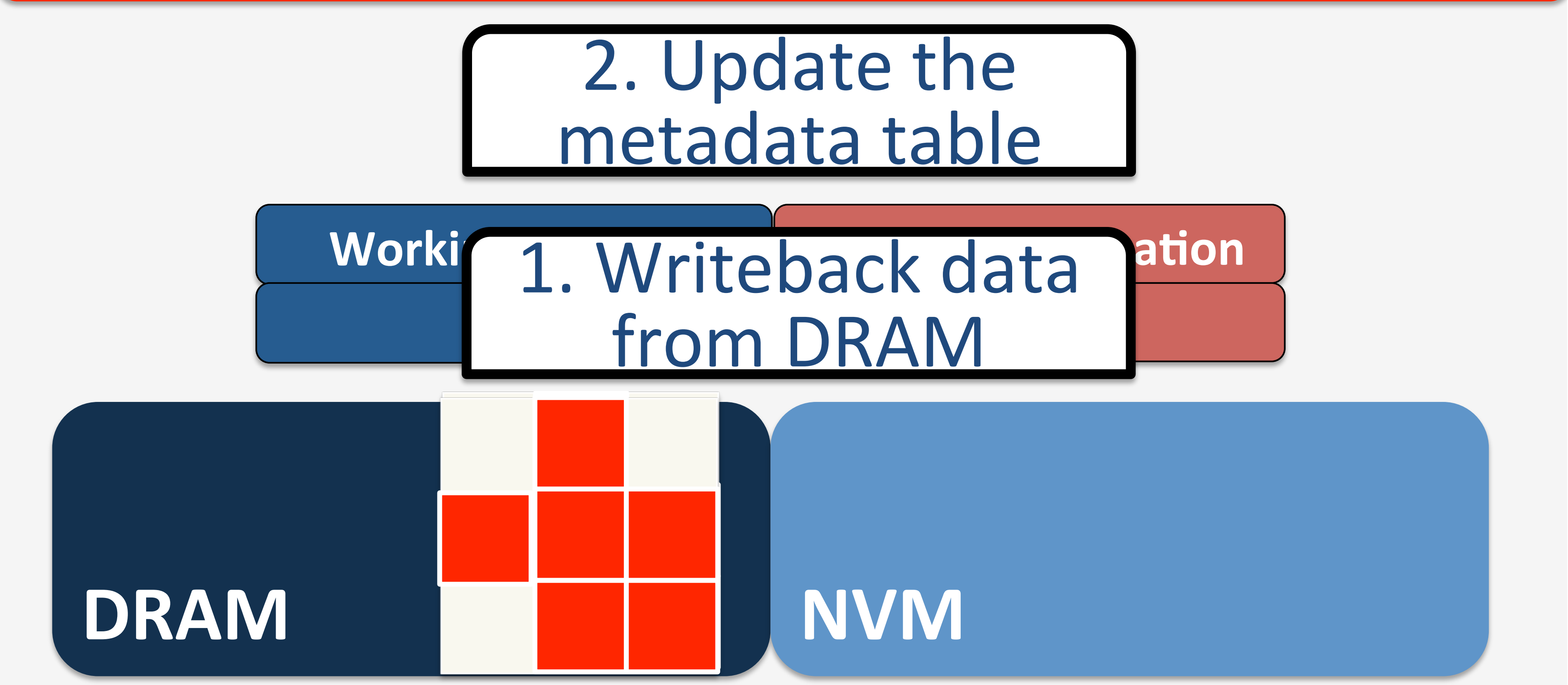
PAGE

CACHE BLOCK

PAGE GRANULARITY	BLOCK GRANULARITY
One Entry Per Page Small Metadata	One Entry Per Block Huge Metadata

2. LATENCY AND LOCATION

DRAM-BASED WRITEBACK



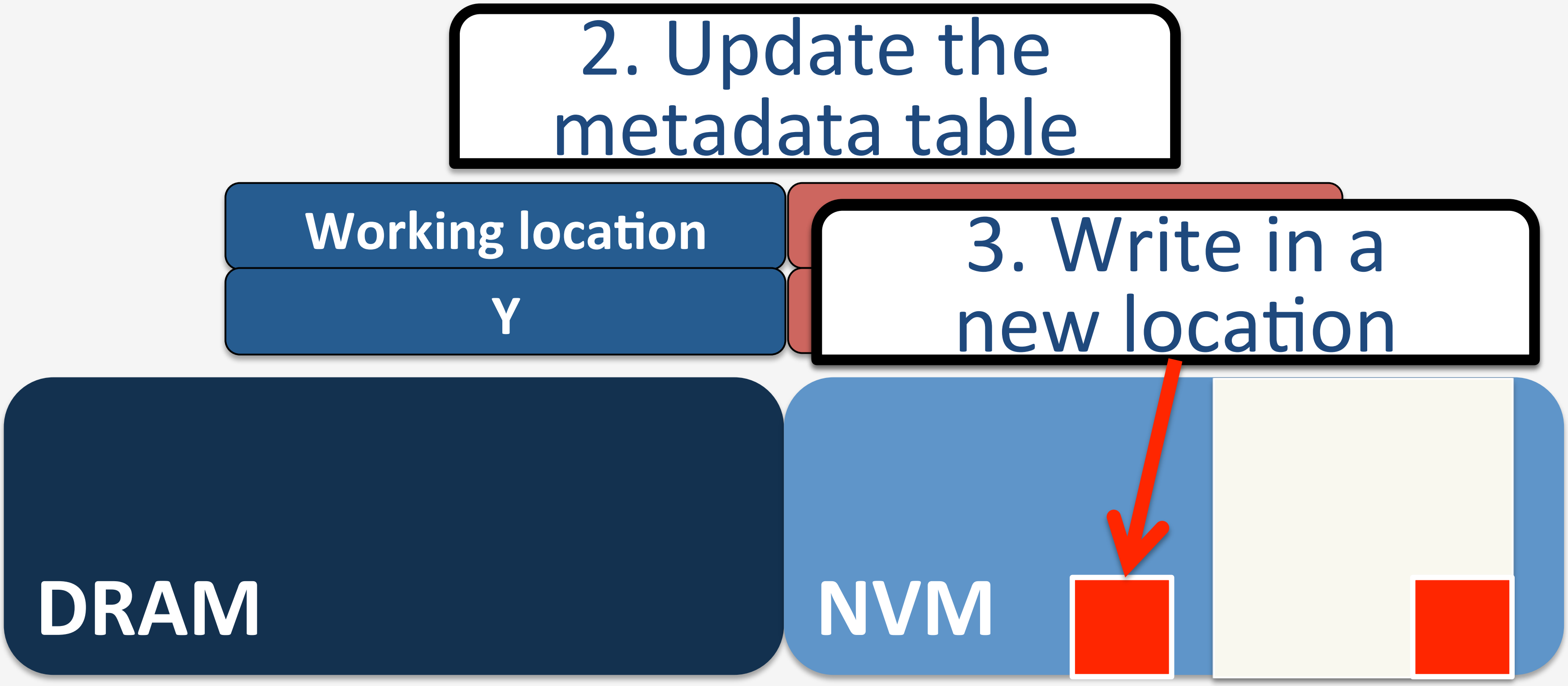
2. Update the metadata table

1. Writeback data from DRAM

Long latency of writing back data to NVM

2. LATENCY AND LOCATION

NVM-BASED REMAPPING



Short latency in NVM-based remapping

ThyNVM KEY MECHANISMS

Checkpointing granularity

- *Small granularity: large metadata*
- *Large granularity: small metadata*

Latency and location

- *Writeback from DRAM: long latency*
- *Remap in NVM: short latency*

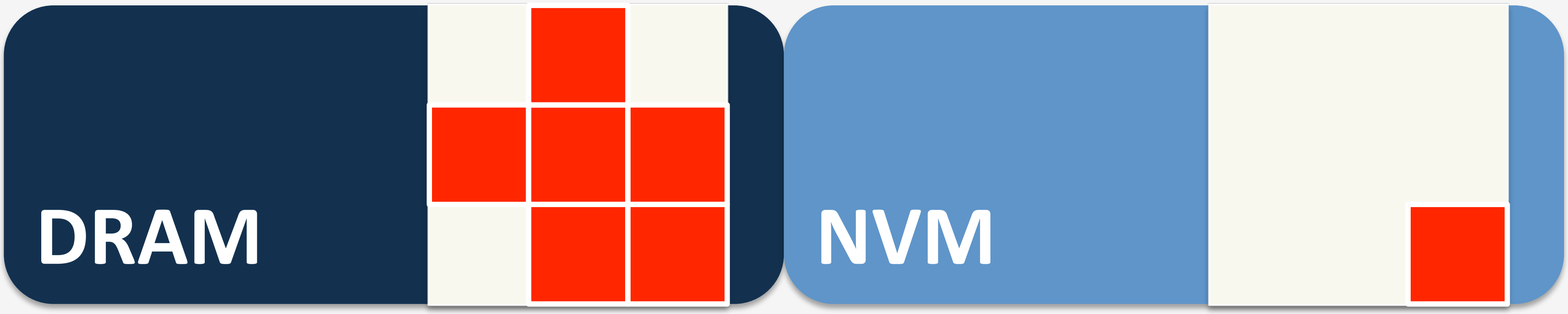
Based on these, we propose two key mechanisms

1. Dual granularity checkpointing
2. Overlap of execution and checkpointing

1. DUAL GRANULARITY CHECKPOINTING

Page Writeback in DRAM

Block Remapping in NVM

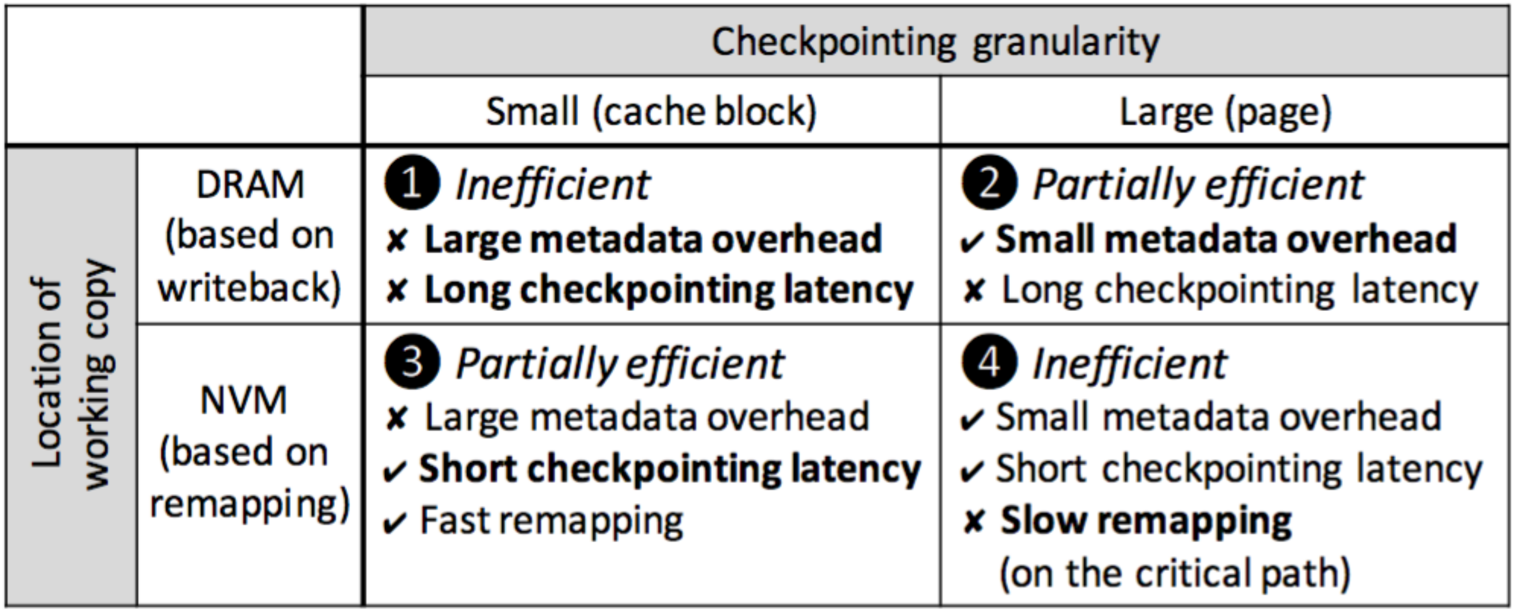


GOOD FOR STREAMING WRITES

GOOD FOR RANDOM WRITES

High write locality pages in DRAM, low write locality pages in NVM

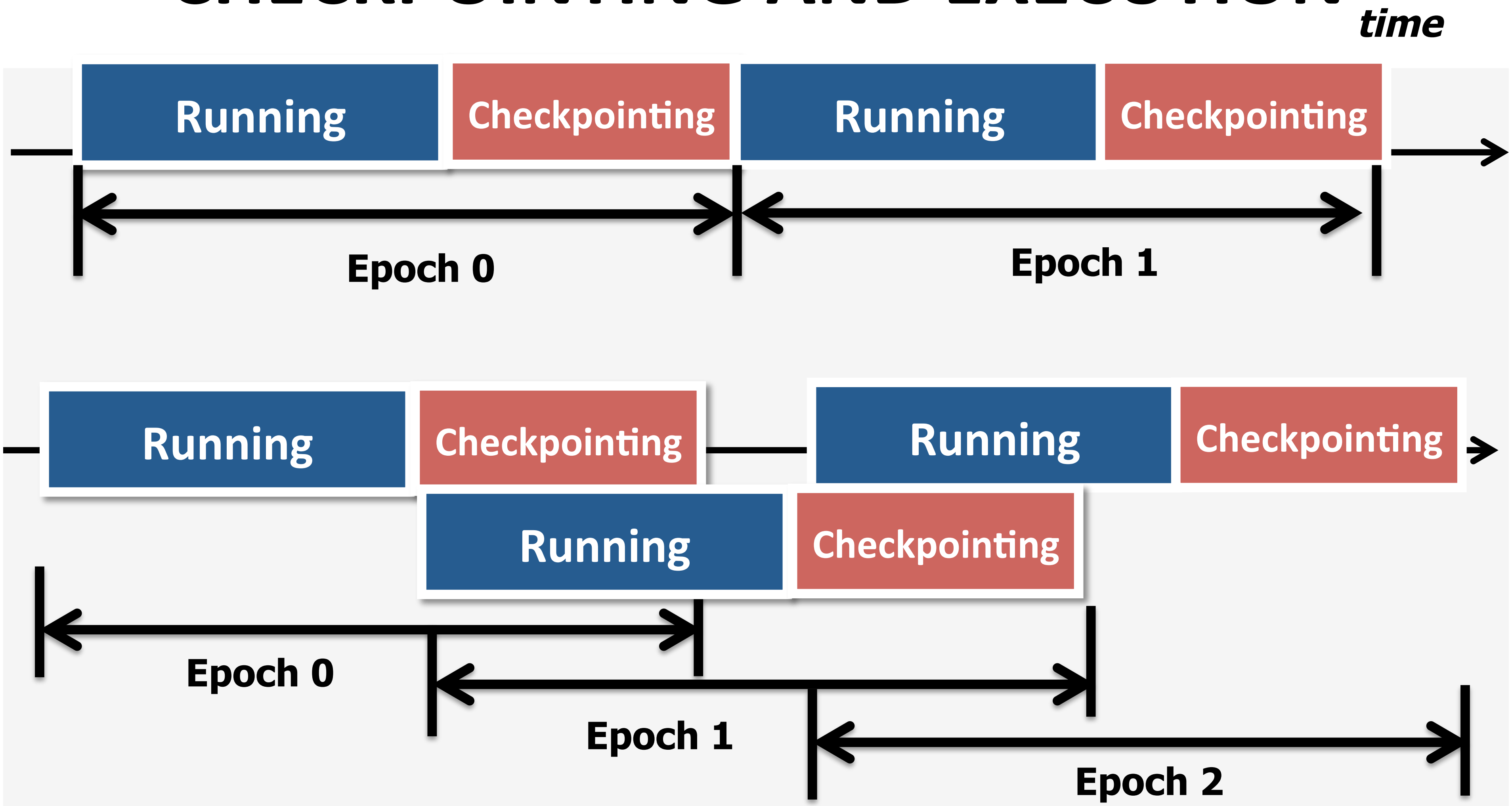
TRADEOFF SPACE



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

Table 1: Tradeoff space of options combining checkpointing granularity choice and location choice of the working copy of data. The table shows four options and their pros and cons. Boldfaced text indicates the most critical pro or con that determines the efficiency of an option.

2. OVERLAPPING CHECKPOINTING AND EXECUTION



Hides the long latency of Page Writeback

OUTLINE

Crash Consistency Problem

Current Solutions

ThyNVM

Evaluation

Conclusion

SYSTEM ARCHITECTURE

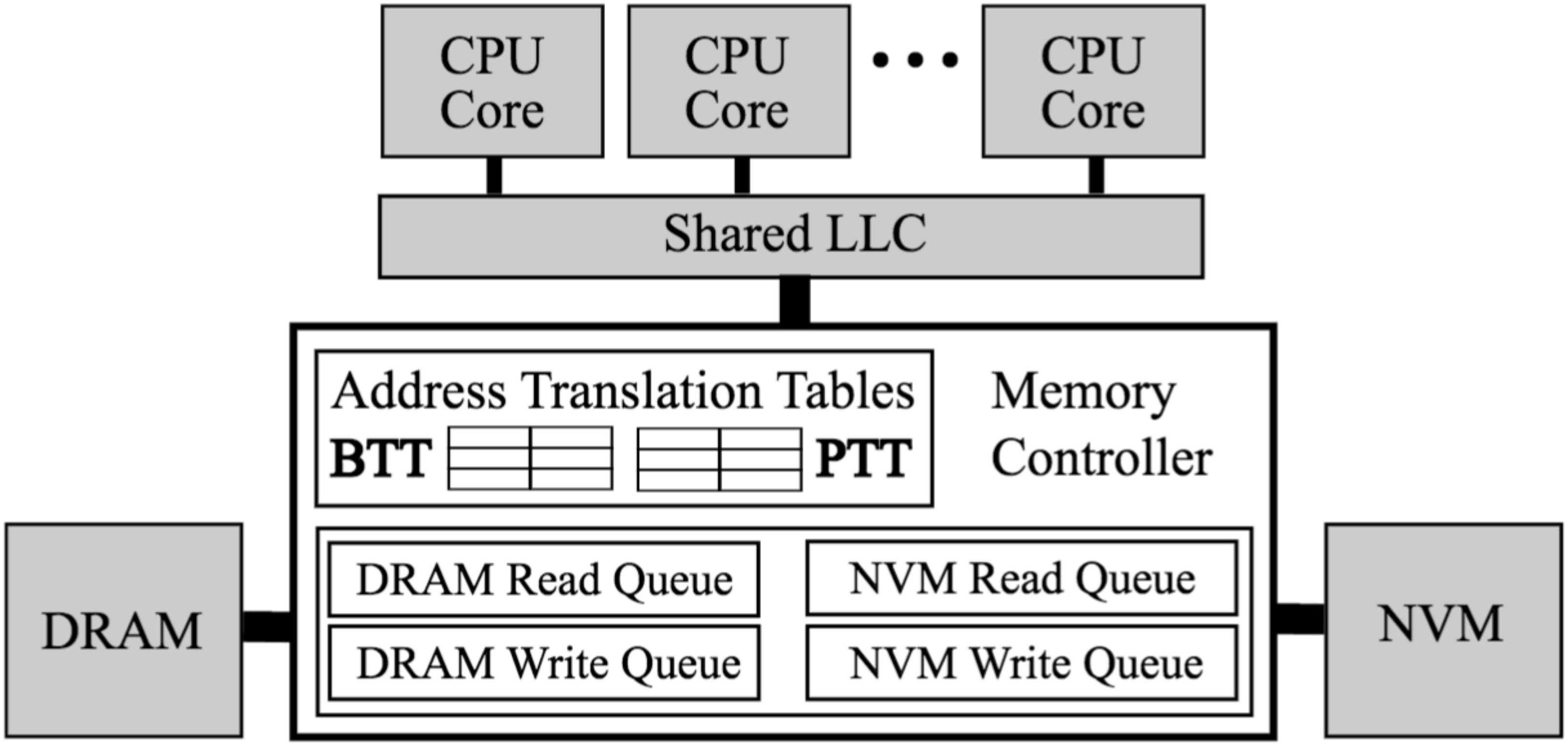
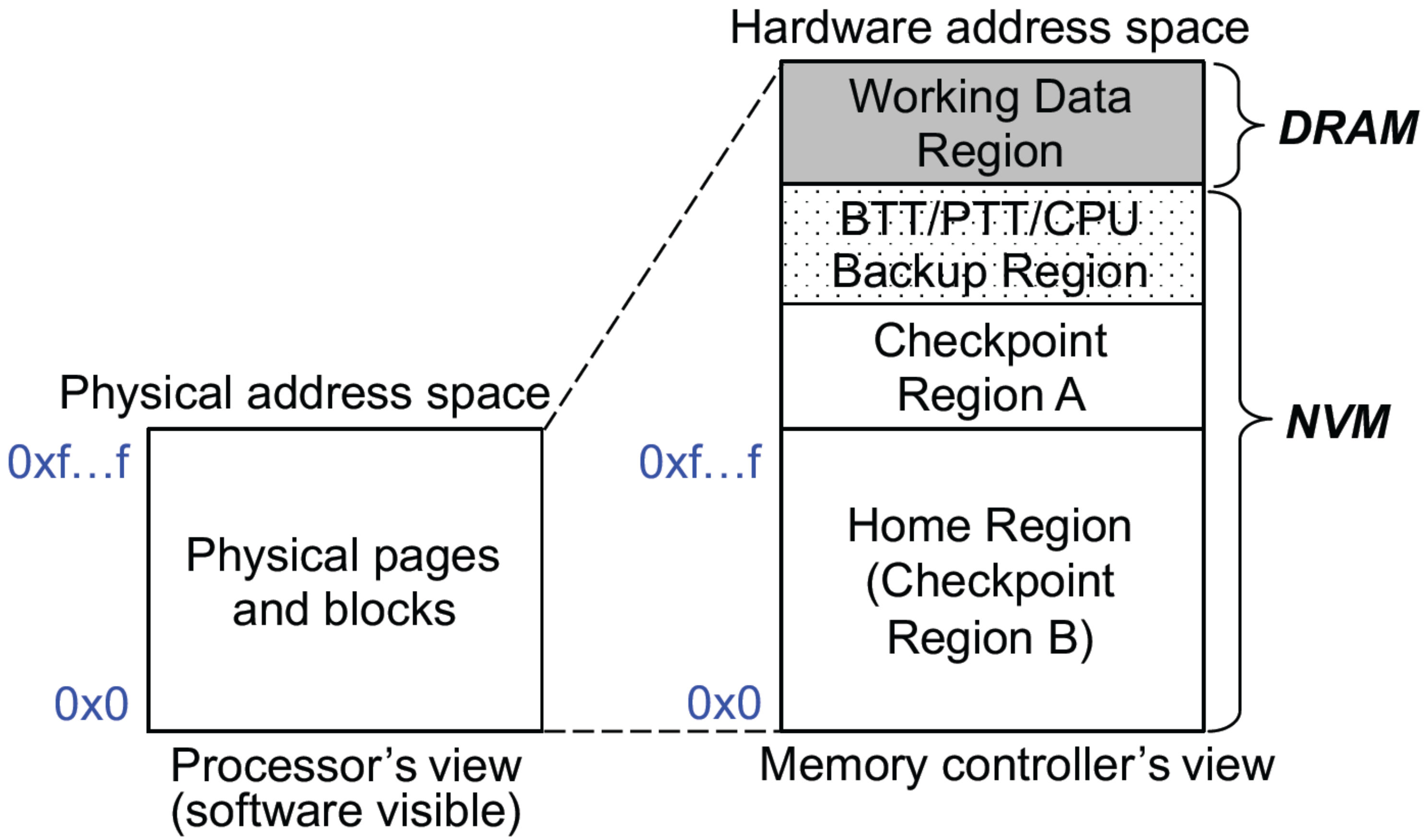


Figure 2: Architecture overview of ThyNVM.

MEMORY ADDRESS SPACE



Working Data Region: W_{active}^{page}, W_{active}^{block} (when creating C_{last})

Ckpt Regions A and B: C_{last}, C_{penult}, W_{active}^{block}

Figure 4: ThyNVM address space layout.

METHODOLOGY

Cycle accurate x86 simulator Gem5

Comparison Points:

Ideal DRAM: DRAM-based, no cost for consistency

– Lowest latency system

Ideal NVM: NVM-based, no cost for consistency

– NVM has higher latency than DRAM

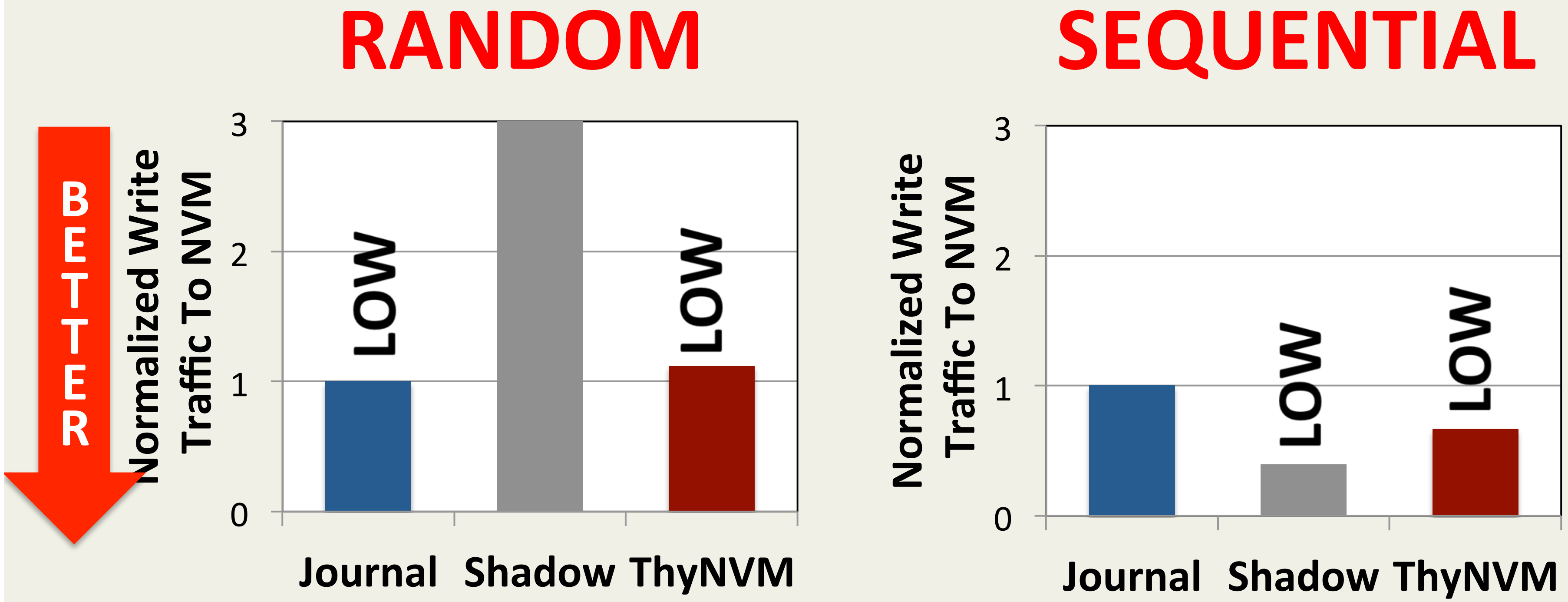
Journaling: Hybrid, commit dirty cache blocks

– Leverages DRAM to buffer dirty blocks

Shadow Paging: Hybrid, copy-on-write pages

– Leverages DRAM to buffer dirty pages

ADAPTIVITY TO ACCESS PATTERN

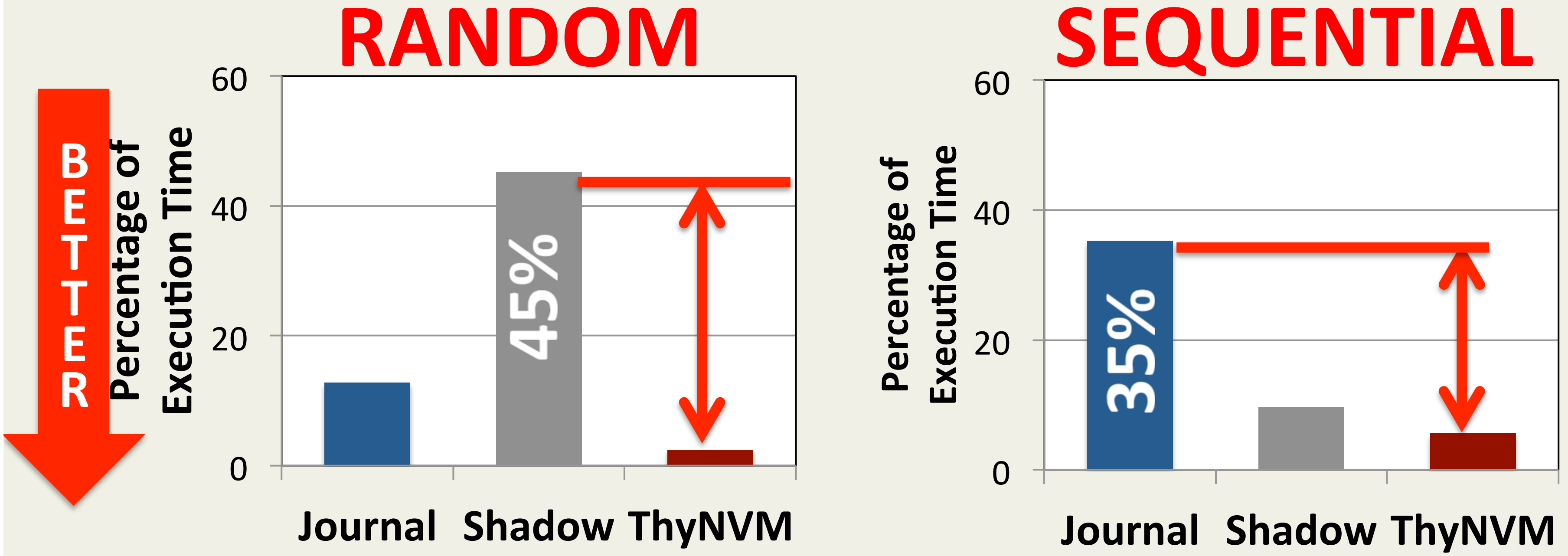

RANDOM

SEQUENTIAL

Journaling is better for Random and Shadow paging is better for Sequential

ThyNVM adapts to both access patterns

OVERLAPPING CHECKPOINTING AND EXECUTION

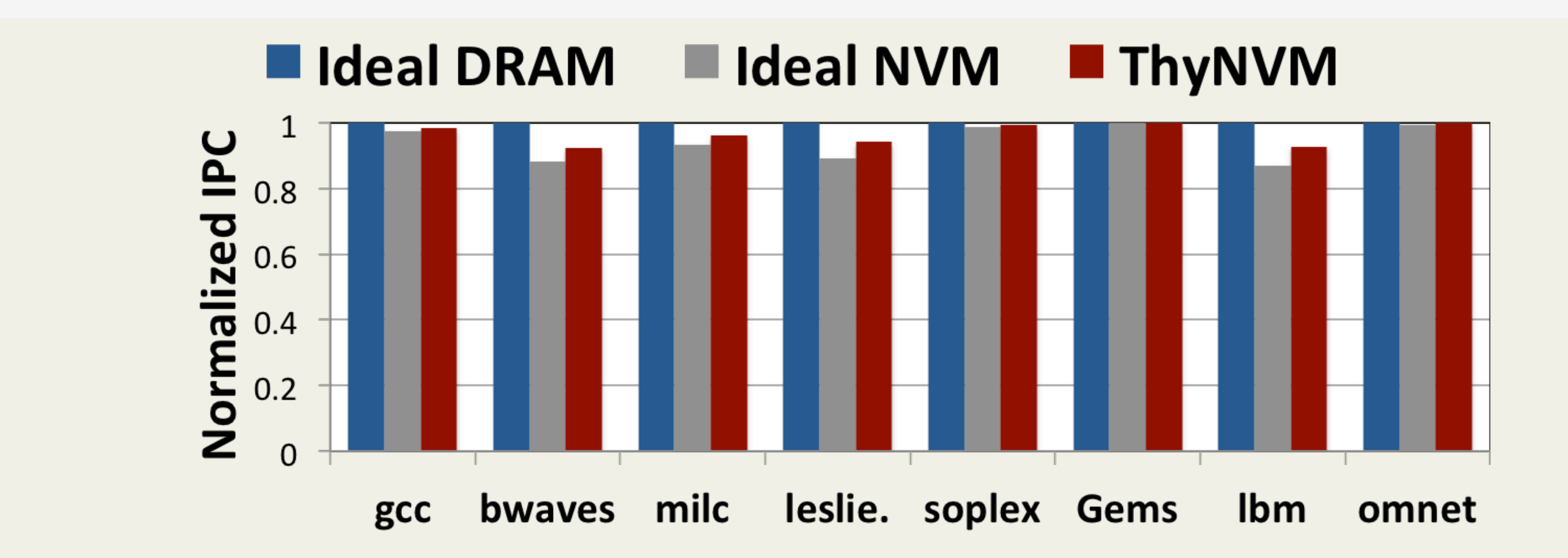

RANDOM

SEQUENTIAL

Can spend 35-45% of the execution on checkpointing

Stalls the application for a negligible time

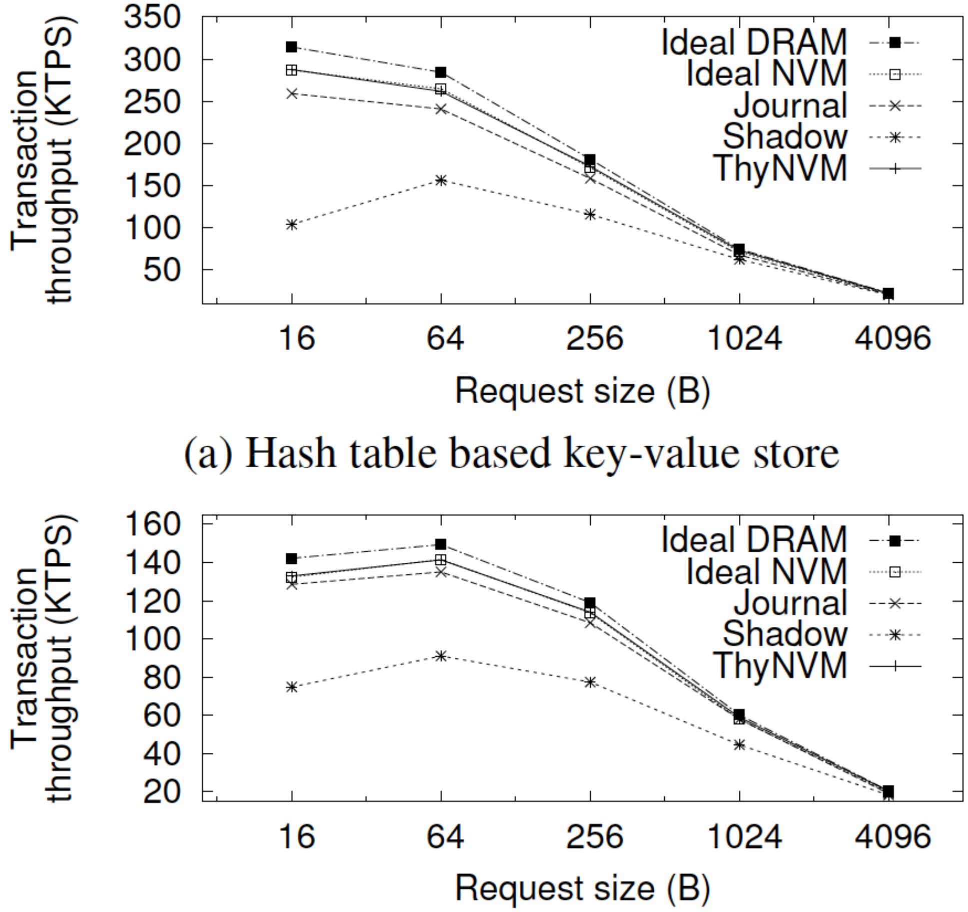
PERFORMANCE OF LEGACY CODE



Within -4.9%/+2.7% of an idealized DRAM/NVM system

Provides consistency without significant performance overhead

KEY-VALUE STORE TX THROUGHPUT



(a) Hash table based key-value store

(b) Red-black tree based key-value store

Figure 9: Transaction throughput for two key-value stores: (a) hash table based, (b) red-black tree based.

Storage throughput close to Ideal DRAM

OUTLINE

Crash Consistency Problem

Current Solutions

ThyNVM

Evaluation

Conclusion

ThyNVM

A new **hardware-based** *checkpointing mechanism*, **with no programming effort**

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

Can enable widespread *adoption* of persistent memory

Source Code and More Available at
http://persper.com/thynvm

ThyNVM

Enabling Software-transparent Crash Consistency In Persistent Memory Systems

Our Other FMS 2016 Talks

- "A Large-Scale Study of Flash Memory Errors in the Field"
  - Onur Mutlu (ETH Zurich & CMU) August 10 @ 3:50pm
  - Study of flash-based SSD errors in Facebook data centers over the course of 4 years
  - First large-scale field study of flash memory reliability
  - Forum F-22: SSD Testing (Testing Track)
- Practical Threshold Voltage Distribution Modeling
  - Yixin Luo (CMU PhD Student) August 10 @ 4:20pm
  - Forum E-22: Controllers and Flash Technology
- "WARM: Improving NAND Flash Memory Lifetime with Write-hotness Aware Retention Management"
  - Saugata Ghose (CMU Researcher) August 10 @ 5:45pm
  - Forum C-22: SSD Concepts (SSDs Track)

Referenced Papers and Talks

- All are available at
  http://users.ece.cmu.edu/~omutlu/projects.htm
  http://users.ece.cmu.edu/~omutlu/talks.htm

- And, many other previous works on
  - NVM & Persistent Memory
  - DRAM
  - Hybrid memories
  - NAND flash memory

Thank you.

Feel free to email me with any questions & feedback

omutlu@ethz.ch
http://users.ece.cmu.edu/~omutlu/

ThyNVM

Software-Transparent Crash Consistency for Persistent Memory

Onur Mutlu
omutlu@ethz.ch

(joint work with Jinglei Ren, Jishen Zhao, Samira Khan, Jongmoo Choi, Yongwei Wu)

August 8, 2016

Flash Memory Summit 2016, Santa Clara, CA

References to Papers and Talks

Challenges and Opportunities in Memory

- Onur Mutlu,
  "Rethinking Memory System Design"
  Keynote talk at
  2016 ACM SIGPLAN International Symposium on Memory Management (***ISMM***), Santa Barbara, CA, USA, June 2016.
  [Slides (pptx) (pdf)]
  [Abstract]

- Onur Mutlu and Lavanya Subramanian,
  "Research Problems and Opportunities in Memory Systems"
  Invited Article in Supercomputing Frontiers and Innovations (***SUPERFRI***), 2015.

Phase Change Memory As DRAM Replacement

- Benjamin C. Lee, Engin Ipek, Onur Mutlu, and Doug Burger, **"Architecting Phase Change Memory as a Scalable DRAM Alternative"**
  Proceedings of the
  *36th International Symposium on Computer Architecture (**ISCA**)*, pages 2-13, Austin, TX, June 2009. Slides (pdf)

- Benjamin C. Lee, Ping Zhou, Jun Yang, Youtao Zhang, Bo Zhao, Engin Ipek, Onur Mutlu, and Doug Burger,
  "Phase Change Technology and the Future of Main Memory"
  *IEEE Micro, Special Issue: Micro's Top Picks from 2009 Computer Architecture Conferences (**MICRO TOP PICKS**)*, Vol. 30, No. 1, pages 60-70, January/February 2010.

STT-MRAM As DRAM Replacement

- Emre Kultursay, Mahmut Kandemir, Anand Sivasubramaniam, and Onur Mutlu, **"Evaluating STT-RAM as an Energy-Efficient Main Memory Alternative"** *Proceedings of the 2013 IEEE International Symposium on Performance Analysis of Systems and Software (**ISPASS**)*, Austin, TX, April 2013. Slides (pptx) (pdf)

Taking Advantage of Persistence in Memory

- 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)

- Jinglei Ren, Jishen Zhao, Samira Khan, Jongmoo Choi, Yongwei Wu, and Onur Mutlu,
  "ThyNVM: Enabling Software-Transparent Crash Consistency in Persistent Memory Systems"
  Proceedings of the 48th International Symposium on Microarchitecture (***MICRO***), Waikiki, Hawaii, USA, December 2015.
  [Slides (pptx) (pdf)] [Lightning Session Slides (pptx) (pdf)] [Poster (pptx) (pdf)]
  [Source Code]

Hybrid DRAM + NVM Systems (I)

- 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) ***Best paper award (in Computer Systems and Applications track).***

- 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.

Hybrid DRAM + NVM Systems (II)

- Dongwoo Kang, Seungjae Baek, Jongmoo Choi, Donghee Lee, Sam H. Noh, and Onur Mutlu,
  "Amnesic Cache Management for Non-Volatile Memory"
  Proceedings of the
  31st International Conference on Massive Storage Systems and Technologies (***MSST***), Santa Clara, CA, June 2015.
  [Slides (pdf)]

NVM Design and Architecture

- HanBin Yoon, Justin Meza, Naveen Muralimanohar, Norman P. Jouppi, and Onur Mutlu,
  "Efficient Data Mapping and Buffering Techniques for Multi-Level Cell Phase-Change Memories"
  *ACM Transactions on Architecture and Code Optimization (**TACO**)*, Vol. 11, No. 4, December 2014. [Slides (ppt) (pdf)]
  Presented at the 10th HiPEAC Conference, Amsterdam, Netherlands, January 2015.
  [Slides (ppt) (pdf)]

- Justin Meza, Jing Li, and Onur Mutlu,
  "Evaluating Row Buffer Locality in Future Non-Volatile Main Memories"
  SAFARI Technical Report, TR-SAFARI-2012-002, Carnegie Mellon University, December 2012.

Our FMS Talks and Posters

- Onur Mutlu, *ThyNVM: Software-Transparent Crash Consistency for Persistent Memory*, FMS 2016.
- Onur Mutlu, *Large-Scale Study of In-the-Field Flash Failures*, FMS 2016.
- Yixin Luo, *Practical Threshold Voltage Distribution Modeling*, FMS 2016.
- Saugata Ghose, *Write-hotness Aware Retention Management*, FMS 2016.
- Onur Mutlu, *Read Disturb Errors in MLC NAND Flash Memory*, FMS 2015.
- Yixin Luo, *Data Retention in MLC NAND Flash Memory*, FMS 2015.
- Onur Mutlu, *Error Analysis and Management for MLC NAND Flash Memory*, FMS 2014.

- FMS 2016 posters:
  - WARM: Improving NAND Flash Memory Lifetime with Write-hotness Aware Retention Management
  - Read Disturb Errors in MLC NAND Flash Memory
  - Data Retention in MLC NAND Flash Memory

Our Flash Memory Works (I)

1. Retention noise study and management

1) 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**, ICCD 2012.
2) Yu Cai, Yixin Luo, Erich F. Haratsch, Ken Mai, and Onur Mutlu, **Data Retention in MLC NAND Flash Memory: Characterization, Optimization and Recovery**, HPCA 2015.
3) Yixin Luo, Yu Cai, Saugata Ghose, Jongmoo Choi, and Onur Mutlu, **WARM: Improving NAND Flash Memory Lifetime with Write-hotness Aware Retention Management**, MSST 2015.

2. Flash-based SSD prototyping and testing platform

4) Yu Cai, Erich F. Haratsh, Mark McCartney, Ken Mai, **FPGA-based solid-state drive prototyping platform**, FCCM 2011.

Our Flash Memory Works (II)

3. Overall flash error analysis

5) Yu Cai, Erich F. Haratsch, Onur Mutlu, and Ken Mai, **Error Patterns in MLC NAND Flash Memory: Measurement, Characterization, and Analysis**, DATE 2012.

6) 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**, ITJ 2013.

4. Program and erase noise study

7) Yu Cai, Erich F. Haratsch, Onur Mutlu, and Ken Mai, **Threshold Voltage Distribution in MLC NAND Flash Memory: Characterization, Analysis and Modeling**, DATE 2013.

Our Flash Memory Works (III)

5. Cell-to-cell interference characterization and tolerance

8) Yu Cai, Onur Mutlu, Erich F. Haratsch, and Ken Mai, **Program Interference in MLC NAND Flash Memory: Characterization, Modeling, and Mitigation**, ICCD 2013.

9) Yu Cai, Gulay Yalcin, Onur Mutlu, Erich F. Haratsch, Osman Unsal, Adrian Cristal, and Ken Mai, **Neighbor-Cell Assisted Error Correction for MLC NAND Flash Memories**, SIGMETRICS 2014.

6. Read disturb noise study

10) Yu Cai, Yixin Luo, Saugata Ghose, Erich F. Haratsch, Ken Mai, and Onur Mutlu, **Read Disturb Errors in MLC NAND Flash Memory: Characterization and Mitigation**, DSN 2015.

Our Flash Memory Works (IV)

7. Flash errors in the field

11) Justin Meza, Qiang Wu, Sanjeev Kumar, and Onur Mutlu, **A Large-Scale Study of Flash Memory Errors in the Field**, SIGMETRICS 2015.

8. Persistent memory

12) Jinglei Ren, Jishen Zhao, Samira Khan, Jongmoo Choi, Yongwei Wu, and Onur Mutlu, **ThyNVM: Enabling Software-Transparent Crash Consistency in Persistent Memory Systems**, MICRO 2015.

Referenced Papers and Talks

- All are available at
  http://users.ece.cmu.edu/~omutlu/projects.htm
  http://users.ece.cmu.edu/~omutlu/talks.htm

- And, many other previous works on NAND flash memory errors and management

Related Videos and Course Materials

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

Additional Slides on Persistent Memory and NVM

Phase Change Memory: Pros and Cons

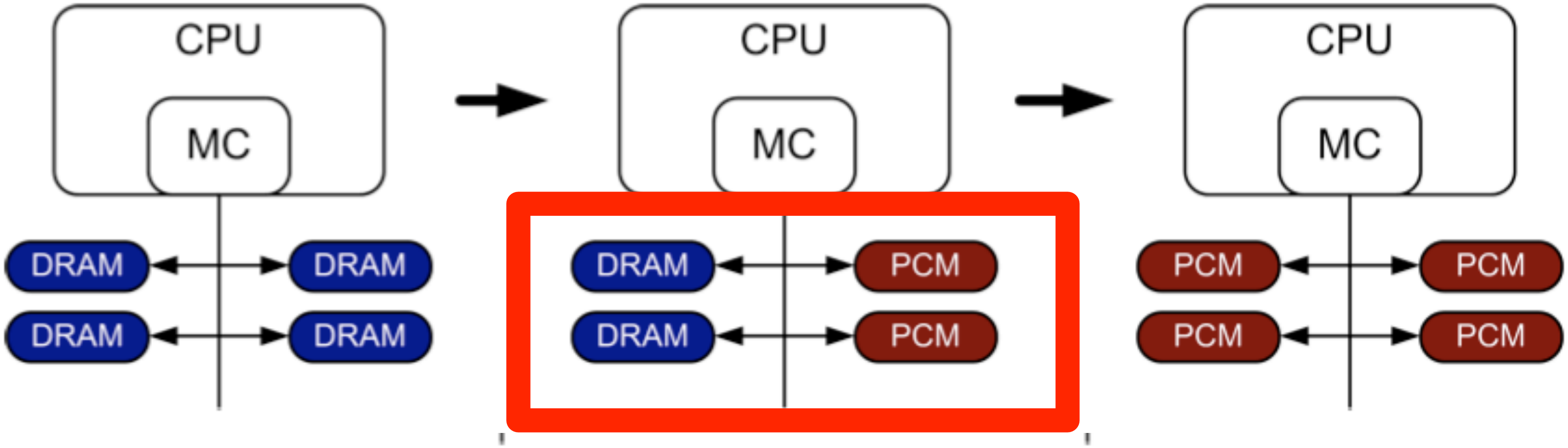
- Pros over DRAM
  - Better technology scaling (capacity and cost)
  - Non volatility
  - Low idle power (no refresh)

- Cons
  - Higher latencies: ~4-15x DRAM (especially write)
  - Higher active energy: ~2-50x DRAM (especially write)
  - Lower endurance (a cell dies after ~10^8 writes)
  - Reliability issues (resistance drift)

- Challenges in enabling PCM as DRAM replacement/helper:
  - Mitigate PCM shortcomings
  - Find the right way to place PCM in the system

PCM-based Main Memory (I)

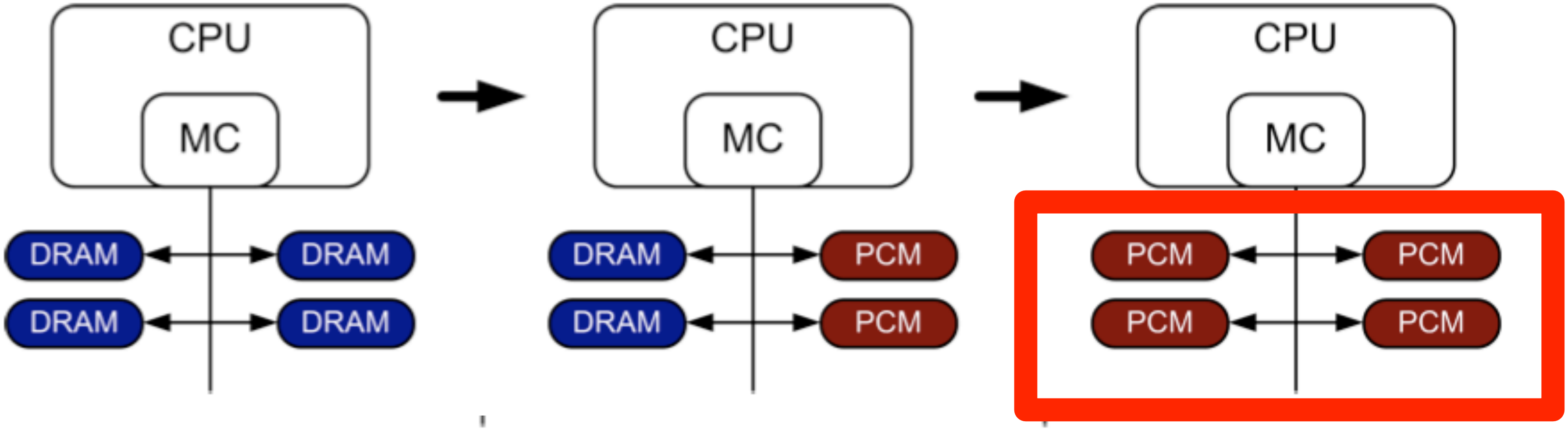
- How should PCM-based (main) memory be organized?



- Hybrid PCM+DRAM [Qureshi+ ISCA'09, Dhiman+ DAC'09]:
  - How to partition/migrate data between PCM and DRAM

PCM-based Main Memory (II)

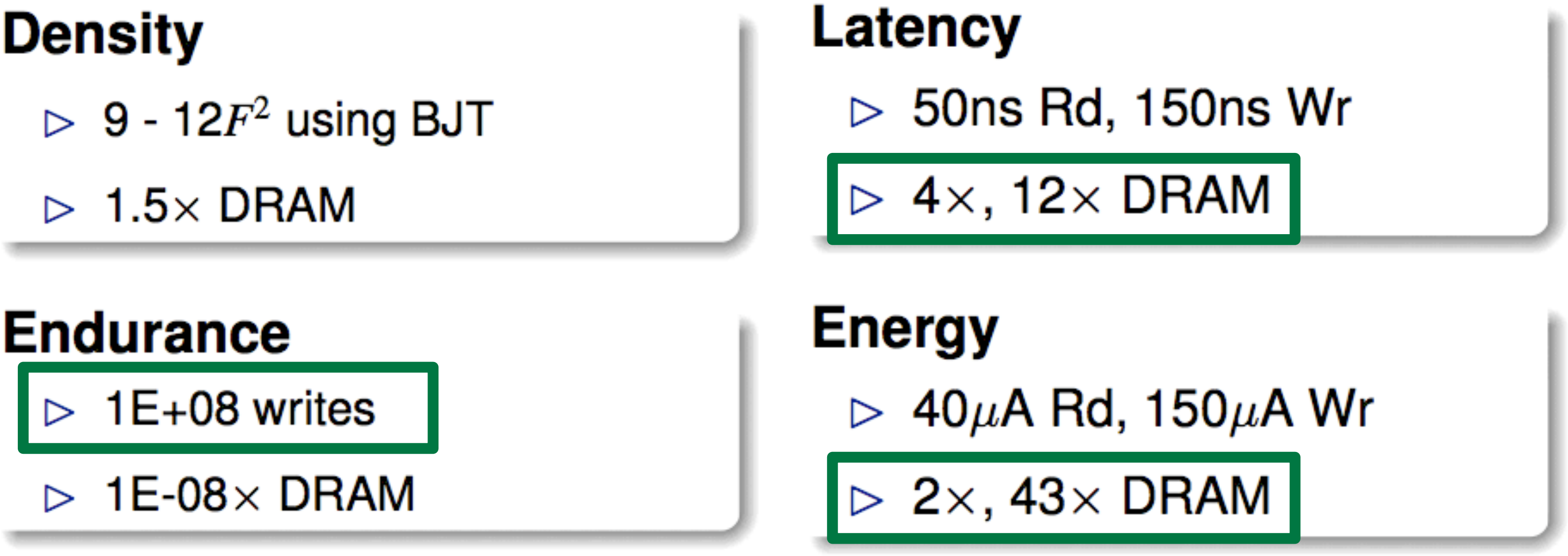
- How should PCM-based (main) memory be organized?



- Pure PCM main memory [Lee et al., ISCA'09, Top Picks'10]:
    - How to redesign entire hierarchy (and cores) to overcome PCM shortcomings

An Initial Study: Replace DRAM with PCM

- Lee, Ipek, Mutlu, Burger, "Architecting Phase Change Memory as a Scalable DRAM Alternative," ISCA 2009.
  - Surveyed prototypes from 2003-2008 (e.g. IEDM, VLSI, ISSCC)
  - Derived "average" PCM parameters for F=90nm



Density

▷ 9 - 12F^2 using BJT

▷ 1.5× DRAM

Latency

▷ 50ns Rd, 150ns Wr

▷ 4×, 12× DRAM

Endurance

▷ 1E+08 writes

▷ 1E-08× DRAM

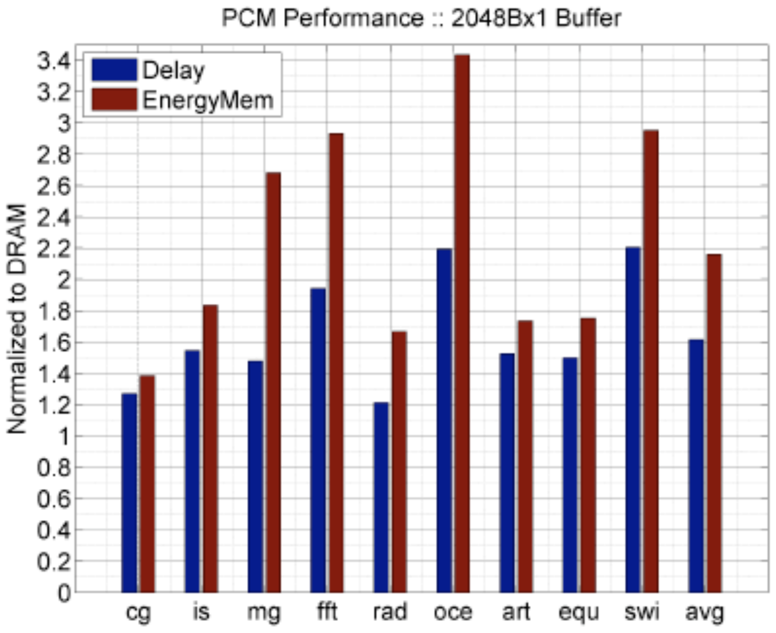
Energy

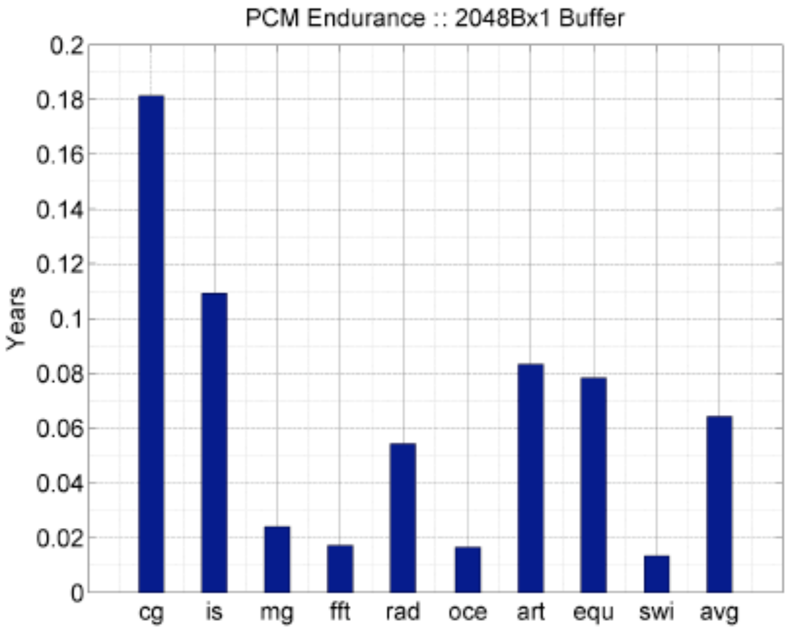
▷ 40μA Rd, 150μA Wr

▷ 2×, 43× DRAM

Results: Naïve Replacement of DRAM with PCM

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

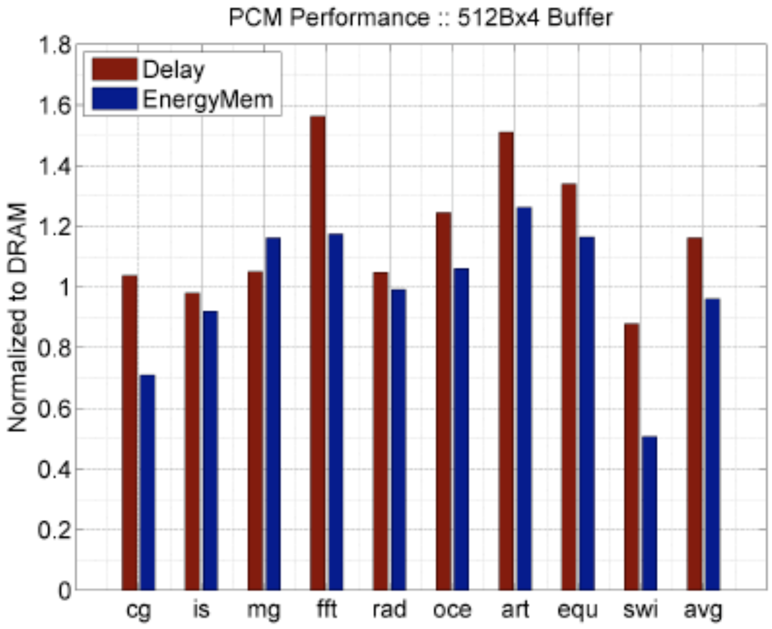


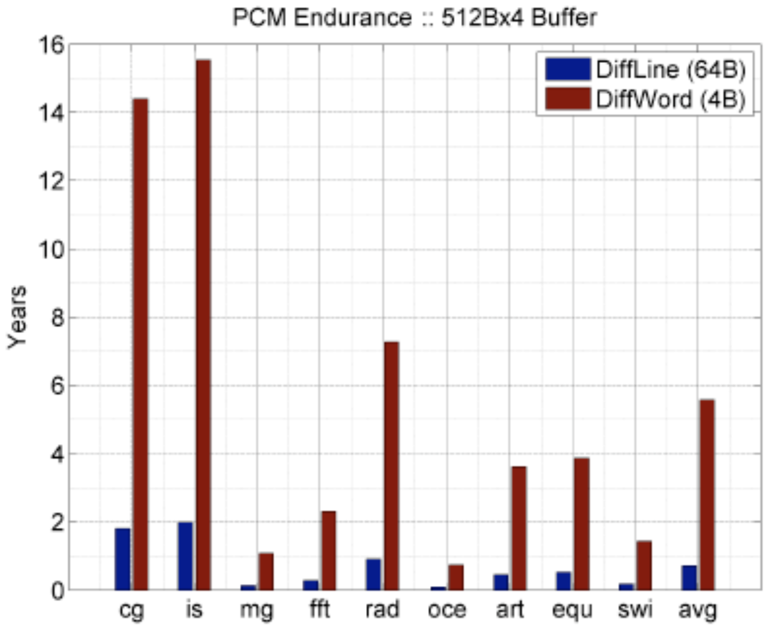


- Lee, Ipek, Mutlu, Burger, "Architecting Phase Change Memory as a Scalable DRAM Alternative," ISCA 2009.

Results: Architected PCM as Main Memory

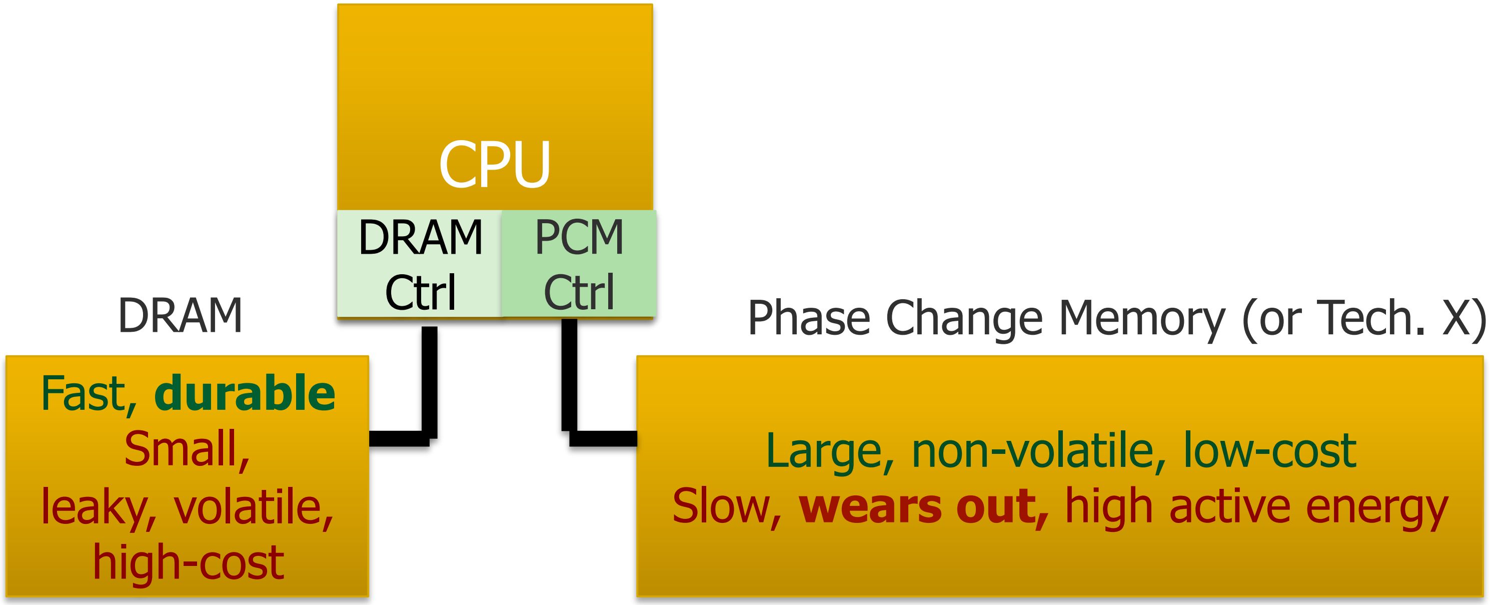
- 1.2x delay, 1.0x energy, 5.6-year average lifetime
- Scaling improves energy, endurance, density





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

A More Viable Approach: Hybrid Memory Systems


CPU
DRAM Ctrl
PCM Ctrl
DRAM
Fast, **durable**
Small,
leaky, volatile,
high-cost
Phase Change Memory (or Tech. X)
Large, non-volatile, low-cost
Slow, **wears out,** high active energy

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

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

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

Data Placement Between DRAM and PCM

- Idea: Characterize data access patterns and guide data placement in hybrid memory

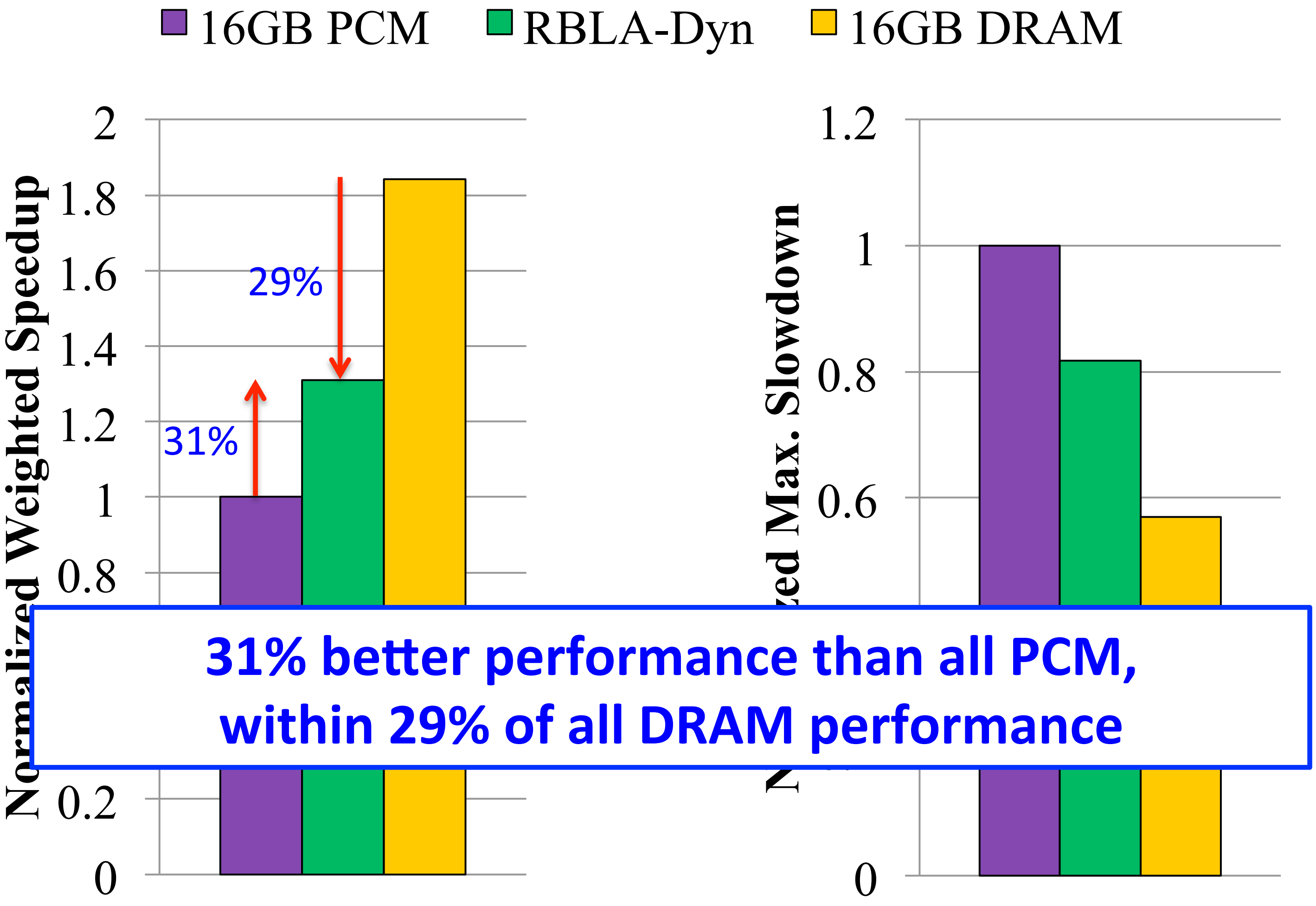
- Streaming accesses: As fast in PCM as in DRAM

- Random accesses: Much faster in DRAM

- Idea: Place random access data with some reuse in DRAM; streaming data in PCM

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

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



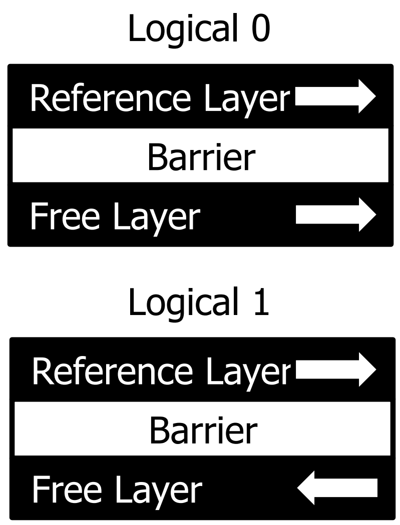
31% better performance than all PCM, within 29% of all DRAM performance

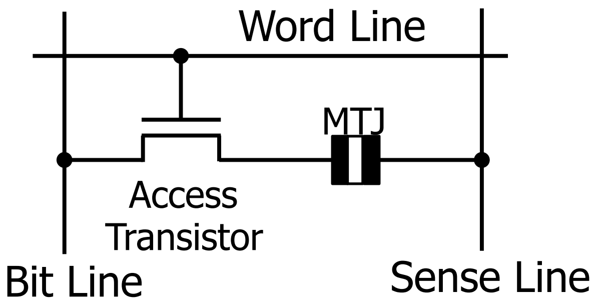
Yoon+, "Row Buffer Locality-Aware Data Placement in Hybrid Memories," ICCD 2012 Best Paper Award.

STT-MRAM as Main Memory

- Magnetic Tunnel Junction (MTJ) device
  - Reference layer: Fixed magnetic orientation
  - Free layer: Parallel or anti-parallel
- Magnetic orientation of the free layer determines logical state of device
  - High vs. low resistance
- Write: Push large current through MTJ to change orientation of free layer
- Read: Sense current flow

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





STT-MRAM: Pros and Cons

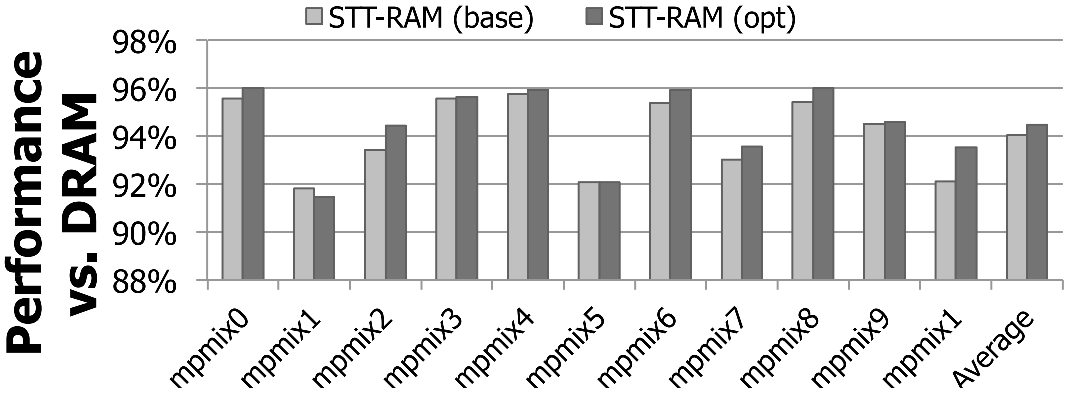
- Pros over DRAM
  - Better technology scaling
  - Non volatility
  - Low idle power (no refresh)

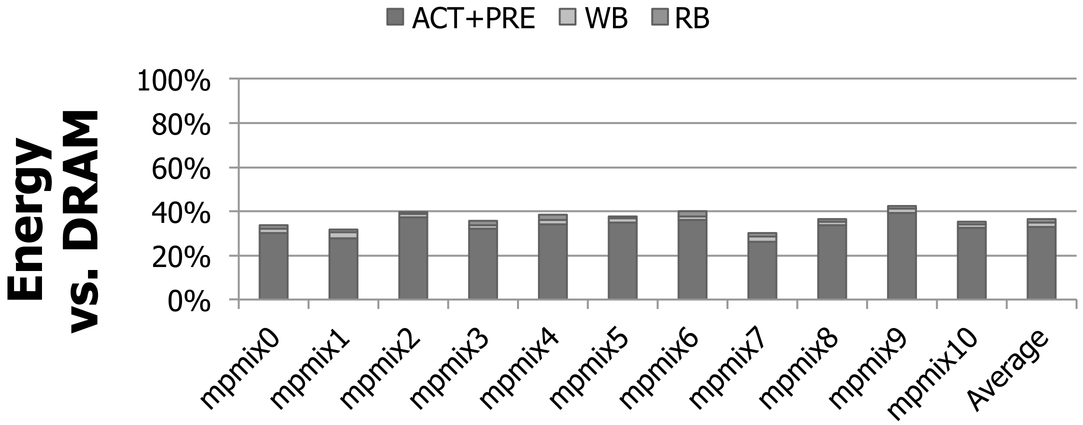
- Cons
  - Higher write latency
  - Higher write energy
  - Reliability?

- Another level of freedom
  - Can trade off non-volatility for lower write latency/energy (by reducing the size of the MTJ)

Architected STT-MRAM as Main Memory

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





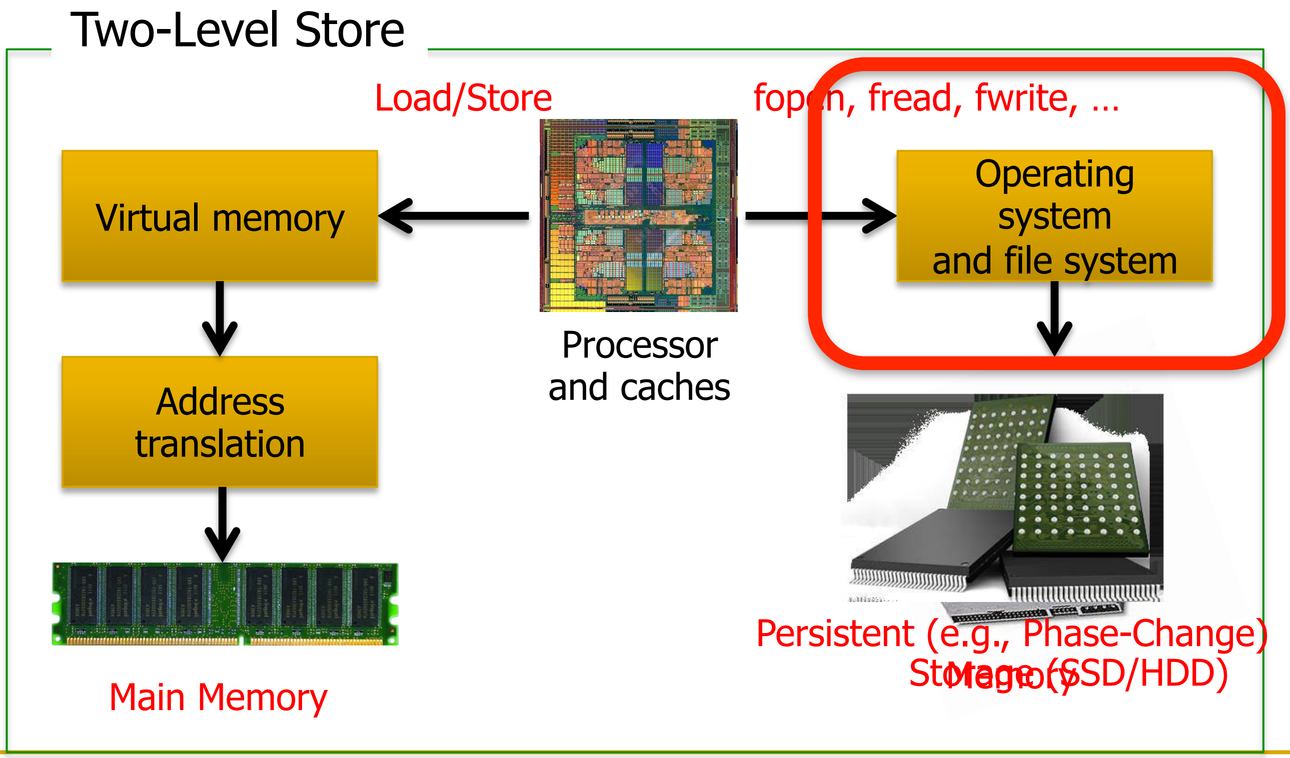
Kultursay+, "Evaluating STT-RAM as an Energy-Efficient Main Memory Alternative," ISPASS 2013.

Other Opportunities with Emerging Technologies

- Merging of memory and storage
  - e.g., a single interface to manage all data
- New applications
  - e.g., ultra-fast checkpoint and restore
- More robust system design
  - e.g., reducing data loss
- Processing tightly-coupled with memory
  - e.g., enabling efficient search and filtering

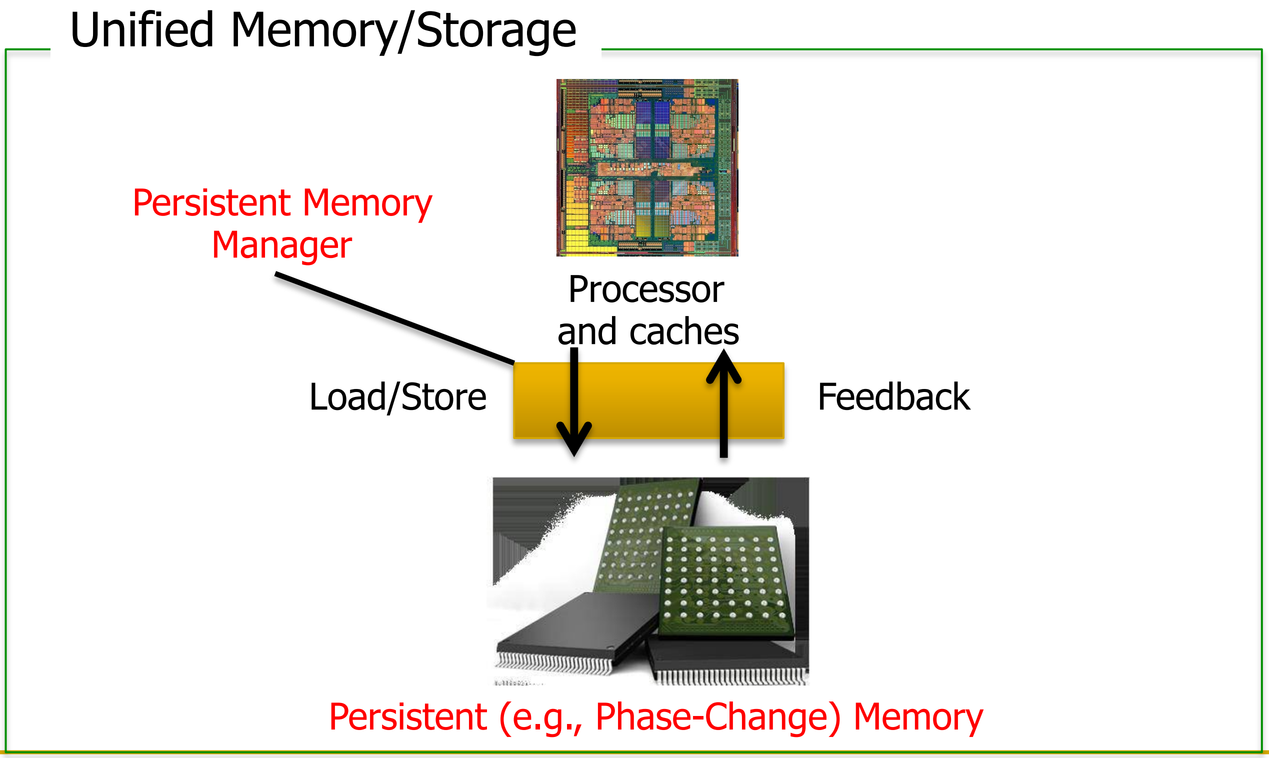
Coordinated Memory and Storage with NVM (I)

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



Coordinated Memory and Storage with NVM (II)

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


Unified Memory/Storage
Persistent Memory Manager
Processor and caches
Load/Store
Feedback
Persistent (e.g., Phase-Change) Memory

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

The Persistent Memory Manager (PMM)

- Exposes a load/store interface to access persistent data
  - Applications can directly access persistent memory → no conversion, translation, location overhead for persistent data

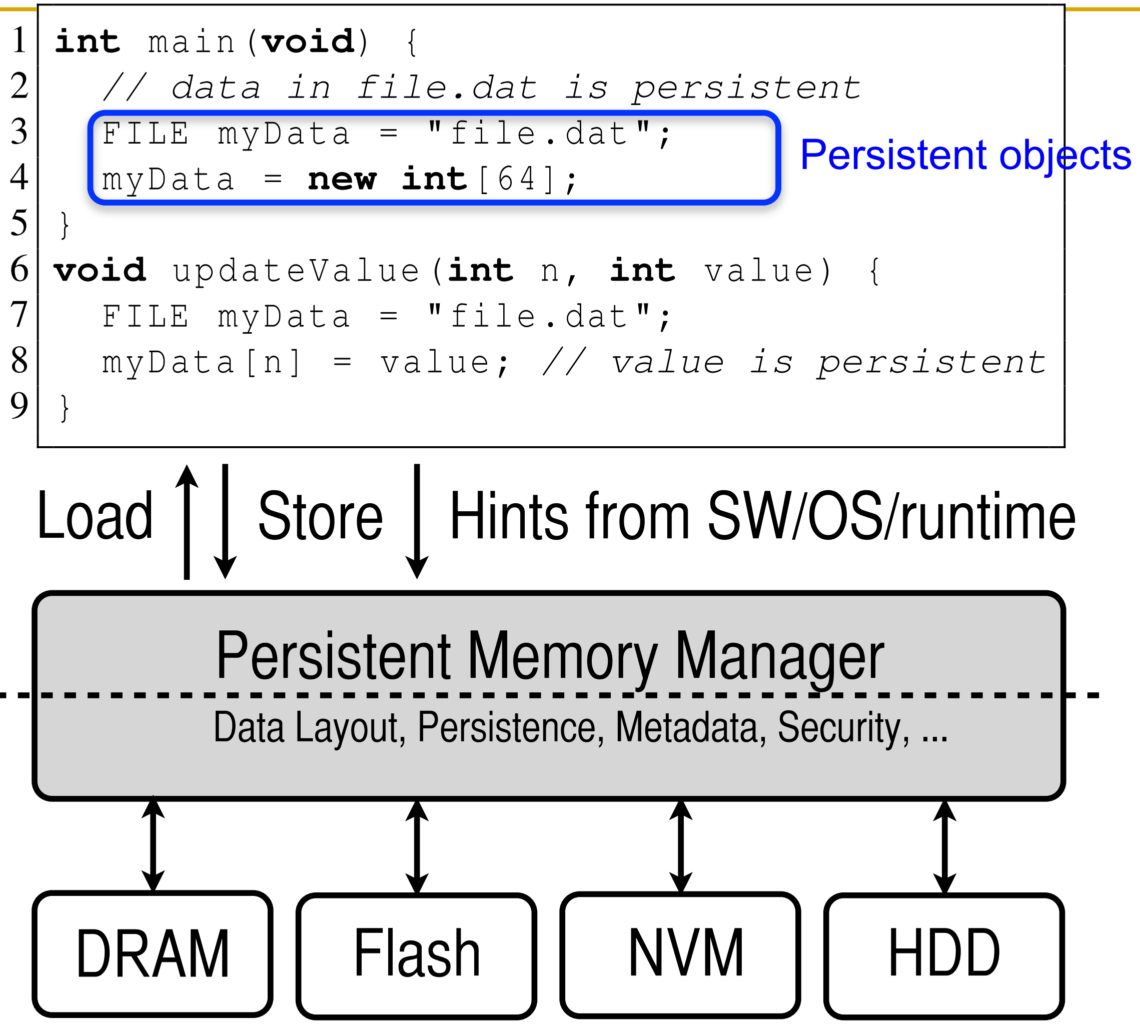
- Manages data placement, location, persistence, security
  - To get the best of multiple forms of storage

- Manages metadata storage and retrieval
  - This can lead to overheads that need to be managed

- Exposes hooks and interfaces for system software
  - To enable better data placement and management decisions

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

The Persistent Memory Manager (PMM)



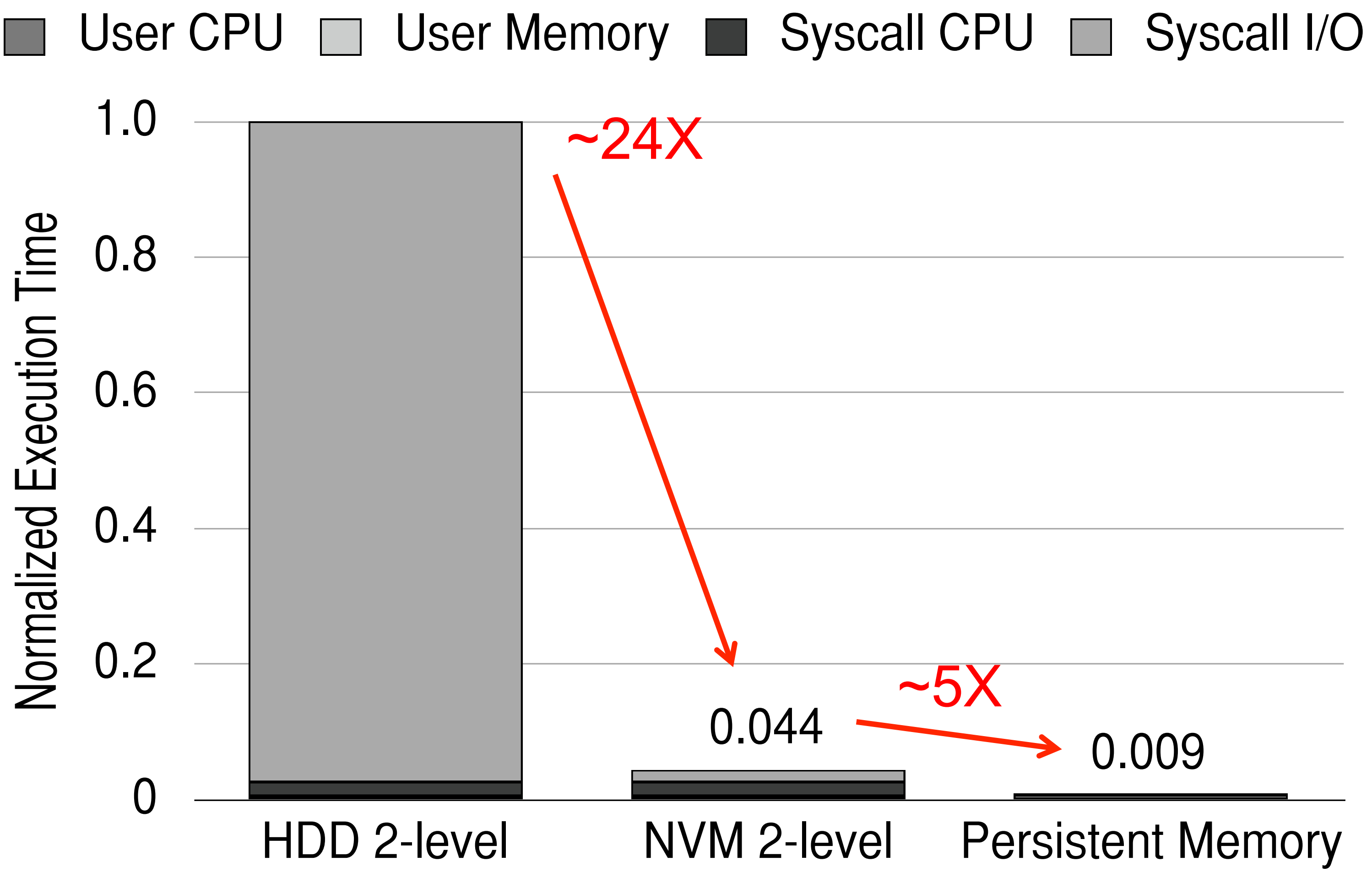
```
int main(void) {
  // data in file.dat is persistent
  FILE myData = "file.dat";
  myData = new int[64];
}
void updateValue(int n, int value) {
  FILE myData = "file.dat";
  myData[n] = value; // value is persistent
}
```

Persistent objects

Load
Store
Hints from SW/OS/runtime
Software
Hardware
Persistent Memory Manager
Data Layout, Persistence, Metadata, Security, ...
DRAM
Flash
NVM
HDD

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

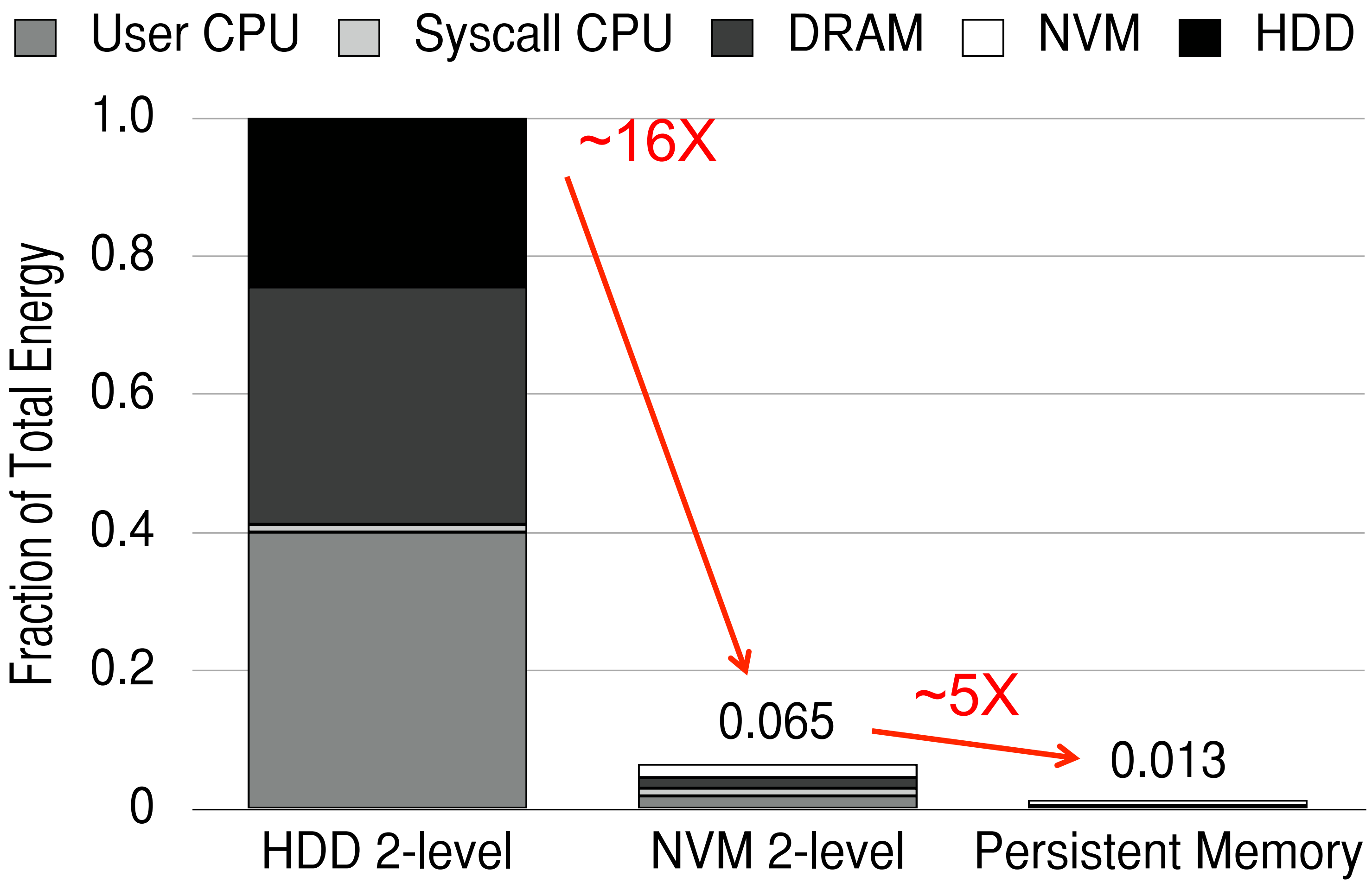
Performance Benefits of a Single-Level Store



SAFARI

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

Energy Benefits of a Single-Level Store



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