Computer Architecture

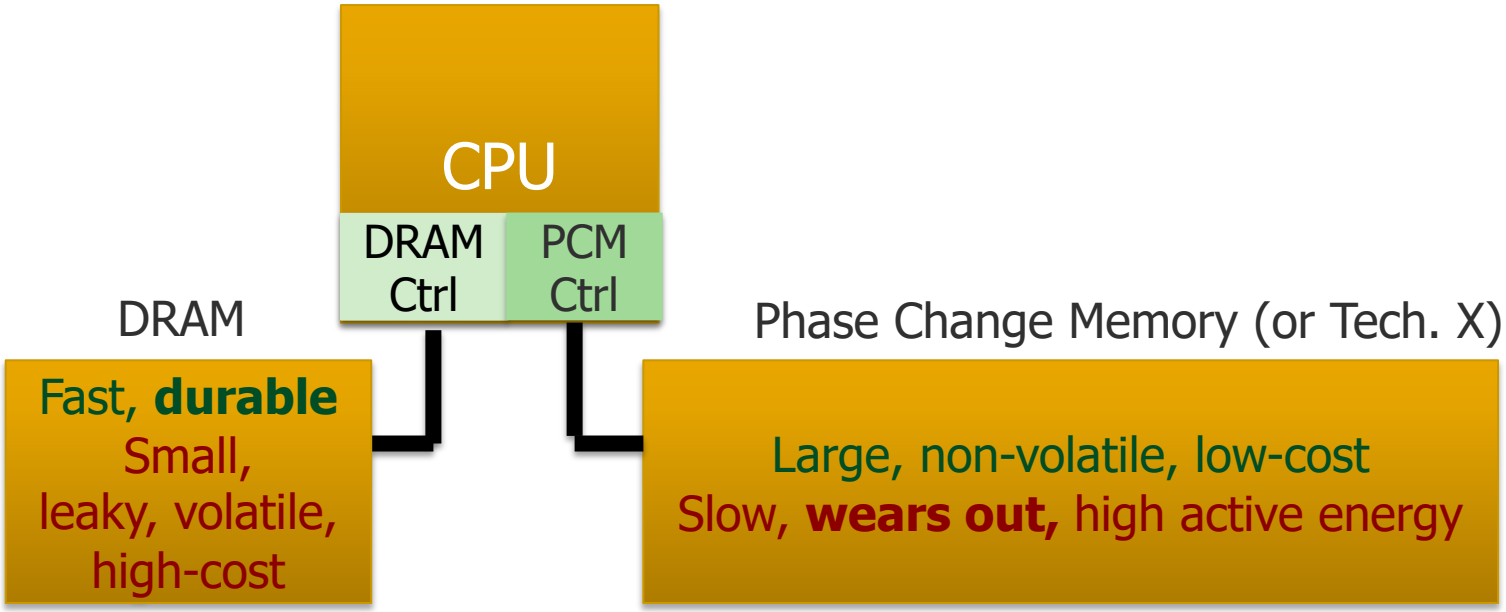
Lecture 19a: Emerging Memory Technologies II

Prof. Onur Mutlu
ETH Zürich

Fall 2019

28 November 2019

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.

Challenge and Opportunity

Providing the Best of Multiple Metrics

with

Multiple Memory Technologies

Hybrid Memory Systems: Issues

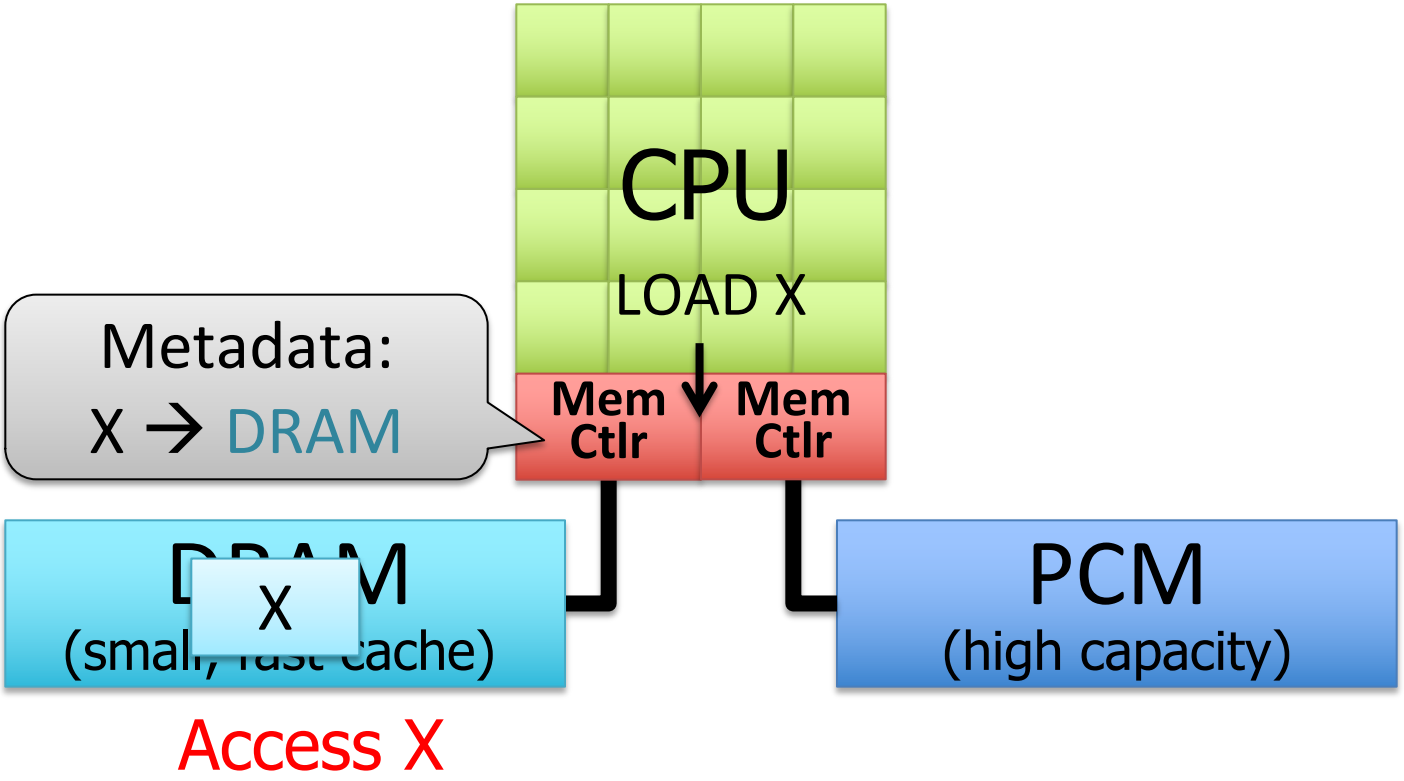
- Cache vs. Main Memory
- Granularity of Data Move/Manage-ment: Fine or Coarse
- Hardware vs. Software vs. HW/SW Cooperative
- When to migrate data?
- How to design a scalable and efficient large cache?
- …

Another Challenge

Designing Effective Large (DRAM) Caches

One 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

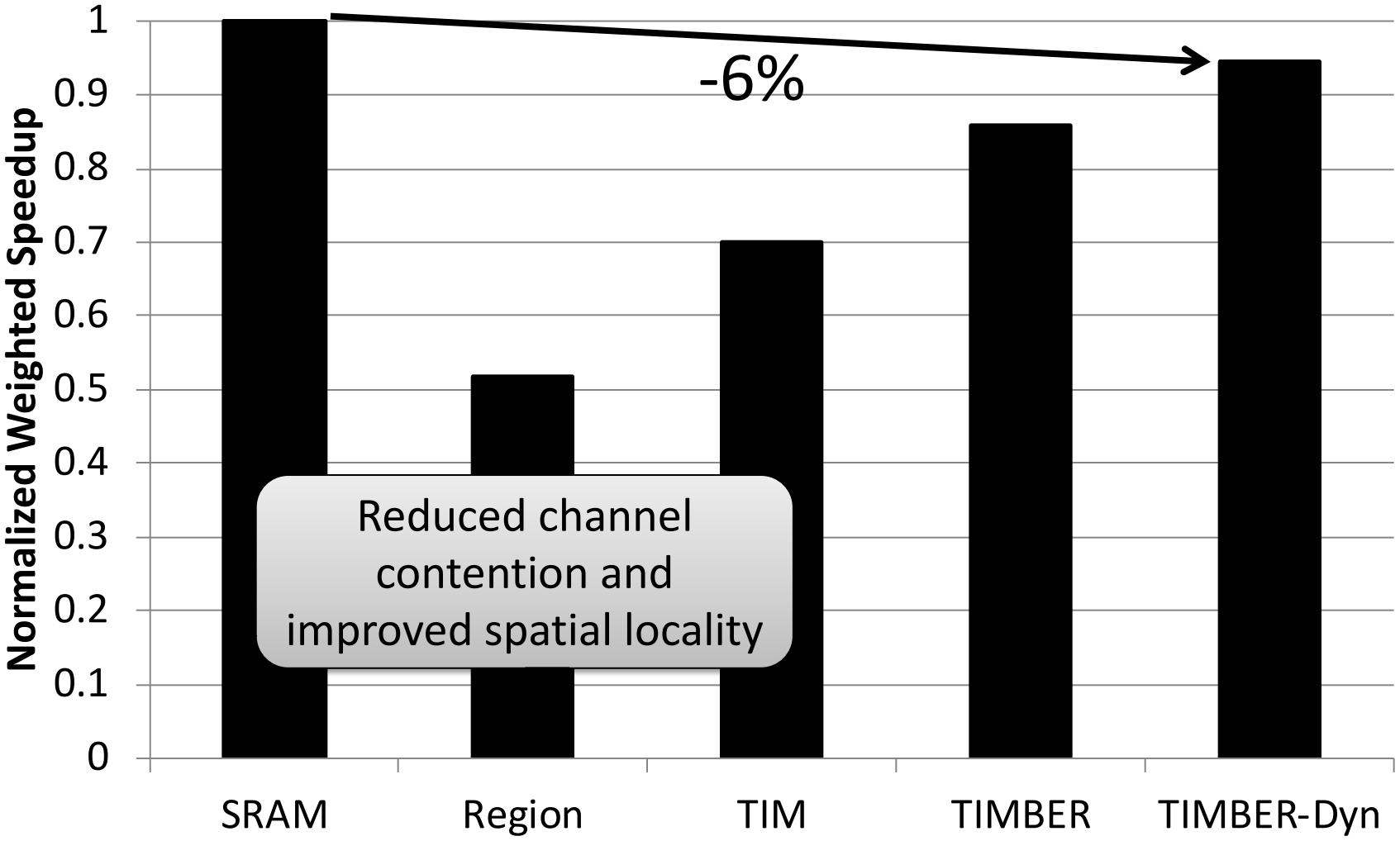
- Idea 2: Cache in on-chip SRAM frequently-accessed metadata
  - Cache only a small amount to keep SRAM size small

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

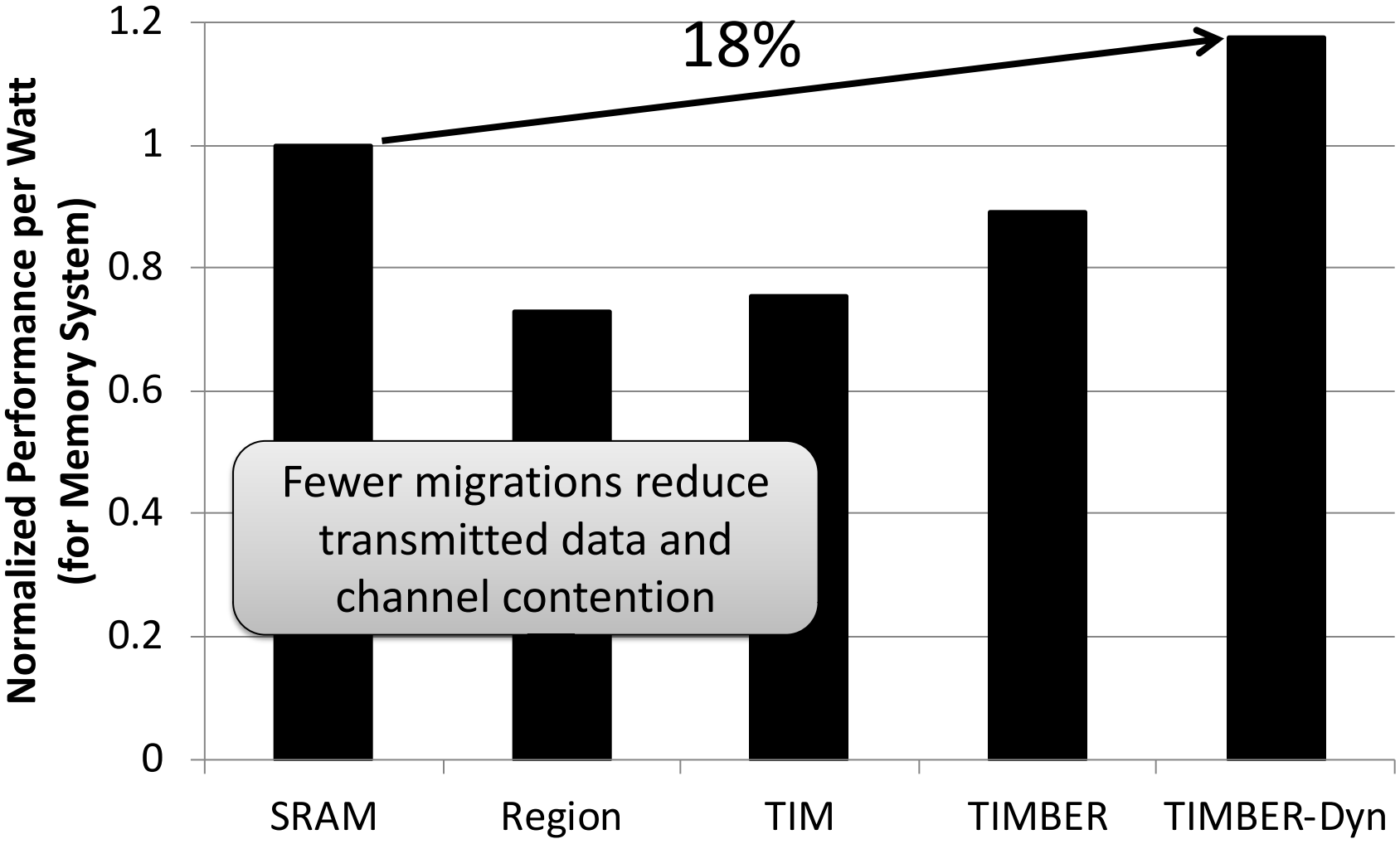
- Idea 3: Simple dynamic caching granularity policy
  - Cost-benefit analysis to determine best DRAM cache block size
  - Group main memory into sets of rows
  - Different sampled row sets follow different fixed caching granularities
  - The rest of main memory follows the best granularity
    - Cost–benefit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

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.

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.

Enabling Efficient and Scalable Hybrid Memories Using Fine-Granularity DRAM Cache Management

Justin Meza* Jichuan Chang† HanBin Yoon* Onur Mutlu* Parthasarathy Ranganathan†
*Carnegie Mellon University †Hewlett-Packard Labs
{meza,hanbinyoon,onur}@cmu.edu {jichuan.chang,partha.ranganathan}@hp.com

DRAM Caches: Many Recent Options

Table 1: Summary of Operational Characteristics of Different State-of-the-Art DRAM Cache Designs – We assume perfect way prediction for Unison Cache. Latency is relative to the access time of the off-package DRAM (see Section 6 for baseline latencies). We use different colors to indicate the high (dark red), medium (white), and low (light green) overhead of a characteristic.

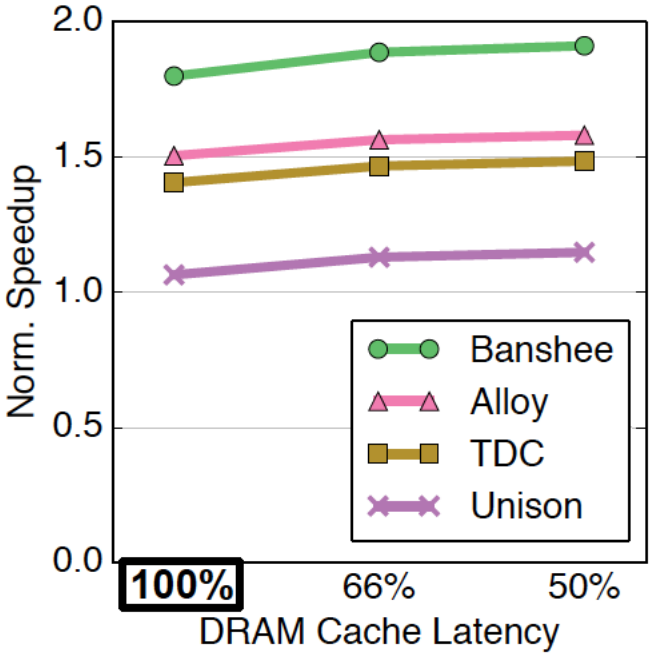
Scheme	DRAM Cache Hit	DRAM Cache Miss	Replacement Traffic	Replacement Decision	Large Page Caching
Unison [32]	In-package traffic: 128 B (data + tag read and update) Latency: ~1x	In-package traffic: 96 B (spec. data + tag read) Latency: ~2x	On every miss Footprint size [31]	Hardware managed, set-associative, LRU	Yes
Alloy [50]	In-package traffic: 96 B (data + tag read) Latency: ~1x	In-package traffic: 96 B (spec. data + tag read) Latency: ~2x	On some misses Cacheline size (64 B)	Hardware managed, direct-mapped, stochastic [20]	Yes
TDC [38]	In-package traffic: 64 B Latency: ~1x TLB coherence	In-package traffic: 0 B Latency: ~1x TLB coherence	On every miss Footprint size [28]	Hardware managed, fully-associative, FIFO	No
HMA [44]	In-package traffic: 64 B Latency: ~1x	In-package traffic: 0 B Latency: ~1x	Software managed, high replacement cost		Yes
Banshee (This work)	In-package traffic: 64 B Latency: ~1x	In-package traffic: 0 B Latency: ~1x	Only for hot pages Page size (4 KB)	Hardware managed, set-associative, frequency based	Yes

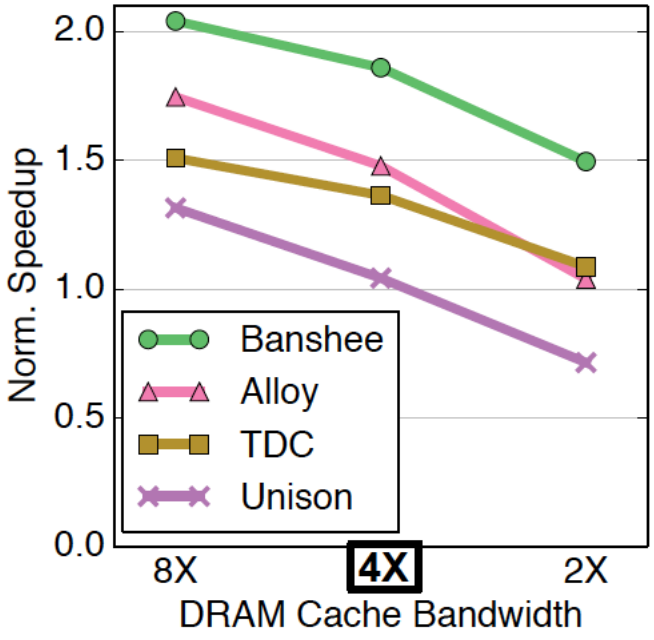
Yu+, "Banshee: Bandwidth-Efficient DRAM Caching via Software/Hardware Cooperation," MICRO 2017.

Banshee [MICRO 2017]

- Tracks presence in cache using TLB and Page Table
  - No tag store needed for DRAM cache
  - Enabled by a new lightweight lazy TLB coherence protocol

- New bandwidth-aware frequency-based replacement policy





More on Banshee

- Xiangyao Yu, Christopher J. Hughes, Nadathur Satish, Onur Mutlu, and Srinivas Devadas,
  "Banshee: Bandwidth-Efficient DRAM Caching via Software/Hardware Cooperation"
  *Proceedings of the 50th International Symposium on Microarchitecture (**MICRO**)*, Boston, MA, USA, October 2017.

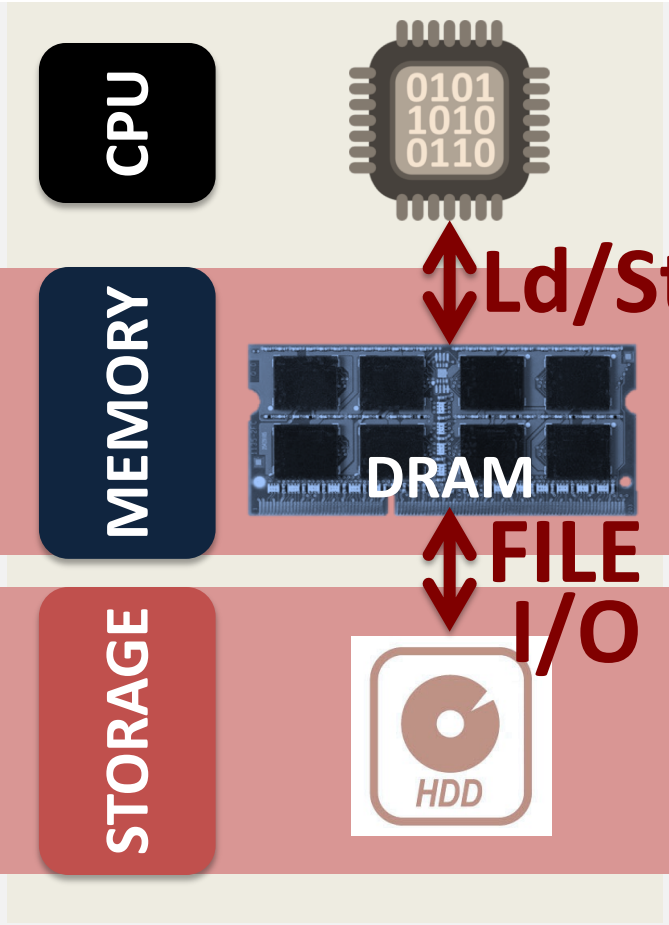
Banshee: Bandwidth-Efficient DRAM Caching via Software/Hardware Cooperation

Xiangyao Yu[1] Christopher J. Hughes[2] Nadathur Satish[2] Onur Mutlu[3] Srinivas Devadas[1]
[1]MIT [2]Intel Labs [3]ETH Zürich

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

TWO-LEVEL STORAGE MODEL



VOLATILE

FAST

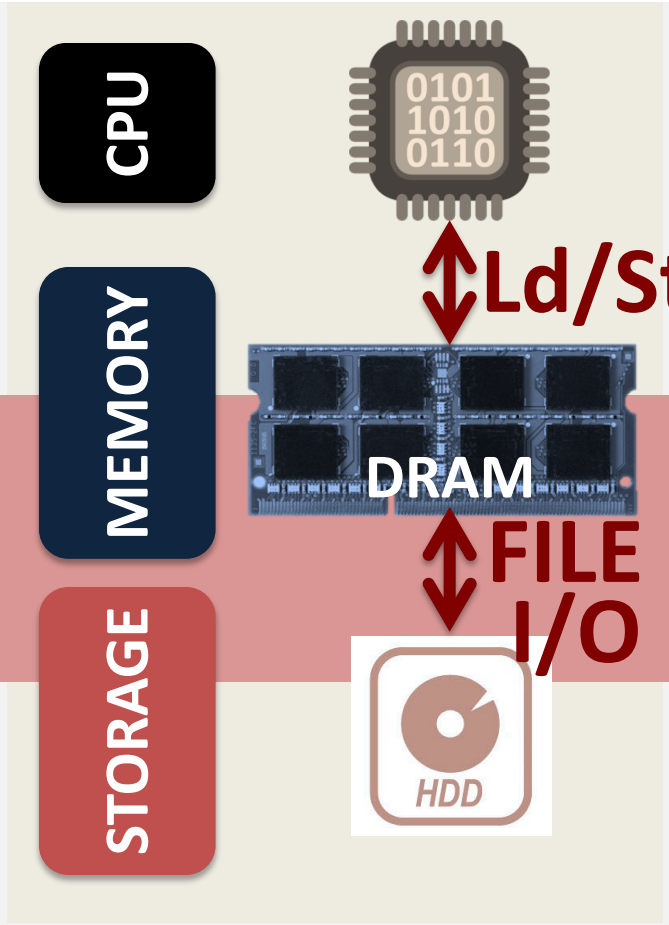
BYTE ADDR

NONVOLATILE

SLOW

BLOCK ADDR

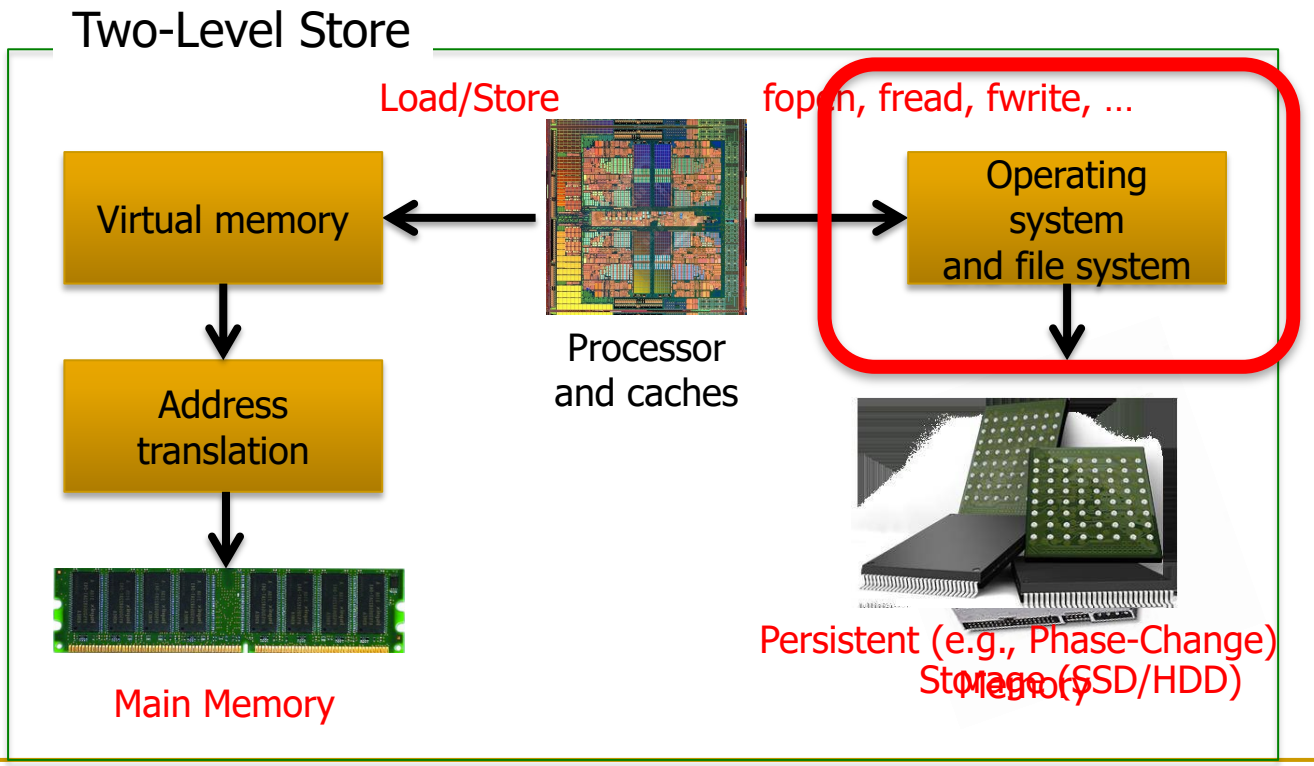
TWO-LEVEL STORAGE MODEL



Non-volatile memories combine characteristics of memory and storage

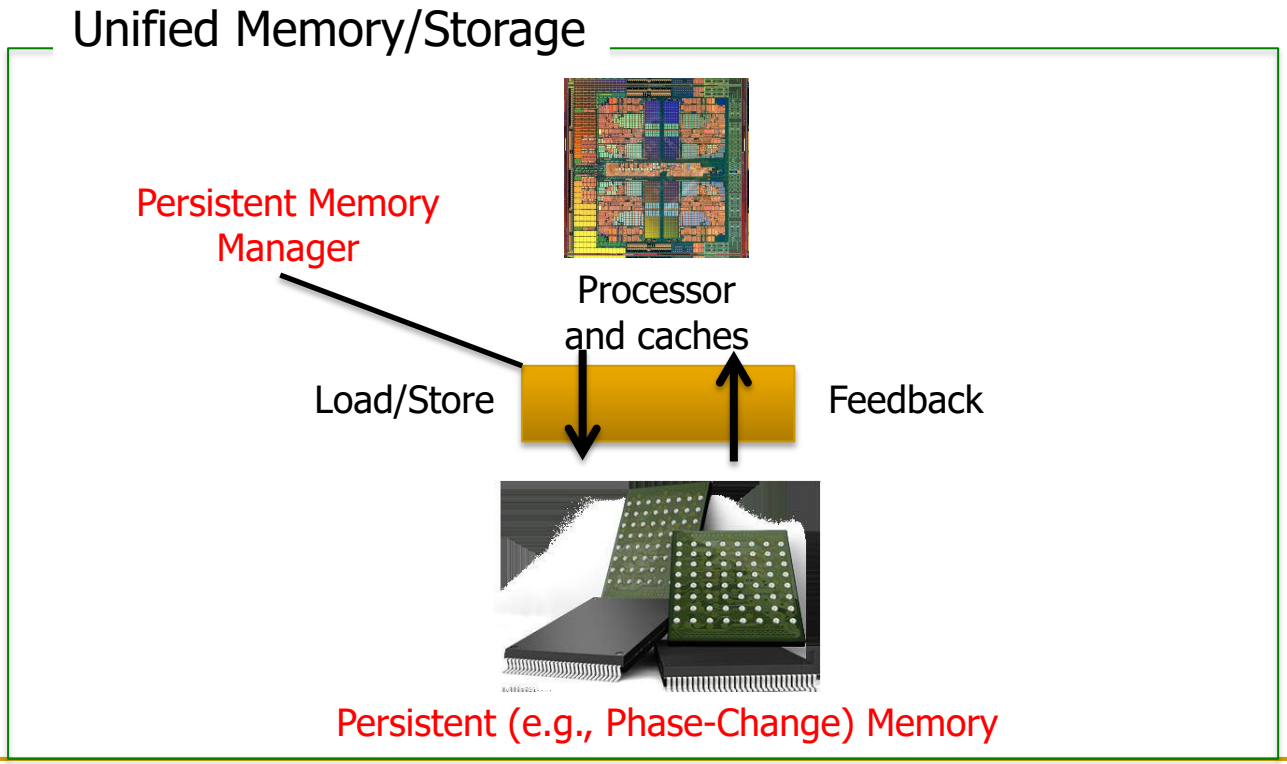
Two-Level Memory/Storage Model

- 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



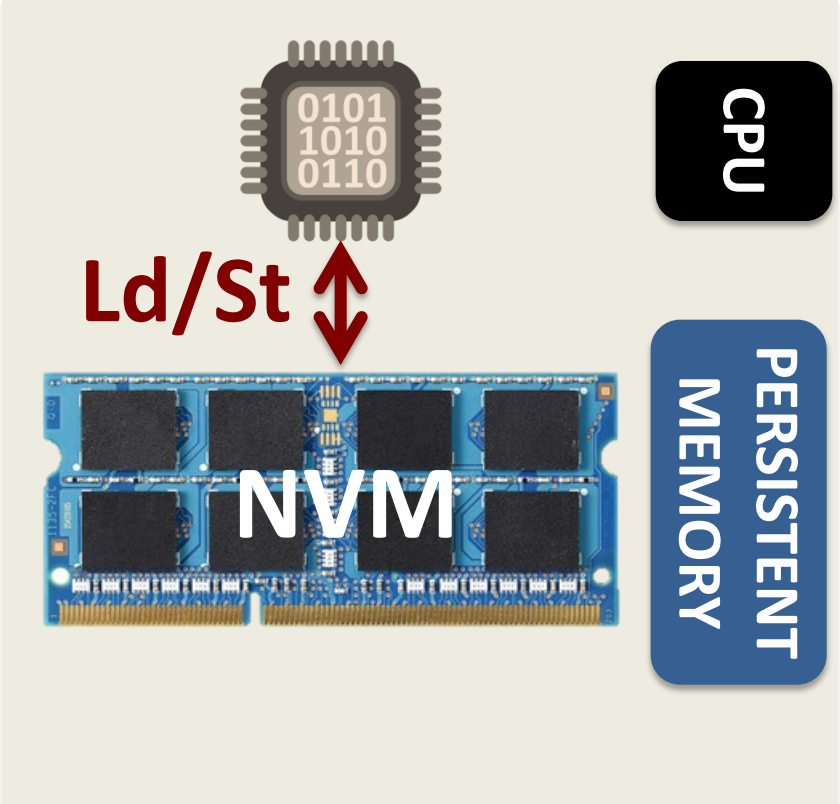
Unified Memory and Storage with NVM

- 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



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

PERSISTENT MEMORY

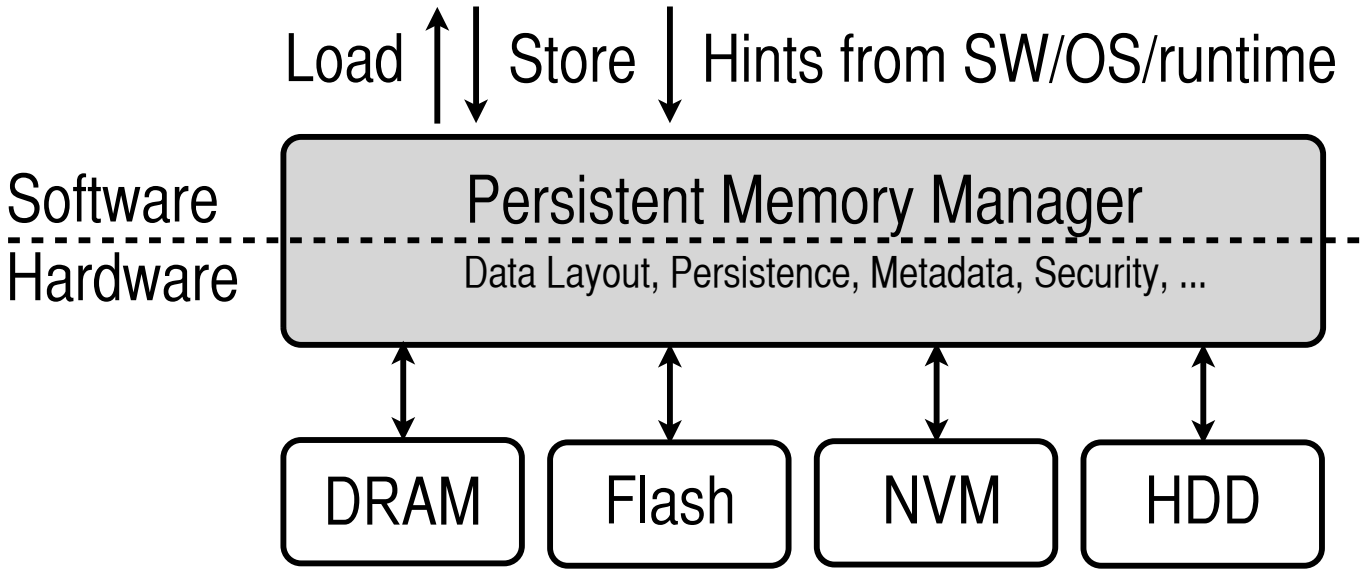


Provides an opportunity to manipulate persistent data directly

The Persistent Memory Manager (PMM)

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

Persistent objects



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

The Persistent Memory Manager (PMM)

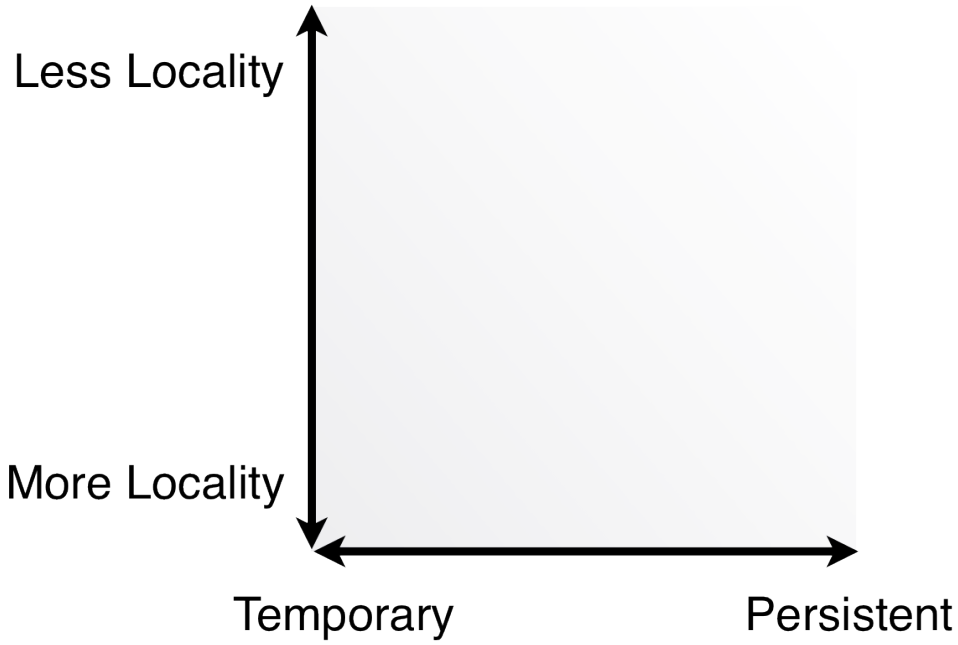
- Exposes a load/store interface to access persistent data
  - Applications can directly access persistent memory → no conversion, translation, location overhead for persistent data
- Manages data placement, location, persistence, security
  - To get the best of multiple forms of storage
- Manages metadata 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.

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

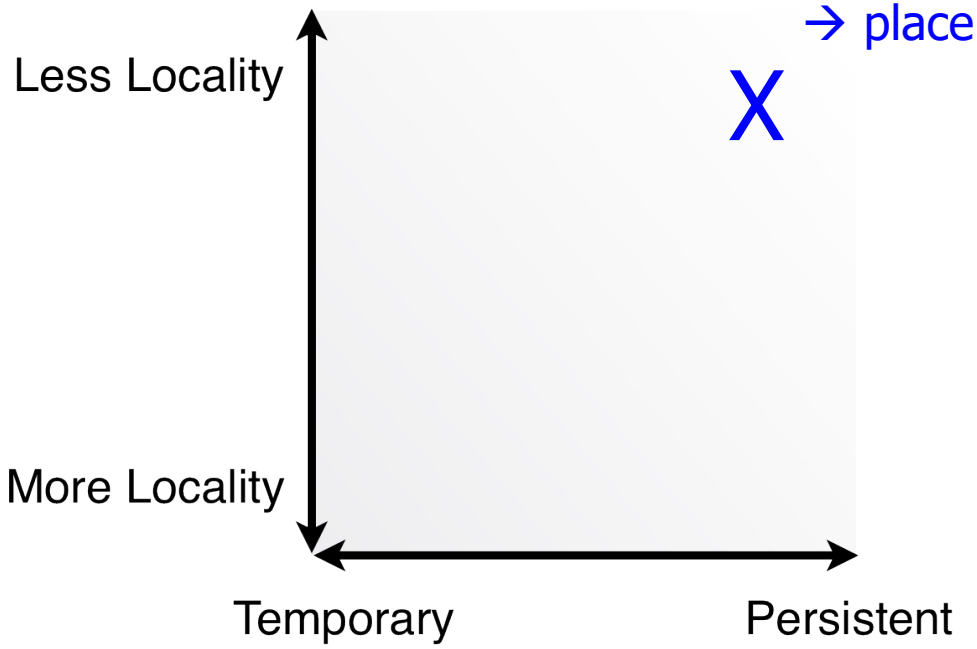
- Performance and energy can benefit from good placement of data among these devices
  - Utilizing the strengths of each device and avoiding their weaknesses, if possible
  - For example, consider two important application characteristics: locality and persistence

Efficient Data Mapping among Heterogeneous Devices

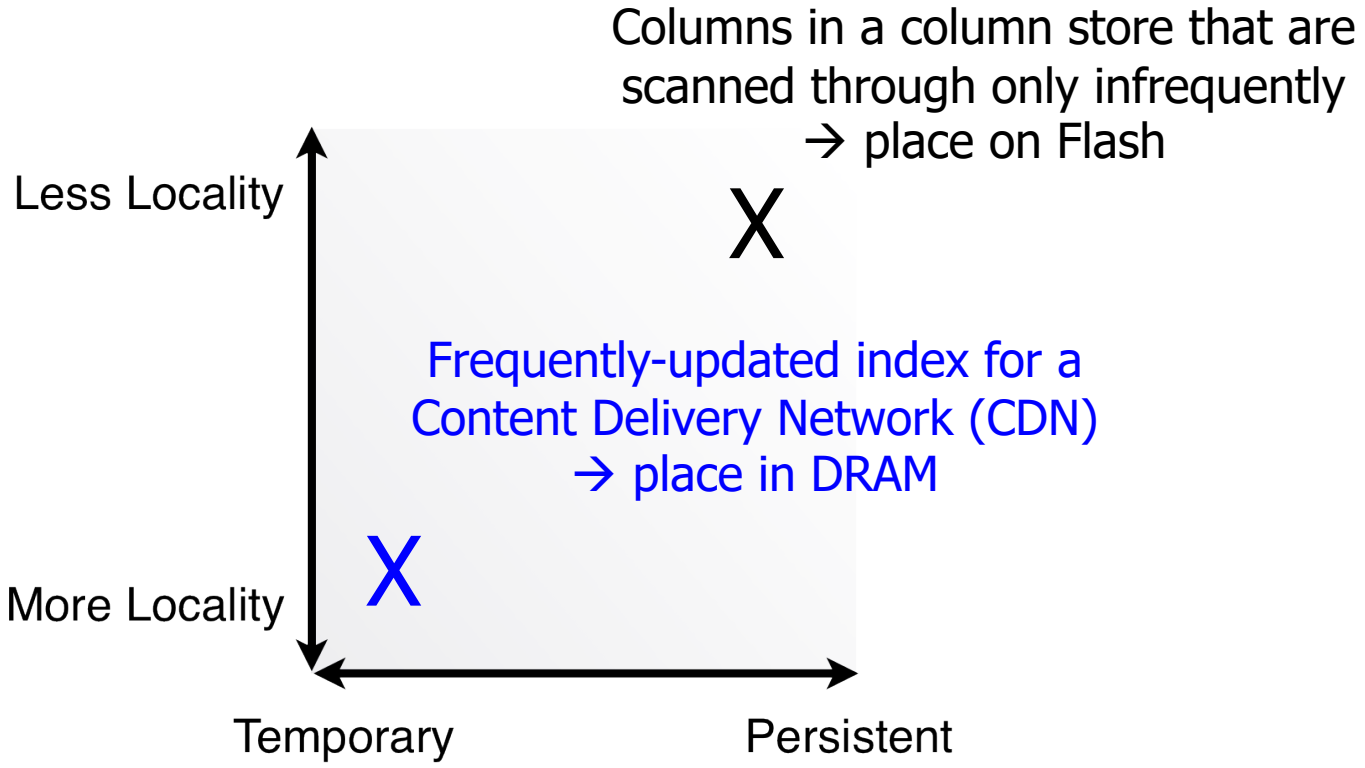


Efficient Data Mapping among Heterogeneous Devices

Columns in a column store that are scanned through only infrequently → place on Flash



Efficient Data Mapping among Heterogeneous Devices

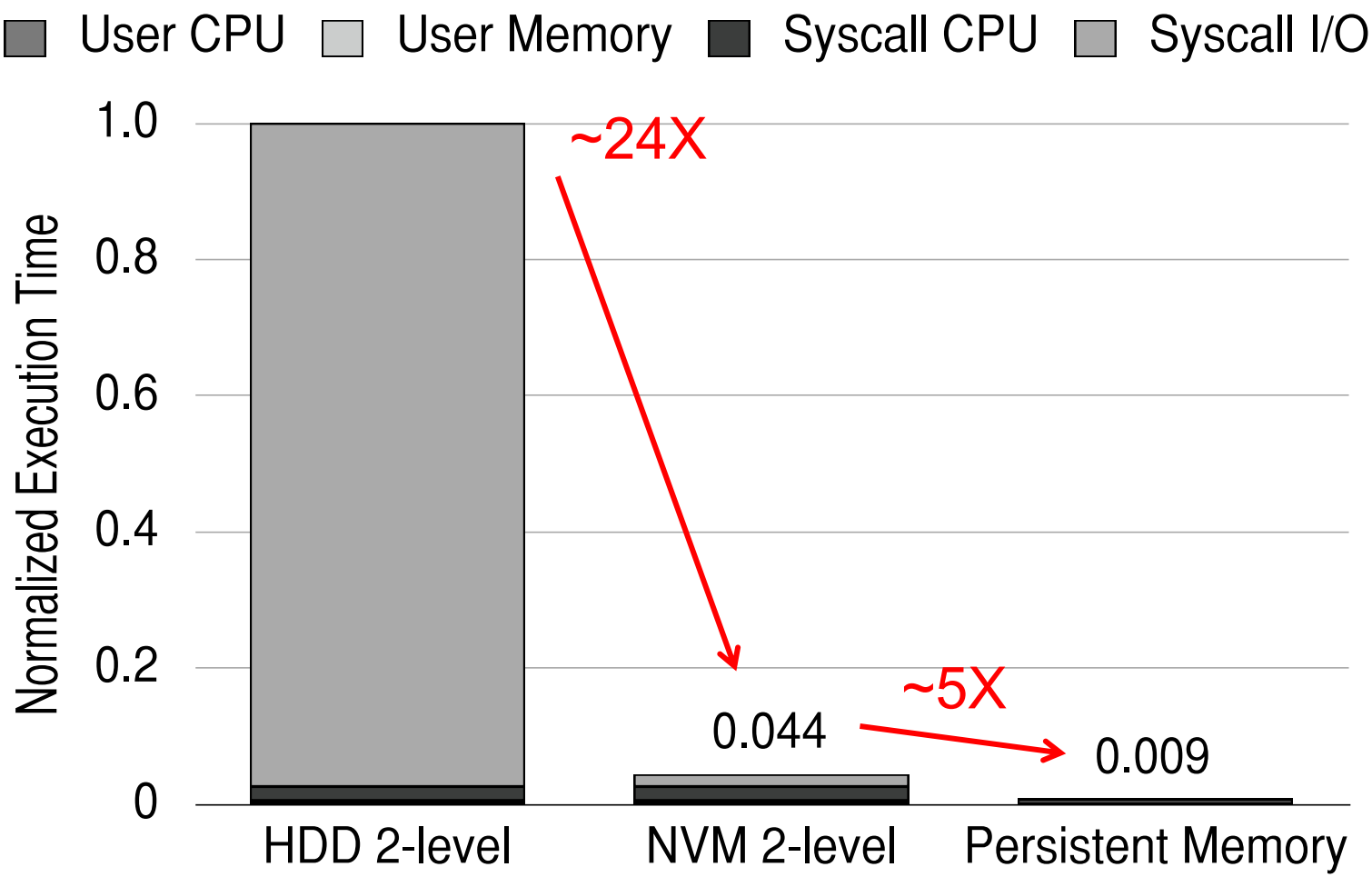


Applications or system software can provide hints for data placement

Evaluated Systems

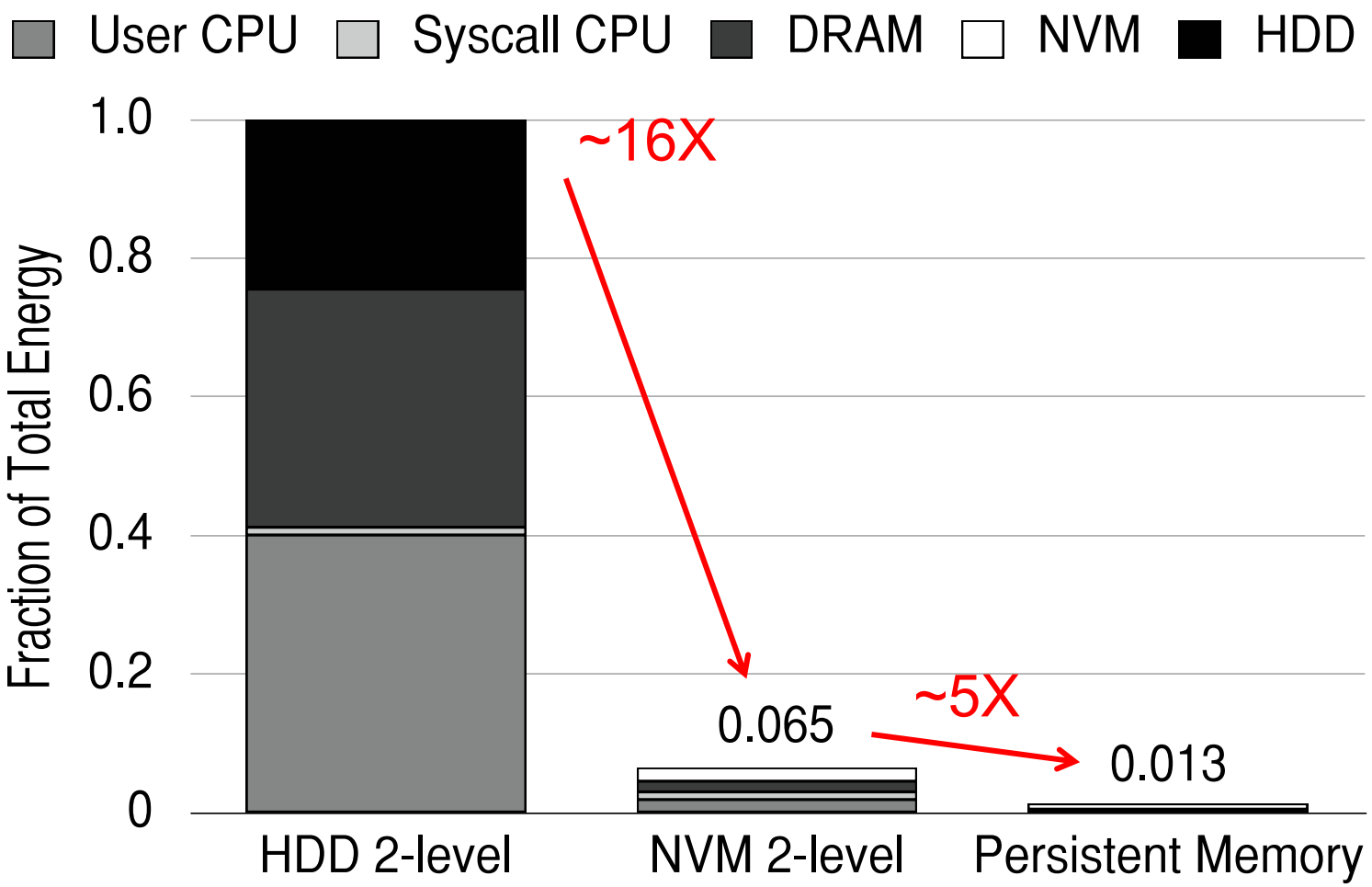
- HDD Baseline
  - Traditional system with volatile DRAM memory and persistent HDD storage
  - Overheads of operating system and file system code and buffering
- NVM Baseline (NB)
  - Same as HDD Baseline, but HDD is replaced with NVM
  - Still has OS/FS overheads of the two-level storage model
- Persistent Memory (PM)
  - Uses only NVM (no DRAM) to ensure full-system persistence
  - All data accessed using loads and stores
  - Does not waste time on system calls
  - Data is manipulated directly on the NVM device

Performance Benefits of a Single-Level Store



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.

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

Challenge and Opportunity

Combined
Memory & Storage

Challenge and Opportunity

A Unified Interface to
All Data

One Key Challenge in Persistent Memory

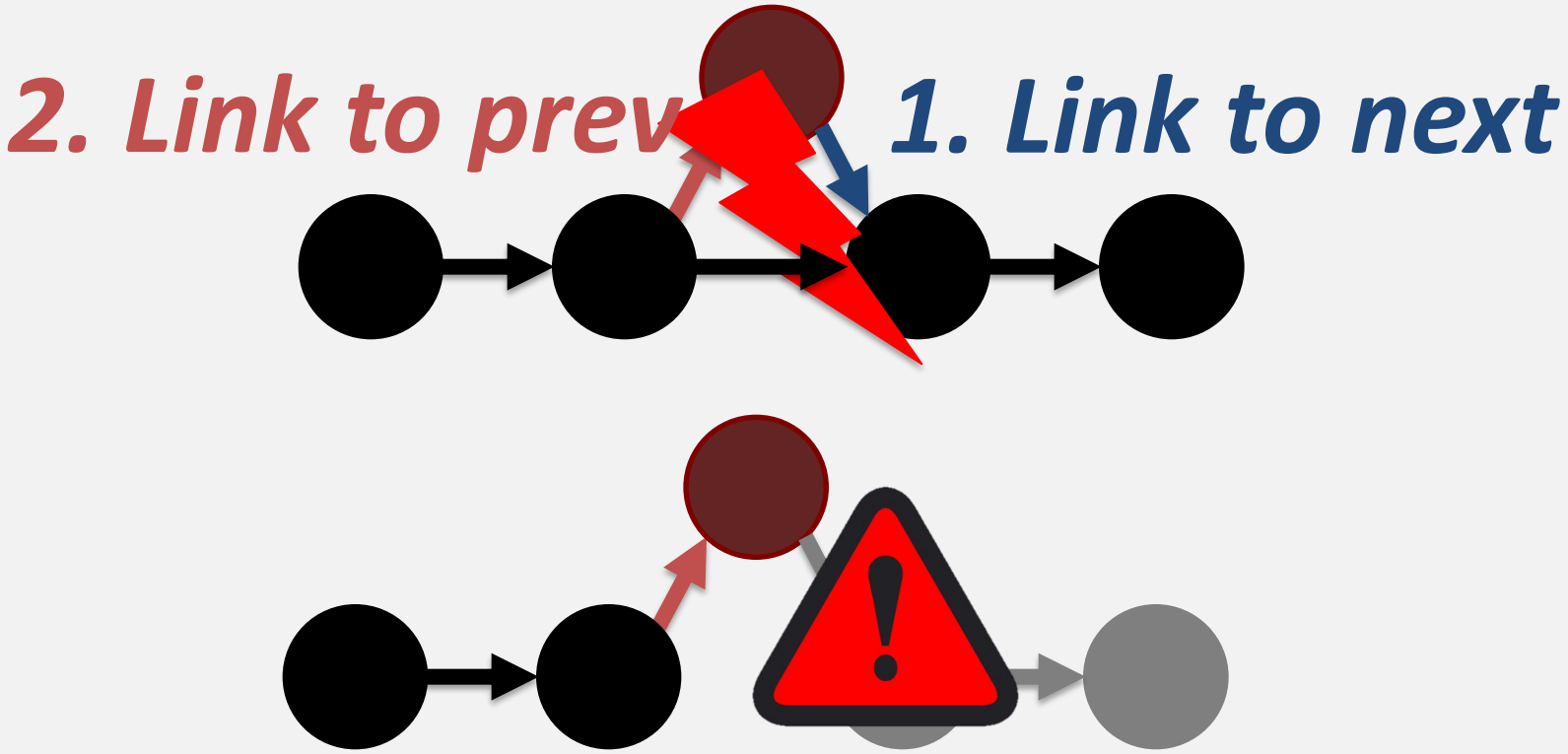
- How to ensure consistency of system/data if all memory is persistent?

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

- Many alternatives in-between…

CRASH CONSISTENCY PROBLEM

Add a node to a linked list



System crash can result in inconsistent memory state

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

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

Prohibited Operation `=` **Third party code can be inconsistent**

Burden on the programmers

OUR APPROACH: ThyNVM

Goal:
Software transparent consistency in persistent memory systems

Key Idea:
Periodically checkpoint state; recover to previous checkpt on crash

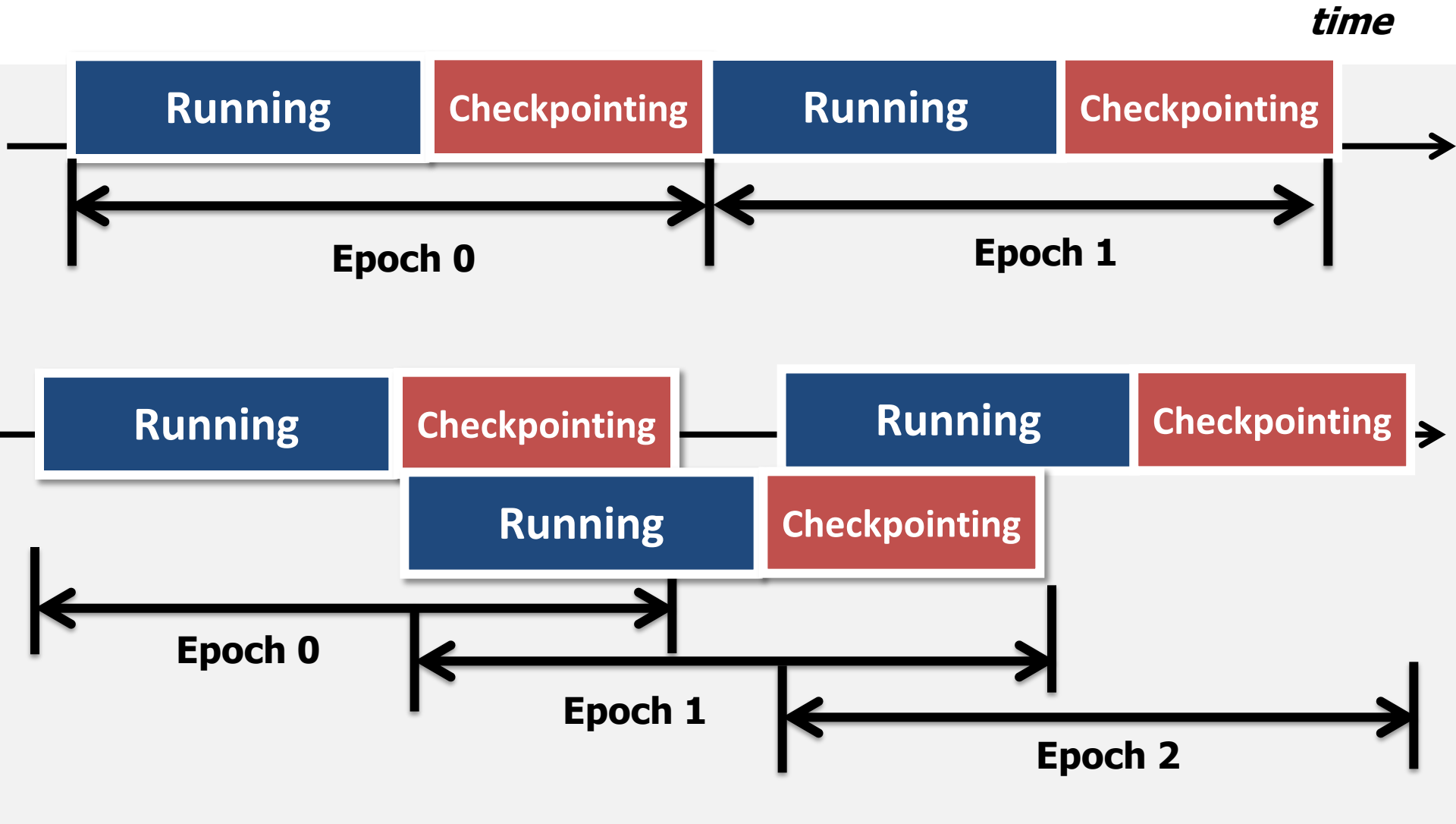
ThyNVM: Summary

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

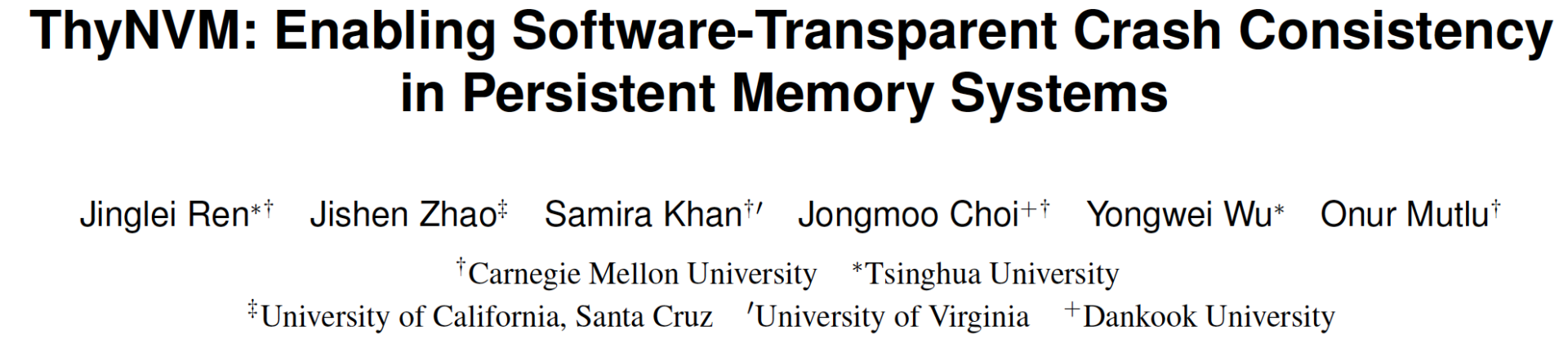
Performs within 4.9% of an *idealized DRAM* with zero cost consistency

2. OVERLAPPING CHECKPOINTING AND EXECUTION



More About ThyNVM

- 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]



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

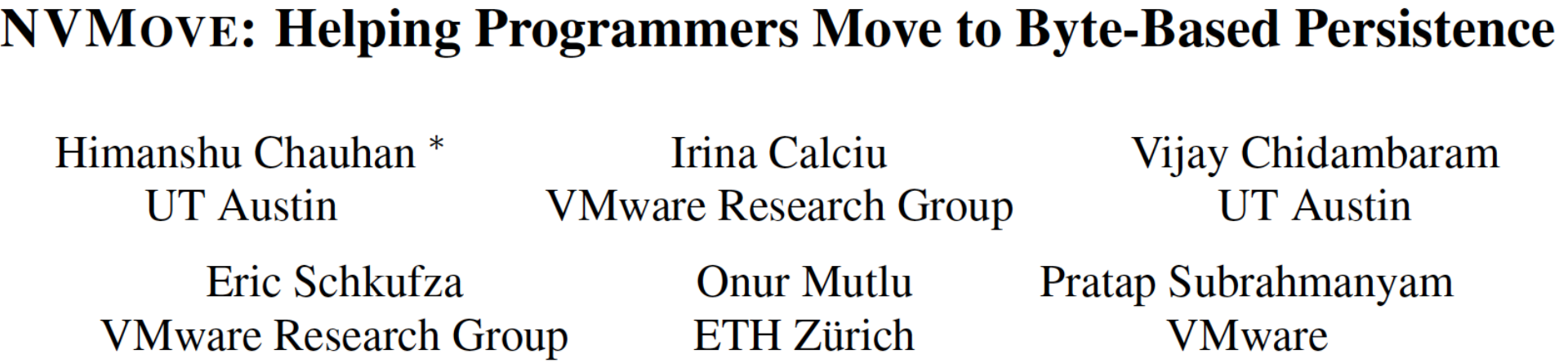
Another Key Challenge in Persistent Memory

Programming Ease

to Exploit Persistence

Tools/Libraries to Help Programmers

- Himanshu Chauhan, Irina Calciu, Vijay Chidambaram, Eric Schkufza, Onur Mutlu, and Pratap Subrahmanyam, **"NVMove: Helping Programmers Move to Byte-Based Persistence"**
  *Proceedings of the 4th Workshop on Interactions of NVM/Flash with Operating Systems and Workloads (**INFLOW**)*, Savannah, GA, USA, November 2016.
  [Slides (pptx) (pdf)]



NVMove: Helping Programmers Move to Byte-Based Persistence

Himanshu Chauhan * — UT Austin
Irina Calciu — VMware Research Group
Vijay Chidambaram — UT Austin
Eric Schkufza — VMware Research Group
Onur Mutlu — ETH Zürich
Pratap Subrahmanyam — VMware

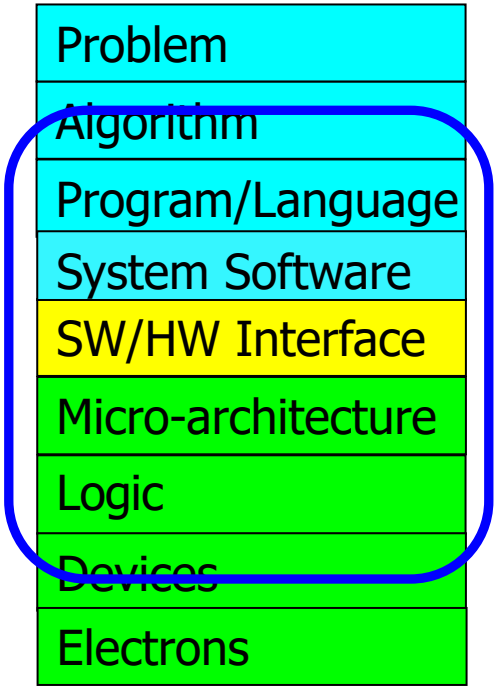
The Future of Emerging Technologies is Bright

- Regardless of challenges
  - in underlying technology and overlying problems/requirements

Can enable:

- Orders of magnitude improvements

- New applications and computing systems



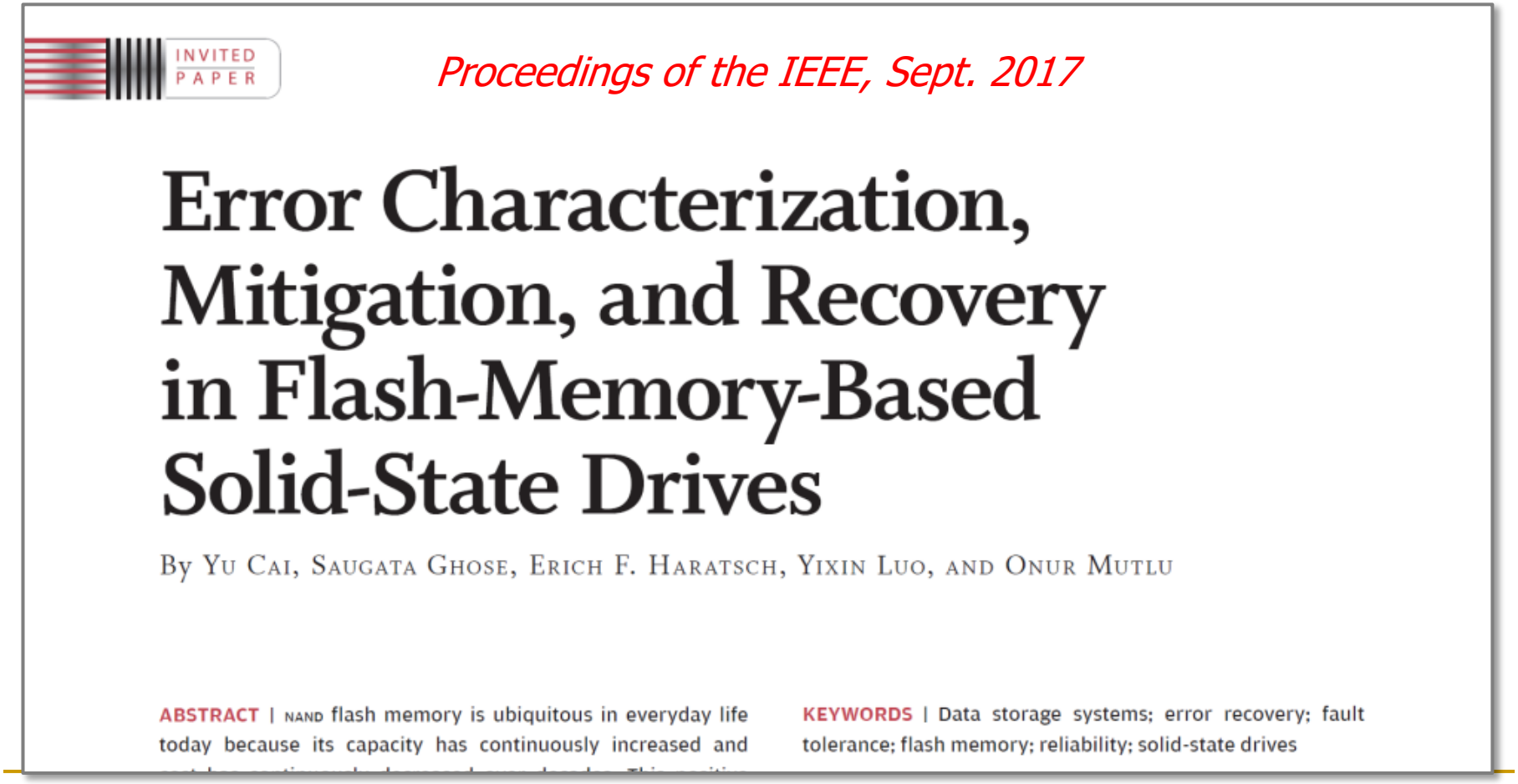
Yet, we have to

- Think across the stack

- Design enabling systems

If In Doubt, Refer to Flash Memory

- A very "doubtful" emerging technology
  - for at least two decades



INVITED PAPER

Proceedings of the IEEE, Sept. 2017

Error Characterization, Mitigation, and Recovery in Flash-Memory-Based Solid-State Drives

By Yu Cai, Saugata Ghose, Erich F. Haratsch, Yixin Luo, and Onur Mutlu

ABSTRACT | NAND flash memory is ubiquitous in everyday life today because its capacity has continuously increased and

KEYWORDS | Data storage systems; error recovery; fault tolerance; flash memory; reliability; solid-state drives

https://arxiv.org/pdf/1706.08642

Computer Architecture

Lecture 17a: Emerging Memory Technologies II

Prof. Onur Mutlu
ETH Zürich

Fall 2019

28 November 2019