Computer Architecture

Lecture 13a: Emerging Memory Technologies II

Prof. Onur Mutlu

ETH Zürich

Fall 2022

10 November 2022

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**
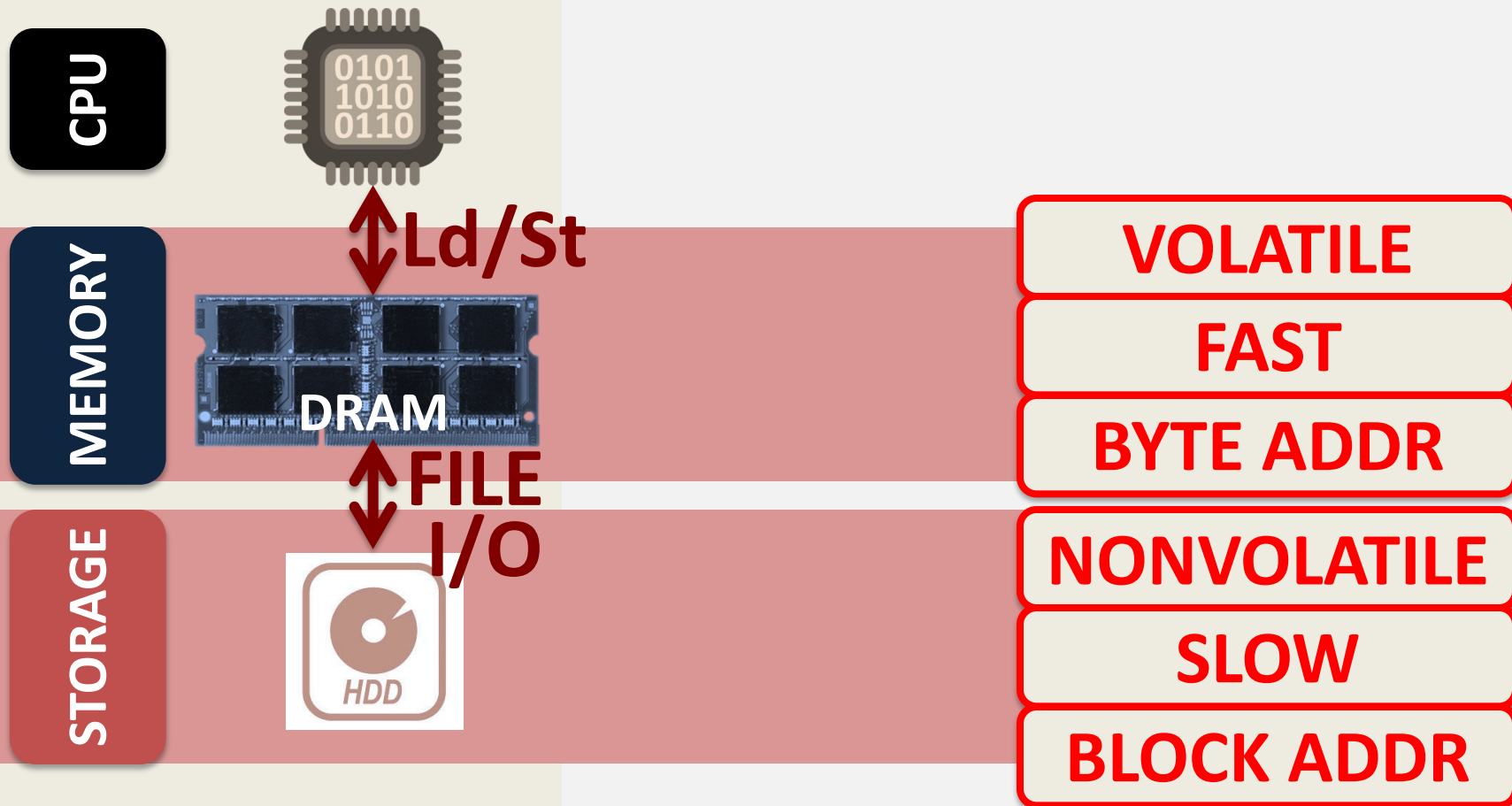 - e.g., enabling efficient search and filtering

New: In-Memory Crossbar Array Operations
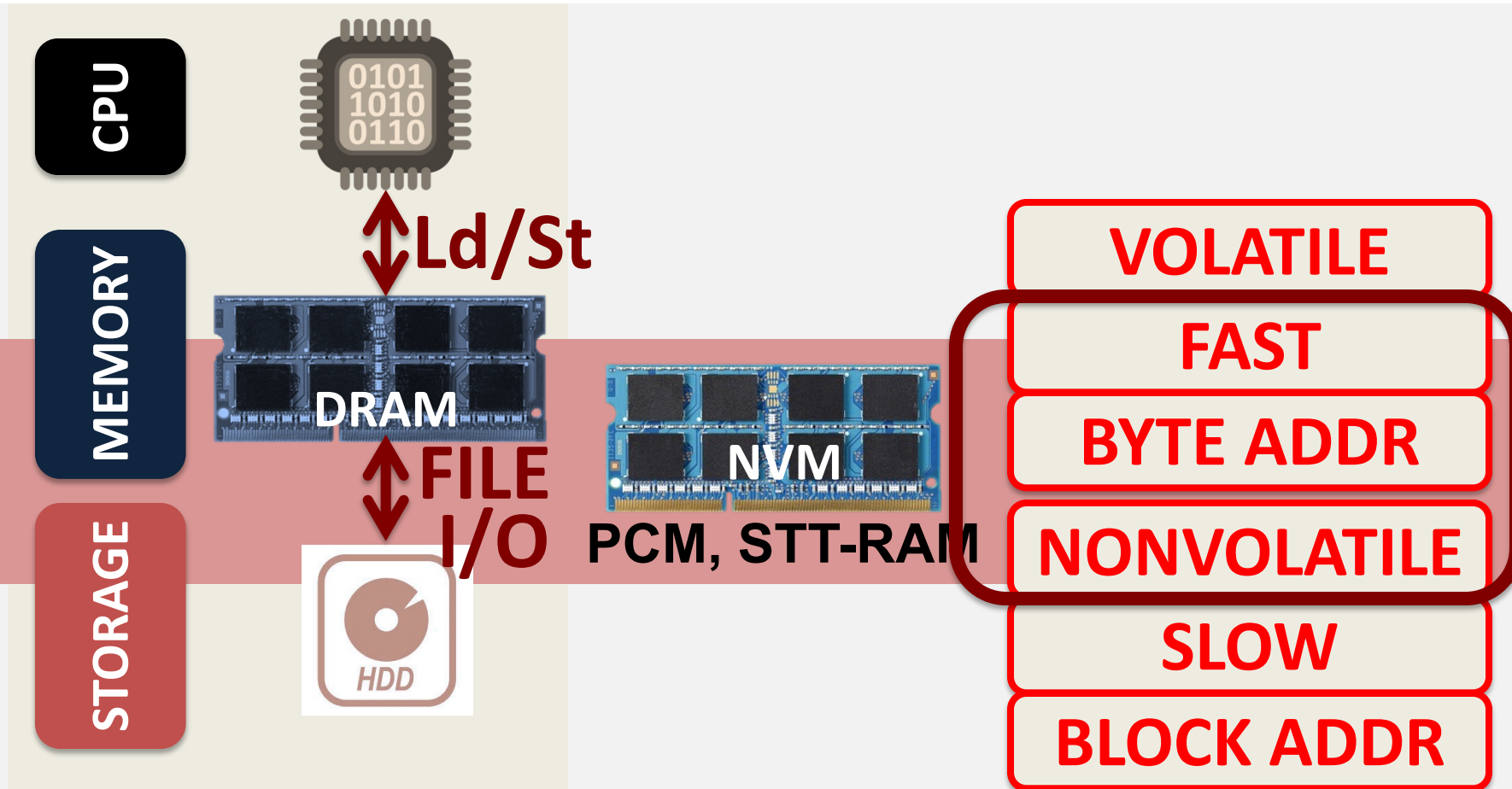
Other Opportunities with Emerging Technologies

- **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**
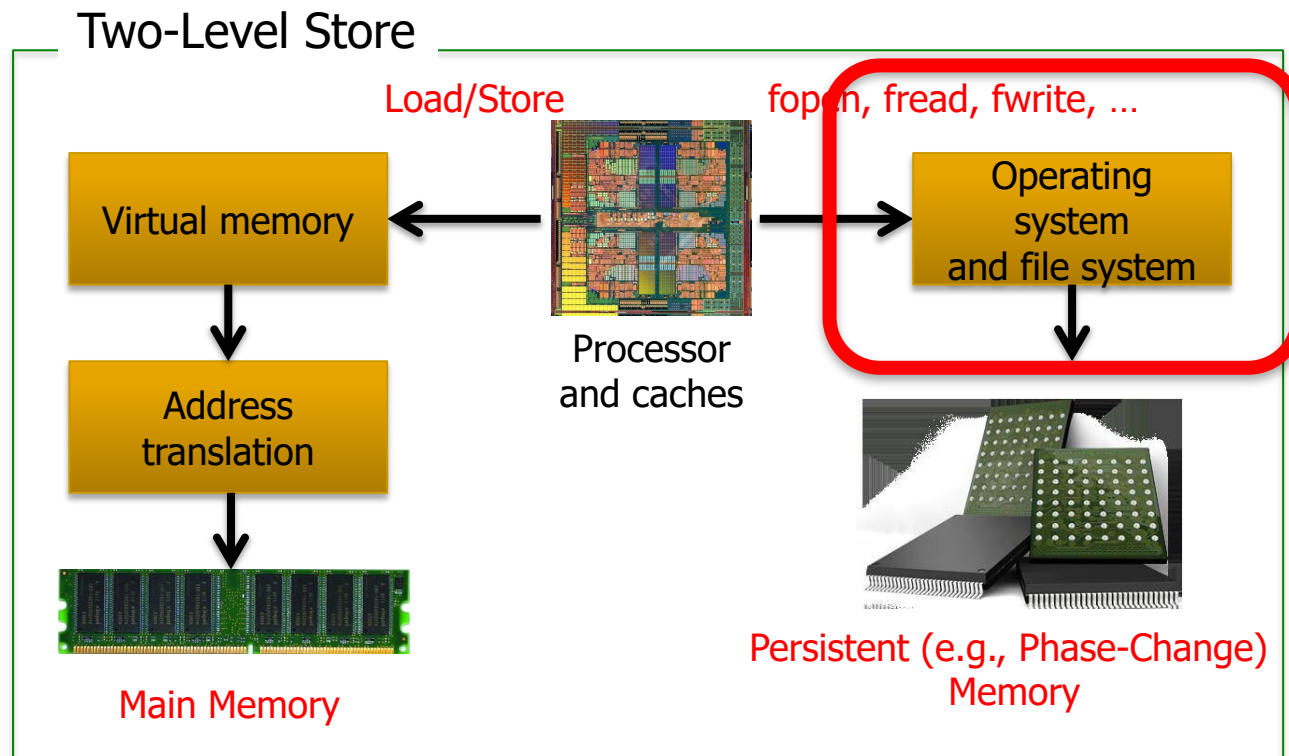   - e.g., enabling efficient search and filtering

TWO-LEVEL STORAGE MODEL

CPU

0101
1010
0110

MEMORY

$\updownarrow$ **Ld/St**

DRAM

STORAGE

$\updownarrow$ **FILE I/O**

HDD

VOLATILE

FAST

BYTE ADDR

NONVOLATILE

SLOW

BLOCK ADDR

TWO-LEVEL STORAGE MODEL

CPU

MEMORY

STORAGE

0101
1010
0110

↕ **Ld/St**

DRAM

FILE I/O

HDD

NVM

PCM, STT-RAM

VOLATILE

FAST

BYTE ADDR

NONVOLATILE

SLOW

BLOCK ADDR

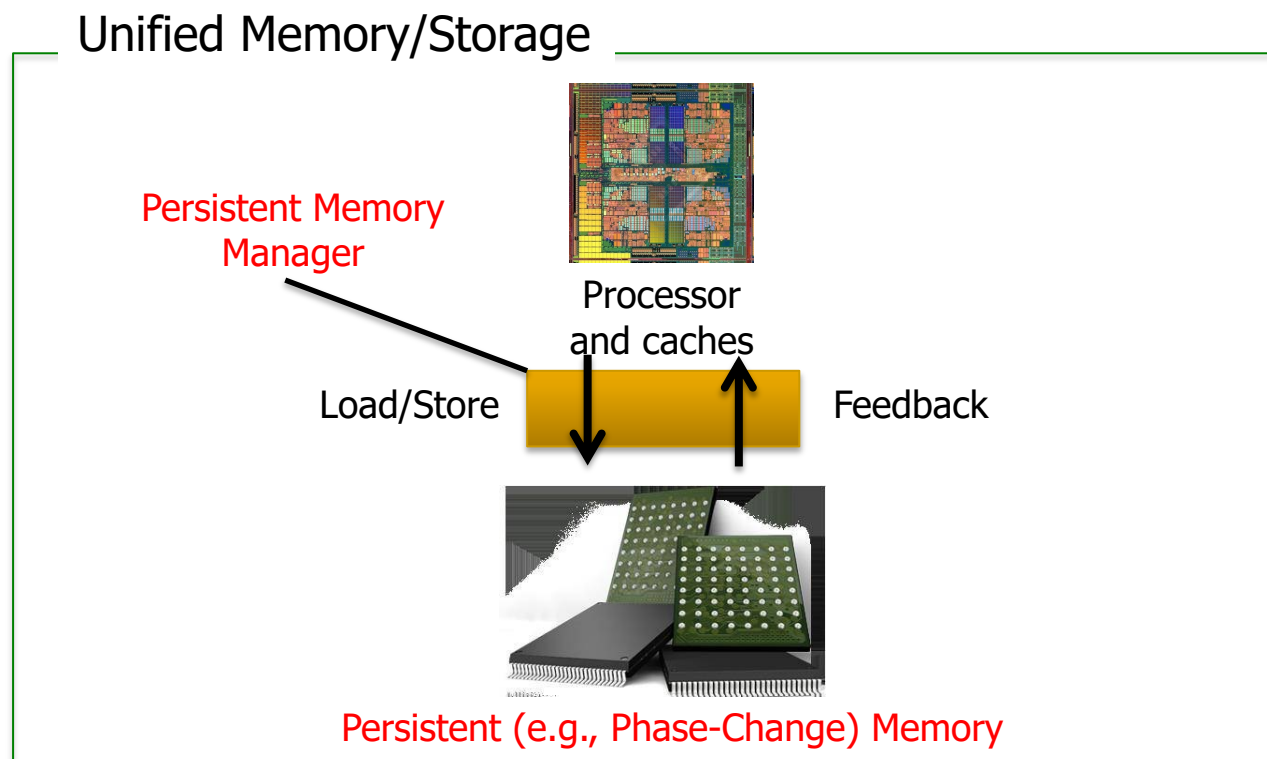**Non-volatile memories combine characteristics of memory and storage**

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
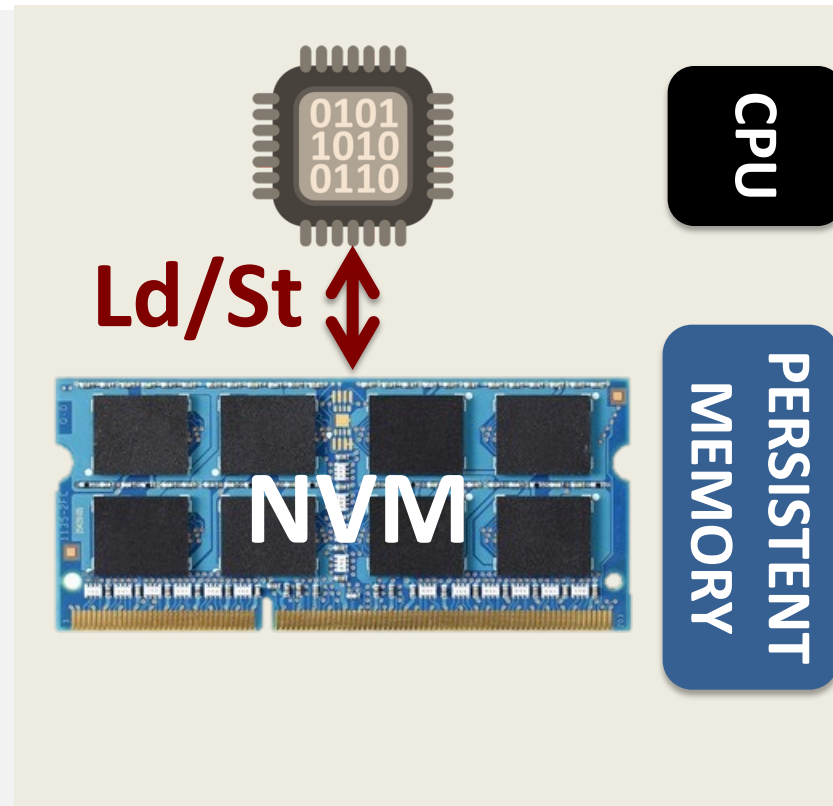
Two-Level Store

Load/Store   fopen, fread, fwrite, …

| Virtual memory | | Operating system and file system |

Processor and caches

Address translation

Main Memory

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

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

Unified Memory/Storage

Persistent Memory Manager

Processor and caches

Load/Store   Feedback

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

SAFARI   Meza+, "A Case for Efficient Hardware-Software Cooperative Management of Storage and Memory," WEED 2013.   8

PERSISTENT MEMORY

Ld/St

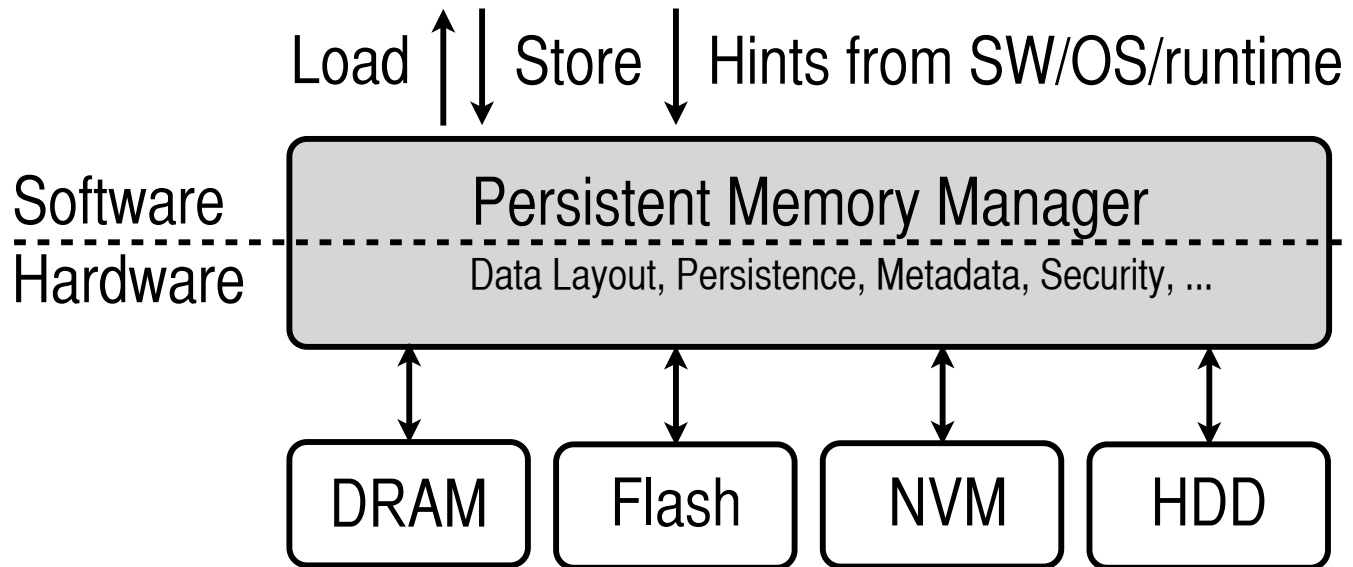**NVM**

CPU

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";        Persistent objects
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  }
```

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

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
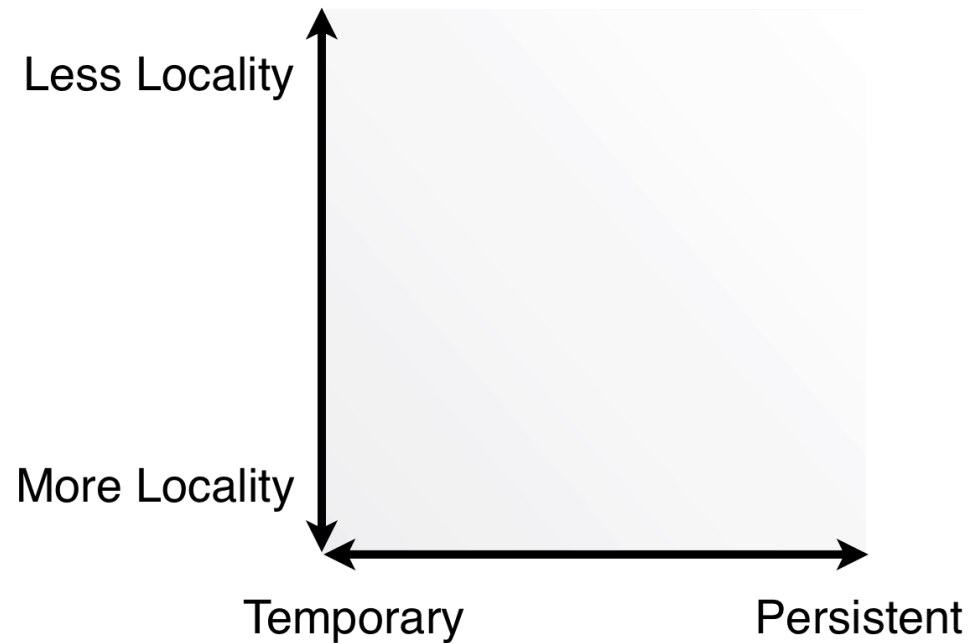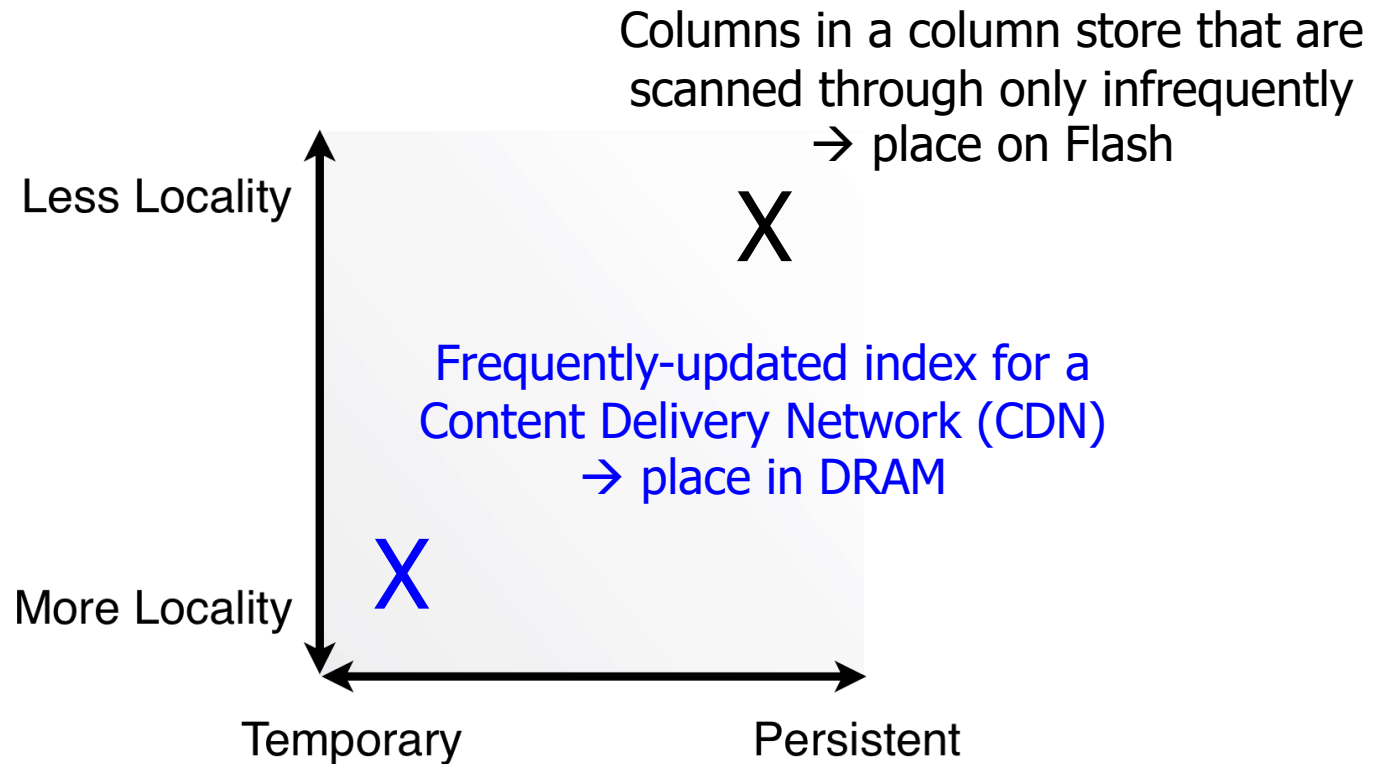
Efficient Data Mapping among Heterogeneous Devices

Efficient Data Mapping among Heterogeneous Devices

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

X

Less Locality

More Locality

Temporary

Persistent

Efficient Data Mapping among Heterogeneous Devices

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

X

Less Locality

Frequently-updated index for a
Content Delivery Network (CDN)
→ place in DRAM

X

More Locality

Temporary   Persistent
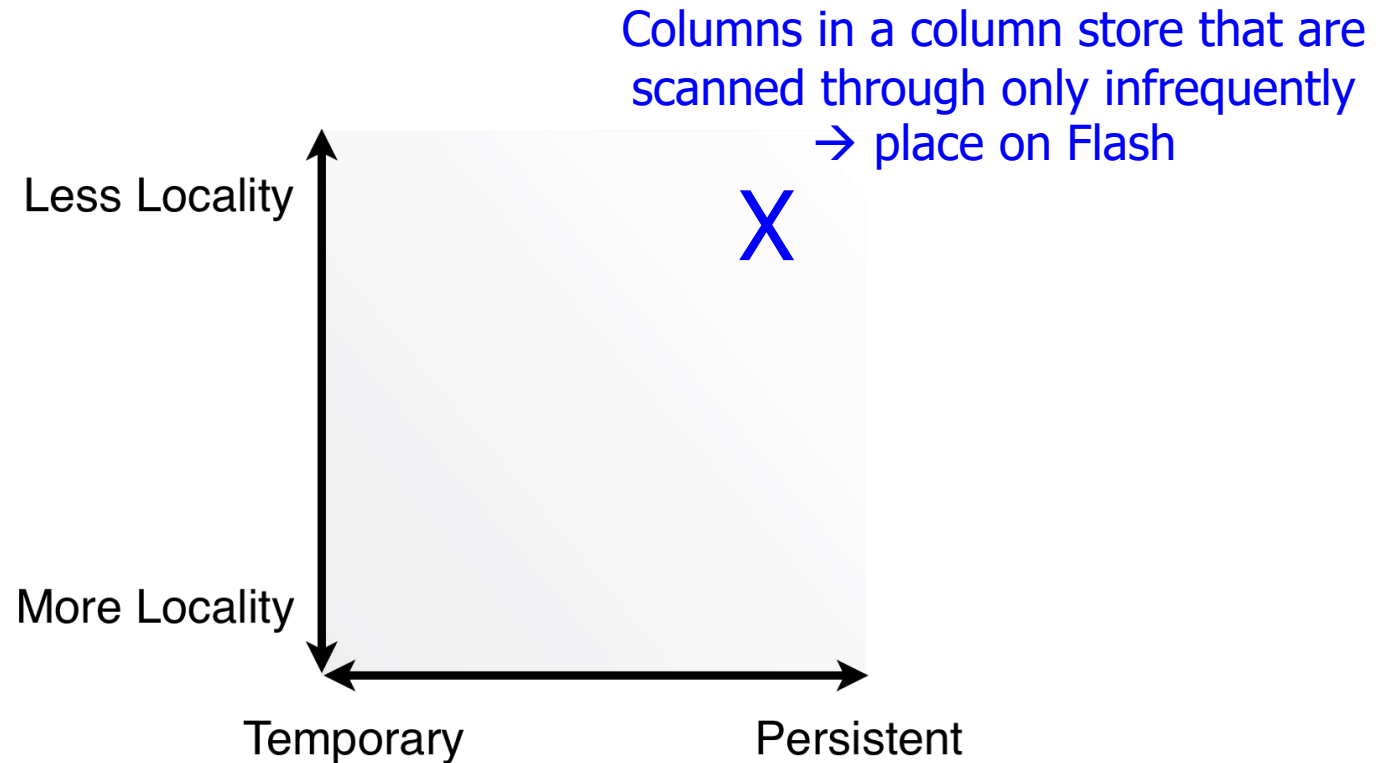
Applications or system software can provide hints for data placement

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)**
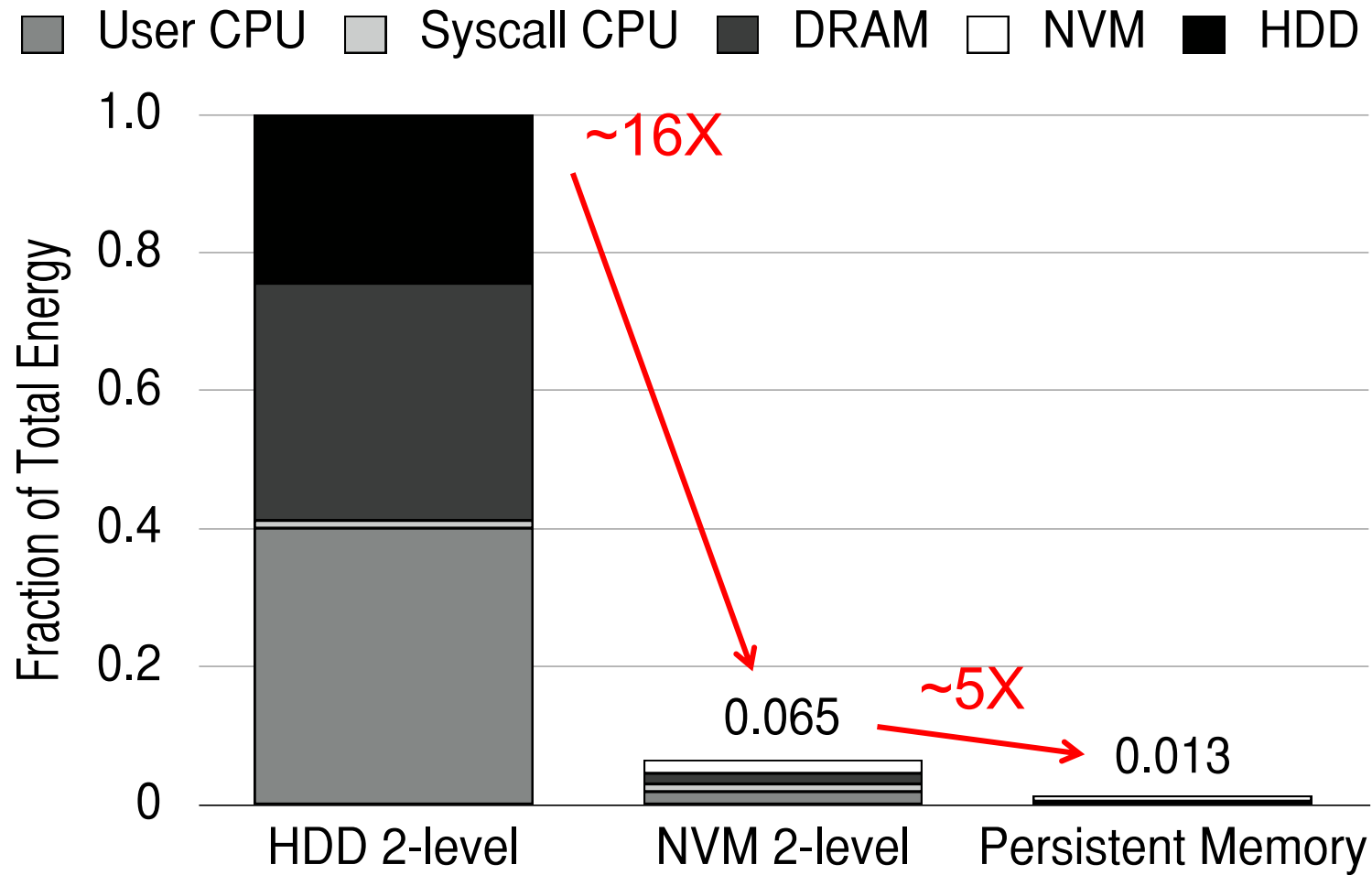   - Uses only NVM (no DRAM) to ensure full-system persistence
    - All data accessed using loads and stores
    - Does not waste time on system calls
    - Data is manipulated directly on the NVM device

Performance Benefits of a Single-Level Store

Meza+, "A Case for Efficient Hardware-Software Cooperative Management of Storage and Memory," WEED 2013.
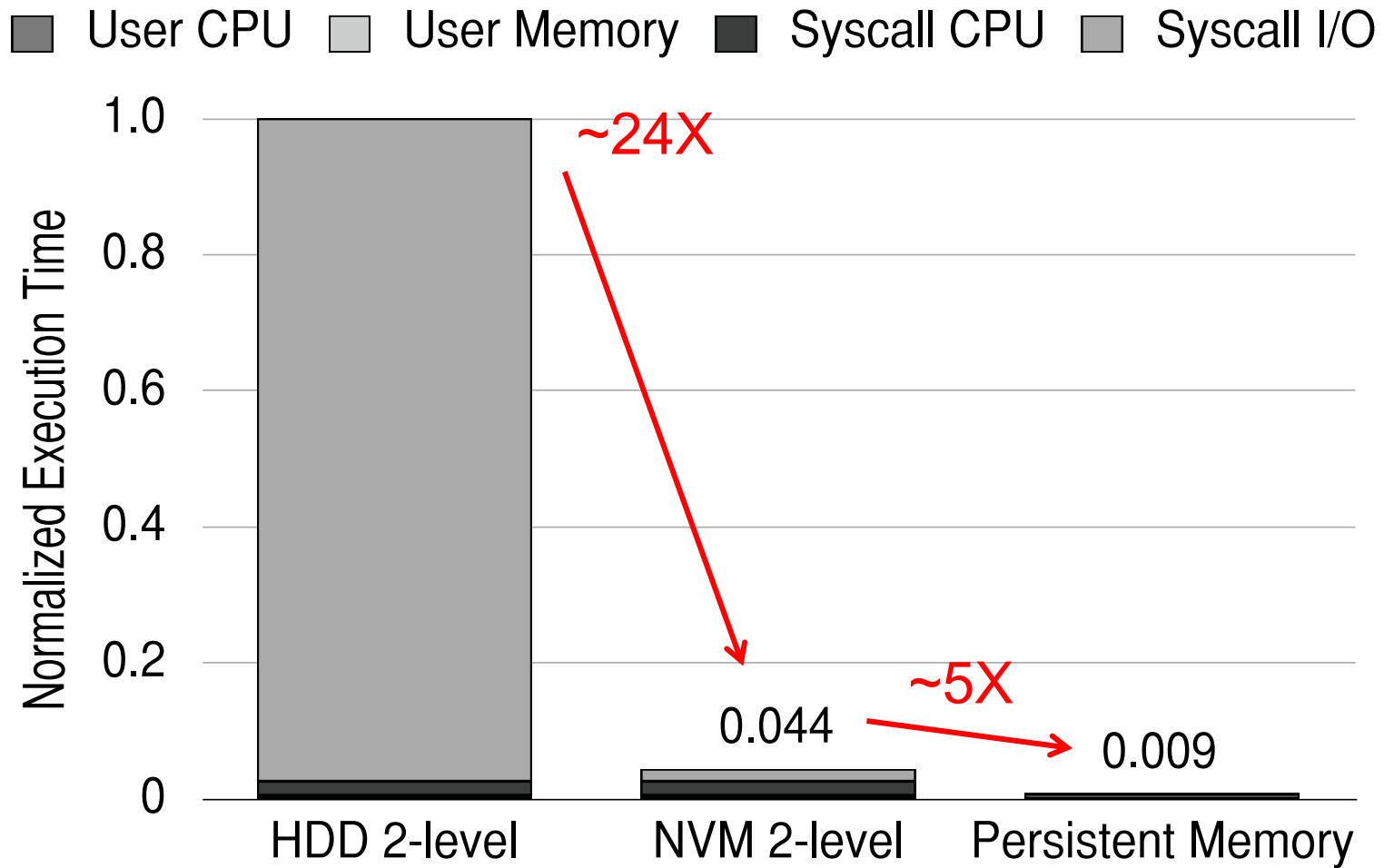
Energy Benefits of a Single-Level Store

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

SAFARI

Challenge and Opportunity

A Unified Interface to **All Data**

SAFARI

# Intel Optane Persistent Memory (2019)

- Non-volatile main memory
- Based on 3D-XPoint Technology

UPMEM Processing-in-DRAM Engine (2019)

- Processing in DRAM Engine

- Includes **standard DIMM modules**, with a **large number of DPU processors** combined with DRAM chips.

- Replaces **standard** DIMMs
  - DDR4 R-DIMM modules
    - 8GB+128 DPUs (16 PIM chips)
    - Standard 2x-nm DRAM process
  - **Large amounts of** compute & memory bandwidth
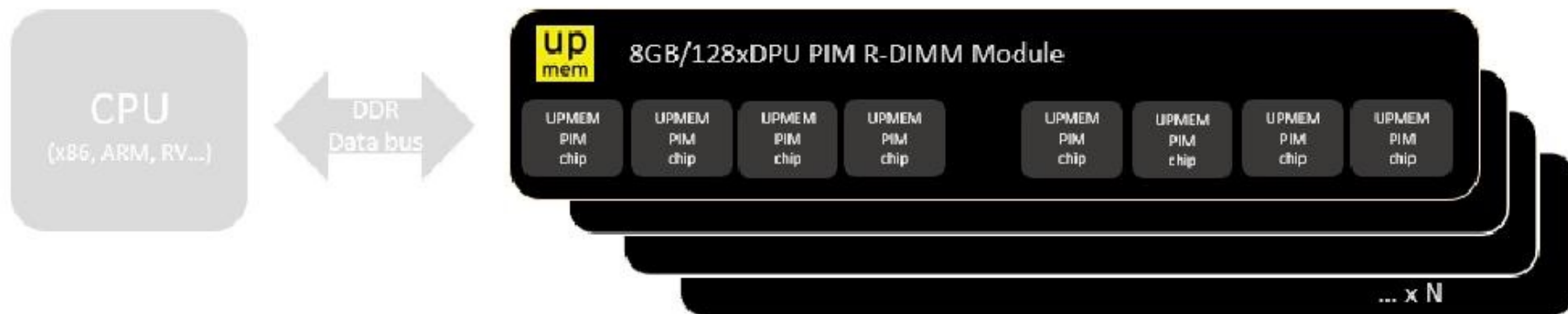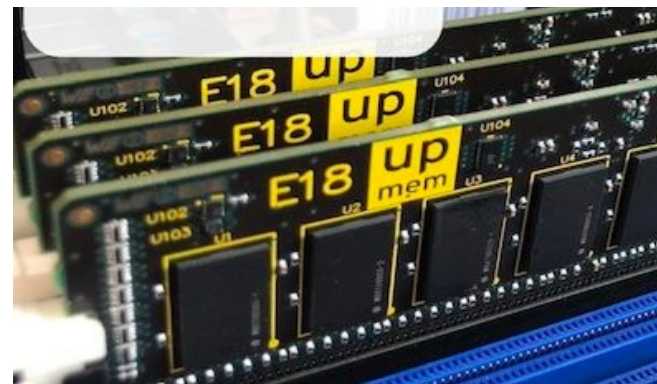
One Key Challenge in Persistent Memory

- 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

2. Link to prev   *1. Link to next*
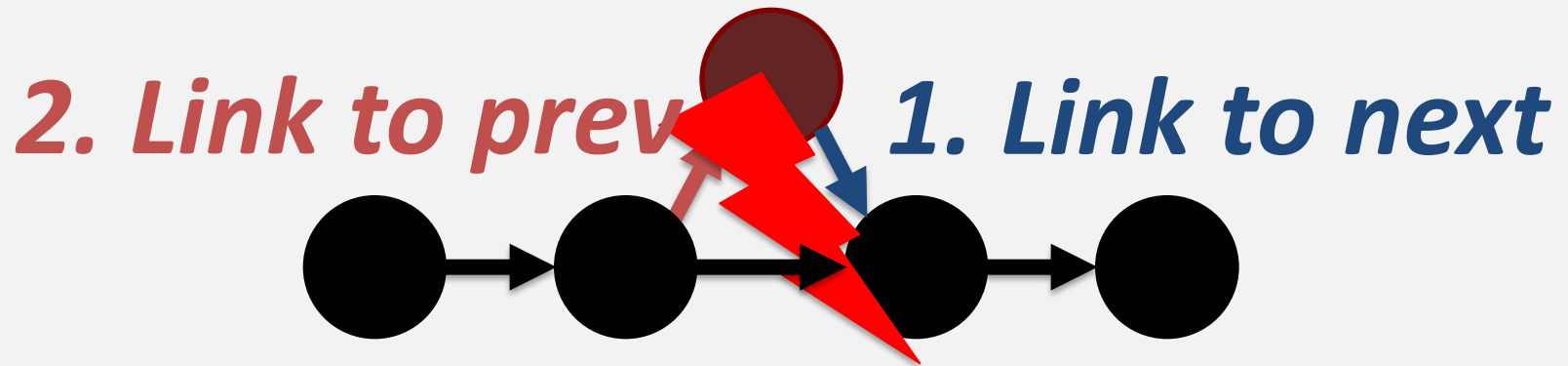
System crash can result in inconsistent memory state

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;
}
```

get_chain(ht, key)

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;
}
```

get_chain(ht, key)

Need a new implementation

TMLIST_FIND

**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;
}
```

get_chain(ht, key)

Need a new implementation

TMLIST_FIND

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

time
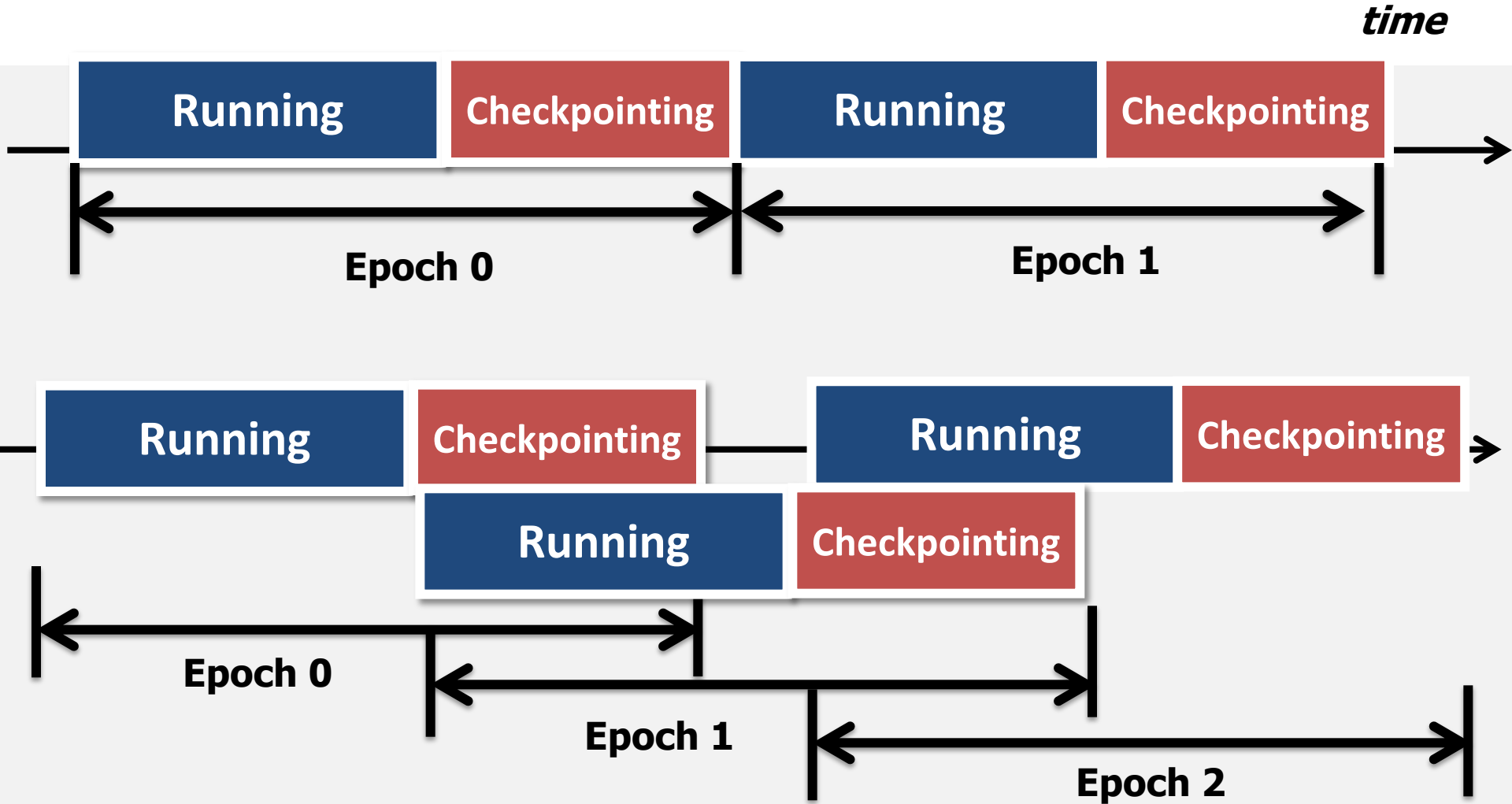
More About ThyNVM

- 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

Consistency Support for Persistent Memory

- Youyou Lu, Jiwu Shu, Long Sun, and Onur Mutlu,
"Loose-Ordering Consistency for Persistent Memory"
Proceedings of the 32nd IEEE International Conference on Computer Design (**ICCD**), Seoul, South Korea, October 2014.
[Slides (pptx) (pdf)]
[Erratum]

Loose-Ordering Consistency for Persistent Memory

Youyou Lu [†], Jiwu Shu [†][§], Long Sun [†] and Onur Mutlu [‡]
[†]Department of Computer Science and Technology, Tsinghua University, Beijing, China
[§]State Key Laboratory of Computer Architecture, Institute of Computing Technology,
Chinese Academy of Sciences, Beijing, China
[‡]Computer Architecture Laboratory, Carnegie Mellon University, Pittsburgh, PA, USA
luyy09@mails.tsinghua.edu.cn, shujw@tsinghua.edu.cn, sun-l12@mails.tsinghua.edu.cn, onur@cmu.edu

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

Security and Data Privacy Issues

Security and Privacy Issues of NVM

- Endurance problems → Wearout attacks

- Hybrid memories → Performance attacks

- Data not erased after power-off → Privacy breaches
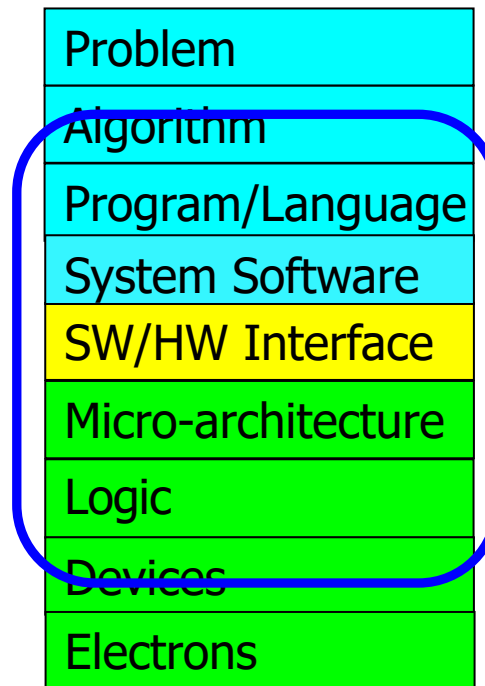
SAFARI

Conclusion

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

| Problem |
| Algorithm |
| Program/Language |
| System Software |
| SW/HW Interface |
| Micro-architecture |
| Logic |
| Devices |
| Electrons |

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

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

https://arxiv.org/pdf/1706.08642

Many Research & Design Opportunities

- Enabling combined memory & storage system

- Computation in/using NVM based memories

- Hybrid memory systems

- Security and privacy issues in persistent memory

- Reliability and endurance related problems

- Virtual memory systems for NVM (e.g., virtual block interface)

SAFARI

Food for Thought: Rethinking Virtual Memory

- Nastaran Hajinazar, Pratyush Patel, Minesh Patel, Konstantinos Kanellopoulos, Saugata Ghose, Rachata Ausavarungnirun, Geraldo Francisco de Oliveira Jr., Jonathan Appavoo, Vivek Seshadri, and Onur Mutlu,
  "The Virtual Block Interface: A Flexible Alternative to the Conventional Virtual Memory Framework"
  Proceedings of the 47th International Symposium on Computer Architecture (**ISCA**), Valencia, Spain, June 2020.
  [Slides (pptx) (pdf)]
  [Lightning Talk Slides (pptx) (pdf)]
  [ARM Research Summit Poster (pptx) (pdf)]
  [Talk Video (26 minutes)]
  [Lightning Talk Video (3 minutes)]

The Virtual Block Interface: A Flexible Alternative to the Conventional Virtual Memory Framework

Nastaran Hajinazar[*][†]   Pratyush Patel[⋈]   Minesh Patel[*]   Konstantinos Kanellopoulos[*]   Saugata Ghose[‡]
Rachata Ausavarungnirun[⊙]   Geraldo F. Oliveira[*]   Jonathan Appavoo[◇]   Vivek Seshadri[▽]   Onur Mutlu[*‡]
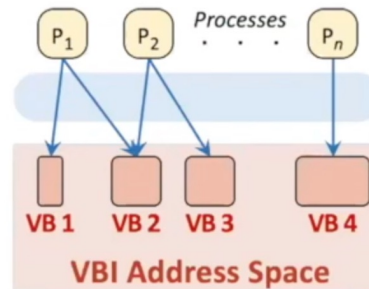
[*]ETH Zürich   [†]Simon Fraser University   [⋈]University of Washington   [‡]Carnegie Mellon University
[⊙]King Mongkut's University of Technology North Bangkok   [◇]Boston University   [▽]Microsoft Research India

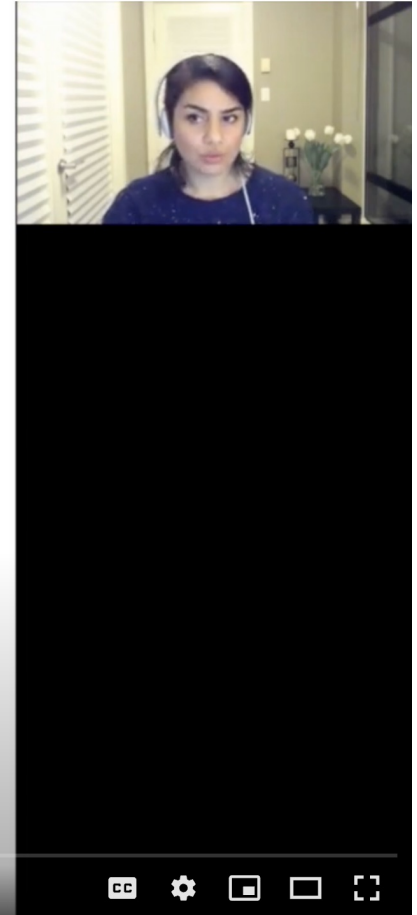# Virtual Block Interface Talk Video

Computer Architecture
Lecture 13a: Emerging Memory Technologies II

Prof. Onur Mutlu

ETH Zürich

Fall 2022

10 November 2022