

# Computer Architecture

## Lecture 16a: Opportunities and Challenges of Emerging Memory Tech.

Prof. Onur Mutlu

ETH Zürich

Fall 2020

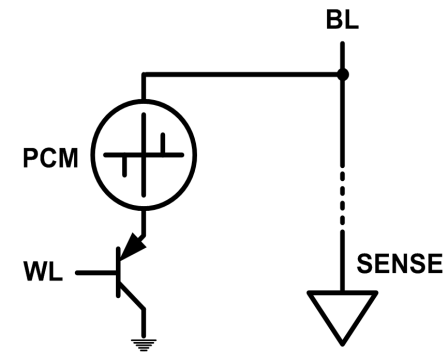
19 November 2020

# Emerging Memory Technologies

# Recall: Emerging Memory 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 2009])
  - ❑ 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?



# Recall: Phase Change Memory: Pros and Cons

---

## ■ Pros over DRAM

- ❑ Better technology scaling (capacity and cost)
- ❑ Non volatile → Persistent
- ❑ Low idle power (no refresh)

## ■ Cons

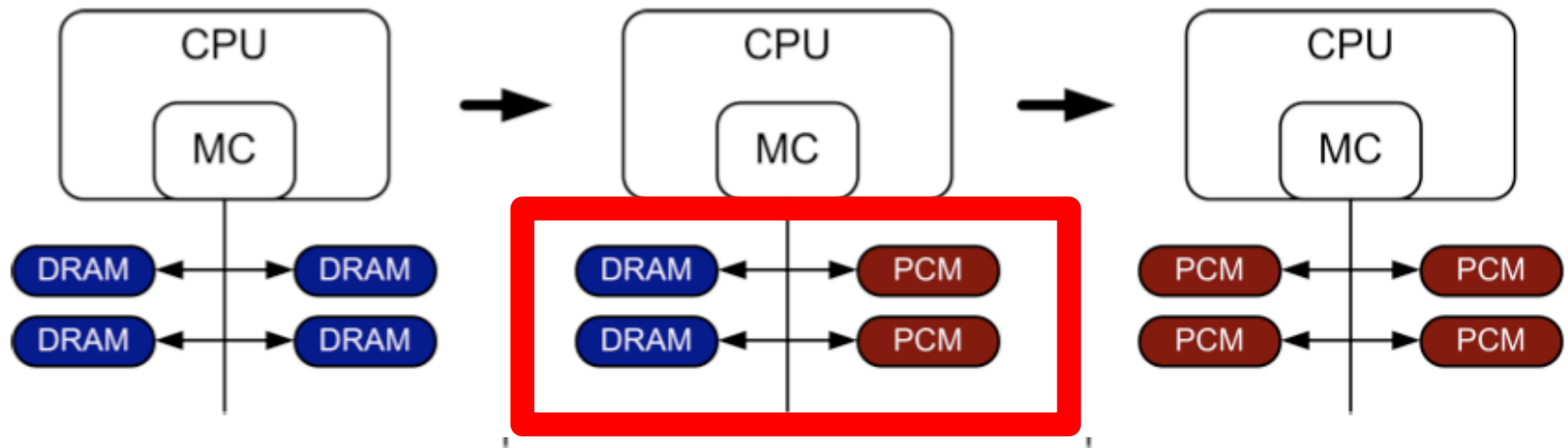
- ❑ Higher latencies:  $\sim 4\text{-}15\times$  DRAM (especially write)
- ❑ Higher active energy:  $\sim 2\text{-}50\times$  DRAM (especially write)
- ❑ Lower endurance (a cell dies after  $\sim 10^8$  writes)
- ❑ Reliability issues (resistance drift)

## ■ Challenges in enabling PCM as DRAM replacement/helper:

- ❑ Mitigate PCM shortcomings
- ❑ Find the right way to place PCM in the system

# Recall: PCM-based Main Memory (I)

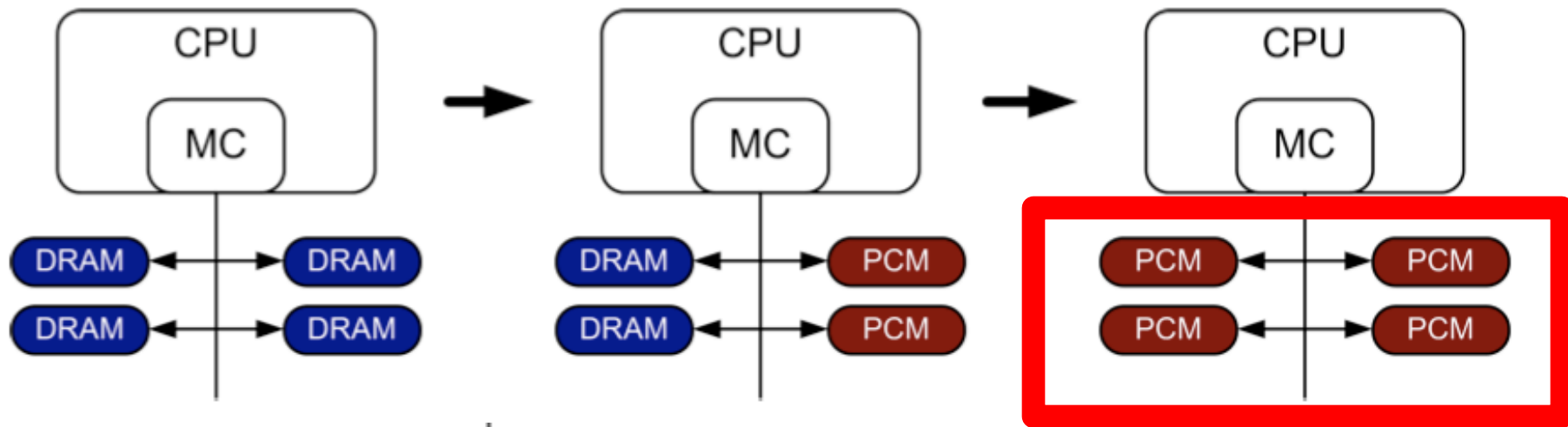
- How should PCM-based (main) memory be organized?



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

# Recall: 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

# Emerging Memory Technologies: Opportunities and Challenges

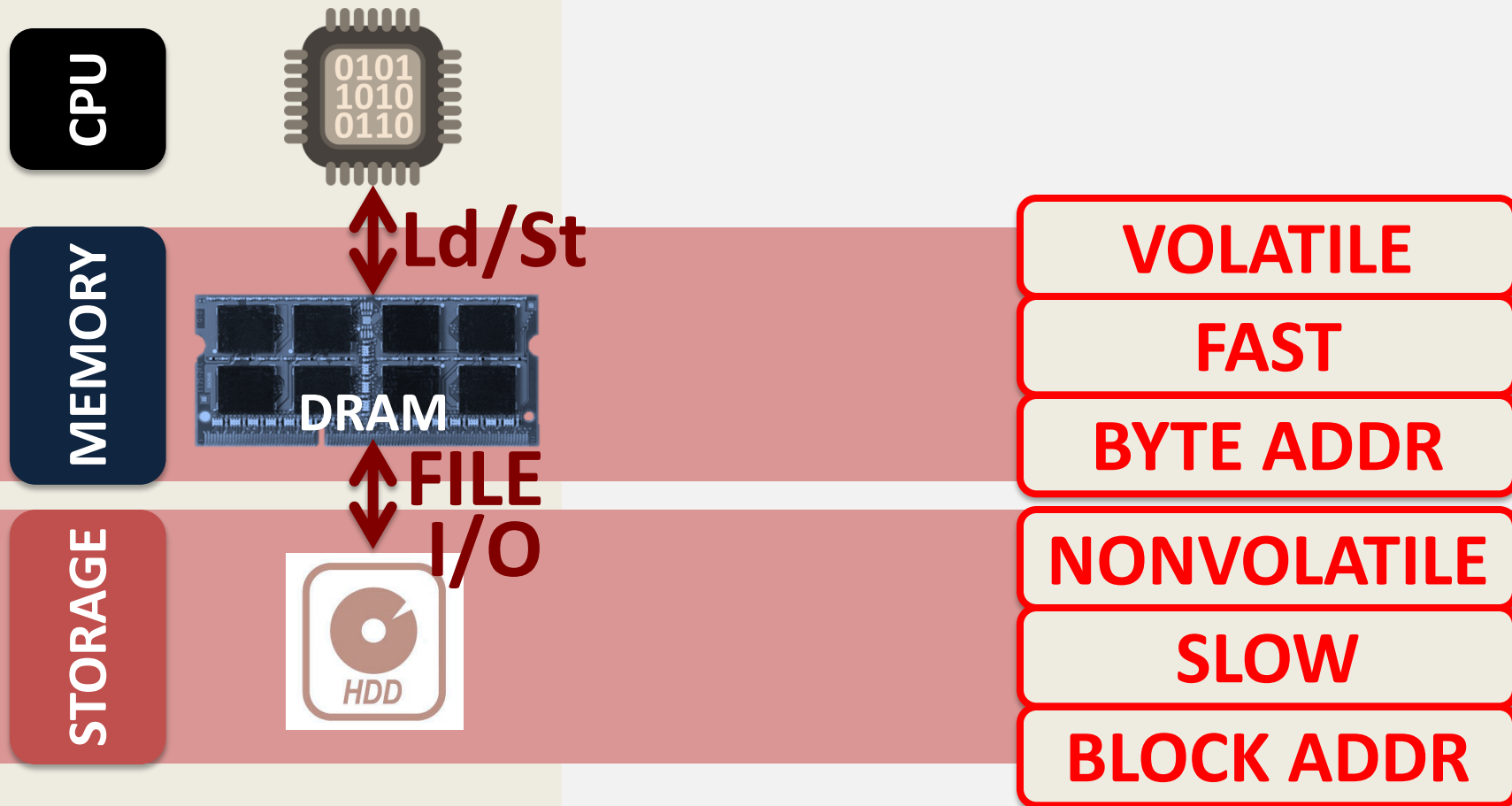
# Other Opportunities with Emerging Technologies

---

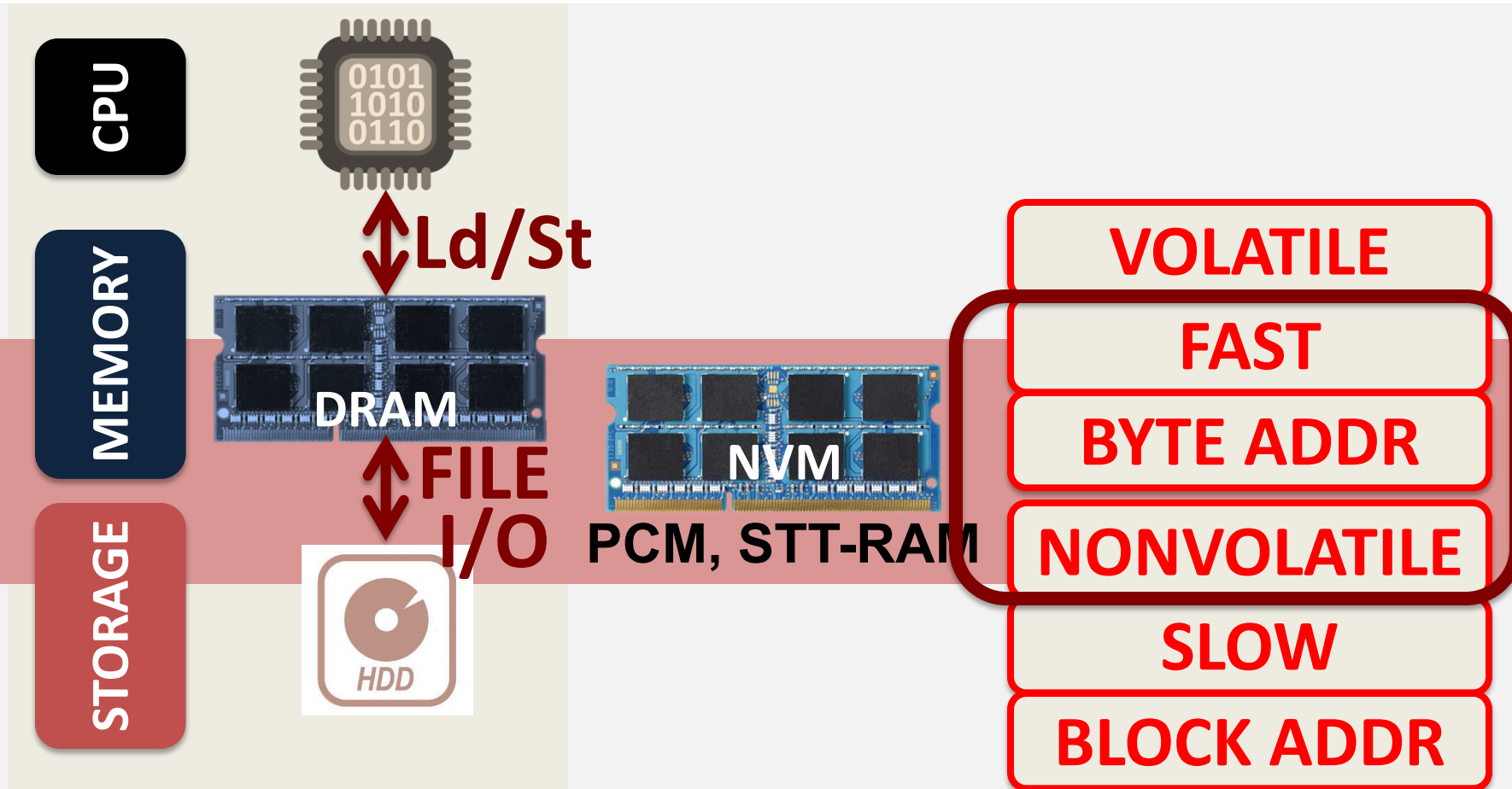
- Merging of memory and storage
  - e.g., a single interface to manage all data
- New applications
  - e.g., ultra-fast checkpoint and restore
- More robust system design
  - e.g., reducing data loss
- Processing tightly-coupled with memory
  - e.g., enabling efficient search and filtering



# TWO-LEVEL STORAGE MODEL



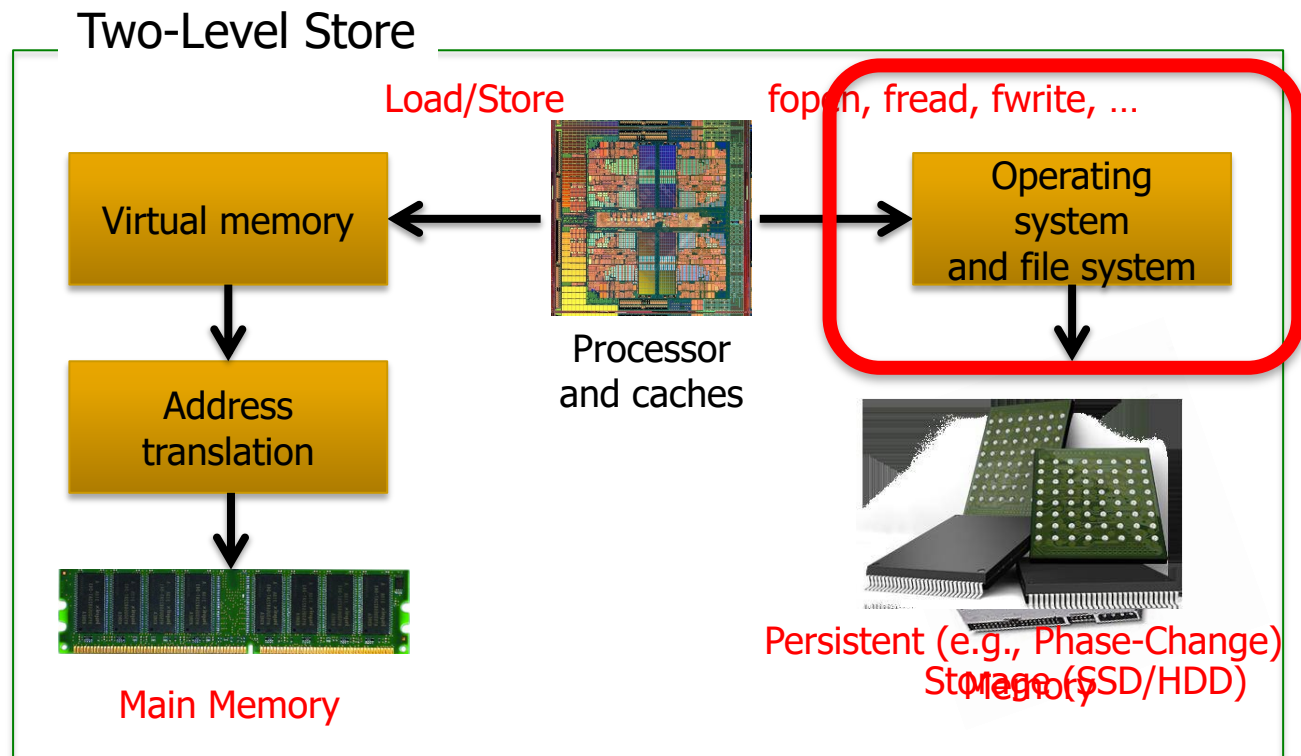
# TWO-LEVEL STORAGE MODEL



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

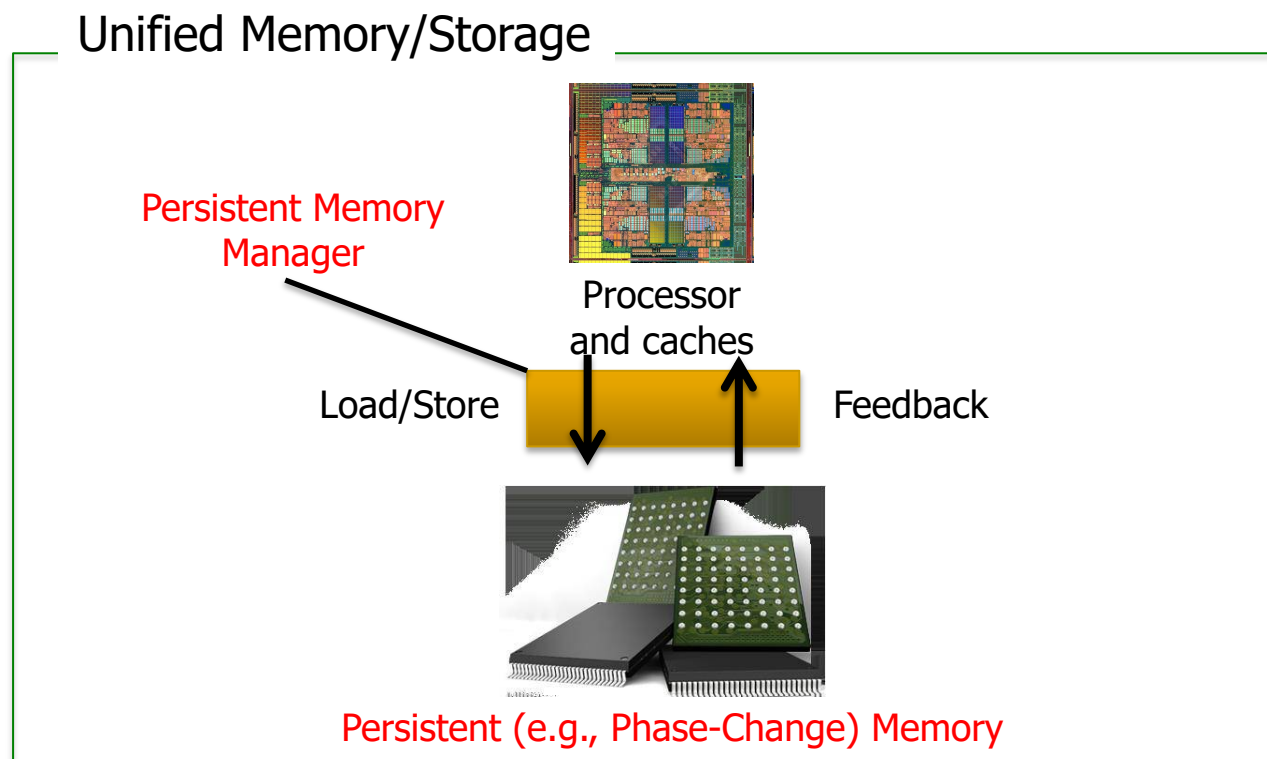
# Two-Level Memory/Storage Model

- 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

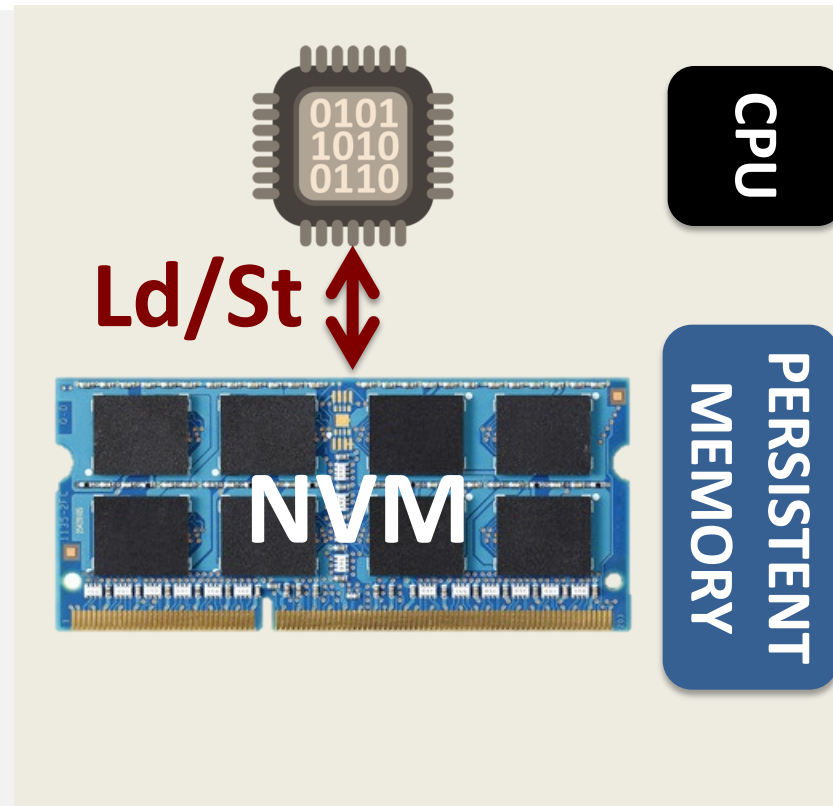


# Unified Memory and Storage with NVM

- 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



# PERSISTENT MEMORY

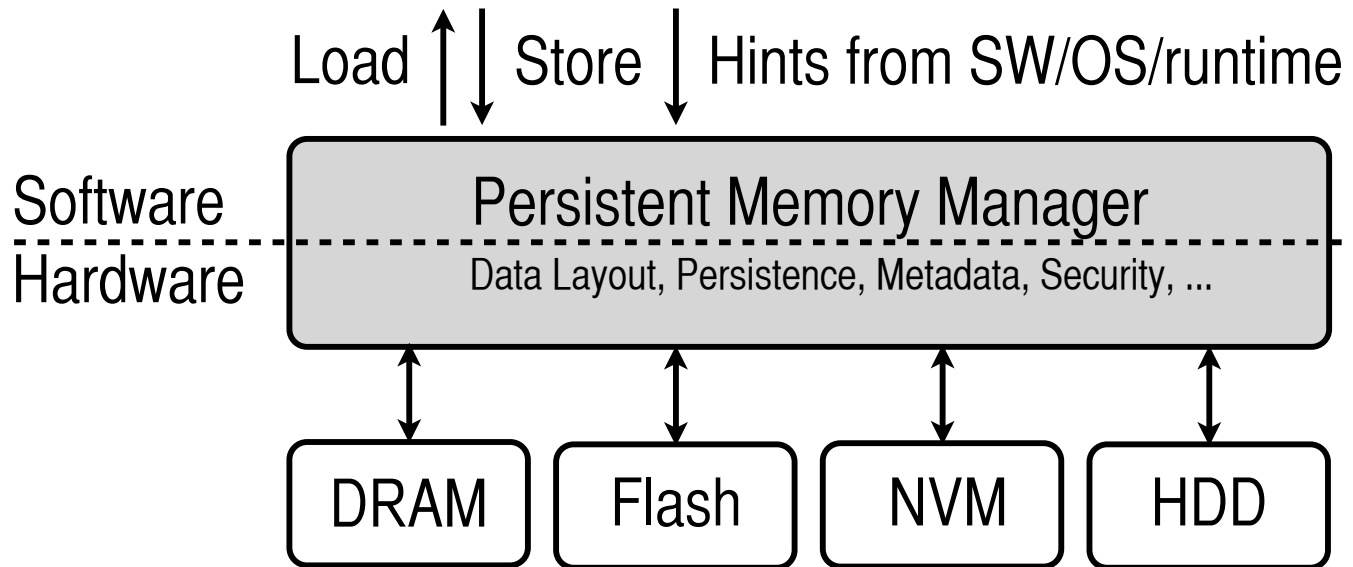


**Provides an opportunity to manipulate persistent data directly**

# 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

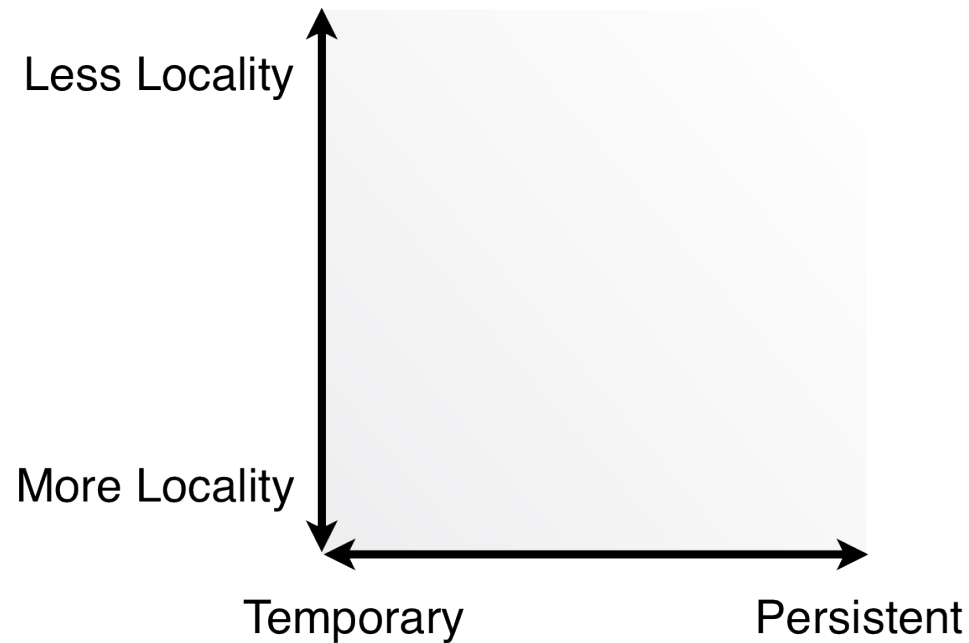
---

- 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



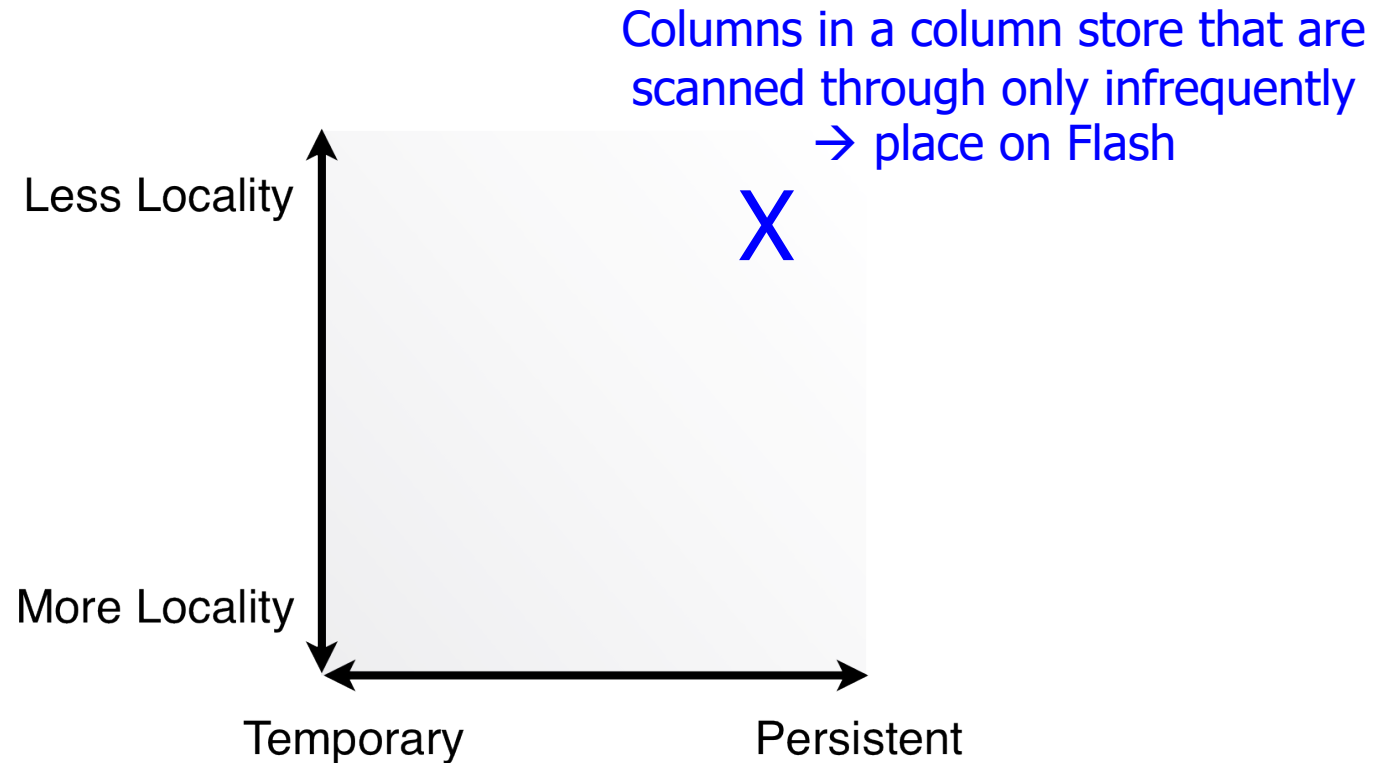
# Efficient Data Mapping among Heterogeneous Devices

---



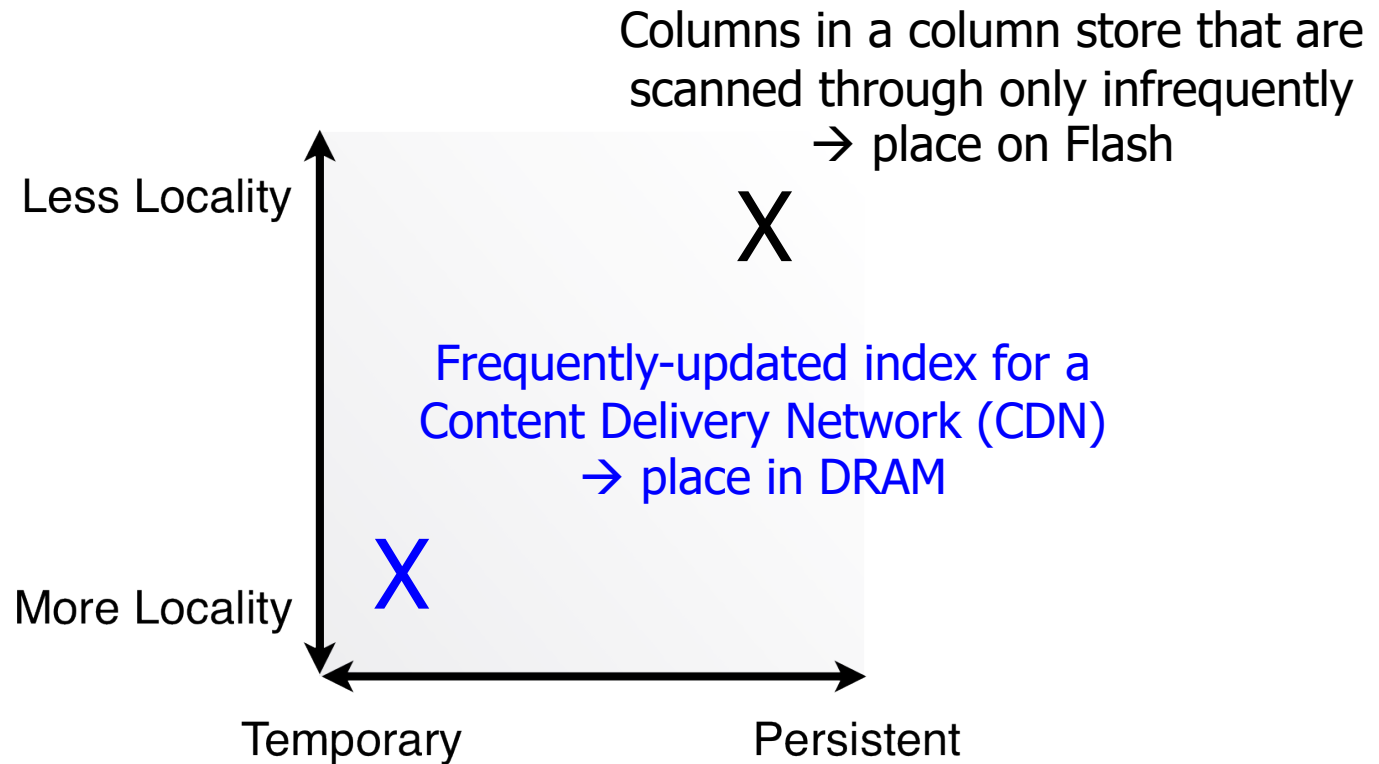
# Efficient Data Mapping among Heterogeneous Devices

---



# 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

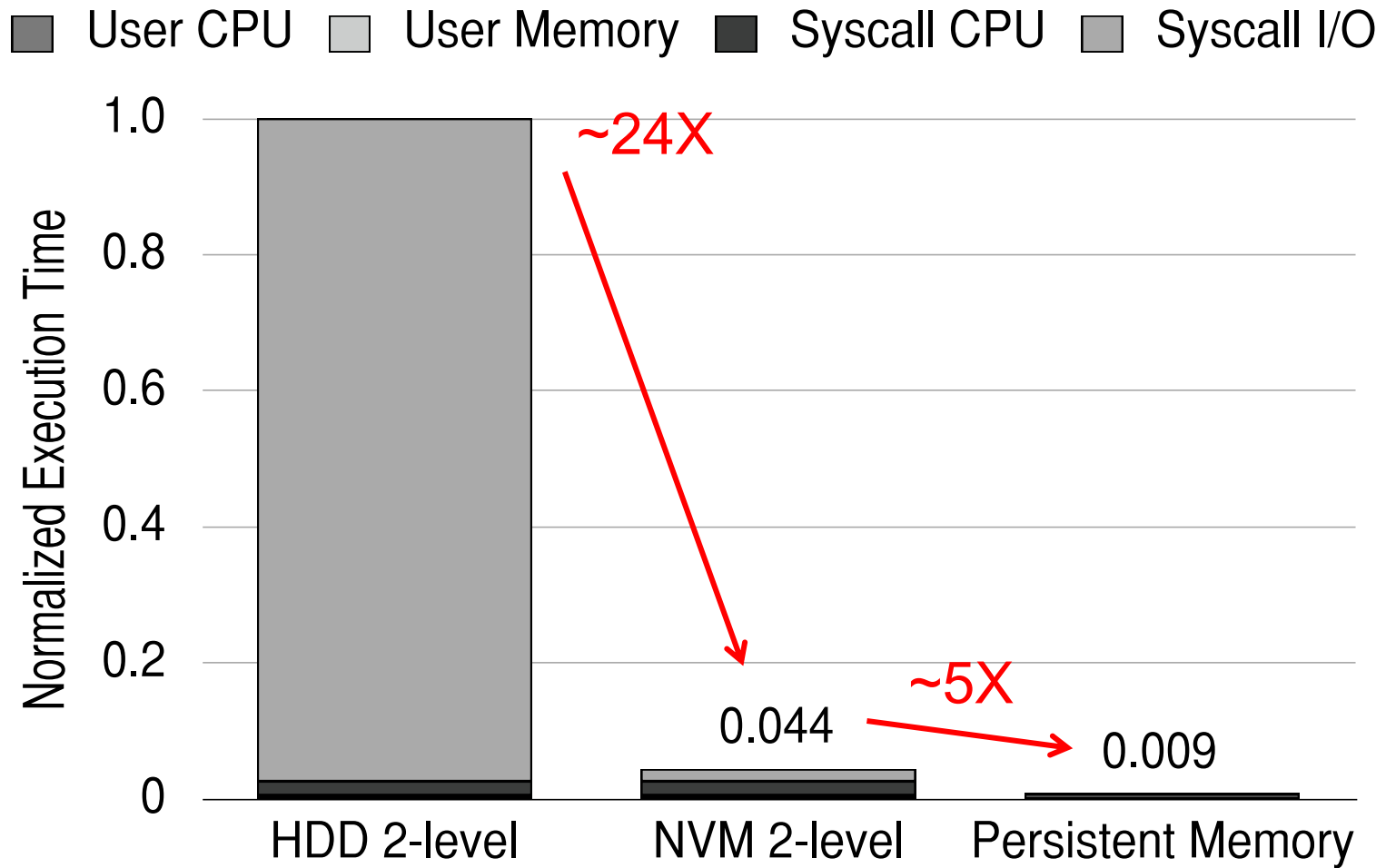
## ■ NVM Baseline (NB)

- ❑ Same as HDD Baseline, but HDD is replaced with NVM
- ❑ Still has OS/FS overheads of the two-level storage model

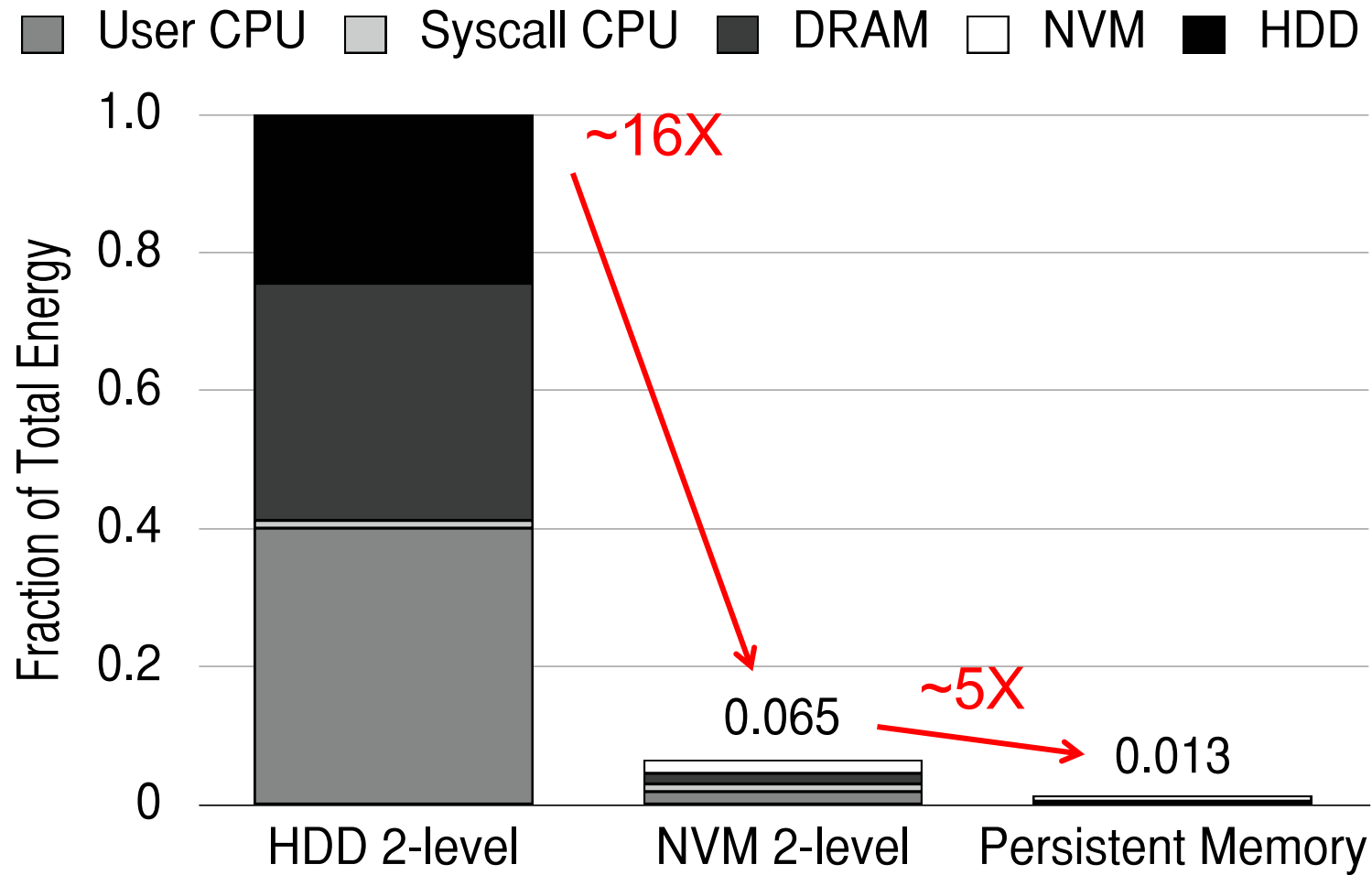
## ■ Persistent Memory (PM)

- ❑ 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



# Energy Benefits of a Single-Level Store



# 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\*<sup>‡</sup>   Jishen Zhao<sup>†</sup>   Yuan Xie<sup>†</sup><sup>§</sup>   Onur Mutlu\*  
\*Carnegie Mellon University   <sup>†</sup>Pennsylvania State University   <sup>‡</sup>Intel Labs   <sup>§</sup>AMD Research

## Combined Memory & Storage



## A Unified Interface to **All Data**

# Intel Optane Persistent Memory (2019)

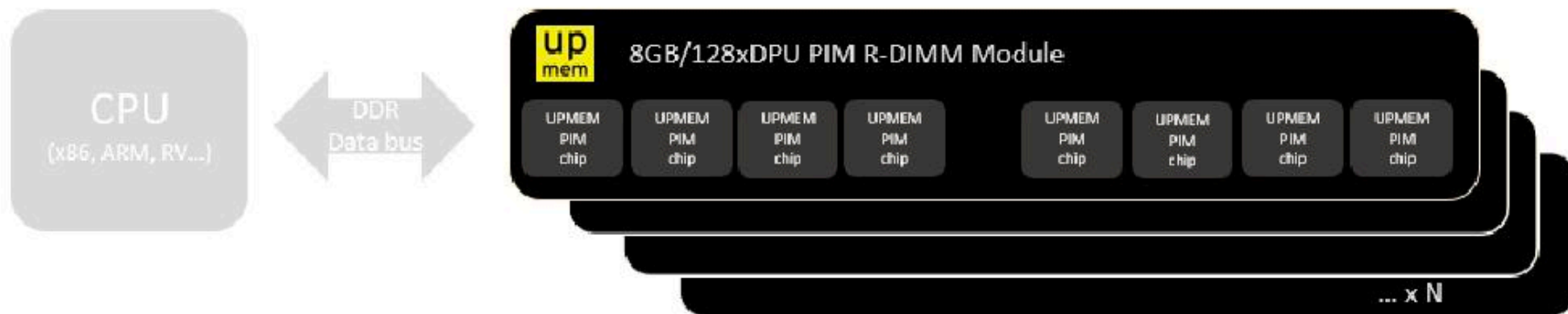
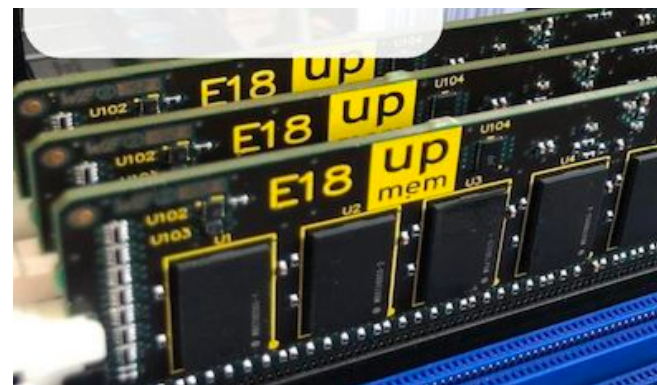
---

- 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



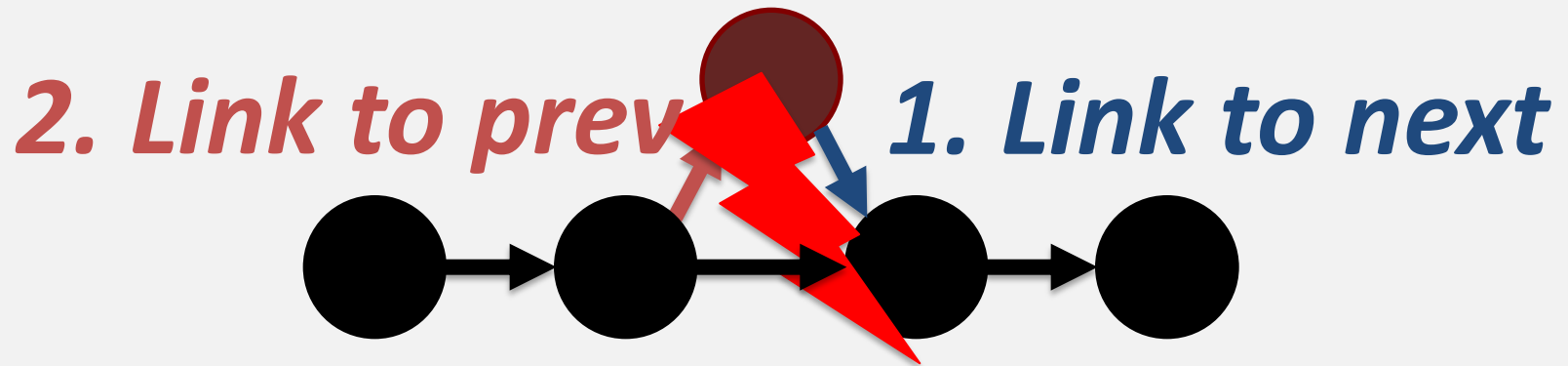
# One Key Challenge in Persistent Memory

---

- 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**



**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(TMARCDECL
```

```
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(TMARCDECL
```

```
hashtable_t* ht, void* key,  
void*data){
```

```
    list_t* chain = get_chain(ht, key);
```

**Need a new implementation**

```
    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(TMARCDECL
```

```
hashtable_t* ht, void* key,  
void* data){
```

```
    list_t* chain = get_chain(ht, key);
```

**Need a new implementation**

```
    pair_t* pair;  
    pair_t updatePair;  
    updatePair.first = key;
```

```
    pair = (pair_t*) TMLIST_FIND(chain,
```

```
                                &updatePair);  
    pair->second = data;
```

**Third party code  
can be inconsistent**

```
}
```

# CURRENT SOLUTIONS

## Manual declaration of persistent components

```
void TMhashtable_update(TMARCDECL
```

```
hashtable_t* ht, void* key,  
void* data){
```

```
list_t* chain = get_chain(ht, key);
```

**Need a new implementation**

```
pair_t* pair;  
pair_t updatePair;  
updatePair.first = key;  
pair = (pair_t*) TMLIST_FIND(chain,  
pair->second
```

**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**

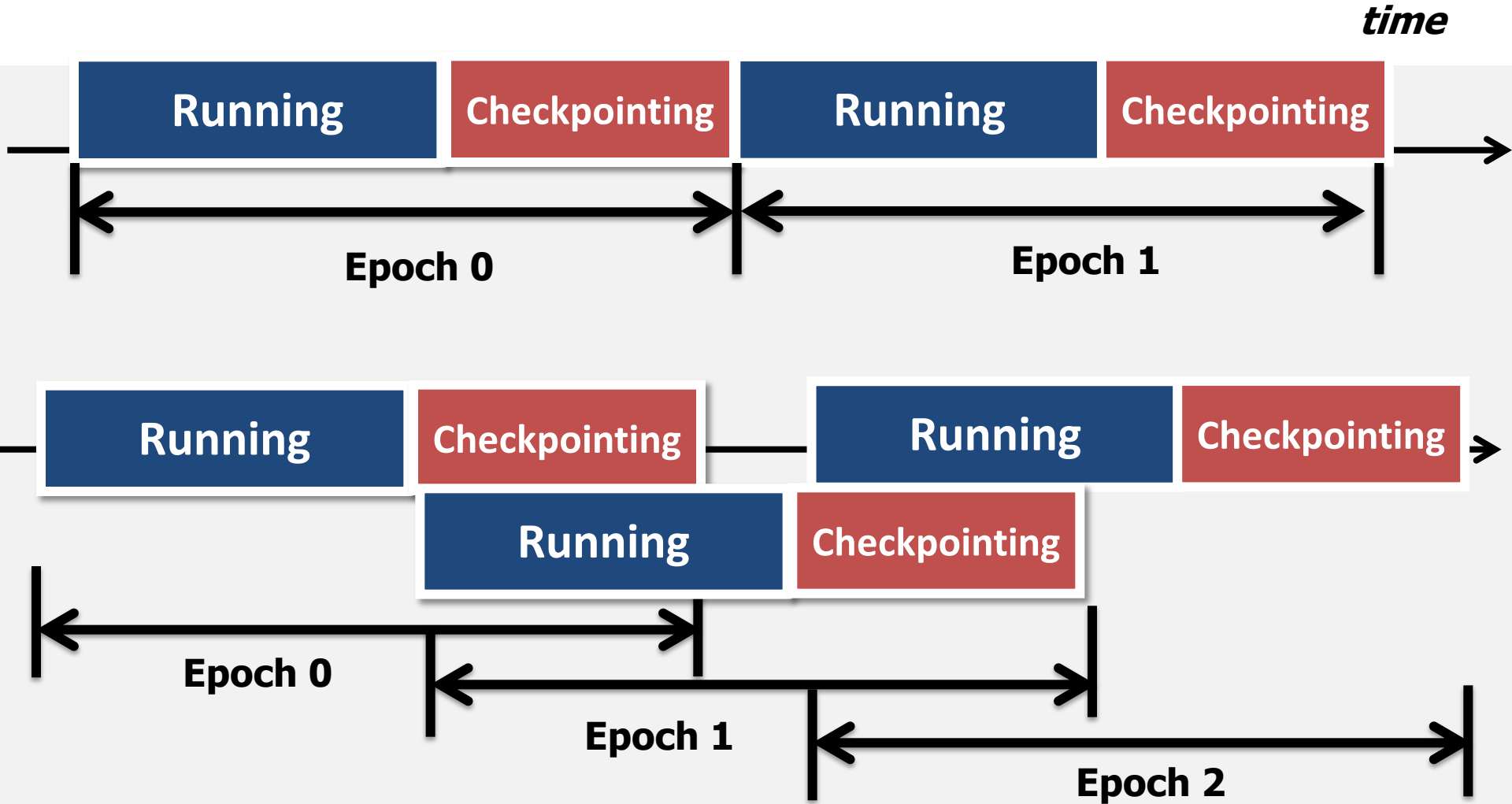
# ThyNVM: Summary

## 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<sup>\*†</sup> Jishen Zhao<sup>‡</sup> Samira Khan<sup>†'</sup> Jongmoo Choi<sup>+†</sup> Yongwei Wu<sup>\*</sup> Onur Mutlu<sup>†</sup>

<sup>†</sup>Carnegie Mellon University    <sup>\*</sup>Tsinghua University

<sup>‡</sup>University of California, Santa Cruz    <sup>'</sup>University of Virginia    <sup>+</sup>Dankook University



## 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

# Conclusion

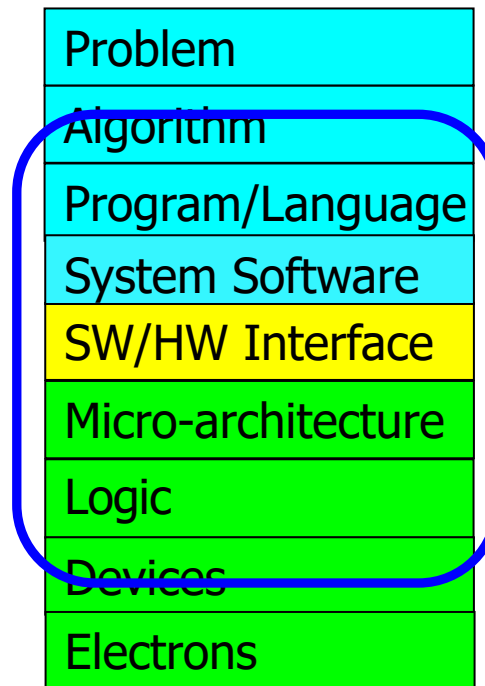
# The Future of Emerging Technologies is Bright

---

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

Can enable:

- Orders of magnitude improvements
- New applications and computing systems



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

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

# Many Research & Design Opportunities

---

- Enabling completely persistent memory
- 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 → virtual block interface



# Computer Architecture

## Lecture 16a: Opportunities and Challenges of Emerging Memory Tech.

Prof. Onur Mutlu

ETH Zürich

Fall 2020

19 November 2020