

# Persistent Memory Architecture Research at UCSC – Workload Characterization and Hardware Support for Persistence

Jishen Zhao

[jishen.zhao@ucsc.edu](mailto:jishen.zhao@ucsc.edu)

Computer Engineering  
UC Santa Cruz

July 12, 2016



UNIVERSITY OF CALIFORNIA  
SANTA CRUZ

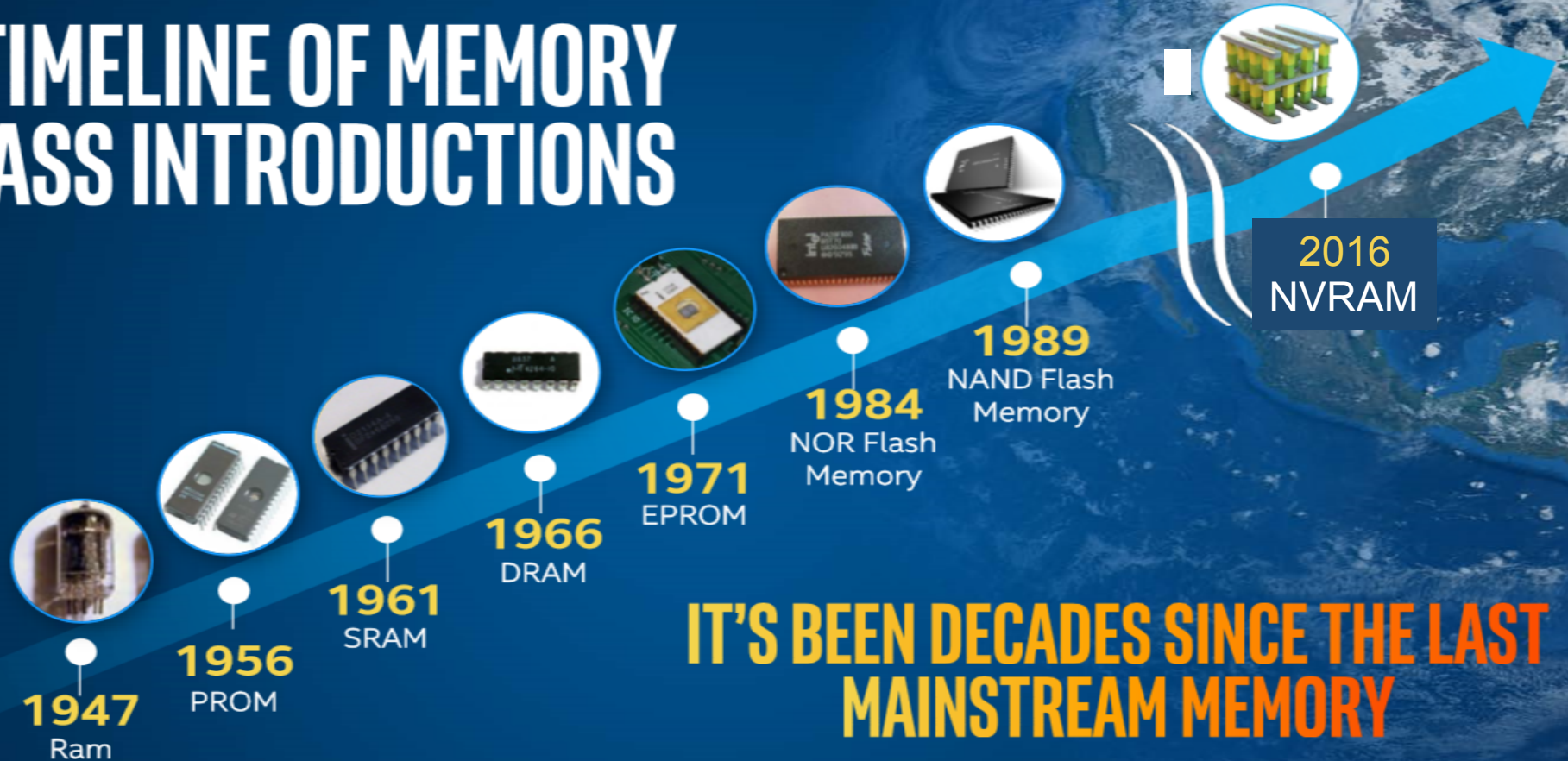
# What is persistent memory?

- **Persistent memory** = <sup>NVRAM</sup> memory + storage

# NVRAM is here...

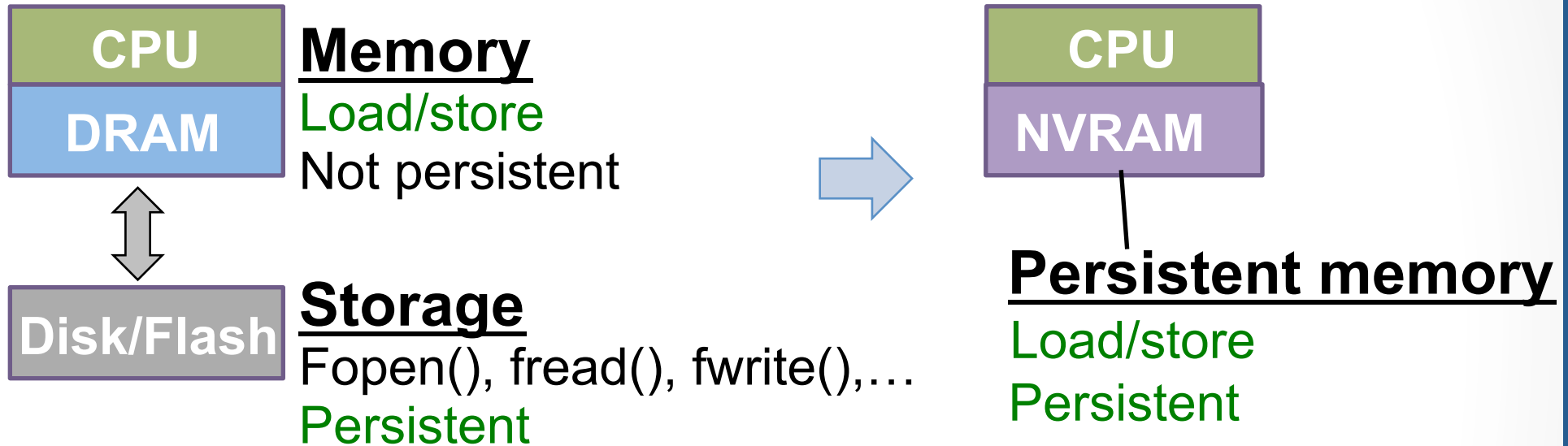
*STT-RAM, PCM, ReRAM, NVDIMM, 3D Xpoint, etc.*

## A TIMELINE OF MEMORY CLASS INTRODUCTIONS



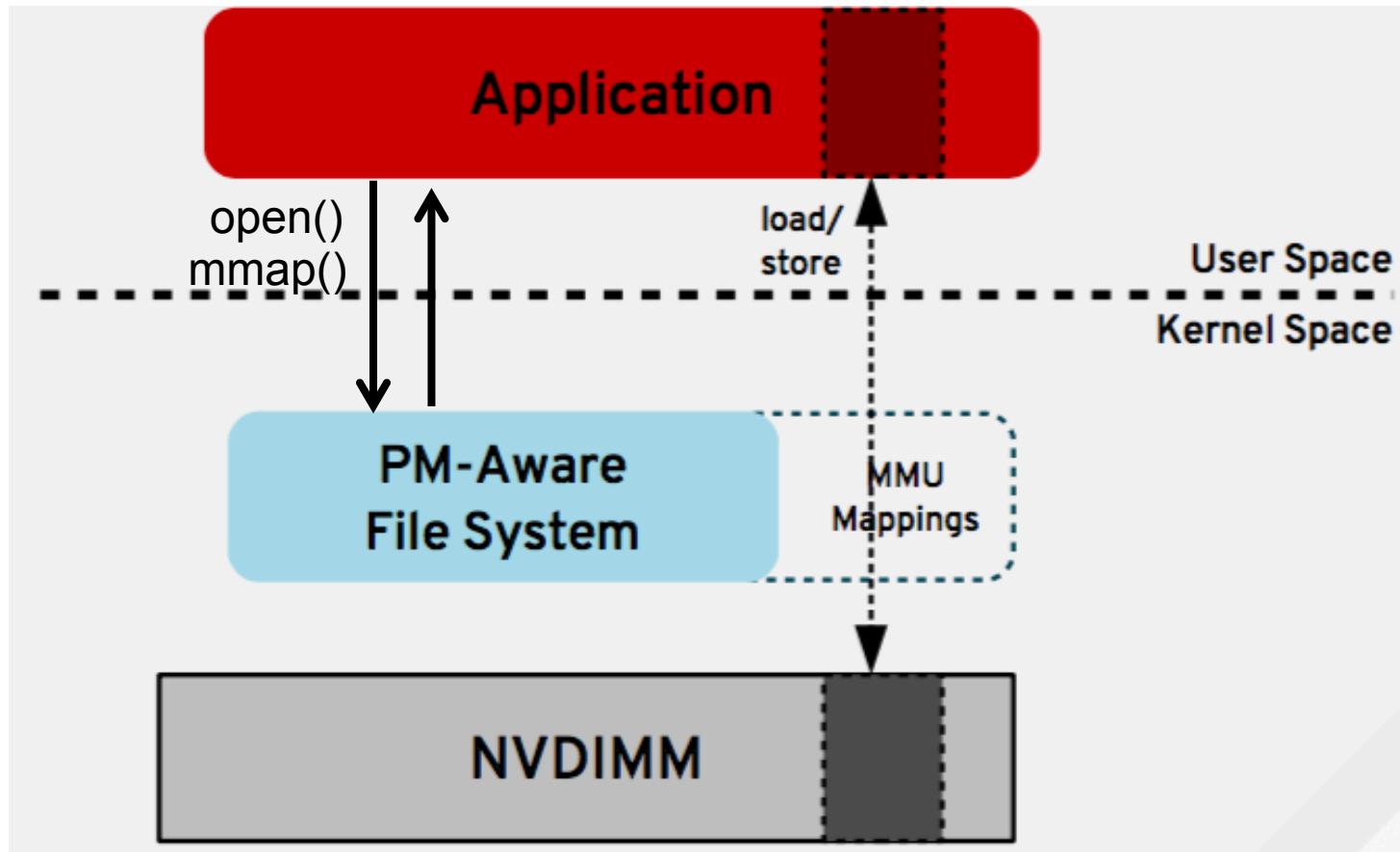
**IT'S BEEN DECADES SINCE THE LAST MAINSTREAM MEMORY**

# Design Opportunities with NVRAM



- Allow in-memory data structures to become permanent immediately
- Demonstrated 32x speedup compared with using storage devices [Condit+ SOSP'09, Volos+ ASPLOS'11, Coburn+ ASPLOS'11, Venkataraman+ FAST'11]

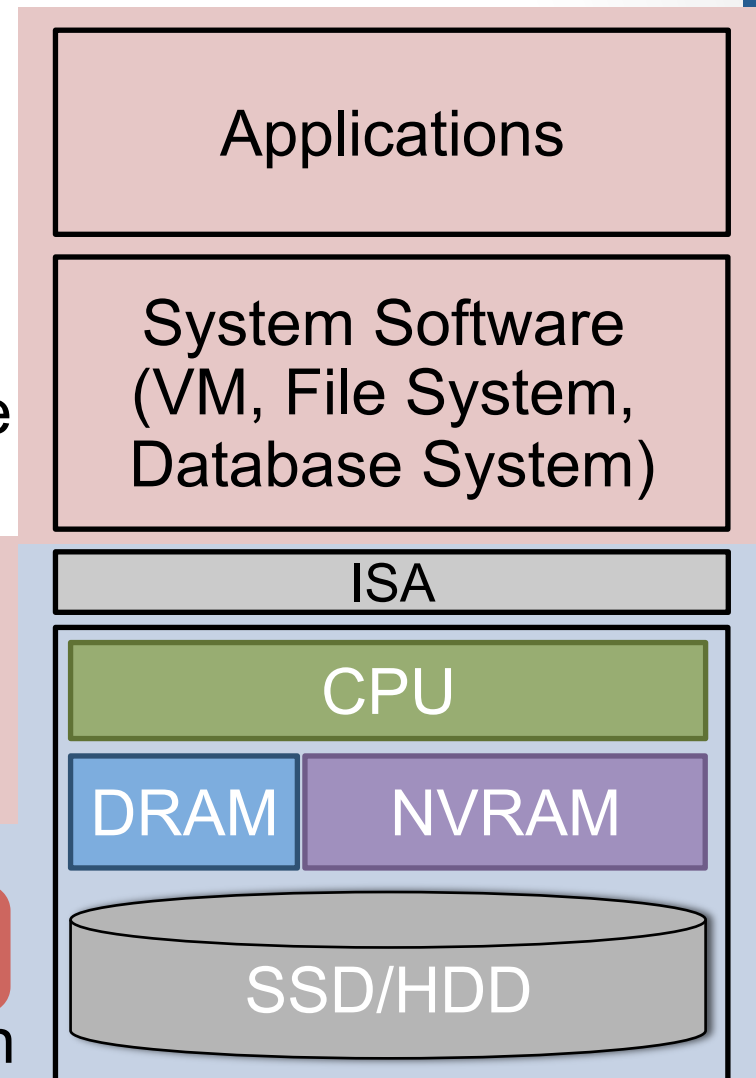
# Executing Applications in Persistent Memory



Jeff Moyer, "Persistent memory in Linux," SNIA NVM Summit, 2016.

# Our research – At the software/hardware boundary

- Workload characterization
  - Exploring persistent memory use cases
  - Identifying system bottlenecks
  - Implications to software/hardware design
- System software
  - Efficient fault tolerance and data persistence mechanisms
- Hardware
  - Developing storage accelerators
  - Redefining the boundary between software and hardware



# Workload Characterization from a hardware perspective

- Motivation
  - Persistent **memory** is managed by both hardware and software
  - Most prior works only profile software statistics, e.g., system throughput
- Objectives
  - Help system designers better understand performance bottlenecks
  - Help application designers better utilize persistent memory hardware
- Approach
  - Profile hardware and software counter statistics
  - Instrument application and system software to obtain insights at micro-architecture level

# Hardware and software configurations

- **CPU**: Intel Xeon CPU E5-2620 v3
- **Memory**: 12GB of pmem + 4GB of main memory partitioned on DRAM (memmap)
- **Operating system**: Linux 4.4.0 kernel
- **Profiling Tools**
  - Linux Perf: collecting software and hardware counter statistics
  - Intel Pin 3.0 instrumentation tool with in-house Pintools
- **File systems evaluated**
  - Ext4 : Journaling of metadata, running on RAMDisk
  - Ext4-**DAX**
    - Journaling of metadata and bypass page cache with DAX
  - NOVA
    - Nonvolatile accelerated log-structured file system [Li+ FAST'16]



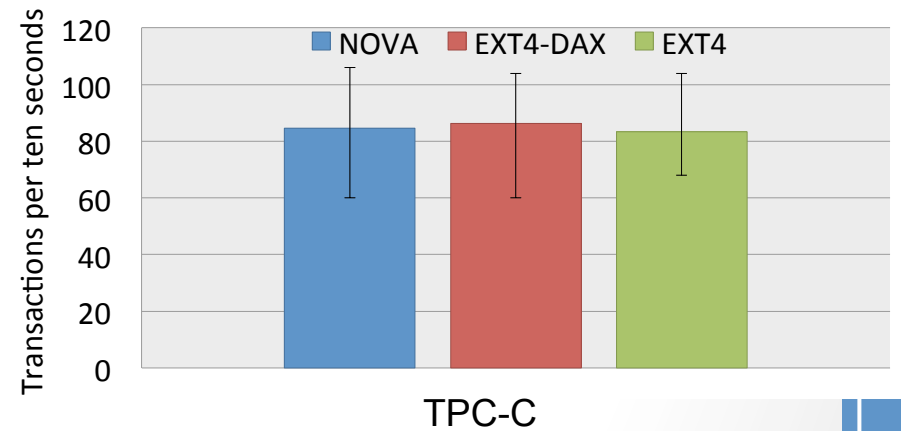
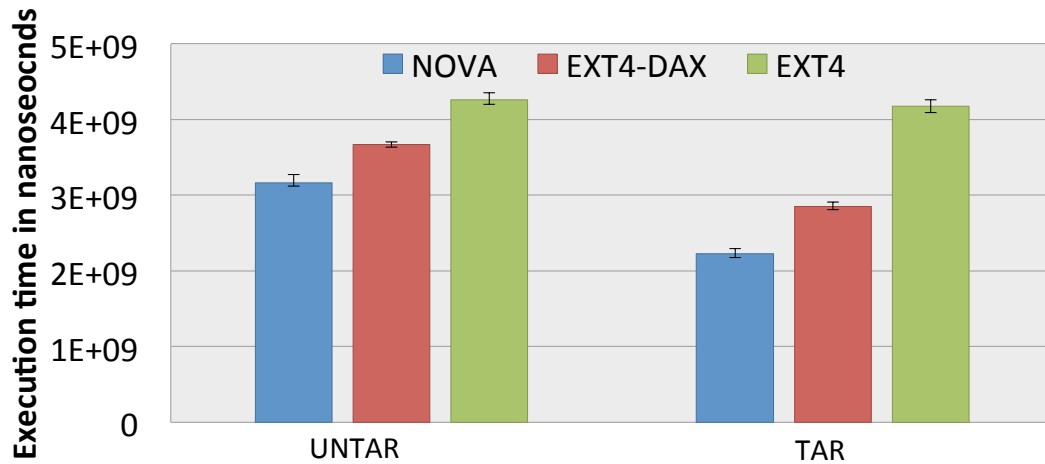
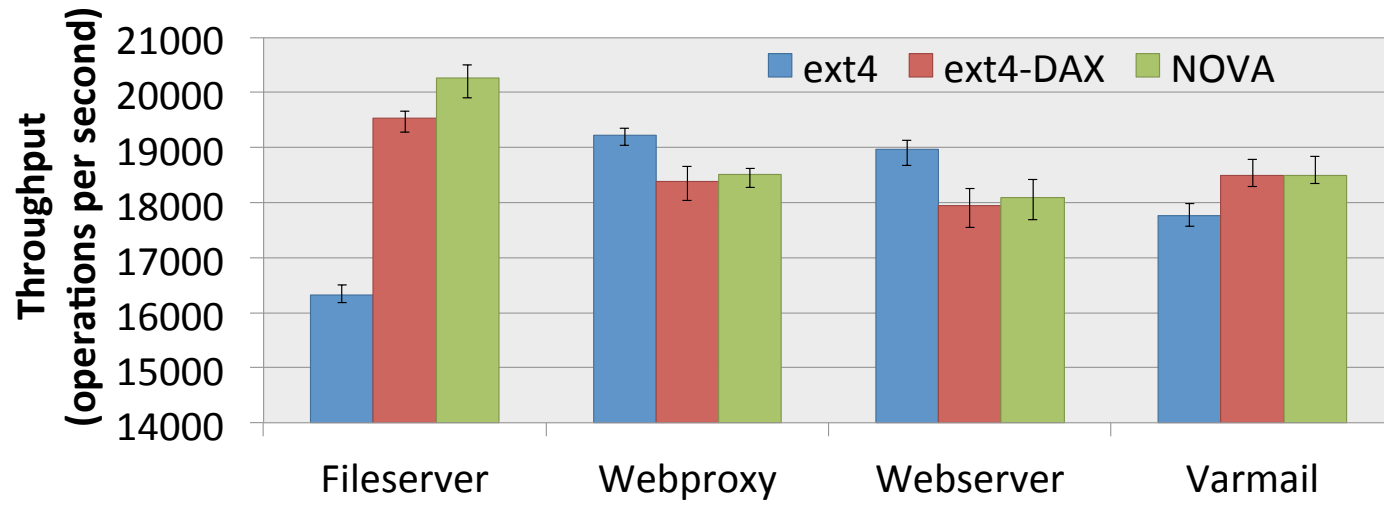
# About DAX

- What is DAX?
  - “Direct Access”
  - Enabling efficient Linux support for persistent memory
  - Allowing file system requests to bypass the page cache allocated in DRAM and directly access NVRAM via loads and stores
- How does Ext4-DAX work?
  - DAX maps storage components directly into userspace
  - \* *True DAX is not supported in Linux yet – accesses still go through DRAM, i.e., directly swaps the pages between DRAM main memory and NVRAM storage.*
- Example of file systems with DAX capability
  - Ext4-DAX, XFS-DAX, Btrfs-DAX → Fedora
  - Intel PMFS
  - NOVA

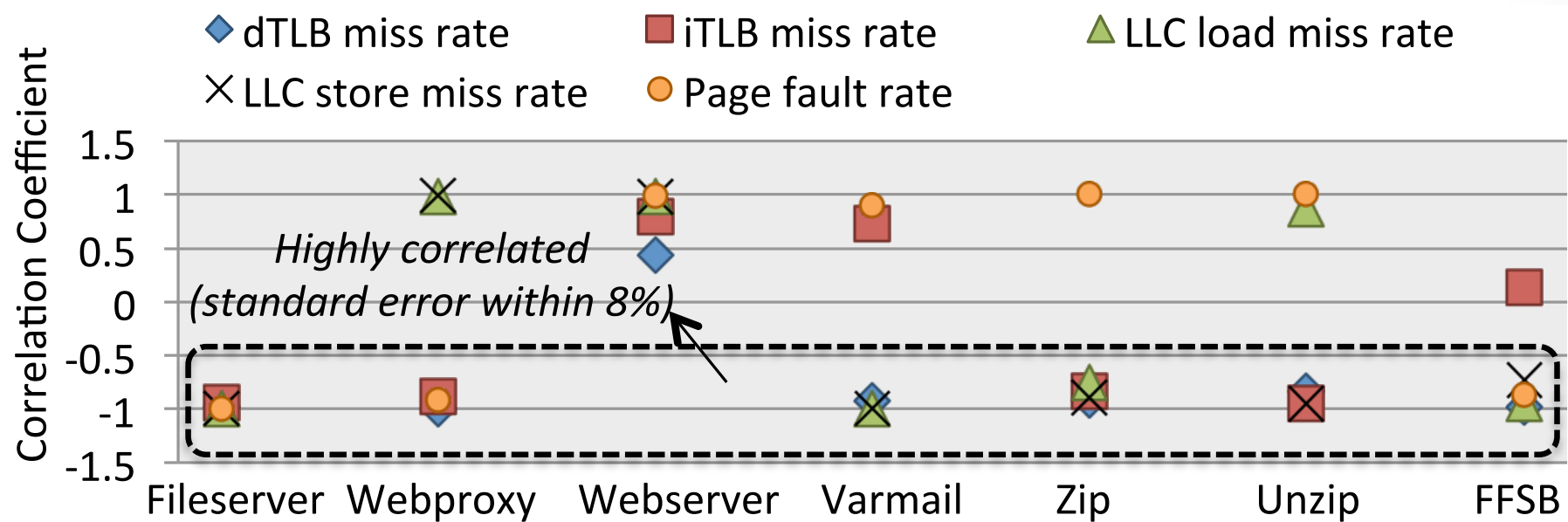
# Current workloads

- Filebench (a widely-used benchmark suite designed for evaluating file system performance)
  - Fileserver, Webproxy, WebServer, Varmail
- **FFSB** (Flexible Filesystem benchmark)
  - Can configure read/write ratio and number of threads
- **Bonnie**
  - measuring file system performance by invoking `putc()` and `getc()`
- File compression/decompression: tar/untar, zip/unzip
- TPC-C running with MySQL
  - A database online transaction processing workload
  - Write intensive, with 63.7% of writes
- In-house micro-benchmarks
- \* *Applications are compiled with **static linking** and stored in NVRAM (pmem) region*

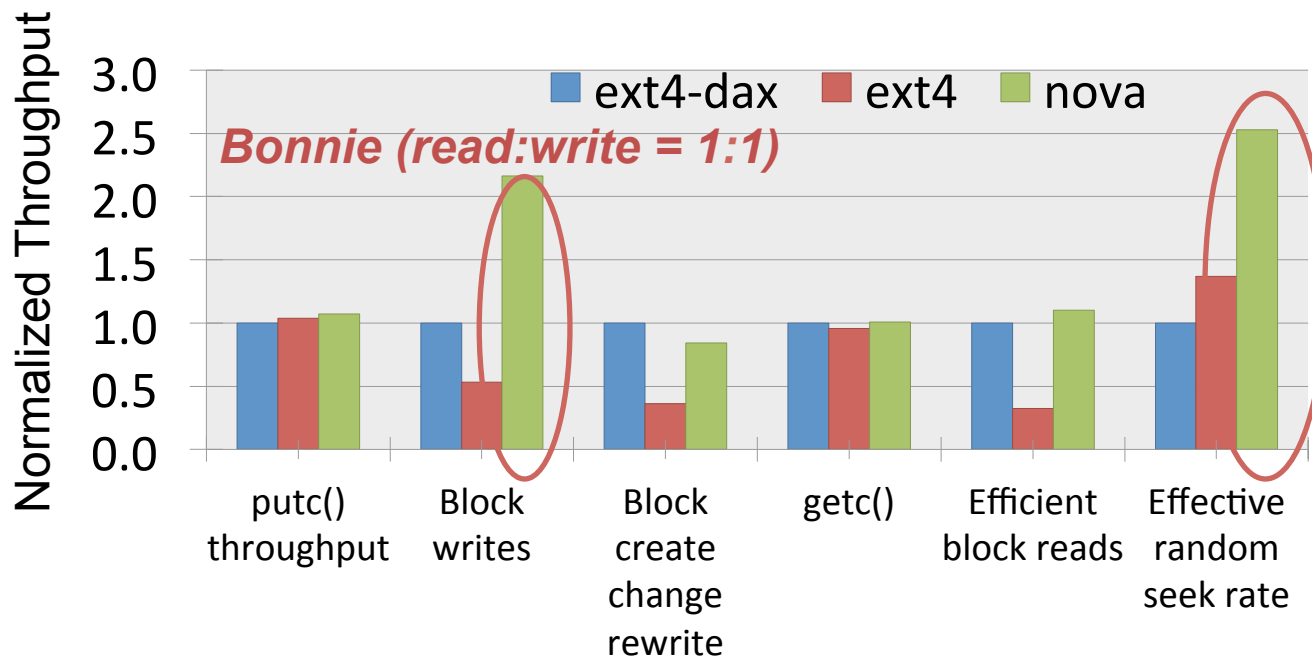
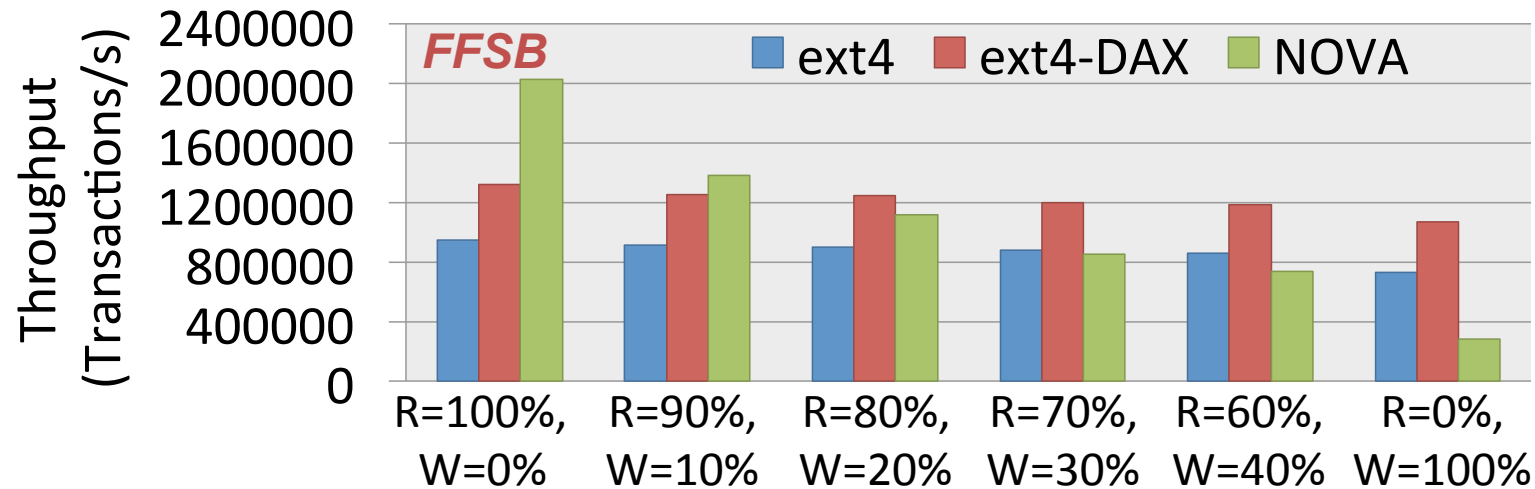
# Workload throughput



# Correlation between system performance and hardware behavior

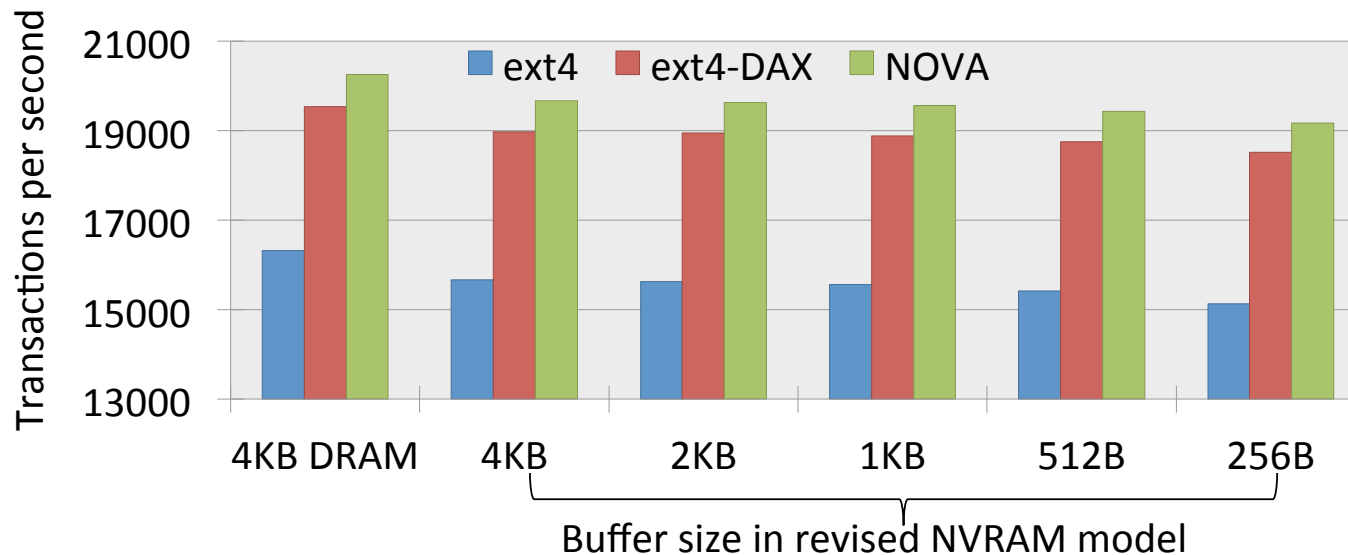
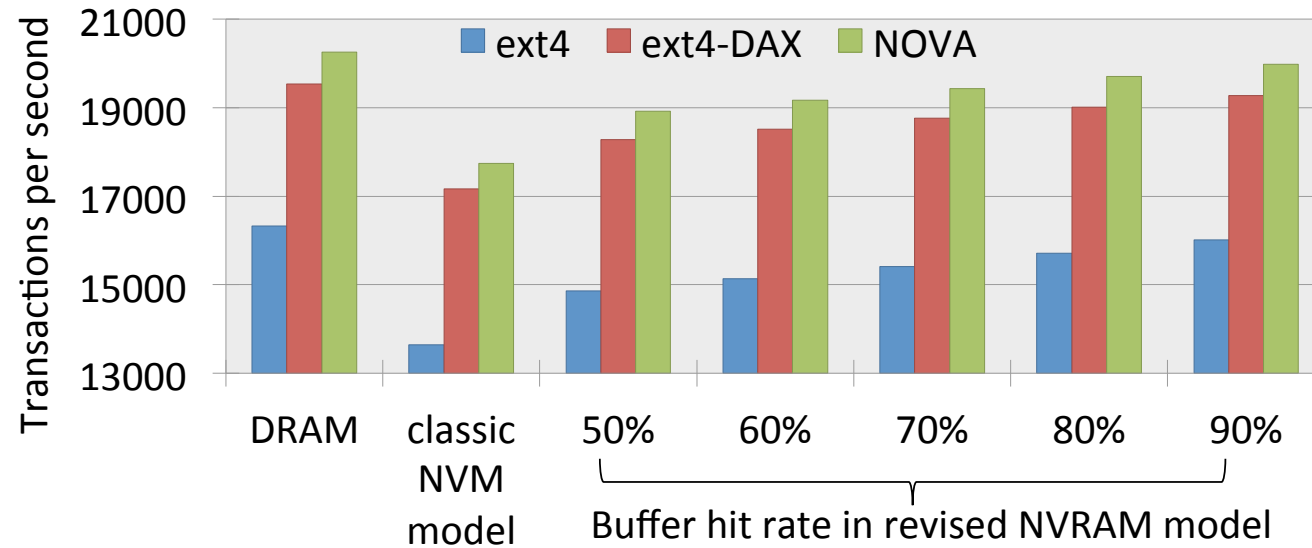


# Throughput vs. Write Intensity



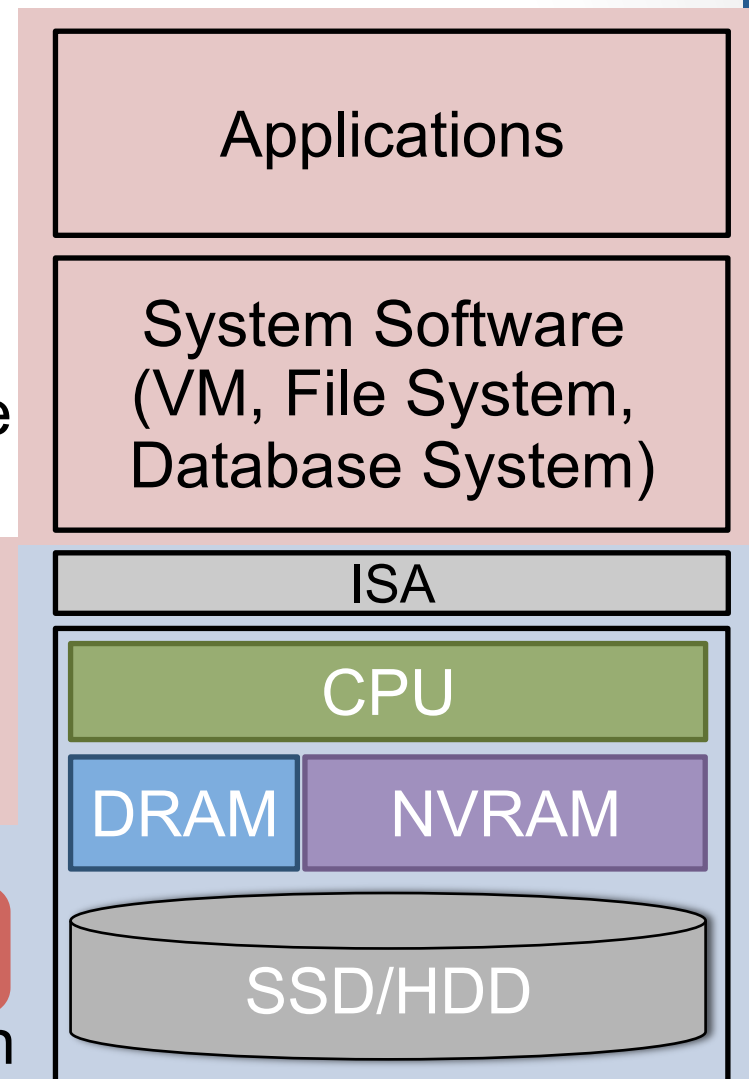
# The impact of workload locality

- NVRAM devices may or may not have an on-chip buffer



# Our research – At the software/hardware boundary

- Workload characterization
  - Exploring persistent memory use cases
  - Identifying system bottlenecks
  - Implications to software/hardware design
- System software
  - Efficient fault tolerance and data persistence mechanisms
- Hardware
  - Developing storage accelerators
  - Redefining the boundary between software and hardware

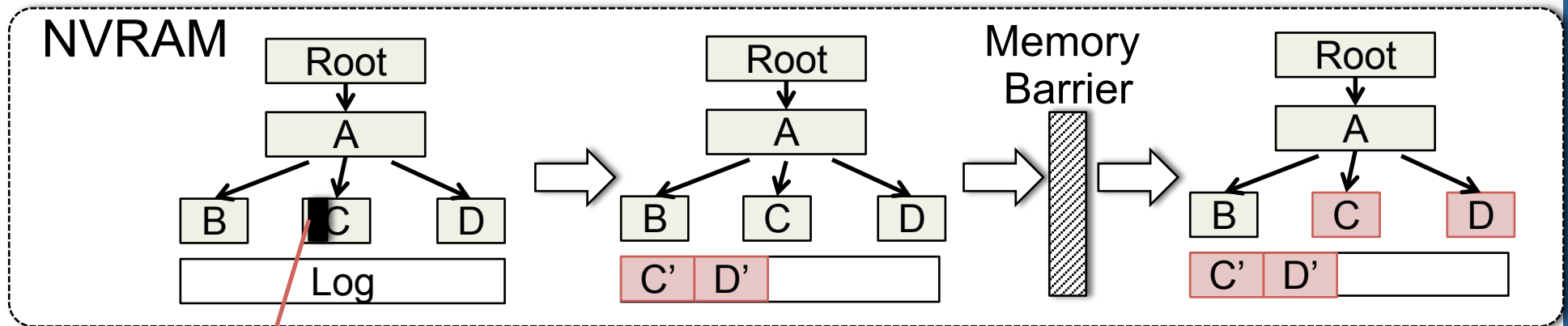


# Logging Acceleration (executive summary)

- **Problem**
  - Traditional software-based logging imposes substantial overhead in persistent memory
    - Even with either undo or redo logging
    - Not to say undo+redo logging as used in many modern database systems
  - Changes in software interface add burden on programmers
- **Solution**
  - Hardware-based logging accelerators
  - Leverage existing hardware information (otherwise largely wasted)
- **Results**
  - 3.3X performance improvement
  - Simplified software interface
  - Low hardware overhead

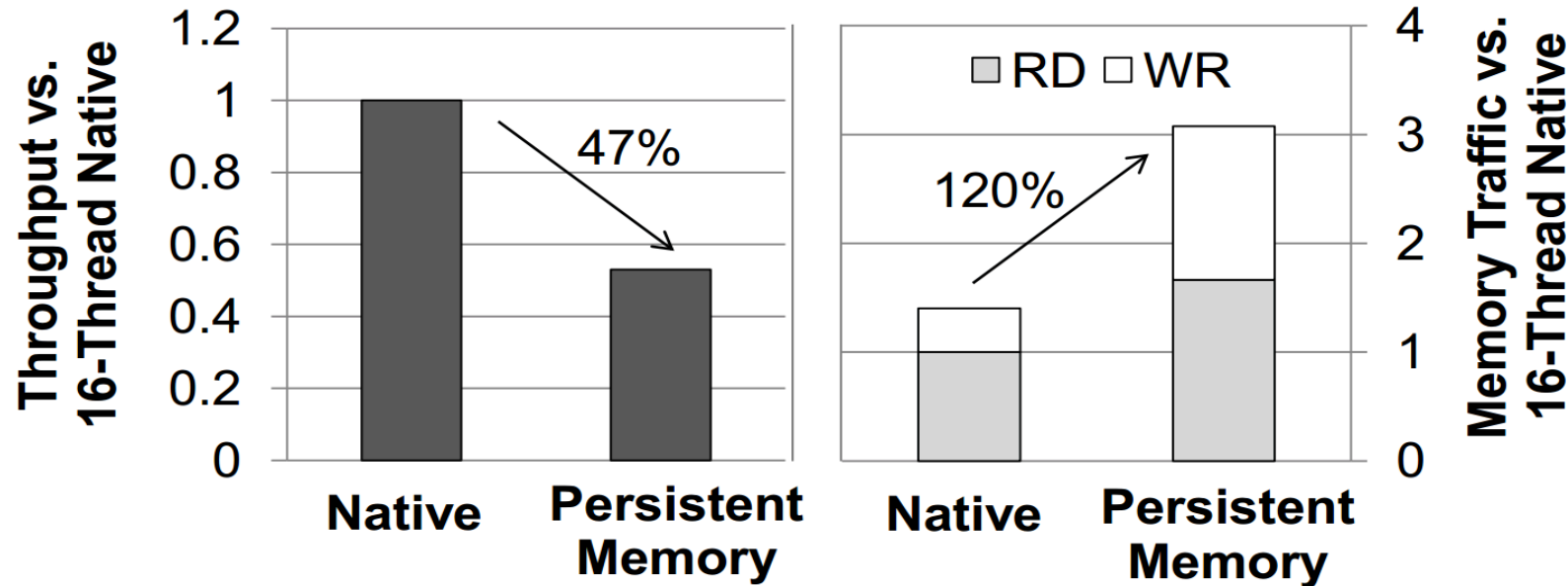


# Logging (Journaling) in Persistent Memory (Maintaining Atomicity)



Size of one store

# Performance overhead of software logging



Zhao+, "Kiln: Closing the performance gap between systems with and without persistence support," MICRO 2013.

# Software interface of software logging

- Memory barriers, strict ordering constraints, and cache flushing all needed for ensuring data persistence


```
1 // Persistent Update
2 tx_begin();
3
4 write_Log( address( A ),
5           old_val( A ), // undo log
6           new_val( A ) ); // redo log
7
8 memfence();
9
10 write( A ); // Update A with new value
11
12 tx_commit();
13
14 // ... Some time later ...
15
16 clwb( address( A ) ); // Write-back A
```

# Our software interface

- ~~Memory barriers, strict ordering constraints, and cache flushing all needed for ensuring data persistence~~

Hardware support for

```
1 // Persistent Update
2 tx_begin();
3
4 write_Log( address( A ),
5           old_val( A ), // undo log
6           new_val( A ) ); // redo log
7
8 memfence();
9
10 write( A ); // Update A with new value
11
12 tx_commit();
13
14 // ... Some time later ...
15
16 clwb( address( A ) ); // Write-back A
```

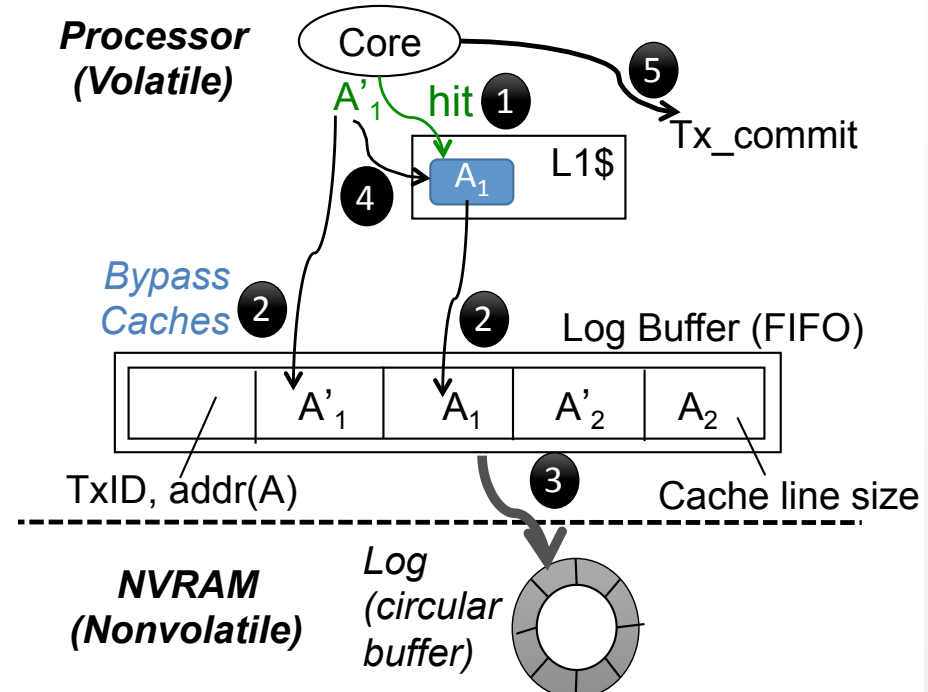
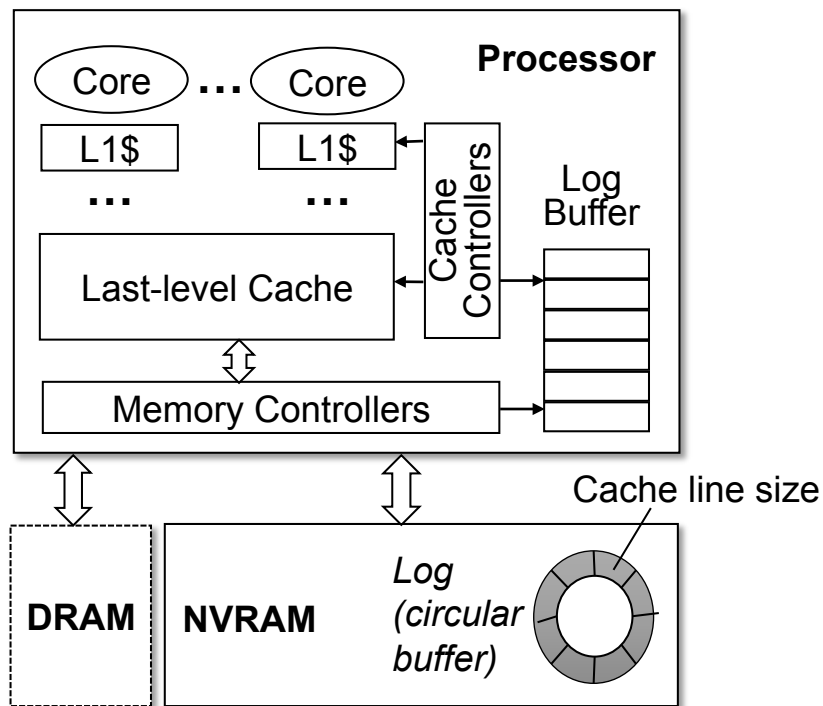


```
1 // Persistent Update
2 tx_begin();
3 write( A ); // Update A with new value
4 tx_commit();
```

# How does it work?

L1 cache hit – we get all that needed for undo+redo log

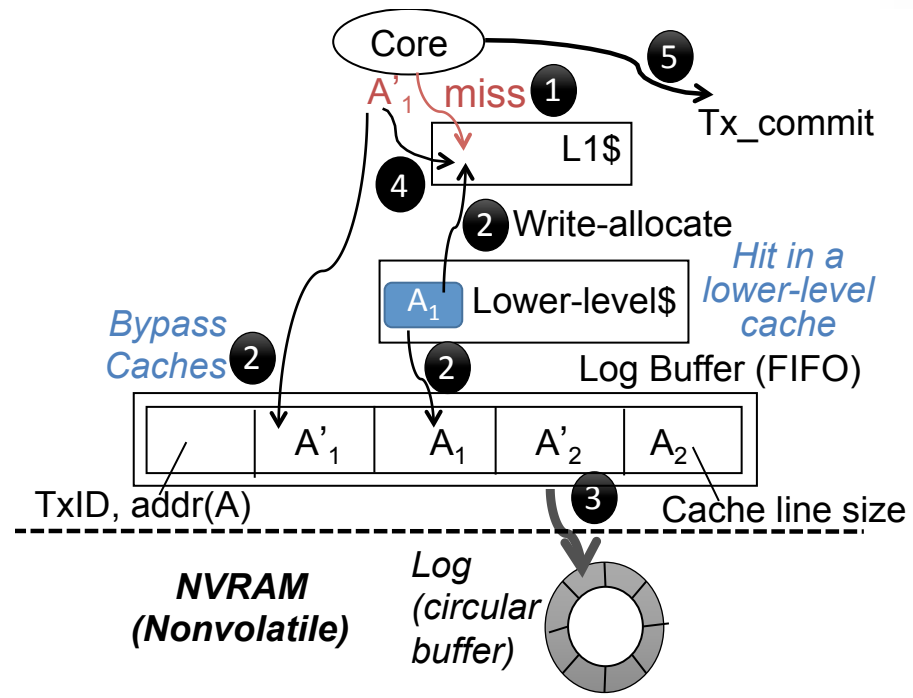
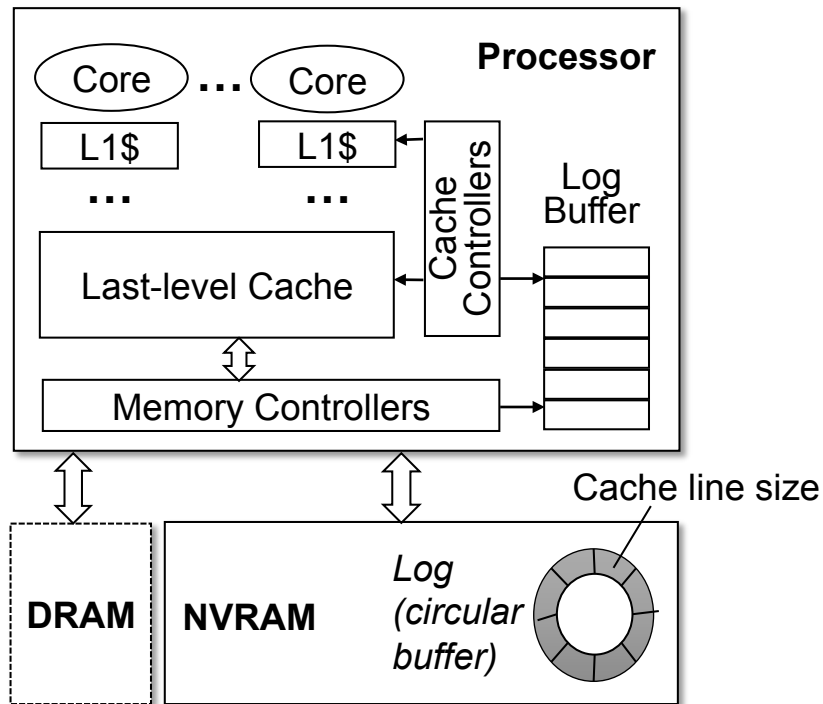
- Writes to persistent memory automatically trigger a write to the log – a software-allocated circular buffer
- Log information includes TxID, address, undo cache line value, and redo cache line value
- Leveraging cache hit/miss handling process to update the log
- Log updates get buffered in the processor



# How does it work?

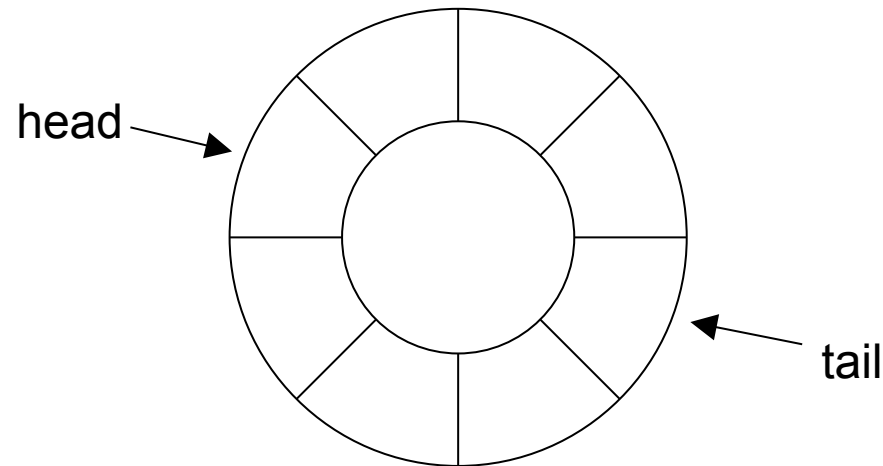
**L1 cache miss** – we get all that needed during “write-allocate”

- Writes to persistent memory automatically trigger a write to the log – a software-allocated circular buffer
- Log information includes TxID, address, undo cache line value, and redo cache line value
- Leveraging cache hit/miss handling process to update the log
- Log updates get buffered in the processor



# Force cache writeback when necessary

- Need to flush CPU caches, when
  - A log entry is almost overwritten by new log updates
  - But the associated data still remains in CPU caches



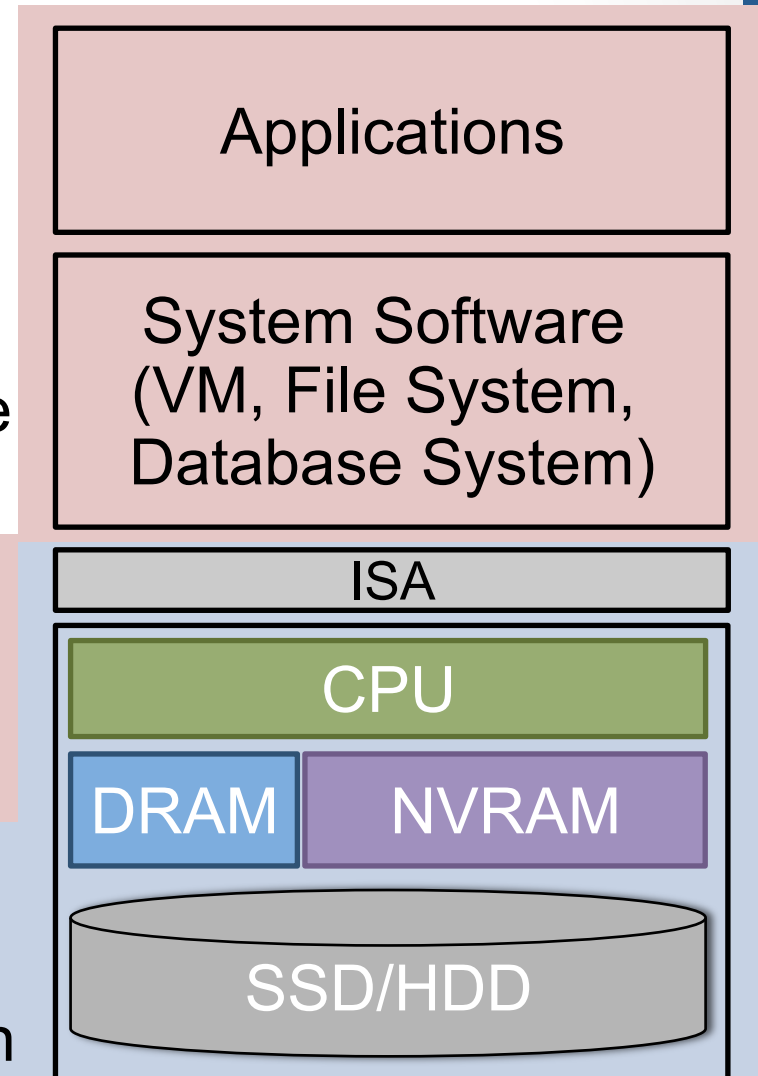
# Results

- McSimA+ simulator running
  - Persistent memory micro-benchmarks
  - A real workload – a persistent version of memcached
- System throughput improved by 1.45x~1.60x on average
- Memcached throughput improved by 3.3x
- Memory traffic reduced by 2.36x~3.12x
- Dynamic memory energy improvement by 1.53x~1.72x
- Hardware overhead
  - 17 bytes of flip-flops
  - 1-bit cache tag information per cache line
  - Multiplexers



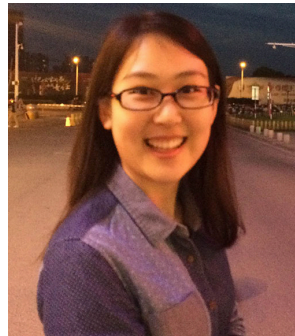
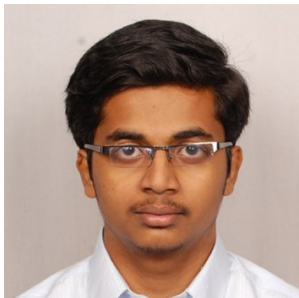
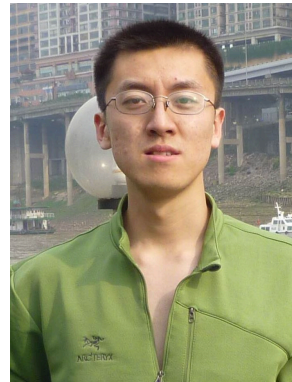
# Summary

- ☑ • Workload characterization
  - Exploring persistent memory use cases
  - Identifying system bottlenecks
  - Implications to software/hardware design
- System software
  - Efficient fault tolerance and data persistence mechanisms
- Hardware
  - ☑ • Developing storage accelerators
  - Redefining the boundary between software and hardware



# UCSC STABLE (SysTem and Architecture lab on scalaBility, reLIability, and Energy-efficiency)

[Email: Jishen.zhao@ucsc.edu](mailto:Jishen.zhao@ucsc.edu)  
<https://users.soe.ucsc.edu/~jzhao/>



# Persistent Memory Architecture Research at UCSC – Workload Characterization and Hardware Support for Persistence

Jishen Zhao

[jishen.zhao@ucsc.edu](mailto:jishen.zhao@ucsc.edu)

Computer Engineering

UC Santa Cruz

July 12, 2016

  
MPSoC'16



UNIVERSITY OF CALIFORNIA  
SANTA CRUZ