# An Analysis of Persistent Memory Use with WHISPER

Sanketh Nalli, Swapnil Haria, Michael M. Swift, Mark D. Hill, Haris Volos*, Kimberly Keeton*

University of Wisconsin-Madison &
*Hewlett-Packard Labs

# Executive Summary

Facilitate better system support for Persistent Memory (PM)

**W**isconsin-**H**P Labs **S**uite for **Per**sistence, a benchmark suite for PM

- 4% accesses to PM, 96% accesses to DRAM
- 5-50 epochs/tx, contributed by memory allocation & logging
- 75% of epochs are small, update just one PM cacheline
- Re-referencing PM cachelines:
  Common in a thread, rare across threads

**H**ands **O**ff **P**ersistence **S**ystem (HOPS) optimizes PM transactions

WHISPER: research.cs.wisc.edu/multifacet/whisper

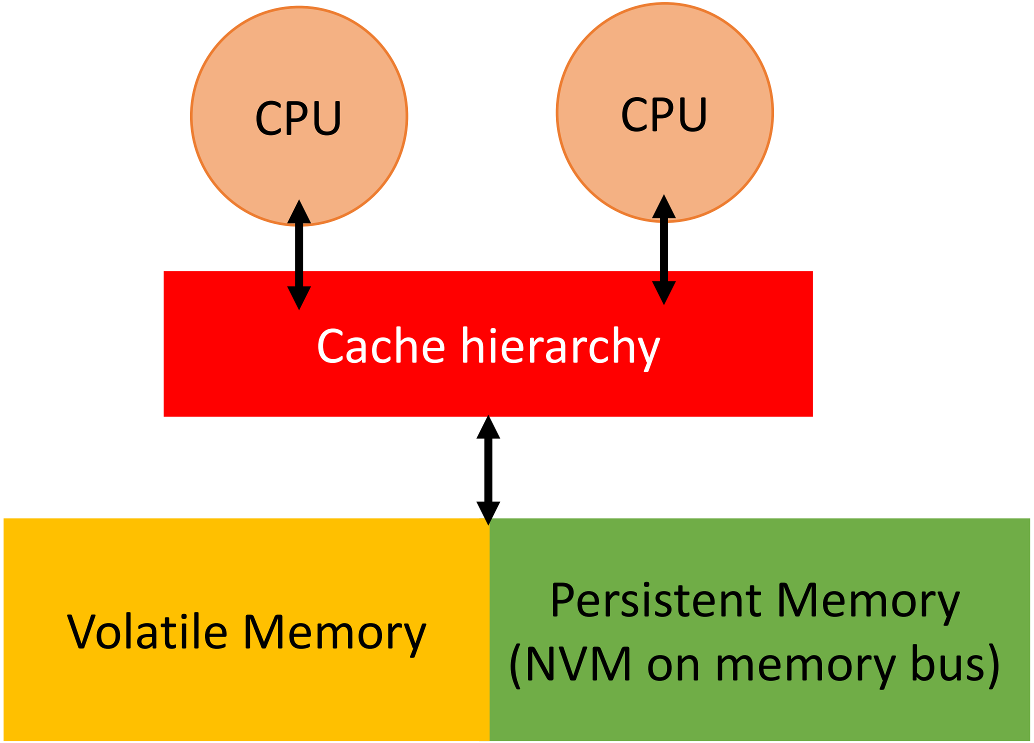
# Outline

➜ WHISPER: **W**isconsin-**H**P Labs **S**uite for **P**ersistence

WHISPER Analysis

HOPS : **H**ands-**O**ff **P**ersistence **S**ystem

# Persistent Memory is coming soon



PM = NVM attached to CPU on memory bus

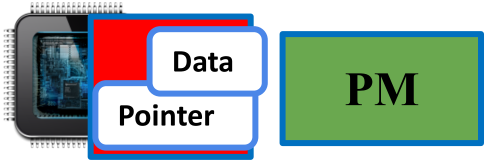
Offers low latency reads and persistent writes

Allows user-level, byte-addressable loads and stores

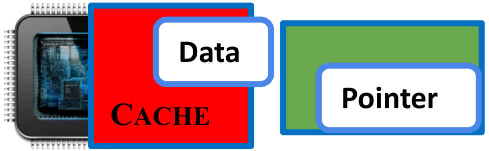
# What guarantees after failure ?

**Durability** = Data survives failure

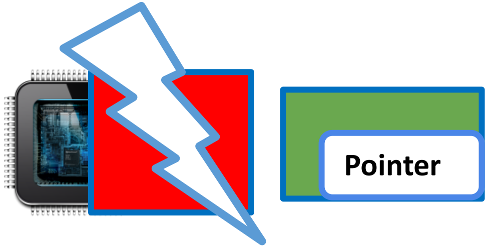
**Consistency** = Data is usable



1 . Data update followed by pointer update in cache

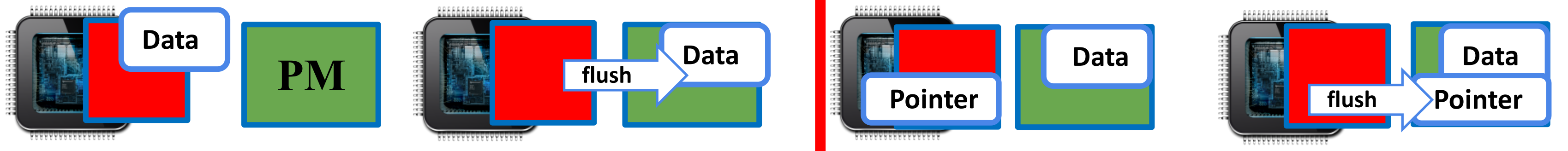


2. Pointer is evicted from cache to PM



3. Data lost on failure, dangling pointer persists

# Achieving consistency



1 . Store data update in cache

2 . Flush data update to PM

3 . Store pointer update in cache

4 . Flush pointer update to PM

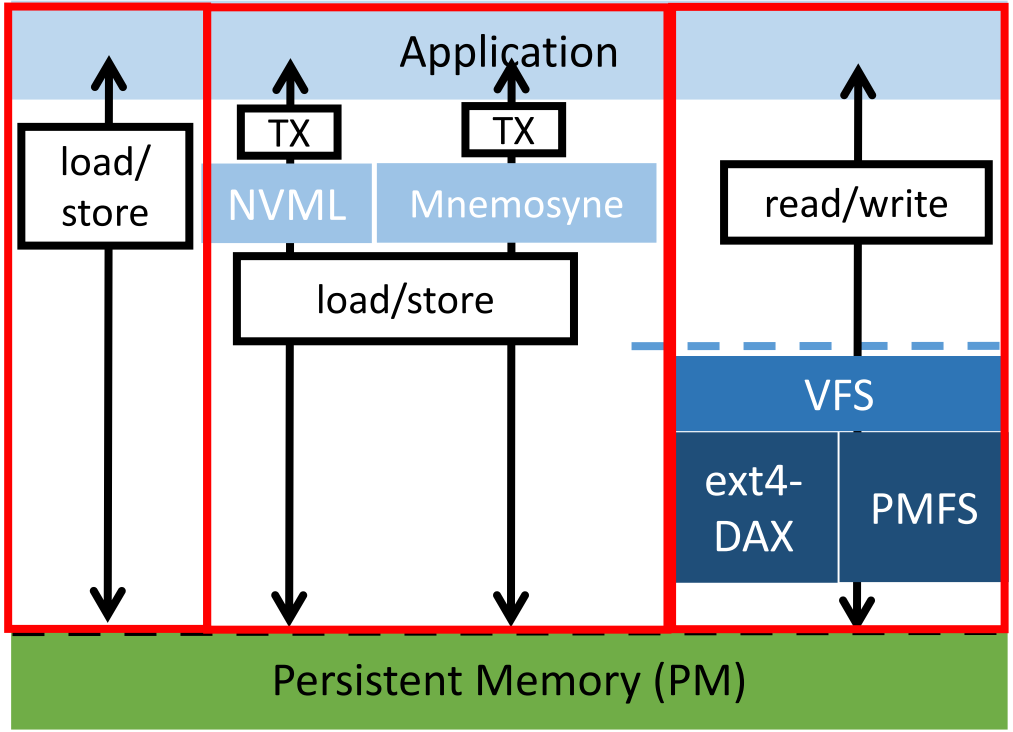
**Ordering** = Useful building block of consistency mechanisms

**Epoch** = Set of writes to PM guaranteed to be durable before ANY subsequent writes become durable

Ordering primitives: SFENCE on x86-64

# PM systems for consistency

- **Native**
  Application-specific optimizations
- **Persistent library**
  Atomic allocations, transactions
- **PM-aware Filesystems**
  POSIX interface



# What’s the problem ?

Lack of standard workloads slows research

Micro-benchmarks not very representative

Partial understanding of how applications use PM

# WHISPER benchmark suite

| Benchmark | Type | Brief description (*Adapted to PM) |
|---|---|---|
| Echo* | KV store | Scalable, multi-version  key-value store |
| N-store* | Database | Fast, in-memory relational DB |
| Redis | NVML | Remote Dictionary Service |
| C-tree | NVML | Microbenchmarks for simulations |
| Hashmap | NVML | Microbenchmarks for simulations |
| Vacation* | Mnemosyne | Online travel reservation system |
| Memcached* | Mnemosyne | In-memory key-value store |
| NFS | PMFS | Linux server/client for remote file access |
| Exim | PMFS | Mail server;stores mails in per-user file |
| MySQL | PMFS | Widely used RDBMS for OLTP |

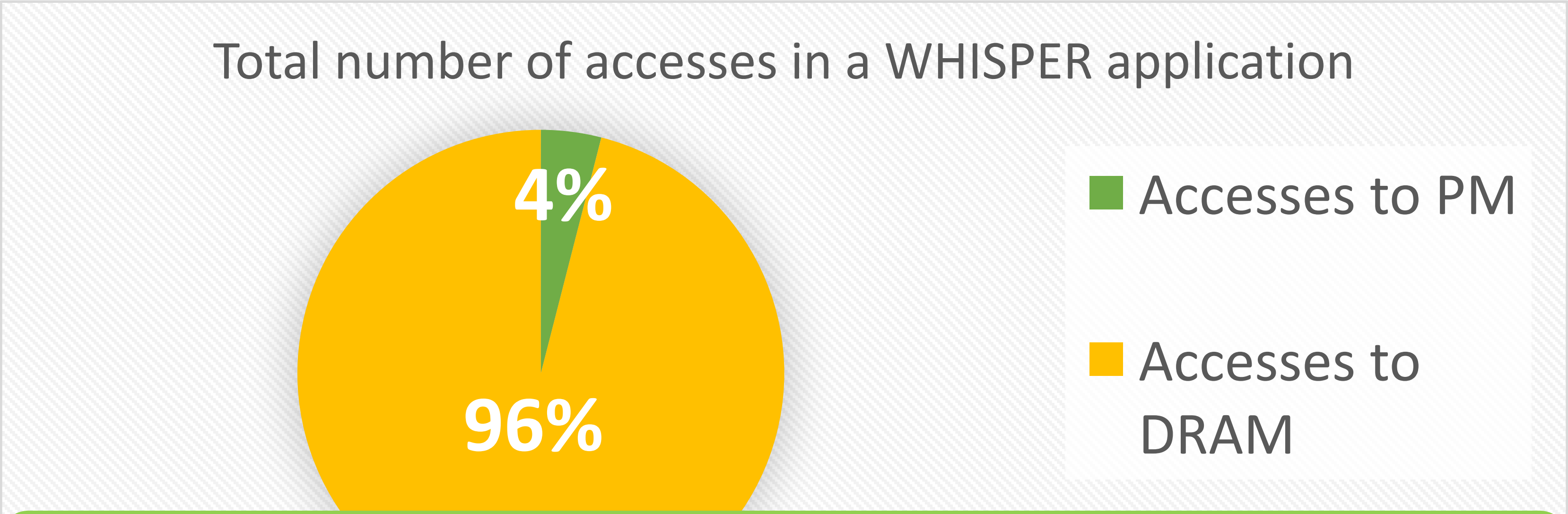
# Outline

✔ WHISPER: **W**isconsin-**H**P Labs **S**uite for **P**ersistence

➜ WHISPER Analysis

HOPS : **H**ands-**O**ff **P**ersistence **S**ystem

# How many accesses to PM ?



Suggestion: Do not impede volatile accesses

# How many epochs/transaction ?

Durability after every epoch impedes execution

**Expectation**: 3 epochs/TX = log + data + commit

**Reality**: 5 to 50 epochs/TX

Suggestion: Enforce durability only at the end of a transaction

# What contributes to epochs ?

Log entries

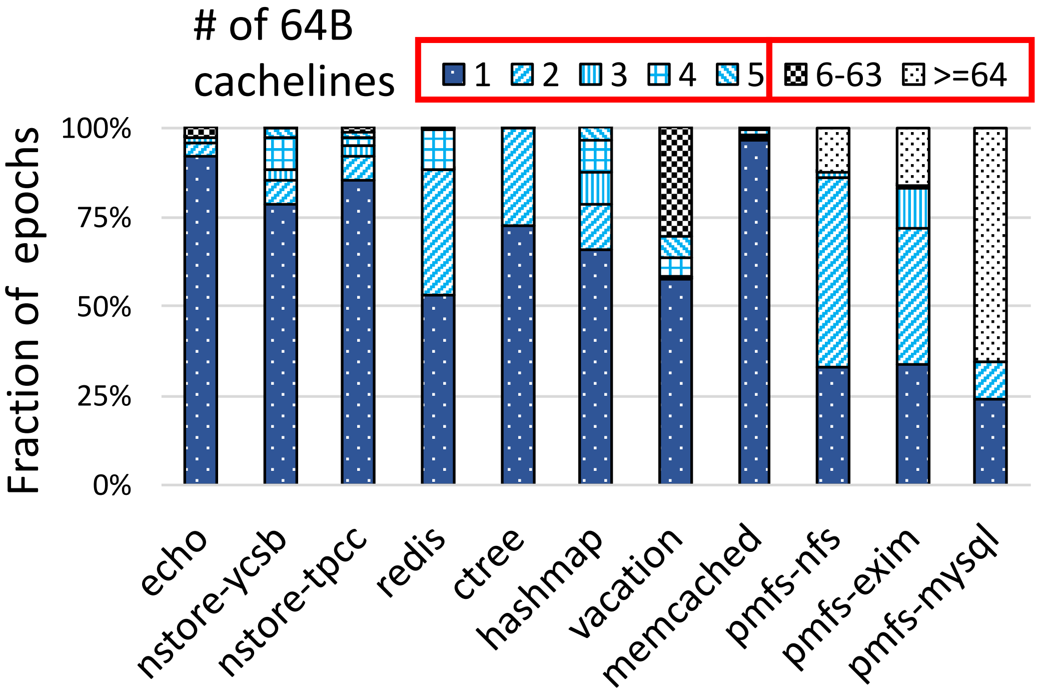
- **Undo log**: Alternating epochs of log and data
- **Redo log**: 1 Log epoch + 1 data epoch

Persistent memory allocation

- 1 to 5 epochs

Suggestion: Use redo logs and reduce epochs from memory allocator

# How large are epochs?



Determines amount of state buffered per epoch

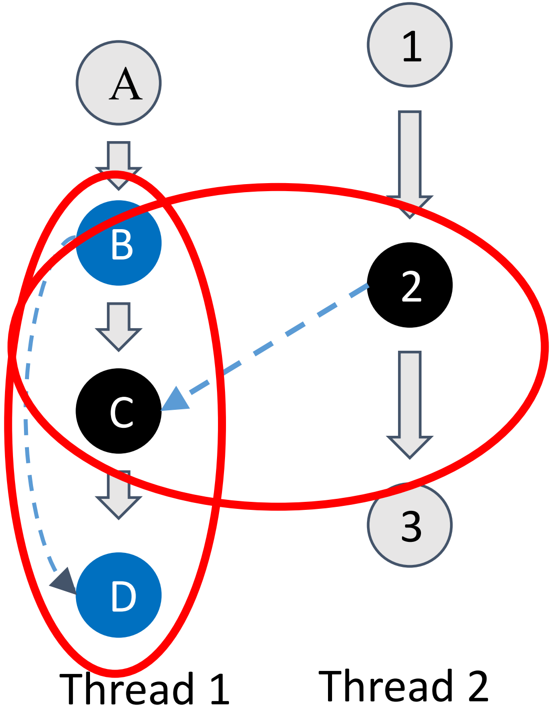
Small epochs are abundant

- **75%** update single cacheline

Large epochs in PMFS

Suggestion: Consider optimizing for small epochs

# What are epoch dependencies ?



Self-dependency: B → D

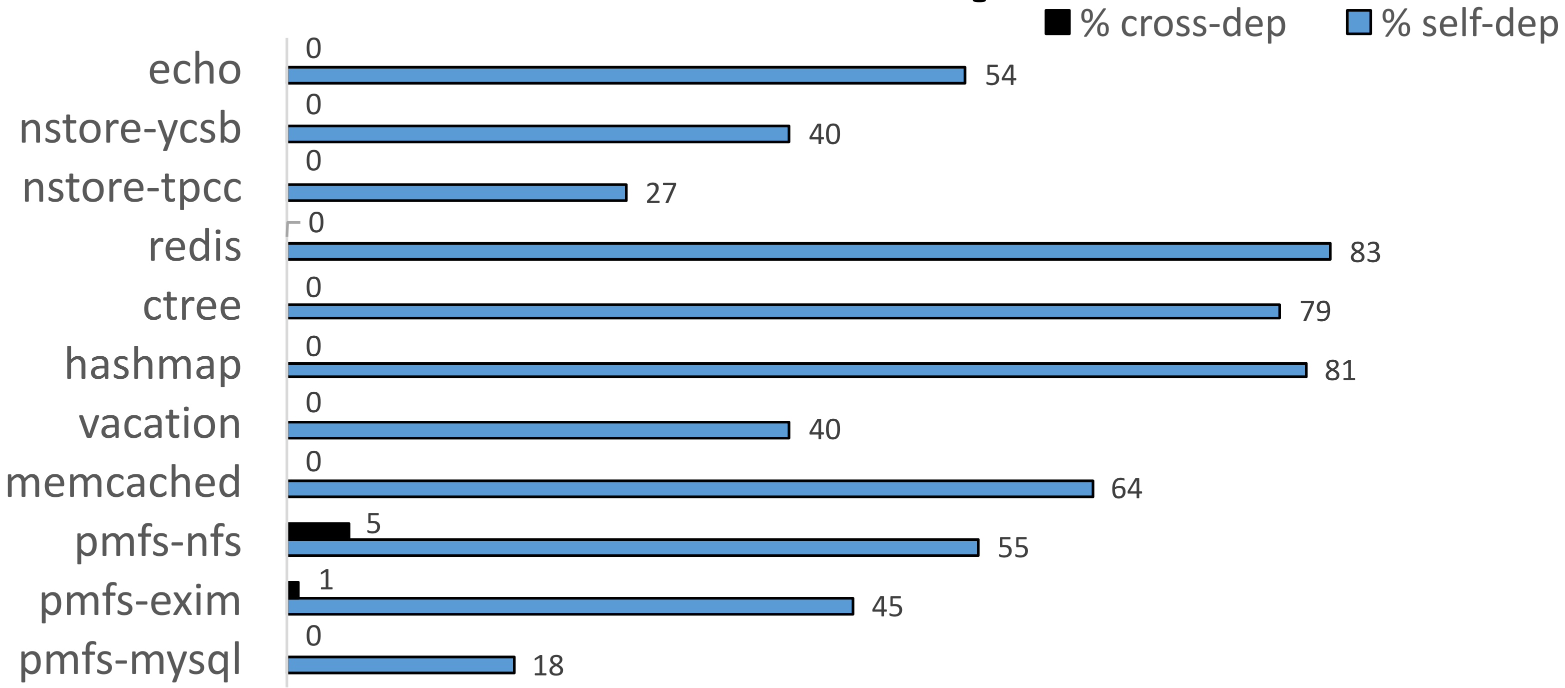
Cross-dependency: 2 → C

Why do they matter ?

- Dependency can **stall** execution

Measured dependencies in 50 microsec window

# How common are dependencies ?



Suggestion: Design multi-versioned buffers

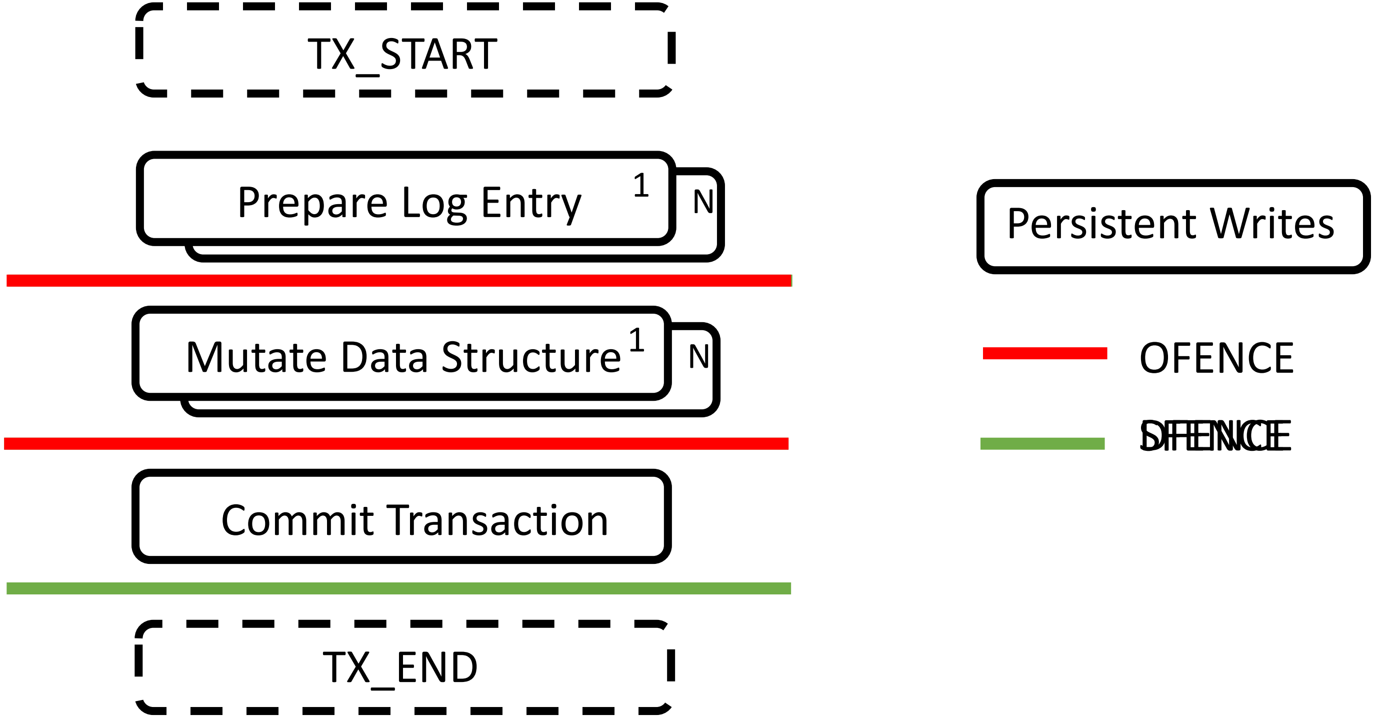
OR avoid updating same cacheline across epochs

# Outline

- ✓ WHISPER: **W**isconsin-**H**P Labs **S**uite for **P**ersistence
- ✓ WHISPER Analysis
- → HOPS : **H**ands-**O**ff **P**ersistence **S**ystem

# ACID Transactions in HOPS

# ACID Transactions



TX_START

Prepare Log Entry 1 … N

Mutate Data Structure 1 … N

Commit Transaction

TX_END

Persistent Writes

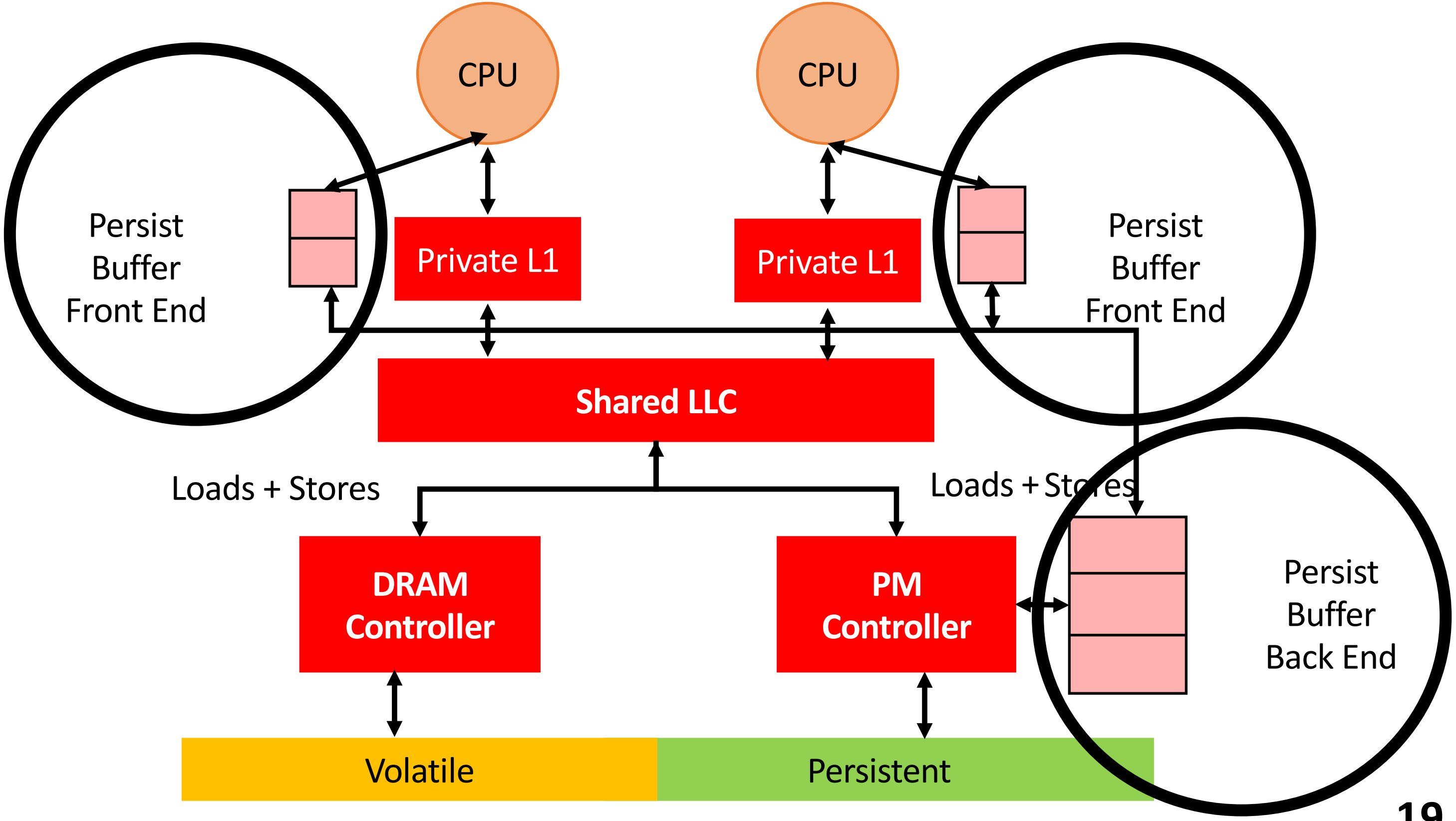
🟥 OFENCE

🟩 SFENCE

🟩 DFENCE

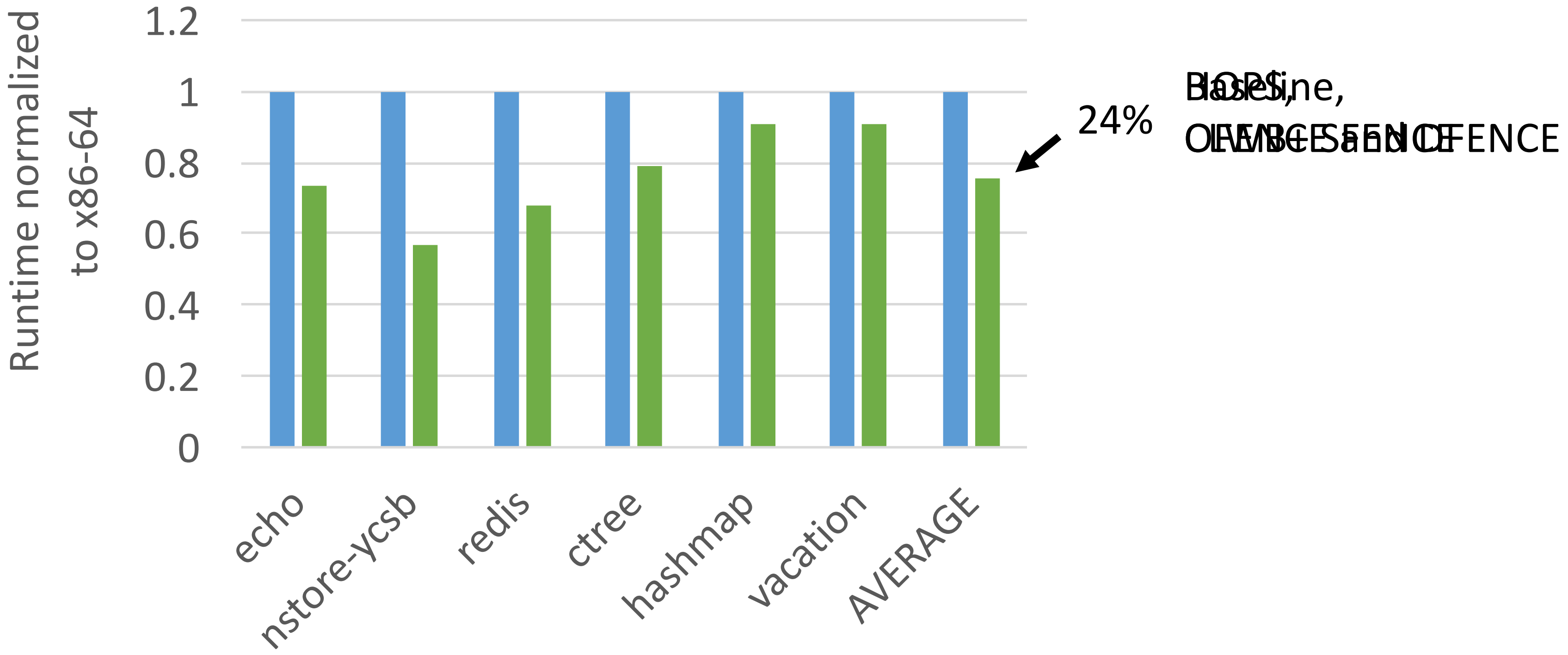
# HOPS Persist Buffers

# HOPS Basic System



| WHISPER | | HOPS |
|---|---|---|
| 4% accesses to PM, 96% to DRAM | ⇨ | Volatile memory hierarchy (almost) unchanged by PBs |
| 5-50 epochs/transaction | ⇨ | Order epochs without flushing |
| Self-dependencies common | ⇨ | Allows multiple copies of same cacheline in PB via timestamps |
| Cross-dependencies rare | ⇨ | Correct, conservative method using coherence & timestamps |

# HOPS Evaluation with WHISPER



# Summary

- Persistent Memory (PM) is coming soon
- Progress is slowed by ad-hoc micro-benchmarks
- We contributed **WHISPER**, open-source benchmark suite
- **HOPS** design, based on WHISPER analysis
- We hope for more similar analysis in the future !

research.cs.wisc.edu/multifacet/whisper/

# Extra

# Summary

- WHISPER: **W**isconsin-**H**P Labs **S**uite for **Per**sistence
- 4% accesses to PM, 96% accesses to DRAM
- 5-50 epochs/TX, primarily small in size
- Cross-dependencies rare, self-dependencies common
- HOPS improves PM app performance by 24%
- More results in ASPLOS'17 paper and code at:

research.cs.wisc.edu/multifacet/whisper/

# A Simple Transaction using Epochs

```
TM_BEGIN();
    pobj.data = 42;
    pobj.init = True;
TM_END();
```

```
transaction_begin:
log[pobj.init] ← True
log[pobj.data] ← 42
write_back(log)
wait_for_write_back()
pobj.init ← True
pobj.data ← 42
write_back(pobj)
wait_for_write_back()
transaction_end
```

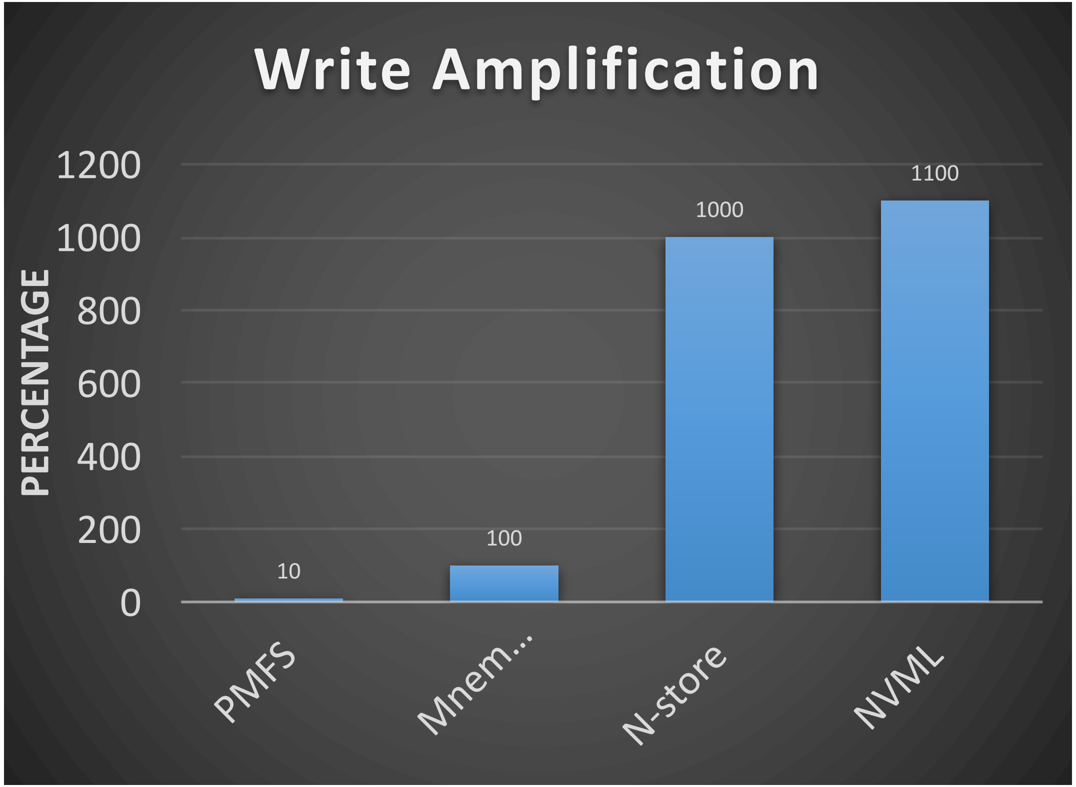
**Epoch 1**

Log entries stored & persisted.

**Epoch 2**

Variables stored & persisted.

# Runtimes cause write amplification



- PMFS
- Mnemosyne
  - Logs every PM write
- PMFS
- NVML
  - Clears log
  - Auxiliary structures
- < 5% writes to PM
- **Non-temporal writes**
  - Mnemosyne logs
  - PMFS user-data