NVOverlay: Enabling Scalable and Efficient High-Frequency Snapshot to NVM

Ziqi Wang[1], Chulhwan Choo[2], Mike A. Kozuch[3], Todd C. Mowry[1], Gennady Pekhimenko[4], Vivek Seshadri[5], Dimitrios Skarlatos[1]

[1]Carnegie Mellon University

[2]Samsumg

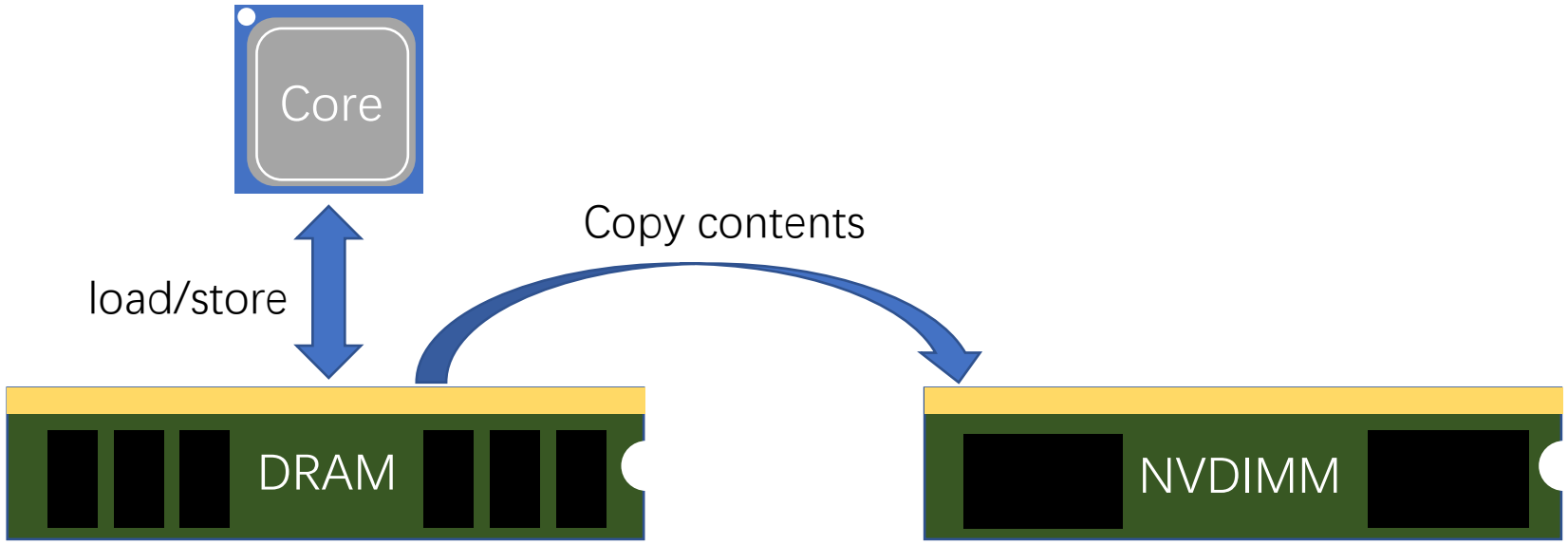
[3]Microsoft Research India

[4]University of Toronto

[5]Intel Labs

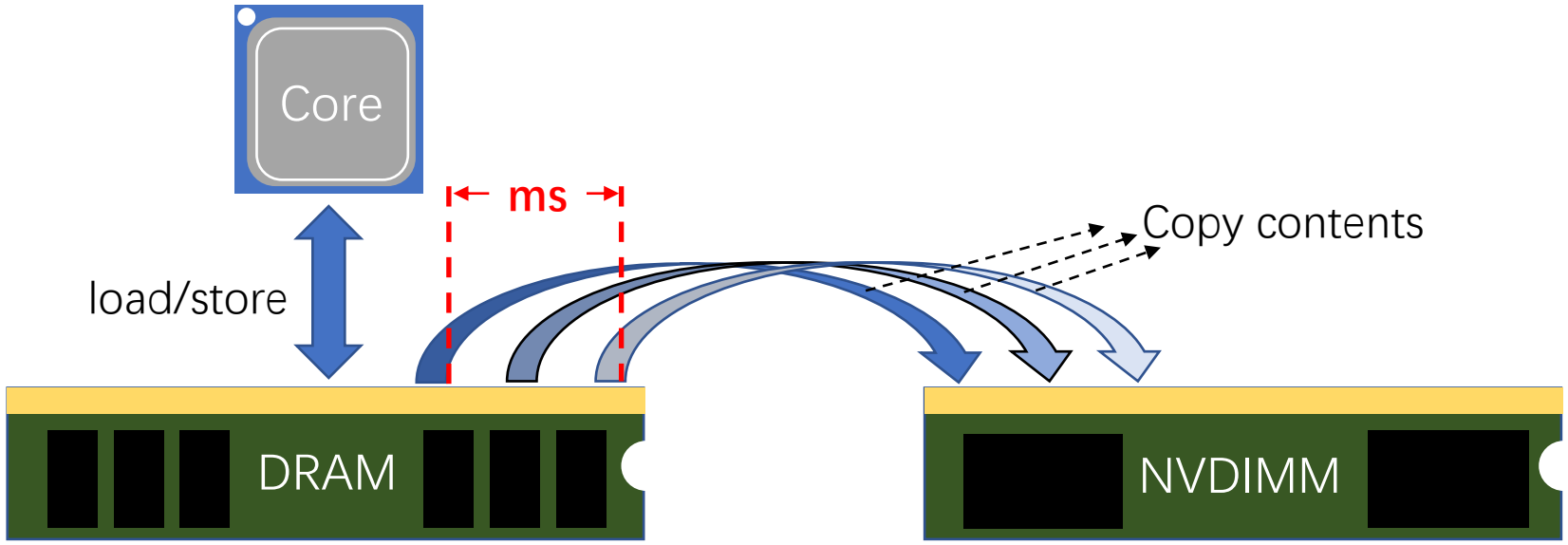
What is “snapshotting”?

- Saving the content of the entire address space to persistent storage for later retrieval



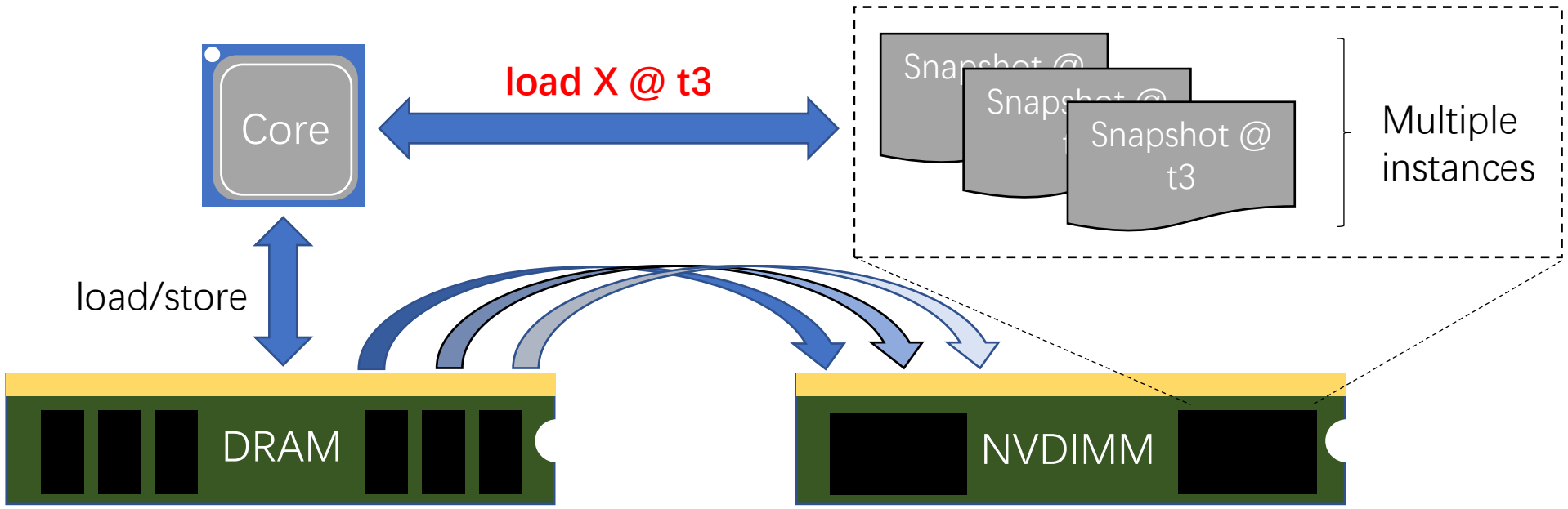
Why is NVOverlay different?

- **High frequency snapshotting (millisecond scale)**

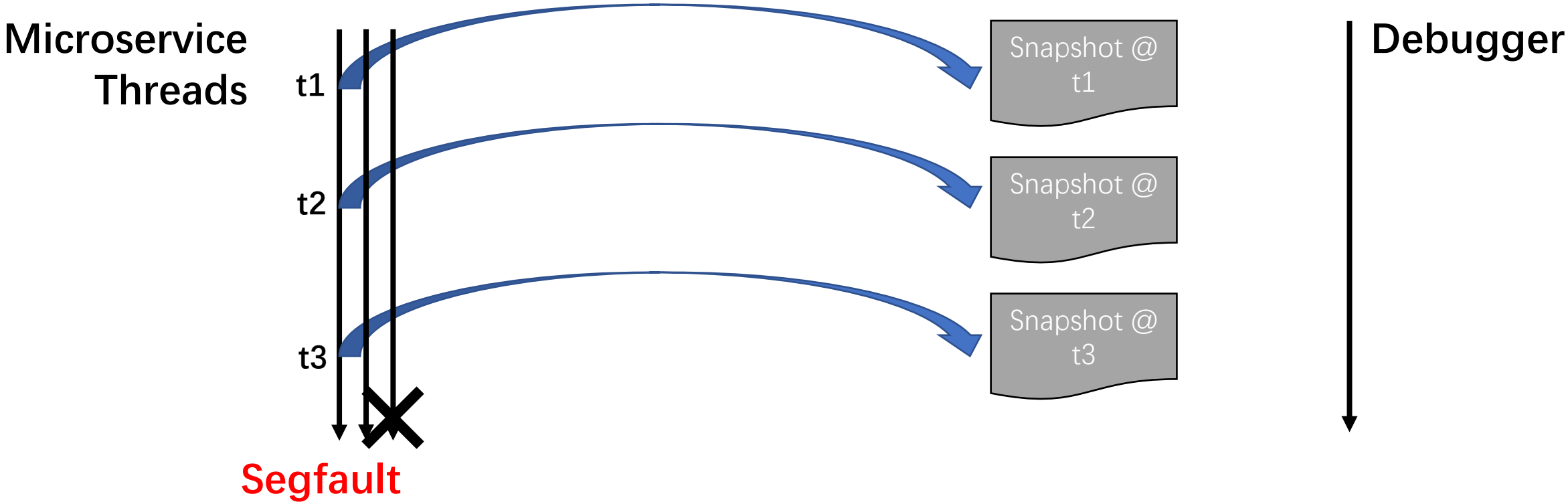


Why is NVOverlay different?

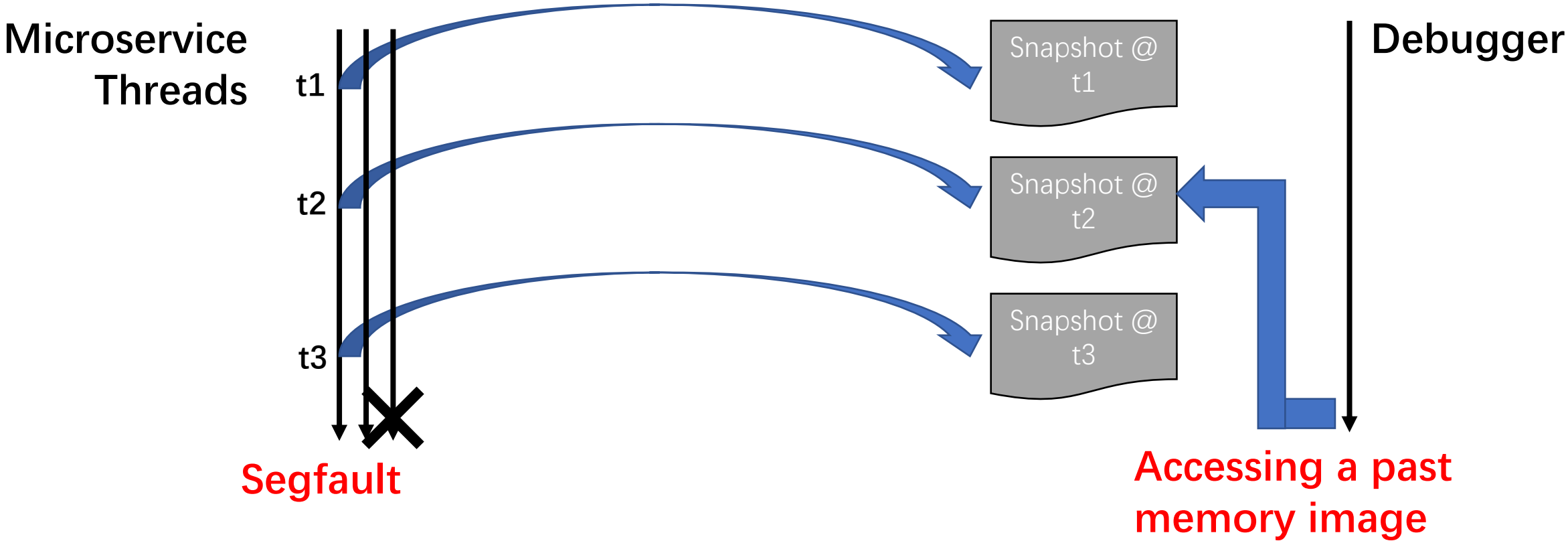
- High frequency, e.g., millisecond scale
- **Support random accesses to multiple snapshots**



Time-Travel Debugging with NVOverlay



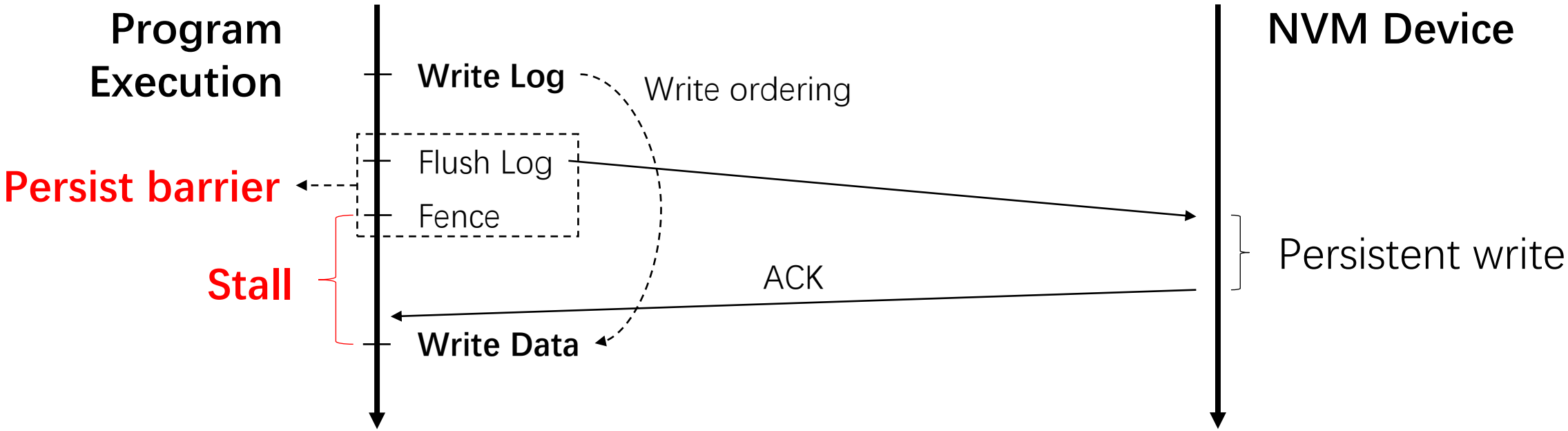
Time-Travel Debugging with NVOverlay



Challenges of High Frequency Snapshotting

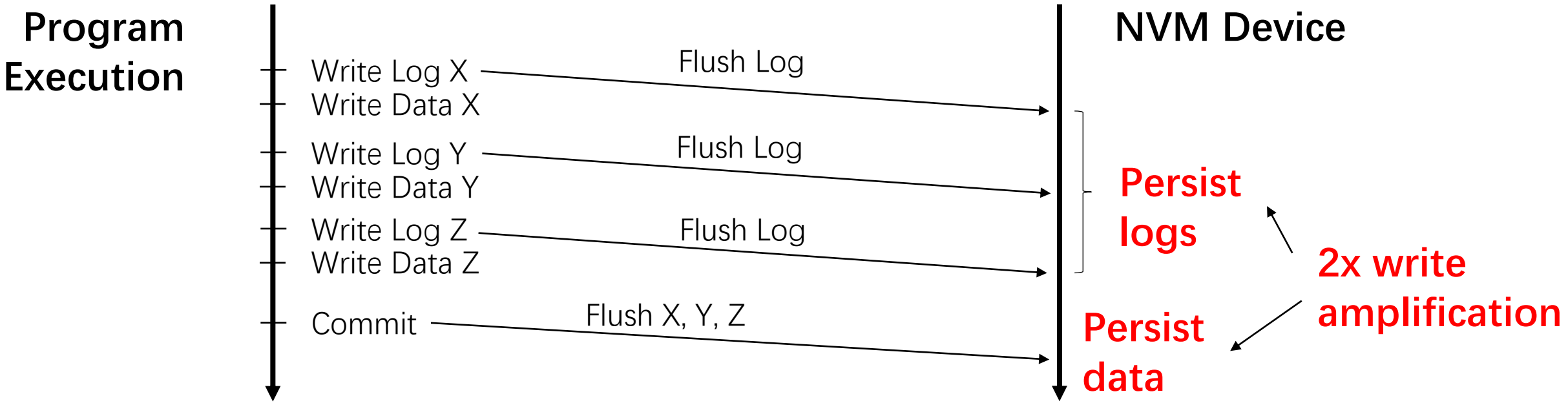
- Avoid persist barrier stalls
- Minimize NVM write amplification
- Scale to large, distributed systems

Avoid Persist Barrier Stalls



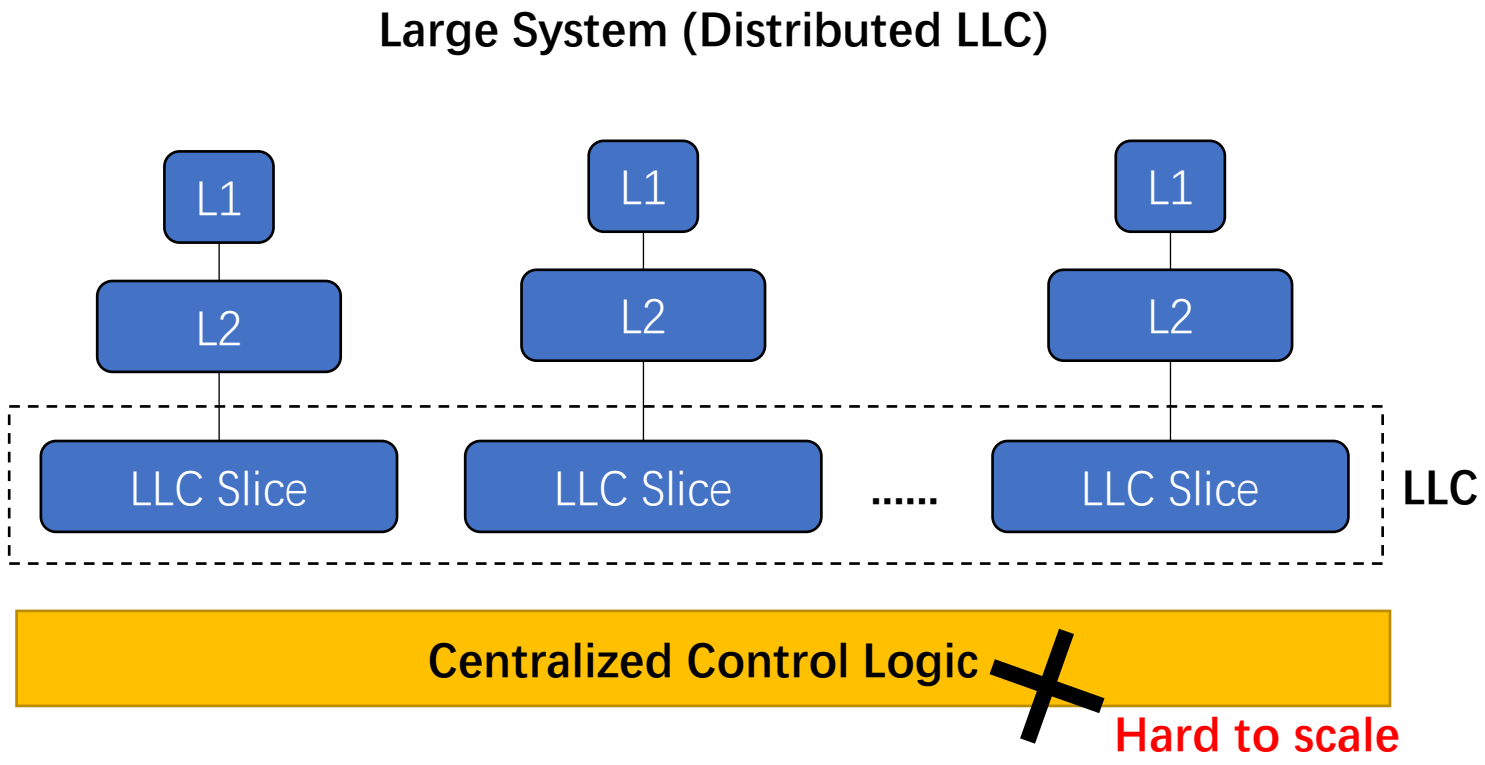
Software Undo Logging

Minimize NVM Write Amplification



Hardware Undo Logging

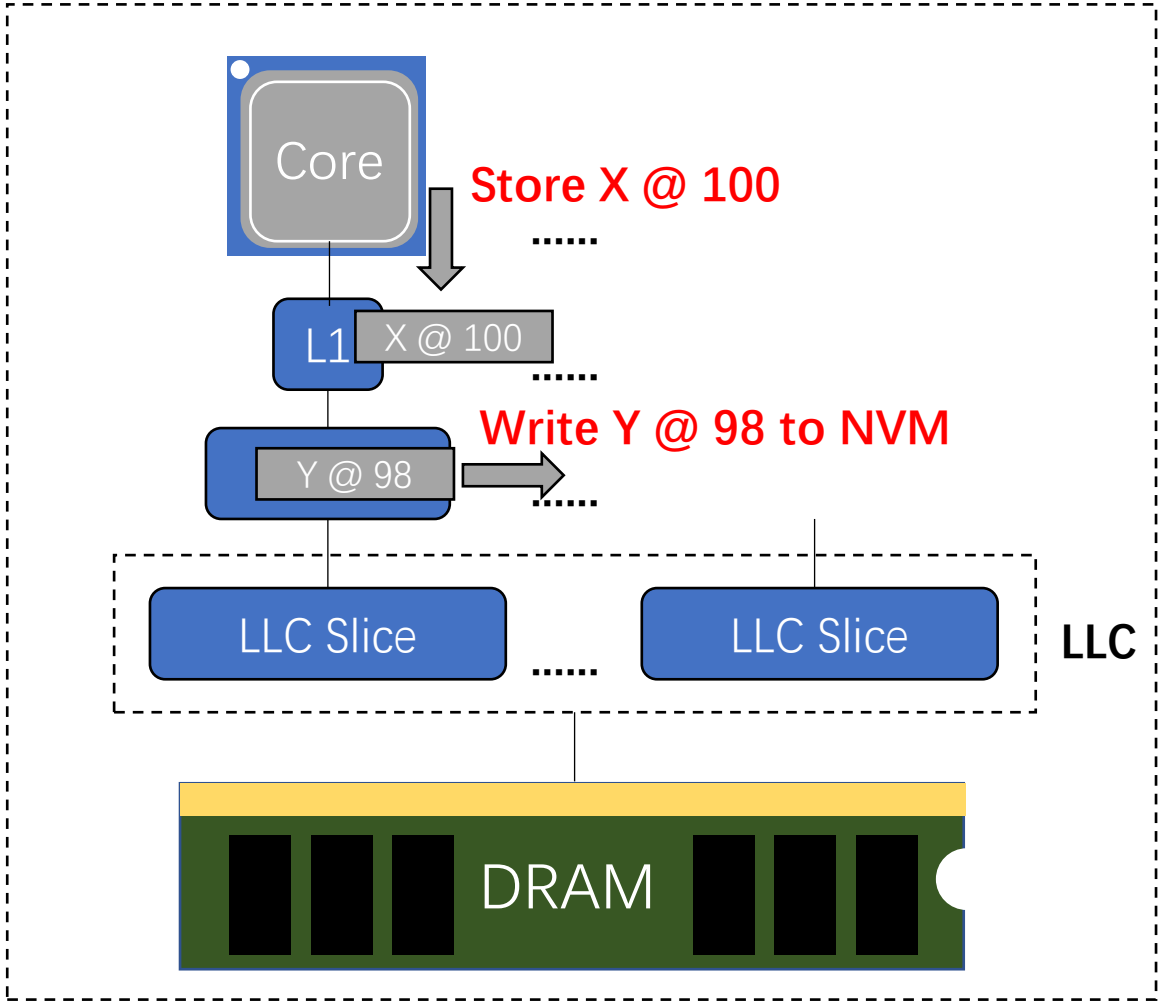
Scale to Large, Distributed Systems



Key Insights and Our Approach

- Two major components
- **Consistent Snapshot Tracking (CST)** for incrementally capturing changes
  - Tags blocks with snapshot number they are lastly written
  - Coordinates block eviction
- **Multi-snapshot NVM Mapping (MNM)** for persisting changes with low
  - Shadow maps blocks to different locations
  - Maintains per-snapshot index

NVOverlay Architecture: CST Frontend

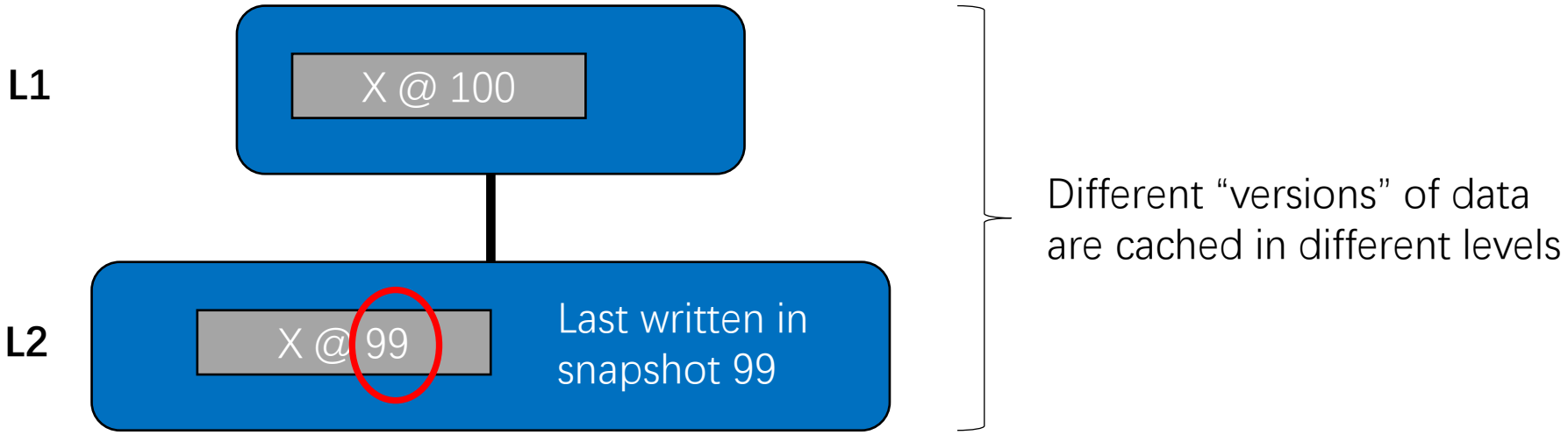


The Coherent Snapshot Tracking (CST) Frontend

- Captures incremental modifications
- Sends dirty data to the NVM
- Maintains consistency of the snapshot

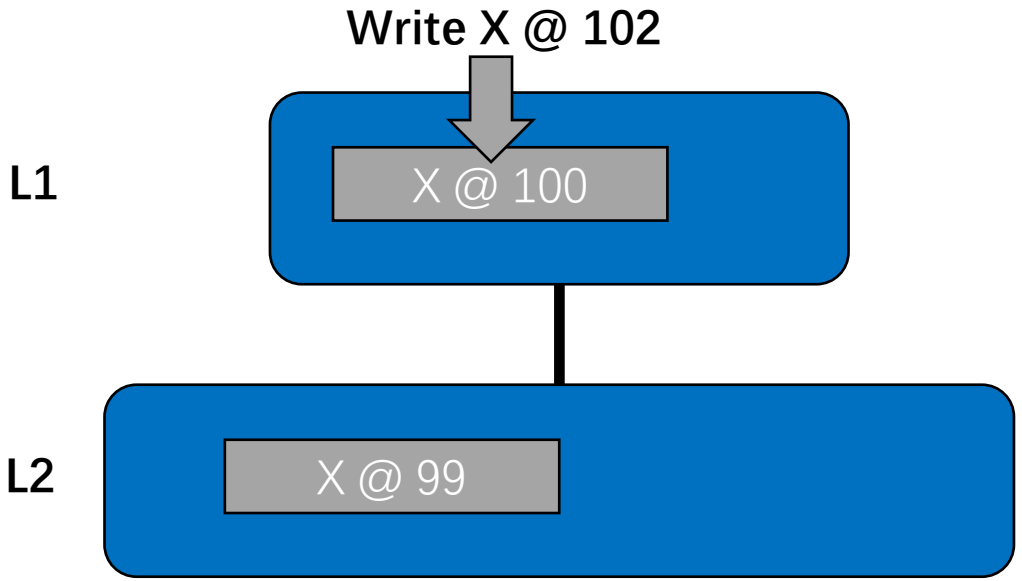
Consistent Snapshot Tracking (CST)

- The cache hierarchy manages snapshot data with multiple copies of data on the same address
  - **Blocks are tagged with the snapshot number they are lastly written**
  - A block is duplicated in the cache before being written in a different snapshot
  - The “old” block is pushed down the hierarchy naturally



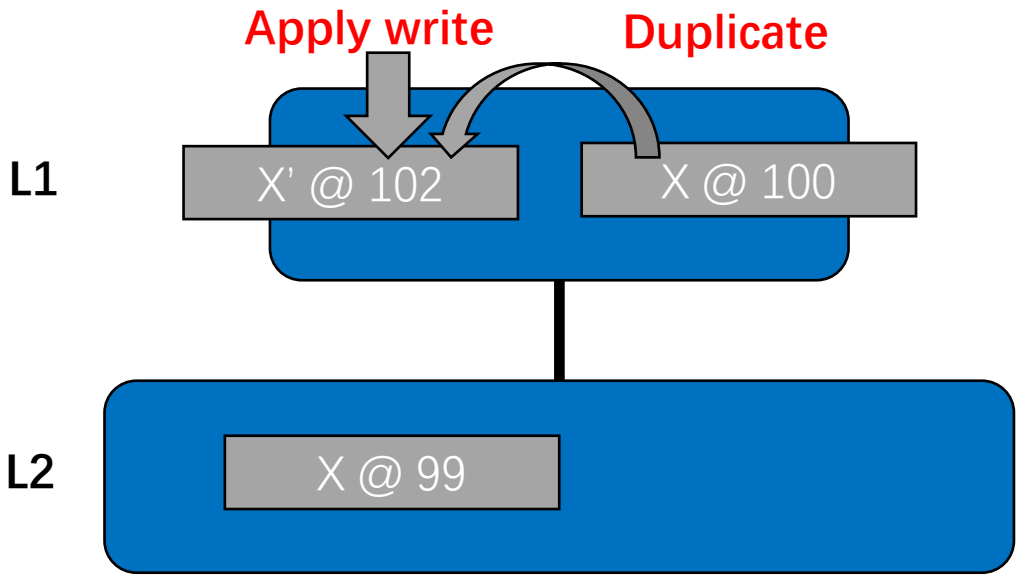
Consistent Snapshot Tracking (CST)

- The cache hierarchy manages snapshot data with multiple copies of data on the same address
  - Blocks are tagged with the snapshot number they are lastly written
  - **A block is duplicated in the cache before being written in a different epoch**
  - The “old” block is pushed down the hierarchy naturally



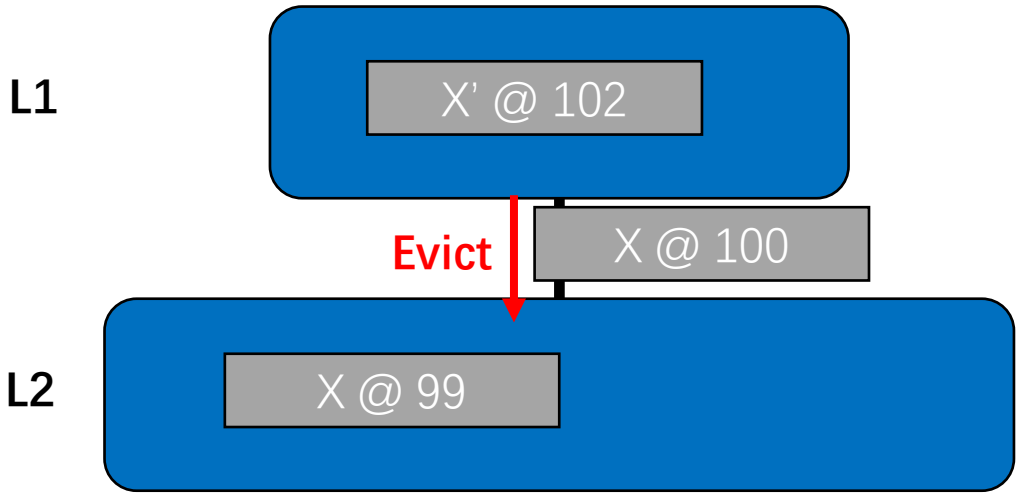
Consistent Snapshot Tracking (CST)

- The cache hierarchy manages snapshot data with multiple copies of data on the same address
  - Blocks are tagged with the snapshot number they are lastly written
  - **A block is duplicated in the cache before being written in a different epoch**
  - The “old” block is pushed down the hierarchy naturally



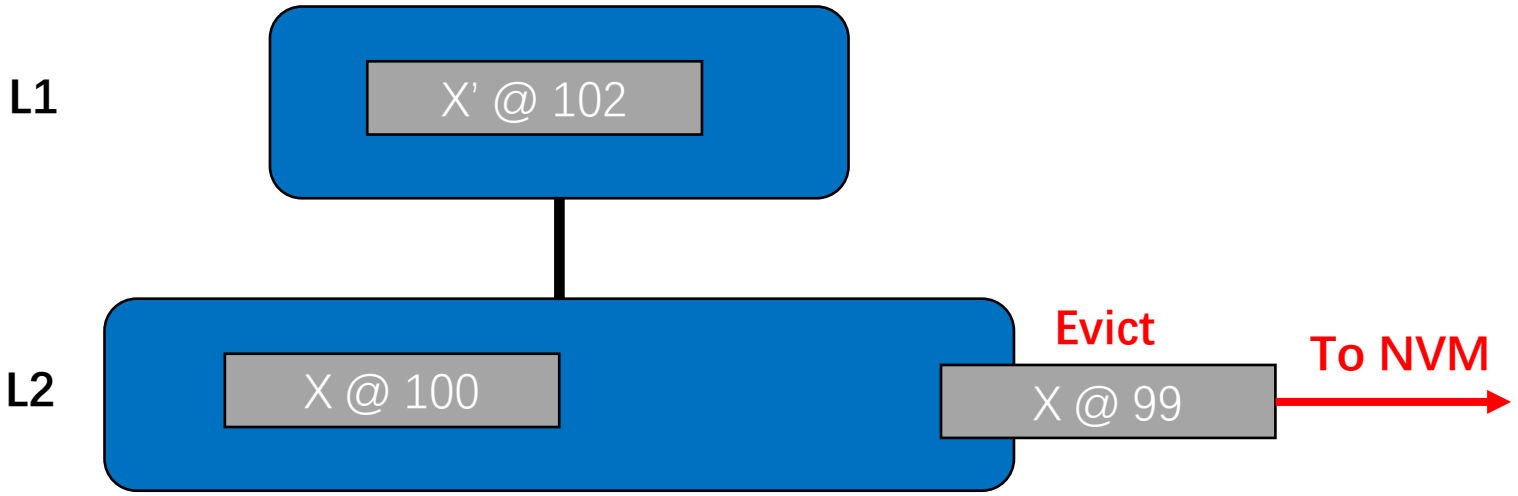
Consistent Snapshot Tracking (CST)

- The cache hierarchy manages snapshot data with multiple copies of data on the same address
  - Blocks are tagged with the snapshot number they are lastly written
  - A block is duplicated in the cache before being written in a different epoch
  - **The “old” block is pushed down the hierarchy naturally**



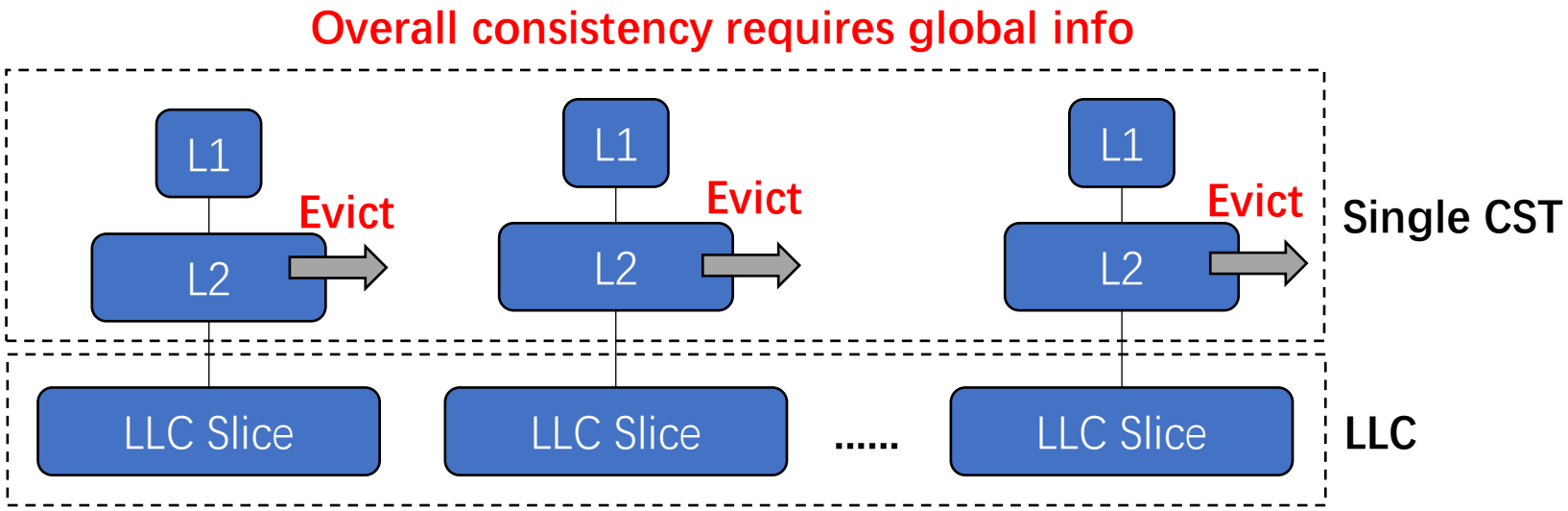
Consistent Snapshot Tracking (CST)

- The cache hierarchy manages snapshot data with multiple copies of data on the same address
  - Blocks are tagged with the snapshot number they are lastly written
  - A block is duplicated in the cache before being written in a different epoch
  - **The “old” block is pushed down the hierarchy naturally**



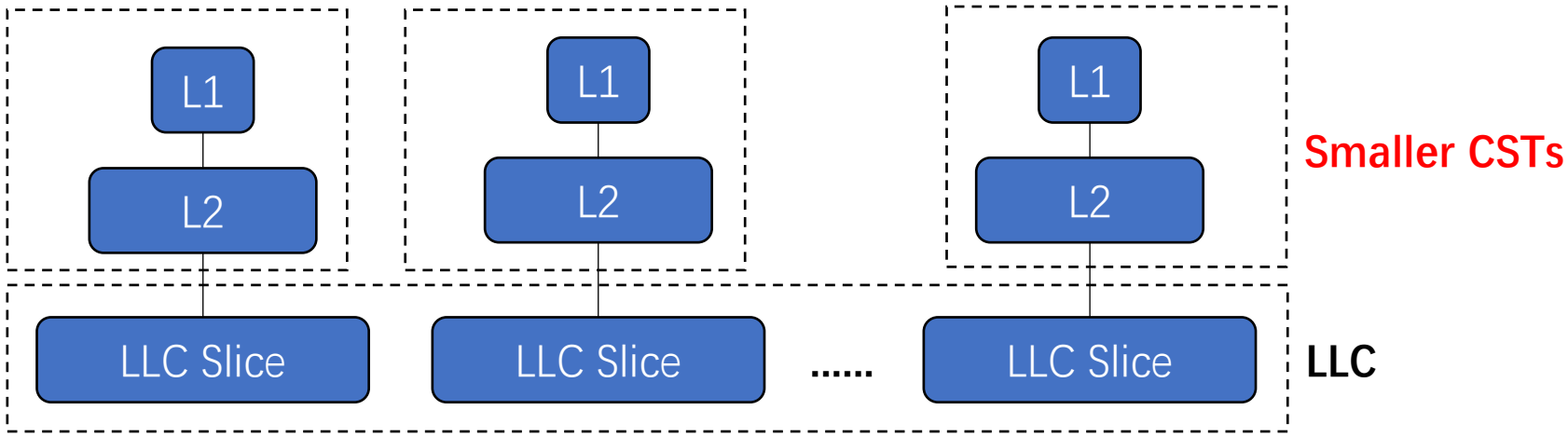
Scale to Multiple Cores

- One consistent “current snapshot” is hard to maintain on large systems with tens of cores, due to data dependency



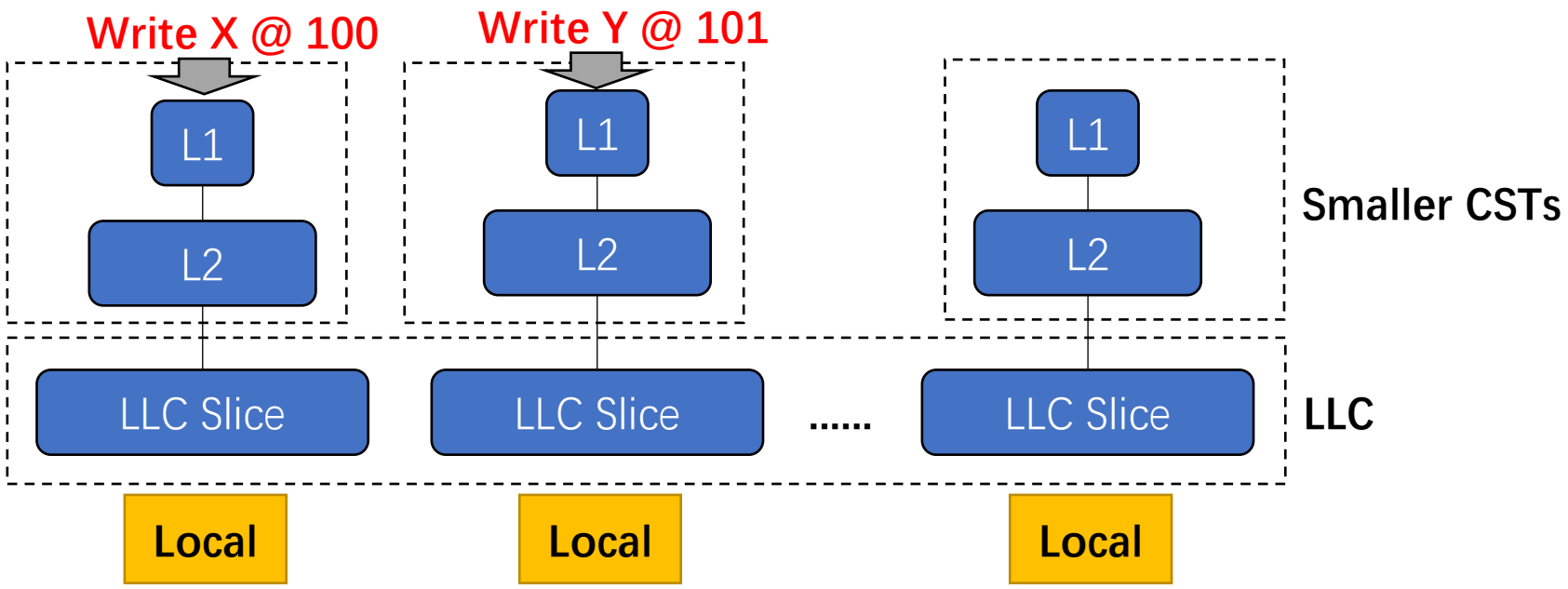
Using Distributed Algorithm to Manage Consistency

- Divide the system into smaller CSTs



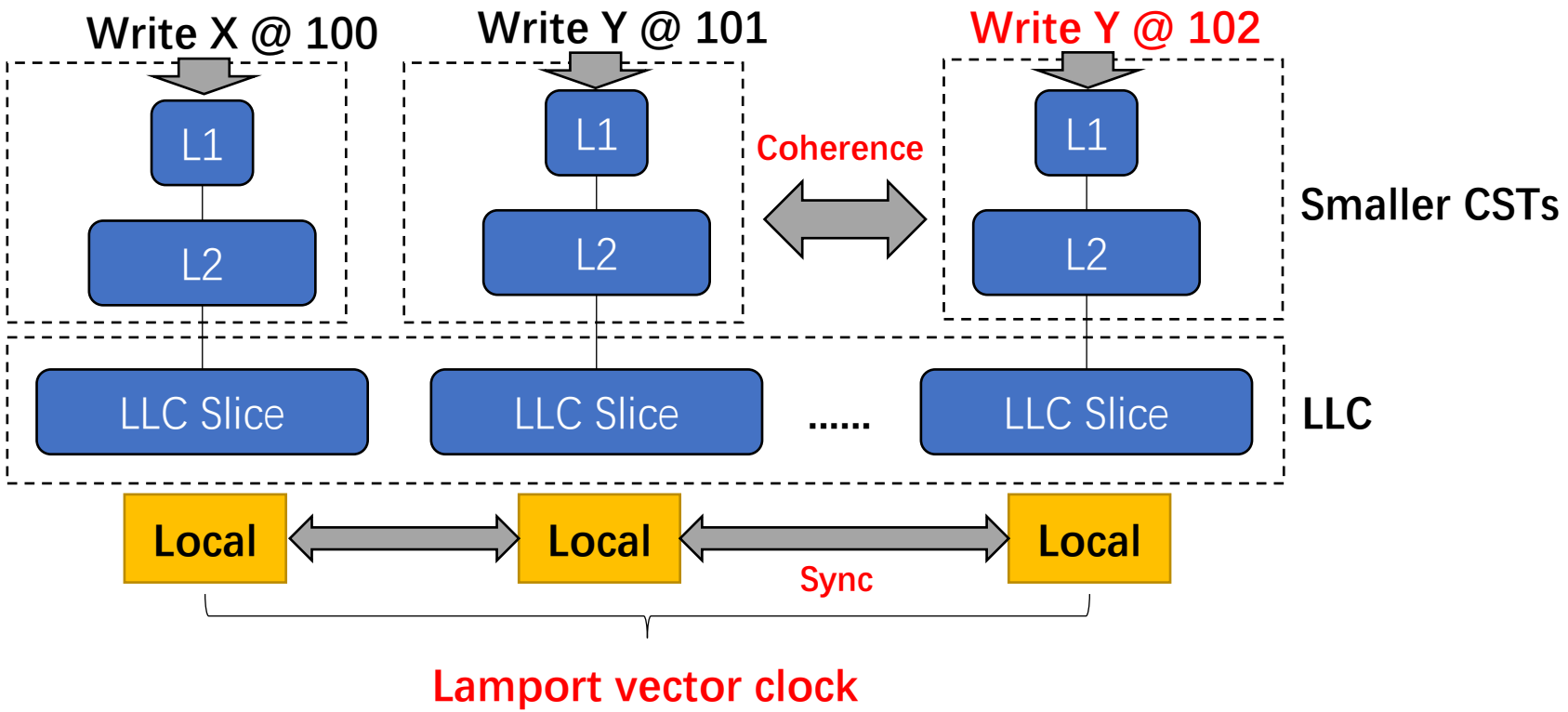
Using Distributed Algorithm to Manage Consistency

- Allow each CST to capture its local snapshot

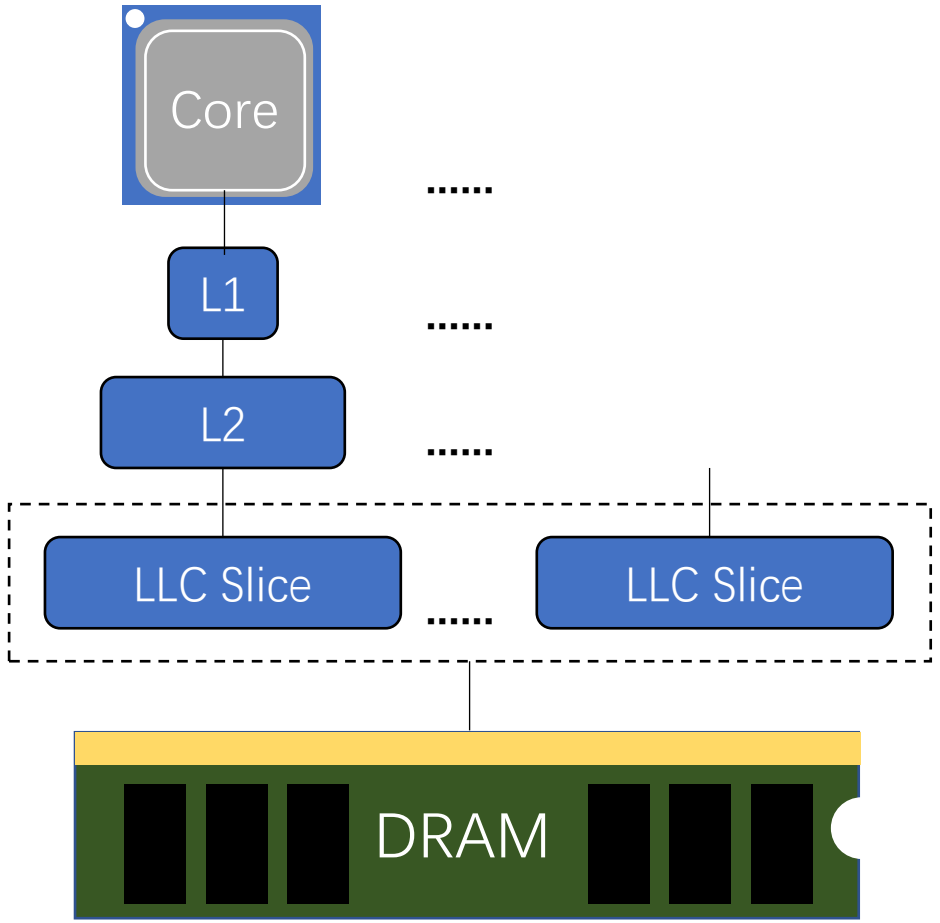


Using Distributed Algorithm to Manage Consistency

- Dependencies are tracked via a **Lamport vector clock**, triggered by **coherence**

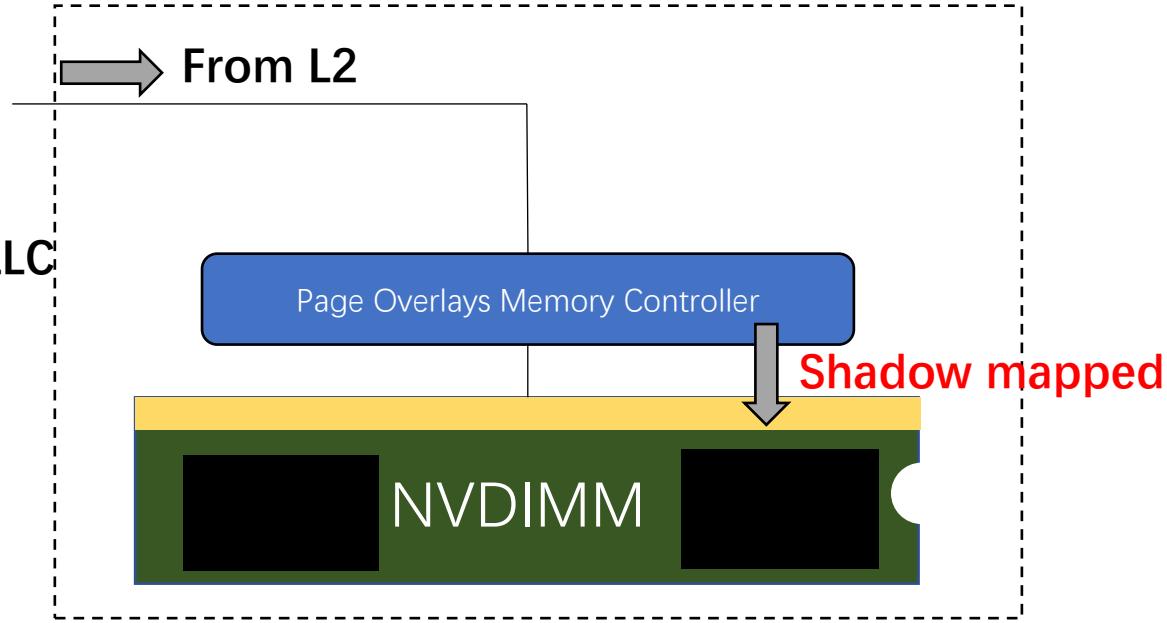


NVOverlay Architecture: Shadow-Mapped NVM Backend

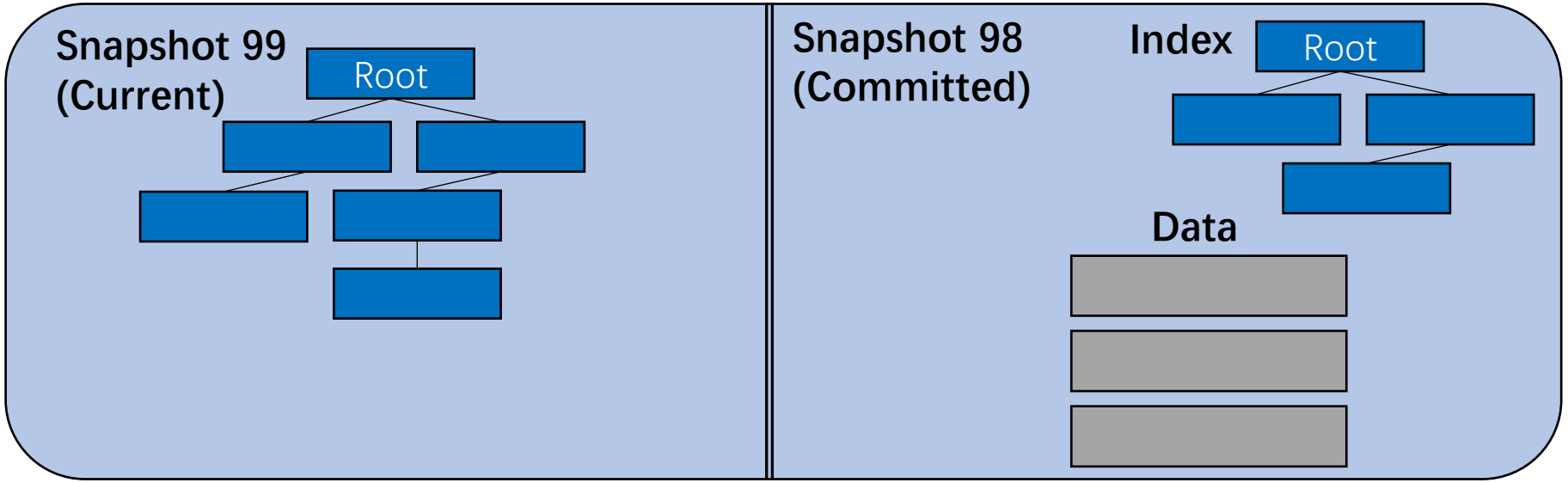


The Multi-snapshot NVM Mapping (MNM) Backend

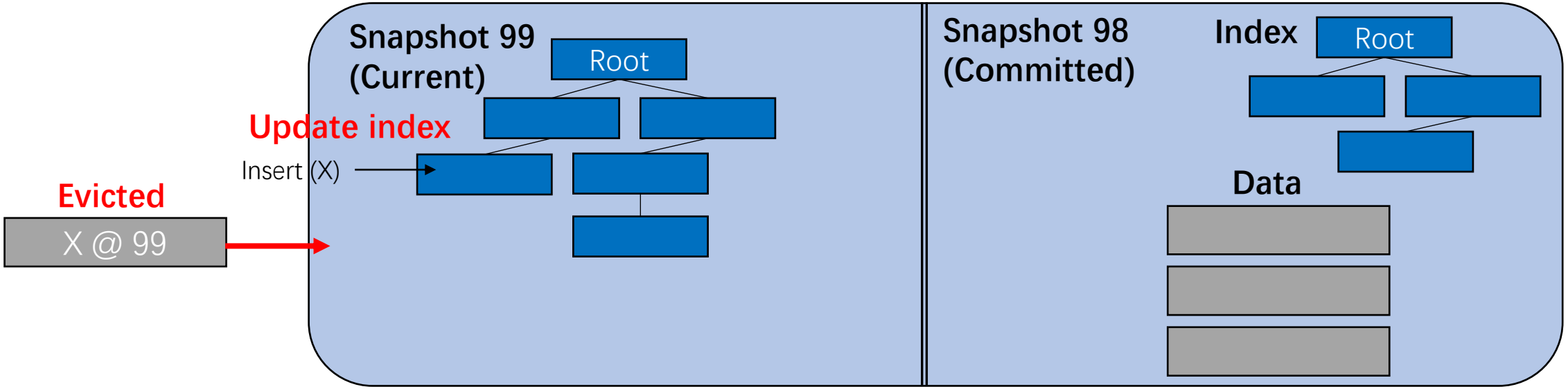
- Manages snapshot instances using Page Overlays (ISCA 2015)
- Hardware shadowing for lower write amplification



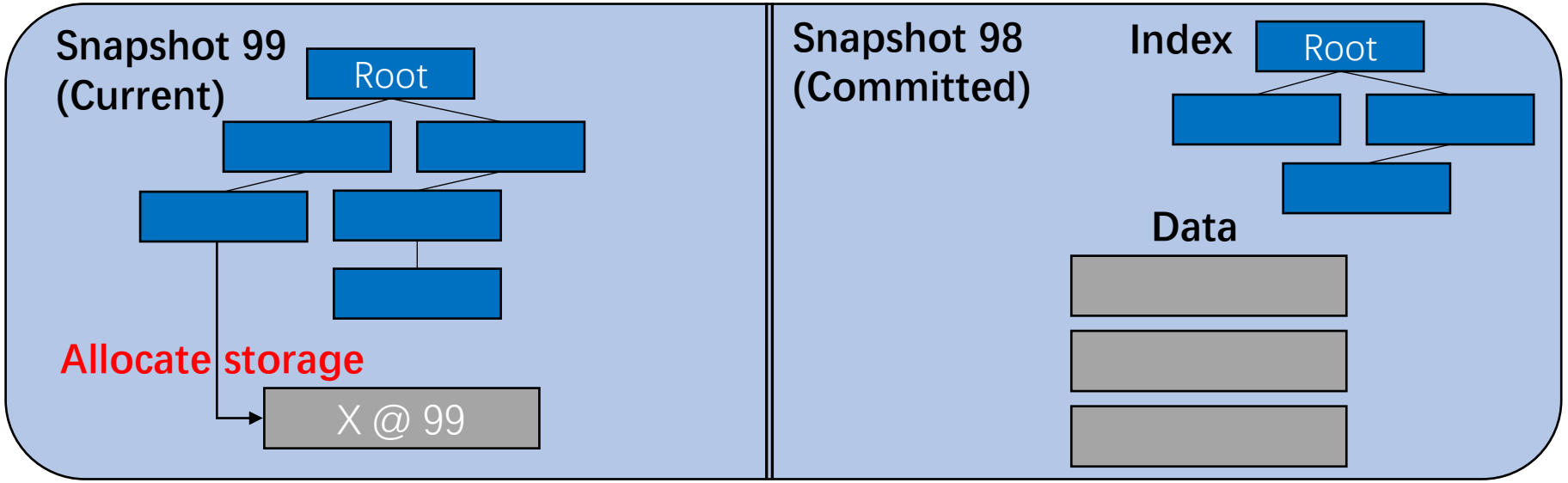
Snapshots are Stored in a Randomly Accessible Format



Memory Controller Updates the Index



Page Overlays Allocates Backing Store



Data is only written once

Reduces write amplification by updating the index

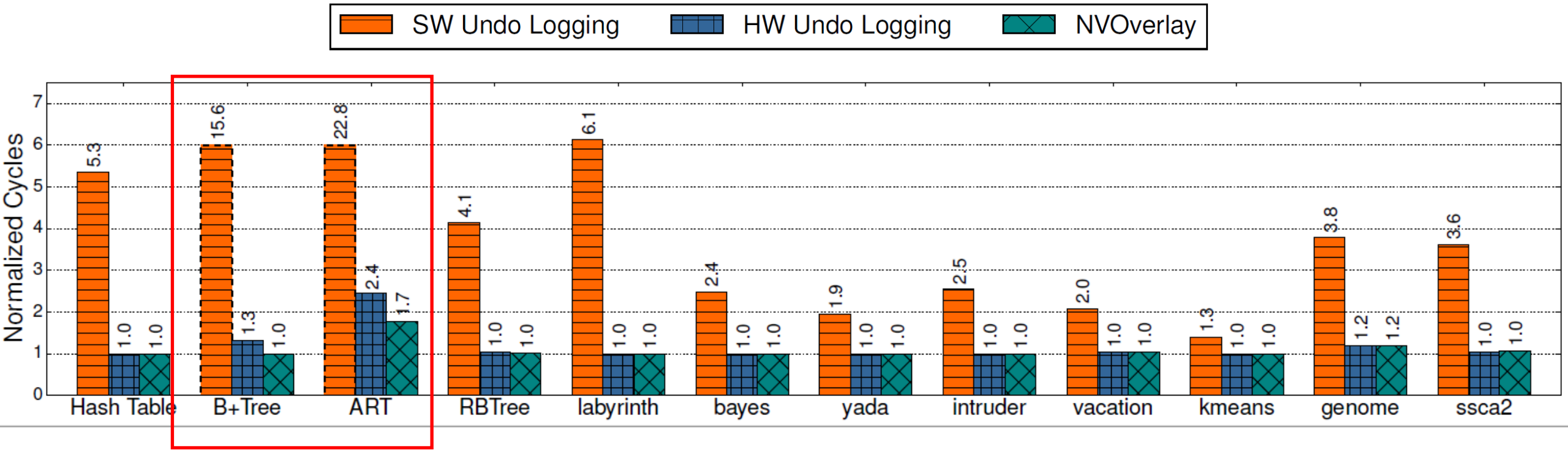
Comparison with Earlier Proposals

	Avoid Persist barrier	Write amplification	Scalable	Multiple Snapshots
Software Logging	✗	High (2x)	✓	✗
Hardware Logging	✓	High (2x)	Maybe	✗
NVOverlay	✓	Low	✓	✓

Evaluation

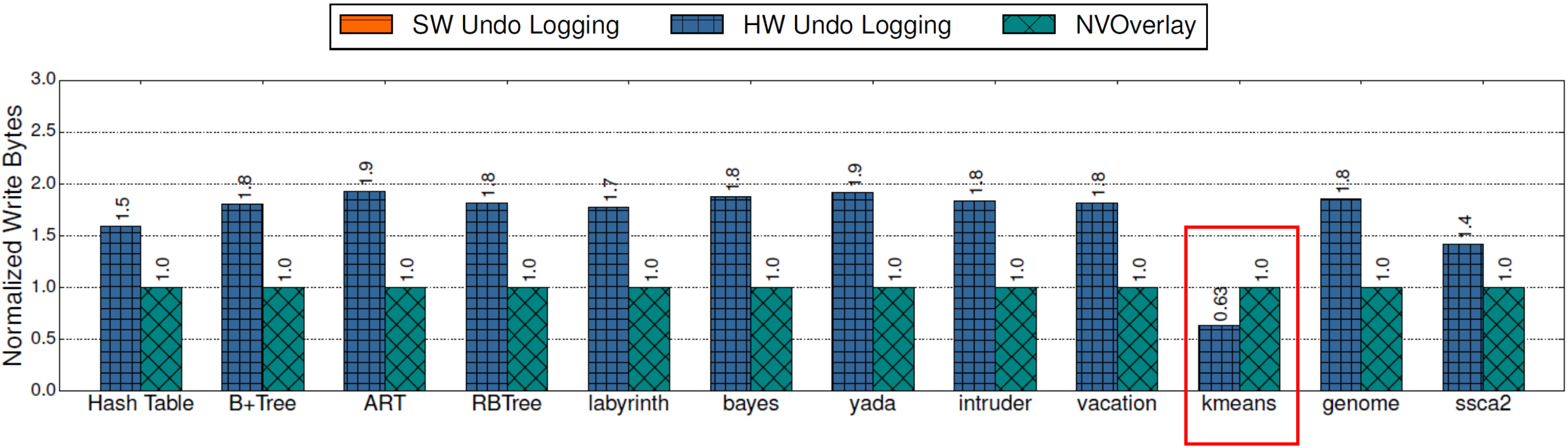
- We evaluate NVOverlay using zsim on STAMP and data structure benchmark
  - 16 cores
  - Application runs on DRAM
  - Snapshot is saved to simulated NVM device
- Compare with software and hardware undo logging in this presentation

Results: Normalized Execution Cycles



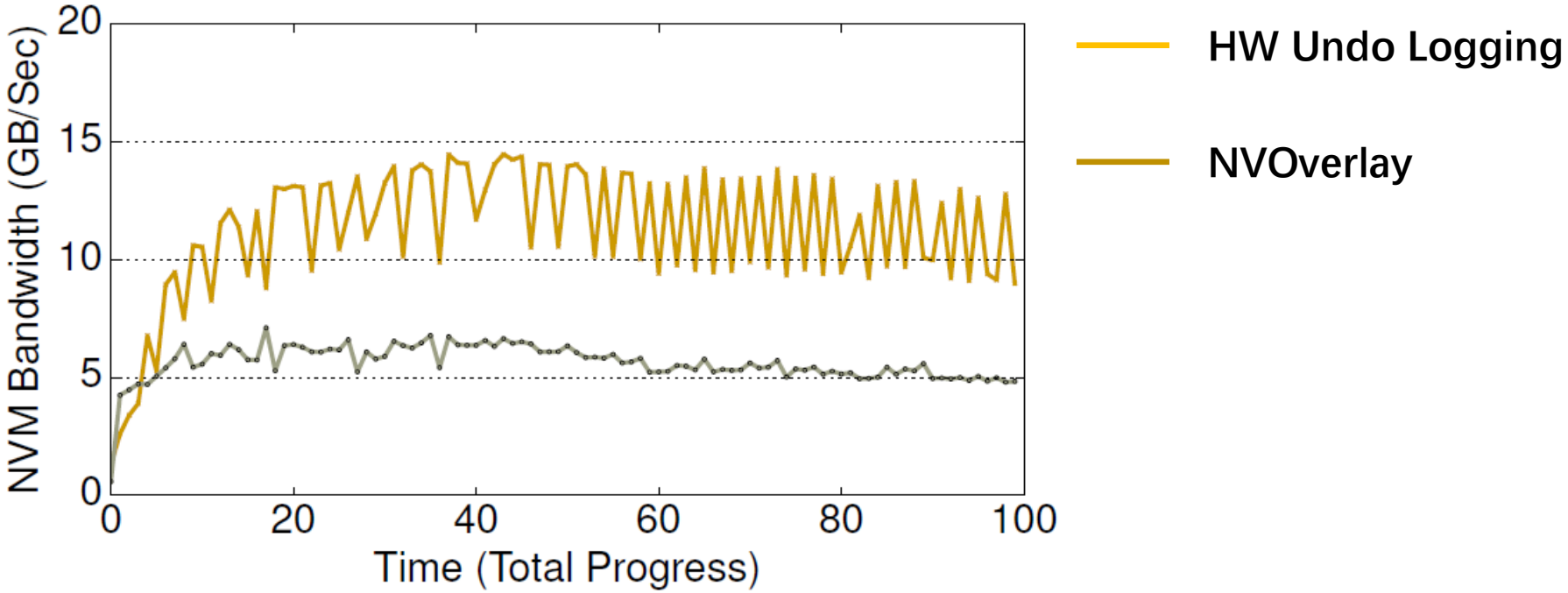
Conclusion: NVOverlay does not incur cycle overhead in most cases

Results: Normalized Write Amplification



Conclusion: NVOverlay effectively reduces write amplification (by at most 2x) compared with logging

Results: Bandwidth Consumption



(a) BTree, 1M Default Epoch

Conclusion: NVOverlay consumes less bandwidth, and does not generate write bursts

Summary

- Contributions
  - Designed **Consistent Snapshot Tracking (CST)** frontend for capturing incremental changes to the address space
  - Designed **Multi-snapshot NVM Mapping (MNM)** backend for managing multiple versions of data as snapshots
  - Proposes a distributed, coherence-based architecture for tracking global dependency using **Lamport vector clocks**