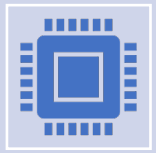
Cache-Coherent Accelerators for Persistent Memory Crash Consistency

Ankit Bhardwaj, Todd Thornley, Vinita Pawar, Reto Achermann,
Gerd Zellweger, and Ryan Stutsman

THE UNIVERSITY OF UTAH®
THE UNIVERSITY OF BRITISH COLUMBIA
vmware®

Persistent Memory is transformative



Direct CPU load-store access

Avoids costly syscalls and kernel storage stack

Application managed data structure persistence

Persistent Memory programming is hard

- Random application and machine crashes
- Reasoning about data consistency after a crash
  - Example: Transfer money from one account to another account
- Most systems use WAL to recover consistency after crash

Persistent Memory programming is hard

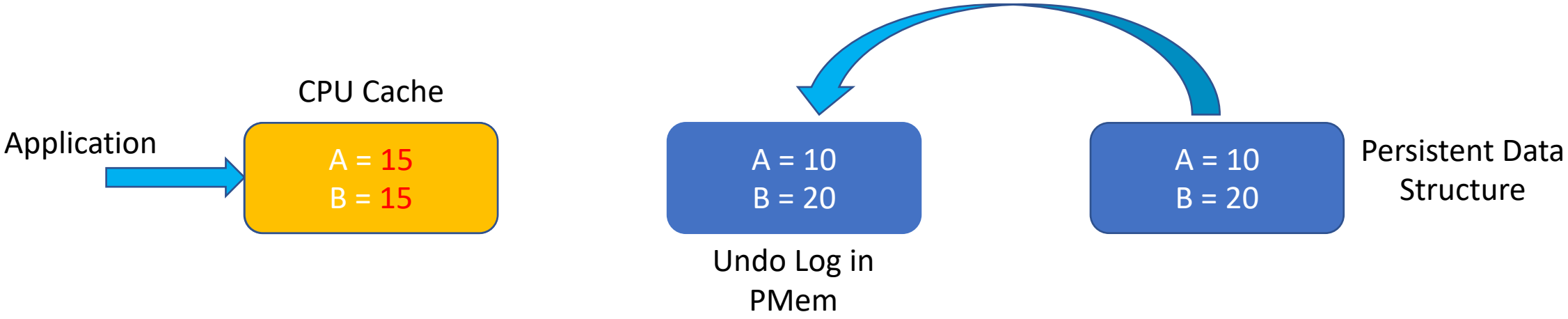
- Random application and machine crashes
- Reasoning about data consistency after a crash
  - Example: Transfer money from one account to another account
- Most systems use WAL to recover consistency after crash

Persistent Memory programming is hard

- Random application and machine crashes
- Reasoning about data consistency after a crash
  - Example: Transfer money from one account to another account
- Most systems use WAL to recover consistency after crash

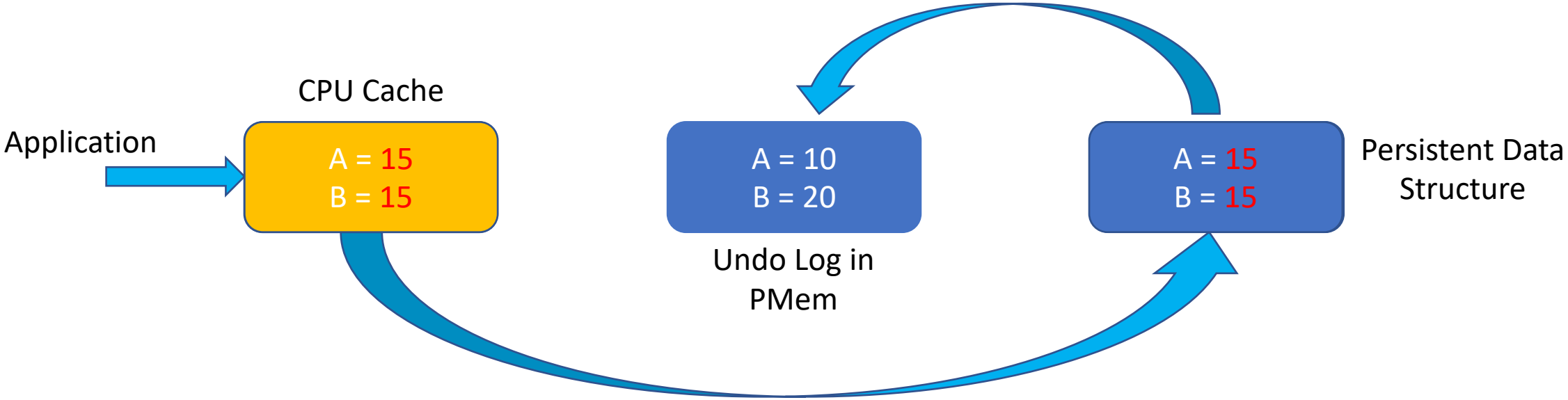
Persistent Memory programming is hard

- Random application and machine crashes
- Reasoning about data consistency after a crash
  - Example: Transfer money from one account to another account
- Most systems use WAL to recover consistency after crash



Persistent Memory programming is hard

- Random application and machine crashes
- Reasoning about data consistency after a crash
  - Example: Transfer money from one account to another account
- Most systems use WAL to recover consistency after crash



Related work

- Many schemes interpose on stores, cutting into PM benefits
- Most such schemes add WAL before the operations
  - Write code from scratch using PMDK, etc.
  - Use automated compiler-based approaches
- Some schemes use page-faults to track changes
  - Page-faults are costly (> µs for x86 hardware)
  - Such schemes have high write-amplification (4KB page size)

Goal

- Transform volatile structures into crash-consistent persistent structures, ex: Hash tables, trees, file system metadata structures, etc.
- No compiler or code changes
- Low write amplification
- Performance near to direct PM access by making the WAL and data write async

Track the data structure changes using cache-coherence messages

Why now?

Possible without any CPU hardware changes

- Cache-coherent accelerators are becoming mainstream
  - Enzian, Intel Harp, VMware PBerry, etc.
- Multiple ongoing standardizations for cc-Accelerators
  - CAPI, CXL, etc.
- Low latency to access cache-coherent devices
  - Near remote NUMA node access latencies

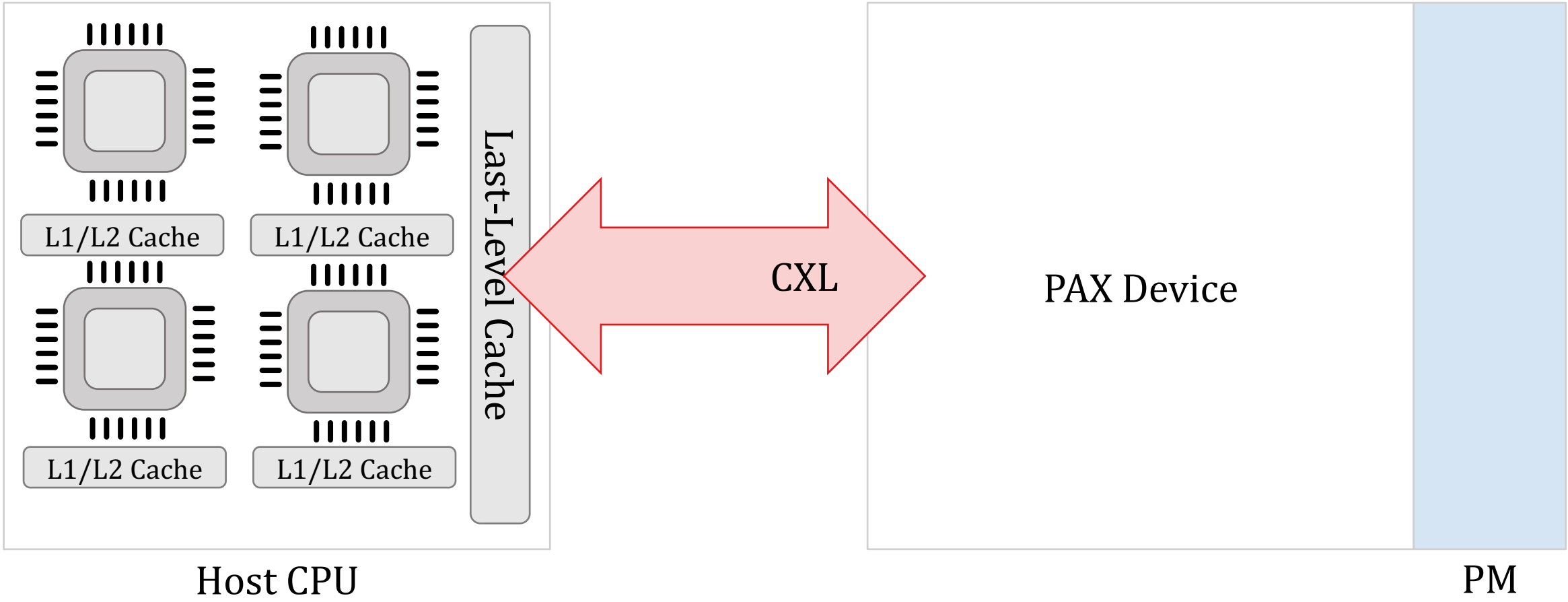
Example of PAX programming model

```
// Use the memory of the PAX device:
allocator = Snapshotter::map_pool("pht")
// Instantiate a HashMap (can be regular DRAM DS):
pht = Persistent<HashMap>(&allocator);

// If CPU cache misses, fetches from device with CXL
pht.get(a, 100)
// Writes: CPU requests cache-line in modified state
// from PAX dev. (which now also logs the write)
pht.insert(b, 200);

// Group commit: write-back, flush undo-log to PMem
pht.persist()
```

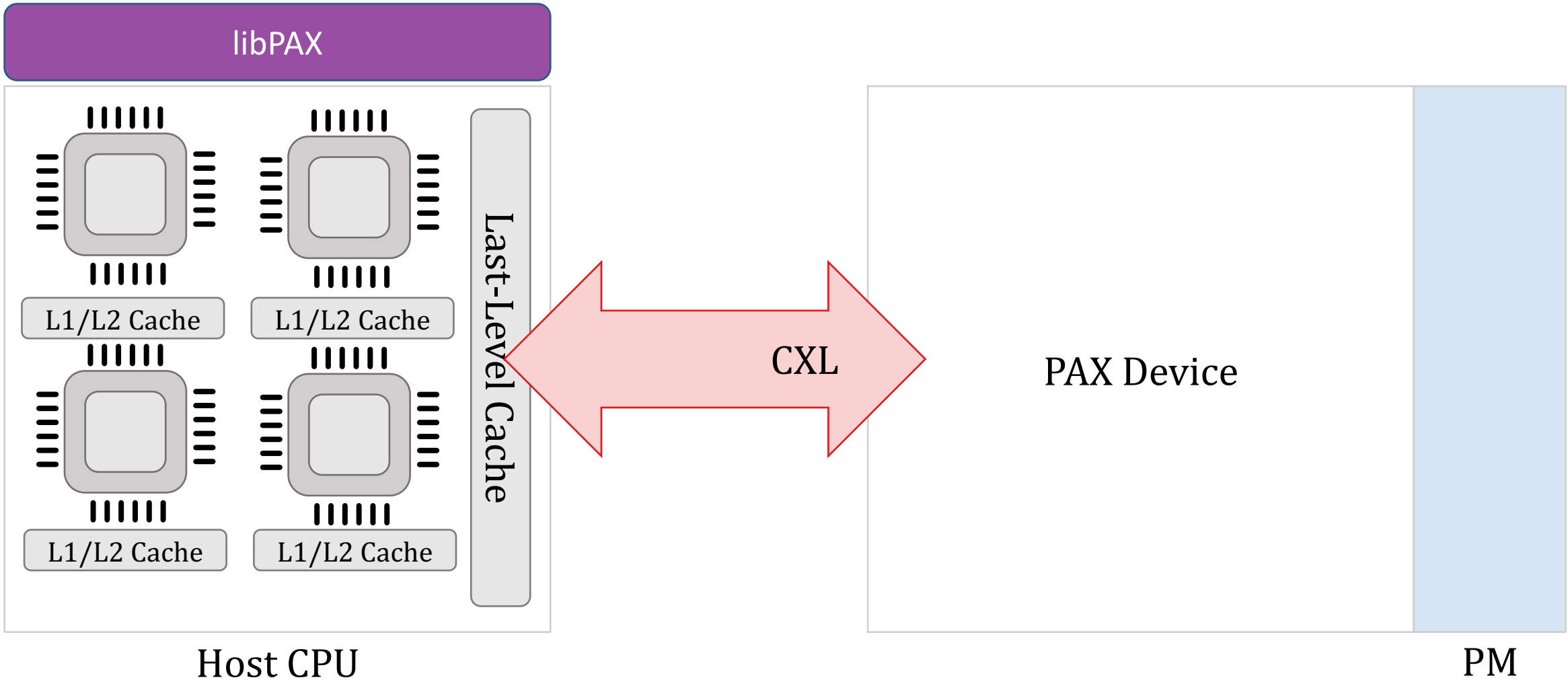
Asynchronous logging and writeback

- libPAX/Application issues persist() after a batch of operations
- WAL entries are written to PM asynchronously
- Writeback is done to PM directly once WAL entries are persisted
- CPU cache is reused within a batch on operations

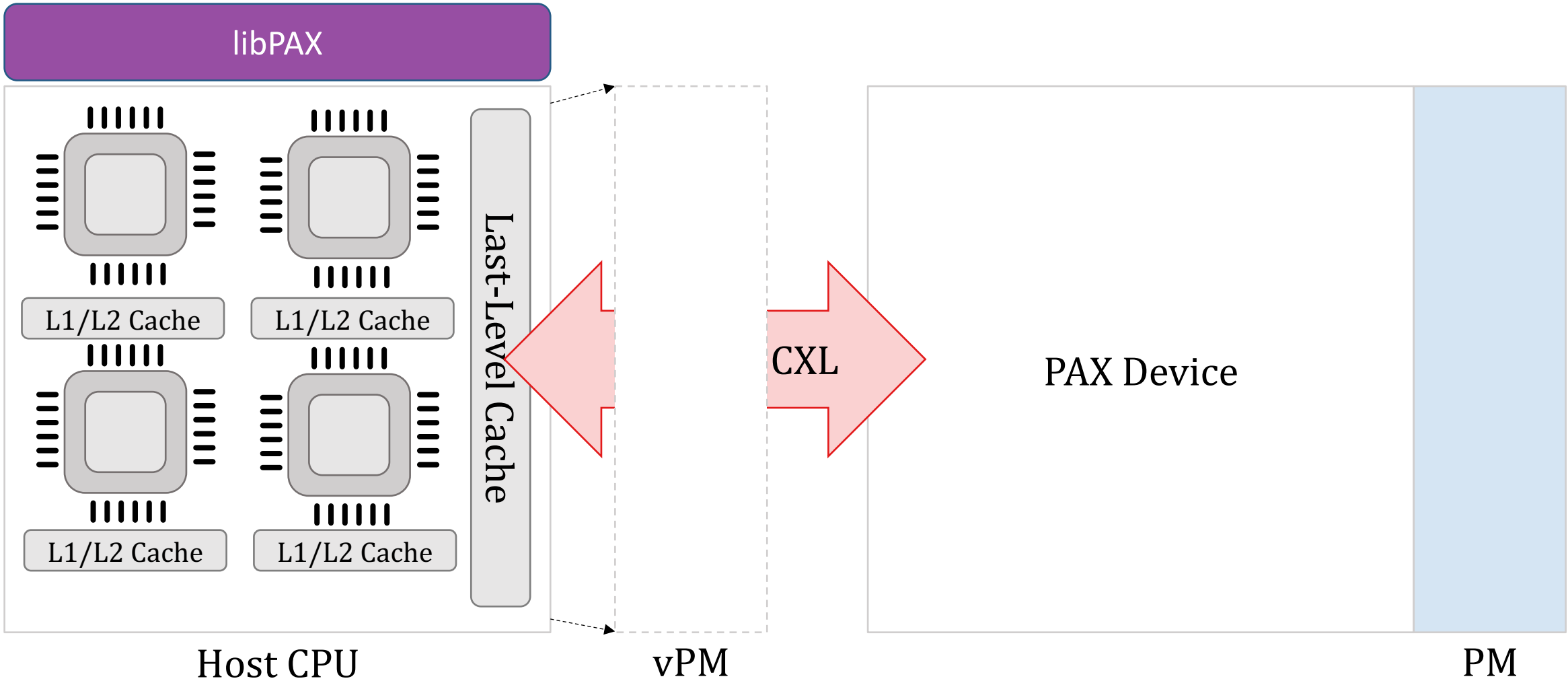
PAX overview



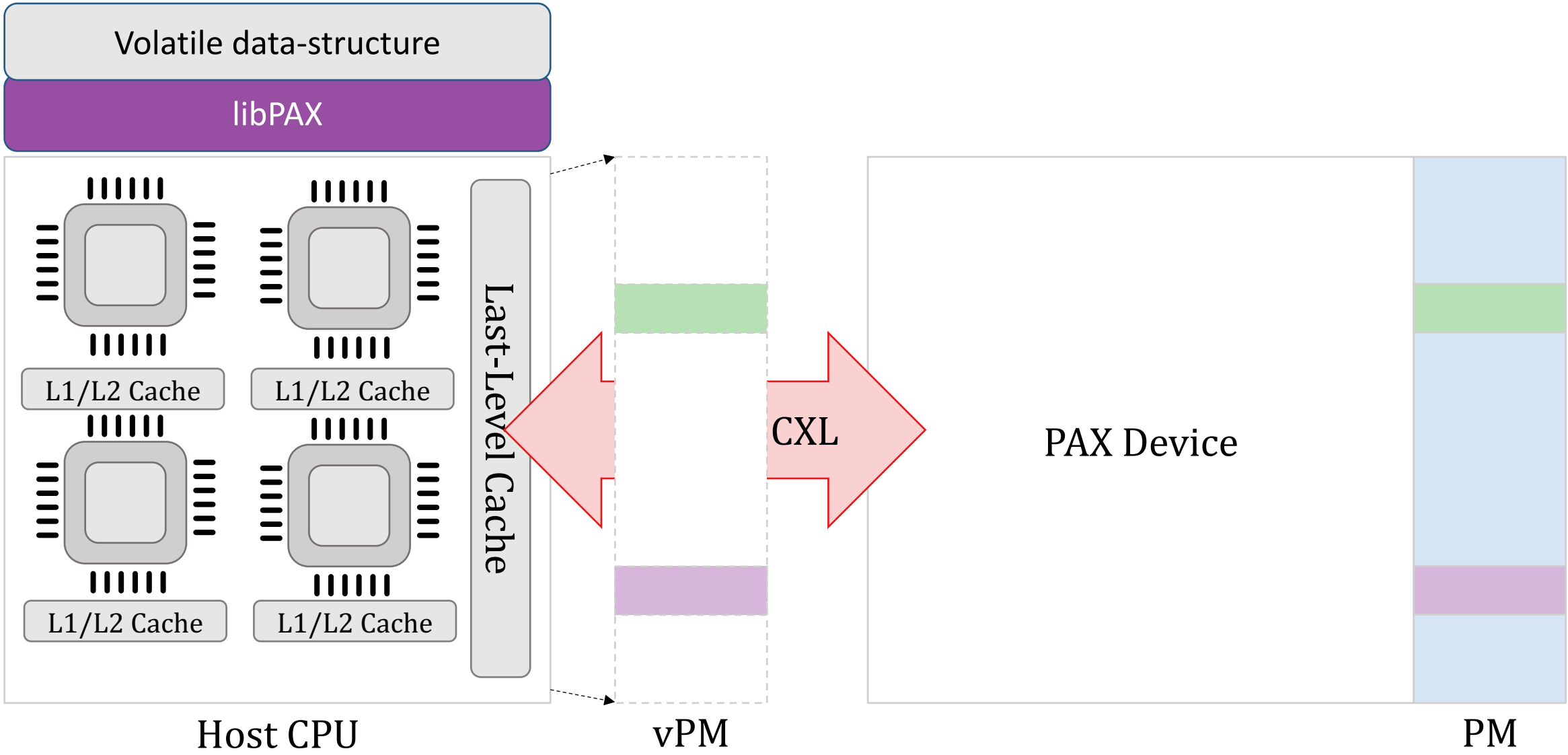
PAX overview



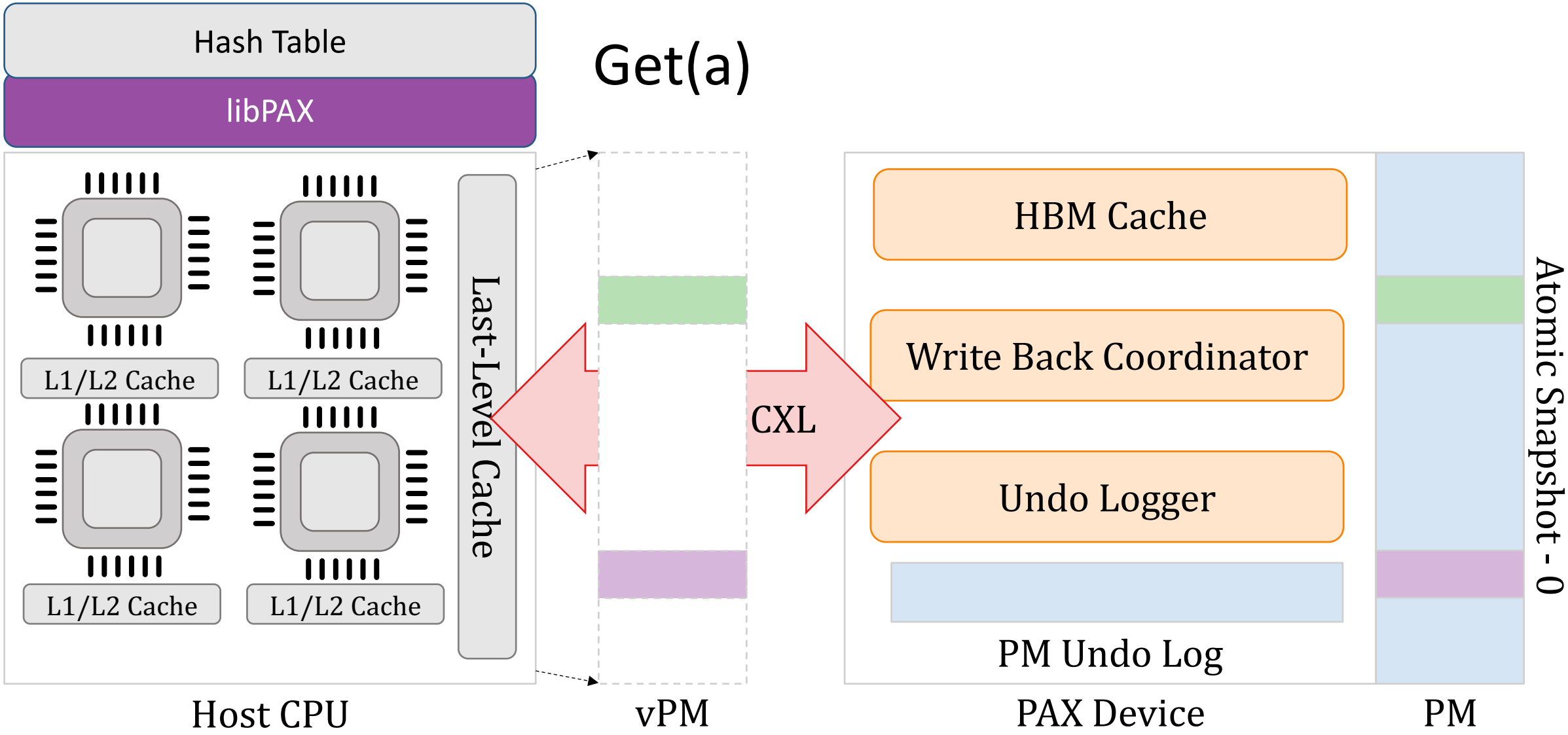
PAX overview



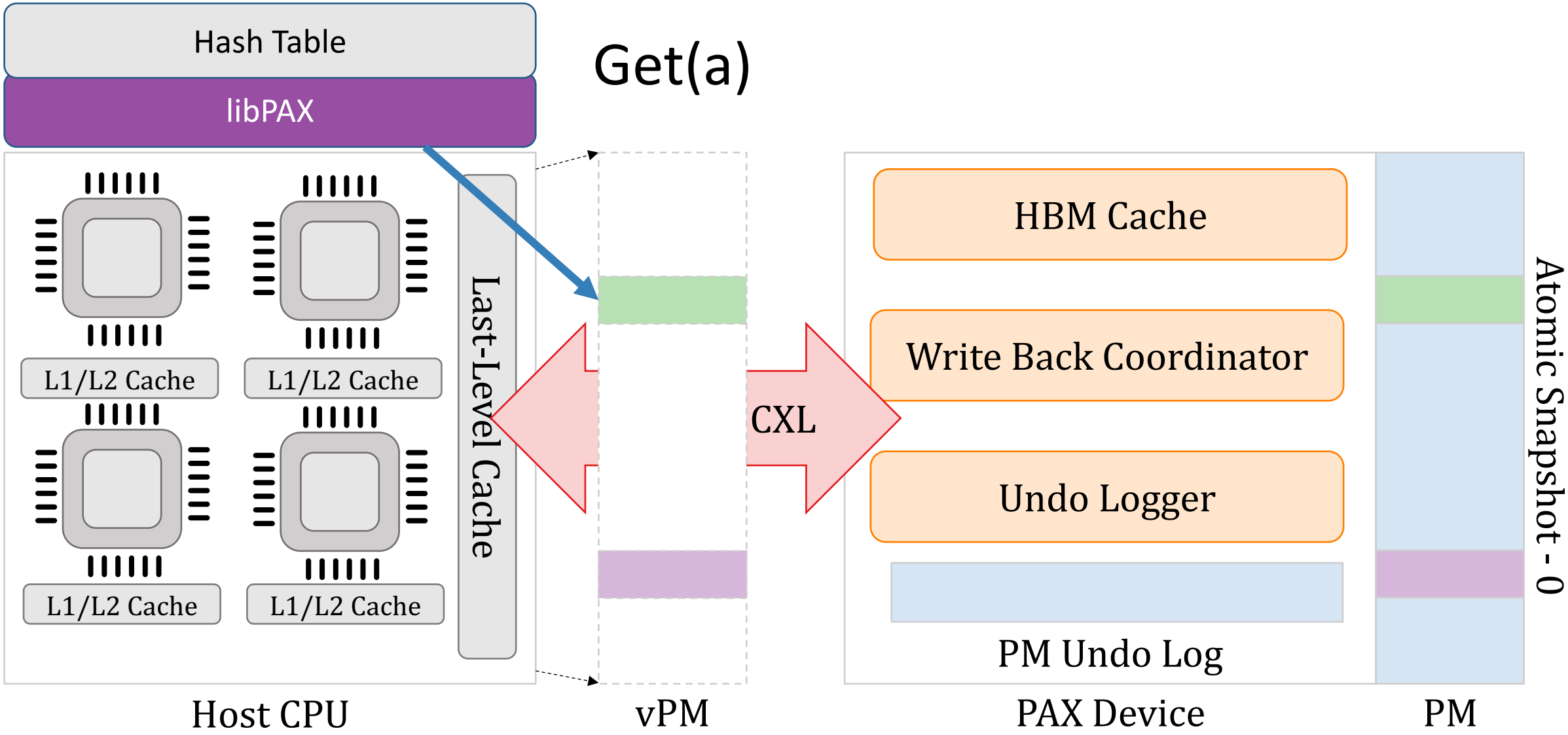
PAX overview



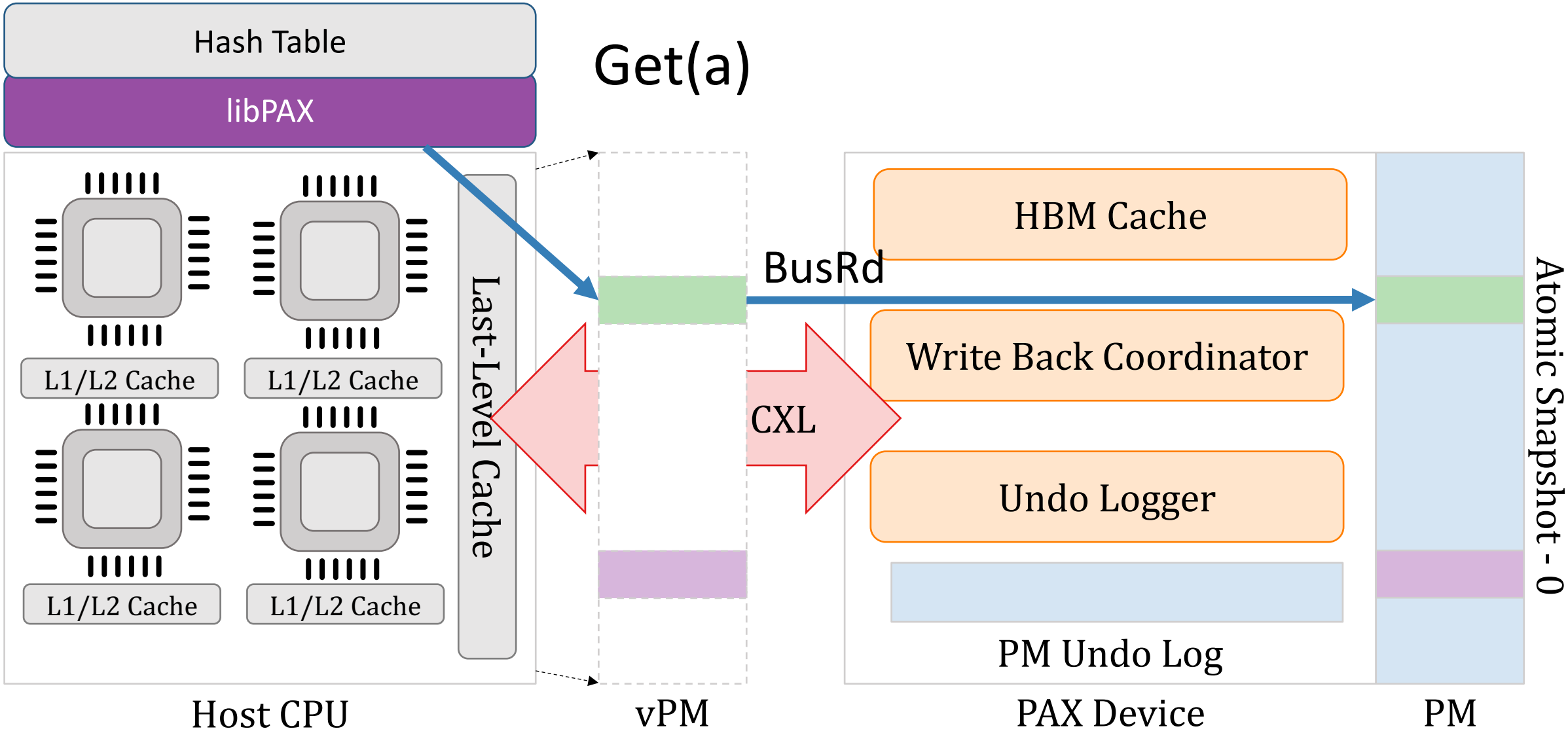
Example walkthrough – Read operation



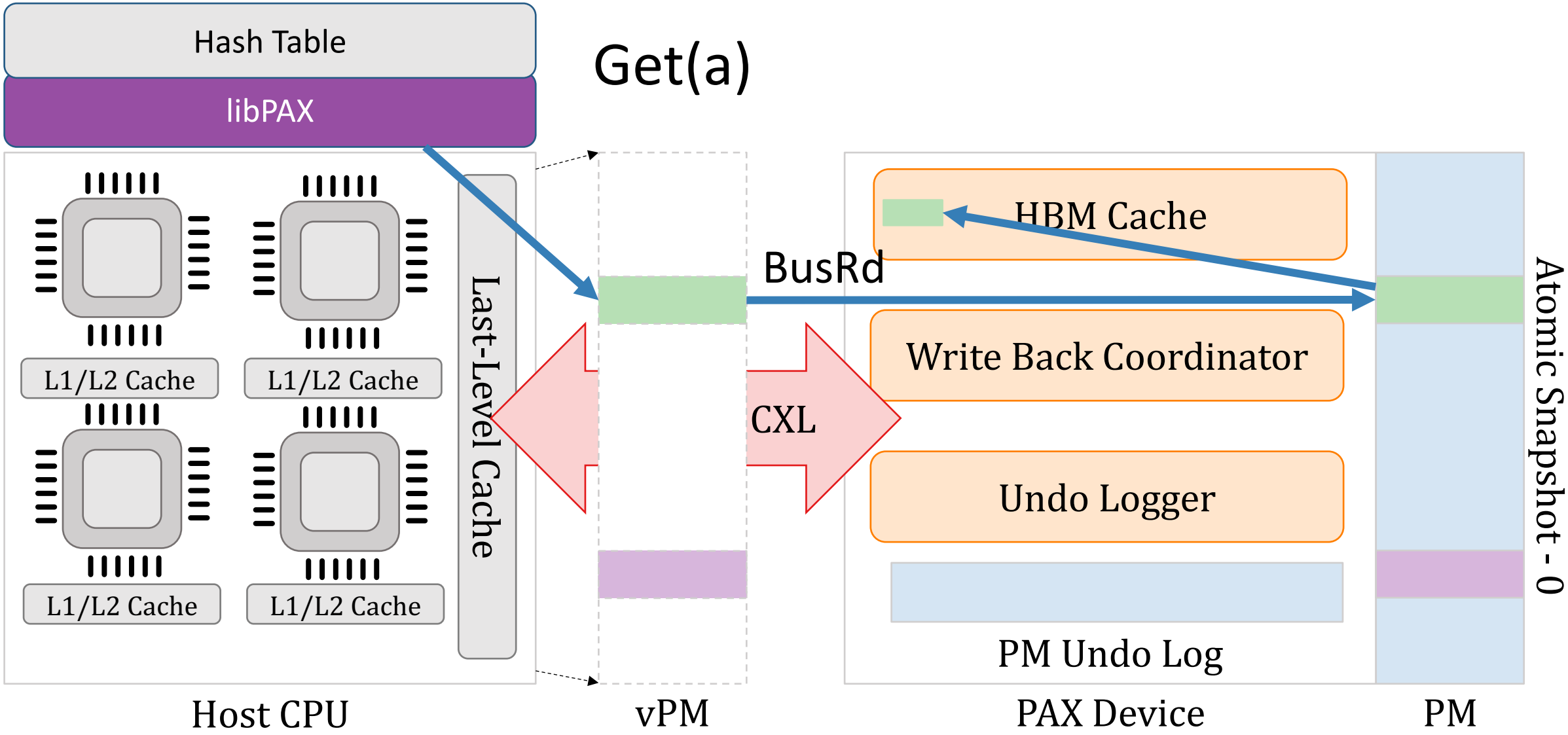
Example walkthrough – Read operation



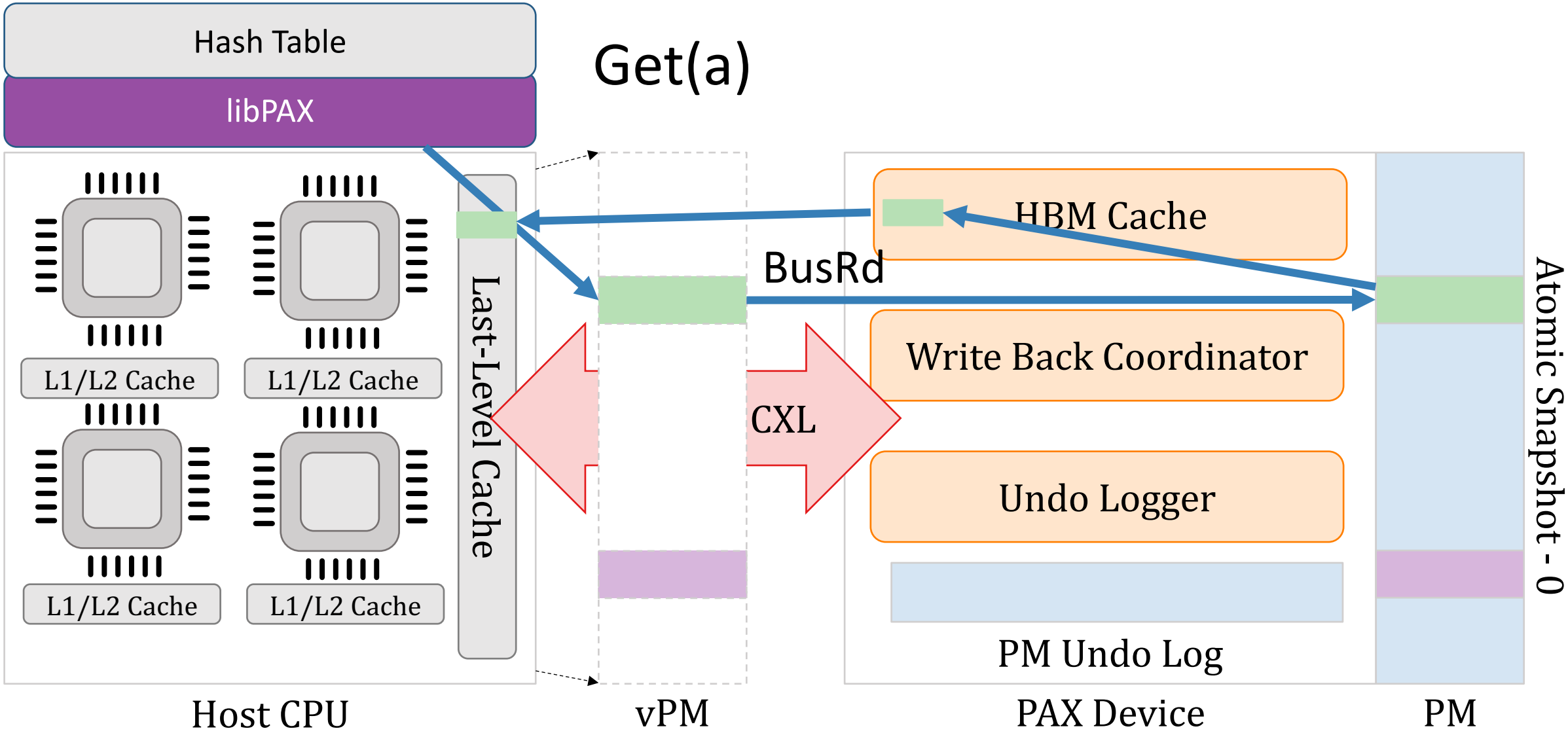
Example walkthrough – Read operation



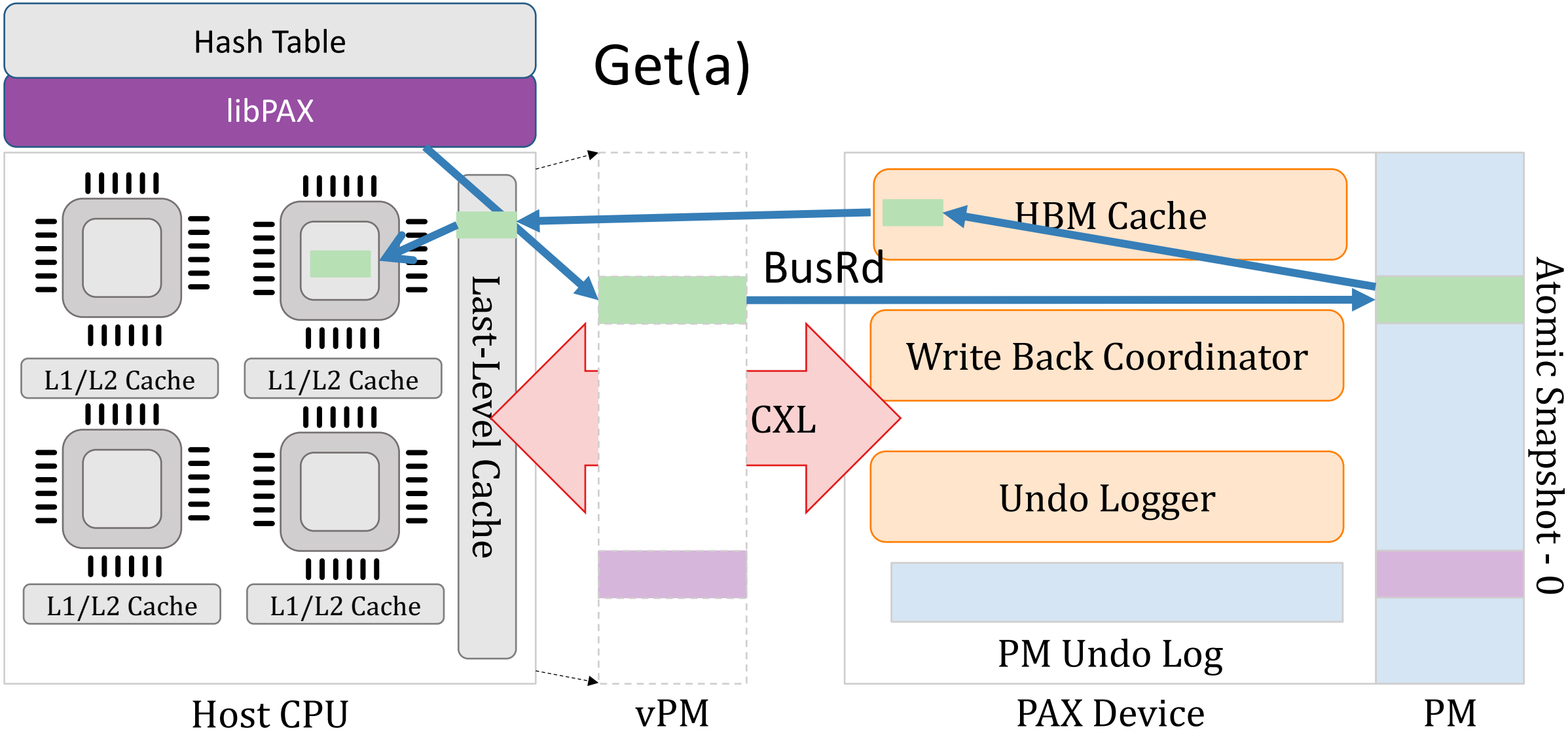
Example walkthrough – Read operation



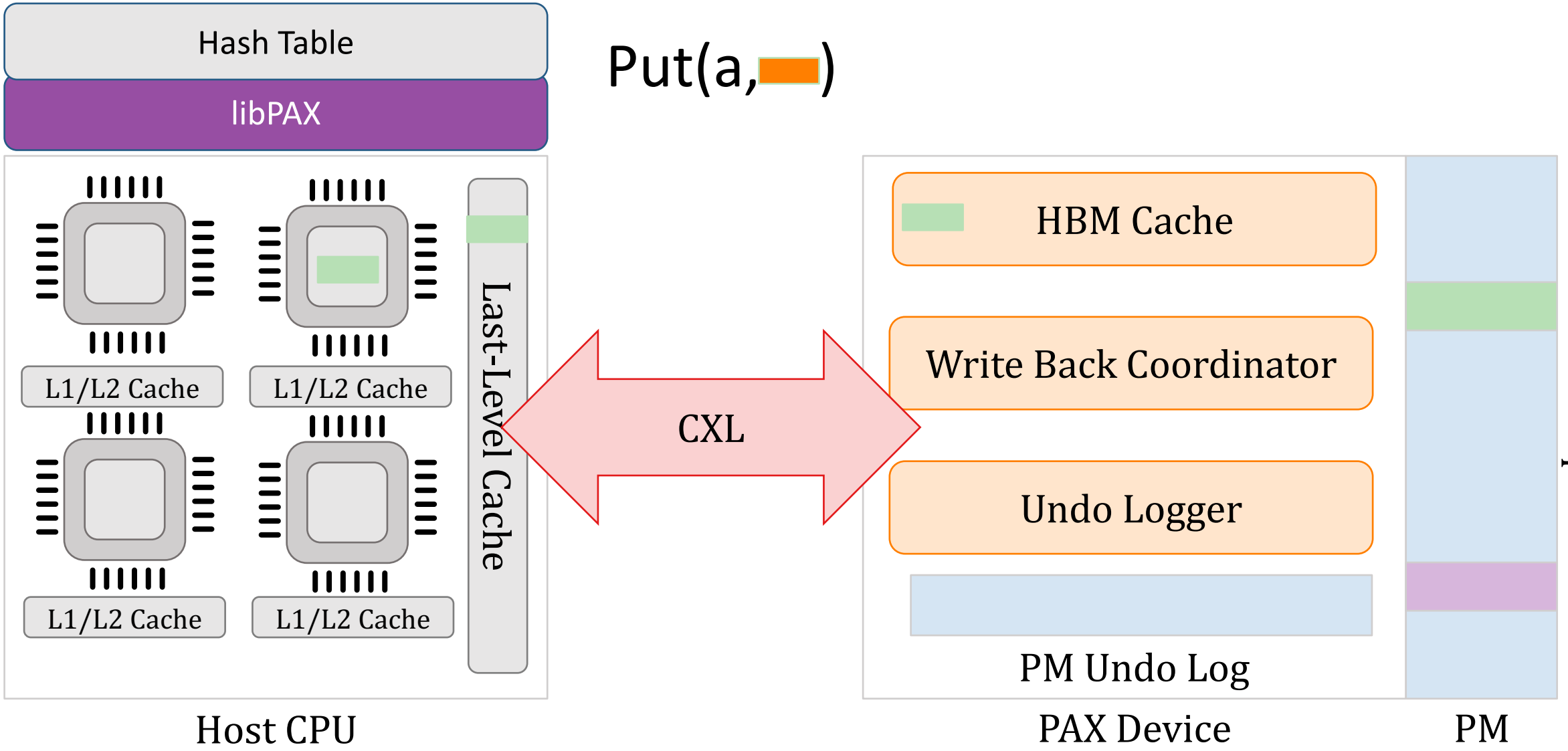
Example walkthrough – Read operation



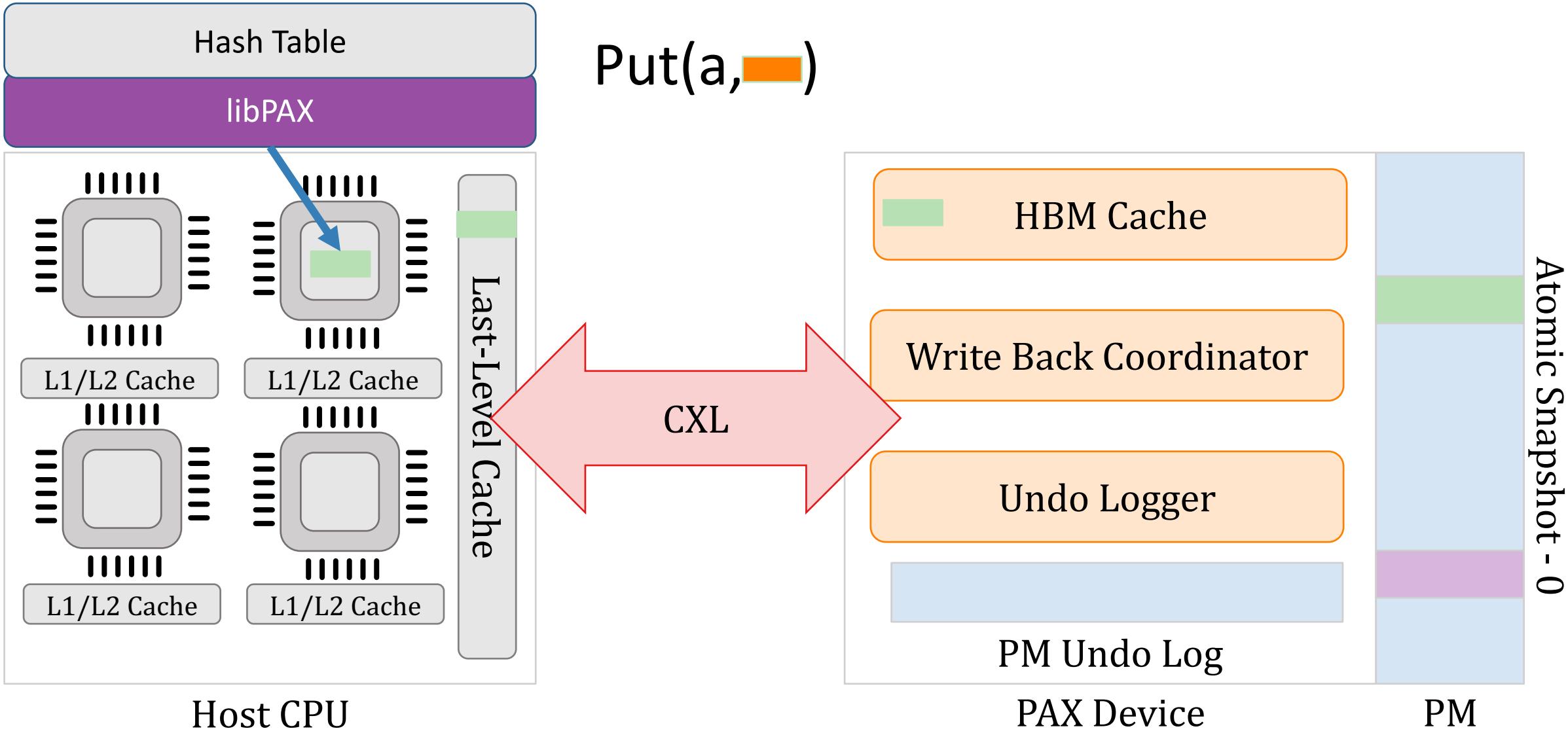
Example walkthrough – Read operation



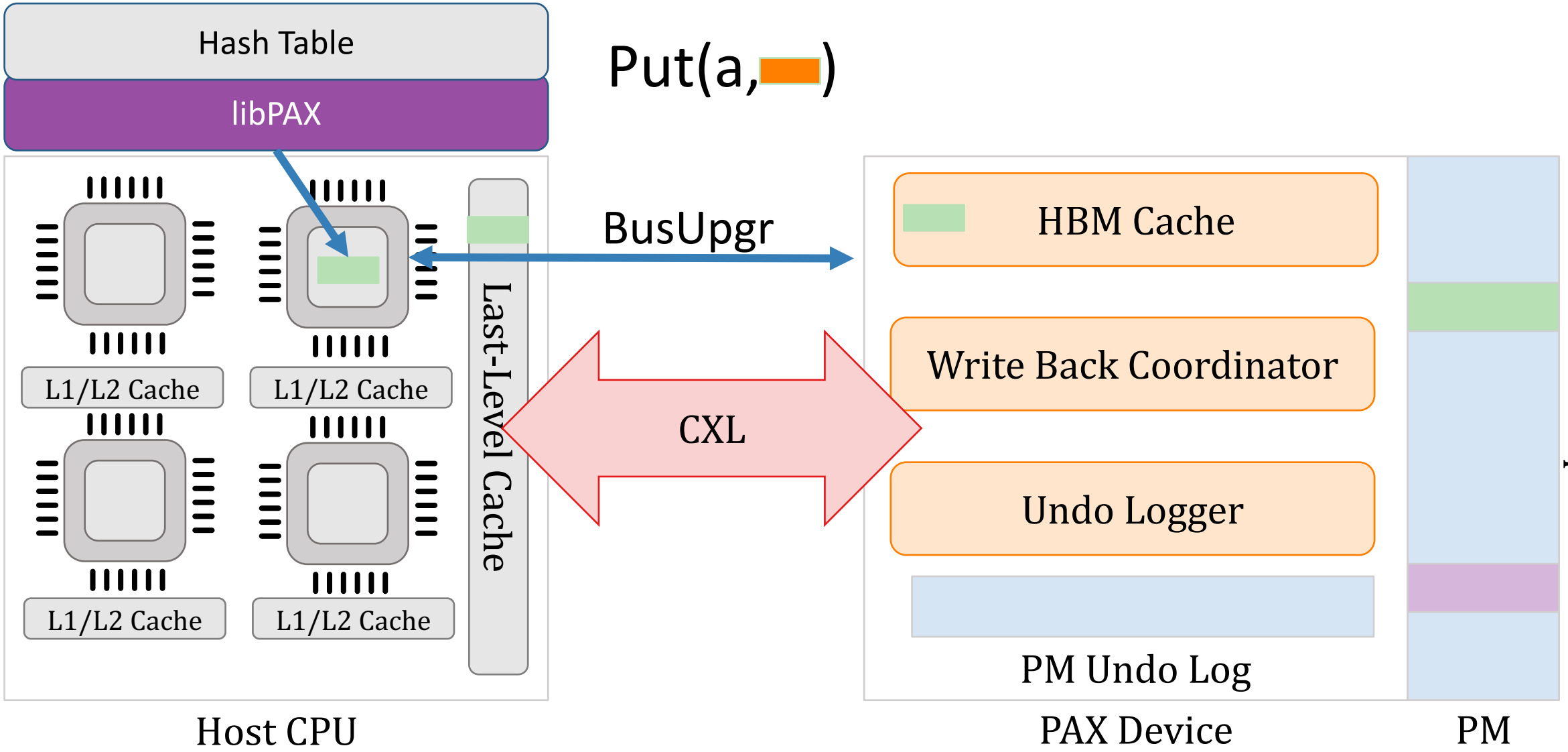
Example walkthrough – Write operation



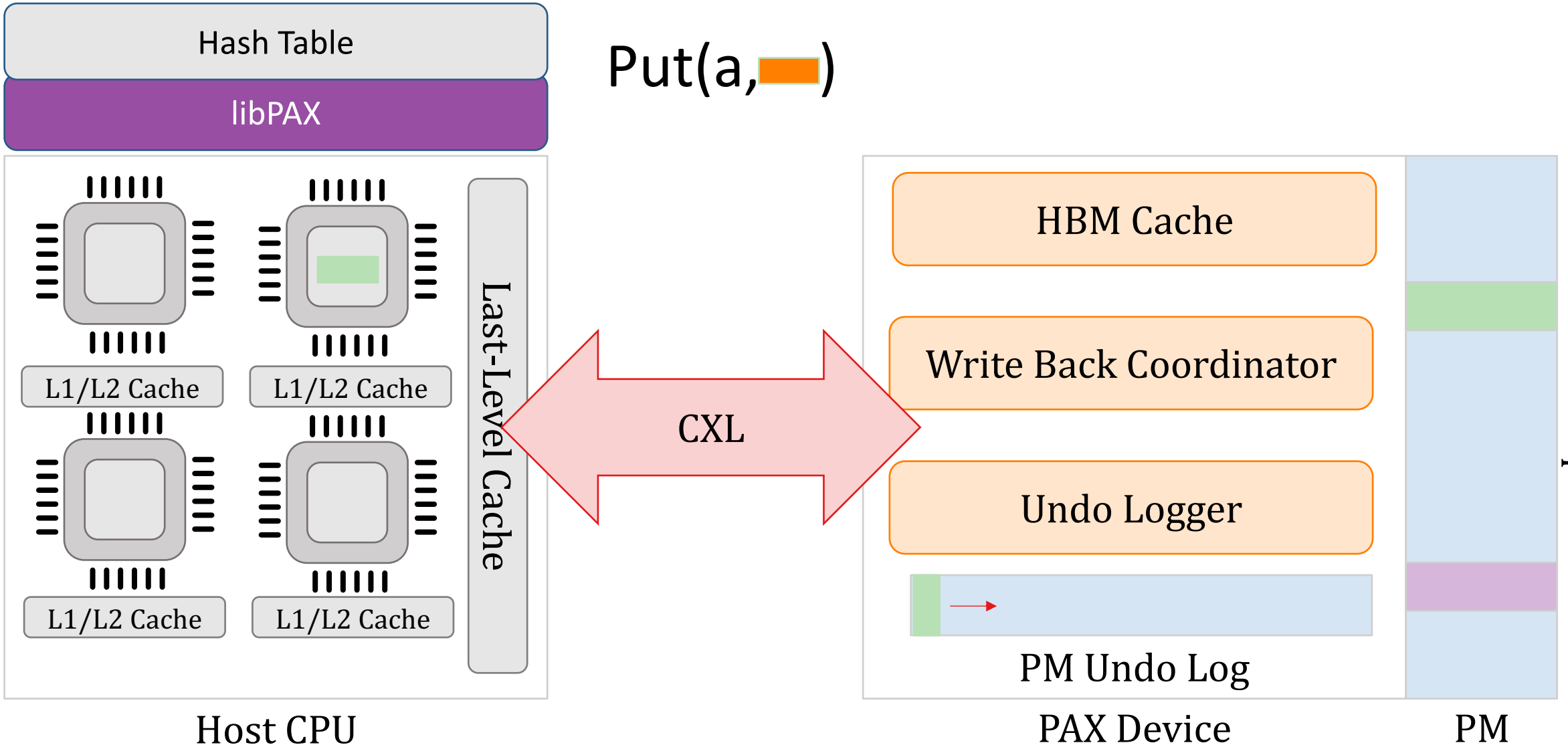
Example walkthrough – Write operation



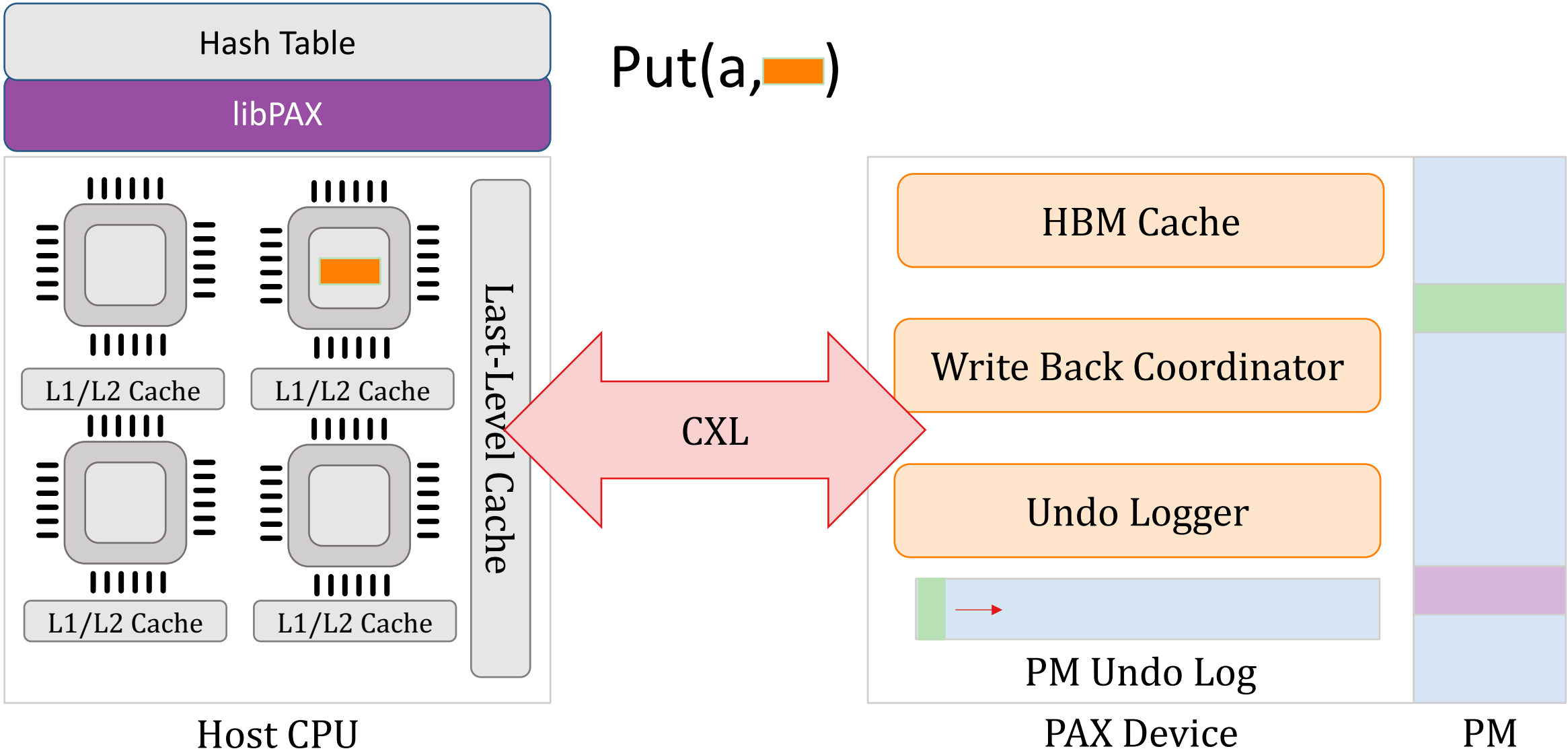
Example walkthrough – Write operation



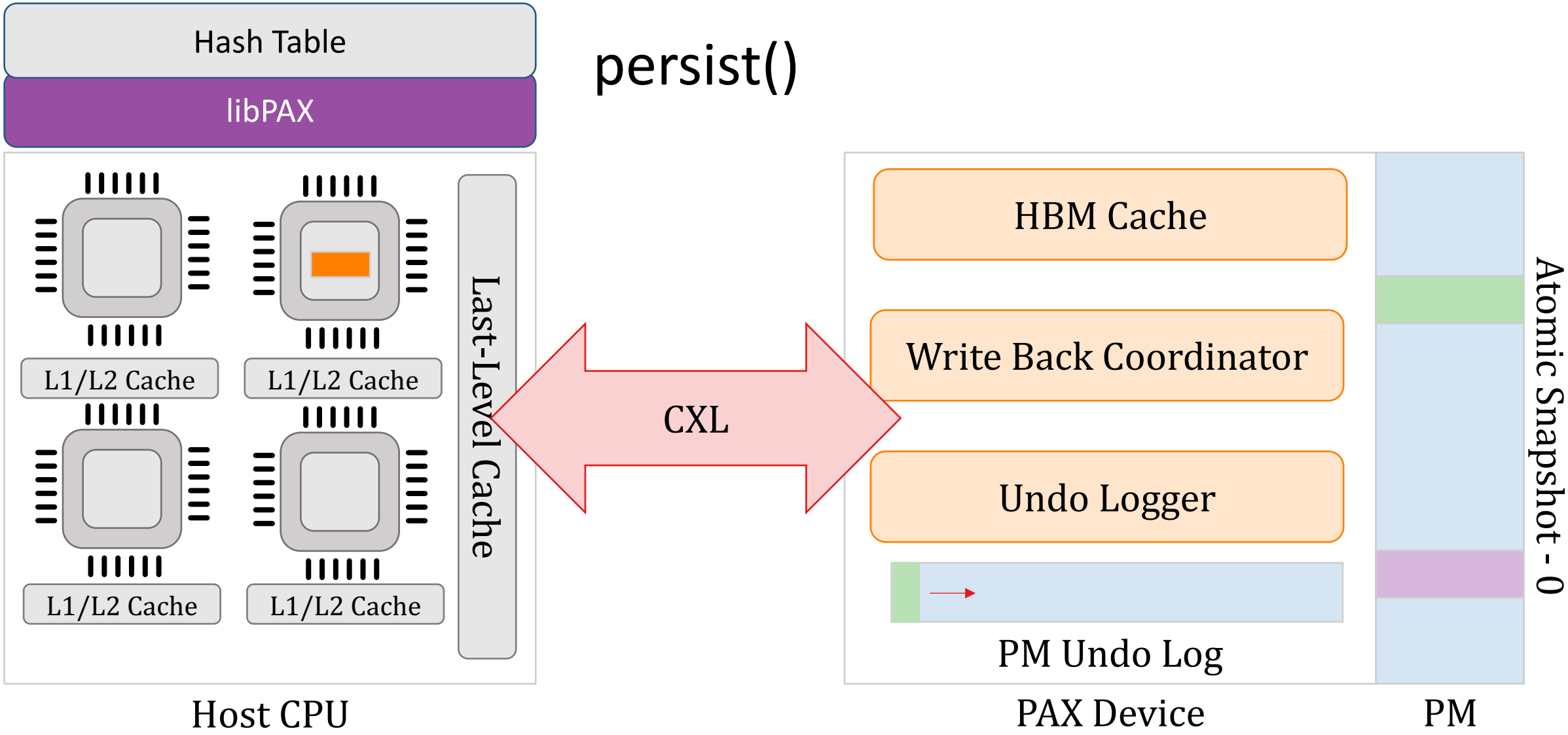
Example walkthrough – Write operation



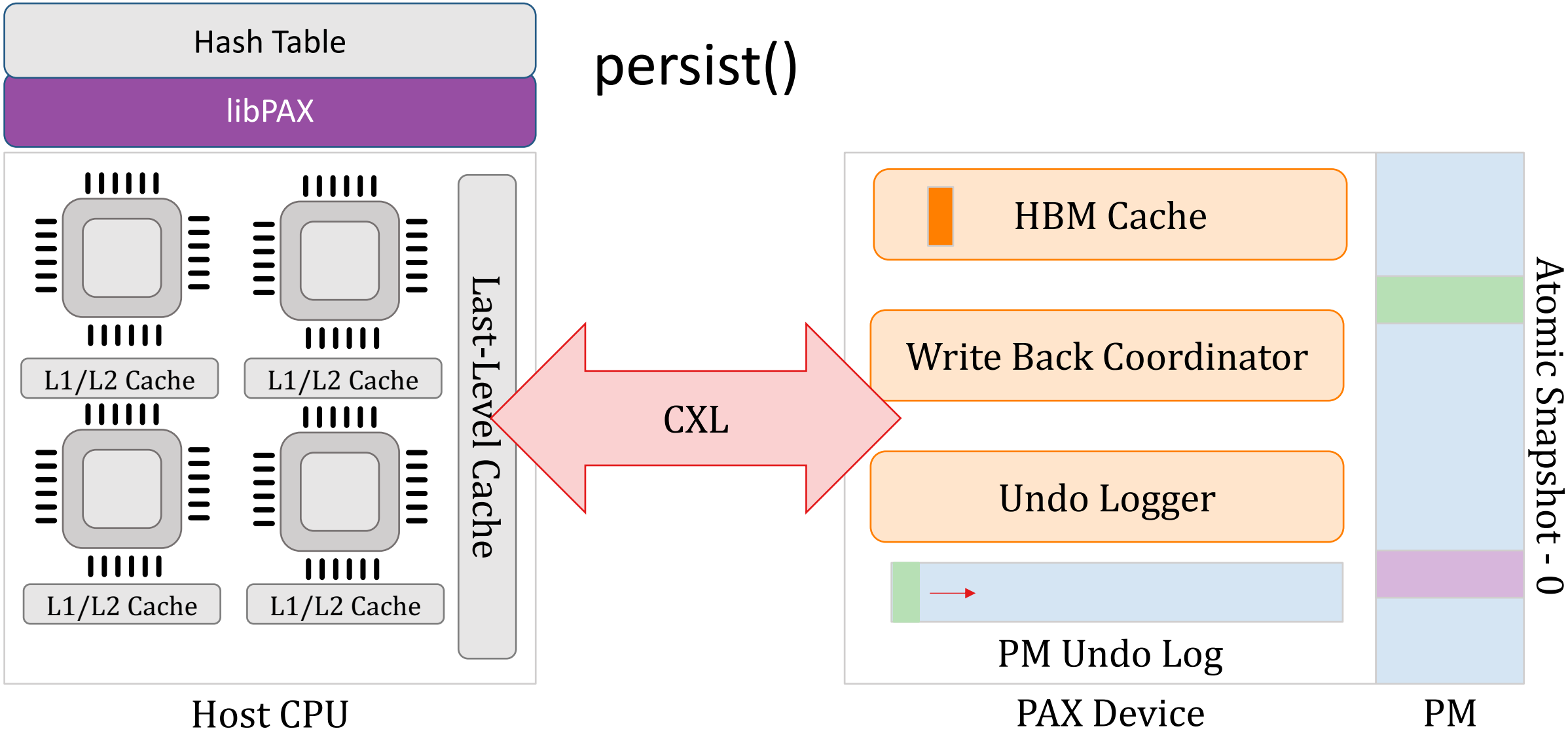
Example walkthrough – Write operation



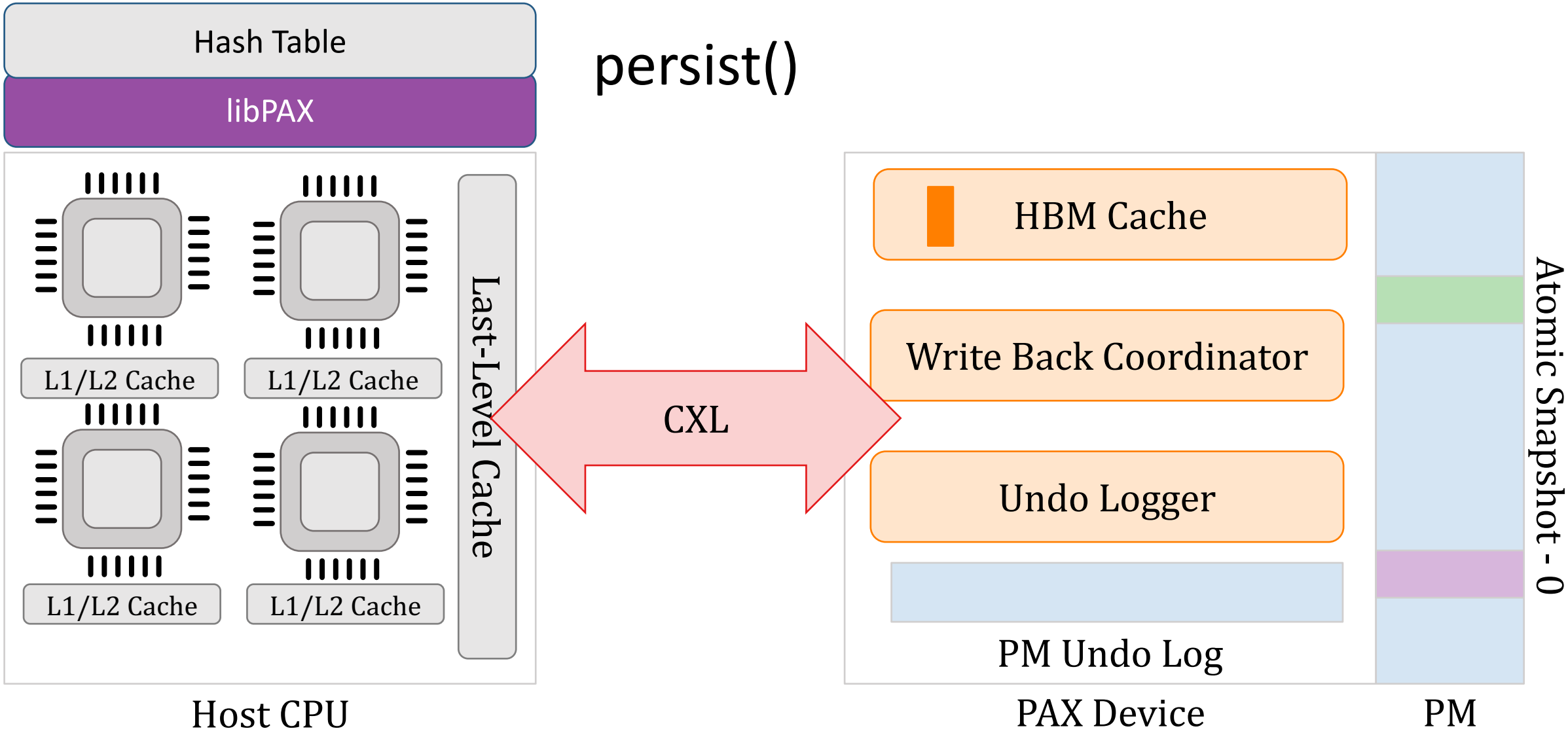
Example walkthrough – Writeback



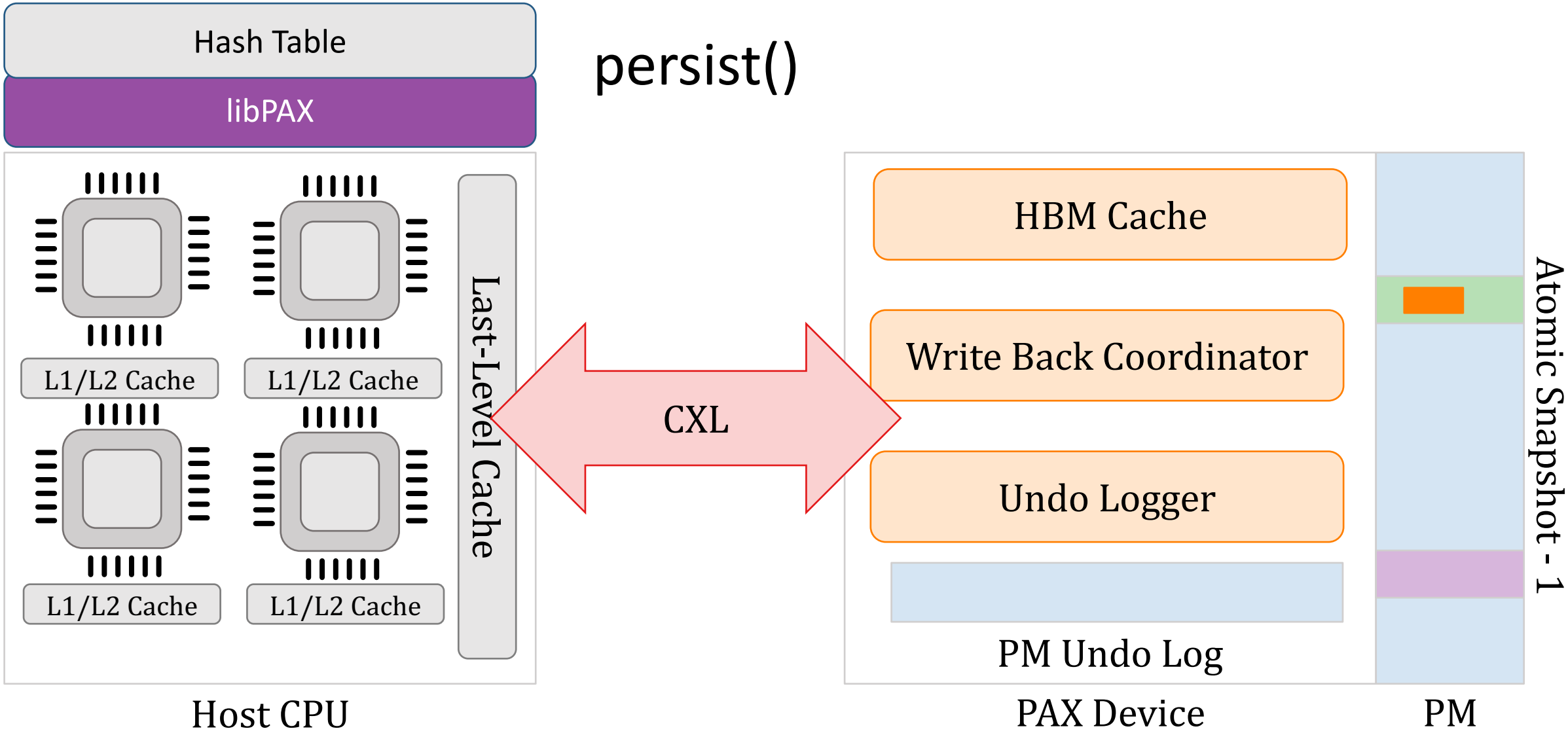
Example walkthrough – Writeback



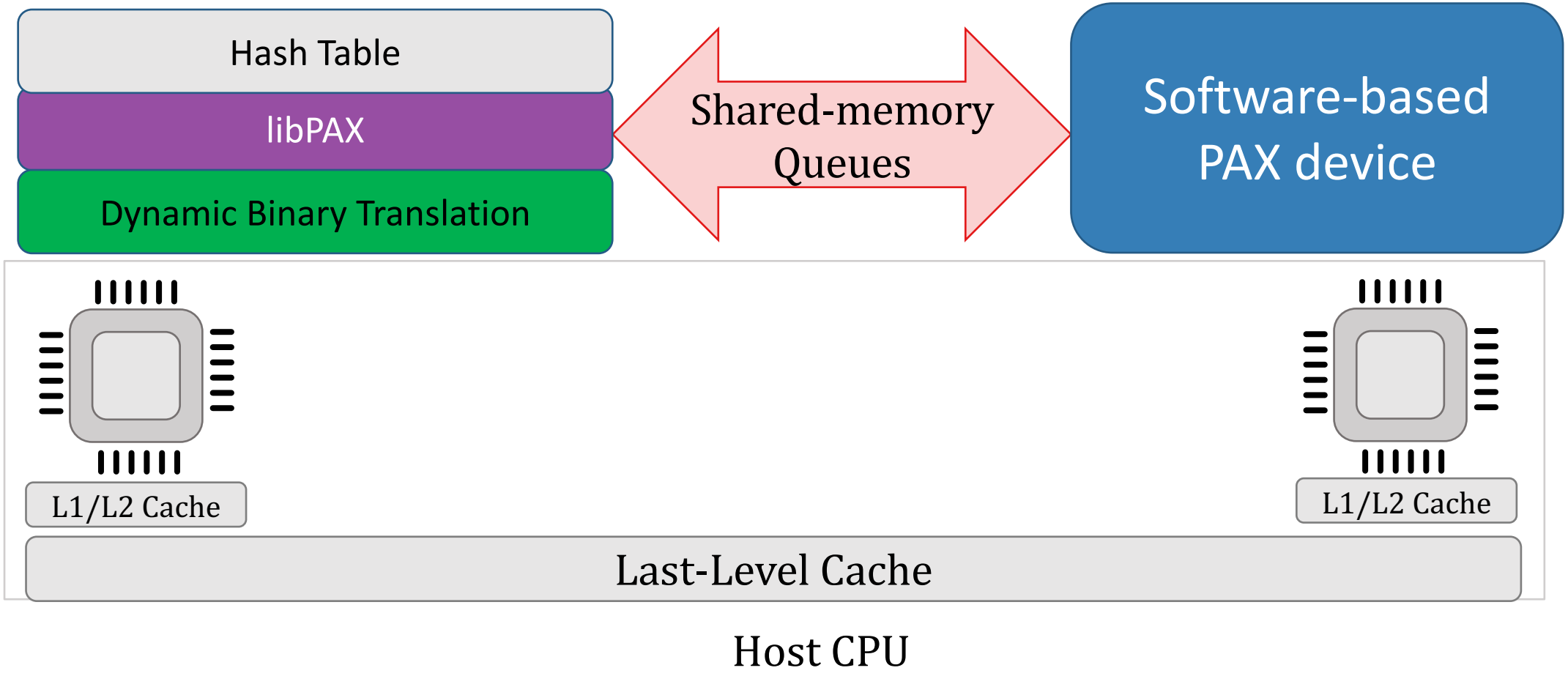
Example walkthrough – Writeback



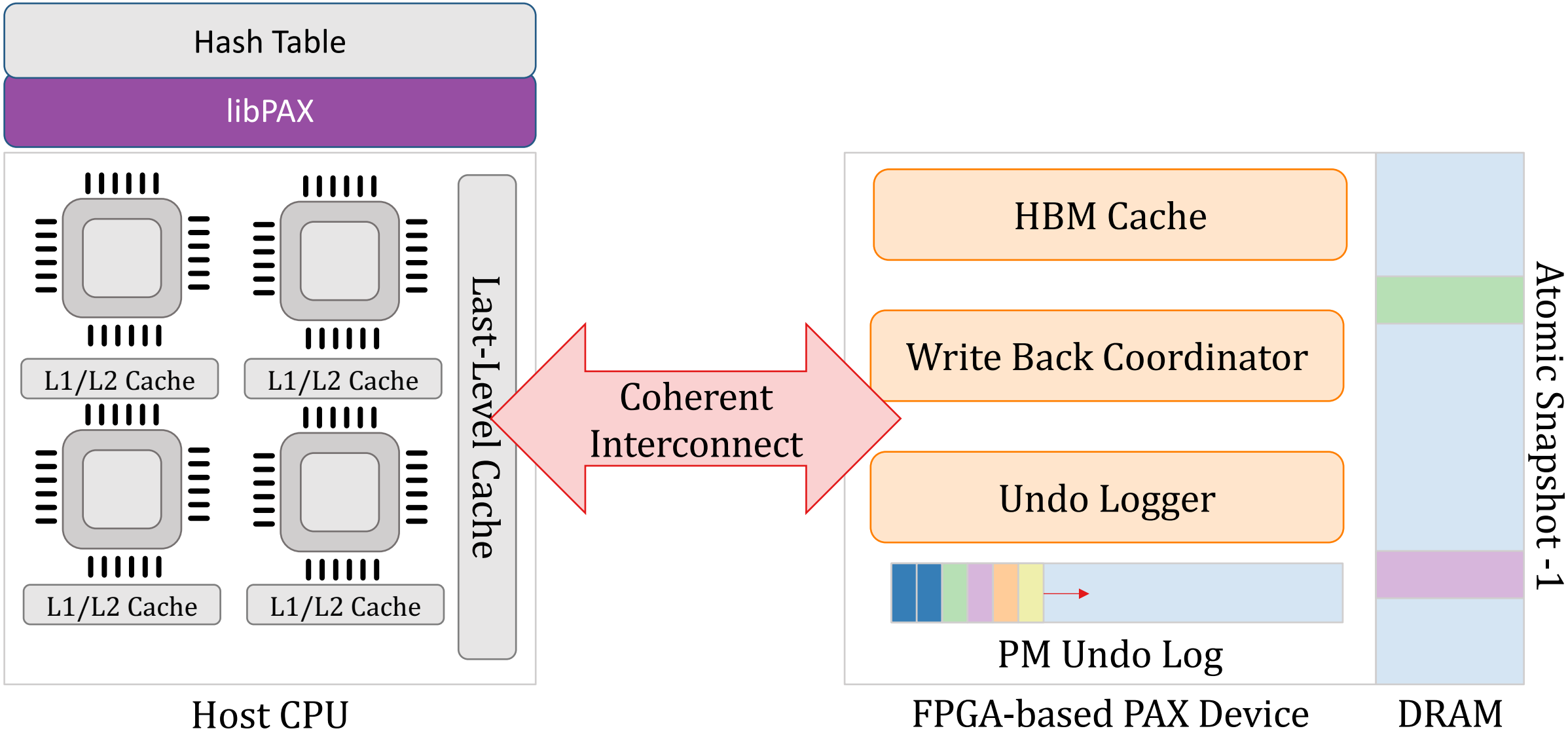
Example walkthrough – Writeback



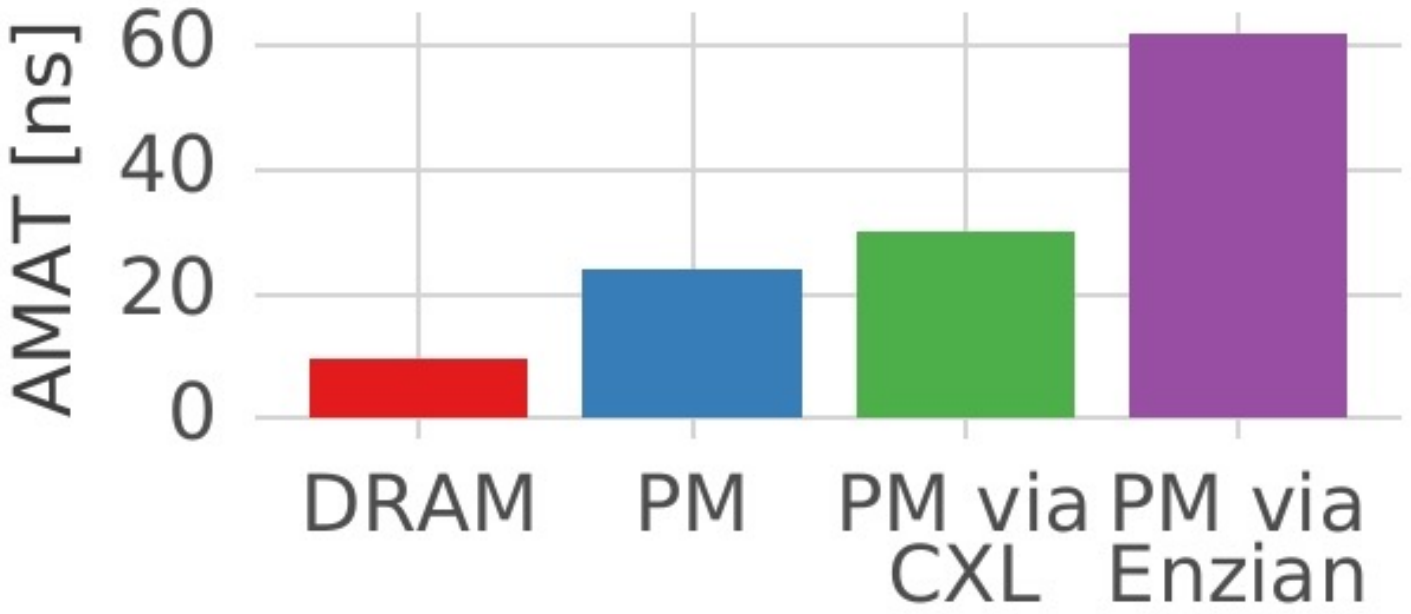
Software-Simulated CXL Accelerator



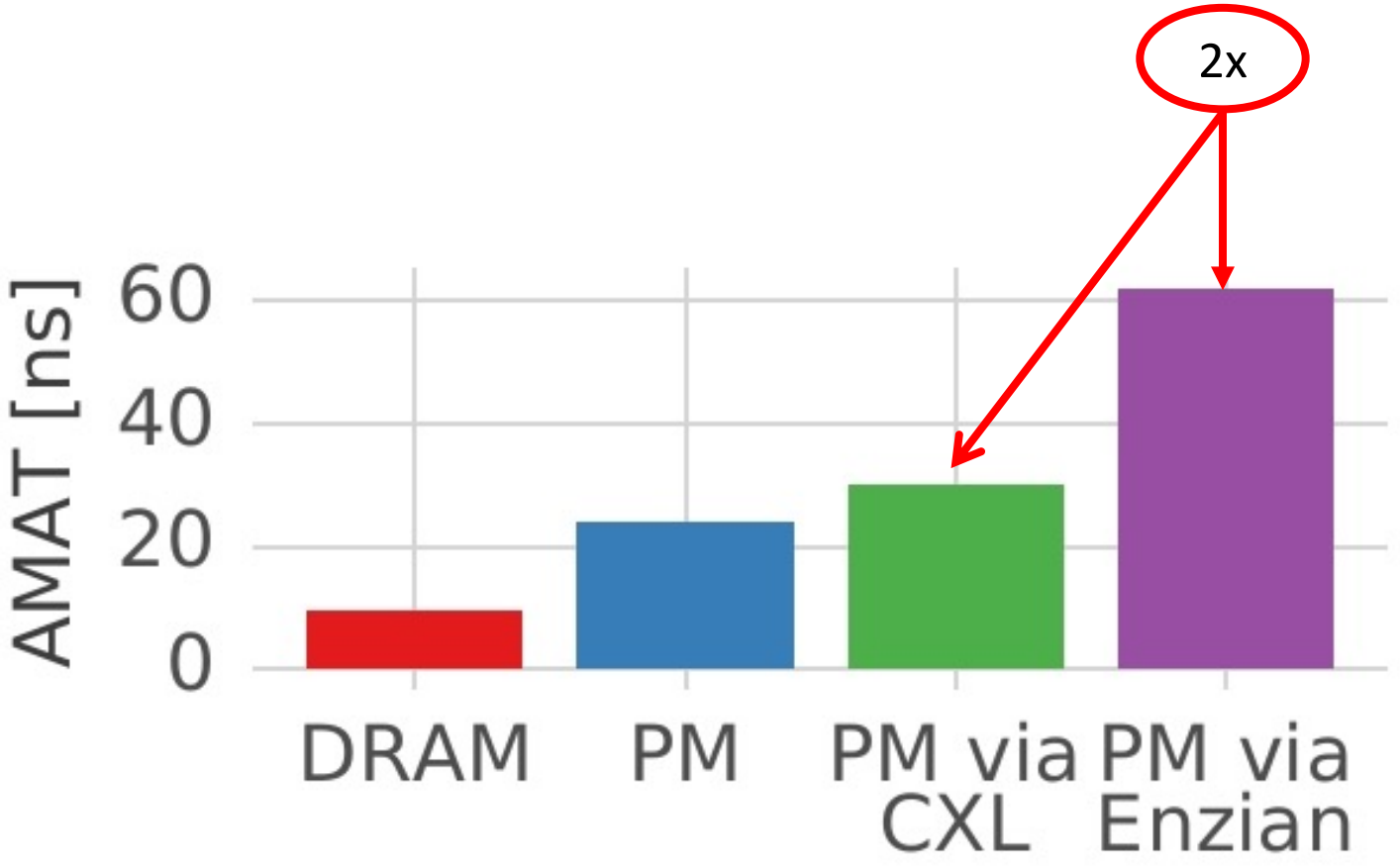
Cache-Coherent FPGA, Enzian



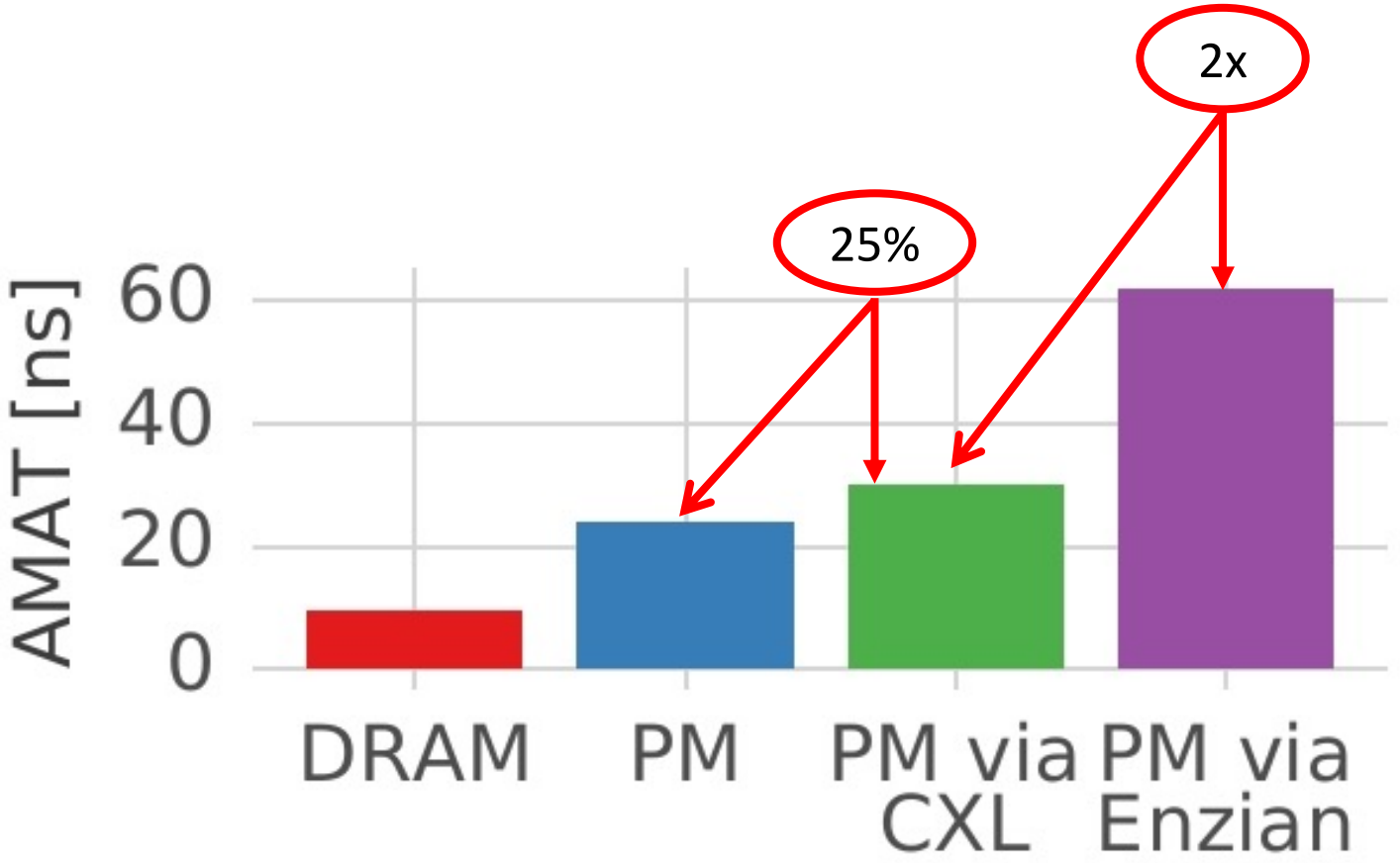
Expected performance with ccFPGA



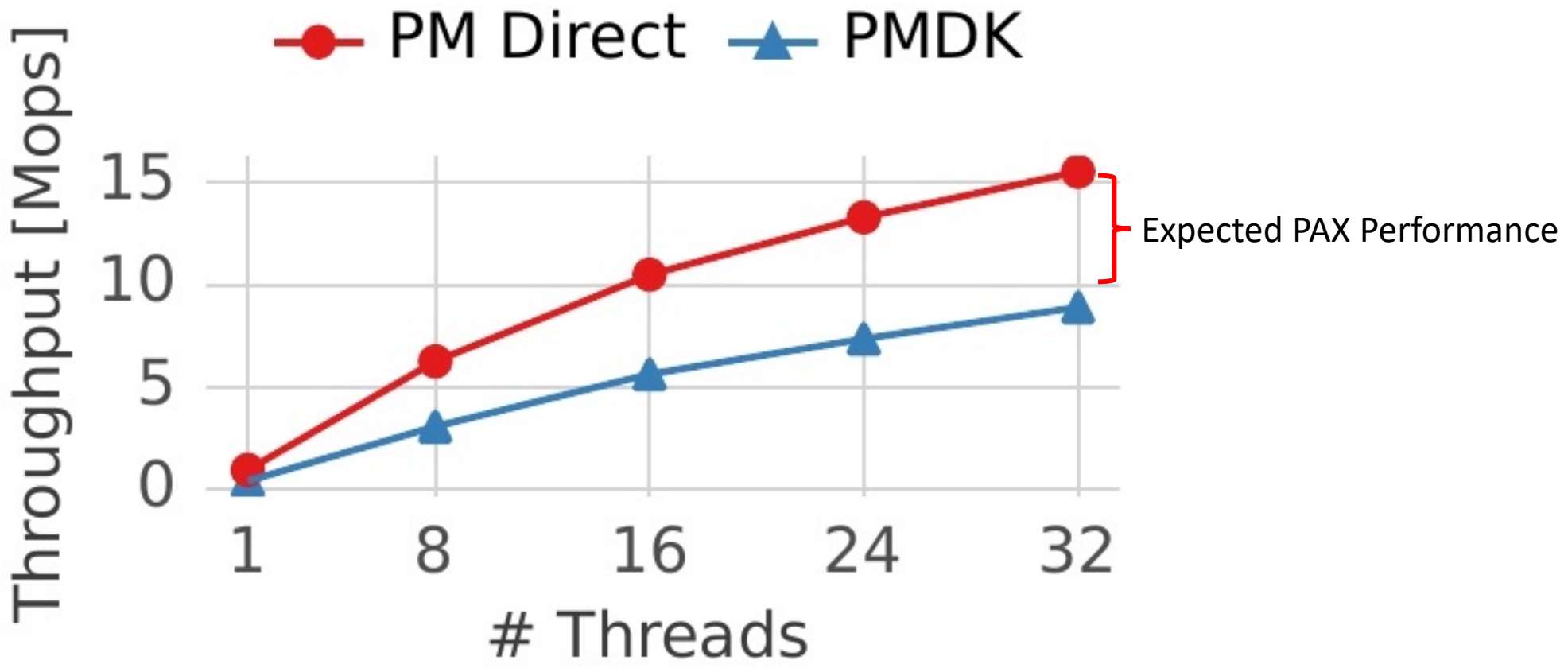
Expected performance with ccFPGA



Expected performance with ccFPGA



Expected performance with ccFPGA



PAX: Key takeaways

- Persistent Memory provides low-latency persistence
- However, persistent programming is complicated
- Most existing schemes do double synchronous PM writes and sometimes increase the write-amplification
- The goal of our work is to provide low-latency, black-box persistent for existing volatile structures using cc-Accelerators