Toward Failure Recoverable And Secured Persistent Memory Systems

IEEE Data & Storage Symposium 2022

Sihang Liu

University of Virginia
(Current)

University of Waterloo
(Joining in 2023 as an Assistant Professor)

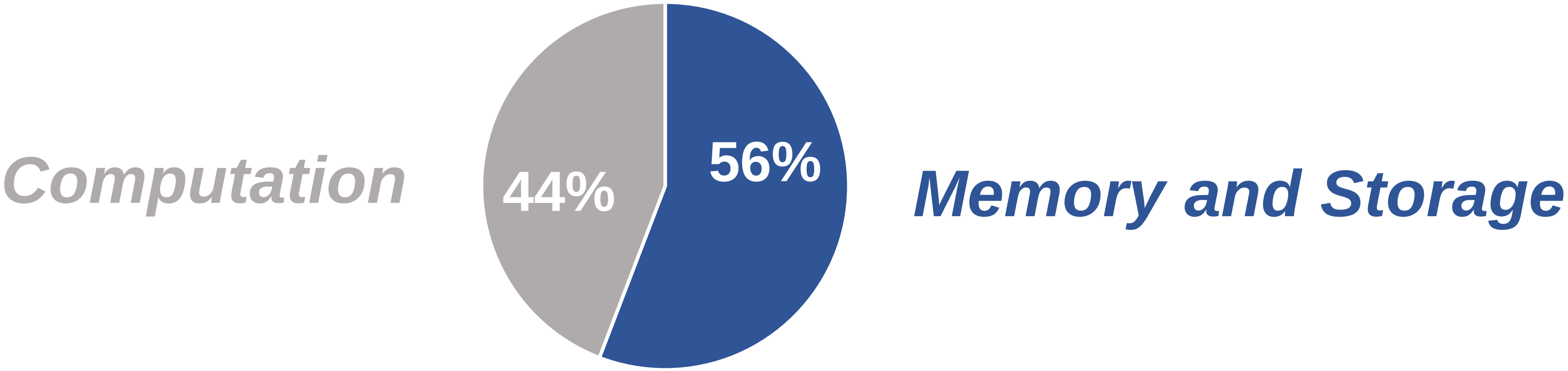
June 9, 2022

Demand for Memory and Storage

Data Analytics

Databases

In-memory Cache



Execution time breakdown in Google cloud workloads

Source: Kanev et al. ISCA’15.

Memory and storage takes the majority of time

Memory and Storage Technologies

Intel Optane Persistent Memory

Memory

DRAM

Storage

HDD/SSD

Volatile
Low-capacity
High-cost
Fast

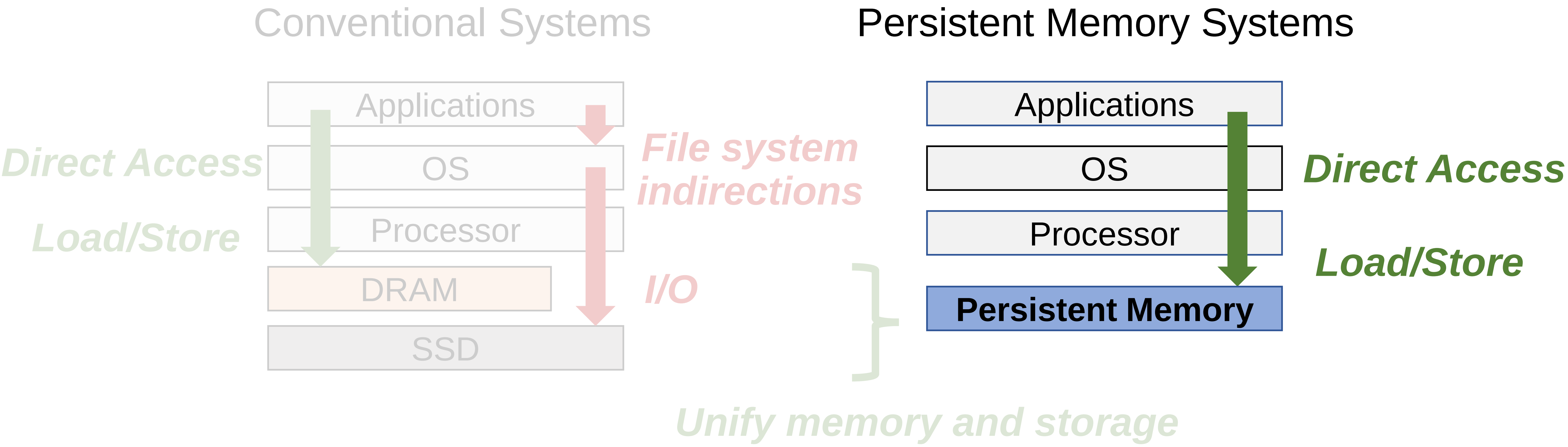
Persistent
High-capacity
Low-cost
Slow

Persistent Memory

Take advantages from both tiers

Persistent memory ***unifies*** memory and storage

System Stack for Persistent Memory

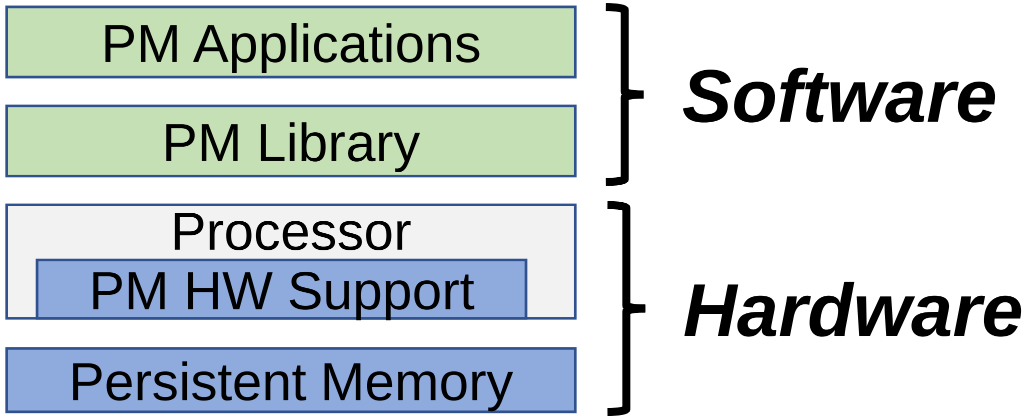


Enable better performance using direct management of persistent data

System Stack Redesign for Persistent Memory

To fully leverage the benefits

Persistent Memory (PM)



PM Applications

PM Library

Processor

PM HW Support

Persistent Memory

Software

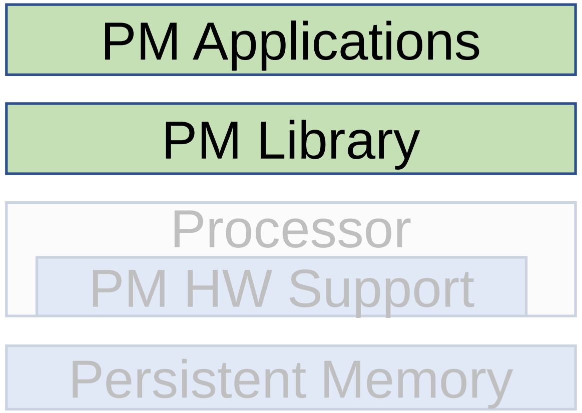
Hardware

System stack redesign

My research:
Seamlessly integrate persistent memory by redesigning both the software and hardware

Program Correctness

System Stack for Persistent Memory



PM Applications

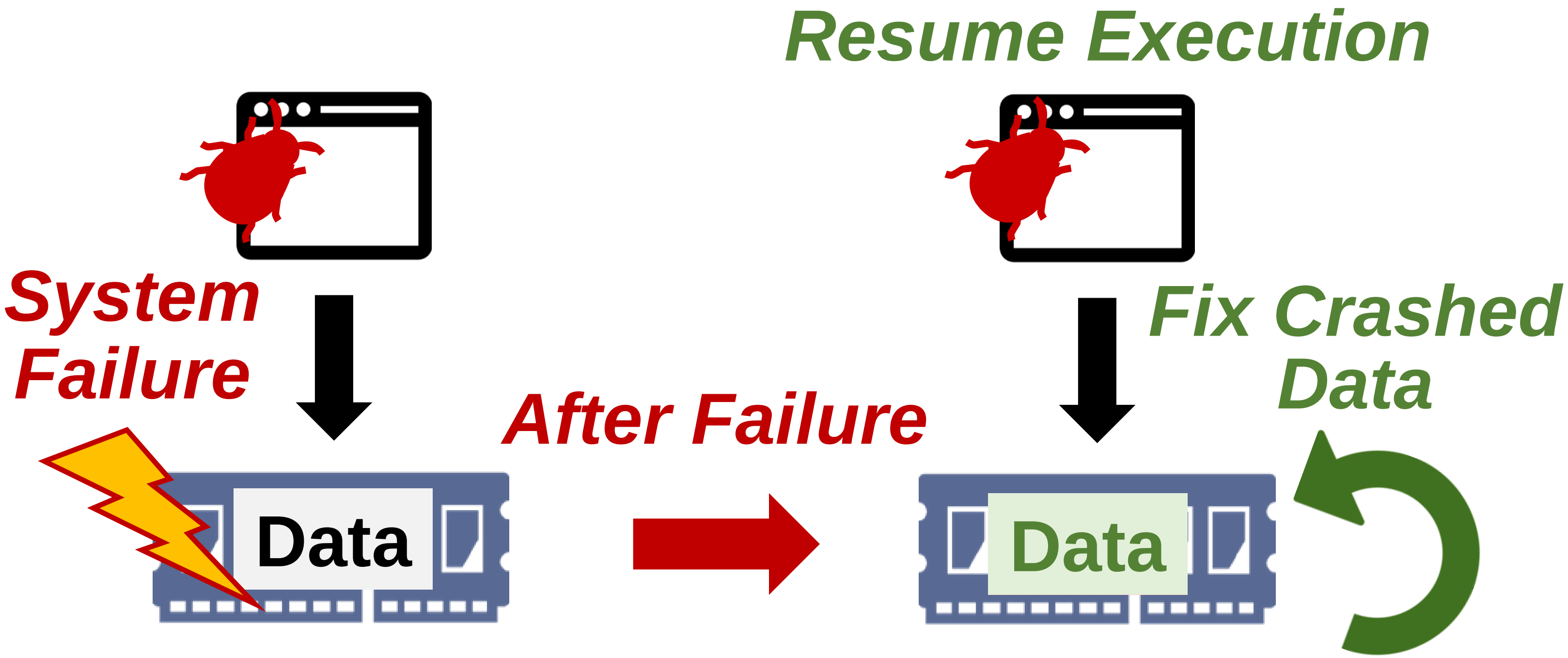
PM Library

Processor

PM HW Support

Persistent Memory

[ASPLOS'21, ASPLOS'20, ASPLOS'19]



Resume Execution

System Failure

After Failure

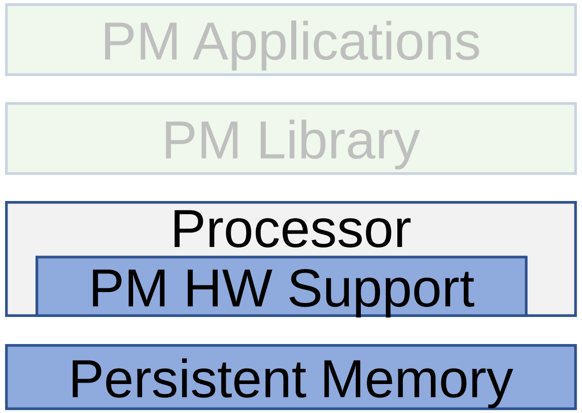
Fix Crashed Data

Persistent Memory System

Ensure ***correct failure-recovery*** for persistent memory programs

Efficiency and Security

System Stack for Persistent Memory



[*In-submission*, PACT'21, ISCA'19, HPCA'18]

Data is persistent

Data

Persistent Memory

Attackers may have physical access

At the same time …

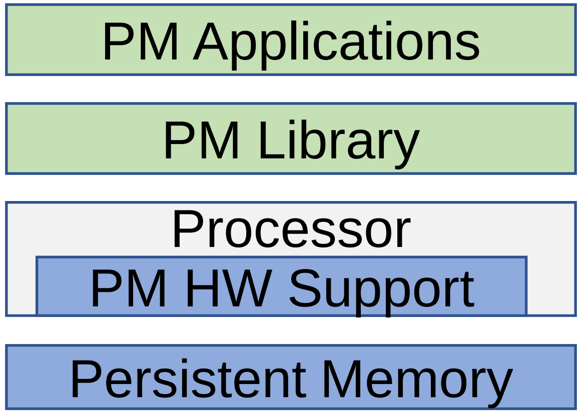
Secured

High-performance

Design ***efficient*** and ***secured*** persistent memory hardware

Outline

System Stack for Persistent Memory



Software Support for Persistent Memory

Testing frameworks for ***failure-recovery issues***

PMTest [ASPLOS'19]

XFDetector [ASPLOS'20]

A test case generator for better ***testing efficiency***

PMFuzz [ASPLOS'21]

Hardware System for Persistent Memory

Efficient and ***secured*** persistent memory hardware

Software-hardware co-designs [HPCA'18, ISCA'19, PACT'21]

New security vulnerabilities in Intel's persistent memory

[In-submission]

Outline

System Stack for Persistent Memory

PM Applications

PM Library

Processor

PM HW Support

Persistent Memory

Software Support for Persistent Memory

Testing frameworks for ***failure-recovery issues***

PMTest [ASPLOS'19]

XFDetector [ASPLOS'20]

A test case generator for better ***testing efficiency***

PMFuzz [ASPLOS'21]

Hardware System for Persistent Memory

Efficient and ***secured*** persistent memory hardware

Software-hardware co-designs

[HPCA'18, ISCA'19, PACT'21]

New security vulnerabilities in Intel's persistent memory

[In-submission]

PMTest:
A Fast and Flexible Testing Framework for Persistent Memory Programs

Sihang Liu, Yizhou Wei, Jishen Zhao, Aasheesh Kolli, and Samira Khan

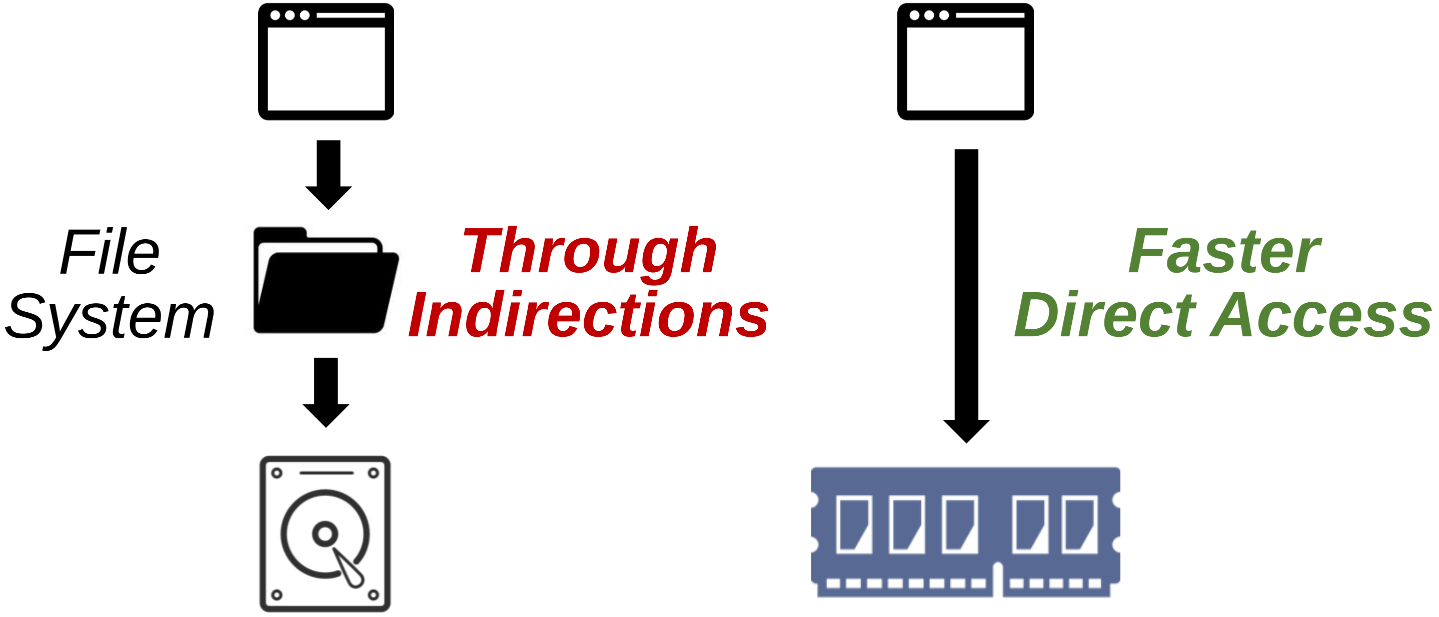
The 2019 International Conference on Architectural Support for Programming Languages and Operating Systems (**ASPLOS**)

NVMW Memorable Paper Award—Finalist

What if the system fails?

Conventional System

Persistent Memory System



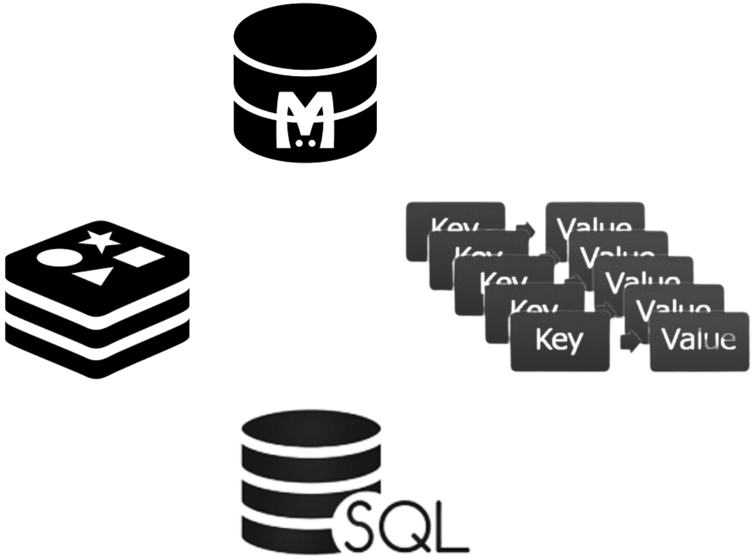
File System

Through Indirections

Faster Direct Access

Slower

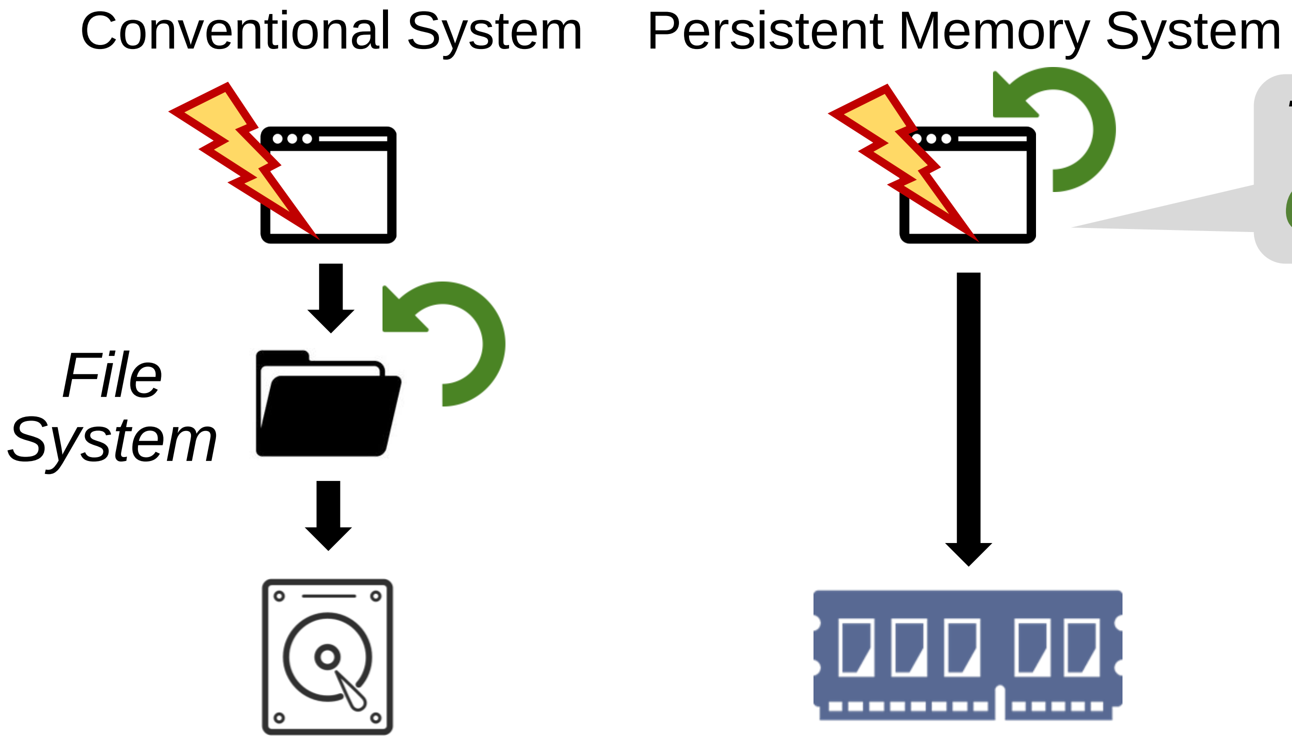
Faster



Storage Applications

Faster, direct access benefits storage applications

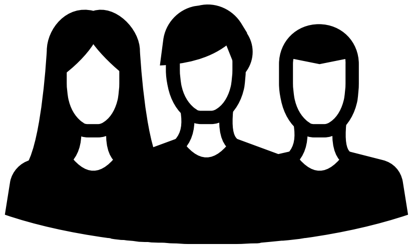
Software for Persistent Memory Systems



The ability to recover:
Crash consistency guarantees

File system handles recovery

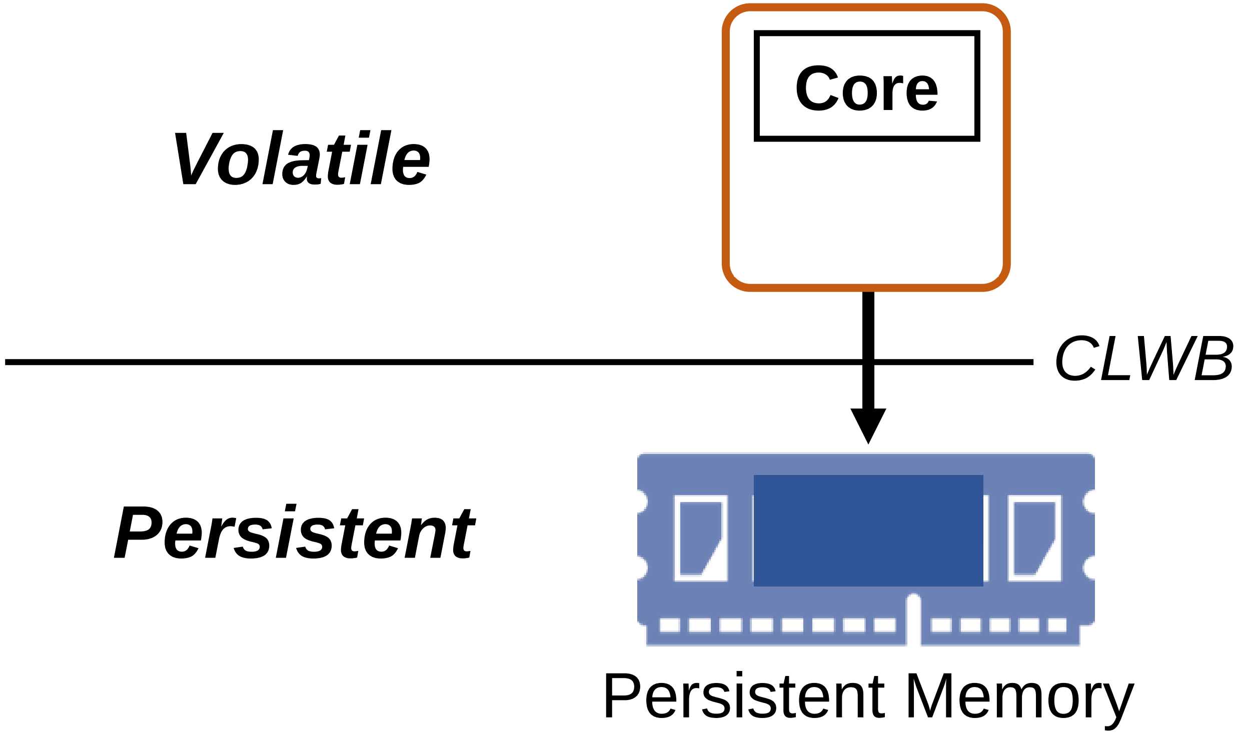
Program customizes recovery



The burden of ***failure-recovery*** lies on the programmers

Programming for Persistent Memory Systems

- Support for crash consistency has ***two fundamental guarantees***
  - ***Persistence***: writes become persistent ***on demand***

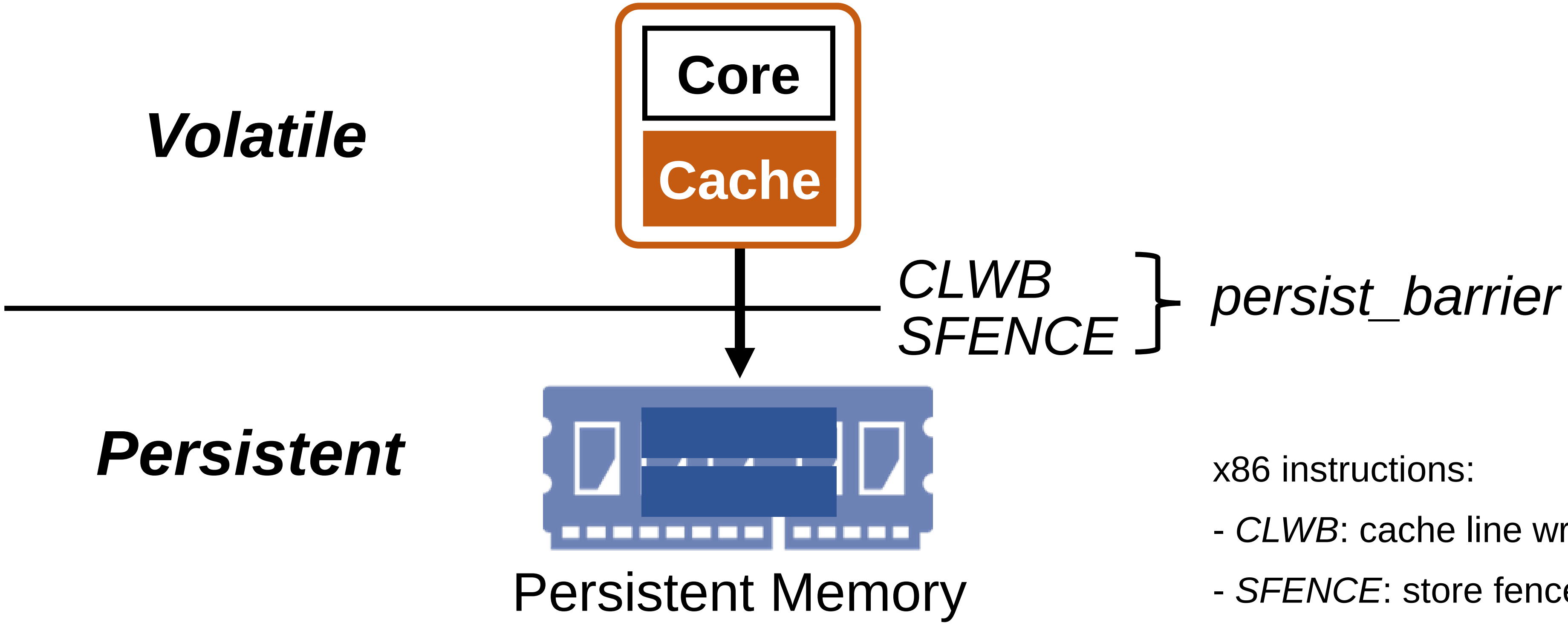


x86 instructions:

- *CLWB*: cache line write back

Programming for Persistent Memory Systems

- Support for crash consistency has ***two fundamental guarantees***
  - ***Persistence***: writes become persistent ***on demand***
  - ***Ordering***: one write becomes persistent ***before*** another



x86 instructions:

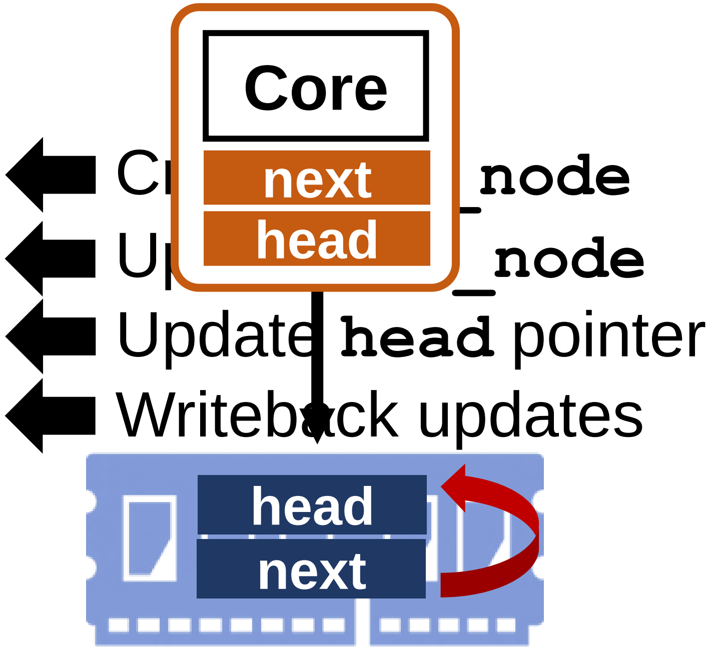
- *CLWB*: cache line write back

- *SFENCE*: store fence for ordering

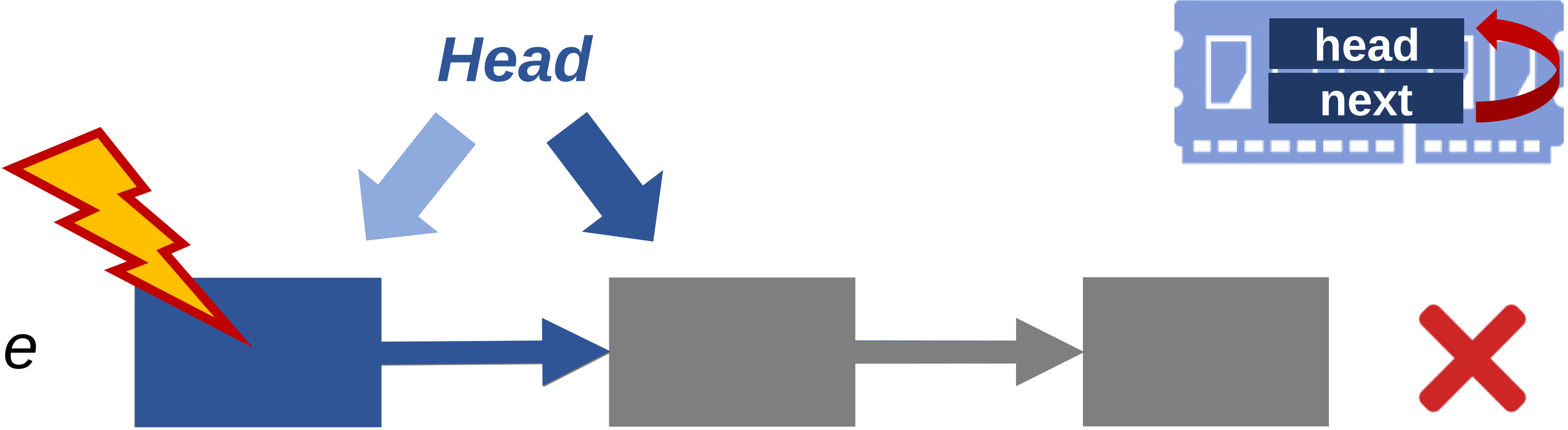
Example of Persistent Memory Programming

```
void listAppend(item_t new_val) {
   node_t* new_node = new node_t(new_val);
   new_node->next = head;
   head = new_node;
   persist_barrier();
}
```

Writes can reorder



In cache



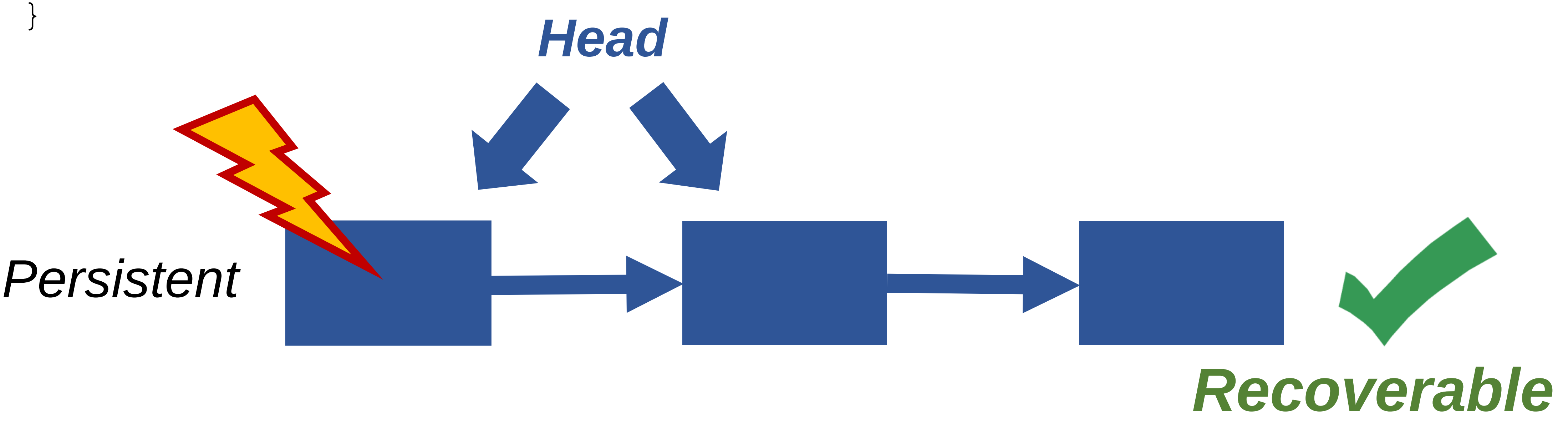
new_node is lost after failure

Unrecoverable

Ensuring crash consistency is hard!

```
void listAppend(item_t new_val) {
  node_t* new_node = new node_t(new_val);
  new_node->next = head;
  persist_barrier();
  head = new_node;
  persist_barrier();
}
```

Enforce writeback before changing **head**



Programming for Persistent Memory Is Hard!

System Experts

Directly using ***low-level primitives*** to implement crash-consistent programs is ***not trivial***

E.g., Software developed by Lenovo has misuse of low-level primitives, e.g., `persist_barrier()`

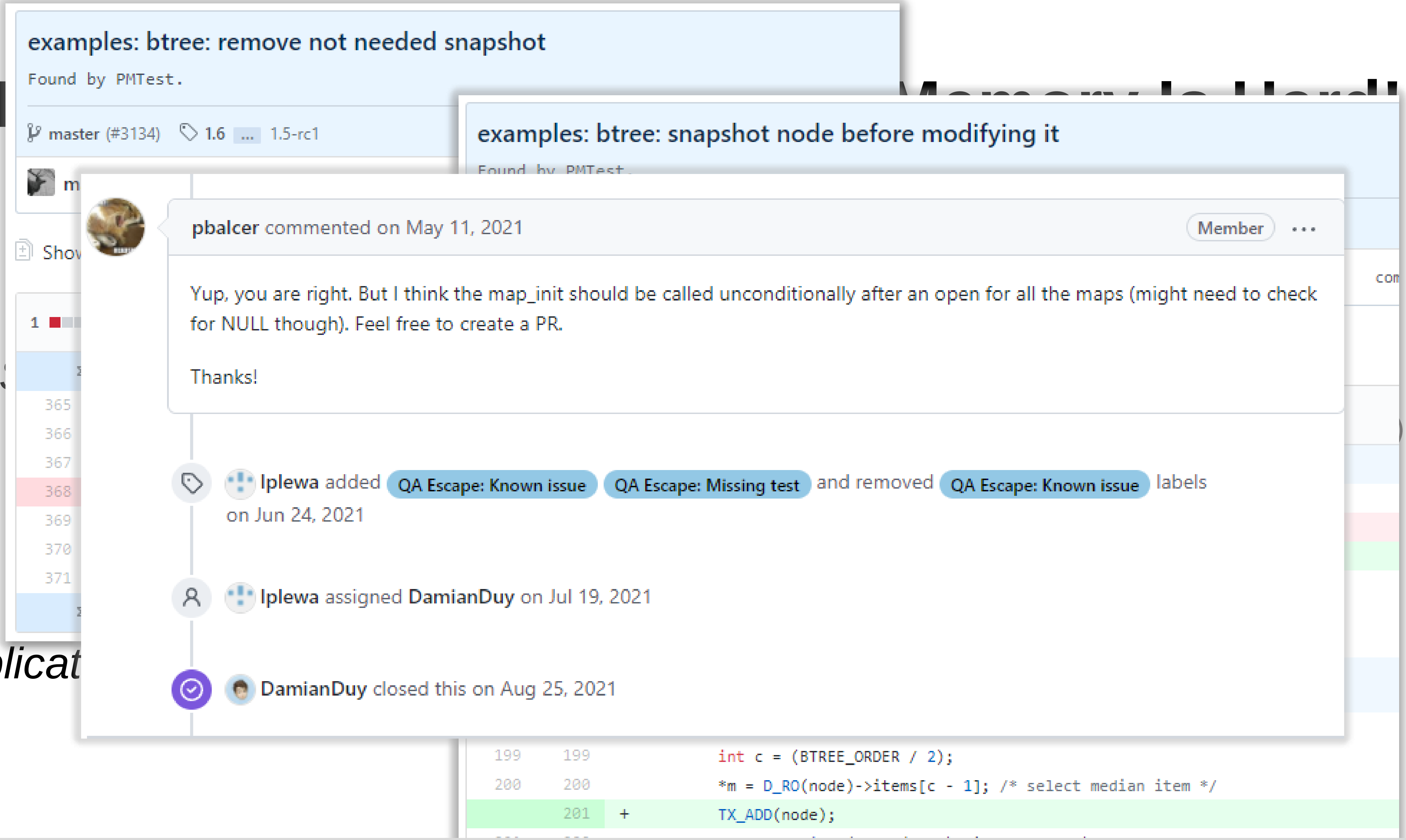
Application Developers

Simplify

Libraries are developed for persistent memory to ***make programming easier***

(e.g., Intel's PMDK library)

Memory Is Hard!



examples: btree: remove not needed snapshot

Found by PMTest.

master (#3134) 1.6 ... 1.5-rc1

examples: btree: snapshot node before modifying it

Found by PMTest.

pbalcer commented on May 11, 2021 Member

Yup, you are right. But I think the map_init should be called unconditionally after an open for all the maps (might need to check for NULL though). Feel free to create a PR.

Thanks!

lplewa added QA Escape: Known issue QA Escape: Missing test and removed QA Escape: Known issue labels on Jun 24, 2021

lplewa assigned DamianDuy on Jul 19, 2021

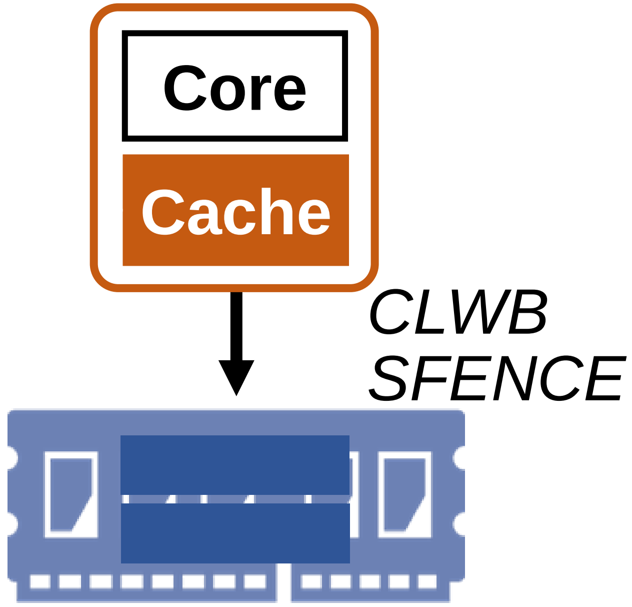
DamianDuy closed this on Aug 25, 2021

```
199  199          int c = (BTREE_ORDER / 2);
200  200          *m = D_RO(node)->items[c - 1]; /* select median item */
     201   +      TX_ADD(node);
```

Applicat

We need to test persistent memory programs

Challenge I: Expose Crash Consistency Issues

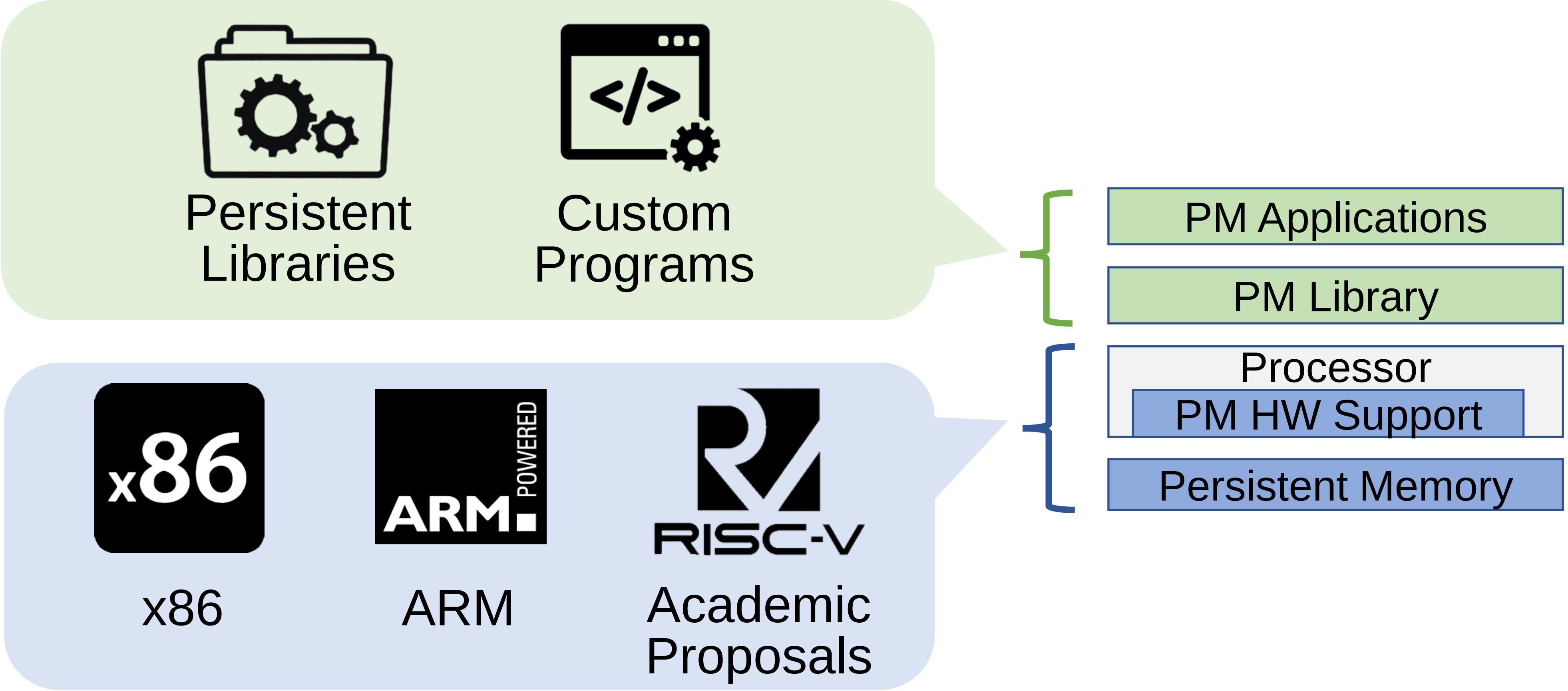


Not exposed to the software

Crash consistency bugs

How can we expose crash consistency bugs?

Challenge II: Various Persistent Memory Systems



Persistent Libraries

Custom Programs

x86

ARM

Academic Proposals

PM Applications

PM Library

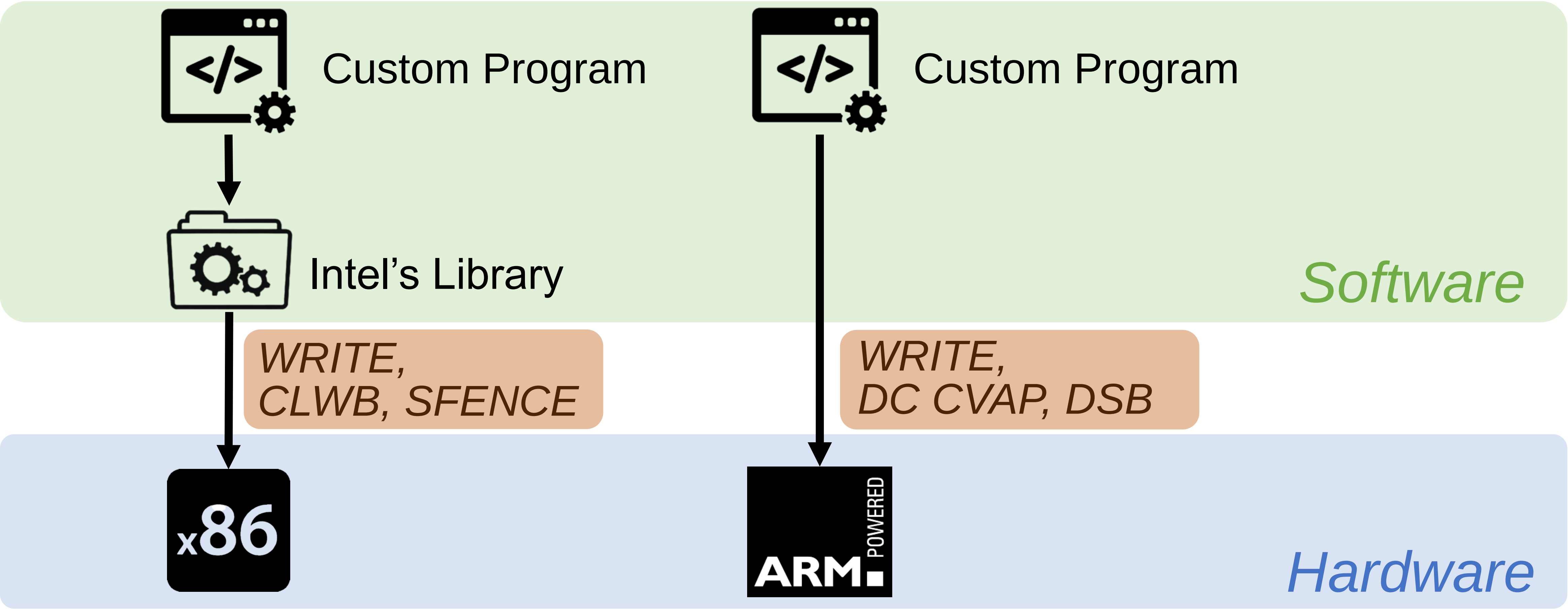
Processor

PM HW Support

Persistent Memory

How can we cover ***various software*** and ***hardware***?

Operations for crash consistency are similar: guarantees of ***ordering*** and ***persistence***



Custom Program

Intel’s Library

WRITE, CLWB, SFENCE

Custom Program

WRITE, DC CVAP, DSB

Software

Hardware

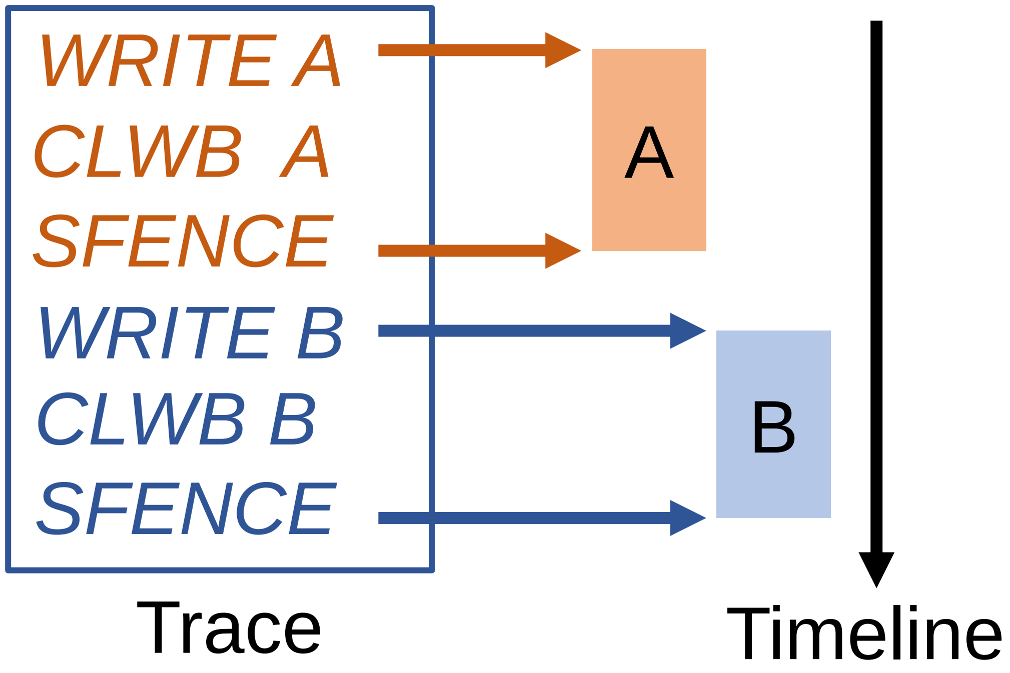
Expose persistence and ordering

Expose Persistence and Ordering

Construct ***persistence intervals*** from instruction trace

➡ *A time interval in which a write may become persistent*

Deduce ***persistence*** and ***ordering***



➡ **A becomes persistent before B**

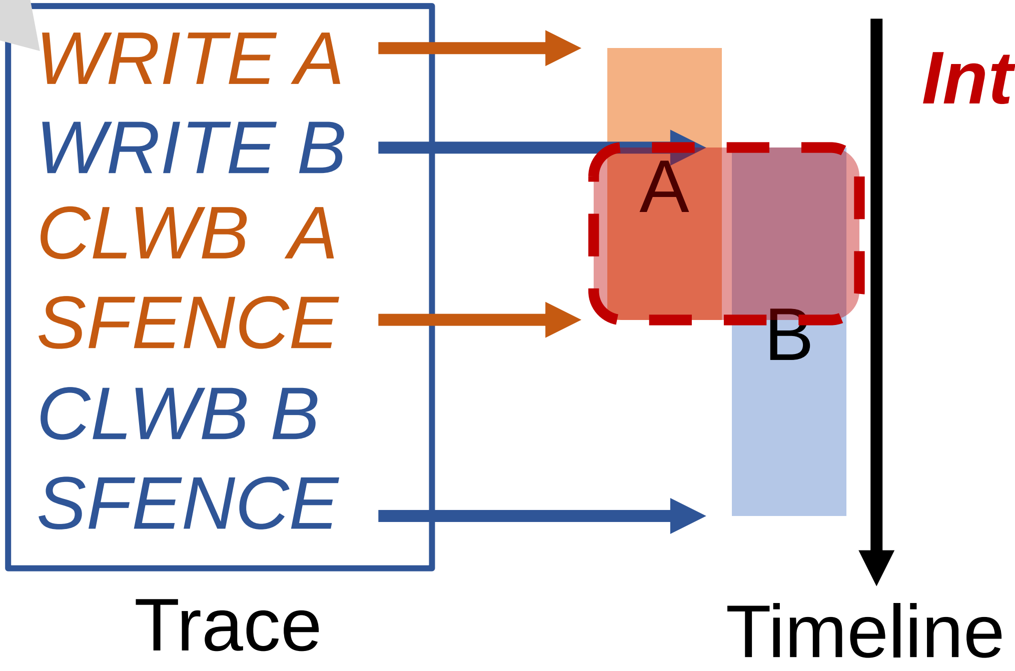
Expose Persistence and Ordering

Construct ***persistence intervals*** from instruction trace

A time interval in which a write may become persistent

rdering

Specification:
A becomes persistent before B



WRITE A
WRITE B
CLWB A
SFENCE
CLWB B
SFENCE

Interleaving

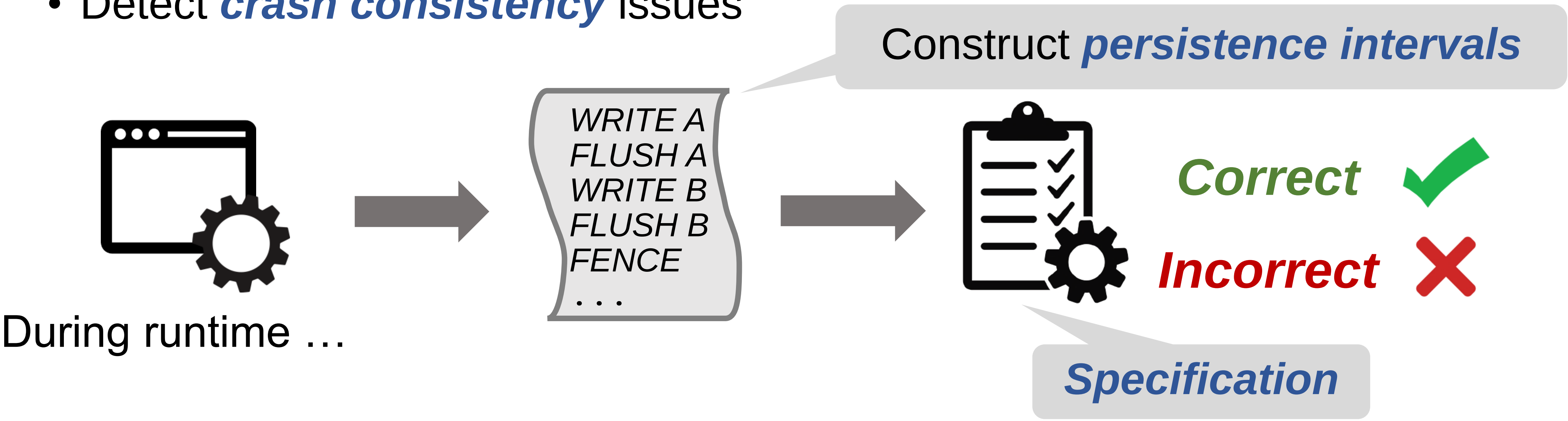
A may NOT become persistent before B

Result: Incorrect

Our Work: PMTest

Workflow:

- ***Tracks accesses*** to persistent memory
- Deduce the ***ordering*** and ***persistence***
- Check against ***specifications***
- Detect ***crash consistency*** issues



During runtime …

WRITE A
FLUSH A
WRITE B
FLUSH B
FENCE
. . .

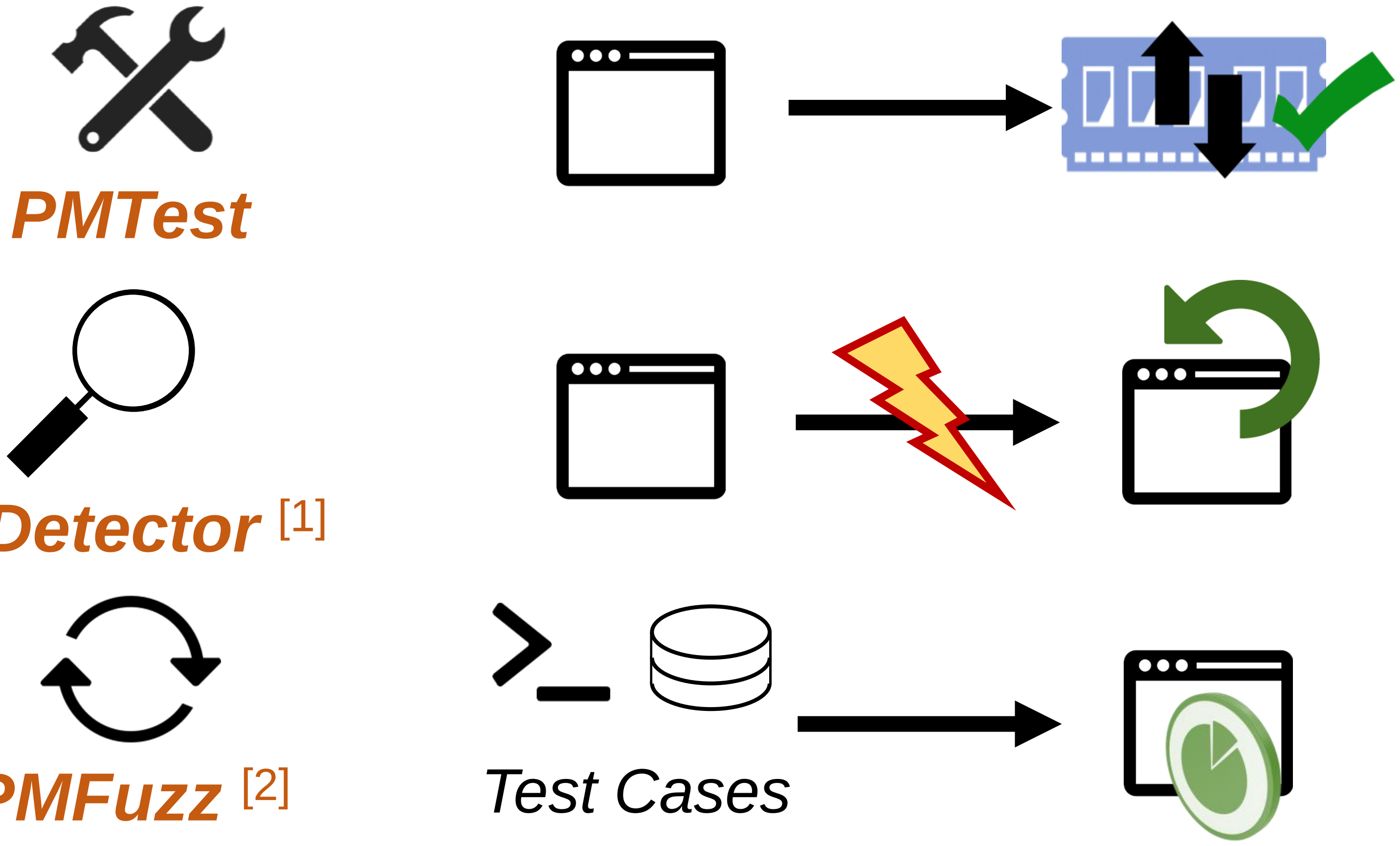
Construct ***persistence intervals***

Correct ✓

Incorrect ✗

Specification

These tools have detected ***18 bugs*** in existing software ***produced by the industry*** for persistent memory systems



PMTest

Ensure correct ***ordering*** and ***persistence***

XFDetector [1]

End-to-end correctness, including ***recovery***

PMFuzz [2]

Test Cases

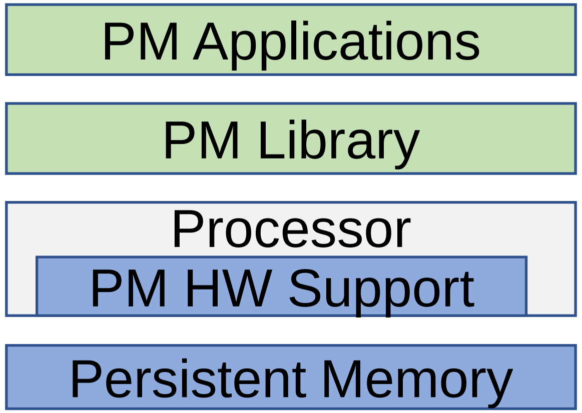
Generate high-coverage ***test cases***

[1] **Cross-Failure Bug Detection in Persistent Memory Programs.**
Sihang Liu, Korakit Seemakhupt, Yizhou Wei, Thomas Wenisch, Aasheesh Kolli, and Samira Khan. **ASPLOS.** 2020.

[2] **PMFuzz: Test Case Generation for Persistent Memory Programs.**
Sihang Liu*, Suyash Mahar*, Baishakhi Ray, and Samira Khan. **ASPLOS.** 2021.

Outline

System Stack for Persistent Memory


PM Applications

PM Library

Processor

PM HW Support

Persistent Memory

Software Support for Persistent Memory

Testing frameworks for ***failure-recovery issues***

PMTest [ASPLOS'19]

XFDetector [ASPLOS'20]

A test case generator for better ***testing efficiency***

PMFuzz [ASPLOS'21]

Hardware System for Persistent Memory

Efficient and ***secured*** persistent memory hardware

Software-hardware co-designs

[HPCA'18, ISCA'19, PACT'21]

New security vulnerabilities in Intel's persistent memory

[In-submission]

Janus:

Optimizing Memory and Storage Support for Non-Volatile Memory Systems

Sihang Liu, Korakit Seemakhupt, Gennady Pekhimenko,
Aasheesh Kolli, and Samira Khan

The 2019 International Symposium on Computer Architecture (**ISCA**)

MICRO Top Picks—Honorable Mention

Persistent Memory Hardware

Persistent memory hardware comes different types of supports

Security

Prevent attackers from stealing or tampering data
Encryption, integrity verification, etc.

Bandwidth

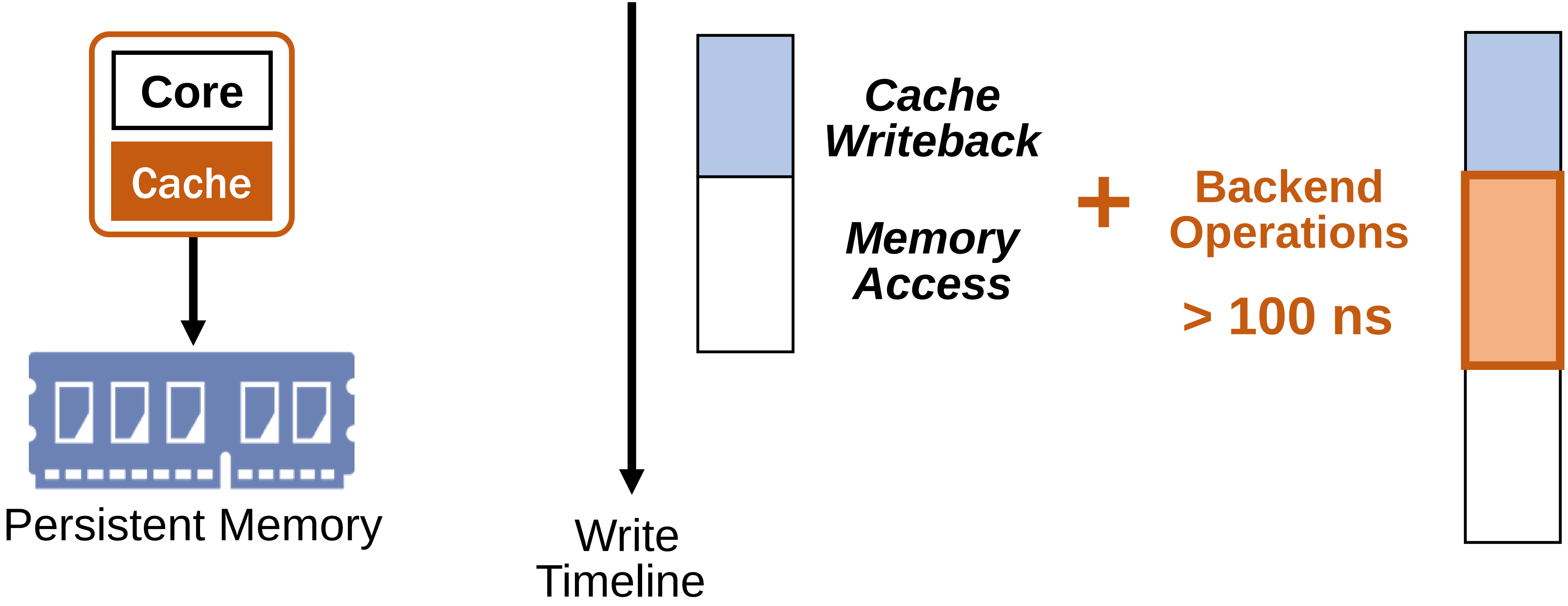
Improve bandwidth
Deduplication, compression, etc.

Lifetime

Extend lifetime
Wear-leveling, error correction, etc.

Backend operations in persistent memory hardware

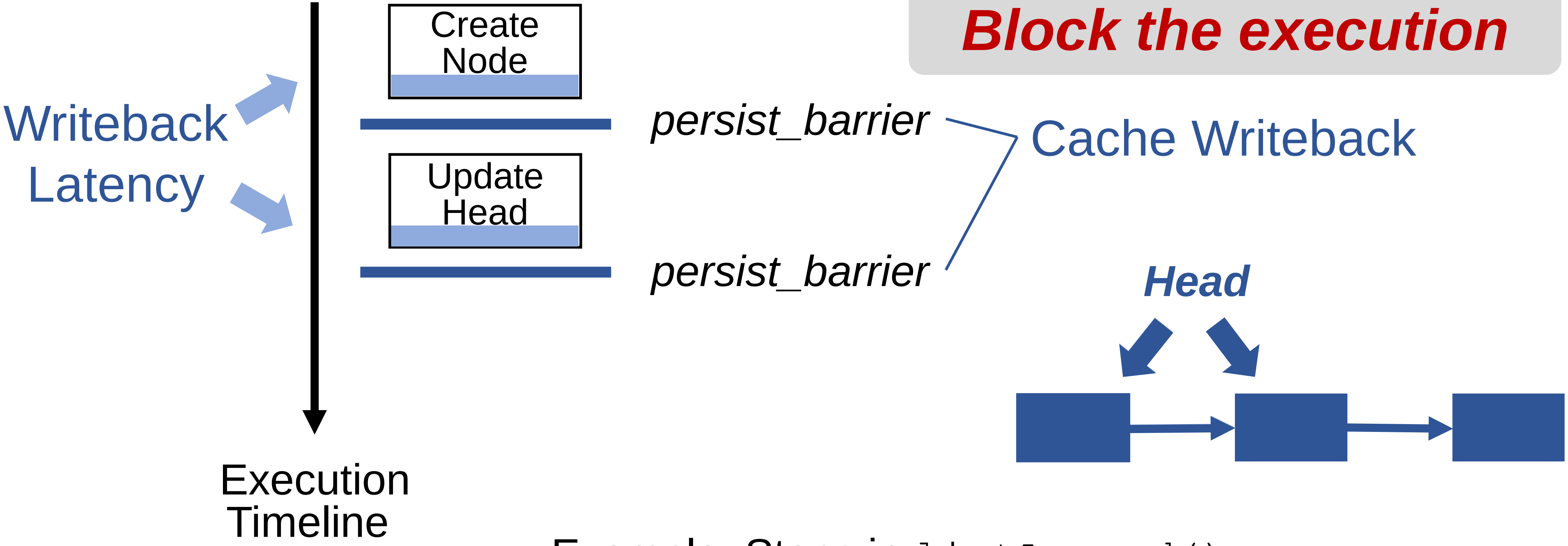
Increased Write Latency



Backend operations ***increase write latency***

Why is write latency critical?

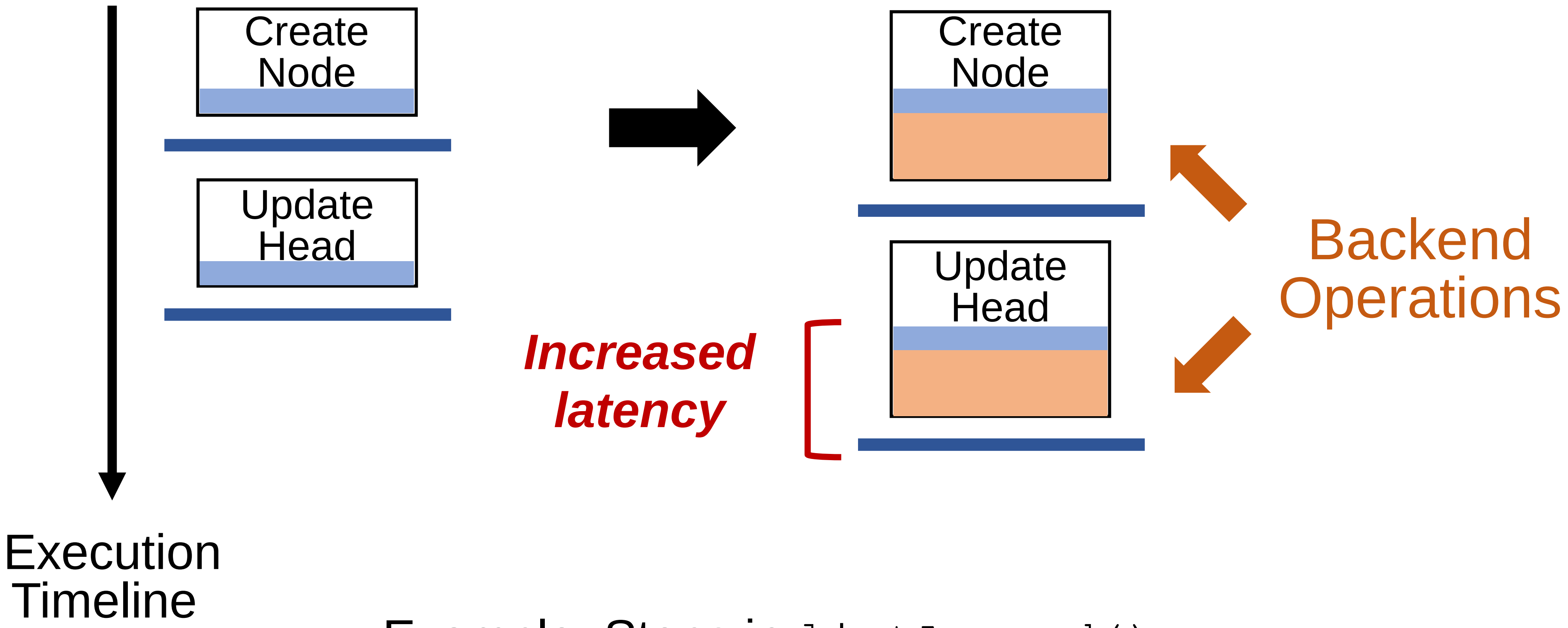
Block the execution



Example: Steps in `listAppend()`

Crash consistency mechanisms put
write latency on the critical path

Overhead of Backend Operations



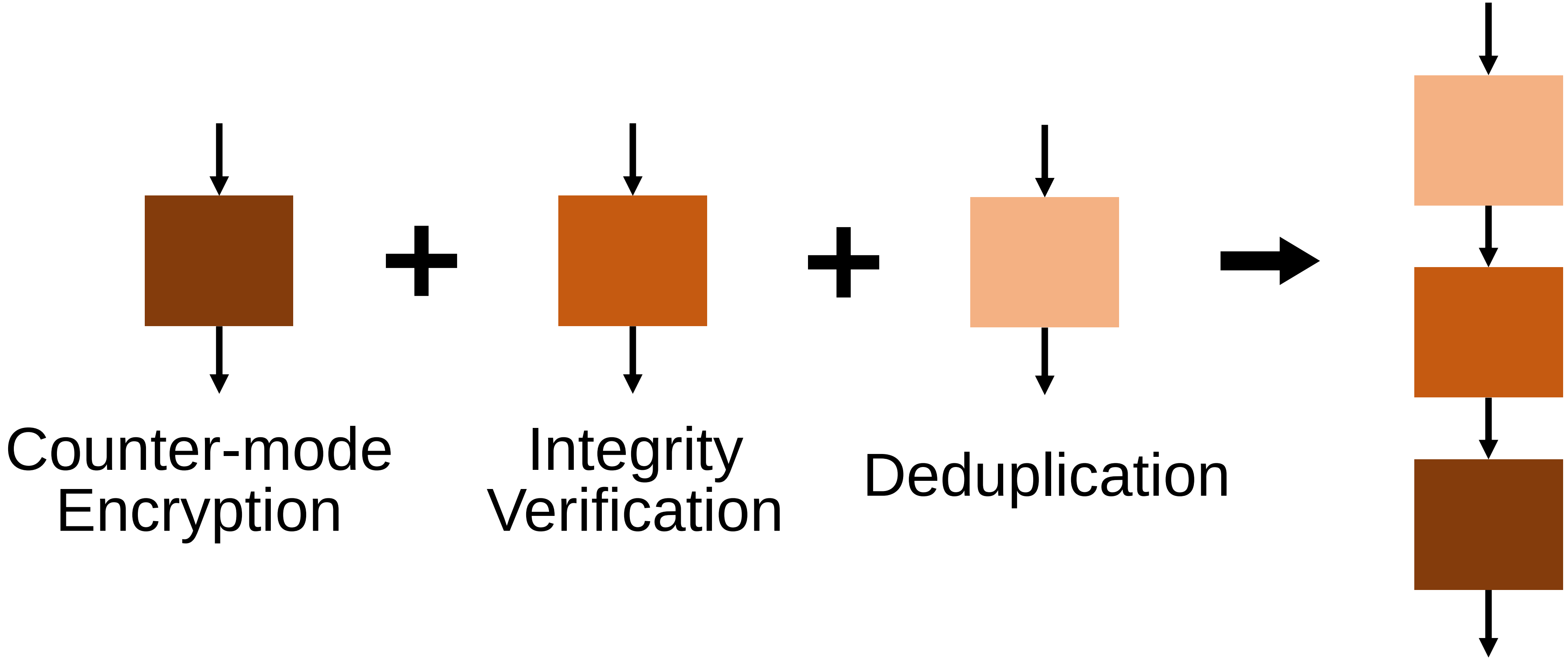
Example: Steps in `listAppend()`

Backend operations ***increase*** the execution time

Challenges in Optimizing Backend Operations

Each backend operation seems ***indivisible***

➡ Integration leads to ***serialized*** operations



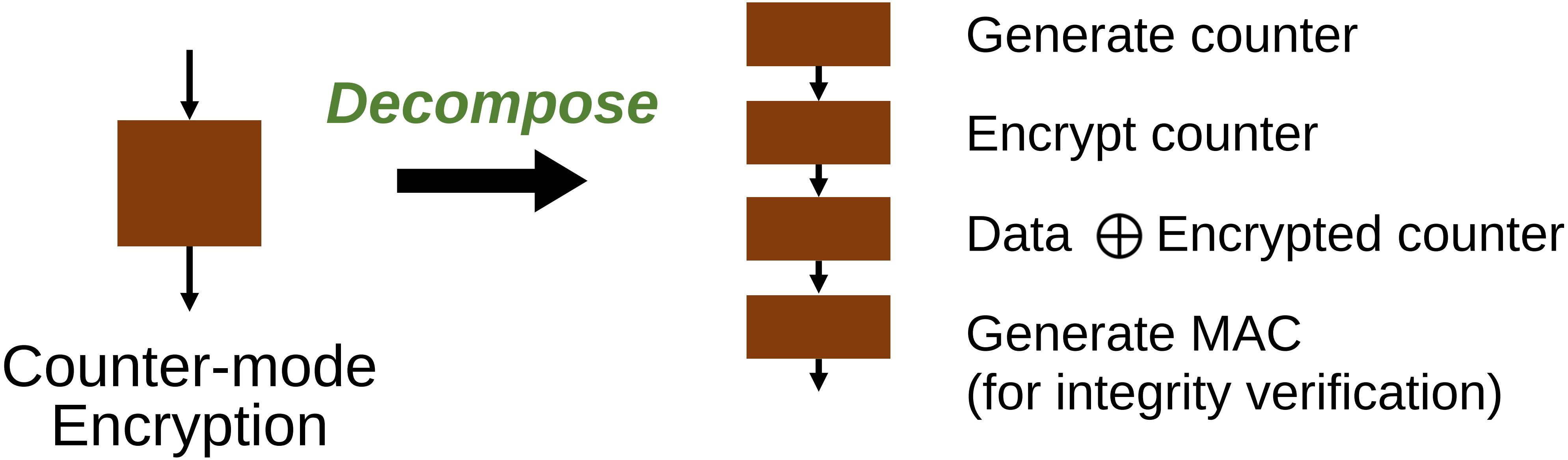
Counter-mode Encryption

Integrity Verification

Deduplication

Decomposition of Backend Operations

However, it is possible to decompose them into ***sub-operations***



Decompose

Counter-mode Encryption

Generate counter

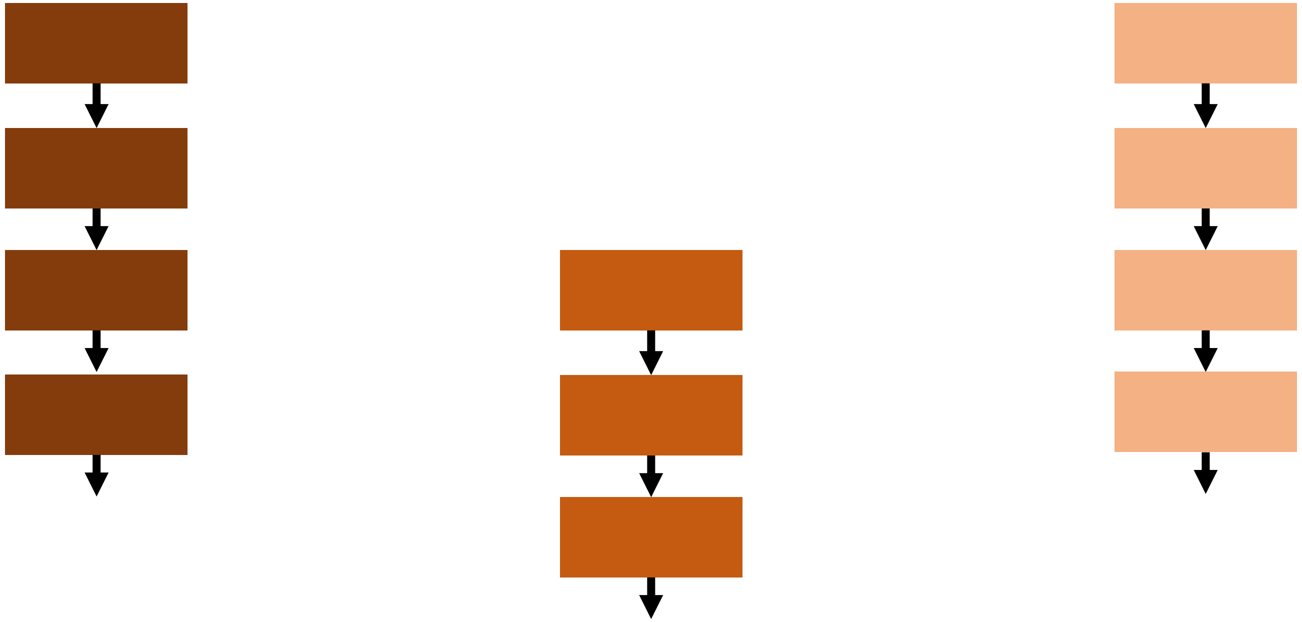
Encrypt counter

Data $\oplus$ Encrypted counter

Generate MAC
(for integrity verification)

Decomposition of Backend Operations

Decomposing the example operations:



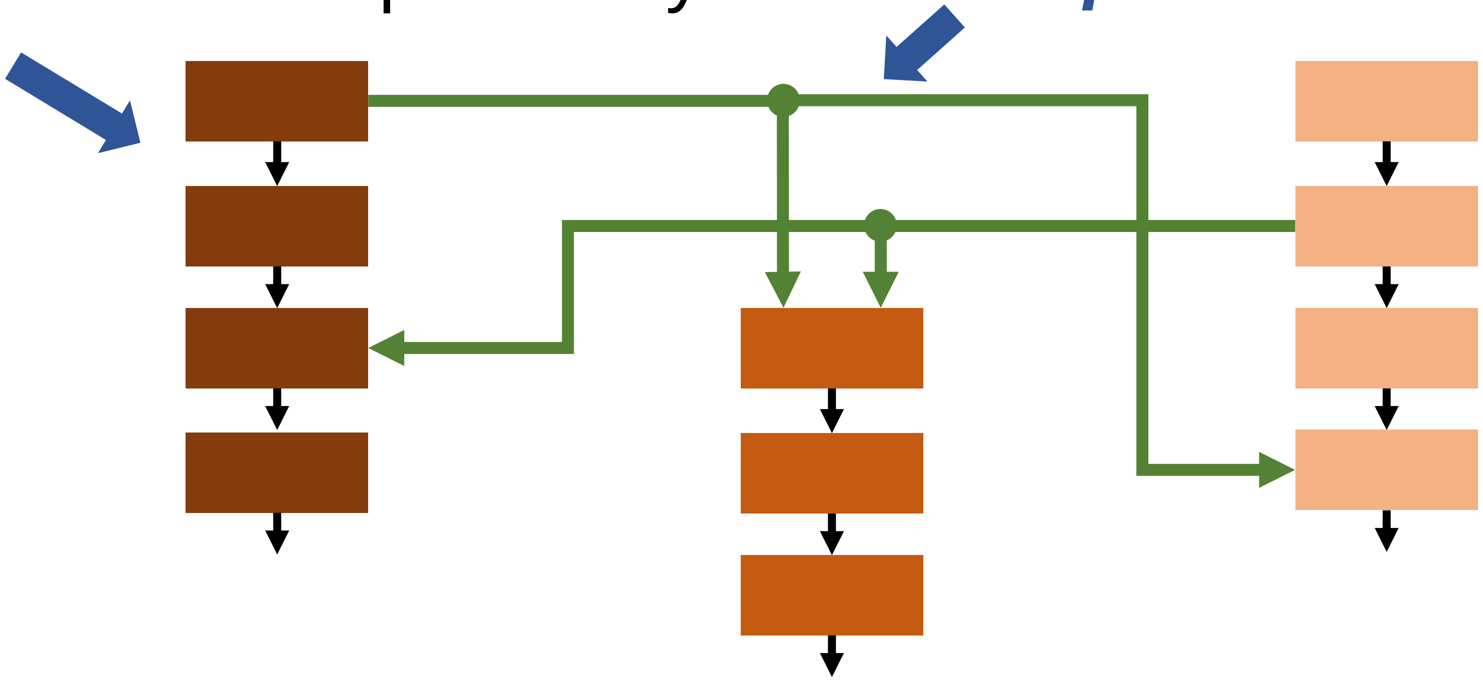
Decomposing backend operations enables more optimizations

Optimization: Parallelization

There are two types of dependencies:

Intra-operation dependency

Inter-operation dependency

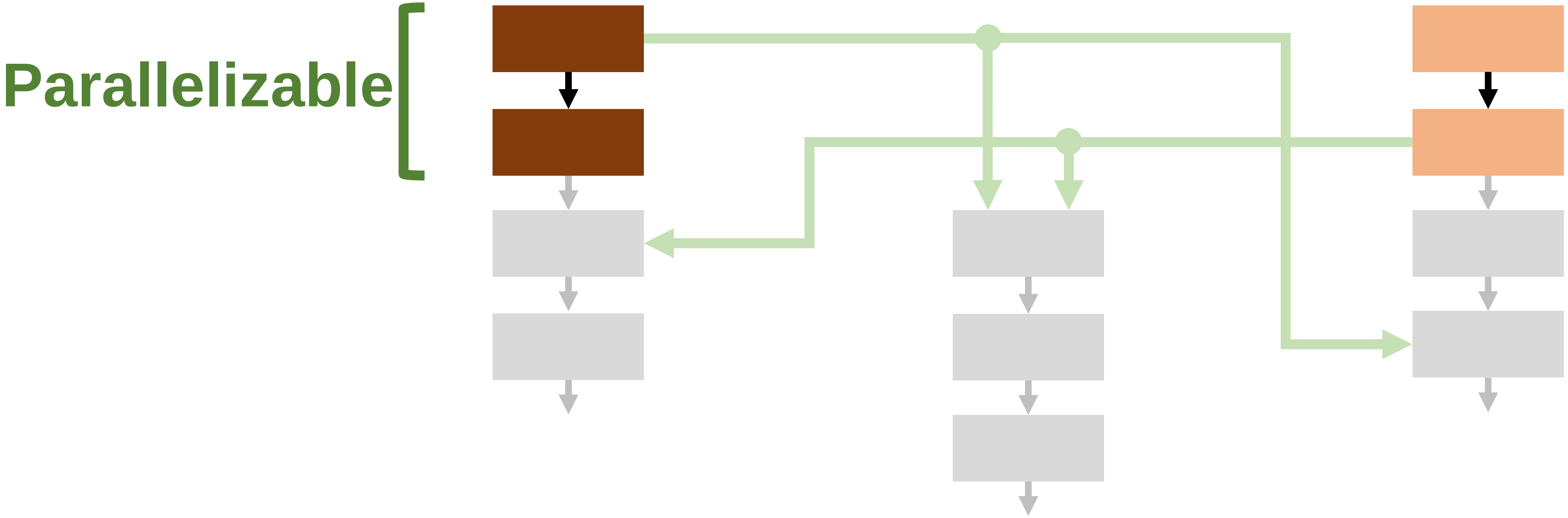


1. Dependency ***within each operation***

Optimization: Parallelization

There are two types of dependencies:

Intra-operation dependency ***Inter-operation*** dependency



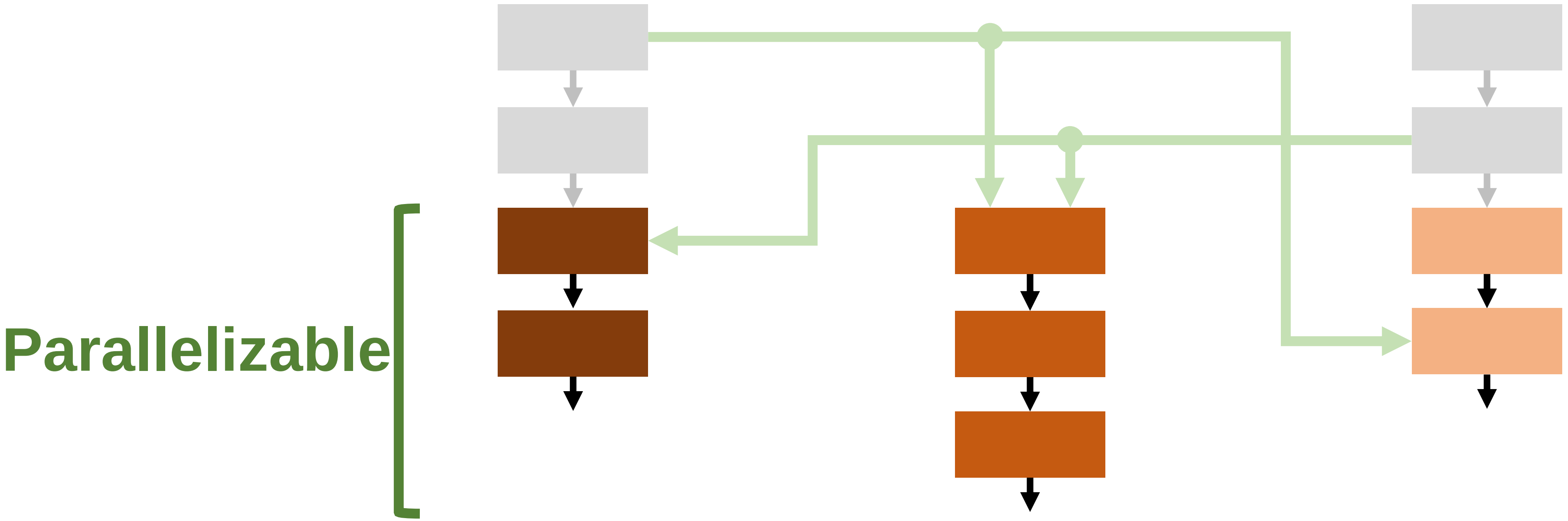
Sub-operations without dependency can execute ***in parallel***

Optimization: Parallelization

There are two types of dependencies:

Intra-operation dependency

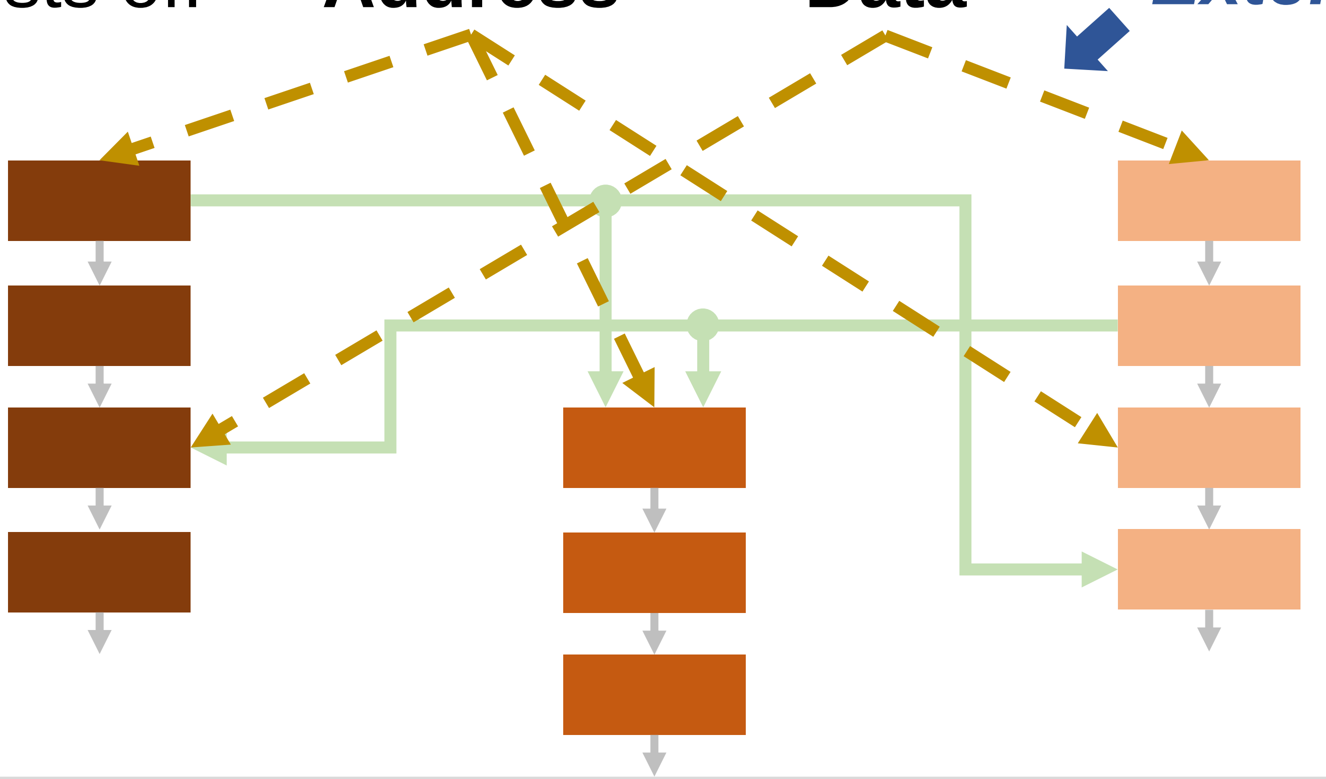
Inter-operation dependency



Sub-operations without dependency can execute ***in parallel***

Optimization: Pre-execution

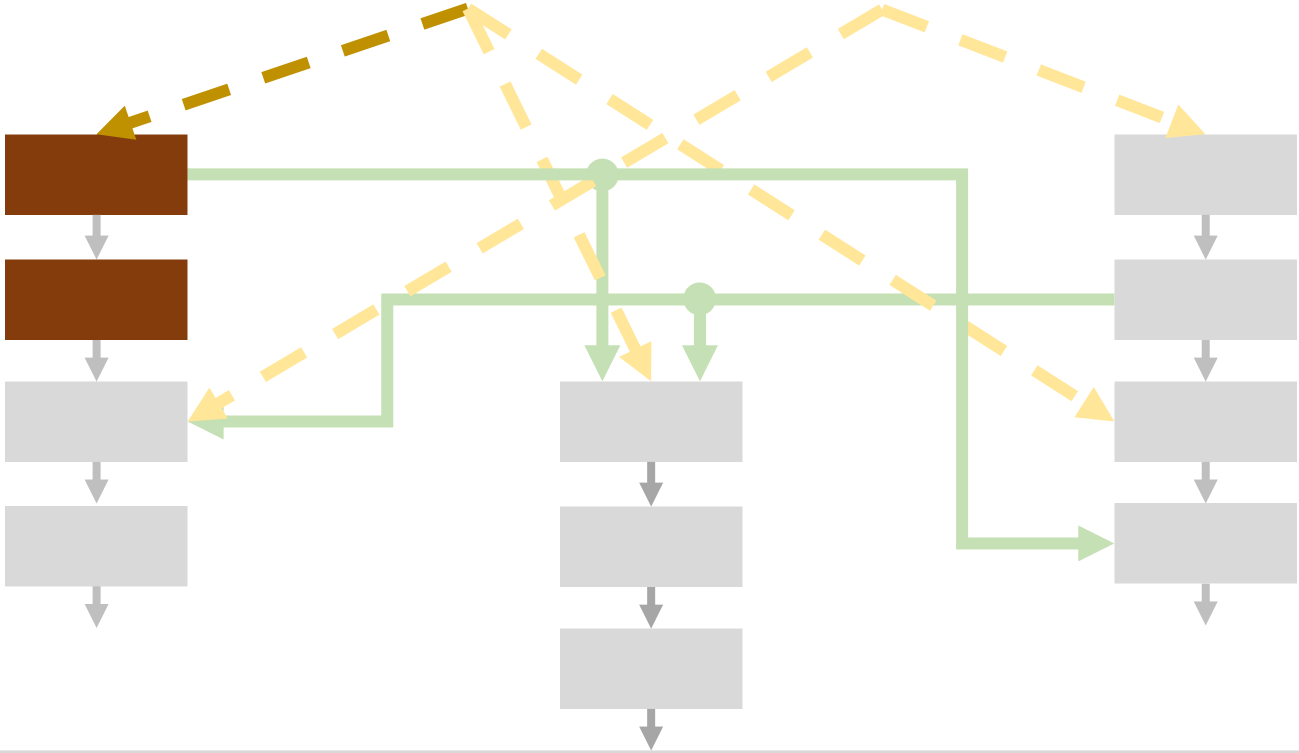
A write consists of: **Address** **Data** ***External*** dependency



Sub-operations can ***pre-execute***
as soon as their data/address dependency is resolved

Optimization: Pre-execution

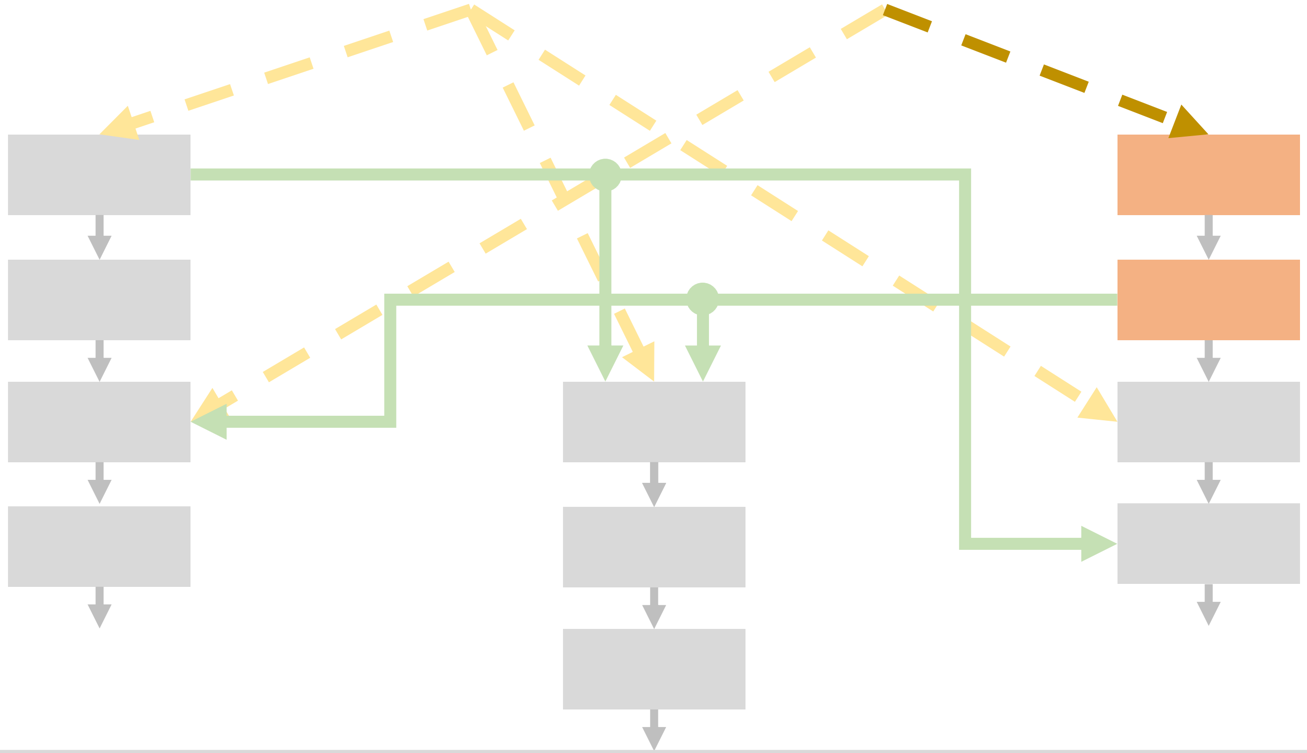
A write consists of: **Address** **Data** ***Address-dependent***



Address-dependent sub-operations can pre-execute as soon as the address of the write is available

Optimization: Pre-execution

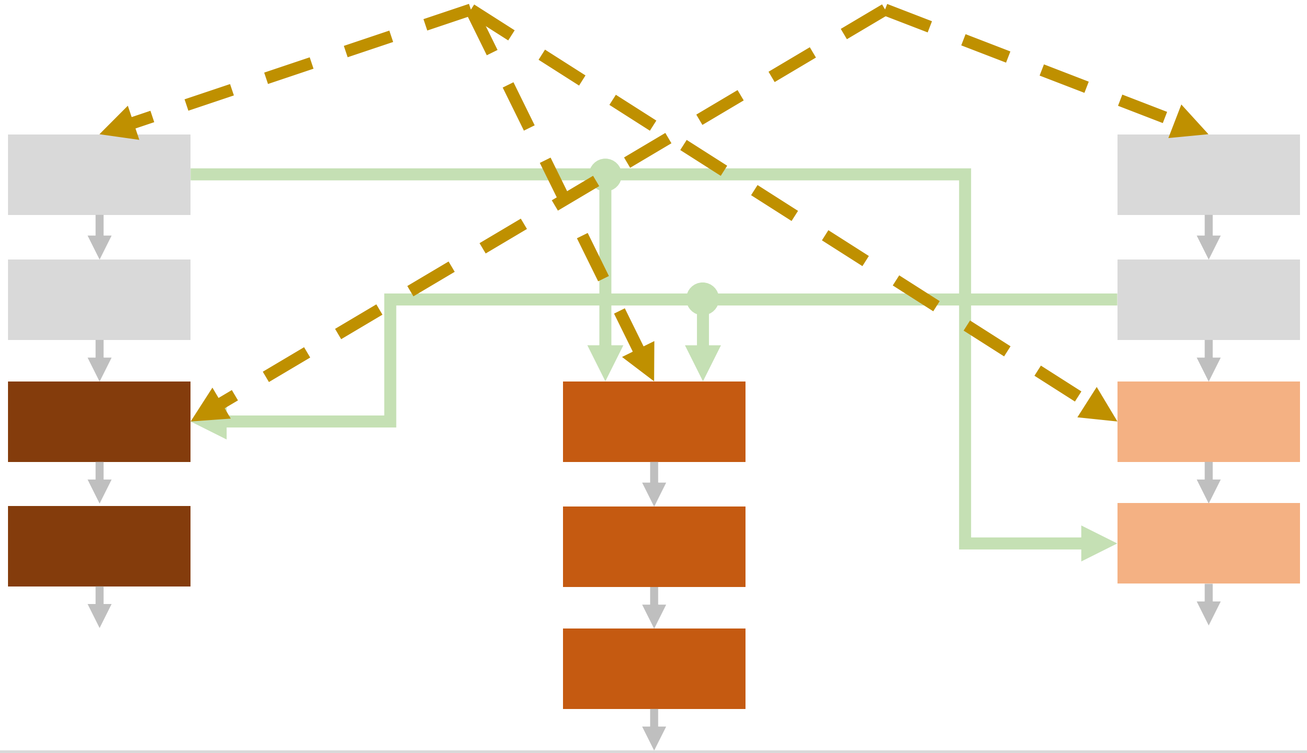
A write consists of: **Address** **Data** ***Data-dependent***



Data-dependent sub-operations can ***pre-execute***
as soon as the data of the write is available

Optimization: Pre-execution

A write consists of: **Address** **Data** ***Both-dependent***



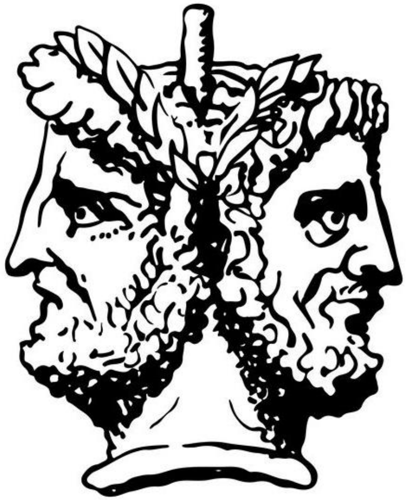
Both-dependent sub-operations can ***pre-execute*** as soon as the data and address of the write are available

Our Work: Janus

Janus is a Roman god with ***two faces***:

One looks into the ***past*** and another into the ***future***

When dependent data and address become available



Pre-execute operations with dependency resolved

Past

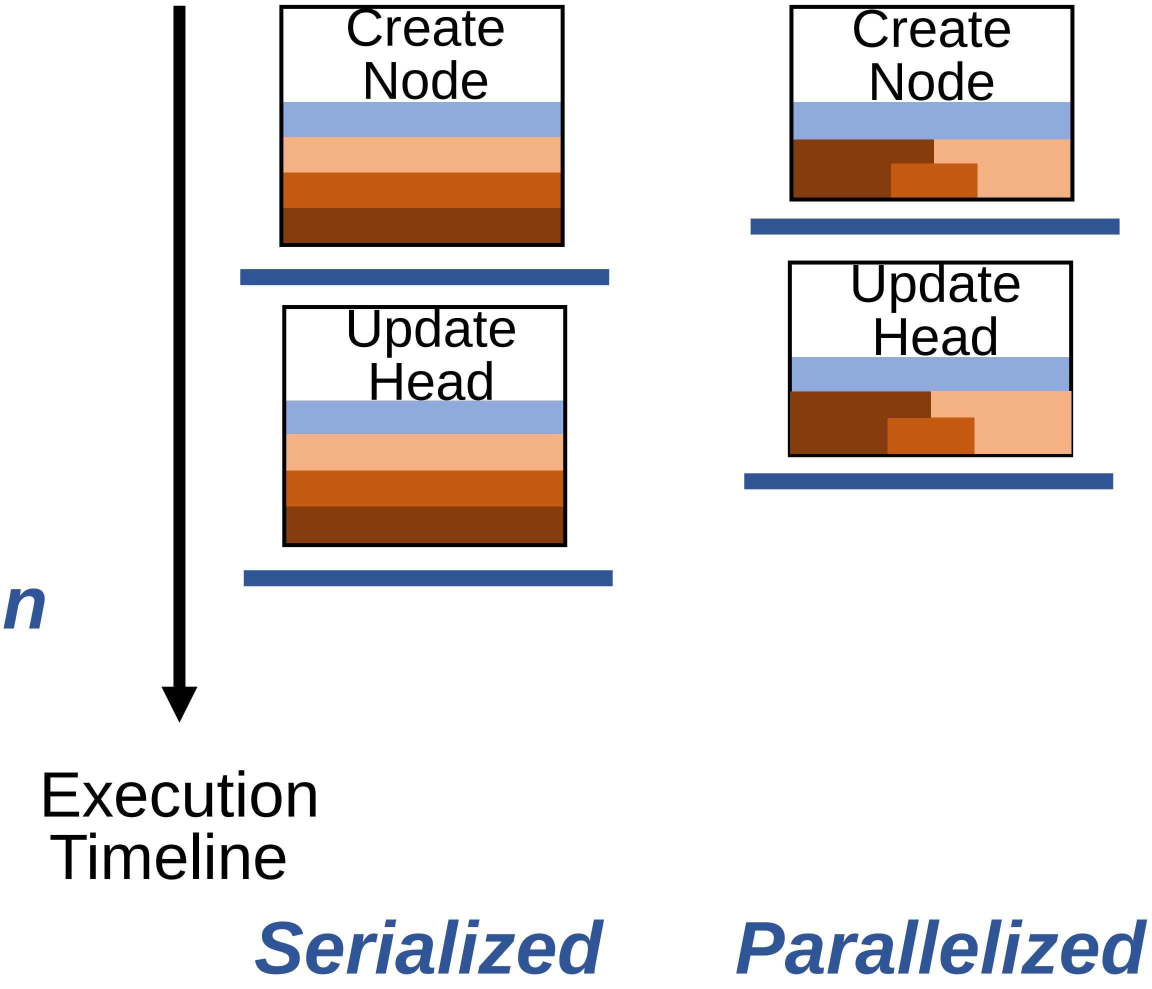
Future

Performance Improvement

🟧 Backend operations
🟦 Original writeback latency

Janus:

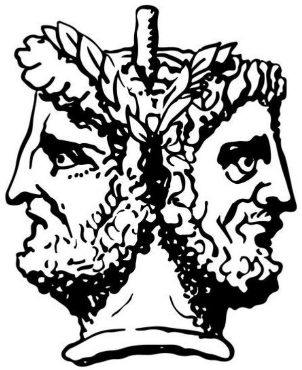
- ***Parallelization***



Parallelization ***reduces the latency*** of each operation

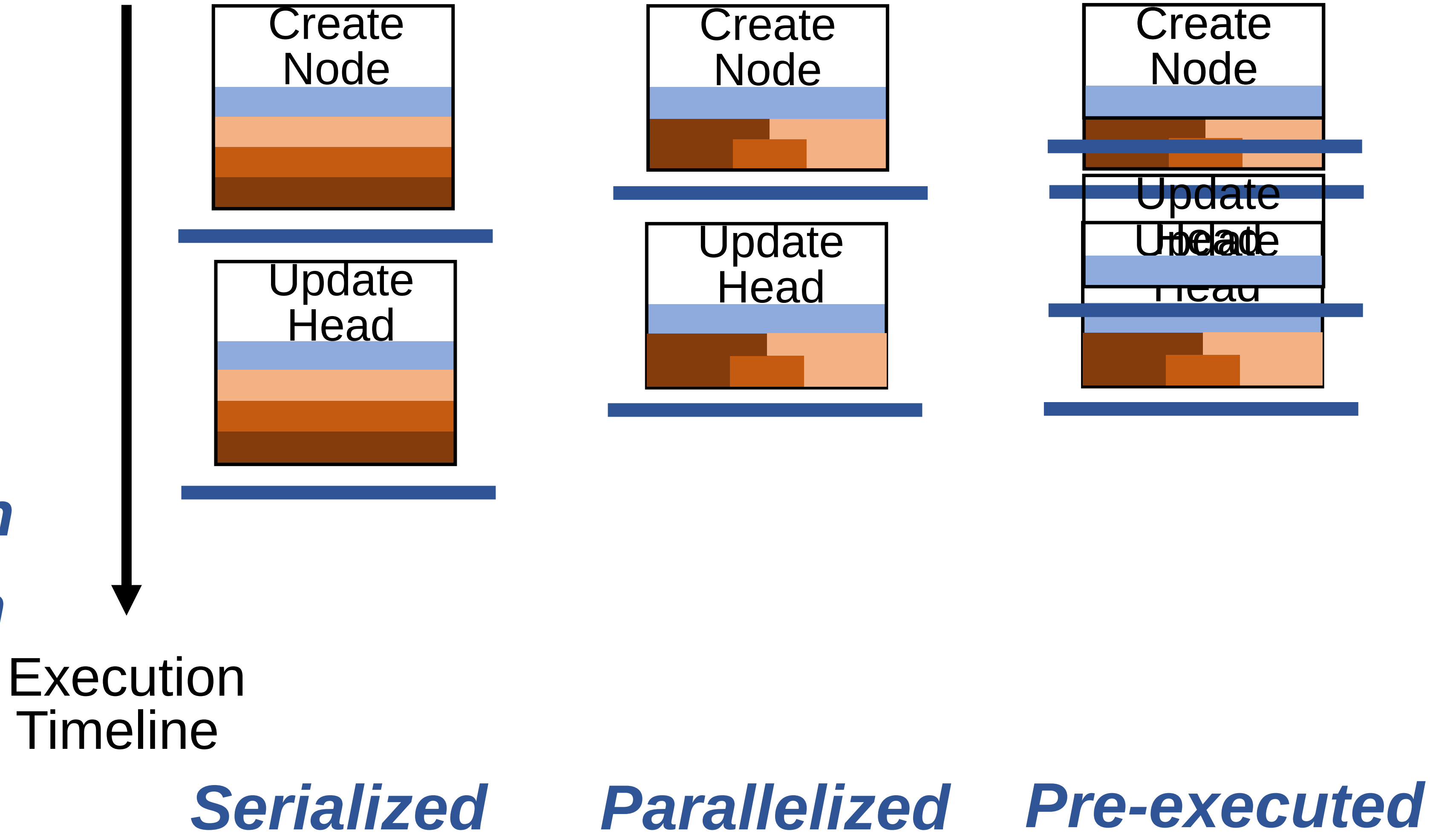
Performance Improvement

🟧 Backend operations
🟦 Original writeback latency



Janus:

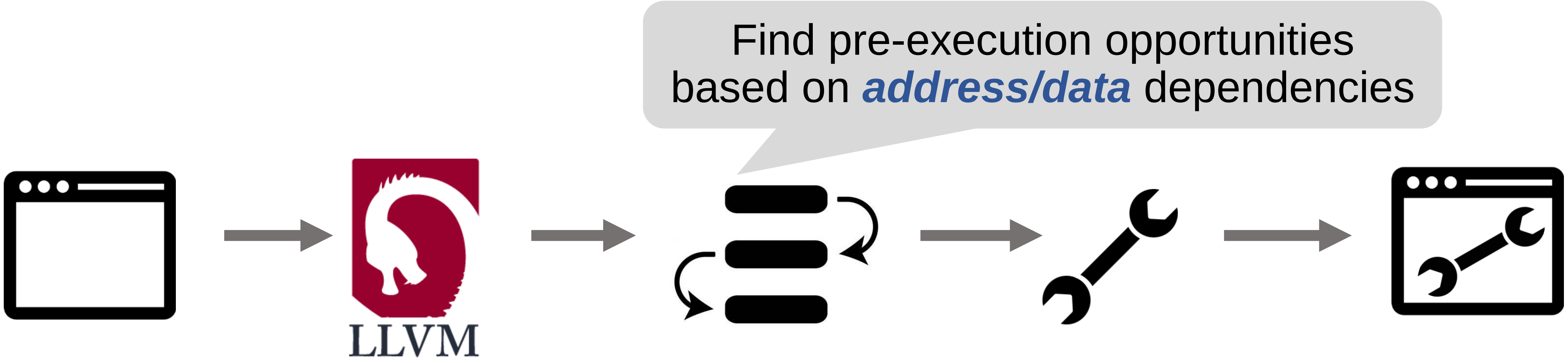
- ***Parallelization***
- ***Pre-execution***



Pre-execution moves the latency ***off the critical path***

Software-Hardware Co-design

Automated software instrumentation for pre-execution



Find pre-execution opportunities based on ***address/data*** dependencies

Original Program

Compiler Pass

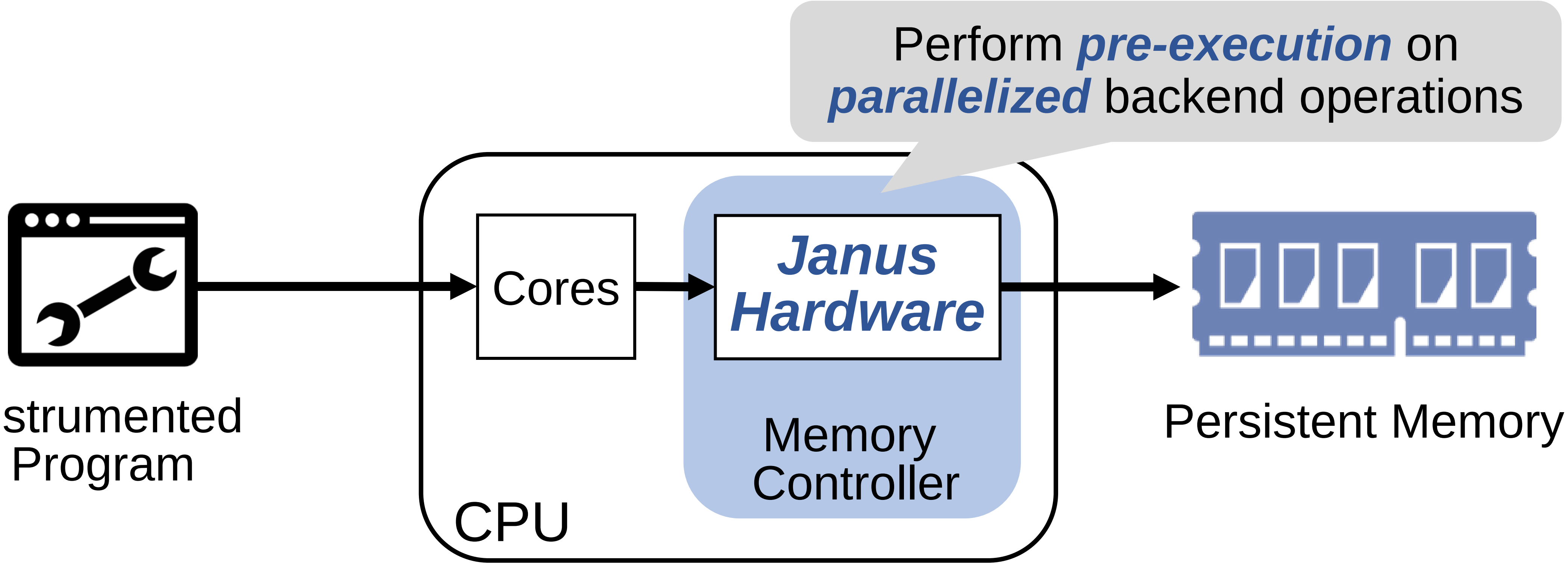
Dependency Analysis

Pre-execution Hint Instrumentation

Instrumented Program

Software-Hardware Co-design

Janus hardware executes the instrumented program


Perform ***pre-execution*** on ***parallelized*** backend operations

Evaluation Methodology

- **Platform -** Gem5 Simulation

Processor	*Out-of-Order, 4GHz*
L1 D/I, L2 cache	*64/32KB, 2MB per core (shared)*
Backend memory operation cache	*512KB per core for each operation (shared)*
Backend memory operation units	*4 units per core*

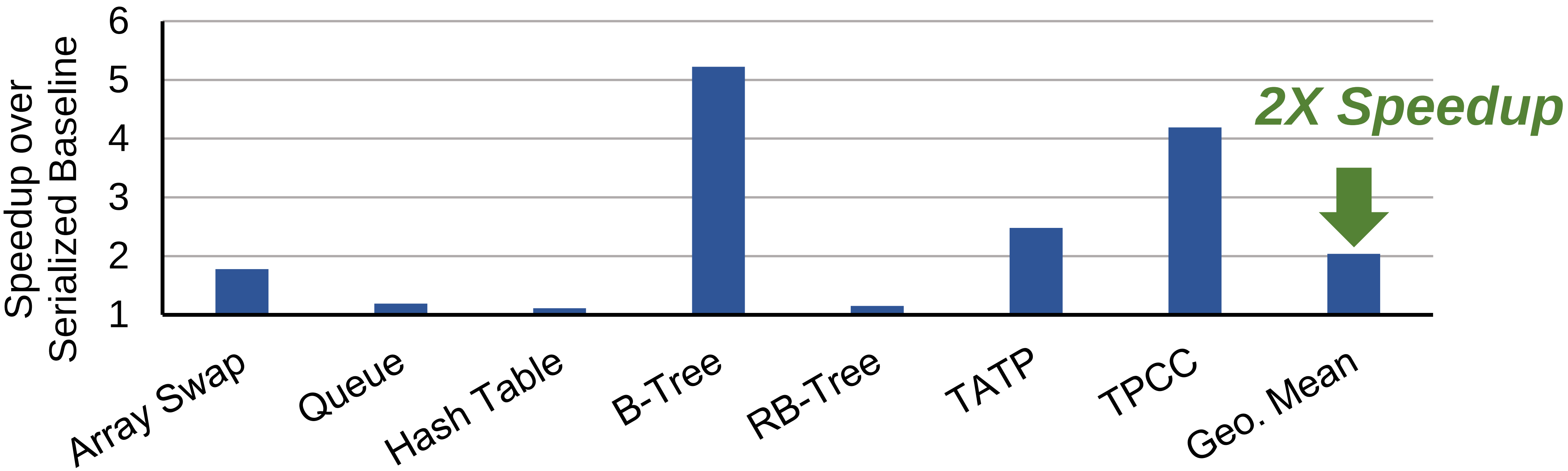
- **Design points**
  - **Baseline**: all backend operations are serialized
  - **Janus**: pre-execute parallelized backend operations

Evaluation Methodology

- **Storage-class workloads**

Array Swap	*Randomly swap two locations in an array*
Queue	*Randomly push/pop items to a queue*
Hash Table	*Randomly insert key-values to a hash table*
B-Tree	*Randomly insert key-values to a b-tree*
RB-Tree	*Randomly insert key-values to a rb-tree*
TATP	*Add items to a telecommunication table with the TATP input generator*
TPCC	*Add items to a hash table with the TPCC input generator*

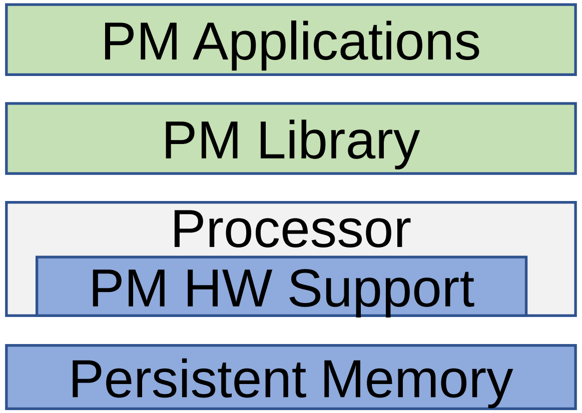
Performance



Janus provides ***2X speedup*** on average

Summary

System Stack for
Persistent Memory



Software Support for Persistent Memory

Testing frameworks for
failure-recovery issues

PMTest
[ASPLOS'19]

XFDetector
[ASPLOS'20]

A test case generator for
better ***testing efficiency***

PMFuzz
[ASPLOS'21]

Hardware System for Persistent Memory

Efficient and ***secured***
persistent memory hardware

Software-hardware co-designs
[HPCA'18, ISCA'19, PACT'21]

***New security
vulnerabilities*** in
Intel's persistent memory

[In-submission]

Future Directions

- Adaption of PM into larger scale systems
  - How can datacenter-scale workloads better utilize persistent memory?
  - How can we redesign the networking system to better leverage the lower latency of PM?
- Integration of computation logic into PM
  - What computation logic can we place on PM to accelerate memory-intensive workloads?
- More security challenges of PM systems
  - How can we design software systems for PM that ensures existing security guarantees?

Toward Failure Recoverable And Secured Persistent Memory Systems

IEEE Data & Storage Symposium 2022

Sihang Liu

University of Virginia
(Current)

University of Waterloo
(Joining in 2023 as an Assistant Professor)

June 9, 2022