PMEM-Spec: Persistent Memory Speculation

(Strict Persistency Can Trump Relaxed Persistency)

Jungi Jeong and Changhee Jung
Purdue University

Session 6A: Hardware for Crash Consistency
NVMW 2021

PURDUE
UNIVERSITY

Department of Computer Science

Executive Summary

- Persistency Model defines persist-orders for failure-recovery

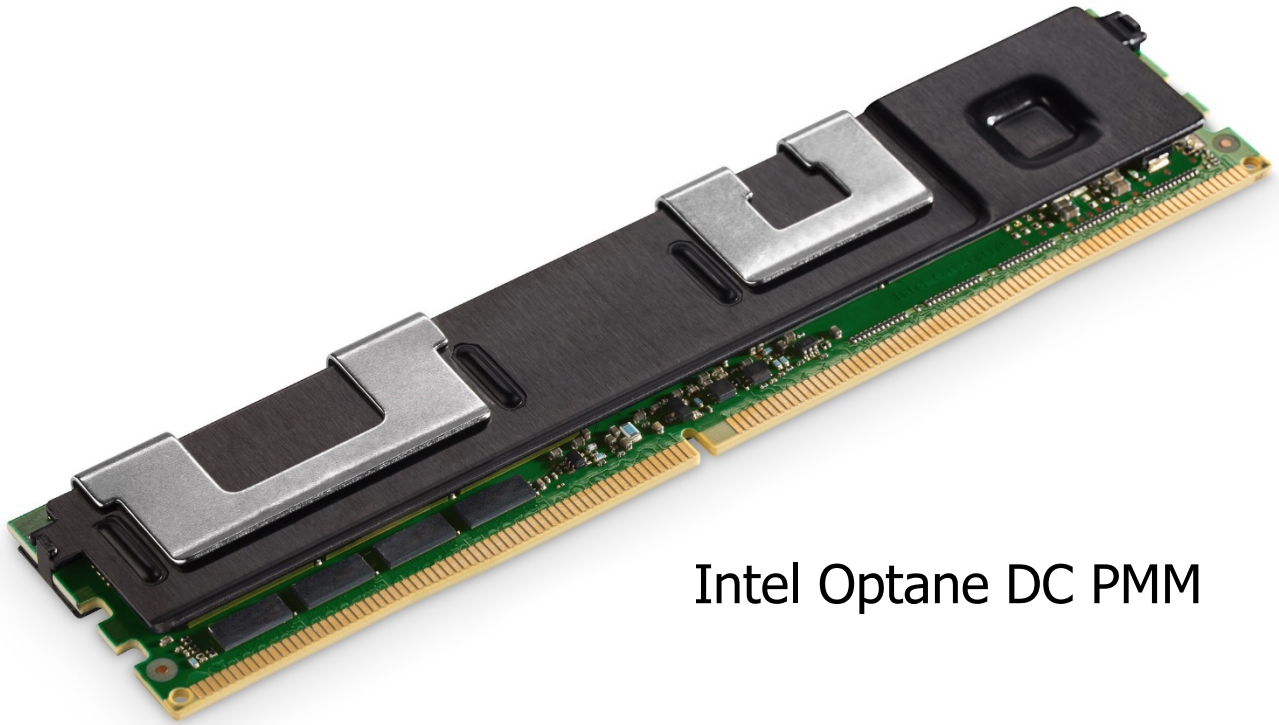
- Challenge for strict persistency ➔ slow!
  - (In general) the more relax, the better performance
  - But relaxing increases programming difficulty (like memory consistency)

- Persistent Memory Speculation
  - HW/SW codesign for strict persistency
  - 10%~27% speedup compared to relaxed persistency
  - First demonstration of strict persistency outperforming relaxed persistency

Persistent Memory (PM) is Here!

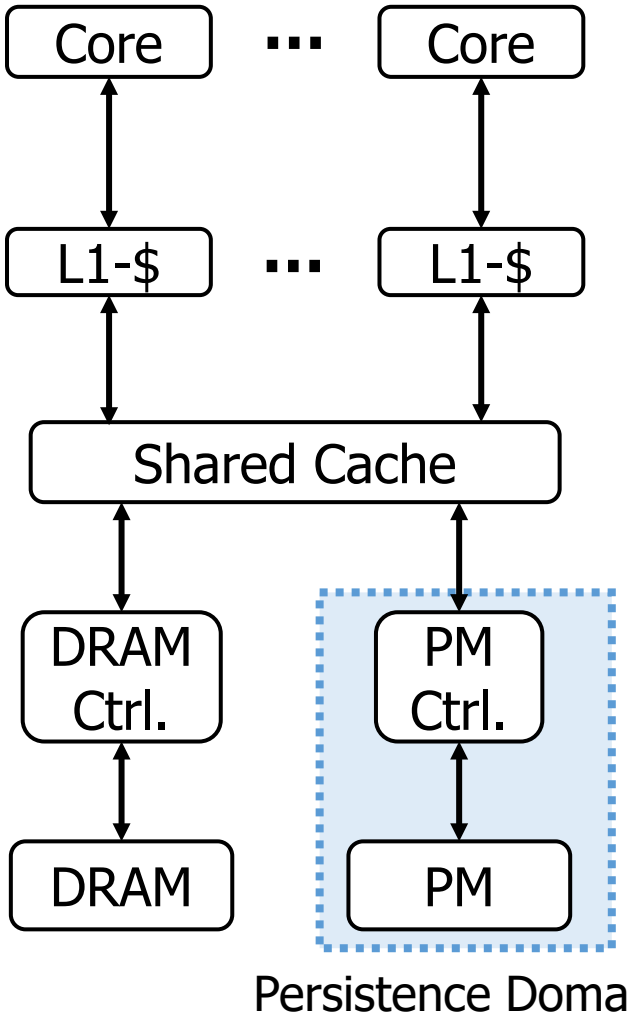
- User-space access to Non-Volatile Memory

- Enables recoverable applications



Intel Optane DC PMM

PM Programming Challenges



- PM Stores must be:
  - **Atomic**
    : via write-ahead logging*
    or shadow paging**
    or idempotent processing$

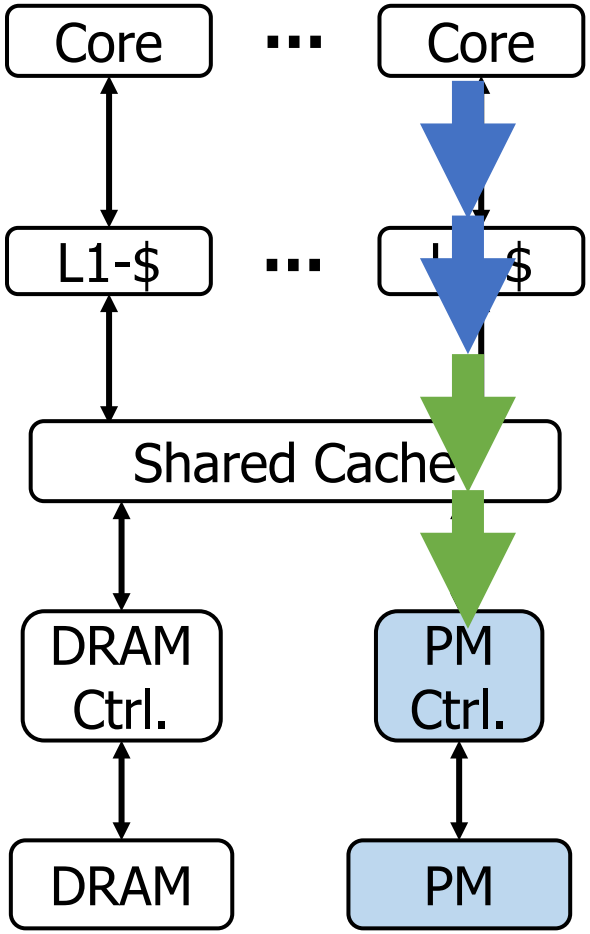
* ASPLOS 2011, ASPLOS 2016, ASPLOS 2017, EuroSys 2017

** ASPLOS 2020 $ MICRO 2018

Ordered (a.k.a. persist-order)
: flush & fence instructions

Target of This Study

PM Store Ordering – Strict Persistency



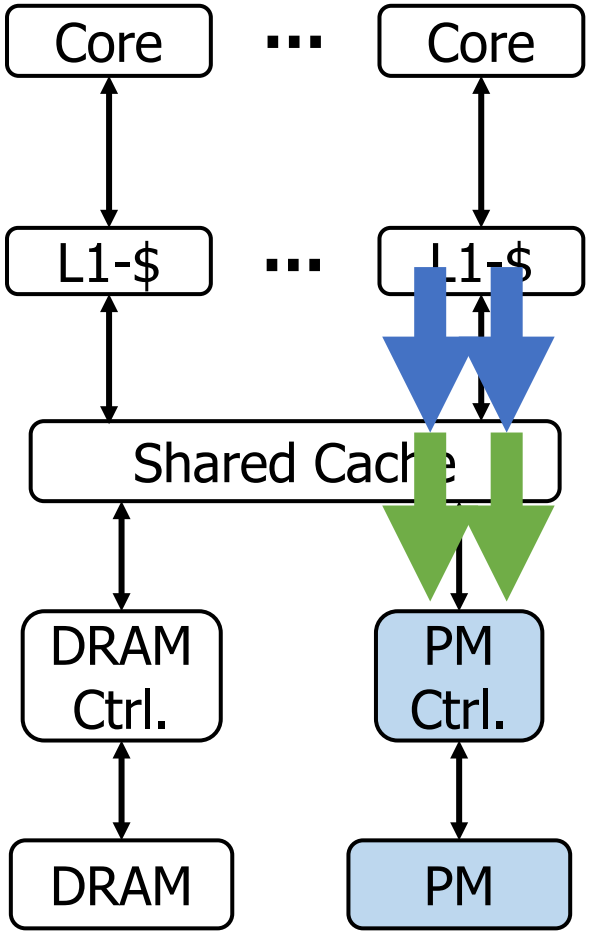
Ex)

Log A
Flush & Fence
Log B
Flush & Fence
Data A
Flush & Fence
Data B
Flush & Fence

Flush & Fence for each PM store

Minimal programming burden (compiler-support)

PM Store Ordering – Relaxed Persistency



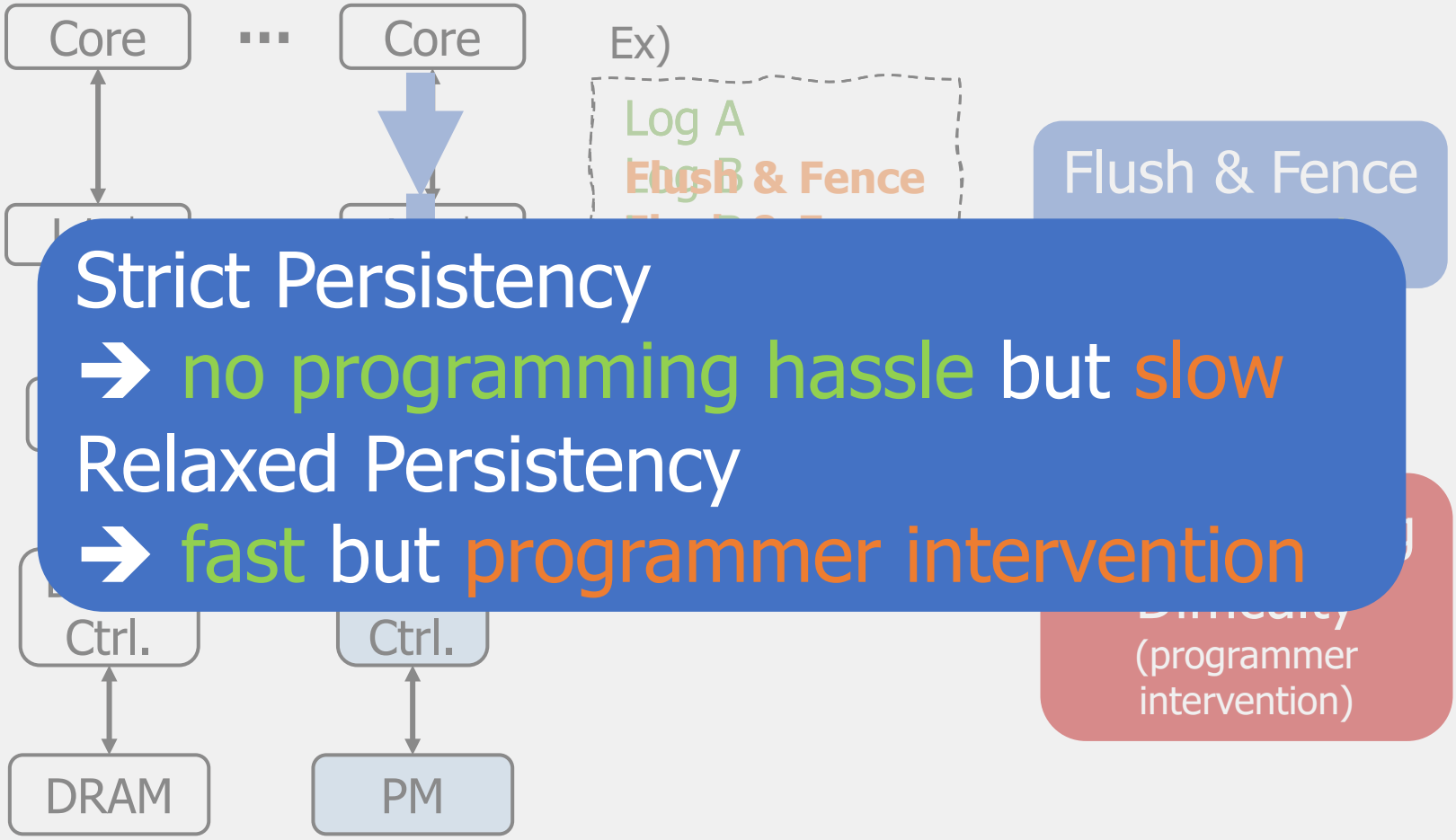
Ex)

Log A
Log B
Flush & Fence
Data A
Data B
Flush & Fence

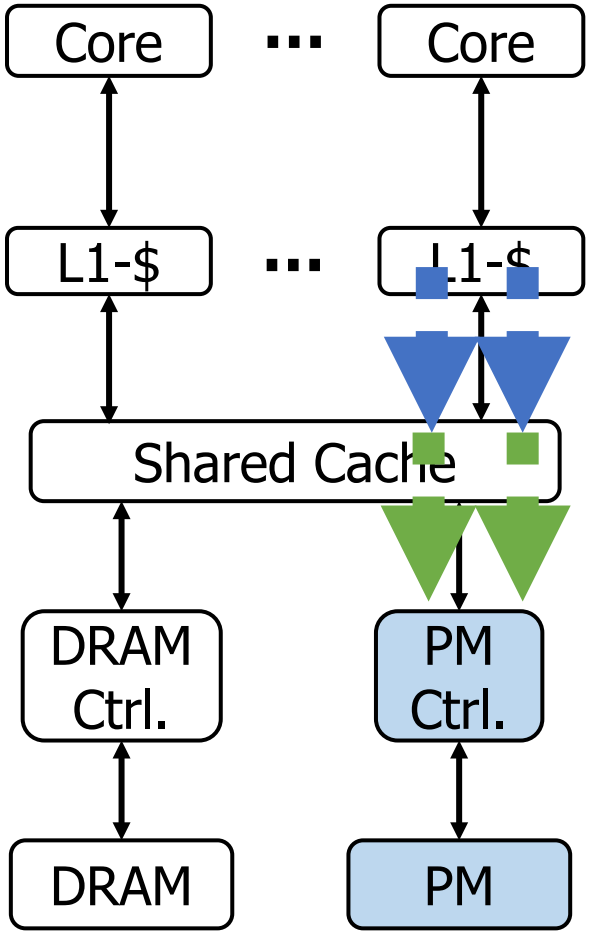
Flush & Fence per epoch

Programming Difficulty (programmer intervention)

PM Store Ordering – Relaxed Persistency



Related Work*: FENCE Overheads



#1. Reducing FENCE costs

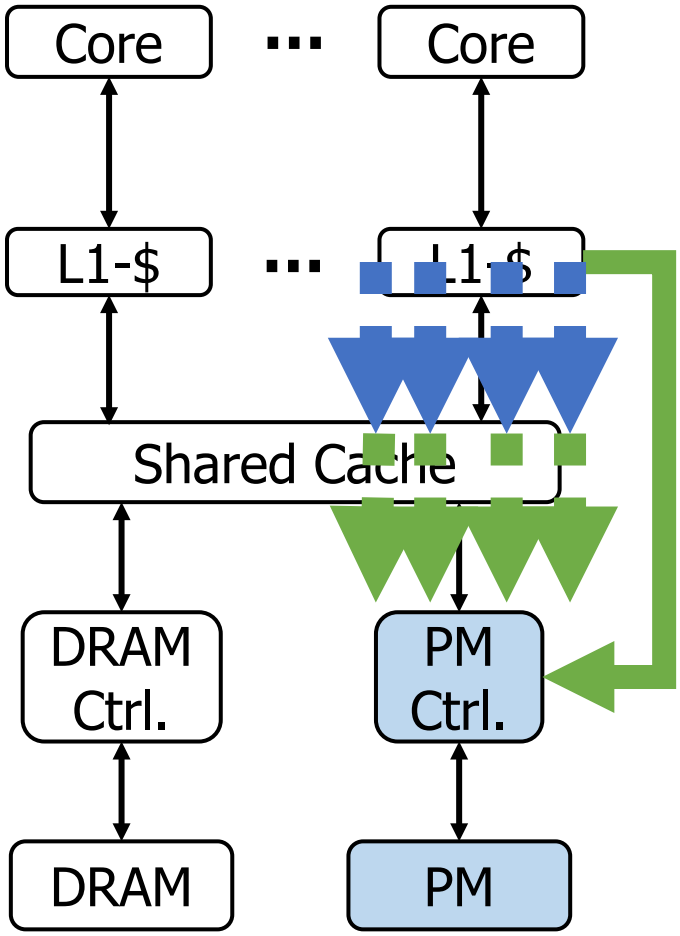
[MICRO 2016] & [ASPLOS 2017]

- Hiding fence latency
- Delegating the persist-order to HW

#2. Reducing # of FENCEs

* [MICRO2016], [ASPLOS2017], [ISCA2020], [MICRO2020]

Related Work*: FENCE Overheads



#1. Reducing FENCE costs

[MICRO 2016] & [ASPLOS 2017]

- Hiding fence latency
- Delegating the enforcement to HW

#2. Reducing # of FENCEs

[ISCA 2020]

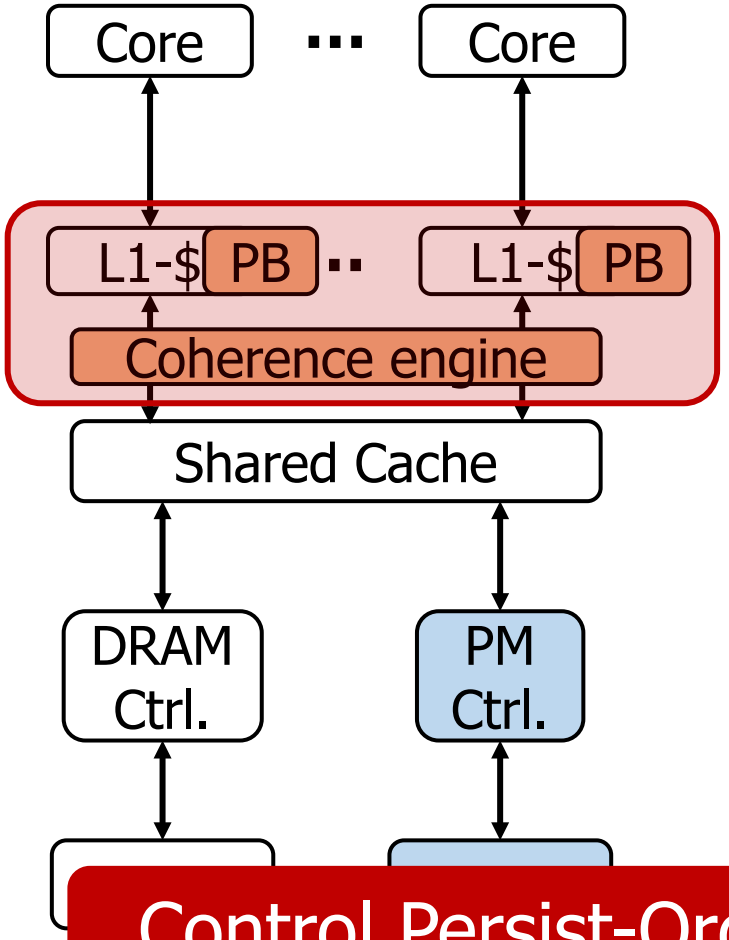
- Further relaxing constraints

[MICRO 2020]

- Multiple store paths to PM

* [MICRO2016], [ASPLOS2017], [ISCA2020], [MICRO2020]

Related Work*: HW Complexity



Intra-thread Persist-Order

L1-$-side Buffers (PBs)

: governs *cache-flush orders* based on persistency model

Inter-thread Persist-Order

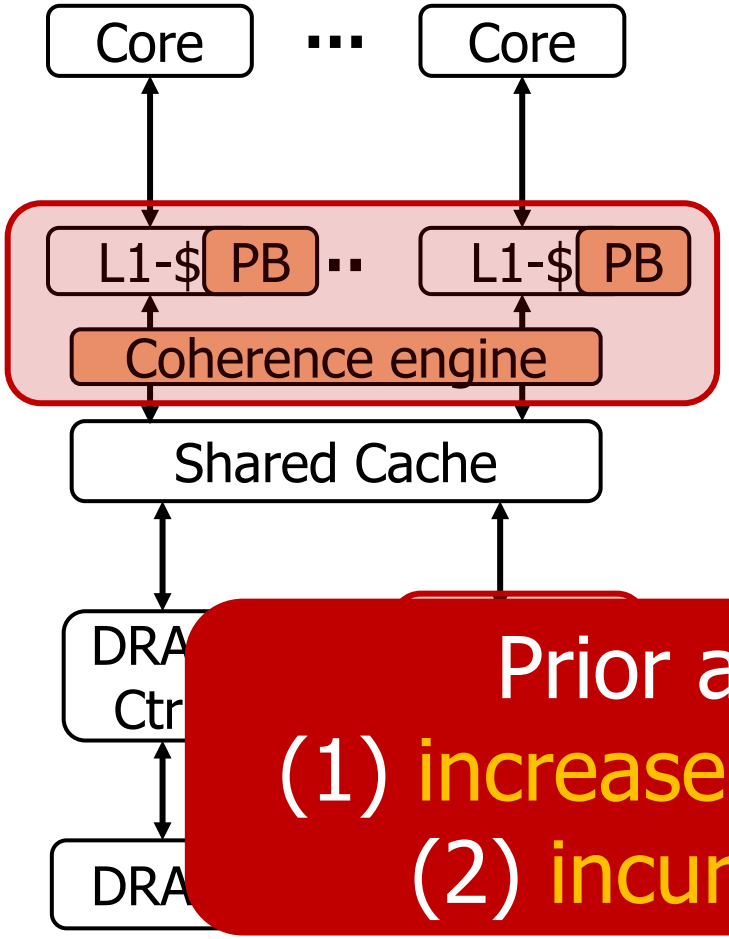
Cache-coherence extensions

: detects inter-thread dependency

Control Persist-Orders in Private L1-Caches

* [MICRO2016], [ASPLOS2017], [ISCA2020], [MICRO2020]

Related Work*: Challenge



L1$-writeback vs. PB Flush

: PB Flush must happen before L1-writeback for a given block

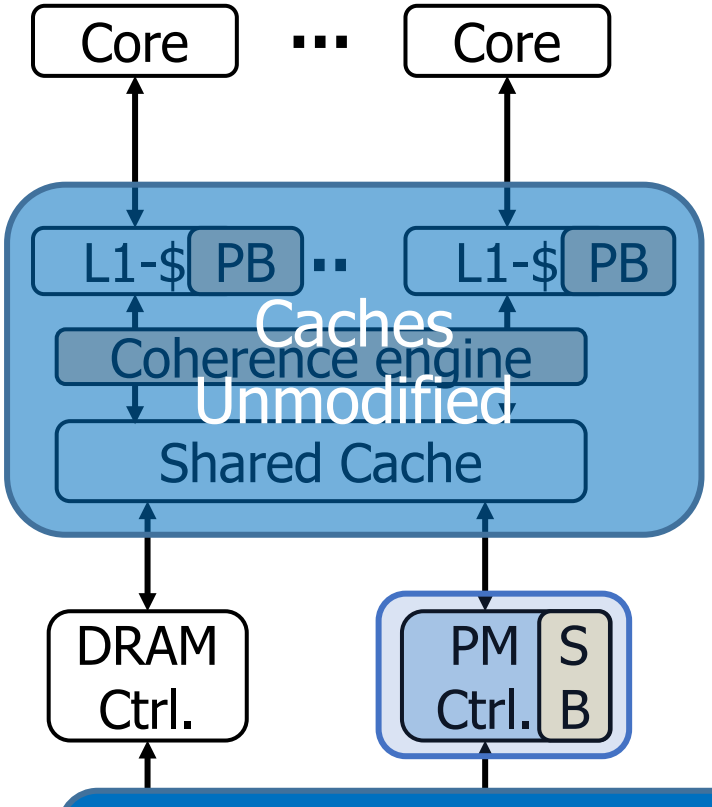
Solutions of Prior Works

- [MICRO2016]
  : *cache-coherent* PBs
- [ASPLOS2017]
  : *sticky-bit* in LLC for tracking &
  rs
  ts in PBs
  0]

Prior approaches:
(1) increase HW complexities
(2) incur extra latency

* [MICRO2016], [ASPLOS2017], [ISCA2020], [MICRO2020]

PMEM-Spec: Persistent Memory Speculation

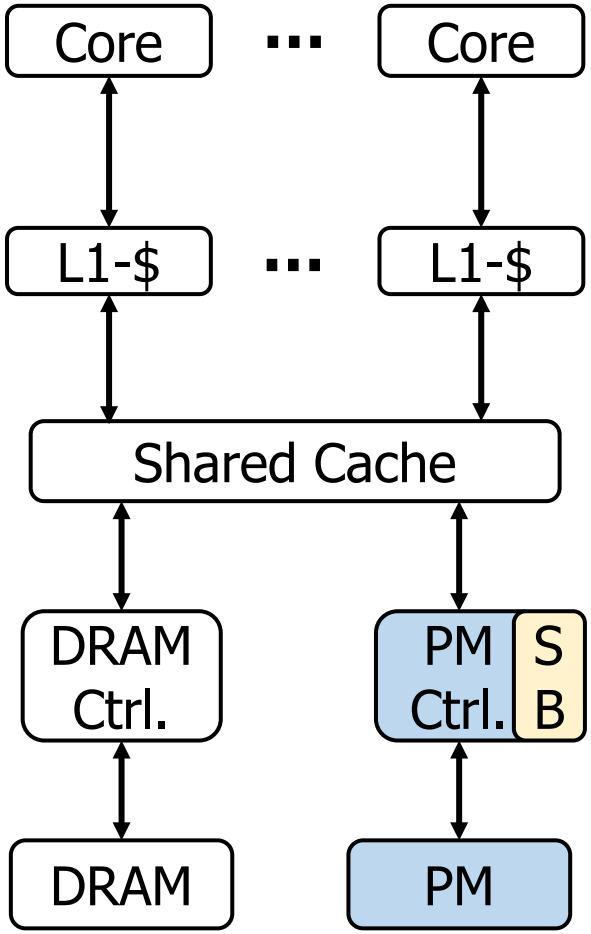


Minimal HW changes

Minimal Program Changes
(like Strict Persistency)

HW/SW Codesign for
High Performance Strict Persistency

PMEM-Spec Key Ideas



#1. Speculate PM Accesses

➔ With *Separated* load/store paths to PM

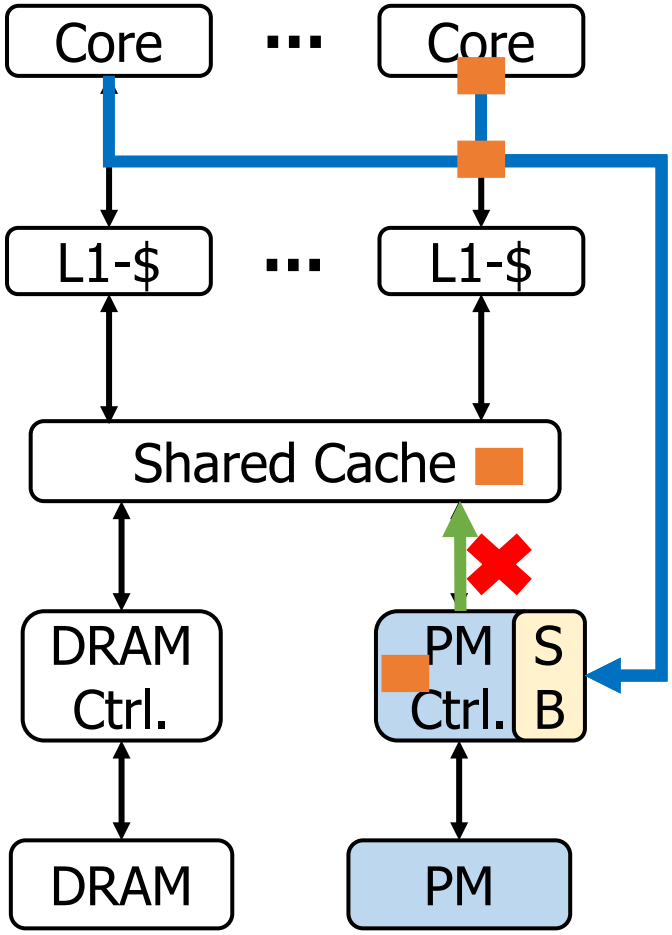
#2. Detect ordering violation (Misspeculation) in HW

➔ With Arch/Comp interaction

#3. Recover from Misspeculation in SW

➔ With *failure-atomic* SW as *virtual* power failure

Separated Load/Store Paths to PM



Persist-path:
FIFO store path to PM

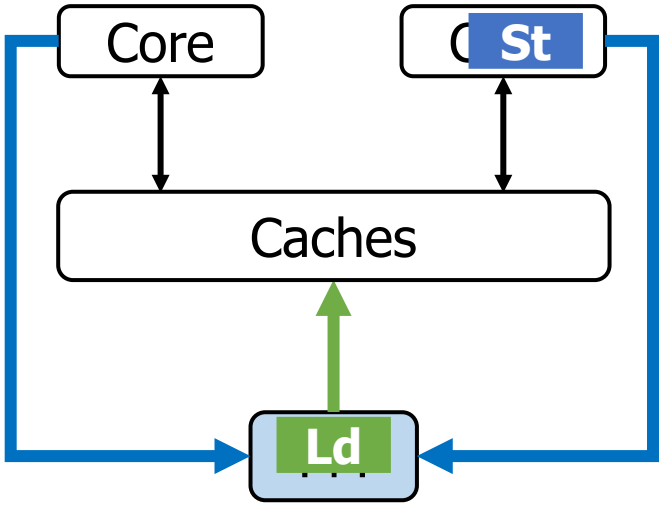
➔ Connect SQ to NVM
➔ Bypass caches
➔ Drop cache writebacks

Regular-path:
Data path through caches

➔ Serves NVM reads only

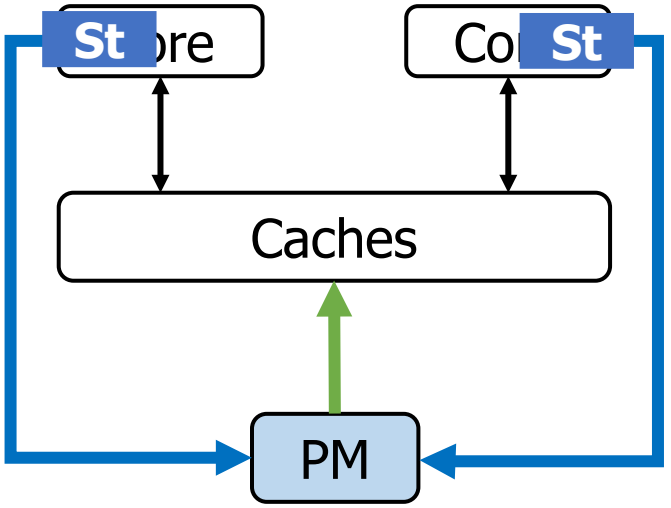
What does PMEM-Spec Speculate?

Load Speculation



: PM load must read latest value from PM

Store Speculation

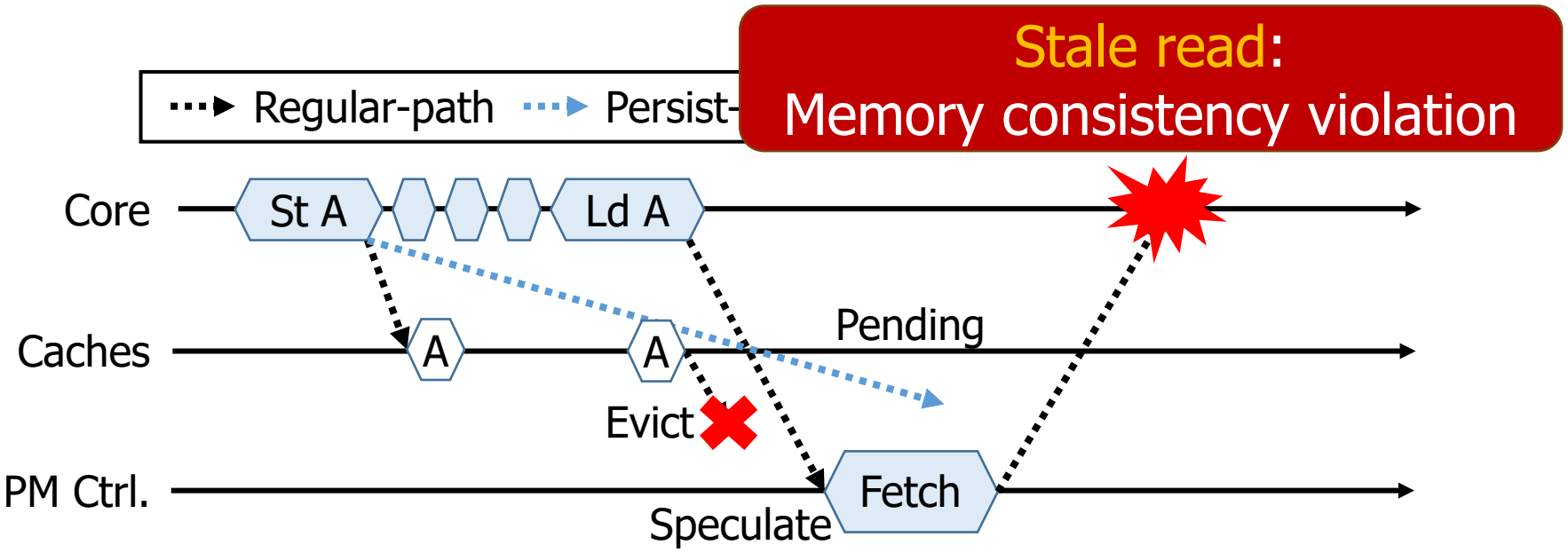


: PM stores must arrive in the correct order

Watch Out for Ordering Violation (Misspeculation)!

Load Misspeculation

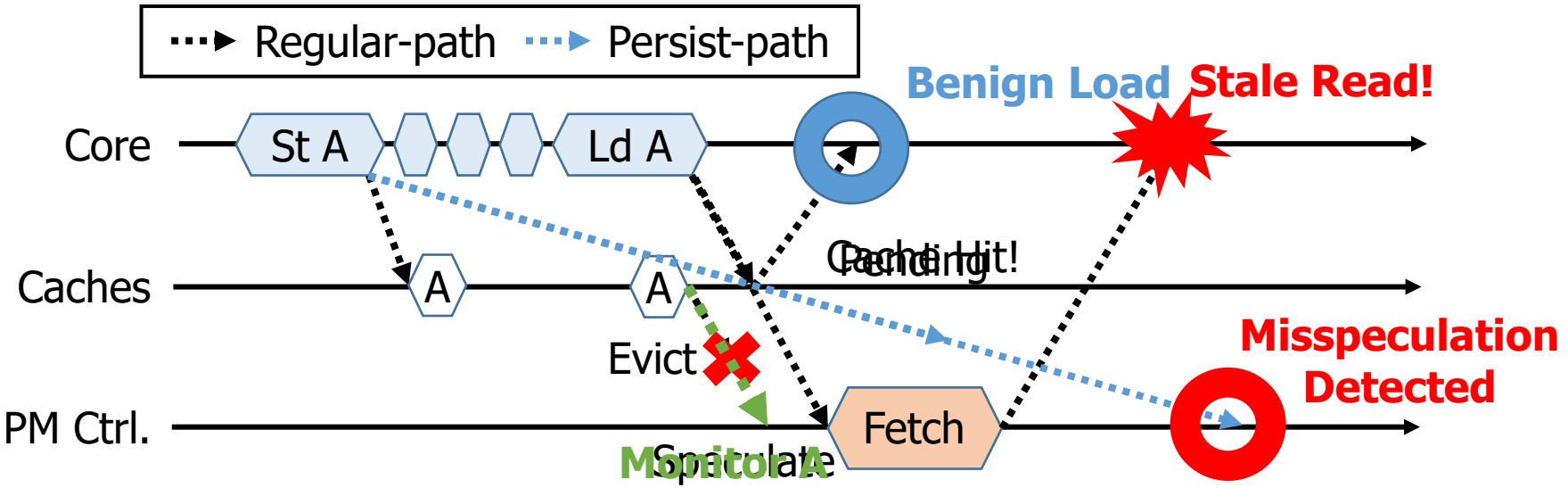
- Symptom: **stale reads**
  - If prior stores are pending in the persist-path
- Cause: latency differences in separated load/store paths



Detecting Load Misspeculation

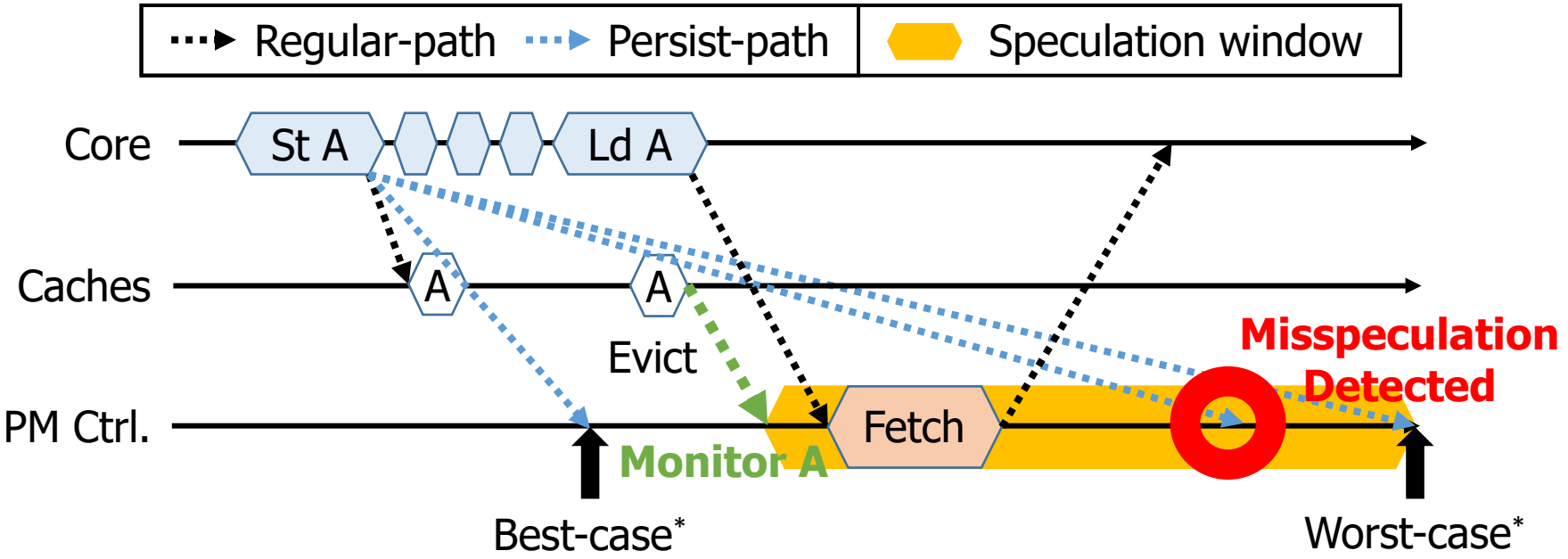
- Observation: if blocks in caches, loads never misspeculate
- Key idea: monitoring *recently evicted* blocks
  - For whether they are overwritten by stores

Q. How long should we monitor?



Detecting Load Misspeculation

- Monitor evicted blocks *until the worst-case persist latency*
  - Fixed in the HW design time*
- **Speculation Window**
  - Starts on LLC-eviction of dirty blocks (not updating PM data)
  - If blocks *being fetched & overwritten* within the window, the fetch was stale

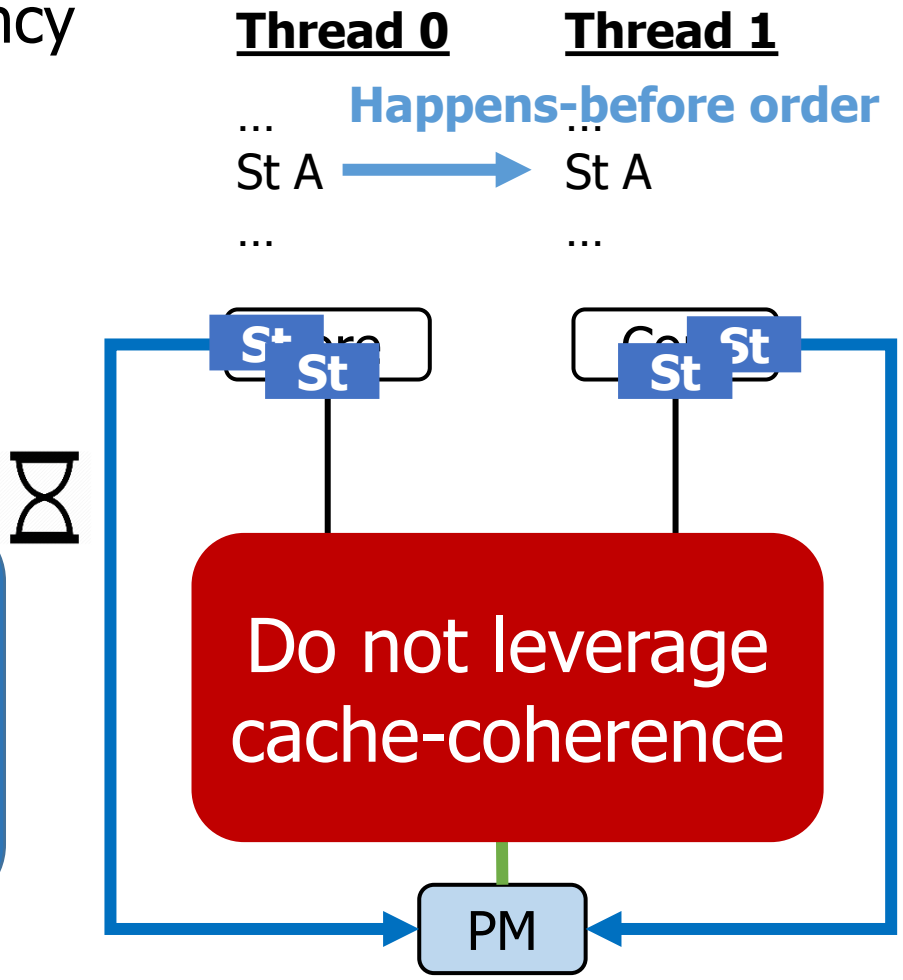


* Best & worst persist-path latencies are determined in HW design time based on HW specification.

Store Misspeculation

- Cause: inter-thread dependency
  - Stores to the *same address* from multiple threads
- Symptom: **out-of-order persists**

Q. How to capture the store-order between threads?
(without cache-coherence)



Detecting Store Misspeculation

- Data-Race-Free (DRF) programs
  - Inter-thread dependency *always* happens in critical sections
- Observation:

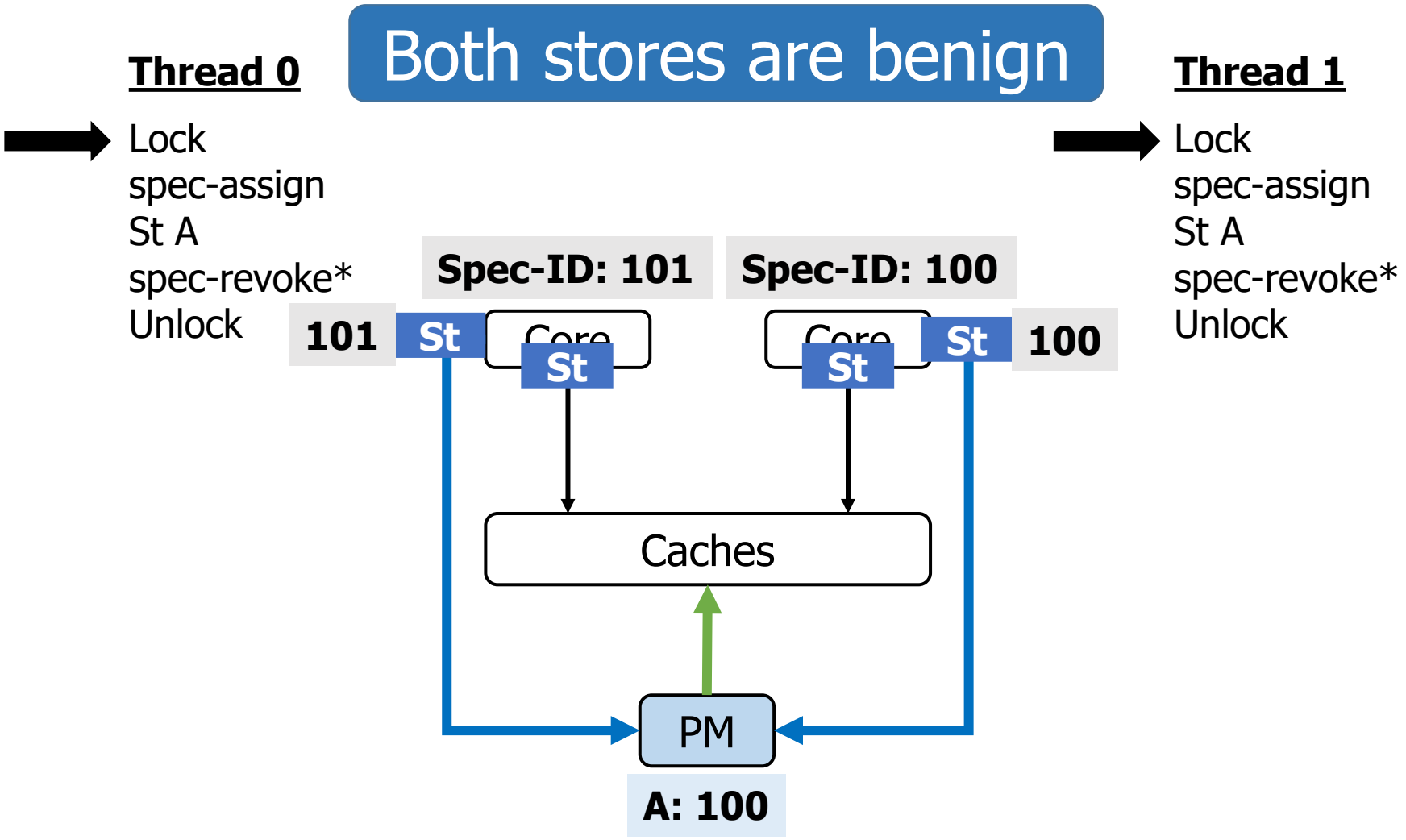
  Critical section execution order == Inter-thread store-order

Thread 0	**Thread 1**
Lock	Lock
St A	St A
spec-assign	spec-assign
Unlock	Unlock
spec-revoke	spec-revoke
Unlock	Unlock

- To convey critical section execution order to HW
  - Speculation ID
    : global-counter incremented when entering critical section
  - Arriving lower IDs after higher IDs ➔ out-of-order arrivals
  - New instructions to assign/revoke the speculation ID to a thread (*spec-assign* / *spec-revoke*)

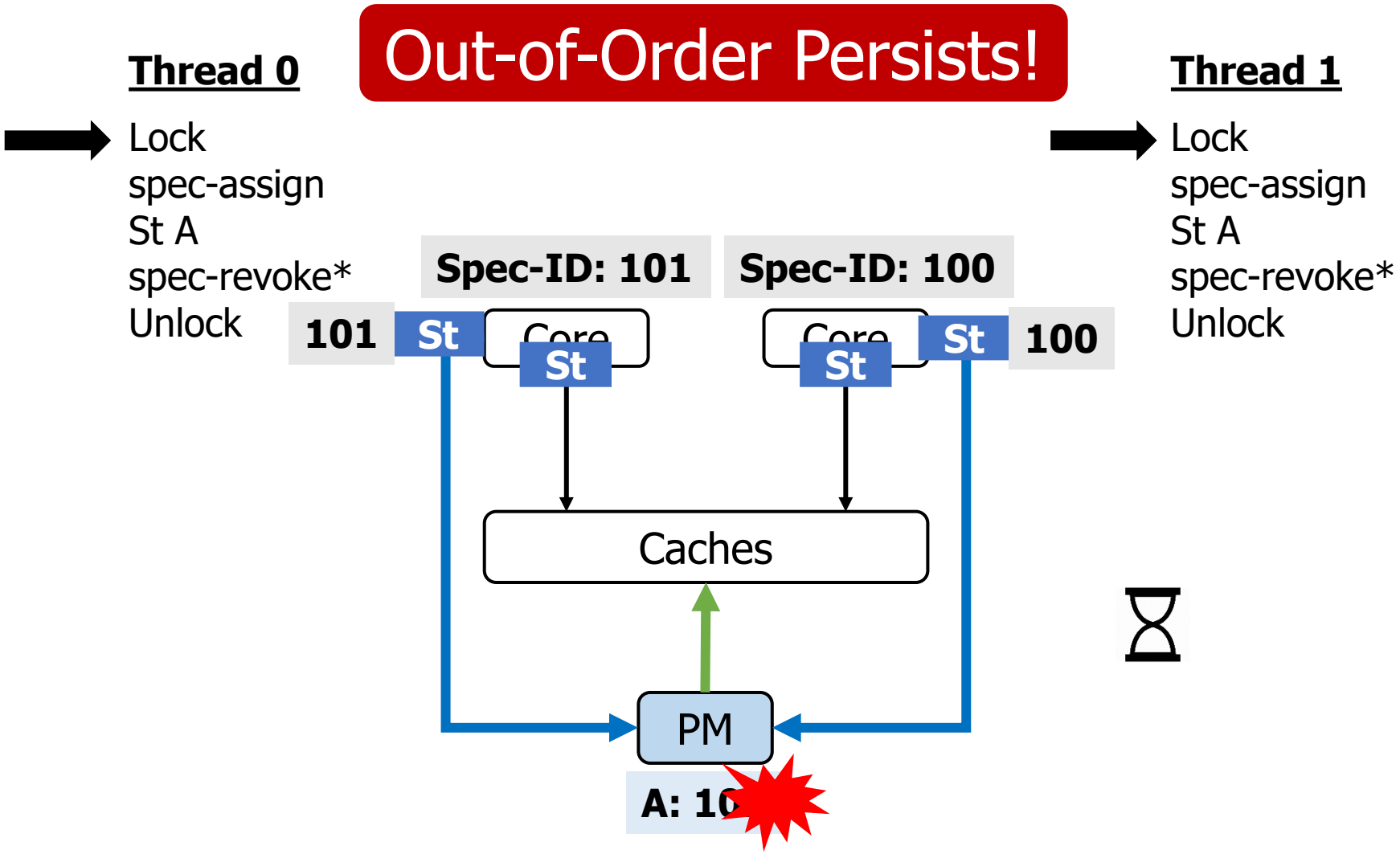
No Programmer Annotation
(compiler-generated codes)

Speculation ID Ex) Benign Store



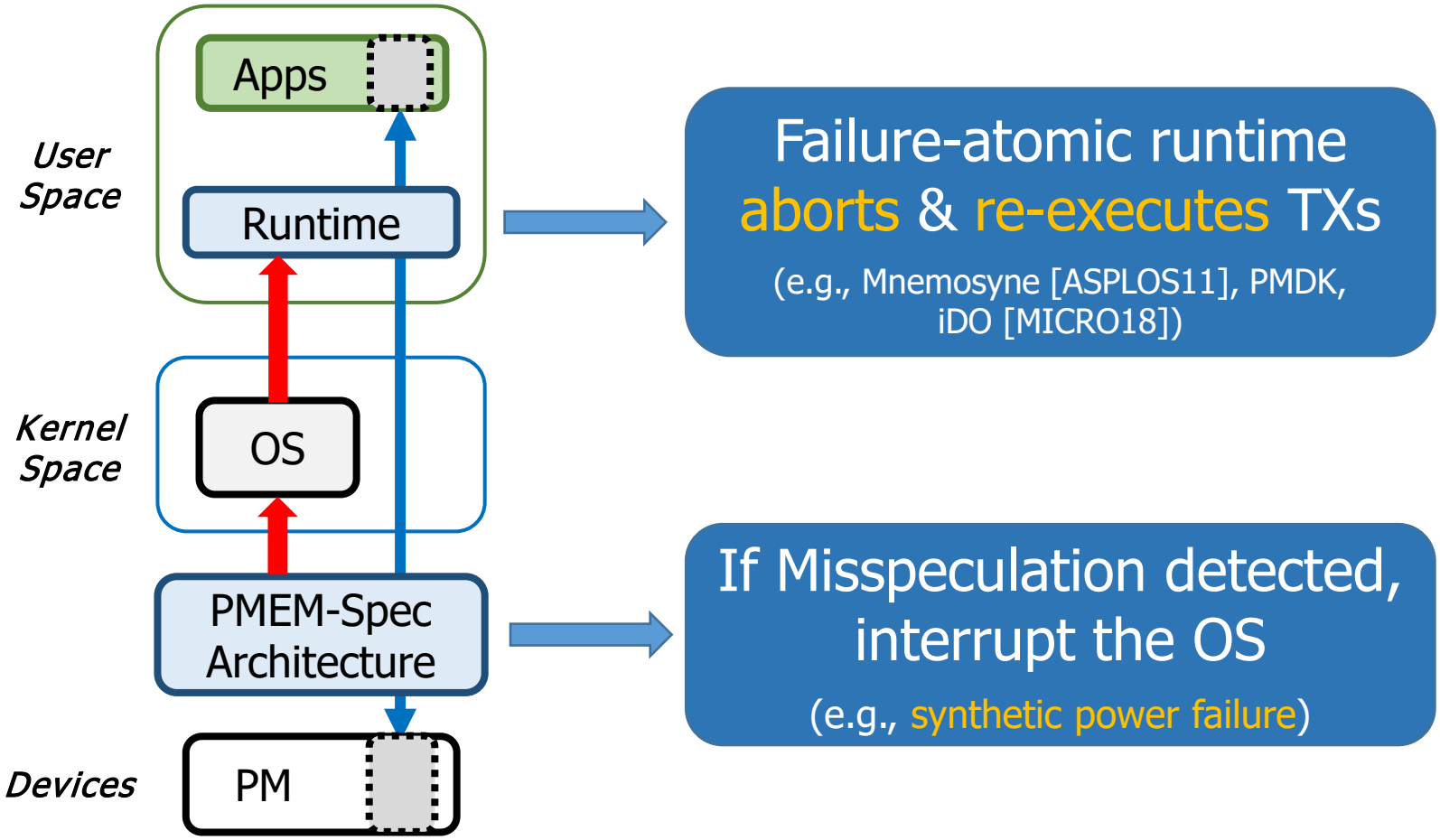
* PMEM-Spec untags the speculation ID when exiting critical sections. Please refer to the paper for details.

Speculation ID Ex) Out-of-Order Persists

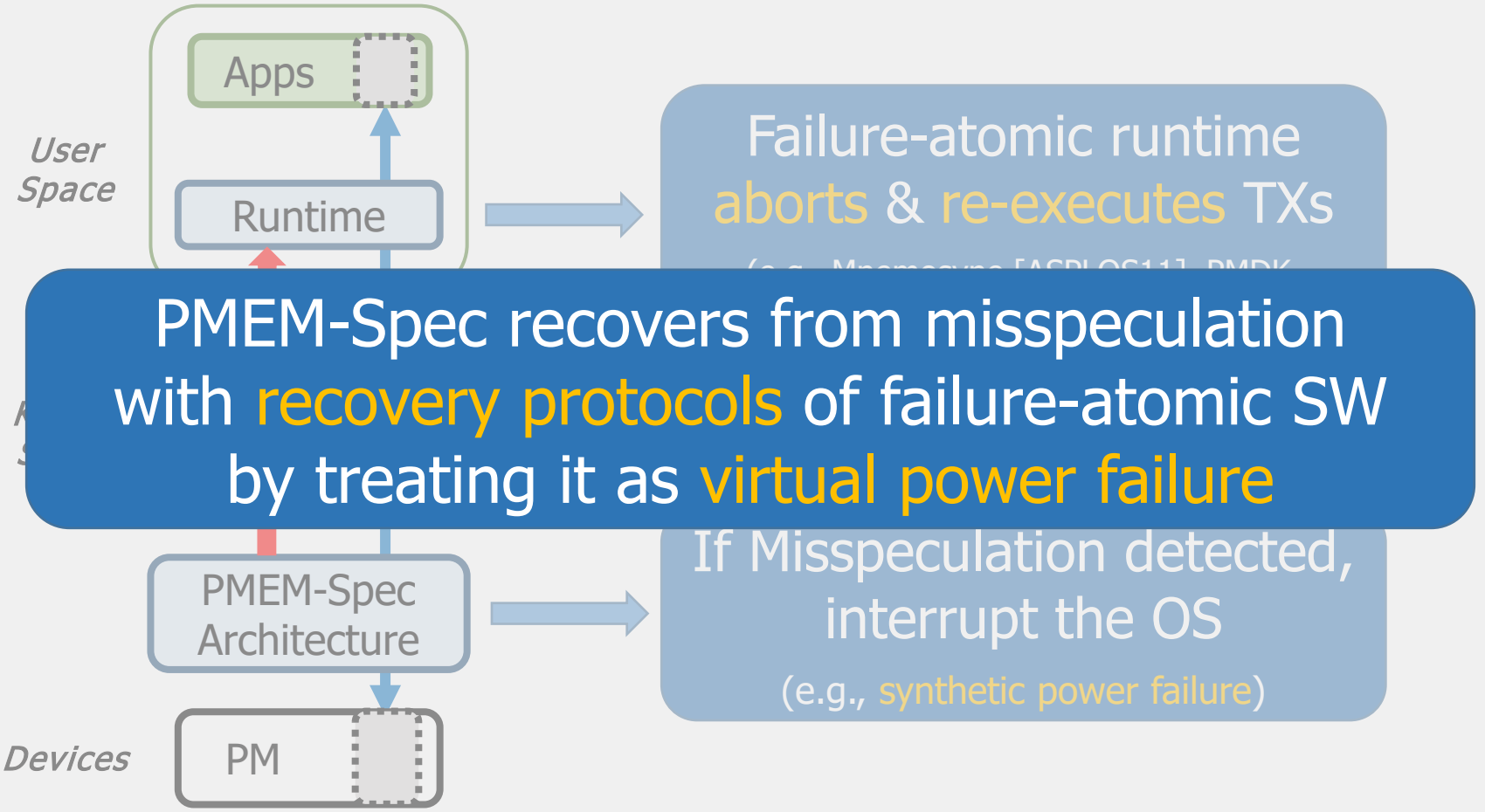


* PMEM-Spec untags the speculation ID when exiting critical sections. Please refer to the paper for details.

Misspeculation Recovery



Misspeculation Recovery



Methodology

- Full system simulation with gem5
  - Linux kernel version: 4.8.13
  - Ubuntu 16.04

Processor	8-core, OoO, 2GHz, x86
L1 I/D cache	Private, 32/64KB, 4-way, 2ns
L2 cache	Shared, 16MB, 16-way, 20ns
PM Controller	32/64-entry read/write queue
PM	Read: 175ns, write: 94ns
Persist-Path	20ns

- Benchmarks

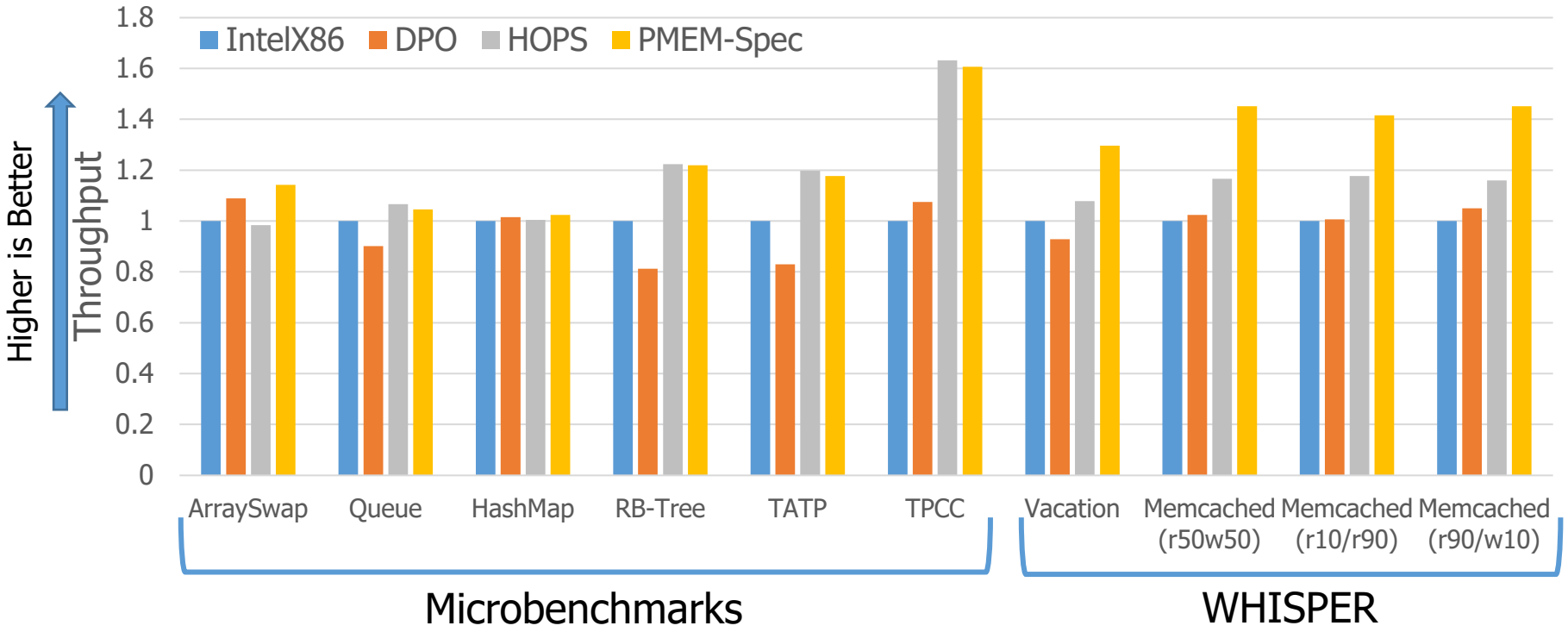
Microbench	Concurrent Queue, Array Swap, HashMap, RB-Tree, TATP, TPCC
WHISPER*	Vacation, Memcached

* S. Nalli et al., ASPLOS 2017.

- Comparing schemes
  - Intel X86 (baseline): Epoch Persistency
  - DPO [MICRO'16]: Strict Persistency
  - HOPS [ASPLOS'17]: Epoch Persistency
  - PMEM-Spec: Strict Persistency

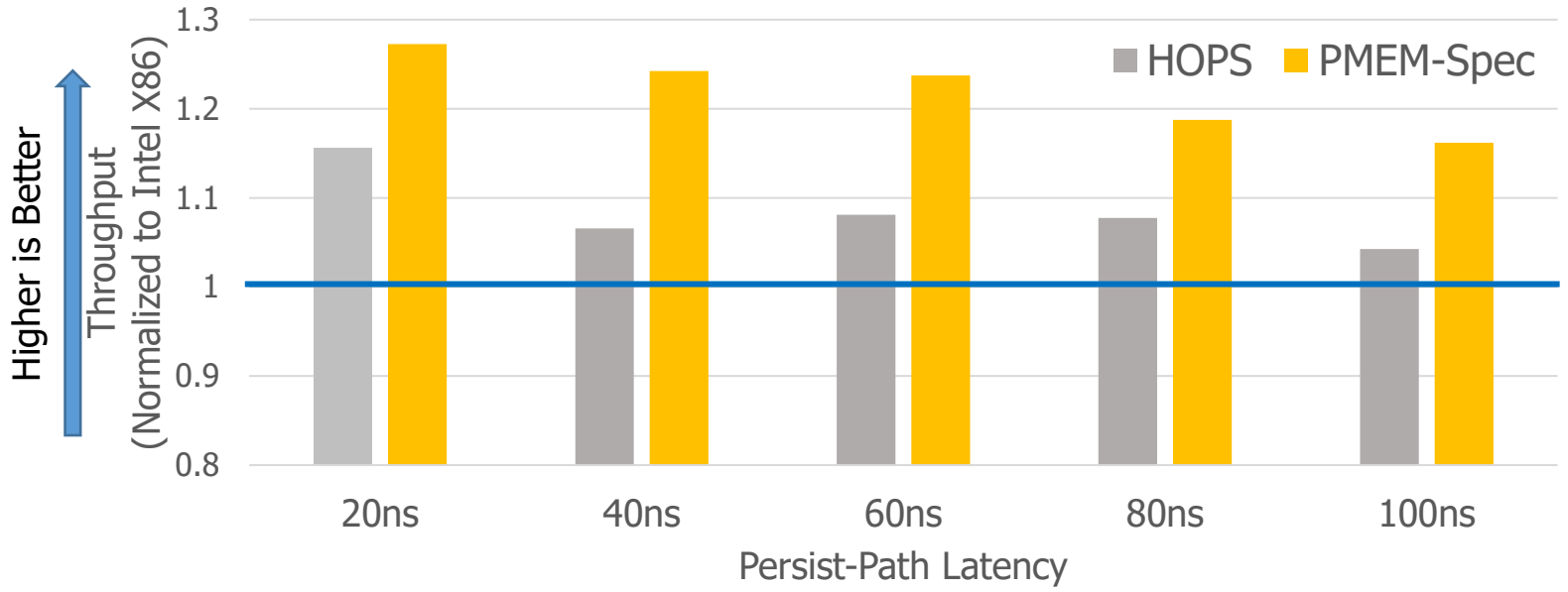
Evaluation – Throughput

- Microbenchmarks: similar to HOPS (Epoch Persistency)
  - Tiny transactions ➔ less room for speculation
- WHISPER: significantly outperforms previous works
  - Larger transactions ➔ advent speculation opportunities



Evaluation – Persist-Path Latency

- Persist-path operations are mostly out of critical paths
- Only at the end of TXs, the persist-path must be drained



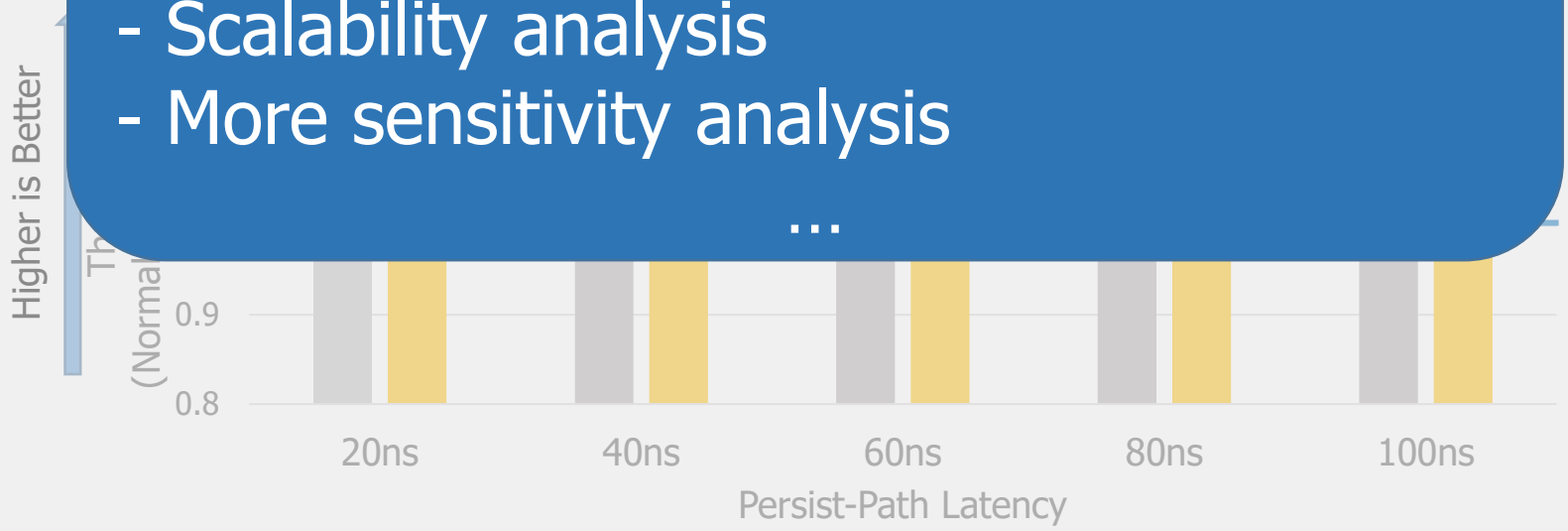
Evaluation – Persist-Path Latency

- Persist-path operations are mostly out of critical paths
- Only at the end of TXs, the persist-path must be drained

More in the paper:

- Speculation buffer in the PM controller
- Runtime & OS Support for PMEM-Spec
- Scalability analysis
- More sensitivity analysis

...

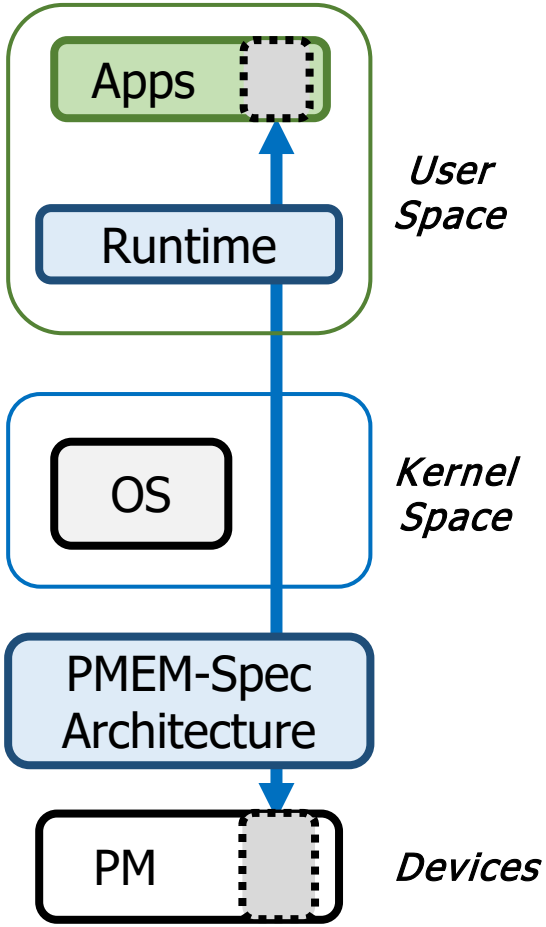


Conclusion: Persistent Memory Speculation

- HW (speculation) / SW (recovery) codesign for persist-order

- With separated load/store paths to PM, Misspeculation is *extremely* rare

- Leading to high performance strict persistency outperforming relaxed persistency



PMEM-Spec: Persistent Memory Speculation

(Strict Persistency Can Trump Relaxed Persistency)

Jungi Jeong and Changhee Jung
Purdue University

Session 6A: Hardware for Crash Consistency
NVMW 2021

PURDUE
UNIVERSITY

Department of Computer Science