SkyByte: Architecting an Efficient Memory-Semantic CXL-based SSD with OS and Hardware Co-design

Haoyang Zhang*, Yuqi Xue*, Yirui Eric Zhou, Shaobo Li, Jian Huang

Systems Platform Research Group

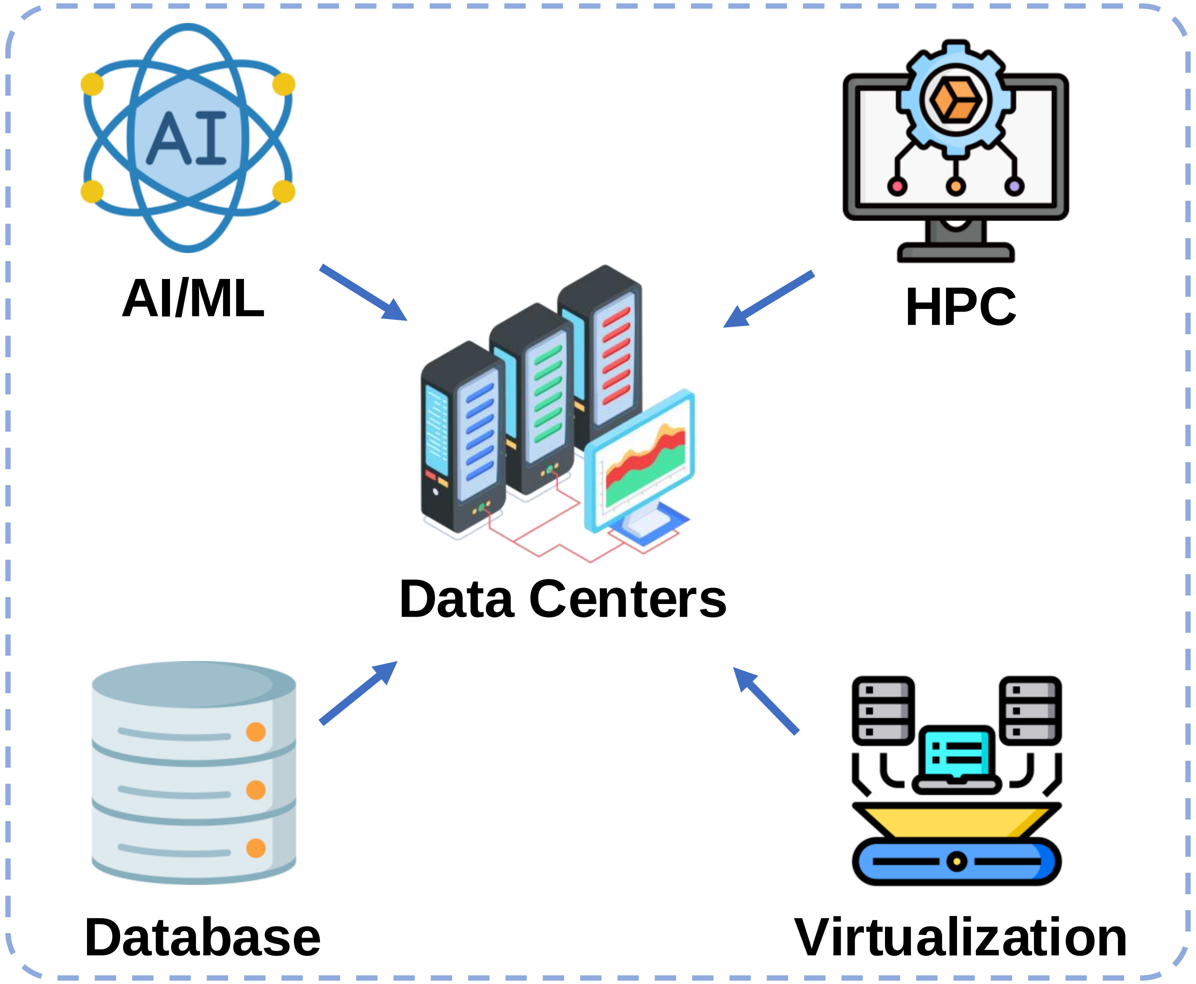
*Co-primary authors.

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

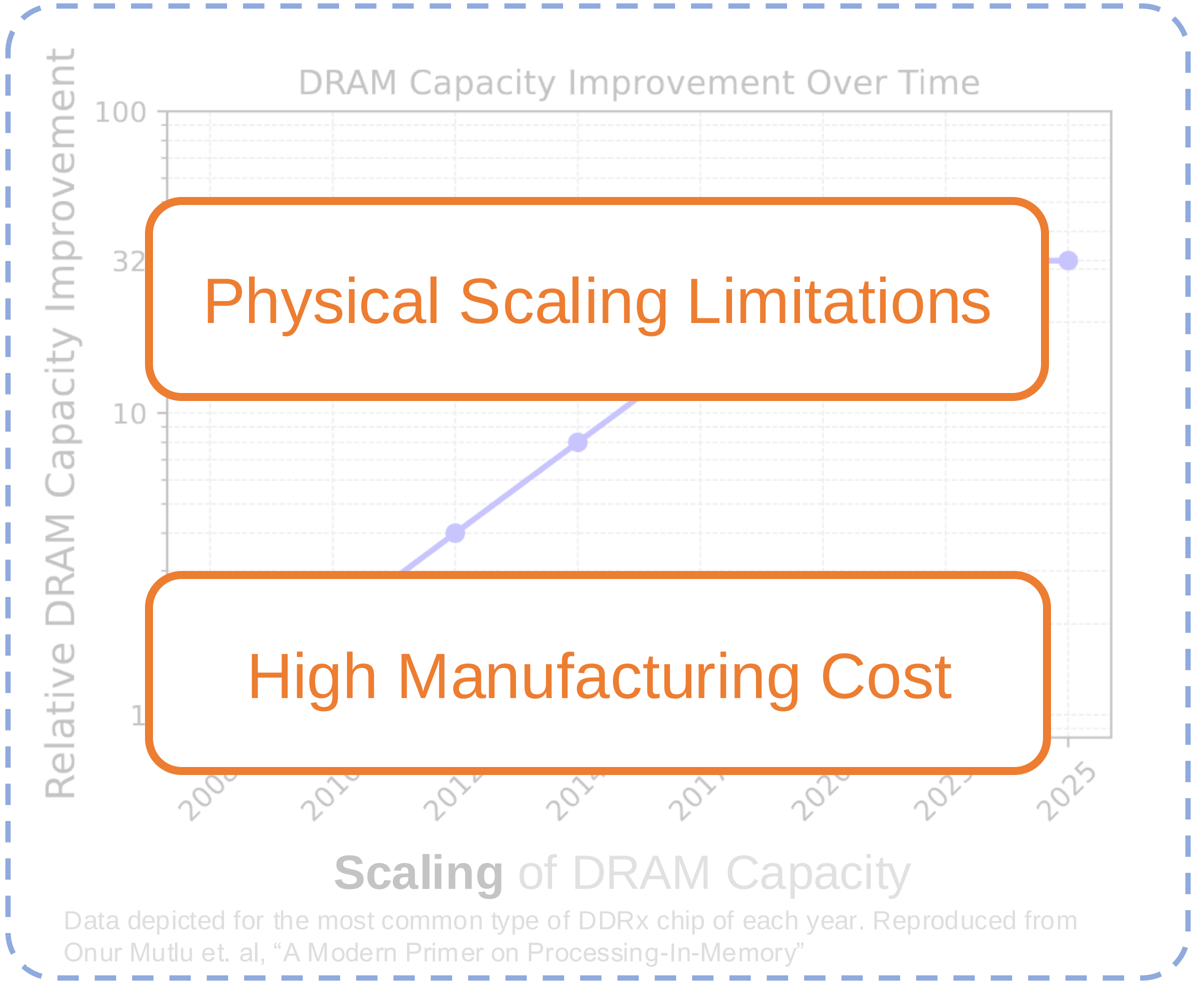
Artifact Badges

Open Research Projects
Research Objects Reviewed
Results Reproduced

The Growing Need for Memory Expansion in Data Centers



Increasing Memory Demands



Scaling of DRAM Capacity

Data depicted for the most common type of DDRx chip of each year. Reproduced from Onur Mutlu et. al, “A Modern Primer on Processing-In-Memory”

DRAM Capacity Scales Slowly

CXL-based SSDs: A Promising Way to Expand Memory Capacity

A variety of CXL-SSD prototypes have been developed

CXL-Flash

ch

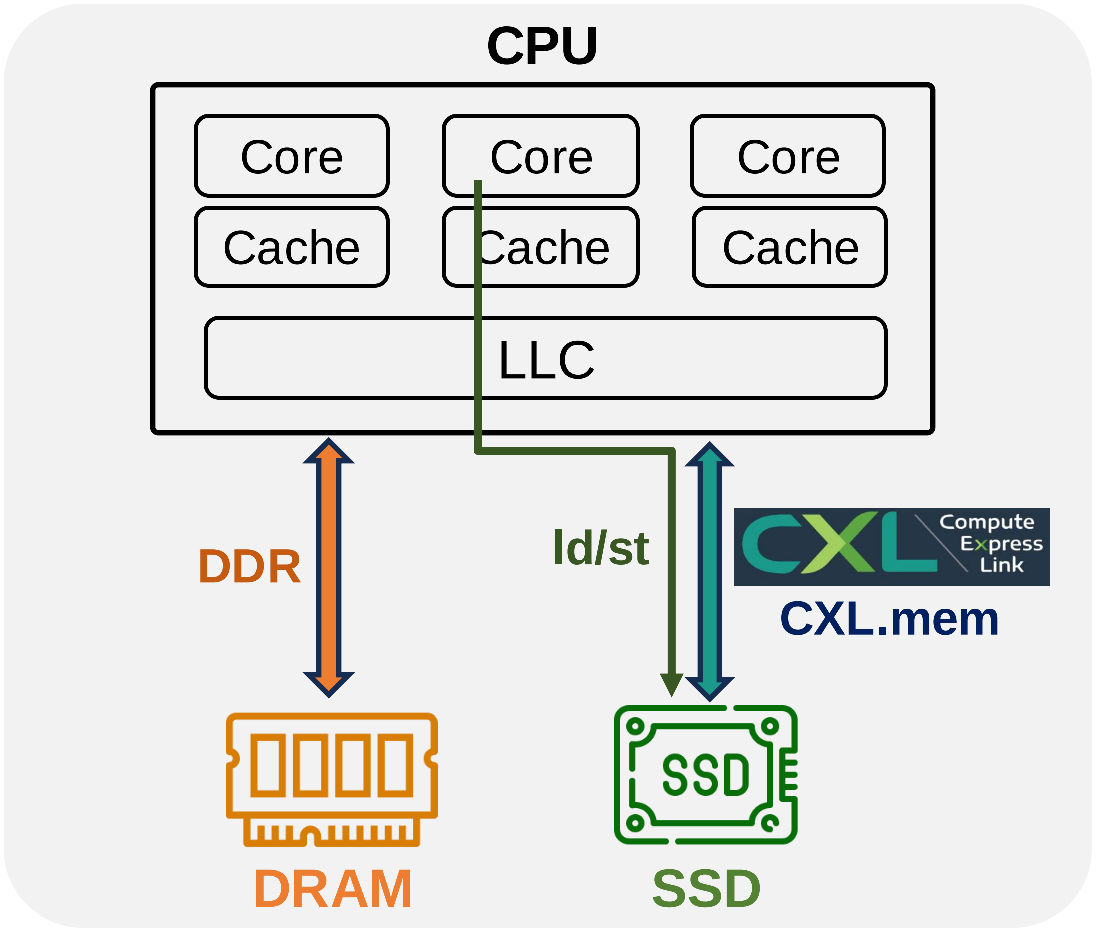
CXL-based SSDs

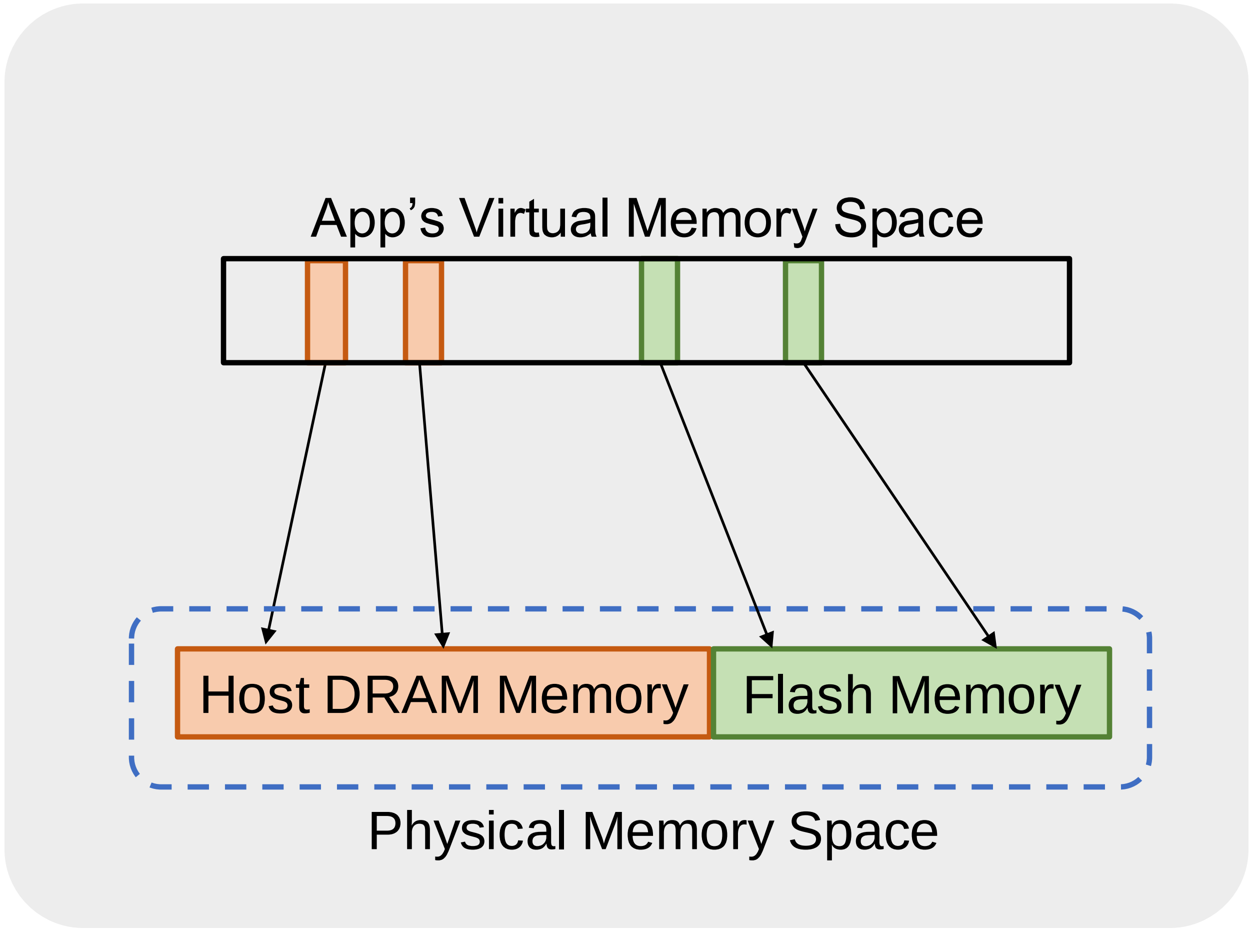
Larger Capacity

Lower Cost

Cacheable and Byte-addressable Access by CPU

System Architecture of CXL-based SSDs

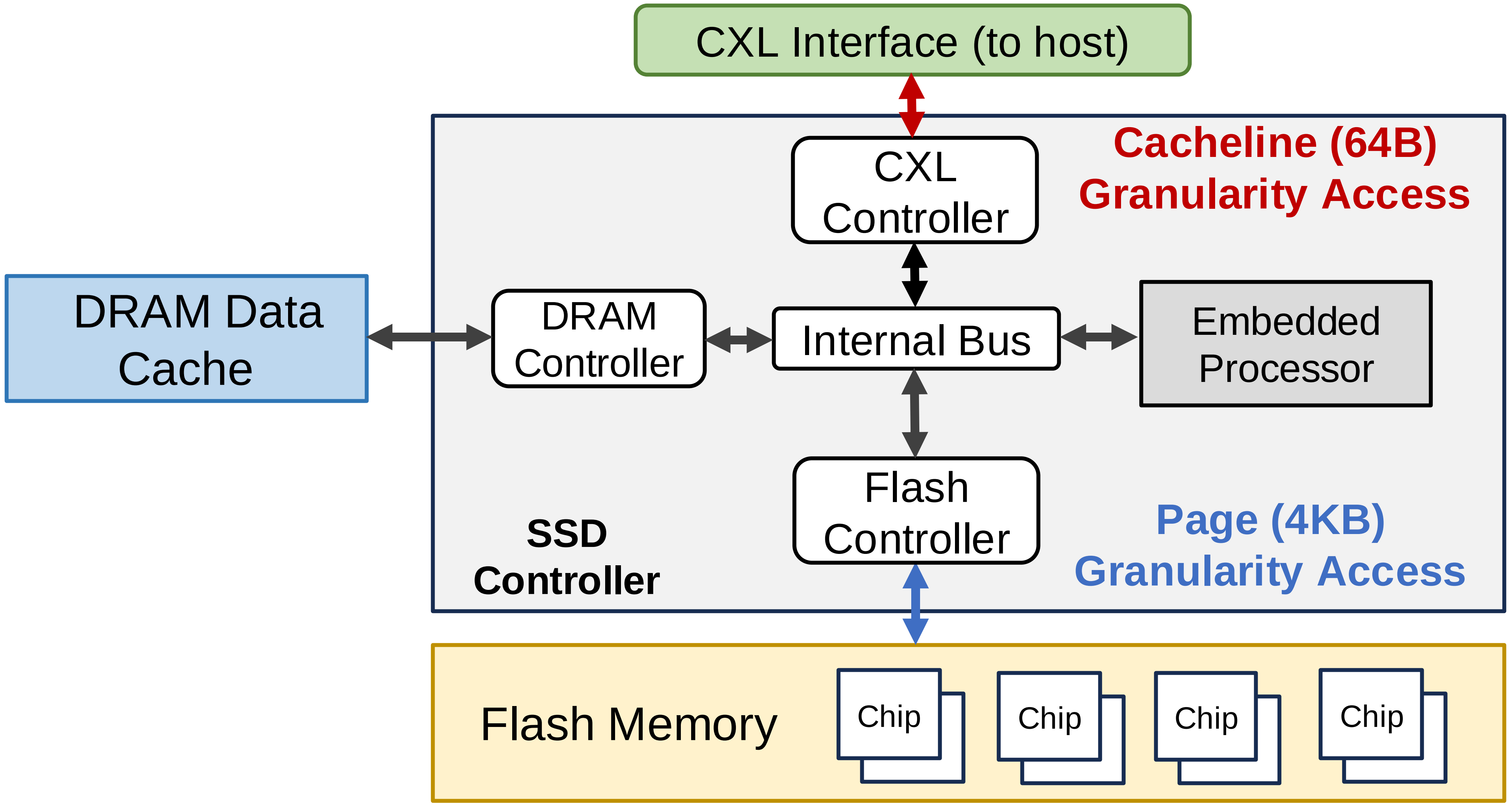




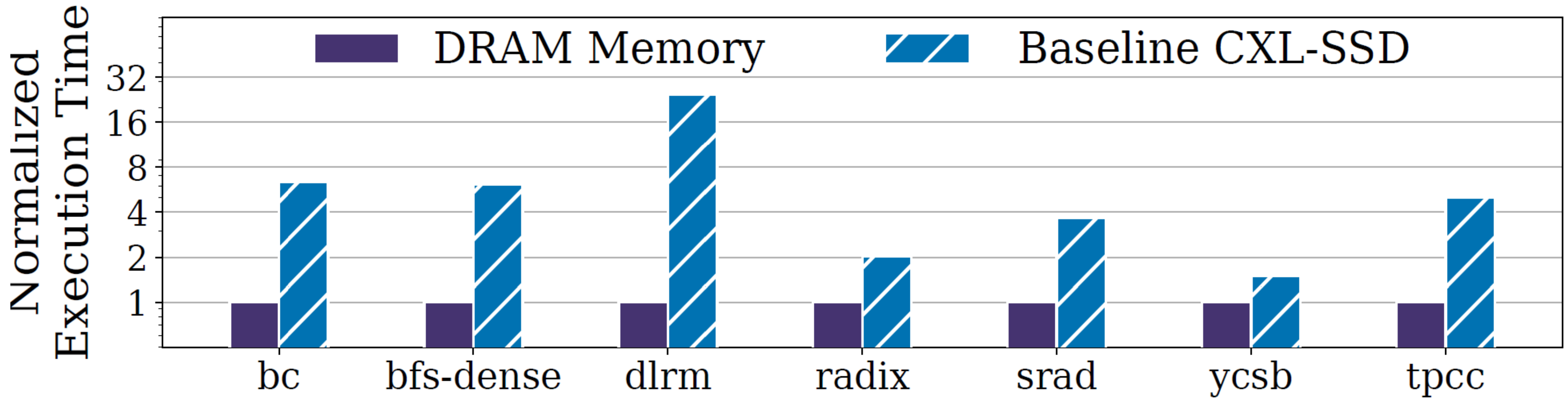
Directly access via load/store instructions

Mapped as Part of System's Physical Memory

Internal Architecture of CXL-SSD



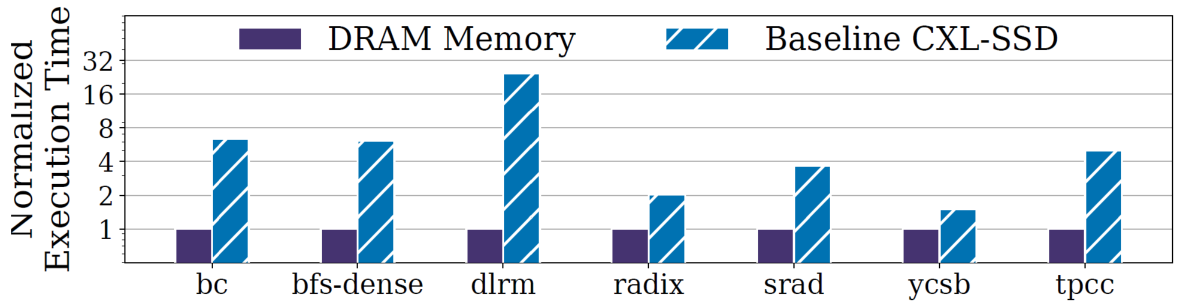
Current CXL-SSDs Face Significant Performance Challenges



End-to-end Execution Time of running different workloads using DRAM v.s. CXL-SSD

CXL-SSDs suffer from 1.5-31.4× worse performance than DRAM!

Current CXL-SSDs Face Significant Performance Challenges



End-to-end Execution Time of running different workloads using DRAM v.s. CXL-SSD

CXL-SSDs suffer from 1.5-31.4× worse performance than DRAM!

Long Flash Access Latency

Excessive Processor Pipeline Stalls

Access Granularity Mismatch

Long Tail Latencies Cause Excessive Processor Pipeline Stalls

Memory Type	DRAM	Flash
Latency	~50 ns	> 3 µs

Z-NAND SSD

- **Read: 3 µs**
- **Program (write): 80 µs**
- **Erase: 1000 µs**

Flash access latency is order of magnitude higher than DRAM latency

Long Tail Latencies Cause Excessive Processor Pipeline Stalls

Using DRAM as Memory

DRAM Access: **~50ns**

(CPU Cache Miss)

Utilize ILP: Run other independent instructions

Can hide DRAM Latency

Using CXL-SSD as Memory

Flash Access: >3µs

(CPU Cache Miss + SSD DRAM Miss)

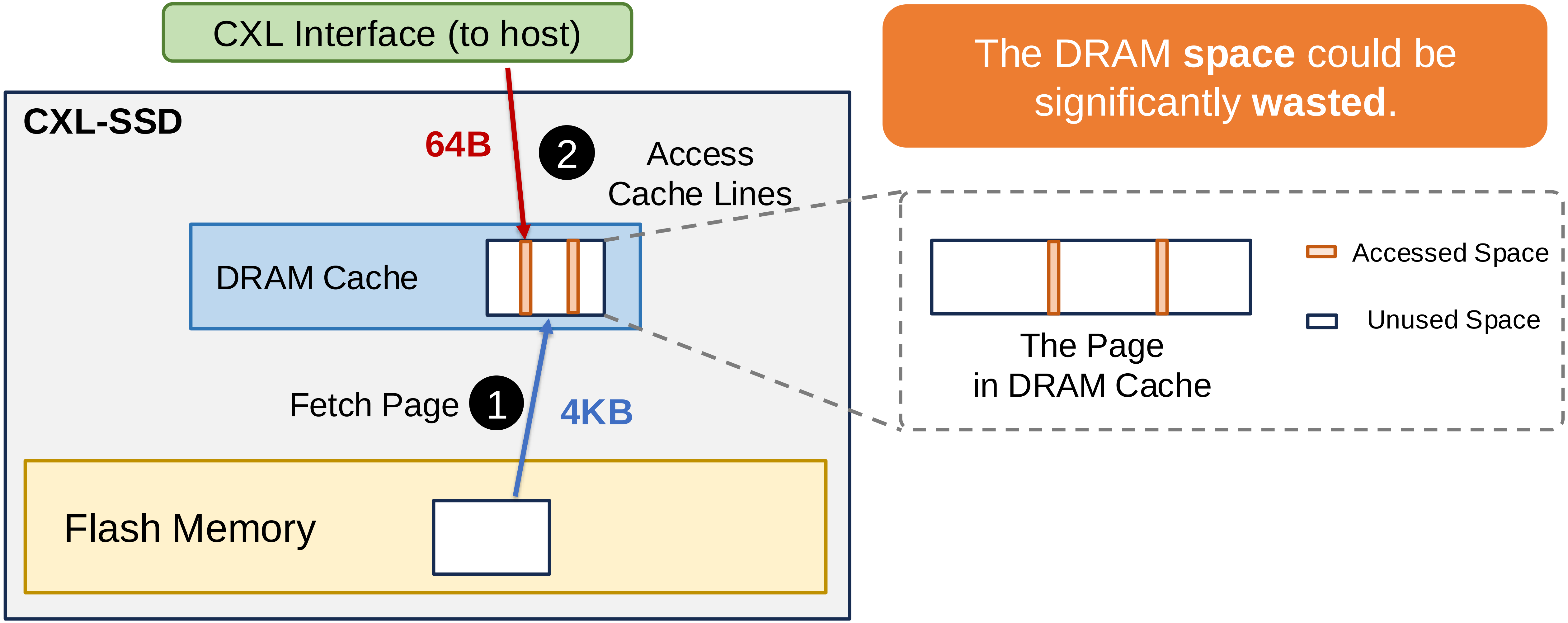
Not Enough ILP /
Not Enough Hardware Resource!

CPU Core Stall!

Low CPU resource utilization
Low SSD bandwidth utilization

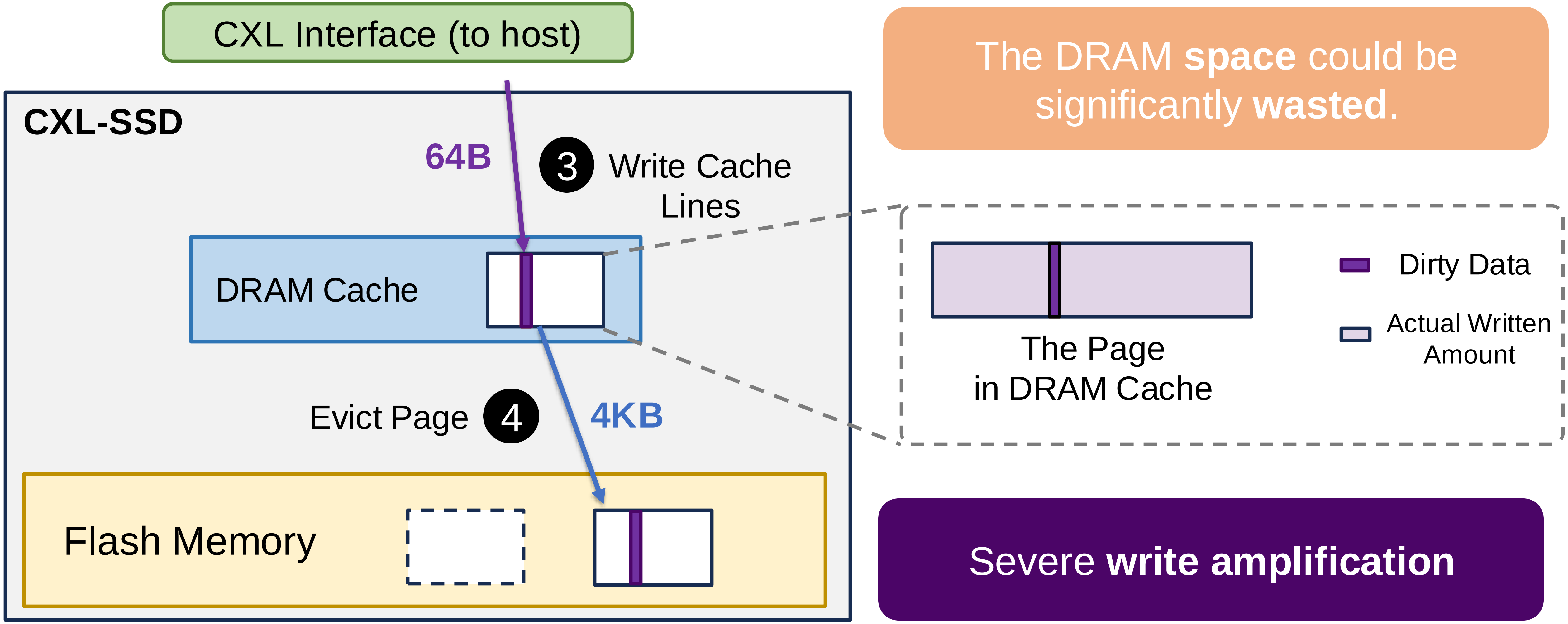
Modern processor techniques (e.g., OoO) are less effective to hide long flash latency

Access Granularity Mismatch Causes Inefficiencies



The DRAM **space** could be significantly **wasted**.

Access Granularity Mismatch Causes Inefficiencies

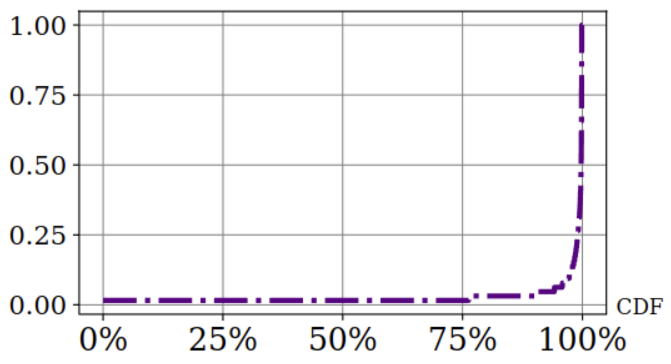


The DRAM **space** could be significantly **wasted**.

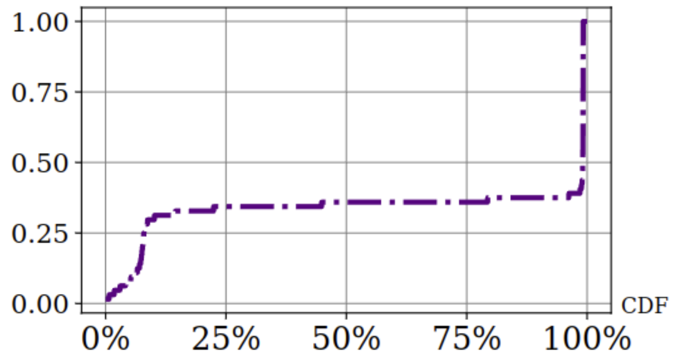
Severe **write amplification**

Many Workloads Exhibit Poor Locality Within Each Page

(a) Ratio of cache line accessed in fetched pages.

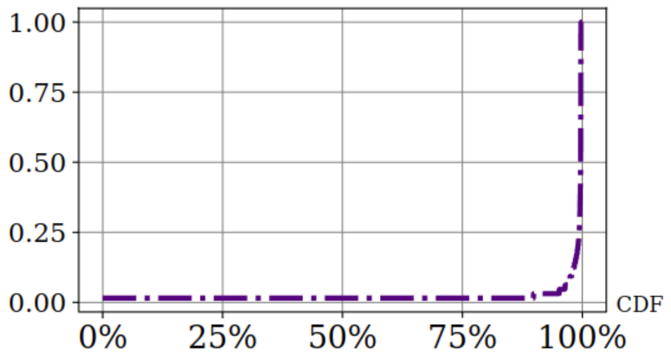


(a) bc

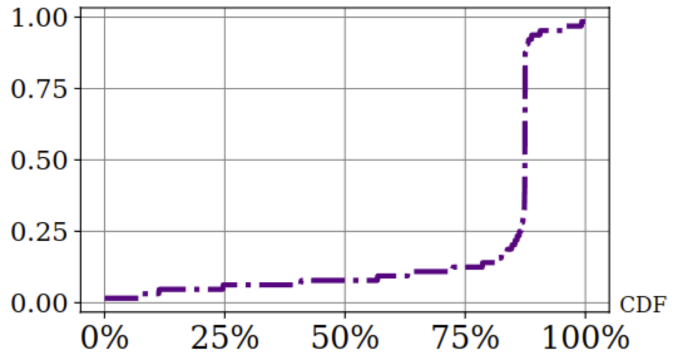


(b) ycsb

(b) Ratio of cache line written in evicted pages.



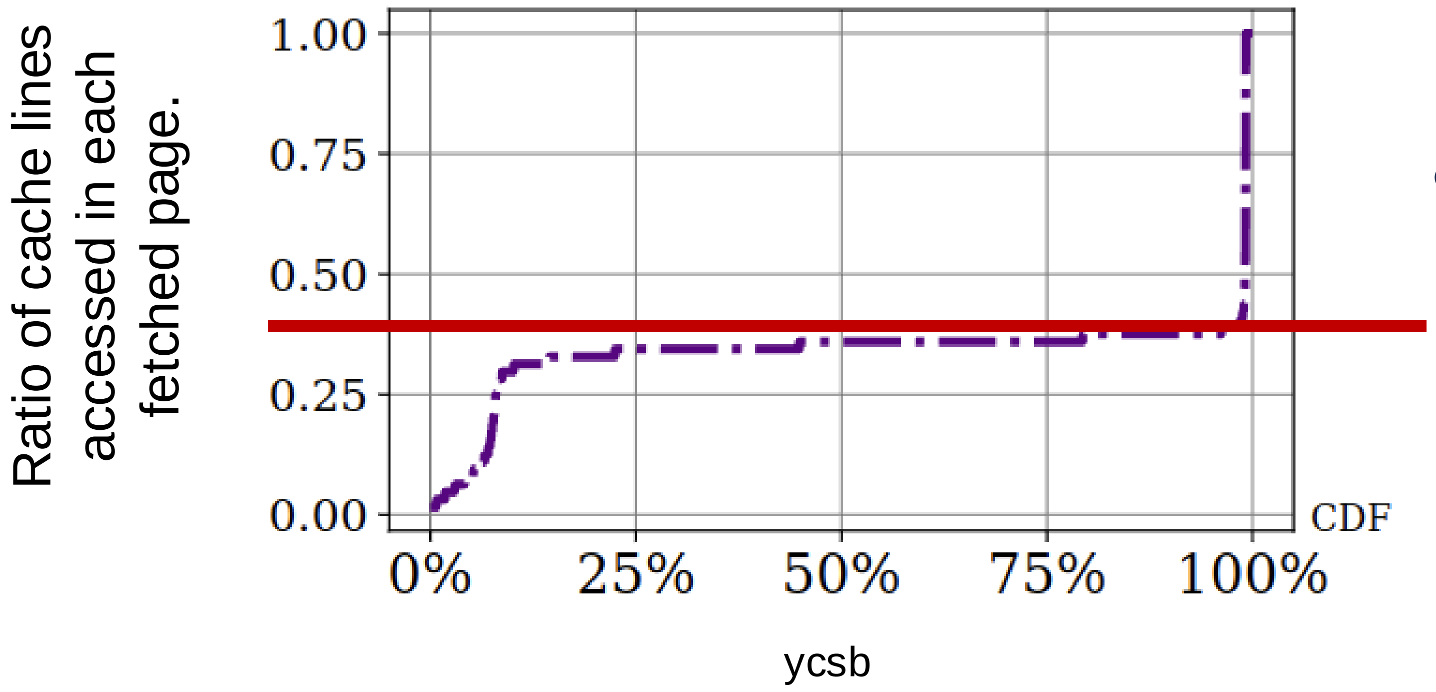
(b) bc



(b) ycsb

Locality Distribution (CDF) of pages read from/flushed to flash memory.
(**Lower line means worse locality**)

Many Workloads Exhibit Poor Locality Within Each Page



In >90% pages, only <40% of the cachelines are accessed!

Locality Distribution (CDF) of pages read from flash memory. (**Lower line means worse locality**)

Many workloads suffer from severe DRAM space waste and write amplifications

SkyByte: A Holistic Approach to Address CXL-SSD Challenges

Long Flash Access Latency

Excessive Processor Pipeline Stalls

Access Granularity Mismatch

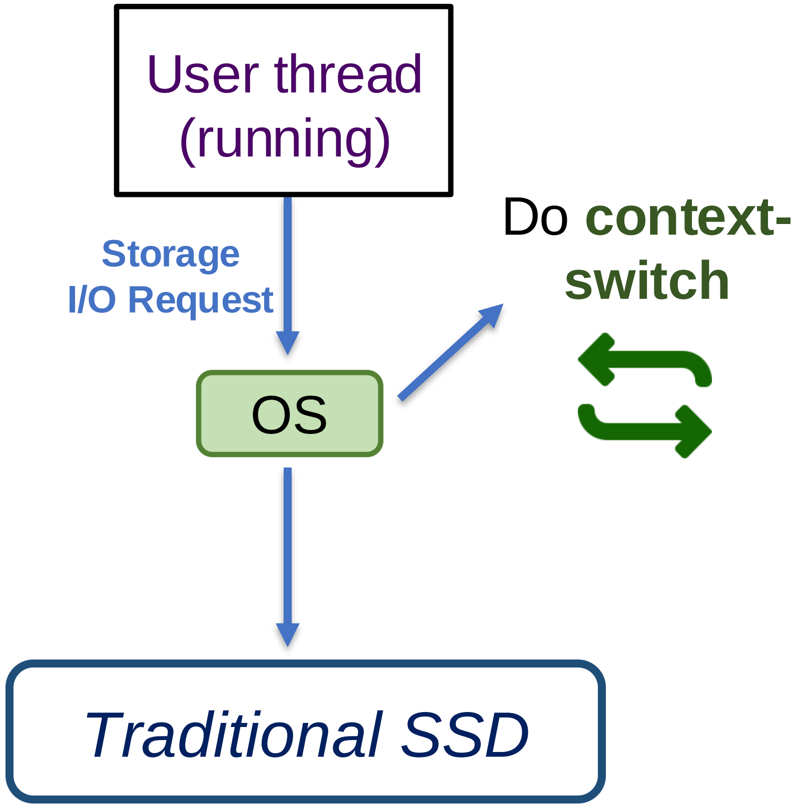
SkyByte

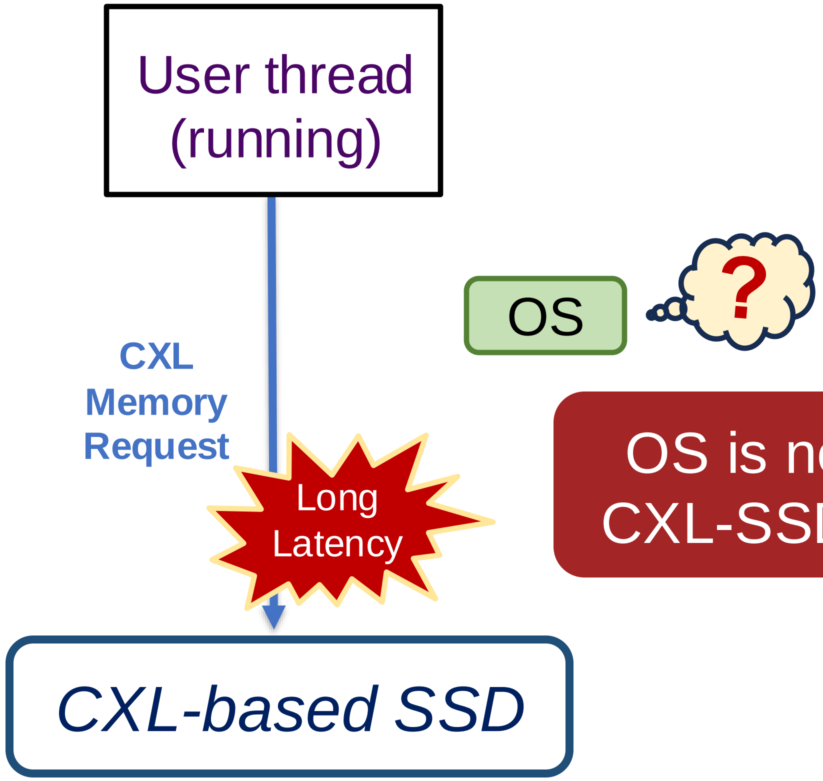
Hide the Flash Access Latency with **Coordinated Context Switches**

Bridge the Granularity Gap by **Rearchitecting** the SSD **DRAM Cache**

Expand SSD DRAM Cache with Host DRAM via **Page Migrations**

Can We Hide Long Flash Latency with OS-Based Context Switches?

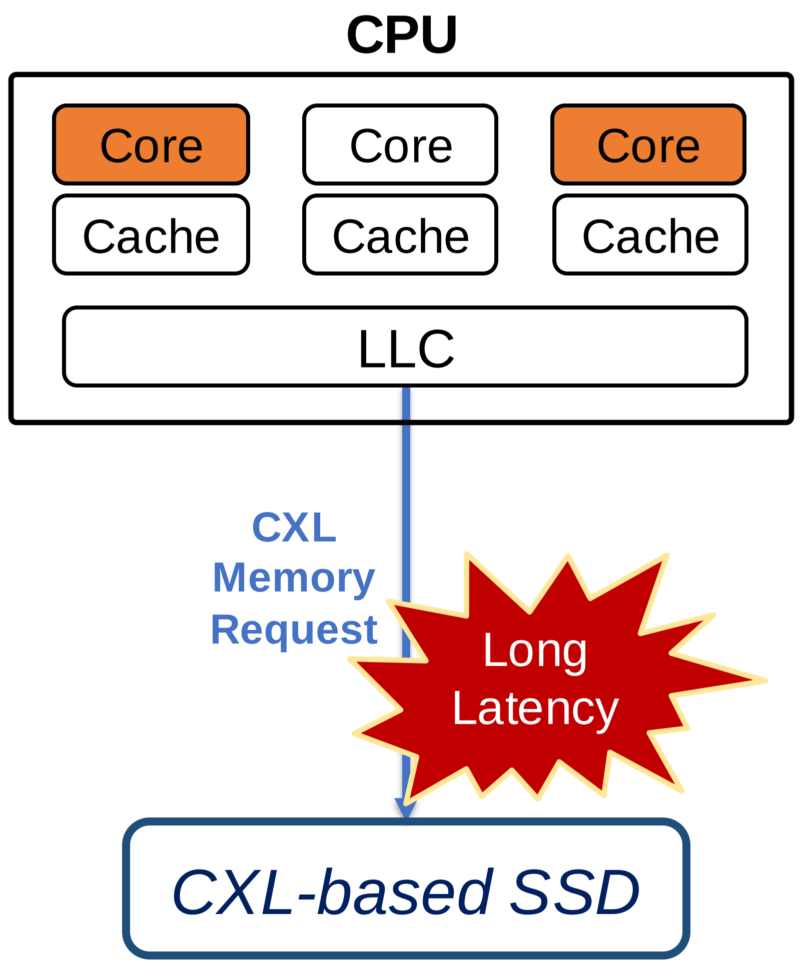




OS is not aware of CXL-SSD accesses!

OS-based context switch opportunity is missing in CXL-SSDs!

Is It Possible to Build a New Context Switch Mechanism for CXL-SSDs?



When?	Only on a Long Flash Latency
Which?	Only Cores Waiting for the Data
How?	OS Should Be Notified

Is It Possible to Build Context Switch Mechanism for CXL-SSDs?

When?	Only on a Long Flash Latency
Which?	Only Cores Waiting for the Data
How?	OS Should Be Notified

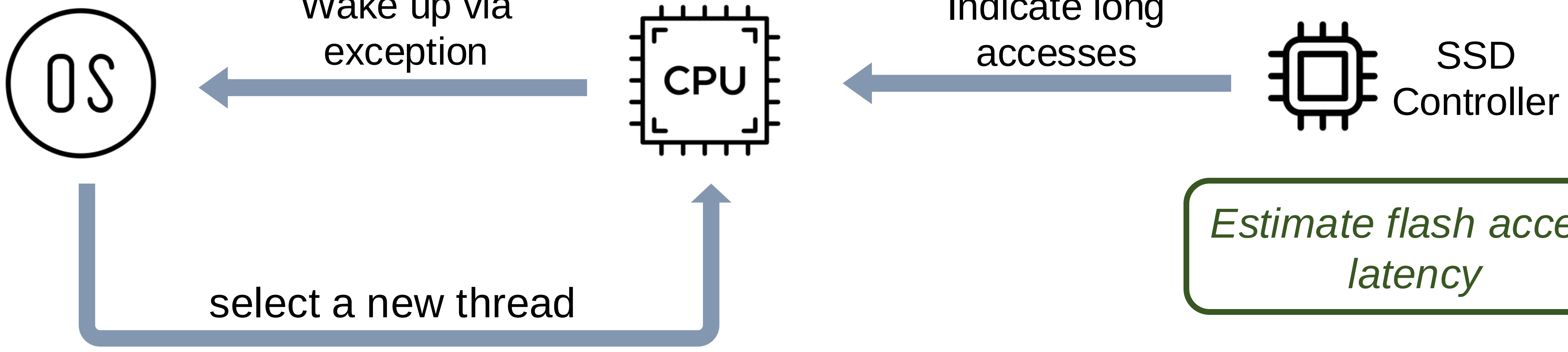
Not Aware of SSD Access Latency

Not Aware of CPU Microarchitectural Status

Cannot Directly Intercept CXL-SSD Accesses

None of them can by itself decide whether to trigger a context switch

Coordinated Context Switch Mechanism for CXL-SSDs

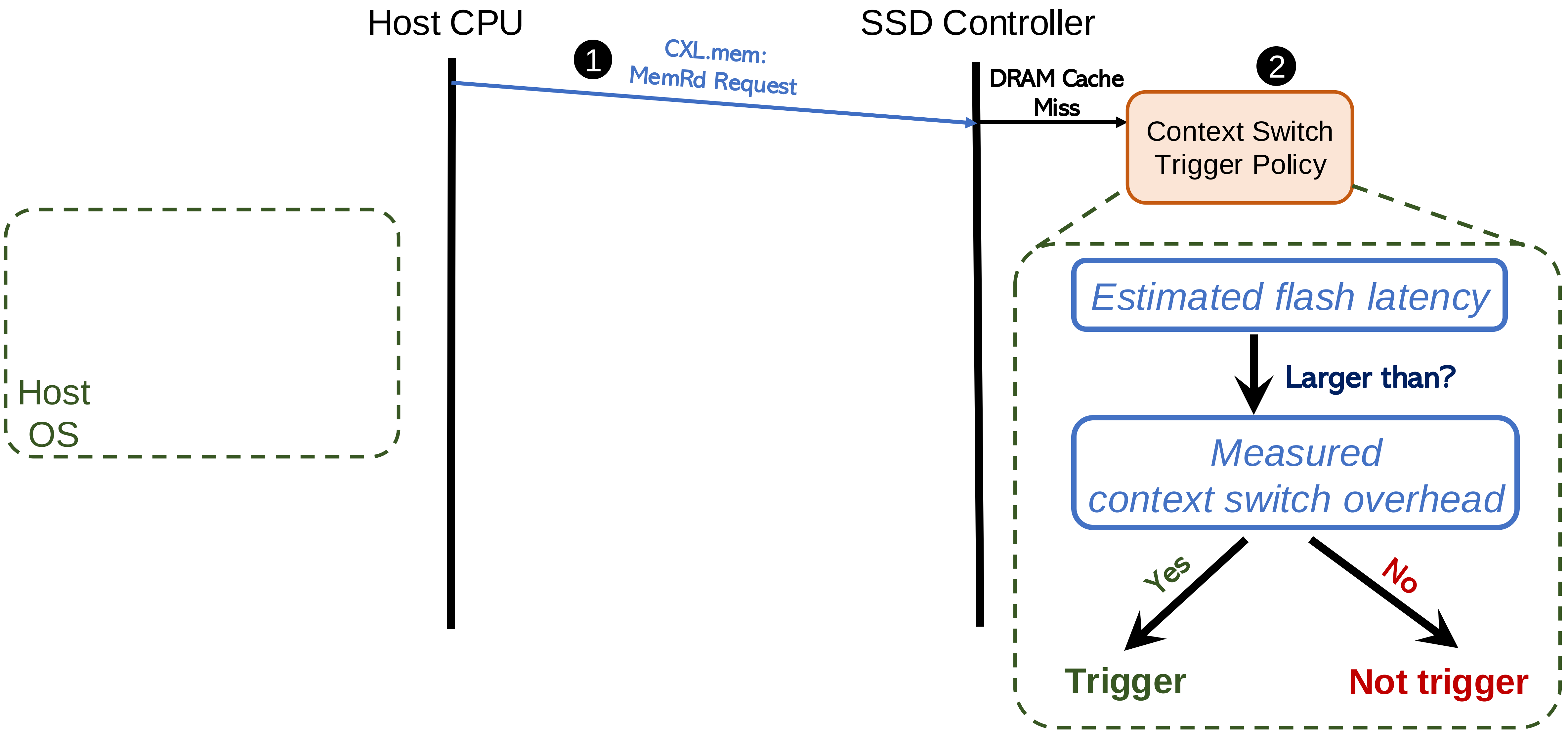


Decide whether to do a switch on specific cores

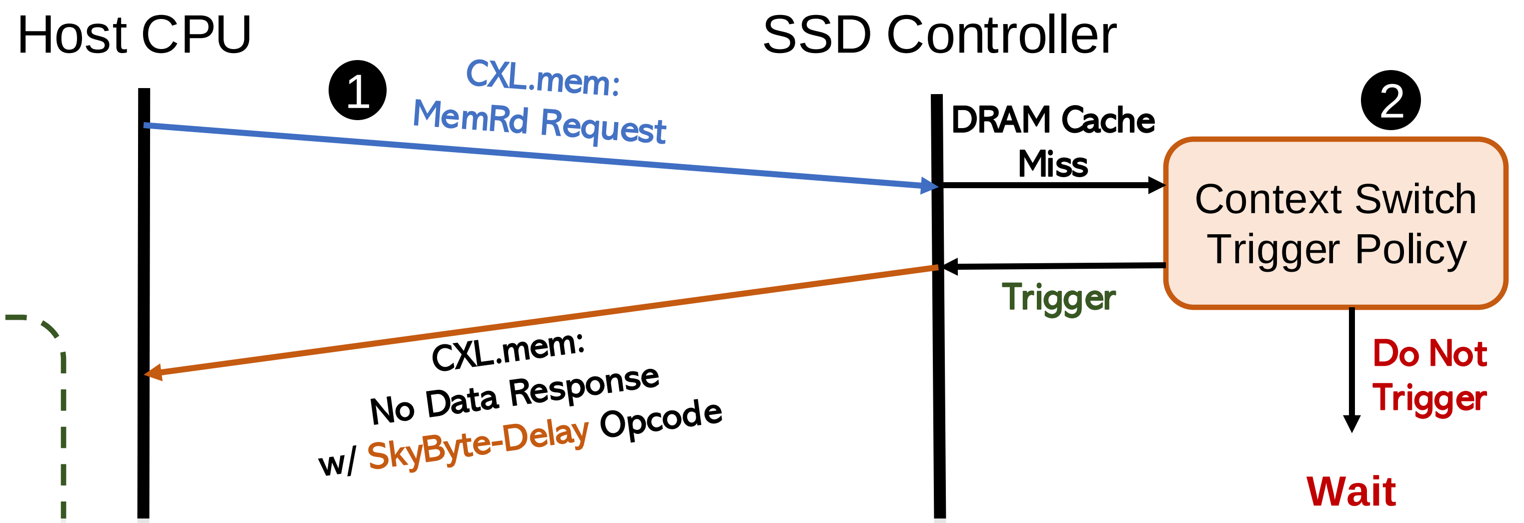
Estimate flash access latency

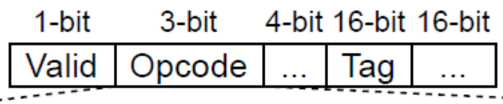
OS can wisely choose the next thread

Procedure of Coordinated Context Switch in SkyByte



Procedure of Coordinated Context Switch in SkyByte



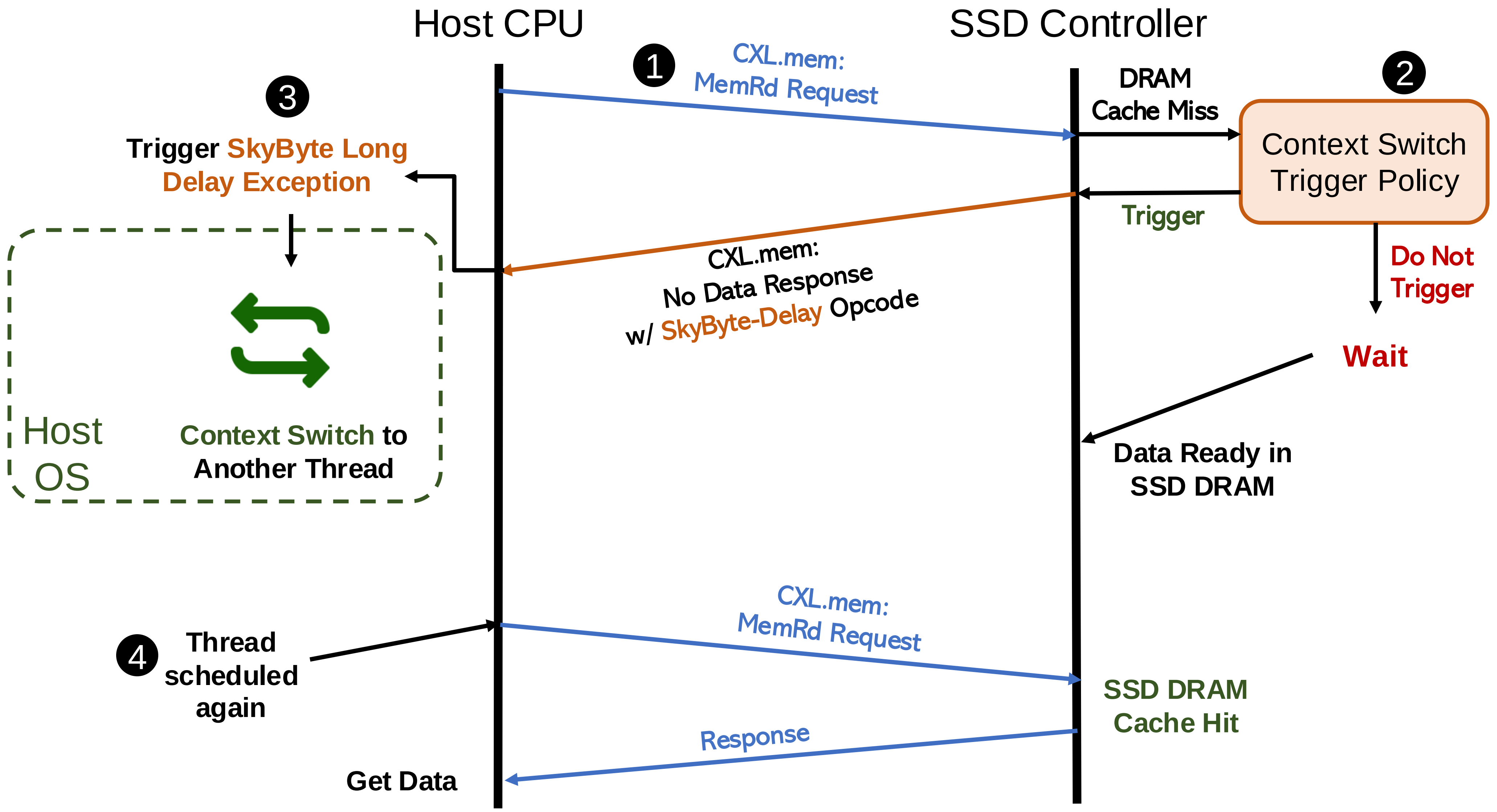


Opcode	Description	Encoding
Cmp	Completions for Writebacks, Reads, and Invalidates	000b
Cmp-S, Cmp-E, BI-ConflictAck	Cache coherence-related opcodes for CXL.cache	001b, 010b, 100b
SkyByte-Delay	Indication from the SSD to the Host for Long Access Delay	111b
Reserved	Other reserved opcodes	Others

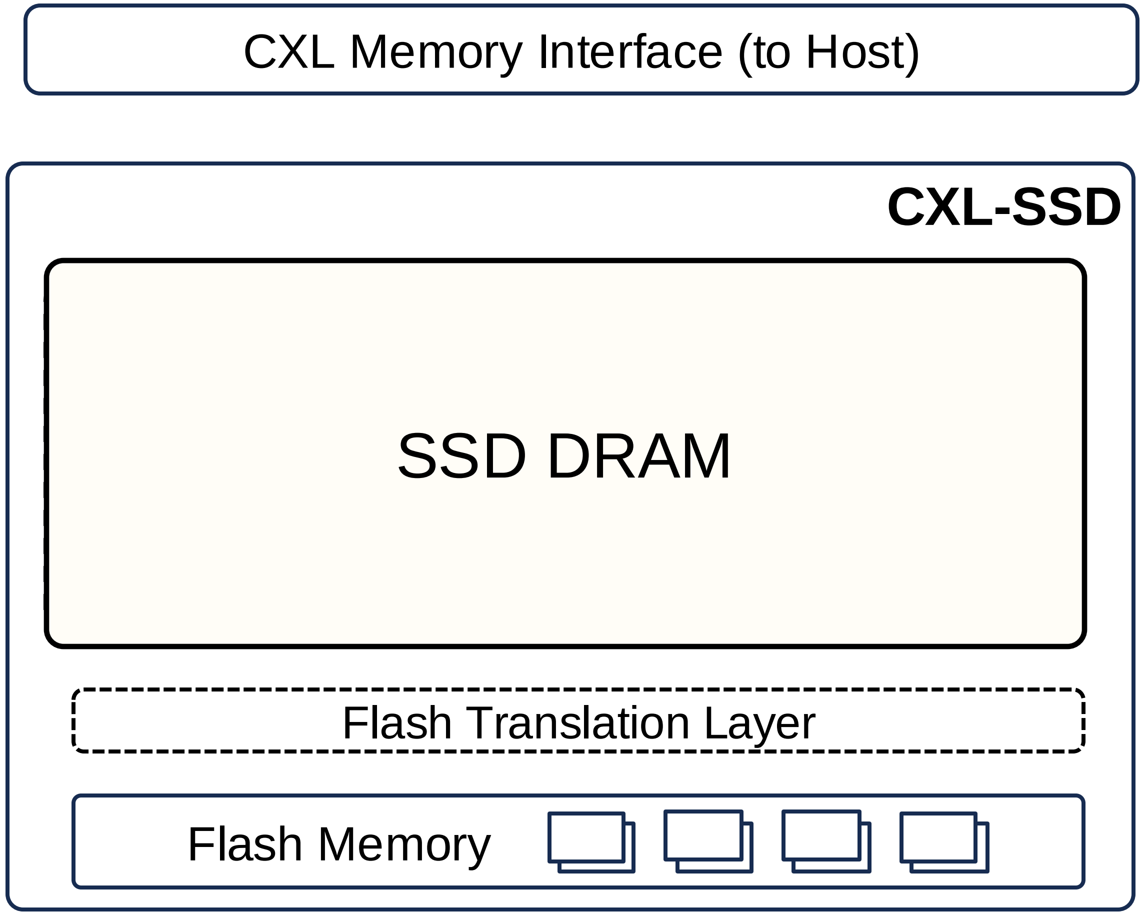
CXL.mem No Data Response (NDR) message format and opcode definitions

We introduced a new opcode using reserved encoding bits

Procedure of Coordinated Context Switch in SkyByte



Deploying A Cache-line Granular Write Log to Bridge the Granularity Gap

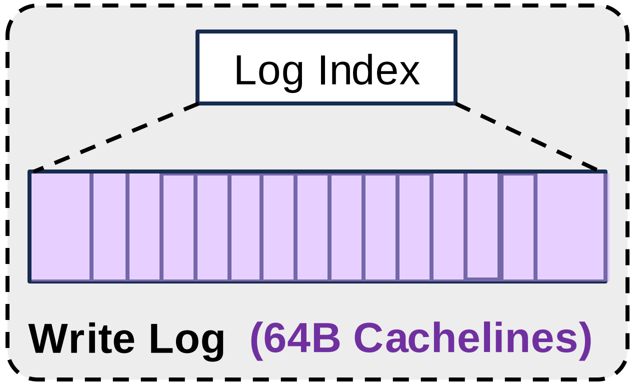


64B (Cacheline) Granularity Write Log

+

4KB (Page) Granularity Data Cache

Deploying A Cache-line Granular Write Log to Bridge the Granularity Gap



64B (Cacheline) Granularity Write Log

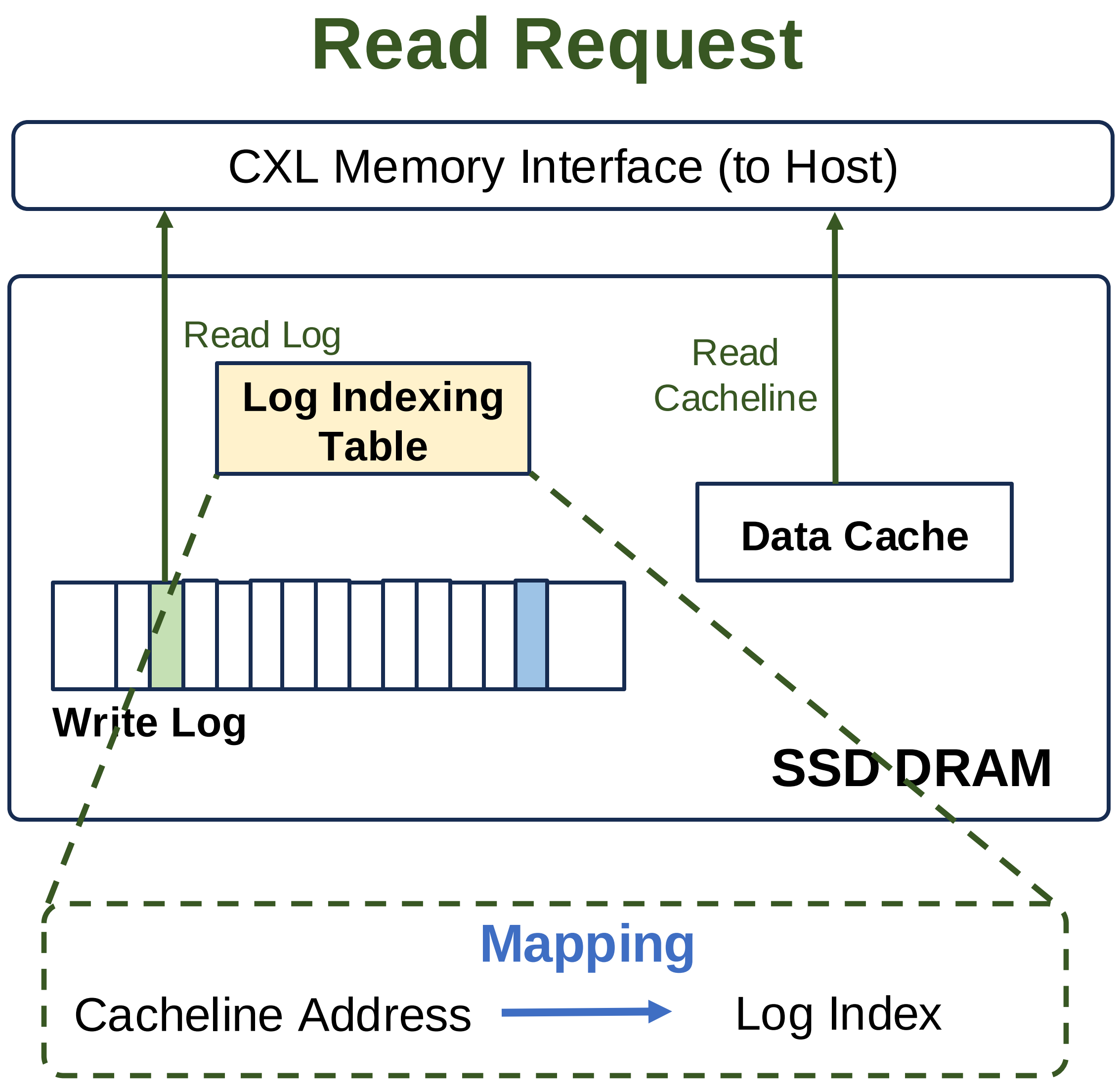
The DRAM **Space** Could Be Significantly **wasted**

Severe **Write Amplification**

Save DRAM Space with A Finer Granularity

Reduce Flash Write Traffic with A Larger Coalescing Window

Deploying an Indexing Table for Fast Log Lookup



Both write log and data cache can hold **newest version of data**

Lookups needed to access **both** data cache and log

Deploy an Indexing **Hash Table** to achieve fast log lookup

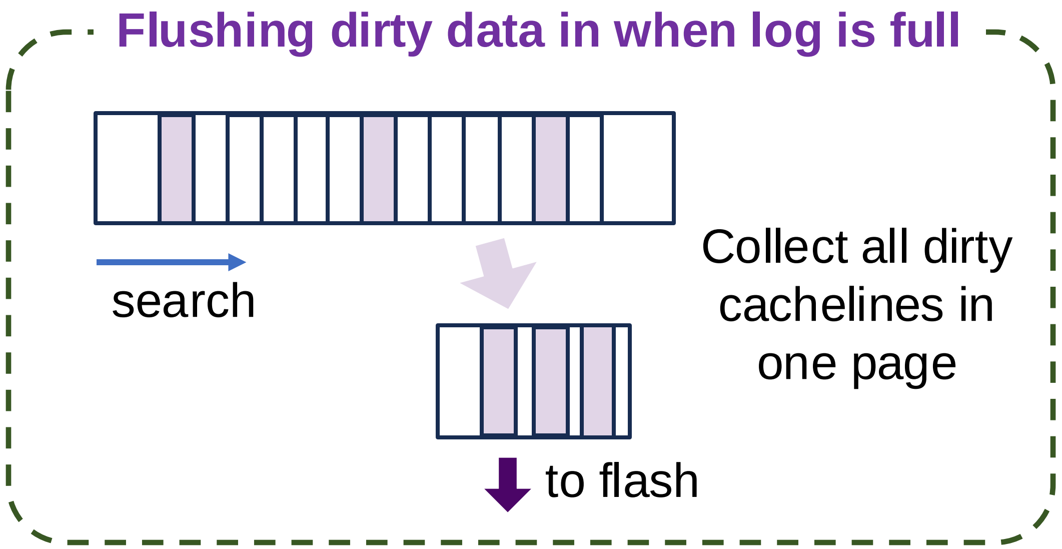
Supporting Efficient Log Indexing with a Two-level Hash Table

Plain Hash Table

Cacheline Address	Log Index

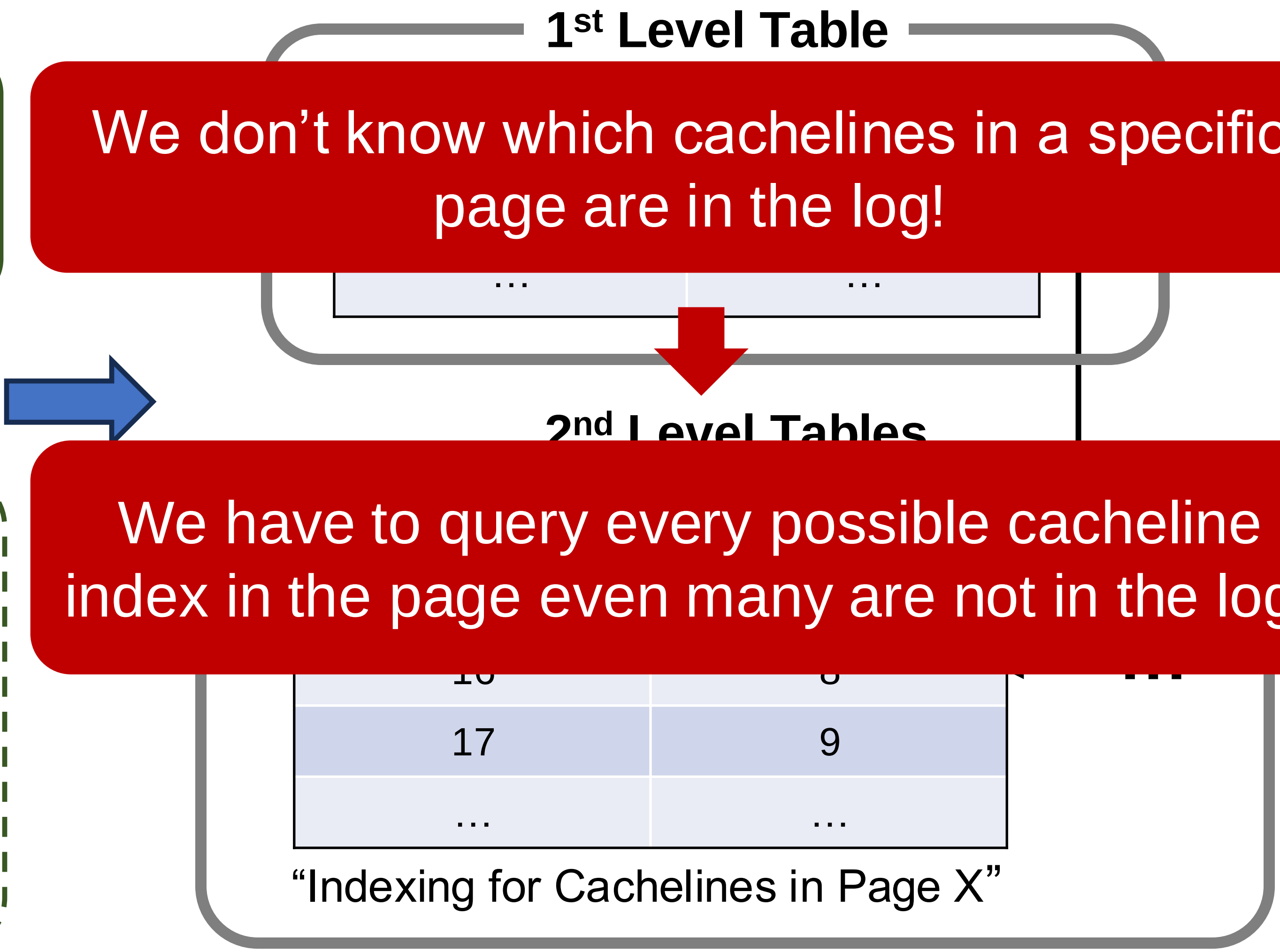
2nd level table will have a small size if only a few cachelines of the page are in the log.

Inefficient!



Flushing dirty data in when log is full
search
Collect all dirty cachelines in one page
to flash

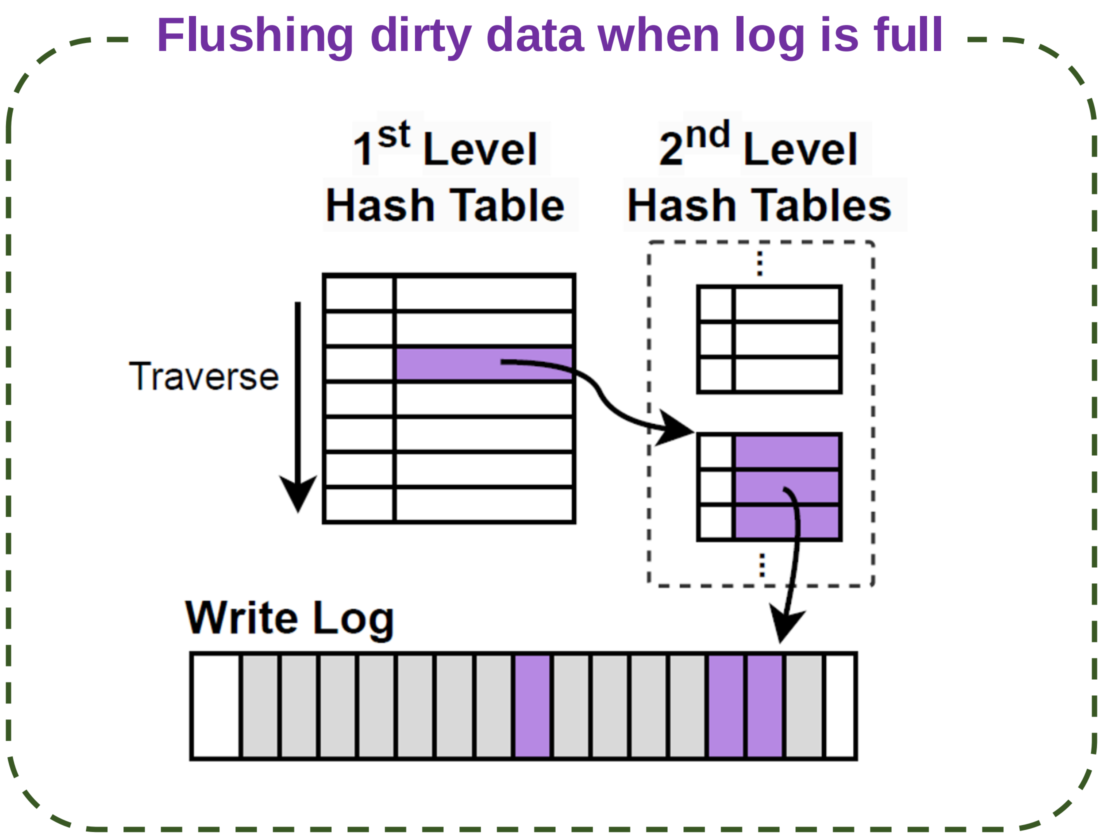
Two-Level Hash Table



We don't know which cachelines in a specific page are in the log!

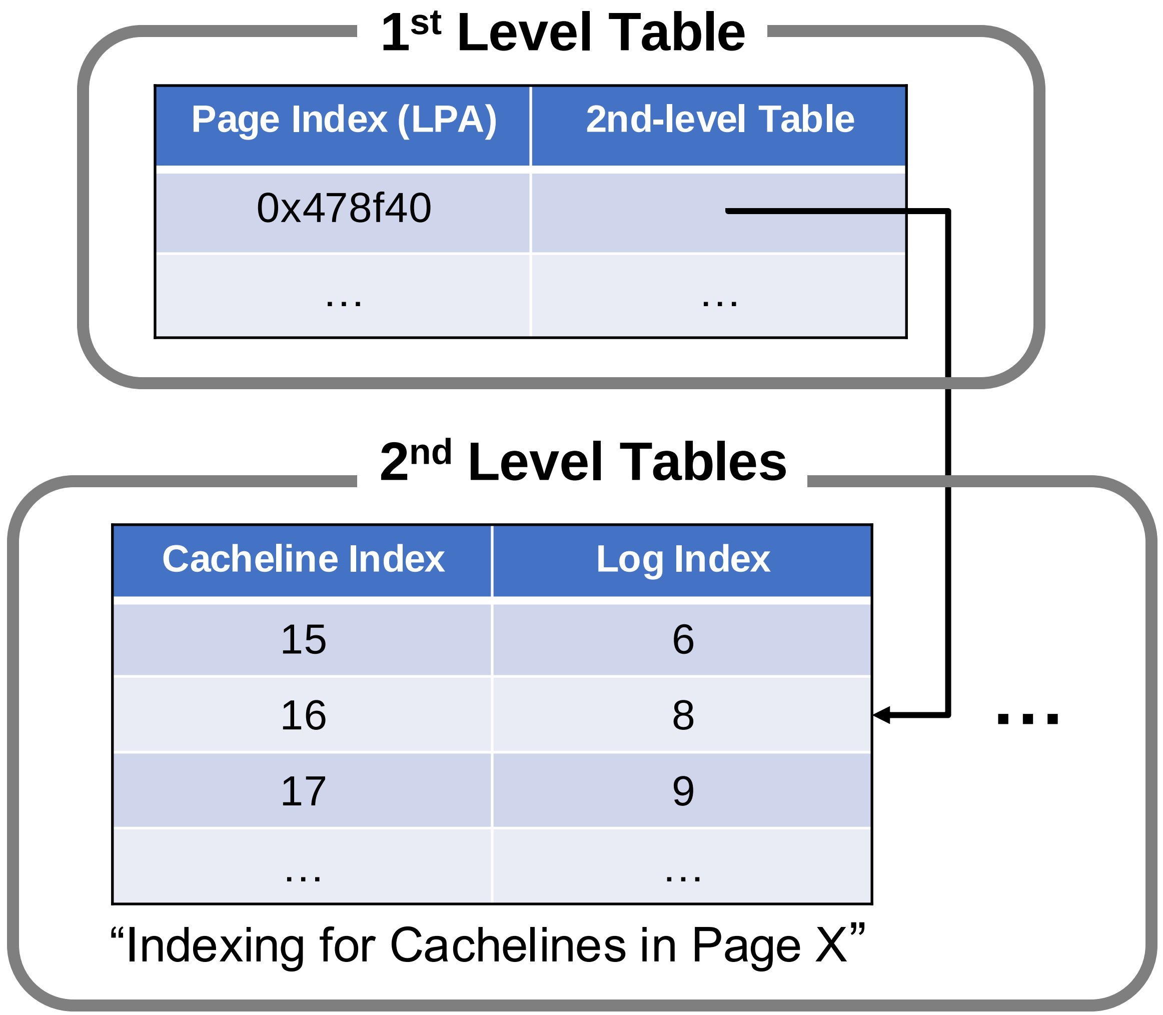
We have to query every possible cacheline index in the page even many are not in the log!

Supporting Efficient Log Indexing with a Two-level Hash Table



Traverse small 2nd level tables to look up all dirty data in each page

Two-Level Hash Table



1st Level Table

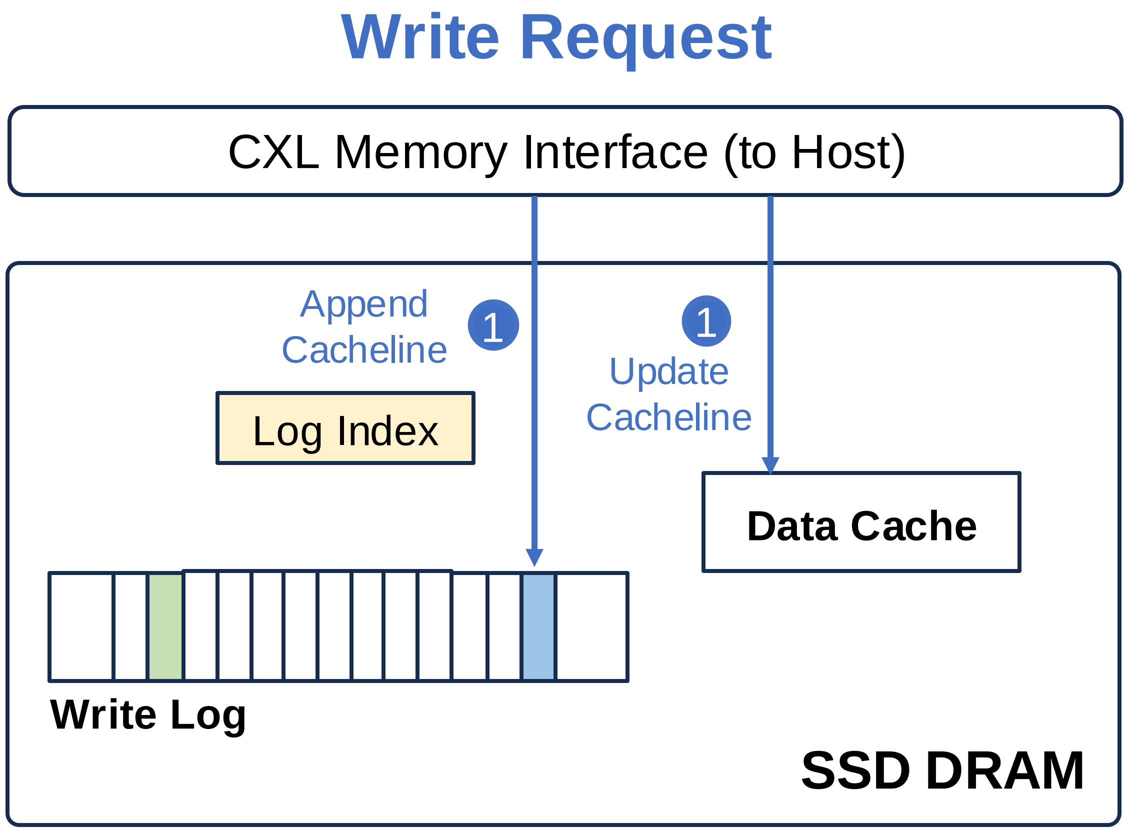
Page Index (LPA)	2nd-level Table
0x478f40	
…	…

2nd Level Tables

Cacheline Index	Log Index
15	6
16	8
17	9
…	…

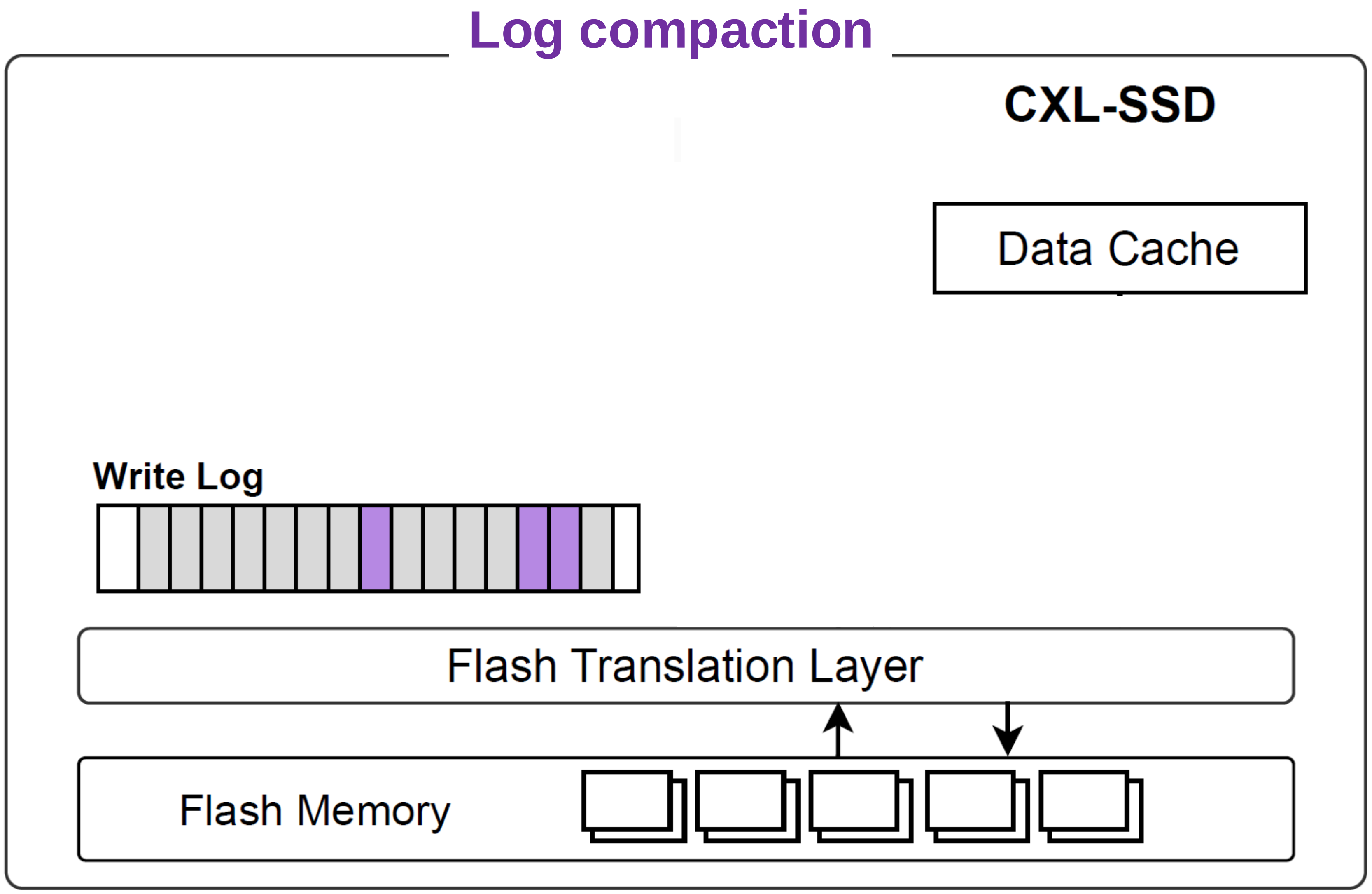
"Indexing for Cachelines in Page X"

Maintaining Data Consistency with Simultaneous Update



Update both write log
and data cache (If hit)

Maintaining Data Consistency with Simultaneous Flushing

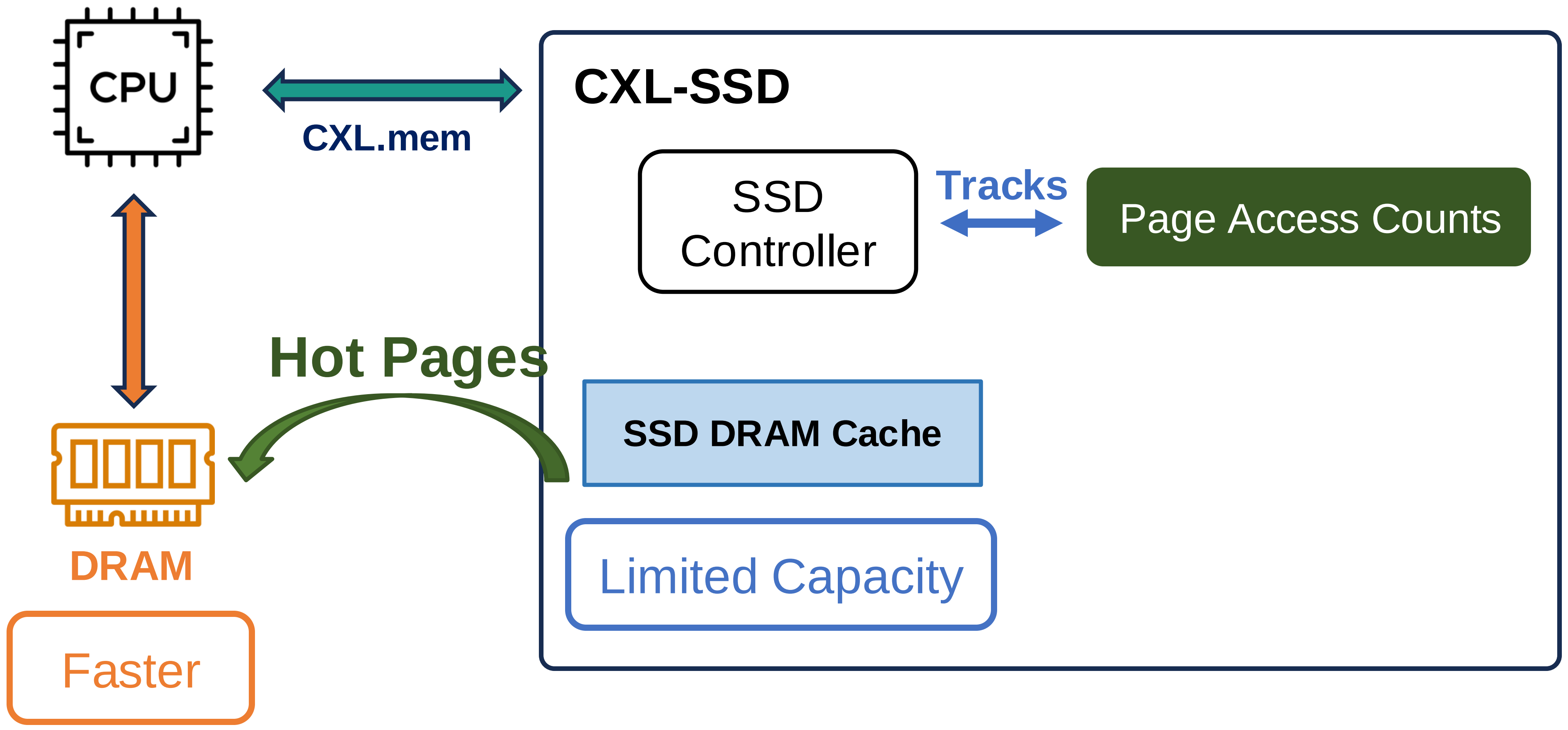


When a page is going to be flushed from log:

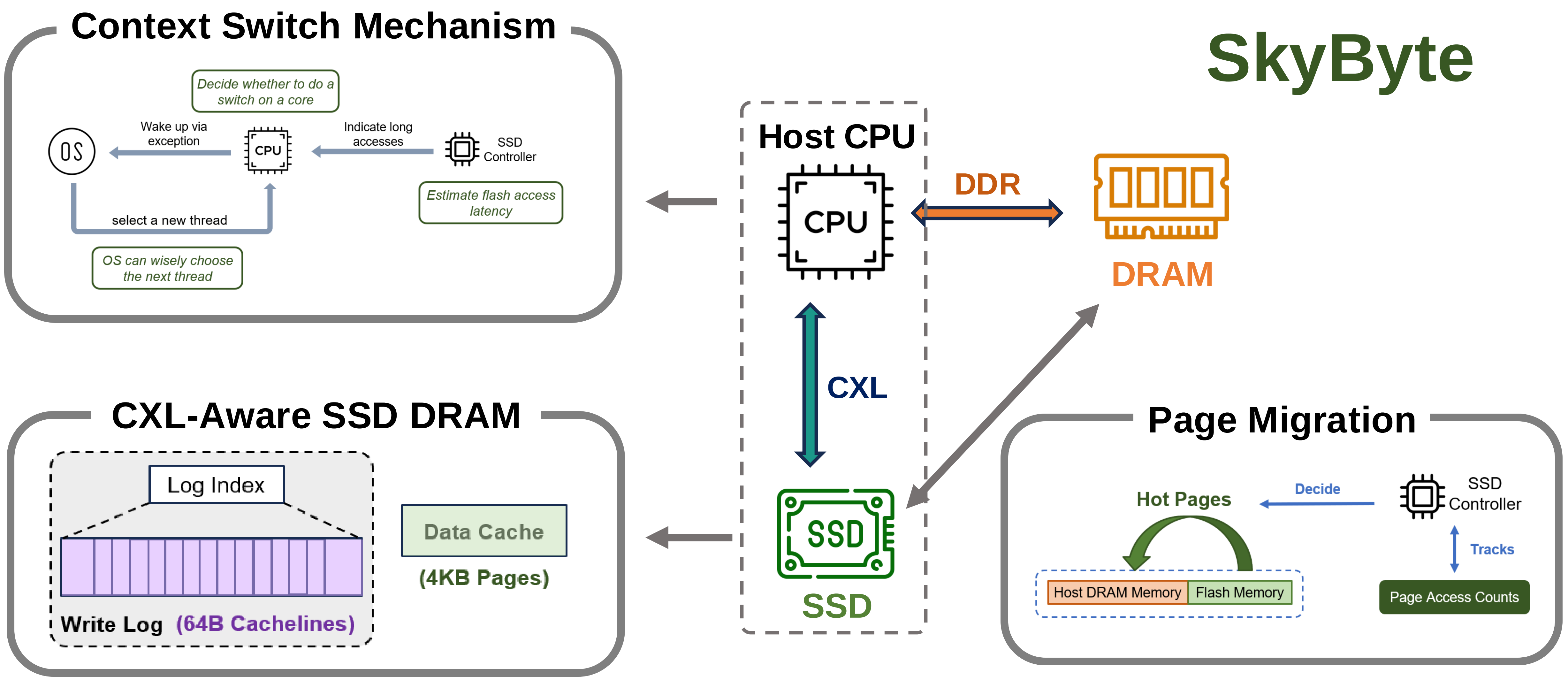
Data Cache Hit: *Flush the page from data cache*

Data Cache Miss: *Fetch from flash, merge data, flush back*

Expand SSD DRAM Cache with Adaptive Page Migration



Put It All Together



SkyByte Evaluation

Implementation

- **Trace Collection:** Intel PIN
- **Simulation**: Trace-driven simulator based on MacSim and SimpleSSD

Benchmarks

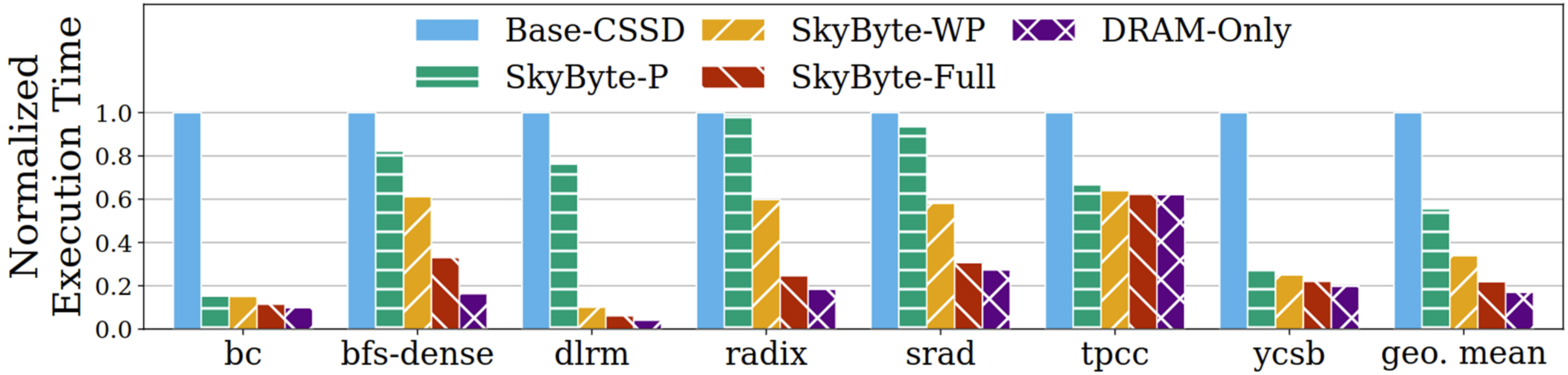
Name	Category	Memory Footprint	Write Ratio
bfs-dense	Graph Processing	9.13GB	25%
bc	Graph Processing	8.18GB	11%
radix	HPC	9.60GB	29%
srad	Image Processing	8.16GB	24%
ycsb	Database	9.61GB	5.0%
tpcc	Database	15.77GB	36%
dlrm	Machine Learning	12.35GB	32%

*SSD DRAM Cache Size Simulated: 512MB

Baselines

- **Base-CSSD:** The SOTA CXL-based SSD
- **DRAM-only:** The ideal case assuming infinite DRAM

End-to-End Performance Improvement of SkyByte



Normalized execution time of SkyByte variants over Base-CSSD (**lower is better**).

Skybyte-**P**: **P**age Migration

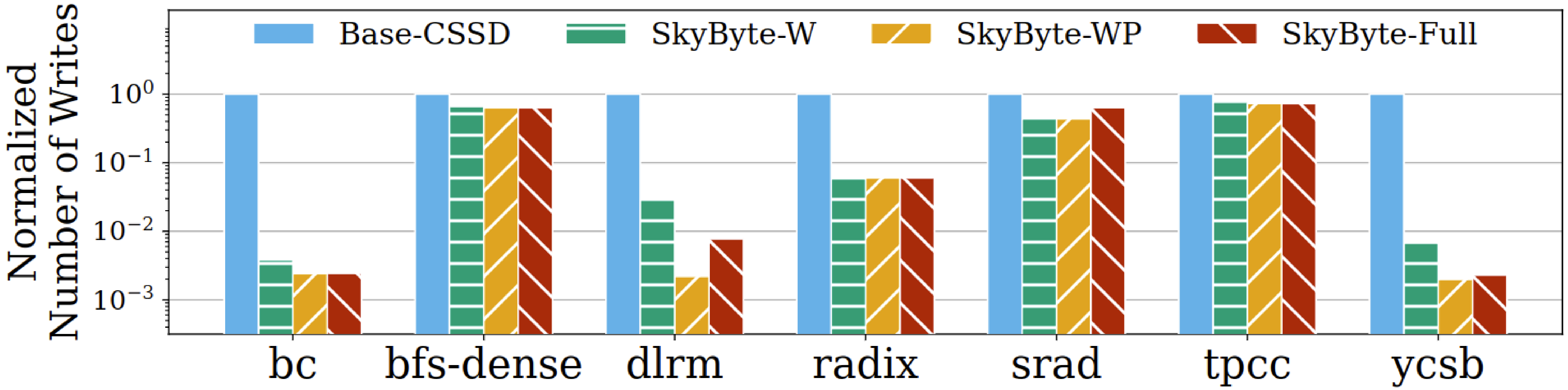
Skybyte-**WP**: **P** + **W**rite Log

Skybyte-**Full**: **WP** +
Context Switch

SkyByte outperforms SOTA CXL-SSD designs by 6.11×

SkyByte reaches 75% of the performance of ideal case

Write Traffic of SkyByte to Flash Chips



Normalized Flash Write Traffic of SkyByte variants over Base-CSSD (log scale, **lower is better**).

Skybyte-**P**: **P**age Migration

Skybyte-**WP**: **P** + **W**rite Log

Skybyte-**Full**: **WP** +
Context Switch

SkyByte reduces the write traffic to flash chips by 23.08×

SkyByte Summary

Coordinated Context Switch

Rearchitecting the SSD **DRAM Cache**

Adaptive Page Migrations

Outperforms SOTA CXL-SSD by 6.11×

Thank You!

Haoyang Zhang

Yuqi Xue, Yirui Eric Zhou, Shaobo Li, Jian Huang

Systems Platform Research Group

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN