LeaFTL: A Learning-based Flash Translation Layer for Solid-State Drives

Jinghan Sun, Shaobo Li, Yunxin Sun, Chao Sun, Dejan Vucinic, and Jian Huang. 2023.

In Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS 2023).

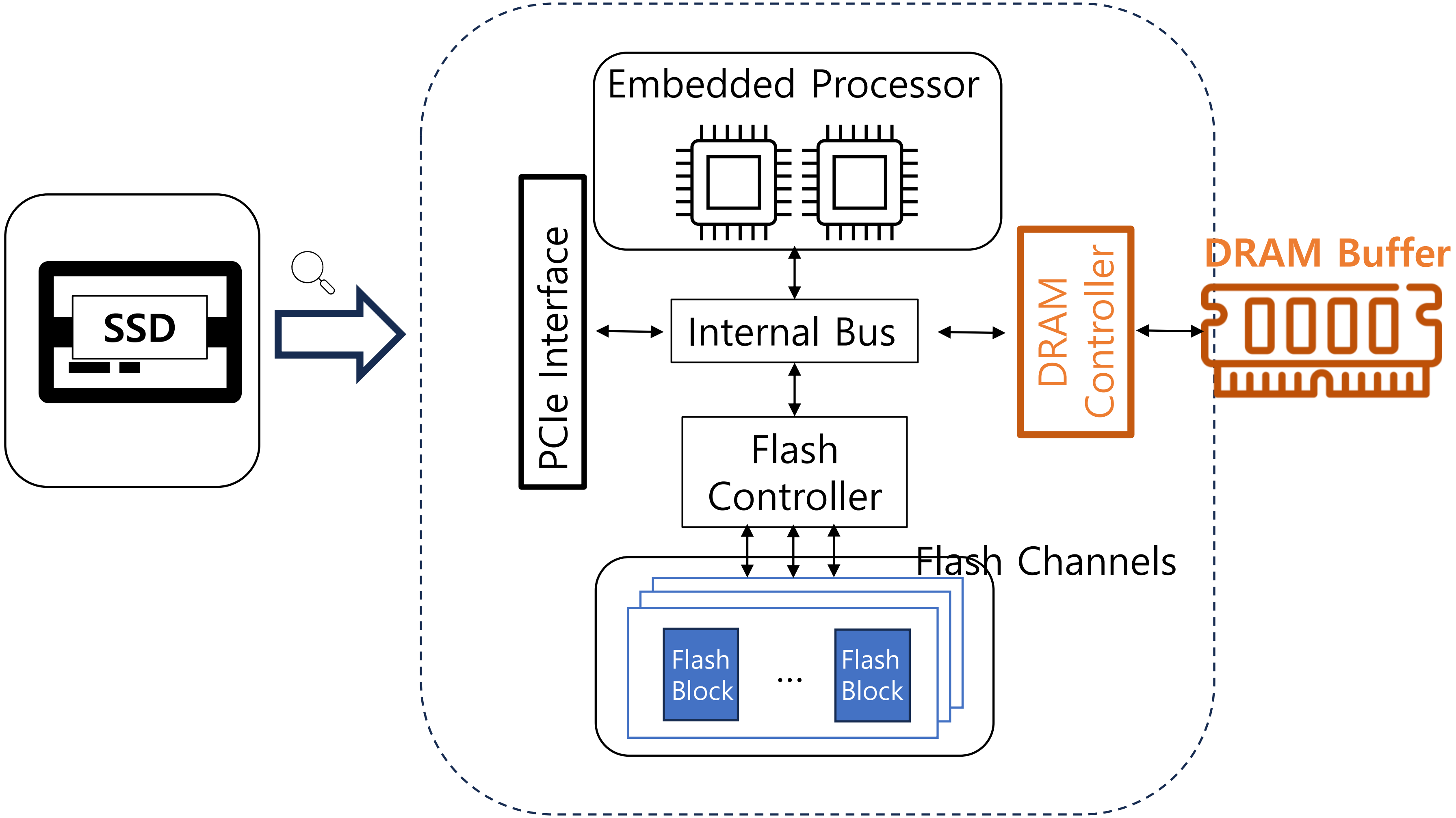
2024. 08. 07

Presentation by Yeongyu Choi

choiyg@dankook.ac.kr

Contents

Introduction

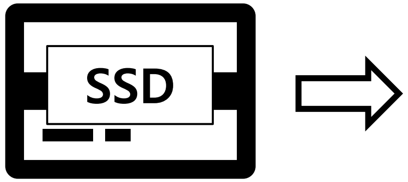


Less Scalable
- High Cost
- Limited Capacity

SSD DRAM is still the Bottleneck

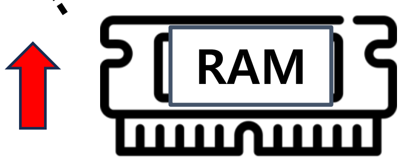
Introduction

Modern SSDs



Address Mapping Table

Logical Address	Physical Address
LPA0	PPA0
LPA1	PPA1
…	…



Direct LPA-PPA mappings require 8 Bytes per mapping

DRAM Capacity can't keep up with SSD Capacity

Approaches on Mappings

Page-Level Mapping:
Direct LPA-PPA mapping for fast lookup

Entire mapping table requires large storage space

Block-Level Mapping:
Reduce mapping table size by storing via blocks

Overhead in Lookup in flash page

Hybrid Mapping:
Takes advantage of a mix of Page-level and Block-level Mappings

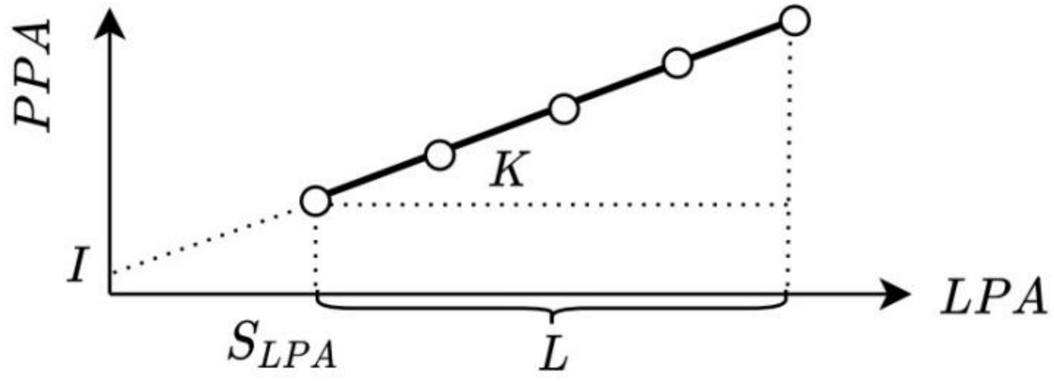
Incurs in significant GC Overhead

LeaFTL
A Learning-Based Flash Translation Layer for Modern SSDs

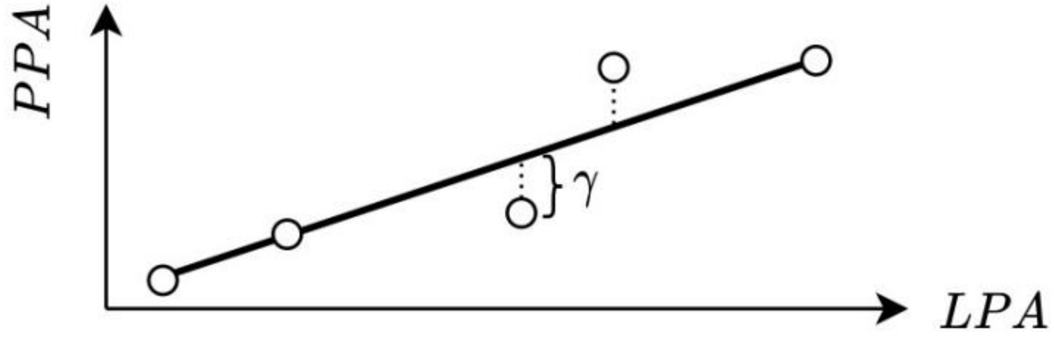
Instead of 1:1 mapping in Page-level Mapping

Exploit learning techniques to identify various LPA-PPA mapping patterns

Learning-Based Techniques



(a) Precise Linear Approximation



(b) Inaccurate Linear Approximation

Learned index are encoded in 8 bytes:

(S_{LPA}, L, K, I)

- S_{LPA}: 1 Byte
- L: 1 Byte
- K: 2 Bytes
- I: 4 Bytes

$$PPA = f(LPA) = \lceil K * LPA + I \rceil$$

where $LPA \in [S_{LPA}, S_{LPA} + L]$

Learned Index Segment

Accurate segments

Approximate segments

Piecewise Linear Regression

(Accurate segments) ← $PPA = f(LPA) = \lceil K * LPA + I \rceil$ → (Approximate segments)

Exact PPA

Not exact PPA, error bound $[-\gamma, \gamma]$

LPA = 2
$\lceil 1.00 * 2 + 32 \rceil$
[K * LPA + I]
PPA = 34

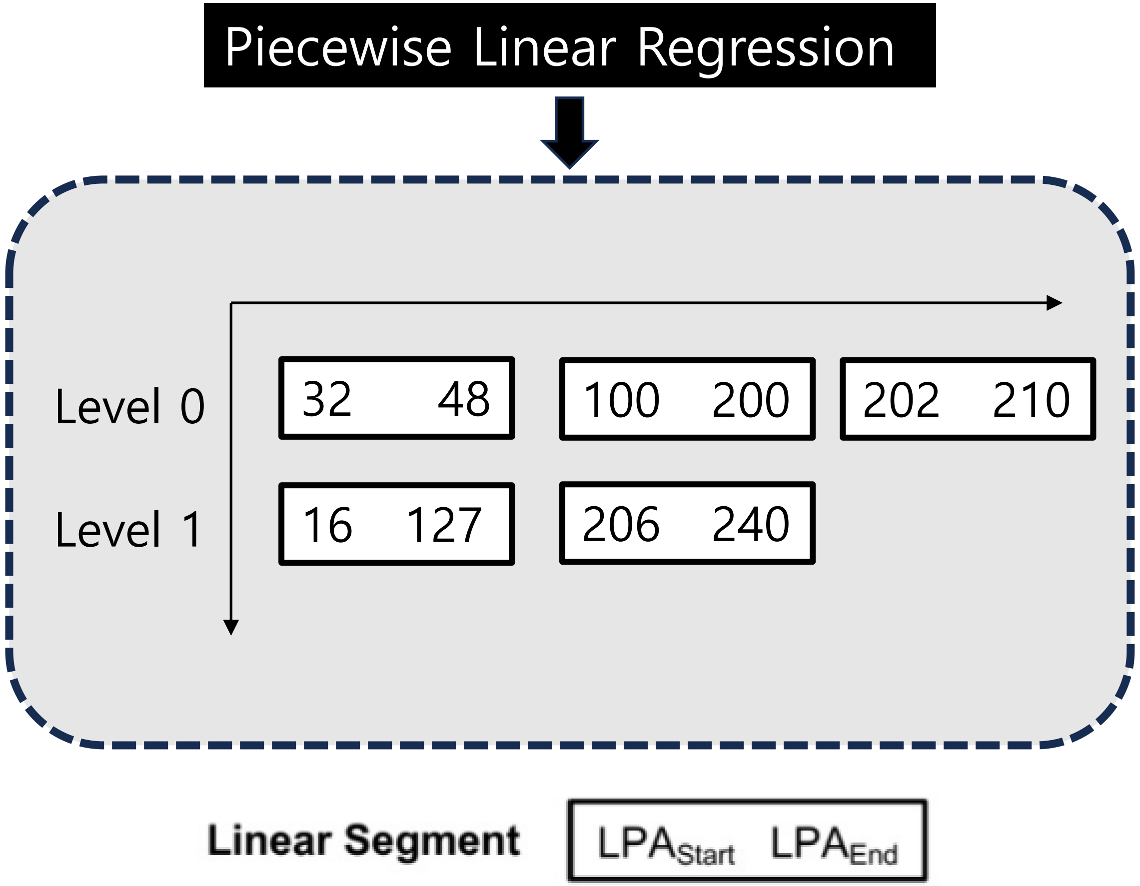
	1B	1B	2B	4B
Segment	S_{LPA}	L	K	I

Type	LPAs	PPAs	Index Segment			
Accurate	[0, 1, 2, 3]	[32, 33, 34, 35]	0	3	1.00	32
Approximate	[0, 1, 4, 5]	[64, 65, 66, 67]	0	5	0.56	64

LPA = 4
$\lceil 0.56 * 4 + 64 \rceil$
[K * LPA + I]
PPA = 66.24(67)

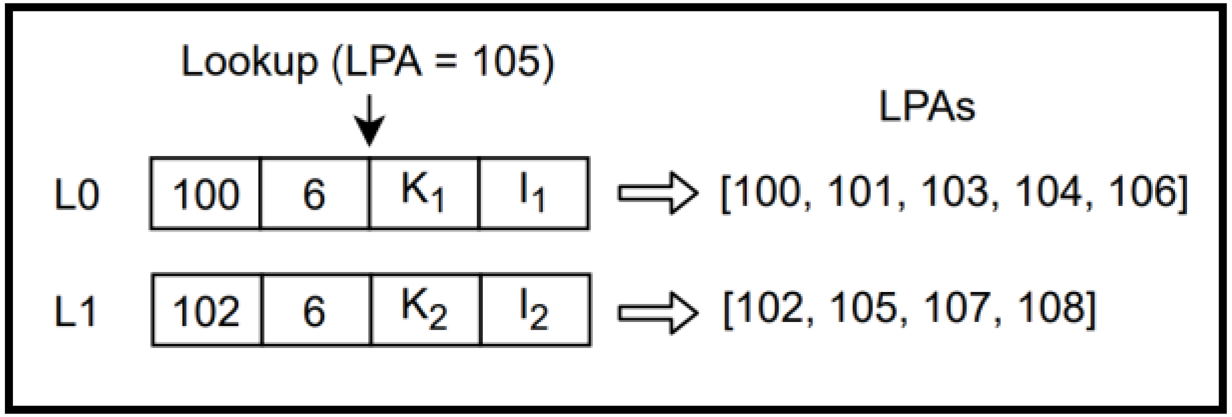
Figure 6: Types of learned segments in LeaFTL.

Log-Structed Address Mapping Table

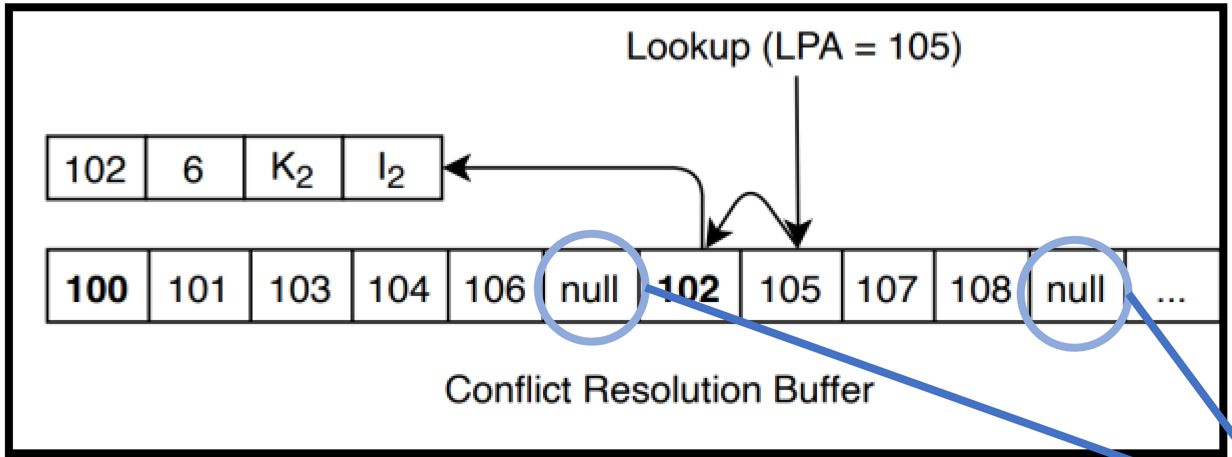


- Non-overlapping segments are sorted by their LPA ranges
- Overlapping segments are allowed across levels

Conflict Resolution Buffer



- Possible to get inaccurate PPA
- Affect to segment compaction

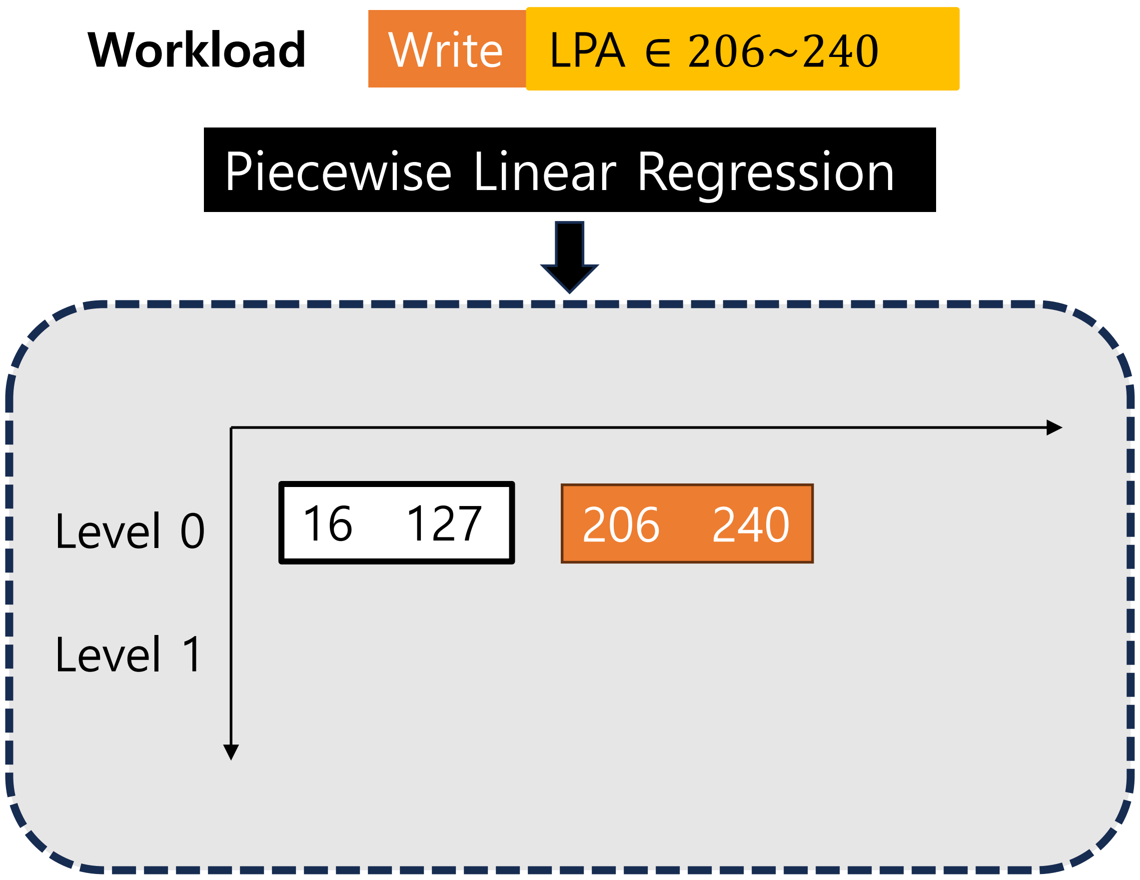


- CRB is a nearly sorted list, follows its **insertion deletion and lookup are fast**
- Each LPA takes only 1 byte and its structured guarantees no redundant LPAs

Very space efficient with at most 256 entries

Separate segments

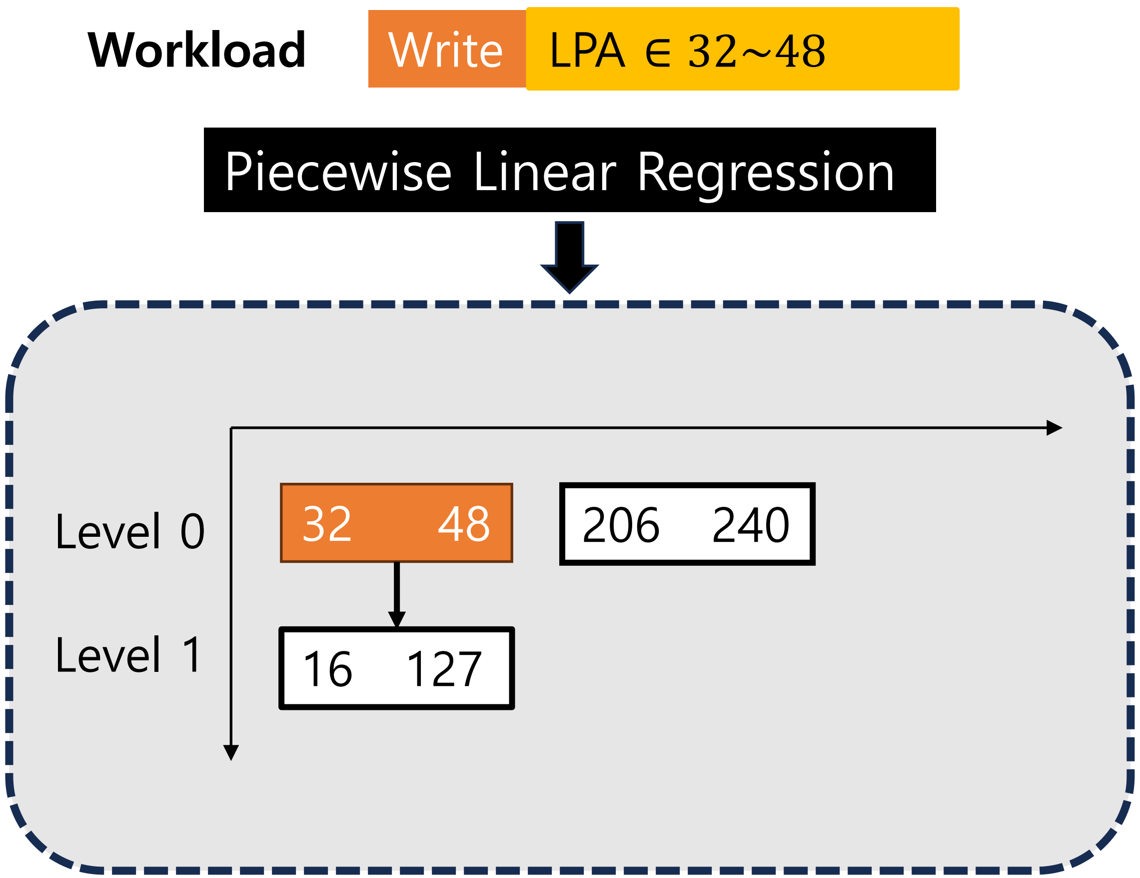
Log-Structured Table Operations



New segments are appended to the topmost level

If it is approximate it is added to CRB

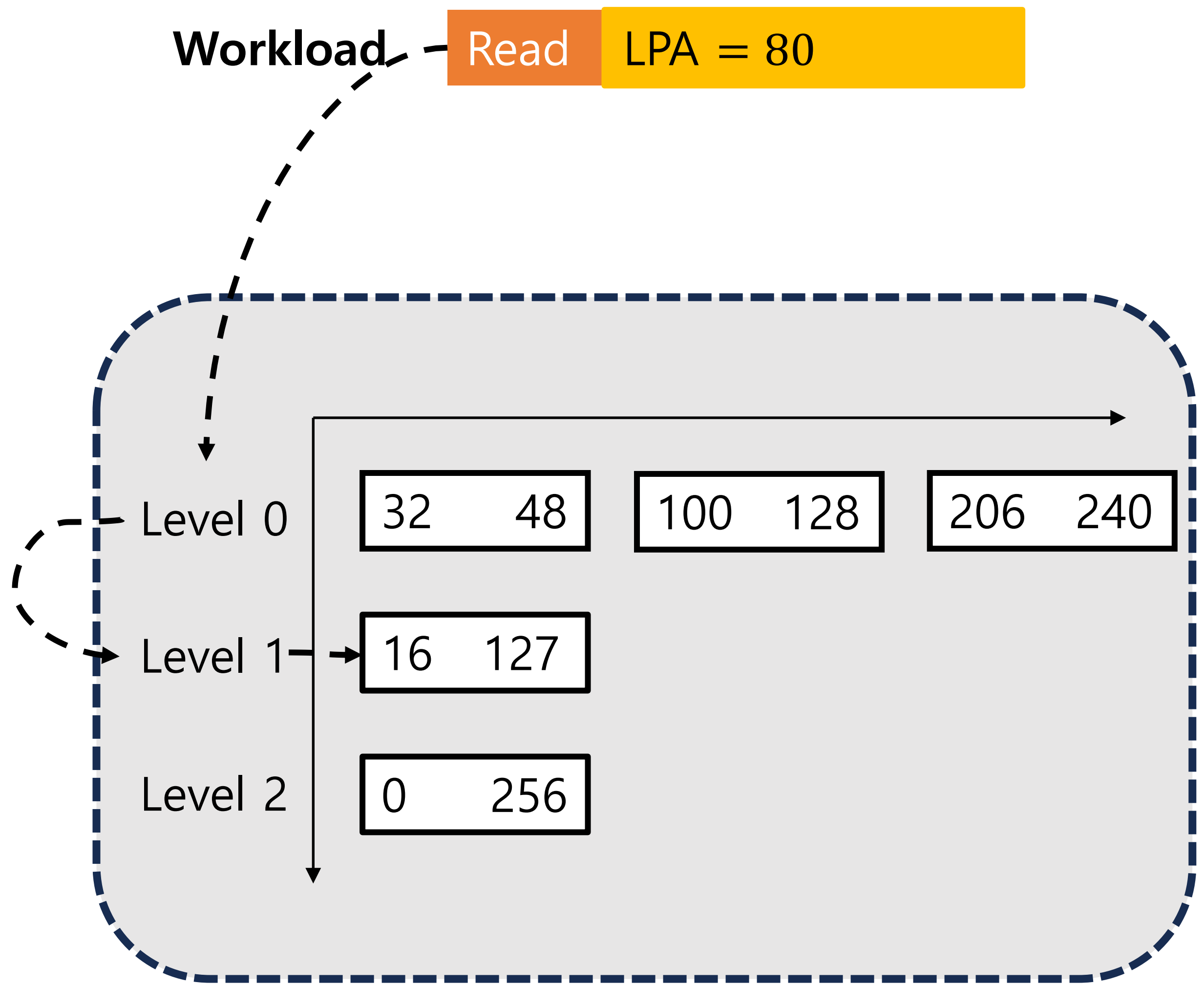
Log-Structured Table Operations



New segments are appended to the topmost level

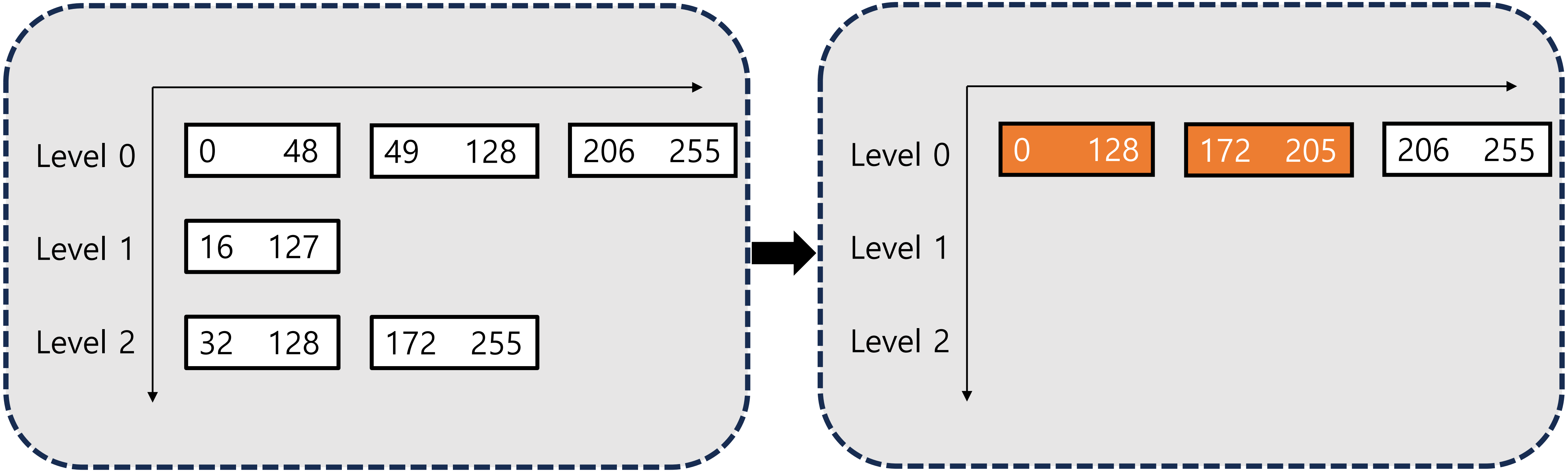
If it is approximate it is added to CRB

Log-Structured Table Operations



Search from the topmost level, and always find the first matching segment

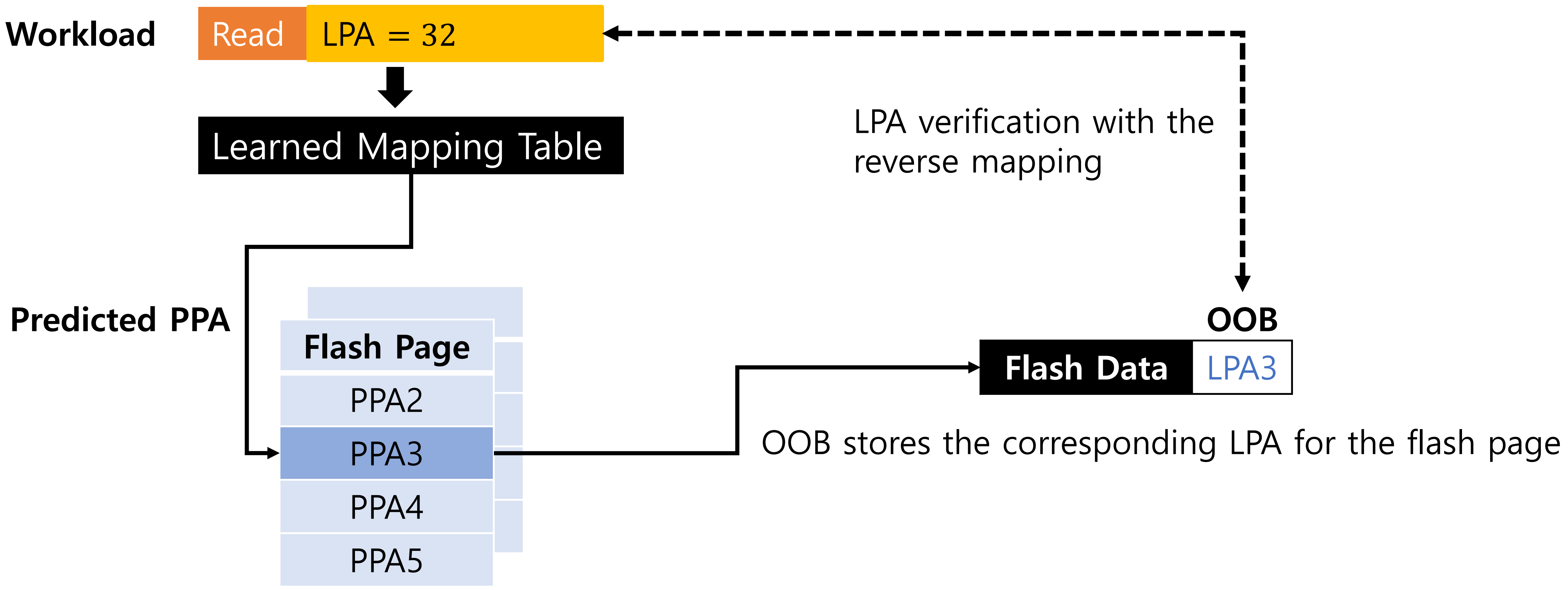
Log-Structured Table Operations



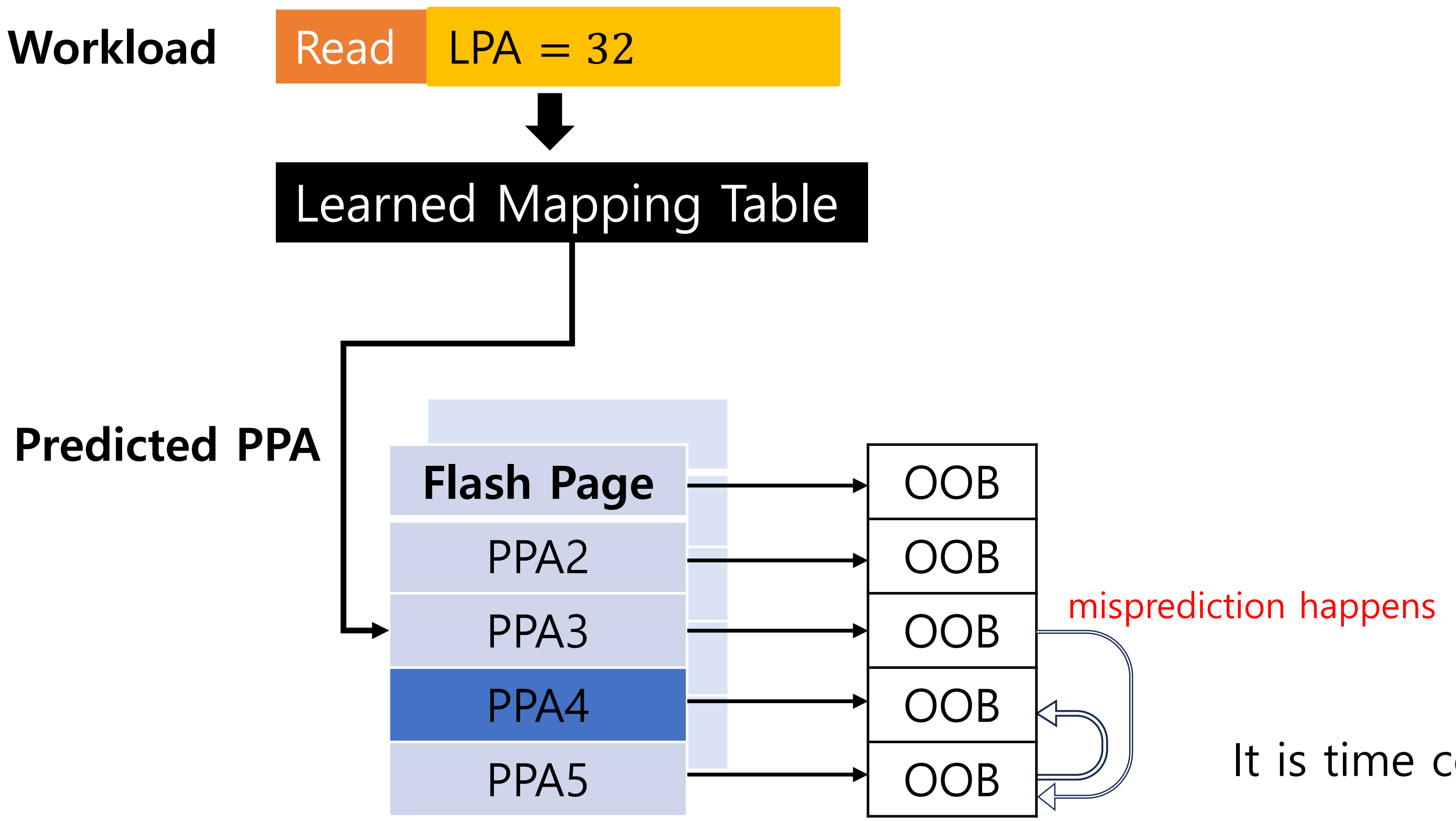
Before Compaction

After Compaction

Misprediction Verification with OOB Metadata (OOB:Out-of-Band)

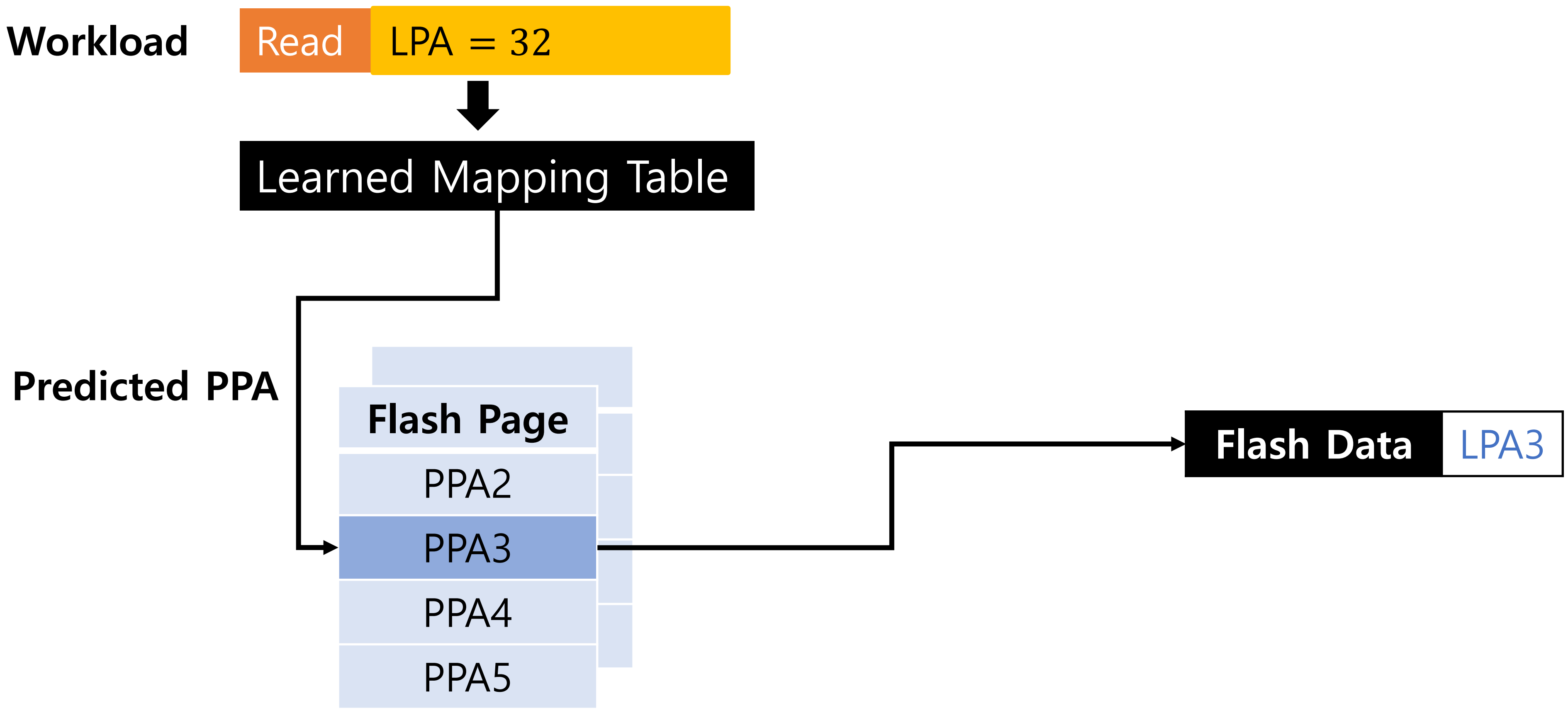


Verifying Address Translation with OOB Metadata (OOB:Out-of-Band)

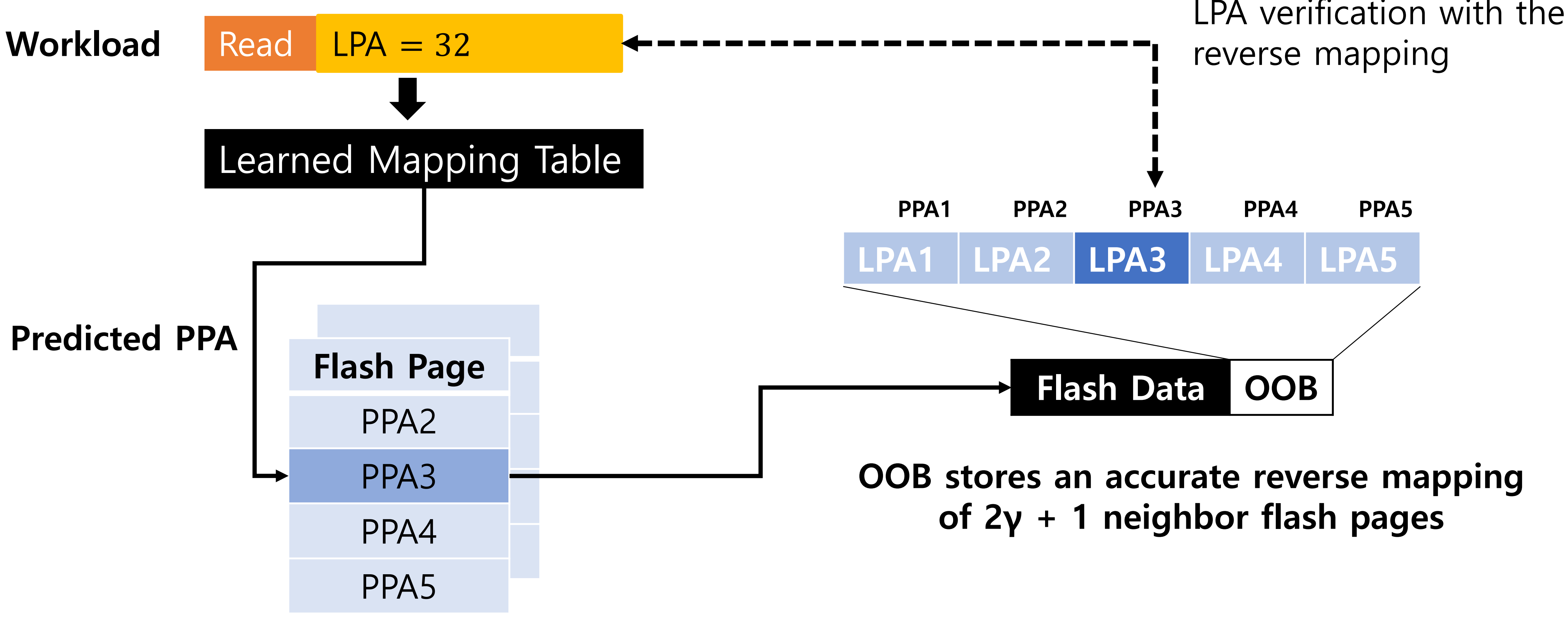


It is time consuming to search the correct flash page

Verifying Address Translation with OOB Metadata (OOB:Out-of-Band)

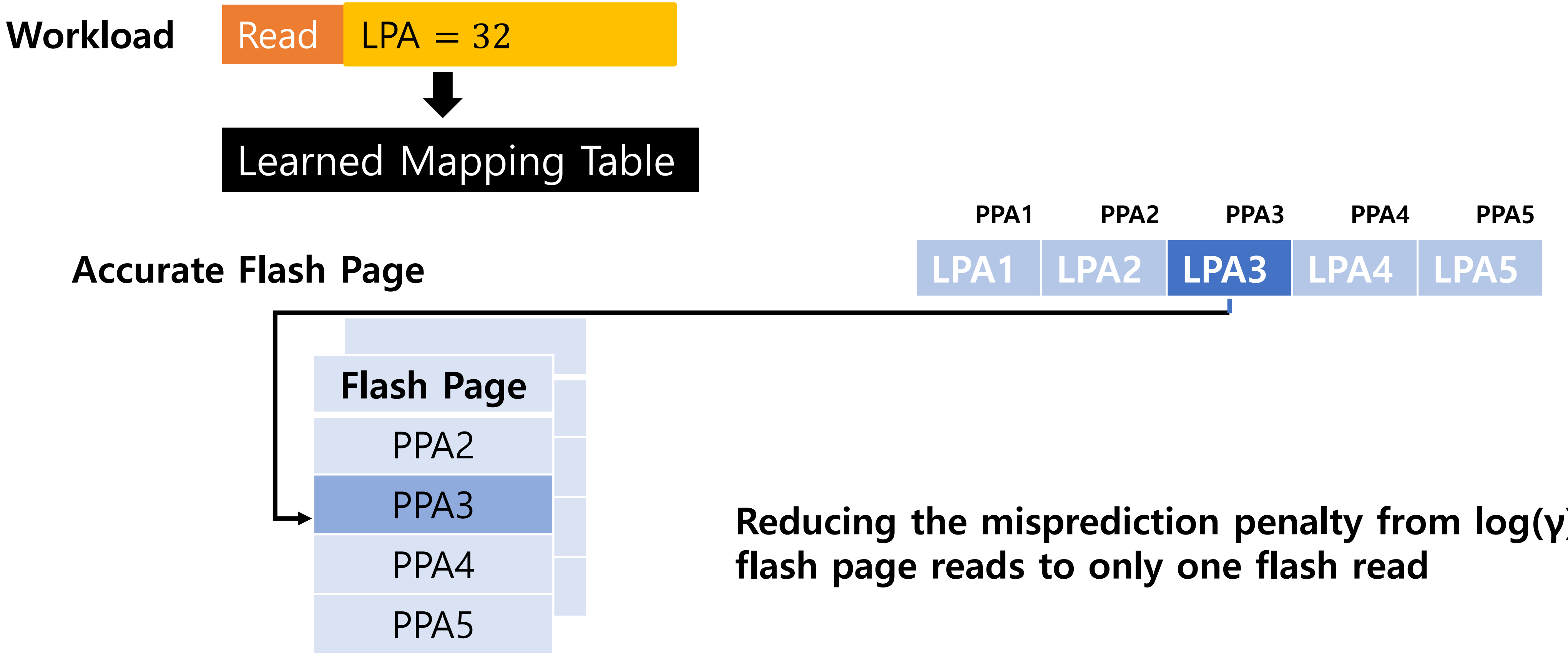


Verifying Address Translation with OOB Metadata (OOB:Out-of-Band)



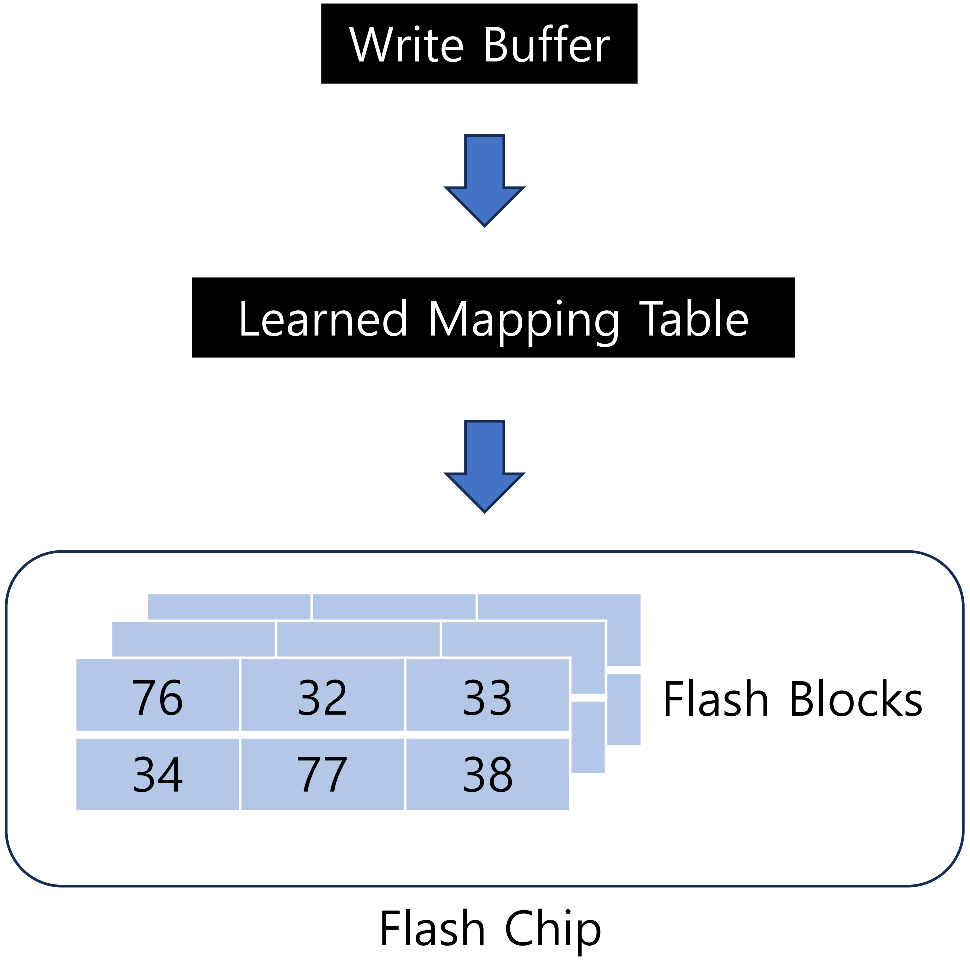
OOB stores an accurate reverse mapping of 2γ + 1 neighbor flash pages

Verifying Address Translation with OOB Metadata (OOB:Out-of-Band)



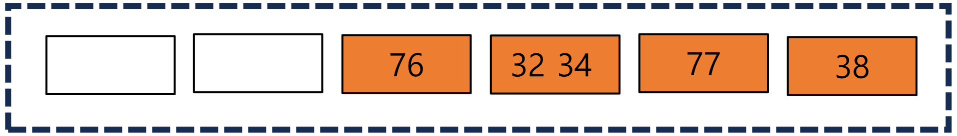
Reducing the misprediction penalty from log(γ) flash page reads to only one flash read

Optimized Flash Allocation



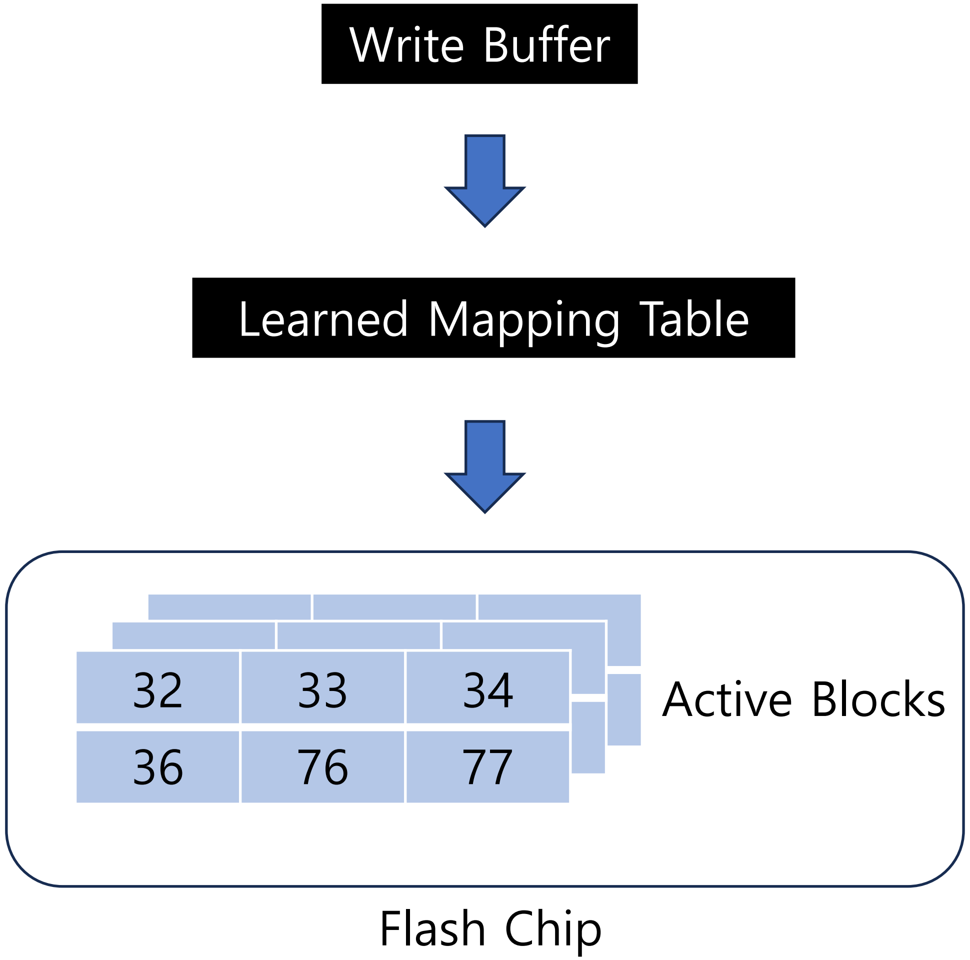
Flushed flash pages

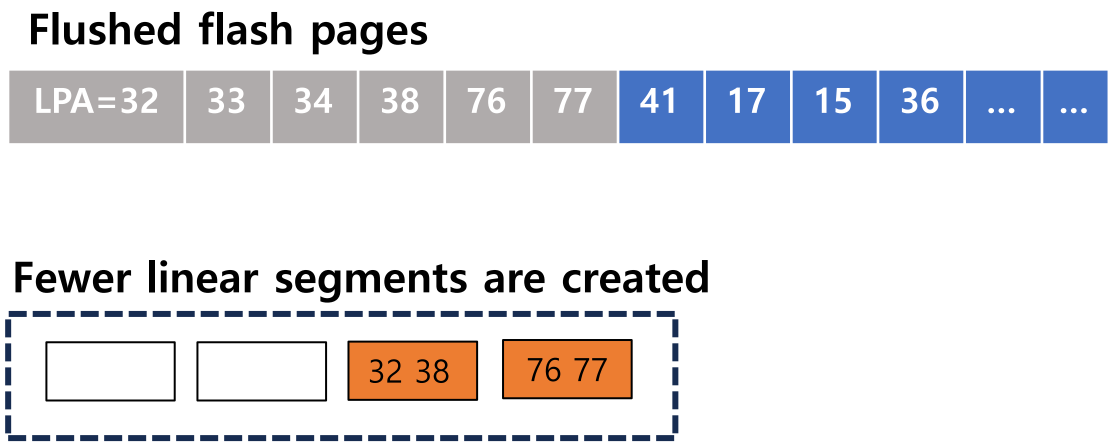
Four new linear segments are created



Flushing flash pages from the write buffer directly is less optimal

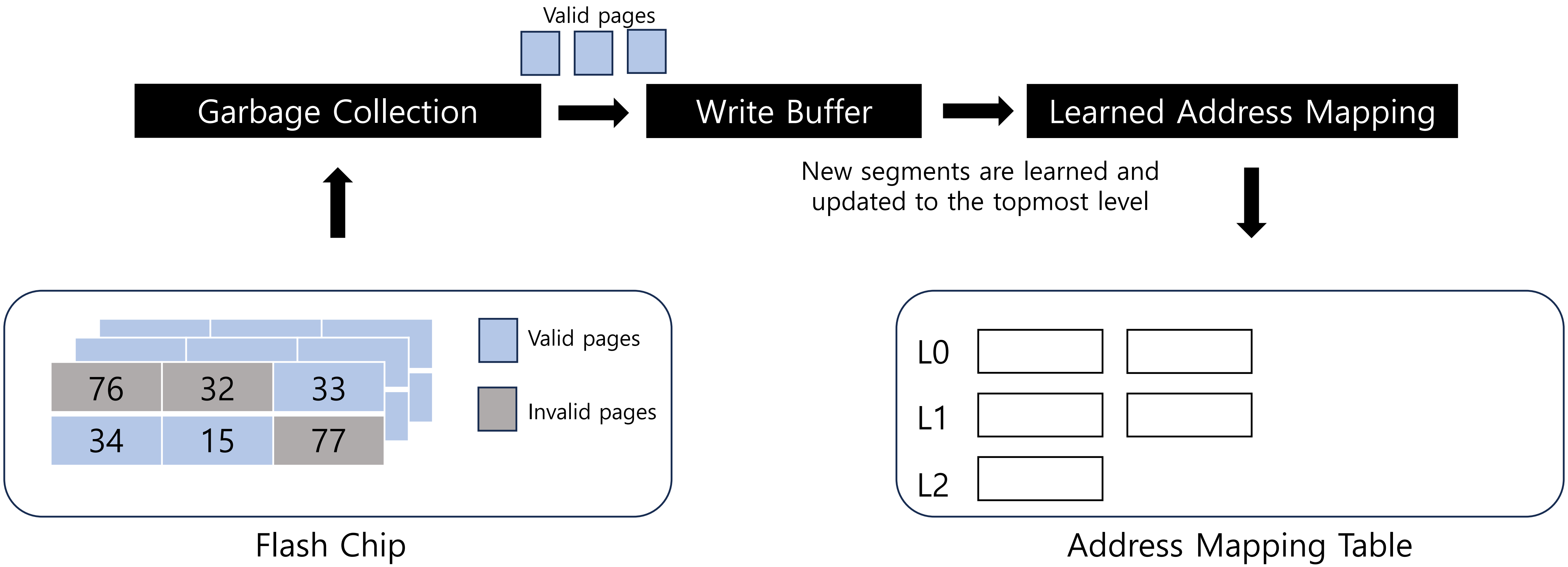
Optimized Flash Allocation



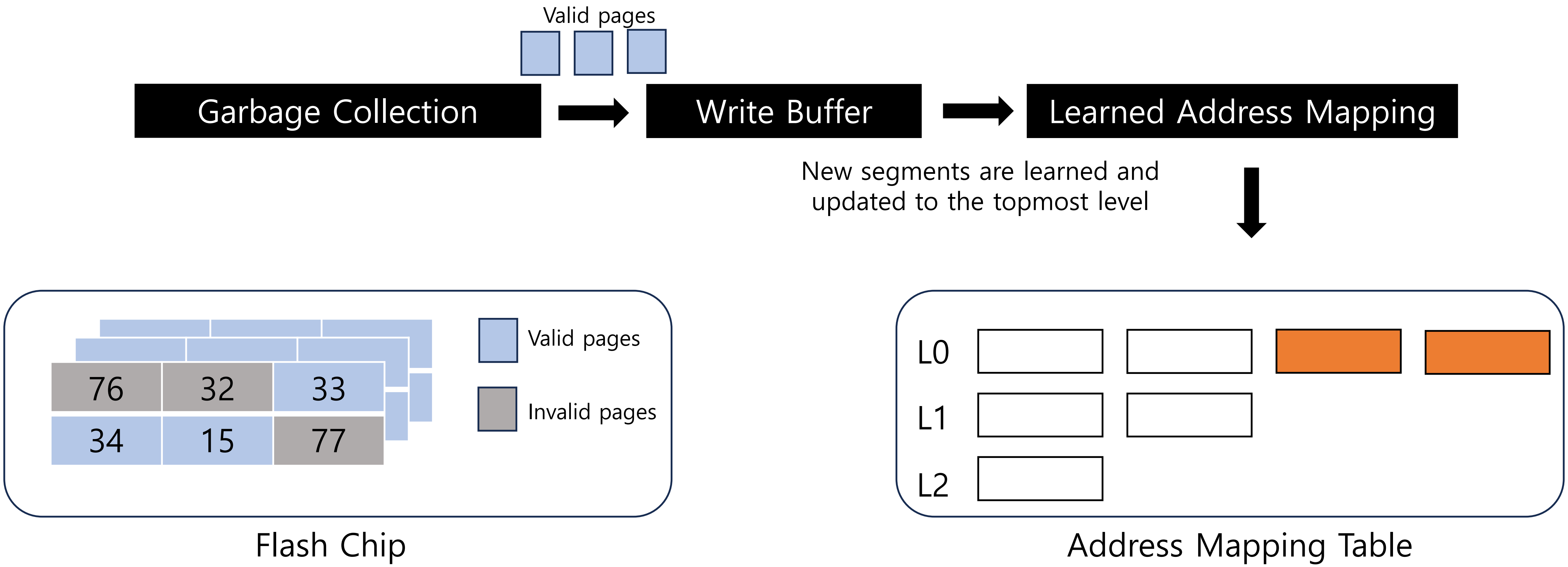


Optimization: Flash pages are sorted by their LPAs before flushed from the write buffer

Coordinated Garbage Collection

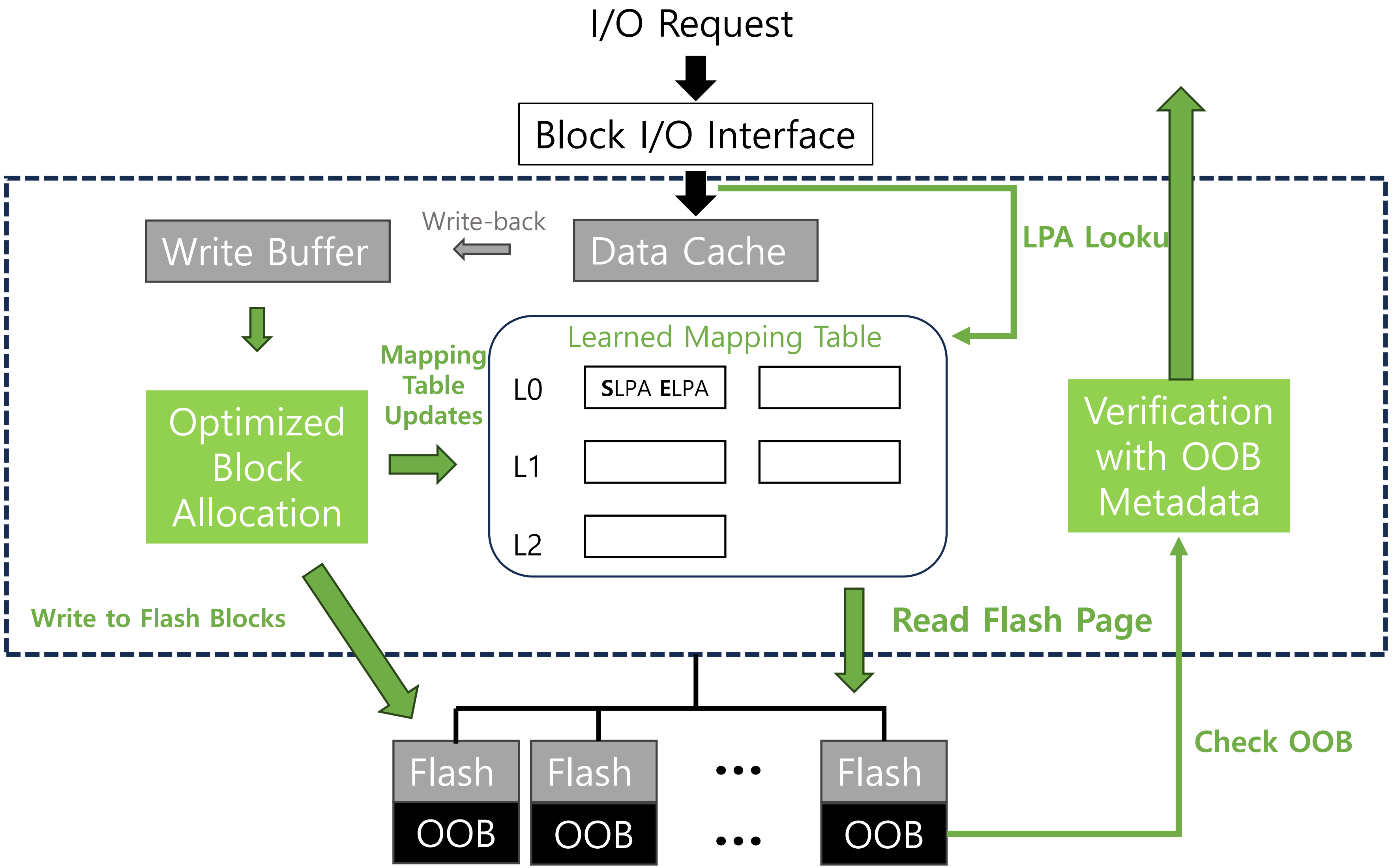


Coordinated Garbage Collection



LeaFTL learns new segments to avoid messing up existing segments in GC

Put It All Together



Evaluation

Implementation Details

- WiscSim Simulator
- A Real 1TB Open-Channel SSD with 16 channels

SSD configuration

Parameter	Value	Parameter	Value
Capacity	2TB	#Channels	16
Page size	4KB	OOB Size	128B
DRAM size	1GB	Pages/block	256
Read latency	20us	Write latency	200us
Erase	1.5ms	Overprovisioning ratio	20%

Workloads

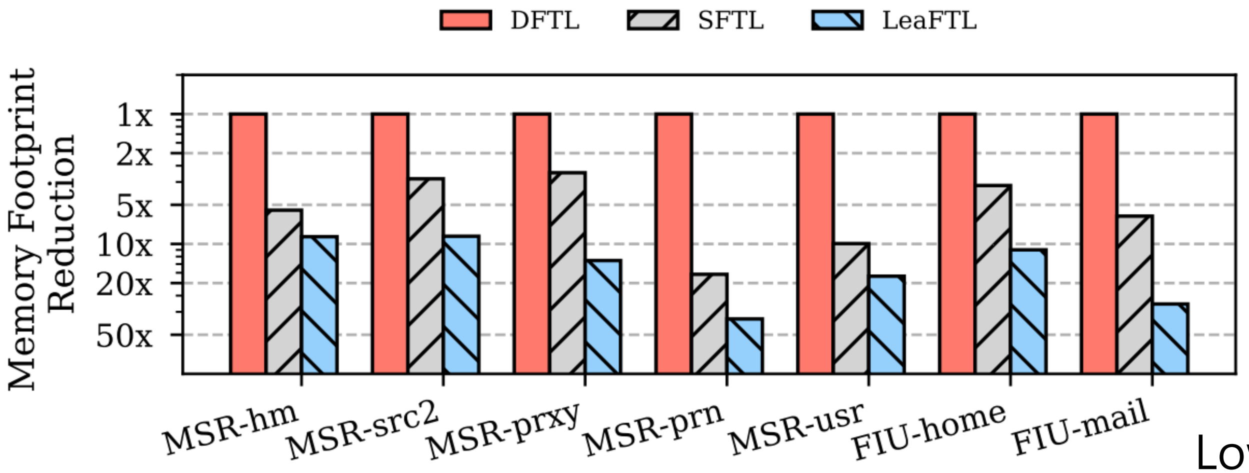
- Block I/O Traces: from enterprise servers and university servers
- Data-Intensive Applications: FileBench, BenchBase

Comparison

- DFTL: Demand-based caching FTL
- S-FTL: Compresses sequential LPA-PPA entries
- LeaFTL with different error bounds (0, 1, 4, 8, 16)

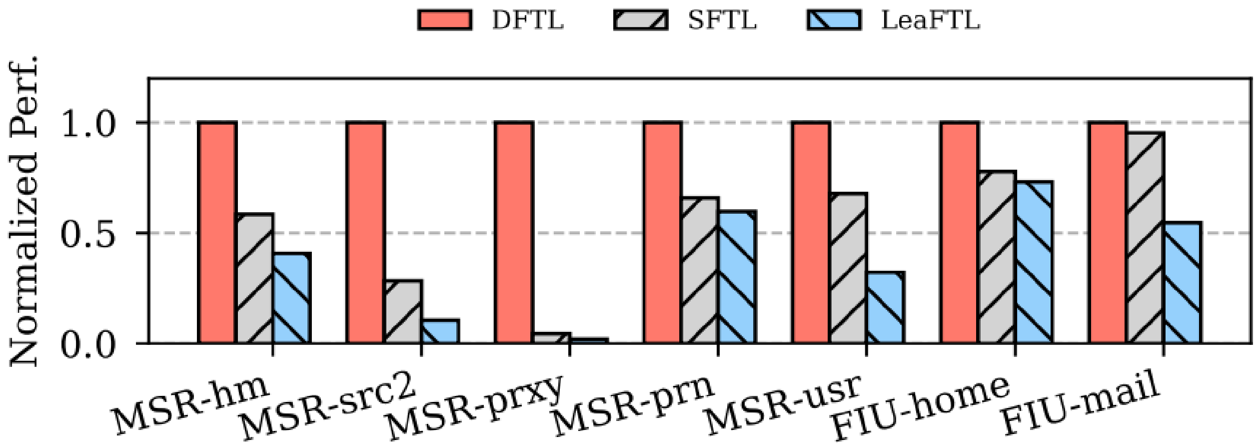
Evaluation

Memory Footprint Analysis & Performance Analysis



➡ **Avg. x2.9 memory saving**

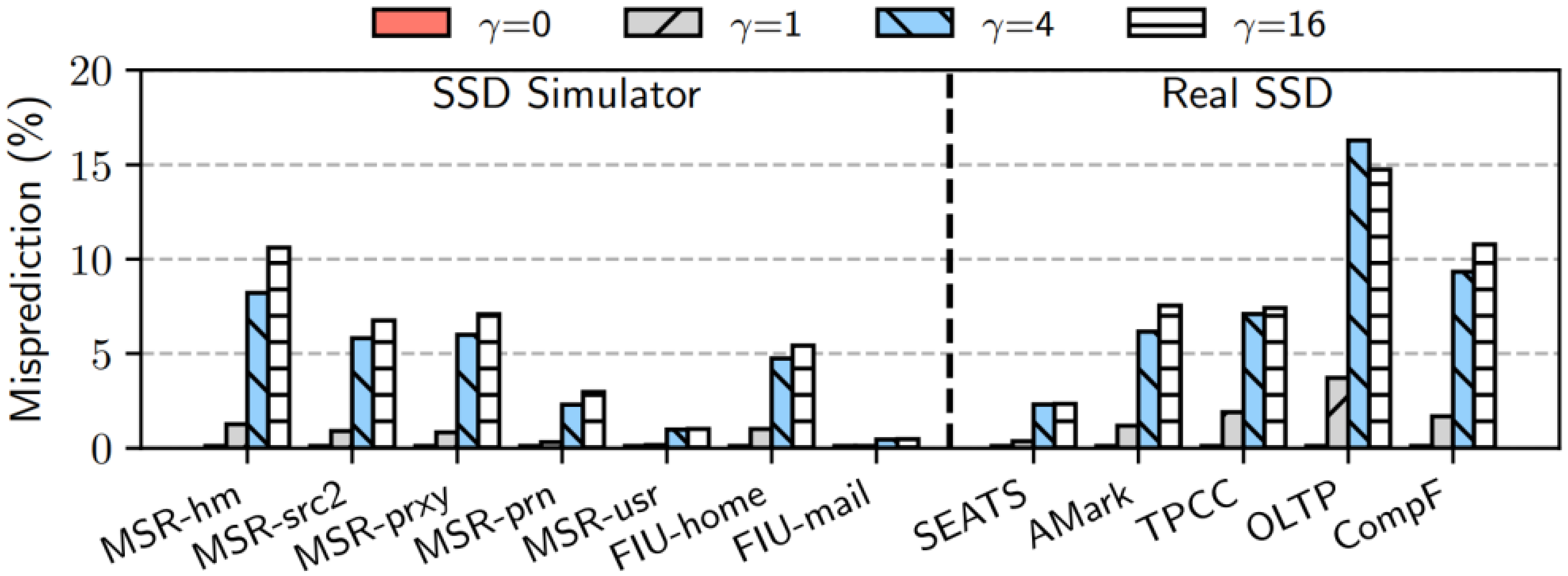
Lower is Better



➡ **Avg. x1.4 reduced storage access latency**

Evaluation

Misprediction

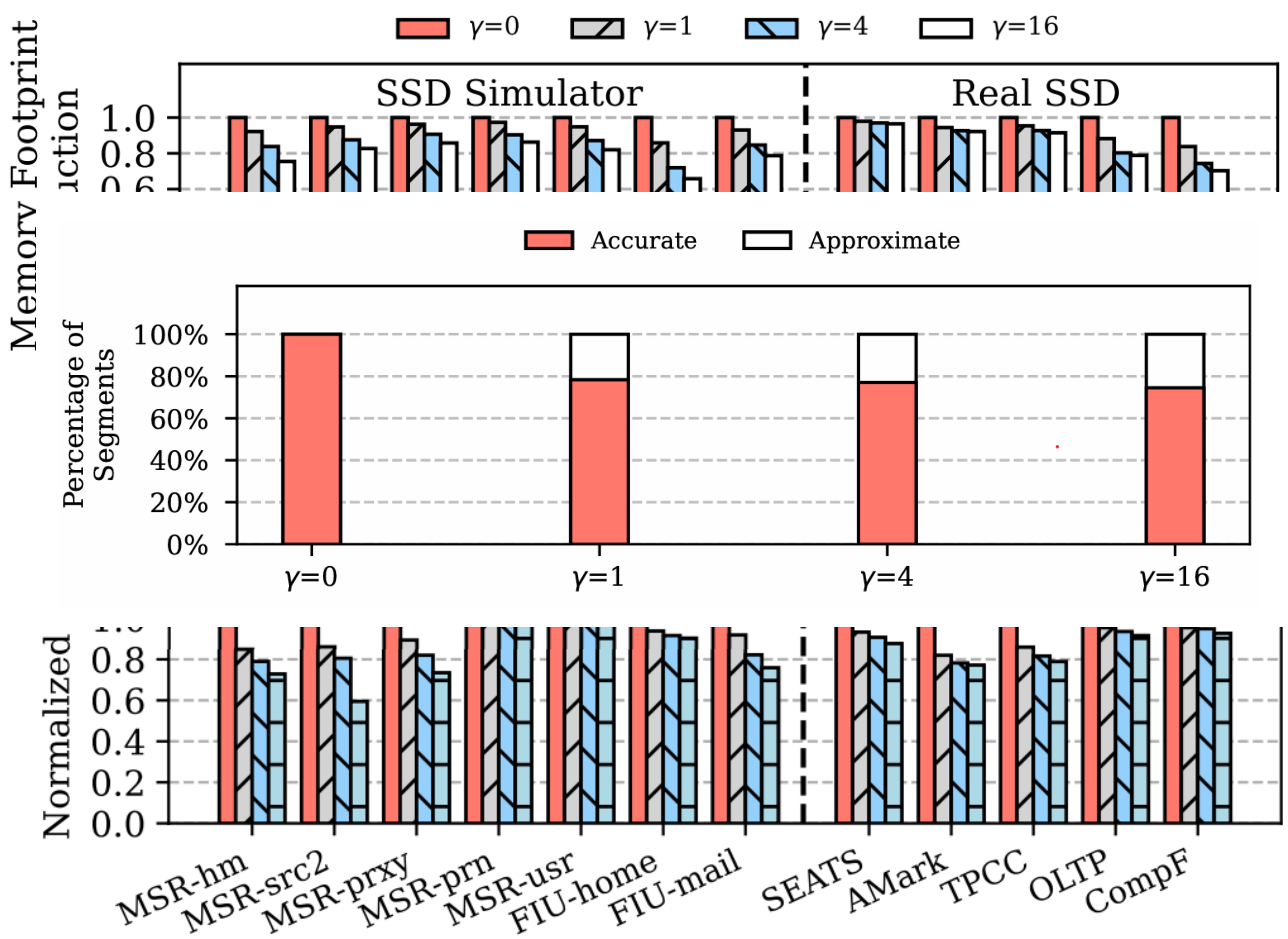


Error bound γ = 16, misprediction ratio <10% Avg.

Every misprediction incurs in only one additional flash read

Evaluation

Memory Footprint Analysis on Different γ



is Better

Avg. x1.3 memory saving (Real SSD Avg. x1.2)

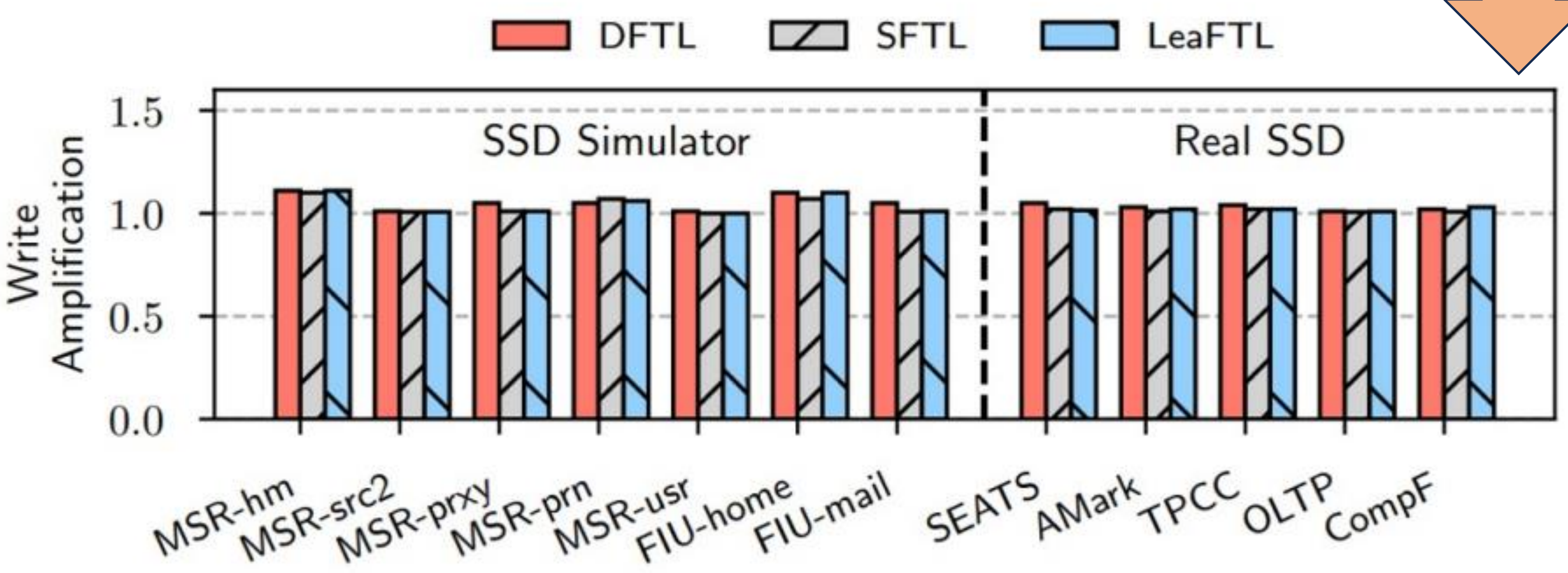
Avg. x1.3 reduced storage access latency (Real SSD Avg. x1.2)

Evaluation

Write Amplification Factor Analysis

WAF, the ratio between the actual and requested flash writes

Lower is Better



Conclusion

- LeaFTL uses a simple but effective learning-based technique to reduce memory consumption
- LeaFTL stores learned segments in a log-structured manner to avoid re-learning
- LeaFTL uses OOB metadata to verify its address translation
- LeaFTL consumes 2.9x less memory and improves performance by 1.4x

Thank you