Persistent Buffer Cache for High-performance Storage Systems

OpenInfra Community Days Korea 2021

UNIST (NECSST lab.)

Hyunsub Song

TABLE OF CONTENTS

TABLE OF CONTENTS

Persistent Memory (PM)

Persistent Memory Features

- Non-volatility
- Byte-level random access
- Fast access time (*nanoseconds*)

Storage

+

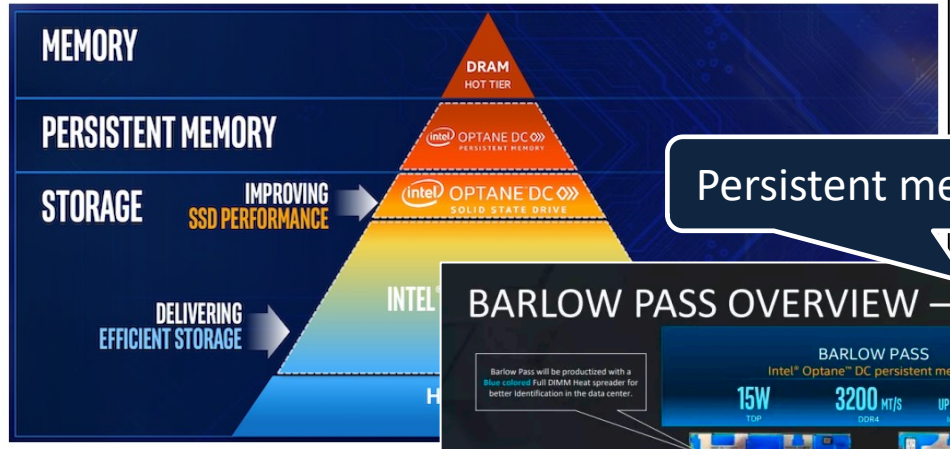
Memory

APRIL 2ND, 2019 by Brian Beeler

Share 107 Tweet Share

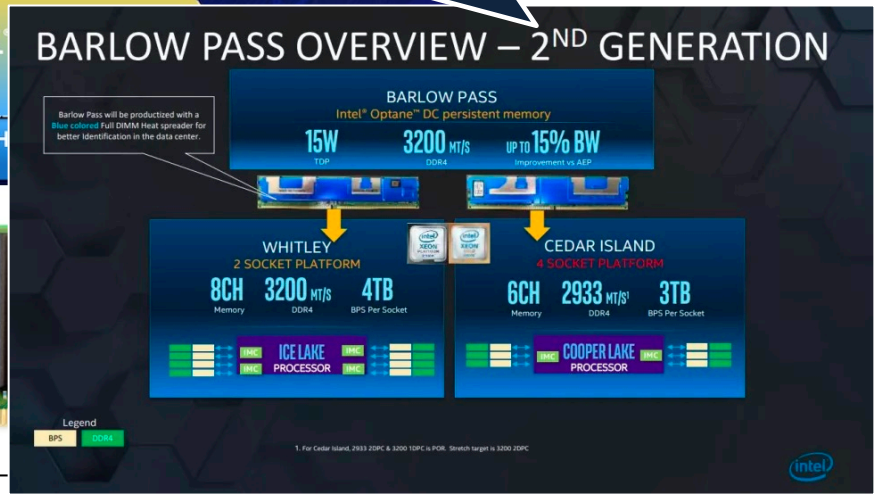
Intel Optane DC Persistent Memory Module (PMM)

Intel has talked about Optane DC Persistent Memory Modules (PMM) publicly for over a year now, espousing the benefits of a new tier of data centric architecture that sits between DRAM and Optane DC SSDs, with sequentially slower SSD and HDD media cascading down the pyramid to tape at the archive level. The goal with persistent memory has always been to move more data closer to the CPU, offering DRAM-like latency with storage-like persistence and capacities. After a year of listening to hardware and software partners talk about the benefits of persistent memory in the lab, with the release of the second generation Intel Xeon Scalable Processors, Optane DC PMEM is now available across a wide variety of server solutions.

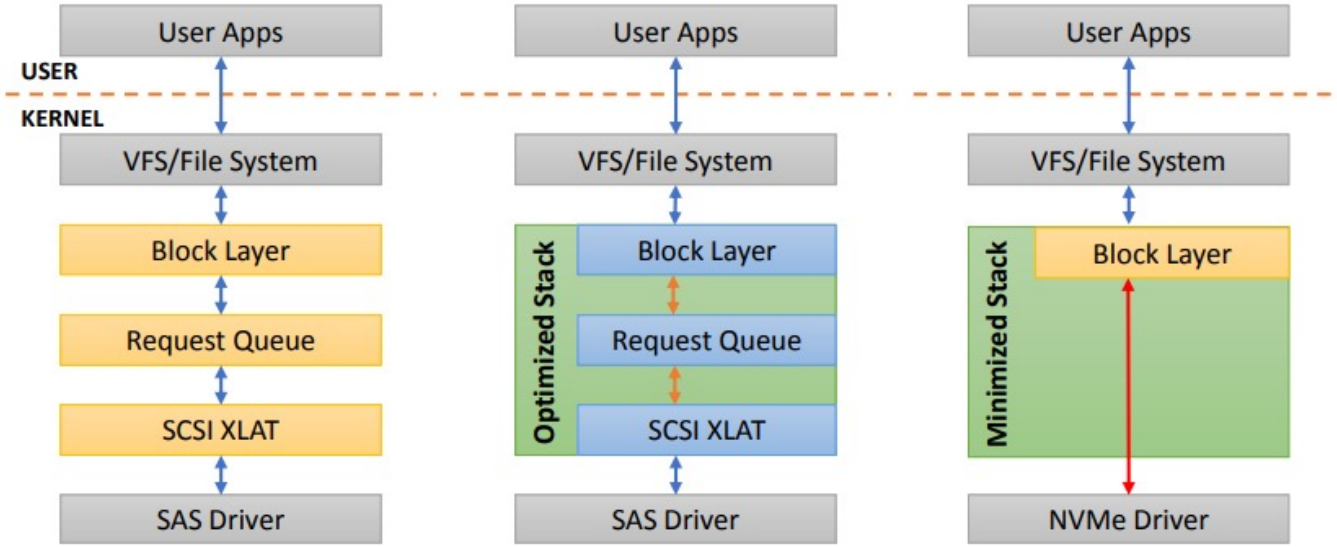


Intel Optane DC Persistent Memory Module

Persistent memory is evolving



Evolution of critical path

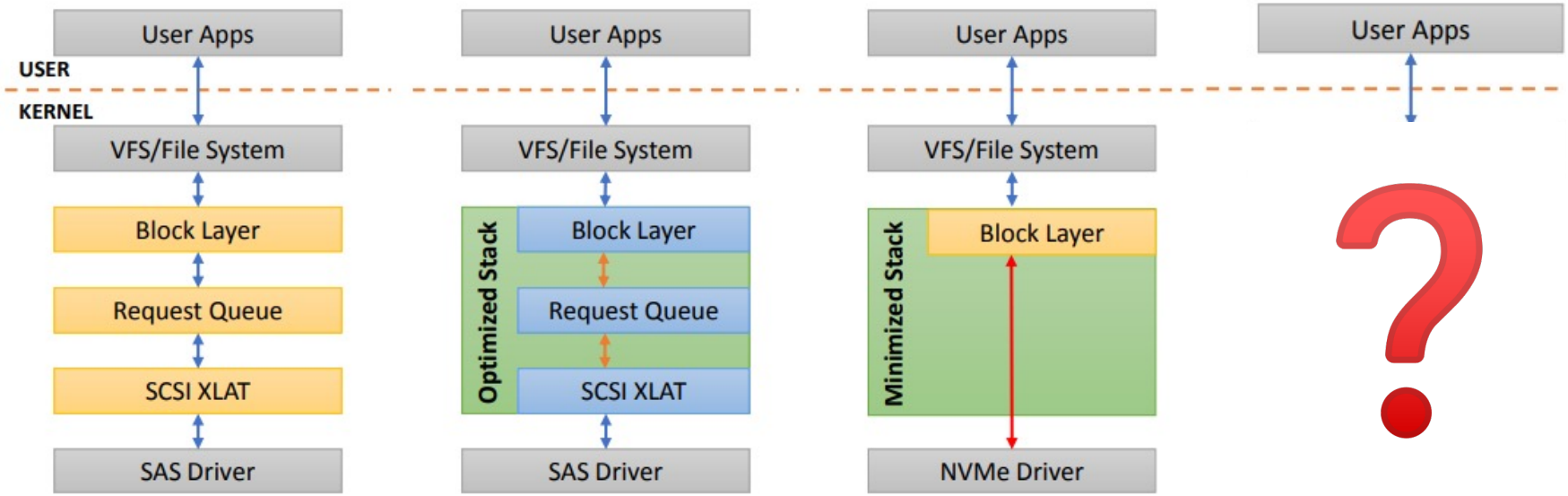


HDD (~10ms)

SAS SSD (~150µs)

NVMe SSD (<100µs)

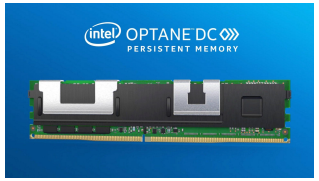
Evolution of critical path



HDD (~10ms)

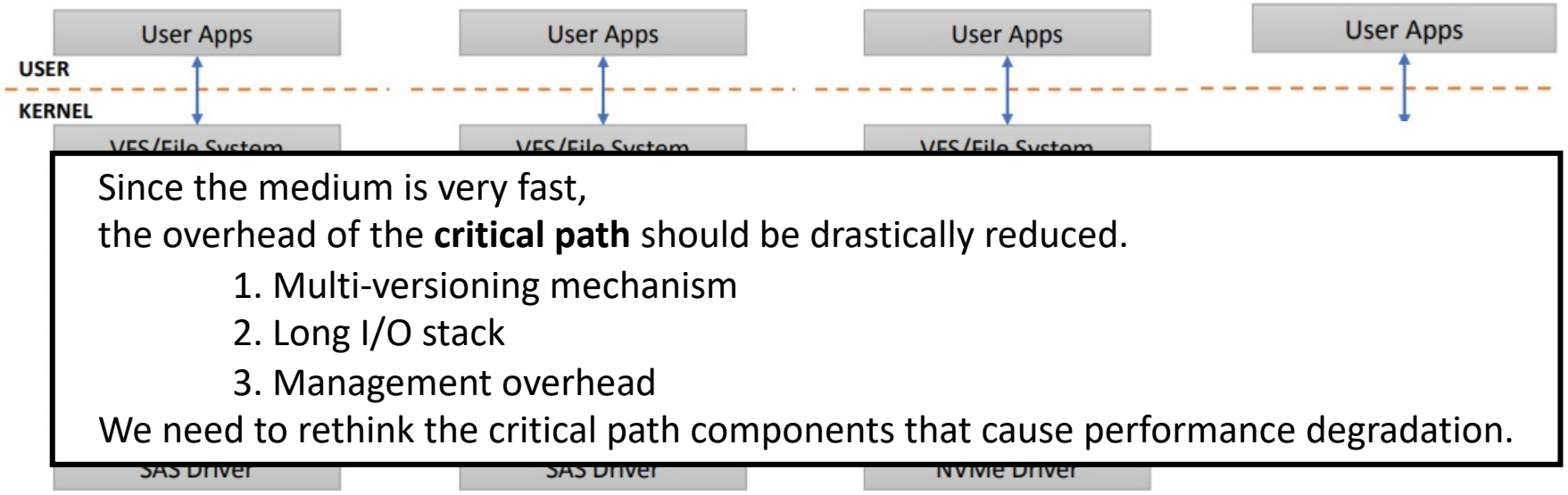
SAS SSD (~150µs)

NVMe SSD (<100µs)



PM (20~100ns)

Evolution of critical path



Since the medium is very fast,
the overhead of the **critical path** should be drastically reduced.

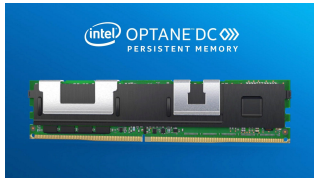
1. Multi-versioning mechanism
2. Long I/O stack
3. Management overhead

We need to rethink the critical path components that cause performance degradation.

HDD (~10ms)

SAS SSD (~150µs)

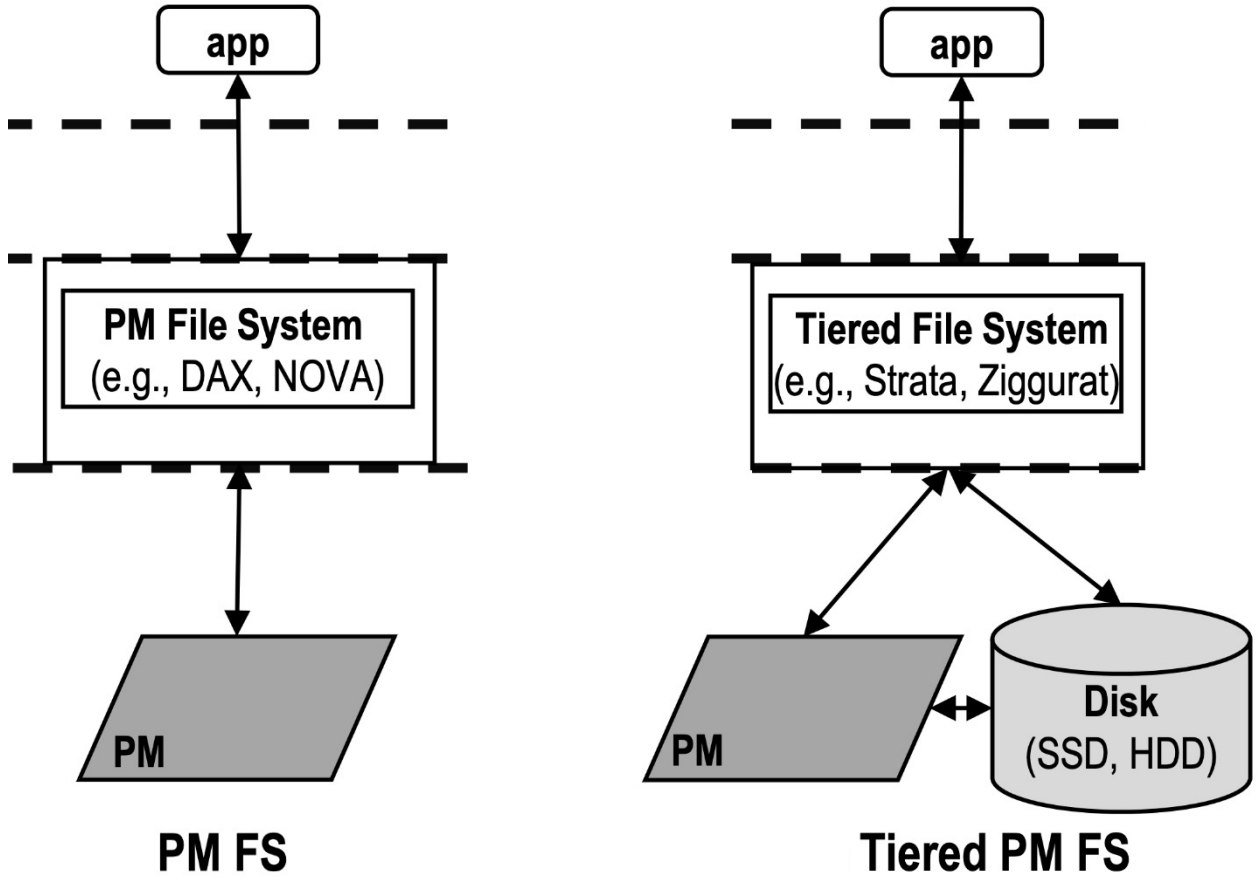
NVMe SSD (<100µs)



PM (20~100ns)

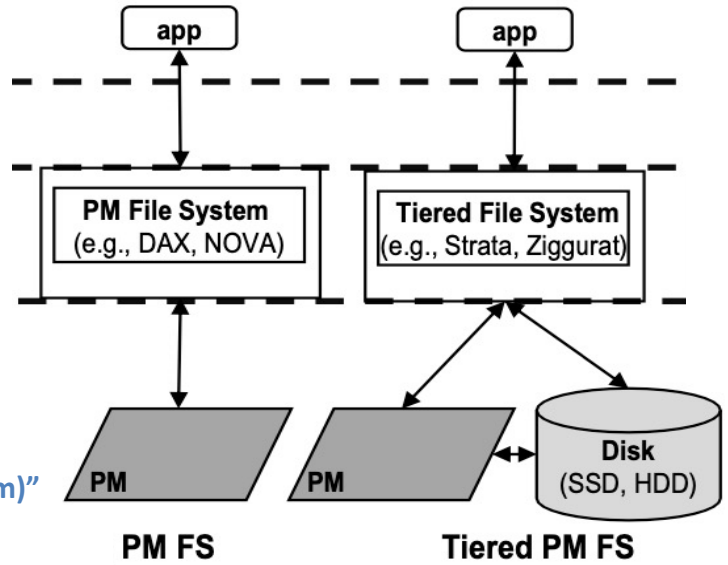
Studies that consider PM as storage

- PM-dedicated file system and tiered PM file system



PM Targeted File Systems

- **Designed to reap PM performance**



SOSP 2009	"BPFS (Better I/O Through Byte-Addressable, Persistent Memory)"
SC 2011	"SCMFS (SCMFS: A File System for Storage Class Memory)"
EuroSys 2014	"PMFS (System Software for Persistent Memory)"
EuroSys 2014	"Aerie (Aerie: Flexible File-System Interfaces to Storage-Class Memory)"
EuroSys 2016	"HiNFS (A High Performance File System for Non-Volatile Main Memory)"
SOSP 2017	"NOVA (NOVA-Fortis: A Fault-Tolerant Non-Volatile Main Memory File System)"
SOSP 2017	"Strata (Strata: A Cross Media File System)"
HotStorage 2019	"EvFS (EvFS: User-level, Event-driven File System for Non- volatile Memory)"
FAST 2019	"Orion (Orion: A Distributed File System for Non-Volatile Main Memory and RDMA-Capable Networks)"
FAST 2019	"Ziggurat (Ziggurat: A Tiered File System for Non- Volatile Main Memories and Disks)"
SOSP 2019	"ZoFS (Performance and Protection in the ZoFS User-space NVM File System)"
SOSP 2019	"SplitFS (SplitFS: Reducing Software Overhead in File Systems for Persistent Memory)"
FAST 2021	"KucoFS (Scalable Persistent Memory File System with Kernel-Userspace Collaboration)"
Linux kernel	"DAX (Ext4-DAX, XFS-DAX)"

But…

- **PM only**
  - PM as end destination media
  - Replace traditional storage?
    - Exception: Strata and Ziggurat
- **Lengthy process to maturity**
  - E.g., Ext4…still in progress
  - Wisdom with age

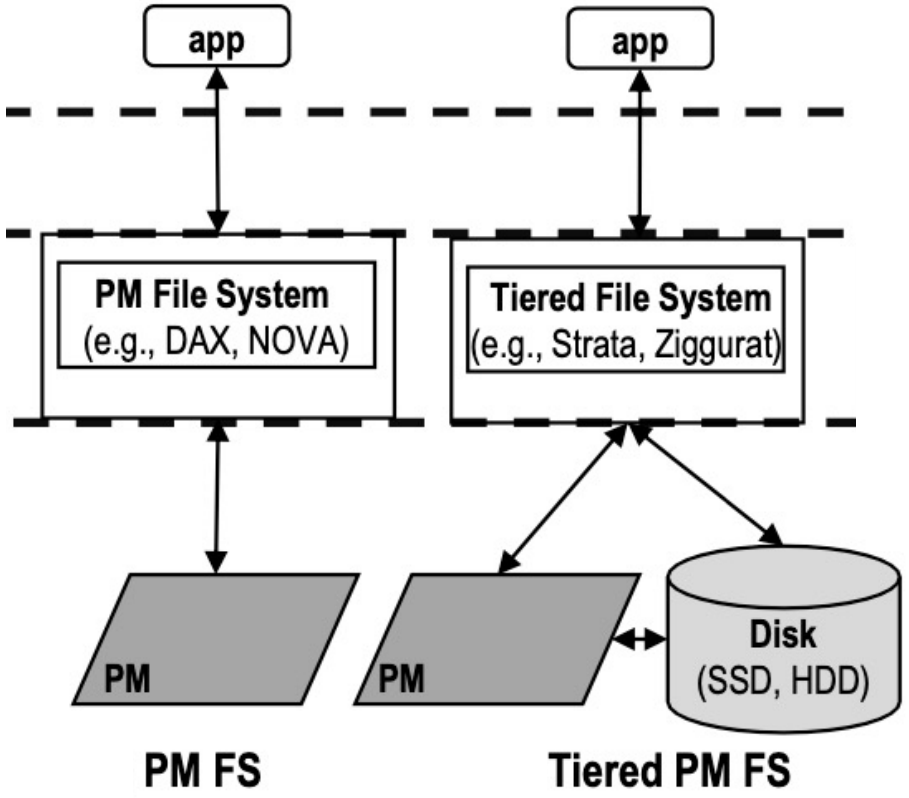
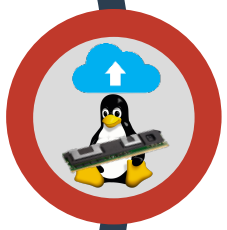
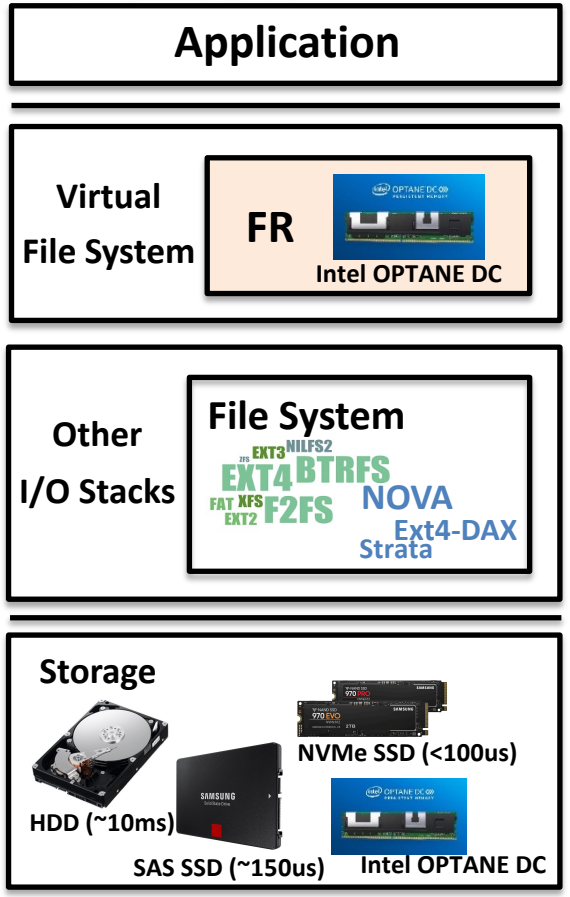


TABLE OF CONTENTS



First Responder (FR)

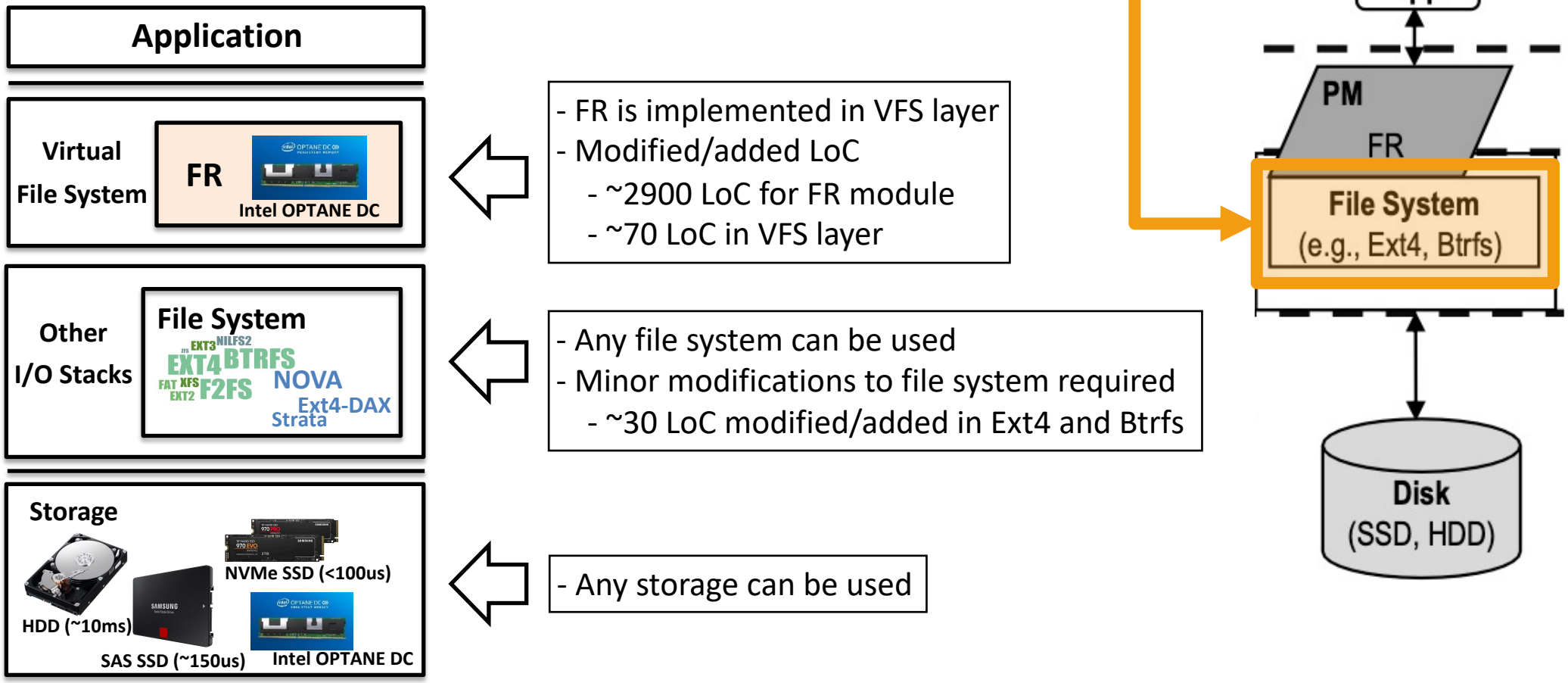
- **PM-based cache-like layer**



- FR not only acts as a cache, but also as a storage using its persistent properties

Goals

- **Keep legacy file system and storage media "as-is”**



Goals

- **PM performance**
  - Lightweight static management

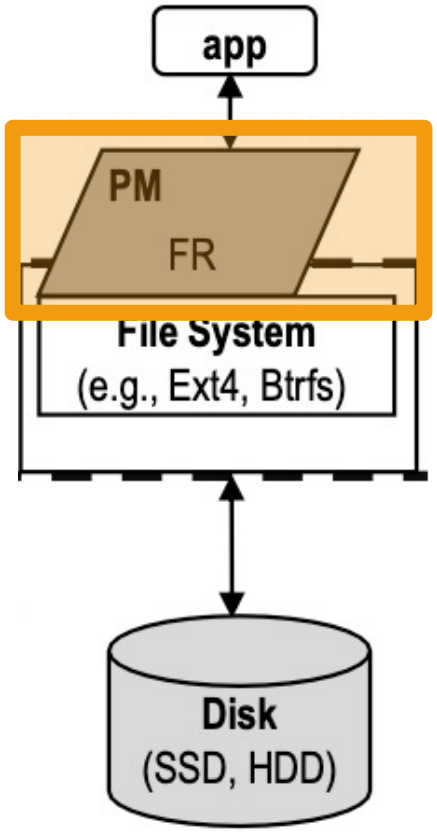
*** Average latency for managing cache for various indexing and management policies**

Our static indexing	**67ns**
Hashing indexing	**75ns**

vs.

Activity		Radix-tree + LRU	Hash + LRU
Radix-tree / Hash	Hashing	-	78ns
	Search	35ns	162ns
	Insert*	19μs	19μs
	Delete	190ns	43ns
LRU	Touch	161ns	153ns
	Add	73ns	65ns
	Remove	88ns	82ns

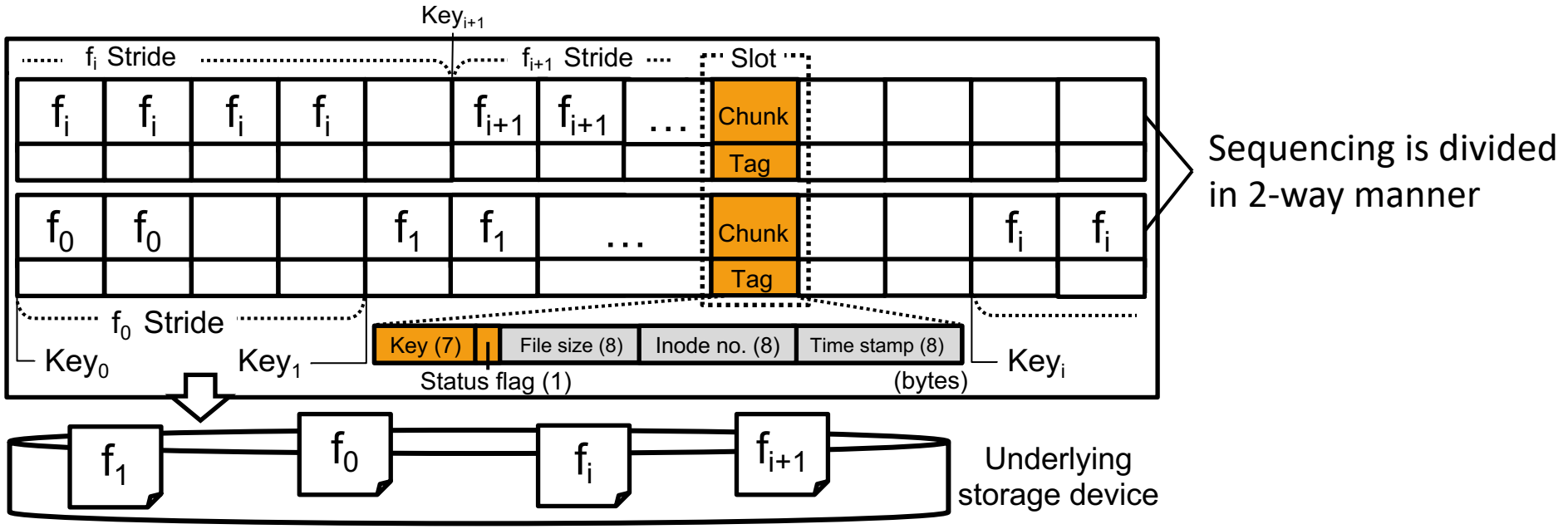
* Insert includes mechanism to find empty blocks



- **Ensure durability/consistency**
  - Static protocol naturally fulfills this

Design

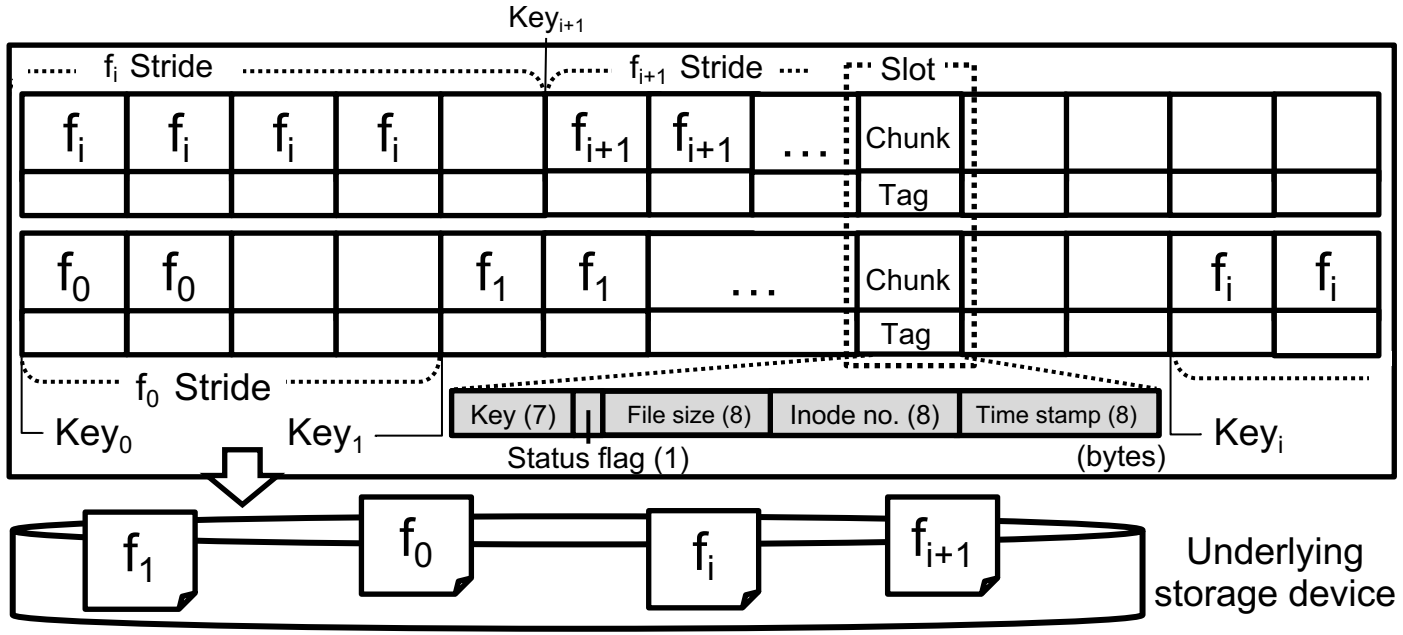
- **Internal components of FR**



- Chunk: Actual data is stored
- Tag: Some file's information and the status of chunk are stored
  - Key used for indexing: ***key*** mod Floor(N/2)
  - Bits in Status flag: V (Valid), N (New)

Design

Layout of FR

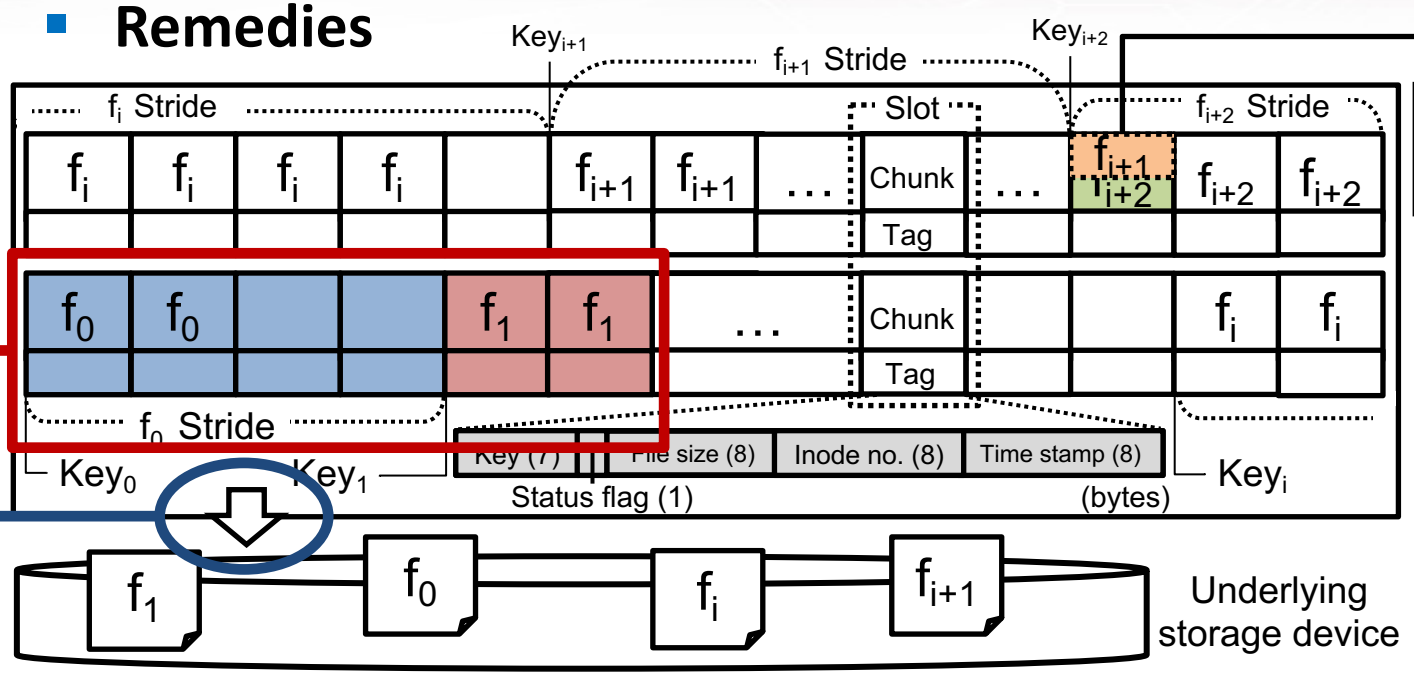


- Static placement/replacement scheme in FR
  - Every files have destined location within FR with no PM allocator

    → Can result in higher miss rate and collision

Design

- **Remedies**



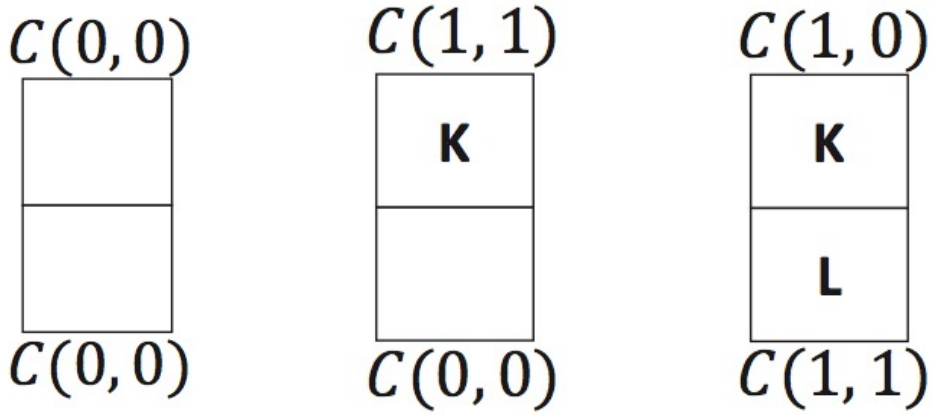
*__Collision__ occurs when data from files invades each other's chunk beyond the stride value

- Sufficient large FR (>> working set)
- Stride: To eliminate invasion in chunk as much as possible
  - Files are positioned apart from each other by stride
- Periodic Flush: To reduce penalty (for clean chunk, there is no penalty for collision)
  - Data is written to chunk, is flushed in the background

Data consistency protocol

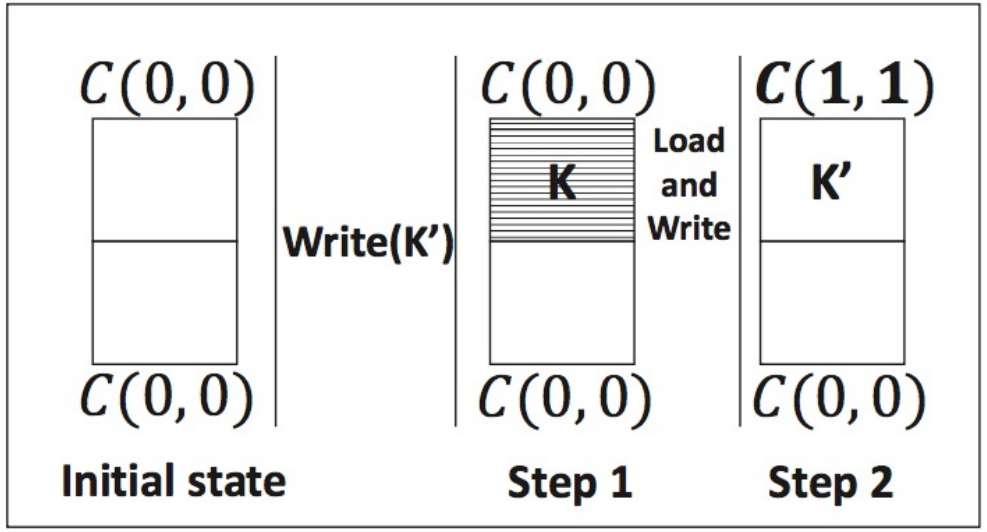
- **Indexing**
  - ***Key*** mod N → ***key*** mod Floor(N/2)

- **Cases**
  - S[C(0, 0), C(0, 0)]: the slot is empty
  - S[C(1, 1), C(0, 0)]: only one chunk has valid data
  - S[C(1, 1), C(1, 0)]: both chunks have valid data



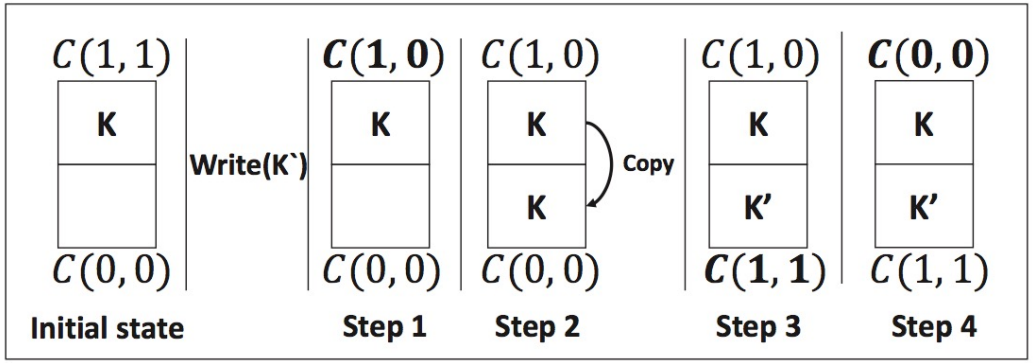
Data consistency protocol

- **Case 1**

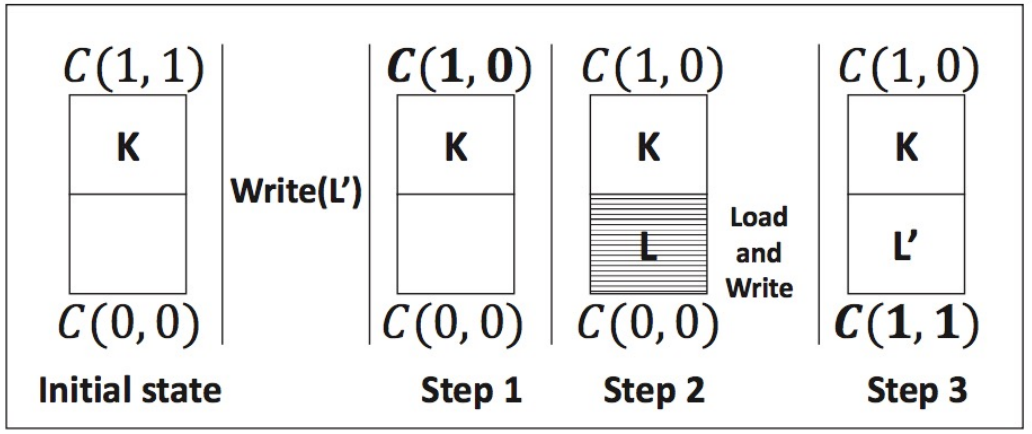


Data consistency protocol

- **Case 2**



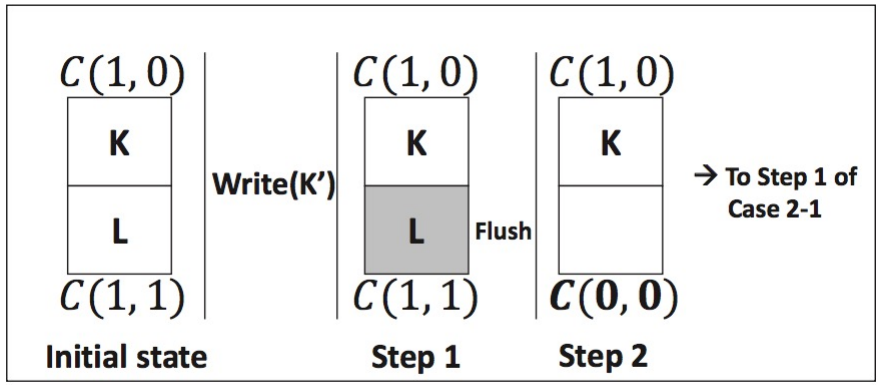
(1)



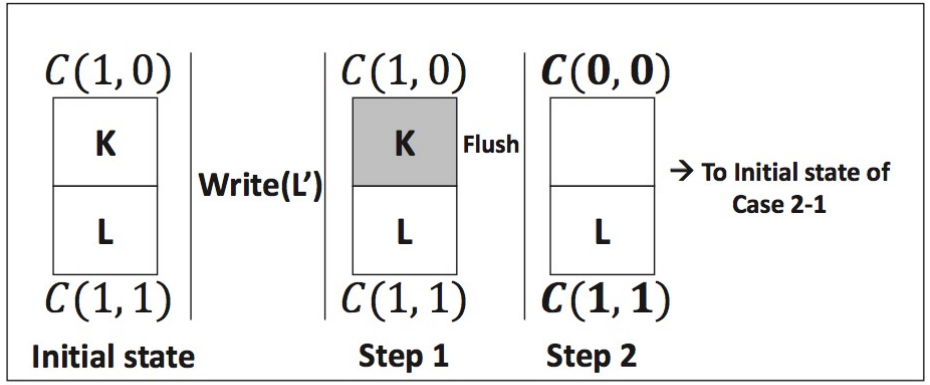
(2)

Data consistency protocol

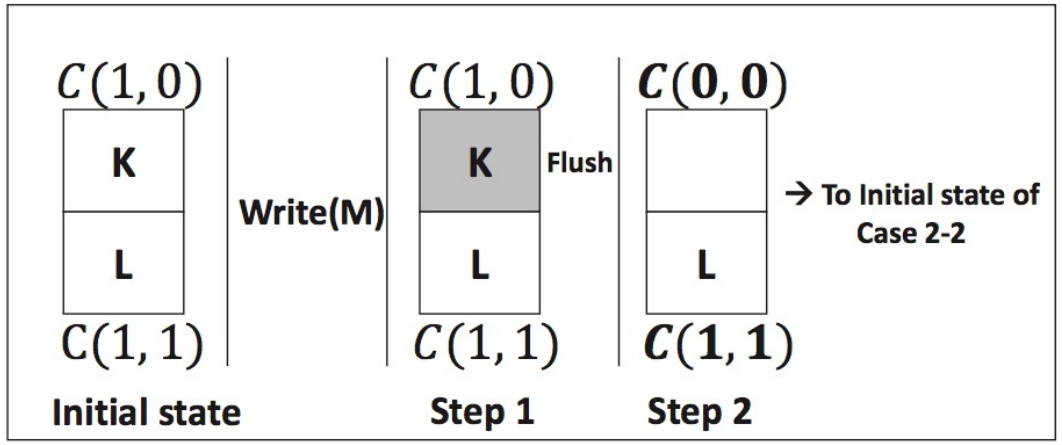
- **Case 3**



(1)



(2)



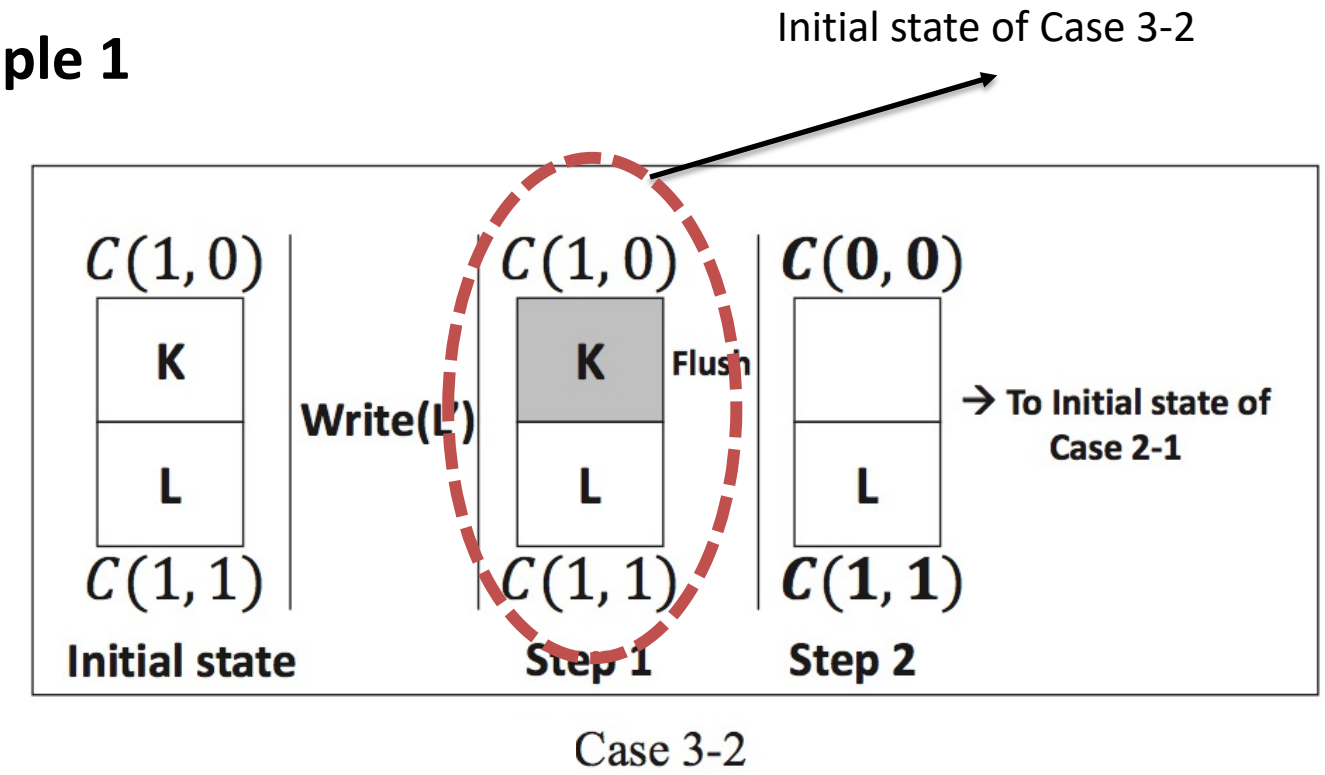
(3)

Failure recovery in FR

- **Key point**
  - If a fault occurs at any step, the recovered state will either be one of the initial states of the figures or have already completed the intended write

Failure recovery in FR

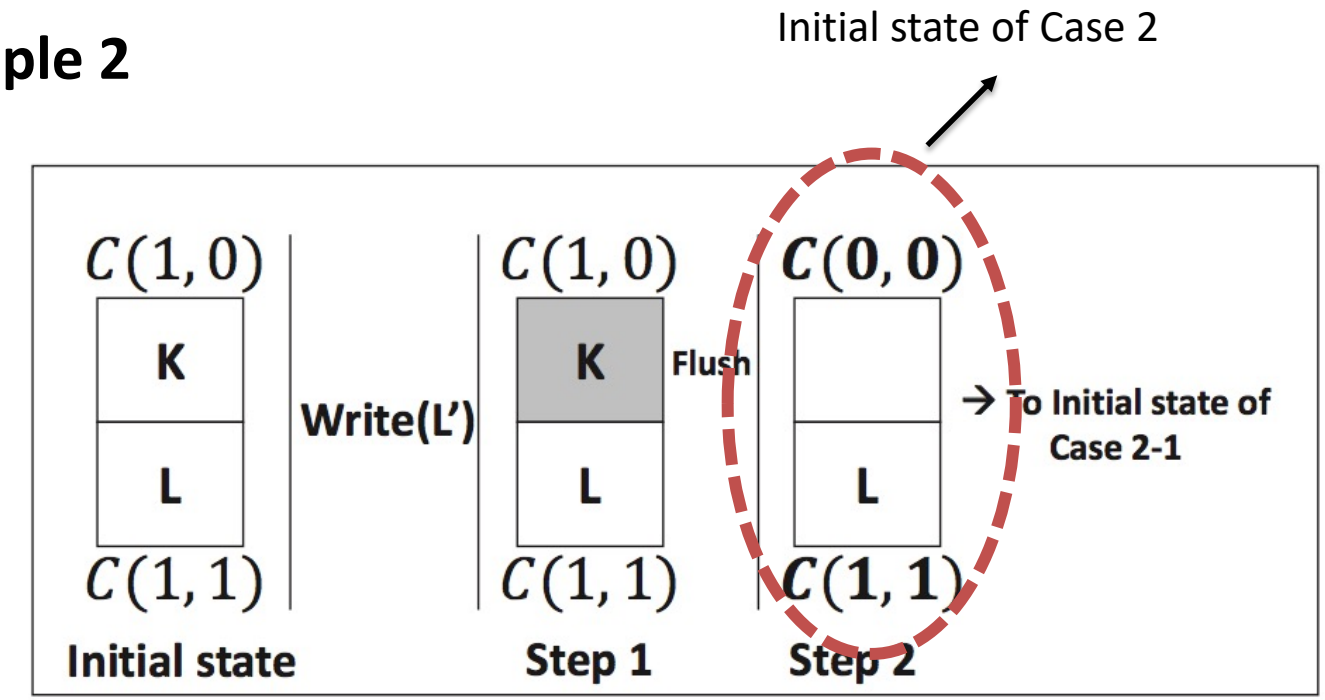
- **Example 1**



Case 3-2

Failure recovery in FR

- **Example 2**

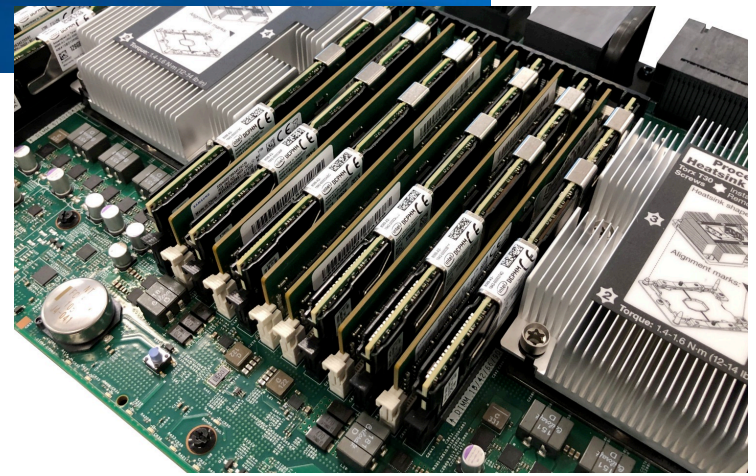


Case 3-2

Performance evaluation

- **System configuration**

	Description
CPU	Intel(R) Xeon(R) Gold 6242 CPU @ 2.80GHz (16 cores)
DRAM	Samsung 32GB 2666MHz DDR4 RDIMM×4 (128GB)
PM	Intel Optane DC Persistent Memory (128GB)
Storage	Samsung V-NAND SSD 860 EVO (1TB)
OS	Linux Ubuntu 18.04.3 LTS (64bit) kernel v4.18



- **Description of experimental comparison**

Notation	Description	Configuration		
		PM	DRAM	Backing storage
FR-X	FR applied to Ext4 (X is period value, e.g., 10ms)	128GB		1TB (SSD)
Ext4	Traditional block-based file system		128GB	1TB (SSD)
DM-WC	DM-Writecache applied to Ext4	128GB	128GB	1TB (SSD)
DAX	PM-aware file system developed based on Ext4			128GB (PM)
NOVA	PM-aware file system			128GB (PM)

Standard workloads

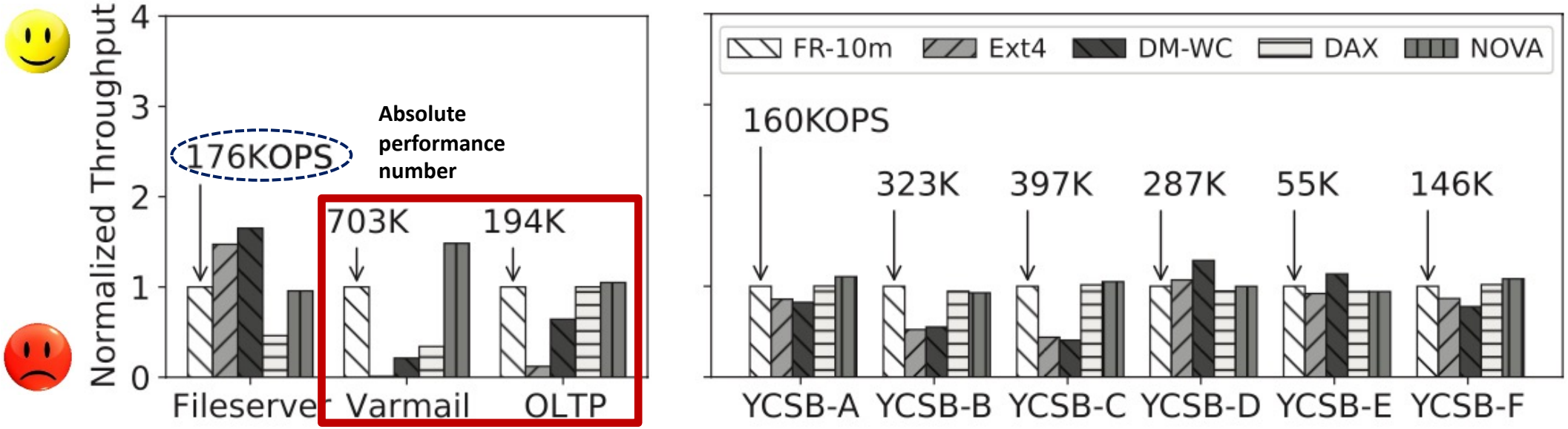
- **Benchmarks**

Filebench	R:W	Mean file size	# of files	**Key-value store**	Data set size	R:W
Fileserver	1:2	128KB	200K	YCSB-A,-F	4GB	1:1
Varmail	1:1	32KB	800K	YCSB-B,-D,-E	4GB	19:1
OLTP	1:1	1.5GB	20	YCSB-C	4GB	1:0

  - Filebench
    - Fileserver: write-intensive workload *without* fsync() calls
    - Varmail and OLTP: have considerable number of fsync() calls
  - YCSB (record selection for -D is Latest, while all others are Zipfian)
    - Application: RocksDB
    - -A, -F: write-intensive workloads
    - -C: read-only workload
    - -B, -D, -E: read-intensive workloads

Standard workloads

- **Overall performance**

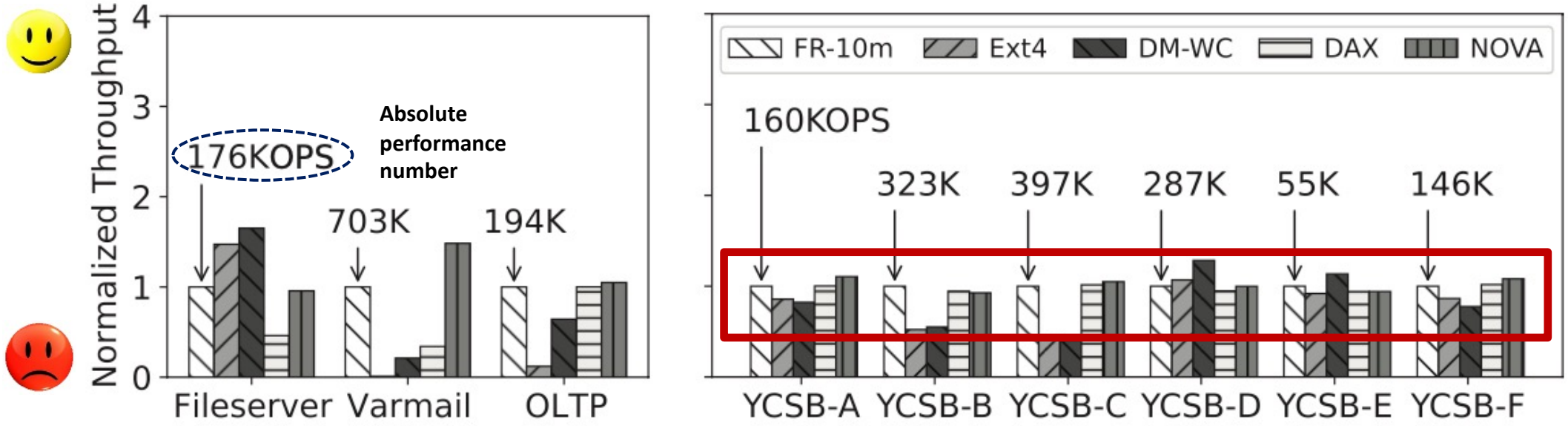


Observations

- [Filebench] For Varmail and OLTP, FR is roughly 94x and 8x better than Ext4 and roughly 4.5x and 1.5x better than DM-WC
  FR performance is slightly lower than NOVA, while DAX suffers for Varmail
- [YCSB] FR, NOVA, and DAX show similar performance
  Ext4 and DM-WC perform worst for YCSB-A, -B, and -C

Standard workloads

- **Overall performance**

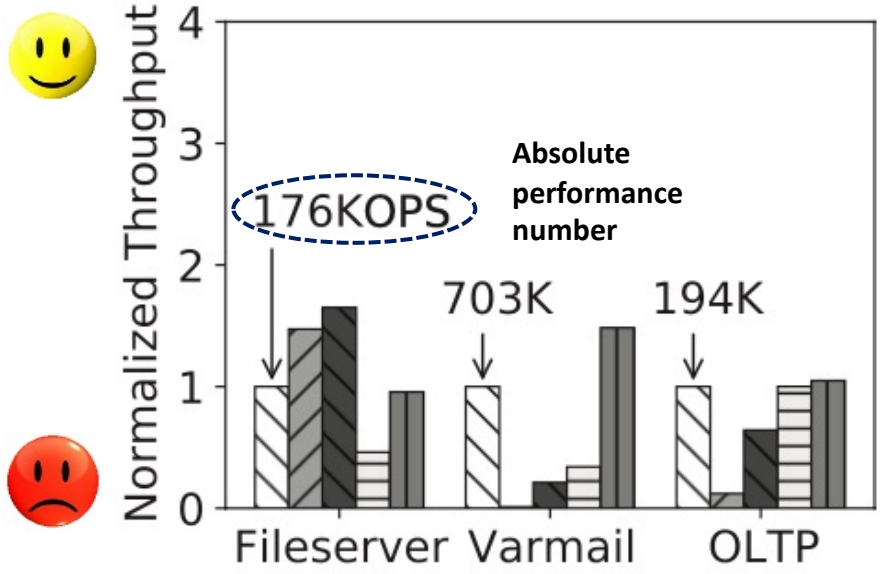


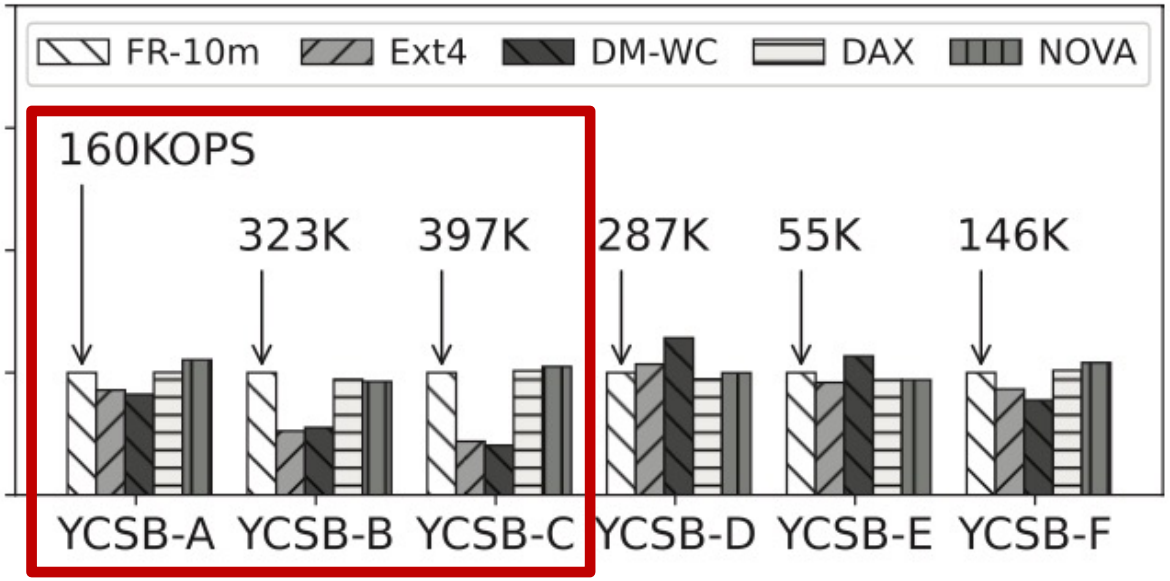
Observations

- [Filebench] For Varmail and OLTP, FR is roughly 94x and 8x better than Ext4 and roughly 4.5x and 1.5x better than DM-WC
  FR performance is slightly lower than NOVA, while DAX suffers for Varmail
- [YCSB] FR, NOVA, and DAX show similar performance
  Ext4 and DM-WC perform worst for YCSB-A, -B, and -C

Standard workloads

- **Overall performance**



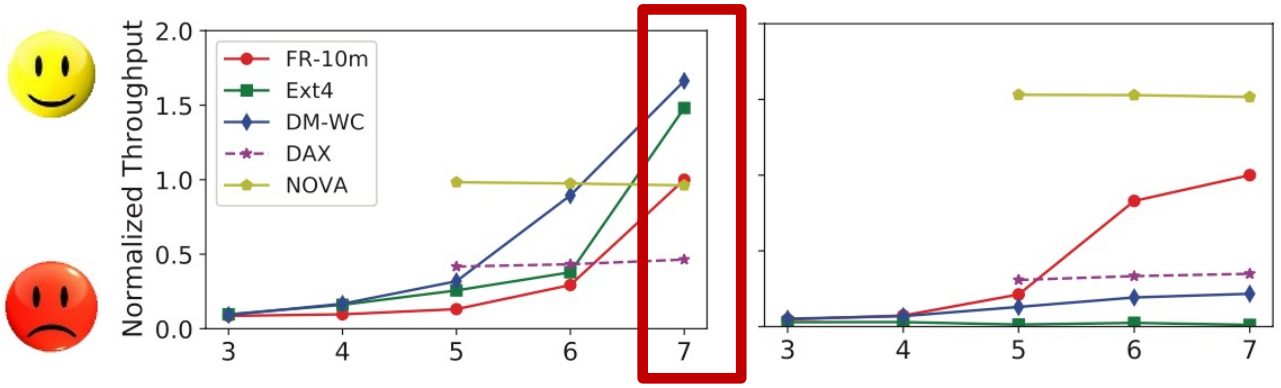


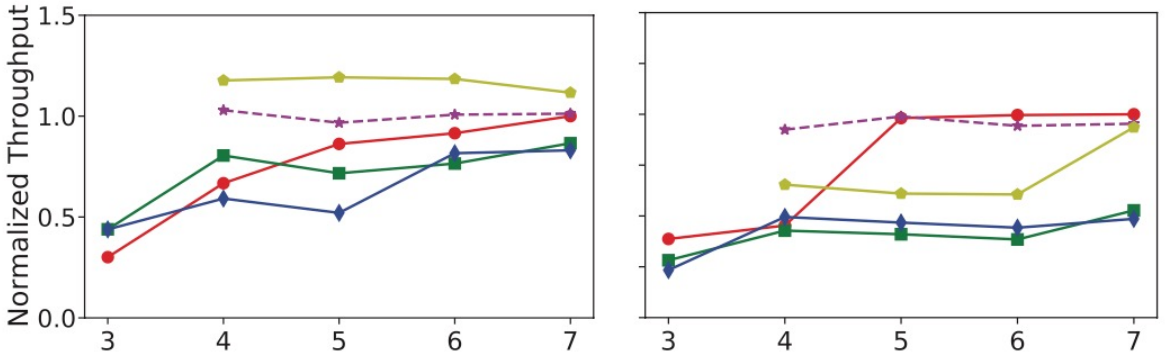
Observations

- [Filebench] For Varmail and OLTP, FR is roughly 94x and 8x better than Ext4 and roughly 4.5x and 1.5x better than DM-WC
  FR performance is slightly lower than NOVA, while DAX suffers for Varmail
- [YCSB] FR, NOVA, and DAX show similar performance
  Ext4 and DM-WC perform worst for YCSB-A, -B, and -C

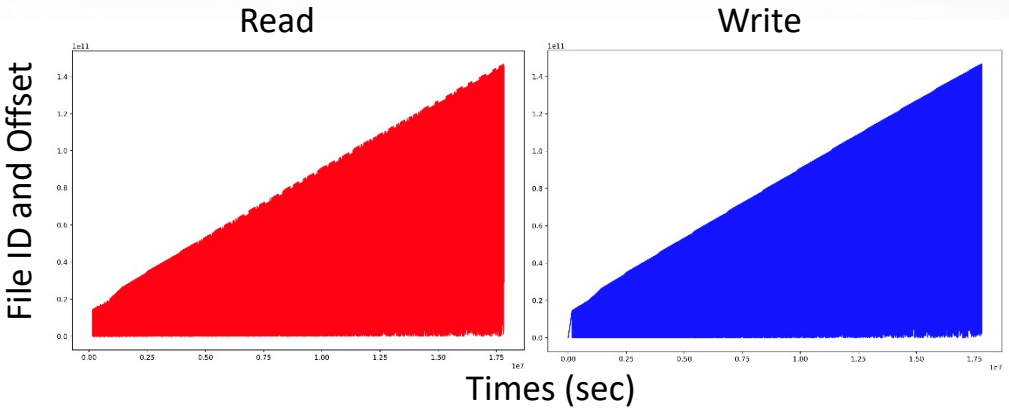
Effect of PM size

- **Performance results for PM size of 2^xGB, where *x* is value of points in *x*-axis**
  - Normalized to the performance of FR when *x* = 7 (128GB)





Dynamic workload



Working set of standard workload (Fileserver)

- **Limitations of standard workloads**
  - Standard workloads do not capture the dynamics of real-world workloads
    - In terms of pattern
      - Standard workloads: Working set does not change with time
      - Real-world workloads: Working set grows and shrinks as time evolves
    - In terms of operation generation
      - Standard workloads: No change in the access intensities of working set over time
      - Real-world workloads: Access intensities are also vary with time

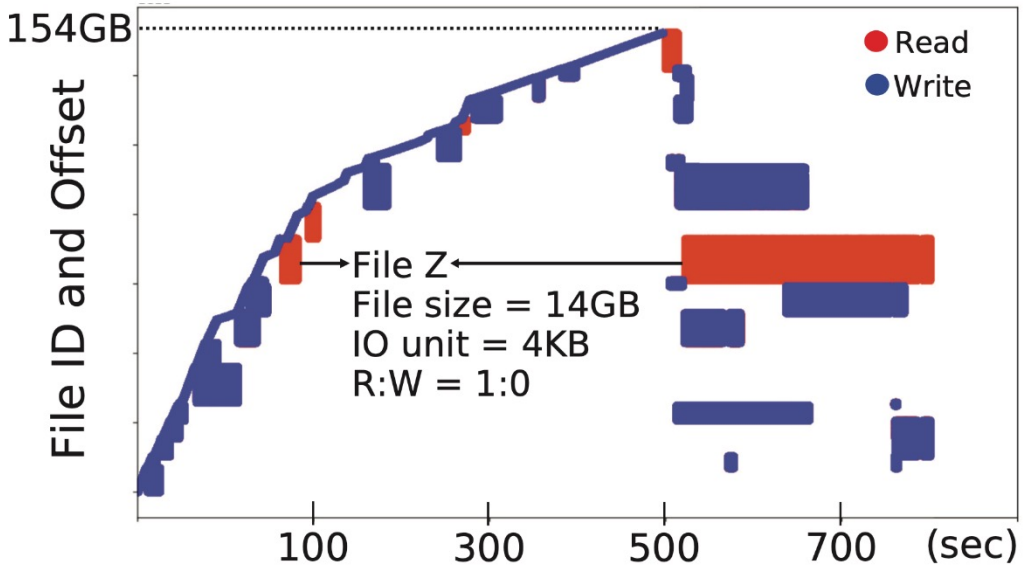
→ Need for **dynamic workload** that is more representative of real-world workloads

Dynamic workload

- **We devise synthetic workloads using I/O testing tool FIO**

	Description
File size	1∼14GB [whole numbers only: 1 to 2 files of each]
Distribution	Pareto 0.1/0.5 [2/2], Zipf 0.2/1.2 [5/1], N [6], R [5]
IO unit	4KB [14], 8KB [7]
Read:Write	1:0 [4], 19:1 [9], 1:1 [8]
Fsync	0∼5% [whole numbers only: 2 to 5 files of each]
Intensity	Request interval: 1μs∼1ms (randomly distributed)

(a) Characteristics of 21 files used to generate synthetic workload (FIO)

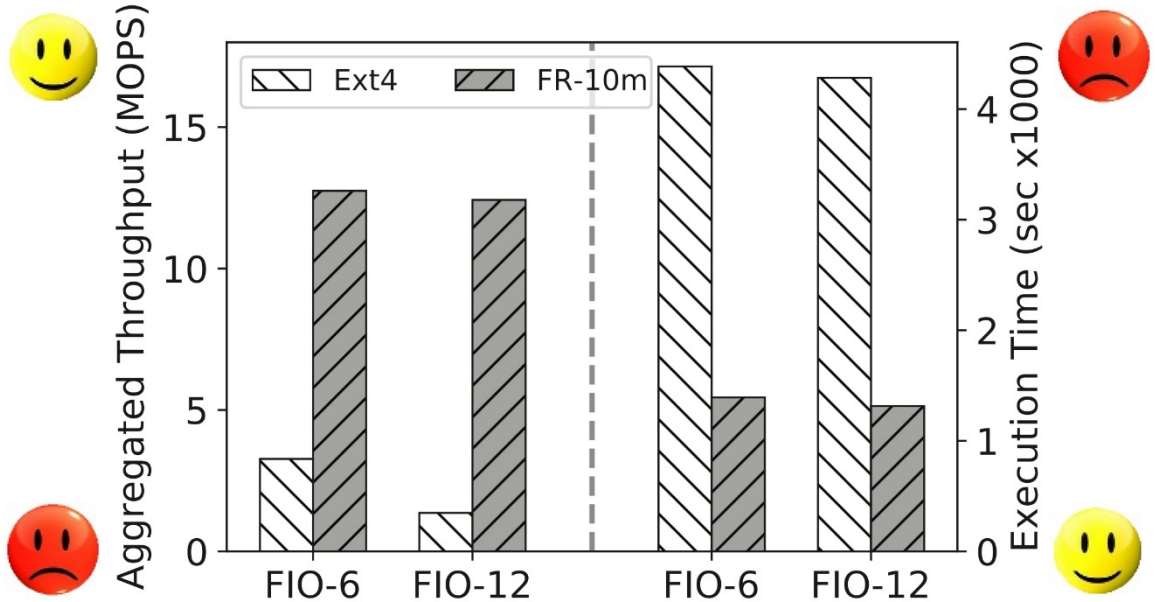


(b) Working set (FIO-12)

Configuration: Total IO size is 575GB
- FIO-6: 6 files, 50GB working set
- FIO-12: 12 files, 100GB working set

Dynamic workload

- **Performance results**



Observations

- FR provides more than 9x higher aggregate throughput and ended over 3x faster than Ext4
- FR is providing immediately durable in-order semantics
- For NOVA and DAX, cannot run as dataset is larger than PM size

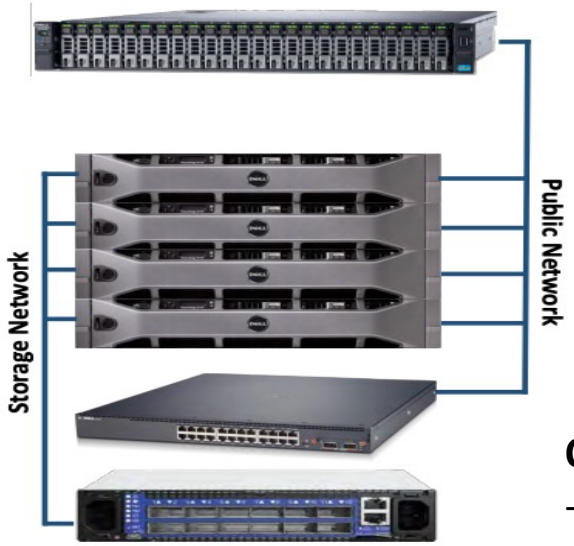
TABLE OF CONTENTS

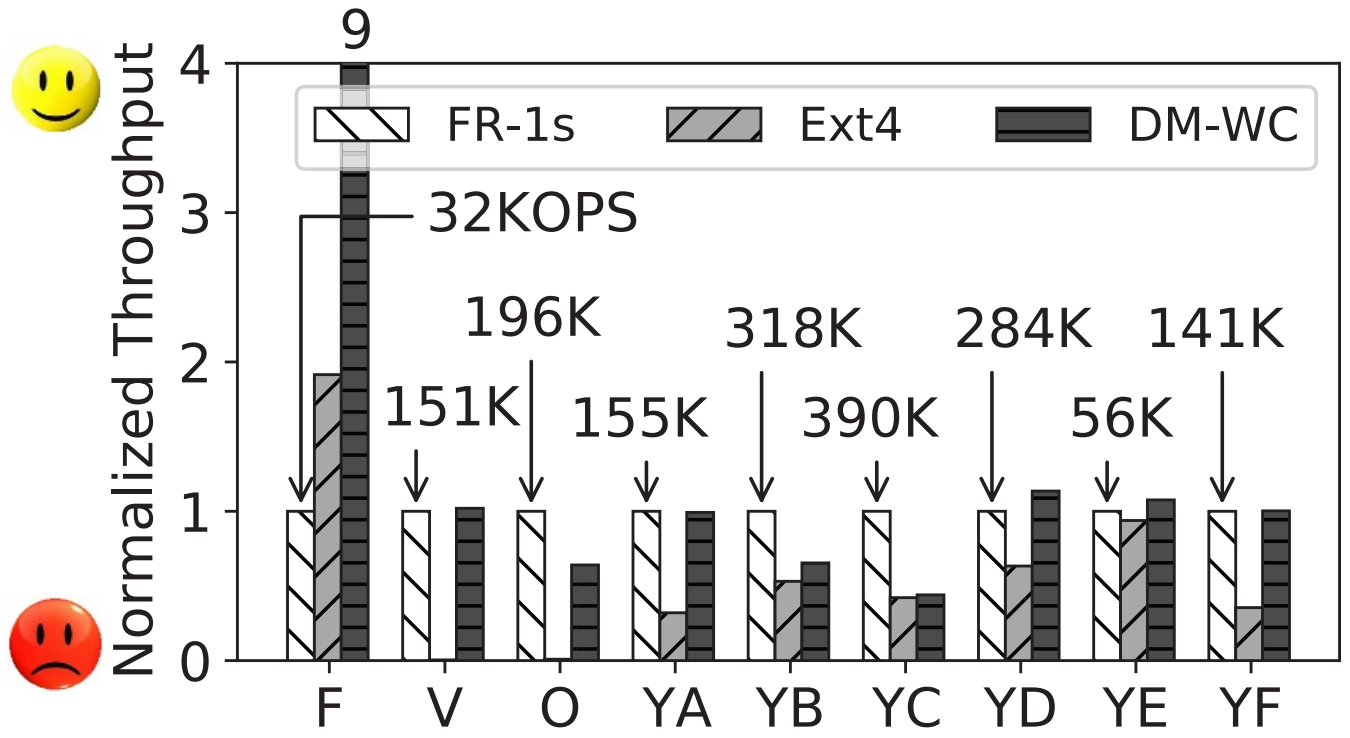
FR on other storage platforms

Step 1: FR on Ceph client

- **FR is applied to the client server running 3 OSDs as block device**

	Description
CPU	Intel(R) Xeon(R) Gold 6242 CPU @ 2.80GHz (16 cores)
DRAM	Samsung 32GB 2666MHz DDR4 RDIMM×4 (128GB)
PM	Intel Optane DC Persistent Memory (512GB)
Storage	Samsung V-NAND SSD 860 EVO (1TB)
OS	Linux Ubuntu 18.04.3 LTS (64bit) kernel v4.18



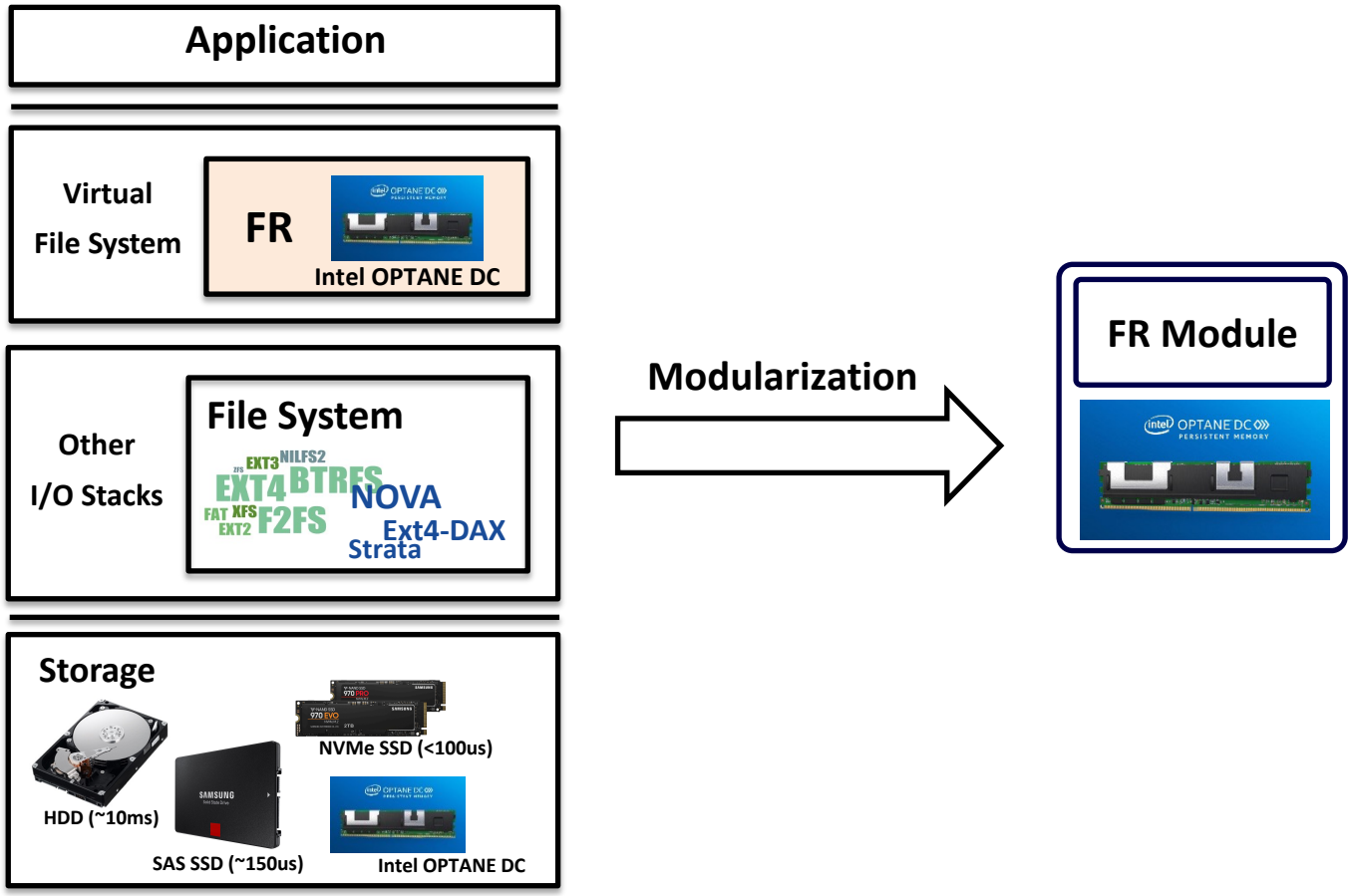


Observations

- Media performance: NVMe SSD > SATA SSD > network-connected OSDs > HDD
- FR and Ext4 have performance changes depending on the media performance
- DM-WC still shows similar performance

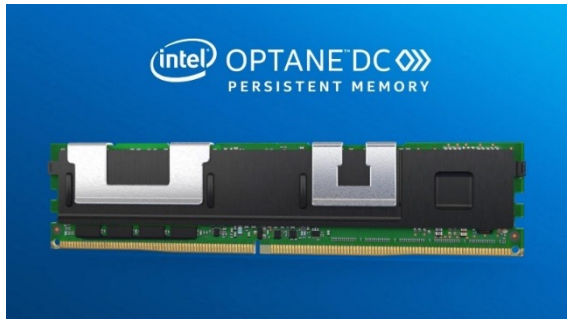
Step 2: FR on Ceph OSD

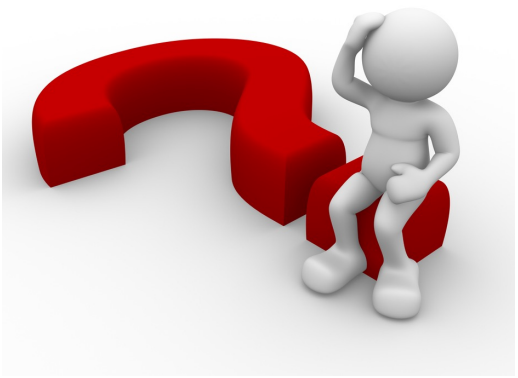
- **FR module can be applied everywhere!!!**



Step 2: FR on Ceph OSD

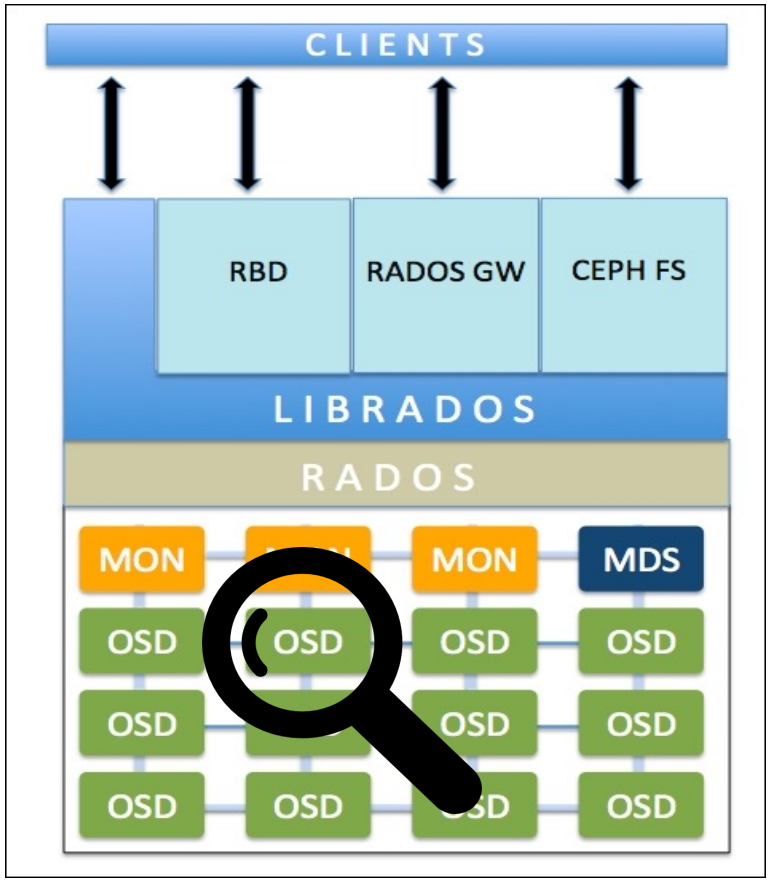
- **What if you apply FR to Ceph?**
  - High performance
  - High reliability





Step 2: FR on Ceph OSD

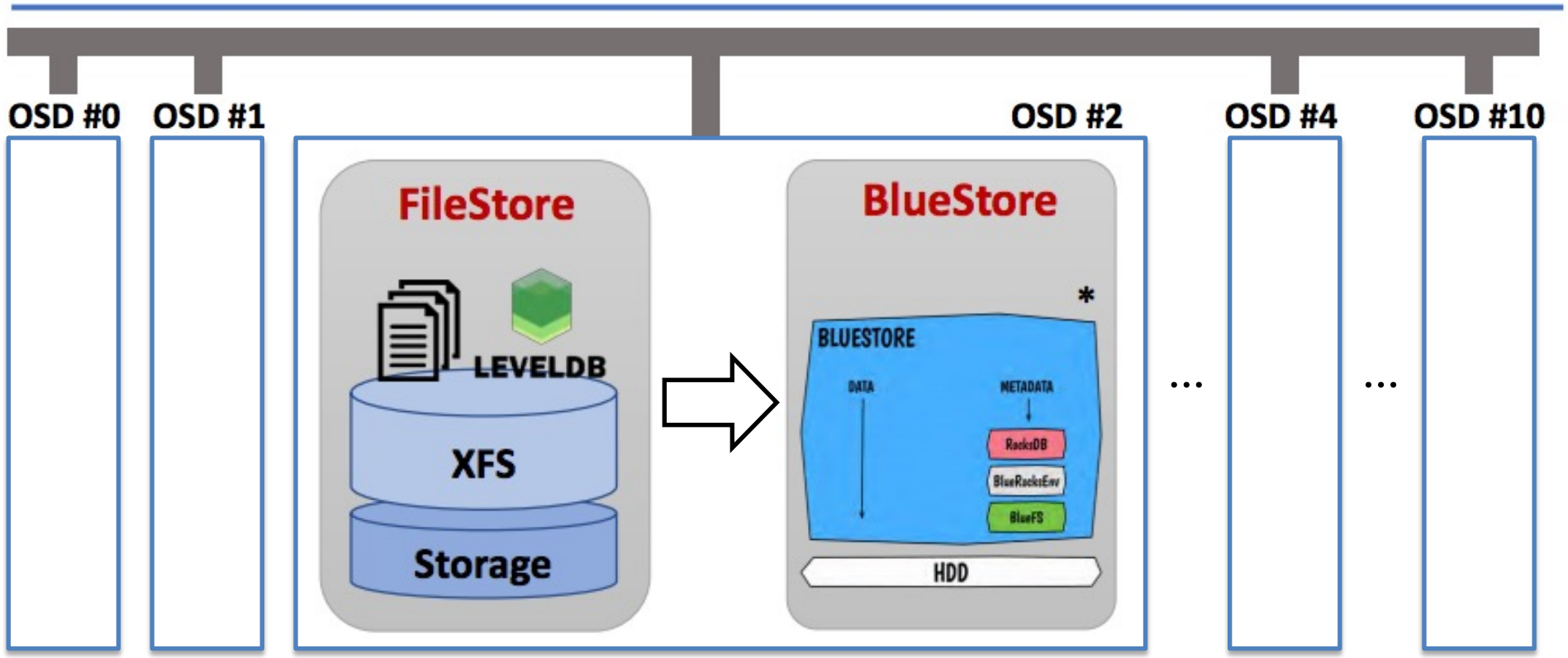
- **Application point**
  - Object Storage Daemon (OSD)



Step 2: FR on Ceph OSD

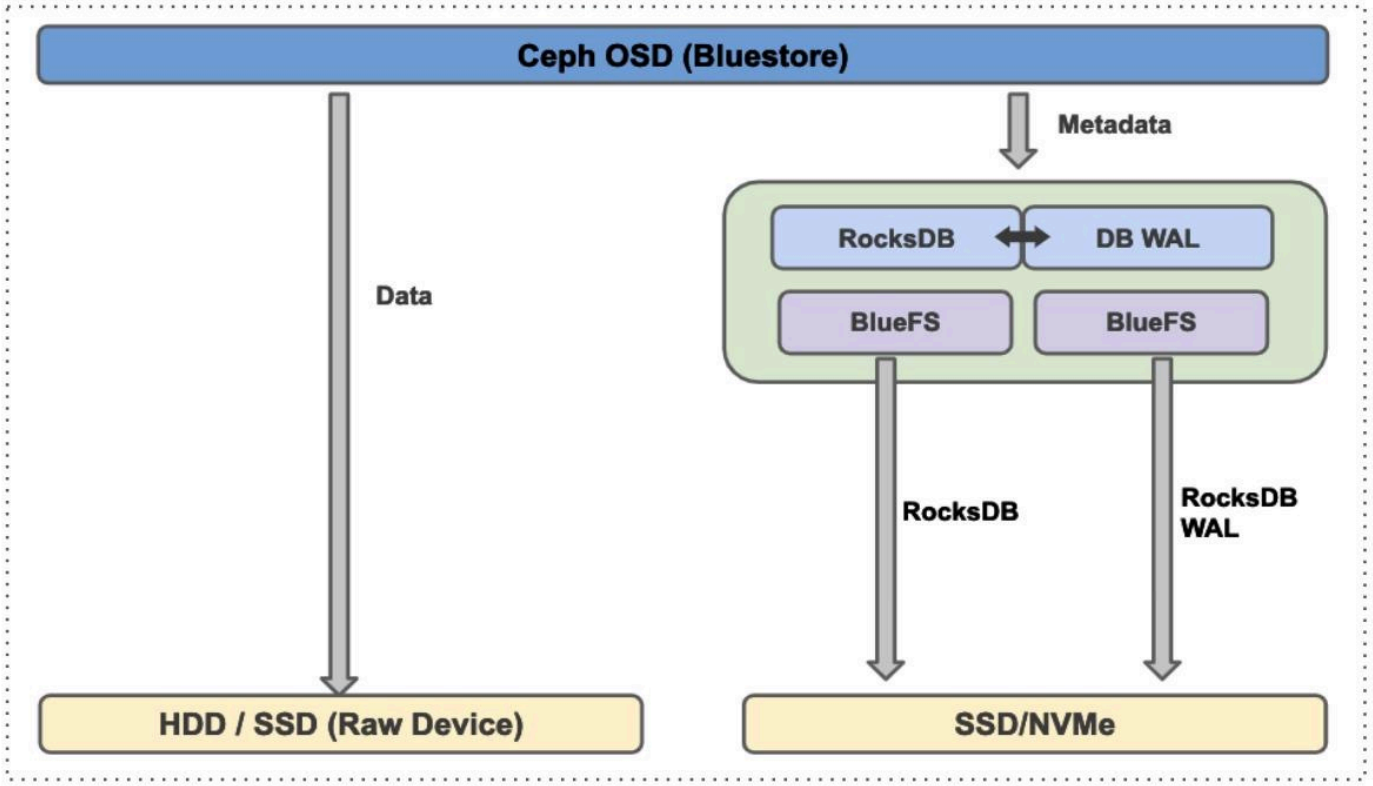
* “File Systems Unfit as Distributed Storage Backends: Lessons from 10 Years of Ceph Evolution,” In Proc. SOSP’19

- **Architecture of Ceph RADOS Block Device (RBD)**



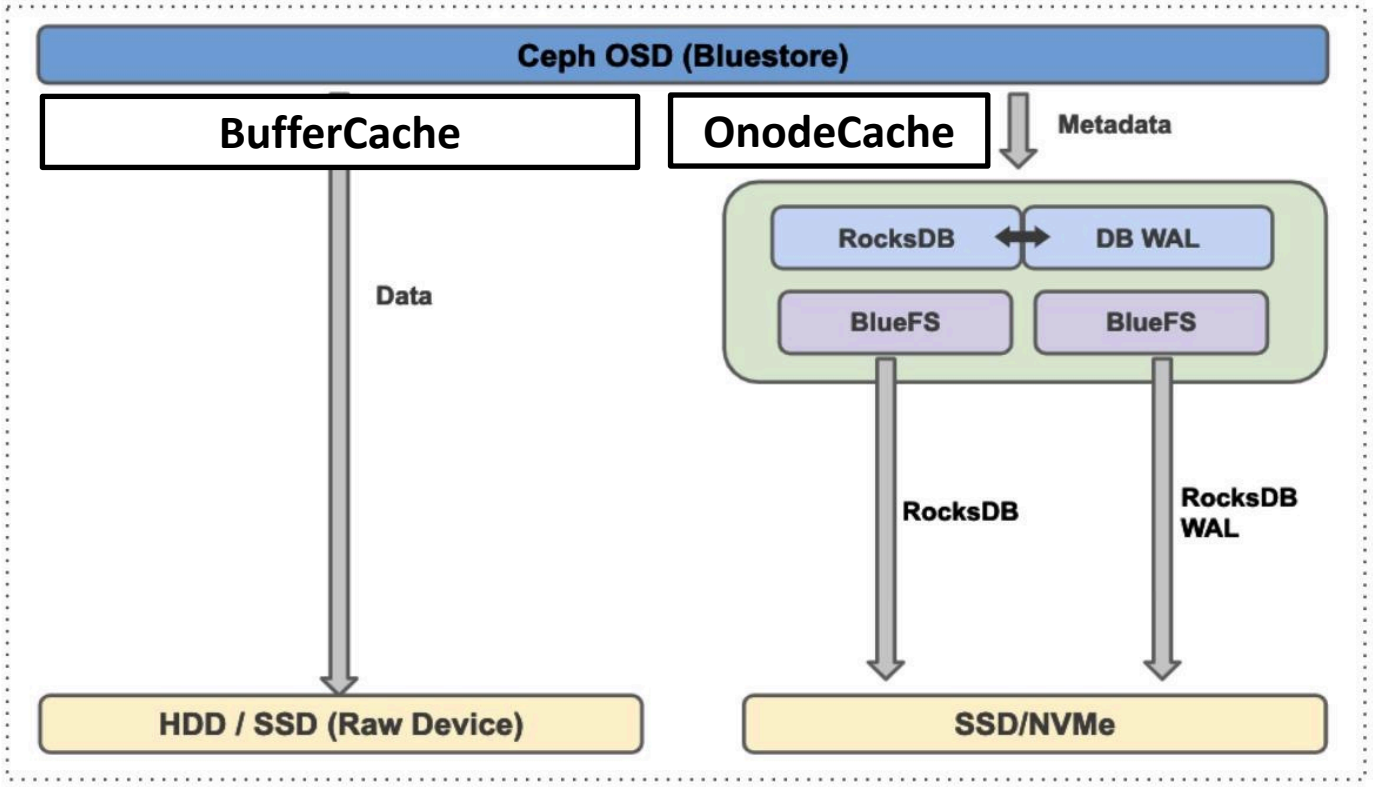
Step 2: FR on Ceph OSD

- **Application point: "BlueStore"**
  - OSD -> Storage backend (BlueStore)



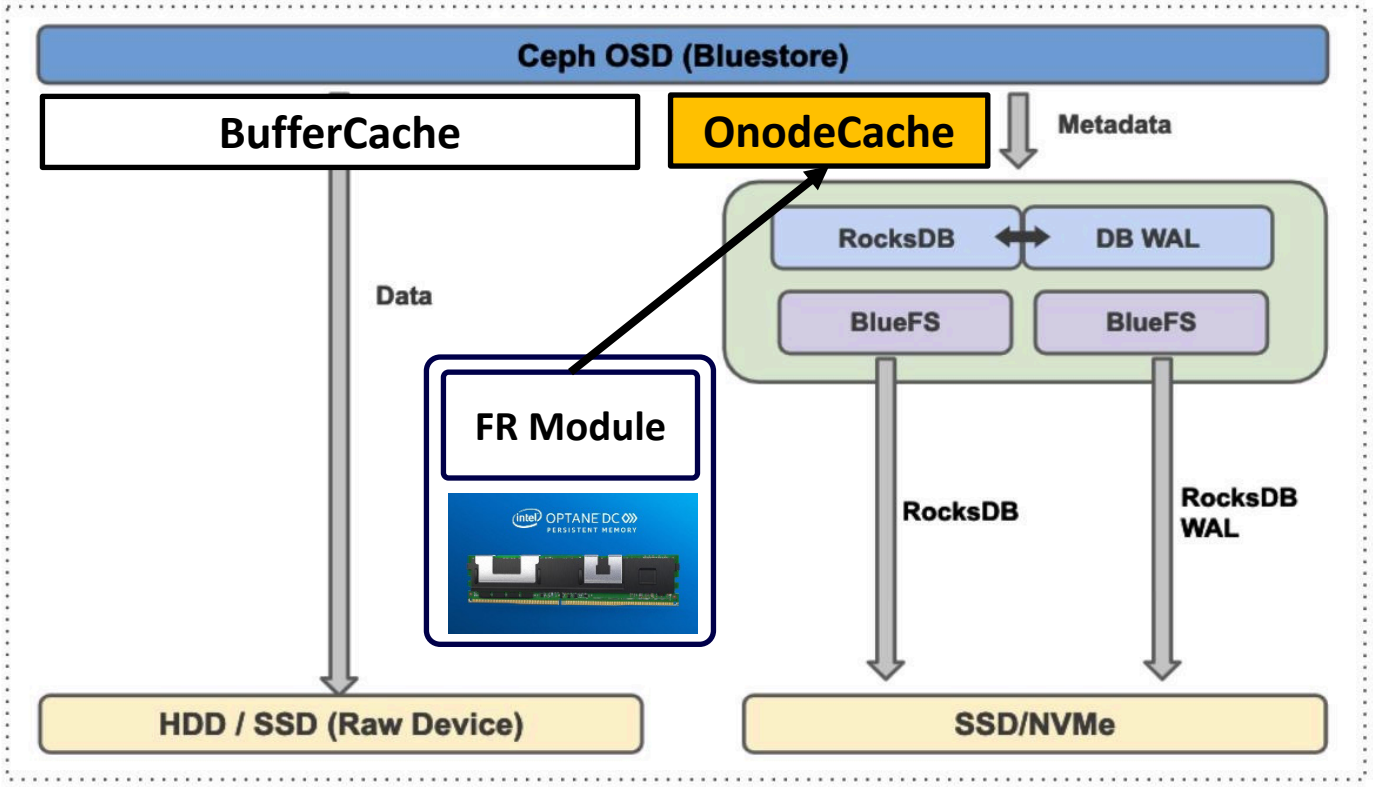
Step 2: FR on Ceph OSD

- **Cache in BlueStore**
  - BlueStore's own write-through cache in user space
  - Uses the 2Q and LRU algorithm



Step 2: FR on Ceph OSD

- **FR in BlueStore**
  - Critical path can be performed with PM performance



Step 2: In progress

- **FR on Ceph**
  - Analysis of target
    - BlueStore, OnodeCache code
  - Implementation of FR
    - Modularizing of FR in user-level (C → C++)
    - Allocation user-level PM pool
    - Applying FR to BlueStore
  - Debugging
  - Validation of FR effect
  - Devising ways to overcome shortcoming

TABLE OF CONTENTS

Summary

- **First Responder (FR)**
  - PM-based cache-like layer
  - Keep legacy storage system and storage media "as-is”
  - PM performance
    - Respond quickly with in-order semantics
    - Hide traditional I/O stack overhead
  - Ensure durability/consistency
    - Protocol implemented with static management

Thank you!!!