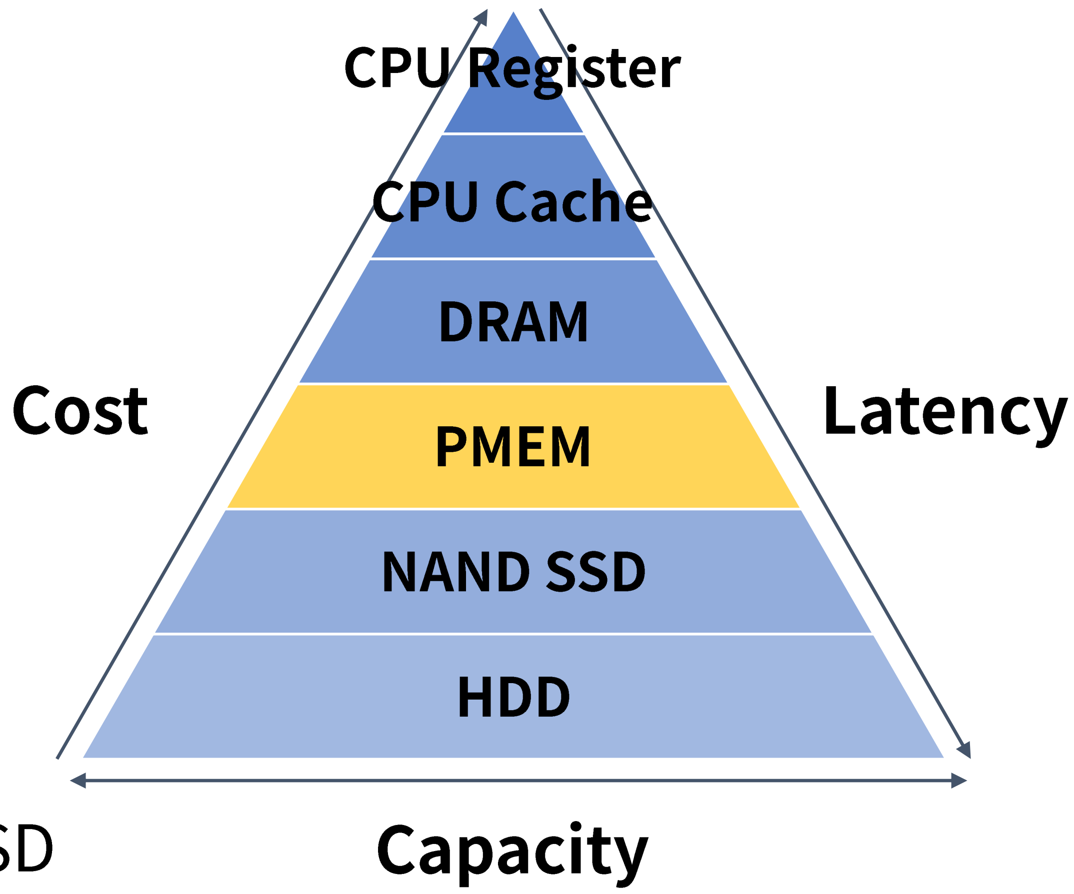
SPFS: On Stacking a Persistent Memory File System on Legacy File Systems

Hobin Woo[1], Daegyu Han[2], Seungjoon Ha[1], Sam H. Noh[3,4], Beomseok Nam[2]

Samsung Electronics[1], SungKyunKwan University[2], UNIST[3], Virginia Tech[4]

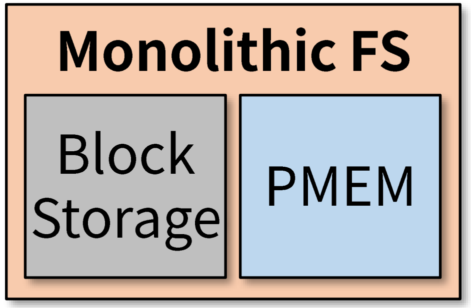
Background

- **Persistent Memory (PMEM)**
  - Low access latency
  - Byte-addressability
  - Persistency
- **Intel® Optane™ DCPMM**
  - First commercialized PMEM product
  - High-capacity and low latency
  - Intermediate layer between DRAM and SSD
  - Killed in 2022
- **Alternatives Products: NVDIMM, MRAM, CXL, …**



Motivation

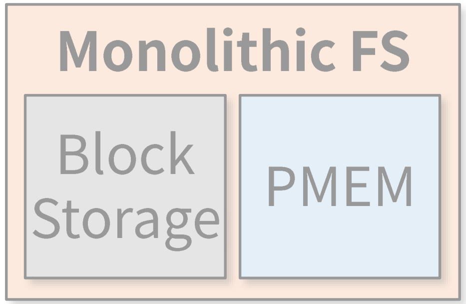
- **File systems for tiered storage – PMEM and conventional block storage**
  - High performance
  - Low-cost capacity



- **Ziggurat, Strata:** Monolithic file system
  - Limitations of managing all types of storage in a single file system
    - Reinventing features of mature file systems (VFS cache, I/O scheduling, LFS, ⋯)
    - Complexity of handling multiple types of storage
    - Impractical deployment

Motivation

- **File systems for tiered storage – PMEM and conventional block storage**
  - High performance
  - Low-cost capacity



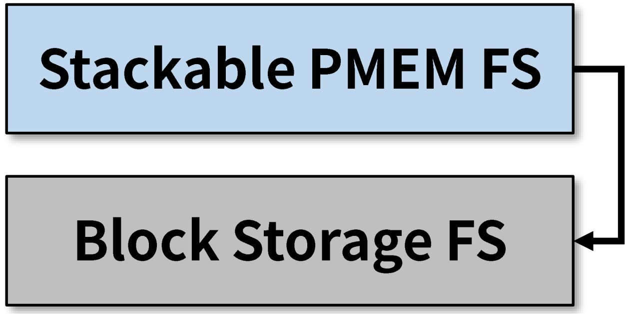
- **Ziggurat, Strata:** Monolithic file system
  - Limitations of managing all types of storage in a single file system

Q: Can we reuse file systems that have been improved for decades?

  - Reinventing features of mature file systems (VFS cache, I/O scheduling,

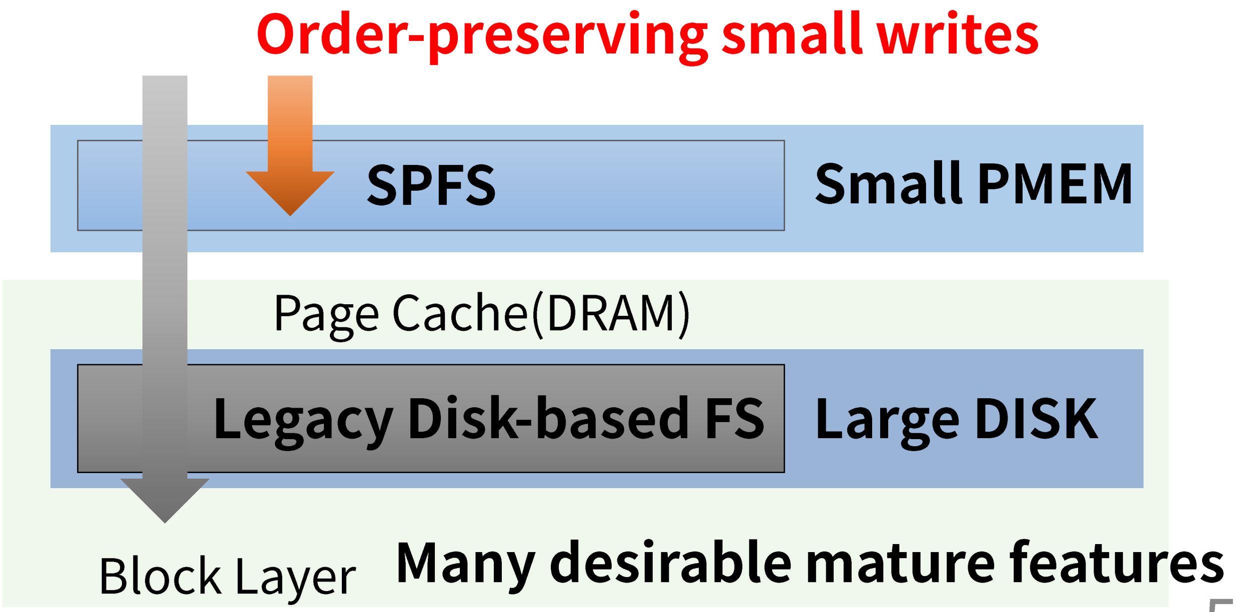
A: Stackable File System: modular, practical

  - Complexity of handling multiple types of storage
  - Impractical deployment



SPFS

- SPFS (Stackable Persistent Memory File System*)
  - **Modular approach** to storage tiering
  - Provide PMEM as **persistent write-cache** to PMEM-oblivious file systems
- **SPFS+x : SPFS can be placed on top of any file system x (EXT4, F2FS, XFS,⋯)**
- **Goal:**
  - Absorb small synchronous writes



*: Available at https://github.com/DICL/spfs

SPFS Challenges

- As a stackable file system, SPFS must be lightweight
- SPFS must be effective in classifying and absorbing synchronous writes

Novel Designs of SPFS

- As a stackable file system, SPFS must be **lightweight**

  → Manages all metadata in lightweight and efficient hash tables

  → Novel *Extent Hashing* algorithm

- SPFS must be **effective in classifying and absorbing synchronous writes**

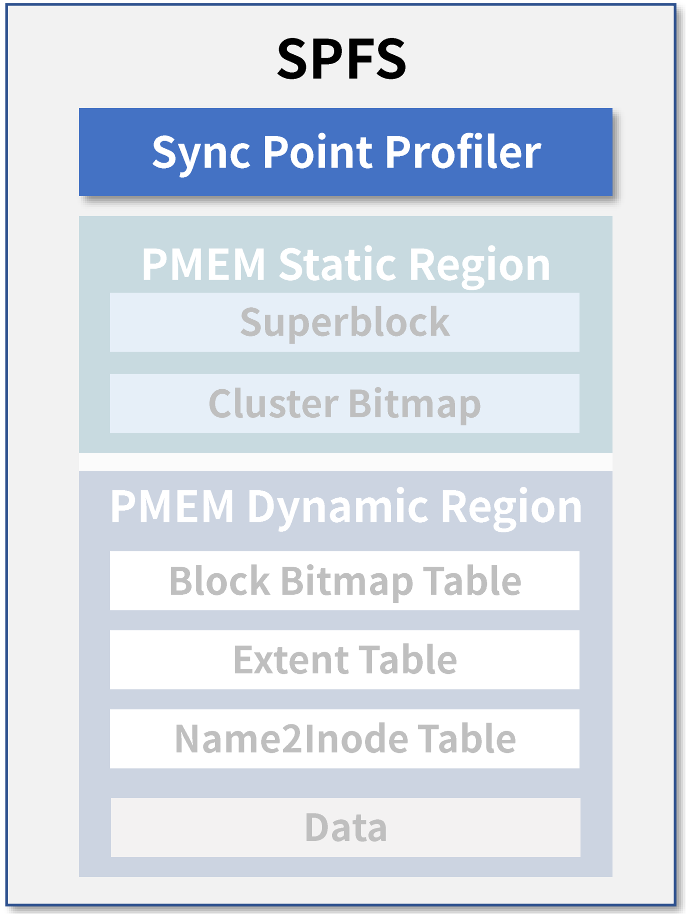
  → Lazy *Sync Point Profiler*

- SPFS+x improves performance of x by ~9.9x for synchronous workloads

Contribution #1

Sync Point Profiler

How is the profiler of SPFS different from the state-of-the-art?



Lazy Sync Point Profiler

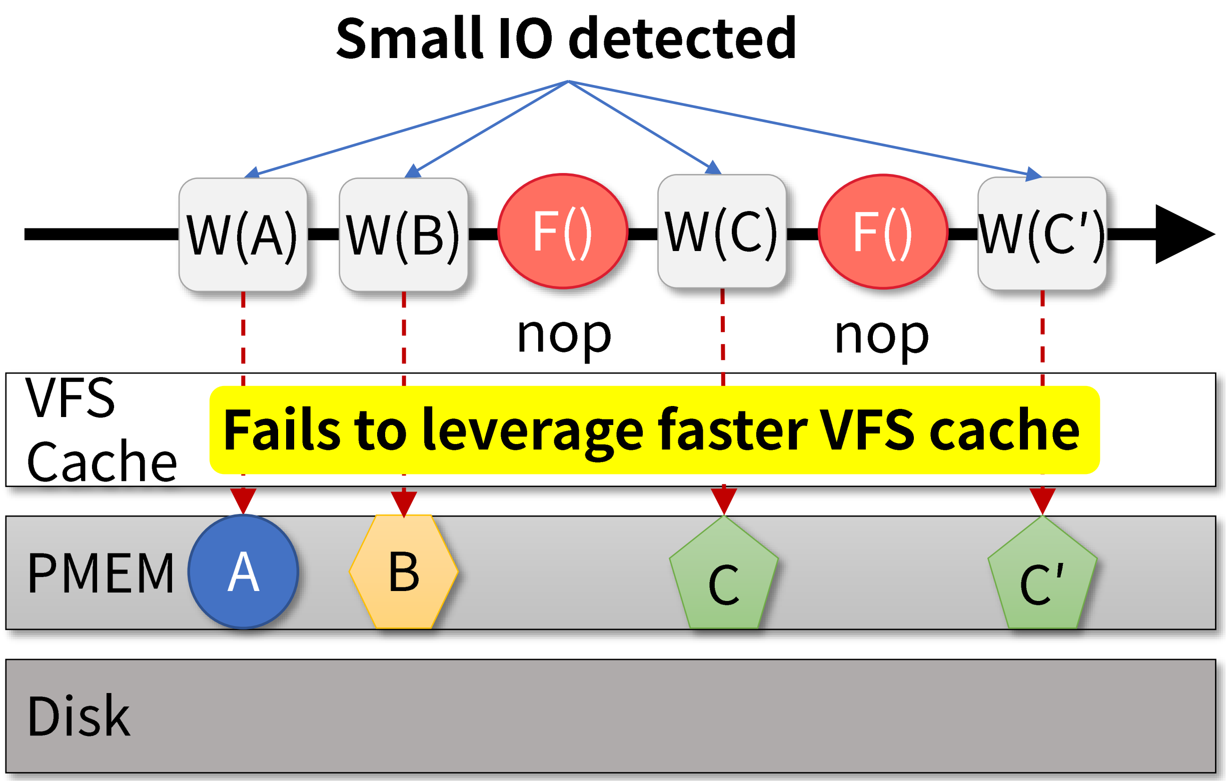
- **Profiler comparison**

	Ziggurat		SPFS	
Profiling Focus	Write size		Synchronicity	
Metric	Individual write size		Bytes written between consecutive `fsync()` calls on the same file	
Tendency	**Eager (*fast-first*)**		**Lazy**	
Workload	**Sync.**	**Async.**	**Sync.**	**Async.**
Large writes	PMEM	Disk	PMEM	Disk
Small writes	PMEM	PMEM	PMEM	Disk

- **VFS cache is better for asynchronous writes**
  - Can not resolve **synchronous small writes**
- **SPFS focuses on synchronicity**

Example: Eager Write Point Profiling vs. Lazy Sync Point Profiling

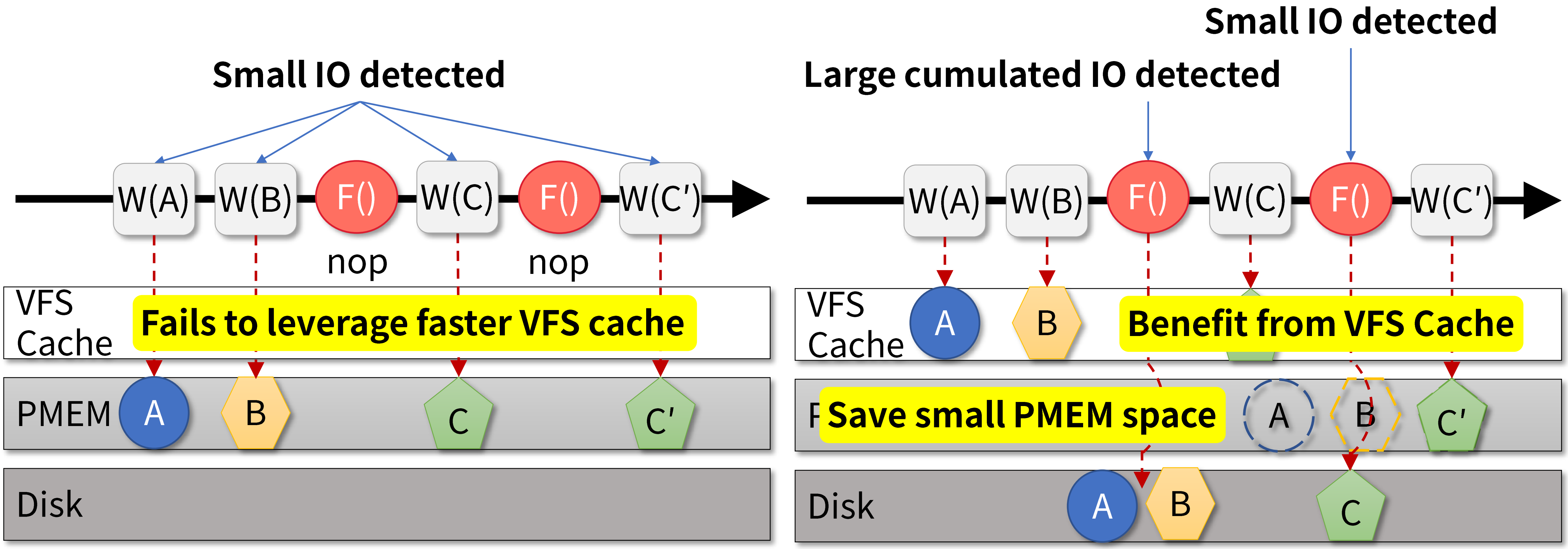
- **Ziggurat**



(a) Write Point Profiler

Example: Eager Write Point Profiling vs. Lazy Sync Point Profiling

- Ziggurat
- SPFS



(a) Write Point Profiler

(b) Sync Point Profiler

Migration to Lower File System

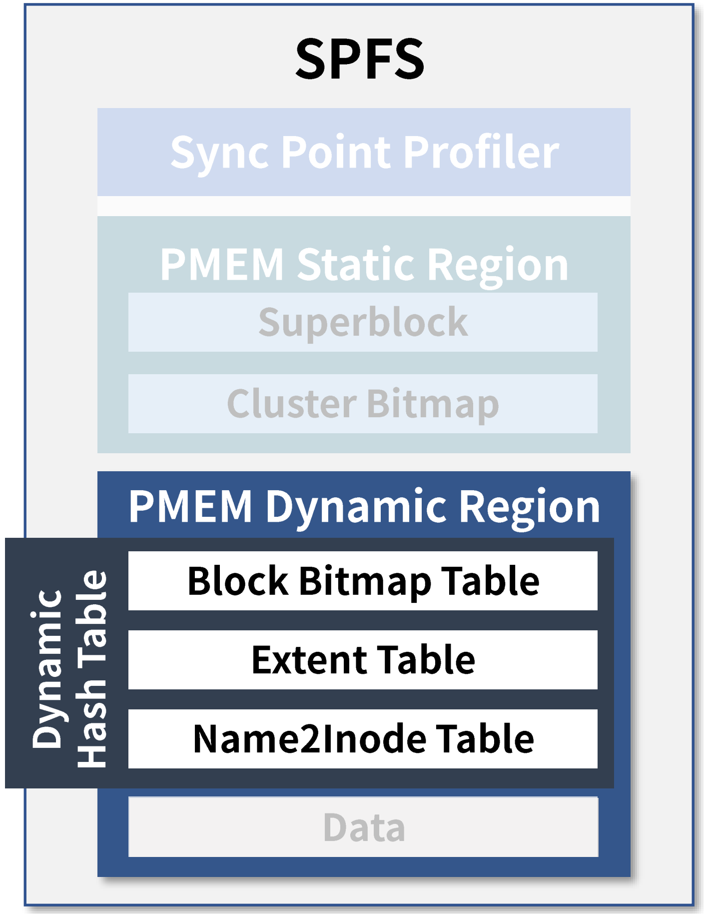
- **Promotion may degrade performance if access pattern changes to**
  - read-intensive
  - large non-transactional writes
- **Sync Factor (SF)**
  - Small value if I/O pattern does not meet the criteria for small sync. writes

$$SF_t = \alpha \cdot weight(IO\_type) + (1 - \alpha) \cdot SF_{t-1}$$

Attenuation factor $(0 < \alpha < 1)$

$\begin{cases} 0 : \text{read-intensive, large update} \\ C : \text{otherwise} \end{cases}$

- **SPFS benefits from VFS cache by demoting read-intensive files to x**



Contribution #2

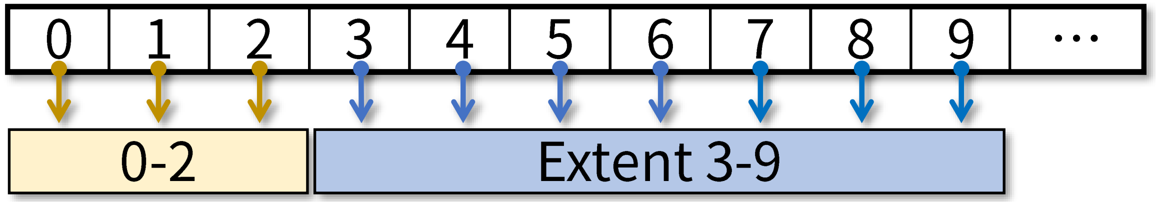
Extent Hashing

SPFS is the first file system that manages
all metadata including extents using a hash table

Extent Hashing

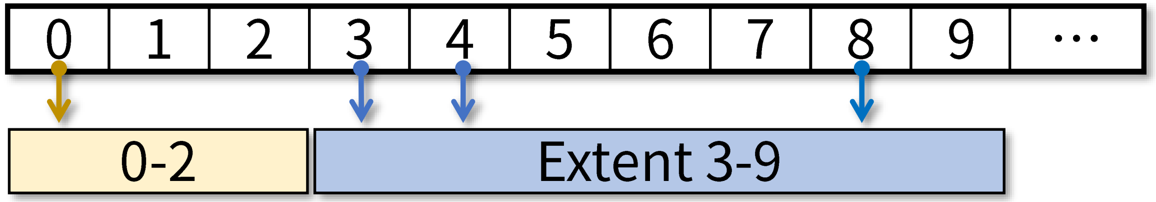
- **Challenge: How to hash extent (i.e., range data)**
  - Suppose extent $E(start\ 3, length\ 7)$, hash function $H(x) = x$

Too many hash entries



Legacy Block Hashing (HashFS)

- **Extent Hashing**
  - Bound the number of entries to $O(\log_2 B)$
    - $O(1)$ for best case

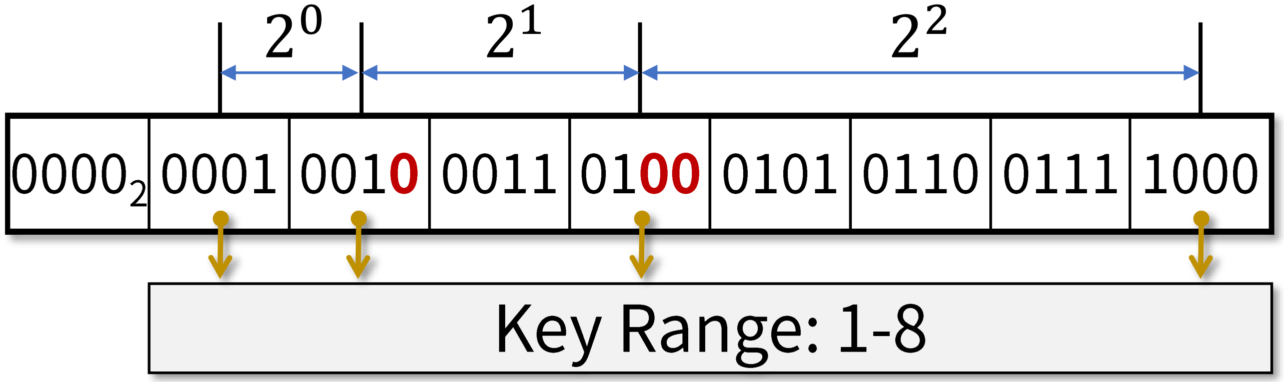


Extent Hashing

Extent Hashing: Insert

- **Insert**
  - Store pointers according to the binary representation of keys in the range
  - ***Stride length***
    - $2^{TNZ(key)}$: Maximum distance to next pointer from the current key
    - $TNZ(x)$
      - Trailing Number of Zero bits of x
      - e.g., $TNZ(11_2) = 0, TNZ(100_2) = 2, TNZ(1010_2) = 1$

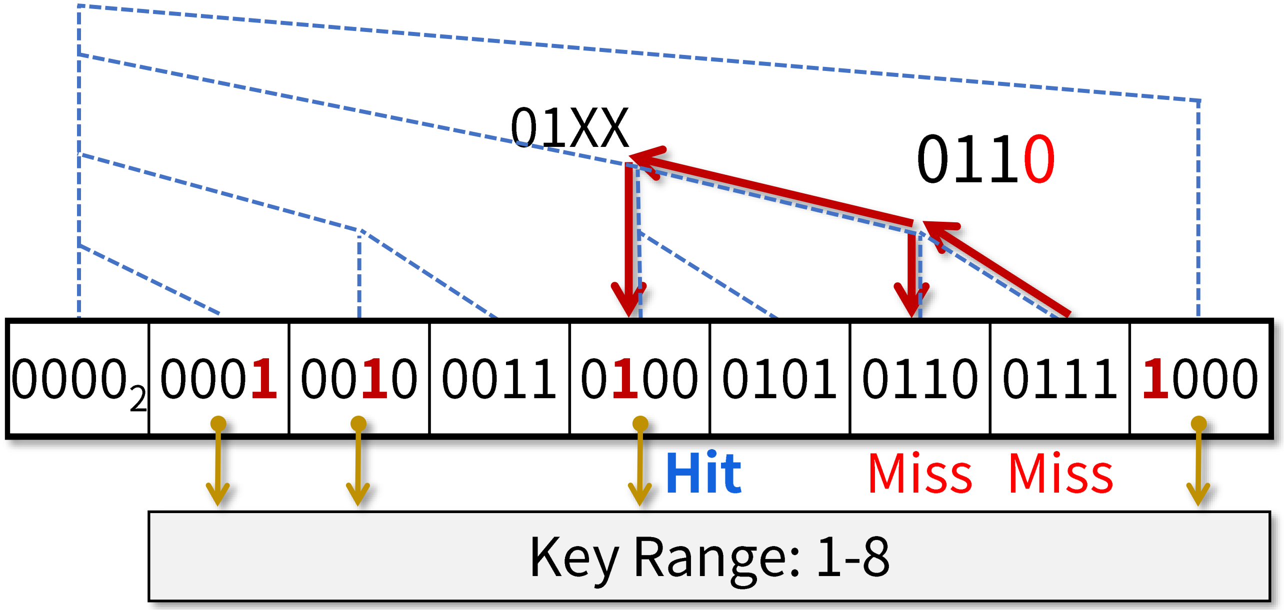
- **Example**



 # pointers: 8 → 4

Extent Hashing: Search

- **Search**
  - **while (search(query key))**
    Flip the rightmost non-zero bit of query key
- **Example:** Find an extent 7(0111_2) belongs to



- **Search Cost:** $O(\log_2 B)$

Evaluation

Experimental Setup

- **Evaluation machines**

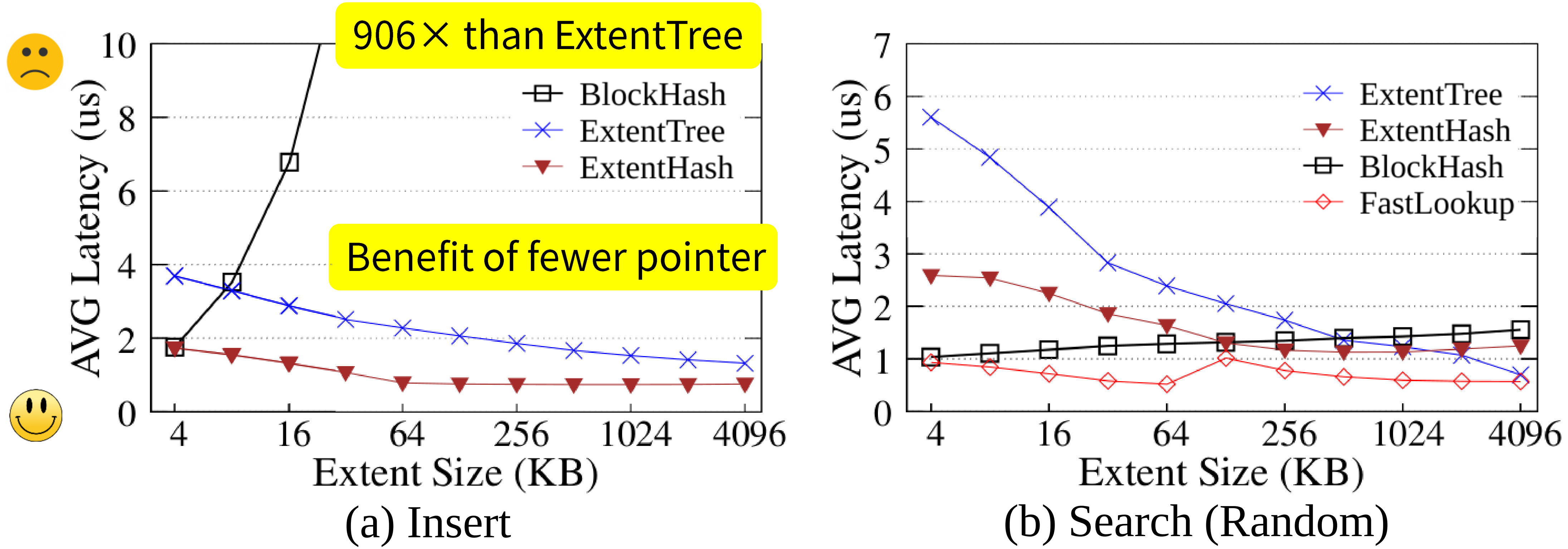
Server	Virtual	Processor	DRAM	PMEM	SSD
DCPMM	-	Dual Intel Xeon Gold 5215 (10 cores, 2.50 GHz)	128 GB DDR4	256 (128×2) GB Intel Optane DCPMM	2 TB Samsung 860 EVO mSATA SSD
NVDIMM-N	QEMU	Dual Intel Xeon Gold 5218 (16 cores, 2.30 GHz)	32 GB DDR4	16 GB Dell EMC NVDIMM-N	512 GB Samsung 970 NVMe SSD

※ Note: Evaluations performed on the NVDIMM-N server are marked with **(NVDIMM-N)** in the title

- **File System Setup**
  - Default mount option
  - EXT4, F2FS, XFS: SSD
  - NOVA: PMEM (DCPMM or NVDIMM-N)
  - Ziggurat: PMEM + SSD
  - SPFS+x: PMEM + x (EXT4, F2FS, XFS)

Extent Hashing: Comparison of file mapping structures

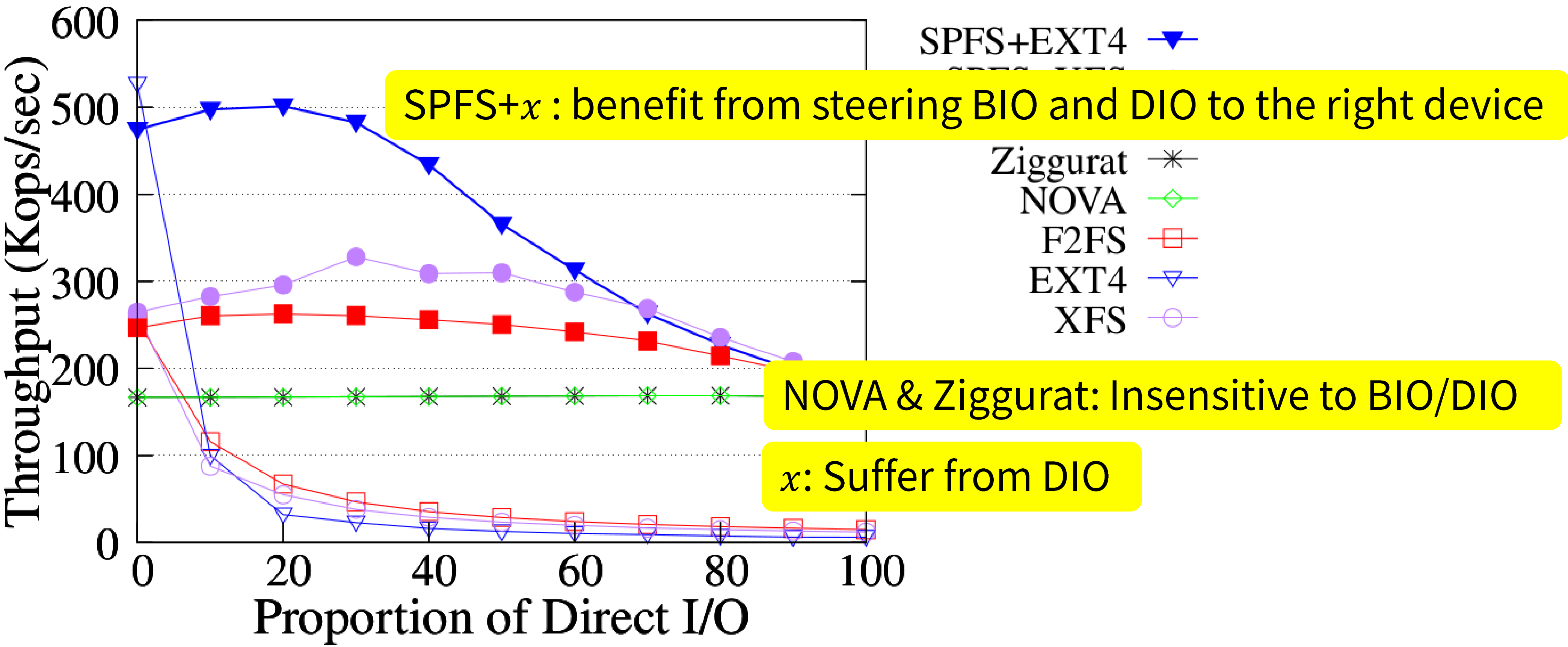
- 8000 256 MB files (KMEM-DAX)



- ExtentTree: Per-file, FAST and FAIR B+tree
- ExtentHash: Global, based on CCEH
- BlockHash: Global, ExtentHash stride=1, no log-scale search (same as HashFS)

Performance Effect of Stacking SPFS on x

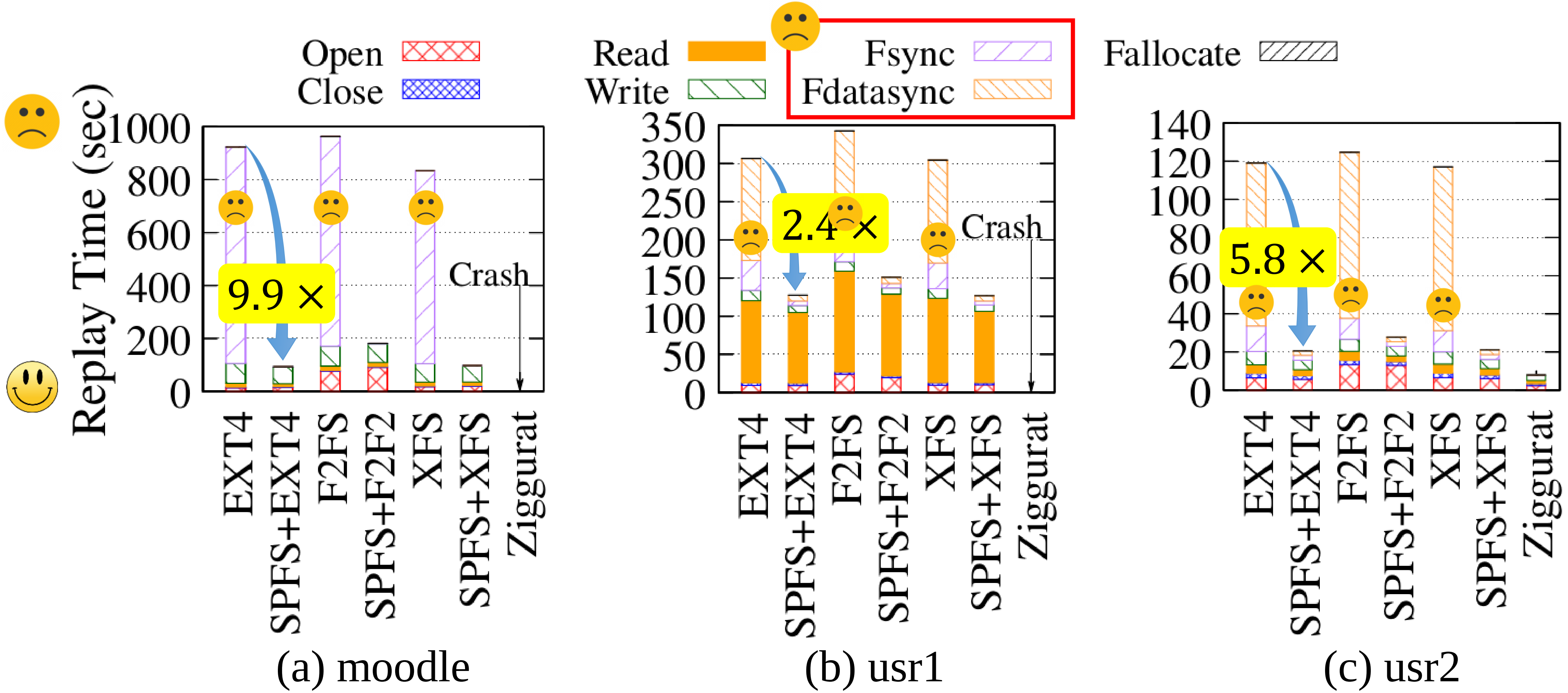
- Mix of buffered I/O (BIO) and direct I/O(DIO) - *fileserver* workload



- **SPFS+x benefits from the device-aware stackable design**
  - Steers BIOs to the lower file system while absorbing the DIOs in DCPMM

Experiments with FIU Trace (small NVDIMM-N)

- Replay transactional traces of FIU: *Moodle, Usr1, Usr2*



Conclusion

- **We designed and implemented SPFS**
  - A stackable file system for PMEM
    - Absorb order-preserving small synchronous writes
  - Take advantage of the legacy block device file systems
    - Provide faster DRAM cache and large capacity
  - Manage all file system metadata in dynamic hash tables
    - Novel Extent Hashing

- **SPFS+x Improves performance of lower file system x by up to $9.9\times$**

Thank You :)

Questions?

hobin.woo@samsung.com
hdg9400@skku.edu