FAST'24

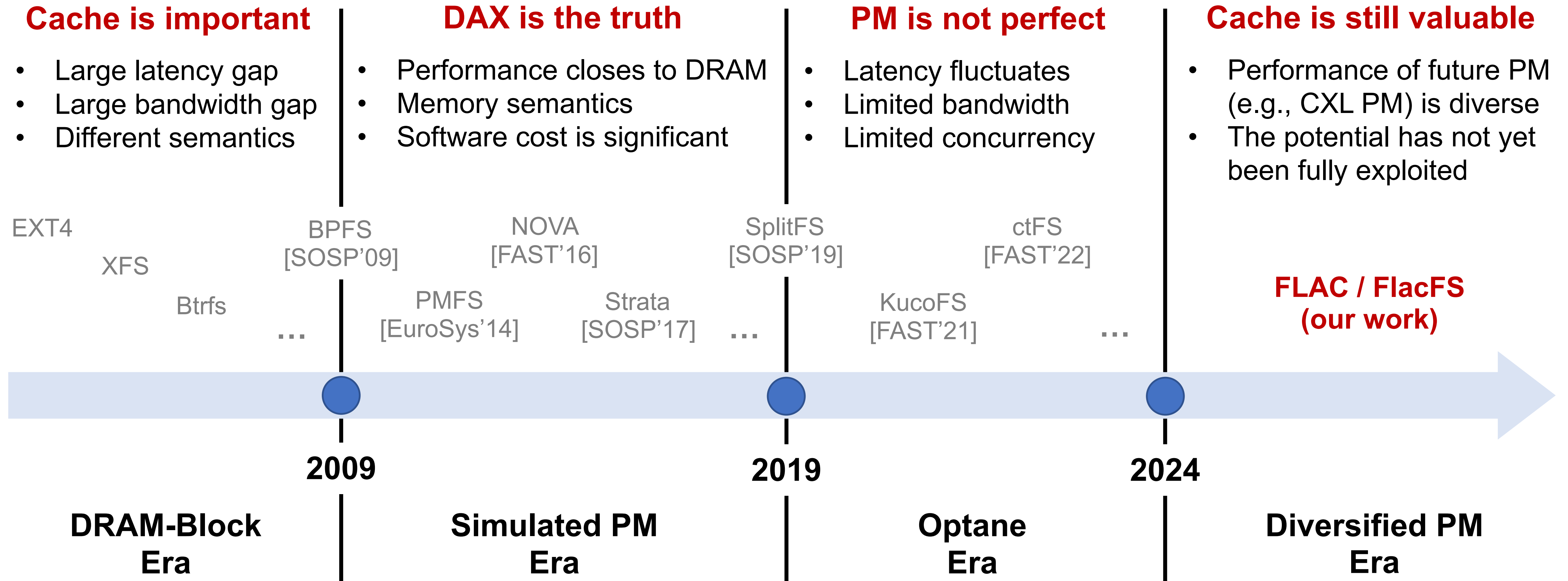
Optimizing File Systems on Heterogeneous Memory by Integrating DRAM Cache with Virtual Memory Management

Yubo Liu[1], Yuxin Ren[1], Mingrui Liu[1], Hongbo Li[1], Hanjun Guo[1], Xie Miao[1], Xinwei Hu[1], and Haibo Chen[1,2]

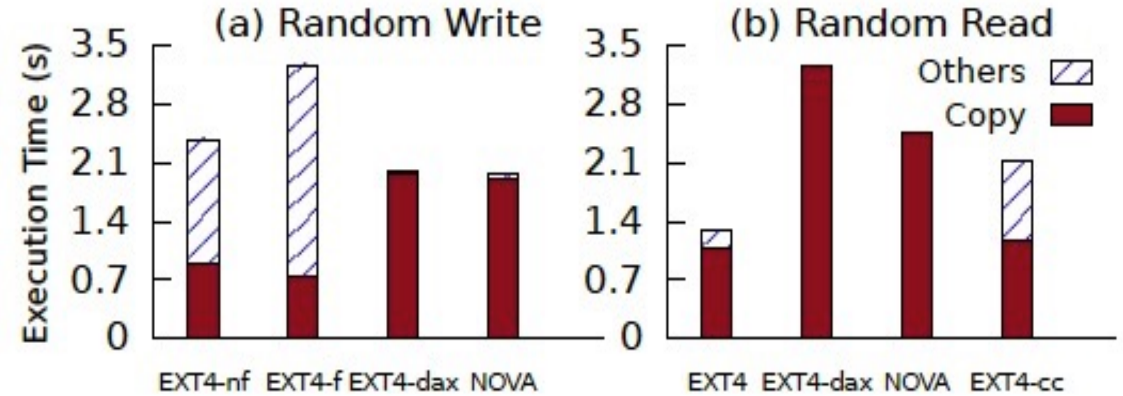
[1] *Huawei Technologies Co., Ltd.* [2] *Shanghai Jiao Tong University*

上海交通大學
Shanghai Jiao Tong University

File Systems meet Heterogeneous Memory



Cache? or Direct Access (DAX)?

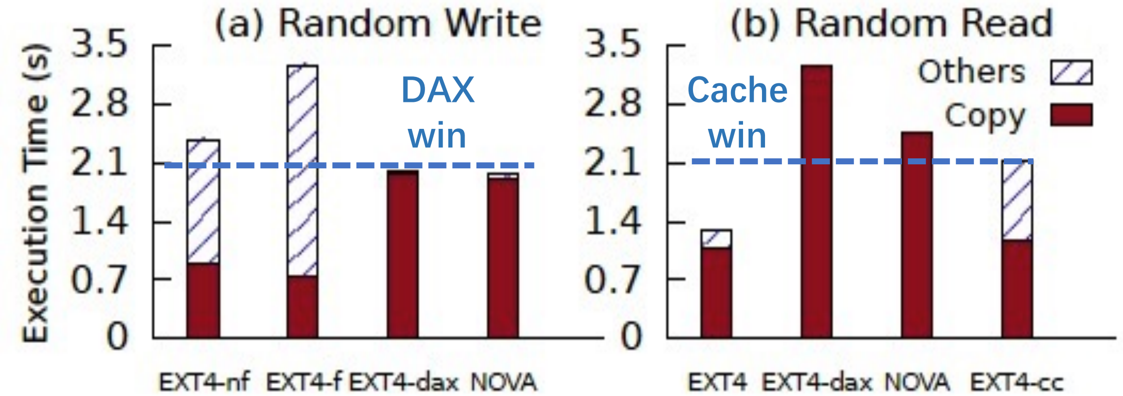


- **Cache-based FS:** EXT4
- **DAX-based FS:** EXT4-DAX, NOVA
- **Experiment Setup:** 10GB data; 2MB I/O; 1 thread

Cache? or Direct Access (DAX)?

Observation 1:

Existing DAX and cache solutions are suboptimal, but DRAM cache still has great value



There is no winner between Cache and DAX

- Performance gap between PM and DRAM cannot be ignored
- Data locality is important for performance optimization
- DAX is an overkill in many real-world scenarios

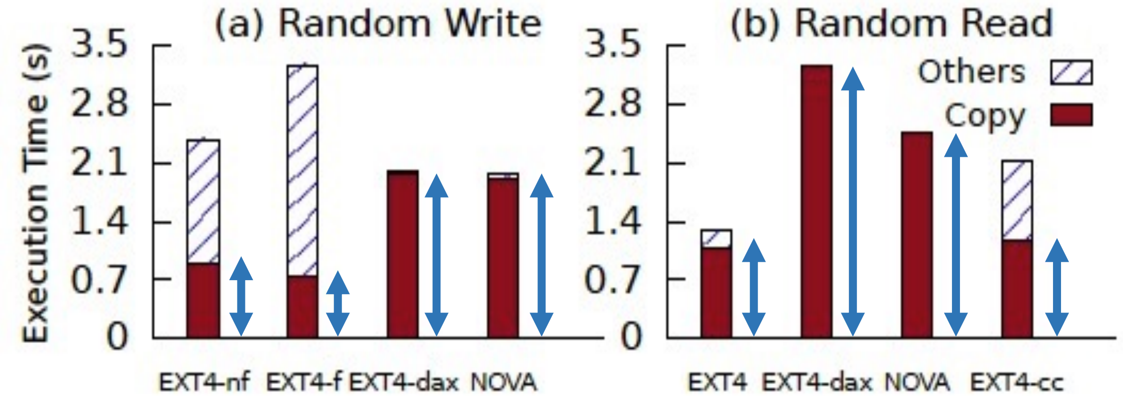
Cache? or Direct Access (DAX)?

Observation 1:

Existing DAX and cache solutions are suboptimal, but DRAM cache still has great value

Observation 2:

Data transfer overhead between the file system and application buffer is significant



- Takes up more than 23% of the total overhead in cache-based file systems
- Takes up more than 96% of the total overhead in DAX-based file systems

Cache? or Direct Access (DAX)?

Observation 1:

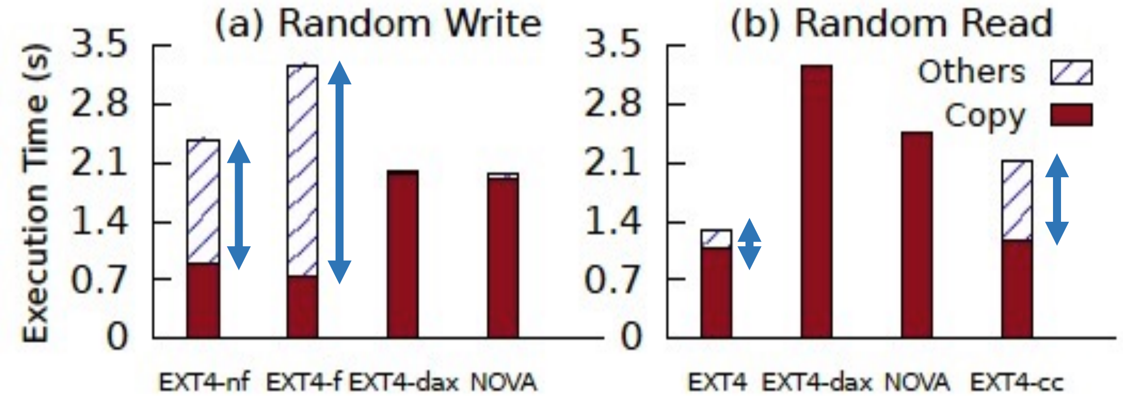
Existing DAX and cache solutions are suboptimal, but DRAM cache still has great value

Observation 2:

Data transfer overhead between the file system and application buffer is significant

Observation 3:

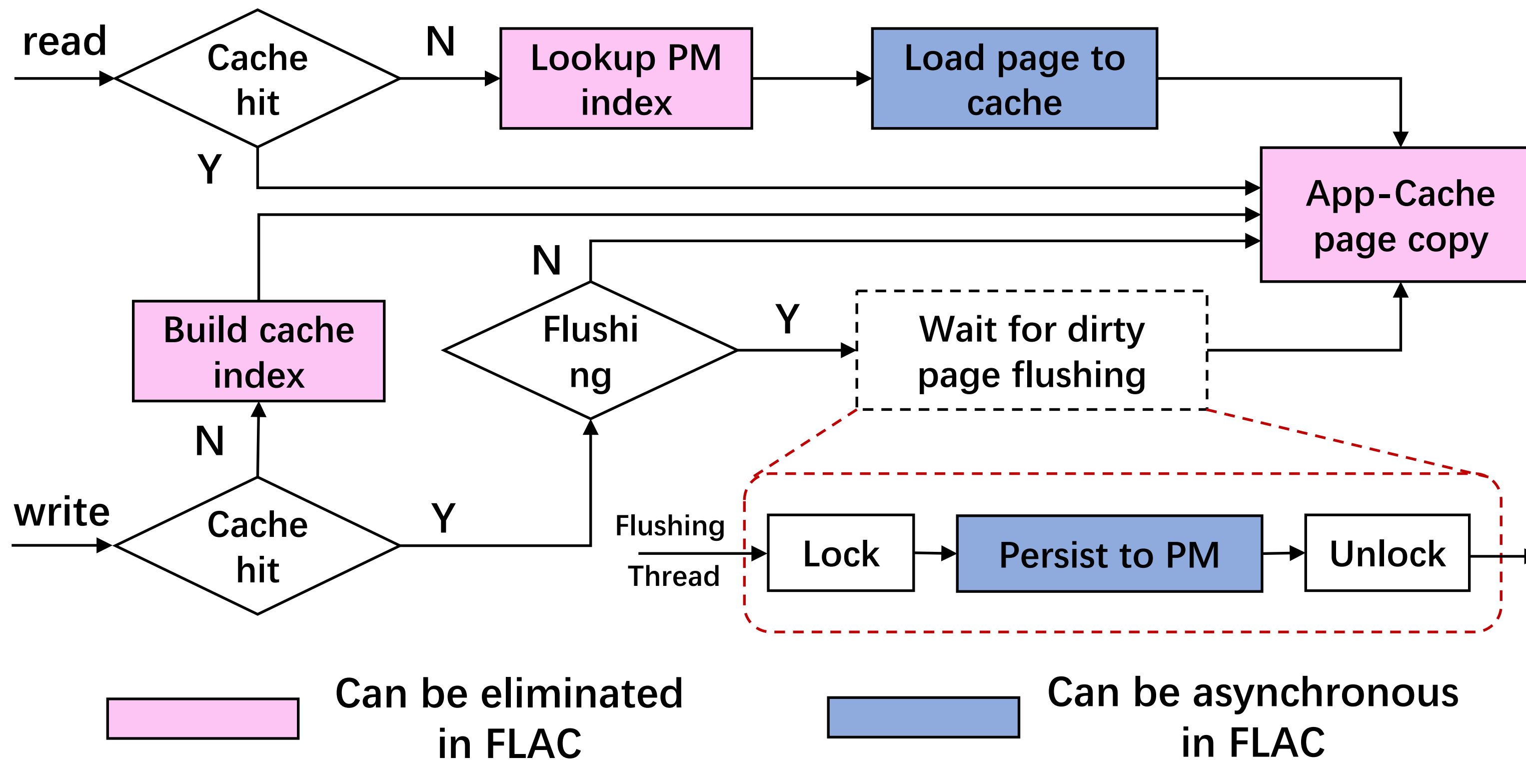
"Cache Tax" is heavy, and it mainly includes the overhead of data synchronization and migration



- Data synchronization (background dirty flushing) lead to 37% performance declines
- Data migration (cache miss handling) lead to 65% performance declines

Motivation

Integrating Cache with Virtual Memory Management



Principle 1:

Optimizing data transfer between application and cache by zero-copy and reducing two-level index overhead

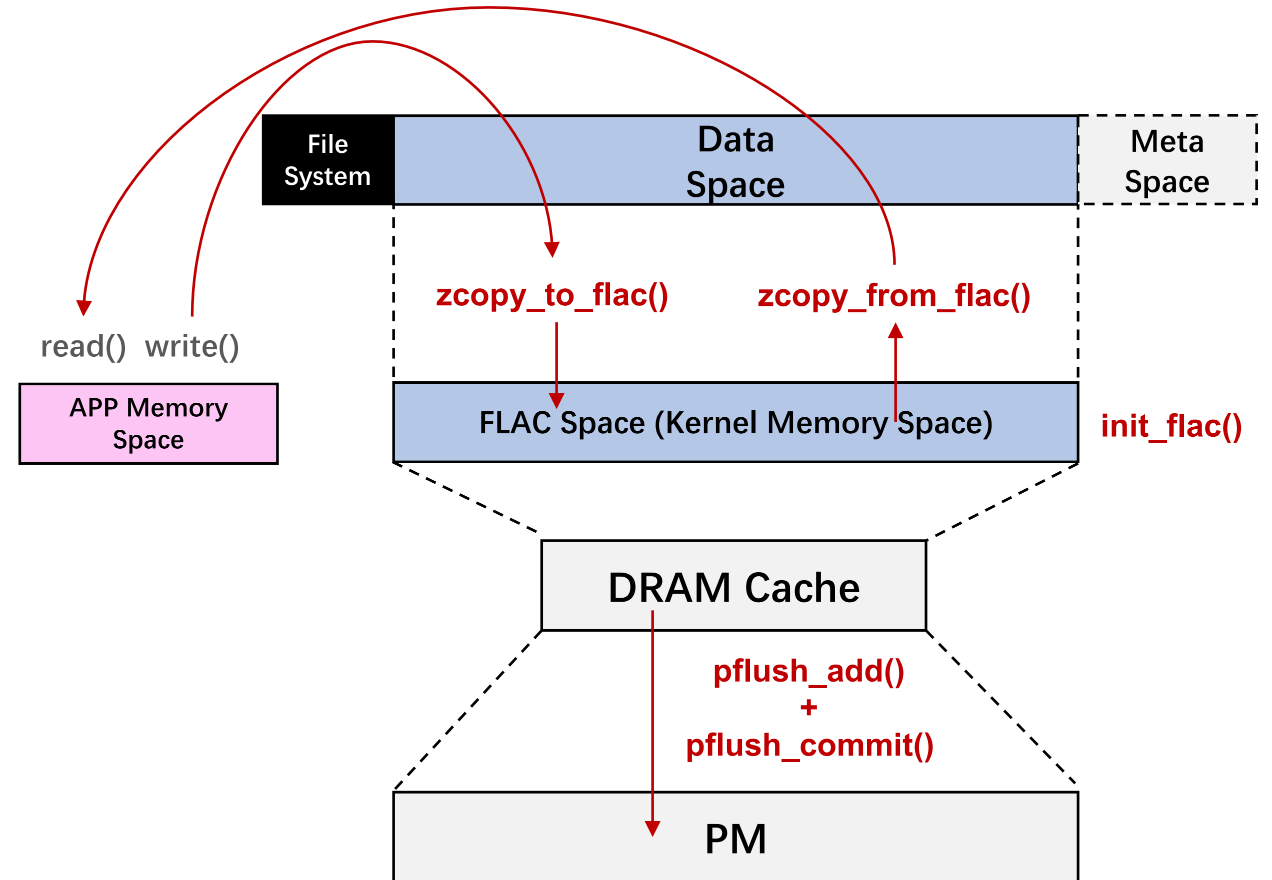
Principle 2:

Reducing the impact of "cache tax" by hiding the data synchronization /migration overhead

FLAC Design

Overview and APIs of FLAC (FLAt Cache)

APIs	Parameters
init_flac	*pm_path*
zcopy_from_flac **zcopy_to_flac**	*from_addr* *to_addr* *size*
pflush_add	*pflush_handle* *addr* *size*
pflush_commit	*pflush_handle* *fs_metalog*
pfree	*addr* *size* *fs_metalog*

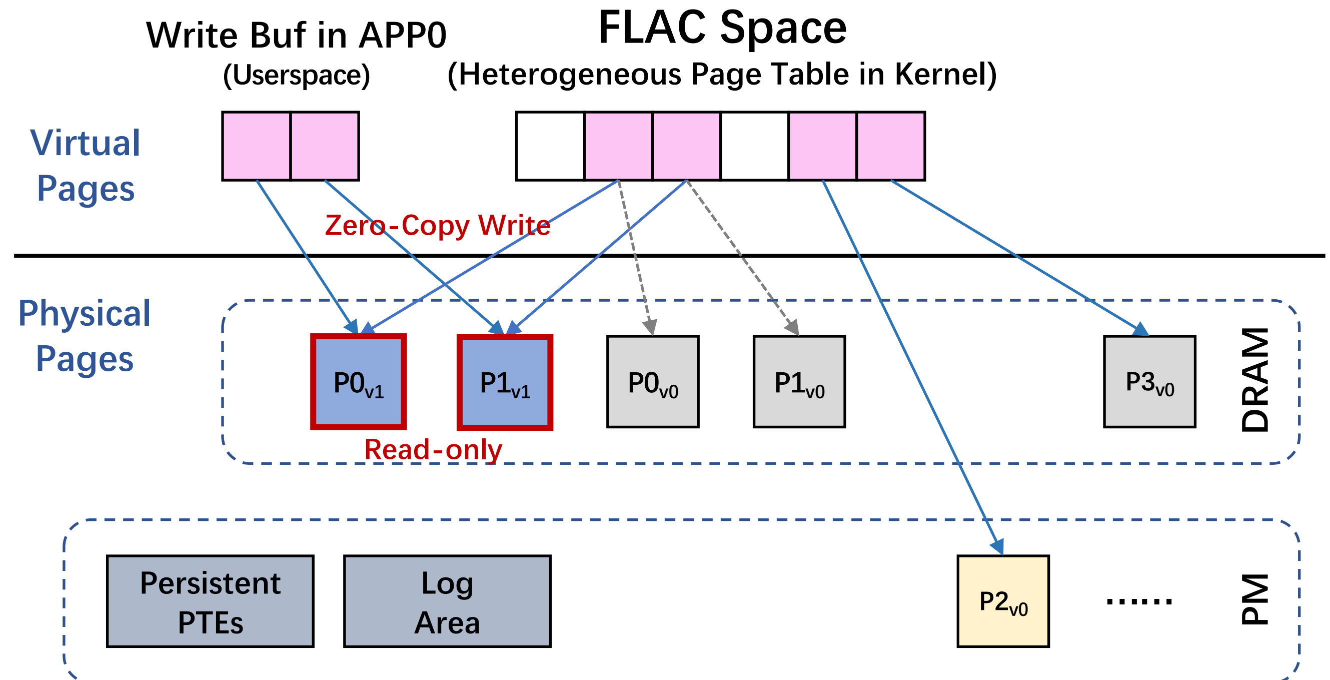


FLAC Design

Optimizing APP-Cache Data Transfer

Tech 1: Zero-Copy Caching

- Heterogeneous Page Table
  - Unified and contiguous virtual memory address space
  - Dynamically mapped to DRAM or PM as the page is cached or evicted
  - PTEs of FLAC space are replicated in PM for fault recovery
- Page Attaching
  - Map physical pages from the source address to destination address
  - Set pages to read-only to ensure security

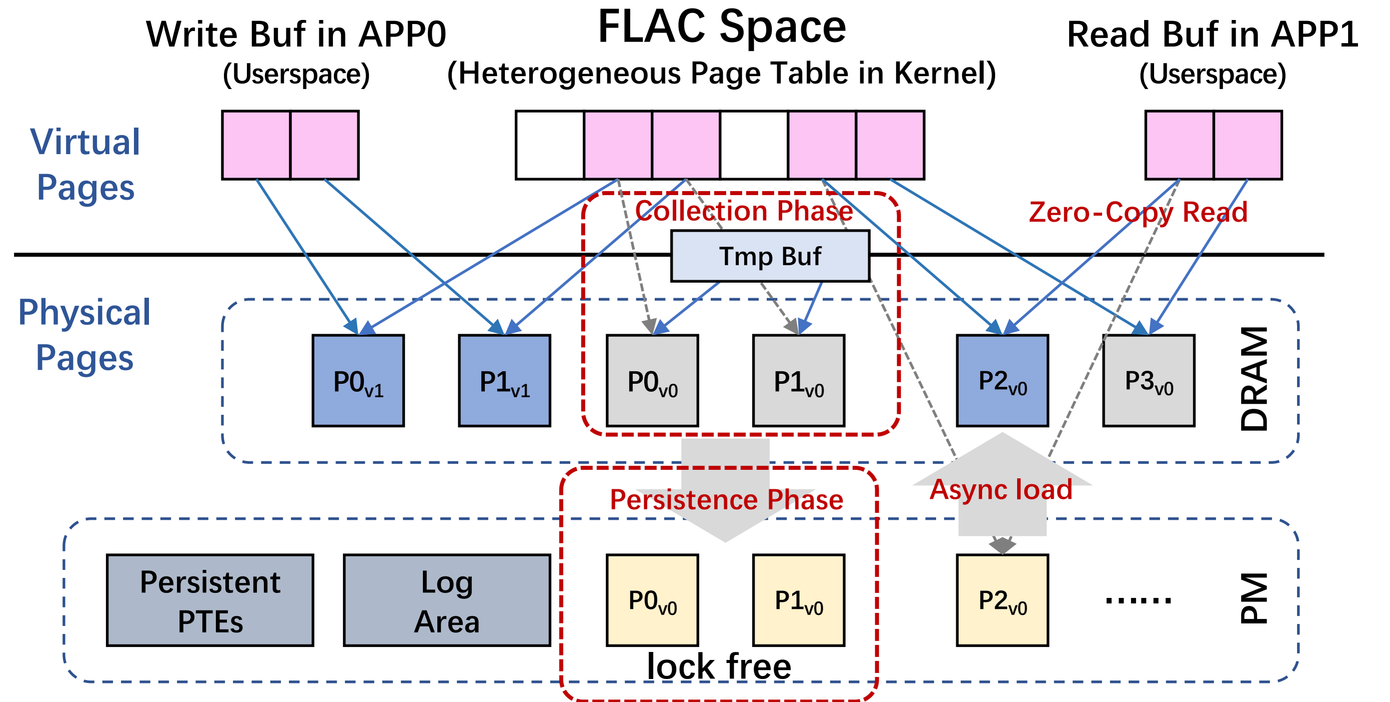


FLAC Design

Reducing the Impact of "Cache Tax" (data synchronization & migration)

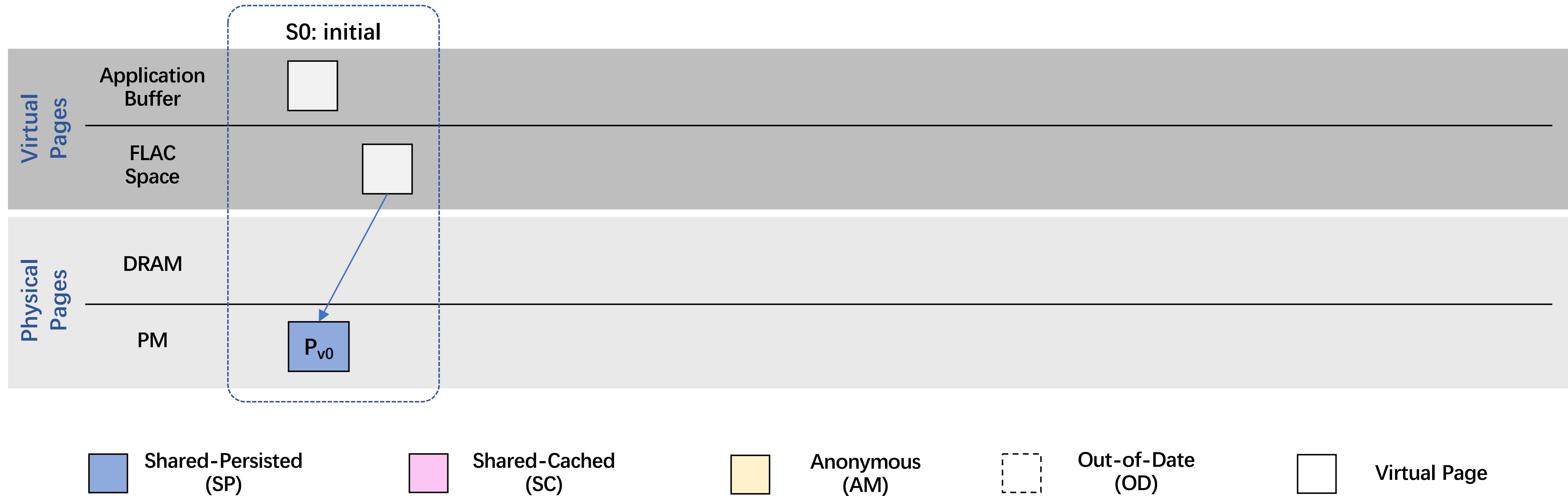
Tech 2: Parallel-Optimized Cache Management

- 2-Phase Flushing
  - Collection phase (lock)
  - Persistence phase (lock-free)
- Async Cache Miss Handling
  - Directly attach missed pages
  - Async load missed pages



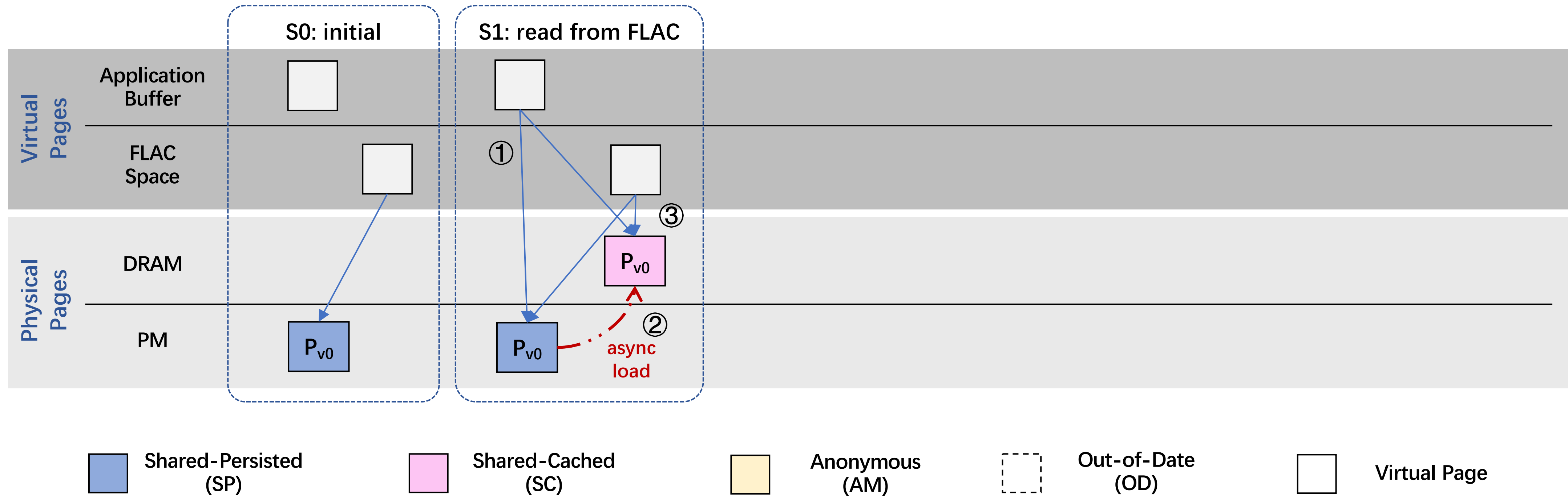
FLAC Design

Page State/Version Transition: An Example



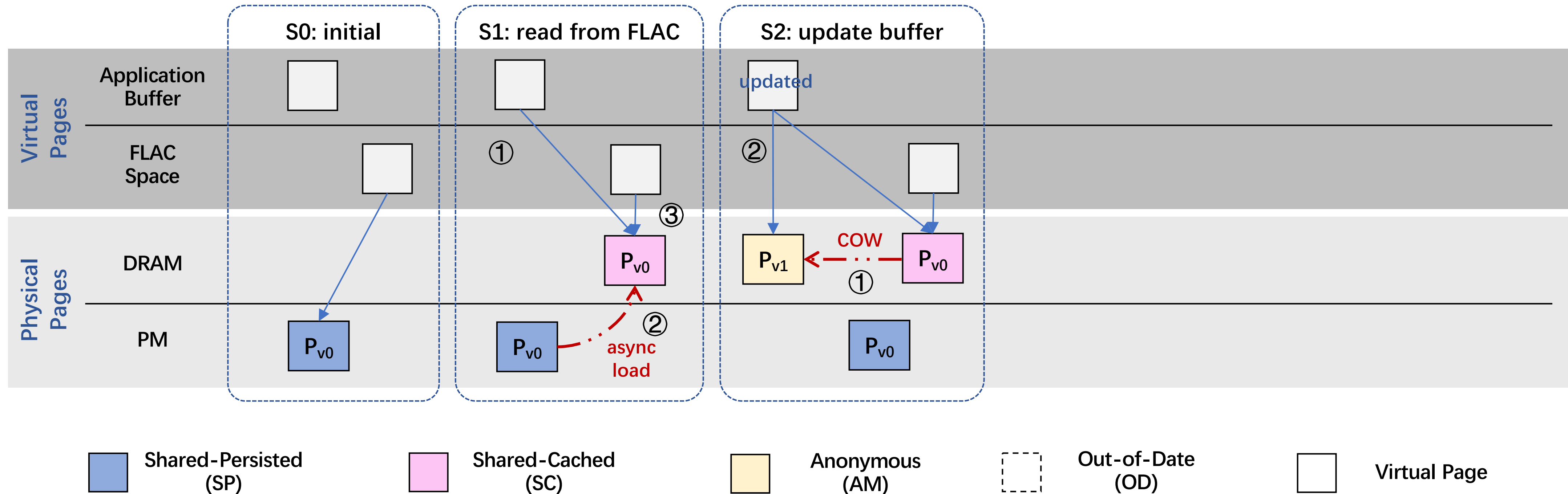
FLAC Design

Page State/Version Transition: An Example



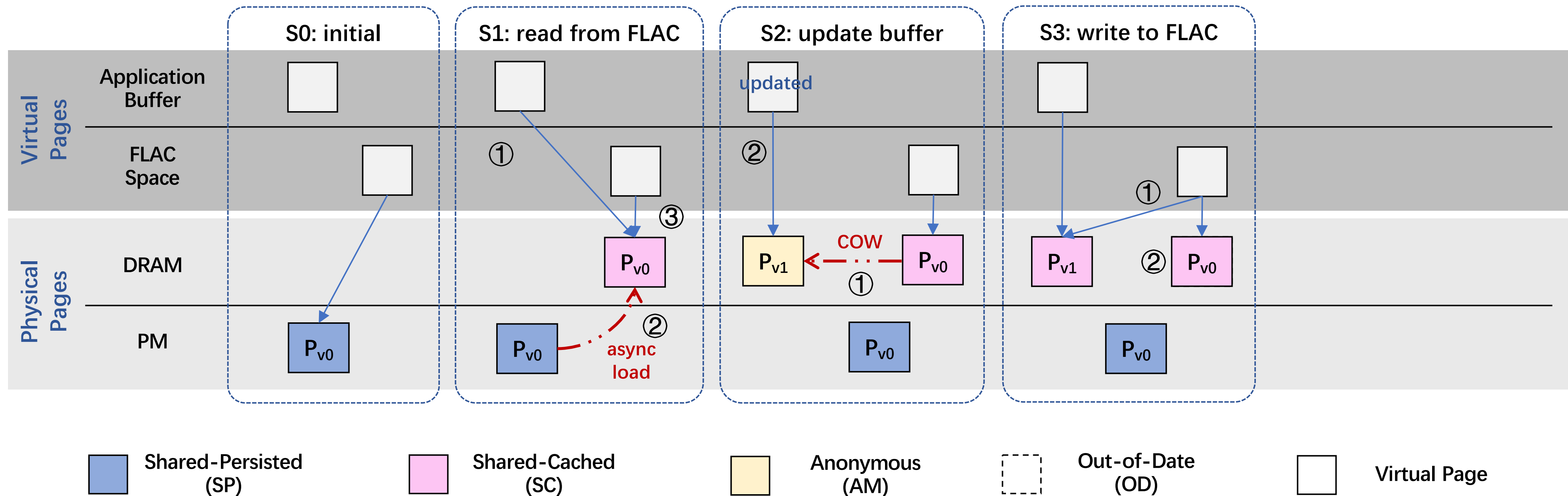
FLAC Design

Page State/Version Transition: An Example



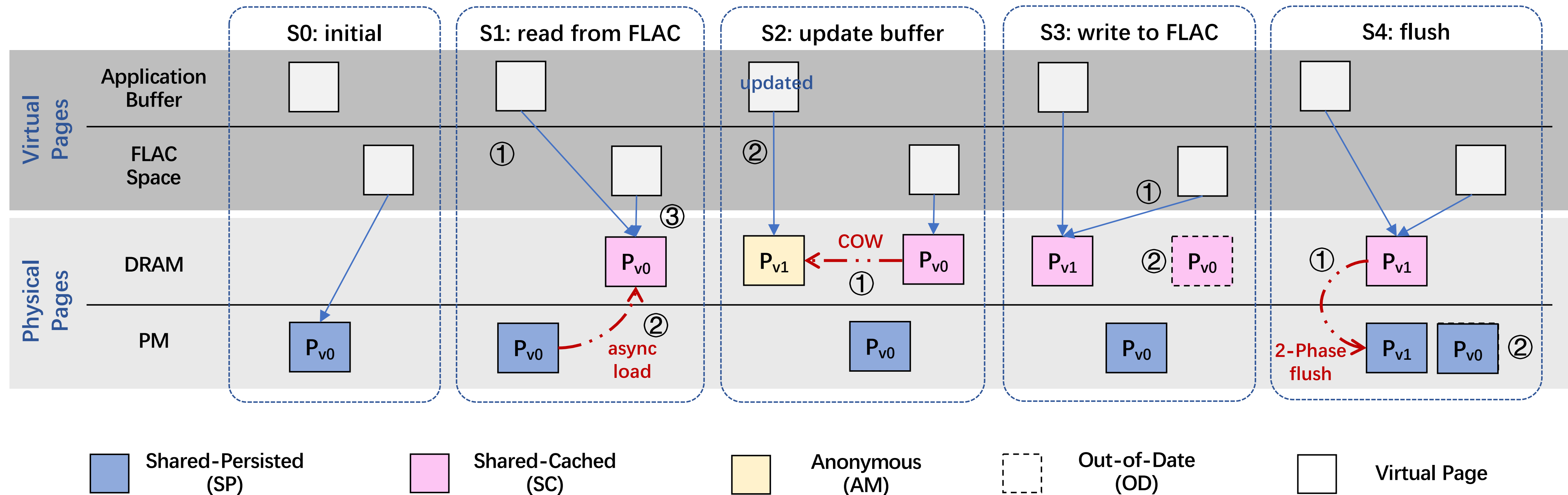
FLAC Design

Page State/Version Transition: An Example



FLAC Design

Page State/Version Transition: An Example



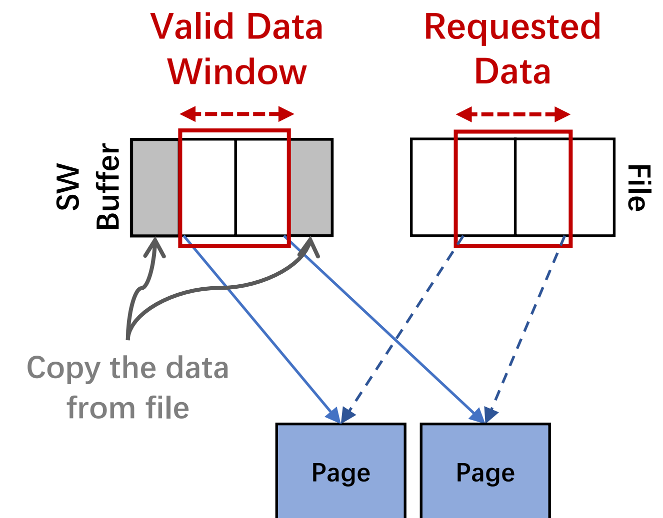
FLAC Design

Challenge#1 Page Unaligned → *Sliding Window Buffer*

- Use ***SWbuf*** to proxy buffer management in application
- Map all pages containing required data
- Use sliding window to denote valid data

Challenge#2 COW Page Fault → *bfault & detach*

- Call ***bfault/detach*** before reusing the R/W buffer
- **Batch fault (bfault)** — For: Need to process data in the buffer
  - Batching the data copies and TLB flushes
- **Detach** — For: Just reuse the space of the buffer
  - Mapping to empty pages in batch



Example: File write by sliding window buffer

Case Study: FlacFS

Architecture

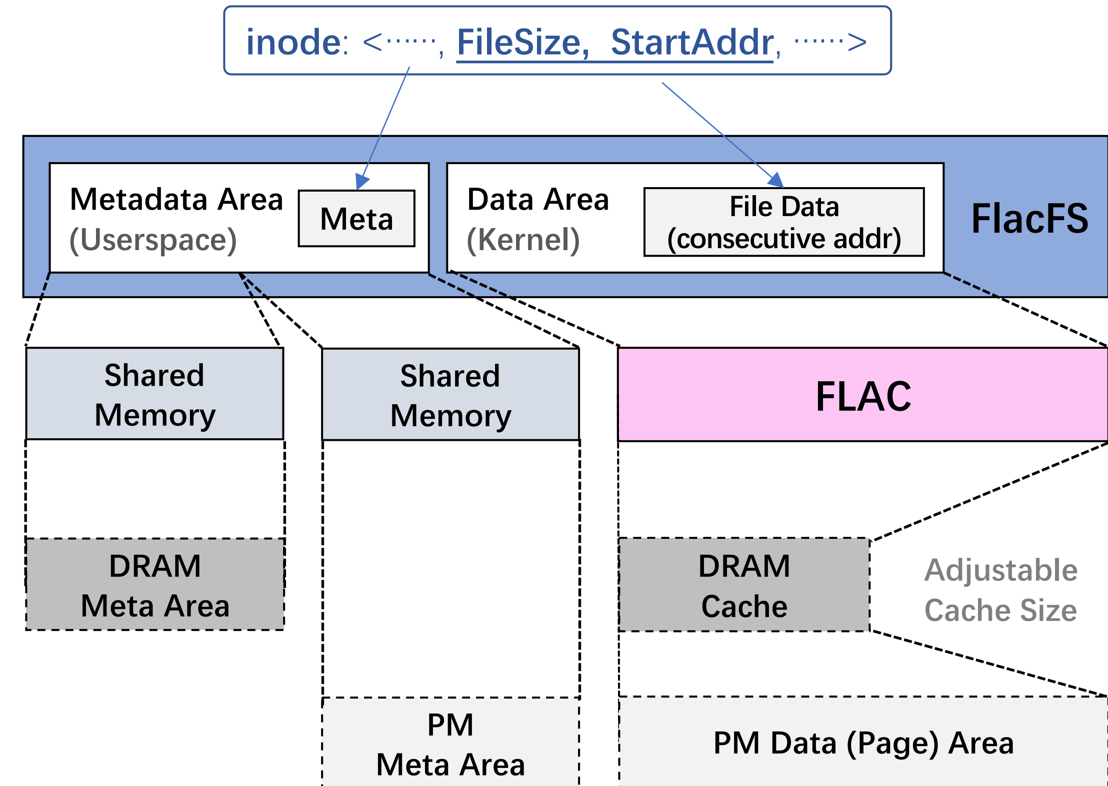
- Metadata area is on shared memory
- Data area is on FLAC space

Metadata & Data Management

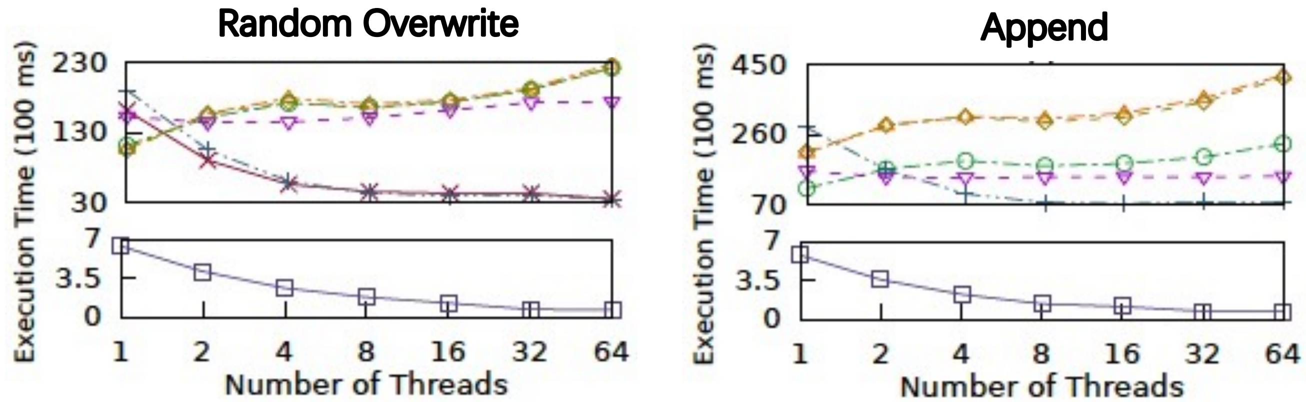
- Inodes hash Table (DRAM+PM)
- File's data is on consecutive address (insighted by ctFS)

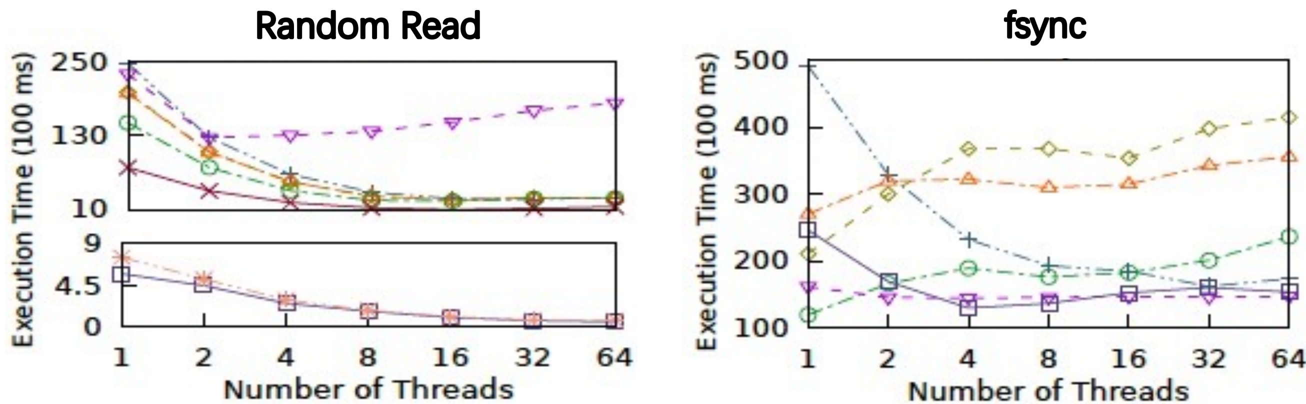
Consistency

- FS-FLAC collaboration logging
  - Put FS-level & FLAC-level metadata into the same log entry
  - Data flushing is log-structured



Benchmark Performance





FlacFS-HIT SplitFS EXT4-DAX NOVA EXT4-HIT

ctFS EXT4-MISS FlacFS-MISS

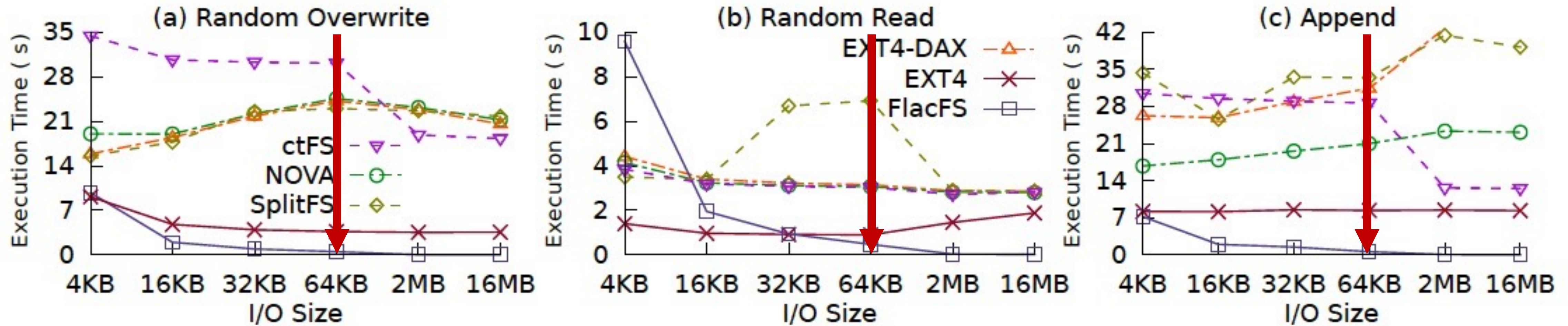
Experiment Setup: 2MB I/O; 64GB data

	NOVA	SplitFS	ctFS	EXT4-DAX	EXT4	FlacFS
Mode	sync	POSIX				
Cons.	Meta					Meta+Data
Cache Flush	N/A				100 ms	10 ms

Summary

- More than one order of magnitude over other FSes in write/read operations
- Better scalability
- Comparable to the best DAX FS and better than EXT4 in fsync

Design Analysis



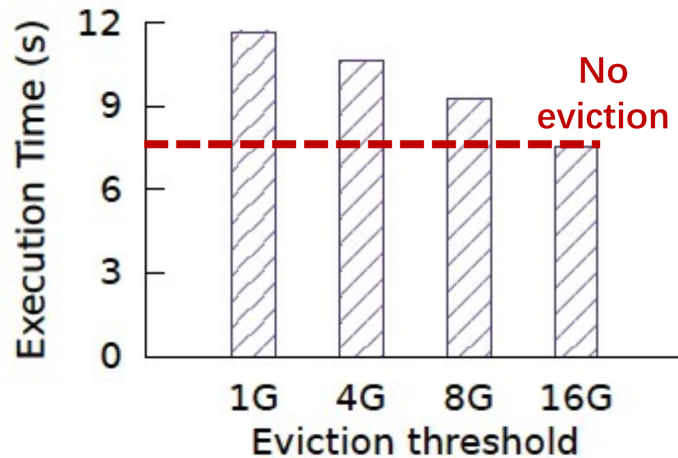
Impact of I/O Size

- 64 concurrent threads
- 64 files
- I/O sizes range from 4KB to 16MB

Summary

- FlacFS is more friendly to I/O >= 64KB
- I/O >= 64KB is common in production

Design Analysis

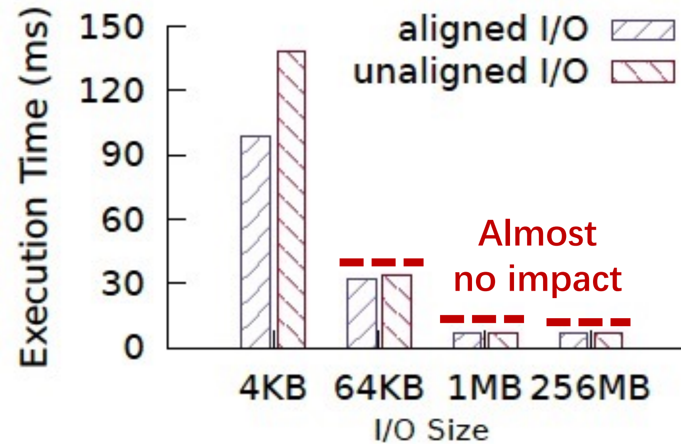


Impact of DRAM Cache Size

- Append 16GB data to files
- The smaller the threshold, the greater the number of eviction

Summary

Page eviction is efficient in FLAC

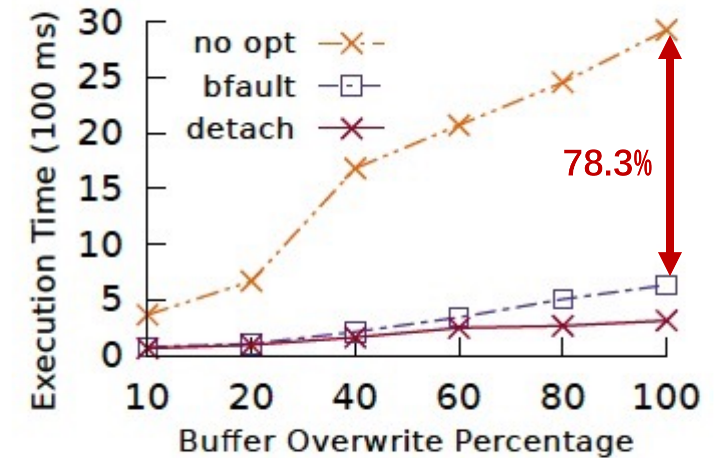


Impact of Page Alignment (swbuf)

- Overwrite 1GB data in the file
- Use sliding window buffer

Summary

Unalignment has little impact on I/O >= 64KB



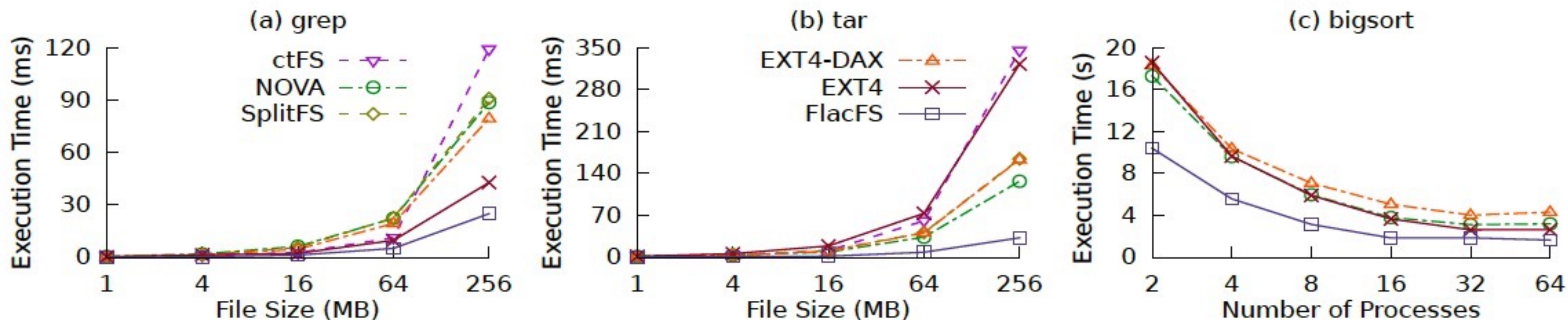
Impact of COW Page Fault

- Rewrite the buffer in different proportions by *memset* after each file access

Summary

bfault/detach significantly reduce the COW page fault overhead

Real-World Application



Experiment Setup

- ***grep:*** read-intensive
- ***tar:*** read- & write-intensive
- ***bigsort:*** read- & write- & compute-intensive (134 million integers)
- All optimizations (bfault/detach) are used where appropriate

Summary

- Up to 6.7X improvement vs. DAX-based FS
- Up to 9.4X improvement vs. Cache-based FS
- bfault/detach is efficient in real-world scenarios

Conclusion

- **Analysis of the cache/DAX solution on heterogeneous memory**
  - Cache has great value if designed properly
  - Data transfer overhead is high
  - “Cache Tax” is heavy
- **FLAC, a flat cache framework for heterogeneous memory**
  - Zero-copy caching
  - Parallel-optimized cache management
- **FlacFS, a file system based on FLAC**
  - Orders of magnitude performance improvement in micro benchmark
  - Several times performance improvement in real-world applications

Thanks :)