How to Teach an Old File System Dog New Object Store Tricks

USENIX HotStorage ’18

Eunji Lee[1], Youil Han[1], Suli Yang[2],
Andrea C. Arpaci-Dusseau[2], Remzi H. Arpaci-Dusseau[2]

Chungbuk National University

WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON

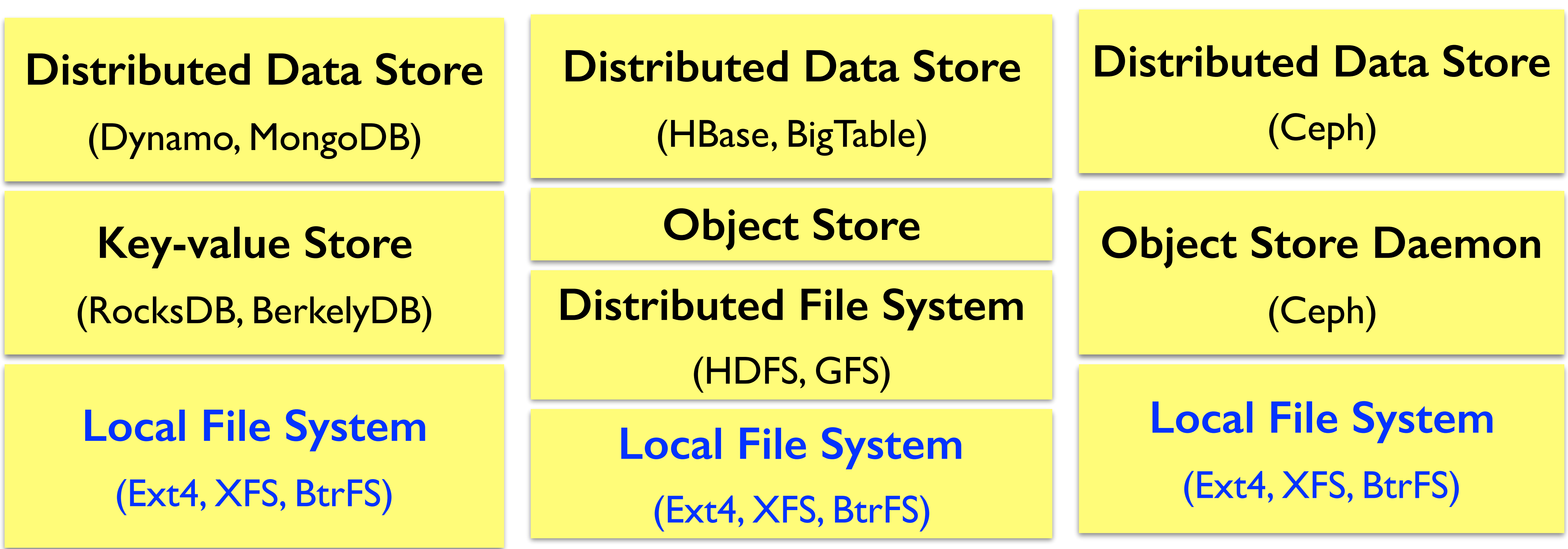
Data-service Platforms

- **Layering**
  - Abstract away underlying details
  - Reuse of existing software
  - Agility: development, operation, and maintenance

Eco System of Data-Service Platform

Often at odds with efficiency

- **Local File System**
  - Bottom layer of modern storage platforms
  - Portability, Extensibility, Ease of Development



Local File System

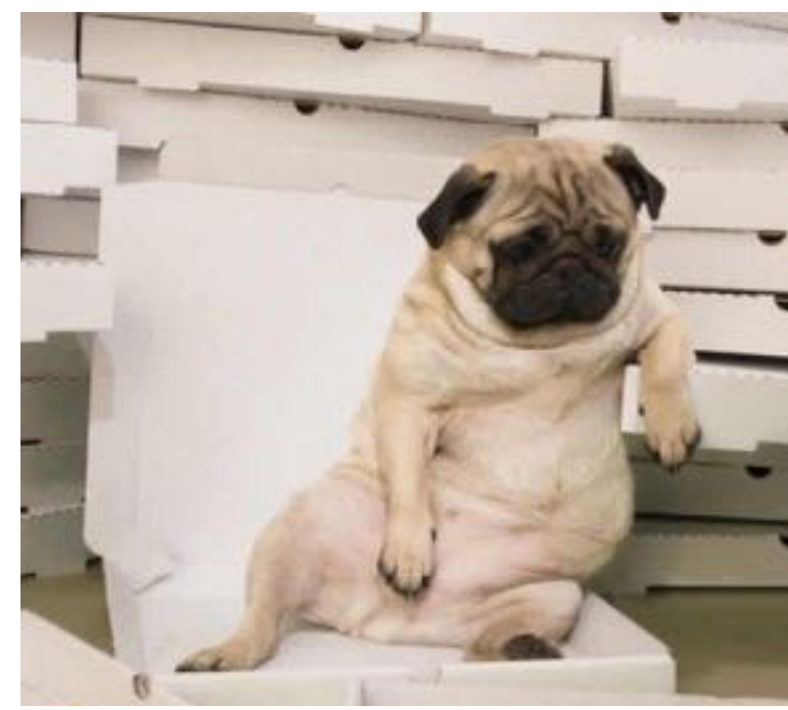
- Not intended to serve as an underlying storage engine
- Mismatch between the two layers
- System-wide optimization
  - Ignore demands from individual applications
  - Little control over file system internals
  - Suffer from degraded QoS
- Lack of required operations
  - No atomic operation
  - No data movement or reorganization
  - No additional user-level metadata

Out-of-control and Sub-optimal Performance

Current Solutions

- Bypass File System
  - Key-value store, Object Store, Database
  - But, reliniquish file system benefits

- Extend file system interfaces
  - Add new features to POSIX APIs
  - Slow and conservative evolution
  - Stable maintenance than specific optimizations



Name: Ext2/3/4
Birth: 1993

Our Approach

- Use a file system as it is, but in a different manner!
- Design patterns of user-level data platform
  - Take advantages of file system
  - Minimize negative effects of mismatches

Contents

- Motivation
- **Problem Analysis**
- SwimStore
- Performance Evaluation
- Conclusion

Data-service Platform Taxonomy

What is the best way to store objects atop a file system?

Mapping

hadoop

SWIFT
Object Storage

ceph

"Object as a file"

Packing

levelDB

RocksDB

amazon DynamoDB

Google BigTable

mongoDB

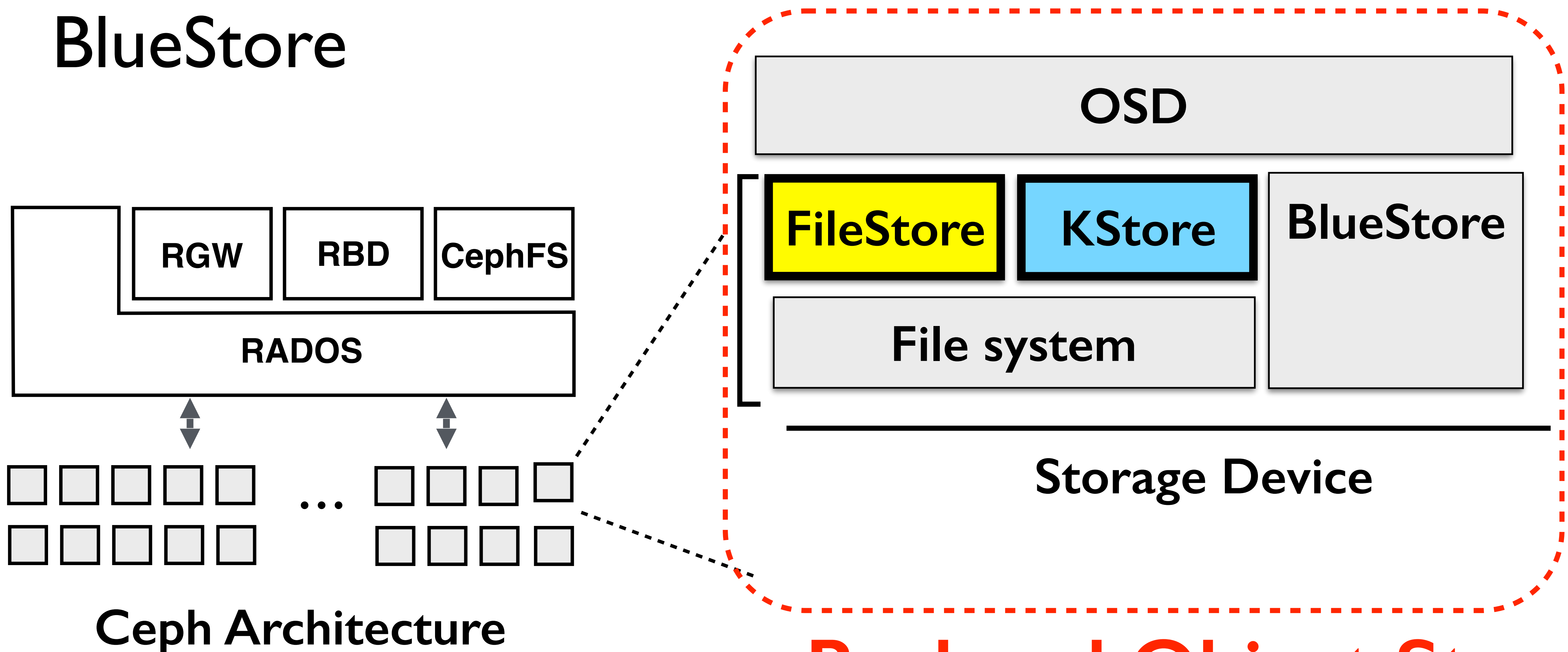
APACHE HBASE

CouchDB relax

"Multiple objects in a file"

Case Study: Ceph

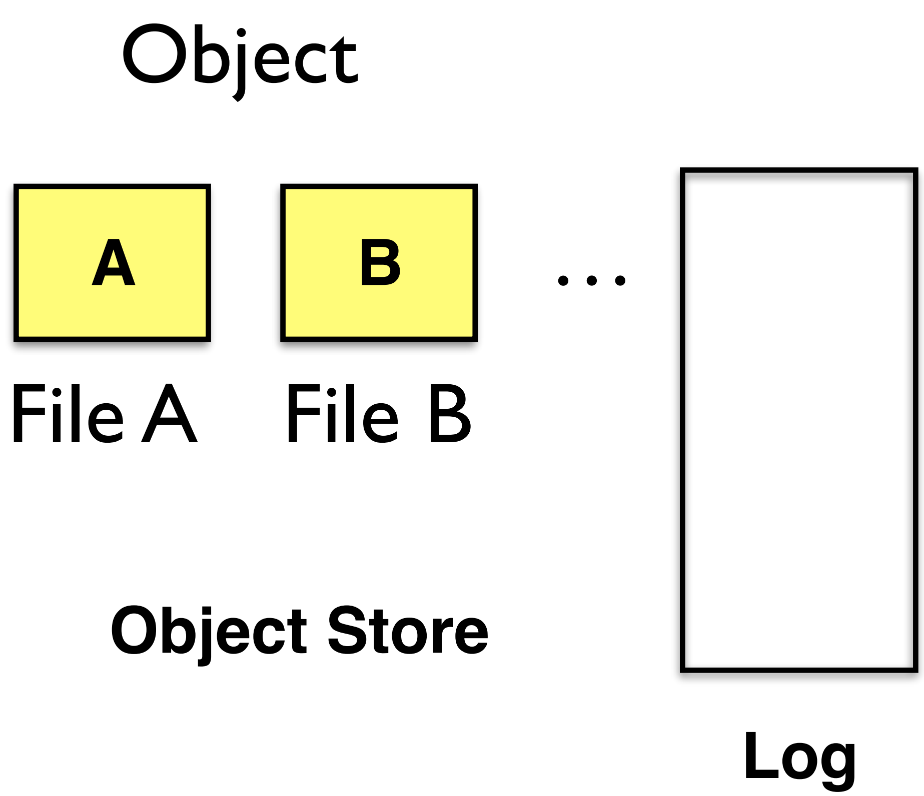
- Backend object store engine
  - FileStore : **mapping**
  - KStore : **packing**
  - BlueStore



Backend Object Store

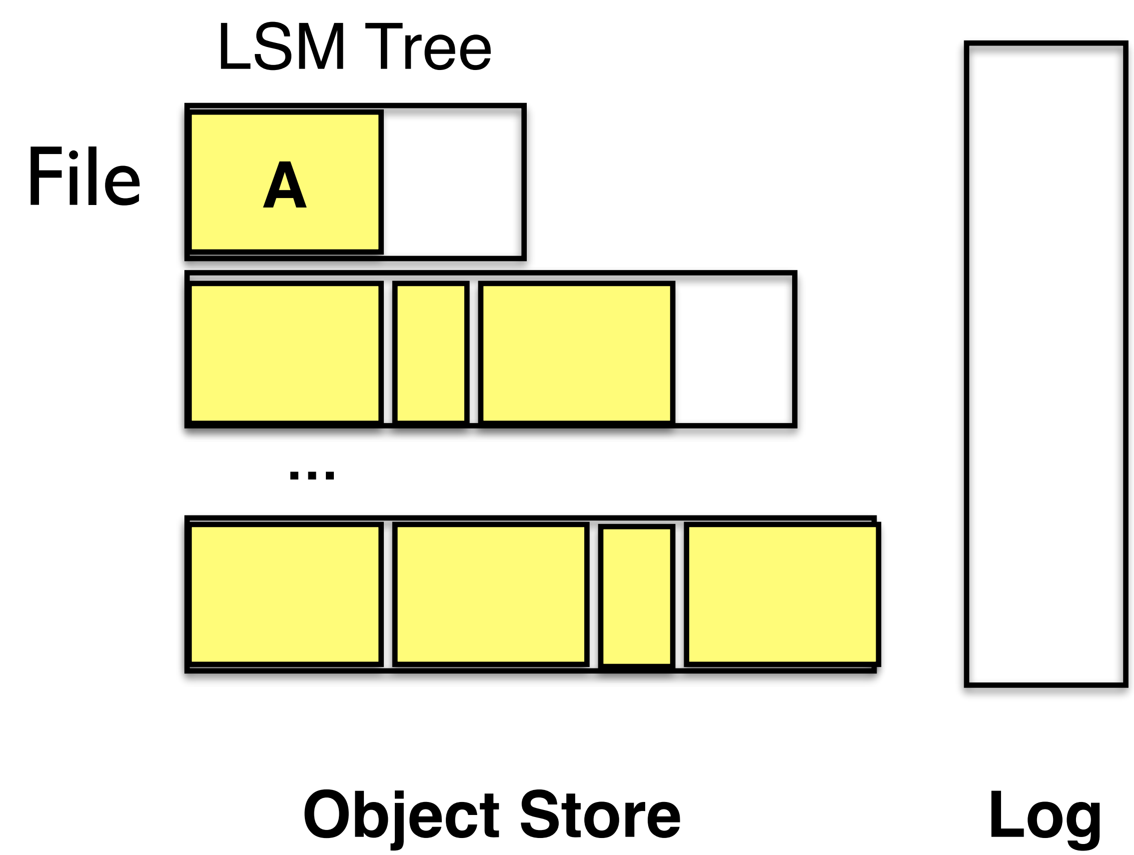
Mapping vs. Packing

FileStore (Mapping)



"Object as a File"

KStore (Packing)



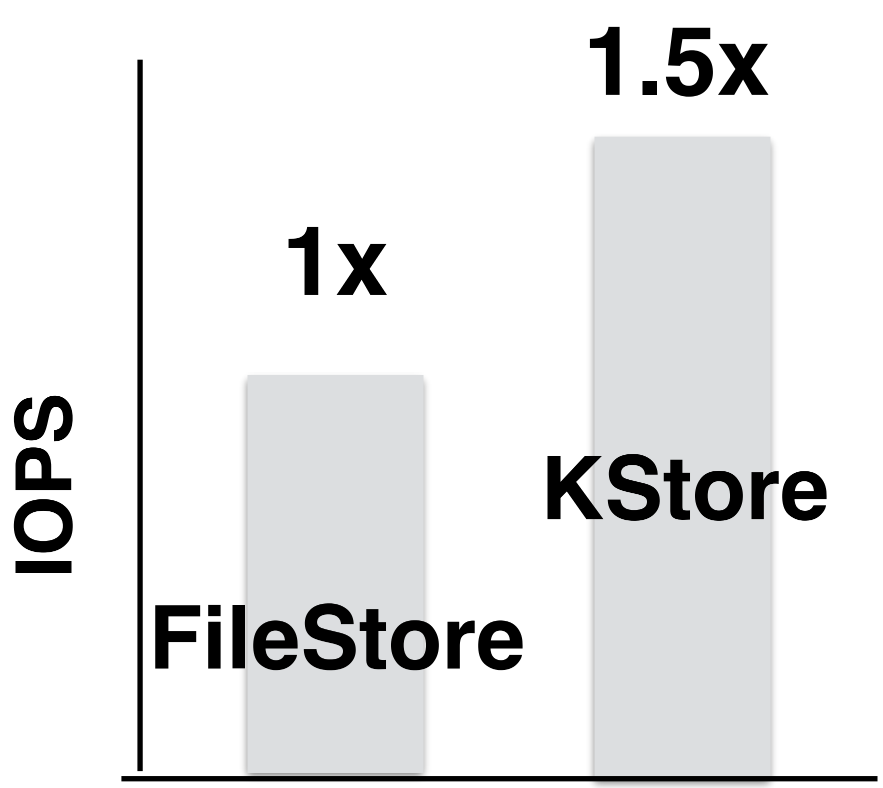
"Multiple Objects in a File"

Experimental Setup

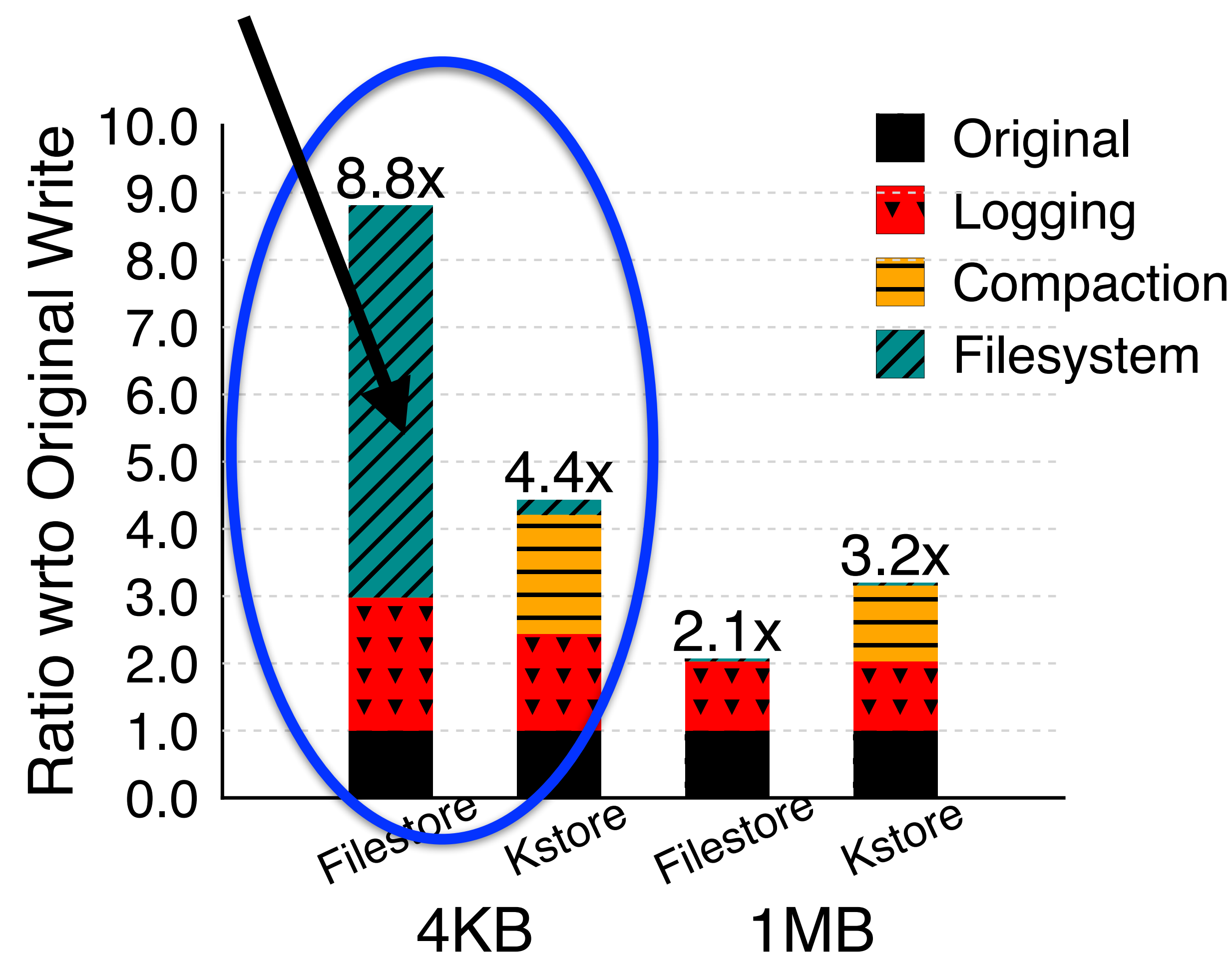
- Ceph 12.01
- Amazon EC2 Clusters
- Intel Xeon quad-core
- 32GB DRAM
- 256 GB SSD x 2
- Ubuntu Server 16.04
- File System : XFS (recommended in Ceph)
- Backend: FileStore, KStore
- Benchmark: Rados
- Metric: IOPS, throughput, write traffic

Performance

- Small Write (4KB)
  - KStore performs better than FileStore by 1.5x
  - Write amplification by file metadata



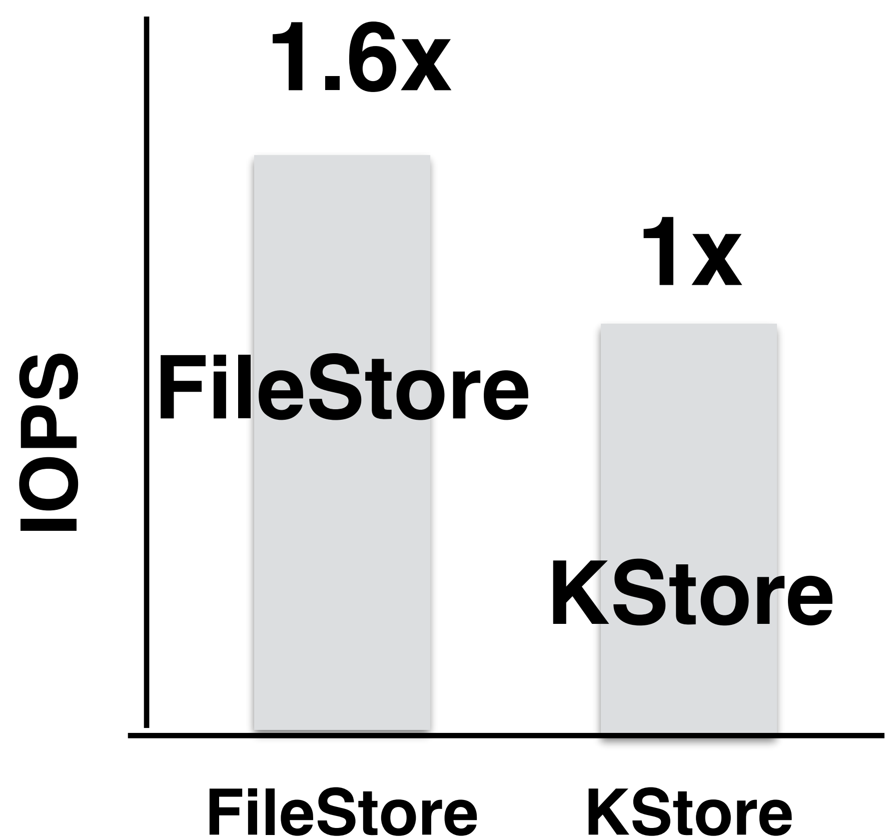
Average IOPS



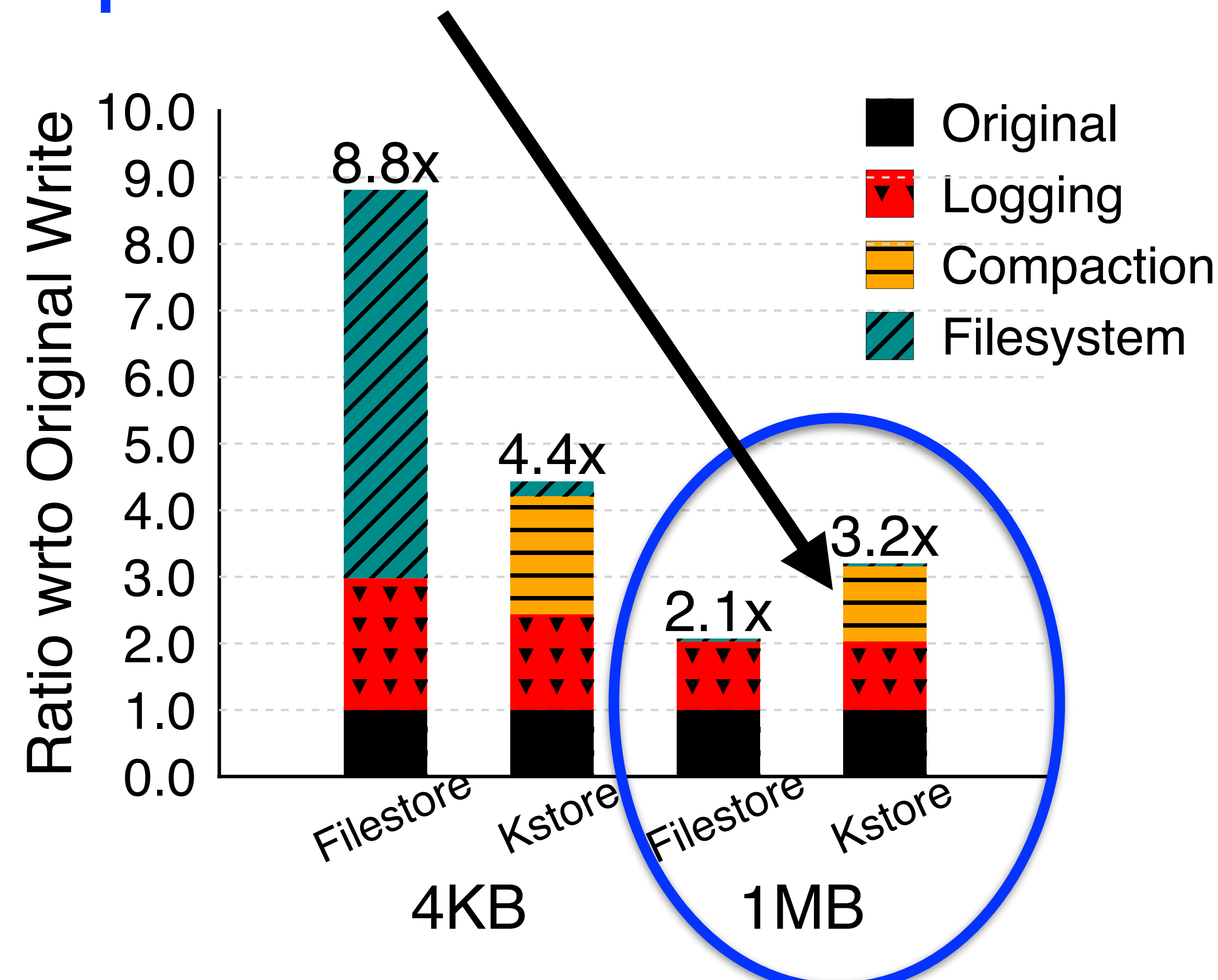
Write Traffic Breakdown

Performance

- Large Write (1MB)
  - FileStore outperforms KStore by 1.6x
  - Write amplification by compaction



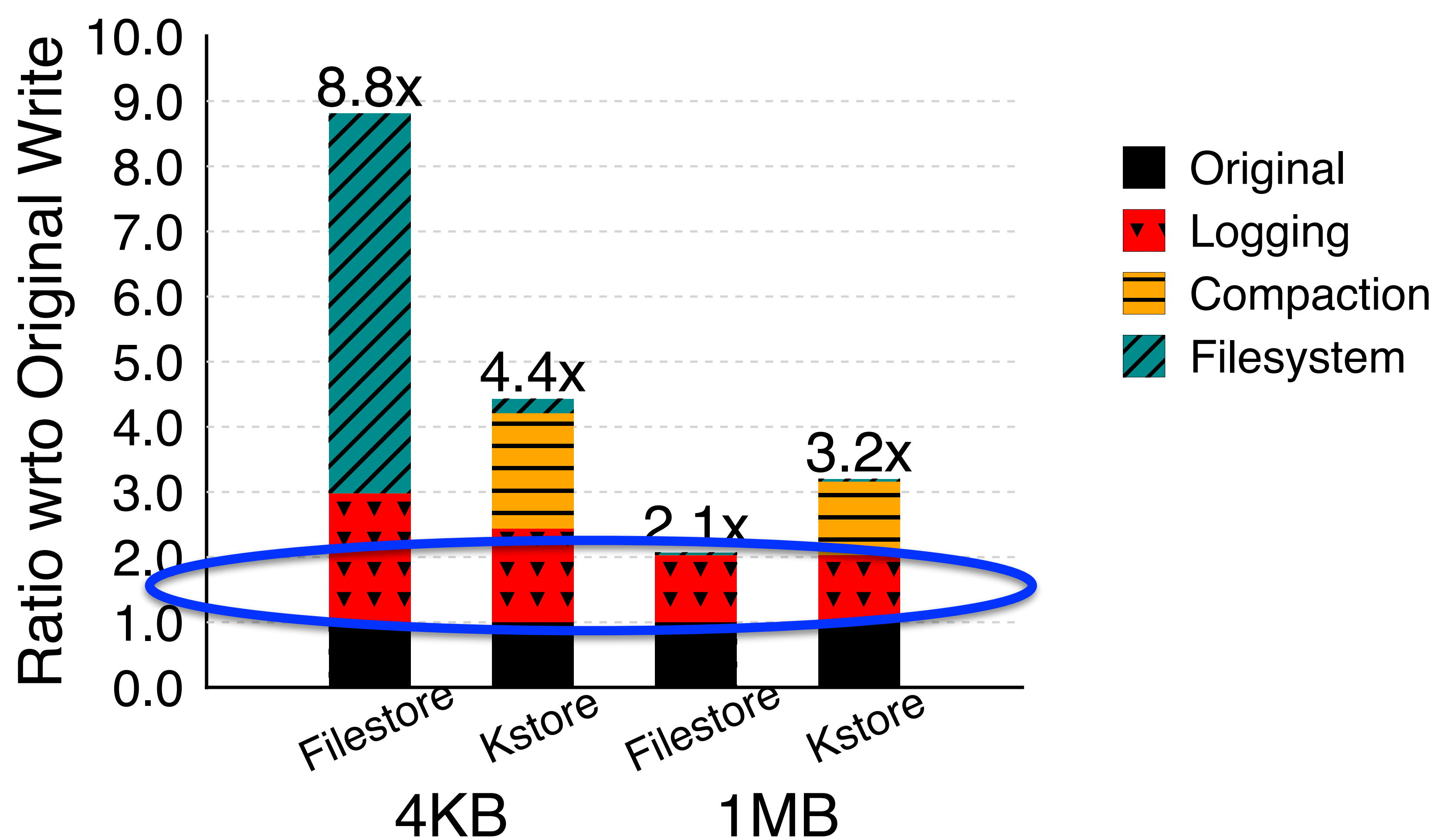
Average IOPS



Write Traffic Breakdown

Performance

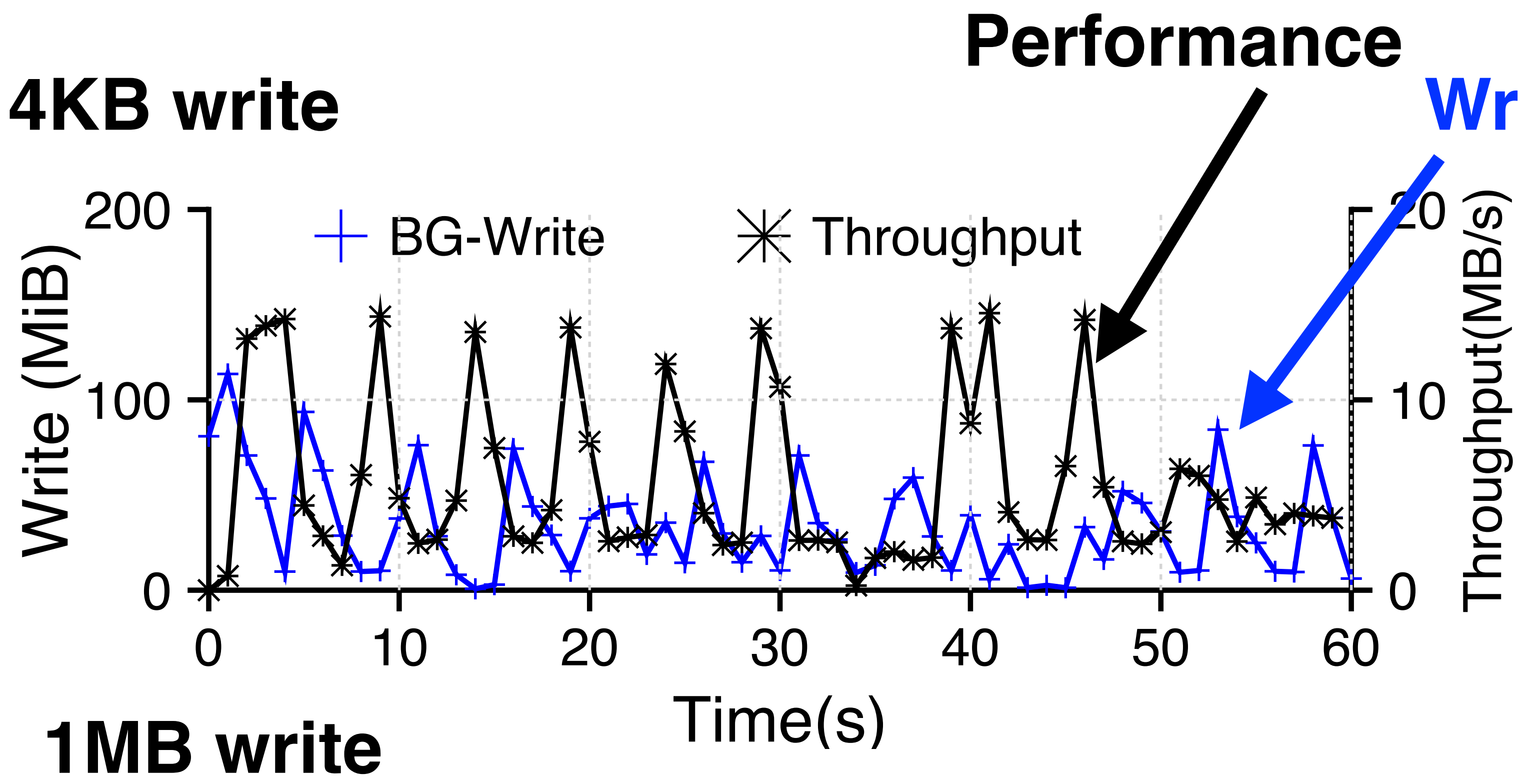
- Lack of atomic update support in file systems
- Double-write penalty of logging
- Halve bandwidth in large writes

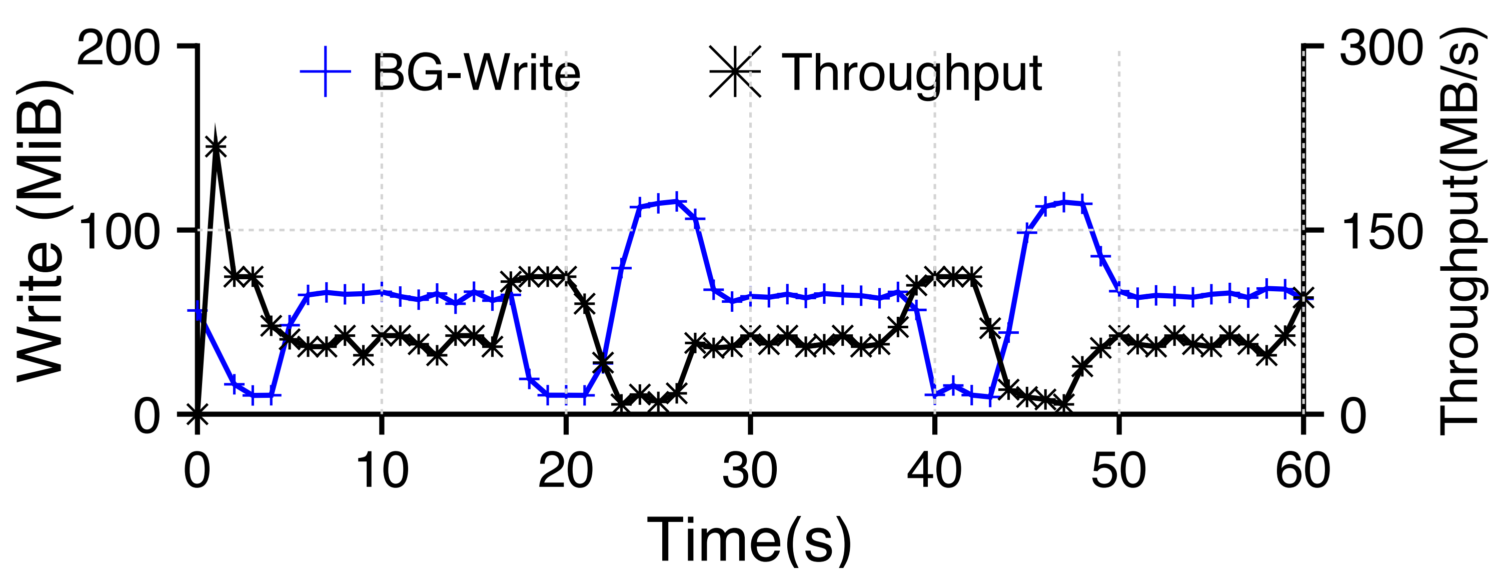


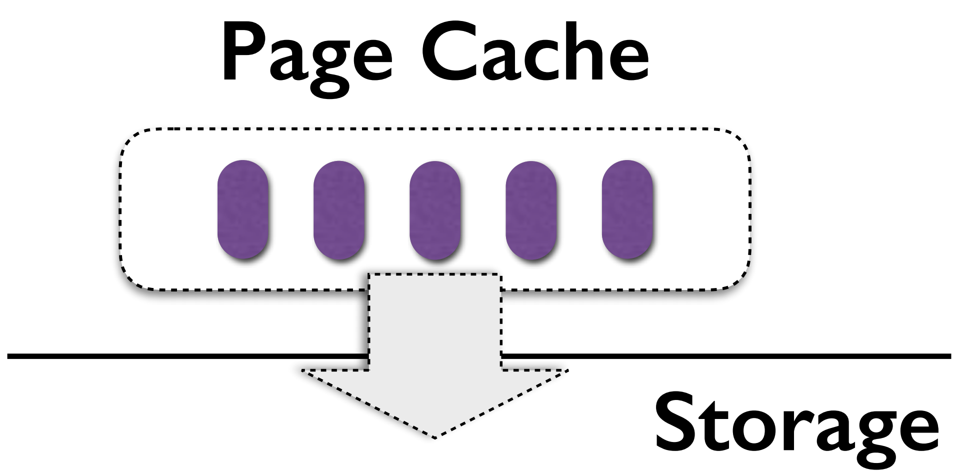
Write Traffic Breakdown

QoS

- FileStore





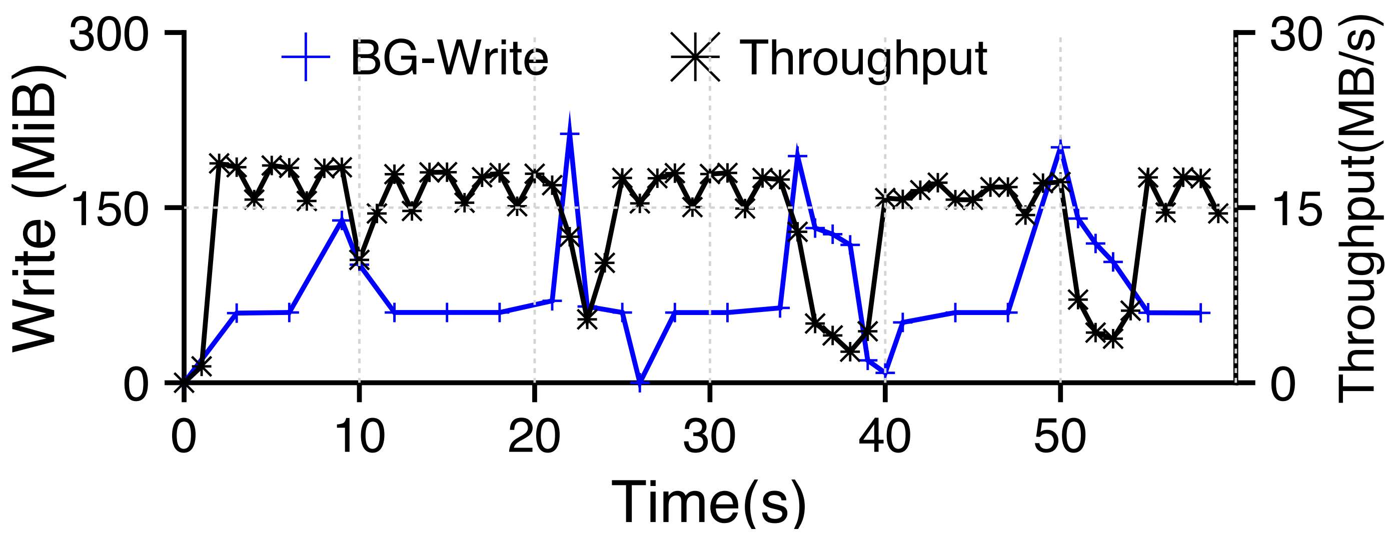


Periodic Flush w. Buffered I/O
Transaction Entanglement

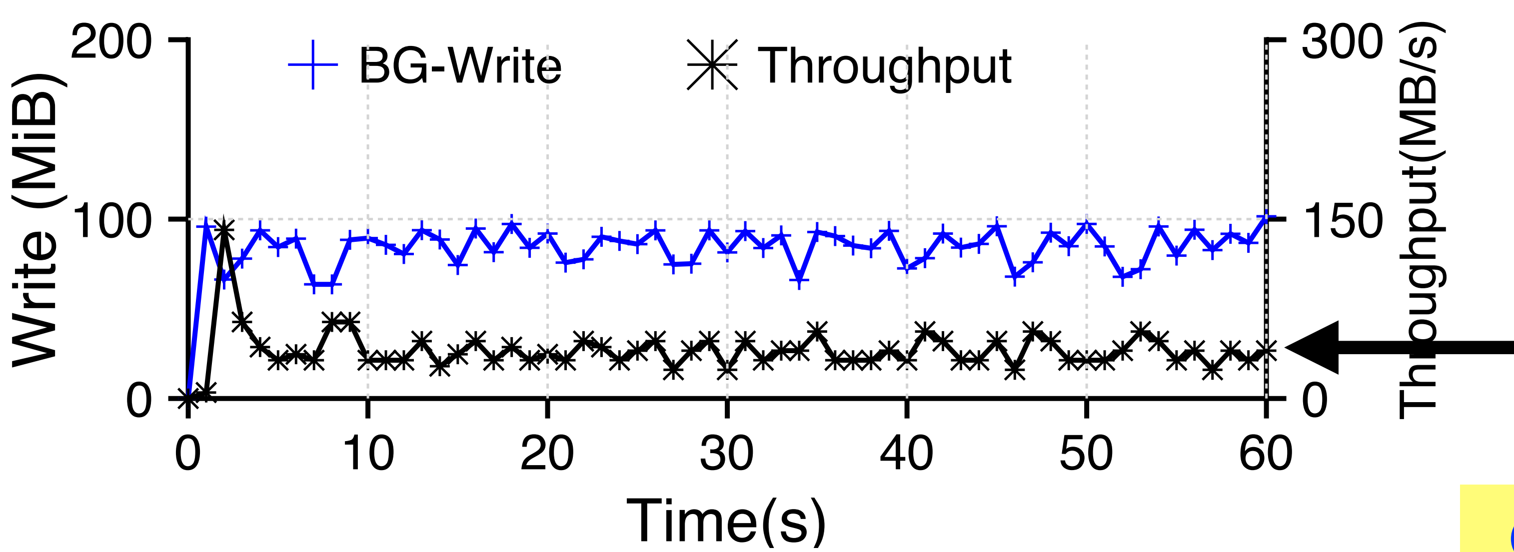
QoS

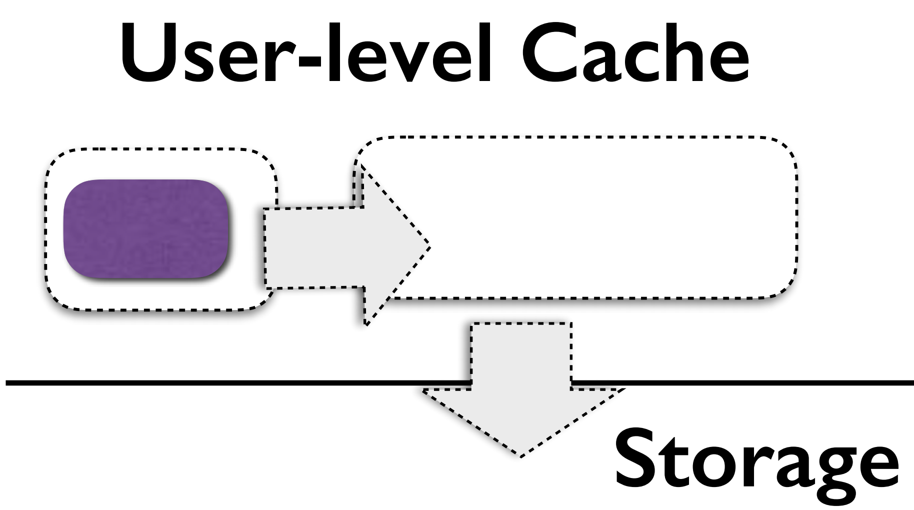
- KStore

4KB write



1MB write





Frequent Compaction
Write amplification by merge

Throughput: 40MB/s

Consistently Poor

Summary

- Performance penalties of file systems
  - Small objects seriously suffer from write amplification caused by filesystem metadata
  - Large writes are sensitive to write traffic increase by Logging in common, and frequent compaction in packing architecture.
  - Buffered I/O and out-of-control flush mechanism in file systems makes it challenging to support QoS.

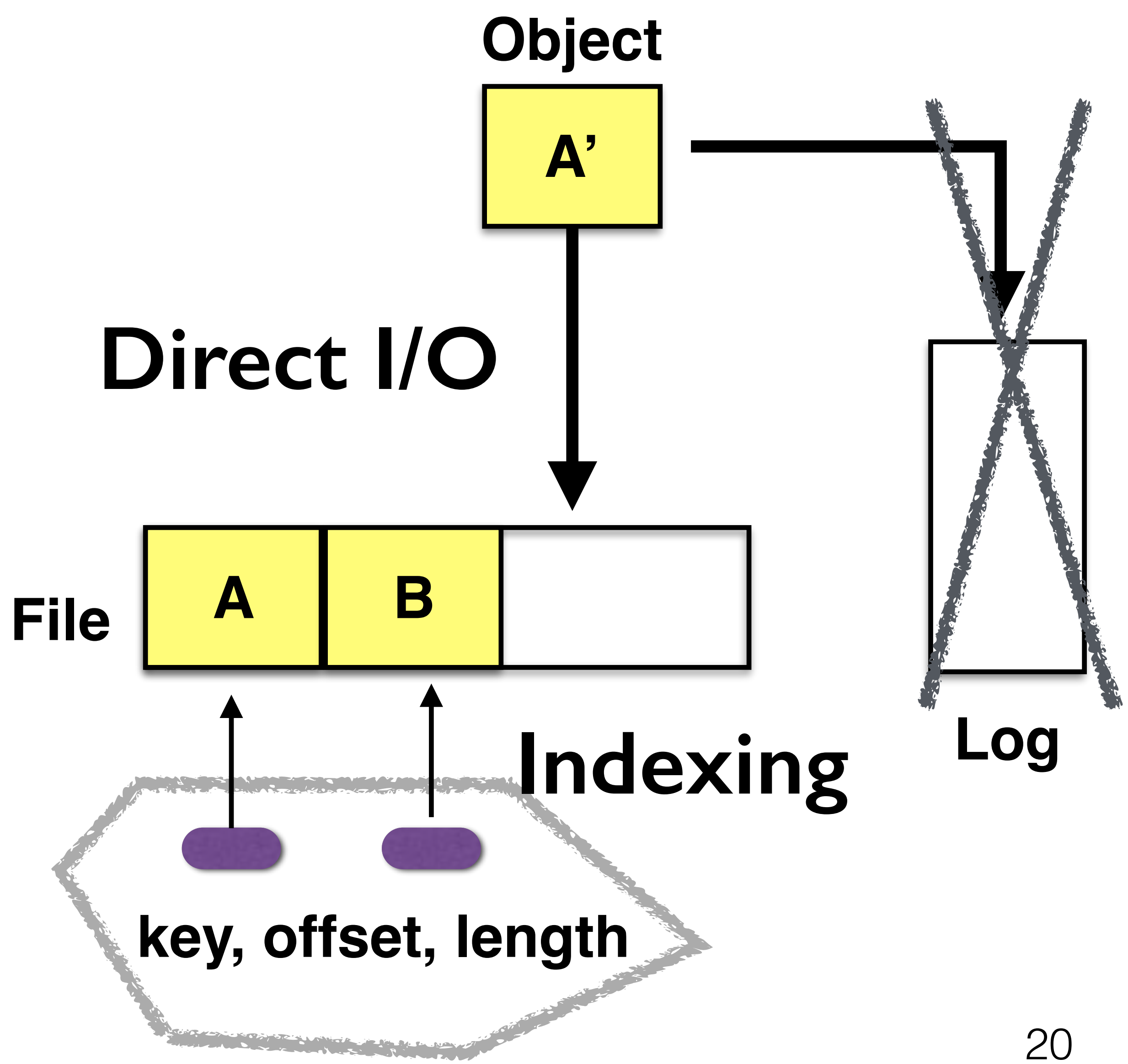
Contents

SwimStore

- Shadowing with Immutable Metadata Store
- Provide consistently excellent performance for all object sizes running over a file system

SwimStore

- Strategy 1. In-file shadowing

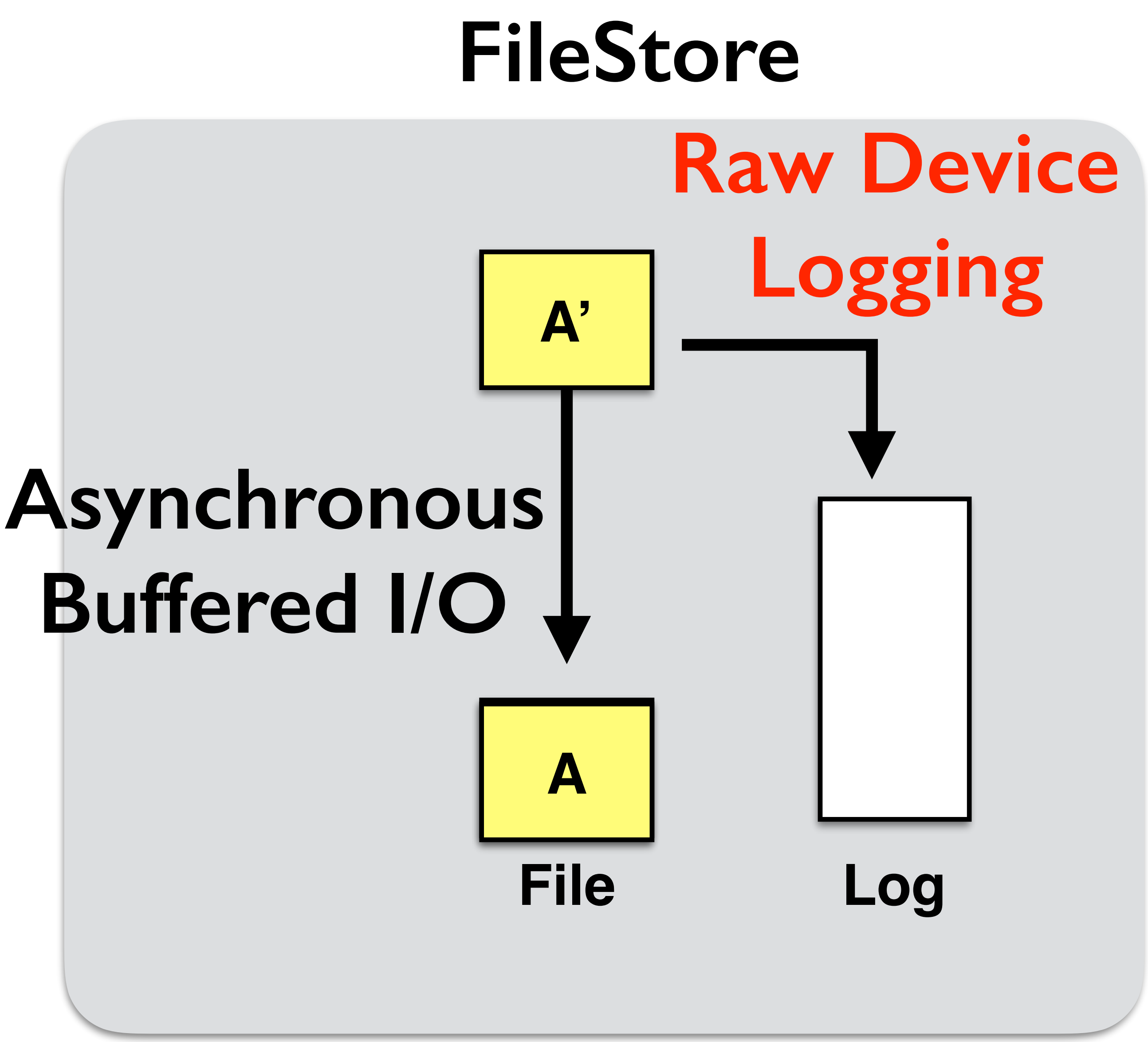


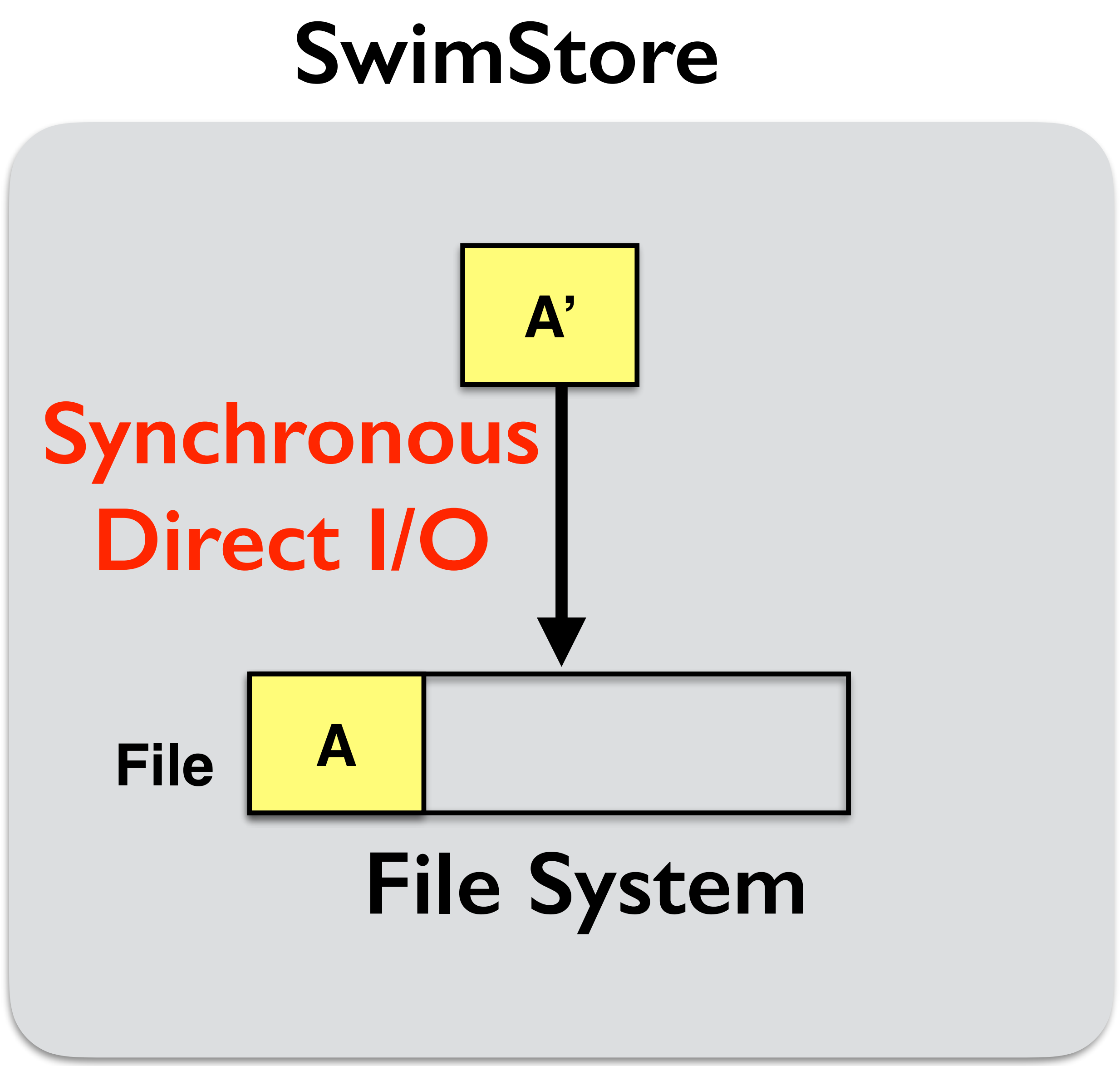
Problems

- ~~Filesystem metadata overhead~~
- ~~Double-write penalty~~
- ~~Performance fluctuation~~
- ~~Compaction cost~~

SwimStore

- Strategy 1. In-file shadowing

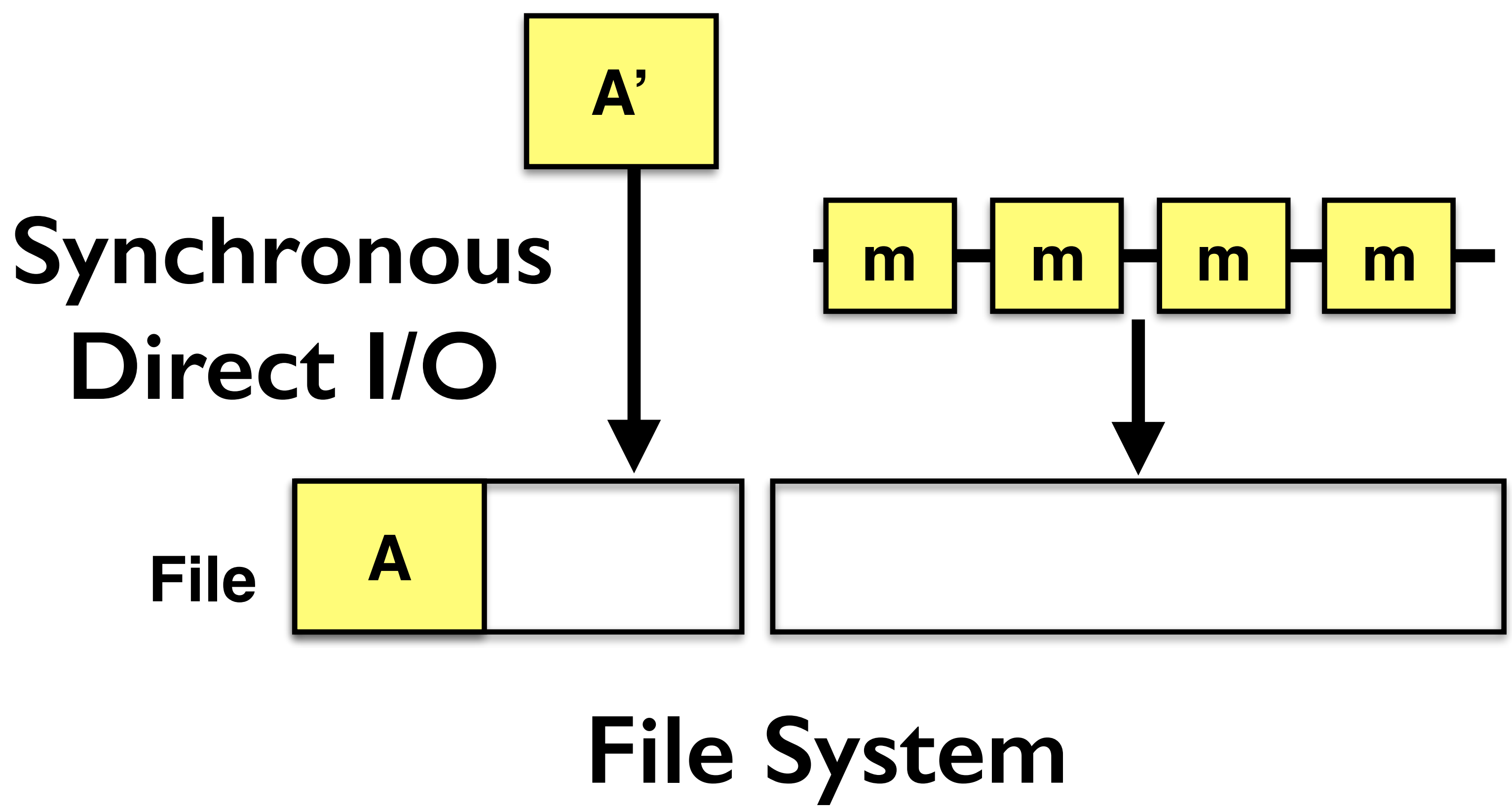




User-facing Latency increases!

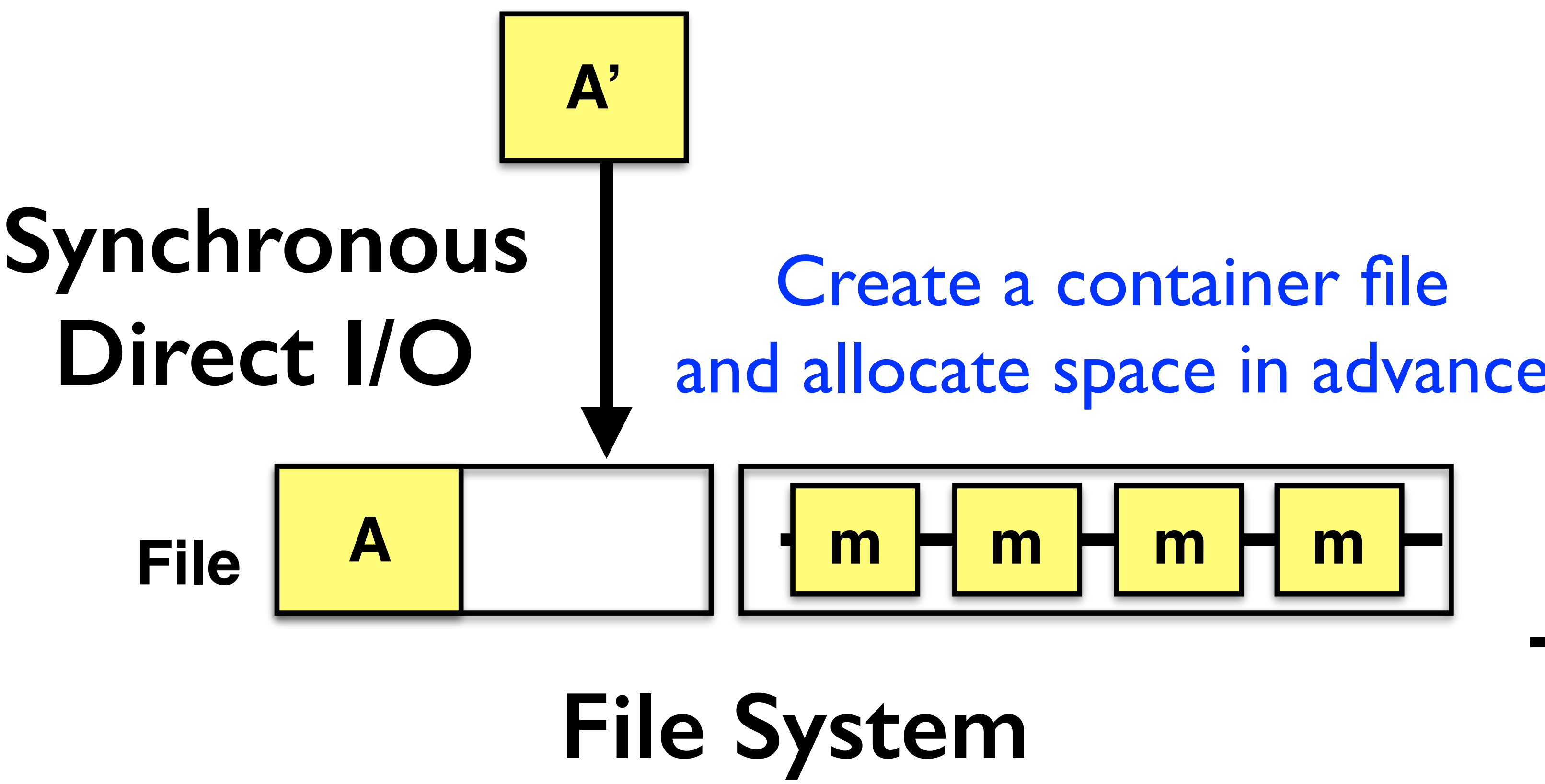
SwimStore

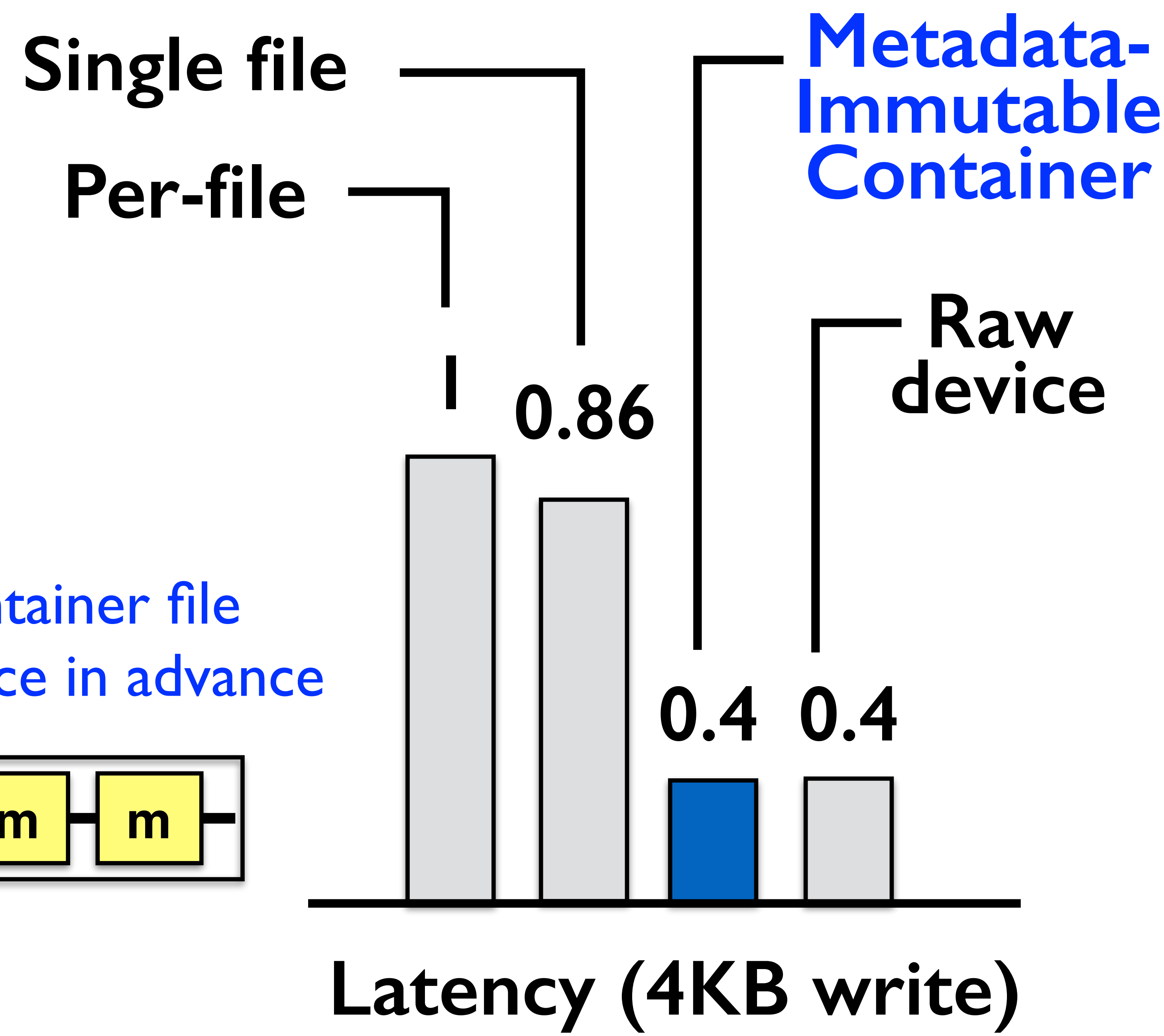
- File system access is slower than raw device access
  - File system metadata (e.g., inode, allocation bitmap, etc.)
  - Transaction entanglement



SwimStore

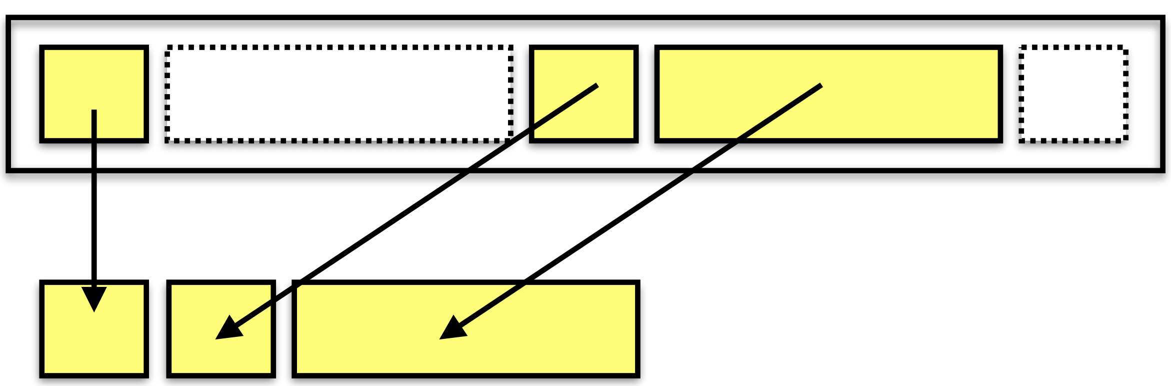
- Strategy 2. Metadata-Immutable Container





SwimStore

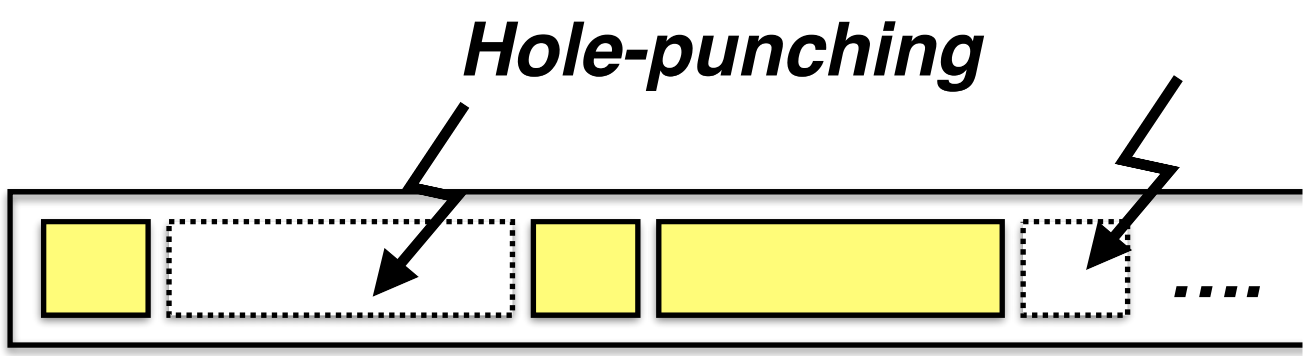
- Strategy 3. Hole-punching with Buddy-like Allocation



Shadowing technique requires the recycling of obsolete data space

SwimStore

- Strategy 3. Hole-punching with Buddy-like Allocation



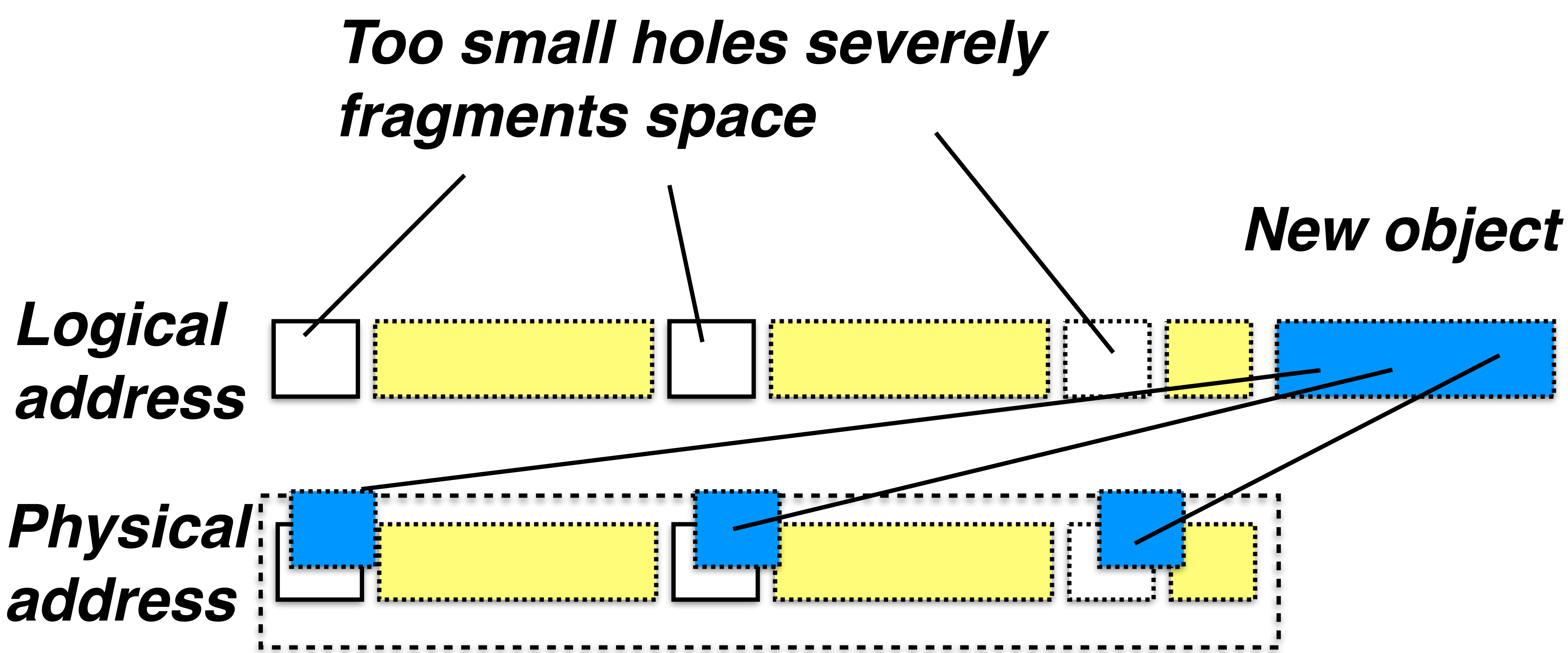
Opportunities

(+) Filesystem has **“infinite address space”**

(+) Filesystem provides “**physical space reclamation**” with punch-hole

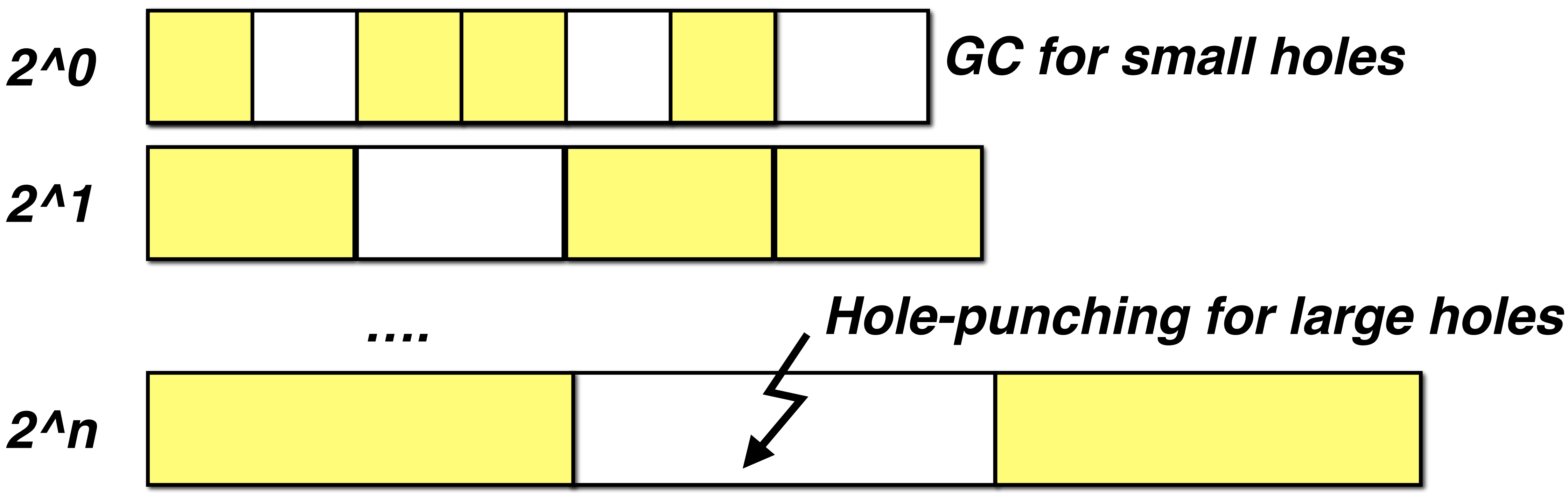
SwimStore

- Strategy 3. Hole-punching with Buddy-like Allocation



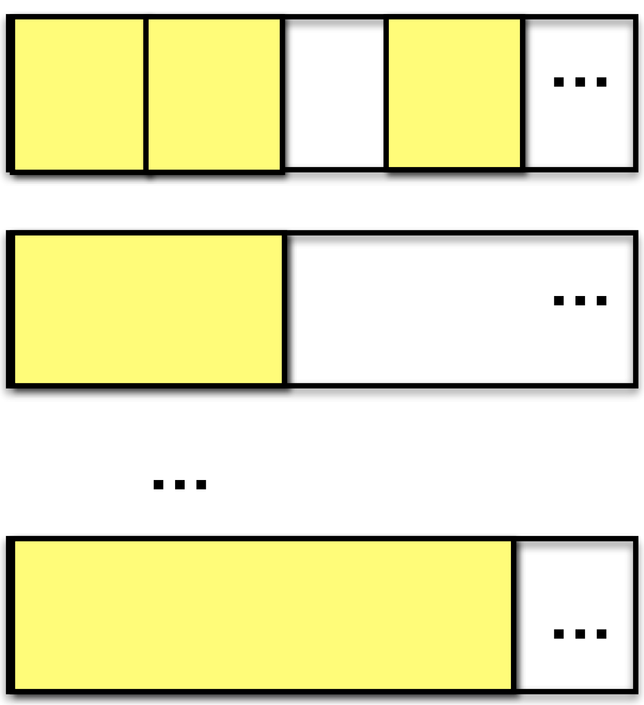
SwimStore

- Strategy 3. Hole-punching with Buddy-like Allocation

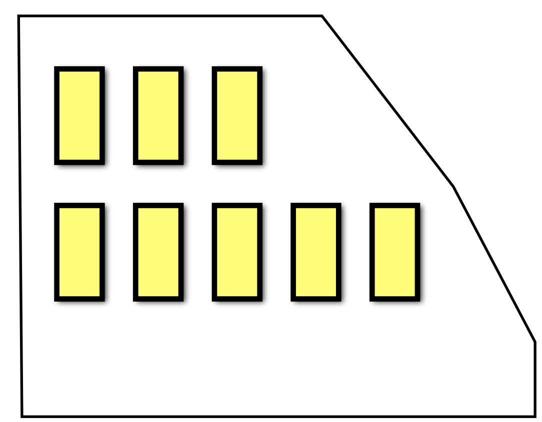


SwimStore

- Architecture



Container File Pool



LSM-Tree (LevelDB)

Metadata
(Indexing, attributes, etc.)

Intent Log
(metadata, checksum)

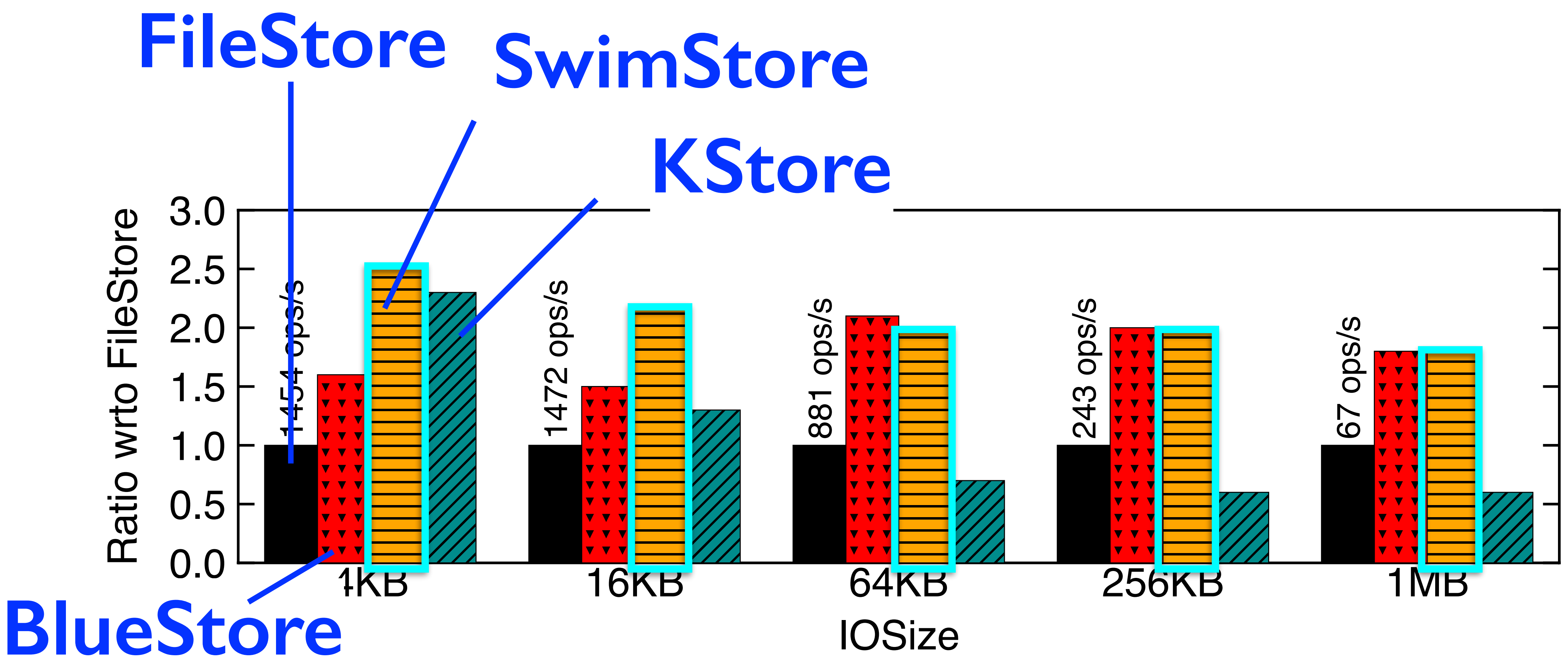
Contents

Experimental Setup

- Ceph 12.01, C++ 12K LOC
- Amazon EC2 Clusters
- Intel Xeon quad-core
- 32GB DRAM
- 256 GB SSD x 2
- Ubuntu Server 16.04
- File System : XFS (recommended in Ceph)
- Backend: FileStore, KStore, BlueStore, SwimStore
- Benchmark: Rados
- Metric: IOPS, throughput, write traffic

Performance Evaluation

- IOPS

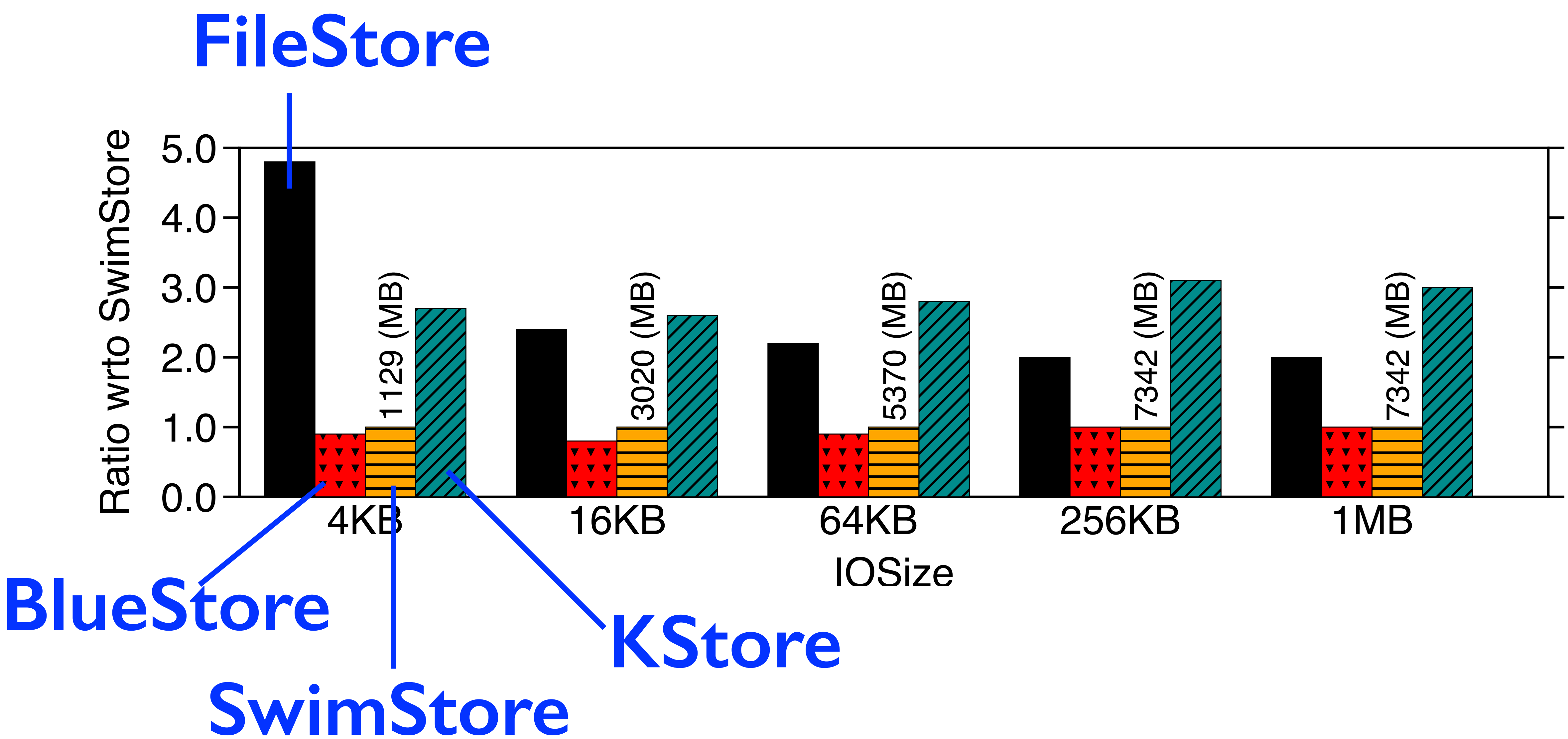


Small Write
2.5x *better than FileStore*
1.6x *better than BlueStore*
1.1x *better than KStore*

Large Write
1.8x *better than FileStore*
3.1x *better than KStore*

Performance Evaluation

- Write Traffic



Contents

- Motivation
- Problem Analysis
- Solution
- Performance Evaluation
- **Conclusion**

Conclusion

- Explore design patterns to build an object store atop a local file system
- SwimStore: a new backend object store
  - In-file shadowing
  - Immutable metadata container
  - Hole-punching with buddy-like allocation
- Provide high performance and little performance variations
- Retain all benefits of the file system

Thank you