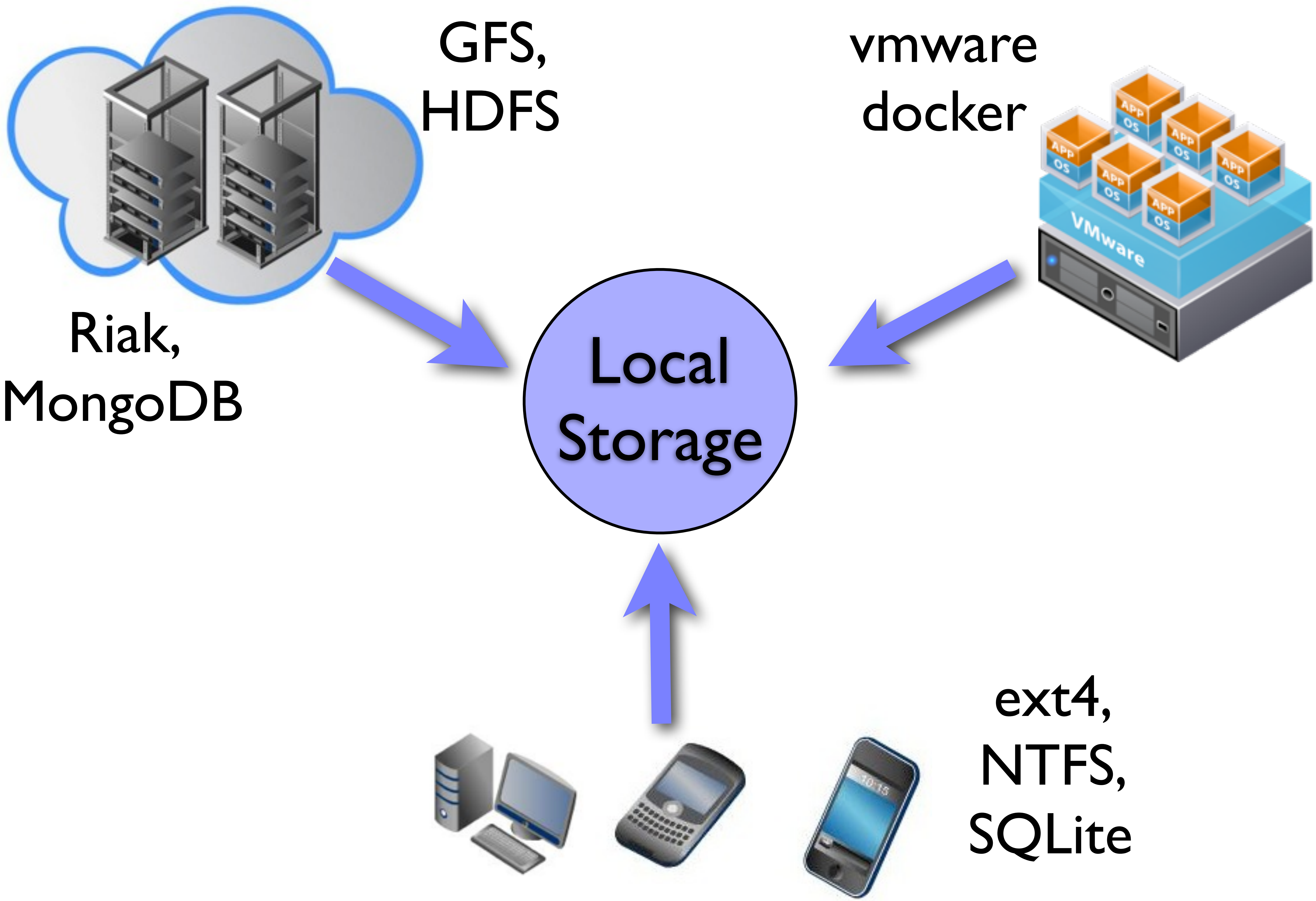
Physical Separation in Modern Storage Systems

Lanyue Lu

Committee:
Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau
Shan Lu, Michael Swift, Xinyu Zhang
University of Wisconsin - Madison

Local Storage Systems Are Important



Data Layout of Storage Systems

Data layout is fundamental

- how to organize data on disks and in memory
- impact both reliability and performance

Locality is the key

- store relevant data together
- locality is pursued in various storage systems
  - file systems, key-value stores, databases
- better performance (caching and prefetching)
- high space utilization
- optimize for hard drives

Problems of Data Locality

New environments

- fast storage hardware (e.g., SSDs)
- servers with many cores and large memory
- sharing infrastructure is the reality
  - virtualization, containers, data centers

Unexpected entanglement

- shared failures (e.g., VMs, containers)
- bundled performance (e.g., apps)
- lack flexibility to manage data differently

New Technique: Physical Separation

Redesign data layout

- rethink existing data layouts
- key: separate data structures
- apply in both file systems and key-value stores

Many new benefits

- IceFS: disentangle structures and transactions
  - isolated failures, faster recovery
  - customized performance
- WiscKey: key-value separation
  - minimize I/O amplification
  - leverage devices' internal parallelism

Research Contributions

A study of Linux file system evolution 1

- the first comprehensive file-system study
- published in FAST '13 (best paper award)

Physical disentanglement in IceFS 2

- localized failure, localized recovery
- specialized journaling performance
- published in OSDI '14

Key-value separation in WiscKey 3

- an SSD-conscious LSM-tree
- over 100x performance improvement
- submitted to FAST '16

Outline

Introduction

Disentanglement in IceFS

- File system Disentanglement
- The Ice File System
- Evaluation

Key-Value Separation in WiscKey

- Key-value Separation Idea
- Challenges and Optimization
- Evaluation

Conclusion

Isolation Is Important

Reliability

- independent failures and recovery

Performance

- isolated performance

Isolation at various scenarios

- computing: virtual machines, Linux containers
- security: BSD jail, sandbox
- cloud: multi-tenant systems

File Systems Lack Isolation

Local file systems are core building blocks

- manage user data
- long-standing and stable
- foundation for distributed file systems

Existing abstractions provide logical isolation

- file, directory, namespace
- just illusion

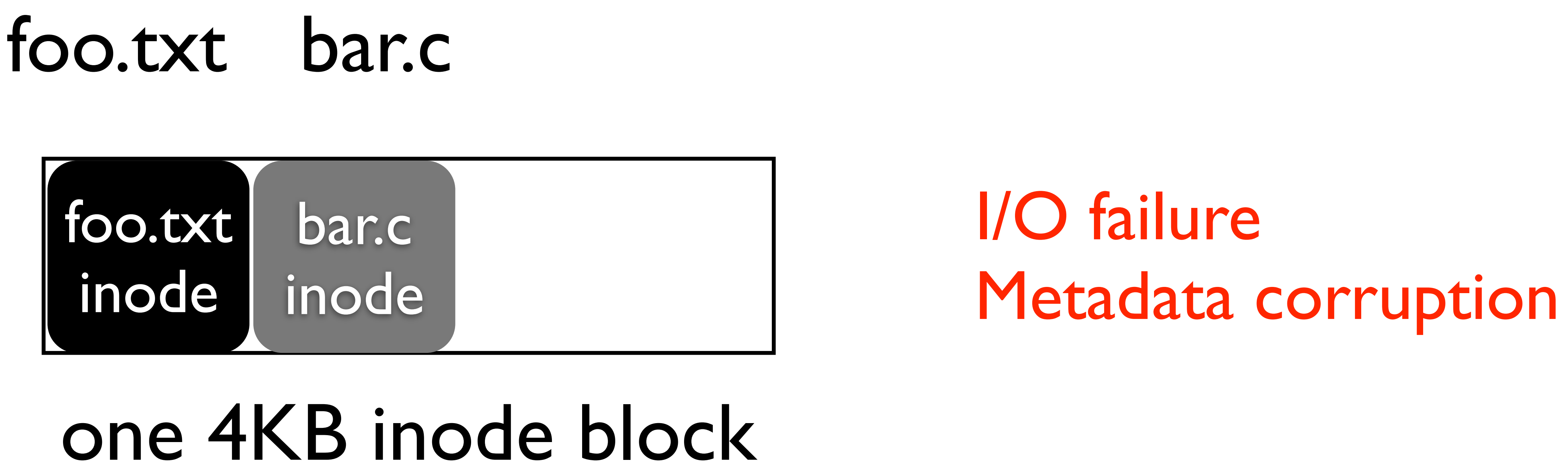
Physical entanglement in local file systems prevents isolation

- entangled data structures and transactions

Metadata Entanglement

Shared metadata for multiple files

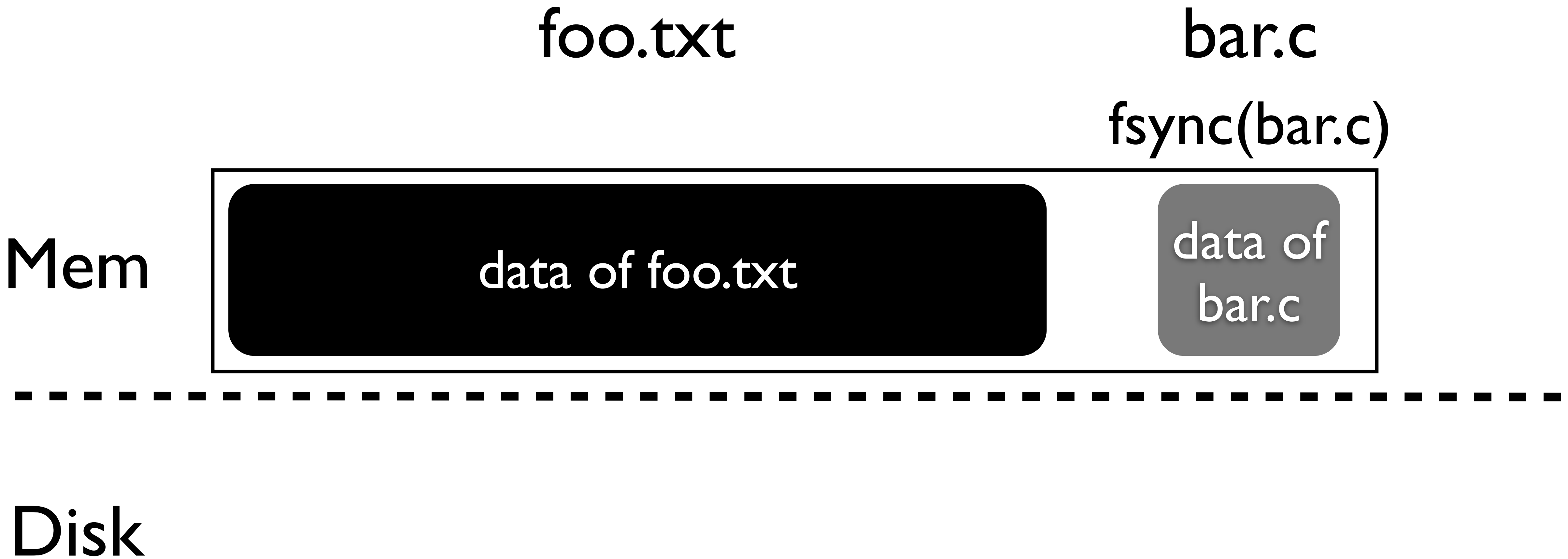
- e.g., multiple files share one inode block
- many shared structures: bitmap, directory block



Problem: faults in shared structures lead to shared failures and recovery

Transaction Entanglement

A shared transaction for all updates



Problem: shared transactions lead to entangled performance

Our Solution: IceFS

Propose a data container abstraction: cube

Disentangle data structures and transactions

Provide reliability and performance isolation

Benefits for local file systems

- isolated failures for data containers
- up to 8x faster localized recovery
- up to 50x higher performance

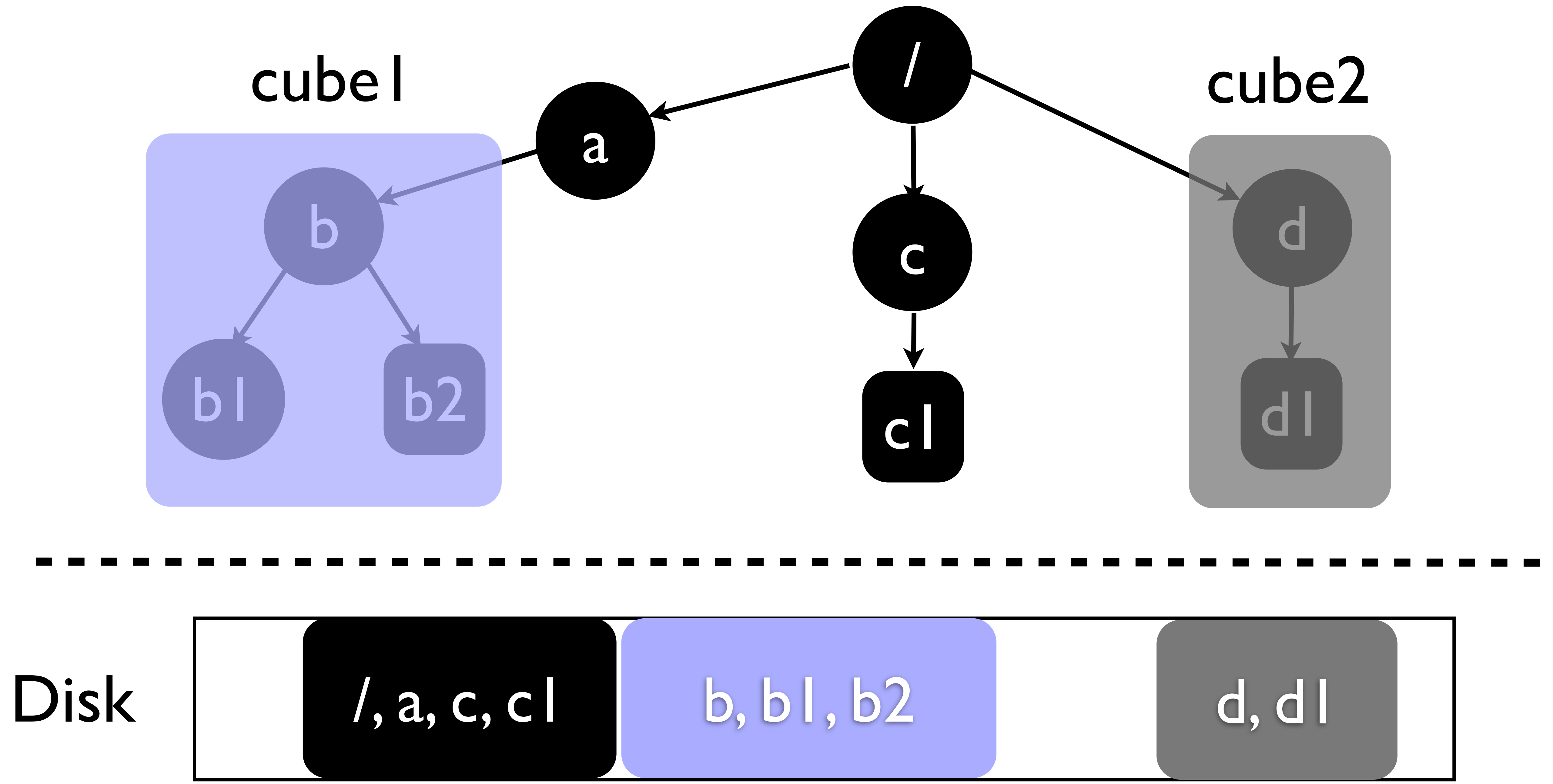
Benefits for high-level services

- virtualized systems: reduce the downtime over 5x
- HDFS: improve the recovery efficiency over 7x

Data Container Abstraction: Cube

An isolated directory in a file system

- ➡ physically disentangled on disk and in memory



Principles of Disentanglement

No shared physical resources

- no shared metadata: e.g., block groups
- no shared disk blocks or buffers

No dependency

- partition linked lists or trees
- avoid directory hierarchy dependency

No entangled updates

- use separate transactions
- enable customized journaling modes

Outline

Introduction

Disentanglement in IceFS

- File system Disentanglement
- The Ice File System
- Evaluation

Key-Value Separation in WiscKey

- Key-value Separation Idea
- Challenges and Optimization
- Evaluation

Conclusion

IceFS Overview

A data container based file system

- isolated reliability and performance for containers

Disentanglement techniques

- physical resource isolation
- directory indirection
- transaction splitting

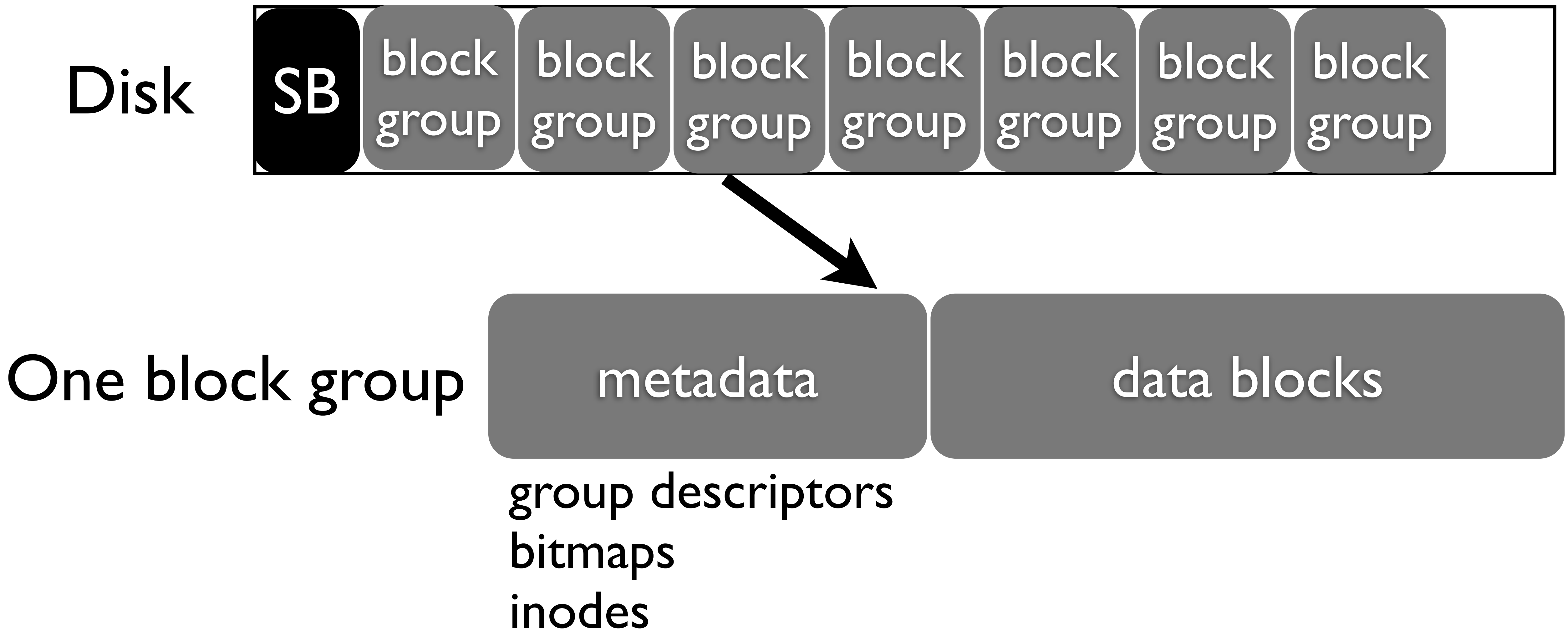
A prototype based on Ext3

- local file system: Ext3/JBD
- kernel: VFS
- user level tool: e2fsprogs

Ext3 Disk Layout

A disk is divided into block groups

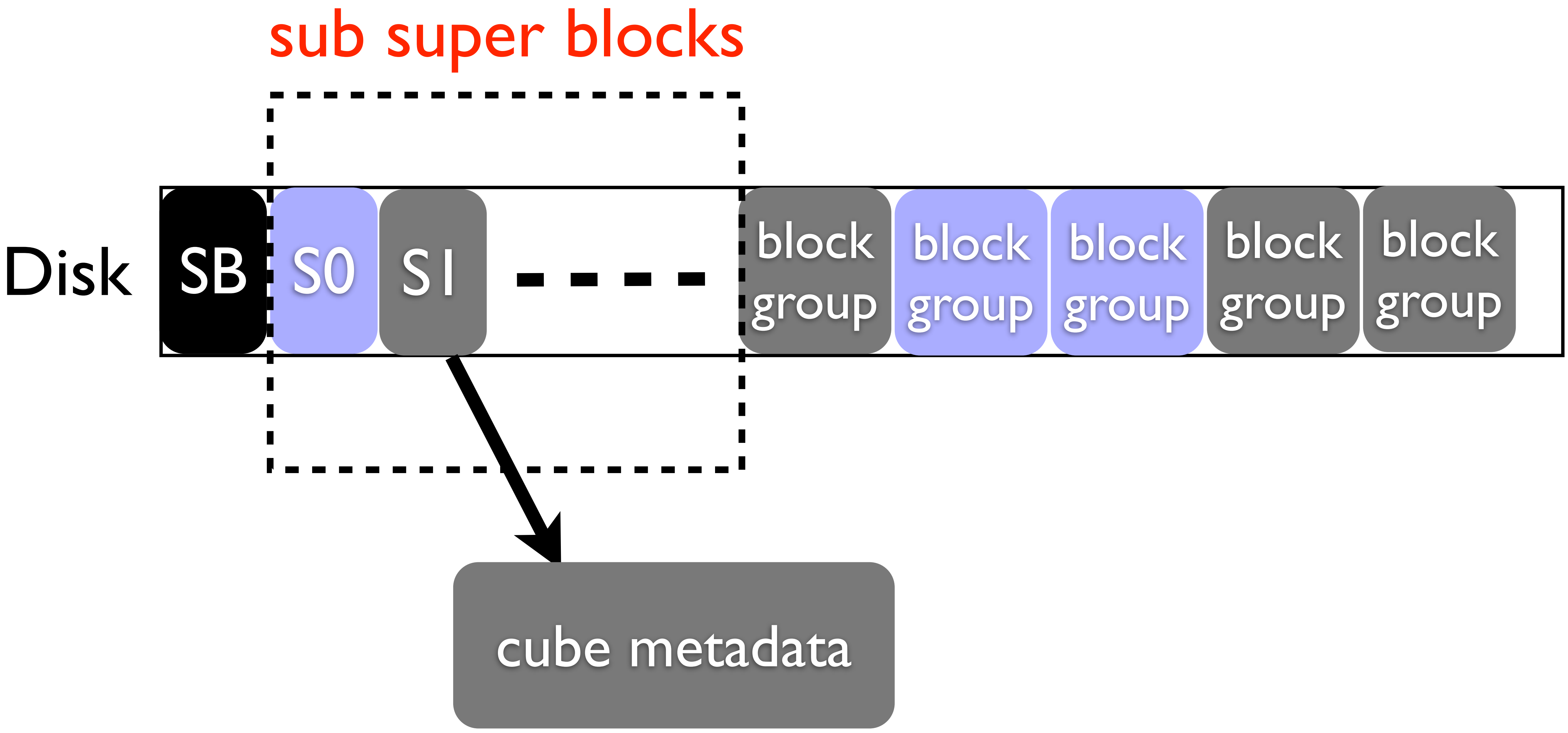
- physical partition for disk locality



IceFS Disk Layout

Each cube has isolated metadata

- sub-super block (Si) and isolated block groups



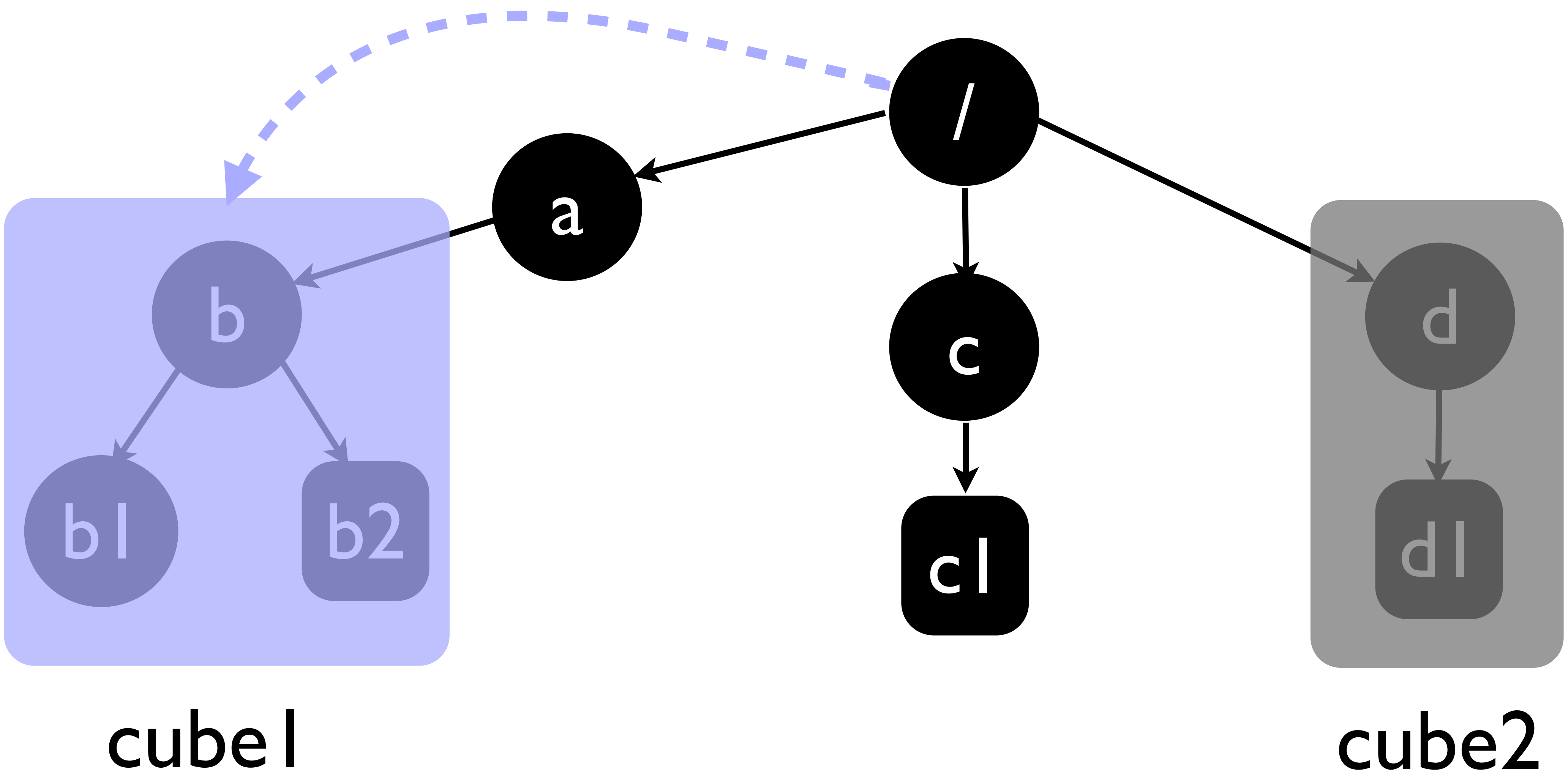
Directory Indirection

1. load cube pathnames from sub-super blocks

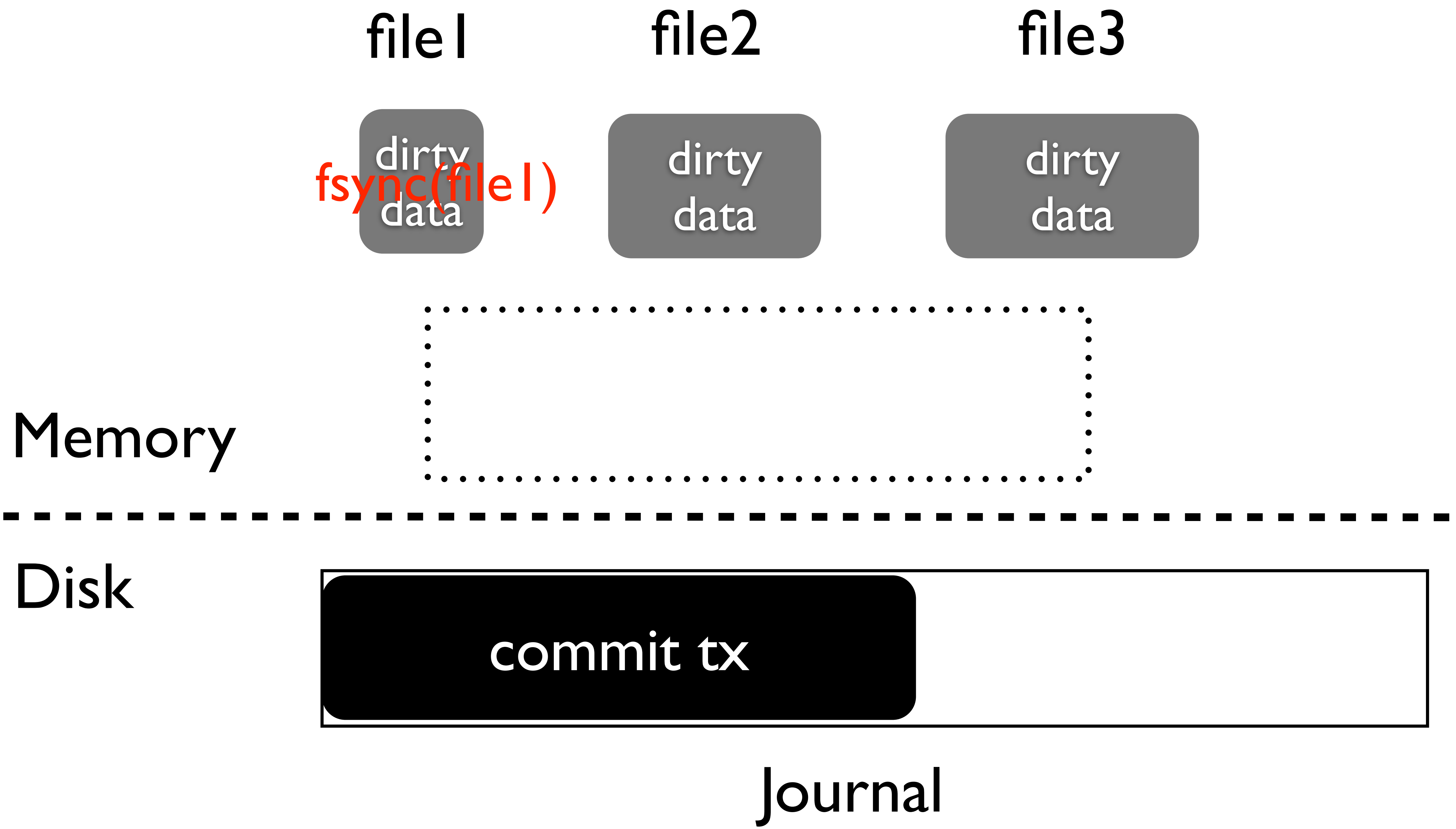
/a/b/, cube1 dentry
/d/, cube2 dentry
... ...

2. pathname prefix match

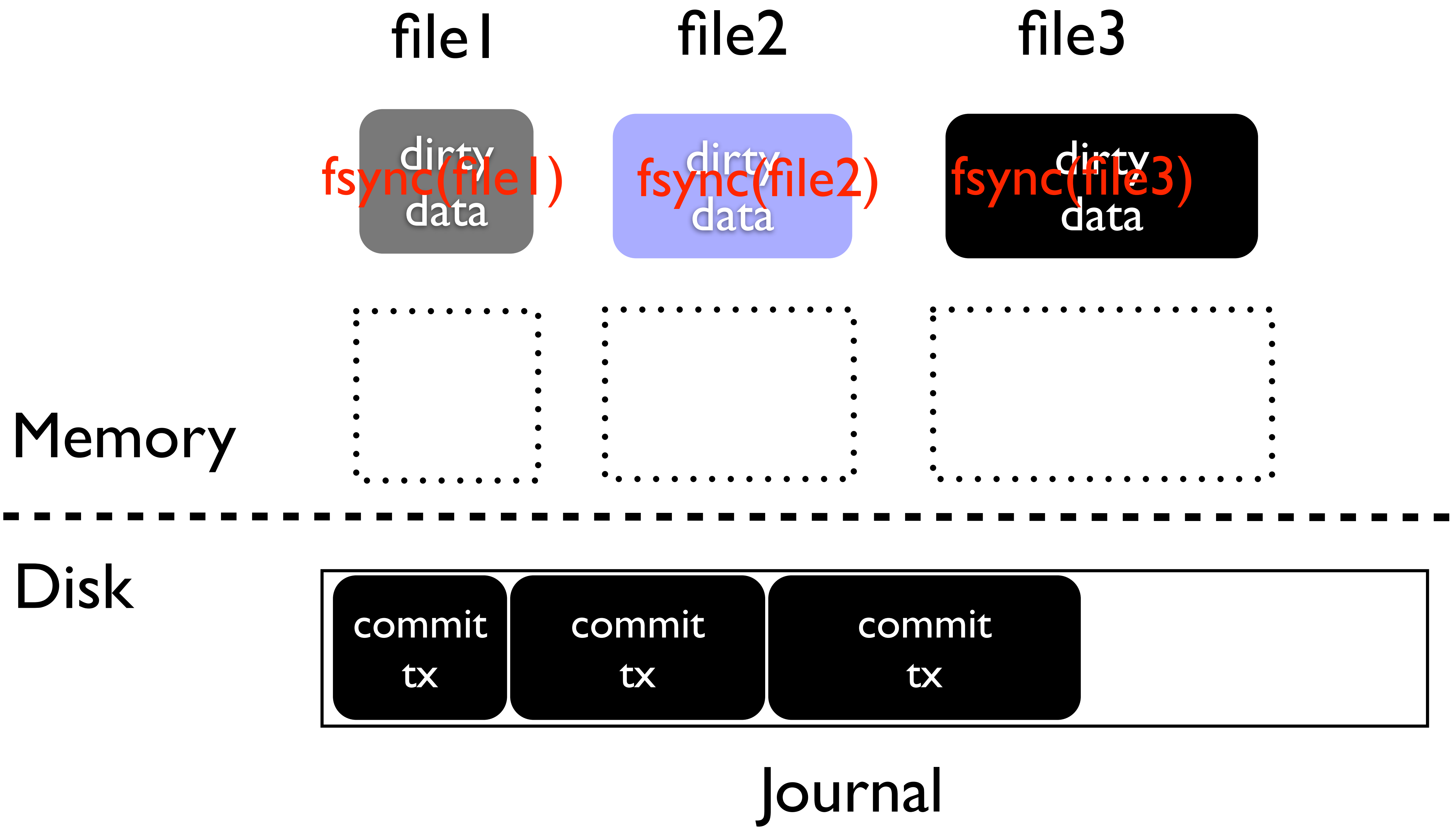
read file "/a/b/b2"
match cube1
jump to cube1 top directory



Ext3/4 Transaction



IceFS Transaction Splitting



Benefits of Disentanglement

Localized reactions to failures

- per-cube read-only and crash
- encourage more runtime checking

Localized recovery

- only check faulty cubes
- offline and online

Specialized journaling

- concurrent and independent transactions
- diverse journal modes (e.g., no journal, no fsync)

Outline

- Introduction
- Disentanglement in IceFS
  - File system Disentanglement
  - The Ice File System
  - Evaluation
- Key-Value Separation in WiscKey
  - Key-value Separation Idea
  - Challenges and Optimization
  - Evaluation
- Conclusion

Evaluation

Does IceFS isolate failures ?

- inject around 200 faults
- per-cube failure (read-only or crash) in IceFS

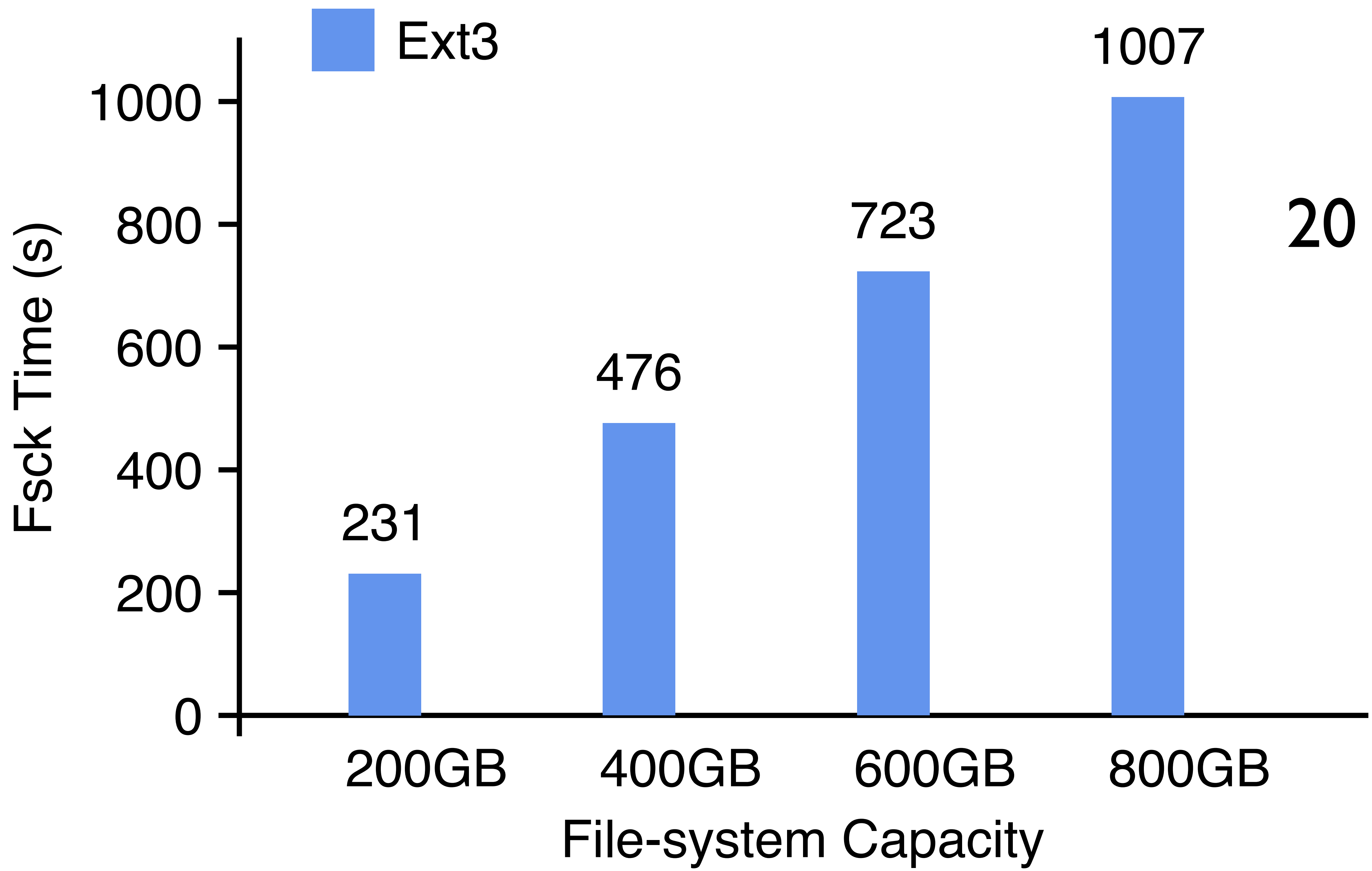
Evaluation

Does IceFS isolate failures ?

- inject around 200 faults
- per-cube failure (read-only or crash) in IceFS

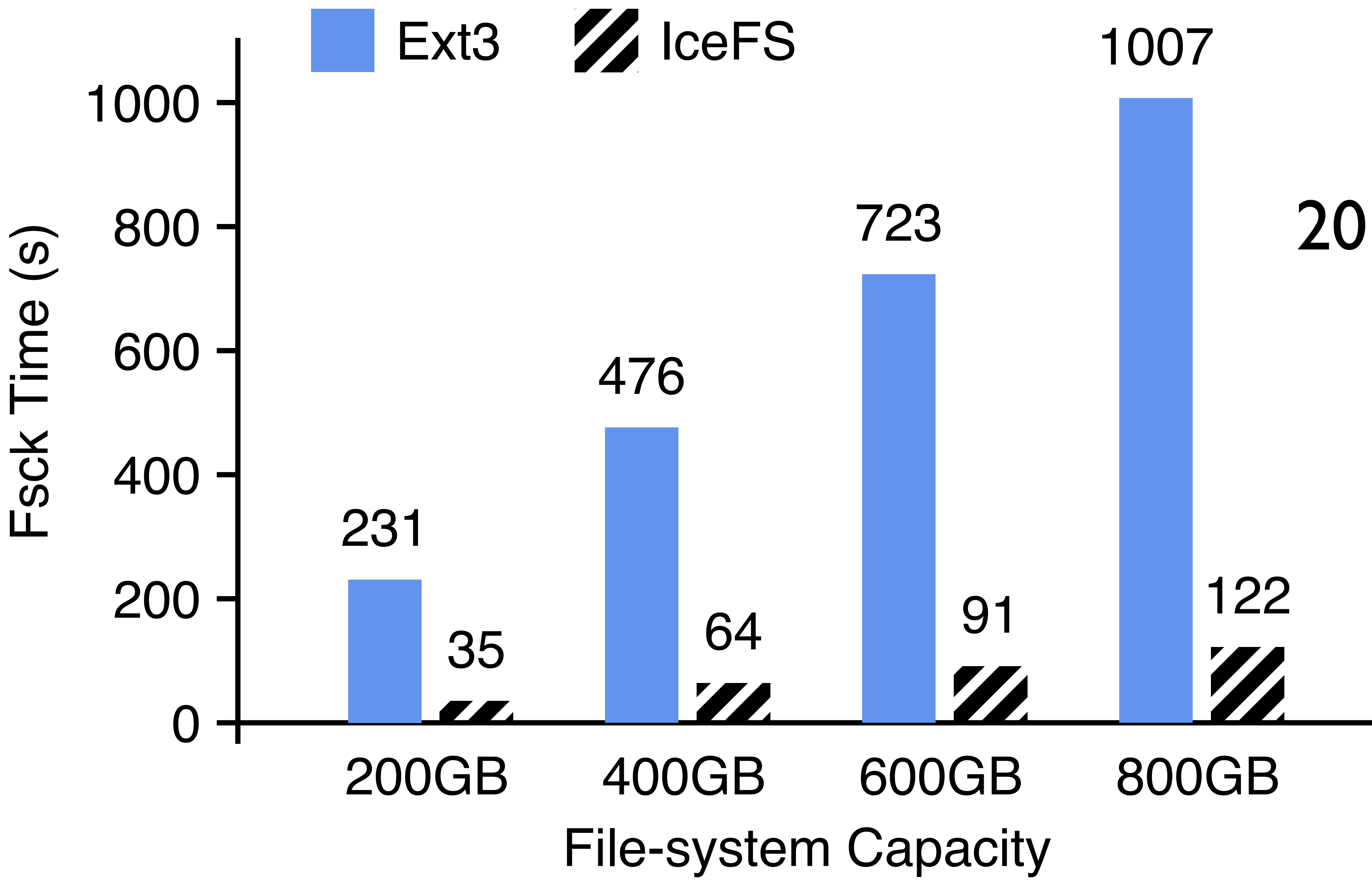
Does IceFS have faster recovery ?

Recovery In Ext3



Ext3:
20 directories

Fast Recovery In IceFS



Ext3:
20 directories

IceFS:
20 cubes

Partial recovery for a cube (up to 8x)

Evaluation

Does IceFS isolate failures ?

- inject around 200 faults
- per-cube failure (read-only or crash) for IceFS

Does IceFS have faster recovery ?

- independent recovery for a cube

Evaluation

Does IceFS isolate failures ?

- inject around 200 faults
- per-cube failure (read-only or crash) for IceFS

Does IceFS have faster recovery ?

- independent recovery for a cube

Does IceFS have better performance ?

Workloads

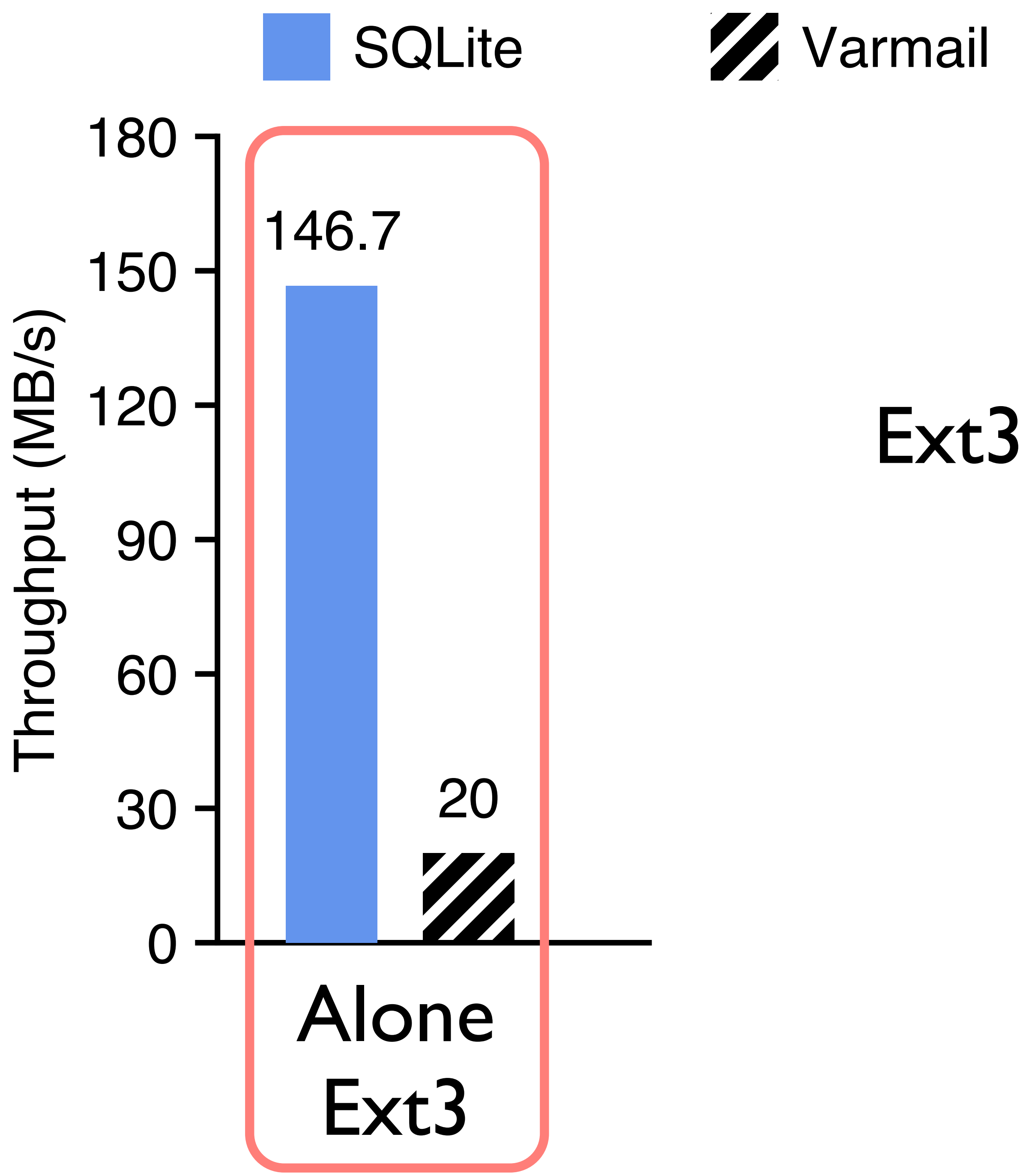
SQLite

- a database application
- sequentially write large key/value pairs
- asynchronous

Varmail

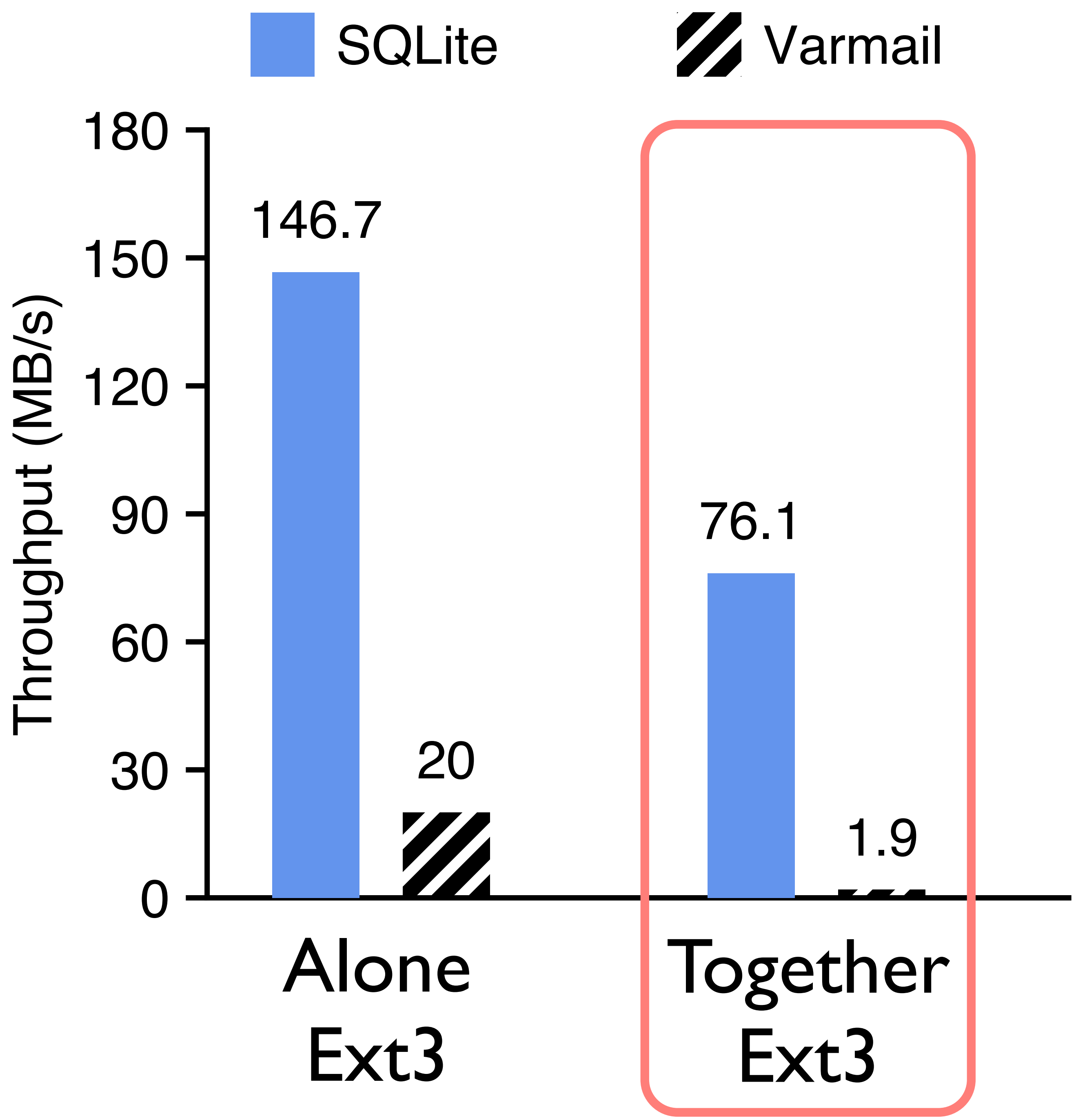
- an email server workload
- randomly write small blocks
- fsync after each write

Ext3 Journaling



Ext3 runs with 2 directories

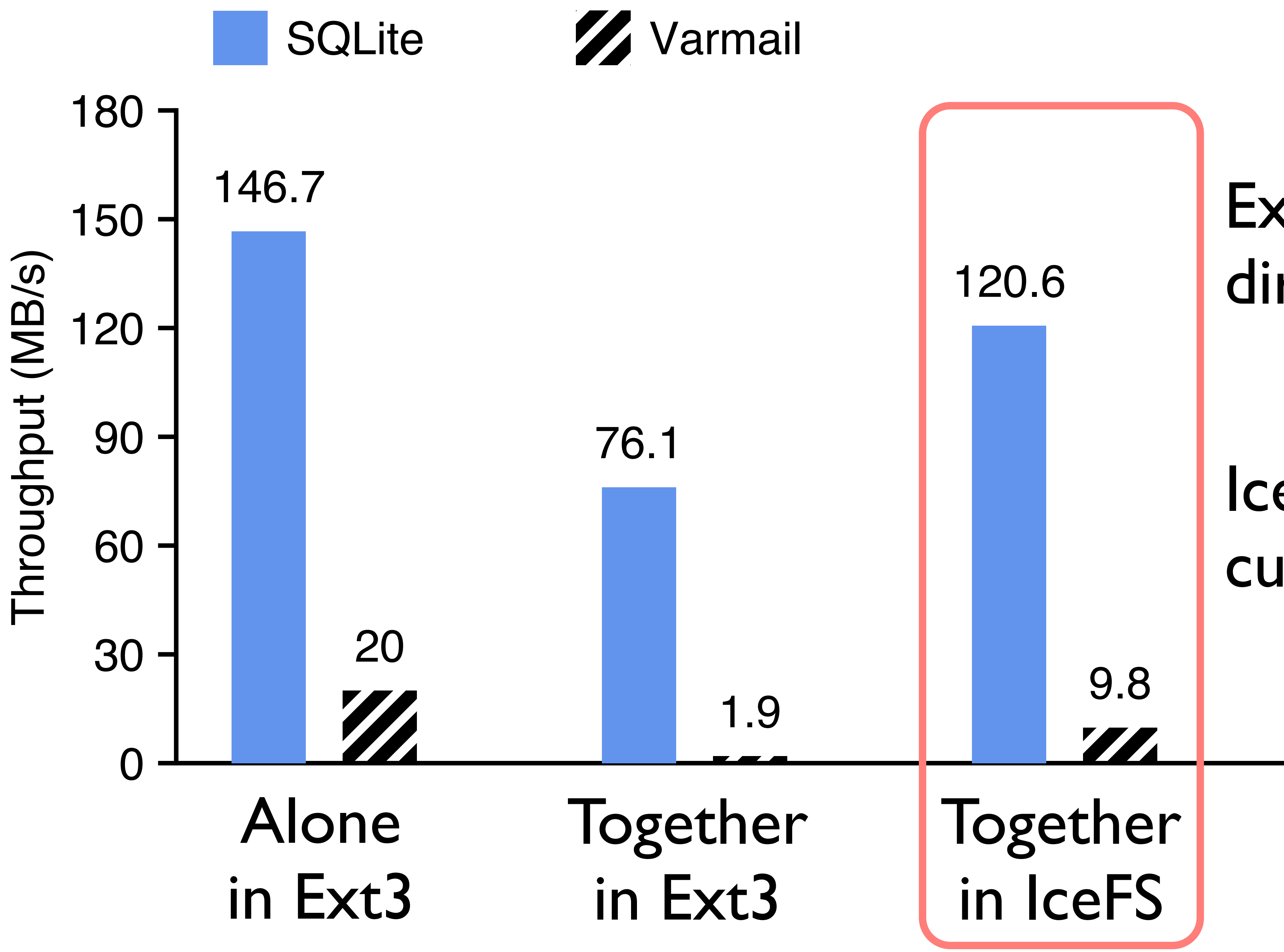
Ext3 Journaling



Ext3 runs with 2 directories

Shared transactions hurt performance (over 10x)

Isolated Journaling In IceFS

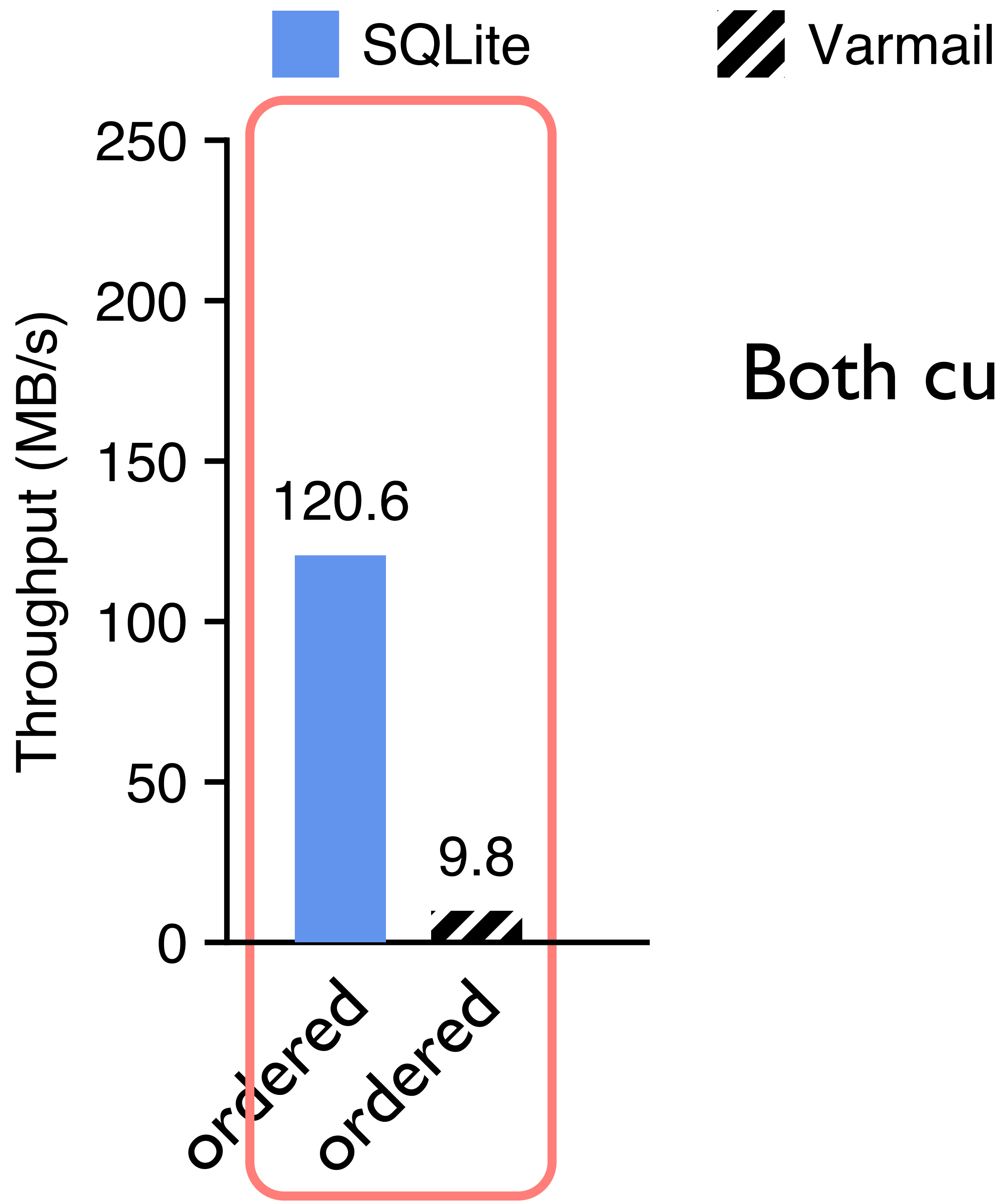


Ext3 runs with 2 directories

IceFS runs with 2 cubes

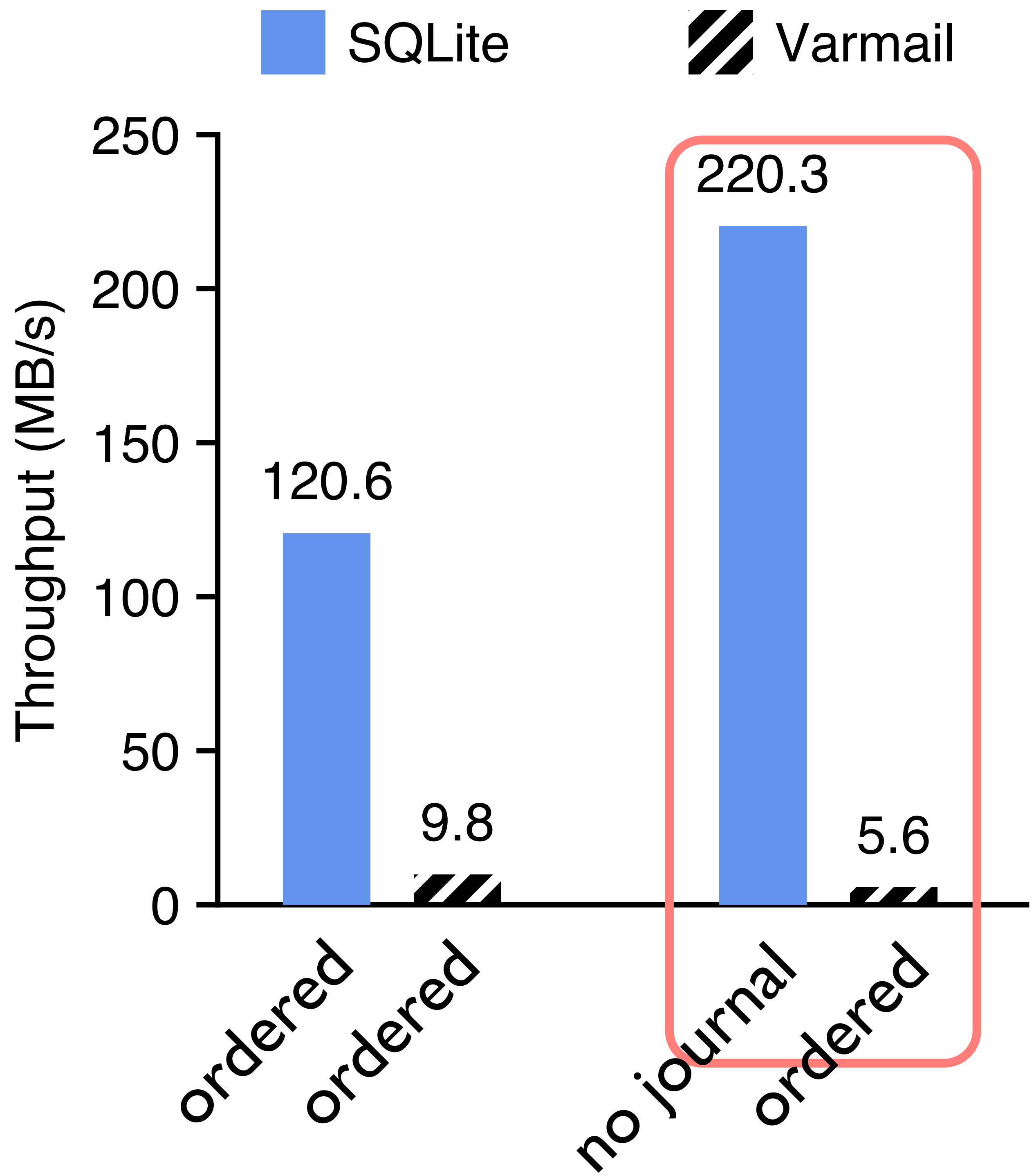
Parallel transactions in IceFS provide isolated performance (over 5x)

Specialized Journaling In IceFS



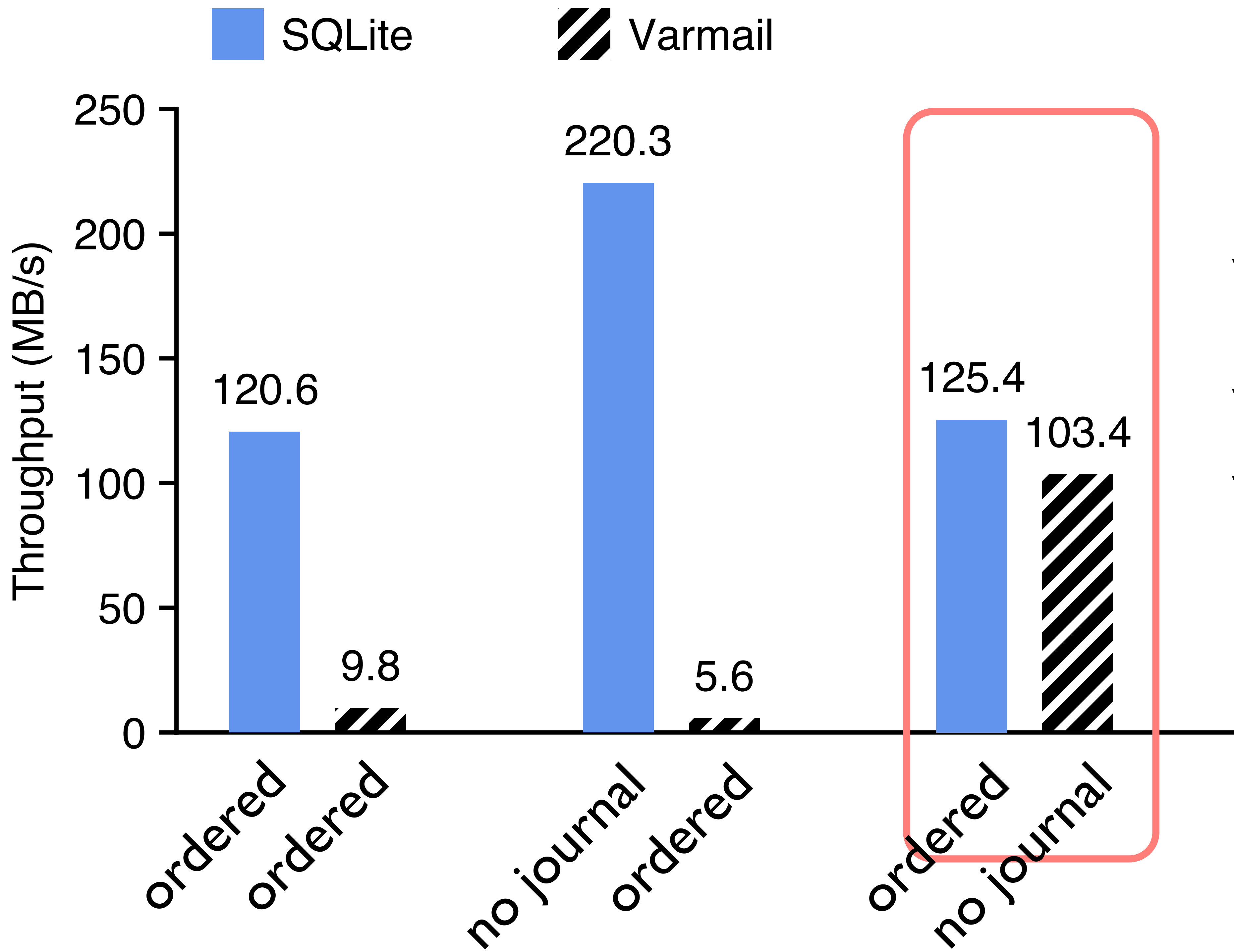
Both cubes use ordered mode

Specialized Journaling In IceFS



SQLite runs with no journal
Varmail runs with ordered

Specialized Journaling In IceFS



SQLite runs with ordered

Varmail runs with no journal

Specialized journaling in IceFS provide flexibility between consistency and performance (over 50x)

Evaluation

Isolate failures ?

- inject around 200 faults
- per-cube failure (read-only or crash) for IceFS

Faster recovery ?

- independent recovery for a cube

Better journaling performance ?

- isolated journaling performance
- flexibility between consistency and performance

Evaluation

Isolate failures ?

- inject around 200 faults
- per-cube failure (read-only or crash) for IceFS

Faster recovery ?

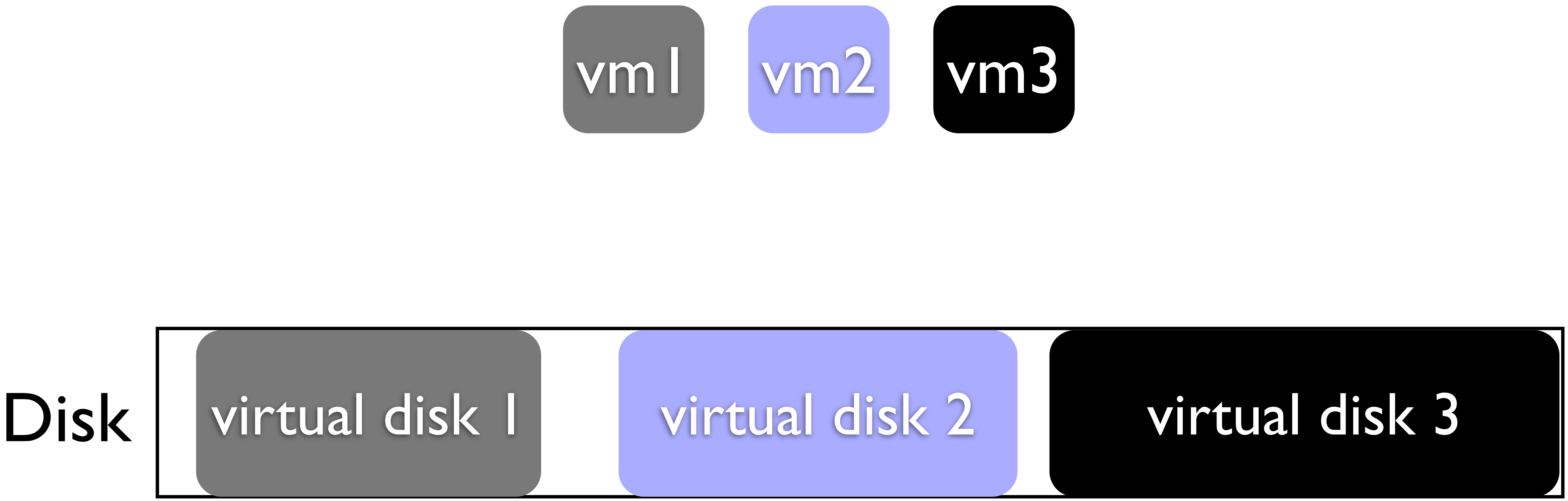
- independent recovery for a cube

Better journaling performance ?

- isolated journaling performance
- flexibility between consistency and performance

Useful for applications ?

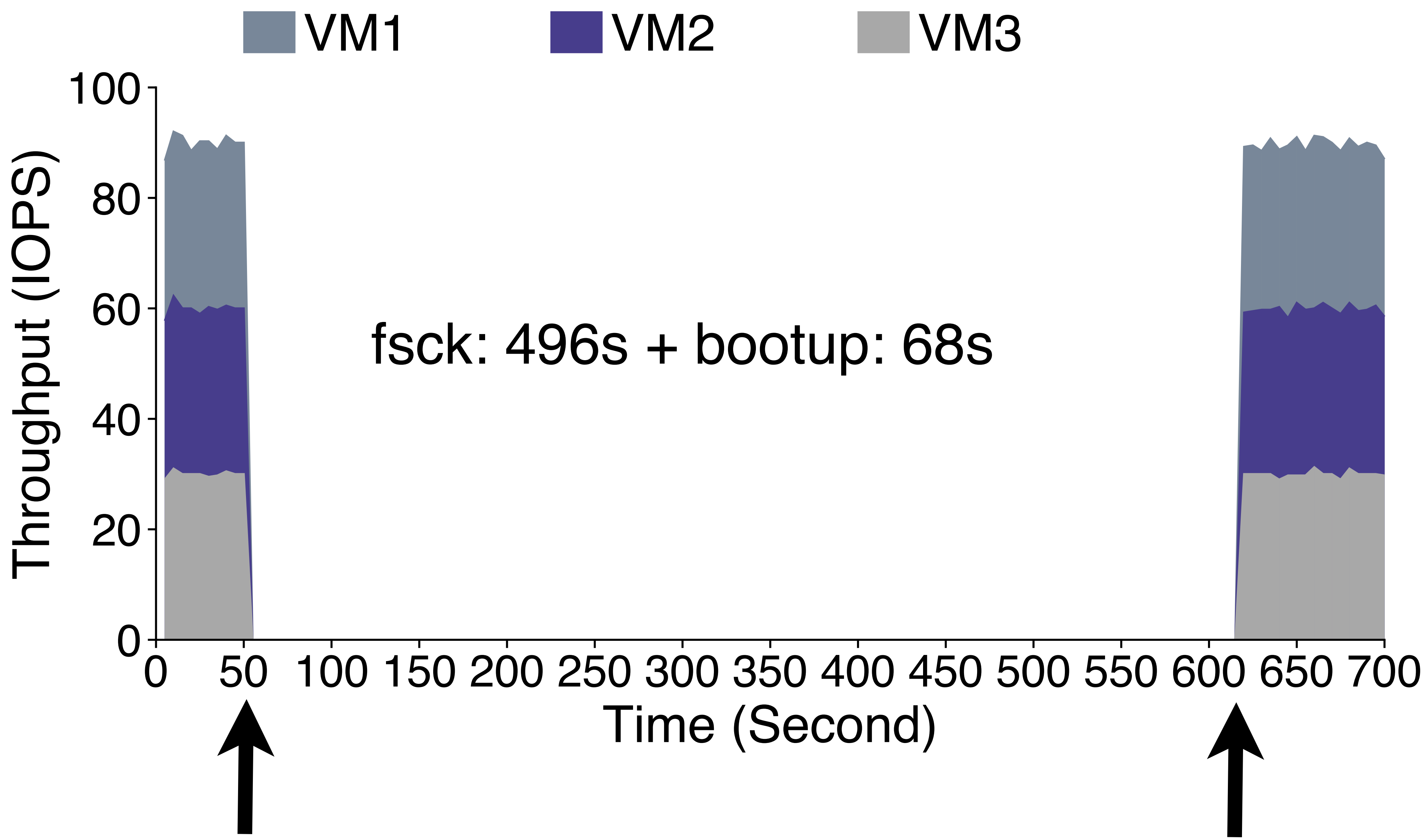
Server Virtualization



Shared file system

Failures and recovery of the shared file system impact all virtual machines

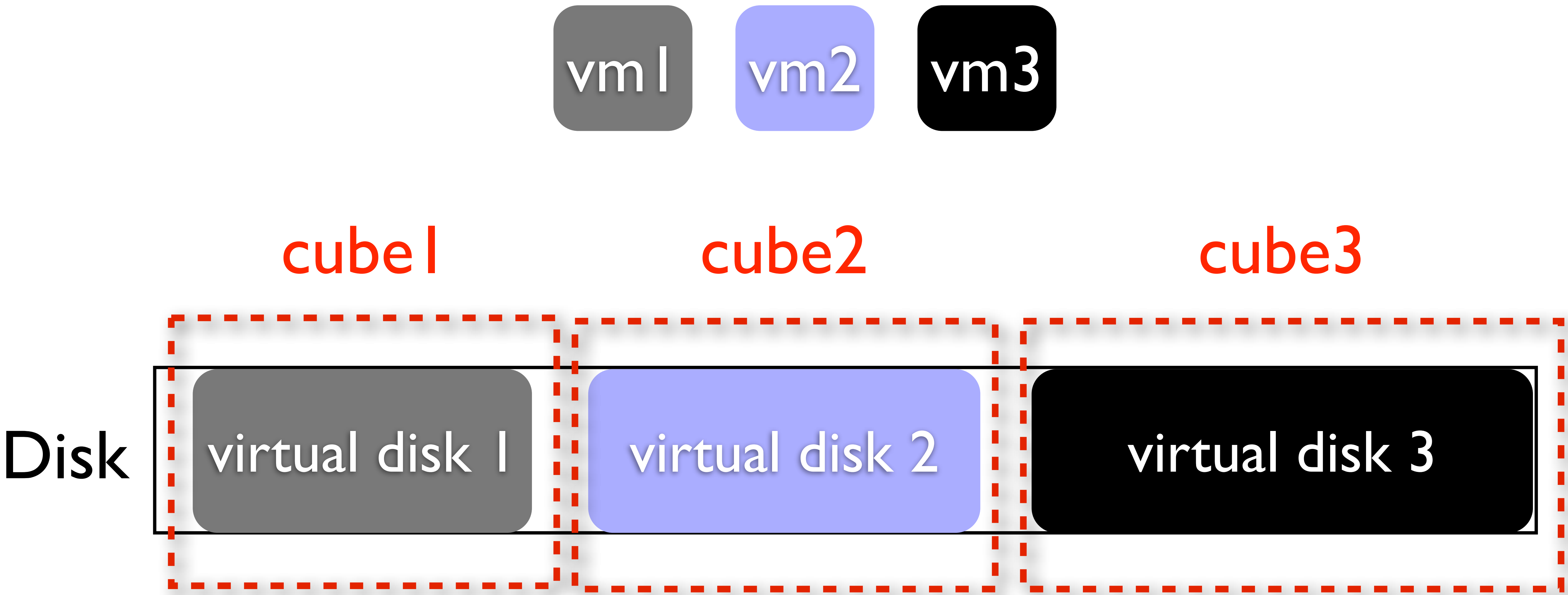
Virtual Machines



Inject metadata corruption

bootup three vms

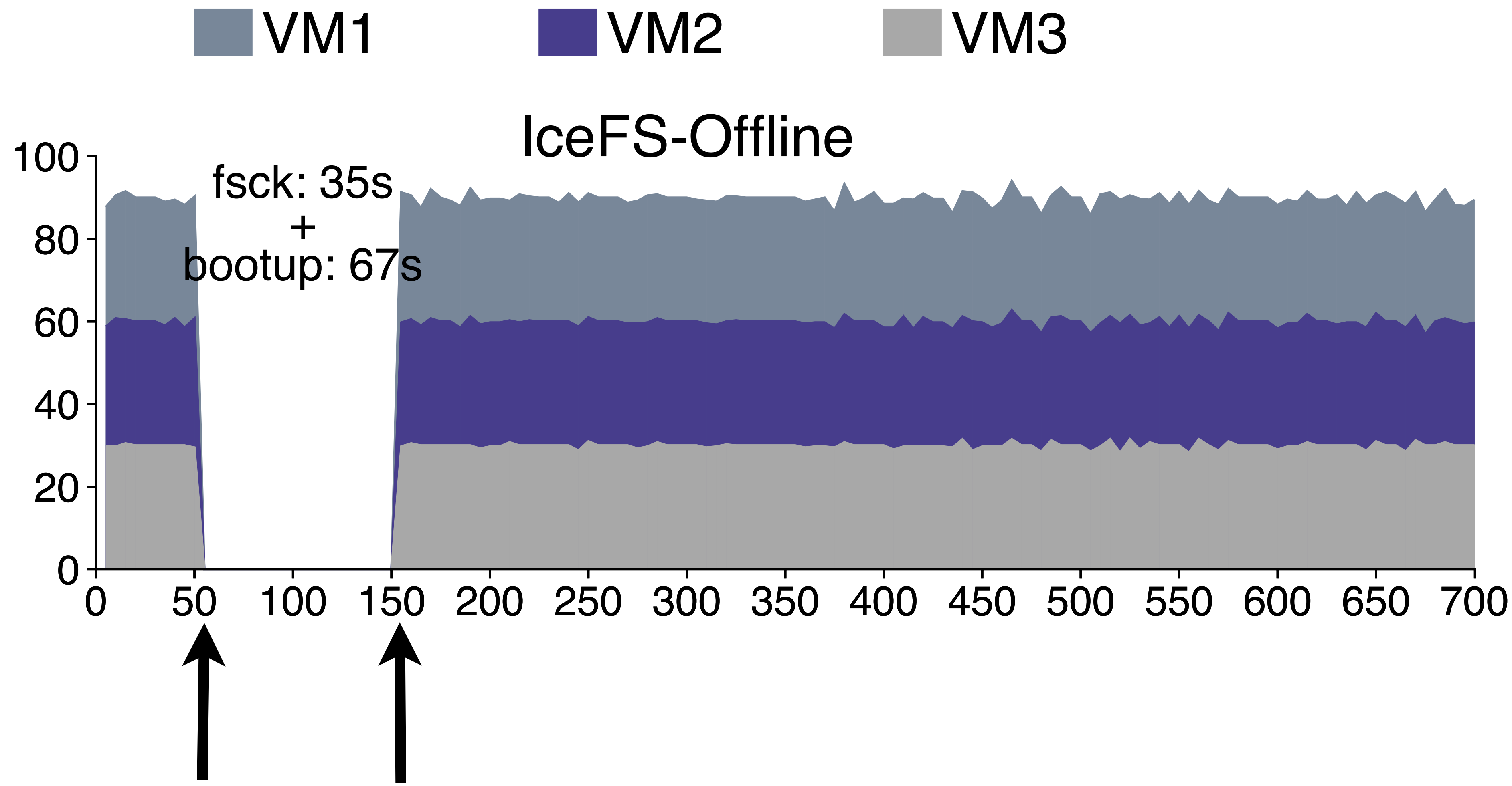
Server Virtualization with IceFS



Shared file system with cubes

Server Virtualization with IceFS

recover
a cube
offline



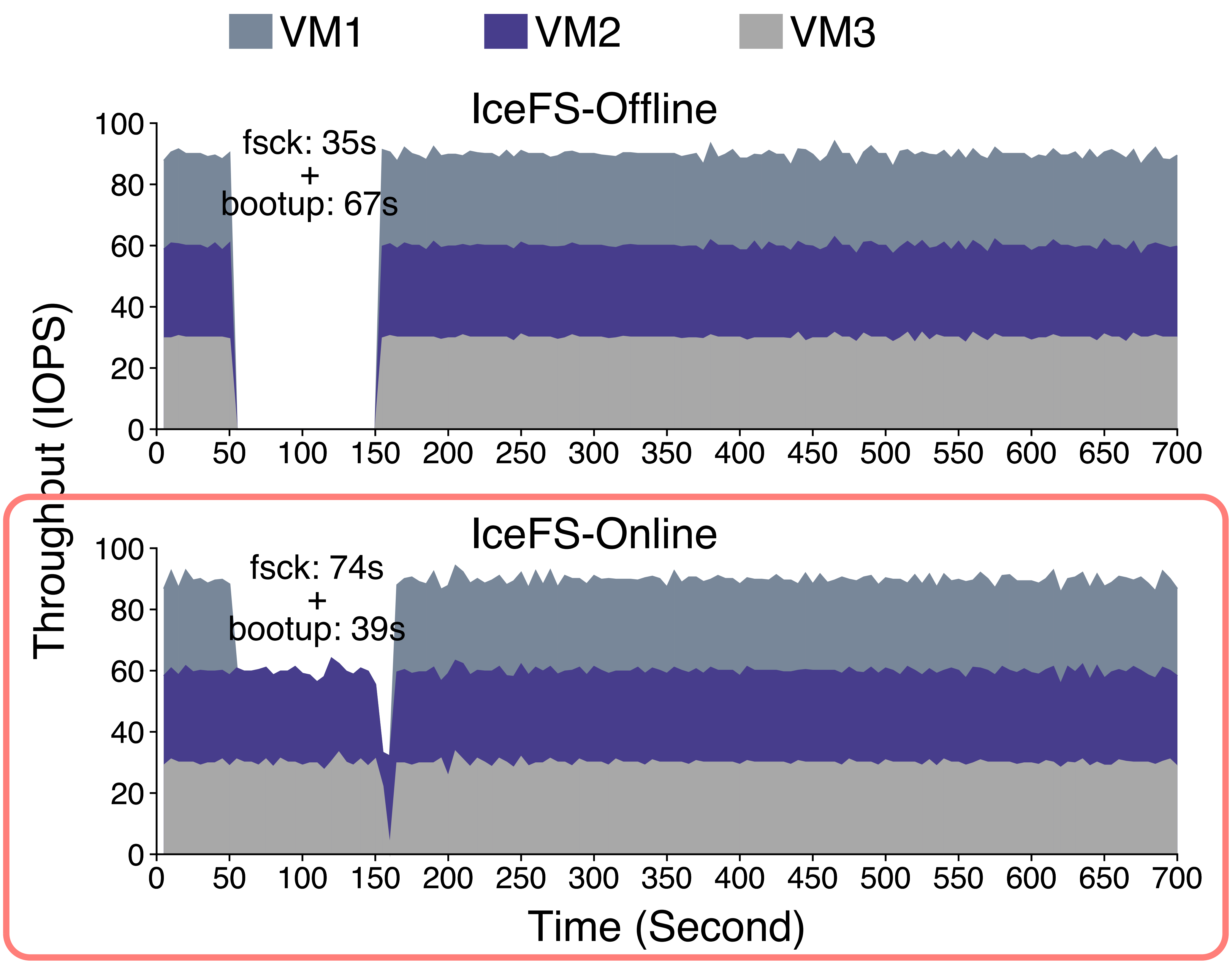
Inject metadata corruption

bootup three vms

Server Virtualization with IceFS

recover a cube offline

recover a cube online



Evaluation

Isolate failures ?

- inject around 200 faults
- per-cube failure (read-only or crash) for IceFS

Faster recovery ?

- independent recovery for a cube

Better journaling performance ?

- isolated journaling performance for cubes
- flexibility between consistency and performance

Useful for applications ?

- significantly reduce system downtime

Summary of IceFS

Local file systems lack physical isolation

- physical entanglement
- reliability and performance problems

IceFS provides isolation with data containers

Computing is becoming virtualized, shared, and multi-tenant

- isolation is the key

Systems need to rethink isolation

- avoid entanglement
- provide useful abstractions for applications

Outline

- Introduction
- Disentanglement in IceFS
  - File system Disentanglement
  - The Ice File System
  - Evaluation
- Key-Value Separation in WiscKey
  - Key-value Separation Idea
  - Challenges and Optimization
  - Evaluation
- Conclusion

Key-Value Stores

Key-value stores are important

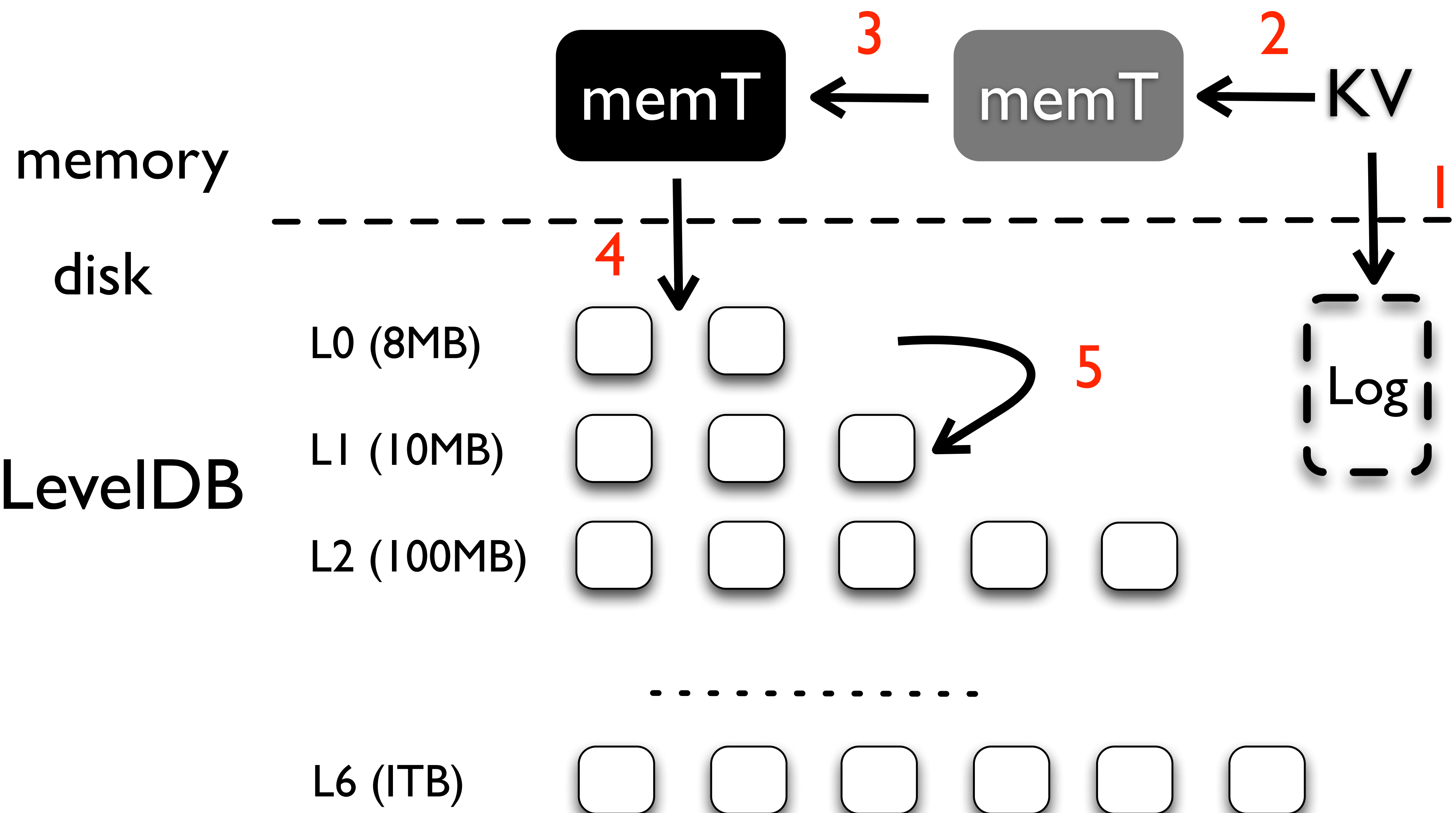
- web indexing, e-commerce, social networks
- local and distributed key-value stores
  - hash table, b-trees
  - log-structured merge trees (LSM-trees)

LSM-tree based key-value stores are popular

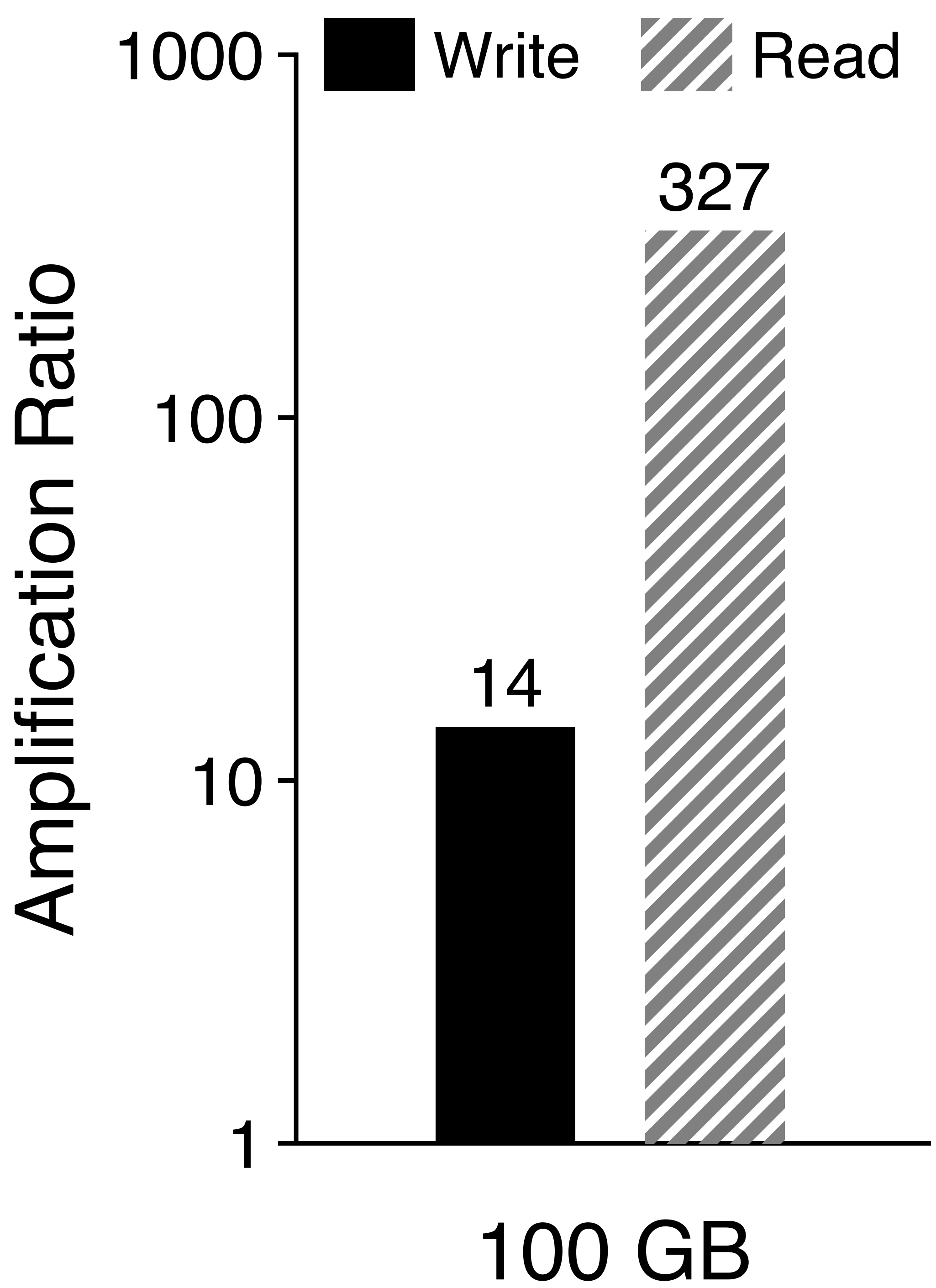
- optimize for write intensive workloads
- advanced features: range query, snapshot
- widely deployed
  - BigTable and LevelDB at Google
  - HBase, Cassandra and RocksDB at FaceBook

LSM-trees Background

Batch and write sequentially
Sort data for quick lookups



I/O Amplification in LSM-trees



Random load:
a 100GB database

Random lookup:
100,000 lookups

Problems:

large write amplification

large read amplification

Why LSM-trees ?

Good for hard drives

- high write throughput
- sequential vs random: can be up to 1000

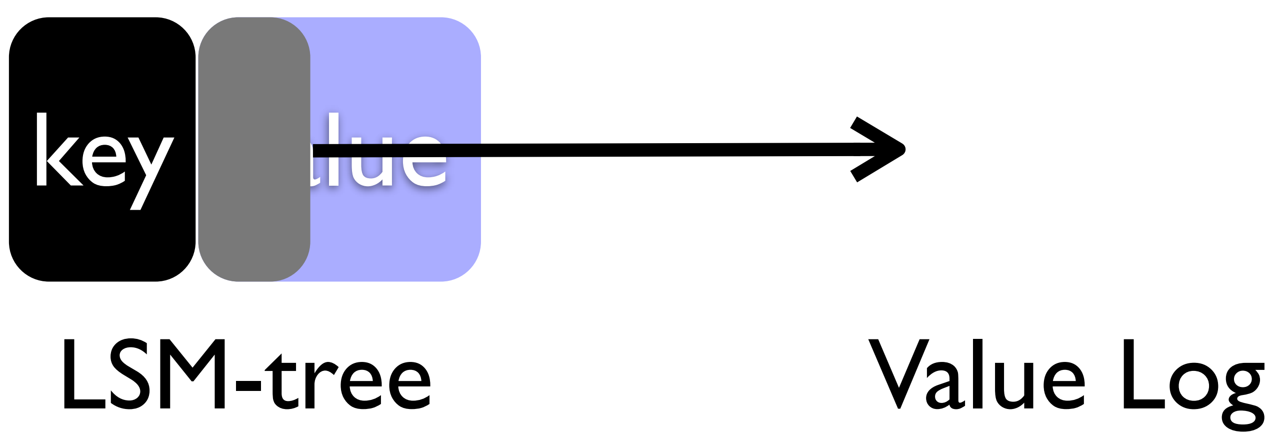
Not optimal for SSDs

- large write/read amplification
  - waste device resource
  - decrease device's lifetime
- unique characteristics of SSDs
  - fast random reads
  - internal parallelism

Our Solution: WiscKey

An SSD-conscious LSM-tree store

- main idea: separate keys and values
- harness SSD's internal parallelism for range queries
- online and light-weight garbage collection
- minimize I/O amplification and crash consistent

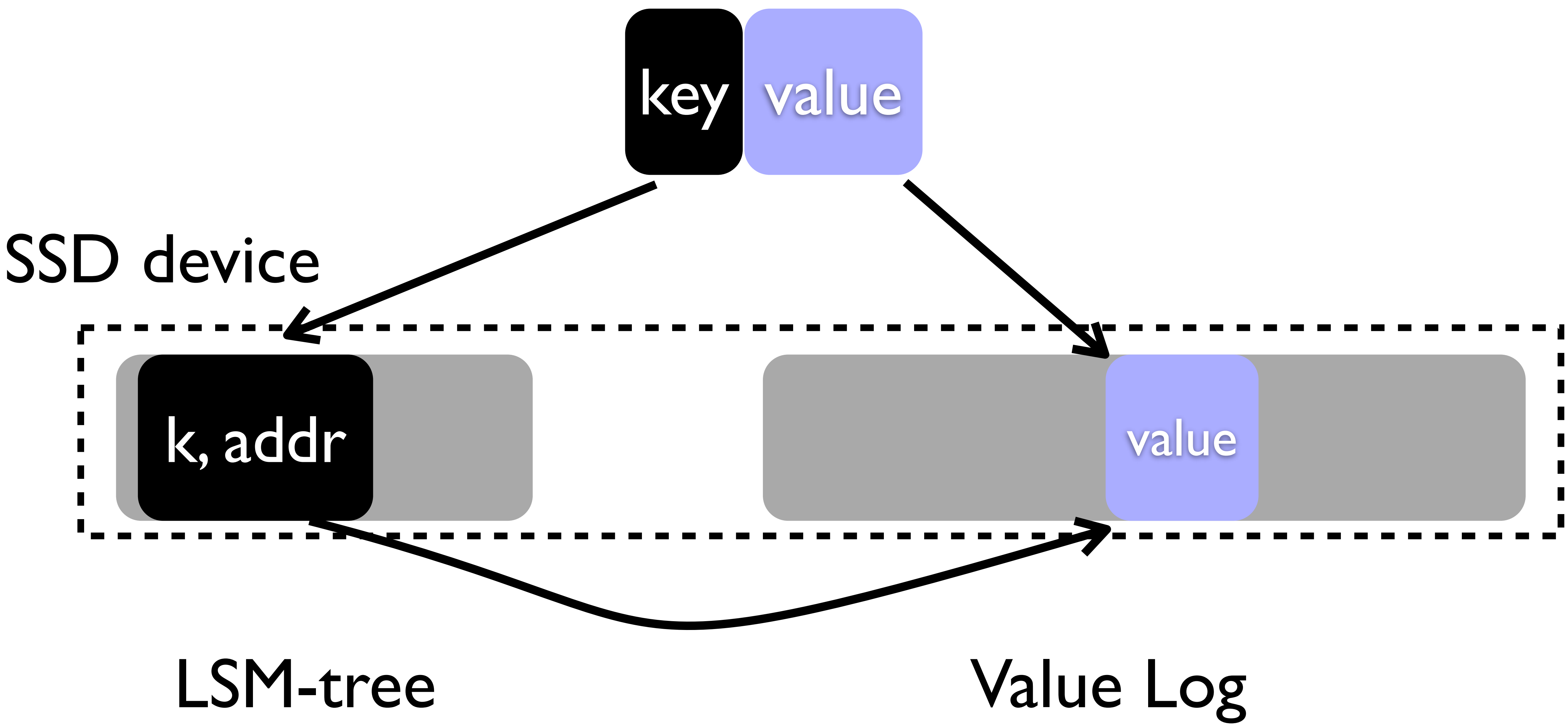


Performance of WiscKey

- 2.5x to 111x for loading, 1.6x to 14x for lookups
- both micro and macro benchmarks

Key-Value Separation

Main idea: only keys are required to be sorted, values can be managed separately



Outline

Introduction

Disentanglement in IceFS

- File system Disentanglement
- The Ice File System
- Evaluation

Key-Value Separation in WiscKey

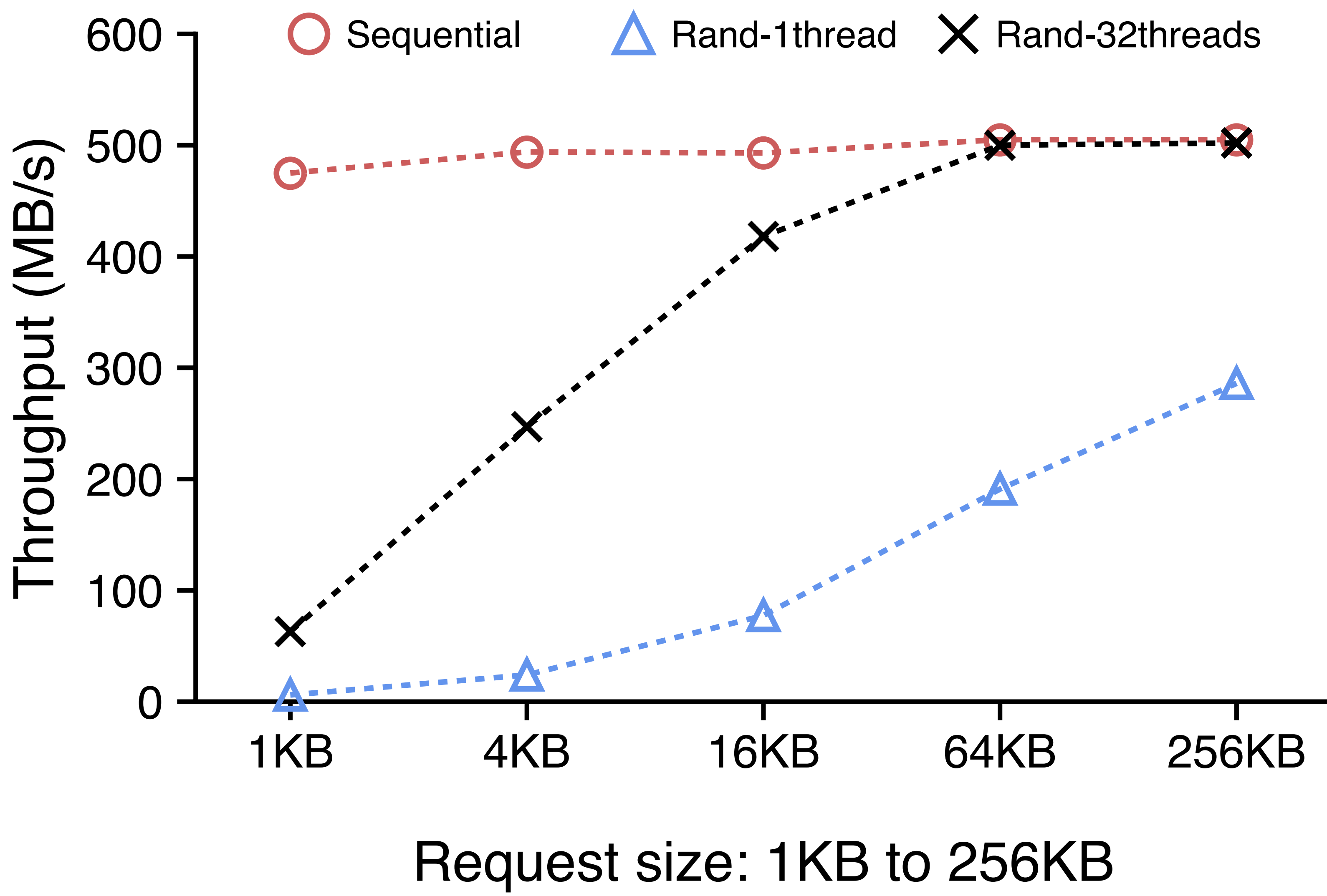
- Key-value Separation Idea
- Challenges and Optimization
- Evaluation

Conclusion

Parallel Range Query

SSD read performance

- sequential, random, parallel



SSD: Samsung 840 EVO 500GB

Reads on a 100GB file on ext4

Parallel Range Query

Challenge

- sequential reads in LevelDB
- read keys and values separately in WiscKey

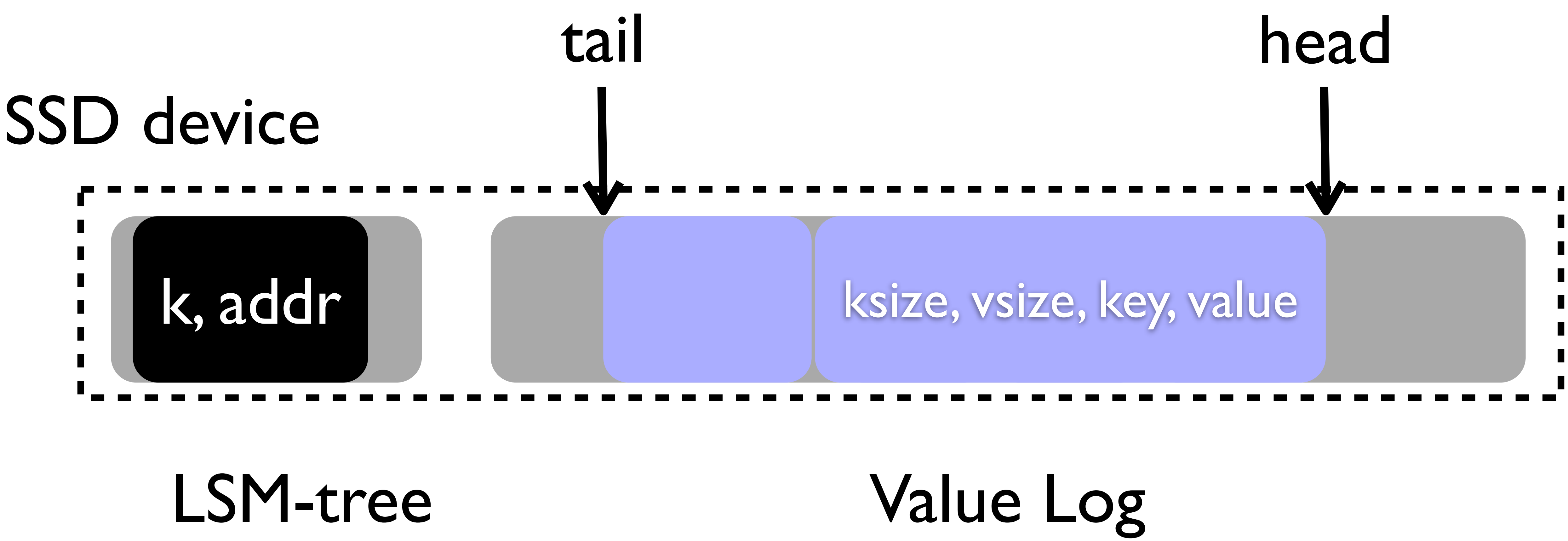
Parallel range query

- leverage parallel random reads of SSDs
- prefetch key-value pairs in advance
  - range query interface: seek(), next(), prev()
  - detect a sequential pattern
  - prefetch concurrently in background

Garbage Collection

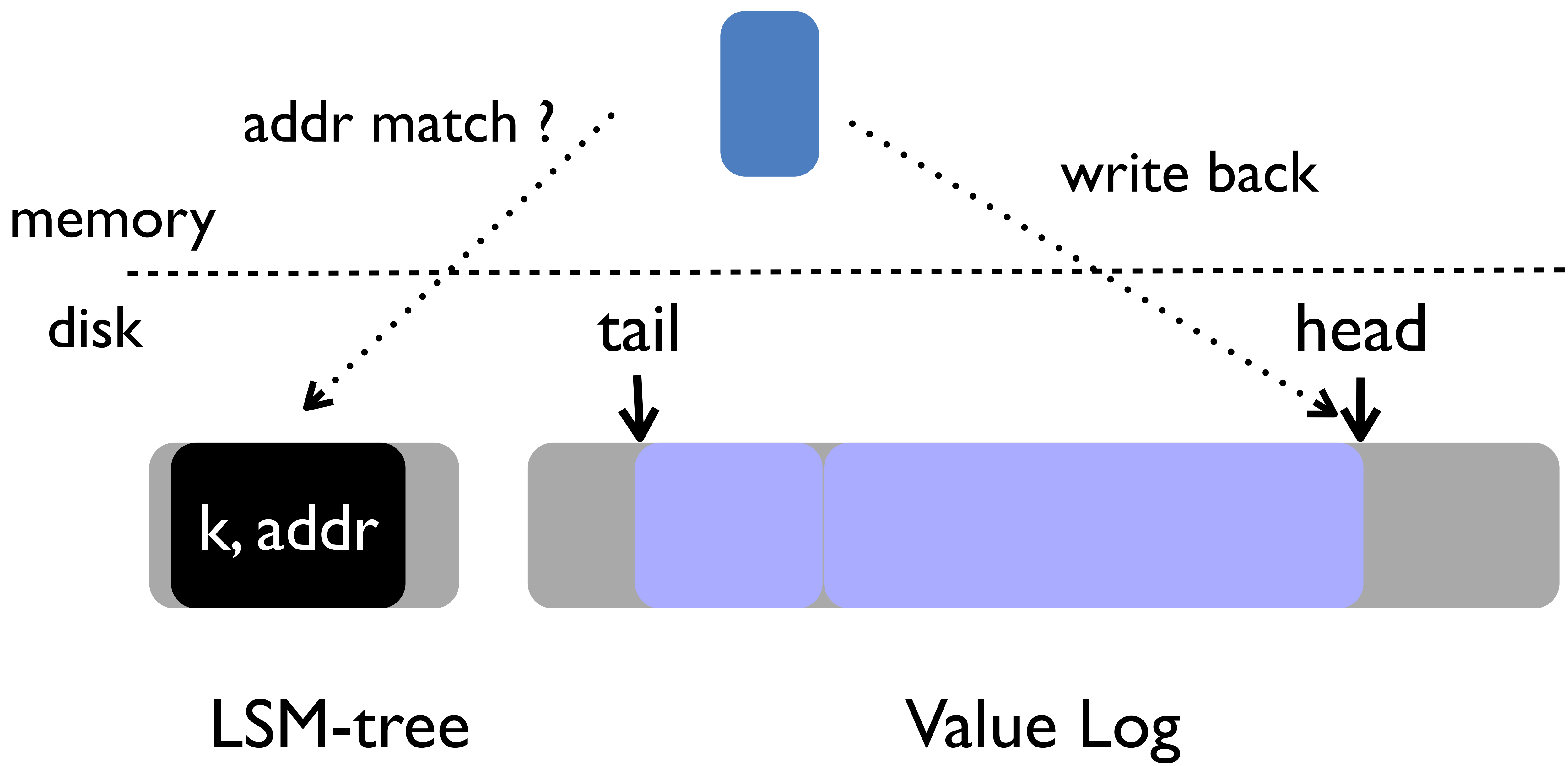
Online and light-weight

- append (ksize, vsize, key, value) in value log
- tail and head pointers for the valid range
- tail and head are stored in LSM-tree



Garbage Collection

1. read from the tail
2. check the LSM-tree
3. write back valid kv pairs
4. free space and update pointers



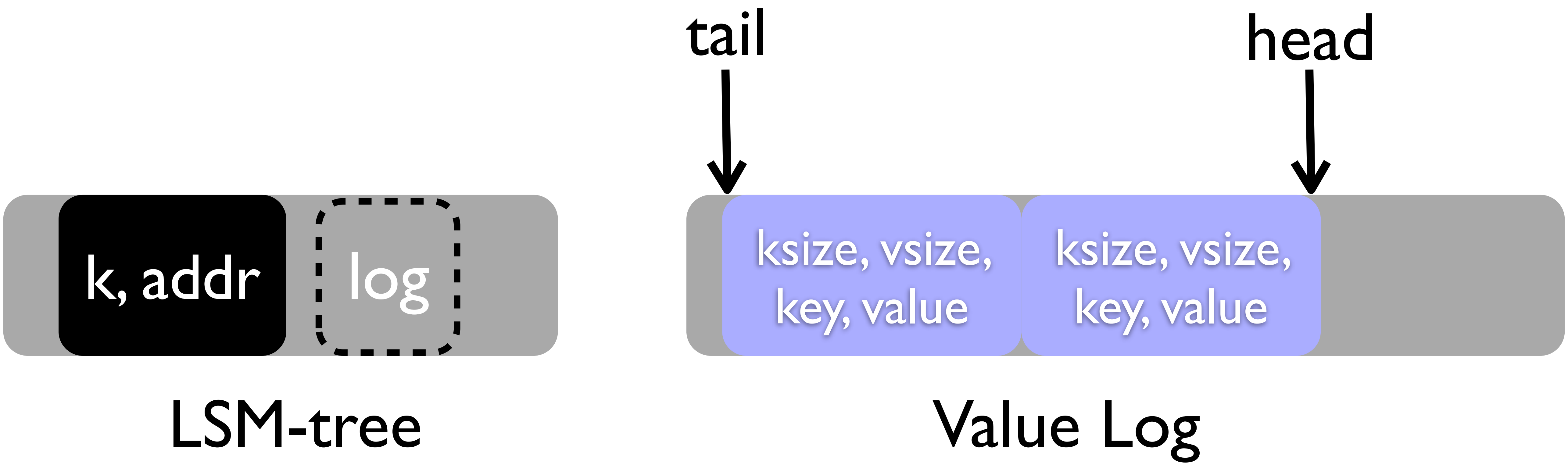
Optimizing LSM-tree Log

LSM-tree log

- used for recovery in case of a crash
- performance overhead for small kv pairs

Remove LSM-tree log in WiscKey

- store head in LSM-tree periodically
- scan the value log from the head to recover



WiscKey Implementation

Based on LevelDB

- a separate vLog file for values
- modify I/O paths to separate keys and values
- straightforward to implement

Range query

- a background thread pool
- detect sequential pattern with the Iterator interface

File-system support

- fadvise to predeclare access patterns
- hole-punching to free space

Outline

Introduction

Disentanglement in IceFS

- File system Disentanglement
- The Ice File System
- Evaluation

Key-Value Separation in WiscKey

- Key-value Separation Idea
- Challenges and Optimization
- Evaluation

Conclusion

Experiment Setup

Testing machine

- 16 cores (3.3 GHz), 64 GB memory
- Samsung 840 EVO SSD (500 GB)
  - maximal sequential read: 500 MB/s
  - maximal sequential write: 400 MB/s

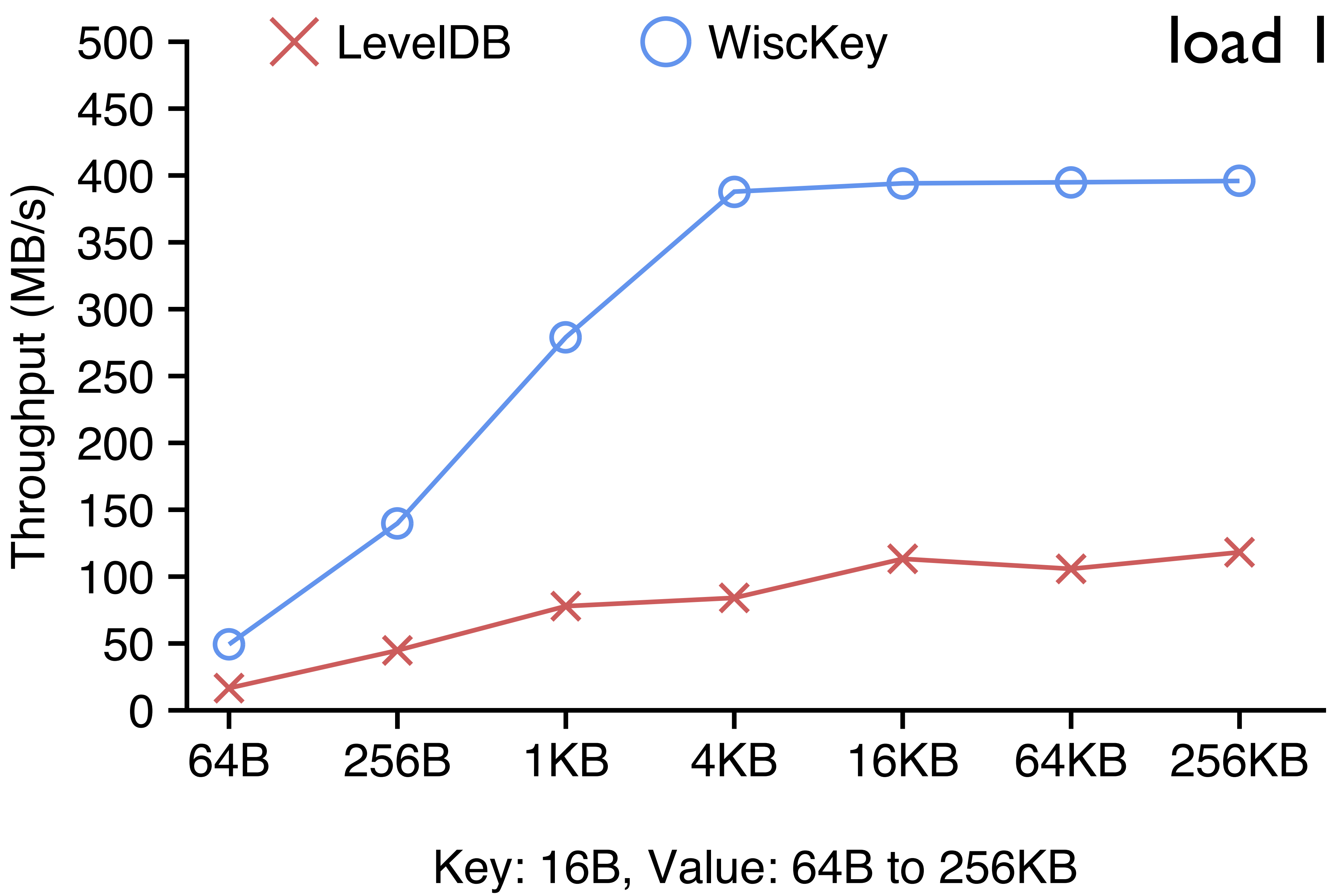
Workloads

- micro benchmarks (db_bench)
- YCSB benchmark

Evaluation

How does key-value separation impact the performance of WiscKey ?

Sequential Load

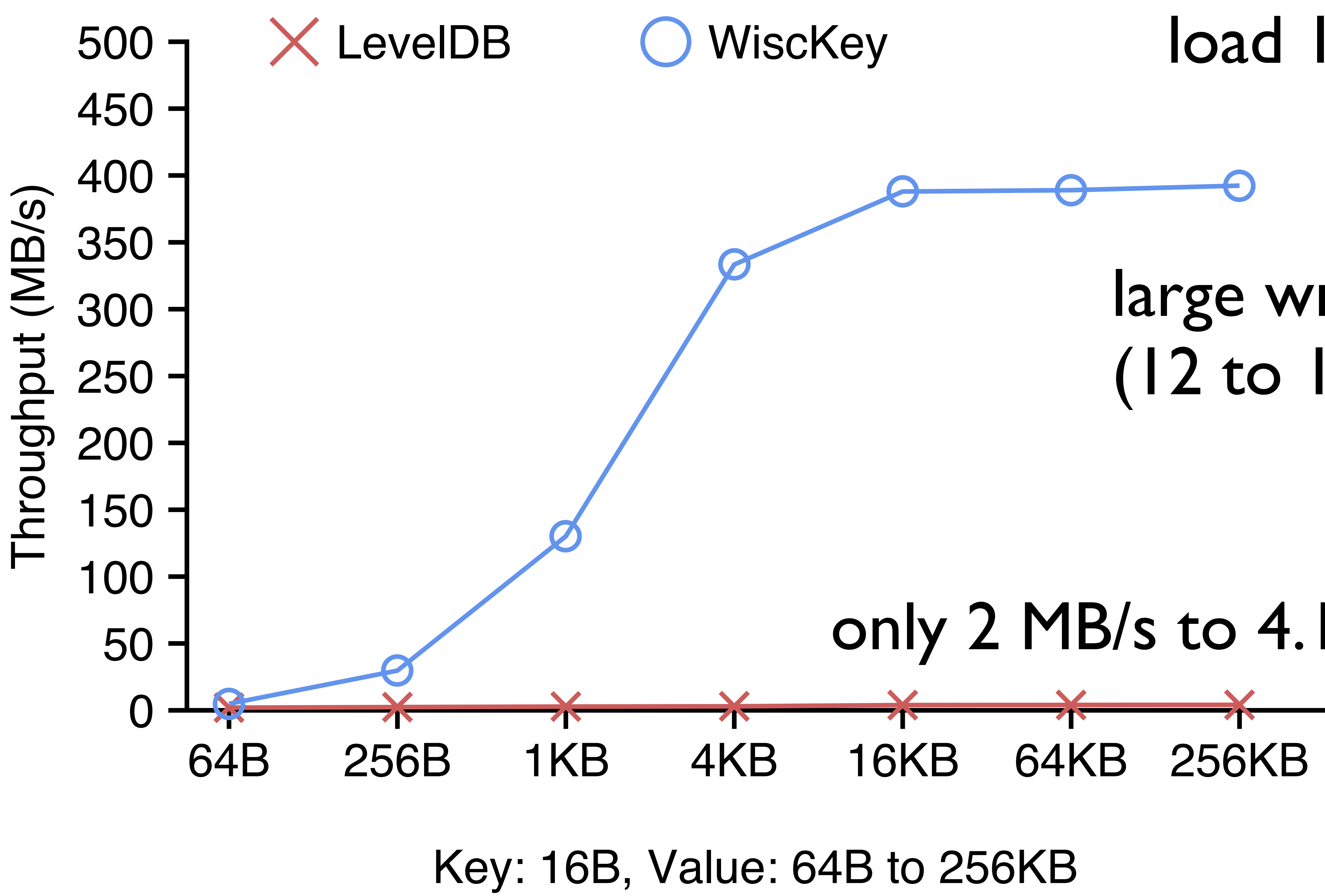


load 100 GB database

log writing in LevelDB has high overhead

WiscKey is over 3x faster due to its write buffer and removing the LSM-tree log

Random Load



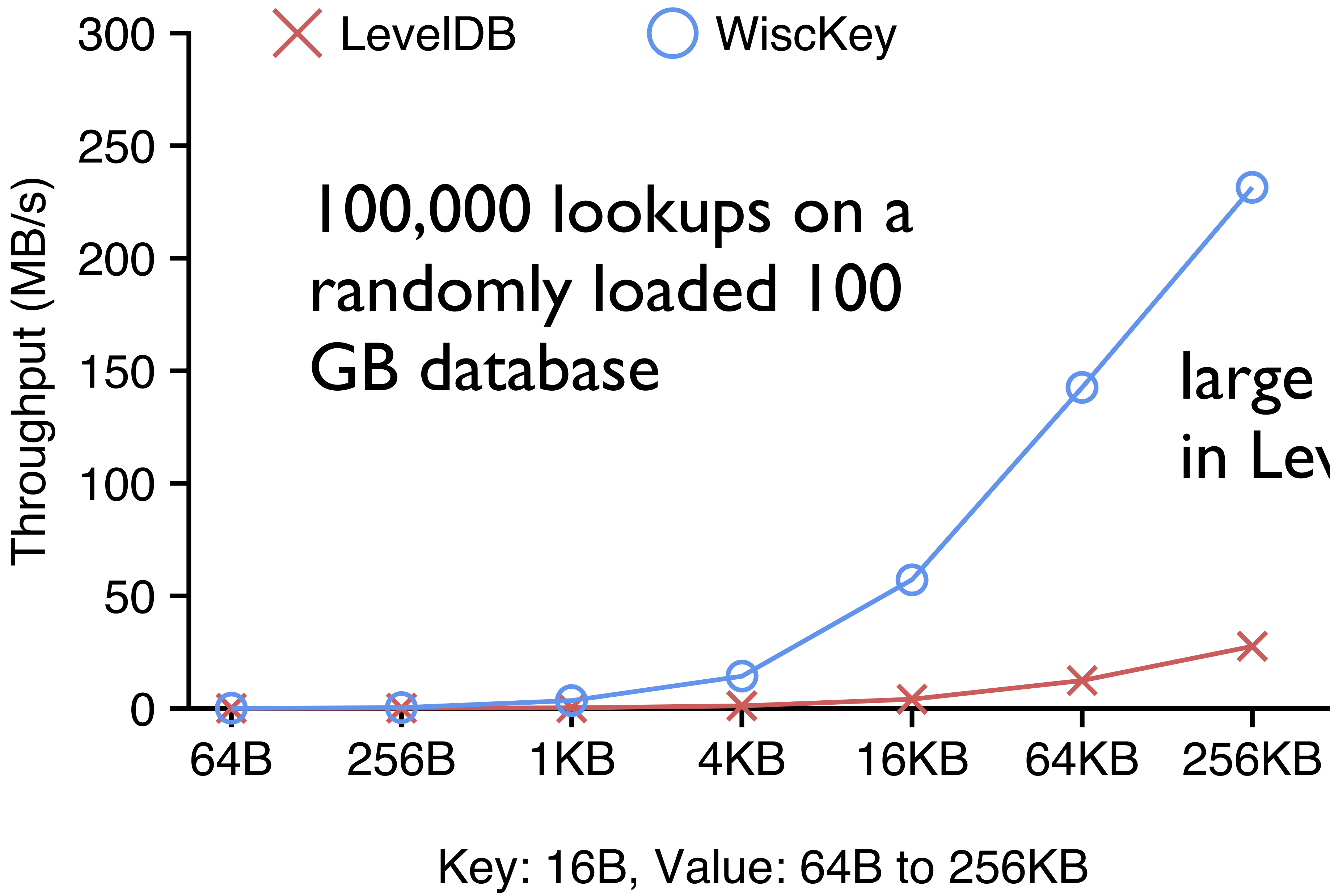
load 100 GB database

large write amplification (12 to 16) in LevelDB

only 2 MB/s to 4.1 MB/s

Small write amplification in WiscKey due to key-value separation (up to 111x in throughput)

Random Lookup



100,000 lookups on a randomly loaded 100 GB database

large read amplification in LevelDB

Smaller LSM-tree in WiscKey leads to better lookup performance (1.6x - 14x)

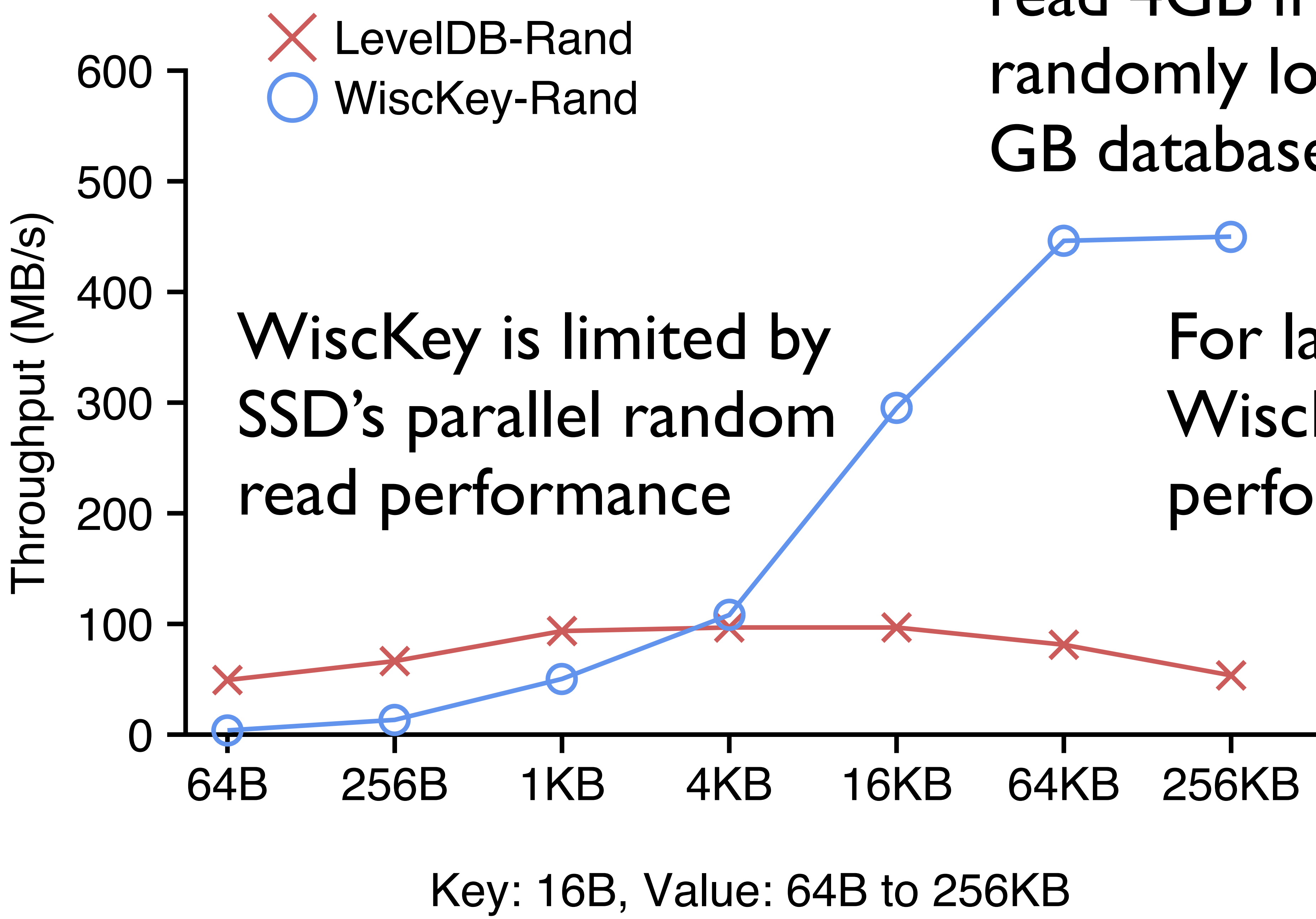
Evaluation

How does key-value separation impact the performance of WiscKey ?

- low write and read amplification
- load (2.5x to 111x), lookup (1.6x to 14x)

Is the parallel range query fast enough ?

Range Query



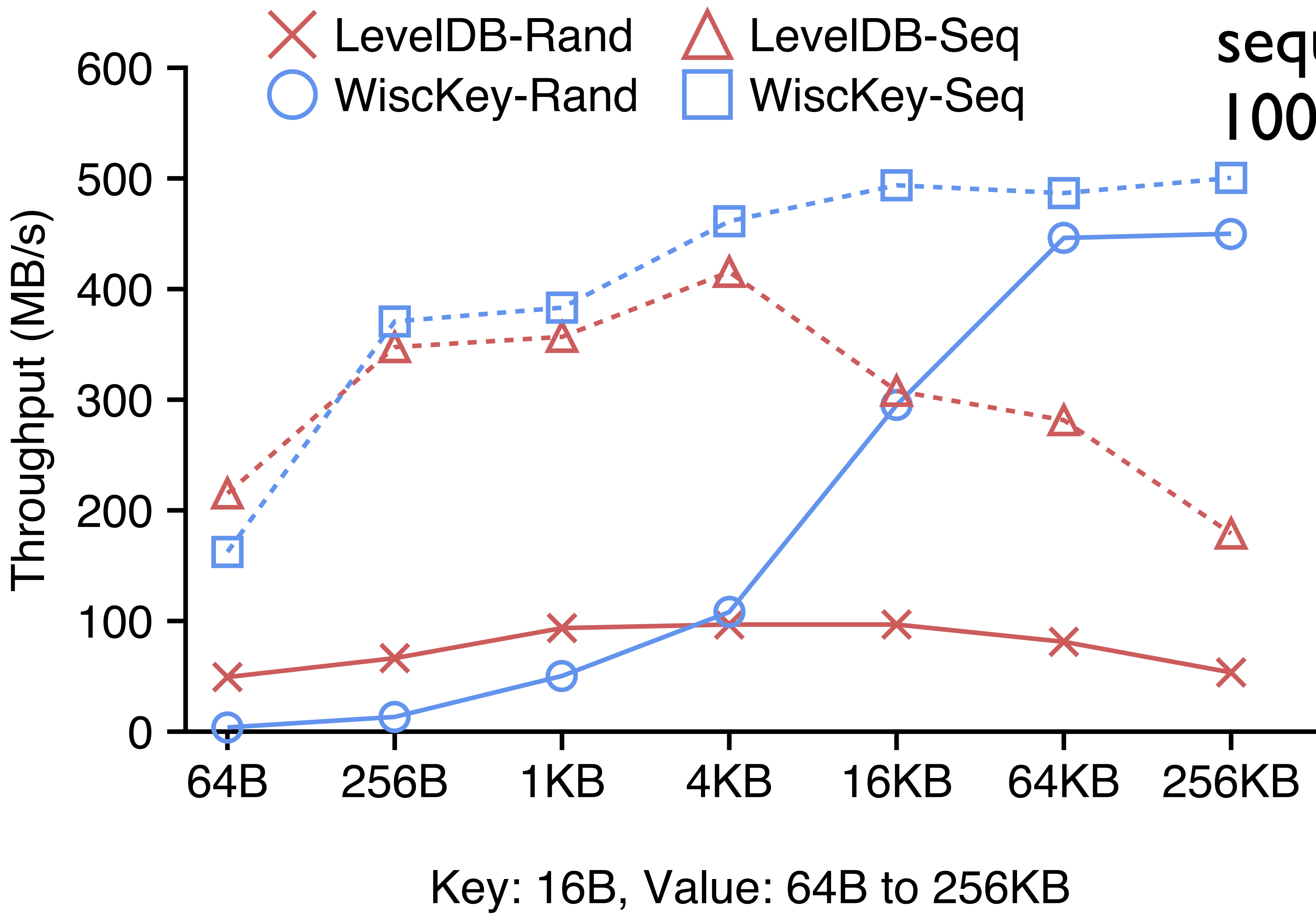
read 4GB from a randomly loaded 100 GB database

WiscKey is limited by SSD's parallel random read performance

For large kv pairs, WiscKey can perform better

Better for large kv pairs, but worse for small kv pairs on an unsorted database

Range Query



read 4GB from a sequentially loaded 100 GB database

Both WiscKey and LevelDB read sequentially

Sorted databases help WiscKey's range query

Evaluation

How does key-value separation impact the performance of WiscKey ?

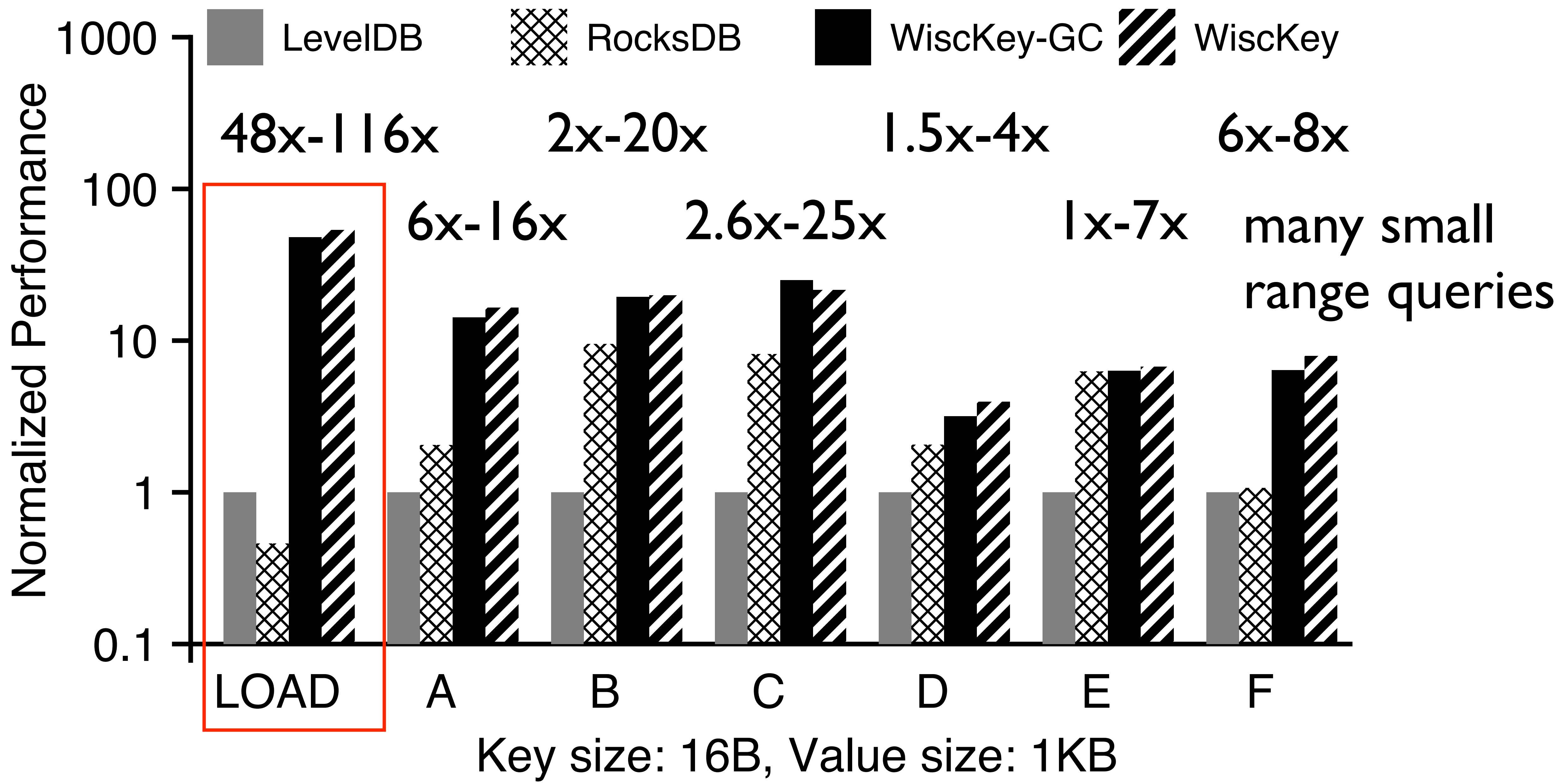
- low write and read amplification
- load (2.5x to 111x), lookup (1.6x to 14x)

Is the parallel range query fast enough ?

- limited by random read performance
- sorting helps

How about real workloads ? What is the effect of garbage collection ?

YCSB Benchmarks



A: 50% R, 50% U; B: 95% R, 5% U; C: 100% R;
D: 95% R, 5% I; E: 95% Scan, 5% I; F: 50% R, 50% RMW

Evaluation

How does key-value separation impact the performance of WiscKey ?

- low write and read amplification
- load (2.5x to 111x), lookup (1.6x to 14x)

Is the parallel range query fast enough ?

- limited by random read performance
- sorting helps

How about real workloads ? What is the effect of garbage collection ?

- faster on all workloads
- performance similar to micro benchmarks

Summary of WiscKey

LSM-trees are not optimized for SSD devices

WiscKey separates keys from values with an SSD-conscious design

Many novel storage systems have been built for hard drives

Transition to new storage hardware

- leverage existing software
- explore new ways to utilize the new hardware
- get the best of two worlds

Outline

Introduction

Disentanglement in IceFS

- File system Disentanglement
- The Ice File System
- Evaluation

Key-Value Separation in WiscKey

- Key-value Separation Idea
- Challenges and Optimization
- Evaluation

Conclusion

Lessons Learned

A large-scale study is feasible and valuable

Research should match reality

History repeats itself

Don’t settle for existing abstraction

Isolation should be a fundamental design goal

Don’t run old software on new hardware

Fundamental details matter

Work on systems extremely slow or unreliable

Conclusion

Local storage systems are important

Physical separation is useful

- improve both reliability and performance over 10x
- better reliability: isolated failures, localized recovery
- better performance: specialized journaling, minimize I/O amplification

Computing and storage are evolving

- virtualized, shared and fast
- physical separation is the key
- IceFS and WiscKey are just a beginning