Strata: A Cross Media File System

Youngjin Kwon

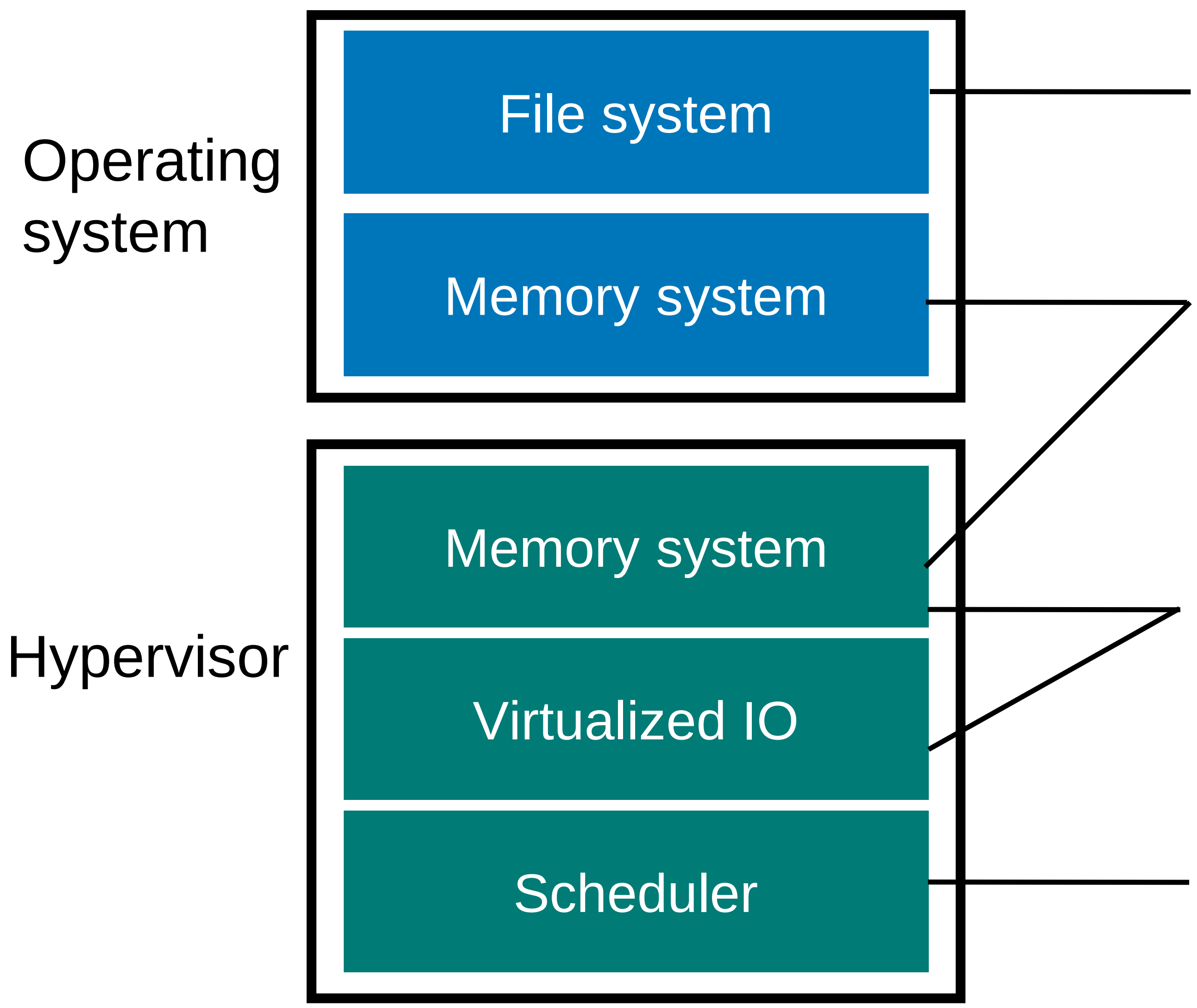
KAIST

with Henrique Fingler, Tyler Hunt,
Simon Peter, Emmett Witchel, Thomas Anderson

The University of Texas at Austin

UNIVERSITY of WASHINGTON

My research



Strata: A Cross Media File System
[SOSP 2017]

Ingens: Efficient Huge Page Management
[OSDI 2016]

Sego: Hypervisor for Untrusted OS
[ASPLOS 2016]

Virtualizing Asymmetric Multi-cores
[ISCA 2011]

Let’s build a fast server

NoSQL store, Database, File server, Mail server …

Requirements

- Small updates (1 Kbytes) dominate
- Dataset scales up to 10 TB
- Updates must be crash consistent

Storage diversification

Byte-addressable: cache-line granularity IO

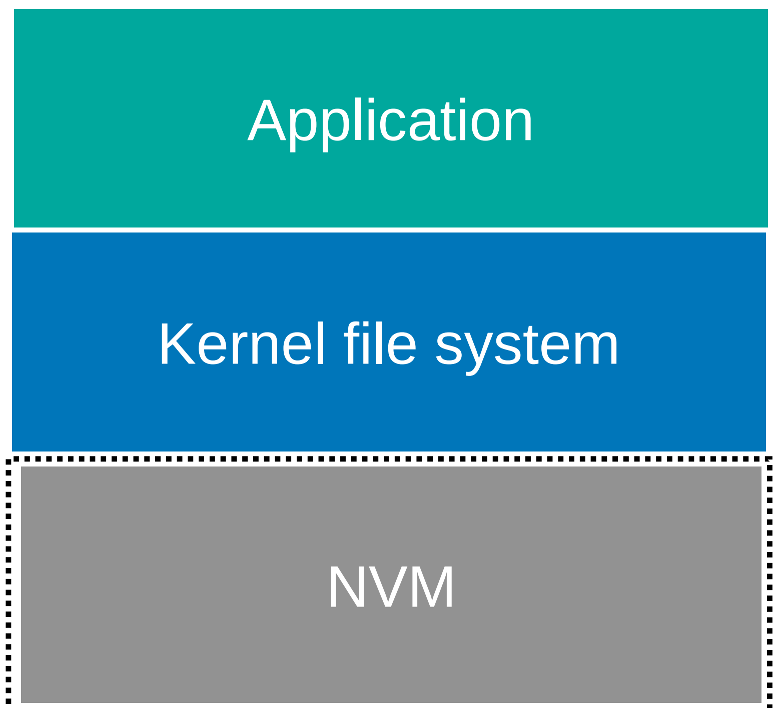
	Latency	$/GB
DRAM	80 ns	35.1
NVM	170 ns	4.5
SSD	10 us	0.48
HDD	10 ms	0.02

Better performance ↑
Higher capacity ↓

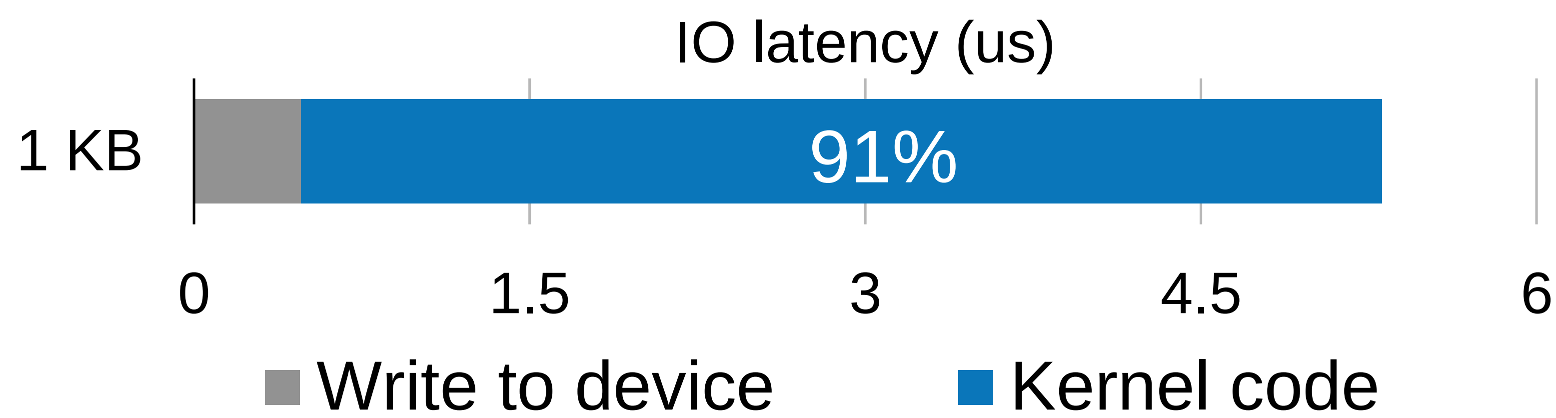
Large erasure blocks need to be sequentially written
Random writes: **5~6x slowdown** due to **GC** **[FAST'15]**

A fast server on today's file system

- **Small updates (1 Kbytes) dominate**
- Dataset scales up to 10TB
- Updates must be crash consistent



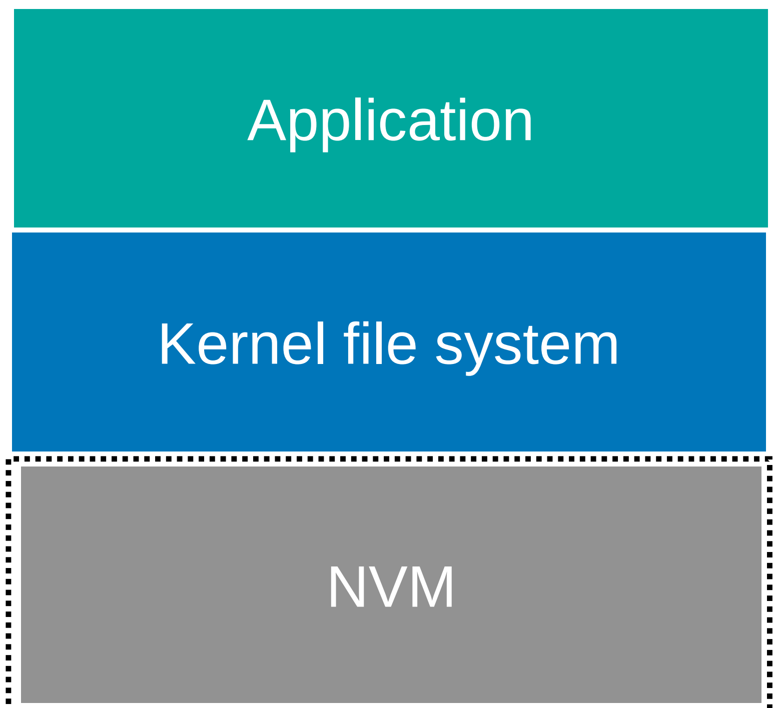
Small, random IO is slow!



NVM is so fast that kernel is the bottleneck

A fast server on today's file system

- Small updates (1 Kbytes) dominate
- **Dataset scales up to 10TB**
- Updates must be crash consistent

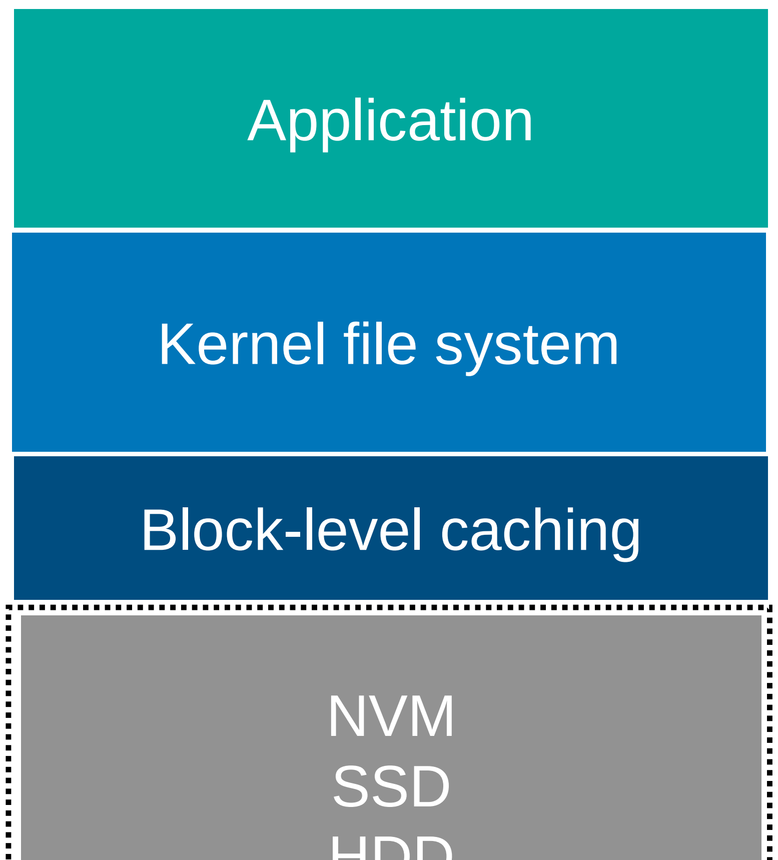


Need huge capacity, but NVM alone is too expensive! ($40K for 10TB)

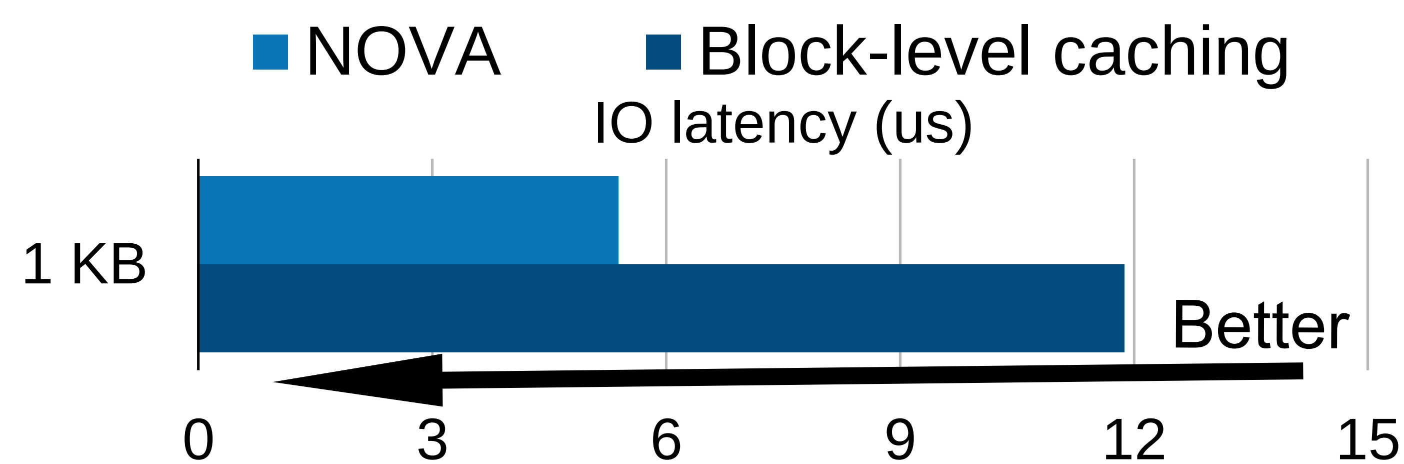
For low-cost capacity with high performance, must leverage multiple device types

A fast server on today's file system

- Small updates (1 Kbytes) dominate
- **Dataset scales up to 10TB**
- Updates must be crash consistent



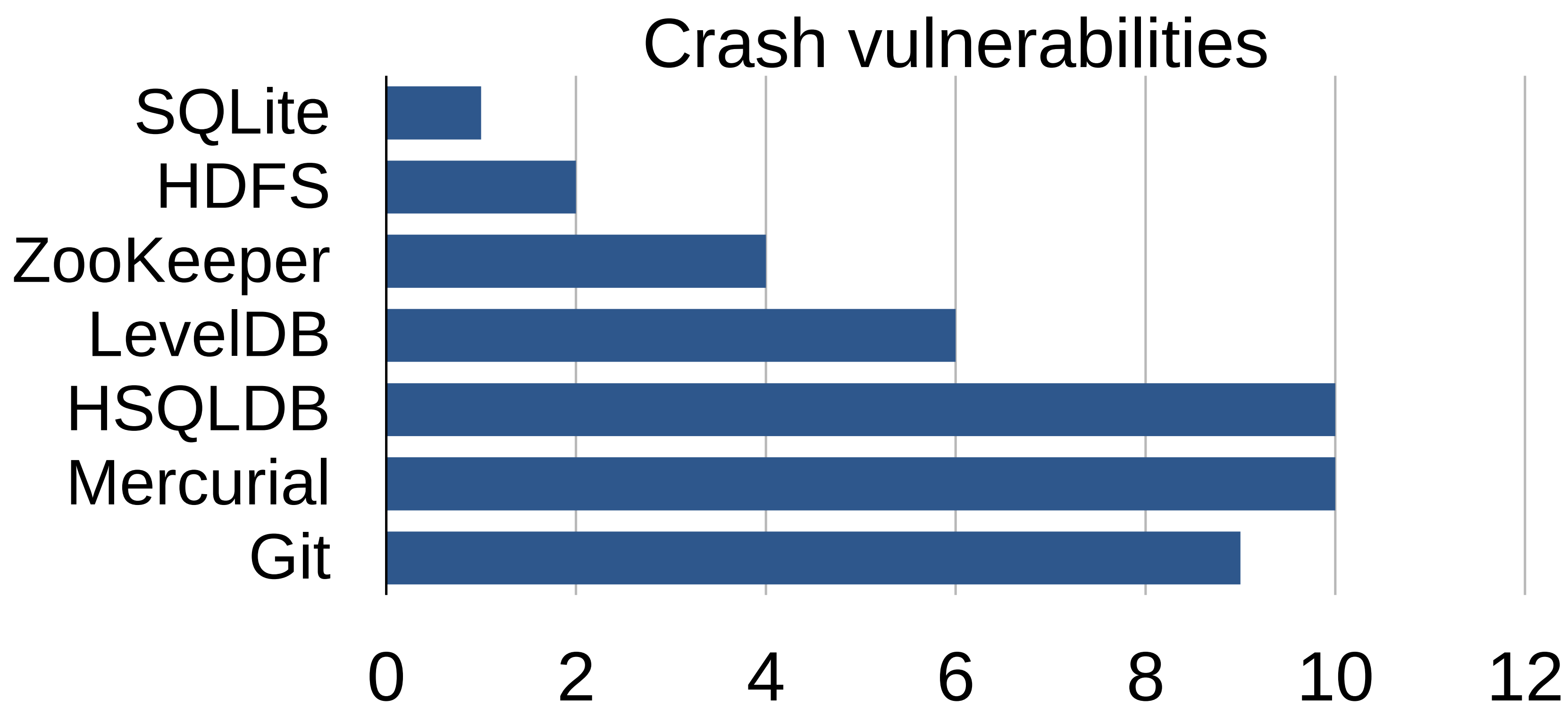
- **Block-level caching manages data in blocks, but NVM is byte-addressable**
- **Extra level of indirection**



Block-level caching is too slow

A fast server on today's file system

- Small updates (1 Kbytes) dominate
- Dataset scales up to 10TB
- **Updates must be crash consistent**



Applications struggle for crash consistency

Problems in today's file systems

- Kernel mediates every operation

  NVM is so fast that kernel is the bottleneck

- Tied to a single type of device

  For low-cost capacity with high performance, must leverage multiple device types

  NVM (soon), SSD, HDD

- Aggressive caching in DRAM, write to device only when you must (fsync)

  Applications struggle for crash consistency

Strata:
A Cross Media File System

Performance: especially small, random IO

- Fast user-level device access

Low-cost capacity: leverage NVM, SSD & HDD

- Transparent data migration across different storage media
- Efficiently handle device IO properties

Simplicity: intuitive crash consistency model

- In-order, synchronous IO
- No fsync() required

Strata's main design principle

LibFS Log operations to NVM at user-level

Performance: Kernel bypass, but private

Simplicity: Intuitive crash consistency

KernelFS Digest and migrate data in kernel

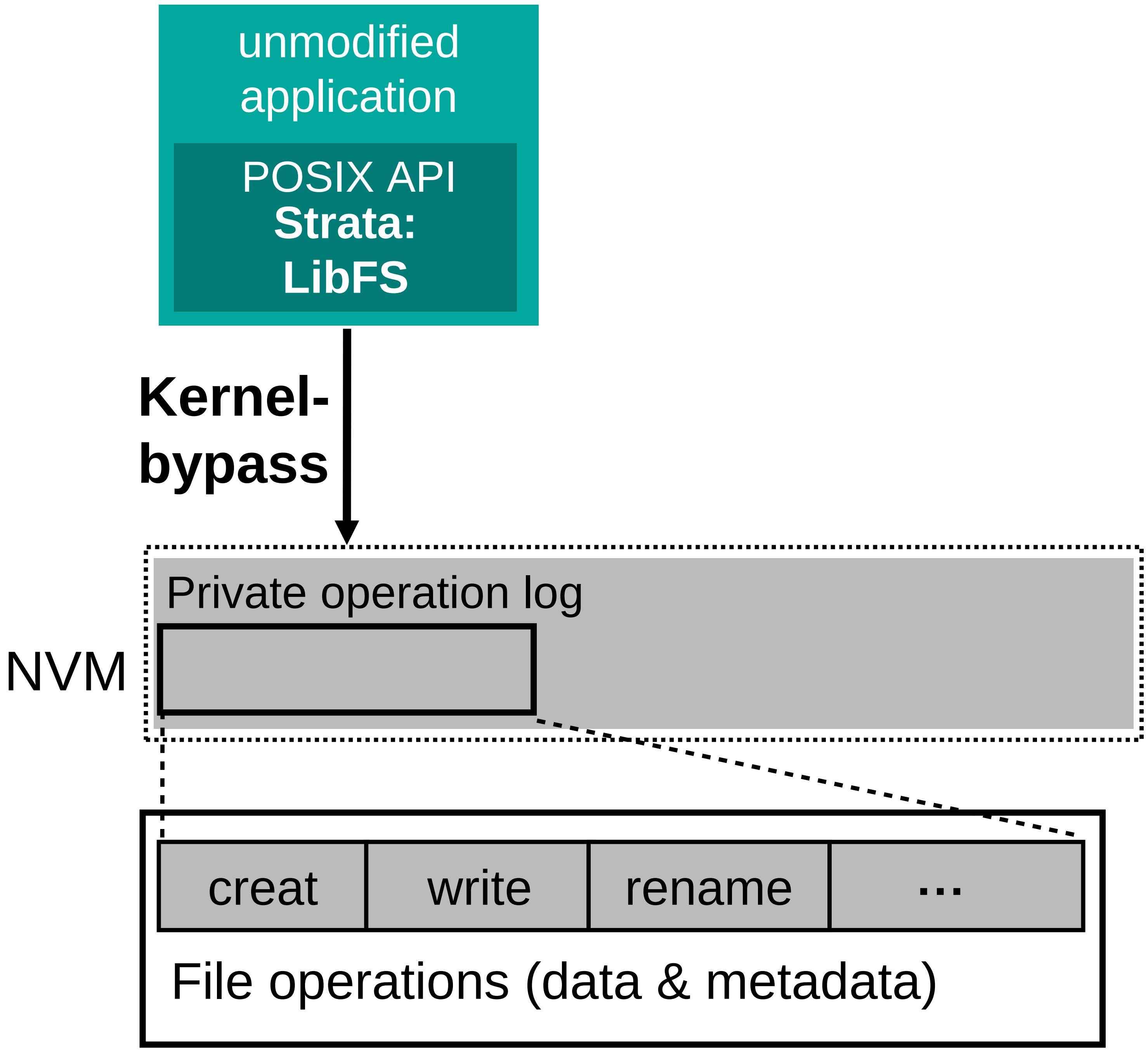
Coordinate multi-process accesses

Apply log operations to shared data

Outline

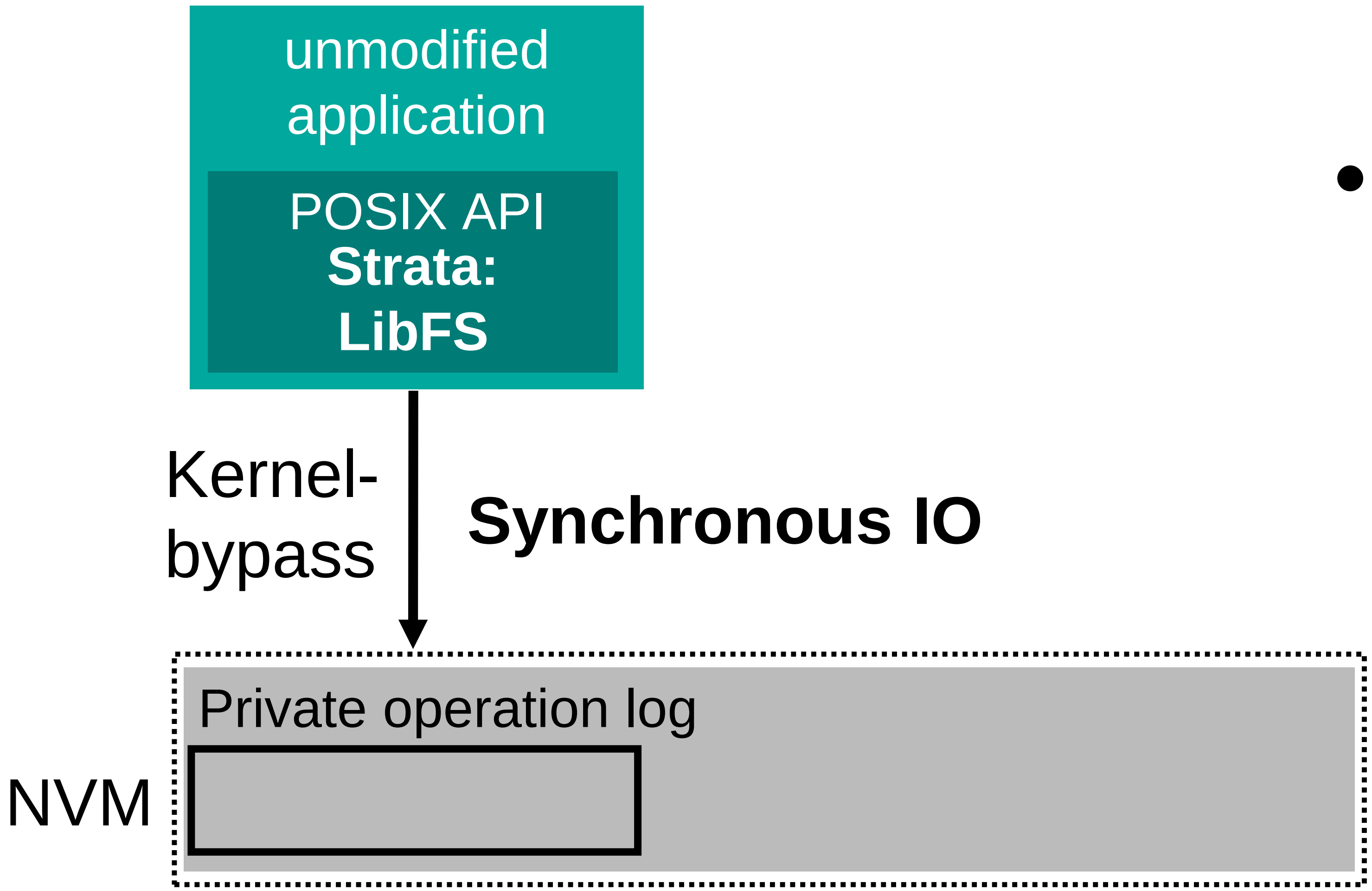
- **LibFS:** Log operations to NVM at user-level
  - Fast user-level access
  - In-order, synchronous IO
- **KernelFS:** Digest and migrate data in kernel
  - Asynchronous digest
  - Transparent data migration
  - Shared file access
- **Evaluation**

Log operations to NVM at user-level



- Fast writes
  - Directly access fast NVM
  - Sequentially append data
  - Cache-line granularity
    - Blind writes
- Crash consistency
  - On crash, kernel replays log

Intuitive crash consistency



- When each system call returns:
  - Data/metadata is durable
  - In-order update
  - Atomic write
    - Limited size (log size)

fsync() is no-op

Fast synchronous IO: NVM and kernel-bypass

Crash consistency example

- File system: EXT4 (ordered mode)
- Assume storage can update 1B atomically

Goal

/strata/file

Foo

Update →

/strata/file

Bar

Possible cases

1. A single write

   write(/strata/file, “Bar”)

Fao

For

…

Not atomic!

Crash consistency example

2. Rollback logging

creat(/strata/log)
write(/strata/log, “Foo”)
write(/strata/file, “Bar”)
unlink(/strata/log)

Reordered and

Possible cases

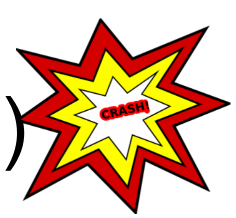
Fao

For

…

3. Rollback logging with ordering

creat(/strata/log)
write(/strata/log, “Foo”)
fsync(/strata/log)
write(/strata/file, “Bar”)
fsync(/strata/file)
unlink(/strata/log)



Possible cases

Fao

For

…

/strata/ may not contain /strata/log

Strata: In-order, synchronous IO with atomicity

4. Correct version

EXT4:

creat(/strata/log)
write(/strata/log, “Foo”)
fsync(/strata/log)
fsync(/strata/)
write(/strata/file, “Bar”)
fsync(/strata/file)
unlink(/strata/log)

Must understand
atomicity, ordering, and durability
(including directory)

Strata:

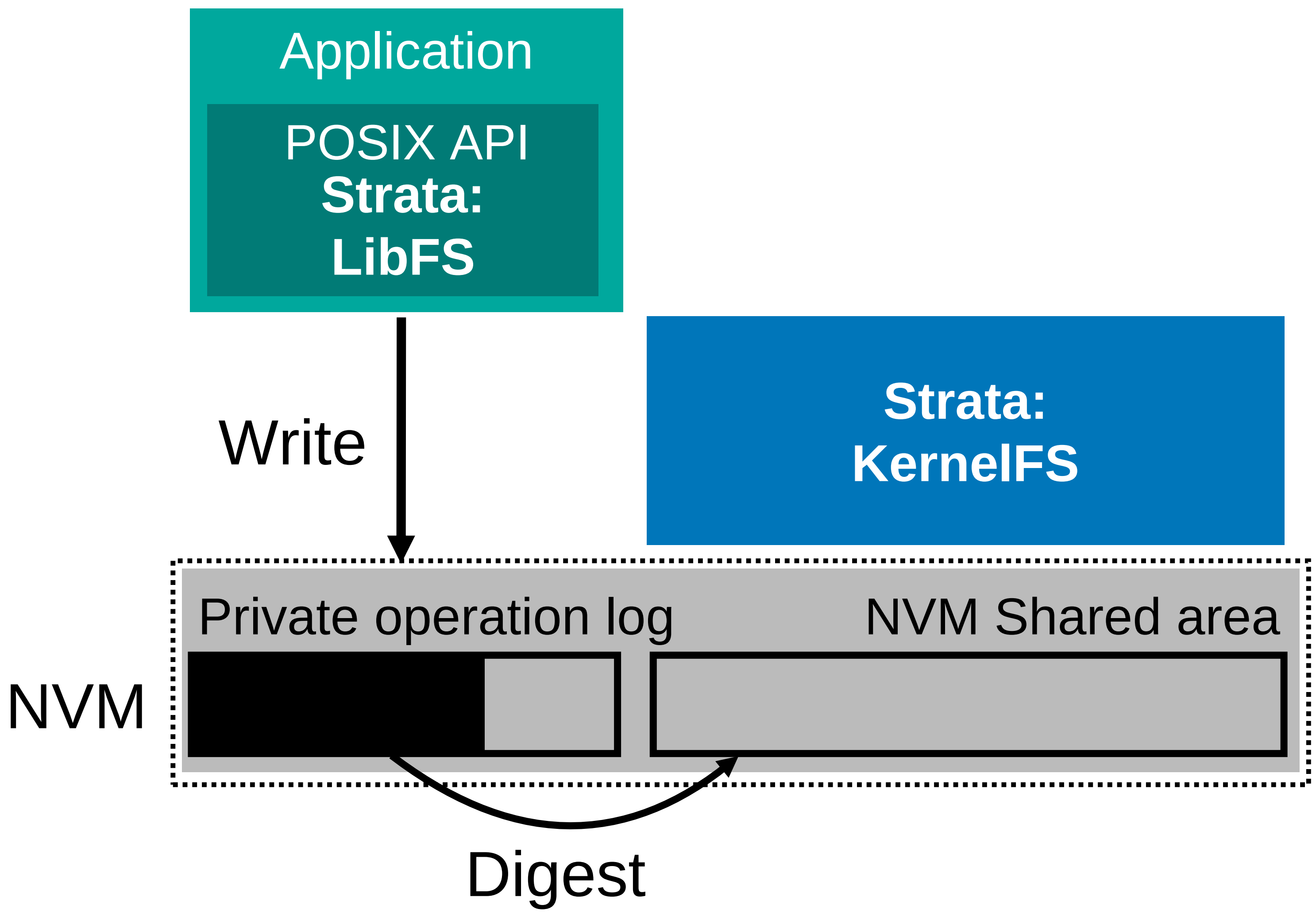
write(/strata/file, “Bar”)

That’s it!

Outline

- **LibFS**: Log operations to NVM at user-level
  - Fast user-level access
  - In-order, synchronous IO
- **KernelFS**: Digest and migrate data in kernel
  - Asynchronous digest
  - Transparent data migration
  - Shared file access
- **Evaluation**

Digest data in kernel



- **Visibility:**
  make private log visible to other applications
- **Data layout:**
  turn write-optimized to read-optimized format (extent tree)
- Large, batched IO
  - Coalesce log

Digest optimization: Log coalescing

SQLite, Mail server: crash consistent update using write ahead logging

Application

Strata: LibFS

Digest eliminates unneeded work

Remove temporary durable writes

Strata: KernelFS

Private operation log

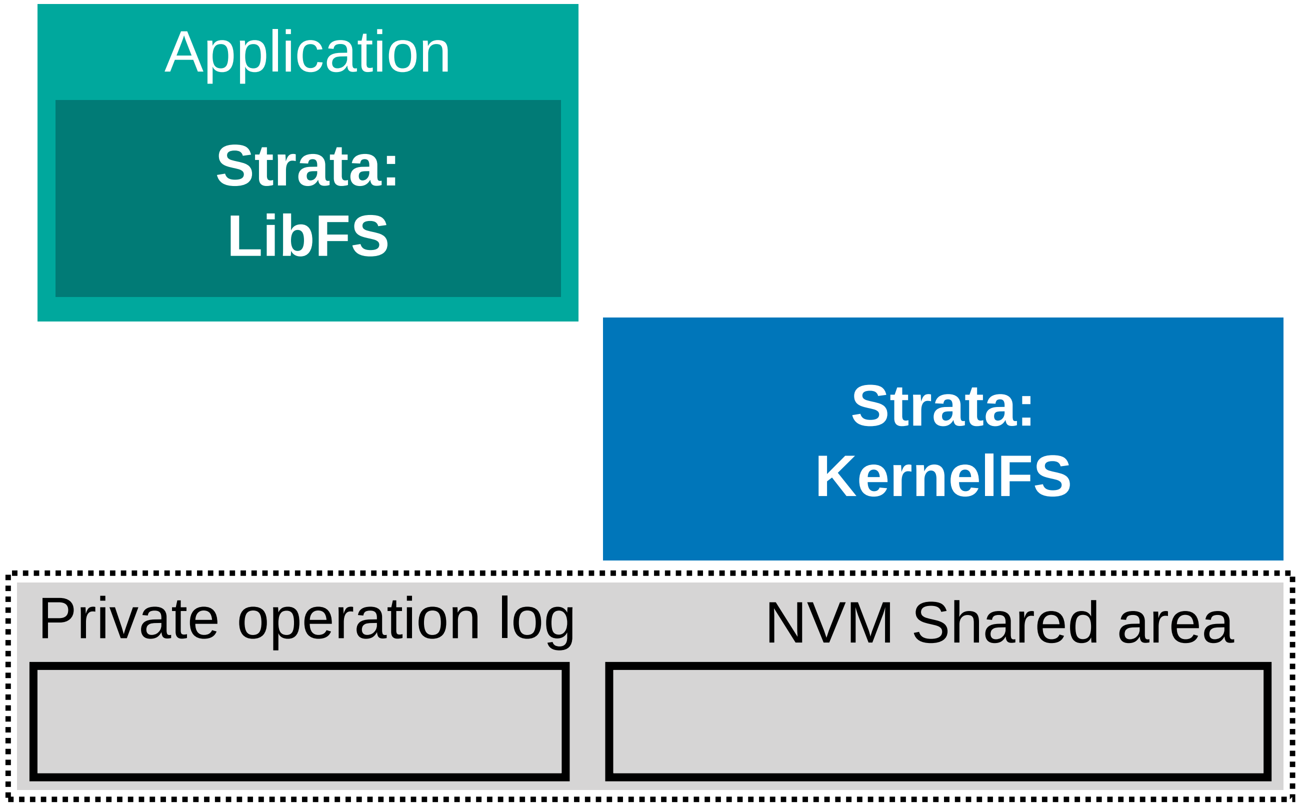
```
Create journal file
Write data to journal file
Write data to database file
Delete journal file
```

NVM Shared area

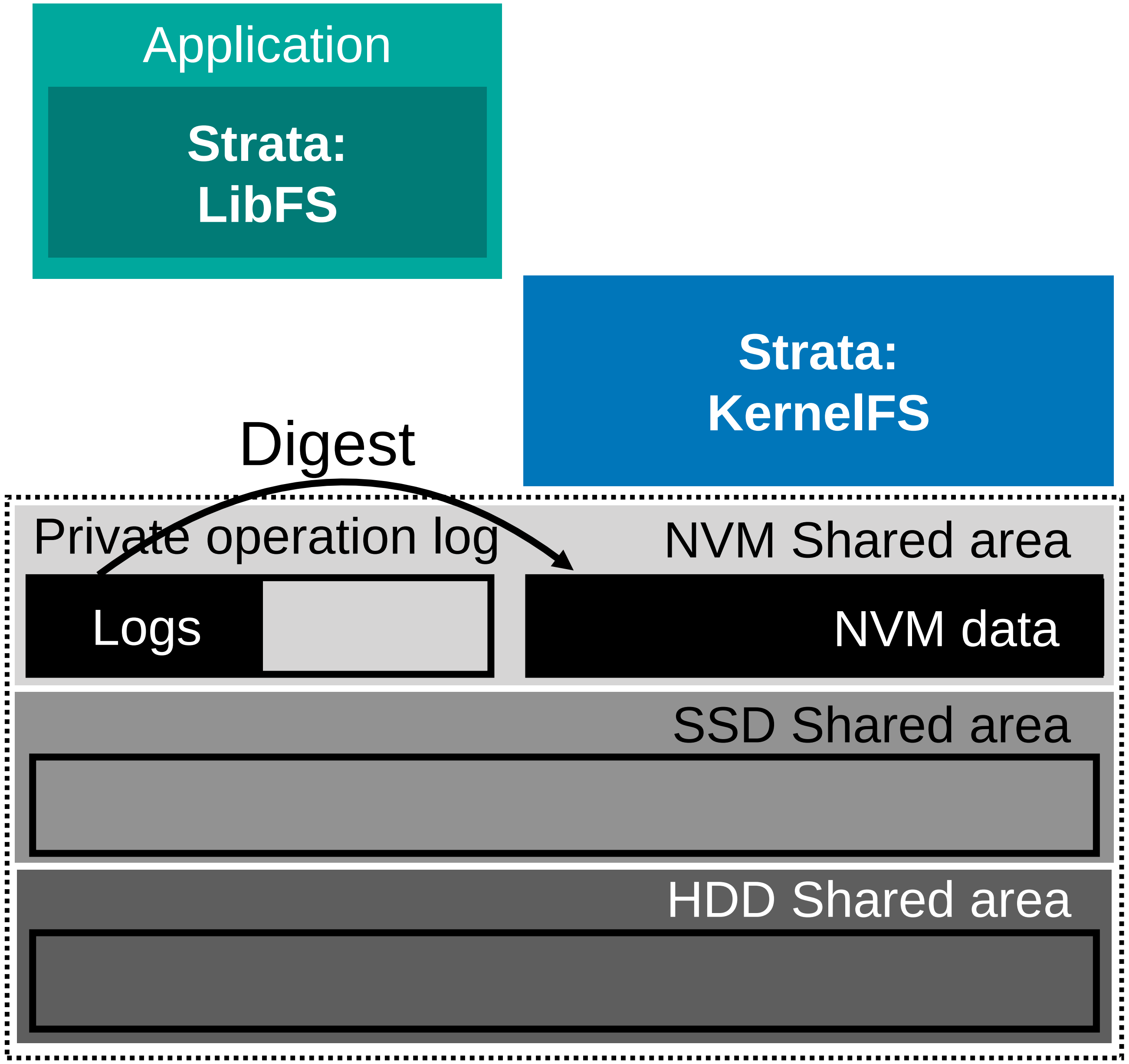
```
Write data to database file
```

Throughput optimization: Log coalescing saves IO while digesting

Digest and migrate data in kernel



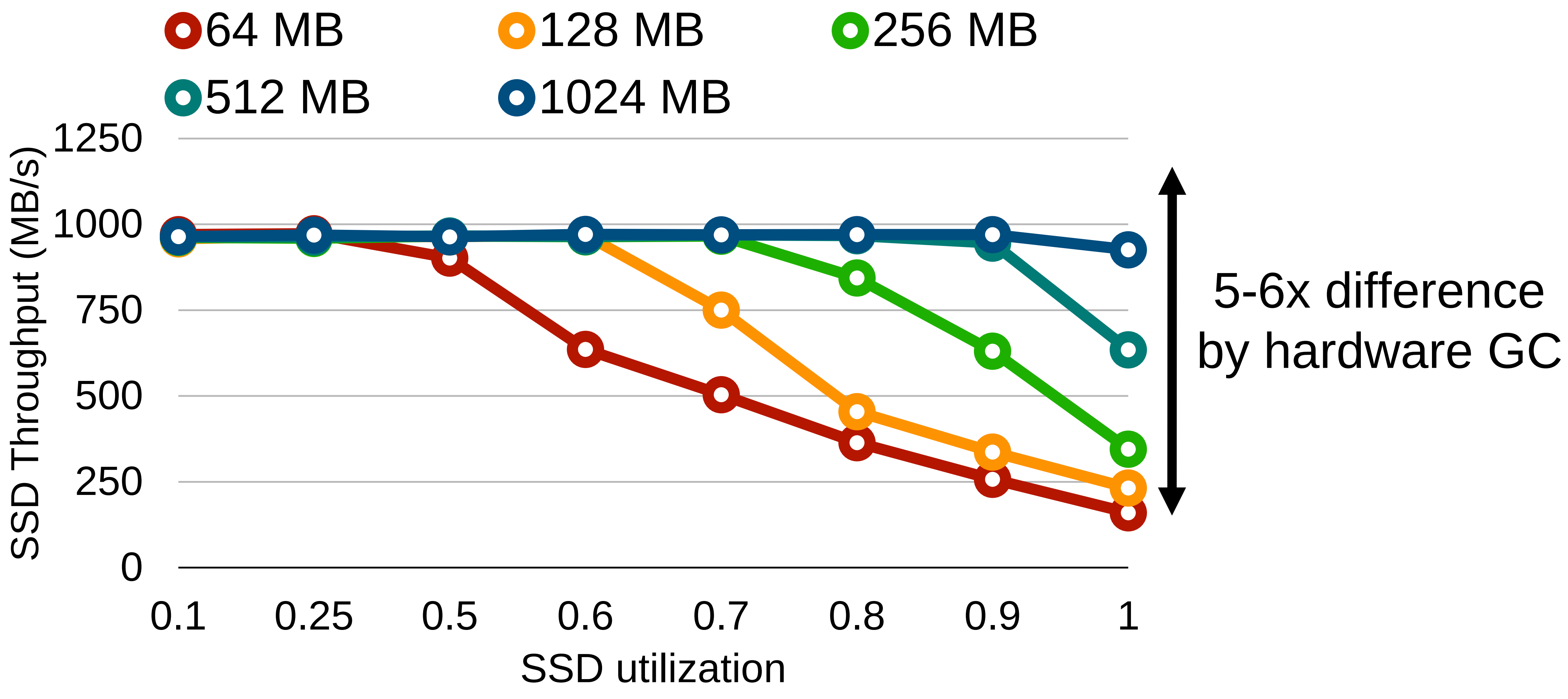
Digest and migrate data in kernel



- Low-cost capacity
  - KernelFS migrates cold data to lower layers
- Handle device IO properties
  - Migrate 1 GB blocks
  - Avoid SSD garbage collection overhead

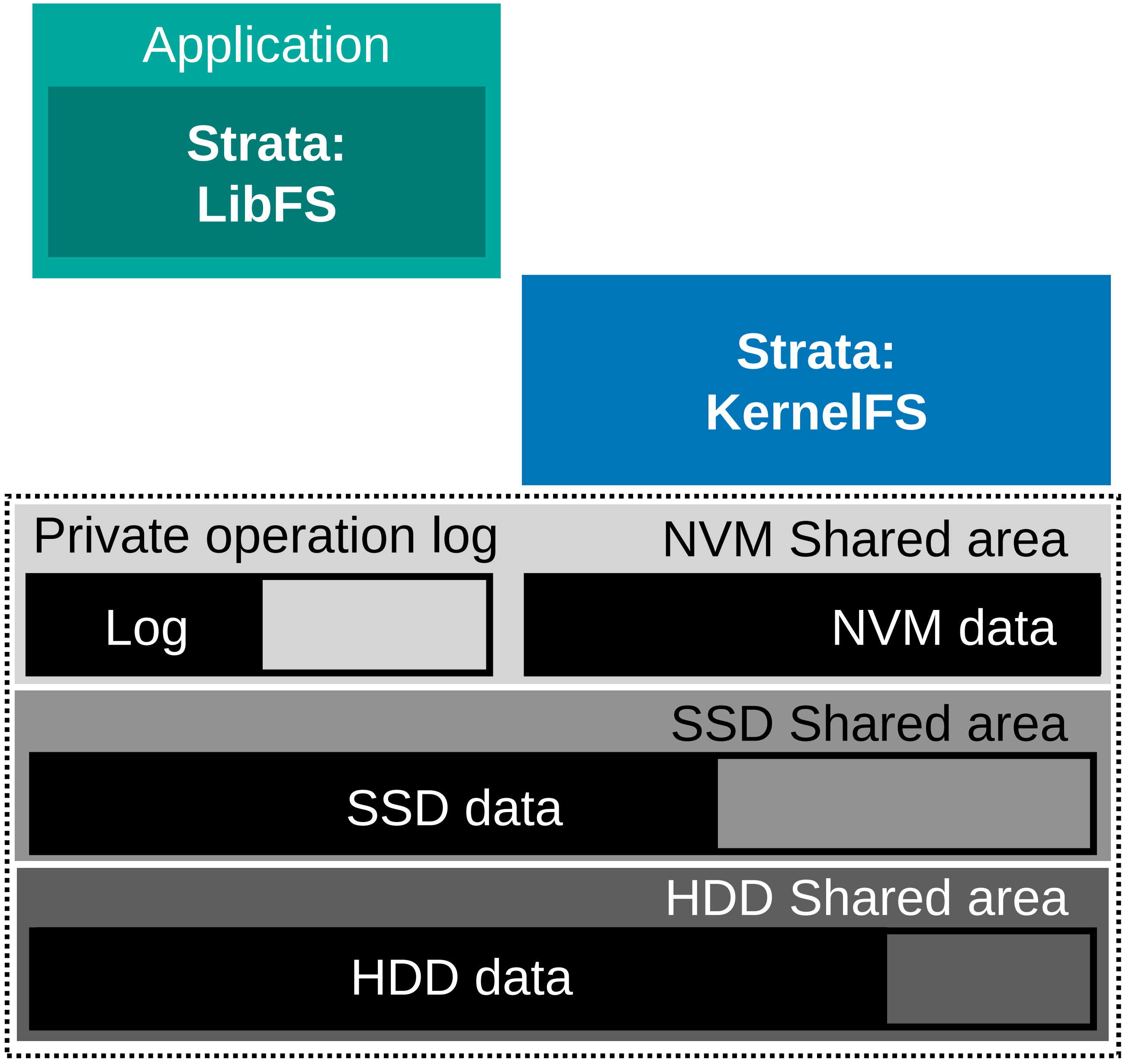
SSD garbage collection overhead

Random overwrite



Large, sequential writes avoid GC

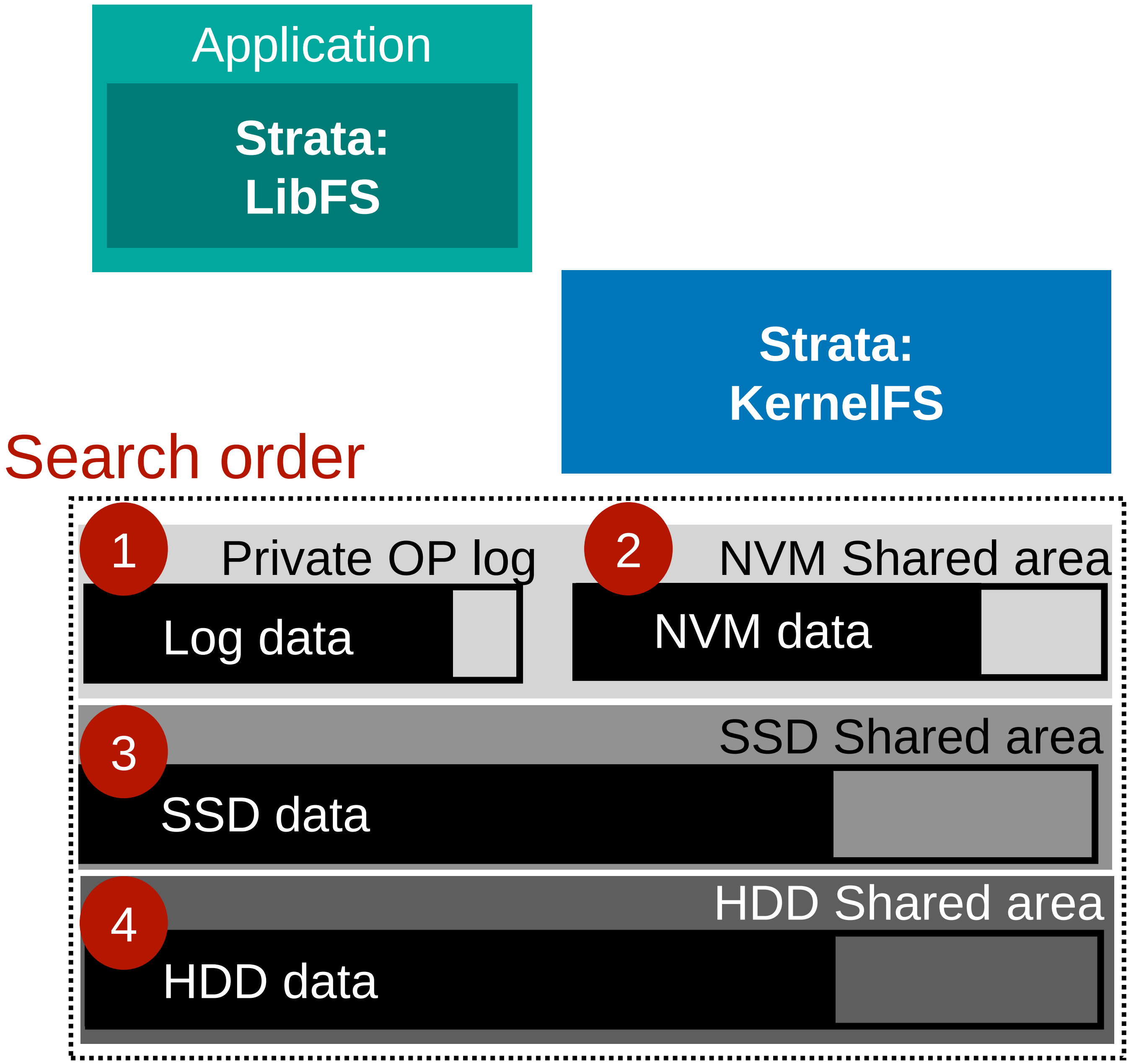
Digest and migrate data in kernel



- Low-cost capacity
  - KernelFS migrates cold data to lower layers
- Handle device IO properties
  - Migrate 1 GB blocks
  - Avoid SSD garbage collection overhead

Higher layers always have up-to-date data

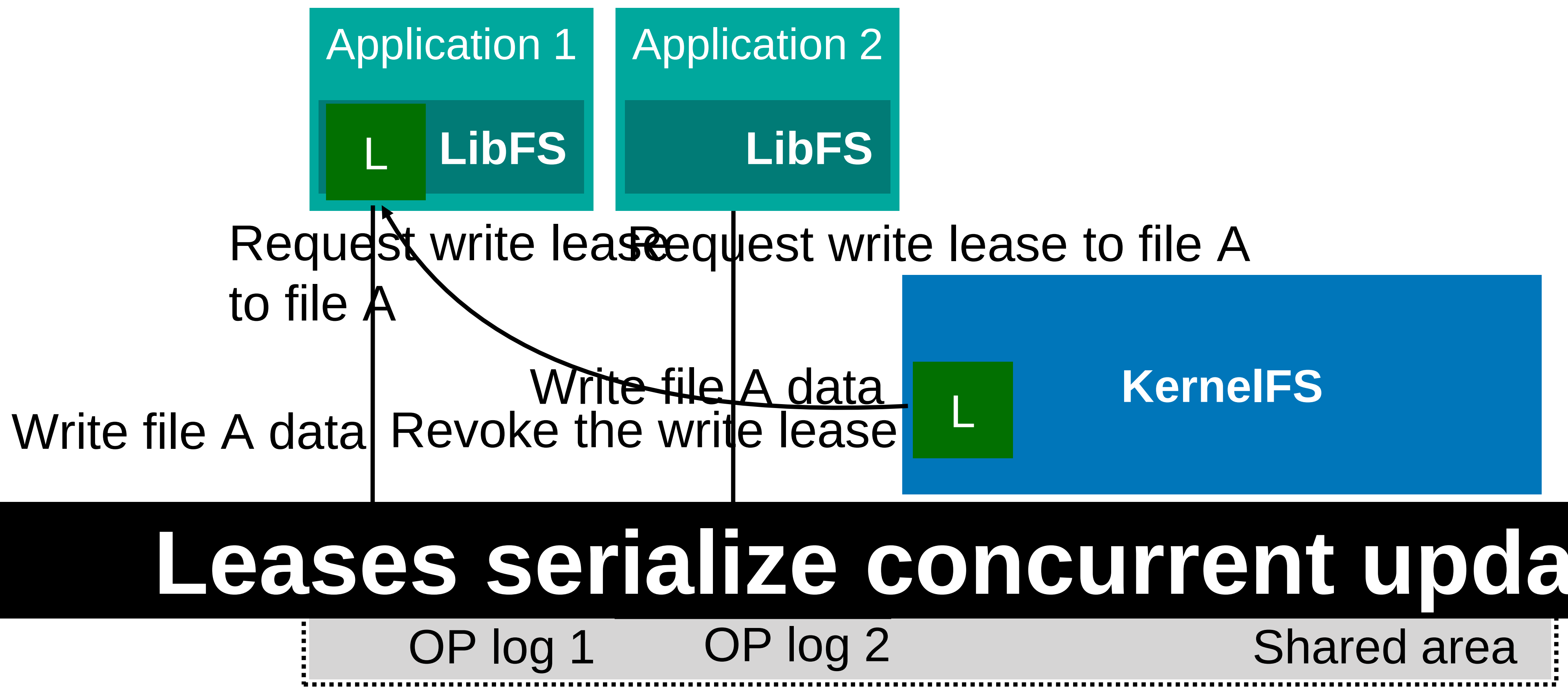
Read: hierarchical search



Shared file access

- **Leases** grant access rights to applications [SOSP'89]
  - Required for files and directories
  - Function like lock, but revocable
  - Exclusive writer, shared readers

Example: concurrent writes to the same file A



Leases serialize concurrent updates

Outline

- **LibFS**: Log operations to NVM at user-level
  - Fast user-level access
  - In-order, synchronous IO
- **KernelFS**: Digest and migrate data in kernel
  - Asynchronous digest
  - Transparent data migration
  - Shared file access
- **Evaluation**

Experimental setup

- 2x Intel Xeon E5-2640 CPU, 64 GB DRAM
  - 400 GB NVMe SSD, 1 TB HDD
- Ubuntu 16.04 LTS, Linux kernel 4.8.12

- Emulated NVM
  - Use 40 GB of DRAM
  - Performance model [Y. Zhang et al. MSST 2015]
    - Throttle latency & throughput in software

Evaluation questions

- **Latency:**
  - Does Strata efficiently support small, random writes?
  - Does asynchronous digest have an impact on latency?
- **Throughput:**
  - Strata writes data twice (logging and digesting).
    Can Strata sustain high throughput?
  - How well does Strata perform
    when managing data across storage layers?

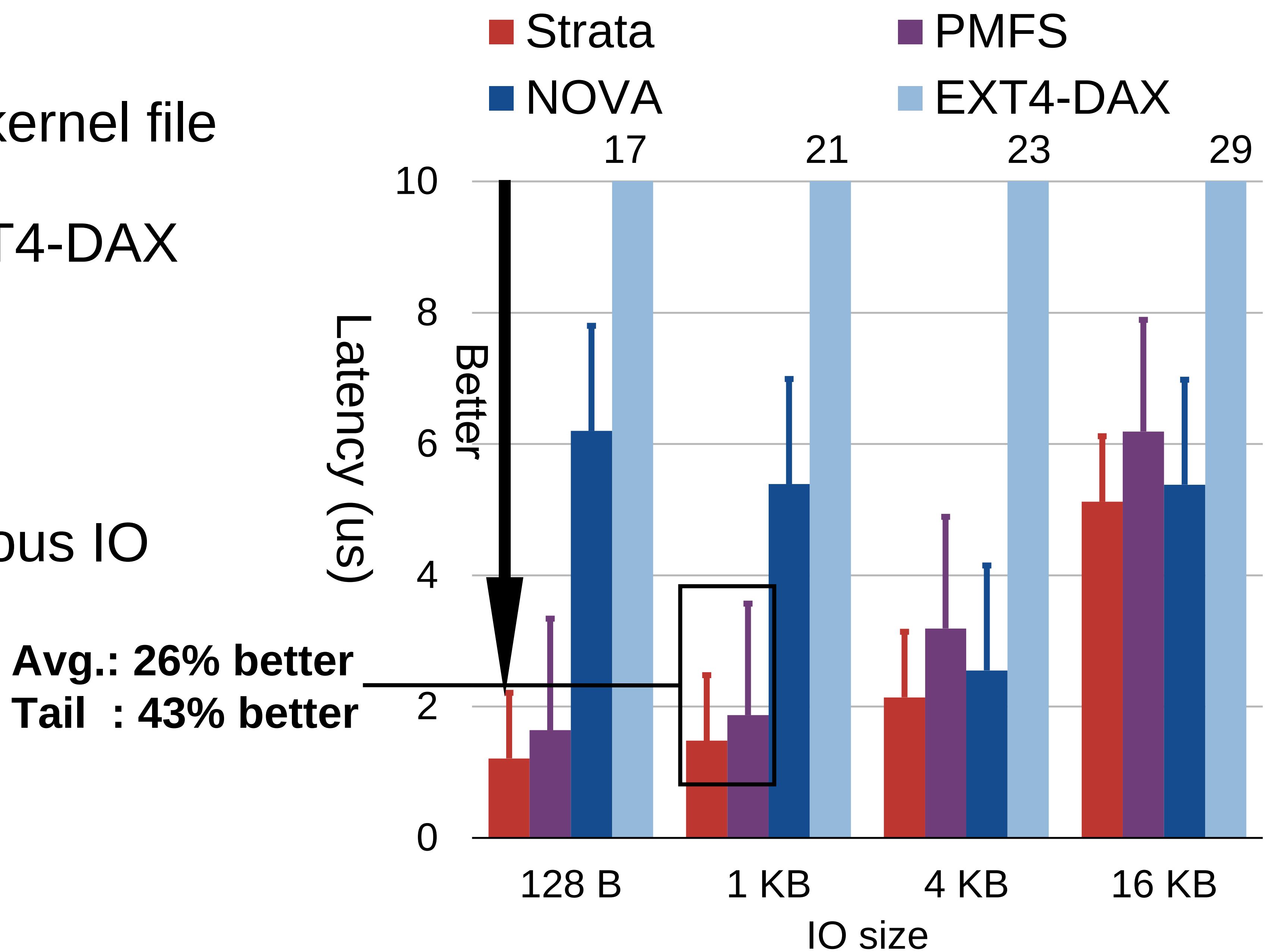
Related work

- **NVM file systems**
  - PMFS[EuroSys 14]: In-place update file system
  - NOVA[FAST 16]: log-structured file system
  - EXT4-DAX: NVM support for EXT4

- **SSD file system**
  - F2FS[FAST 15]: log-structured file system

Microbenchmark: write latency

- Strata logs to NVM
  - Compare to NVM kernel file systems:
    PMFS, NOVA, EXT4-DAX
- Strata, NOVA
  - In-order, synchronous IO
  - Atomic write
- PMFS, EXT4-DAX
  - No atomic write

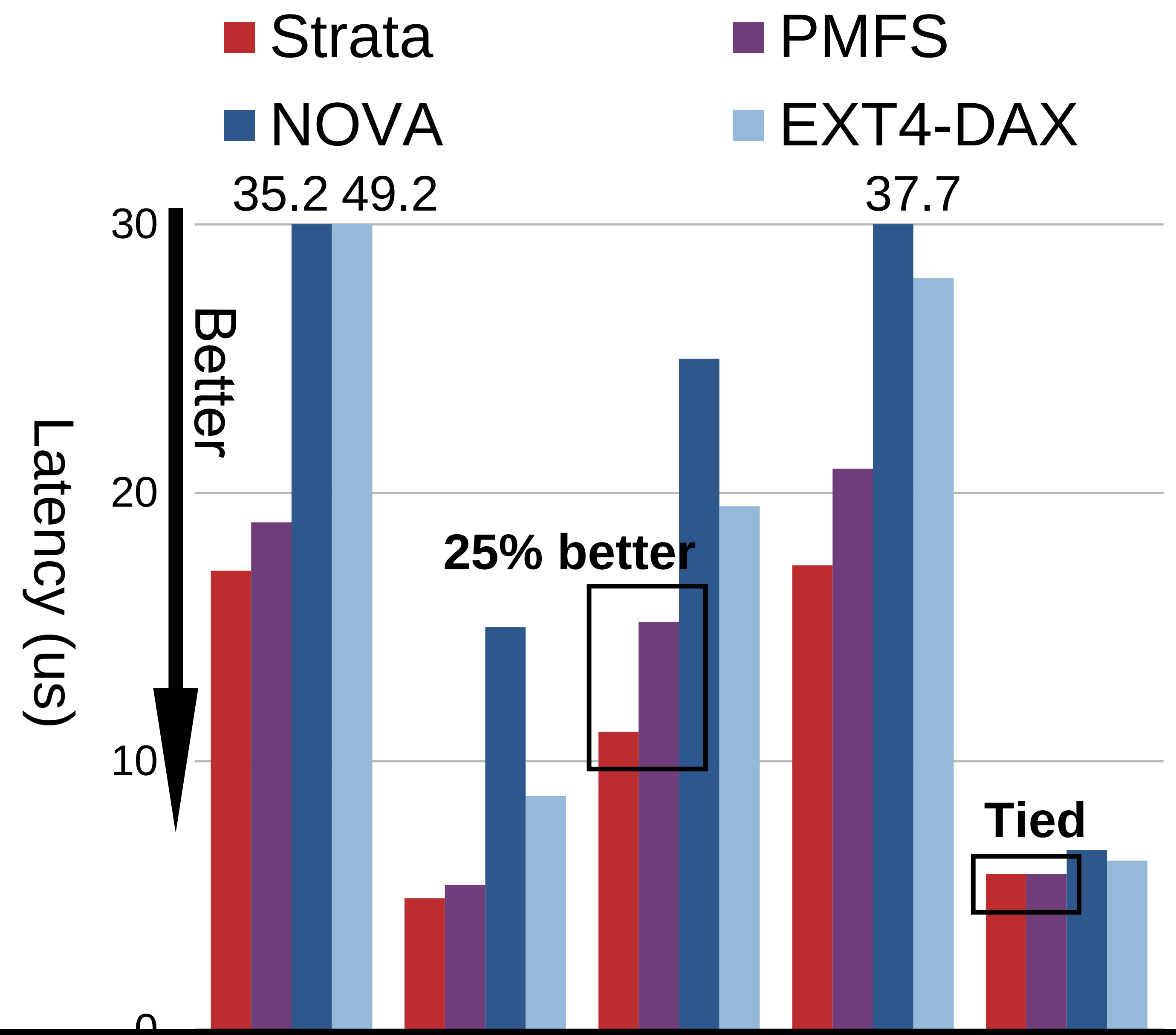


Latency: LevelDB

- LevelDB (NVM)
  - Key size: 16 B
  - Value size: 1 KB
  - 300,000 objects

- **Workload causes asynchronous digests**

- Fast user-level logging
  - Random write
    - 25% better than PMFS
  - Random read



IO latency not impacted by asynchronous digest

Evaluation questions

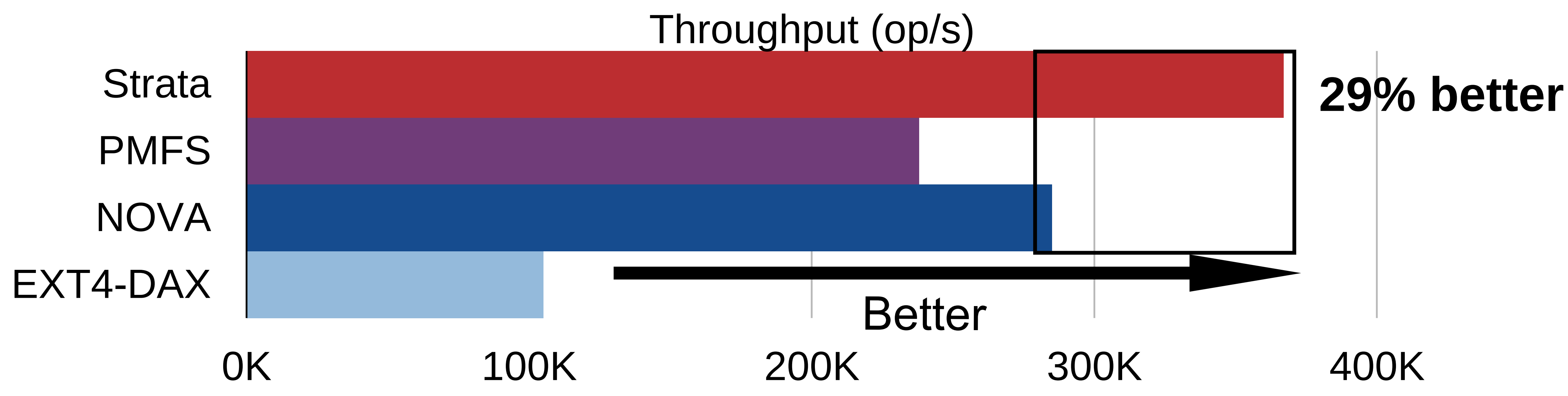
- **Latency:**
  - Does Strata efficiently support small, random writes?
  - Does asynchronous digest have an impact on latency?
- **Throughput:**
  - Strata writes data twice (logging and digesting). Can Strata sustain high throughput?
  - How well does Strata perform when managing data across storage layers?

Throughput: Varmail

No kernel file system has both low latency and high throughput:

- PMFS: better latency
- NOVA: better throughput

Strata achieves both low latency and high throughput



Log coalescing eliminates 86% of log entries, saving 14 GB of IO

Throughput: data migration

File server workload from Filebench

- Working set starts at NVM, grows to SSD, HDD
- Read/Write ratio is 1:2

User-level migration

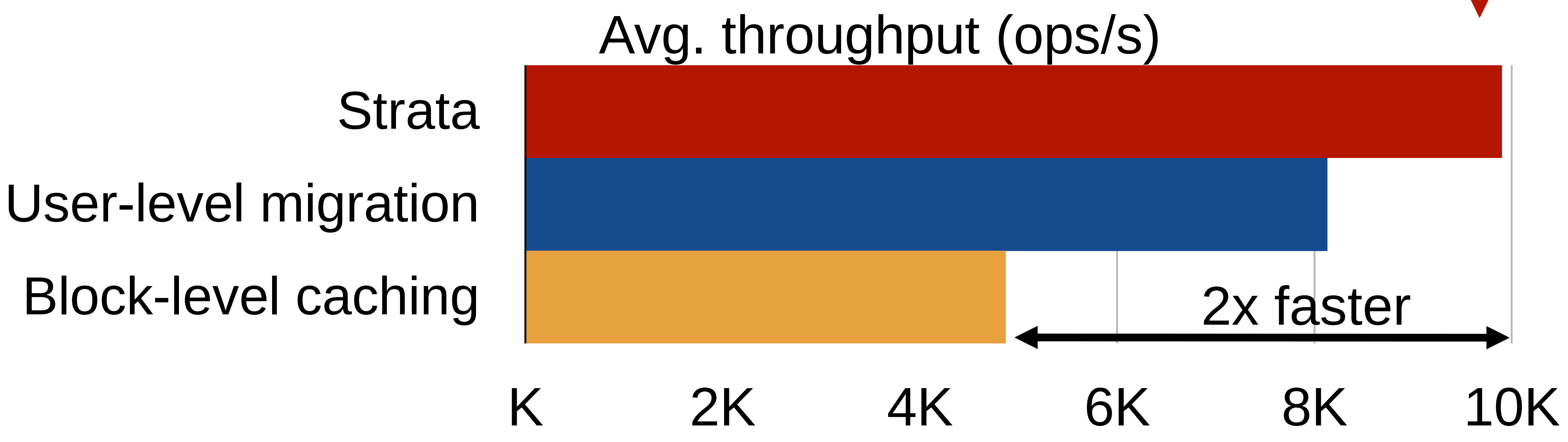
- LRU: whole file granularity
- Treat each file system as a black-box
- NVM: NOVA, SSD: F2FS, HDD: EXT4

Block-level caching

- Linux LVM cache, formatted with F2FS

22% faster than user-level migration

Cross layer optimization: placing hot metadata in faster layers



Before concluding strata, let's re-evaluate design

Good system research should have

Timely problem

Principled approach

General solution

Timely problem?

- Does Strata address system issues for emerging technologies?
- Does Strata address recent applications’ requirement?

Principled approach?

- Show me a sentence or a table to describe your system

	Previous systems	Strata
Performance	Use complex kernel	Kernel-bypass
Low-cost capacity	Designed for a single type of storage	Asynchronous digest Transparent data migration
Simplicity	Complex crash consistency	In-order, synchronous IO

General solution?

- Is Strata a general purpose file system?
- Does Strata work with only NVM?
- Does Strata work with only SSD?

Conclusion

Server applications need fast, small random IO on vast datasets with intuitive crash consistency

Strata, a cross media file system, addresses these concerns

Performance: low latency, high throughput

- Novel split of LibFS, KernelFS
- Fast user-level access

Low-cost capacity: leverage NVM, SSD & HDD

- Asynchronous digest
- Transparent data migration with large, sequential IO

Simplicity: intuitive crash consistency model

- In-order, synchronous IO

Source code is available at
https://github.com/ut-osa/strata
https://github.com/Dahca/strata **(active)**