Strata:
A Cross Media File System

Youngjin Kwon, Henrique Fingler, Tyler Hunt,
Simon Peter, Emmett Witchel, Thomas Anderson

The University of Texas at Austin

UNIVERSITY of WASHINGTON

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

	Latency	$/GB
DRAM	100 ns	8.6
NVM (soon)	300 ns	4.0
SSD	10 us	0.25
HDD	10 ms	0.02

Storage diversification

Byte-addressable: cache-line granularity IO

	Latency	$/GB
DRAM	100 ns	8.6
NVM (soon)	300 ns	4.0
SSD	10 us	0.25
HDD	10 ms	0.02

Better performance

Higher capacity

Storage diversification

Byte-addressable: cache-line granularity IO

	Latency	$/GB
DRAM	100 ns	8.6
NVM (soon)	300 ns	4.0
SSD	10 us	0.25
HDD	10 ms	0.02

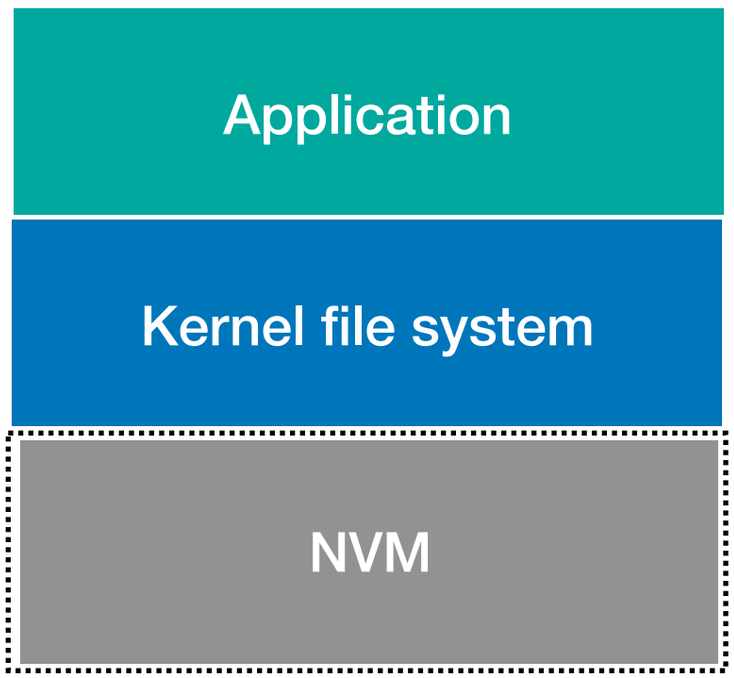
Better performance

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

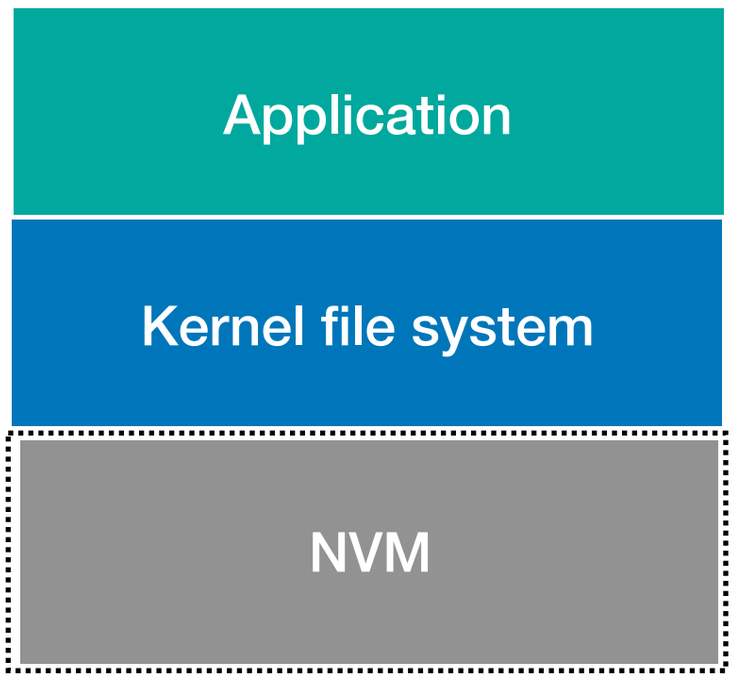
- **Small updates (1 Kbytes) dominate**
- Dataset scales up to 10TB
- Updates must be crash consistent



Kernel file system:
NOVA [FAST 16, SOSP 17]

A fast server on today's file system

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

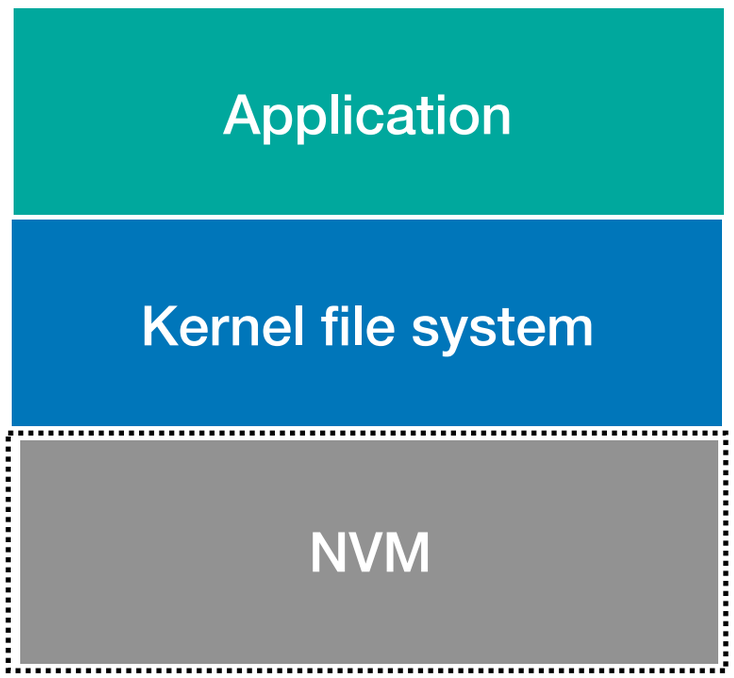


Small, random IO is slow!

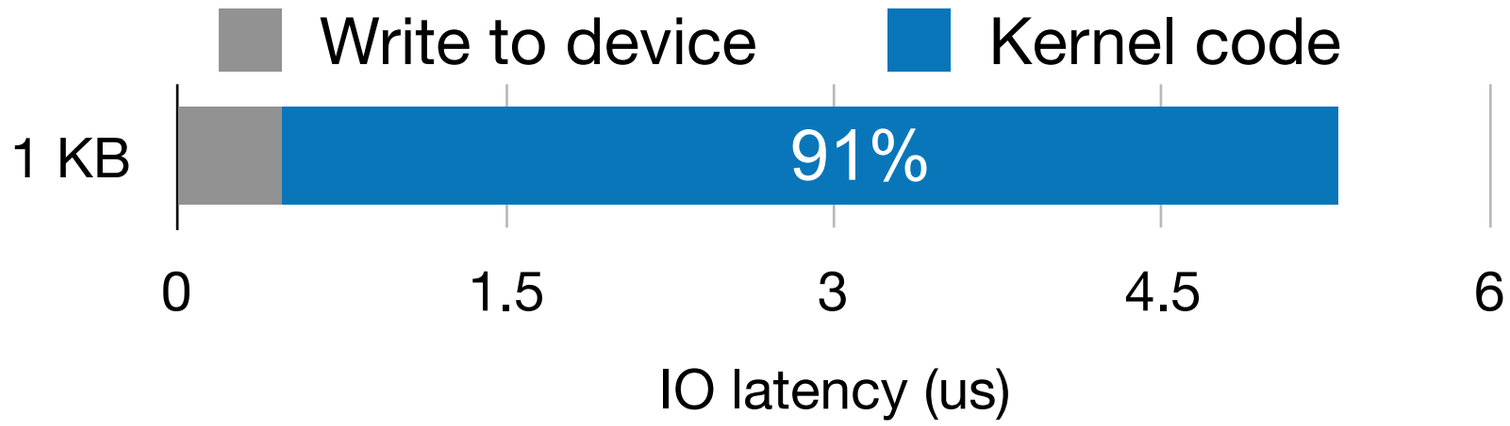
Kernel file system:
NOVA [FAST 16, SOSP 17]

A fast server on today's file system

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



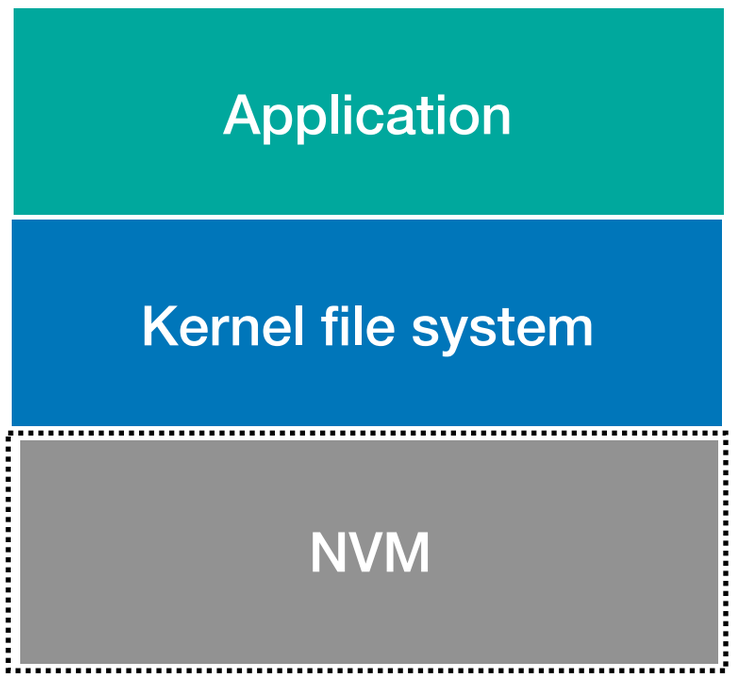
Small, random IO is slow!



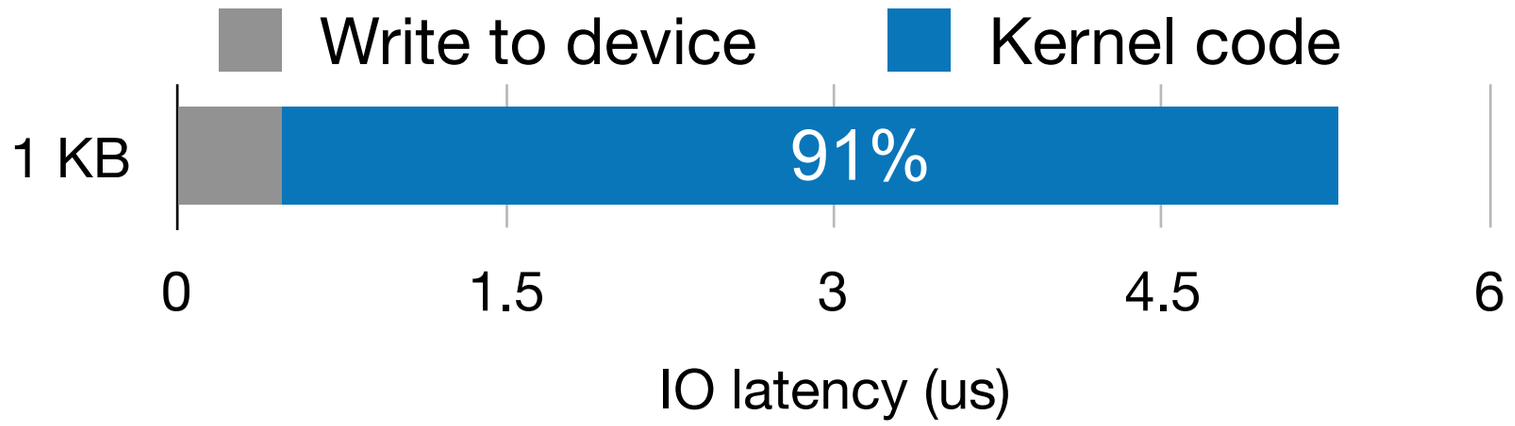
Kernel file system:
NOVA [FAST 16, SOSP 17]

A fast server on today's file system

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



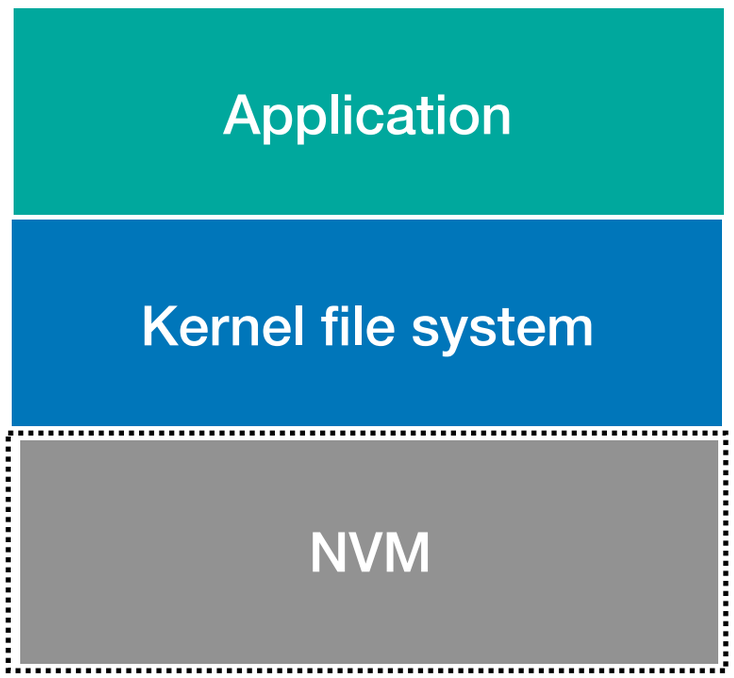
Small, random IO is slow!



NVM is so fast that kernel is the bottleneck

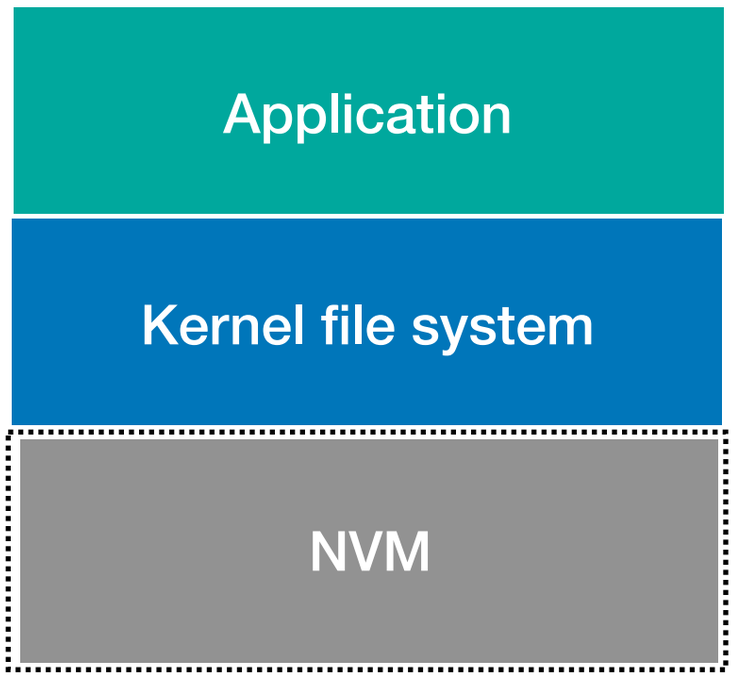
A fast server on today's file system

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



A fast server on today's file system

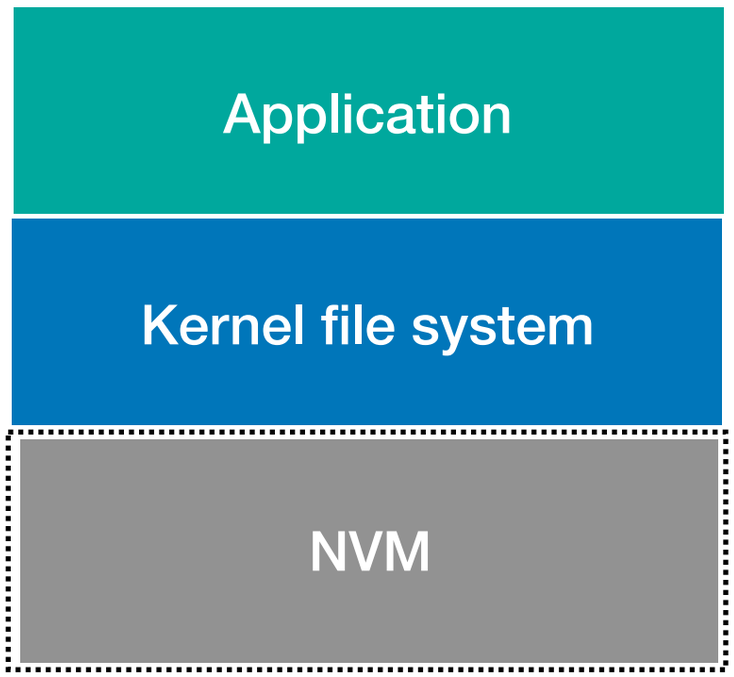
- Small updates (1 Kbytes) dominate
- **Dataset scales up to 10TB**
- Updates must be crash consistent



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

A fast server on today's file system

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

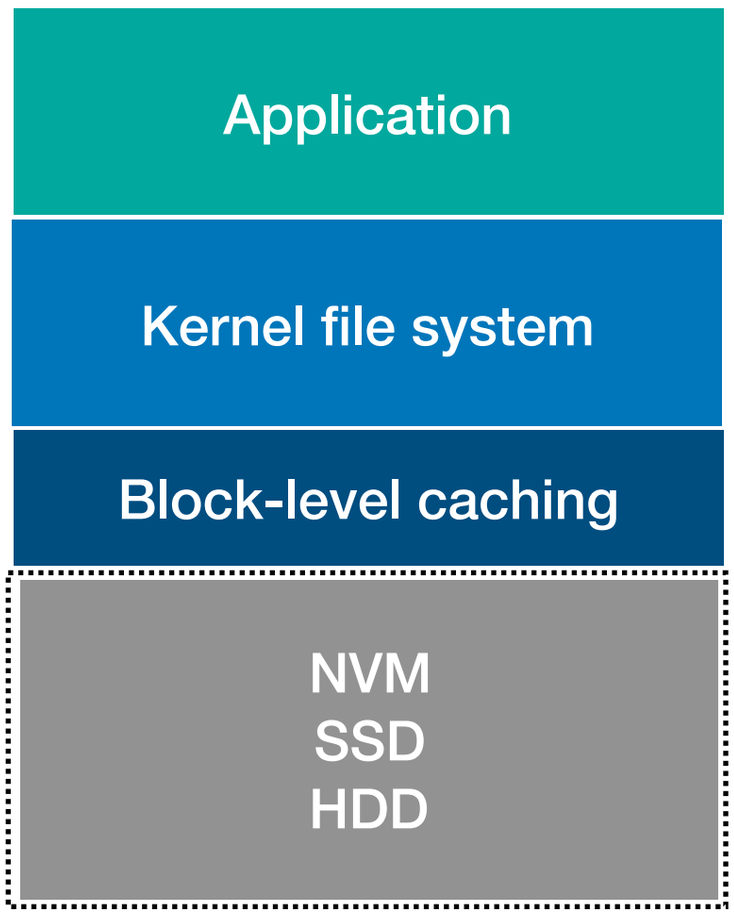


Need huge capacity, but NVM alone is too expensive! ($40K for 10TB)

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

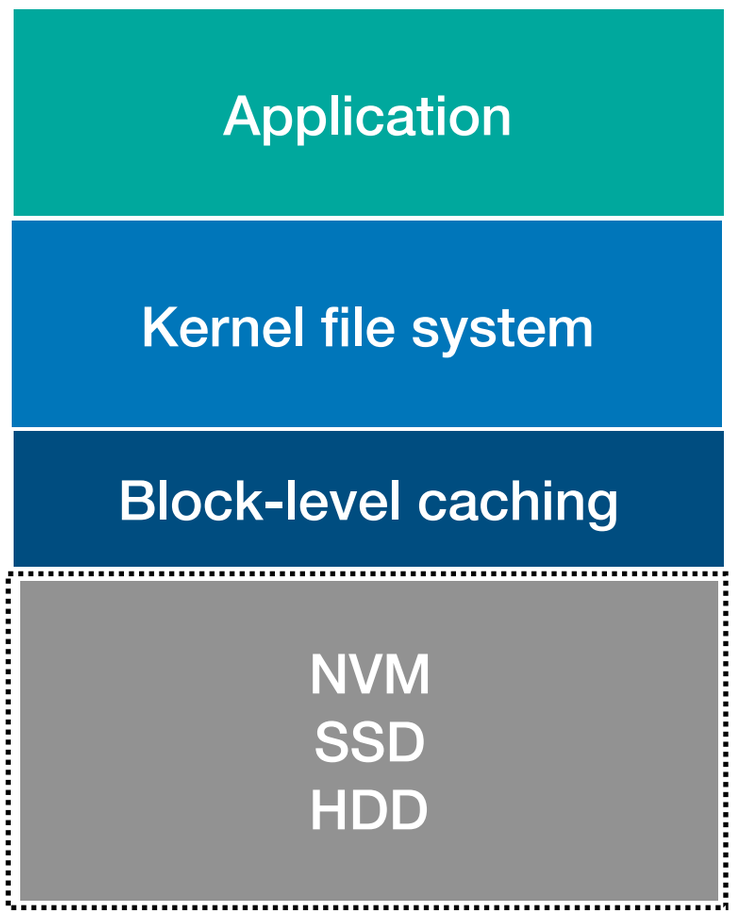
A fast server on today's file system

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



A fast server on today's file system

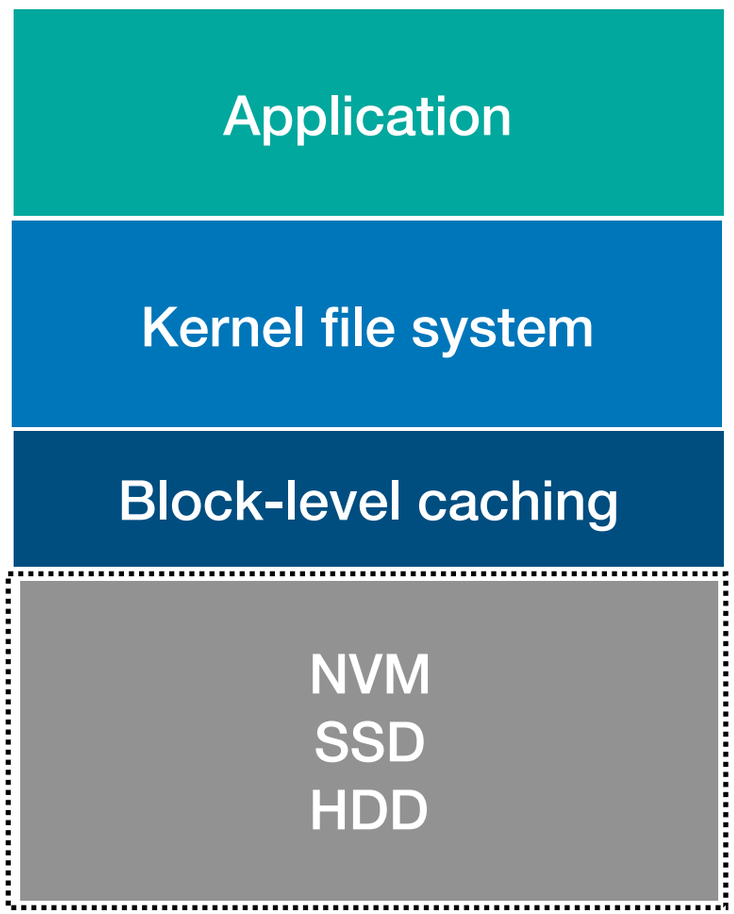
- Small updates (1 Kbytes) dominate
- **Dataset scales up to 10TB**
- Updates must be crash consistent



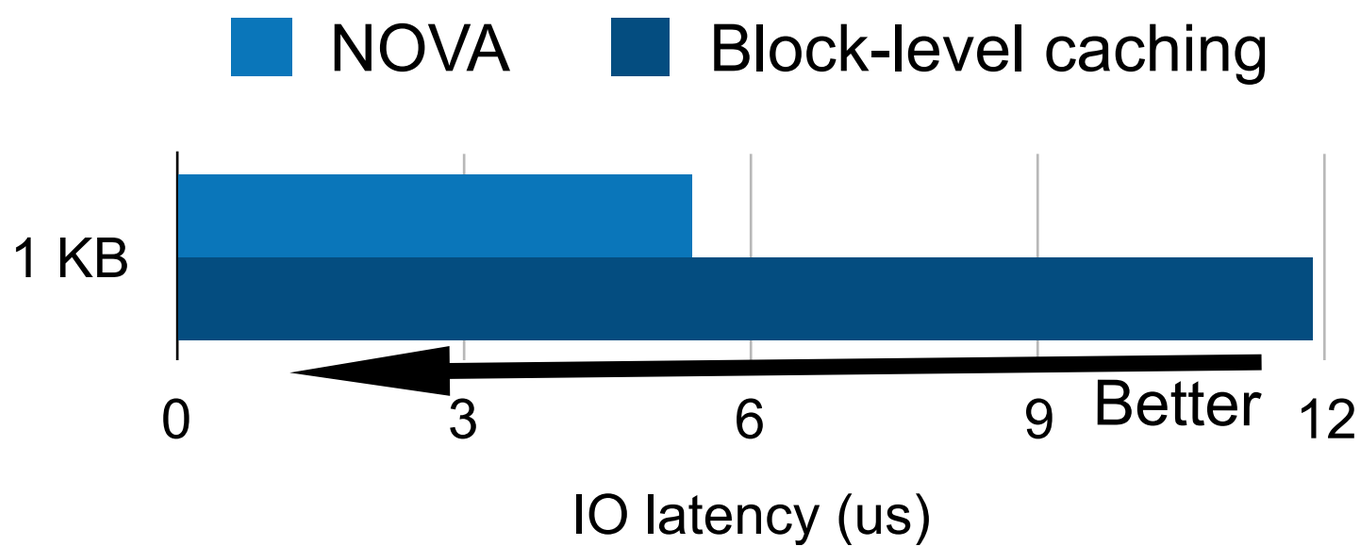
- **Block-level caching manages data in blocks, but NVM is byte-addressable**
- **Extra level of indirection**

A fast server on today's file system

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

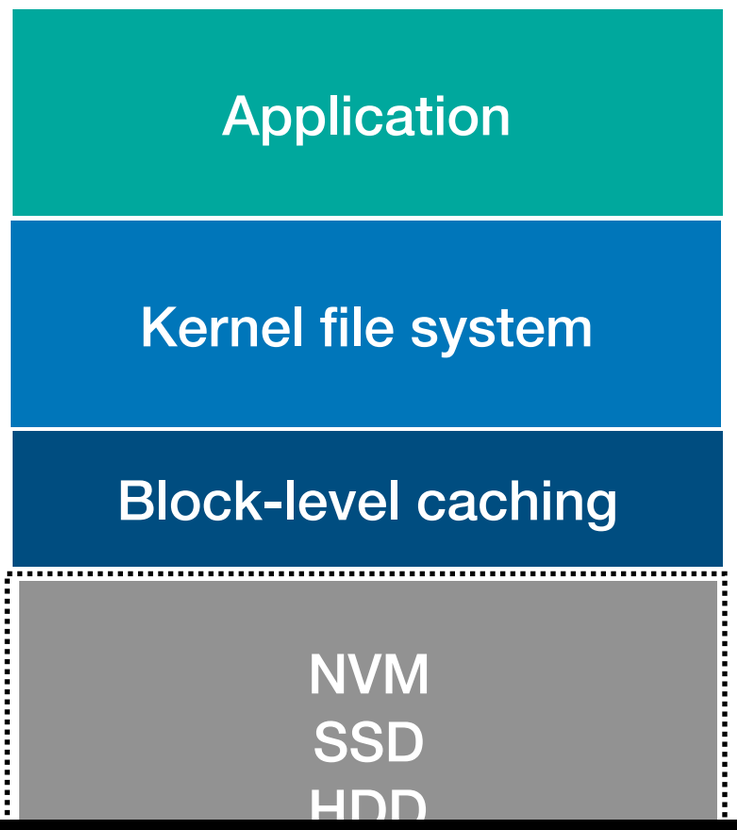


- **Block-level caching manages data in blocks, but NVM is byte-addressable**
- **Extra level of indirection**

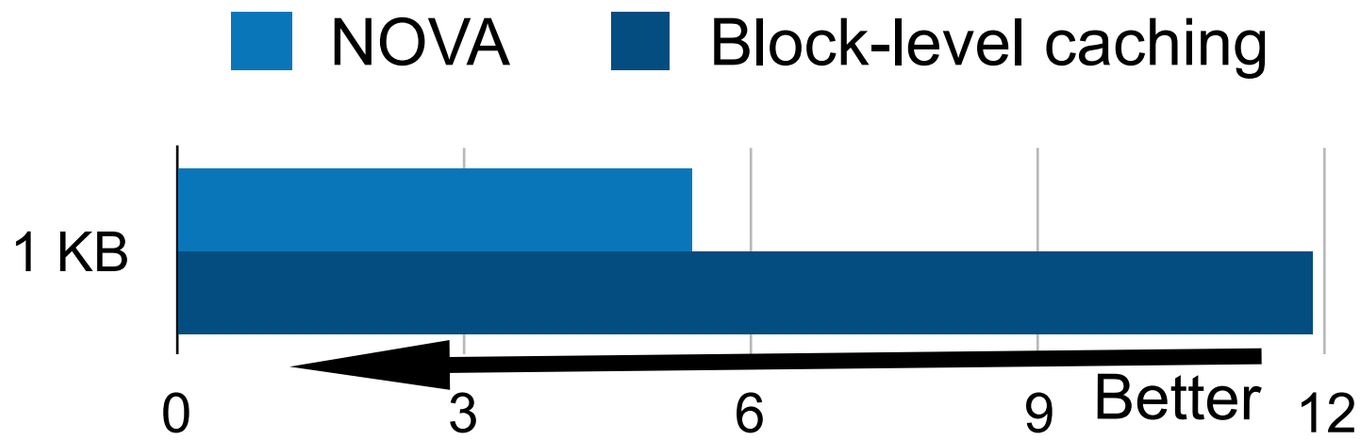


A fast server on today's file system

- 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**



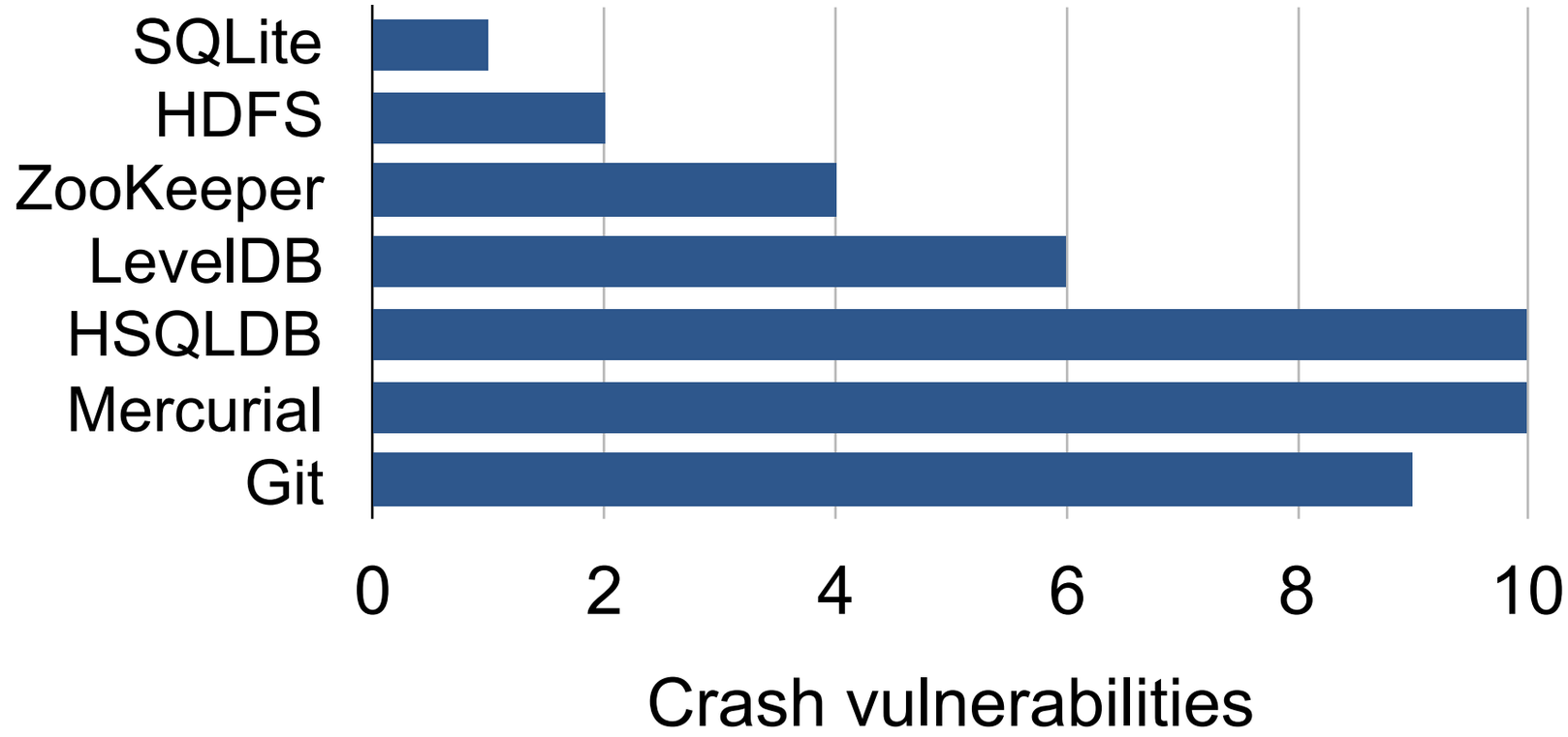
Block-level caching is too slow

A fast server on today's file system

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

A fast server on today's file system

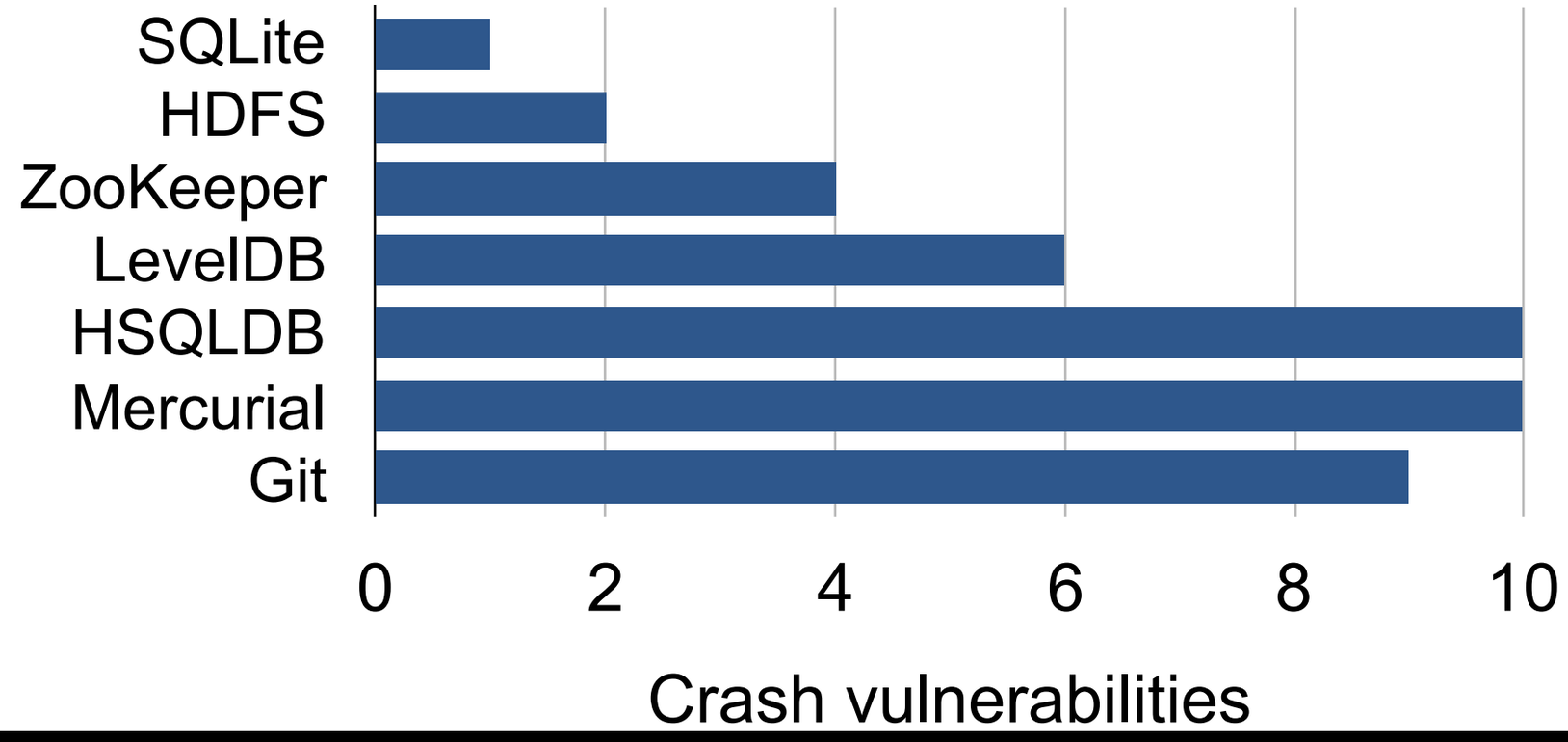
- Small updates (1 Kbytes) dominate
- Dataset scales up to 10TB
- **Updates must be crash consistent**



Pillai et al., OSDI 2014

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: main design principle

Log operations to NVM at user-level

Digest and migrate data in kernel

Strata: main design principle

Log operations to NVM at user-level

Performance: Kernel bypass, but private

Digest and migrate data in kernel

Strata: main design principle

Log operations to NVM at user-level

Simplicity: Intuitive crash consistency

Performance: Kernel bypass, but private

Digest and migrate data in kernel

Strata: main design principle

Log operations to NVM at user-level

Performance: Kernel bypass, but private

Simplicity: Intuitive crash consistency

Digest and migrate data in kernel

Coordinate multi-process accesses

Strata: main design principle

Log operations to NVM at user-level

Performance: Kernel bypass, but private

Simplicity: Intuitive crash consistency

Digest and migrate data in kernel

Apply log operations to shared data

Coordinate multi-process accesses

Strata: 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

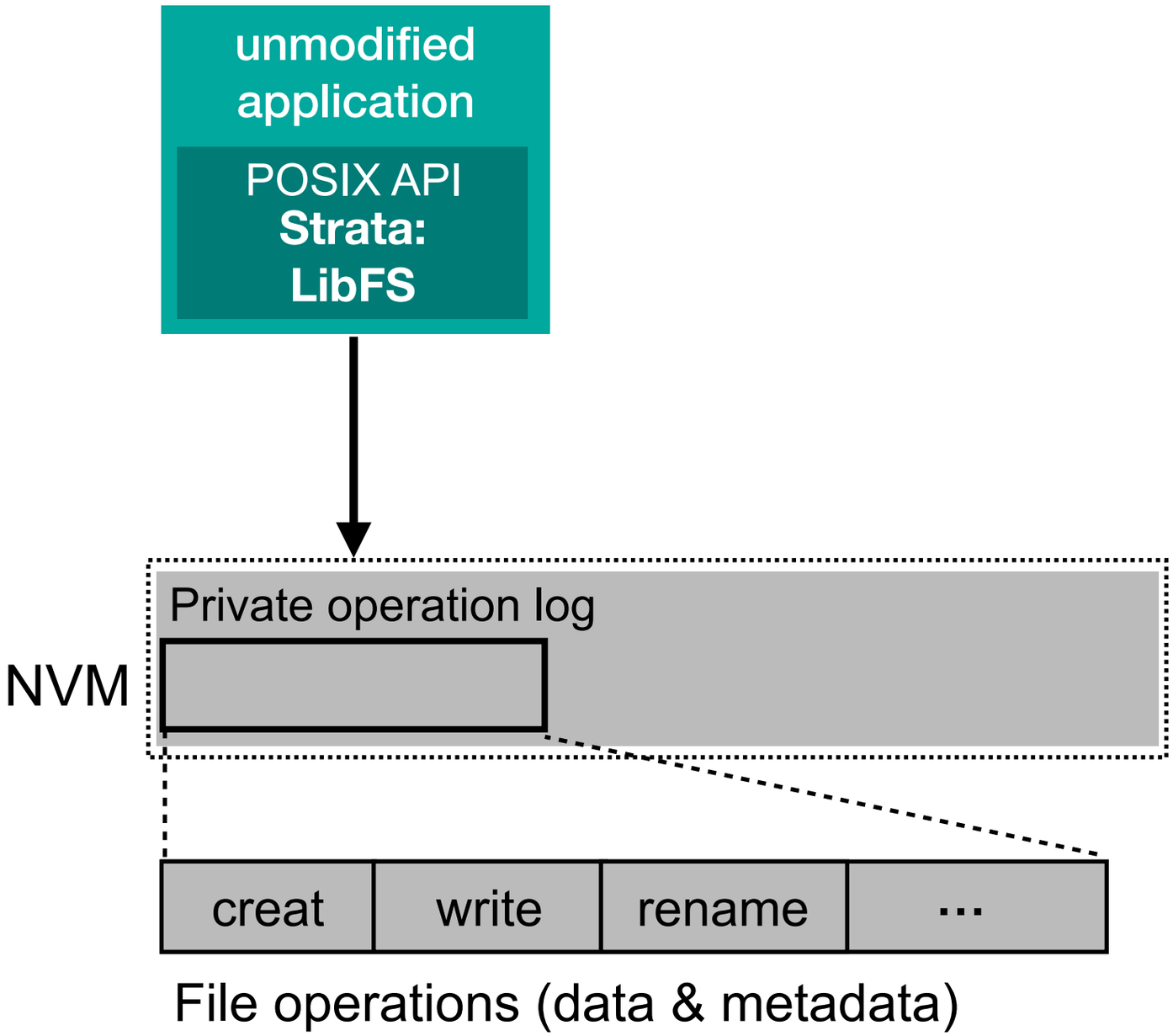
Apply log operations to shared data

Coordinate multi-process accesses

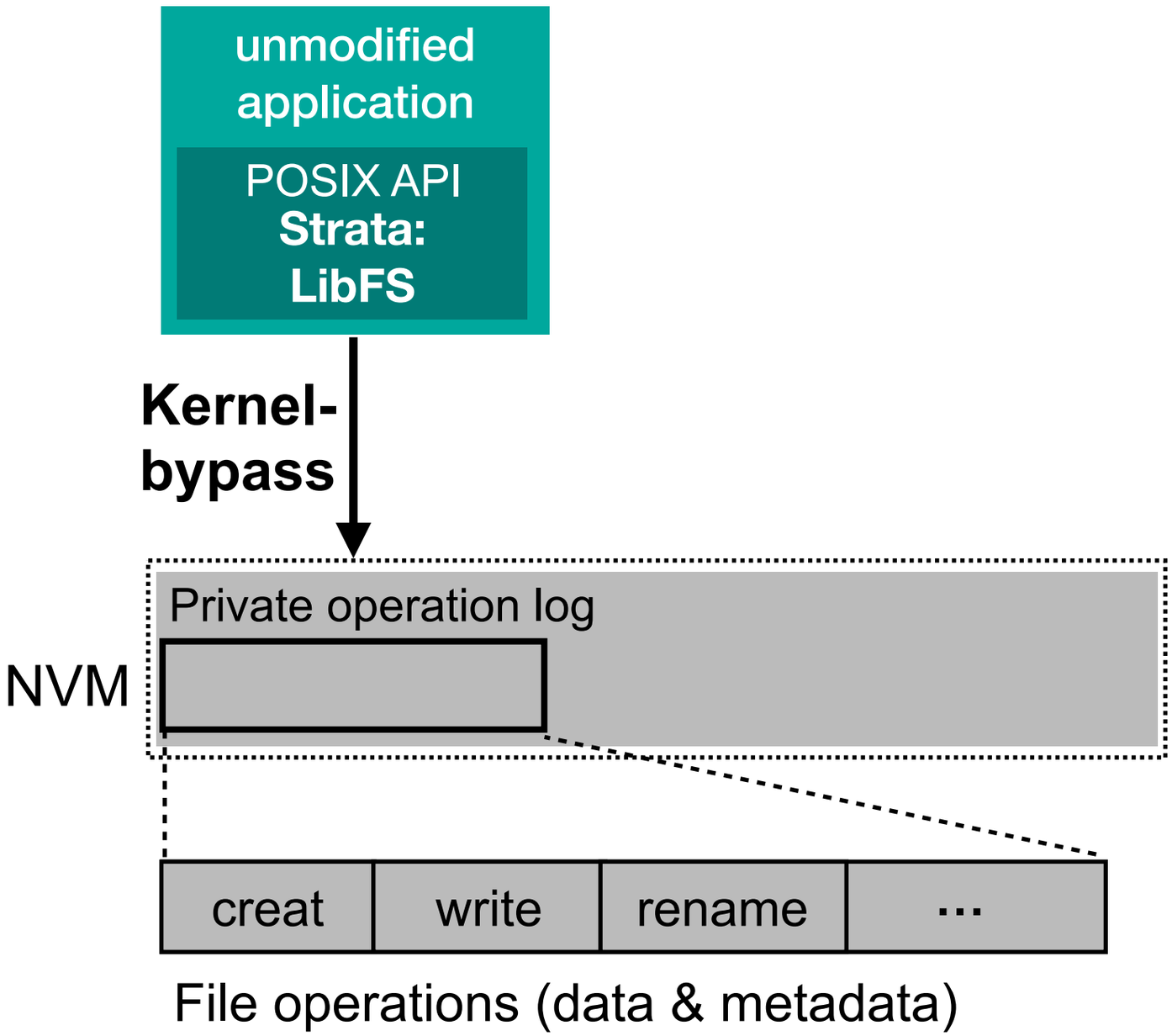
Outline

- **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

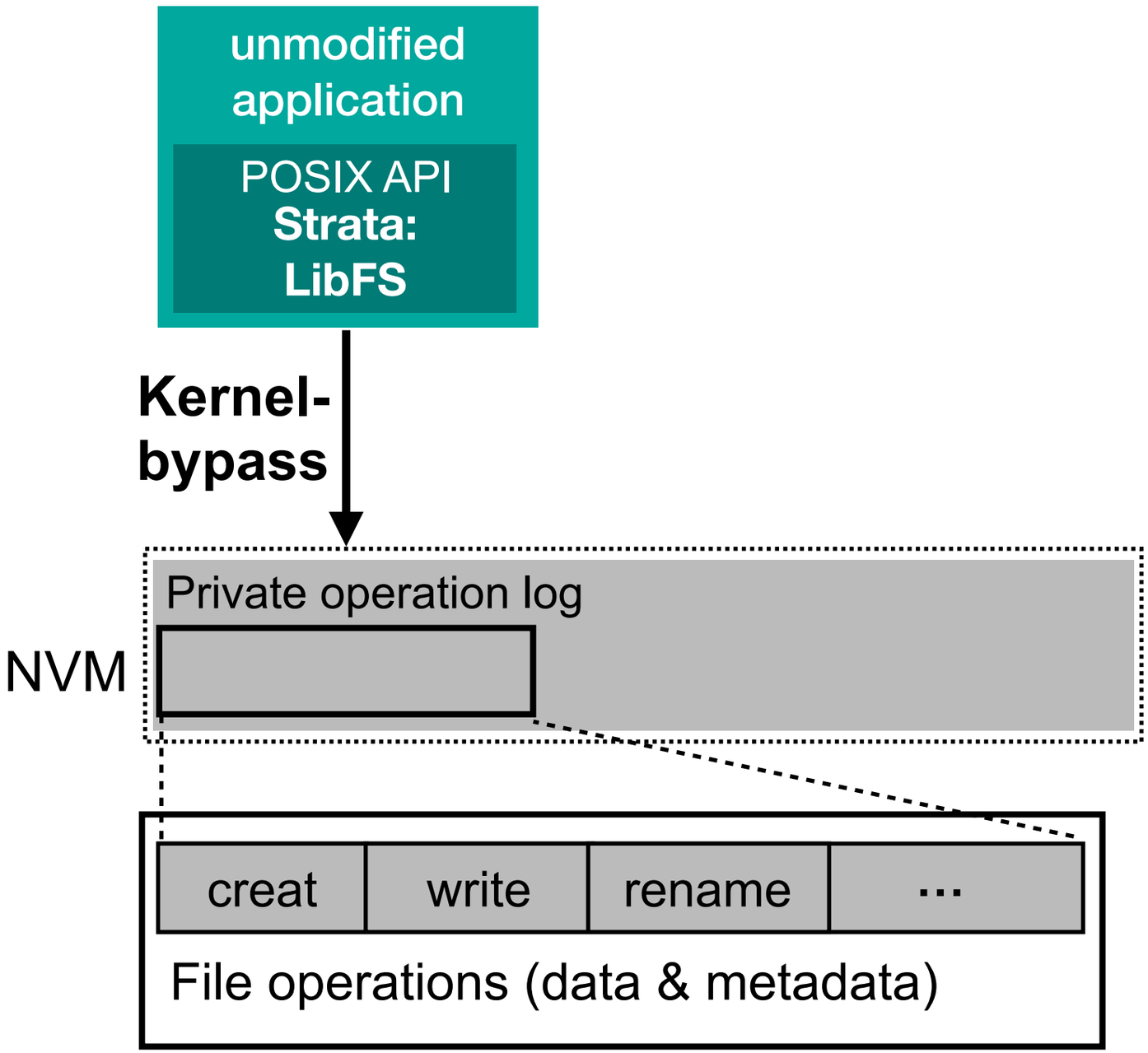


Log operations to NVM at user-level



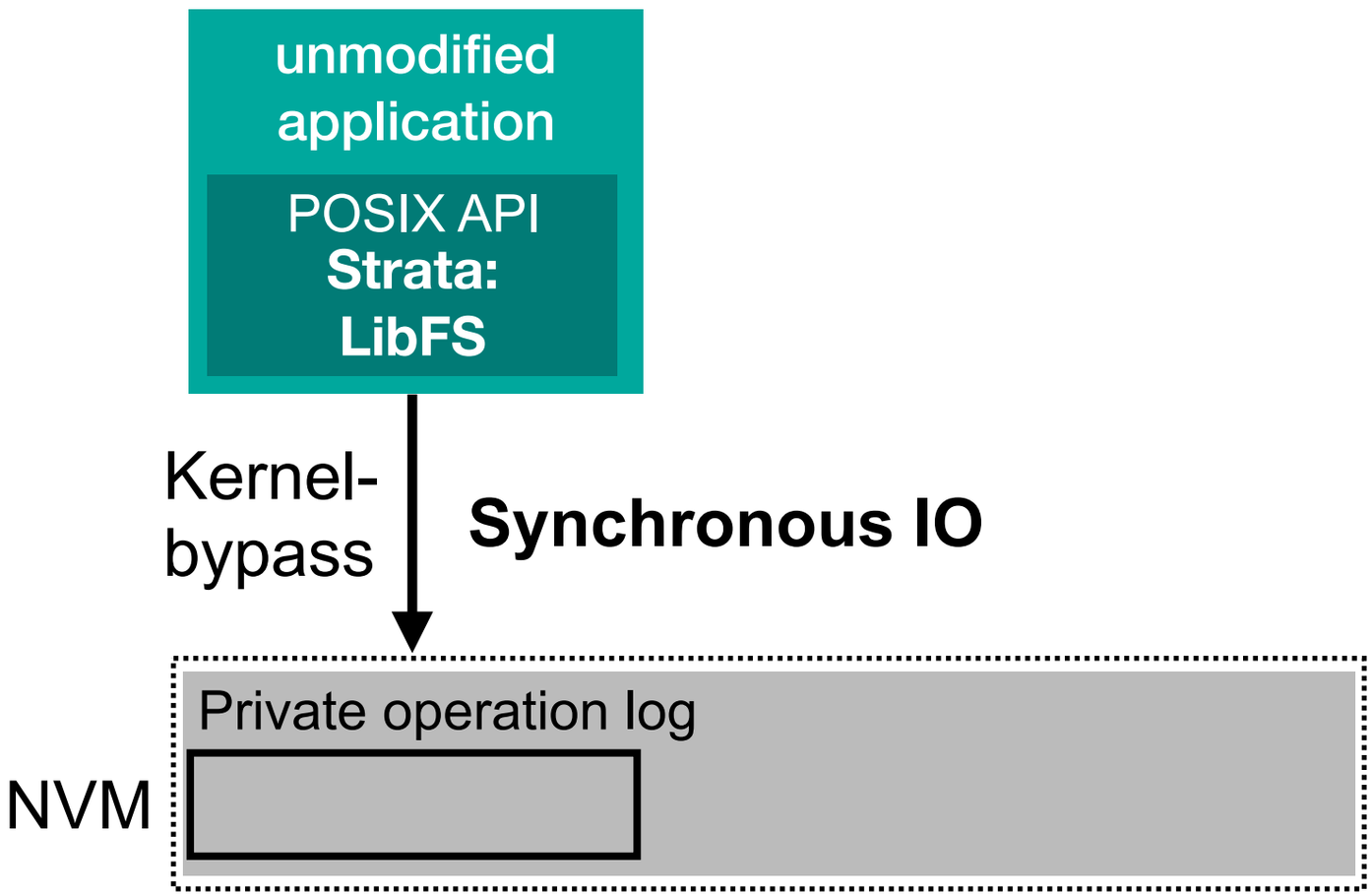
- Fast writes
  - Directly access fast NVM
  - Sequentially append data
  - Cache-line granularity
  - Blind writes

Log operations to NVM at user-level

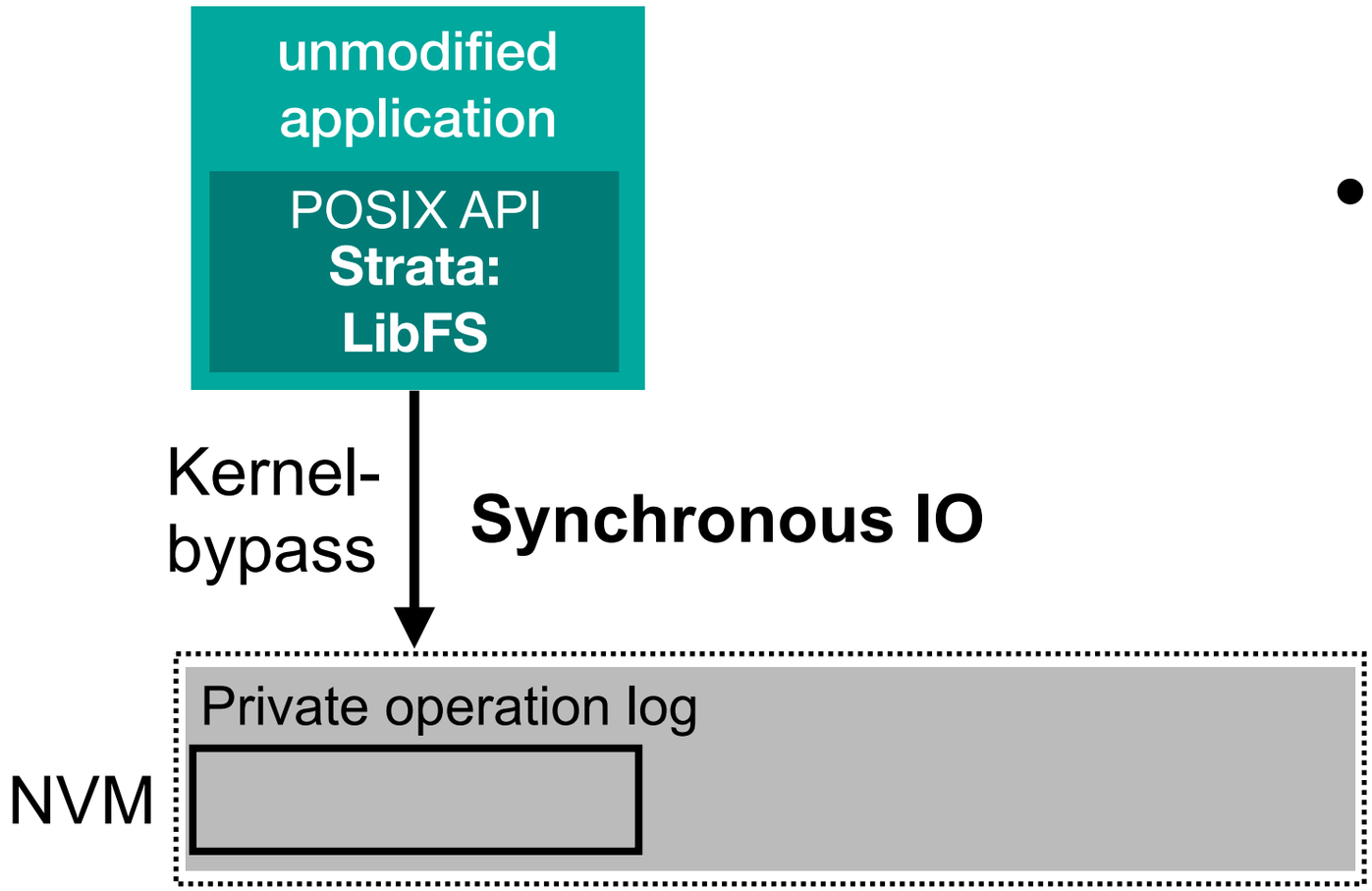


- Fast writes
  - Directly access fast NVM
  - Sequentially append data
  - Cache-line granularity
    - Blind writes
- Crash consistency
  - On crash, kernel replays log

Intuitive crash consistency

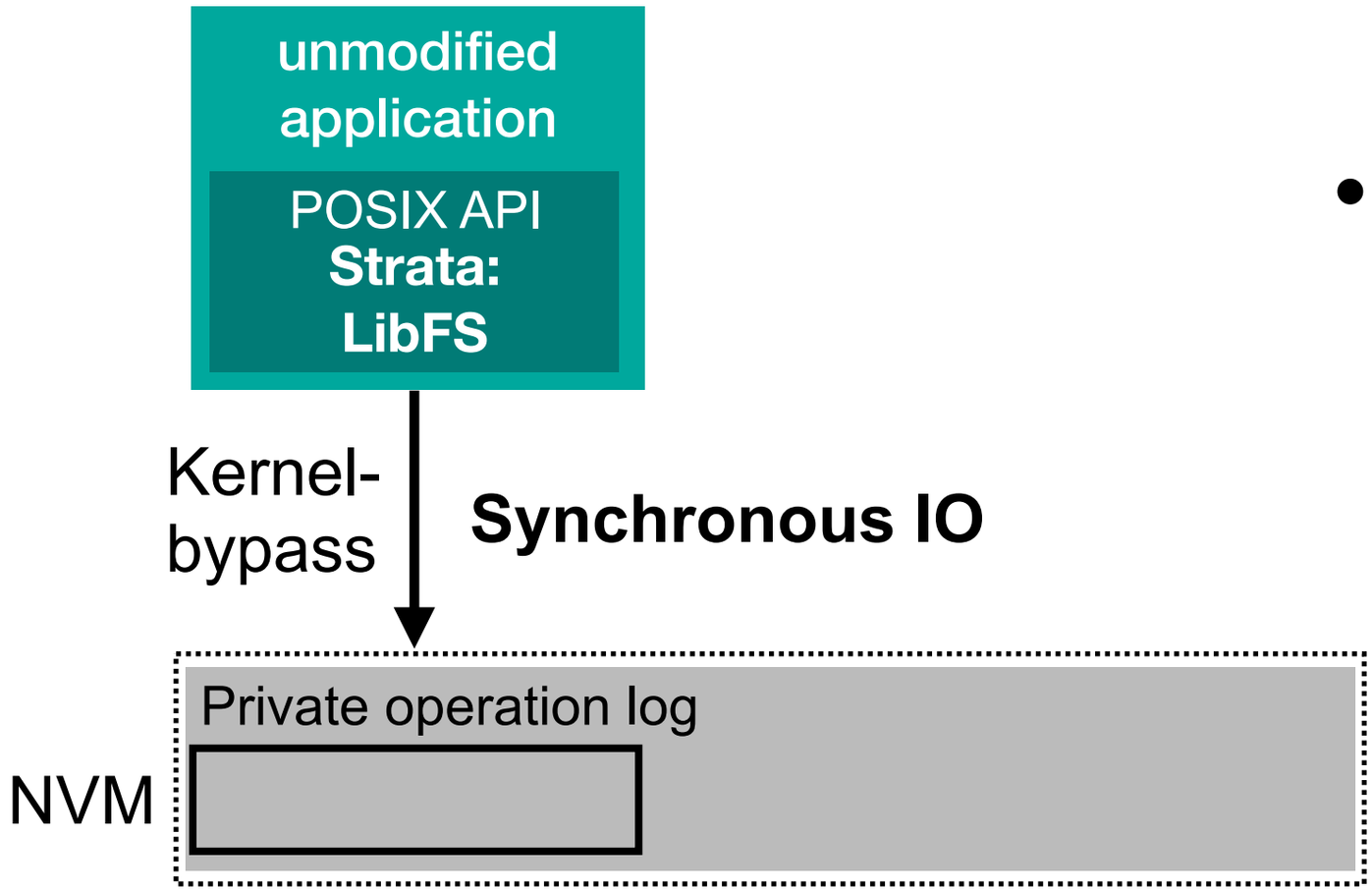


Intuitive crash consistency



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

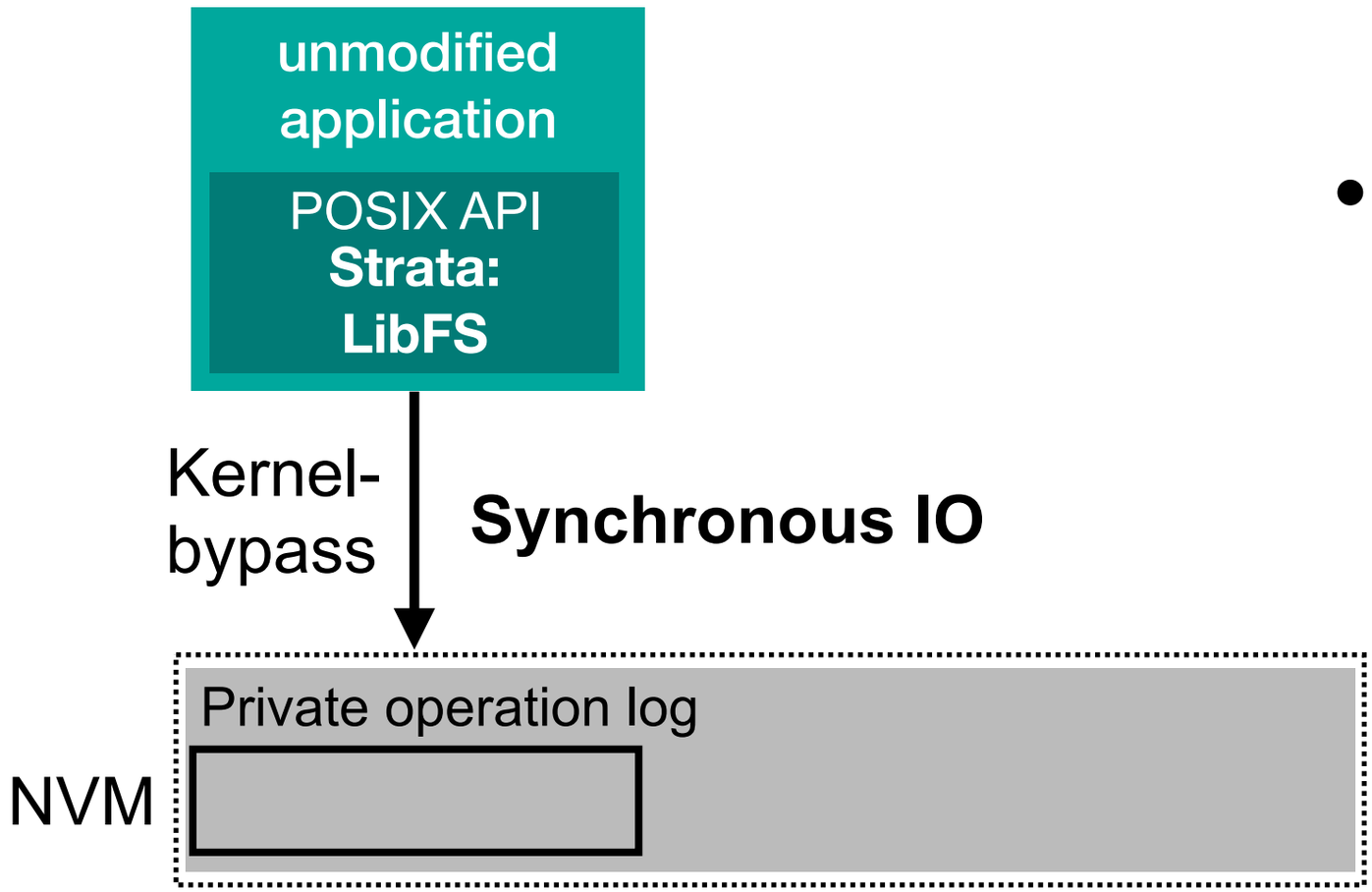
Intuitive crash consistency



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

fsync() is no-op

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

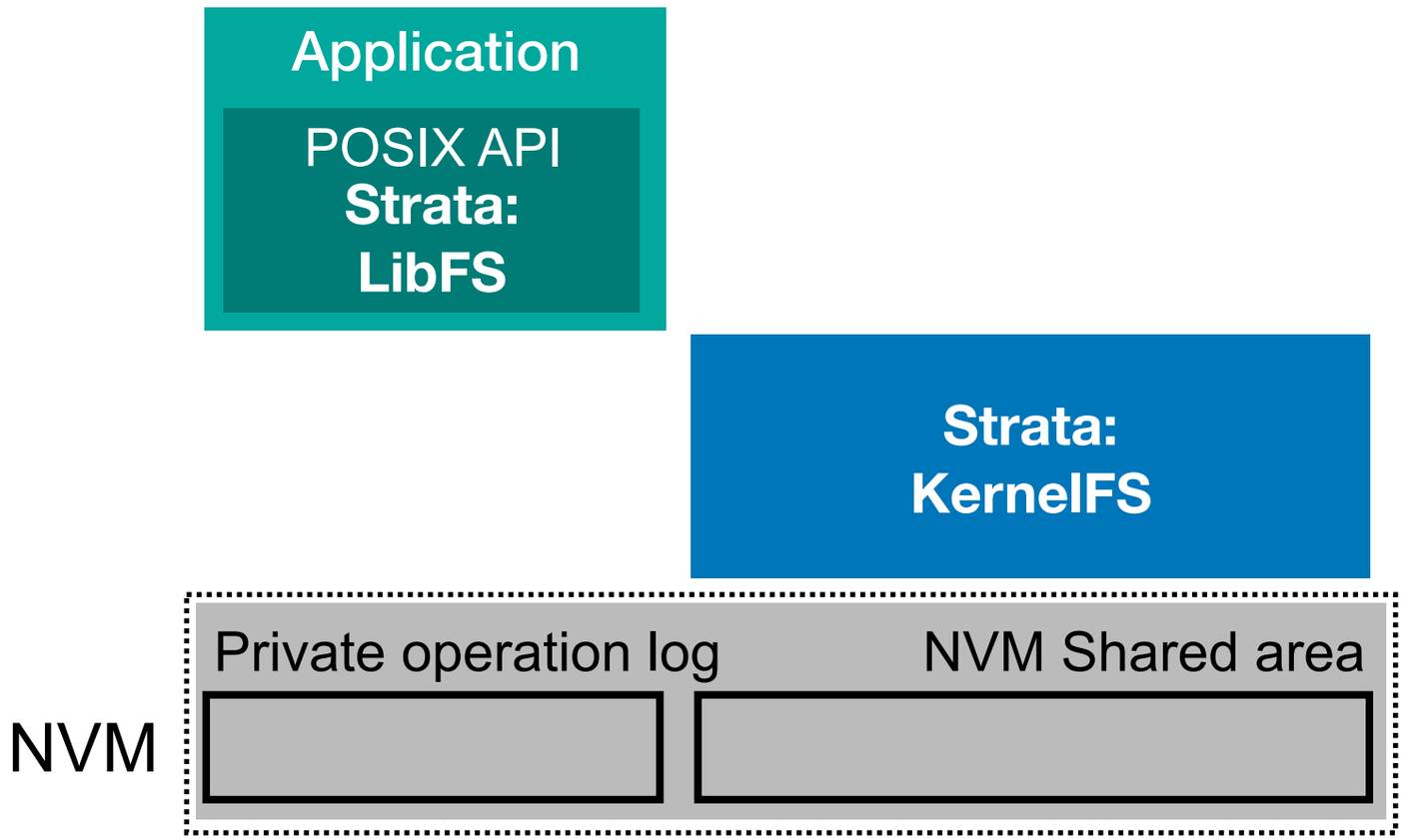
Outline

- **LibFS**: Log operations to NVM at user-level
  - Fast user-level access
  - In-order, synchronous IO

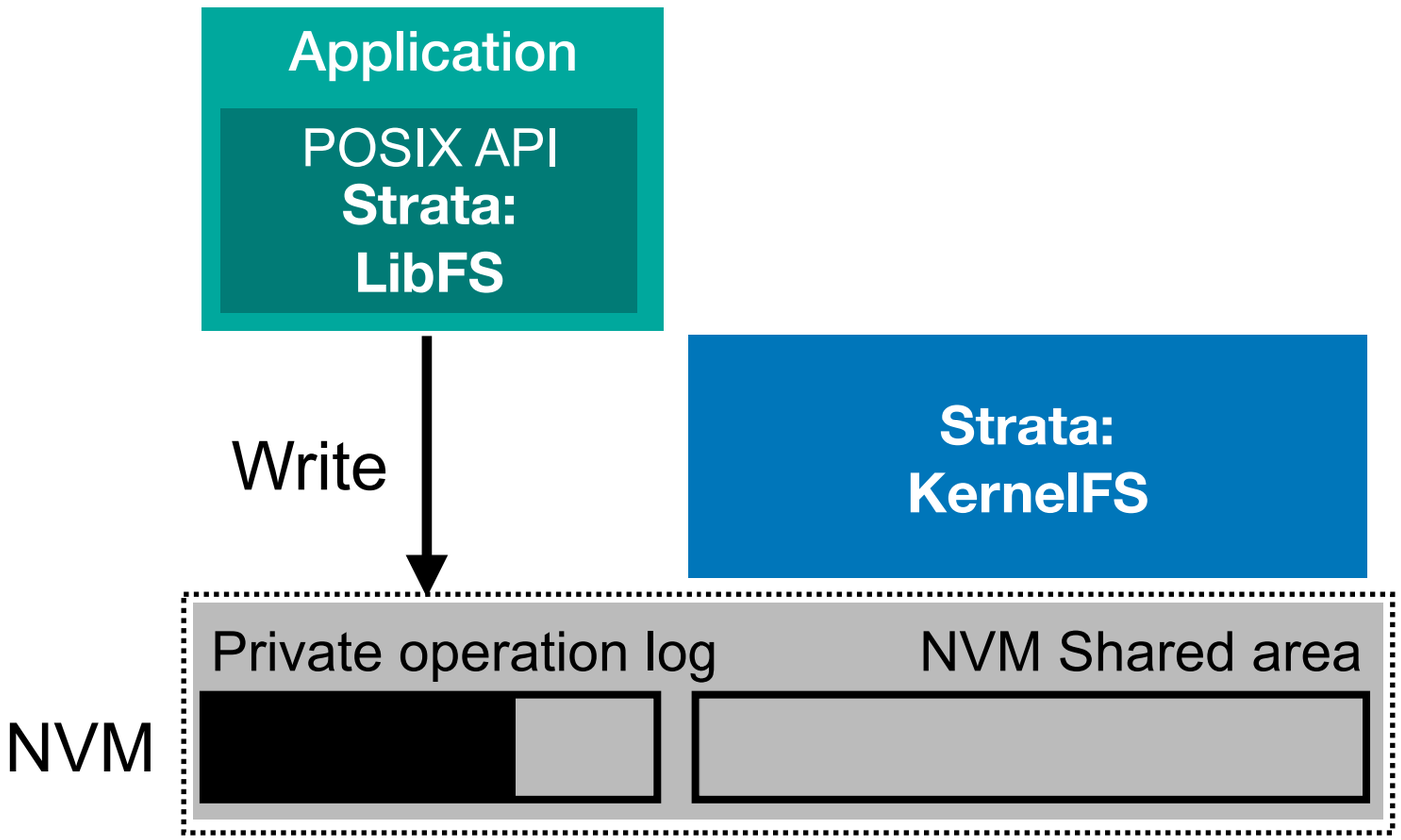
- **KernelFS**: Digest and migrate data in kernel
  - Asynchronous digest
  - Transparent data migration
  - Shared file access

- **Evaluation**

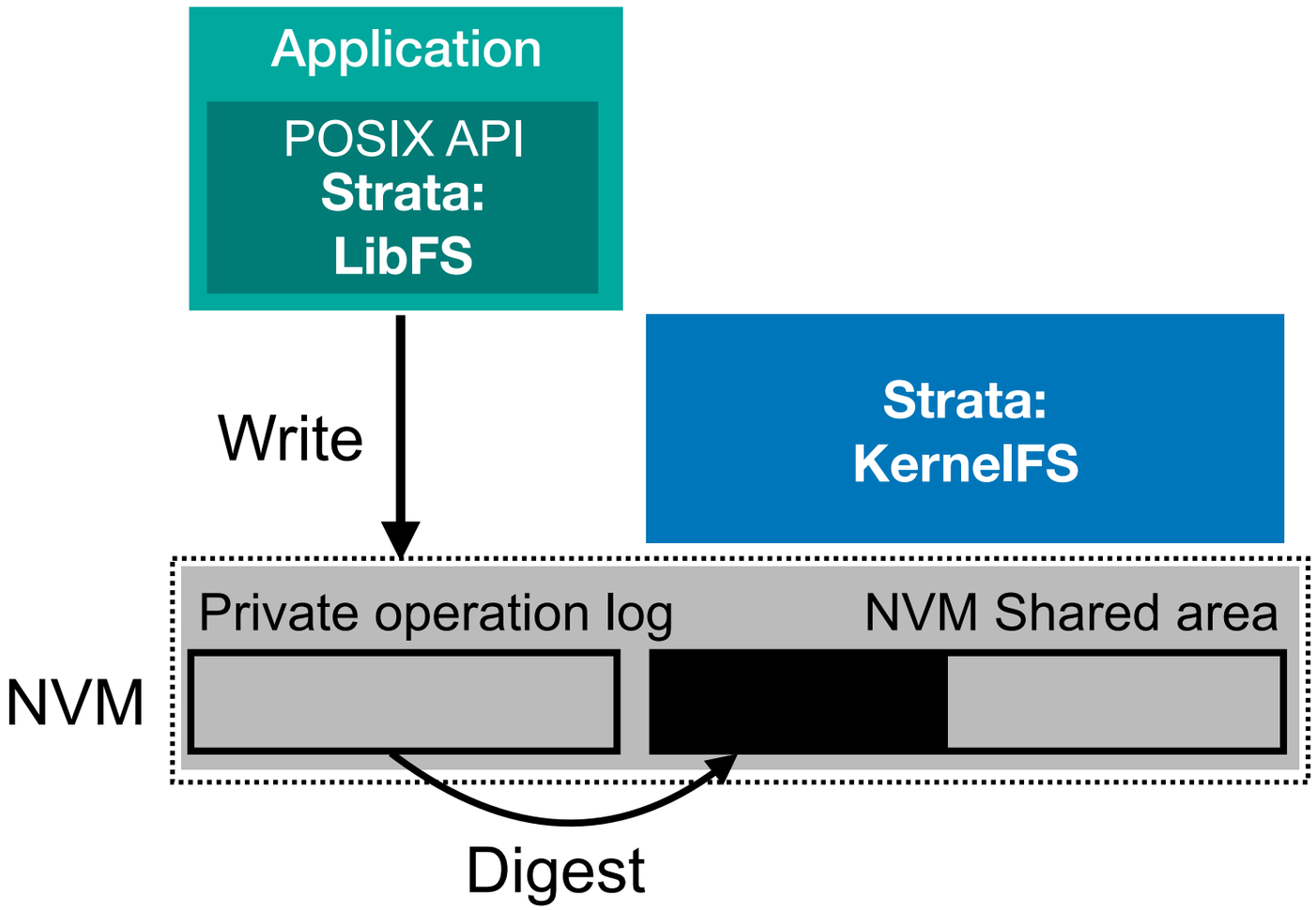
Digest data in kernel



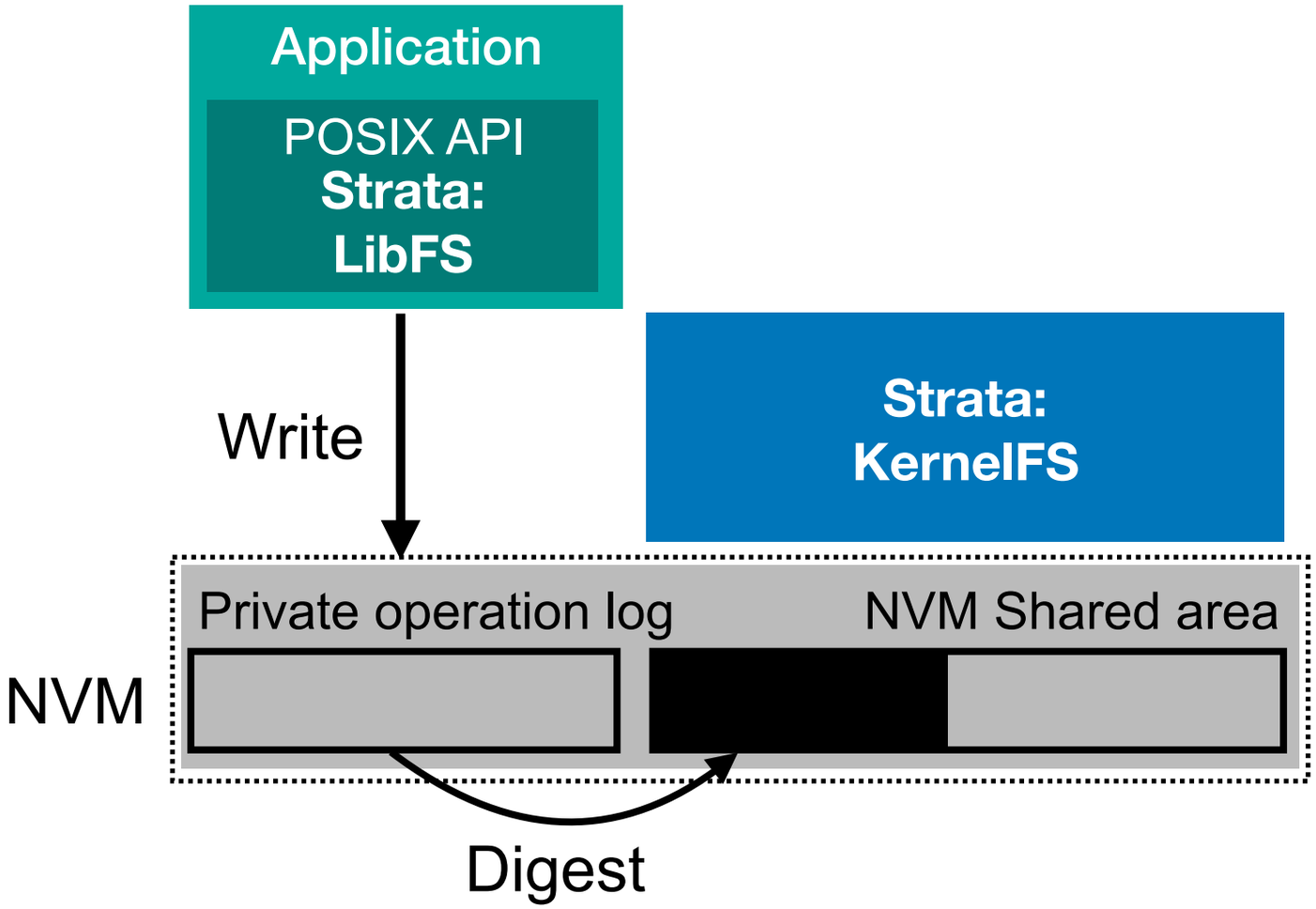
Digest data in kernel



Digest data in kernel



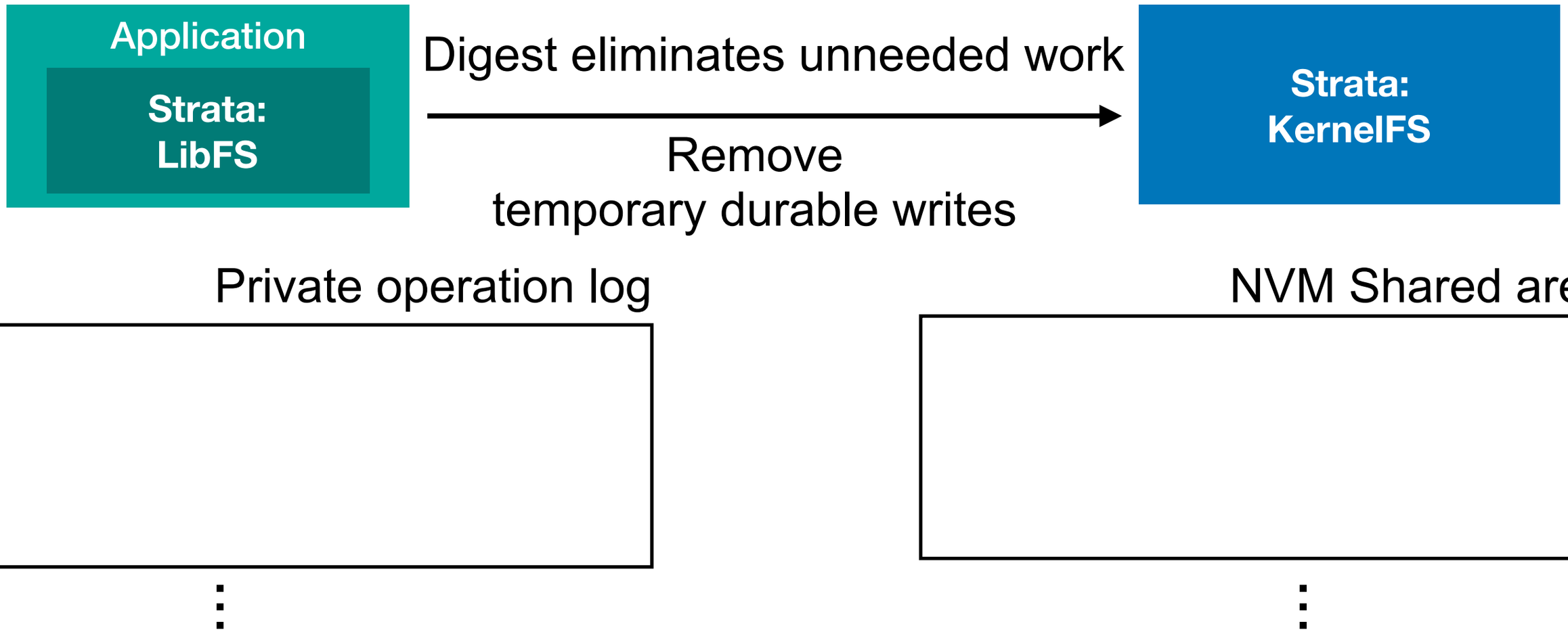
Digest data in kernel



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

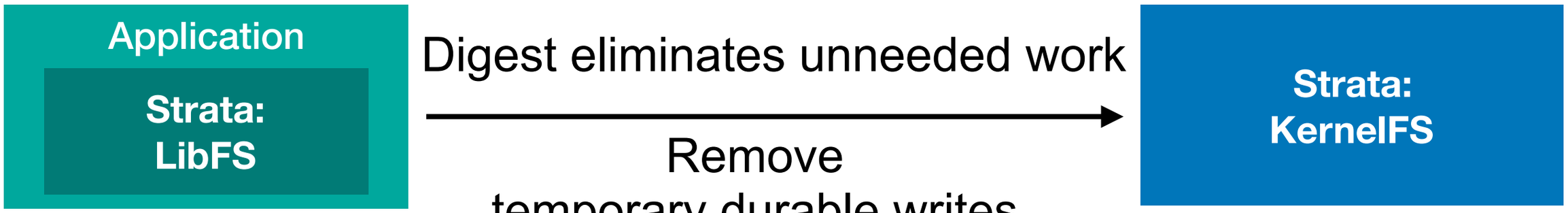
Digest optimization: Log coalescing

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



Digest optimization: Log coalescing

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



Private operation log

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

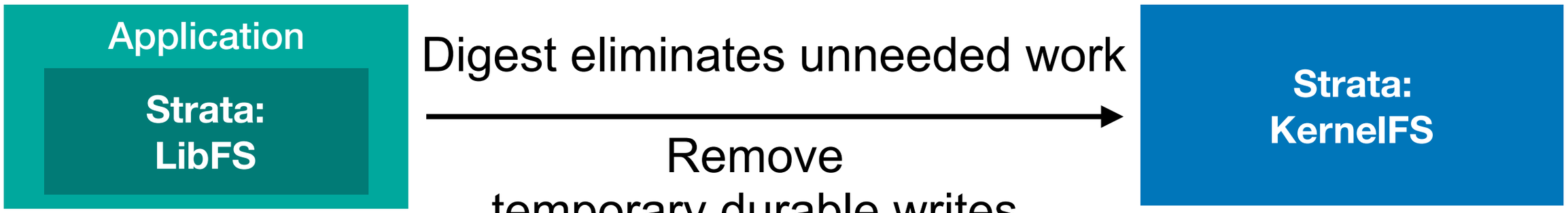
⋮

NVM Shared area

⋮

Digest optimization: Log coalescing

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



Private operation log

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

⋮

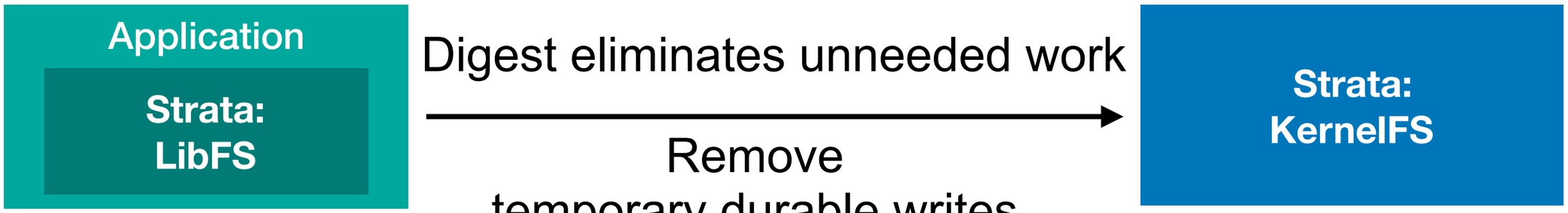
NVM Shared area

```
Write data to database file
```

⋮

Digest optimization: Log coalescing

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



Private operation log

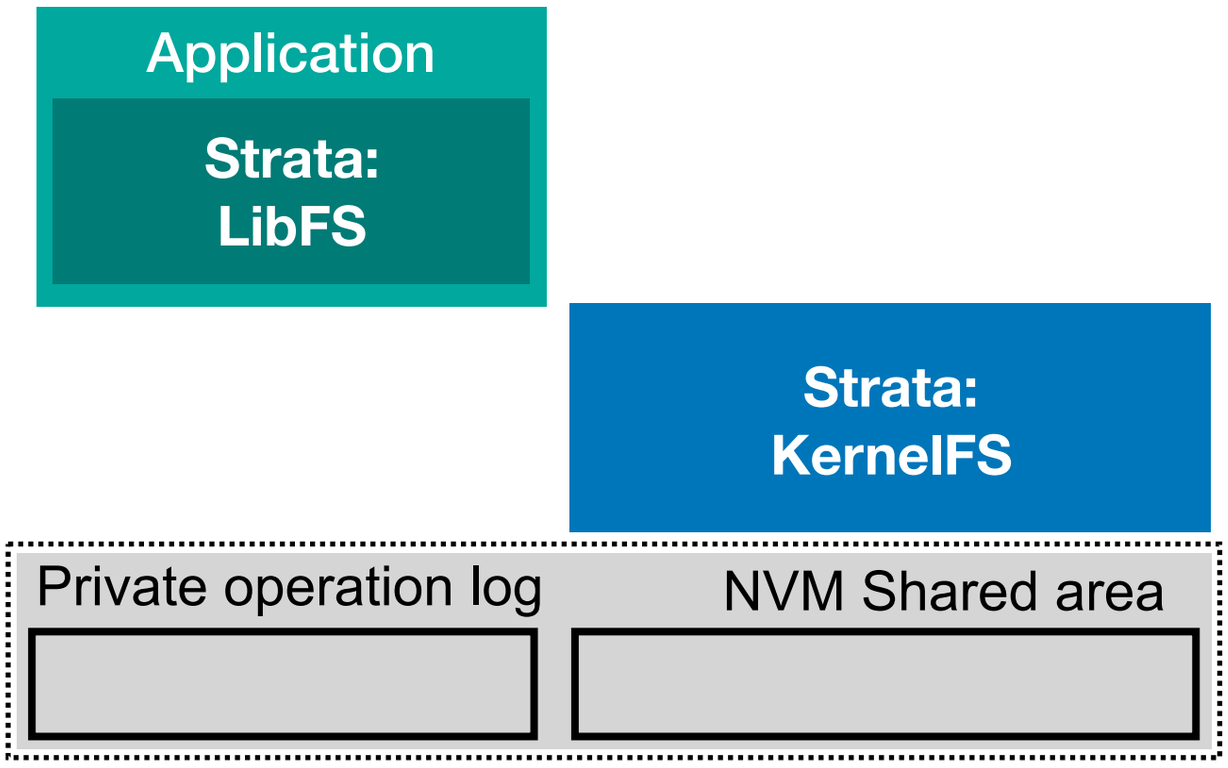
```
Create journal file
Write data to journal file
Write data to database file
Delete journal file
```

NVM Shared area

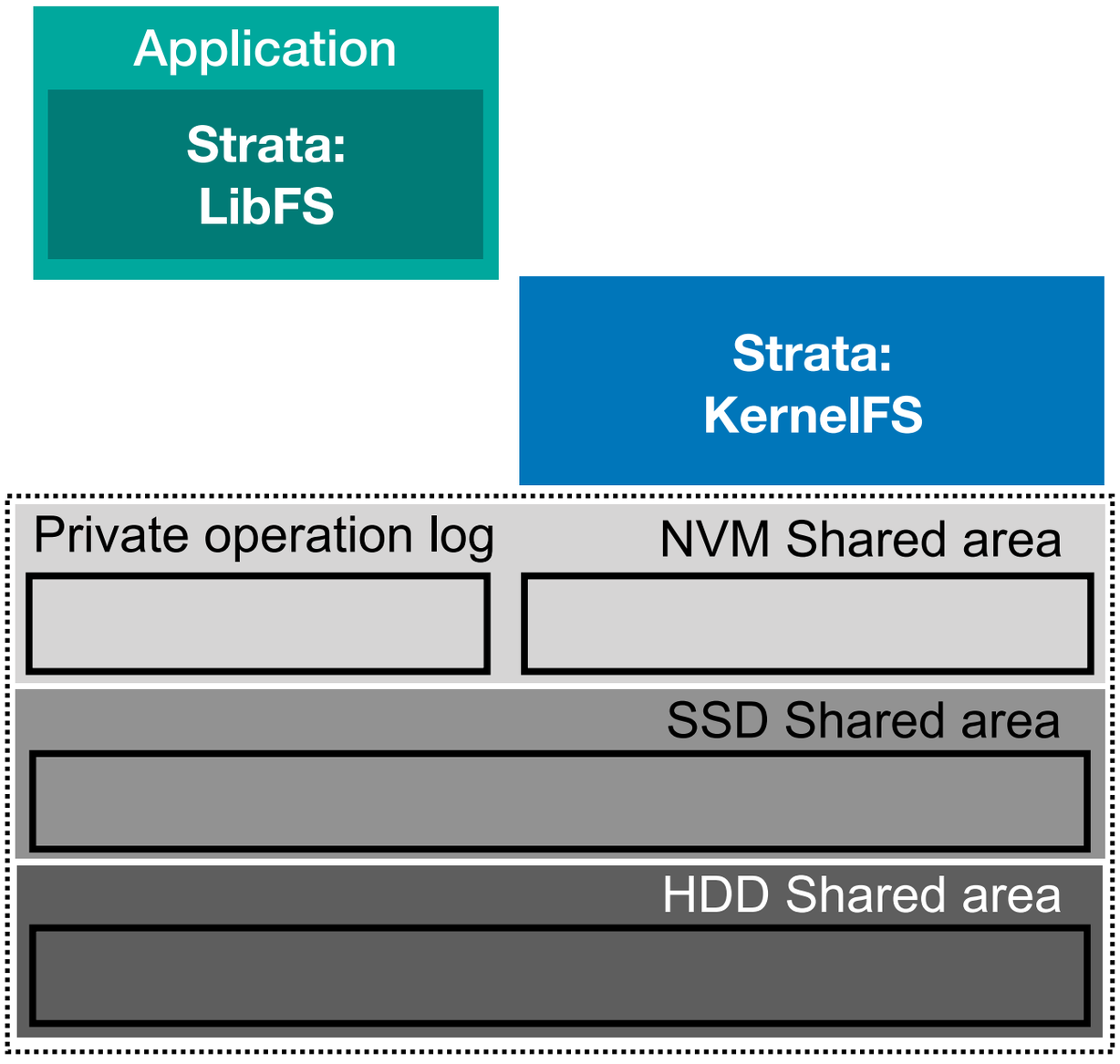
```
Write data to database file
```

Throughput optimization: Log coalescing saves IO while digesting

Digest and migrate data in kernel

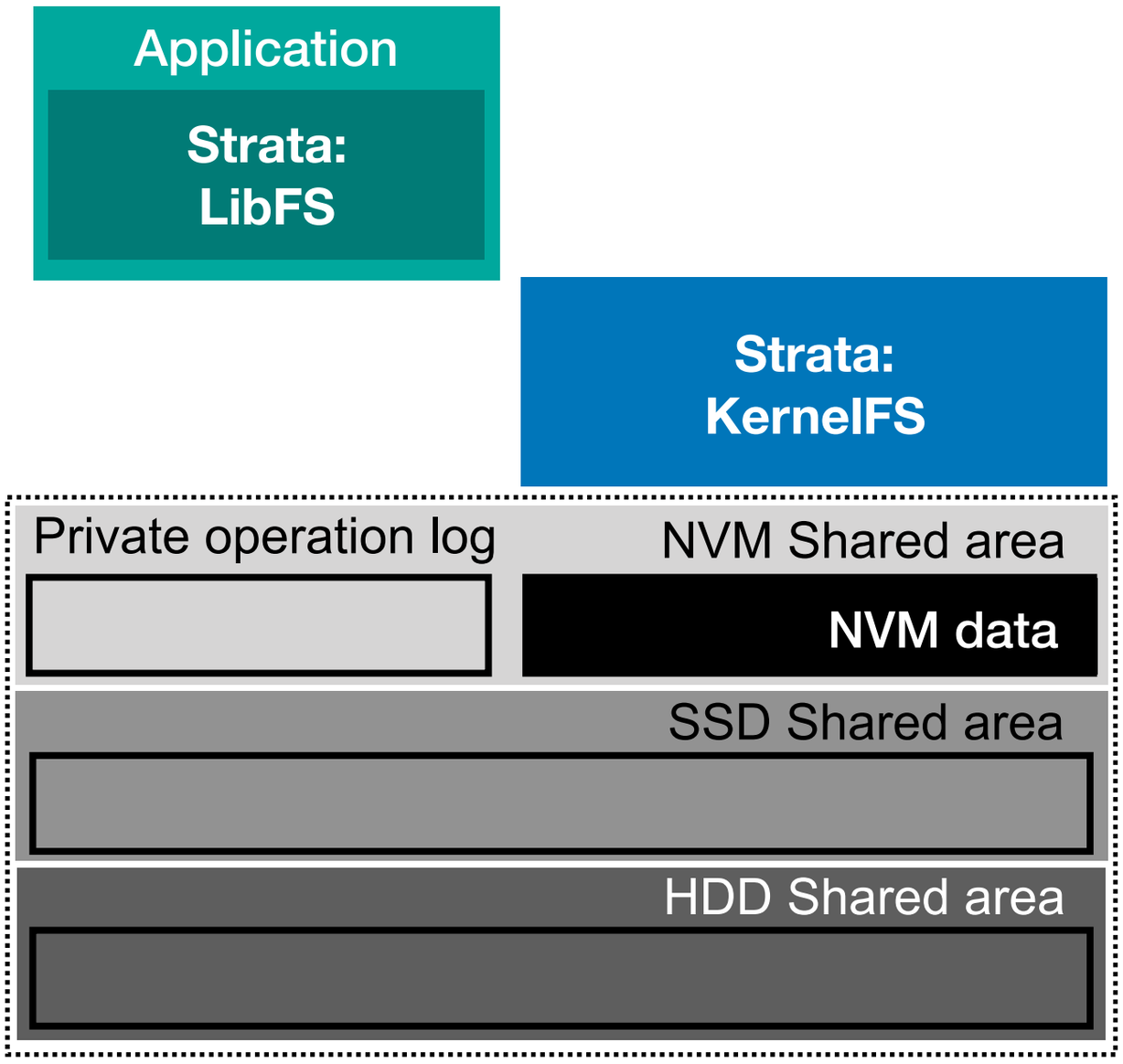


Digest and migrate data in kernel



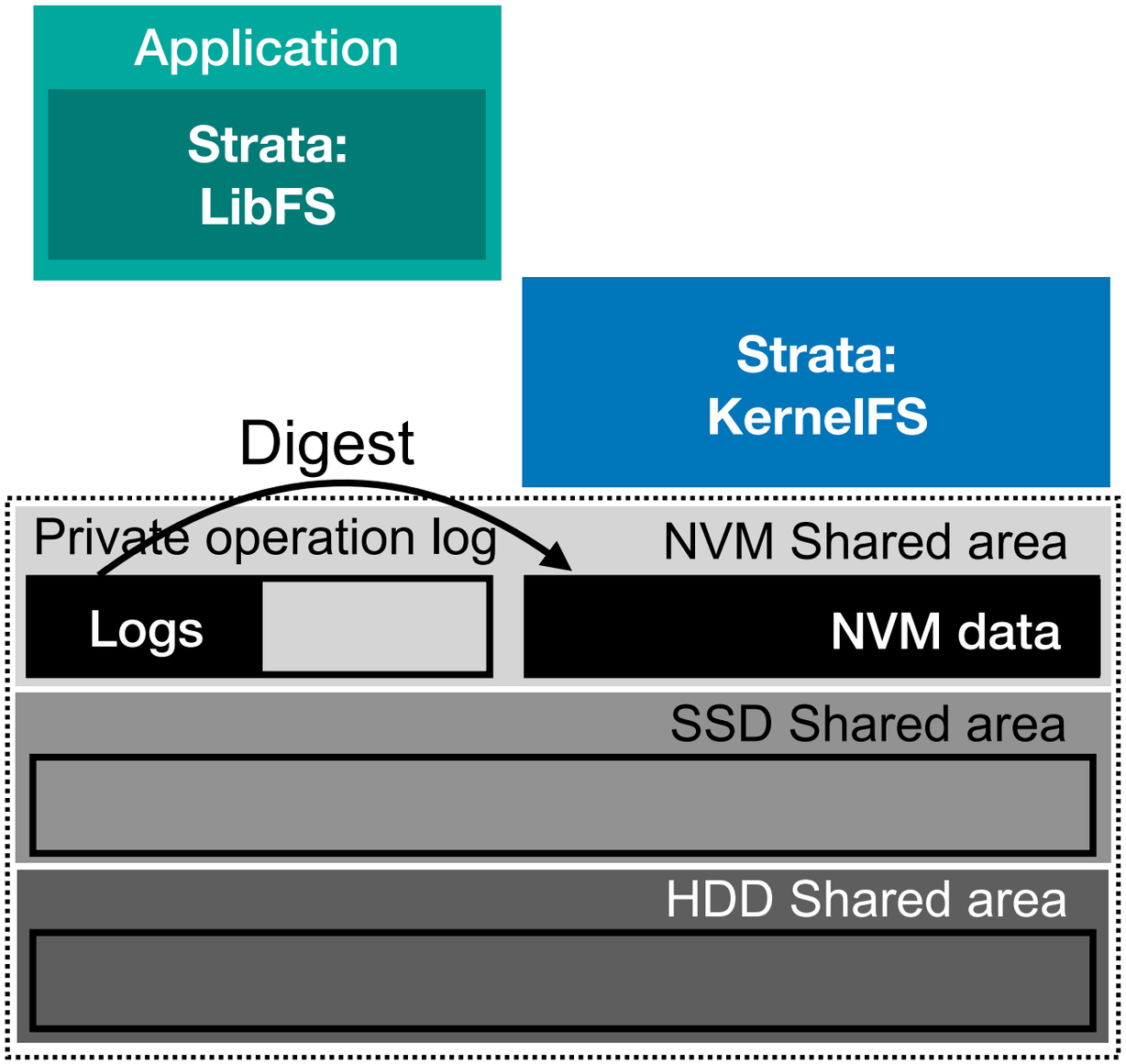
- Low-cost capacity
  - KernelFS migrates cold data to lower layers

Digest and migrate data in kernel



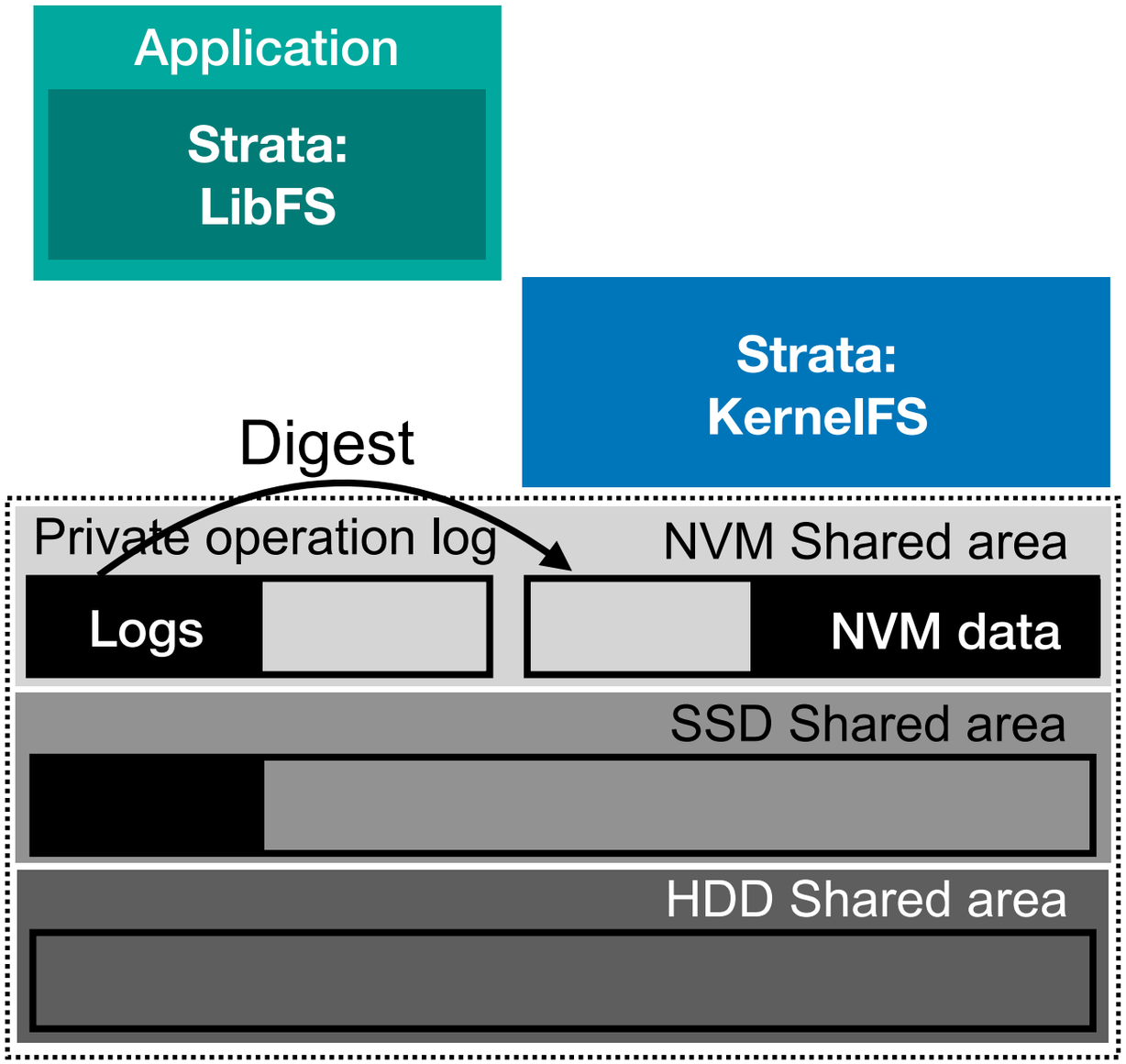
- Low-cost capacity
  - KernelFS migrates cold data to lower layers

Digest and migrate data in kernel



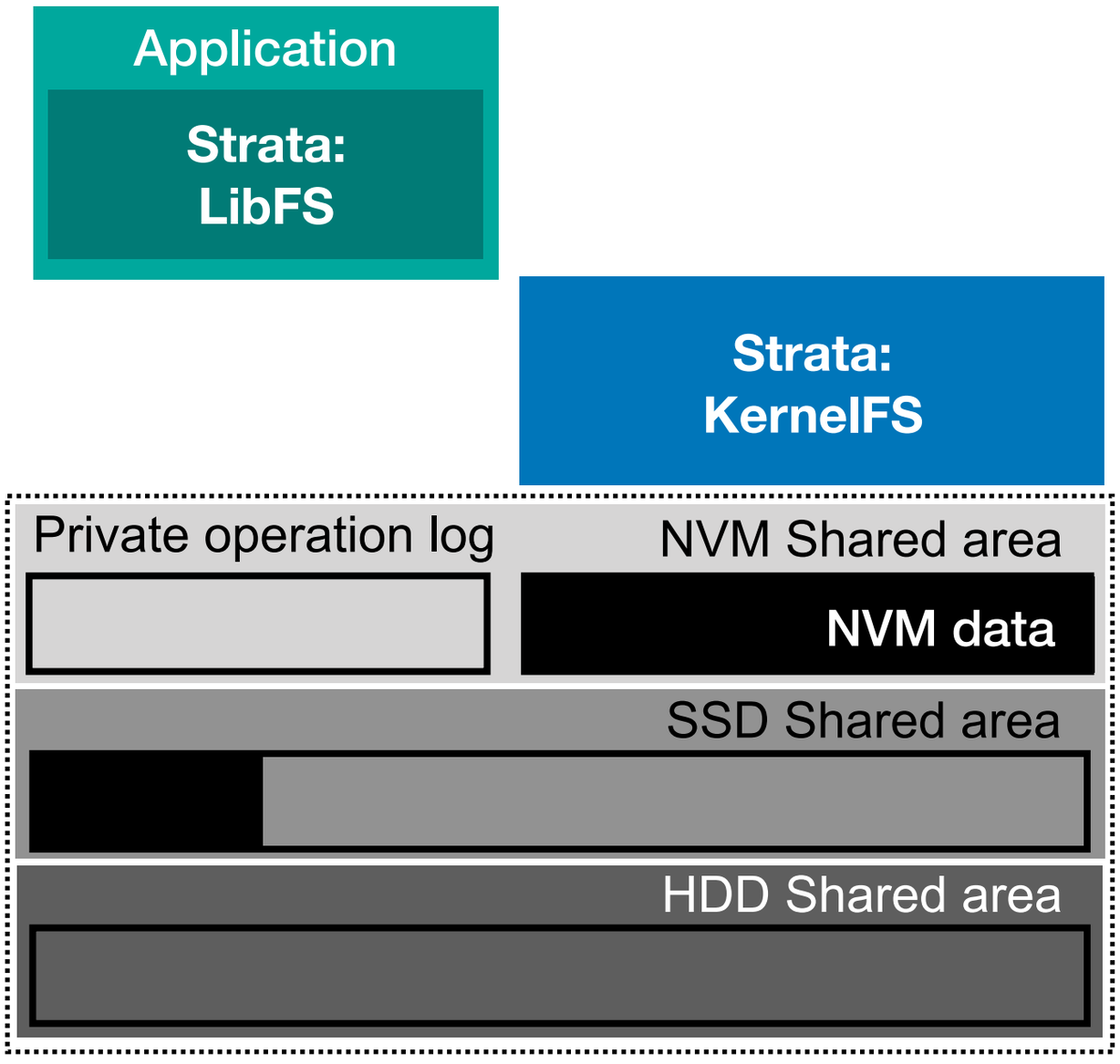
- Low-cost capacity
  - KernelFS migrates cold data to lower layers

Digest and migrate data in kernel



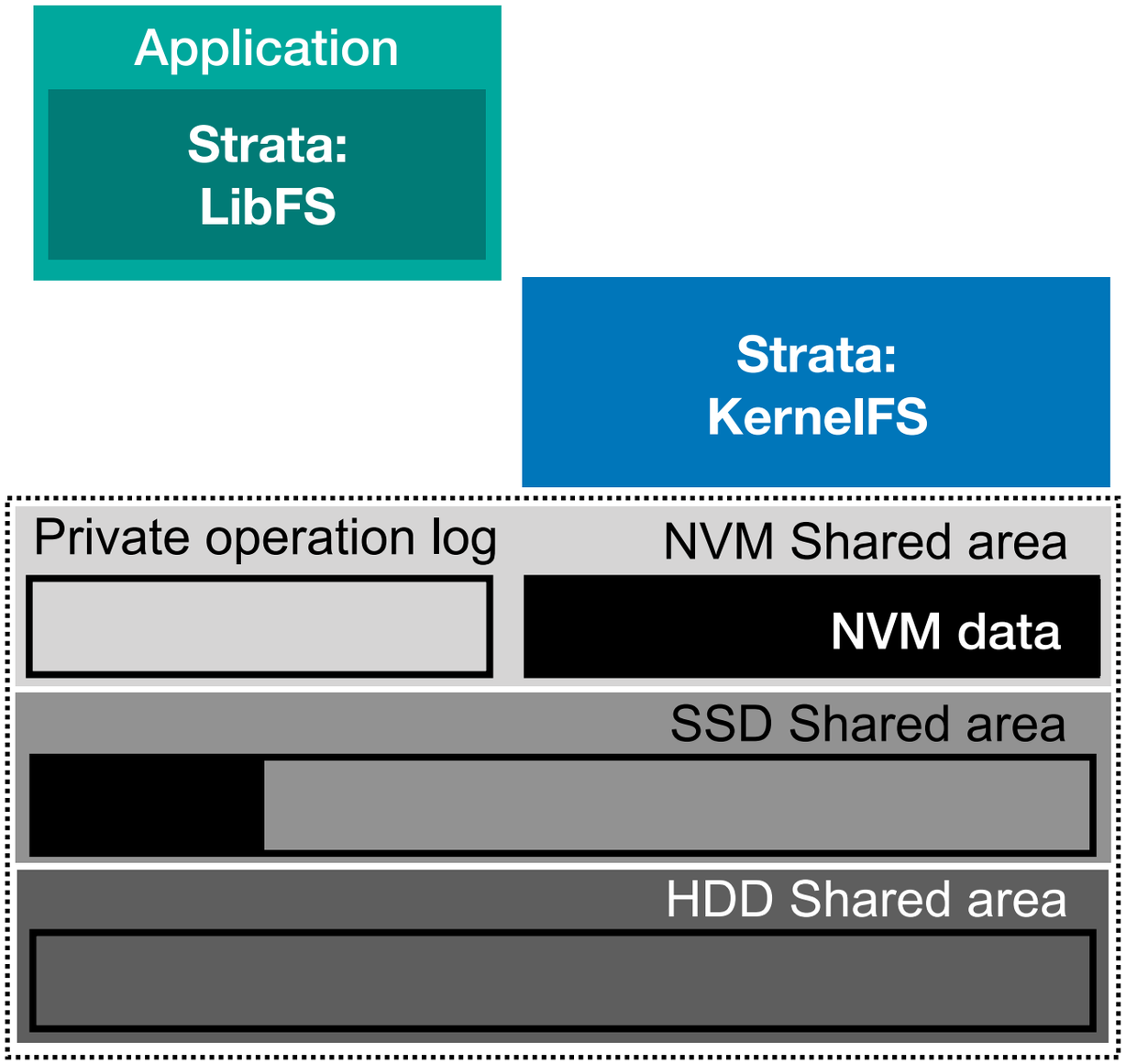
- Low-cost capacity
  - KernelFS migrates cold data to lower layers

Digest and migrate data in kernel



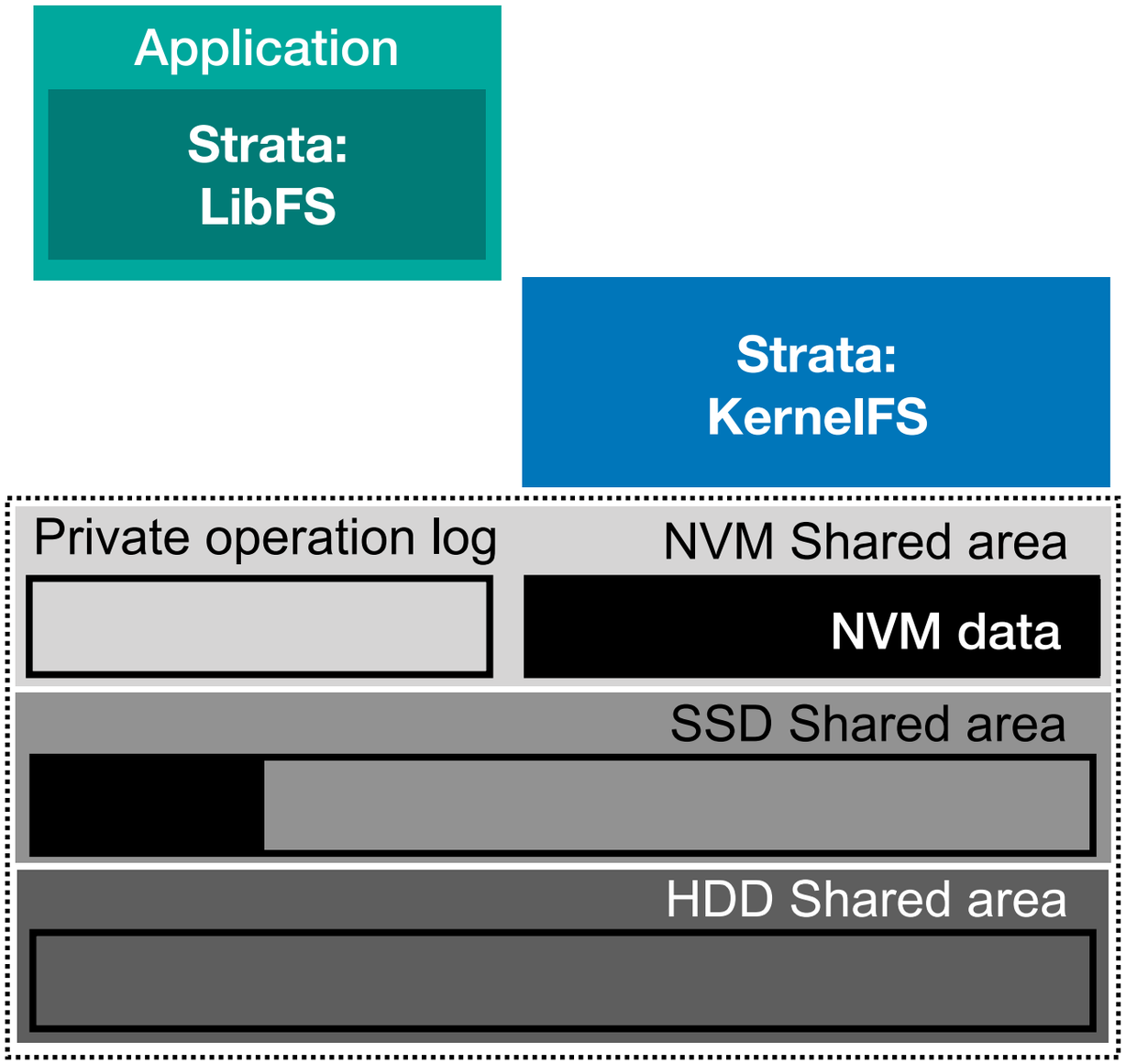
- Low-cost capacity
  - KernelFS migrates cold data to lower layers

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

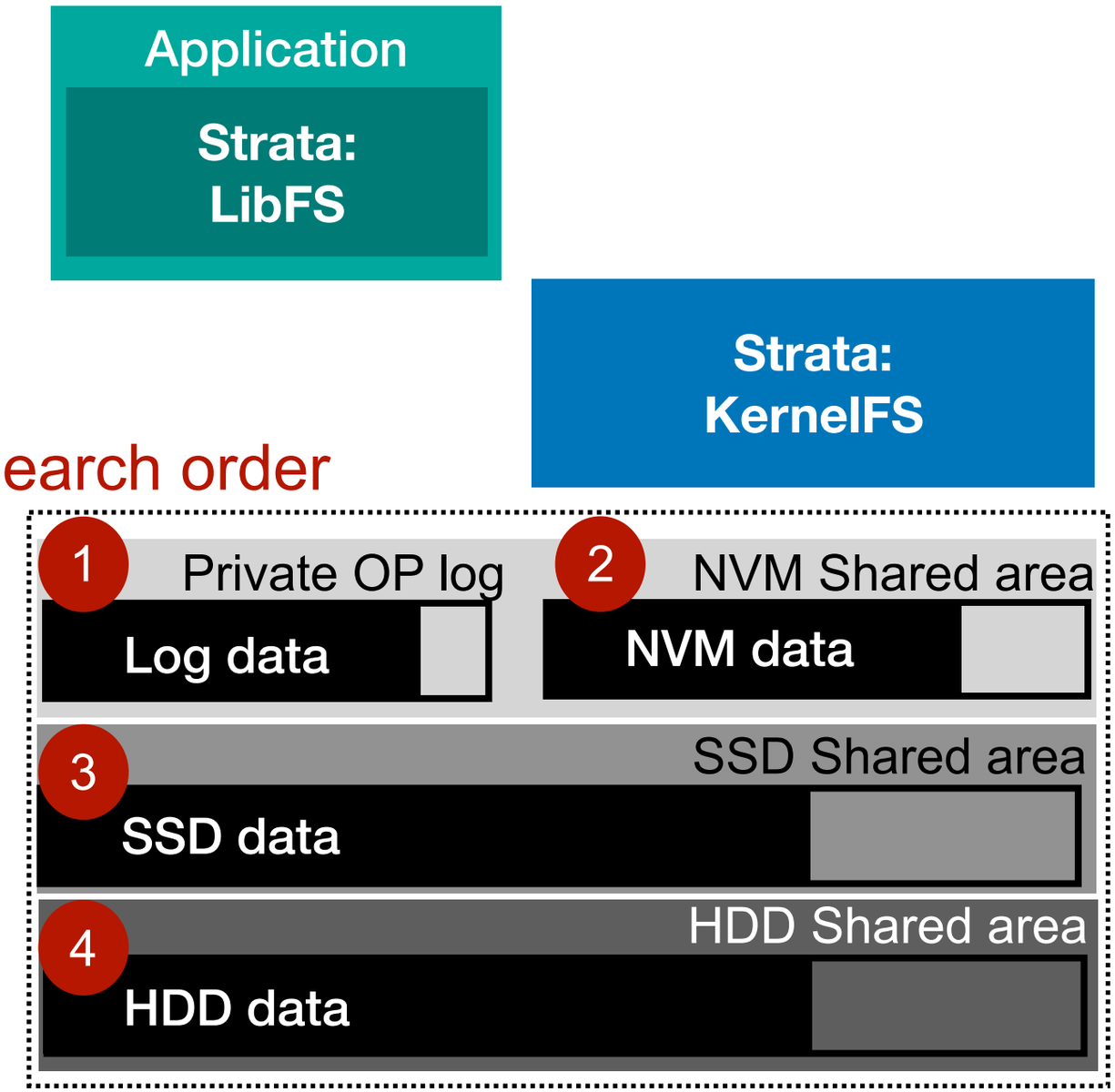
Digest and migrate data in kernel



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

Resembles log-structured merge (LSM) tree

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

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

- On revocation, LibFS digests leased data
  - Private data made public before losing lease

- **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?

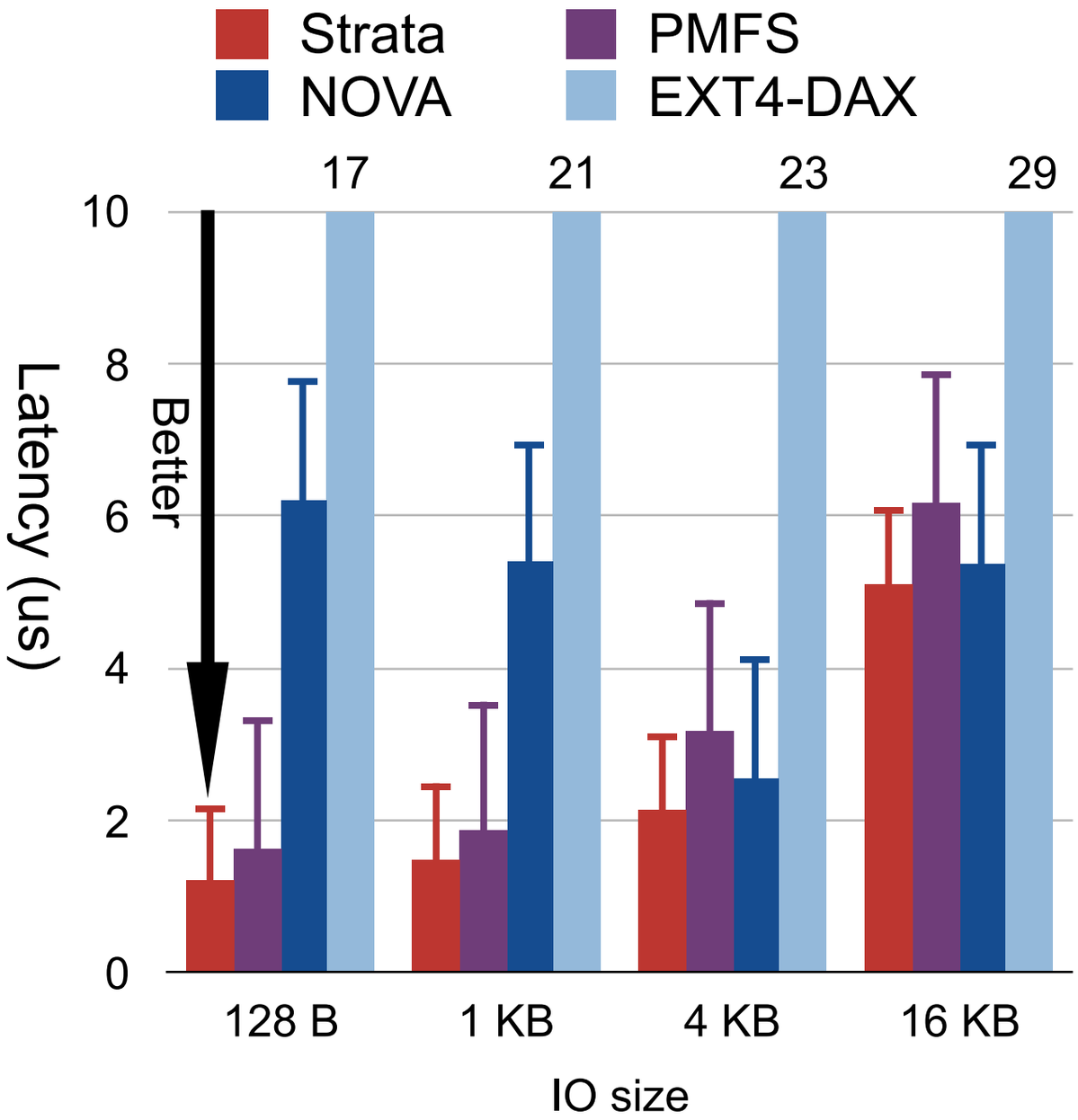
Related work

- **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

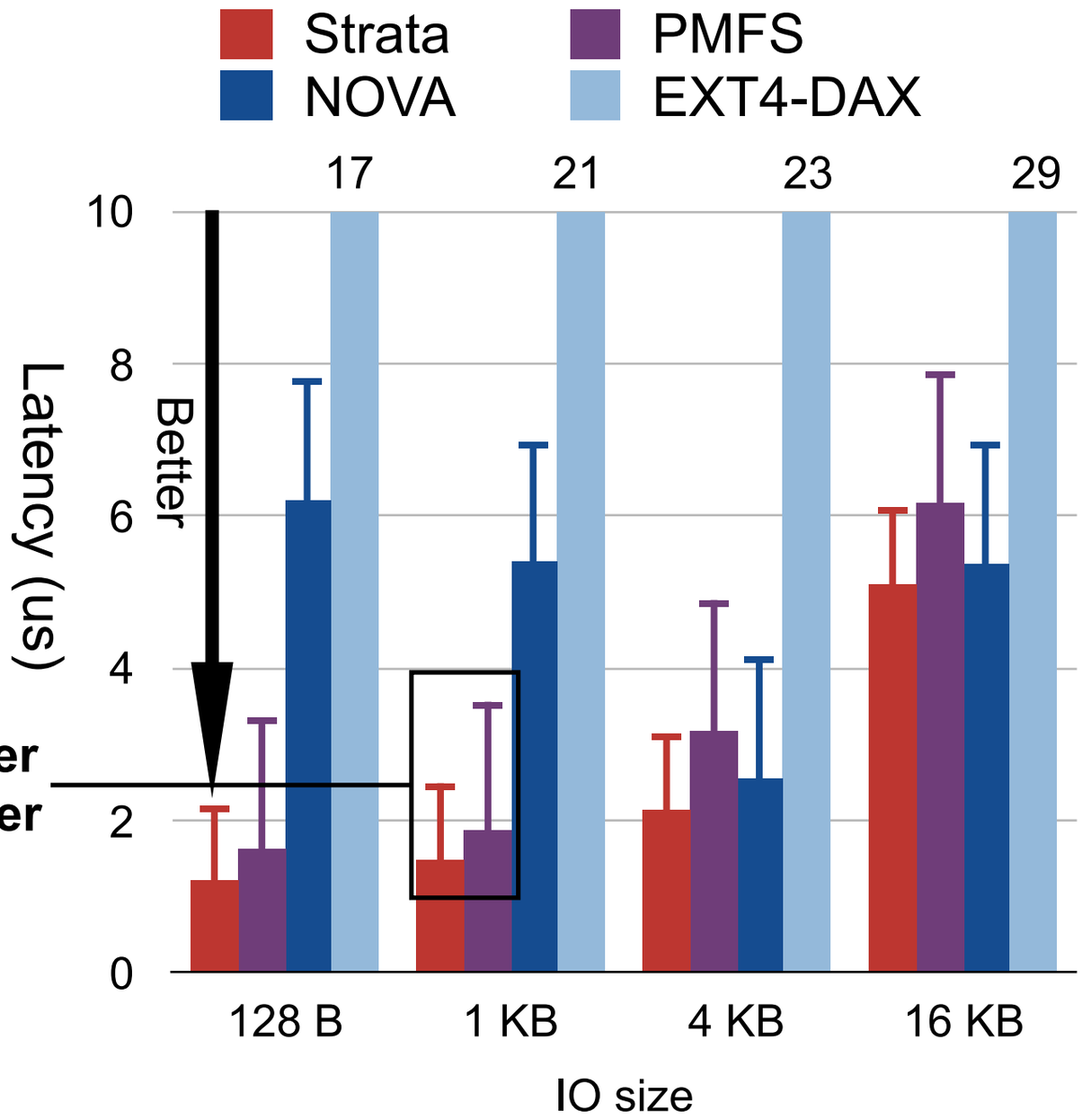
Microbenchmark: write latency

- 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



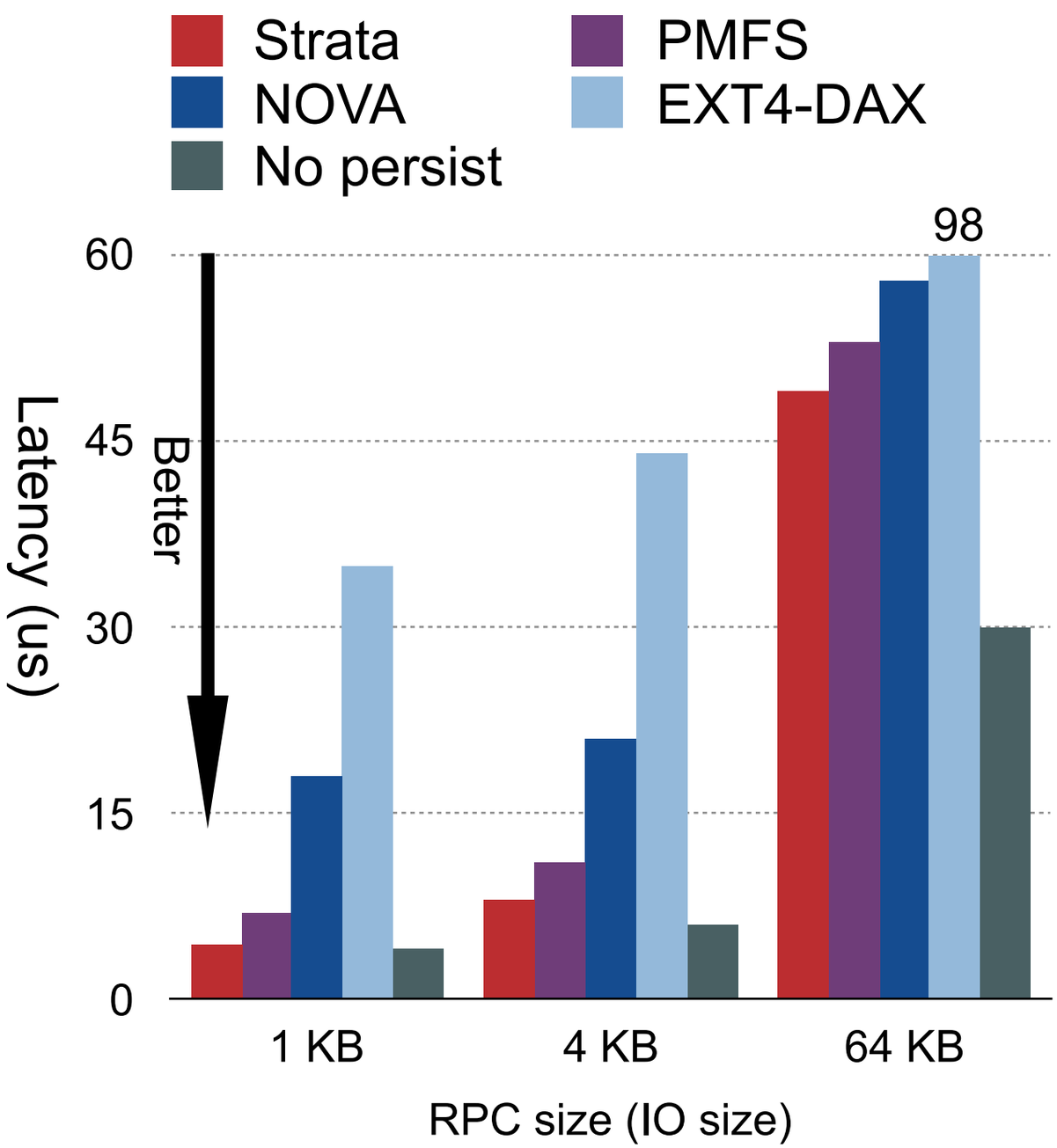
Microbenchmark: write latency

- 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



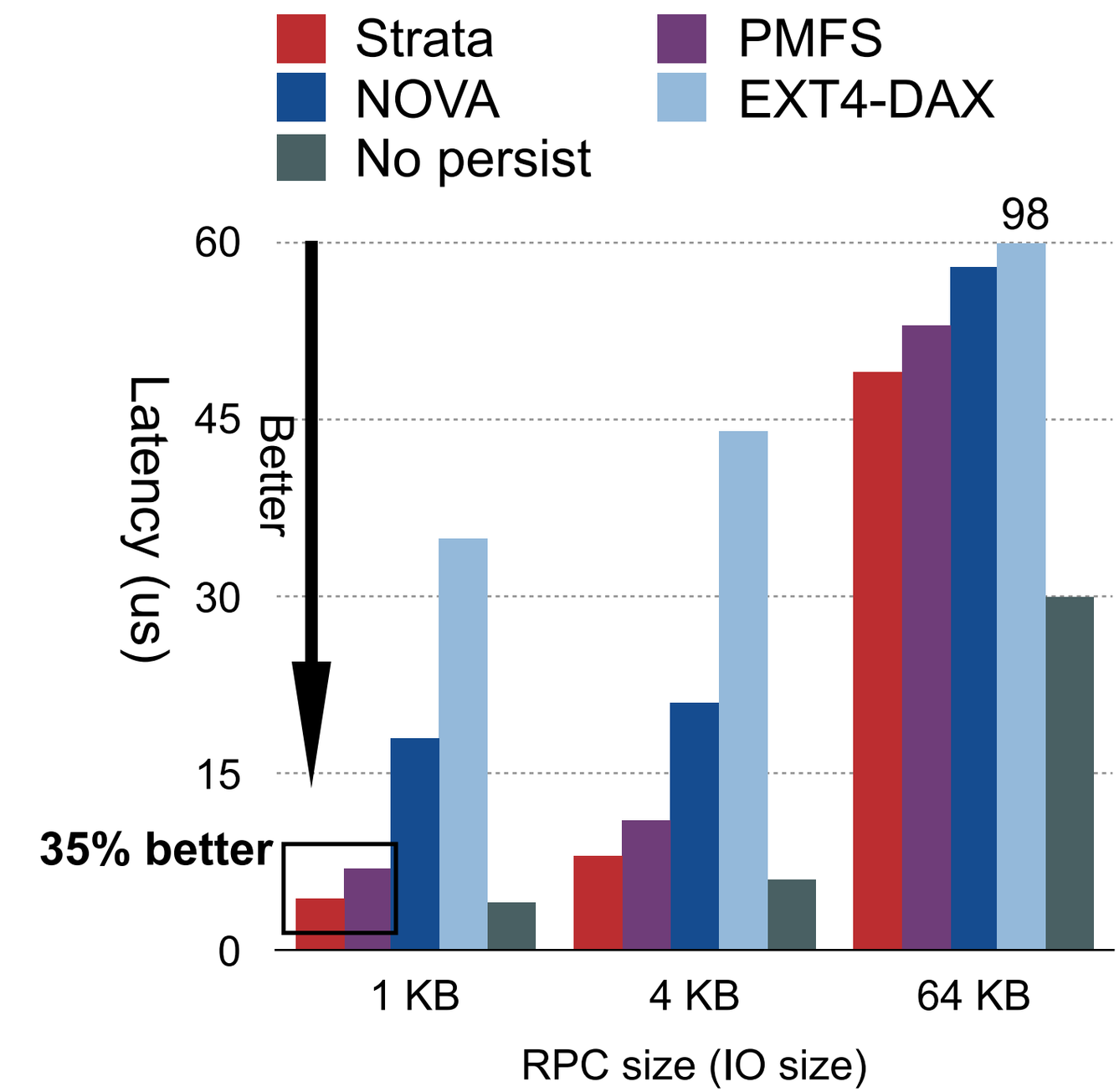
Latency: persistent RPC

- Foundation of most servers
  - Persist RPC data before sending ACK to client
- RPC over RDMA
  - 40 Gb/s Infiniband NIC
- For small IO (1 KB)
  - 25% slower than No persist
  - 35% faster than PMFS
    7x faster than EXT4-DAX



Latency: persistent RPC

- Foundation of most servers
  - Persist RPC data before sending ACK to client
- RPC over RDMA
  - 40 Gb/s Infiniband NIC
- For small IO (1 KB)
  - 25% slower than No persist
  - 35% faster than PMFS
    7x faster than EXT4-DAX

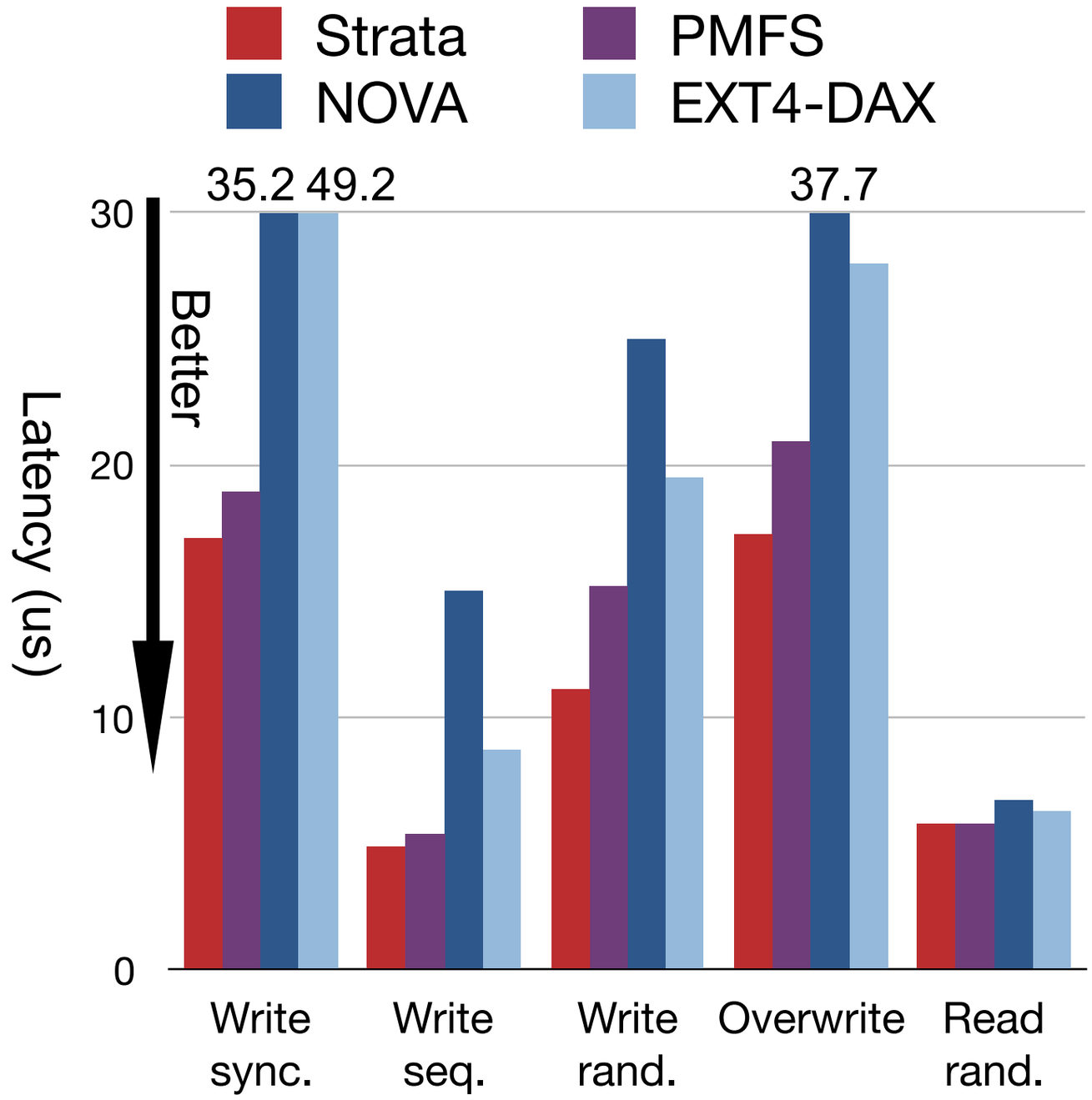


Latency: LevelDB

- 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
    - Tied with PMFS

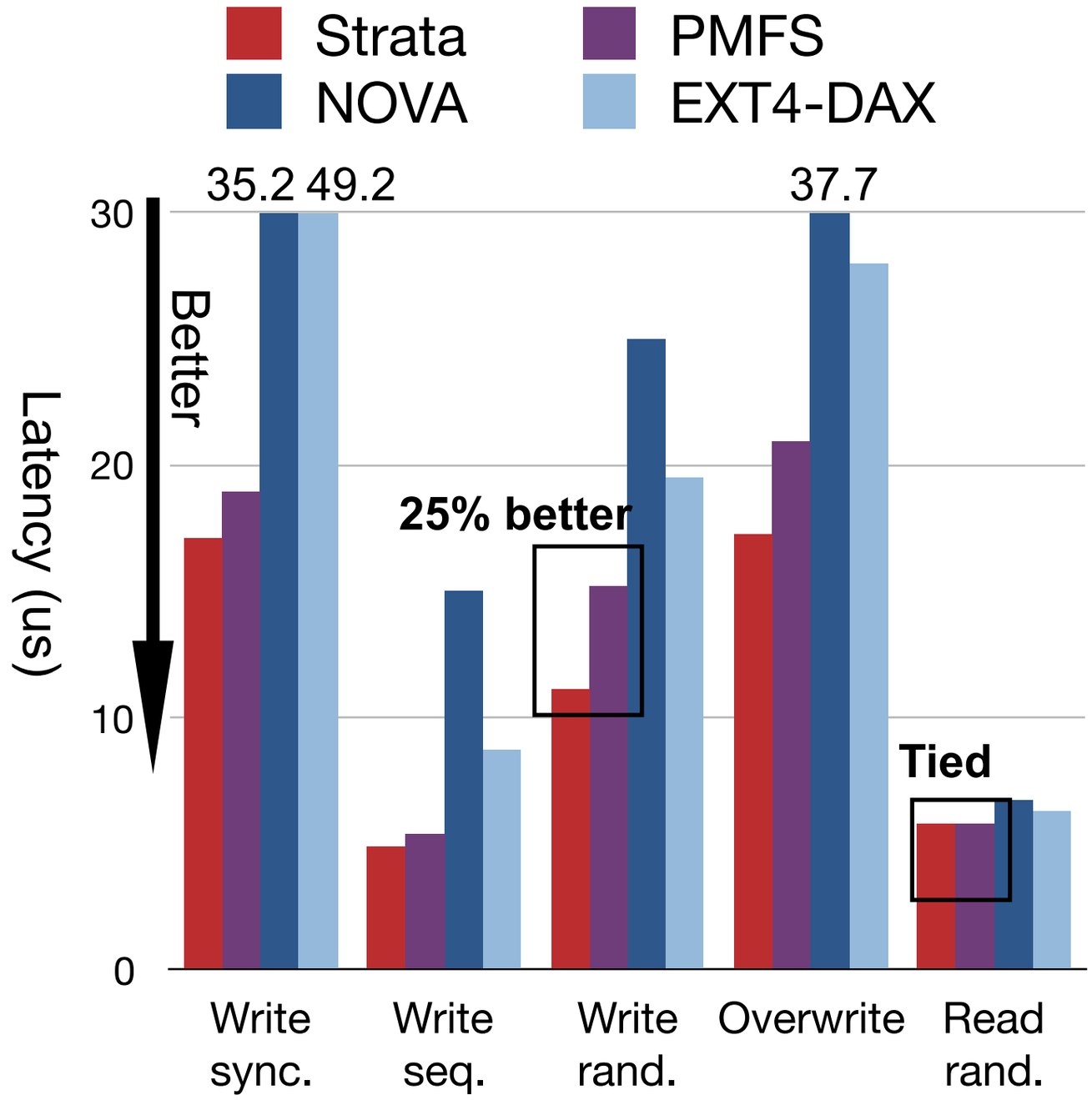


Latency: LevelDB

- 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
    - Tied with PMFS

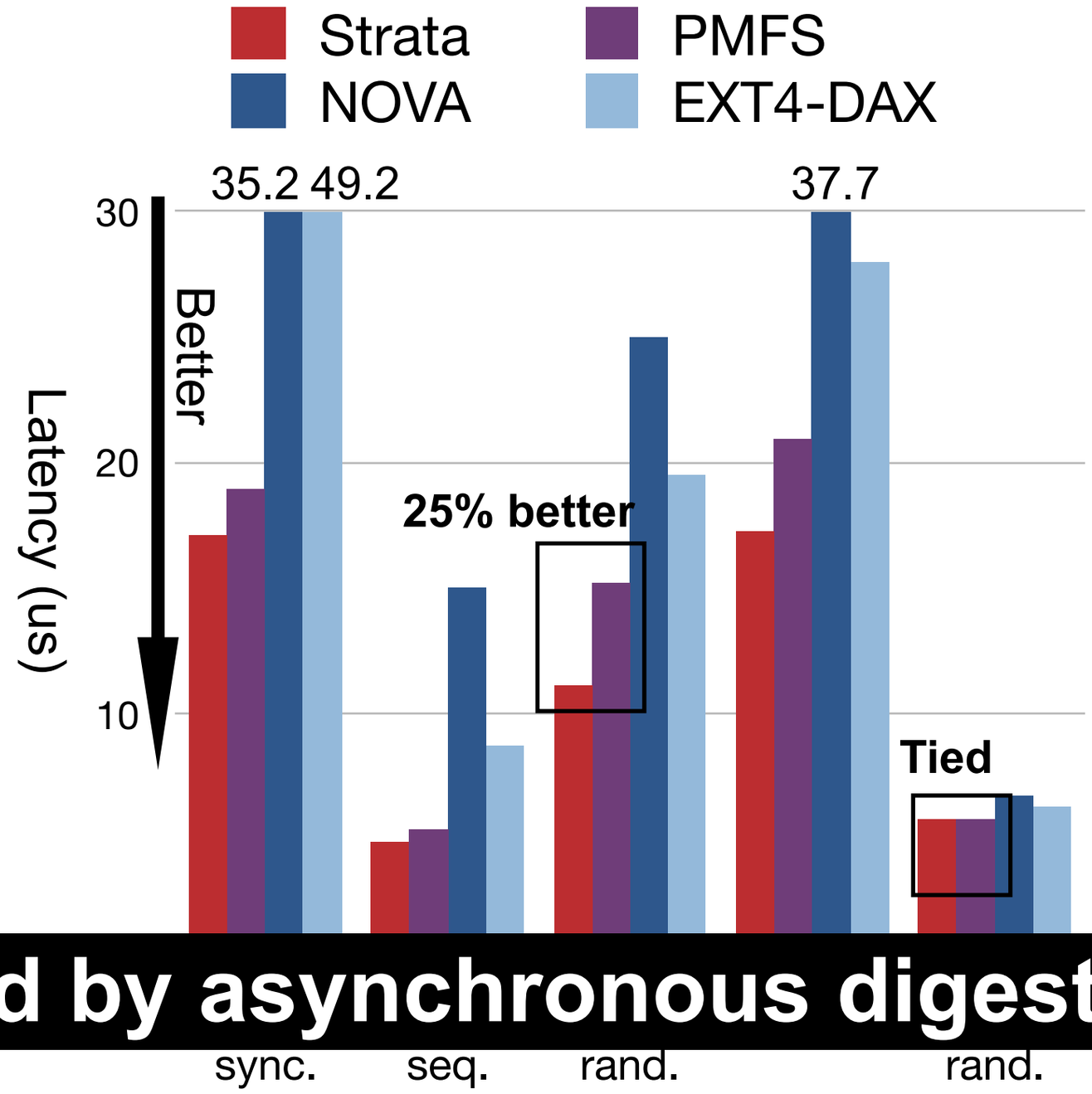


Latency: LevelDB

- 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
    - Tied with PMFS



IO latency not impacted by asynchronous digest

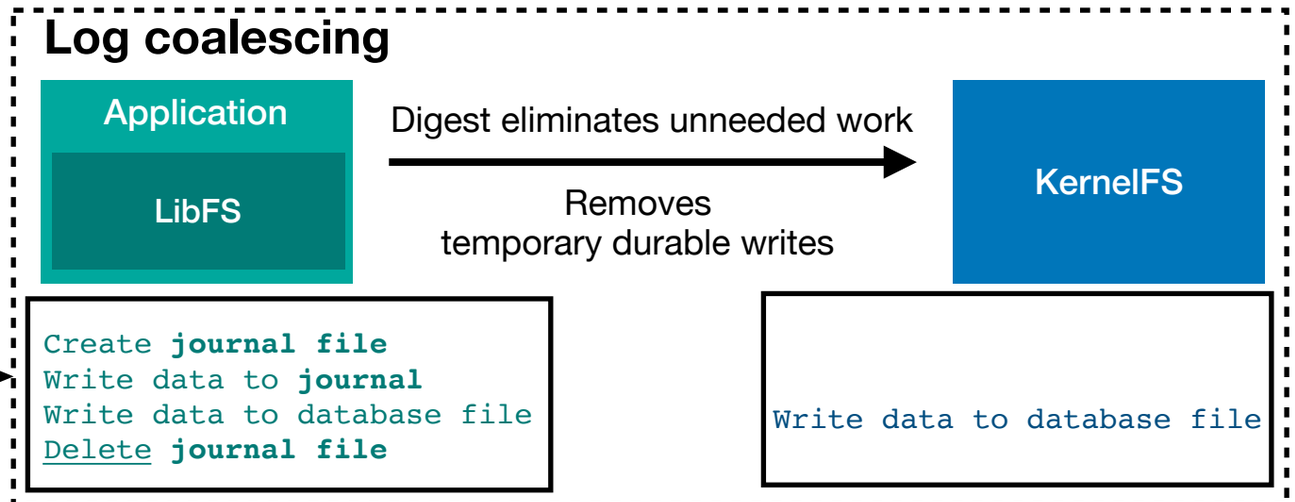
Evaluation questions

- **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

Mail server workload from Filebench

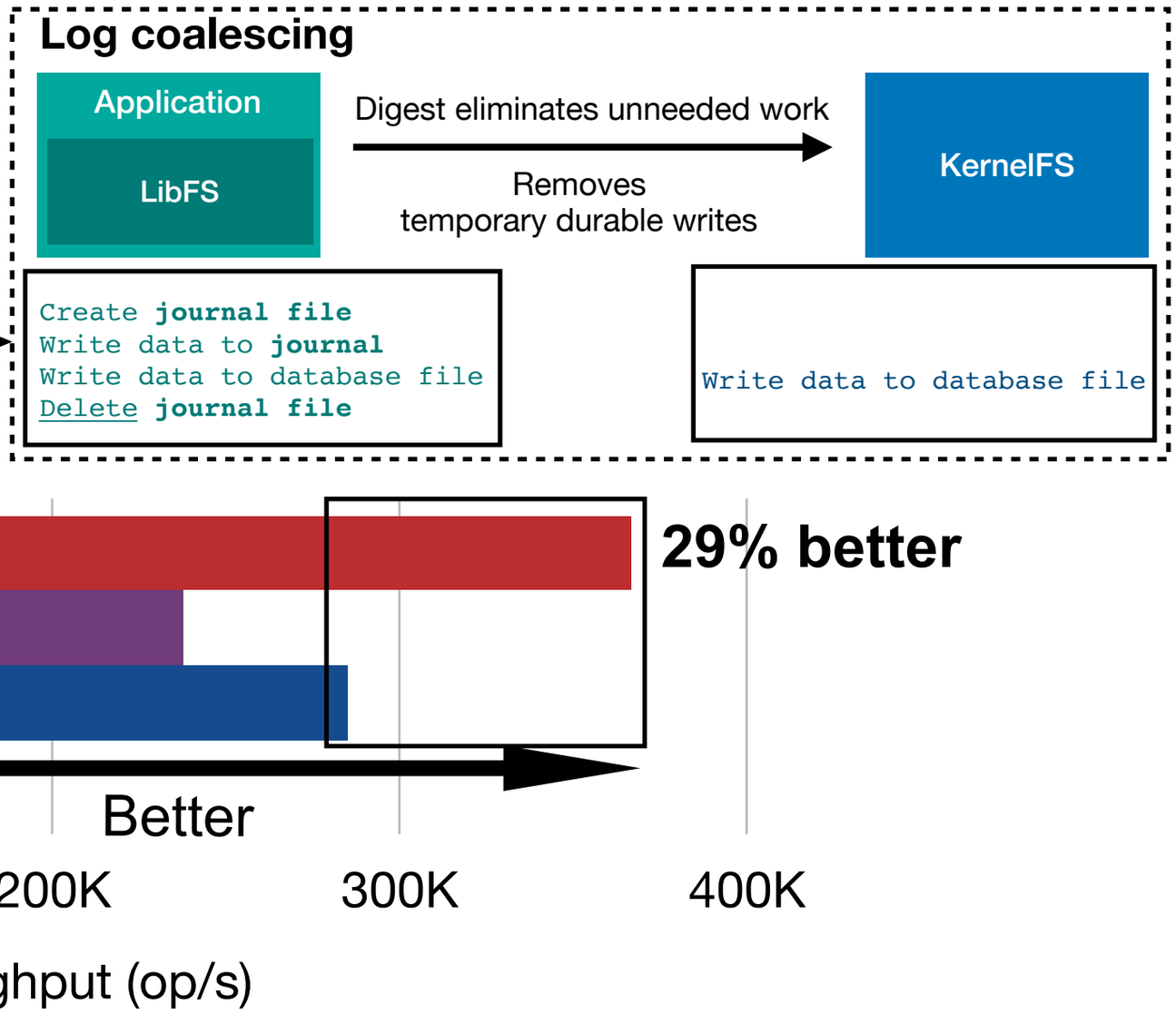
- Using only NVM
- 10000 files
- Read/Write ratio is 1:1
- **Write-ahead logging**

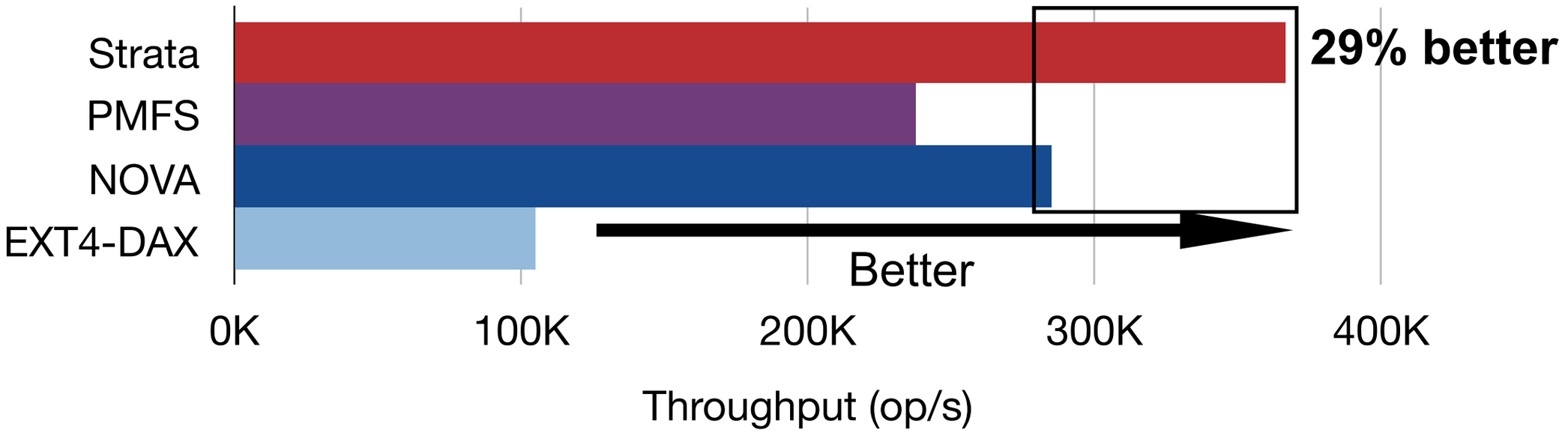


Throughput: Varmail

Mail server workload from Filebench

- Using only NVM
- 10000 files
- Read/Write ratio is 1:1
- **Write-ahead logging**

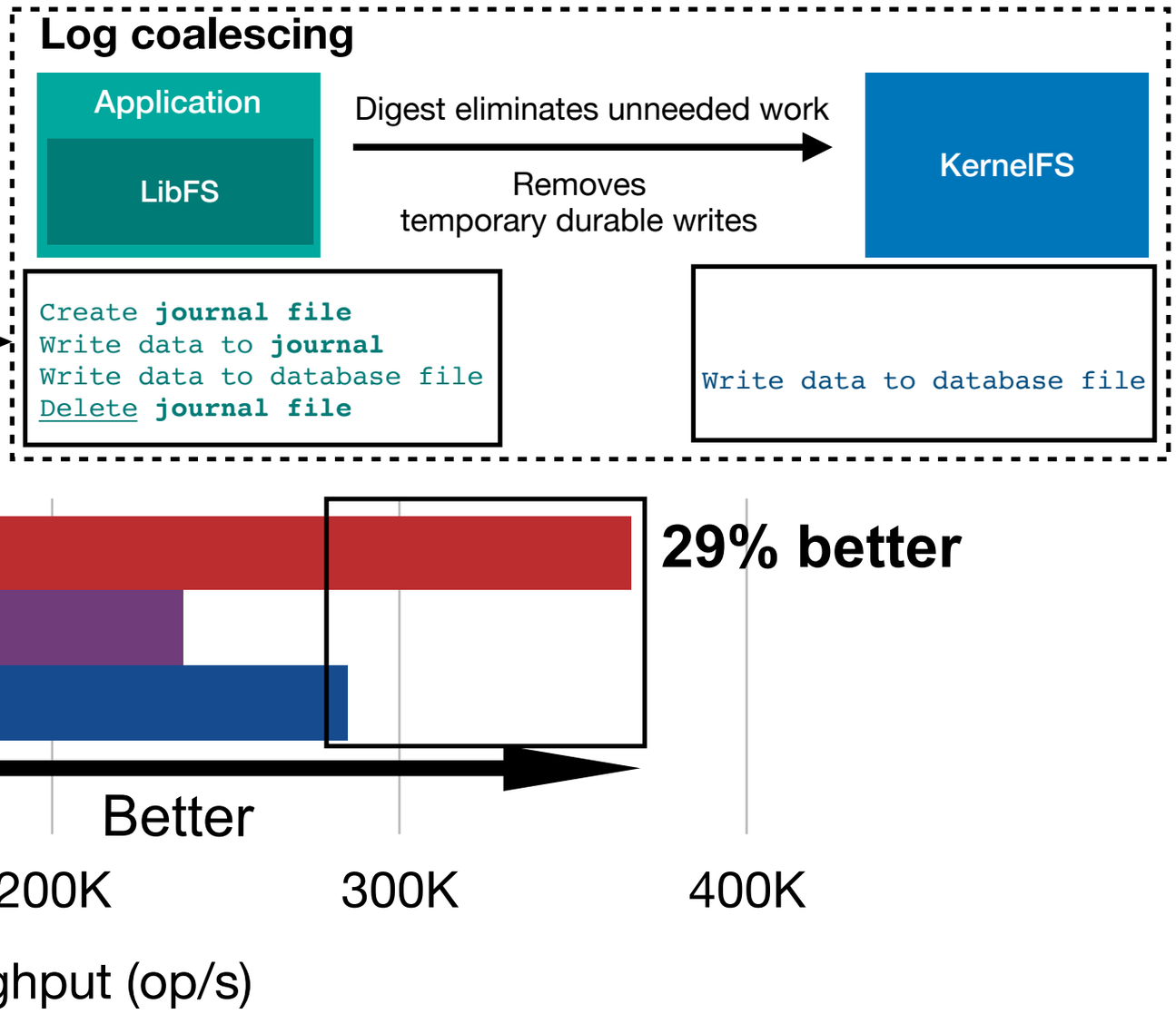




Throughput: Varmail

Mail server workload from Filebench

- Using only NVM
- 10000 files
- Read/Write ratio is 1:1
- **Write-ahead logging**



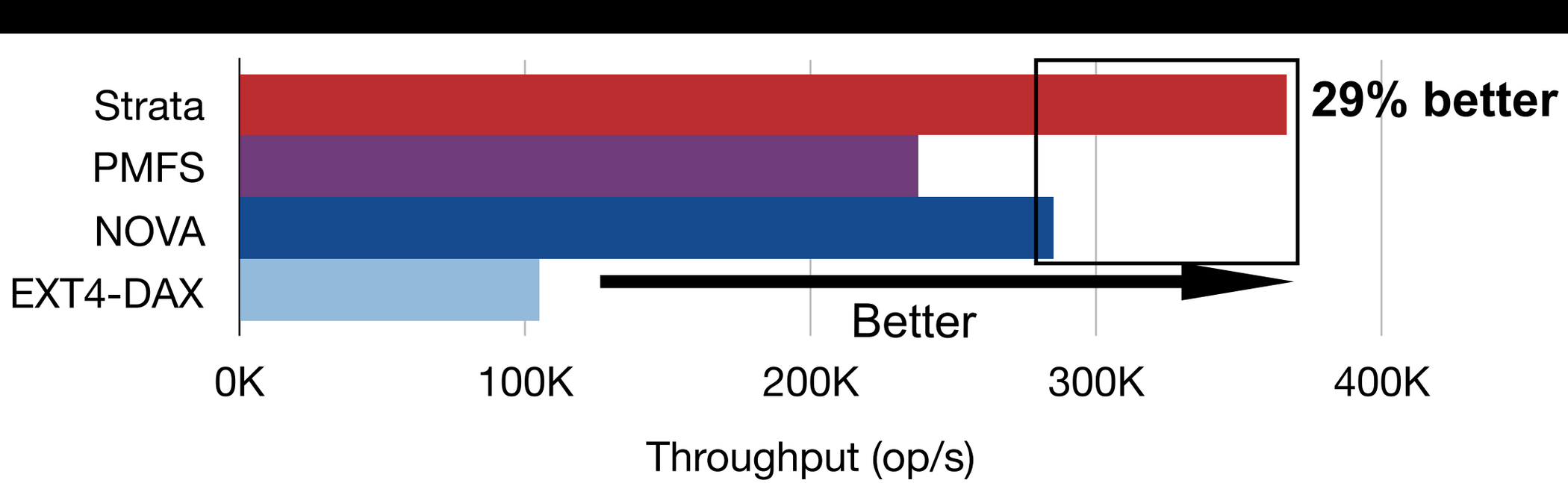
Log coalescing eliminates 86% of log entries, saving 14 GB of IO

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

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

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

Throughput: data migration

File server workload from Filebench

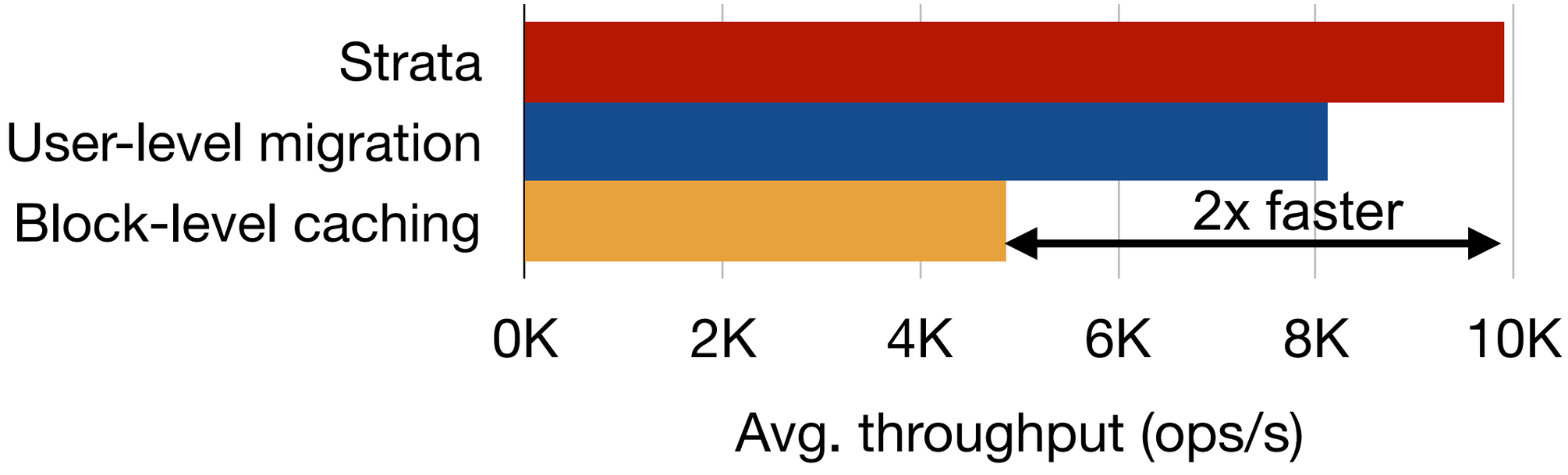
- Working set starts at NVM, grows to SSD, HDD
- Read/Write ratio is 1:2

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



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

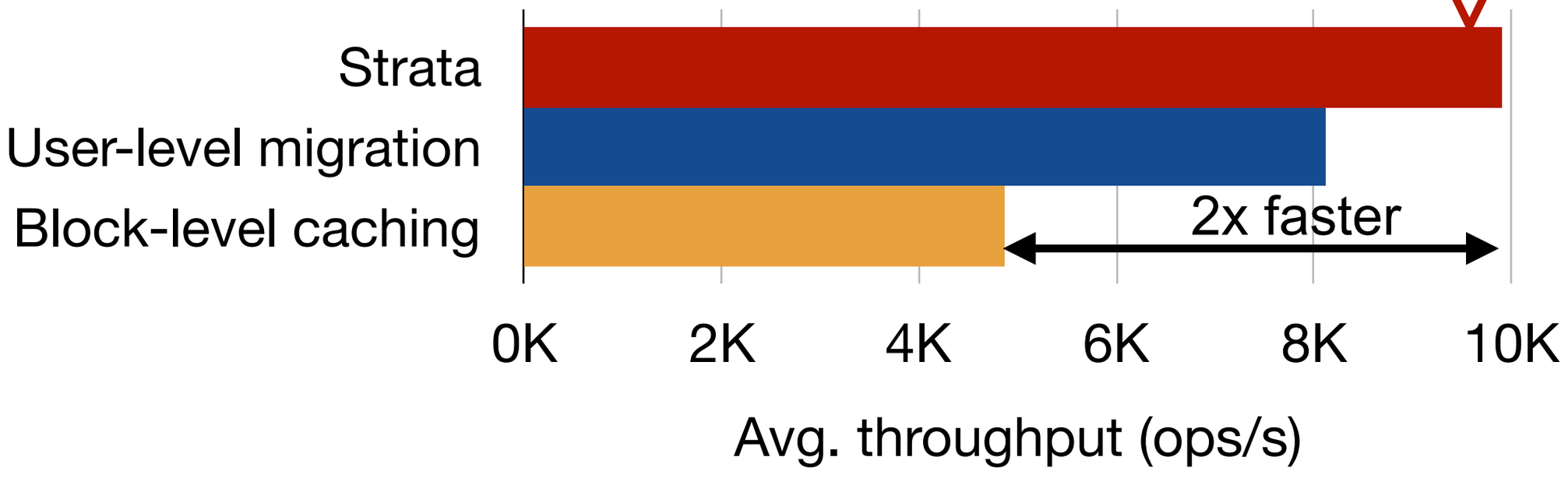
- LRU: whole file granularity
- Treat each file system as a black-box
- NVM: NOVA, SSD: F2FS, HDD: EXT4

Block-level caching

- Linux LVM cache, formatted with F2FS

22% faster than user-level migration

Cross layer optimization: placing hot metadata in faster layers



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

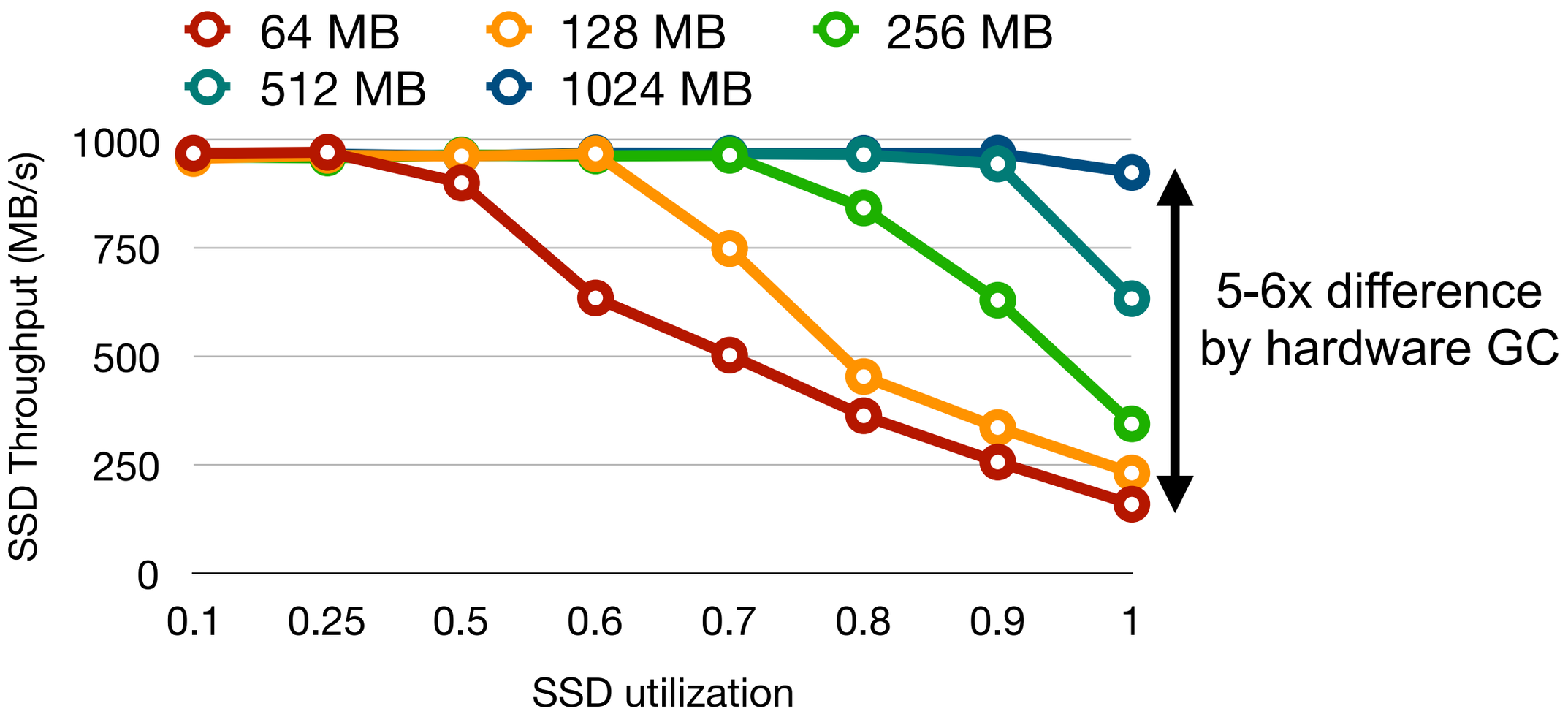
Source code is available at
https://github.com/ut-osa/strata

Backup

Device management overhead

SSD, HDD prefer large sequential IO

For example, SSD **Random** write:



Sequential writes avoid management overhead