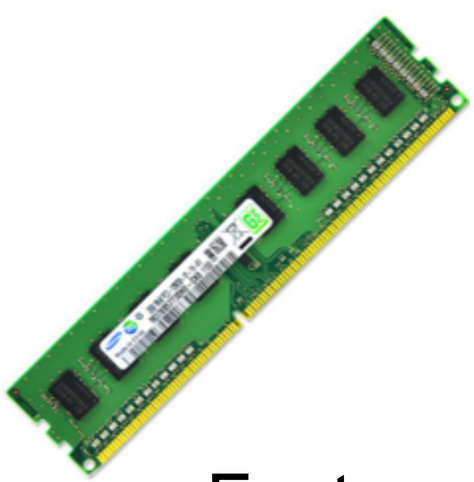
SplitFS: Reducing Software Overhead in File Systems for Persistent Memory

Rohan Kadekodi, Se Kwon Lee, Sanidhya Kashyap*,
Taesoo Kim, Aasheesh Kolli, Vijay Chidambaram

* on the job market

Persistent Memory (PM)

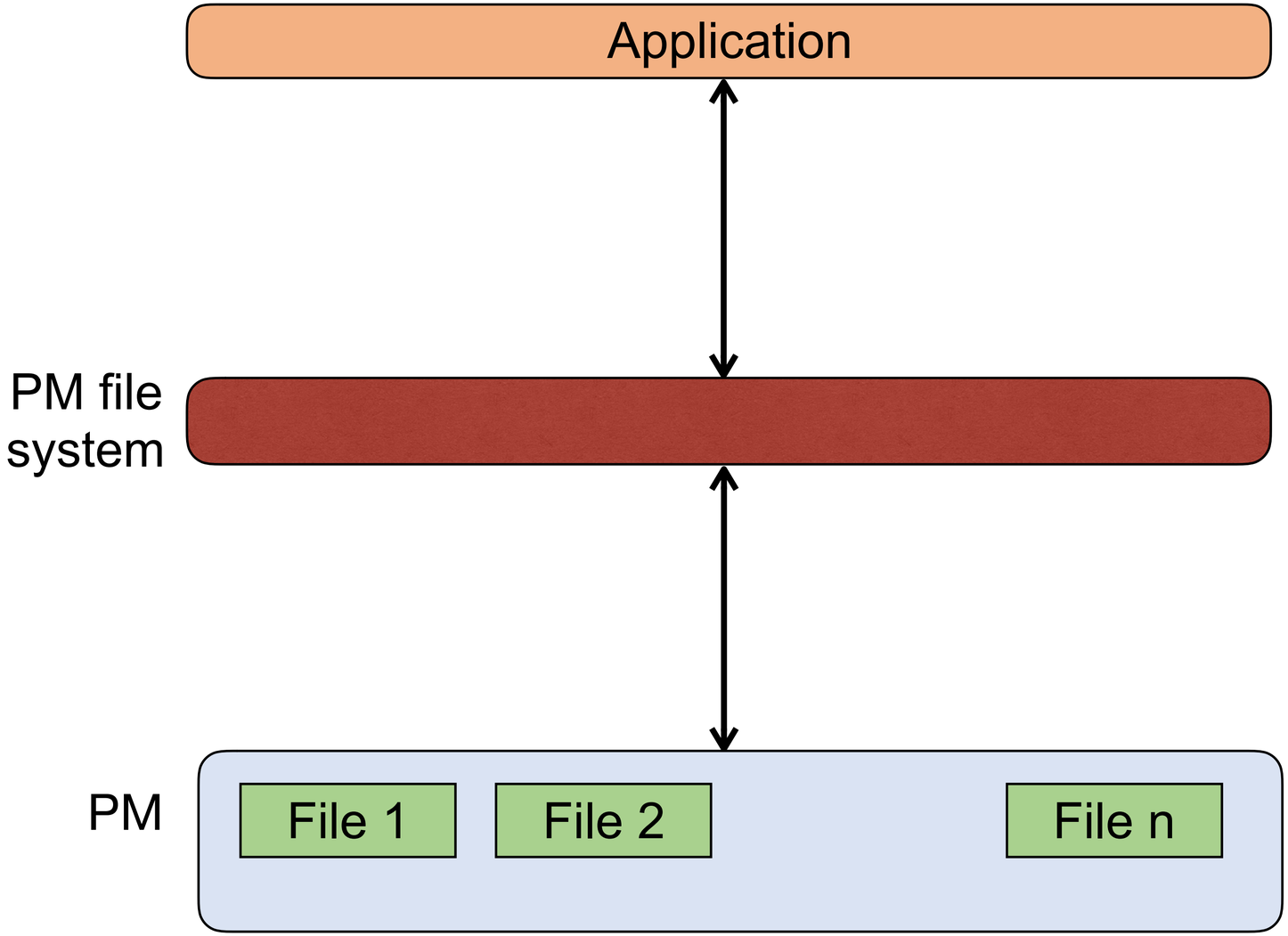
Non-volatile



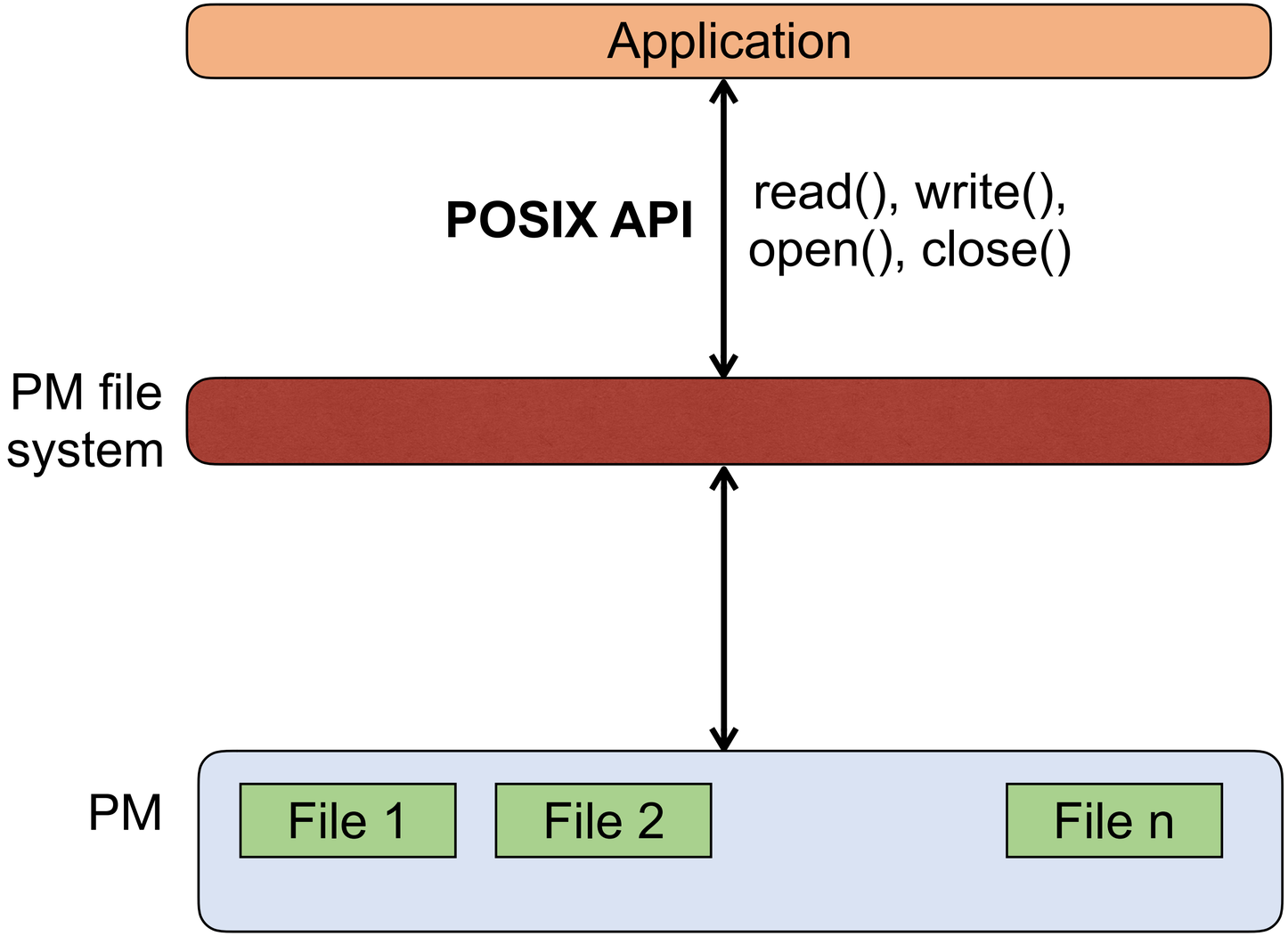
Fast

PM file systems

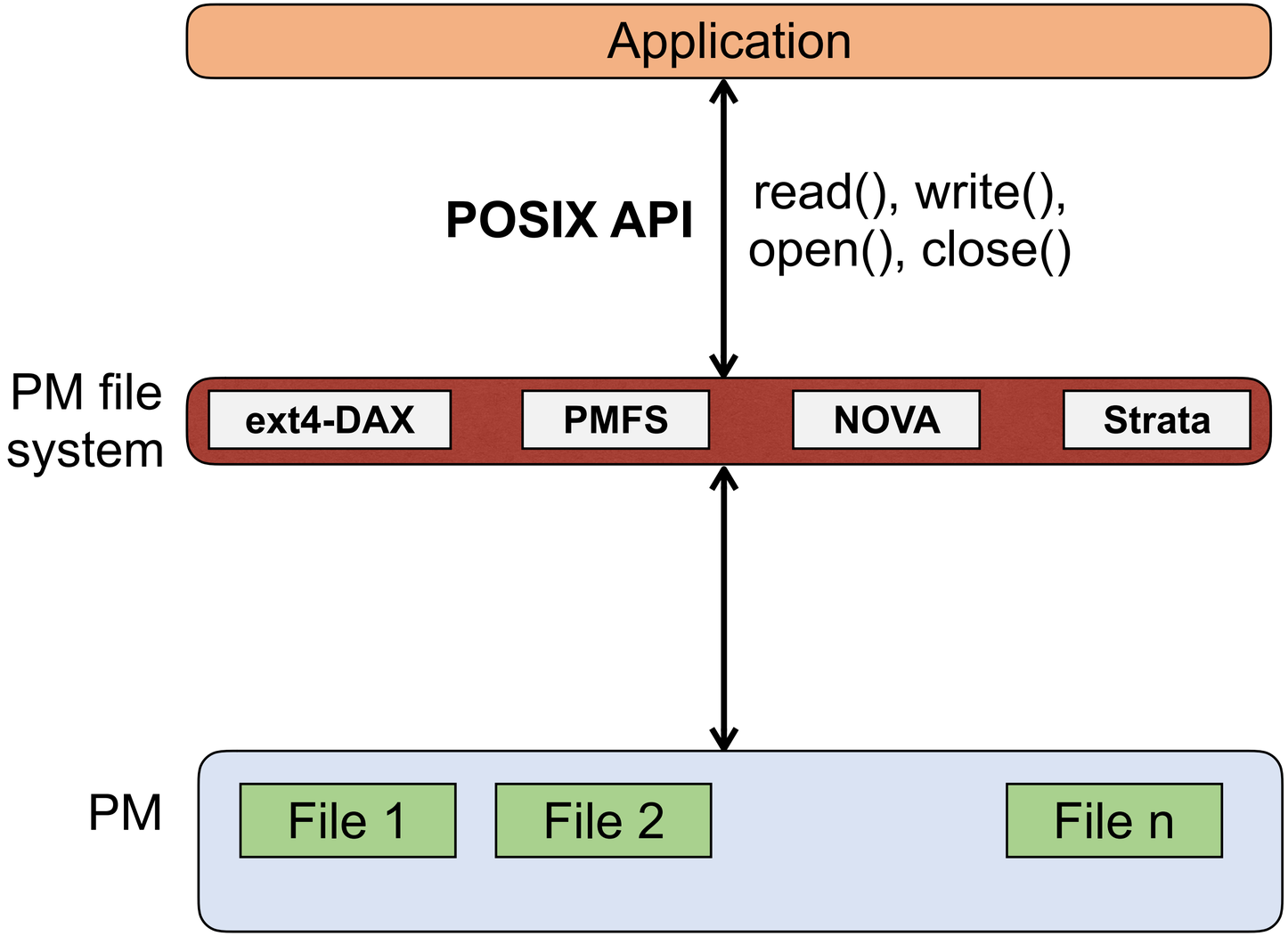
PM file systems



PM file systems



PM file systems



ext4-DAX

ext4-DAX

Modification of the ext4 file system for Persistent Memory

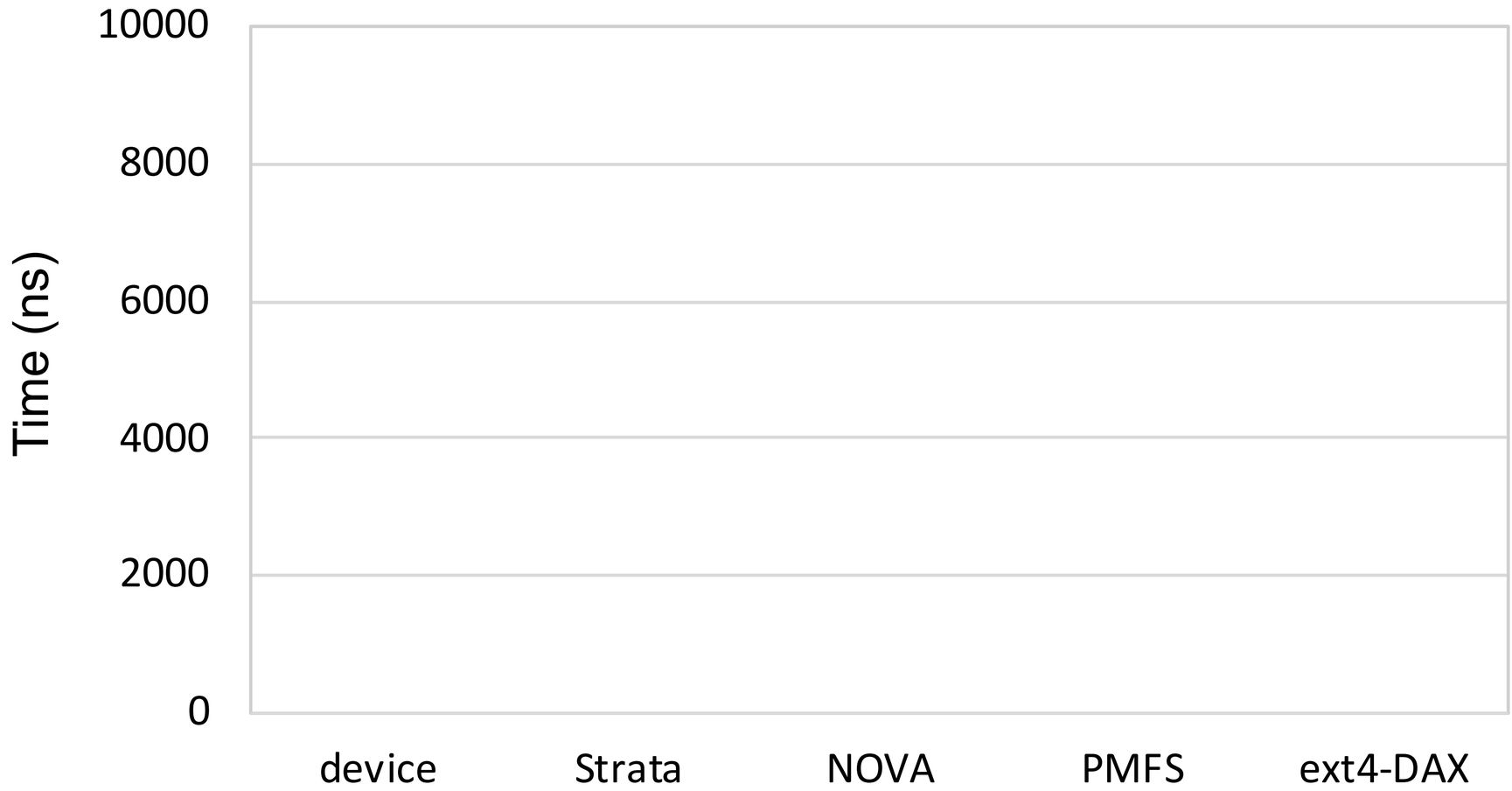
Works with modern Linux kernels

Under active development by the ext4 community

Only PM file system that is widely used

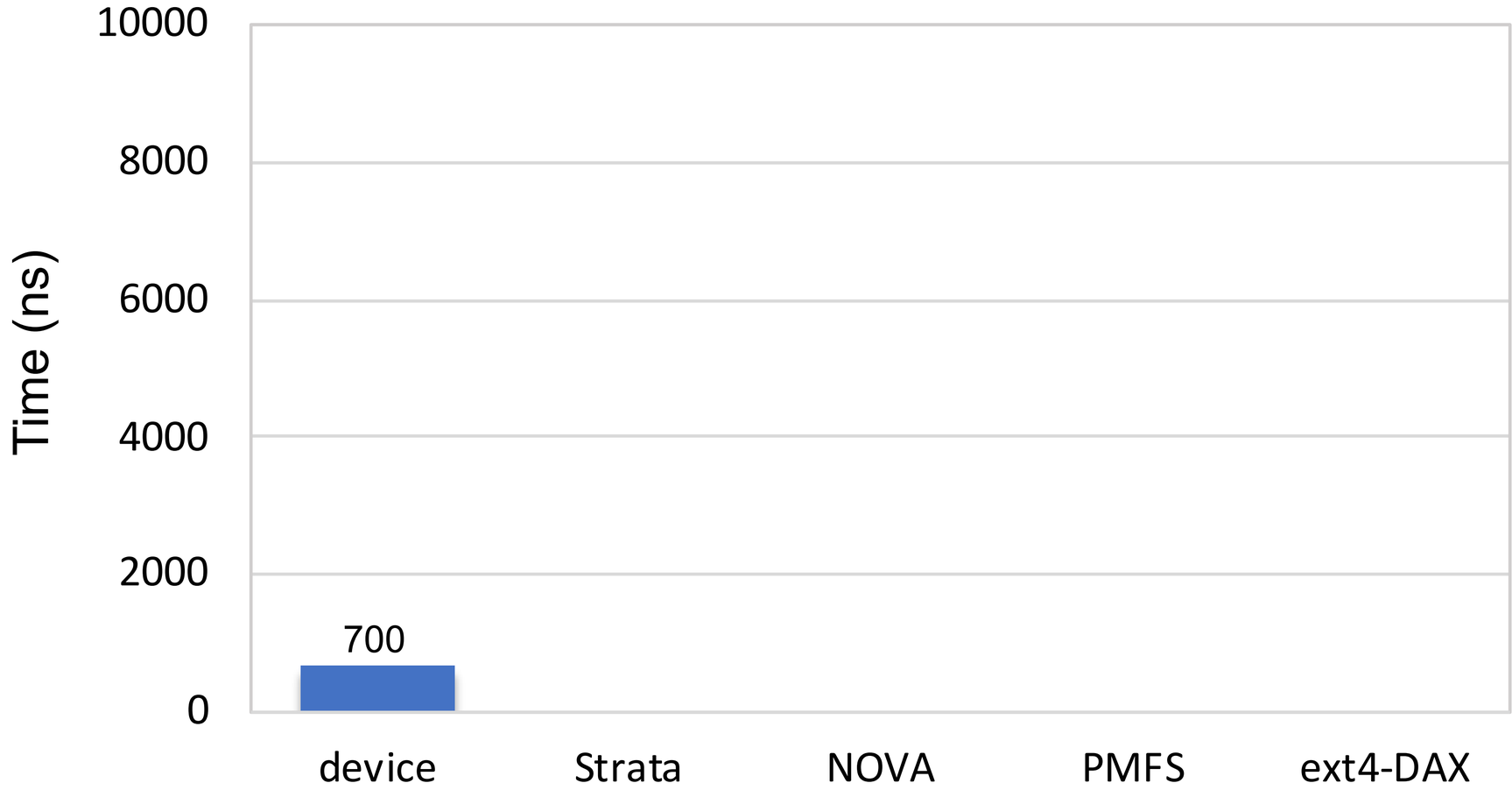
Software Overhead in File Systems

- Append 4KB data to a file



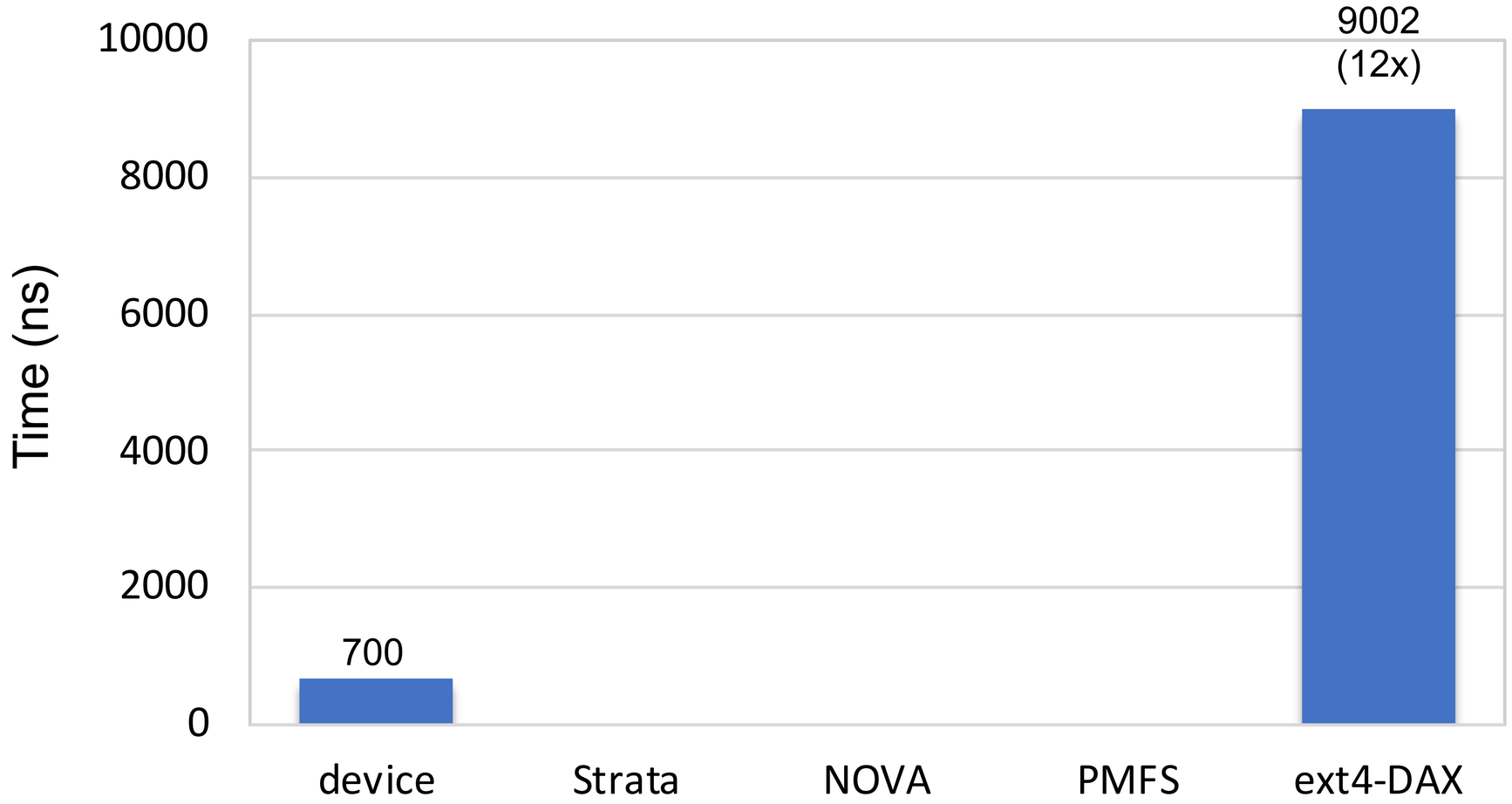
Software Overhead in File Systems

- Append 4KB data to a file
- Time taken to copy user data to PM: ~700 ns



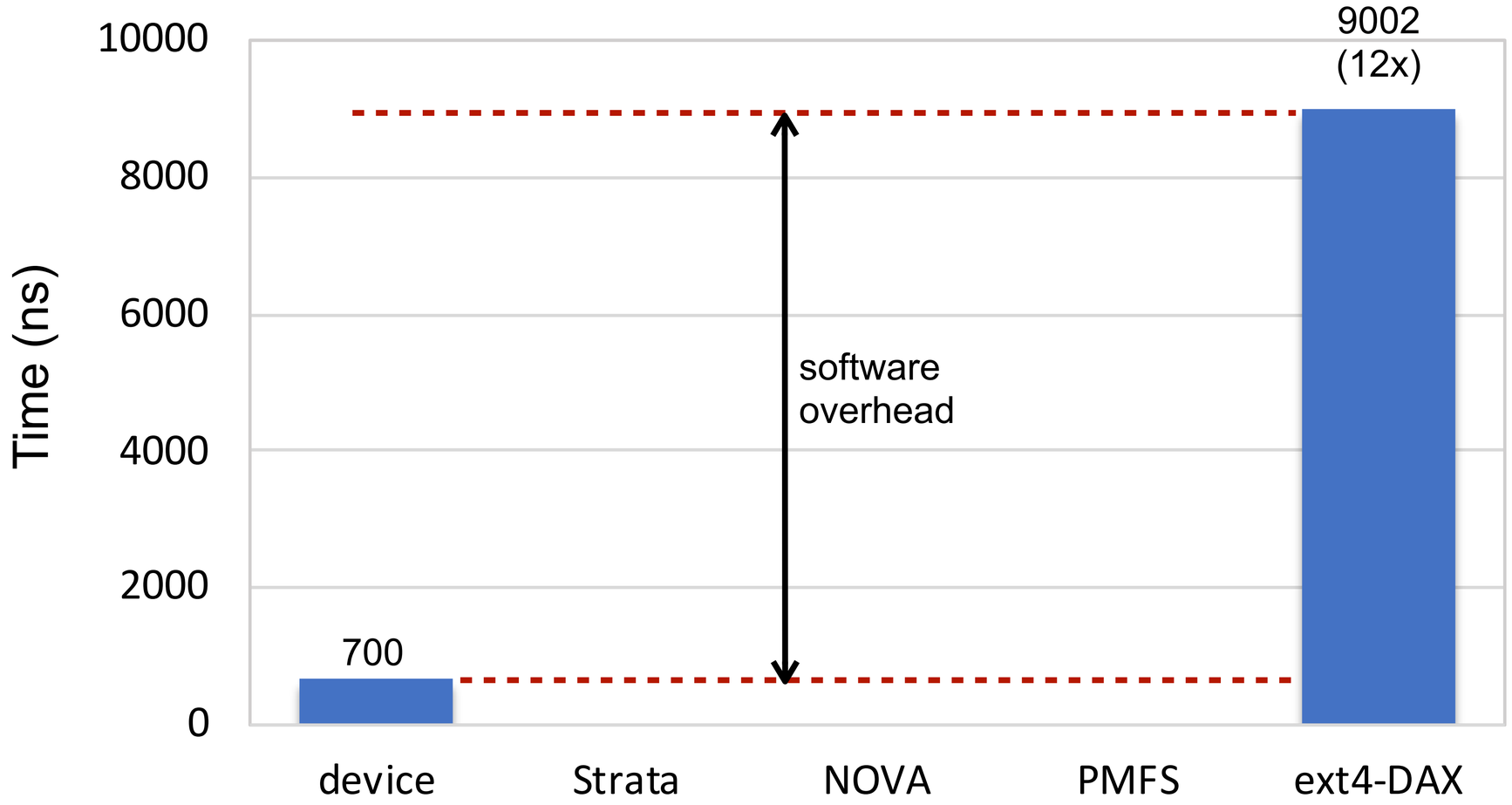
Software Overhead in File Systems

- Append 4KB data to a file
- Time taken to copy user data to PM: ~700 ns



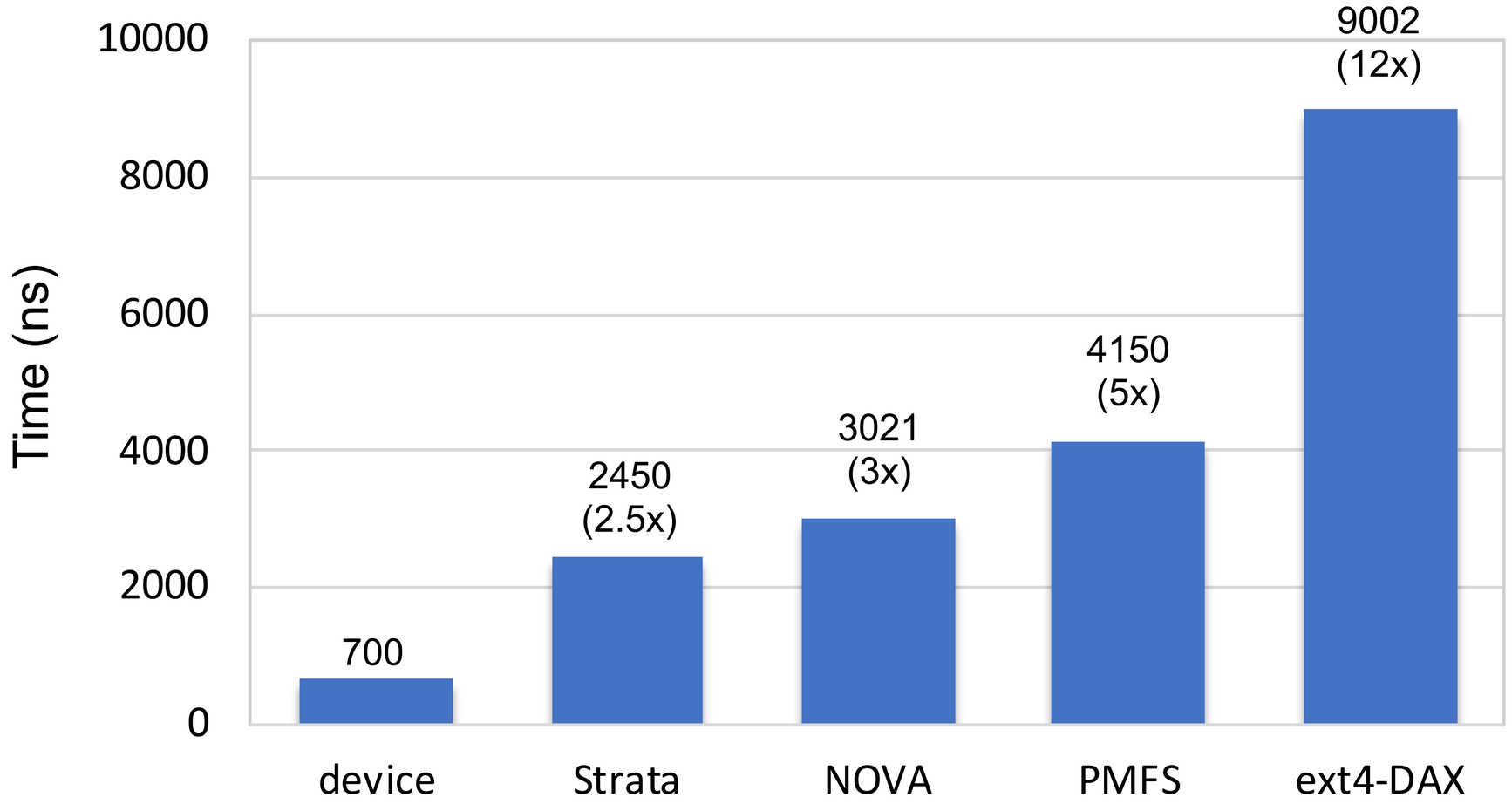
Software Overhead in File Systems

- Append 4KB data to a file
- Time taken to copy user data to PM: ~700 ns



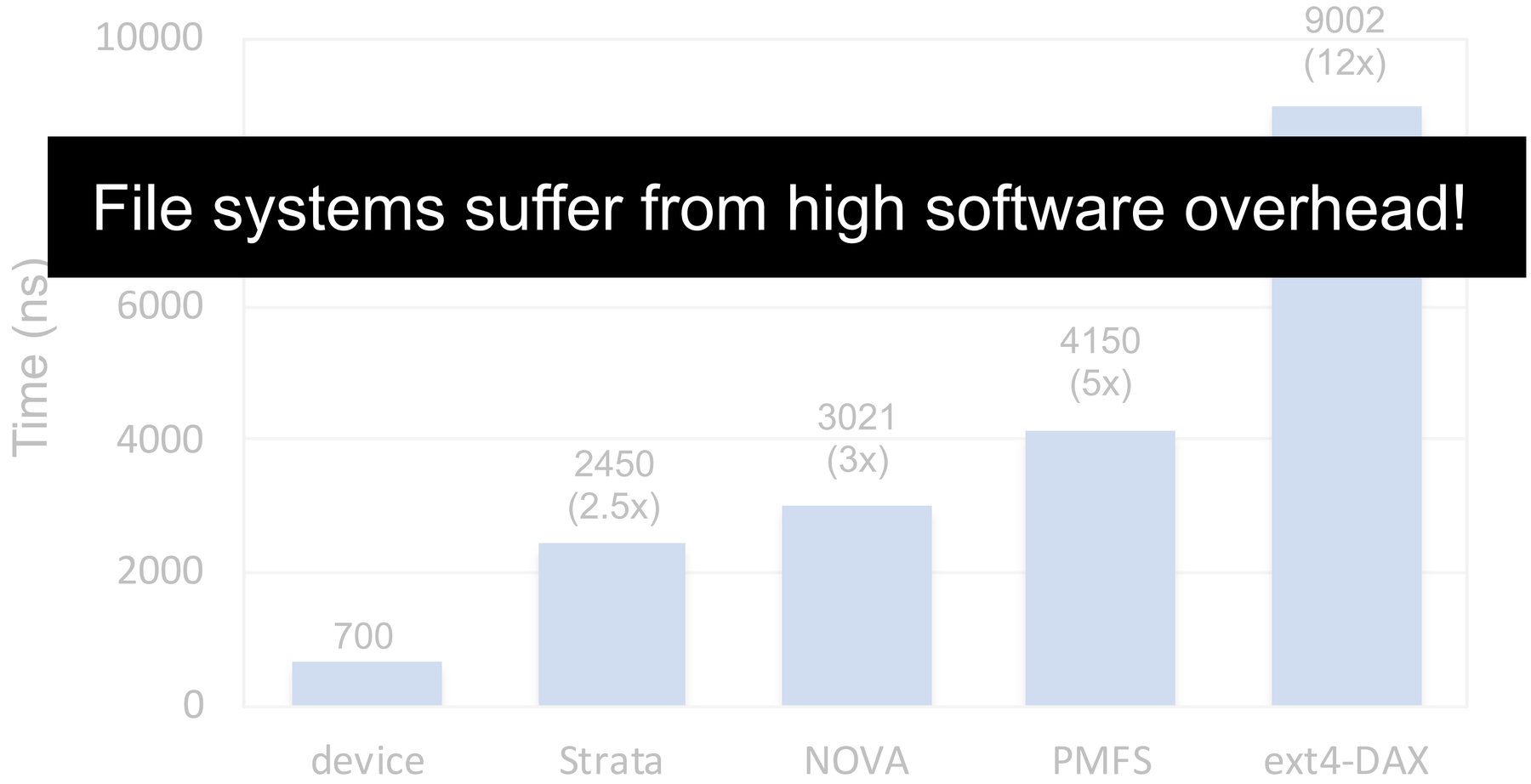
Software Overhead in File Systems

- Append 4KB data to a file
- Time taken to copy user data to PM: ~700 ns



Software Overhead in File Systems

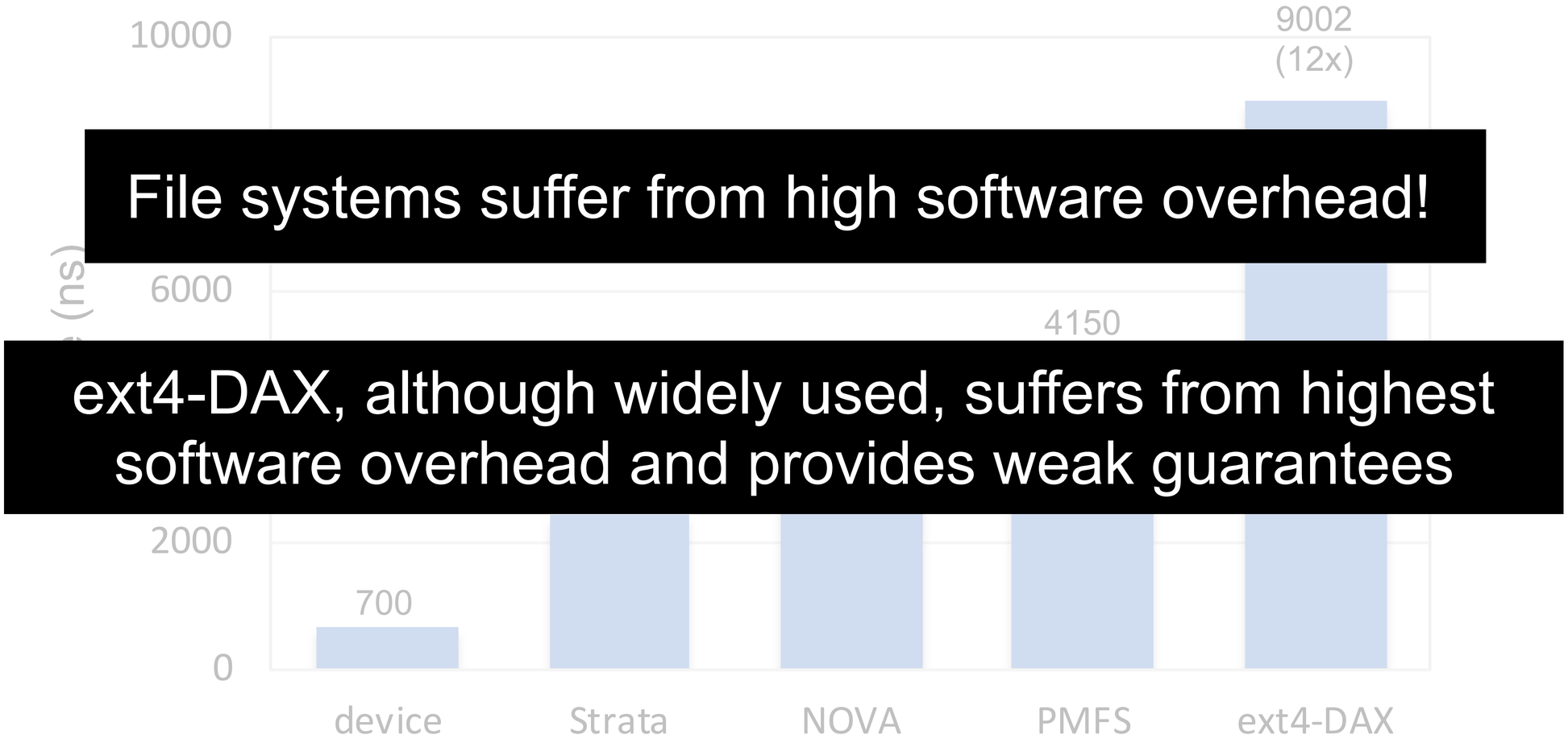
- Append 4KB data to a file
- Time taken to copy user data to PM: ~700 ns



File systems suffer from high software overhead!

Software Overhead in File Systems

- Append 4KB data to a file
- Time taken to copy user data to PM: ~700 ns



File systems suffer from high software overhead!

ext4-DAX, although widely used, suffers from highest software overhead and provides weak guarantees

Goals

Goals

- Low software overhead

Goals

- Low software overhead
- Strong consistency guarantees

Goals

- Low software overhead
- Strong consistency guarantees
- Leverage the maturity and active development of ext4-DAX

SplitFS

POSIX file system aimed at reducing software overhead for PM

SplitFS

POSIX file system aimed at reducing software overhead for PM

SplitFS serves data operations from user space and metadata operations using the ext4-DAX kernel file system

SplitFS

POSIX file system aimed at reducing software overhead for PM

SplitFS serves data operations from user space and metadata operations using the ext4-DAX kernel file system

Provides strong guarantees such as atomic and synchronous data operations

SplitFS

POSIX file system aimed at reducing software overhead for PM

SplitFS serves data operations from user space and metadata operations using the ext4-DAX kernel file system

Provides strong guarantees such as atomic and synchronous data operations

Reduces software overhead by up to 17x compared to ext4-DAX

Improves application throughput by up to 2x compared to NOVA

https://github.com/utsaslab/splitfs

Outline

- Target usage scenario
- High-level design
- Handling data operations
- Consistency guarantees
- Evaluation

Outline

- Target usage scenario
- High-level design
- Handling data operations
- Consistency guarantees
- Evaluation

Target usage scenario

Target usage scenario

SplitFS is targeted at POSIX applications which use read() / write() system calls in order to access their data on Persistent Memory.

Target usage scenario

SplitFS is targeted at POSIX applications which use read() / write() system calls in order to access their data on Persistent Memory.

SplitFS does not optimize for the case when multiple processes concurrently access the same file

Outline

- Target usage scenario
- High-level design
- Handling data operations
- Consistency guarantees
- Evaluation

High-level Design

High-level Design

SplitFS lies both in user space as well as in the kernel.

- Data operations are served from user space
- Metadata operations are served from ext4-DAX

High-level Design

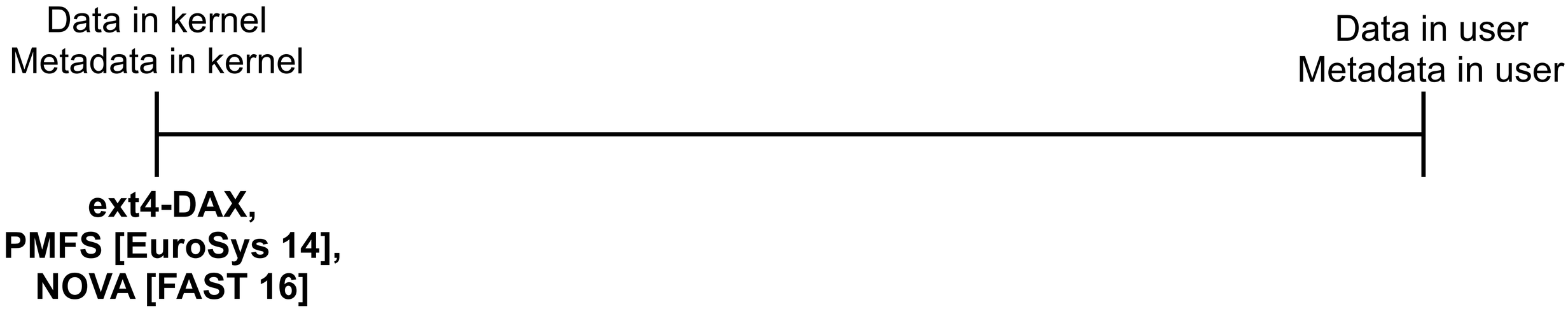
SplitFS lies both in user space as well as in the kernel.

- Data operations are served from user space
- Metadata operations are served from ext4-DAX

High-level Design

SplitFS lies both in user space as well as in the kernel.

- Data operations are served from user space
- Metadata operations are served from ext4-DAX



High-level Design

SplitFS lies both in user space as well as in the kernel.

- Data operations are served from user space
- Metadata operations are served from ext4-DAX

High-level Design

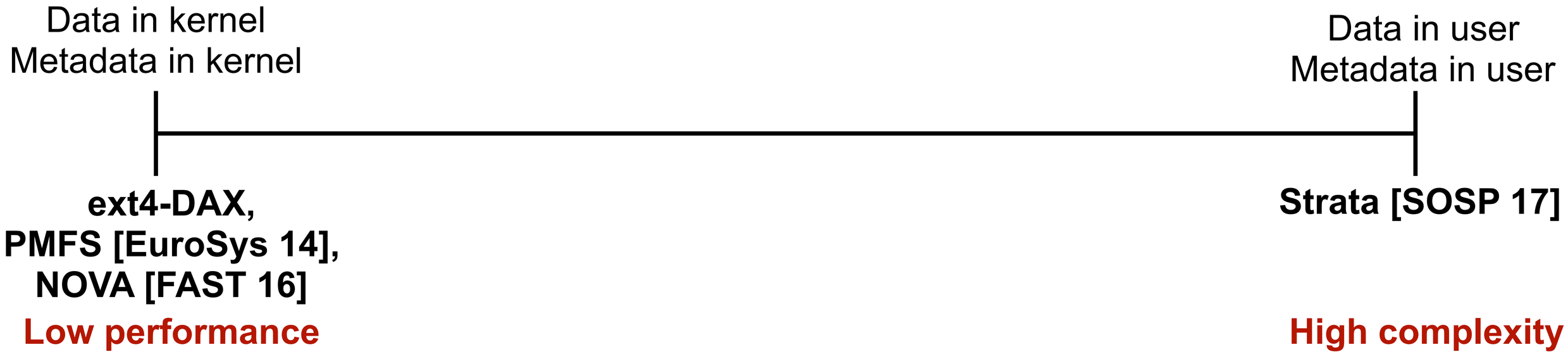
SplitFS lies both in user space as well as in the kernel.

- Data operations are served from user space
- Metadata operations are served from ext4-DAX

High-level Design

SplitFS lies both in user space as well as in the kernel.

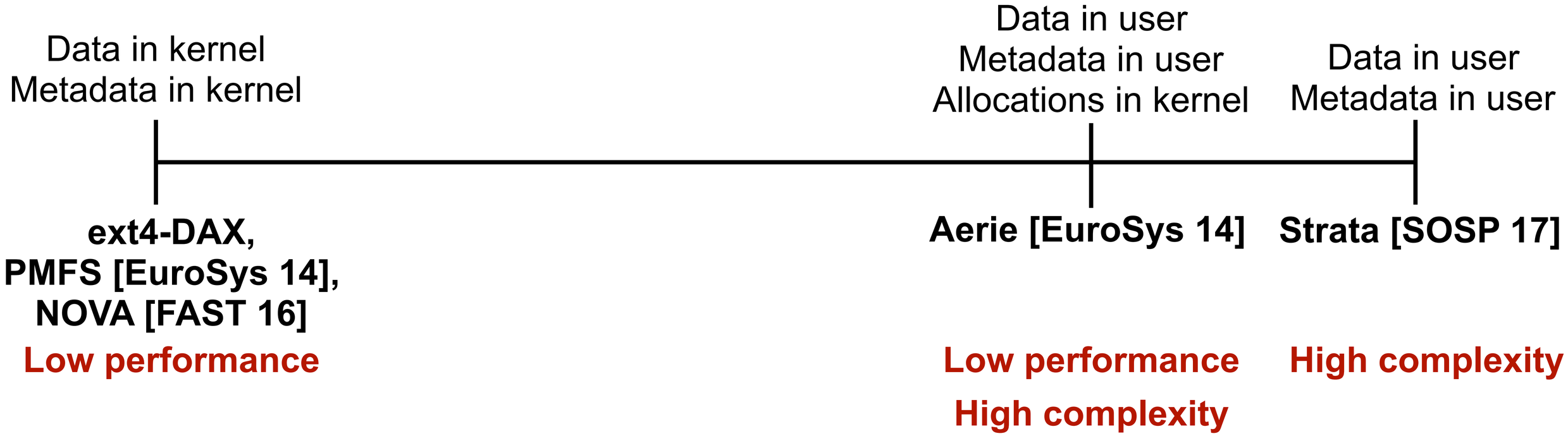
- Data operations are served from user space
- Metadata operations are served from ext4-DAX



High-level Design

SplitFS lies both in user space as well as in the kernel.

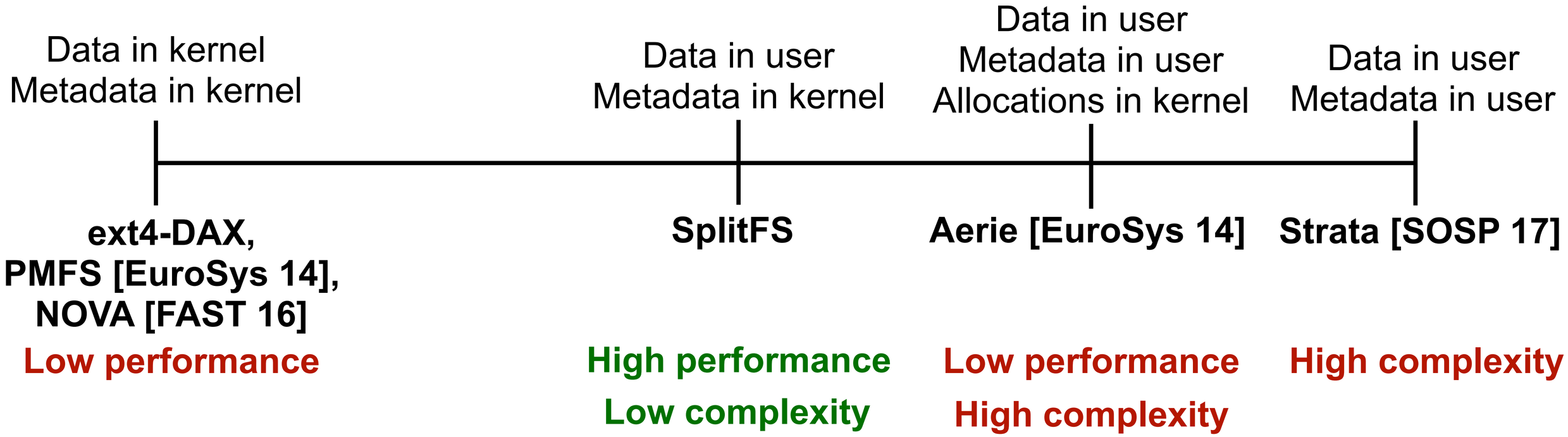
- Data operations are served from user space
- Metadata operations are served from ext4-DAX



High-level Design

SplitFS lies both in user space as well as in the kernel.

- Data operations are served from user space
- Metadata operations are served from ext4-DAX



High-level Design

High performance

Low complexity

High-level Design

High performance

Accelerate data operations from user space

- Data operations are common and simple

Low complexity

High-level Design

High performance

Accelerate data operations from user space

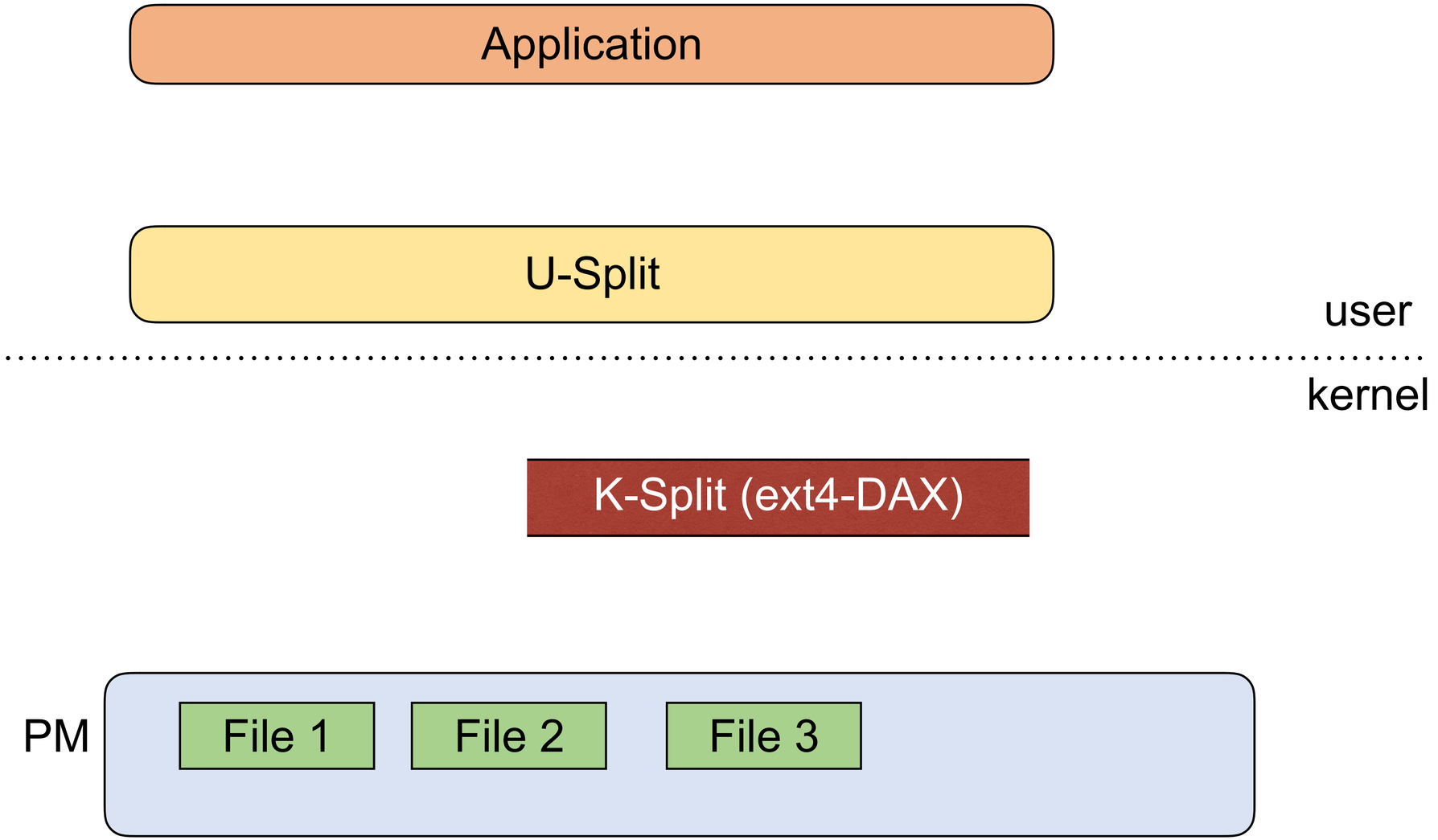
- Data operations are common and simple

Low complexity

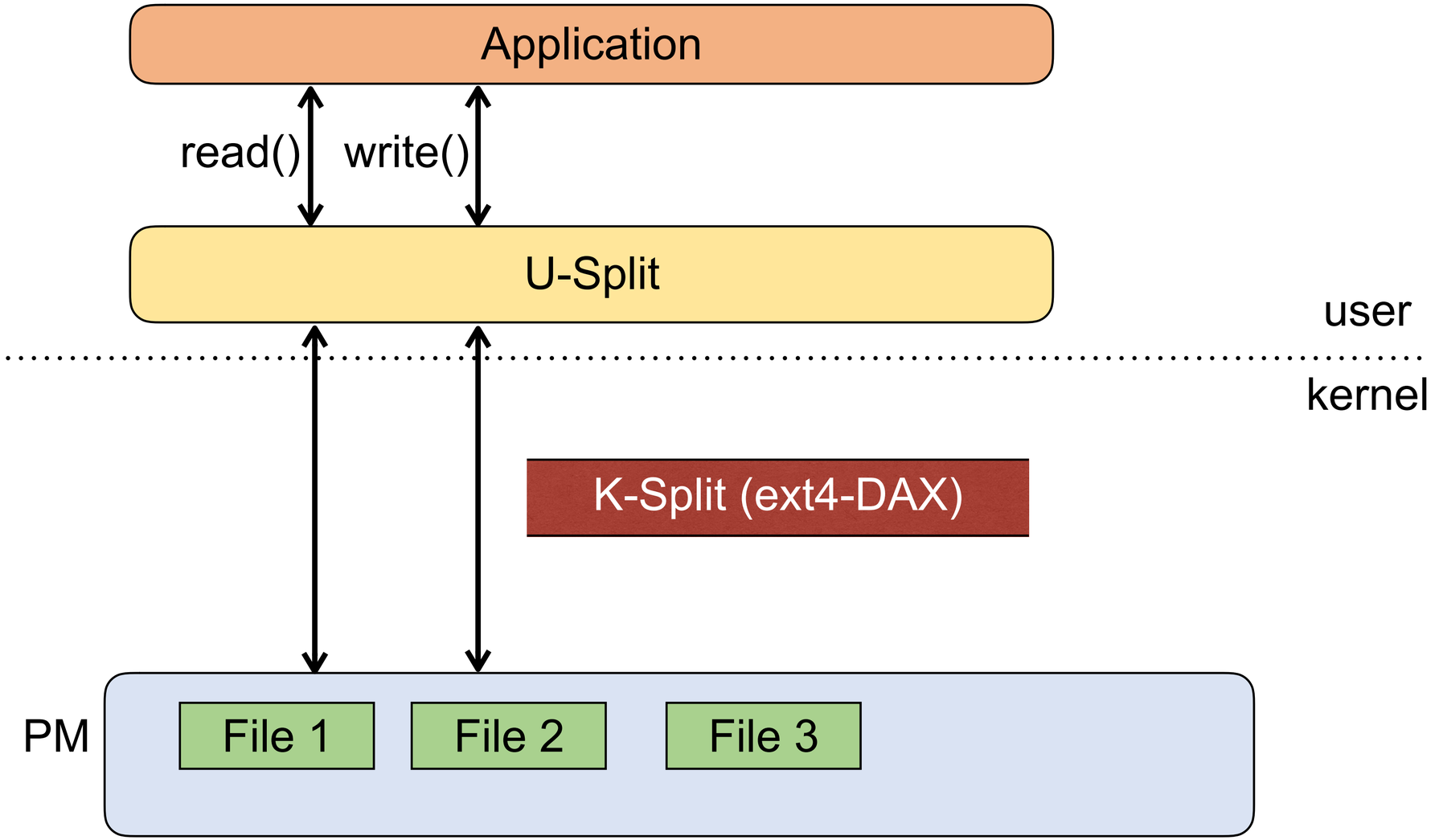
Use ext4-DAX for metadata operations

- Metadata operations are rare and complex
- POSIX has many complex corner-cases

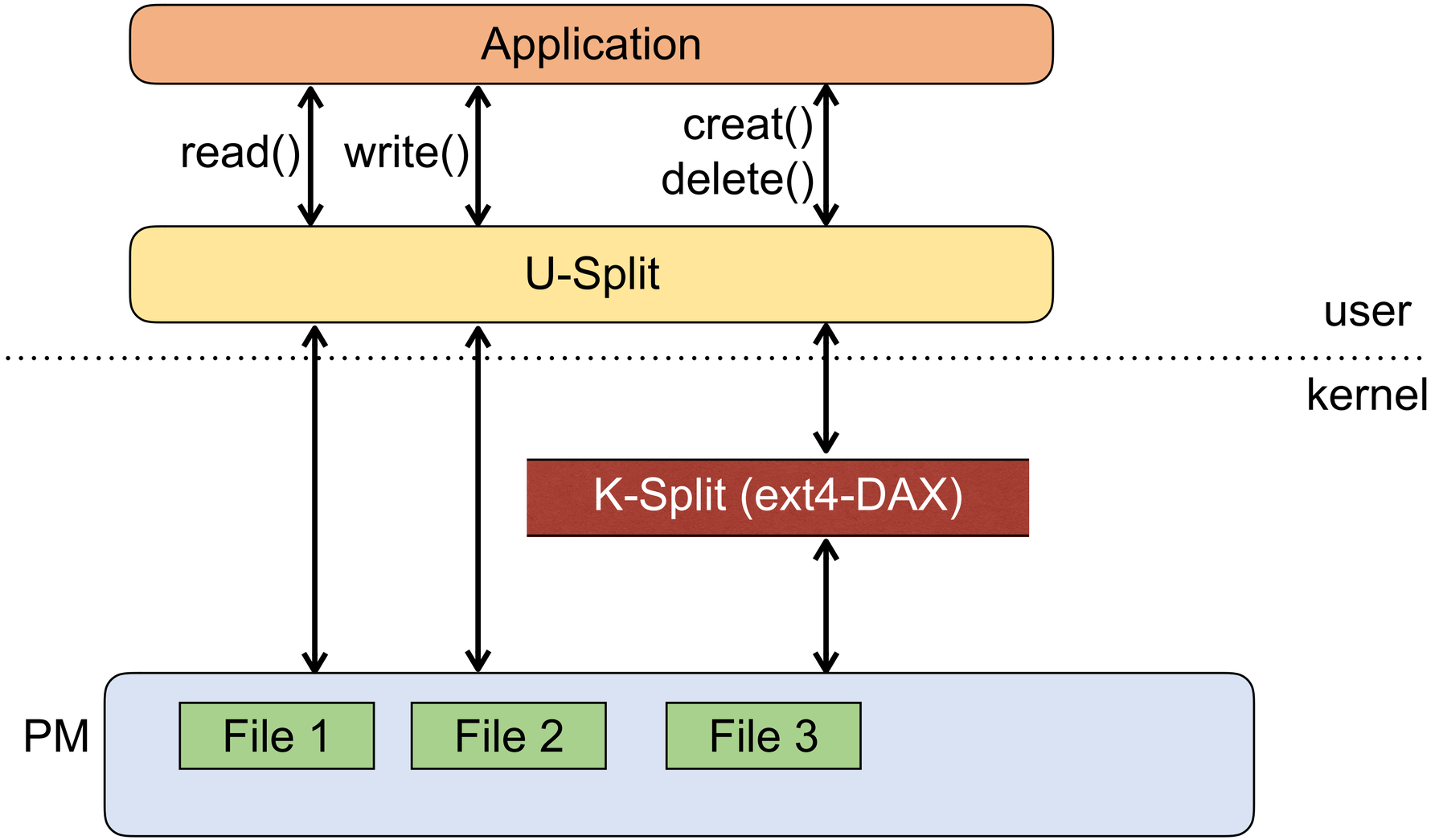
High-level Design



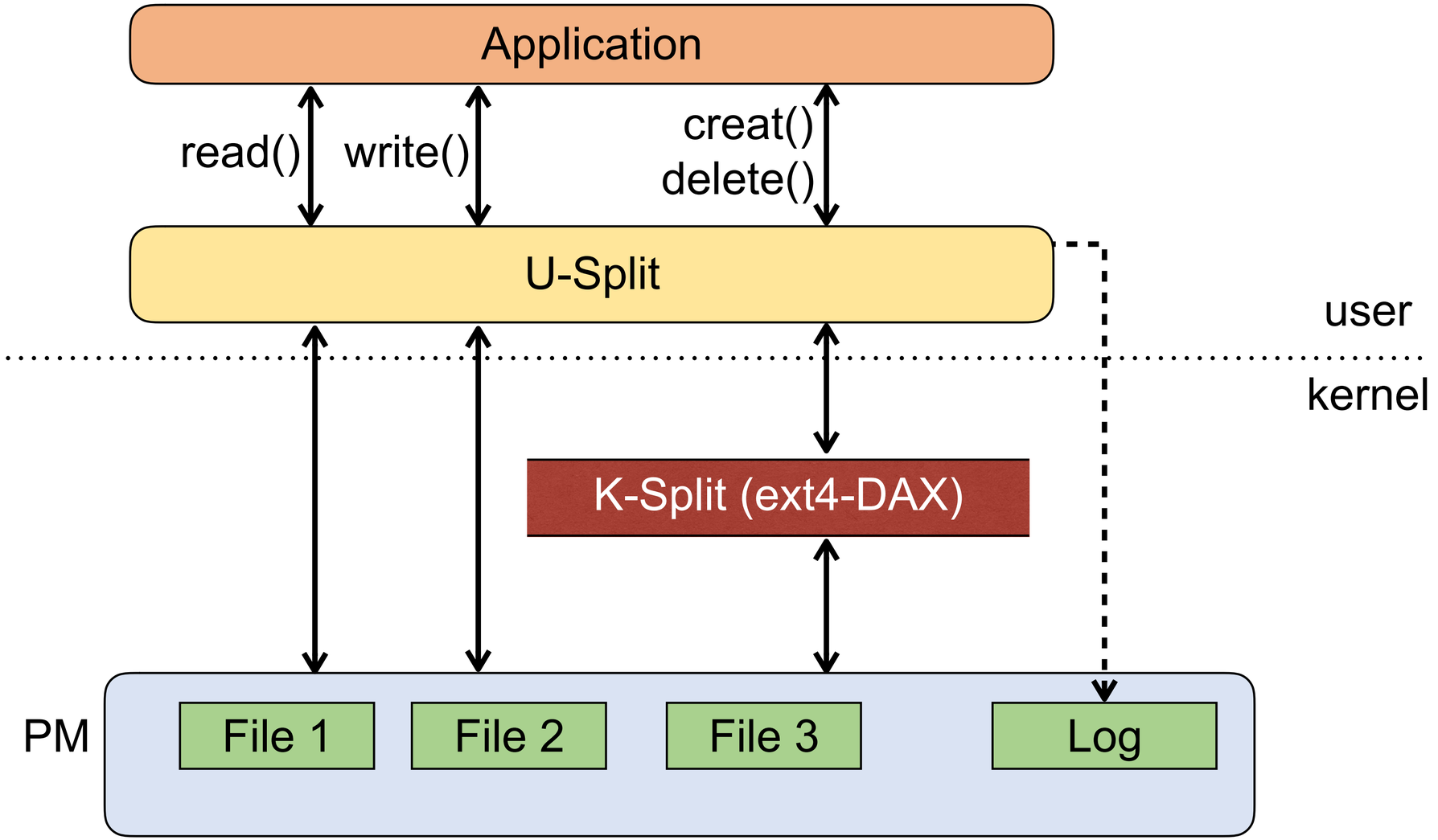
High-level Design



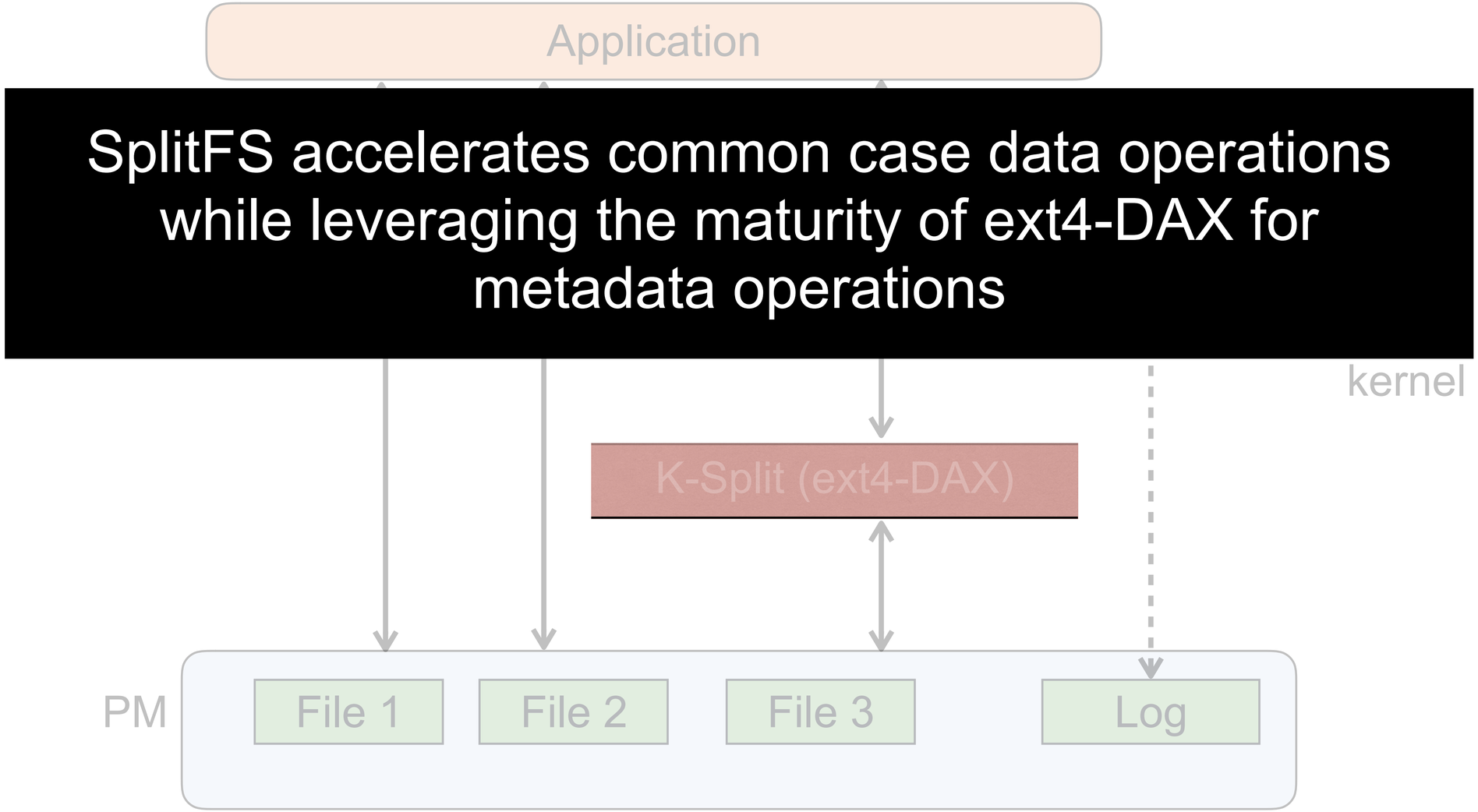
High-level Design



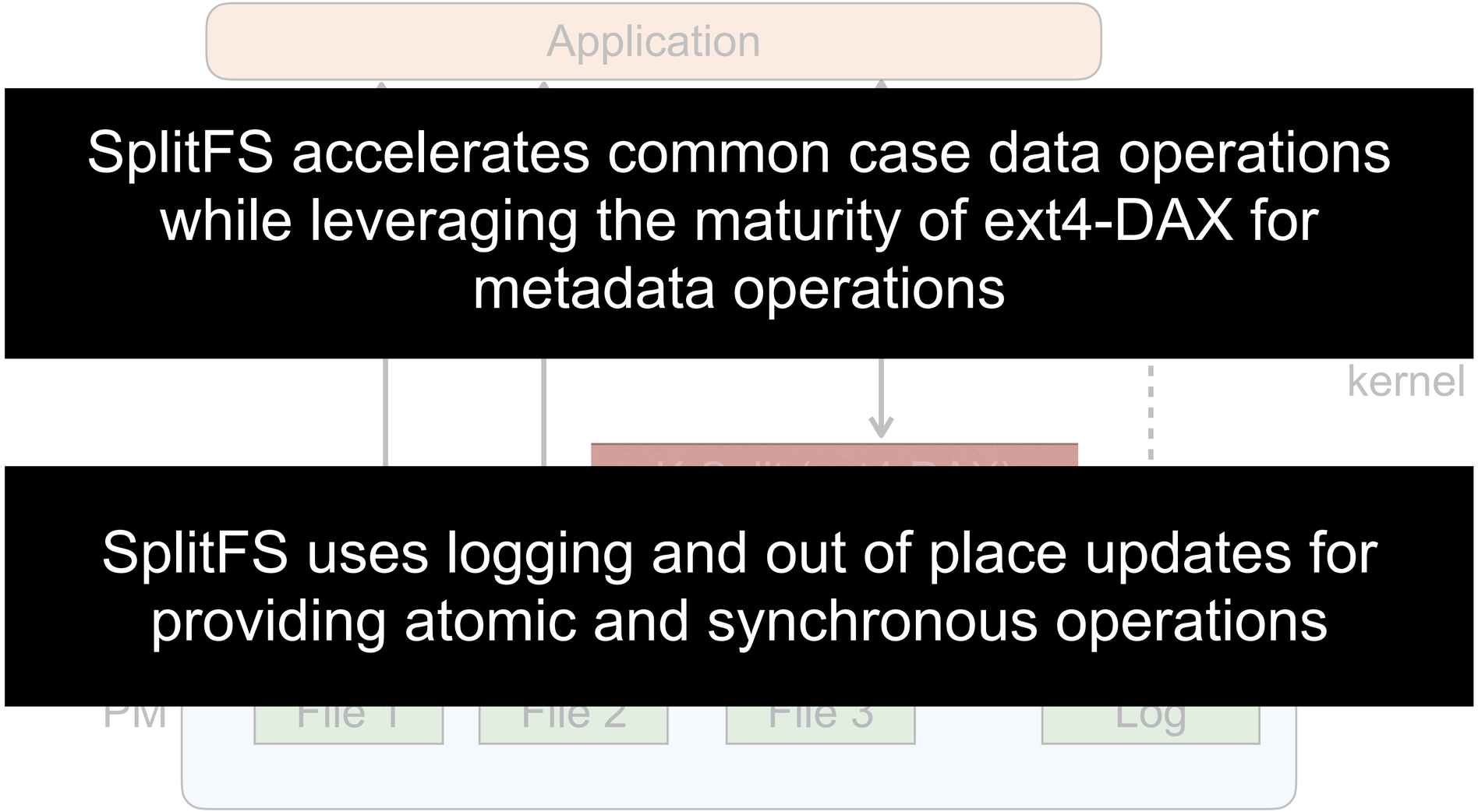
High-level Design



High-level Design



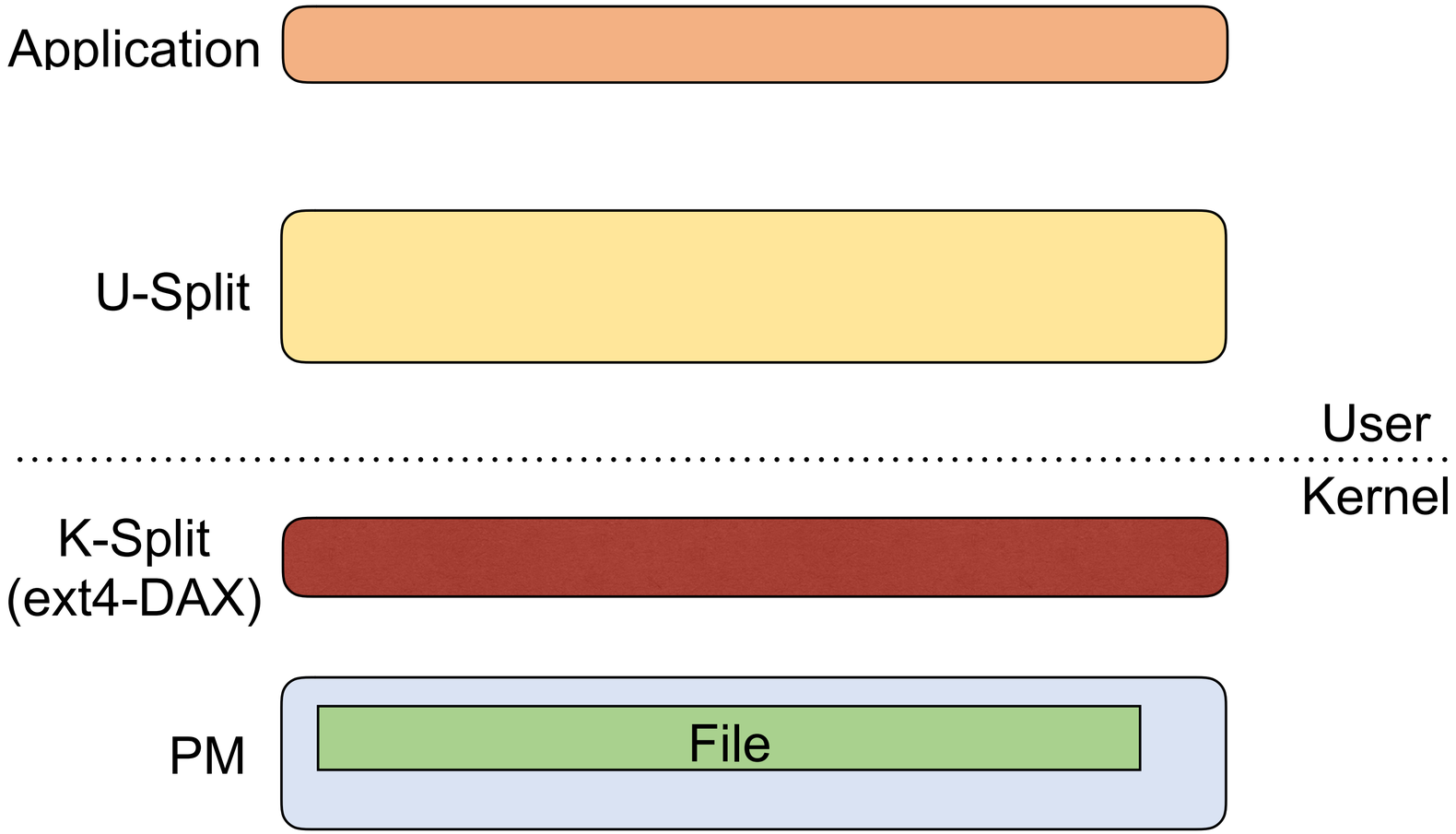
High-level Design



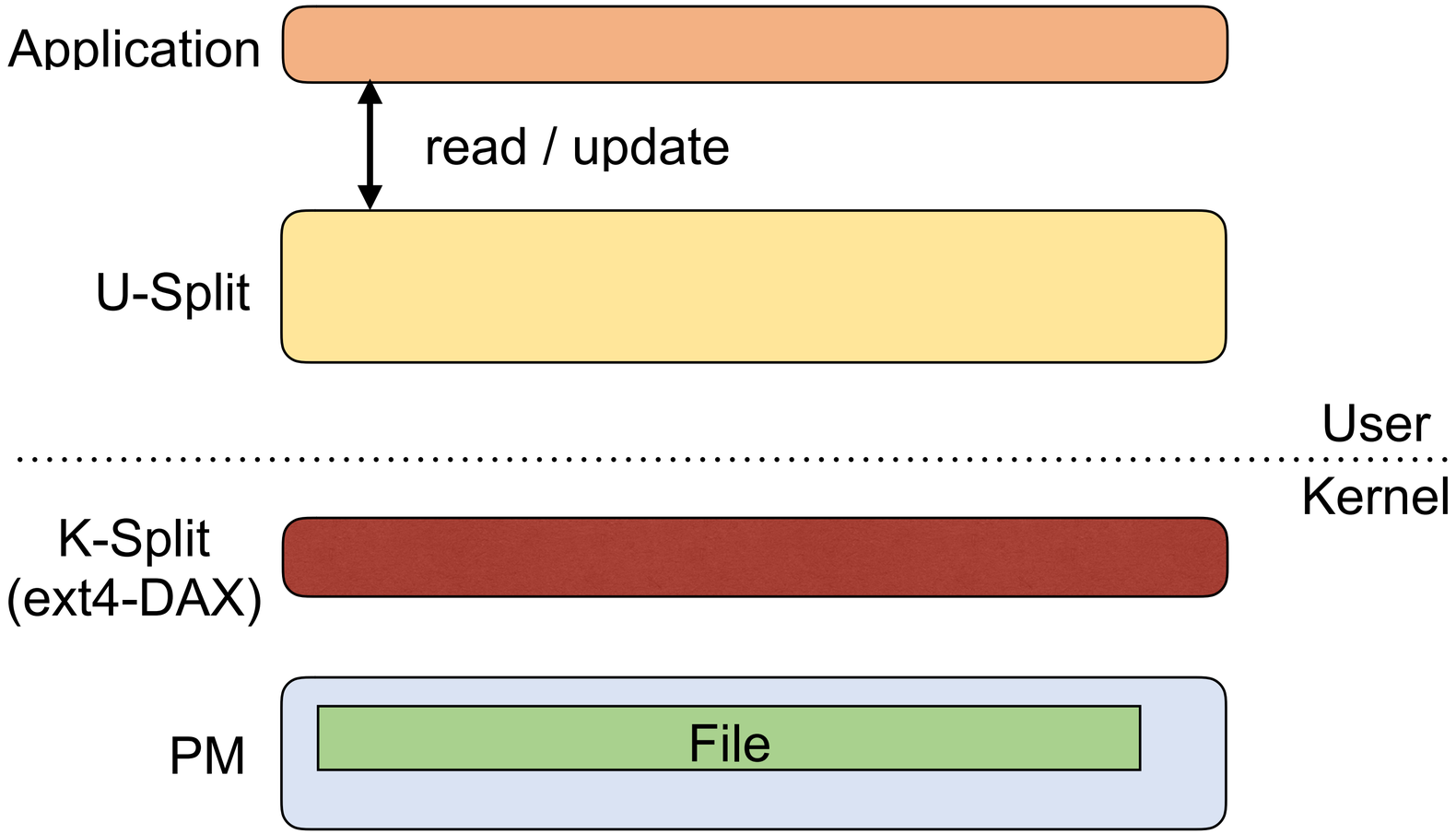
Outline

- Target usage scenario
- High-level design
- Handling data operations
  - Handling file reads and updates
  - Handling file appends
- Consistency guarantees
- Evaluation

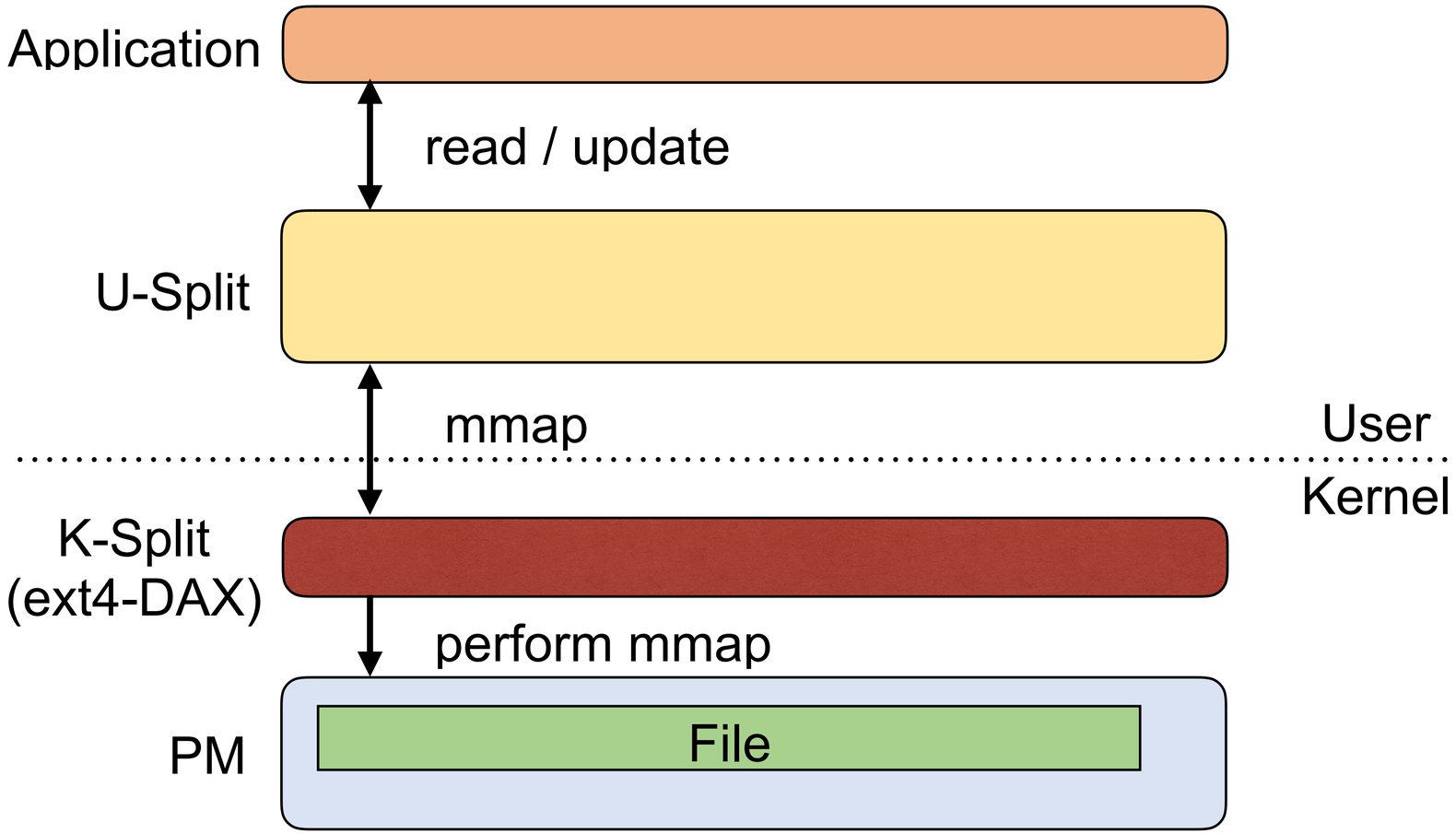
Handling reads and updates



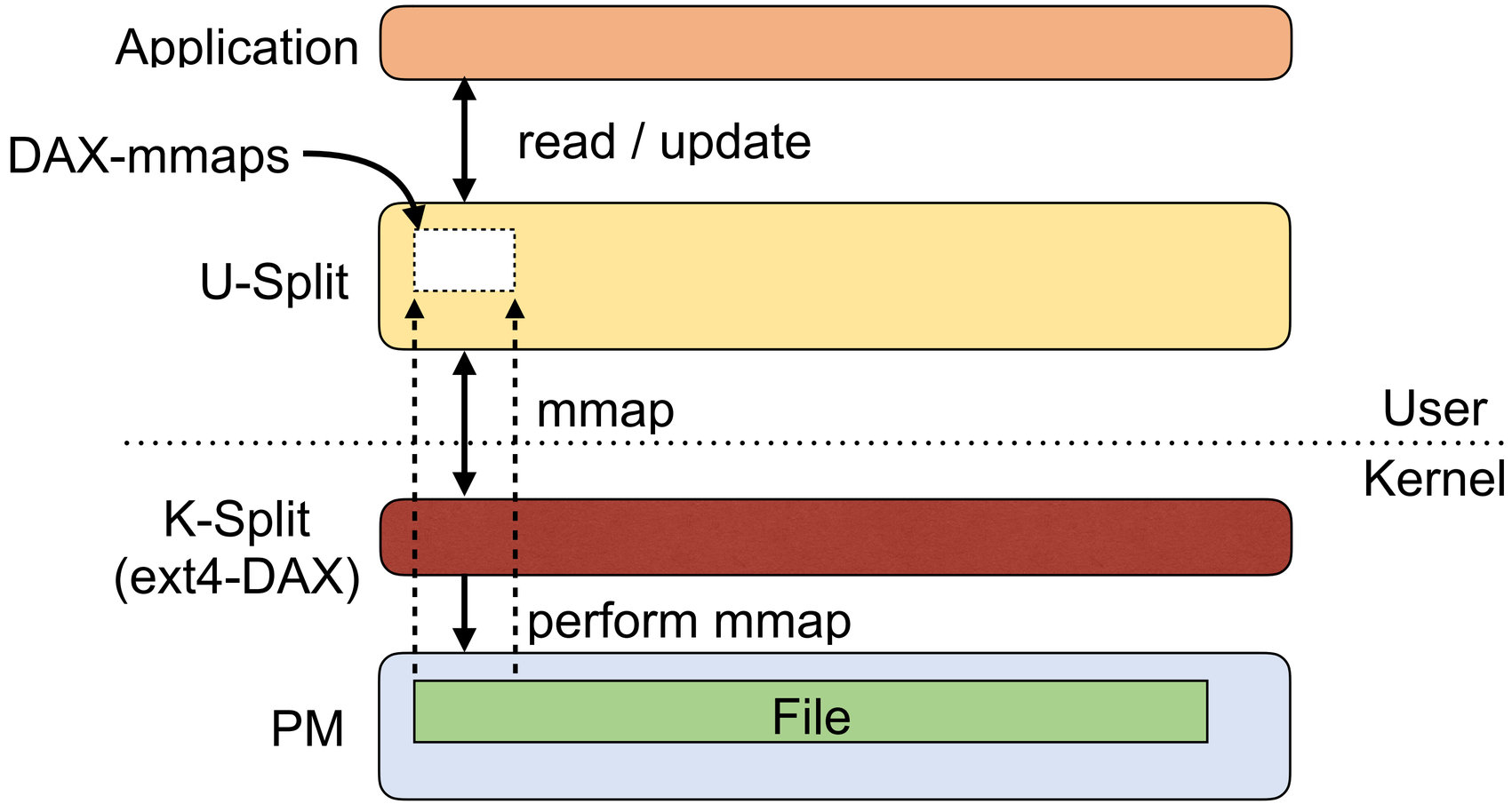
Handling reads and updates



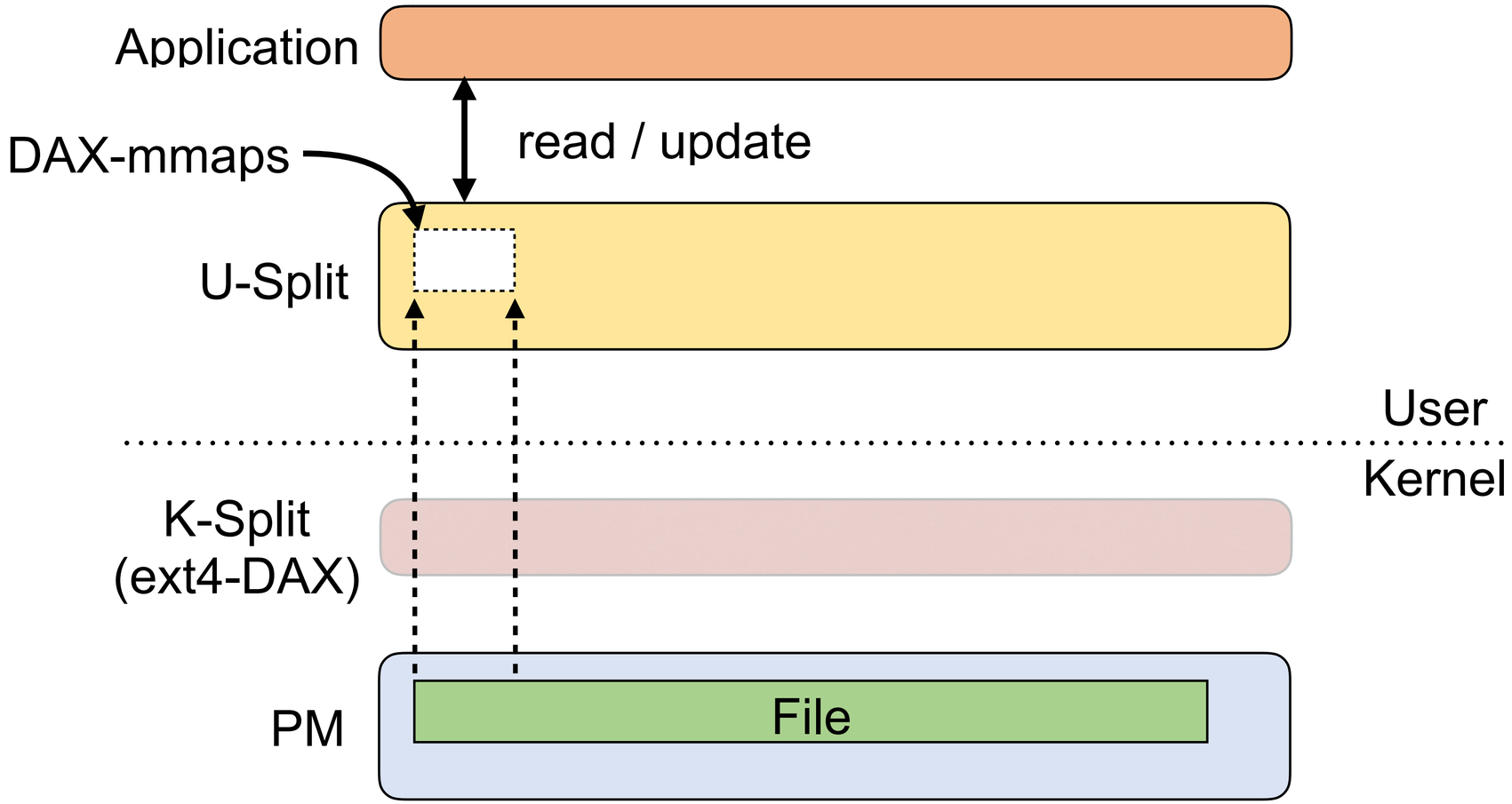
Handling reads and updates



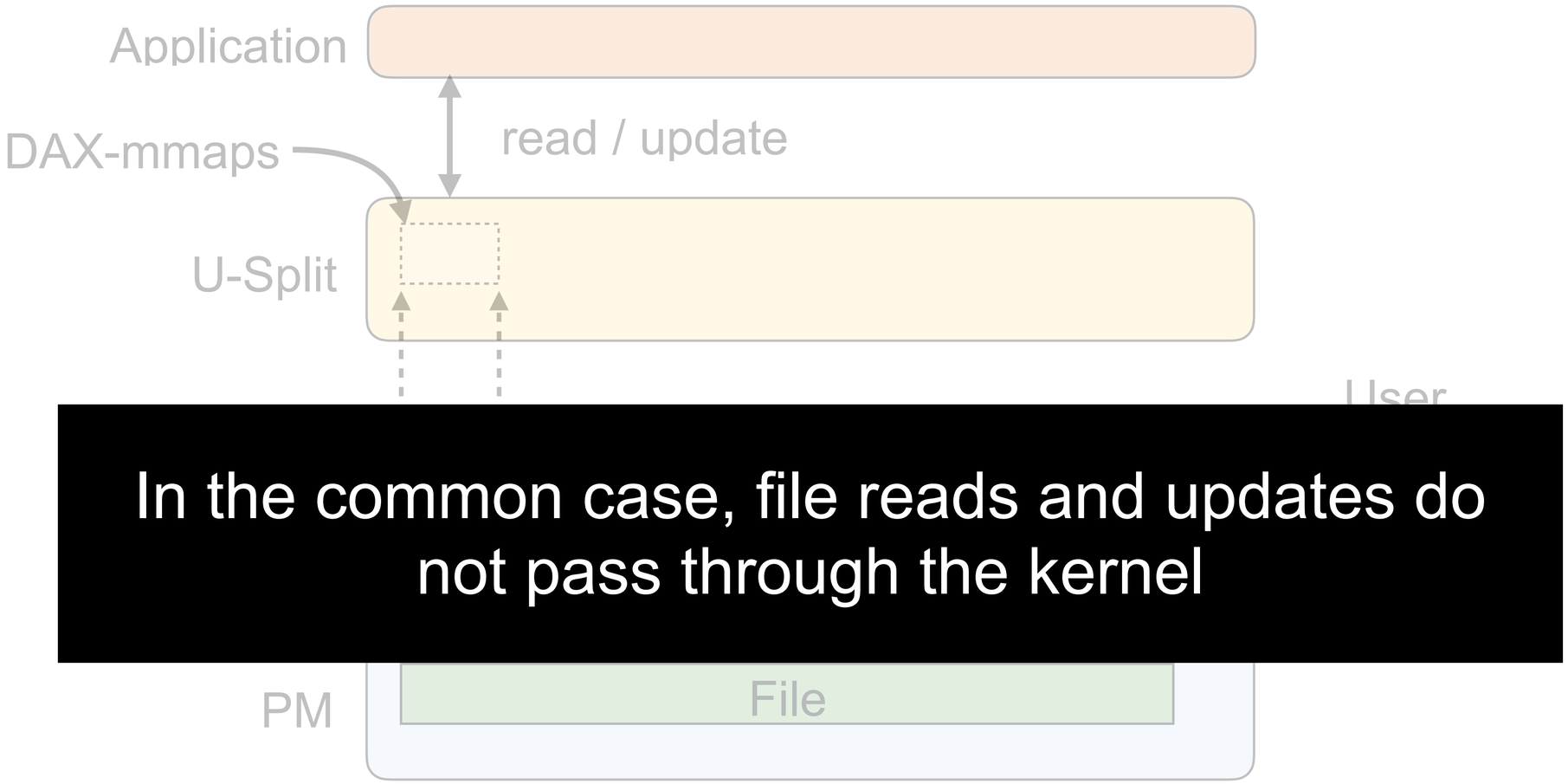
Handling reads and updates



Handling reads and updates



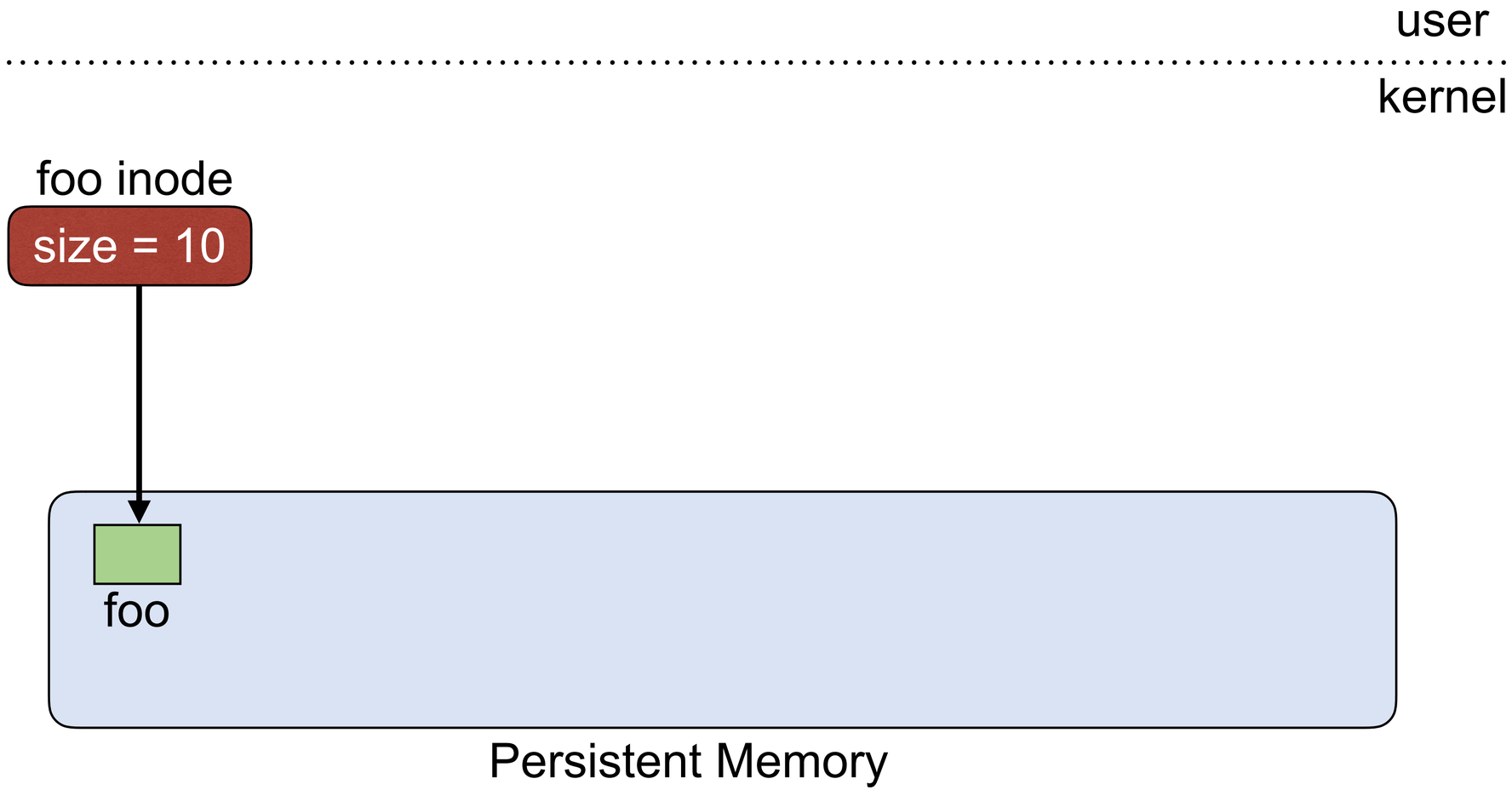
Handling reads and updates



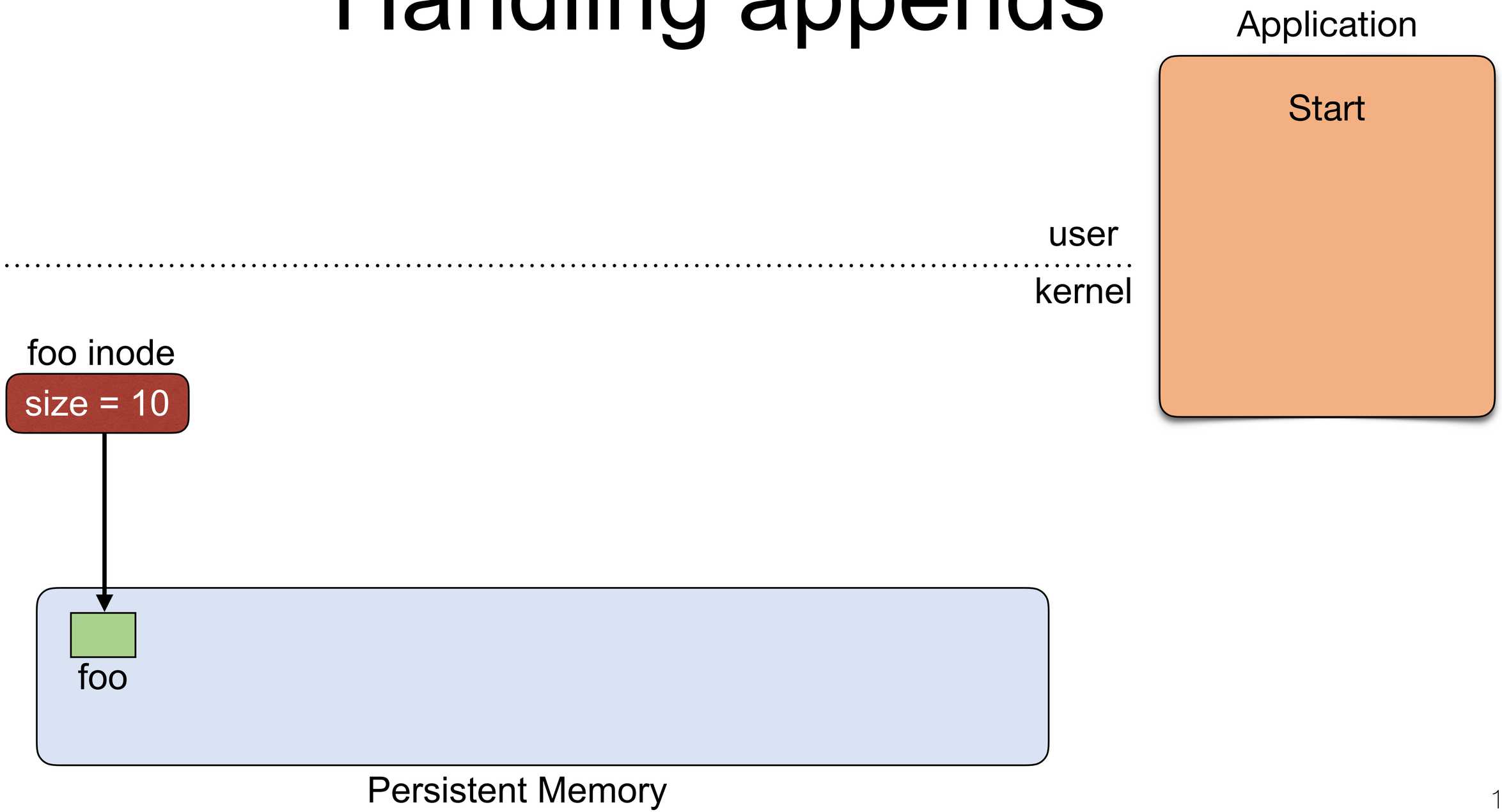
Outline

- Target usage scenario
- High-level design
- Handling data operations
  - Handling file reads and updates
  - Handling file appends
- Consistency guarantees
- Evaluation

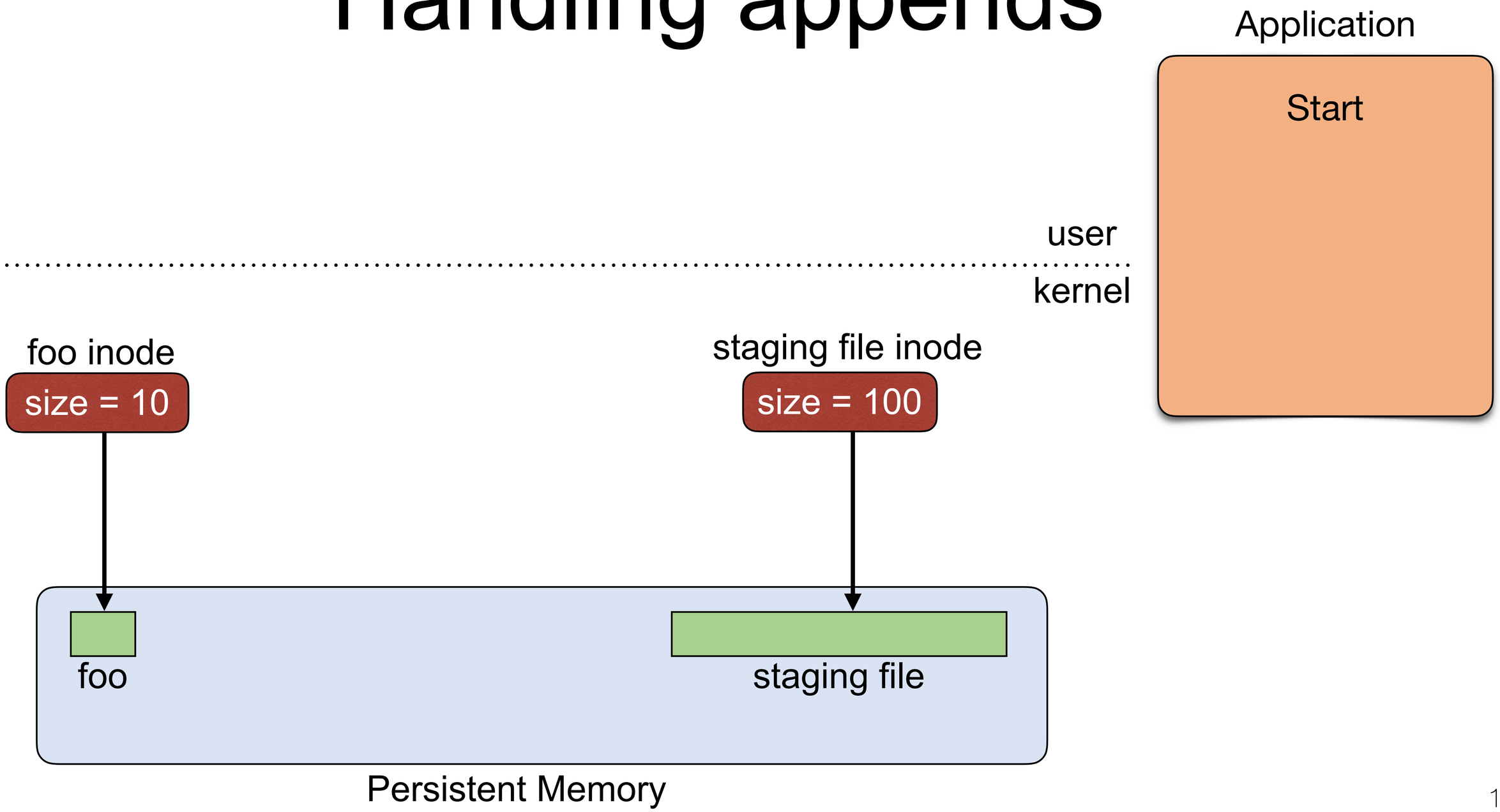
Handling appends



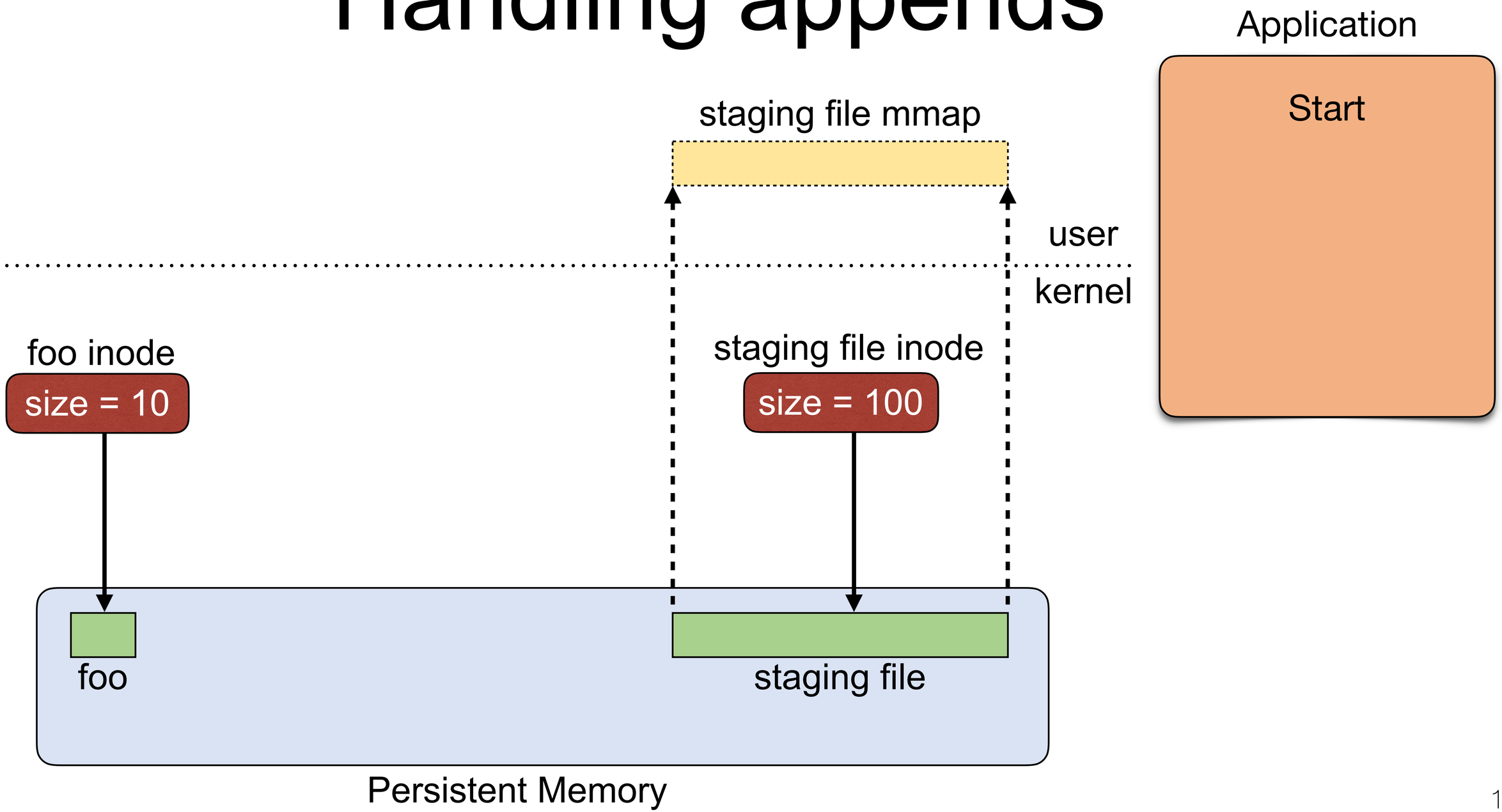
Handling appends



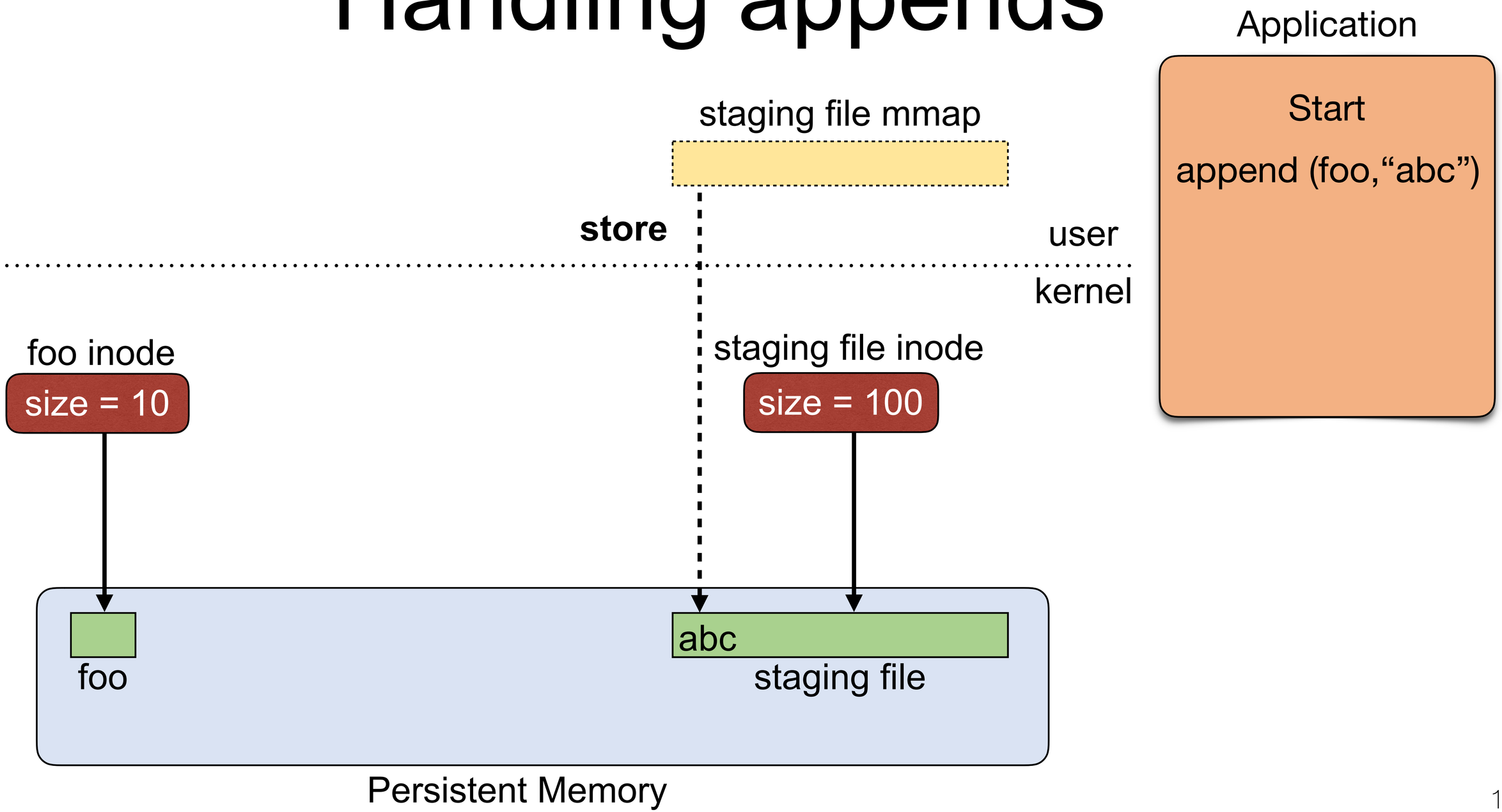
Handling appends



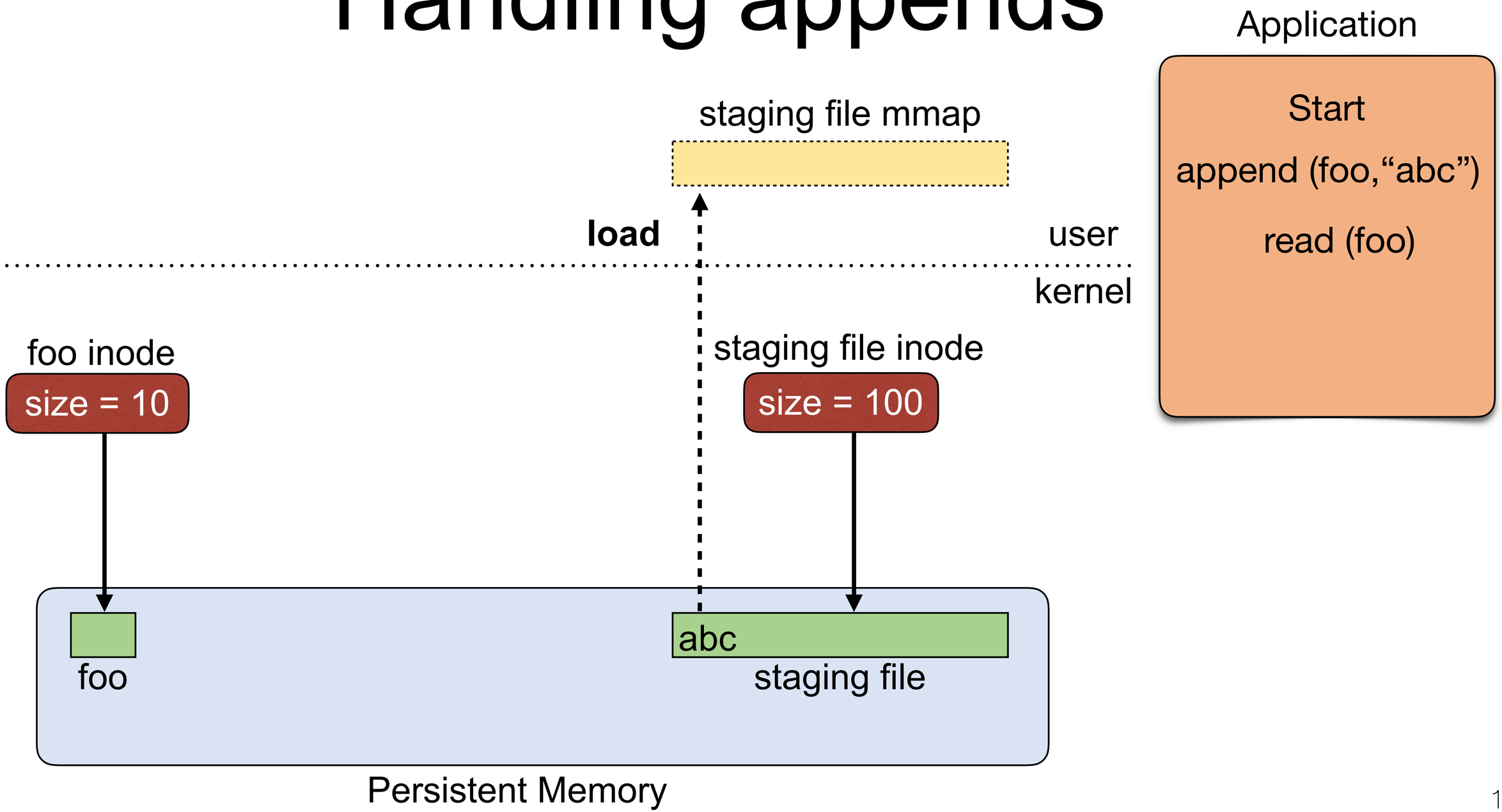
Handling appends



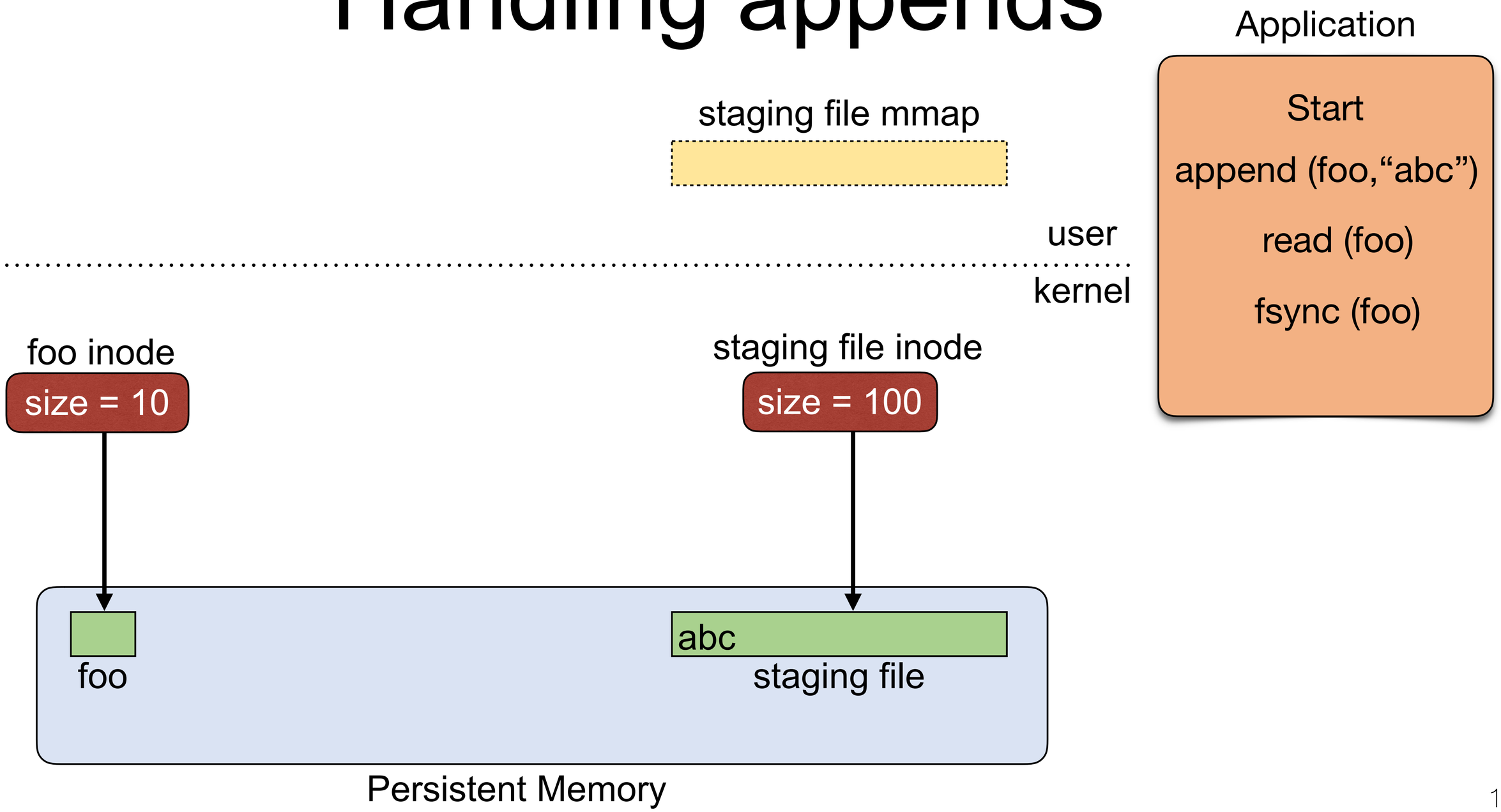
Handling appends



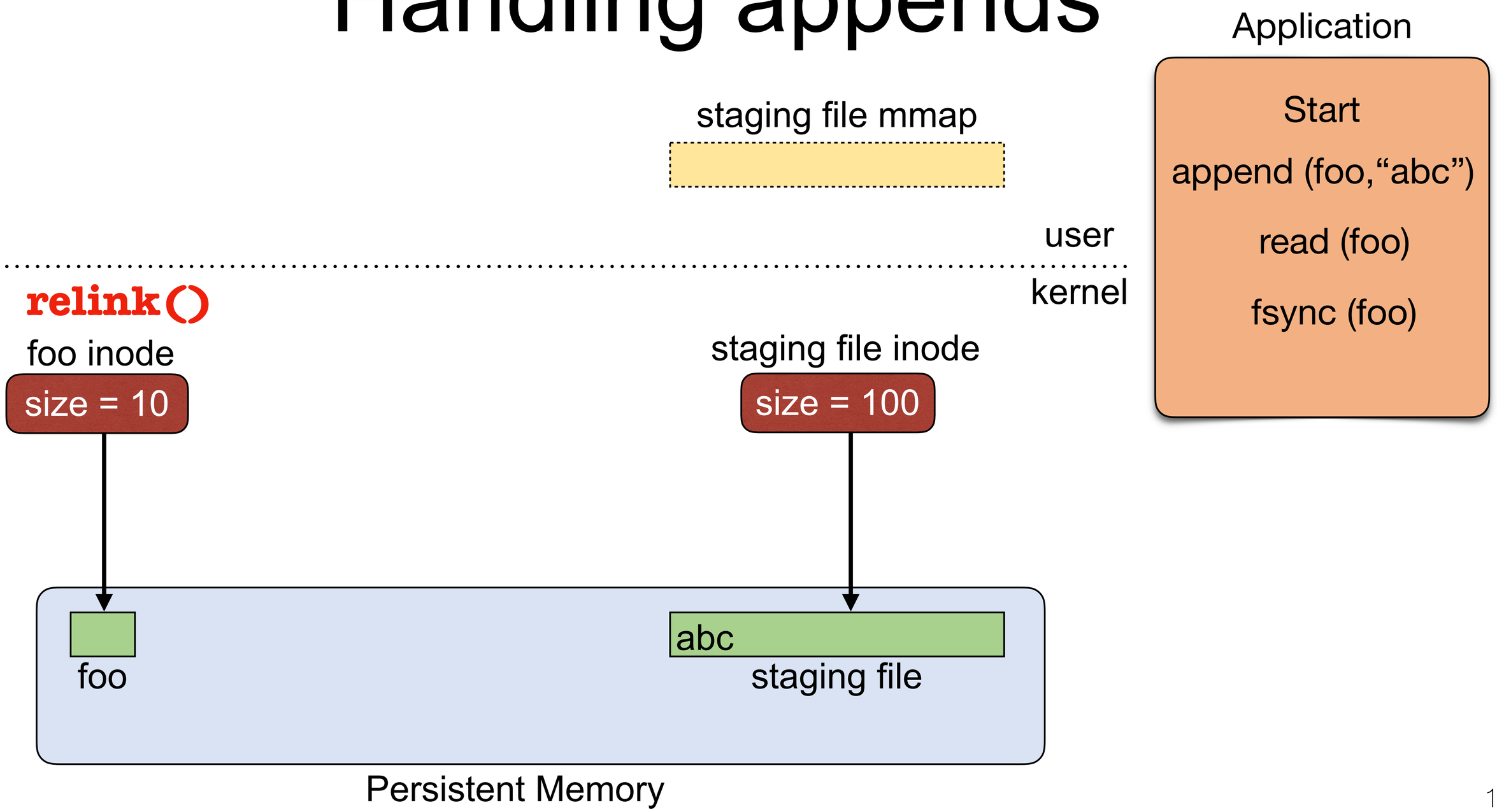
Handling appends



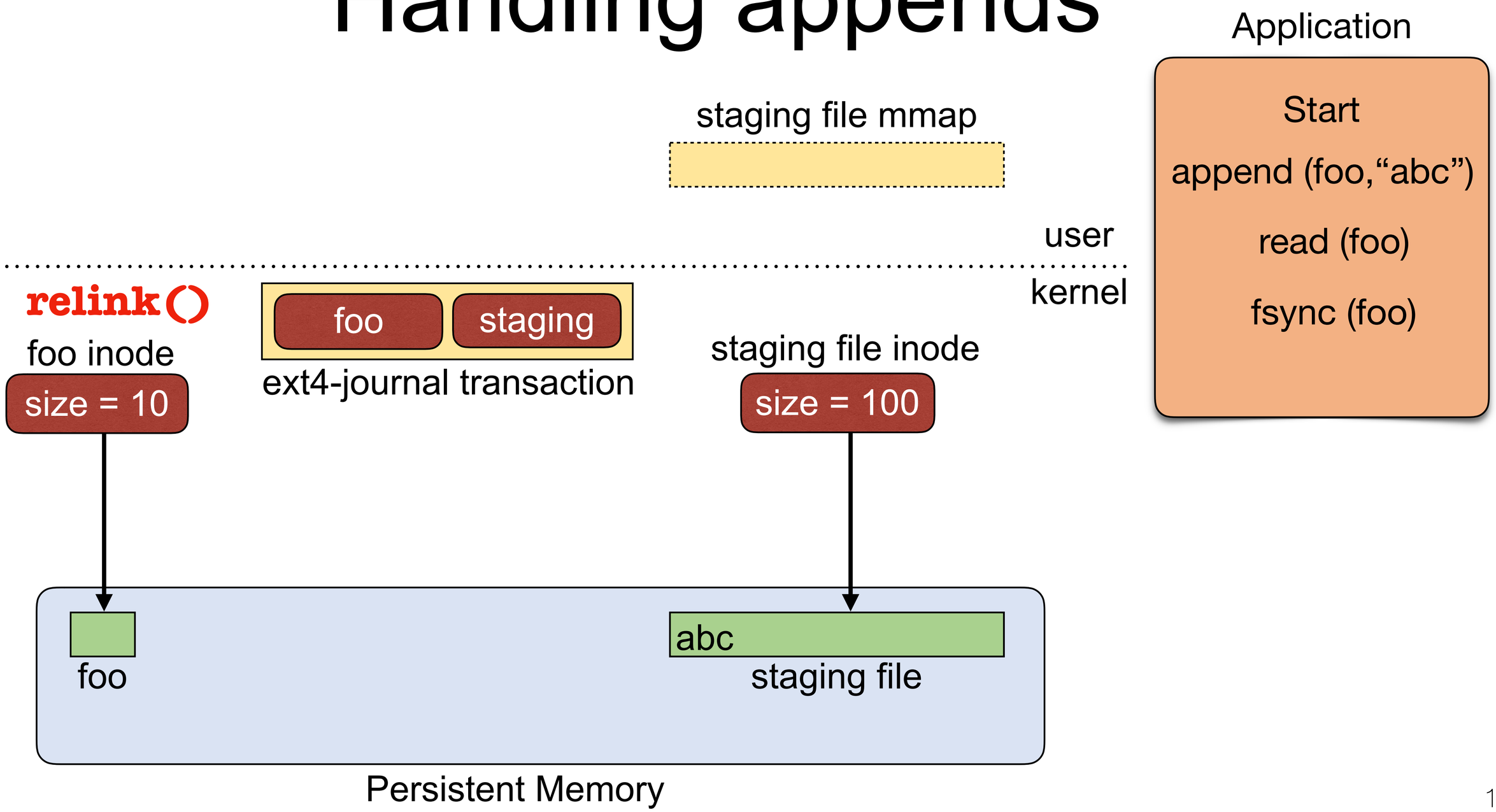
Handling appends



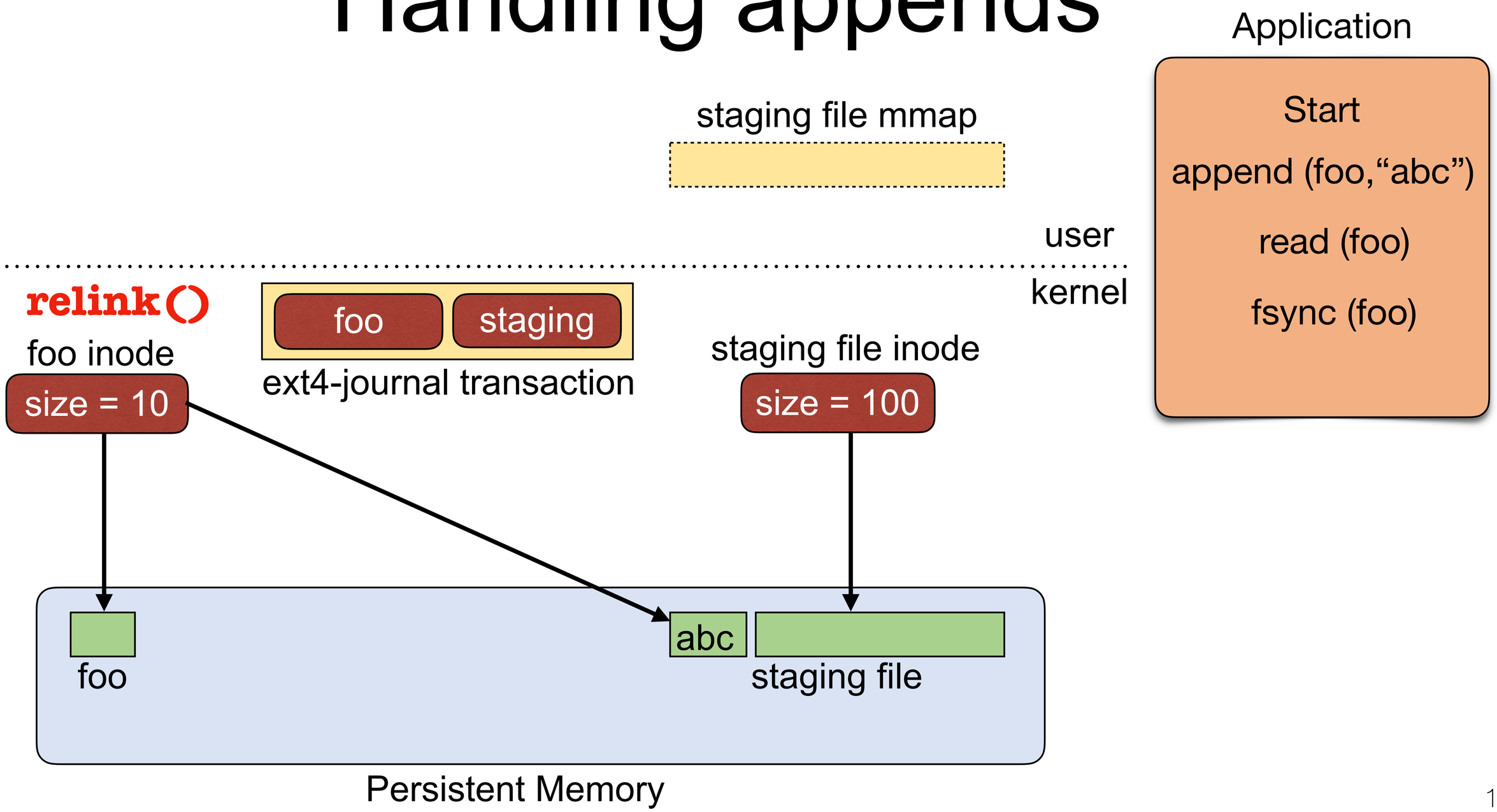
Handling appends



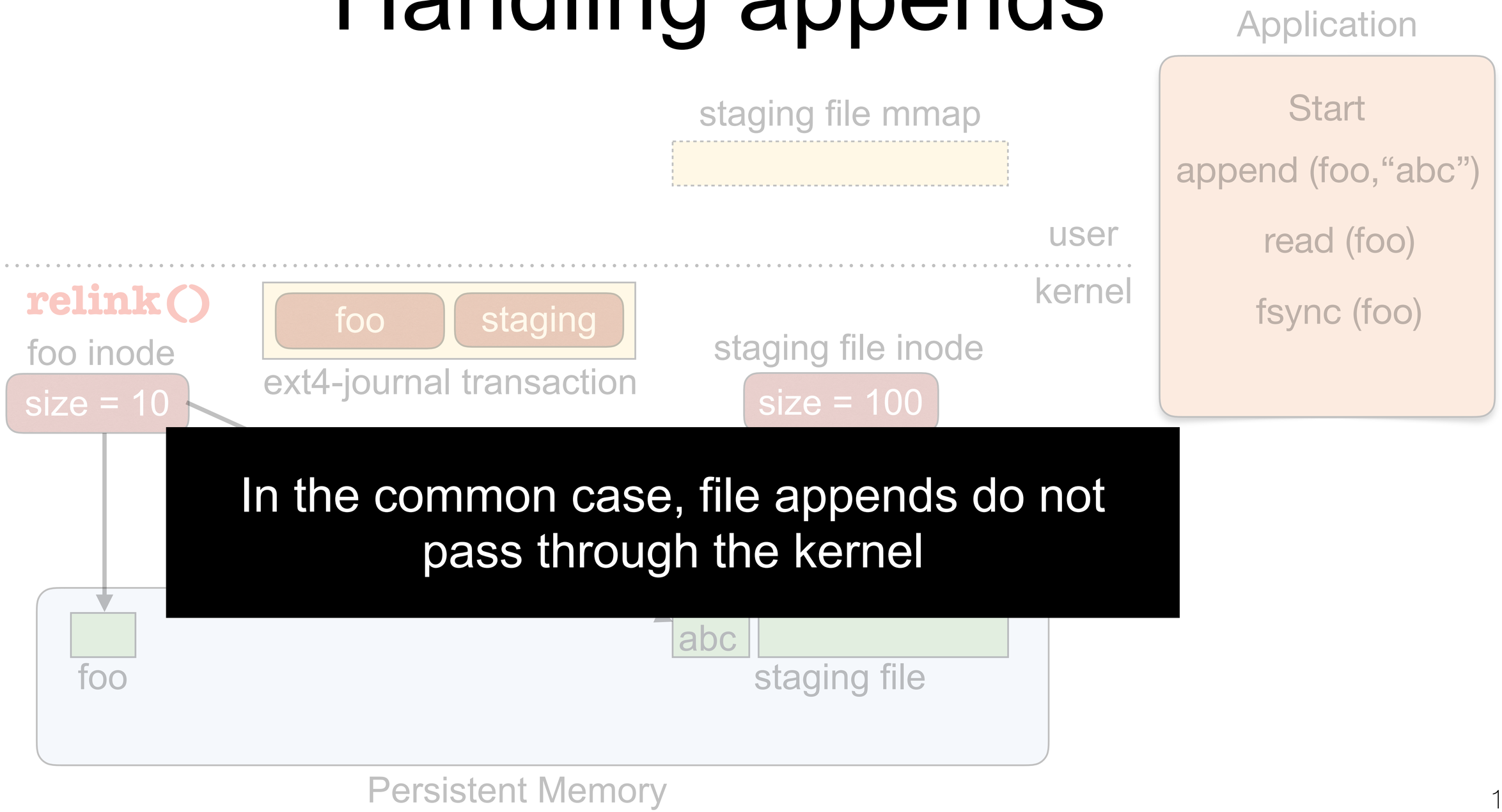
Handling appends



Handling appends



Handling appends



Outline

- Target usage scenario
- High-level design
- Handling data operations
- Consistency guarantees
- Evaluation

Consistency Guarantees

Mode	Metadata Atomicity	Synchronous Operations	Data Atomicity	File System
POSIX				ext4-DAX, SplitFS-POSIX
Sync				PMFS, SplitFS-Sync
Strict				NOVA, Strata, SplitFS-Strict

Consistency Guarantees

Mode	Metadata Atomicity	Synchronous Operations	Data Atomicity	File System
POSIX	✓			ext4-DAX, SplitFS-POSIX
Sync	✓			PMFS, SplitFS-Sync
Strict	✓			NOVA, Strata, SplitFS-Strict

Consistency Guarantees

Mode	Metadata Atomicity	Synchronous Operations	Data Atomicity	File System
POSIX	✓	✗		ext4-DAX, SplitFS-POSIX
Sync	✓	✓		PMFS, SplitFS-Sync
Strict	✓	✓		NOVA, Strata, SplitFS-Strict

Consistency Guarantees

Mode	Metadata Atomicity	Synchronous Operations	Data Atomicity	File System
POSIX	✓	✗	✗	ext4-DAX, SplitFS-POSIX
Sync	✓	✓	✗	PMFS, SplitFS-Sync
Strict	✓	✓	✓	NOVA, Strata, SplitFS-Strict

Consistency Guarantees

Mode	Metadata Atomicity	Synchronous Operations	Data Atomicity	File System
POSIX	✓	✗	✗	ext4-DAX, SplitFS-POSIX
Sy				IFS, S-Sync
Strict	✓	✓	✓	NOVA, Strata, SplitFS-Strict

Optimized logging is used in order to provide stronger guarantees in sync and strict modes

Optimized logging

Optimized logging

SplitFS employs a per-application log in sync and strict mode, which logs every logical operation

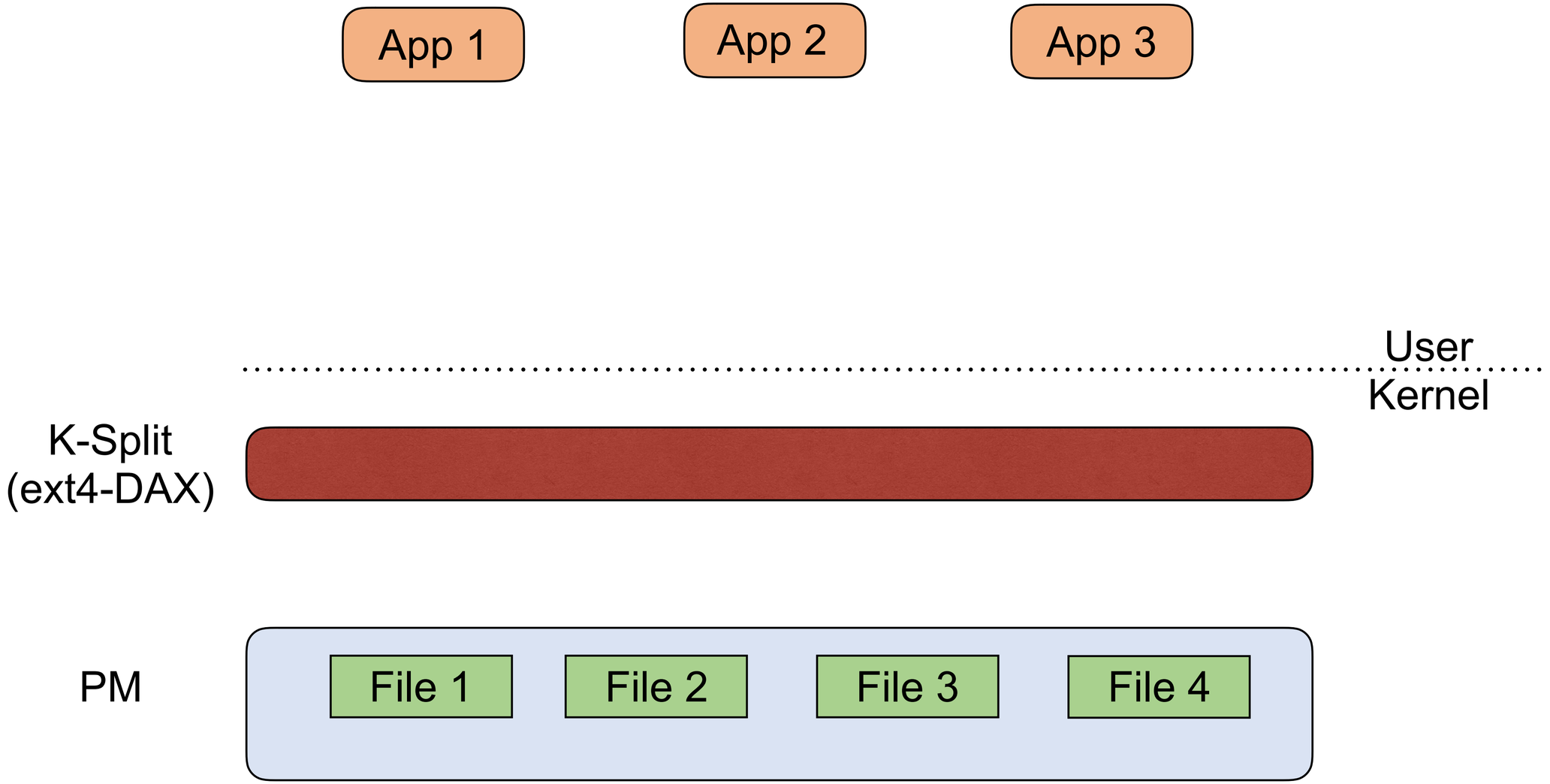
Optimized logging

SplitFS employs a per-application log in sync and strict mode, which logs every logical operation

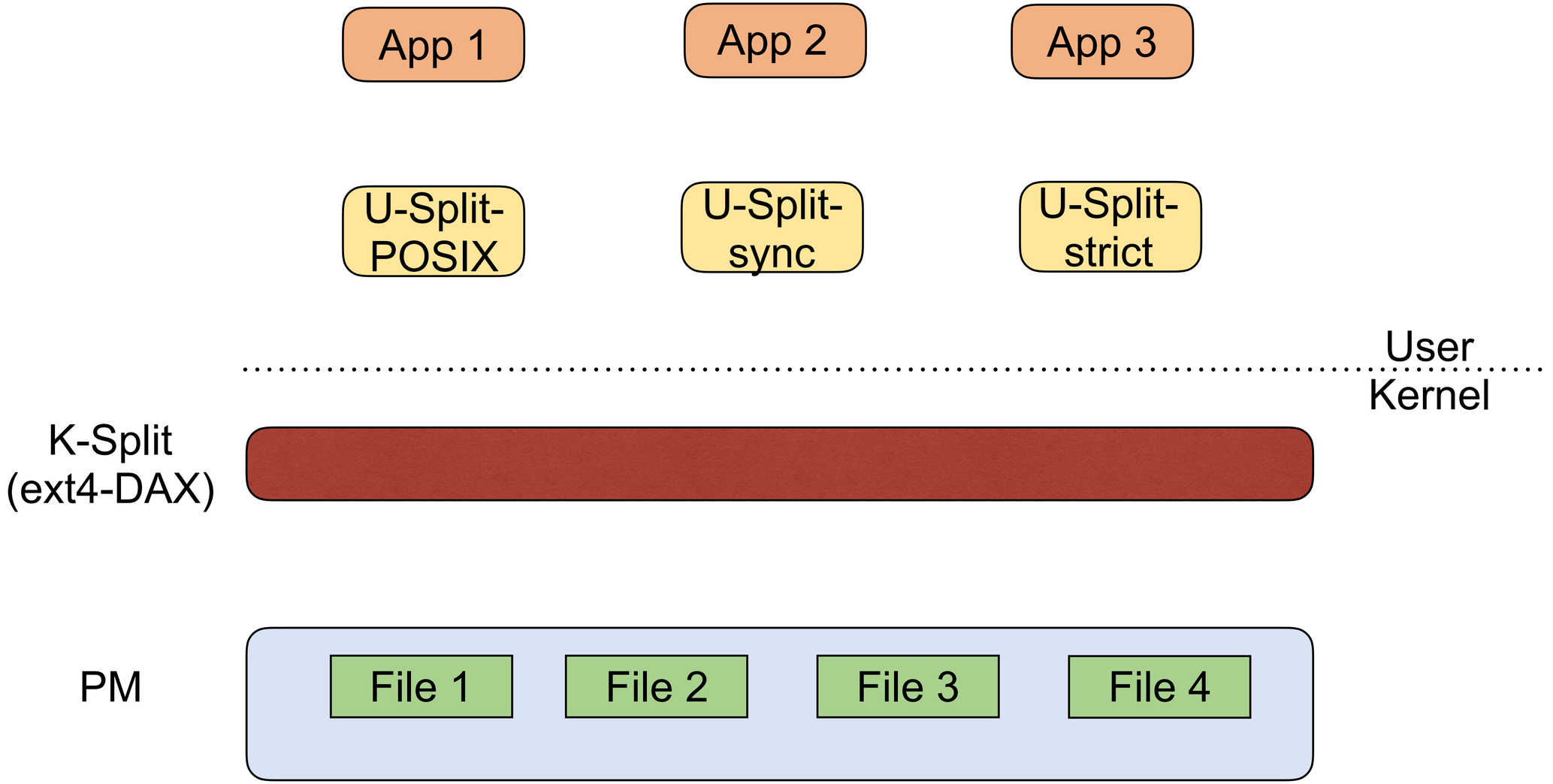
In the common case

- Each log entry fits in one cache line
- Persisted using a single non-temporal store and sfence instruction

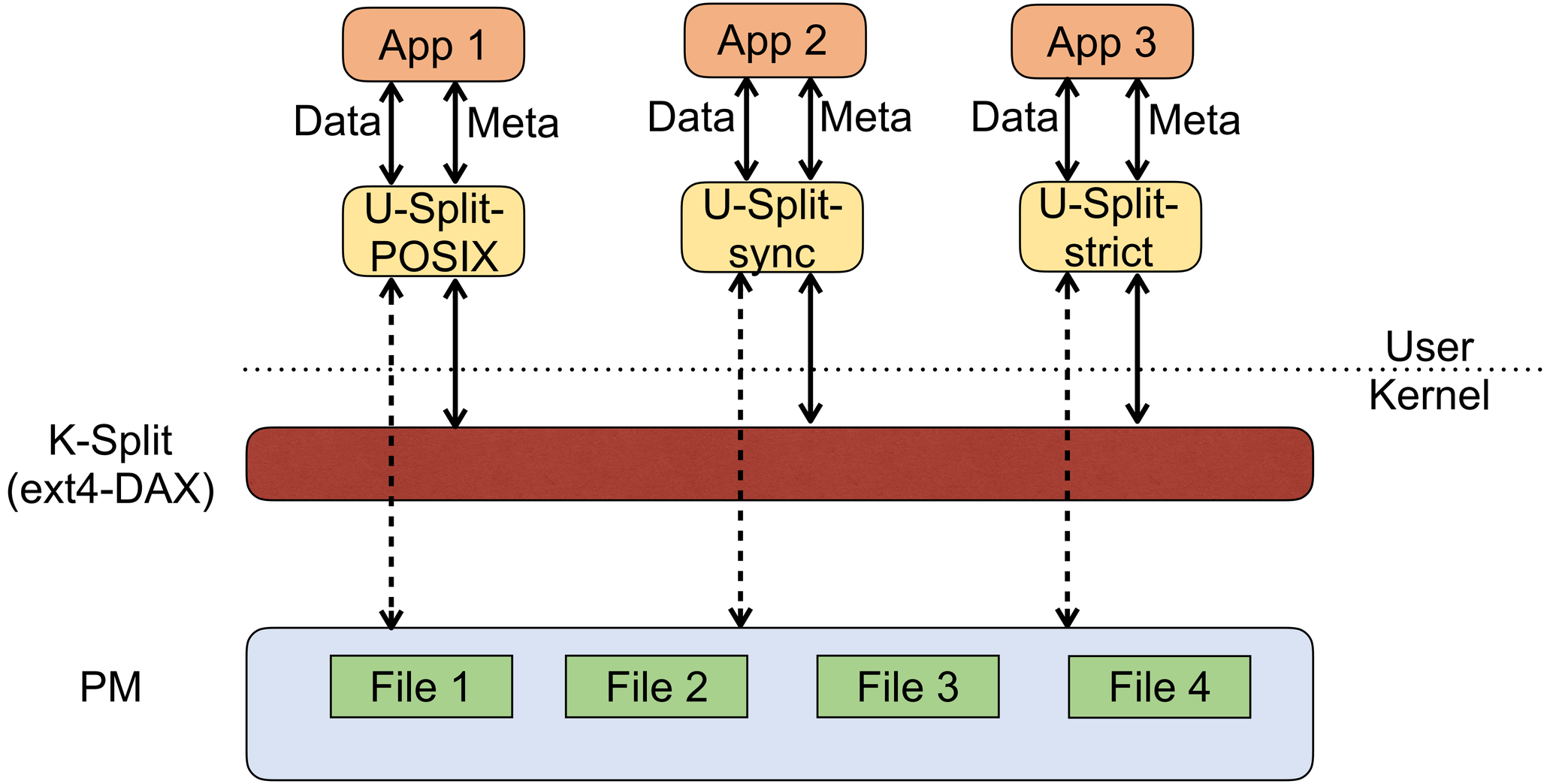
Flexible SplitFS



Flexible SplitFS



Flexible SplitFS



Visibility

Visibility

When are updates from one application visible to another?

Visibility

When are updates from one application visible to another?

- All metadata operations are immediately visible to all other processes

Visibility

When are updates from one application visible to another?

- All metadata operations are immediately visible to all other processes
- Writes are visible to all other processes on subsequent fsync()

Visibility

When are updates from one application visible to another?

- All metadata operations are immediately visible to all other processes
- Writes are visible to all other processes on subsequent fsync()
- Memory mapped files have the same visibility guarantees as that of ext4-DAX

SplitFS Techniques

Technique	Benefit

SplitFS Techniques

Technique	Benefit
SplitFS Architecture	Low-overhead data operations, Correct metadata operations

SplitFS Techniques

Technique	Benefit
SplitFS Architecture	Low-overhead data operations, Correct metadata operations
Staging + Relink	Optimized appends, No data copy

SplitFS Techniques

Technique	Benefit
SplitFS Architecture	Low-overhead data operations, Correct metadata operations
Staging + Relink	Optimized appends, No data copy
Optimized Logging + out-of-place writes	Stronger guarantees

Outline

- Target usage scenario
- High-level design
- Handling data operations
- Consistency guarantees
- Evaluation

Evaluation

Evaluation

Setup:

- 2-socket 96-core machine with 32 MB LLC
- 768 GB Intel Optane DC PMM, 378 GB DRAM

Evaluation

Setup:

- 2-socket 96-core machine with 32 MB LLC
- 768 GB Intel Optane DC PMM, 378 GB DRAM

File systems compared:

- ext4-DAX, PMFS, NOVA, Strata

Evaluation

Setup:

- 2-socket 96-core machine with 32 MB LLC
- 768 GB Intel Optane DC PMM, 378 GB DRAM

File systems compared:

- ext4-DAX, PMFS, **NOVA**, Strata

Does SplitFS reduce software overhead compared to other file systems?

How does SplitFS perform on data intensive workloads?

How does SplitFS perform on metadata intensive workloads?

Does SplitFS reduce software overhead compared to other file systems?

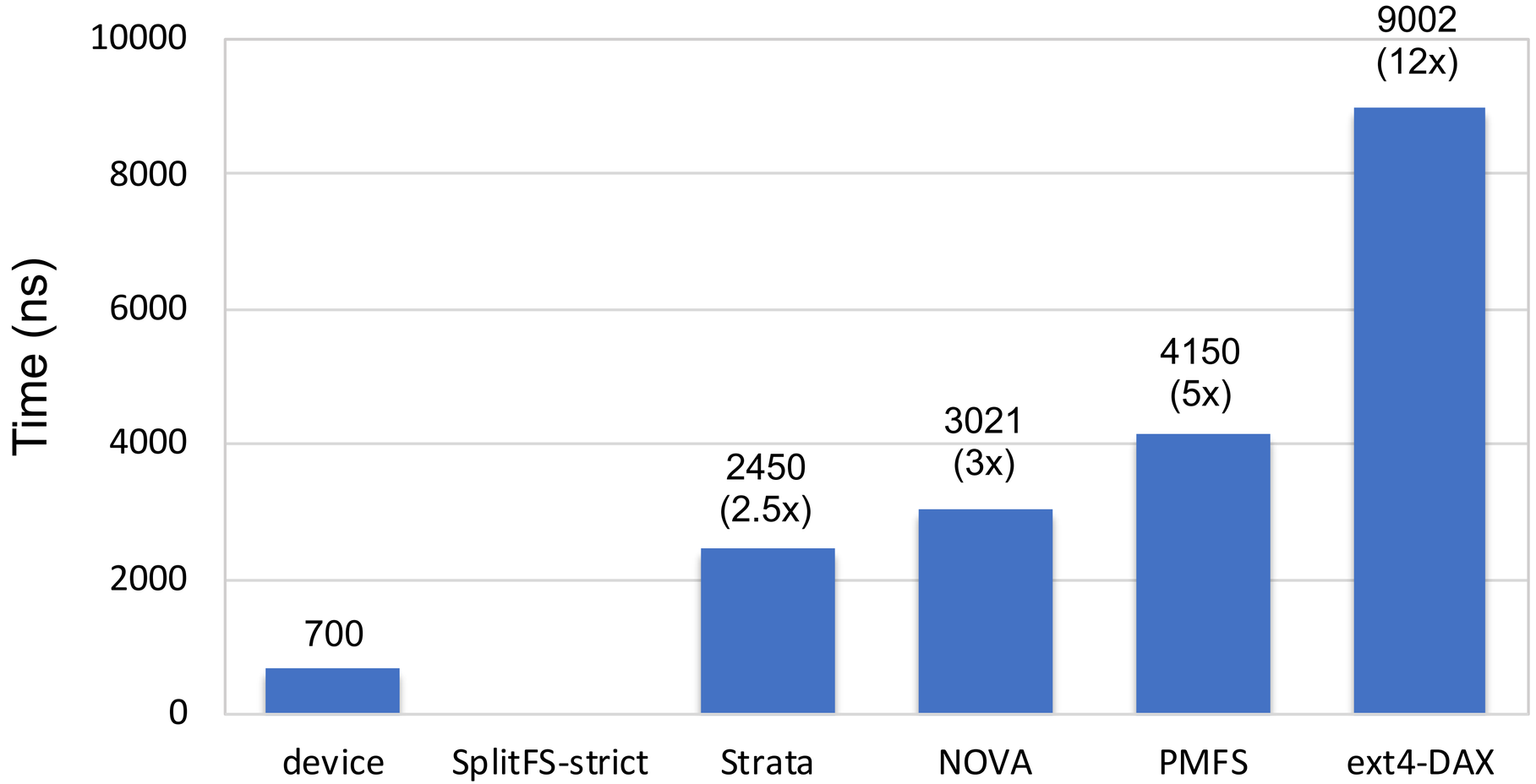
How does SplitFS perform on data intensive workloads?

How does SplitFS perform on metadata intensive workloads?

- < 15% overhead for metadata intensive workloads

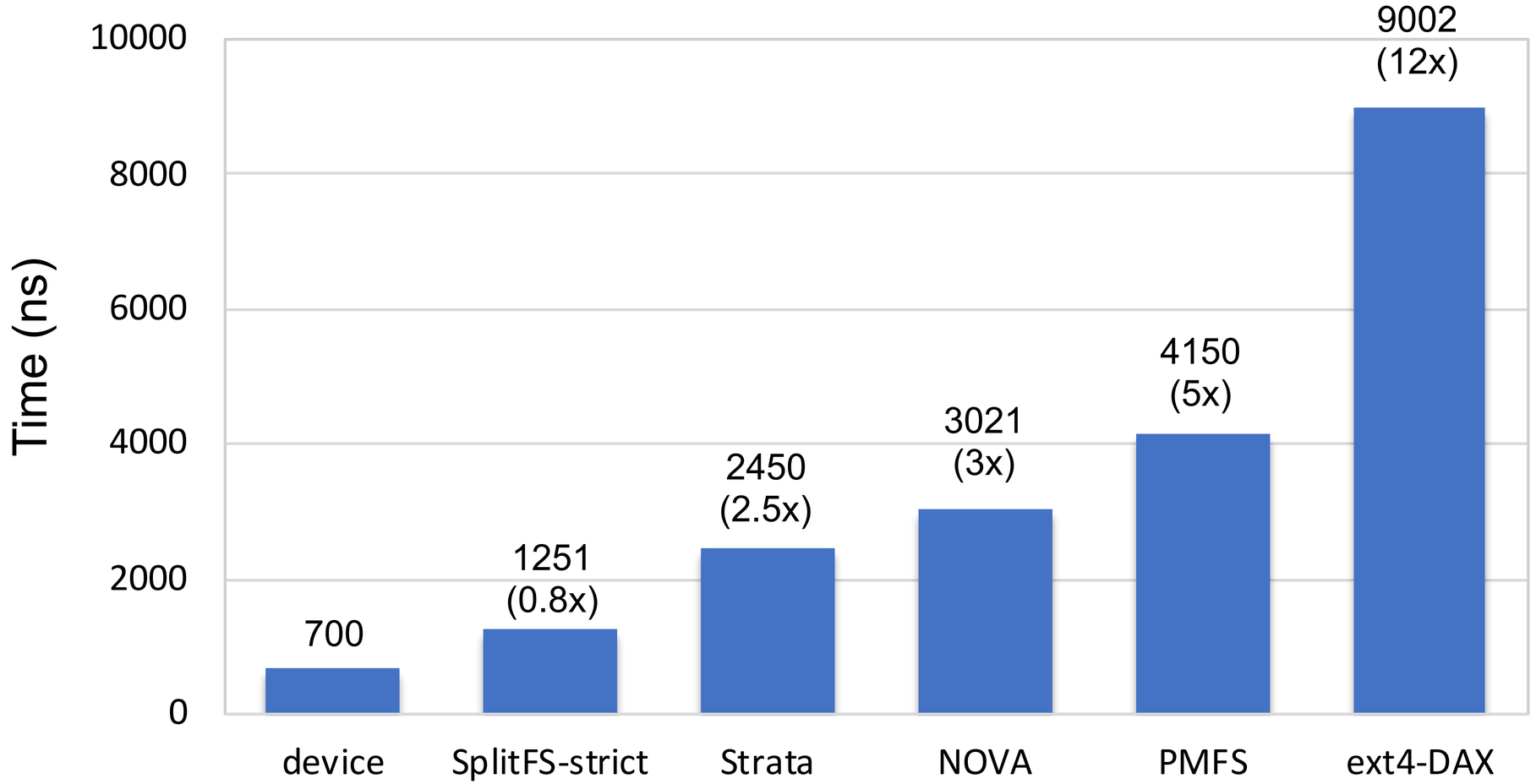
Software Overhead of SplitFS

- Append 4KB data to a file
- Time taken to copy user data to PM: ~700 ns



Software Overhead of SplitFS

- Append 4KB data to a file
- Time taken to copy user data to PM: ~700 ns



Workloads

Microbenchmarks

Seq reads

Seq writes

Rand reads

Rand writes

Appends

Data intensive

YCSB on LevelDB

Redis

TPCC on SQLite

Metadata intensive

Tar

Git

Rsync

Workloads

Microbenchmarks

Seq reads

Seq writes

Rand reads

Rand writes

Appends

Data intensive

YCSB on LevelDB

Redis

TPCC on SQLite

Metadata intensive

Tar

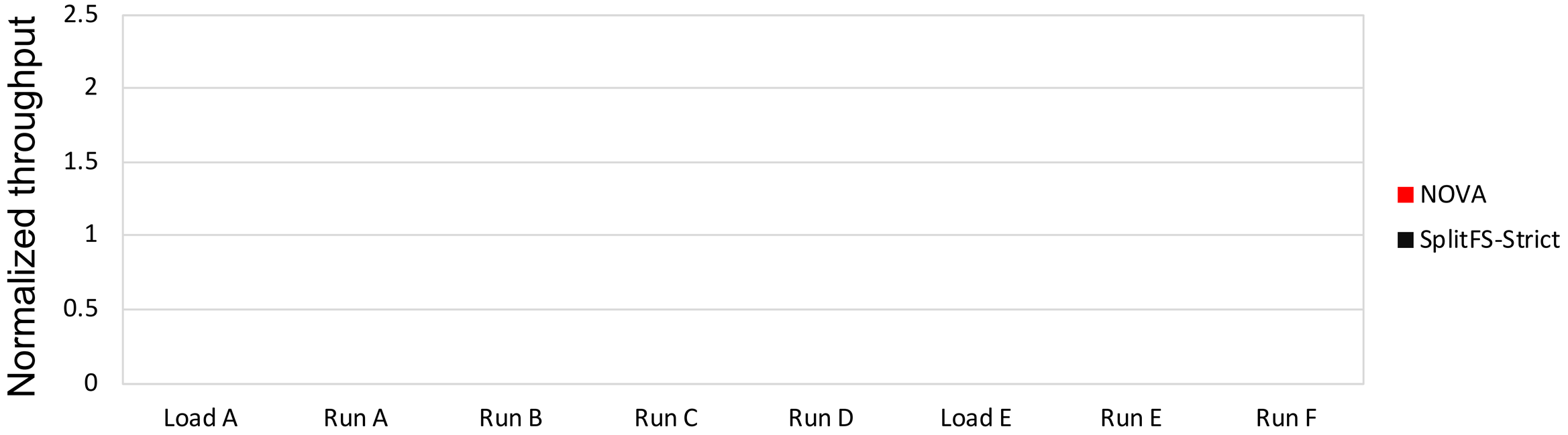
Git

Rsync

YCSB on LevelDB

Yahoo! Cloud Serving Benchmark - Industry standard macro-benchmark

Insert 5M keys. Run 5M operations. Key size = 16 bytes. Value size = 1K



Load A - 100% writes

Run A - 50% reads, 50% writes

Run B - 95% reads, 5% writes

Run C - 100% reads

Run D - 95% reads (latest), 5% writes

Load E - 100% writes

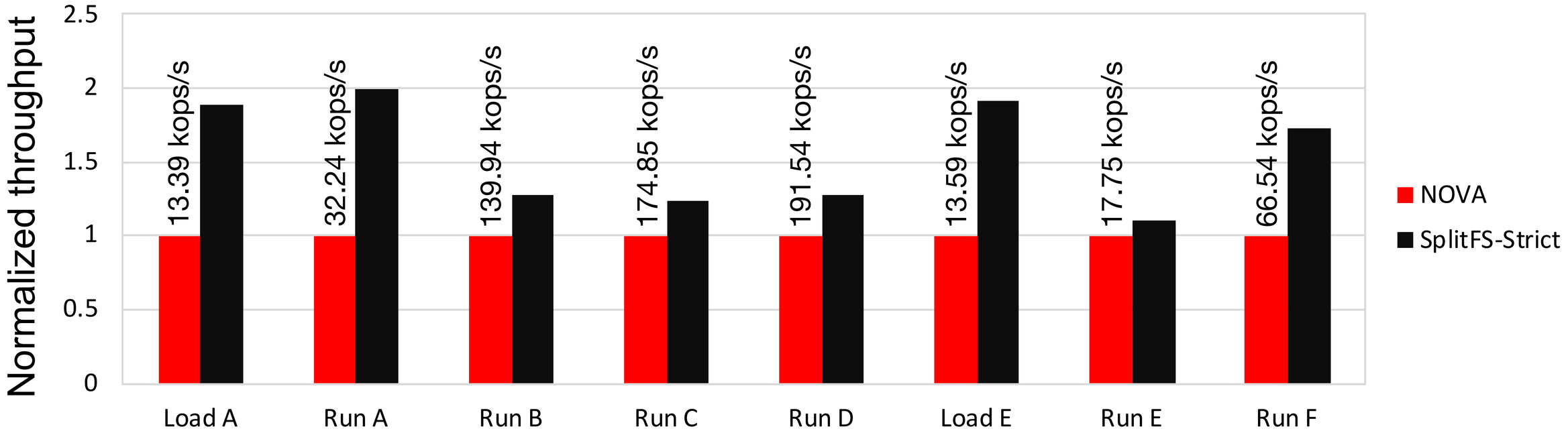
Run E - 95% range queries, 5% writes

Run F - 50% reads, 50% read-modify-writes

YCSB on LevelDB

Yahoo! Cloud Serving Benchmark - Industry standard macro-benchmark

Insert 5M keys. Run 5M operations. Key size = 16 bytes. Value size = 1K



Load A - 100% writes

Run A - 50% reads, 50% writes

Run B - 95% reads, 5% writes

Run C - 100% reads

Run D - 95% reads (latest), 5% writes

Load E - 100% writes

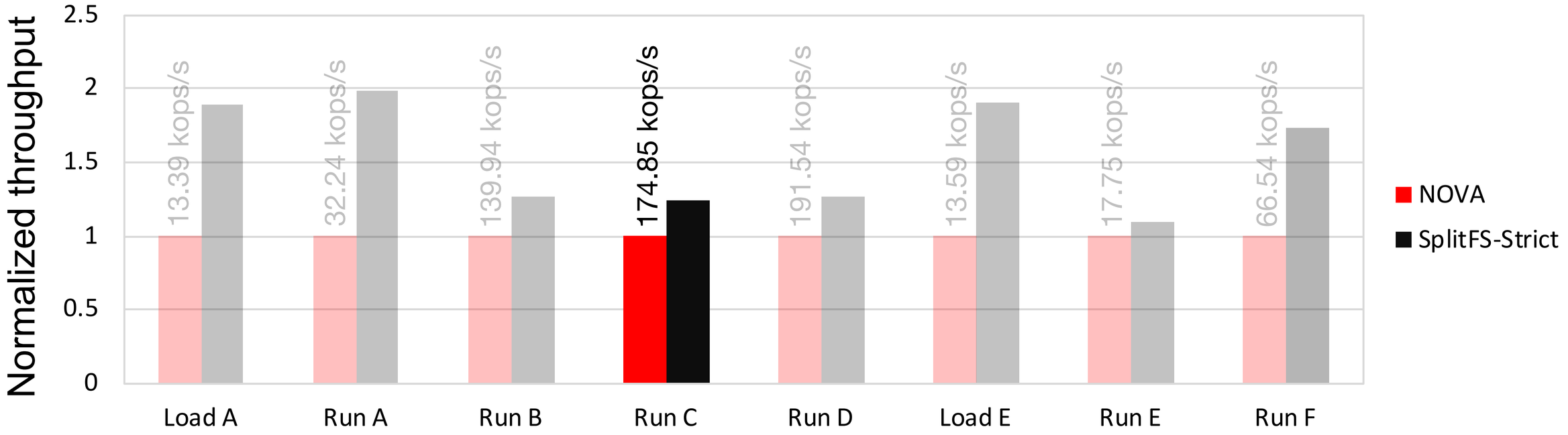
Run E - 95% range queries, 5% writes

Run F - 50% reads, 50% read-modify-writes

YCSB on LevelDB

Yahoo! Cloud Serving Benchmark - Industry standard macro-benchmark

Insert 5M keys. Run 5M operations. Key size = 16 bytes. Value size = 1K



Load A - 100% writes

Run A - 50% reads, 50% writes

Run B - 95% reads, 5% writes

Run C - 100% reads

Run D - 95% reads (latest), 5% writes

Load E - 100% writes

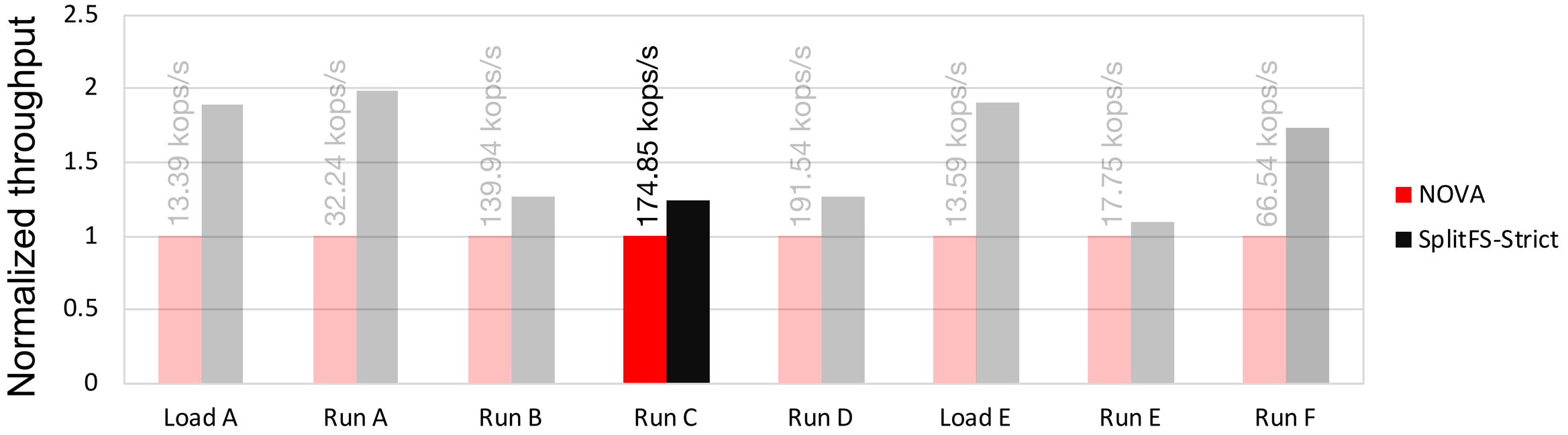
Run E - 95% range queries, 5% writes

Run F - 50% reads, 50% read-modify-writes

YCSB on LevelDB

Yahoo! Cloud Serving Benchmark - Industry standard macro-benchmark

Insert 5M keys. Run 5M operations. Key size = 16 bytes. Value size = 1K

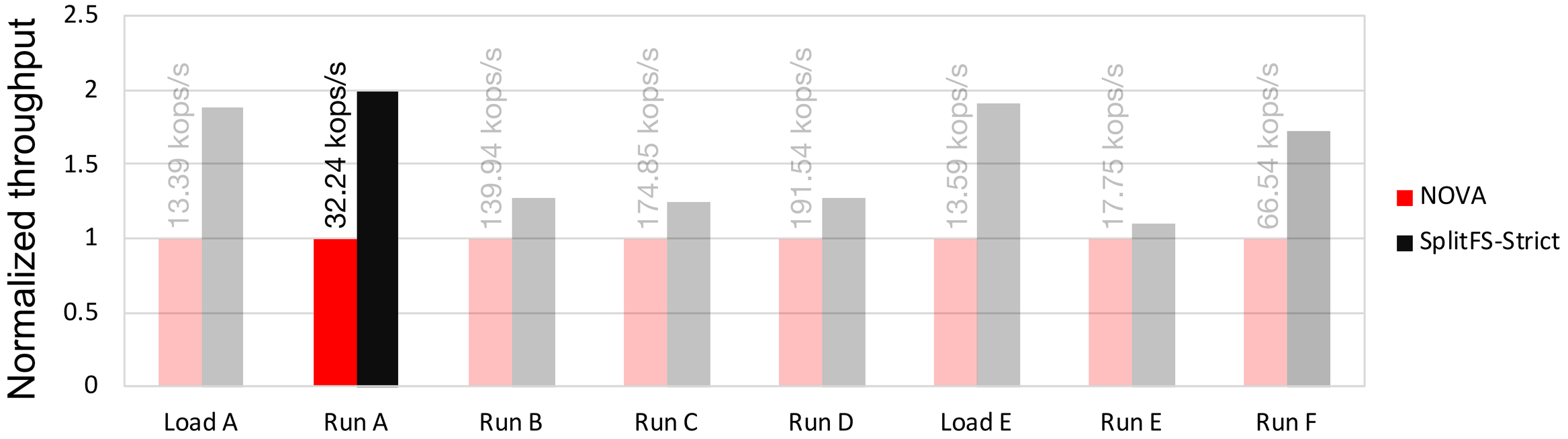


Read-heavy workloads optimized because of converting reads to loads

YCSB on LevelDB

Yahoo! Cloud Serving Benchmark - Industry standard macro-benchmark

Insert 5M keys. Run 5M operations. Key size = 16 bytes. Value size = 1K



Load A - 100% writes

Run A - 50% reads, 50% writes

Run B - 95% reads, 5% writes

Run C - 100% reads

Run D - 95% reads (latest), 5% writes

Load E - 100% writes

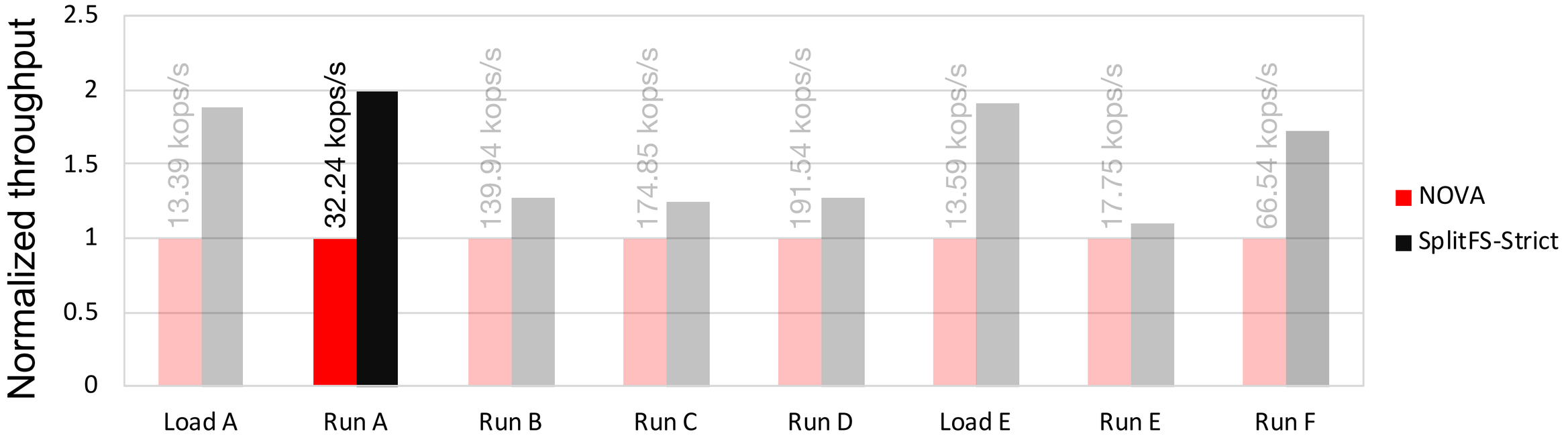
Run E - 95% range queries, 5% writes

Run F - 50% reads, 50% read-modify-writes

YCSB on LevelDB

Yahoo! Cloud Serving Benchmark - Industry standard macro-benchmark

Insert 5M keys. Run 5M operations. Key size = 16 bytes. Value size = 1K



Write-heavy workloads optimized because of staging and relink

SplitFS

SplitFS

SplitFS introduces a new architecture for building PM file systems that…

reduces software overhead,

provides strong guarantees,

and leverages the widely-used ext4-DAX

SplitFS

SplitFS introduces a new architecture for building PM file systems that…

reduces software overhead,

provides strong guarantees,

and leverages the widely-used ext4-DAX

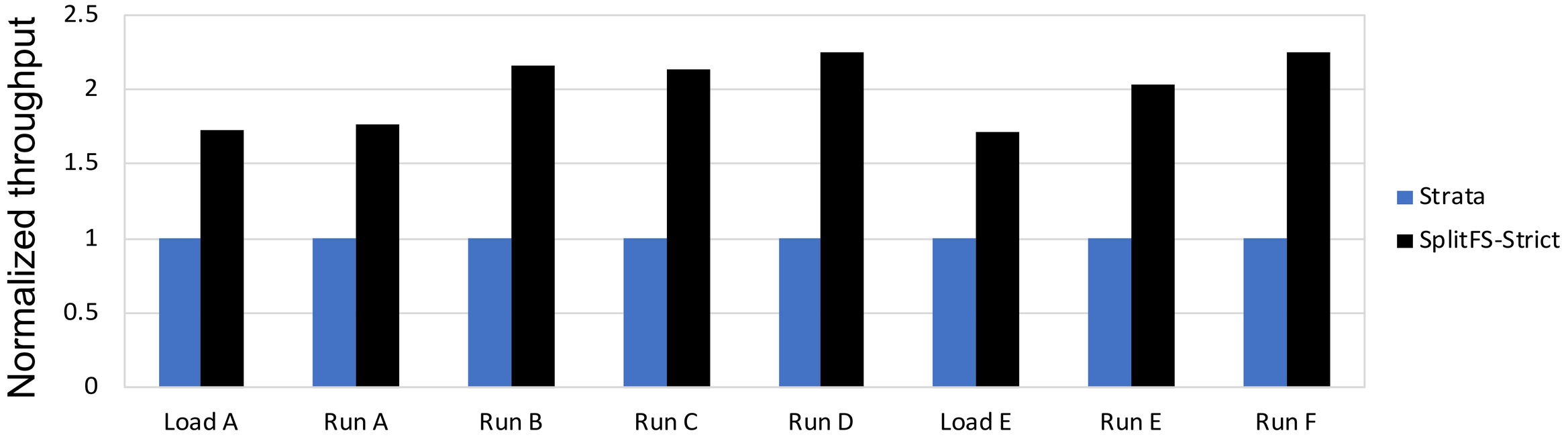
https://github.com/utsaslab/splitfs

Backup Slides

YCSB on LevelDB

Yahoo! Cloud Serving Benchmark - Industry standard macro-benchmark

Insert 5M keys. Run 1M operations. Key size = 16 bytes. Value size = 1K



Load A - 100% writes

Run A - 50% reads, 50% writes

Run B - 95% reads, 5% writes

Run C - 100% reads

Run D - 95% reads (latest), 5% writes

Load E - 100% writes

Run E - 95% range queries, 5% writes

Run F - 50% reads, 50% read-modify-writes