FAST'21

Scalable Persistent Memory File System with Kernel-Userspace Collaboration

Youmin Chen, Youyou Lu, Bohong Zhu,

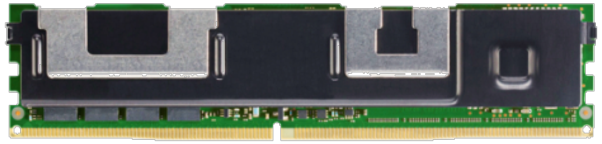
Andrea C. Arpaci-Dusseau[†], Remzi H. Arpaci-Dusseau[†], Jiwu Shu

Tsinghua University [†]University of Wisconsin - Madison

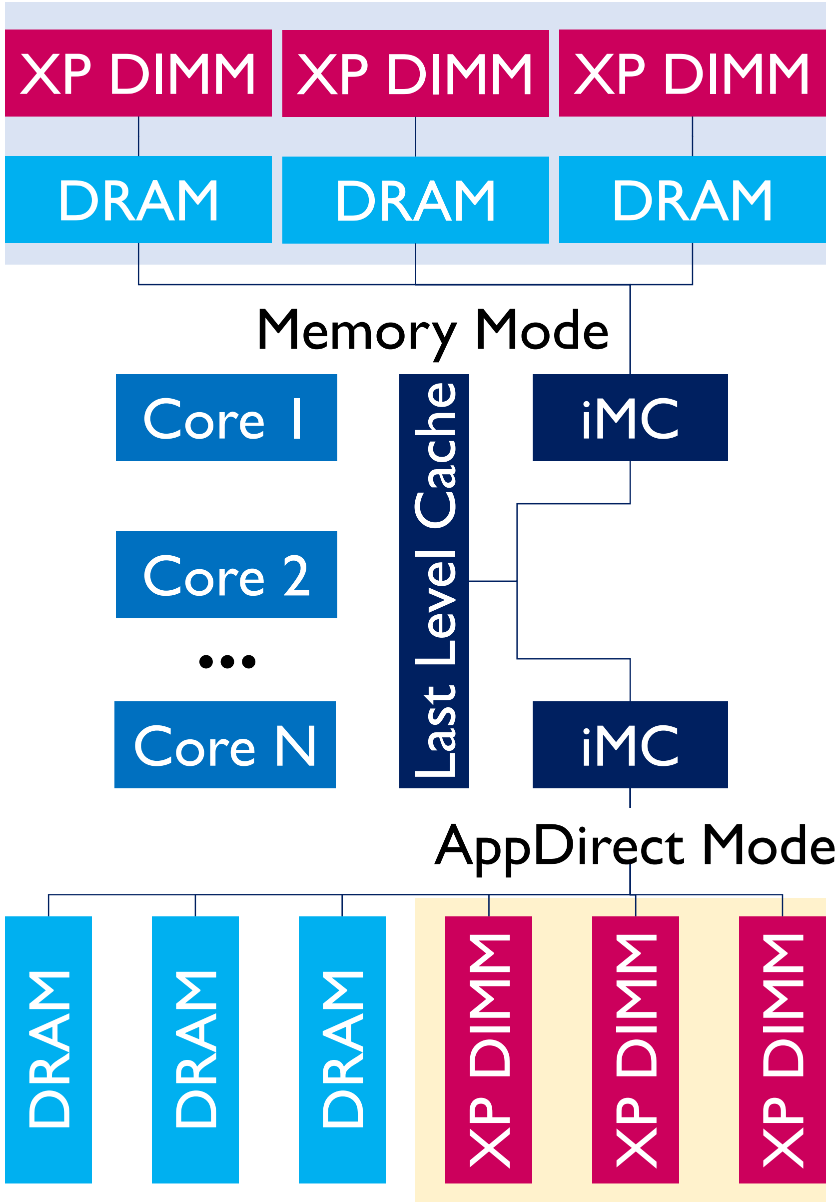
Persistent Memory

- **Hardware Features**
  - Byte-addressability (attached to memory bus)
  - Data persistence (like hard disks)
  - High performance (high bandwidth, low latency)
- Intel Optane DC Persistent Memory
  - Commercially available in 2019
  - Read: 6.7 GB/s per DIMM
  - Write: 2.3 GB/s per DIMM
  - App-Direct Mode vs. Memory Mode

intel OPTANE DC PERSISTENT MEMORY



DIMM Capacity	128GB, 256GB, 512GB
Speed	2666 MT/sec
Platform Capacity	6TB (3TB/cpu)

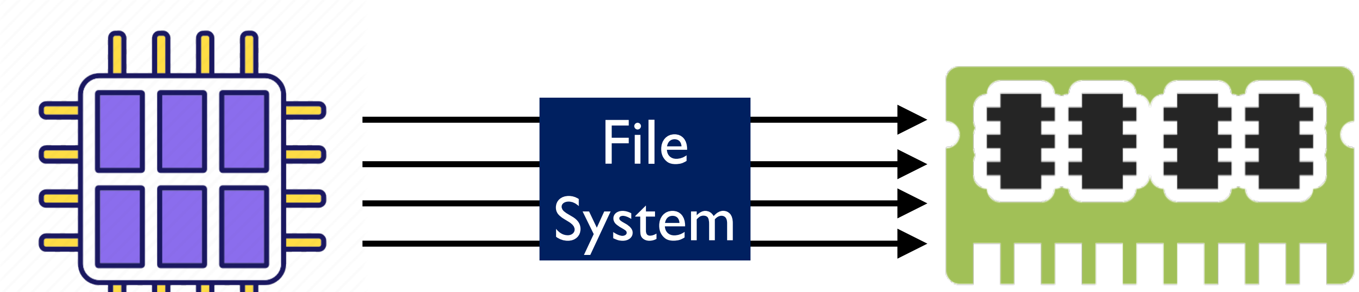


Images are reshaped from "An Empirical Guide to the Behavior and Use of Scalable Persistent Memory", FAST'20

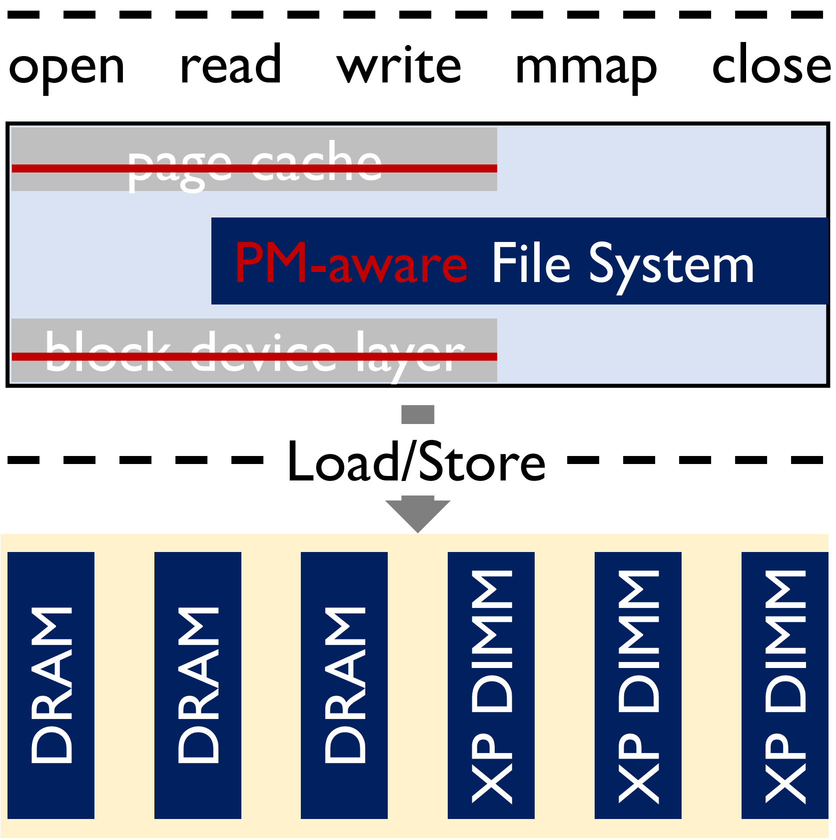
Persistent Memory File Systems

Except for file-based APIs, more expectations…

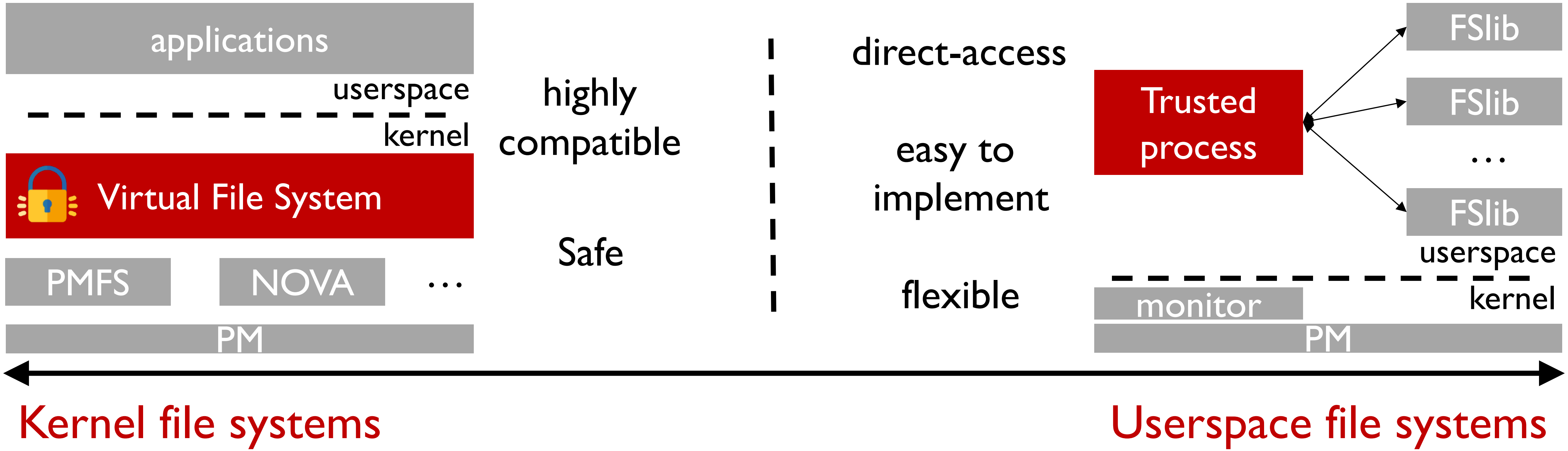
- **High performance**
  - Low software overhead
  - Efficient space mgmt.
  - Light-weight consistency guarantees
- **High scalability**
  - Multicore platform
  - High concurrency PM devices



Scale well ?!

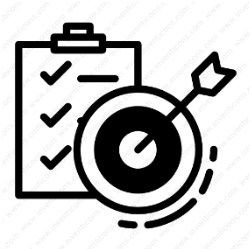


Two Popular PM File System Architectures



- **Software overhead**
  - VFS, syscall
- **Scalability**
  - coarse-grained lock mgmt.

- **Scalability**
  - centralized component E.g., TFS in Aerie [Eurosys'14]; KernFS in Strata [SOSP'17]
- **Vulnerable to stray writes**



Our design goal:

Combine good properties of both kernel and userspace file systems, while delivering high scalability!

Our Approach

A Kernel and userspace Collaborative architecture (i.e., Kuco)

- Based on a client-server processing model
  - Kfs: processes metadata operations, and enforces access control
  - Ulib: provides standard APIs to applications and interacts with Kfs
- Key idea: Shifting tasks from Kfs to Ulib as much as possible
  - Metadata operations: ***collaborative indexing***
  - Write operations: ***two-level locking***
  - Read operations: ***versioned reads***
- Achievements
  - One order of magnitude higher throughput for high-contention workloads
  - Fully saturates the PM bandwidth for data operations

Outline

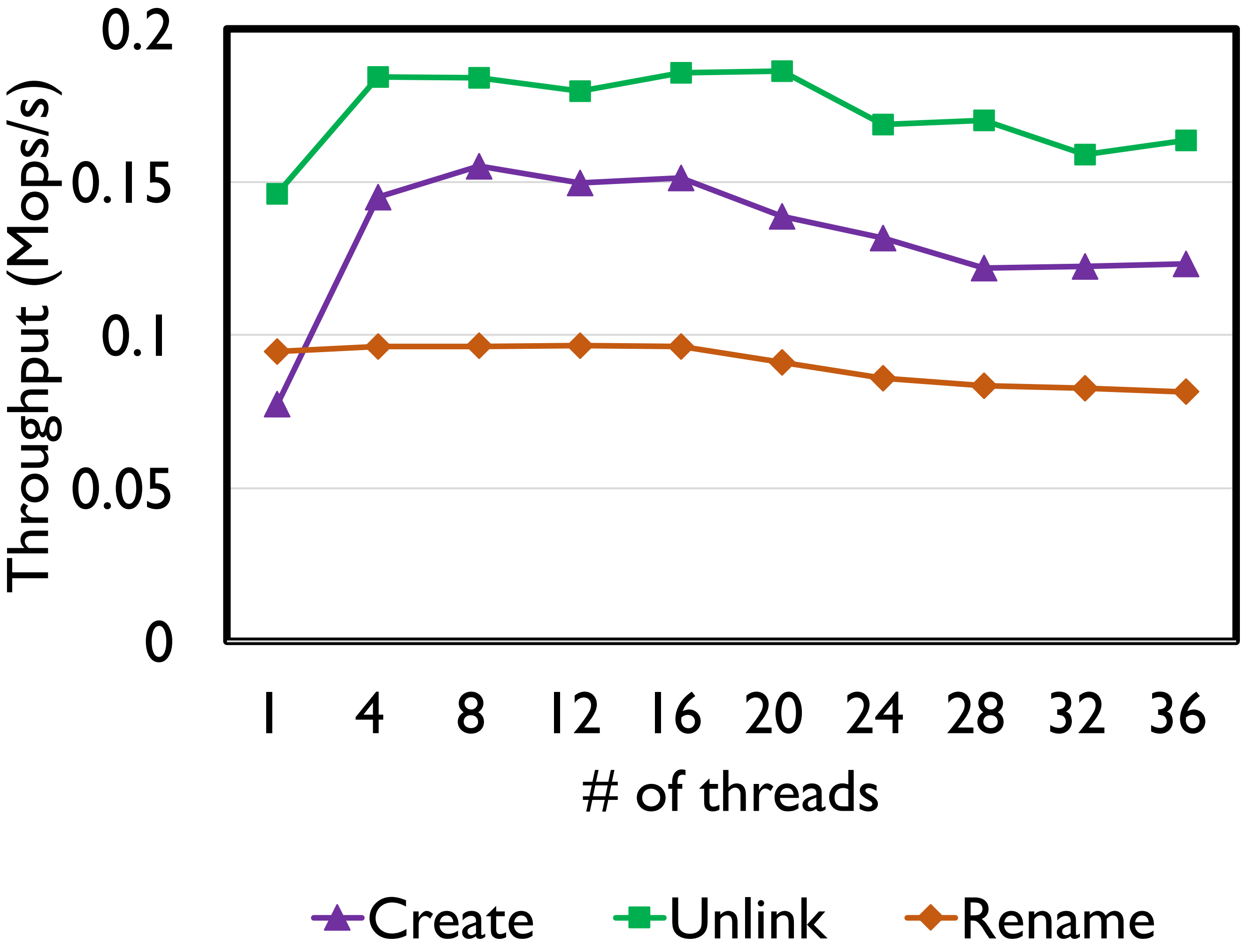
- Introduction
- PM File System Scalability
- Kuco: Kernel-Userpace Collaboration
- Results
- Summary & Conclusion

Scalability I (kernel file systems)

NOVA [FAST'16]:

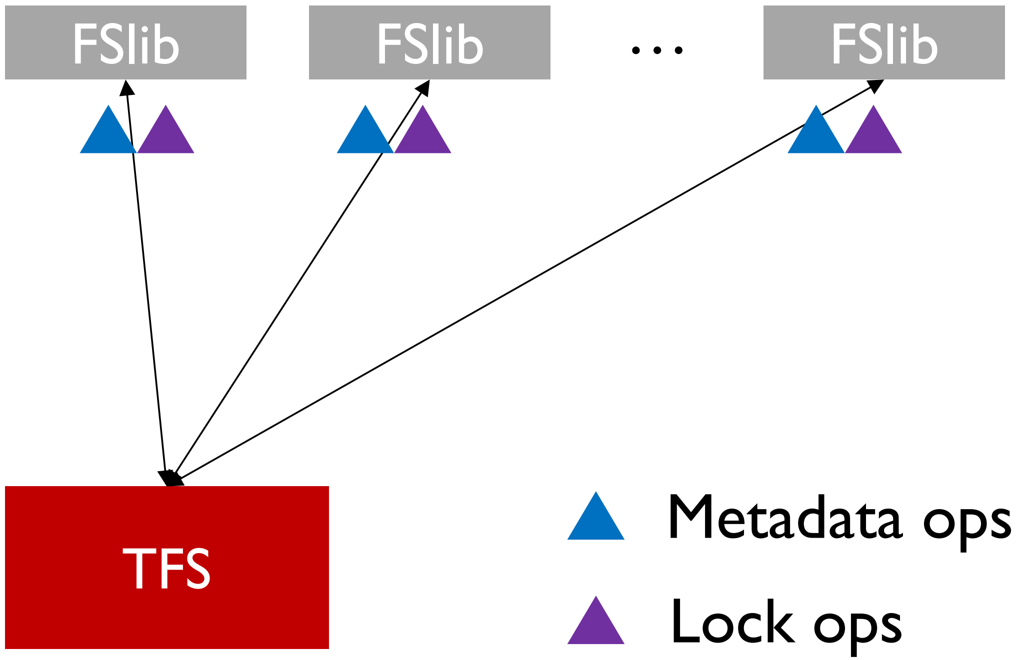
a PM-aware file system

- Avoids using global locks
  - Per-inode log
  - Partitioned free space mgmt.
- Scalable designs do not scale
  - Concurrent operations in shared folders do not scale at all
  - VFS is the culprit: locks the parent directory when updating sub-files

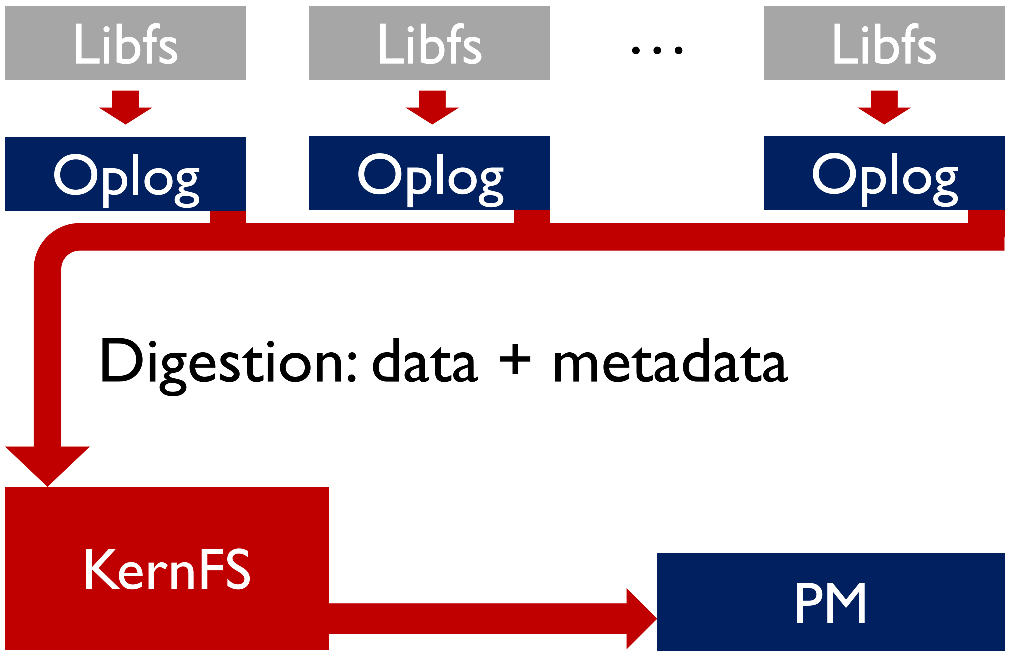


Scalability II (userspace file systems)

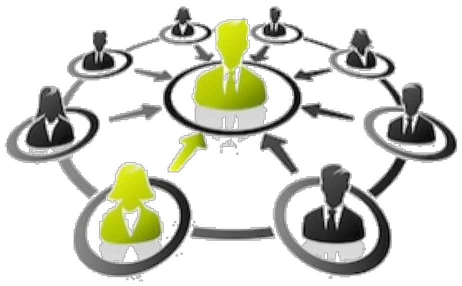
Aerie [Eurosys'14]



Strata [SOSP'17]



Centralized components limit overall scalability!

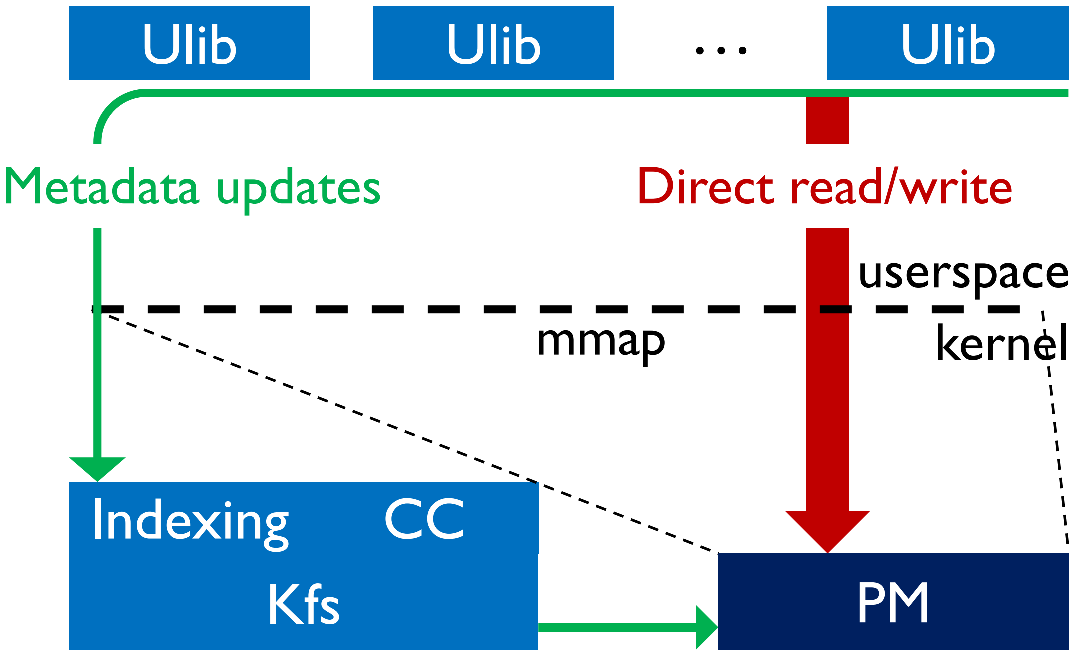


Outline

- Introduction
- PM File System Scalability
- Kuco: Kernel-Userpace Collaboration
- Results
- Summary & Conclusion

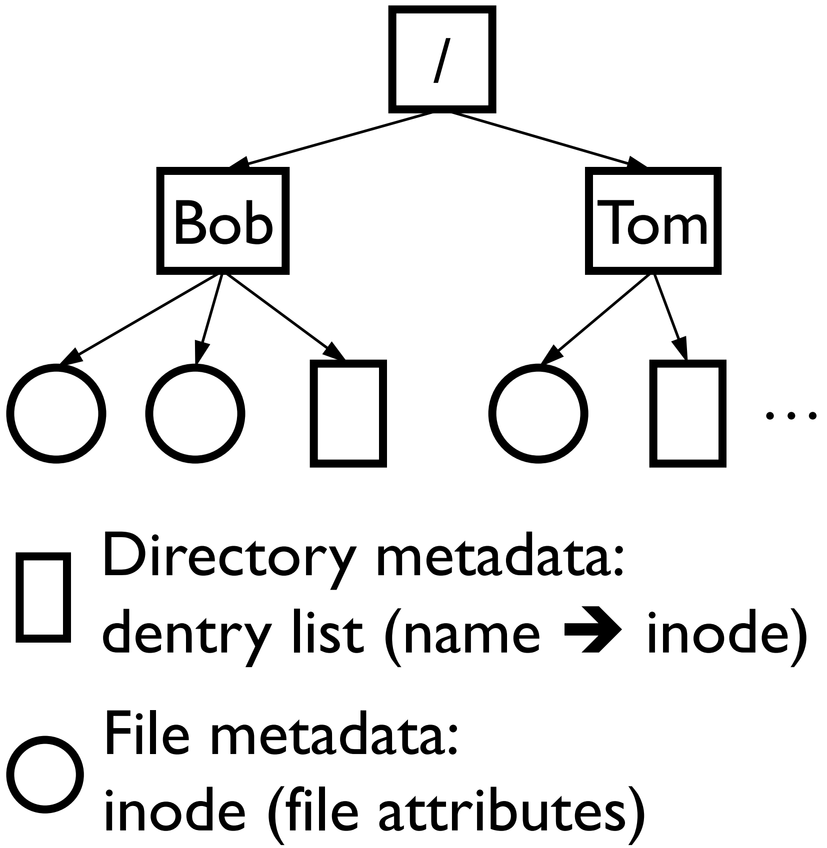
Kuco: Kernel-Userspace Collaboration

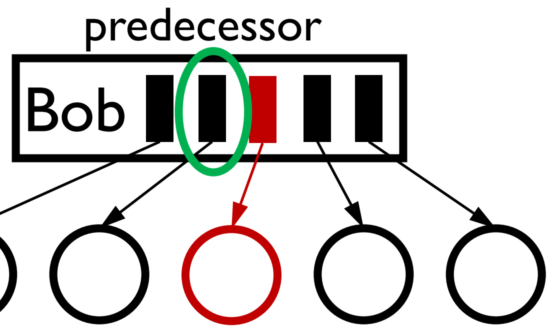
- PM space is mapped to userspace
  - Ulib read/write file data directly
  - Kfs updates metadata & enforce access control
- Client/server processing model
  - Overall throughput is determined by how fast the Kfs processes each request
  - Tmax = 1 / L, where L is the latency for Kfs to process a request
- Key idea: shifting tasks from Kfs to Ulib
  - Improves scalability by reducing the value of L
  - Metadata indexing (i.e., pathname resolution)
  - Concurrency control



Collaborative Indexing

- Pathname resolution
  - Recursive and random memory access
  - Large directories or deep hierarchies
- Collaborative indexing
  - PM space is mapped to userspace
  - Ulib pre-locates metadata items in userspace before sending a request to Kfs
  - Kfs update metadata items directly with the given addresses
- Examples: Creat()
  - create a new inode, insert a dentry
  - Ulib passes the address of the predecessor of the target dentry in the parent dentry list



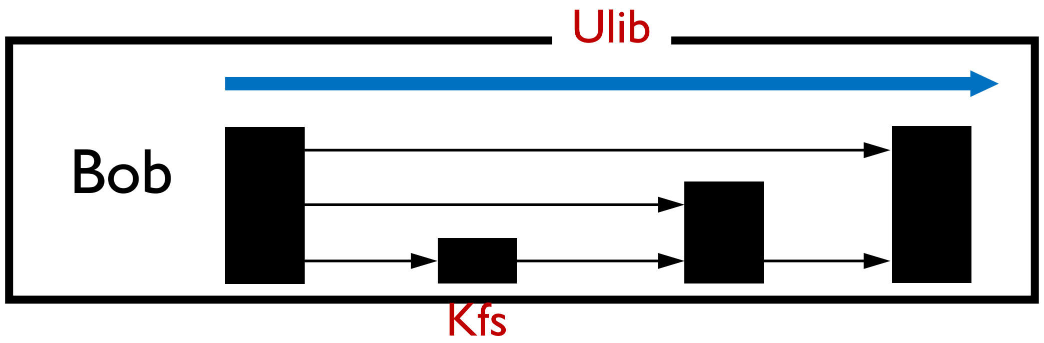


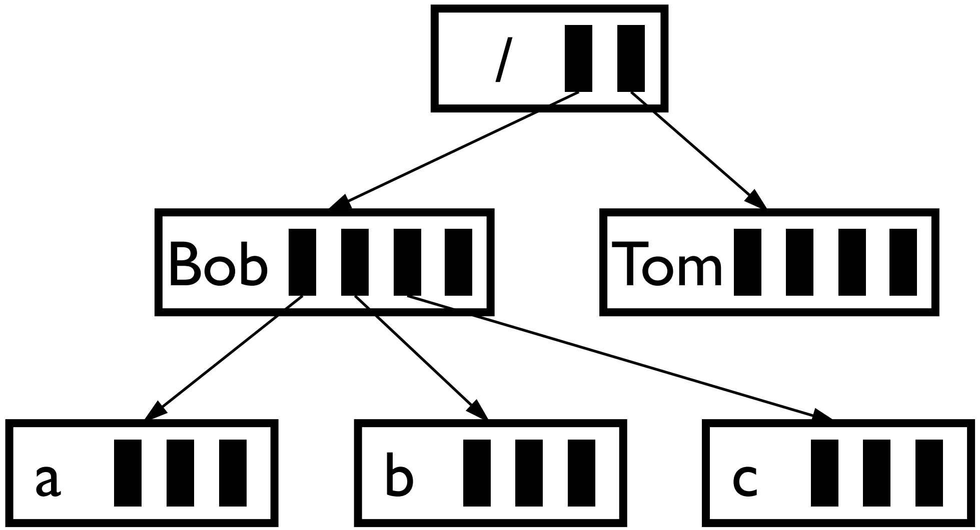
Correctness & Safety

- ❖ Concurrent updates
  - ❖ Q1: Ulib may read inconsistent metadata when Kfs is updating it
  - ❖ Q2: Ulib may send obsolete metadata to Kfs when another Ulib changed this metadata

1) Pointers should point to consistent items

  - ❖ Ulib may read inconsistent items when Kfs is updating concurrently
  - ❖ Each dentry list is managed via a lock-free skip list





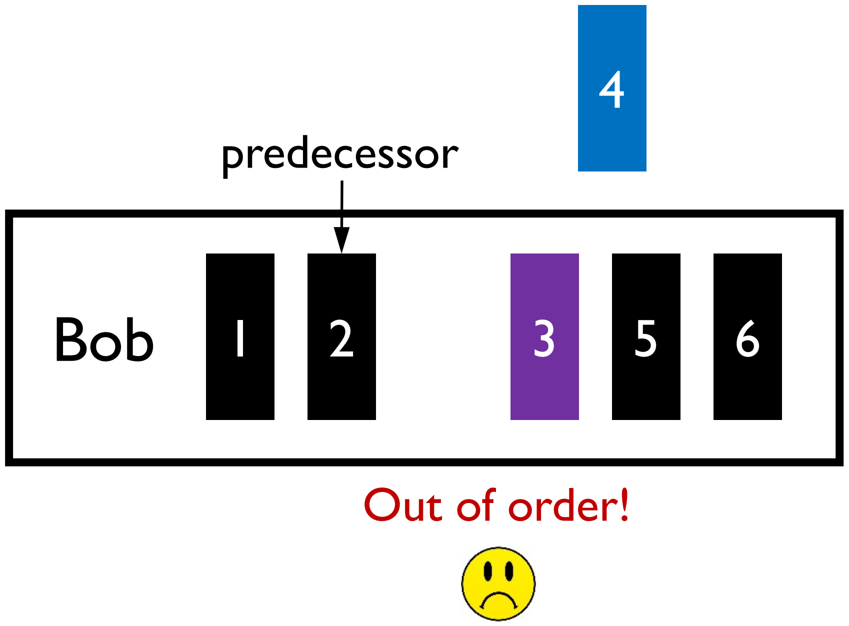
- directory tree consists of hierarchical dentry lists
- Only insert & delete operations (including rename)

Correctness & Safety

- Concurrent updates:
  - Q1: Ulib may read inconsistent metadata when Kfs is updating
  - Q2: Ulib may send obsolete metadata to Kfs when another Ulib changed this metadata

1) Pointers should point to consistent items

2) Pointers should be up-to-date
- Ulibs are scanning in a lock-free manner
  - *Epoch-Based Reclamation* prevents reading deleted items
- Predecessor may be no longer a predecessor
  - Rechecking prevents reading obsolete items



More details: checkout our paper

- Two-level locking
  - Between different processes: distributed lease
  - Between different threads within the same process: Userspace range lock
- Three-phase writes
  - Avoid stray writes
- Versioned reads
  - Old and new copies of written pages are kept due to a CoW way
  - Kuco enables the readers to read a consistent snapshot of file data w/o interacting with Kfs by embedding extra version bits in the block mapping
- Read protection
- Crash consistency & data layout

Outline

- Introduction
- PM File System Scalability
- Kuco: Kernel-Userpace Collaboration
- Results
- Summary & Conclusion

Experimental Setup

Hardware Platform

CPU	2 Xeon Gold 6240m CPUs (**36 physical cores**)
DRAM	384 GB (32GB/DIMM)
PM	12 Optane DCPMMs (**3 TB, 256 GB/DIMM**),
Operating System	Ubuntu 19.04, Linux 5.1

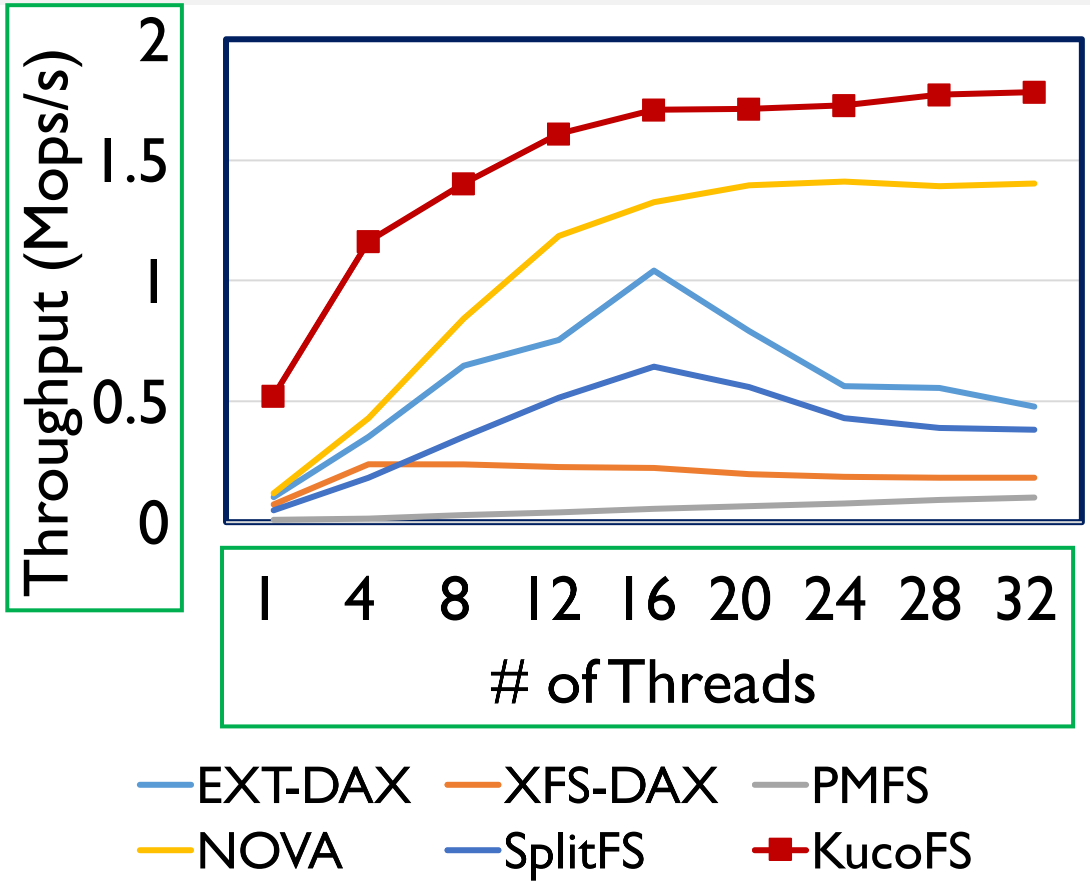
Compared Systems

Kernel File System	Ext4-DAX, XFS-DAX, NOVA [FAST'16]
Userspace File System	Aerie [Eurosys'14], Strata [SOSP'17], SplitFS [SOSP'19]

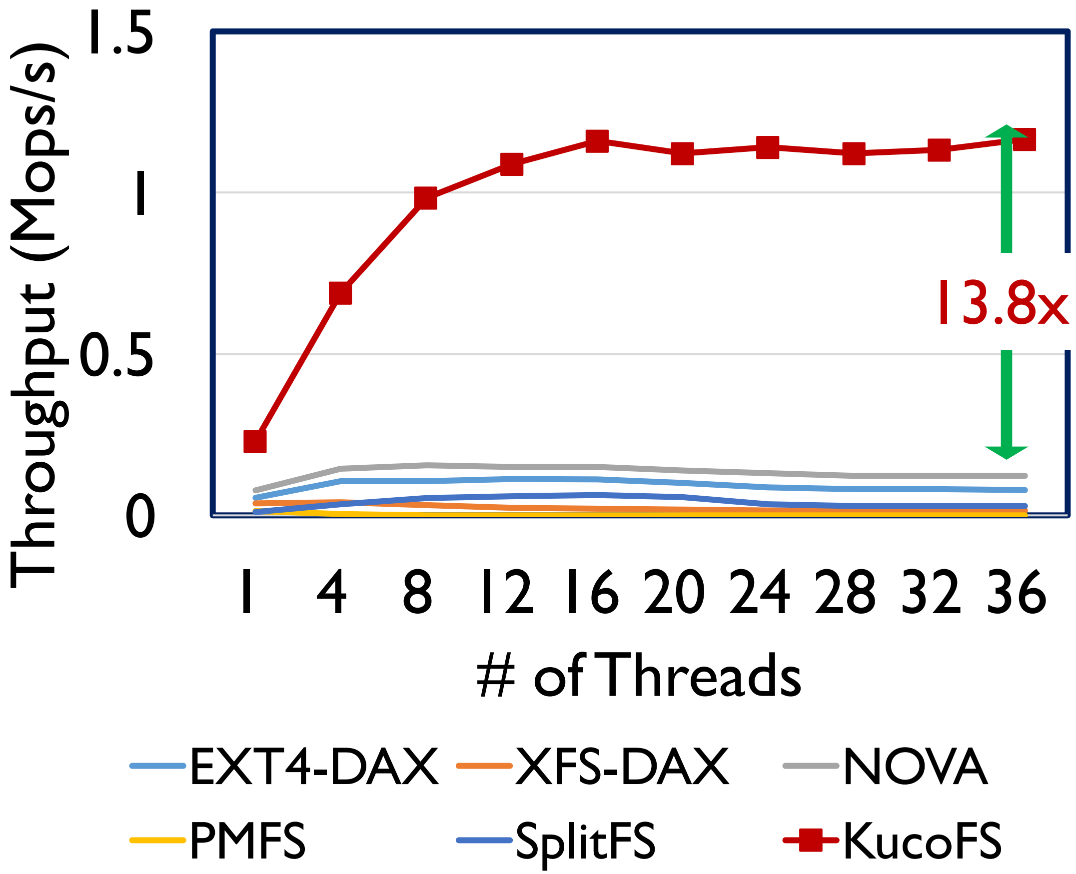
Benchmark

- FxMark: sharing level (low/medium/high), mode (data/metadata), operation (write, creat, …)
- Filebench (Fileserver, Webserver, Webproxy, Varmail)

Metadata scalability



(a) Threads create files in private folders



(b) Threads create files in a shared folder

(1) Kfs only performs very light-weight work
(2) No lock is required (all updates are delegated to Kfs)

Outline

- ❖ Introduction
- ❖ PM File System Scalability
- ❖ Kuco: Kernel-Userpace Collaboration
- ❖ Results
- ❖ Summary & Conclusion

Summary & Conclusion

- PM file systems are desired to deliver high scalability
  - Kernel file systems: VFS is hard to bypass
  - Userspace file systems: requires a centralized coordinator
- Coarse-grained split between kernel and userspace: SplitFS
  - Metadata operations are process by Ext4
  - Data operations are conducted in userspace
  - Still hard to scale
- PM-aware file system requires a fine-grained task split and collaboration between kernel and userspace:
  - Kuco: combine the advantages of both parts while delivering high scalability

Thanks for watching!

WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON