Max: A Multicore-Accelerated File System for Flash Storage

Xiaojian Liao, Youyou Lu, Erci Xu, Jiwu Shu

Tsinghua University

Agenda

- Background
- Motivation
- Design and Implementation
- Evaluation
- Conclusion

Background: Storage trend

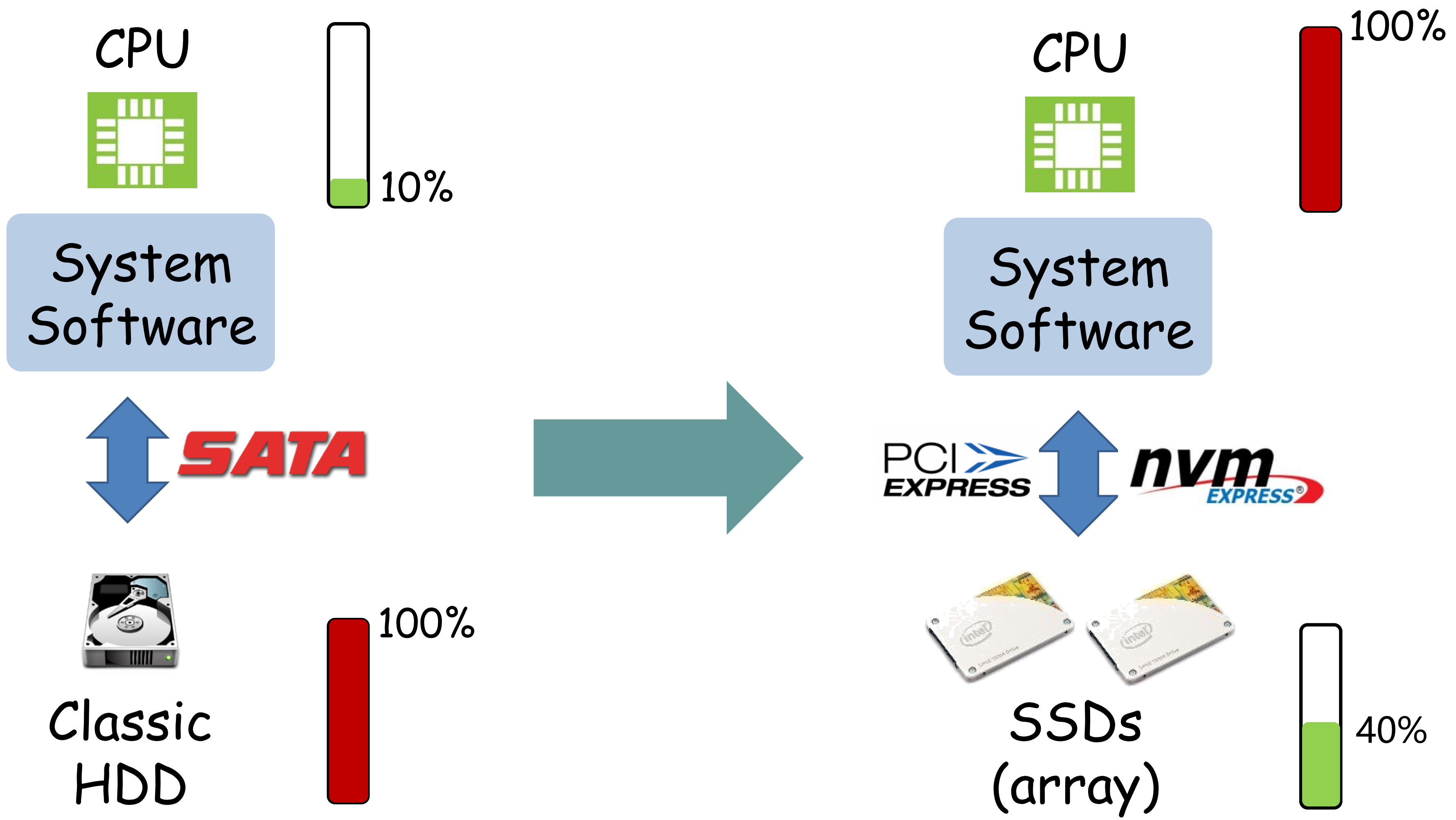
Modern storage hardware delivers higher write bandwidth.

	HDD	SATA SSD	NVMe SSD
Bandwidth	~80 MB/s	~500 MB/s	~5 GB/s

Flash-based SSDs provide higher sequential write bandwidth.

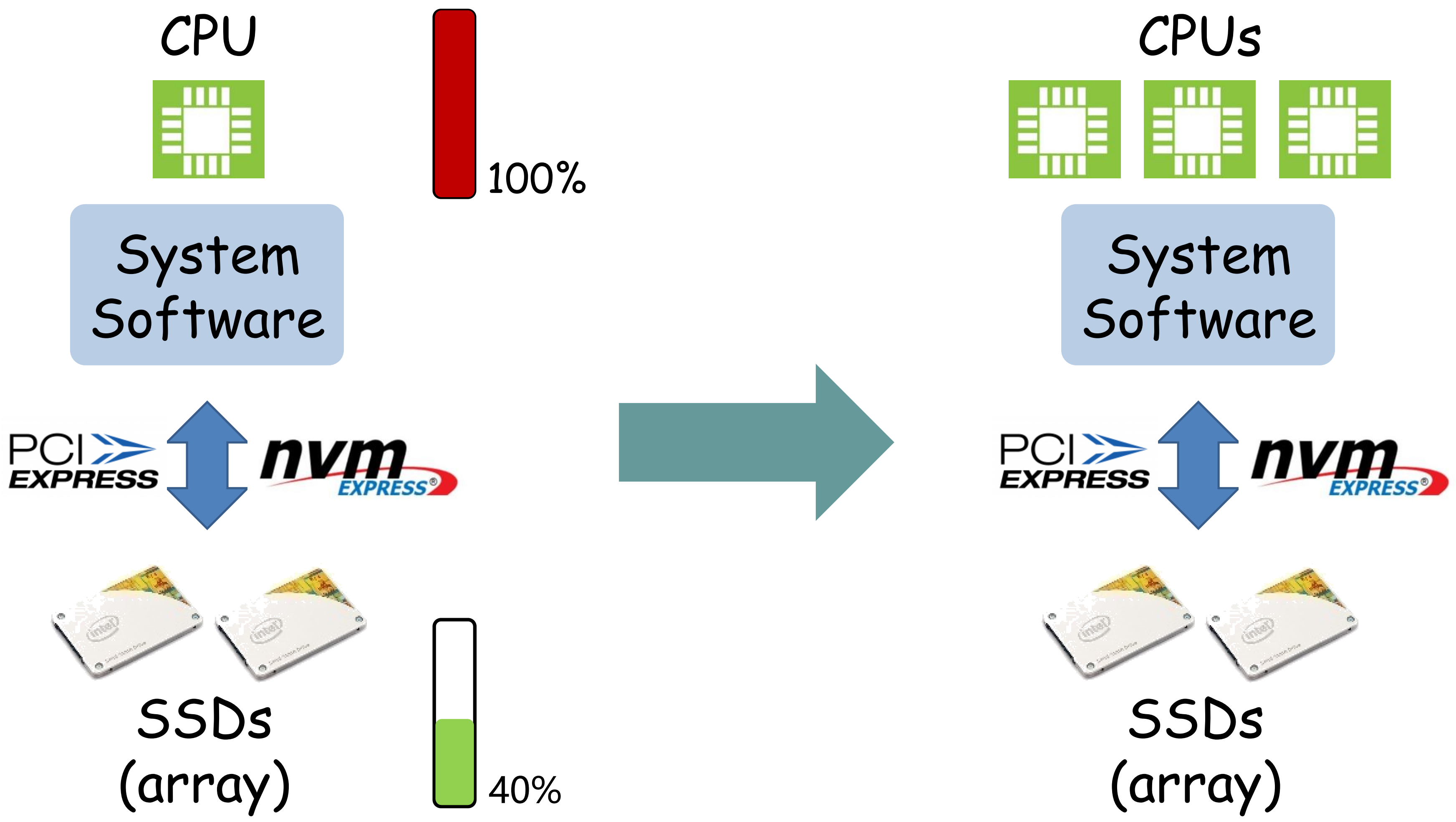
Year	2014	2020	2021
Product (media)	Intel DC P3700 (MLC)	Samsung 980pro (V-NAND MLC)	Intel D5-P5316 (QLC)
Seq. bandwidth	1900 MB/s	5 GB/s	3.6 GB/s
Rand. bandwidth	600 MB/s	3.2 GB/s	31 MB/s
Seq. / Rand.	3.2	1.6	116

Background: Storage system



The CPU or system software become the bottlenecks.

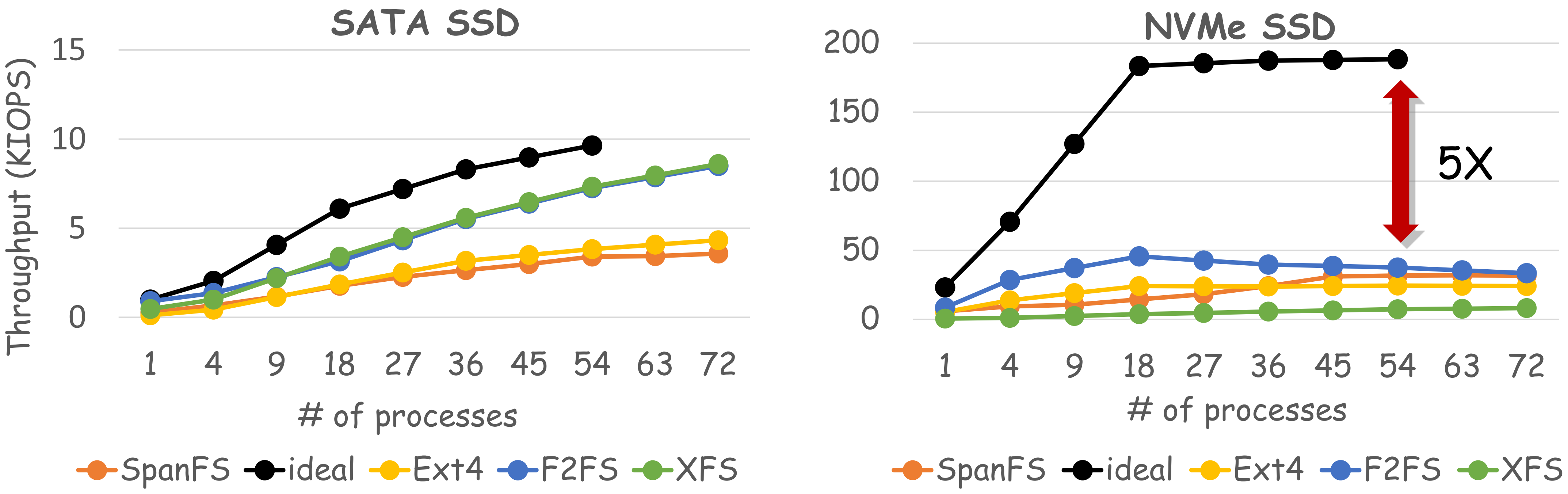
Background: Storage system



Employing multicore CPUs to the system software.

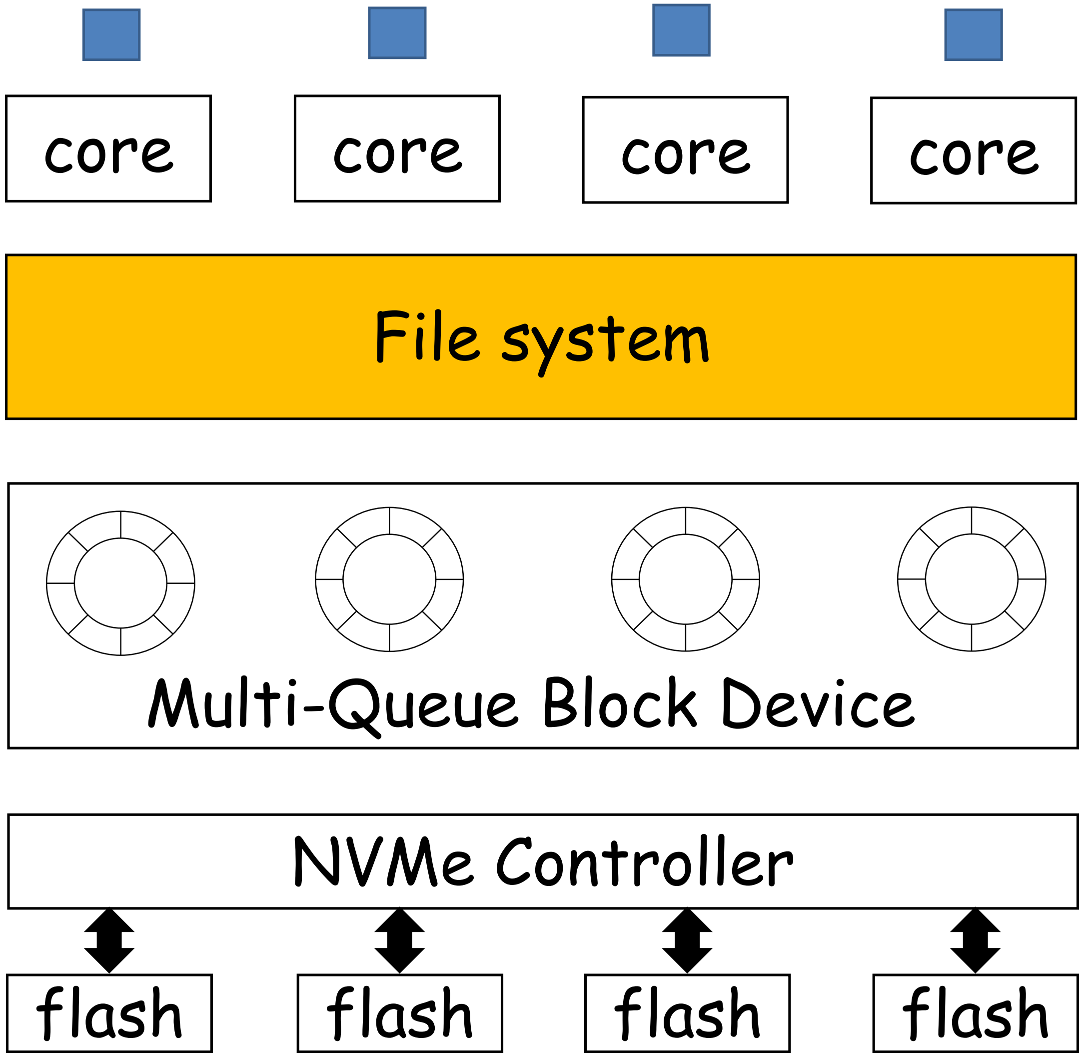
Motivation: Multicore scalability

- Workload: concurrent I/Os including *create()*, *write()*, *fsync()* and *unlink()*.
- ideal: partition the drive and the system software; run an independent F2FS atop each partition.
- Other Linux file systems: multiple CPU cores share a single drive partition.



Existing Linux file systems fail to scale for high performance SSDs.

Motivation: Analysis



Concurrent I/Os and multicore CPUs

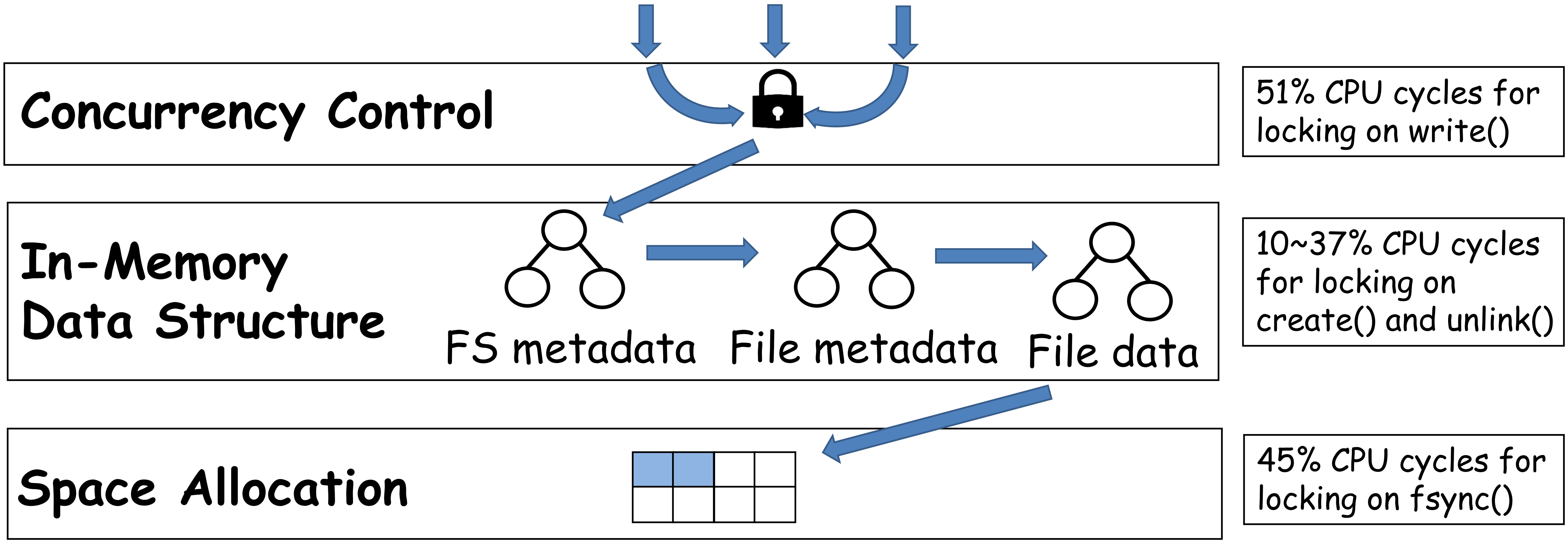
File system becomes the scalability bottleneck

Multicore-friendly design of block layer and device driver

High internal data parallelism of the SSD

Motivation: Analysis

Use F2FS, an existing LFS designed for flash-based SSD, as an example



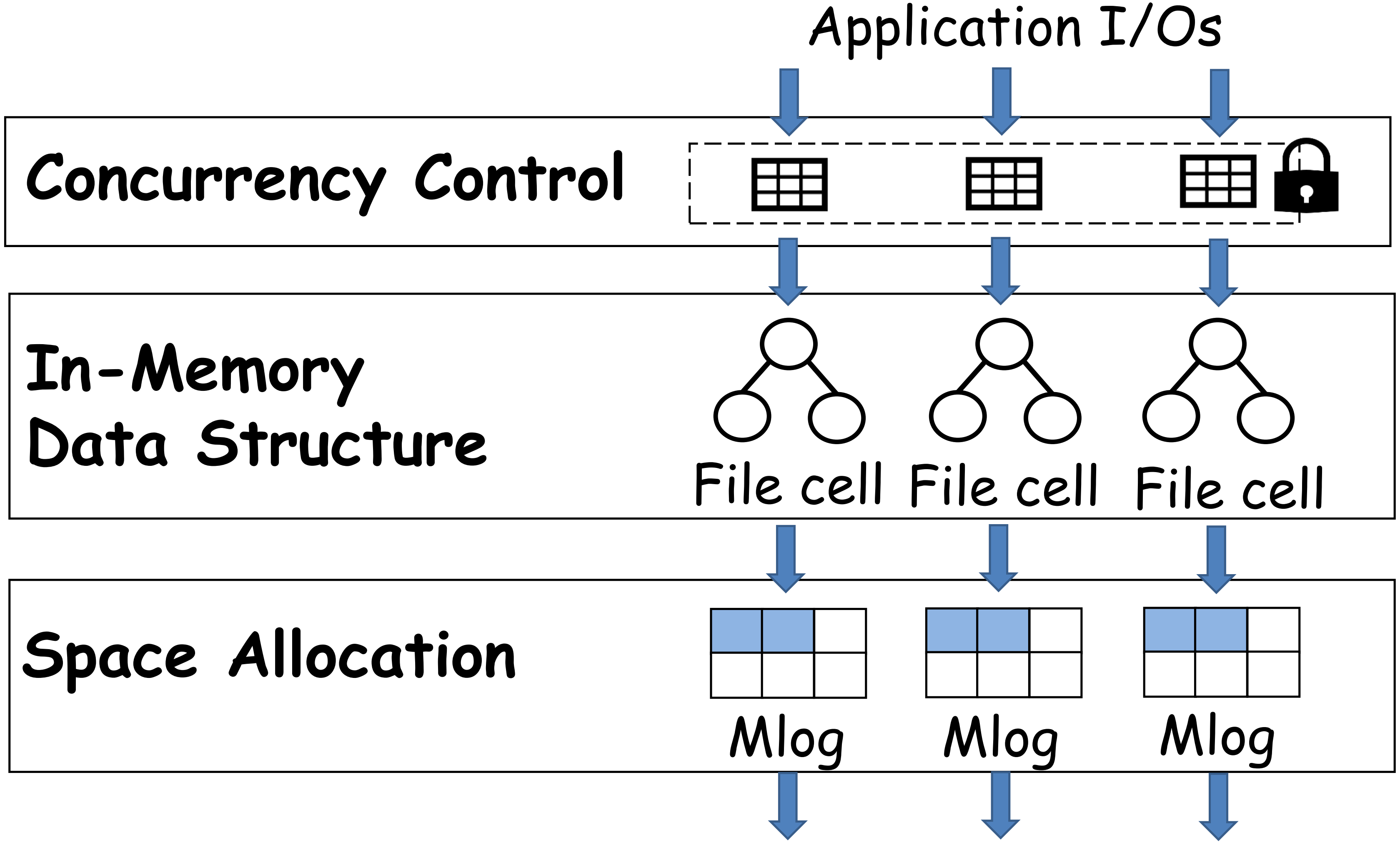
CPU becomes the bottleneck; I/O device is underutilized.

Agenda

- Background
- Motivation
- Design and Implementation
- Evaluation
- Conclusion

Max overview

Key idea: sequential I/O (log structure) + efficient CPU use (sharding)



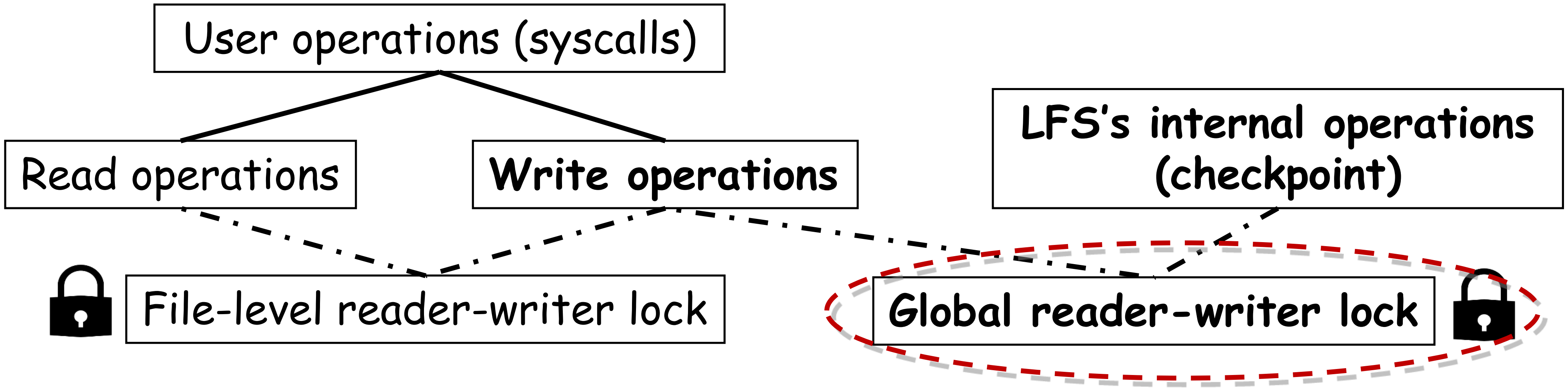
Reader Pass-through Semaphore for CC

File cell to scale the access to IMDS

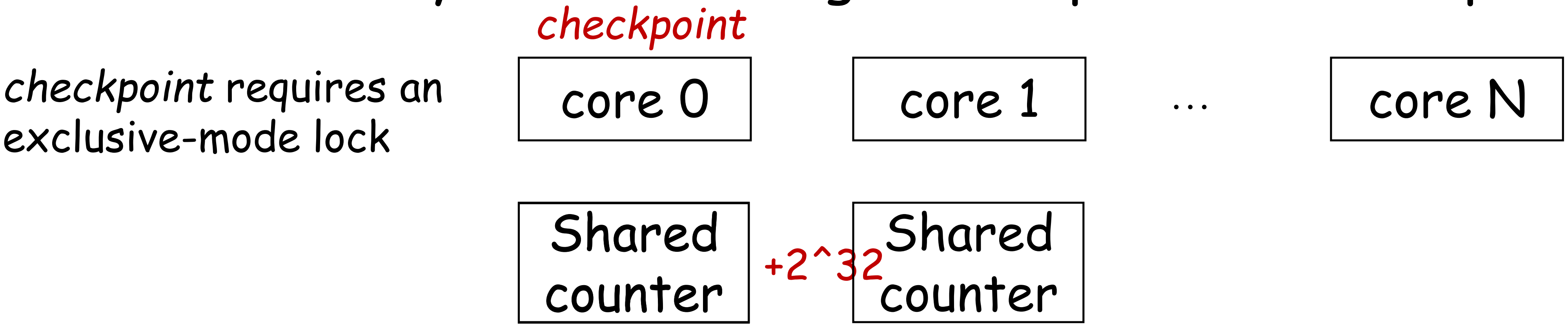
Mlog for concurrent SA and persistence

LFS's concurrency control

1. Operations and concurrency control of classic LFS

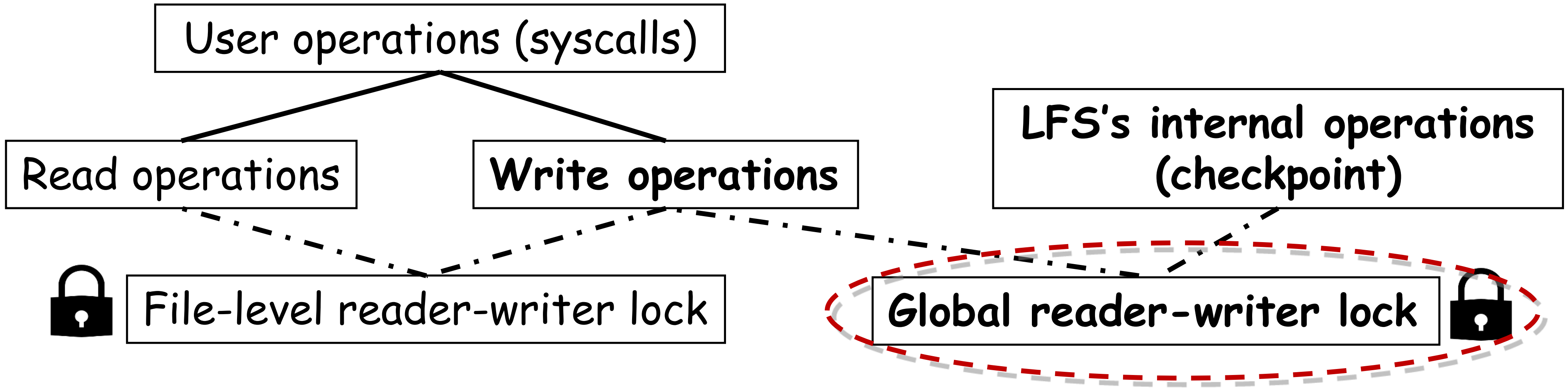


2. Concurrency control among write ops and a checkpoint

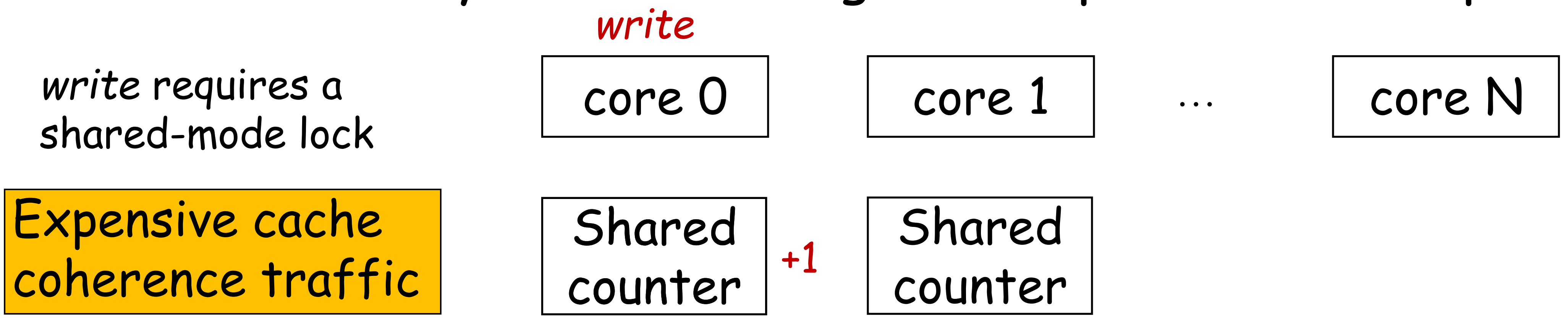


LFS's concurrency control

1. Operations and concurrency control of classic LFS

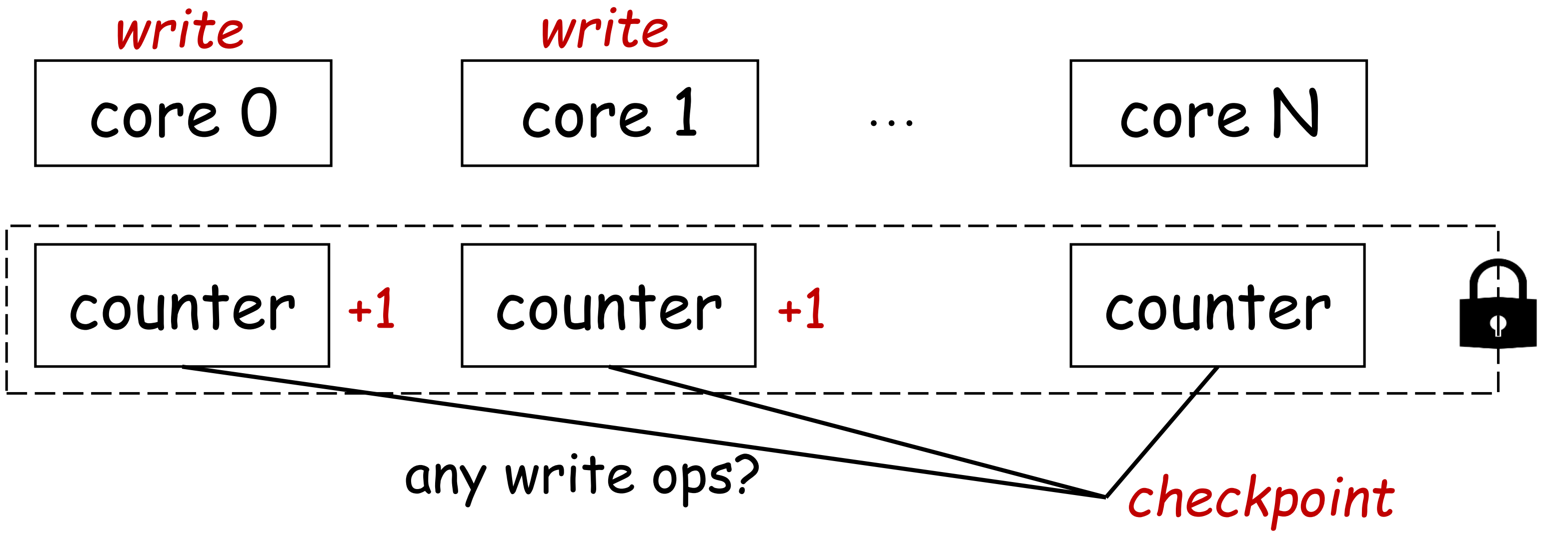


2. Concurrency control among write ops and a checkpoint



Reader Pass-through Semaphore

Key idea: per-core counters + CPU scheduler's free rides



aggressive polling (ns)

low latency but interfere other cores (e.g., IPIs)

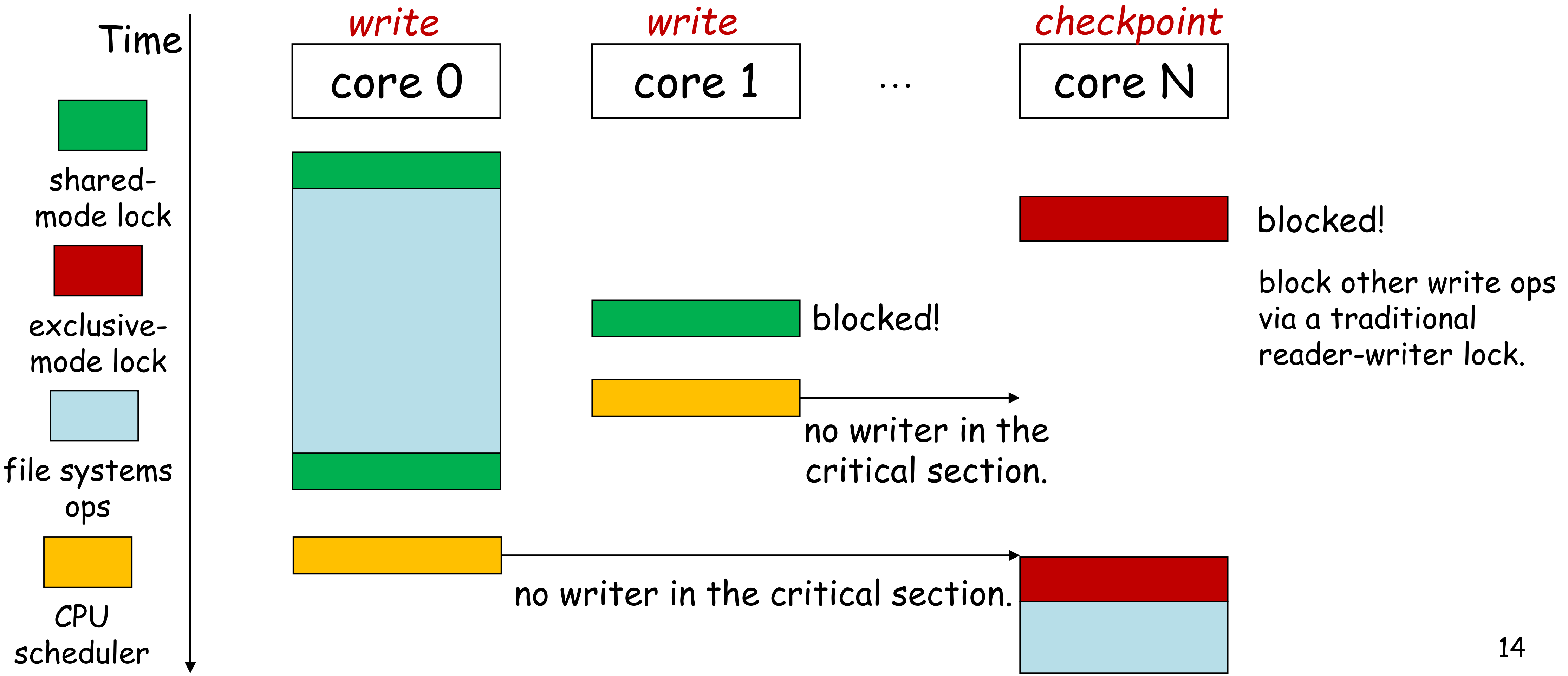
scheduler-assisted check (us-scale, consistent with SSDs' latency)

lazy wait (ms)

high latency and slows the checkpoint

Reader Pass-through Semaphore

The FS in kernel space frequently triggers CPU scheduler, especially when a syscall is finished.



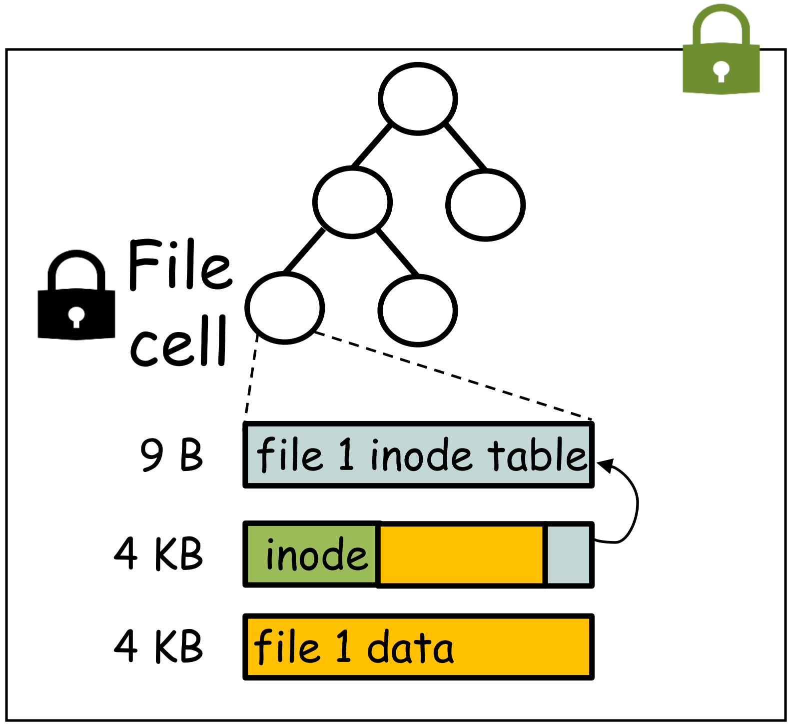
File Cell

Key idea: partition and reorganize in-memory data structure

File cell group 0
inode number % N = 0

…

File cell group N
inode number % N = N-1



write(), read(), stat(), fsync():

1. *lock group/indexing in shared-mode*
2. *lock file cell of target file*
3. *file operations*

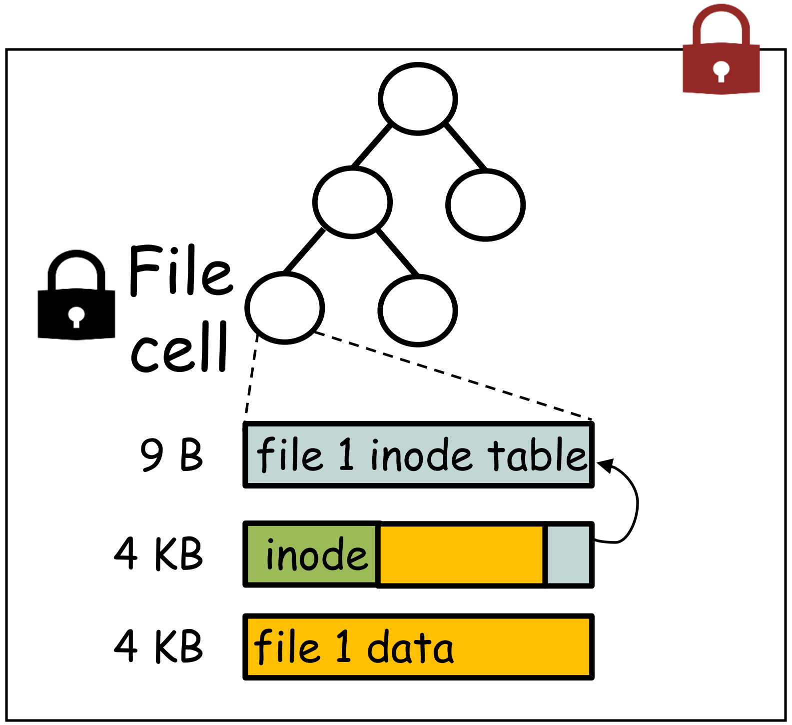
File Cell

Key idea: partition and reorganize in-memory data structure

File cell group 0
inode number % N = 0

…

File cell group N
inode number % N = N-1

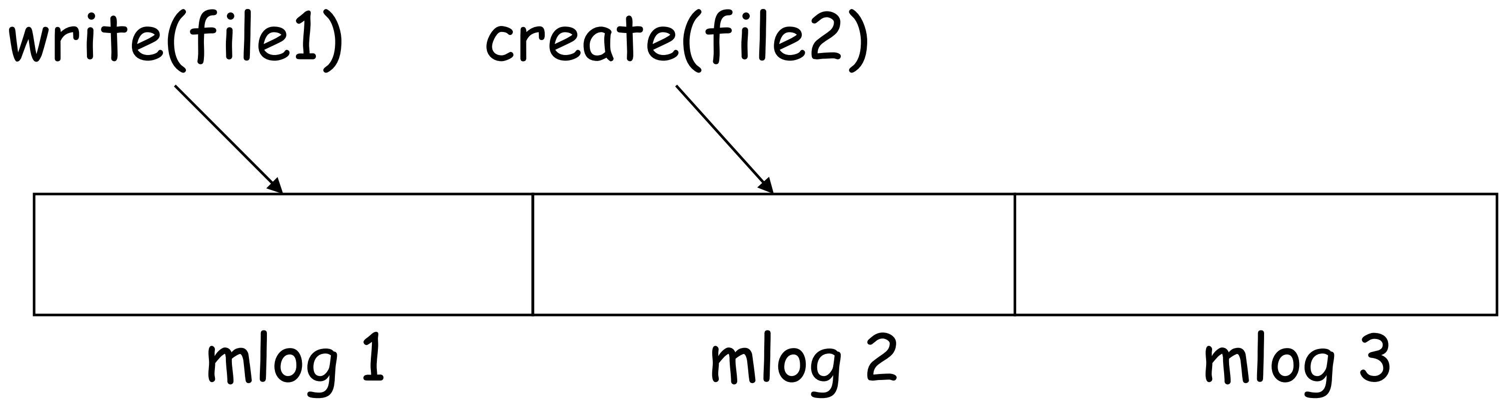


creat(), unlink(), mkdir():

1. *lock group/indexing in exclusive-mode*
2. *lock file cell of target files and directory*
3. *file operations*

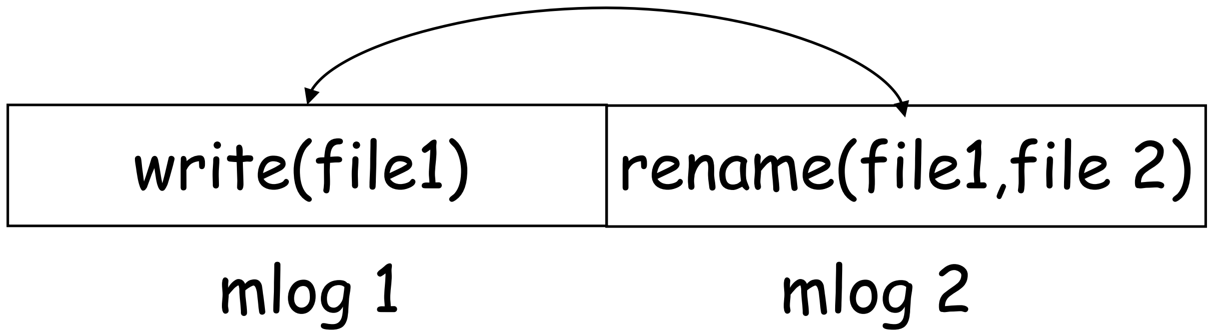
Mlog

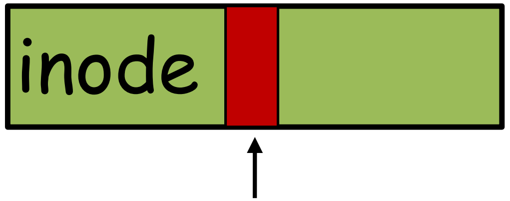
Key idea: multiple logs, each serves for atomic file operations



- dispatch atomic file operations in a round-robin fashion
- each mlog maintains its internal consistency

How to keep consistency over multiple mlogs?





Use a global version number to decide the persistence ordering

Agenda

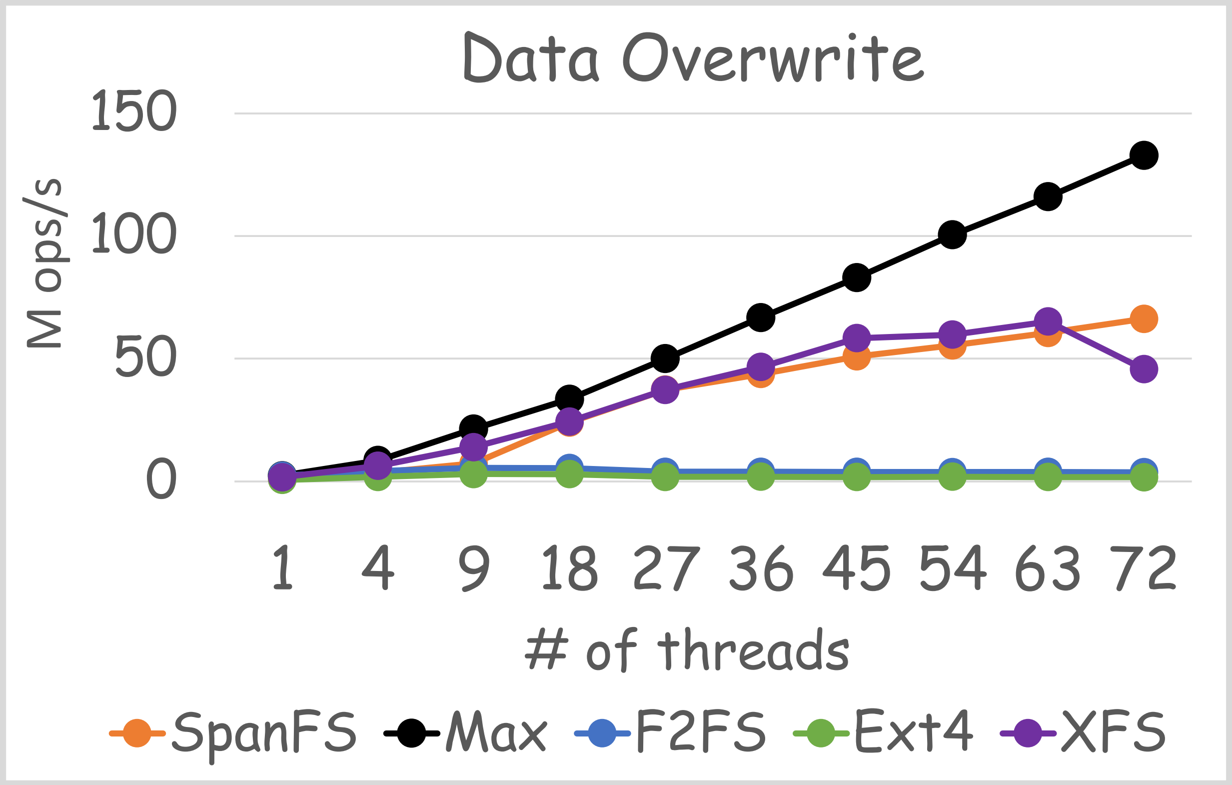
- Background
- Motivation
- Design and Implementation
- Evaluation
- Conclusion

Evaluation

CPU	4 Intel Xeon Gold 6140 CPU, each with 18 cores, totally 72 physical CPU cores
SSD	Intel DC P3700 2 TB SSD
Compared system	Linux vanilla kernel 4.19.11; ext4, XFS, F2FS [FAST'15], SpanFS [ATC'15]
Workloads	• Data and metadata scalability; • Varmail and RocksDB; • Upper bound evaluation against tmpfs;

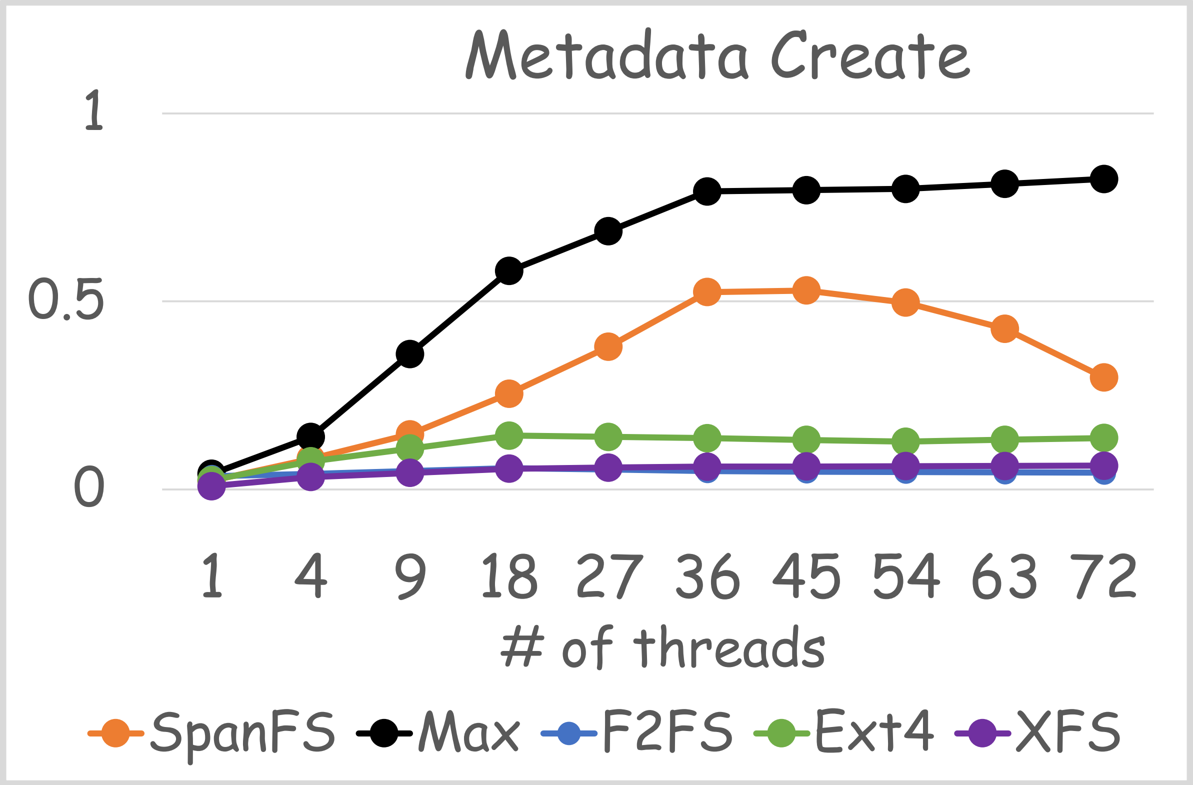
Data and metadata scalability

Parallel data intensive ops



Max-72threads = 56 x Max-1thread

Parallel metadata intensive ops

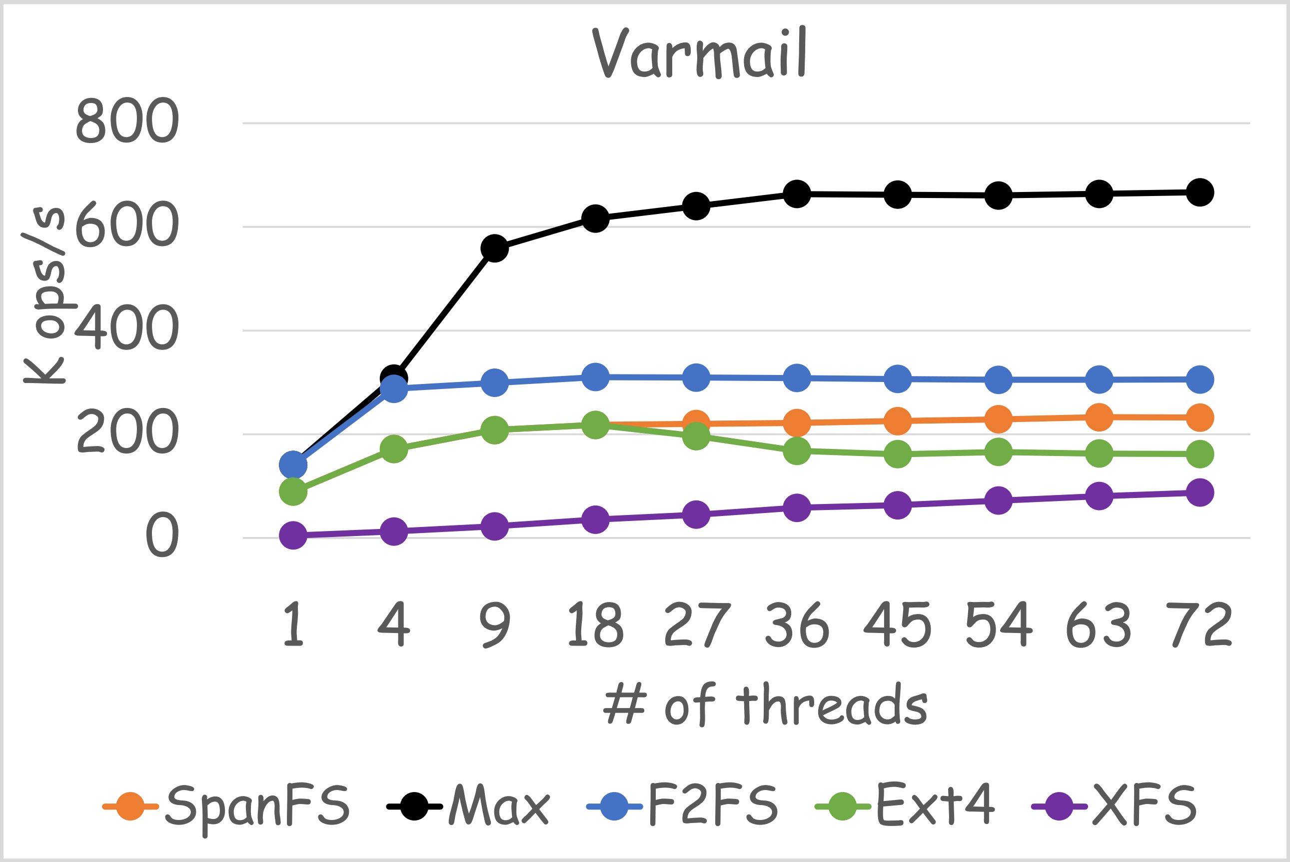


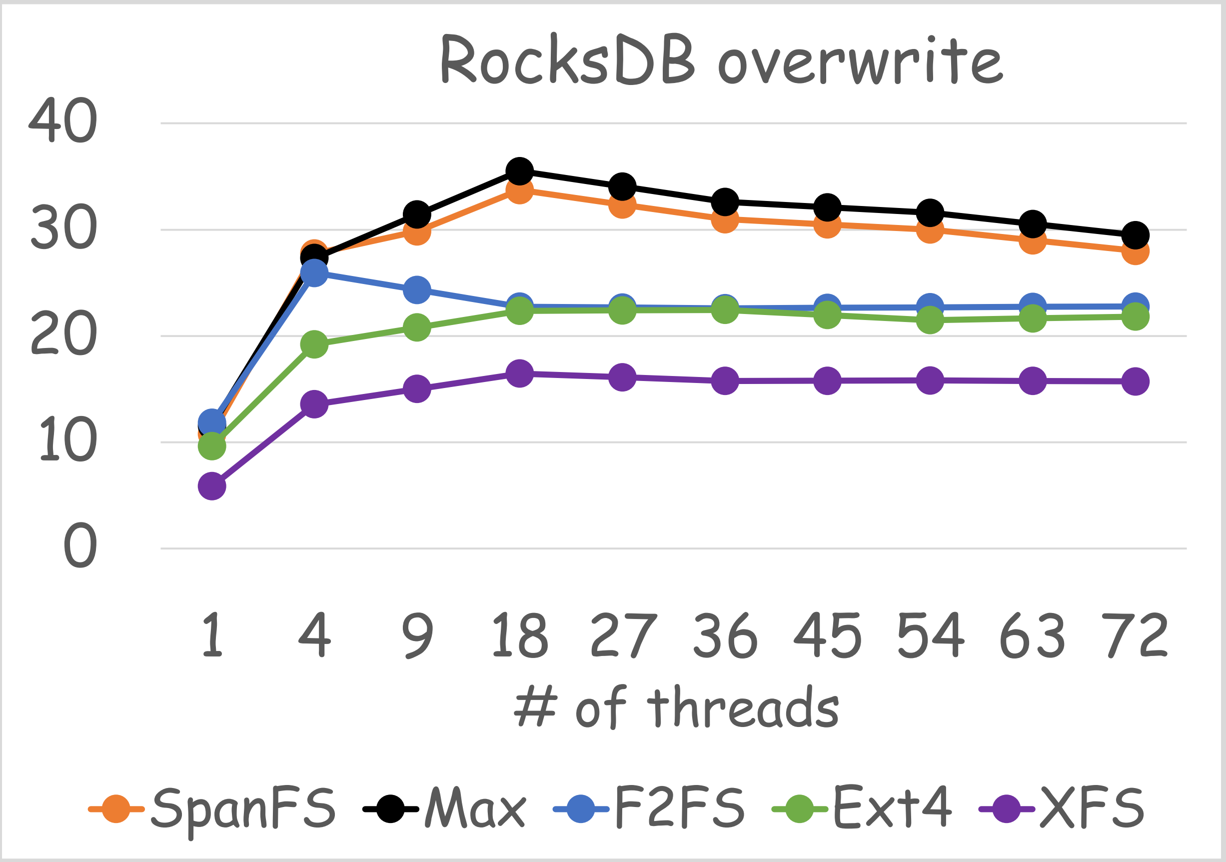
Max = 2.8 x SpanFS
Max = 18.6 x F2FS

Application scalability

create, unlink and fsync intensive

compaction intensive, value size = 8 KB



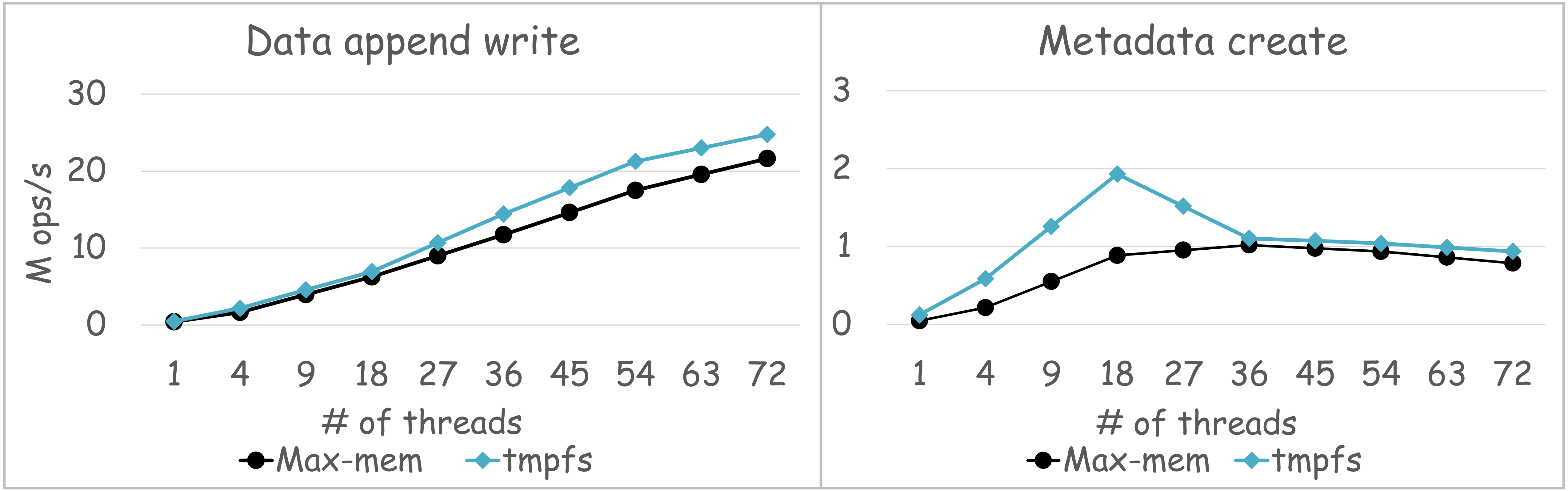


Max = 2.9 x SpanFS = 2.1 x F2FS

Max > SpanFS = 1.5x F2FS

Upper bound evaluation

tmpfs: a simple wrapper of Linux VFS, a memory-based file system
Max-mem: disable fsync and page cache flushes to avoid duplicate on-disk copy
tested device: 20GB memory-backed RAMdisk



The throughput of Max-mem comes close to tmpfs

Conclusion

- Max: A Multicore-Accelerated File System for Flash Storage
  - Reader Pass-throughput Semaphore to improve the concurrency control
  - File cell to scale the accesses of in-memory data structures
  - Mlog to parallel the persistence functions

- Performance evaluation
  - Performs significantly better than existing Linux file systems
  - Achieve near-optimal performance for some file operations (i.e., append write and create).

Source code: https://github.com/thustorage/max

Q&A

Thank You!

Max: A Multicore-Accelerated File System for Flash Storage

Xiaojian Liao, Youyou Lu, Erci Xu, Jiwu Shu

Tsinghua University

liao-xj17@mails.tsinghua.edu.cn