DeLFS: A Decentralized Log-Structured File System for Manycores

Yongseok Son



Systems and Storage Lab
Chung-Ang University

Motivation

Modern data-intensive applications

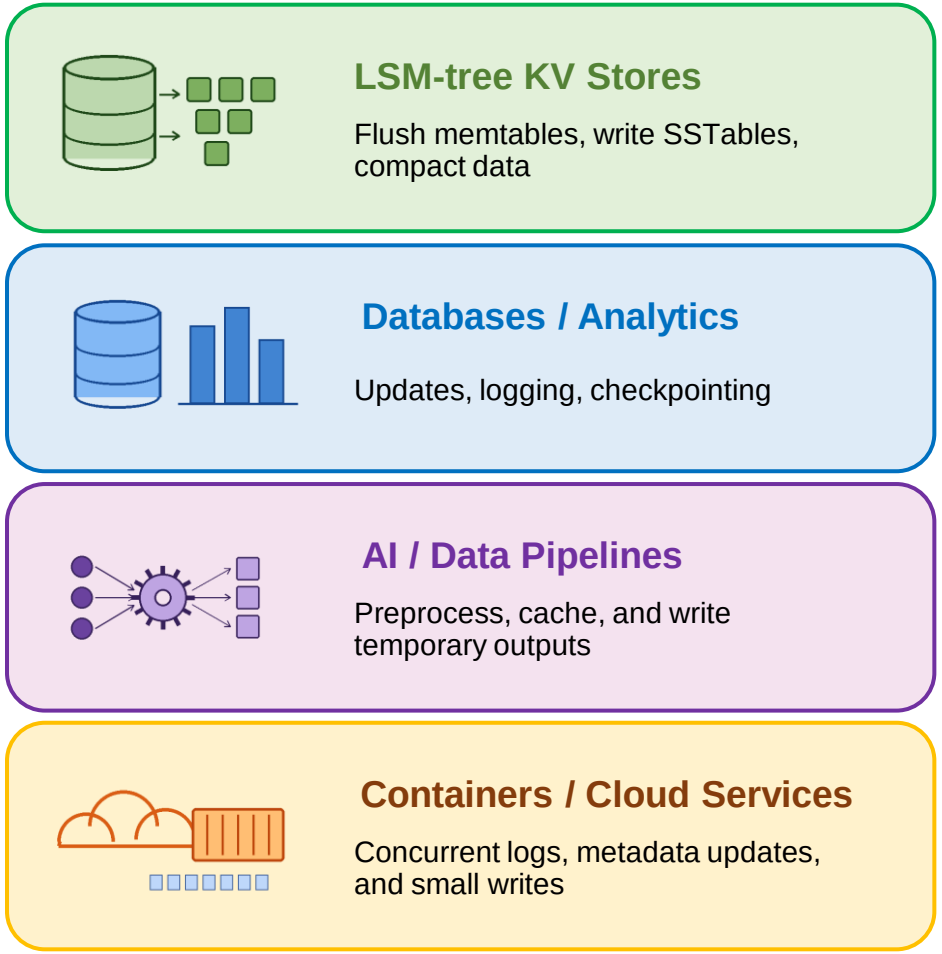
- **Key-value stores** [HATS (FAST'26), DecouKV (ATC'25)]
  - Flush memtables, write SSTables, and repeatedly compact data
- **Databases and analytics** [Mako (OSDI'25)]
  - Generate updates, transaction logs, and periodic checkpoints
- **AI and data pipelines** [Seneca (FAST'26), AITURBO (FAST'26)]
  - Preprocess data, manage caches, and write temporary outputs
- **Containers and cloud services** [FlacIO (FAST'25), CoFS (FAST'26)]
  - Generate concurrent logs, metadata updates, and large volumes of I/O traffic

High I/O concurrency

- Modern applications and services issue I/O operations concurrently

Software bottlenecks on fast hardware

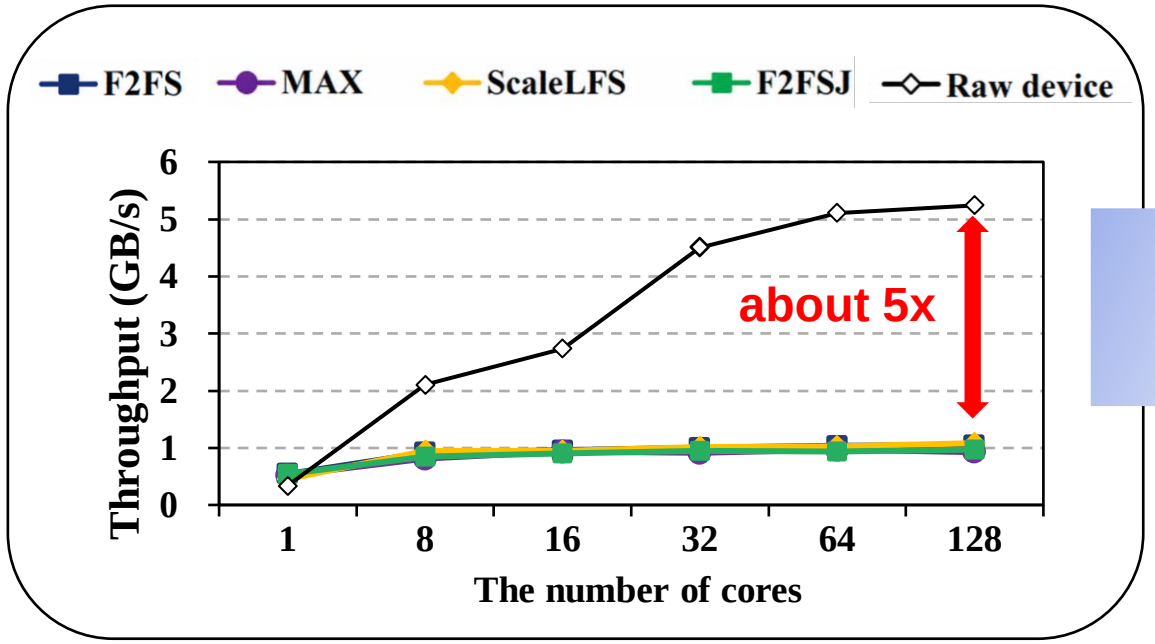
- Many-core CPUs and NVMe SSDs provide high parallelism
- However, the file system can become the bottleneck before the device is saturated

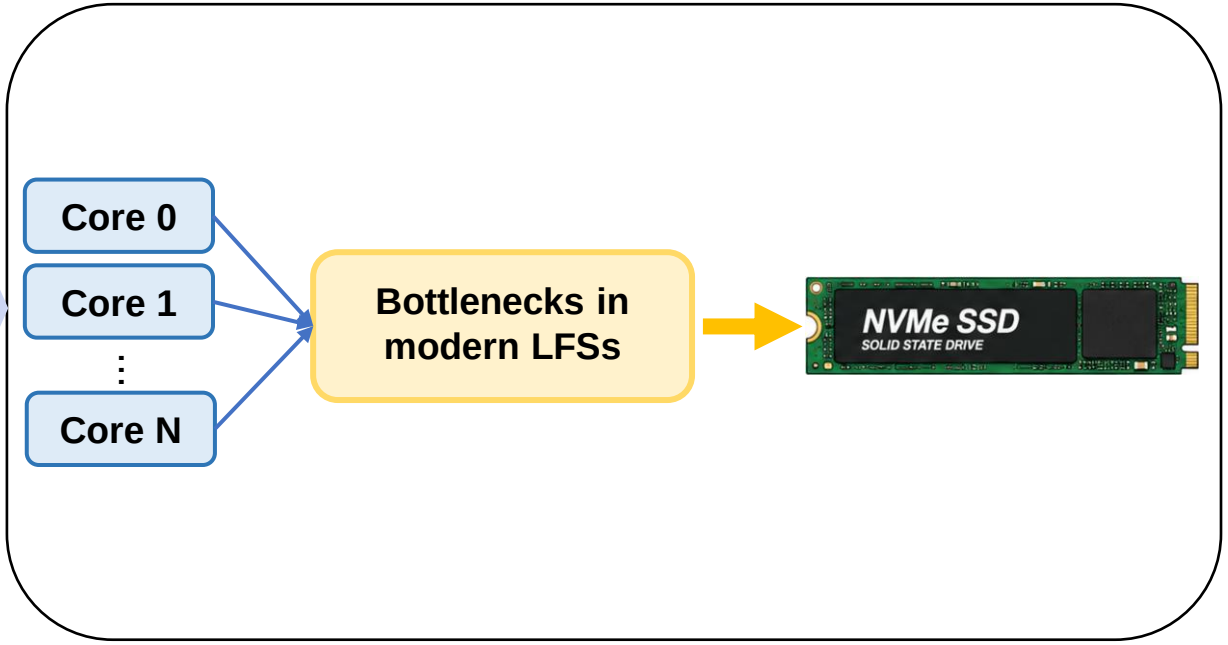


Motivation

- **Raw device scales well**
  - Raw-device throughput reaches about 5 GB/s

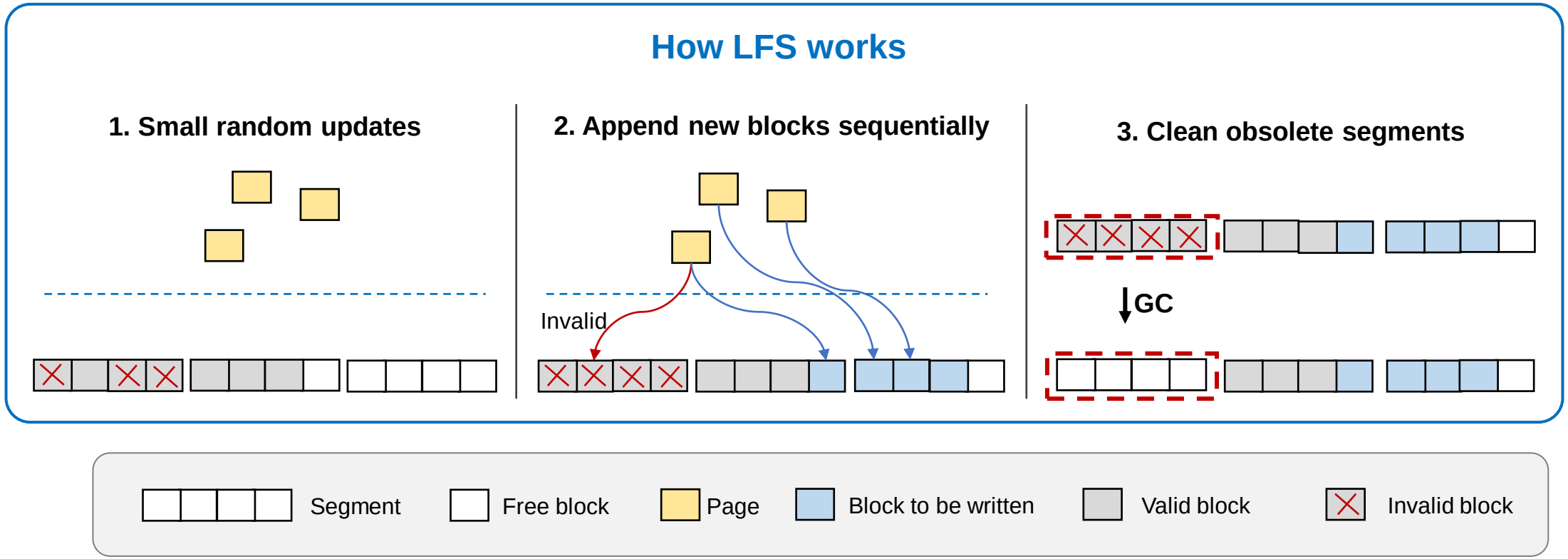
- **Modern LFSs exhibit a scalability gap on many cores**
  - F2FS [FAST'15], MAX [ATC'21], ScaleLFS [FAST'25], and F2FSJ [OSDI'25] remain near 1 GB/s





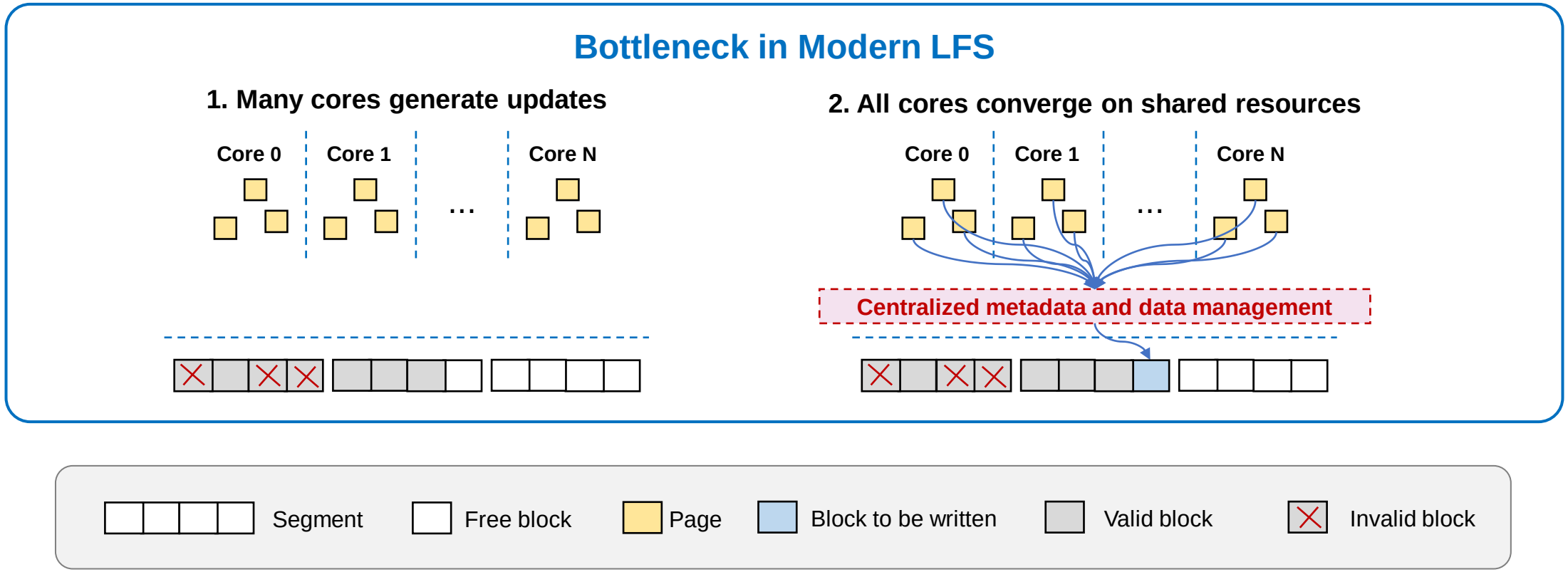
Background: How LFS Works

- **LFS appends updates instead of overwriting in place**
  - New versions are written sequentially, old versions become invalid, and obsolete segments are later reclaimed



Analysis: Centralized Metadata and Data Management

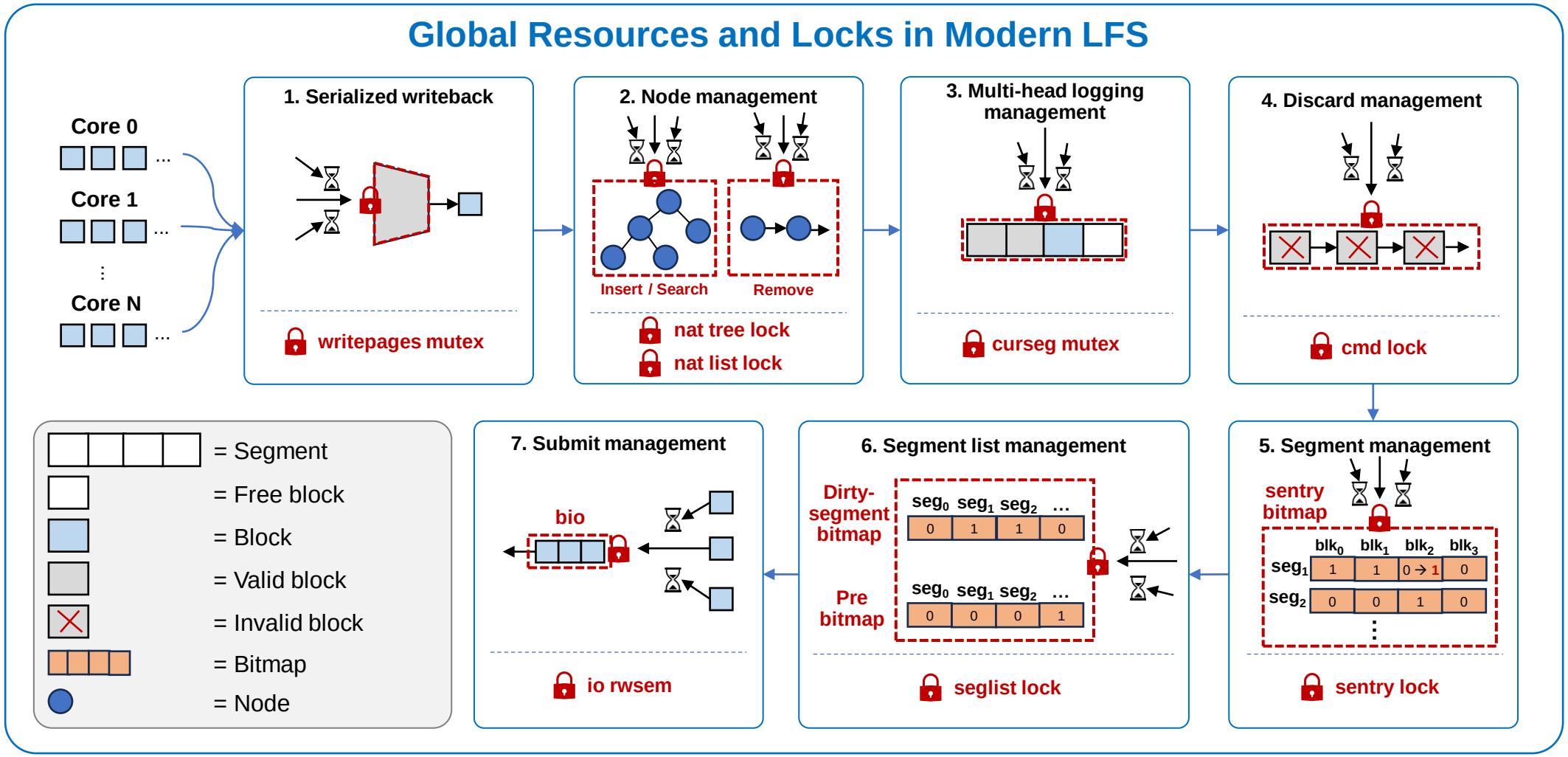
- **Concurrent and parallel updates contend on centralized metadata and data management**
  - Multiple file operations are serialized across many cores



Analysis: Scalability Obstacles

What are the scalability obstacles in modern LFSs?

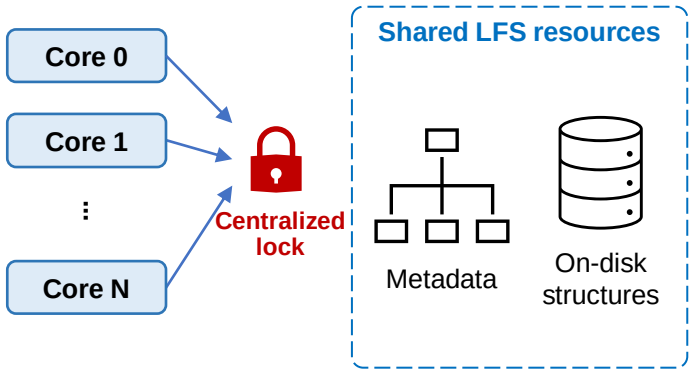
- Modern LFSs suffer from multiple **centralized resources and locks** in metadata and data management



Design: Key Idea

1. Existing Problem: Centralized LFS

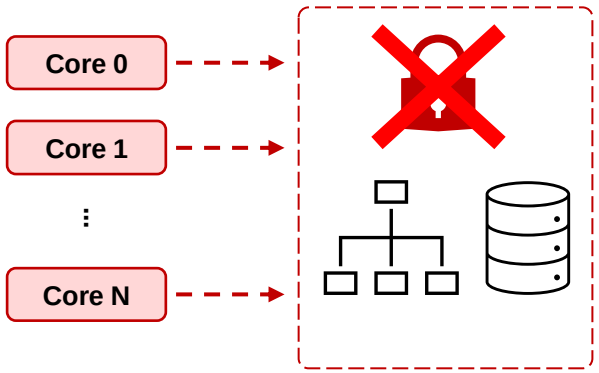
Many centralized resources are protected by their locks with large critical sections



Locks ensure **correctness** and **robustness**, but cause **contention and limit scalability on many-core** systems.

2. Naive approach: Remove locks

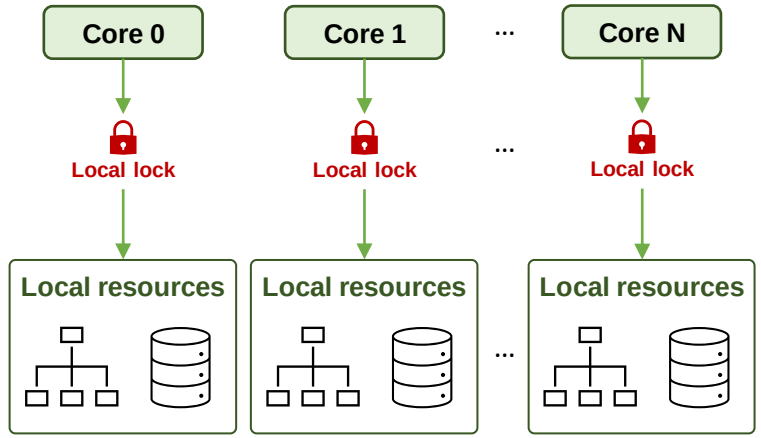
Simply removing locks seems attractive, but it is hard and unsafe



- Hard to maintain **correctness**
- Risk of **inconsistency and crashes**
- May be less **robust**

3. Our idea: Decentralized LFS

Split large resources and their locks, and distribute lock ownership across cores



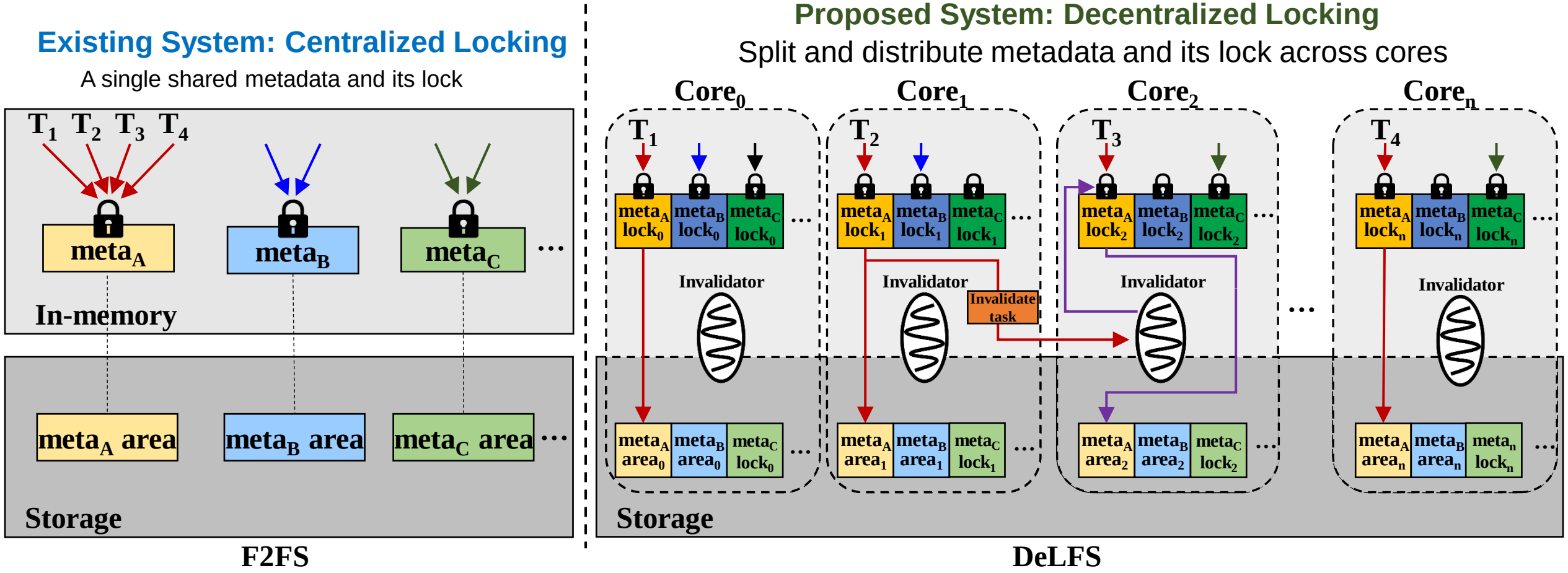
- Maintains **correctness** and **robustness**
- Reduces **contention** significantly
- Improves **scalability** on many-core systems

Design: Key Idea

Decentralized Locking
"Still locking, but smarter."

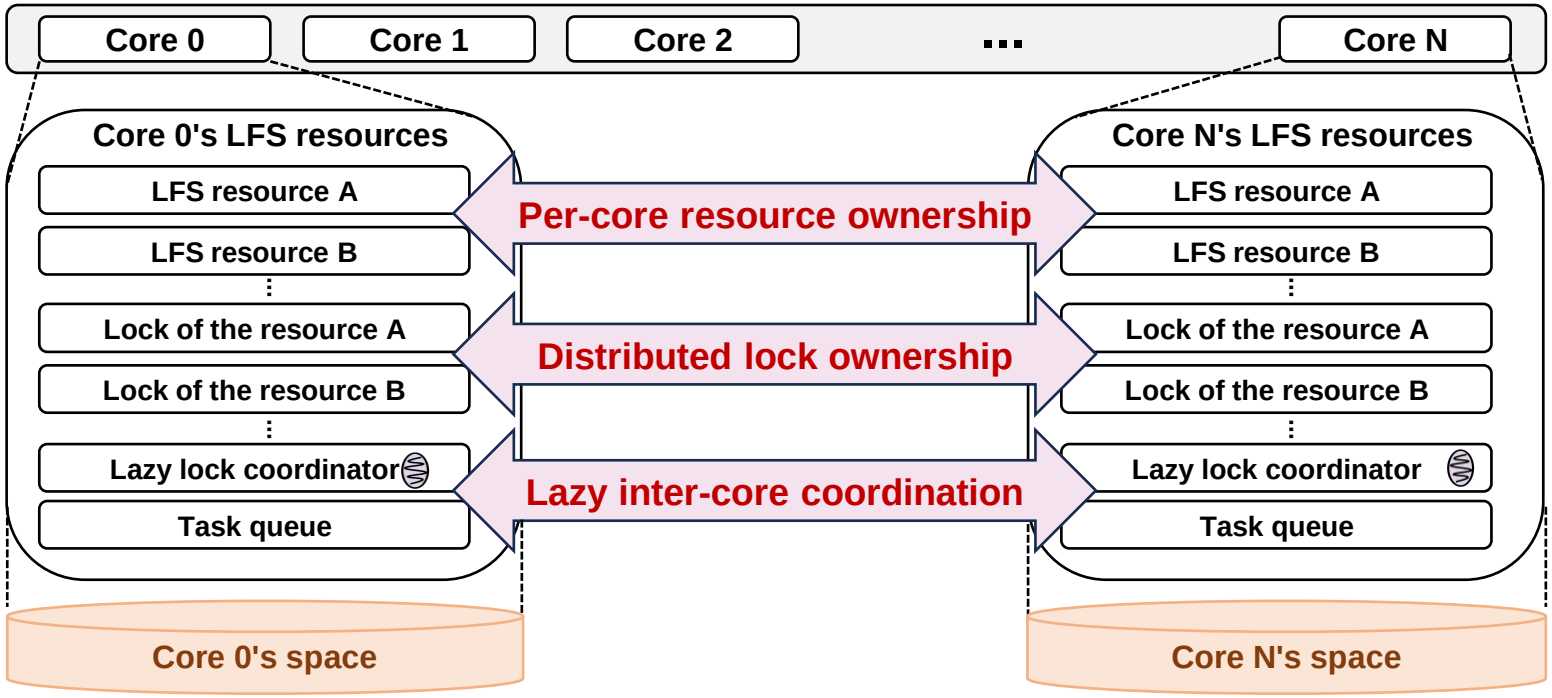
DeLFS

- Key idea is to leverage the potential of distributing lock ownership across cores safely, enabling a holistic, decentralized LFS architecture



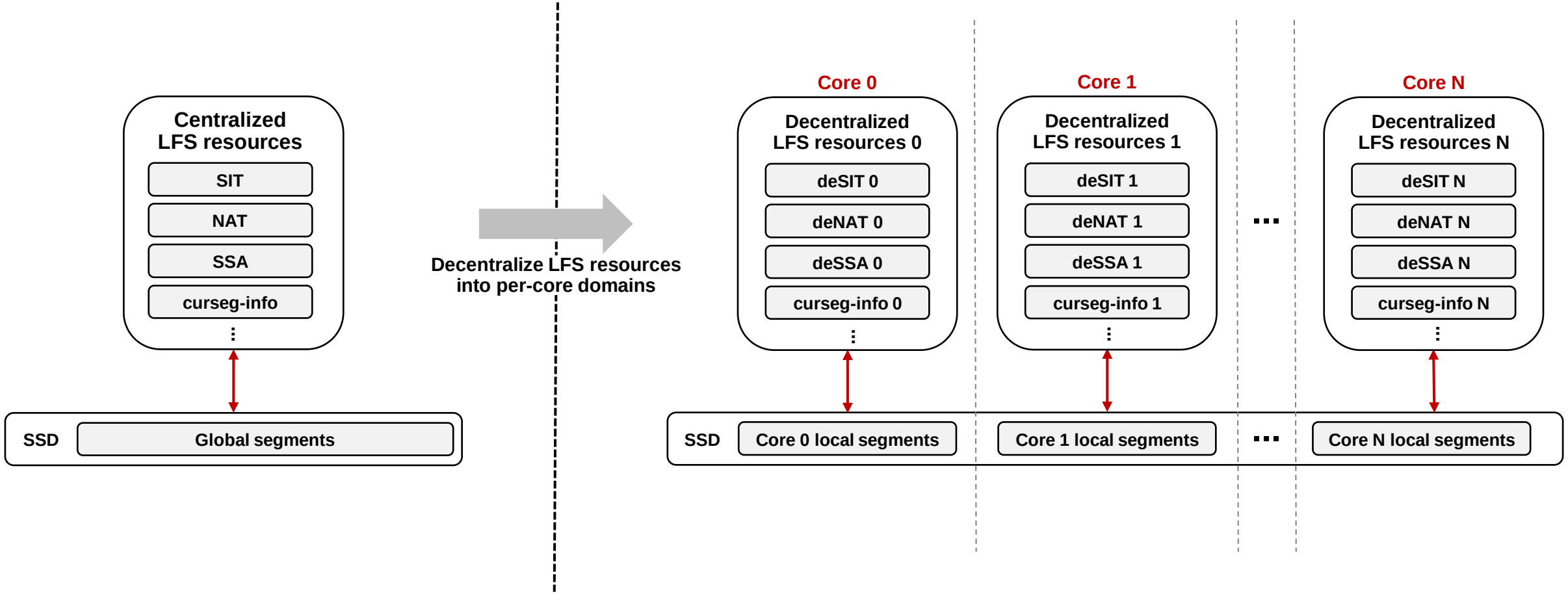
Design: Overall Architecture of *DeLFS*

- ***DeLFS***
  - A fully **decentralized metadata/data organization and its management**
  - Protected by **decentralized locking** and orchestrated by **lazy lock coordination** across cores
- ***DeLFS* consists of the following three techniques**
  - Decentralized organization and management
  - Decentralized and decoupled locking
  - Lazy lock coordination



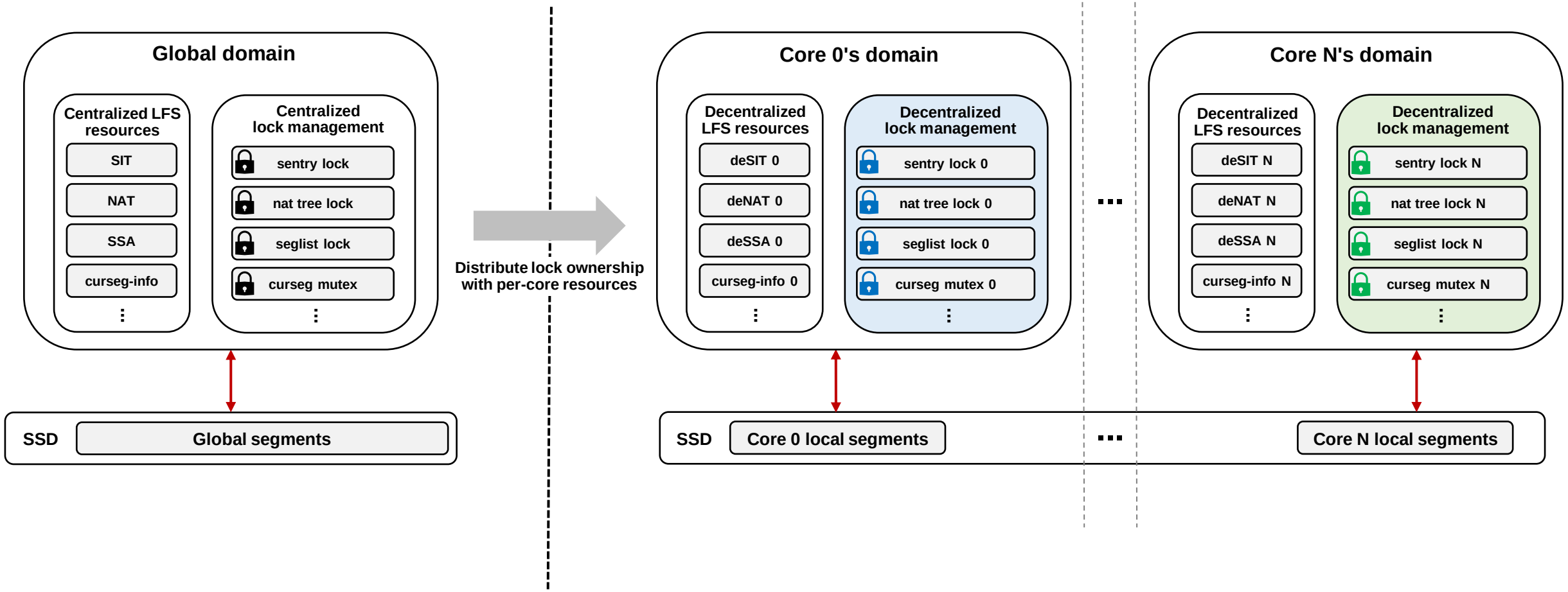
Design: Decentralized Organization and Management

- ***DeLFS* decentralizes LFS resources into per-core domains**
  - **Metadata and data resources are decentralized** so each core primarily manages its own local resources



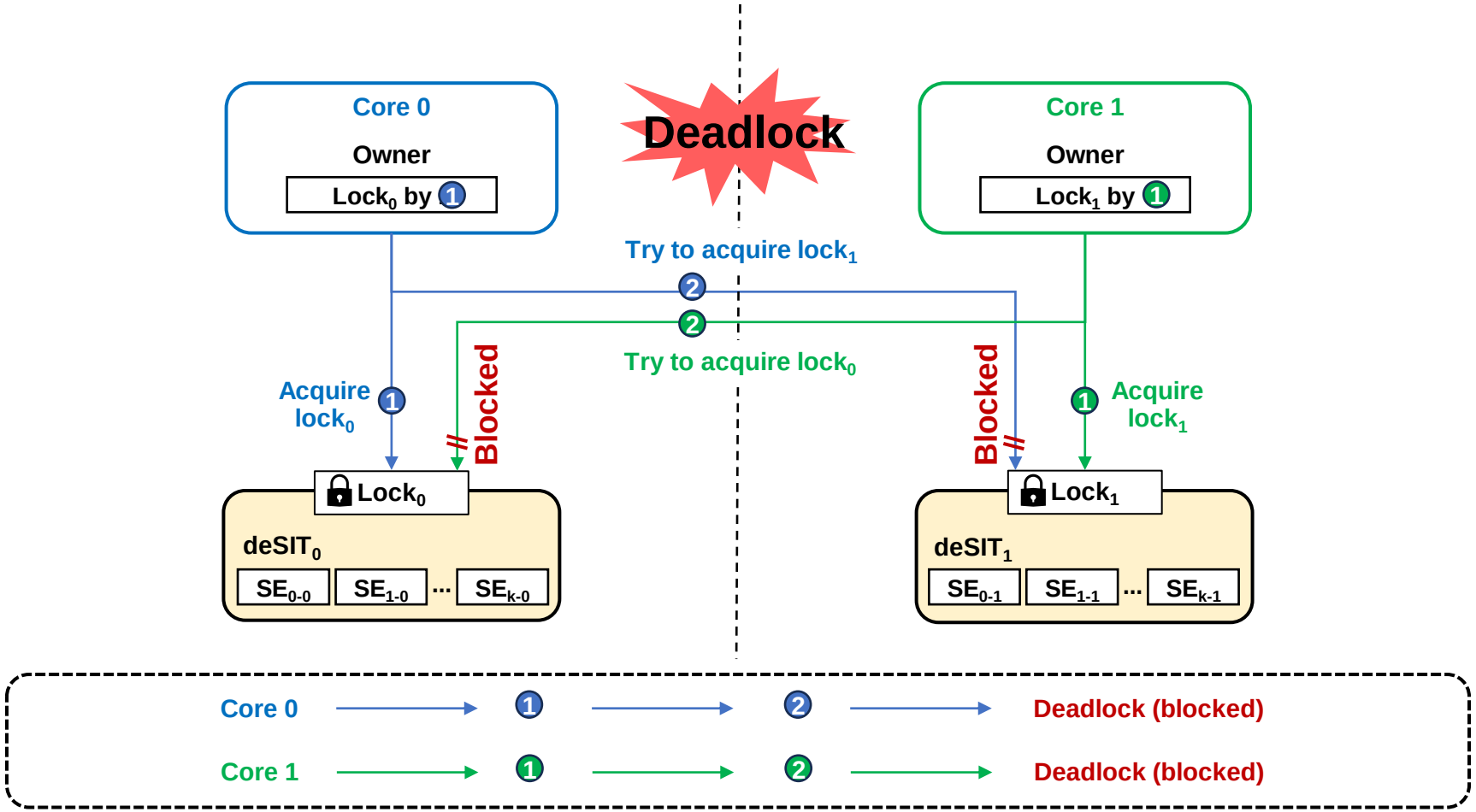
Design: Decentralized and Decoupled Locking (DDL)

- **Locks follow resource ownership, rather than remaining centralized**
  - Each core primarily acquires locks on its own decentralized resources
  - **Cross-core accesses still require a remote lock**



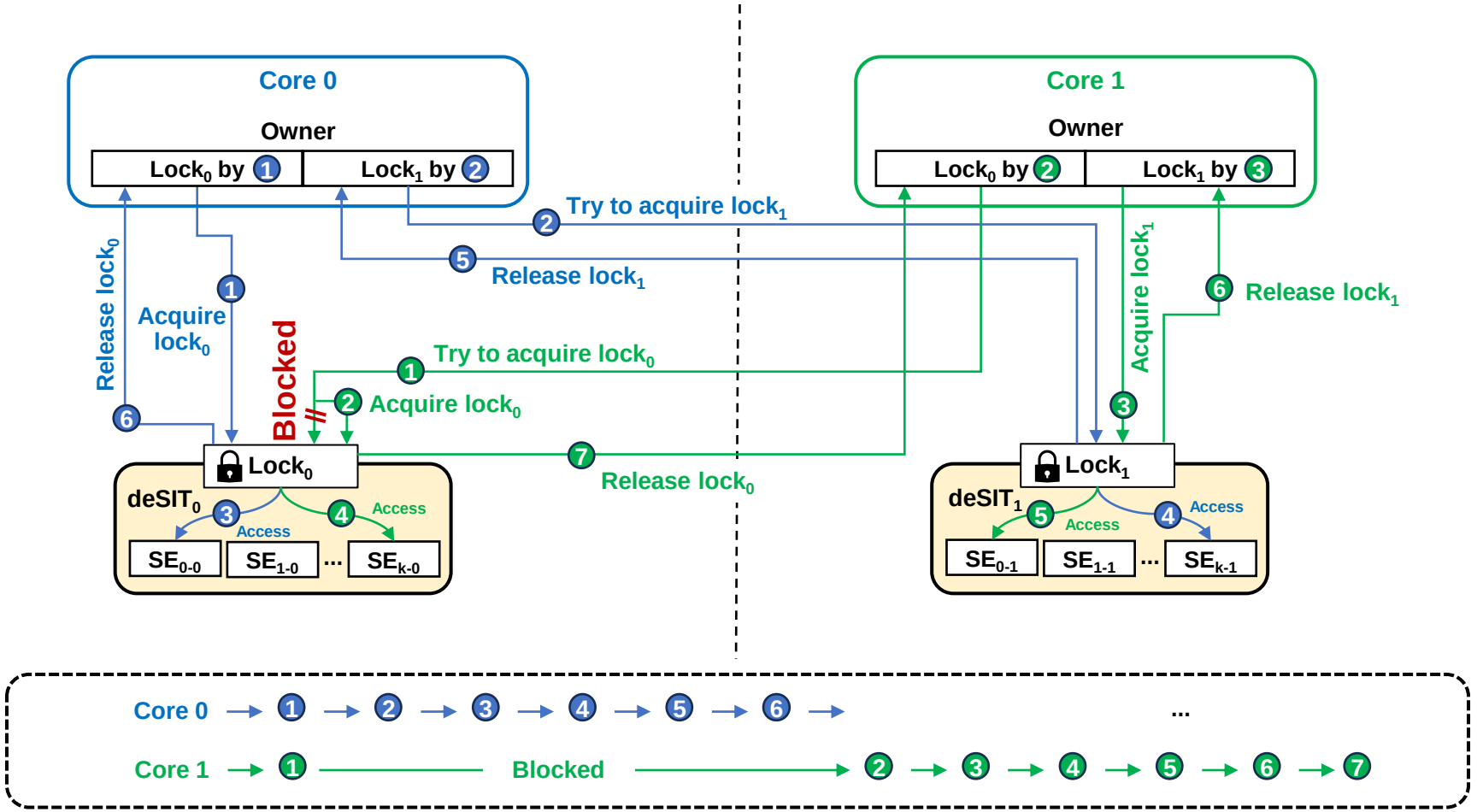
Design: Decentralized and Decoupled Locking (DDL)

- **Naive decentralized locking can cause deadlock**
  - An overwrite requires valid processing on a local SIT entry and invalid processing on a remote SIT entry



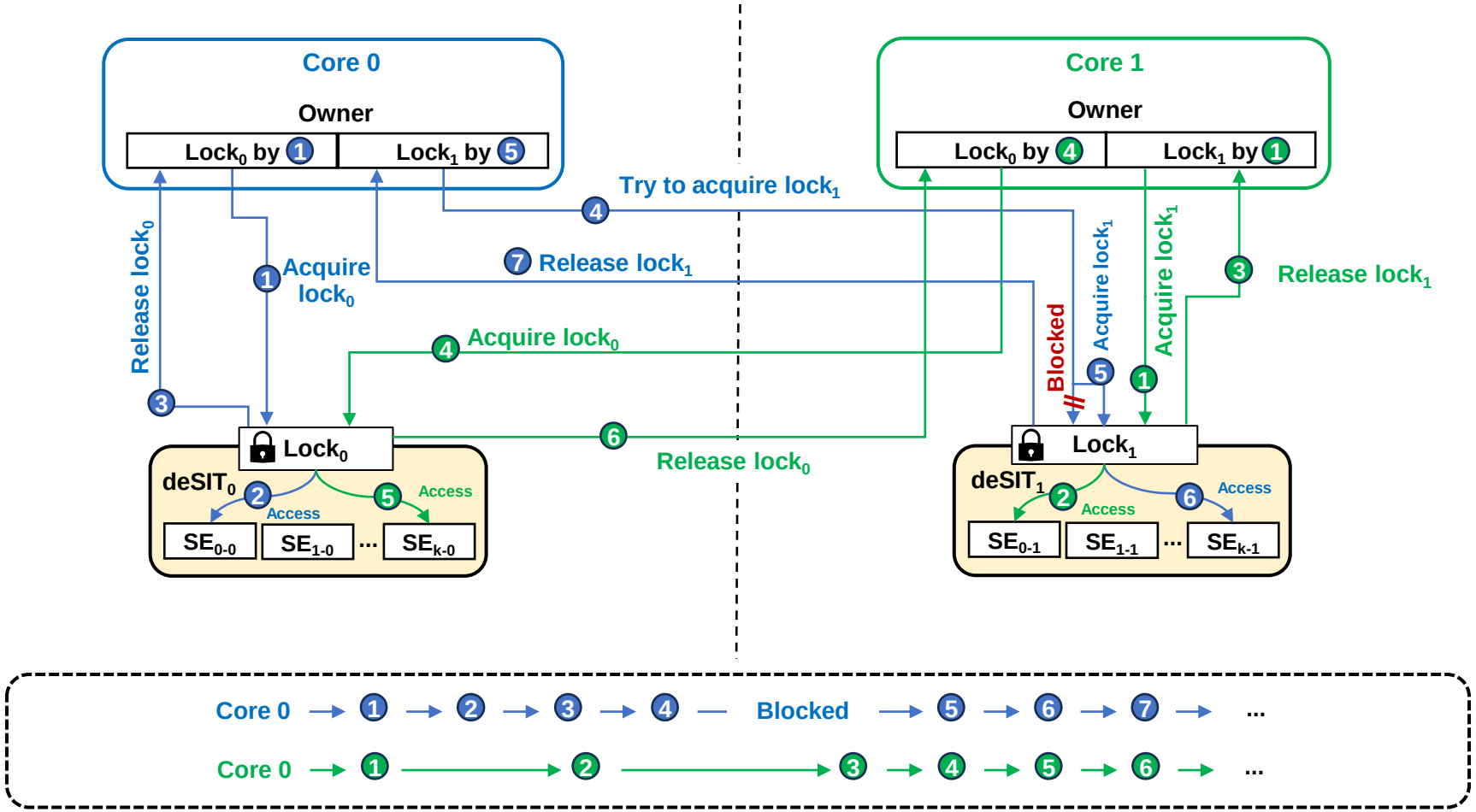
Design: Decentralized and Decoupled Locking (DDL)

- **Strict ordering acquires locks from lower-numbered cores first**
  - This prevents deadlock, but **introduces blocking between cores**



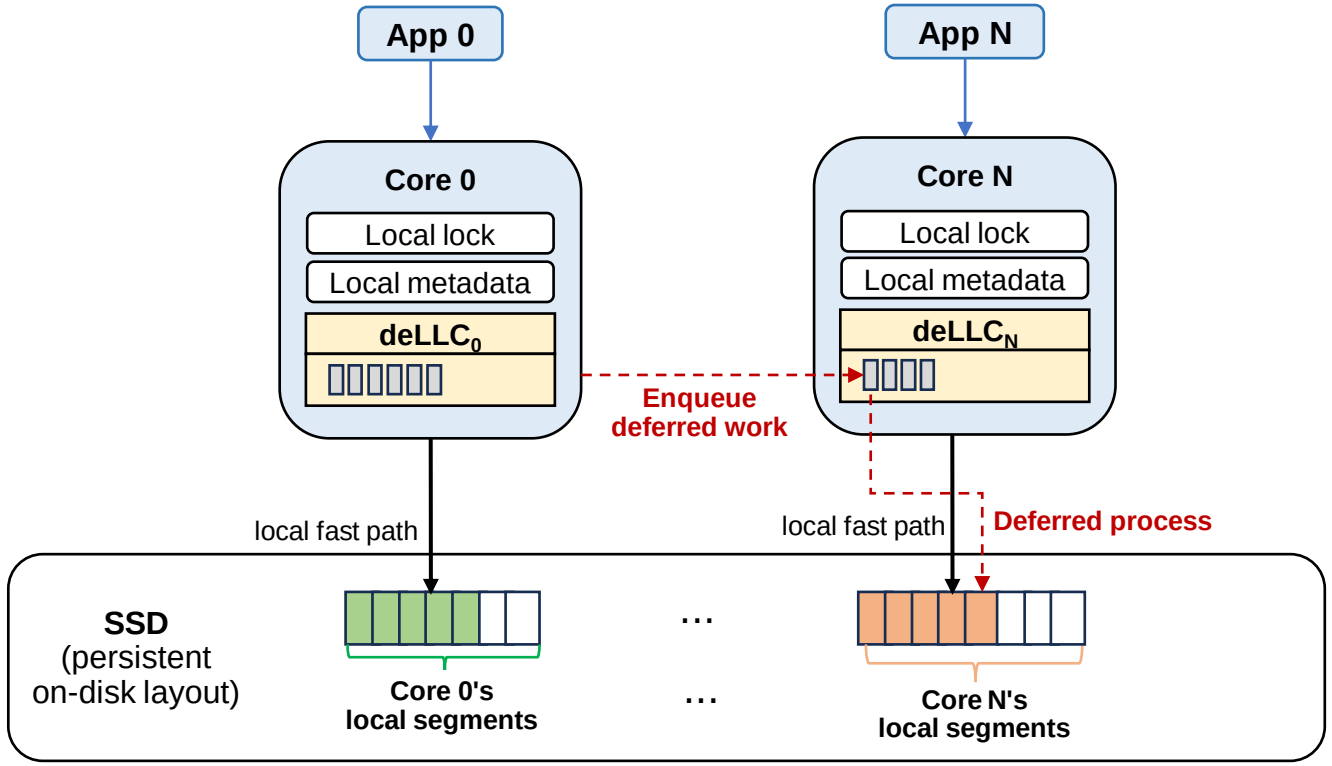
Design: Decentralized and Decoupled Locking (DDL)

- ***DDL* decouples valid and invalid processing into separate lock-update-unlock sequences**
  - Each core **never holds more than one lock** of the same type at a time



Design: Lazy Lock Coordination

- **Lazy lock coordination separates the critical path from deferrable invalid processing**
  - Decentralized lazy lock coordinator (*deLLC*) performs invalid processing for pages in remote spaces on behalf of the application thread

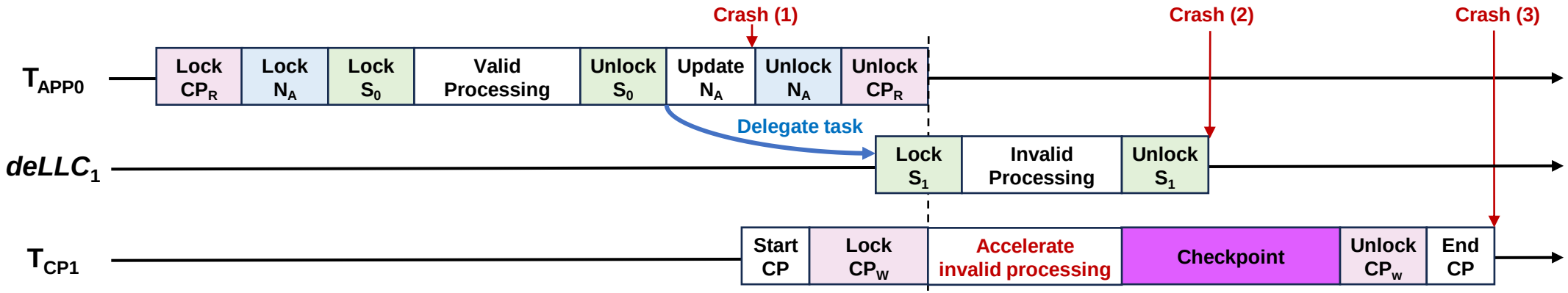


Design: Lazy Lock Coordination

Crash consistency with *deLLC*

- The checkpoint waits for and cooperatively processes all pending delegated tasks
- This ensures that deferred invalid processing completes before checkpointing

$CP_{R/W}$	Checkpoint reader/writer lock
N_A	Node A
S_n	sentry_lock$_n$
CP	Checkpoint



In-memory consistency

Once the node points to the new block, the old block is no longer referenced

Crash consistency (on-disk recovery)

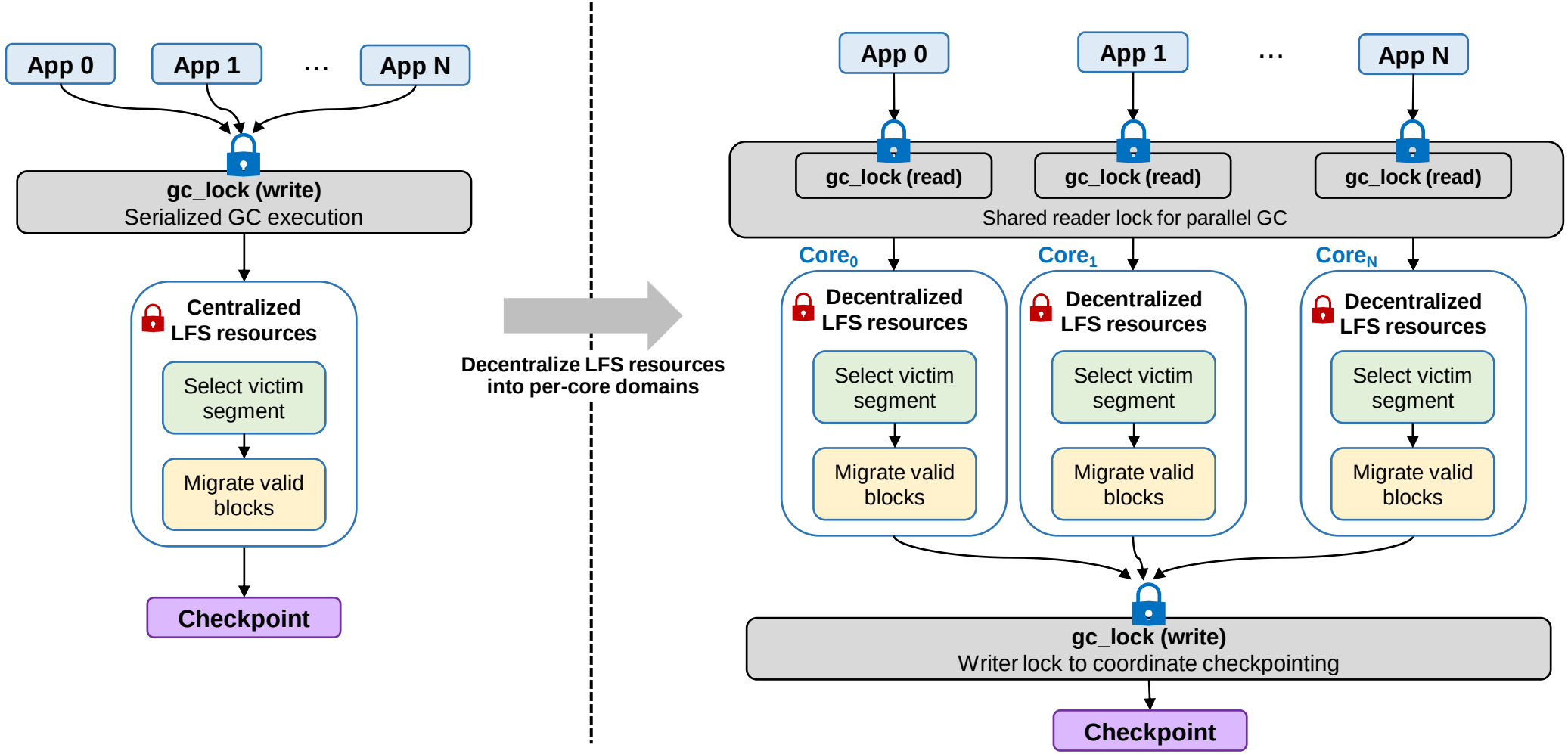
Crash (1)
Checkpoint is not completed
Recovery restores the previous state

Crash (2)
Checkpoint is not completed
Recovery restores the previous state

Crash (3)
Checkpoint is completed with the commit record
Recovery restores the newly updated node and SIT entries

Design: Decentralized Garbage Collection (GC)

- **Each core performs the GC procedure based on its own decentralized resources**
  - *DeLFS* leverages the GC reader lock during GC execution and the GC writer lock at checkpoint time



Evaluation: Experimental Setup

Platform Setup

OS	Ubuntu 22.04 LTS
Kernel	Linux 6.6.8
CPU	2 × AMD EPYC 7713 64-core processors (Totally 128 physical cores, HT disabled)
Memory	DDR4 96GB
SSD	FireCuda 530 2TB

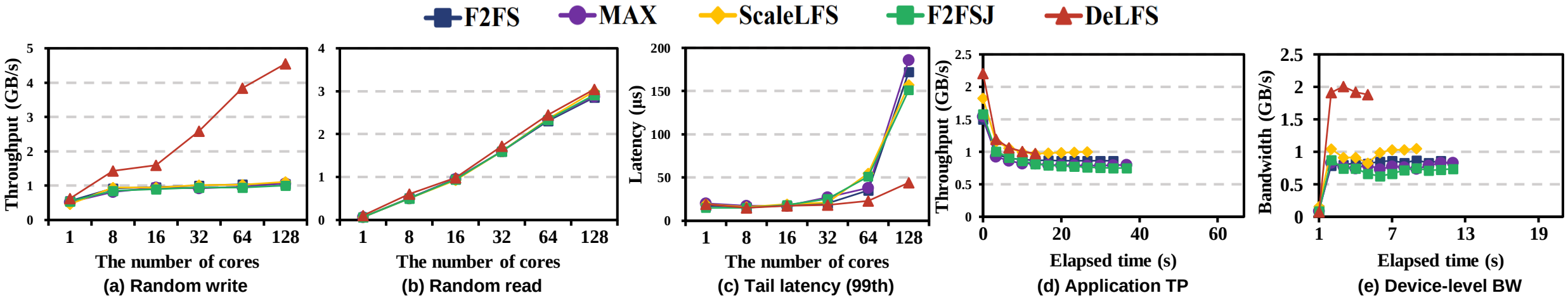
Workloads

- **Micro-benchmark**: FIO
- **Macro-benchmark**: Fileserver, Varmail, Videoserver
- **Real-world application**: RocksDB

Evaluation: Micro-benchmark

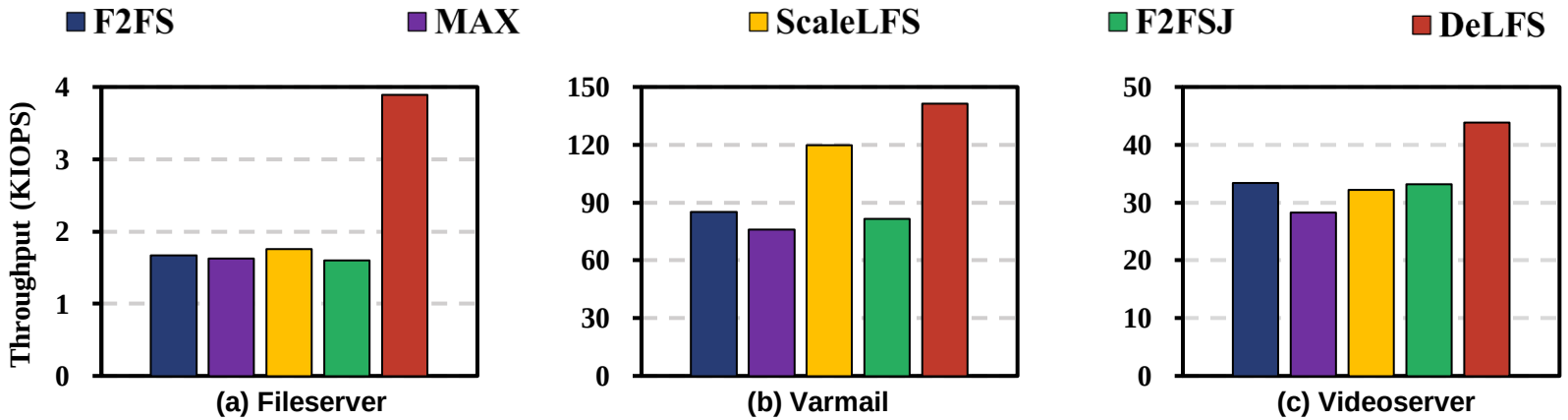
Throughput, Latency, and GC

- **Scalability with number of cores**
  - *DeLFS*: **Scales well** up to 128 cores
    - ✓ **Up to 4.50× higher throughput**
  - LFSs: Saturates at 1 GB/s
- **Latency with number of cores**
  - *DeLFS*: **Maintains stable low latency**
    - ✓ **Up to 3.73× lower tail latency**
  - LFSs: Latency spikes significantly
- **GC performance**
  - *DeLFS* achieves up to 2.77× lower execution time and 2.32× higher device-level bandwidth



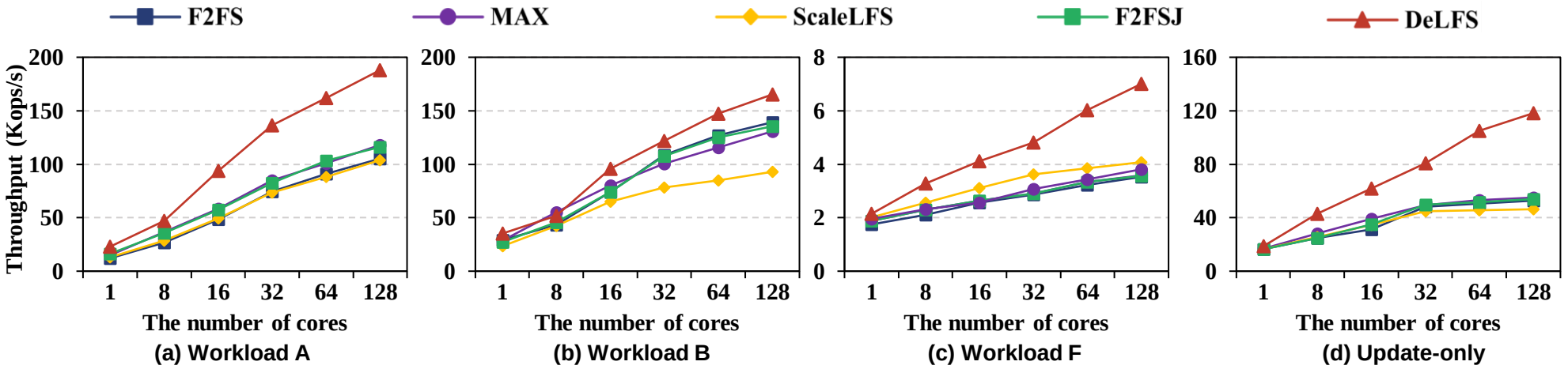
Evaluation: Macro-benchmark

- **Filebench workloads**
  - **Fileserver**
    - Write and data-intensive workload
    - Up to **2.43× higher throughput**
  - **Varmail**
    - Data-intensive and metadata-intensive workload
    - Up to **1.86× higher throughput**
  - **Videoserver**
    - Read-intensive workload
    - Up to **1.55× higher throughput**

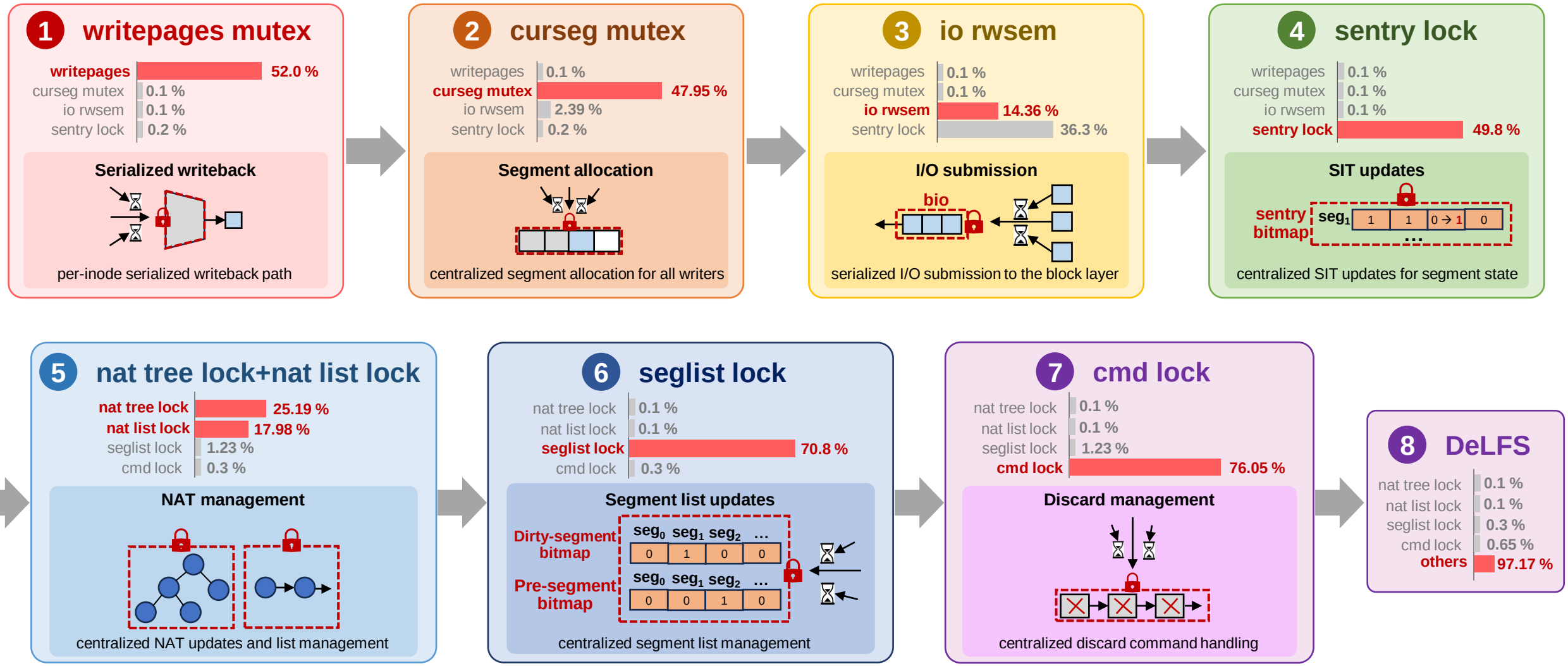


Evaluation: Real-world Application

- **RocksDB with YCSB workloads**
  - **Workload A (1:1 = read:update ratio)**
    - Up to **1.81× higher throughput**
  - **Workload B (9.5:0.5 = read:update ratio)**
    - Up to **1.77× higher throughput**
    - An inefficient write path can significantly impact read performance
  - **Workload F (1:1 = read:read-modify-write ratio)**
    - Up to **1.98× higher throughput**
    - The read-modify-write requests stress both the write and read paths
  - **Update-only (100% update)**
    - Up to **2.54× higher throughput**



Evaluation: Breakdown of DeLFS



Evaluation: Skewed Space Allocation

Available-space pressure

- We **pre-fill the device and run Fileserver** with continuous file creation/deletion.

Performance

- For the 75%, 50%, 25%, and 10% available space, *DeLFS* outperforms F2FS by 2.37×, 1.93×, 1.63×, and 1.17×, respectively
- *DeLFS* performance gradually decreases as the available space decreases because **local free-pool depletion incurs cross-core overheads**

Limited free space

- At 1% available space, both file systems show **almost the same performance** because the available capacity is severely limited

LFS	Metric	Available space				
		1.5 TB (75%)	1 TB (50%)	0.5 TB (25%)	0.2 TB (10%)	0.02 TB (1%)
F2FS	# of create/delete	128 K	135 K	132 K	128 K	37 K
	Total writes	1.16 TB	1.19 TB	1.18 TB	1.14 TB	0.75 TB
	Throughput	1574 ops/s	1650 ops/s	1627 ops/s	1578 ops/s	458 ops/s
DeLFS	# of create/delete	305 K	260 K	217 K	151 K	38 K
	Total writes	3.02 TB	2.60 TB	2.24 TB	1.57 TB	0.81 TB
	Throughput	3730 ops/s	3180 ops/s	2662 ops/s	1848 ops/s	468 ops/s

Conclusion

- **Key contributions**
  - **Decentralized organization**: decentralizes LFS resources into per-core domains
  - **Decentralized and decoupled locking**: eliminates global lock contention safely
  - **Lazy lock coordination**: separates critical and deferrable paths

- **Impact**
  - Improves both application I/O and GC I/O processing
  - Achieves up to **4.50× higher throughput** than existing LFSs
  - Delivers up to **2.00× higher sustained performance** than ScaleLFS

- **Please refer to the paper for further details**

- **DeLFS is open source now:**
  - https://github.com/syslab-CAU/DeLFS

Q&A

Thank you for your attention