Decentralized, Epoch-based F2FS Journaling with Fine-grained Crash Recovery

Yaotian Cui[1], Zhiqi Wang[1], Renhai Chen[2], Zili Shao[1]

[1]The Chinese University of Hong Kong, China

[2]College of Intelligence and Computing, Tianjin University, China

Outline

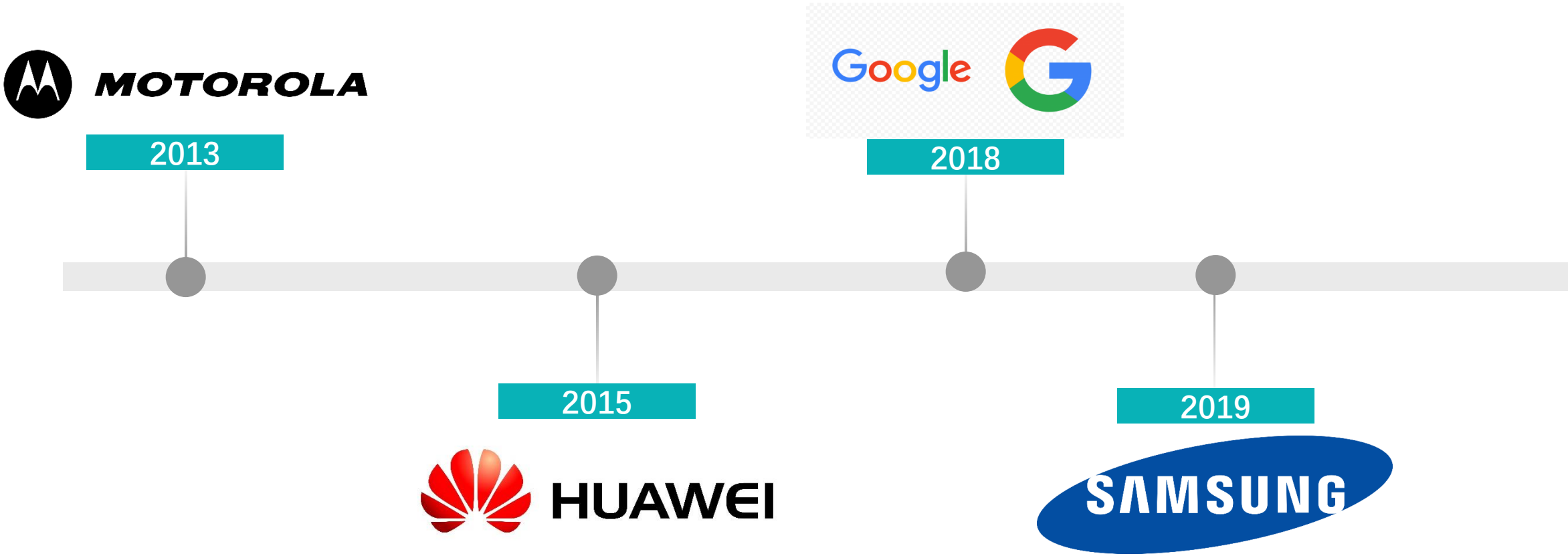
- Background
- Motivation
- Design
- Evaluation

Outline

- Background
- Motivation
- Design
- Evaluation

F2FS (Flash-Friendly File System)

- F2FS is widely used as filesystems in Android smart phones.



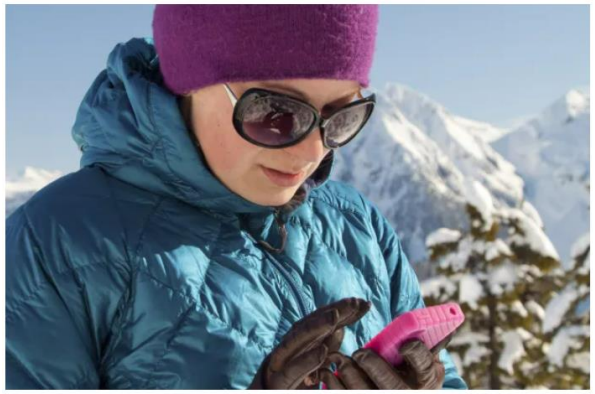
Crash Recovery is Vital

Low temperature shuts down Android phones.

COLD SNAP Millions of Android owners warned their phone could shut down in freezing temperatures – three tips to keep them running

Including a nifty trick that'll help your Android survive Arctic-level chills

Millie Turner, Technology & Science Reporter
Published: 4:52 ET, Jan 15 2024 | **Updated:** 4:56 ET, Jan 15 2024



Your phone gets chilly too (Picture: Getty)

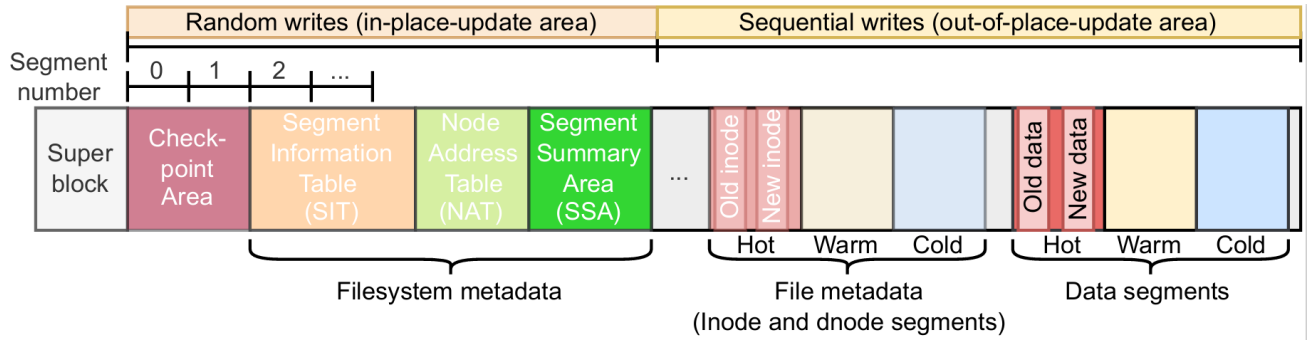
It's not just humans who aren't enjoying this icy Arctic blast – your smartphone can't cope in the cold either.

You may have already noticed it being glitchy, perhaps the touch screen has been behaving oddly, or more likely, your battery is **dying even faster than normal**.

Outdated applications / System upgrades cause phones reboot.

F2FS Overview

- F2FS manages storage space at the unit of segments (e.g. 2MB per segment).
- On-disk state.
  - In-place-update areas.
    - Checkpoint area.
      - Roll back crash recovery.
    - Filesystem metadata
      - Segment information table (SIT): Bitmap and free block number.
      - Node address table (NAT): Inode/dnode mapping table.
      - Segment summary area (SSA): Reverse mapping for garbage collection.
  - Out-of-place-update areas.
    - File metadata
      - Inode/Dnode.
    - File Data



F2FS Checkpointing

- In a crash, F2FS can roll back to a consistent state with its most recent checkpoint.

- F2FS checkpointing severely degrades the system performance.
  - Once triggered (via threshold/timeout), all dirty in-memory data and metadata will be persisted.
  - All file reads/writes are blocked.

Issues of F2FS Checkpointing

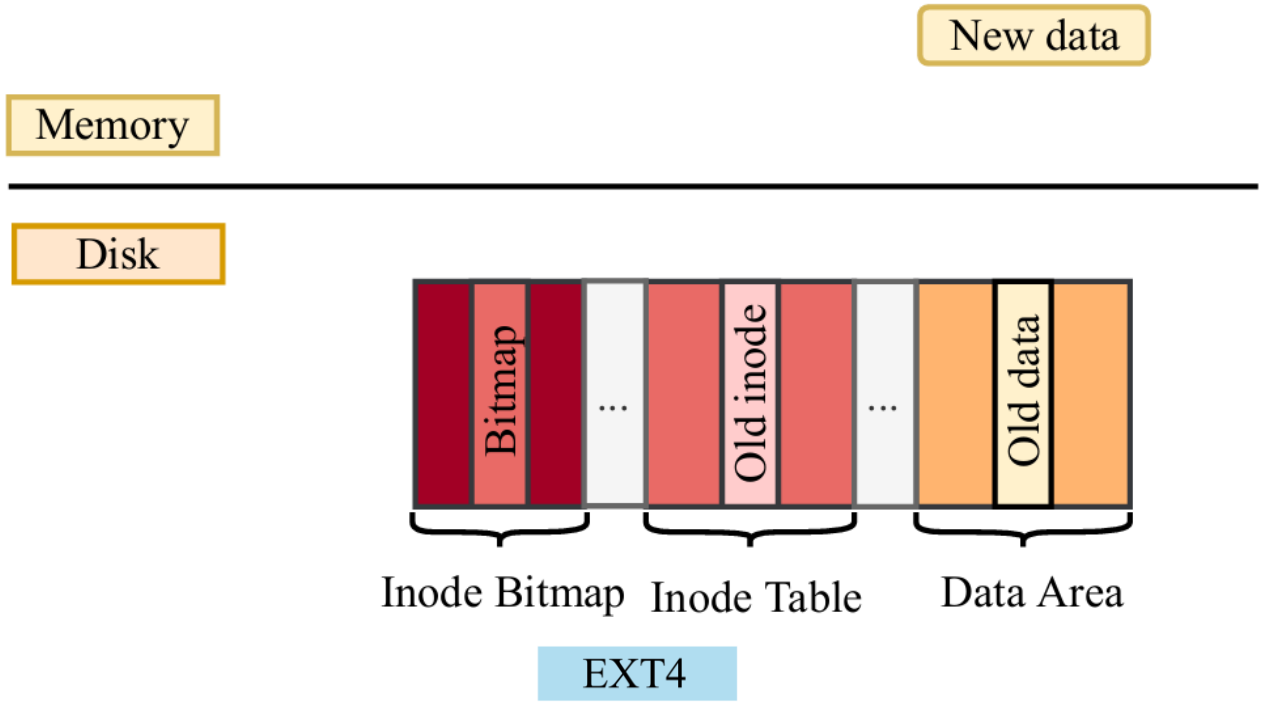
- Problem1: Time Overhead.
  - Long tail latency (e.g., P90 latency > 240ms) during checkpointing.
- Problem2: Data and Metadata Loss.
  - More data and metadata loss due to coarse-grained crash recovery.
- Problem3: Inconsistency with Roll-forward Recovery.
  - Data and metadata can be inconsistent in no-barrier and POSIX fsync modes.

Outline

- Background
- **Motivation**
- Design
- Evaluation

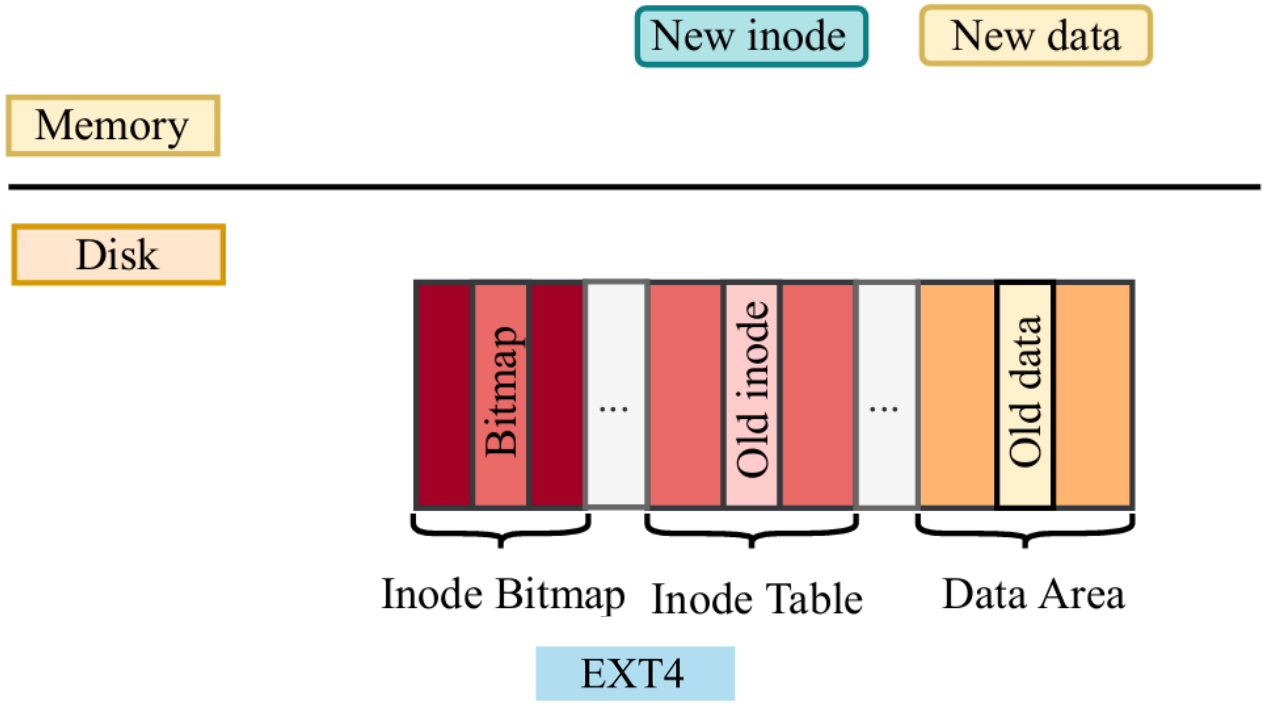
Journaling in In-place-update Filesystem (EXT4)

- Ordered journal mode
  - Data flushing strictly precedes committing metadata journals.
  - Only metadata are journaled.



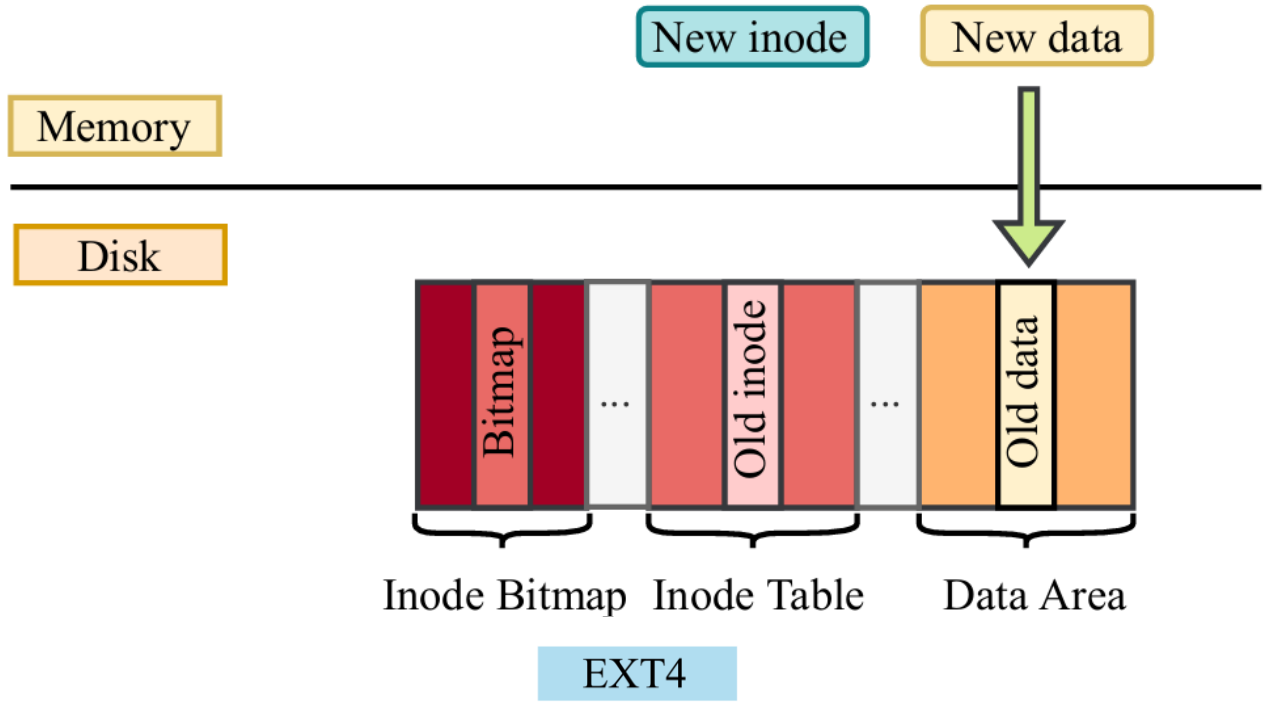
Journaling in In-place-update Filesystem (EXT4)

- Ordered journal mode
  - Data flushing strictly precedes committing metadata journals.
  - Only metadata are journaled.



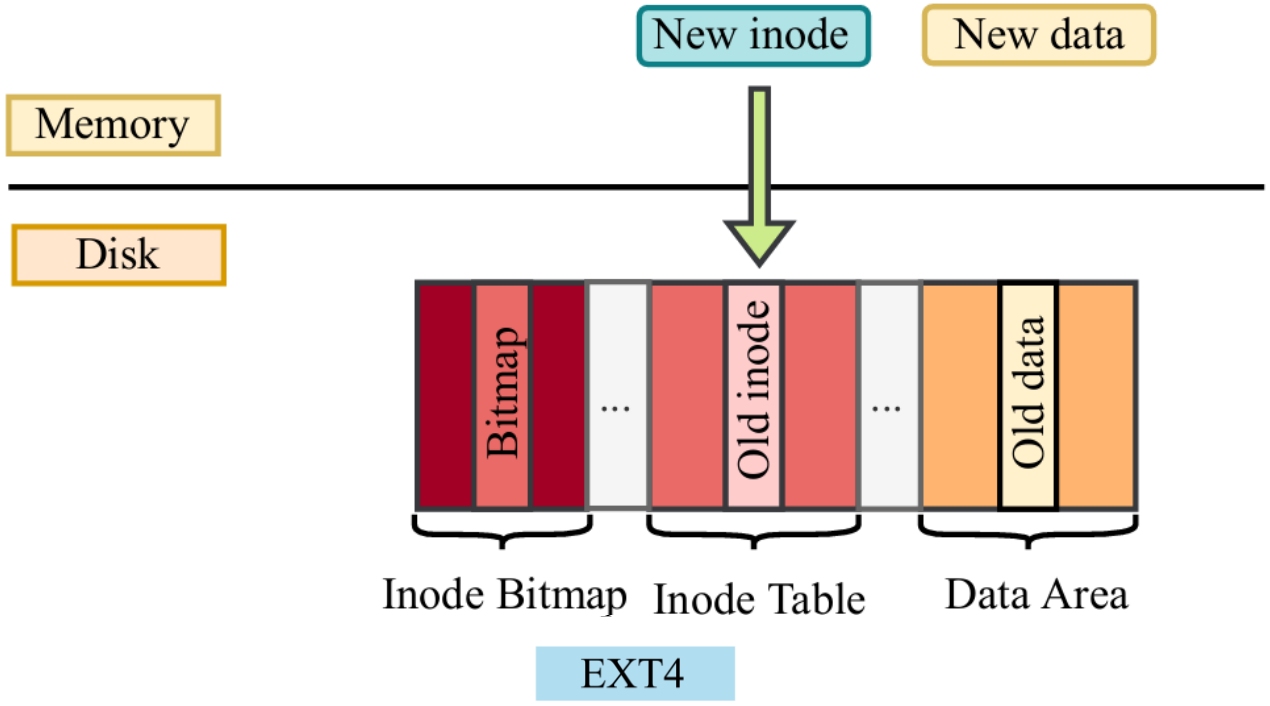
Journaling in In-place-update Filesystem (EXT4)

- Ordered journal mode
  - Data flushing strictly precedes committing metadata journals.
  - Only metadata are journaled.



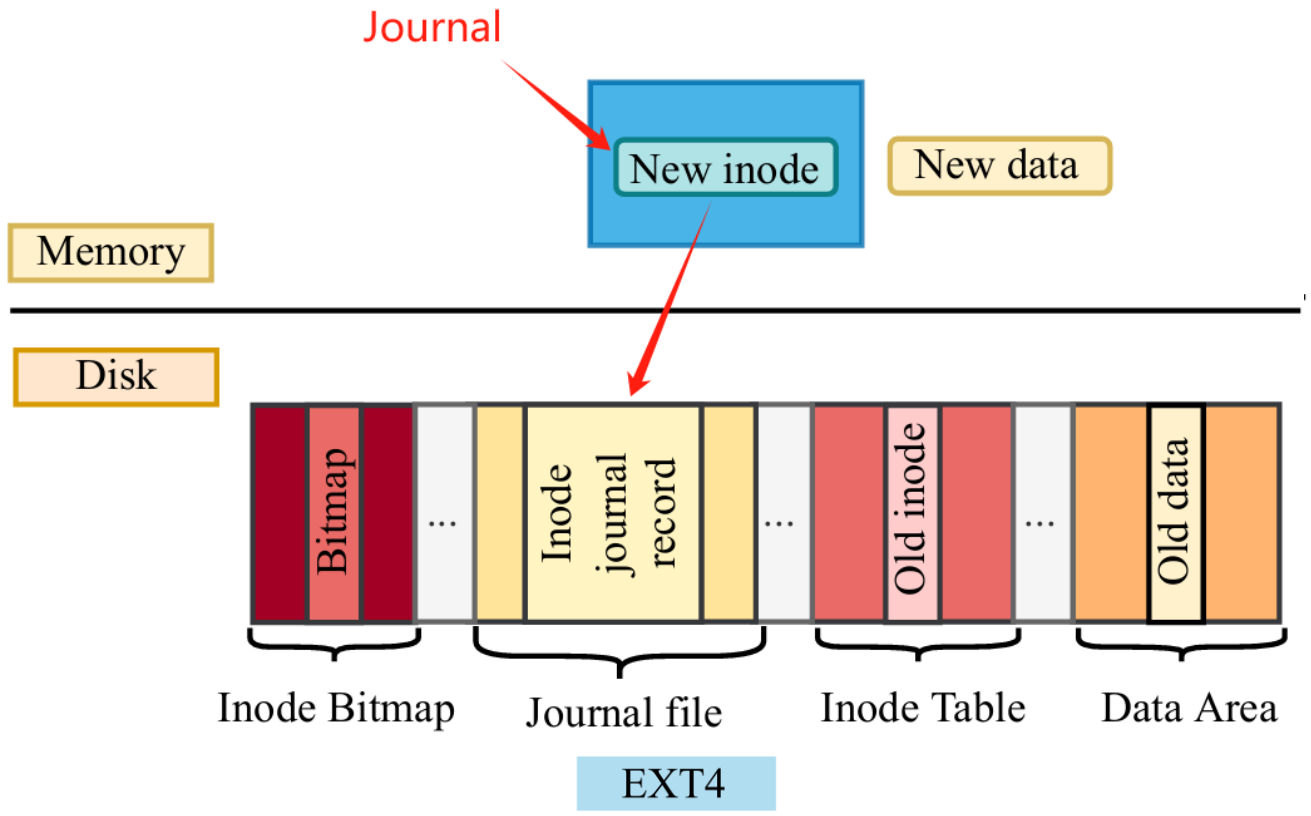
Journaling in In-place-update Filesystem (EXT4)

- Ordered journal mode
  - Data flushing strictly precedes committing metadata journals.
  - Only metadata are journaled.



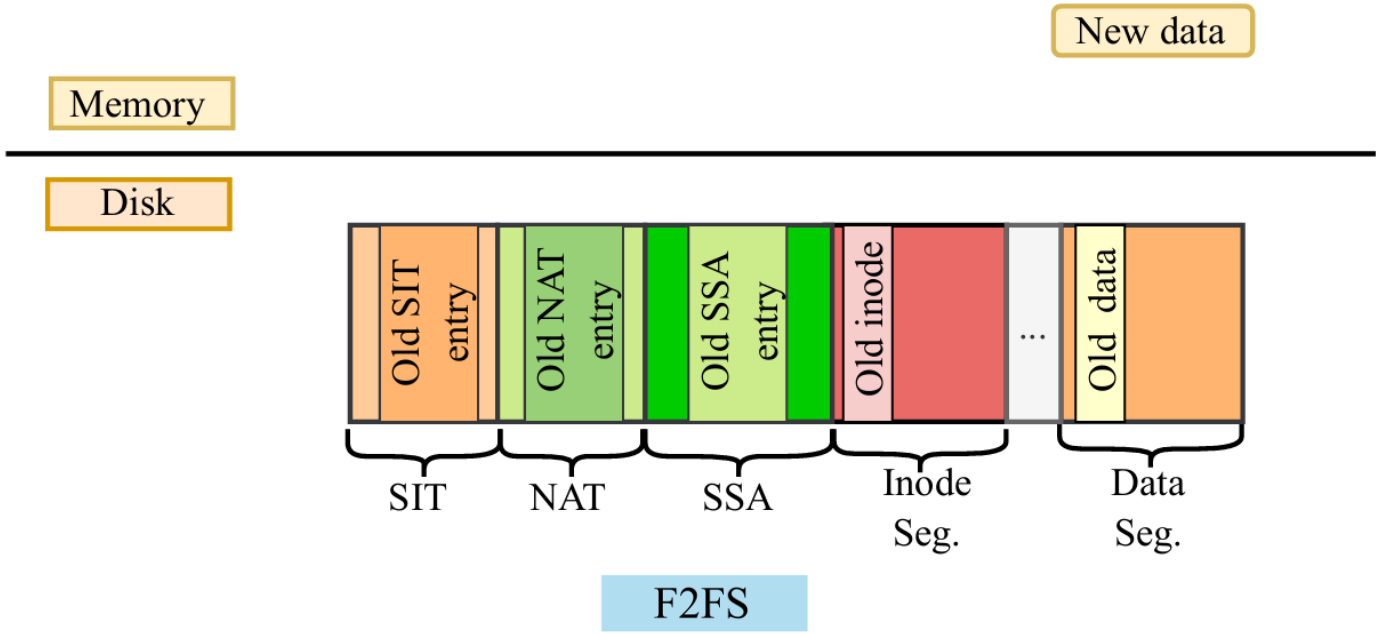
Journaling in In-place-update Filesystem (EXT4)

- Ordered journal mode
  - Data flushing strictly precedes committing metadata journals.
  - Only metadata are journaled.



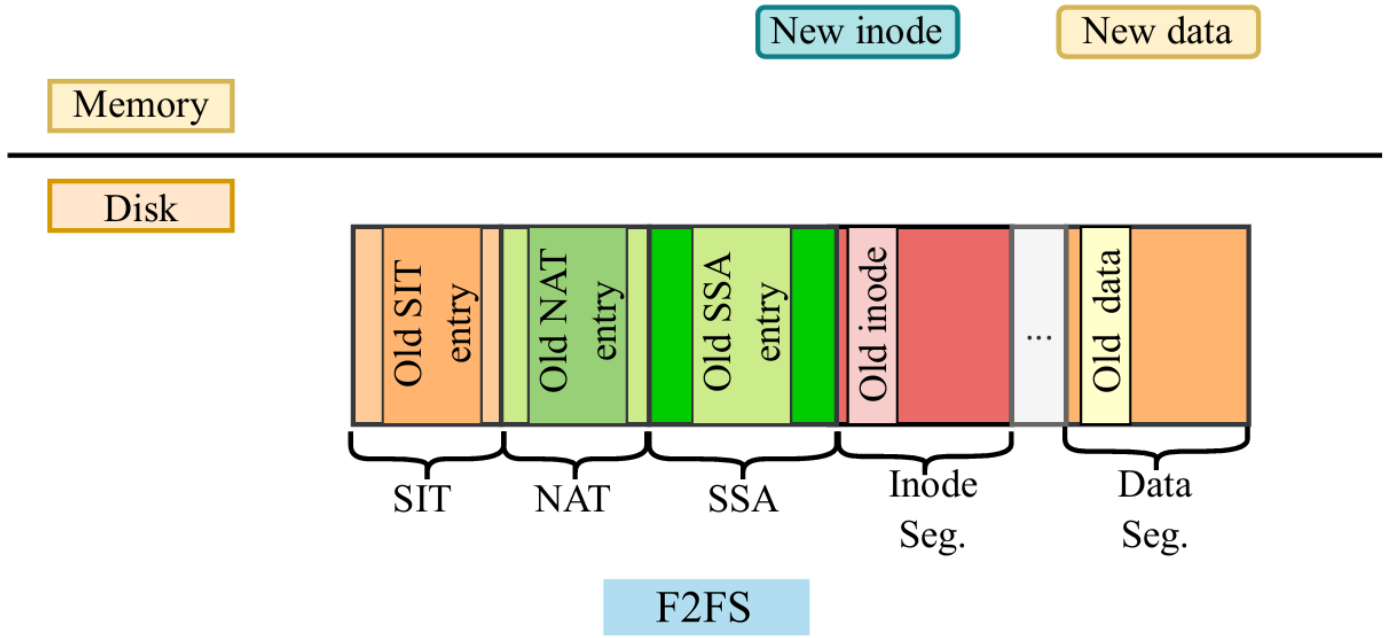
Journaling in F2FS (Out-of-place-update FS)

- Ordered journal mode
  - Need to journal both filesystem metadata and file metadata.



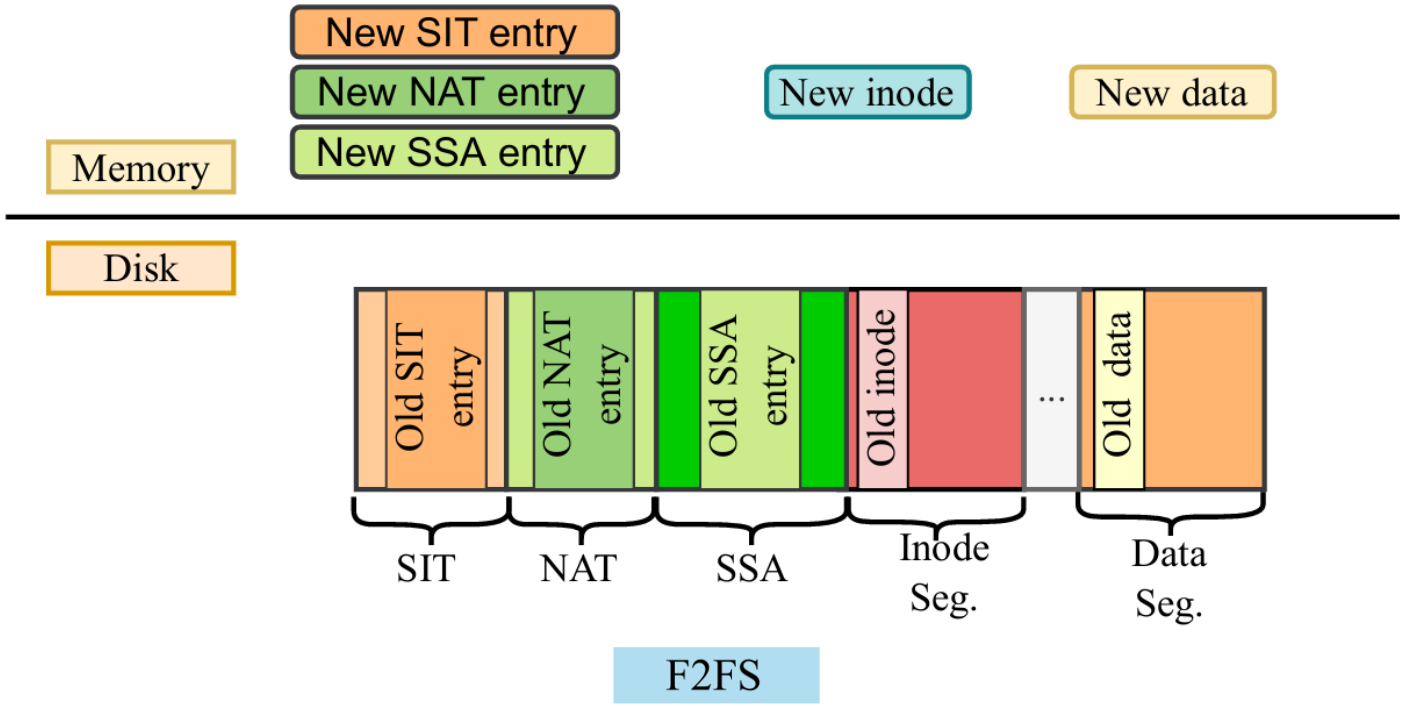
Journaling in F2FS (Out-of-place-update FS)

- Ordered journal mode
  - Need to journal both filesystem metadata and file metadata.



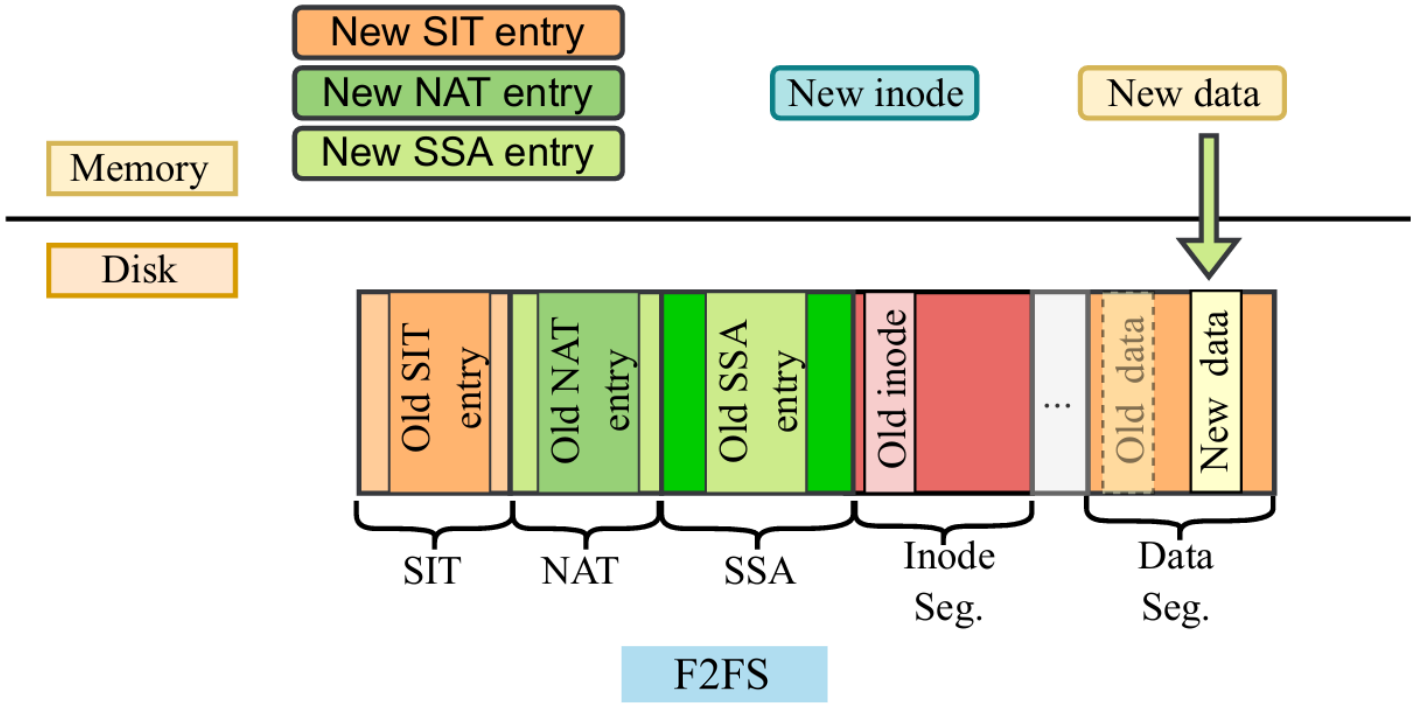
Journaling in F2FS (Out-of-place-update FS)

- Ordered journal mode
  - Need to journal both filesystem metadata and file metadata.



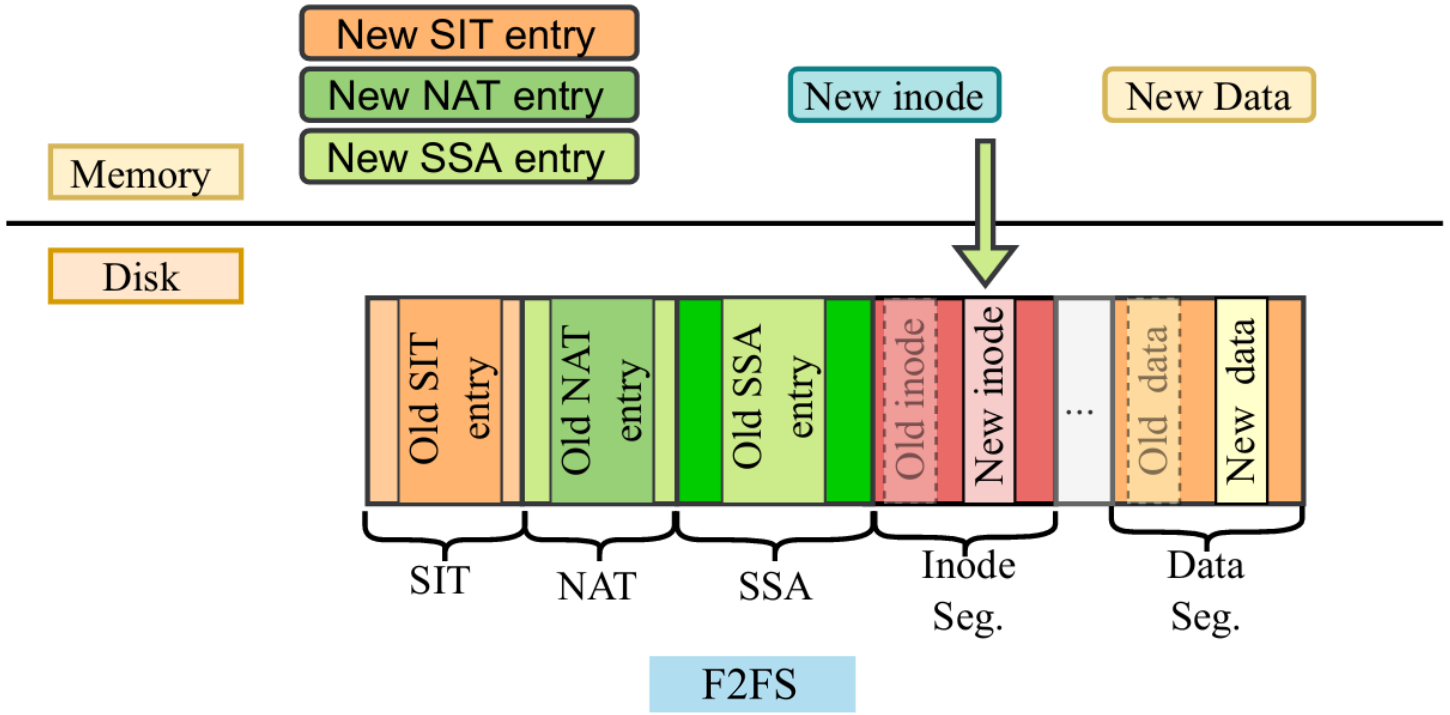
Journaling in F2FS (Out-of-place-update FS)

- Ordered journal mode
  - Need to journal both filesystem metadata and file metadata.



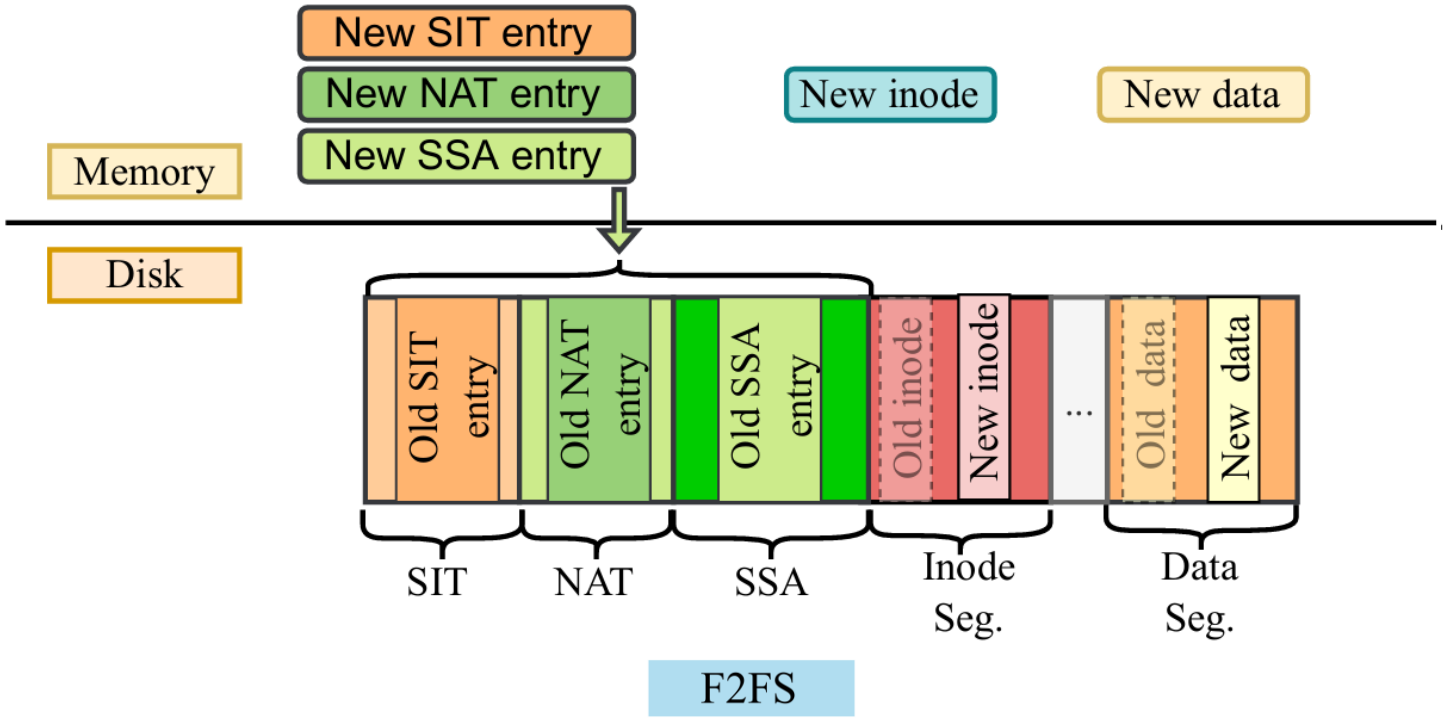
Journaling in F2FS (Out-of-place-update FS)

- Ordered journal mode
  - Need to journal both filesystem metadata and file metadata.



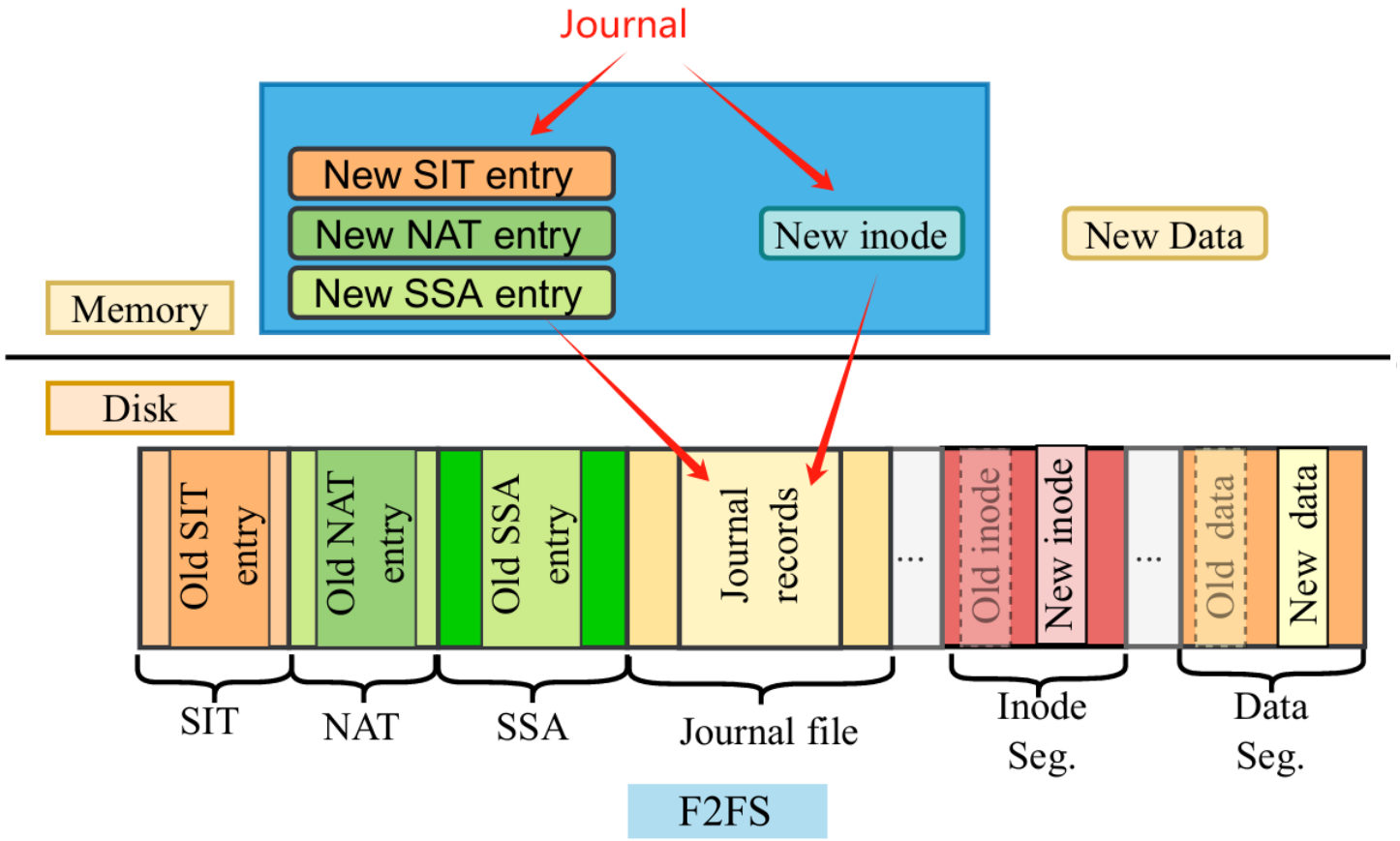
Journaling in F2FS (Out-of-place-update FS)

- Ordered journal mode
  - Need to journal both filesystem metadata and file metadata.



Journaling in F2FS (Out-of-place-update FS)

- Ordered journal mode
  - Need to journal both filesystem metadata and file metadata.

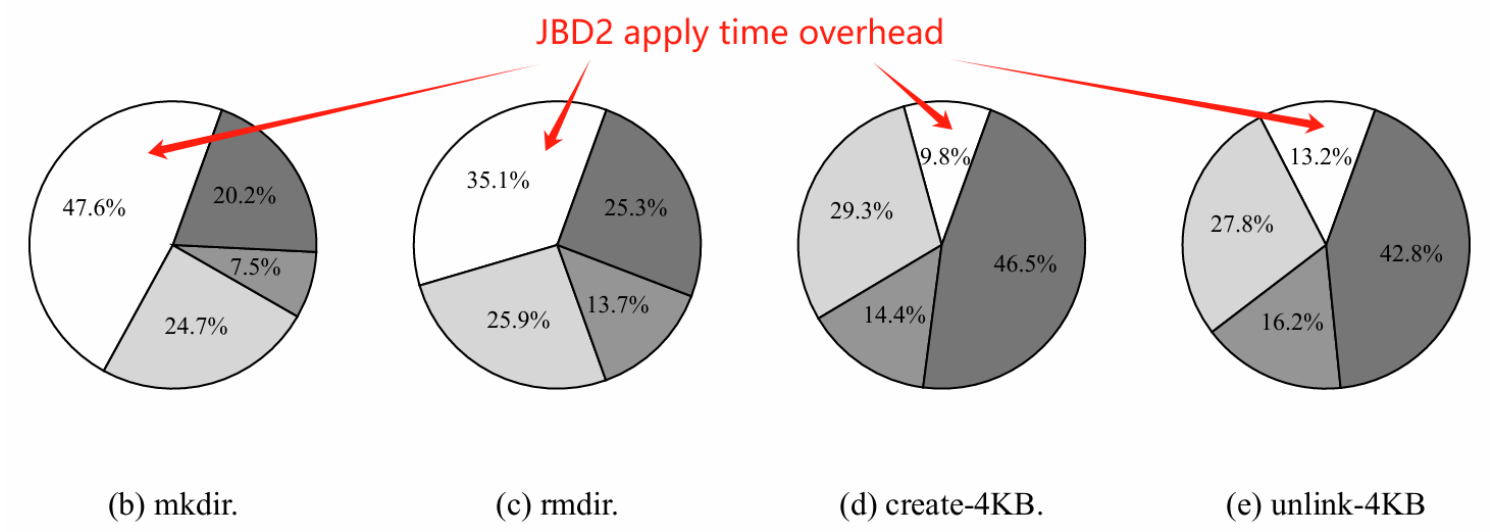


Outline

- Background
- Motivation
- **Design**
- Evaluation

Issue 1: Big Overhead in Page-based Journaling

- Storage and journal apply time overhead
  - Not only more space is occupied in journal files, but also journal apply will be triggered earlier.
  - E.g., JBD2 apply occupies a big portion of the total journaling time, up to 47.6%.

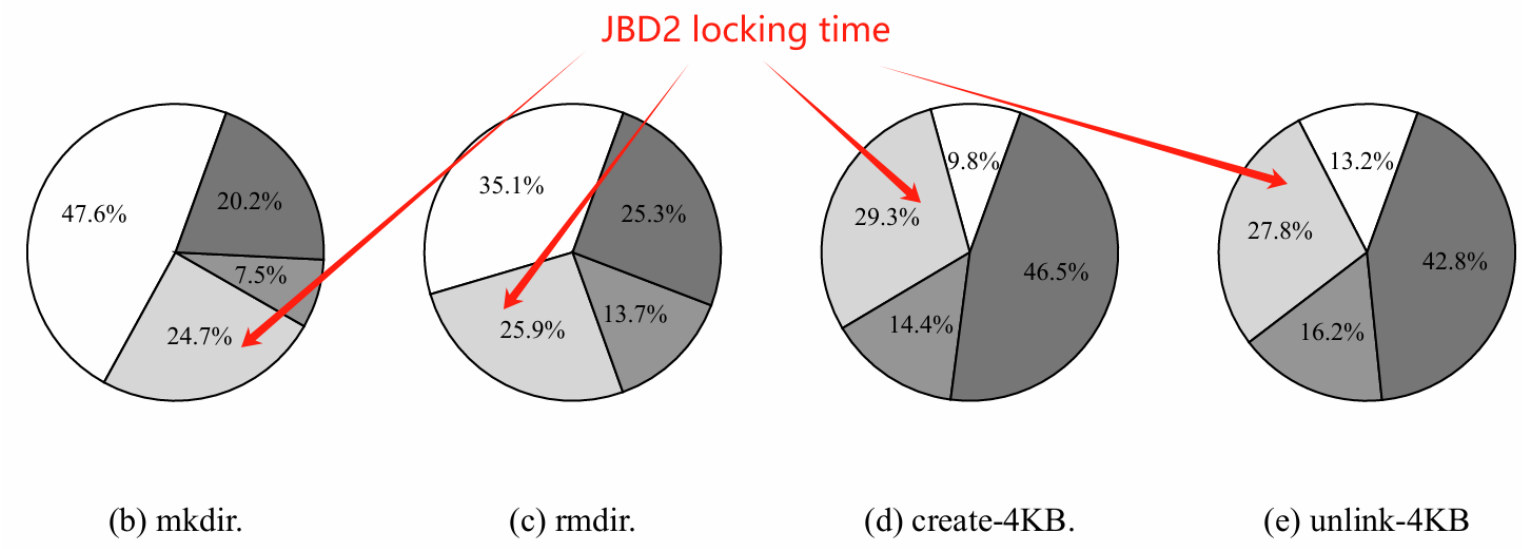


Solution

- Only journal metadata changes
  - We embed journal logging into per-inode log lists.
    - File metadata changes from the inode .
    - Filesystem metadata changes from SIT/NAT/SSA.
  - Require less log space.
  - Journal apply is not triggered frequently.

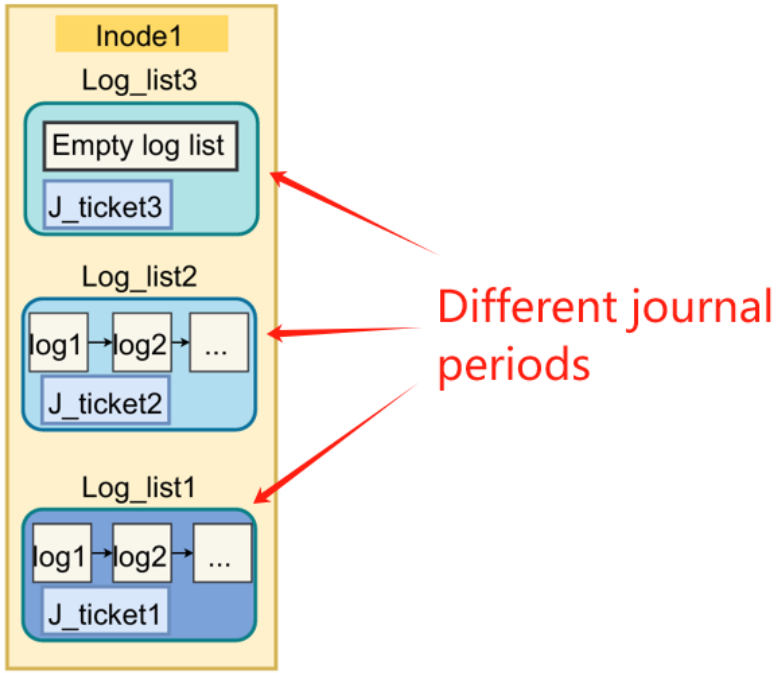
Issue 2: Intensive Locking

- Intensive locking contention in JBD2.
  - Lock on a journal ticket.
  - Lock on a log list.
  - Locking time occupies 24.7% - 29.3% of the total journaling time.



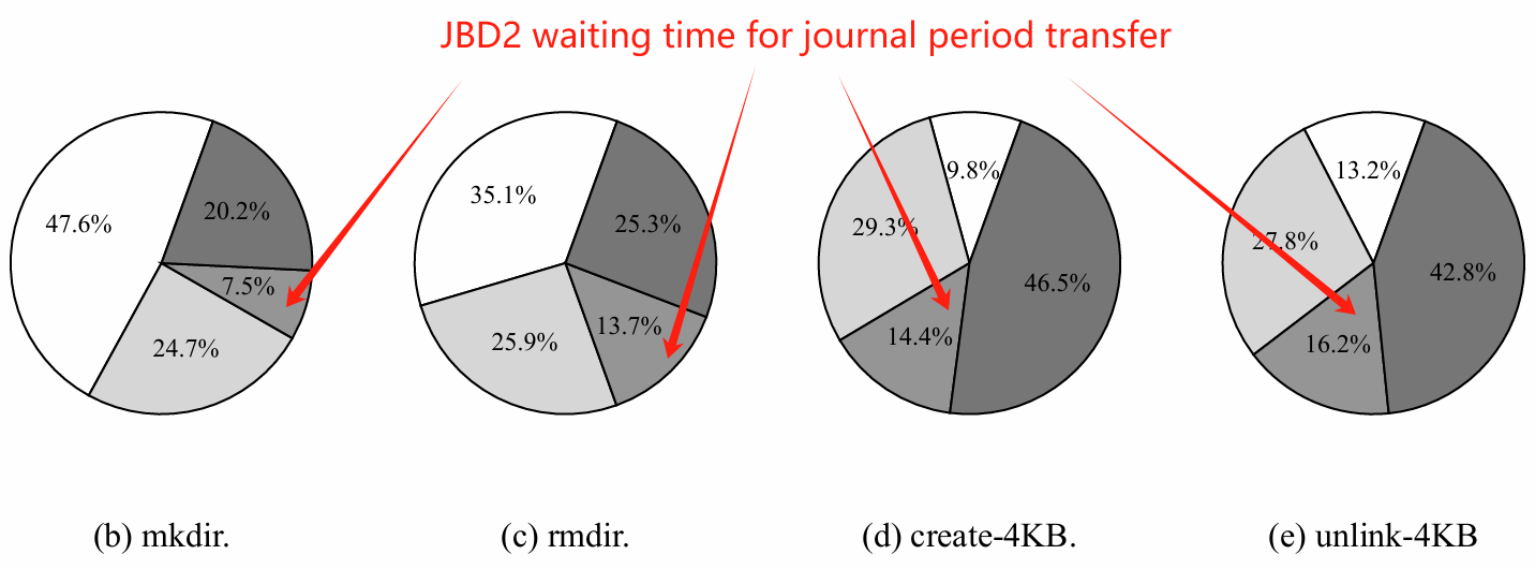
Solution

- Decentralized per-inode log lists
  - For each inode, a per-inode log list is devised to record all of its related logging.
  - Significantly reduces lock contention and interference among inodes when recording metadata changes.



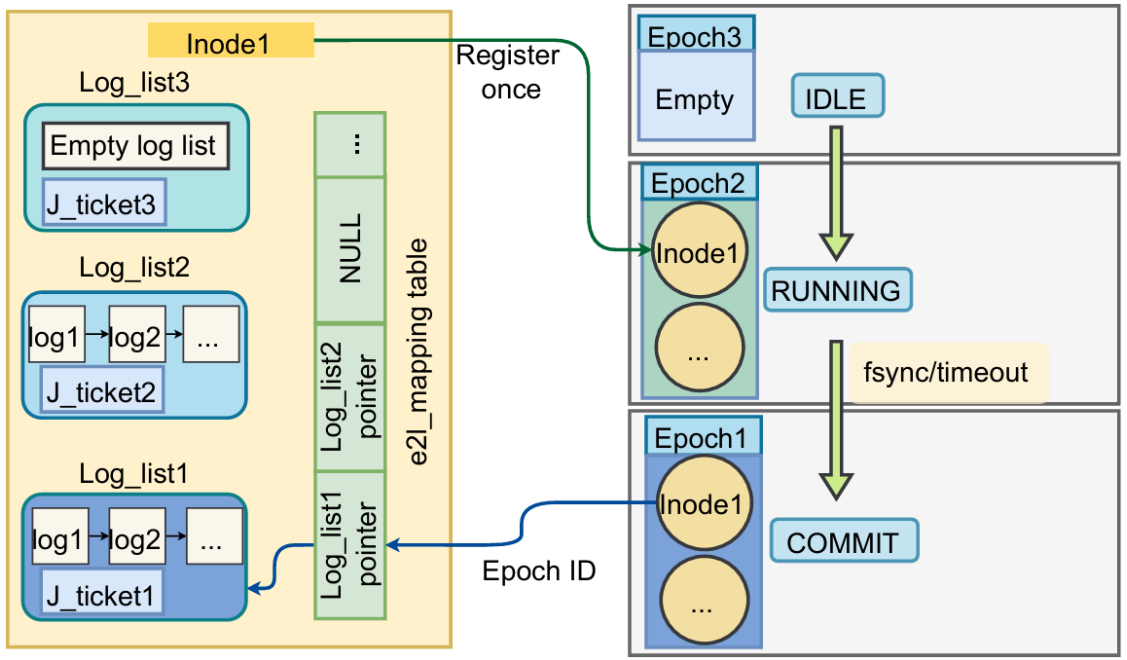
Issue 3: Long Waiting Times During Journal Period Transfer

- Long waiting times during journal period transfers in JBD2.
  - A new running transaction cannot be issued when the previous one starts to commit.
  - E.g., the waiting time for the journal period transfers occupies about 7.5% - 16.2% of the total journaling time.



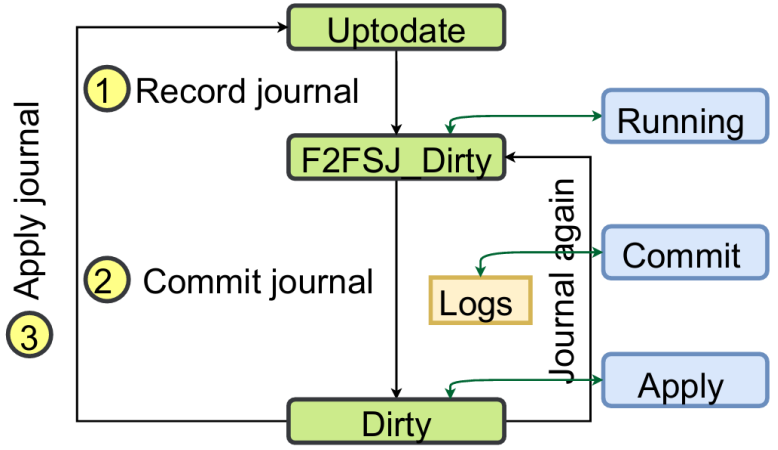
Solution

- Decouple data/control plane with epoch
  - Each journal period is associated with a new epoch.
  - Record metadata changes of an inode into its per-inode list (data plane).
  - Only register the inode information into the epoch (control plane).



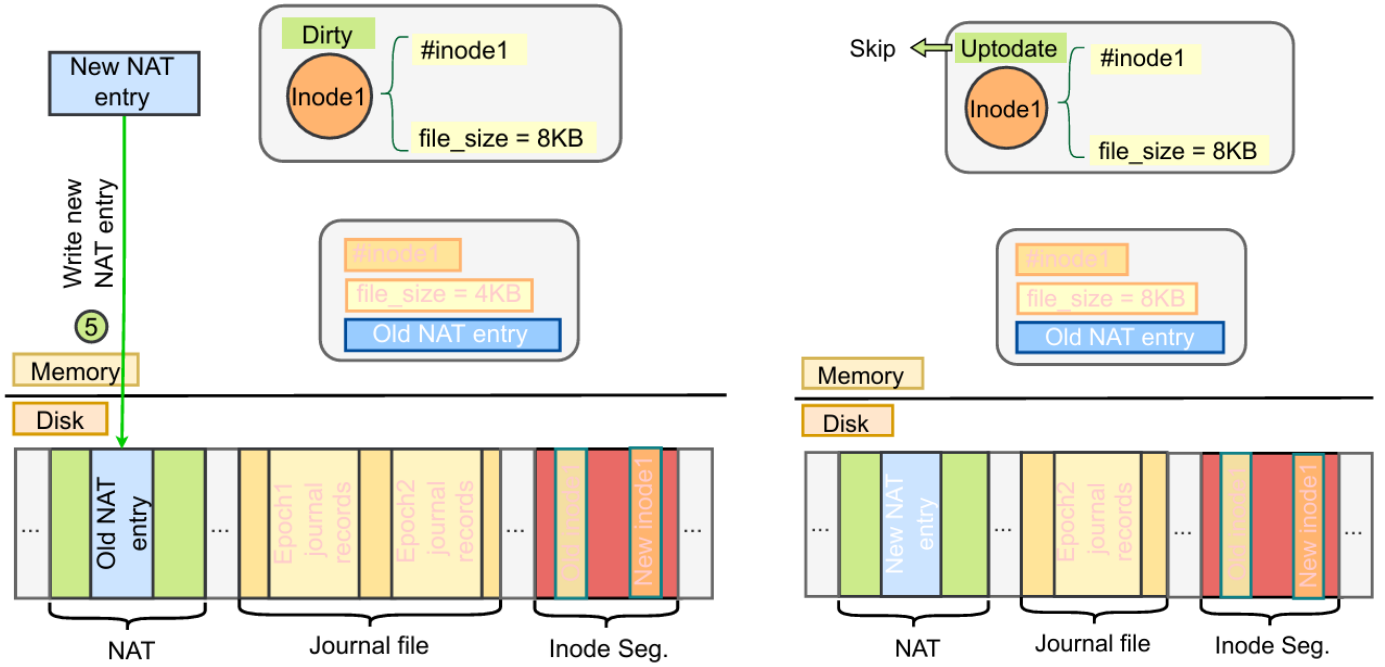
Journal Apply

- Journal Process with Page State Changes.
  - *Uptodate*
    - This page has the same content as its on-disk counterpart.
  - *F2FSJ_Dirty*
    - This page has been modified.
    - The changes have been recorded but not committed.
    - The data has not been flushed to disk.
  - *Dirty*
    - This page has been modified.
    - The changes have been committed but not applied.
    - The data has been flushed.
    - This page can be used for journal apply.



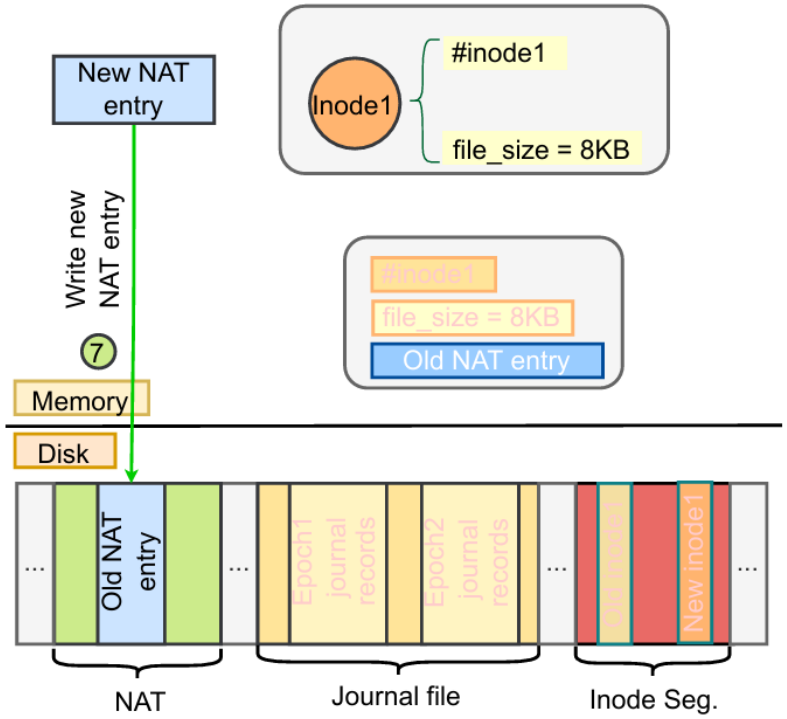
Fast-forward-to-latest Journal Apply

- If the page state is *Dirty.*
  - ➢ Directly flush the Dirty page and subsequently altering its state to *Uptodate*.
- If the page state is *Uptodate*.
  - ➢ Skip this record.
- If the page state is *F2FSJ_Dirty.*
  - ➢ Apply on-disk journal record.



Crash Recovery

- Crash recovery will be performed when there are journal records that are not applied.
  - Read journal records from journal file and apply them one by one.
    - Locate and read the old inode to memory, apply metadata changes with logs.
    - Update the new inode and its SIT/NAT/SSA entries to disk.



Outline

- Background
- Motivation
- Design
- **Evaluation**

Experiment Setup

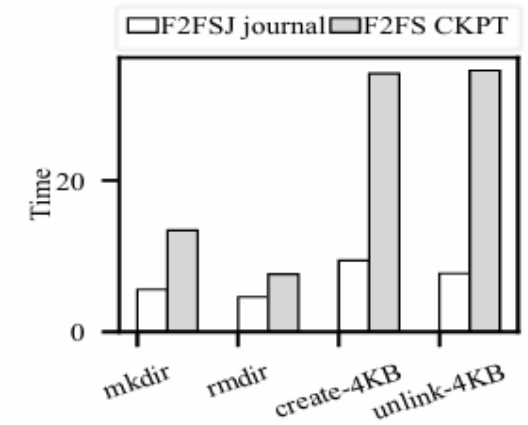
- Implementation
  - Define various log types.
  - Modify *struct f2fs_inode_info* to implement log lists and e2l_mapping tables.
  - Implement two kernel threads for journal commit and apply.
  - 256MB log area.
- Environment.
  - Intel's i9-10850K (19 CPU cores), 64GB DRAM.
  - 256GB Samsung 980pro PCIe4.0 NVMe SSD.
  - Linux kernel v5.15.39.

Benchmarks

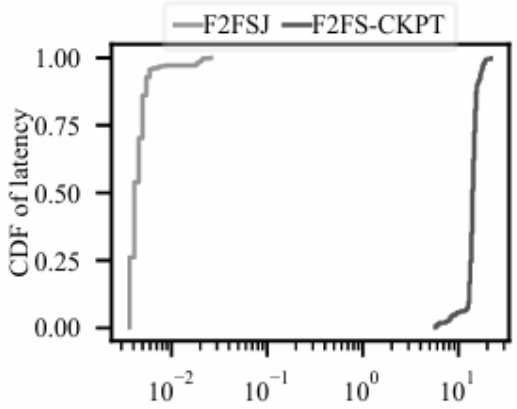
Categories	Name	Description
Metadata-intensive workloads with Filebench [9]	mkdir	Create 4M directories.
	rmdir	Delete 4M directories.
	create-empty	Create 4M empty files.
	unlink-empty	Delete 4M empty files.
Small-file-intensive workloads with Filebench	create-4KB	Create 4M 4KB-files.
	unlink-4KB	Delete 4M 4KB-files.
	copy-4KB	Create and copy 2M 4KB-files.
Data-intensive workloads for a 32GB-file with FIO [11]	Seq_write	Sequentially write.
	Rand_write	Randomly Write.
	Seq_read	Sequentially read.
	Rand_read	Randomly read.
	SeqWR_W	Write bandwidth of mixed Sequential writes/reads.
	SeqWR_R	Read bandwidth of mixed Sequential writes/reads.
	RanWR_W	Write bandwidth of mixed random write/reads.
	RanWR_R	Read bandwidth of mixed random write/reads.
Realistic applications with Filebench and APP commands	Varmail	Frequently create and delete small files with fsync.
	OLTP	Intensively write and append data to files.
	Fileserver	Intensively write and read small files.
	Webserver	Intensively read small files with fewer writes.
	Webproxy	Intensively read small files with fewer writes and deletions.
	git clone	Download a remote 4.7GB code repository.
	cp -r	Copy a local 3.9GB linux source code.
	make Linux	Compile Linux with local source code.
Android workloads with Mobibench [21]	Twitter	Replay I/O trace of Twitter.
	Facebook	Replay I/O trace of Facebook.
	SQlite	SQlite performance with varying journal mode.

Evaluation Results

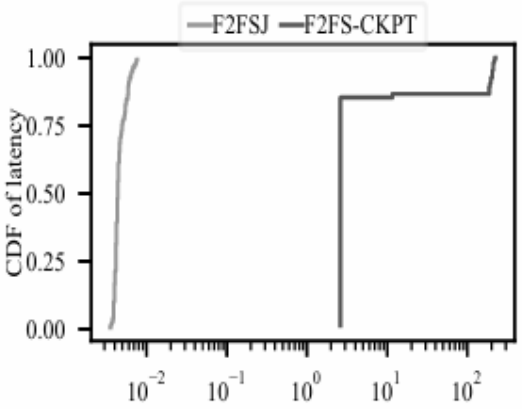
- F2FSJ vs F2FS checkpointing
  - Significantly reduce checkpointing time by 1.7x – 4.9x.
  - Eliminate checkpointing long tail latency.
  - Reduce average latency up to 35%.



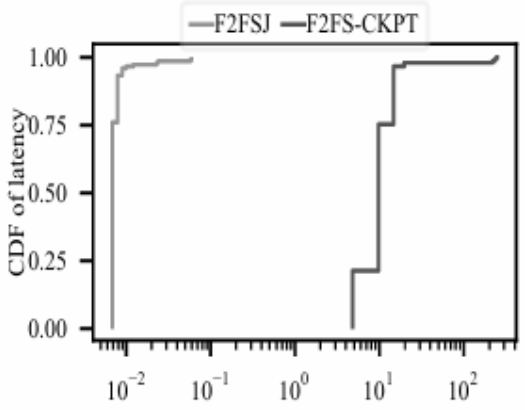
(a) Journal/checkpoint time (s).



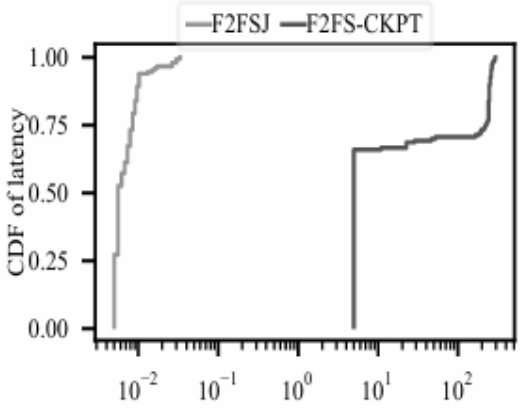
(b) CDF latency (ms) for mkdir.



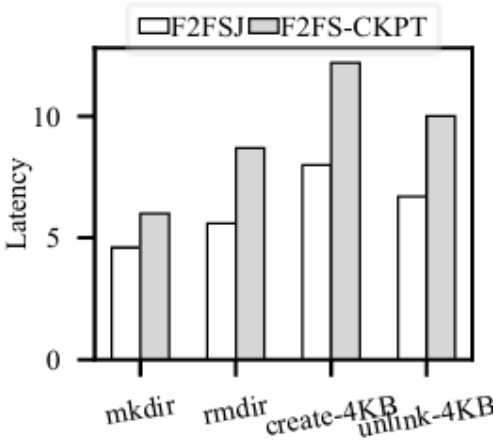
(c) CDF latency (ms) for rmdir.



(d) CDF latency (ms) for create-4KB.



(e) CDF latency (ms) for unlink-4KB.



(f) Average latency (us).

Conclusion

- F2FSJ eliminates the long tail latency of checkpoint and provides fine grained crash recovery.
  - It only journals the metadata changes to reduce I/O and storage overheads.
  - It employs a decentralized journal design featuring per-inode log lists to minimize lock contention and interference.
  - It decouples data/control planes to eradicate waiting times during journal period transfer.
  - It uses fast-forward-to-latest journal apply to consolidate multiple small writes.
- The source code and detailed evaluation have been released for public access.
  - https://github.com/10033908/F2FS-J

Thank you!