Multi-version Checkpointing for Flash File Systems

Shih-Chun Chou

Outline

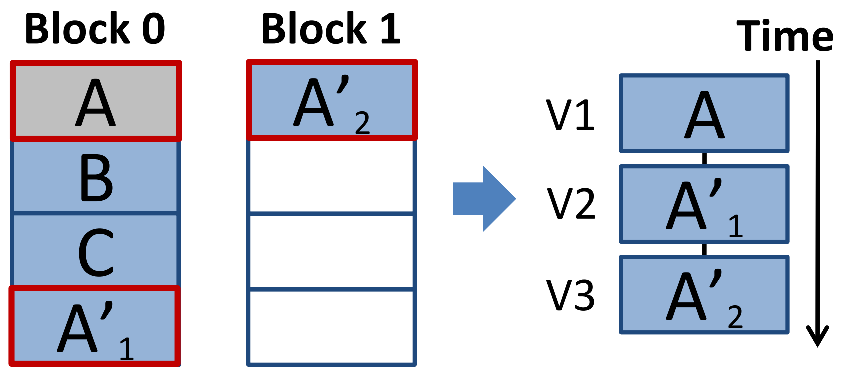
- Introduction
- System Architecture
- Multi-version Checkpointing for Flash File Systems
- Analysis and Experimental Results
- Conclusion

Introduction(1/2)

- Flash memory is widely adopted in various storage systems.
  - e.g., multiple-level-cell (MLC)
- However, their high bit error rates and low endurance give rise to serious challenges on the reliability issue.
- Although stronger error correction codes can be applied to enhance their reliability, they are less capable of recovering flash page failures caused by the increasing burst-error rates and decreasing block endurance when a flash block has endured more and more erases.

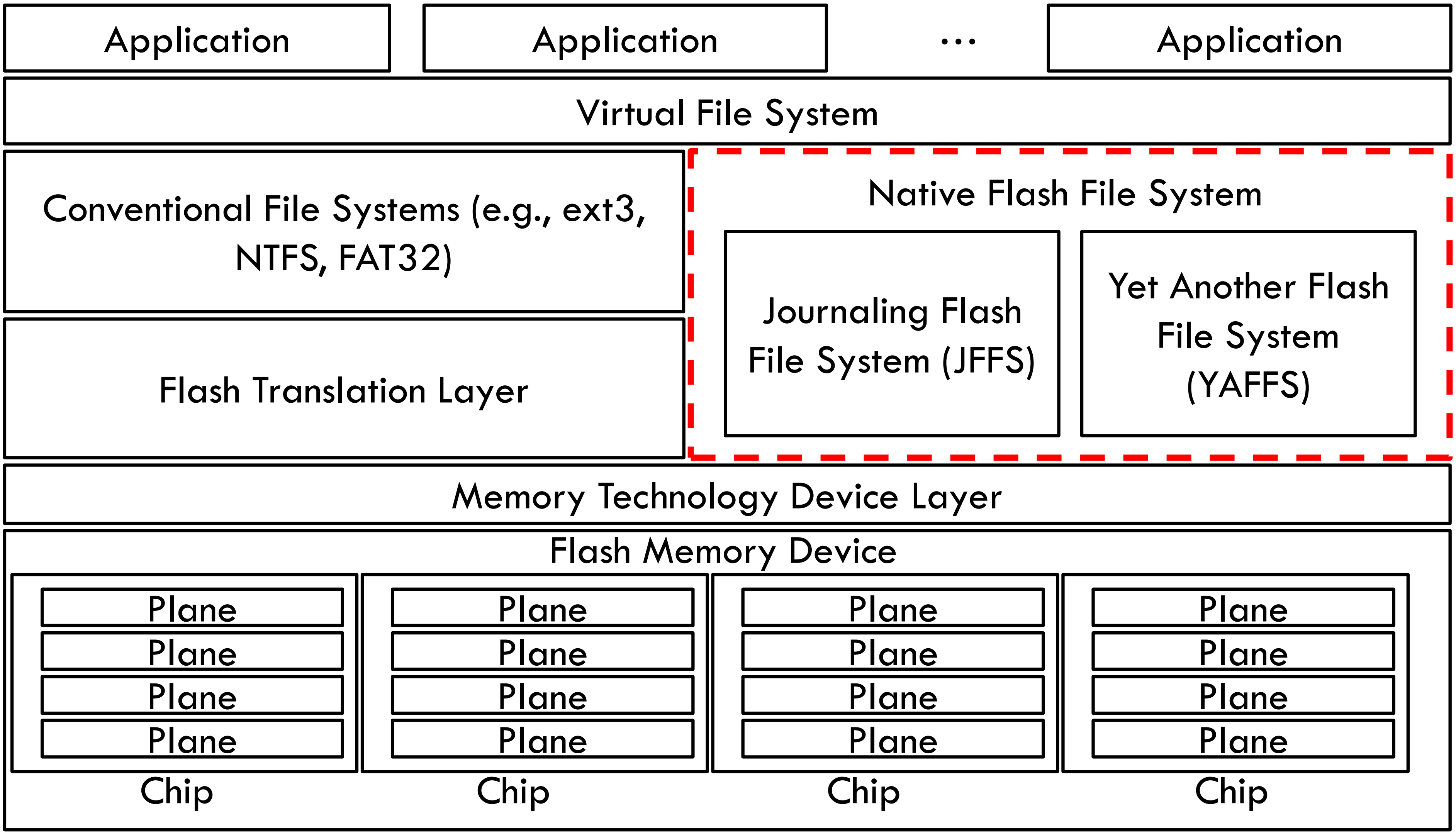
Introduction(2/2)

- Out-place updating
  - Innate overheads of flash memory
  - Multiple versions being kept in flash memory



- Such an observation motivates this work on how to covert the drawback of the coexistence of multi-version data into the advantage to enhance the reliability of flash memory.

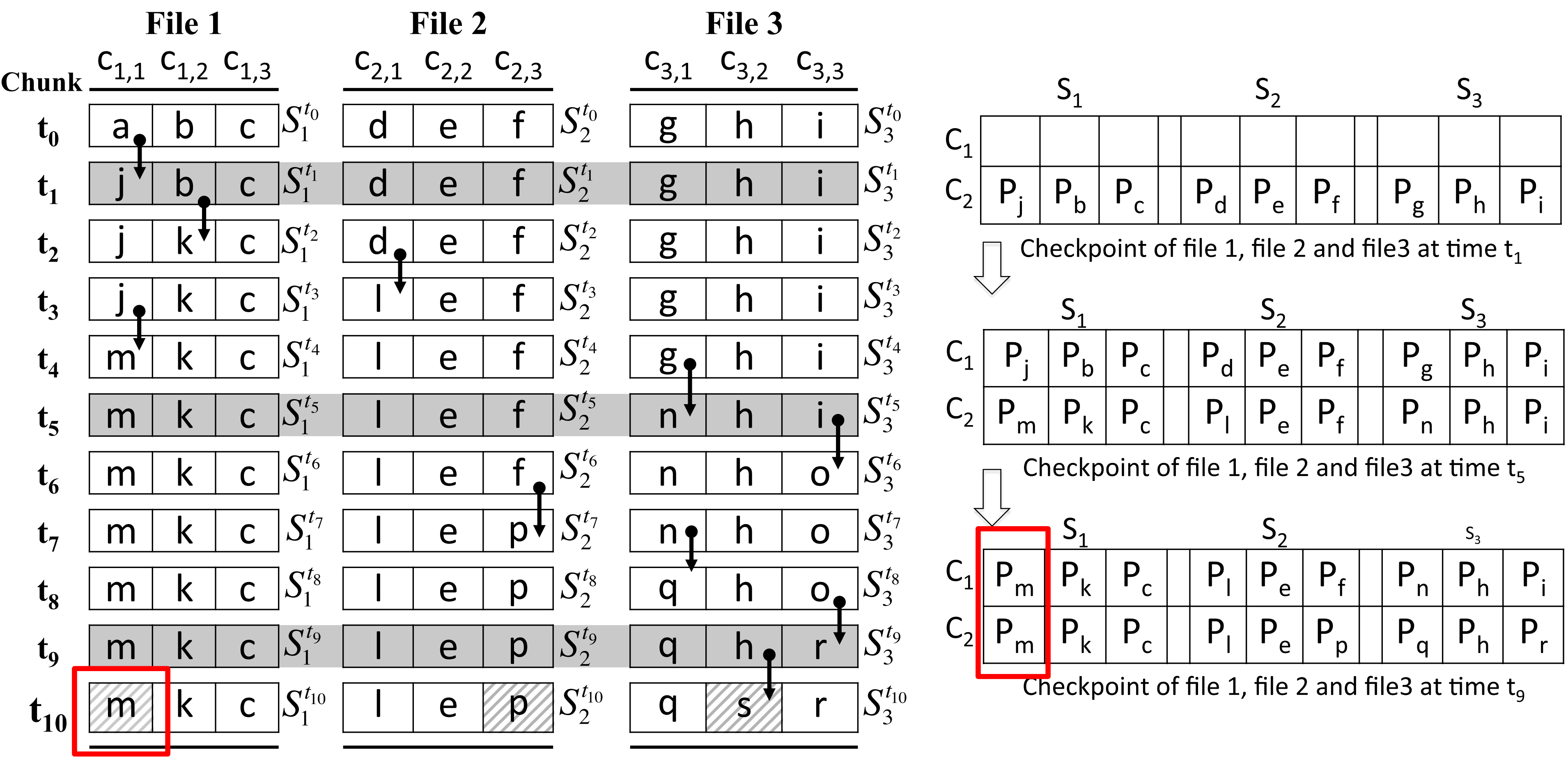
System Architecture



Multi-version Checkpointing for Flash File Systems

- A checkpoint-based strategy to guarantee the data integrity of the whole flash file system
  - minimal management and space overheads.
- Main ideas
  - To utilize the co-existence fact of multiple versions of the same data, due to out-place updates.
  - To maintain multiple checkpoints of the file system.
- The technical problem falls on
  - how to maintain the checkpoints of a flash file system with minimized space overheads.
  - how to roll a flash file system back to the most-recent consistent version with minimized rollback overheads.

Two-version checkpointing strategy

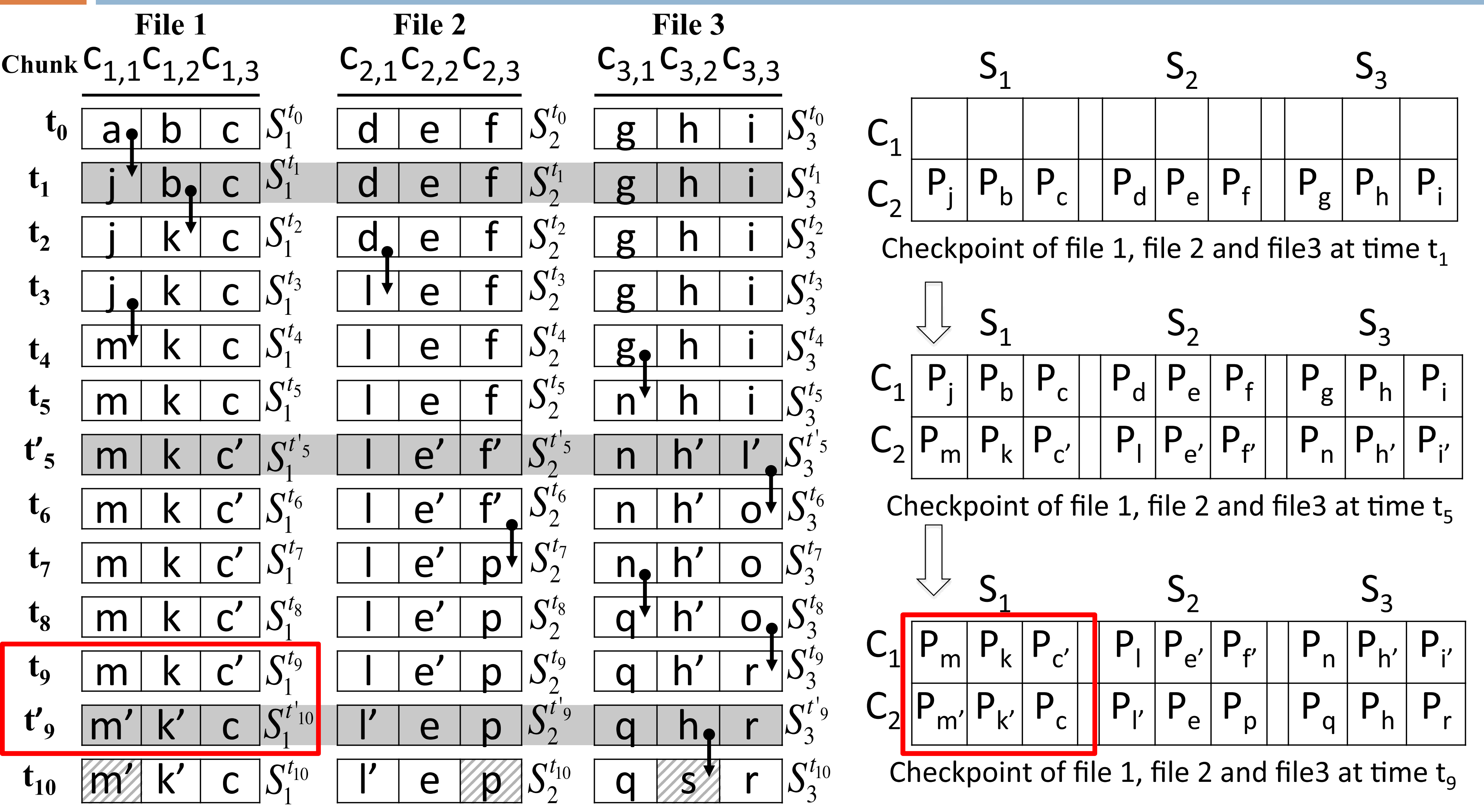


- An example with three 3-page (or 3-chunk) files in the flash file system to elaborate how the two-version checkpointing strategy works.

The two-version Control Mechanism

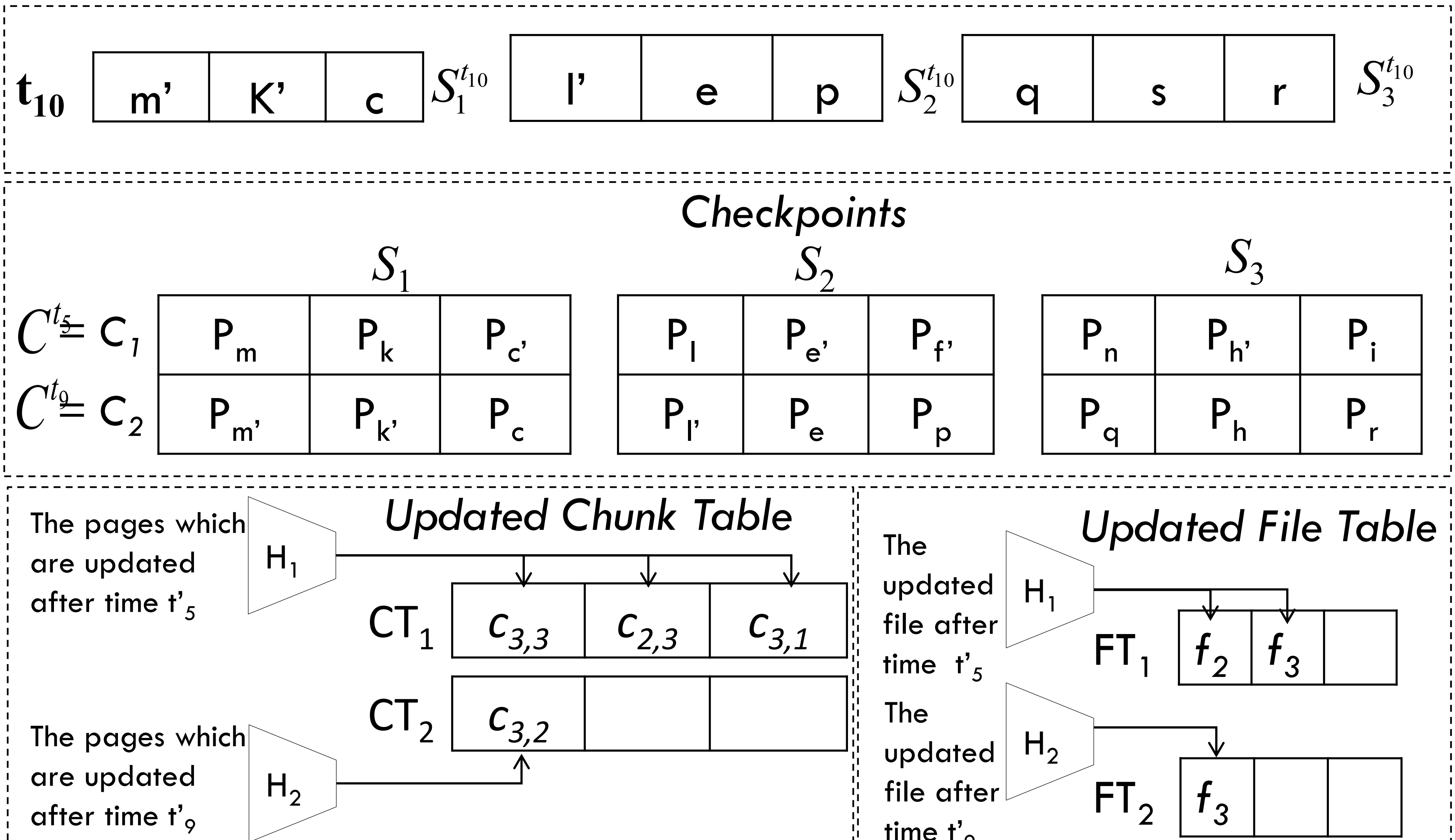
- Chunk Duplication
  - To avoid improper discarding of an early version when a new checkpoint is made.
  - To duplicate the data of a chunk to another page during the creating of a new checkpoint if this chunk has not been updated after the previous checkpoint was made.
- Chunk Tracking
  - To prevent unnecessary scans of chunks/files on recovering a file system back to a consistent version.
  - To know which chunks/files are updated after a checkpoint is made.

An example of chunk duplication



An example of chunk tracking

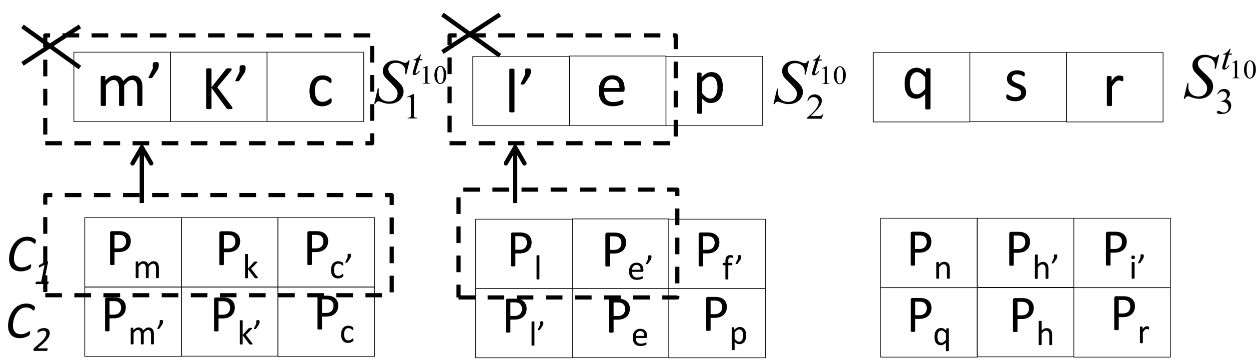
The snapshot of files at time t_{10}



Recovery: Case 1

If the crashed chunk $c_{i,j}$ is not in CT_1 and CT_2:

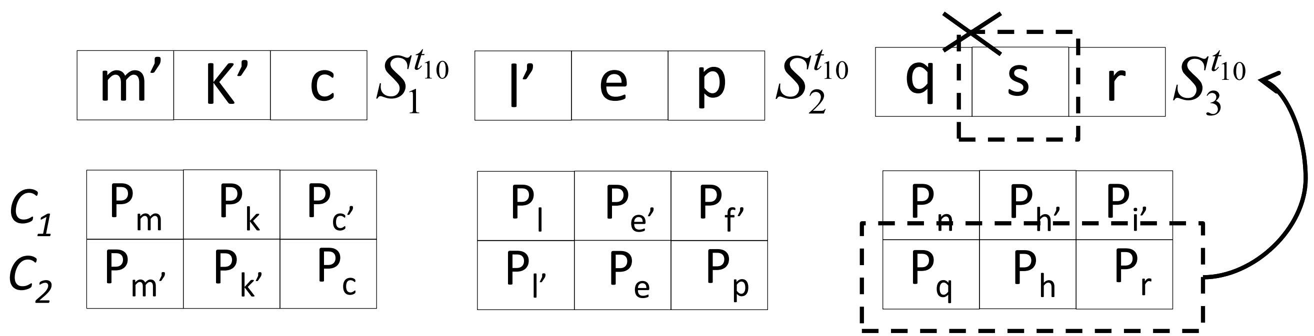
1.1 Rollback $c_{i,j}$ to $c_{i,j,1}$ in C_1 .



Recovery: Case 2

If the crashed chunk $c_{i,j}$ is in CT_2.

2.1 Roll back all files f_i in FT_2 to C_2 .

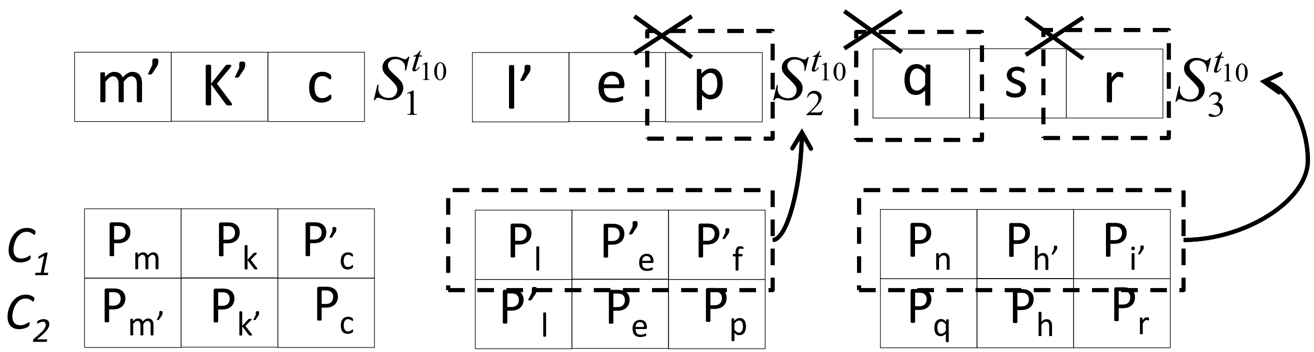


Recovery: Case 3

If the crashed chunk $c_{i,j}$ is in CT_1.

3.1 Roll back all files f_j in FT_2 to C_1 .

3.2 Roll back all files f_j in FT_1 to C_1 .



The Experimental Setup(1/2)

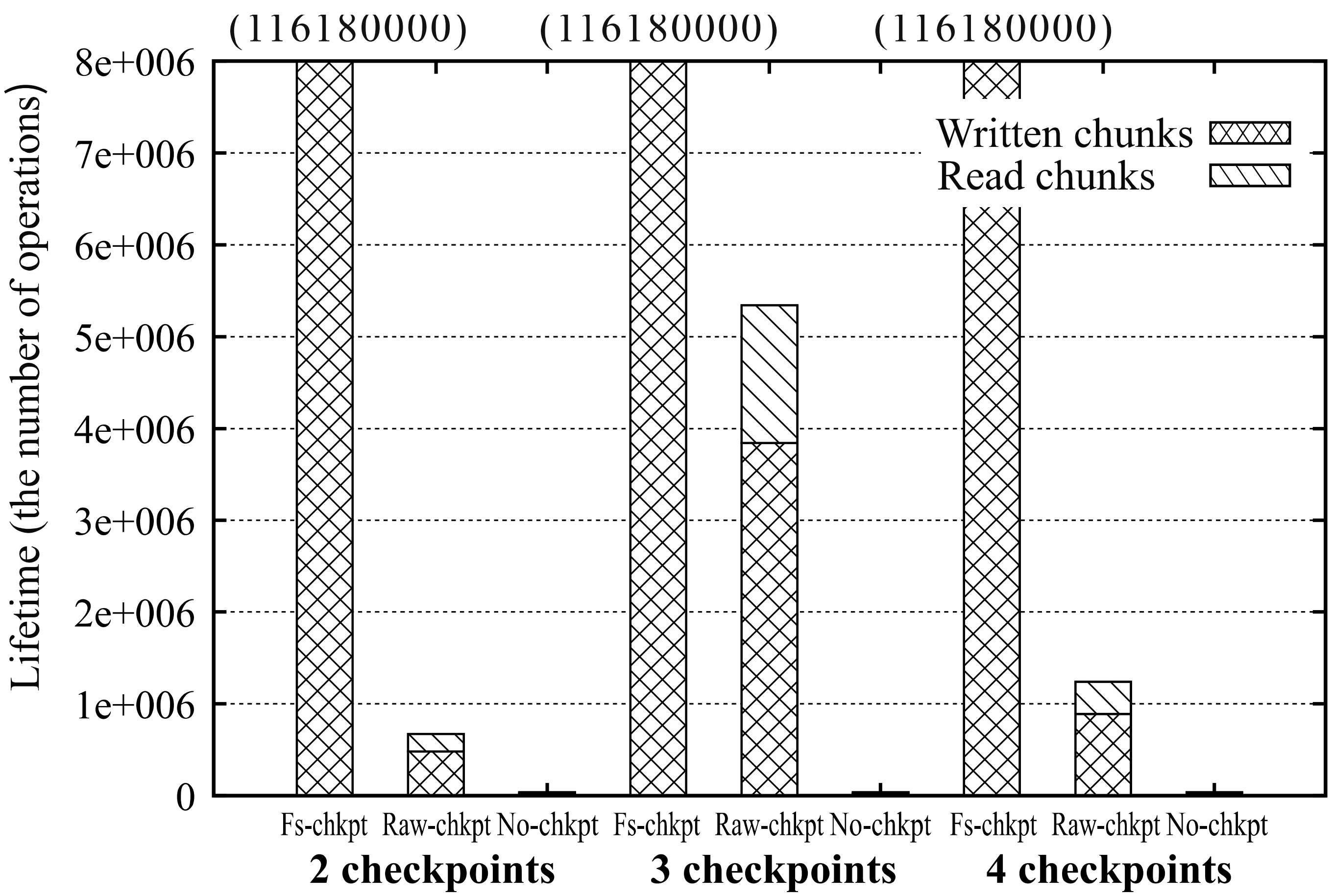
Property	Value
Chip size / block size / page size	512 MB / 128 KB / 2 KB
Erase time / write/program time Page read time / serial access time	3 ms / 900 μs. 50 μs / 25 ns./byte.
Endurance (P/E cycles)	5,000

The Experimental Setup(2/2)

- The flash memory device is made unreliable by setting a data error rate of 10^{-4} as each chunk is being accessed.
- The Bonnie++ benchmark is repetitively run for 5,000 iterations for performance testing.
- 1,024 files (each of 8 MB size) are created with 2KB chunk size.
- The Postmark benchmark randomly generates 1,000 small files (whose sizes range from 500 bytes to 9.77 KB) and 500,000 random file I/O transactions (each addressing a 512-byte chunk) for the experiments.

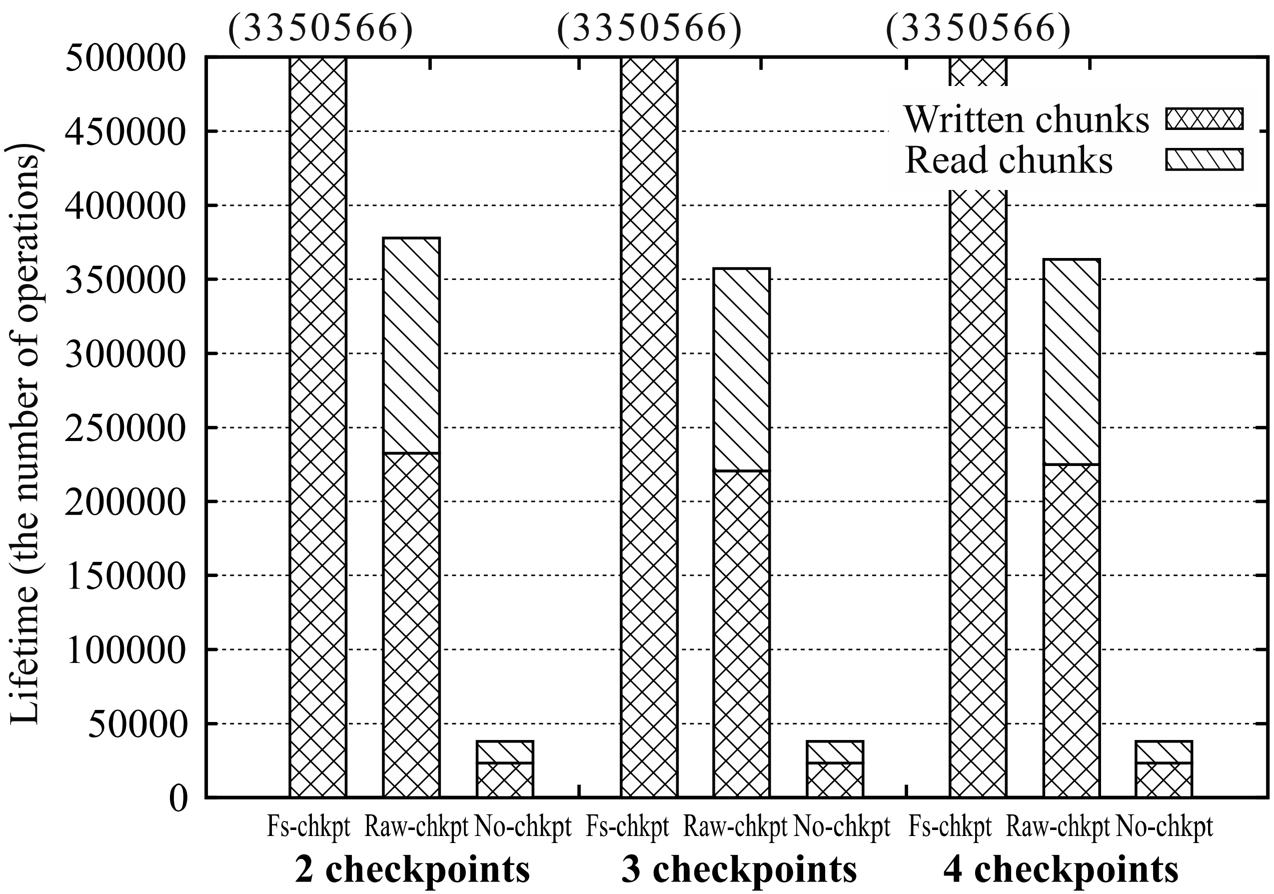
Lifetime with Bonnie++ benchmark

- Checkpoint interval of 10,000.



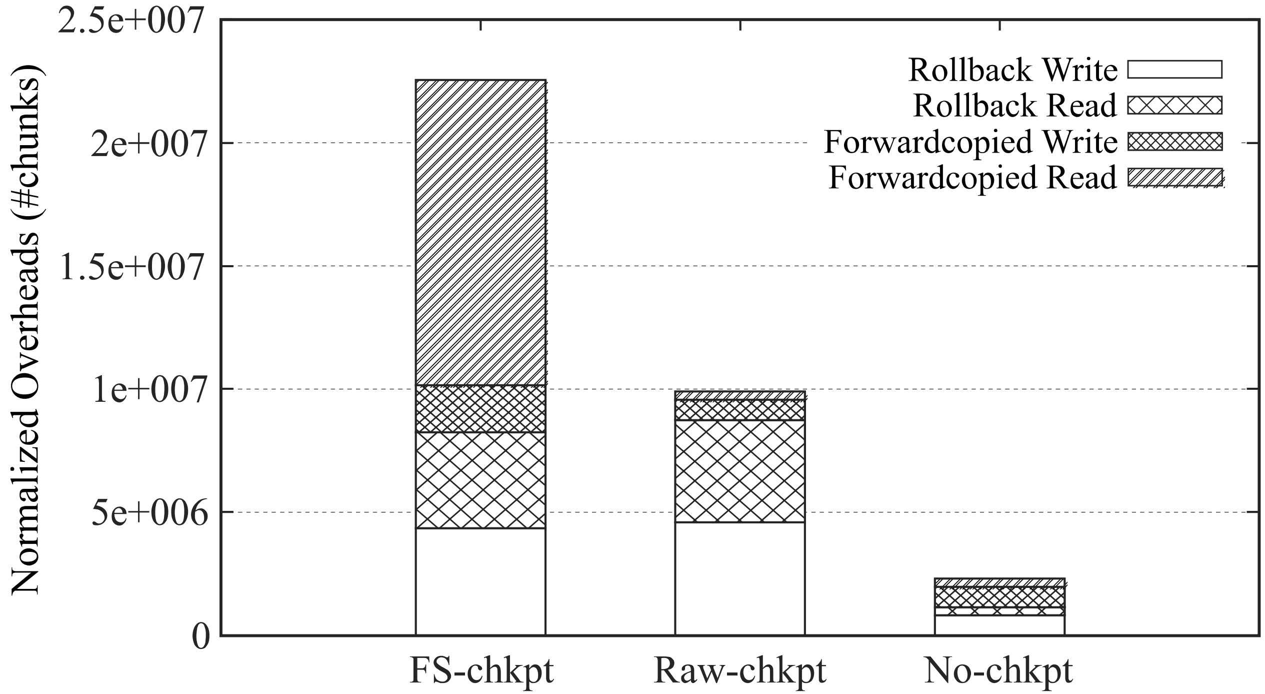
Lifetime with Postmark benchmark

- Checkpoint interval of 10,000.



The overhead

- The extra performance overheads is limited within 2.2X against Rawchkpt
- To get at least 172.00 times of relative lifetime improvement against Raw-chkpt.



Conclusion (1/2)

- This paper proposes a multiversion checkpointing strategy to guarantee the integrity and consistency of flash file systems when some unrecoverable flash pages occur.
- A control mechanism with the support of chunk duplication and chunk tracking is designed
  - to avoid improper discarding of an early version on making a new checkpoint
  - to prevent unnecessary scans/rollbacks of chunks/files on file system recovery

Conclusion (2/2)

- A recovery mechanism is then presented to restore a corrupted file system back to a consistent version after the corruption of flash pages
- In the future
  - how to extend the proposed strategy to FTL designs.
  - the performance of applying different garbage collection policies.

Q & A

Thank you for your listening