# Journaling and Log-Structured File Systems

(Chapters 42, 43)

CS 4410

Operating Systems


Cornell CIS

COMPUTING AND INFORMATION SCIENCE

[R. Agarwal, L. Alvisi, A. Bracy, M. George, F.B. Schneider, E. Sirer, R. Van Renesse]

# Fault-tolerant Disk Update

Problem: many file system operations require multiple disk updates. What if there is a crash half-way?

## Use Journaling (aka) Write-Ahead Logging

- **Idea**: Protocol where performing a **single** disk write causes multiple disk writes to take effect.

- **Implementation**: New on-disk data structure ("journal") with a sequence of blocks containing updates *plus* …

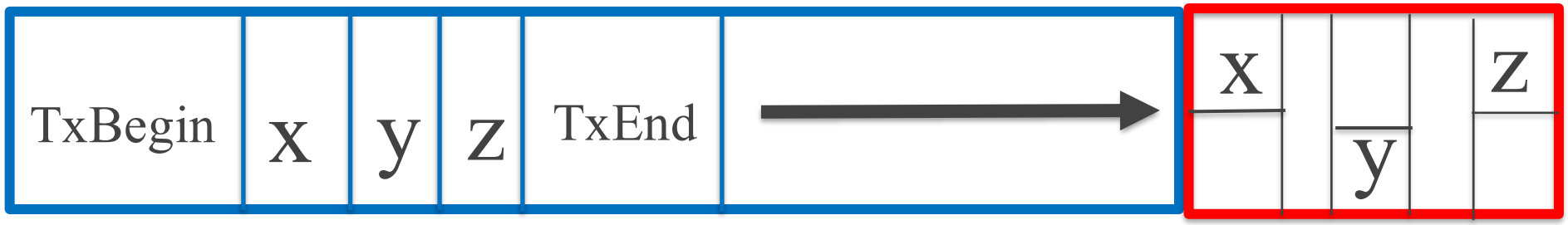
# Journal-Update Protocol Step

write x; write y; write z

*implemented by*

- Append to journal: TxBegin, x, y, z
- Wait for completion of disk writes.
- Append to journal: TxEnd
- Wait for completion of disk write.
- Write x, y, z to final locations in file system

*called* **checkpoint step**



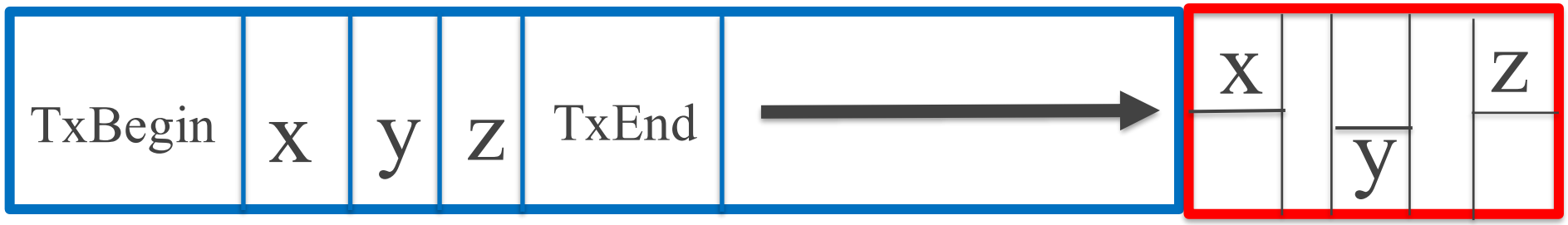
# Journal-Update Protocol Step

write x; write y; write z

*implemented by*

- Append to journal: TxBegin, x, y, z
- Wait for completion of disk writes.
- Append to journal: TxEnd
- Wait for completion of disk write.
- Write x, y, z to final locations in file system

*why??*



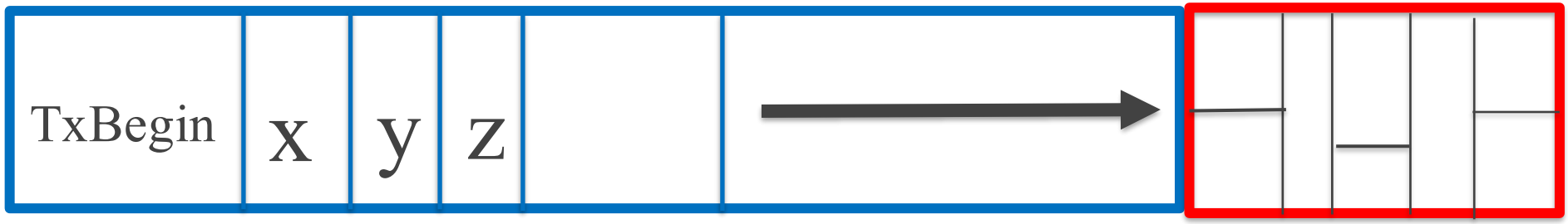
# What if Crash?

write x; write y; write z

*implemented by*

- Append to journal:  TxBegin, x, y, z
- Wait for completion of disk writes.
- Append to journal: TxEnd
- Wait for completion of disk write.
- Write x, y, z  to final locations in file system.

crash!




**Recovery protocol for TxBegin … TxEnd**:

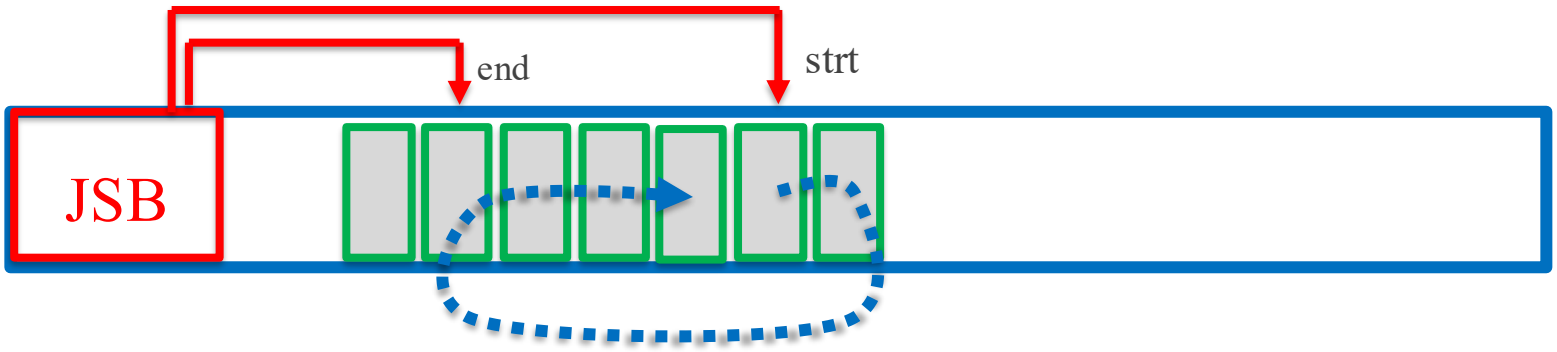
- **if** TxEnd present **then** redo writes to final locations following TxBegin
- **else** ignore journal entries following TxBegin

# Full-Journal Recovery Protocol

- Replay journal from start, writing blocks as indicated by checkpoint steps.

<u>Infinite Journal → Finite Journal:</u>

- introduce **journal super block** (JSB) as first entry in journal: JSB gives start / end entries of journal.
- view journal as a circular log
- delete journal entry once writes in checkpoint step complete.



# Performance Optimizations

- Eliminate disk write of TxEnd record.
  - Compute checksum of xxx in “TxBegin xxx TxEnt”
  - Include checksum TxBegin
  - Recovery checks whether all log entries present

- Eliminate disk write when data block
  - Use journal only for metadata
  - Guarantees integrity of file system data structure but not data integrity

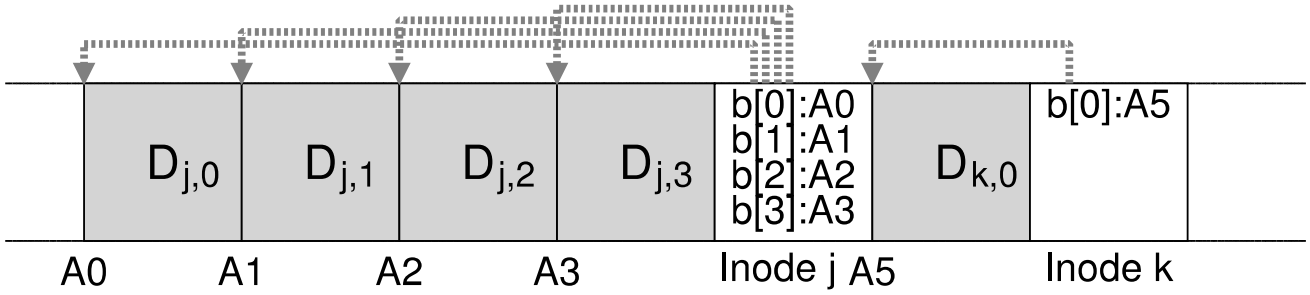
# Log-Structured File Systems

Technological drivers:

- System memories are getting larger
  - Larger disk cache
  - Reads mostly serviced by cache
  - Traffic to disk mostly writes
- Sequential disk access performs better
  - Avoid seeks for even better performance
  - Better wear leveling on SSDs

**Idea**: Buffer writes and store as single log entry on disk. Disk becomes one long log!

# Storing Data on Disk



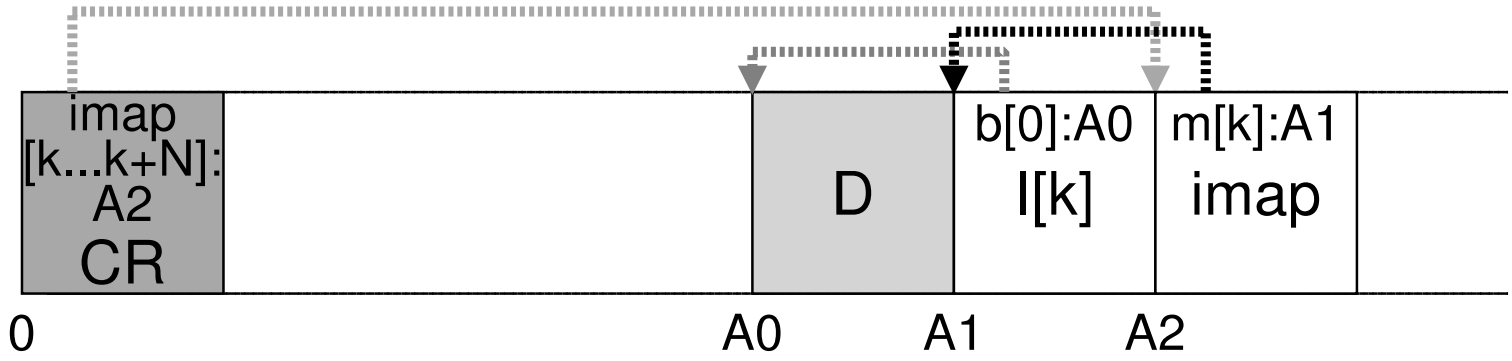

- Updates to file j and k are buffered into a memory *segment*
- Inode for a file points to log entry for data
- An entire segment is written at once.

# How to Find Inode on Disk

In UFS: inode nbr → location on disk
In LFS: location of inode on disk changes…

LFS: Maintain **inode Map** in pieces and store updated piece on disk.

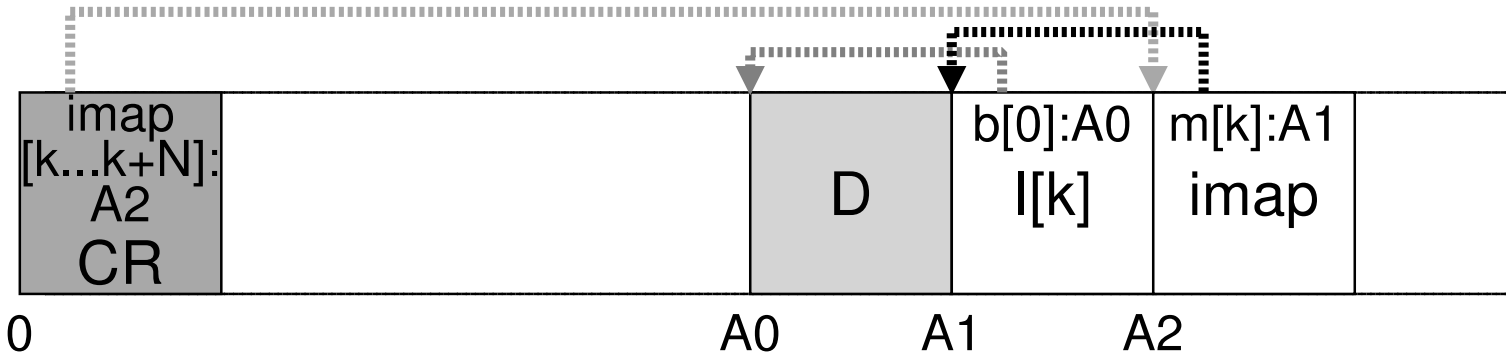
- For write performance: Put piece(s) at end of segment
- **Checkpoint Region**: Points to <u>all</u> inode map pieces and is updated every 30 secs. Located at fixed disk address. Also buffered in memory



# To Read a File in LFS

- [Load checkpoint region CR into memory]
- [Copy inode map into memory]
- [Read appropriate inode from disk]
- [Read appropriate file (dir or data) block]

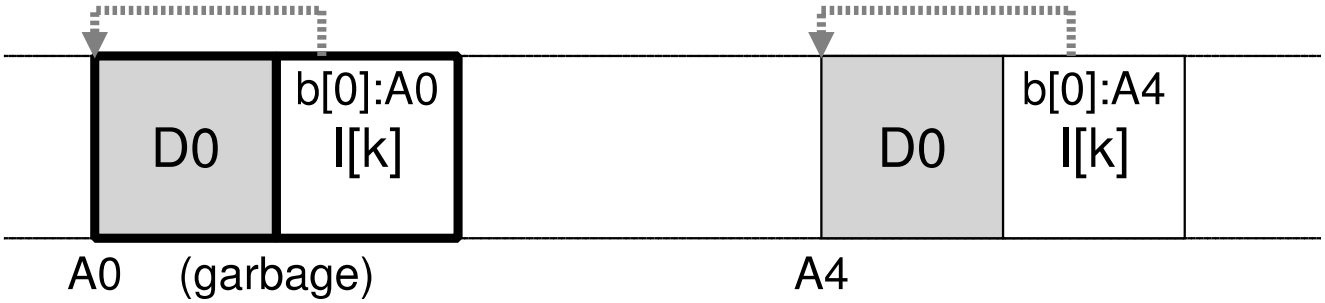
[…] = step not needed if information already cached.



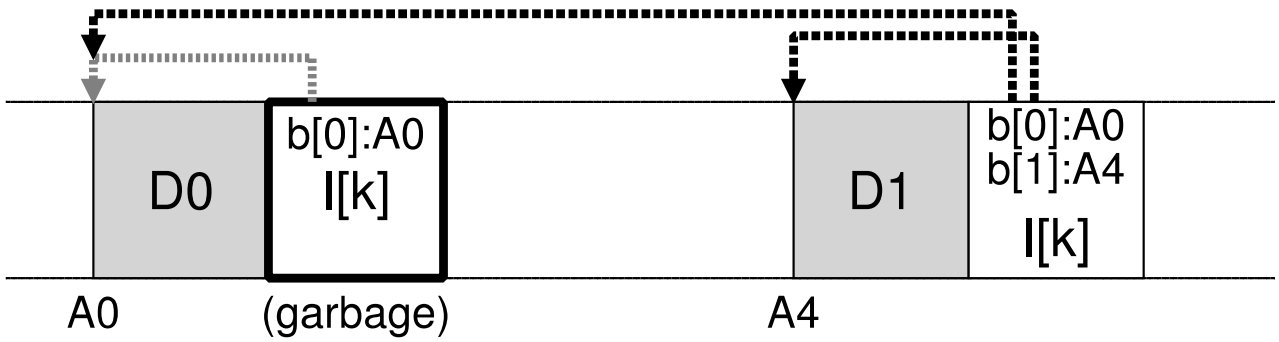
# Garbage Collection

Eventually disk will fill. But many blocks ("garbage") not reachable via CR, because they were overwritten.

Ex. 1




Ex. 2



# LFS Cleaner

Protocol:

1. read segment;
2. identify garbage blocks within;
3. copy non-garbage blocks to new segment;
4. append new segment to disk log

Each segment includes **segment summary block** that includes for each data block D in segment:

- inode number
- Offset in the file for that inode

Read the block for that <inode, offset> from LFS to reveal if D is live (=) or it is garbage (=!).

# Crash Recovery

## LFS writes to disk: CR and segment.

After a crash:

- Find most recent consistent CR (see below)
- Roll forward by reading next segment for updates.

Crash-resistant atomic CR update:

- Two copies of CR: at start and end of disk.
- Updates alternate between them.
- Each CR has timestamp ts(CR,start) at start and ts(CR,end) at end.
  - CR consistent if ts(CR,start)=ts(CR,end)
- Use consistent CR with largest timestamp