The Log Structured File System (LFS)

Williams College
CSCI 333

This Video

- Log Structured File System (LFS)
    - Motivation
    - Design Trade-offs
    - Implementation Details

Trends That Motivated LFS

- RAM sizes were growing
- Random I/O was slower than sequential I/O, and the gap seemed destined to widen

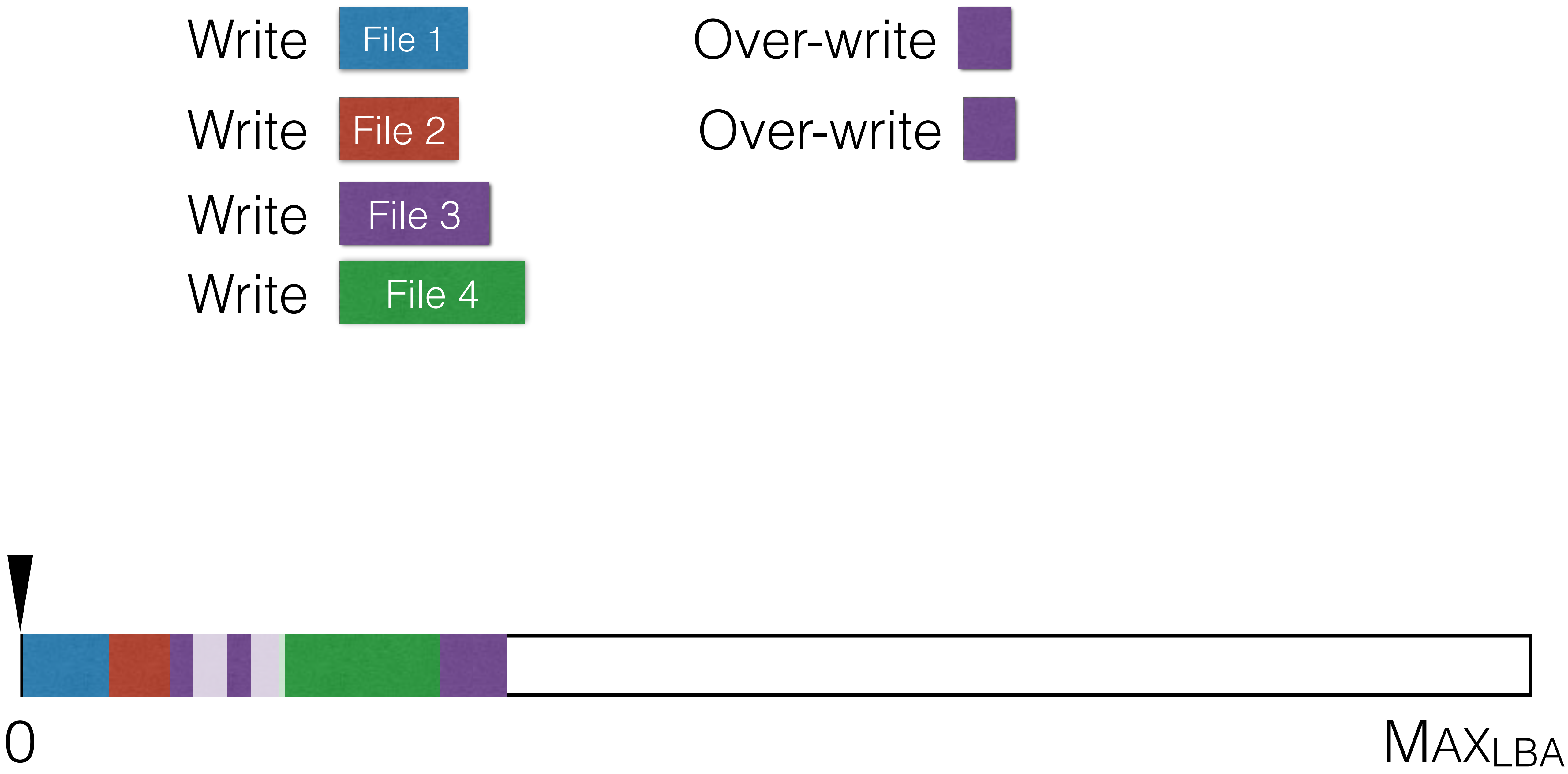
With more RAM, we can satisfy many of our reads from cache: optimizing writes is important.

High Level Idea

Treat the disk like an append-only circular log.

- All updates are written **out-of-place**
- **Garbage collect** stale data to reclaim space

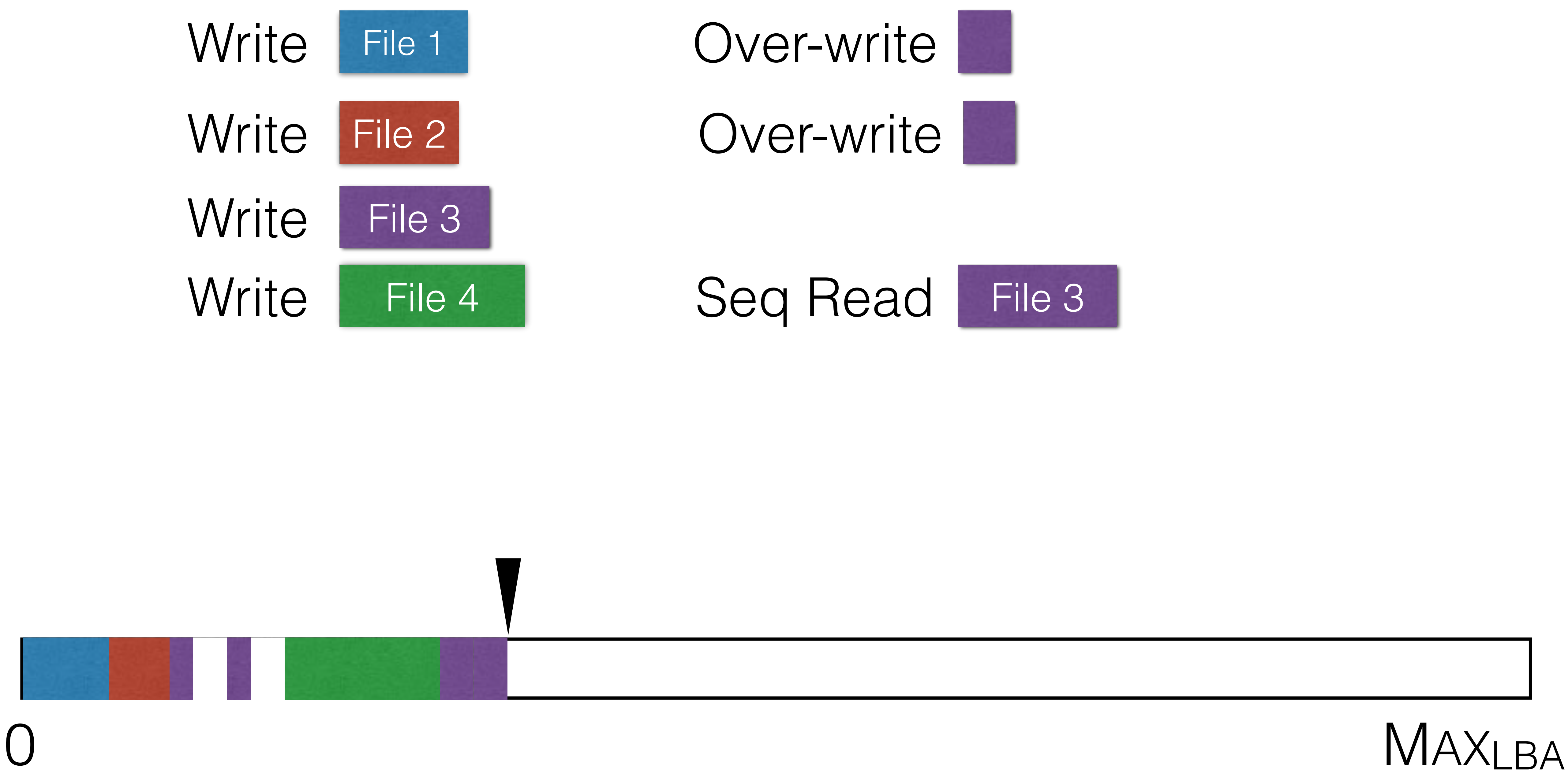
High Level Idea



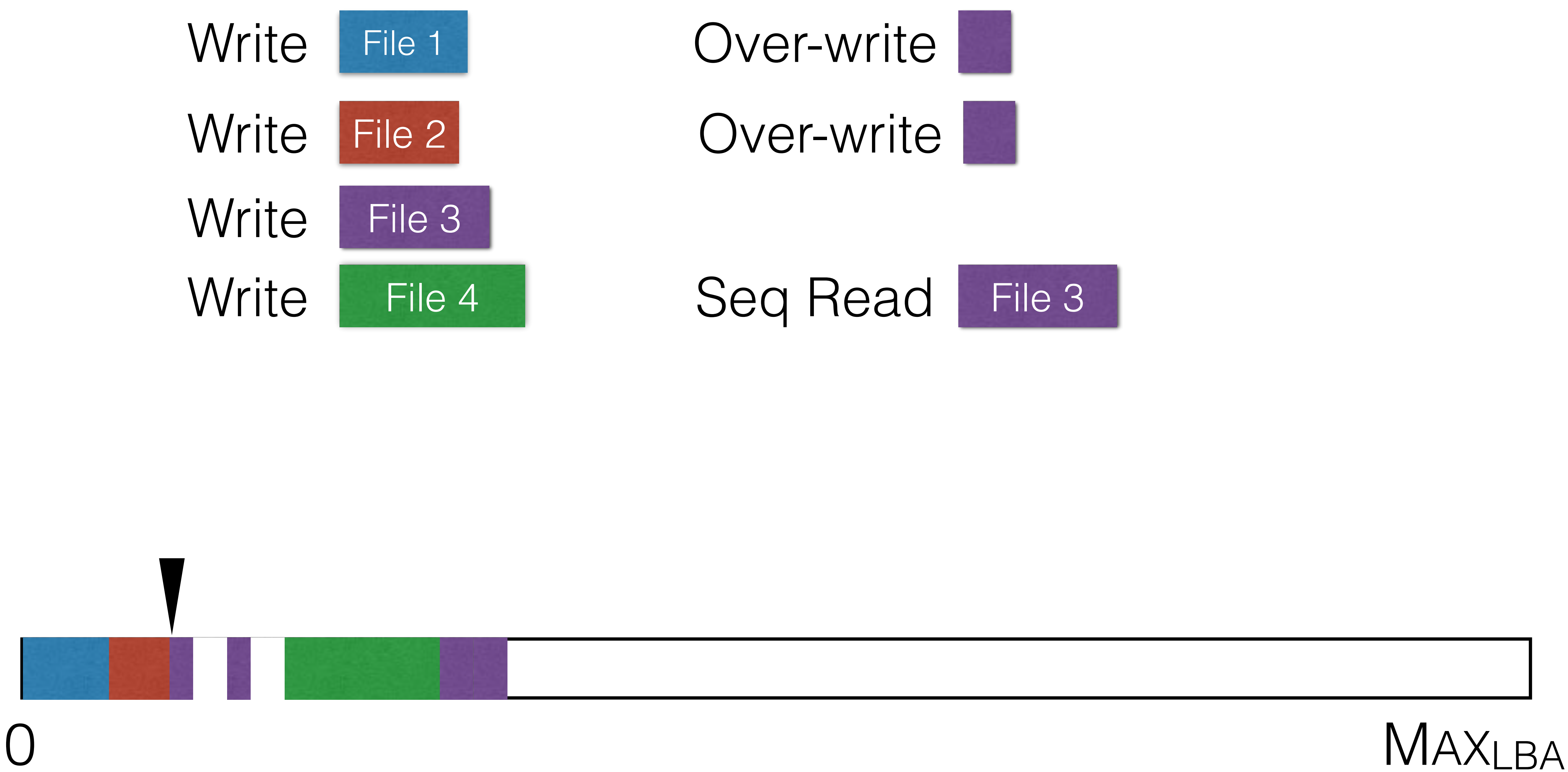
Observations

- Seeking to update data is expensive, but if we always append, we never need to seek to write new data!
- When we "overwrite" data, we write new version "**out-of-place**", logically deleting previous version
  - New versions may be far from logical neighbors

High Level Idea



High Level Idea



Design Tradeoffs

<u>Logging (e.g., LFS)</u>

- Sequential Writes ✓
- Random Writes ✓
- Sequential Reads ✗

LFS Challenges

The high level idea is relatively clear, but the details are where things become tricky:

- Sequential writes are good, but how do we avoid the performance penalty of writing small blocks?
- How do we reclaim data that is overwritten/deleted?
- How do we keep track of our metadata?
  - Do we write inodes out of place too?
  - How do we track the latest version of an inode?
  - How do we initialize our file system when we mount?

Avoiding Small I/Os Using Segments

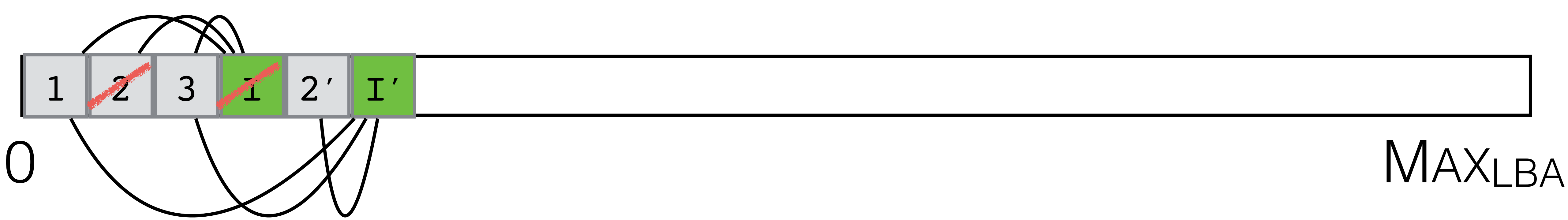
Segments: Buffering Updates

- Writing to consecutive LBAs does not guarantee we avoid I/O setup costs
- Example: write to LBA_x, …*wait*…, then write to LBA_{x+1}
  - While waiting, the disk rotates. May cost a full rotation to write LBA_{x+1}!
- Instead, keep all updates in memory until you've accumulated a sizable batch, then "stream" them
  - The LFS batch size is the segment
    - The optimal segment size is a function of the disk itself: we want to achieve a good fraction of the peak bandwidth by amortizing the setup cost

Managing Structures: Inodes and the Imap

Inodes & Out-of-Place Updates

- Workload: Write three new blocks to a file
- How do we know how to find them?
  - An **inode**. Since we just wrote new data, we need to write our new inode as well.
- If we overwrite those blocks, we don't immediately delete them. We write our new copies, and then write a new inode that refers to them.



Imap: Inode Index

- **Imap**: a structure that contains the address of the most recent version of each on-disk inode.
- Every file update creates a new version of an inode
  - We update the in-memory Imap to refer to this new version
  - We write the portion of the inode map that contains this entry so the Imap is persistent

Keeping Track of Metadata With Checkpoints

Checkpoints: Keeping LFS Consistent

Checkpoint region is the one part of the system that is written “in-place”

- Describes where the inode map lives
- Stores location of log start/end

Checkpoint writes alternate between 2 locations

- If LFS crashes during a checkpoint, it can safely start from the other checkpoint

Steps to Mounting an LFS instance

- Read both checkpoint regions; select most recent valid checkpoint
- Use checkpoint information to construct the imap
- Replay consistent segments from the log (starting after the tail of the checkpoint's log) to roll forward

Reclaiming Space with Garbage Collection

Reclaiming Garbage

- During our first pass through the disk, we can write sequentially
- But each time we overwrite a block, we create a "hole" in our log where the now "stale" version lives
  - When we get to the end of our disk and circle back, our free space is fragmented: many small holes
  - We could "stitch" our log through the holes, but holes might not be large enough to fit a segment
    - If we can't create large segments of free space, we would need to seek on our writes!

Segments & Garbage

- **Idea**: divide our log into segments, and **garbage collect** segments
- **Segment Summary Info**: each segment stores metadata that notes the inode number that corresponds to each of its blocks
  - To determine if a block is stale, read its inode and see if the inode points to this block (live) or to another block (stale)
  - To garbage collect a segment, migrate all live blocks to a new segment, update their inodes to point to that new copy, and then reclaim the old segment
    - This creates a large contiguous region of free space!

LFS Legacy

LFS ideas are everywhere

- SSD internals mirror segment design & restrictions
- SMR HDDs & Zoned Namespace SSDs have an API that exactly mirrors the LFS design
- F2FS is a modern log-structured file system backed by Samsung for phones & SSDs

Not Suitable For Everyone

- Garbage collection
  - Great if it can be done in the background, but not always possible
- Aging: file system performance degradation over time
  - Sequential read after random write
  - Defragmentation?