The early 90s

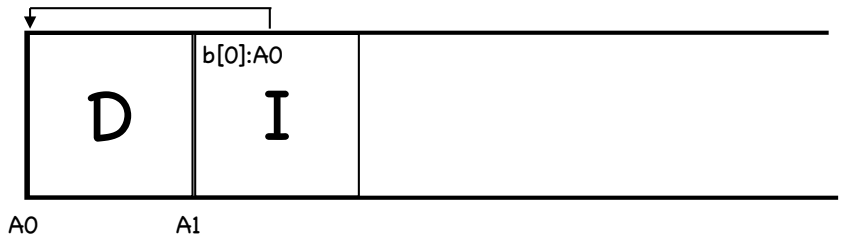
- Challenges
  - Existing FSs perform poorly on many workloads
    - many writes to create a file of 1 block; many short seeks
  - File systems are not RAID-aware:
    - big write amplification for RAID-4/5
- Opportunities
  - Growing memory sizes
    - file systems can afford large block caches
    - performance dominated by write performance
  - Growing gap in random vs sequential I/O
    - transfer bandwidth increases 50%-100% per year
    - seek/rotational delay decrease by 5%-10% per year

Log Structured File Systems

- Instead of adding a log to the existing FS disk layout, use all disk as a log
  - buffer all updates (including metadata!) into an in-memory segment
  - when segment is full, write to disk in a long sequential transfer to unused part of disk
- Never overwrite existing data
  - always write segments to free locations
  - much improved disk throughput

Log Structured File Systems

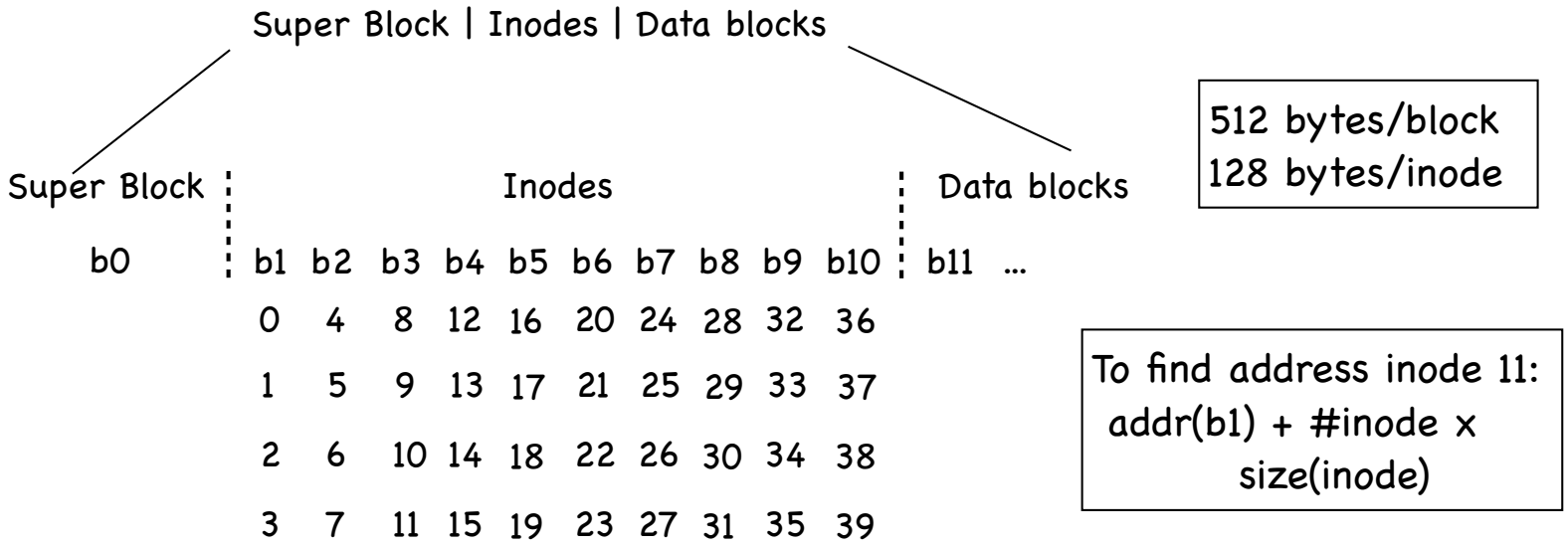
- But how does it work?
  - suppose we want to add a new block to a 0-sized file
  - not enough to write to log just the data block...
  - ...we have to update the inode too!
- LFS places both data block and inode in its in-memory segment



- Leverages write buffering to write a chunk of updates (a segment) all at once

Finding i-nodes

- in UFS, just index into inode array



- FFS is the same, with i-nodes divided among block groups and stored at known locations

- But in LFS i-nodes are scattered everywhere on disk!

Finding inodes in LFS

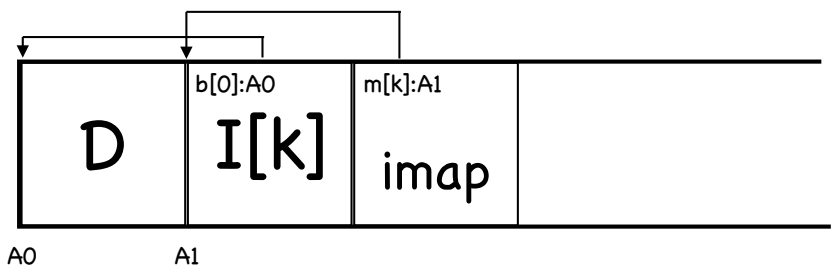
- Inode map: a table indicating where each inode is on disk: Imap(i#) -> disk address of i#
- Needs to be kept persistent... on disk! Where?

  Option 1: At a fixed location on disk

  we would have to seek to it often for updates

  Option 2: Just add pieces of the inode map to the log

  no seeks when writing Imap



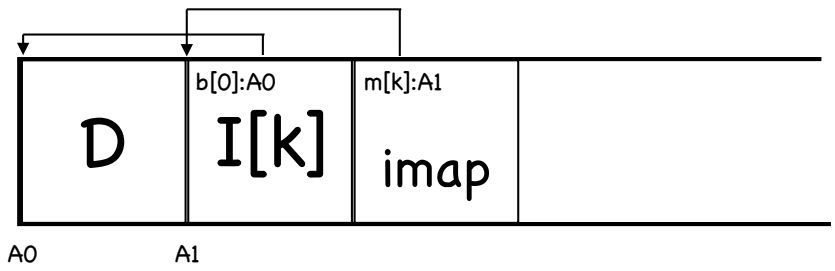
But now we need to find on disk the Imap pieces!

Finding the Imap in LFS

- Normally, Inode map cached in memory
  - On disk, its pieces can be found by accessing a fixed checkpoint region (CR)
    - CR contains pointers to pieces of Imap
    - updated periodically (every 30 seconds)

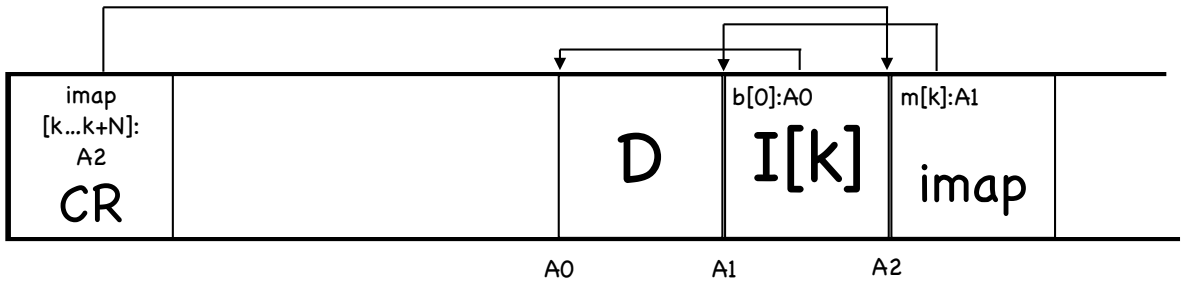
Finding the Imap in LFS

- Normally, Inode map cached in memory
  - On disk, its pieces can be found by accessing a fixed checkpoint region (CR)
    - CR contains pointers to pieces of Imap
    - updated periodically (every 30 seconds)

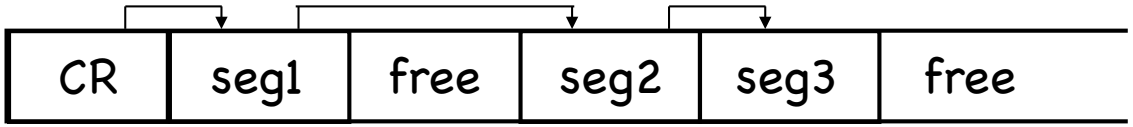


Finding the Imap in LFS

- Normally, Inode map cached in memory
  - On disk, its pieces can be found by accessing a fixed checkpoint region (CR)
    - CR contains pointers to pieces of Imap
    - updated periodically (every 30 seconds)



- Disk layout:



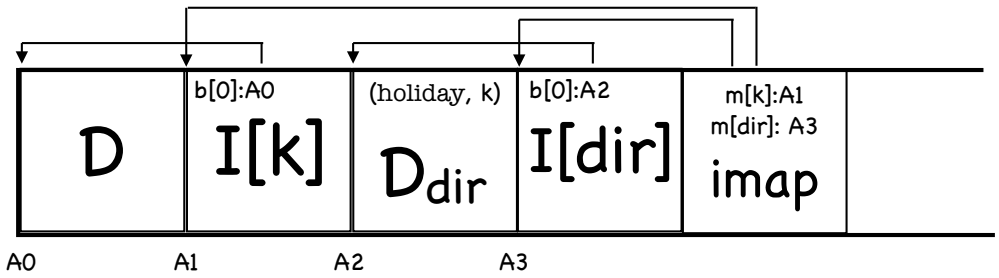
Reading from disk in LFS

- Suppose nothing in memory...
  - read checkpoint region
  - from it, read and cache entire inode map
  - from now on, everything as usual
    - read inode
    - use inode's pointers to get to data blocks
- When the imap is cached, LFS reads involve virtually the same work as reads in traditional file systems

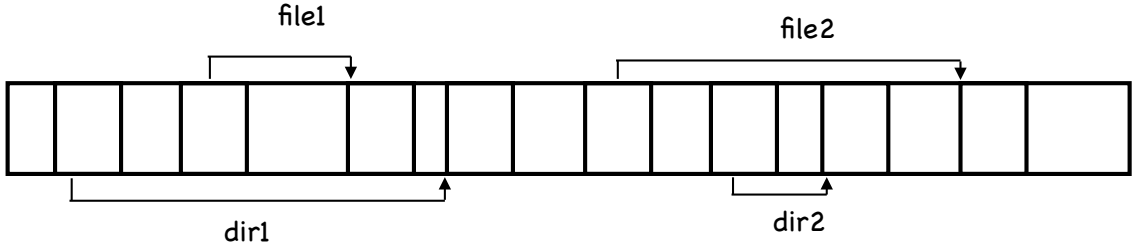
modulo an imap lookup

Directories

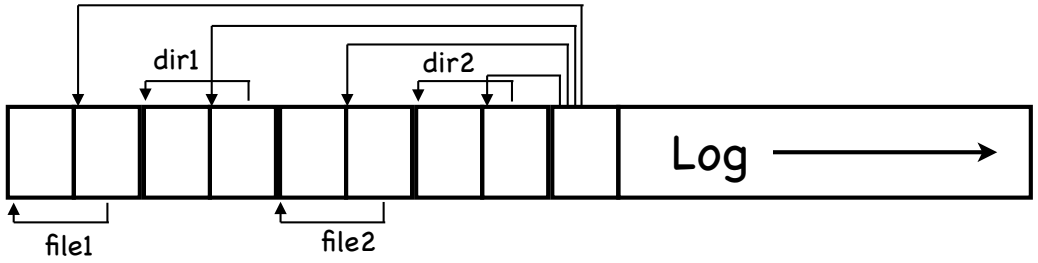
- Creating file holiday in a directory requires writing
  - file's inode
  - file's data
  - update data block of file's directory
  - update directory's inode
- LFS just writes this info to the log in a segment



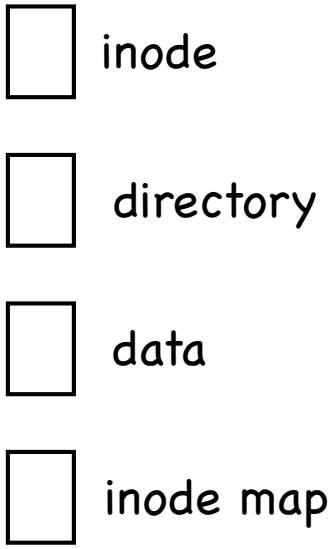
LFS vs UFS



Unix File System



Log-structured File System



Blocks written to create two 1-block files: dir1/file1 and dir2/file2 in UFS and LFS

Garbage collection

- As old blocks of files are replaced by new ones, segment in log become fragmented
- Cleaning used to produce contiguous space on which to write
  - compact M fragmented segments into N new segments, newly written to the log
  - free old M segments
- Cleaning mechanism:
  - How can LFS tell which segment blocks are live and which dead?
- Cleaning policy
  - How often should the cleaner run?
  - How should the cleaner pick segments?

Who's alive?
Segment Summary Block

- Found at the beginning of each segment
  - Written once, when segment is written; never updates
- For each data block in segment, SSB holds
  - The file the data block belongs to (inode#)
  - The offset (block#) of the data block within the file
- During cleaning, to determine whether data block D is live:
  - find D's inode# I and block# in SSB
  - use imap to find where inode I is currently on disk
  - read inode I (if not already in memory)
  - check whether a pointer for block block# refers to D's address

Which segments to clean, and when?

- When? Many options...
  - when disk is full
  - periodically
  - when you have nothing better to do
- Which segments?
  - utilization: how much it is gained by cleaning
  - age: how likely is the segment to change soon
    - better to wait on cleaning a "hot" block, since free blocks are going to quickly build up again
    - better cleaning a "cold" block, even if it has fewer dead blocks, since the remaining blocks are likely to stay alive longer

Crash recovery

- The journal is the file system!
- On recovery
  - read checkpoint region
    - may be out of date (written periodically)
    - may be corrupted (crash occurred while updating CR)

Crash recovery: corrupted CR

- Keep two CRs, at the beginning and end of disk
  - updated alternately
- When updating CR
  - write a timestamp block
  - write CR
  - write a second timestamp block
- Inconsistent timestamps indicate corruption
- Read most recent CR with consistent timestamp blocks

Crash recovery: out-of-date CR

- On recovery
  - read latest uncorrupted CR
  - roll forward
    - start from where checkpoint says log ends
    - read through next segments to find valid updates not recorded in checkpoint
      - when a new inode is found, update imap
      - when a data block is found that belongs to no inode, ignore it