Log-Structured File Systems

Anda Iamnitchi

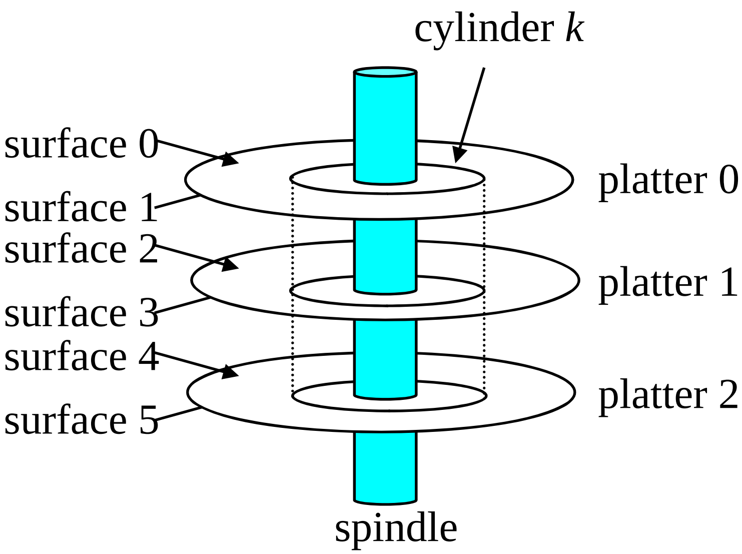
Basic Problem

- Most file systems now have large memory caches (buffers) to hold recently-accessed blocks
- Most reads are thus satisfied from the buffer cache
- From the point of view of the disk, most traffic is write traffic
- So to speed up disk I/O, we need to make writes go faster
- But disk performance is limited ultimately by disk head movement
- With current file systems, adding a block takes several writes (to the file and to the metadata), requiring several disk seeks

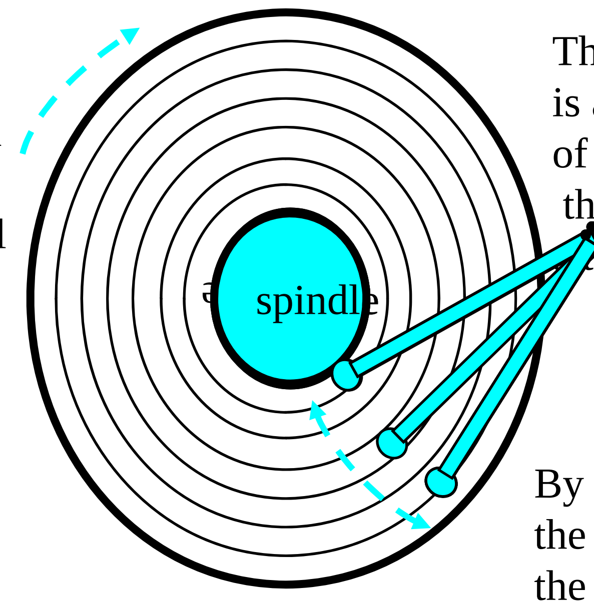
Motivation for Log-FS

- Technology:
  - Disk is the bottleneck (1): Faster CPU, faster memory (thus cache), larger disks, but not faster disk.
- Workloads: random access to the disk (2): small reads and writes
- Existing file systems:
  - Spread information on disk (3)
  - Synchronous writes (4)

Disk is the Bottleneck (1)



The disk surface spins at a fixed rotational rate



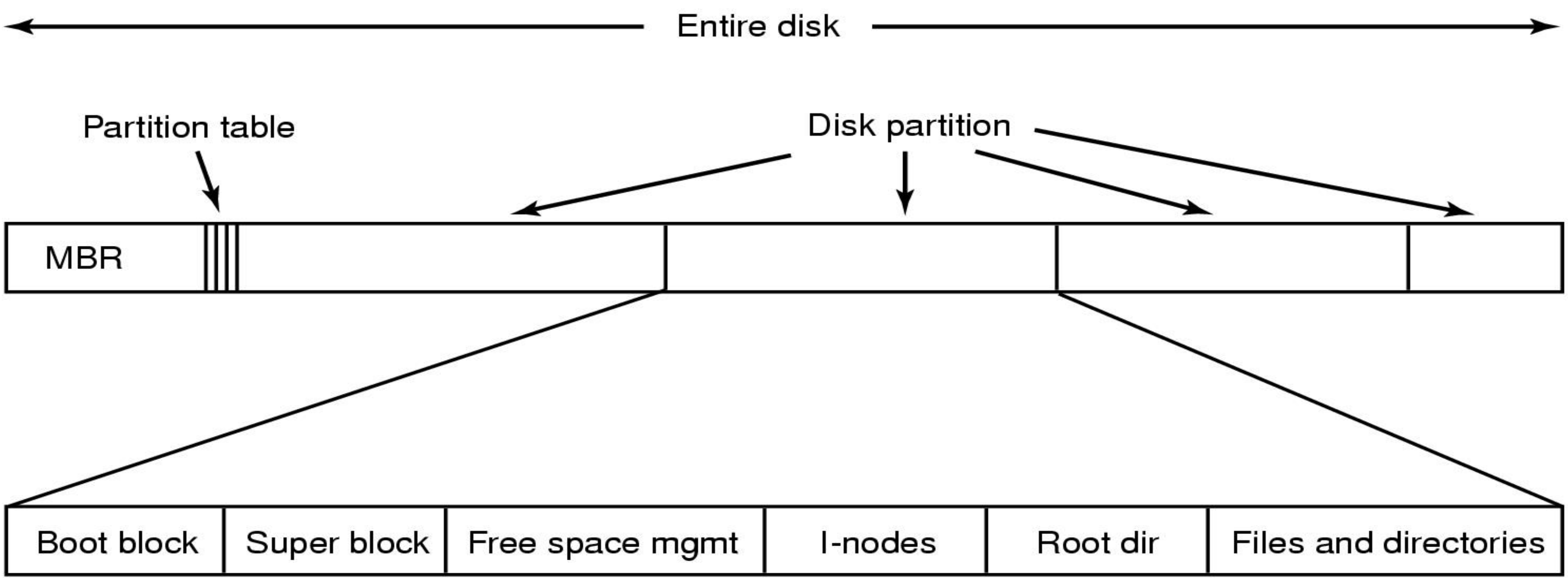
The read/write *head* is attached to the end of the *arm* and flies over the disk surface on a thin cushion of air.

By moving radially, the arm can position the read/write head over any track.

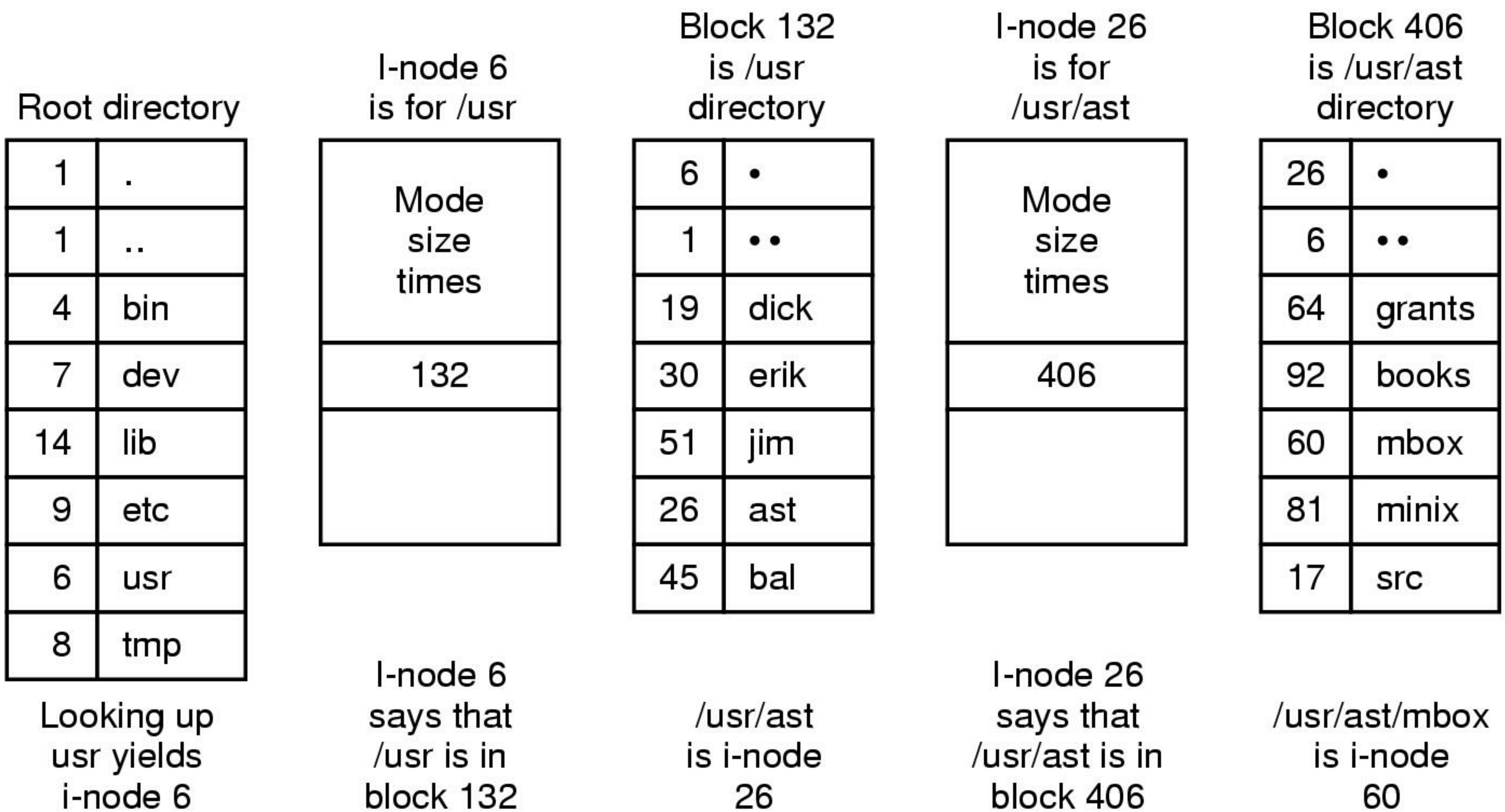
Small Reads and Writes (2)

- Typical office and engineering applications
  - Small files

Existing File Systems (3): Spread Information on the Disk



Looking up */usr/ast/mbox* in UNIX



Existing File Systems (4): Synchronous Writes

- For metadata

LFS: Basic Idea

- An alternative is to use the disk as a *log*
- A log is a data structure that is written only at the head
- If the disk were managed as a log, there would be effectively no head seeks
- The “file” is always added to sequentially
- New data and metadata (inodes, directories) are accumulated in the buffer cache, then written all at once in large blocks (e.g., segments of .5M or 1M)
- This would greatly increase disk throughput
- The paper: How does this really work? How do we read? What does the disk structure look like? How to recover from crash? Etc.

Issues (1): Retrieving information from Logs

- (since data and metadata are written together, sequentially)
  - inode map records current location of each inode
  - the inode map itself is divided into blocks written on the disk
  - a fixed region on each disk keeps track of all inode map blocks
  - inode map small enough to fit into the memory

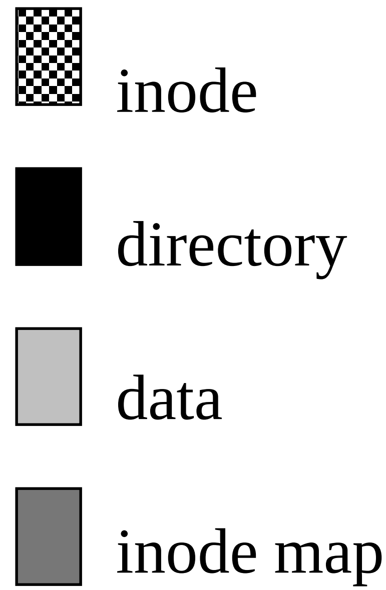
Issues(2): Manage free space

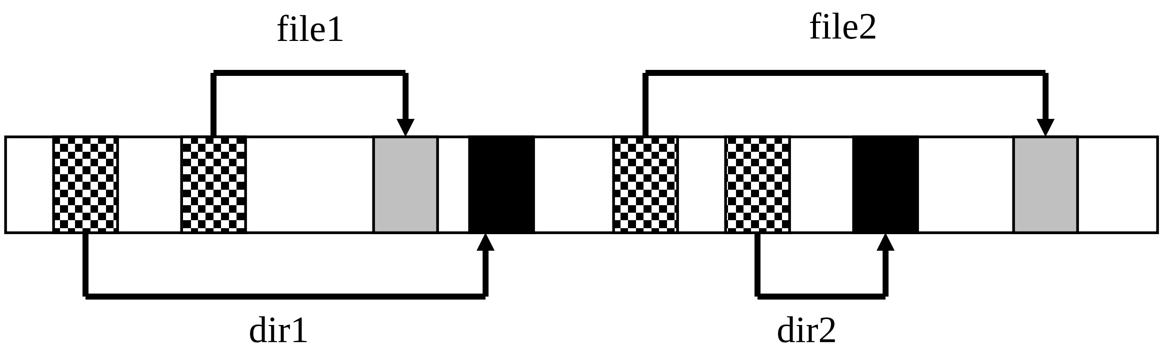
- Through a combination of threading and copying
  - fixed-size extents called segments (512 KB or 1 MB)
  - identify live data from segments
  - **copy** live data in a compacted form and **clean** the remaining segments

LFS Data Structures

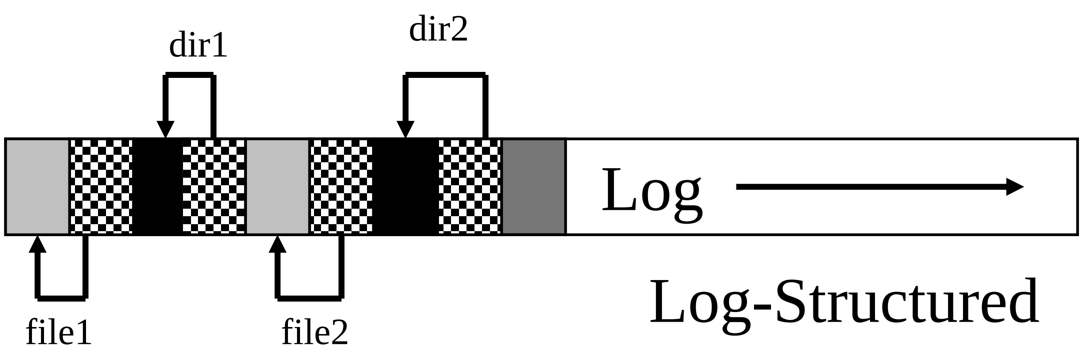
- **inodes:** as in Unix, inodes contain physical block pointers for files
- **inode map:** a table indicating where each inode is on the disk
  - inode map blocks are written as part of the segment; a table in a fixed checkpoint region on disk points to those blocks
- **segment summary**: info on every block in a segment
- **segment usage table**: info on the amount of “live” data in a block

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

LFS: Read and Write

- Every write causes new blocks to be added to the current segment buffer in memory; when that segment is full, it is written to the disk
- Reads are no different than in Unix File System, once we find the inode for a file (in LFS, using the inode map, which is cached in memory)
- Over time, segments in the log become fragmented as we replace old blocks of files with new block
- Problem: in steady state, we need to have contiguous free space in which to write

Cleaning

- The major problem for a LFS is *cleaning*, i.e., producing contiguous free space on disk
- A cleaner process “cleans” old segments, i.e., takes several non-full segments and compacts them, creating one full segment, plus free space
- The cleaner chooses segments on disk based on:
  - utilization: how much is to be gained by cleaning them
  - age: how likely is the segment to change soon anyway
- Cleaner cleans “cold” segments at 75% utilization and “hot” segments at 15% utilization (because it’s worth waiting on “hot” segments for blocks to be rewritten by current activity)

Segment Cleaning

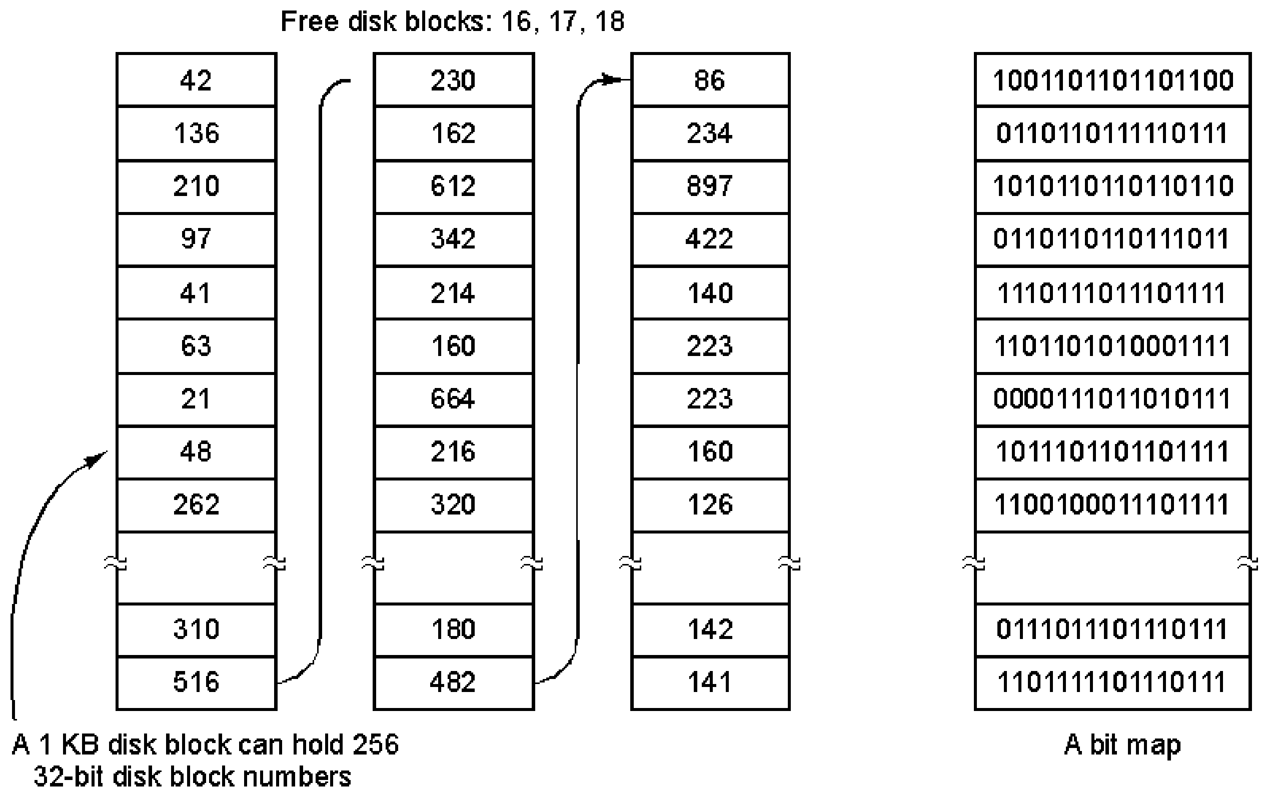
Segment summary block as part of each segment: identifies each piece of information in the segment

- useful for crash recovery, as well
- can be more than one summary block per segment (each summary block corresponds to one log write; if the segments are larger than the number of dirty blocks buffered in the file cache, and thus more than one log write fits in the segment.
- specifies for each block what it is: e.g., for each file data block it specifies the file number and the block number within the file
- distinguishes between live blocks and deleted or overwritten blocks

Segment usage table: a table records for each segment the number (count) of live bytes in the segment and the most recent modified time of any block in the segment

- These values are used by the segment cleaner when choosing segments to clean.
  - If count == 0, segment can be reused without cleaning
- Segment usage table is saved in the log, but the addresses of the blocks of the segment usage table are saved in the checkpoint region.

Free Space Maintenance in Traditional FS?



(a) (b)

Crash Recovery (in general)

- **Consistency checking** – compares data in directory structure with data blocks on disk, and tries to fix inconsistencies
  - Can be slow and sometimes fails
- Use system programs to **back up** data from disk to another storage device (magnetic tape, other magnetic disk, optical)
- Recover lost file or disk by **restoring** data from backup

Crash Recovery in Log-FS

- Last few operations are always at the end of the log
- **Checkpoint**: position in the log where all file system structures are consistent and complete
  - To create a checkpoint:
    - Writes all information to the log
    - Writes a checkpoint region to a special fixed position on disk
- **Roll-forward**:
  - Scans through the records written in the log after the last checkpoint
  - Uses *directory operation logs*

Group Work: File System Operations

1. Open an existing file
2. Create a new file
3. Remove a file
4. Rename a file
5. Modify an existing file
6. Modify a directory

LFS Summary

- Basic idea is to handle reads through caching and writes by appending large segments to a log
- Greatly increases disk performance on writes, file creates, deletes, ….
- Reads that are not handled by buffer cache are same performance as normal file system
- Requires cleaning demon to produce clean space, which takes additional cpu time