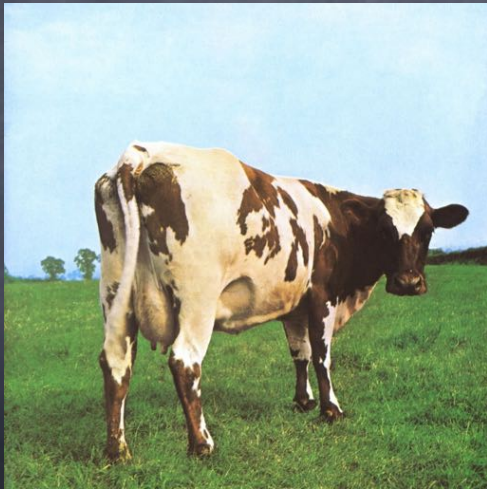


# Back to

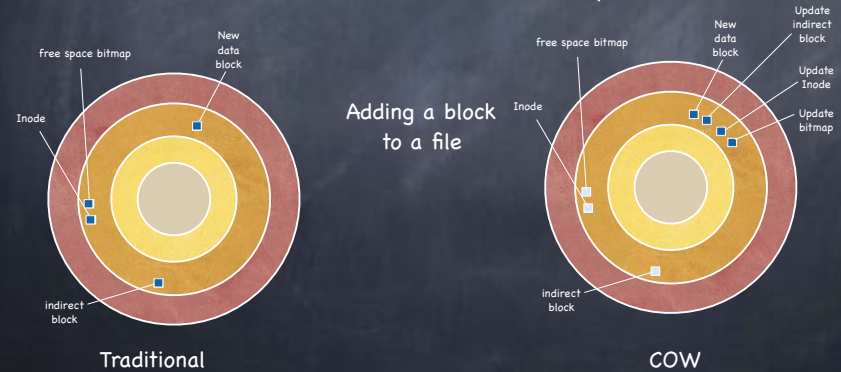


Where is this from?



# COW File Systems (copy-on-write)

- Data and metadata not updated in place, but written to new location
  - transforms random writes into sequential writes

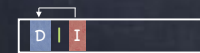


# The early 90s

- Growing memory sizes
  - file systems can afford large block caches
  - most reads can be satisfied from block cache
  - performance dominated by write performance
- Growing gap in random vs sequential I/O performance
  - transfer bandwidth increases 50%-100% per year
  - seek and rotational delay decrease by 5%-10% per year
  - using disks sequentially is a big win
- Existing file system perform poorly on many workloads
  - 6 writes to create a new file of 1 block
    - ▶ new inode | inode bitmap | directory data block that includes file | directory inode | new data block storing content of new file | data bitmap
  - lots of short seeks

# Log Structured File Systems

- Use 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
- Virtually no seeks
  - much improved disk throughput
- But how does it work?
  - suppose we want to add a new block to a 0-sized file
  - LFS paces **both data block and inode** in its in-memory segment

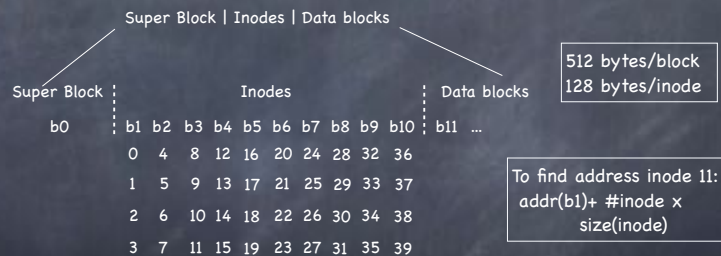


Fine.

But how do we find the inode?

# Finding inodes

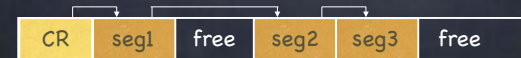
- in UFS, just index into inode array



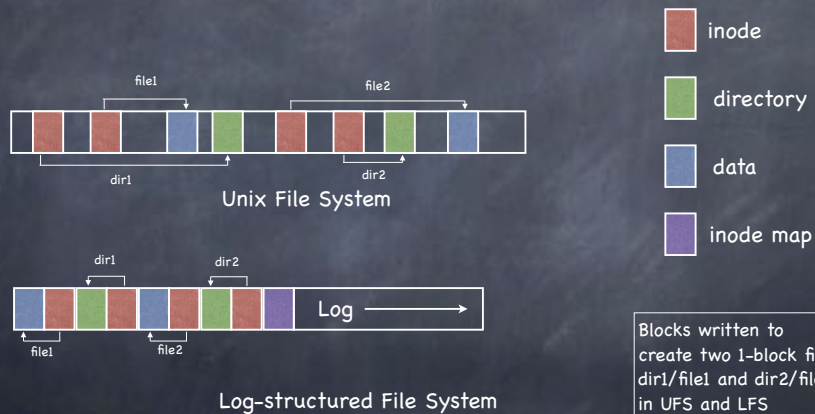
- Same in FFS (but Inodes are at divided (at known locations) between block groups)

# Finding inodes in LFS

- Inode map:** a table indicating where each inode is on disk
  - Inode map blocks are written as part of the segment
  - ... so need not seek to write to imap
- but how do we find the blocks of the Inode map?
  - Normally, Inode map cached in memory
  - On disk, found in a fixed **checkpoint region**
    - updated periodically (every 30 seconds)
- The disk then looks like



# LFS vs UFS



# 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

# Garbage collection

- ④ As old blocks of files are replaced by new, segment in log become fragmented
- ④ **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
- ④ Cleaning mechanism:
  - ❑ How can LFS tell which segment blocks are live and which dead?
    - ▶ Segment Summary Block
- ④ Cleaning policy
  - ❑ How often should the cleaner run?
  - ❑ How should the cleaner pick segments?

# Segment Summary Block

- ④ Kept at the beginning of each segment
- ④ 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
- ④ During cleaning, to determine whether data block D is live:
  - ❑ use inode# to find in imap where inode is currently on disk
  - ❑ read inode (if not already in memory)
  - ❑ check whether a pointer for block block# refers to D's address
- ④ Update file's inode with correct pointer if D is live and compacted to new segment

# Which segments to clean, and when?

- ④ When?
  - ❑ when disk is full
  - ❑ periodically
  - ❑ when you have nothing better to do
- ④ Which segments?
  - ❑ utilization: how much it is gained by cleaning
    - ▶ segment usage table tracks how much live data in segment
  - ❑ age: how likely is the segment to change soon
    - ▶ better to wait on cleaning a hot block, since free blocks are going to quickly reaccumulate

# Crash recovery

- ④ The journal is the file system!
- ④ On recovery
  - ❑ read checkpoint region
    - ▶ may be out of date (written periodically)
    - ▶ may be corrupted
      - 1) two CR blocks at opposite ends of disk / 2) timestamp blocks before and after CR
      - use CR with latest consistent timestamp blocks
  - ❑ 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