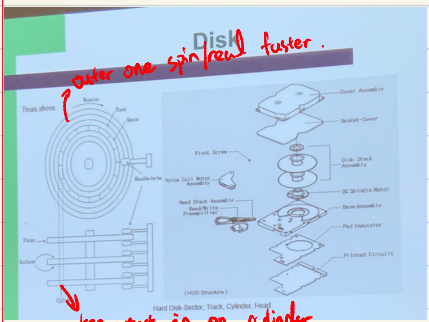
11/4

- Marshall K. McKusick, William N. Joy, Samuel J. Leffler, and Robert S. Fabry, "A Fast File System for Unix," ACM Transactions on Computer Systems, 2(3), August 1984, pp. 181-197.

Q: In FFS, reading is always at least as fast as writing. In old UFS, writing was 50% faster. Why is this?

- Mendel Rosenblum and John K. Ousterhout, "The Design and Implementation of a Log-Structured File System," Proceedings of the 13th ACM Symposium on Operating Systems Principles, December 1991.

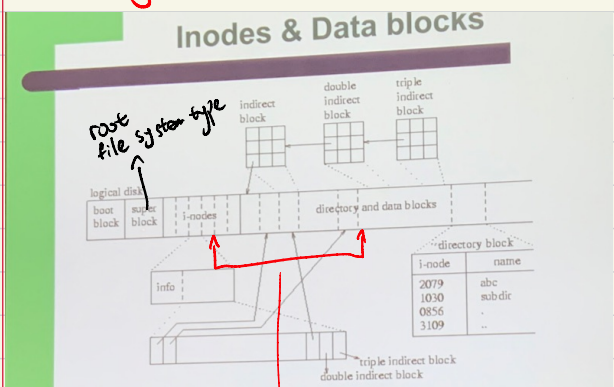
Q: When we want to read a block in LFS on disk, how do we figure out where it is?



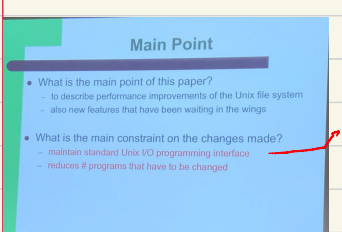
outer one spin fast faster.

keep most in on cylinder

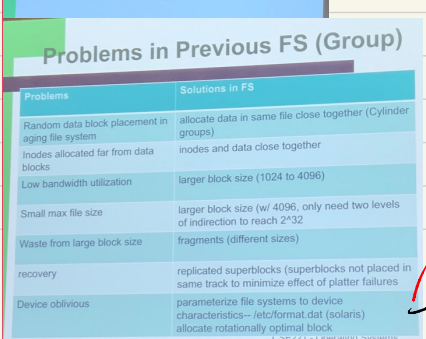
moving head is expensive.



too far between inode and data.



keep same interface.



abandon since disk are smarter as well.
disk need to relocate head to a diff plate

e.g. giving some padding since disk is spinning.

FFS Allocation

- FFS has a two-level allocator, global and local. What does the global allocator try to achieve?
  - localize related data within a cylinder group
    - e.g., all inodes for a directory in same cylinder group
  - distribute unrelated data across cylinder groups
    - e.g., new directories placed in cyl group with >avg #inodes
- What does the local allocator try to achieve?
  - optimal local placement
    - rotationally optimal block in same cylinder
    - block in same cyl group
    - search through cyl groups
- What is one requirement of the new FFS?
  - requires a minimum of free space so that cyl block allocations can be done: 10%
  - this is why you will see some file systems with >100% utilization

11/4/21

Performance

- The paper states that the throughput rate is strongly tied to the amount of free space maintained. Why?
  - if free space is low, blocks have to be spread across cyl groups
- In FFS, reading is always at least as fast as writing. In old UFS, writing was 50% faster. Why is this?
  - writes are async. FS can reorder requests to minimize total seek time
  - In FFS, blocks are placed on disk in a much better layout
    - even with sync reads, placement roughly corresponds to a good ordering of disk requests
    - writes have higher overhead because they must allocate new blocks

write cache, wait and write it once.

New Features

- Interesting to see that these features were not in the original Unix FS implementation:
  - long file names
  - file locking
    - with advent of NFS, became much less useful
  - symbolic links
  - rename!
  - quotas

Discussion

- Some of these things are obsolete. Which ones?
  - device parameters
    - cylinders in today's disks vary in size depending upon radius
    - host no longer knows cylinder and track boundaries
  - rotation allocation not done
    - blocks allocated sequentially
    - drives do read ahead if there is a gap in seq commands
- Current Unix filesystems (and NT) are extent-based w/ journaling
  - extents
    - allocate multiple consecutive blocks on disk
    - can read/write multiple blocks at once to amortize seek (cf VMS, LFS)
  - journaling: as described in related work in LFS paper

// second

Before we get into this paper...

- LFS spawned a vocal controversy between Ousterhout and Seltzer about whether LFS was better than an extent-based FFS
- The main complaint was that cleaning required too much overhead, and that it was a source of hidden costs.
- The issue has never been settled definitively, although most disk-based file systems today are extent-based FFS rather than LFS.

Overview

- What is the main point of this paper?
  - file system designed to exploit h/w and workload trends
- What are these trends?
  - disk b/w is scaling, but latency is not
  - large main memories
    - large file buffer caches
    - absorb a greater fraction of read requests
    - use for write buffers as well
    - coalesce small writes into large writes
  - workloads
    - small file access

CSE221 - Operating

Seq: ok

Small I/O: stop. get some help

Buffer cache => hit them, no need wonk

Motivation

- What are problems with FFS?
  - layout
    - possibly related files are physically separated
    - inodes are separate from files
    - directory entries are separate from inodes
    - bottom line is only 5% disk b/w utilization
      - but FFS says 15-50%...what gives?
      - small vs large I/Os, accounting for seeks to inodes then data blocks
  - synchronous writes
    - metadata requires synchronous writes
    - w/ small files, most writes are metadata and hence
      - synch writes defeat write caches

CSE221 - Operating

synch → less to follow the sequence.

A New Motivation

- New type of memory technology
  - SSD, Flash Memory, etc

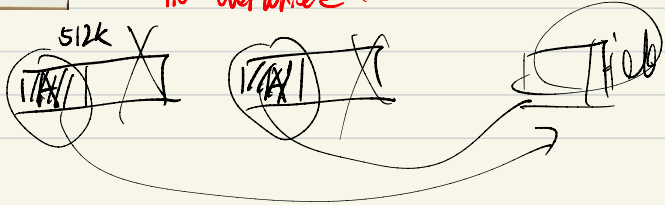
Disks	Flash, SSD, etc
(+) Lost cost per bit	(+) Random Access
(-) Mechanical movement (SPM & VCM)	(+) Non-volatile
(-) High power consumption (10-15W)	(+) Low Power Consumption (2W)
(-) Heavy weight compared to flash	(-) Erase before Write
	(-) Erasing operation in the unit of block (not page)
	(-) Maximum # of erase operations per cell
	(-) High cost per bit

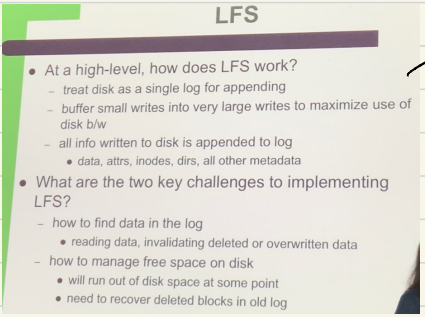
erase then write

so, if there is a hot spot, frequently update. data will not be readable

LFS avoid hot spot

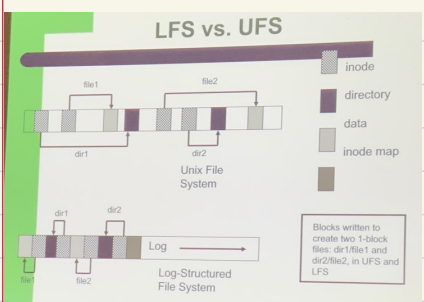
no overwrite.





→ the IMAP (all pointer to INODE)

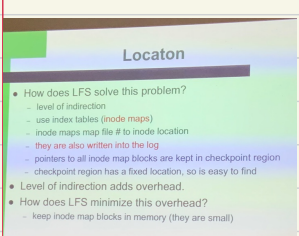
will always write to the end of the disk.

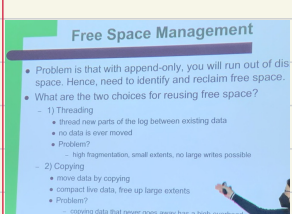


JFS: Both used

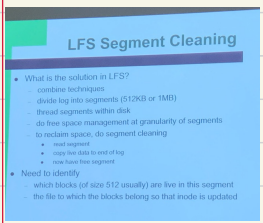
has a buffer like LFS

other file are normal FFS





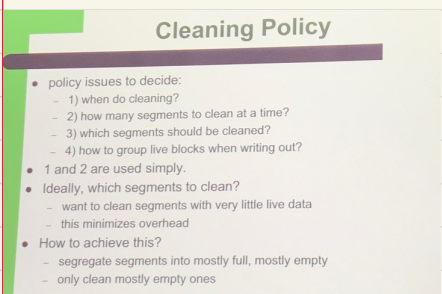
Solution



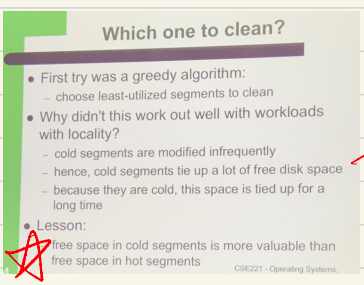
threading between segment

copy on segment

guarantee to have at least one seg SEQ



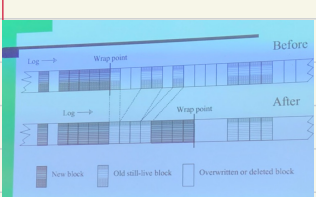
But not working?

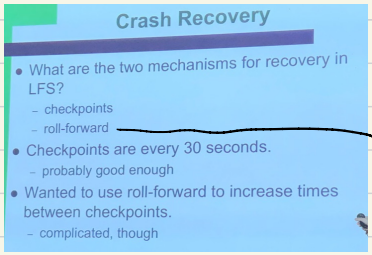


It is because recent one might be replace Quickly,

just write and you don't even need cleaning!

?





Checkpoint is at fixed location

since we may circle the disk, we need to scan the whole disk.

the checkpoint may not be the latest, but we can scan and catch from that point.