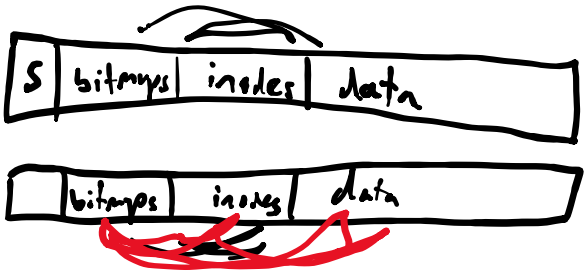
FSCK lab: this Friday

Final: timed take-home comprehensive (study packet)

Log-structured FS (WAFL, ZFS, btrfs) FTL (SSDs)

FFS: treat disk like a disk

Split related files into block group



Creating a 1-block file

- read dir inode
- read dir contents (check for unique)

read inode bitmap

write inode bitmap

write dir data

write dir inode (updateLength)

read data bitmap

Still lots of seeks/rotates

What happens with cache

reads: avoids

writes: delayed + scheduled ← all over the disk

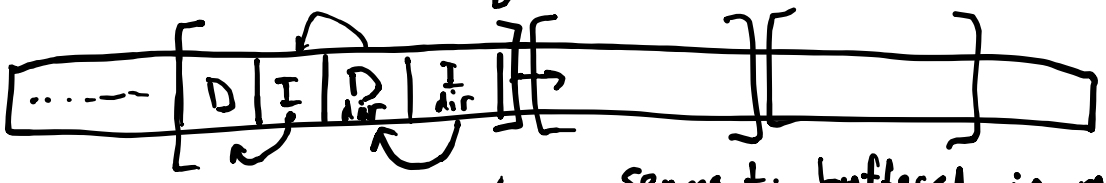
Real key to performance: use disk sequentially

reads: hard from disk (but reads might go away with cache)

writes: easy: we can choose

For now assume: all reads from cache

How to write sequentially



segment (1MB) e.g.

segment: buffered in memory to amortize away disk rotation

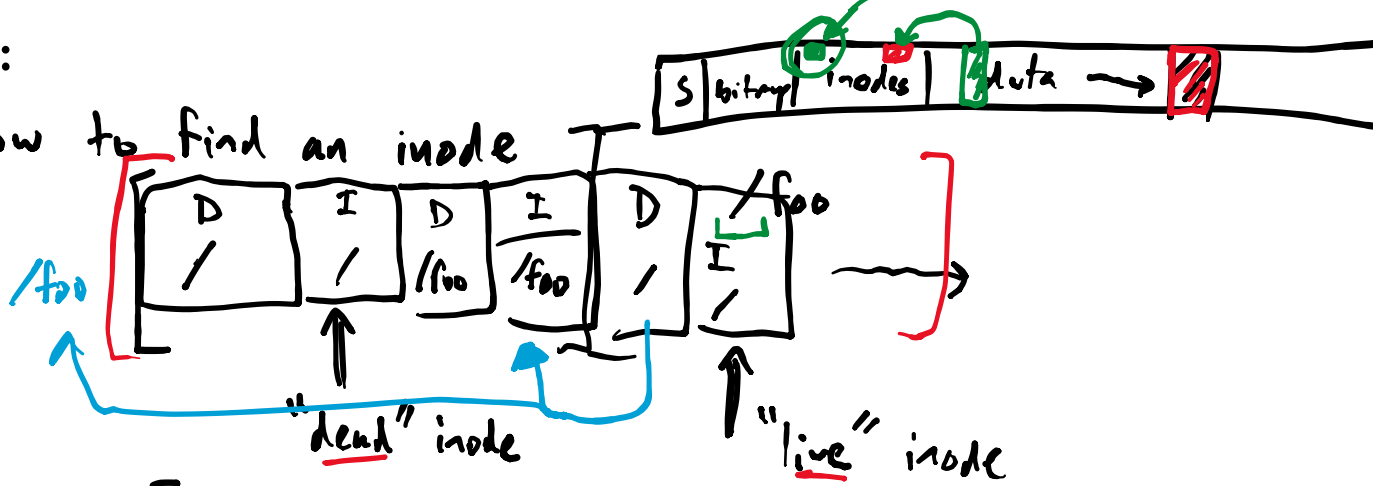
Protocol: 1. put data/inode into in-memory segment

2. when seg is full, write to disk

Problems:

1. how to find an inode

well known



inode map[inode #] = addr latest copy of inode

→ compact enough to fit in memory

Protocol: Checkpoint Region CR

1. fill segment, write inode (in mem)
2. write seg (seq.)

[CR → inode map] → inode → file

3. Occasionally (30 sec.) update CR with inode map

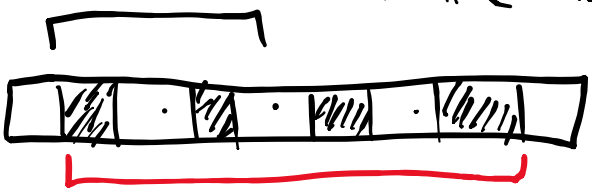
Crashes: 1. between CR update
no problem, but might lose 30 sec. of Fs operation
↳ roll forward

2. during CR update
similar (Tx)

2. disk gets full

Cleaner (garbage collector)
goal: find dead blocks
& make them available for reuse

how to tell if they're dead?
inode → look in latest imap
data → is live inode pointing to data block



Fragmentation

read N segments, write M live segment

how big should segment be?

which blocks to clean? hot/cold

bug vs. feature

WAFL
↳ versioned/time travel FS

COW
↳ copy on write

Flash (SSD) operation

read: random access 10 µs

write { erase: (before writing, must erase entire block) — few ms
program: write data 100s µs

reads/writes → flash operations
cheap — expensive
↳ FTL
flash translation layer

FS
block IO

[Direct Mapped FTL]
- writes are expensive!
- reliability: wear out
  wear-leveling

Most FTLs are [log-structured] (but FS not)
only write full blocks to end of log
random access reads are fast

flash devices have in-memory mapping
GC still a problem → lots of choices

Trends
1. FS API is loved
2. OS has a lot of flexibility w/ imp
3. Devices are getting smarter