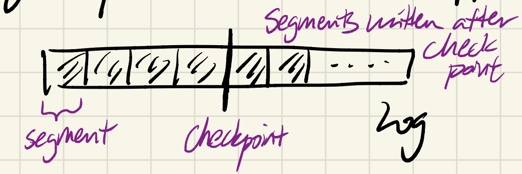
12/1/23

LFS Review

→ large sequential writes : append updates the log (datablock, inode, inode map)

↳ cached in memory



(log contains multiple versions to metadata / data)

⇓

checkpoint region (fixed loc)

→ loc of inode map pieces at time of checkpoint

→ tracks last checkpointed segment

How do we update the Checkpoint Region?

→ CR spans over multiple blocks, update must be atomic!

→ txn_begin [timestamp]

→ CR blocks

→ txn_commit [timestamp]

upon a crash, if begin & commit has matching timestamps, the tx is valid! Otherwise, CR is invalid.

☆ What to do when CR is invalid?

If there's only 1 CR, we no longer have a consistent checkpoint of our filesystem

→ 2 CRs, toggle update

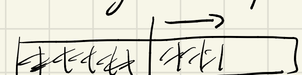
→ both valid, pick newer one

→ only 1 valid, pick the valid one

→ is it possible for both to be invalid?

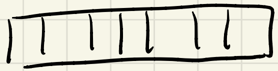
☆ Recall that CR is written at an infrequent interval ⇒ may recover consistent but stale fs state!

→ roll forward by apply valid segments past the checkpoint



checkpoint

Garbage Collection

→ Segment 

some blocks are live & some are garbage

↓

each has a segment summary (sometimes multiple...)
tracking each data block's inode # & offset

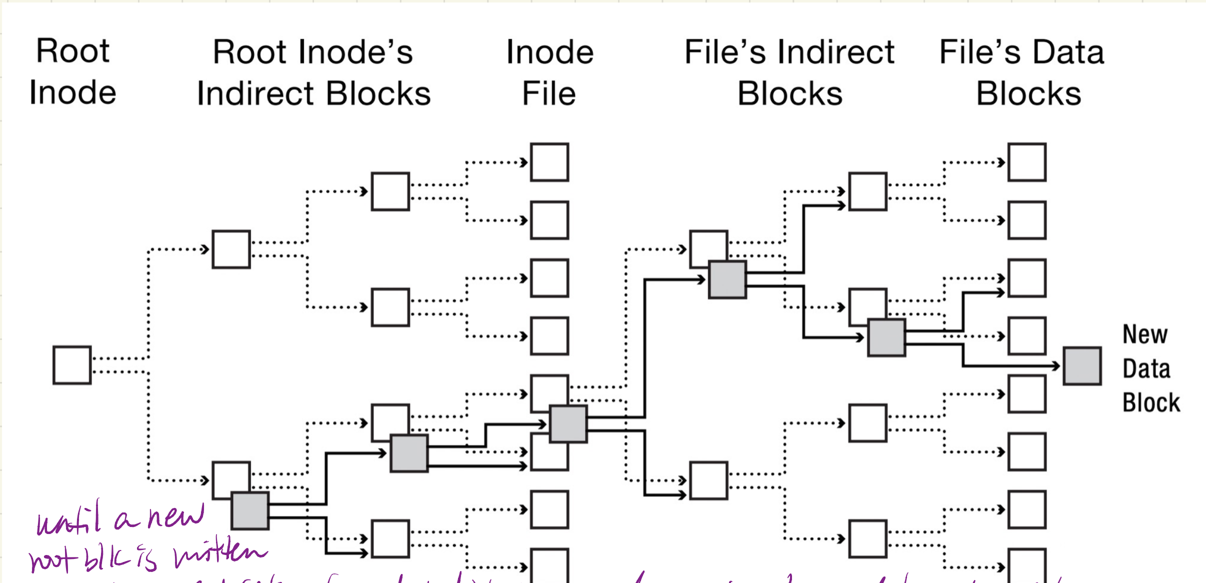
if we look up the inode map using this info and
find a match, block is live, otherwise, block is garbage

→ compact live blocks
within multiple segments,
write into a new segment!

→ threshold for compaction
if 90% are live, probably shouldn't compact

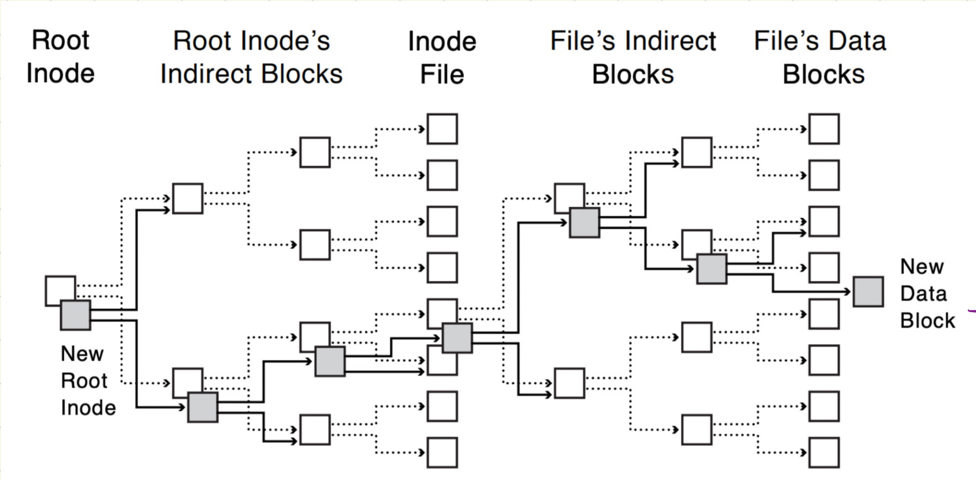
→ hot vs. cold segment
some data might be updated more frequently,
delay compaction could see more garbage.

→ one more cow fs : ZFS



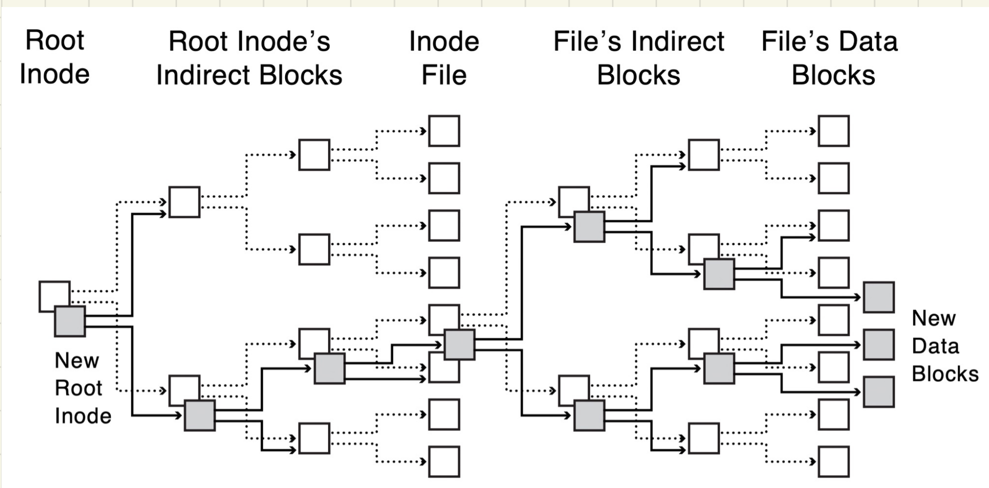
until a new root blk is written & point to new sets of updated blocks, no change is observed by the filesys.

→ actually carries out recursive update (no logical ptr like LFS's inode map)



↙ updates appear atomically after the new root inode becomes active

→ isn't this a lot of disk writes for just adding one new data block?



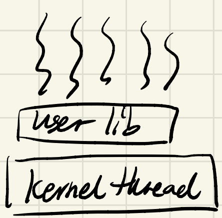
* buffer more updates in memory to amortize the cost of path rewrite!

Also supports logging for perf.
→ logical logging (only log operations, not changed blocks)

User Level Threads

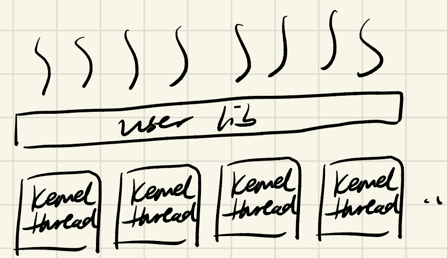
→ kernel threads (TCB)
  → pthread API, managed & scheduled by kernel
  → creation ⇒ system call, every context switch involves a mode switch (kernel → user)
  → MB of stack

→ user threads
  → managed & created via user libraries & runtimes
  → smaller / adaptive stack size
  → every context switch & creation is just a procedure call



N user threads
→ own stack
→ own rip (func)
→ own state (reg & TCB)

N user threads



M kernel threads