11/28 Crash Consistency Continued

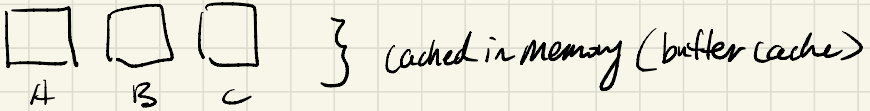
Core Problem

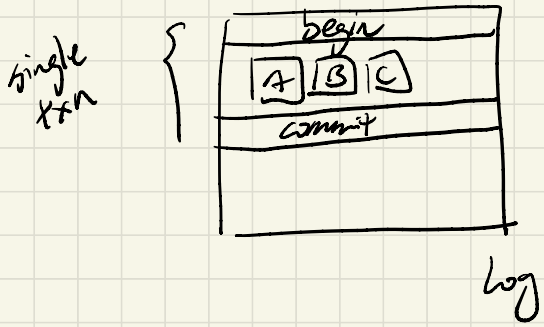
→ need atomic multiblock update to keep filesys consistent

→ device only offers atomic single block update, need SW solution!

Journaling / Logging

→ write changes to journal, commit & persist, then apply updates





Impact on Performance

① every operation will have to wait for disk I/O. (persist log & apply changes)

② writing data & metadata to disk twice (log + actual)

How to improve the performance?

① Sync Less (not on every operation)

→ every operation doesn't need to be immediately durable

→ we batch writes in memory and only persist upon an explicit fsync()

↳ how does that affect logging?

↳ have a global txn that log multiple operations, commit on fsync.

* kernel also periodically push changes to disk
* batching lets us sync less often, but also absorbs updates to the same block, meaning that we might have less data to sync

② Log Less (do we really need to log data?)

→ what happens if we just log metadata?
(filesys just needs metadata to be consistent to function, data is user's problem :))

→ Metadata journaling (gives up a bit on crash consistency but much faster)
↳ common journaling mode (ext4 default mode)
↳ steps: 1). write data to their actual location
2). then log metadata changes & persist the log

What happens if we crash here?
- we may see partial updates to data
- metadata hasn't changed, still consistent
- is directory data considered data or metadata?

fsync
→ persist a file (fd)
↳ persist log (but global txn!)
↳ only sync the file, doesn't include any dependencies
(eg. for newly created files, parent dir isn't synced)

Other ways to ensure crash consistency

→ Copy on Write

↳ journaling writes once to the journal & then applies the updates

↳ why can't we just use the journaled block as our new data?

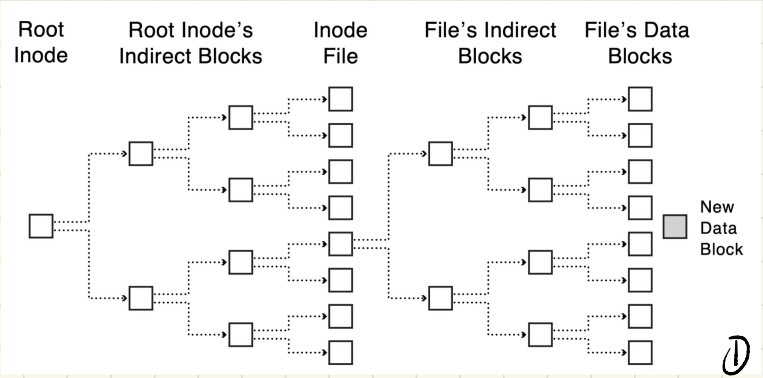
✱ b/c blocks might store pointers to other blocks, moving the blocks around will make previous pointers invalid.

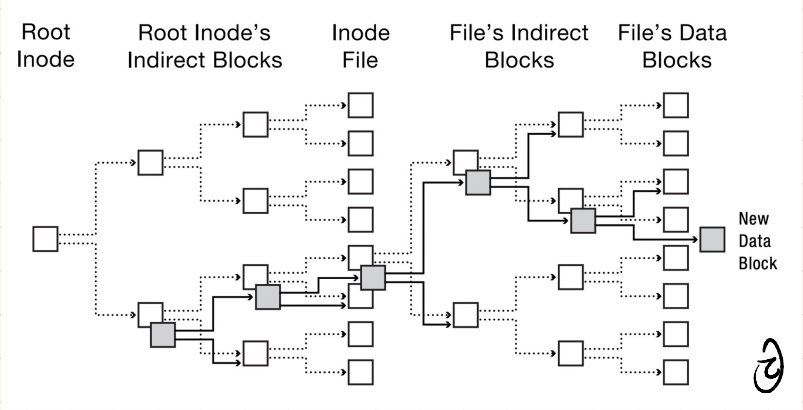
↳ how can we solve this problem?

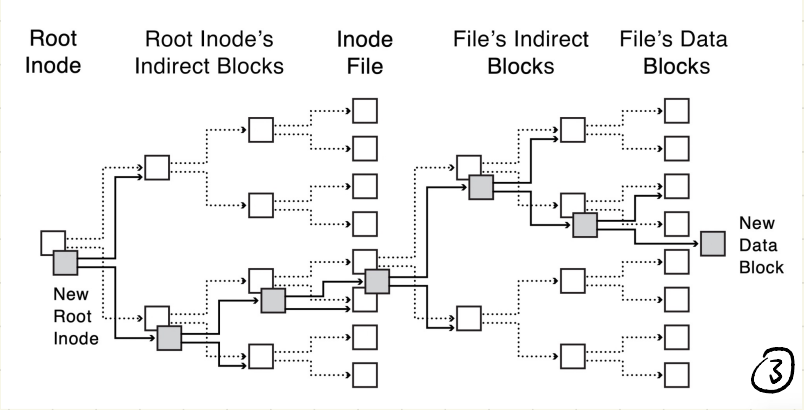
① update related blocks to point to the new location

② add a layer of indirection, block stores logical ptr to another block, track a separate logical → physical block map. (similar to FTL in SSD)

✱ requires changes to the file's layout design


Root Inode
Root Inode's Indirect Blocks
Inode File
File's Indirect Blocks
File's Data Blocks
New Data Block
1


Root Inode
Root Inode's Indirect Blocks
Inode File
File's Indirect Blocks
File's Data Blocks
New Data Block
2


Root Inode
Root Inode's Indirect Blocks
Inode File
File's Indirect Blocks
File's Data Blocks
New Root Inode
New Data Block
3