Week 14.a
CS 5600
04/10 2023

1. last time
2. fs5600 interfaces (continued)
3. crash recovery

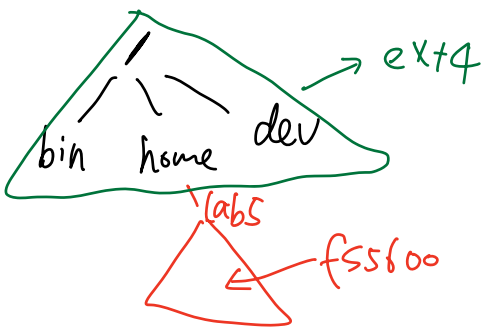
- labs regrading
- OHs

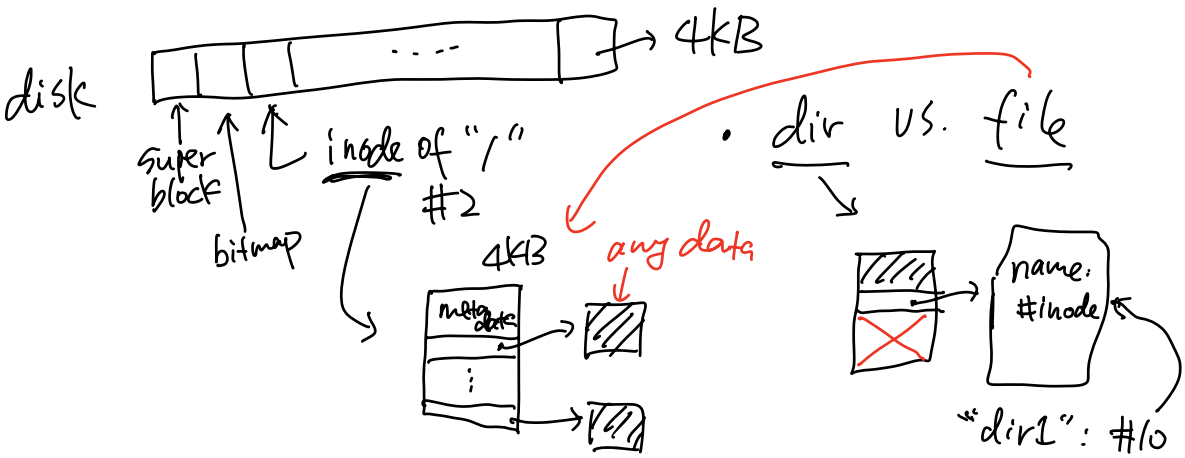
Friday 1:30PM - 5:30PM ← request

email us

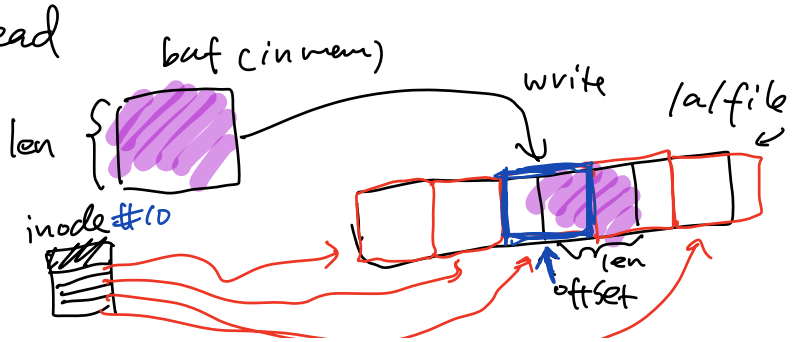
1. last time

- dir
- fs5600





- interfaces
  - read



* fs_write, "write("/a/file", buf, len, offset)" (pseudocode)

Question: how it works?

- path walk, "/a/file" → inode #10
- load block#10 (inode#10)
- offset/PAGESIZE → block#
- write len of bytes to ↓

Question: what metadata to update?

- bitmap, ptr
- mtime, ctime, size

* mkdir("/dir1/", 0644)

Question: what does "0644" mean? permission [RWX]

↳ oct. base of 8.

110 100 100

owner group world

RW- R-- R--

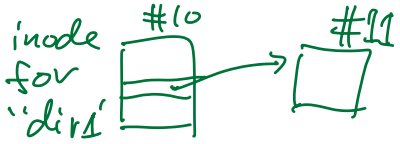
Question: how it works?

- "/" (inode#2)
- allocate a page ⇒ inode #10
- init inode #10
- load data block of "/"; add "dir1 : #10"

data block

- allocate (#11) data block for inode#10
- metadata of "/"

Question: how many "block_write()" will happen in this process?



- bitmap
- inode #10
- block# 11 (empty)
- data block of "/"
- "/" inode #2 (metadata)

- Crash recovery

problem: a lot of DSs ; Crash ! then what?

There are five writes in mkdir("/dir1", 0644):

1. block#1: bitmap, for allocating new blocks
2. block#10: create "dir1" inode
3. block#11: init data block
4. block#3: add direntry "dir1" to the parent dir ("/")
5. block#2: update metadata of the parent dir

Crash 1 / Crash 2 / Crash 3 / Crash 4

"dir1": #2

step 2: #11

- Crash 1: lost blocks.
- Crash 2: lost blocks; "dir1" ; arbitrary garbage of "dir1"
- Crash 3: unaccessible "dir1"
- Crash 4: stale metadata

Challenge: atomicity ← ACID → isolation

Solutions:

A. ad-hoc fix (fsck)

B. COW

C. Journaling

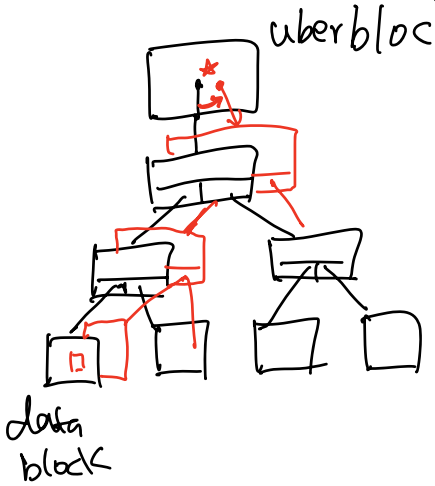
A.

- fix 1: check lost pages (Crash 1) → go from inode #2 ("/") track ALL files/dirs and find used blocks
- fix 2: "lost+found" (Crash 2)
- fix 3: (Crash 3)
- fix 4: NOTHING (Crash 4)

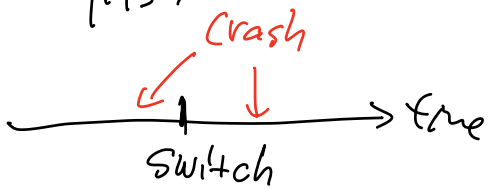
There are five writes in mkdir("/dir1", 0644):

1. block#1: bitmap, for allocating new blocks (Crash 1)
2. block#10: create "dir1" inode (Crash 2)
3. block#11: init data block (Crash 3)
4. block#3: add direntry "dir1" to the parent dir ("/") (Crash 4)
5. block#2: update metadata of the parent dir

B. COW approach (zfs btrfs)



benefits:

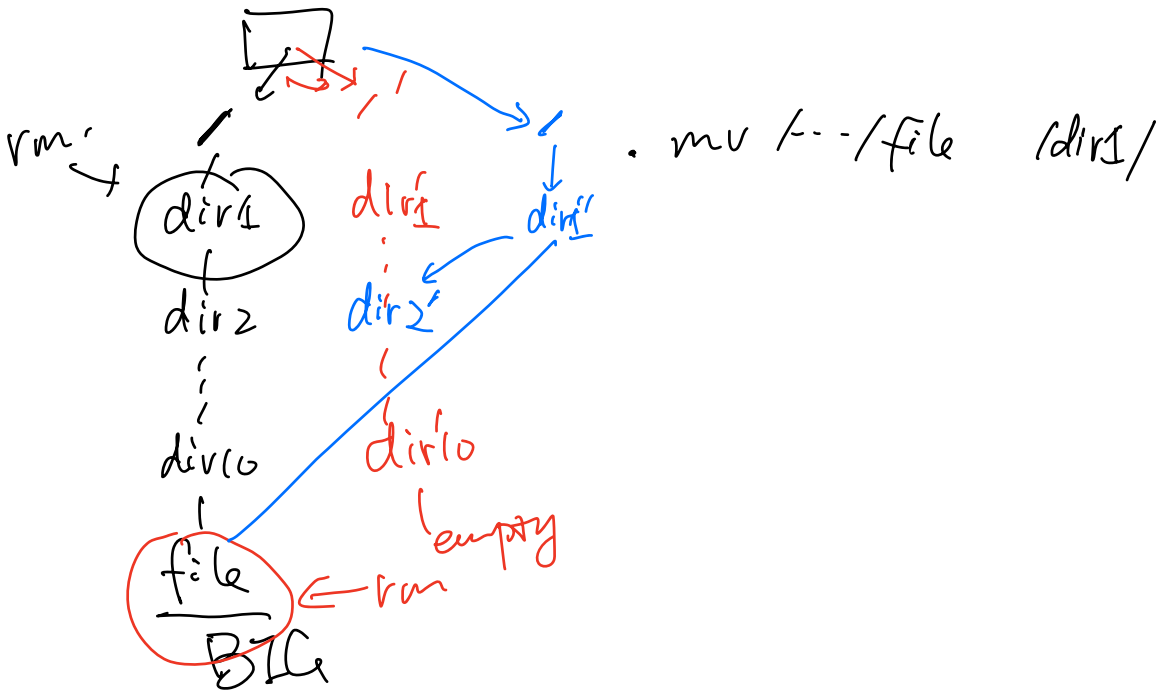


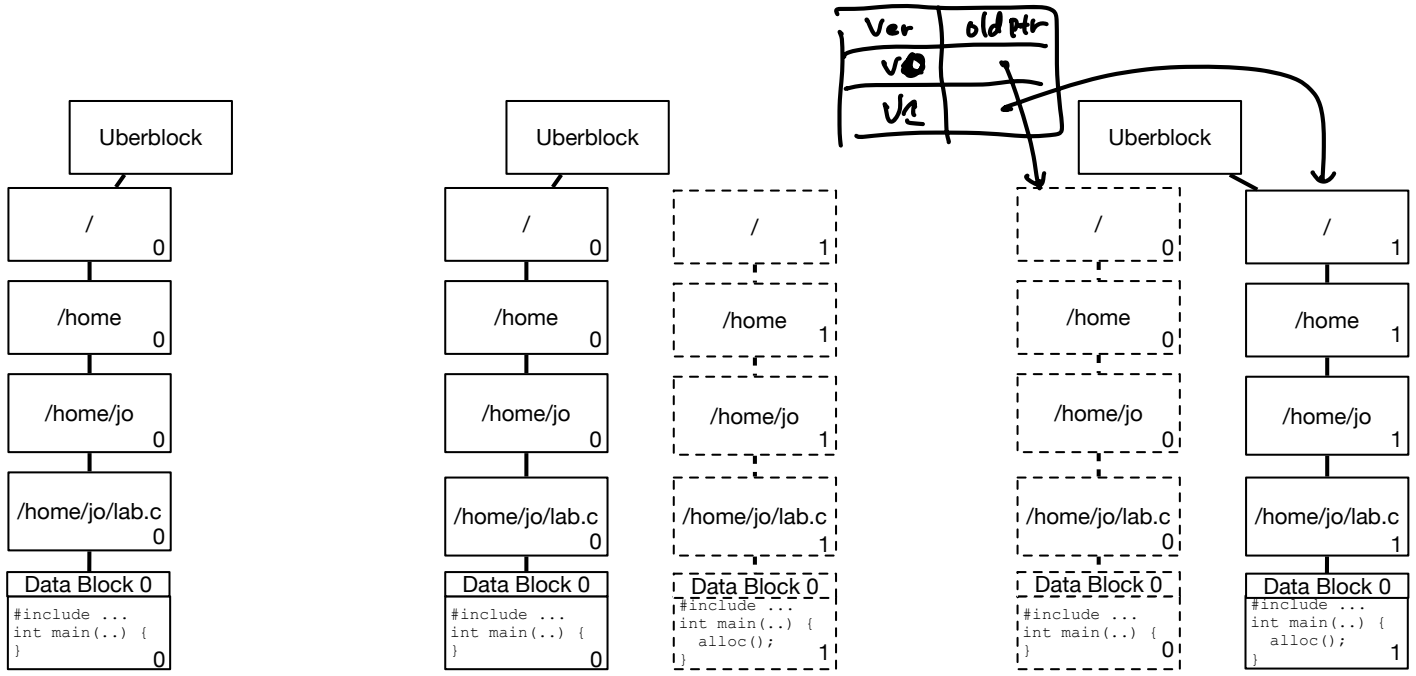
at a cost:

① space

② write amplification

Q: zfs runs out of space, is deleting files a good idea?





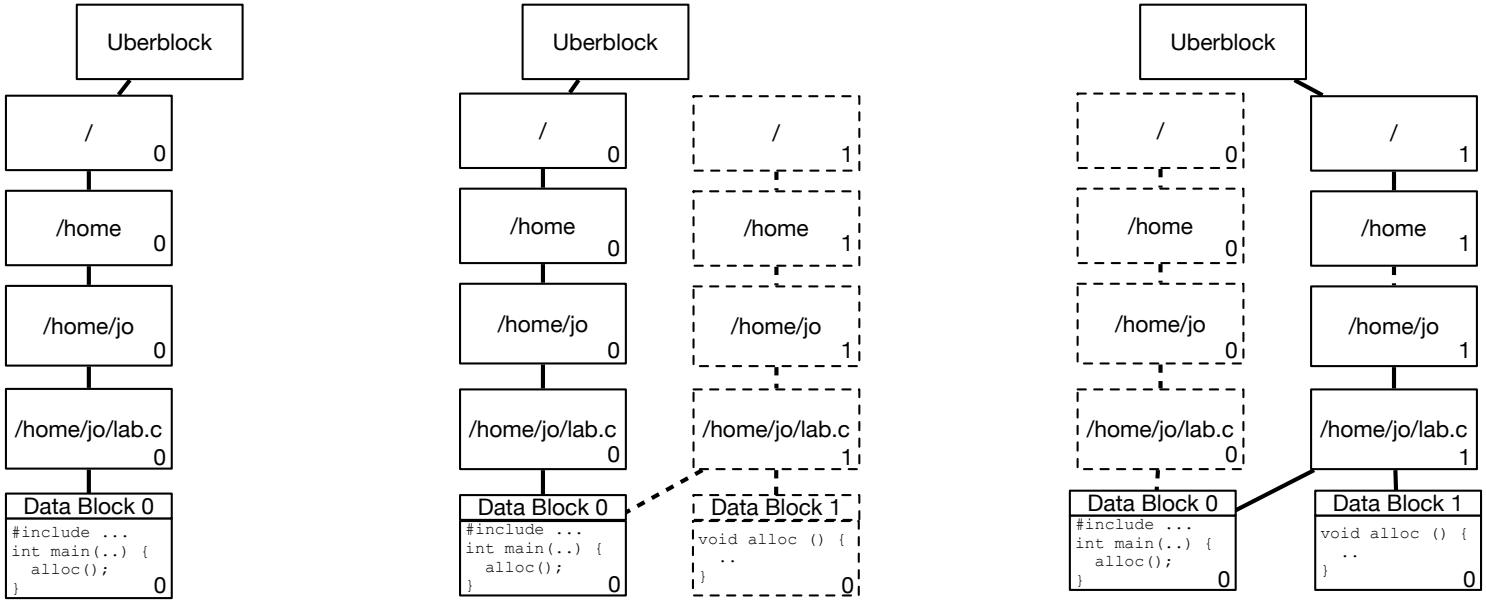
(a) Initial State

(b) System allocates and creates new versions of all modified blocks.

(c) System updates Uberblock to point to new version of blocks.

Figure 1: Copy-on-write filesystem: modifying a data block

Borrowed from NYU CS202 with minor updates:
https://cs.nyu.edu/~mwalfish/classes/21sp/lectures/handout13.pdf

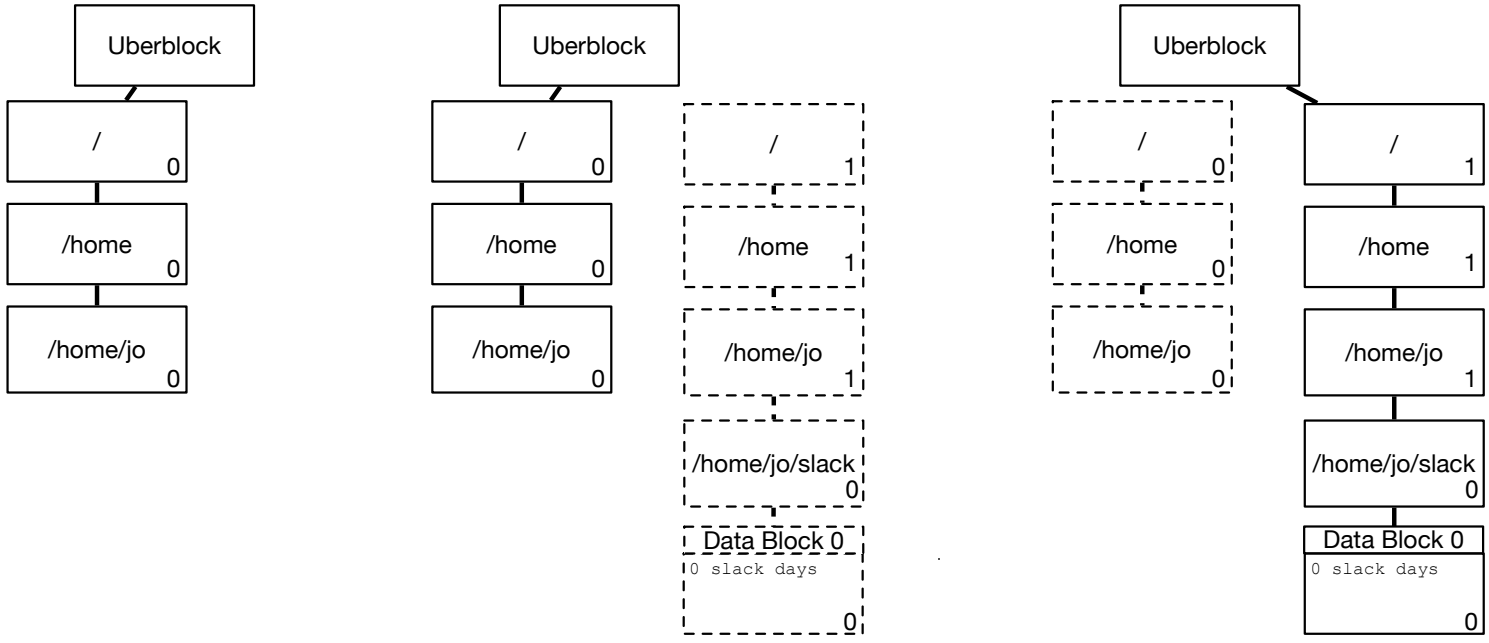


(a) Initial State

(b) System allocates and creates new versions of all modified blocks.

(c) System updates Uberblock to point to new version of blocks.

Figure 2: Copy-on-write filesystem: adding a data block



(a) Initial State

(b) System allocates and creates new versions of all modified blocks.

(c) System updates Uberblock to point to new version of blocks.

Figure 3: Copy-on-write filesystem: creating a file

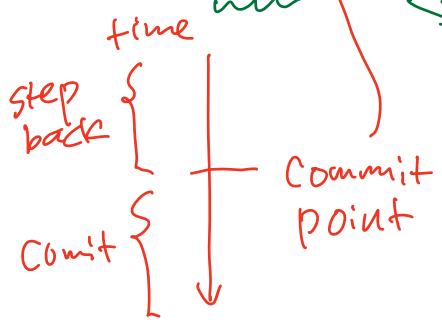
C. Journaling

"never modify the only copy"

fs op → transaction → Redo log (ext3/4)
                    → Undo log

Redo logging:

* Step 1: planning
* Step 2: begin txn ← write TxnBegin to Journal
* Step 3: journal write — 5x □ blocks
* ~~Step 4: commit txn~~ ← write TxnCommit
* Step 5: checkpointing → write to real fs.



Recovery:

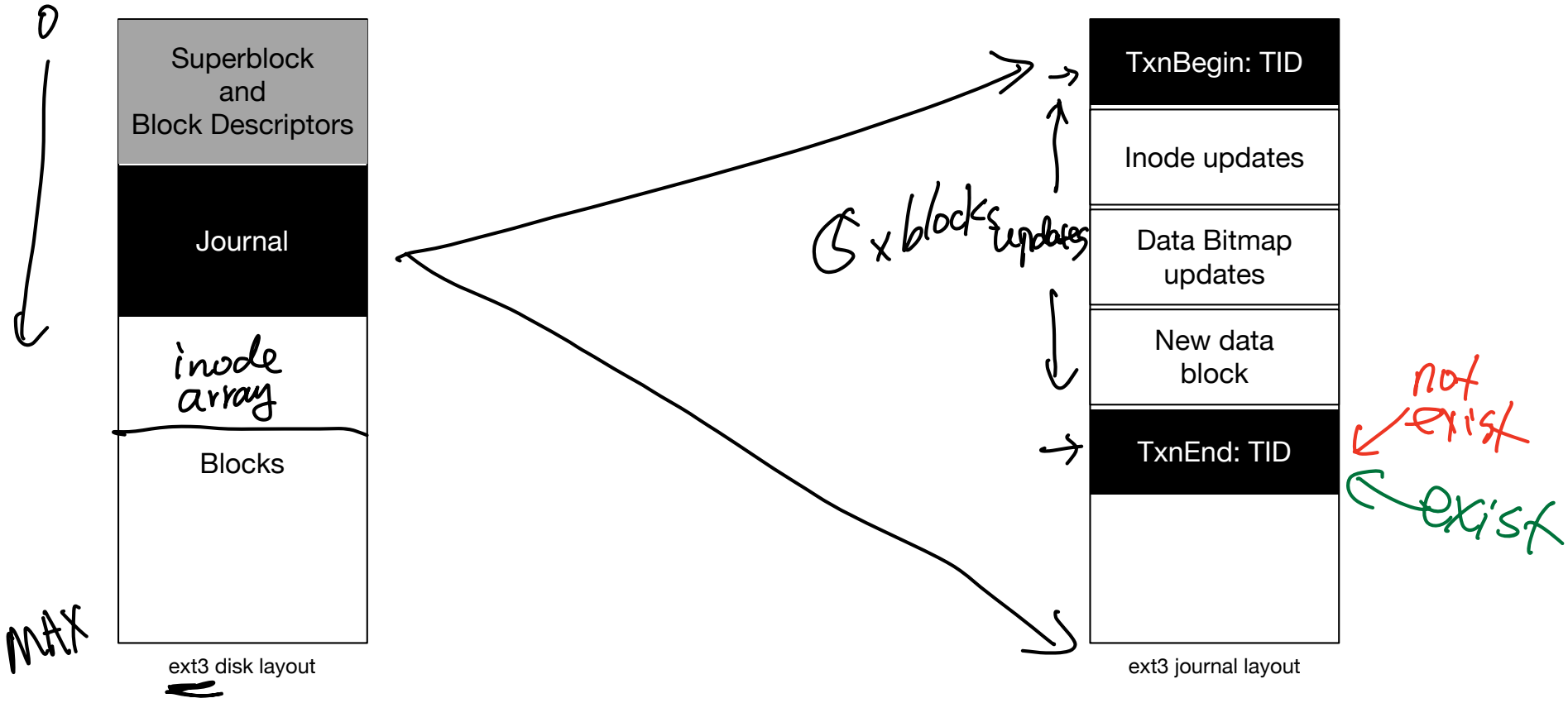


Figure 4: Redo logging in a filesystem