

☑ 1. Last time

☑ 2. Crash recovery

☑ intro

☑ ad hoc

☑ copy-on-write (cow)

☑ journaling = logging = WAL
 ↓ ↓

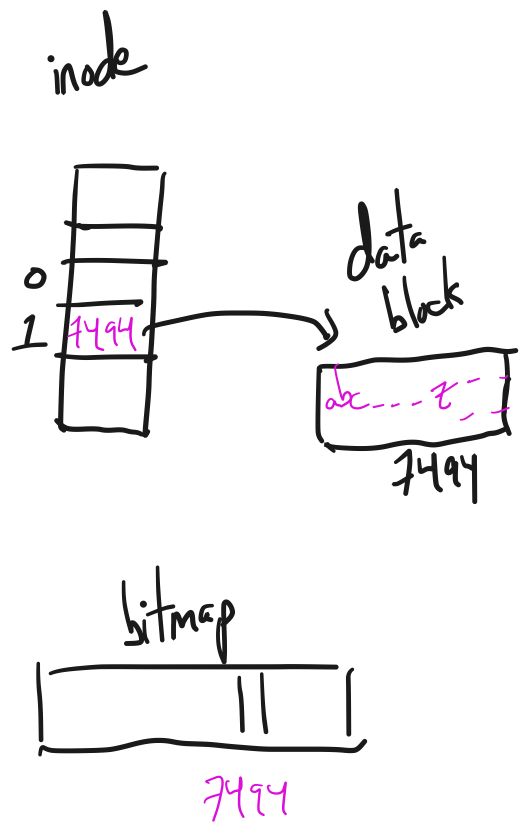
Crash recovery

- intro
- ad hoc
- copy-on-write
- journaling

Ex.

C program

```
fd = open();  
seek(fd, 4KB);  
write(fd, buf, 256);
```



- add to inode
- write data to the block

- update the bitmap

What happens if there is a crash anywhere in this list of operations?

Ad hoc

Goal: metadata consistency, not data consistency

Approach: send FS updates to the disk in such a way that if there is a crash, fsck can clean up inconsistencies.

Example: file create/write!

- first write data to file blocks on disk

crash?

- then update/write inode

crash?

- then mark inode "allocated" in inode bitmap

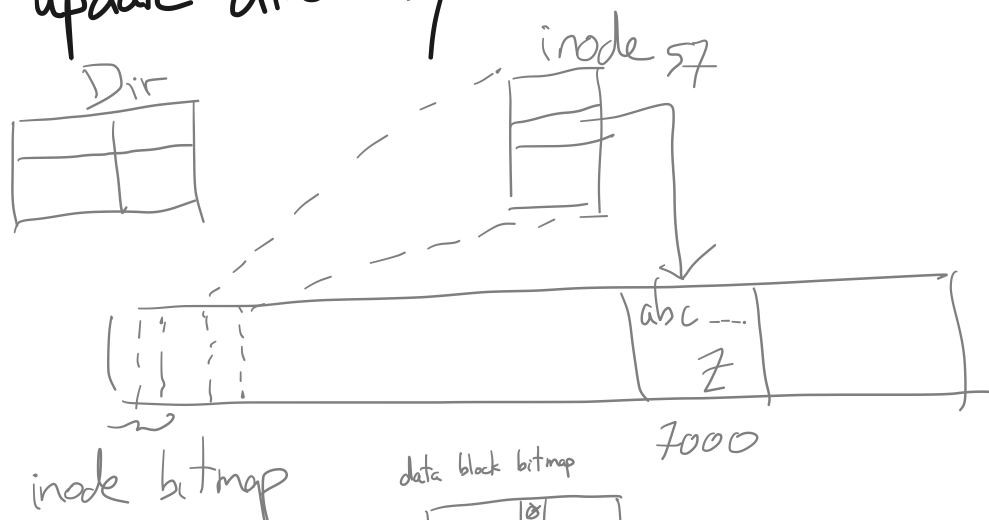
crash?

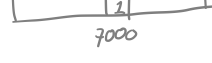
- then mark data blocks "allocated" in bitmap

crash?

- then update directory <name, i#>

crash?





Copy on write

ZFS, btrfs, APFS

Goal: metadata and data consistency

Spent space

Approach: never modify a block, always make a copy

Exception: root block, Uberblock

Journaling

Golden rule of crash atomicity:

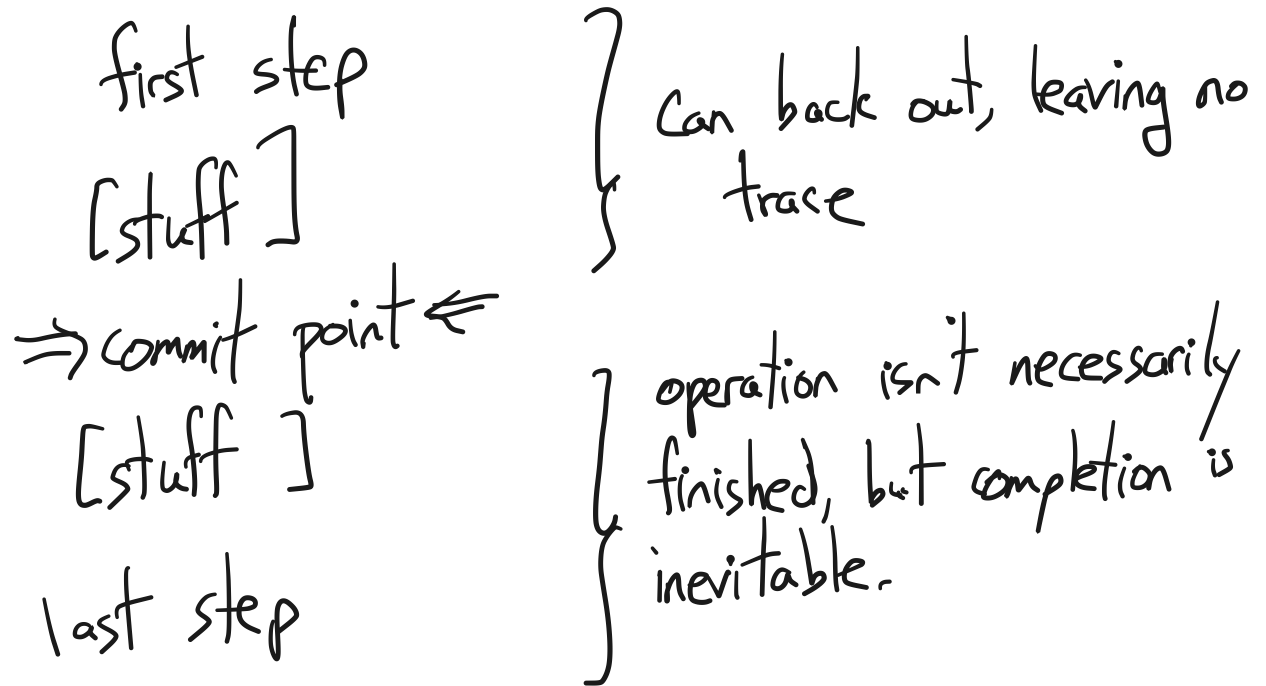
"Never modify the only copy."

Borrow an idea: transactions, from DBs

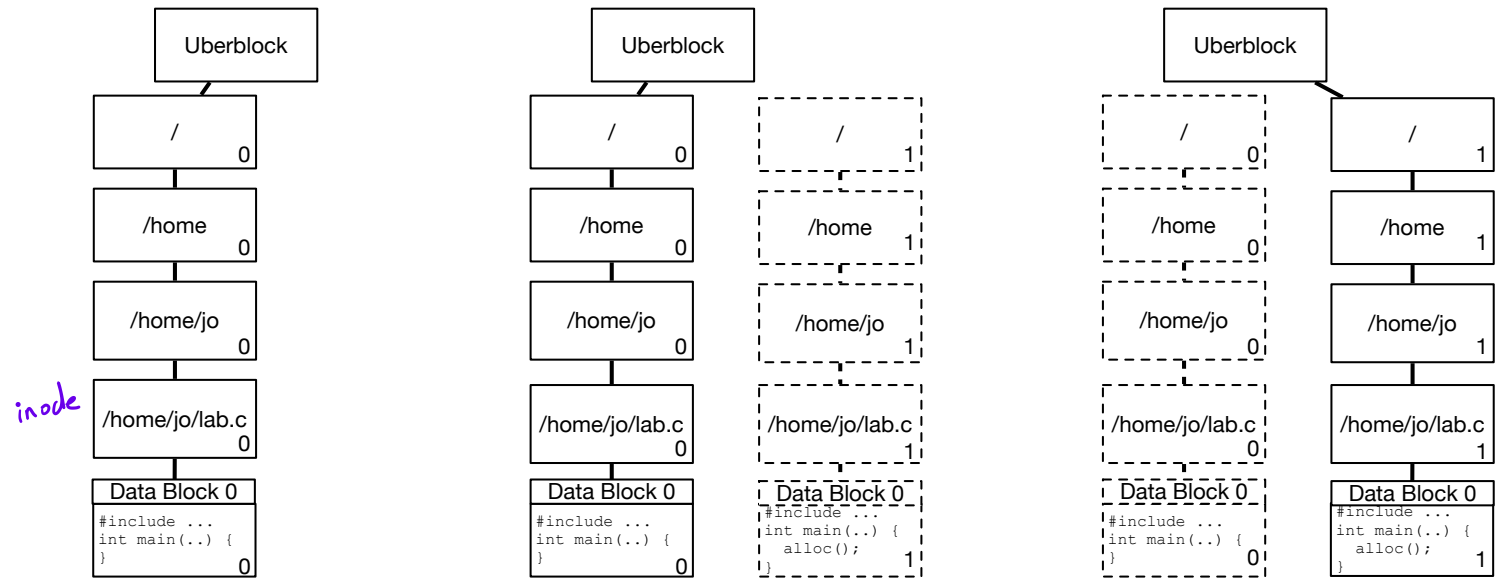
here, an op is: create a file, delete a file, ...

sub.op: a component of the op

concept: commit point:



- WAL == redo logging
- undo logging
- redo + undo logging

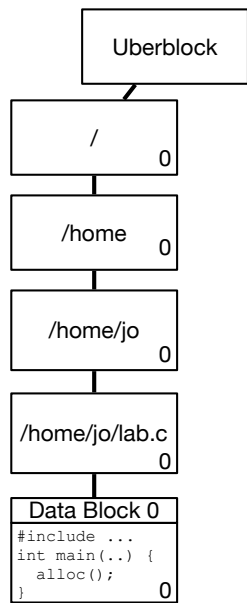


(a) Initial State

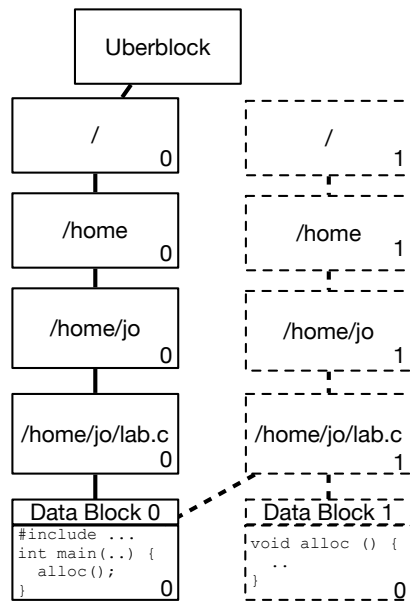
(b) System allocates and creates new versions of all modified blocks.

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

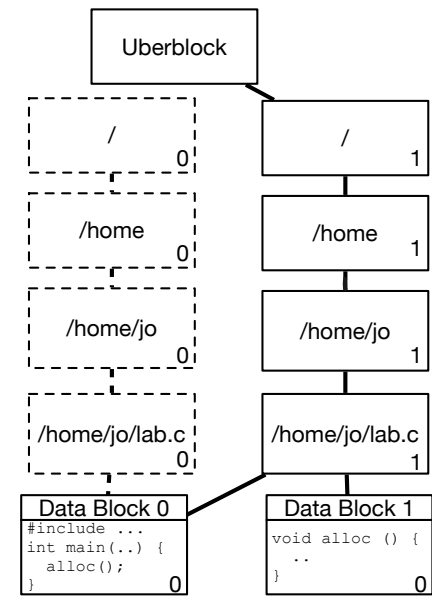
Figure 1: Copy-on-write filesystem: modifying a data block



(a) Initial State

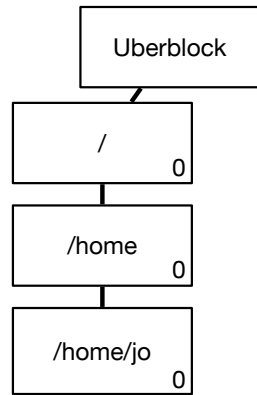


(b) System allocates and creates new versions of all modified blocks.

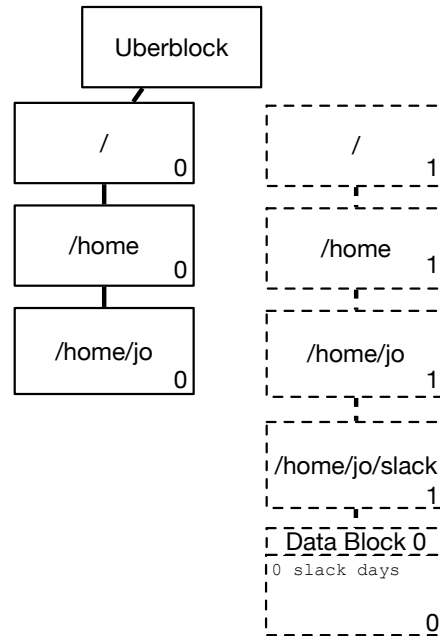


(c) System updates Uberblock to point to new version of blocks.

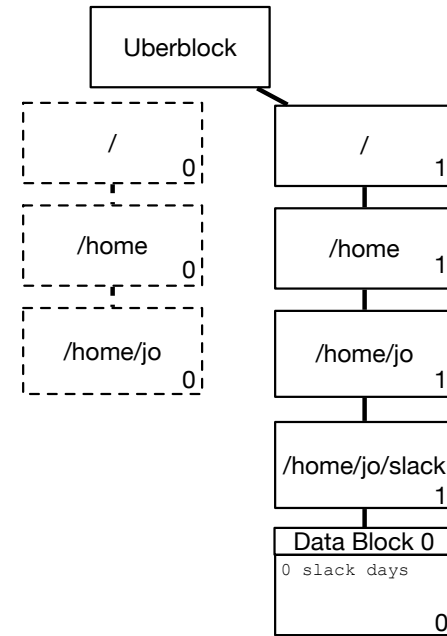
Figure 2: Copy-on-write filesystem: adding a data block



(a) Initial State



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



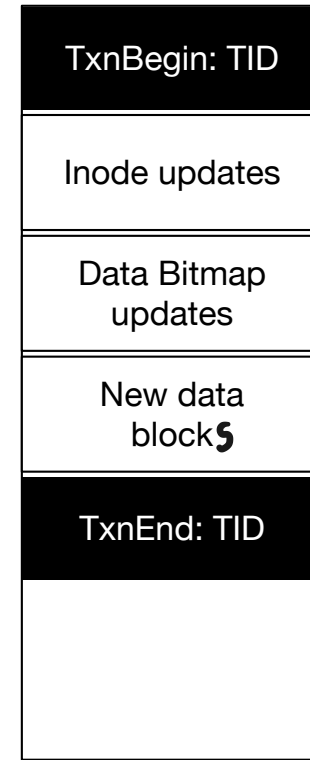
(c) System updates Uberblock to point to new version of blocks.

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

vocab: checkpointing



ext3 disk layout



ext3 journal layout

online:

op():

1. what (in the FS) would change?
 2. where in the log can we write?
 3. logs all intended changes
 4. writes TxnEnd
 5. FS asynchronously updates the data structures
- checkpointing

Figure 4: Redo logging in a filesystem

recovery:

go through log
for all Tx where TxnEnd matches TxnBegin:
apply the updates to the FS data structures

Undo logging

- ① What would change
- ② Where in the log we can write it
- ③ log changes || apply them to the on disk data structures

(in parallel)

(of course logging a single entry before the modifications)

④ write Tx End

Recovery

- ① go through log
- ② look for all transactions without Tx End record
go backwards
undo all entries

Redo + undo logging