- ☑ 1. Last time
- ☐ 2. Crash recovery
  - ☐ intro
  - ☐ ad hoc
  - ☐ copy-on-write (COW)
  - ☐ journaling

# Crash recovery

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

Ex.

C program

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

inode



0
1  7494  →  data block

abcdef...
7494

bitmap

| | | || | |

- 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

  Spend space

  Approach: <u>never</u> modify a block, always make a copy

  Exception: root block, Uberblock

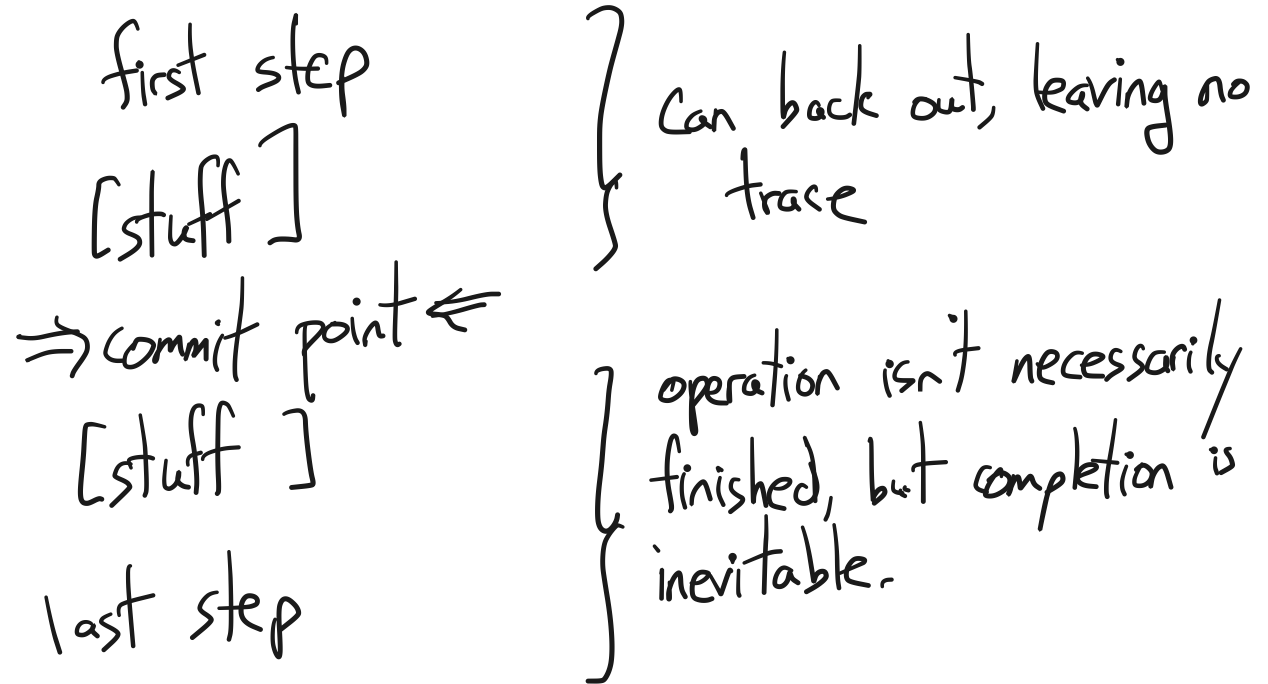---

Journaling

                                        Saltzer-Kaashoek

  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:

first step

[stuff]

}  can back out, leaving no trace

⇒ commit point ⇐

[stuff]

last step

} operation isn't necessarily finished, but completion is inevitable.

☐ WAL == redo logging

☐ undo logging

☐ redo + undo logging

# CS202 Handout 12

```
┌─────────────────┐          ┌─────────────────┐          ┌─────────────────┐
│   Uberblock     │          │   Uberblock     │          │   Uberblock     │
└─────────────────┘          └─────────────────┘          └─────────────────┘
```
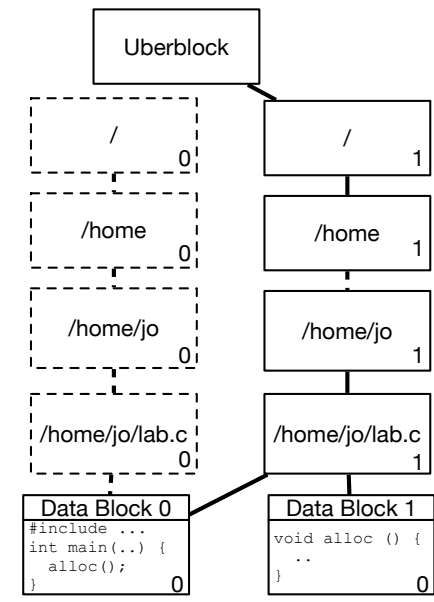
| | | |
|---|---|---|
| **(a) Initial State** | **(b) System allocates and creates new versions of all modified blocks.** | **(c) System updates Uberblock to point to new version of blocks.** |
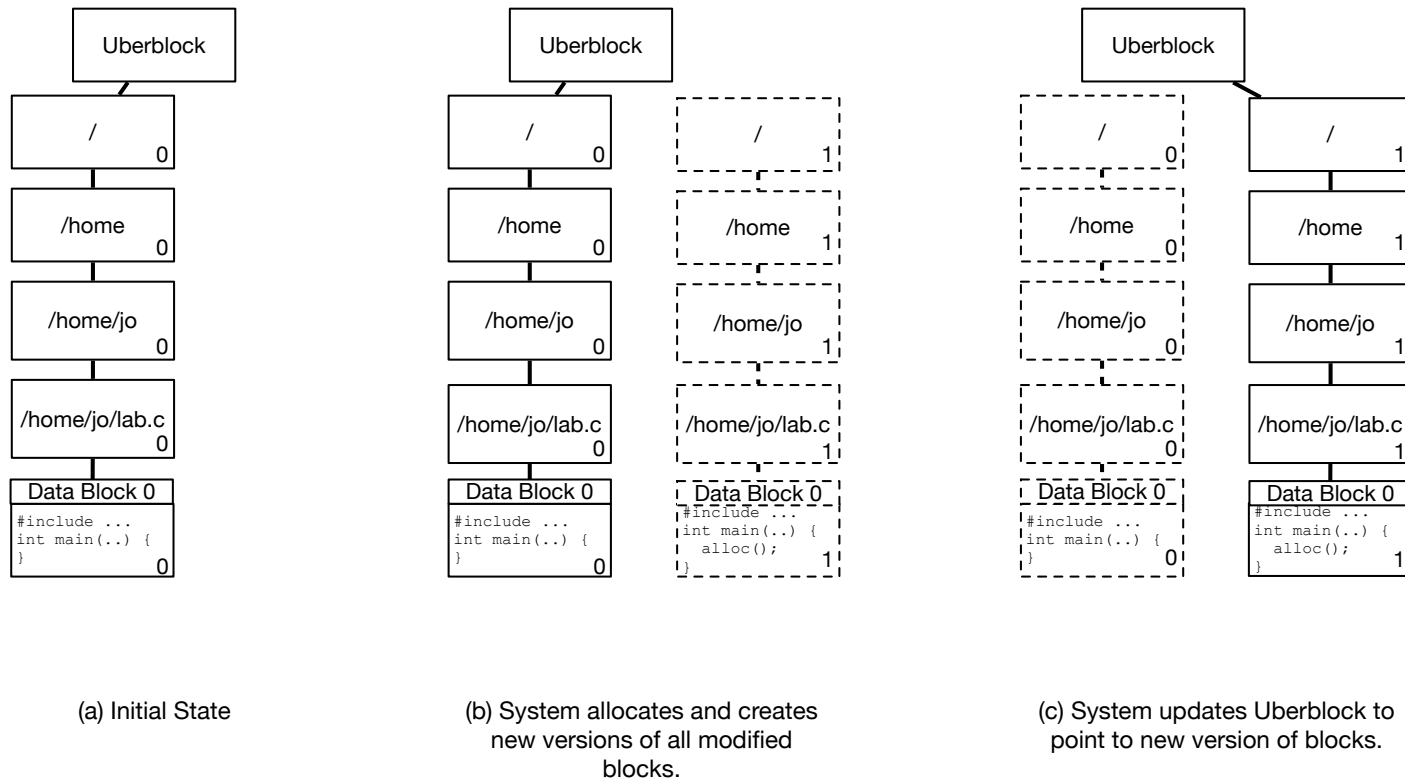
Figure 1: Copy-on-write filesystem: modifying a data block
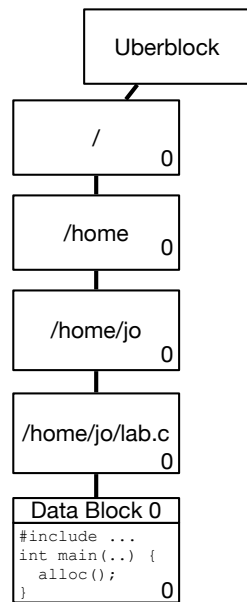
1

Figure 2: Copy-on-write filesystem: adding a data block
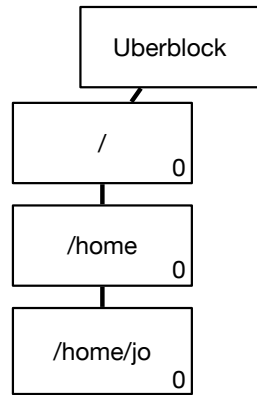
(a) Initial State

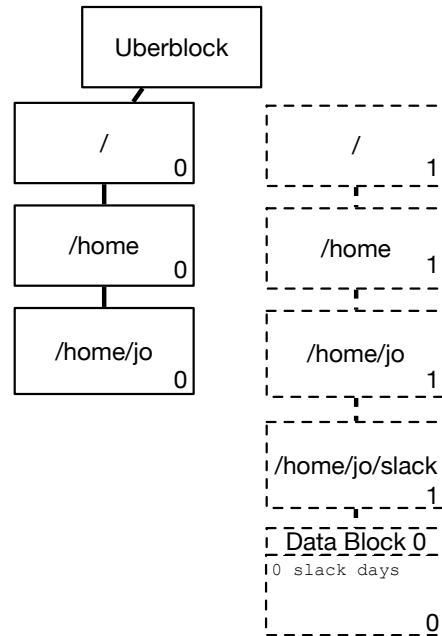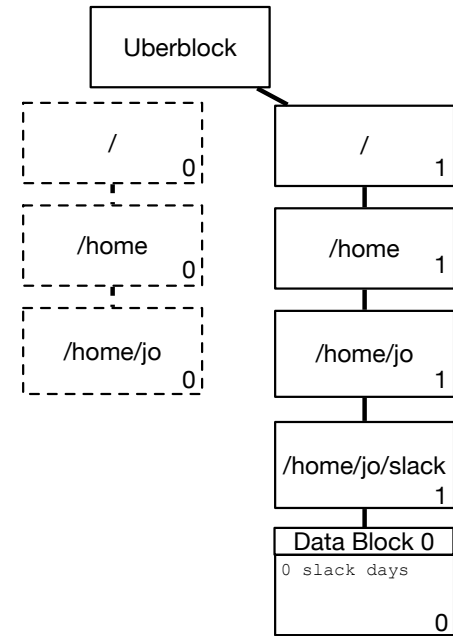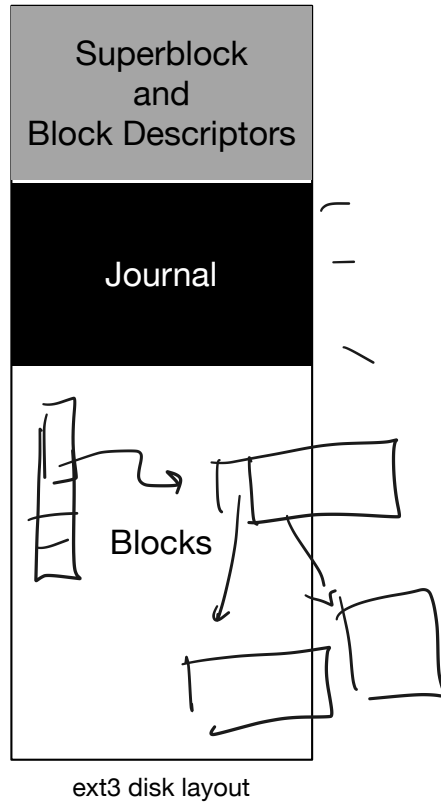(b) System allocates and creates
new versions of all modified
blocks.

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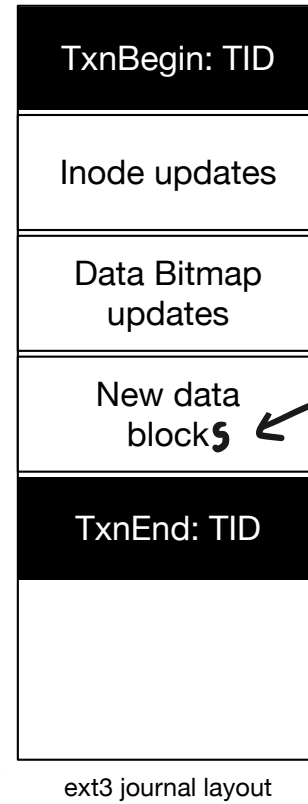point to new version of blocks.

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

Log == journal

| Superblock and Block Descriptors |
|---|
| Journal |
| Blocks |

ext3 disk layout

| TxnBegin: TID |
|---|
| Inode updates |
| Data Bitmap updates |
| New data blocks |
| TxnEnd: TID |
| |

note type on photocopied handout

ext3 journal layout

Figure 4: Redo logging in a filesystem

online:

op():

1. what would change?
2. where in the log can we write?
3. logs all intended changes
4. writes TxEnd
5. FS asynchronously update the data structures

recovery:
go thru log
for all tx where TxEnd matches TxBegin:
apply the updates to the data structures

4