(FS CONSISTENCY)
JOURNALING

- → what is crash?
- → why a problem? (orders)
- → approaches
  - → fsck
  - → journaling

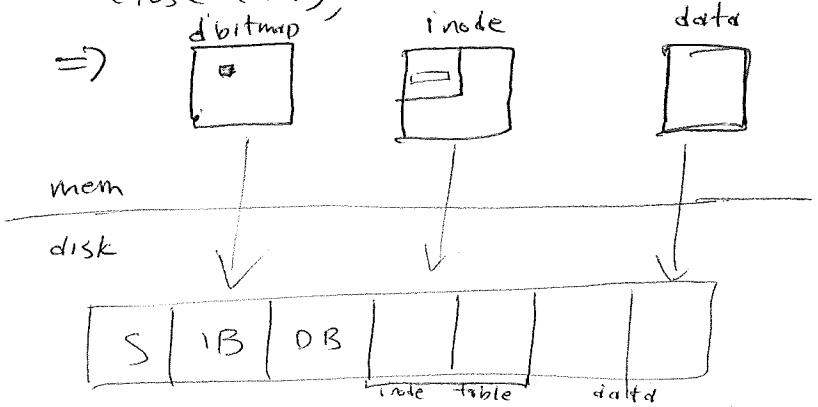
Problem: Crash Consistency

What is a crash? (unexpected interruption of running OS)
- → power loss
- → kernel panic, reboot
- → user hard reset

Why important for FS?
- → could be "in the middle" of an update to persistent state (not a big deal for process, VM, etc.)

Example: File Create
updates to parent dir inode, data
inode bitmap, inode itself

Example: File Append
```
fd = open("file", O_WRONLY);
lseek(fd, 0, SEEK_END);
write(fd, buf, size);
close(fd);
```



Possible write orders? (really just six possibilities) [1 blocks, 2 blocks]

① D	③ D	⑤ I	⑦ I	⑨ DB	⑪ DB	← crash?
② I	④ DB	⑥ D	⑧ DB	⑩ I	⑫ D	← crash?
DB	I	DB	D	D	I	

no problem: 1, 3

inode/d bitmap inconsistent: 2, 4, 5, 6*, 7, 9, 11, 12

consistent but garbage: 8, 10

* could just fix bitmap

FS "consistent" iff all on-disk info is in agreement on state of FS

Solution #1: (lazy) file system checker (fsck)

⇒ Scan entire disk (inodes, bitmaps, directories, indirect blocks, etc.)

⇒ Find + fix inconsistencies

note: not done on clean unmount

Solution #2: (eager) Journaling (WAL)

Basic idea: new on-disk structure called journal

① Use it to write down what you're about to do

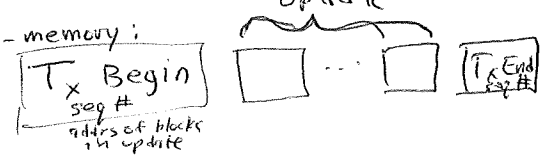
② Then, do it.

if you crash while doing ②, it's OK, just use journal to figure out/fix (recovery)

GOAL: make multi-block update ATOMIC (all or nothing)

Issue 1: Location of journal
usually near begin of disk
OR, could be on own device

Issue 2: Basics of a Transaction

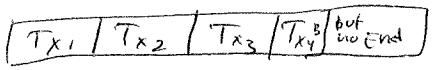


Protocol #1:
1) write [T_B, contents, T_E] to log (wait)
2) checkpoint: bring FS up to date by writing contents to final locations

Problem: writes may complete out of order (disk sched)

Protocol #2:
1) log write (T_B, contents); wait
2) log commit (T_E); wait
3) checkpoint

Recovery: scan log, find all valid Tx's, replay
(if 2 has not completed, not valid Tx!)



Problem: each data block written twice

⇒ solution: meta-data only journaling