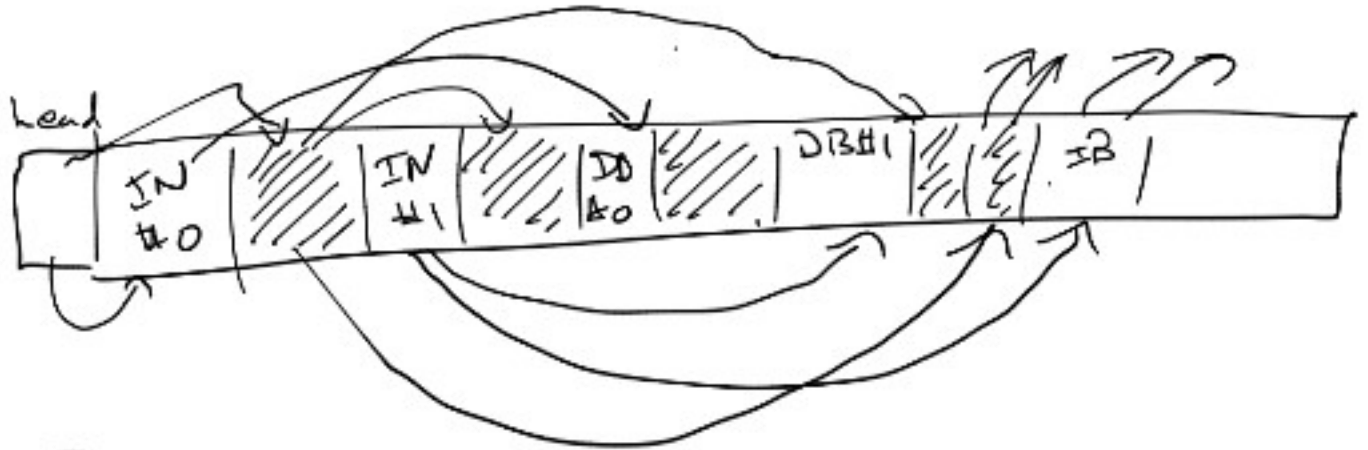
Lecture 19: recovery

- Fsck
- Journaling
- Log-structured FS
- Quiz (really!)

Two data structures:

- inodes & files
- free list



Invariants:

- every block is part of a file (no orphans)
  block with no references to it: orphan
- no block is both free & in a file (lead to same block being in two files: stomp on each other)

Can lose power / shut down unexpectedly at any time: what state is disk in.

allocate():

find a free block
mark it not free
shut down! → add it to inode of new file.

How to remain consistent?

- prevent power outages.
  universal power supply (battery)
  raise interrupt when power is about to be out.
  could respond by not starting any new writes, just wait for existing to complete

FS recovery

- when we boot, check to see if the disk is in a consistent state, if not, fix it. (fsck : filesystem check)
  - traverse entire FS. (both files & free list) checking invariants.
    - if a block is in neither: add to free list
    - if in both, remove from free list.
  - read entire disk! (only if shut down unexpectedly).

Journaling: improve (drastically) speed of fsck

- before writing to FS (structure)
  (allocation / dealloc)
  make a note indicating what you're about to do.
  then do the operation
  later: mark it as done.

```
alloc():
    find a free block b
    write to journal: "I'm about to allocate block b
        as inode for file /home/mdg/new.txt"
    remove b from free list   <- sync()
    add b to filesystem       <- sync()
    mark journal entry complete.
```

crash:

OK →

Journal indicates need to ensure b not in FL, b is in inode structure.

OK. →

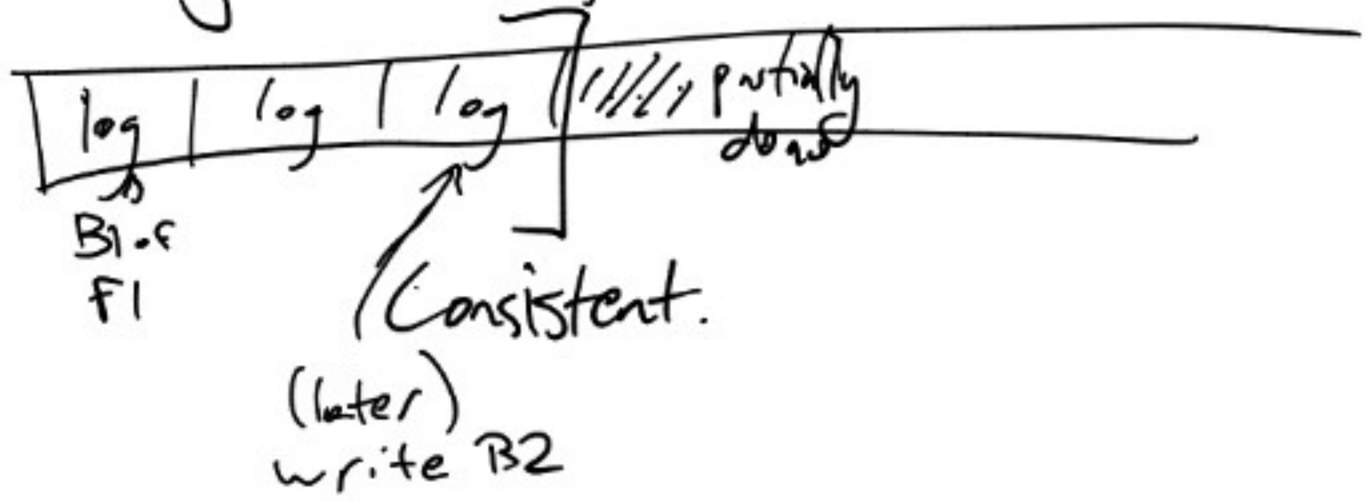
- Need journal write to complete before start FS writes.
- sync(): wait for indicated writes to complete.

Log structured Filesystem

idea: everything goes in journal (journal is filesystem).

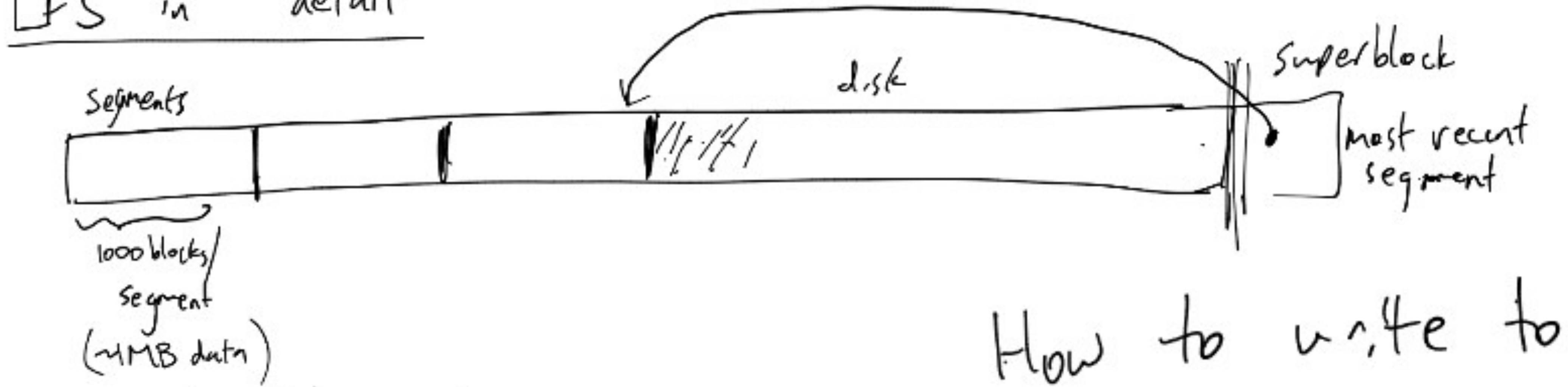
advantage: only need to append to end of FS, never need to go back to do new writes.

- always a prefix that is consistent.



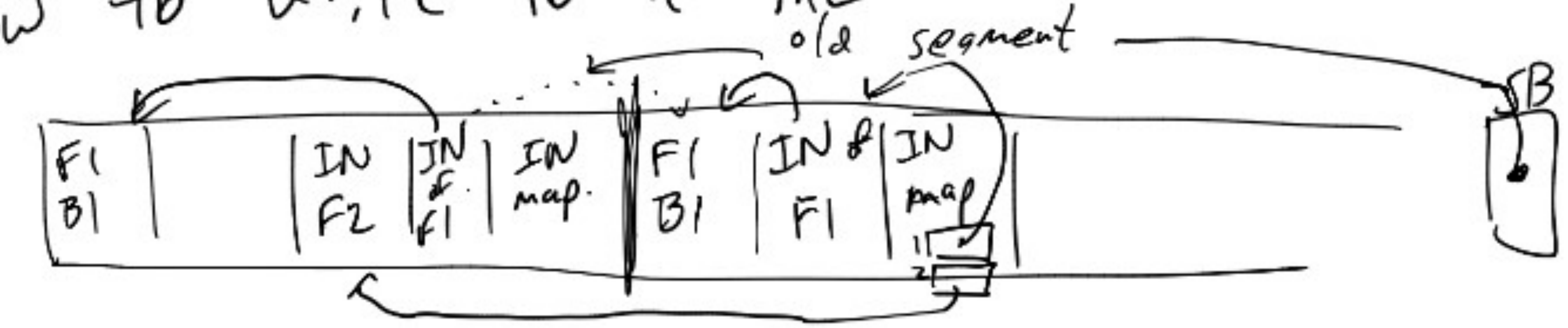
- If we assume everything cached in memory never read from disk, only write new data.
- all writes sequential: fast.

LFS in detail



- each (complete) segment represents a consistent state.
- need to know what is the most recent complete Segment.
  (pointer in 'superblock', only updated after seg. completely written).

How to write to a file?



want to overwrite F1, B1,
can't go back & change
→ create new copy in current segment, modify that. (in memory)
→ also need to update INode (and indirect blocks, ...)
→ Inode map: a table mapping inode #s to disk addresses.

(updates in an in-memory buffer (cached copy of new segment))

Checkpoint:
Write buffer to disk in empty segment,
update super block

Lose everything saved between checkpoints on outage.