Lab 4 Details

Quick notes

- Lab 4 Design Doc due Friday (2/24) (tomorrow)!
- Lab 4 due 3/10
- Hard deadline for all labs: 3/17
- Second practice quiz posted :)

Part A: Writable FileSys

Dinode Data Layout

- Need to change the data layout in stuct dinode to support multiple extents
- Once it's changed, update readi to work with the new layout
  - also need to update mkfs.c as it writes some inodes
  - mkfs.c sets up the initial file system image, including some executable file (labxtest, ls, exec, echo, cat..) and their inodes
- Would be good to have a function that takes an offset and finds the block #
  - can use in both readi and writei to find which block to read/write to

icache

- Disk operation are slow
  - Thus, we have a cache of inodes
- icache.inodefile is initialized at system startup
- icache.inode is an in-memory cache of most-recently-used inodes
  - They are not in order! Use iget to search the cache and irelease to release the cache!
- Difference between inode and dinode
  - In memory vs on disk
  - Need to synchronize them: read_dinode (provided, used in locki) move data from disk to memory. write_dinode move data from memory to disk (not provided)

```
struct {
 struct spinlock lock;
 struct inode inode[NINODE];
 struct inode inodefile;
} icache;
```

read_dinode

```
// Reads the dinode with the passed inum from the inode file.
// Threadsafe, will acquire sleeplock on inodefile inode if not held.
void read_dinode(uint inum, struct dinode *dip) {
    int holding_inodefile_lock = holdingsleep(&icache.inodefile.lock);
    if (!holding_inodefile_lock)
        locki(&icache.inodefile);

    readi(&icache.inodefile, (char *)dip, INODEOFF(inum), sizeof(*dip));

    if (!holding_inodefile_lock)
        unlocki(&icache.inodefile);

}
```

```
// offset of inode in inodefile
#define INODEOFF(inum) ((inum) * sizeof(struct dinode))
```

- What does the function do?
  - Reads in struct dinode at index `inum` from inodefile
- Having a similar write_dinode() can be helpful (not provided in starter code)
  - When should we write dinode?

Helpful Functions

- mkfs.c:

```
nblocks = dinode.size/BSIZE + (dinode.size % BSIZE == 0 ? 0 : 1);
```

  ^ existing code to compute how many blocks are needed given a file size
  (feel free to do your own math, but just know that this math is correct)

- fs.c:
  - *read_dinode*: returns the dinode for a given inumber
  - *iget*: returns the in memory inode for a given inumber, inode may not have cached information from dinode
  - *locki*: locks the inode and guarantees that inode has info from dinode
  - *dirlookup*: finds the offset of a directory entry with matching name
    - skips over dirent with inum of 0

Tips

- write & append:
  - append = writing past end of file
  - if you are just overwriting an existing block of data, do you need to update its dinode?
  - what if you are appending more data to the file, do you need to update its dinode?
- balloc, bfree, bmark:
  - balloc and bfree only updates cached bitmap sectors in memory
  - this is done through setting the bp->flag dirty in bmark
  - if you want to write bitmap sector back to disk, you need to call bwrite yourself on the bp (handle to the changed bitmap sector)

Part C: Crash Safety

Journaling

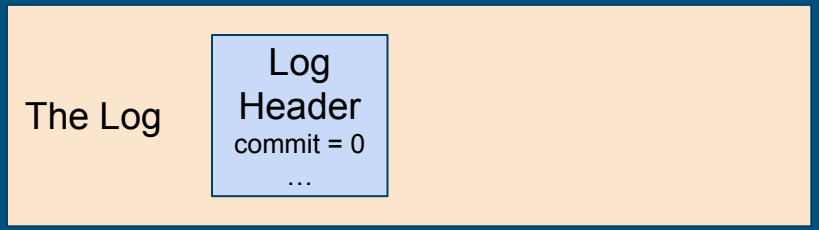
For any operation which must write multiple disk blocks atomically…

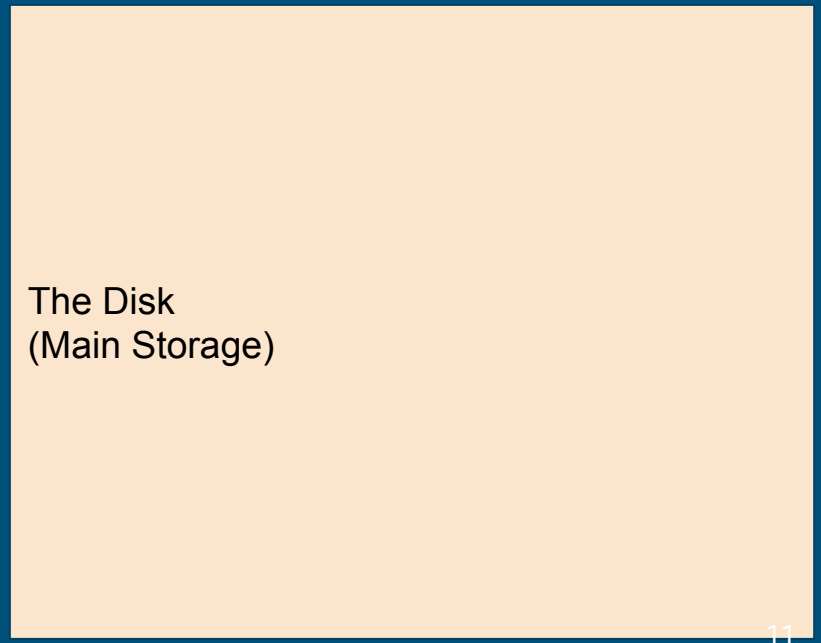
1) Write new blocks into the log, rather than target place. Track what target is.
2) Once all blocks are in the log, mark the log as “committed”
3) Copy data from the log to where they should be
4) Clear the commit flag

On system boot, check the log. If not committed, do nothing. If so, redo the copy (copy is idempotent)

Step 1: “log_begin()”

Make sure the log is cleared

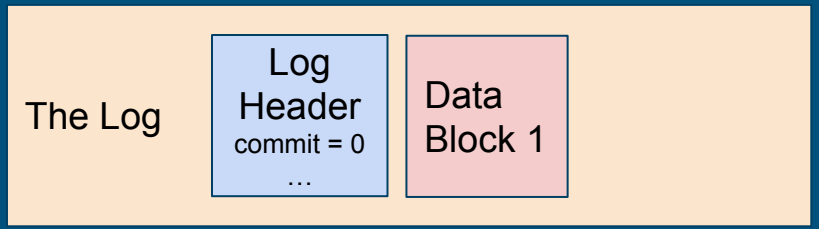




Step 2: “bwrite(data block 1)”

Write into the log, rather than the place in the inode/extents region we want it to go

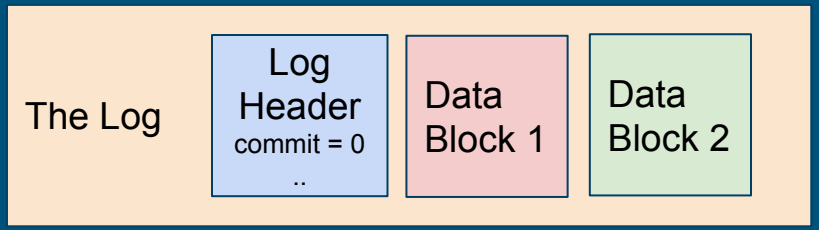
Also need to track the actual location of the data block so you know where to write logged blocks to on recovery!



The Disk
(Main Storage)

Step 3: “bwrite(data block 2)”

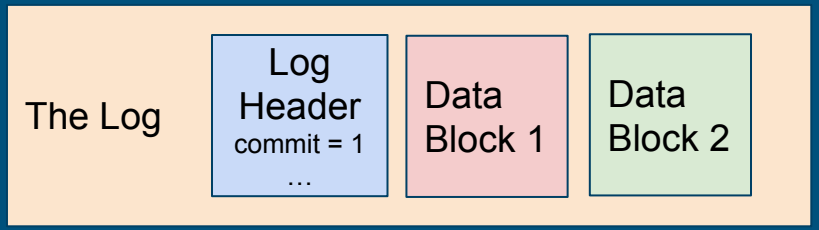
Write into the log, rather than the place in the inode/extents region we want it to go



The Disk
(Main Storage)

Step 4: “log_commit()” [1]

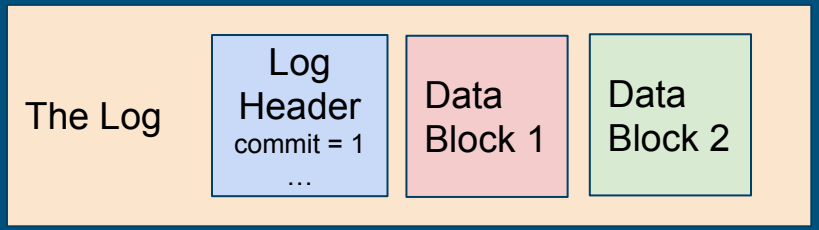
Mark the log as “committed”

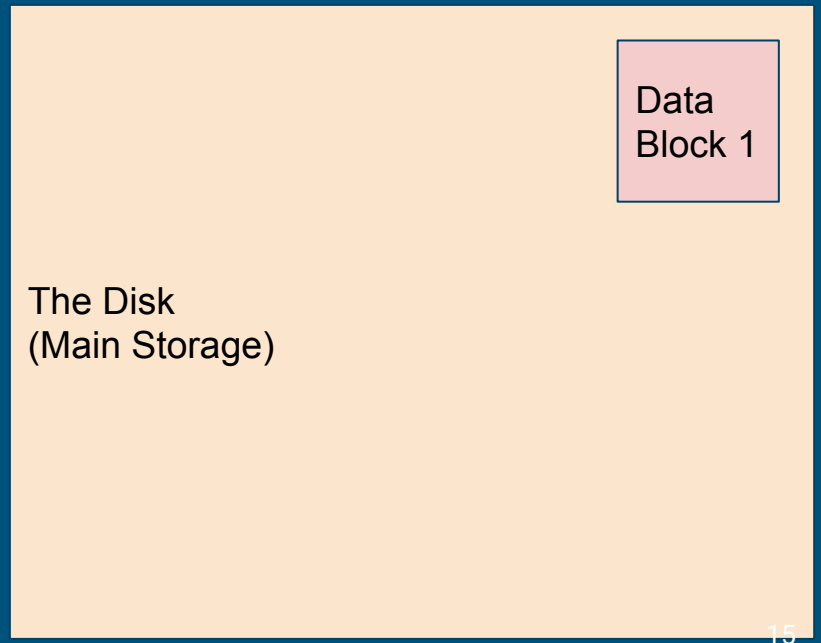


The Disk
(Main Storage)

Step 5: “log_commit()” [2]

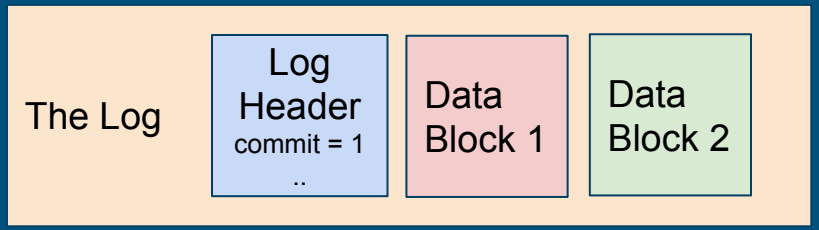
Copy the first block from log onto disk

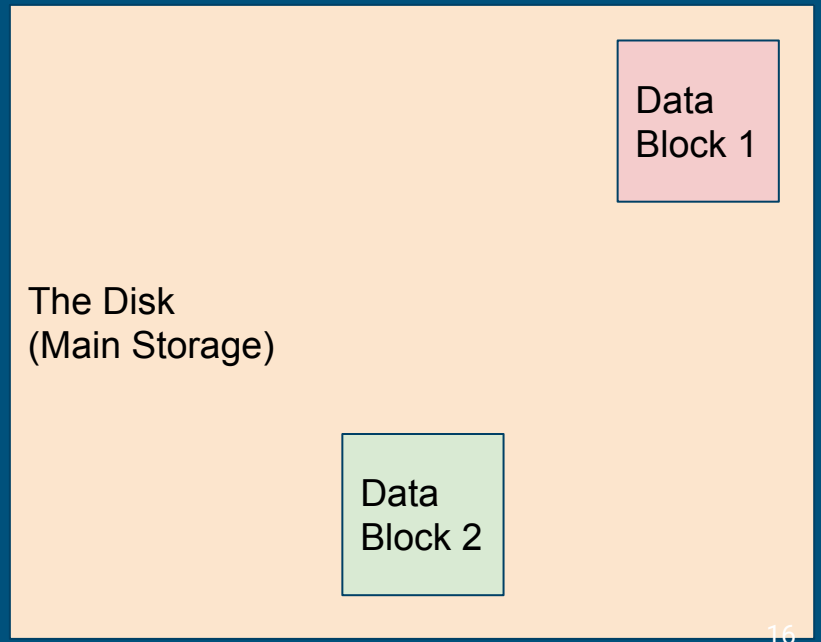




Step 6: “log_commit()” [3]

Copy the second block from log onto disk

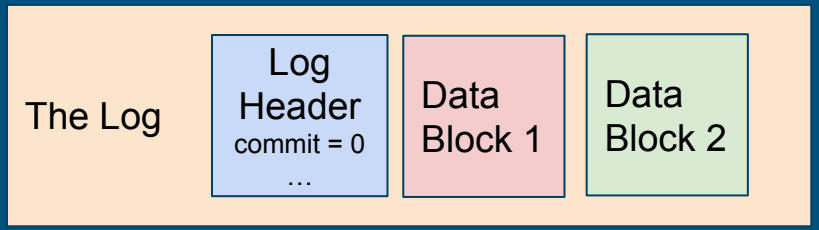


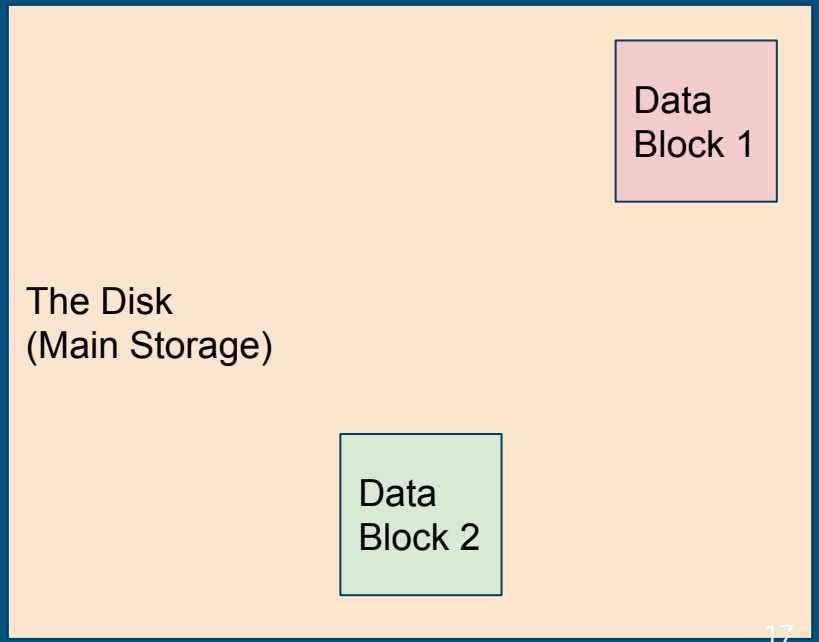


Done!

We have both data blocks 1 and 2 on disk - everything was successful.

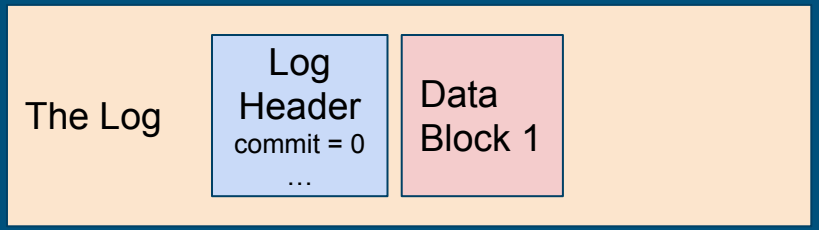
For efficiency, we can zero out the commit flag so the system doesn't try to redo this

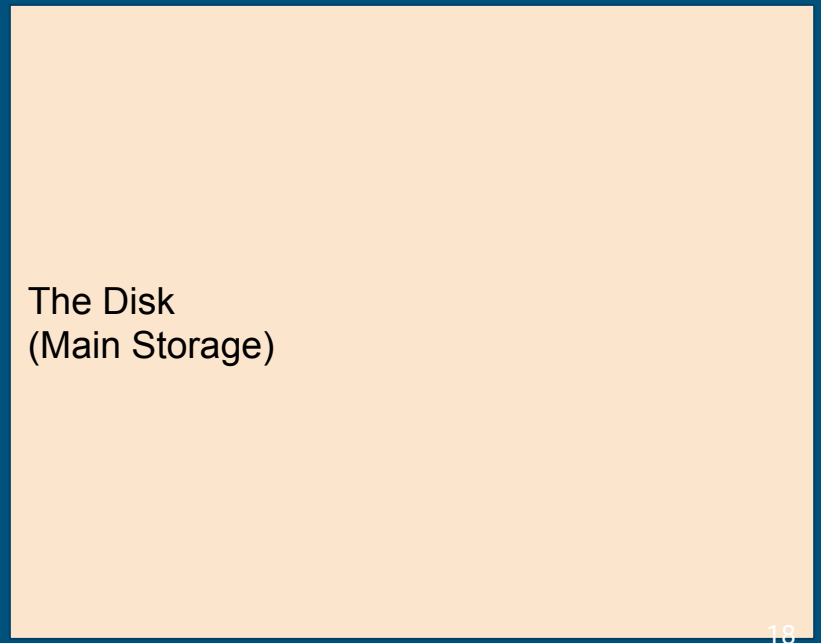




Example: before commit—CRASH

On reboot…
There's no commit in the log, so we should *not* copy anything to the disk

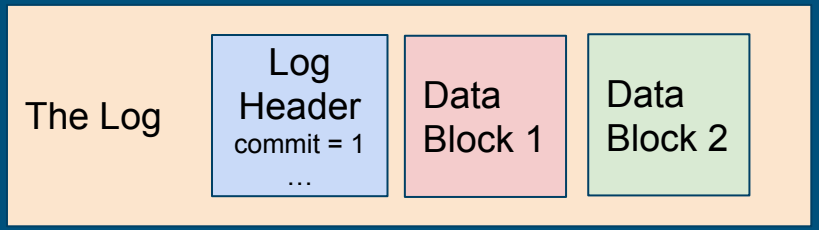


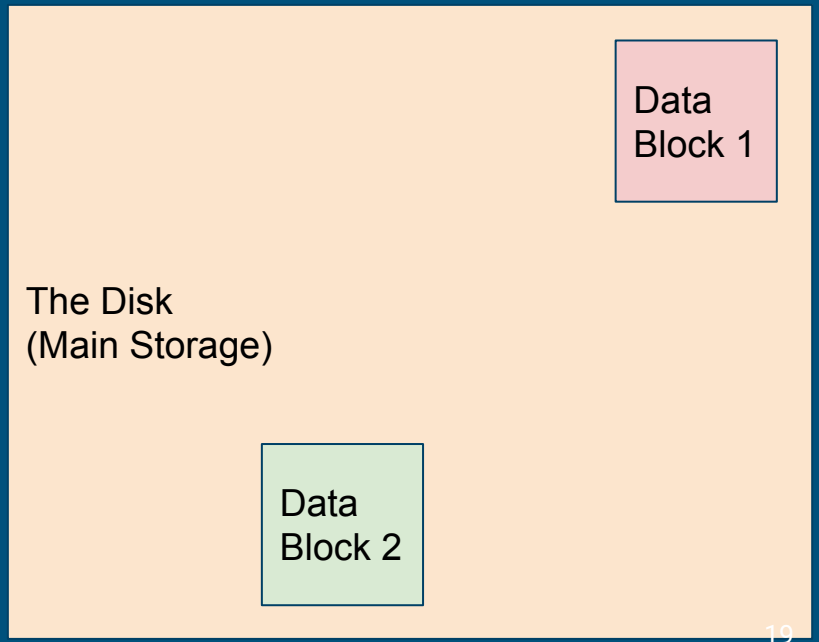


Example: after commit, before clear—CRASH

On reboot, we see that there *is* a commit flag

We can then copy block 1 and 2 to disk -- even though DB1 *was* already copied over, overwriting it with the same data is fine



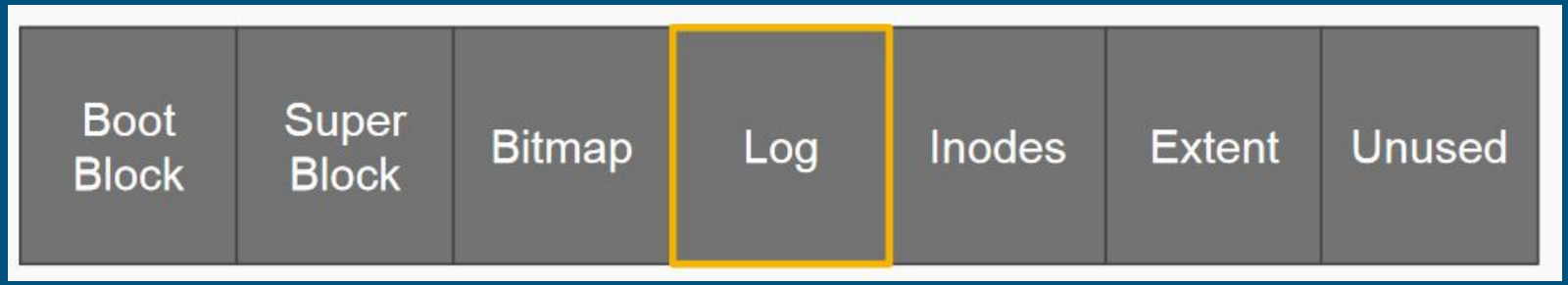


Where to Log?

It’s just blocks on disk, so you can put it anywhere you want (within reason)

After-bitmap, before-inodes is a pretty good place

You’ll need to update the superblock struct and mkfs.c (mkfs.c initializes the disk during the compiling process)



Log API

- The spec recommends designing an API for yourself for log operations:
  - **log_begin_tx()**: (optional) begin the process of a transaction
  - **log_write()**: wrapper function around normal block writes
  - **log_commit_tx()**: complete a transaction and write out the commit block
  - **log_apply()**: log playback when the system reboots and needs to check the log for disk consistency
    - Where/when should this be called? (Hint: inspect **kernel/fs.c**)

What should log_write() do differently?

- log_write() intended to be a wrapper function for bwrite() operations
- Instead of writing the block to its location on disk, we want to:
  - Write the block information to our log region
  - Update the log header with the location of the block

What happens after log_write()?

- Once all block writes in transaction have called log_write(), log_commit_tx() will be called
- Commit
  - Flush commit block to disk
  - Copy blocks from previous log_writes to their actual location on disk
  - Reset commit flag