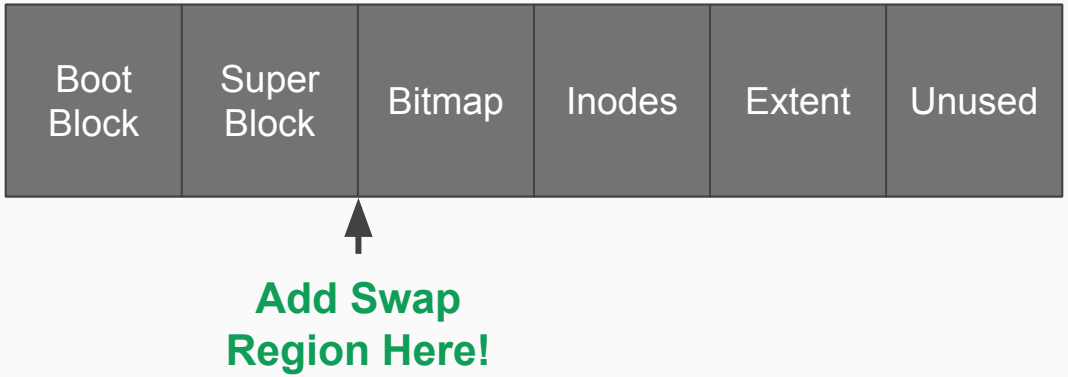
Section 8: Lab 4 Details

CSE 451 18WI

How to add the swap region?

Seems simple, but then you look at **`mkfs.c`**…

(Remember mkfs.c is **run by the host**, aka your computer's OS, before xk is booted. It sets up the disk for xk. It is **not** linked into the XK kernel)



mkfs.c

mkfs.c runs on the host. It creates a disk image and saves it to the given file name.

If you ran "**mkfs fs.img**", the program would create a file named "**fs.img**" containing a disk image with the layout shown on the previous slide and the given files stored in the filesystem.

argv[1]

```
87  if(argc < 2){
88    fprintf(stderr, "Usage: mkfs fs.img files...\n");
89    exit(1);
90  }
91
92  assert((BSIZE % sizeof(struct dinode)) == 0);
93  assert((BSIZE % sizeof(struct dirent)) == 0);
94
95  fsfd = open(argv[1], O_RDWR|O_CREAT|O_TRUNC, 0666);
96  if(fsfd < 0){
97    perror(argv[1]);
98    exit(1);
99  }
```

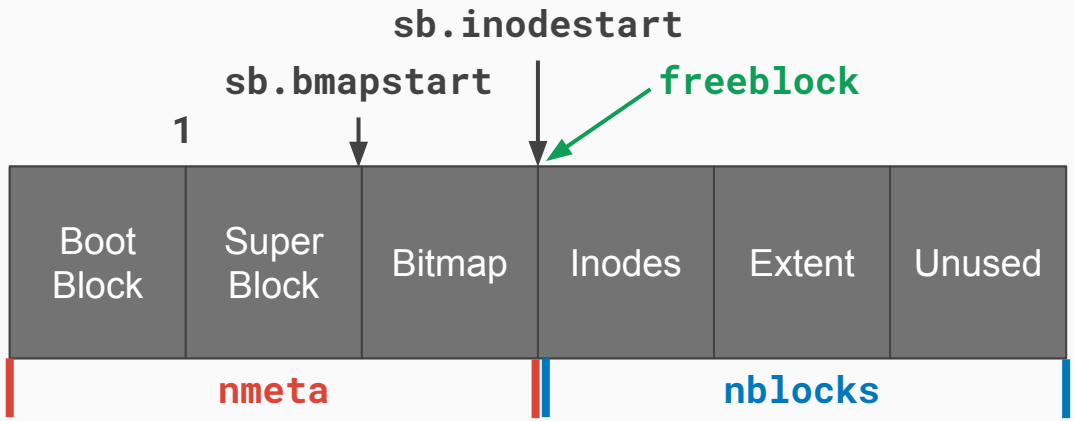
mkfs.c

The super block holds metadata about the file system, such as its size (`sb.size`), the number of blocks (`sb.nblocks`), and the starts of different regions (`sb.bmapstart`, `sb.inodestart`).

`freeblock` is used to keep track of the next free block in `mkfs.c`

```
102 nmeta = 2 + nbitmap;
103 nblocks = FSSIZE - nmeta;
104
105 sb.size = xint(FSSIZE);
106 sb.nblocks = xint(nblocks);
107 sb.bmapstart = xint(2);
108 sb.inodestart = xint(2+nbitmap);
    ...
112 freeblock = nmeta; // the first free block that
                        we can allocate
```

Super block



bread,
bwrite,
brelse

bread

- Reads data from disk
- Takes two arguments:
  - dev - the device
    - Use ROOTDEV, found in inc/param.h
  - block_no - The block number to write to

```
struct buf *buf = bread(dev, block_no);
memmove(mem, buf->data, BSIZE);
brelse(buf);
```

Always call brelse to help XK keep track of references to buffered disk blocks!

bwrite

- Writes data to disk
- First need to read data into the buffer, then you can modify the buffer
- Changes to the buffer won't be flushed to disk until you call bwrite
- Don't forget to call brelse after!

```
struct buf *buf = bread(dev, block_no);
memmove(buf->data, P2V(ph_addr), BSIZE);
bwrite(buf);
brelse(buf);
```

Let’s think!

What will happen when forking a process with some of its memory stored in the swap region?

You found a page to evict and know its virtual address, on what conditions should you update a vspace’s entry?

Concurrency Notes

- Cannot hold a spin lock while reading/writing to/from disk
- *Cannot* **acquiresleep()** a sleep lock while holding a spin lock
  - Since it may call **sleep()**, which calls **sched()**
  - You *can* **acquire()** a spin lock while holding a sleep lock
- If **kalloc()** swaps a page out
  - It may call **vspaceinvalidate()**, which may in turn call **kalloc()**
  - You might get a **acquire()** panic if you're not careful!
- Lots of potential concurrency bugs so be careful!