Lab 4 Details

Even more file stuff

Admin

- Lab 4 due next Friday, 6/4
  - Design doc feedback should be back by end of this weekend

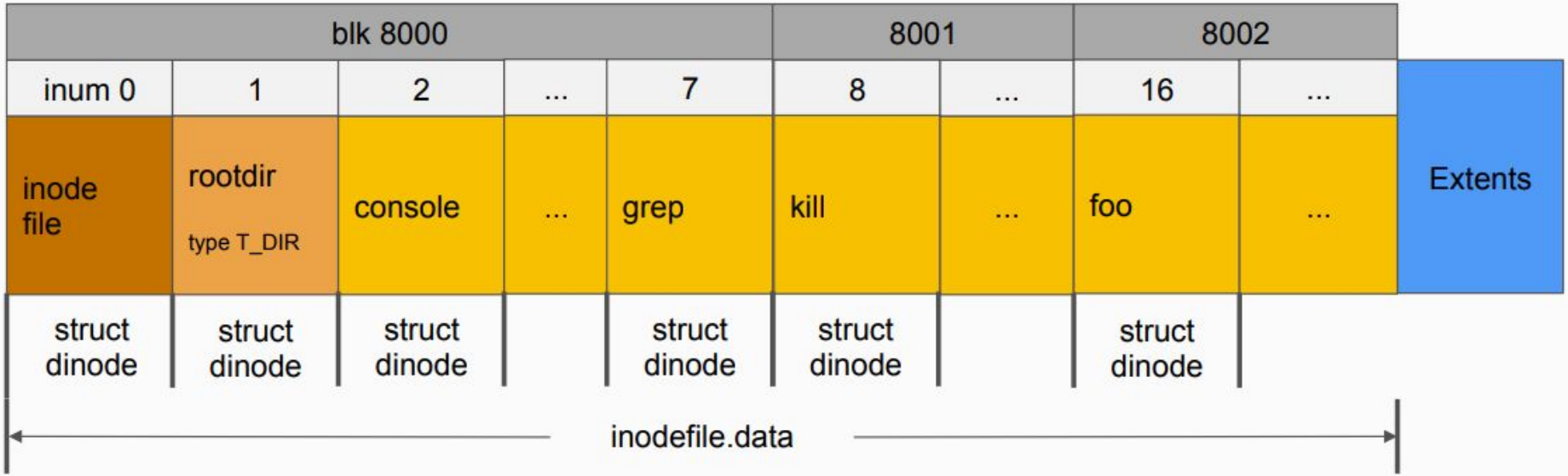
HARD DEADLINE (FOR EVERYTHING)

Part A: File Operations

Inodefile

- The inodefile is the “inodes” section on disk, which stores the table of inodes (struct dinode)
  - Reading from and writing to inodefile is just like reading/writing for a normal file
- 0th inode is the inodefile itself
  - Data field in 0th inode corresponds to inodes region
- 1st inode is the root directory
  - Data field is array of directory entries (struct dirent)
- icache.inodefile points to the inode file

Inodefile


blk 8000
8001
8002
inum 0
1
2
...
7
8
...
16
...
inode file
rootdir
type T_DIR
console
...
grep
kill
...
foo
...
Extents
struct dinode
struct dinode
struct dinode
struct dinode
struct dinode
struct dinode
inodefile.data

Helpful functions

iget: create a cache entry for the in-memory copy of the inode, but the entry is empty (doesn’t synchronize with dinode)

locki: copy information from dinode to the in-memory inode cache

read_dinode: read the dinode from the disk

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?

Bitmap

- Each block contains 512 bytes
  - Each block in bitmap represents 512 * 8 = 4096 blocks
    - (i.e., block at sb.bmapstart -> blocks 0-4095 , sb.bmapstart + 1 -> 4096-8191, etc.
  - Need to use bitmasking to mark blocks in bitmap
- Some useful macros
  - BBLOCK(b, sb) -> block number in bitmap containing b

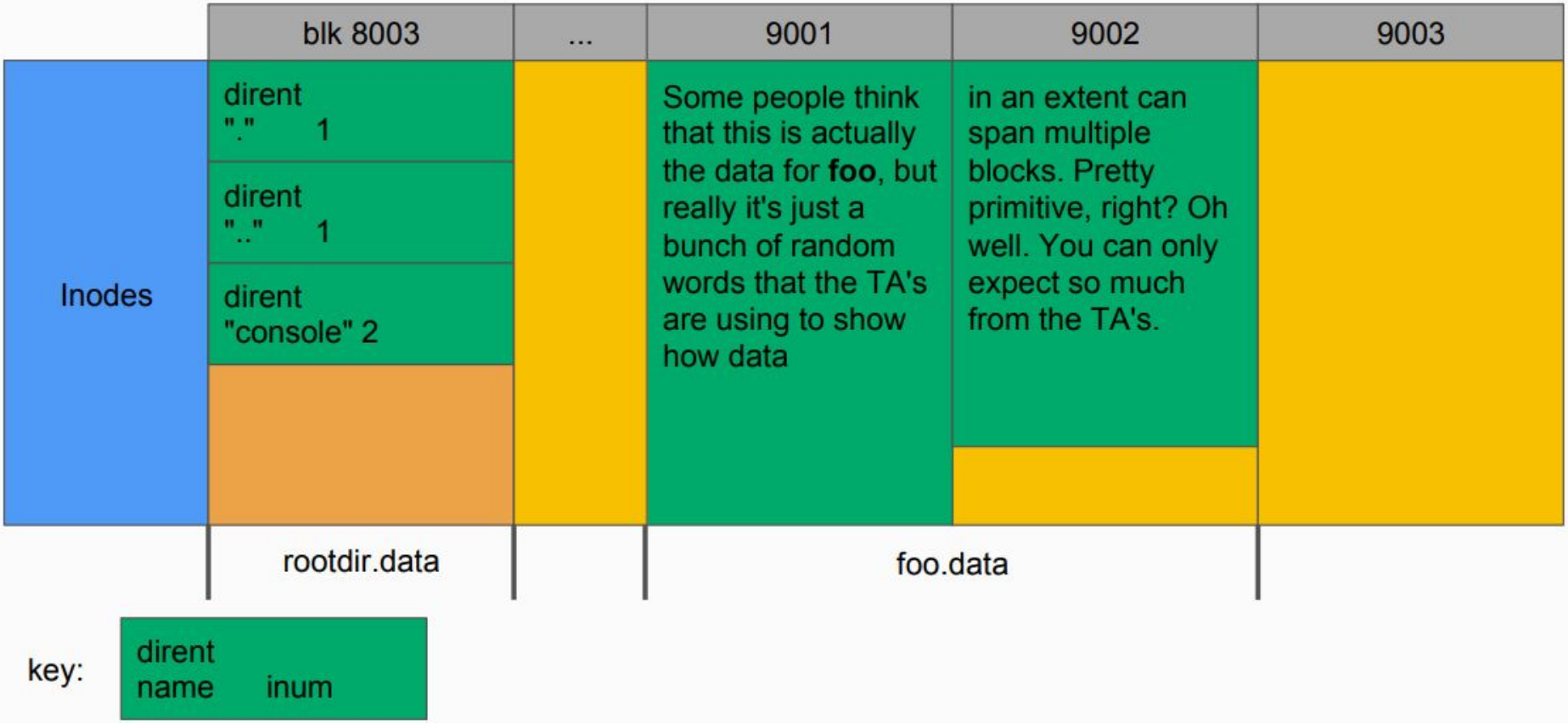
```
// Bitmap bits per block
#define BPB (BSIZE * 8)

// Block of free map containing bit for block b
#define BBLOCK(b, sb) ((b) / BPB + (sb).bmapstart)
```

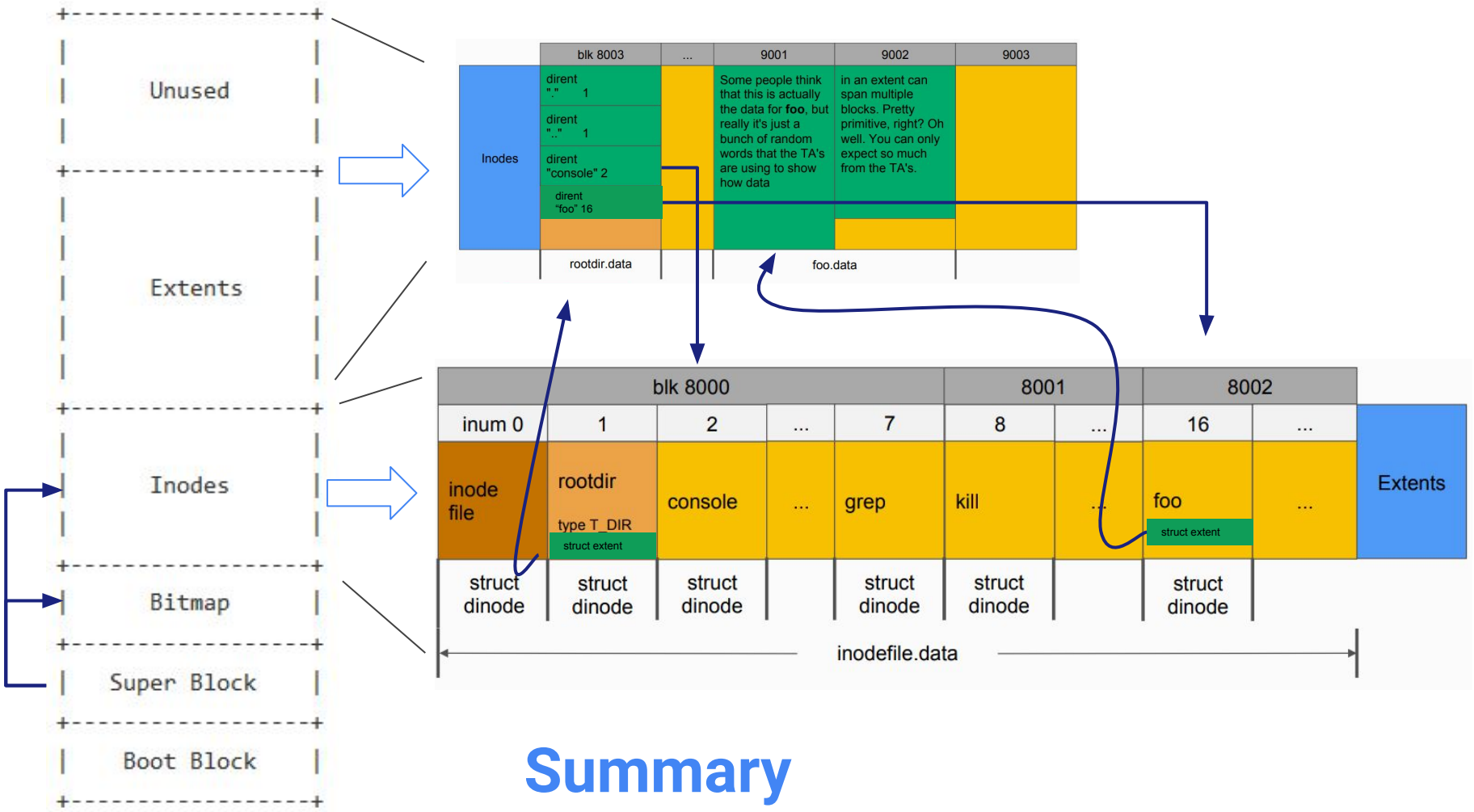
Extents

- Extents region - where the actual data for files in the filesystem lives (excluding the initial inode file)

- Extent - sequence of contiguous blocks of disk
  - When allocating an extent for a file, all blocks in the extent should be marked used in the bitmap even if no data is written yet
    - “Reserving” contiguous blocks for file to use

Extents


blk 8003
...
9001
9002
9003
Inodes
dirent "." 1
dirent ".." 1
dirent "console" 2
Some people think that this is actually the data for **foo**, but really it's just a bunch of random words that the TA's are using to show how data
in an extent can span multiple blocks. Pretty primitive, right? Oh well. You can only expect so much from the TA's.
rootdir.data
foo.data
key:
dirent
name
inum


Unused
Extents
Inodes
Bitmap
Super Block
Boot Block
blk 8003
...
9001
9002
9003
Inodes
dirent "." 1
dirent ".." 1
dirent "console" 2
dirent "foo" 16
Some people think that this is actually the data for foo, but really it's just a bunch of random words that the TA's are using to show how data
in an extent can span multiple blocks. Pretty primitive, right? Oh well. You can only expect so much from the TA's.
rootdir.data
foo.data
blk 8000
8001
8002
inum 0
1
2
...
7
8
...
16
...
inode file
rootdir
type T_DIR
struct extent
console
...
grep
kill
...
foo
struct extent
...
Extents
struct dinode
struct dinode
struct dinode
struct dinode
struct dinode
struct dinode
inodefile.data

Summary

Part B: Crash Safety

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_recover()**: 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
  - keep the block in memory until transaction successfully commits (performance optimization)

- To write to a block but *keep changes in memory*
  - Look into setting B_DIRTY bit for that block when calling bwrite - this will ensure the changes are are not immediately flushed to disk

What should log_write() do differently?

- Once all block writes in transaction have called log_write(), log_commit_tx() will be called

- Commit
  - Flush commit block to disk
  - Flush dirty blocks from previous log_writes to their actual location on disk
    - How?
  - Reset commit flag

Questions?

Good luck on Lab 4!