# Project 3A

CS111 Discussion 1C

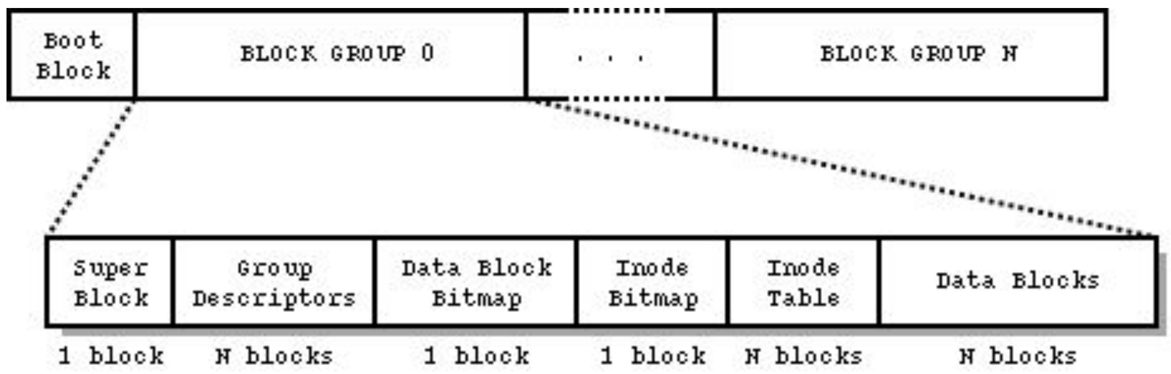
Note: You can form groups of 2 for this project

## EXT2 filesystem

- You will be provided with a filesystem image (EXT2)
- You will explore it
  - You will mount it as a file, which means that:
    - You can read a specific number of byte at a specific offset
    - You will have to repeat these operations for all content
    - The problem becomes calculating the correct offset
- You will output a summary to stdout, describing:
  - The Superblock
  - Groups
  - Free Lists
  - Inodes
  - Indirect Blocks
  - Directories
- This involves a lot of simple code, **if** you have a clear image of what a filesystem is

# EXT2 Filesystem

Read this carefully. Everything you need is in there.



Where to start? We want to find correct offsets.
The superblock! We know where it is

Note: File offsets are not addresses in memory! They can't be dereferenced with *
Note: A function that takes in a block number and outputs its offset would be useful

## Superblock

- Contains all information about the filesystem:
  - # inodes
  - # blocks
  - How many blocks/inodes are free
  - # blocks/inodes per group
  - log(block size)
- Is located at an offset of 1024 bytes from the start of the device
  - This is the only data that we know how to localize, it is the starting point
  - We know the superblock is 1 block long
- You will need to refer to superblock data several times: make it accessible! (save it in a structure)

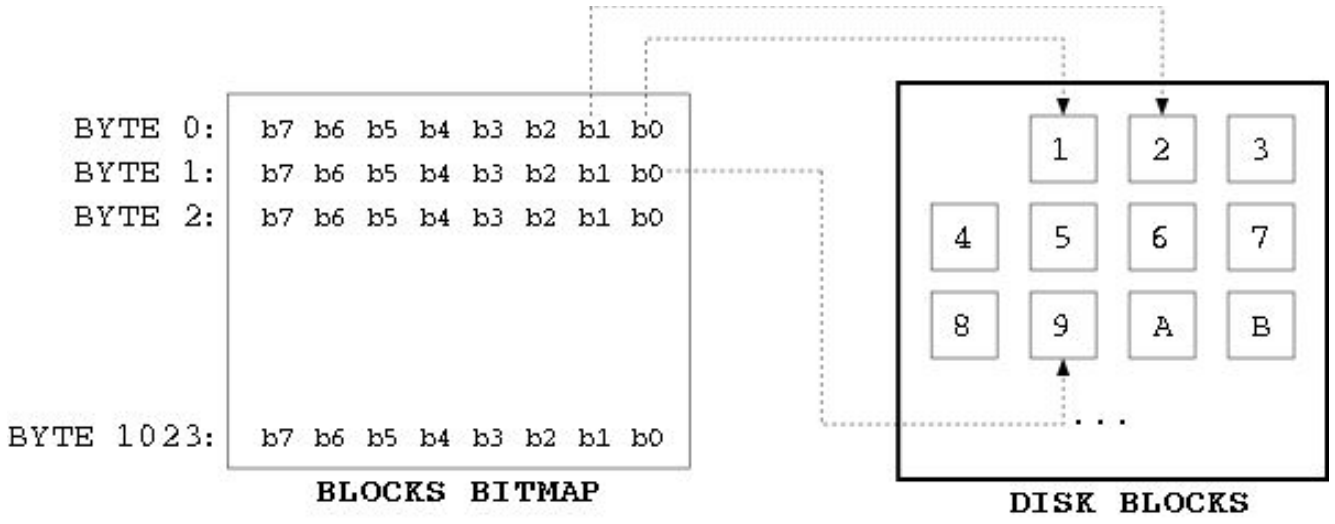
## Block Group Descriptor Table

Superblock offset

- Located at block 2, so at block_size + 1024
  - Block size is in the superblock
- It is an array of block group descriptors. Each of these block group descriptors provide (for 1 group):
  - The location of the inode bitmap/table
  - The location of the block bitmap
  - # of free blocks/inode
- This table can be stored on multiple blocks
  - How many? -> Superblock

## Block Bitmap

- Where is it located?
  - Bg_block_bitmap of its block group descriptor
- What does it represent?
  - Current state of a block within that block group (1 = used / 0=free)
- How is the information represented?



B1: bit 0 of byte 0
B2 : bit 1 of byte 0
.
.
.
B8 : bit 7 of byte 0
B9 : bit 0 of byte 1

## Inode Bitmap

- Where is it located?
  - Bg_inode_bitmap in group descriptor
- What does it represent?
  - Current state of an inode within the inode table (1=used)
- Works similarly to a Block Bitmap

## Inode Table

- Where is it located?
  - bg_inode_table in the group descriptor
- It is an array of inodes. Each of those represent a single file (a directory, socket, buffer, symbolic link, regular file):
  - size and owner
  - access/modification/creation times (seconds since 1/1/70)
  - the number of blocks containing the data of the inode
  - which blocks contain the data of the file pointed to by the inode
  - Note that there is no filename in the inode

- The first few entries of the inode table are reserved

## Locating an inode

- They are numerically ordered
  - Inode number -> index in the inode table

- The size of the inode table is fixed at format time
  - There is a cap on the number of entries
  - The size is (# of inodes * inode size) (cf superblock)

- Therefore the local inode index (relative to the current block group) is :
  - (inode_idx -1) % inodes_per_group

## Inode pointer structure

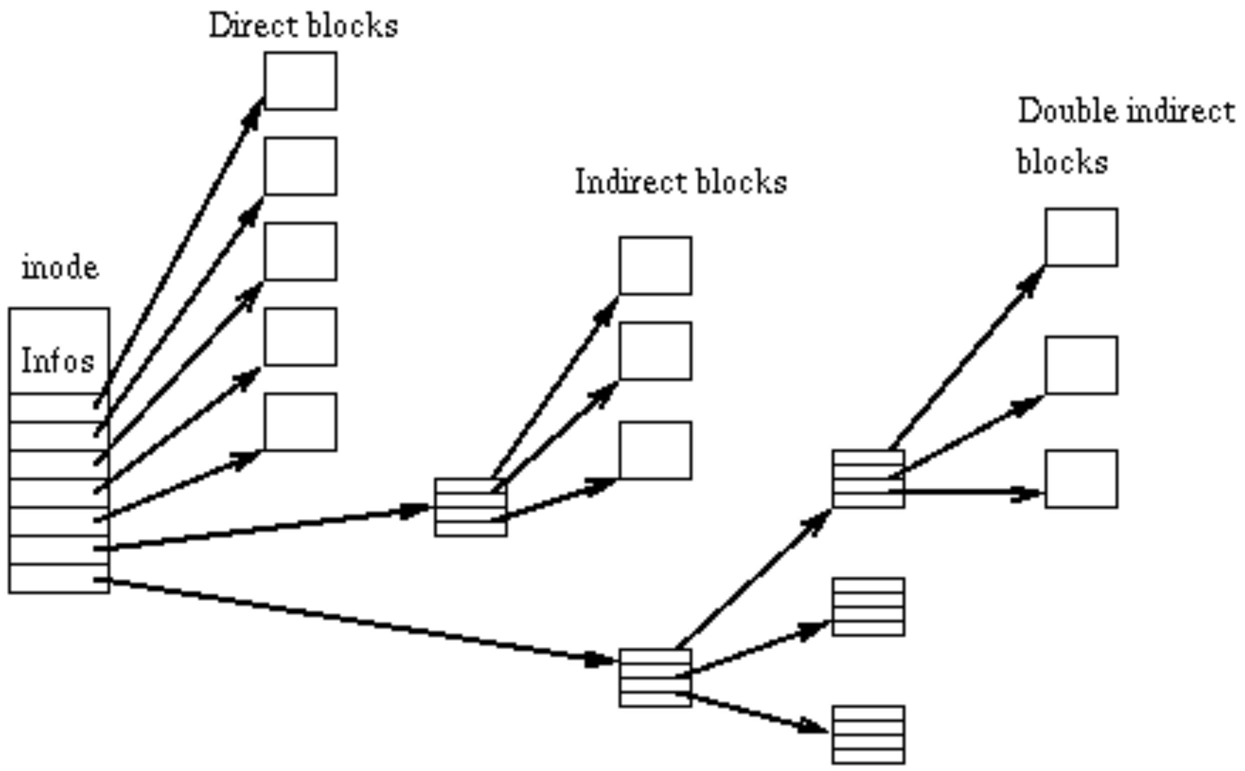
- The inode points to the blocks that contain its data

- The block numbers are stored in a table
  - The first 12 are ‘direct’ blocks
  - The 13th block is the ‘indirect block’
  - The 14th block is the ‘doubly indirect block’
  - The 15th block is the ‘triply indirect block’
  - If a 0 is encountered, all subsequent block pointers should be 0 (no other blocks defined)

- The indirect block structure is used to be able to reference files that would need more that 15 blocks with a single (fixed size inode)

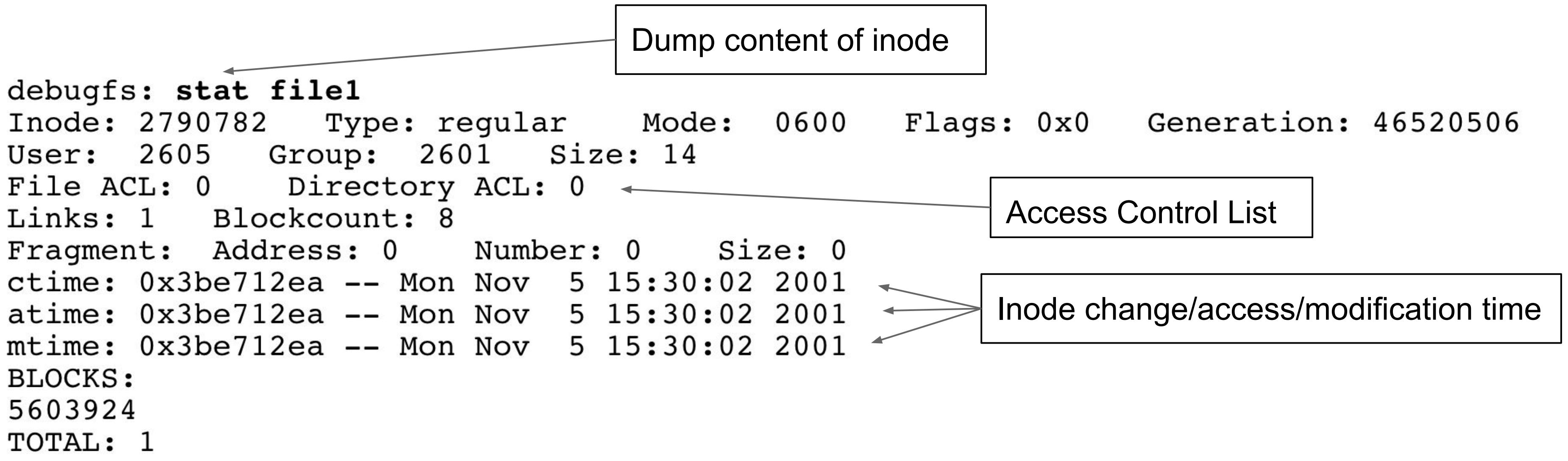
## Inode pointer structure

A picture is probably helpful:



## Debugfs

- Download the image and mount it
  - This requires sudo: work on your machine
- You can explore it using debugfs(8)
  - You will need to research this tool to properly interpret values



```
debugfs: stat file1
Inode: 2790782   Type: regular    Mode:  0600   Flags: 0x0   Generation: 46520506
User:  2605   Group:  2601   Size: 14
File ACL: 0    Directory ACL: 0
Links: 1   Blockcount: 8
Fragment:  Address: 0    Number: 0    Size: 0
ctime: 0x3be712ea -- Mon Nov  5 15:30:02 2001
atime: 0x3be712ea -- Mon Nov  5 15:30:02 2001
mtime: 0x3be712ea -- Mon Nov  5 15:30:02 2001
BLOCKS:
5603924
TOTAL: 1
```

## Debugfs

- Other useful commands include:
  - bd : block dump
  - testi <inode> : test if inode is marked allocated in the inode bitmap
  - testb <block> : test if block is marked allocated in the block bitmap

- You can use this tool while writing your program to verify that you are exploring the filesystem correctly

## General code structure

- Find/analyze/report superblock
- Copy info to common structure
- Read group descriptions
  - How many blocks per group?
  - How many inodes per group?
- Find and report both bitmaps
- Analyze all allocated inodes
  - Is it a directory? A regular file?
  - What is the level of indirection?

## Some useful notes

- Number of blocks per group?

## Some useful notes

- Number of blocks per group?
  - You can find this in the superblock, except for the last group
- What is the block size?

## Some useful notes

- Number of blocks per group?
  - You can find this in the superblock, except for the last group
- What is the block size?
  - 1024 << superblock.s_log_block_size
- Is the file a regular file or a directory?

## Some useful notes

- Number of blocks per group?
  - You can find this in the superblock, except for the last group
- What is the block size?
  - 1024 << superblock.s_log_block_size
- Is the file a regular file or a directory?
  - Answer is in the first field of the inode : the ‘mode’

- Write a function that takes in a block number and returns the absolute address (offset) of that block

- Write a function that reads a given number of blocks at a given offset

## Iterating through inodes

- What is the level of indirection?
  - For a 1KiB block size:
    - Block 1 to 12 are direct
    - Block 13 to 268 are indirect
    - How many double indirect blocks?

## Iterating through inodes

- What is the level of indirection?
  - For a 1KiB block size:
    - Block 1 to 12 are direct
    - Block 13 to 268 are indirect
    - How many double indirect blocks? (256)*(256) = 65536
    - How many triple indirect blocks?

## Iterating through inodes

- What is the level of indirection?
  - For a 1KiB block size:
    - Block 1 to 12 are direct
    - Block 13 to 268 are indirect
    - How many double indirect blocks? (256)*(256) = 65536
    - How many triple indirect blocks? (256)*(256)*(256) = 16777216

- How does your offset change when you go through it?
  - Direct blocks -> blocksize
  - Indirect blocks?

## Iterating through inodes

- What is the level of indirection?
  - For a 1KiB block size:
    - Block 1 to 12 are direct
    - Block 13 to 268 are indirect
    - How many double indirect blocks? (256)*(256) = 65536
    - How many triple indirect blocks? (256)*(256)*(256) = 16777216
- How does your offset change when you go through it?
  - Direct blocks -> blocksize
  - Indirect blocks? Multiply by blocksize per indirection level

## A word on the trivial image

- You are provided with a small filesystem and the correct analysis output for that filesystem
- You are expected to be able to reproduce 'trivial.csv' from the given filesystem

**BUT**

- Doing so won't guarantee that your program is 100% correct, this tests basic functionality
- Also note that any access will modify the inodes!
  - This means that you won't be able to refer to the provided .csv
  - Use a fresh copy / mount in read only

## FAQ

- How do I mount the image on a mac?

  ```
  hdiutil attach -readonly -nomount ./trivial.img
  ```

- Can I assume that there will only be a single group in the filesystem used for grading?

  Yes!