CS 111
Computer Systems Principles
Section 1E Week 6

Tengyu Liu

Project 3A – Interpreting EXT2 Filesystem

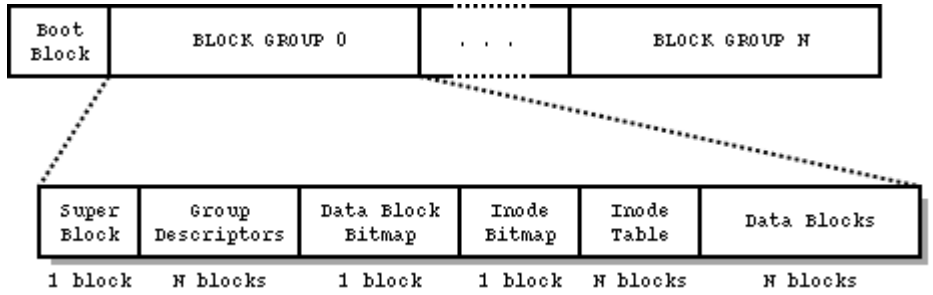
- Goal: Parse an EXT2 filesystem image and produce a summary
  - Task: Open filesystem as a **file** and **read** from specific offsets
  - Difficulty: Calculating offsets for different elements
  - Summaries include:
    - Superblock
    - Groups
    - Free lists
    - Inodes
    - Indirect blocks
    - Directories
  - A lot of simple code if you have a clear image of what a filesystem is

pread() vs. read()

- Both are responsible for reading a file
- read() from a file twice will produce different results
  - You changed the pointer to the position (offset) in file each time
- pread() is idempotent: calling it twice produces the same result
  - You manage your own offset
  - `ssize_t pread(int fd, void *buf, size_t count, off_t offset)`
  - The difficulty is calculating the correct offset

EXT2 Filesystem

- Read this carefully. Everything you need is in here.



- Start by reading the superblock. Superblock tells us how to compute correct offsets.
- Note:
  - File offsets are not addresses in memory. They cannot be dereferenced with *
  - A function that takes a block number and outputs its offset would be helpful

Superblock

- Contains all the 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. This is our 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 struct.

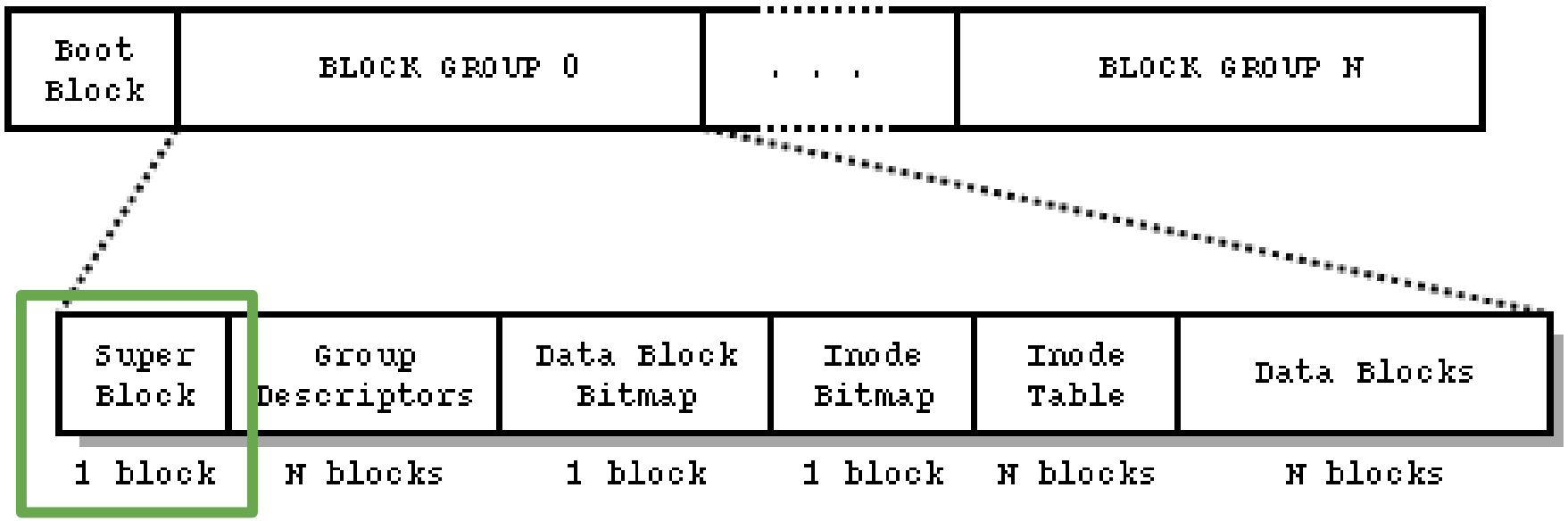
How to read a superblock?

- Superblock is organized as described in
  - `struct ext2_super_block` in ext2_fs.h
- How to verify if it is a superblock?
  - Check `__u16 s_magic` with `#define EXT2_SUPER_MAGIC 0xEF53`
- After finding the superblock, report the stats mentioned in the spec

We still don’t know the block size

- In `struct ext2_super_block`
  - `__u32 s_log_block_size; /*log2(Block size) */`
  - The block size is computed using this 32bit value as the number of bits to shift left the value 1024. This value may only be non-negative.
  - `/* bsize = EXT2_MIN_BLOCK_SIZE << s_log_block_size    */`

Where are we?



1024 1024+bsize

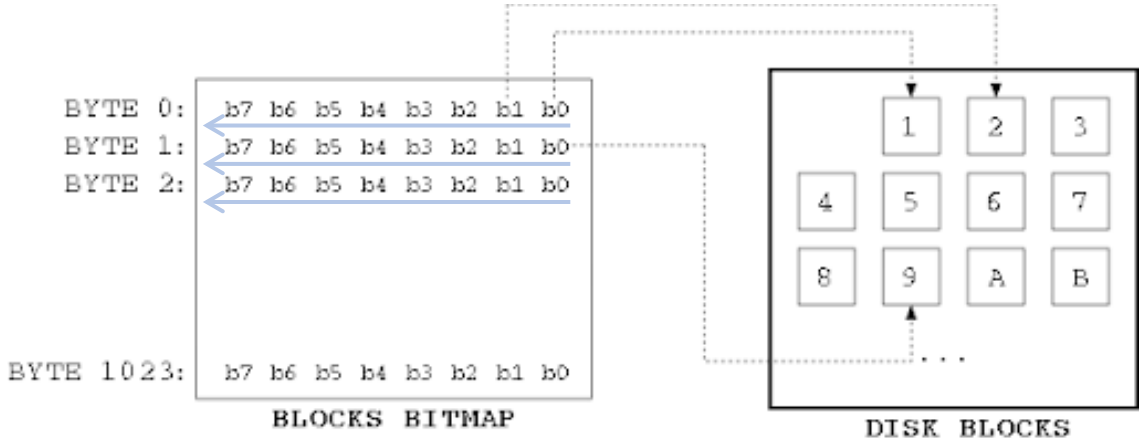
```
/* bsize = EXT2_MIN_BLOCK_SIZE << s_log_block_size     */
```

Block Group Descriptor Table

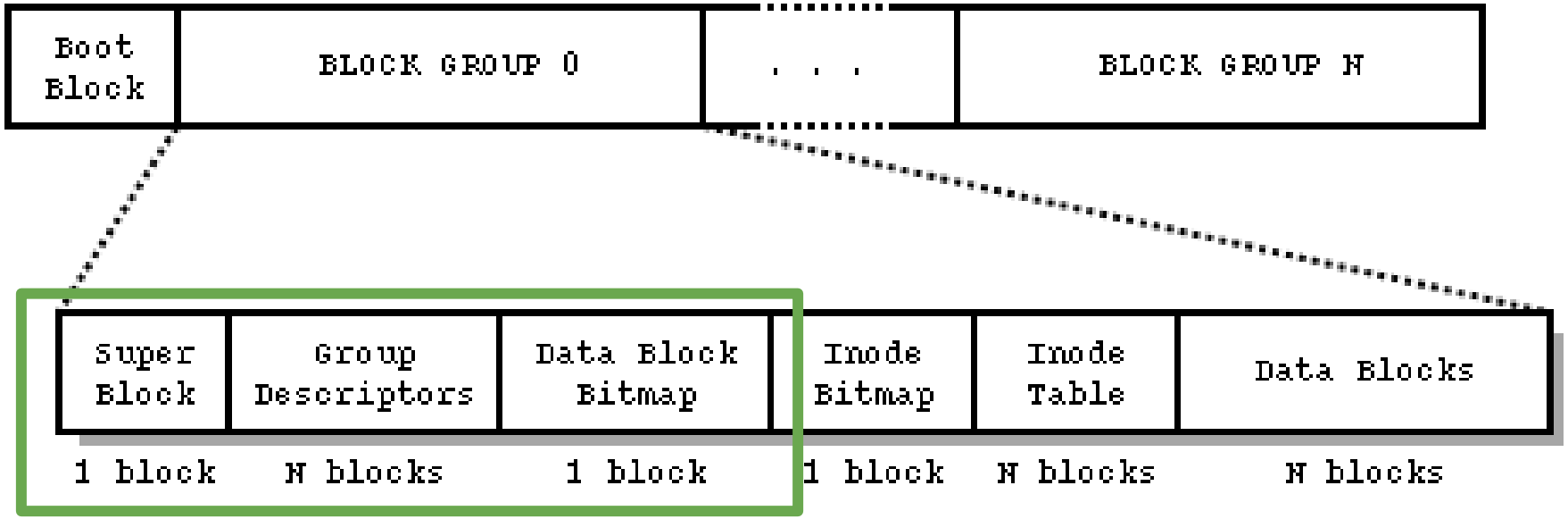
- Located at block 2, so at block_size + 1024
- An array of block group descriptors. Each of the descriptors provide
  - The location of the inode bitmap / table
  - The location of the block bitmap
  - # of free blocks / inode
- Parse with `struct ext2_group_desc`

Block Bitmap

- Where is it located?
  - `ext2_group_desc.bg_block_bitmap` gives the block number of the block bitmap
  - Use the block number and the block size to calculate the byte offset
  - Read 1 block of data into buffer starting from the correct offset
  - This is your block bitmap
- What does it represent?
  - Current state of a block within that block group (1=used, 0=free)



Where are we?


Boot Block
BLOCK GROUP 0
. . .
BLOCK GROUP N
Super Block
Group Descriptors
Data Block Bitmap
Inode Bitmap
Inode Table
Data Blocks
1 block
N blocks
1 block
1 block
N blocks
N blocks

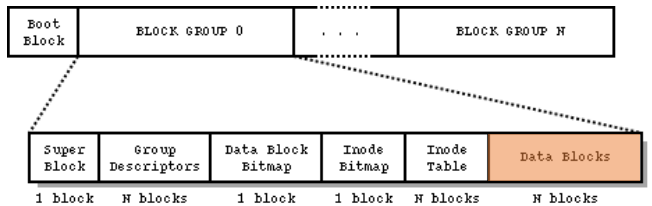
Inode bitmap

- Location is indicated by bg_inode_bitmap
- Represents the current state of an inode within the inode table
- Works similarly as a block bitmap

Blocks vs. inodes

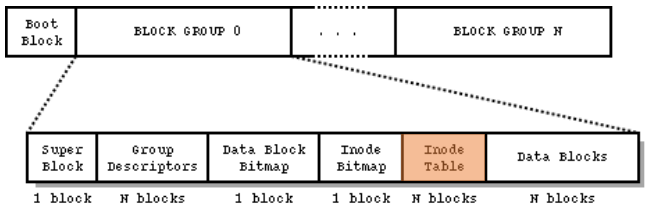
blocks

- Storage space in a filesystem that is used for storing data in files
  - Text
  - Image
  - Video
  - Binary data
  - …

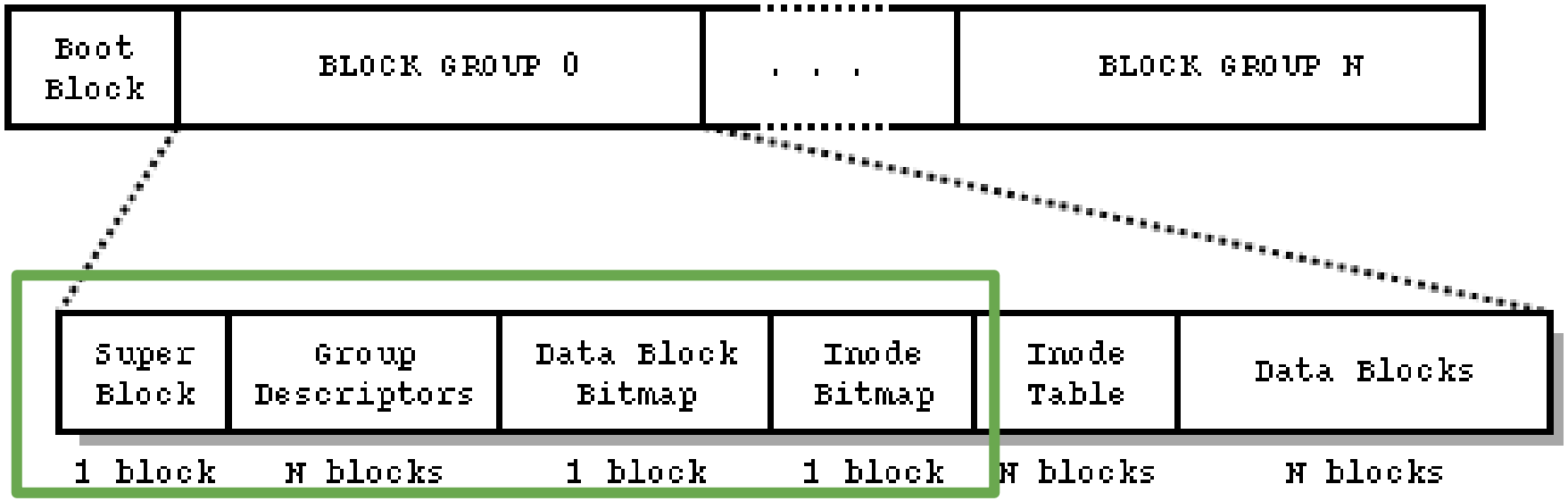


inodes

- Data structures in a filesystem that is used for storing meta-data of files
  - File type
  - Owner
  - Size
  - Create/modify/last-access time
  - …



Where are we?


Boot Block
BLOCK GROUP 0
. . .
BLOCK GROUP N
Super Block
Group Descriptors
Data Block Bitmap
Inode Bitmap
Inode Table
Data Blocks
1 block
N blocks
1 block
1 block
N blocks
N blocks

Inode table

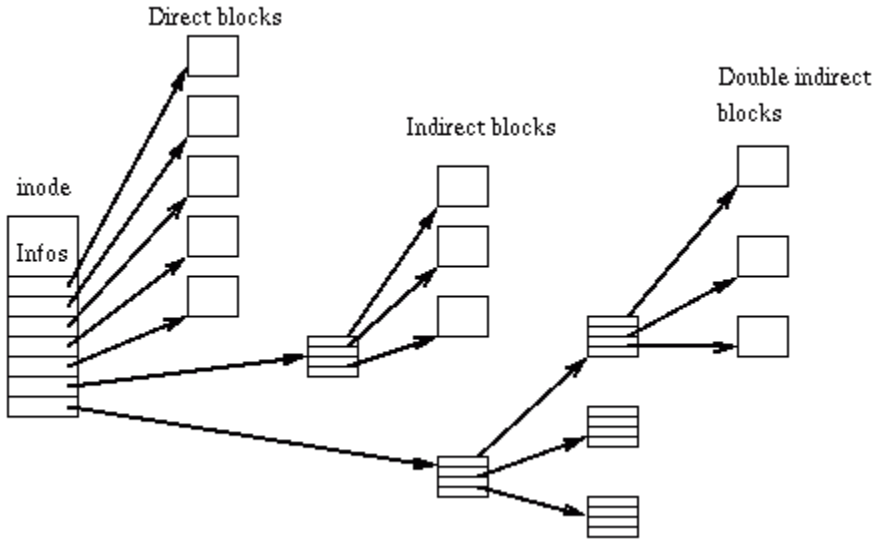
- Location is indicated by bg_inode_table
- An array of inodes
- Each represents a single file
  - Directory, socket, buffer, symbolic link, regular file, …
- Contains information
  - Size and owner
  - Access / modification / creation times (seconds since 1/1/1970)
  - Number of blocks containing the data of the inode
  - Which blocks contain the data of the file pointed to by the inode
  - 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)
  - Information in superblock
- 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
  - The documentation mentions that if a 0 is encountered, all subsequent block pointers should be 0 (no other blocks defined)
    - **This is not true for this lab**
  - The indirect block structure is used to reference files that need more than 15 blocks with a single fixed size inode

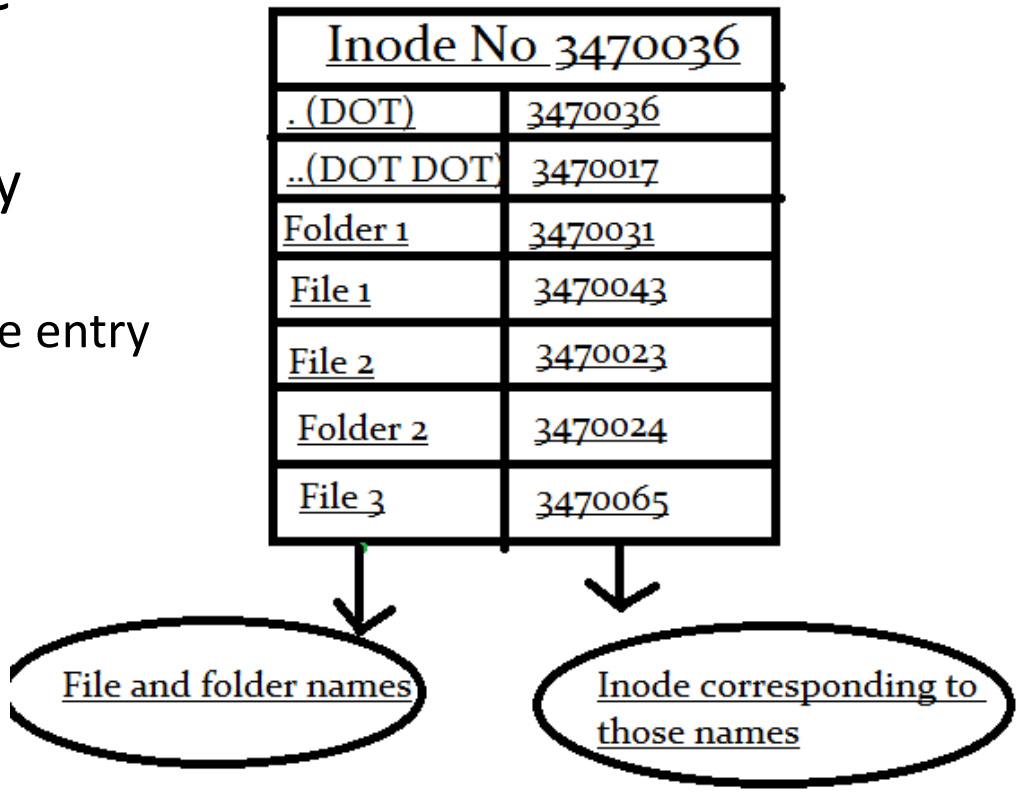


Inode i_mode

- i_mode has information about
  - File format
  - Access rights
- <sys/stat.h> defines macros to identify file type and permissions
  - inode.i_mode & S_IFLNK // checks if the inode is a link
  - inode.i_mode & S_DIR // checks if the inode is a directory

Directory Structure

- Stored in data blocks, pointed by an inode
- Stored as a linked list
  - Each node in the list is a struct ext2_dir_entry
    - The name of the entry
    - The inode pointing to the data associated with the entry
    - The distance to the next netry
- How do you find a file?
  - Iterate through the directory structure
  - Just like going through a linked list
- Entries do not span several blocks



$ls /home/user0/
music/
test.txt
bin/

/home/user0

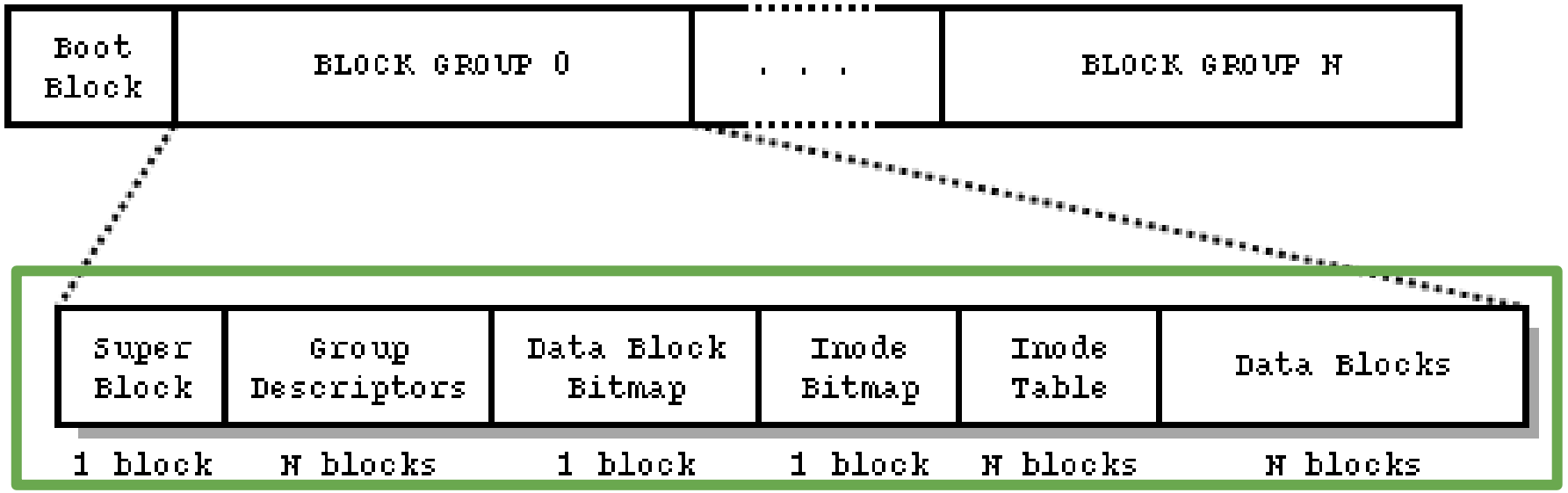
block: 203

i_ node = 13
i_block[0] = 203

block 203

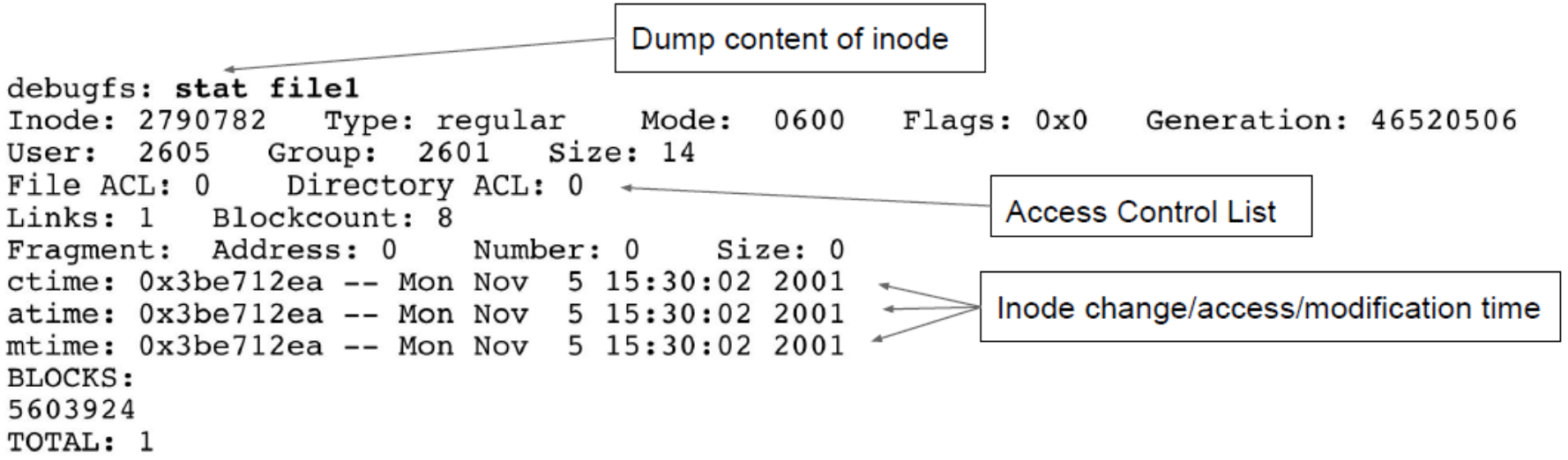
	inode	rec_len	name_len	file_type	name								
offset: 0	13	12	1	2	.	\0	\0	\0					file: .
offset: 12	10	12	2	2	.	.	\0	\0					file: ..
offset: 24	18	16	5	2	m	u	s	i	c	\0	\0	\0	file: music
offset: 40	15	16	8	1	t	e	s	t	.	t	x	t	file: test.txt
offset: 56	19	12	3	2	b	i	n	\0					file: bin

Where are we?


Boot Block
BLOCK GROUP 0
. . .
BLOCK GROUP N
Super Block
Group Descriptors
Data Block Bitmap
Inode Bitmap
Inode Table
Data Blocks
1 block
N blocks
1 block
1 block
N blocks
N blocks

Misc.

- Downloading the image and mounting it
  - This requires sudo permission
  - Work on your local machine
- You can explore the image with debugfs(8)
  - You will need to research this tool to properly interpret values



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 to verify that you are parsing the filesystem correctly.

General code structure

- Always make the superblock data easily accessible to you
- 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 or a regular file?
  - What is the level of indirection?

Some useful notes

- What is the block size?
  - 1024 << superblock.s_log_block_size
- Is the file a regular file or a directory?
  - Look into the i_mode field in inode
- Write a function that takes a block number and returns the absolute address (offset) of that block
- Write a function that reads a given number of blocks at a give offset

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.
- 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 every time or mount in read only mode

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!