COP4610
Introduction to Operating Systems
Project #3: Implementing a FAT32 File System

Info

- TA/Grader: **Juan Pablo Conde**
  first middle last
- Office hours Wed 3:30-4:30pm, Thu 11:30am-12:30pm, or by appointment
- Deadline: **December 1st, 11:59pm**
  - No late submissions
- Work in teams of 3 members
  - Can work alone with TA approval (not encouraged)
  - Submit your team information on Canvas

A word of advice

- Start early!
  - Takes time to understand (and to code)
- Ask questions
  - Recitation
  - TA's office hours
- Separate the workload evenly
  - Report problems to TA ASAP
    - Team-splitting only authorized by TA
    - Team members that work significantly less will get points deducted
  - If more than one working on the same computer and only one pushes changes, say it on the report

A word of advice

- Work together
  - No need to work on the same computer, but discuss things together
  - Split workload but keep in touch
  - Help teammates struggling with tasks
  - Be proactive

Don’t even try to cheat

- Submissions compared
  - Other groups
  - Other submissions from previous classes
  - Public repositories
- Subject to penalties for violating academic honor policy
  - Fail the class
  - Other penalties
  - Not worth the risk
  - Fact: Two groups failed last spring

What if you didn’t contribute?

- Too bad for you (bad grade)
  - Grades assigned individually
- Appeals on individual grading need to be justified
  - Teammates must support what you say (good luck)
  - Provide Gitlog
  - Other evidence might be requested by the TA
- **Keep this in mind, especially if you are expecting to graduate this semester ;)**

Outline

- FAT32 overview
- Implementation overview
- Hexadecimal and endianness refresher
- Exploring the FAT32 image file
- FAT32 specs
  - Layout
  - Reserved region
  - FAT
  - Data region

FAT32 overview

User space

Kernel

Hardware interface

FAT32 overview

- A file system
- Defines how data is stored in the storage device
  - and also how it is accessed
- File allocation
  - How do I store a file?
- Navigation
  - How do I find a specific file and the blocks that belong to it?
  - How do I navigate through directories?
- Allocated and free space management

Implementation overview

- C language
- No kernel programming
  - Normal executable
- Work on image file
  - Raw bytes
  - Contents have a FAT32 structure
- Program features
  - Read input from user
  - Manipulate the image to access/write data

Hexadecimal refresher

- Base 16: digits range from 0 to 15
  - Represented by digits `0–9` and `A–F`
  - Prefix '`0x`' to differentiate from decimal
- Binary to Hex matching
  - `0000` to `1111` → `0` to `F` (or 0 to 15 in decimal)
  - 4 bits needed to represent a hex digit (2^4)
- Two hex digits = 8 bits = 1 byte
  - E.g: `0xA1`, `0xBE`, `0x04`

Endiannes refresher

- Big endian
  - Most significant byte corresponds to least significant digits
    - E.g.: byte seq. `2A 3B 91 23` → `0x2A3B9123`
- Little endian
  - Least significant byte corresponds to least significant digits
    - E.g.: byte seq. `2A 3B 91 23` → `0x23913B2A`
- Intel architecture uses **little endian**

Exploring the FAT32 image file

- Two approaches
  - Mount the image
  - Use a hex editor
    - **Hexedit (Linux)**
    - Hex Fiend (Mac)
    - HxD (Windows)

Mounting the FAT32 image

- In a Linux environment:

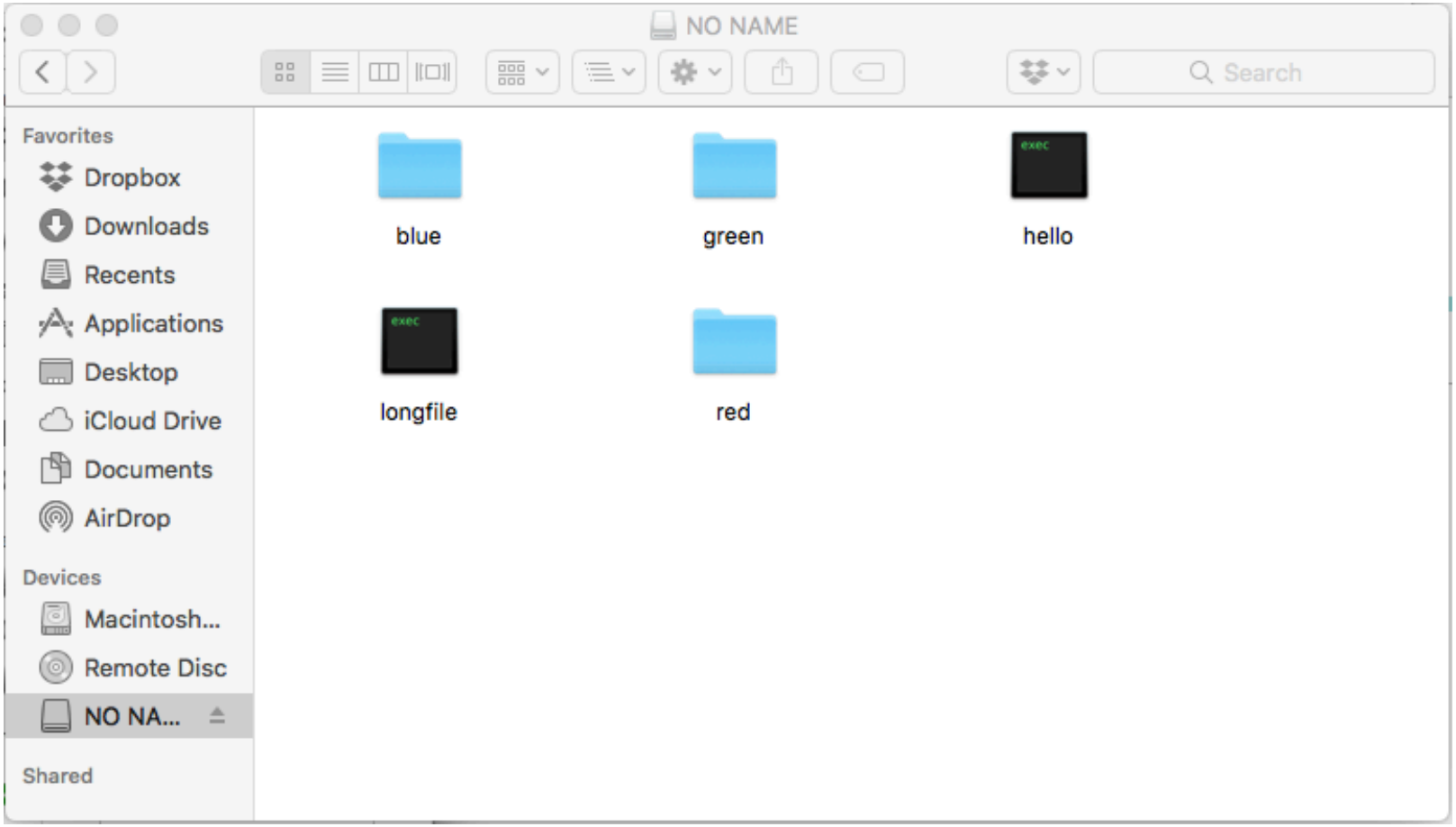
```
$ mkdir -p mnt_pnt
$ sudo mount -o loop /path/to/image mnt_pnt
$ cd mnt_pnt
```

- Explore the image as if it was a normal volume
  - CLI, Finder, Explorer, …
- To unmount:

```
$ sudo umount mnt_pnt
```

- Note: Your program has to work with the image **unmounted**. Mount only to browse the image

Mounting the FAT32 image



Hexedit

- Often included in Linux distributions
- Free downloads for hex viewers available
  - View raw data content of an image file
- Test your project by exploring the image
  - Hexedit
  - Mounting
- Opening the image:

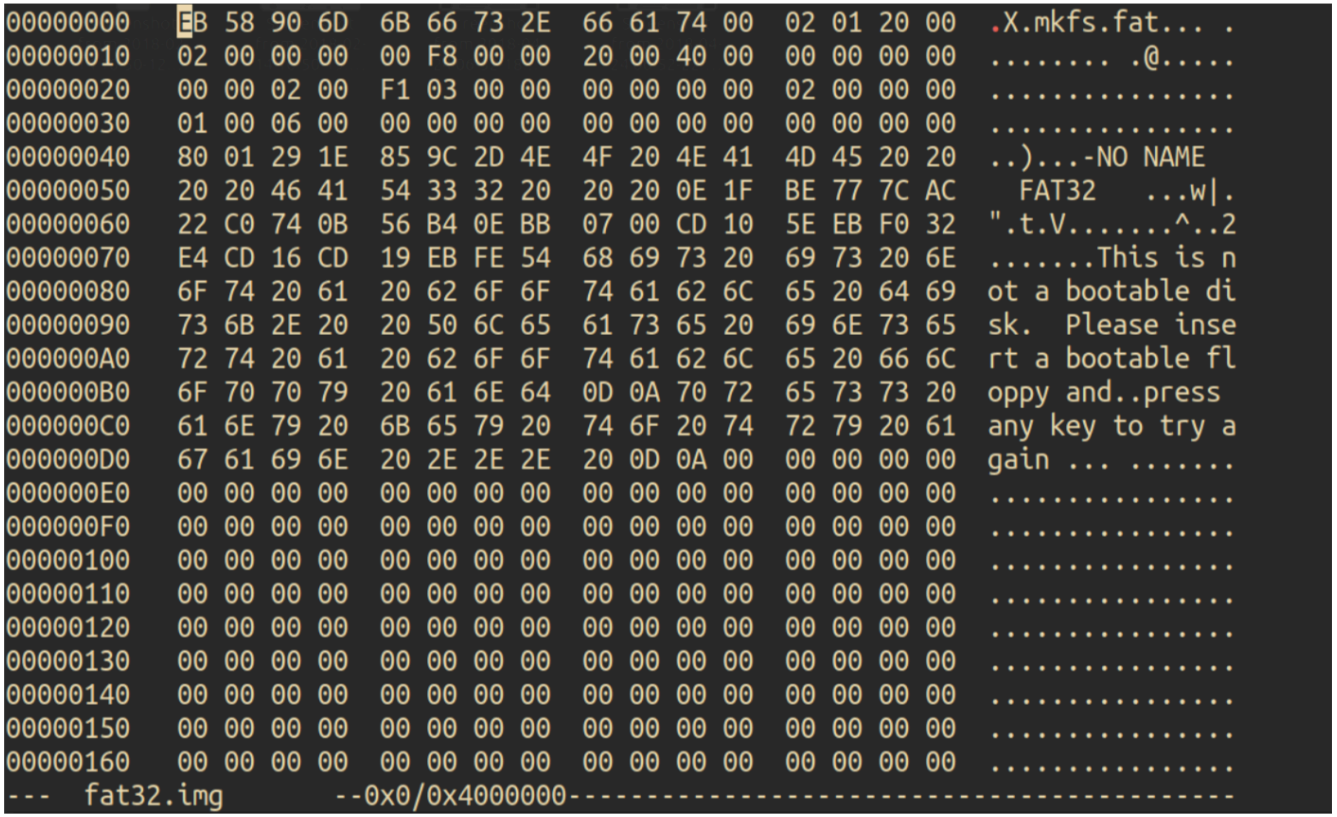
```
$ hexedit fat32.img
```

Hexedit

Starting byte offset for each line

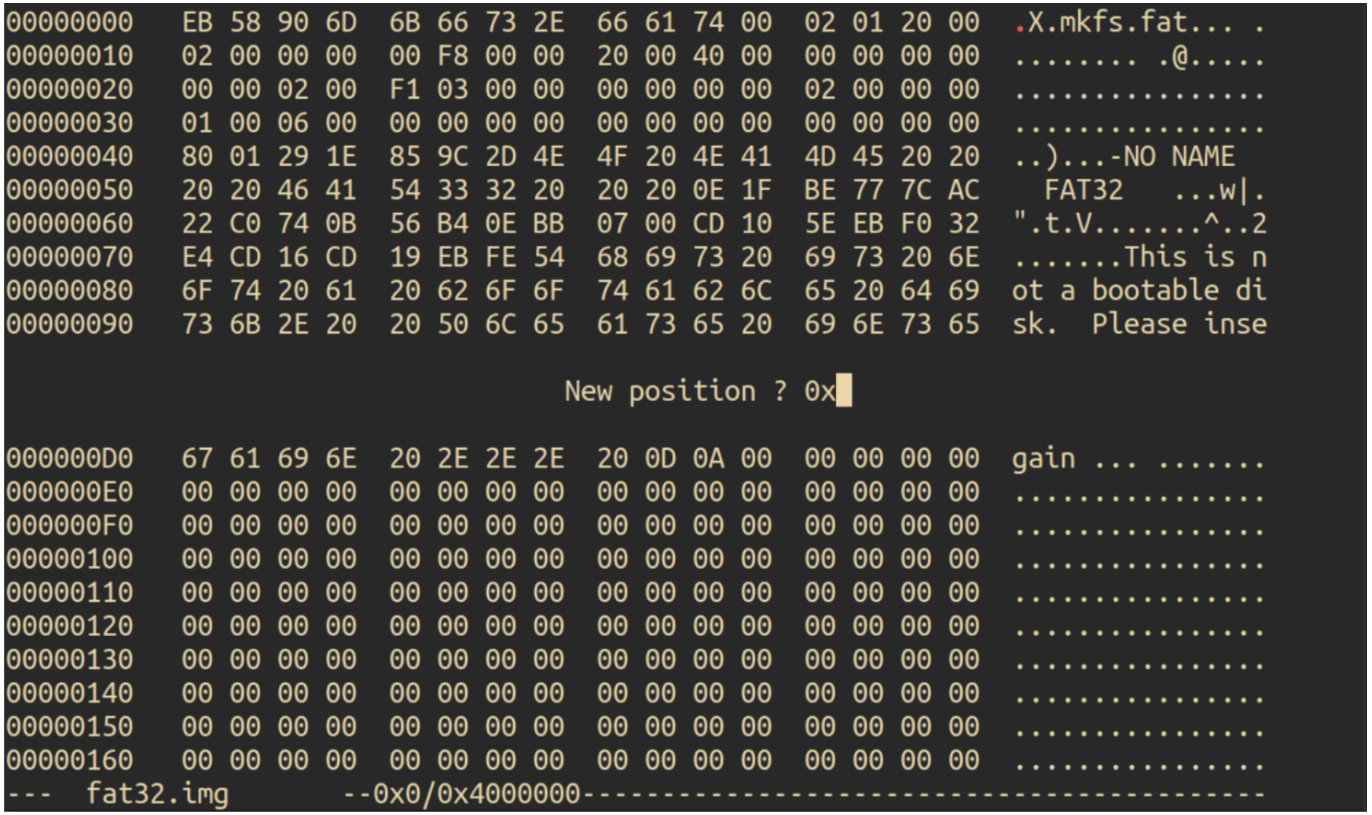
Data bytes

ASCII representation

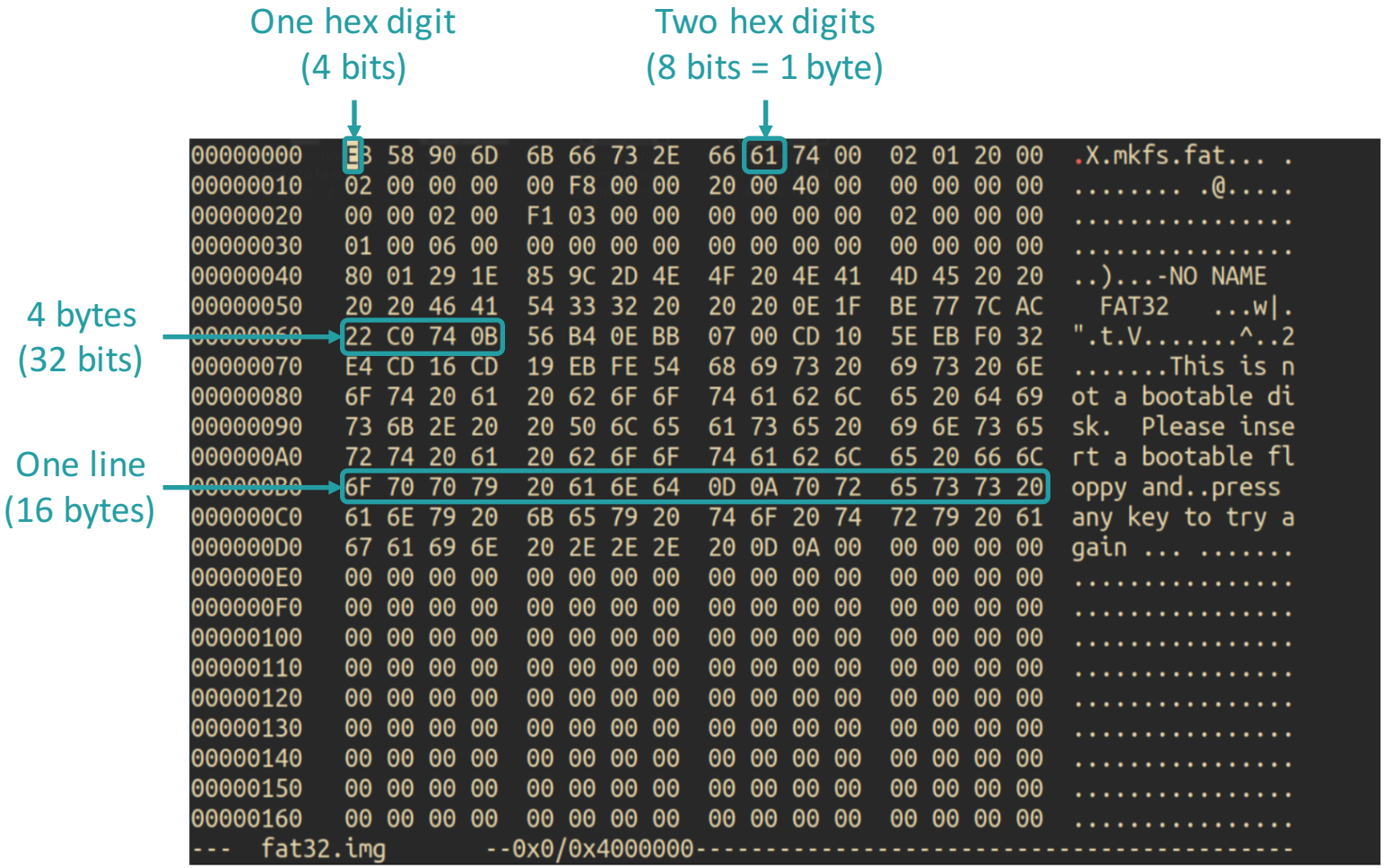


Hexedit

- Use `ctrl+g` or `F4` to jump to a new line



Hexedit


One hex digit
(4 bits)
Two hex digits
(8 bits = 1 byte)
4 bytes
(32 bits)
One line
(16 bytes)
00000000 EB 58 90 6D 6B 66 73 2E 66 61 74 00 02 01 20 00 .X.mkfs.fat... .
00000010 02 00 00 00 00 F8 00 00 20 00 40 00 00 00 00 00 ....@.....
00000020 00 00 02 00 F1 03 00 00 00 00 00 00 02 00 00 00 ....
00000030 01 00 06 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
00000040 80 01 29 1E 85 9C 2D 4E 4F 20 4E 41 4D 45 20 20 ..)...-NO NAME
00000050 20 20 46 41 54 33 32 20 20 20 0E 1F BE 77 7C AC FAT32 ...w|.
00000060 22 C0 74 0B 56 B4 0E BB 07 00 CD 10 5E EB F0 32 ".t.V.......^..2
00000070 E4 CD 16 CD 19 EB FE 54 68 69 73 20 69 73 20 6E ....This is n
00000080 6F 74 20 61 20 62 6F 6F 74 61 62 6C 65 20 64 69 ot a bootable di
00000090 73 6B 2E 20 20 50 6C 65 61 73 65 20 69 6E 73 65 sk. Please inse
000000A0 72 74 20 61 20 62 6F 6F 74 61 62 6C 65 20 66 6C rt a bootable fl
000000B0 6F 70 70 79 20 61 6E 64 0D 0A 70 72 65 73 73 20 oppy and..press
000000C0 61 6E 79 20 6B 65 79 20 74 6F 20 74 72 79 20 61 any key to try a
000000D0 67 61 69 6E 20 2E 2E 2E 20 0D 0A 00 00 00 00 00 gain ... .
000000E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
000000F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
00000100 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
00000110 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
00000120 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
00000130 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
00000140 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
00000150 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
00000160 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
--- fat32.img --0x0/0x4000000----------

Hexedit

- To make things easier to read, set each line to 16 bytes
- For further help, press F1
  - list of commands and their descriptions

FAT32 terms

- Byte (8 bits)
  - The smallest addressable unit in modern processors
- Sector
  - Smallest addressable unit on a storage device
- Cluster
  - FAT32 specific term
  - A group of sectors representing a chunk of data
- FAT (File Allocation Table)
  - Maps files to data

Byte/Sector/Cluster

FAT32 layout

- Boot sector
- Extended boot sector
- File system information sector
- Root cluster
- Other reserved sectors

- Actual data for files and directories

- FAT: a guide for traversing the data region
- Groups of clusters chained together

0 offset

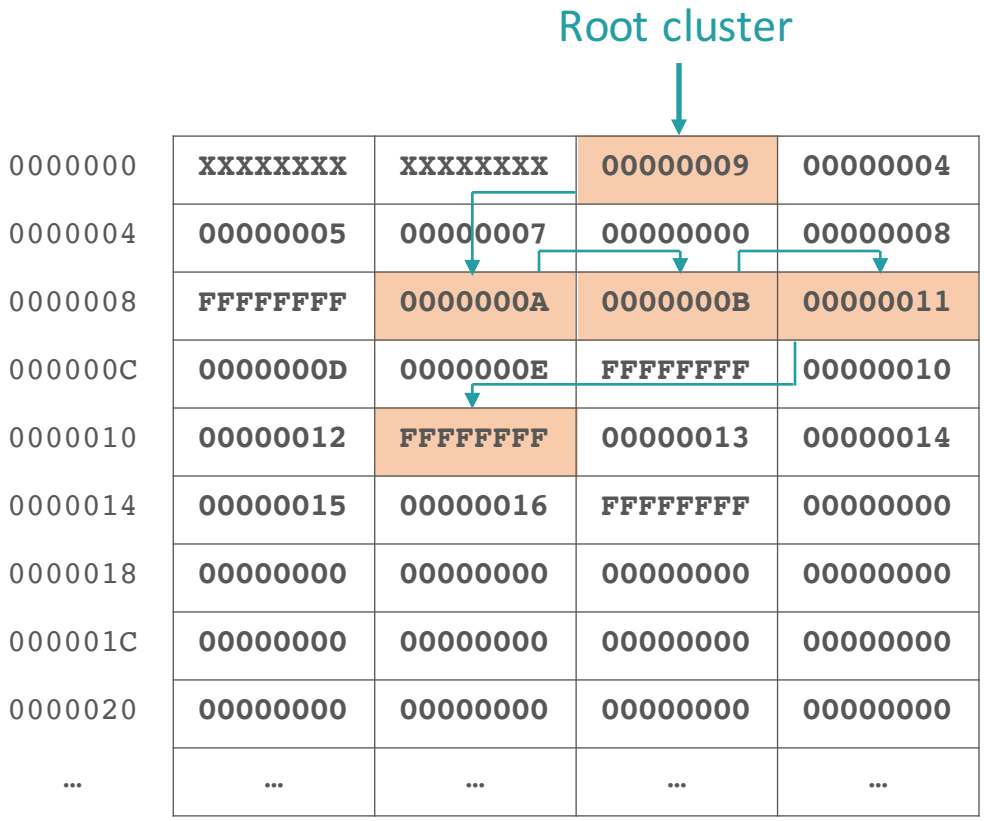
Reserved region

- Contains information about the file system itself
- When the file system is mounted, the OS reads the reserved region
  - Determines the type of file system
  - Characteristics
    - Size of clusters
    - Number of FATs
    - …

FAT (File Allocation Table)

- Maps the clusters of data
  - Cluster is the smallest addressable unit for the FAT
- Keeps track of which clusters belong to which files
- Need first cluster to access a file
  - Then find subsequent clusters by using the FAT

FAT



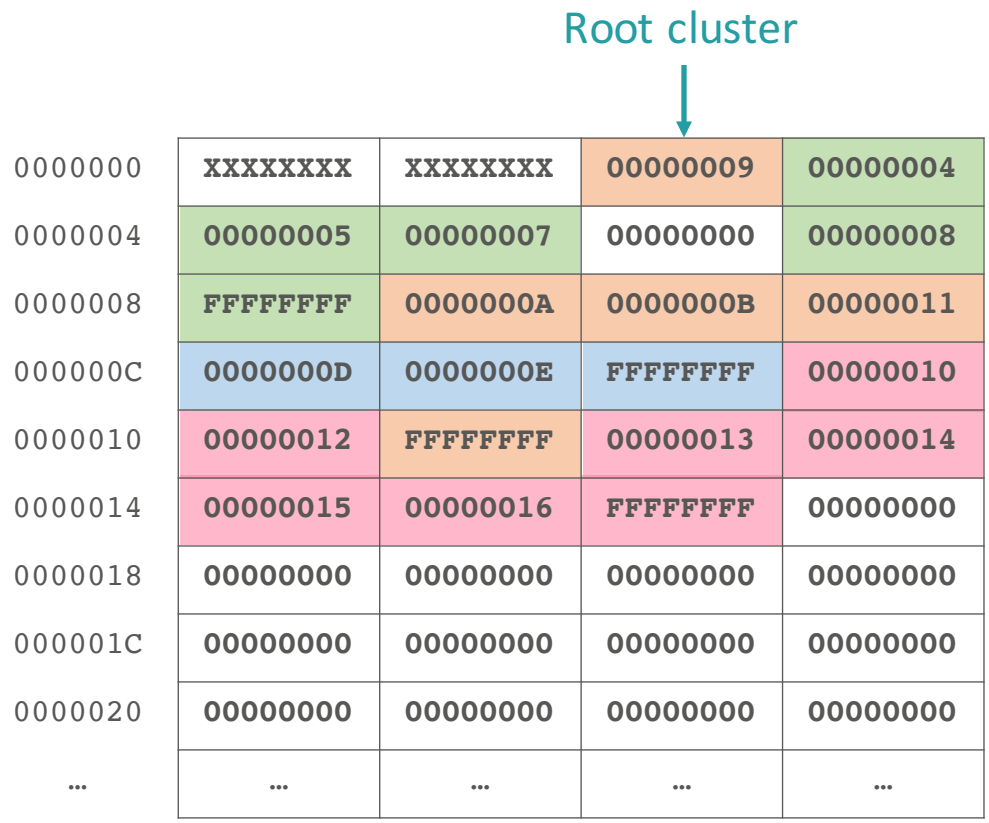
The root cluster is the first cluster for the root directory (noticed root cluster is 2?)

FAT entry indicates where the next cluster is located

- If 0x0: cluster not allocated
- If 0xFFFFFFFF*: last cluster
- Otherwise: indicates next cluster of the file/directory

*also check for values 0x0FFFFFF8 to 0x0FFFFFFE in the image, which indicate last cluster too.

FAT



Root directory

2 → 9 → A → B → 11

File 1

3 → 4 → 5 → 7 → 8

File 2

C → D → E

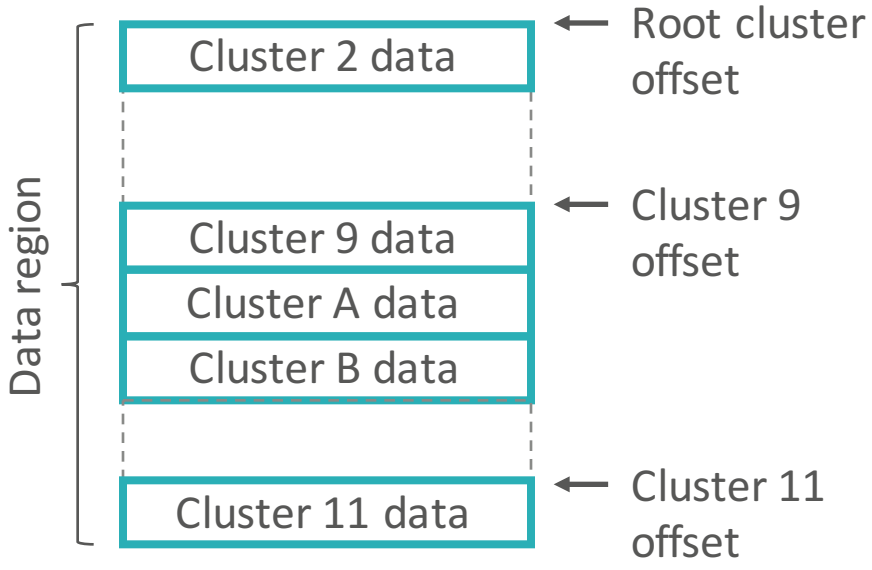
File 3

F → 10 → 12 → 13 → 14 → 15 → 16

FAT

- So what do the lists of clusters mean?
  - Indicate the offsets within the data region
  - Data for file (or directory) starts at first cluster's offset, then continues on second's, and so on.

E.g: root directory
Clusters:
2 → 9 → A → B → 11



Data region

- Two types of data
  - Files
  - Directories (folders)
- Data stored in clusters
  - Accessing the data requires:
    - Knowing first cluster
    - FAT

Directories

- Composed of directory (DIR) entries
- Directory data (data region)
  - List of DIR entries
  - Each element represents an item (file, subdirectory)
- E.g: directory `my_dir`
  - Contains `a.txt`, `b.txt`, `my_subdir`
    - `my_dir` has a DIR entry for each in data region
    - DIR entries can be all in the same cluster or distributed among different clusters.
    - DIR entries belong to only one cluster (not fragmented)

Directories

- DIR entry
  - Fixed-size data structures
    - Contains information about the entry
      - Name
      - Size
      - Attributes
      - **First cluster number**
      - …
    - Check FAT32 specs

Directories

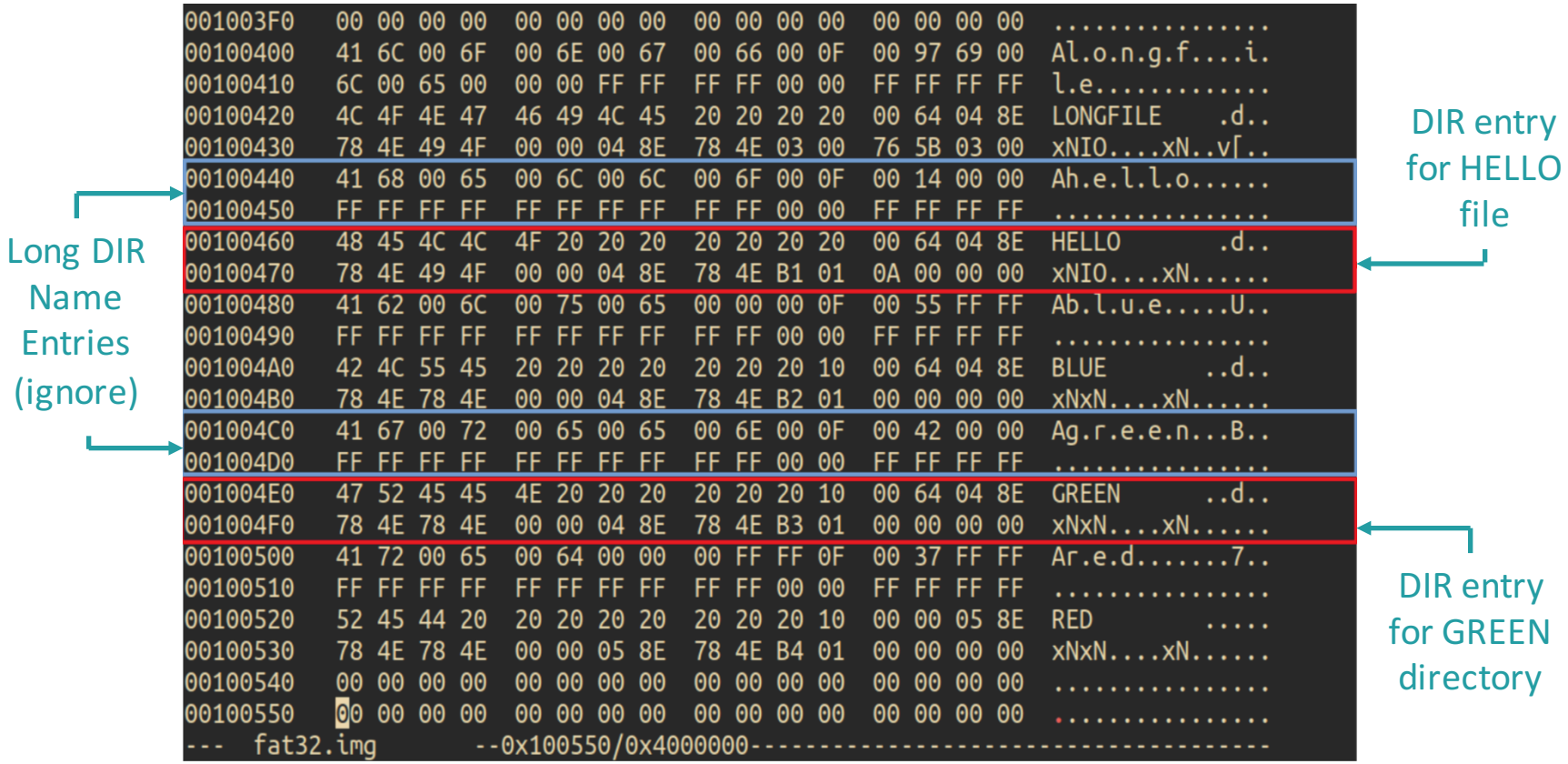
- DIR entry
  - Example:

```
/mnt/mnt_pnt $ ls
blue green hello longfile red
```

    - In the data region (for the root directory):
      - Entries for directories `blue`, `green` and `red`
      - Entries for files `hello` and `longfile`
  - Every directory (except root directory) will have entries for
    - `.` (current directory)
    - `..` (parent directory)

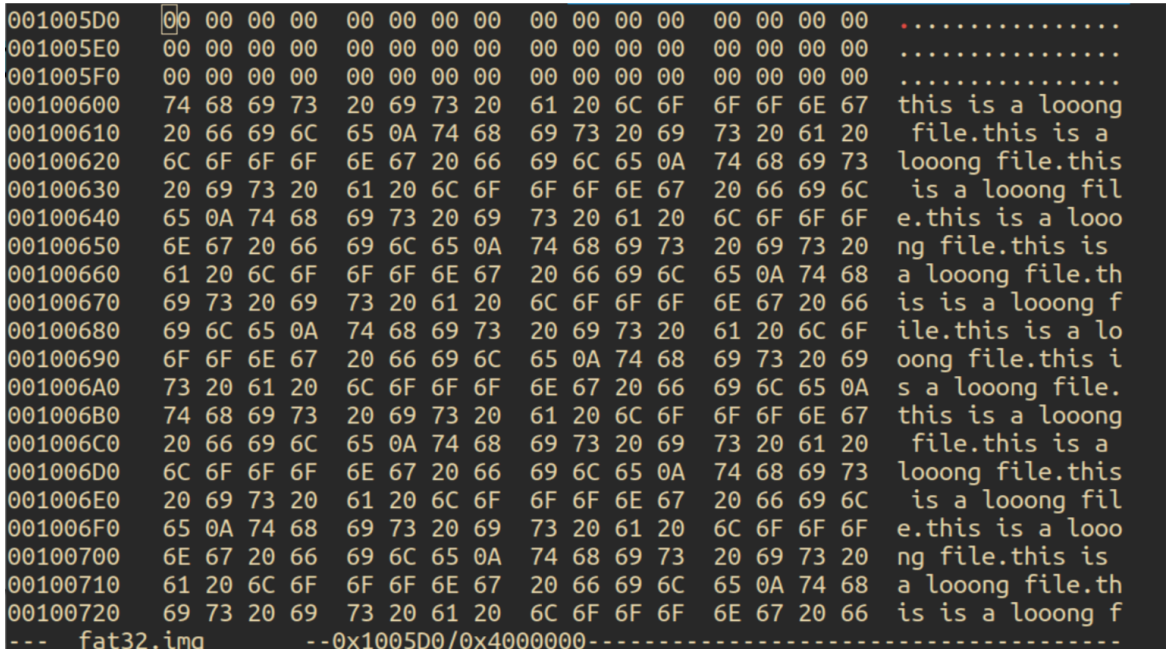
Directories

- DIR entries



Files

- Just the content of the files in the data region
  - Plain data. No DIR entries or similar
- File's clusters make up the whole file



Questions?