

Assignment 3: Introduction to File Systems

Introduction

- The goal of project 3 is to understand
 - basic file system design and implementation
 - file system testing
 - data serialization/de-serialization
- At the end of the project, you will feel like a file system expert!

Outline

- Background
 - Mounting file systems
- Project 3
 - Specification
 - Downloading and testing file system image
 - General FAT32 data structures
 - Endian-ness

Mounting File Systems

Unix File Hierarchy

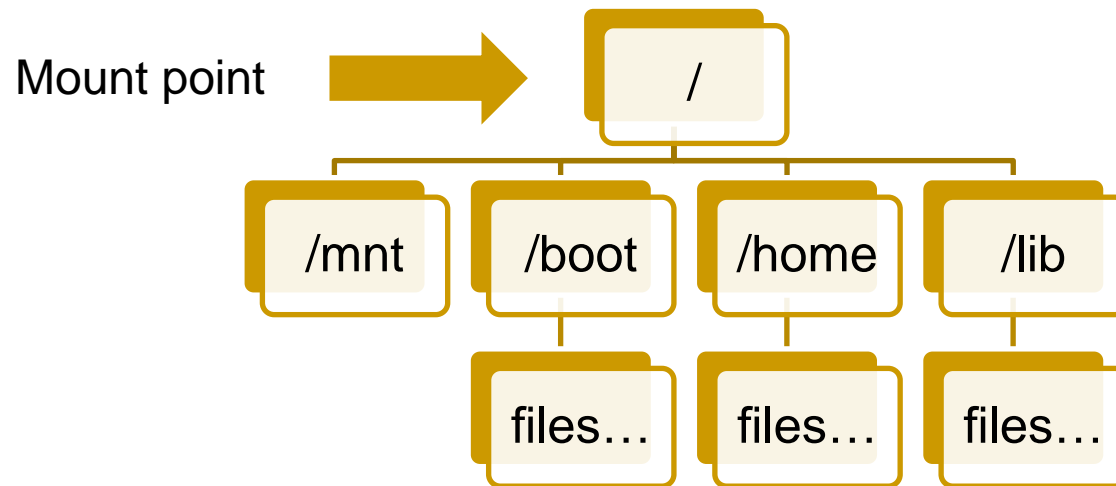
- All files accessible in a Unix system are arranged in one big tree
 - Also called the ***file hierarchy***
 - Tree is rooted (starts) at */*
- These files can be spread out over several devices
- The **mount** command serves to attach the file system found on some device to the big file tree

'mount' command

```
■ mount  
■ mount <device> <mount directory>
```

- Typing 'mount' without arguments shows you what is mounted and where
- Second example attaches a device or partition to a directory
 - Must have root privileges

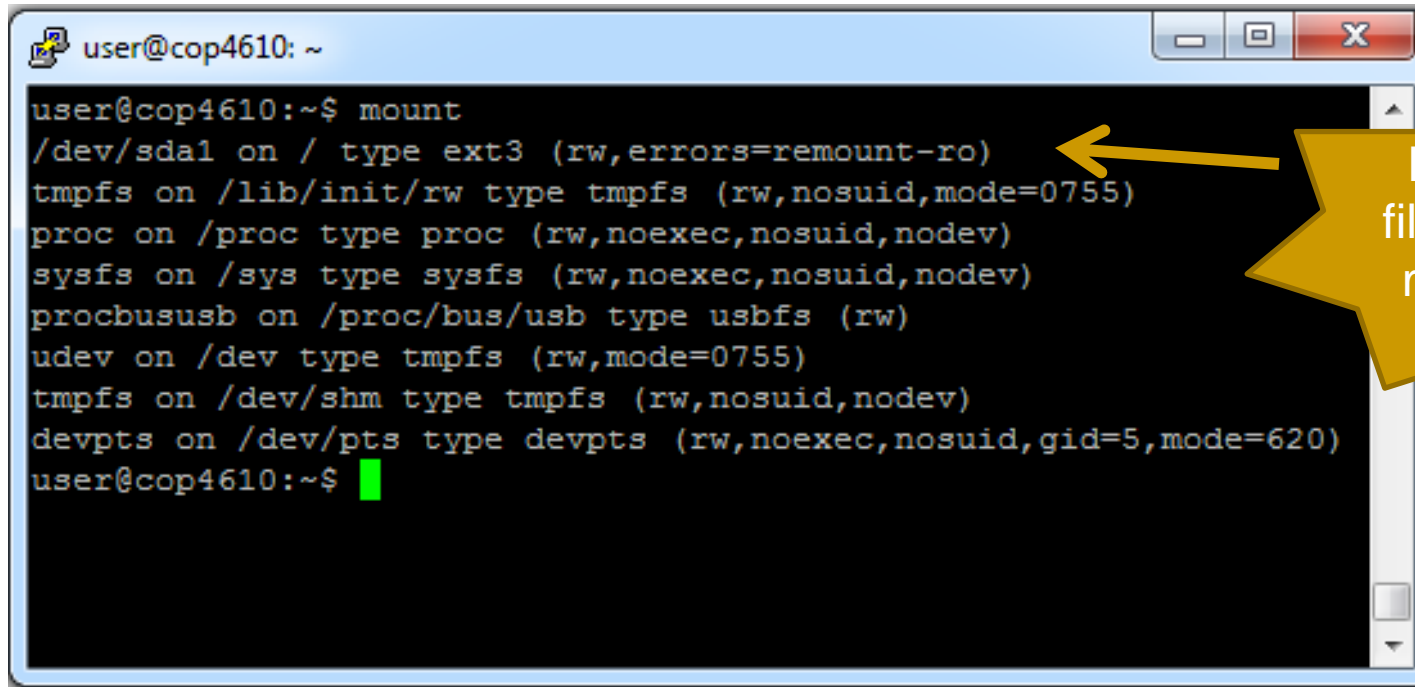
Mount Example



The device sda partition 1 is mounted at “/”. All files and dirs below “/” come from this device.

Mount Example

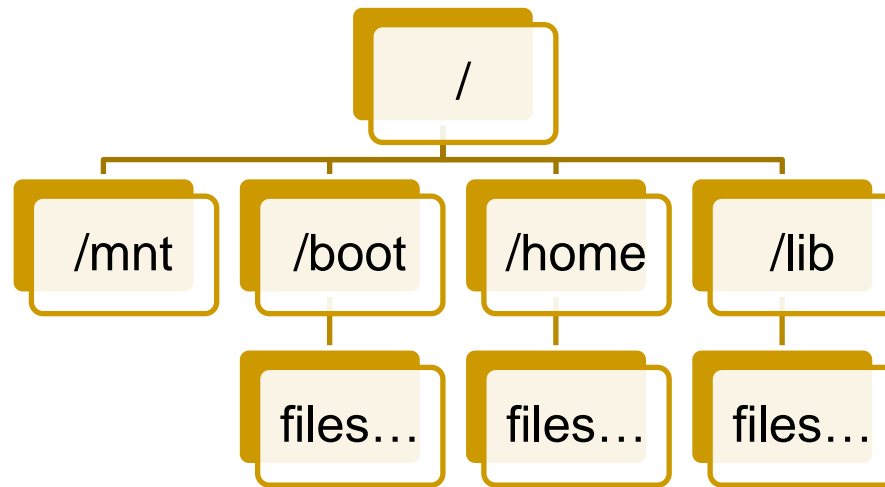
- Type command 'mount' without any arguments to see what is mounted and where



```
user@cop4610: ~  
user@cop4610:~$ mount  
/dev/sda1 on / type ext3 (rw,errors=remount-ro)  
tmpfs on /lib/init/rw type tmpfs (rw,nosuid,mode=0755)  
proc on /proc type proc (rw,noexec,nosuid,nodev)  
sysfs on /sys type sysfs (rw,noexec,nosuid,nodev)  
procbususb on /proc/bus/usb type usbfs (rw)  
udev on /dev type tmpfs (rw,mode=0755)  
tmpfs on /dev/shm type tmpfs (rw,nosuid,nodev)  
devpts on /dev/pts type devpts (rw,noexec,nosuid,gid=5,mode=620)  
user@cop4610:~$
```

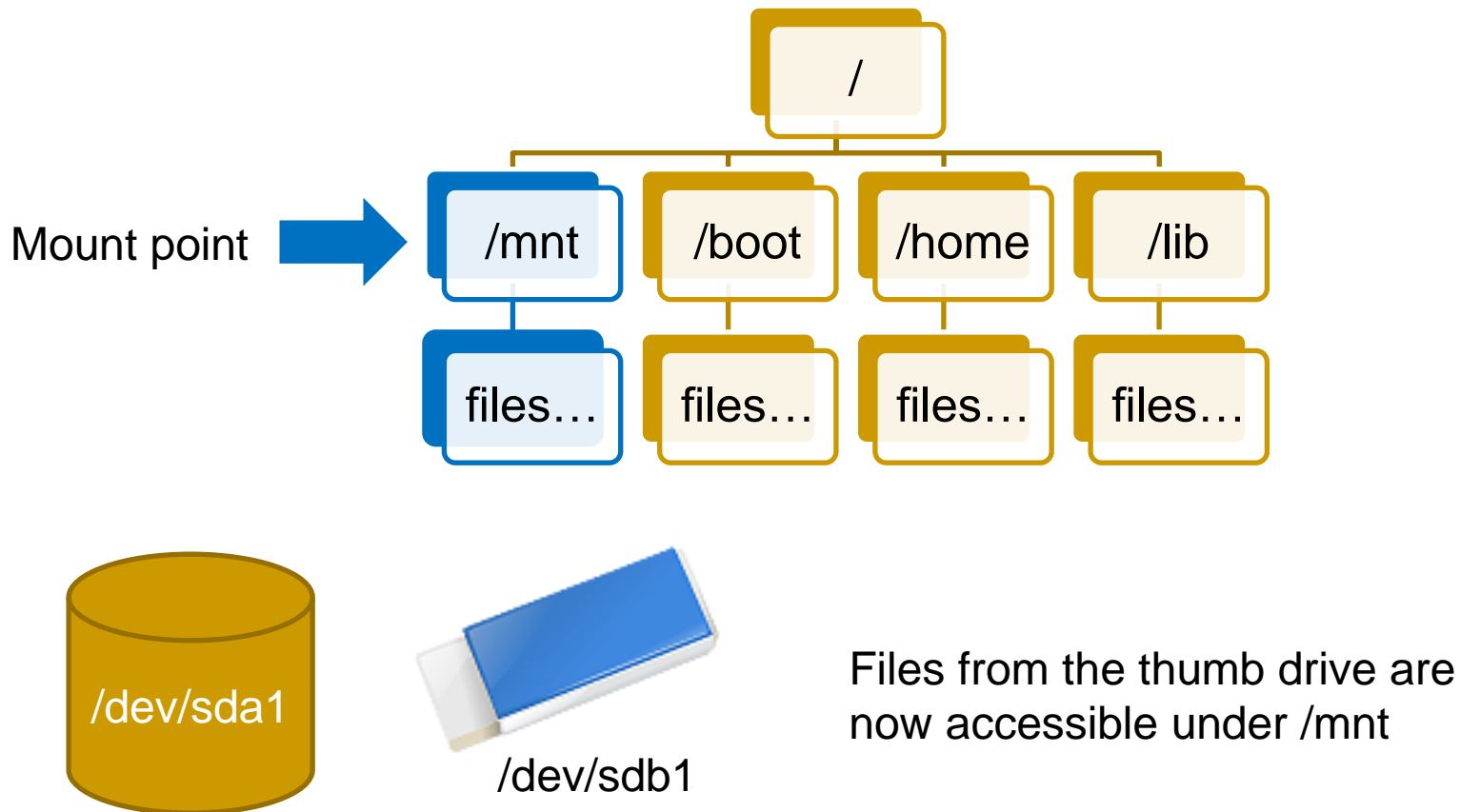
Root “/”
file system
mounted

Mount Example



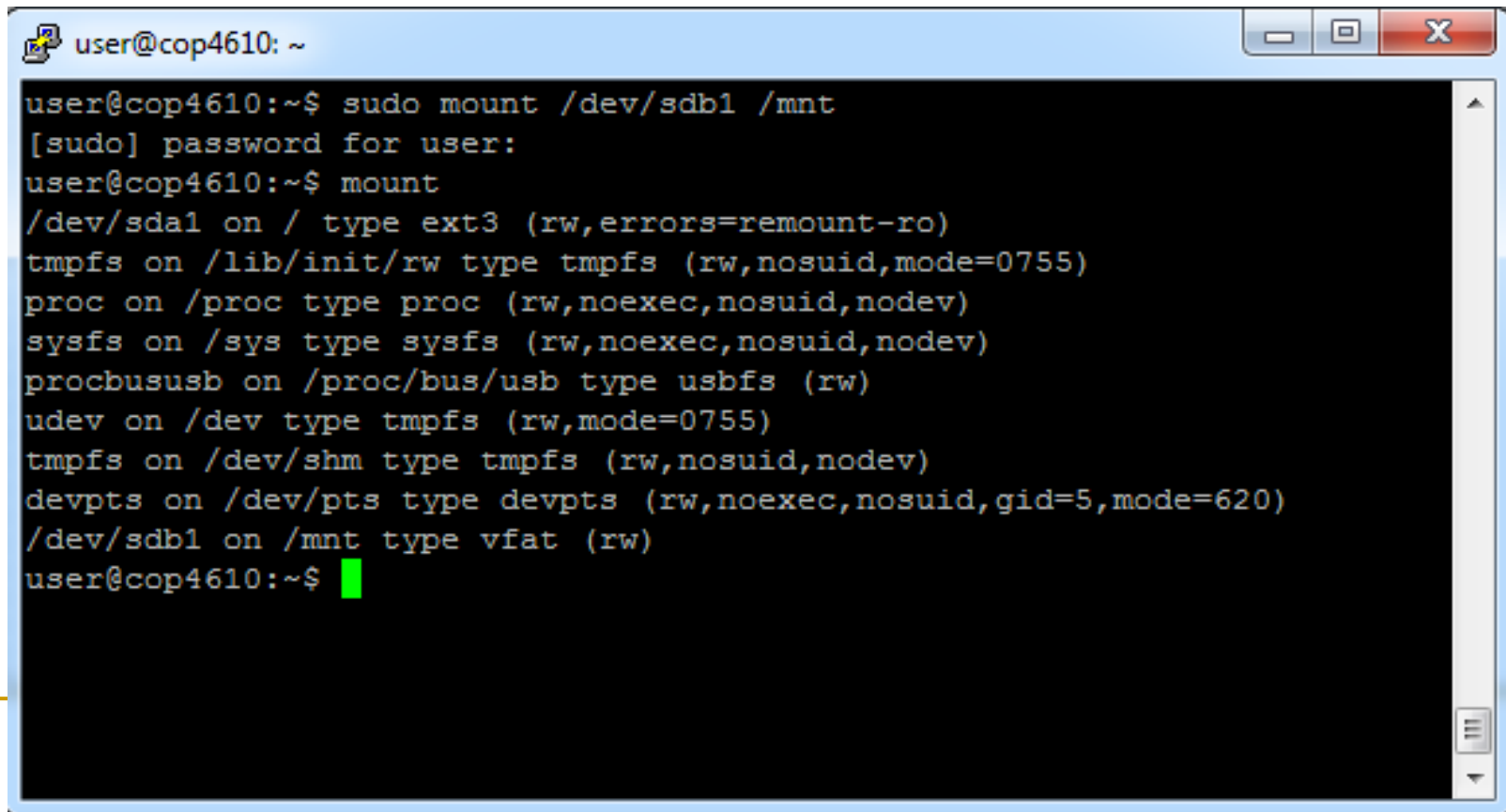
Now suppose we attach a thumb drive and want our thumb drive files accessible under `/mnt...`

File Hierarchy Example



Mount Example

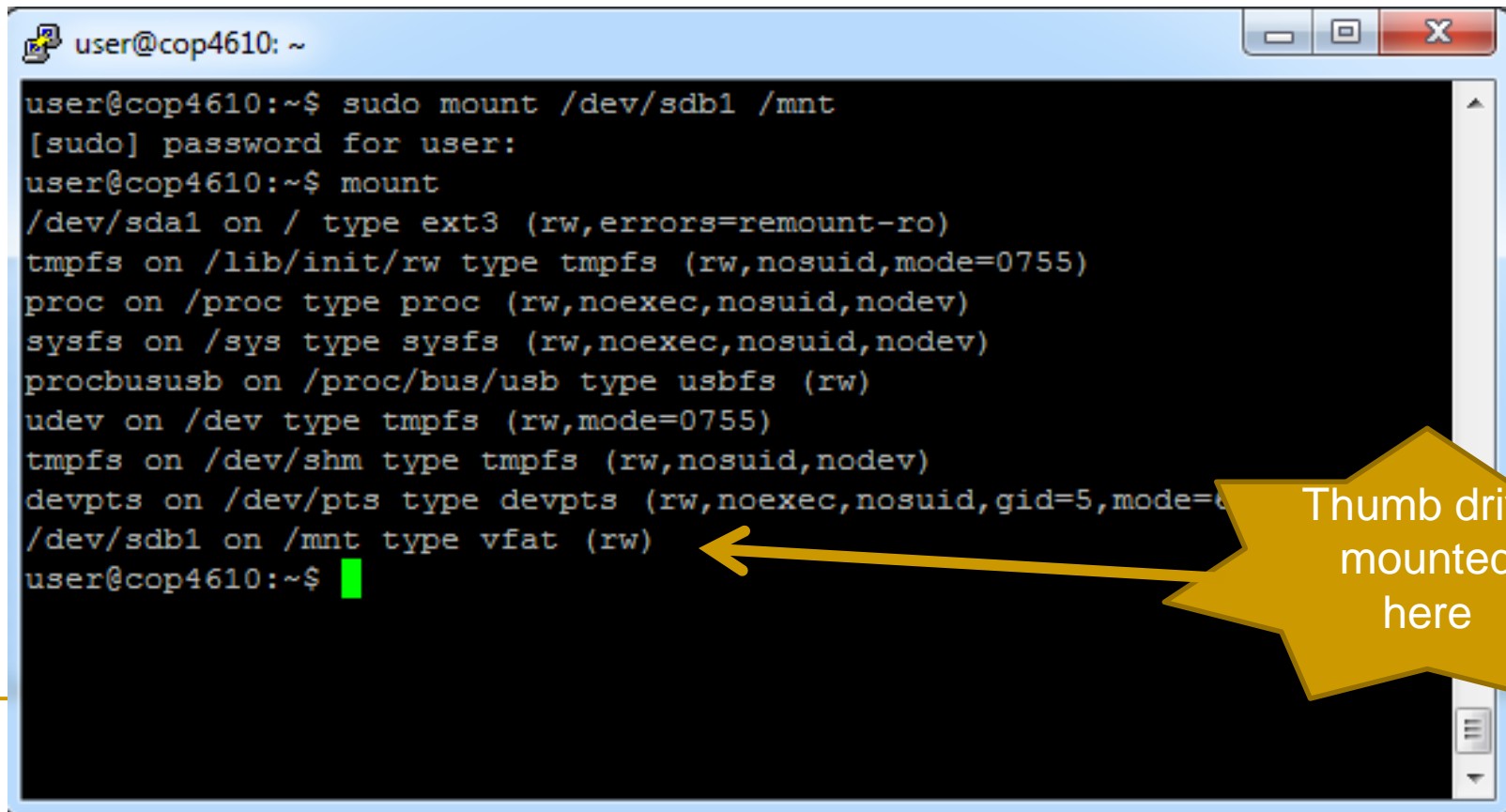
- The 'mount' command can dynamically attach new devices to new mount points

A terminal window titled 'user@cop4610: ~' with standard window controls (minimize, maximize, close). The terminal shows the following commands and output:

```
user@cop4610:~$ sudo mount /dev/sdb1 /mnt
[sudo] password for user:
user@cop4610:~$ mount
/dev/sda1 on / type ext3 (rw,errors=remount-ro)
tmpfs on /lib/init/rw type tmpfs (rw,nosuid,mode=0755)
proc on /proc type proc (rw,noexec,nosuid,nodev)
sysfs on /sys type sysfs (rw,noexec,nosuid,nodev)
procbususb on /proc/bus/usb type usbfs (rw)
udev on /dev type tmpfs (rw,mode=0755)
tmpfs on /dev/shm type tmpfs (rw,nosuid,nodev)
devpts on /dev/pts type devpts (rw,noexec,nosuid,gid=5,mode=620)
/dev/sdb1 on /mnt type vfat (rw)
user@cop4610:~$
```

Mount Example

- The 'mount' command can dynamically attach new devices to new mount points



```
user@cop4610: ~  
user@cop4610:~$ sudo mount /dev/sdb1 /mnt  
[sudo] password for user:  
user@cop4610:~$ mount  
/dev/sda1 on / type ext3 (rw,errors=remount-ro)  
tmpfs on /lib/init/rw type tmpfs (rw,nosuid,mode=0755)  
proc on /proc type proc (rw,noexec,nosuid,nodev)  
sysfs on /sys type sysfs (rw,noexec,nosuid,nodev)  
procbususb on /proc/bus/usb type usbfs (rw)  
udev on /dev type tmpfs (rw,mode=0755)  
tmpfs on /dev/shm type tmpfs (rw,nosuid,nodev)  
devpts on /dev/pts type devpts (rw,noexec,nosuid,gid=5,mode=0600)  
/dev/sdb1 on /mnt type vfat (rw)  
user@cop4610:~$
```

Thumb drive mounted here

Un-mount Command

- `umount <dir>`

- In our example where the thumb drive was mounted at `/mnt`, we can issue
 - `$> umount /mnt`
 - Must have root privileges
 - This means you can play with the mount command on your virtual machine, but not on the class server

Loopback File Systems

- A loopback file system is an image of a complete file system within a single file
- All the sector contents are concatenated in order within the file (one big char array).
- Linux can mount one of these like any other filesystem.
`mount -o loop disk1.iso /mnt/disk`
- You will use one of these for your project
- You will NOT be mounting it!!

Figuring out names of devices

- /etc/fstab – Has list of devices and file systems that get auto-mounted on boot
- 'dmesg' command shows output when plugging in a dynamic device

Project 3

More than you wanted to know about
FAT32..

Project 3

- You will create a user-space utility to manipulate a FAT32 file system image
 - No more kernel programming!
- Utility must understand a few basic commands to allow simple file system manipulation
- Utility must not corrupt the file system and should be robust

File System Image

- Manipulation utility will work on a pre-configured FAT32 ***file system image***
 - Actually a file
- File system image will have raw FAT32 data structures inside
 - Just like looking at the raw bytes inside of a disk partition

File System Image

- Your FAT32 manipulation utility will have to
 - Open the FAT32 file system image
 - Read parts of the FAT32 file system image and interpret the raw bytes inside to service your utility's file system commands...

...just like a file system!

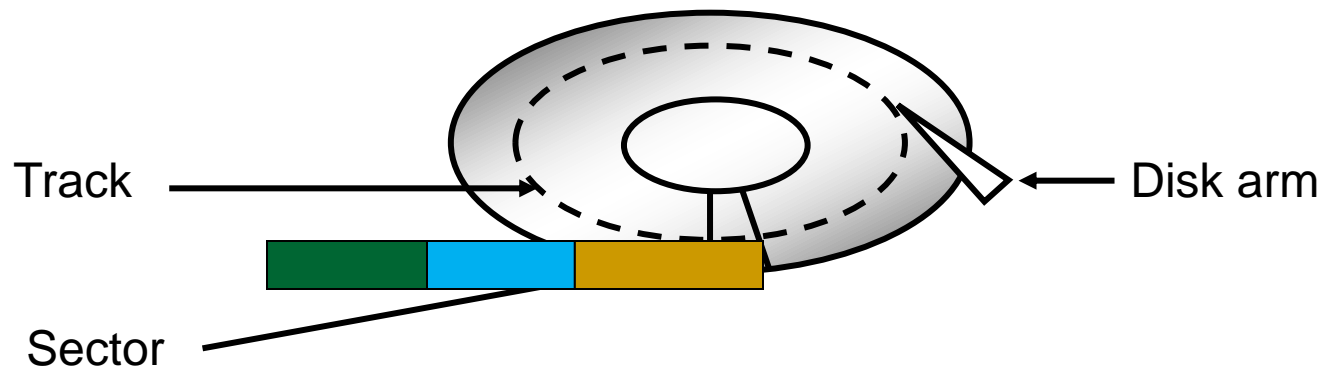
General FAT32 Data Structures

Terminology

- **Byte** – 8 bits of data, the smallest addressable unit in modern processors
- **Sector** – Smallest addressable unit on a storage device. Usually this is 512 bytes
- **Cluster** – FAT32-specific term. A group of sectors representing a chunk of data. Usually called a *block*.
- **FAT** – Stands for *file allocation table* and is a map of files to data

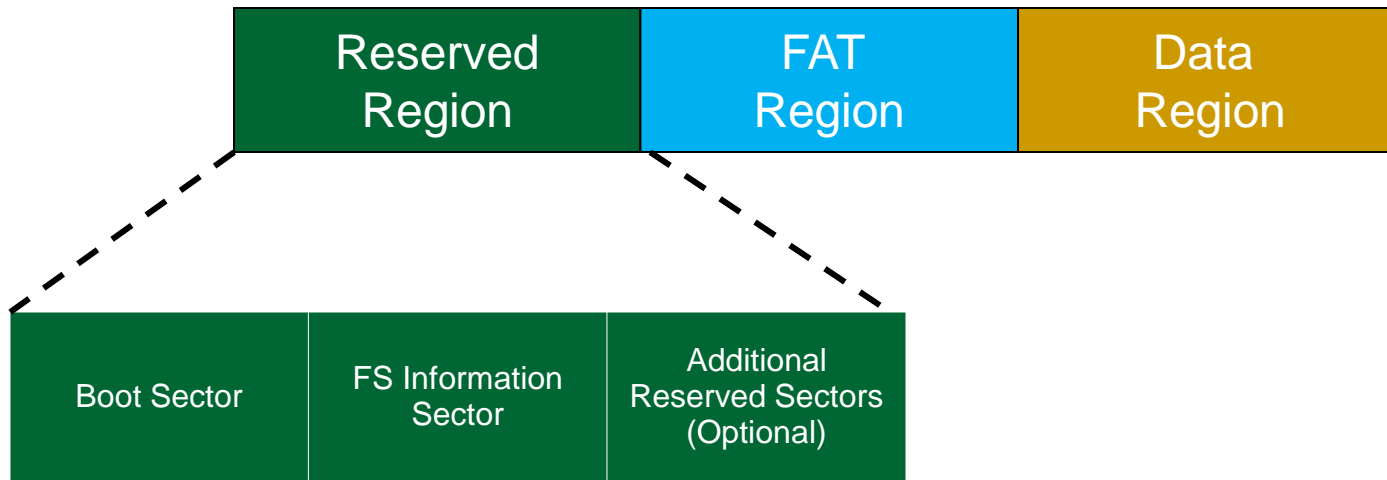
FAT32 Disk Layout

- 3 main regions...



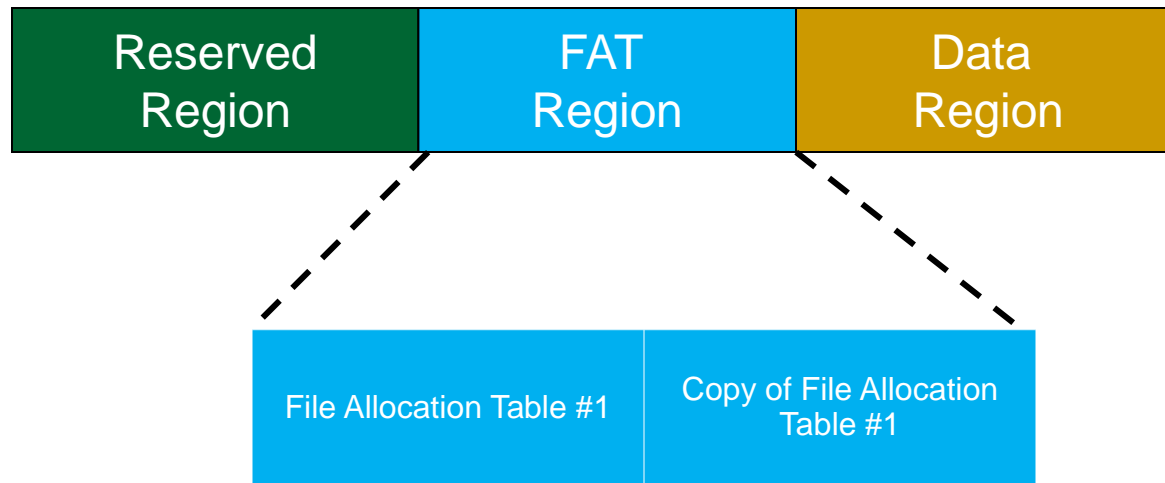
Reserved Region

- **Reserved Region** – Includes the boot sector, the extended boot sector, the file system information sector, and a few other reserved sectors



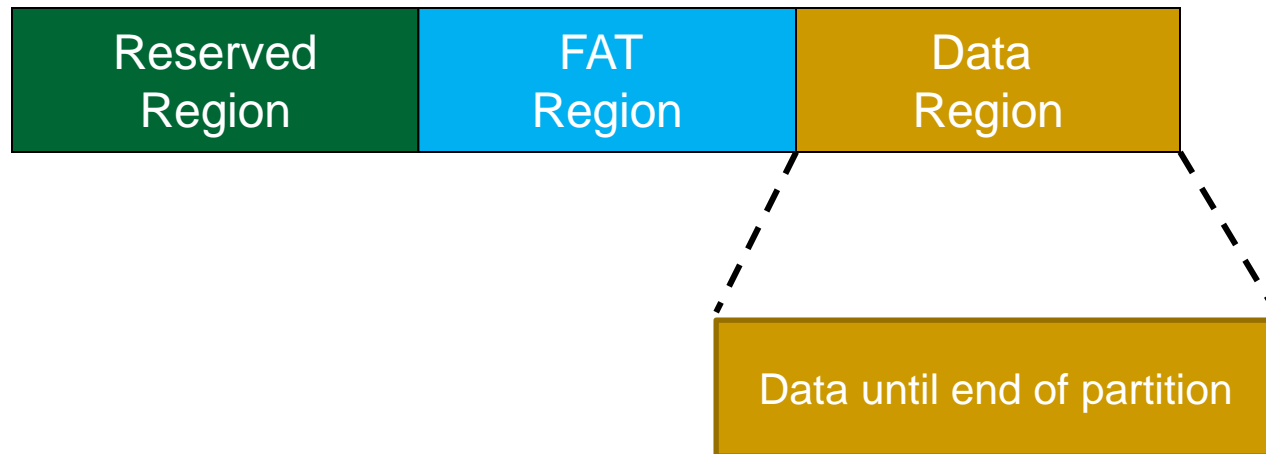
FAT Region

- **FAT Region** – A map used to traverse the data region. Contains mappings from cluster locations to cluster locations



Data Region

- **Data Region** – Using the addresses from the FAT region, contains actual file/directory data



Endian

Big or little?

Machine Endianness

- The endianness of a given machine determines in what order a group of bytes are handled (ints, shorts, long longs)
 - Big-endian – most significant byte first
 - Little-endian – least significant byte first
- This is important to understand for this project, since FAT32 is always formatted as little-endian

FAT32 Endianness

- The following are a few cases where endianness matters in your project:
 - ❑ Reading in integral values from the FAT32 image
 - ❑ Reading in shorts from a FAT32 image
 - ❑ Combining multiple shorts to form a single integer from the FAT32 image
 - ❑ Interpreting directory entry attributes

Visualizing Endianness

Value = 13371337 (0x00CC07C9)

index	0	1	2	3
little endian	0xC9	0x07	0xCC	0x00
big endian	0x00	0xCC	0x07	0xC9

Next Steps

- Take a look at the fat32 specification file from Microsoft
 - Somewhat confusing – just like the real world
- Take a look at the image file
 - You *may* mount it in Linux, but that doesn't contribute to your project