Arizona State University

CSE 469: Computer and Network Forensics

Topic 3: Drives, Volumes, and Files

01110111 01100101 01101100 01100011 01101111
01101101 01100101 00100001 00001010

https://www.youtube.com/watch?v=zmNFgM71HgU

0x77 0x65 0x6c 0x63 0x6f 0x6d 0x65 0x21 0x0a

https://onlineasciitools.com/convert-ascii-to-hexadecimal

Dr. Jaejong Baek | Spring 2020

Review: Base Conversion, Endianness, and Data Structures

Converting Between Bases

- Decimal Number: 35,812

	10,000 (10^4)	1,000 (10^3)	100 (10^2)	10 (10^1)	1 (10^0)
Denary	3	5	8	1	2

- Binary Number: 1001 0011

	128 (2^7)	64 (2^6)	32 (2^5)	16 (2^4)	8 (2^3)	4 (2^2)	2 (2^1)	1 (2^0)
Bit weight								
Binary	1	0	0	1	0	0	1	1
Bit position label	MSB							LSB

MSB: most significant bit, high-order bit, left-most bit

LSB: Least Significant Bit, low-order bit, right-most bit

Converting Between Bases

- Hexadecimal Number: 0x8BE4

4,096 (16^3)	256 (16^2)	16 (16^1)	1 (16^0)
8	11	14	**4**
MSB			LSB

- 0xB = 11
- 0xE = 14

Binary and Hexadecimal

- 1001 0100 to Hexadecimal

1001	0100

↓

0x9	0x4

=148

- 0x98 to binary

0x9	0x8

↓

1001	1000

=152

Analog Example: Data Structure

- Paper form

SUN Card Application

Please fill out the following form

Name:

Address:

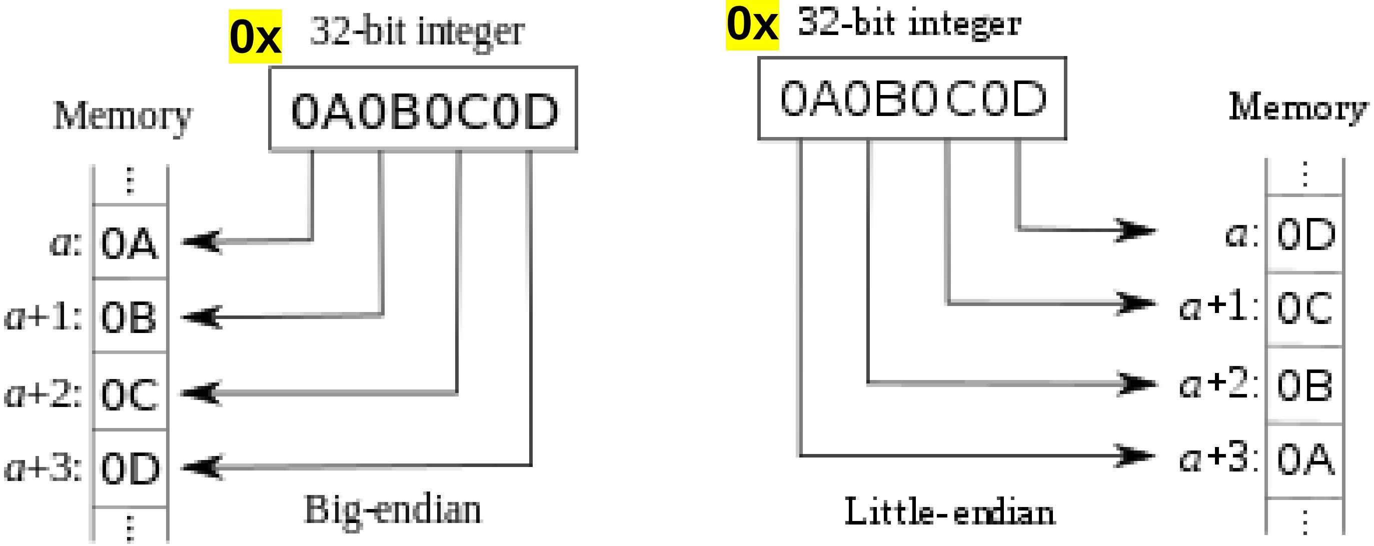
...

Data Structures: Considerations

- Data Size
  - Need to **allocate** a location on a storage device.
  - A **byte** can hold only **256** values.
    - Byte = 8 bits = 2^8 = 256
    - The smallest amount of data we'll work with.
- Organizing multiple-byte values:
  - Big-endian ordering.
  - Little-endian ordering.

Endianness (1/2)

- Refers to the sequential order in which bytes are arranged into larger numerical values when stored in memory or when transmitted over digital links.



https://en.wikipedia.org/wiki/Endianness

Endianness (2/2)

- Big-endian ordering:
  - Puts the **Most Significant Byte** of the number in the **first** storage byte.
  - Sun SPARC, Motorola Power PC, ARM, MISP.

- Little-endian ordering:
  - Puts the **Least Significant Byte** of the number in the **first** storage byte.
  - IA32-based systems.

ASU Arizona State University

Endianness: Example 1

Actual Value: 0x12345678 (4 Bytes)

- Big-endian ordering

23	24	25	26	27	28
00	**12**	**34**	**56**	**78**	00

- Little-endian ordering

23	24	25	26	27	28
00	**78**	**56**	**34**	**12**	00

Endianness: Example 2

- 0xFF00AA11

Little Endian	
Address	Contents
4003	FF
4002	00
4001	AA
4000	11

Big Endian	
Address	Contents
4003	11
4002	AA
4001	00
4000	FF

https://chortle.ccsu.edu/AssemblyTutorial/Chapter-15/ass15_4.html

Endianness and Strings

- Does Endianness affect letters and sentences?
  - The most common techniques is to encode the characters using ASCII and Unicode.
  - ASCII:
    - In Hexadecimal, 0x00 Through 0x7F.
    - Including control characters (0x07 – Bell Sound).
    - 1 byte per character.
    - The endian ordering does not play a role since each byte stores the value of a character.
    - Many times, the string ends with the NULL character (0x00).

ASCII Example

String: 1 Main St.

23	24	25	26	27	28	29	30	31	32	33
31	**20**	**4D**	**61**	**69**	**6E**	**20**	**53**	**74**	**2E**	**00**
1		M	a	i	n		S	t	.	

https://en.wikipedia.org/wiki/ASCII

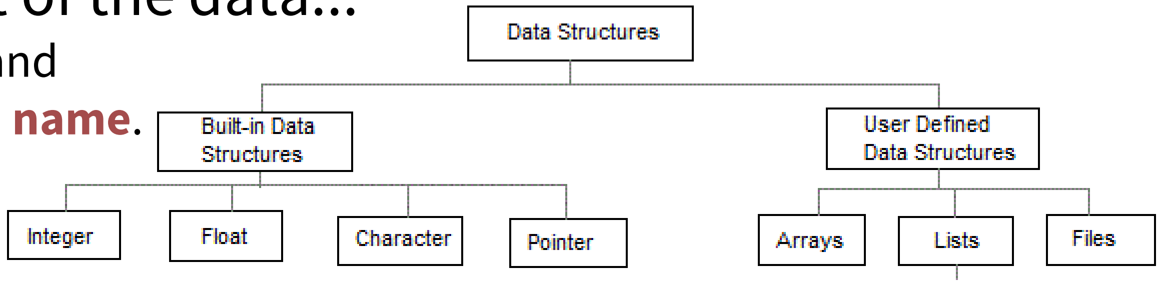
Unicode

- Version 12.1 (May 2019) supports 137,994 characters.
  - Covers 163 modern and historic scripts, as well as multiple symbol sets and emoji.
- 4-bytes per character: U+0044 = D
- Three methods:
  - UTF-32 – uses a 4-byte value for each character.
  - UTF-16 – stores the most heavily used characters in a 2-byte value and the lesser-used characters in a 4-byte value.
  - UTF-8 – uses 1, 2, or 4 bytes to store a character and the most frequently used bytes use only 1 byte.
- Different methods make different tradeoffs between processing overhead and usability.

https://en.wikipedia.org/wiki/Unicode

Data Structures

- Describes the layout of the data...
  - broken up into **fields** and
  - each field has **size** and **name**.



- Write operation:
  - Refer to the appropriate data structure to determine **where** each value should be written.

- Read operation
  - Need to determine **where the data starts** and then refer to its data structure to find out **where the needed values are** (offset from the start).

Data Structure: Example

Byte Range	Description
0-1	2-byte house number
2-31	30-byte ASCII street name

```
0000000: 0100 4d61 696e 2053 742e 0000 0000 0000   ..Main St....
0000016: 0000 0000 0000 0000 0000 0000 0000 0000   ..............
0000032: bb02 536f 7574 6820 4d69 6c6c 4176 652e   ??
0000048: 0000 0000 0000 0000 0000 0000 0000 0000
```

The byte offset in decimal

16 bytes of the data in hexadecimal

ASCII equivalent

Data structures are important!!

Data Structure: Example 2

Byte Range	Description
0-3	4-byte house number
4-31	28-byte ASCII street name

```
0000000: 0100 0000 4d61 696e 2053 742e 0000 0000
0000016: 0000 0000 0000 0000 0000 0000 0000 0000
0000032: 0000 0100 4d61 6265 7920 5761 7900 0000
0000048: 0000 0000 0000 0000 0000 0000 0000 0000
```

Record 1:

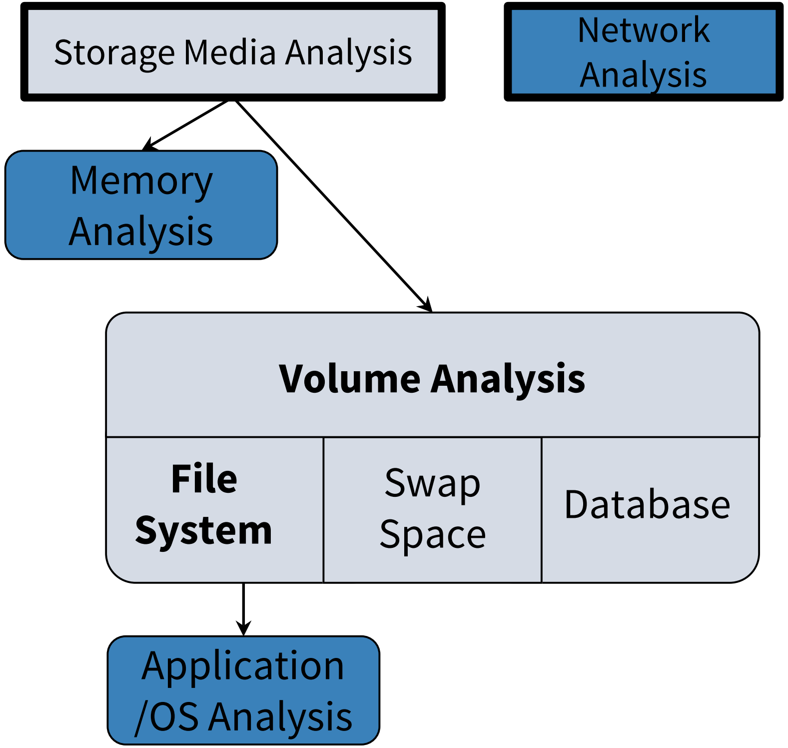
House number: ___________ Street name:____

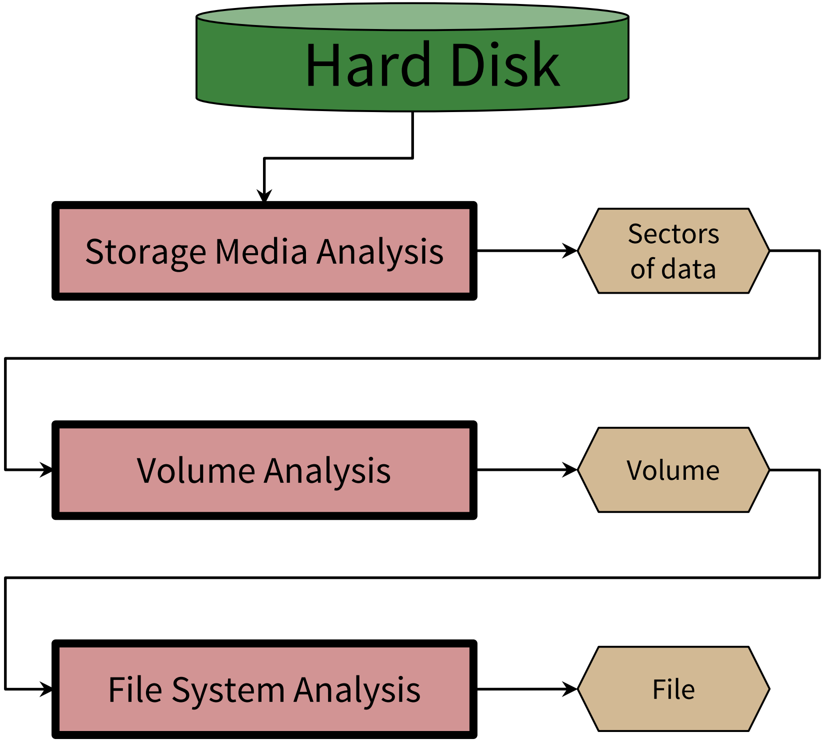
Record 2:

House number: ___________ Street name:____

Layers of Forensic Analysis

Layers of Forensic Analysis





Layers of Analysis (1)

- Storage media (Physical) layer analysis:
  - Non volatile storage such as hard disks and flash cards.
  - Organized into partitions / volumes:
    - Collection of ***storage locations*** that a user or application can write to and read from.
    - Contents are file system, a database, or a temporary swap space.

- Volume layer analysis:
  - Analyze data at the volume (logical drive) level.
  - Determine ***where*** the file system or other data are located.
  - Determine ***where*** we may find hidden data.

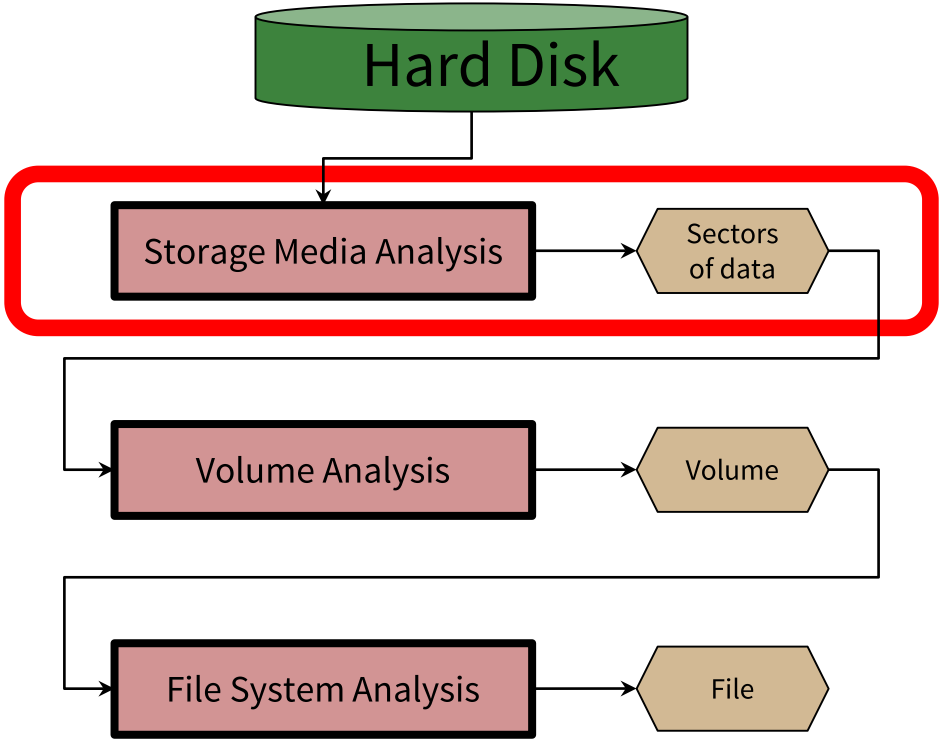
Physical disk	Partition	Filesystem	Drive letter
Hard Disk 1	Partition 1	NTFS	C:
	Partition 2	FAT32	D:
Hard Disk 2	Partition 1	FAT32	E:

Layers of Analysis (2)

- File system layer analysis:
  - A collection of ***data structures*** that allow an application to create, read, and write files.
  - Purpose: To find files, to recover deleted files, and to find hidden data.
  - The result could be ***file content***, ***data fragments***, and ***metadata*** associated with files.

- Application layer analysis:
  - The structure of each file is based on the application or OS that created the file.
  - Purpose: To ***analyze files*** and to determine ***what program we should use***.

Disk Drive Geometry


Hard Disk
Storage Media Analysis
Sectors of data
Volume Analysis
Volume
File System Analysis
File

Storage Media Analysis

- Hard Disk Geometry
  - Head: The device that reads and writes data to a drive.
  - Track: Concentric circles on a disk platter.
  - Cylinder: A column of tracks on disk platters.
  - Sector: A section on a track.

Inside a Hard Drive

Head

Disk Platter

Head Actuator

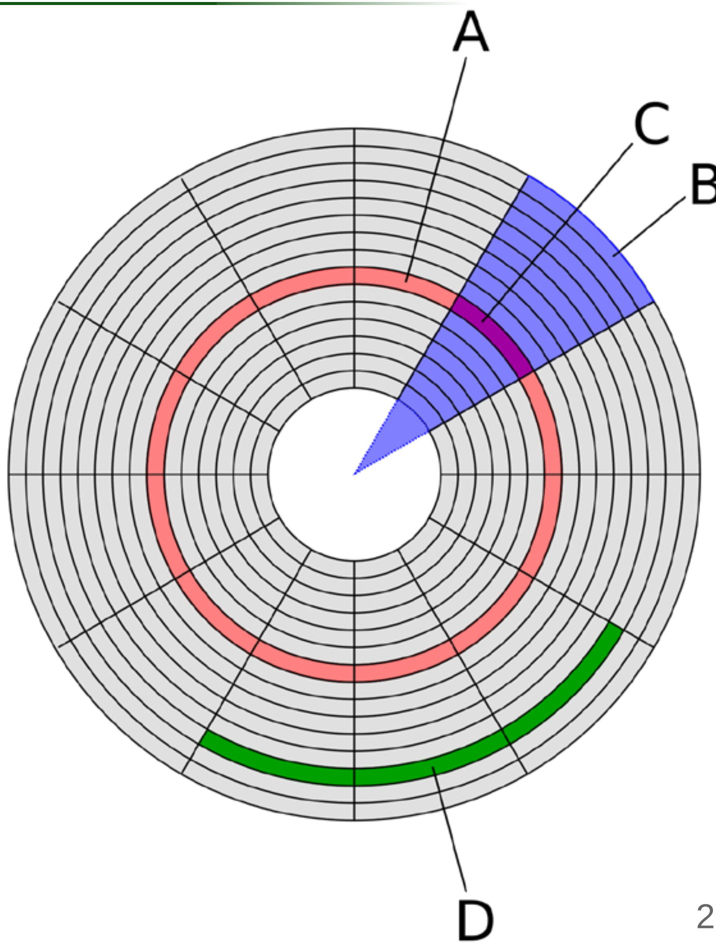
Head Arm

Chassis

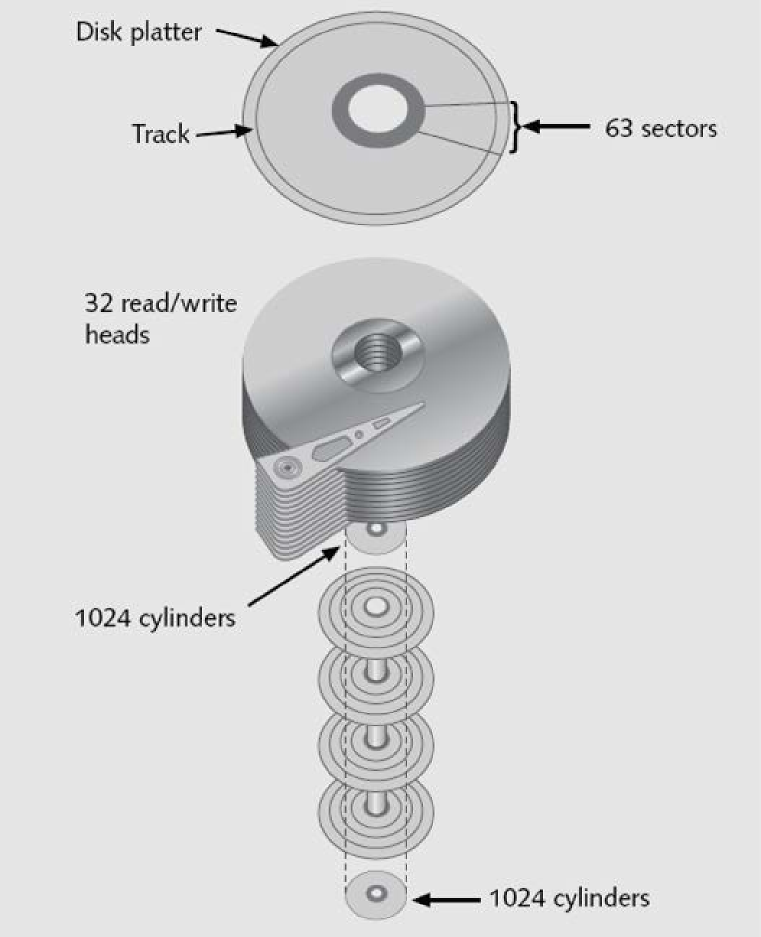
https://www.youtube.com/watch?v=BlB49F6ExkQ

Tracks, Sectors, and Clusters

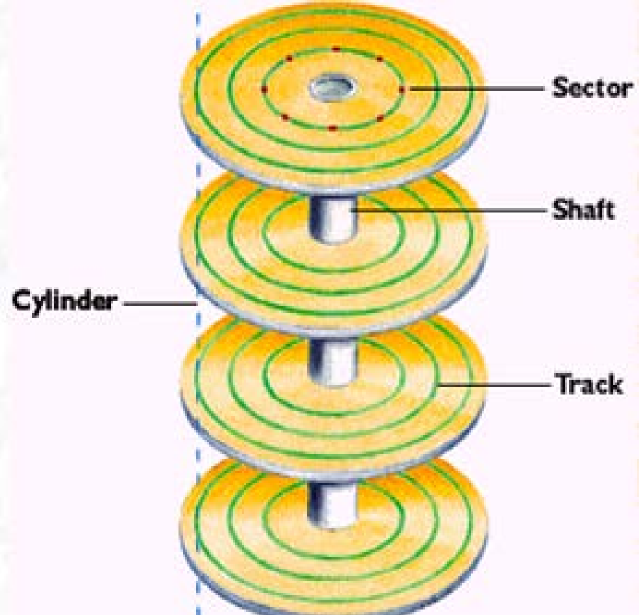
- Platters are divided into concentric rings called ***tracks*** (A).
- Tracks are divided into wedge-shaped areas called ***sectors*** (C).
  - A sector typically holds 512 bytes of data.
  - A collection of sectors is called a ***cluster*** or ***block*** (D).
- (B) is apparently called a *geometrical sector* (uncommon).



Cylinders

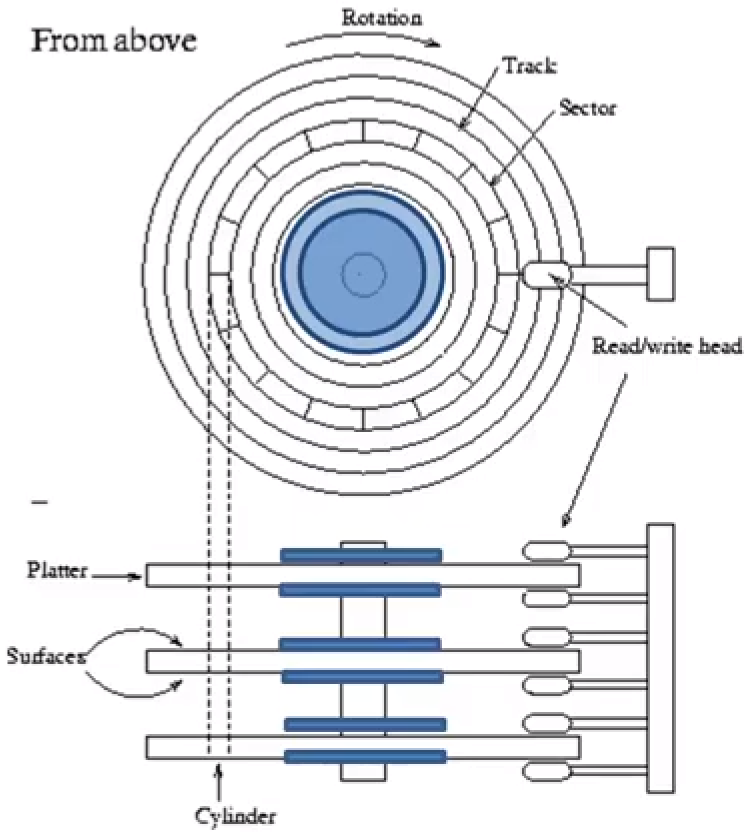


- A ***cylinder*** is a vertical set of similar tracks for all platters



CHS Addresses

- ***Cylinders/Tracks***: Numbered from the outside in, **starting at 0-1023**.
  - All sectors of all tracks in cylinder 0 will be filled up before using cylinder 1.
- ***Heads***: Numbered from the bottom up, **starting at 0-254**.
  - All platters are double-sided, one head per side.
- ***Sectors***: Each sector is numbered, **starting at 1-64**.
  - 1 sector typically holds 512 bytes of data.
- First sector has CHS address: **0,0,1**



https://www.youtube.com/watch?v=tDRuA_6Go8c&t=484s

Logical Block Address (LBA)

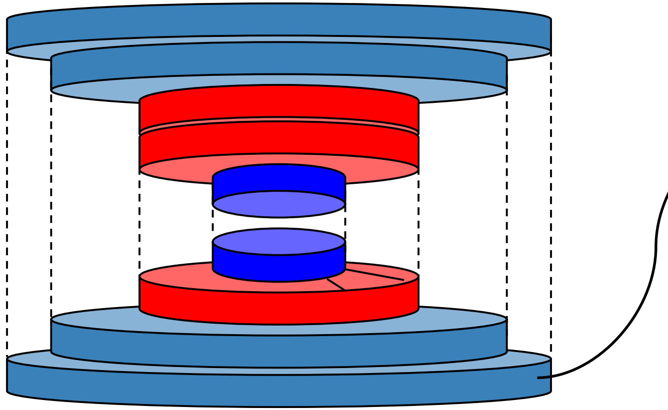
- CHS addresses have a limit of 8 GB.
  - Not enough bits allocated to store values in the Master Boot Record of disks.
- Logical Block Addresses (LBA) overcome this:
  - Single address (integer indexes) instead of CHS tuples
  - **Starts at 0**, so LBA 0 == CHS 0,0,1.
  - To convert from CHS, need to know:
    - CHS address.
    - Number of heads per cylinder (***HPC***).
    - Number of sectors per track (***SPT***).

LBA value	CHS tuple
0	0, 0, 1
1	0, 0, 2
2	0, 0, 3
62	0, 0, 63
63	0, 1, 1
945	0, 15, 1
1007	0, 15, 63
1008	1, 0, 1

CHS to LBA Conversion

- LBA = (((**C** * ***HPC***) + **H**) * ***SPT***) + **S** -1

= num of platters * 2



- CHS (***x***,***y***,***z***)
  - Locate the ***x***-th cylinder and calculate the number of sectors
  - Locate the ***y***-th head and calculate the number of sectors
  - Add (***z***-1) sectors

Address Conversion: Practice 1

- Given a disk with **16 heads** per cylinder and **63 sectors** per track, if we had a CHS address of **cylinder 2**, **head 3**, and **sector 4**, what would be the LBA (a.k.a CHS (2,3,4))?

LBA = (((**C** * HPC) + **H**) * SPT) + **S** -1

(((2*16) + 3) * 63)+ 4-1

=2208

Arizona State University

Address Conversion: Practice 2

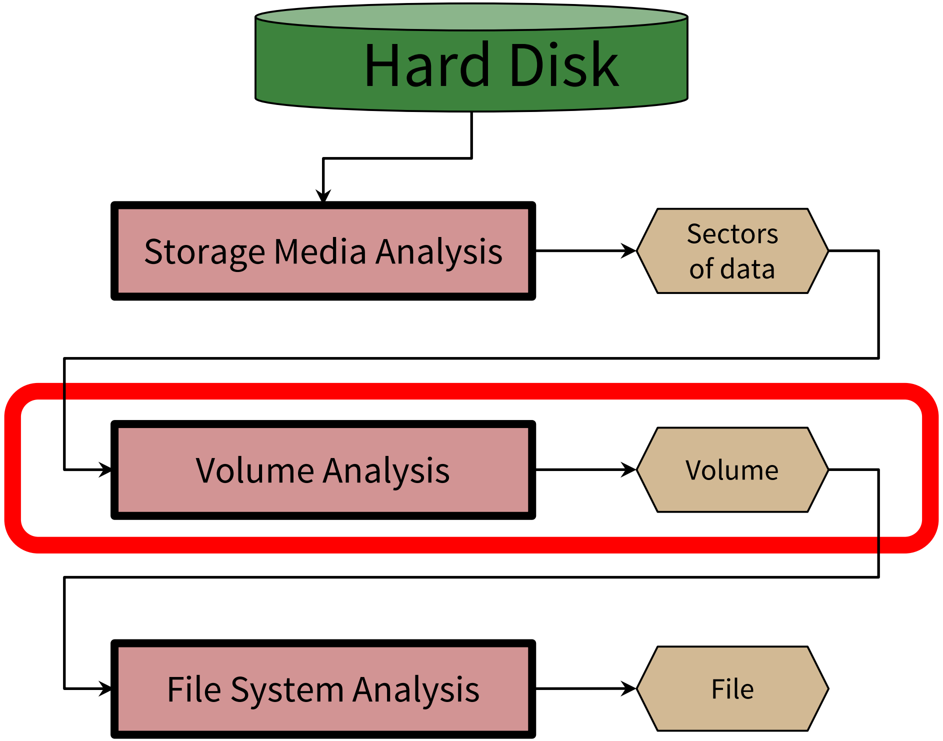
- Given a disk with **16 heads** per cylinder and **63 sectors** per track, if we had a CHS address of **cylinder 4**, **head 3**, and **sector 2**, what would be the LBA (a.k.a CHS (4,3,2))?

LBA = (((**C** * HPC) + **H**) * SPT) + **S** -1

$$(((4*16) + 3) * 63) + 2 - 1$$

$$= 4222$$

Volumes and Partitions


Hard Disk
Storage Media Analysis
Sectors of data
Volume Analysis
Volume
File System Analysis
File

Volume Analysis

- Volume/Partition:
  - Collection of ***addressable sectors*** that an OS or application can use for data storage.
  - Used to store file system and other structured data.

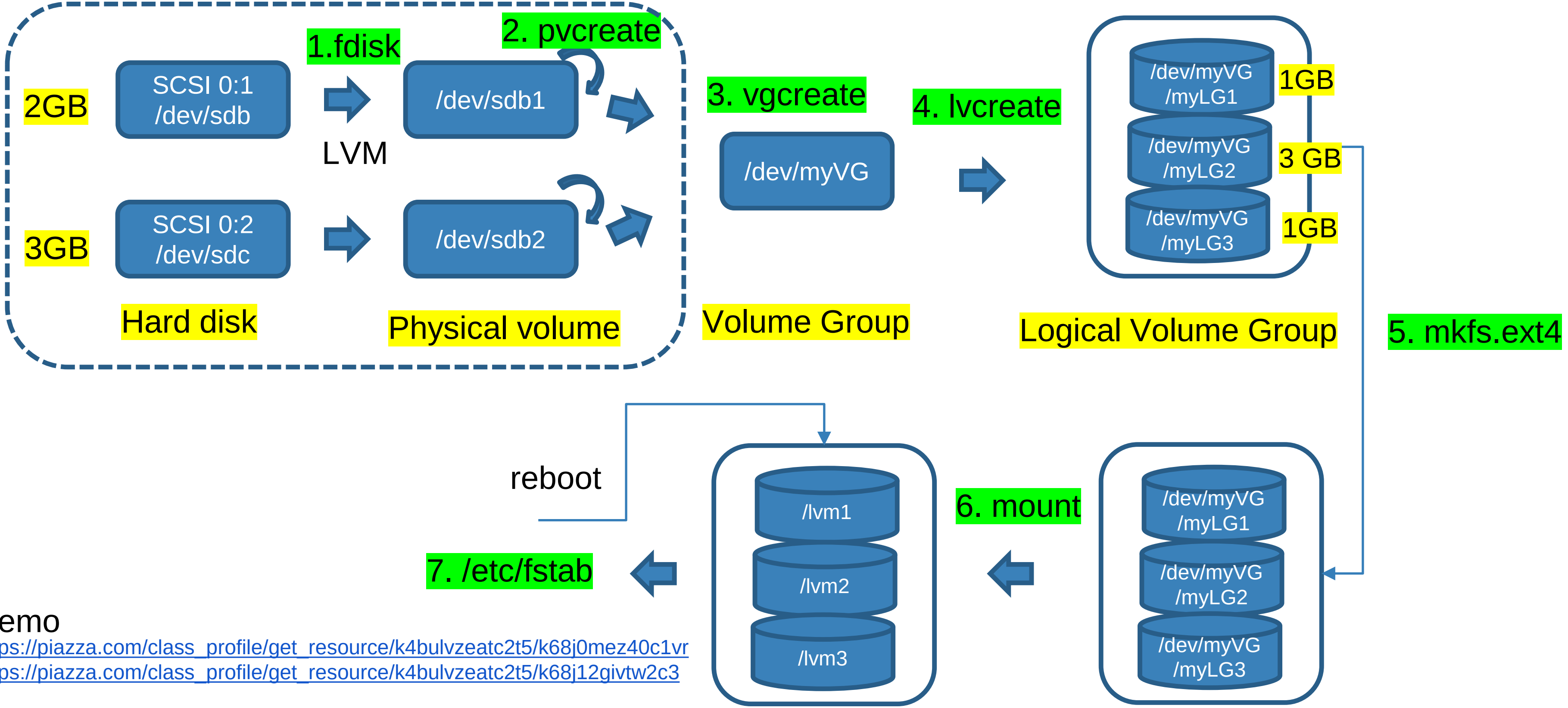
- Purpose of Volume Analysis:
  - Involves looking at the data structures that are involved with partitioning and assembling the bytes in storage devices.

Logical Volume Management (LVM)

Img source: https://en.wikipedia.org/wiki/File:LVM1.svg

- Managing the volumes for data storage separately from the underlying physical disks.
- Creating logical volumes which can be extended and reduced, making it possible to add capacity or to remove capacity from a volume.
  Ex) 2 TB + 3 TB => 1 TB, 3 TB, 1 TB
- Most modern Linux distributions are LVM-aware to the point of being able to have their root file systems on a logical volume.

LVM Exercise



Demo

https://piazza.com/class_profile/get_resource/k4bulvzeatc2t5/k68j0mez40c1vr

https://piazza.com/class_profile/get_resource/k4bulvzeatc2t5/k68j12givtw2c3

Add two hard disks, 2GB and 3GB in Vmware

Run with Root user or w/ sudo command
$ su –

1. create partition as LVM: fdisk
fdisk /dev/sdb
n : n partition
partition p : primary
1 : partition number
sectors : default (2048, 4194303)
t : type
Partition type lvm) : 8e
p : print, check the Linux LVM
w : save

fdisk /dev/sdc //repeat

2. create pysical volume: pvcreate
pvcreate /dev/sdb1
pvcreate /dev/sdc1

3. create volume group: vgcreate
vgcreate myVG /dev/sdb1 /dev/sdc1
vgdisplay // check the VG size

4. creat logical volume: lvcreate
lvcreate --size 1G --name myLG1 myVG
lvcreate --size 3G --name myLG2 myVG
lvcreate --extents 100%FREE --name myLG3 myVG

ls -l /dev/myVG

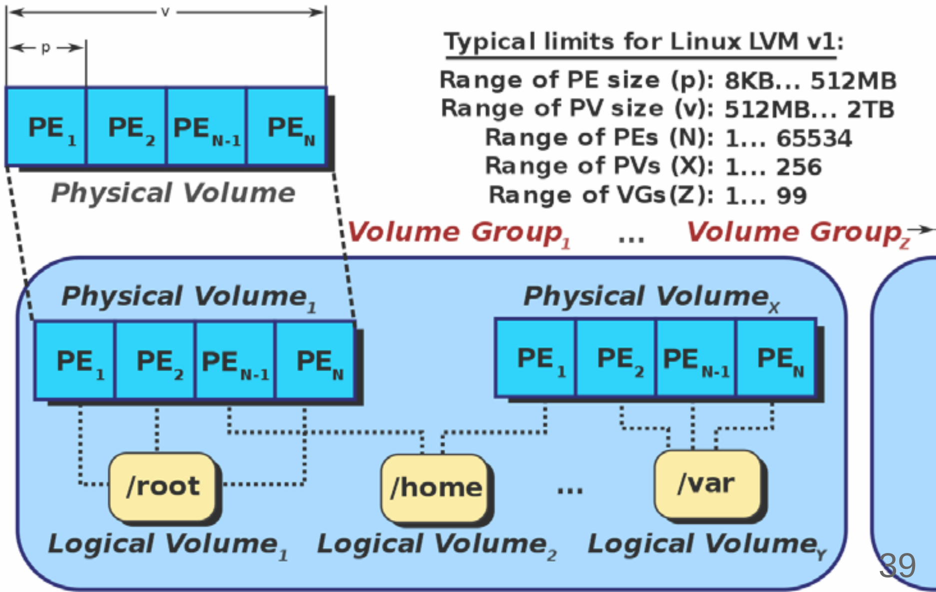
5. format: mkfs.ext4
mkfs.ext4 /dev/myVG/myLG1
mkfs.ext4 /dev/myVG/myLG2
mkfs.ext4 /dev/myVG/myLG3

6. mount
mkdir /lvm1 /lvm2 /lvm3
mount /dev/myVG/myLG1 /lvm1
mount /dev/myVG/myLG2 /lvm2
mount /dev/myVG/myLG3 /lvm3
df // check

7. edit /etc/fstab
vi /etc/fstab
// add these at the end of the file
/dev/myVG/myLG1 /lvm1 ext4 default 1 2
/dev/myVG/myLG2 /lvm2 ext4 default 1 2
/dev/myVG/myLG3 /lvm3 ext4 default 1 2

Logical Volume Management (LVM)

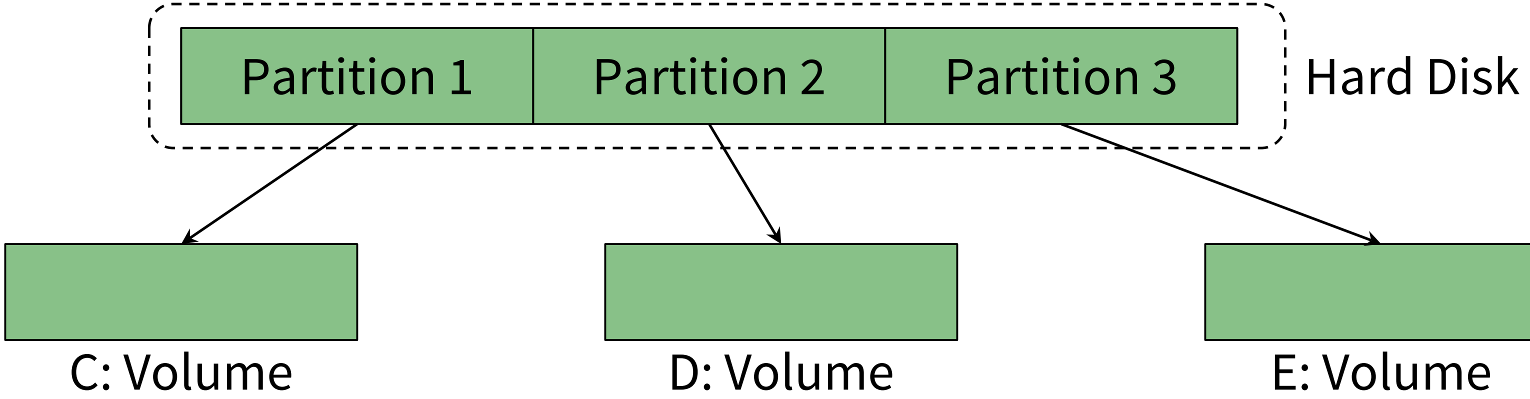
- LVM terms:
  - Physical volumes (PVs): Hard disks or hard disk partitions
  - Physical Extents (PEs): Basically clusters (groups of sectors)
  - Physical Volume Group (PVG): Pool of PEs
  - Logical Extents (LEs): Logical mapping to PEs
  - Volume Group (VG): Pool of LEs
  - Logical Volumes (LVs): Concatenation of LEs



Img source: https://en.wikipedia.org/wiki/File:LVM1.svg

Partitions

- Collection of ***consecutive*** sectors in a volume.
- Each OS and hardware platform use a different partitioning method.



Partitions: Purpose

- Partitions organize the layout of a volume.
- Essential data are the ***starting*** and ***ending*** location for each partition.
- Common partition systems have one or more tables and each table describes a partition:
  - Starting sector of the partition: start sector x 512 bytes
  - Ending sector of the partition: the length x 512 bytes
  - Type of partition (0x04)

the number of the sector

fdisk -l

```
Disk /dev/sda: 40 GiB, 42949672960 bytes, 83886080 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0x7c63f384

Device     Boot    Start      End  Sectors  Size Id Type
/dev/sda1  *        2048 81788927 81786880   39G 83 Linux
/dev/sda2       81790974 83884031  2093058 1022M  5 Extended
/dev/sda5       81790976 83884031  2093056 1022M 82 Linux swap / Solaris
```

Master Boot Record (MBR)

- First sector (CHS 0,0,1) stores the disk layout (512 Bytes).
- Each **partition entry** has the structure shown on the next slide. *($sudo parted, p) ($ sudo hexdump –C /dev/sda –n 512)*

Offset	Description	Size
0x0000	Bootstrap code area	446 Bytes
0x01BE	1st Partition Entry (Primary)	16 Bytes
0x01CE	2nd Partition Entry	16 Bytes
0x01DE	3rd Partition Entry	16 Bytes
0x01EE	4th Partition Entry	16 Bytes
0x01FE-01FF	**Boot Signature (0x55 0xAA)**	2 Bytes

MBR Partition Entry (0x01BE, 16Byte)

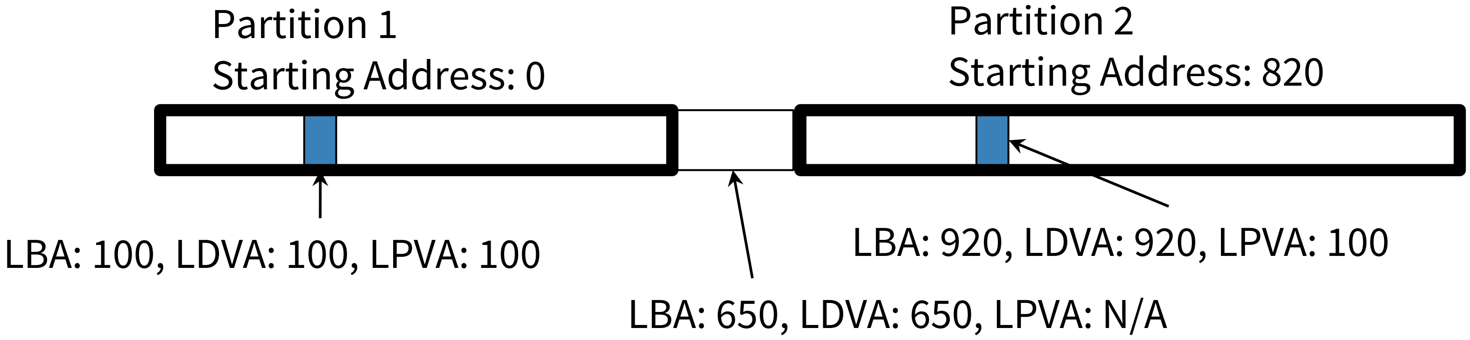
Offset	Description	Size
0x00	Current State of Partition (Flag) (0x00=Inactive, 0x80=Active)	1 byte
0x01	Beginning of Partition - Head	1 byte
0x02	Beginning of Partition - Cylinder/Sector	1 word (2 bytes)
0x04	Type of Partition (0x83 = Linux FS)	1 byte
0x05	End of Partition - Head	1 byte
0x06	End of Partition - Cylinder/Sector	1 word (2 bytes)
0x08	LBA of First Sector in the Partition	1 double word (4 bytes)
0x0C	Number of Sectors in the Partition	1 double word

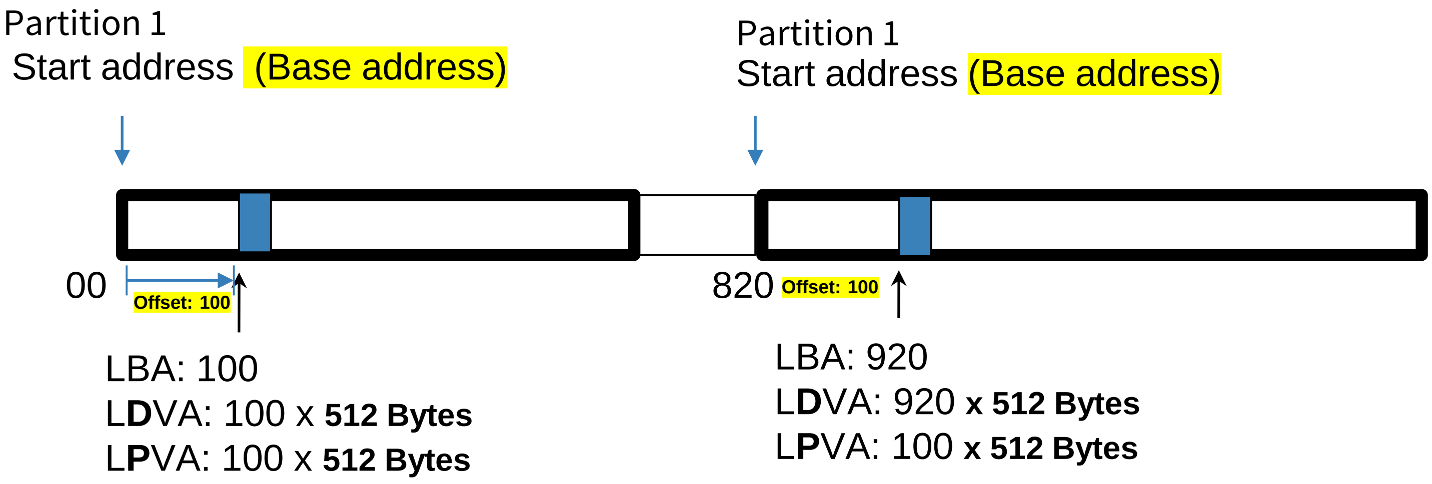
Note on MBRs

- Maximum addressable storage space: 2 TiB.
  - 32bit addressing, 2^32 = 4G, 4Gx 512byte = 2^{40} bytes
- In the process of being superseded by the GUID Partition Table (GPT) scheme.
  - A little more complicated, not going to explain here.
  - GPTs offer limited backwards compatibility.
- See Wikipedia for more info:
  - https://en.wikipedia.org/wiki/Master_boot_record
  - https://en.wikipedia.org/wiki/GUID_Partition_Table
- Tons of supported partition types (offset 0x04):
  - https://en.wikipedia.org/wiki/Partition_type

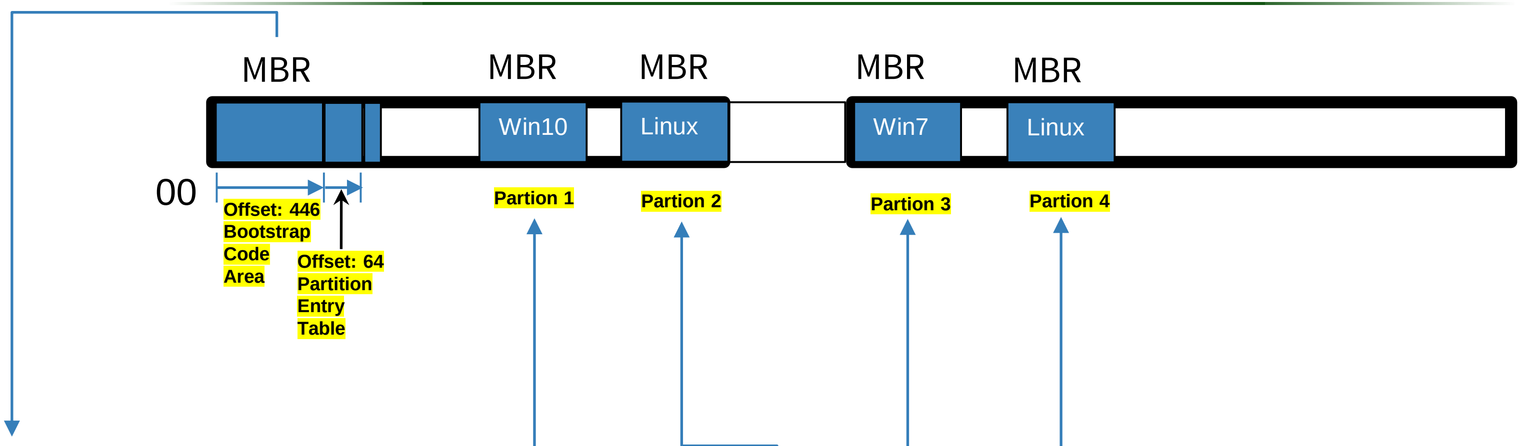
Sector Addressing

- Logical Volume Address:
  - Logical "Disk" Volume Address (LDVA)
    - Relative to the start of the volume.
  - Logical "Partition" Volume Address (LPVA)
    - Relative to the start of the partition.





Sector based address (CHS, LBA) needs to be multiplied by 512 bytes to map to disk.



Offset	Description	Size
0x0000	Bootstrap code area (boot loader)	446 Bytes
0x01BE	1st Partition Entry	16 Bytes
0x01CE	2nd Partition Entry	16 Bytes
0x01DE	3rd Partition Entry	16 Bytes
0x01EE	4th Partition Entry	16 Bytes
0x01FE-01FF	**Boot Signature (0x55 0xAA)**	2 Bytes

Offset	Description	Size
0x00	Current State of Partition (Flag) (0x00=Inactive, 0x80=Active)	1 byte
0x01	Beginning of Partition - Head	1 byte
0x02	Beginning of Partition - Cylinder/Sector	1 word (2 bytes)
0x04	Type of Partition (0x83 = Linux FS)	1 byte
0x05	End of Partition - Head	1 byte
0x06	End of Partition - Cylinder/Sector	1 word (2 bytes)
0x08	LBA of First Sector in the Partition	1 double word (4 bytes)
0x0C	Number of Sectors in the Partition	1 double word

469: Computer and Network Forensics

Partition Analysis Steps

1. Locate the partition tables.
2. Process the data structures to identify the layout since we need to know the offset of a partition.
   - It is important to discover the **partition layout** of the volume because not all sectors need to be assigned to a partition and they **may contain data from a previous file system or that the suspect was trying to hide**.
3. Conduct the consistency checks:
   - Looks at the last partition and compares its starting location with the end of its parent partition.
   - To determine where else evidence could be located besides in each partition.

Note: To analyze the data inside a partition, we need to consider what type of data it is—normally it's a file system.

Extraction of Partition Contents

- Need to extract the data in or in between partitions to a separate file.
- Tools:
  - `dd` tool:
    - `if`, `of`, `bs` (512 bytes), `skip` (blocks to skip), `count` (blocks to copy)
  - `mmls` tool from the Sleuth Kit.
  - Any hex editor.

Volume Analysis

```
# mmls -t dos disk1.dd
    Units are in 512-byte sectors
        Slot  Start      End        Length     Description
    00: ----- 0000000000 0000000000 0000000001 Table #0
    01: ----- 0000000001 0000000062 0000000062 Unallocated
    02: 00:00 0000000063 0001028159 0001028097 Win95 FAT32 (0x0B)
    03: ----- 0001028160 0002570399 0001542240 Unallocated
    04: 00:03 0002570400 0004209029 0001638630 OpenBSD (0xA6)
```


```
# dd if=disk1.dd of=part1.dd bs=512 skip=63 count=1028097
# dd if=disk1.dd of=part2.dd bs=512 skip=2570400 count=1638630
```

Volume Analysis (MBR)

```
0000432: 0000 0000 0000 0000 0000 0000 0000 0001
0000448: 0100 07fe 3f7f 3f00 0000 4160 1f00 8000
0000464: 0180 0bfe 3f8c 8060 1f00 cd2f 0400 0000
```

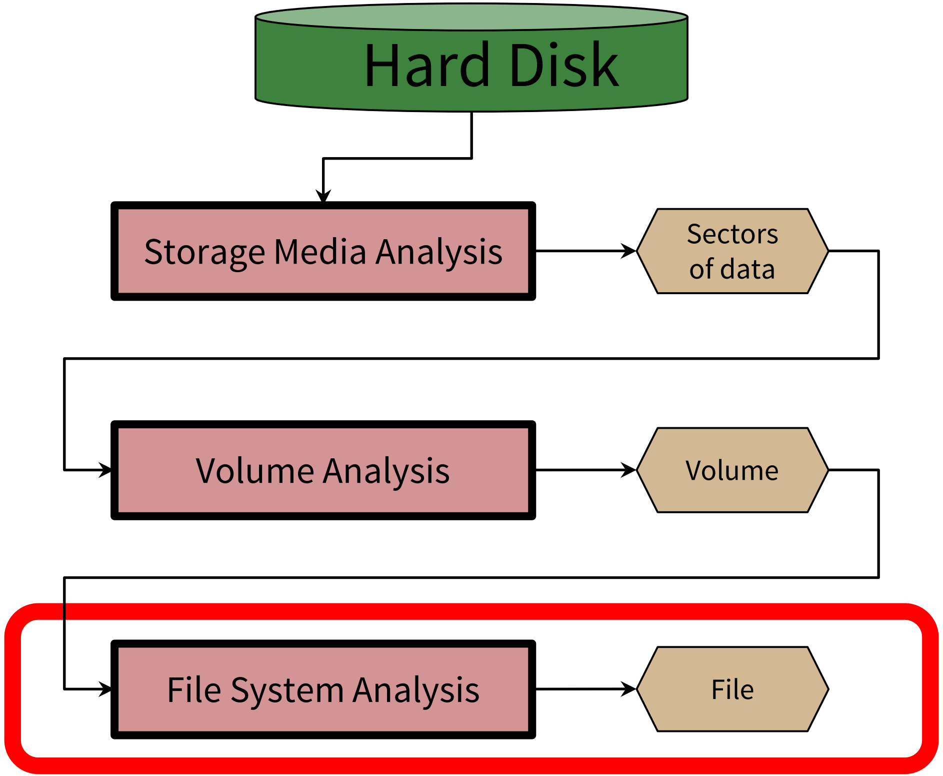
The first 446 bytes contain boot code

The byte offset in decimal

16 bytes of the data in hexadecimal

# of Partition	Flag	Type	Starting Sector	Size
1	0x00	0x07	0x0000003f (63)	0x001f6041 (2,056,257)
2				

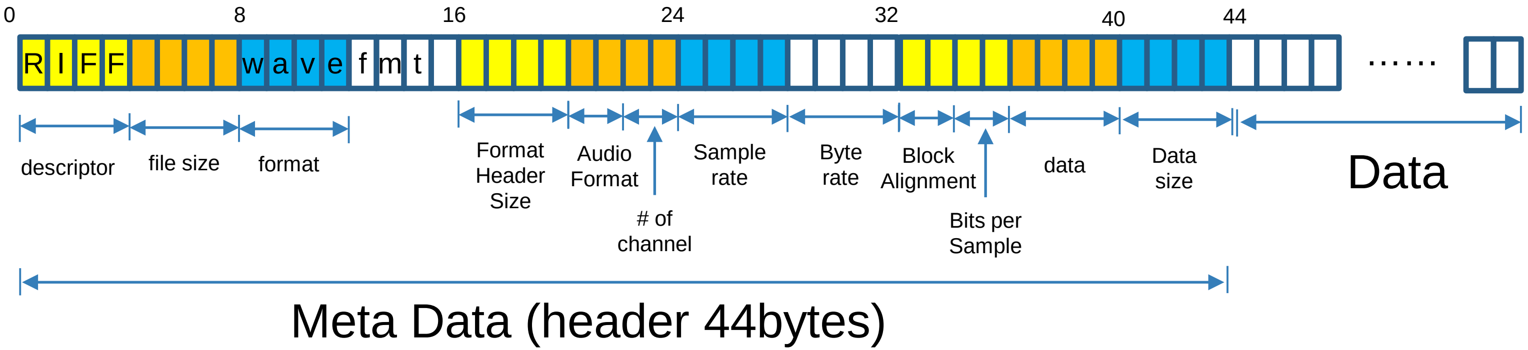
Files and Directories


Hard Disk
Storage Media Analysis
Sectors
of data
Volume Analysis
Volume
File System Analysis
File

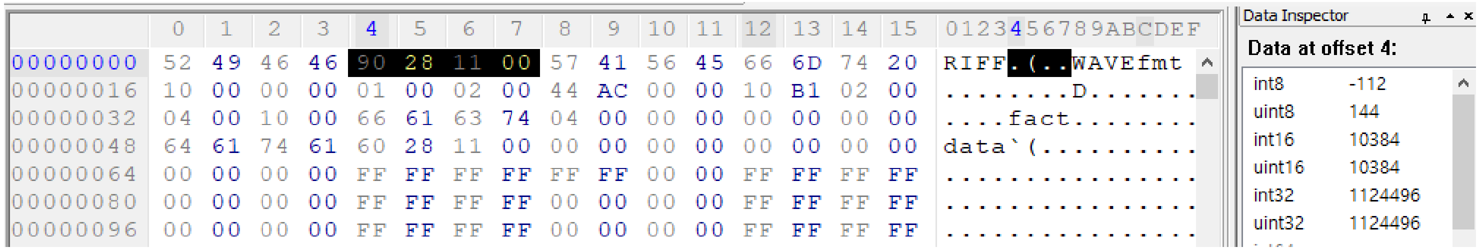
What is a File?

- A (potentially) large amount of information or data that lives a (potentially) very long time
  - Often *much* larger than the memory of the computer
  - Often *much* longer than any computation
  - Sometimes longer than life of machine itself
- (Usually) organized as a linear array of bytes or blocks
  - Internal structure is imposed by application
  - (Occasionally) blocks may be variable length
- (Often) requiring concurrent access by multiple processes
  - Even by processes on different machines!

Example wave file



http://soundfile.sapp.org/doc/WaveFormat/



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0123456789ABCDEF
00000000	52	49	46	46	90	28	11	00	57	41	56	45	66	6D	74	20	RIFF.(..WAVEfmt
00000016	10	00	00	00	01	00	02	00	44	AC	00	00	10	B1	02	00	........D.......
00000032	04	00	10	00	66	61	63	74	04	00	00	00	00	00	00	00	....fact........
00000048	64	61	74	61	60	28	11	00	00	00	00	00	00	00	00	00	data`(.........
00000064	00	00	00	00	FF	FF	FF	FF	FF	FF	00	00	FF	FF	FF	FF	....
00000080	00	00	00	00	FF	FF	FF	FF	00	00	00	00	FF	FF	FF	FF	....
00000096	00	00	00	00	FF	FF	FF	FF	00	00	00	00	FF	FF	FF	FF	....

Data Inspector

Data at offset 4:

int8	-112
uint8	144
int16	10384
uint16	10384
int32	1124496
uint32	1124496

File Systems and Disks

- User view:
  - File is a ***named***, ***persistent*** collection of data.

- OS & file system view:
  - File is collection of disk blocks — i.e., a ***container***.
  - File System ***maps*** file names and offsets to disk blocks.

Fundamental Ambiguity...

- Is the ***file*** the "container of the information" or the "information" itself?
- Almost all systems confuse the two.
- Almost all people confuse the two.

File Attributes

- Name:
  - Although the name is not always what you think it is!
- Type:
  - May be encoded in the name (e.g., `.cpp`, `.txt`)
- Dates/Time:
  - Creation, updated, last accessed, etc.
  - (Usually) associated with container.
  - Better if associated with content.
- Size:
  - Length in number of bytes; occasionally rounded up.
- Protection:
  - Owner, group, etc.
  - Authority to read, update, extend, etc.
- Locks:
  - For managing concurrent access.
- ...

File Metadata

- Definition:
  - Information ***about*** a file. Data ***about*** the data.
- Maintained by the file system.
- Separate from file itself.
- Usually attached or connected to the file.
- Some information visible to user/application:
  - Dates, permissions, type, name, etc.
- Some information primarily for OS:
  - Location on disk, locks, cached attributes

Some Common File Types

		Manager
DLP	Data	DAQLab
DLS	Setup	Norton Disklock
DMF	Music format (Delusion Digital Music File)	Delusion
DMF	DeleD files	
DMG	Apple Disk Image	Mac OS X (Disk Utility),
DMO	Demo	Derive
DMP	Dump file (e.g. screen or memory)	
DMP	DUMP utility output file	RT-11
DMS	Amiga file archive	DISKMASHER
DNG	Digital Negative (DNG), a publicly available archival format for the raw files generated by digital cameras	At least 30 camera models from at least 10 manufacturers, and at least 200 software products
DNT	data	RSNetWorx Project
DO	Java servlet file	Web browser (Various)
DOC	A Document, or an ASCII text file with text formatting codes in with the text; used by many word processors	Microsoft Word and others
DOCM	Microsoft Word 2007 Master document	Microsoft Word 2007
DOCX	Office Open XML Text document	Microsoft Word
DOF	Delphi project options file	Borland Delphi

https://www.slideshare.net/shubhamrastogi11/file-name-extensions

https://piazza.com/class_profile/get_resource/k4bulvzeatc2t5/k6hxfzphsg616i

https://filesignatures.net/

File Operations

- *Create*, *Delete*:
  - Conjure up a new file; or forget about an existing one.
- *Open*, *Close*
  - Gain or relinquish access to a file.
  - OS returns a ***file handle*** – an internal data structure letting it cache internal information needed for efficient file access.
- *Read*, *Write*, *Truncate*
  - *Read*: Return a sequence of *n* bytes from file.
  - *Write*: Replace *n* bytes in file, and/or append to end.
  - *Truncate*: Throw away all but the first *n* bytes of file.
- *Seek*, *Tell*
  - *Seek*: Reposition file pointer for subsequent reads and writes.
  - *Tell*: Get current file pointer.

File – A Very Powerful Abstraction

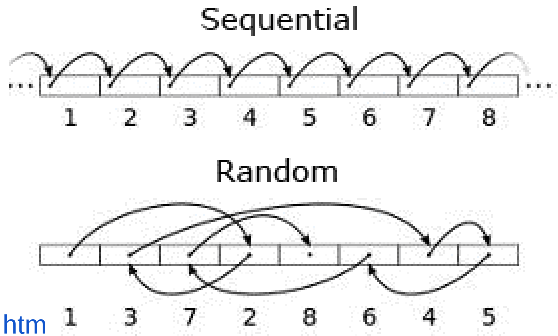
- Documents, code.
- Databases:
  - Very large, possibly spanning multiple disks.
- Streams:
  - Input, output, keyboard, display.
  - Pipes, network connections, …
- Virtual memory backing store.
- Temporary repositories of OS information.
- Anytime you need to remember something beyond the life of a particular process/computation.

Methods for Accessing Files

- *Sequential* Access Methods (SAM)
- *Random* Access Methods (RAM)
- *Keyed* (or indexed) Access Methods (KAM)

Sequential Access Method

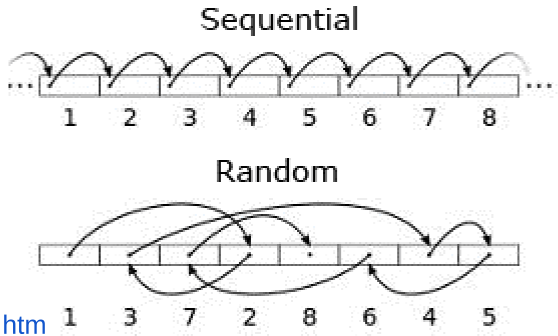
- Read all bytes or records *in order* from the beginning.
- (Over) Writing implicitly truncates files.
- Cannot jump around.
  - Possible to rewind or back up.
- Appropriate for certain media or systems:
  - Magnetic tape or punched cards
  - Video tape (VHS, etc.)
  - Unix-Linux-Windows pipes
  - Network streams



 https://www.computerhope.com/jargon/s/sequacce.htm

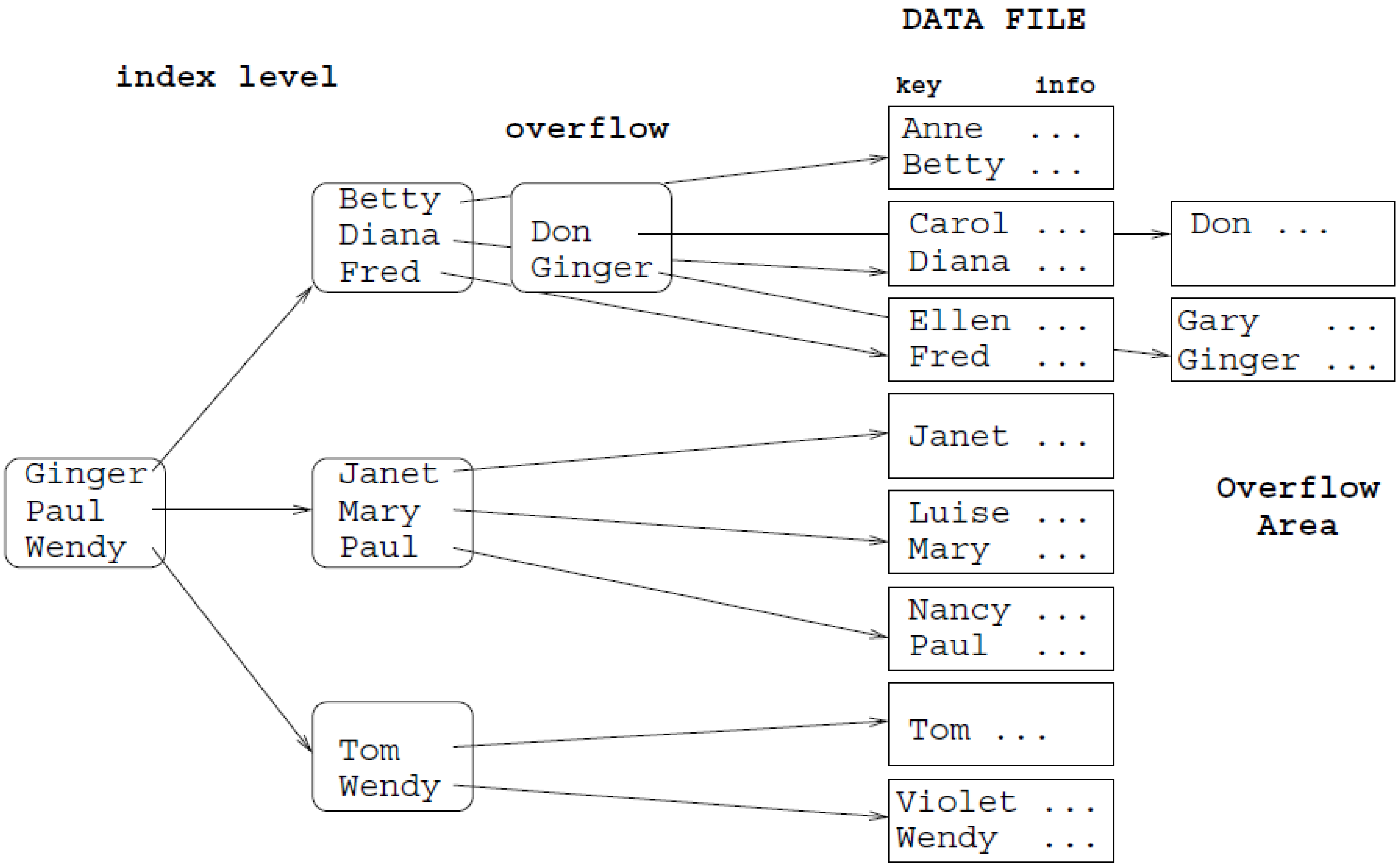
Random Access Method

- Bytes/records can be read in any order.
- Writing can:
  - Replace existing bytes or records.
  - Append to end of file.
  - Cannot insert data between existing bytes!
- Seek operation moves current ***file pointer***.
  - Maintained as part of "open" file information.
  - Discarded on close.
- Typical of most modern information stor
  - Database systems.
  - Randomly accessible multimedia (CD, DVD, etc).
  - …



Keyed (or Indexed) Access Methods

- Access items in file based on the contents of (part of) an item in the file.
- Provided in older commercial operating systems (IBM ISAM).
- (Usually) handled separately by modern database systems.



https://it.wikipedia.org/wiki/ISAM

Directory (folder) – A Special Kind of File

- A tool for users and applications to organize and find files.
  - User-friendly names.
  - Names that are meaningful over long periods of time.

- The data structure for OS to locate files (i.e., containers) on disk.

Directory Structures

- Single level:
  - One directory per system, one entry pointing to each file.
  - Small, single-user or single-use systems.
    - PDA, cell phone, etc.
- Two-level:
  - Single “master” directory per system.
  - Each entry points to one single-level directory ***per user***.
  - Uncommon in modern operating systems.
- Hierarchical:
  - Any directory entry may point to:
    - Individual file.
    - Another directory.
  - Common in most modern operating systems.

Directory Considerations

- Efficiency: locating a file quickly.
- Naming: convenient to users.
  - Separate users can use same name for separate files.
  - The same file can have different names for different users.
  - Names need only be unique within a directory.
- Grouping: logical grouping of files by properties.
  - e.g., all Java programs, all games, …

Directory Organization – Hierarchical

- Most systems support idea of current (working) directory
  - Absolute names – fully qualified from root of file system:
    - `/usr/group/foo.c, ~/kernelSrc/config.h`
  - Relative names – specified with respect to working directory:
    - `foo.c, bar/bar2.h`
  - A special name – the working directory itself:
    - "."
- Modified Hierarchical – Acyclic Graph (no loops) and General Graph:
  - Allow directories and files to have multiple names.
  - Links are file names (directory entries) that point to existing (source) files.

Links

- Symbolic (soft) links:
  - Unidirectional relationship between a filename and the file.
  - Directory entry contains *text* describing *absolute* or *relative* path name of original file.
  - If the source file is deleted, the link exists but pointer is invalid.
- Hard links:
  - Bidirectional relationship between file names and file.
  - A hard link is directory entry that points to a source file's metadata (i-node).
  - Metadata maintains *reference count* of the number of hard links pointing to it – *link reference count*.
  - Link reference count is decremented when a hard link is deleted.
  - File data is deleted and space freed when the link reference count goes to zero.

Path Name Translation (1)

- Assume that I want to open `/home/lauer/foo.c`:
  `fd = open(“/home/lauer/foo.c”, O_RDWR);`
- The filesystem does the following:
  - Opens directory **/** – the root directory is in a known place on disk.
  - Search root directory for the directory **home** and get its location.
  - Open **home** and search for the directory **`lauer`** and get its location.
  - Open **`lauer`** and search for the file **`foo.c`** and get its location.
  - Open the file **`foo.c`**.
  - Note that the process needs the **appropriate permissions** at *every* step.
- …

Path Name Translation (2)

- …
- File Systems spend a lot of time walking down directory paths:
  - This is why **open** calls are separate from other file operations.
  - The filesystem attempts to cache prefix lookups to speed up common searches:
    - **"~"** for user's home directory.
    - **"."** for current working directory.
  - Once open, file system caches the metadata of the file.

Directory Operations

- *Create*:
  - Make a new directory.
- *Add entry*, *Delete entry*:
  - Invoked by file create & destroy, directory create & destroy.
- *Find*, *List*:
  - Search or enumerate directory entries.
- *Rename*:
  - Change name of an entry without changing anything else about it.
- *Link*, *Unlink*:
  - Add or remove entry pointing to another entry elsewhere.
  - Introduces possibility of loops in directory graph.
- Destroy:
  - Removes directory; must be empty.

Directories - Last Thoughts

- Orphan:
  - A file not named in any directory.
  - Cannot be opened by any application (or even OS).
  - May not even have name!
- Tools:
  - (e2)FSCK – check & repair file system, find orphans.
    (file system consistency check)
- Special directory entry: ".." ⇒ parent directory:
  - Essential for maintaining integrity of directory system.
  - Useful for relative naming.

Bonus Topic:
How Does a PC boot?

Why is Booting Required?

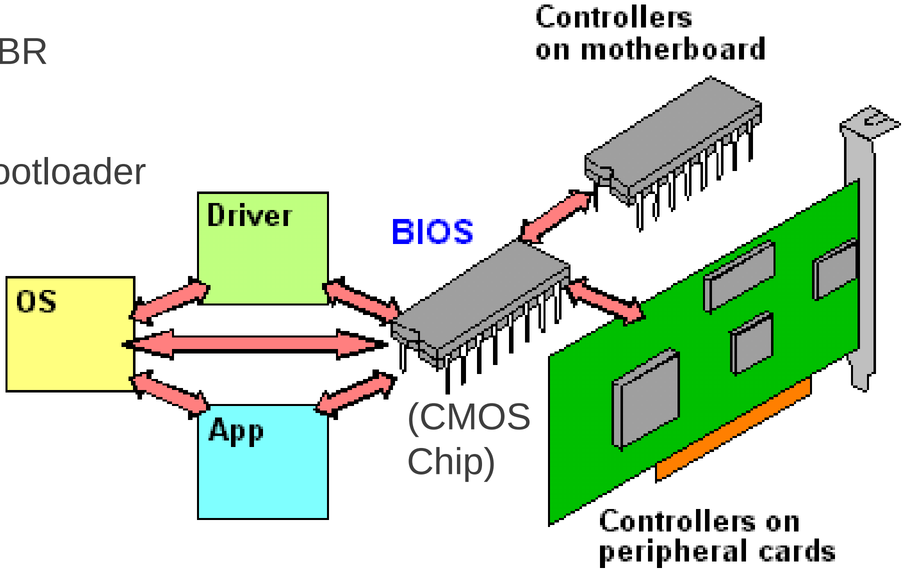
- Hardware doesn't know where the operating system resides and how to load it.
- Need a special program to do this job – **Bootstrap** loader.
  - E.g. BIOS – Boot Input Output System.
- Bootstrap loader locates the kernel (OS), loads it into main memory and starts its execution.
- In some systems, a simple bootstrap loader fetches a more complex boot program from disk, which in turn loads the kernel.

How Boot process occurs?

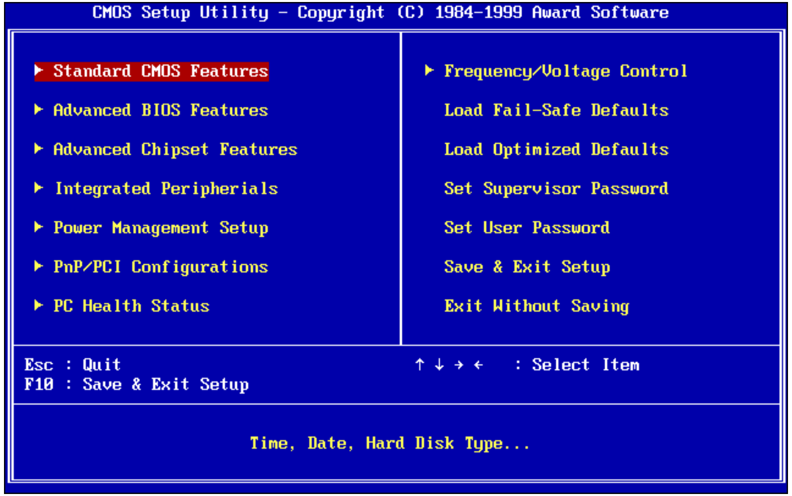
- Reset event on CPU (power up, reboot) causes instruction register to be loaded with a predefined memory location. It contains a jump instruction that transfers execution to the location of Bootstrap program.
- This program is in form of ROM, since RAM is in unknown state at system startup. ROM is convenient as it needs no initialization and can’t be affected by virus.

BIOS Interaction

1. POST (Power-On Self Test)
2. MBR
3. Bootloader



BIOS Interaction



Since 1980s, CMOS
(Complementary Metal-Oxide-Semiconductor)

In 2007, UEFI
(Unified Extended Firmware Interface)
Windows Vista Service PK1, Windows 7

Unified Extensible Firmware Interface(UEFI)

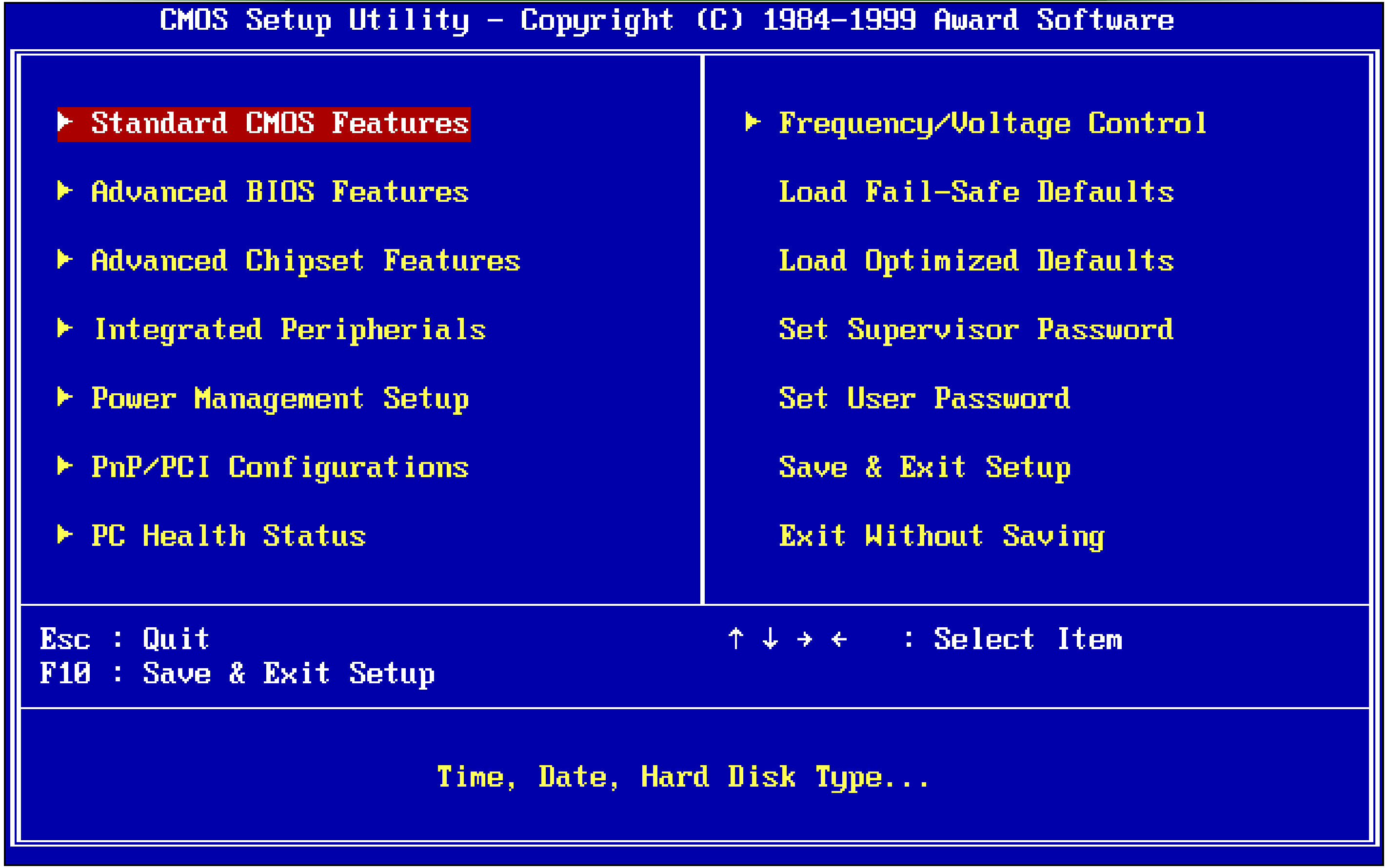
- Can boot from drives of 2.2 TB or larger (9.4 zettabytes)
- The GPT(GUID Partition Table) partitioning scheme instead of MBR
- Can run in 32-bit or 64-bit mode and has more addressable address space than BIOS
- Supports Secure Boot: checking for validity to ensure no malware has tampered with the boot process. (Microsoft's certificate stored in UEFI)
- UEFI is essentially a tiny OS runs on top of the PC's firmware

https://www.howtogeek.com/56958/htg-explains-how-uefi-will-replace-the-bios/

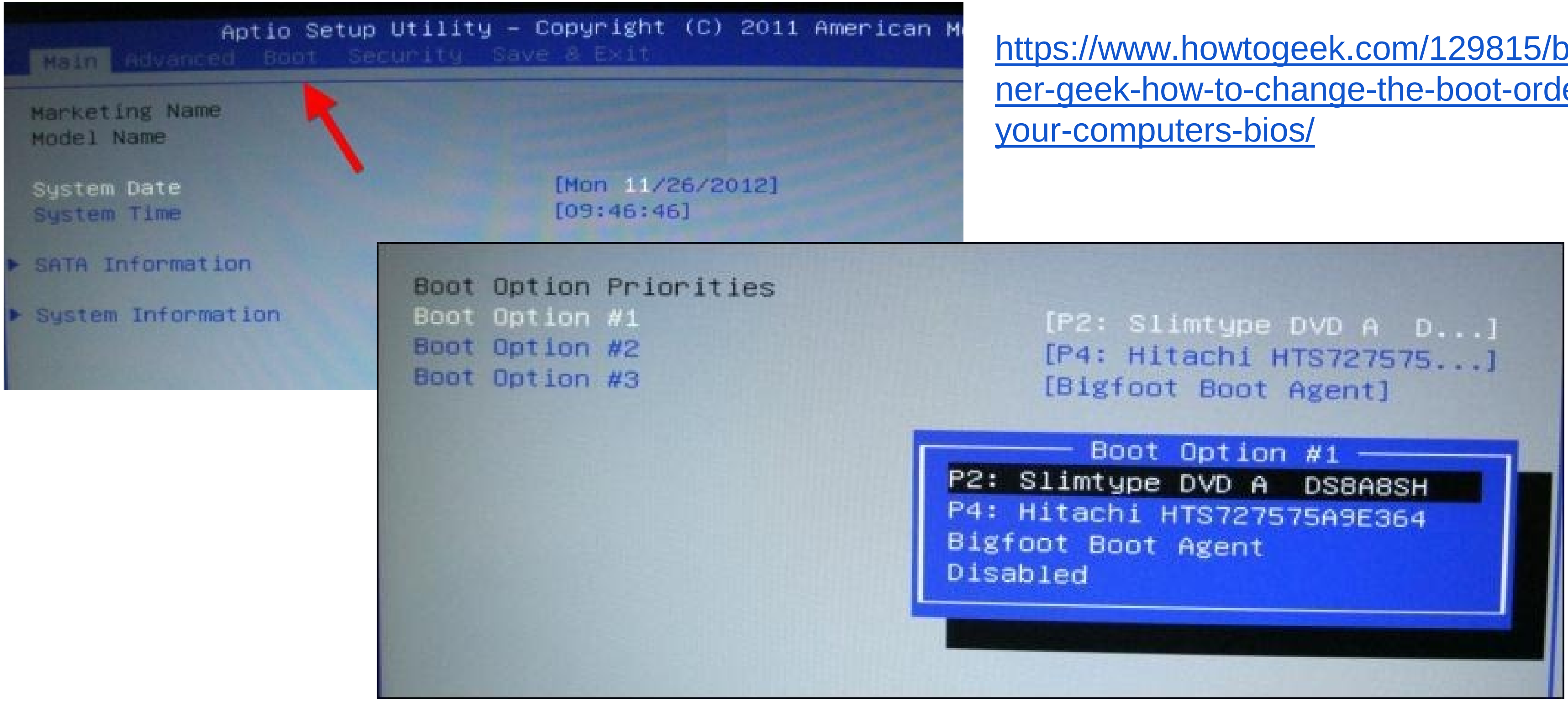
Tasks performed at boot up

- Run diagnostics to determine the state of machine. If diagnostics pass, booting continues.
- Runs a Power-On Self Test (*POST*) to check the devices that the computer will rely on, are functioning.
- BIOS goes through a preconfigured list of devices until it finds one that is bootable. If it finds no such device, an error is given and the boot process stops.
- Initializes CPU registers, device controllers and contents of the main memory. After this, it loads the OS.

BIOS Setup


CMOS Setup Utility - Copyright (C) 1984-1999 Award Software
Standard CMOS Features
Advanced BIOS Features
Advanced Chipset Features
Integrated Peripherials
Power Management Setup
PnP/PCI Configurations
PC Health Status
Frequency/Voltage Control
Load Fail-Safe Defaults
Load Optimized Defaults
Set Supervisor Password
Set User Password
Save & Exit Setup
Exit Without Saving
Esc : Quit
F10 : Save & Exit Setup
↑ ↓ → ← : Select Item
Time, Date, Hard Disk Type...

Boot Procedure



https://www.howtogeek.com/129815/beginner-geek-how-to-change-the-boot-order-in-your-computers-bios/

Tasks performed at boot up

- On finding a bootable device, the BIOS loads and executes its boot sector. In the case of a hard drive, this is referred to as the master boot record (*MBR*) and is often not OS specific.
- The MBR code checks the partition table for an active partition. If one is found, the MBR code loads that partition's boot sector and executes it.
- The boot sector is often operating system specific, however in most operating systems its main function is to load and execute a kernel, which continues startup.

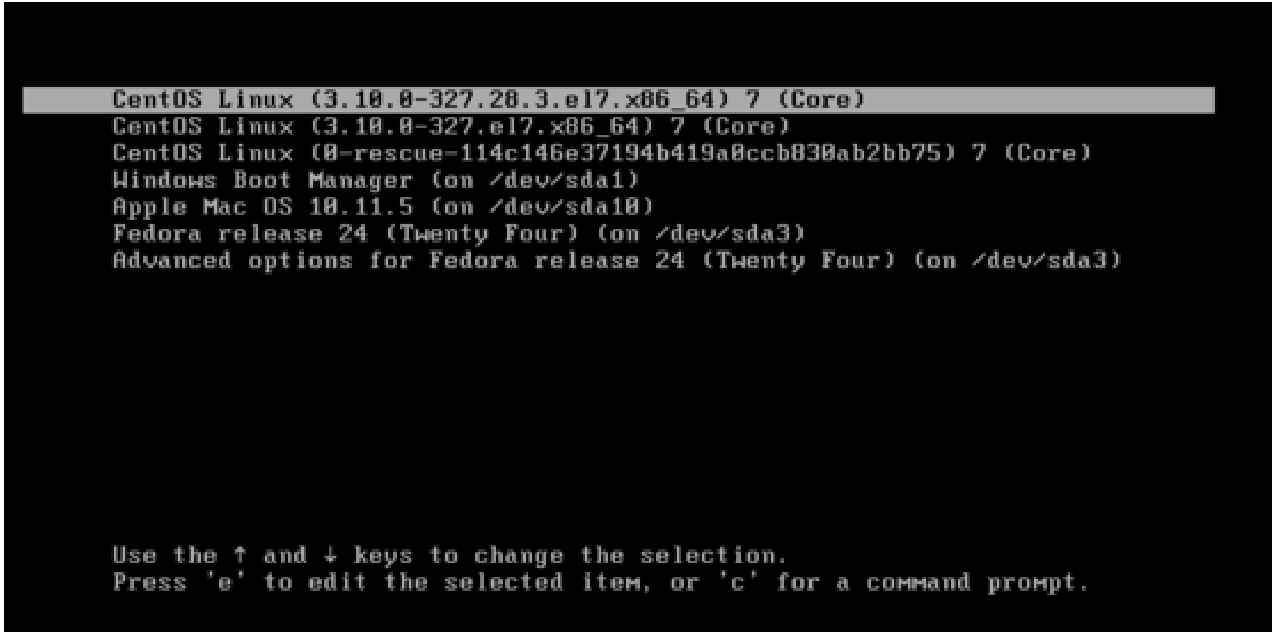
Secondary Boot Loaders

- If there is no active partition or the active partition's boot sector is invalid, the MBR may load a secondary boot loader and pass control to it and this secondary boot loader will select a partition (often via user input) and load its boot sector.

- Examples of secondary boot loaders
  - GRUB – GRand Unified Bootloader
  - LILO – LInux LOader
  - NTLDR – NT Loader

GRUB

(GNU GRand Unified Bootloader)



https://fedoramagazine.org/what-is-grub2-boot-loader/

Booting and ROM

- System such as cellular phones, PDAs and game consoles stores entire OS on ROM. Done only for small OS, simple supporting hardware, and rugged operation.
- Changing bootstrap code would require changing ROM chips.
  - EPROM – Erasable Programmable ROM.
- Code execution in ROM is slower. Copied to RAM for faster execution.
  - Writing data to RAM is faster than ROM relatively.

Example: DOS

- After identifying the location of boot files, ***BIOS*** looks at the **first sector** (512 bytes) and copies information to specific location in RAM (7C00H) - Boot Record.
- Control passes from BIOS to a **program residing in the boot record**.
- Boot record loads the initial system file into RAM. For DOS, it is **IO.SYS** .
- The initial file, IO.SYS includes a file called **SYSINIT** which loads the remaining OS into the RAM.
- SYSINIT loads a system file **MSDOS.SYS** that knows how to work with BIOS.
- One of the first OS files that is loaded is the system configuration file, **CONFIG.SYS** in case of DOS. Information in the configuration file tells loading program which OS files need to be loaded (e.g. drivers)
- Another special file that is loaded is one which tells what specific applications or commands user wants to be performed as part of booting process. In DOS, it is **AUTOEXEC.BAT**. In Windows, it's **WIN.INI** .

Example: Windows 10

- Phase 1 – **Preboot (BIOS)**
  - the PC's firmware is in charge and initiates a POST and loads the firmware settings.
  - the system identifies a valid system disk and reads the MBR.
- Phase 2 – **Windows Boot Manager (Boot loader)**
  - The system starts the Windows Boot Manager (%SystemDrive%\bootmgr)
- Phase 3 – **Windows Operating System Loader**
  - Starts the Windows loader (Winload.exe: %SystemRoot%\system32**winload.exe**)
- Phase 4 – **Windows NT OS Kernel**
  - Essential drivers required to start the Windows kernel are loaded into memory (%SystemRoot%\system32**ntoskrnl.exe**)
  - The kernel loads the system registry hive into memory and loads the drivers that are marked as BOOT_START.
  - The kernel passes control to the session manager process (**Smss.exe**).
  - Load Wind32k.sys
  - Winlogon

https://docs.microsoft.com/en-us/windows/client-management/img-boot-sequence

admin> bcdedit /v

Dual boot

- **Dual booting** is the act of installing multiple operating systems on a computer, and being able to choose which one to boot when switching on the computer. The program, which makes dual booting possible is called a boot loader.

Example: Linux Partition

- When installing an OS on a computer from scratch, here is how the partition table is created.
- The hard disk is denoted as “hda” where hd=hard disk, and the third letter could mean the hard-disk on the system. For e.g. the first hard disk is “hda”, the second is “hdb”.
- When the partitioning is done, “hda0” is the place of MBR. “hda1” is the primary partition. Then a secondary partition may be created which is further subdivided into logical drives. Another OS could be installed on any of these logical drives.
  - hda0 – MBR
  - hda1 – Primary Partition e.g. Windows XP
  - hda2 – Secondary Partition
  - hda3 – Logical Drive 1 (FAT32 or NTFS partition)
  - hda4 – Logical Drive 2 (FAT32 or NTFS partition)
  - hda5 – Logical Drive 3 (Swap for Linux Partition)
  - hda6 – Logical Drive 4 (Root for Linux Partition)

The above example is a simple example.

Specific cases can be different.