

Topic 3: Technical Concepts

F20FO/F21FO – Digital Forensics

Mike Just (Edinburgh)

Ryad Soobhany (Dubai)

Recap – Topic 2

- Topic 2 overview: Digital forensics & digital evidence
 - Digital forensics evolution
 - Digital investigations & evidence
 - Digital forensics tools

Topic overview

- Topic 3 overview: Technical concepts
 - Operating systems
 - Data storage
 - File systems
 - Numbering systems

Operating Systems

Operating System (OS)

- What is an OS
- Data stored in an OS
 - User data
 - System data
- Data representation in an OS

What is an Operating System

- A set of computer programs that manage the hardware and software resources of a computer
- The operating system interface can be
 - Command line interpreter
 - Graphical user interface
- Most relevant for us, the OS provides access to the data storage in the computing device

Functions of an OS

Partitions and formats storage devices

Creates a standard for naming files and folders

Maintains the integrity of files and folders

Error Recovery

Security of the file system

OS Data – User Data

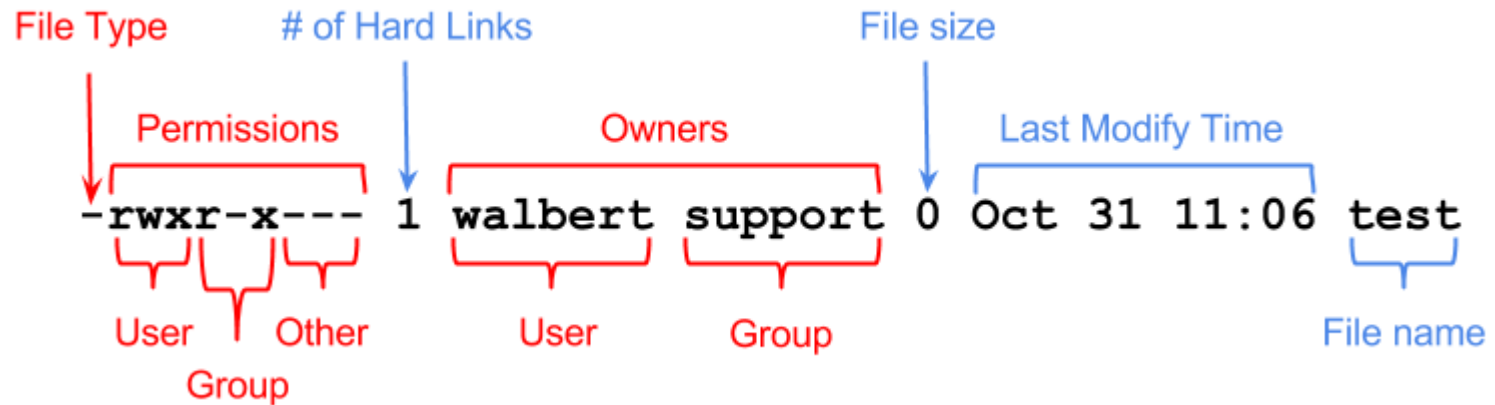
- An OS stores data created by a user in the form of files and directories (Linux) / folders (MS)
- This includes that **data content** added by the user, such as
 - File/directory names
 - Content, e.g., media files, word processing files, web downloads, email, etc.
- It also includes **metadata** added by either the OS, or related application, such as
 - OS data such as creation/modification times, file permissions, physical location in data storage
 - Application data such as geographical information, name of file creator
- An OS also provides data services, such as encryption

OS Data – User Data

- User data can be operated on in various ways, such as opened/read, edited, communicated
- Users can view/list sets of files to access them, or view their metadata
- Permissions set by the OS control access to the data

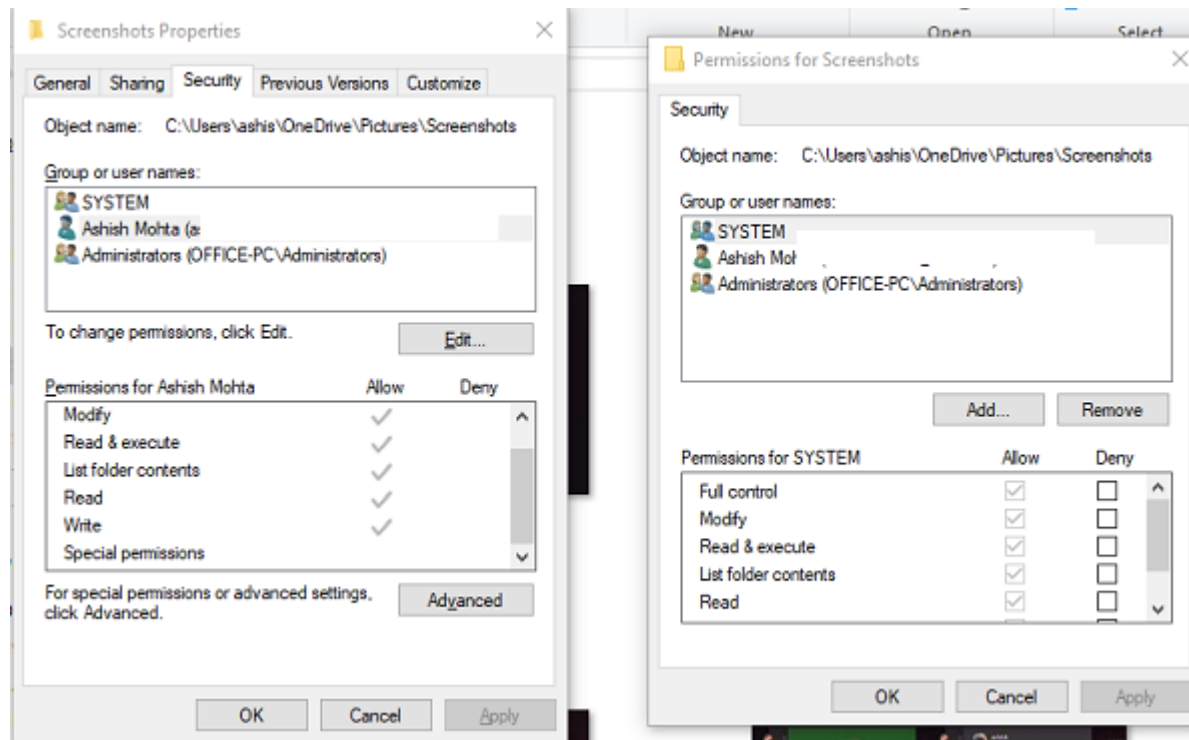
OS Data – User Data

- Example: typical information from listing a file via commandline in Linux



OS Data – User Data

- Example: Viewing and editing file properties in Windows 11



<https://www.thewindowsclub.com/>

OS Data – User Data

- For digital forensics, the same user tools can be used to view a file and its properties
 - Though care needs to be taken to ensure that the tools don't modify the file or its metadata
 - Thus, it's safer to use DF tools
- Further, when a file has been deleted at some point, DF tools are required to determine if the file (or part of it) is recoverable
 - And then to recover it!

OS Data – System Data

- In addition to user data, an OS stores system data that is used by the OS and related applications
- Some types of system data
 - System configuration data
 - Log file data
 - Process data
- System data can also be useful to a DF investigation

OS Data – System Configuration Data

- Data about the operating system, users, network settings, and software
- In Linux, this data is typically in the `/etc/` directory
- `$ cat /etc/os-release/` will show (“catalogue”) information about the computing platform
- The following commands will show information about users and groups on the system

```
$ cat /etc/passwd | column -t -s :
```

```
$ cat /etc/group
```

OS Data – System Configuration Data

- In Windows, one can use the systeminfo command at a command line*

```
PS C:\Users\mj8> systeminfo

Host Name:                               HWLAP0030-MACS
OS Name:                                  Microsoft Windows 11 Enterprise
OS Version:                              10.0.26100 N/A Build 26100
OS Manufacturer:                         Microsoft Corporation
OS Configuration:                        Member Workstation
OS Build Type:                            Multiprocessor Free
Registered Owner:                         Information Services
Registered Organization:                  Heriot-Watt University
Product ID:                               00330-80000-00000-AA179
Original Install Date:                    13/05/2025, 22:17:08
System Boot Time:                         08/12/2025, 09:43:50
System Manufacturer:                      Dell Inc.
System Model:                             Precision 3560
System Type:                              x64-based PC
Processor(s):                             1 Processor(s) Installed.
                                           [01]: Intel64 Family 6 Model 140 Stepping 1 GenuineIntel ~1382 Mhz
BIOS Version:                             Dell Inc. 1.30.0, 12/07/2023
Windows Directory:                        C:\WINDOWS
System Directory:                          C:\WINDOWS\system32
Boot Device:                              \Device\HarddiskVolume1
```

* <https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/systeminfo>

OS Data – System Configuration Data

- In Windows, system configuration data is stored in the **Windows Registry** (database)
- In Linux and Windows, the system configuration information can be viewed and edited (depending on the permissions)
- Care must be taken with editing any system configuration data
- For digital forensics, system configuration data is useful to determine information such as
 - who had access to a system
 - resources available on the system
 - network connections made
 - commands that have been run

OS Data – Log Data

- An OS will log activities recording events that are critical for digital forensics, such as system events (e.g., adding users), authentication logs and application behaviour
- In Linux, log files are stored in `/var/log/`
- In Windows, log files are typically stored in `C:\Windows\System32\winevt\Logs`, and viewed in the Event Viewer application
- Logged information is very useful for activities such as attributing actions, and creating a timeline of actions

OS Data – Process Data

- An OS will use **processes** to manage and coordinate numerous activities, such as opening/terminating an application
- Processes are necessary for scheduling access to resources such as memory and devices such as I/O devices or printers
- Some process actions will be logged, but not all
 - Typically, significant events are logged, such as errors or warnings, though this can vary

OS Data – Process Data

- In Linux, process files are stored in `/proc/` and can be view with commands such as `ps`
- In Windows, the `tasklist` or `get-process` commands can be used, or a variety of applications such as *Process Explorer* or *Resource Monitor*
- Process data is stored in files, however the data is only stored in physical memory (RAM) and not to disk storage
 - Files in `/proc/` have content and can be read but have size 0 (zero) according to the `ls` command!
- Thus, process data is typically only accessible during *Live Forensics*, and not from a disk image

Data Representation

- Through user applications (e.g., file explorer, word processor, browser) we typically view data in the computer as files containing blocks of alphanumeric characters
 - That is, combinations of letters, numbers, and punctuation
- In the OS, data is physically stored in memory using bits (0 or 1) and combinations of bits (e.g., 8 bits for 1 byte)

Data Representation

- For digital forensics, it is important to understand how data is stored and represented in lower levels of the computer
- From the physical storage using bits and bytes, a DF expert that interfaces with an OS will typically encounter other representations such as hexadecimal and octal (as well as binary and decimal)
- We refer to these representations as Number Systems

Common Number Systems

System	Base	Symbols	Used by Humans	Used by DF experts	Used in Computer Storage
Decimal	10	0, 1, ... 9	Yes	Yes	No
Binary	2	0, 1	No	Yes	Yes
Octal	8	0, 1, ... 7	No	Yes	No
Hexadecimal	16	0, 1, ... 9, A, B, ... F	No	Yes	No

JULIA EVANS
@b0rk

hexadecimal

binary numbers are
hard to read

1101010111011001 1101010111001001



it's hard to tell if these are different!

Also, binary numbers are too long: the
number 4,000,000,000 is this in binary:

11101110011010110010100000000000

decimal has
problems too

11010101 = 213

11000101 = 197

These are easier to read, but 213
and 197 look COMPLETELY
different, even though the binary
numbers only differ by 1 bit.

This is bad if you're trying to think
about the binary representation.

hexadecimal makes
binary data more
readable

Every hexadecimal digit represents 4
bits. So 1 byte (8 bits) is 2 digits.

1101 0101 1100 0101
d 5 c 5

11010101 = d5 ← it's much easier to
11000101 = c5 ← see the similarities!

there are 16 hex digits

binary	decimal	hex	binary	decimal	hex
0000	0	0	1000	8	8
0001	1	1	1001	9	9
0010	2	2	1010	10	a
0011	3	3	1011	11	b
0100	4	4	1100	12	c
0101	5	5	1101	13	d
0110	6	6	1110	14	e
0111	7	7	1111	15	f

0x means it's hex

the ASCII code
for space is 0x20



that starts with
0x, so it means
32 and not 20!



things hexadecimal
is used for

- color codes! (eg 0xFF00FF)
- memory addresses!
- displaying binary data!
(like with hexdump)

Typical Low-Level View

Original File
(simple UTF-8 text file)

xxd Hex Dump

```

: cat abc
abcdefghijklmnopqrstuvwxyz
ABC DEF HIJ KLM NOP QRS TUV WXY Z
1234567890
!@#$%^&*() `~ [] {} \ | < > , . ? /
: xxd abc
00000000: 6162 6364 6566 6768 696a 6b6c 6d6e 6f70  abcdefghijklmnop
00000010: 7172 7374 7576 7778 797a 0a41 4243 2044  qrstuvwxyz.ABC D
00000020: 4546 2048 494a 204b 4c4d 204e 4f50 2051  EF HIJ KLM NOP Q
00000030: 5253 2054 5556 2057 5859 205a 0a31 3233  RS TUV WXY Z.123
00000040: 3435 3637 3839 300a 2140 2324 255e 262a  4567890.!@#$%^&*
00000050: 2829 2060 7e20 5b5d 207b 7d20 5c7c 203c  () `~ [] {} \ | <
00000060: 3e20 2c2e 203f 2f0a  > , . ? / .
: █

```

Line Number in Hexidecimal Hexidecimal Values ASCII Interpretation of Value

Capital A
Binary : 1000001
Decimal: 65
Hexidecimal: 0x41 ComputerHope.com

- View of a text file using xxd application <https://www.computerhope.com/unix/xxd.htm>
- Such views of different representations are common at low-level

ASCII Table

- User data are all represented by characters from the ASCII table
 - ASCII = American Standard Code for Information Interchange
- The standard ASCII table has 128 characters
 - It includes letters, numbers, punctuation
 - As well as “non-printable” characters, e.g., line feeds, escape, delete, tab
 - The 128 characters can be encoded in 7 bits (since $2^7 = 128$)
- Extended tables are used to represent additional characters
 - Such as accented letters
- The total number of characters in such a table is 256, which is encoded in 8 bits (since $2^8 = 256$)

ASCII Table

Original ASCII table with 95 printable characters (of 128 total characters)

<https://github.com/tomgibara/ascii-table>

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0 ^@ NUL NULL	1 ^A SOH START OF HEADING	2 ^B STX START OF TEXT	3 ^C ETX END OF TEXT	4 ^D EOT END OF TRANSM.	5 ^E ENQ ENQUIRY	6 ^F ACK ACKNOWLEDGE	7 ^G BEL BELL	8 ^H BS BACKSP.	9 ^I HT CHARACT. TAB'TION	10 ^J LF LINE FEED	11 ^K VT LINE TAB'TION	12 ^L FF FORM FEED	13 ^M CR CARRIAGE RETURN	14 ^N SO SHIFT OUT	15 ^O SI SHIFT IN
1	16 ^P DLE DATALINK ESCAPE	17 ^Q DC1 DEVICE CONTROL 1	18 ^R DC2 DEVICE CONTROL 2	19 ^S DC3 DEVICE CONTROL 3	20 ^T DC4 DEVICE CONTROL 4	21 ^U NAK NEG. ACKNOWLEDGE	22 ^V SYN SYNCH. IDLE	23 ^W ETB END OF TRANS.	24 ^X CAN CANCEL	25 ^Y EM END OF MEDIUM	26 ^Z SUB SUBSTITUTE	27 ^[ESC ESCAPE	28 ^\ FS INFO. SEP. 4	29 ^] GS INFO. SEP. 3	30 ^^ RS INFO. SEP. 2	31 ^_ US INFO. SEP. 1
2	32 SPACE	33 ^_excl. EXCLAM. MARK	34 ^_quot QUOT. MARK	35 ^_num NUMBER SIGN	36 ^_dollar DOLLAR SIGN	37 ^_percnt PERCENT SIGN	38 ^_amp AMPER-SAND	39 ^_apos APOS-TROPHE	40 ^_lpar LEFT PAREN.	41 ^_rpar RIGHT PAREN.	42 ^_* ASTERISK	43 ^_plus PLUS SIGN	44 ^_comma COMMA	45 ^_- HYPHEN-MINUS	46 ^_. FULL STOP	47 ^_sol SOLIDUS
3	48 0 DIGIT ZERO	49 1 DIGIT ONE	50 2 DIGIT TWO	51 3 DIGIT THREE	52 4 DIGIT FOUR	53 5 DIGIT FIVE	54 6 DIGIT SIX	55 7 DIGIT SEVEN	56 8 DIGIT EIGHT	57 9 DIGIT NINE	58 : COLON	59 ; SEMI-COLON	60 < LS.-THAN SIGN	61 = EQUALS SIGN	62 > GR.-THAN SIGN	63 ? QUEST-ION MARK
4	64 @ COMM'IAL AT	65 A	66 B	67 C	68 D	69 E	70 F	71 G	72 H	73 I	74 J	75 K	76 L	77 M	78 N	79 O
5	80 P	81 Q	82 R	83 S	84 T	85 U	86 V	87 W	88 X	89 Y	90 Z	91 [LEFT SQ. BRACKET	92 \ REVERSE SOLIDUS	93] RT. SQ. BRACKET	94 ^ CIRCUM'X ACCENT	95 _ LOW LINE
6	96 grave GRAVE ACCENT	97 a	98 b	99 c	100 d	101 e	102 f	103 g	104 h	105 i	106 j	107 k	108 l	109 m	110 n	111 o
7	112 p	113 q	114 r	115 s	116 t	117 u	118 v	119 w	120 x	121 y	122 z	123 { L. CURLY BRACKET	124 VERTICAL LINE	125 } R. CURLY BRACKET	126 ~ TILDE	127 ^? DEL DELETE

ASCII Table

Each ASCII character with the different number system representations

<https://catonmat.net/ascii-cheat-sheet>

We'll look more at the numbering systems later in the lecture

Dec	Hex	Oct	Bin	Char	Dec	Hex	Oct	Bin	Char	Dec	Hex	Oct	Bin	Char	Dec	Hex	Oct	Bin	Char
0	0x00	000	00000000	NUL	32	0x20	040	01000000	space	64	0x40	100	10000000	@	96	0x60	140	11000000	`
1	0x01	001	00000001	SOH	33	0x21	041	01000001	!	65	0x41	101	10000001	A	97	0x61	141	11000001	a
2	0x02	002	00000010	STX	34	0x22	042	01000010	"	66	0x42	102	10000010	B	98	0x62	142	11000010	b
3	0x03	003	00000011	ETX	35	0x23	043	01000011	#	67	0x43	103	10000011	C	99	0x63	143	11000011	c
4	0x04	004	00000100	EOT	36	0x24	044	01000100	\$	68	0x44	104	10000100	D	100	0x64	144	11000100	d
5	0x05	005	00000101	ENQ	37	0x25	045	01000101	%	69	0x45	105	10000101	E	101	0x65	145	11000101	e
6	0x06	006	00000110	ACK	38	0x26	046	01000110	&	70	0x46	106	10000110	F	102	0x66	146	11000110	f
7	0x07	007	00000111	BEL	39	0x27	047	01000111	'	71	0x47	107	10000111	G	103	0x67	147	11000111	g
8	0x08	010	00001000	BS	40	0x28	050	01010000	(72	0x48	110	10010000	H	104	0x68	150	11010000	h
9	0x09	011	00001001	TAB	41	0x29	051	01010001)	73	0x49	111	10010001	I	105	0x69	151	11010001	i
10	0x0A	012	00001010	LF	42	0x2A	052	01010010	*	74	0x4A	112	10010010	J	106	0x6A	152	11010010	j
11	0x0B	013	00001011	VT	43	0x2B	053	01010011	+	75	0x4B	113	10010011	K	107	0x6B	153	11010011	k
12	0x0C	014	00001100	FF	44	0x2C	054	01010100	,	76	0x4C	114	10010100	L	108	0x6C	154	11010100	l
13	0x0D	015	00001101	CR	45	0x2D	055	01010101	-	77	0x4D	115	10010101	M	109	0x6D	155	11010101	m
14	0x0E	016	00001110	SO	46	0x2E	056	01010110	.	78	0x4E	116	10010110	N	110	0x6E	156	11010110	n
15	0x0F	017	00001111	SI	47	0x2F	057	01010111	/	79	0x4F	117	10010111	O	111	0x6F	157	11010111	o
16	0x10	020	00100000	DLE	48	0x30	060	01100000	0	80	0x50	120	10100000	P	112	0x70	160	11100000	p
17	0x11	021	00100001	DC1	49	0x31	061	01100001	1	81	0x51	121	10100001	Q	113	0x71	161	11100001	q
18	0x12	022	00100010	DC2	50	0x32	062	01100010	2	82	0x52	122	10100010	R	114	0x72	162	11100010	r
19	0x13	023	00100011	DC3	51	0x33	063	01100011	3	83	0x53	123	10100011	S	115	0x73	163	11100011	s
20	0x14	024	00100100	DC4	52	0x34	064	01100100	4	84	0x54	124	10100100	T	116	0x74	164	11100100	t
21	0x15	025	00100101	NAK	53	0x35	065	01100101	5	85	0x55	125	10100101	U	117	0x75	165	11100101	u
22	0x16	026	00100110	SYN	54	0x36	066	01100110	6	86	0x56	126	10100110	V	118	0x76	166	11100110	v
23	0x17	027	00100111	ETB	55	0x37	067	01100111	7	87	0x57	127	10100111	W	119	0x77	167	11100111	w
24	0x18	030	00110000	CAN	56	0x38	070	01110000	8	88	0x58	130	10110000	X	120	0x78	170	11110000	x
25	0x19	031	00110001	EM	57	0x39	071	01110001	9	89	0x59	131	10110001	Y	121	0x79	171	11110001	y
26	0x1A	032	00110010	SUB	58	0x3A	072	01110010	:	90	0x5A	132	10110010	Z	122	0x7A	172	11110010	z
27	0x1B	033	00110011	ESC	59	0x3B	073	01110011	;	91	0x5B	133	10110011	[123	0x7B	173	11110011	{
28	0x1C	034	00110100	FS	60	0x3C	074	01110100	<	92	0x5C	134	10110100	\	124	0x7C	174	11110100	
29	0x1D	035	00110101	GS	61	0x3D	075	01110101	=	93	0x5D	135	10110101]	125	0x7D	175	11110101	}
30	0x1E	036	00110110	RS	62	0x3E	076	01110110	>	94	0x5E	136	10110110	^	126	0x7E	176	11110110	~
31	0x1F	037	00110111	US	63	0x3F	077	01110111	?	95	0x5F	137	10110111	_	127	0x7F	177	11110111	DEL

Data Storage

Computer Memory

- There are generally two types of memory:
 - Primary memory
 - Secondary storage media
- **Primary memory** includes random-access memory (RAM) and cache memories
 - It is volatile memory
 - Data typically remains in memory so long as power is supplied
 - Removal of power doesn't necessarily cause the immediate disappearance of memory (which is important for a quick-acting DF expert)

Computer Memory

- **Secondary storage** media includes hard disks/drives, CDs/DVDs, SSDs (solid state drives), memory cards
 - It is non-volatile storage
 - Data remains in storage for long-term preservation
 - Though data can be erased, and storage does degrade over time
- Secondary storage is viewed by user applications (through the OS) through different drives, e.g., C:/drive
 - Might be multiple, named drives on a single hard drive (e.g., when the drive is partitioned)

Computer Memory

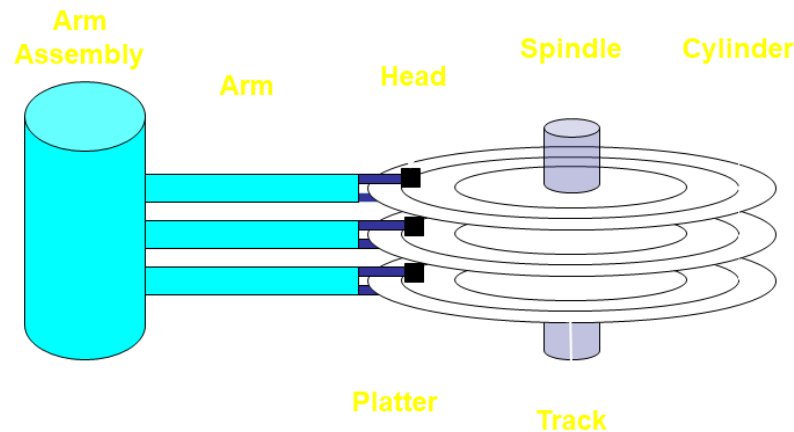
- Physically, the storage of data happens in one of three ways
 - Electromagnetism (hard drives)
 - Microscopic electrical transistors (flash), and
 - Reflecting light (CDs, DVDs, etc.)
- We'll mostly focus on hard drives for the remaining discussion of computer storage

Secondary Storage Interfaces

- There are several standard interfaces to hard disk drives (HDDs), including
 - SAS (Serial Attached SCSI*). Standard for enterprise HDDs
 - SATA (Serial ATA**). Standard for non-enterprise HDDs.
- Interfaces to SSDs include SATA, but more commonly today interfaces such as PCIe/NVMe and M.2
- Not surprisingly, modern interfaces tend to be faster
- SCSI (Small Computer System Interface) is an older interface.
- ATA (Advanced Technology Attachment) is an older interface.

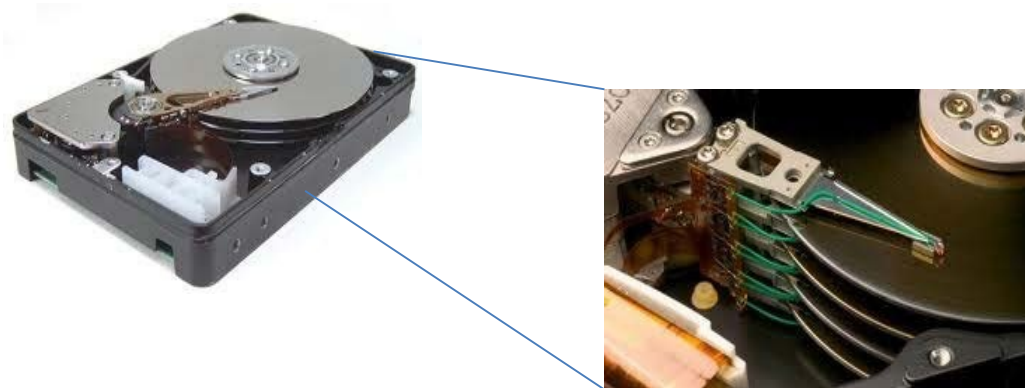
Hard Disk

- Contain one or more circular platters that can be stacked on top of each other and spin at the same time
- The bottom and top of each platter is coated with a magnetic media



Hard Disk

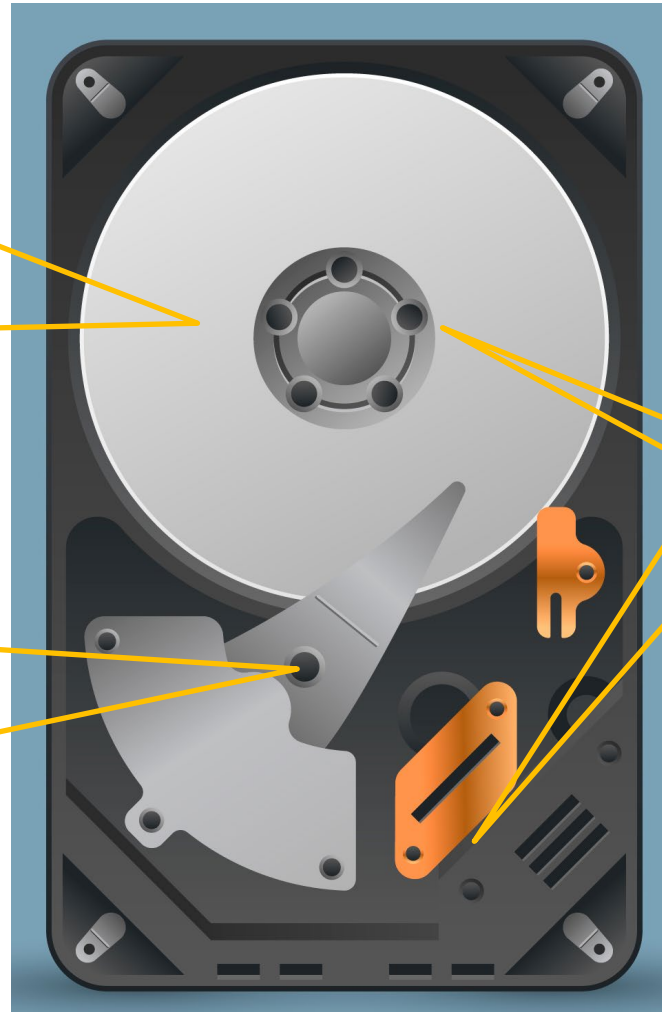
- Inside the disk is an arm that moves back and forth, and it has a head on the top and bottom of each platter that can read and write data



How Hard Disk Works

Step 3: When applications request disk access, the read/write heads move to the FAT/NTFS to determine the current or new location of the data

Step 4: The head actuator positions the read/write head arms over the correct location on the platters to read/write data

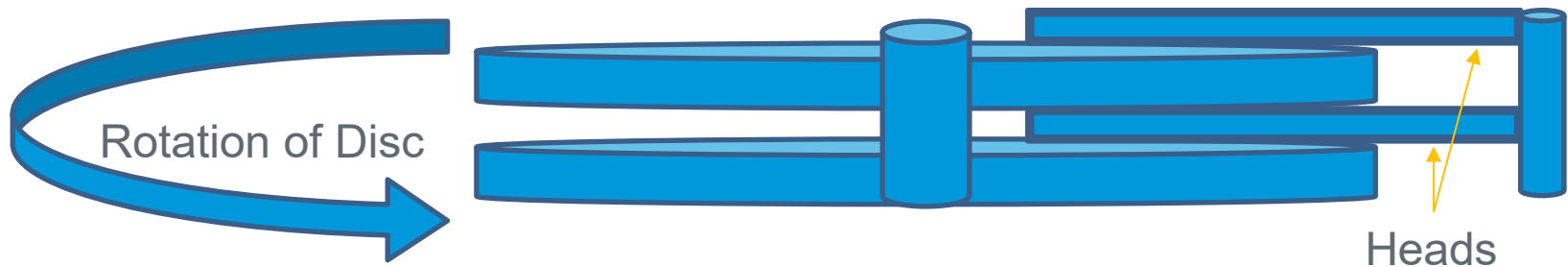


Step 1: Circuit board controls the movement of the head actuator and a small motor

Step 2: A small motor spins the platters the entire time the computer is running

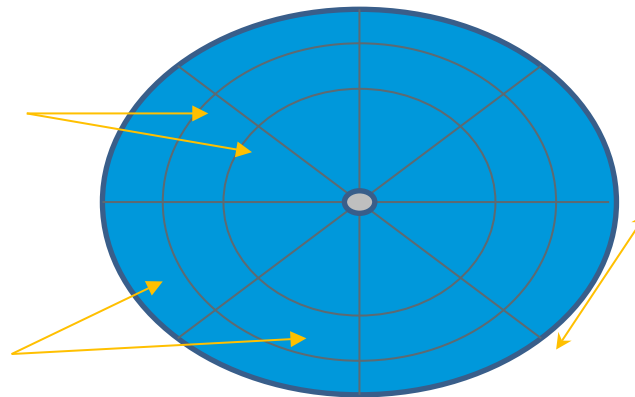
Image by [Freepik](#)

Physical Structure of Disk



Track: a narrow circular strip read by a head. When more than one track is present, i.e. more than 1 platter this is known as a cylinder

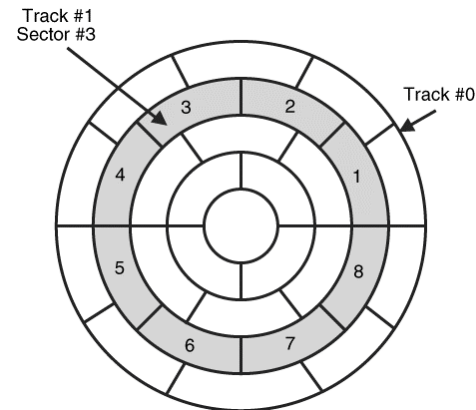
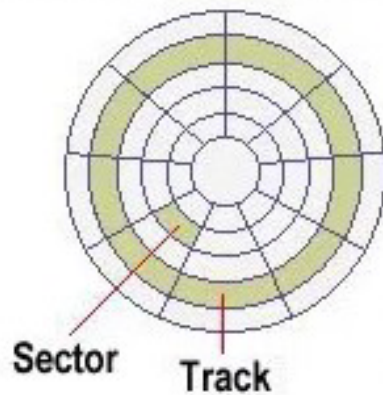
Cluster: the minimum space allocated to the storage of data (e.g. file)



Sector: a subdivision of a track containing a fixed amount of data, typically 512 bytes

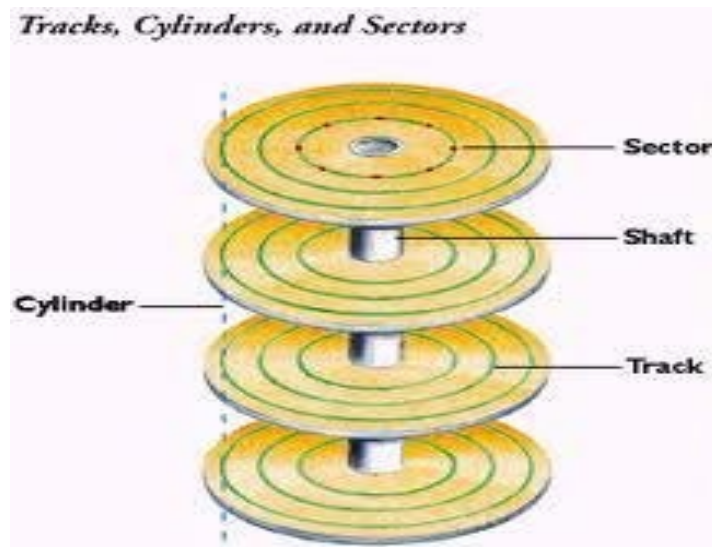
Magnetic Disks

- Each platter is divided in concentric **tracks**
- Each track divided in **sectors**, which is the
 - smallest addressable storage unit, typically 512 bytes
 - basic unit of data storage on a hard disk



Magnetic Disks

- A vertical set of tracks among all disks is called a **cylinder**



Typical Hard Disks

- Platter diameters: 3.7", 3.3", 2.6"
- Number of platters: 1-5
- Aluminum with a deposit of magnetic material
- Some disks also use glass platters
 - E.g. IBM/Hitachi products
 - Better surface uniformity and stiffness but harder to deposit magnetic material

Disk Formatting

- When a new hard disk is built, it's just a slab of magnetic material or empty electrons

Low-level formatting (physical formatting)

- Performed on the blank platters to create data structure for tracks and sectors
- Organises both sides of each platter into tracks and sectors to define where items will be stored on the disk
- Divides a disk into sectors that the disk controller can read and write

Disk Formatting

Logical Formatting (Partitioning)

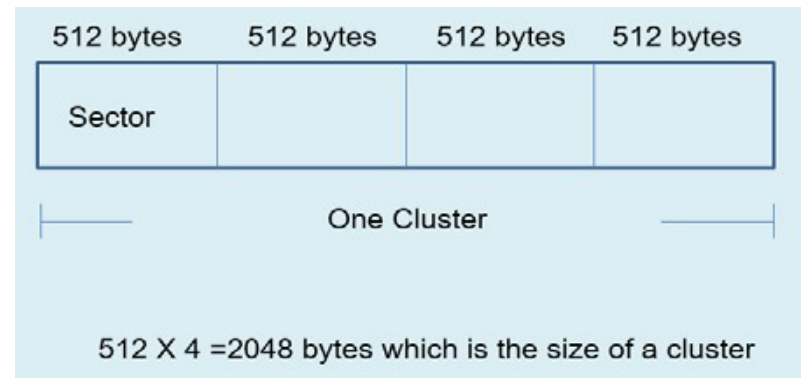
- divides hard disk into separate areas called partitions
- each partition functions as if it were a separate hard disk drive
- Creates the file system data and structures
- Marks all sectors as free space

High-level format

- Depends on file system; FAT, NTFS, EXT, HFS...
- For each partition, which is a table
- Information/addresses used to locate files on the disk

Disk Structure

- A **data unit** is
 - Typically, a set of sectors (2, 4, 8, or more)
 - The smallest amount of space a file can occupy
 - Data units are referred to as either **clusters** (Windows) or **blocks** (Linux)



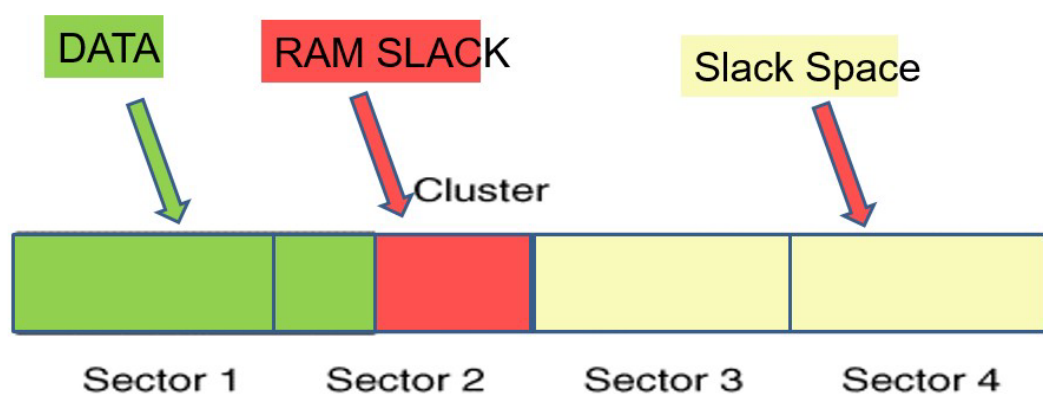
Disk capacity = No of Cylinders × No of Heads × No of Sectors × 512 bytes

Disk Structure

- There are implications to this architecture of data units and sectors
- For example, consider a 512-byte sector and a cluster with 4 sectors
- Now consider storing an 800-byte file. This means there will be
 - One full sector: 512 bytes of data
 - One partial sector: 288 bytes of data, 224 bytes unused
 - Two empty sectors: 512 x 2 bytes of unused data
- This is referred to as **slack space**, and it is important to digital forensics

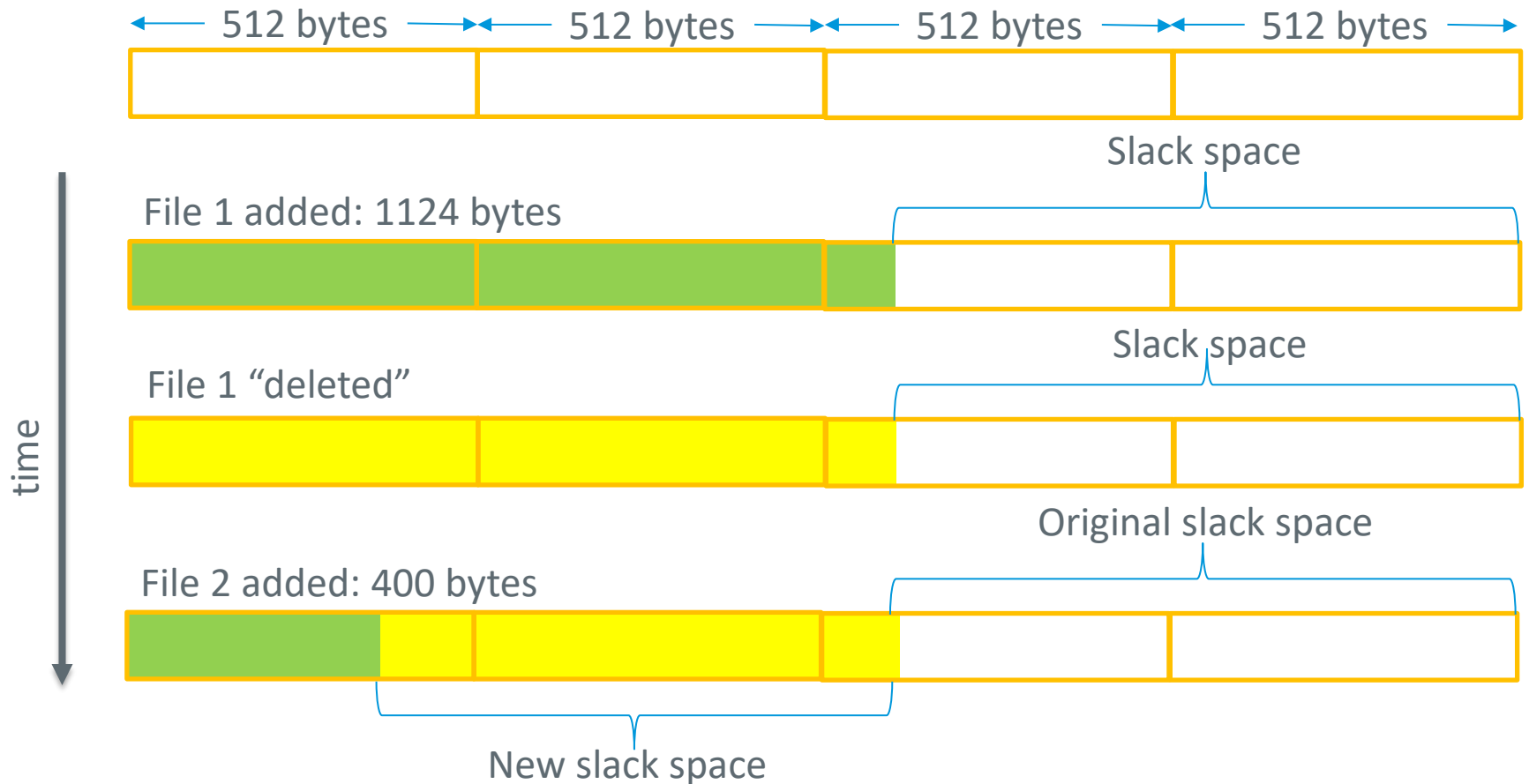
File Slack

- **Slack space** occurs when the size of a file is not multiple of a data unit size
- A file must allocate a full data unit, even if it needs only a small part of the unit
- Older versions of Windows used to temporarily write bytes from RAM to end-of-sector slack, though this is no longer done
- We now refer to all unused data in the data unit as slack space (RAM SLACK + Slack Space from figure below)



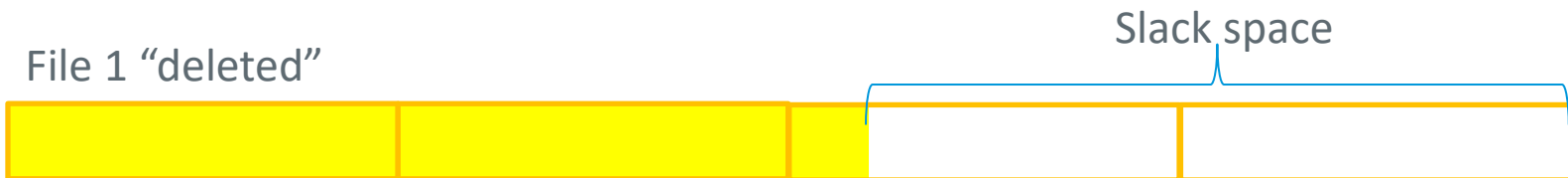
File Slack Example

Consider a 4-sector cluster with 512 byte sectors

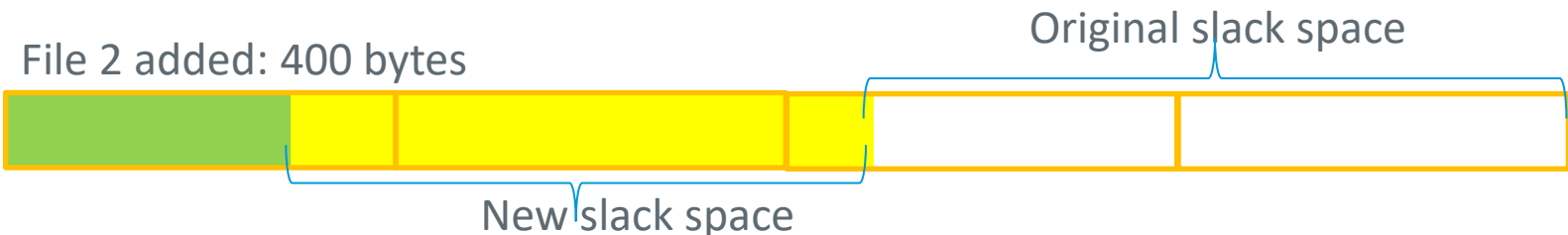


File Slack

- Depending on how File 1 is deleted, it's contents may be recoverable from storage
 - In most cases “deletion” only removes the links to stored data from OS, not overwriting the data
 - Thus, the storage highlighted in yellow is often still recoverable (thus, all of File 1 might be recovered)

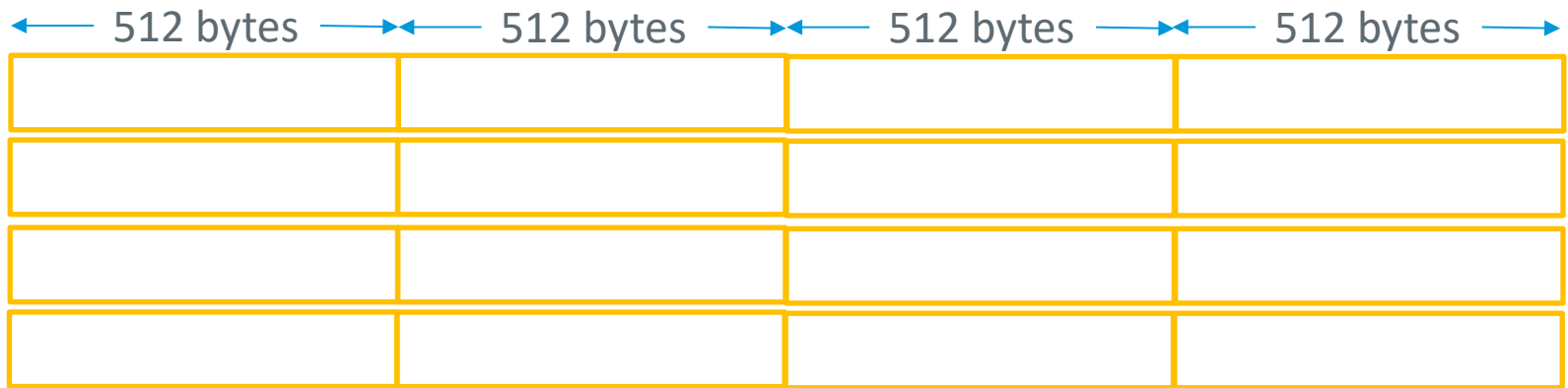


- When a shorter File 2 is added, it creates a “new slack space” (thus, part of File 1 might be recovered)

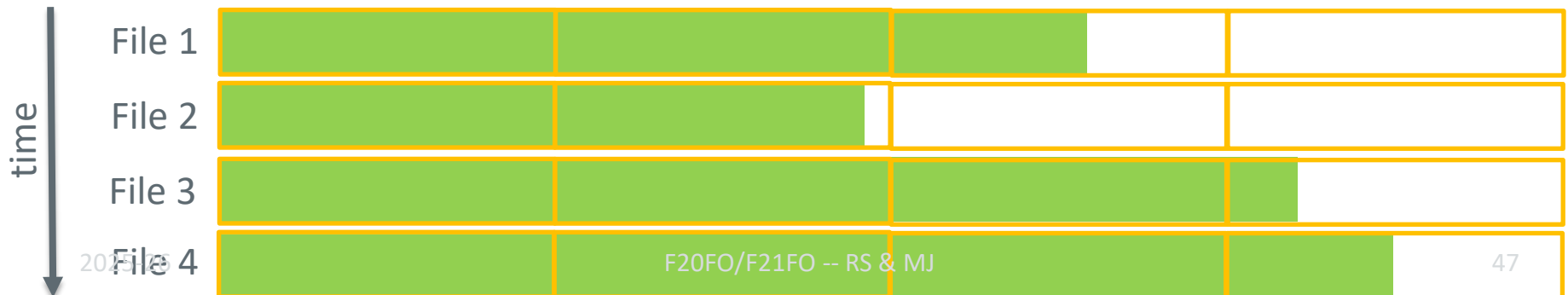


File Fragmentation

- There is a related issue with how data is stored and managed by the OS due to the structure of the data units
- Consider four, 4-sector clusters each with 512-byte sectors
- Consecutive storage read left-to-right and top-to-bottom
- We'll ignore slack space in this example



Files 1-4 added: each various sizes less than 2048 bytes



File Fragmentation

← 512 bytes → ← 512 bytes → ← 512 bytes → ← 512 bytes →

Files 2 and 4 deleted



File 5 added: 2648 bytes



File Fragmentation

- In this case, the newly added File 5 is **fragmented**, meaning that it is stored across non-consecutive data units
- For normal data access, the OS can manage this: it now manages two links/pointers to the location of File 5
 - In older OS versions, fragmentation was a significant issue that would slow OS operations
 - Today, it is managed well with periodic defragmenting by the OS
- However, if File 5 is “deleted” so that the OS no longer has the pointers to the file, recovery of the file for digital forensics can be more difficult since the file exists in multiple data units
 - DF tools will use techniques such as partial OS information, statistical similarities, and pattern matching to group data units together for file recovery

File Fragmentation

- There's a Linux command that can be used to see if a particular file is fragmented
- Consider the two examples, stored in clusters with 8 512-byte sectors
- Example: 6-byte file

```
$ filefrag -b512 -v temp.txt
Filesystem type is: ef53
File size of temp.txt is 6 (8 block of 512 bytes)
  ext: logical_offset:    physical_offset: length: expected: flags:
    0:         0..        7: 1167344.. 1167351:         8:         last,eof
temp.txt: 1 extent found
```

- Notice that even a 6-byte file “reserves” the full size of a data unit (in this case, 8 sectors)

File Fragmentation

- Example: 2,821,185 byte file

```
$ filefrag -b512 -v bigTemp.pdf
Filesystem type is: ef53
File size of bigTemp.pdf is 2821185 (5512 blocks of 512 bytes)
  ext: logical_offset:      physical_offset: length: expected: flags:
    0:      0..      4095: 29945856.. 29949951:    4096:
    1:    4096.. 5511: 29999104.. 30000519:    1416: 29949952: last,eof
bigTemp.pdf: 2 extents found
```

- In this case two “extents” were used to store this file (meaning two data units, which are also referred to as blocks for Linux)
- Note the use of flags to say whether or not a data unit (referred to as an “extent”) is the last
 - This is an example of “partial information” that might be used for a digital forensics file recovery

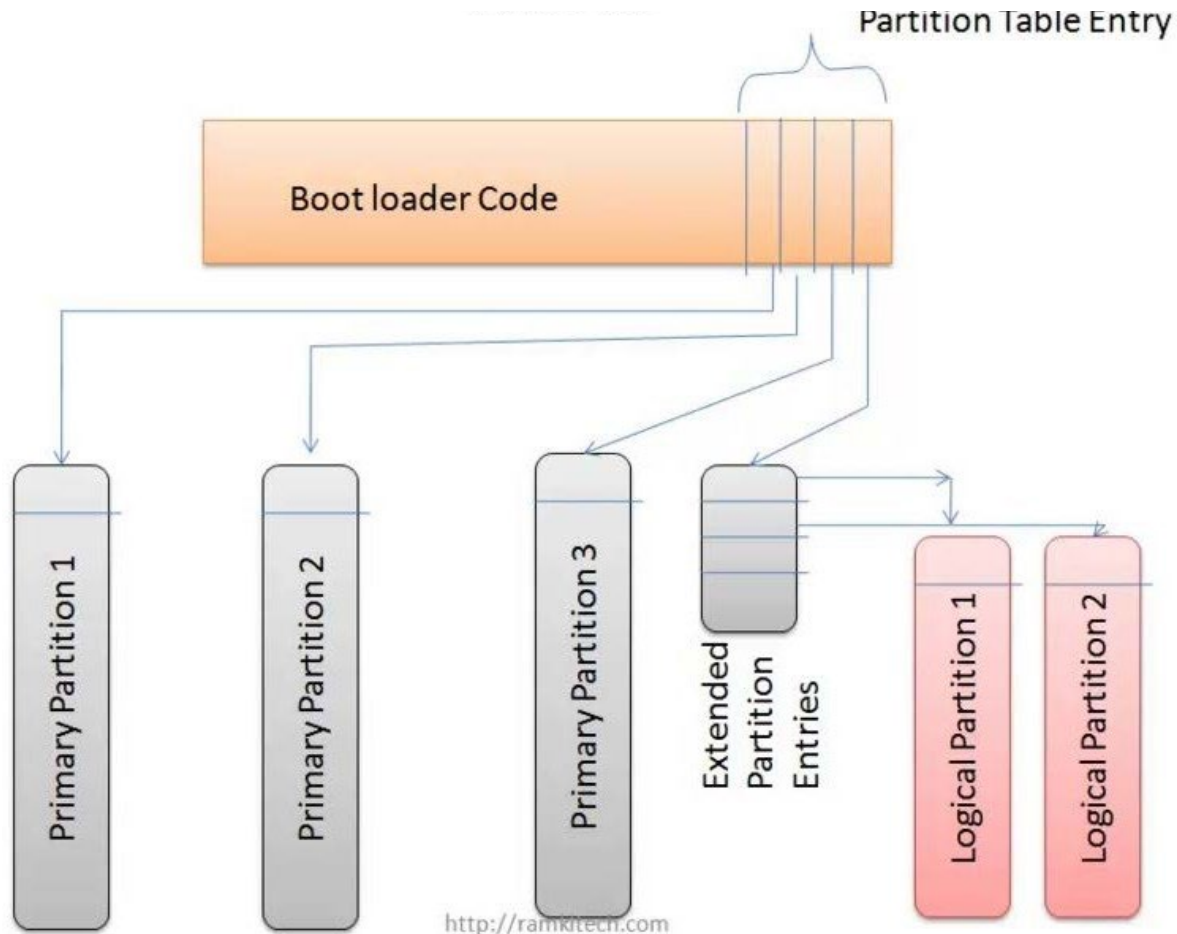
Data Structure on Disk

- Hard disks have a **Master Boot Record (MBR)** in Sector 0
 - defining the partitions on the disk
- Each partition has a **Volume Boot Record (VBR)**
 - defining the details of the partition
 - Each VBR has some form of FAT (file allocation table) defining the location of clusters of data for each file

Overview of HDD



- MBR resides in front of the first partition
 - The first 512-bytes of the disk
 - In Sector 0
- MBR contains
 - Boot code
 - how to process the partition table and how to locate the operating system
 - Partition table
 - 4 entries, one for each partition
- Signature **55 AA**



Boot Loader Code

- The partition that is to be booted is indicated by the bootable flag
 - Standard boot code for a system with only one OS is indicated by a flag set to **0x80**
 - Or the boot code may prompt the user to choose a partition for booting

Structure of MBR

The first 446 bytes

The next 64 bytes (bytes 446-509)

the next 16 bytes is the Partition Table

Offset (d)	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	
00000000000000	33	C0	8E	D0	BC	00	7C	8E	C0	8E	D8	BE	00	7C	BF	00	Sector 0
00000000000016	06	B9	00	02	FC	F3	A4	50	68	1C	06	CB	FB	B9	04	00	
00000000000032	BD	BE	07	80	7E	00	00	7C	0B	0F	85	0E	01	83	C5	10	
00000000000048	E2	F1	CD	18	88	56	00	55	C6	46	11	05	C6	46	10	00	
00000000000064	B4	41	BB	AA	55	CD	13	5D	72	0F	81	FB	55	AA	75	09	
00000000000080	F7	C1	01	00	74	03	FE	46	10	66	60	80	7E	10	00	74	
00000000000096	26	66	68	00	00	00	00	66	FF	76	08	68	00	00	68	00	
00000000000112	7C	68	01	00	68	10	00	B4	42	8A	56	00	8B	F4	CD	13	
00000000000128	9F	83	C4	10	9E	EB	14	B8	01	02	BB	00	7C	8A	56	00	
00000000000144	8A	76	01	8A	4E	02	8A	6E	03	CD	13	66	61	73	1C	FE	
00000000000160	4E	11	75	0C	80	7E	00	80	0F	84	8A	00	B2	80	EB	84	
00000000000176	55	32	E4	8A	56	00	CD	13	5D	EB	9E	81	3E	FE	7D	55	
00000000000192	AA	75	6E	FF	76	00	E8	8D	00	75	17	FA	B0	D1	E6	64	
00000000000208	E8	83	00	B0	DF	E6	60	E8	7C	00	B0	FF	E6	64	E8	75	
00000000000224	00	FB	B8	00	BB	CD	1A	66	23	C0	75	3B	66	81	FB	54	
00000000000240	43	50	41	75	32	81	F9	02	01	72	2C	66	68	07	BB	00	
00000000000256	00	66	68	00	02	00	00	66	68	08	00	00	00	66	53	66	
00000000000272	53	66	55	66	68	00	00	00	66	68	00	7C	00	00	66		
00000000000288	61	68	00	00	07	CD	1A	5A	32	F6	EA	00	7C	00	00	CD	
00000000000304	18	A0	B7	07	EB	08	A0	B6	07	EB	03	A0	B5	07	32	E4	
00000000000320	05	00	07	8B	F0	AC	3C	00	74	09	BB	07	00	B4	0E	CD	
00000000000336	10	EB	F2	F4	EB	FD	2B	C9	E4	64	EB	00	24	02	E0	F8	
00000000000352	24	02	C3	49	6E	76	61	6C	69	64	20	70	61	72	74	69	
00000000000368	74	69	6F	6E	20	74	61	62	6C	65	00	45	72	72	6F	72	
00000000000384	20	6C	6F	61	64	69	6E	67	20	6F	70	65	72	61	74	69	
00000000000400	6E	67	20	73	79	73	74	65	6D	00	4D	69	73	73	68	6E	
00000000000416	67	20	6F	70	65	72	61	74	69	6E	67	20	73	79	73	74	
00000000000432	65	6D	00	00	00	63	7B	9A	B5	9A	45	2F	00	00	00	20	
00000000000448	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00000000000464	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00000000000480	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00000000000496	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	55	AA

Structure of MBR

MASTER BOOT RECORD

Boot Loader Code

Partition Table

Signature Bytes

000	FA	EB	01	00	8C	C8	8E	D8	8E	C0	8E	D0	BC	00	7C	FB	:	·_...i+Ä+Ä+Ä-+.l_
016	BE	00	7C	BF	00	06	B9	00	01	F3	A5	E9	00	8A	BE	B2	:	+..l+...!..._N_.è+
032	07	38	0C	74	3C	BB	00	08	51	0F	B6	0C	BA	80	00	E8	:	.8.t<+...Q...!Ç.
048	A5	00	59	72	21	46	FE	C5	81	C3	00	02	38	0C	75	E8	:	N.Yr!F_+ü+...8.u_
064	33	C0	BE	00	08	03	04	46	46	E2	FA	3B	06	B0	07	0F	:	3++....FF_..:._..
080	85	0E	00	E9	4C	02	BE	10	07	E8	6E	00	BE	6A	07	EB	:	à..._l.+..._n.+j._
096	03	BE	42	07	E8	63	00	33	C0	CD	16	33	C0	BF	00	08	:	..+B._c.3+-..3++..
112	B9	00	7C	F3	AB	BE	AE	07	B9	04	00	83	C6	10	80	3C	:	!..l_?+«..!...â!..Ç<
128	80	74	0B	38	2C	75	02	E2	F2	BE	F6	06	EB	37	8B	D6	:	Çt.8,u.____+..._7i+
144	49	74	09	83	C6	10	38	2C	75	EF	E2	F7	BB	00	7C	8B	:	It.â!..8,u____+.!i
160	F2	8B	14	8B	4C	02	E8	2E	00	72	12	8B	FB	81	BD	FE	:	_i.ilL....x.i_ü+
176	01	55	AA	75	0D	B8	50	00	8B	EE	06	53	CB	BE	10	07	:	.U-u.+P.i_.3-+..._
192	EB	03	BE	27	07	E8	02	00	EB	FE	B4	0E	BB	07	00	AC	:	_.+!.._..._ ..+...%
208	CD	10	38	3C	75	F9	C3	60	BF	05	00	B8	01	02	CD	13	:	-.8<u*+`+...+...-
224	73	0A	4F	74	06	33	C0	CD	13	EB	F0	F9	61	C3	00	00	:	s.0t.3+-..._*a+..
240	00	00	00	00	00	00	0A	0D	50	61	72	74	69	74	69	6F	:Partitio
256	6E	20	74	61	62	6C	65	20	69	6E	76	61	6C	69	64	00	:	n table invalid.
272	0A	0D	45	72	72	6F	72	20	72	65	61	64	69	6E	67	20	:	..Error reading
288	73	65	63	74	6F	72	00	0A	0D	4F	70	65	72	61	74	69	:	sector...Operati
304	6E	67	20	73	79	73	74	65	6D	20	6D	69	73	73	69	6E	:	ng system missin
320	67	00	4D	42	52	20	63	6F	72	72	75	70	74	21	20	52	:	g.MBR corrupt! R
336	75	6E	20	42	6F	6F	74	4D	61	67	69	63	20	63	6F	6E	:	un BootMagic con
352	66	69	67	75	72	61	74	69	6F	6E	0A	0D	50	72	65	73	:	figuration..Pres
368	73	20	61	6E	79	20	6B	65	79	20	74	6F	20	62	6F	6F	:	s any key to boo
384	74	20	61	63	74	69	76	65	20	70	61	72	74	69	74	69	:	t active partiti
400	6F	6E	0A	0D	00	00	00	00	00	00	00	00	00	00	00	00	:	on.....
416	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	:
432	50	51	02	03	04	05	00	03	0D	21	0D	21	00	00	80	01	:	PQ.....!...!...Ç.
448	01	00	0B	FE	7F	09	3F	00	00	00	4B	34	41	00	80	00	:	..._D.?....K4A...
464	C1	FF	0F	FE	FF	FF	EE	3D	D2	01	6B	D1	C1	01	80	00	:	-_.____=-.K--...
480	41	0A	17	FE	FF	91	8A	34	41	00	86	D6	9E	00	80	00	:	A..._æè4A.è+P...
496	C1	92	83	FE	FF	FF	12	0D	E0	00	DC	30	F2	00	55	AA	:	-æâ____..._O_.U-

Boot Loader Code

Partition Table

Signature Bytes

00 : -- Should NOT be used in an actual table entry! It does **not** indicate an *unknown* type, but rather an **empty** entry; in which case, all other *fields* in that 16-byte entry should be *zero-filled* as well.

01 : -- 12-bit FAT (remember: this is on hard disks, not floppy disks, so you'll rarely see this today).

02 : -- XENIX root

03 : -- XENIX */usr* (obsolete)

04 : -- 16-bit FAT, partition; of sizes **less than 32 MB**.

05 : -- **Extended Partition**; within the first 1024 cylinders of a disk drive (also see **0F** below).

06 : -- 16-bit FAT, partition; **greater than or equal to 32 MB**.

07 : -- **Installable file systems: HPFS or NTFS**. Also, QNX and Advanced Unix.

08 : -- AIX bootable partition, AIX (Linux), SplitDrive, OS/2 (through Version 1.3), Dell partition spanning multiple drives (array), Commodore DOS.

09 : -- AIX data partition, AIX bootable (Linux), Coherent file system, QNX.

0A : -- Coherent swap partition, OPUS or OS/2 Boot Manager.

0B : -- 32-bit FAT

0C : -- 32-bit FAT, using INT 13 Extensions.

Relative Offsets (within entry)- byte number for each partition	Length (bytes)	Contents
0	1	Boot Indicator (80h = active)
1 - 3	3	Starting CHS values
4	1	Partition-type Descriptor
5 - 7	3	Ending CHS values
8 - 11	4	Starting Sector read this from right to left as it is stored in little endian
12 - 15	4	Partition Size (in sectors) read this from right to left as it is stored in little endian

368	73	20	61	6E	79	20	6B	65	79	20	74	6F	20				
384	74	20	61	63	74	69	76	65	20	70	61	72	74				
400	6F	6E	0A	0D	00	00	00	00	00	00	00	00	00	00	00	00	00
416	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
432	50	51	02	03	04	05	00	03	0D	21	0D	21	00	00	80	01	00
448	01	00	05	FE	7F	09	3F	00	00	00	45	34	41	00	00	00	00
464	C1	FF	01	FE	FF	FF	EE	3D	D2	01	6B	D1	C1	01	00	00	00
480	41	0A	17	FE	FF	91	8A	34	41	00	86	D6	9E	00	00	00	00
496	C1	92	83	FE	FF	FF	12	0D	E0	00	DC	30	F2	00	55	A9	00

```

on. . . . .
PQ. . . . . ! ! C
. . . _ □ ? . . K4A. . .
- . . . . = - K - . .
A. . . æè 4A. è + P. . .
- 8a . . . . . 0 . U

```

Partition Table Entries

- Bytes 446 – 509
- Each entry in the partition table (16 bytes) has the following fields
 - Starting CHS address
 - Ending CHS address
 - Each uses 10-bits for cylinder, 8-bits for head, 6-bits for sector
 - Starting LBA address
 - Either CHS or LBA is used, but usually not both
 - Number of sectors in partition
 - Type of partition – FAT, NTFS, etc.
 - Flags
 - Identify which partition is bootable, thus which operating system will be loaded

Partition Table

Partition size

Not bootable

Allows 4 partitions

65	6D	00	00	00	00	63	7B	9A	B5	9A	45	2F	00	00	00	20
21	00	07	FE	FF	FF	00	08	00	00	00	58	70	70	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	55

Located at File Offset 446,
for 64 bytes (4 x 16 bytes).

Partition Entry

Byte 0 = Bootable Flag (0x80 or 0x00 – Yes or No)

Byte 1-3 = Starting CHS Address

Byte 4 = Partition Type

Byte 5-7 = End CHS Address

Byte 8-11 = Starting LBA Address

Byte 12-15 = Size in Sectors

416	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
432	50	51	02	03	04	05	00	03	0D	21	0D	21	00	00	50	01
448	01	00	0B	FE	7F	09	3F	00	00	00	4B	34	41	00	00	00
464	C1	FF	0F	FE	FF	FF	EE	3D	D2	01	6B	D1	C1	01	00	00
480	41	0A	17	FE	FF	91	6A	34	41	00	86	D6	9E	00	00	00
496	C1	92	83	FE	FF	FF	12	0D	E0	00	DC	30	F2	00	55	AA

Partition size is in little Endian

- 4B 34 41 00 needs to be read from right to left: 00 41 34 4B
- Convert Hexadecimal to Decimal:

00 41 34 4B (Hex)  4273227 (Dec)

- 4273227 is the partition size in Sector.
- Each Sector is 512 bytes
- Partition size in bytes is: $4273227 \times 512 = 2187892224$
- Size in Kbyte: $2187892224 / 1024 / 1024 = 2,086 \text{ Mbyte} = 2\text{GB}$

The Need to Partition

- OS may not support a logical drive as big as your disk; make the physical drive multiple logical drives
- May not want all of the disk to expire at the same time (system versus user space)
- Multiple OSs on the same disk (multi-boot)
- Different permissions/access
- Different file systems

SSD Drives

- Solid-State Drives
- Uses flash memory to store data
- No spinning disks or read-write head
- Uses both traditional interfaces
 - SATA
- Or newer interfaces
 - mSATA
 - NVMe
- As noted earlier, the program-erase cycles and “wear levelling” on SSDs can impact the ability to recover data from an SSD

How SSD Handles Data

- Stores data in pages
- Pages forms blocks
 - loosely analogous to sectors in HDD
- SSD always writes to empty pages
- Data stored ‘randomly’ in drive

GUID Partition Table

- SSD drives uses GPT
- GPT allows
 - volumes bigger than 2 TB
 - up to 128 partitions
- Can access the 'GPT Protective Partition'

TRIM in SSD

- The TRIM command is a feature of SSD
- The OS informs the SSD which blocks of data are deleted
- All deleted blocks are 'zeroed'
- TRIM enabled by default in internal drives (SATA, eSATA)
- TRIM not enabled:
 - RAID, external SSD
 - not supported by certain OS
- TRIM makes recovery of deleted files more challenging

File Systems

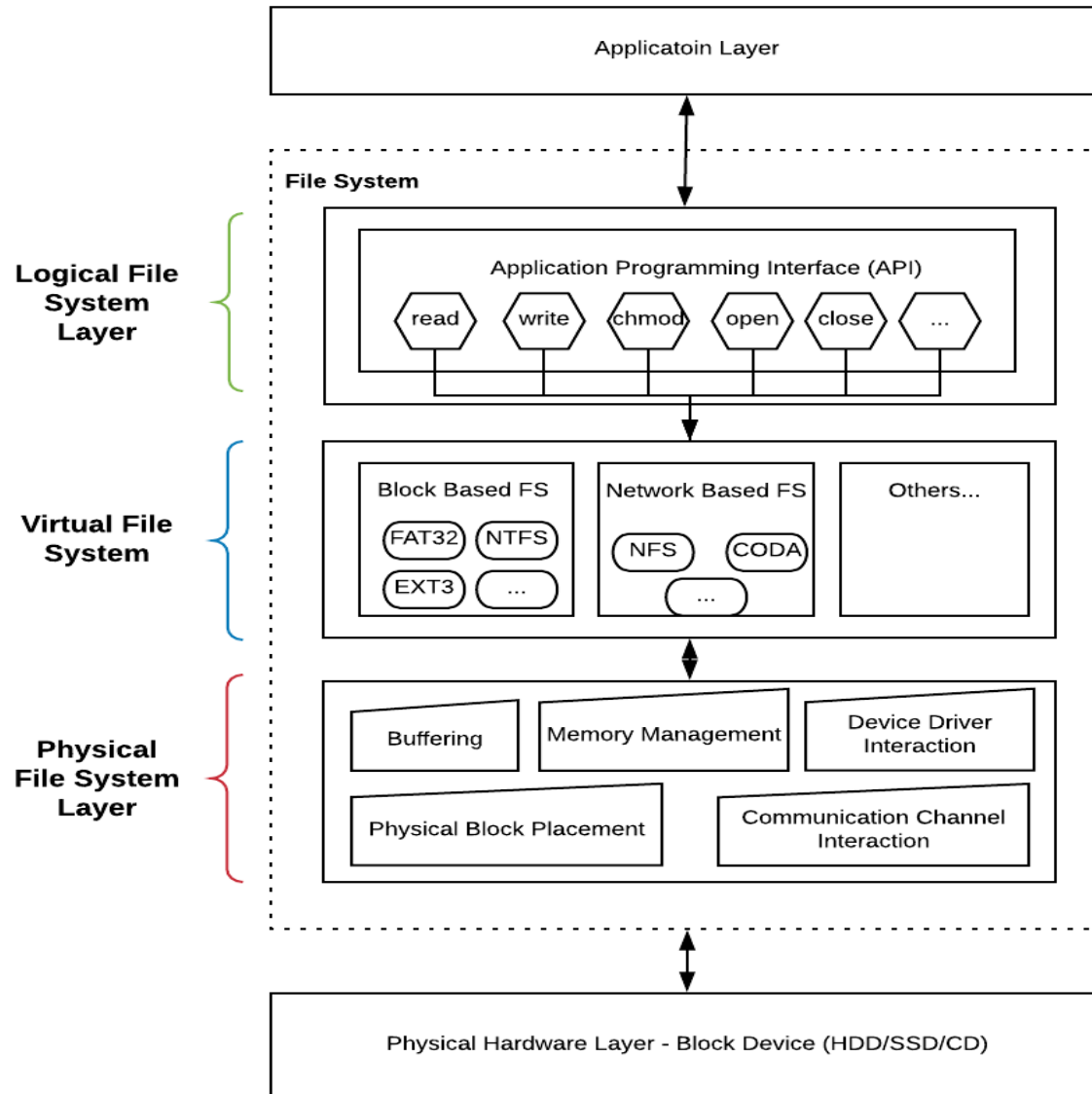
File Systems

- File systems manage how data is *stored* and *retrieved* in a computer system
- They consist of *structural data* and *user data* understood by users and computers
- File system architectures (FAT32, Ext3, etc.) provide different methods of tracking data on physical media with their own data structures and look-up tables

File System Layers

- **Logical Layer**: User application-level API for commands; read, write, chmod, etc.
- **Virtual Layer** (optional): Allow access to multiple physical file systems
- **Physical Layer**: Interacts with hardware, performing block and memory management, device driver interaction, etc.

File System Layers



File System Terminology

- **Sector**: Smallest addressable section of memory which holds a static amount of data (512/2048/4096 bytes)
- **Inode**: Data structure that contains meta data pointers and structures
- **Data Unit**: Standard sized container for storing **content data**. Different systems have different names, e.g., cluster or block

File System Terminology

- Most modern systems typically use a 4096 byte sector size with data units of 1, 2, 4, 8, etc. (powers of 2) sectors
- This will often be referred to as the physical sector size, and may also have a smaller logical sector size (e.g., 512 bytes) for backwards compatibility
 - Thus, from the OS point of view, the sector size is smaller

FS Data Categories

- **File System:** Overview of other data
- **Content:** File contents organised in data units
- **Meta Data:** Describes files, e.g., access times, file sizes, users
- **File Name:** User-friendly name, rather than meta data address
- **Application:** Special features / add'nl functionality, such as quota data or journalling

File System Terminology

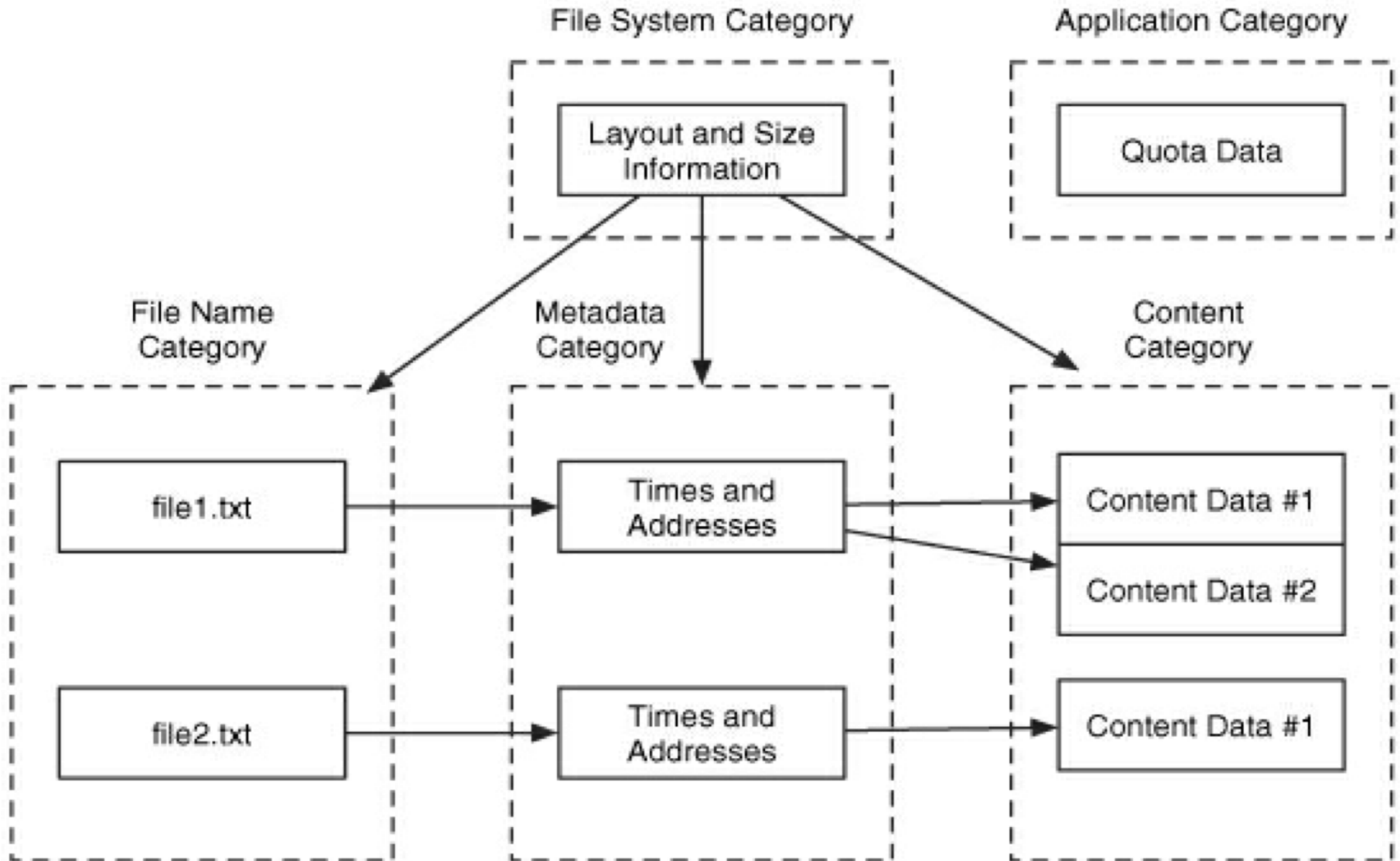
- You can see this in Linux Ubuntu, for example

```
$ lsblk -t /dev/sda
```

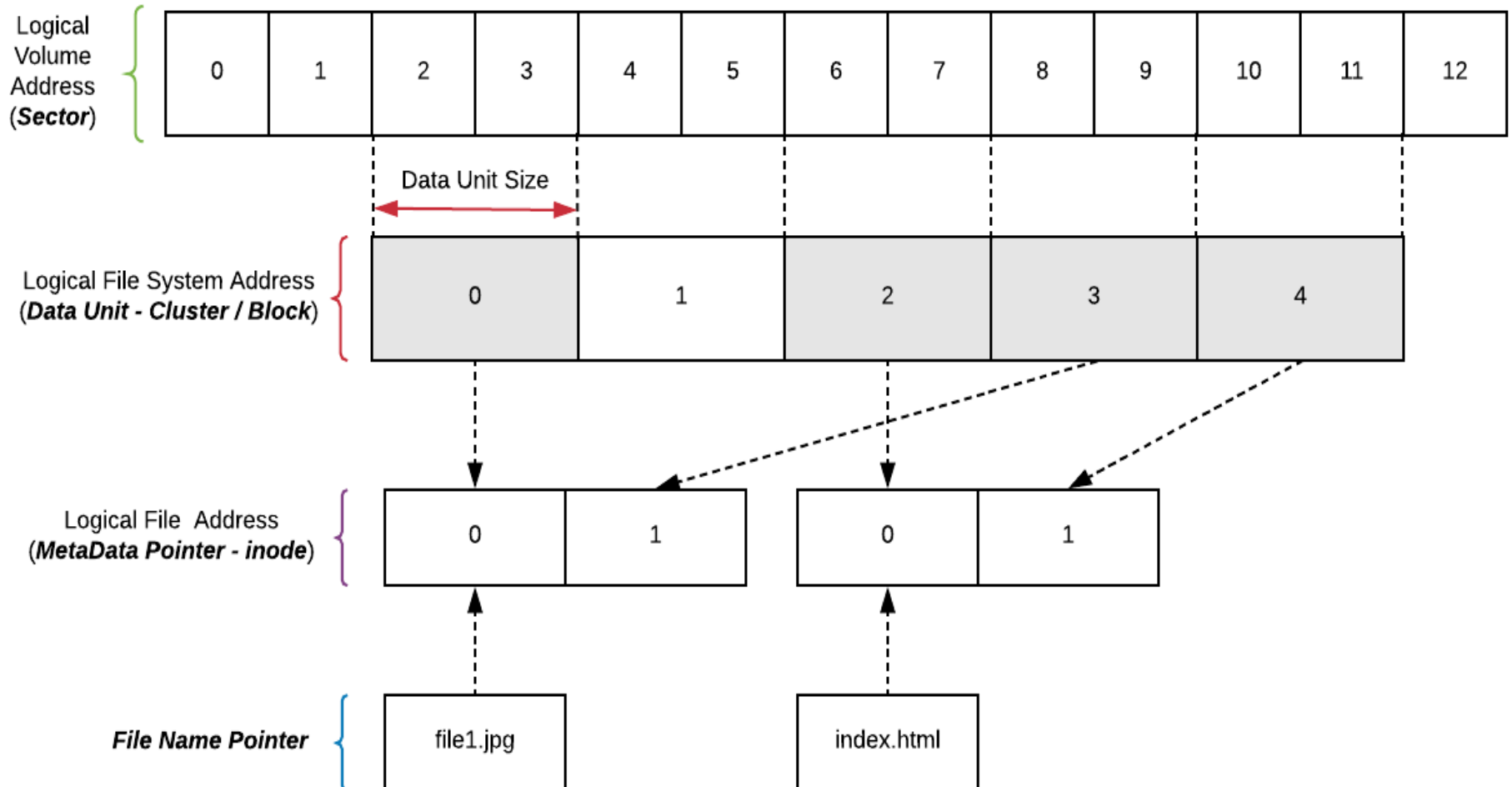
NAME	ALIGNMENT	MIN-IO	OPT-IO	PHY-SEC	LOG-SEC	ROTA	SCHED	RQ-SIZE	RA	WSAME
sda	0	4096	0	4096	512	1	mq-deadline	64 128	0B	
└sda1	0	4096	0	4096	512	1	mq-deadline	64 128	0B	
└sda2	0	4096	0	4096	512	1	mq-deadline	64 128	0B	

- In this case, the `lsblk` (“list block devices”) highlights a physical sector size (`PHY-SEC`) of 4096 bytes, and logical sector size (`LOG-SEC`) of 512 bytes
- In Windows, via a PowerShell, you can consult `C:/Get-PhysicalDisk | Format-List`

FS Categories Interaction



FS Categories By Example



File System Architectures

Examples include:

- **FAT**: File Allocation Table (FAT8/16/32).
Commonly found on removable media
- **NTFS**: New Technology File System. Default for Windows
- **Ext**: Extended File System (Ext2/3/4). Default for Linux.

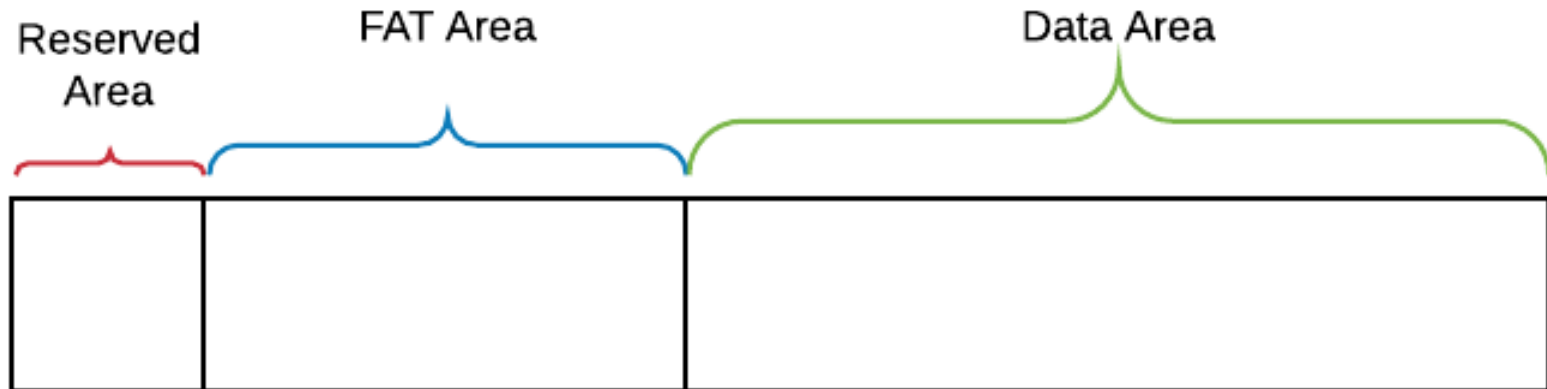
FAT History

- Simplistic FS developed for Windows DOS and Windows 9* operating systems
- Support by both Windows and Unix OS's
- Replaced by NTFS as primary file system for Windows in XP era
- Mainly used now for SD cards and USB flash drives
- Extended from 8-bit to 16-bit, to 32-bit to support larger memory
- Are derivatives, such as exFAT and FAT+

FAT32 Information

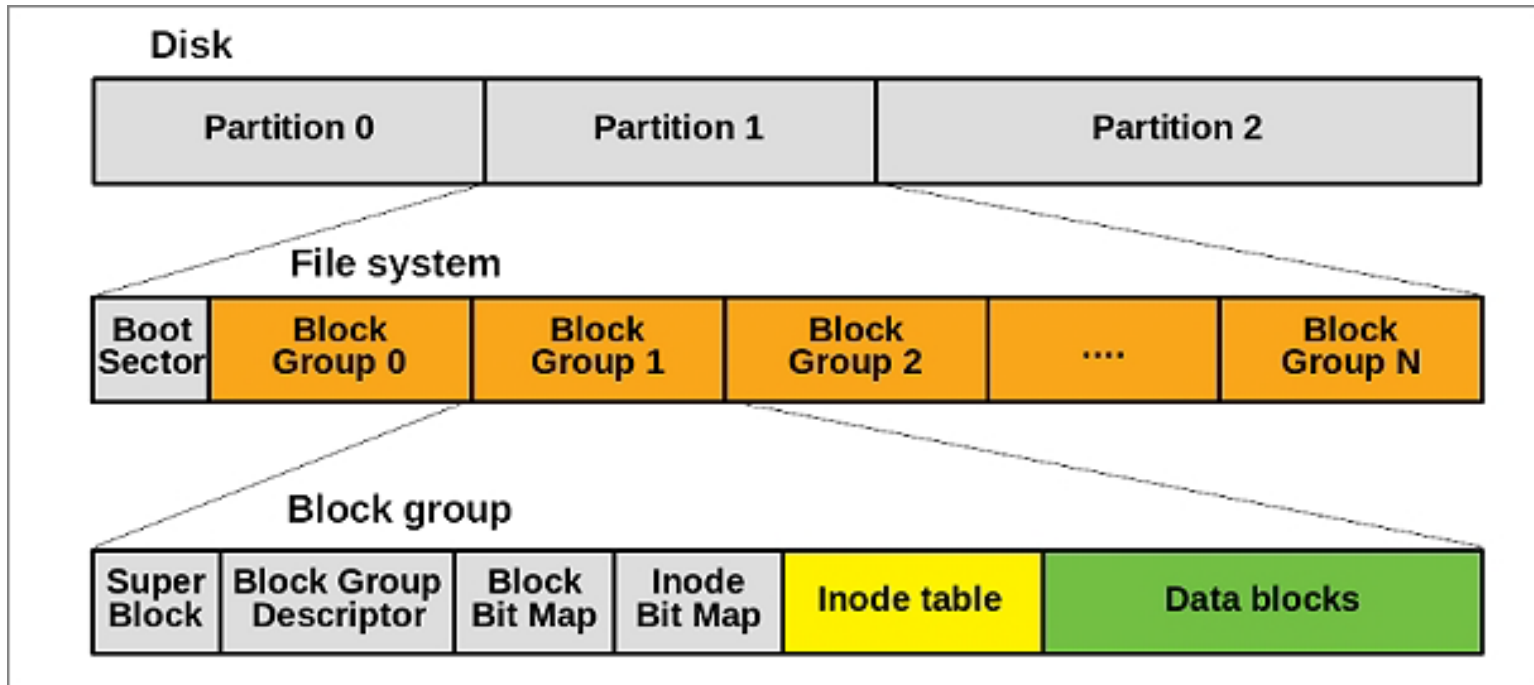
- Data units are called clusters (as opposed to blocks)
- Two overarching data structures
 - **File Allocation Table (FAT)**: Stores next cluster for a file, holds allocation status of clusters
 - **Directory Entries**: one per file/directory containing: file name, size, starting address of content, and other metadata

FAT32 Physical Layout



Ext2 Structure

- File system management and organisation can vary, especially as new file systems aim to be more efficient



Ext2 Structure

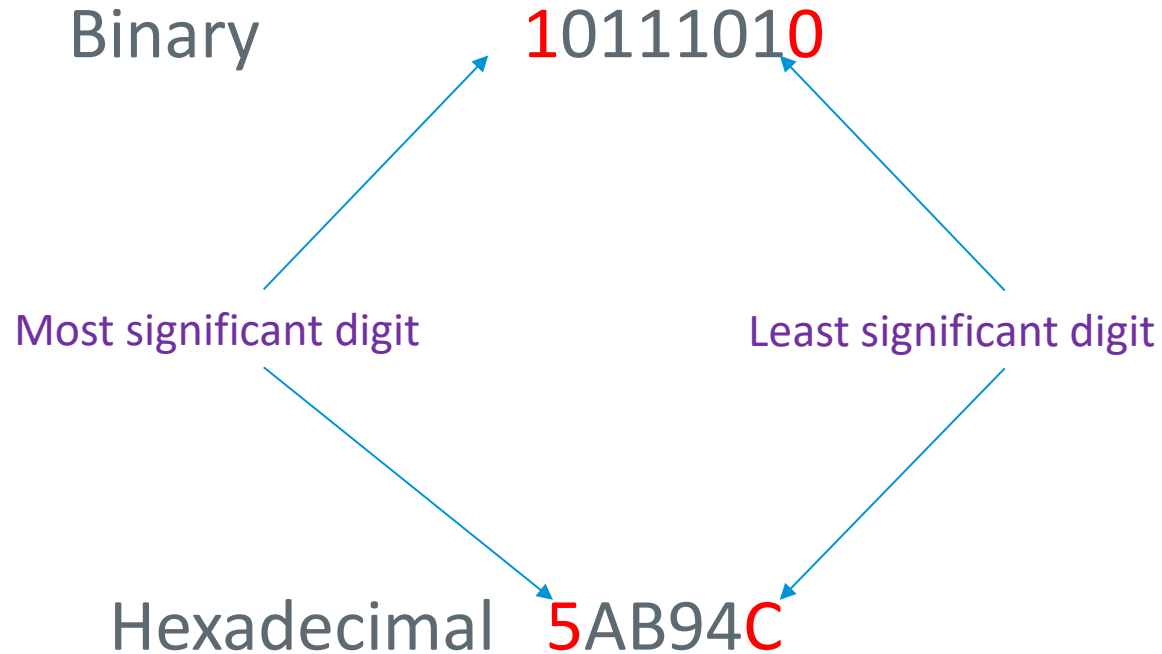
- Note that partition content is split into several block groups
- This provides several advantages, such as
 - Reduced file access times as sectors in block groups are normally physically close, reducing “head seek” times
 - Reduced fragmentation as it aims to store files within single block groups
 - Redundancy as the “super block” information for all groups is repeated across block groups
 - Stability as errors can ideally be localised to block groups
- Reduced fragmentation and stability are advantageous to digital forensics recovery

Numbering Systems

Introduction to Numbering Systems

- We are all familiar with the decimal number system (Base 10)
- Some other number systems that are common in DF
 - Binary (Base 2)
 - Octal (Base 8)
 - Hexadecimal (Base 16)
- Knowledge of other number systems is important for finding and recovering data with digital forensics

Significant Digits



Note: The above two numbers are NOT the same

Counting in Base Systems I

Decimal	Binary	Octal	Hexadecimal
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7

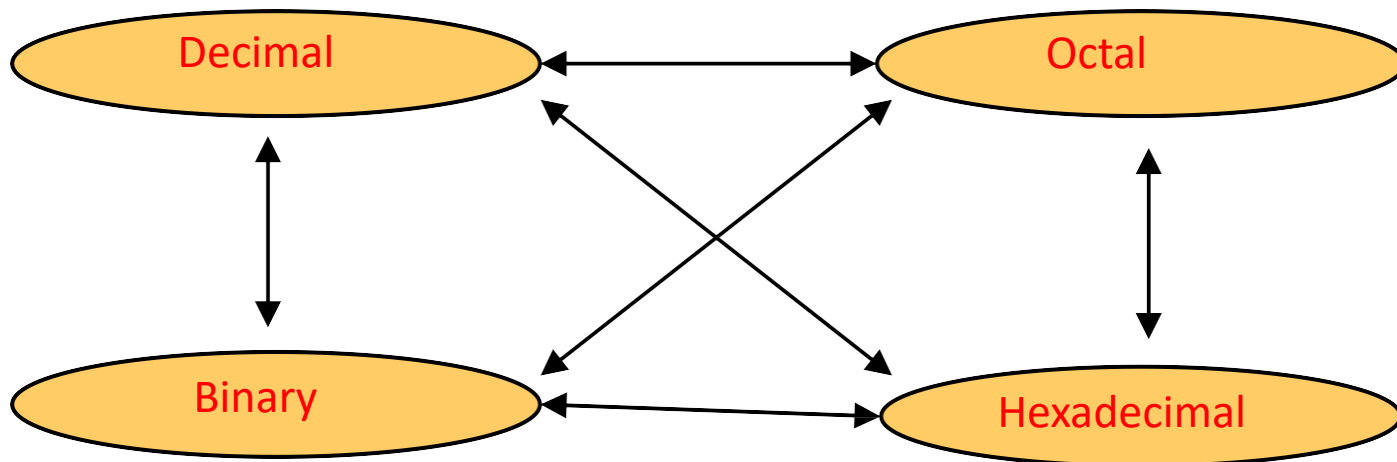
Counting in Base Systems II

Decimal	Binary	Octal	Hexadecimal
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

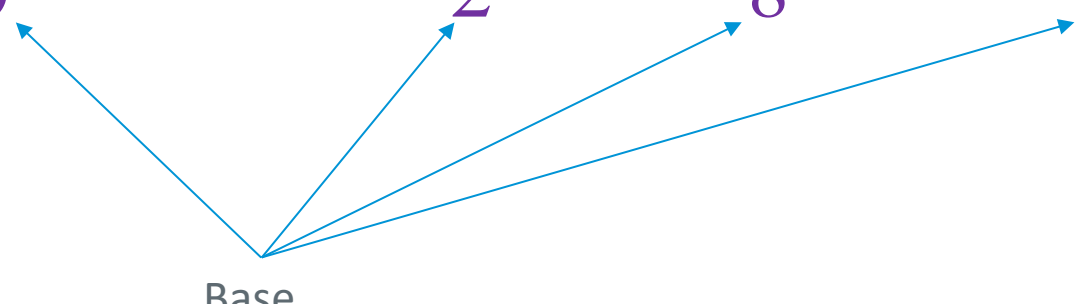
Counting in Base Systems III

Decimal	Binary	Octal	Hexadecimal
16	10000	20	10
17	10001	21	11
18	10010	22	12
19	10011	23	13
20	10100	24	14
21	10101	25	15
22	10110	26	16
23	10111	27	17

Conversion Among Bases



Example

$$25_{10} = 11001_2 = 31_8 = 19_{16}$$


Base

Note: The subscripts above are only for visual representations, and aren't used in low-level displays. For some disambiguation, hex is often written using the prefix "0x", e.g., 0x19 for the hex number "19".

Number System Conversions

- For various reasons, it is often necessary to be able to convert between the different number systems, e.g., given a binary number, you need to compute its value in hex
- There are common techniques used for the conversions, including multiplication, division and group of number characters
- Since binary, octal and hex all have bases that are a power of 2, converting between them is easier
- Converting to/from decimal is a bit more work

Number System Conversions

- Octal and hex are particularly convenient number systems for low level work
- This is because their bases are a power of two
- For octal, there are 8 possible numbers: 0,...,7
 - These 8 numbers can be represented by exactly 3 bits
- For hex, there are 16 possible numbers: 0,...,9,a,...,f
 - These 8 numbers can be represented by exactly 3 bits
- Because computers speak “binary” with two possible values, any number systems based on powers of 2 are very convenient and efficient
 - Decimal is not so convenient, but it is the number system that humans use, and is thus necessary for human understanding

Number System Conversions

- The following slides show how to do the conversions, with some examples
- The slides are grouped into 4 sections
 - Converting to Decimal
 - Converting to Binary
 - Converting to Octal
 - Converting to Hexadecimal (Hex)

Numbering Systems

-- Converting to Decimal

Converting to Decimal

- Can convert binary, octal or hex to decimal using the same process
- General process for converting base b number
 - Multiply each number by b^n , where n is the “weight” of the bit, and b is the base of the original number
 - The weight is the position of the number, starting from 0 on the right
 - Add the results

Binary to Decimal

- Multiply each bit by 2^n , where n is the “weight” of the bit
- The weight is the position of the bit, starting from 0 on the right
- Add the results

Example

$$101011_2 \Rightarrow \begin{array}{rclcl} 1 & \times & 2^0 & = & 1 \\ 1 & \times & 2^1 & = & 2 \\ 0 & \times & 2^2 & = & 0 \\ 1 & \times & 2^3 & = & 8 \\ 0 & \times & 2^4 & = & 0 \\ 1 & \times & 2^5 & = & 32 \\ \hline & & & & 43_{10} \end{array}$$

Octal to Decimal

- Multiply each digit by 8^n , where n is the “weight” of the digit
- The weight is the position of the digit, starting from 0 on the right
- Add the results

Example

$$\begin{array}{rcl} 724_8 & \Rightarrow & \\ 4 \times 8^0 & = & 4 \\ 2 \times 8^1 & = & 16 \\ 7 \times 8^2 & = & 448 \\ \hline & & 468_{10} \end{array}$$

Hexadecimal to Decimal

- Multiply each digit by 16^n , where n is the “weight” of the digit
- The weight is the position of the digit, starting from 0 on the right
- Add the results

Example

$$\begin{array}{rcllcllcll} ABC_{16} & \Rightarrow & C & \times & 16^0 & = & 12 & \times & 1 & = & 12 \\ & & B & \times & 16^1 & = & 11 & \times & 16 & = & 176 \\ & & A & \times & 16^2 & = & 10 & \times & 256 & = & 2560 \\ & & & & & & & & & & \hline & & & & & & & & & & 2748_{10} \end{array}$$

Numbering Systems

-- Converting to Binary

Converting to Binary

- Can convert octal, hex to binary as follows
 - Since both octal and hex have a base that is a power of 2 (octal $2^3=8$, hex $2^4=16$) they each have a natural binary representation
 - For an octal or hex number, we can convert each character individually to binary
- Decimal, which is base 10 (and not a power of 2) takes a bit more work to convert to binary
 - It is the reverse of the calculation (division) used for converting binary to decimal

Decimal to Binary

- Divide by two, keep track of the remainder
- First remainder is bit 0 (LSB, least- significant bit)
- Second remainder is bit 1
- Etc.

Example

$$125_{10} = ?_2$$

2		125	
2		62	1
2		31	0
2		15	1
2		7	1
2		3	1
2		1	1
		0	1

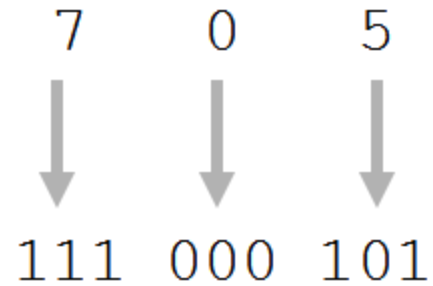
$$125_{10} = 1111101_2$$

Octal to Binary

- Convert each **octal** digit to a **3-bit** equivalent binary representation

Example

$$705_8 = ?_2$$



$$705_8 = 111000101_2$$

Hexadecimal to Binary

- Convert each hexadecimal digit to a 4-bit equivalent binary representation

Example

$$10AF_{16} = ?_2$$

1	0	A	F
↓	↓	↓	↓
0001	0000	1010	1111

$$10AF_{16} = 0001000010101111_2$$

Numbering Systems

-- Converting to Octal

Converting to Octal

- It is quite straightforward to convert binary and hex numbers to octal, simply by grouping bits together
- Converting decimal to octal requires some division, similar to converting decimal to binary
 - It is the reverse of the calculation (division) used for converting octal to decimal


Decimal to Octal

- Divide by 8
- Keep track of the remainder

Example

$$1234_{10} = ?_8$$

8	1234	
8	154	2
8	19	2
8	2	3
	0	2



$$1234_{10} = 2322_8$$

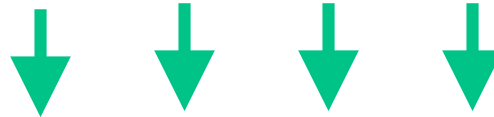
Binary to Octal

- Group bits in threes, starting on right
- Convert to octal digits

Example

$$1011010111_2 = ?_8$$

1 011 010 111



1 3 2 7

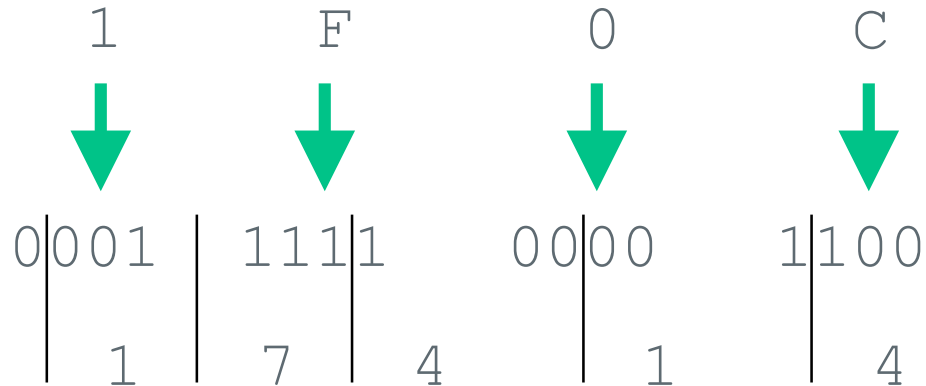
$$1011010111_2 = 1327_8$$

Hexadecimal to Octal

- Use binary as an intermediary

Example

$$1F0C_{16} = ?_8$$



$$1F0C_{16} = 17414_8$$

Numbering Systems

-- Converting to Hex

Converting to Hex

- Conversion to hex uses the same processes as converting to octal
- Thus
 - To convert binary and octal numbers to hex, simply by grouping bits together
 - Converting decimal to hex requires some division, similar to converting decimal to binary or octal
 - It reverses of the calculation (division) used for converting hex to decimal

Decimal to Hexadecimal

- Divide by 16
- Keep track of the remainder

Example

$$1234_{10} = ?_{16}$$

16	1234	
16	77	2
16	4	13 = D
	0	4

$$1234_{10} = 4D2_{16}$$

Binary to Hexadecimal

- Group bits in fours, starting on right
- Convert to hexadecimal digits

Example

$$1010111011_2 = ?_{16}$$

$$\begin{array}{ccc} 10 & 1011 & 1011 \\ \downarrow & \downarrow & \downarrow \\ 2 & B & B \end{array}$$

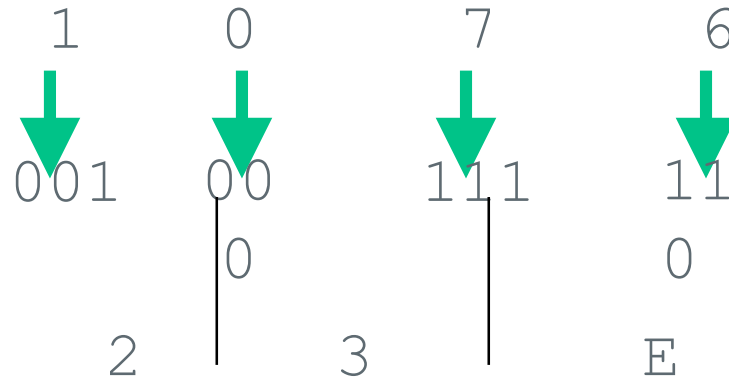
$$1010111011_2 = 2BB_{16}$$

Octal to Hexadecimal

- Use binary as an intermediary

Example

$$1076_8 = ?_{16}$$



$$1076_8 = 23E_{16}$$

Topic overview

- Topic 3 overview: Technical concepts
 - Operating systems
 - Data storage
 - File systems
 - Numbering systems

Next topic

- Topic 4 overview: Evidence types & collection
 - Files and file signatures
 - File signature analysis