



File Types

# Types of files

## *Confusion: Text files vs. Binary Files*

- **Text files (ASCII / UNICODE)**
  - *Bytes of data are organised as characters from respective character sets*
- **Binary files**
  - *Data in a specific format that requires interpretation.*
- **Text files vs. Binary Files**
  - *All files are in Binary*
  - *Text Files are formatted in chunks of 8 bits or 16 bits*
  - *Files in any other format are Binary Files*

# File types

- Most files contain a specific types of information
  - *A Java program*
  - *A JPEG image*
  - *A BITMAP image*
  - *An MP3 clip*
- The kind of information is the **file type**
  - *So the File System knows which operations it can do*
  - *Most OS have associations between file types and applications*

# File Types    *Extensions*

- File names are often separated by a full-stop into 2 parts
  - *Main name*
  - *File extension*
- The **file extension** was used by the OS to identify the type of file
  - *But is not necessarily the actual file type*
- Windows 10 will inspect the file to ascertain the actual file type
  - *Looking at the file header*

Extension	File Type
.txt	Text data file
.mp3, .au, .wav	Audio file
.gif , .tiff , .jpg	Image file
.doc , .odt	Word processing files
.java , .sql	Programming source file

# Anatomy of an ASCII File

The screenshot shows the WinHex application window titled 'WinHex - [textfile.txt]'. The menu bar includes File, Edit, Search, Navigation, View, Tools, Specialist, Options, Window, and Help. The toolbar contains various icons for file operations and editing. The main window displays the file 'textfile.txt' with the following hex and ASCII data:

Offset	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
00000000	54	68	69	73	20	69	73	20	61	20	74	65	78	74	20	66	This is a text f
00000016	69	6C	65														ile

Overlaid on the bottom half of the WinHex window is a 'Hexadecimal / ANSI ASCII' table. This table maps hexadecimal values to their corresponding ASCII characters. The table is organized into rows and columns, with hexadecimal values on the left and the corresponding ASCII characters on the right.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
10	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
20		!	"	#	\$	%	&	'	(	)	*	+	,	-	.	/
30	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
50	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
60	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
70	p	q	r	s	t	u	v	w	x	y	z	{		}	~	
80	€		,	f	"	...	†	‡	^	%	Š	<	€		Ž	
90		,	,	"	"	*	-	-	~	™	š	>	œ		ž	Ÿ
A0		i	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	­	®	¯
B0	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
C0	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
D0	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
E0	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
F0	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

On the right side of the WinHex window, there is a sidebar with file information for 'textfile.txt':

- File size: 19 B (19 bytes)
- Default Edit Mode: original
- Undo level: 0
- Undo reverses: n/a
- Creation time: 17/01/2017 14:48:45
- Last write time: 17/01/2017 14:48:45
- n/a Size: n/a



File  
signature

# Anatomy of a Binary file (*jpeg*)

## *File Headers*

(JFIF)  
JPEG File Interchange Format

The screenshot shows the WinHex application window with the file [64px-A\_Smiley.jpg] open. The main pane displays the hex dump of the file's header. A red box highlights the sequence of bytes: FF D8 FF E0 4A 46 49 46 00. Annotations with arrows point to specific parts of this sequence:

- FF D8 = Start of image marker**: Points to the first byte (FF) at offset 0.
- FF E0 = JFIF marker**: Points to the third byte (FF) at offset 2.
- Partial File Header**: A box spanning from offset 0 to 15, covering the initial bytes of the header.
- JFIF Identifier**: Points to the four-byte sequence 4A 46 49 46 (the 'JFIF' ASCII string in little-endian) starting at offset 6.

The right-hand pane shows the file's metadata, including its name (64px-A\_Smiley.jpg), size (549 KB / 561,997 bytes), and creation time (17/01/2017 15:22:27).

# File signatures

- There file signature databases
  - [\*Filesignatures.net\*](http://filesignatures.net)
- Wikipedia often has high quality listings of the entire file header



# File operations

- Create a file
- Delete a file
- Open a file
- Close a file
- Read data from a file
- Write data to a file
- Reposition the current file pointer in a file
- Append data to the end of a file
- Truncate a file
  - *ie. delete all or part of it*
- Rename a file
- Copy a file

# File protection

- Multi-user Systems
- Access control
  - *Controls who can access files*
    - Who can read
    - Who can write
    - Who can execute

# Cyber-security triad

## ■ Three dimensions of cyber-security:

- *Confidentiality*
- *Integrity*
- *Accessibility*

### **Confidentiality:**

- Preventing access
- Keep the *bad-guy* out

### **Accessibility:**

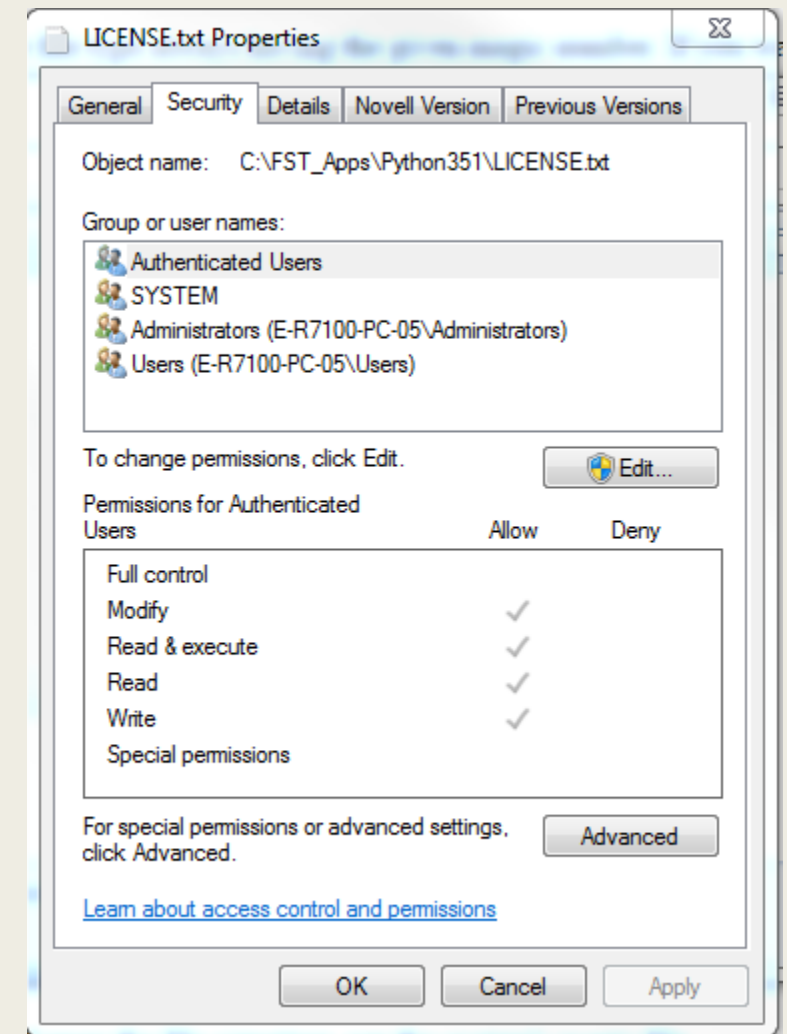
- Ensure access
- Make sure the *good-guy* can access the data

### **Integrity:**

- Keep control of any changes made to the data
- Who can change it
- Keep track of any changes

# File permissions (windows)

- NTFS:
  - *Access Control Lists (ACL's)*
    - Each file has list of user identities with permissions
  - *Explorer*
    - File Permissions
    - Security
      - *Different user, different permissions*
- No multi-user security for FAT32



# Windows permissions classifications

- Full control

- *File can be written to/read from*
- *Permissions can be read and modified*
- *Ownership can be changed*
- *Folder can be listed and entries deleted*

- Modify

- *Same as Full control*
- *But cannot change permissions or ownership*

# Windows permissions classifications

- Read/Execute
  - *File can be read or executed as a program*
  - *Folder can be listed and traversed*
- Read
  - *File can be read*
    - But not executed
  - *Folder can be listed*
    - But not traversed
- Write
  - *File can be modified*
  - *Files/subfolders can be created in a folder*
    - But NOT deleted
- List folder contents (for folders only)
  - *Same as Read/Execute, but not available for files, and only inherited by folders*

# Security inheritance

- Windows:

- *New file or subfolder created, will inherit it's parent's permissions by default*
- *You can override*

- Unix:

- *Permissions are not inherited for newly created files*
- *Based on user's `umask`*
- *Mask of permissions specific to that user – octal absolute format*

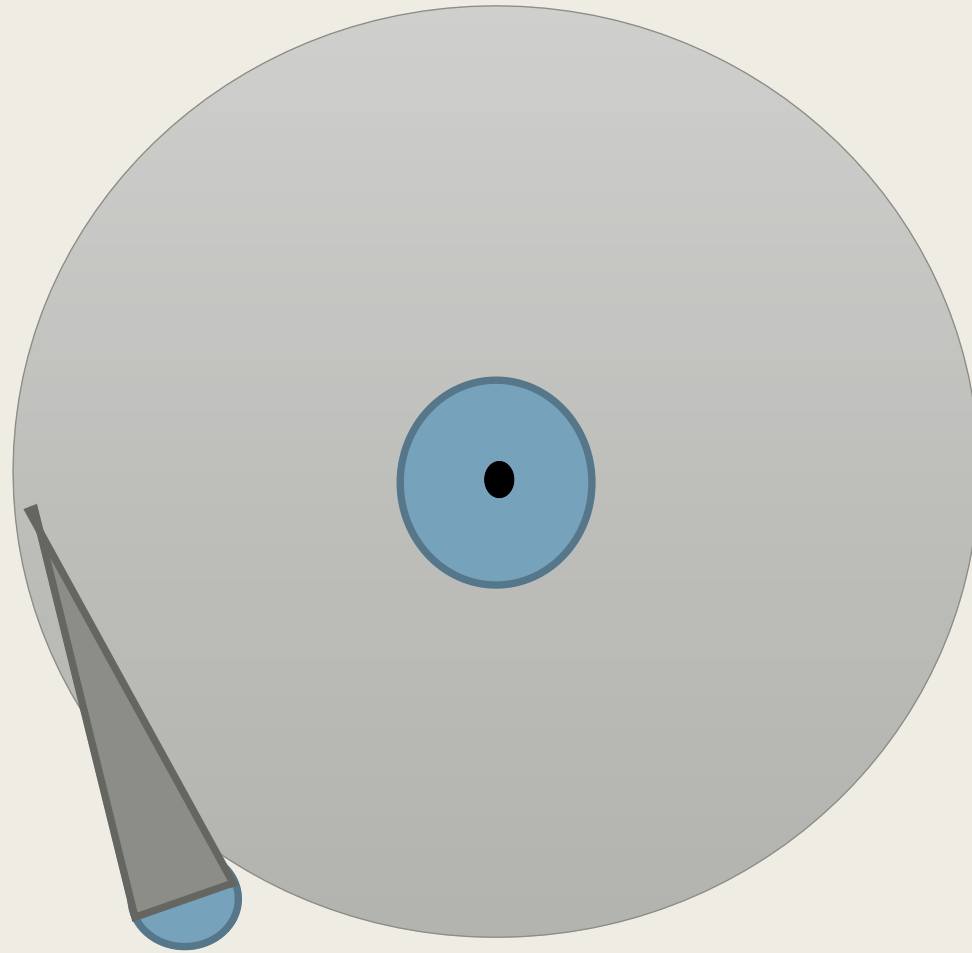
## Disk scheduling



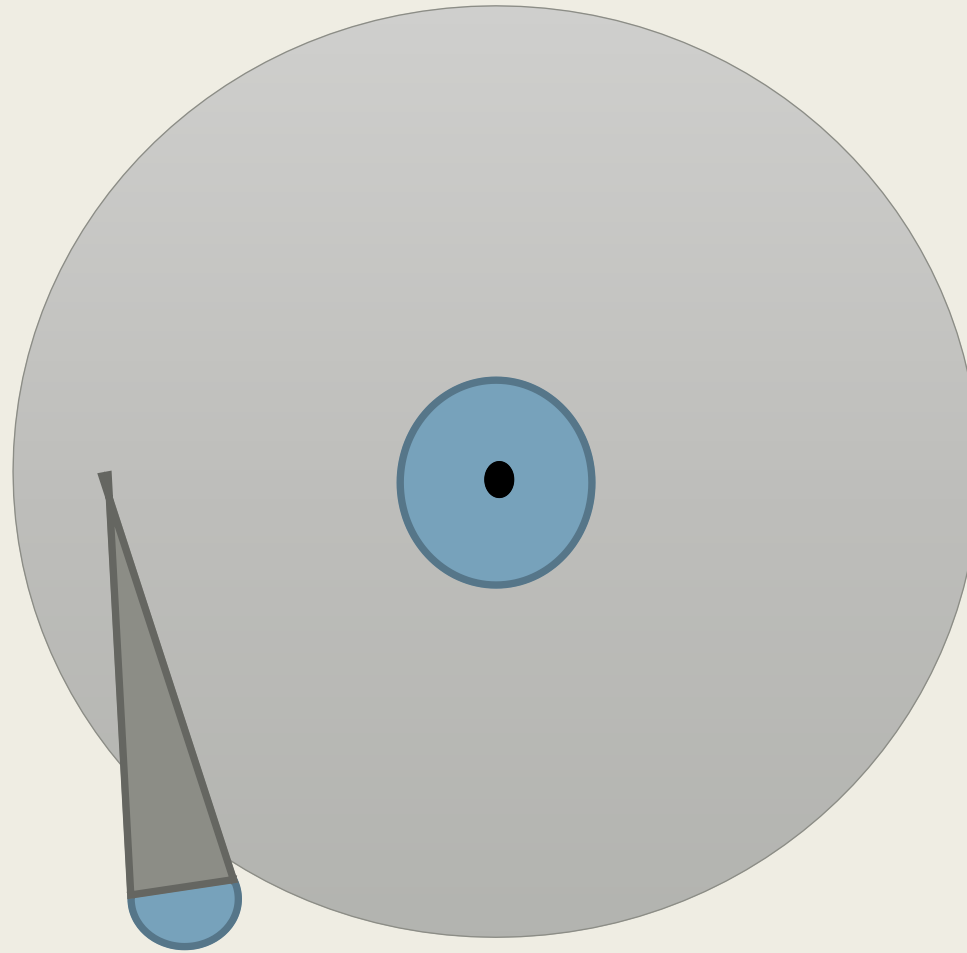
# Disk scheduling

- Must be efficient
- Multiple processes, multiple requests to access disk
- Disk scheduling techniques to manage request:
  - ***First-come, first-served (FCFS)***: Requests are serviced in the order they arrive, irrespective of positions of heads
  - ***Shortest-Seek-Time-first (SSTF)***: Minimise movement of disk heads
  - ***SCAN***: Disk heads continuously move in and out, servicing requests as the locations are found.
  - ***C-SCAN*** : Circular scan
  - ***Look*** : Like SCAN, but does not scan all the way to edge
  - ***C-Look*** : Like C-SCAN

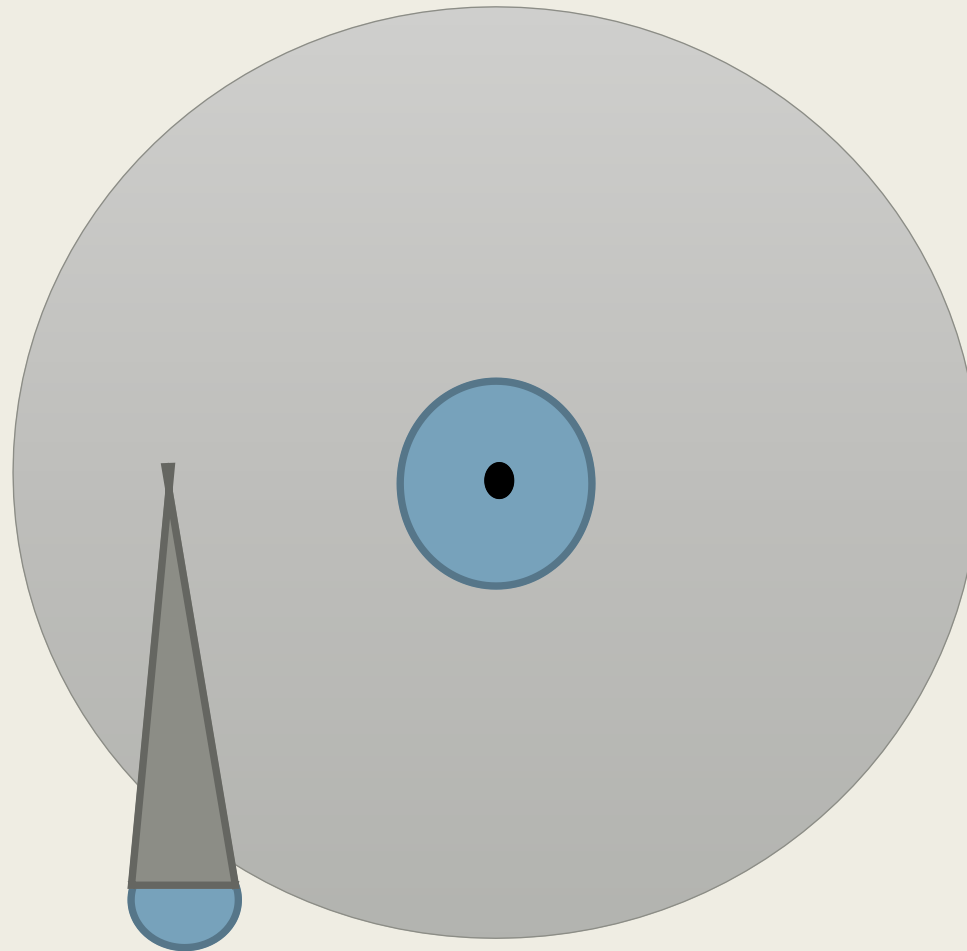
# Remember Anatomy of a disk



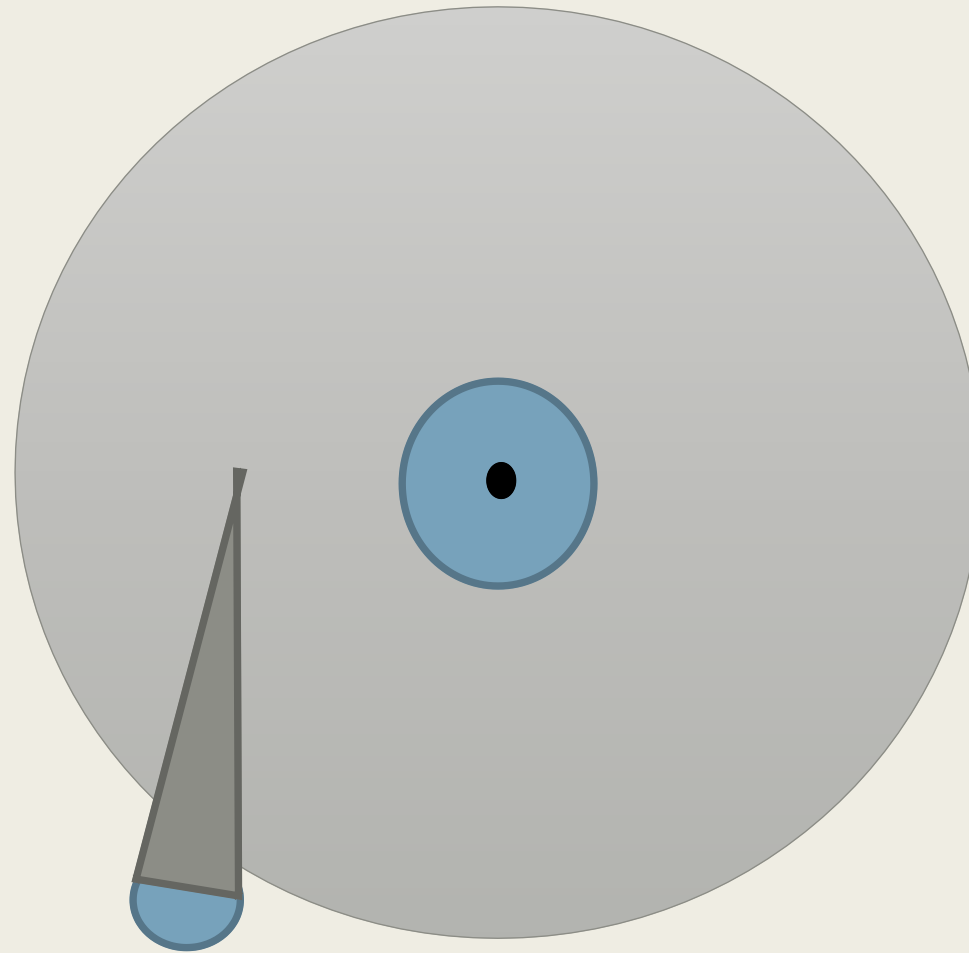
# Remember Anatomy of a disk



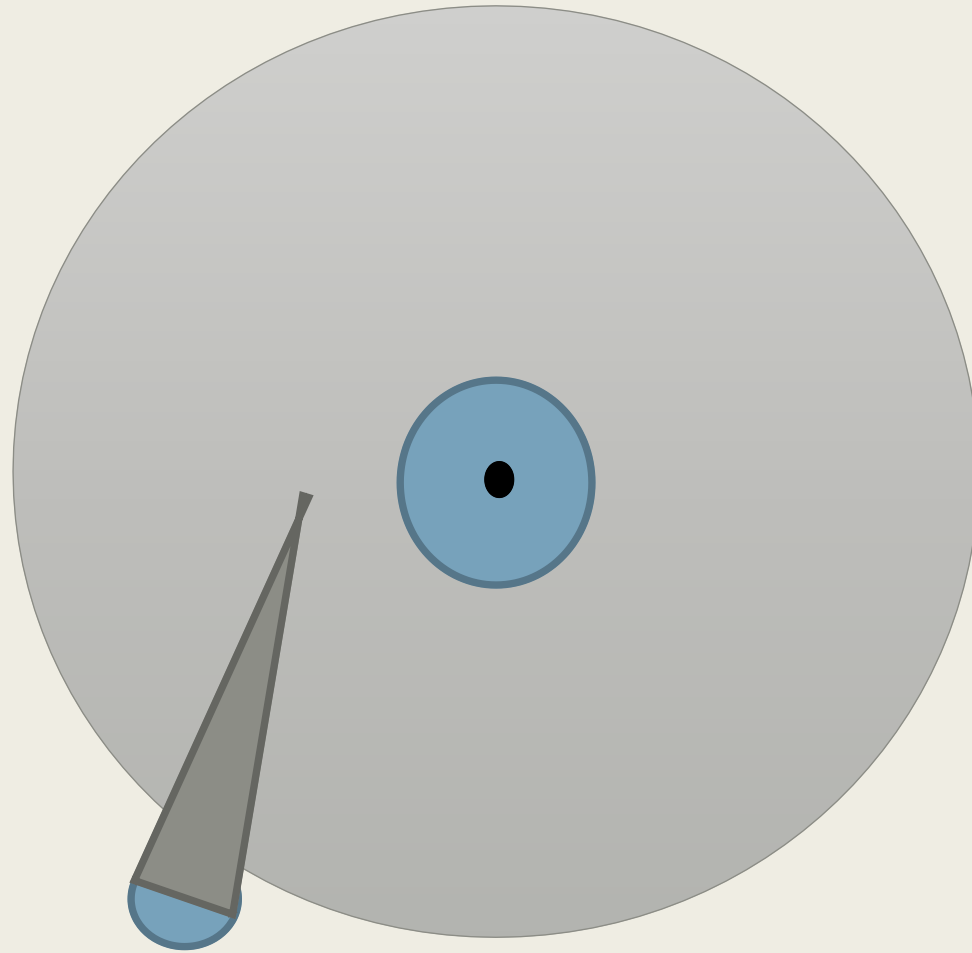
# Remember Anatomy of a disk



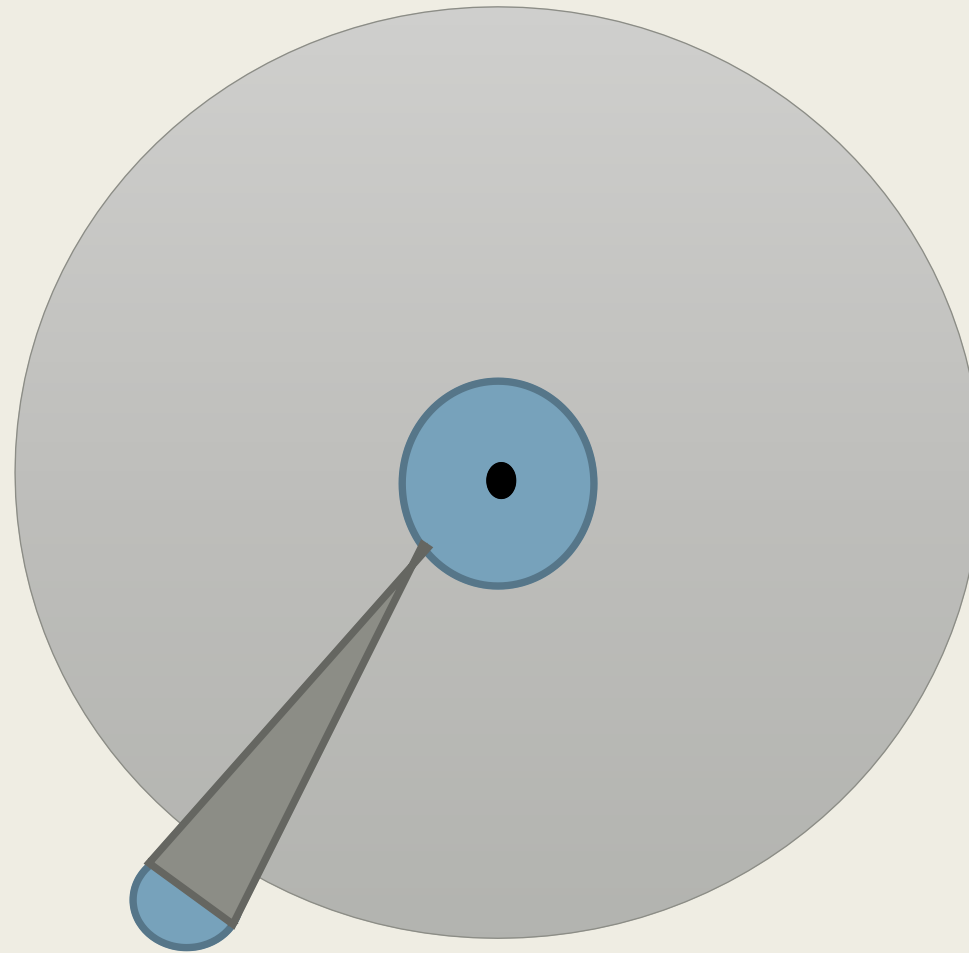
# Remember Anatomy of a disk



# Remember Anatomy of a disk



# Remember Anatomy of a disk



# Example:

- Work queue: 23, 89, 132, 42, 187
- There are 200 cylinders 0-199
- The disk head starts at number 100

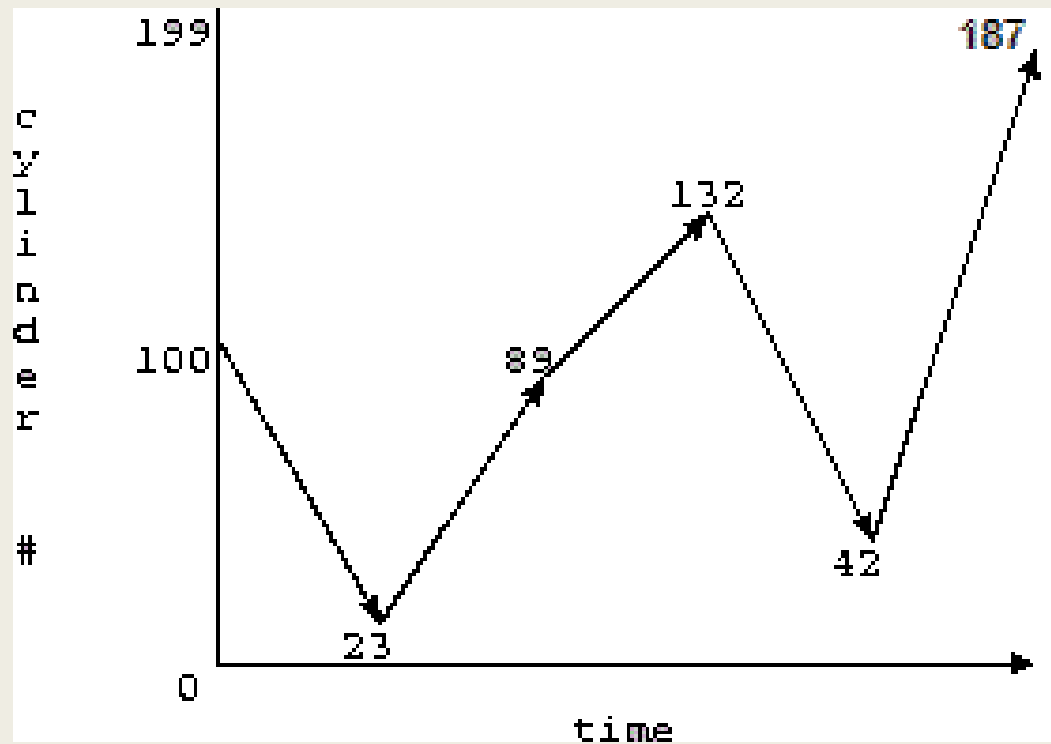


# First-Come-First-Served:

- Work queue: 23, 89, 132, 42, 187

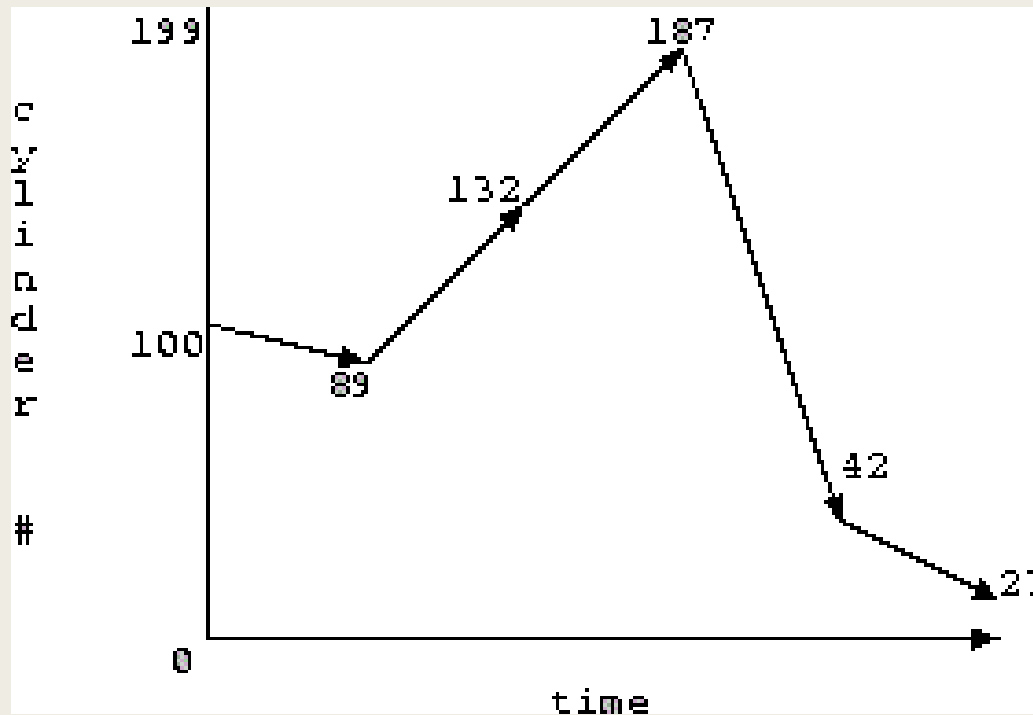
- Total seek length:

$$|23 - 100| + |89 - 23| + |132 - 89| + |132 - 89| + |42 - 132| + |187 - 42| = 421$$



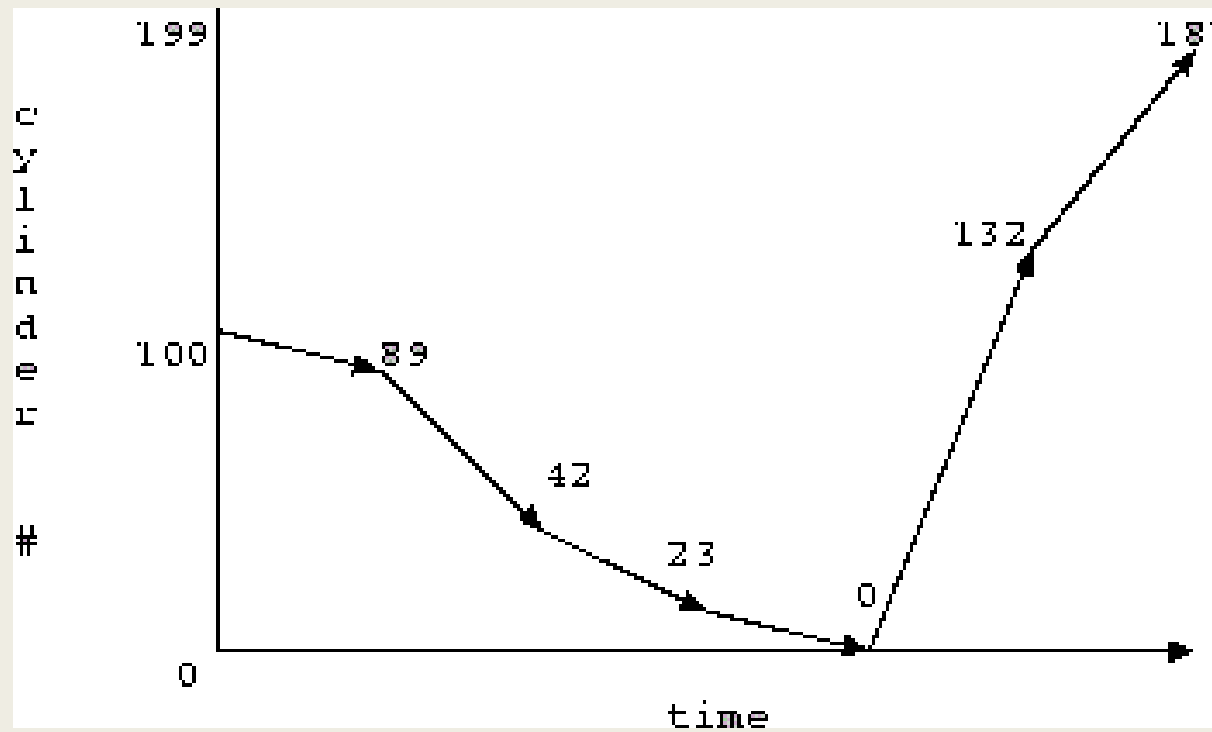
# Shortest-Seek-Time-First:

- Work queue: 23, 89, 132, 42, 187
- Can be inefficient
  - *Multiple changing directions*
  - *Starvation*



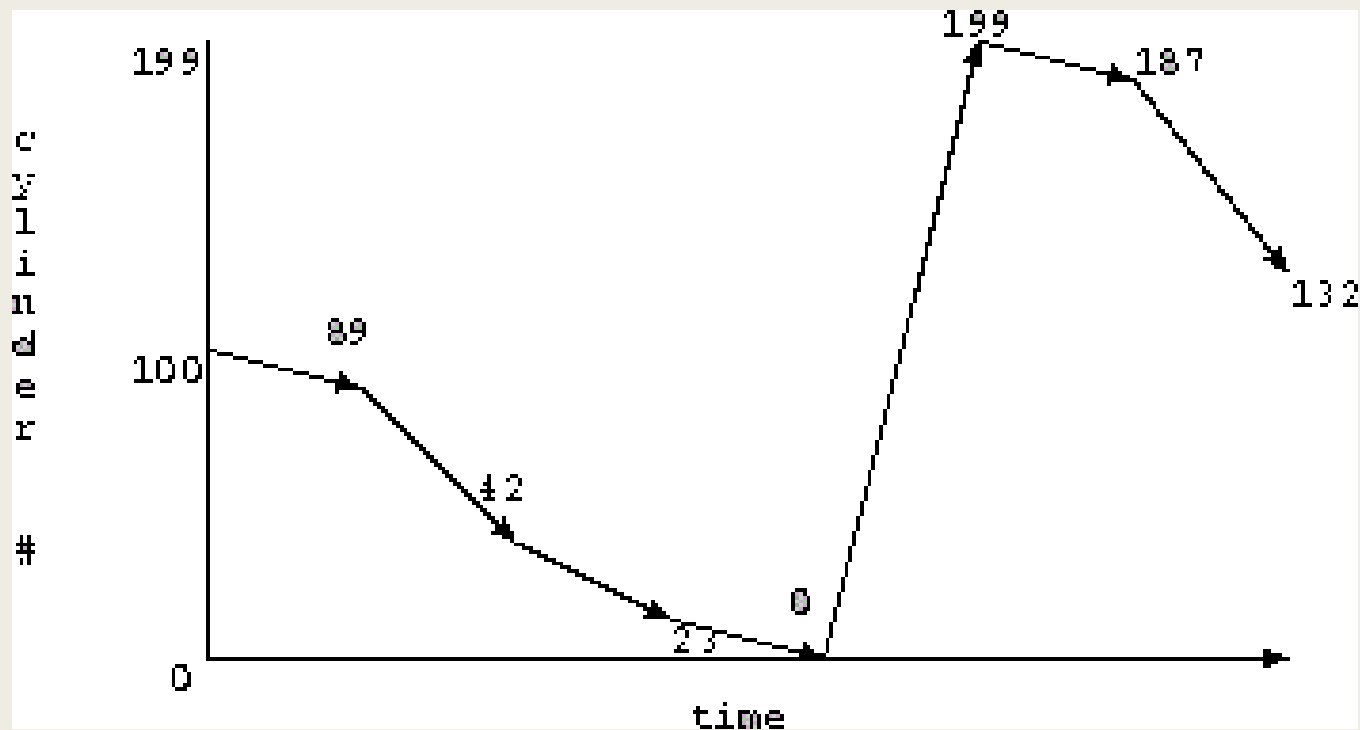
# SCAN:

- Work queue: 23, 89, 132, 42, 187
- Elevator
- Sweeps the disk, to-and-fro
- LOOK is similar



# C-SCAN:

- Work queue: 23, 89, 132, 42, 187
- Elevator
- Sweeps the disk, but one-direction



# Performance:

- Depends on number of requests
- SCAN & C-SCAN are good for systems that place a heavy load on the disk, less likely to cause starvation
- Default: SSTF or LOOK
  - *PRIORITY*

# What we covered:

- Partition Sector Zero
  - *FAT16*
  - *Reading & understanding*
- Volumes & Partitions
- Windows & Unix partitions
- Directory structures & terminology
  - *Root Directory*
  - *Parent Directory & Subdirectory*
  - *Absolute Path Names & Relative Path Names*
- File types
- File operations
- Unix file systems
- Disk scheduling

# Further reading:

- File signatures reference:
  - <https://file signatures.net/>
  - [http://www.garykessler.net/library/file\\_sigs.html](http://www.garykessler.net/library/file_sigs.html)
- Indexing & Disk scheduling:
  - *Operating Systems: Internals & Design Principles, Williams Stalling (7<sup>th</sup> ed.)*
    - PP. 550-552 & 510-512
    - Online, see Reading List

# Tutorial exercise:

- File Types:
  - *How to establish exactly what the file type of a file is*
  - *Even if the extension is wrong*



# Thank you

© The University of Westminster (2021)

These notes were modified from the lecture slides generated by Noam Weingarten.  
The right of Noam Weingarten to be identified as author of this work has been asserted by them in accordance with the Copyright, Designs and Patents Act 1988