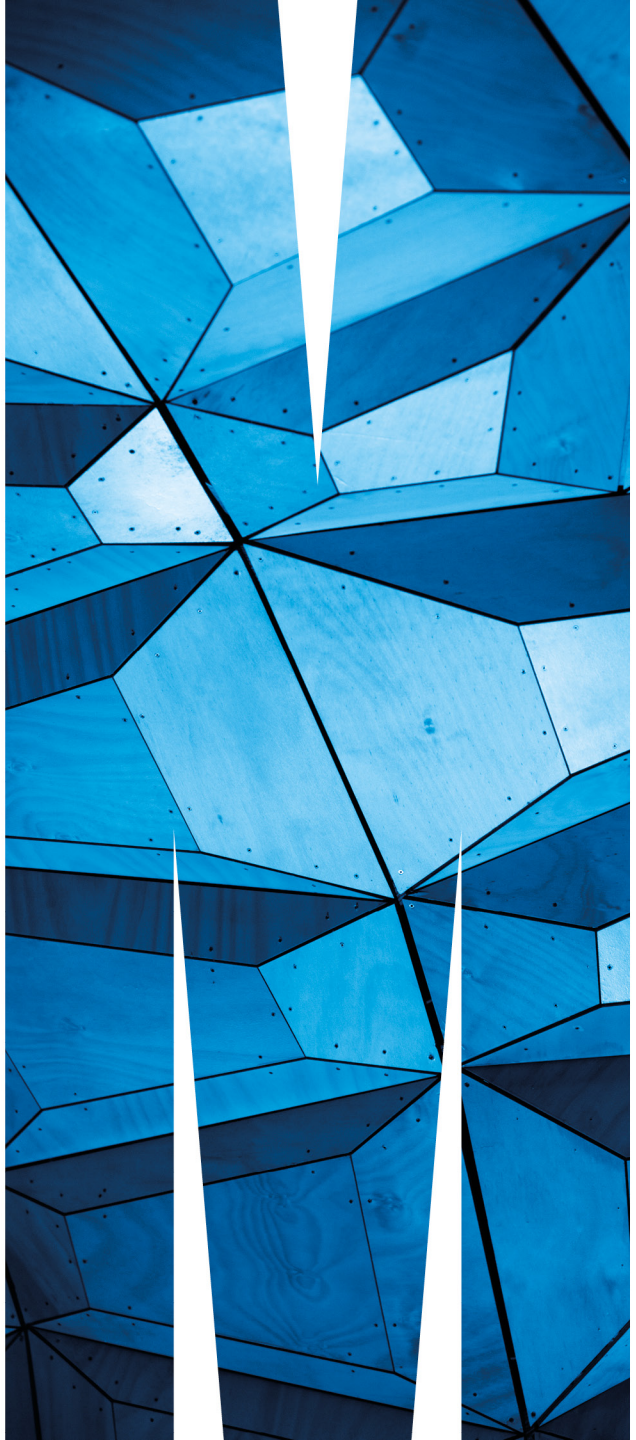
MONASH University

MONASH
INFORMATION
TECHNOLOGY

# FILE MANAGEMENT

LECTURE 4/FIT2100/Semester 2, 2020

WEEK 4



# FILE MANAGEMENT

## INTRODUCTION

### LEARNING OUTCOMES

- Understand the basic concepts of **files** and **file systems**
- Understand how the OS boots from a hard drive
- Discuss approaches used in real-world file systems
- Describe the low-level Unix system calls used to **access** and **manipulate** the filesystem

### READING

- Stallings: Chapter 12
- Extra information about Windows FAT filesystem: https://social.technet.microsoft.com/wiki/contents/articles/6771.the-fat-file-system.aspx



MONASH University

# WHAT IS A FILE?

- A file is a sequential list of characters (bytes)
- Interpretation of data is determined by the application program
- Text file:
  - File is made up of characters
  - Each byte is assigned a character code value
  - When printing the file contents, the terminal displays actual characters
- Binary files
  - Bytes are used to store values directly
  - e.g. 4 bytes can be used to represent a 32 bit integer
  - 1 byte can store a value from 0 to 255.
  - Byte values do not *need* to match readable character codes
  - Trying to view the file as *text* shows nonsense characters



MONASH University

# WHAT IS A FILESYSTEM?

**SOFTWARE WITHIN THE OPERATING SYSTEM**

- Manages data on the disk
  - Translates block data on the disk into abstract concepts of files and directories
- Provides services for accessing and manipulating files

**STRUCTURED DATA STORED ON THE DISK ITSELF**

- Multiple approaches exist
- Different operating systems use different filesystems
- Two most common approaches
  - File Allocation Tables (Windows)
  - Inode-based systems (UNIX).

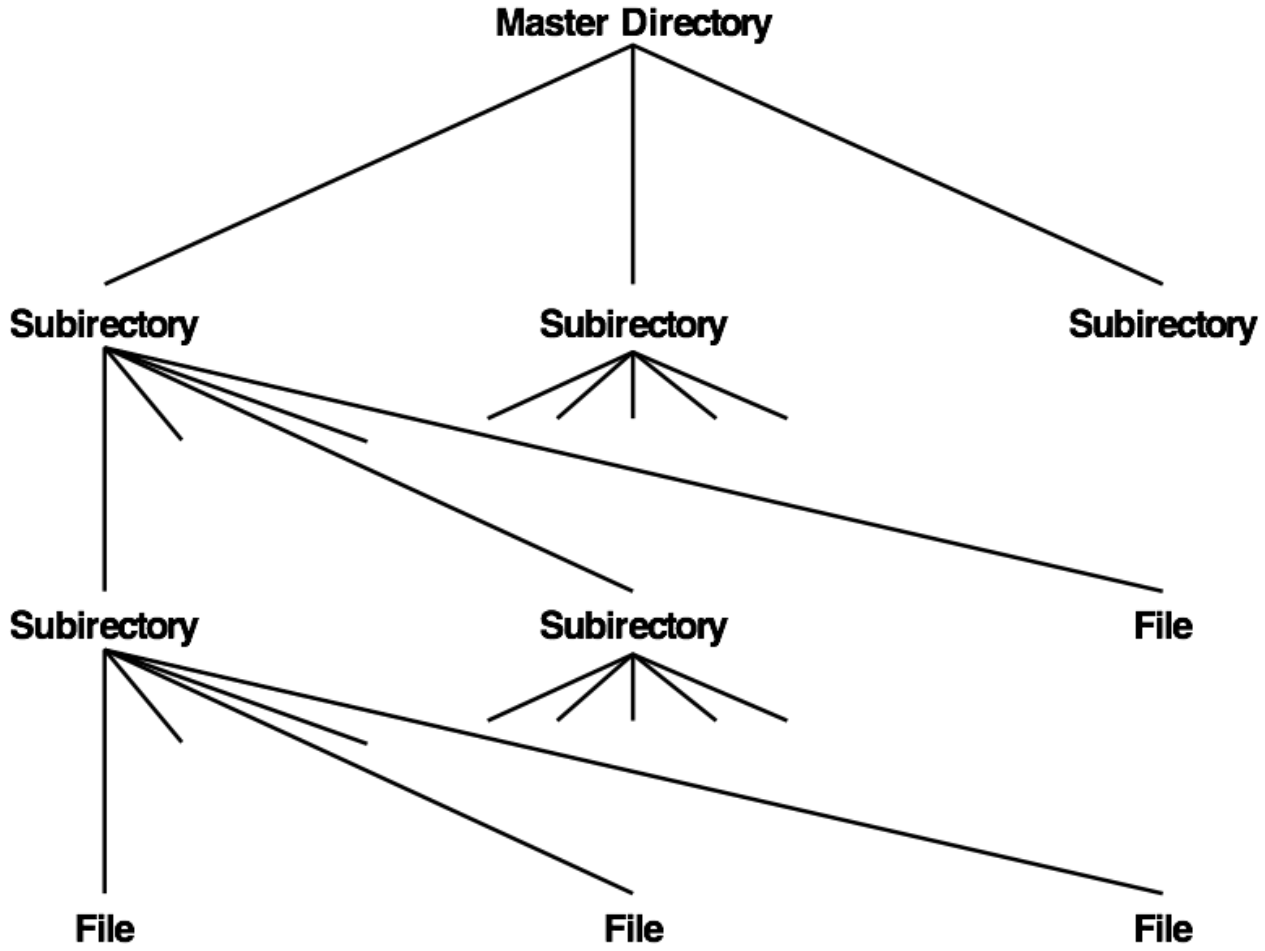


MONASH University

# HIERARCHICAL FILESYSTEMS

AKA ‘TREE-STRUCTURED DIRECTORY’

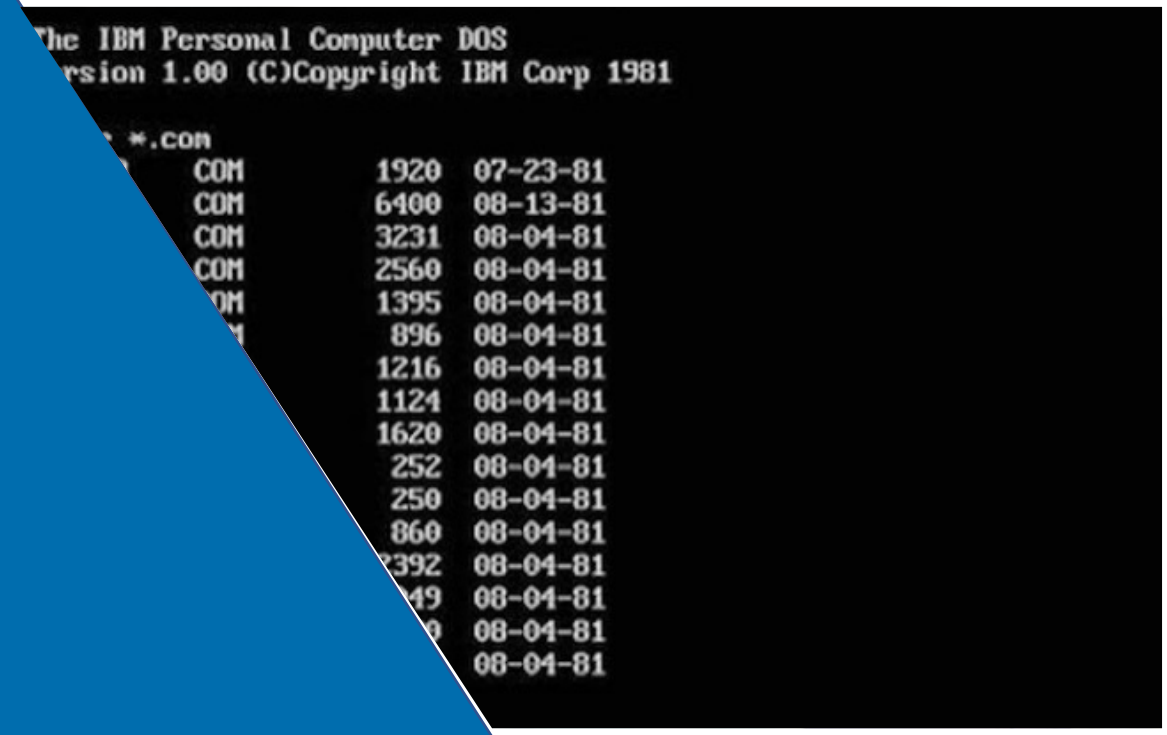
- Multiple file-organisation approaches are possible
- Tree-structured or ‘hierarchical’ is most common
- A ‘root directory’ or ‘master directory’ contains files and sub-directories.






MONASH University

# REAL-WORLD FILE SYSTEMS (FAT)



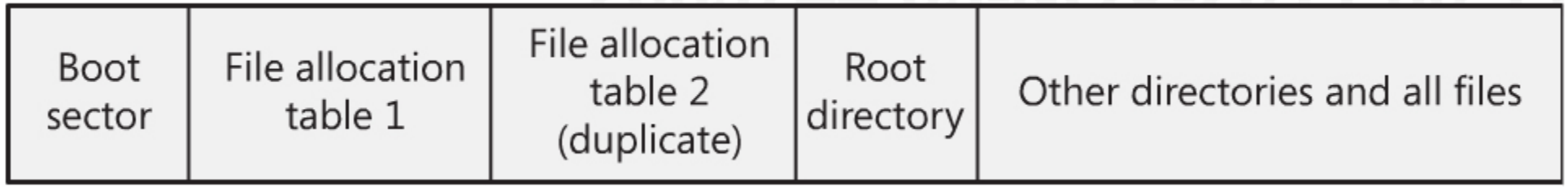

MONASH University

# FAT

## FILE ALLOCATION TABLE

**COMMON ON WINDOWS SYSTEMS (FAT32, ETC.)**

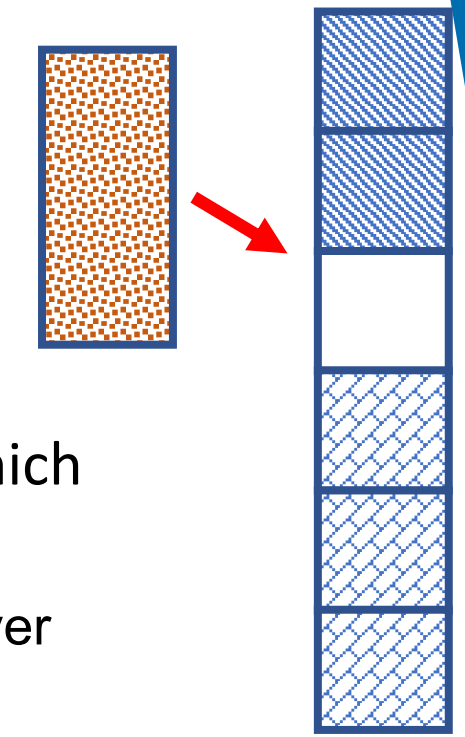
- Reference: https://social.technet.microsoft.com/wiki/contents/articles/6771.the-fat-file-system.aspx
- Drive is split into equal-size **blocks** (FAT32: a 1TB hard drive is split into 32KB 'clusters')
- A file may occupy one or more blocks
- A **table** at the start of the disk contains an entry for every block on the drive.
- In case the table is corrupted, files can no longer be located.
    - Two copies of the table are kept, for redundancy



# FRAGMENTATION

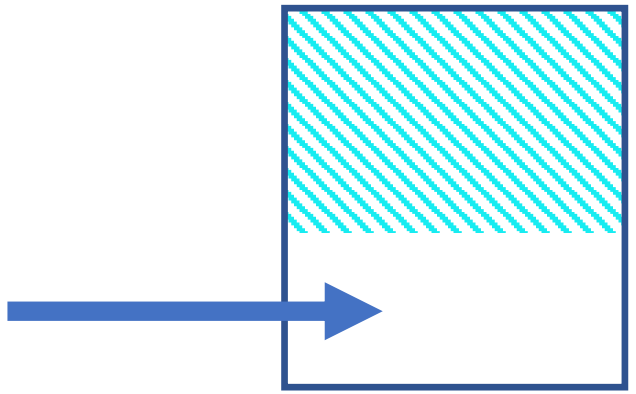
## TYPES OF FRAGMENTATION

### EXTERNAL FRAGMENTATION



- External fragmentation
- Free blocks between files, but the spaces are not large enough to fit certain files
    - e.g. a small file was deleted, leaving a space which is too small to fit a new larger file.
- **Fragments** of wasted space unless the file can be split over multiple fragments.
- Was a problem in very old filesystems.

### INTERNAL FRAGMENTATION



- If a file is smaller than a single block
- Unused space inside the block
    - Cannot be allocated to other files

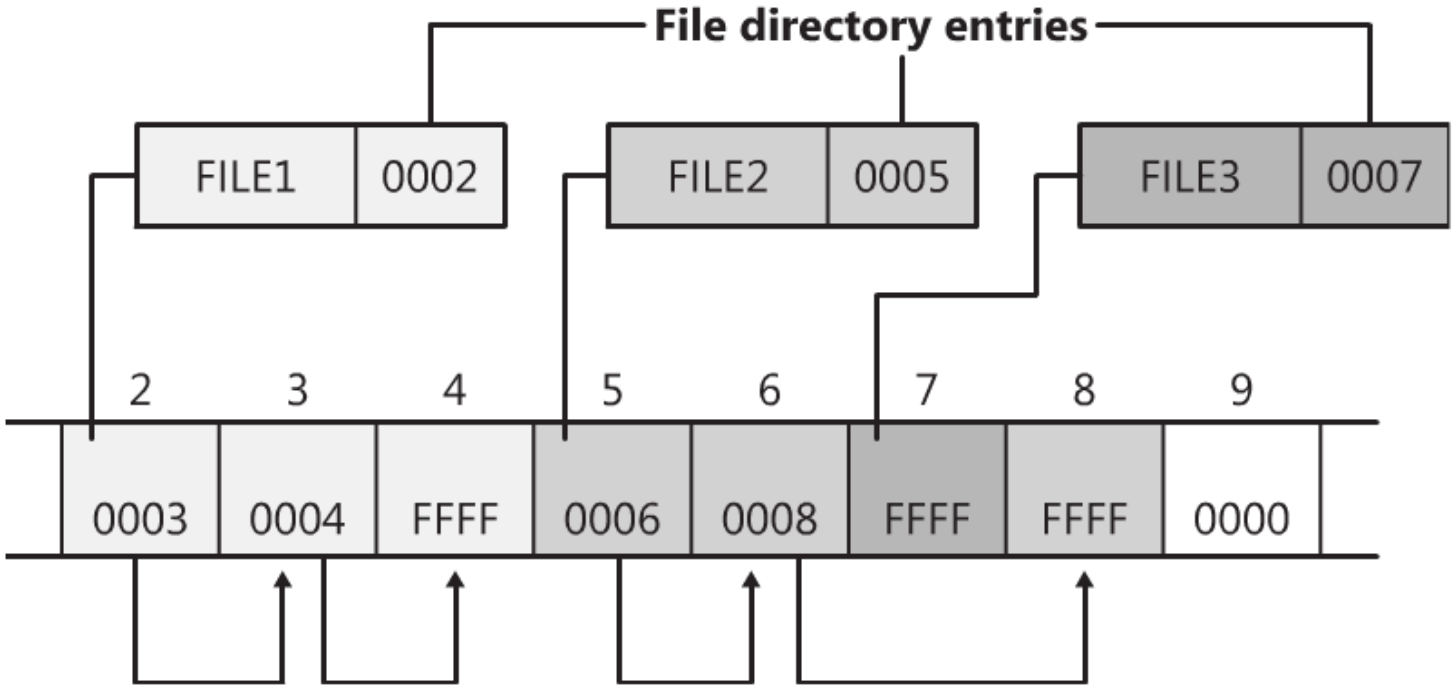


MONASH University

# HOW A DIRECTORY WORKS (FAT)

DIRECTORY = INDEX OF FILE LOCATIONS ON DISK

- A directory stores the address of the **starting block** of each file.
- Files are allocated to the first empty block.
- If the file is too large for one cluster, available clusters are chained together
  - Each block stores a reference to the next one where the file continues
  - By following the chain, the entire file can be read



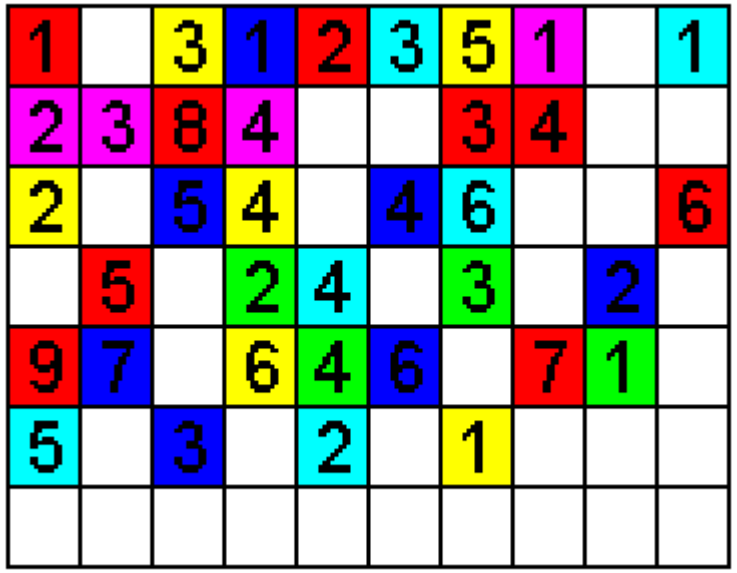



MONASH University

# FILE SYSTEM AGEING

FAT

**Defragmenting disk**
Creative commons attribution:
XZise, Wikimedia Commons




**NO EXTERNAL FRAGMENTATION**

- Chaining blocks together means that no blocks are wasted
  - File does not need to be contiguous on the disk
- Files are added and deleted, creating gaps of free sectors
- A large file may not fit into a single gap
  - File is spread over different tracks
- The hard drive head must move to different locations to access the entire file
  - Access times get slower over time
- Disk requires periodic '**defragmentation'** to improve performance.
  - Files are rearranged into contiguous blocks



MONASH University

# REAL-WORLD FILE SYSTEMS (UNIX)

MONASH University

# TYPES OF FILES IN UNIX

**Regular, or ordinary**

- contains arbitrary data in zero or more data blocks

**Directory**

- contains a list of file names plus pointers to associated inodes

**Special**

- contains no data but provides a mechanism to map physical devices to file names

**Named pipes**

- an interprocess communications facility

**Links**

- an alternative file name for an existing file

**Symbolic links**

- a data file that contains the name of the file it is linked to

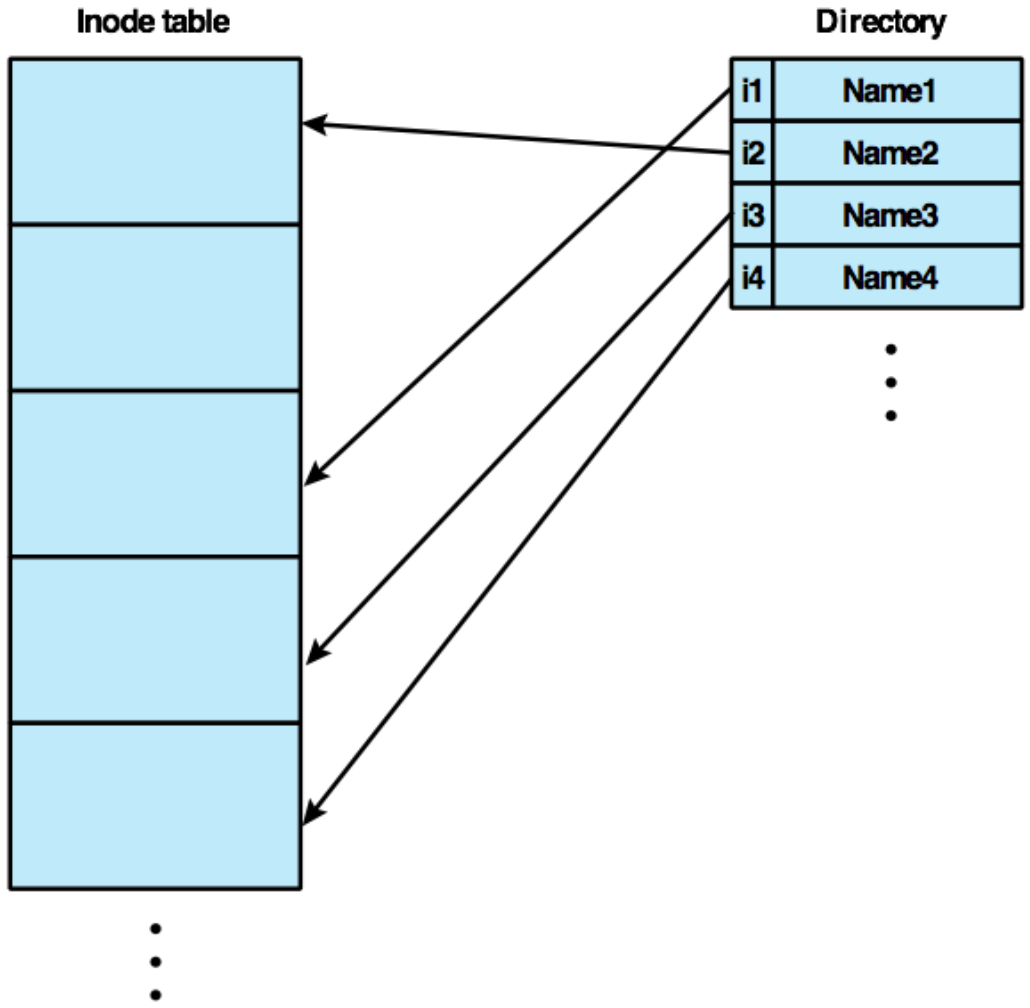
# UNIX FILESYSTEMS: INODES

- All types of UNIX files are stored in a table of **inodes**
  - ‘Index nodes’
  - An alternative system to file allocation table (FAT) systems
- The inode for a file stores key information about that file
- Each file is controlled by exactly one inode
- Several filenames may be associated with a single inode
  - The same physical file can be mapped to multiple directories.

# HOW DO DIRECTORIES WORK?

## INODE-BASED FILESYSTEMS

- A directory is stored like a regular file
- Directory contains a table of inode numbers and filenames
- Files are located by looking up the appropriate inode in the inode table.



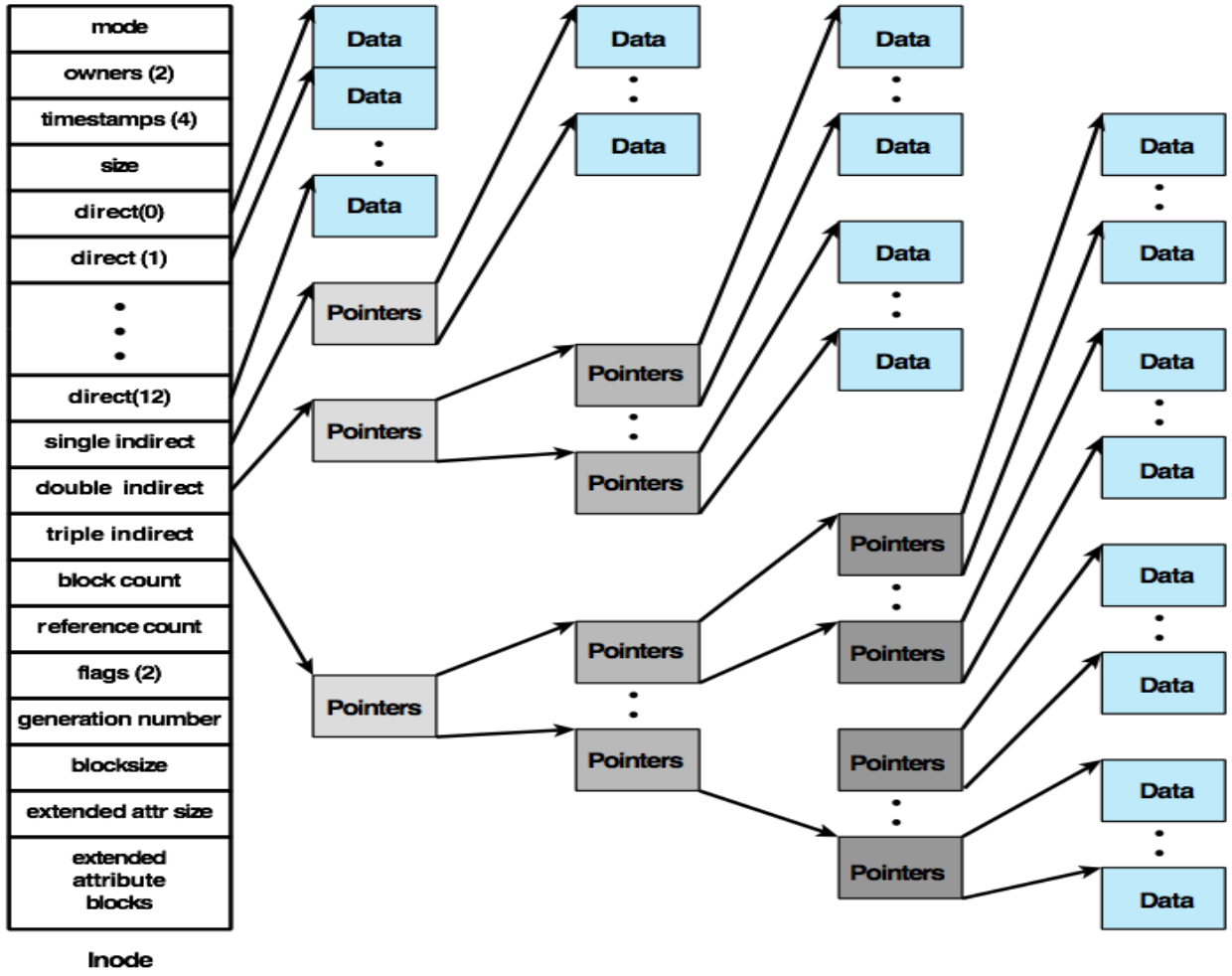



MONASH University

# UNIX: FILE SYSTEM STRUCTURE

## INODE AND FILE

- Each inode contains various information about a file, and how to locate it on disk.
- When a file is opened, its corresponding inode is loaded into main memory.
- If a file requires 12 blocks on disk or fewer, these addresses are listed **directly** in the inode itself.
- If the file is larger than 12 blocks, the filesystem allocates special **pointer blocks** containing the remaining addresses
- If the file is very large, multiple levels of pointer blocks are used






MONASH University

# BOOTING THE OPERATING SYSTEM

MONASH University

# BOOTING THE O.S.

AKA `BOOTSTRAPPING'

**A PROBLEM**

- Before the computer is turned on, the OS lives only on the hard drive
- But, without the OS running, how do you locate files on the hard drive?

**BOOTSTRAPPING**

- The idea of using a small, primitive system to start up a slightly more advanced system.
- Many approaches exist, we will look at a common one on PCs.



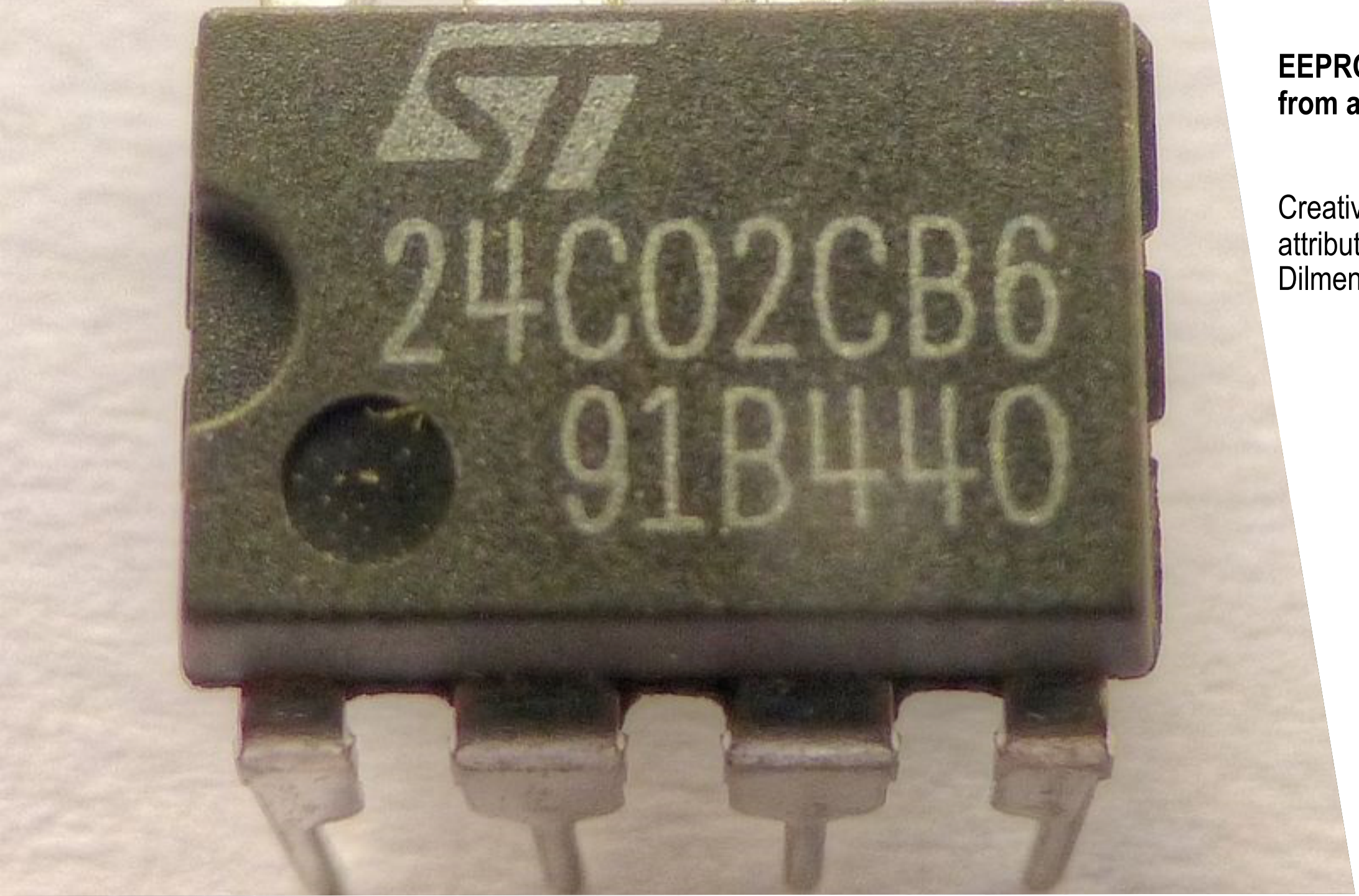
MONASH University

# BIOS

BASIC INPUT OUTPUT SYSTEM

**PRIMITIVE SOFTWARE ON A HARDWARE CHIP**

- EEPROM
  - **E**lectrically **E**raseable **R**ead **O**nly **M**emory
- A tiny area at the very end of main memory that is **non-volatile**
- Machine code is already here when the power is turned on
  - CPU always looks here for the first code to execute
- Contains instructions for recognising connected hard drives
- Can remember hard drive configuration using a small battery inside the computer
- Does not need to know how to read a filesystem, only how to load raw bytes from the hard drive into memory



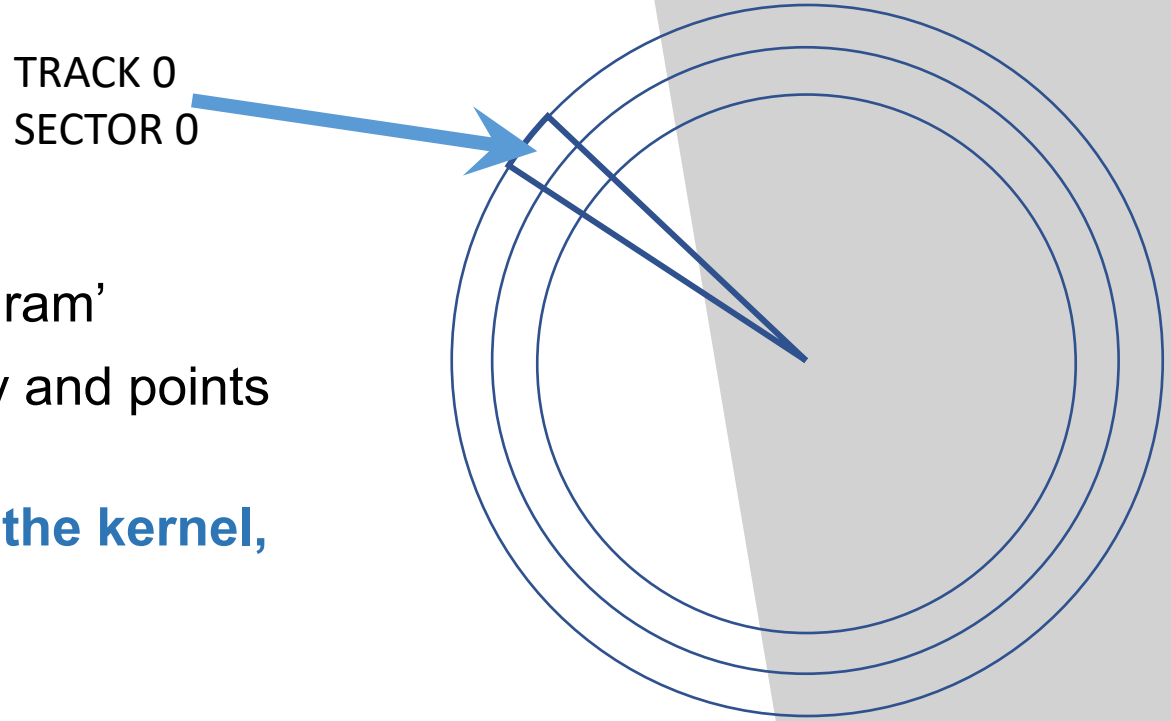

**EEPROM module from a typical PC**

Creative Commons attribution: Nevit Dilmen

MONASH University

# MASTER BOOT RECORD

MBR – ALSO KNOWN AS BOOT SECTOR




**TRACK 0, SECTOR 0**

- The first 512 bytes on a bootable hard drive
- Contains machine code, known as ‘bootstrap program’
- The computer’s BIOS loads this into main memory and points the CPU to it – transfers control.
- **Contains instructions for locating and loading the kernel, which loads the rest.**

**EXISTS INDEPENDENTLY OF THE FILESYSTEM**

- The MBR is not a ‘file’
  - Just raw bytes read directly from start of the hard drive
- If this OS hasn’t been loaded yet, there are no services available for locating files on the rest of the drive.



MONASH University

# ACCESSING THE FILESYSTEM IN LINUX

MONASH University

# SYSTEM CALLS FOR FILE I/O

## OS PROVIDES ACCESS TO THE FILESYSTEM VIA SYSTEM CALLS

- **open(…)**
  - The operating system allocates a number (file descriptor) to each open file within a running process
  - The OS also stores the current read/write position within the file
- **write(…)**
  - The program tells the OS the starting address in main memory and how many bytes to write into the file
- **read(…)**
  - The program gives the OS a main memory address to read file contents into, and how many bytes may be read into memory (i.e. array size).
  - The operating system increments the file position after each read/write.



MONASH University

# SYSTEM CALLS (2)

- **lseek(…)**
  - The OS changes the current read/write position to a different place within the file
- **close(…)**
  - The OS de-allocates the file descriptor information from memory.

- Many other system calls include: **creat()**, **link()**, **unlink()**, **chmod(), chdir(), lchown(), stat(), mkdir(),** etc.



MONASH University

# SUMMARY (LECTURE 4)

## FILE MANAGEMENT

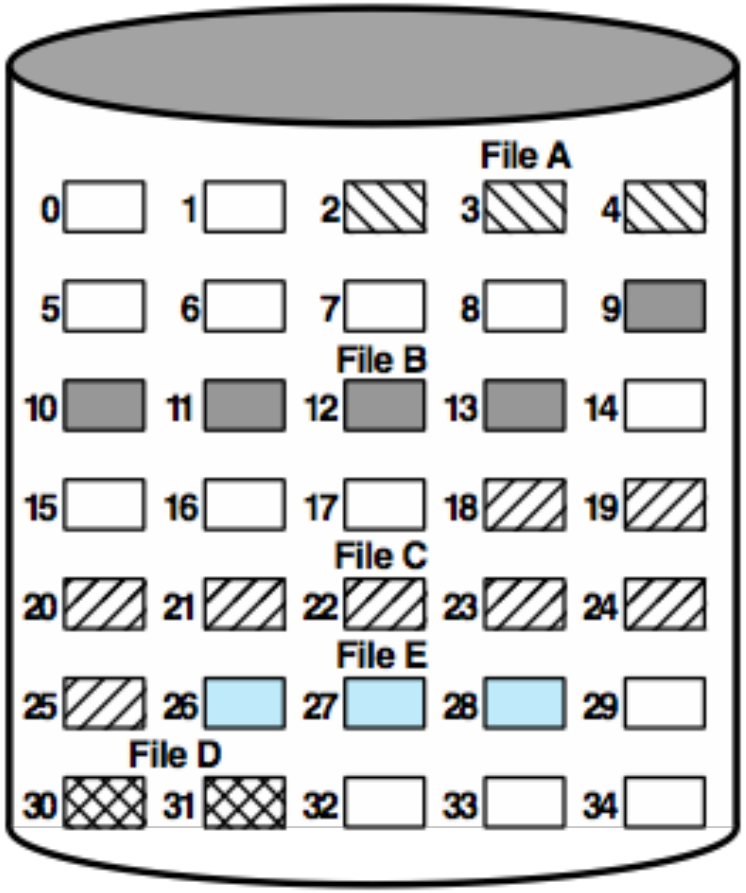
- A file management system is a set of system software that provides services to users, and structured data on the drive itself
- Various approaches to file management exist
  - File allocation table systems
  - Inode-based systems
- The computer must be able to boot the operating system from the hard drive even before the filesystem software has been loaded.
- **Next week: processes**

# SUPPLEMENTARY SLIDES

MONASH University

# File Allocation: Contiguous

External fragmentation

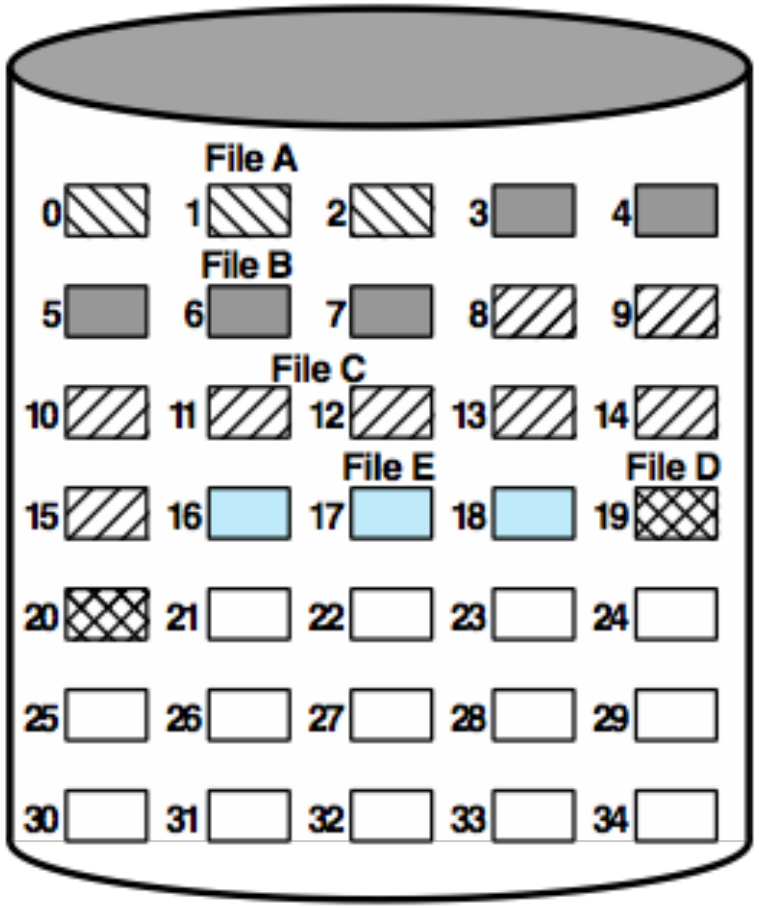



File Allocation Table

| File Name | Start Block | Length |
|---|---|---|
| File A | 2 | 3 |
| File B | 9 | 5 |
| File C | 18 | 8 |
| File D | 30 | 2 |
| File E | 26 | 3 |

- A single contiguous block is allocated to a file at the time of file creation
- FAT needs just one single entry for each file, showing the starting block and the length of the file

# File Allocation: Contiguous



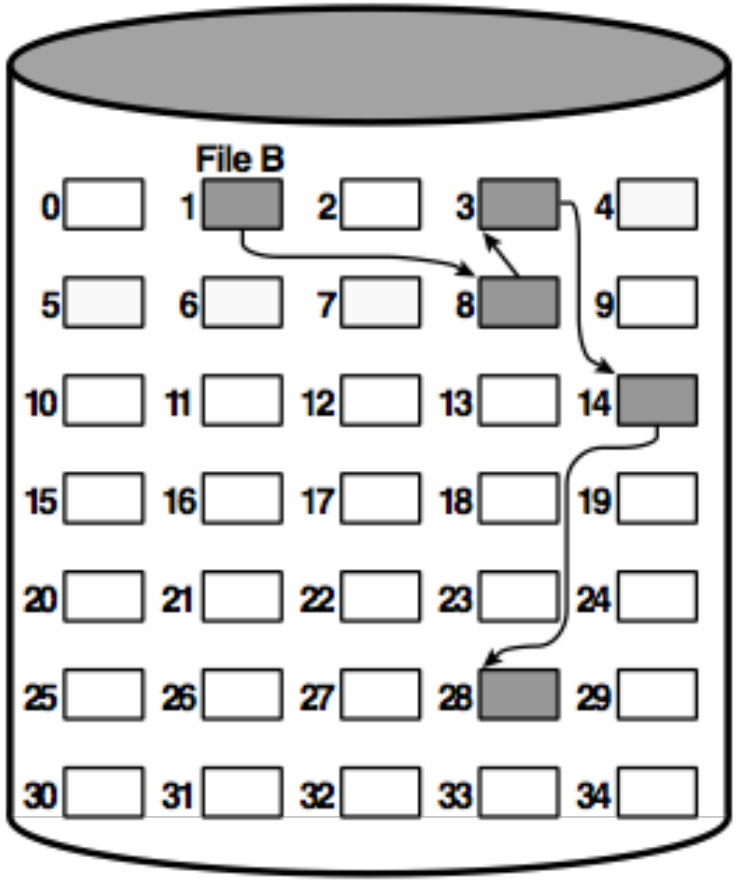

File Allocation Table

| File Name | Start Block | Length |
|---|---|---|
| File A | 0 | 3 |
| File B | 3 | 5 |
| File C | 8 | 8 |
| File D | 19 | 2 |
| File E | 16 | 3 |

**(after compaction)**

# File Allocation: Chained

No external fragmentation



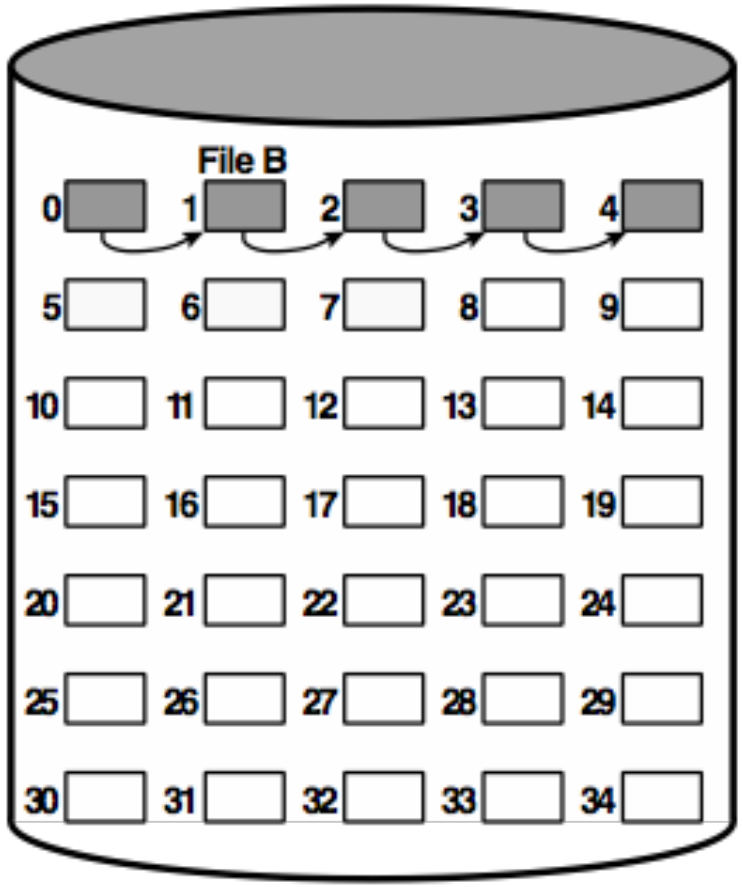

File Allocation Table

| File Name | Start Block | Length |
|---|---|---|
| ... | ... | ... |
| File B | 1 | 5 |
| ... | ... | ... |

- Allocation is on an individual block basis
- Each block contains a pointer to the next block in the chain
- FAT needs just a single entry for each file

# File Allocation: Chained

No accomodation for the principle of locality



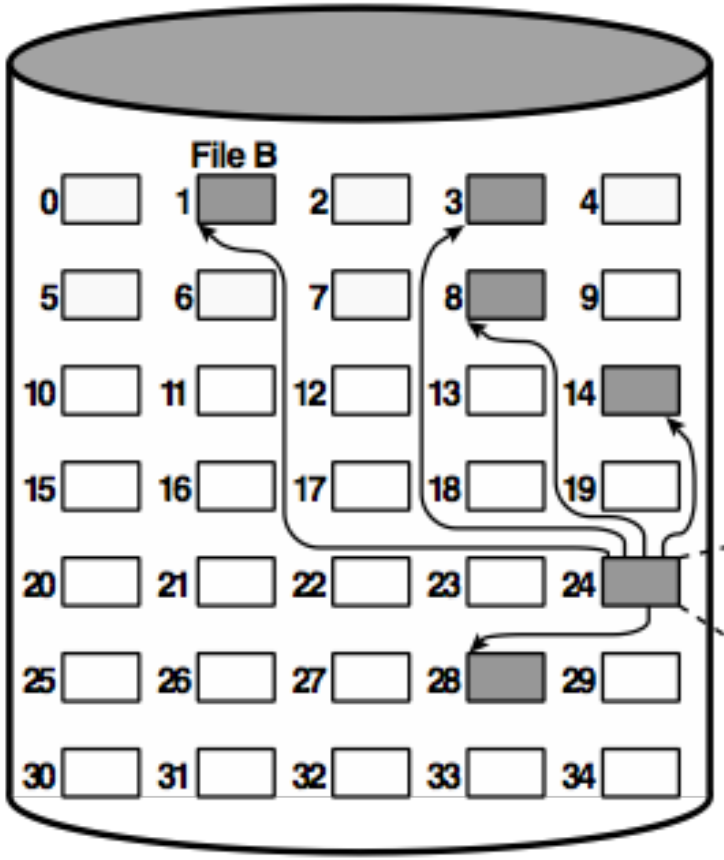

File Allocation Table

| File Name | Start Block | Length |
|---|---|---|
| ... | ... | ... |
| File B | 0 | 5 |
| ... | ... | ... |

(after consolidation)

# File Allocation: Indexed with Block Portions

Eliminates external fragmentation



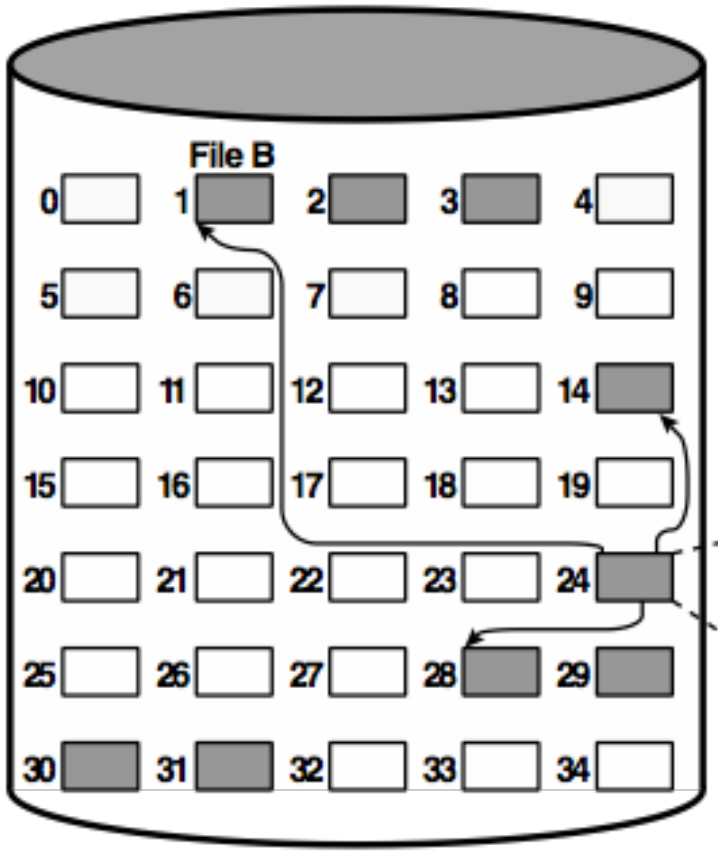

File Allocation Table

| File Name | Index Block |
|---|---|
| • • • | • • • |
| File B | 24 |
| • • • | • • • |

1
8
3
14
28

- FAT contains a separate 1-level index for each file
- The index has one entry for each portion allocated to the file

# File Allocation: Indexed with Variable-Length Portions




File Allocation Table

| File Name | Index Block |
|---|---|
| • • • | • • • |
| File B | 24 |
| • • • | • • • |

| Start Block | Length |
|---|---|
| 1 | 3 |
| 28 | 4 |
| 14 | 1 |

Improves locality

Indexed allocation supports both sequential and direct access to the file