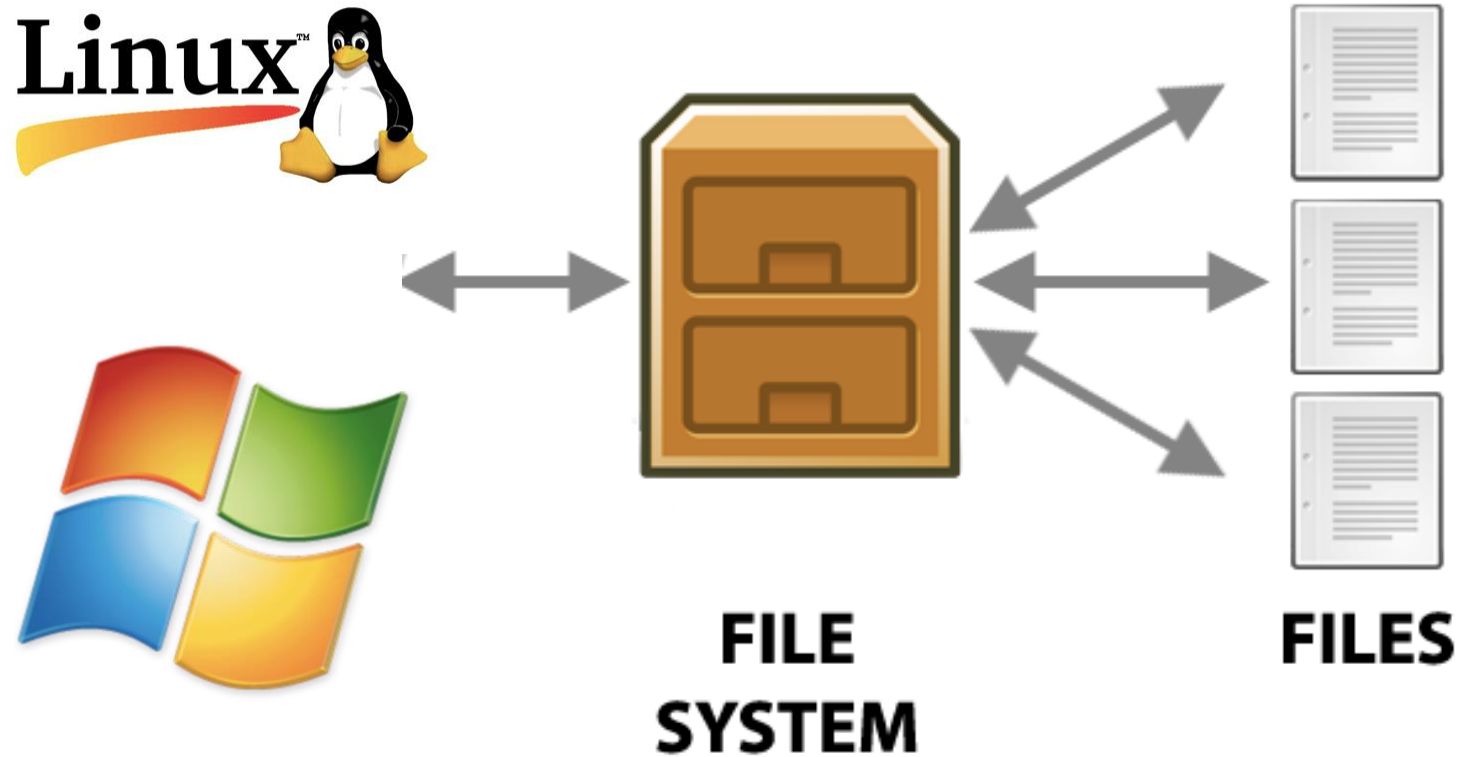
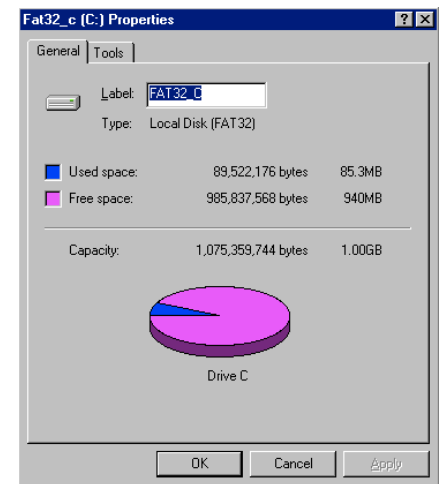
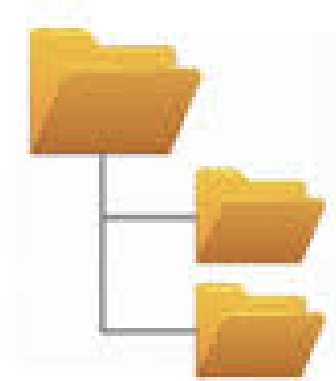


# File Systems



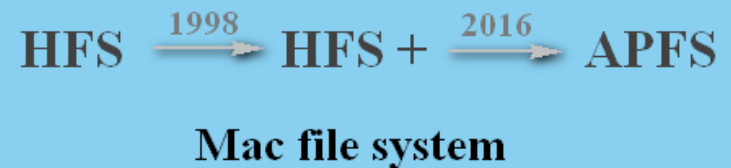
# File Management System

- The File System is the most visible aspect of an Operating System.
- It provides the mechanism for :
  - Storage
  - Access to both Data and Programs of the Operating system



# Typical Similarities Among File Systems

- A top-level directory
- Support for hierarchical directories
- Metadata about files (e.g. creation data)
- Management of Free and Used Space
- Protection and Security



# Files and the File System

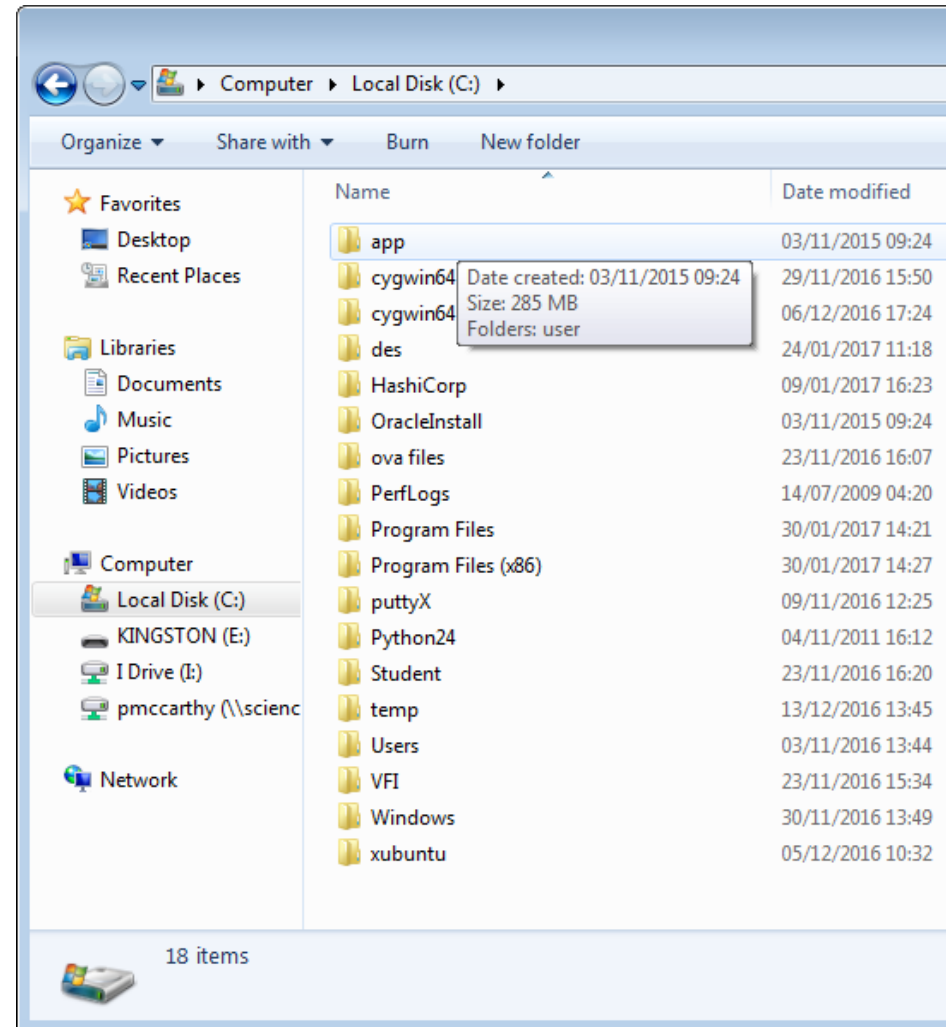
- A **file** is a
  - **named collection of data**
  - stored on a secondary storage device,  
e.g. hard disks, flash memory (and SSD), and DVD.
- The file system provides a series of
  - low-level functions to manipulate files on a disk  
**e.g. open, close, read, write, etc.**
- The file system is part of the
  - kernel and the operating system
- Different types of files exist
  - Text file, source file, executable file.

# File System Objectives

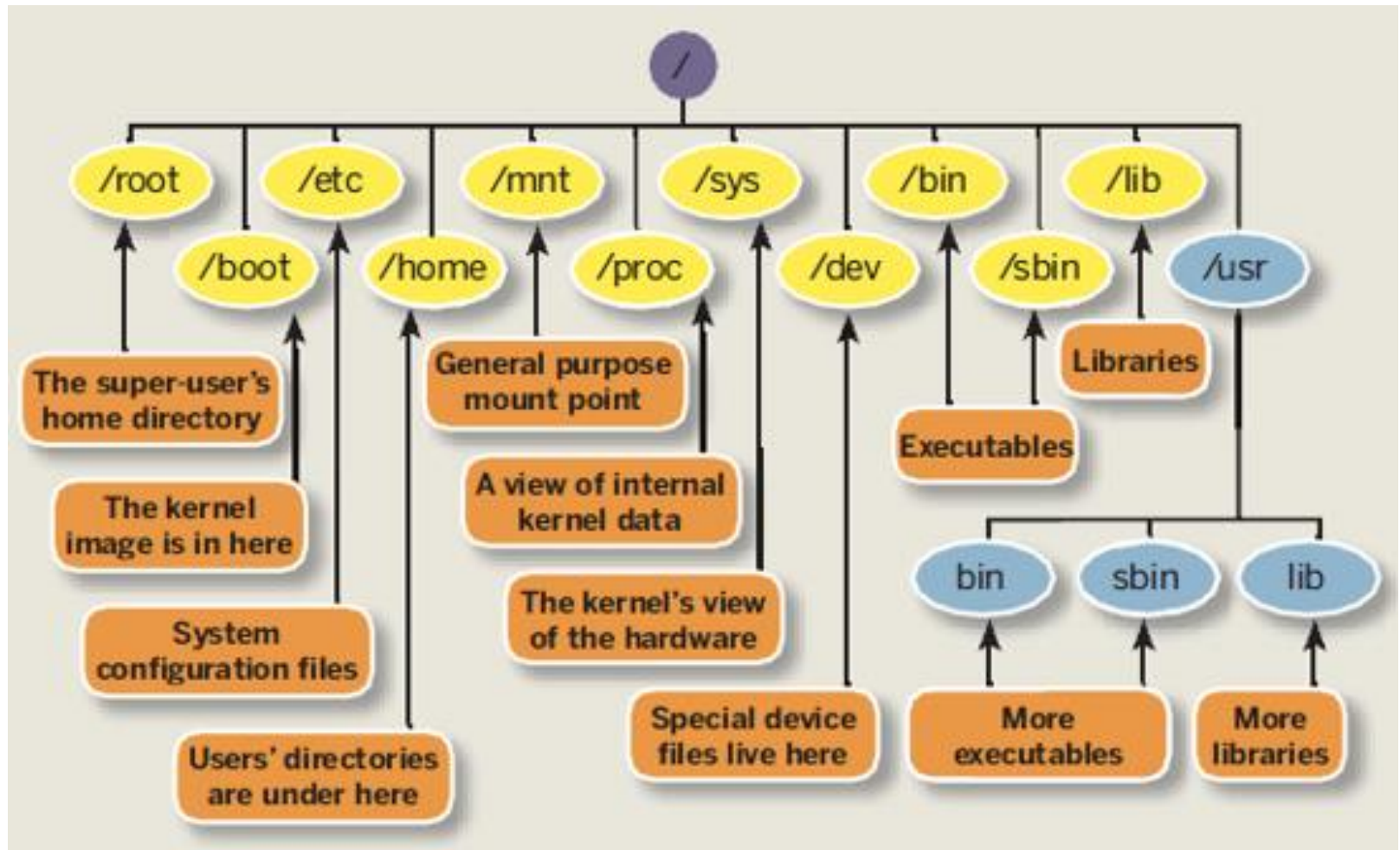
1. Creation and deletion of files
2. Organise data in a way that the user can access it quickly and easily
3. Provide a logical view for the user and hide the physical implementation
  - Abstract the user from the implementation
  - Allows files to be referred to by a user defined name, rather than a machine address
    - Hard disk, track, sector, etc
  - Actual location of files on the disk transparent to the user
  - Manage directory structures and space allocation for each I/O device
4. Permit sharing of data and programs together with file security and protection of file integrity

# Files and the File System

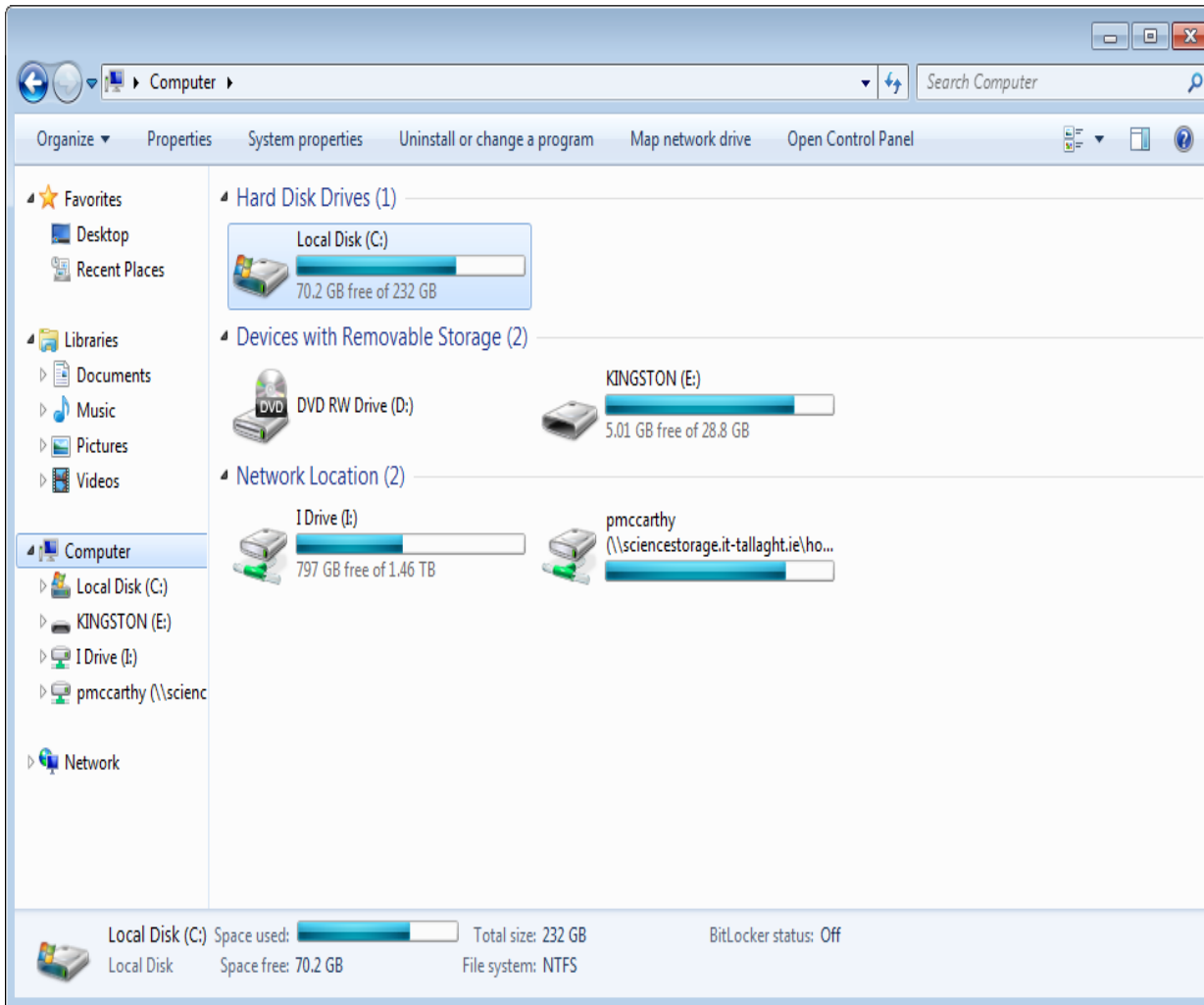
- Each file is stored in a directory, which is a hierarchical structure, with the root directory at the top.
- In a computer's filing system on the hard disk, the root directory is the directory (or 'folder') from which all other directories will be created.



# Typical Linux File System Hierarchy



# Typical Windows File System Hierarchy



## Typical directories

- C:\Users\moconnor
- C:\Program Files
- C:\Windows
- C:\temp



# Partitions?

**Disk partitioning** is the act of dividing a drive into multiple logical storage units referred to as *partitions*, to treat one physical disk drive as if it were multiple disks.

## Some advantages of Partitioning

- Separation of the [operating system](#) (OS) and program files from user files. This allows [image backups](#) (or [clones](#)) to be made of only the operating system and installed software.
- Keeping frequently used programs and data near each other.
- Having cache and log files separate from other files. These can change size dynamically and rapidly, potentially making a file system full. (how to solve?)
- Use of [multi-boot](#) setups, which allow users to have more than one operating system on a single computer. E.g. [Linux](#), and [Microsoft Windows](#). Why not just use VM? Also... What's the advantage of VM?
- Protecting or isolating files, to make it easier to recover a corrupted file system or operating system installation. If one partition is corrupted, other file systems may not be affected.
- Raising overall computer performance on systems where smaller file systems are more efficient. For instance, large hard drives with only one [NTFS](#) file system typically have a very large [sequentially accessed Master File Table](#) (MFT) and it generally takes more time to read this MFT than the smaller MFTs of smaller partitions.

## NOTE re Web address root directory

- The root-level folder of your website is the folder where your index page (homepage) is stored on the server. Visitors can access your root directory and any files or directories inside it (provided you link to them or the visitor knows the specific URL). So best keep as separate partition.

# Files and the File System, continued

- Disk Management in Microsoft Windows
  - Disk partitions identified by letters of the alphabet
  - Root directories: C:\, D:\, E:\, etc.
- Unix/ Linux does not use letters to identify partitions
  - All partitions viewed under root directory '/'

Volume	Layout	Type	File System	Status	Capacity	Free Space	% Free	Fault Toleran
DATA (D:)	Partition	Basic	FAT32	Healthy	19.52 GB	9.91 GB	50 %	No
New Volume (E:)	Partition	Basic	NTFS	Healthy	16.81 GB	4.76 GB	28 %	No
WINDOWS (C:)	Partition	Basic	NTFS	Healthy (System)	19.53 GB	7.36 GB	37 %	No
ZYNET (H:)	Partition	Basic	FAT32	Healthy	27.92 GB	15.50 GB	55 %	No

The screenshot shows the Windows XP Disk Management console. It displays two disks: Disk 0 (55.88 GB) and Disk 1 (27.94 GB). Disk 0 contains three partitions: WINDOWS (C:) (19.53 GB NTFS, Healthy System), DATA (D:) (19.53 GB FAT32, Healthy), and New Volume (E:) (16.81 GB NTFS, Healthy). Disk 1 contains one partition: ZYNET (H:) (27.94 GB FAT32, Healthy). Below the disk list, two CD-ROM drives are shown, both with 'No Media'. A legend at the bottom indicates that blue represents a Primary partition, green represents an Extended partition, and yellow represents a Logical drive.

Windows XP Disk Manager

# Files and the File System, continued

- Files are stored in *directories*
  - A *directory* itself can consist of zero or more files or directories
- By default, each directory contains two files: one indicating the current directory (.) and the other represents the parent directory (..)

```

C:\system>dir
Volume in drive C is WINDOWS
Volume Serial Number is E8D8-186A

Directory of C:\system

08/02/2006  22:25    <DIR>          .
08/02/2006  22:25    <DIR>          ..
               0 File(s)              0 bytes
               2 Dir(s)      7,804,776,448 bytes free

C:\system>_
  
```

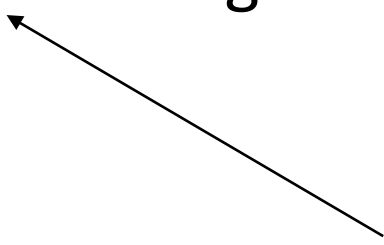
What will the following commands do?

- cd .
- cd ..
- cd \

# File Operations

- **Consider a File as an entire entity we want to**
  - Copy, Move
  - List, Print
  - Load and execute a program
  - Load file into memory
  - Store file from memory
  - Append data from memory to file
  - Compile, assemble a file

# File Operations

- **Within a file, that results in**
    - **Open** a file
    - **Read** a number of bytes from file
    - **Write** a number of bytes to a file
    - **Move** the file **pointer** forward or backward
    - Move file **pointer** to beginning of a file
    - **Close** a file
- 
- Current Position in file

The moving of the file pointer is called 'seek'

# Directory Operations

---

- Directories are formed as part of the tree structure
- **Examples of Directory operations are**
  - **Create** a new (empty) directory
  - **Move** a directory about the tree hierarchy
  - **Move a file** from one directory to another
  - **Rename** a directory
  - **Remove** a directory

# File Access Methods

---

How do we read in the data  
stored in our files?

# File Access Methods

---

- Files need to be accessed
  - They are located on the hard disk and must be moved between the hard disk and the RAM
- Different approaches are used to move the data between the devices
  - **Sequential Access**
  - **Random (Direct) Access**
  - **Indexed Access**



# File Access Methods

## Sequential Access

---

- Simplest type
- File is **read in sequence from beginning to end**
- Can be slow unless you need to read all data in the file
- Example: editors and compilers

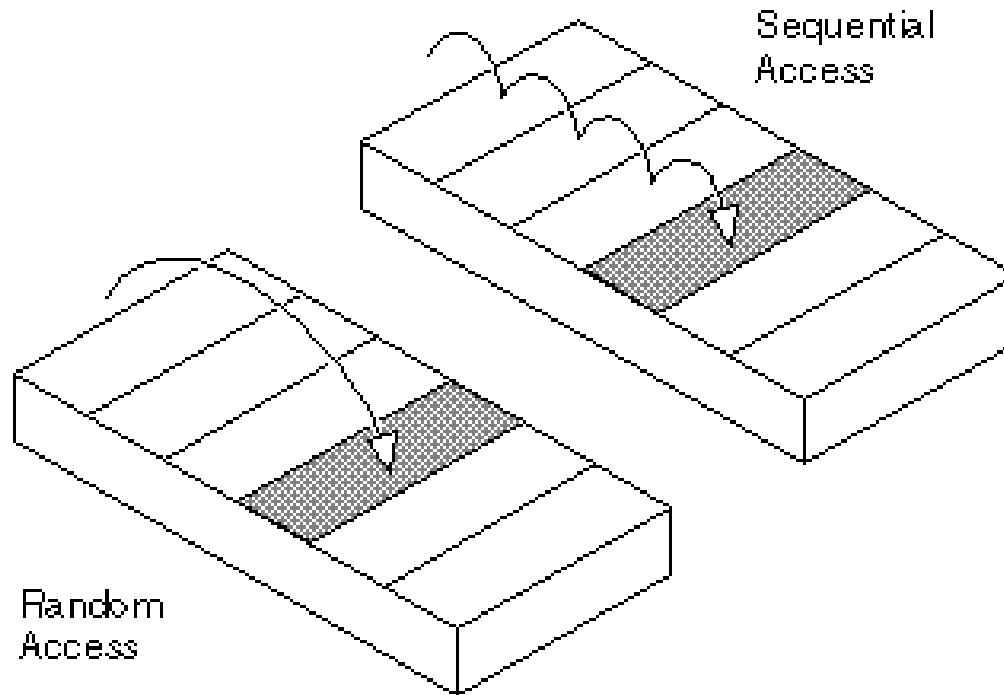
# File Access Methods

## Random (Direct) Access

---

- The file is viewed as a numbered sequence of fixed length **blocks**
- Allows read / write anywhere in the file
  - can randomly access it
- Allows for quicker file access
  - Can skip over large parts of the file that are not relevant

# Sequential vs. Random



# File Access Methods

## Indexed Access

- Another means for accessing and viewing records in a file by **key index**
- It is **a modification of the direct access approach** to finding files
- A file has an index (like a book) that contains pointers (address of where it is in memory) to the various blocks.
  - The index is searched first and then the pointer is used to access the file directly and find the desired record.

# File Allocation Methods

---

How do we store all these files  
on our Disk

# File Allocation Methods

- An **allocation method** refers to how disk blocks are allocated for files
- Disks provide the bulk of secondary storage on which a file system is maintained
- The main problem is
  - how to allocate space to these files so that disk **space** is utilised **effectively**, and
  - so that files can be **accessed quickly**

# File Allocation Methods

- A number of different approaches are used
  - Different File Systems use different approaches
- We look at **3 different types** of file allocation methods:
  1. **Contiguous**
  2. **Non-contiguous Linked – Block Chaining**
  3. **Non-contiguous Indexed – Block Mapping**
- Each approach uses the blocks in different ways and each have different advantages and disadvantages

# File Allocation Methods

## Contiguous Allocation

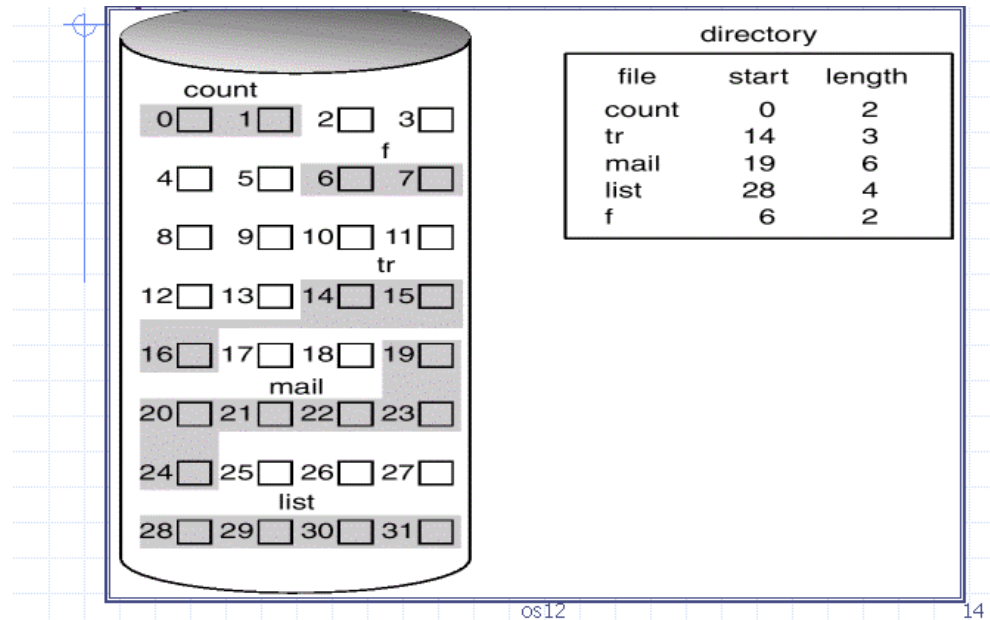
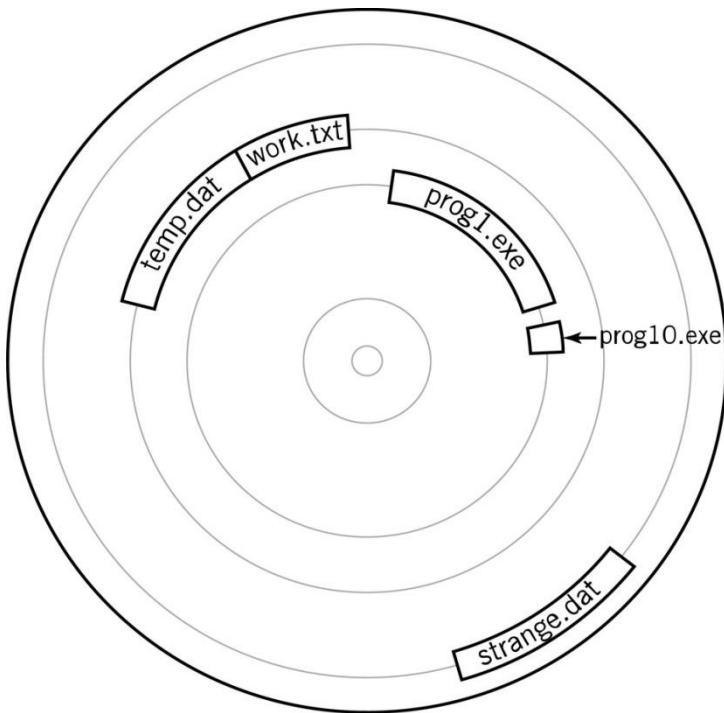
---

- Each file occupies a set of contiguous blocks on the disk
- Simple – only starting location (block #) and length (number of blocks) are required
- Random access
- Wasteful of space (dynamic storage-allocation problem)
  - What happens when a file is deleted and can never get a file small enough to fit?
- Files cannot grow



# Contiguous Storage Allocation

2 examples

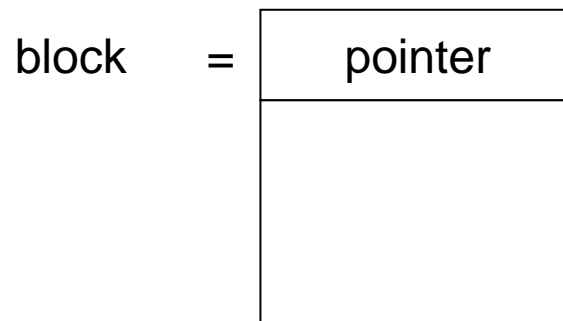


# File Allocation Methods

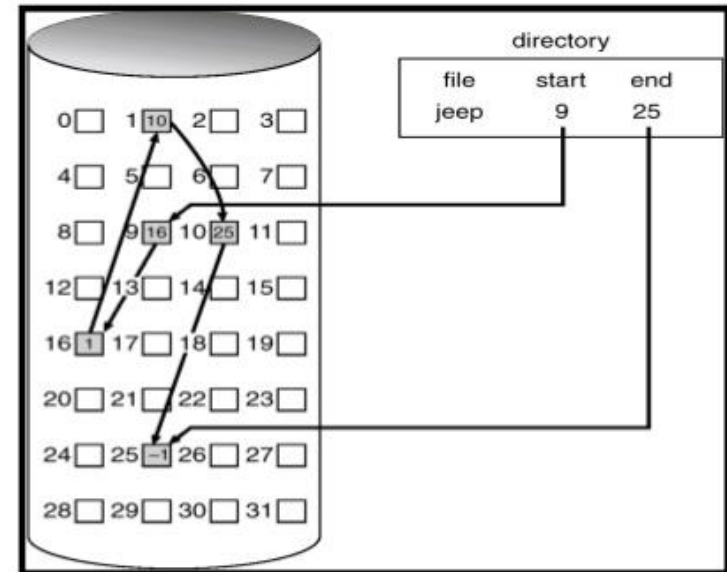
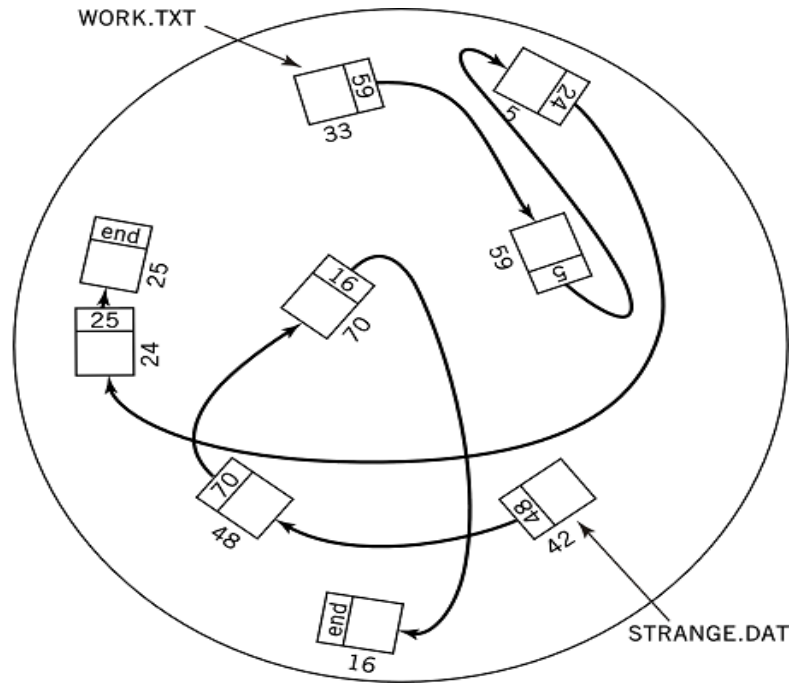
## Linked Allocation

- **Non-contiguous**

- Each file is a linked list of disk blocks:  
blocks may be scattered anywhere on the disk.
- Variant – links in both directions
- Can be thought of as **block chaining**



# Linked Allocation



<http://raj-os.blogspot.in/>

5

- Note: in the illustration above, the pointers are stored in the data blocks themselves.
- However, be aware that the linked-list of pointers may instead be stored externally from the data blocks (in a table for example.) 27

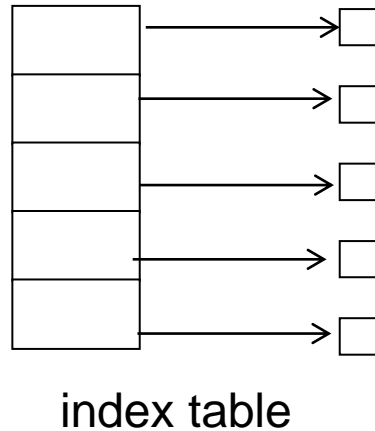
# File Allocation Methods

## Indexed Allocation

---

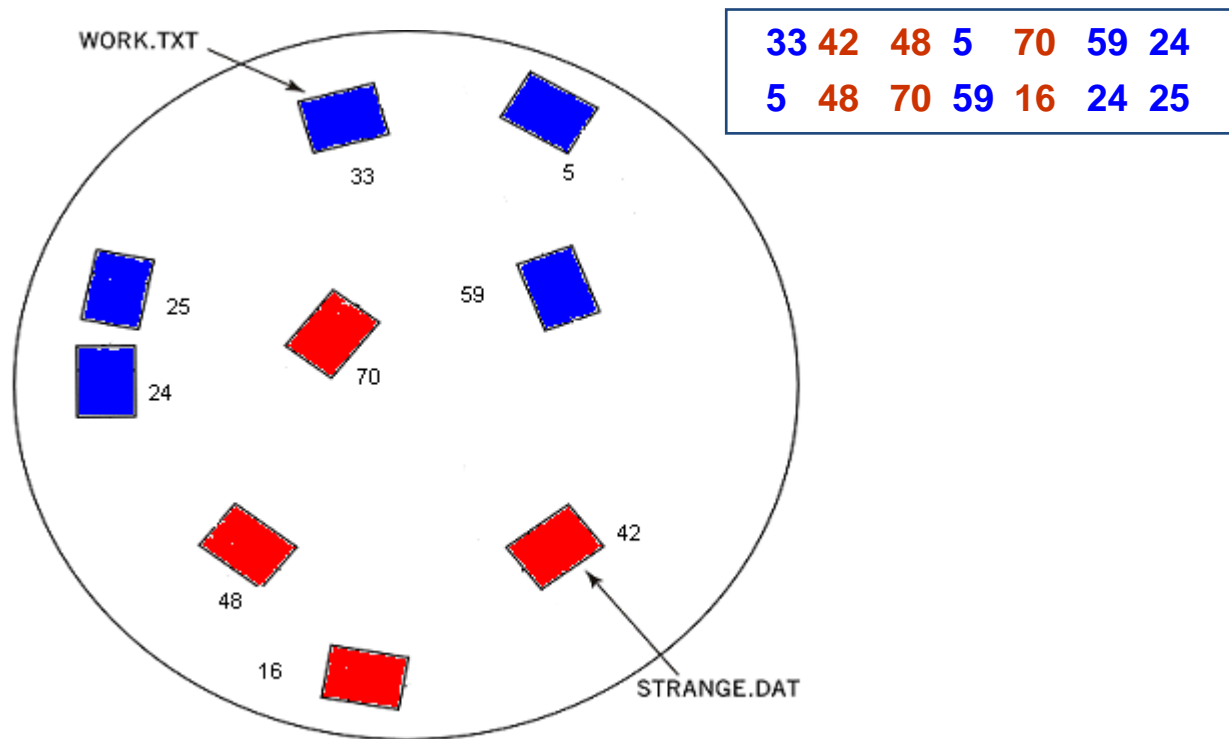
- Also called **Block Mapping**
- It is **Non-contiguous**
- It is an index which may be viewed as an externalised list of pointers to block addresses
- Frees up more of the block to store application data

# Indexed Allocation, continued

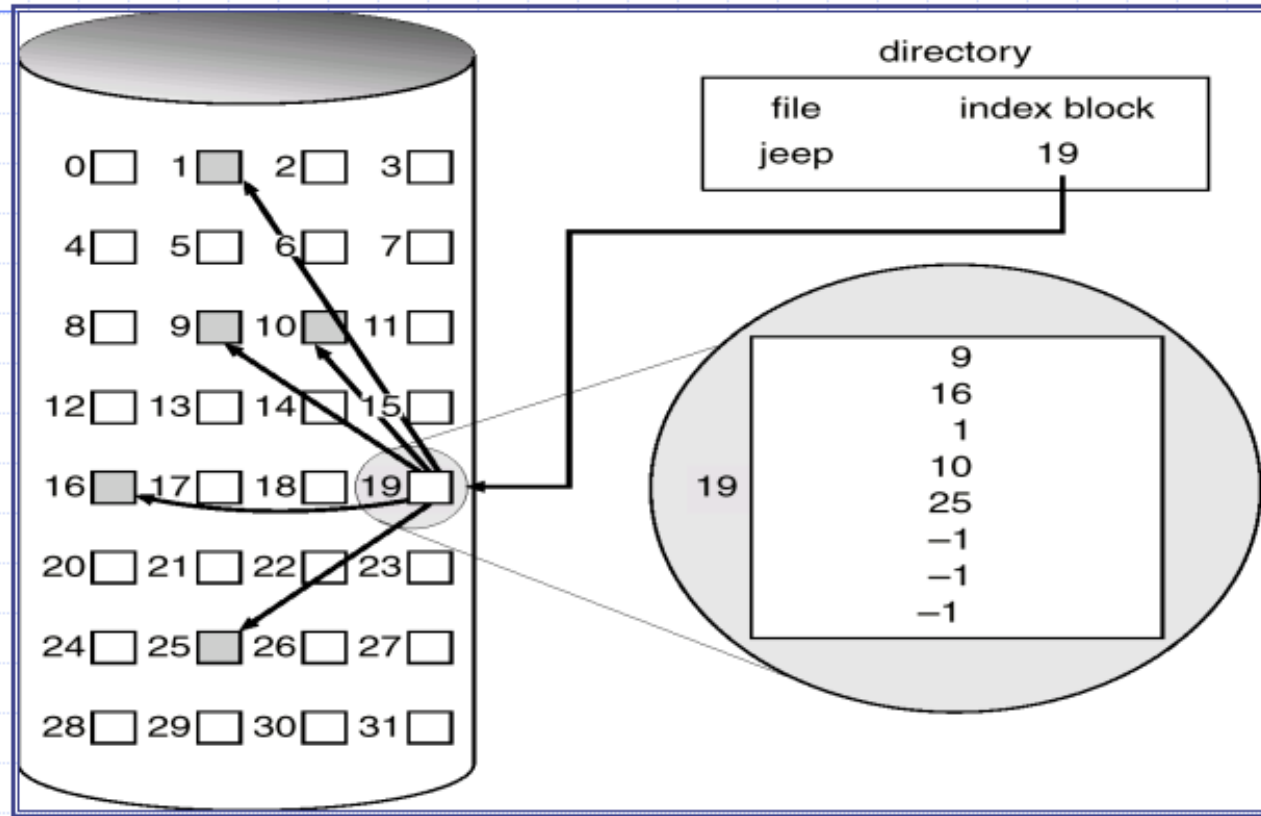


- Brings all pointers together into the *index block*.

# Indexed /Block -Mapped Allocation



# Indexed example



os12

22

# File Allocation Methods

---

Comparing them



# Comparison of File Allocation Strategies - Contiguous Storage

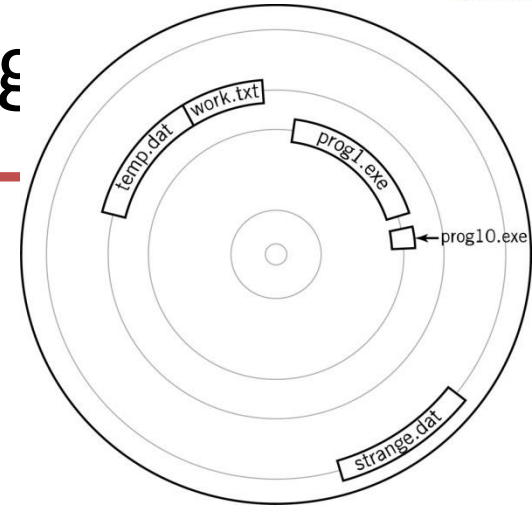
- Contiguous Structure

- *Advantages:*

- the most efficient way to read the file as all the file blocks are immediately adjacent to one another and can be read very quickly.

- *Disadvantages:*

- Very wasteful of disk space
    - Can lead to 'holes' appearing in the disk allocation strategy.
    - These holes add up to wasted and to fix this problem requires extensive disk activity to fill in the holes.



# Comparison of File Allocation Strategies – Block Linking/Chaining

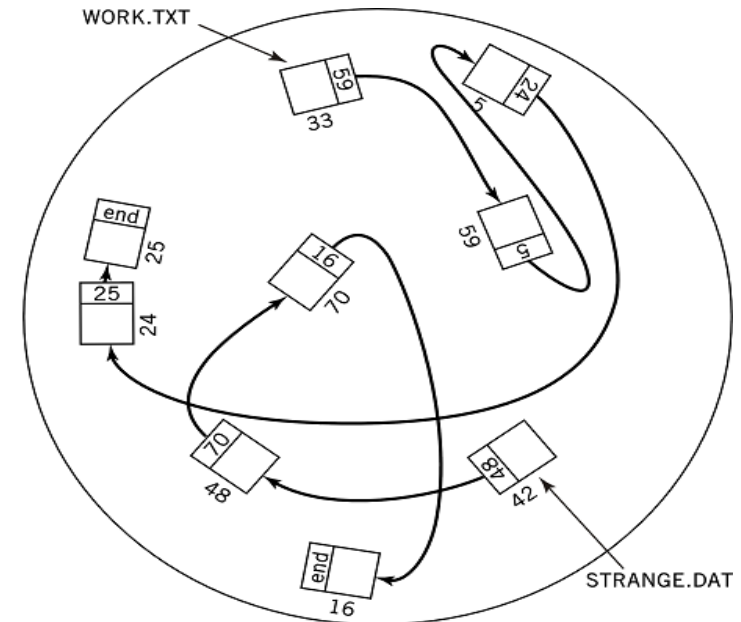
- Block Linked/Chained

- *Advantages:*

- Efficient use of disk space
      - Files can be placed in any block which can fit the data.
      - Adding to a file is easy
    - Free-space management system
      - No problem with wasted space, therefore no fragmentation problem

- *Disadvantages:*

- No Random Access
      - To access a block, you must read all prior ones
    - The chaining pointer requires extensive re-arrangement if the file is moved.
    - The chaining pointer uses up some space inside the data block itself, and the file data can use this space.



# Comparison of File Allocation Strategies – Block Mapping

- Block Mapping:

- *Advantages:*

- Doesn't require high maintenance chaining pointers inside the blocks, and no wasted space in the block
- Random rapid access

- *Disadvantages:*

- The map structure can take up a lot of storage space if the file is large.
  - Must be set aside from the start even if never fully used.
- The table that stores the map structure needs to be updated if the file is moved or altered.
- If map becomes damaged you cannot access your files <sup>35</sup>

