



# Comp 310

# Computer Systems and Organization

## Lecture #3

## The Components of an Operating System

Prof. Joseph Vybihal



# Announcements

- TAs: TBA
- Assignment #1 out by Friday or Monday
- Unix Tutorials soon



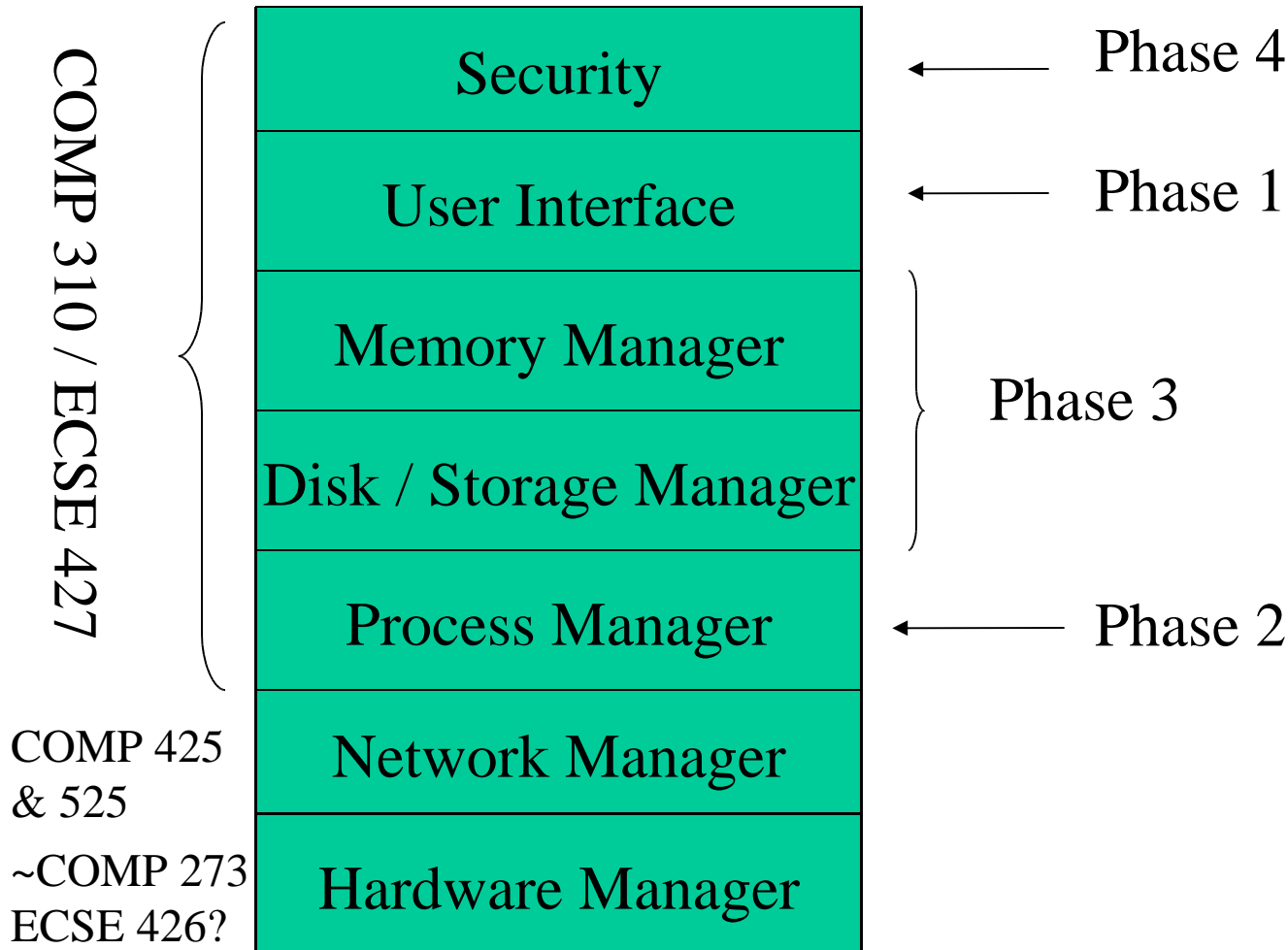
# Part 1

## Abstraction, Layers and Virtualization



# Basic OS Architecture

(Course Table of Contents!)



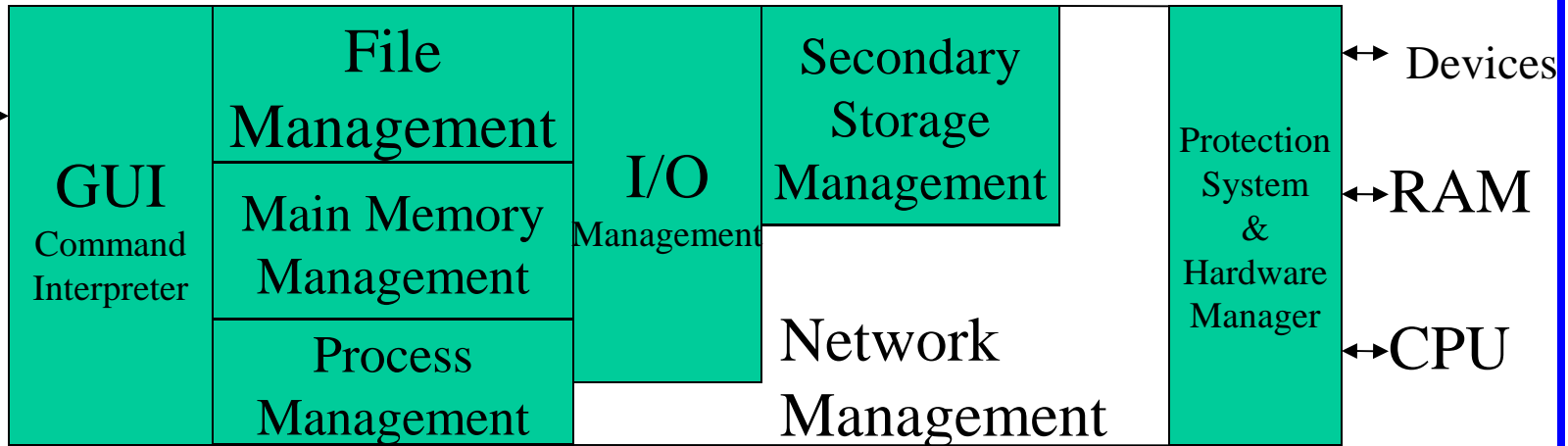


# Abstraction

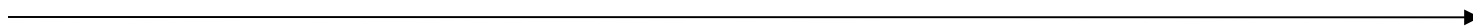
# Architectural Flow



USER



*Abstractions*



## Dependencies

Notice that the user of the computer is very far away from the actual hardware of the machine. Their understanding of how the hardware behaves is based on the OS layers.

**Virtualization**



# Building an OS is all about: Analogy, Abstraction and Virtualization

Define terms?



# For Example: Disk Anatomy



A write-protect square can prevent accidentally writing on the disk; the square's plastic window can be open (for write-protection) and closed with your fingernail.

A hard plastic cover protects the disk from dirt and damage.

A spring-loaded shutter exposes the surface of a mounted disk so it can be read from and written to.

A label can be placed on the disk to indicate its contents.

*My Data*

1.44 MB disks have a high density indicator hole here; 720 KB disks have no such hole.

A metal hub at the center of the disk is used to engage the disk in the drive.

Liners remove dirt from the disk's surfaces as it spins.

**FIGURE 2-13**

atomy of a diskette.

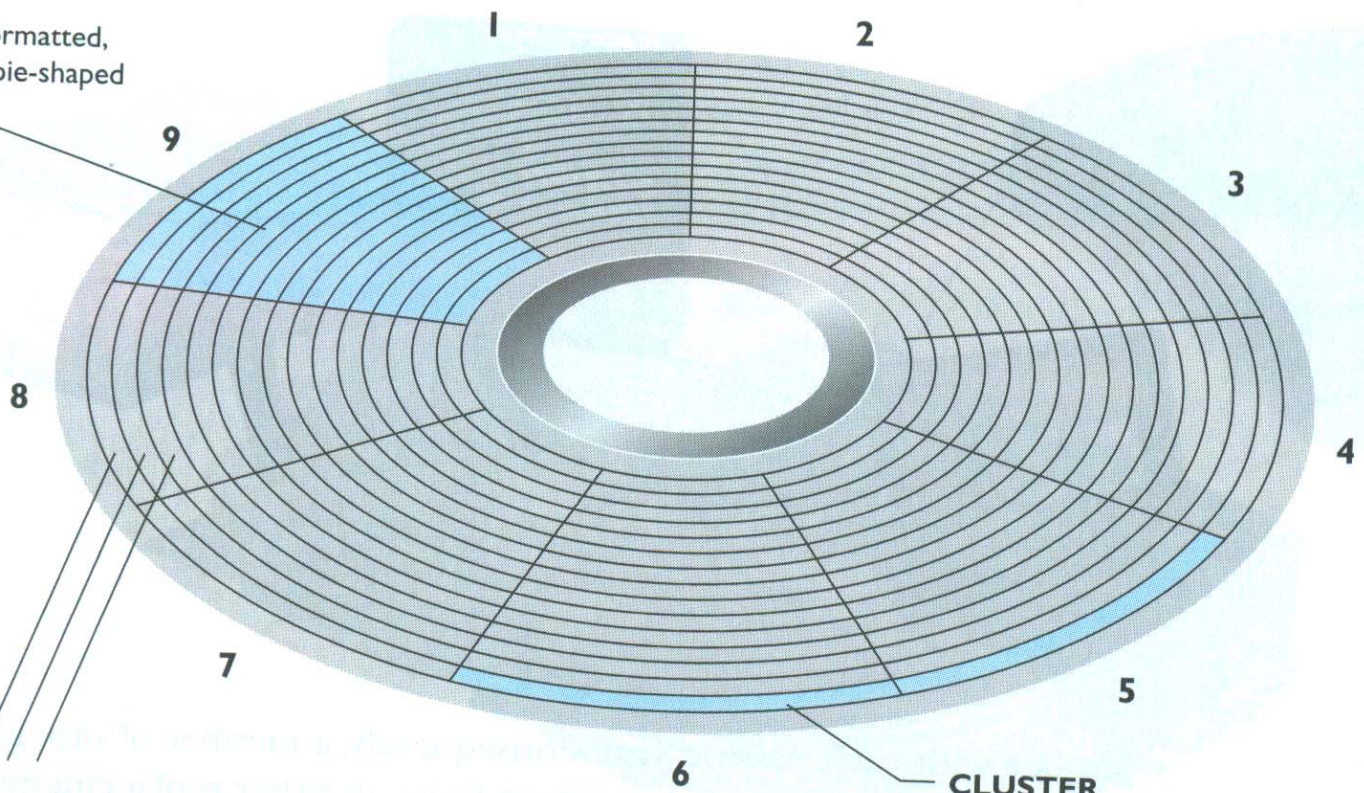
The plastic surfaces of the diskette are coated with a magnetizable substance so that data can be recorded.



# Disk Structure

## SECTORS

When a disk is formatted, it is divided into pie-shaped sectors.



## TRACKS

Disks come from the factory with concentric tracks on them. Tracks are encoded with 0- and 1-bits as data and programs are inscribed by users.

## CLUSTER

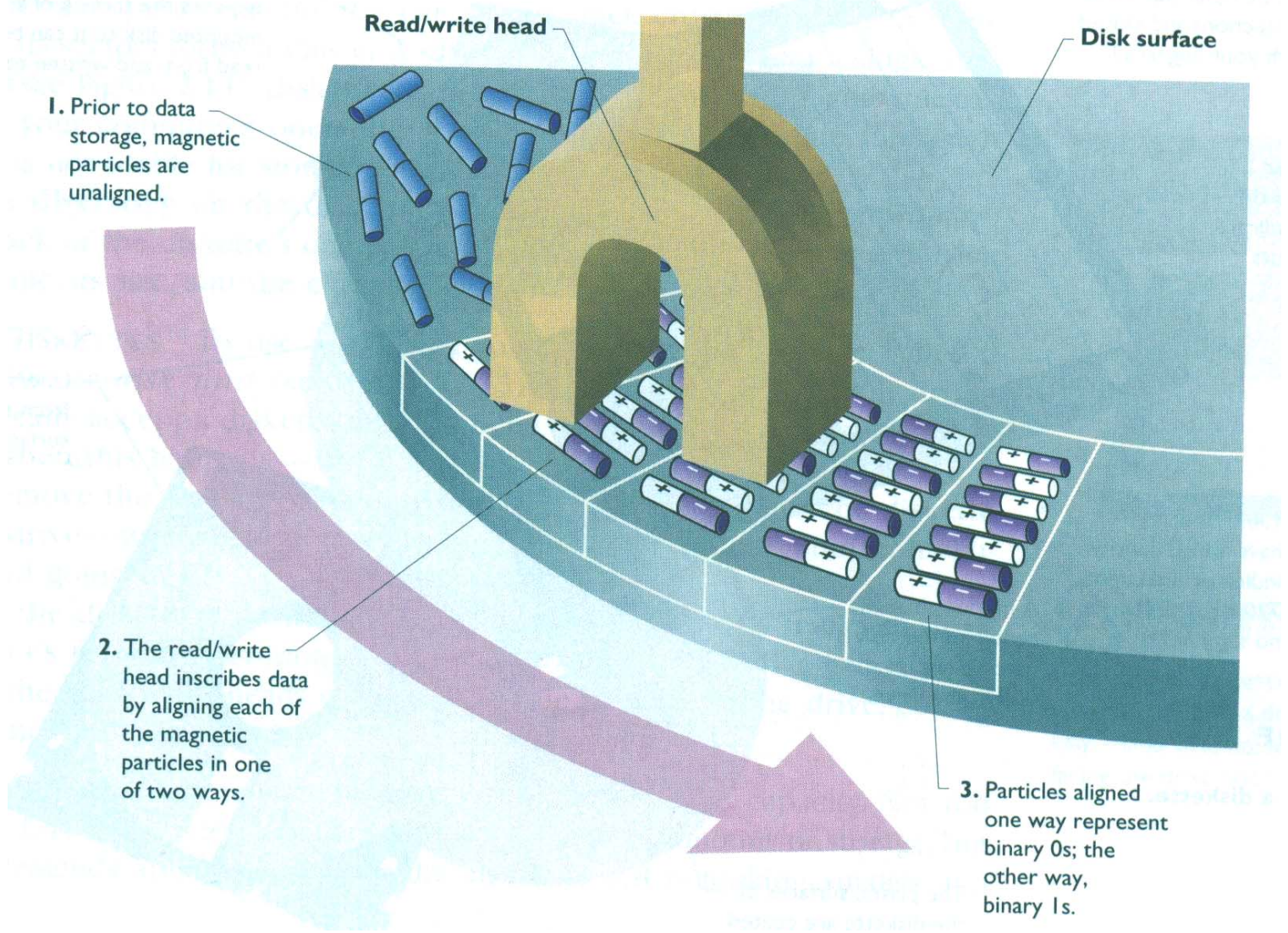
The part of a track crossed by two or more contiguous sectors forms a cluster, the smallest addressable unit of disk storage.

Formatting a disk...  
Other structures?





# Disk Physics



Two sided disks? 3D Volumes?



# How should files be built?

- File Structure? (Any suggestions)
  - Type 1: BOF Symbol with length in bytes
  - Type 2: EOF Symbol (Problems?)
- Folder / Directory Structure? (Any suggestions)
  - Type 1: Scan disk for files & built list is RAM
  - Type 2: File Allocation Table (Problems?)
    - File name
    - Pointer to the first byte of the file (track/sector/offset)
    - Statistics (size, date)
    - Ownership
    - Access Rights

Does it matter  
which way we do  
it?



# Important Question

- Since the computer is a machine, how can we make it usable by a user – how can we represent the computer to a user?
  - What analogies?
  - How do we display these analogies?
  - How can we program them?

(tabulate and discuss)



# Part 2

## A Rapid Component Overview



# The User Interface



# The Command Interpreter

What is the analogy here?  
How was it programmed?

## Purpose:

- Presents to the user an abstract interface to the computer
- Gets requests from the user
- Displays results & system messages to the user

## Types of Command Interpreters

- Job Control Language (in code)
  - load program, set paths, data source locator
- Batch/Bash Processing (as script)
  - script language to execute many OS commands (Unix & DOS use it)
- Command Line
  - text driven, command driven (DOS & Unix)
- GUI
  - icons, folders, windows (Mac, Windows, Unix)

```
Microsoft(R) Windows DOS
(C)Copyright Microsoft Corp 1990-2001.

C:\>mem

655360 bytes total conventional memory
655360 bytes available to MS-DOS
578252 largest executable program size

4194304 bytes total EMS memory
4194304 bytes free EMS memory

19922944 bytes total contiguous extended memory
0 bytes available contiguous extended memory
15580160 bytes available XMS memory
MS-DOS resident in High Memory Area

C:\>
```



# Windowed comparison

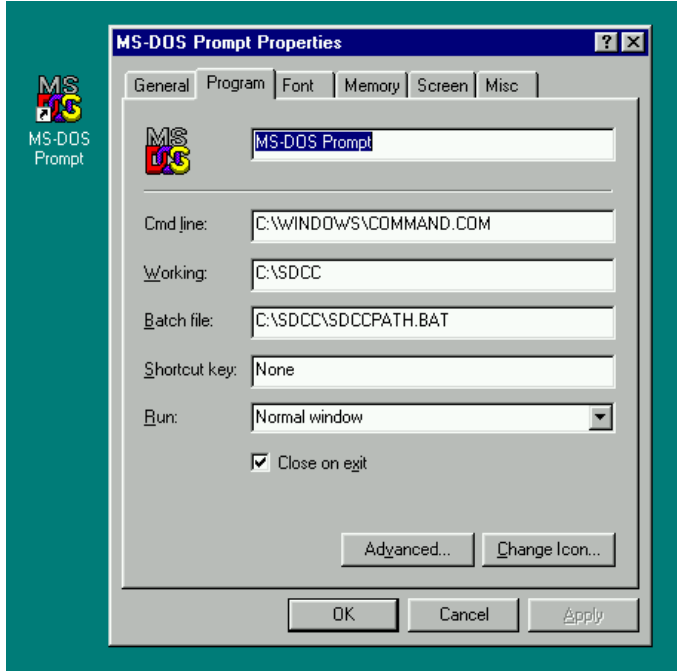
*Batch Processing puts this into a script file*

```

COMMAND LINE

C:> dir
C:> cc -o myprog myprog.c
C:> dir myprog.exe
C:> myprog

```



```

Microsoft(R) Windows DOS
(C)Copyright Microsoft Corp 1990-2001.

C:\>mem

 655360 bytes total conventional memory
 655360 bytes available to MS-DOS
 578352 largest executable program size

4194304 bytes total EMS memory
4194304 bytes free EMS memory

19922944 bytes total contiguous extended memory
 0 bytes available contiguous extended memory
15580160 bytes available XMS memory
 MS-DOS resident in High Memory Area

C:\>

```

Single-user multi-process

*Single-user multi-process in Unix when we use the & terminator*

Single-user single-process



## *JOB CONTROL LANGUAGE*

```
exe = myprog.class  
in_data = c:\data\sample.txt  
out_data = <SCREEN>  
Tape = accounting
```

Used between programs in a queue waiting to be executed.

An OS programming language.

## *BATCH PROCESSING*

(text file: doit.bat)

```
del *.obj  
cc -o myprog myprog.c  
if (exist myprog.exe) myprog  
else echo "File not found"
```

Almost a full programming language!  
Batch interpreters are built into the OS  
with their own run-time memories  
(shell memory).





# Why different user interfaces?

## Benefits

- GUI?
  - Short learning curve
  - More accessible to public
  - Attractive look and feel
- Command-Line?
  - Del J\*.obj much faster than clicking and locating all J\*.obj files
  - Direct access to OS switches and settings (faster to do things)
  - Programmable
- Batch/Script Processing?
  - Personal procedural automation
  - Job Batching (sequentially grouping your work in an automated process)
- Job Control Language?
  - Pre-load resources before program runs
  - Notify OS the resources needed (OS can now schedule for it)
  - Alleviates resource request congestion



# Managing Files



# File Management

## Defines the Meaning of:

What is the analogy here?  
How was it programmed?

- The “file” concept (physical structure of files on medium)
- File organization (directories, folders, rooms, doors)
- Accessing secondary devices (naming, addressing, security)

## Operations to Program:

- Create, delete, read, write, append to a file (abstract functions)
- Directory / Folder structure
- Backing up of files
- Security Privileges
- Commands to access file by byte address

*An abstraction*



# Managing Memory

Suggest implementations?

# Main Memory Management

## Operations:

- Tabulates free and used memory
- Operates the Run-time Stack and Heap for each process
- Knows which memory segment is owned by which user
- Allocates and de-allocates memory

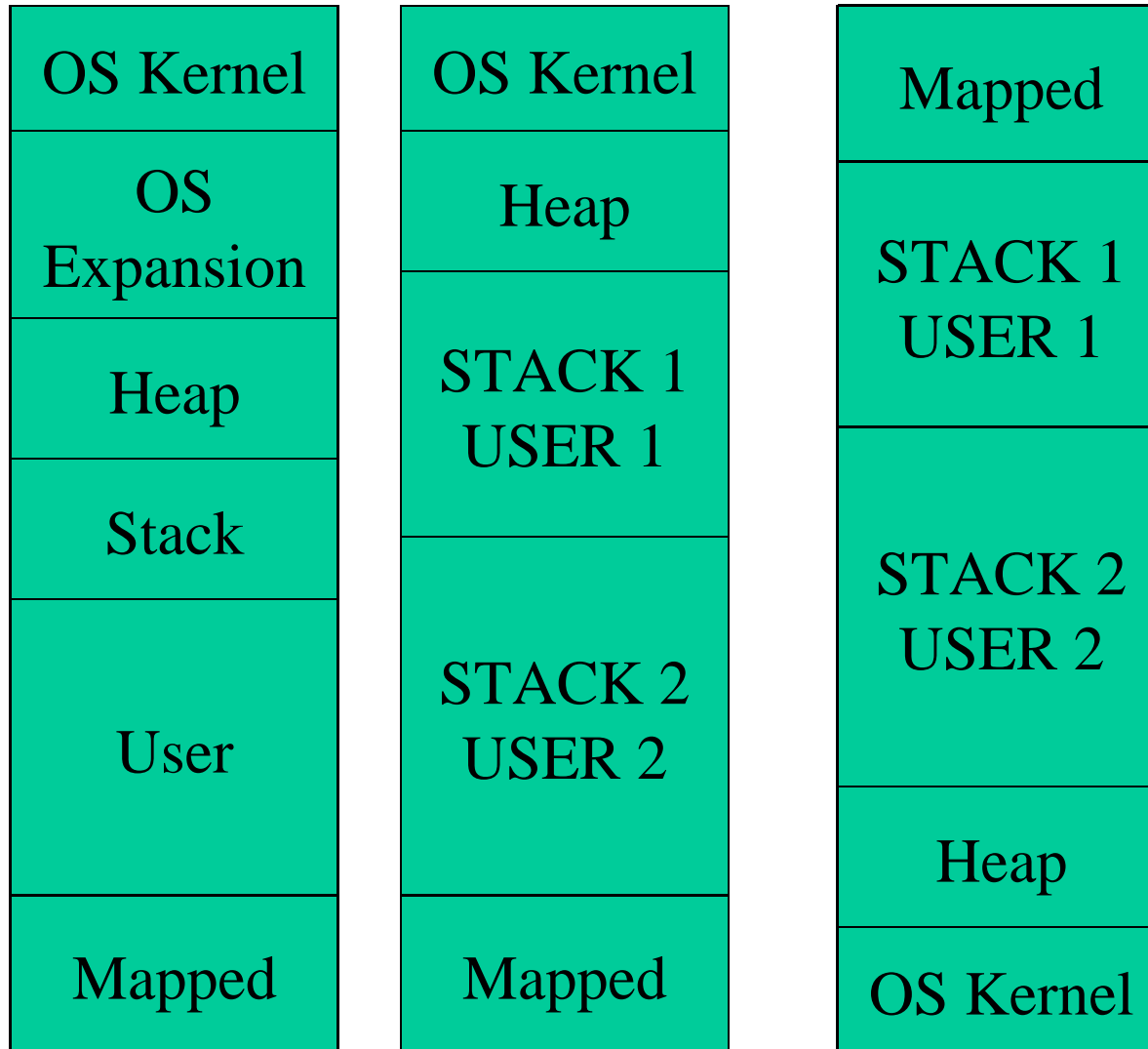
## Defines the Meanings of:

- Where things are to be stored in RAM
- OS Space, User Space, System Space, etc.
- Virtual Memory
- Shared Memory





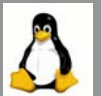
# Example RAM Organizations



Is there a good way?



# Managing Executing Programs (ie. Processes)



# Process Management

## Defines the Meaning of:

- The “executing program” concept (called a process)
- Process intercommunication

## Operations:

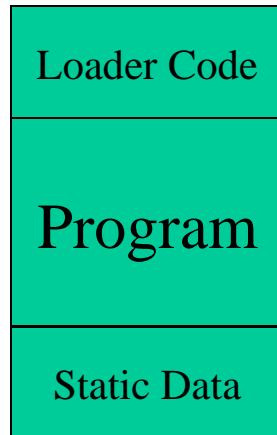
- The program counter, instruction counter, register swap area
- Task Switch concept
- Creating, deleting, suspending, resuming a process
- Process synchronization, communication & deadlocking
- Caching: strategies that optimize the use of the CPU cache when executing user programs

Low-level of OS (often in assembler)

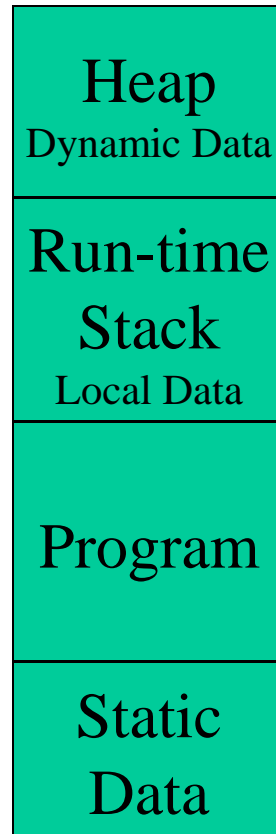




## ON DISK



## IN RAM



## OS PROCESS MAP

### Table of Processes

- Start address
- Owner
- Boundary (min/max space)
- Status (run,suspend,error)
- Current instruction exec
- CPU Register Swap Area
- Bottom and Top of stack
- Heap head pointer
- etc.

Constructed as a table  
or a linked list.



# Managing I/O Devices



# I/O Management

## Operations and Properties:

- Low-level of OS (often in assembler)
- Buffering, caching and spooling algorithms
- Device-driver routines
- Additional-driver management space (at install time)
- I/O Interrupt table

## Notes:

- Buffer: commonly an array of bytes used to speed the access of data
- Spooling: Fundamental queuing routines for printers, CPU, etc.
- Caching: Strategy used to temporarily store possibly needed info in RAM



# Secondary Storage Management

## Operations and Properties:

- Low-level of OS (often in assembler)
- Free-space management (de/fragmentation of files)
- Storage allocation and organization (physical byte organization)
- Disk scheduling routines
- The actual assembler programs to read/write/append files
- Access file given a byte → track/sector/offset/cylinder

How does data get written?

**File Manager** deals with files from an abstract higher level while **Secondary Storage Manager** is related to the actual physical nature of data on a disk, works closely with the **I/O Manager**.



# Hard Disk Drive

## SEALED PACK

The hard disk is hermetically sealed in a case to keep it free of air contamination.

## CIRCUIT BOARD

Below the disk is a circuit board that contains the disk controller. This board makes sure the disk is rotating at a constant speed and tells the heads when to read and write.

## READ/WRITE HEADS

There is a read/write head for each disk surface. On most systems, the heads move in and out together and will be positioned on the same cylinder.

## MOUNTING SHAFT

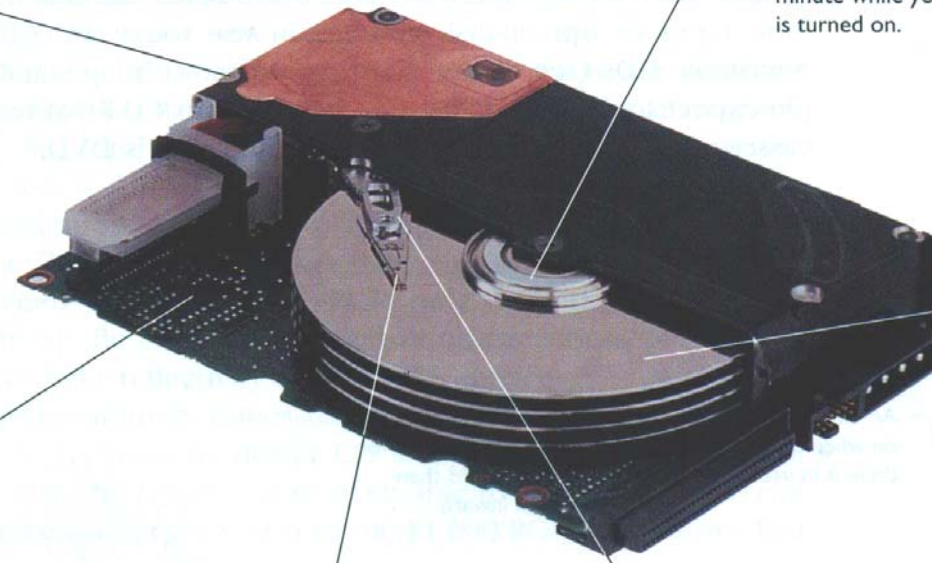
The mounting shaft is always spinning at a speed of several thousand revolutions per minute while your computer is turned on.

## DISK CYLINDERS

On hard disks, the same relative track on each surface forms a disk cylinder. Cylinders are used in the formation of disk addresses.

## ACCESS MECHANISM

The access mechanism moves the read/write heads in and out together between the disk surfaces to access required data.



A byte = Cylinder, Sector, Track, Offset path

# Network Management

## Operations:

- Communication and loss of data algorithms:
  - This module takes care of the actual opening of a communication path from one machine to another and how to convert the data into a signal that will be transmitted over the medium.
- Deadlock avoidance:
  - The process by which we overcome those times when two processes have usage access to a device the other needs.
- Node location management:
  - how do we refer to and address resources on the network?

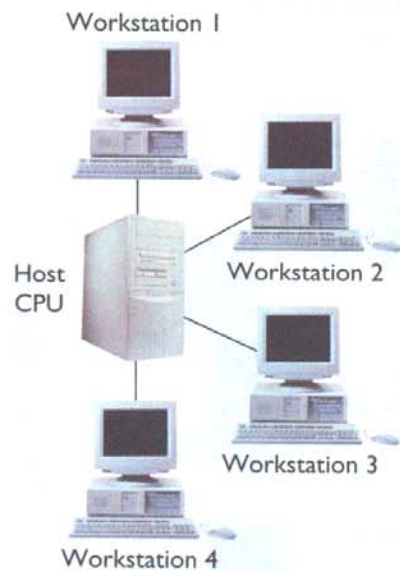




# Topology Knowledge

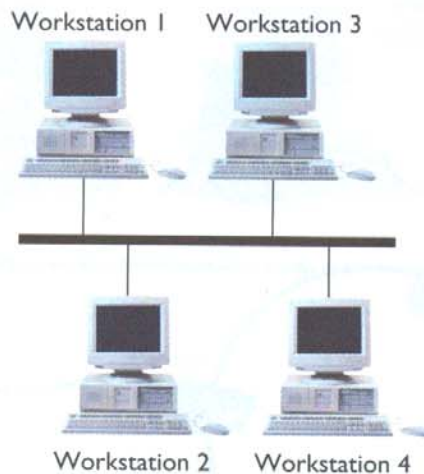
## BASIC TOPOLOGIES

Most network topologies follow a simple star, bus, or ring pattern.



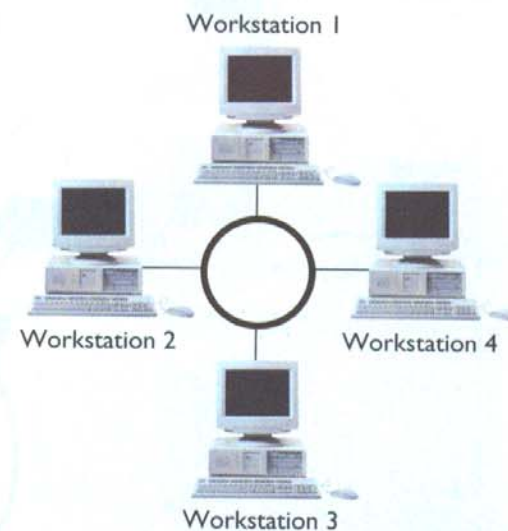
### STAR NETWORK

A star network often consists of a mainframe host that's connected to several workstations in a point-to-point fashion.



### BUS NETWORK

A bus network uses a high-speed cable that workstations tap into—in the manner shown—to pick up and drop off messages.



### RING NETWORK

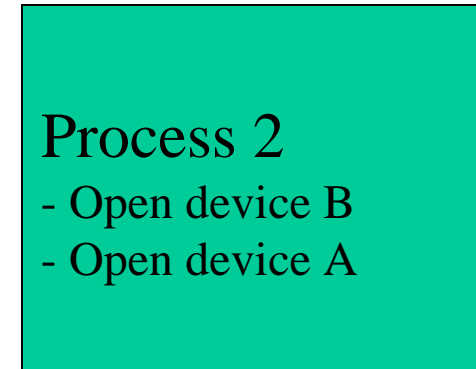
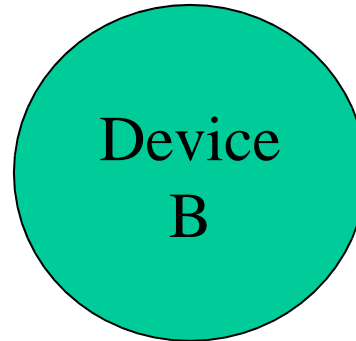
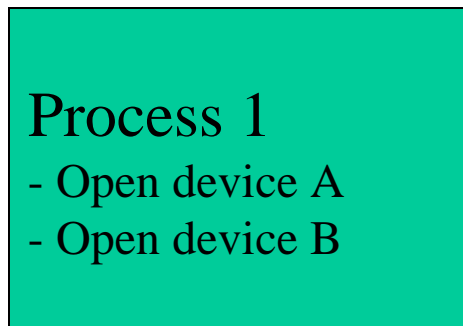
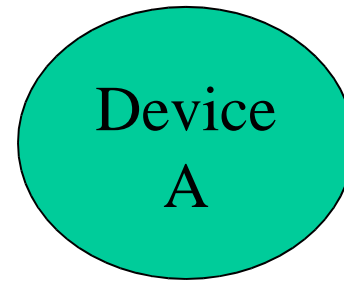
In a ring network, computers and other devices are connected in a loop.

- Topology
- Communication Rules
- Signals & Mediums
- Encryption ...



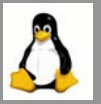
# Deadlocking

- Networking
- Processes
- Resources



Assuming that both processes are executing at the same time, we now have a problem. P1 has access to A and P2 has access to B. Now P1 wants access to B but must wait until P2 is done with B. **BUT** P2 wants access to A and must wait until P1 is done - **STUCK!**





# Security Issues



# The Protection System

The following devices **MUST** never be tampered with:

- RAM - A process can access only its own space
- DEVICE - Only one process can access a resource at a time
- OS - No one should be able to damage the OS
- PROCESS - Must only communicate via the OS
  - Should not have direct access to the hardware
  - Should not have direct access to any other process



# Part 2

## At Home



# Things to try out

- Get two applications to access the same resource. What does your OS do?

Try with:

- Files (open, delete)
  - Two FAX Applications faxing at the same time
2. Become familiar with the Linux or DOS command-line interface. Learn how to use any 20 commands.