


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



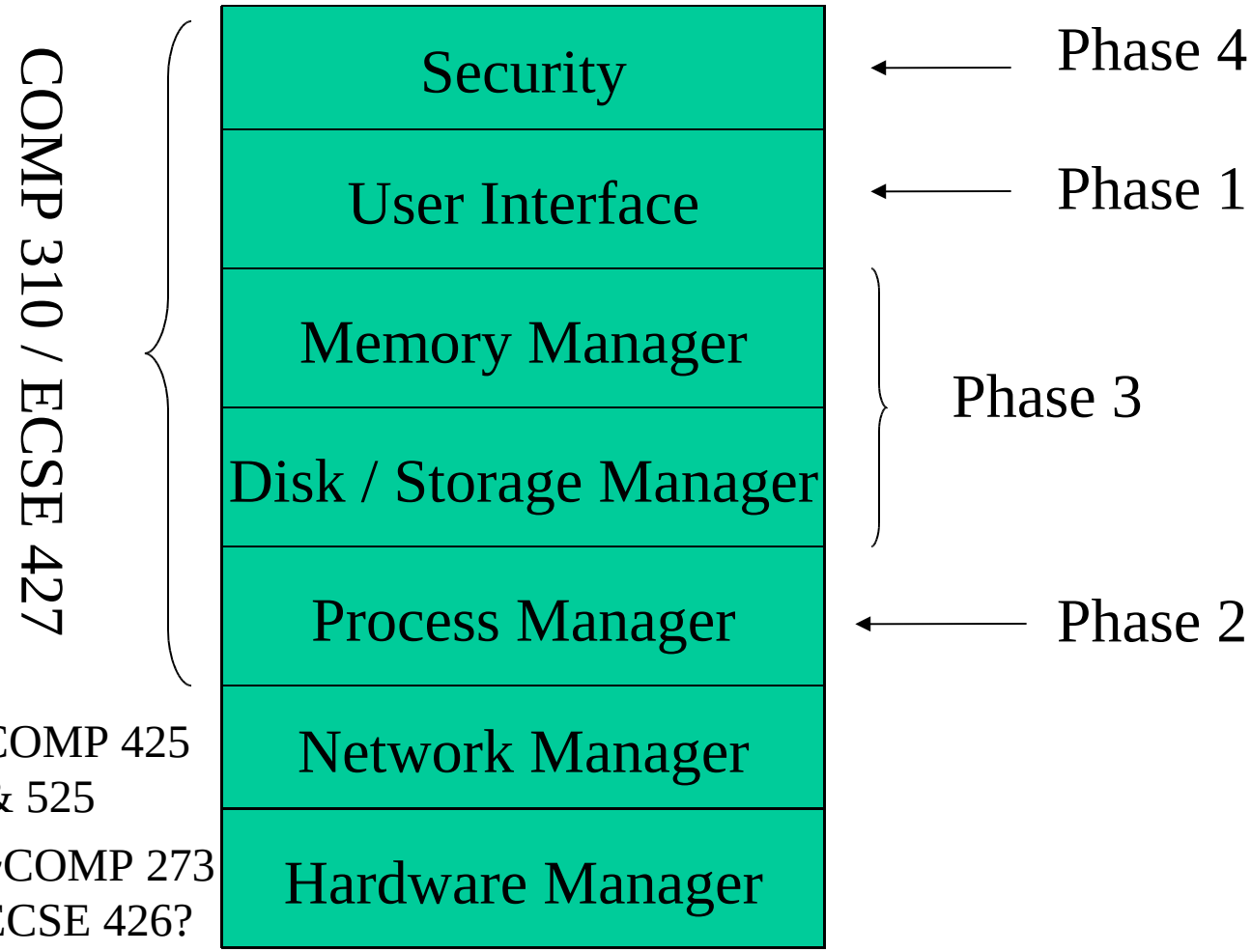

Part 1

Abstraction, Layers and Virtualization




Basic OS Architecture

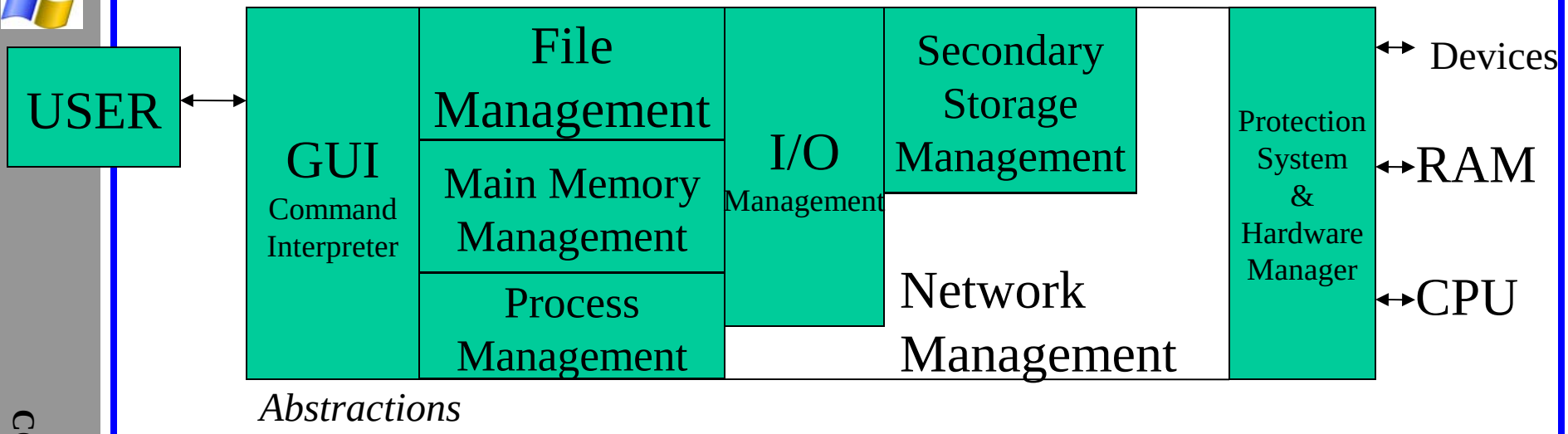
(Course Table of Contents*!*)






Abstraction

Architectural Flow



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.




Building an OS is all about:

Analogy, Abstraction and Virtualization

Define terms?

For Example: Disk Anatomy

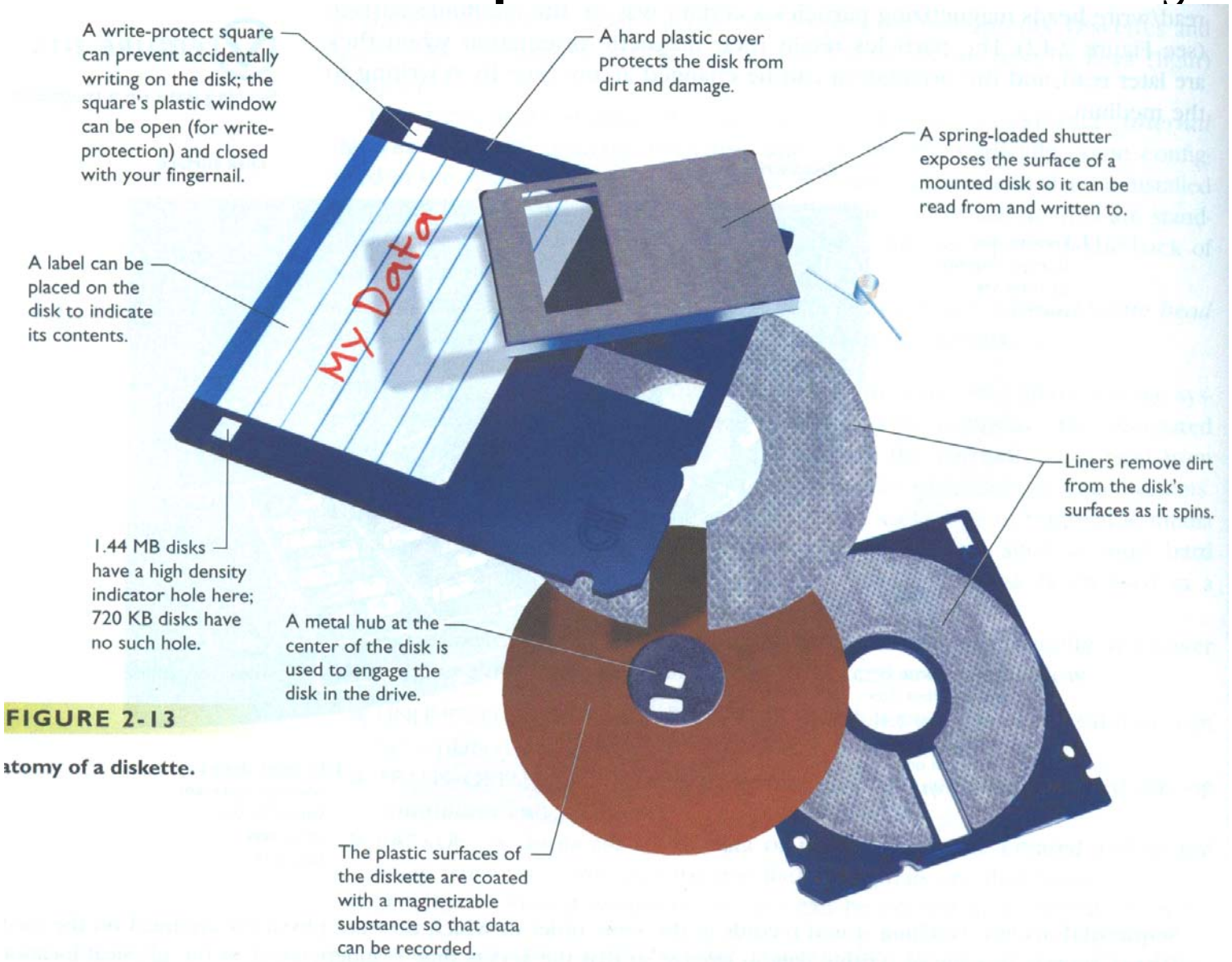
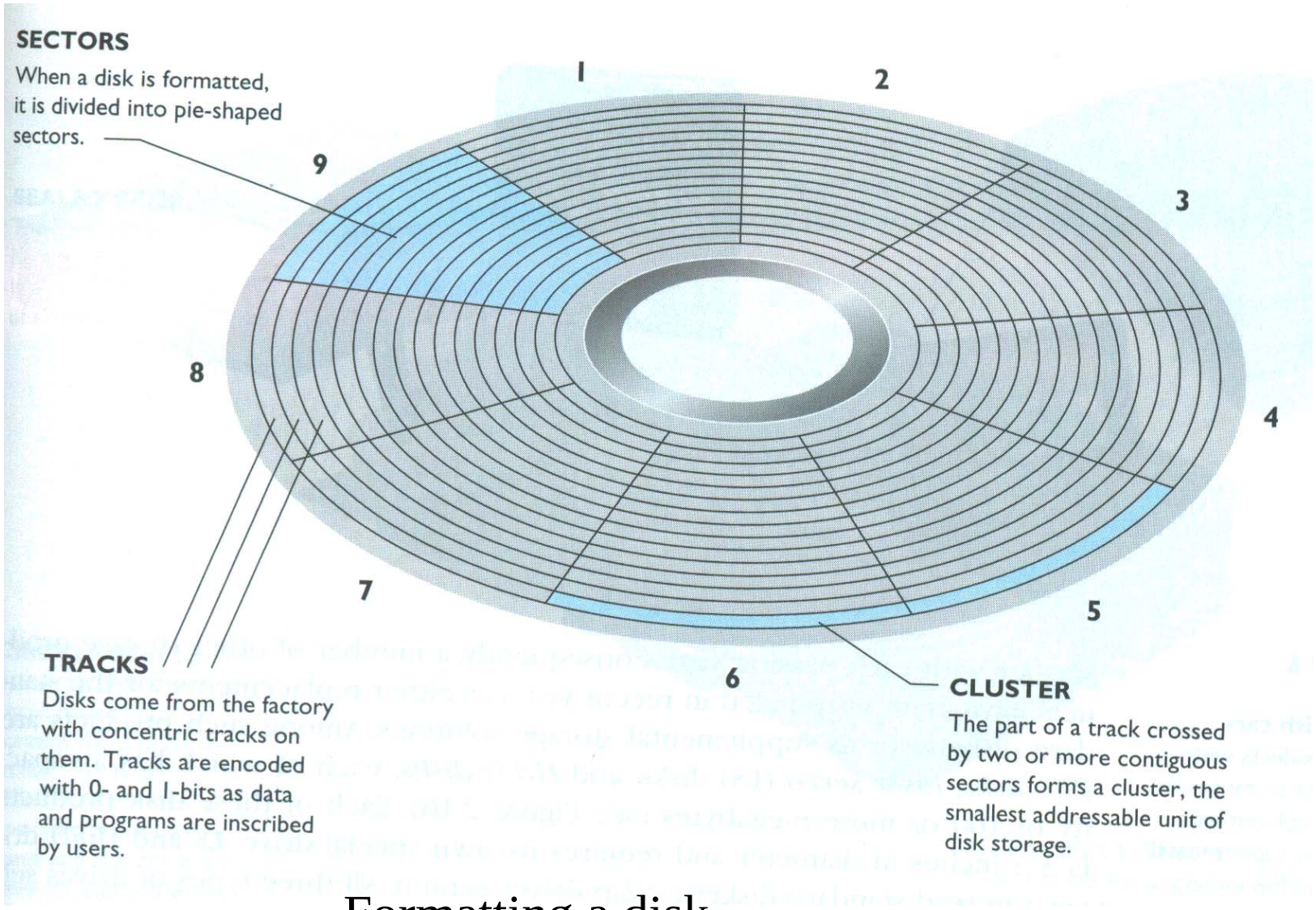


FIGURE 2-13

atomy of a diskette.



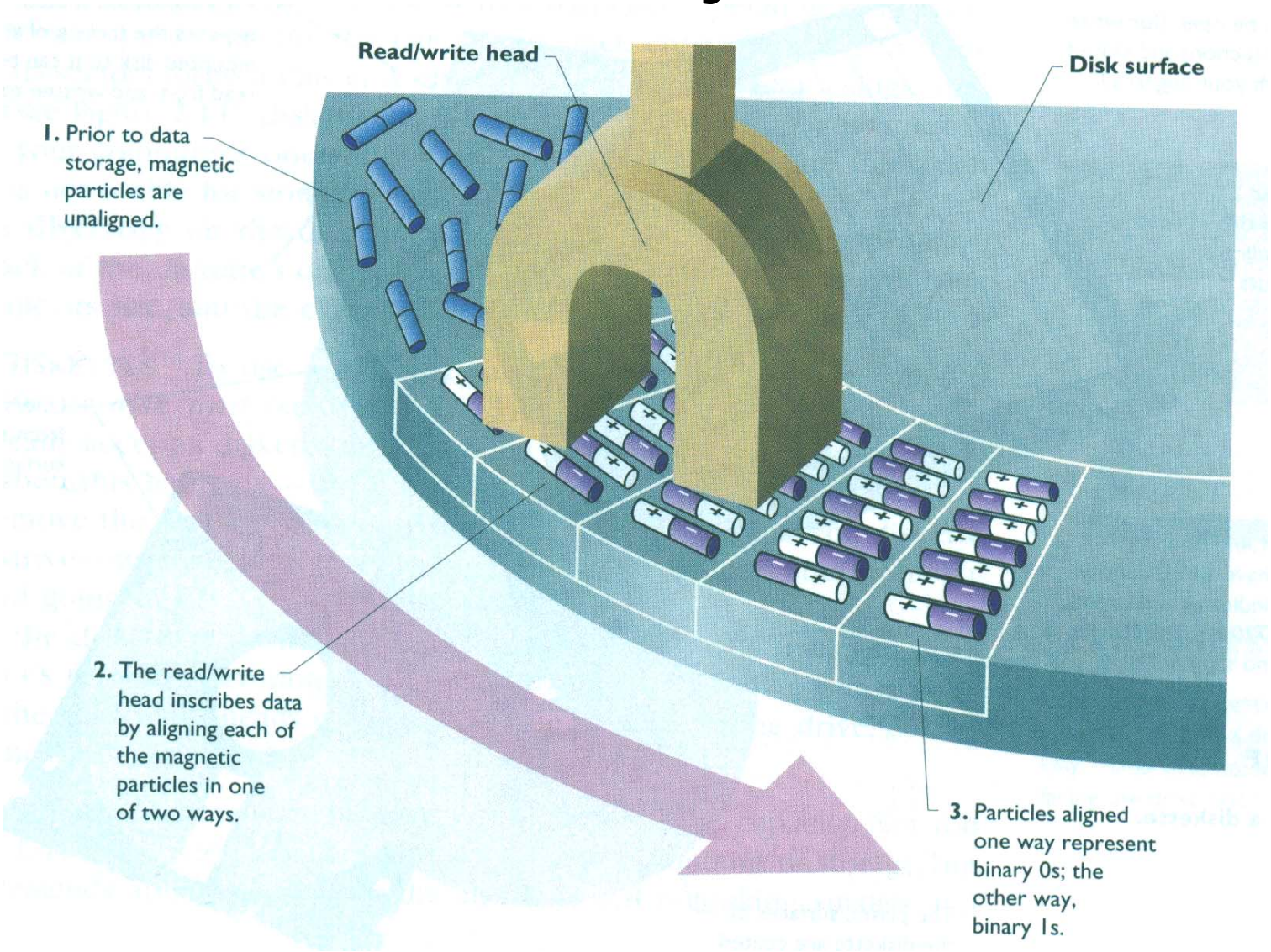

Disk Structure



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 locater
- 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)

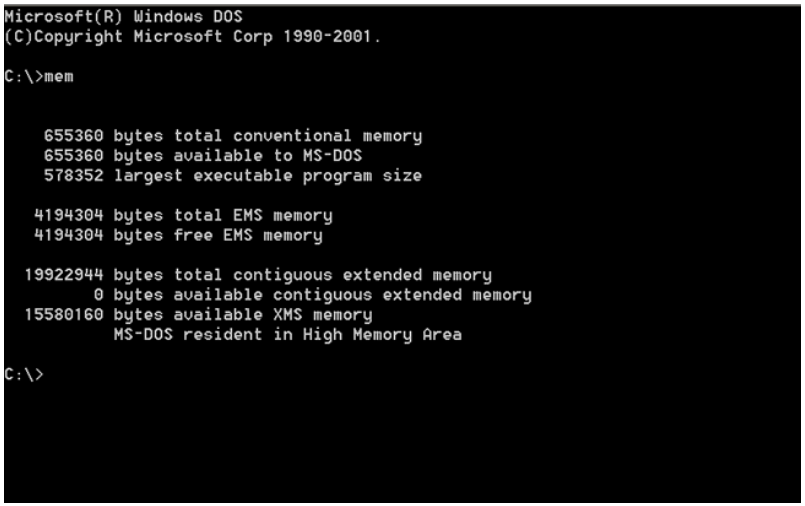



Windowed comparison

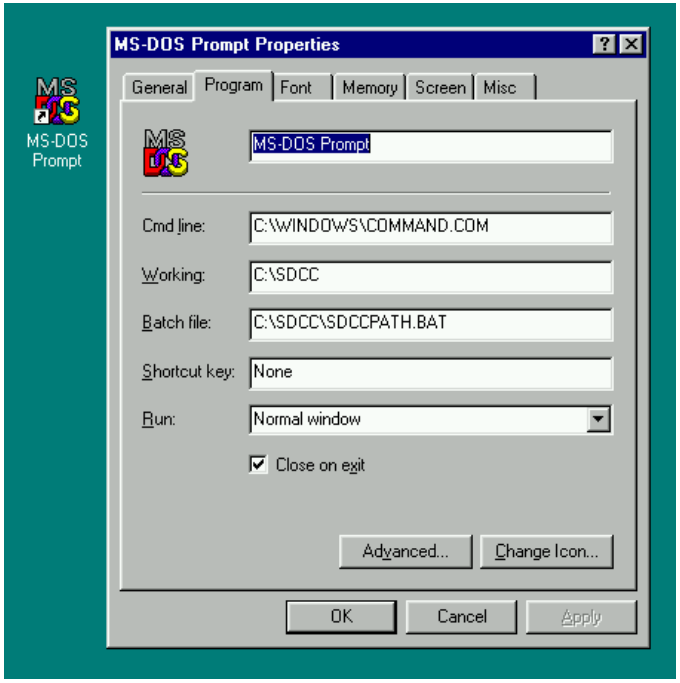
Batch Processing puts this into a script file

COMMAND LINE

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



Single-user single-process



Single-user multi-process

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




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 then 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

What is the analogy here?
How was it programmed?

Defines the Meaning of:

- 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




Main Memory Management

Operations:

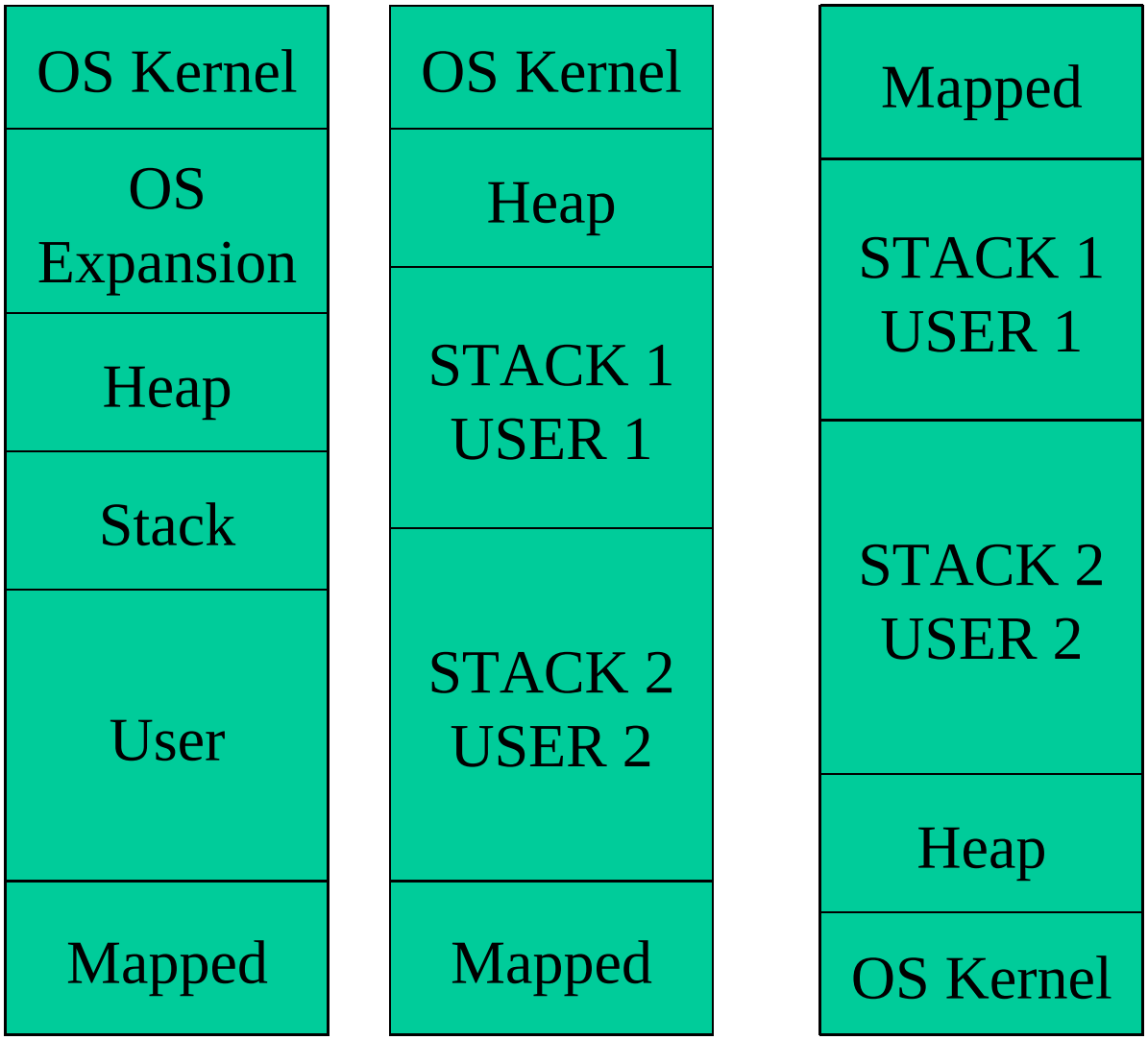
- Tabulates free and used memory
- Operates the Run-time Stack and Heap for each process
- Knows which memory segment is owned by which user
- Allocates and de-allocates memory

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

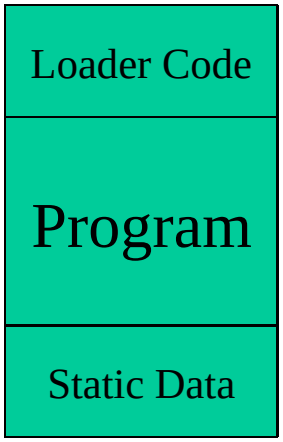
Operations:

- 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

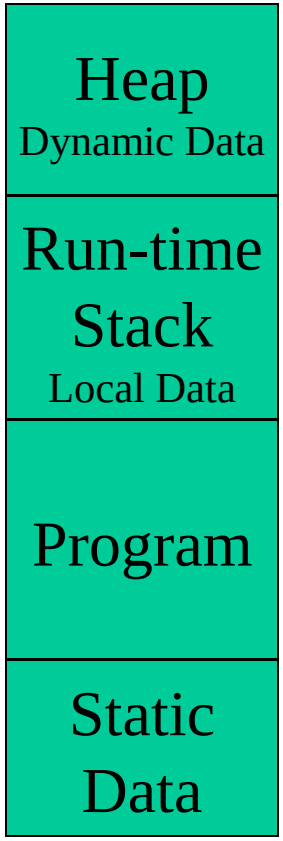
Low-level of OS (often in assembler)




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

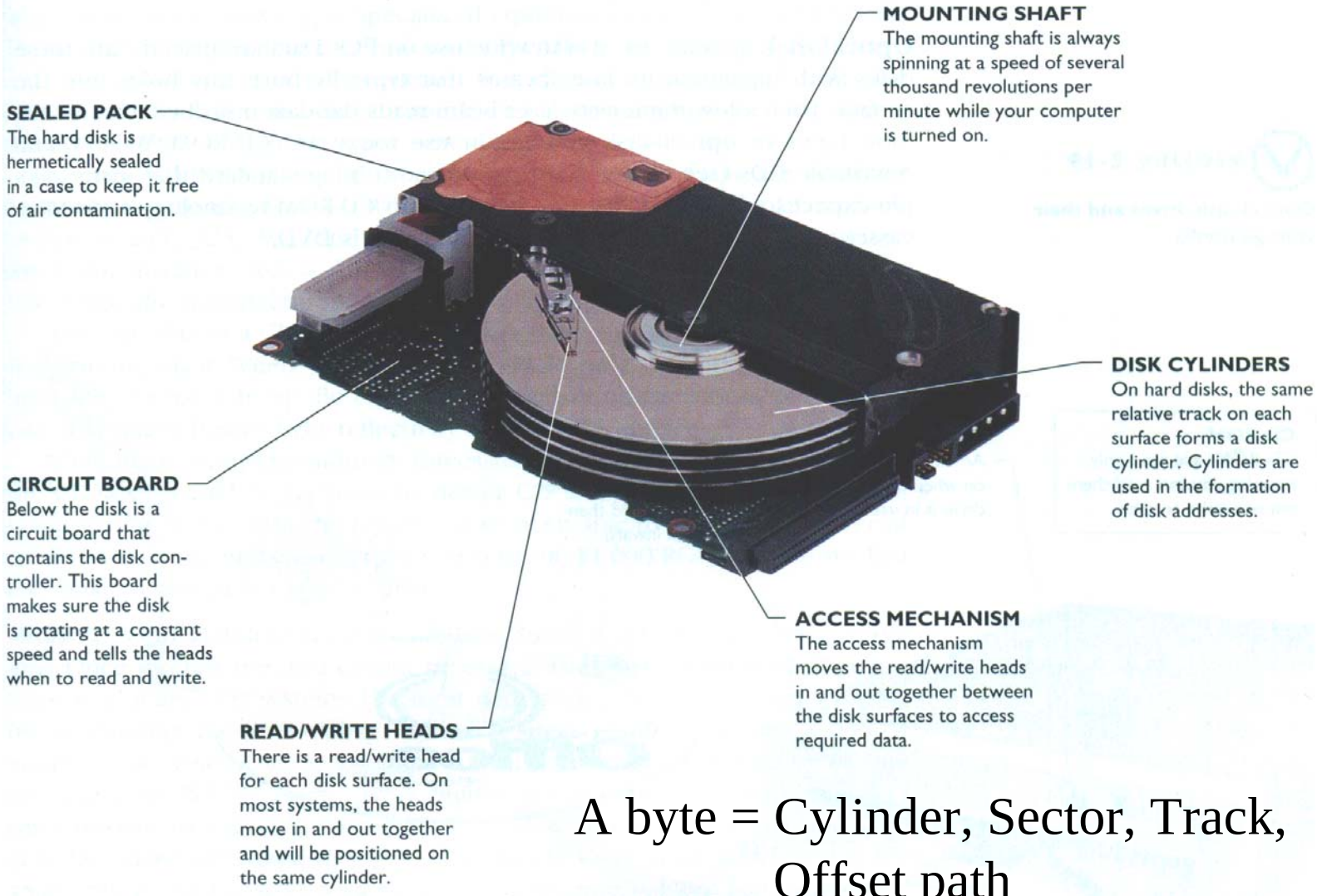
Operations and Properties:

- 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



A byte = Cylinder, Sector, Track, Offset path

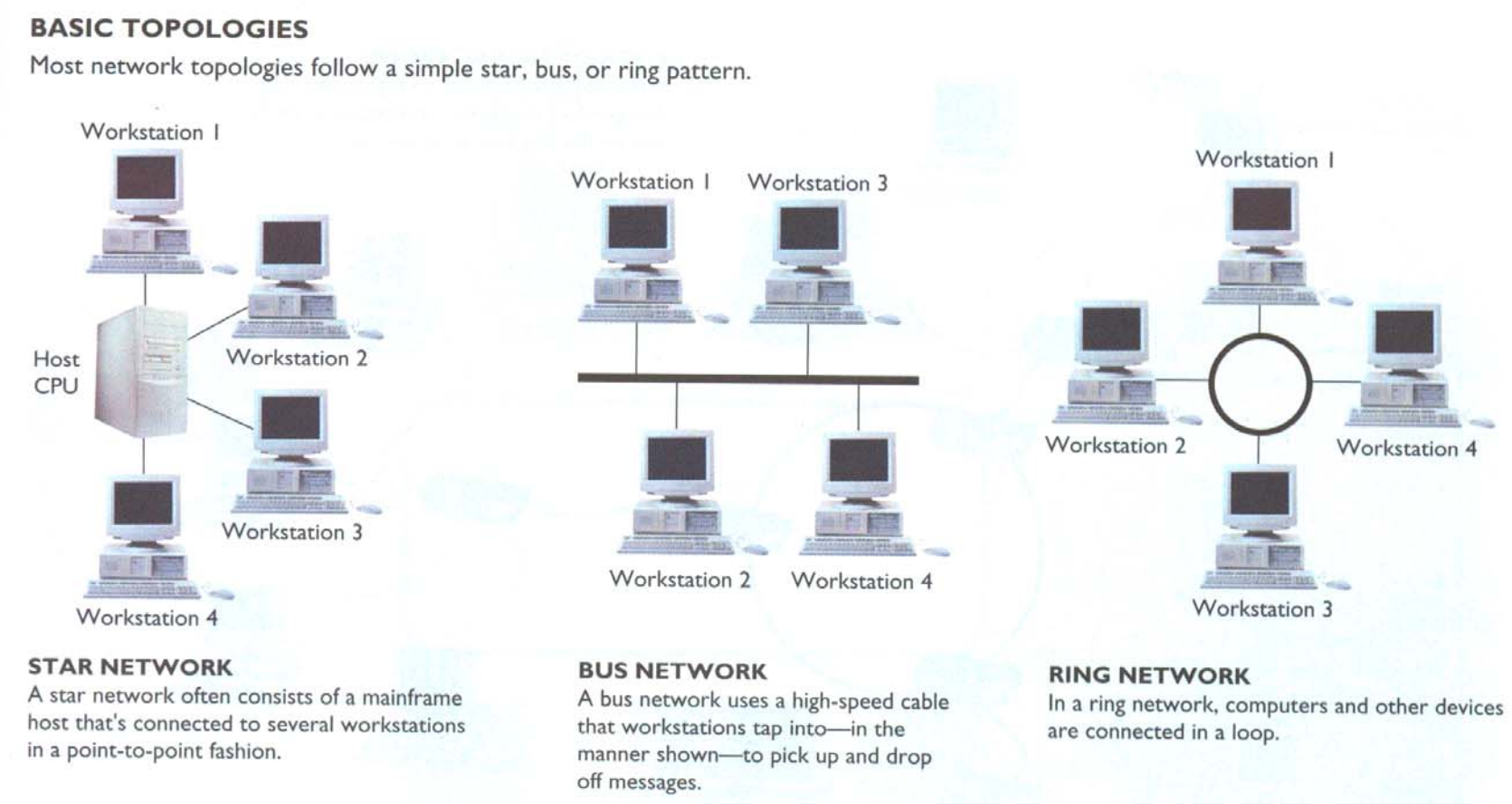



Network Management

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

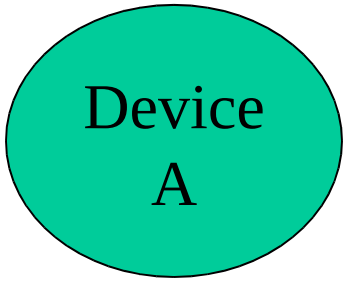


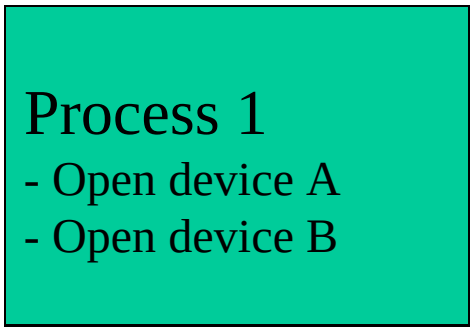
- Topology
- Communication Rules
- Signals & Mediums
- Encryption …

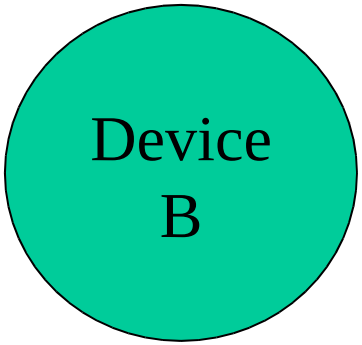


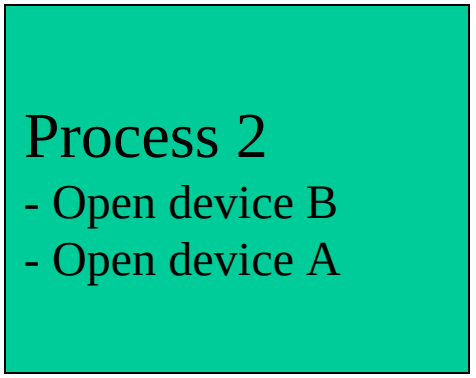

Deadlocking

- 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.