Scuola Superiore Sant'Anna

Real-Time System Laboratory

Operating Systems

Introduction

Giuseppe Lipari

Introduction

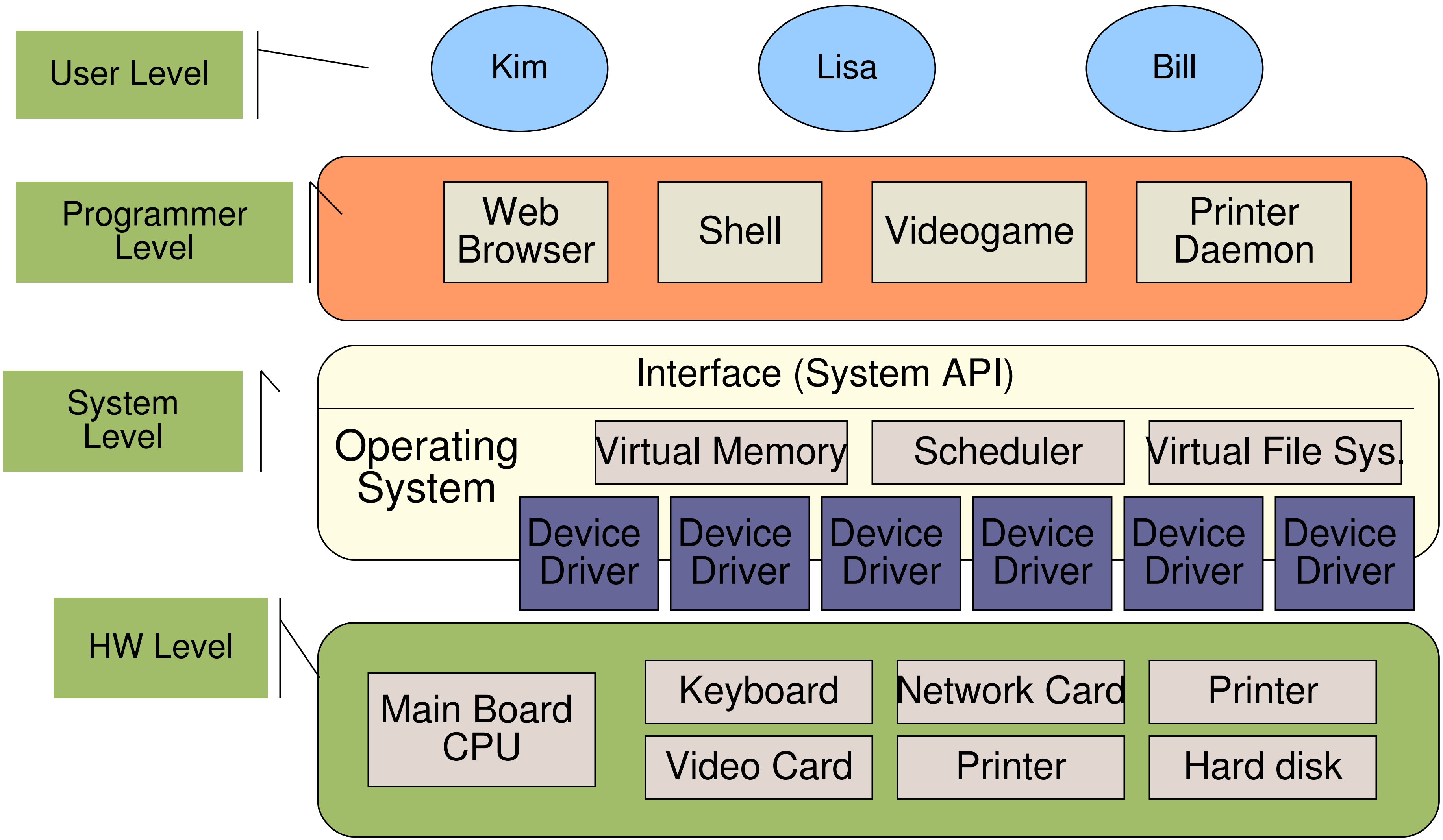
Fundamentals

- Algorithm:
  - It is the logical procedure to solve a certain problem
  - Informally specified as a sequence of elementary *steps* that an "execution machine" must follow to solve the problem
  - not necessarily expressed in a formal programming language!
- Program:
  - It is the implementation of an algorithm in a programming language
  - Can be executed several times with different inputs
- Process:
  - An instance of a program that, given a set of inputs values, produces a set of outputs

Operating System

- An operating system is a program that
  - Provides an "*abstraction*" of the physical machine through a simple interface
  - Each part of the interface is a "*service*"
- An OS is also a resource manager
  - With the term "resource" we denote all physical entities of a computing machine
  - The OS provides access to the physical resources
  - The OS provides *abstract resources* (for example, a file, a virtual page in memory, etc.)

Levels of abstraction



Abstraction mechanisms

- Why abstraction?
  - Programming the HW directly has several drawbacks
    - It is difficult and error-prone
    - It is not portable
  - Suppose you want to write a program that reads a text file from disk and outputs it on the screen
    - Without a proper interface it is virtually impossible!

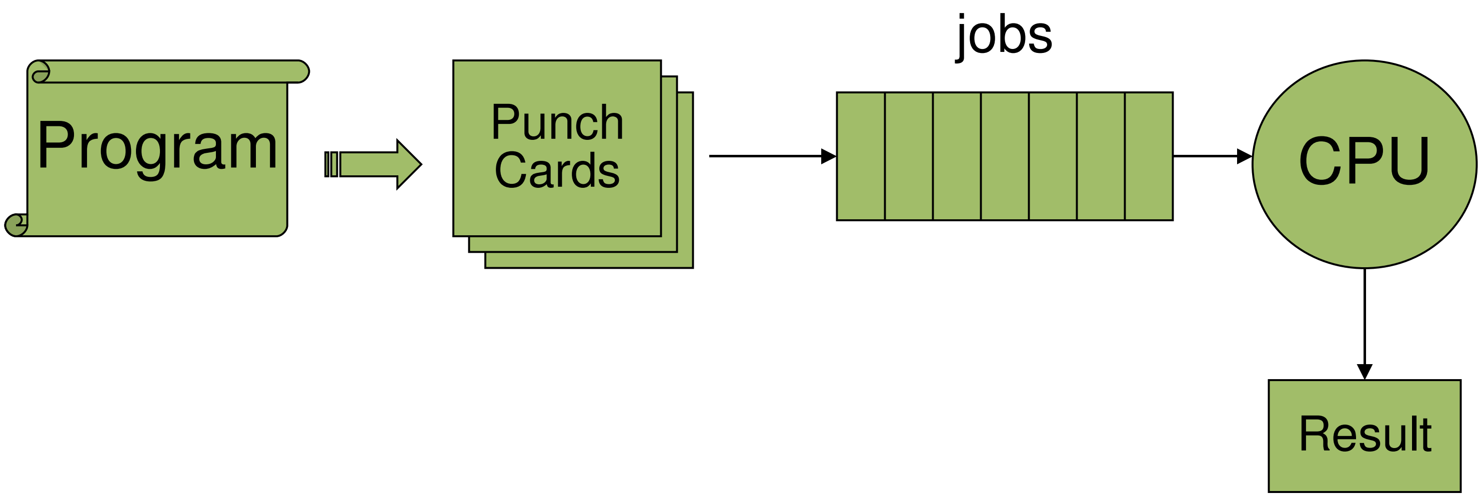
Abstraction Mechanisms

- Application programming interface (API)
  - Provides a convenient and uniform way to access to one service so that
    - HW details are hidden to the high level programmer
    - Applications do not depend on the specific HW
    - The programmer can concentrate on higher level tasks
  - Example
    - For reading a file, linux and many other unix OS provide the **open(), read()** system calls that, given a “file name” allow to load the data from an external support

Historical Perspective

- In the beginning was the batch processor
  - Huge machines, not very powerful
  - Used mainly for scientific computation and military applications
  - Program were executed one at time
    - They were called jobs
  - Program were simple sequential computations
    - Read the input
    - Compute
    - Produce output
  - Non-interactive!

Batch processor



- Batch = non-interactive
- The program could not be interrupted or suspended (non-preemptive)
- Scheduling:
  - Priority based (e.g. first the military...)
  - FIFO
  - Shortest job first (SJF)

Drawbacks

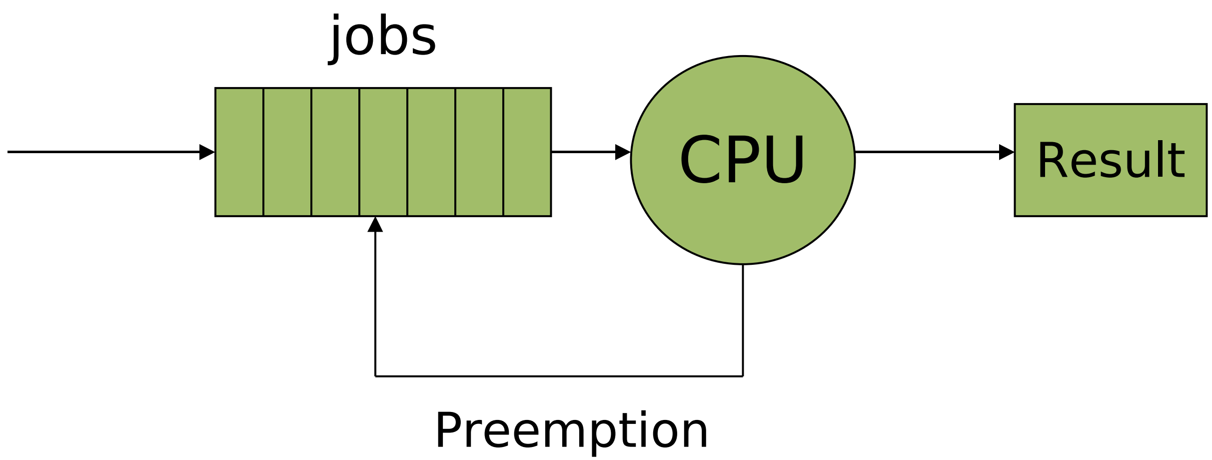
- CPU was inactive for long intervals of time
  - While reading the punch cards, the CPU had to wait
  - The punch card reader was very slow
- Solution: spooling
  - Use a magnetic disk (a faster I/O device)
  - Job were grouped into “job pools”
  - While executing one job of a pool, read the next one into memory
  - When a job finishes, load the next one from the disk
  - Spool = symultaneous peripheral operation on-line

Interactivity

- The need for interaction
  - For reading input from the keyboard *during* the computation
  - For showing intermediate results
  - For saving intermediate result on magnetic support
- Input/output
  - It can be done with a technique called *polling*
    - Wait until the device is ready and get/put the data
    - Handshaking
  - Again, the CPU was inactive during I/O operations

Multi-programming

- The natural evolution was “concurrency”
  - IDEA: while a job is reading/writing from/to a I/O device, schedule another job to execute (preemption)



Multi-programming

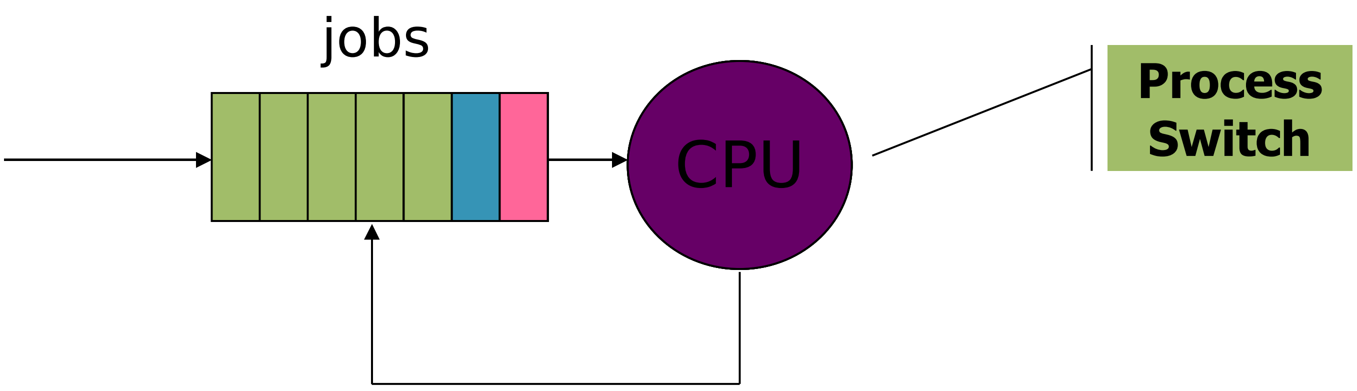
- Multi-programming is very common in real-life
  - Consider a lawyer that has many clients
    - FIFO policy: serving one client at time, from the beginning until the court sentence
    - In italy, a sentence can be given after more than 10 years. Imagine a poor lawyer trying to survive with on client only for ten years!
    - In reality, the lawyer adopts a TIME SHARING policy!
  - All of us adopts a time-sharing policy when doing many jobs at the same time!

The role of the Operating System

- Structure of a multi-programmed system
  - Who decides when a job is suspended?
  - Who decided who is to be executed next?
    - In the first computers, these tasks were carried out by the application itself
    - Each job could suspend itself and pass the “turn” to the next job (*co-routines*)
    - However, this is not very general or portable!
  - Today, the OS provide the multiprogramming services
    - The *scheduler module* chooses which job executes next depending on the status of the system

Time sharing systems

- In time sharing systems
  - The time line is divided into “slots”, or “rounds”, each one of maximum lenght equal to a fixed time quantum
  - If the executing job does not block on a I/O operation befone the end of the quantum, it is suspended to be executed later



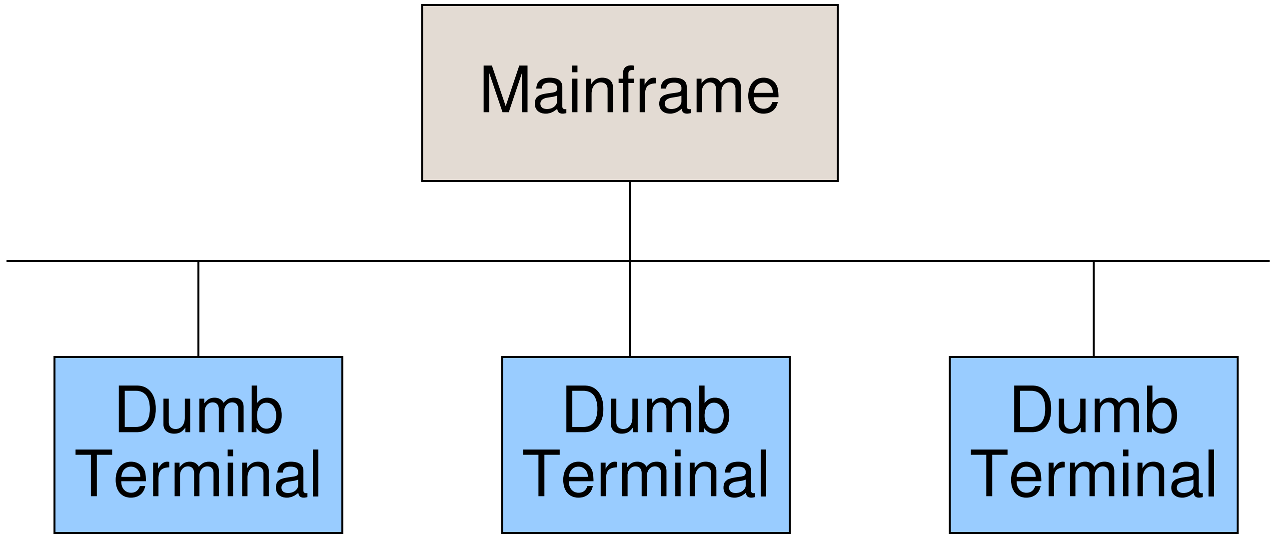
Time sharing systems

- In time sharing systems
  - Each process executes approximately as it were alone on a slower processor
  - The OS (thanks to the scheduler) “virtualizes” the processor
    - One single processor is seen as many (slower) parallel processors (one for each process)
  - We will see that an OS can virtualize many HW resources
    - Memory, disk, network, etc
- Time sharing systems are not predicatable
  - The amount of execution time received by one process depends on the number of processes in the system
  - If we want predictable behavior, we must use a RTOS

Multi-user systems

- The first computers were very powerful and very expensive
  - An university could afford only one *mainframe*, but many people needed to access the same computer
  - Therefore, the mainframe would give simultanous access to many users at the same time
  - This is an obvious extension of the multi-process system
    - One or more processes for each user

Multi-user systems



- The terminals had no computing power
  - A keyboard + a monitor + a serial line
  - Every computation was carried out in the mainframe
  - It is like aving one computer with many keyboards and videos

Multi-user system

- Another dimension was necessary
  - The concept of user and account was born
  - The first privacy concerns were raised
    - Access rules
    - Passwords
    - Criptography was applied for the first time in a non-military environment!
  - This makes the system more complex!

Distributed systems

- Finally, distribution was introduced
  - Thanks to the DARPA, the TCP/IP protocol was developed and internet was born
    - The major universities in the USA connected their mainframes
    - Mail, telnet, ftp, etc
    - The natural evolution was internet and the world wide web
  - All of this was possible thanks to
    - The freedom of circulation of ideas
    - The “liberal” environment in universities
    - The need for communication and sharing information

Distributed systems

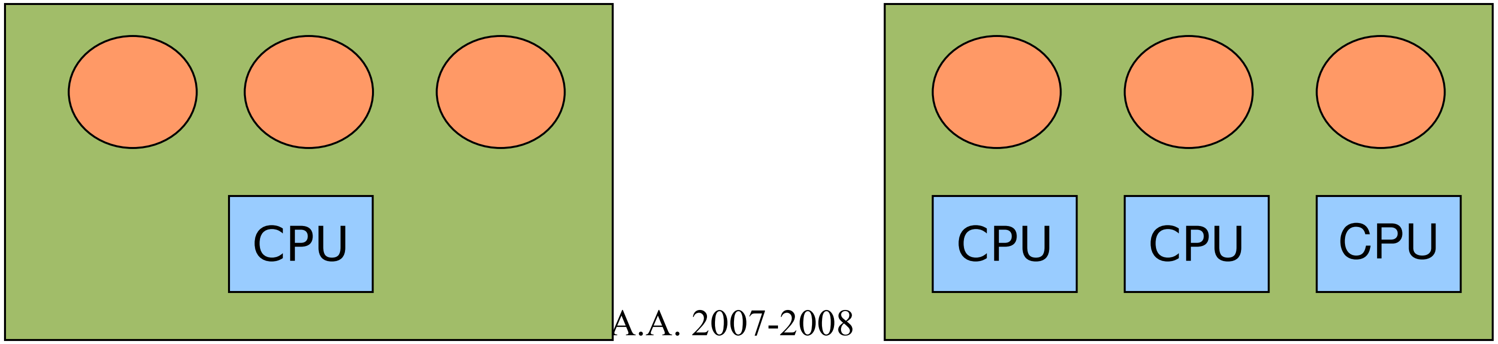
- More flexibility
  - Client/server architectures
    - One server provides “services” to remote clients
    - Example: web, ftp, databases, etc
  - It is possible to “distribute” an application
    - Different “parts” execute on different computers and then communicate each other to exchange information and synchronise
    - Massively parallel programs can be easily implemented
  - Migration
    - Processes can “move” from one computer to another to carry out a certain service
    - Examples: agents, videogames, applets, etc

Classification of Operating Systems

- The OS provides an abstraction of a physical machine
  - To allow portability
  - To make programmer's life easier
- The level of abstraction depends on the application context
  - It means that the kind of services an OS provides depend on which kind of services the application requires
    - General purpouses OS should provide a wide range of services to satisfy as many users as possible
    - Specialised OS provide only a group of specialised services
  - OS can be classified depending on the application context
    - General purpouse (windows, linux, etc), servers, micro-kernel, embedded OS, real-time OS

Services

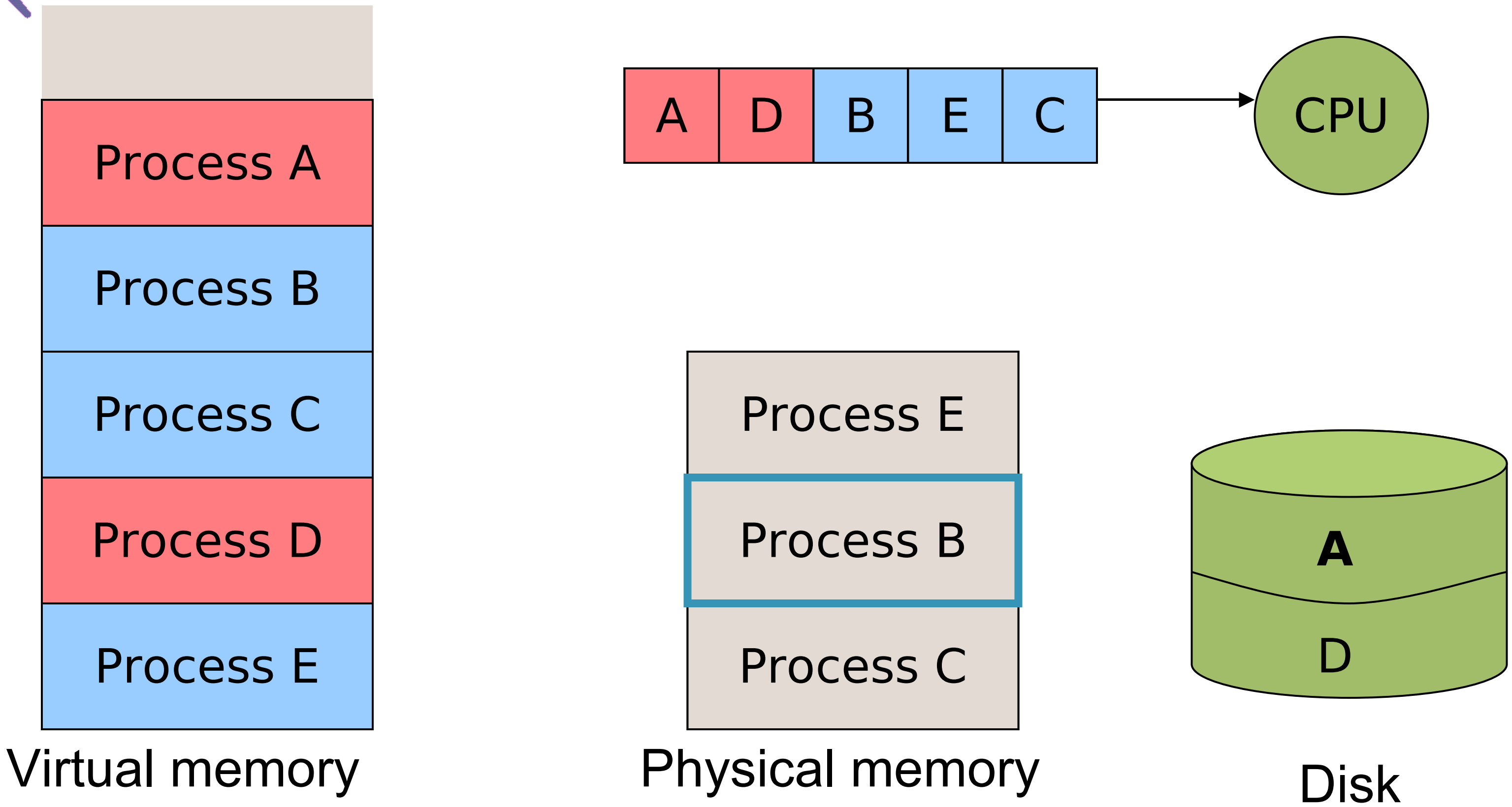
- Virtual processor
  - An OS provides “concurrency” between processes
    - Many processes are executed at the same time in the same system
    - Each process executes for a fraction of the processor bandwidth (as it were on a dedicated slower processor)
  - Provided by the scheduling sub-system
  - Provided by almost all OS, from nano-kernels to general-purpouse systems



Services

- Virtual memory
  - Physical memory is limited;
  - In old systems, the number of concurrent processes was limited by the amount of physical memory
  - IDEA: extend the physical memory by using a “fast” mass storage system (disk)
    - Some of the processes stay in memory, some are temporarily saved on the disk
    - When a process must be executed, if it is on the disk it is first loaded in memory and then executed
    - This technique is called “swapping”

Virtual memory and physical memory



- Virtual memory is very large (virtually infinite!)
- The program functionality does not depend on the size of the memory
- The program performance could be reduced by the swapping mechanism

Virtual Memory

- Advantages
  - Virtual infinite memory
  - The program is not limited by the size of the physical memory
- Disadvantages
  - If we have too many programs, we spend most of the time swapping back and forth
  - Performance degradation!
  - Not suitable for real-time systems
    - It is not possible to guarantee a short response time because it depends on the program location

Virtual File System

- Basic concepts
  - File: sequence of data bytes
    - It can be on a mass storage (hard disk, cd-rom, etc.)
    - It can be on special virtual devices (i.e. RAM disks)
    - It can be on a remote system!
  - Directory: list of files
    - Usually organised in a tree
    - Represents how files are organised on the mass storage system
- Virtualisation
  - In most OS, external serial devices (like the console or the video terminal) can be seen as files (i.e. stdin, stout , stderr)

Virtual file system

- A good virtual file system provides additional features:
  - Buffering & caching
    - For optimising I/O from block devices
  - Transactions
    - For example the Reiser FS
  - Fault tolerance capabilities
    - For example, the RAID system
- Virtual file system is not provided by all OS categories
  - Micro and nano kernels do not even provide a file system!

Privacy and access rules

- When many users are supported
  - We must avoid that non-authorised users access restricted information
  - Usually, there are two or more “classes” of users
    - Supervisors
    - Normal users
  - Each resource in the system can be customised with proper “access rules” that prevent access from non-authorised users
    - For example, the password file should be visible only to the system supervisor