Operating Systems Overview

operating system

- no clear traditional definition
  – each definition cover a distinct aspect
- an interface between applications and hardware
  – true, this was the first reason for having an OS
- a set of programs that provides basic functionalities for managing system resources
  – true, but the OS is not only a “library” of functionalities
- a program that “controls” the execution of application programs
  – e.g. it decides which program is running and when

not on
the book

my definition

- a software that does not **depend** on any other software in the computer
  - do you know what “depend” means?
  - in a certain sense it only depends on hardware

in current PC and servers it also is...

- a software that...
  - considers “activities” to be carried on and...
  - assigns “resources” to them
    - “activities” usually means “processes”
- the latter is the most interesting definition

Operating System as a Resource Manager

- **resource**: anything needed for program execution, e.g.
  - cpu time
  - I/O devices
  - memory
  - executable code
  - etc.
- an os manages...
  - resources
    - comprising “internal” data structures
  - processes

not on
the book

my definition

if the hardware is so small that cannot carry on many tasks (e.g. a mobile phone)...
- less and less frequently

- ...it makes simple for applicative software to interface with hardware

- hw interface on bigger systems is not the primary concern
  - handled with “drivers”

kernel

- its the OS as defined above
- it is always in main memory
- contains most frequently used functionality
- also called “nucleus”

not on
the book

the kernel

- it is a very interesting piece of software!
- it can be studied from several point of views
  - structural
    - which concepts it realizes?
    - which data structures it adopts?
    - what modules it contains?
    - what are the interfaces among modules?
  - behavioral
    - which strategies adopt to optimize usage resources?
    - which algorithms it implements to maintain its data structures?
  - synchronization
    - oops! many things happens in the kernel at the same time. Consistency is hard to maintain. Large number of techniques uses. Completeley indpendent by OS study, but largely developed by the same community.

not on
the book

kernel

- didactic approach vs. reality
  - structural
    - what is the easiest way to realize a “decent” kernel?
    - what is there in a real kernel?
  - behavioral
    - what are the easiest strategies?
    - what are the ones that are adopted in practice?
  - synchronization
    - too complicated... there is a specific course for this ;)

not on
the book

system software

- it is the software that is usually installed when the OS is installed on a computer system
  - e.g.
    basic C runtime libraries,
    sw to access the hard disk directory,
    graphical user interface,
    etc.

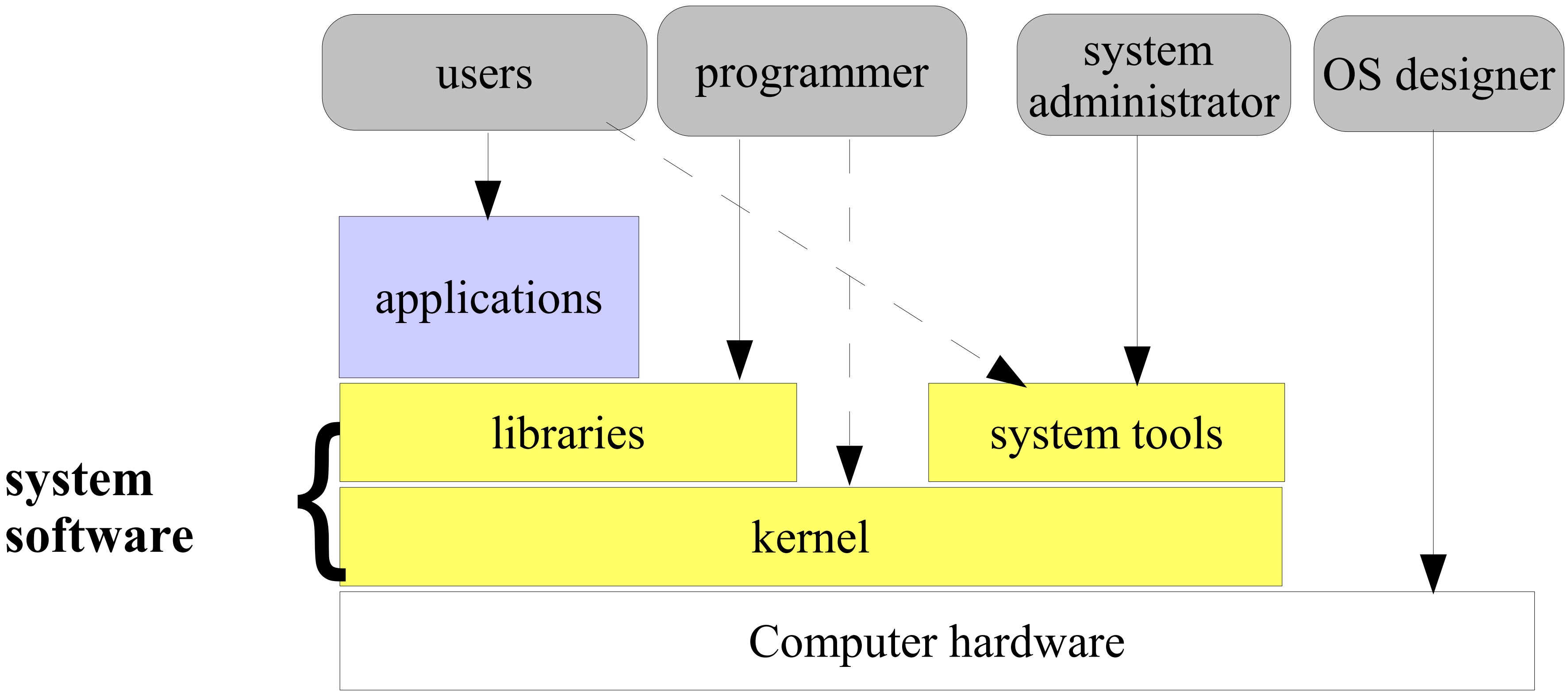
not on the book

overloaded language

- actually "operating system" is an overloaded word
- it means
  - the **kernel** or
  - the kernel **plus** the other system software
- meaning should be clear from the context

not on
the book

computer system layers



OS objectives

- convenience
  - it should make the computer more convenient to use (for programmers and users)
- efficiency
  - it should allow computer system resources to be used in an efficient manner
- ability to evolve
  - it should permit effective development, testing, and installation of new software (debug, isolation)

services provided by the os

- services for users
  - **program execution, usually many at the same time**
  - error detection and response
  - support for program development
  - security (login, user confinement, etc.)
  - accounting
- services for programs (and programmers)
  - resource management
    - es. memory and cpu time
  - access to I/O devices
    - es. files

efficiency: i/o vs. cpu

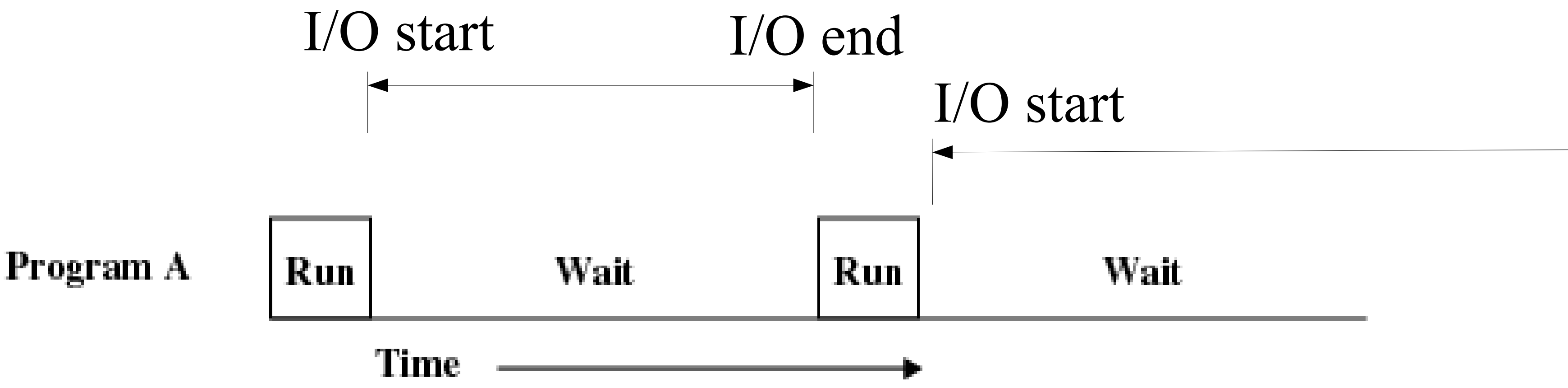
Read one record from file	15 µs
Execute 100 instructions	1 µs
Write one record to file	15 µs
TOTAL	31 µs

$$\text{Percent CPU Utilization} = \frac{1}{31} = 0.032 = 3.2\%$$

Figure 2.4 System Utilization Example

uniprogramming

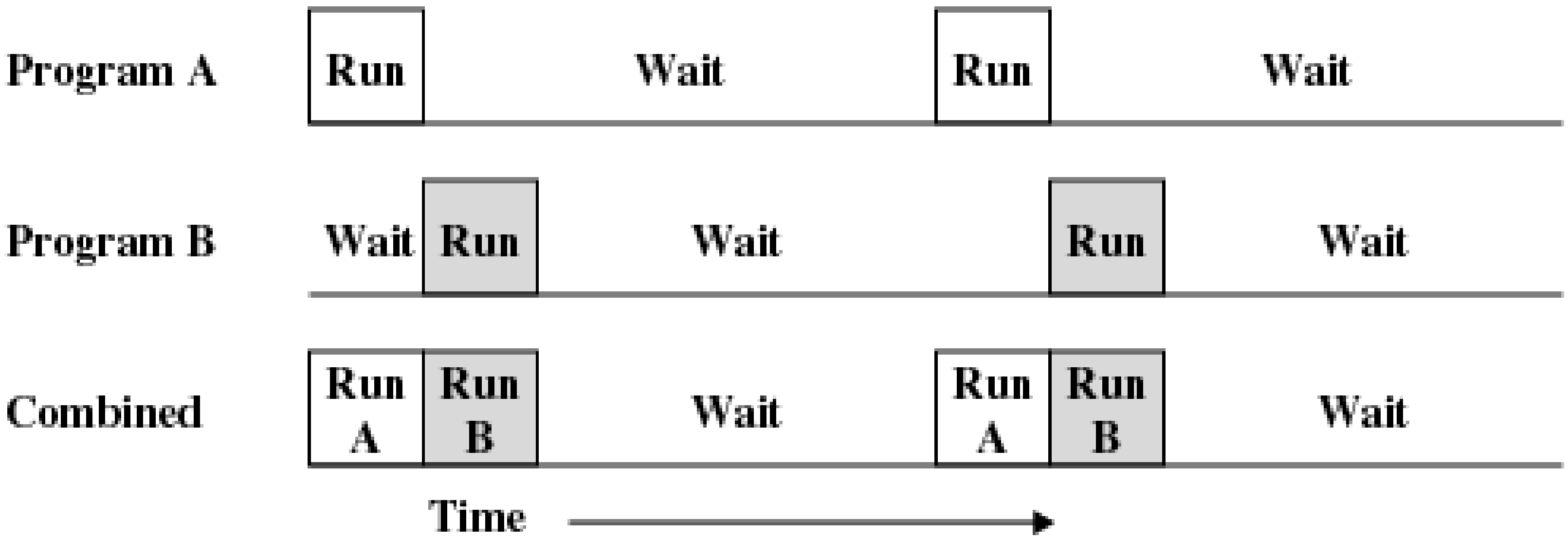
- simple approach: execution must wait for I/O to complete before preceding



(a) Uniprogramming

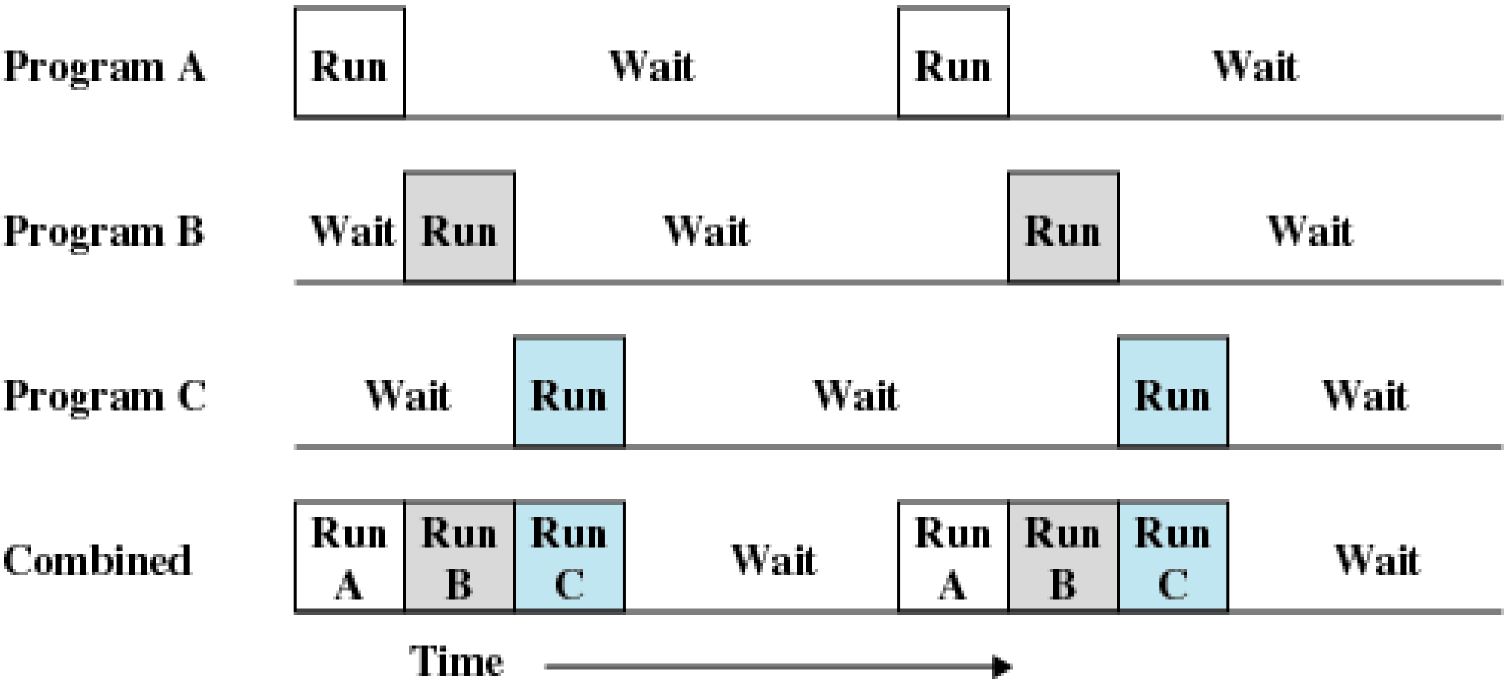
multiprogramming

- when one job needs to wait for I/O, the processor can switch to the other job



(b) Multiprogramming with two programs

multiprogramming



(c) Multiprogramming with three programs

not on
the book

the user's point of view

- from the point of view of the user is a way to keep many applications active at the same time
  - e.g. I got frustrated by “uniprogrammed” smartphones need to close an app before open another one loosing current work
- PC started to be multiprogrammed in 90s (about)
  - windows 3.1, Mac classic
  - GUIs greatly increase the demand of the user
- kernels that support multiprogramming are much more complex
  - but, it is a “must have”
  - … and it is one of the main topics of this course

not on
the book

I/O bound vs. CPU bound processes

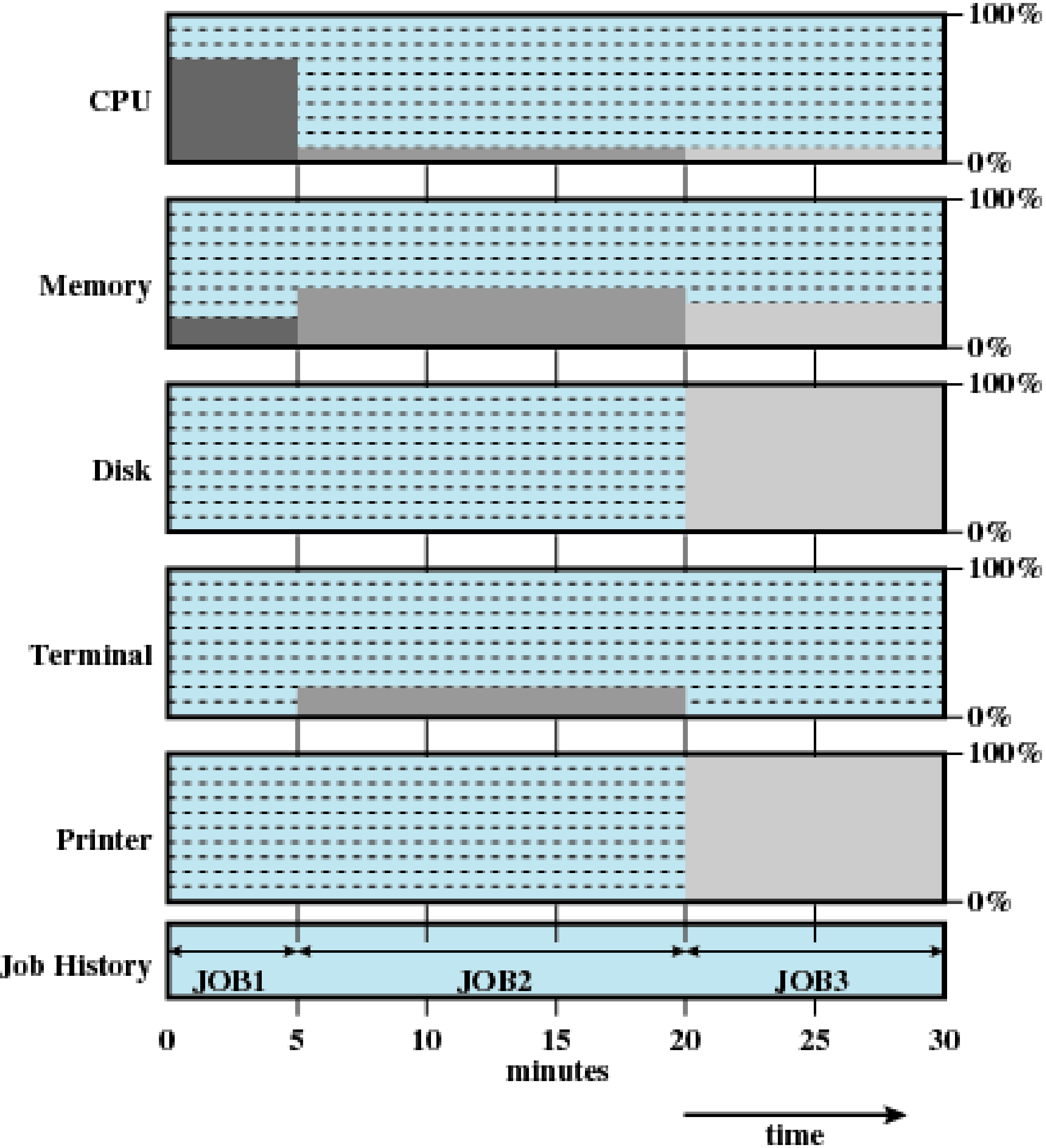
- I/O bound
  - mostly waiting for some data to arrive
  - use cpu for requesting more data from devices
  - e.g. DBMS, interactive applications
- cpu bound
  - mostly perform computation
  - rarely performs i/o to get data to compute
  - e.g. multimedia coding/encoding, complex graphic rendering, etc.

I/O bound vs. CPU bound processes

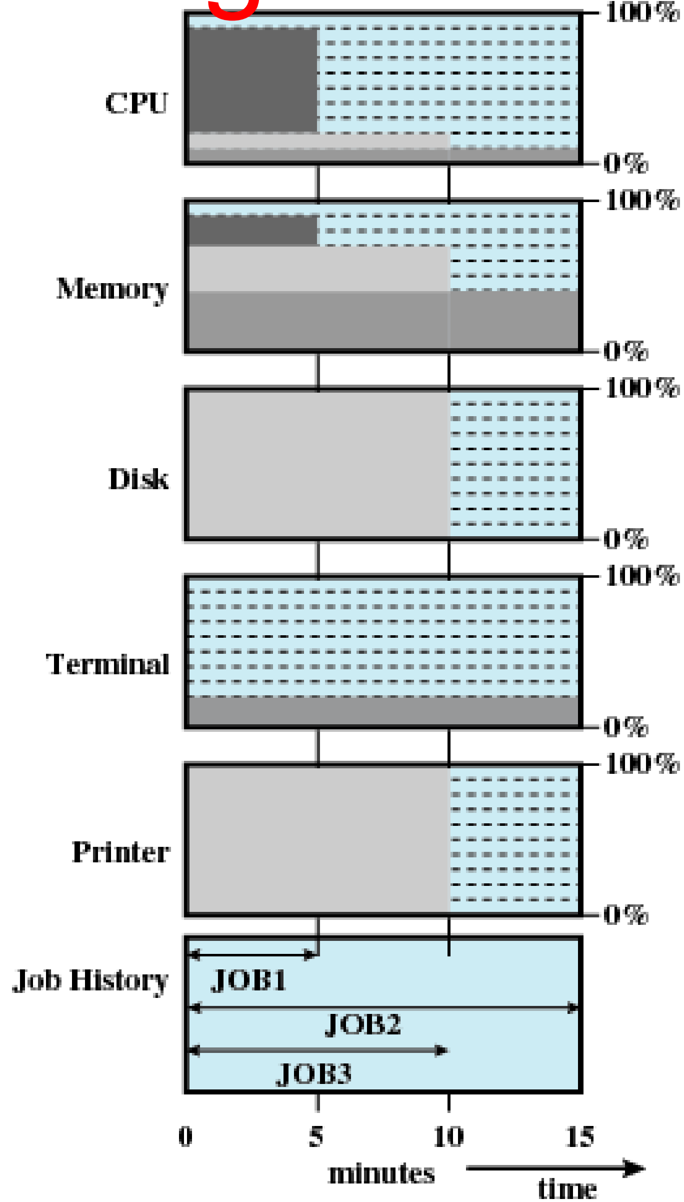
- a more complex example
  - sharable resources: cpu time and memory
  - non-sharable resources: disk, terminal, printer

	JOB1	JOB2	JOB3
Type of job	Heavy compute	Heavy I/O	Heavy I/O
Duration	5 min	15 min	10 min
Memory required	50 M	100 M	75 M
Need disk?	No	No	Yes
Need terminal?	No	Yes	No
Need printer?	No	No	Yes
	CPU bound	I/O bound	I/O bound

Utilization Histograms



(a) Uniprogramming



(b) Multiprogramming

Major Achievements of Modern OSes

- Processes
- Memory Management
- Information protection and security
- Scheduling and resource management
- System structure

not on
the book

processes

- a program in execution (running) on a computer
- a unit of activity characterized by
  – an associated set of system resources
    - memory regions
    - open files
    - etc.
  – at least one execution **thread** with its current state of CPU

threads

- the entity that can be assigned to, and executed on, a processor
  - it is meaningful only within a process
  - described by
    - the value of the program counter
    - the value of the CPU regisers
- in modern operating systems a process may contains one or more thread
- we always assume it contains one thread

security: user/system mode

- processes execute in **user mode**
  - certain *privileged* machine instructions may not be executed
  - only a restricted part of main memory can be accessed (**user space**)
- kernel executes in **system mode**
  - a.k.a. **kernel mode** or **supervisor mode**
  - privileged instructions can be executed
  - protected areas of memory may be accessed (**kernel space**)

many processes, one CPU

- only one process can be executed by one CPU
- the other processes are somehow “forzen”
- what is needed to resume it is saved somewere
  - execution context

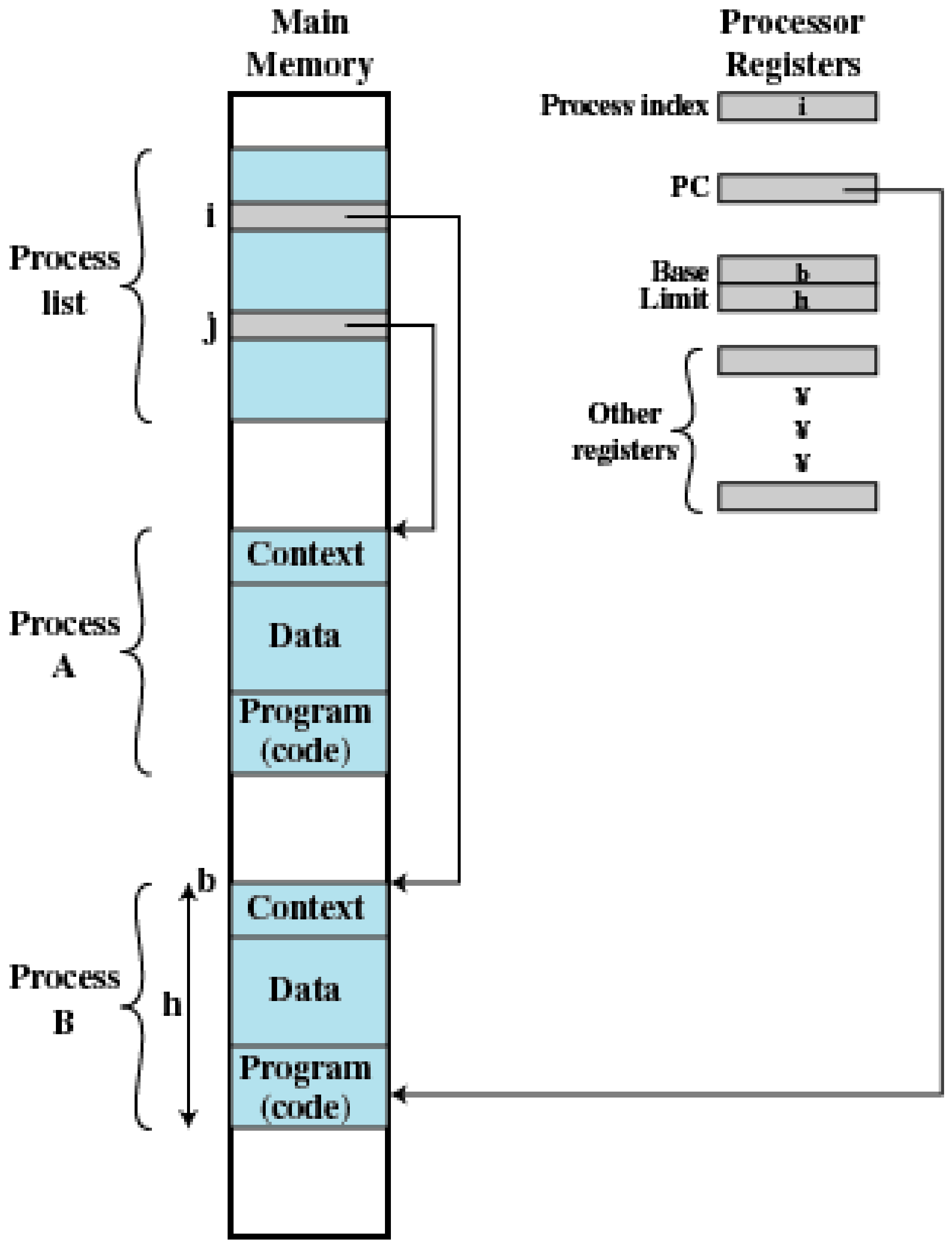
time sharing

- CPU time is shared among multiple users or processes
- illusion of more CPUs

execution context

- cpu registers
- priority of the process
- is the process waiting for I/O? on which device?
- etc.
- etc.
- etc.
- etc.
- ...

process representation



Memory Management

- Process isolation
- Automatic allocation and management
- Support of modular programming
- Protection and access control
- Long-term storage

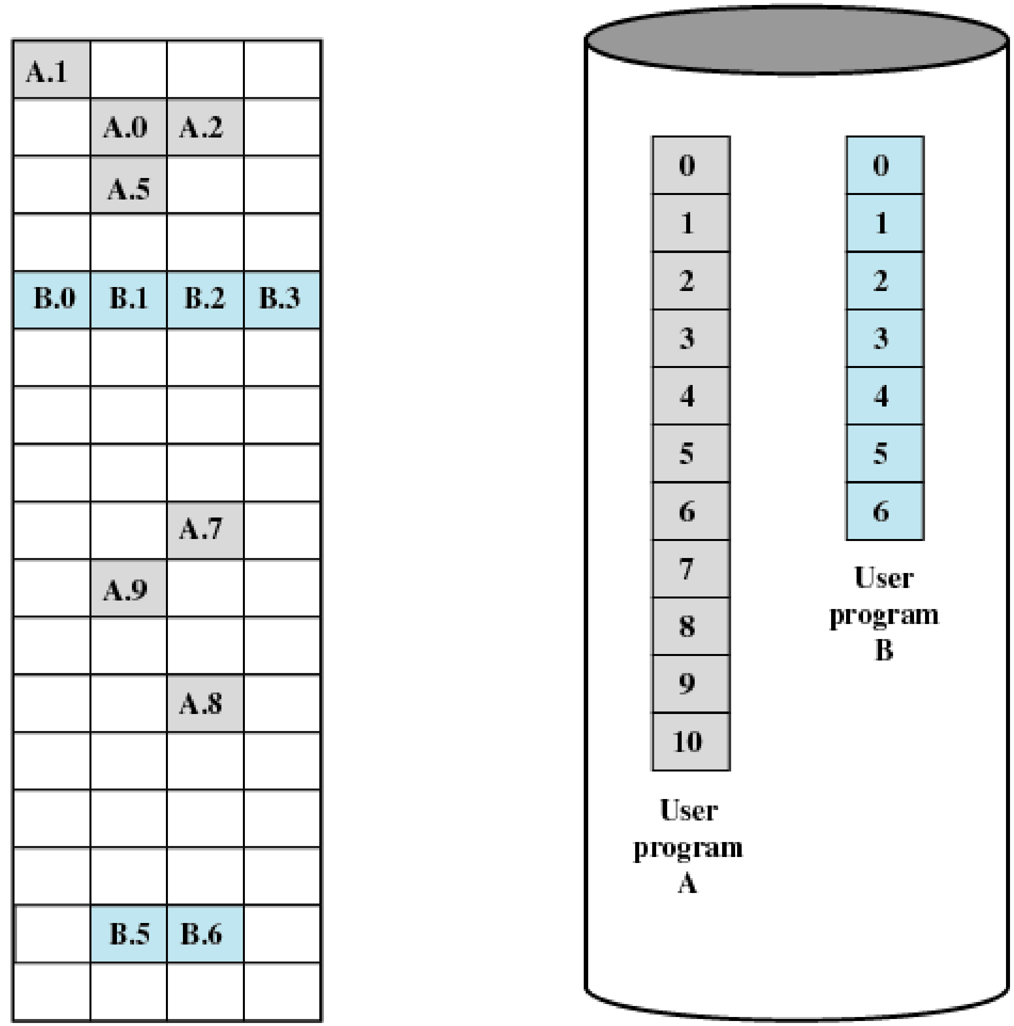
Virtual Memory

- Allows programmers to address memory from a logical point of view
- Virtual memory can be much larger than Real Memory
  - processes see a large virtual address space
- Real Memory is used only for (part of the) processes that are really need it

Paging

- It allows process memory to be divided in a number of fixed-size blocks, called pages
- Virtual address is made of page number and an offset within the page
- Each page may be located anywhere in physical memory
- Real address or physical address in main memory are managed only by the kernel

Virtual Memory and Main Memory



Main Memory

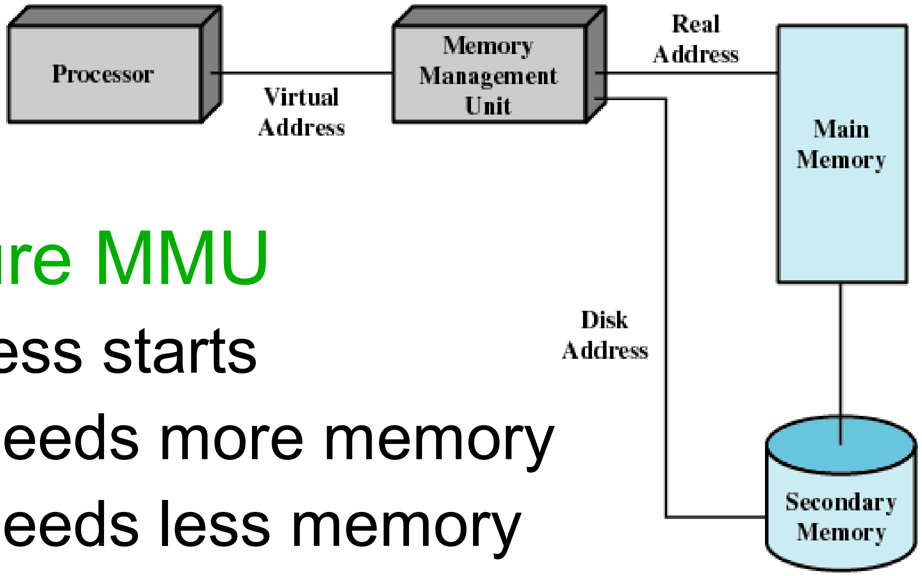
Main memory consists of a number of fixed-length frames, each equal to the size of a page. For a program to execute, some or all of its pages must be in main memory.

Disk

Secondary memory (disk) can hold many fixed-length pages. A user program consists of some number of pages. Pages for all programs plus the operating system are on disk, as are files.

Virtual Memory Addressing

- virtual addresses management
  - the Memory Management Unit of the CPU translates addresses from virtual to real
    - each time an machine instruction refers to memory
    - this is very very frequent!



  - the kernel configure MMU
    - when a new process starts
    - when a process needs more memory
    - when a process needs less memory

Scheduling and Resource Management

- Fairness
  - Give equal and fair access to resources
- Differential responsiveness
  - Discriminate among different classes of jobs
- Efficiency
  - Maximize throughput, minimize response time, and accommodate as many users as possible

Scheduling Elements

- queues
  - at least one for each resource
- CPU
  - short term
    - contains processes in main memory and ready to run
    - short term scheduler / dispatcher
      - simple approach: round robin (circular queue)
  - long term
    - new jobs waiting for the processor
    - long term scheduler
- I/O
  - at least queue for each device
  - interrupts

Modern Operating Systems

- microkernel architecture
  - Assigns only a few essential functions to the kernel
    - Address spaces
    - Interprocess communication (IPC)
    - Basic threading/scheduling
  - everything else is implemented as a process
  - this is more a trend than reality

Modern Operating Systems

- Symmetric multiprocessing (SMP)
  - There are multiple processors or cores
  - These cores share same main memory and I/O facilities
  - All cores can perform the same functions
  - Each core can execute one thread at time

Modern Operating Systems

- Multithreading
  - Each process has one or more execution threads that
    - can run concurrently, possibly on distinct cores
    - share and can access all other resources of a process
  - Very important now that we have many processors
    - relevant mostly for CPU bound applications

Modern Operating Systems

- Virtualization
  – OS support emulation of several instances of "fully-fledged" machines
  – enabling technology for cloud computing
  – efficiency problems: virtualization overhead
  – can exploit hardware support

Modern Operating Systems

- Mobile
  - hardware runs on battery: power management
  - hardware is small: need for space/time efficiency
  - wirelessly connected most of the time
  - interactive: end-user friendliness
  - app marketplace
  - big security concerns