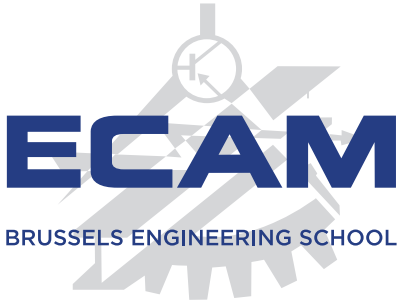


SS201μ Introduction to the Internal Design of Operating Systems

Session 1

Definition and Structure of Operating Systems

Sébastien Combéfis

Winter 2020

Objectives

- Define computing system and operating system (OS)

  What it is, what it does and the relation between both

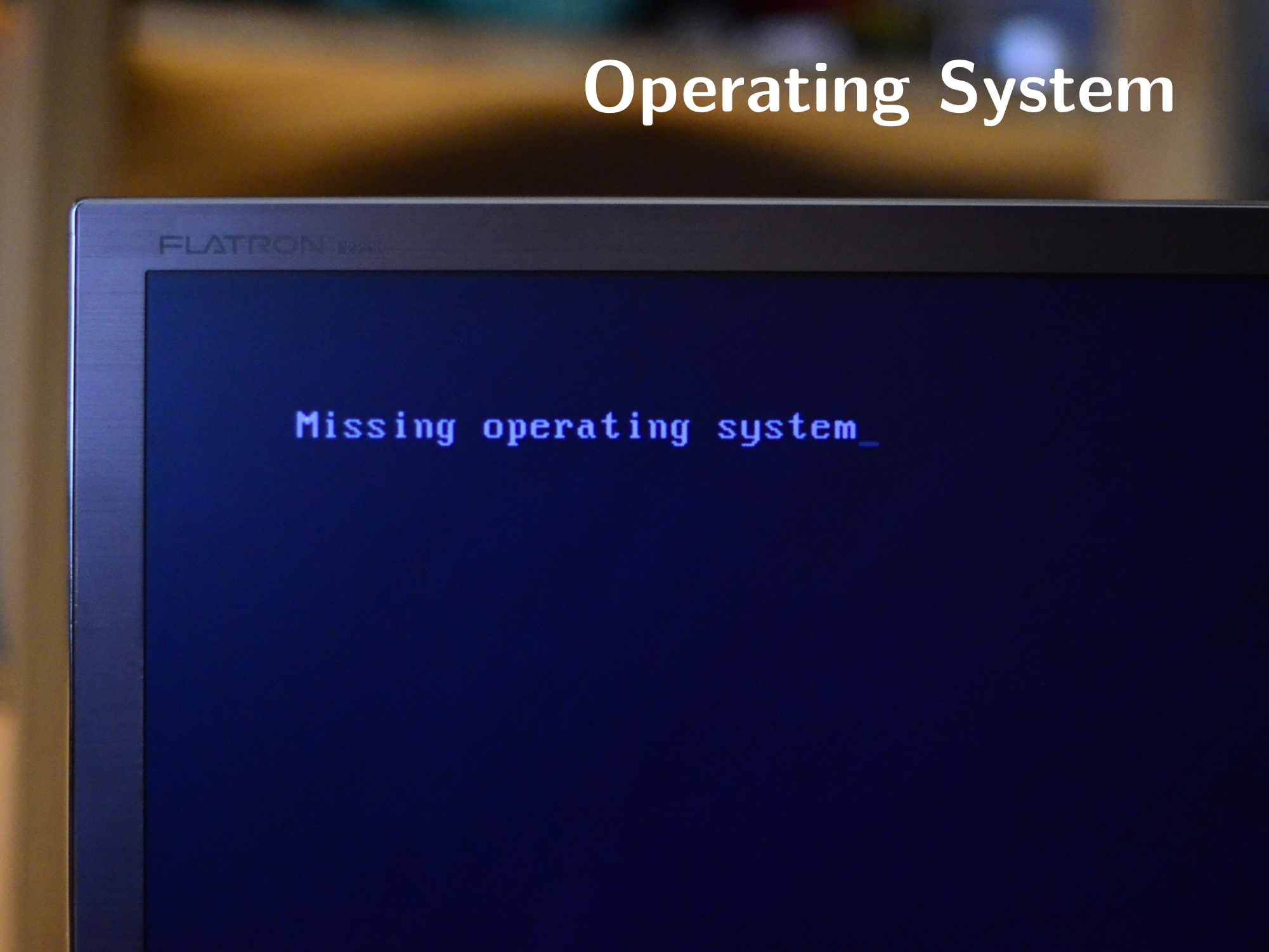
- Understand the services provided by an OS

  To the user and to the system through system calls

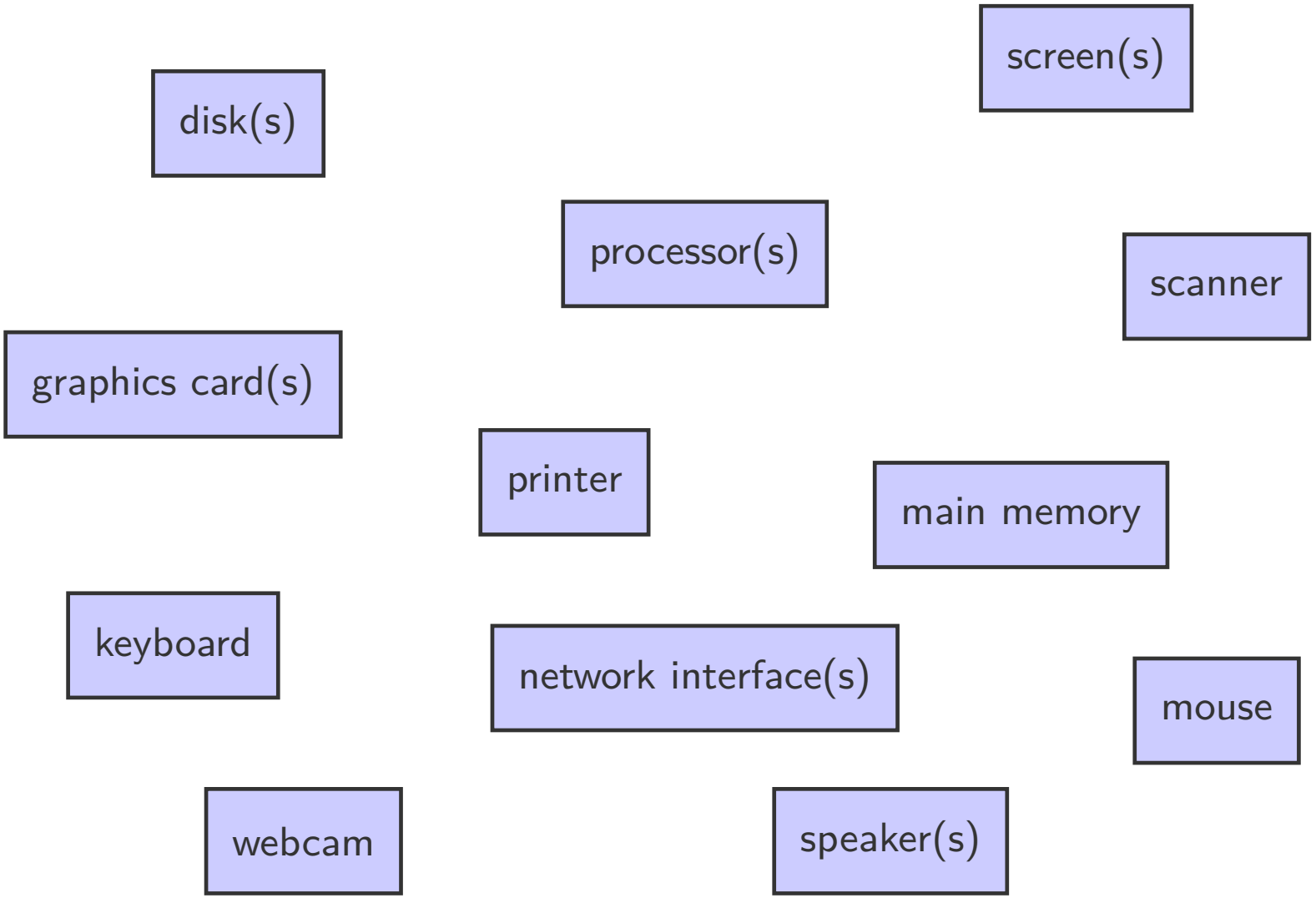
- Discover how the OS handles CPU, memory and input/output

  With software abstractions stored and managed by the OS

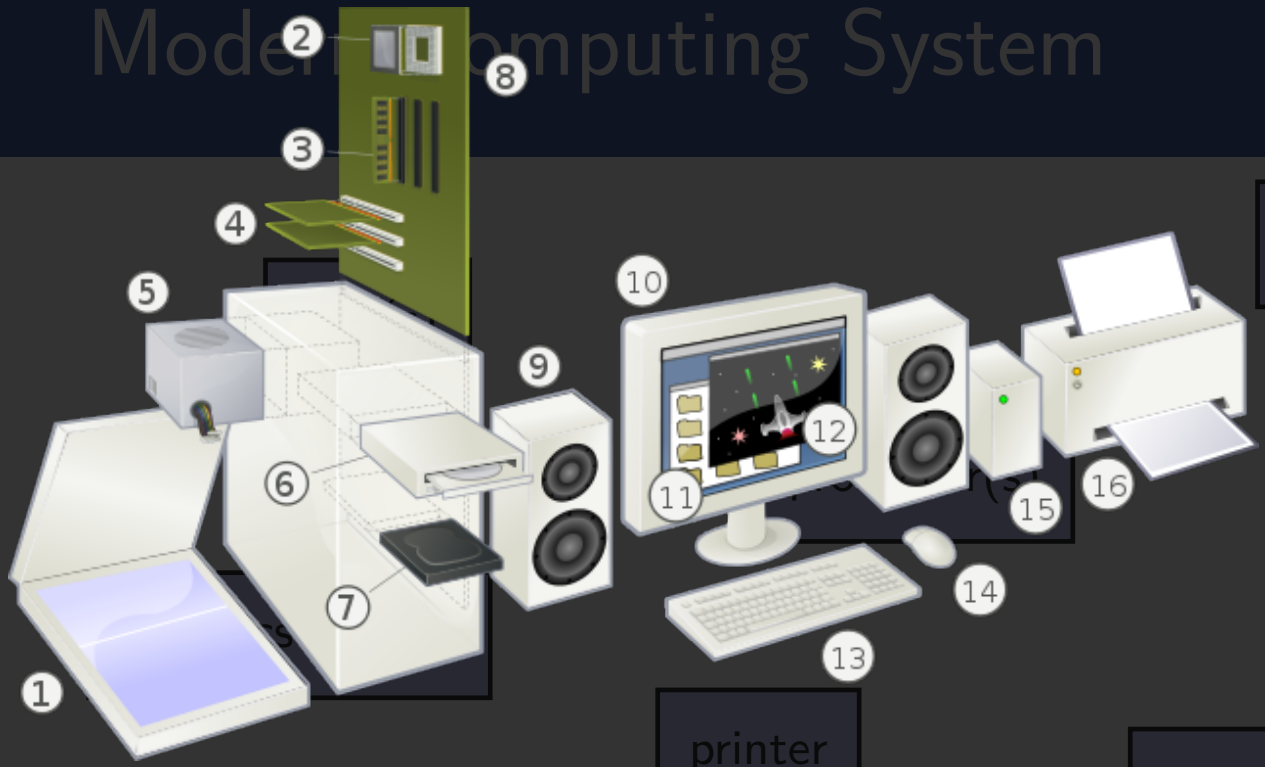
Operating System



Modern Computing System



Modern Computing System



sc

printer

main memory

vork interface(s)

Computing vs Operating System

- Computing system composed of three elements
  - Hardware
  - Software
  - Data
- The OS provides an environment to use these resources

  No useful functions in itself, used by other programs
- Modern OS controlled by interruptions

  OS code only executed following events reported by interruptions

Operating System Example

Operating System

- Intermediate software layer between user and hardware

  Provides user programs an interface to the hardware

- Three main objectives for an OS
  - **Practical**: easy to use by users (PC, mobile)
  - **Efficient**: optimise the use of hardware (mainframes)
  - **Scalable** : addition of new system functions

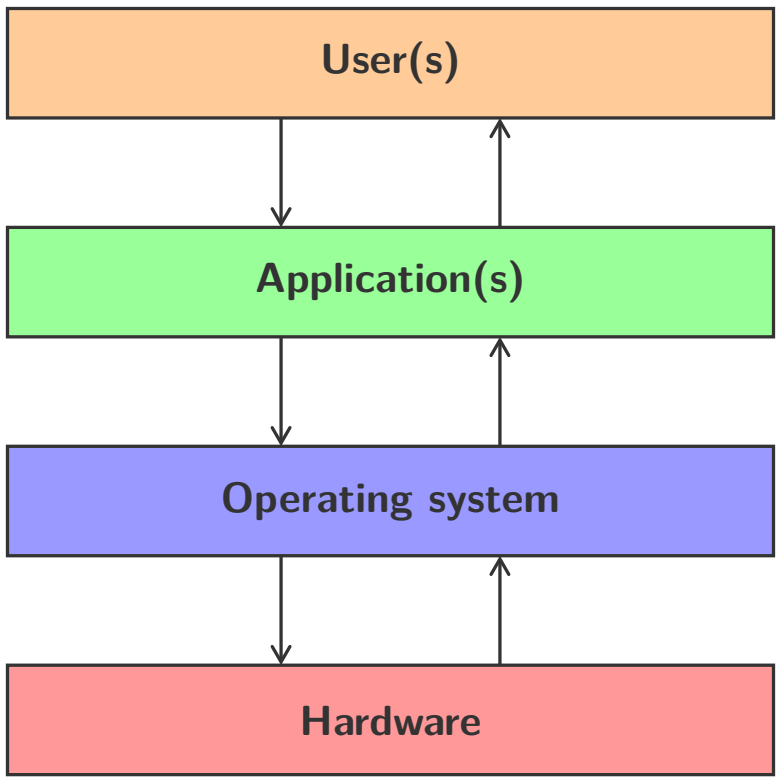
- Very large and very complex software system

  Created piece by piece with a clear delimitation

Computing System Structure (1)

- A computing system can be seen with four layers

  Each layer uses services from below to provide services to above



Computing System Structure (2)

- The hardware provides computing resources

  Processor (CPU), memory, input/output devices (I/O)

- The operating system coordinates the use of hardware

  Coordination between applications and users

- The applications solve problems from the users

  Word processing, compiler, video game, web browser, etc.

- The users uses the computing sustem

  People, machine, other computers, etc.

Definition

- There is no universally adopted definition

  Great diversity: toaster, car, space shuttle, house, game machine, industrial control system, etc.

- *"Everything in the box when you buy an OS"*

  Microsoft 1998 lawsuit with Internet Explorer

- *"Only program running on the computer all the time"*

  Kernel, system programs and application programs

Operating System Services

OS Abstraction

- Environment for applications to work

  Providing an access to the hardware for applications

- Several abstractions are proposed by the OS

  Transparent handling of the hardware

Hardware	**Abstraction**
CPU and memory	Processes and threads
Storage on disk	Files
Network interface	Sockets, RPC, RMI
Display	Drawing libraries and windows

OS Function

- The OS acts as a service provider

  File system, standard libraries, window system, etc.

- The OS works as a coordinator on three aspects
  - **Protection**: jobs must not interfere with each other
  - **Communication**: jobs must be able to interact tohether
  - **Resource management**: resource sharing must be facilitated

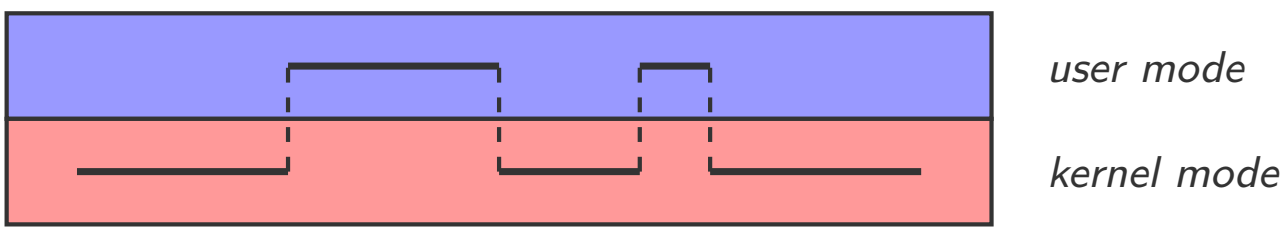
OS Operation

- The OS does nothing as long as there is nothing to do

  Processes to execute, I/O to handle, users to respond to

- Alternation between two modes of execution

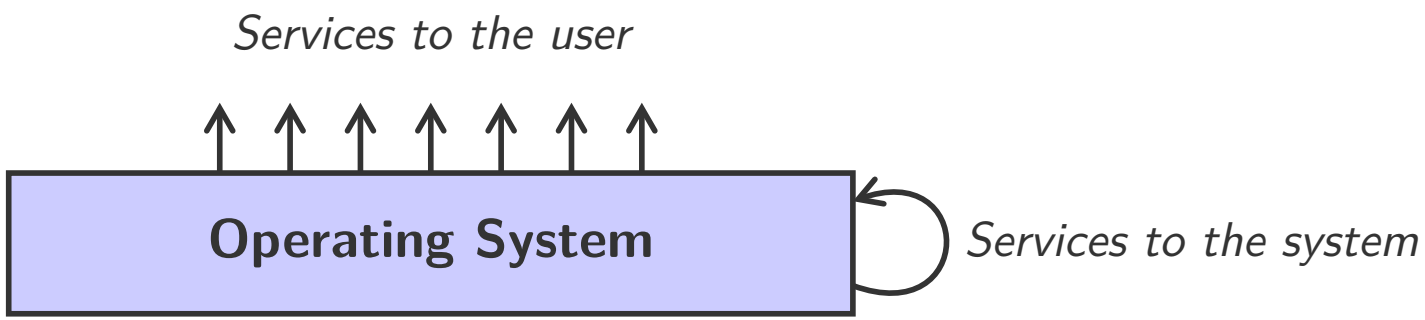
  Distinguishing the execution of the OS code from the user code



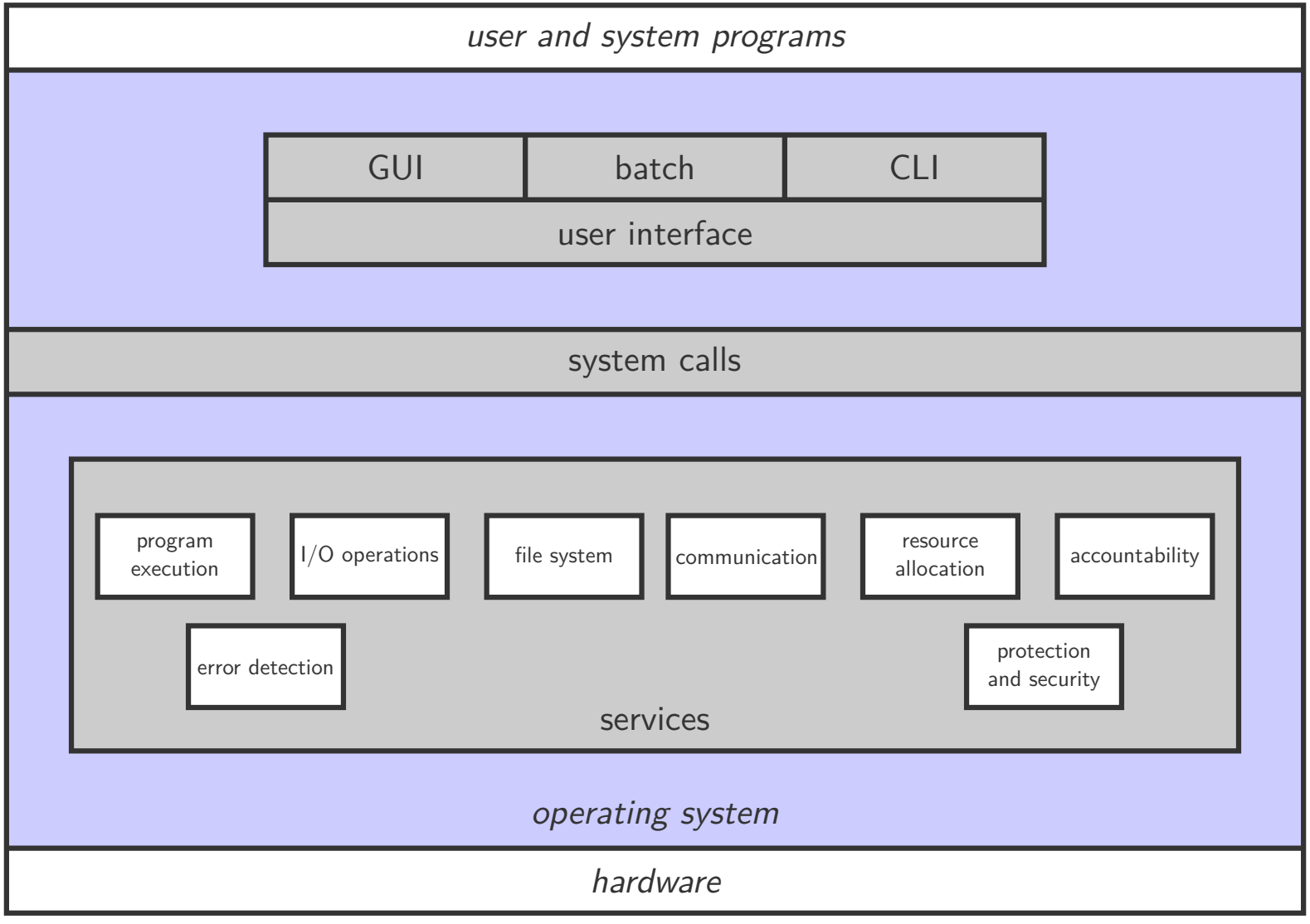
OS Services (1)

- The OS proposes two kinds of services
  - To the user
  - To the system itself
- Services offered vary by the OS

  But common service classes are identifiable



OS Services (2)



System Call

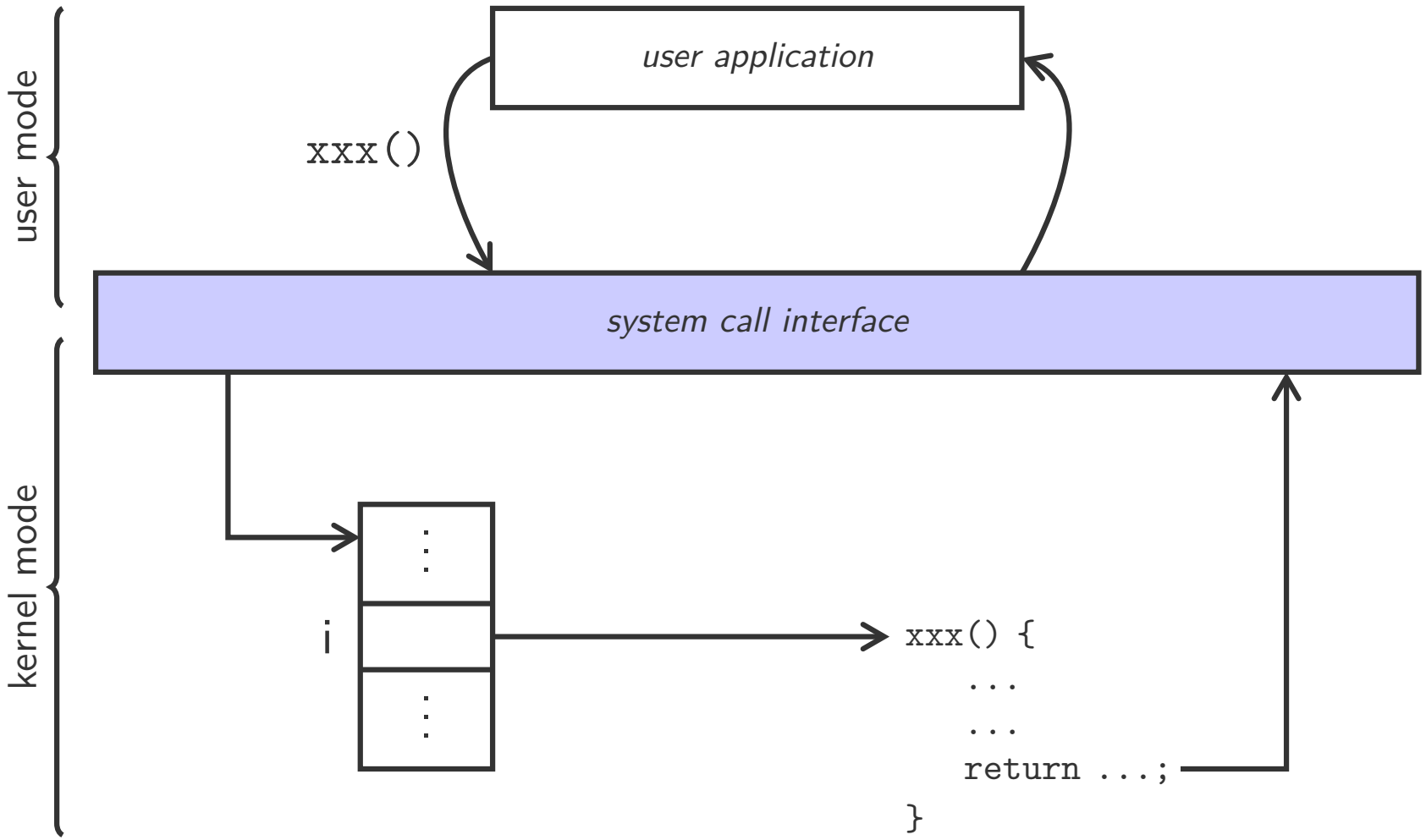
- Using the services of the OS through system calls
  - Interface to the services made available by the OS
  - Generally accessible as C/C++ routines
- Very low level tasks to write in assembly language

  For example to make direct access to the hardware
- Application Programming Interface (API) (Win32, POSIX, Go...)
  - Encapsulate system calls sequence to avoid direct call
  - Access to an API from a library (`libc`, for example)

File Copy Example (1)

1. Get the name of the source file
   - Display a sentence on the standard output
   - Read the standard input
2. Get the name of the destination file
   - Display a sentence on the standard output
   - Read the standard input
3. Open the source file, create and open the destination file
4. As long as reading the file does not fail
   - Read a character from the source file
   - Write the character to the destination file
5. Close the source and destination files

System Call Execution (1)



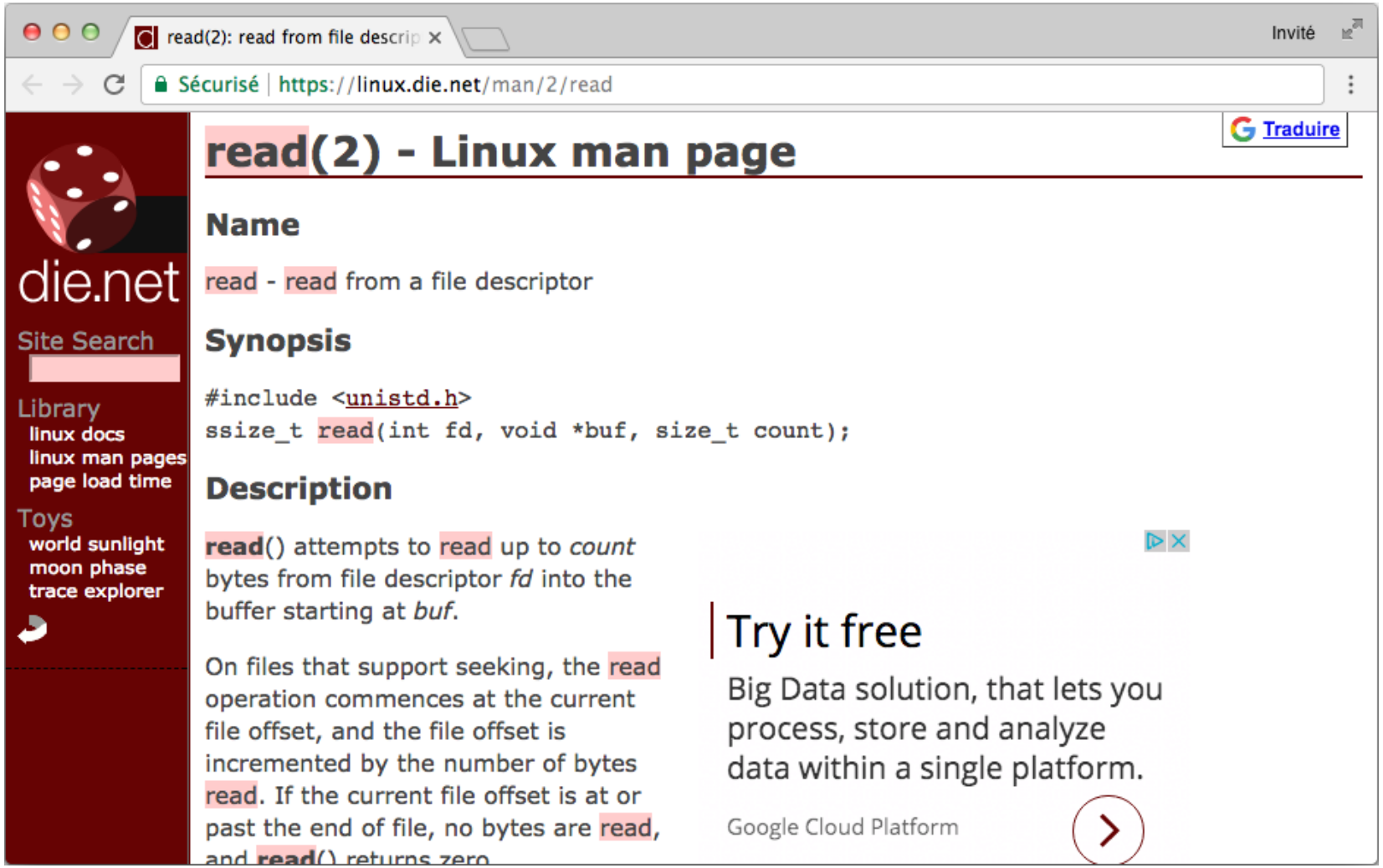
System Call Execution (2)

- Switching from user mode to kernel mode

  The system call is executed in a privileged mode of the OS

- Management through a system call interface
  - Relay by intercepting system calls in the API
  - Use an associative table numbering the system calls
- The user program ignores system call details
  - It must respect the API and pass correct parameters
  - It must understand the returned value

System Call Documentation



File Copy Example (2)

- Simplified version of a file copy

  No error management at all

```
1  void copy(char *src, char*dst)
2  {
3      char buffer[BUFFSIZE];
4      int srcFile = open(src, O_RDONLY);
5      int dstFile = open(dst, O_WRONLY | O_CREAT, 0666);
6
7      if (srcFile != -1 && dstFile != -1) {
8          int r;
9          while ((r = read(srcFile, buffer, BUFFSIZE)))
10             write(dstFile, buffer, r);
11
12         close(srcFile);
13         close(dstFile);
14     }
15 }
```

System Call Categories

- Available system calls depend on the OS

  Several system calls are common to most OS

- Six main categories of system calls can be identified
  - Process control
  - File management
  - Device management
  - IT maintenance
  - Communication
  - Protection

System Program

- Interface with the OS thanks to system program

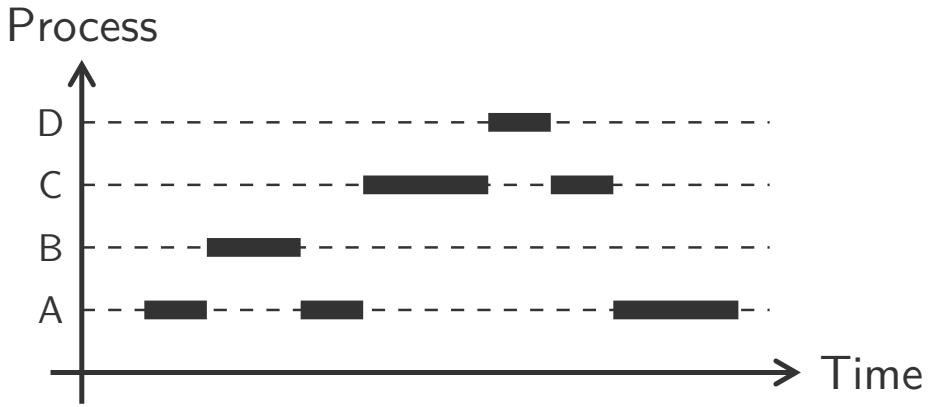
  System utility helps program development/execution

- Several categories of system programs
  - File management: `ls`, `mv`, `mkdir`, `rm`, etc.
  - Status information: `whoami`, `time`, `ps`, `top`, etc.
  - Text file editing: `vi`, `nano`, etc.
  - Support for programming languages: `cc`, `java`, etc.
  - Program loading and execution: `gdb`, `bash`, etc.
  - Communication: `ftp`, `mail`, `ssh`, etc.
  - Background services: `bg`, `screen`, `tmux`, etc.

Process and Memory

Process

- An OS can execute several programs
  - Resources sharing: CPU, memory, I/O devices, etc.
  - Compartmentalisation of programs
- A process is a running program
  - **Programme**: passive and on the disk
  - **Process**: active and in the memory



Process Abstraction

- A process is an abstraction for CPU resources

  Executable software is organised in sequential processes

- Each process has its own virtual processor

  It feels like it is the only one running on the processor

- No timing assumptions

  A process can be stopped at any time by the OS

Process Management

- The OS must continuously manage the executed processes

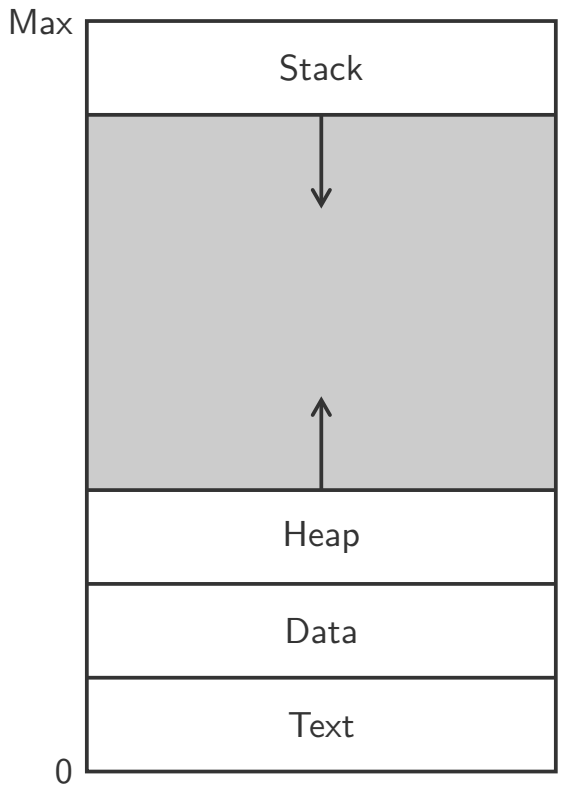
  On some systems, they are always active

- Several operations are made by the OS
  - Creation and deletion of processes
  - CPU allocation (*scheduling*) and other resource allocation
  - Synchronisation and communication
  - Deadlock management

Memory Structure

- A process is not only composed of the code of the program

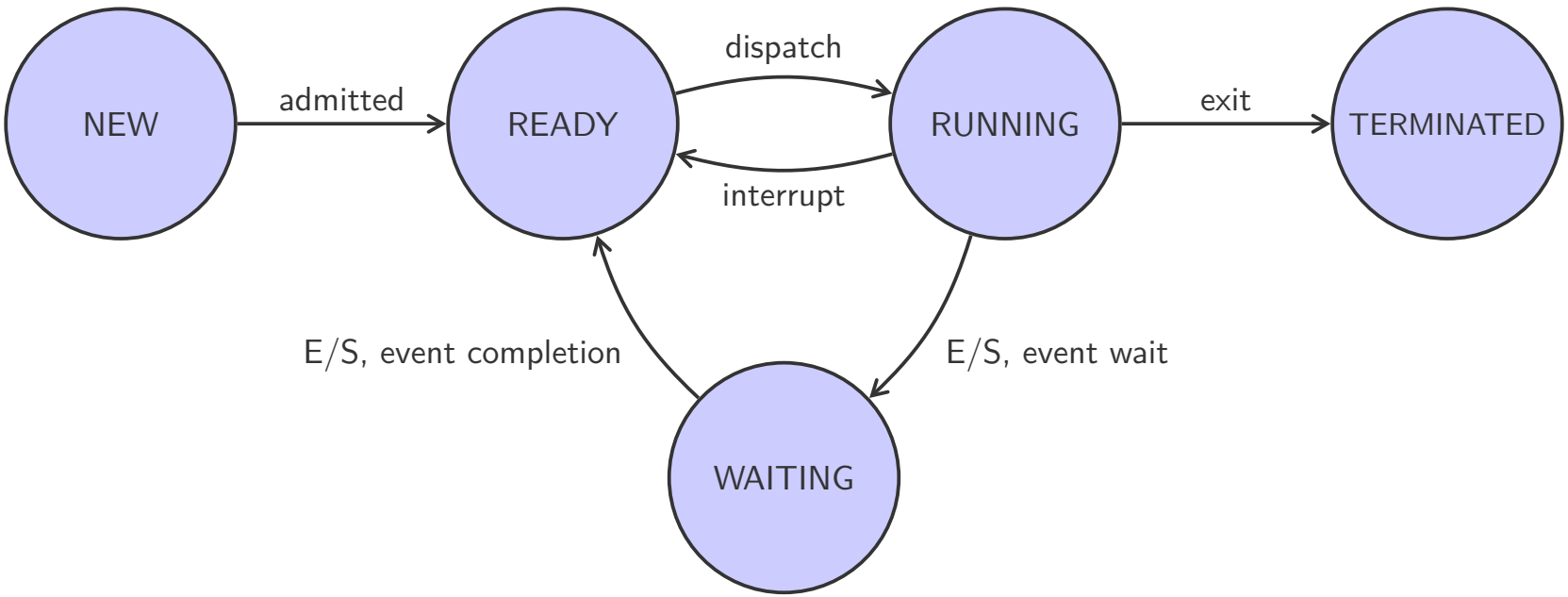
  It is only one of the area used in memory



Five-State Model (1)

- A process can be in different states during its execution

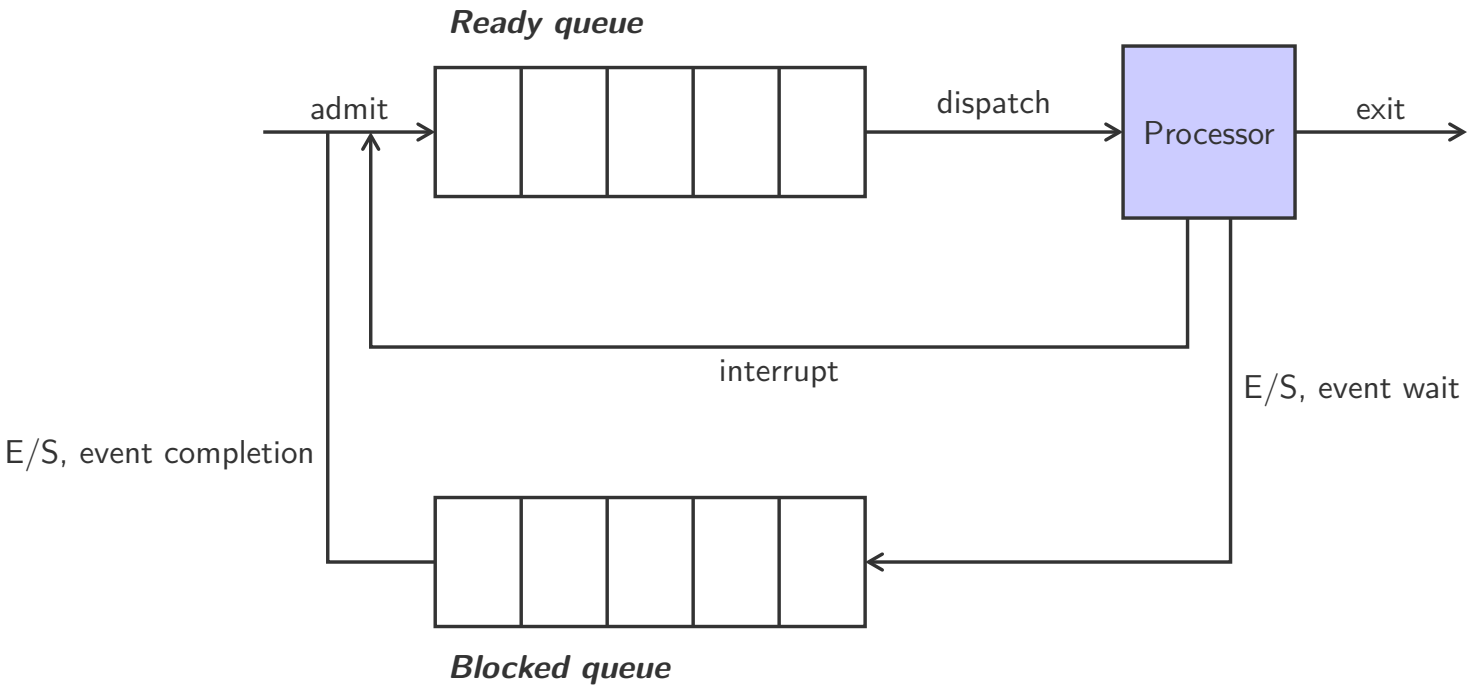
  Five main states are common to most OS



Five-State Model (2)

- The OS maintains several data structures to manage processes

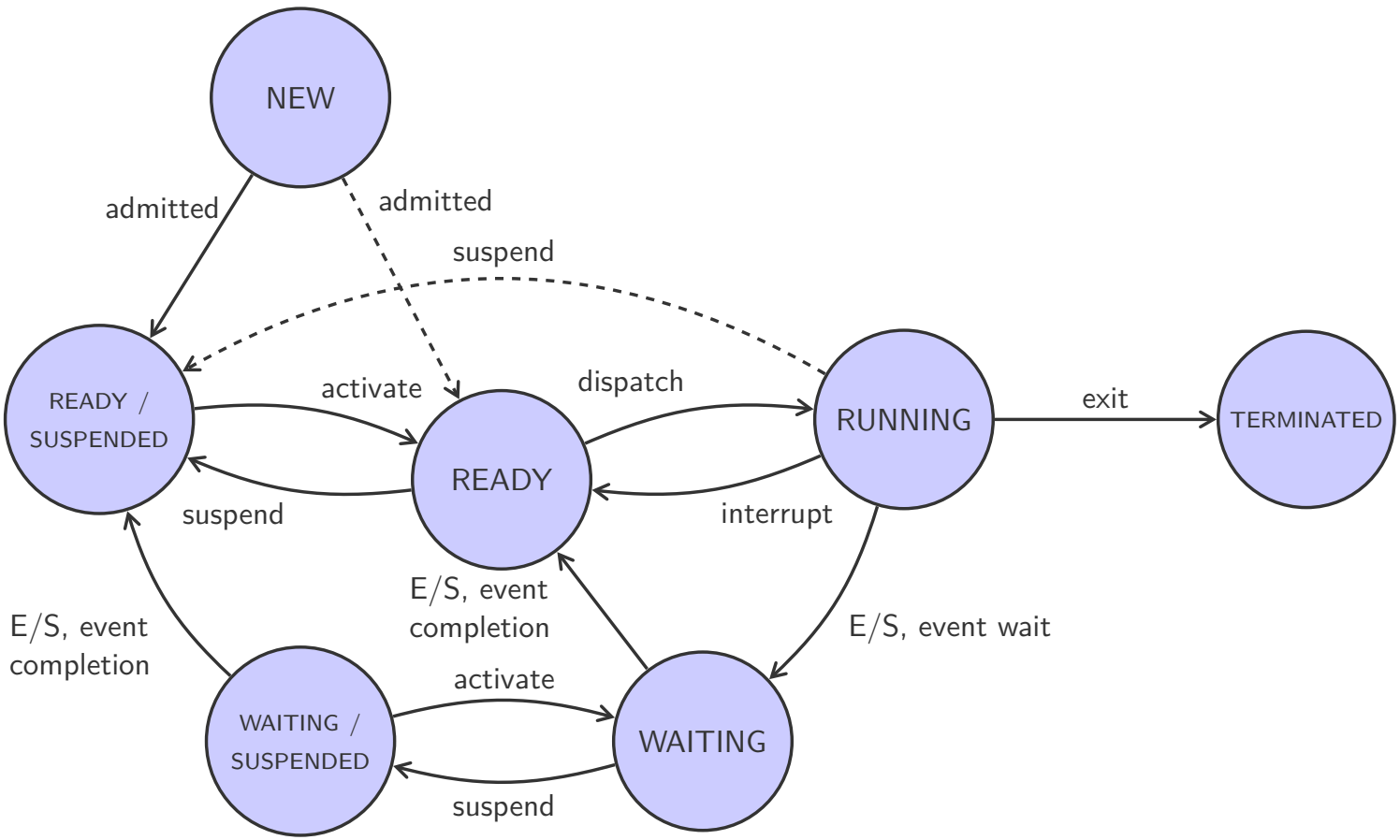
  Several FIFO queues can be used to handle process execution



Process Suspension

- A process can be temporarily suspended

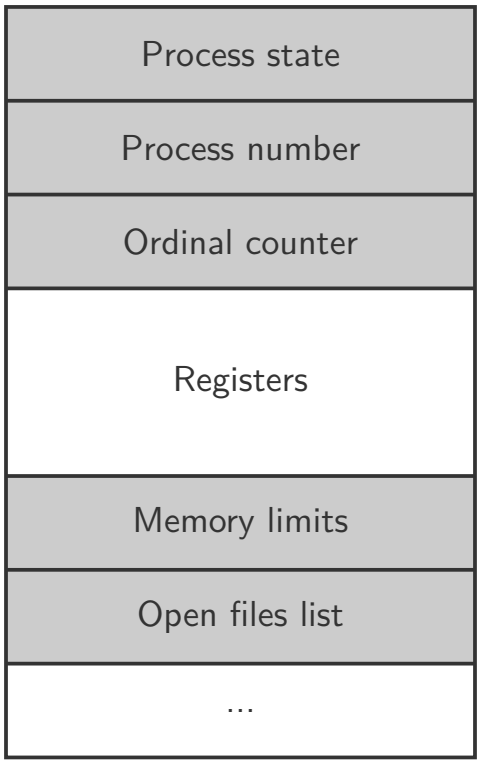
  Moved from the memory to the disk (swapping)



Process Table

- The OS keeps a process table in memory

  Each process has its own entry in this table



Memory

- Program loaded in memory becomes a process

  Disk → main memory

- Several algorithms are used to manage the memory

  1. Machine level management
  2. Pagination and segmentation

- The memory is just a large byte array

  Each byte is uniquely identified by an address

Memory Unit

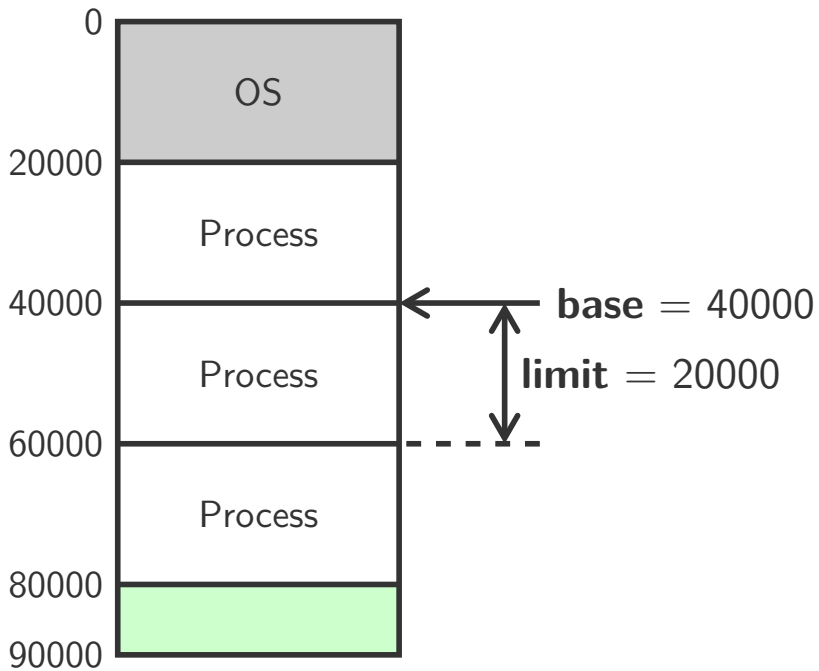
- When executing an instruction during a CPU cycle
  - **Load** instruction from the memory
  - **Store** instruction to the memory
- The memory unit sees a flow of memory addresses

  The unit responds to the requests without any interpretation
- Physical addresses manipulation

  Addresses of the bytes located in the main memory

Memory Protection (1)

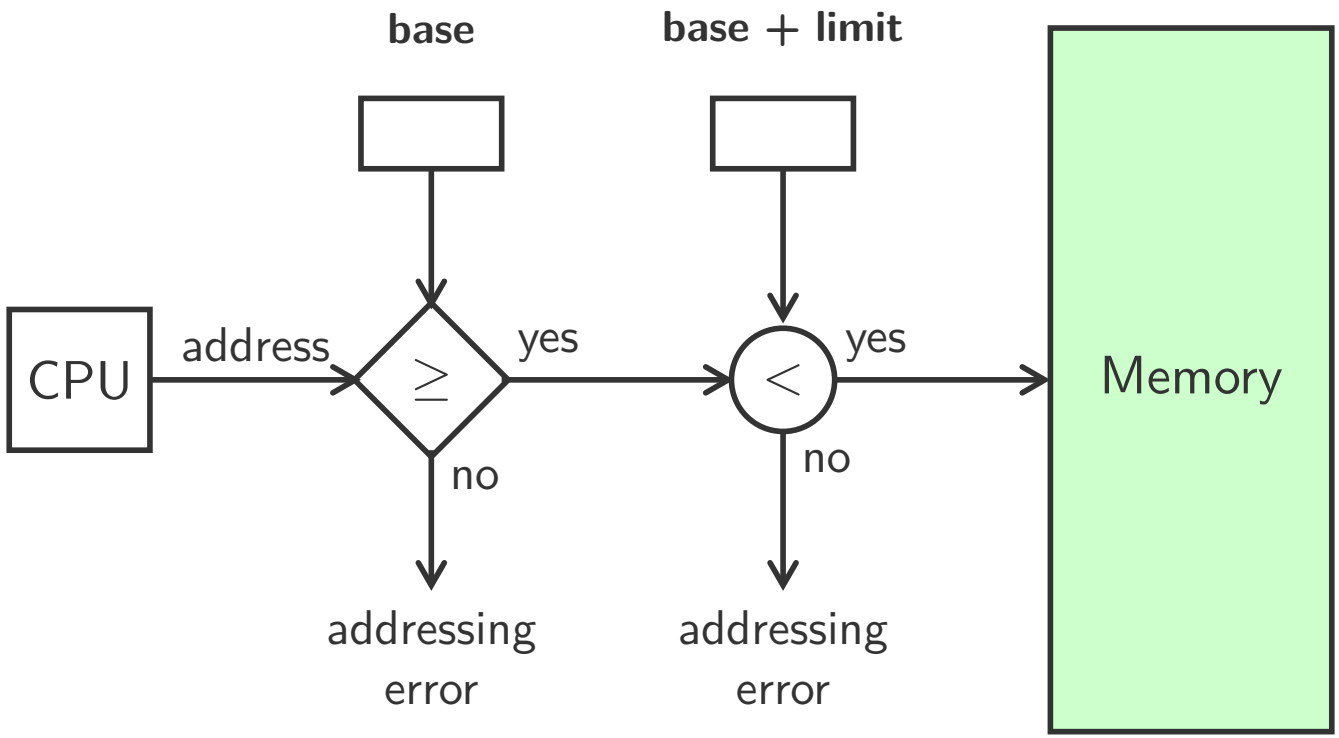
- Two registers are use to delimit the memory area
  - **base**: the smallest legal physical address
  - **limit**: the size of the memory area



Memory Protection (2)

- Only the OS can access the **base** and **limit** registers

  Using a privileged instruction in kernel mode



Physical/Logical Address

- Program and date must be in memory, before

  Size limitation corresponds to the physical memory

- Two separate and different address spaces
  - **Physical** addresses generated by the CPU
  - **Logical** addresses loaded in the memory-address register

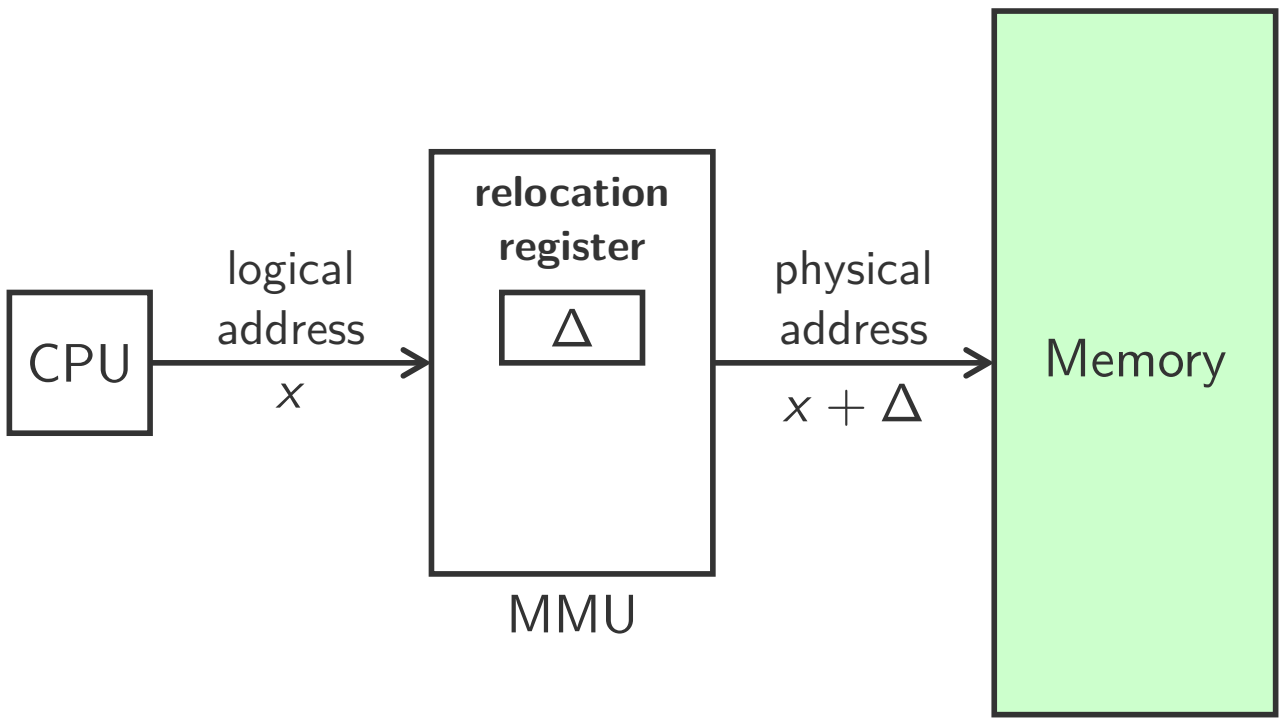
    This address is read by the memory unit

Memory Management Unit

- Hardware system to convert physical $\rightarrow$ logical address

  The user program only manipulates logical addresses

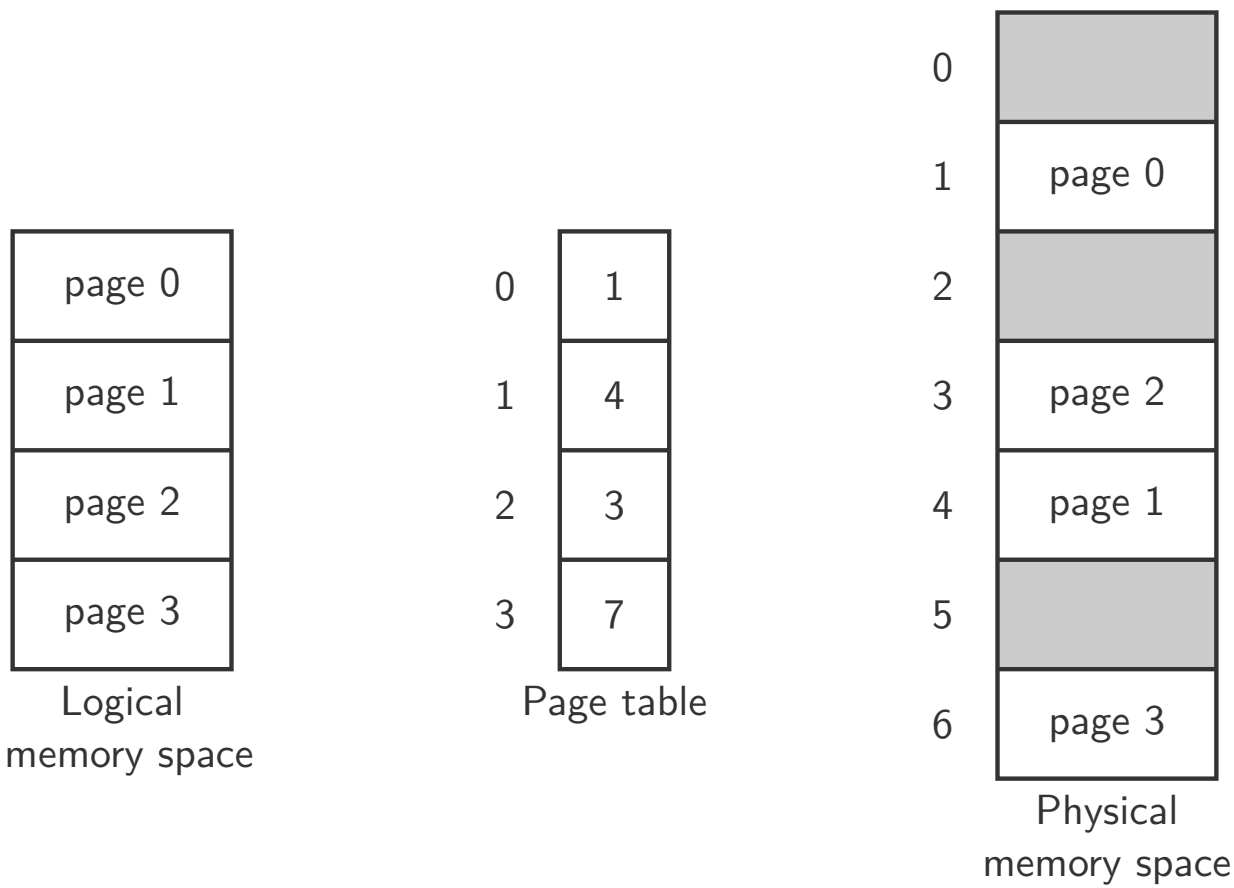
- A simple MMU just adds the value of a **relocation register**



Pagination

- Main memory divided into fixed size frames

  Logical memory divided into pages with the same size



Address Computation (1)

- A logical address is a two-tuple

  $\langle$*page number, shift*$\rangle$

- The OS holds one page table for each process

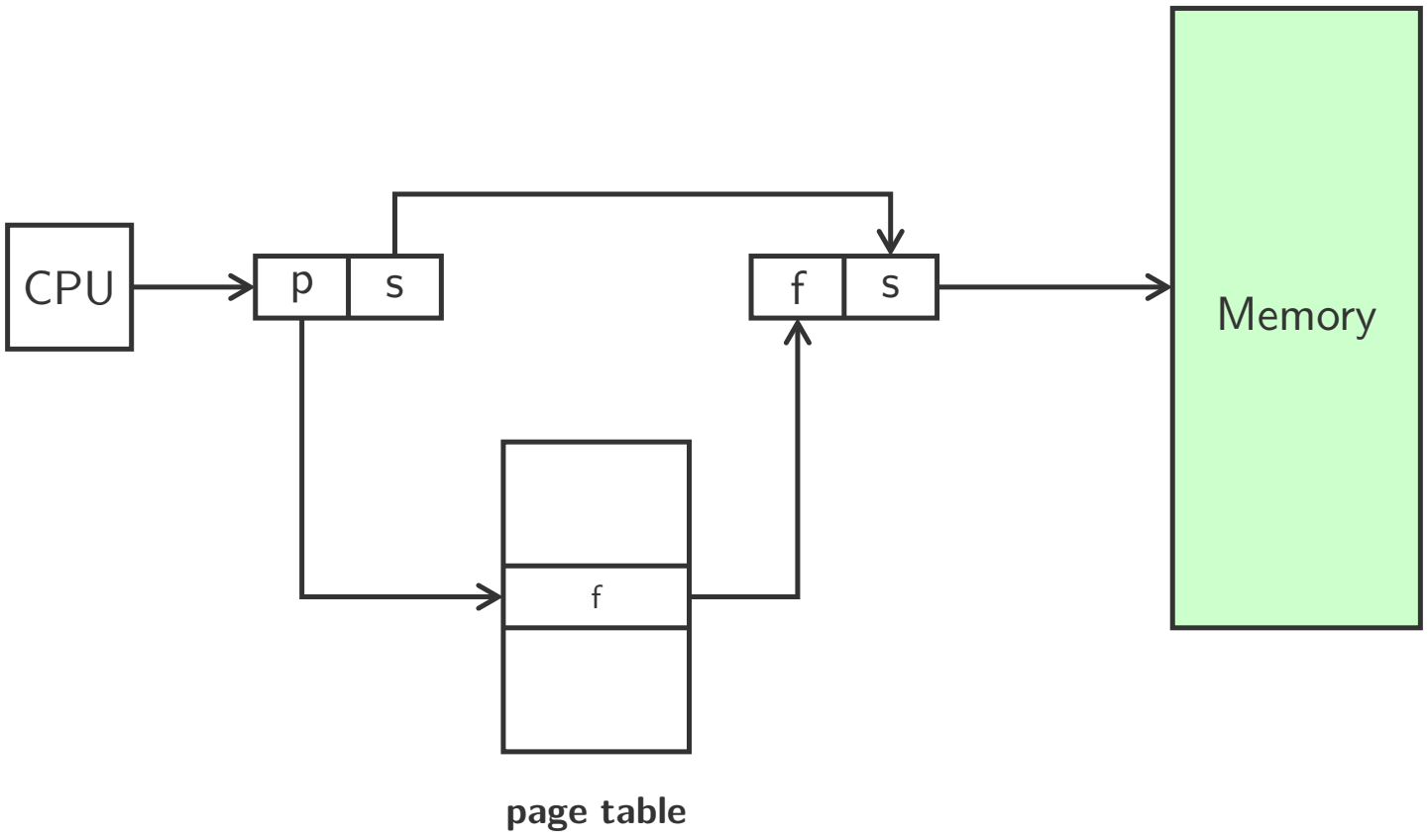
  Entries with physical address of the page

- Two registers used to configure the page table
  - Page-Table Base Register (PTBR)
  - Page-Table Length Register (PTLR)

Address Computation (2)

- The address computation uses data from the page table

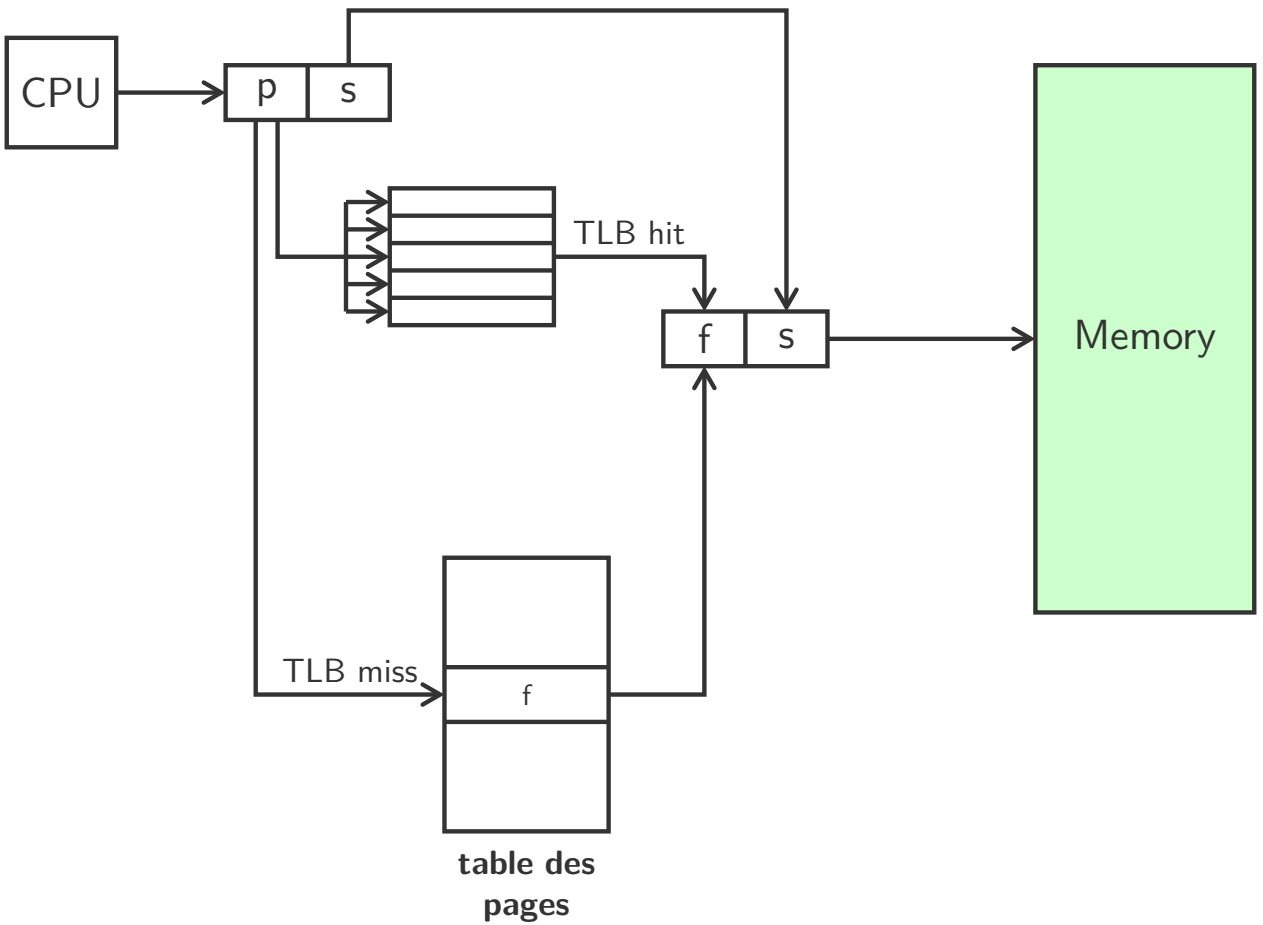
  The shift is applied to the frame, once identified



Translation Look-Aside Buffer (TLB)

- The process can be speed up with a fast cache memory

  Decreasing the number of main memory accesses



References

- Abraham Silberschatz, Peter B. Galvin, & Greg Gagne (2013). *Operating System Concepts*, John Wiley & Sons, ISBN: 978-1-11809-375-7.
- William Stallings (2017). *Operating Systems: Internals and Design Principles*, Pearson, ISBN: 978-1-29221-429-0.

Credits

- quapan, November 26, 2011, https://www.flickr.com/photos/hinkelstone/7050697671.
- Beao, September 9, 2009, http://commons.wikimedia.org/wiki/File:Personal_computer,_exploded_6.svg.
- Evan-Amos, February 19, 2017, http://commons.wikimedia.org/wiki/File:PS4-Console-wDS4.jpg.
- McZusatz, January 22, 2013, http://commons.wikimedia.org/wiki/File:Boeing_787-8_N787BA_cockpit.jpg.
- Jochembr, January 5, 2014, http://commons.wikimedia.org/wiki/File:Snackomatic.jpg.
- Dennis Skley, August 12, 2014, https://www.flickr.com/photos/dskley/14711793077.
- Jean-Pierre Dalbéra, September 23, 2014, https://www.flickr.com/photos/dalbera/15766751411.