Mohamed Boudiaf
University
-M'sila-

جامعة محمد بوضياف - المسيلة
Université Mohamed Boudiaf - M'sila

Faculty of Mathematics
and Computer Science

Computer Science
Department

OPERATING SYSTEMS II

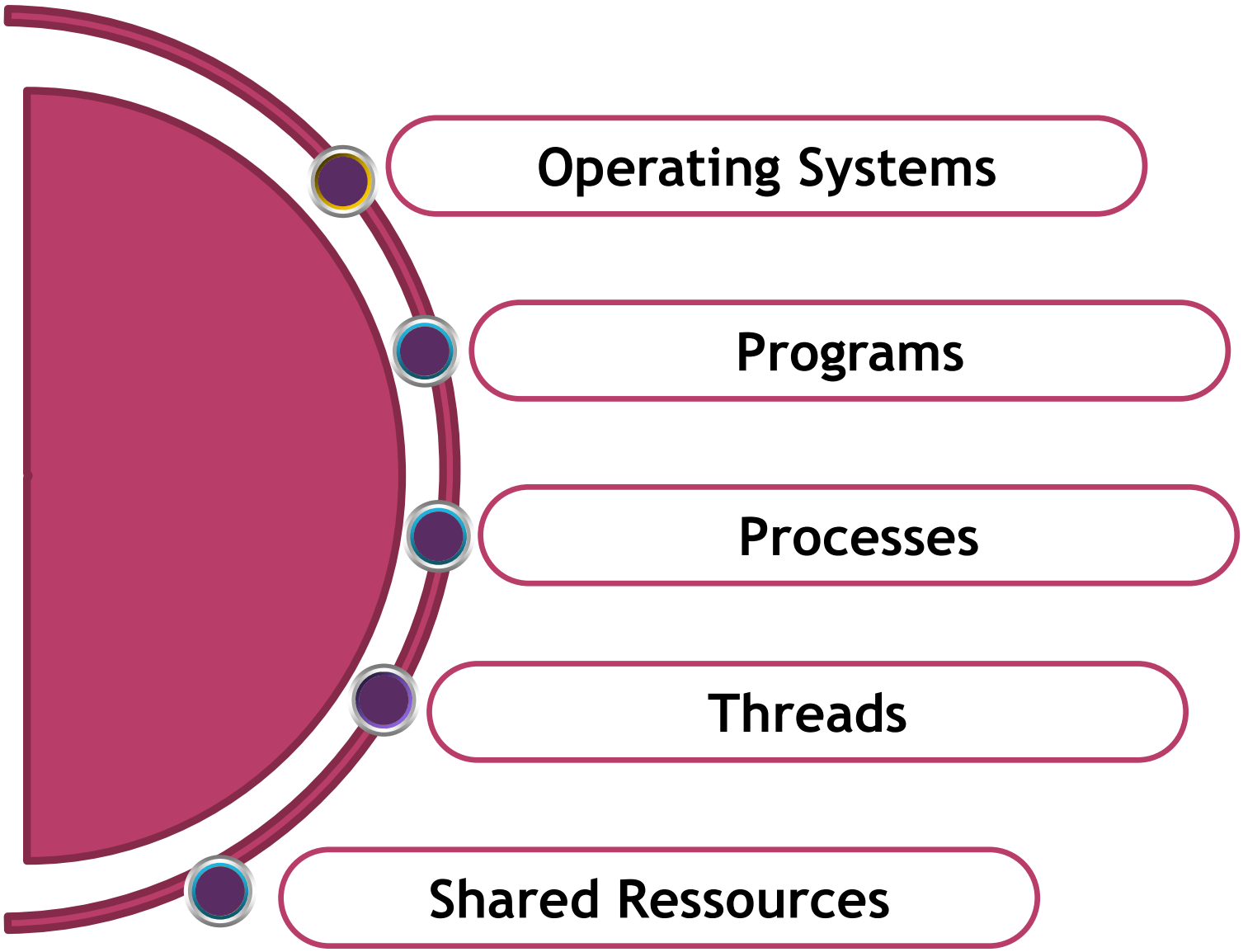
CHAPTER I: INTRODUCTION

3rd Year, Computer Science – SI

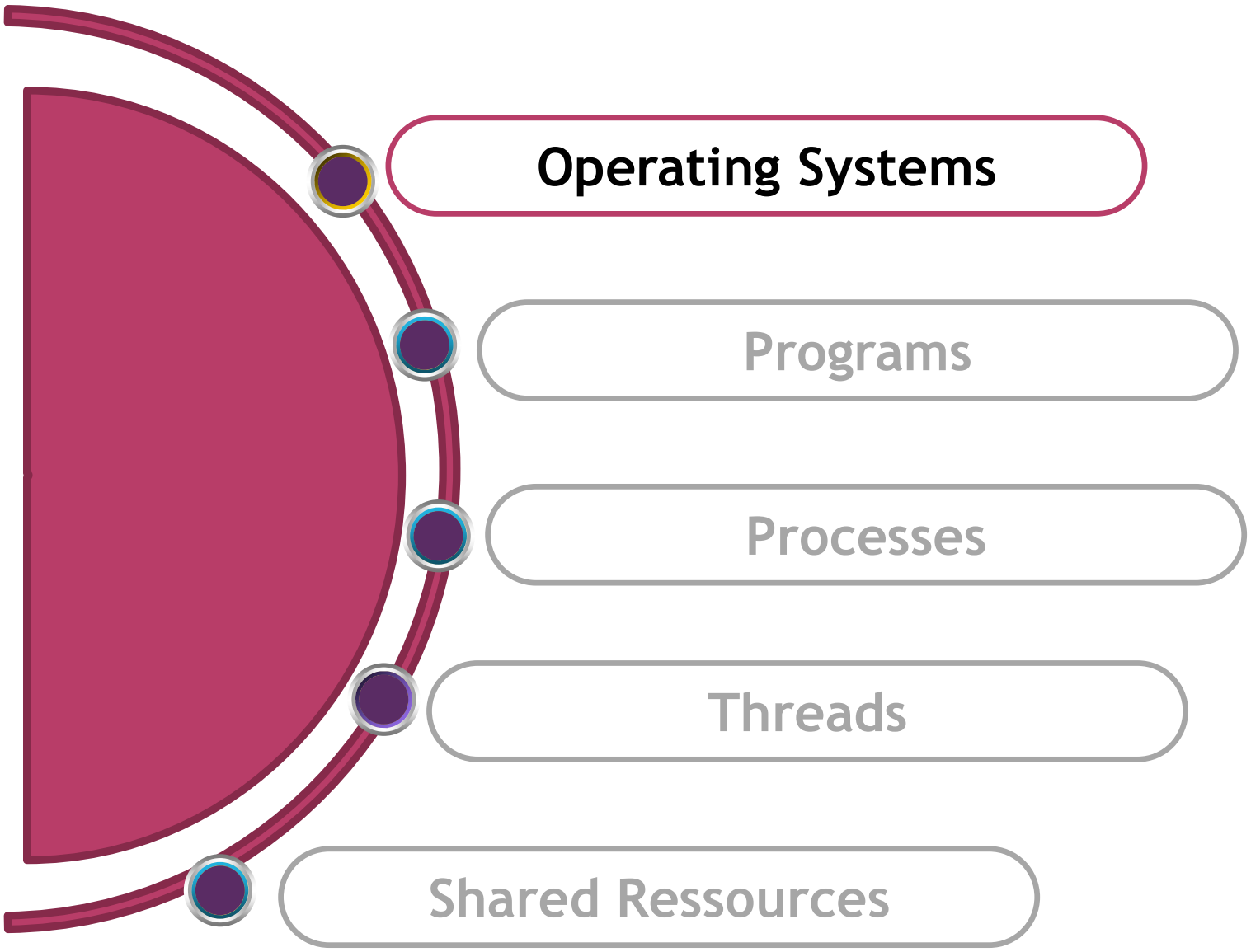
Dr. Hanene CHERAIT

2025-2026

OUTLINE

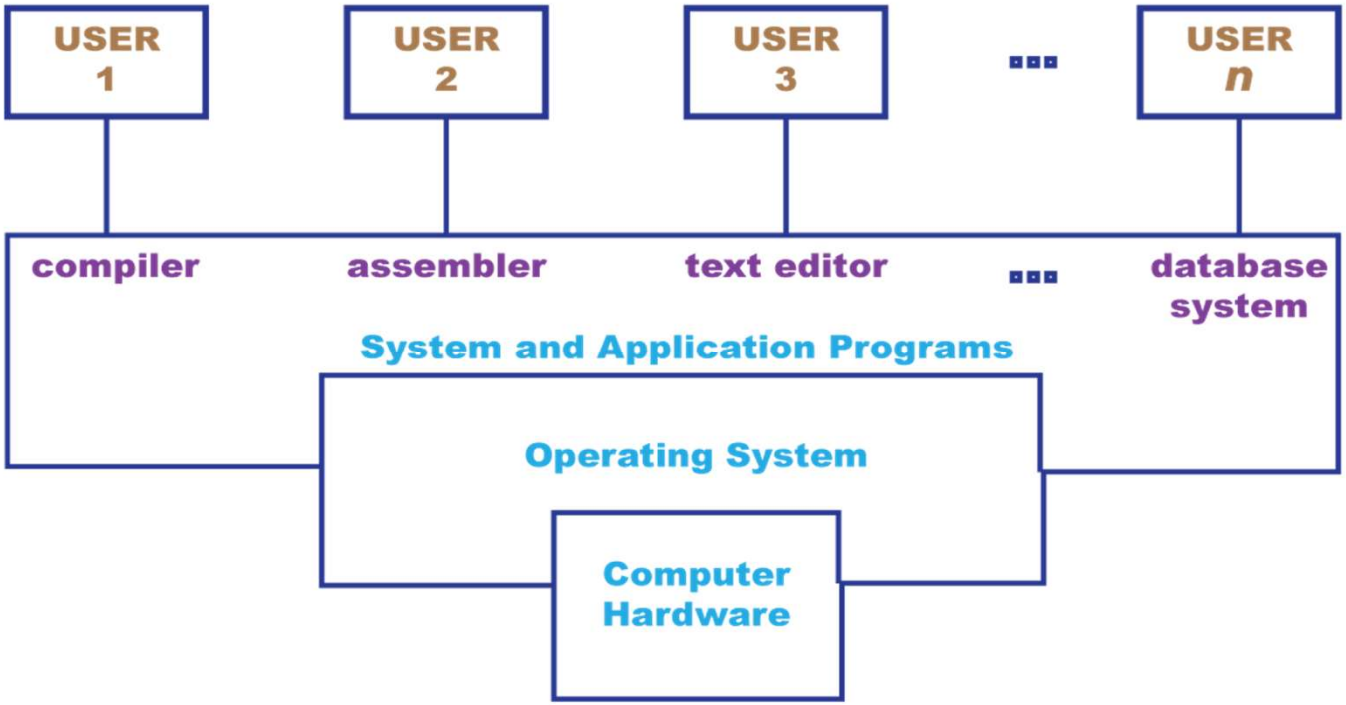

Operating Systems
Programs
Processes
Threads
Shared Ressources

OUTLINE


Operating Systems
Programs
Processes
Threads
Shared Ressources

THE OPERATING SYSTEM (OS)

- A computing system consists of:
- **Hardware**
- **System software**
  - Also called **Operating System (OS)**
- **Application programs**



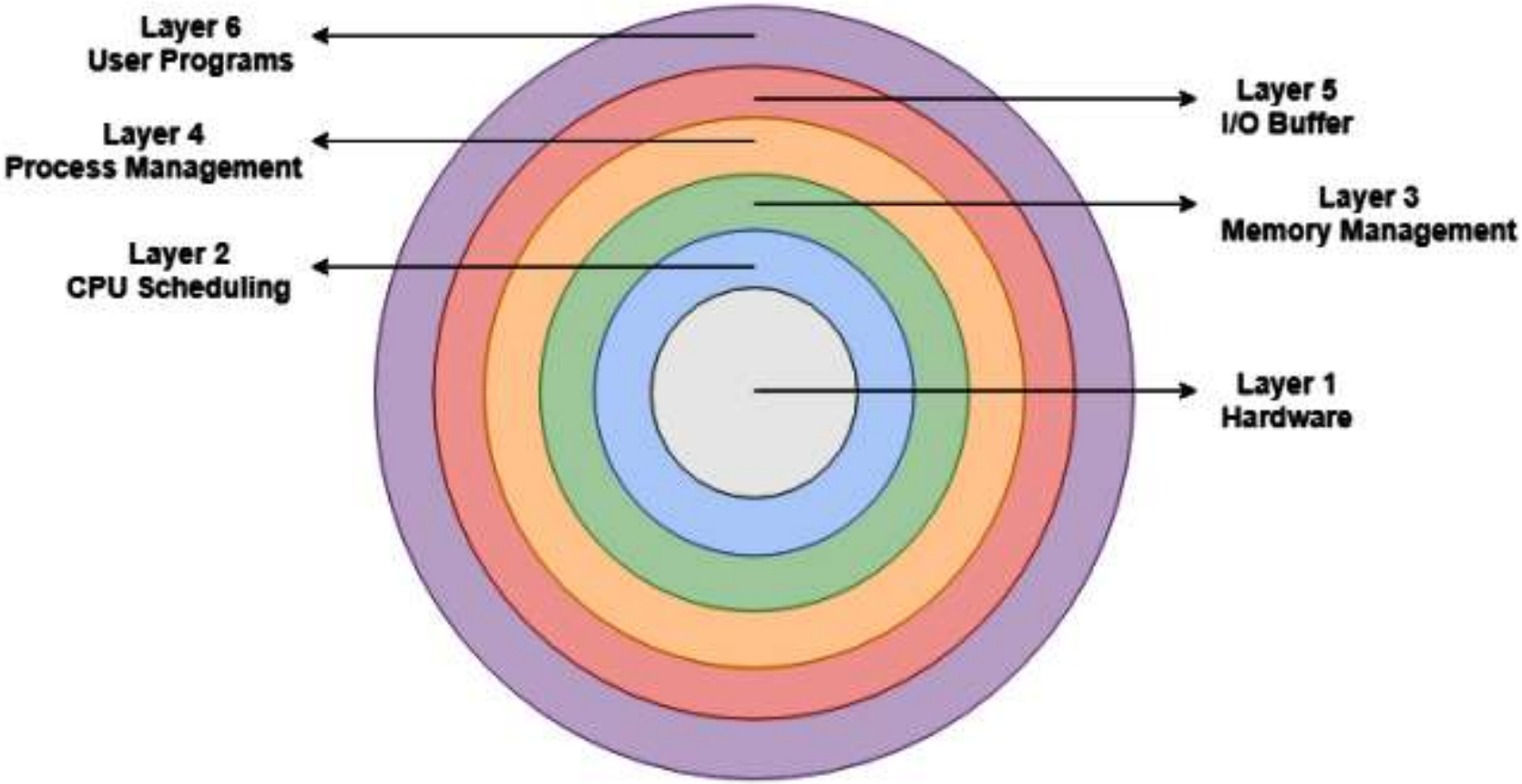
THE OPERATING SYSTEM (OS)

- A program acting as an intermediary between the user and the computer hardware.
- An environment that allows a user to run programs.
- An OS is a software layer with the following characteristics:
  - Simplified user interface
  - Efficient and optimal management of computer resources (memory, CPU, peripherals, etc.)
  - File management
  - Process management

KEY FUNCTIONS OF AN OS

- **Process Management**
  - Execution scheduling
  - Event management
- **Information Management**
  - Memory management
  - File management
- **Communication Management**
  - Input/Output (I/O) management
  - Network management
- **Security and Protection**
  - Firewall (e.g., Windows Firewall)

STRUCTURE OF AN OS

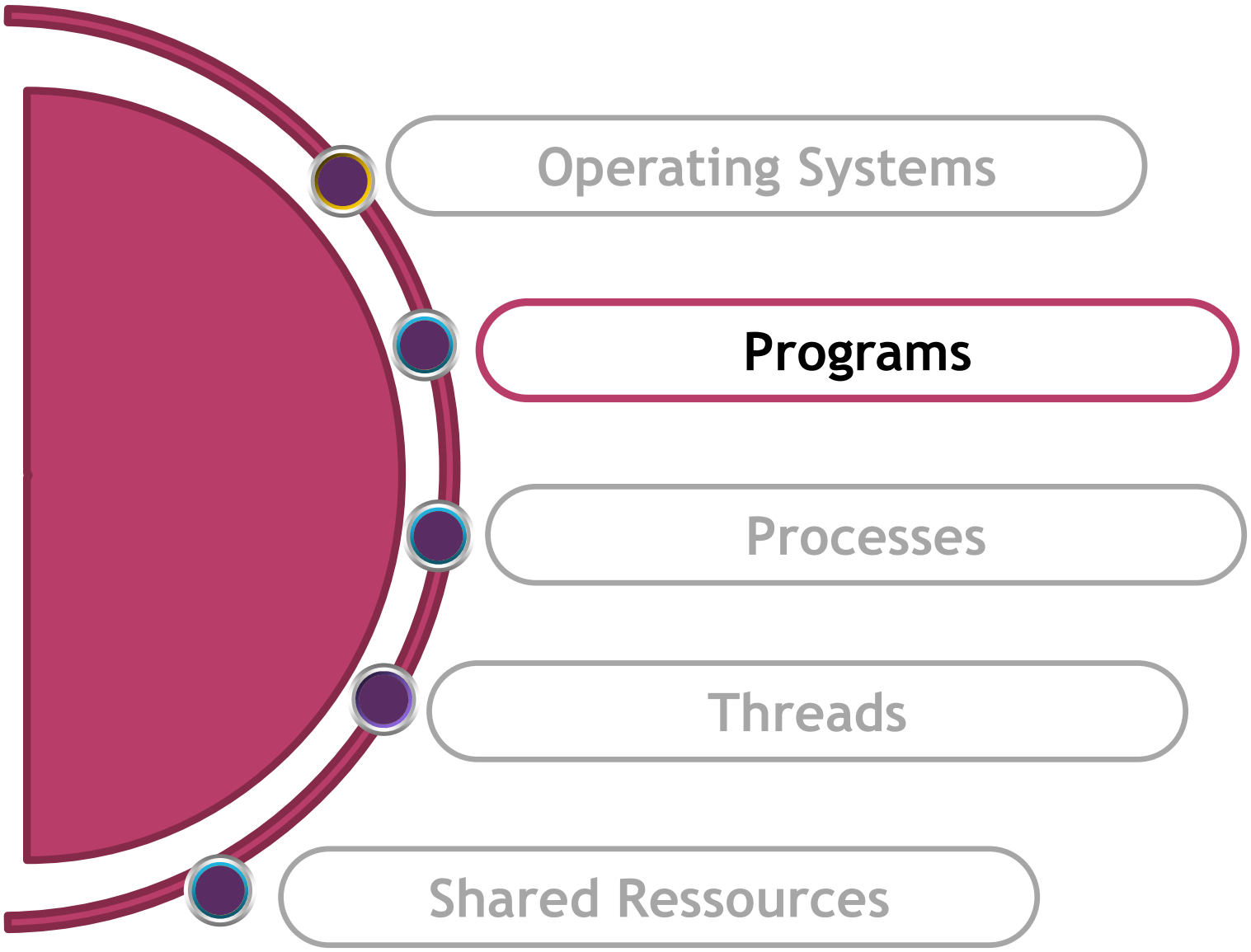


Layers of operating system

TYPES OF OPERATING SYSTEMS

- An Operating System depends on the type of resources to be managed:
  - **Multitasking System**
    - Several programs run simultaneously on the computer.
    - **Multiprogramming**
  - **Multiprocessor System**
    - Utilization of several CPUs.
    - Management of parallel activities.
  - **Multiuser System**
    - Network (web server, etc.).
  - **Distributed System**
    - Several autonomous computers interconnected in a network cooperate to accomplish well-defined activities.
  - **GRID System**
    - A large number of interconnected resources.

OUTLINE


Operating Systems
Programs
Processes
Threads
Shared Ressources

CONCEPT OF A PROGRAM

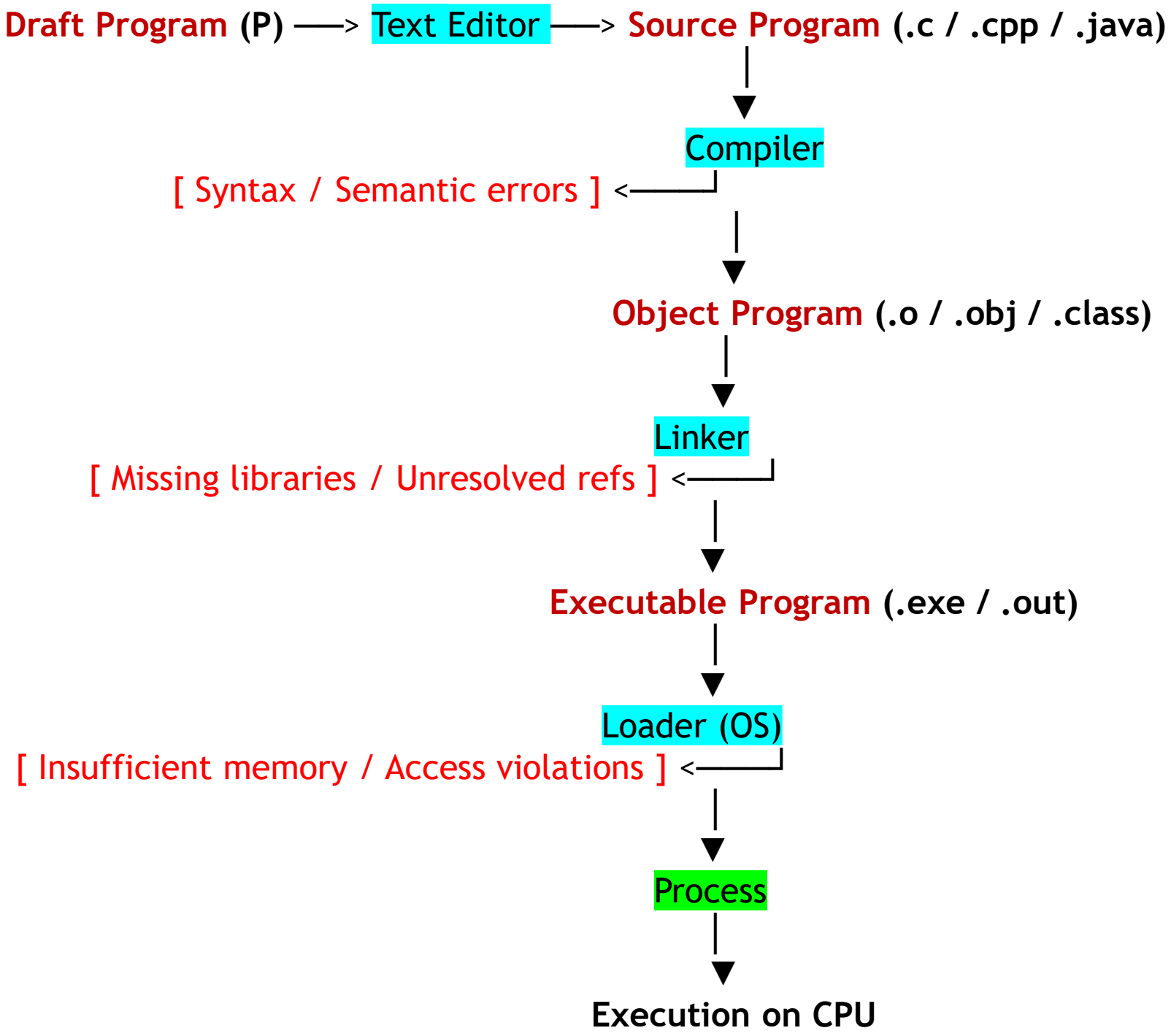
- The computer can only handle binary, that is, a sequence of 0 and 1.
- It is necessary to use a programming language in order to write readable instructions that are understandable by humans.
- These programs are translated into machine language (binary) by a compiler.
- A program is a sequence of instructions executable by the computer.
- It is a static entity occupying space on disk or in memory; as long as it is not executed, it produces no action.

EXECUTION FLOW OF A PROGRAM IN AN OPERATING SYSTEM

- The development of a program, from problem analysis to debugging, requires numerous software tools that together form a programming environment.

- These tools rely on the services of the operating system.

- A program is generally written in a high-level language such as C, Visual Basic, Java, Python etc.


Draft Program (P) —> Text Editor —> Source Program (.c / .cpp / .java)
Compiler
[Syntax / Semantic errors] <
Object Program (.o / .obj / .class)
Linker
[Missing libraries / Unresolved refs] <
Executable Program (.exe / .out)
Loader (OS)
[Insufficient memory / Access violations] <
Process
Execution on CPU

EXECUTION FLOW OF A PROGRAM IN AN OPERATING SYSTEM

- **Text Editor**
  - A program that allows editing and saving code in a file.
  - It is an interactive software that lets you enter text using a keyboard and save it into a file.
  - Example: Notepad, vi, vim, emacs, etc.

- **Compiler**
  - A program that converts source code written in high-level languages (Pascal, C, Ada, Java, etc.) into a program written in machine language (binary).

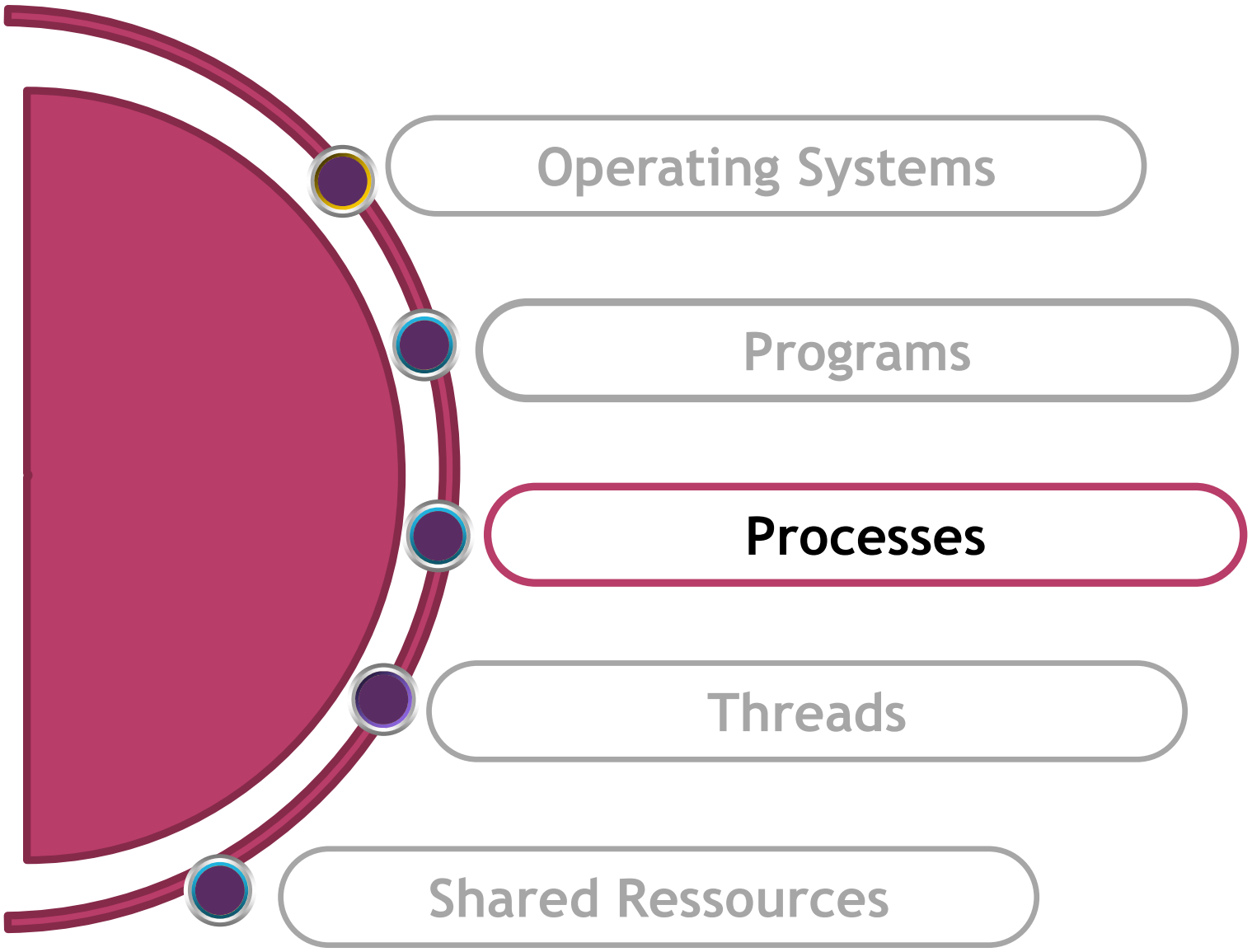
EXECUTION FLOW OF A PROGRAM IN AN OPERATING SYSTEM

- **Linker**
  - A source program may consist of:
    - Instructions
    - Locally defined data
    - External data
    - External procedures or subprograms (function libraries)
  - The linker is software that allows combining several programs into a single program.
  - Before executing the program, it is necessary to assemble the non-local parts and the external procedures with the program.

EXECUTION FLOW OF A PROGRAM IN AN OPERATING SYSTEM

- **Loader**
  - The role of the loader is to load the object module, obtained after linking, into the main memory in order to execute it.
  - The loader is a program that installs or loads an executable into main memory from an address determined by the operating system.

OUTLINE

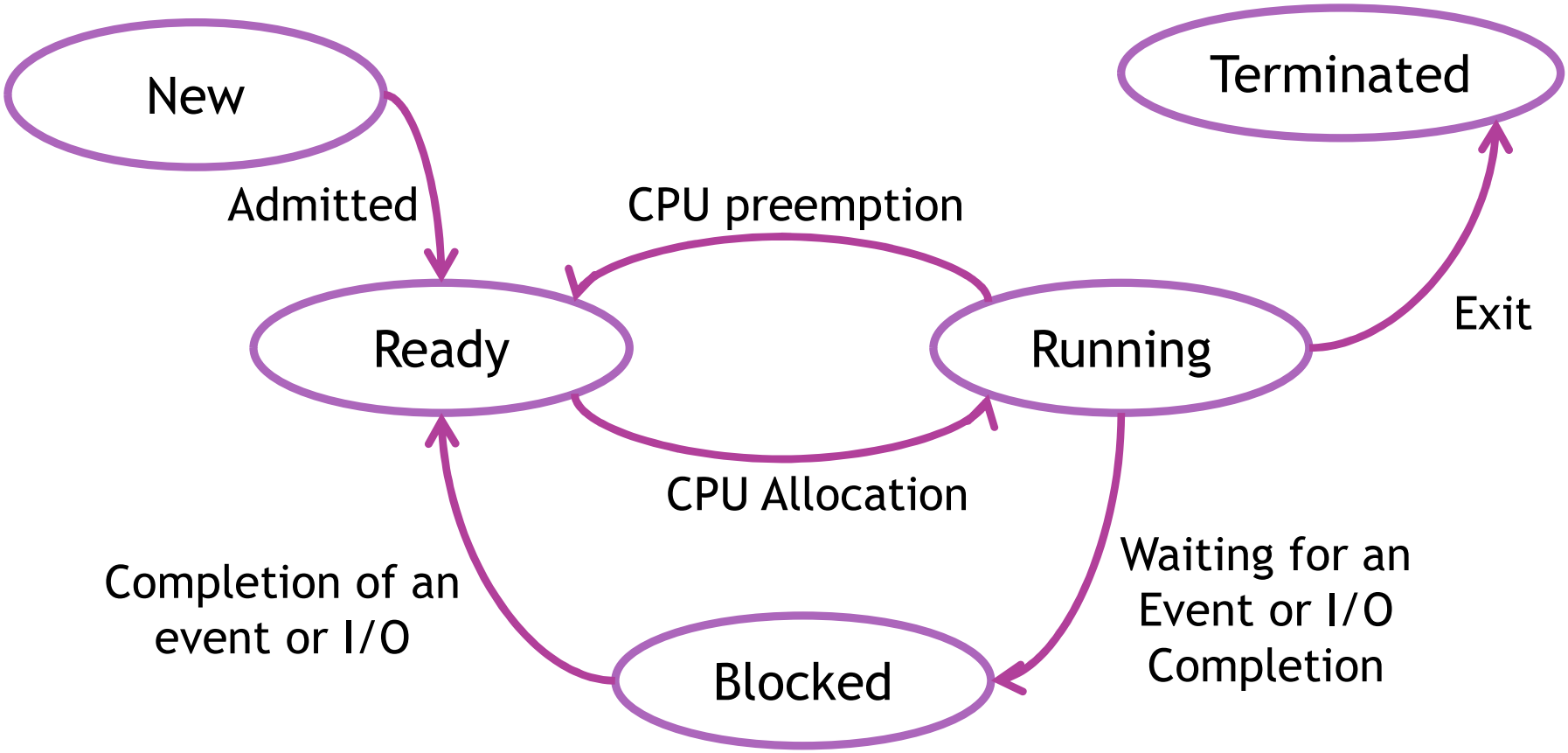

Operating Systems
Programs
Processes
Threads
Shared Ressources

PROGRAM VS PROCESS

- **Program**
  - A static entity occupying space on disk or in memory.
  - As long as it is not executed, it produces no action.
  - It can generate one or more processes.
- **Process**
  - A sequence of actions produced by executing a series of instructions.
  - A dynamic entity that is born, lives, and dies.
  - Process = instructions + resources

A process is a program in execution.

PROCESS STATES



PROCESS STATES

- **New**: The process has just been created.
- **Ready**: The process is ready for execution. It is placed in the ready queue, waiting for CPU allocation.
- **Running (Active):** The process is currently executing. It has been assigned to a free CPU, and its instructions are being executed.
- **Blocked (Waiting):** The process is waiting for a physical or logical resource other than the CPU to continue execution (memory, completion of I/O, reception of a signal, etc.).
- **Terminated**: The process has finished its execution.

PROCESS CONTROL BLOC

- Each process is represented in the operating system by a **Process Control Block (PCB).**
- The PCB contains:
  - **Process identifier (PID)**
  - **Current state of the process**
  - **Program counter** indicating the next instruction to execute
  - **Memory management information** (e.g., memory page tables used by the process)
  - **Variables and procedures used**
  - **Process priority**
  - Etc.
- Process Table
  - All the Process Control Blocks (PCBs)
  - Creating a process involves reserving and initializing an entry in this table.

PROCESS OPERATIONS

1. **Process creation**
2. **Processus termination**

PROCESS OPERATIONS

- **Creation**
  - During its execution, a process can **create multiple new processes**.
  - System call for process creation
    - Commande **create process-name**
  - The process performing the creation is called the **parent**, and the created process is called the **child**.
    - The parent process can **allocate its resources** among its children.
    - The parent process can **transfer initialization data** to the child.

PROCESS OPERATIONS

- **Process Termination**
- can occur in **two ways:**
  - **Normal termination:**
    - At the end of the execution of its last instruction.
  - **Abnormal termination:**
    - If a malfunction is detected,
    - or the process is no longer needed.
    - **Command: KILL process-name**

- Process termination results in:
  - **Destruction of its descendants**
  - **Release of its entry** in the process table
  - **Release of the resources** occupied by the process

PROCESS RELATIONSHIPS

- All modern operating systems are capable of performing **multiple tasks simultaneously**.
  - **Multiple processes** can run at the same time.

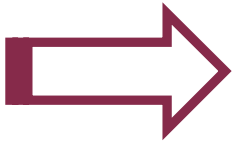
- Processes in a system **interact with each other** in two main ways:
  - **Cooperation**
  - **Competition**

COOPERATION

- Two or more processes can **cooperate to achieve a common goal** or produce a joint result.
- A process is called **cooperative** if it can **affect other processes running in the system** or be affected by them.
  - Any process that **shares data with other processes** is a cooperative process.
- **Types of Cooperation:**
  - **Variable Sharing (Indirect Cooperation)**
    - Processes do not need to know each other explicitly.
    - They **share a logical address space** containing both data and code.
  - **Message Passing (Direct Cooperation)**
    - Processes have **send and receive primitives** that allow them to communicate.

COMPETITION

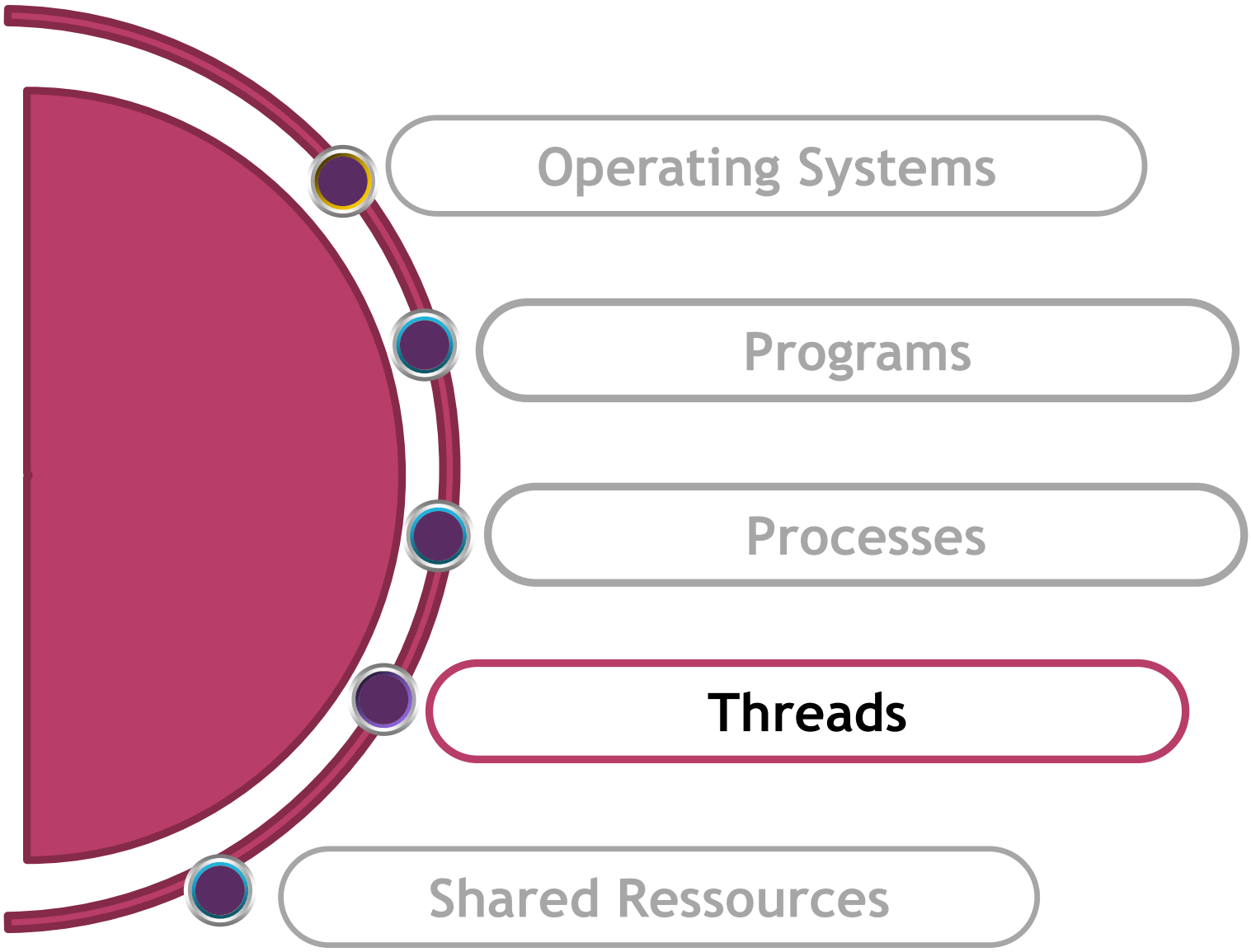
- When two or more processes are **competing for the use of a resource**
  - memory, procedure, variable, I/O device, file, etc.



➢This resource is a **critical resource**.
➢These processes are called **competing (or concurrent) processes**.

- For the CPU, all processes are competing. To solve problems related to conflicts over the use of critical resources:
  - Synchronization

OUTLINE


Operating Systems
Programs
Processes
Threads
Shared Ressources

THREADS

A process is defined by the **resources it uses** and the **memory location on which it executes**.

It would be desirable for **resources to be shared** and for them to be **accessed concurrently**.

Solution: Create **multiple threads** within the same process.

The concept of a thread aims to allow **multiple execution threads to share a set of resources** and work together to accomplish a given task.

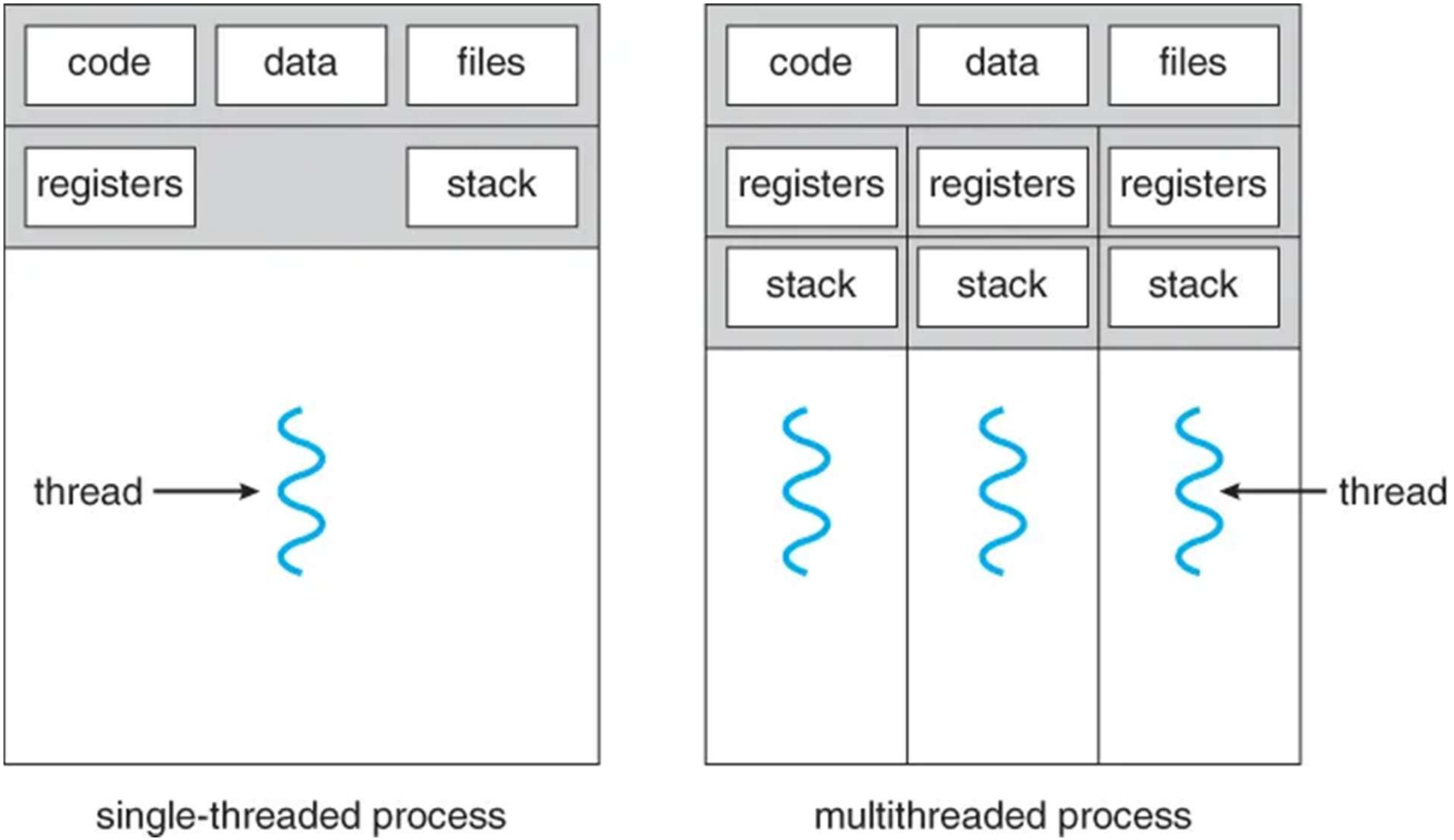
THREADS

Definition: A thread is a **lightweight process.**

- Each thread includes a set of properties:
  - **Unique identifier**
  - **Program counter**
  - **Call stack** and a number of registers
- Threads within the same process **share the code and data sections** as well as other **operating system resources**.
  - **Example:** open files and signals
- **Multithreading:**
  - Multiple threads are part of the same process

THREADS

Definition: A thread is a **lightweight process.**



single-threaded process

multithreaded process

ADVANTAGES OF USING THREADS

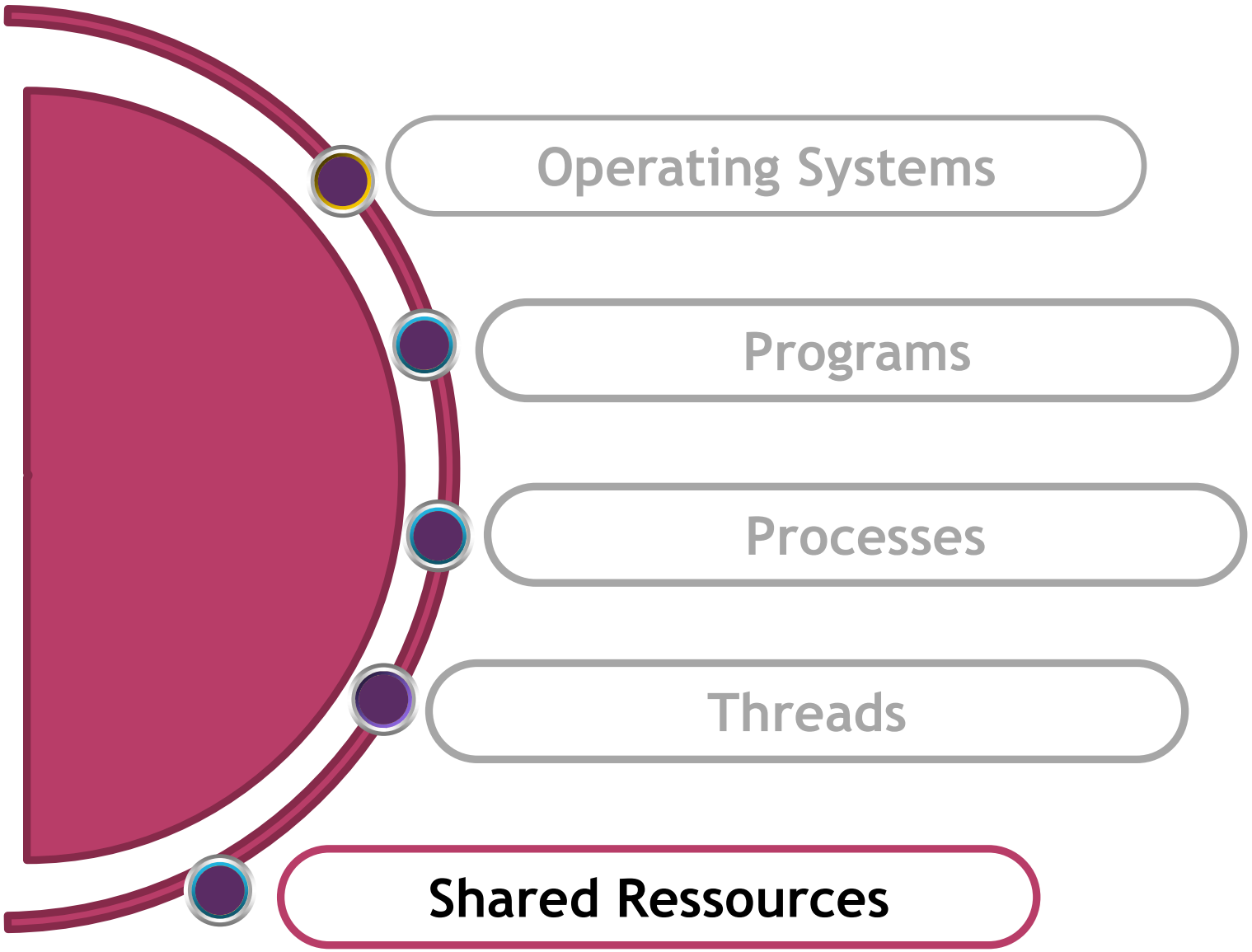
Simplification of context, even when working on a **single-processor machine.**

- **Processes**
  - Each process is assigned a **context** that contains a large amount of information, making its creation costly.
  - **Examples:**
    - **Multiprocessor machines** → a high number of processes are instantiated
    - **Client/server applications** → many copies of a server process are instantiated to serve different clients
- **Threads**
  - The context is **reduced to the essential information** needed for its execution.
  - The **virtual space of a thread** consists of:
    - The **code executed by the thread**
    - The **data defined within the thread**
    - A **call stack specific to the thread**

EXAMPLES OF THREADS

- The majority of applications and software running on today's PCs are **multithreaded**.
- **A word processor:**
  - A thread that **renders graphics**
  - A thread that **captures keystrokes** from the user
  - A thread that **performs background spell-checking and grammar corrections**
- **A web browser:**
  - A thread that **displays images or text**
  - A thread that **retrieves data from the network**

OUTLINE


Operating Systems
Programs
Processes
Threads
Shared Ressources

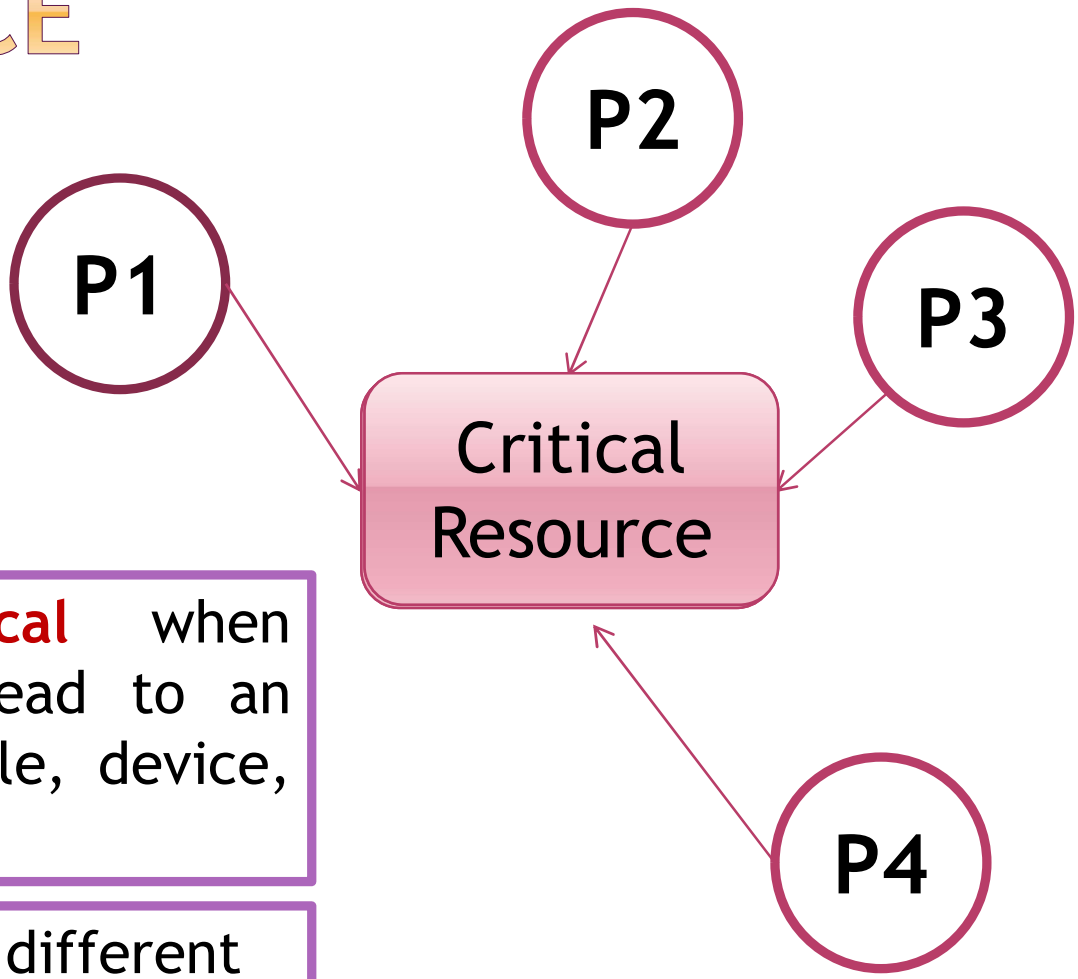
CONCEPT OF A RESOURCE

- **Any entity required by a process to execute**
- **Two types of resources:**
  - **Physical resources:** CPU, memory, input/output units, devices, etc.
  - **Logical (virtual) resources:** variables, files, databases, communication channels, etc.
- The **use of a resource** requires following a **specified usage protocol** that defines how to manipulate the resource.
  - A **resource allocator** is responsible for handling requests for **allocation, release, and possibly preemption** of each resource.

SHARED RESOURCE

- For each resource, an important characteristic is **the number of processes that can use it simultaneously:**
  - **Any number**
    - No control is needed.
    - For example, a file opened in read-only mode can be accessed by any number of processes.
  - **Several but limited**
    - It is necessary to **control allocations** to ensure the limit is not exceeded.
    - **Synchronization** is required.
  - **At most one process using the resource**
    - The resource is called a **critical resource**.
    - Processes are said to be in **mutual exclusion** for access to this resource.

SHARED RESOURCE



A resource is called **critical** when **concurrent access** to it can lead to an **inconsistent state** (variable, table, device, etc.).

Synchronization of the actions of different processes on shared resources.

The part of a program where access to a shared resource occurs is called a **critical section (CS).**

CRITICAL SECTION

- A **critical section** is a set of instructions in a program that can produce **unpredictable (or inconsistent) results** when executed simultaneously by different processes.

- It is a sequence of instructions that operates on **one or more shared (critical) resources** and requires **exclusive use** of these resources.

THANK YOU FOR YOUR ATTENTION

NEXT CHAPTER

PROCESS SYNCHRONIZATION

That wraps up this chapter. Up next, we'll dive into process synchronization and see how processes can work together safely when sharing resources using synchronization techniques.