Princípios da Computação

Processes.

Instituto Superior de Engenharia do Porto

What is a process?

- A **process** is a **program in execution**, representing an active instance of a program.

- A program is a static entity (typically stored in a file), consisting of a set of instructions that define a sequence of operations for the computer to perform.

- Unlike a program, a process is a dynamic entity that performs operations and interacts with system resources.

- **A program becomes a process when it is loaded into memory and begins execution.**

Anatomy of a process

- **Constant Part:**
    - Program Code (Text Section): The unchanging set of instructions that define the process's operations.
- **Dynamic Part:**
    - Data (global variables, stack, heap), program counter, CPU registers, opened devices.
- The dynamic part evolves during execution, capturing the progress and the current state of the process.

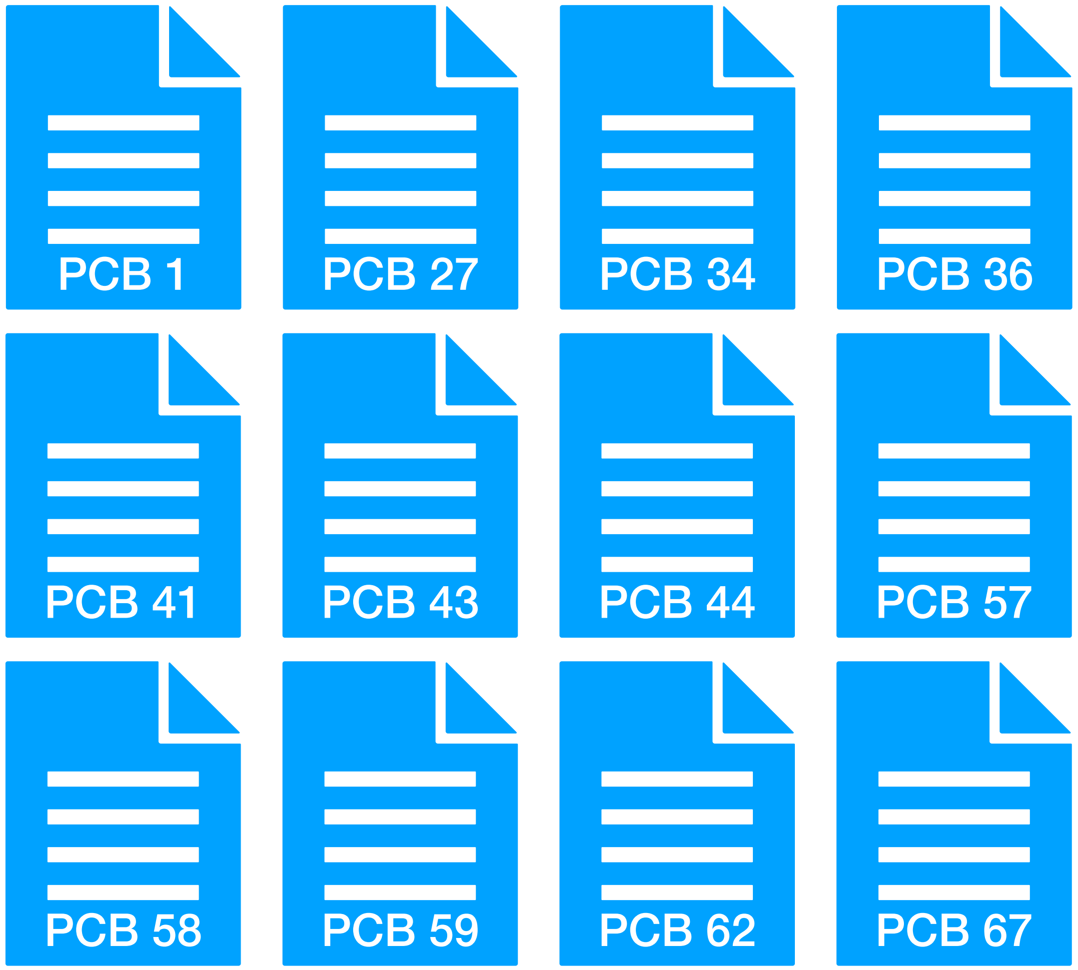
What a process needs to run

- **Memory:** text and data.
- **CPU Time:** to execute instructions and perform computations.
- **I/O Devices:** access to peripherals like keyboards, disks, and network interfaces.
- **Files:** for accessing persistent data.

How does the OS handles multiple processes?

- The kernel maintains a **Process Control Block (PCB)** for each process in the system.
- The PCB stores all the essential information required by the operating system to efficiently manage and track processes.
- The **Process Table** is a data structure maintained by the operating system to keep track of all active processes.

PROCESS TABLE



Instituto Superior de Engenharia do Porto

Information in the Process Control Block

- **Process Identifier (PID):** A unique number that identifies the process.
- **Process State:** Current state of the process (if it is running or not).
- **Program Counter:** Address of the next instruction to execute.
- **CPU Registers:** Snapshot of all process-specific registers.
- **CPU Scheduling Information:** Priorities and scheduling queue pointers.
- **Memory Management:** Details of memory allocated to the process.
- **I/O Status:** Allocated I/O devices and list of open files.
- **Accounting Information:** CPU usage, elapsed time, and time limits.

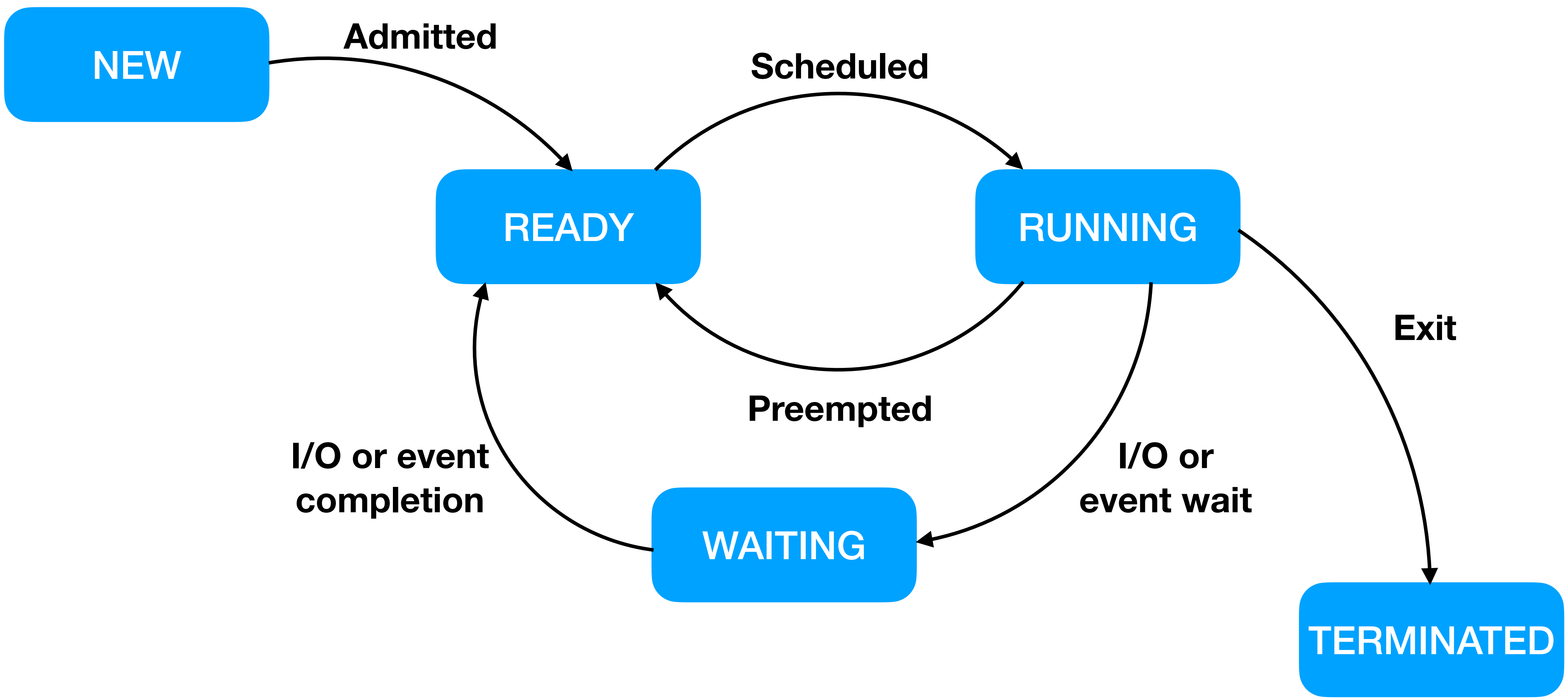
Process Identifier (PID)

- Operating systems assign each process a **unique identifier** (PID).
- Traditionally, PIDs are **serial numbers**, ensuring efficient assignment.
- When the PID limit is reached, the system rolls back and reassigns released identifiers.
- In Unix systems, **process number 1** is a special process.
    - It is the **init process**, responsible for initialising the system.
    - This process remains active until the system is shut down.

Process State

- As a process progresses, it transitions between the following states:
- **New:** The process is being created.
- **Running:** The process instructions are being executed by a processor.
- **Waiting:** The process is paused, awaiting a specific event.
- **Ready:** The process is ready and waiting to be assigned to a processor.
- **Terminated:** The process has finished execution.

Process state transitions



isep Instituto Superior de Engenharia do Porto

Process lifecycle transitions

- **New to Ready:** A new process undergoes initialization, including tasks like allocating memory and loading the program into memory. Once formed, it transitions to the Ready state.

- **Ready to Running:** The operating system selects a ready process for execution, assigns it to a processor, and transitions it to the Running state.

- **Waiting to Ready:** When the reason for a process's wait is resolved (e.g., an I/O operation completes or the awaited event occurs), the process transitions back to the Ready state.

Process lifecycle transitions

- **Running to Waiting:** A running process pauses when it requests an I/O operation or waits for an event (via a system call), transitioning to the Waiting state.

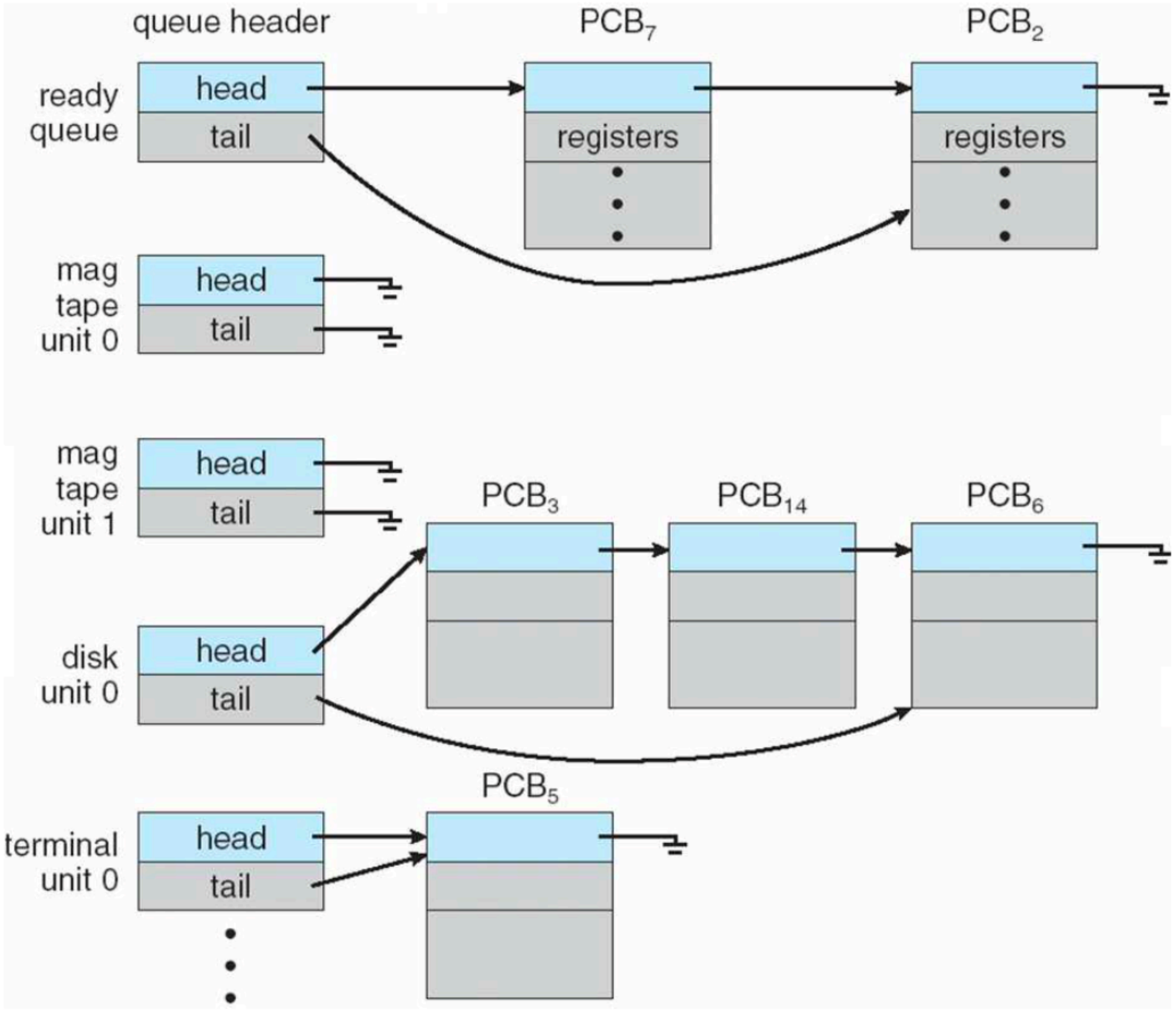
- **Running to Ready:** The operating system preempts a running process to assign the CPU to another ready process, moving the preempted process to the Ready state.

- **Running to Terminated:** When a process completes its execution, it transitions to the Terminated state.

Process scheduling overview

- To maximize CPU utilization and facilitate quick process switching, the process scheduler determines which process will execute next on the CPU.
- Scheduling involves managing these key queues:
    - **Job Queue:** Contains all processes in the system, including those not yet loaded into main memory.
    - **Ready Queue:** Consists of processes in main memory that are ready and waiting for CPU execution.
    - **Device Queues:** Hold processes waiting for access to a specific I/O device.
- Processes dynamically transition between these queues depending on their state and resource requirements.

Ready and device queues



Instituto Superior de Engenharia do Porto

Context switching

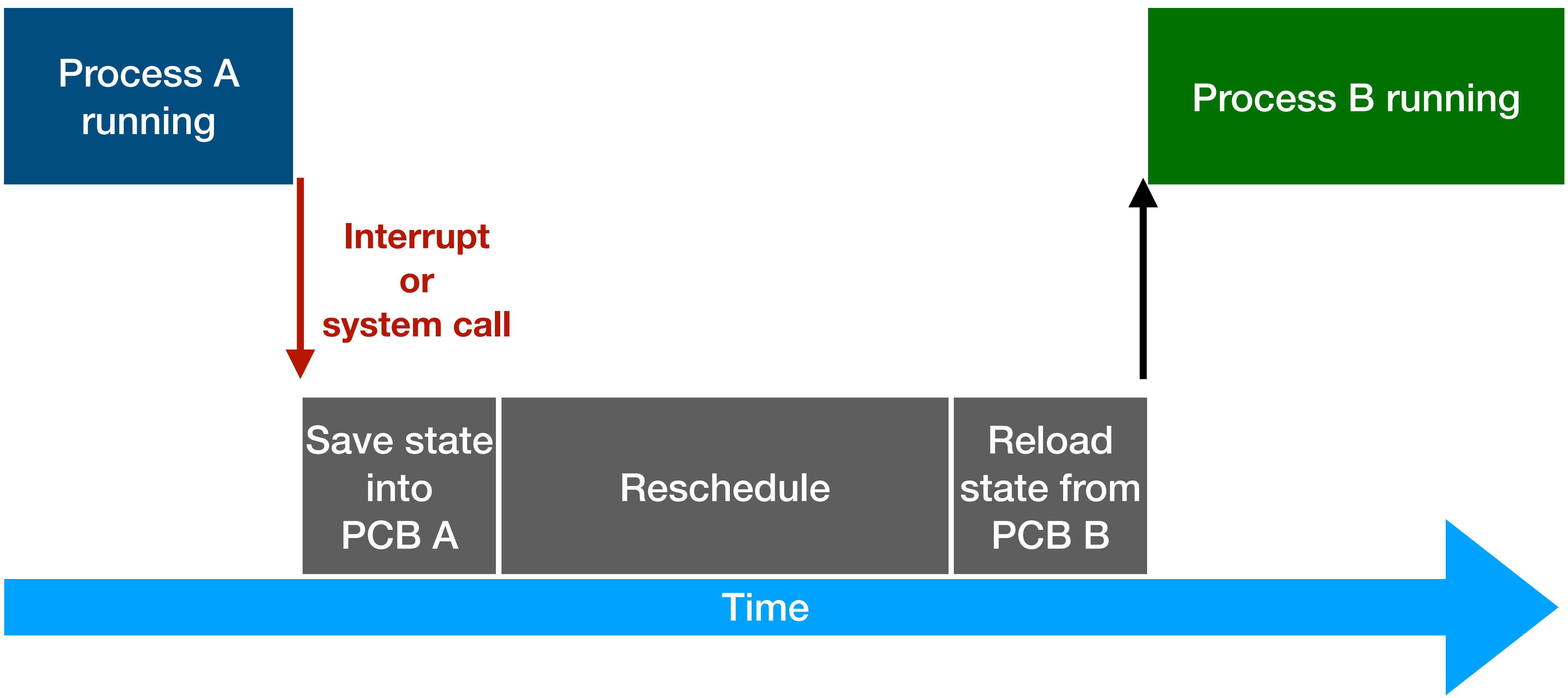
- When the CPU switches to a different process, the system must **save the state of the old process** and **load the saved state of the new process**. This operation is called a **context switch**.

- The process context is stored in the Process Control Block (PCB).

  - Save in memory the values in general-purpose registers, program counter (PC), stack pointer (SP), and other CPU-specific registers.

  - Update accounting information such as CPU usage, execution time, and other resource usage statistics.

Context switch



Impact of context switch overhead

- Context-switch time is pure overhead since the system performs no productive work during the switch.
- The complexity of the operating system and the size of the PCB directly affect the duration of a context switch.

Balancing context switching

- To maximize CPU utilisation, the operating system must strike a balance in the frequency of context switching:

    - **Too frequent switching:** Excessive context switching increases overhead because the system spends more time saving and loading process states rather than executing useful work.

    - **Too infrequent switching:** Rare context switching may cause longer response times and underutilize the CPU, especially in time-sharing systems where multiple processes compete for CPU time.

- An optimal balance ensures efficient CPU utilisation while maintaining responsiveness and fair process scheduling.

Schedulers overview

- **Short-Term Scheduler** (CPU Scheduler):
  - Selects the next process to execute and allocates the CPU.
  - Often the only scheduler in simpler systems.
  - Invoked frequently (in the scale of milliseconds), so it must operate quickly to minimize overhead.
- **Long-Term Scheduler** (Job Scheduler):
  - Determines which processes to admit into the ready queue.
  - Invoked infrequently (every few seconds or minutes), allowing it to operate more slowly.

Schedulers overview

- **Medium-Term Scheduler**
  - Temporarily removes processes from memory to disk (known as swapping) to reduce memory load and improve CPU scheduling efficiency.
    - **Swap out:** Moves processes from the ready or waiting state to a suspended state (from memory to disk) to free up system resources.
    - **Swap in:** Restores suspended processes back to the ready queue (from disk to memory) when sufficient resources are available.
  - Invoked as needed, typically in response to high memory demand or resource contention.
  - **Disk thrashing** occurs when excessive swapping causes the OS to spend more time accessing the disk than executing processes, leading to a significant decline in system performance.