Princípios da Computação

Operating systems: reactive software by nature.

OS: goals and expectations

isep Instituto Superior de Engenharia do Porto

General goals of an operating system

- Efficient resource management
- Reliable system operation
- Support for multitasking and concurrency
- Security and stability
- **Focus on balancing user and system needs.**

OS: Resource manager and controller

- **Resource Allocator:**
    - Manages all system resources.
    - Resolves conflicting requests to ensure efficient and fair resource usage.
- **Control Program:**
    - Oversees program execution to prevent errors.
    - Ensures the system is used correctly and securely.

User oriented functions

- **User interface:**
    - Provide users the access to OS functionality: CLI, GUI.
- **Program execution:**
    - Load and run applications seamlessly.
    - Provide multitasking and responsiveness.
- **File system management:**
    - Structured storage and retrieval of data.

System oriented functions

- **Resource allocation:**
    - Dynamic CPU, memory, and I/O scheduling.
    - Fair and efficient use of hardware.
- **Process Management:**
    - Handling concurrent tasks and process prioritisation.
    - Deadlock detection and resolution.

System oriented functions

- **Protection and security:**
    - Safeguard hardware and the OS by enforcing controlled access to resources.
    - Prevent programs from interfering with one another, ensuring they cannot corrupt each other’s code or data.
    - Require user authentication to grant authorised access to system resources.
    - Block unauthorised external access attempts.

System oriented functions

- **Accounting:**
    - Record the type and amount of resources used by each user.
    - Enable user billing based on resource consumption (e.g. cloud services).
    - Provide usage statistics to support system reconfiguration and optimisation.

How is it done?

The kernel

- The kernel is the central component that directly interacts with hardware and oversees all low-level operations.

- Initiates system operations by being the first program loaded into memory.

- It has full access to hardware and critical system functions.

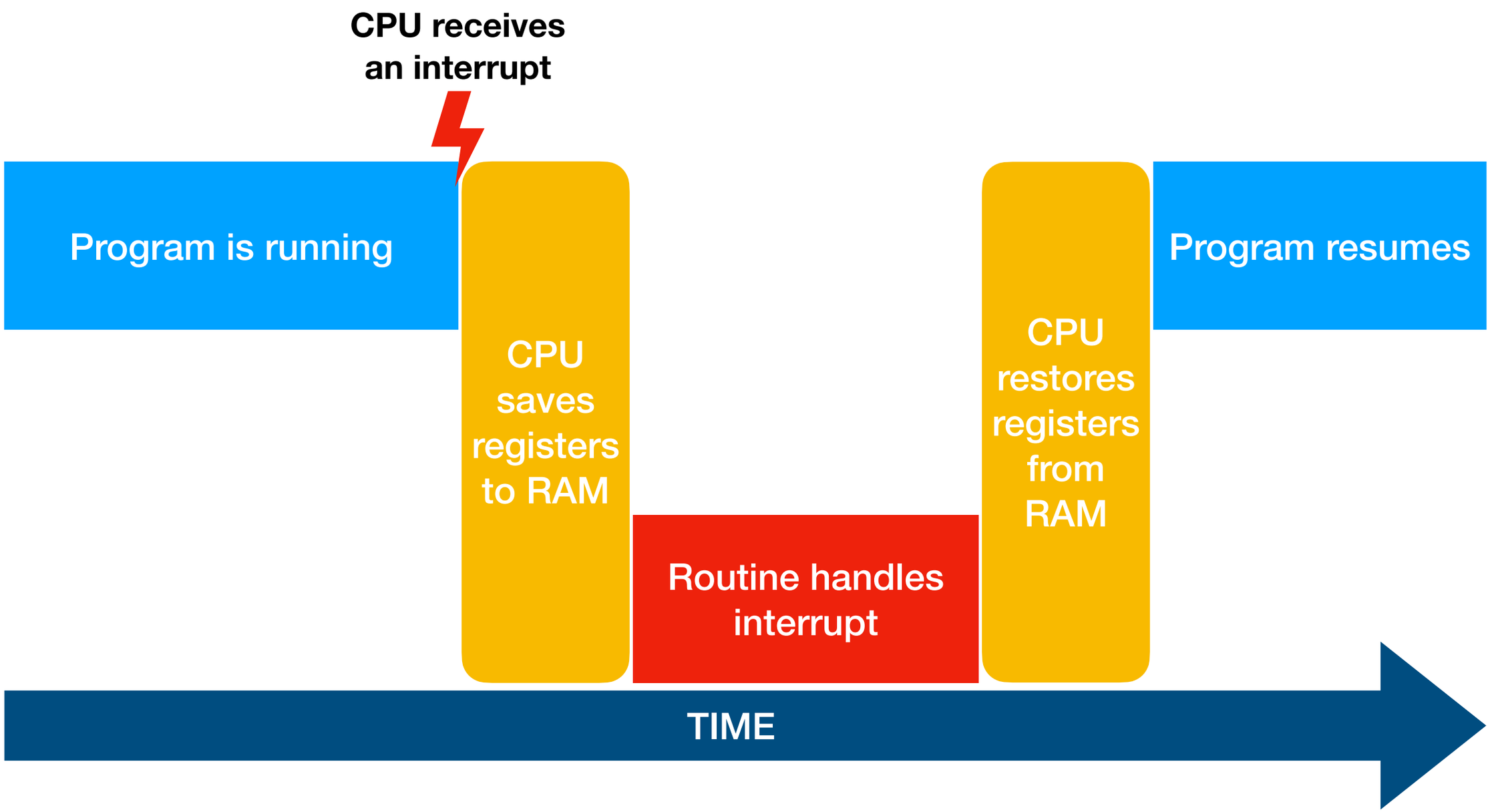
Kernel: reactive and asynchronous software

- The OS continuously waits for events (e.g., I/O requests, user inputs, interrupts).
- It executes routines only when triggered by these events.
- Events occur unpredictably and are handled immediately or queued for processing.

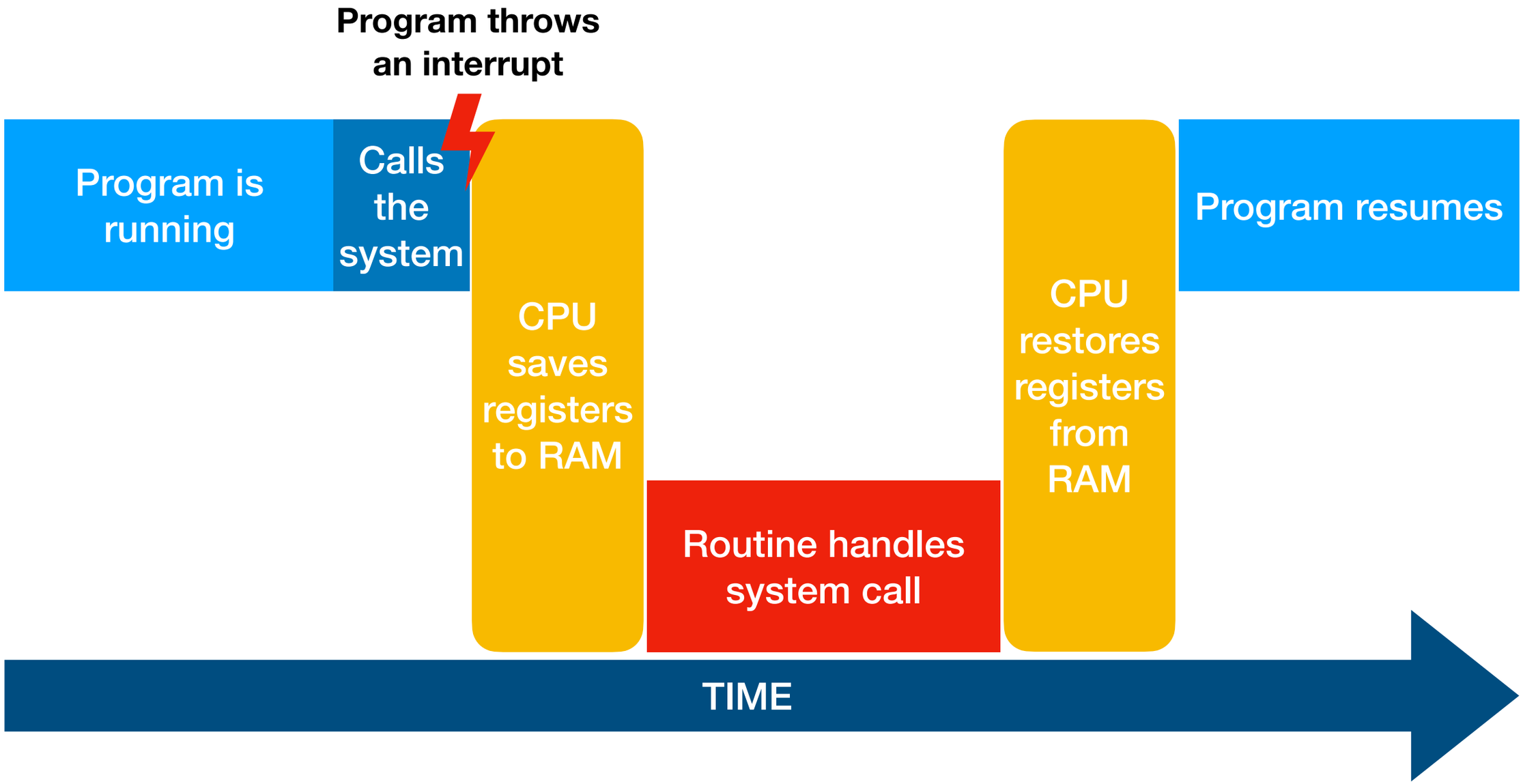
Calling the kernel

- **Interrupts:** signals from hardware or software that require immediate attention.
    - Examples: Timers, keyboard input, network packets, hardware failures.
- **System Calls:** mechanisms for applications to request OS services, triggering event-handling routines.
    - Often involve generating an interrupt to transition to a kernel-handling routine.

Interrupt


CPU receives an interrupt
Program is running
CPU saves registers to RAM
Routine handles interrupt
CPU restores registers from RAM
Program resumes
TIME

System call


Program throws an interrupt
Program is running
Calls the system
CPU saves registers to RAM
Routine handles system call
CPU restores registers from RAM
Program resumes
TIME

Kernel mode vs. User mode

- **Kernel Mode:** Full access to hardware and critical system resources.

  - Access to the full instruction set; access to all memory!

- **User Mode:** Restricted access to prevent applications from directly interfering with the system.

  - Access only to non-privileged instructions; access only to allocated memory.

Kernel mode vs. User mode

- These modes are enforced by the processor to ensure system stability and security.
    - The kernel runs in **kernel mode** and has unrestricted access to system resources, allowing it to perform all necessary operations.
    - Applications run in **user mode**, where access to critical resources is restricted to prevent unsafe operations that could compromise the system.
- Mode switching occurs through interrupts, ensuring safe transitions between user and kernel operations.

System call revisited