Operating Systems

Lecture 2:

PROCESSES

Lecture Topics

- Process Concept
- Process States and Scheduling
- Operations on Processes
- Cooperating Processes
- Threads
- Inter-process Communication

Process Concepts

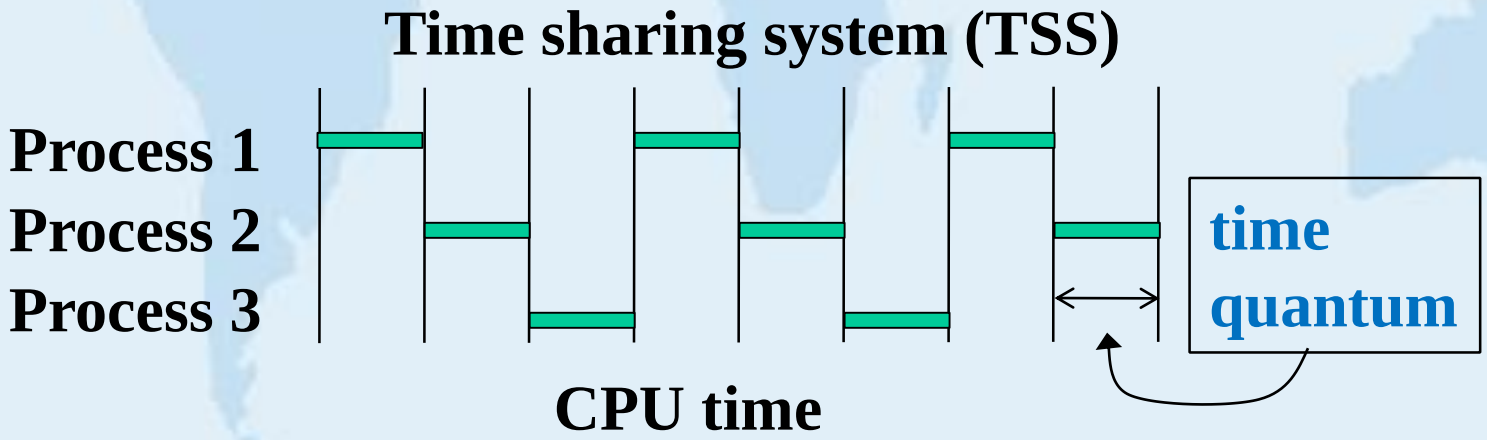
- **A Process (or Job)** is a program in execution. It is more than a program code.
- **A process** includes:
  - **Text section** – program code.
  - The value of **program counter**.
  - The contents of the **processor's registers**.
  - The **process stack** – temporary data (return addresses, local variables, etc.)
  - A **data section** – global variables.

Process Management

- The **OS** is responsible for
  - Process **creation,**
  - Process **execution,**
  - Process **deletion/removing.**
- The **OS** also provides of mechanisms for
  - Process **Synchronization,**
  - Process **Communication,**
  - Deadlock **Handling.**

Process Management

- The **OS** decides which process should be executed next by the CPU.
- Each process usually runs for a **short** amount of time.
- In multiprogramming, It appears that many processes are running **concurrently**.

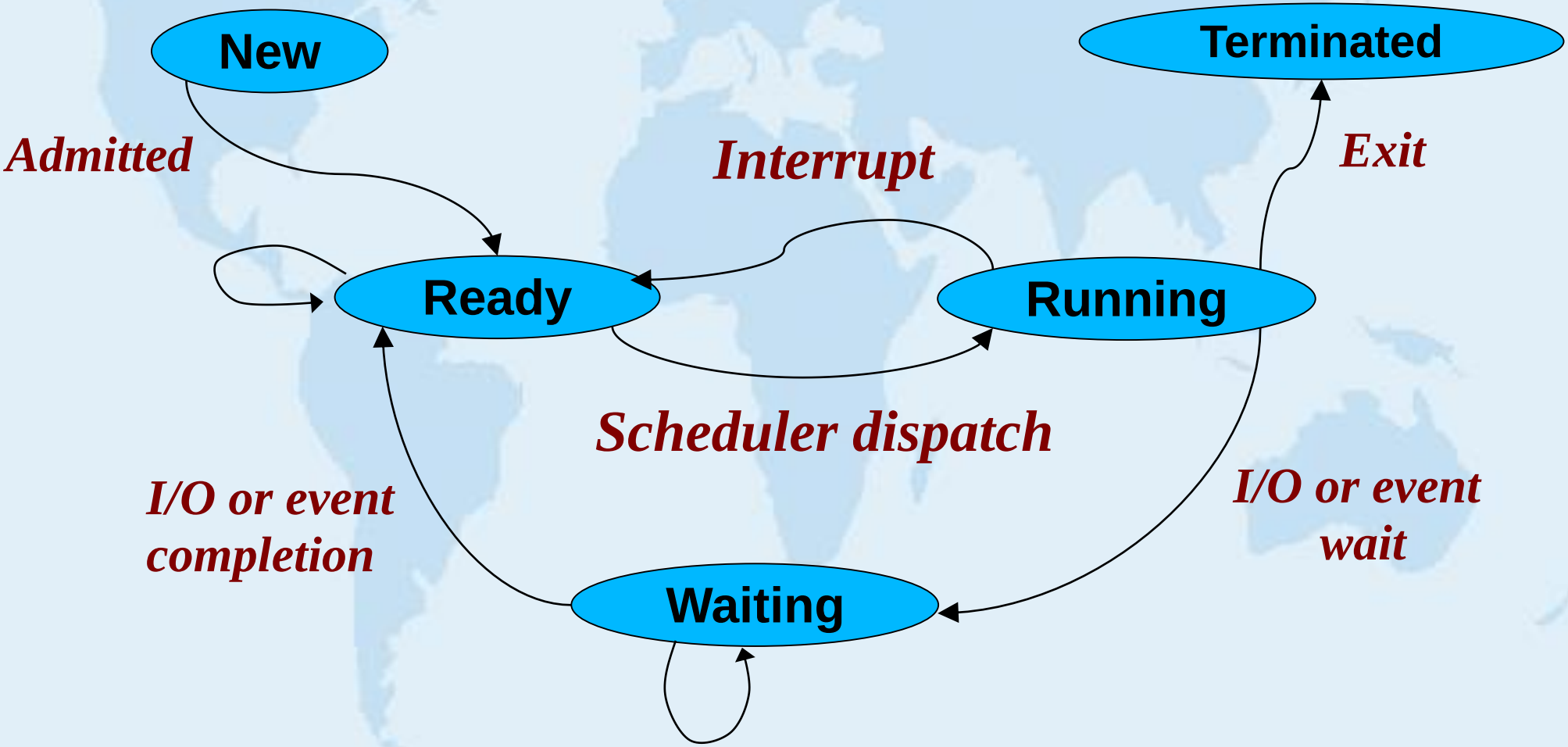


Process States

A **process** may be in one of the following states:

- **New**: The process is being created.
- **Running**: Instructions are being executed.
- **Waiting**: The process is waiting for some event to occur (such as an I/O completion or reception of a signal).
- **Ready**: The process is waiting to be assigned to a processor (CPU).
- **Terminated**: The process has finished execution.

Diagram of Process States

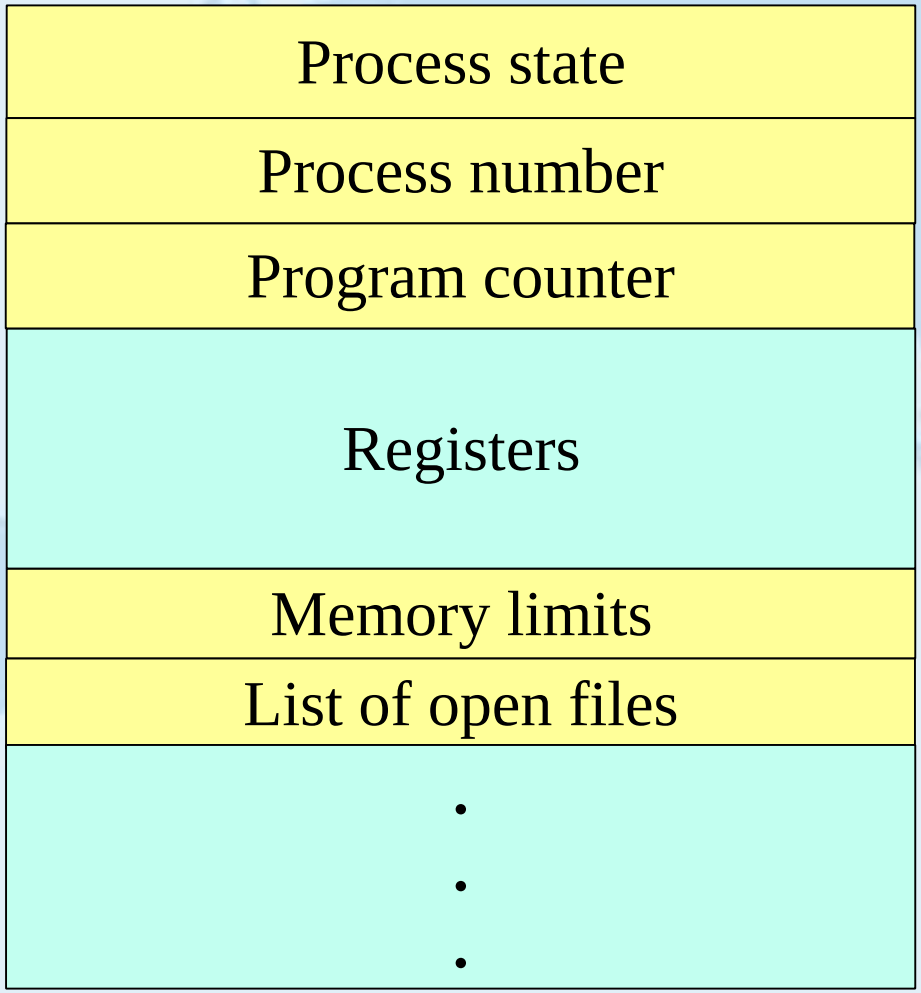


Process Control Block

Each process is represented in the **OS** by a **Process Control Block (PCB).** The information in PCB includes:

- ***Process state:*** ready, running, waiting, ...
- ***Program counter:*** next instruction address.
- ***CPU registers:*** accumulators, index, stack pointers.
- ***CPU scheduling information:*** process priority.
- ***Memory-management information:*** base address, memory limits.
- ***Accounting information:*** CPU time, job number.
- ***I/O status information:*** list of open files, etc.

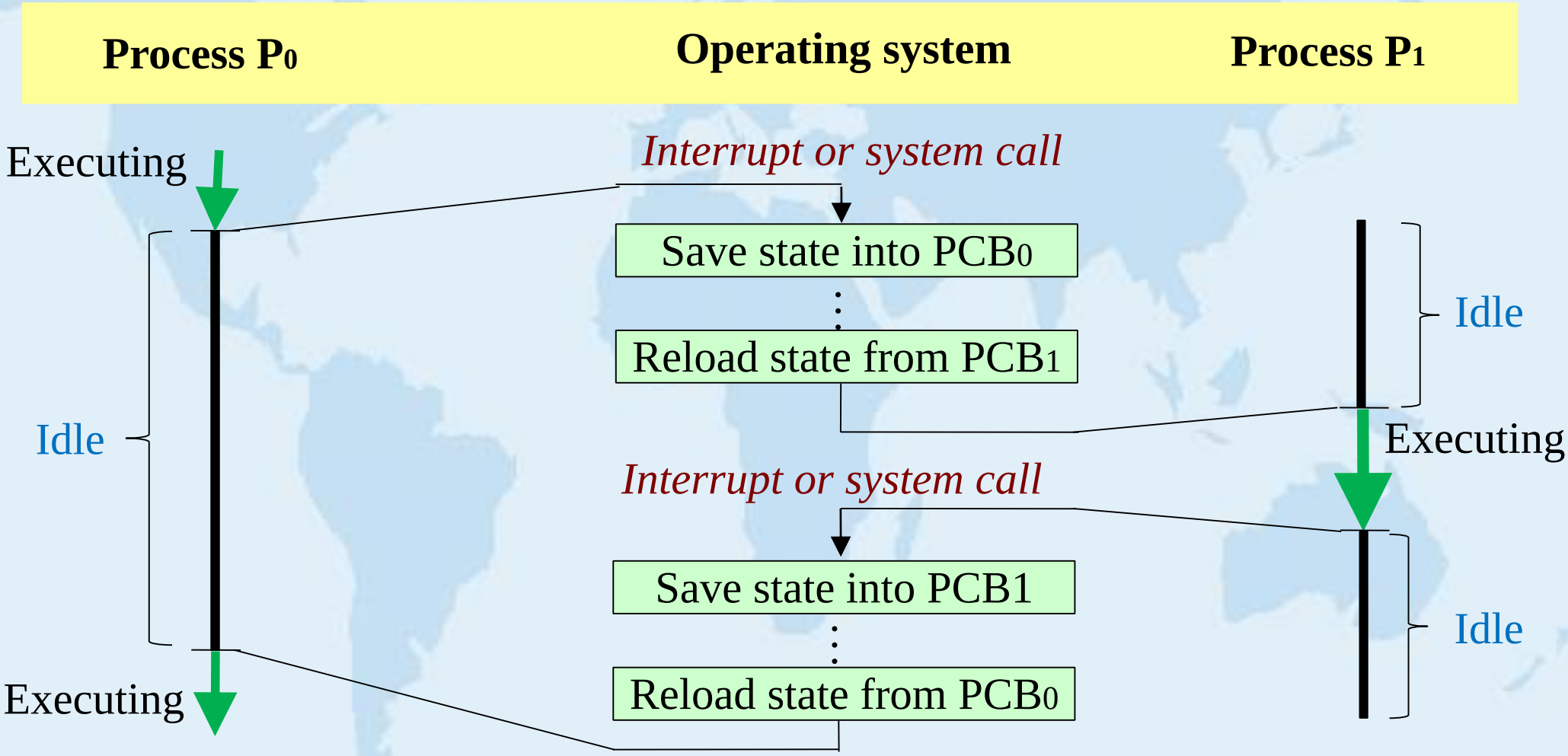
Process Control Block



Context Switch

- CPU switching from one process to another process requires:
  - Saving the state (PCB) of the **old** process,
  - Loading the saved state (PCB) of the **new** process.
- This task is known as **context switch**.
- Context-switch times are highly dependent on hardware support.

CPU Switch Diagram



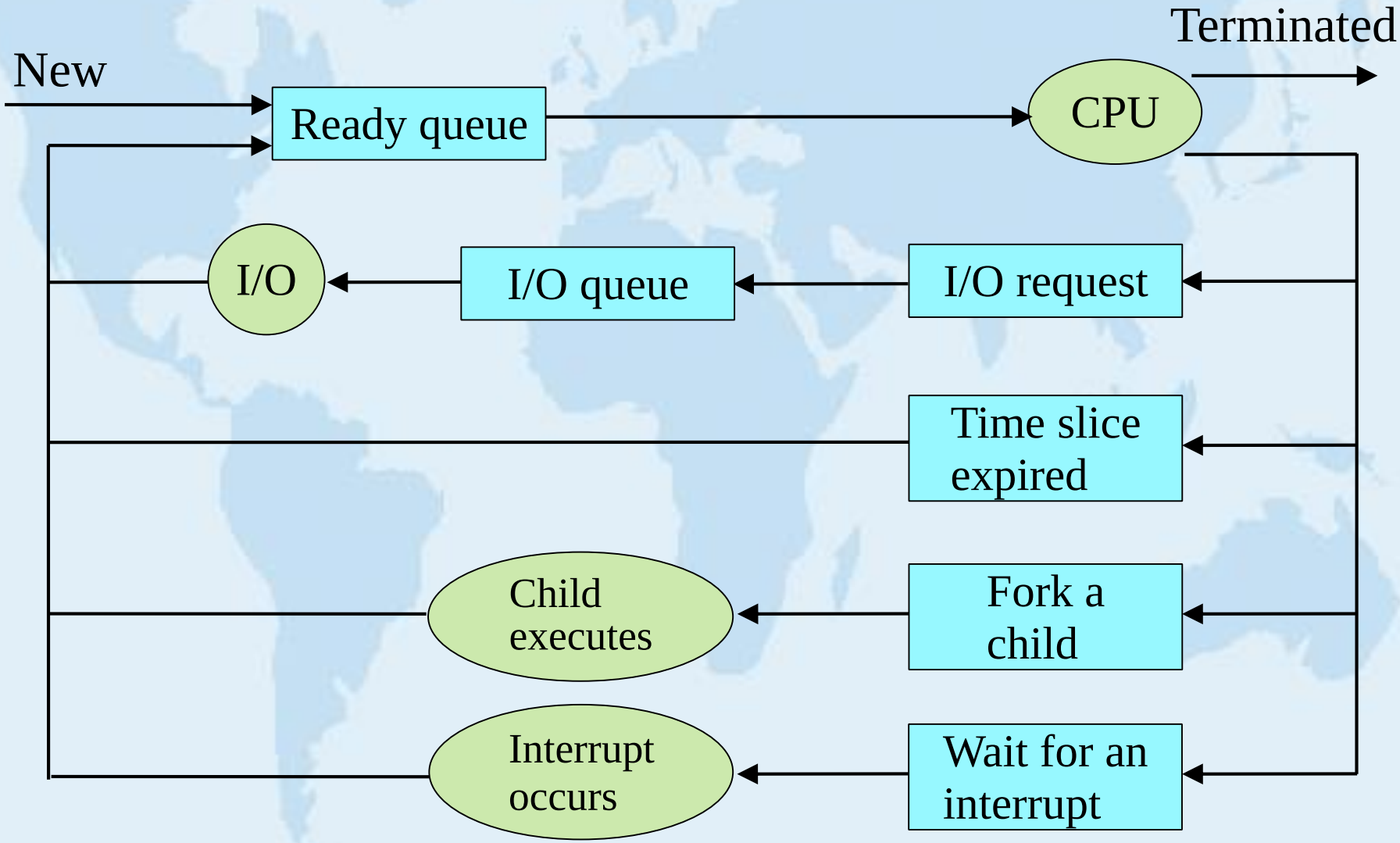
Process Scheduling

- Only **one** process can be in the running state.
- Other processes must **wait** until CPU is free and can be rescheduled.
- This means that we can have several processes in the **same** state. For each state, we have a **queue** of processes.
- The **selection** of a suitable process from the queue is called **scheduling**.

Scheduling queues

- Processes entered the system: **job queue**.
  - Job queue holds all processes in the system.
- Ready processes: **ready queue**.
  - Set of processes in memory waiting to execute.
- Processes waiting for a particular I/O devices: **device queue**.
  - Each device has its own queue.
- Process scheduling is represented by a **queuing diagram**.

Queuing diagram



Schedulers

- A process migrates between the various scheduling queues throughout its life time.
- For scheduling purposes, the **OS** must select processes from scheduling queues in some fashion.
- This selection process is carried out by an appropriate **scheduler.**

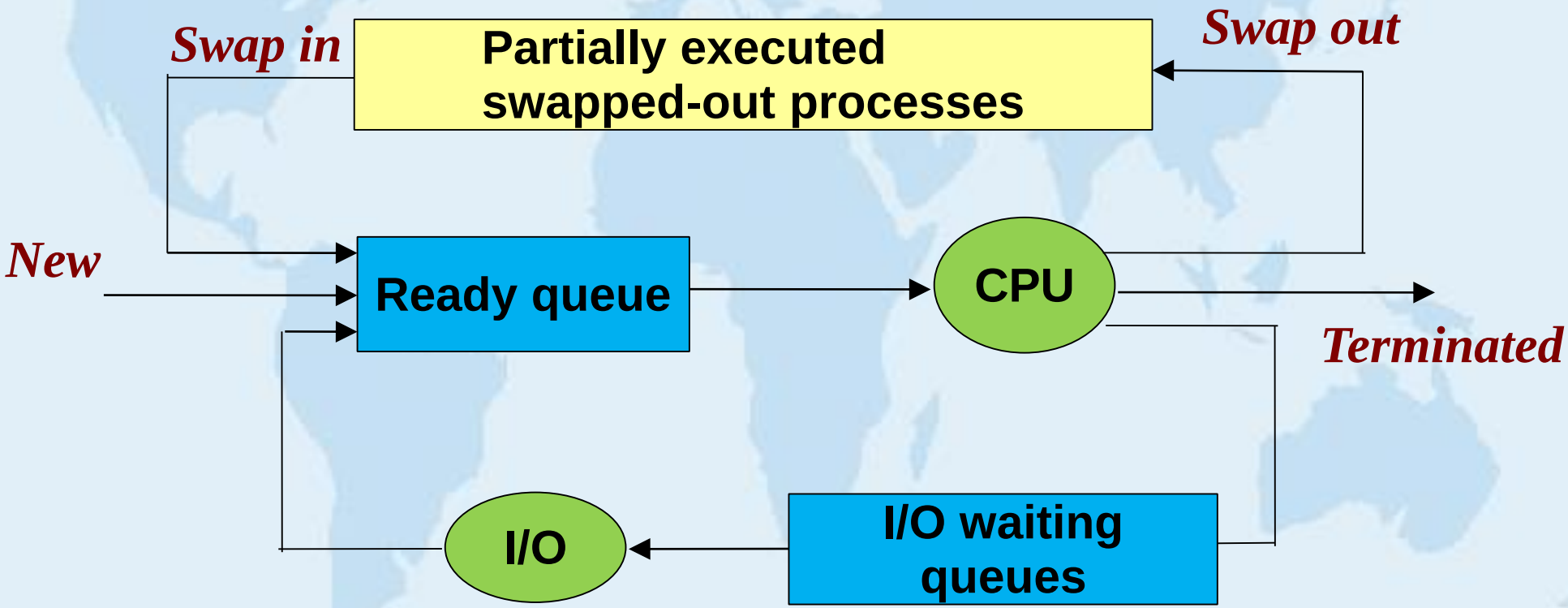
Schedulers

- **Short-term scheduler** or **CPU scheduler:** Selects a process from the ready queue and allocates it to the CPU.
- **Long-term scheduler** or **Job scheduler:** Selects processes from the mass-storage device (e.g., HD) and loads them into the memory for execution. Needs good selection (balance) between:
  - **I/O bound processes** which spend more time on I/O than computation, and
  - **CPU bound processes:** which spend more time on computation.

Medium-term Scheduler

- Sometimes it can be advantageous to **remove** processes from memory and reduce the degree of multiprogramming.
- At some later time, *removed process* can be **re-introduced** into the memory and its execution can be continued.
- This scheme is called **swapping** and it is performed by the **medium-term scheduler.**

Queuing diagram with medium-term scheduling

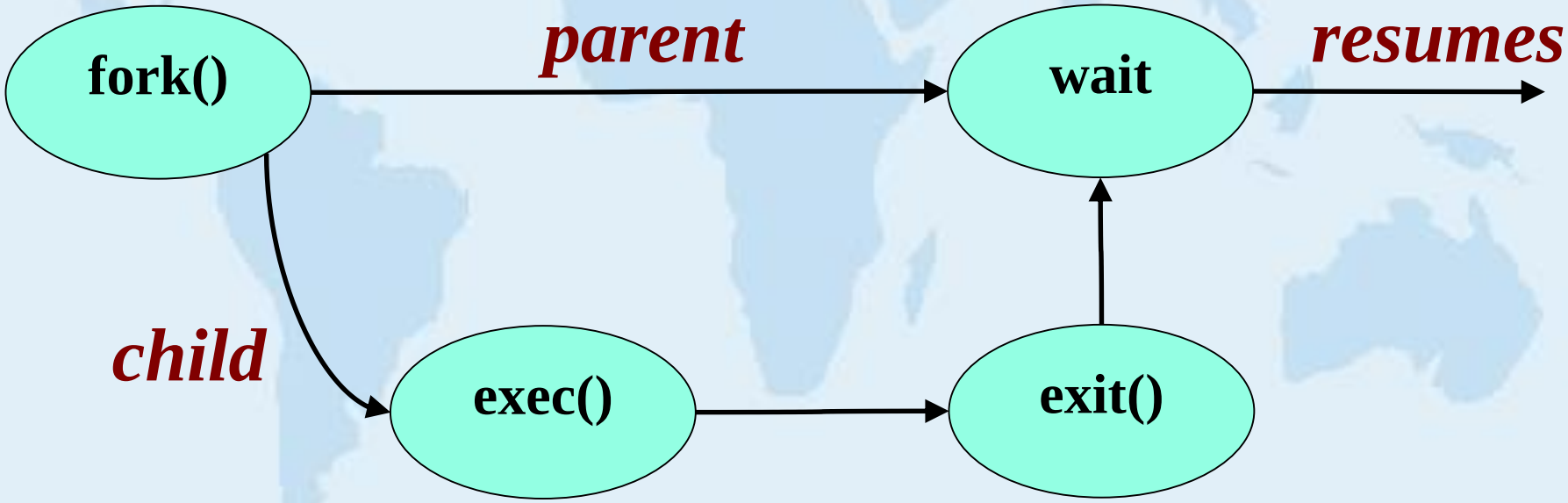


Process Creation

- Parent process creates child processes, which, in turn create other processes, forming a **tree** of processes.
- **Resource sharing:**
  - Parent and child processes share all resources.
  - Child processes share subset of parent's resources.
  - Parent and child processes share no resources.
- **Execution:**
  - Parent and child processes execute concurrently.
  - Parent waits until child process terminates.
- **Address space:**
  - Child process duplicates the parent's address space.
  - Child process has different program loaded into it.

Process Creation

- **UNIX examples**:
  - **Fork()** system call creates new process:



Process Termination

- Process executes last statement and asks the operating system to delete it **(exit)**.
  - Output data from child to parent (via **fork**).
  - Process' resources are de-allocated by operating system.
- Parent may terminate execution of child processes **(abort)** in following cases:
  - Child has exceeded its allocated resources.
  - Task assigned to the child is no longer required.
  - Parent is exiting.
    - Operating system *does not* allow child to continue if its parent terminates.
    - ***Cascading termination.***
    - UNIXes allow for orphaned child processes (init() becomes their parent).

Inter-process Communication

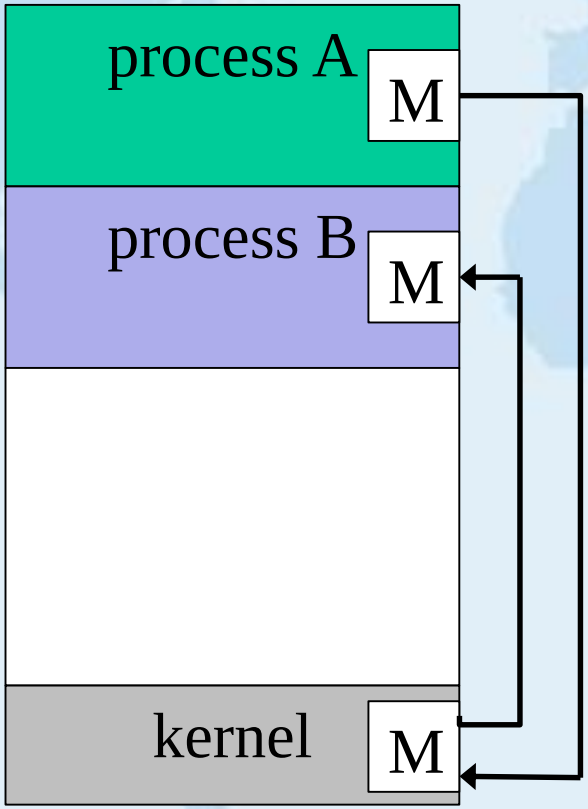
- **Independent process.** A process cannot affect or be affected by any other executing process.
- **Cooperating process.** A process can affect or be affected by other executing processes. A process that shares data with other processes is a cooperating process.

Inter-process Communication

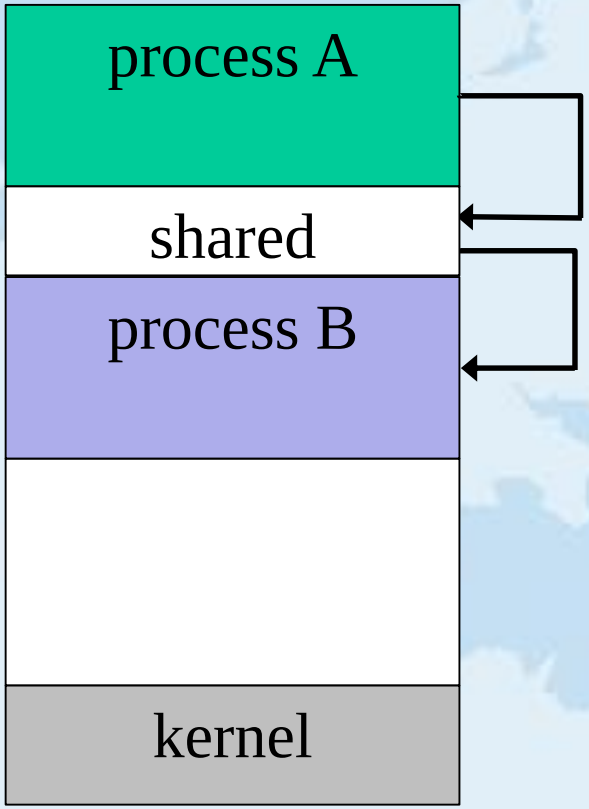
- Process cooperation is needed for:
  - **Information sharing** - several users are interested in the same piece of data.
  - **Computation speed up** – a task can be split into subtasks which can be executed in parallel on a computer with multiple processors or I/O channels.
  - **Modularity** – system functions are divided into separate processes or threads.
  - **Convenience** – serving multiple users / multitasking.

Inter-process Communication

Message passing



Shared memory



Communication models

Inter-process Communication

- **Message passing** communication model:
  - Useful for exchanging smaller amounts of data.
  - No need to avoid conflicts.
  - Easier to implement.
- **Shared memory** communication model:
  - Provides maximum speed and convenience.
  - No need for system calls (except for establishing the shared memory area).
  - Needs a method to avoid simultaneous access to the shared memory.

Message passing systems

To communicate, processes can use:

- **Direct Communication**
  - **send**(P, message). Send a message to process P.
  - **receive**(Q, message). Receive a message from process Q.
- **Indirect Communication**
  - **send**(A, message). Send a message to mailbox A.
  - **receive**(A, message). Receive a message from mail box A.

Synchronization

Communication takes place through the system calls **send()** and **receive().** Message passing may be either **blocking** (synchronous) or **nonblocking** (asynchronous):

- **Blocking send:** The sending process is blocked until the message is received by the receiving process or by the mailbox.
- **Non-blocking send:** The sending process sends the message and resumes operation.
- **Blocking receive:** The receiver blocks until a message is available.
- **Non-blocking receive:** The receiver retrieves either a valid message or a null.

Buffering

- If two processes communicate with each other, **a communication link** exists between them.
- A link has some **capacity** that determines the number of messages that can reside in it temporarily.
- Usually, it is called a **buffer.**

Buffering

Messages exchanged by the processes reside in a temporary queue. Such queues are implemented with:

- **Zero capacity**. The queue cannot have any message waiting in it. The sender must wait until the recipient receives the message.
- **Bounded capacity**. The queue has finite size ***n***, at most ***n*** messages can reside in it. The sender must wait if the queue is full and the receiver must wait if the queue is empty.
- **Unbounded capacity**. The queue has infinite size and any number of messages can wait in it. The sender is never delayed.

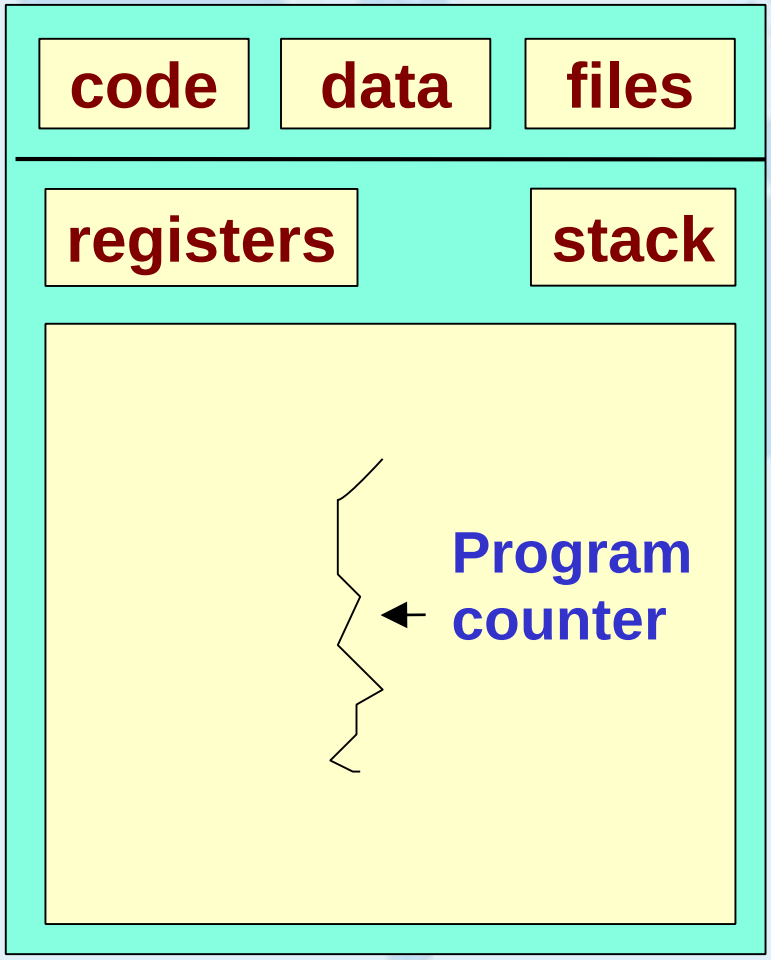
Threads

- Processes discussed above are also called **heavyweight processes**.
- We can decrease amount of data stored in *Process Control Blocks* (PCB) as well as the Context Switching Time allowing sub-processes inside a process.
- A **Thread** or **lightweight process** (LWP) is a sub-process inside a process.

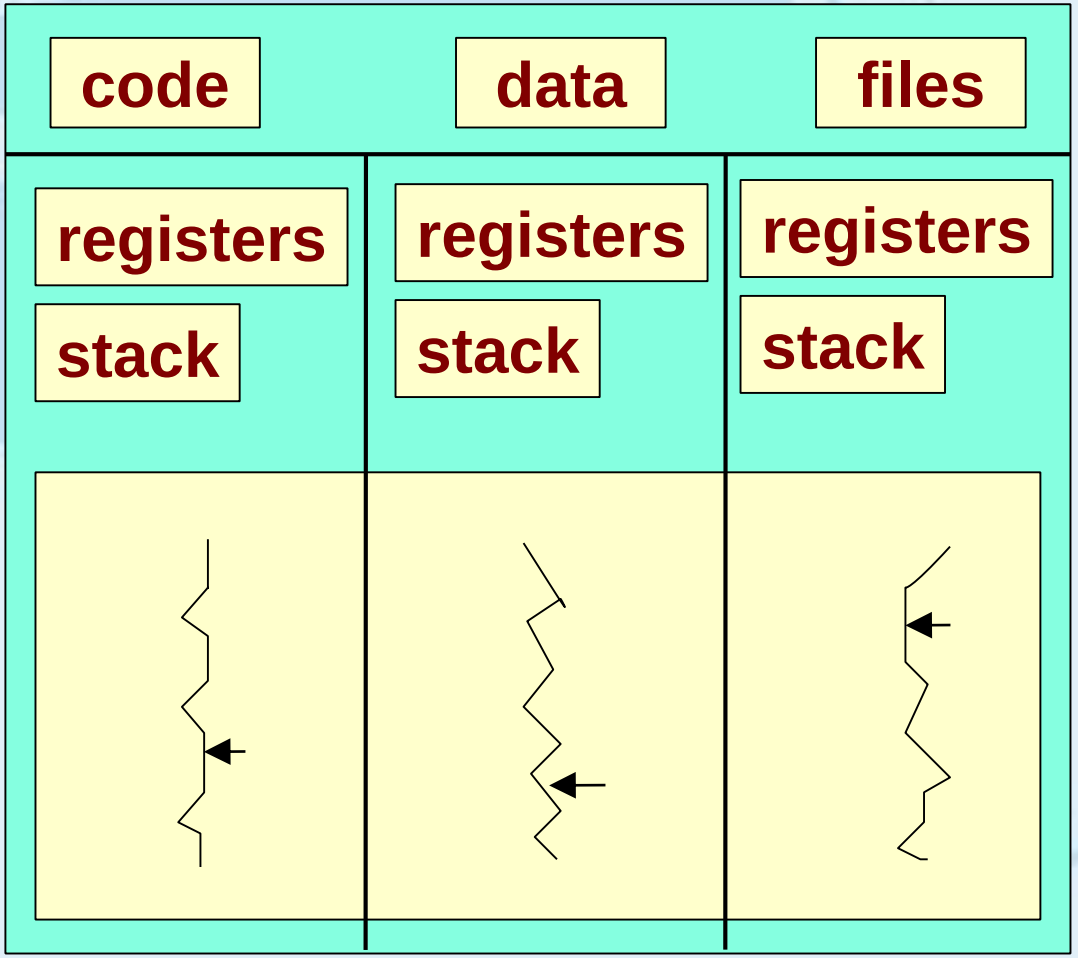
Threads

- Each thread has **separate:**
  - *program counter,*
  - *register set,*
  - *stack space.*
- It **shares** with peer threads its code section, data section, and OS resources such as open files and signals (collectively known as a task).
- A **heavyweight** process is equal to a job with one thread.

Single vs. Multiple Threads



Single-threaded process

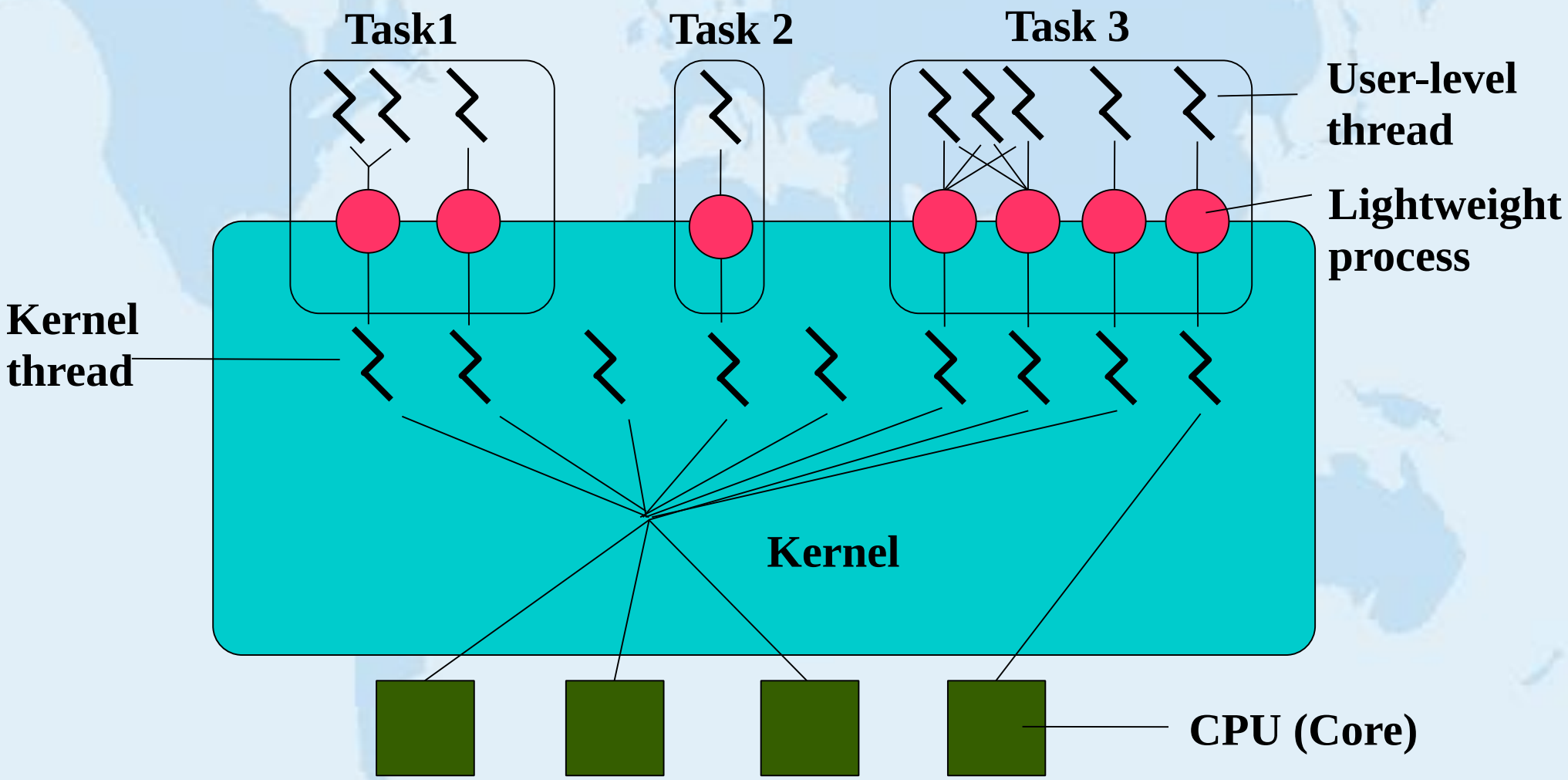


Multi-threaded process

Thread Scheduling

- CPU switching among peer threads and the creation of threads is **inexpensive**, compared with context switches among heavyweight processes.
- Although thread context switch still requires a register set switch, **no memory-management** related work needs to be done.
- Some systems implement **user-level threads** in user-level libraries, so thread switching does not need to call the OS, and to cause an interrupt to the kernel.

Threads in Solaris



That’s all for today!