Distributed Systems

Grade 4

Dr. Dalia Elwi

M.D, Ph.D.

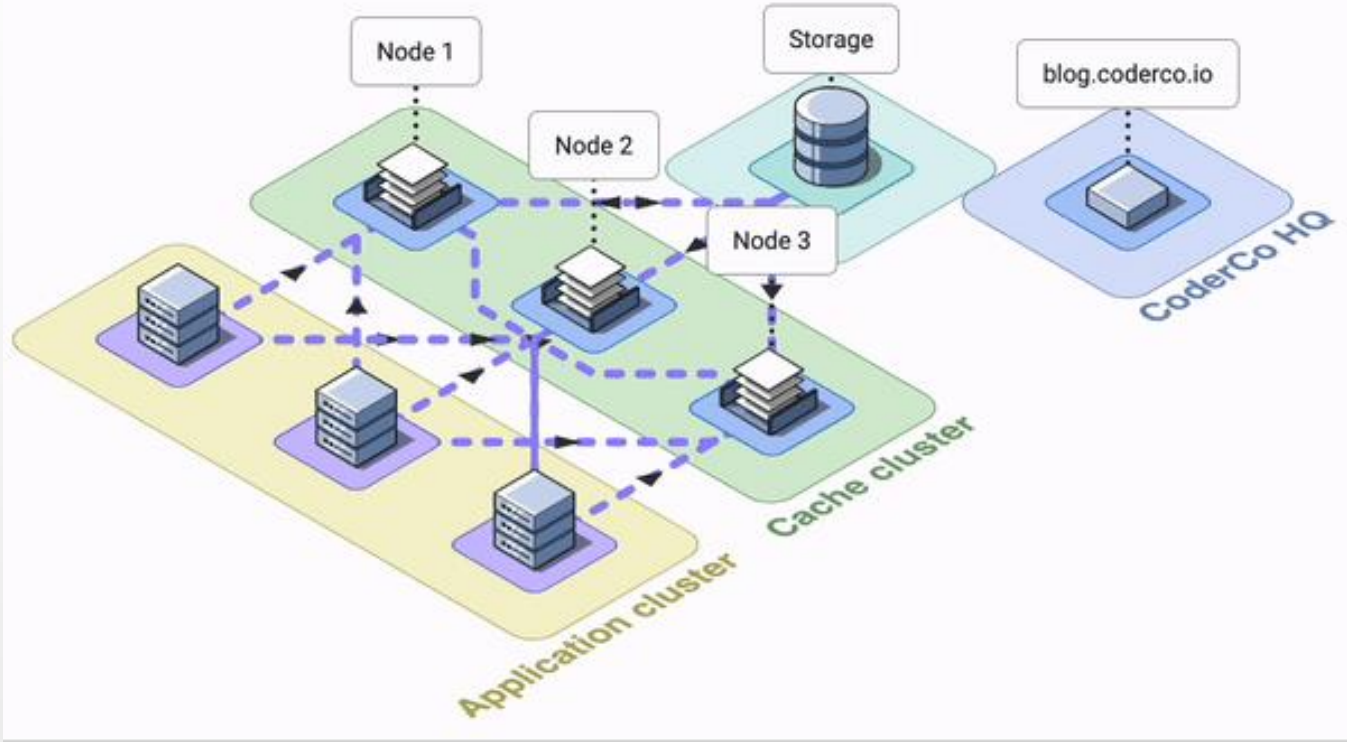
Lecturer of Computer Science

Faculty of Computer and information Science, MU

Dalia_elwi@mans.edu.eg

Distributed Systems

Lecture 3
Process and Thread



Program

- A **program** is a collection of instructions (code) a computer can execute to perform a specific task.
- A program remains passive on the storage disk or in nonvolatile memory until it is invoked by the user or another program, at this point, it becomes an active process in the system.
- There are many types of programs, including:

1. Operating system Programs,
2. User applications: include programs such as word processing, web browsing, or emailing a message to another computer.

Process vs Thread

- **Process**
  - An independent unit (an instance of a program) of execution with its own memory space.
  - Example: Running two instances of Chrome → two separate processes.
  - Each process has: code, data, and one or more threads.
- **Thread**
  - A lightweight unit of execution inside a process.
  - Threads **share** the process's memory and resources.
  - Example: Chrome using multiple threads → one for UI, one for networking.
- 👉 In distributed systems:
  - **Processes** provide isolation between applications.
  - **Threads** enable concurrency and parallelism within one application.

Features of Process

1. Unlike a program (Passive), a process is dynamic (Active), i.e., it can be created and deleted over time.
2. A process can be in various states, such as new, ready, running, waiting, or terminated.
3. The process is represented by PCB (**Process Control Block**), which contains the information of the process state.
4. Multiple processes can be executed concurrently.
5. It allows multitasking by saving the current execution process, which can be resumed later (Context Switch).
6. There can be multiple instances of a single program, and each instance of that running program is a process.
7. Each process has a separate memory address space, which means that a process runs independently and isolated from other processes. It cannot directly access data in other processes (it is not shared).

Program-Process Comparison

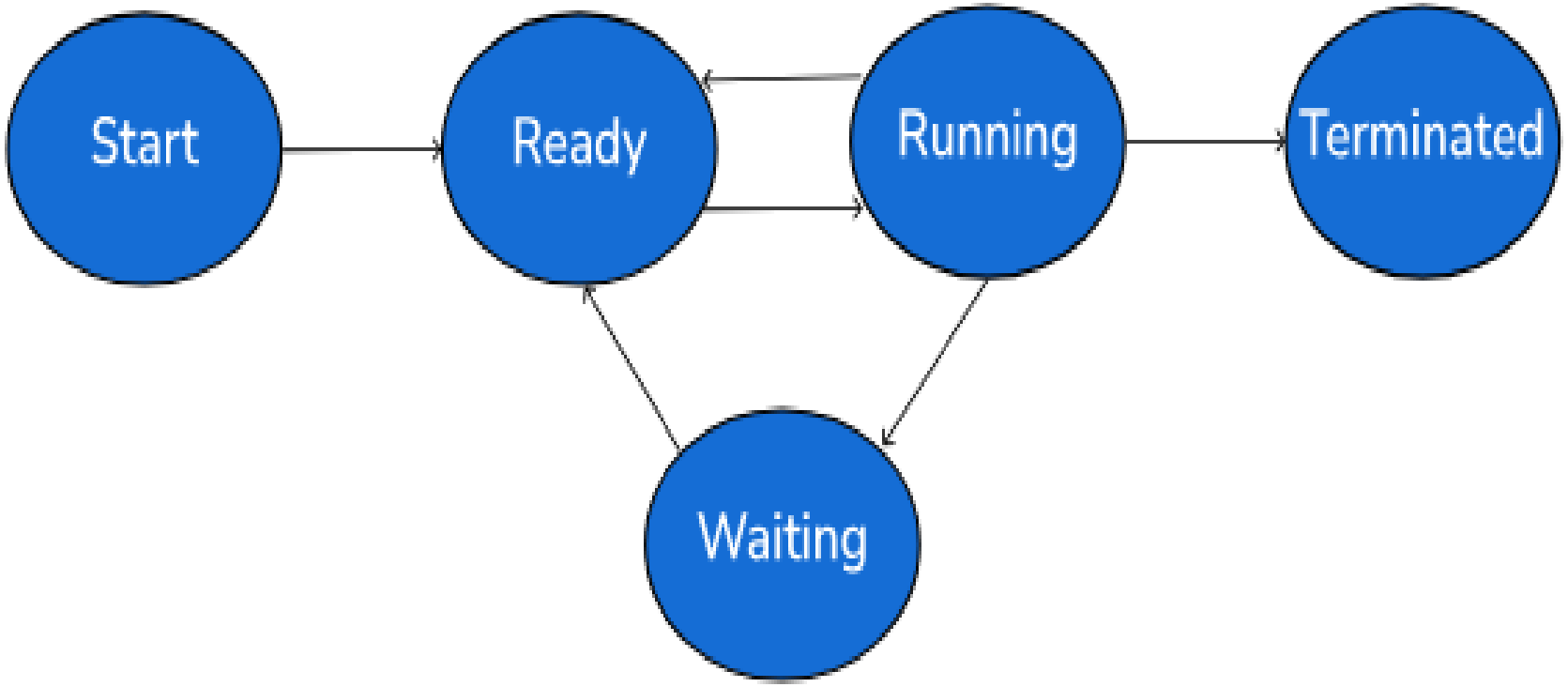
	Program	Process
Definition	Programs in OS are a static sequence of instructions stored in a file.	The process is an active instance of a program in execution.
Storage	Stored in Disk.	Reside in main memory during execution.
Execution	It can't execute by itself.	Executed by the system.
Resource Allocation	Doesn't have any resources.	Has resources like memory and CPU .
Instance	Single instance on disk.	Multiple instances can run concurrently.
Dependency	It can exist without a process.	It can't exist without a program.
Entity Type	Passive Entity	Active Entity
Interaction	Doesn't interact with other programming languages.	It can communicate with other programming languages with inter-process communication.
Lifecycle	If not modified, it remains for a lifetime.	It follows the steps such as ready, running, and terminated.

Process States and Lifecycle

A process goes through the following states:

1. **Start**: This is the initial state. A process is in this state on its creation.
2. **Ready**: The process is assigned 'Ready' state when it is ready to be processed. The process is moved from the 'Start' state to the 'Ready' state after it is moved into memory. The process can be moved here from the 'Running' state as well if some other process is moved to the 'Running' state.
3. **Running**: The process is moved to the 'Running' state when it is assigned to a processor and its processing starts. At a single point in time, a single process will be in the 'Running' state on a single processor.
4. **Waiting**: The process is moved to the 'Waiting' state if the process is waiting for any resource.
5. **Terminated**: The process is moved to the 'Terminated' state once its execution is completed or it is terminated by the OS.

Process States and Lifecycle



Process Control Block (PCB)

A process is internally represented by the operating system using a process control block (PCB). It is also known as Task Control Block.

A PCB contains the following information:

1. Process Identification (process id)
2. Process State
3. Process Control Information (Scheduling Information, Program Counter, Process Privileges, CPU Registers, etc.)

Context Switching

- A context switch happens when a process is moved out from the CPU, and another process is provided to the CPU. The process is generally paused and can be restored later.
- The process state is changed from 'Running' to 'Waiting' during a context switch.
- The major reasons for a context switch are:
    1. The process is doing I/O operation (Requires I/O processor instead of CPU)
    2. The process is waiting for a synchronization operation to complete
    3. The process is running for a long-time and may starve other processes
- Context Switching is an essential part of achieving multitasking.

Preemptive and Non-Preemptive Scheduling

Preemption is the act of temporarily interrupting an executing task with an intention to resume it later. This is done by schedulers through context switching.

There are different scheduling algorithms being either preemptive or non-preemptive.

- **Preemptive scheduling** algorithms preempt a low priority process with a high priority process based on some criteria.
- **Non-preemptive scheduling** algorithms don't preempt a process after the process enters the 'Running' state.

Process Scheduling Algorithms

The primary objective of a process scheduling algorithm is to maximize CPU Utilization by keeping it as busy as possible.

While trying to maximize CPU utilization, the algorithms are also supposed to have the following objectives:

- Fair allocation of CPU
- Enforce priorities, if present
- Prevent processes holding key resources
- Maximum Throughput
- Minimum Turnaround Time
- Minimum Waiting Time
- Minimum Response Time

Process Scheduling Algorithms

The following are the most important process scheduling algorithms:

- **First-Come-First-Serve (FCFS):** Scheduling is done according to the arrival time of the process. Easy to implement and understand but has a high average waiting time.
- **Shortest Job First (SJF):** Processes with the shortest (CPU burst) time are prioritized. The average waiting time is low but difficult to know the CPU burst time of the process in advance.
- **Shortest Remaining Time First (SRTF):** Preemptive version of SJF. Preempted when there is a job with lesser remaining time. Same pros and cons as SJF.
- **Round-Robin (RR):** Preemptive Algorithm where each process is provided with a fixed time after which it is preempted. It maintains a cyclical queue. Easy to implement and avoid starvation but requires a lot of context switches.
- **Priority-based:** Non-preemptive algorithm where each process is assigned a priority and the process with the highest priority is picked up. In the case of multiple processes with the same priority, FCFS is done on them. Gives priority to important processes but might lead to starvation for low-priority processes.

Process Synchronization

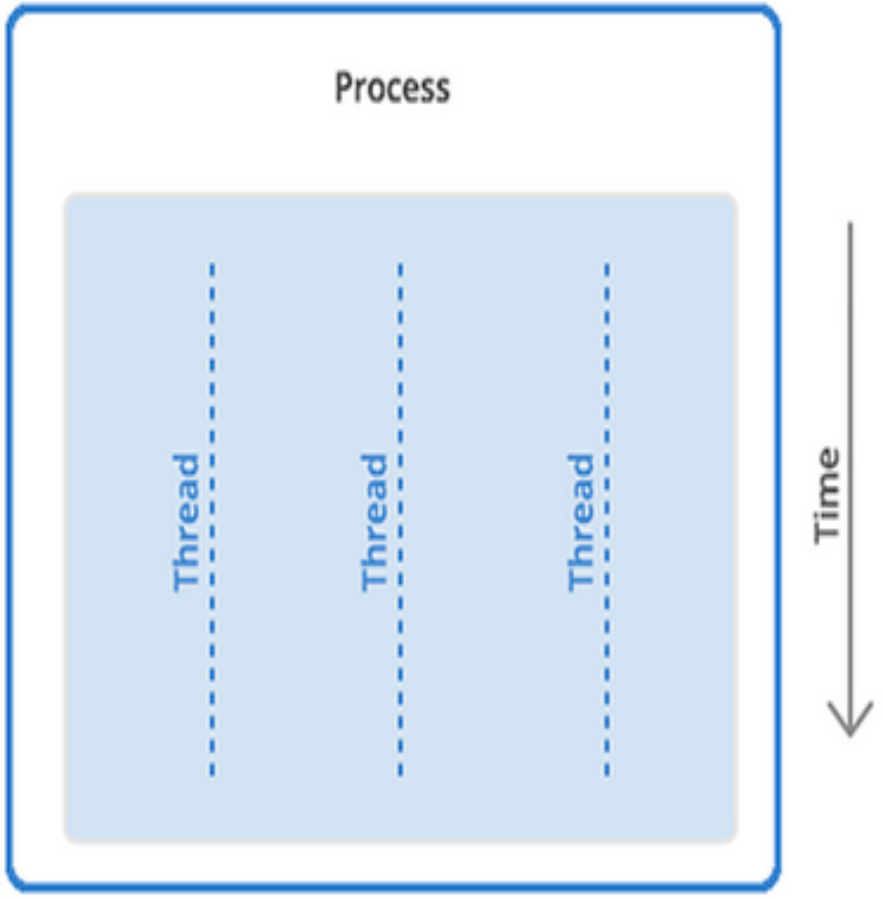
Processes can be categorized into two types:

- **Independent Processes:** The execution of one process does not affect the execution of another process.
- **Cooperative Processes:** The execution of one process affects the execution of another process.

Process Synchronization is the task of synchronizing cooperative processes such that the processes do not have access to shared data or resources **(Printer)** at the same time. This helps avoid inconsistency in data.

Threads

- A thread is the unit of execution within a process. A process can have anywhere from just one thread to many threads.
- Example: For instance, consider you have a program that is designed to do two things: download a file from the internet and write a text file on your computer. Without threading, your computer would first have to finish downloading before it could start writing. But with threading, your computer can perform both these actions concurrently
- If a single thread executes in a process, it is known as **a single-threaded** And if multiple threads execute simultaneously, then it is known as **multithreading.**



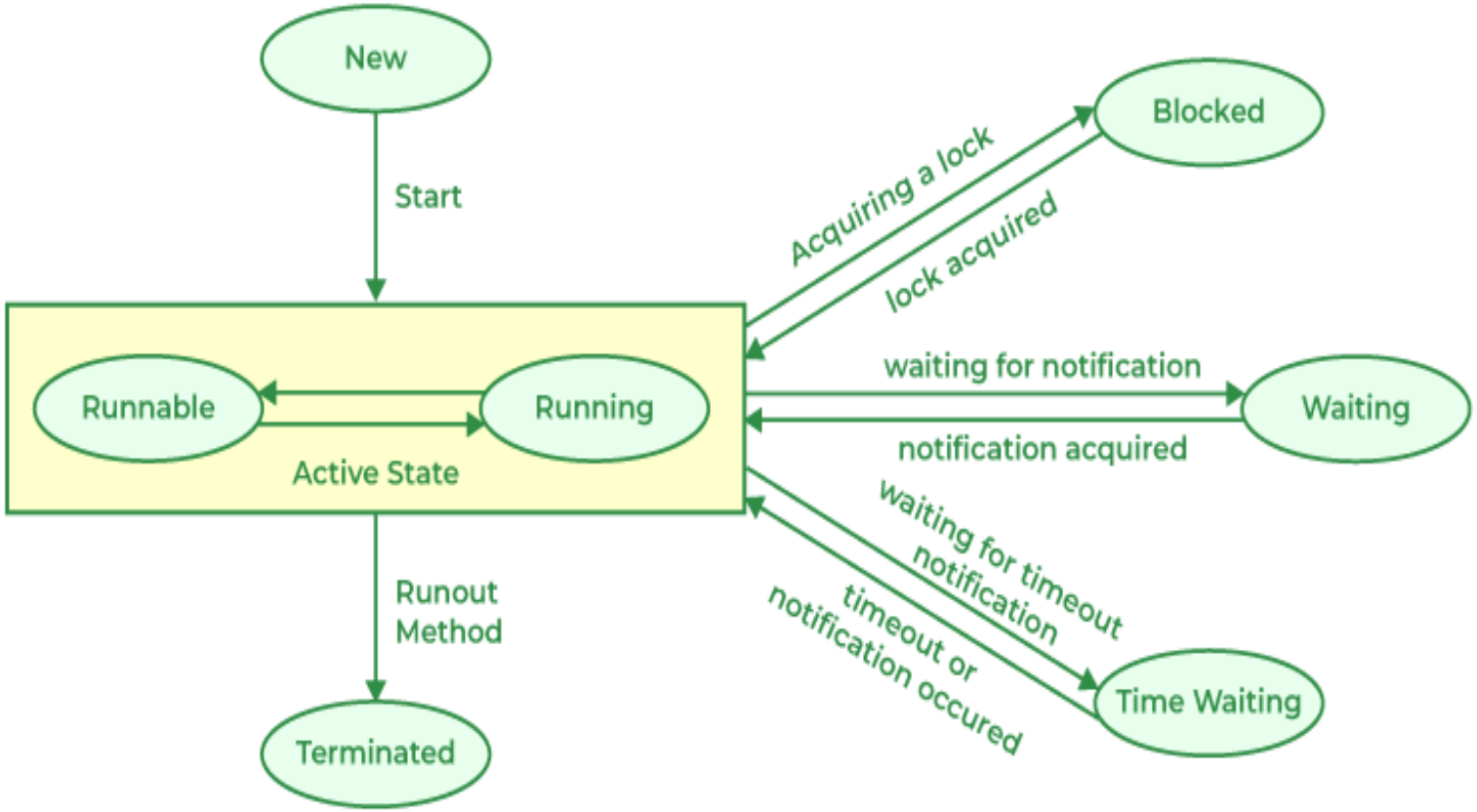
Concurrency and Parallelism

- **Concurrent** processing means that different tasks are progressing at the same time. This does not mean that they are running in **parallel**.
  - Concurrency works by **scheduling** different threads to make the maximum utilization of a single CPU (Multitasking).
- **Parallelism** means that different threads are being processed at the same time.
  - This can be done by distributing the different threads across **different CPUs** (Multiprocessing).
- We can achieve maximum performance by combining concurrency and parallelism in our applications.

States of Thread

- A thread lies only in one of the shown states at any instant:

1. New State
2. Runnable State
3. Blocked State
4. Waiting State
5. Timed Waiting State
6. Terminated State



States of Thread

1. **New Thread:**
    - When a new thread is created, it is in the new state. The thread has not yet started to run when the thread is in this state.
    - When a thread lies in the new state, its code is yet to be run and hasn't started to execute.
2. **Runnable State:**
    - A thread that is ready to run is moved to a runnable state.
    - In this state, a thread might be running, or it might be ready to run at any instant of time. It is the responsibility of the thread **scheduler** to give the thread, time to run.
    - A multi-threaded program allocates a fixed amount of time to each individual thread.
    - Each thread runs for a short while and then pauses and switches the CPU to another thread so that other threads can get a chance to run. When this happens, all such threads that are ready to run, waiting for the CPU and the currently running thread lie in a runnable state.
3. **Blocked:**
    - The thread will be in blocked state when it is trying to acquire a lock but currently the lock is acquired by the other thread.
    - The thread will move from the blocked state to runnable state when it acquires the lock.

States of Thread

4. **Waiting state:**
    - The thread will be in waiting state when it calls wait() method or join() method.
    - It will move to the runnable state when other thread will notify or that thread will be terminated.
5. **Timed Waiting:**
    - A thread lies in a timed waiting state when it calls a method with a time-out parameter.
    - A thread lies in this state until the timeout is completed or until a notification is received. For example, when a thread calls sleep or a conditional wait, it is moved to a timed waiting state.
6. **Terminated State:**
    - A thread terminates because of either of the following reasons:
        - Because it exits normally. This happens when the code of the thread has been entirely executed by the program.
        - Because there occurred some unusual erroneous event, like an unhandled exception.

Thread Priority

- In a Multithreading environment, thread scheduler assigns the processor to a thread based on the priority of the thread.
- Whenever we create a thread, it always has some priority assigned to it.
- Priority can either be given by machine while creating the thread or it can be given by programmer explicitly.

Virtualization in Distributed Systems

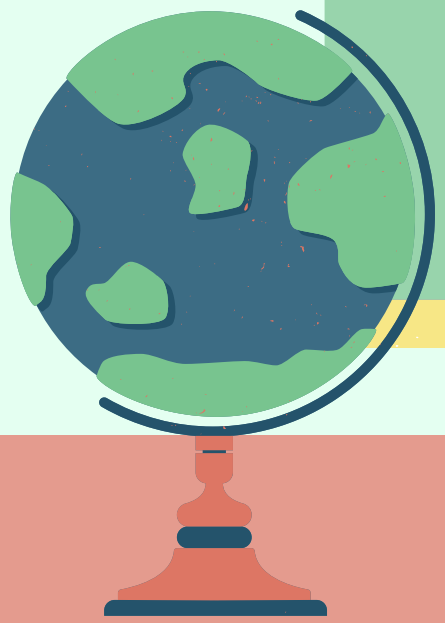
- **Virtualization:** abstraction of physical hardware into multiple virtual machines (VMs).
- Each VM runs its own OS and processes → strong **isolation**.
- Useful in distributed systems for:
  - Running multiple OS on one physical machine.
  - Migration of VMs between servers.
  - Fault isolation (a crash in one VM doesn't affect others).
- **Examples:** VMware, KVM, VirtualBox.

Containers in Distributed Systems

- **Containers:** lightweight virtualization at the **OS level**.
- Unlike VMs, containers share the same kernel but isolate processes.
- Faster, smaller than VMs.
- Useful in distributed systems for:
  - Microservices (each service in a container).
  - Scalability (containers can be scale up quickly).
- **Examples:** Docker, Podman.

Any Questions

??




Thank
You