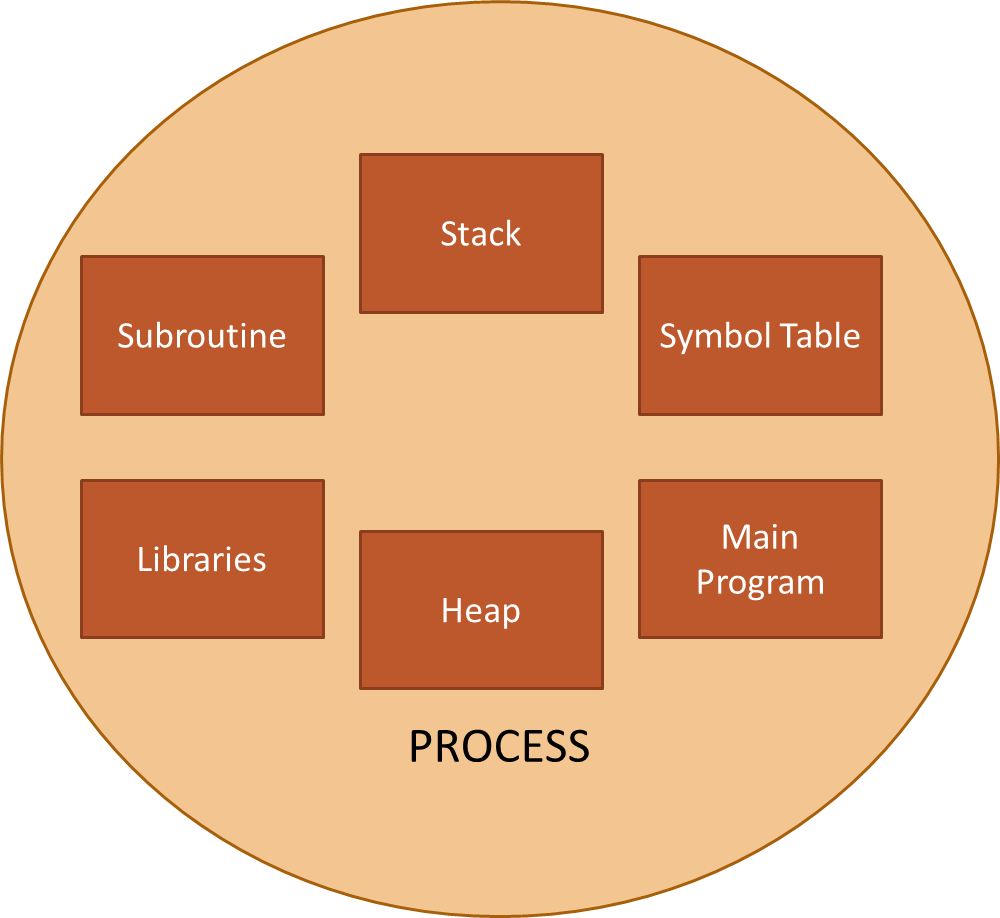
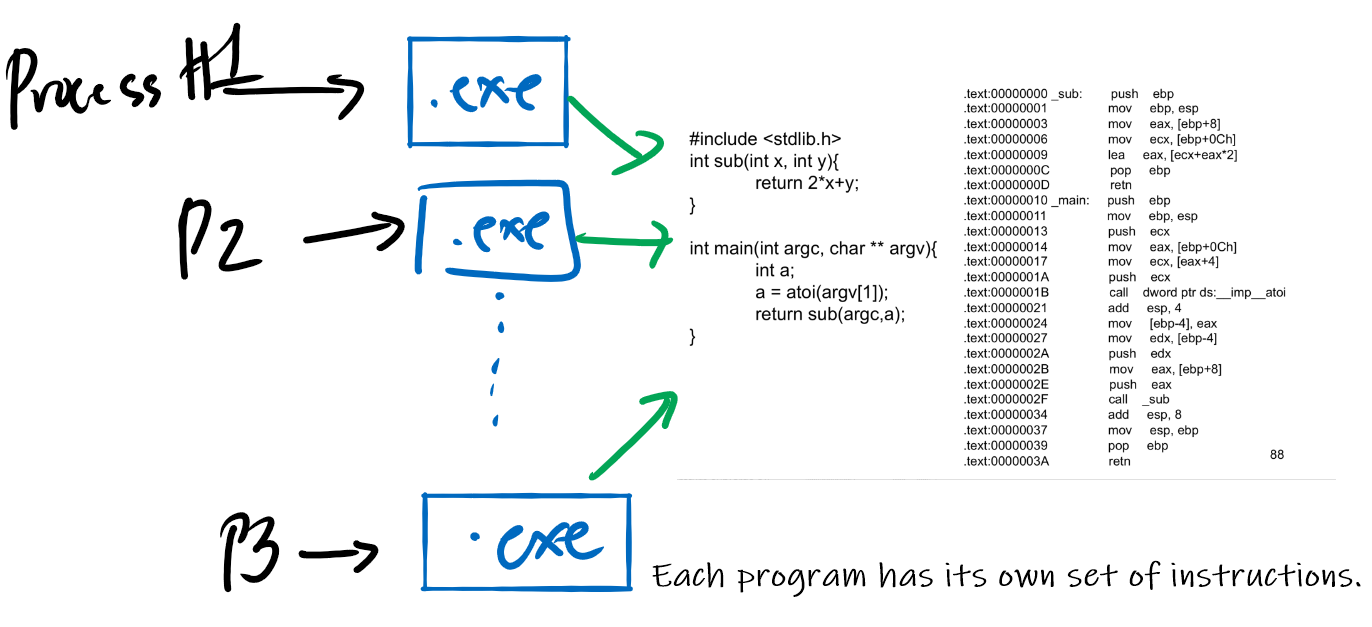
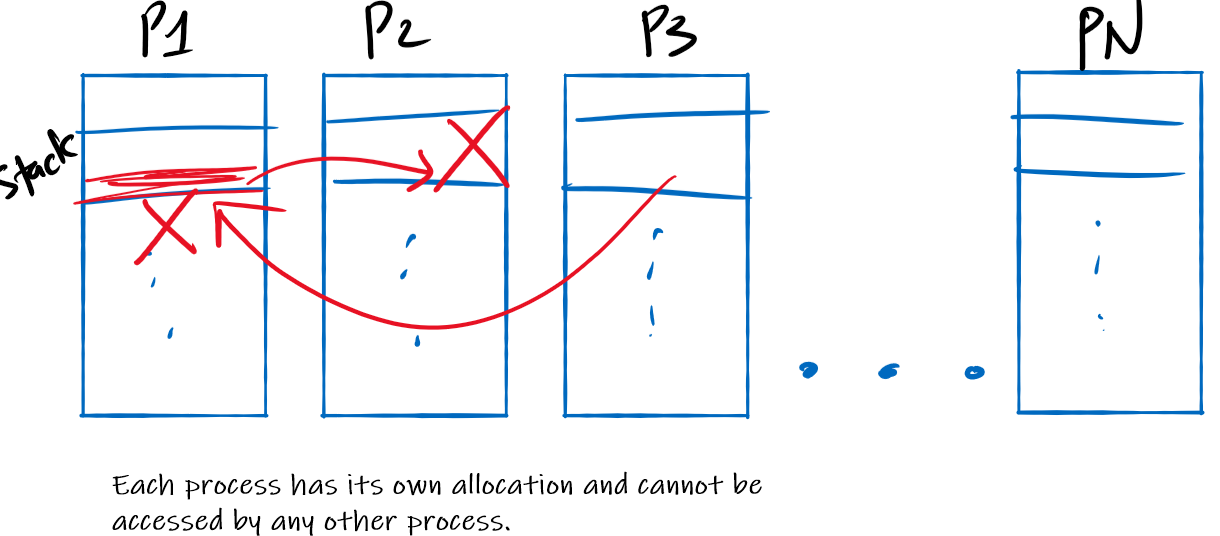
|  |  |
| --- | --- |
| **GDPARCM Lecture –Processes vs Threads** | Instructor: Neil Patrick Del Gallego |

**WHAT IS A PROCESS?**

* One program = one process.
* A standalone application that contains memory, stack, heap, etc.
* In C++, we use fork() method to create a duplicate process of the program.
* Context switch between the process is time consuming.
* We don’t typically practice developing programs using multiple processes.

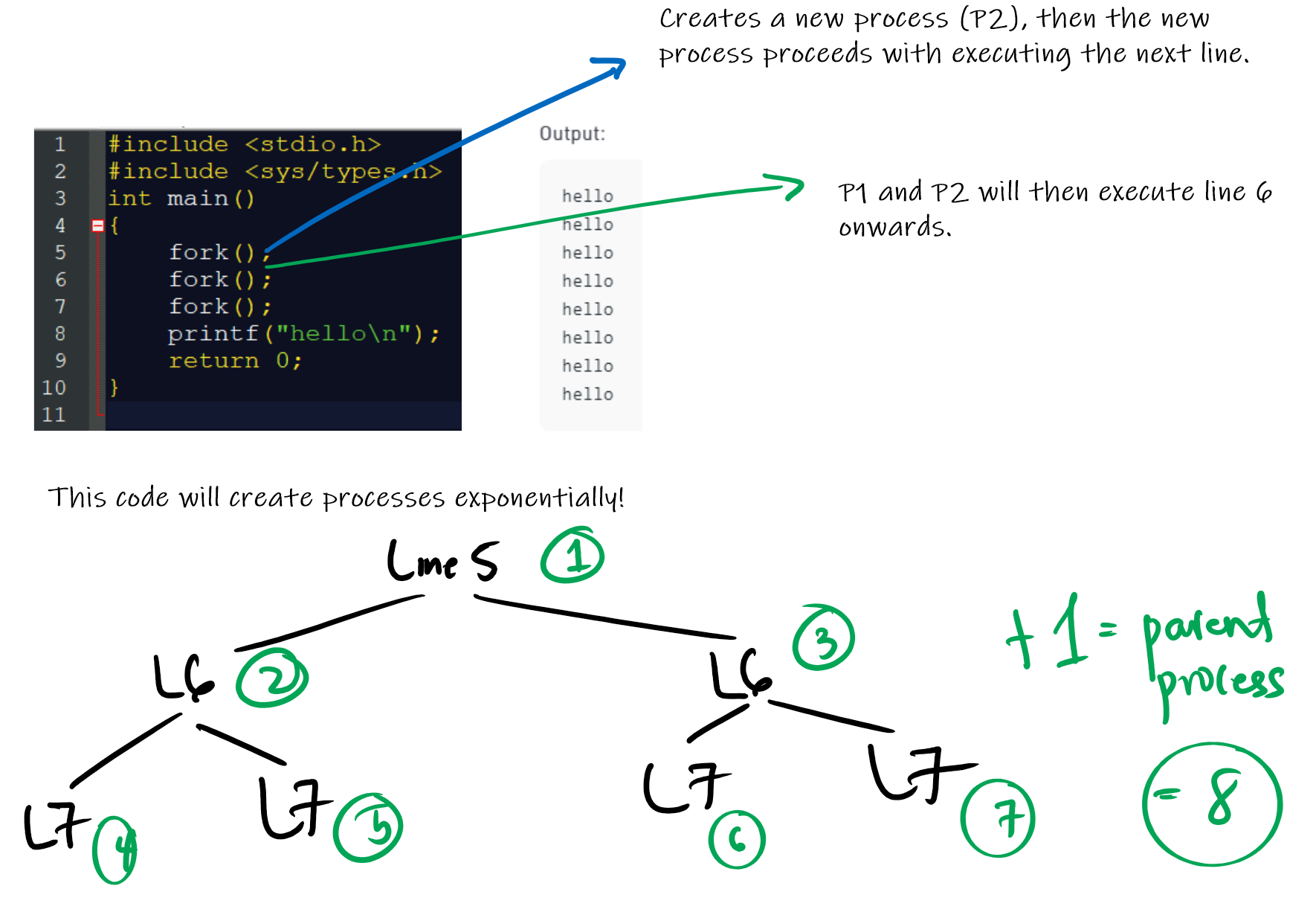




**Brief Example of Multi-Processing in C++**

Fork system call is used for creating a new process, which is called child process, which runs concurrently with the process that makes the fork() call (parent process). After a new child process is created, both processes will execute the next instruction following the fork() system call.

**NOTE:** Calling fork() is only applicable for UNIX systems! This is also why multiprocessing for games are rarely done, as game programs/engines are typically created in Windows systems.



**Another Example**

|  |  |
| --- | --- |
|  |  |

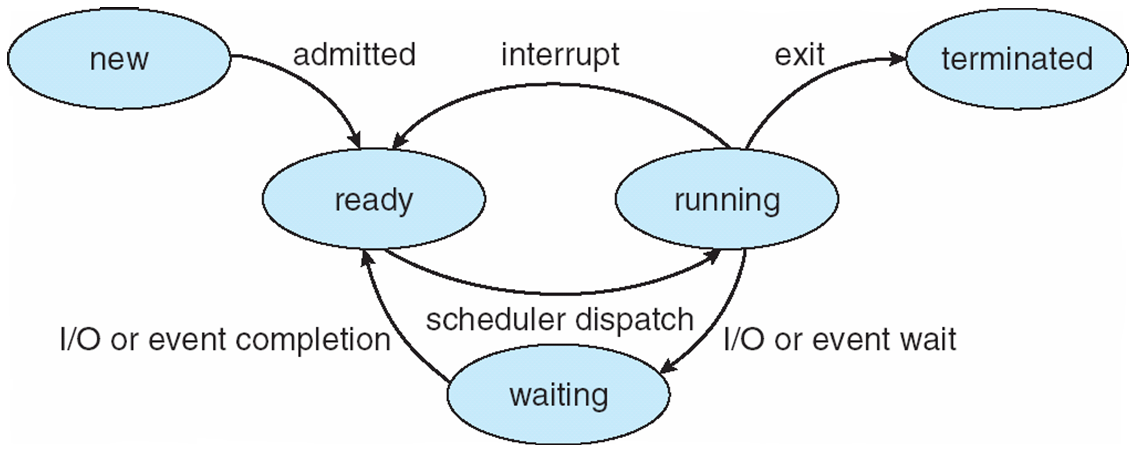
**Process transition**

* A program is in **passive** state when it is stored on disk (as an executable file).
* A program becomes a process when the executable file is loaded into memory (**active** state).
* Execution of program started via GUI mouse clicks, command line entry of its name, etc.
* One program can have several processes. E.g. consider a user opening a program multiple times.

**Process states**

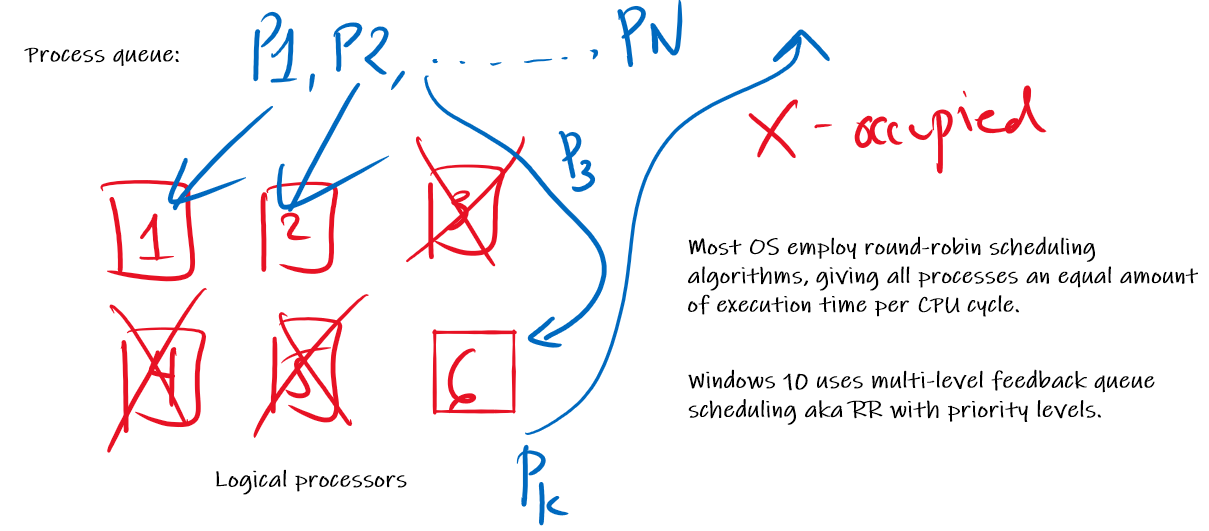
As a process executes, it changes **state**

* + **new**: The process is being created
  + **running**: Instructions are being executed
  + **waiting**: The process is waiting for some event to occur
  + **ready**: The process is waiting to be assigned to a processor
  + **terminated**: The process has finished execution

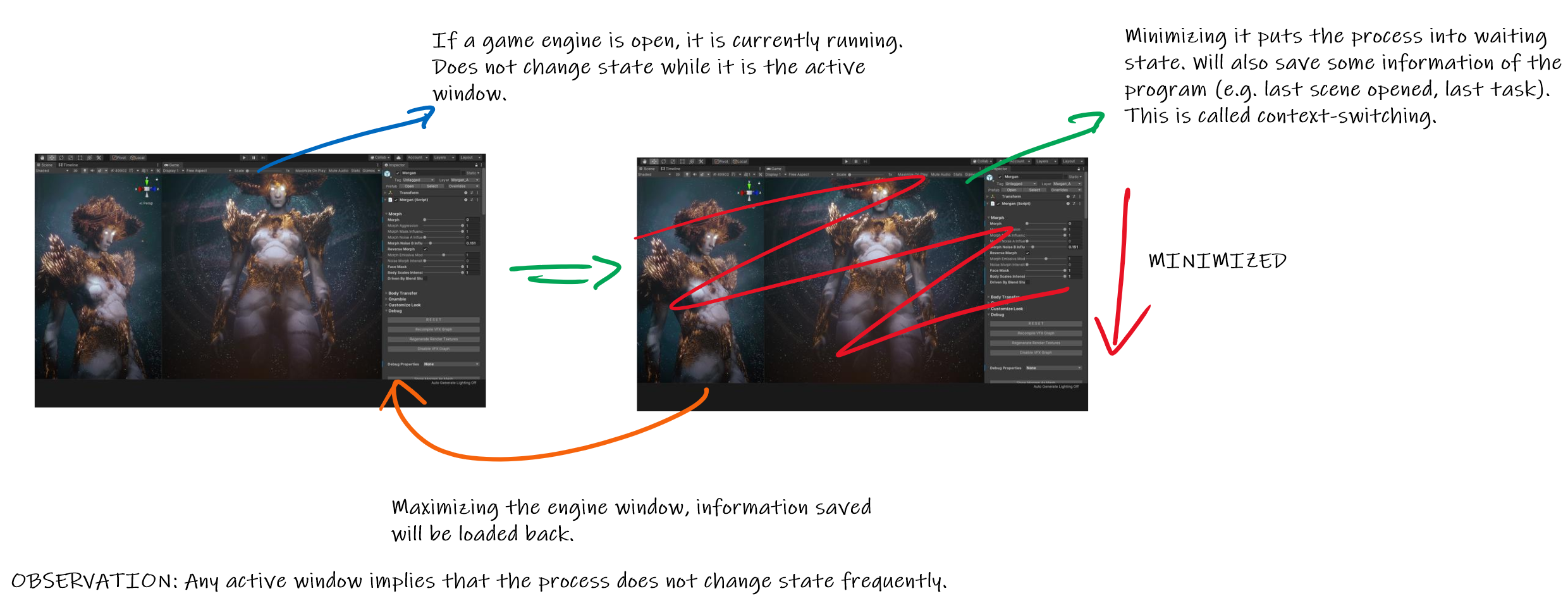


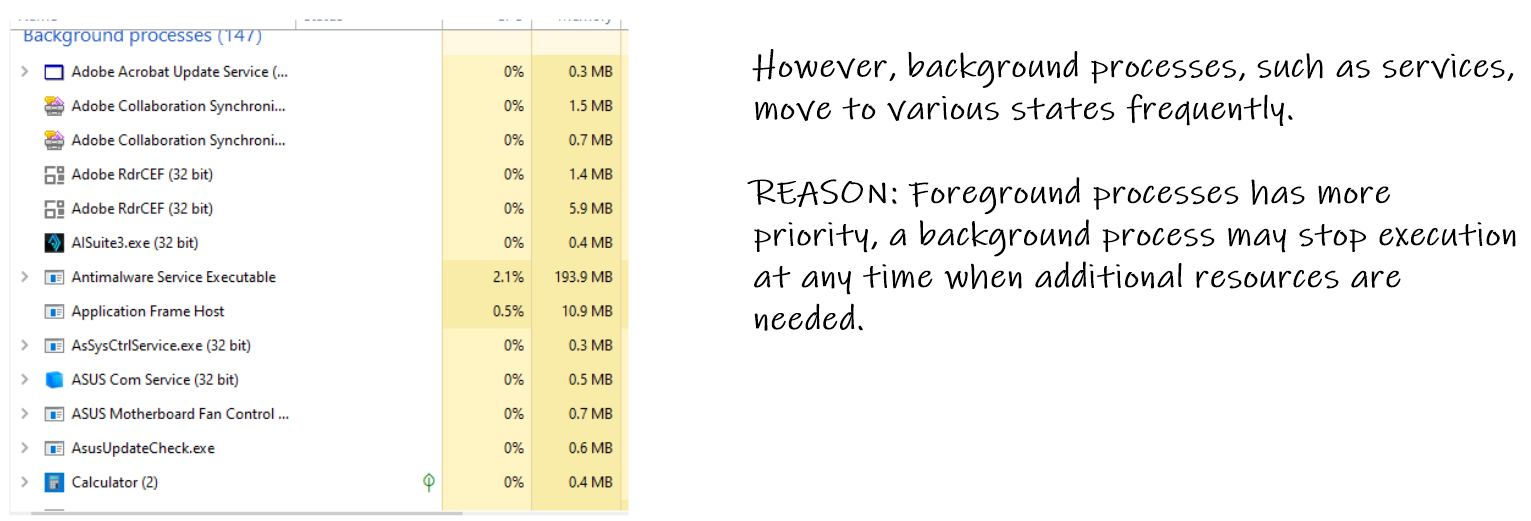
Very similar to application lifecycle for mobile.

Recall process/CPU scheduling



Processes can move to different states frequently. Consider this example:





6_01.pdf**CPU Scheduling – Basic Concepts**

* One goal of the OS is to have maximum CPU utilization. Keep CPU busy at all time and finish as many processes as possible.
* A CPU can be on idle state such as the following case on the right. Program has “scattered” sections of I/O-related code. Whenever there is an I/O code, the process must be removed from the CPU and move to the device queue (and gets executed in the I/O system program).
* A commercial OS will never have long CPU idle times even if the user is not seeing anything on-screen.

**Terms**

* CPU – not the physical CPU hardware. A logical unit for executing a process.
* Ready queue – holds processes that should be executed in the CPU.
* Short-term scheduler - selects from among the processes in ready queue and allocates to the CPU.
* Long-term scheduler – decides which processes should go first to the short-term scheduler. NOTE: Not all OS have this.
* CPU scheduling decisions may take place when a process:

1. Switches from running to waiting state

2. Switches from running to ready state

3. Switches from waiting to ready

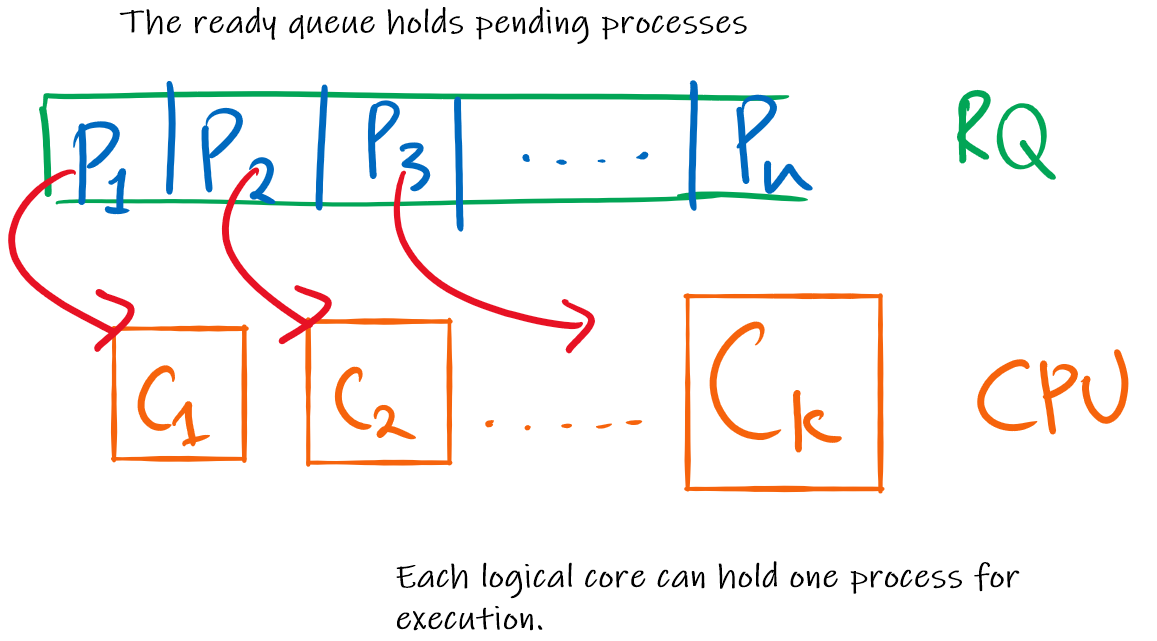
4. Terminates

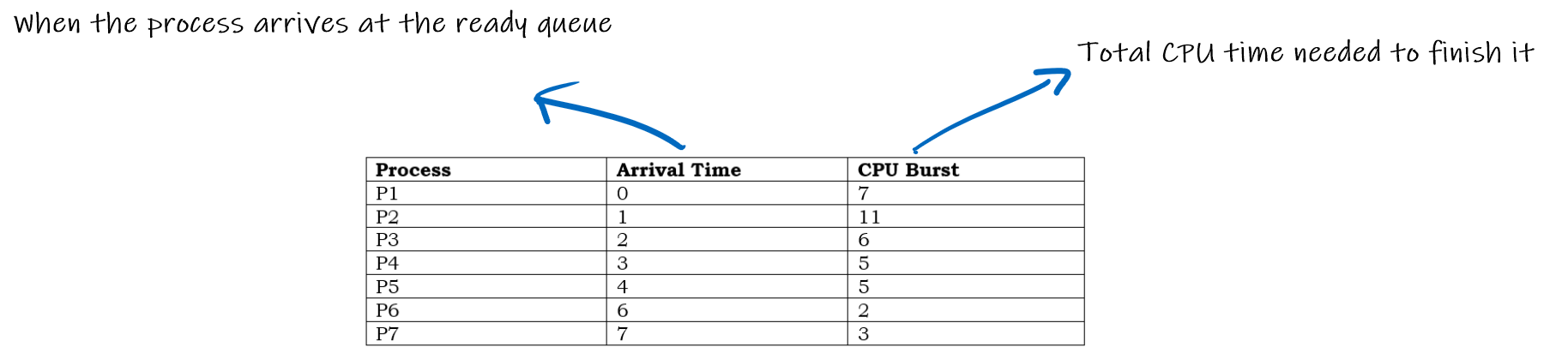
**Scheduling Criteria**

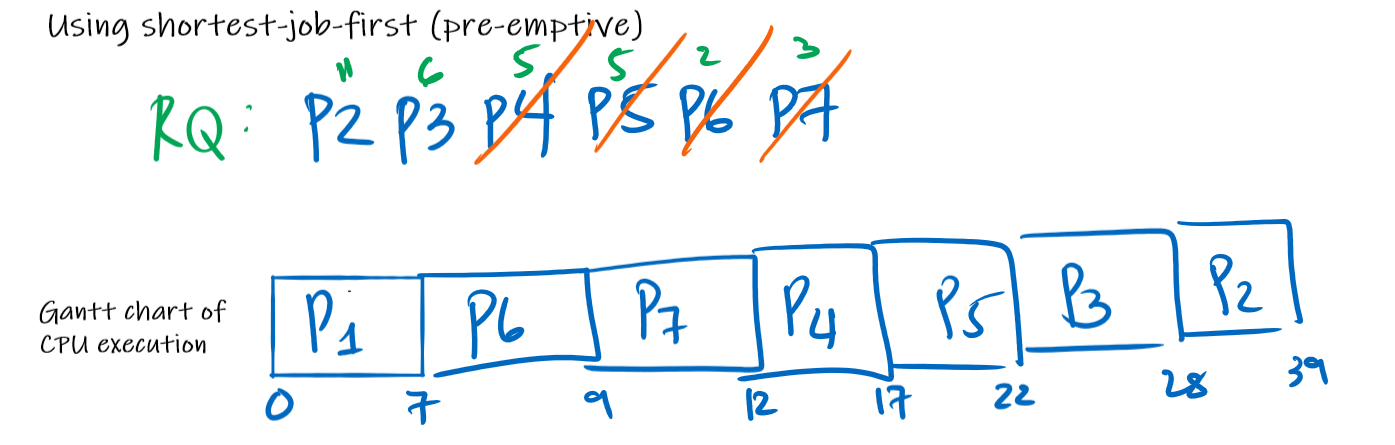
* **CPU utilization** – keep the CPU as busy as possible
* **Throughput** – # of processes that complete their execution per time unit
* **Turnaround time** – amount of time to execute a particular process
* **Waiting time** – amount of time a process has been waiting in the ready queue
* **Response time** – amount of time it takes from when a request was submitted until the first response is produced, not output (for time-sharing environment).

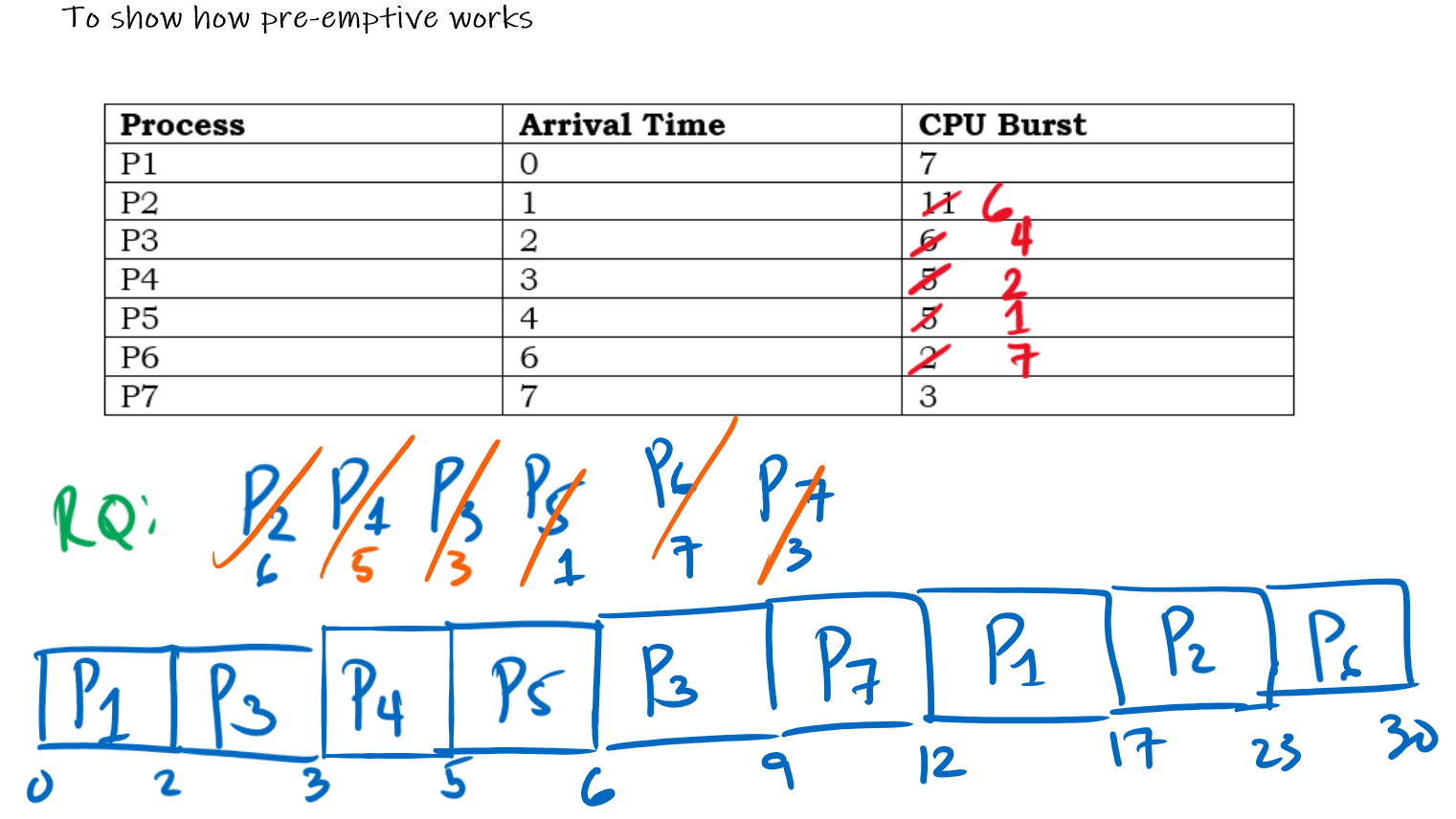
**Demonstration of CPU Scheduling: Shortest-Job-First (Pre-Emptive)**

Shortest-job-first, by the name itself, selects processes with the shortest CPU burst for scheduling. Consider the following example:



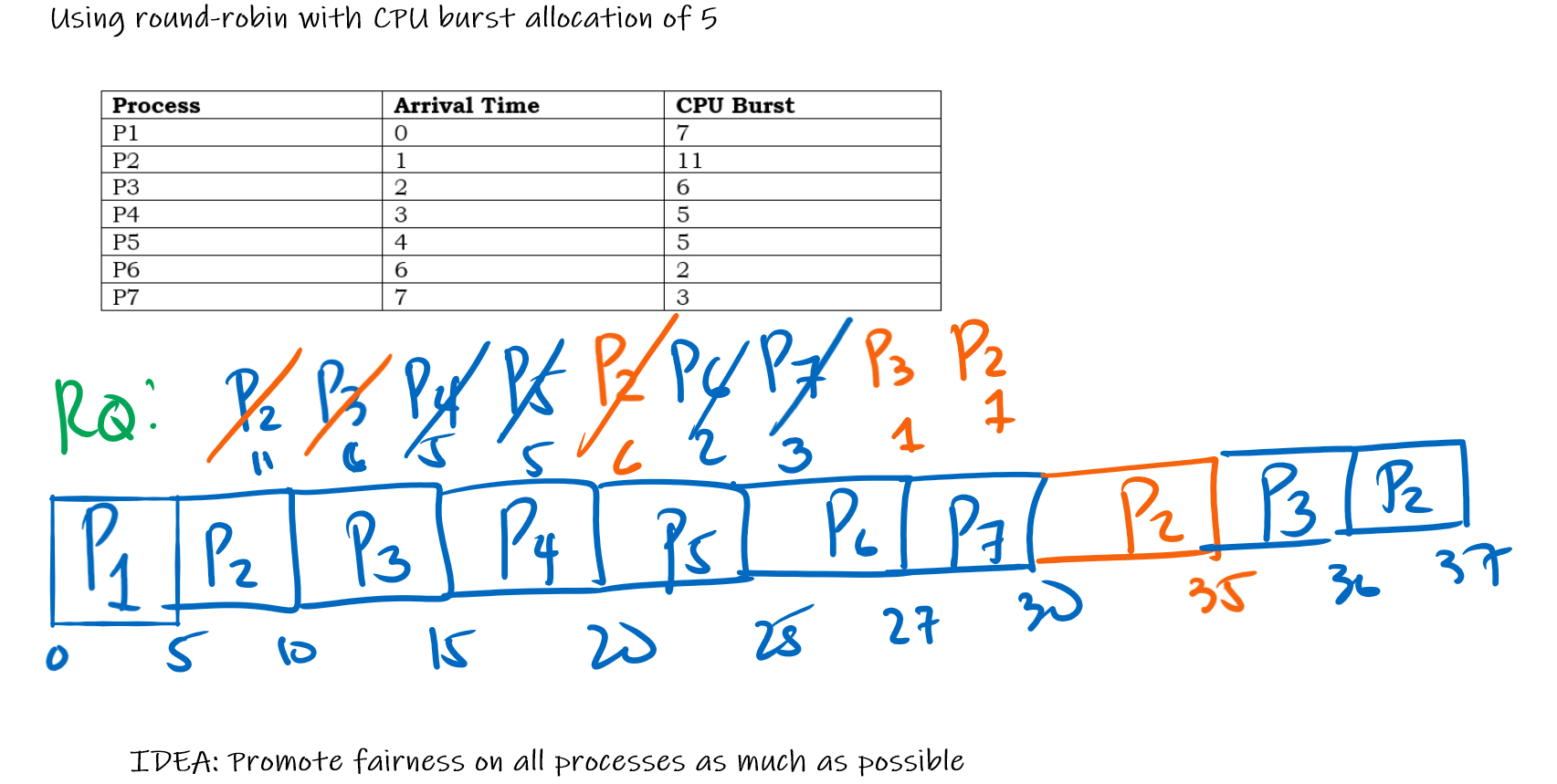




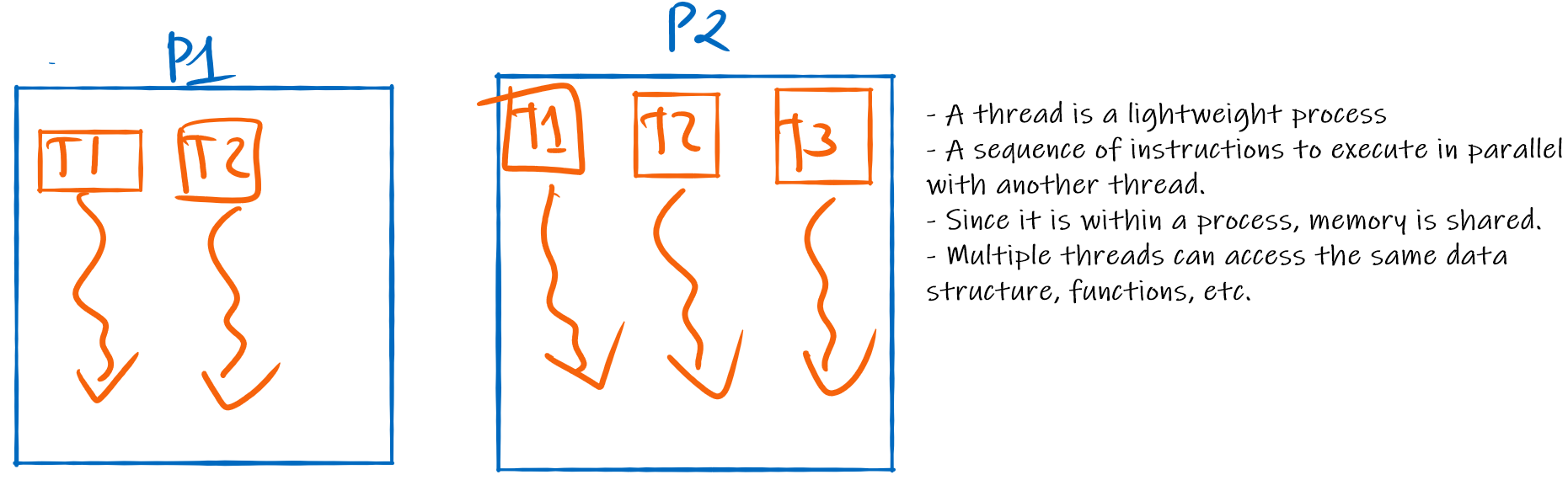


**Demonstration of CPU Scheduling: Round-Robin**

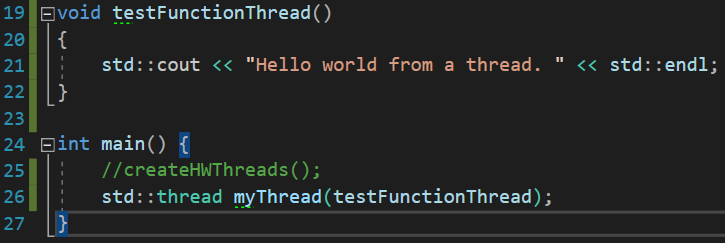
Promotes fairness policy by cycling through all processes that arrive evenly. Each process receives a fair amount of CPU time for execution.



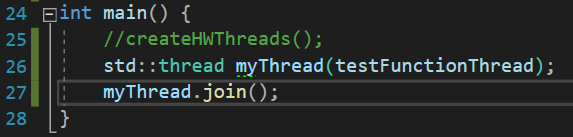
**WHAT IS A THREAD?**



**Hands-On Activity:** Create your first “Hello World” thread in C++.

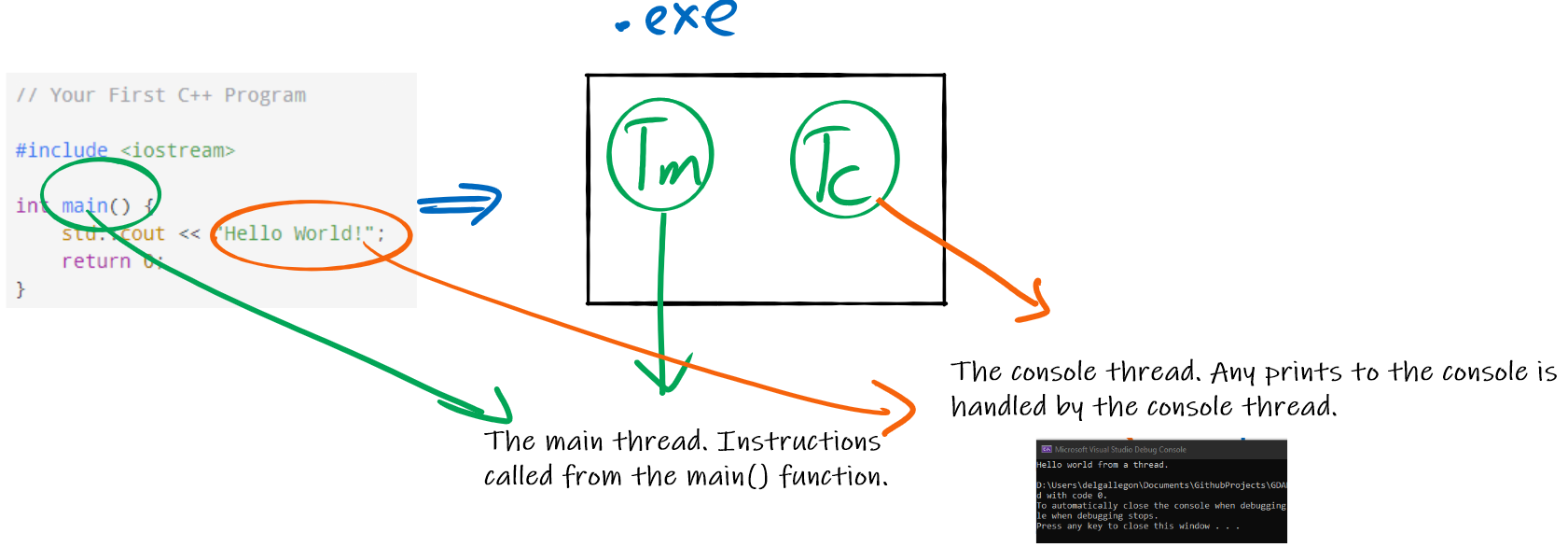


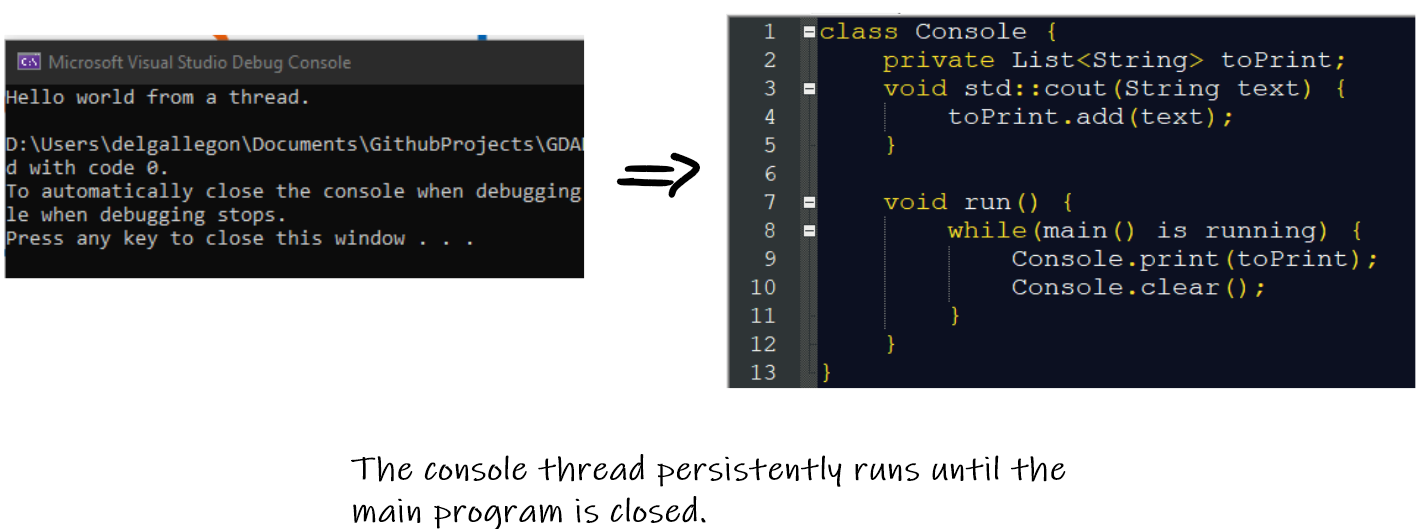
NOTE: Running the above program will trigger a runtime error. C++ has a guarding mechanism that should force the developer to call either join() or detach().



**A Windows program – Default thread handling**.

A default Windows program always has the following set of threads: **main thread, UI thread**. See illustration below.

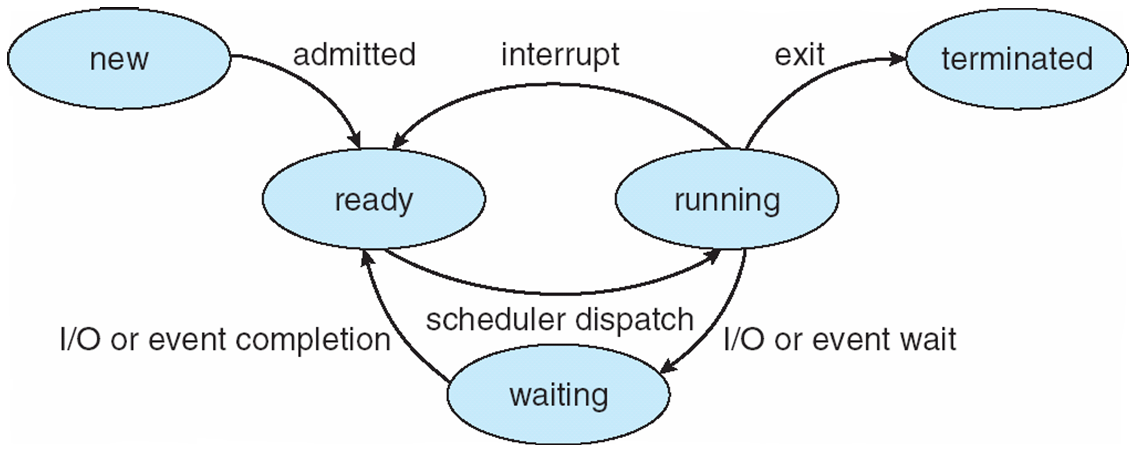






**NOTE:** All the spawned threads are independent from one another. Information is being passed to each thread through **message passing** (e.g. std::cout, render/draw frame in SFML/OPENGL).

Since threads are lightweight processes, they also undergo the same states as well.



**ACTIVITY:** For both Unity and Unreal engine, identify the different background threads that run when you use the engine. Explain the purpose of each thread.