Priority scheduling is a non-preemptive algorithm and one of the most common scheduling algorithms in batch systems. Each process is assigned a priority. Process with the highest priority is to be executed first and so on.  
Processes with the same priority are executed on first come first served basis. Priority can be decided based on memory requirements, time requirements or any other resource requirement.

1. Preemptive Scheduling:  
 Preemptive scheduling is used when a process switches from running state to ready state or from waiting state to ready state. The resources (mainly CPU cycles) are allocated to the process for the limited amount of time and then is taken away, and the process is again placed back in the ready queue if that process still has CPU burst time remaining. That process stays in ready queue till it gets next chance to execute.

Algorithms based on preemptive scheduling are: [Round Robin (RR)](https://www.geeksforgeeks.org/program-round-robin-scheduling-set-1/), [Shortest Job First (SJF basically non preemptive)](https://www.geeksforgeeks.org/program-shortest-job-first-sjf-scheduling-set-1-non-preemptive/) and [Priority (non preemptive version)](https://www.geeksforgeeks.org/operating-system-priority-scheduling-different-arrival-time-set-2/), etc.

2. Non-Preemptive Scheduling:  
Non-preemptive Scheduling is used when a process terminates, or a process switches from running to waiting state. In this scheduling, once the resources (CPU cycles) is allocated to a process, the process holds the CPU till it gets terminated or it reaches a waiting state. In case of non-preemptive scheduling does not interrupt a process running CPU in middle of the execution. Instead, it waits till the process complete its CPU burst time and then it can allocate the CPU to another process.

# **Priority Inversion and Priority Inheritance**

Imagine three (3) tasks of different priority: tLow, tMed and tHigh. tLow and tHigh access the same critical resource at different times; tMed does its own thing.

1. tLow is running, tMed and tHigh are presently blocked (but not in critical section).
2. tLow comes along and enters the critical section.
3. tHigh unblocks and since it is the highest priority task in the system, it runs.
4. tHigh then attempts to enter the critical resource but blocks as tLow is in there.
5. tMed unblocks and since it is now the highest priority task in the system, it runs.

tHigh can not run until tLow gives up the resource. tLow can not run until tMed blocks or ends. The priority of the tasks has been inverted; tHigh though it has the highest priority is at the bottom of the execution chain.

To "solve" priority inversion, the priority of tLow must be bumped up to be at least as high as tHigh. Some may bump its priority to the highest possible priority level. Just as important as bumping up the priority level of tLow, is dropping the priority level of tLow at the appropriate time(s). Different systems will take different approaches.

When to drop the priority of tLow ...

1. No other tasks are blocked on any of the resources that tLow has. This may be due to timeouts or the releasing of resources.
2. No other tasks contributing to the raising the priority level of tLow are blocked on the resources that tLow has. This may be due to timeouts or the releasing of resources.
3. When there is a change in which tasks are waiting for the resource(s), drop the priority of tLow to match the priority of the highest priority level task blocked on its resource(s).

Method #2 is an improvement over method #1 in that it shortens the length of time that tLow has had its priority level bumped. Note that its priority level stays bumped at tHigh's priority level during this period.

Method #3 allows the priority level of tLow to step down in increments if necessary instead of in one all-or-nothing step.

Different systems will implement different methods depending upon what factors they consider important.

* memory footprint
* complexity
* real time responsiveness
* developer knowledge