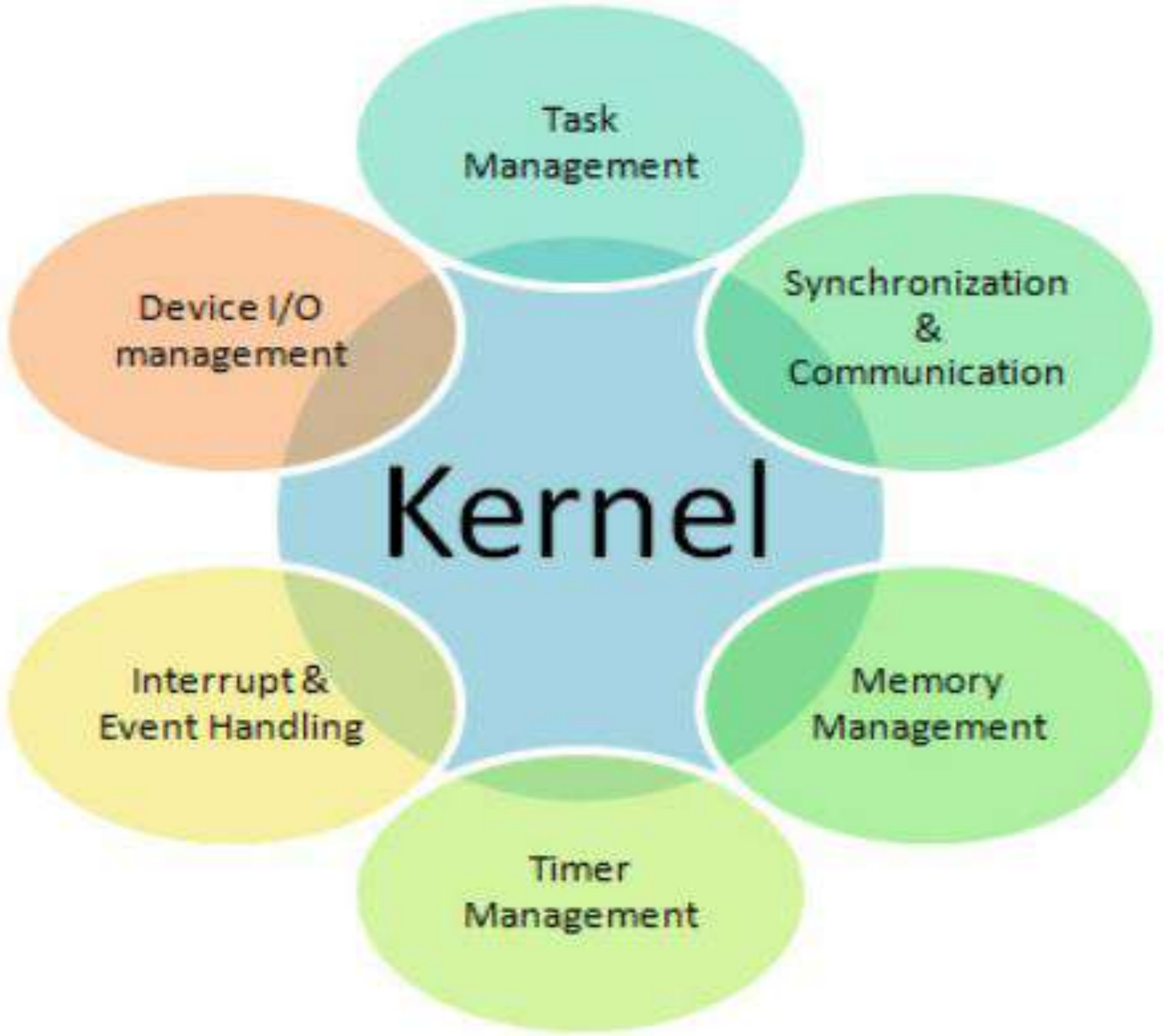
UNIT 5

SCHEDULING IN RTOS

Services/Functions offered by RTOS



Task management

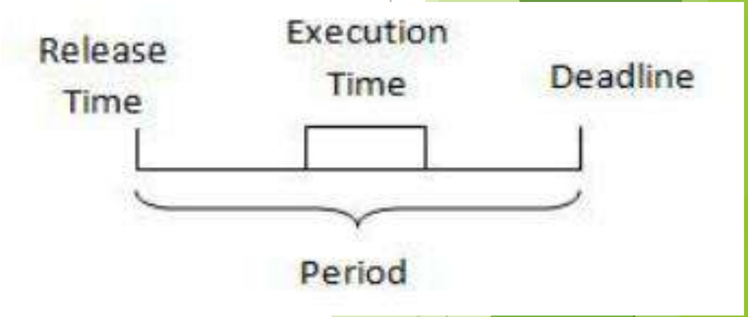
- The application is decomposed into small, schedulable, and sequential program units known as "Task"
- Execution and is governed by three time critical properties;
- release time : Release time refers to the point in time from which the task can be executed
- Deadline: Deadline is the point in time by which

the task must complete

- execution time: Execution time denotes the time

the task takes to execute.

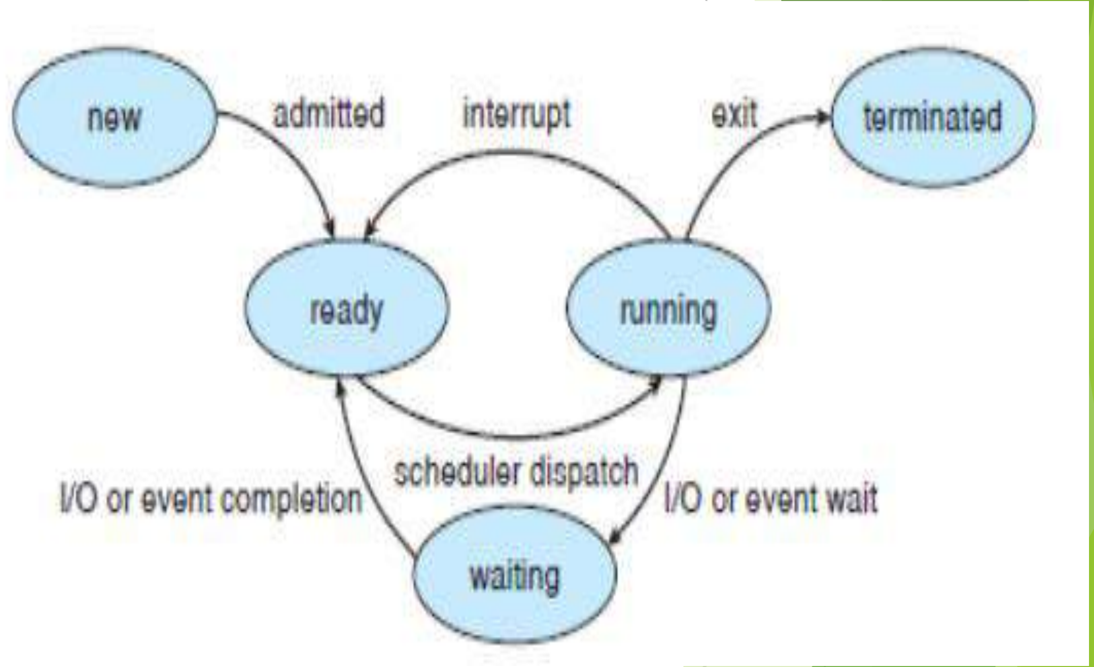
Task management is known as scheduling



Time stages/scheduling states

Running to waiting-N
Running to termination- N
Running to ready - P
Waiting to ready - P

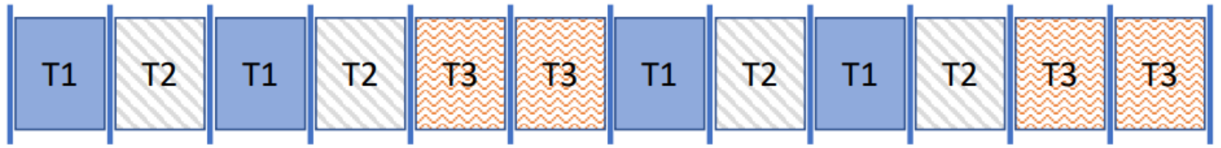
- Dormant : Task doesn’t require computer time
- Ready: Task is ready to go active state, waiting processor time
- Active: Task is running
- Suspended/waiting: Task put on hold temporarily
- Pending: Task waiting for resource.



During the execution of an application program, individual tasks are continuously changing from one state to another. However, only one task is in the running mode (i.e. given CPU control) at any point of the execution

Scheduler:

- The scheduler keeps record of the state of each task and selects from among them that are ready to execute and allocates the CPU to one of them.

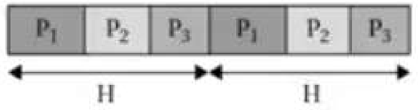


- Choosing the orde
- Scheduling Policy: dy state to running state

The part of the OS or kernel that is responsible for determining which task to run next

Scheduling methods/Policy

1) Simple scheduling- based on TDMA

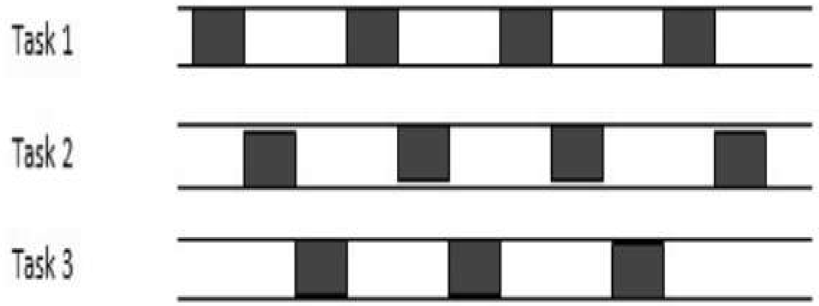


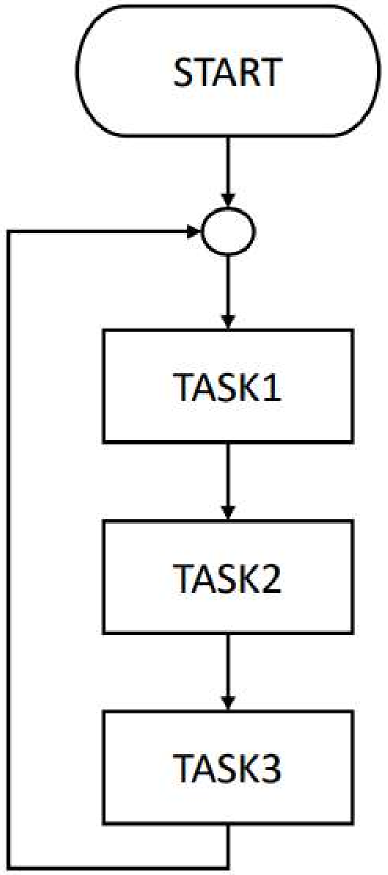
H is hyper period

2) Round Robin scheduling

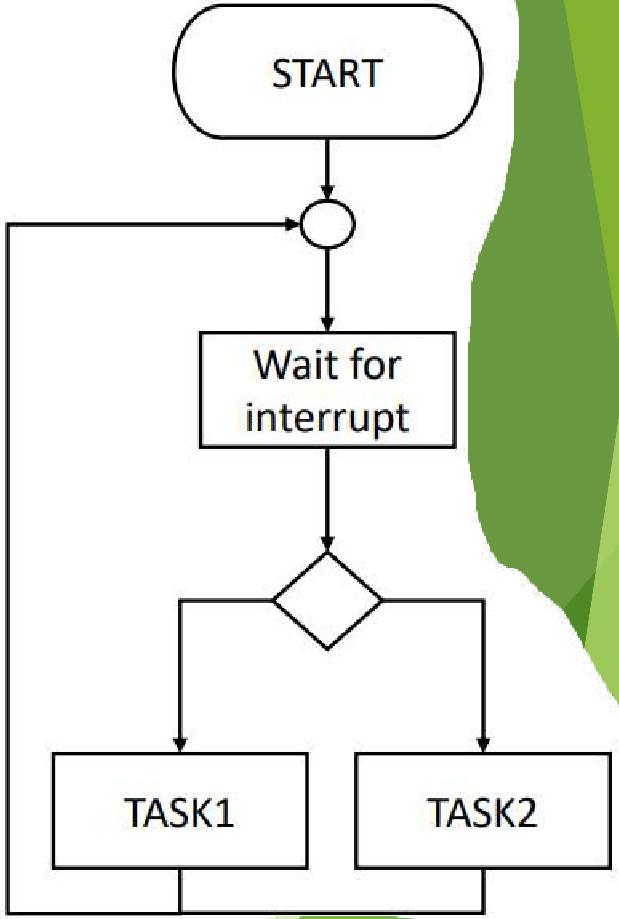
Each process is evaluated one after the other

Hyper period may be left empty if no task to perform



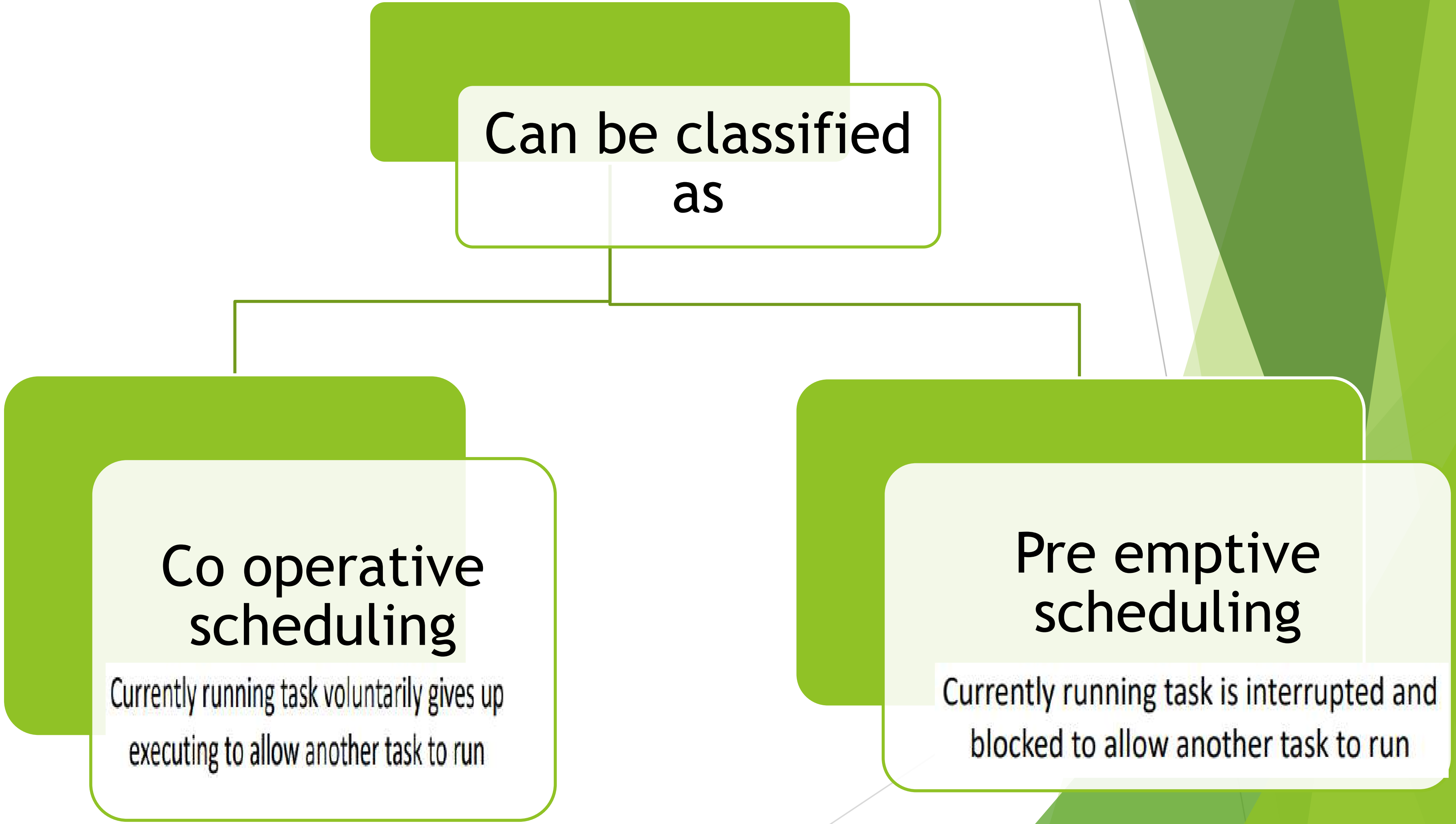


Polled



Interrupt driven

Scheduling Types

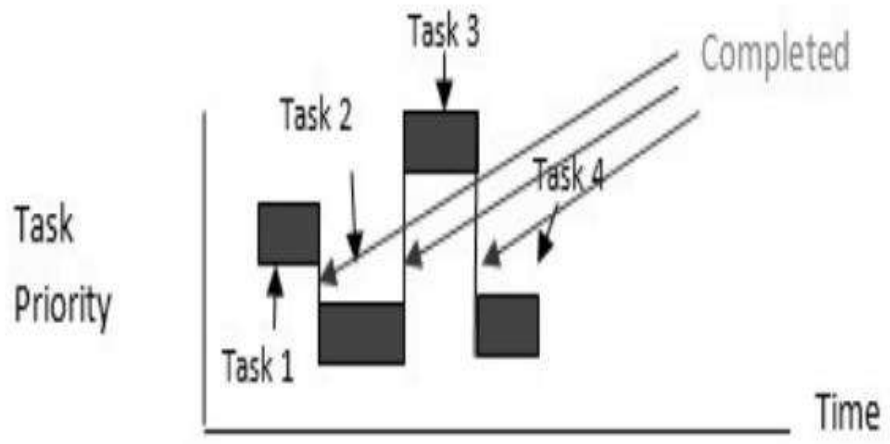


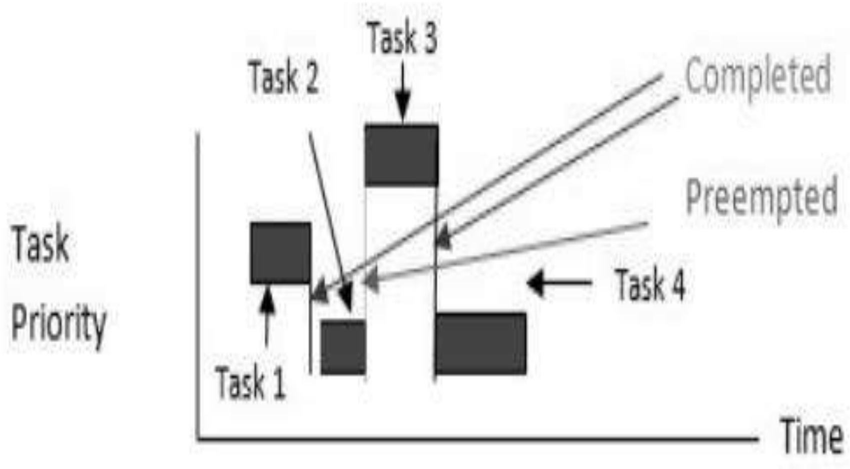
Currently running task voluntarily gives up executing to allow another task to run

Currently running task is interrupted and blocked to allow another task to run

Scheduling strategies

Non Preemptive or Co operative multitasking	Preemptive or Priority Multitasking
Currently running task voluntarily gives up executing to allow another task to run	Currently running task is interrupted and blocked to allow another task to run





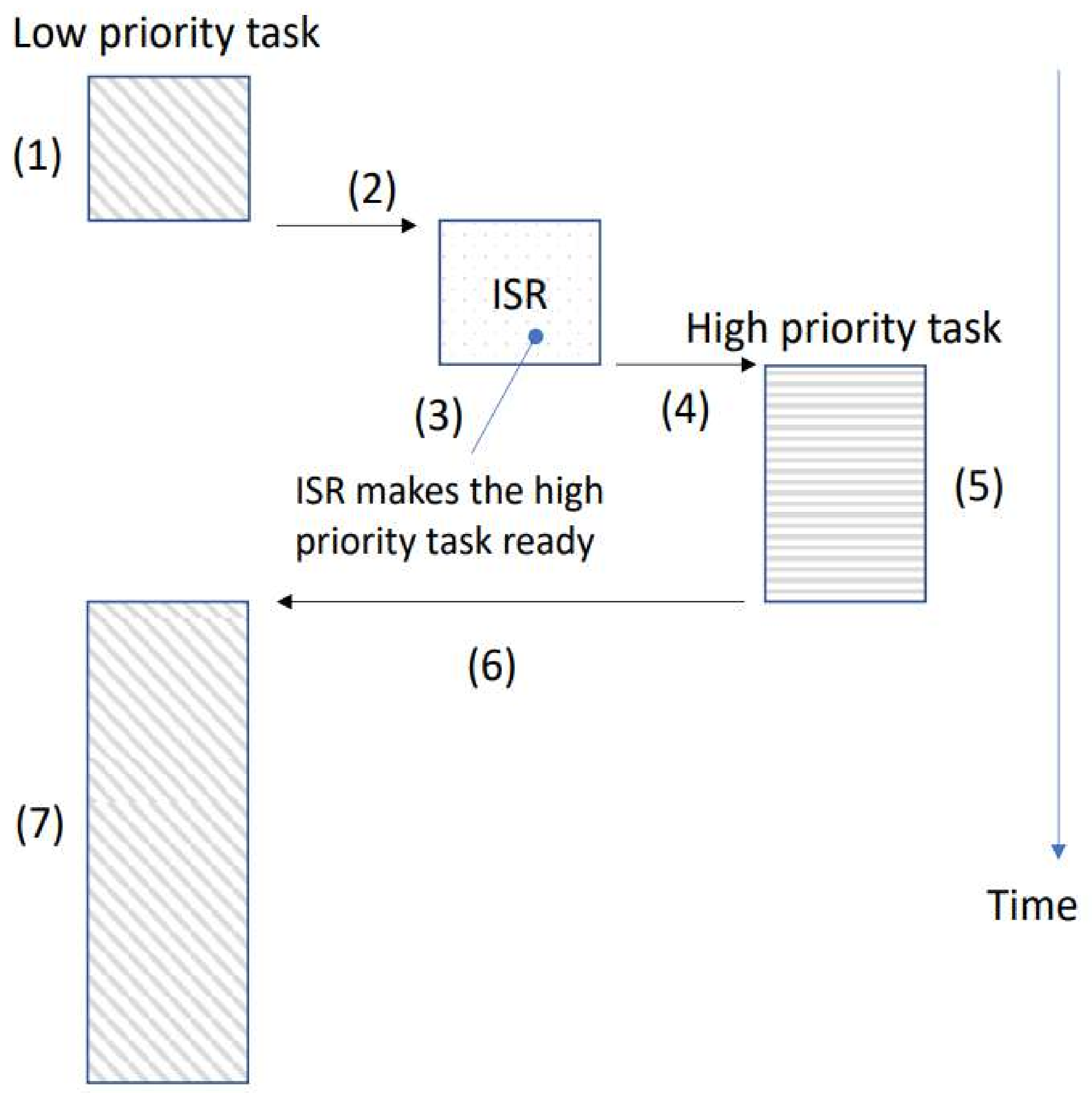
Pre emptive

- Disadvantage: In non preemptive scheduling each ready task cooperates to let the running one finish.

  A long execution time of low priority task waits at least until that finishes.
- In computing preemption is the act of temporarily interrupting a task being carried out by a computer system
- with the intention of resuming the task at a later time.
- Such a change is known as a context switch. It is normally carried out by a privileged task or part of the system known as a preemptive scheduler.
- When a higher priority task needs to be executed, the RTOS must save all the information needed to eventually resume the task being suspended. Typically consists of most/all of the CPU registers

Preemptive

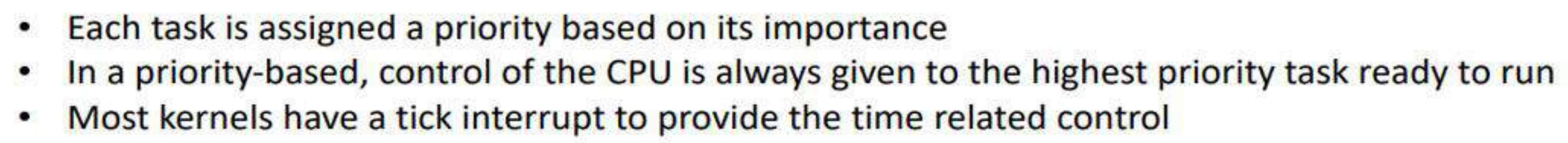
- The highest priority task ready to run is always given control of the CPU.
- When a task makes a higher priority task read to run, the current task is preempted (suspended), and the higher priority task is immediately given control of the CPU.
- If an ISR makes a higher priority task ready, when the ISR completes, the interrupted task is suspended, and the new higher priority task is resumed.
- Execution of the highest priority task is deterministic
- Corruption of data can occur if the higher priority task preempts a lower priority task that is using e.g. a shared resource



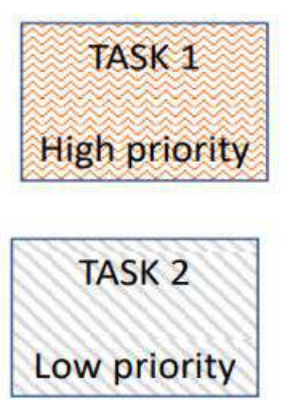
Preemptive

When the currently running task is interrupted to allow a higher priority task to run, is called

Priority based Preemptive scheduling

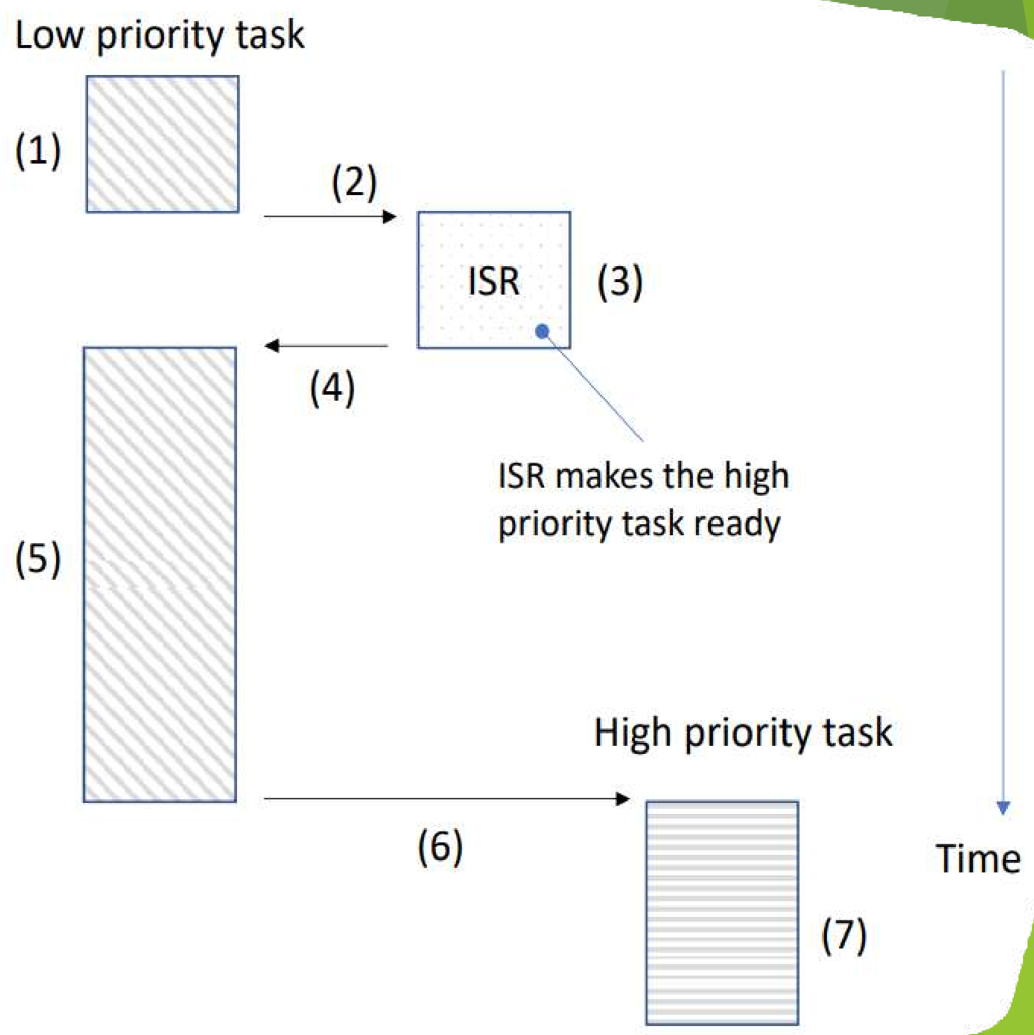


- Each task is assigned a priority based on its importance
- In a priority-based, control of the CPU is always given to the highest priority task ready to run
- Most kernels have a tick interrupt to provide the time related control



Non Preemptive

- Requires that each task does something to explicitly give up control of CPU
- To maintain the illusion of concurrency, this process must be done frequently
- Also called cooperative multitasking; tasks cooperate with each other to share the CPU
- Asynchronous events still handle by ISR
    - An ISR can make a higher priority task ready to run
    - but it always returns to the interrupted tasks
- The new higher priority task gains control of CPU only when the current task gives up the CPU
- The most important drawback is responsiveness
- Response time is nondeterministic
- Very few commercial kernels are non-preemtive



Common Scheduling Techniques

- Round Robin

  Static scheduling algorithm

  It may be non preemptive or preemptive

- Early deadline first (EDF)

  Dynamic scheduling algorithm.

  Will run the tasks which has the shortest time to its deadline.

  Generally preemptive.

- Rate-Monotonic scheduling (RMS)

  Fixed priority.

  Static priorities assign according to the cycle duration, that is, a short task duration gives a higher priority.

  Generally preemptive.

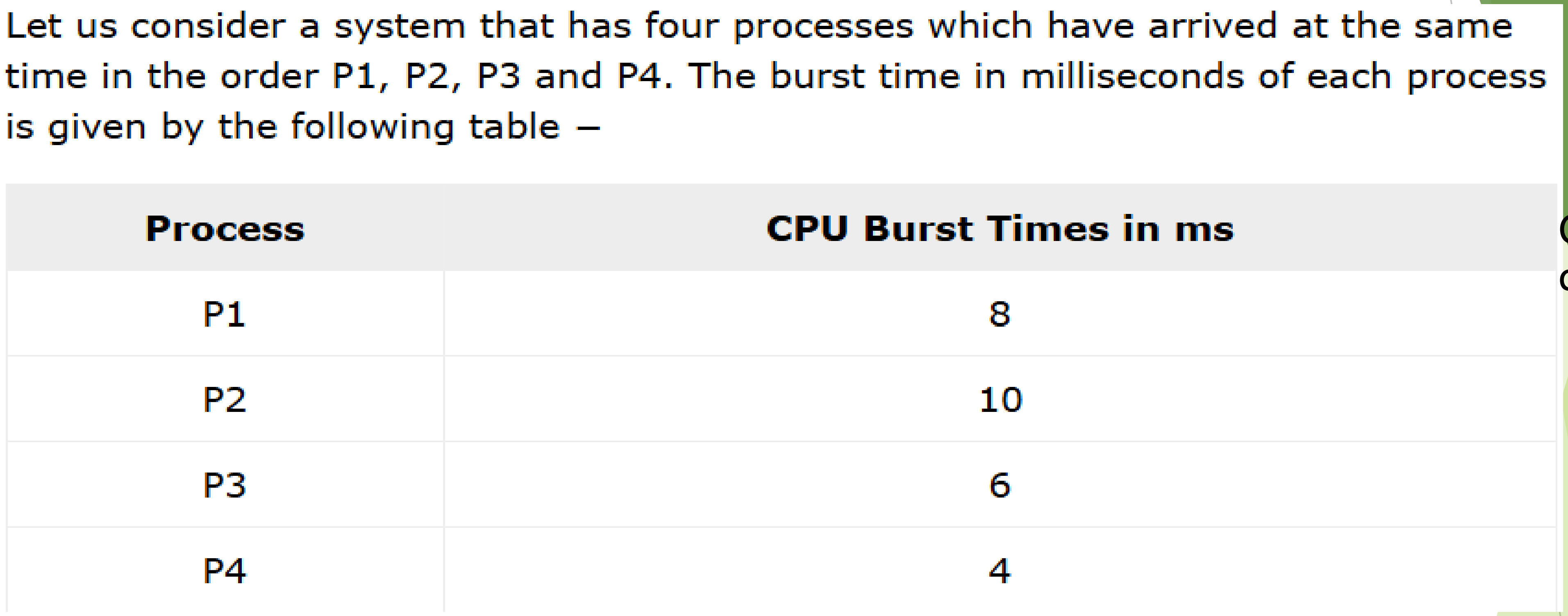
Round Robin scheduling (non preemptive)

- In RR scheduling, each process gets equal time slices (or time quantum) for which it executes in the CPU in turn wise manner.
- When a process gets its turn, it executes for the assigned time slice and then relinquishes the CPU for the next process in queue.
- RR is a fair scheduling strategy where all processes get equal share to execute in turn wise manner.
- The performance of RR scheduling is vastly dependent on the chosen time quantum
- Through RR scheduling strategy, none of the processes go into starvation.
- It is widely used for its simple working principle

Round Robin algorithm

- New process that arrives in the system is inserted at the end of the ready queue in FCFS manner.
- The first process in the queue is removed and assigned to the CPU.
- If the required burst time is less than or equal to the time quantum, the process runs to completion.
- The scheduler is invoked when the process completes executing to let in the next process in the ready queue to the CPU.
- If the required burst time is more than the time quantum, the process executes up to the allotted time quantum and added to the end of the queue.
- Context switch occurs and the next process in the ready queue is assigned to the CPU.
- The above steps are repeated until there are no more processes in the ready queue

Example



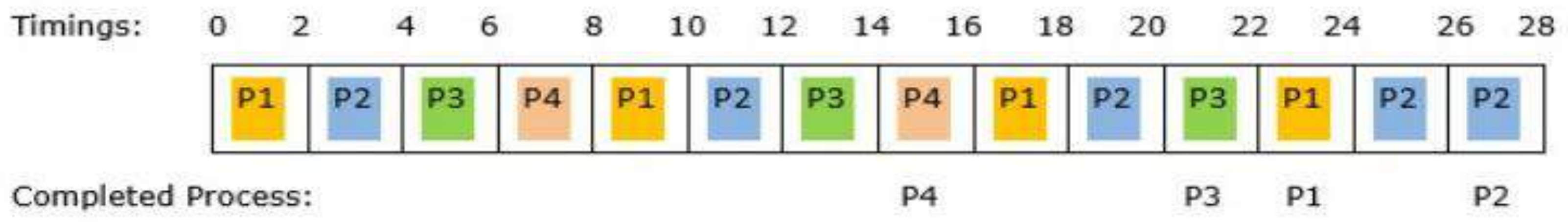
Let us consider a system that has four processes which have arrived at the same time in the order P1, P2, P3 and P4. The burst time in milliseconds of each process is given by the following table –

Process	CPU Burst Times in ms
P1	8
P2	10
P3	6
P4	4

Consider time quantum of 2ms

Evaluate average turnaround time

solution



Average Turnaround Time

Average TAT = Sum of Turnaround Time of each Process / Number of Processes

$$= (\mathrm{TAT}_{P1} + \mathrm{TAT}_{P2} + \mathrm{TAT}_{P3} + \mathrm{TAT}_{P4})/4$$

= (24 + 28 + 22+ 16) / 4 = 22.5 ms

Disadvantages of RR scheduling method:

1. The performance of Round Robin scheduling is highly dependent upon the chosen time quantum.
2. If the chosen time quantum is too small, the CPU will be very busy in context switching, i.e. swapping in swapping out processes to and from the CPU and memory. This would reduce the throughput of the system since more time will be expended in context switching rather than actual execution of the processes.
3. RR scheduling does not give any scope to assign priorities to processes. So, system processes which need high priority gets the same preference as background processes.

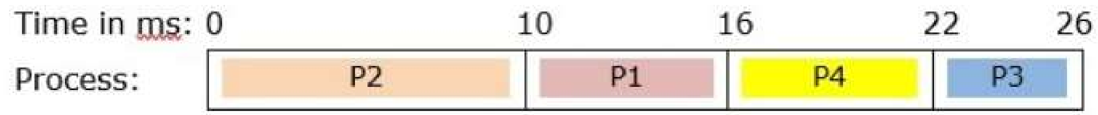
Round Robin- priority driven (preemptive scheduling)

Suppose that we have a set of four processes that have arrived at the same time in the order P1, P2, P3 and P4. The burst time in milliseconds and their priorities of the processes is given by the following table –

Process	CPU Burst Time in ms	Priority
P1	6	2
P2	10	1
P3	4	3
P4	6	2

Considering that a lower priority value means higher priority. Evaluate Average turn around time

- Highest priority task will be executed first.
- If the process has same priority the one that arrives first in the table will be allocated the recourses.



Let us compute the average turnaround time and average waiting time from the above chart.

Average Turnaround Time

= Sum of Turnaround Time of each Process/ Number of Processes

= (TAT_{P1} + TAT_{P2} + TAT_{P3} + TAT_{P4}) / 4

= (16 + 10 + 26 + 22) / 4 = 18.5 ms

Priority inversion problem

- Scheduling the processes without considering the resources that the processes require, can cause priority inversion.
- In this a low-priority process blocks execution of a higher priority process by keeping hold of its resources.
- Priority inversion problem occurs commonly in real time kernels.
- Example : Consider task 1 has a higher priority than task 2 and task 2 has a higher priority than task 3. Assume task 1 and task 3 share a resource through mutual exclusion. While task 3 is executing and holding the resource if task 2 is ready, it is scheduled because it has higher priority. At this time, even though task 1 has higher priority it cannot execute because the blocked task 3 is holding the shared resource. That is, a lower priority process is blocking a higher priority process. This is the priority inversion problem.

Solution to priority inversion

- Assigning Task priorities
- There are two major ways to assign priorities
- Static priorities- The priorities of the task that do not change during execution are called as static priorities. Once the Priority of the task is assigned, its value is retained till the end or completion of task.

Example: Rate Monotonic Scheduling (RMS)

- Dynamic priorities- The priorities of the task that are dynamically changing during the execution are called as dynamic priorities. These priorities will change at each and every instant of time based on the current scenario.
- Example: Earliest Deadline First (EDF)

Rate Monotonic Scheduling (RMS)

- Example: For the processes shown in figure draw a Gantt chart to show the execution of RMS algorithm.
- Solution:

1) Check whether the process can be executed within deadline

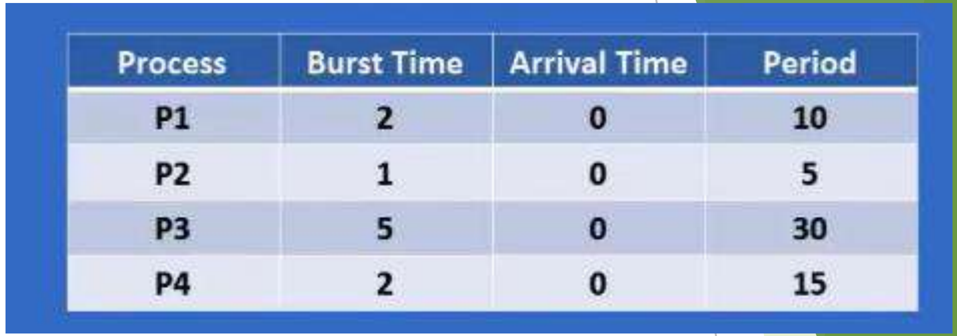
∑Ui ≤ n (2^1/n -1)

Ui(CPU utilization) = Burst time / Period

Here, n = 4 (P1, P2, P3, P4)

2) To get the total execution time- evaluate the LCM of all the processes period

Here LCM of 10, 5, 30, 15 = 30



Process	Burst Time	Arrival Time	Period
P1	2	0	10
P2	1	0	5
P3	5	0	30
P4	2	0	15

U1 = 2/10
U2 = 1/5
U3 = 5/10
U4 = 2/15
∑Ui = 0.7
n (2^1/n -1) = 0.756
Since 0.7 ≤ 0.756
We can apply RMS and it will not miss the deadline

Gantt chart

- Process with least period will have highest priority

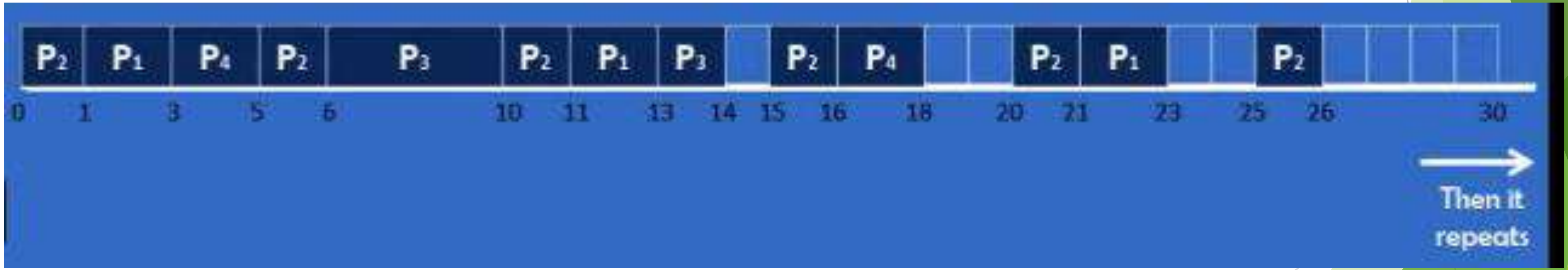
here P2 > P1> P4> P3

- Process with same period will have priority order according to how they are listed in the table.
- Implementation- In time frame of 5, P2 will executed 1 time.

  In time frame of 10, P1 will executed 2 time.

  In time frame of 15, P4 will executed 2 time.

  In time frame of 5, P3 will executed 3 time.

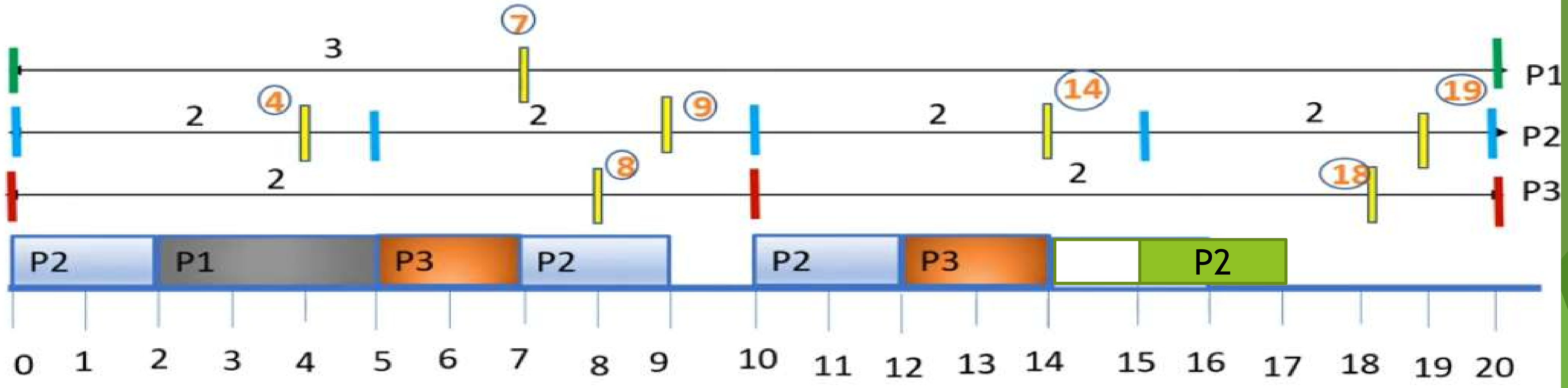


Earliest Deadline First (EDF)

Process	Capacity	Deadline	period
P1	3	7	20
P2	2	4	5
P3	2	8	10

Criteria:

Take L.C.M of the period: 20
CONSIDER THE EARLIEST DEADLINE



OVER