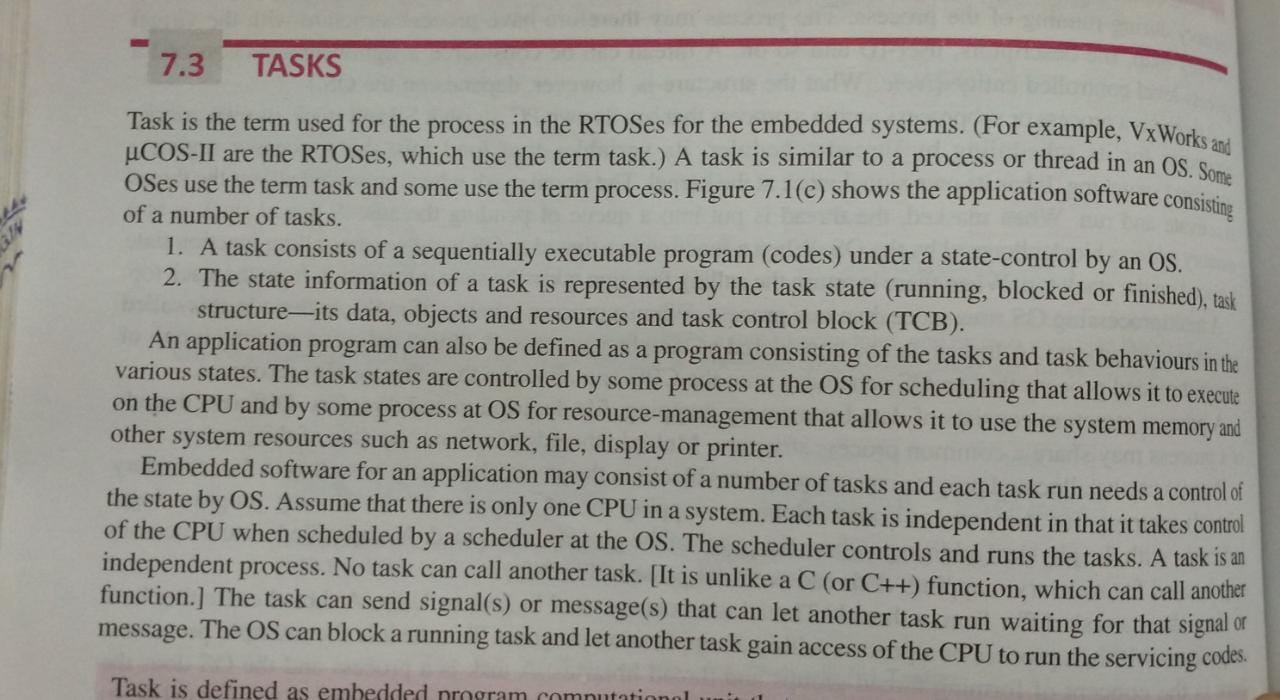
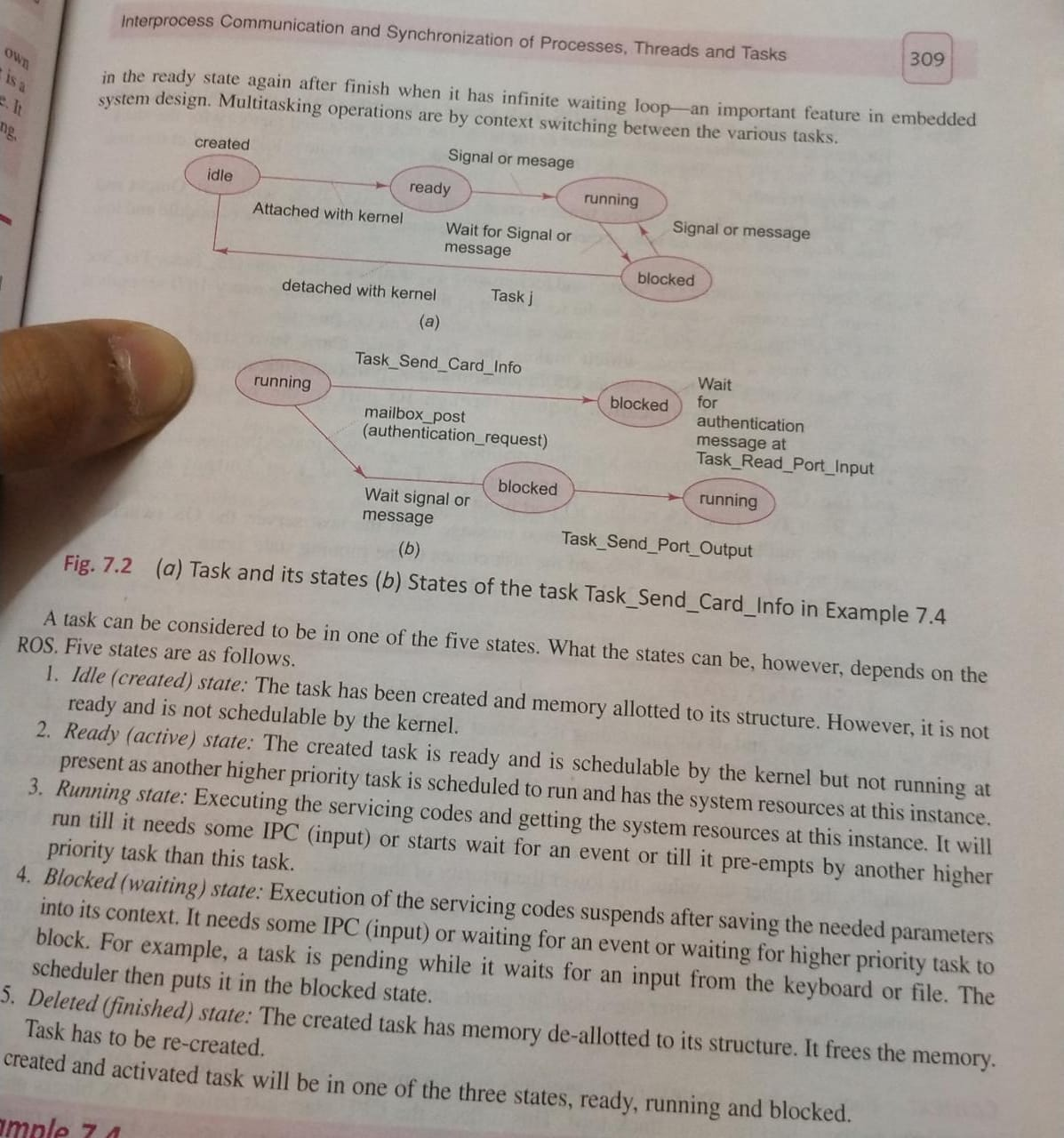
**Agenda**

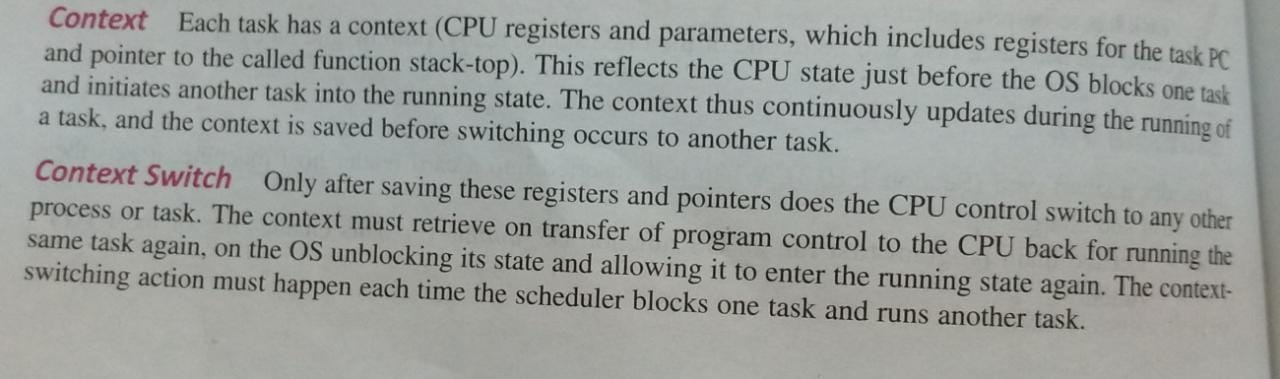
Go through all ppts right from the ppt 4 page 4 to ppt 15.

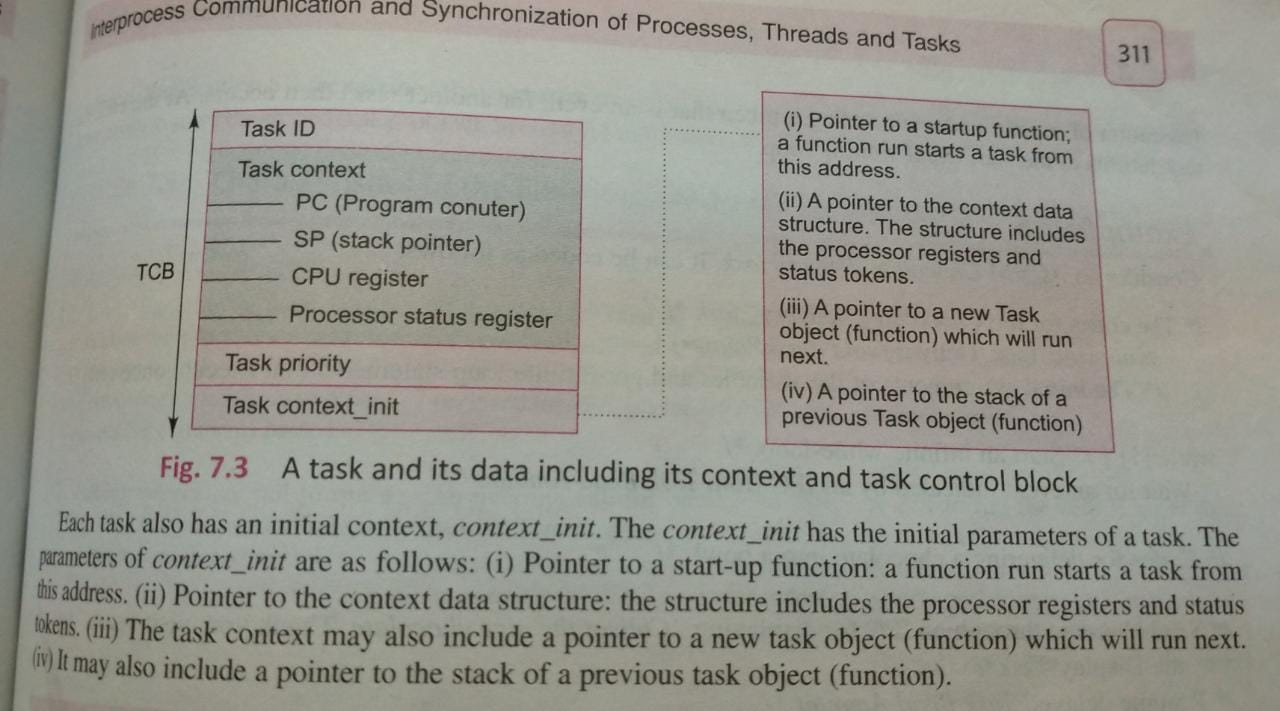
**Task and its state:**

****

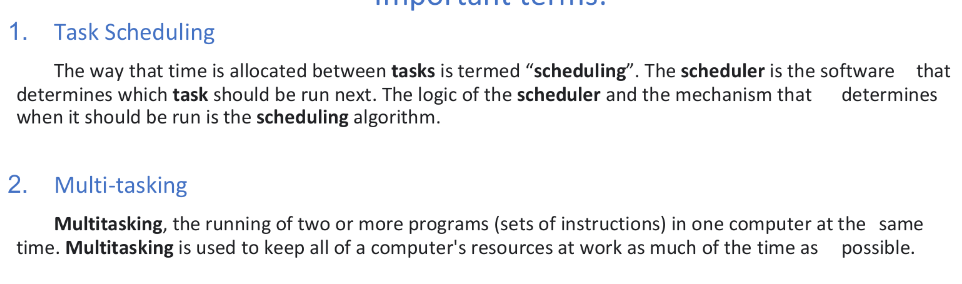
****

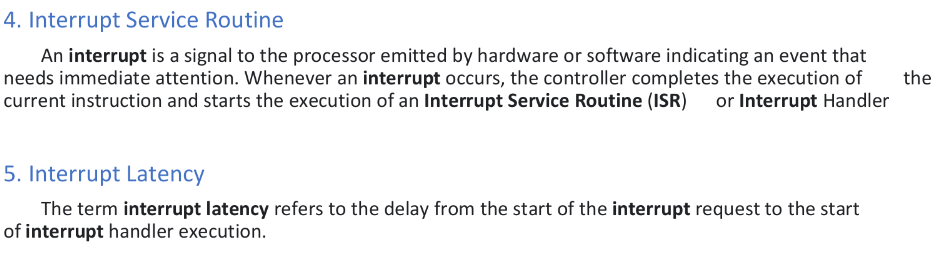
**Context switching and TCB**

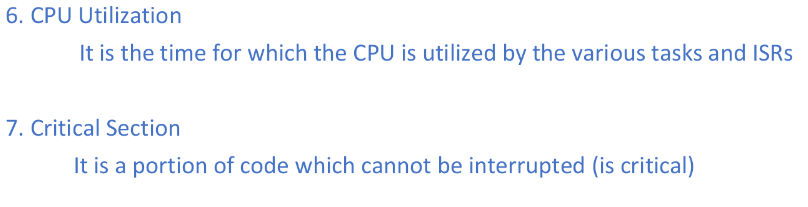
****

****

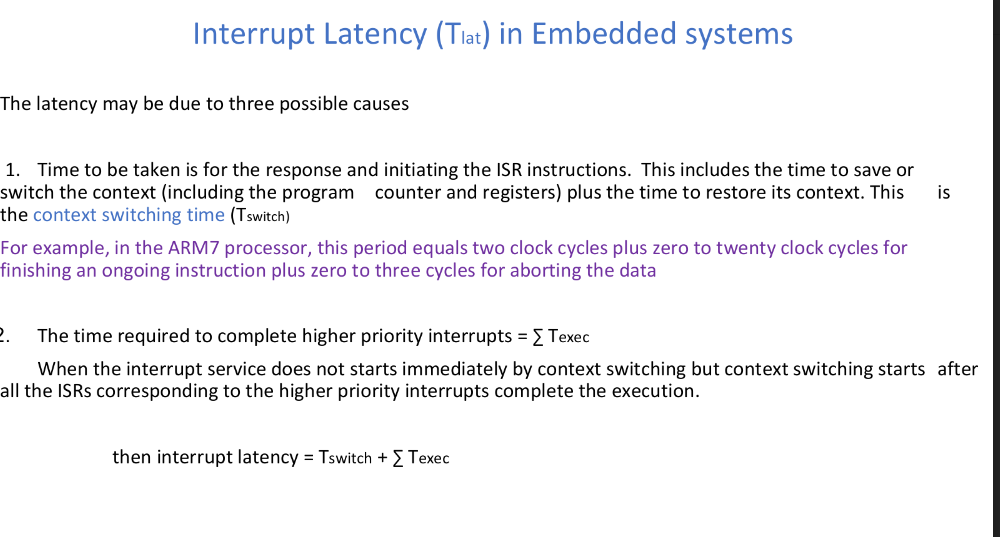
**Important Terms**

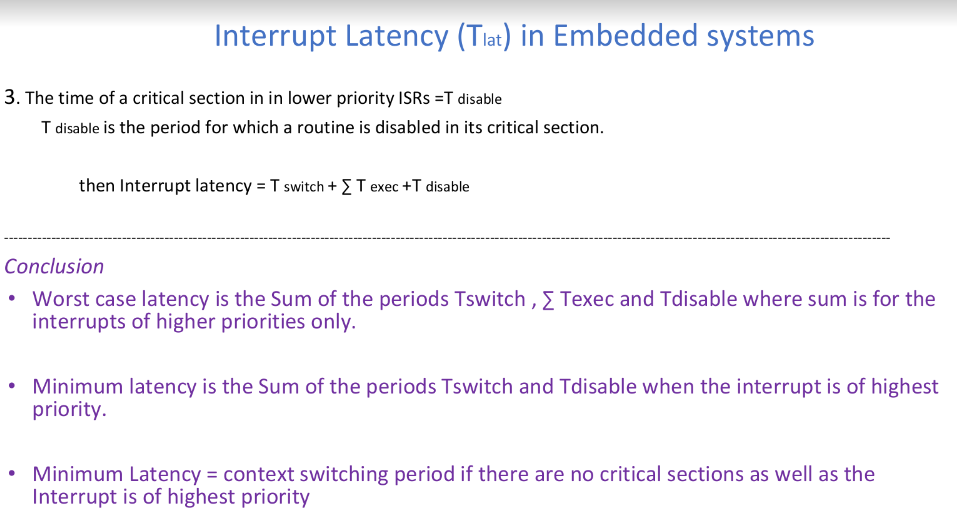
****

****

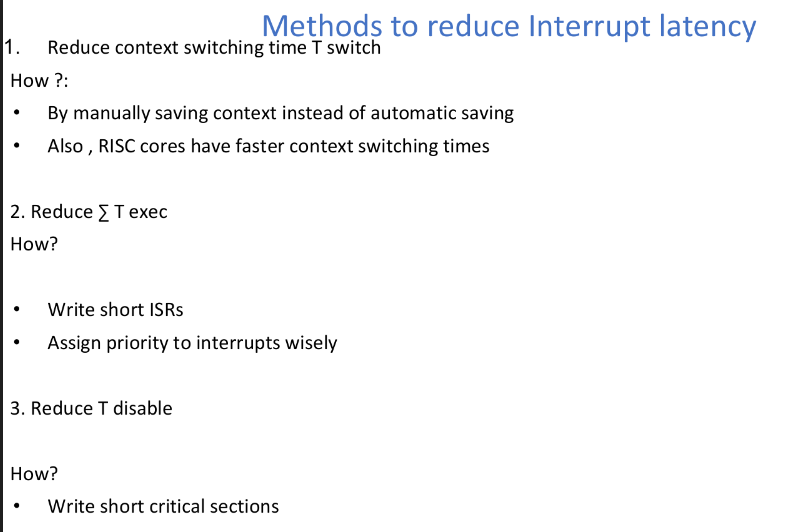


**Terminology of Interrupt latency.**

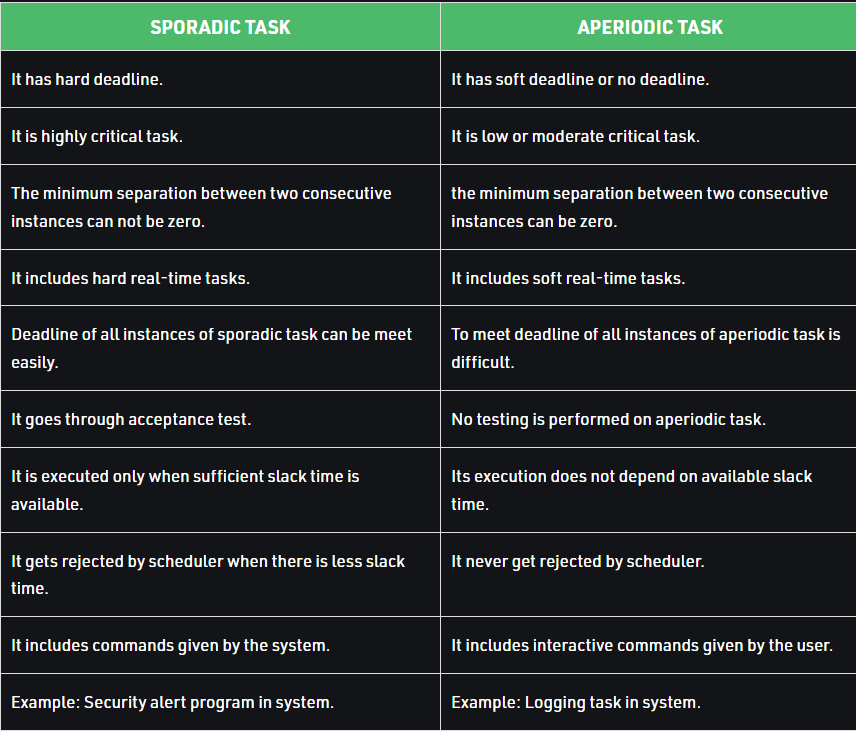




**Methods to reduce interrupt latency : Pending**

****

**Sporadic vs Aperiodic**

****

****

**Scheduling algorithm:-**

Preemptive SJF - In SJF scheduling, the process with the lowest burst time, among the list of available processes in the ready queue, is going to be scheduled next.

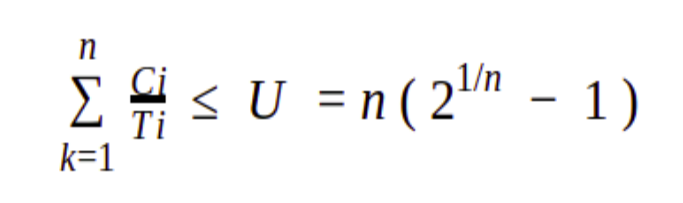
Round-Robin -  
Round Robin scheduling algorithm is one of the most popular scheduling algorithm which can actually be implemented in most of the operating systems. This is the **preemptive version** of first come first serve scheduling. The Algorithm focuses on Time Sharing. In this algorithm, every process gets executed in a **cyclic way**. A certain time slice is defined in the system which is called time **quantum**. Each process present in the ready queue is assigned the CPU for that time quantum, if the execution of the process is completed during that time then the process will **terminate** else the process will go back to the **ready queue** and waits for the next turn to complete the execution.

Priority -  
In Priority scheduling, there is a priority number assigned to each process. In some systems, the lower the number, the higher the priority. While, in the others, the higher the number, the higher will be the priority. The Process with the higher priority among the available processes is given the CPU. There are two types of priority scheduling algorithm exists. One is **Preemptive** priority scheduling while the other is **Non Preemptive** Priority scheduling.  
The difference between preemptive priority scheduling and non preemptive priority scheduling is that, in the preemptive priority scheduling, the job which is being executed can be stopped at the arrival of a higher priority job.

The priority number assigned to each of the process may or may not vary. If the priority number doesn't change itself throughout the process, it is called **static priority**, while if it keeps changing itself at the regular intervals, it is called **dynamic priority**.

Rate Monotonic Scheduling   
**Rate monotonic scheduling** is a priority algorithm that belongs to the static priority scheduling category of [Real Time Operating Systems](https://www.geeksforgeeks.org/real-time-operating-system-rtos/). It is preemptive in nature. The priority is decided according to the cycle time of the processes that are involved. If the process has a small job duration, then it has the highest priority. Thus if a process with highest priority starts execution, it will preempt the other running processes. The priority of a process is inversely proportional to the period it will run for.

A set of processes can be scheduled only if they satisfy the following equation



Earliest Deadline First.  
EDF uses priorities to the jobs for scheduling. It assigns priorities to the task according to the absolute deadline. The task whose deadline is closest gets the highest priority. The priorities are assigned and changed in a dynamic fashion. EDF is very efficient as compared to other scheduling algorithms in real-time systems. It can make the CPU utilization to about 100% while still guaranteeing the deadlines of all the tasks.

Inter-process communication:

Message queues

Mailbox

Event timers

Task synchronization: Need

When two or more process cooperates with each other, their order of execution must be preserved otherwise there can be conflicts in their execution and inappropriate outputs can be produced.

A cooperative process is the one which can affect the execution of other process or can be affected by the execution of other process. Such processes need to be synchronized so that their order of execution can be guaranteed.

The procedure involved in preserving the appropriate order of execution of cooperative processes is known as Process Synchronization.

Issues -

Deadlock  
Every process needs some resources to complete its execution. However, the resource is granted in a sequential order.

1. The process requests for some resource.
2. OS grant the resource if it is available otherwise let the process waits.
3. The process uses it and release on the completion.

A Deadlock is a situation where each of the computer process waits for a resource which is being assigned to some another process. In this situation, none of the process gets executed since the resource it needs, is held by some other process which is also waiting for some other resource to be released.

Let us assume that there are three processes P1, P2 and P3. There are three different resources R1, R2 and R3. R1 is assigned to P1, R2 is assigned to P2 and R3 is assigned to P3.

After some time, P1 demands for R1 which is being used by P2. P1 halts its execution since it can't complete without R2. P2 also demands for R3 which is being used by P3. P2 also stops its execution because it can't continue without R3. P3 also demands for R1 which is being used by P1 therefore P3 also stops its execution.

In this scenario, a cycle is being formed among the three processes. None of the process is progressing and they are all waiting. The computer becomes unresponsive since all the processes got blocked.

**Race condition**

A Race Condition typically occurs when two or more threads try to read, write and possibly make the decisions based on the memory that they are accessing concurrently.

**live Lock-**

Livelock occurs when two or more processes continually repeat the same interaction in response to changes in the other processes without doing any useful work. These processes are not in the waiting state, and they are running concurrently. This is different from a deadlock because in a deadlock all processes are in the waiting state.

Each of the two processes needs the two resources and they use the polling primitive enter\_reg to try to acquire the locks necessary for them. In case, the attempt fails, the process just tries again. If process A runs first and acquires resource 1 and then process B runs and acquires resource 2, no matter which one runs next, it will make no further progress, but neither of the two processes blocks

**Types of scheduling**

**1.Cooperative scheduling**

In this kind of scheduling, no task blocks a running task ie all tasks cooperate with others.*Co-operative* , where the tasks themselves run until they tell the OS when they can be context switched (for I/O, etc.). This algorithm can be implemented with the FCFS.

Example: Washing Machine

Task Get User Input--------Task Fill Water------Task Wash Cycle 1------Task Drain cycle 1----Task

wash cycle 2------Task drain cycle 2------Task dry cycle

**2. Time Slicing**

* In this scheduling method, each task is given a time–slice is assigned to each task ( timer is used)
* The time-slice depends on the type of task
* Works well for equal priority tasks

**3. Round Robin Scheduling**

To [schedule processes](https://en.wikipedia.org/wiki/Process_scheduler) fairly, a round-robin scheduler generally employs [time-sharing](https://en.wikipedia.org/wiki/Time-sharing), giving each job a time slot or *quantum*[[5]](https://en.wikipedia.org/wiki/Round-robin_scheduling#cite_note-McConnell2004-5) (its allowance of CPU time), and interrupting the job if it is not completed by then. The job is resumed next time a time slot is assigned to that process. If the process terminates or changes its state to waiting during its attributed time quantum, the scheduler selects the first process in the ready queue to execute. In the absence of time-sharing, or if the quanta were large relative to the sizes of the jobs, a process that produced large jobs would be favored over other processes.

Round-robin algorithm is a pre-emptive algorithm as the scheduler forces the process out of the CPU once the time quota expires.

For example, if the time slot is 100 milliseconds, and *job1* takes a total time of 250 ms to complete, the round-robin scheduler will suspend the job after 100 ms and give other jobs their time on the CPU. Once the other jobs have had their equal share (100 ms each), *job1* will get another allocation of [CPU](https://en.wikipedia.org/wiki/CPU) time and the cycle will repeat. This process continues until the job finishes and needs no more time on the CPU.

**4. Priority-based scheduling**

In this method, tasks have a defined priority parameter and the scheduler schedules by

looking at the priority of the ready tasks.

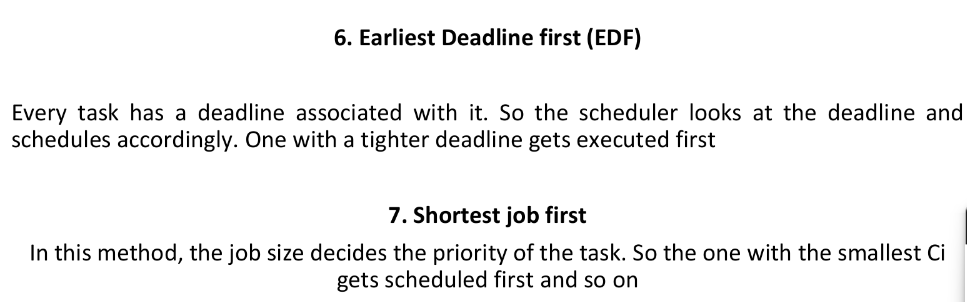
Also pre-emption happens. For example: If a higher priority task becomes ready while a

lower priority task is running, the lower priority task gets preempted( blocked)

Most real-time systems employ the Priority-based scheduling method.

**5. FCFS / FIFO**

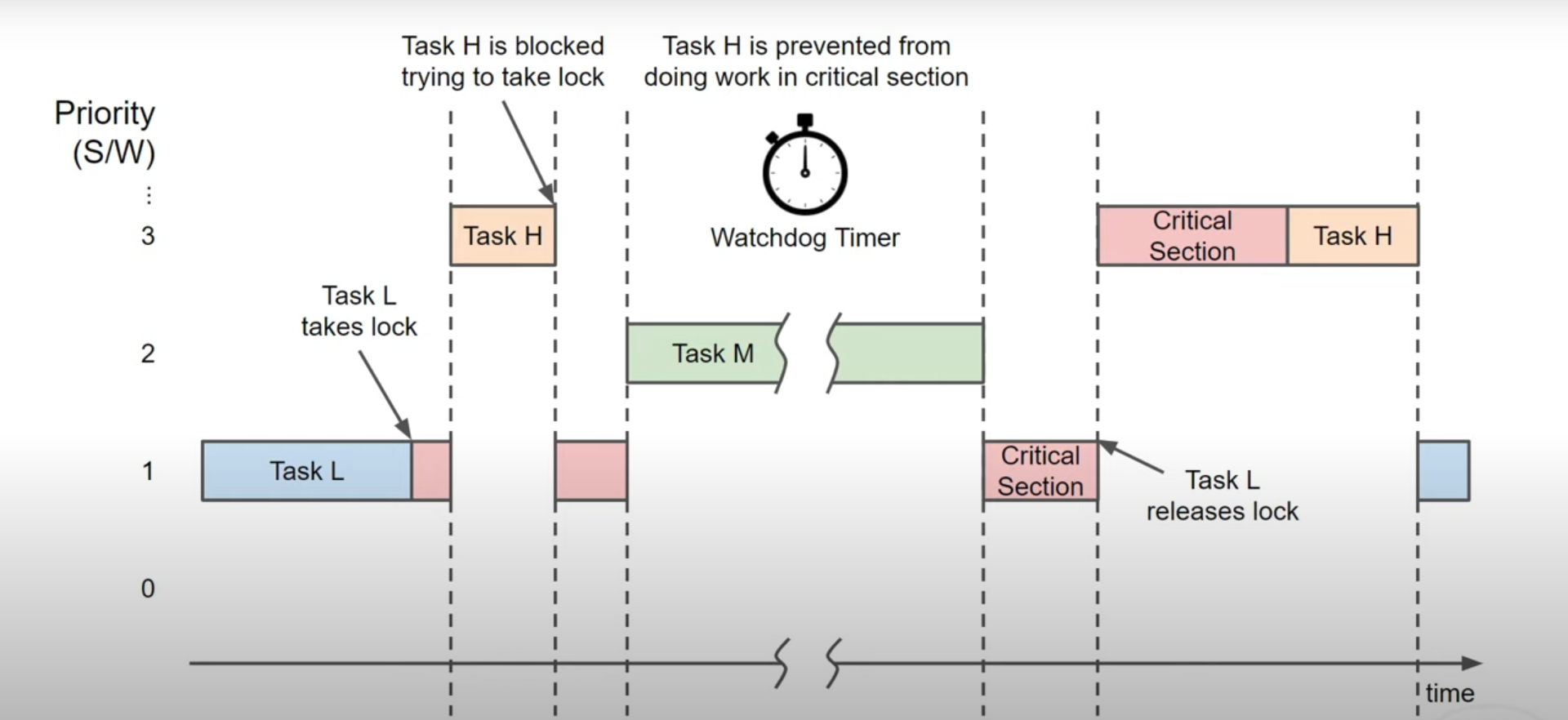
*First Come First Served (FCFS)/Run-To-Completion* , where tasks in the READY queue are executed in the order they entered the queue and where these tasks are run until completion when they are READY to be run (see Figure 9-19). Here, nonpre-emptive means there is no BLOCKED queue in an FCFS scheduling design. The response time of a FCFS algorithm is typically slower than other algorithms.



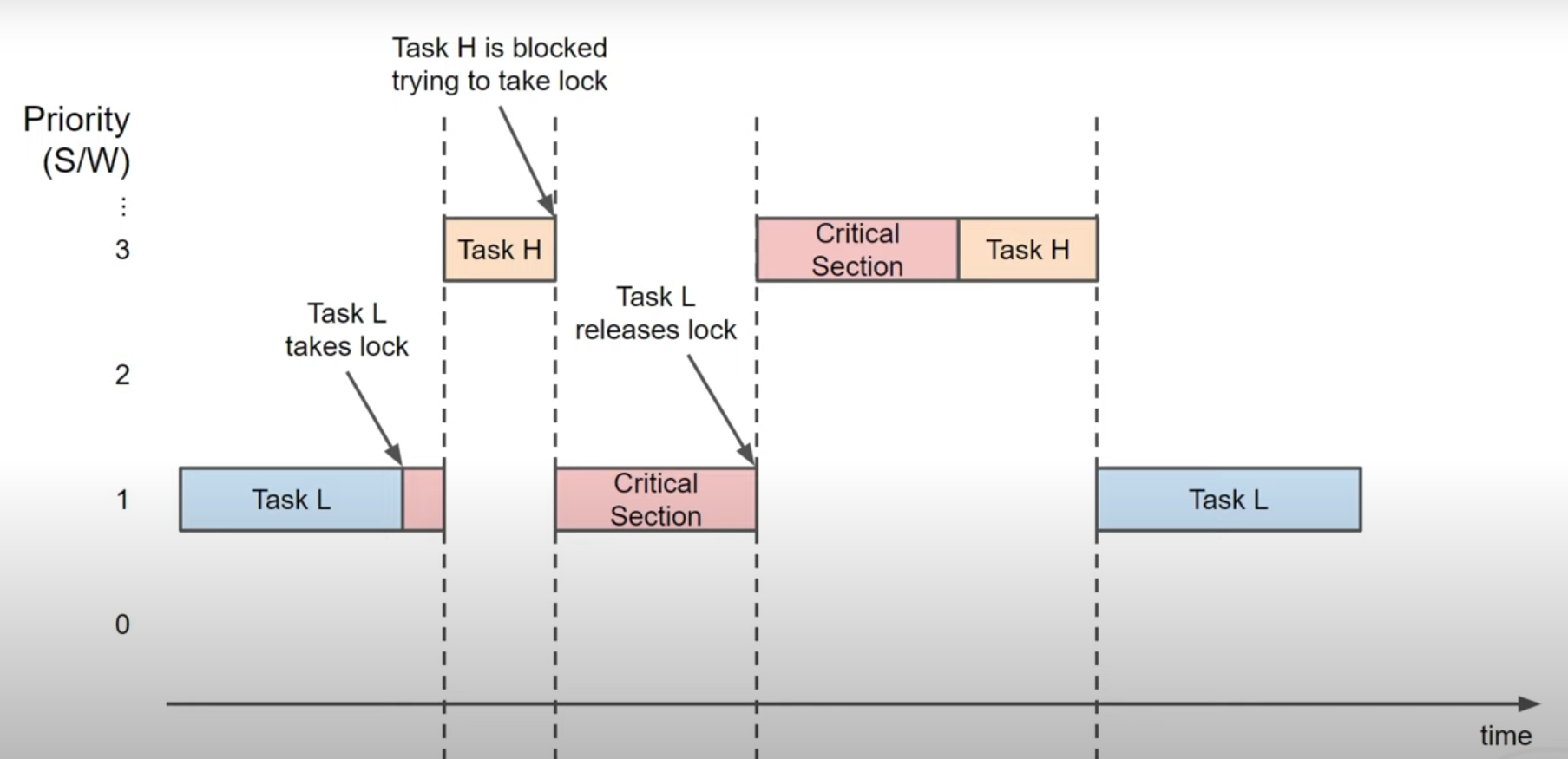
Types of priority inversion: **Pending**

And ways to mitigate it: **Pending**

**Unbounded Priority inversion**

****

**Bounded Priority inversion**

****

**Shared Data Problem: Pending**