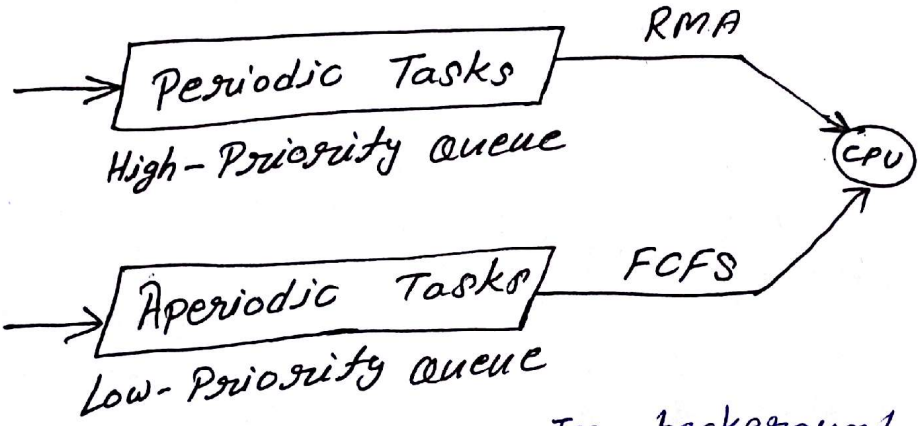
✳ Aperiodic Task Scheduling :-

All computation activities are either aperiodic or periodic. Periodic tasks are time-driven and aperiodic tasks are usually event-driven. Aperiodic tasks may have hard, soft or non-real-time requirements, depending on the specific application.

The simplest method to handle a set of soft aperiodic activities in the presence of periodic tasks is to schedule them in background; that is when there are no periodic instances ready then execute aperiodic instance.



In background scheduling, two queues are needed to implement the scheduling mechanism: one with a higher priority for periodic tasks and other with lower priority reserved for aperiodic requests. Tasks are taken from the aperiodic queue only when the periodic queue is empty. The activation of a new periodic instance causes any aperiodic tasks to be immediately preempted.

In this diagram RMA algorithm is used for periodic tasks and FCFS for aperiodic tasks.

✳ Polling Server :-

The average response time of aperiodic tasks can be improved with respect to background scheduling through the use of a server. A server is characterized by a period P_s and a computation time C_s or e_s, called server capacity.

The polling server, at regular time interval equal to the period P_s, it becomes active and serves any pending aperiodic requests within the limit of its capacity C_s. If no aperiodic requests are pending, it suspends itself until the beginning of its next period. If any aperiodic request arrives just after the server has suspended, it must wait until the beginning of the next polling period, when the server capacity is ~~set~~ replenished at its full value.

EX

$T_i = (C_i \text{ or } e_i, P_i)$

$T_1 = (1, 4)$

$T_2 = (2, 6)$

$T_s = (2, 5) \rightarrow$ (server capacity, Period)

Aperiodic Request ⇒	Request Time	Execution Time
	2	2
	8	1
	12	2

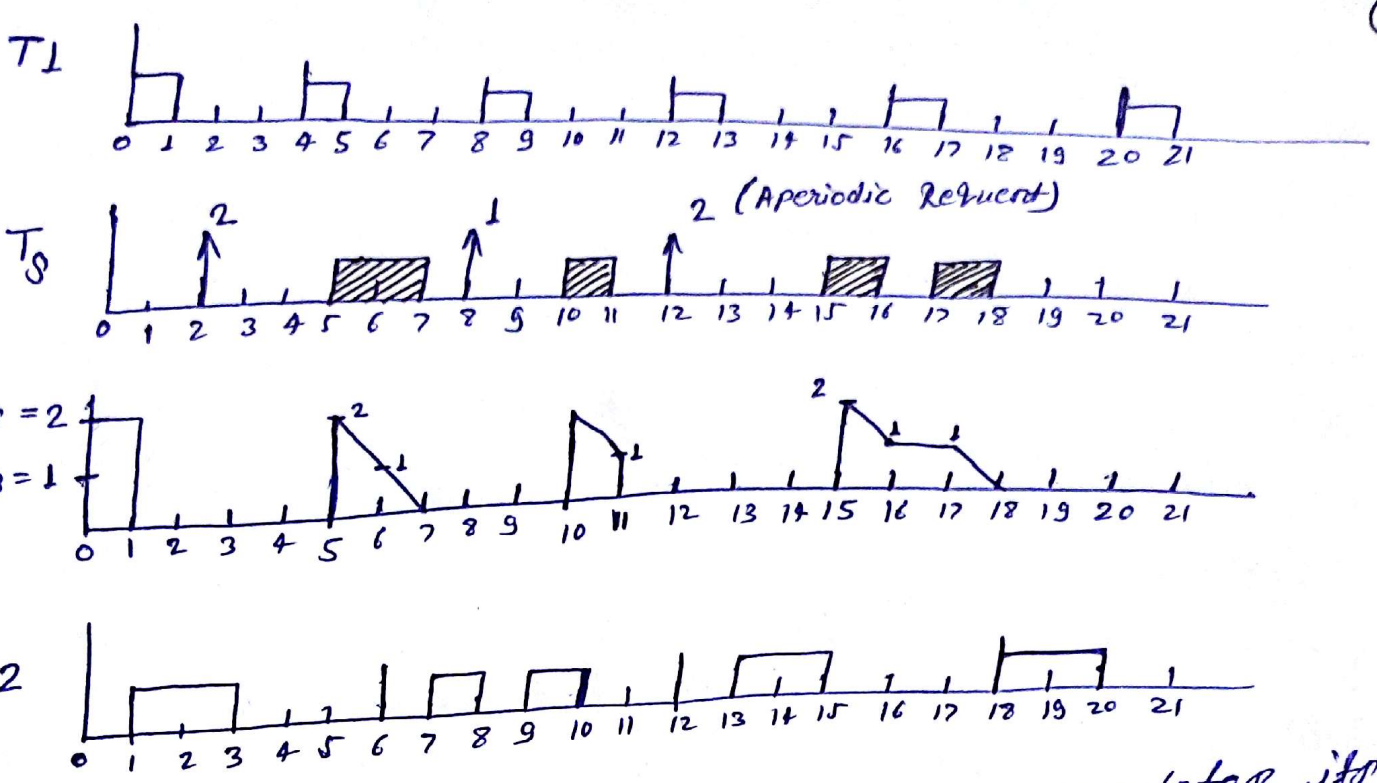
~~Ex~~ Schedule the tasks using RMA and polling server for aperiodic requests.

Sol^n

In RMA ⇒ Period ↓ ⇒ Priority ↑

So $T_1 > T_S > T_2$

In this example, the polling server has period $(P_s) = 5$ and a capacity $C_s = 2$. At time $t = 0$, the processor is assigned to task T_1, which is the highest priority task according to RMA.



At time $t=1$, T1 completes its execution and the processor is assigned to PS. However, since no aperiodic requests are pending, the server suspends itself and its capacity is used by periodic tasks. The aperiodic request arriving at time $t=2$ cannot receive immediate service but must wait until the beginning of the second server period ($t=5$). At this time, the capacity is replenished at its full value ($C_s = 2$), and used to serve the aperiodic task until completion. The capacity has been totally consumed, thus the server becomes idle.

The second aperiodic request only uses half of the server capacity, the remaining half is discarded because no other aperiodic tasks are pending. At $t=16$, the third aperiodic request is preempted by task T1, and the server capacity is preserved, because aperiodic not completed its execution.

* Deferrable Server :-

The Deferrable server (DS) algorithm is a service technique introduced by Lehoczky, Sha and Strosnider. It is a Bandwidth preservable server. It improve the average response time of aperiodic requests with respect to polling server. Unlike Polling, DS preserves its capacity if no requests are pending; and the capacity is maintained until the end of the period. So that aperiodic requests can be serviced at the same server's priority at anytime. At the beginning of any server period, the capacity is replenished at its full value.

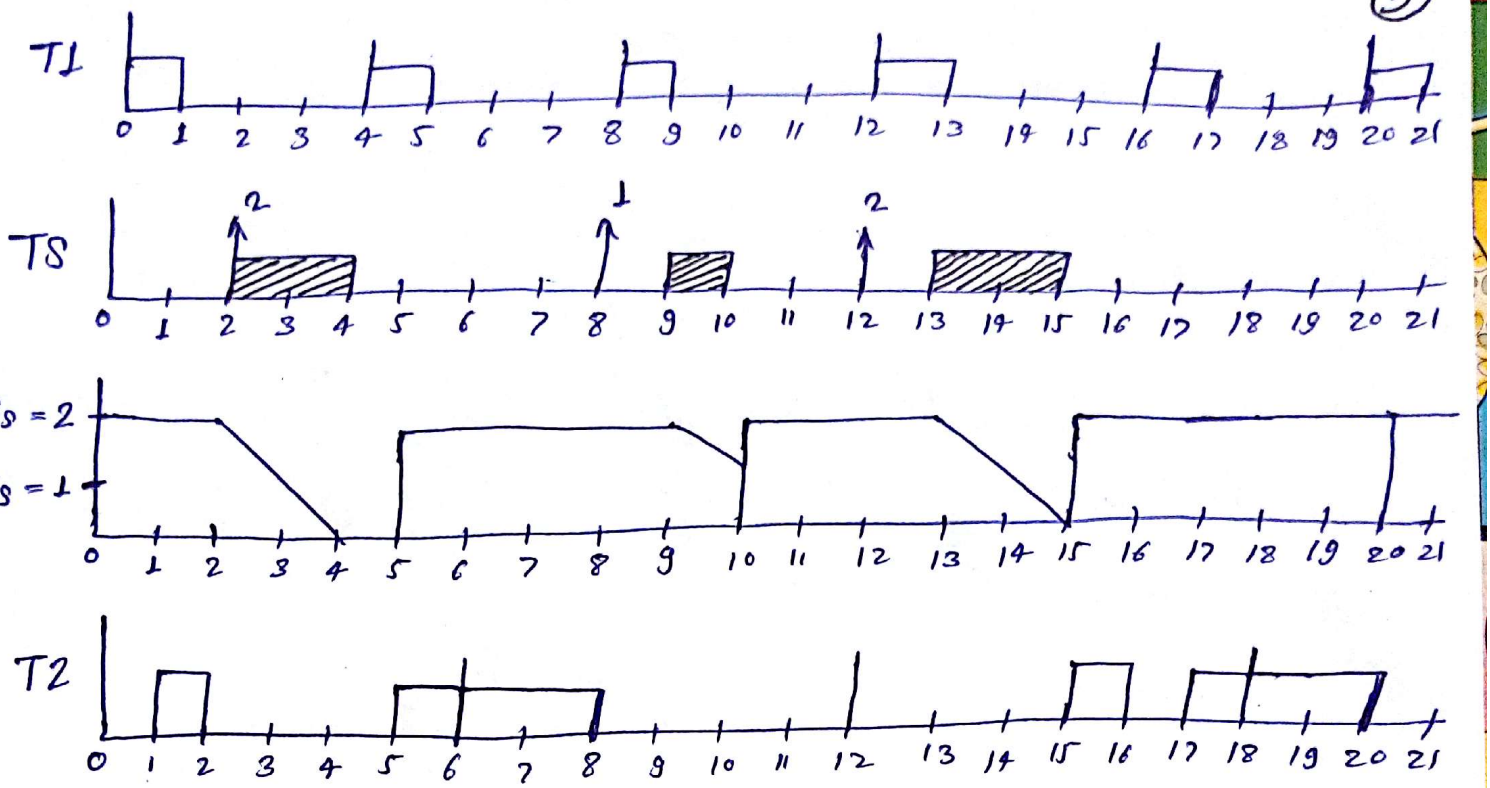
EX Example of Polling server.

Sol^n In RMA Period ↓ ⇒ Priority ↑

$T1 > TS > T2$

In DS algorithm, using the same task set and the same server parameters ($C_s = 2$, $P_s = 5$), At time $t = 1$, when T1 is completed, no aperiodic requests are pending; hence the processor is assigned to task T2. The DS capacity is not used by periodic tasks, but it is preserved for future aperiodic arrivals.

Thus when the first aperiodic requests arrives at time $t = 2$, it receives immediate service. Since the server capacity is exhausted at time $t = 4$, no other requests can be serviced before the next period.



At time $t=5$, C_s is replenished at its full value and preserved until the next arrival. The second request is arrived at time $t=8$, but it is not served immediately, because T_1 is active and has a higher priority.

Thus, DS provides much better aperiodic responsiveness than polling, since it preserves the capacity until it is needed.

* Priority Exchange :-

The Priority Exchange (PE) has a slightly worse performance with respect to DS in terms of aperiodic responsiveness.

Like DS, the PE algorithm uses a periodic server for servicing aperiodic requests. However, it differs from DS in the manner in which the capacity is preserved. PE preserves its capacity by

exchanging it for the execution time of a lower Priority periodic task.

At the beginning of each server Period, the capacity is replenished at its full value. If aperiodic requests are pending and the server is ready with the highest priority, then the requests are serviced using the available capacity; otherwise C_s is exchanged for the execution time of the active Periodic task.

When a priority exchange occurs between a periodic task and a PE server, the server capacity (C_s) is not lost but preserved at a lower priority. If no aperiodic requests arrive to use the capacity, priority exchange continues with other lower-priority tasks until either the capacity is used for aperiodic service or no tasks are active.

Ex

$T_i = (C_i \text{ or } e_i, P_i)$

$T_1 = (4, 10)$

$T_2 = (8, 20)$

$T_S = (1, 5) \rightarrow$ (server capacity, Period)

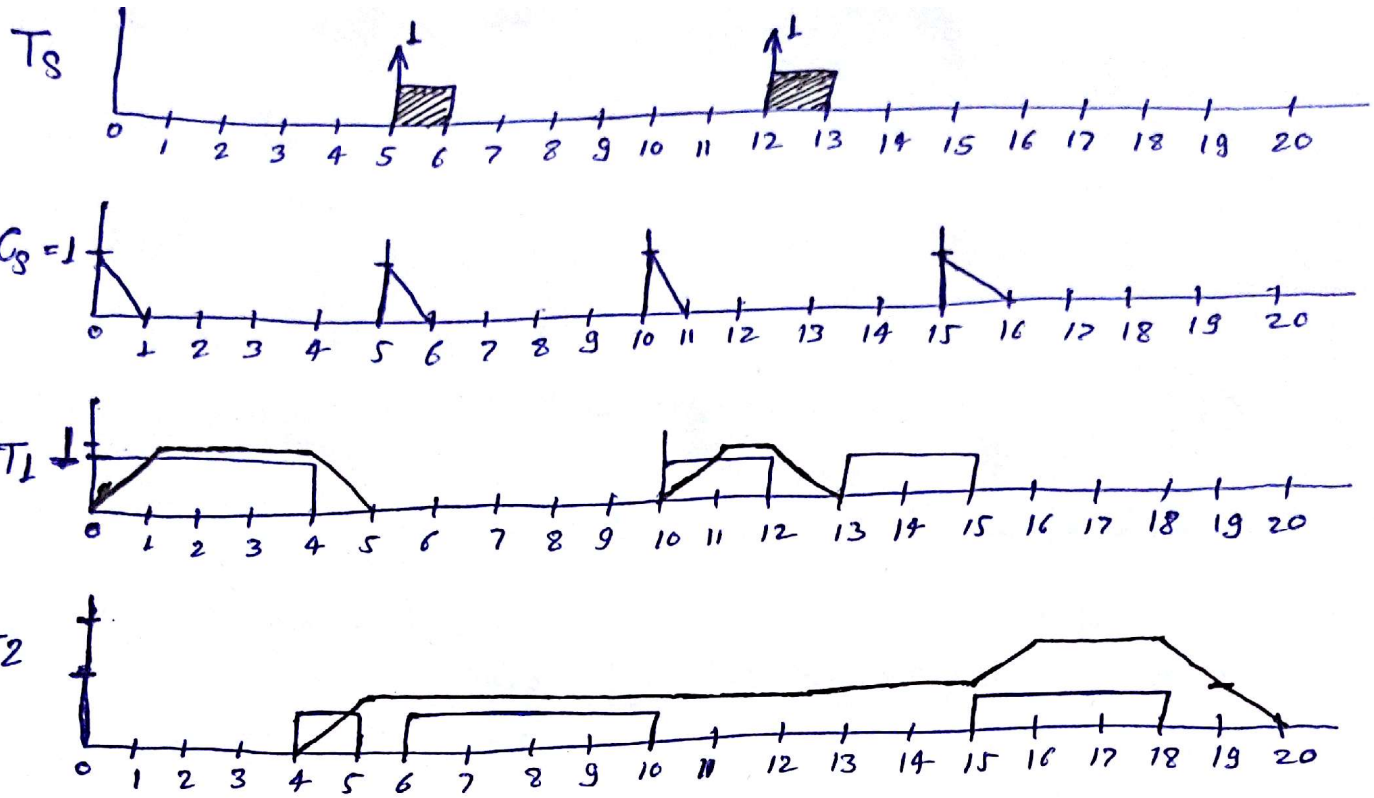
Aperiodic Request ⇒

Request Time	Execution Time
5	1
12	1

Schedule the tasks using Priority Exchange for aperiodic requests.

Soln: Priority sequence: $T_S > T_1 > T_2$

In this example, at time $t=0$, the PE server is brought at its full capacity, but no aperiodic requests are pending, so C_S is exchanged with the execution time of task T_1, which has second priority after T_S. At time $t=4$, T_1 completes and T_2 begins to execute. Again, since no aperiodic tasks



are pending, another exchange takes place between T_1 and T_2. At time $t=5$, the capacity is replenished and it is used to execute the first aperiodic request.

At time $t=10$, the server capacity is replenished ~~at the highest~~ but due to lack of aperiodic request, it is exchange with task T_1.

At time $t=12$, the server capacity accumulated at T_1 is used to execute the second aperiodic request. At time $t=15$, again the server capacity is replenished, but the capacity is exchanged with the running task T_2.

Finally at time $t=18$, the remaining capacity is accumulated at the priority level of task T_2 is gradually discarded because no tasks are active.

* Sporadic Server :-

The Sporadic Server (SS) algorithm is used to ~~create~~ service aperiodic request and like DS, it preserves the server capacity until an aperiodic request occurs. SS algorithm differs from DS in the way it replenished its capacity. In DS and PE, their capacity periodically replenish to its full value at the beginning of each server period; but SS replenished its capacity only after it has been consumed by aperiodic task execution.

In SS algorithm, the capacity is replenished according to the following rule -

RT : It denotes the replenishment time at which the SS capacity will be replenished.

RA : It denotes the replenishment amount that will be added to the capacity at time RT.

t_A : Time at which aperiodic request ~~execute~~ get response.

$RT = t_A + P_S$ → period of SS task

RA = Execution of aperiodic request at t_A.
↓ Total

Ex $T_i = (C_i \text{ or } e_i, P_i)$ Aperiodic Request ⇒

$T_1 = (3, 10)$

$T_2 = (4, 15)$

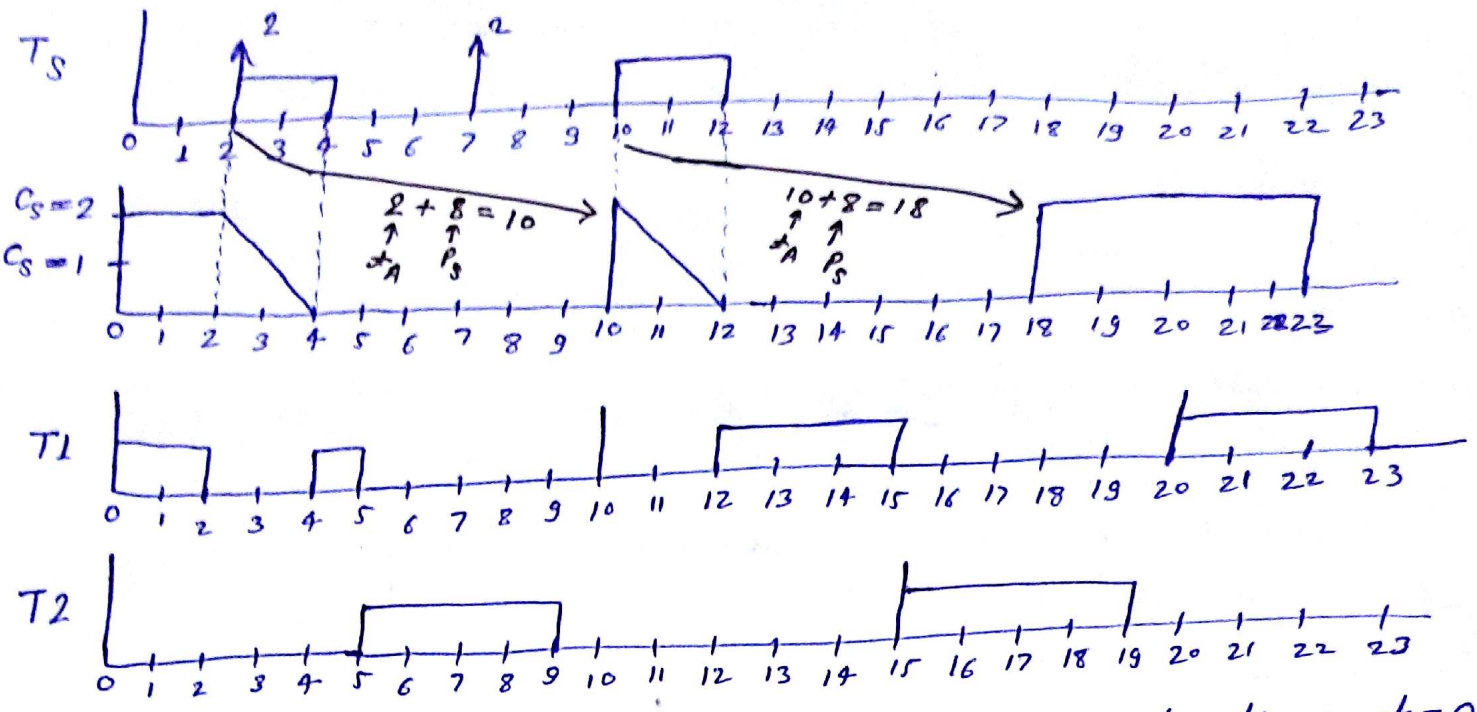
$T_S = (2, 8) \rightarrow$ (server capacity, period)

Request Time	Execution Time
2	2
7	2

Schedule following tasks using sporadic server.

Soln Priority ⇒ Period ↓ ⇒ Priority ↑

So $TS > T1 > T2$



In this example, at time $t=0$, their is no aperiodic request, so second priority task T_1 is scheduled. At time $t=2$, first aperiodic request arrives, which preempt task T_1 due to higher priority and consumes whole server capacity. The server capacity is replenish with amount $RA_1 = 2$ at time $RT_1 = 2 + 8 = 10$.

The second aperiodic request arrives at $t=7$, but server capacity $C_S = 0$. In this case, aperiodic request get response as soon as server capacity $C_S > 0$ and RT_2 will be set at that time.

At $t=10$, the server capacity is replenished and it is consumed by pending second aperiodic request. The second aperiodic request get response at $t_A = 10$, so $RT_2 = 10 + 8 = 18$ and the replenishment amount $RA_2 = 2$, which is server capacity utilized by second aperiodic request.

* Slack Stealing :-

The Slack Stealing algorithm is another aperiodic service technique, which offers improvements in response time over the previous service methods. This algorithm does not create a periodic server for aperiodic task service. Rather it creates passive task, referred to as the Slack Stealer, which attempts to make time for servicing aperiodic tasks by stealing all the processing time from the periodic tasks without causing their deadlines to be missed. If $C_i(t)$ is the remaining computation time of task T_i, at time t, then the slack of a task T_i is -

$$Slack_i(t) = d_i - t - C_i(t)$$

where d_i is the absolute deadline.

Absolute deadline = Release time + Relative deadline

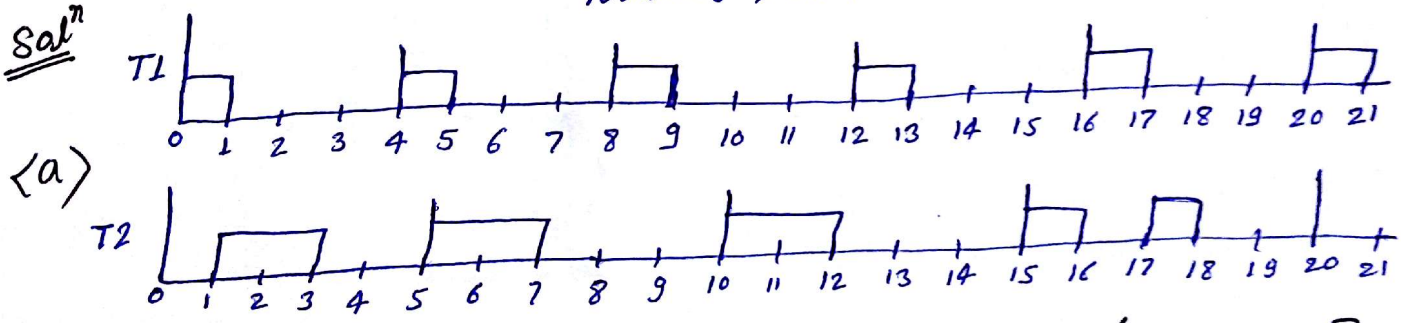
Ex

	e_i	P_i
T_1 ⇒	1	4
T_2 ⇒	2	5

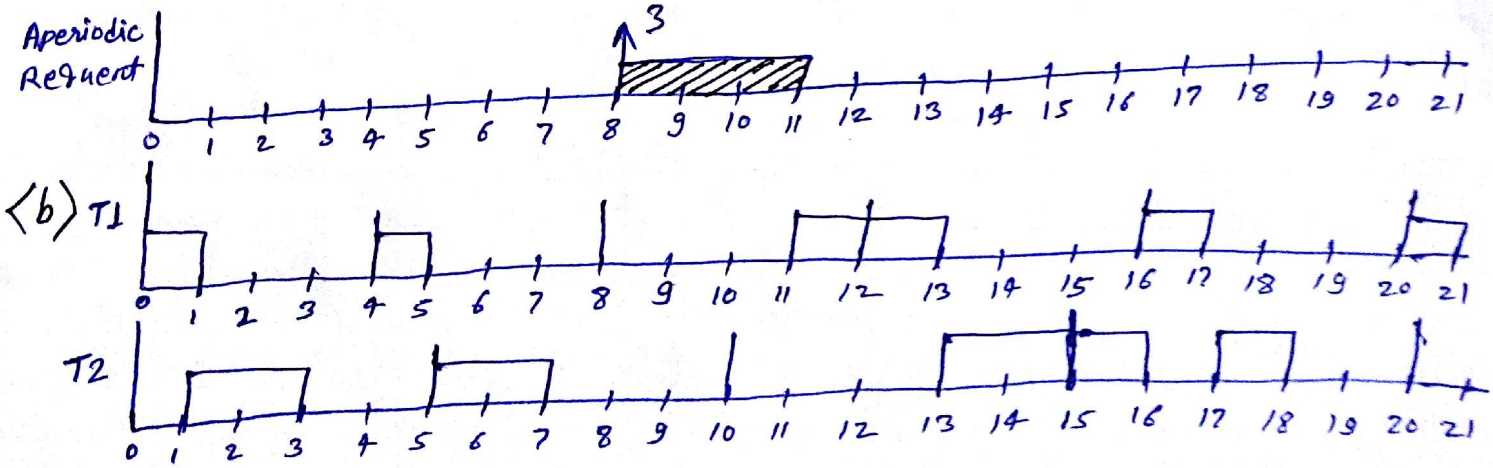
Aperiodic Request ⇒ At time $t_0 = 8$ & $e = 3$

Priority ⇒ T1 > T2

Sol^n



[when ~~there~~ ~~is~~ no aperiodic request arrive]



When an aperiodic request arrives, the slack stealer steals all the available slack from periodic tasks and uses it to execute aperiodic requests as soon as possible. If no aperiodic requests are pending, periodic tasks are normally scheduled by RMA.

In figure (b), at time $t = 8$, an aperiodic ~~and a periodic~~ request arrive. Same time a periodic request of task T_1 is arrive, so aperiodic request take slack time from T_1 –

$$\begin{aligned} Slack_1(t) &= d_1 - t - c_1(t) \\ &= 12 - 8 - 1 \\ &= 3 \end{aligned}$$

So aperiodic request takes slack time three unit and it also has three units execution time. Aperiodic request complete its execution in this slack time after that periodic tasks are sheduled

- Flexible computation
- Flexible Applications
- Imprecise computation model
- Firm Deadline Model

} Please Refer from Text Book.