MSE | MASTER OF SCIENCE IN ENGINEERING

MA_EmbReal

Mix of Periodic and Aperiodic Tasks

Version: 1.2

Serge Ayer – Luca Haab | 17.03.2025 | Cours MSE

Recall: Our Mission

Recall: Our mission

- Program with a mix of periodic / aperiodic tasks
  - Address first scheduling of periodic tasks
  - Add aperiodic tasks
  - Add dependencies among tasks
- Demonstrate that a schedule is feasible given a set of tasks with their constraints and dependencies
  - Use known bounds and elaborate a feasible schedule
  - Compute bounds for blocking times
- Use the appropriate scheduling algorithm in simulation and practice
- Implement a system that meets timing constraints
  - With functional safety concepts
  - With timing constraints watchdogs

Aperiodic Tasks

Aperiodic Tasks

- Aperiodic tasks = arrival time is not predictable
- Scheduling algorithms can be classified using the $\alpha \mid \beta \mid \gamma$ notation (Graham et al., 1979)
  - α: environment (uniprocessor, multiprocessor, distributed, etc..)
  - β: task/resource characteristics (preemptive, independent, etc..)
  - γ: optimality criterion or performance measure (feasible or infeasible schedules)
- Our interest goes towards:
  - α: 1
  - β: dependent, possibly preemptive
  - γ: maximum lateness or the maximum number of late tasks

Copy of the paper: https://mat.uab.cat/~alseda/MasterOpt/79_03_scheduling_survey.pdf

Some typical scenarios

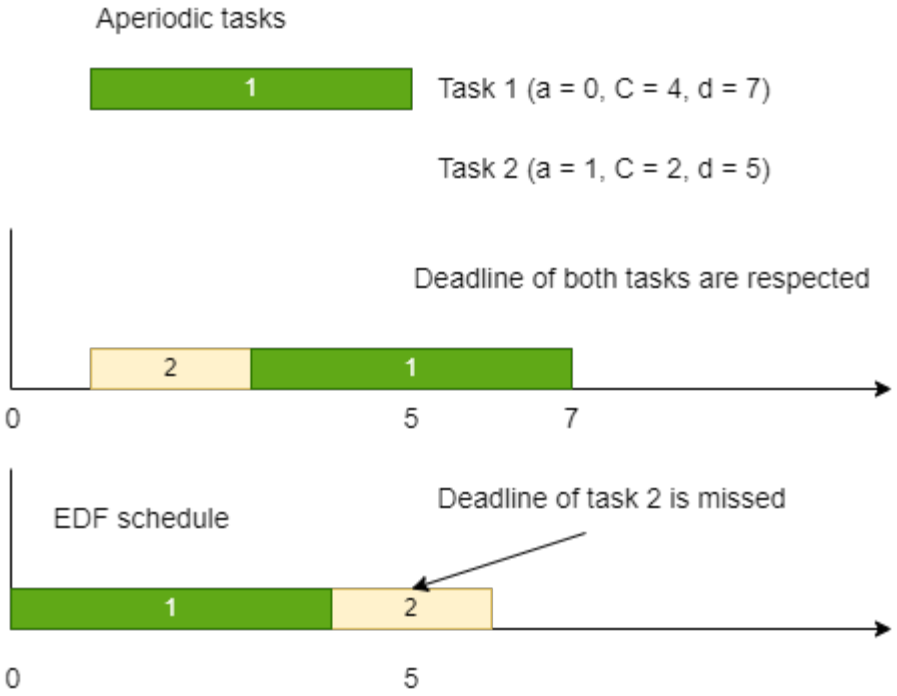
- $\alpha \mid \beta \mid \gamma = 2 \mid$ sync $\mid L_{max}$
  - Schedules aperiodic tasks on a 2-processor systems for minimizing the maximum lateness
  - Tasks are independent tasks with synchronous arrival times with different computation times and deadlines
- $\alpha \mid \beta \mid \gamma = 1 \mid$ preem $\mid L_{max}$
  - Schedules aperiodic tasks on a single processor for minimizing the maximum lateness.
  - Arbitrary arrival times, preemptive scheduling

EDF (Horn's Algorithm)

- Solves the problem of scheduling a set of n independent tasks on a uniprocessor system
  - Tasks have dynamic arrivals
  - Preemption is allowed
- EDF theorem:
  - "Given a set of n independent tasks with arbitrary arrival times, any algorithm that at any instant executes the task with the earliest absolute deadline among all the ready tasks is optimal with respect to minimizing the maximum lateness"
  - This also means that EDF is optimal in the sense of feasibility: if there exists a feasible schedule for a task set, then EDF will find it.
- Guarantee of feasibility:
  - When tasks have dynamic arrival times, guarantying feasibility can only be done dynamically.
  - When a task arrives, it is accepted only if the new task set is feasible

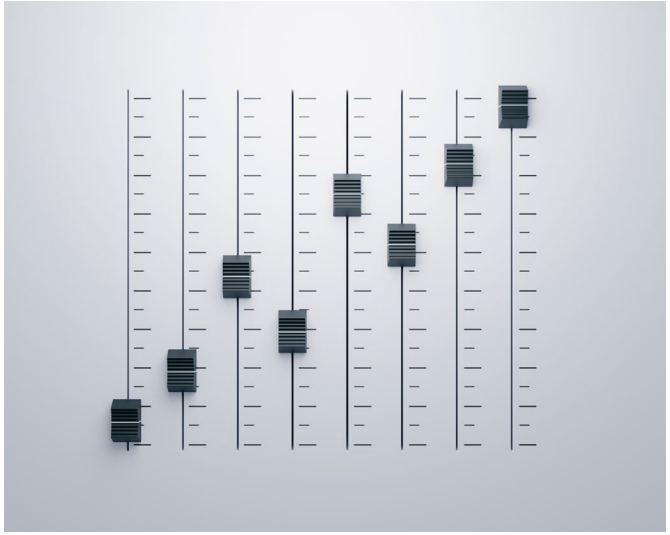
EDF (Horn's Algorithm)

- It is NOT optimal if preemption is not allowed
- When preemption is not allowed, the stated problem becomes NP-hard



Hybrid tasks: Mix of Periodicity and Aperiodicity

- Most practical real-time systems require both types of tasks
  - Periodic tasks are time-driven and execute critical control activities
  - Aperiodic tasks are event-driven and may have hard, firm or soft real-time requirements
  - Some aperiodic tasks may also have no real-time requirement
- Hybrid task sets require to
  - Guarantee the schedulability of all critical tasks
  - Provide good average response times for other tasks



Hybrid tasks: Mix of Periodicity and Aperiodicity

- Off-line guarantee for aperiodic tasks
  - Only with a proper assumption on the maximum arrival rate for critical aperiodic tasks
  - This sets a bound on the load for aperiodic tasks that are thus characterized by a minimum interarrival time
- Without further assumption, on-line guarantee has to be computed
  - Tasks entering the system go through an acceptance test
  - This can only be done for tasks with firm or soft real-time constraints, since tasks with hard real-time constraints cannot be rejected
  - Aperiodic tasks with firm or soft real-time constraints are usually called sporadic tasks

Assumptions for Scheduling of Hybrid Tasks

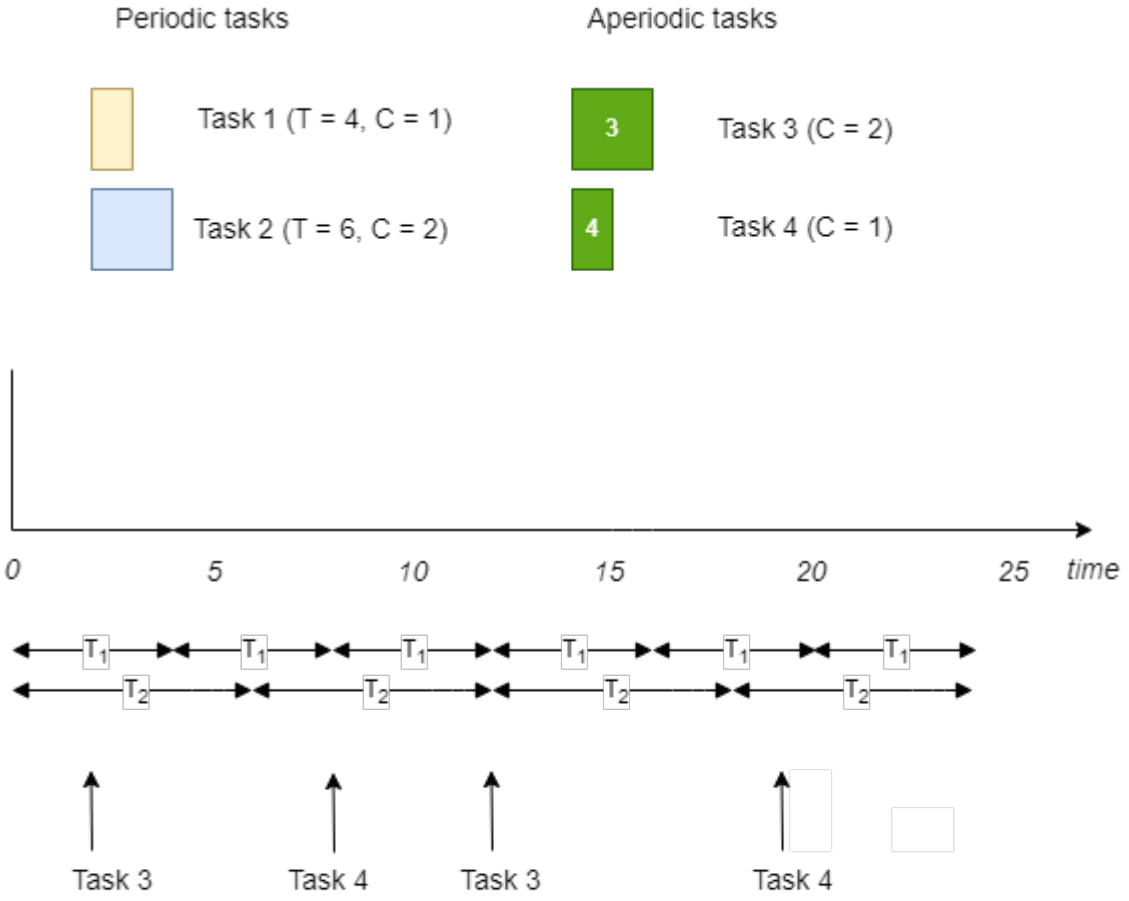
- Periodic tasks are scheduled using RM (fixed priority)
- All periodic tasks starts simultaneously ($\phi_i = 0$) with deadline matching the period ($T_i = D_i$)
- Arrival times of aperiodic tasks are unknown
- Minimum interarrival time of each aperiodic task is equal to its deadline
- All tasks are preemptable

Handling Aperiodic Tasks

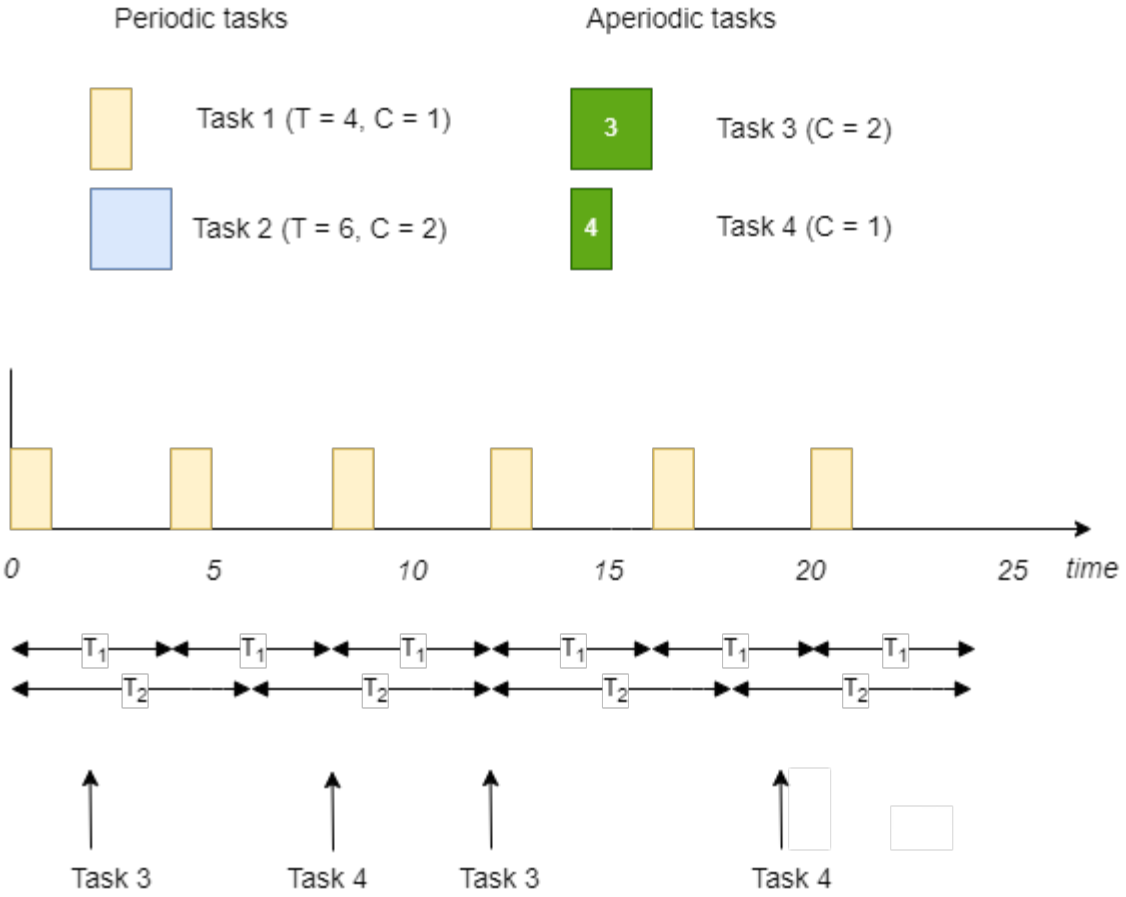
Aperiodic Tasks: Background Scheduling

- Schedule aperiodic tasks in the background
  - in time slots where no periodic task is ready to execute
- Schedulability of periodic tasks is left unchanged
  - Background scheduling does not influence the execution of periodic tasks
- Simple to implement
  - RM as usual (tasks are queued based on their fixed priority)
  - Add a low priority queue (e.g. FCFS) for aperiodic tasks
- Main disadvantage
  - When the load of periodic tasks is high, then the response time of aperiodic tasks can be too long
- Applicable only when
  - The load of periodic tasks is low enough
  - The timing constraints of aperiodic tasks are not stringent

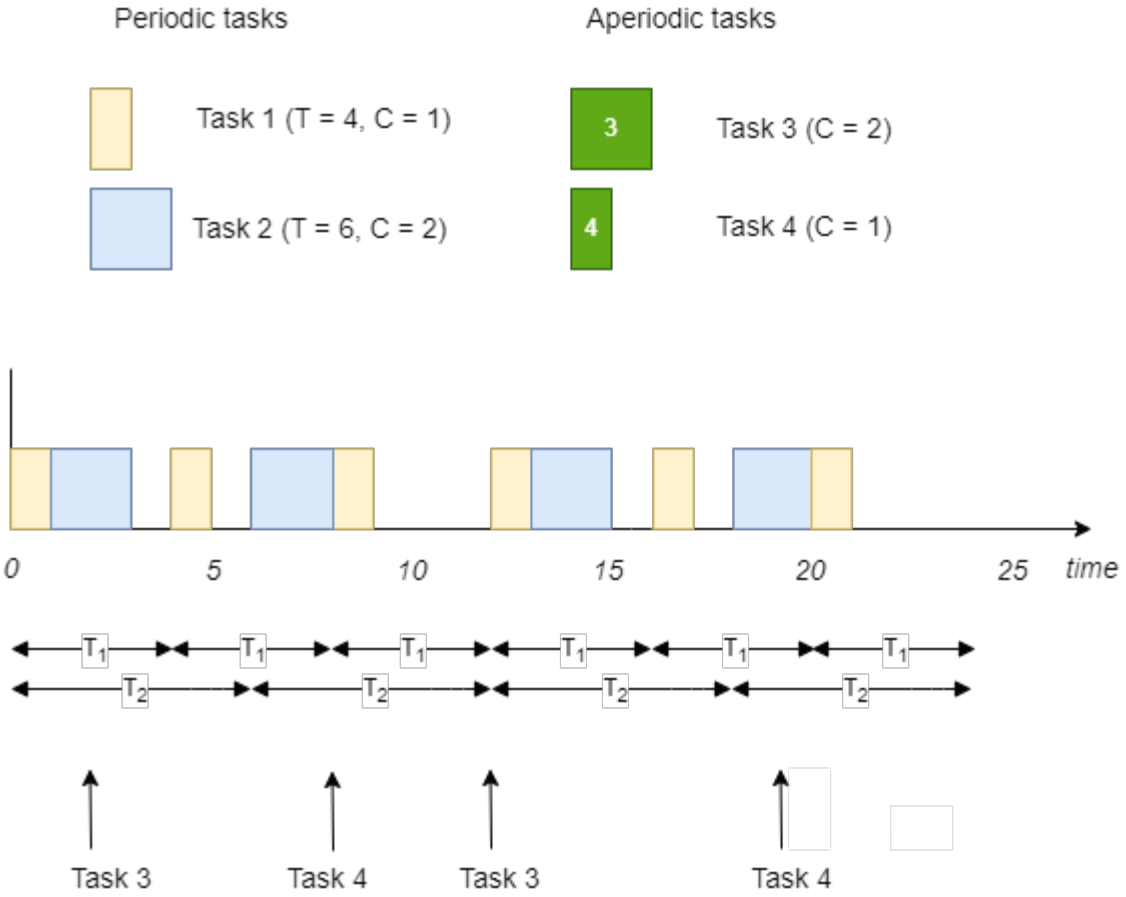
Background Scheduling



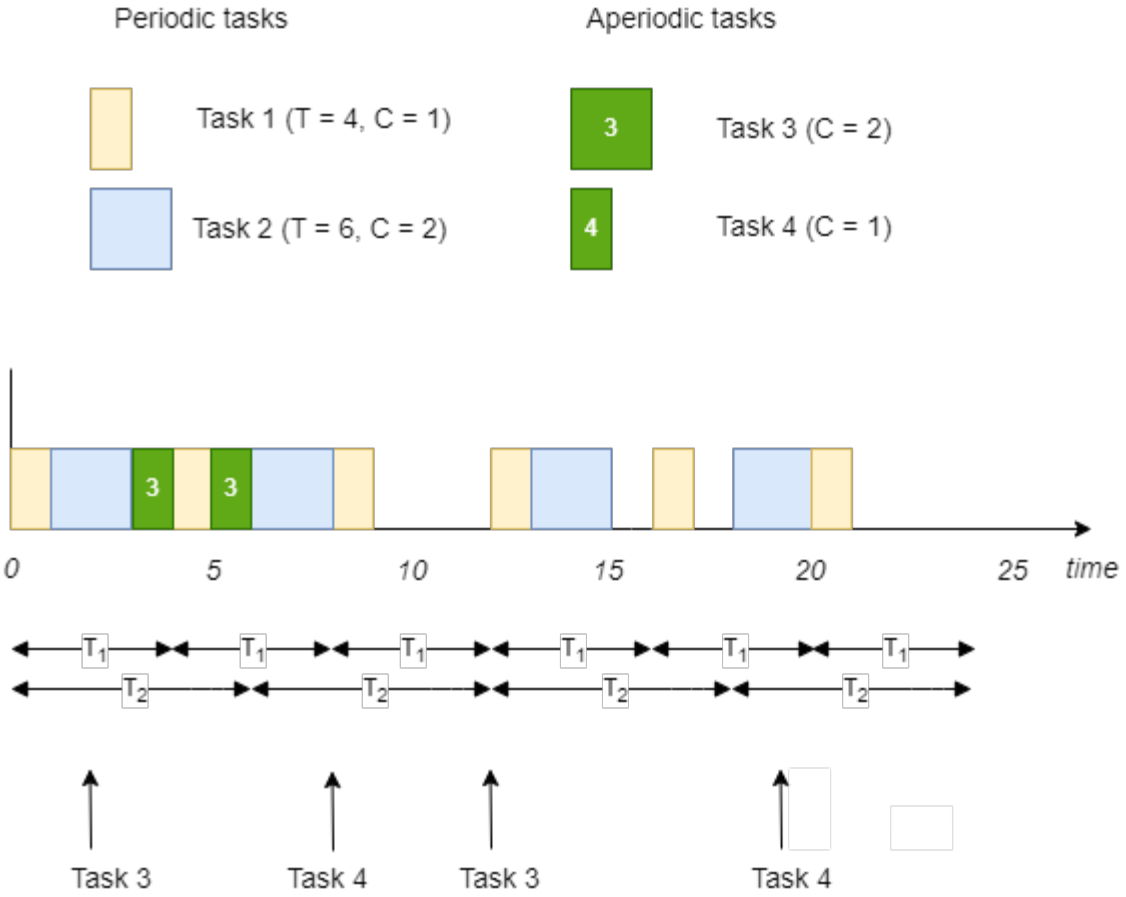
Background Scheduling



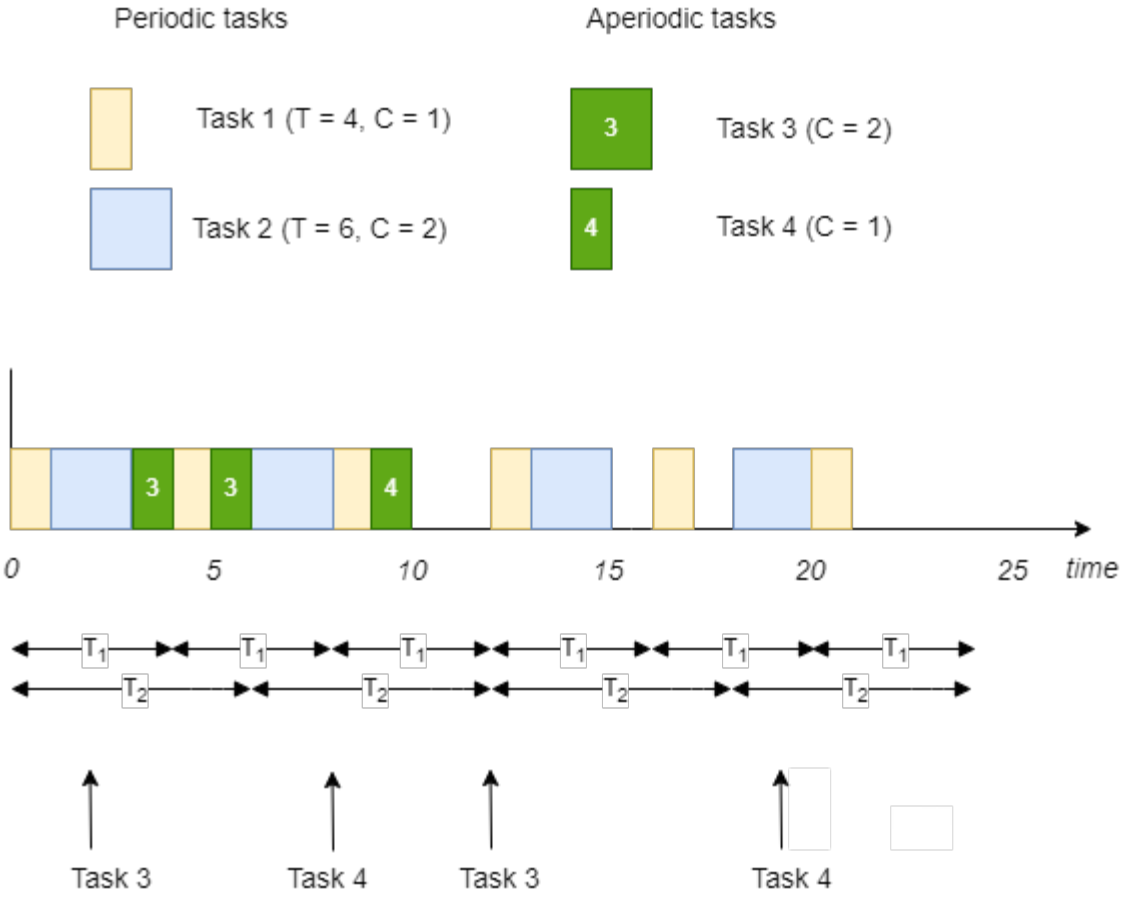
Background Scheduling



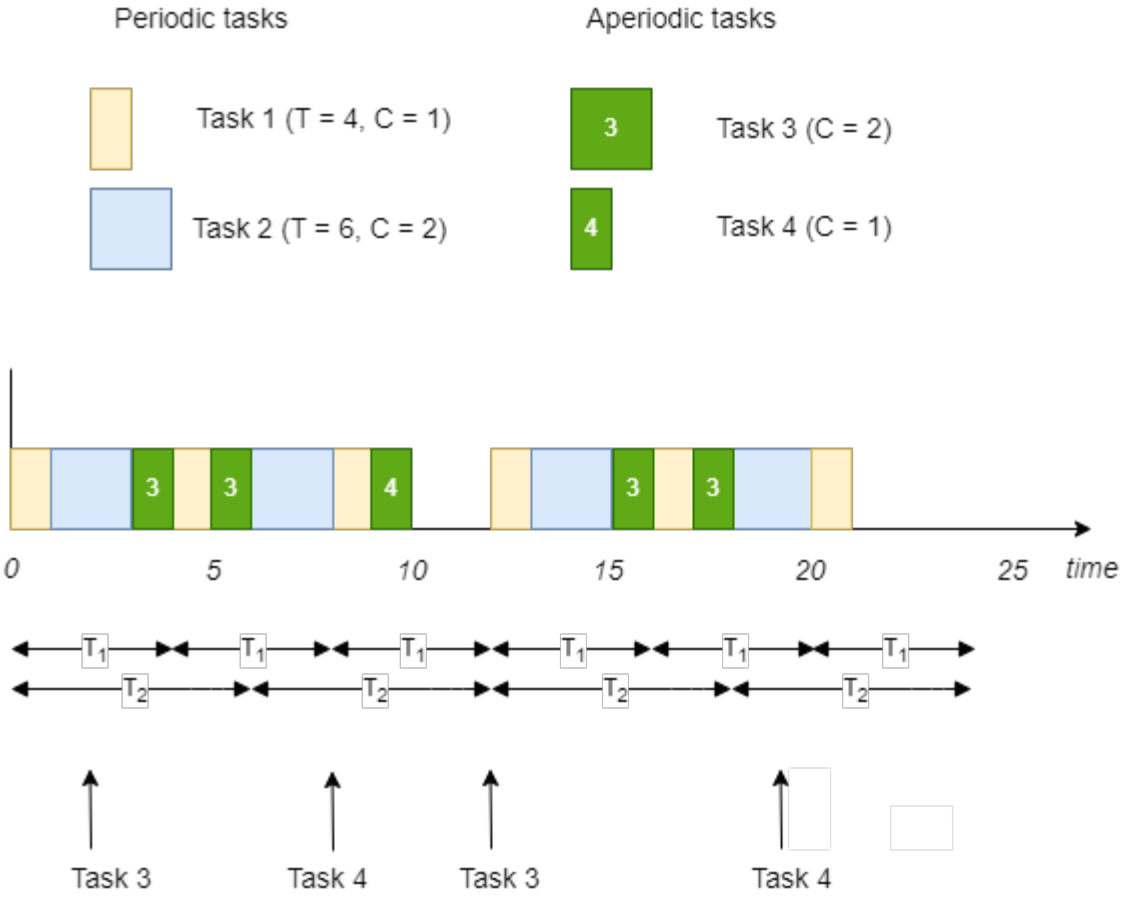
Background Scheduling



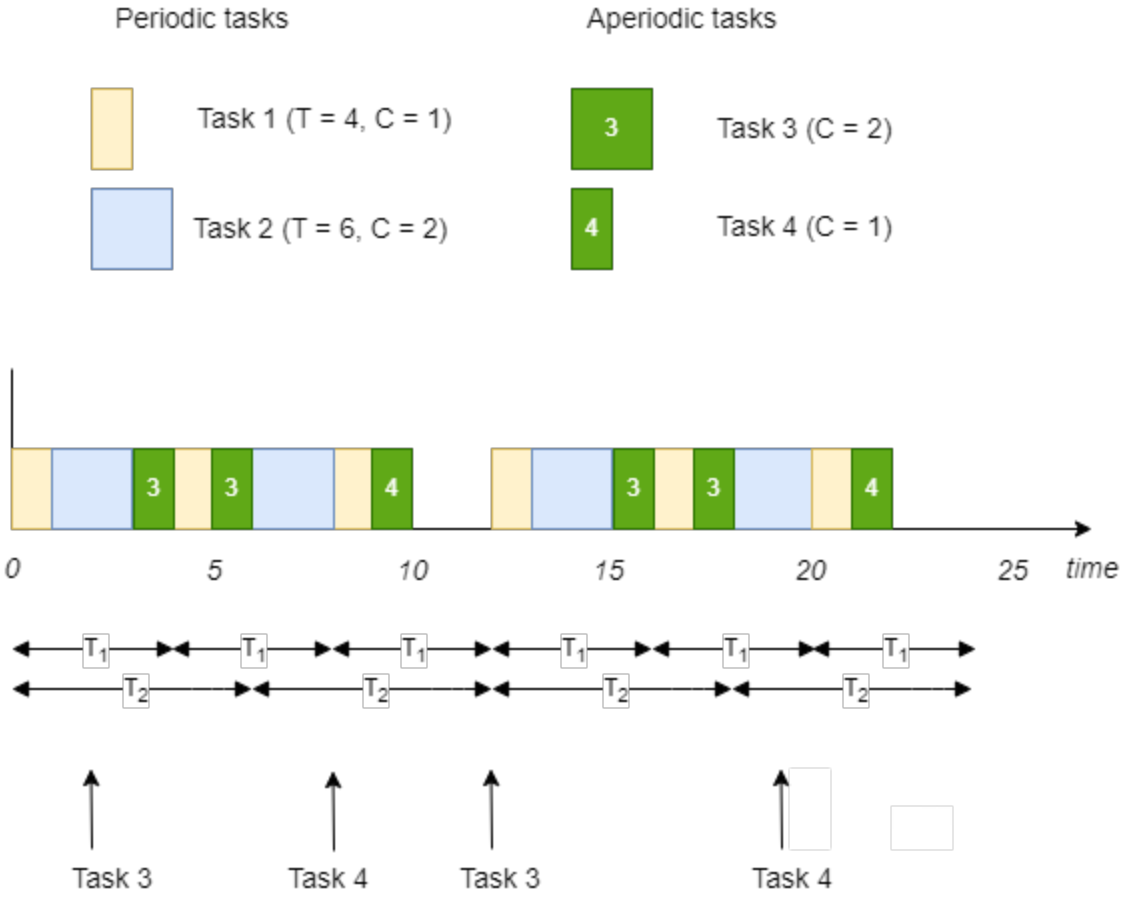
Background Scheduling



Background Scheduling



Background Scheduling



Sporadic Tasks: Servers

- Background scheduling is not appropriate for sporadic tasks
  - Sporadic tasks have to meet some timing constraints
  - Demonstrating the schedulability of aperiodic tasks is not feasible in general with background scheduling
- Improve the average response time of sporadic tasks
  - Add a periodic task for serving sporadic tasks as soon as possible
  - This periodic task has its own period T_s and computation time C_s (server capacity or budget)
- The server task is scheduled with the same algorithm used for periodic tasks
  - Sporadic tasks are served within the limit of the budget
  - The ordering of sporadic tasks does not depend on the scheduling algorithm itself
  - The ordering can be based on arrival time, computation time, deadline or any other criterion.
- Different flavors
  - Depending on priority
  - Depending on how budgets are replenished

Sporadic Tasks: Polling Server

- Server implementing a simple algorithm
  - At interval T_s, the polling server task becomes active for serving the pending sporadic tasks
  - The task is executed within the limit of the computation time C_s
  - If no (further) sporadic task is pending, the polling server task suspends itself
  - In this case, cooperative scheduling is applied: since the task suspends itself, other periodic tasks may execute as soon as the task is suspended
- Schedulability tests
  - In the worst case, it acts as adding a polling server task with the given T_s and C_s to the set of periodic tasks
  - Sporadic tasks do not get more than C_s every T_s, so adapted bounds are valid

$$\sum_{i=1}^{n} \frac{C_i}{T_i} + \frac{C_s}{T_s} \leq (\mathrm{n}+1)\,(2^{\frac{1}{(n+1)}} - 1)$$

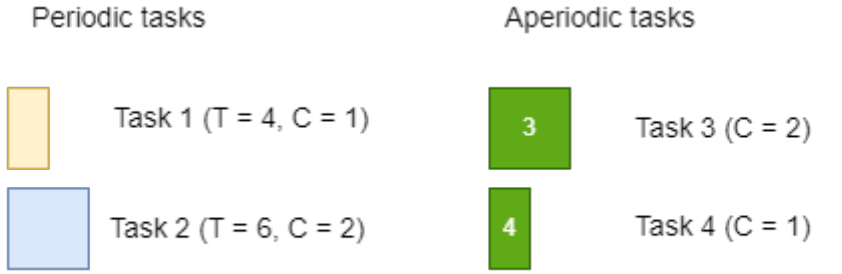
$$\prod_{i=1}^{n} (U_i + 1) \leq \frac{2}{U_s + 1}$$

  - Note that more than one polling server task can be created, with different priorities
  - Schedulability analysis applies in the same way in this case

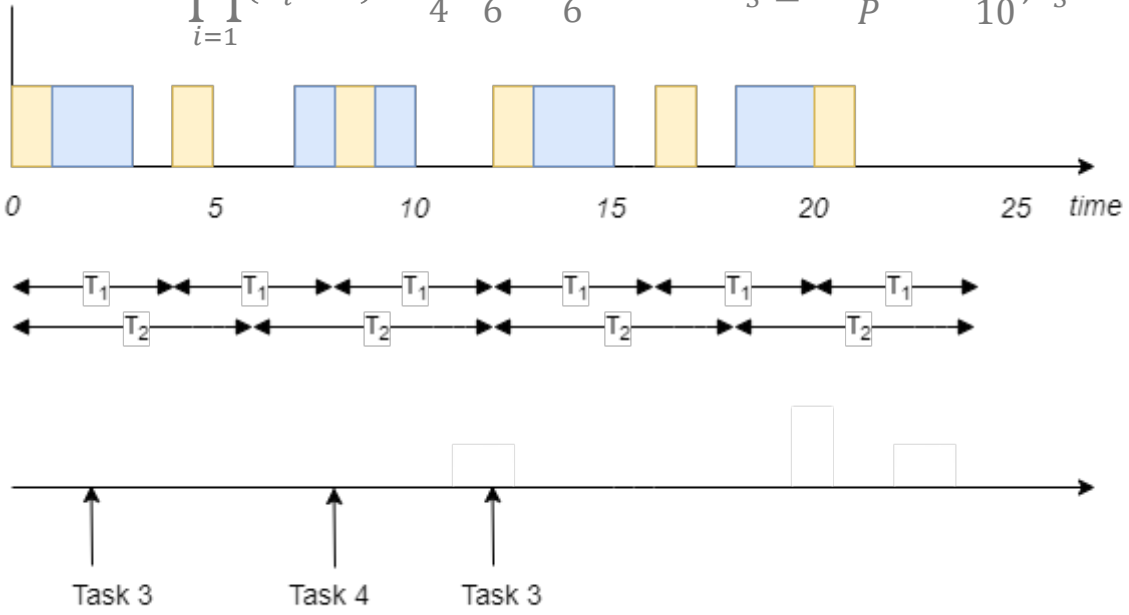
Polling Server Dimensioning

- Given a known set of periodic tasks with a feasible schedule, how can T_s and C_s be computed such that feasibility remains
- In other words, we need to find $U_s = \frac{C_s}{T_s}$ that guarantees feasibility of the augmented task set.
- Bound for U_s: given $P \stackrel{\text{def}}{=} \prod_{i=1}^{n}(U_i + 1)$, we have $P \leq \frac{2}{U_s+1}$ and thus $U_s \leq \frac{2-P}{P}$
- This gives a bound for U_s and an infinite number of (T_s, C_s) pairs
- For improving responsiveness of sporadic tasks, choose the smallest T_s, meaning a task with higher priority under RM
- Balance this choice with runtime overhead
  - It is not useful to choose $T_s \leq T_1$, where T_1 is the shortest period among all periods
  - For privileging the server, choose $T_s = T_1$ and $C_s = U_s T_s$
  - In the case that the server gets the highest priority, the schedulability of a sporadic task (T_a, C_a) can be guaranteed based on (T_s, C_s) (formula relating all values)
  - Note that the schedulability of a sporadic task can be established by computing the maximum response time

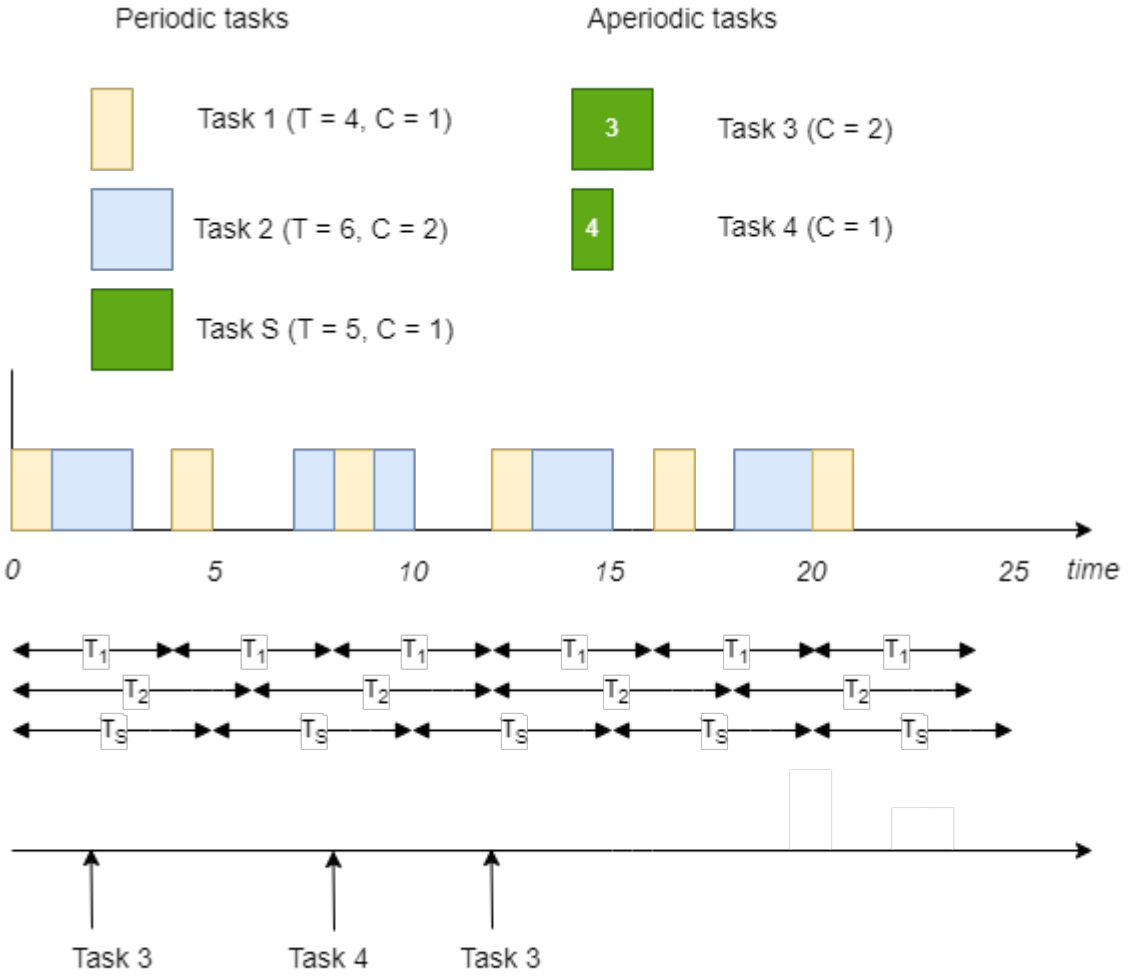
Polling Server



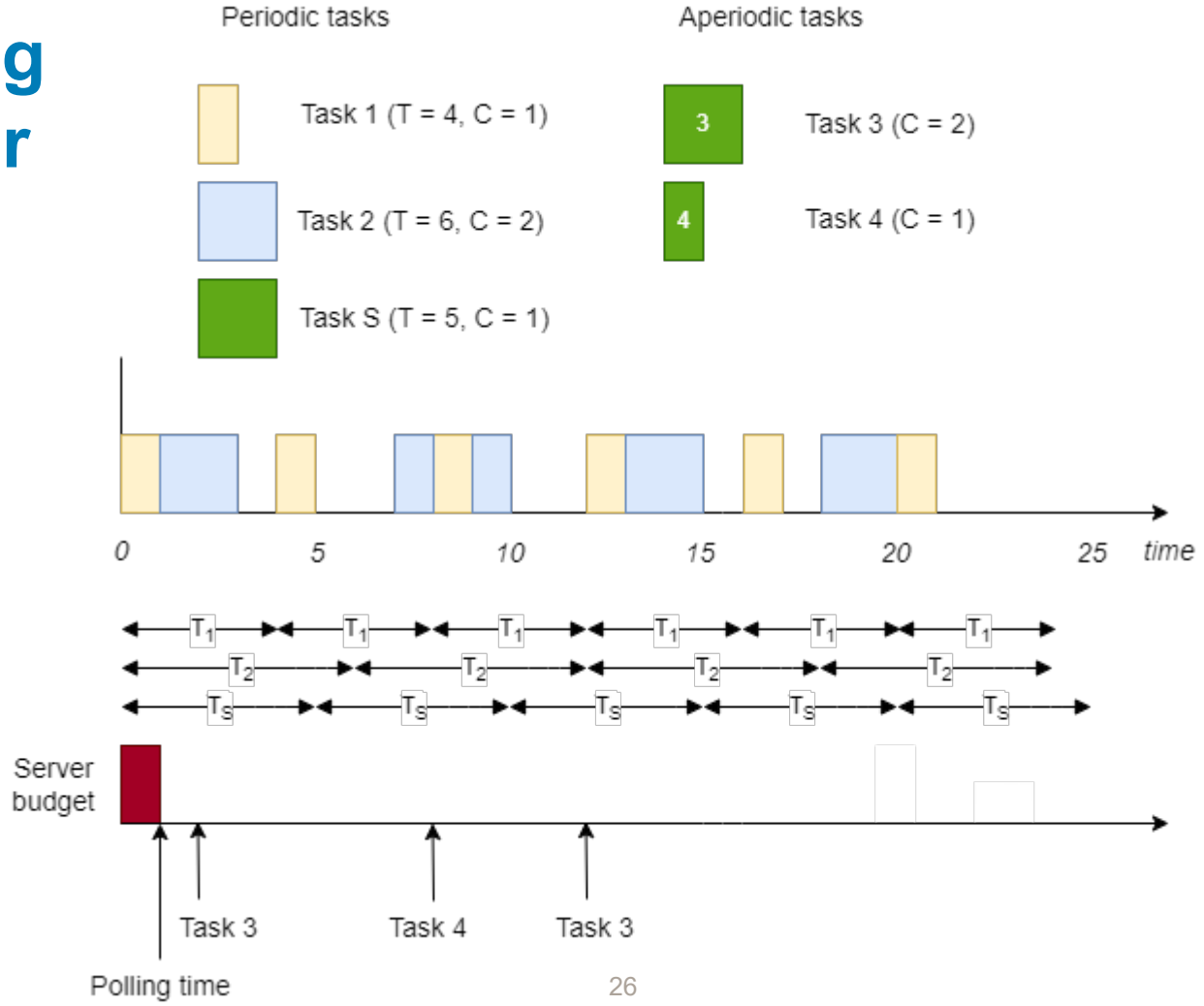
$$P = \prod_{i=1}^{2}(U_i + 1) = \frac{5}{4}x\frac{8}{6} = \frac{10}{6} \leq 2 \qquad U_S \leq \frac{2 - P}{P} = \frac{2}{10}, T_S = 5, C_S = 1$$



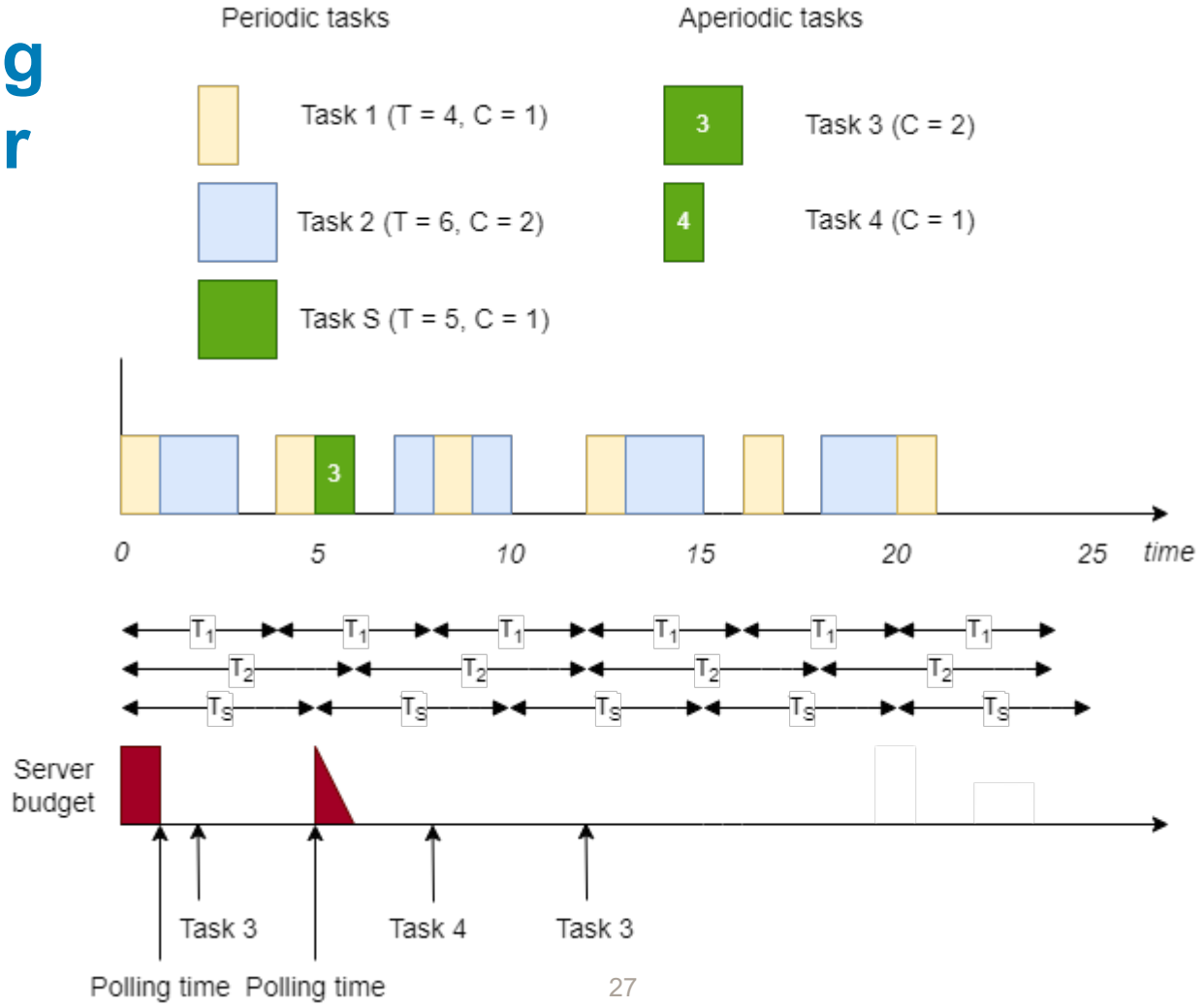
Polling Server



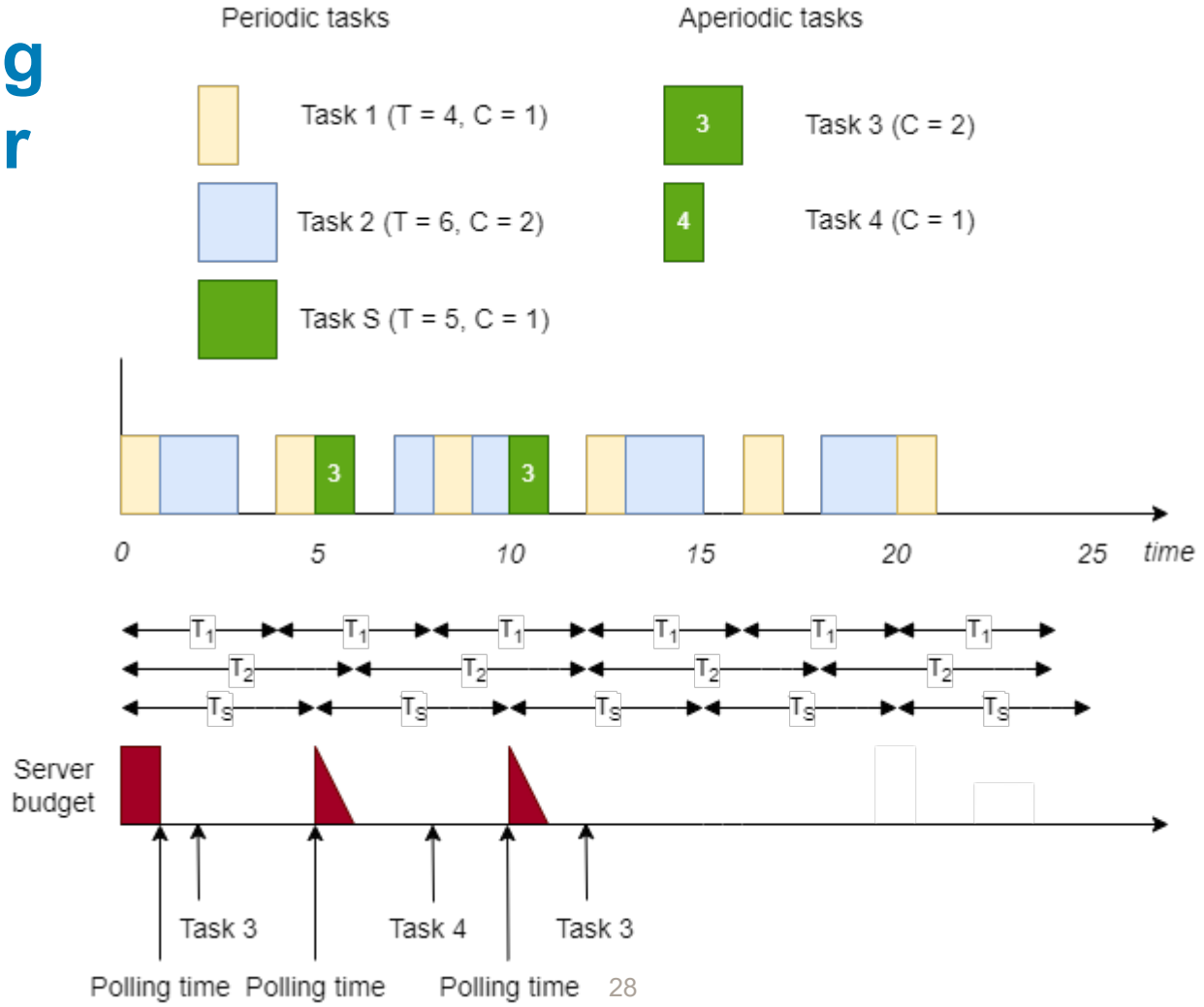
Polling Server



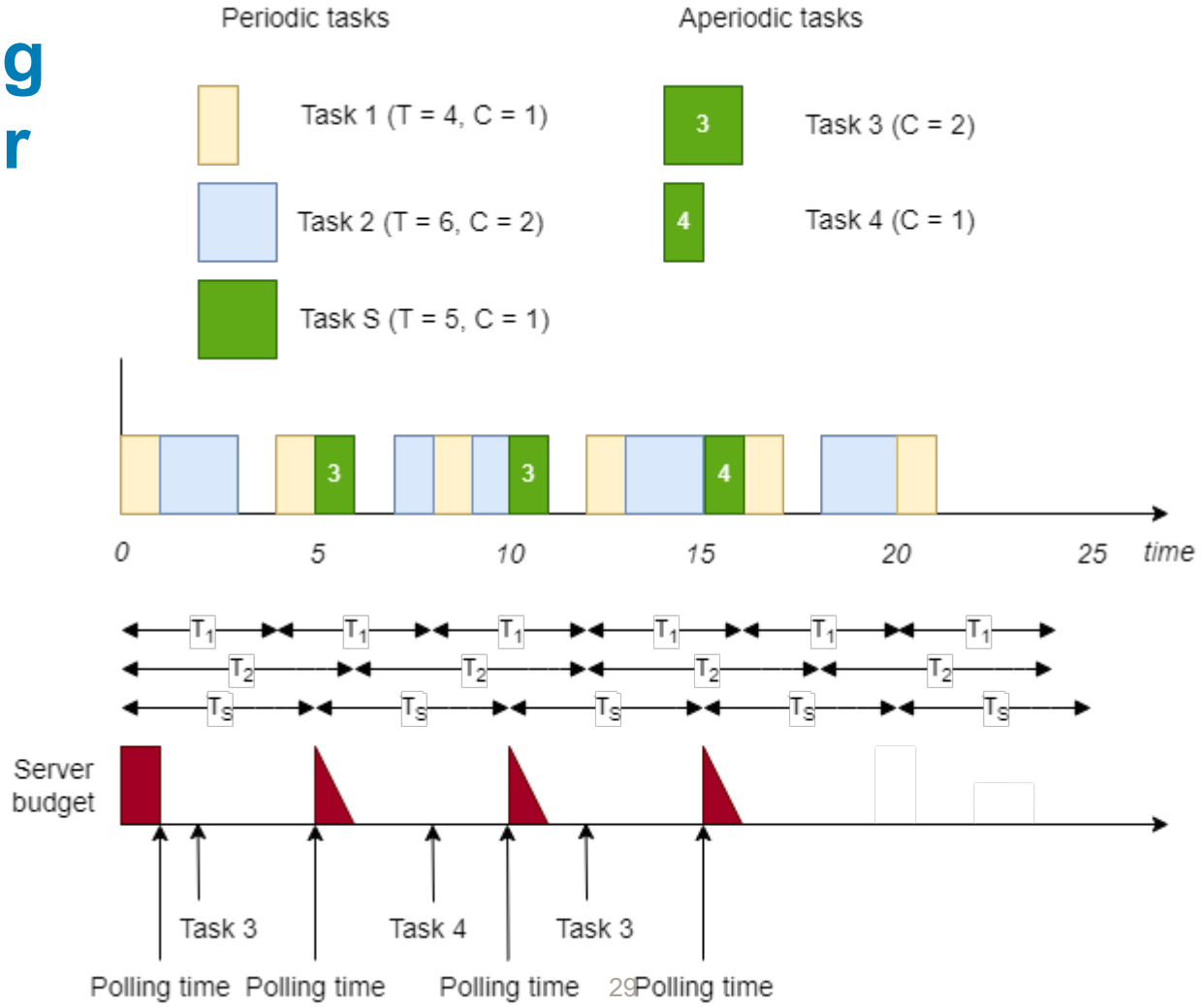
Polling Server



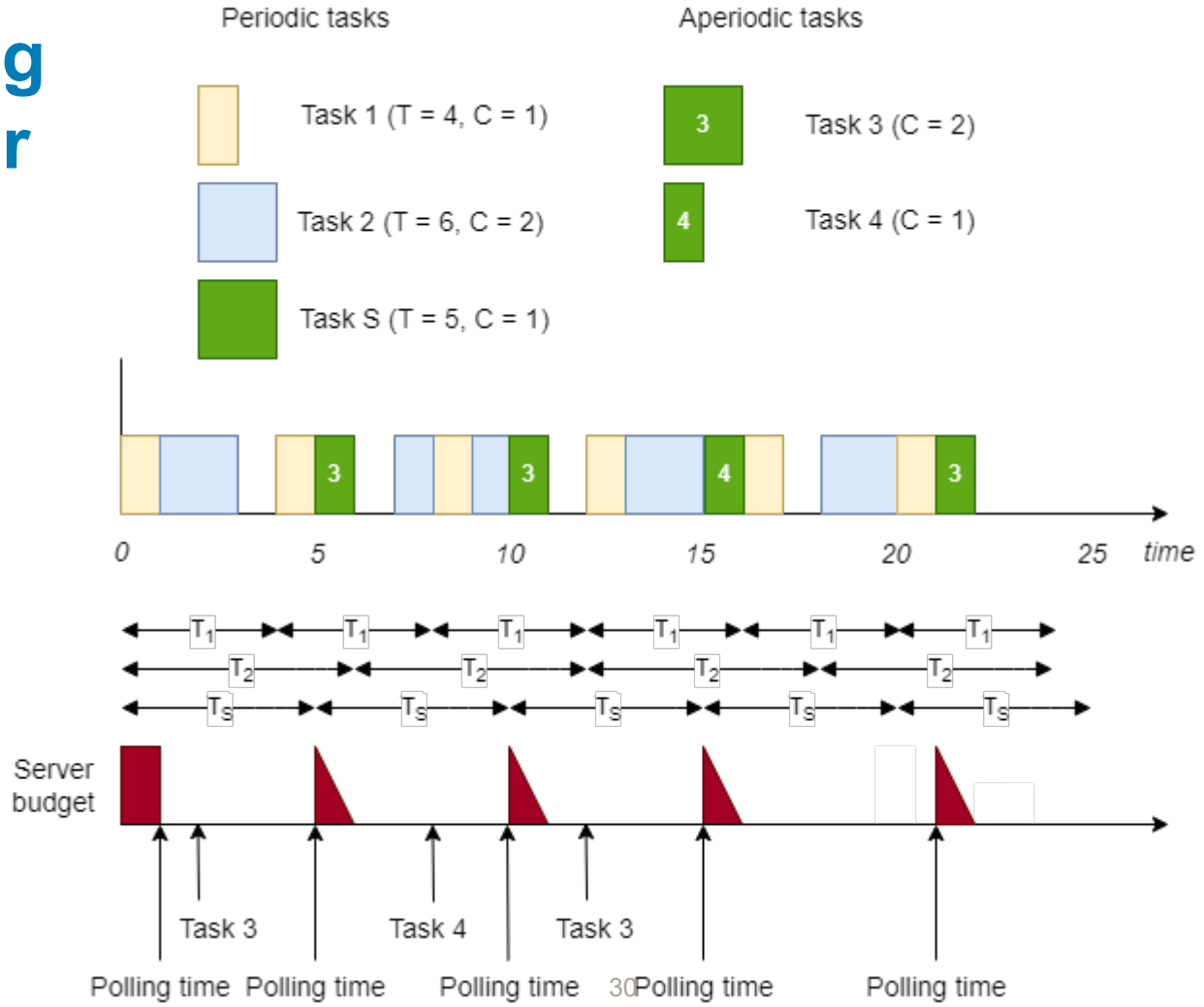
Polling Server



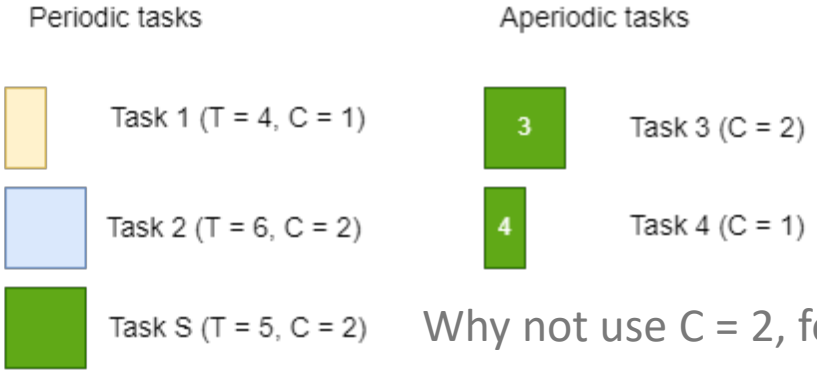
Polling Server



Polling Server

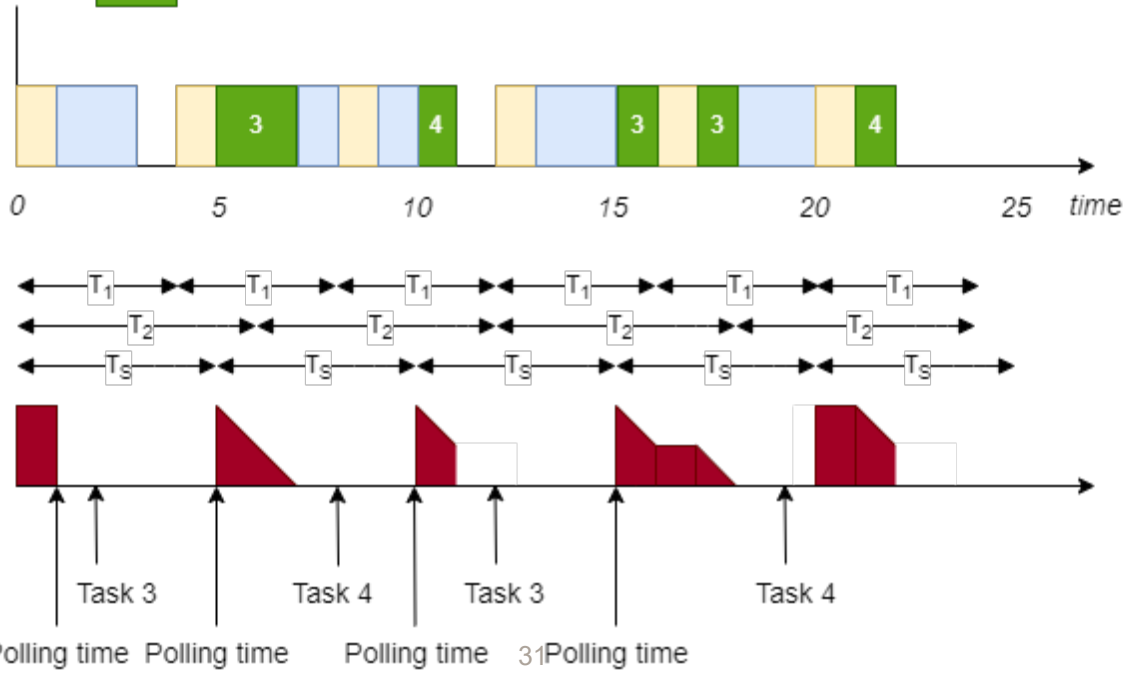


Polling Server

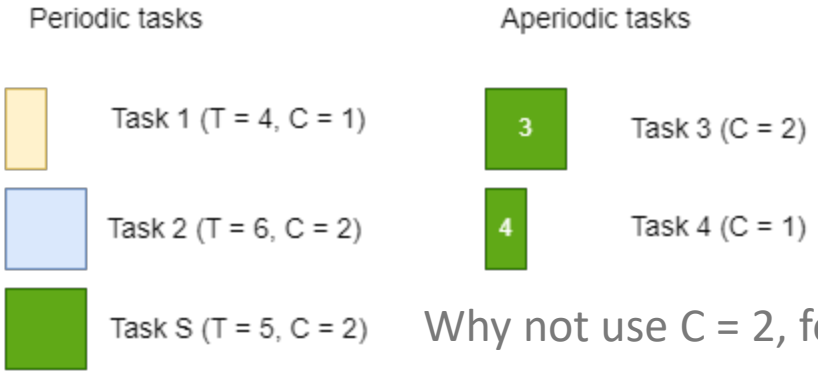


$$U = \frac{59}{60} < 1$$

Why not use C = 2, for a larger server budget?

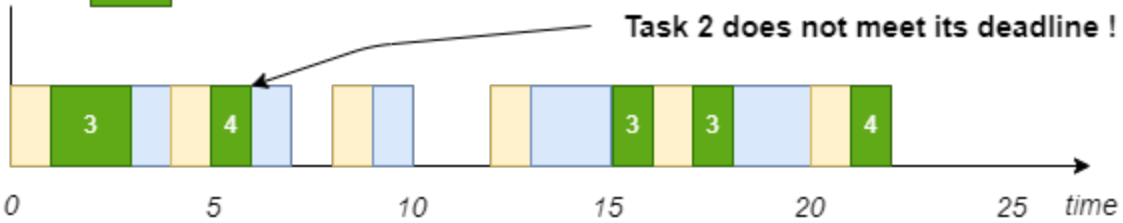


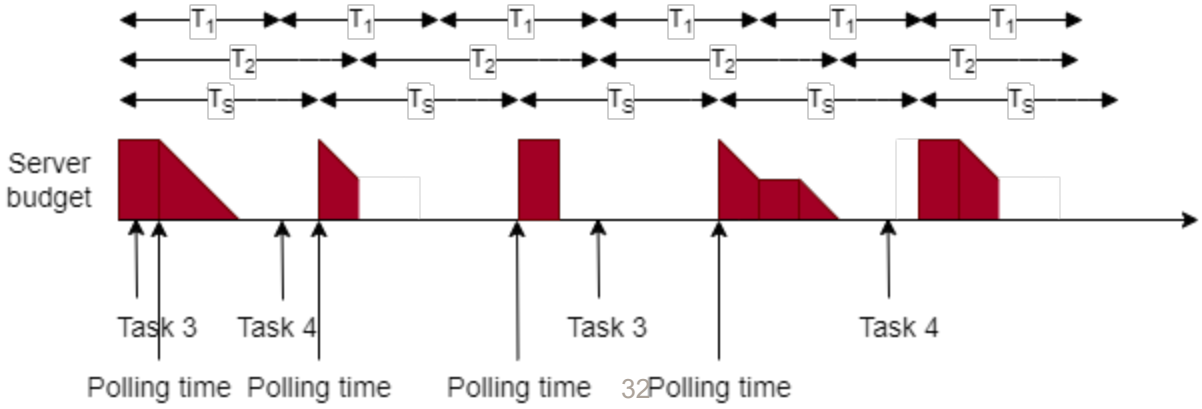
Polling Server



$$\prod_{i=1}^{3}(U_i + 1) = \frac{14}{6} > 2\,!$$

Why not use C = 2, for a larger server budget?



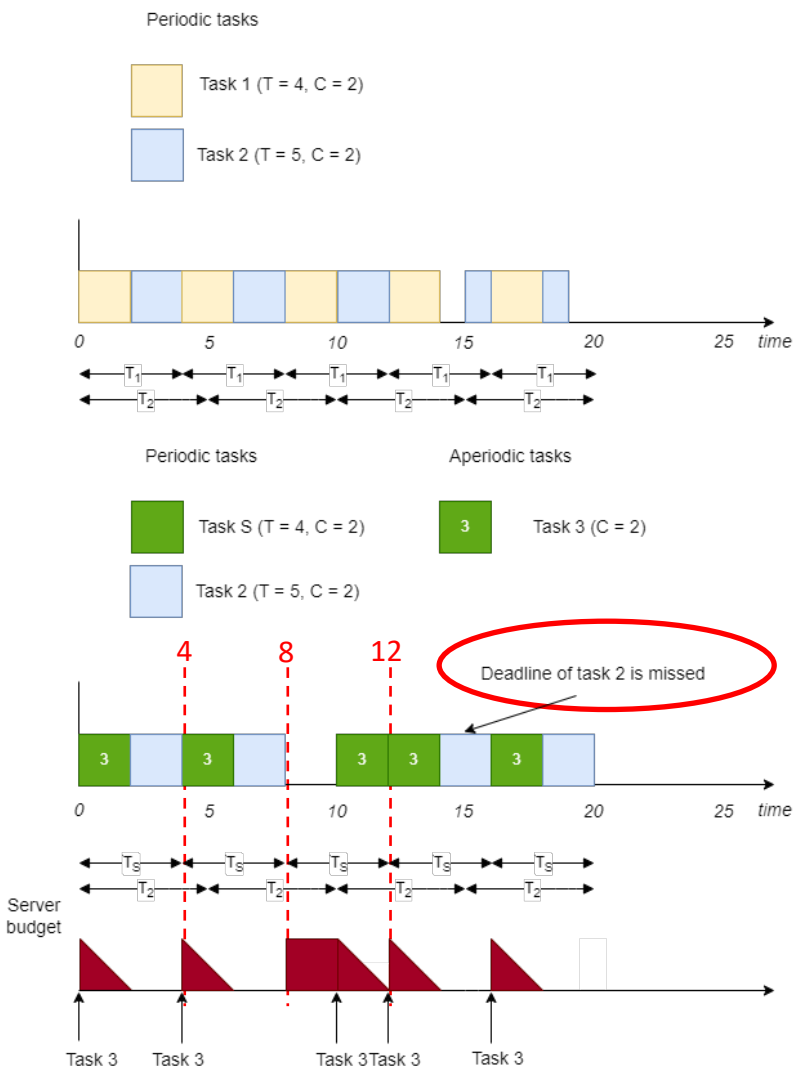


Sporadic Tasks: Deferrable Server

- Similar to Polling Server
- Differentiates itself in the way the server budget is replenished
  - The budget is consumed when the server executes (similar)
  - Once scheduled, the unused budget is retained throughout the entire period
  - So the budget can be used whenever an sporadic task is ready to execute
  - No need to arrive before the polling instant
- This improves the response time of sporadic tasks

Deferrable Server Dimensioning

- With a deferrable server, one of the RM rule breaks:
  - At any time, RM executes the highest-priority task that is ready
- This is not true anymore with a deferrable server, since the execution of a highest-priority task can be deferred
- The schedulability bound is lower and needs to be recomputed



Deferrable Server Dimensioning

- Given a set of n periodic tasks with utilization factor U_p and a deferrable server with utilization factor U_s, then the schedulability of the period task set (n periodic tasks + deferrable server) under RM is guaranteed if

$$U_p \leq n\,(K^{\frac{1}{n}} - 1)$$

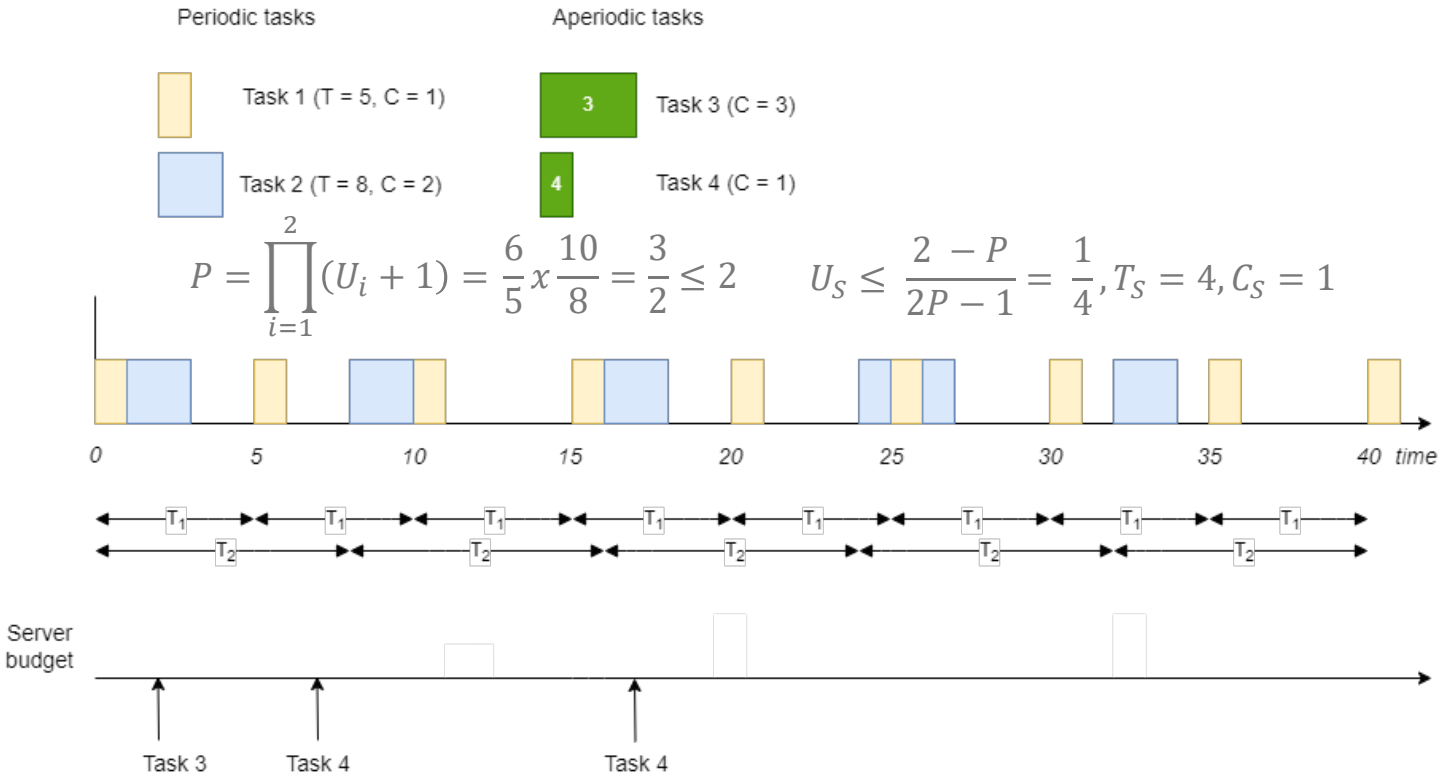
$$\text{where } K = \frac{U_s+2}{2U_s+1}$$

- Using the Hyperbolic bound, the schedulability with a deferrable server is guaranteed if

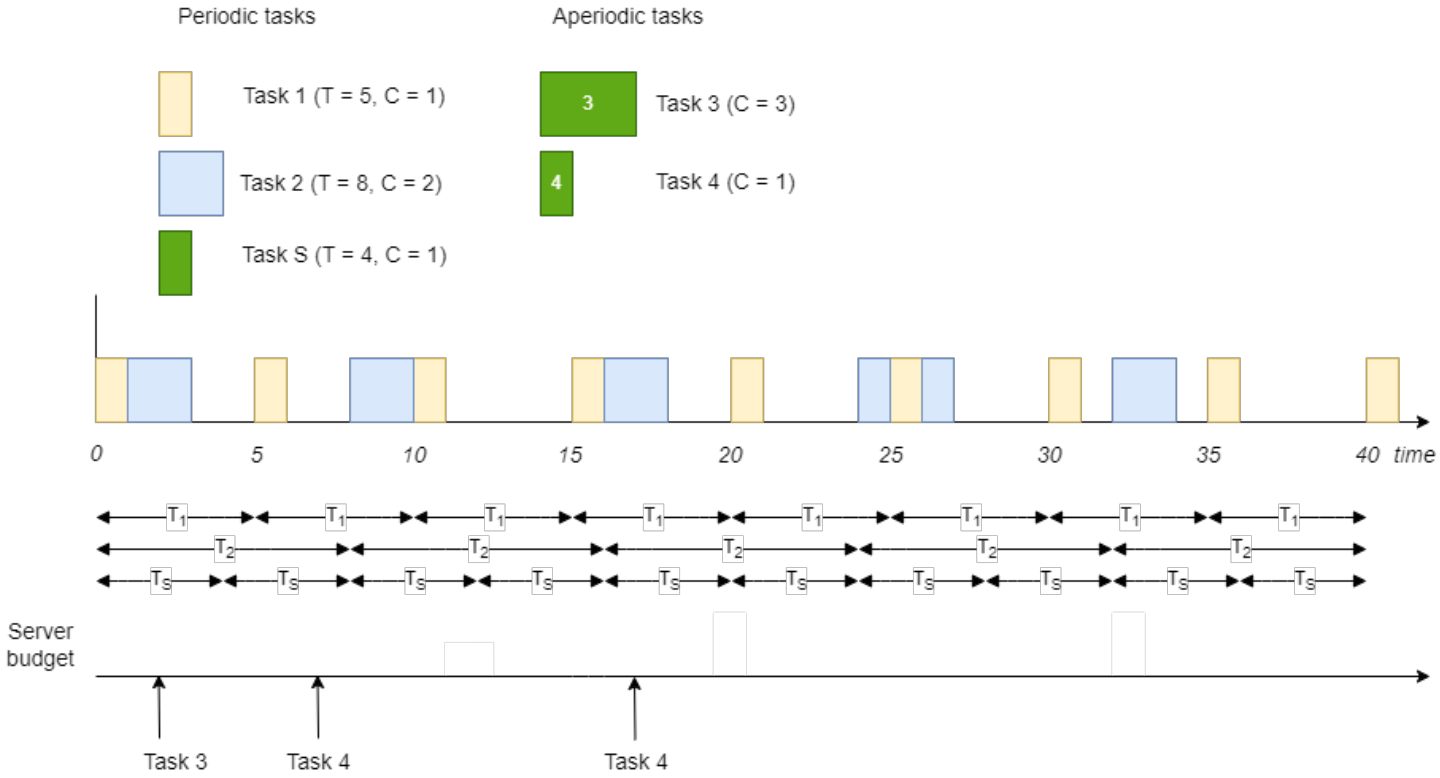
$$given\ \mathrm{P} \stackrel{\text{def}}{=} \prod_{i=1}^{n}(U_i + 1)$$

$$P \leq \frac{U_s+2}{2U_s+1}\ and\ thus\ U_s \leq \frac{2-P}{2P-1}$$

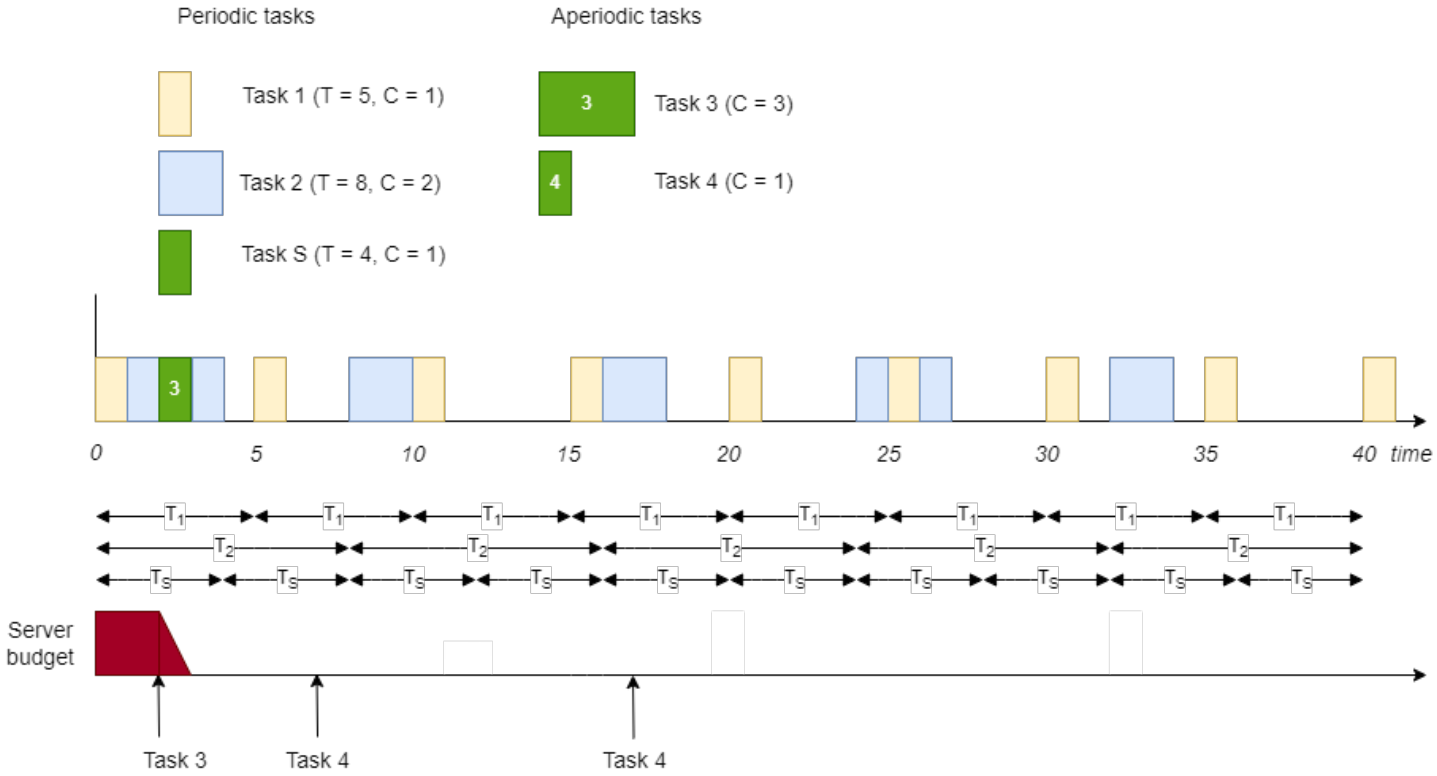
Deferrable Server



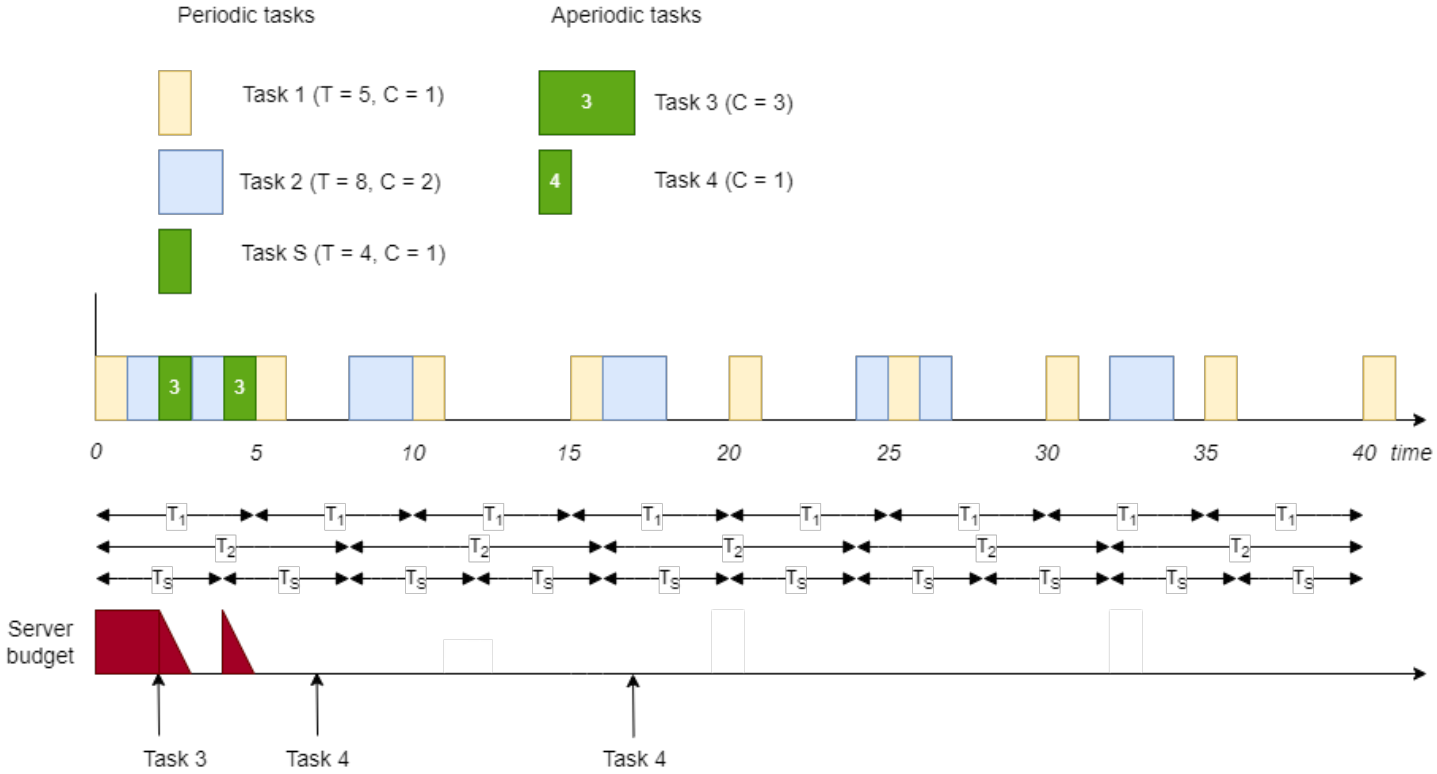
Deferrable Server



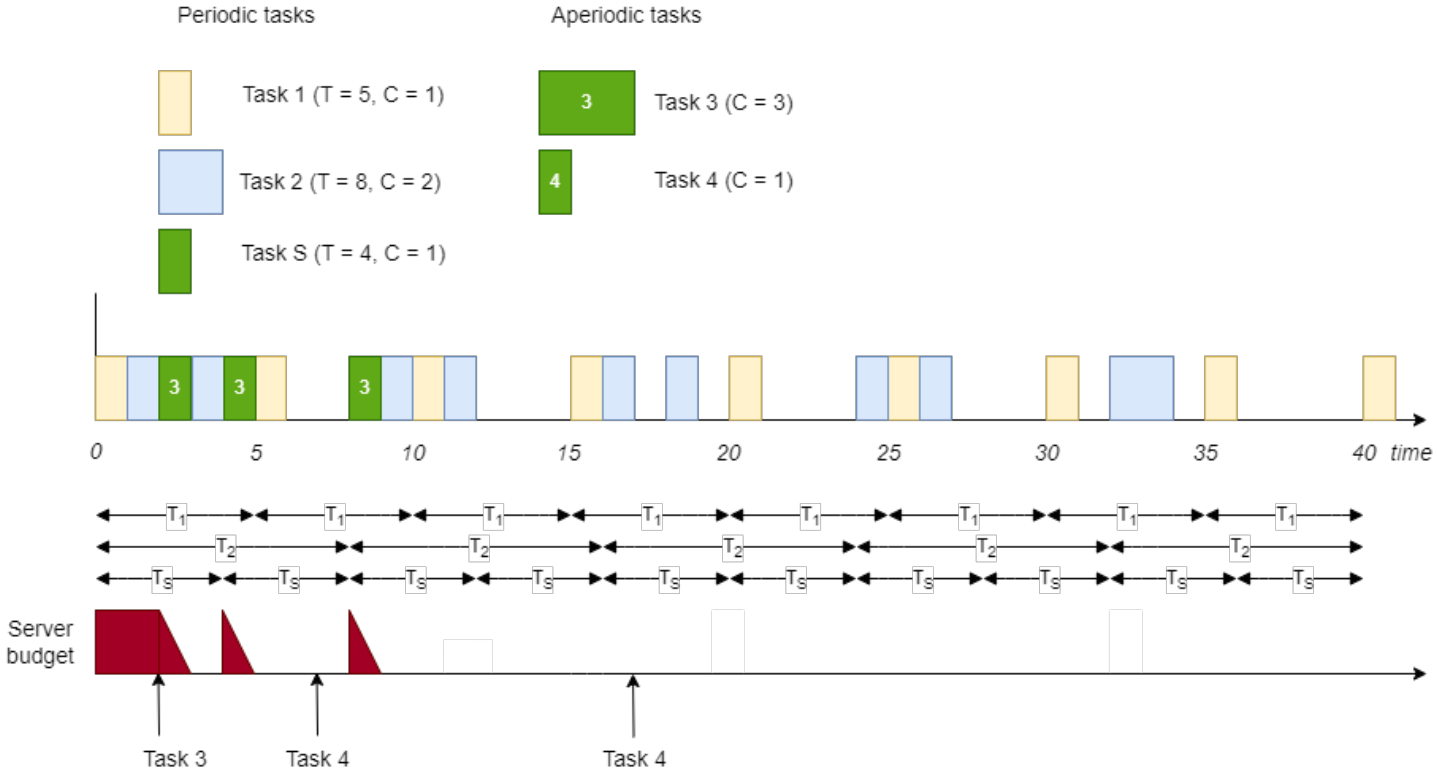
Deferrable Server



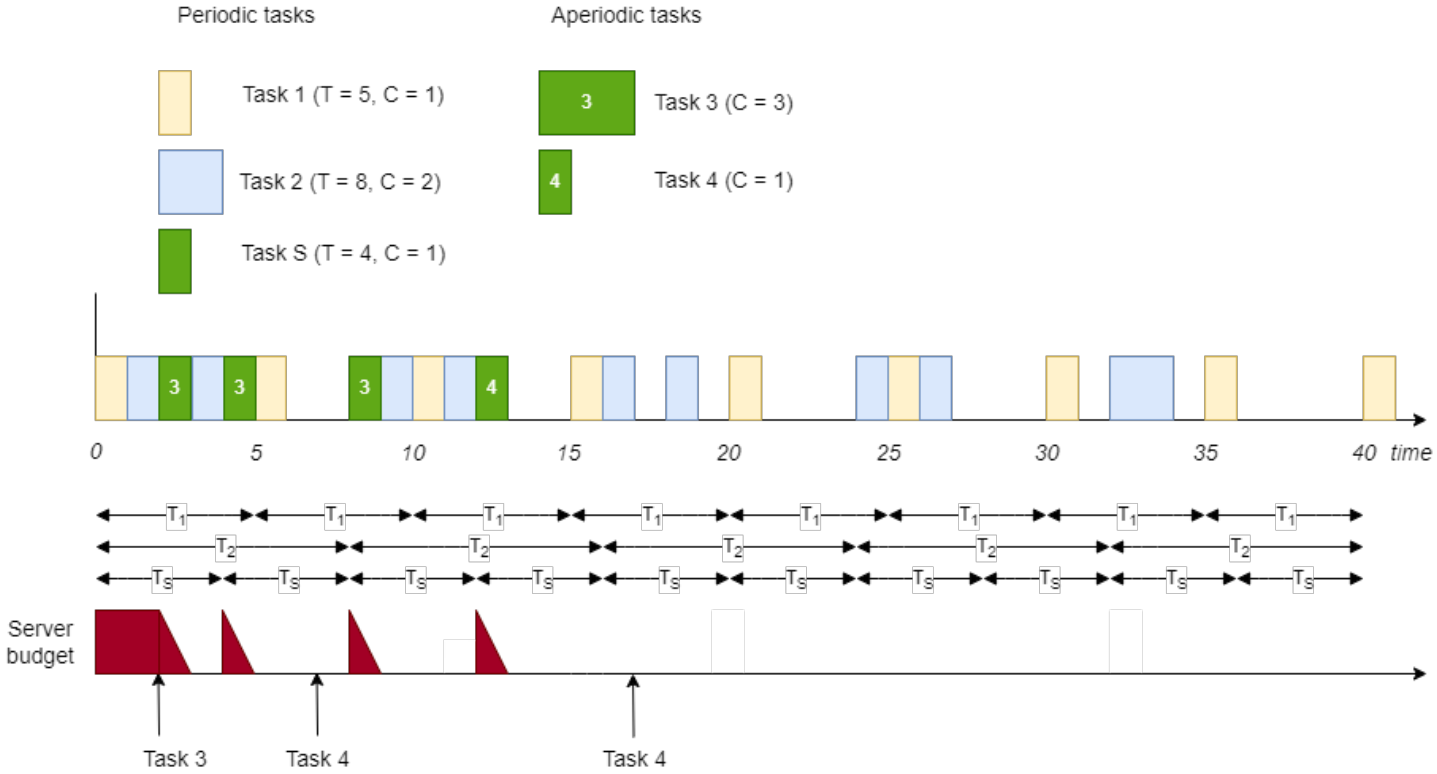
Deferrable Server



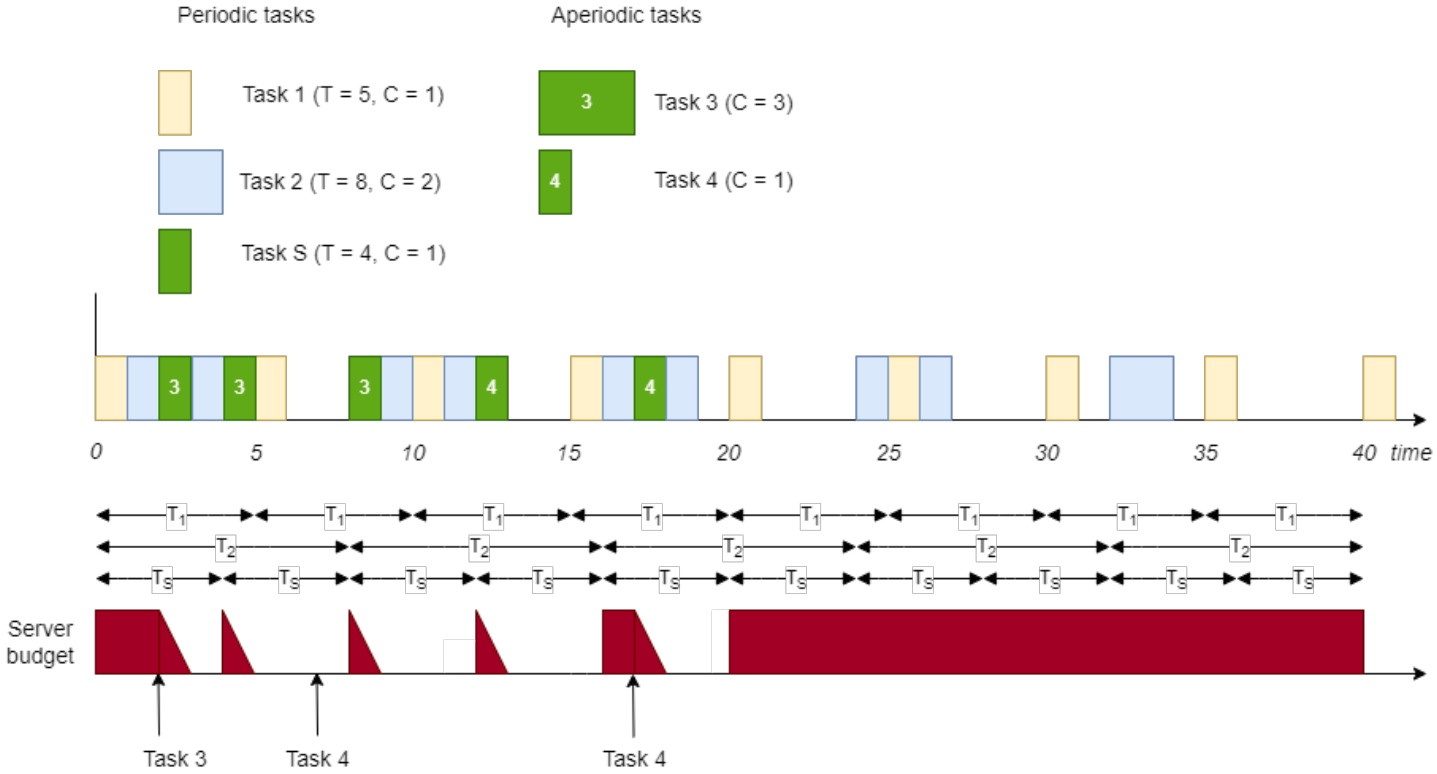
Deferrable Server



Deferrable Server



Deferrable Server



Aperiodic Guarantee using Servers

- Aperiodic Guarantee = Guarantee that an aperiodic task with timing constraint can be performed within the deadline
- The task is accepted only if the guarantee exists = acceptance test
- For both Polling and Deferrable Servers, the acceptance test can be computed on-line
  - Very hard to be computed off-line
- This test is computed by estimating the worst-case response time
- It can be computed only when the server task has the highest priority among all tasks
  - Highest priority means that the server task always executes at the beginning of its period
  - In this case, the finishing time of the aperiodic request can be estimated precisely
- The computation of the online guarantee is left out from this lecture
  - However, the principle of an acceptance test and of its use in a system should be understood

Other fixed-priority servers

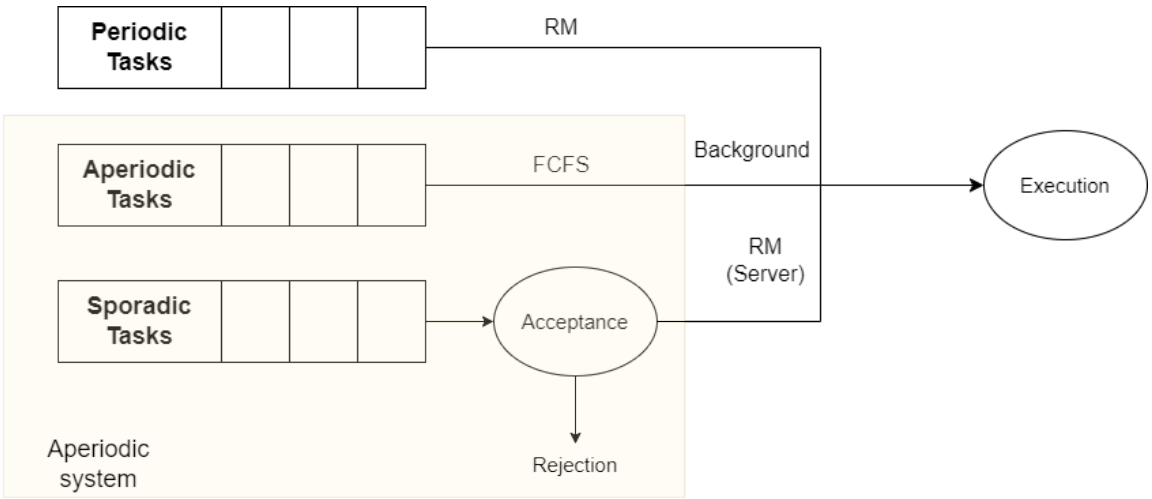
- Mostly differ in the way the server budget is replenished or in the way the server budget is allocated
- Priority Exchange:
  - Server budget is preserved by exchanging it for the execution time of lower-priority tasks
- Sporadic Server:
  - Enhances the average response time of aperiodic tasks
  - Preserves the utilization bound of the periodic task set
- Slack Stealing
  - Provides a better average response time as compared to other fixed-priority servers
  - Steals time from periodic tasks without causing their deadlines to be missed
  - Better because there is no benefit in early completion of periodic tasks (as long as theirs deadlines is respected)

Summary: fixed-priority servers

	Performance	Computational Complexity	Memory Requirement	Implementation Complexity
Background Scheduling	Very Low	Low	Low	Low
Polling Server	Low	Low	Low	Some constraints
Deferrable Server	Ok	Low	Low	Low

Other servers provide better performance at the cost of more complexity and memory requirement

Summary: system model



- Single processor
- All tasks are preemptable
- Periodic tasks scheduled using RM
- Aperiodic tasks are executed using a background task
- Sporadic tasks are executed using a server task
  - Sporadic task can be rejected if they do not pass the acceptance test
- Aperiodic task queues have their own queuing strategy (FCFS, priority-based, etc.)
- A system can implement all or only some of these sub-systems