Embedded Operating Systems

Che-Wei Chang

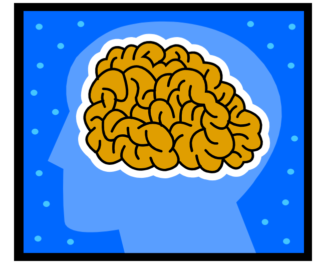
chewei@mail.cgu.edu.tw

Department of Computer Science and Information Engineering, Chang Gung University

Course Roadmap

Basic Concepts

- Embedded System Design Concepts
- Embedded System Developing Tools and Operating Systems
- Embedded Linux and Android Environment

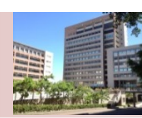


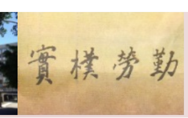
Core Technology

- Real-Time System Design and Scheduling Algorithms
- System Synchronization Protocols

Real Implementation

- System Initialization and Memory Management
- Power Management Techniques and System Routine
- Embedded Linux Labs and Exercises on Android





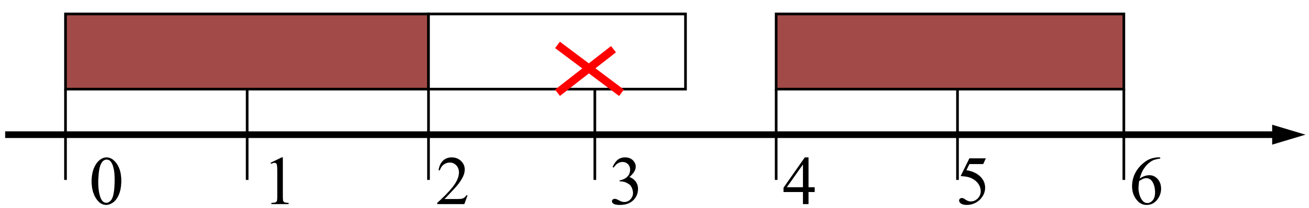
Real-Time Scheduling

Motivation

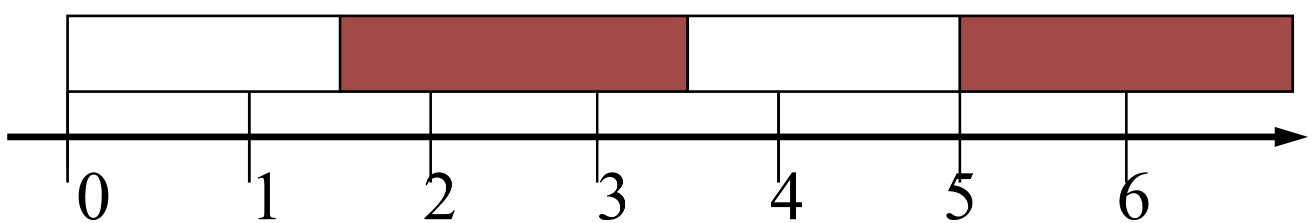
- Studying: 2 days per 4 days

  Playing Basketball: 1.5 days per 3 days

- Case 1: Studying is always more important



- Case 2: Doing whatever is more urgent



Questions

- Can we find an **optimal** scheduler that always produces a feasible schedule whenever it is possible to do so?
  - What does **optimality** means?
- Can we find a quick schedulability test for a set of processes?
  - Is it simple and accurate?
- How do we model scheduling overheads, such as the cost of context switching?

Tentative Assumptions

- Processes are independent
- Processes are all periodic
- The deadline of a request is its next request time
- A scheduler consists of a priority assignment policy and a priority-driven scheduling mechanism

Reference: C.L. Liu and James. W. Layland, “Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment,” JACM, Vol. 20, No.1, January 1973, pp. 46-61

Definitions

- The **response time** of a request for a process is the time span between the request and the end of the response to that request
- A **critical instant** of a process is an instant at which a request of that process has the longest response time
- A **critical interval** for a process is the time interval between the start of a critical instant and the deadline of the corresponding request of the process

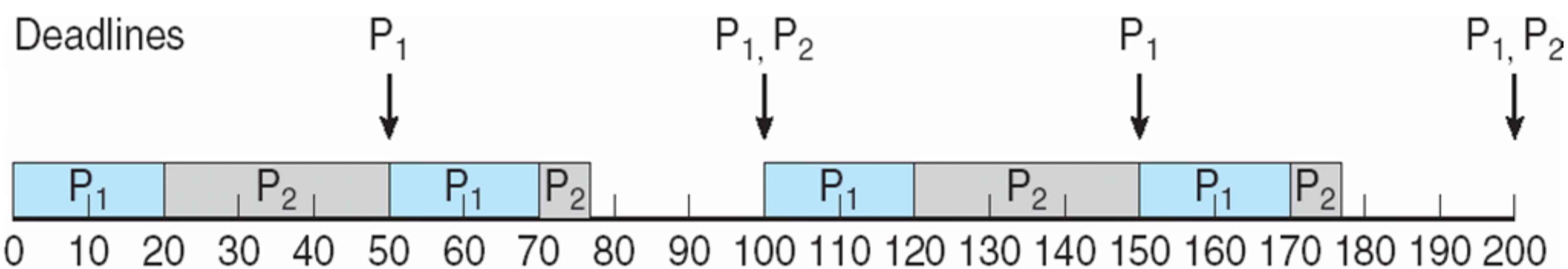
  ➔ A critical instant for any process occurs whenever the process is requested simultaneously with requests for all higher priority processes

An observation: If a process can complete its execution within its critical interval, it is schedulable at all time!

A Static Scheduling Algorithm—Rate Monotonic Scheduling

- A static priority is assigned to each task based on the inverse of its period
  - A task with shorter period ➔ higher priority
  - A task with longer period ➔ lower priority
  - For example:
    - P_1 has its period 50 and execution time 20
    - P_2 has its period 100 and execution time 37

    ➔ P_1 is assigned a higher priority than P_2



Property of Rate Monotonic Scheduling

- The **rate monotonic** (RM) priority assignment assigns processes priorities according to their request rates
  - If a feasible fixed priority assignment exists for some process set, then the rate monotonic priority assignment is feasible for that process set

→ The optimal fixed priority assignment

Proof. Exchange the priorities of two tasks if their priorities are out of RMS order.

Task A	Task B	Task C
Period: 10 Priority: 1	Period: 23 Priority: 2	Period: 15 Priority: 3

If it is feasible

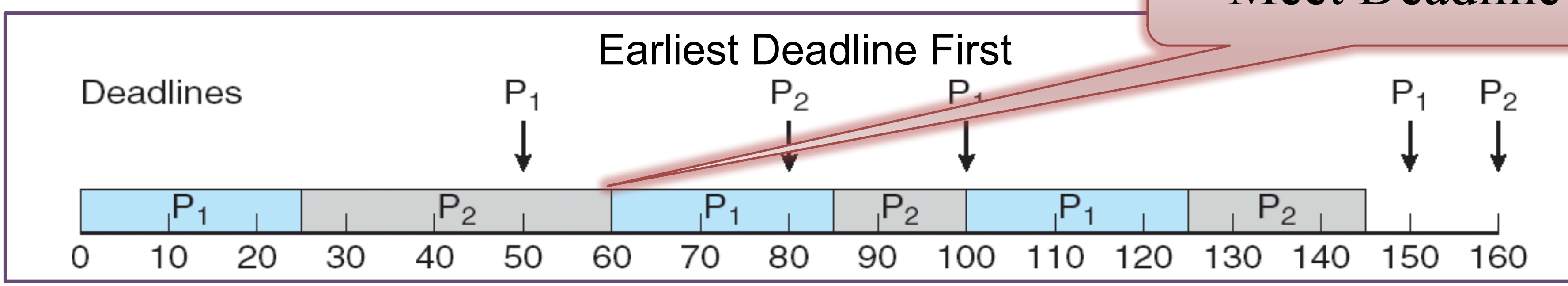
Task A	Task C	Task B
Period: 10 Priority: 1	Period: 15 Priority: 2	Period: 23 Priority: 3

RM must be feasible

A Dynamic Scheduling Algorithm—Earliest Deadline First Scheduling

- Dynamic priorities are assigned according to deadlines
  - The earlier the deadline, the higher the priority
  - The later the deadline, the lower the priority
  - For example:
    - P_1 has its period 50 and execution time 25
    - P_2 has its period 80 and execution time 35

Meet Deadline



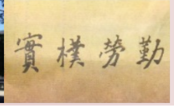
Real-Time Analysis

- For a task τ_i with the period P_i and the execution time C_i, the utilization U_i of τ_i is defined as $U_i = \frac{C_i}{P_i}$
- For a real-time task set **T** the total utilization of the task set is $\sum_{\tau_i \in \mathbf{T}} U_i$
- If $\sum_{\tau_i \in \mathbf{T}} U_i \leq 69\%$, Rate Monotonic Scheduling can schedule all tasks in **T** to meet all deadlines
  - More precisely, for n tasks, the i-th task can meet deadline if

$$\sum_{j=1}^{i} U_i \leq i\left(2^{1/i} - 1\right)$$

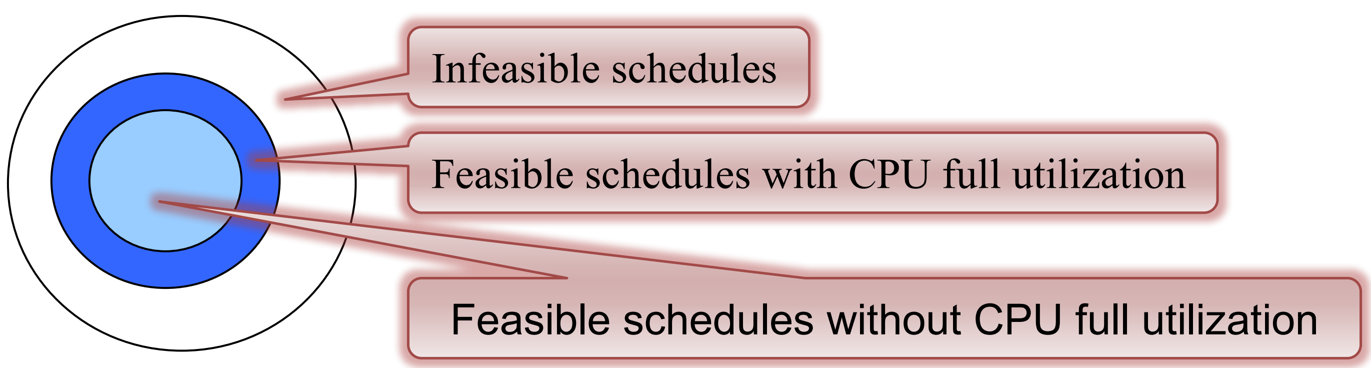
- If and only if $\sum_{\tau_i \in \mathbf{T}} U_i \leq 100\%$, Earliest Deadline First Scheduling can schedule all tasks in **T** to meet all deadlines

Reference: C.L. Liu and James. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," JACM, Vol. 20, No.1, January 1973, pp. 46-61



CPU Utilization

- For a given priority assignment, a process set **fully utilizes** the processor if the priority assignment is feasible for the set and if any increase in the run time of any processes in the set will make the priority assignment infeasible
  - EDF: 100% ➔ fully utilize, <100% ➔ not fully utilize
  - RM:



RM and EDF (1/2)

- The achievable utilization factor of the EDF algorithm is 100%.The EDF algorithm is an optimal dynamic priority scheduling policy in the sense that a process set is schedulable if its CPU utilization is no larger than 100%.
- The achievable utilization factor of the RMS algorithm is about *ln2* (~69%). The RMS algorithm is an optimal fixed priority scheduling policy in the sense that if a process set is schedulable by some fixed priority scheduling algorithm, then it is schedulable by the RMS algorithm.

RM and EDF (2/2)

- For a set of m processes with the RM fixed priority order, the i-th process is schedulable if

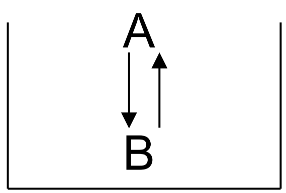
$$\sum_{j=1}^{i} \frac{c_j}{p_j} \leq i\left(2^{1/i} - 1\right)$$

- For a set of m processes with the EDF scheduling, all process will miss deadlines when the total utilization is more than 100%

Scheduling Overheads

- **Context Switching**
  - Needed either when a process is preempted by another process, or when a process completes its execution
  - Stack Discipline

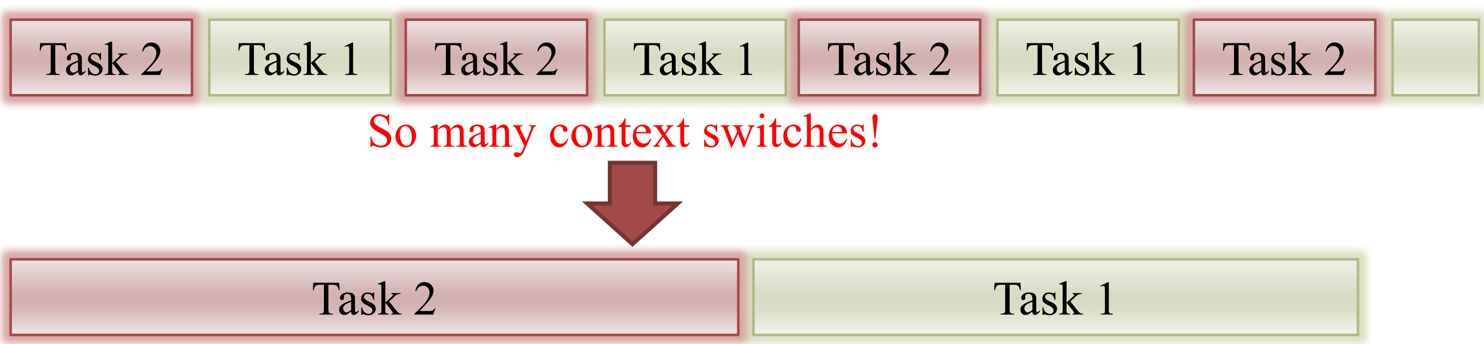
    If process A preempts process B, process A must complete before process B can resume



If it is obeyed, charge the cost of preemption (context switching cost) once to the preempting process!

Least Slack Time Algorithm

- The least slack time algorithm (LST), which assigns processes priorities inversely proportional to their slack times is also optimal if context switching cost can be ignored
  - The slack time of a process is d(t) - t - c(t)
    - t: current time
    - d(t): deadline
    - c(t): remaining execution time
  - An example
    - The time t = 0
    - Two task have the same deadline 20
    - Task 1 has c(t) = 7, and task 2 has c(t) = 8



Process Synchronization

Basic Concept

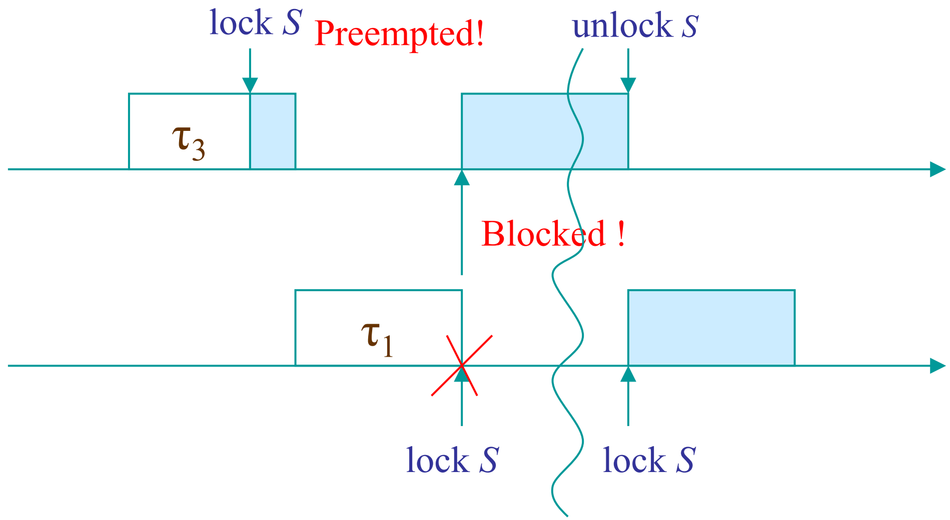
- Processes might share non-preemptible resources or have precedence constraints
- Papers for discussion:
  - L. Sha, R. Rajkumar, J.P. Lehoczky, “Priority Inheritance Protocols: An Approach to Real-Time Synchronization,” IEEE Transactions on Computers, 1990.
  - A.K. Mok, “The Design of Real-Time Programming Systems Based on Process Models,” IEEE Real-Time Systems Symposium, Dec 1994.

Process Synchronization

- Motivation
  - Can we find an efficient way to analyze the schedulability of a process set (systematically)
  - What kinds of restrictions on the use of communication primitives are needed so as to efficiently solve the restricted scheduling problem
  - How can we control the priority inversion problem
  - The lengths of critical sections might be quite different

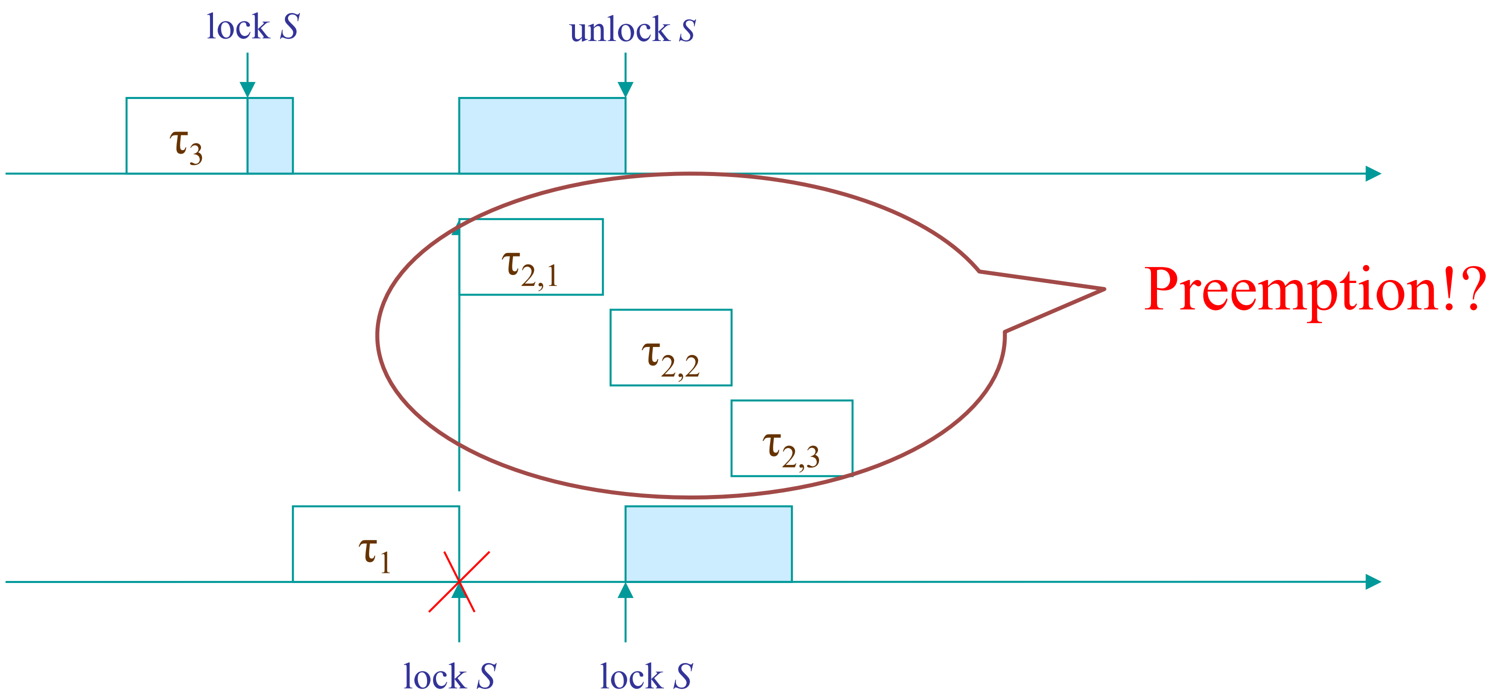
Blocking and Preemption

- Blocking: a higher-priority process is forced to wait for the execution of a lower-priority process
- Preemption: a low-priority process is forced to wait for the execution of a high-priority process



Priority Inversion

- When there are a lot of tasks having priority between that of τ_1 and τ_3, there are a lot of priority inversions

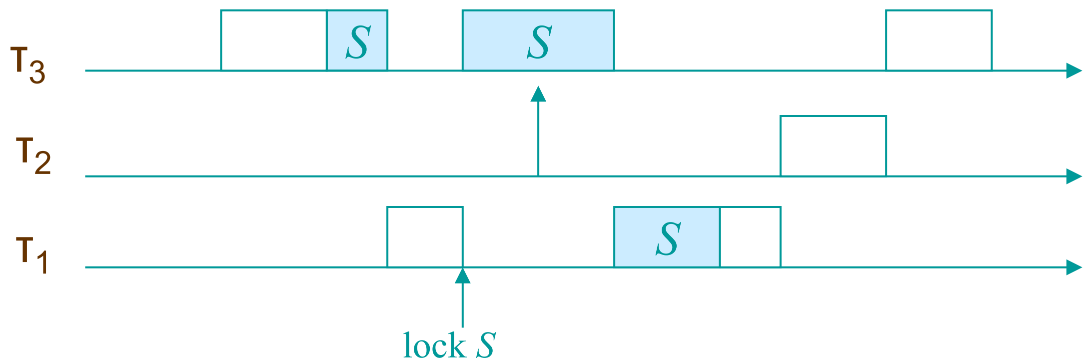


Priority Inheritance Protocol (PIP)

- Priority-Driven Scheduling
  - The process which has the highest priority among the ready processes is assigned the processor
- Synchronization
  - Process τ_i must obtain the lock on the semaphore guarding a critical section before τ_i enters the critical section
  - If τ_i obtains the required lock, τ_i enters the corresponding critical section; otherwise, τ_i is blocked and said to be blocked by the process holds the lock on the corresponding semaphore
  - Once τ_i exits a critical section, τ_i unlocks the corresponding semaphore and makes its blocked processes ready
- Priority Inheritance
  - If a process τ_i blocks higher priority processes, τ_i inherits the highest priority of the process blocked by τ_i
  - Priority inheritance is transitive

Properties of PIP

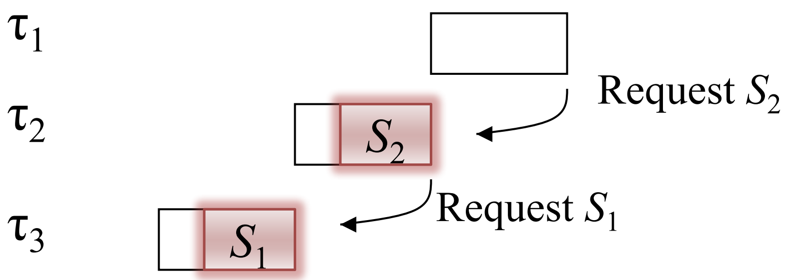
- No priority inversion



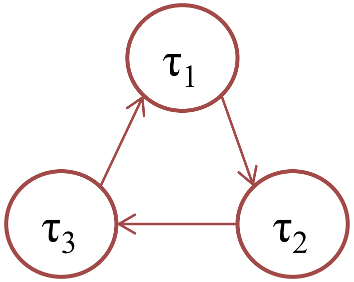
- A semaphore S can be used to cause inheritance blocking to task J only if S is accessed by a task which has a priority lower than that of J and might be accessed by a task which has a priority equal to or higher than that of J.

Concerns of PIP

- A chain of blocking is possible



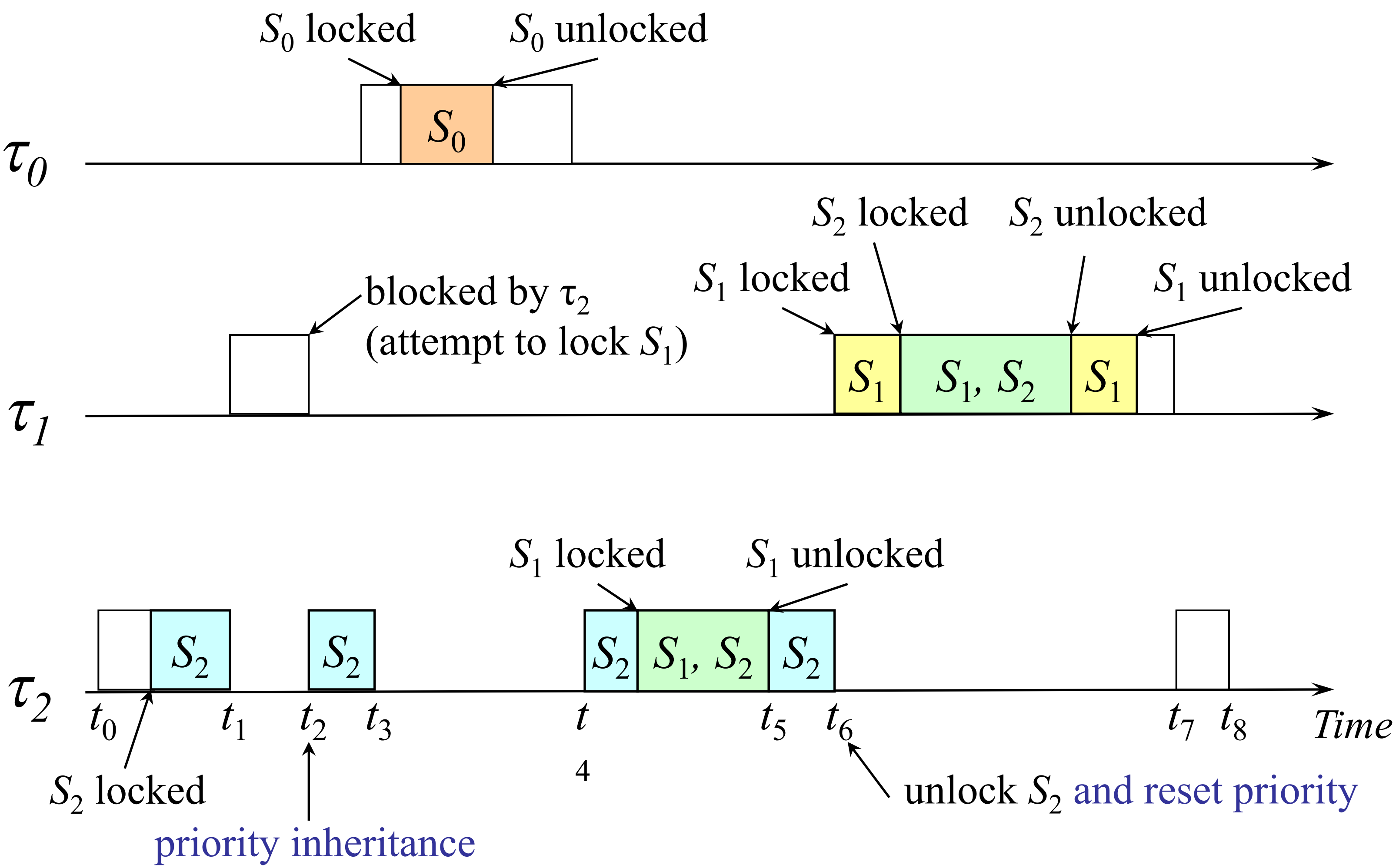
- A deadlock can be formed



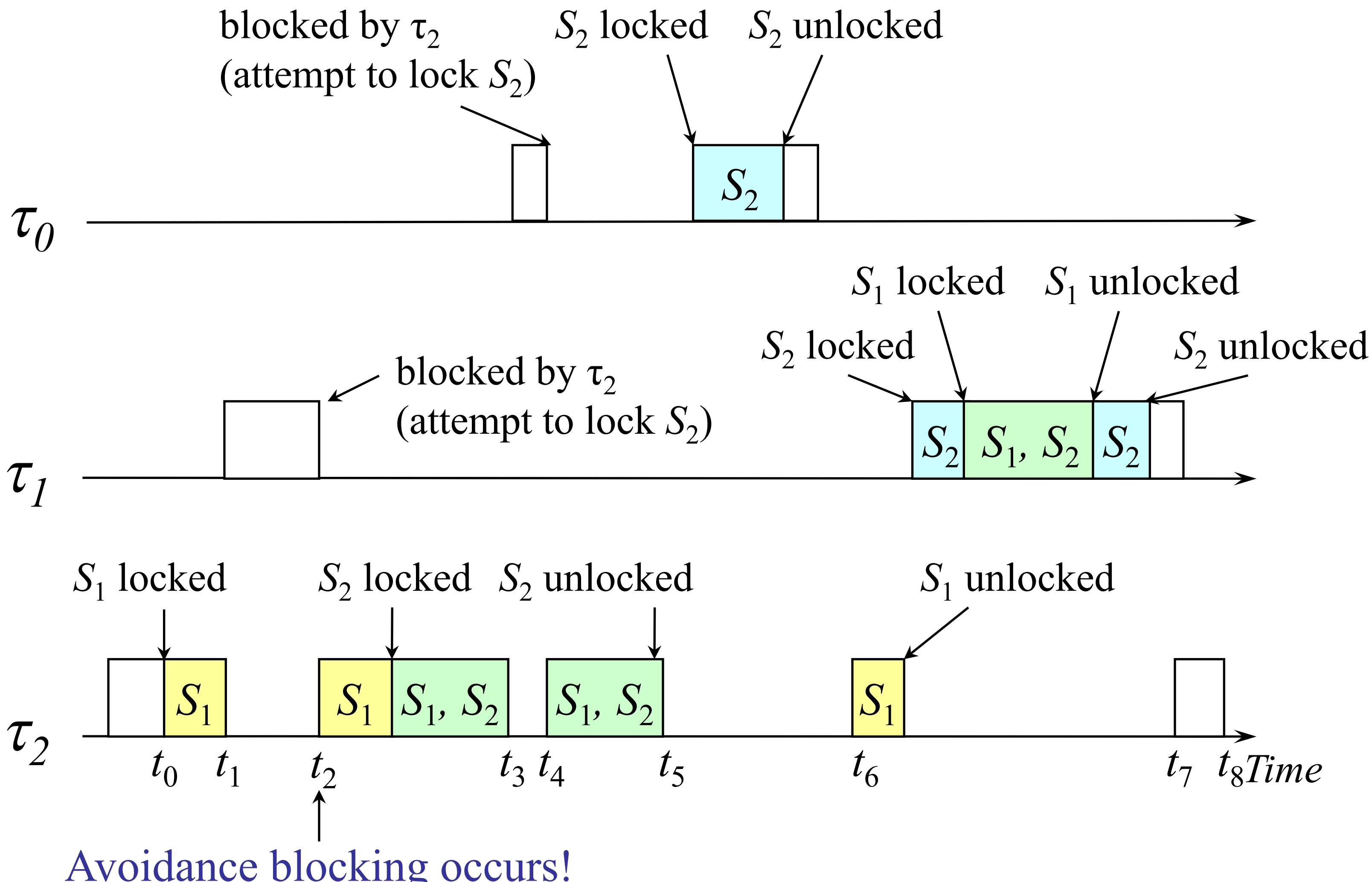
Priority Ceiling Protocol (PCP)

- The priority ceiling of a semaphore is the priority of the highest priority task that may lock the semaphore
- The Basic Priority Inheritance Protocol + Priority Ceiling
- A task J may successfully lock a semaphore S if S is available, and the priority of J is higher than the highest priority ceiling of all semaphores currently locked by tasks other than J
- Priority inheritance is transitive

Example: Deadlock Avoidance



Example: Chain Blocking Avoidance



Properties of PCP

- The priority ceiling protocol prevents transitive blockings
- The priority ceiling protocol prevents deadlock
- No job can be blocked for more than one critical section of any lower priority job
- A set of n periodic tasks under the **priority ceiling protocol** can be scheduled by the **rate monotonic algorithm** if the following conditions are satisfied:

$$\forall i, \quad 1 \le i \le n, \quad \sum_{j=1}^{i-1} \frac{c_j}{p_j} + \frac{c_i + B_i}{p_i} \le i\left(2^{1/i} - 1\right),$$

where B_i is the worst-case blocking time for τ_i, and each task will be blocked on once in a period

Rate Monotonic Analysis

Periodic Requirements (1/2)

Task τ_1: C_1=20, P_1=100, U_1=0.2
Task τ_2: C_2=40, P_2=150, U_2=0.267
Task τ_3: C_3=100, P_3=350, U_3=0.286

- Total utilization: $75.3\% \leq 3\left(2^{\frac{1}{3}} - 1\right) = 77.9\%$
- 24.7% of the CPU is usable for lower-priority background computation

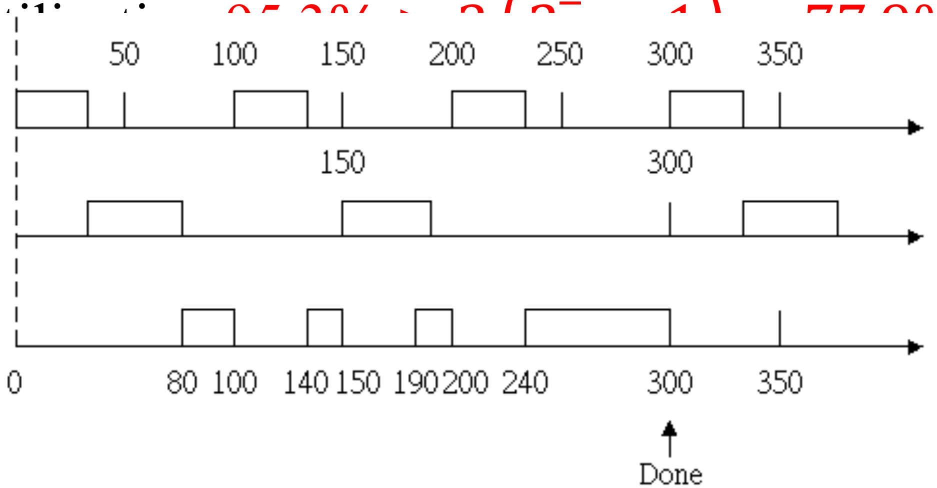
Periodic Requirements (2/2)

Task τ_1: C_1=40, P_1=100, U_1=0.4

Task τ_2: C_2=40, P_2=150, U_2=0.267

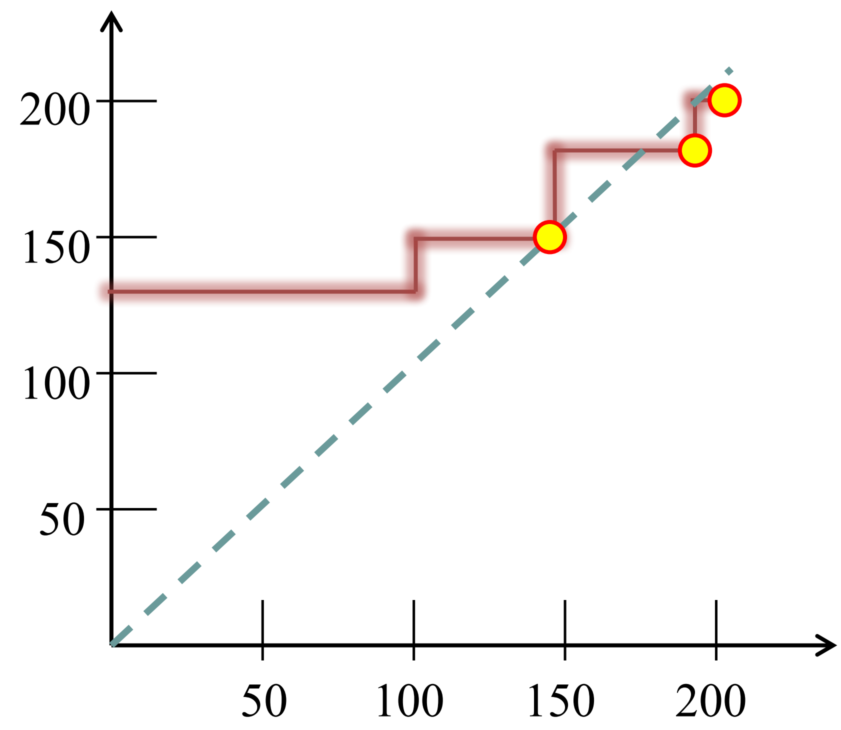
Task τ_3: C_3=100, P_3=350, U_3=0.286

- The utilization of the first two tasks: $66.7\% \le 2\left(2^{\frac{1}{2}} - 1\right) = 82.8\%$
- The total



Rate Monotonic Analysis (RMA)

- A RMA Example:
  - $\tau_1(20,100)$, $\tau_2(30,150)$, $\tau_3(80, 210)$, $\tau_4(100,400)$
  - τ_1
    - $c_1 <= 100$
  - τ_2
    - $c_1 + c_2 <= 100$ or
    - $2c_1 + c_2 <= 150$
  - τ_3
    - $c_1 + c_2 + c_3 <= 100$ or
    - $2c_1 + c_2 + c_3 <= 150$ or
    - $2c_1 + 2c_2 + c_3 <= 200$ or
    - $3c_1 + 2c_2 + c_3 <= 210$
  - τ_4
    - $c_1 + c_2 + c_3 + c_4 <= 100$ or
    - $2c_1 + c_2 + c_3 + c_4 <= 150$ or
    - …



RMA with Blocking Consideration (1/2)

- A RMA Example with blocking time:
  - $\tau_1(20,100)$, $\tau_2(30,150)$, $\tau_3(80, 210)$, $\tau_4(100,400)$
  - τ_1: $(S_1, 5)$
  - τ_2: $(S_2, 15)$
  - τ_3: $(S_1, 10)$, $(S_3, 5)$
  - τ_4: $(S_2, 5)$, $(S_3, 20)$
- What is the priority ceiling of each semaphore?
  - S_1: τ_1, S_2: τ_2, S_3: τ_3
- When PCP is adopted (block once), what is the blocking time of each task?
  - τ_1: 10, τ_2: 10, τ_3: 20, τ_4: 0

RMA with Blocking Consideration (2/2)

- A RMA Example with blocking time:
  - For each task, we have to consider the execution time, period, and blocking time
  - $\tau_1(20,100,10)$, $\tau_2(30,150,10)$, $\tau_3(80, 210,20)$, $\tau_4(100,400,0)$
  - τ_1
    - $b_1 + c_1 <= 100$
  - τ_2
    - $b_2 + c_1 + c_2 <= 100$ or
    - $b_2 + 2c_1 + c_2 <= 150$
  - τ_3
    - $b_3 + c_1 + c_2 + c_3 <= 100$ or
    - $b_3 + 2c_1 + c_2 + c_3 <= 150$ or
    - $b_3 + 2c_1 + 2c_2 + c_3 <= 200$ or
    - $b_3 + 3c_1 + 2c_2 + c_3 <= 210$
  - τ_4
    - …

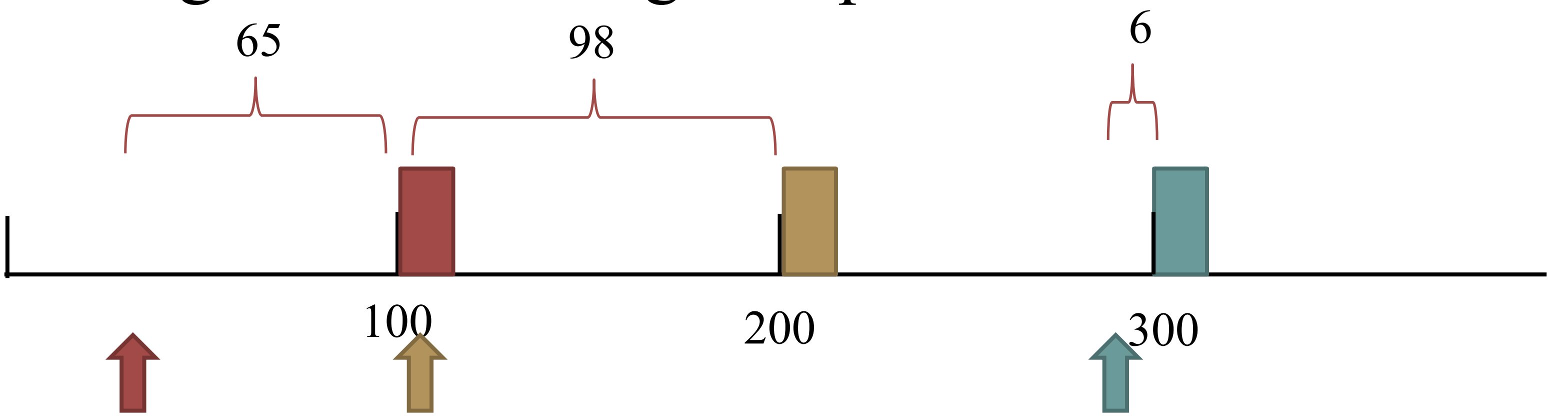
Aperiodic Servers

Observation of Aperiodic Tasks

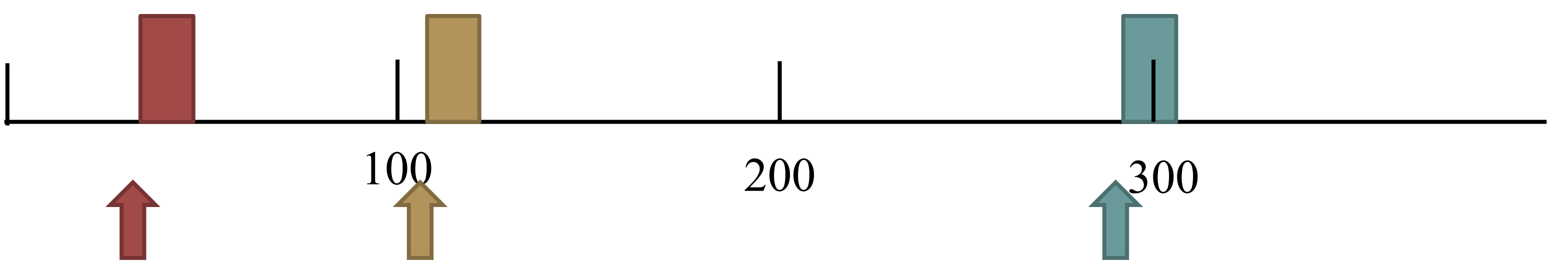
- Aperiodic tasks run at irregular intervals
- Aperiodic deadlines
  - Hard deadline: minimum inter-arrival time
  - Soft deadline: best average response time
- Services such as
  - User requests
  - Device interrupts
  - …

Scheduling Aperiodic Tasks

- Polling Server~ Average Response Time = 50 units

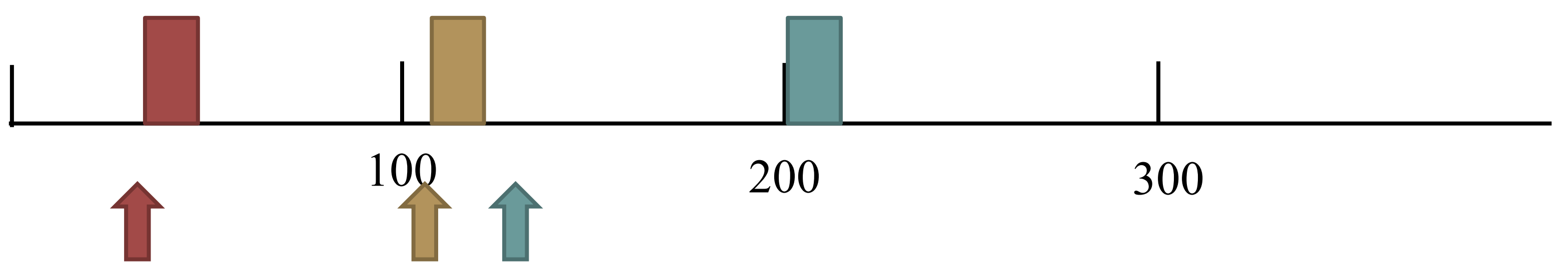


- Interrupt Server ~ Average Response Time = 1 unit



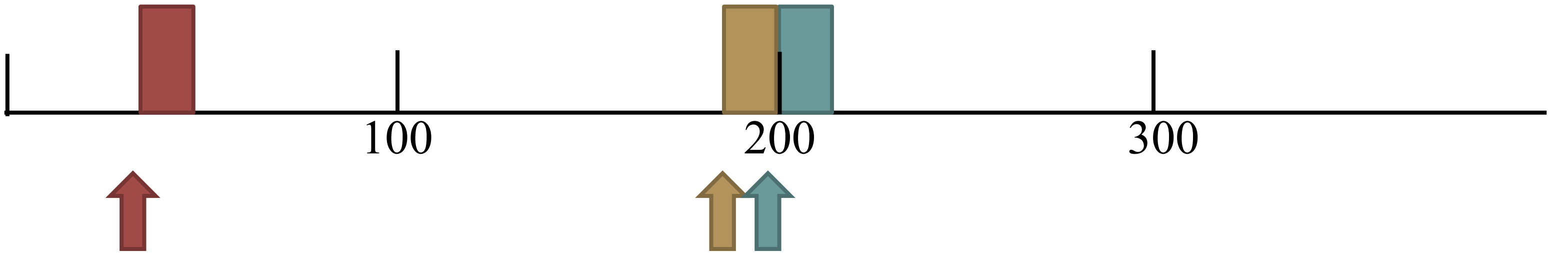
Deferrable Server

- Polling Server: the average response time is long
- Interrupt Server: the computing time of aperiodic tasks is difficult to limited
- Deferrable Server
  - In each period, a deferrable server has a execution budget
  - When execution budget is used up, server execution drops to a lower (background) priority

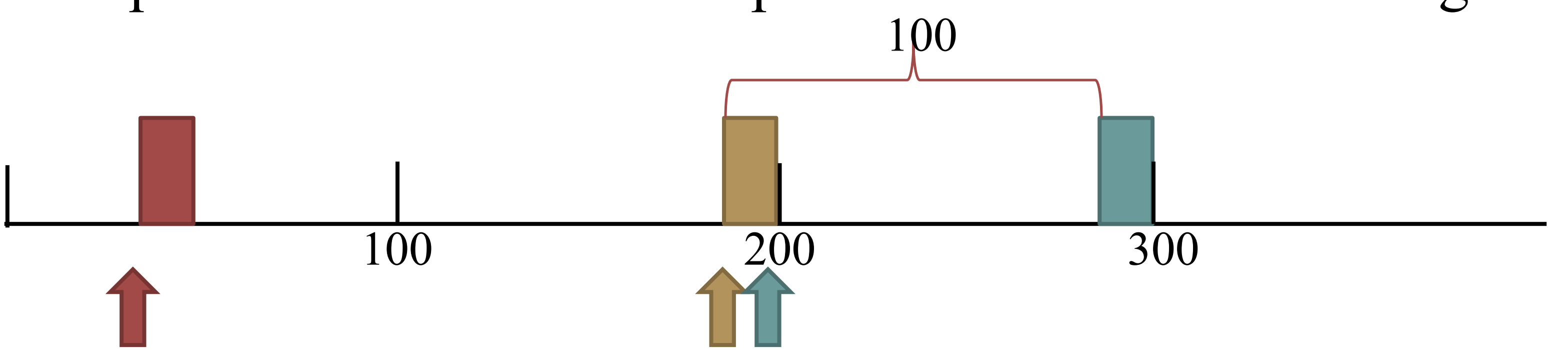


Sporadic Server

- Deferrable Server might consume two times of the execution budget in short time



- Sporadic Server
  - Replenishment occurs one "period" after the start of usage

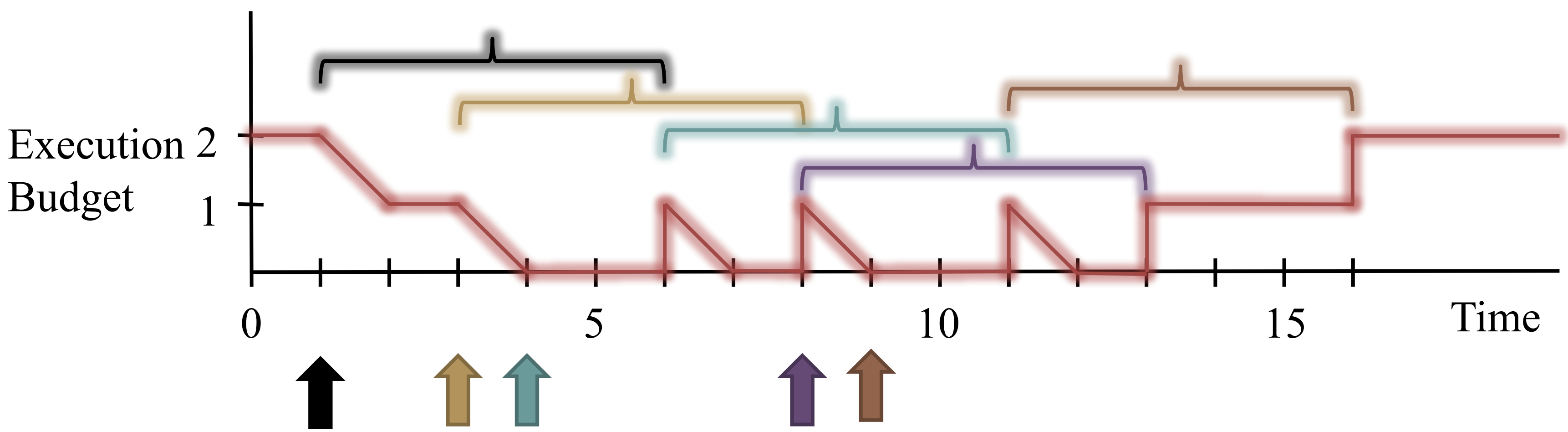


Properties of Sporadic Server

- A sporadic server differs from a deferrable server in its replenishment policy:
  - A 100 ms deferrable server replenishes its execution budget every 100 ms, no matter when the execution budget is used
  - The affect of a sporadic server on lower priority tasks is no worse than a periodic task with the same period and execution time

An Example of Sporadic Server

- A sporadic server has a replenishment period 5 and an execution budget 2
- Each event consumes the execution 1
- Events arrive at 1, 3, 4, 8, 9



Properties of Sporadic Server

- For a sporadic server has a replenishment period X and an execution budget Y
  - Given a set of sporadic tasks, If
    - Each of the aperiodic tasks has its minimum inter-arrival time no less than X
    - The total execution of the task set is no more than Y
  - All sporadic tasks can meet the deadline constraints
- When a system consists of periodic tasks and sporadic servers
  - A sporadic server with replenishment period X and an execution budget Y can be consider as a periodic task with a period X and an execution time Y
  - The system can then use analysis scheme of RM or EDF