CISTER - Research Center in
Real-Time & Embedded Computing Systems

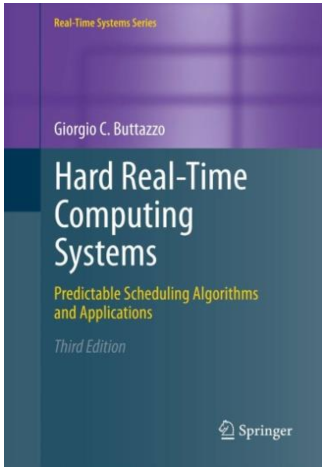
Introduction to Real-Time Systems

Cláudio Maia

CISTER Summer Internship 2017

Course Details

- 2 Modules of 2 hours each
- Theoretical Assignment
  - RTA for RM Algorithm
- Reference Book
- Reference Notes



What is a Real-Time System?

- A RTS is a system which correct behavior depends not only on the result of the computation but also on the time at which the results are produced
- Examples of RTS that you know?

What is a Real-Time System?

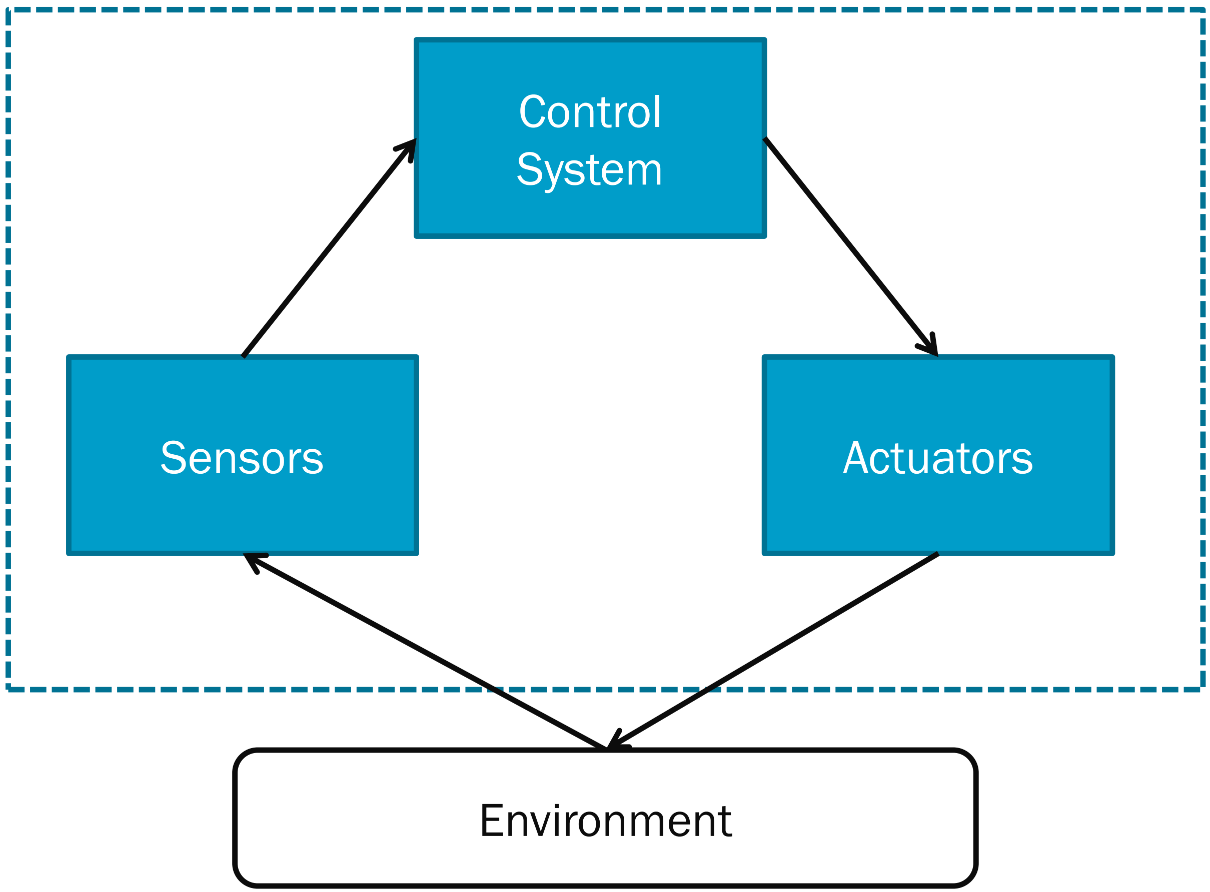
- A RTS is a system which correct behavior depends not only on the result of the computation but also on the time at which the results are produced
- Examples of RTS that you know?
  - Flight control systems
  - Robotics and industrial automation
  - Automotive and Railway
  - Military and space applications

Airbag Example

- https://www.youtube.com/watch?v=Bw0Ps8-KDIQ

- Requirement of an airbag controller system?
  - When a crash is detected, the system should fire the airbag
  - Fire early: System is inefective
  - Fire later: Passengers may get hurt

Generic Model of RTS



Notion of Time

- Time is strictly related to the environment in which the system operates

- It does not make sense to design a real-time system without considering the timing characteristics of the environment
  - E.g., a flight control system

Notion of Time

- The term real time is subject to different interpretations (not always correct!!)
  - E.g., a system operates in real time if it is able to quickly react to external events
    - Interpretation: A system is considered to be real-time if it is fast
    - Is this interpretation correct?

Notion of Time

- The term real time is subject to different interpretations (not always correct!!)
  - E.g., a system operates in real time if it is able to quickly react to external events
    - Interpretation: A system is considered to be real-time if it is fast
    - Is this interpretation correct?
      - Fast computing: minimize average response time
      - Real-time computing: meet the individual timing requirement of each task

RTS Properties

- Timeliness
  - Results have to be correct in their computed value and in the time domain
- Predictability
  - Predict the consequences of any scheduling decision
  - Guarantee in advance that all critical timing requirements will be met

Predictability

- How can predictability be affected in a system?
  - Internal characteristics of the processor: pipelining, cache memory, direct memory access (DMA), etc.
    - Improve the average performance of the processor but introduce non-determinism
    - Affect the precise estimation of the worst-case execution times (WCETs)
  - Scheduling algorithms
  - Synchronization mechanisms

Scheduling Concepts

- **Process:** computation that is executed by the CPU in a sequential manner
  - synonym of task and thread
- **Concurrent tasks:** tasks that can overlap their execution in time
- **Scheduling policy:** Criterion that defines how the CPU is assigned to tasks
- **Scheduling Algorithm:** set of rules that determine the order in which tasks are executed
- **Dispatching:** Operation concerned with the allocation of the CPU to a task selected by the scheduling algorithm

Task States

- Task can be either in
  - Execution: if it has been selected by the scheduling algorithm
  - Waiting for the CPU: if another task is executing

- **Active task:** task that can potentially execute on the processor, independently on its actual availability

- **Ready task:** task waiting for the processor
-
- **Running task:** task in execution

- **Ready queue:** Queue that stores all ready tasks waiting for the processor

Scheduling Concepts

- **Preemption:** Operation of suspending the running task (without requiring its cooperation) and inserting it into the ready queue in order to resume it later

- Importance of preemption
  - Tasks performing exception handling may need to preempt existing tasks so that responses to exceptions may be issued in a timely fashion
  - Tasks having different importance (phone call vs. video playing)
    - Critical tasks can start as soon as they arrive.
  - Allows for higher processor utilization
  - Destroys program locality and introduces runtime overheads
    - Inflates the execution time of tasks

Scheduling

- A **schedule** is an assignment of tasks to the processor so that each task is executed until completion

- **Schedule**: function $\sigma : R+ \longrightarrow N$ such that $\forall t \in R+$, $\exists t1, t2$ such that $t \in [t1, t2)$ and $\forall t \in [t1, t2)$ $\sigma(t) = \sigma(t)$
  - $\sigma(t)$ is an integer step function
  - $\sigma(t) = k$, with $k > 0$, means that task k is executing at time t
  - $\sigma(t) = 0$, CPU is idle

Schedule Example

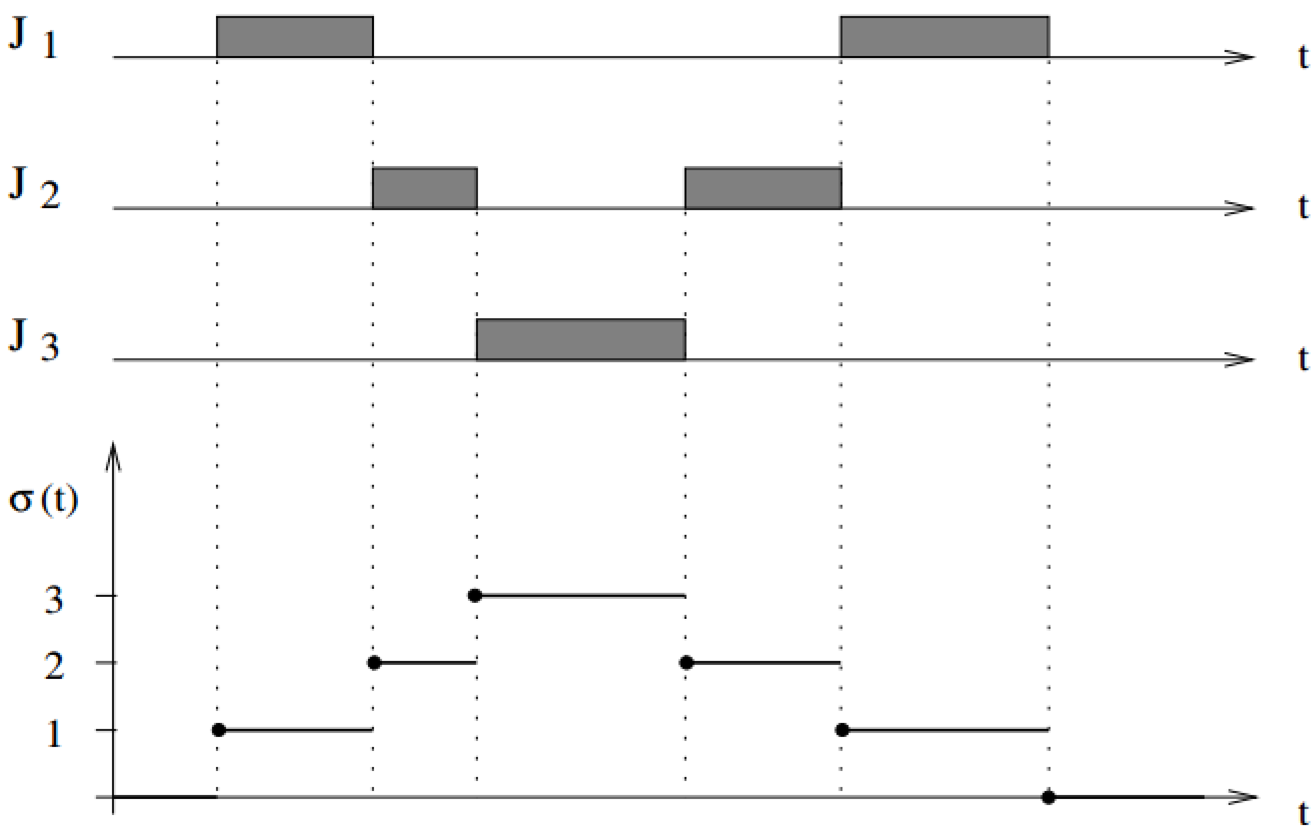
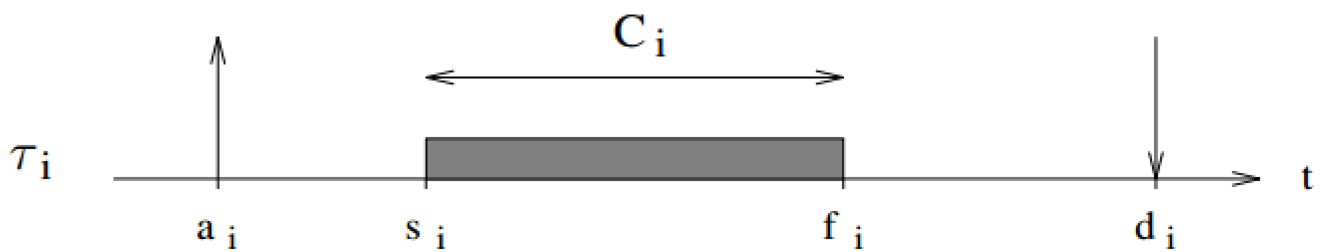


Figure 2.3 Example of a preemptive schedule.

Task Parameters



Task Parameters

- **Arrival** (or release time r_i) (a_i): time when a task becomes ready for execution
- **Computation time** (C_i): time necessary for completion of a task
- **Absolute deadline** (d_i): time before which a task should be completed
- **Start time** (s_i): time at which a task starts its execution
- **Finishing time** (f_i): time at which a task finishes its execution

Task Parameters

- Relative deadline: Di = di – ai
- Response time: Ri = fi – ai
- Lateness (delay of a task): Li = fi – di (can be negative)
- Tardiness (exceeding time, time a task stays active after its deadline): E_i = max(0, Li)
- Slack time (laxity): Xi = di – ai – C , maximum time that a task can be delayed on its activation to complete within deadline

Types of RT Tasks

- Hard: producing results after its deadline may cause catastrophic consequences for the system under control

- Soft: producing the results after its deadline has still some utility for the system under control, nevertheless causing a performance degradation

- Firm: producing the results after its deadline is useless for the system, but does not cause any damage

Periodicity

- Regularity of a task's activation
- Periodic tasks: infinite sequence of identical activities, called **instances** or **jobs**, that are regularly activated at a constant rate
- Characterized by (Ci, Ti, Di)
  - sensory data acquisition
  - control loops
  - system monitoring

Periodicity

- Aperiodic tasks: task activations are not regularly interleaved
- Sporadic task: task where its consecutive jobs are separated by a minimum inter-arrival time

Scheduling Algorithms

- Preemptive vs. Non-preemptive
  - Preemptive algorithms: the running task can be interrupted at any time in order to assign the processor to another active task and according to a predefined scheduling policy
  - Non-preemptive algorithms: a task, once started, is executed by the processor until completion
    - Scheduling decisions are taken when the task completes

Scheduling Algorithms

- Static vs. Dynamic
  - Static algorithms: scheduling decisions are made based on fixed parameters, assigned to tasks before their activation
  - Dynamic algorithms: scheduling decisions are based on dynamic parameters that may change during system evolution

Scheduling Algorithms

- Offline vs. Online
  - Offline: scheduling algorithm is executed on the entire task set before task's activation
    - Schedule is stored in a table and later executed by a dispatcher
  - Online: scheduling decisions are taken at runtime every time a new task enters the system or when a running task terminates

Scheduling Algorithms

- Optimal vs. Heuristic
  - Optimal: the algorithm minimizes a given cost function defined over the task set or finds a feasible schedule (if it exists) if no cost function is specified
  - Heuristic: there is a heuristic function taking scheduling decisions
    - A heuristic algorithm tries to find the optimal schedule, but does not guarantee to find it

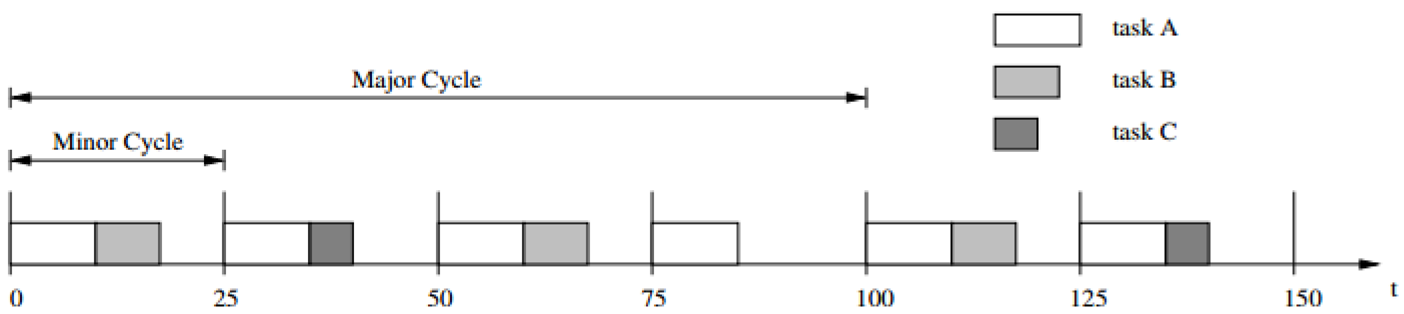
Cyclic Executive

- One of the most used approaches to handle periodic tasks

- Static, Off-line schedule
- Divides the temporal axis into slots of equal length
  – One or more tasks can be executed
- Minor cycle: duration of the time slot
- Major cycle: minimum interval of time after which the schedule repeats itself
  – Also called **hyperperiod**

Cyclic Executive

- Task periods
  - Task A = 25
  - Task B = 50
  - Task C = 100



Cyclic Executive

- Advantages
  - Simple to implement
    - timer to interrupt with a period equal to the minor cycle
    - main program invokes tasks in the order given in the major cycle
  - Runtime overhead is low
    - No context switch is needed

Cyclic Executive

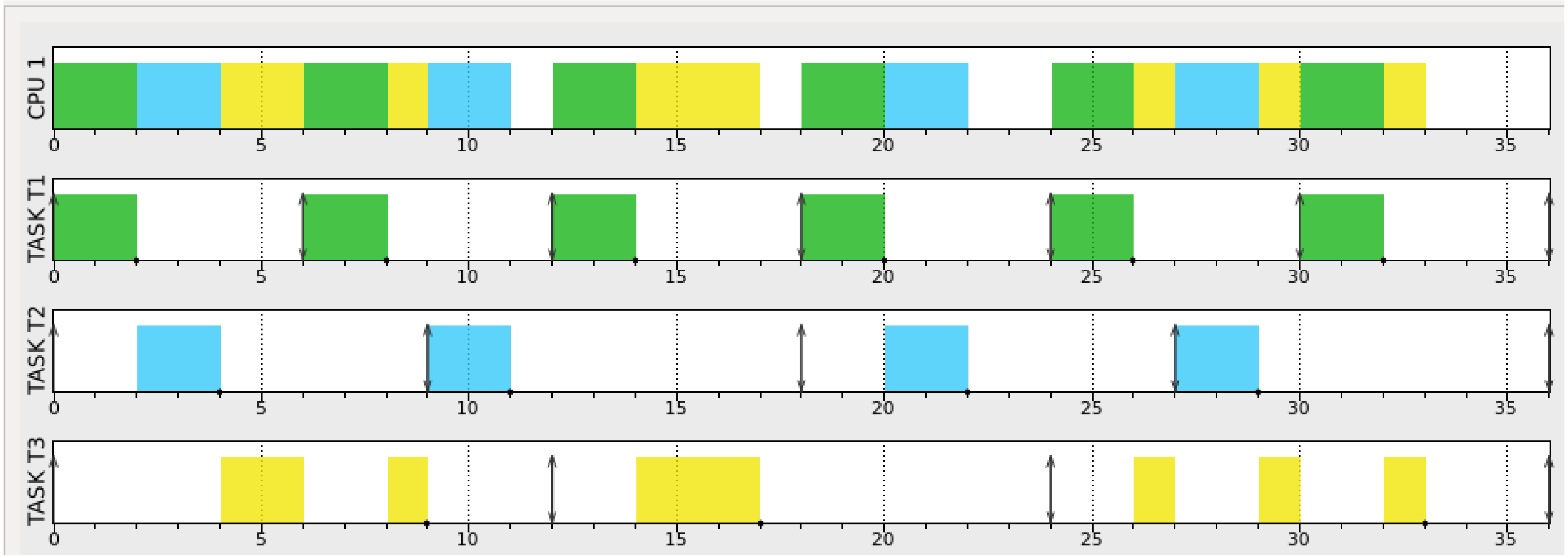
- Disadvantages
  - Affected by overload conditions
    - Domino effect or inconsistent state
  - Sensitive to application changes
    - Scheduling sequence may need to be reconstructed from scratch
      - Change in computation time or change in frequency
  - Difficult to handle aperiodic activities efficiently without changing the task sequence

Rate Monotonic (RM)

- Priority Assignment Rule
  - Assigns priorities to tasks according to their request rates
  - Tasks with higher request rates (i.e., , with shorter periods) have higher priorities
- Static priority assignment
  - Periods are constant and equal to the deadline of the task (implicit deadlines)
  - A priority is assigned to the task before execution and does not change over time
- Preemptive/Non-preemptive
  - If preemptive, the currently executing task is preempted if a new task arrives with a shorter period

Rate Monotonic

- Example 1
  - $\tau_1 = (2, 6, 6)$, $\tau_2 = (2, 9, 9)$, $\tau_3 = (3, 12, 12)$



Deadline Monotonic (DM)

- DM's priority assignment weakens the restriction "period equals deadline" in static priority schemes

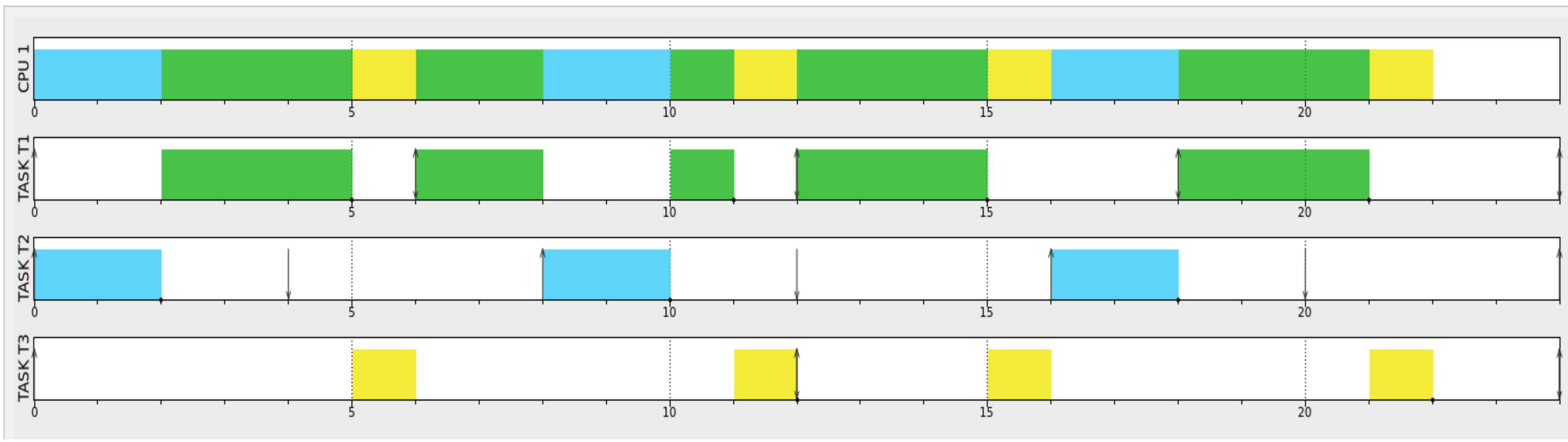
- Tasks can have relative deadlines (D_i) less than or equal to their period
  - Constrained deadlines

Deadline Monotonic (DM)

- Priority Assignment Rule
  - Each task is assigned a fixed priority inversely proportional to its relative deadline
  - Task with the shortest relative deadline is executed first
- Static priority assignment
  - Deadlines are constant
- Preemptive/Non-preemptive
  - If preemptive, the currently executing task is preempted if a new task with shorter relative arrives into the system

Deadline Monotonic (DM)

- Example
  - $\tau_1 = (3, 6, 6)$, $\tau_2 = (2, 4, 8)$, $\tau_3 = (2, 12, 12)$

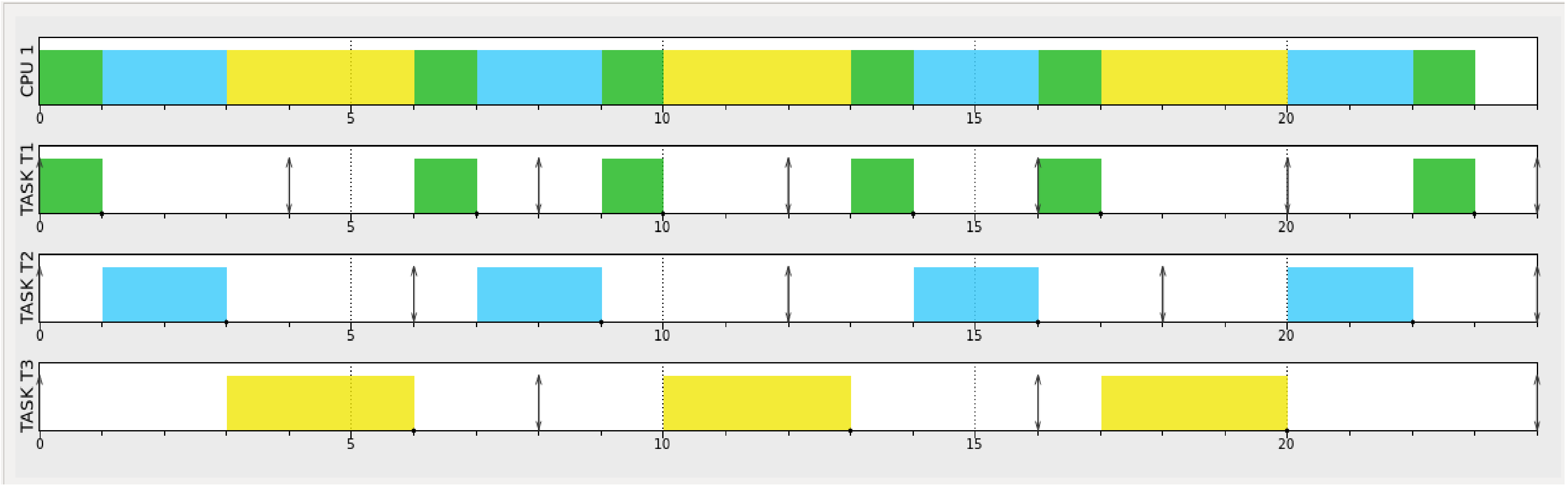


Earliest Deadline First (EDF)

- Priority assignment rule
  - At any instant the task with the earliest absolute deadline among all ready tasks is the one that should be executing
- Dynamic priority assignment
  - Tasks are selected according to the absolute deadline
- Preemptive/Non-preemptive
- Does not depend on the task's period

Earliest Deadline First (EDF)

- Example
  - $\tau_1 = (1,4,4)$, $\tau_2 = (2,6,6)$, $\tau_3 = (3,8,8)$



Schedulability Assumptions

- The instances of a periodic task i are regularly activated at a constant rate (i.e., period T_i)
- All instances of a periodic task have the same Ci and implicit Di
- All tasks in the set are independent
  - no precedence relations and no resource constraints

Utilization Analysis

- Task utilization: fraction of processor time spent executing task T_i
  - $U_i = C_i/T_i$

- Processor Utilization factor: fraction of processor time spent in the execution of the task set

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i}$$

  - computational load on the CPU due to the periodic task set

Utilization Analysis

- Schedulable Utilization of a scheduling algorithm ($U_{UB}(\Gamma, A)$)
  - Maximum value of U below which a task set is schedulable and above which is not schedulable
  - Depends on the task set and the algorithm

- Least upper bound $U_{lub}(A)$ of the processor utilization factor
  - Minimum of the utilization factors over all task sets that fully utilize the processor
  - $U_{lub}(A) = \min_{\Gamma} U_{UB}(\Gamma, A)$

Rate Monotonic

- Schedulable Utilization

$$\sum_{i=1}^{n} \frac{Ci}{Ti} \leq n(2^{\frac{1}{n}} - 1)$$

n	1	2	3	4
U	1	0.82	0.78	0.76

- As the value of n increases the schedulable utilization converges to
  - U_{lub} = ln 2 ~ 0.69

- The above condition is suficient but not necessary

Rate Monotonic

- The schedulability test consists in
  - Compute task set utilization U
  - If $U \leq U_{lub}$, the task set is schedulable
  - if $U > 1$ the task set is not schedulable
  - if $U_{lub} < U \leq 1$, the task set may or may not be schedulable

EDF

- Schedulable Utilization

$$\sum_{i=1}^{n} \frac{Ci}{Ti} \leq 1$$

- EDF is an optimal algorithm, in the sense that if a task set is schedulable, then it is schedulable by EDF

EDF vs RM

- RM is optimal among fixed-priority algorithms
  - If a task set can be scheduled by fixed-priority algorithm then it can be scheduled by Rate Monotonic algorithm

- EDF can schedule all task sets that can be scheduled by RM (in fact any FP algorithm), but not vice versa

RTA – Preemptive RM

- Verify if a task set is schedulable by determining the worst case response time of each task in the set
  - Compute the WCRT Ri for task τ_i
  - If Ri ≤ Di, then the task is schedulable
  - Else the task is not schedulable

- It is a necessary and sufficient test

- The worst-case processor demand occurs when all tasks are released simultaneously (Liu and Layland, 1973)
  - Also known as the critical instant

RTA – Preemptive RM

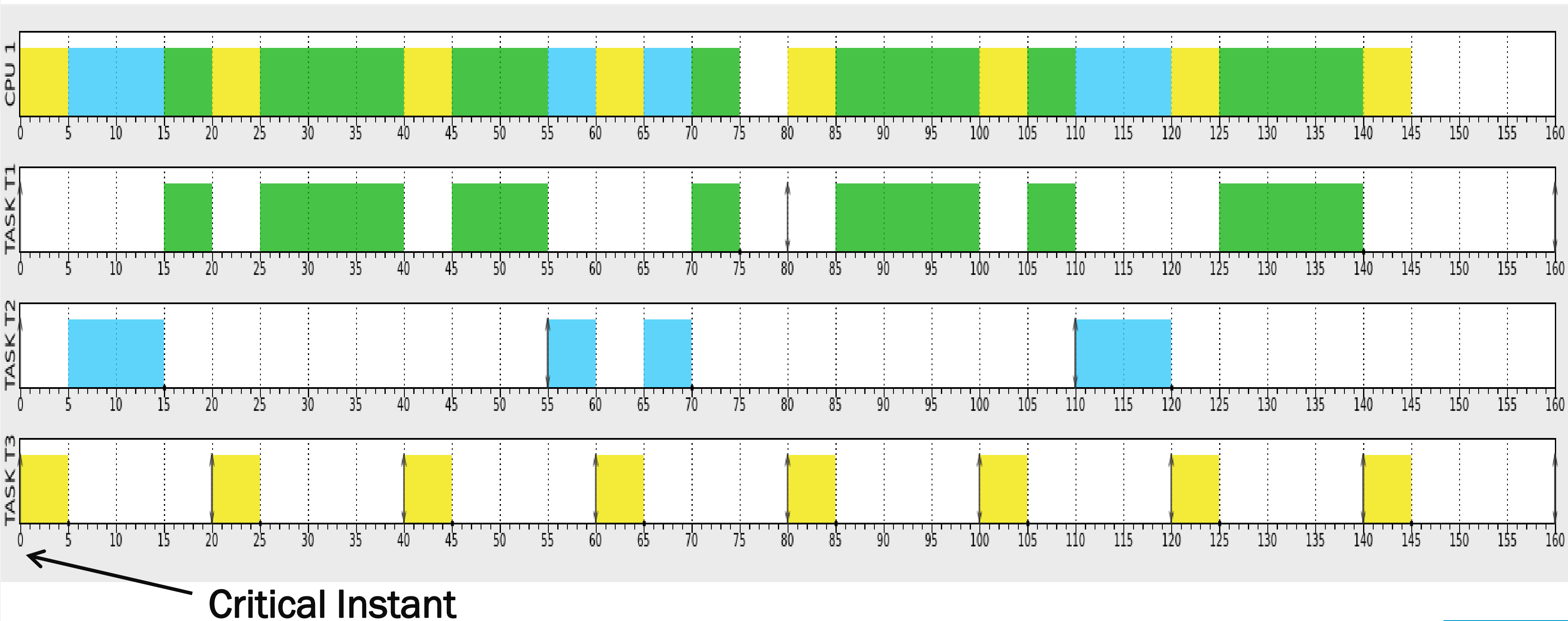
- For each task τ_i, the response-time R_i is given by the sum of its WCET and the interference imposed by higher priority tasks

$$R_i = C_i + I_i$$

- Interference
  - Cumulative length of all intervals of time in which a task is ready to execute but it cannot due to the execution of higher priority tasks

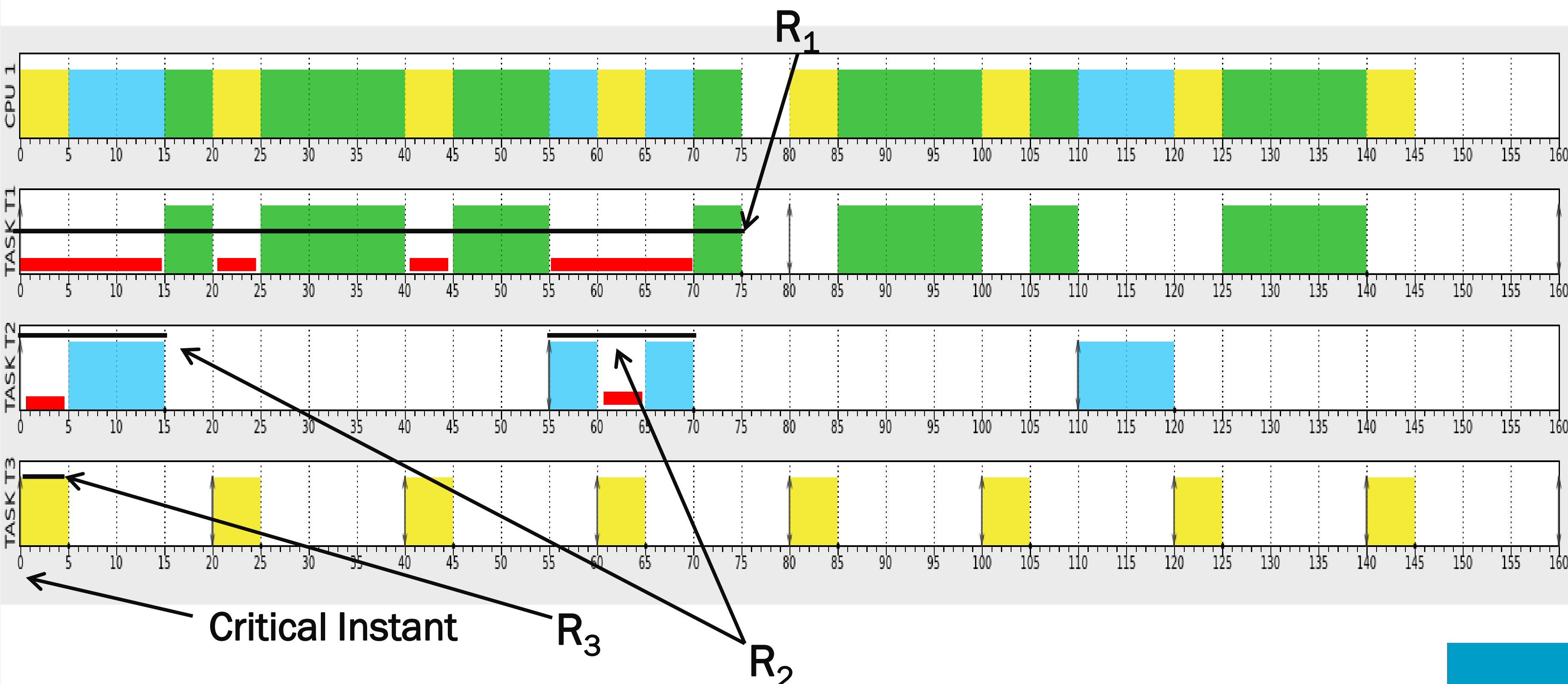
RTA – Preemptive RM

- Example using preemptive RM
  - $\tau_1 = (35, 80)$, $\tau_2 = (10, 55)$, $\tau_3 = (5, 20)$
- What are the response times and interference suffered by each task?



RTA – Preemptive RM

- Example using preemptive RM
  - $\tau_1 = (35, 80)$, $\tau_2 = (10, 55)$, $\tau_3 = (5, 20)$
- What are the response times and interference suffered by each task?



RTA – Preemptive RM

- In preemptive RM a task may be preempted until the end of its execution
  - But the end of its execution is R_i which is the value we want to compute

- What tasks interfere with the lowest priority task?
- How many times do each task execute within the interval R_1?

RTA – Preemptive RM

- How many times do each task execute within the interval R_1?
  - Task 2: $\left\lceil \dfrac{R_1}{T_2} \right\rceil = \left\lceil \dfrac{75}{55} \right\rceil = 2$
  - Task 3: $\left\lceil \dfrac{R_1}{T_3} \right\rceil = \left\lceil \dfrac{75}{20} \right\rceil = 4$
  - Meaning: Each time Task 2 executes, it interferes 2 x 20 time units in the execution of Task 1

ceiling(x) returns the least integer greater than or equal to x

RTA – Preemptive RM

- How to compute the interference for a given task?

RTA – Preemptive RM

- How to compute the interference for a given task?

$$I_i = \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil \times C_j$$

- Replacing I_i in $R_i = C_i + I_i$ by the above eq.

$$R_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil \times C_j$$

RTA – Preemptive RM

- What is the issue with the following equation?

$$R_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil \times C_j$$

RTA – Preemptive RM

- What is the issue with the following equation?

$$R_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil \times C_j$$

- We are facing a recurrent equation

$$R_i^{n+1} = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i^n}{T_j} \right\rceil \times C_j$$

RTA – Preemptive RM

- To solve it we use $R_i^0 = C_i$
- The recurrence stops when
  - $R_i^{n+1} = R_i^n$ or $R_i \leq D_i, \forall i$

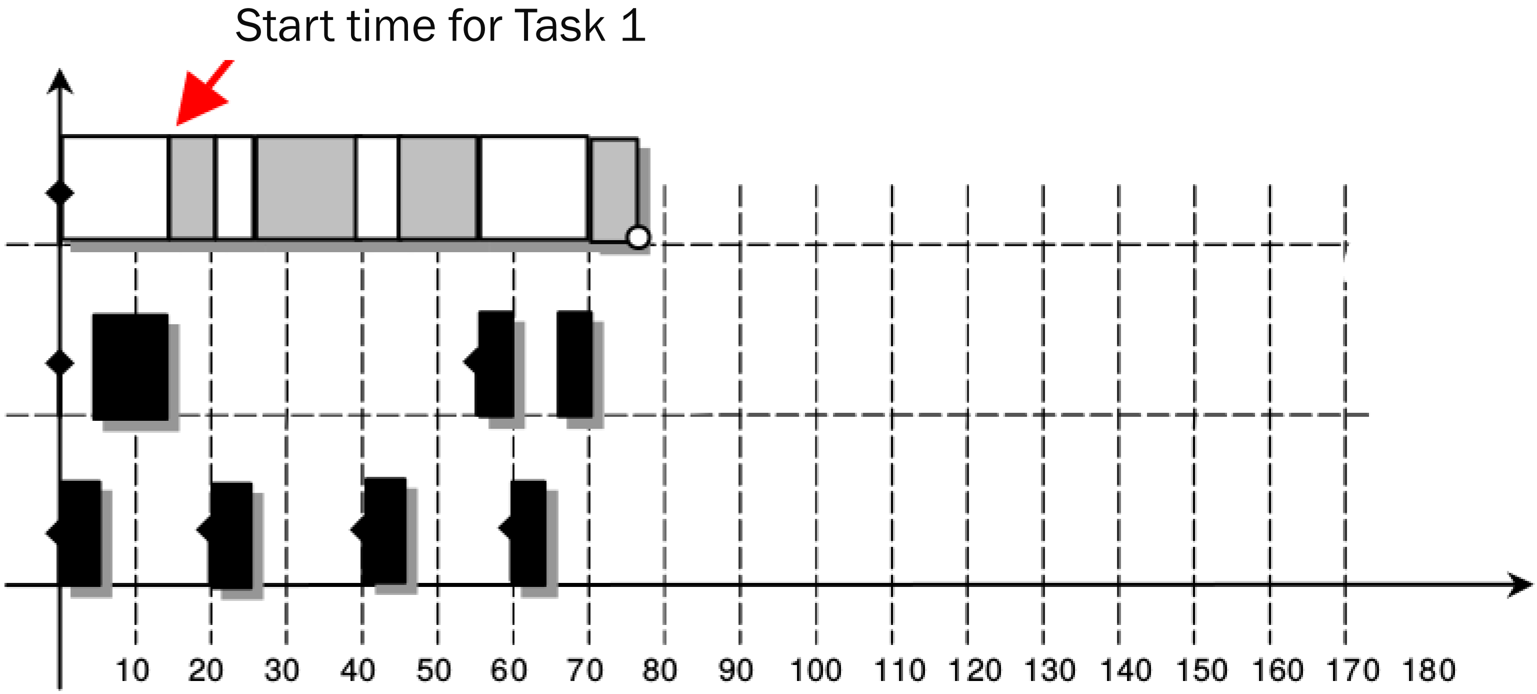
- We are determining the instant of time when
  - No higher priority task than task i is pending in the system
  - Task i already completed its execution

RTA – Preemptive RM

- Compute the WCRT for the above task set
  - $\tau_1 = (35, 80)$, $\tau_2 = (10, 55)$, $\tau_3 = (5, 20)$

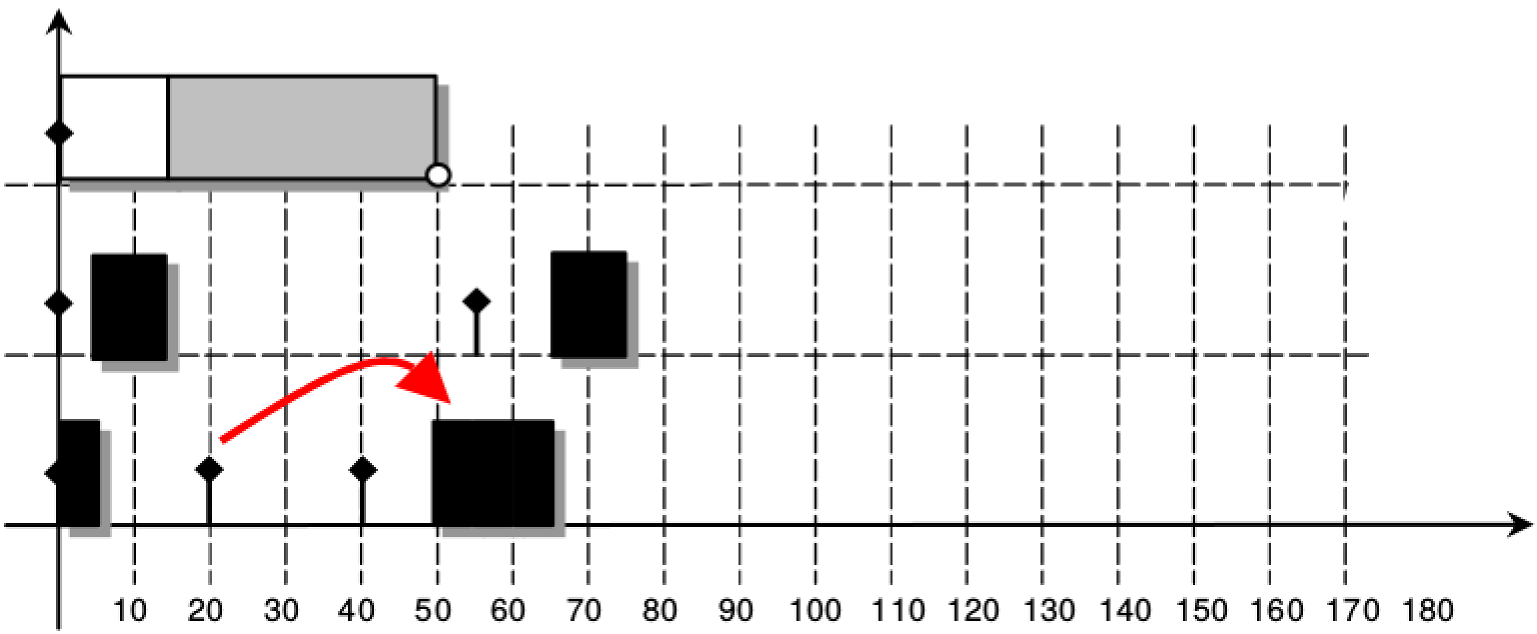
RTA – Non-Preemptive RM

- Preemptive case



RTA – Non-Preemptive RM

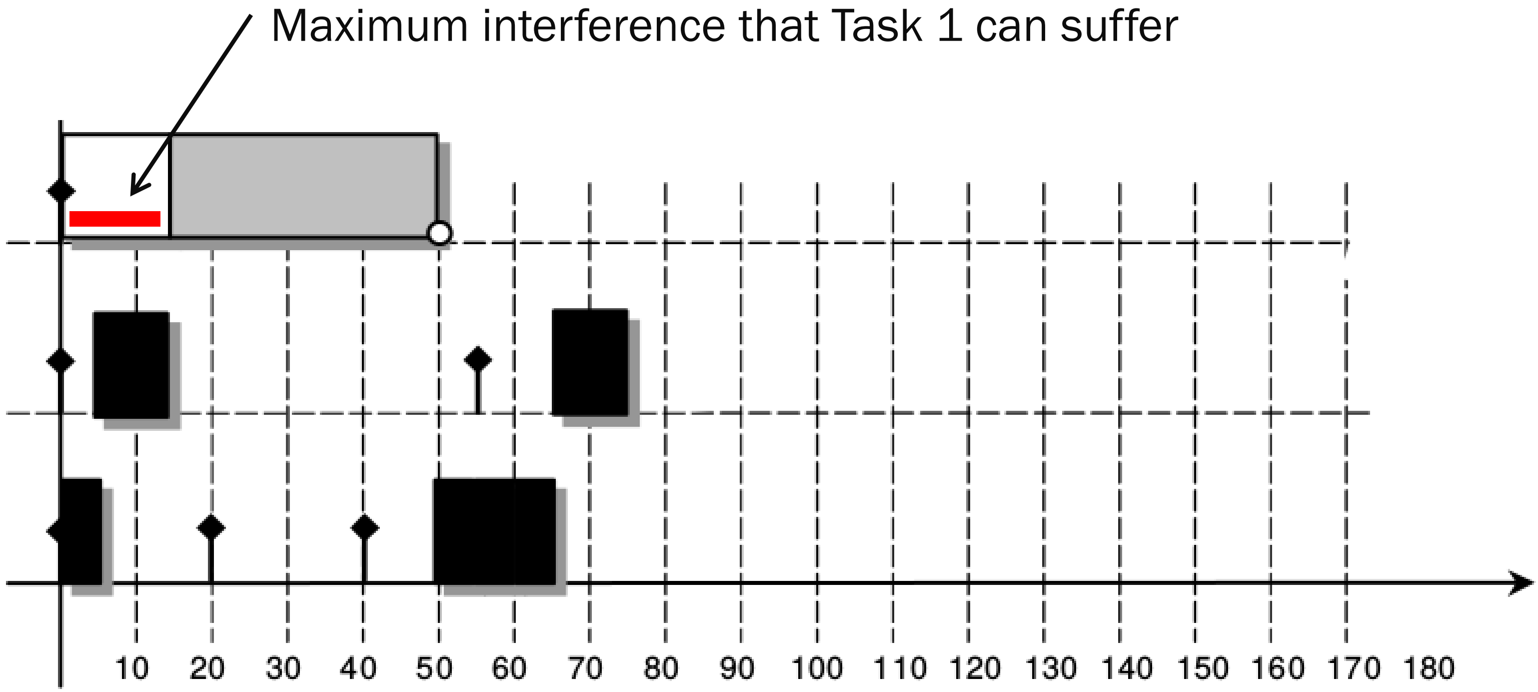
- If Task 1 is non-preemptive, then Task 3 misses its deadline at time instant 20



RTA – Non-Preemptive RM

- We are determining the instant of time when
  - No higher priority task than task i is pending in the system
  - At this point, task i executes without preemption
- Task does not depend on its response time

RTA – Non-Preemptive RM

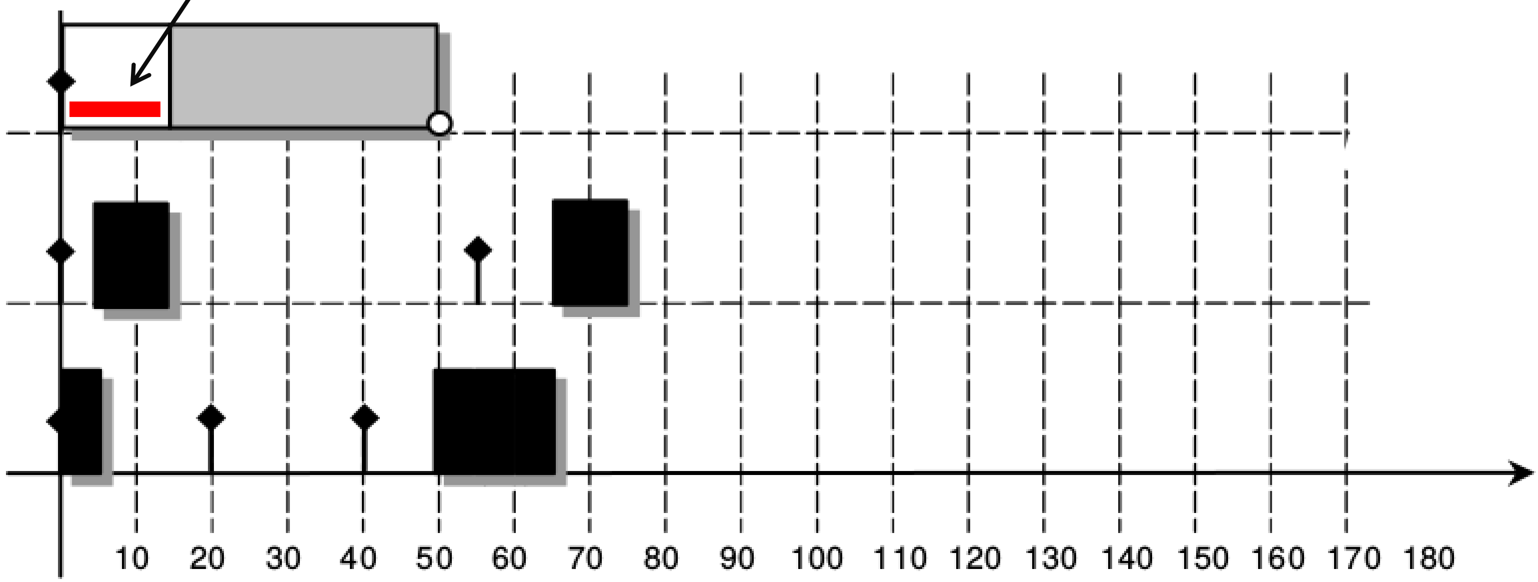


RTA – Non-Preemptive RM

- Instead of computing R_i as in the preemptive case, in the non-preemptive case we need to compute I_i first and then we add C_i time units to it

RTA – Non-Preemptive RM

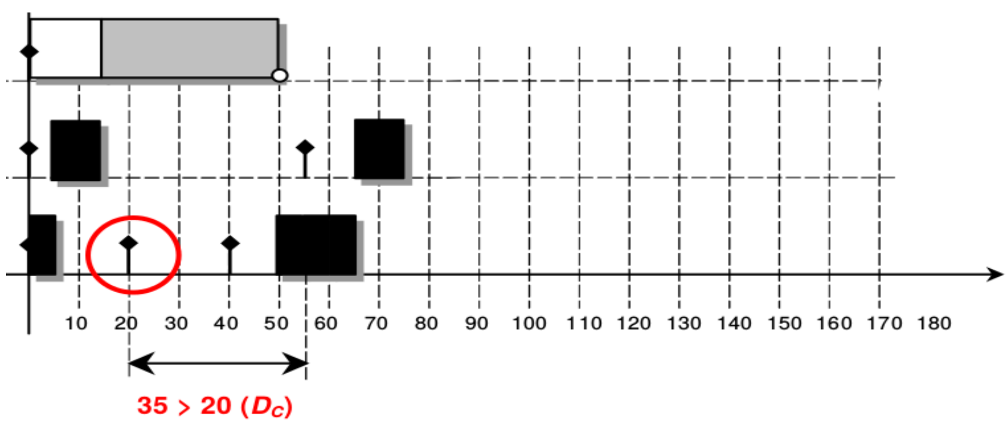
Maximum interference that Task 1 can suffer



$$I_i^{n+1} = \sum_{j \in hp(i)} \left\lceil \frac{I_i^n}{T_j} \right\rceil \times C_j$$

RTA – Non-Preemptive RM

- But in non preemptive systems, a task with lower priority can block the execution of a higher priority job
- If blocking is ignored, wrong results may be obtained

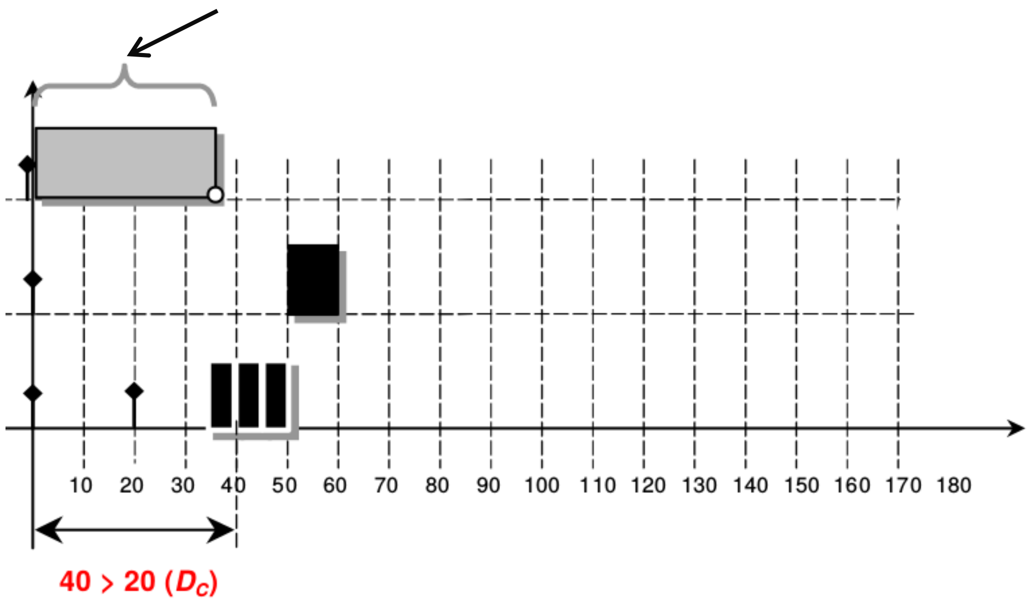


RTA – Non-Preemptive RM

- What is the highest blocking that task 3 can get?
  - Highest blocking situation occur when task 1 is released into the system epsilon time units before task 2 and task 3

RTA – Non-Preemptive RM

Largest blocking caused by task 1 in other higher priority tasks



RTA – Non-Preemptive RM

- In the non preemptive case the interference is given by

$$I_i^{n+1} = B_i + \sum_{j \in hp(i)} \left\lceil \frac{I_i^n}{T_j} \right\rceil \times C_j$$

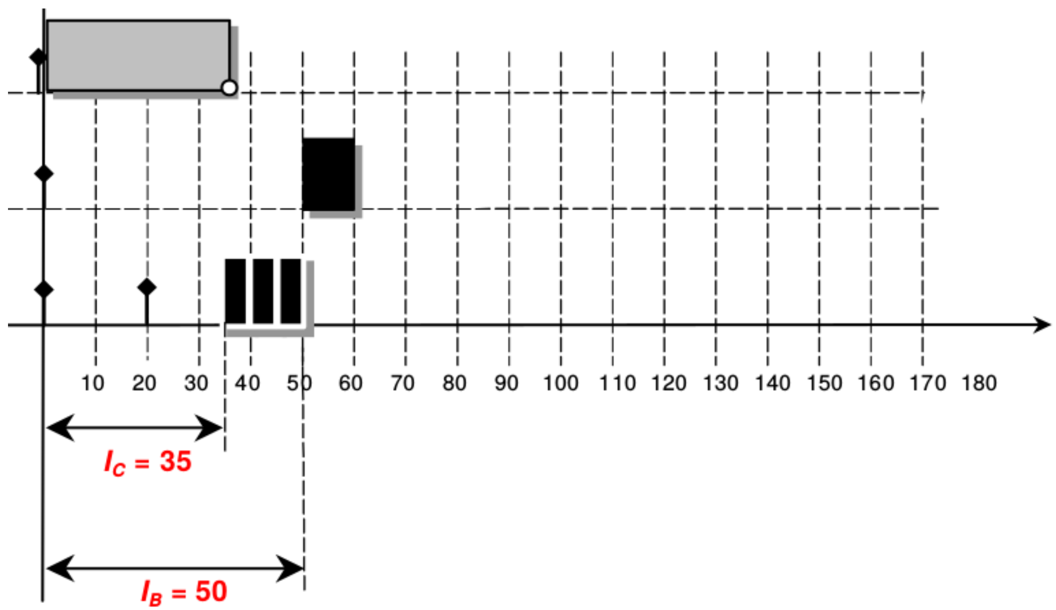
- The first value of the iteration is computed as follows

$$I_i^0 = B_i + \sum\nolimits_{j \in hp(i)} C_j$$

- where $Bi = \begin{cases} 0 & \text{, if i is the lowest priority task} \\ \max\{Cj\} & \text{, j are the task with lower priority than i} \end{cases}$

RTA – Non-Preemptive RM

- Interference values for the example



Summary

- Schedulability tests
  - Utilization based tests
    - Rate Monotonic and Earliest Deadline First
  - Response time Analysis
    - Preemptive and Non-preemptive RM
- RM
  - Utilization based test is sufficient and for large n it has a $U_{lub} = 0.69$
  - RTA: sufficient and necessary test for arbitrary deadlines and task with no offsets