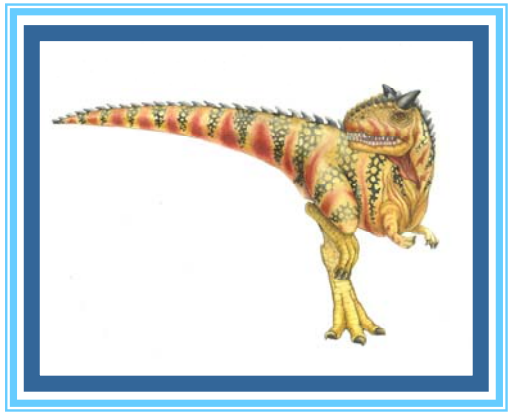
# Chapter 6: Real-Time Scheduling

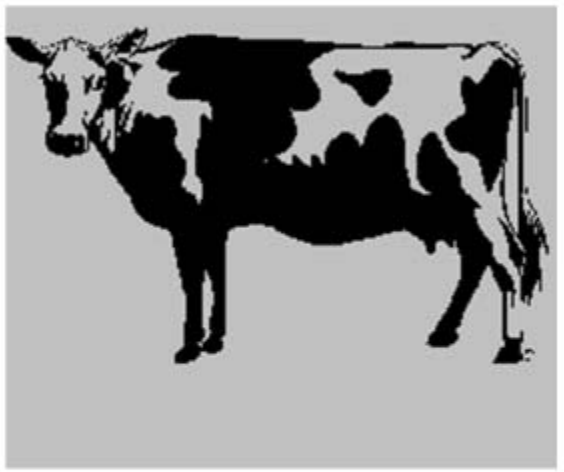


# Real-Time Scheduling Is Not Fair

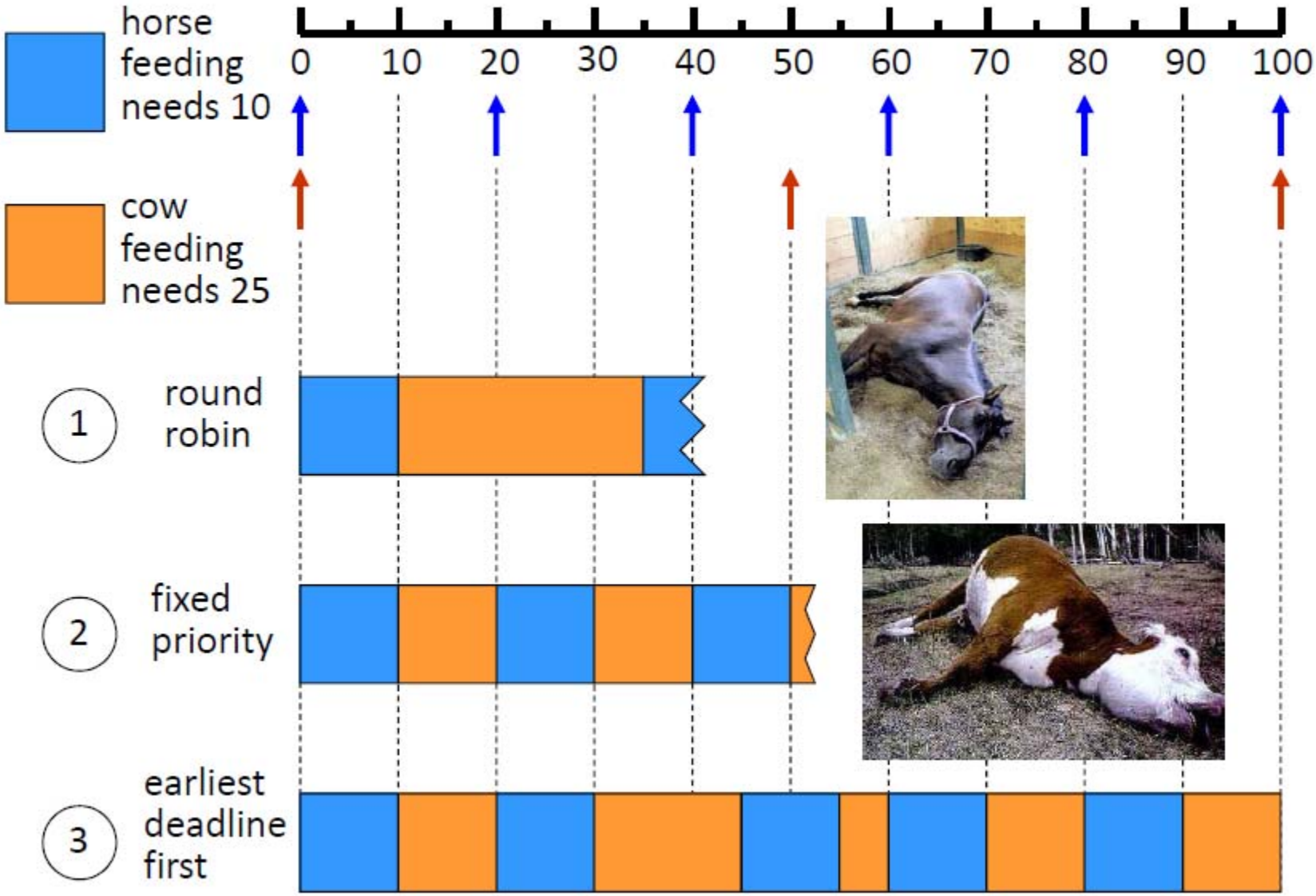
- Main goal of an RTOS scheduler is to meet task deadlines, instead of throughput, latency and response time, etc.
- If you have five homework assignments and only one is due in half an hour, you work on that one first
- Fairness does not help you meet deadlines

# The Cowboy Scheduling Problem

Feed horse periodically every 20 minutes for 10 minutes each period



Feed cow periodically every 50 minutes for 25 minutes each period

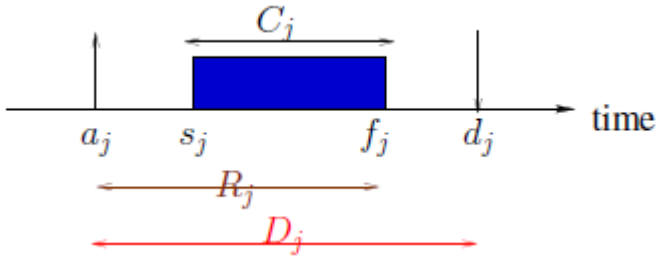

horse
feeding
needs 10
cow
feeding
needs 25
0
10
20
30
40
50
60
70
80
90
100
1
round
robin
2
fixed
priority
3
earliest
deadline
first

# Real-Time Scheduling Policies

- Important Questions for real-time scheduling
  - What scheduler is guaranteed to meet all task deadlines for a given workload?
  - Given a scheduler, how do we know that it will work for a given workload?
  - Is there an “optimal” scheduler independent of workload?

# Timing parameters of a job $J_j$

- Arrival time ($a_j$) or release time ($r_j$) is the time at which the job becomes ready for execution
- Computation (execution) time ($C_j$) is the time necessary to the processor for executing the job without interruption.
- Absolute deadline ($d_j$) is the time at which the job should be completed.
- Relative deadline ($D_j$) is the time length between the arrival time and the absolute deadline.
- Start time ($s_j$) is the time at which the job starts its execution.
- Finishing time ($f_j$) is the time at which the job finishes its execution.
- Response time ($R_j$) is the time length at which the job finishes its execution after its arrival, which is $f_j$ - $a_j$.




# Feasibility of Schedules and Schedulability

- A schedule is feasible if all jobs can be completed according to a set of specified constraints.
- A set of jobs is schedulable if there exists a feasible schedule for the set of jobs.
- A scheduling algorithm is optimal if it always produces a feasible schedule when one exists (under any scheduling algorithm).

# Evaluating A Schedule

- For a job $J_j$:
- Lateness $L_j$: delay of job completion with respect to its deadline.

$$L_j = f_j - d_j$$

- Tardiness $E_j$: the time that a job stays active after its deadline.

$$E_j = max\{0, L_j\}$$

- Laxity (or Slack Time)($X_j$): The maximum time that a job can be delayed and still meet its deadline.

$$X_j = d_j - a_j - C_j$$

# Metrics of Scheduling Algorithms (for Jobs)

- Given a set ***J*** of *n* jobs, the common metrics are to minimize:
  - Average response time: $\sum_{J_j \in \mathbb{J}} \frac{f_j - a_j}{|\mathbb{J}|}$
  - Makespan (total completion time): $\max_{J_j \in \mathbb{J}} f_j - \min_{J_j \in \mathbb{J}} a_j$
  - Total weighted response time: $\sum_{J_j \in \mathbb{J}} w_j (f_j - a_j)$
  - Maximum latency: $L_{max} = \max_{J_j \in \mathbb{J}} (f_j - d_j)$
- Number of late jobs: $N_{late} = \sum_{J_j \in \mathbb{J}} miss(J_j)$, where $miss(J_j) = 0$ if $f_j <= d_j$, and $miss(J_j) = 1$ otherwise.

# Hard/Soft Real-Time Systems

- Hard Real-Time Systems
  - If any hard deadline is ever missed, then the system is incorrect
  - The tardiness for any job must be 0
  - Examples: Nuclear power plant control, flight control
- Soft Real-Time Systems
  - A soft deadline may occasionally be missed
  - Various definitions for “occasionally”
    - minimize the number of tardy jobs, minimize the maximum lateness, etc.
  - Examples: Telephone switches, multimedia applications
- We mostly consider hard real-time systems in this lecture.

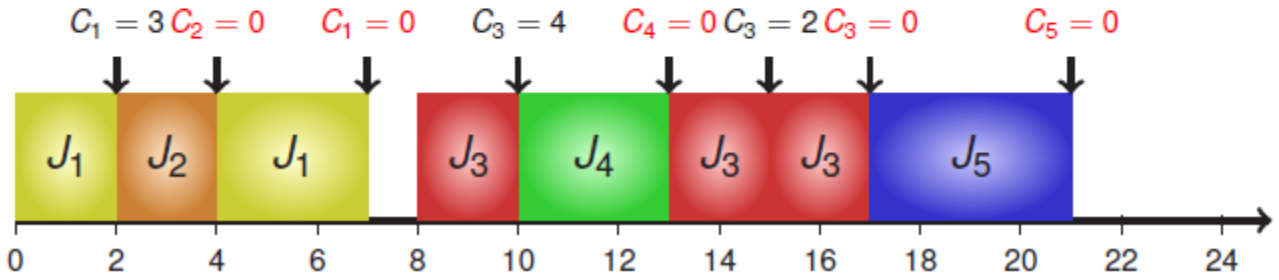
# An Example: Shortest-Job-First (SJF)

- At any moment, the system executes the job with the shortest remaining time among the jobs in the ready queue.

| | $J_1$ | $J_2$ | $J_3$ | $J_4$ | $J_5$ |
|---|---|---|---|---|---|
| $a_j$ | 0 | 2 | 8 | 10 | 15 |
| $C_j$ | 5 | 2 | 6 | 3 | 4 |
| $d_j$ | 6 | 8 | 20 | 14 | 22 |

Exercise

What is the average response time of the above schedule?

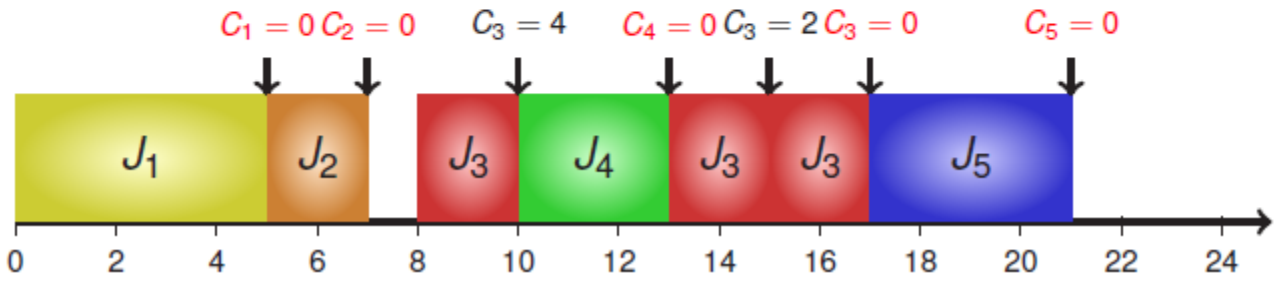



# An Example: Earliest-Deadline-First (EDF)

- At any moment, the system executes the job with the earliest absolute deadline among the jobs in the ready queue.

| | $J_1$ | $J_2$ | $J_3$ | $J_4$ | $J_5$ |
|---|---|---|---|---|---|
| $a_j$ | 0 | 2 | 8 | 10 | 15 |
| $C_j$ | 5 | 2 | 6 | 3 | 4 |
| $d_j$ | 6 | 8 | 20 | 14 | 22 |

Exercise
What is the average response time of the above schedule?

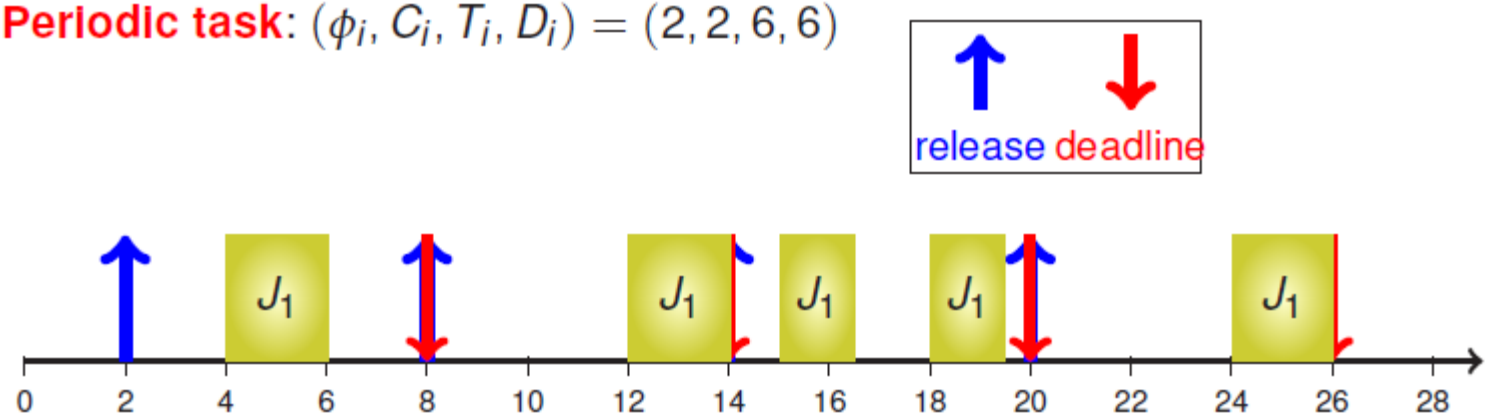



# Recurrent Task Models

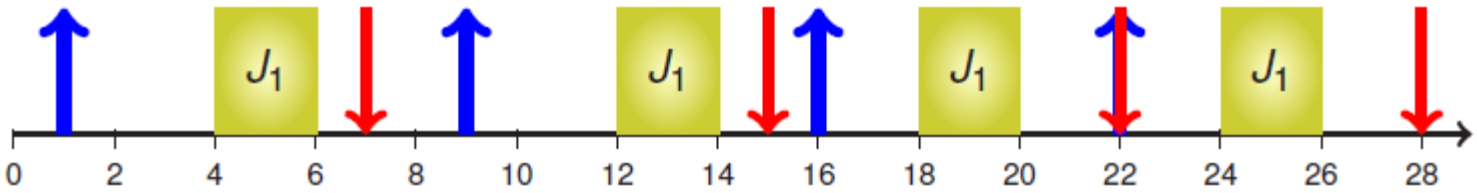
- When jobs (usually with the same computation requirement) are released recurrently, they can be modeled by a recurrent task
- Periodic Task $t_i$:
  - A job is released exactly and periodically by a period $T_i$
  - A phase $\varphi_i$ indicates when the first job is released
  - A relative deadline $D_i$ for each job from task $t_i$
  - ($\varphi_i$, $C_i$ , $T_i$, $D_i$) is the specification of periodic task $t_i$, where $C_i$ is the worst-case execution time.
- Sporadic Task $t_i$:
  - $T_i$ is the minimal time between any two consecutive job releases
  - A relative deadline $D_i$ for each job from task $t_i$
  - ($C_i$, $T_i$, $D_i$) is the specification of sporadic task $t_i$ , where $C_i$ is the worst-case execution time.
- Aperiodic Task: Identical jobs released arbitrarily.

# Examples of Recurrent Task Models

**Periodic task**: $(\phi_i, C_i, T_i, D_i) = (2, 2, 6, 6)$




**Sporadic task**: $(C_i, T_i, D_i) = (2, 6, 6)$



# Relative Deadline <=> Period

- For a task set, we say that the task set is with
- implicit deadline when the relative deadline $D_i$ is equal to the period $T_i$ , *i.e.*, $D_i = T_i$, for every task $t_i$,
- constrained deadline when the relative deadline $D_i$ is no more than the period $T_i$, *i.e.*, $D_i <= T_i$, for every task $t_i$, or
- arbitrary deadline when the relative deadline $D_i$ could be larger than the period $T_i$ for some task $t_i$.

# Some Definitions for Periodic Tasks

- The jobs of task $t_i$ are denoted $J_{i,1}$, $J_{i,2}$, . . . . . ..
- Synchronous system: Each task has a phase of 0.
- Asynchronous system: Phases are arbitrary.
- Hyperperiod: Least common multiple (LCM) of $T_i$.
- Task utilization of task $t_i$ : $u_i = C_i / T_i$.
- System utilization: $\sum_{\tau_i} u_i$.

# Feasibility and Schedulability for Recurrent Tasks

- A schedule is feasible if all the jobs of all tasks can be completed according to a set of specified constraints.
- A set of tasks is schedulable if there exists a feasible schedule for the set of tasks.
- A scheduling algorithm is optimal if it always produces a feasible schedule when one exists (under any scheduling algorithm).

# Schedulability Analysis

- Schedulability for Static-Priority Scheduling
  - Utilization-Based Analysis (Relative Deadline = Period)
  - Demand-Based Analysis
- Schedulability for Dynamic-Priority Scheduling

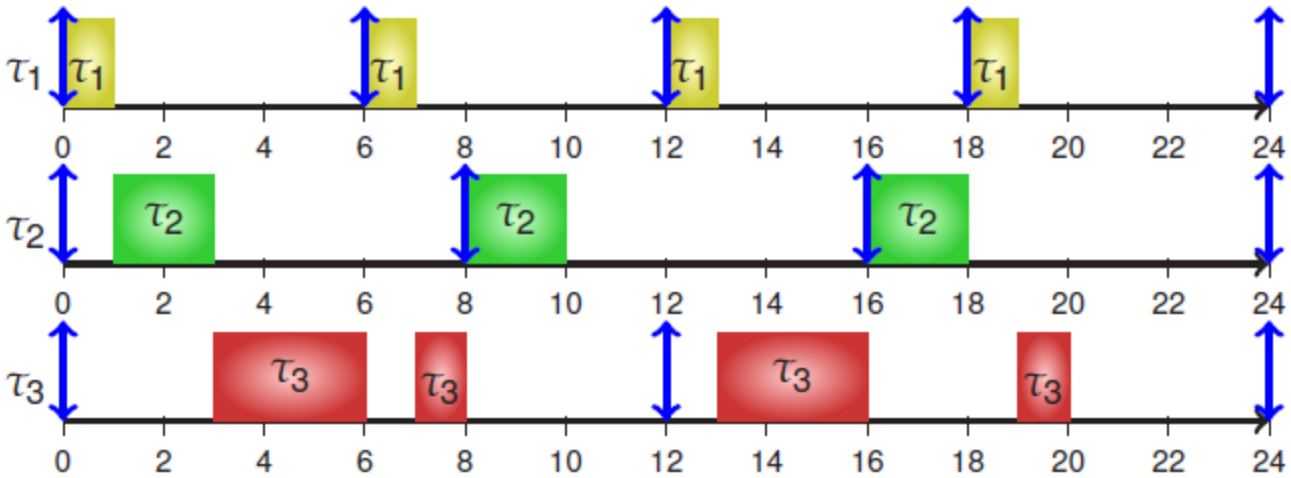
# Static-Priority Scheduling

- Different jobs of a task are assigned the same priority.
  - $\pi_i$ is the priority of task $t_i$.
  - $HP_i$ is the subset of tasks with higher priority than $t_i$.
  - Note: we will assume that no two tasks have the same priority.
- We will implicitly index tasks in decreasing priority order, *i.e.*, $t_i$ has higher priority than $t_k$ if $i < k$.
- Which strategy is better or the best?
  - largest execution time first?
  - shortest job first?
  - least-utilization first?
  - most importance first?
  - least period first?

# Rate-Monotonic (RM) Scheduling

- Priority Definition: A task with a smaller period has higher priority, in which ties are broken arbitrarily.

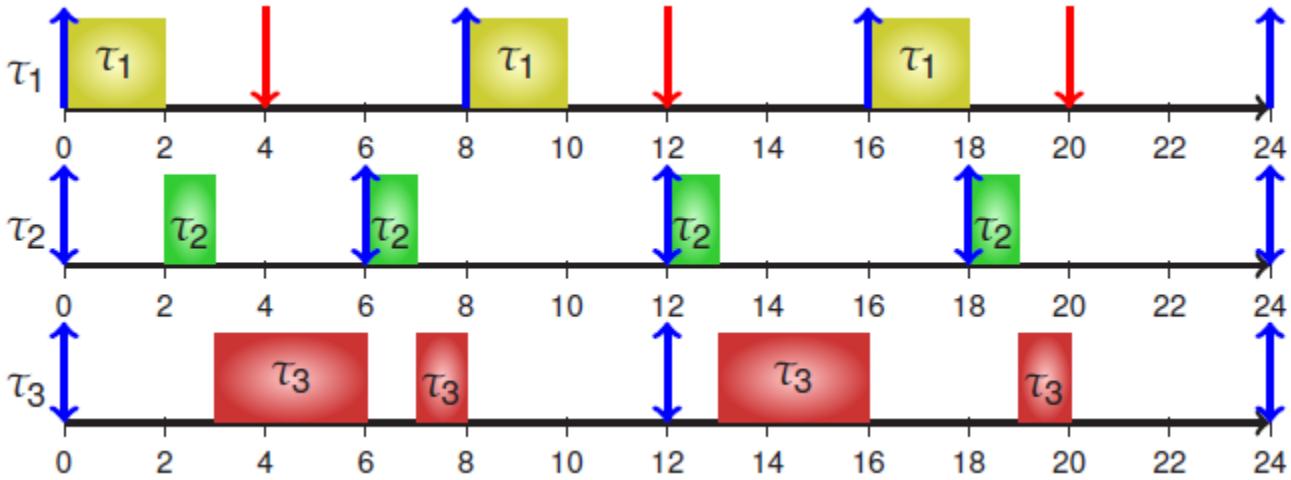
- Example Schedule: $t_1$ = (1, 6, 6), $t_2$ = (2, 8, 8), $t_3$ = (4, 12, 12). [($C_i$, $T_i$, $D_i$)]




# Deadline-Monotonic (DM) Scheduling

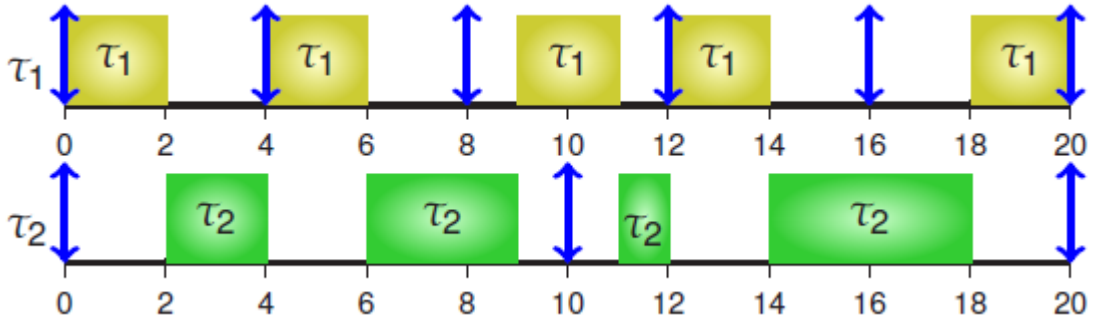
- Priority Definition: A task with a smaller relative deadline has higher priority, in which ties are broken arbitrarily.

- Example Schedule: $t_1$ = (2, 8, 4), $t_2$ = (1, 6, 6), $t_3$ = (4, 12, 12). [($C_i$, $T_i$, $D_i$ )]




# Optimality (or not) of RM and DM

- Example Schedule: $t_1$ = (2, 4, 4), $t_2$ = (5, 10, 10)




- The above system is schedulable.
- No static-priority scheme is optimal for scheduling periodic tasks: However, a deadline will be missed, regardless of how we choose to (statically) prioritize $t_1$ and $t_2$.
- Corollary: Neither RM nor DM is optimal

# Optimality Among Static-Priority Algorithms

- Theorem: A system of T independent, preemptable, synchronous periodic tasks that have relative deadlines equal to their respective periods can be feasibly scheduled on one processor according to the RM algorithm whenever it can be feasibly scheduled according to any static priority algorithm.
- Exercise: Complete the proof.
- Note: When $D_i$ <= $T_i$ for all tasks, DM can be shown to be an optimal static-priority algorithm using similar argument. Proof left as an exercise.

# Liu and Layland Bound

- Theorem: [Liu and Layland] A set of n independent, preemptable periodic tasks with relative deadlines equal to their respective periods can be scheduled on a processor according to the RM algorithm if its total utilization U is at most $n(2^{1/n} - 1)$. In other words, $U_{lub}(RM, n) = n(2^{1/n} - 1) >= 0.693$.

| $n$ | $U_{lub}(RM, n)$ | $n$ | $U_{lub}(RM, n)$ |
|---|---|---|---|
| 2 | 0.828 | 3 | 0.779 |
| 4 | 0.756 | 5 | 0.743 |
| 6 | 0.734 | 7 | 0.728 |
| 8 | 0.724 | 9 | 0.720 |
| 10 | 0.717 | $ln2$ | 0.693 |

# Utilization-Based Test for EDF Scheduling

- Theorem: A task set T of independent, preemptable, periodic tasks with relative deadlines equal to their periods can be feasibly scheduled (under EDF) on one processor if and only if its total utilization U is at most one.

# Relative Deadlines Less than Periods

- Theorem: A task set T of independent, preemptable, periodic tasks with relative deadlines equal to or less than their periods can be feasibly scheduled (under EDF) on one processor if:

$$\sum_{k=1}^{n} \frac{C_k}{\min\{D_k, T_k\}} \leq 1.$$

- Note: This theorem only gives sufficient condition.

# Comparison between RM and EDF (Implicit Deadlines)

## RM

- Low run-time overhead: *O(1)* with priority sorting in advance
- Optimal for static-priority
- Schedulability test is NP-hard (even if the relative deadline = period)
- Least upper bound: 0.693
- In general, more preemption

## EDF

- High run-time overhead: *O(log n)* with balanced binary tree
- Optimal for dynamic-priority
- Schedulability test is easy (when the relative deadline = period)
- Least upper bound: 1
- In general, less preemption

# End of Chapter 6