RT Scheduling Anomalies

Claim - Real-time computing is not equivalent to fast computing!

Some questions…

1) Does increase of hardware result in superior performance always?
2) Does use of shortest tasks first lead to minimum makespan?

…

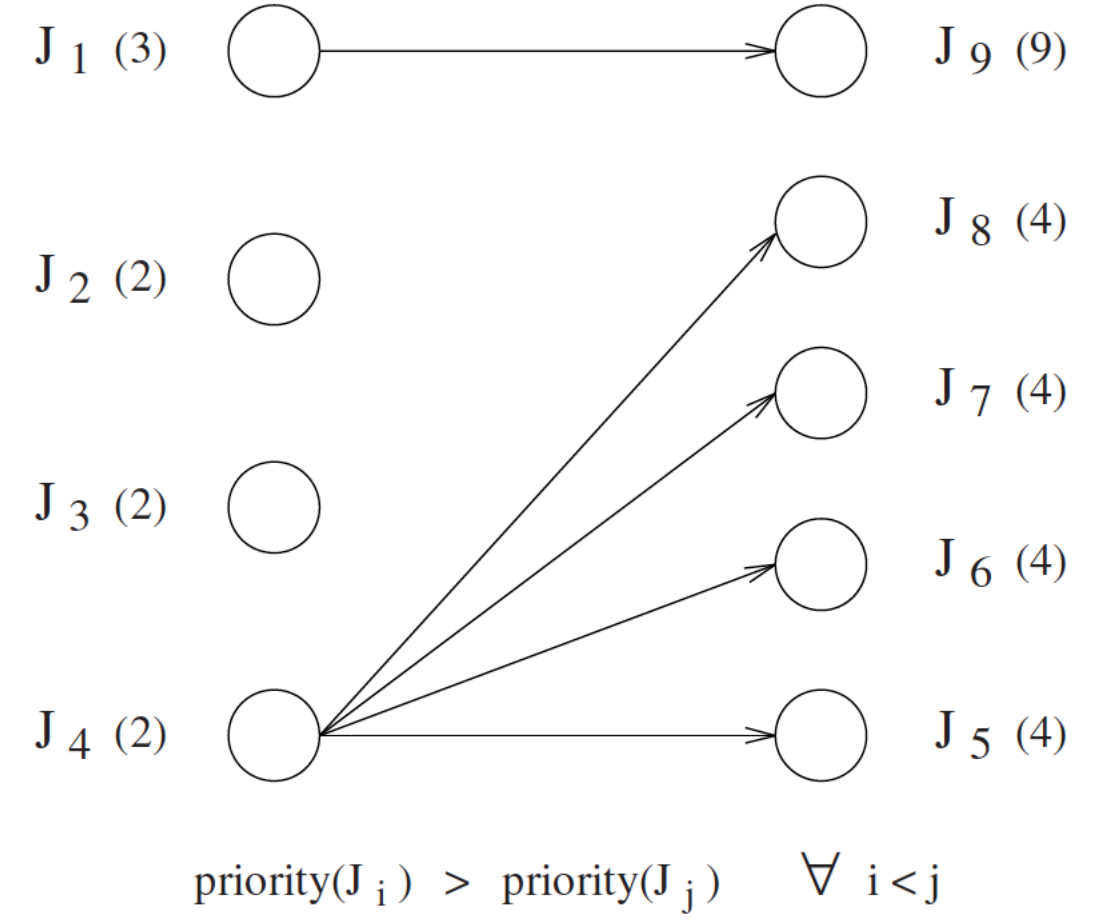
Above situations – *Richard's Anomalies* described by Graham (1976); We will use task sets with precedence relations executed in a multiprocessor environment

RT Scheduling Anomalies

- **Theorem (Graham, 1976)** If a task set is optimally scheduled on a multiprocessor with some priority assignment, a fixed number of processors, fixed execution times, and precedence constraints, then increasing the number of processors, reducing execution times, or weakening the precedence constraints **can/may** increase the schedule length.

- This implies that, if tasks have deadlines, then adding resources (for example, an extra processor) or relaxing constraints (less precedence among tasks or fewer execution times requirements) **can/may** make things worse!

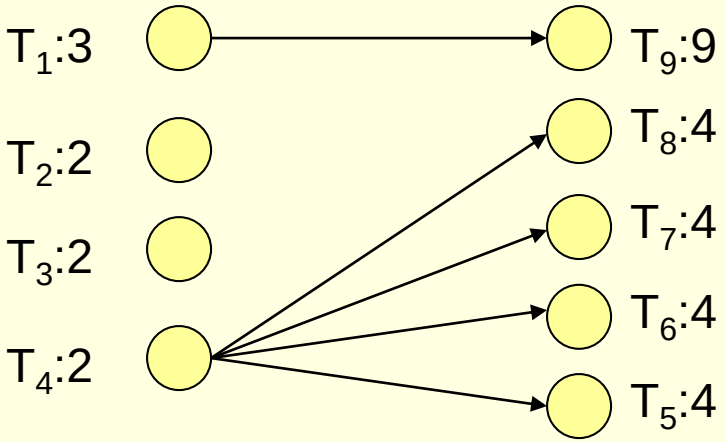
Example – Scheduling Anomalies



Precedence constraint task;

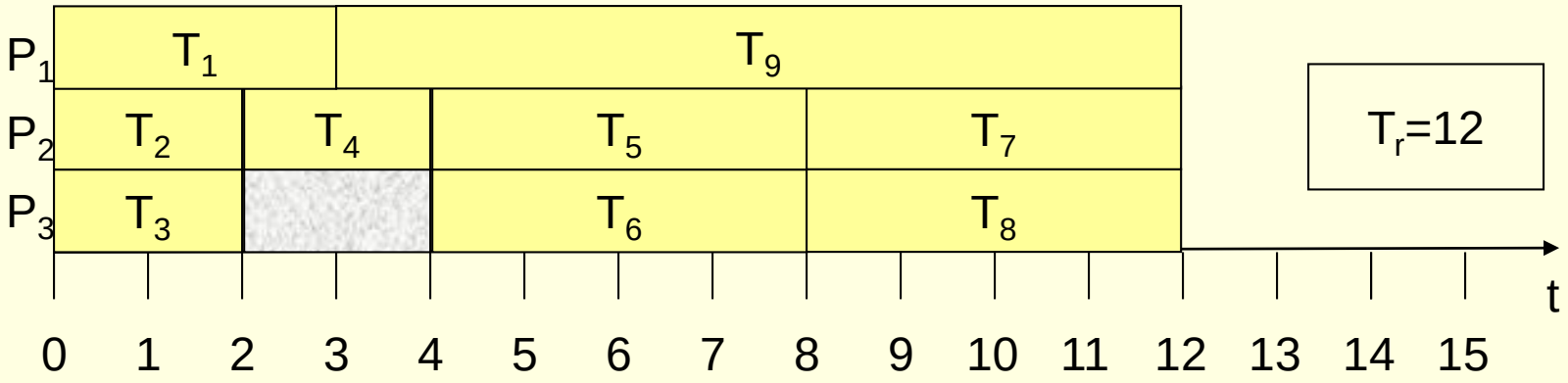
Numbers in the brackets indicate execution times of the tasks;

Case: Anomalies under increased processors

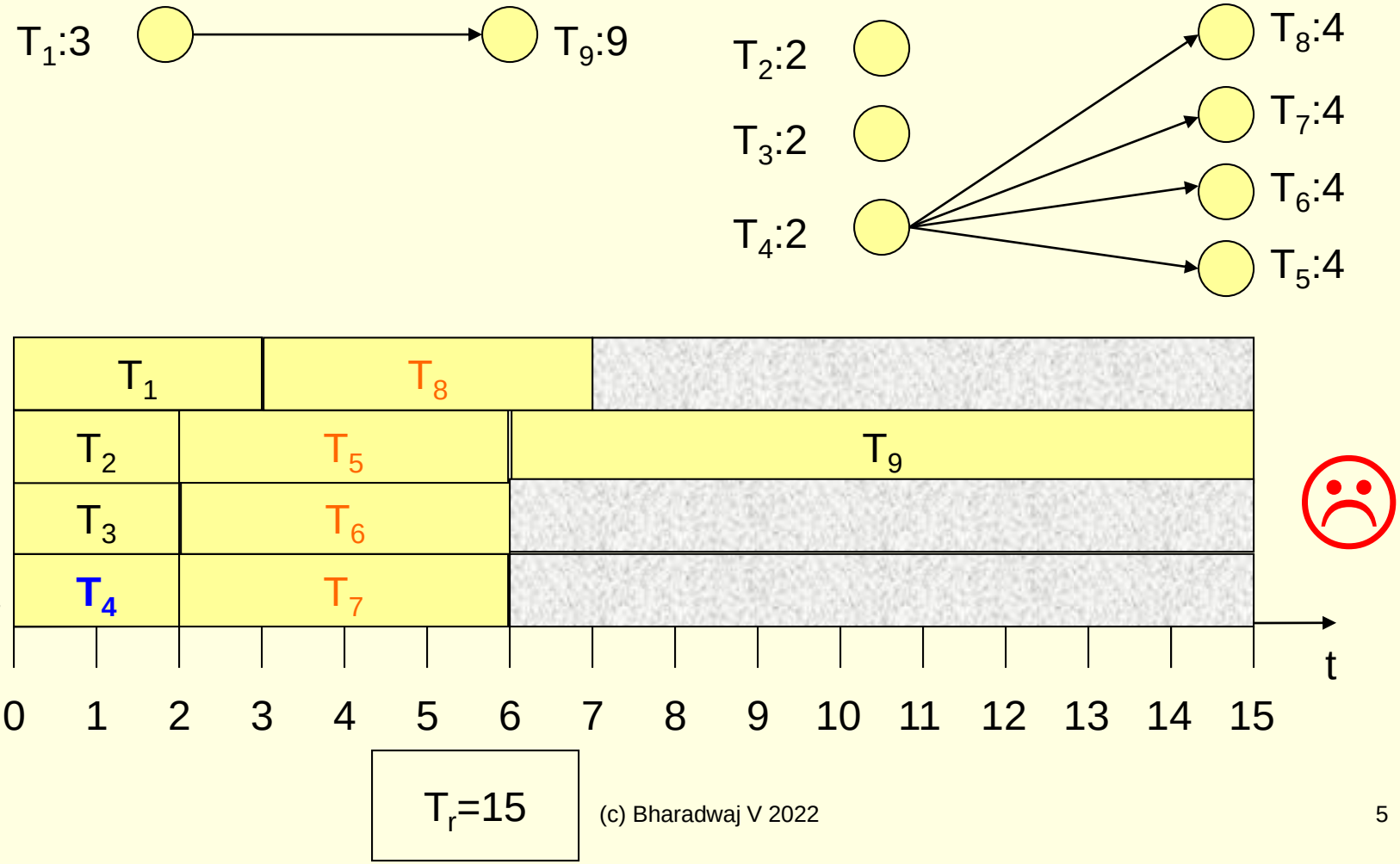


With three processors;

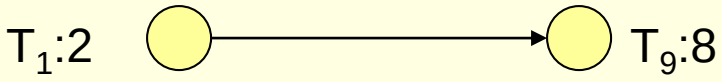
Criteria: Highest priority task is assigned to the first available processor



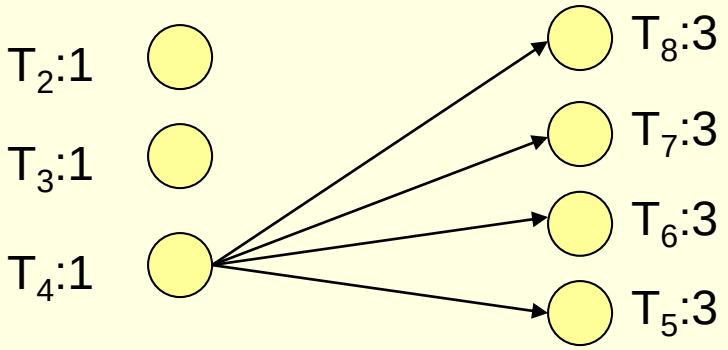
Case: Anomalies under increased processors

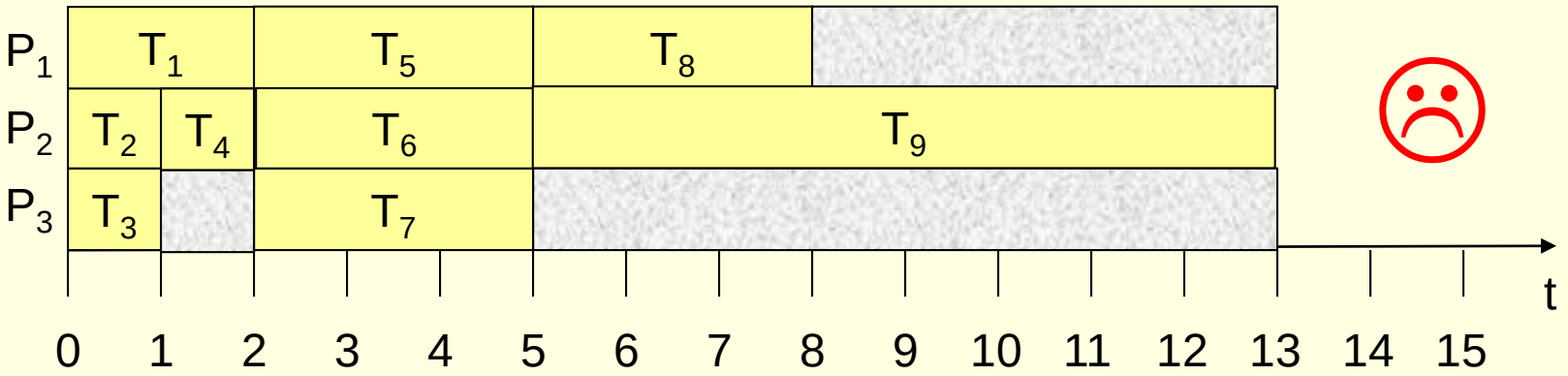


Case: Anomalies when using shorter tasks



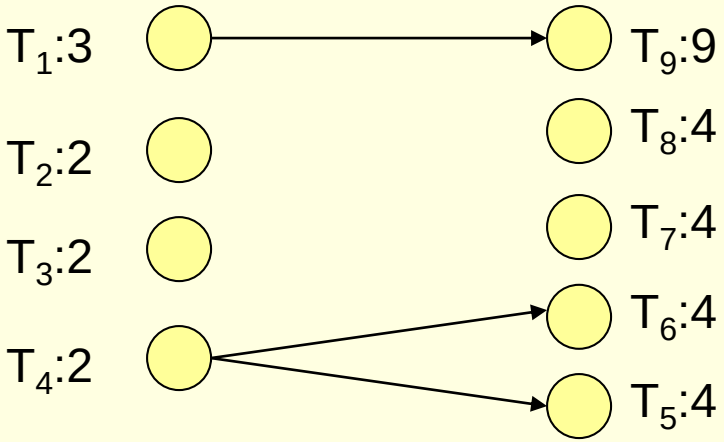
Criteria: Highest priority task is assigned to the first available Processor; Reduce the task comp time by 1 unit for all tasks;





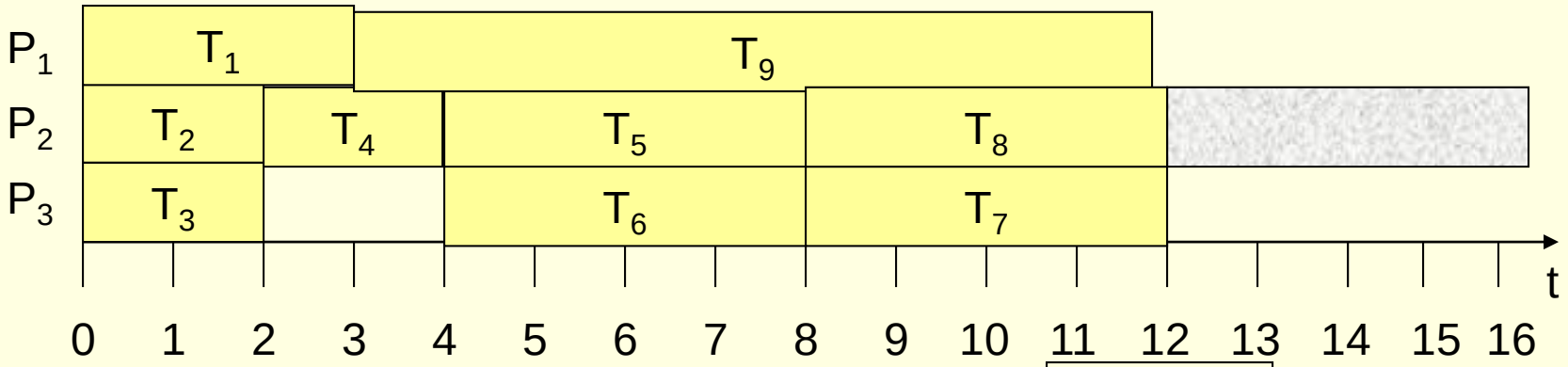
T_r=13

Case: Anomalies under precedence constraints weakened/released



Removing the precedence relations between job J4 and jobs J7 and J8

Strictly following the priority we obtain the following schedule.



T_r=12

Handling overloaded conditions

- **Situation** - Computational demand requested by the task set exceeds the processor capacity, and hence not all tasks can complete within their deadlines;
- *Why this happens?*
  - Bad design of the RTS (Challenge during peak loads)
  - Inability to handle simultaneous events/tasks;
  - Hardware faults (Malfunctioning of i/p devices, resulting in generation of anomalous interrupts);
  - OS exemptions – Too much racing with the input task arrival rates;
  - Other environmental related factors;

Handling overloaded conditions

- Workload definition:

Non-RT applications: As per queuing theory the definition of workload (or also referred to as traffic intensity) is: $\rho = \lambda.\hat{C}$, where λ is the mean arrival rate of tasks, and $\hat{C}$: mean service time of a task;

RT Applications: A system is overloaded when, based on worst-case assumptions, there is no feasible schedule for the current task set, and hence, one or more tasks will miss their deadlines

Handling overloaded conditions

- Preemptable periodic tasks: If the task set consists of **n** independent preemptable periodic tasks, whose relative deadlines are equal to their period, then the system load **ρ** is equivalent to the processor utilization factor U:

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

Where C_i is the execution time and T_i is the period of the task i

- In this case, a load **ρ > 1** means that the total computation time requested by the periodic activities in their hyper-period exceeds the available time on the processor; therefore, the task set cannot be scheduled by any algorithm.

Handling overloaded conditions

- *For a generic set of loads occurring in a dynamic RTS, what is an apt definition of the workload?*
- In such systems, the load varies at each job activation and it is a function of the jobs' deadlines. In general, the load in a given interval [t_a , t_b] can be defined as:

$$\rho(t_a, t_b) = \max_{t_1, t_2 \in [t_a, t_b]} \frac{g(t_1, t_2)}{t_2 - t_1}$$

Not suitable! Why?

g(.,.) is the processor demand

- **Instantaneous load** (suitable for dynamic RTS) **ρ(t):**
  - Practical definition that can be used to estimate the current load in dynamic real-time systems

What is it?

Handling overloaded conditions

- Deriving instantaneous load:

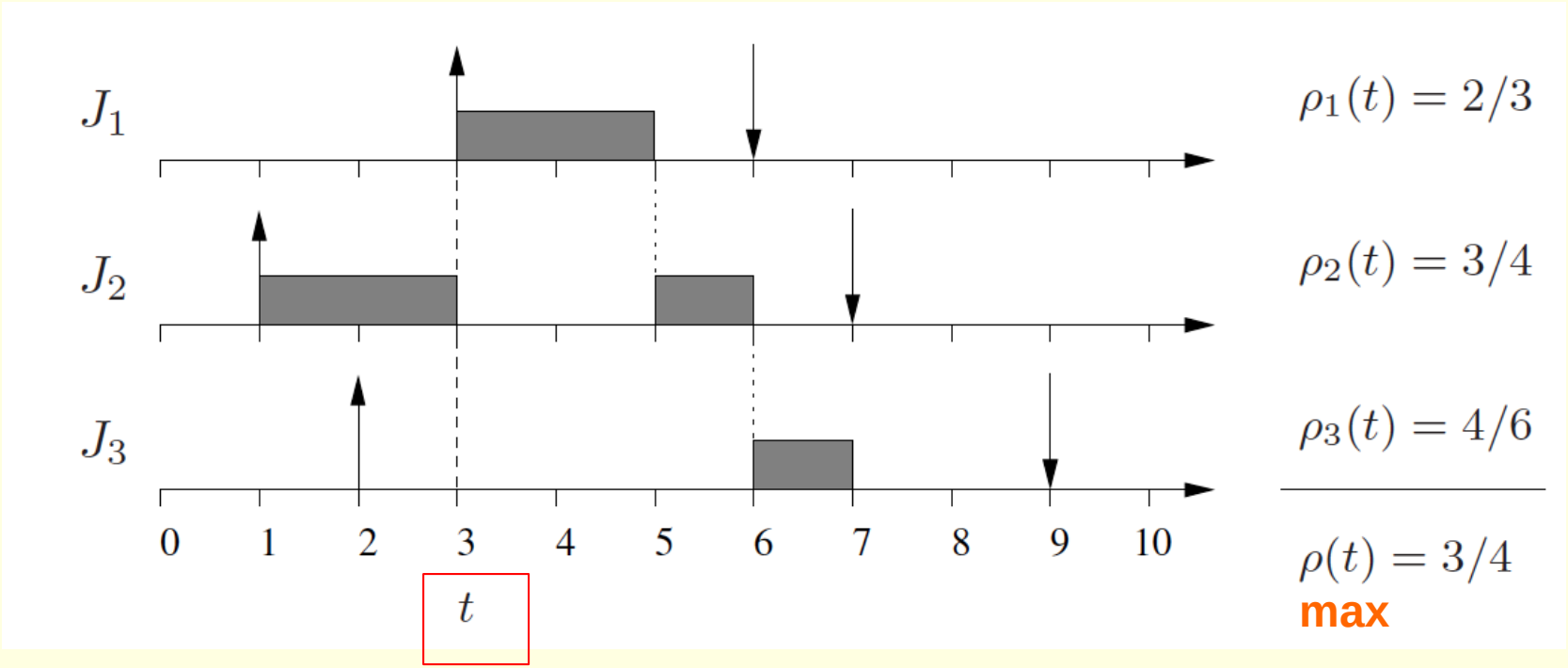
Compute the load in **all** intervals from the current time **t** and each deadline (d_i) of the active jobs. Hence, the intervals that need to be considered for the computation are: $[t, d_1], [t, d_2], \ldots, [t, d_n]$. In each interval $[t, d_i]$, the partial load $\rho_i(t)$ due to the first i jobs is given by:

$$\rho_i(t) = \frac{\sum_{d_k \leq d_i} c_k(t)}{(d_i - t)},$$

where $c_k(t)$ refers to the remaining execution time of job J_k with deadline less than or equal to d_i . Therefore, the total load at time t is $\rho(t) = \max\{\rho_i(t)\}$

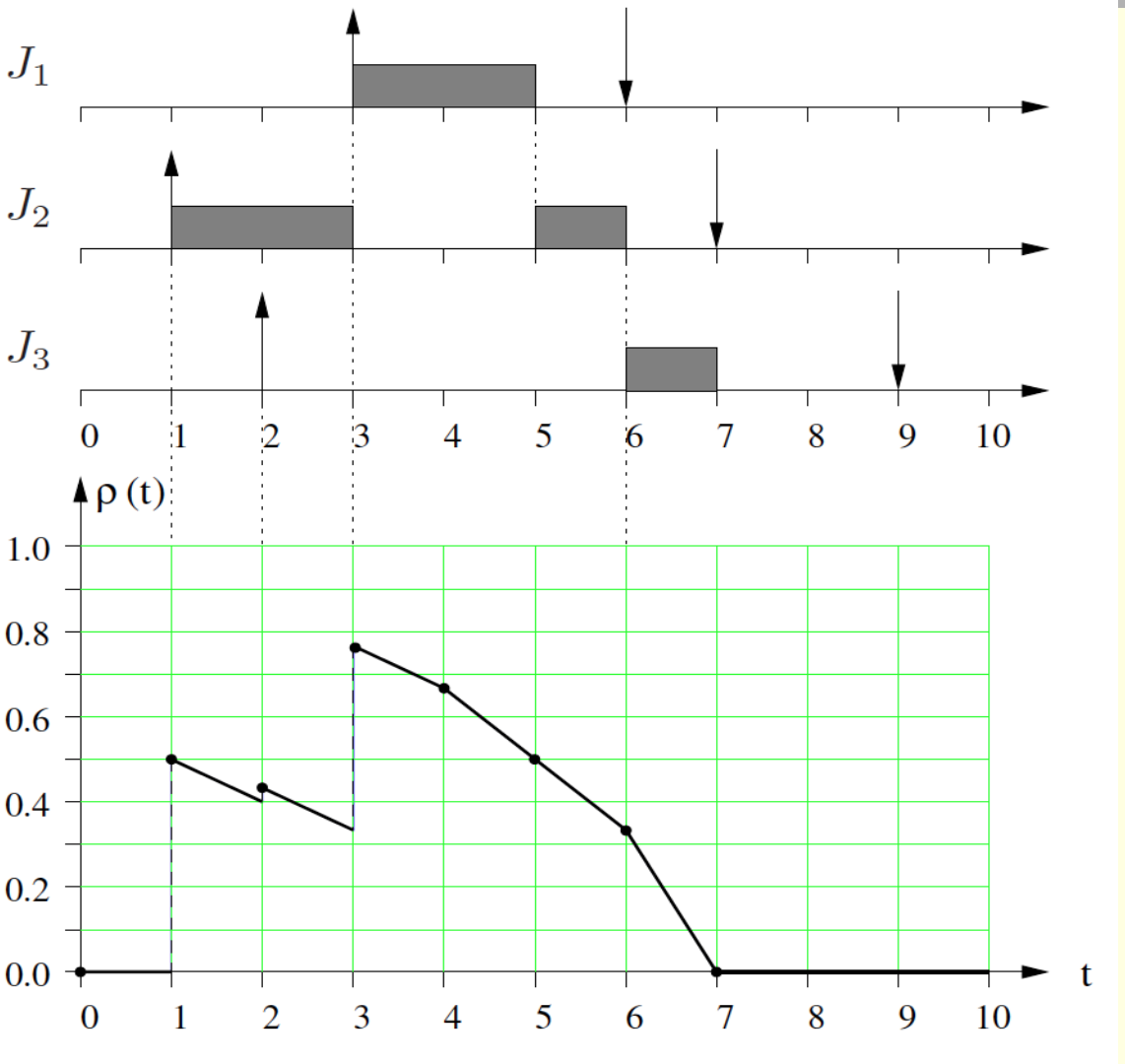
Handling overloaded conditions

Example: For a set of 3 jobs, with their respective deadlines shown, at t=3, the total load is given by:



Handling overloaded conditions

Example (Cont'd):



Handling overloaded conditions

- After obtaining the profile of the total load as shown above, required amount of computational resources can be dispatched, if anomalies can be taken care. This is purely in view of meeting the deadlines.

- **Definition**: A computing system is said to experience an **overload** when the computation time demanded by the task set in a certain interval of time exceeds the available processing time in the same interval.
- **Definition**: A task (or a job) is said to experience an **overrun** when exceeding its expected utilization. An overrun may occur either because the next job is activated before its expected arrival time (activation overrun), or because the job computation time exceeds its expected value (execution overrun).