Real-time Systems

Chapter 4:
Basic concepts of scheduling

Ngo Lam Trung

Dept. of Computer Engineering

Contents

- Introduction of scheduling
- Constraints of real-time tasks
- Classification of scheduling algorithms
- Scheduling anomalies

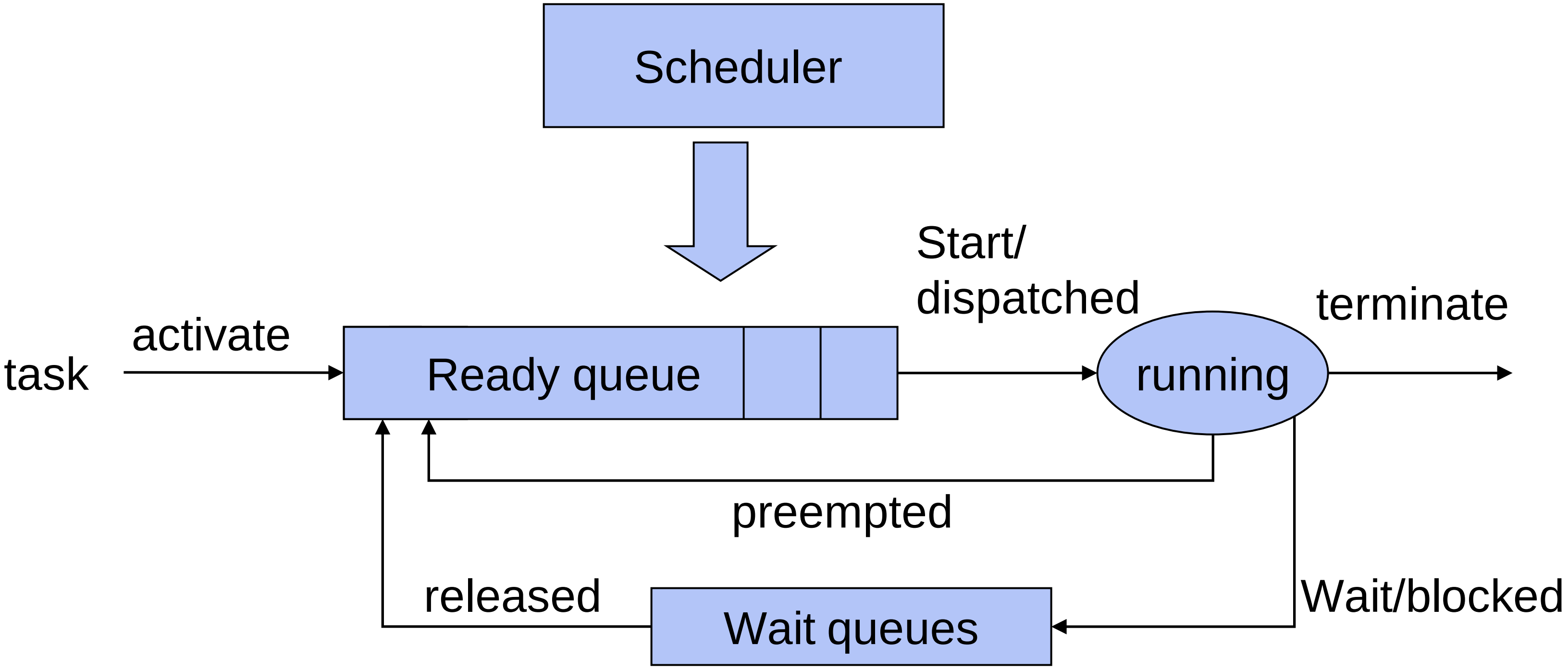
Introduction

- Why do we need scheduling?
  - There are always more tasks than processors.
  - Multiple tasks run concurrently on uniprocessor system.
- Scheduling policy: the criterion to assign the CPU time to concurrent tasks
- Scheduling algorithm: the set of rules that determines the order in which tasks are executed

→ *What is the main difference between scheduling in RTOS and GPOS?*

An illustration of scheduling

- All activated tasks enters “ready queue” at first.
- The scheduler selects one task in the Ready queue according to the tasks’ priorities allocated based on the scheduling algorithm.
- The selected task is dispatched and becomes in “running” state.
- After the selected task is completed, it is removed from the Ready queue.



Preemption

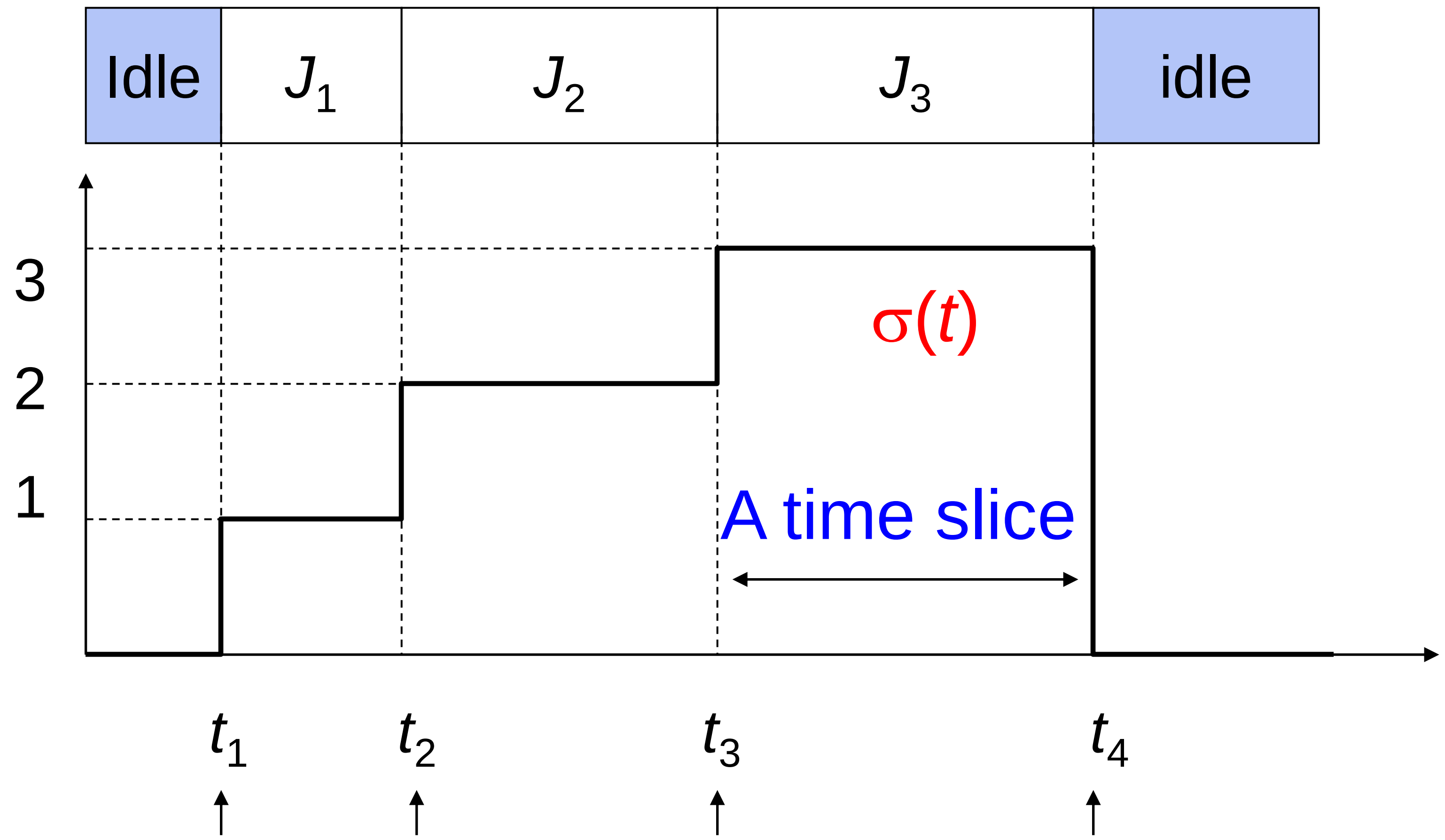
- The running task can be interrupted at any point, so that a more important task that arrives can immediately gain the processor.
- The to-be-preempted task is interrupted and inserted to the ready queue, while CPU is assigned to the most important ready task which just arrived.
- Why preemption is needed in real-time systems?
  - Exception handling of a task
  - Treating with different criticalities of tasks, permits to anticipate the execution of the most critical activities
  - Efficient scheduling to improve system responsiveness

Notation of scheduling (1)

- $J = \{J_1,\ldots,J_n\}$ A set of tasks
- $\sigma:\mathbf{R}^+\rightarrow\mathbf{N}$ A schedule
  - A function mapping from time to task to assign task to CPU
  - If $\sigma(t)=i$ for $\forall t\in[t_1,t_2)$, task J_i is executed during time duration $[t_1,t_2)$.
  - If $\sigma(t)=0$, the CPU is *idle*.

- Simple translation
  - CPU time is divided in to time slices $[t_1,t_2)$
  - During a time slice $\sigma(t)$=const, representing the task that is executed

Notation of scheduling (2)

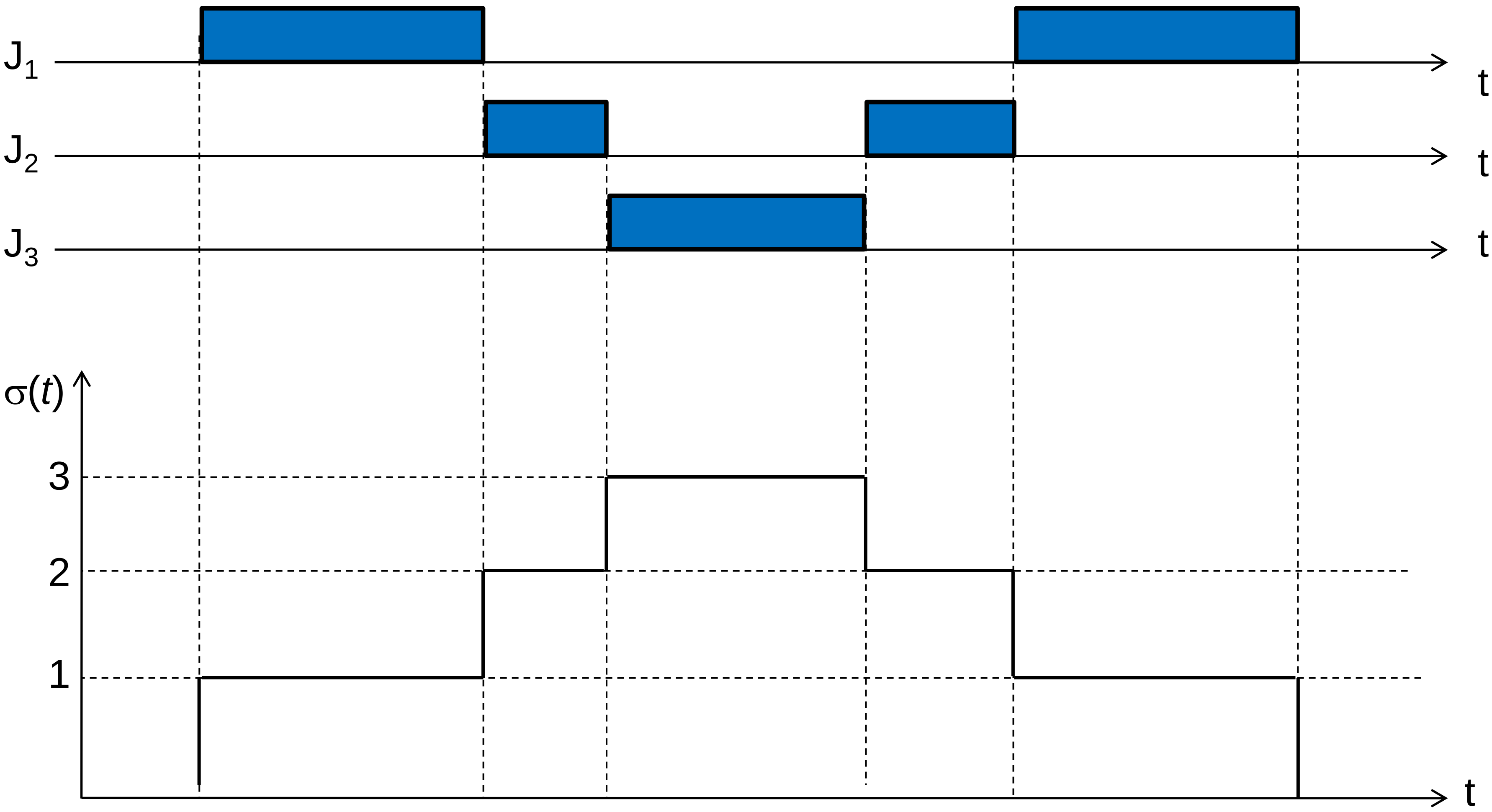


Context switching are performed at these times

Notation of scheduling (3)

- Preemptive schedule
  - A schedule in which the running task can be arbitrarily suspended at any time, to assign the CPU to another task
- Feasible schedule
  - A schedule that all tasks can be completed according to a set of specified constraints
- Schedulable set of tasks
  - A set of tasks that has at least one feasible schedule by some scheduling algorithm

Example of preemptive schedule



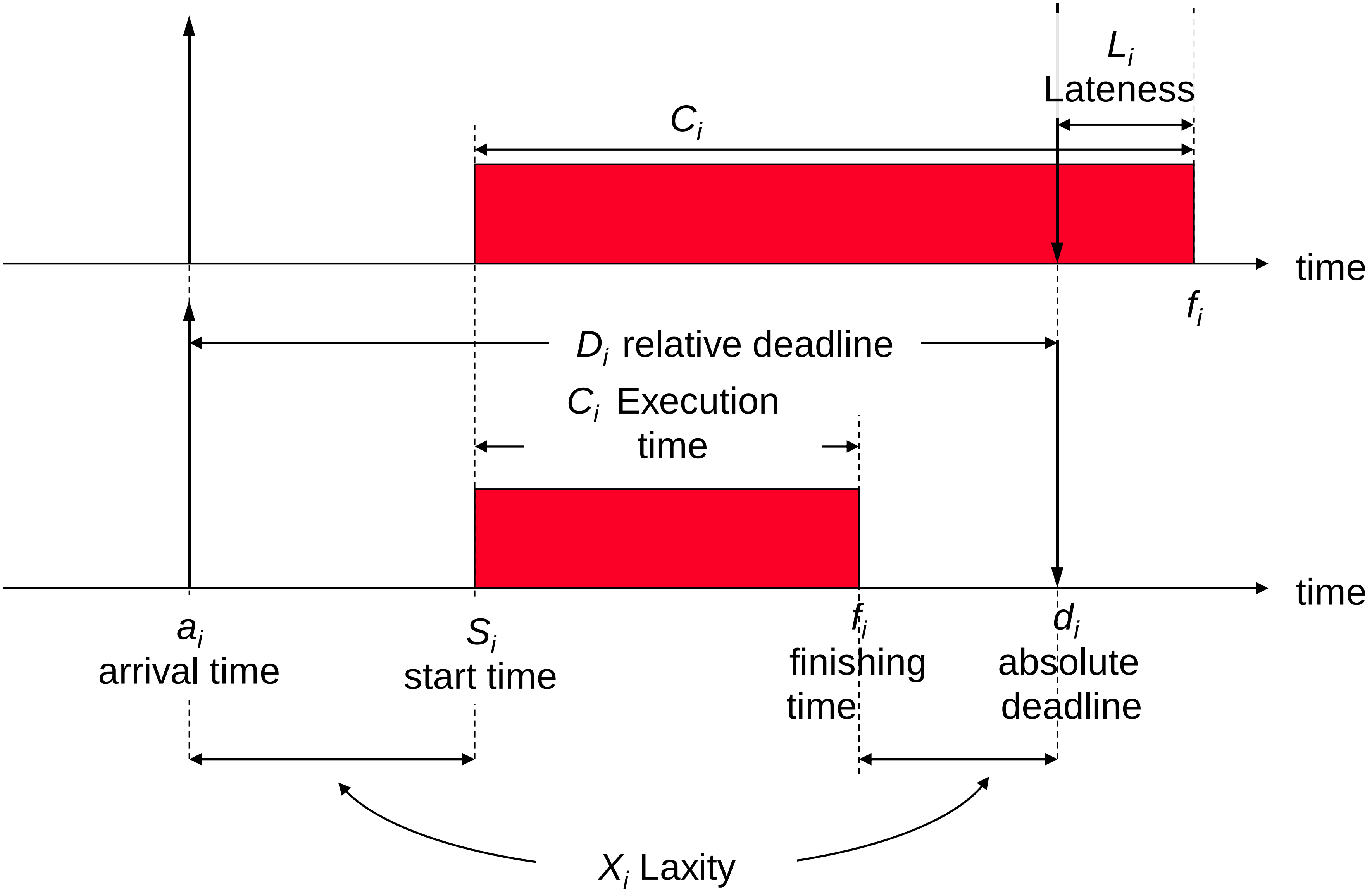
Types of task constraints

- Timing constraints
- Precedence constraints
- Resource constraints

Timing constraints

- Timing constraints:
  - Constraints on execution time, is the time that must be meet in order to achieve the desired behavior.
  - Typical constraint: deadline
    - Relative deadline: deadline is specified with respect to the arrival time
    - Absolute deadline: deadline is specified with respect to time zero.
- Classification of real-time tasks
  - Hard : completion after deadline can cause catastrophic consequence
  - Soft : missing deadline decreases the performance of the system but does not jeopardize its correct behavior

Task parameters(1)



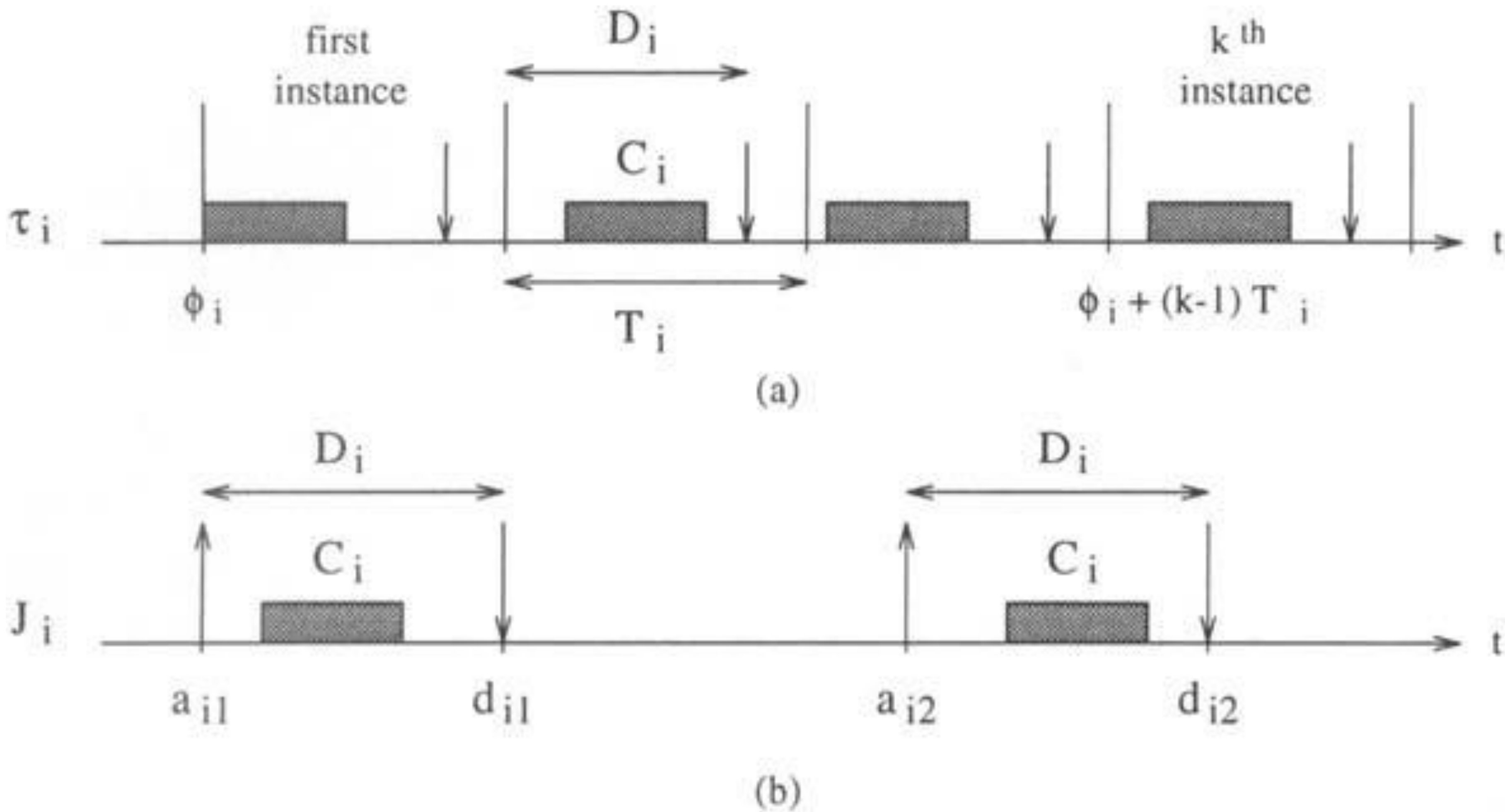
Task parameters(2)

- Other task parameters
  - Response time: different between the finishing time and the request time $R_i = f_i - r_i$
  - Criticality: Hard or Soft
  - Value v_i: relative importance of task with respect to the other tasks
  - Lateness: the delay of a task completion with respect to its deadline $L_i = f_i - d_i$
  - Tardiness or *Exceeding time*: $E_i = max(0, L_i)$ is the time a task stays active after its deadline.
  - Laxity or *Slack time* $X_i = d_i - a_i - C_i$ is the maximum time a task can be delayed on its activation to complete within its deadline

Task parameters(3)

- Regularity of task activation
  - Periodic tasks: infinite sequence of identical activities (jobs) **activated at constant rate**
  - Aperiodic: infinite sequence of identical activities (jobs) with **irregular activation**
  - Sporadic: aperiodic tasks where consecutive activation are separated by some minimum inter-arrival time

Periodic task and aperiodic task

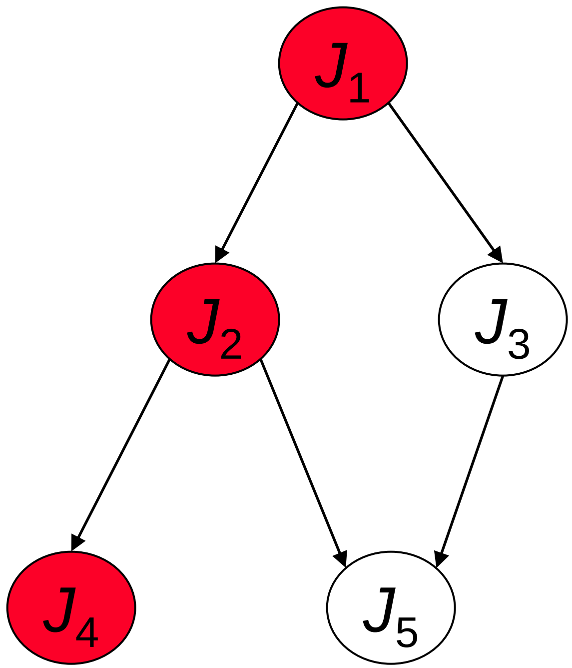


Precedence constraints

- Tasks can have precedence constraint:
  - A task has to be executed after another task is completed
- Notation:
  - Precedence relations are described by a directed acyclic graph *G*.
  - $J_a \prec J_b$: task J_a is a predecessor of task J_b
  - $J_a \rightarrow J_b$: task J_a is an immediate predecessor of J_b

An example of precedence relations

To be executed earlier



To be executed later

$$J_1 \prec J_2$$
$$J_1 \rightarrow J_2$$
$$J_1 \prec J_4$$
$$J_1 \not\rightarrow J_4$$

Resource constraints

- Resource
  - Any software structure that can be used by the process to advance its execution
  - Ex: data structure, a set of variables, main memory area, a file, a piece of program, a set of registers of a peripheral device
- Private resource:
  - A resource dedicated to a particular process
- Shared resource:
  - A resource that can be used by more tasks

Shared resource & critical section

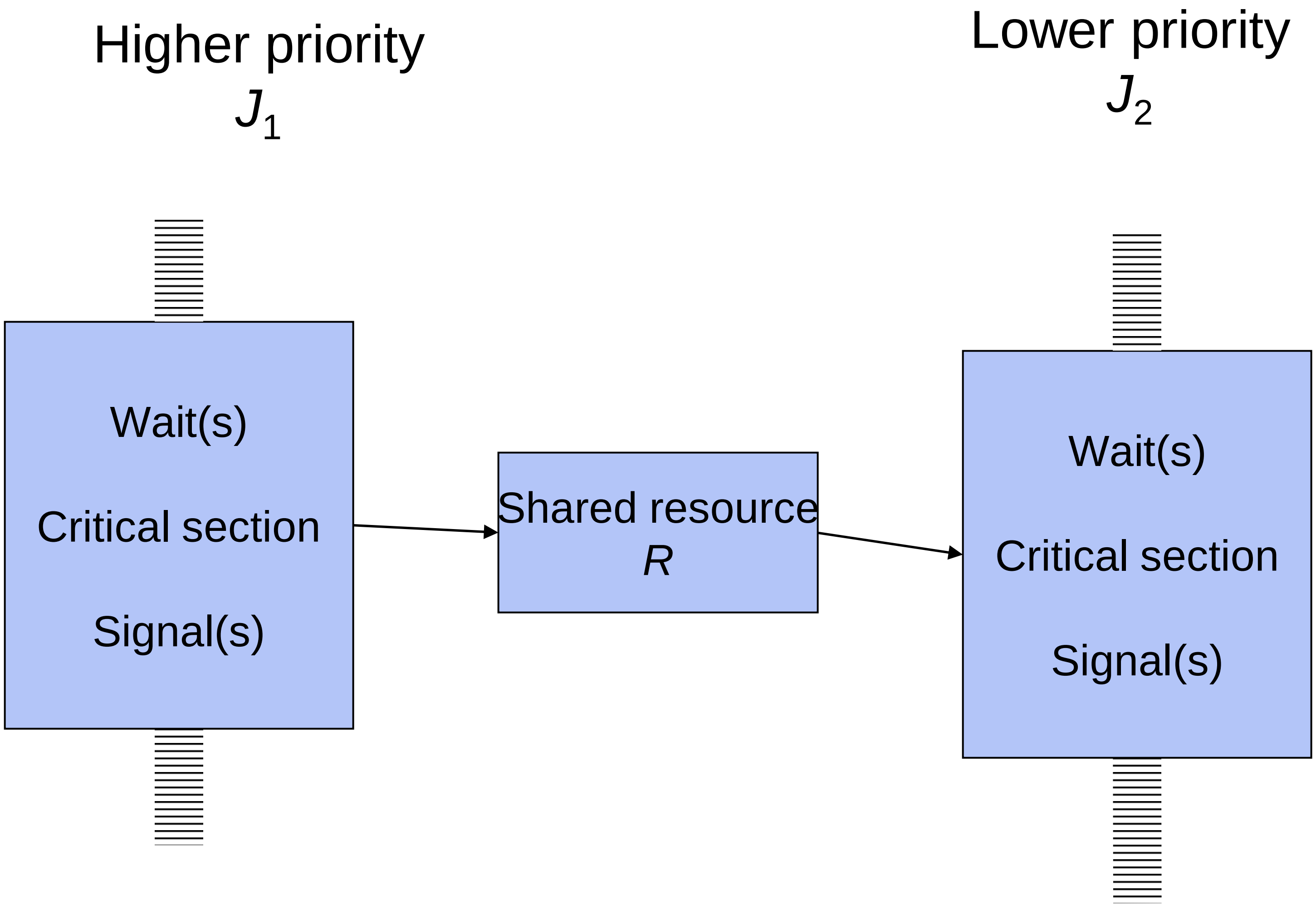
- Many shared resources do not allow simultaneous access

  → require mutual exclusion

- Critical section
  - A piece of code under mutual exclusion constraints
  - Created by synchronization mechanism

- When tasks have resource constrains, they have to be synchronized

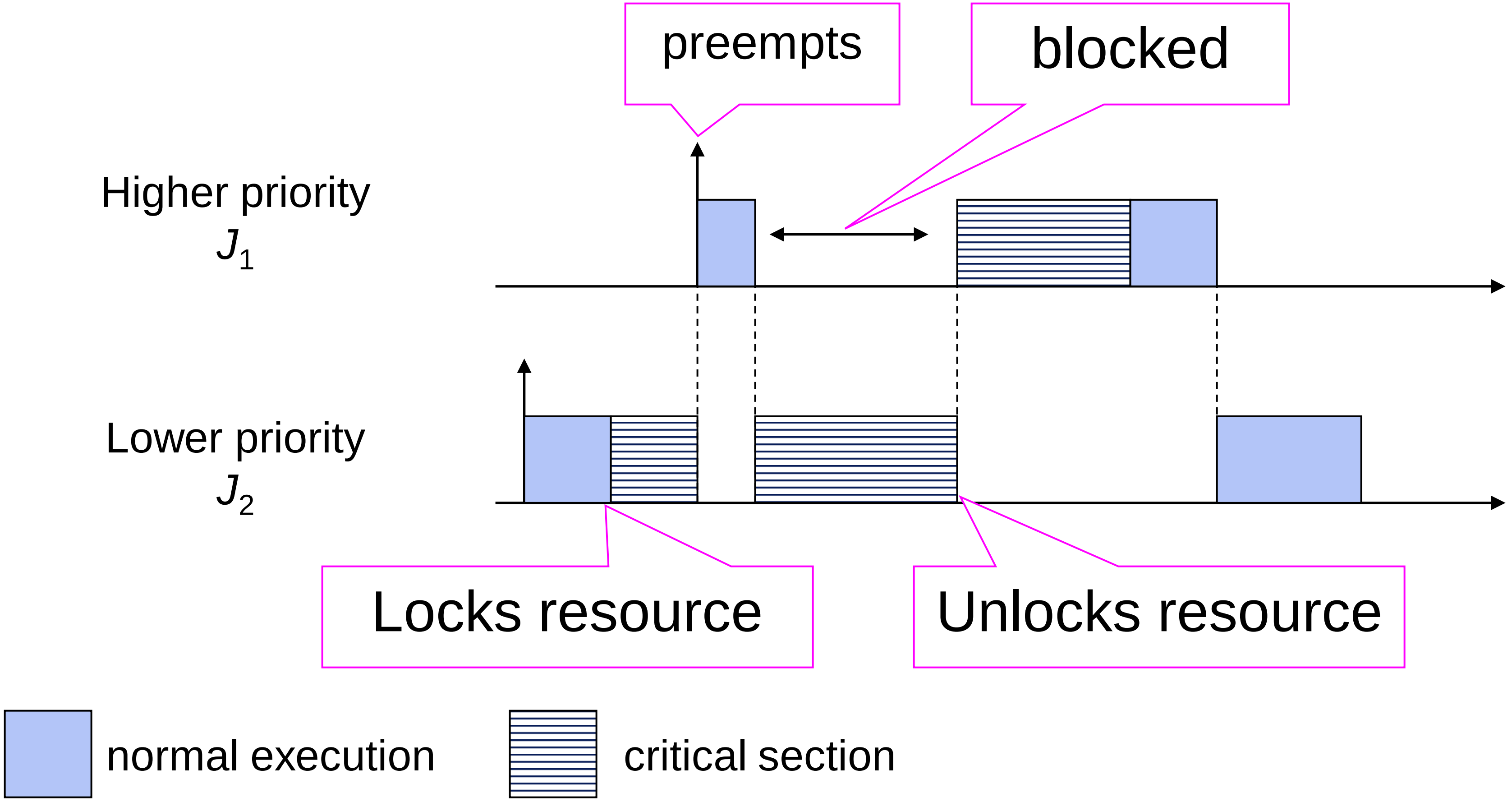
An example of mutual exclusion



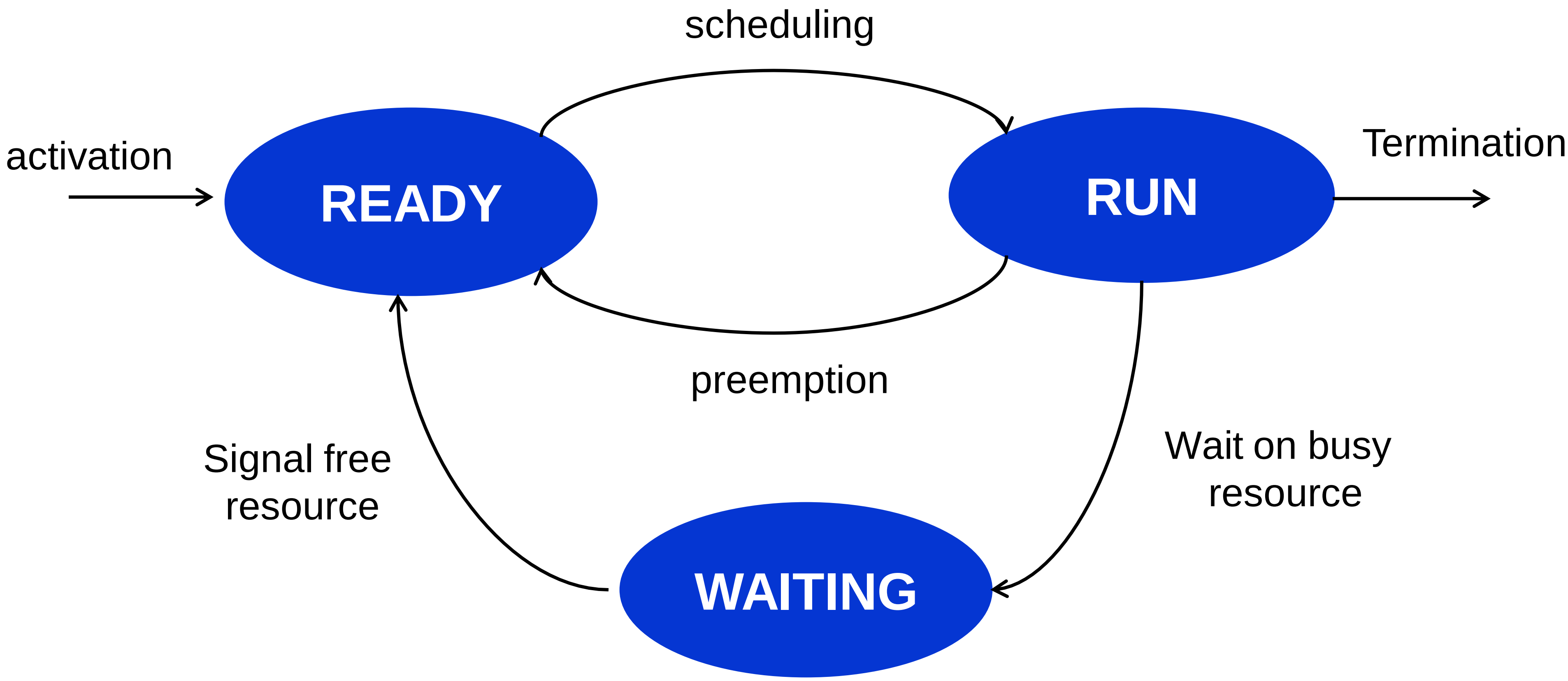
Critical section is created by using the binary semaphore s

An example of mutual exclusion

- Scheduling with preemption



Waiting state caused by resource constraint



Definition of scheduling problems

- *Given*
  - $J = \{J_1,\ldots,J_n\}$: A set of tasks
  - $P = \{P_1,\ldots,P_m\}$: A set of processors
  - $R = \{R_1,\ldots,R_r\}$: A set of resources
  - With precedence constraints and timing constraints
- Scheduling problem:
  - Assigning processors from *P* and resource from *R* to tasks from *J* under given constraints

Complexity of scheduling algorithm

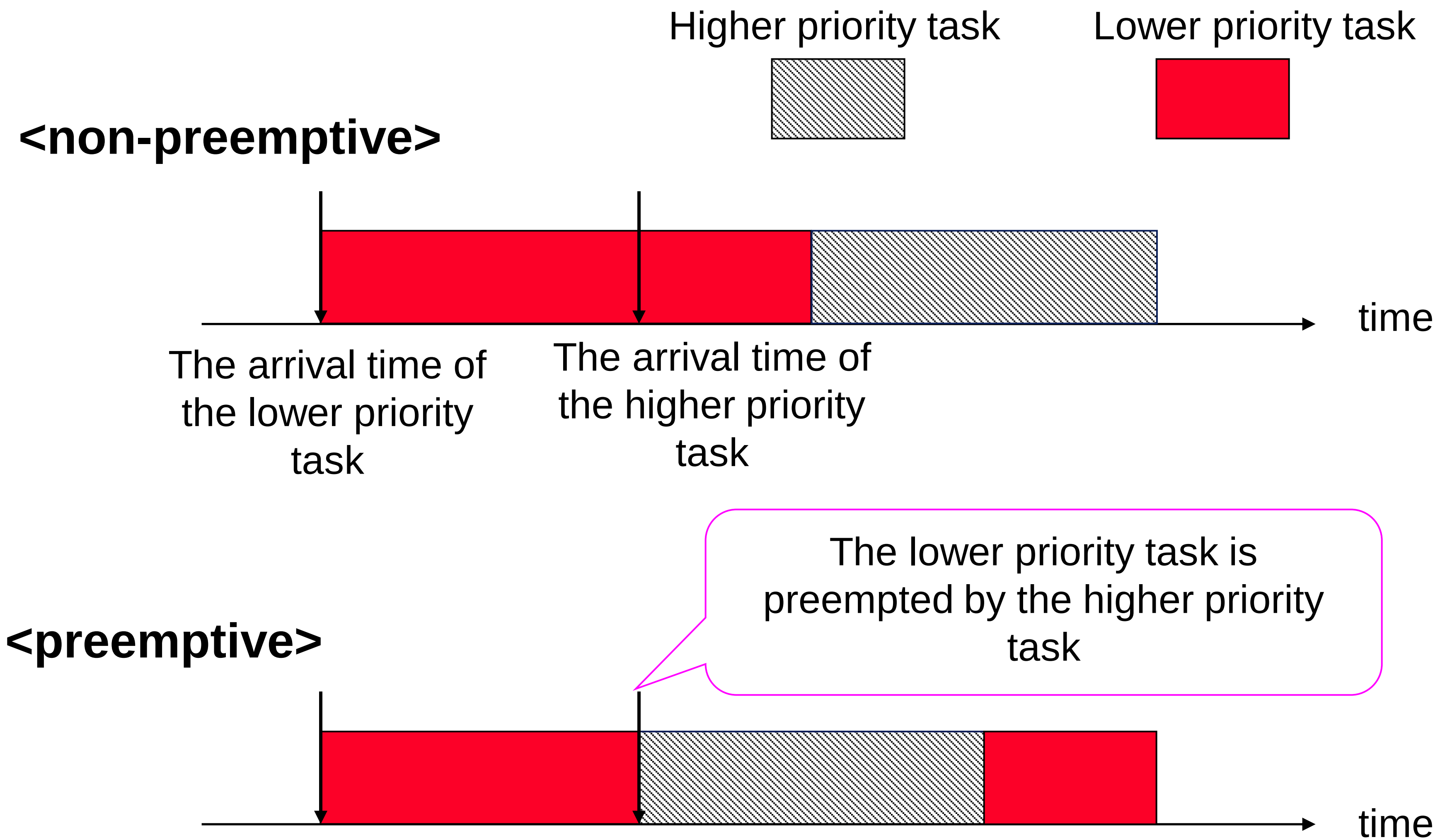
- Complexity of scheduling decision problem
  - NP-complete in general
  - Has strong influence on the performance of dynamic real-time systems
- Practical approach:
  - Simplify computer architecture: uniprocessor
  - Adopt additional conditions: preemptive model, priority, task activation…

  ➔ Result in different classes of problem that can be solved by different scheduling algorithms

Classification of scheduling algorithms (1)

- Preemptive
  - The running task can be interrupted at any time.
- Non-preemptive
  - A task, once started, is executed by the processor until completion without interruption by any other tasks.

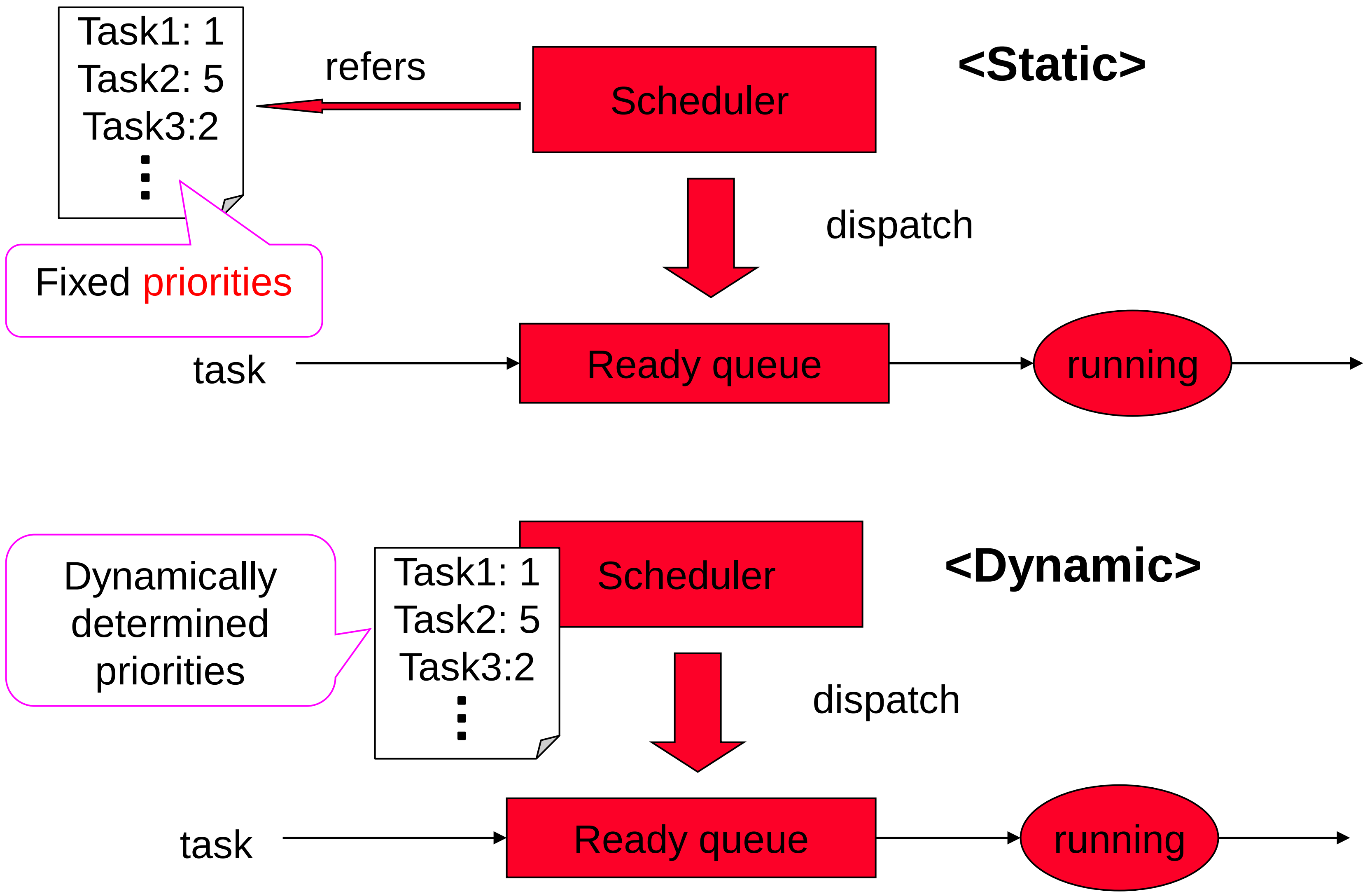
Preemptive & non-preemptive algorithms



Classification of scheduling algorithms (1)

- Static
  - Scheduling decisions are based on fixed parameters, assigned to tasks before their activations.
- Dynamic
  - Scheduling decisions are based on dynamic parameters that may change during system evolution.

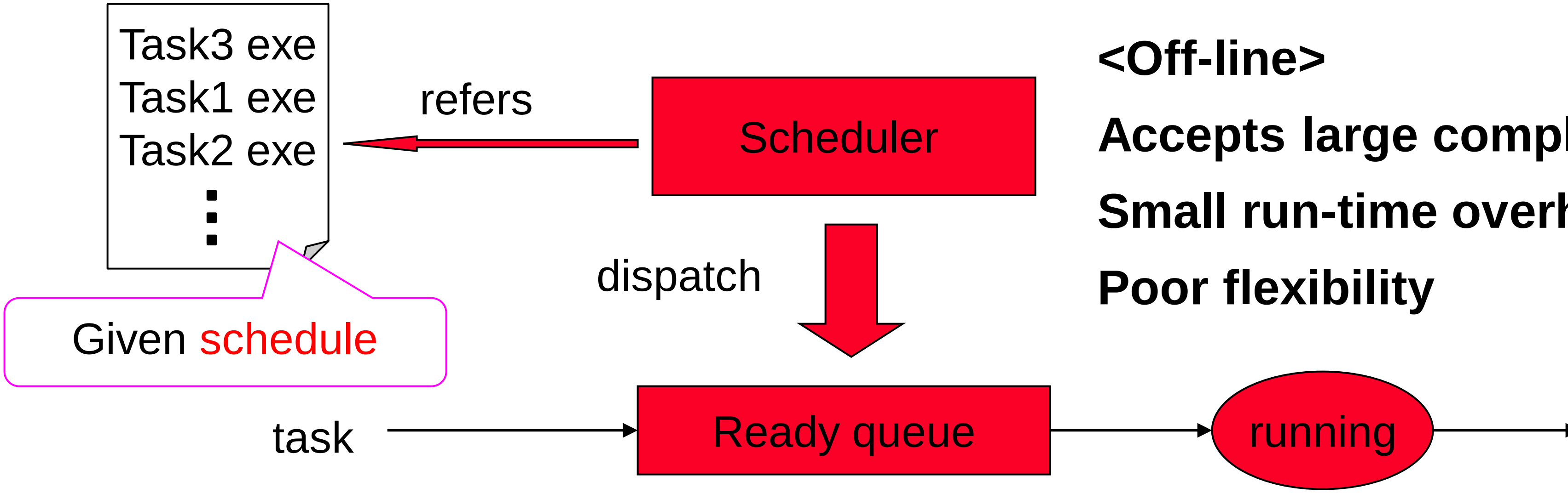
Static & dynamic algorithms



Classification of scheduling algorithms (2)

- Off-line
  - Scheduling decision is executed on the entire task set before actual task execution.
- On-line
  - Scheduling decisions are taken at runtime every time a new task enters the system or when a running task terminates.

Off-line & on-line algorithms

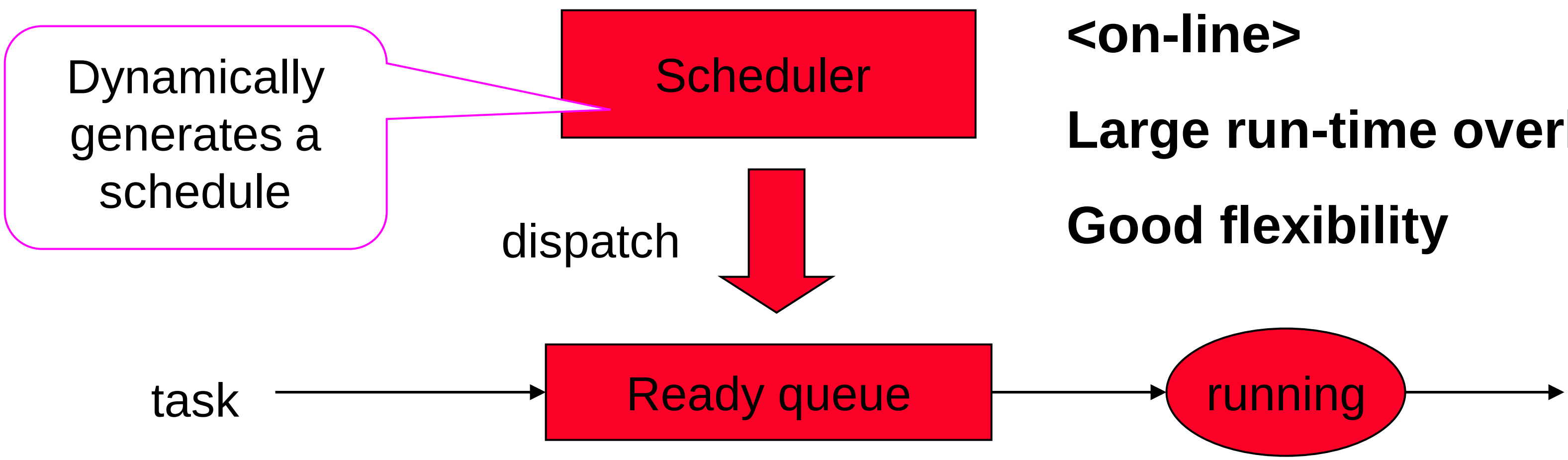


<Off-line>

Accepts large complexity

Small run-time overhead

Poor flexibility



<on-line>

Large run-time overhead

Good flexibility

Classification of scheduling algorithms (3)

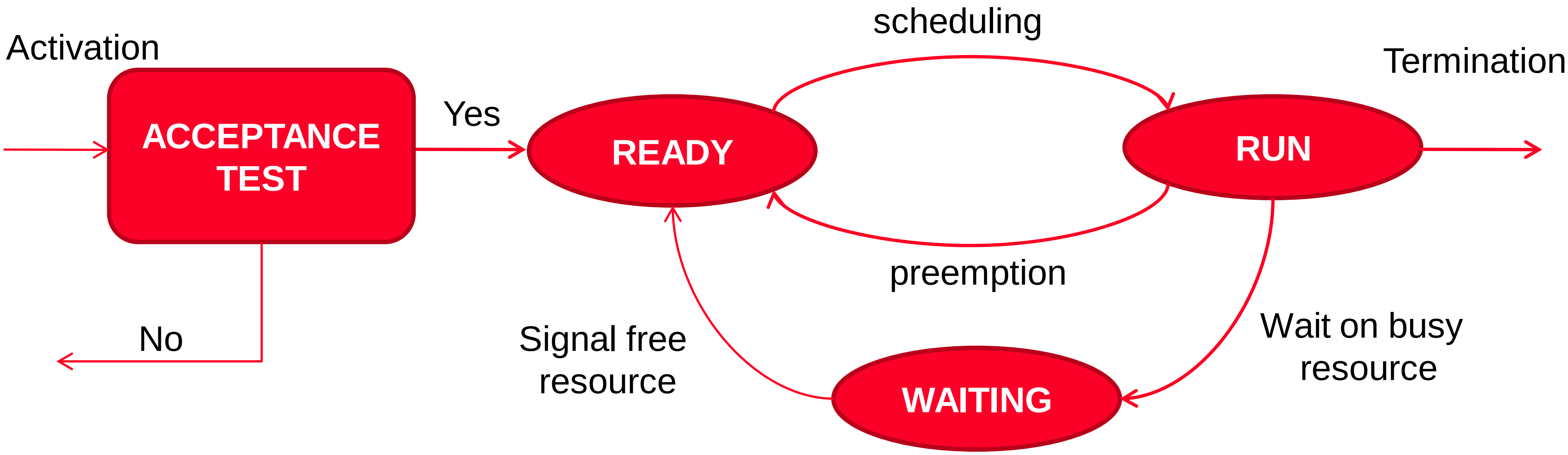
- Optimal
  - (1) Minimizes some given cost function defined over the task set.
  - (2) If no cost function is given, optimal algorithm only fail to meet deadline only if no other algorithm can meet the deadline.
- Heuristic
  - Searches for a feasible schedule using an objective function (heuristic function).
  - Does not guarantee to find the optimal schedule.

Guarantee-based algorithms(1)

- In hard real-time systems, feasibility of the schedule should be guaranteed before task execution.

- In order to guarantee feasibility:
  - Static real-time systems:
    - Off-line scheduling is used. Requires high predictability, and the system becomes inflexible.
  - Dynamic real-time systems:
    - On-line scheduling with acceptance test

Dynamic real-time schedule



Guarantee-based algorithms(2)

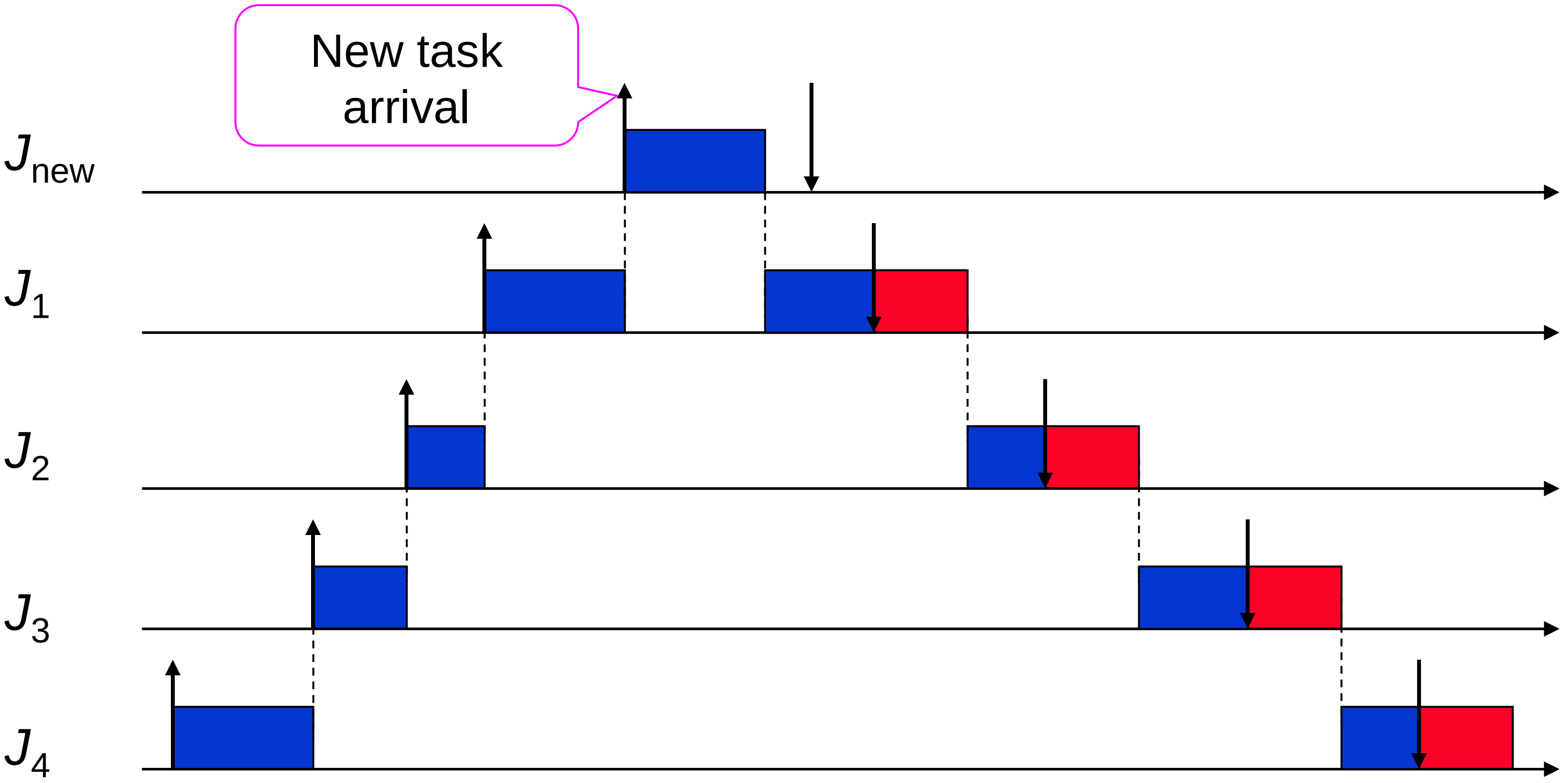
- Acceptance test in on-line scheduling
  - For every arrival of new task J_{new}, it is checked whether $J' = J \cup \{J_{new}\}$ is schedulable or not.
  - If J' is schedulabe, J_{new} is accepted.
  - Otherwise, J_{new} is rejected.
  - Generally based on worst-case assumption

- Demerit: unnecessary rejection

- Merit: detecting potential overload situations
  - Can avoid domino effects

Domino effect

- A dangerous phenomena under transient overload
- The arrival of a new task causes all previously guaranteed tasks to miss their deadlines.



Best-effort algorithms

- In soft real-time systems, some deadline misses can be acceptable.
  - Ex. Multimedia applications

- Best-effort algorithms
  - Accepts all tasks that arrived
  - Aborts some tasks under real overload conditions
  - Cannot guarantee feasibility

- Demerit: domino effect

- Merit: good average performance & avoiding unnecessary rejection

Metrics for performance evaluation(1)

- Performance of scheduling algorithms is evaluated by a cost function.
- An optimal scheduling algorithm generates a schedule that minimizes a given cost function.
- Examples of cost functions
  - Average response time
  - Total completion time
  - Weighted sum of completion times
  - Maximum lateness
  - Maximum number of late tasks
  - Utility functions

Cost functions table

Average response time

$$\bar{R} = \frac{1}{n}\sum_{i=1}^{n}(f_i - a_i)$$

Total completion time:

$$t_c = max_i(f_i) - min_i(a_i)$$

Weighted sum of completion times:

$$t_w = \sum_{i=1}^{n} w_i f_i$$

Maximum lateness

$$L_{max} = max_i(f_i - d_i)$$

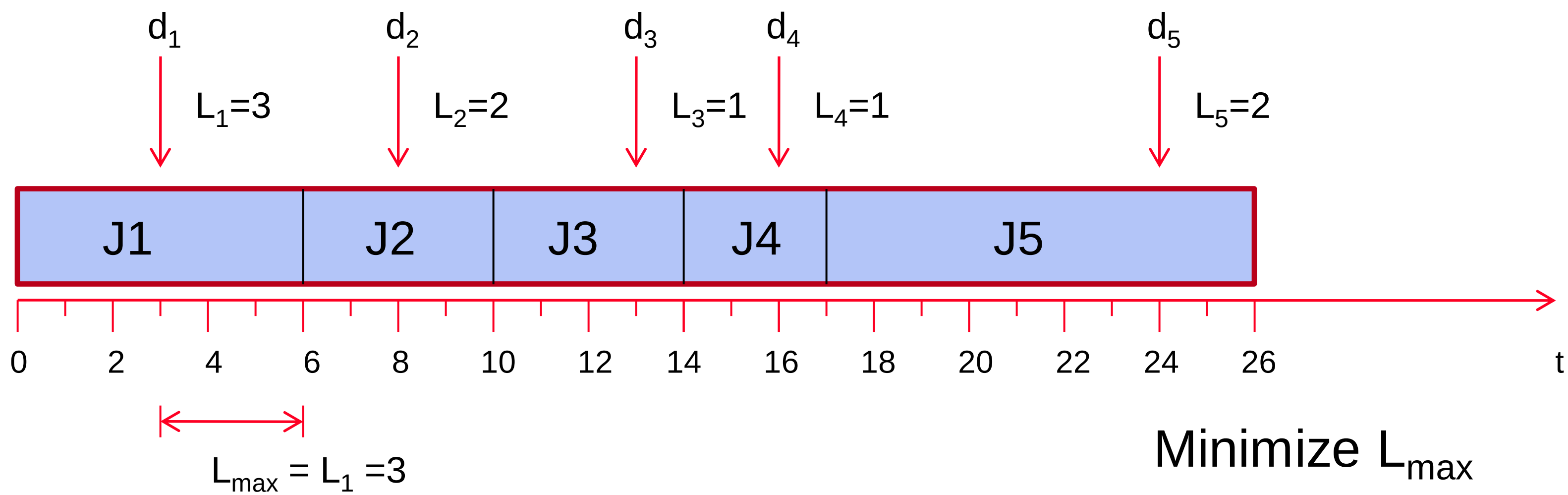
Maximum number of late task

$$N_{late} = \sum_{i=1}^{n} miss\,(f_i)$$

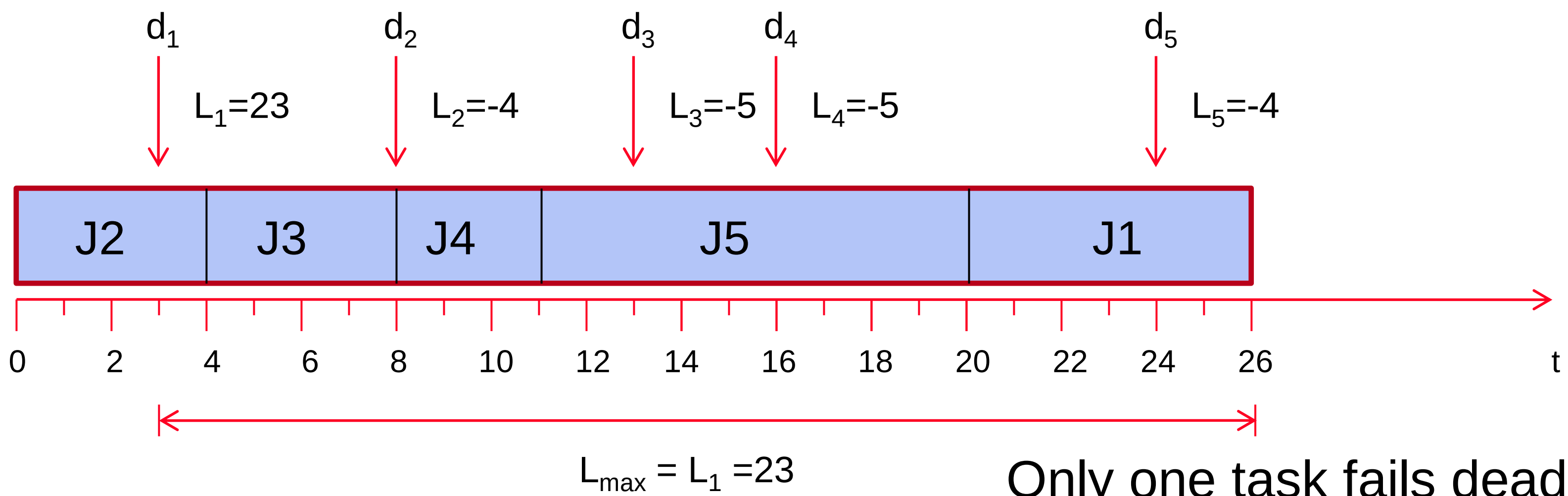
Where

$$miss\,(f_i) = \begin{cases} 0 & if\ f_i < d_i \\ 1 & otherwise \end{cases}$$

Example



Minimize L_{max}

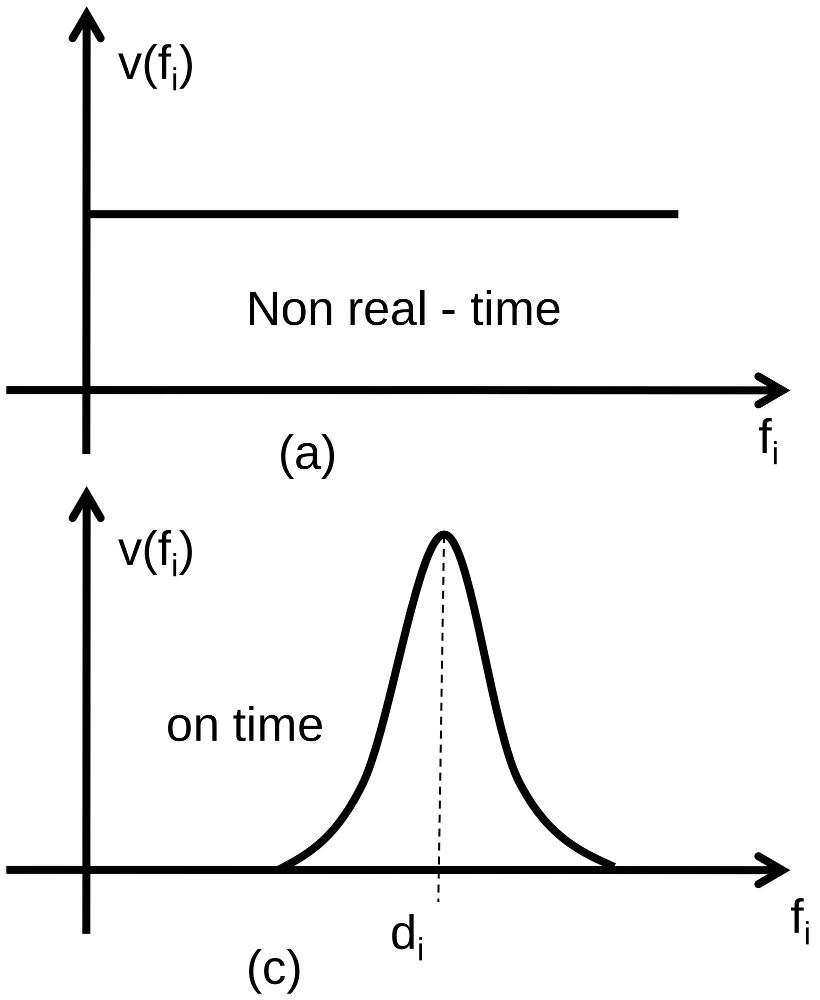


Only one task fails deadline

Utility function

- A kind of cost function with respect to the completion time of a task
- Evaluates lateness in a real-time task that depends on the completion time
- Typical utility functions
  - Non real-time
  - Soft
  - On-time
  - Firm
- To evaluate whole schedule, the cumulative value of utility function values of all tasks is used.

Examples of cost functions



Scheduling anomalies

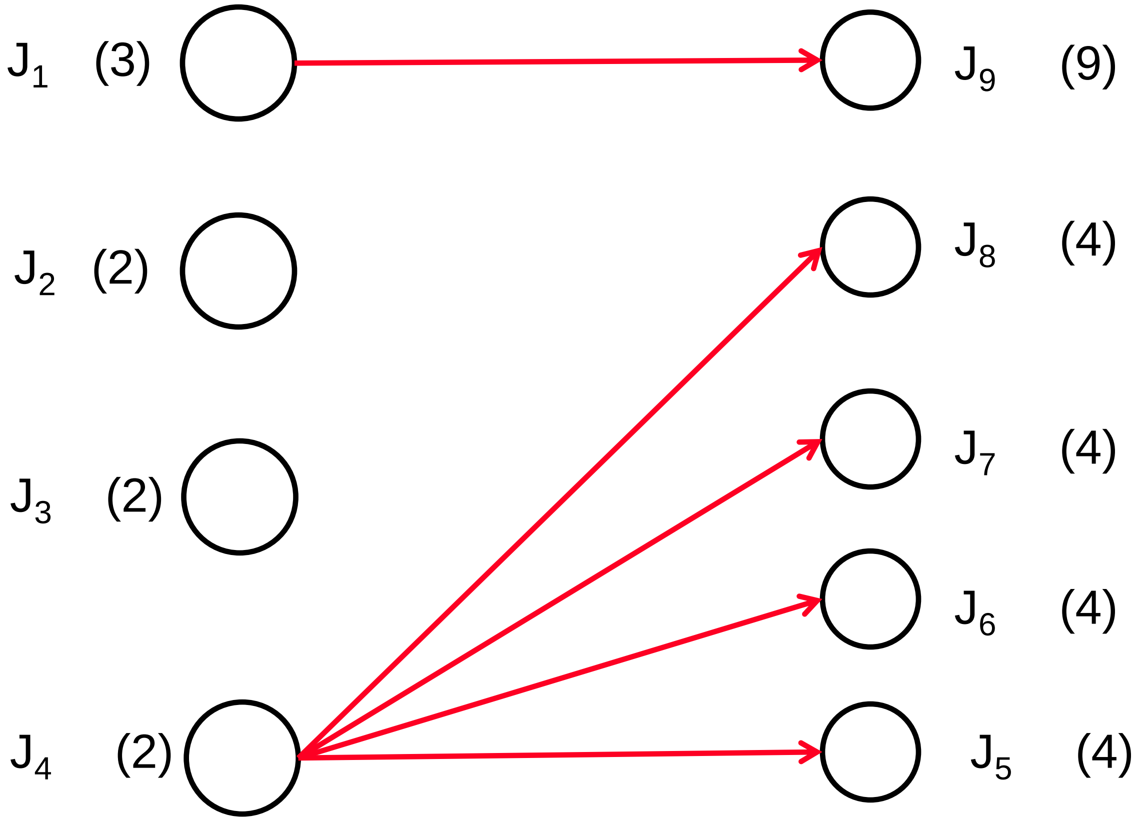
- Real-time computing $\neq$ fast computing
- Increase of computing power $\neq$ improvement of the performance

- **Graham's theorem**
  - 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 increase the schedule length.

Examples of Graham's theorem(1)

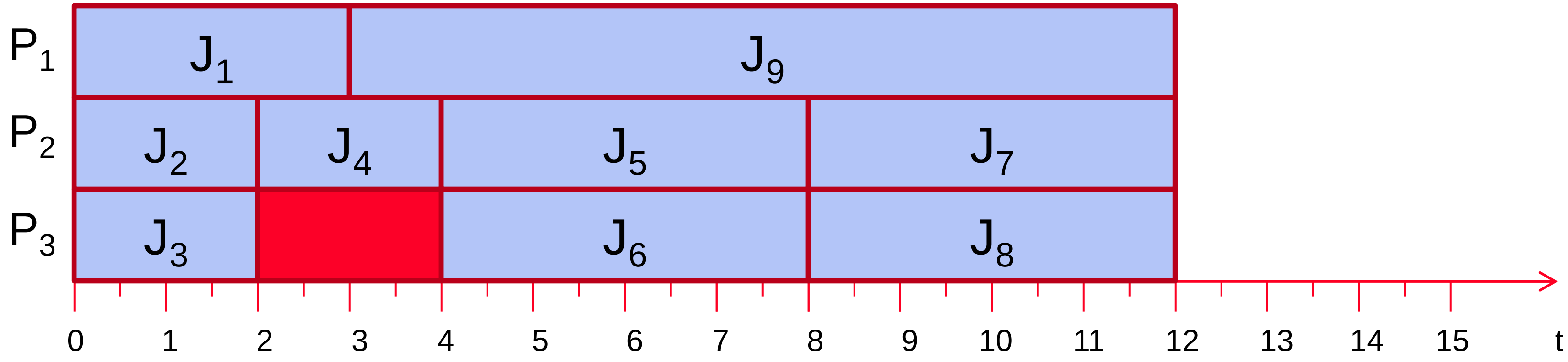
- $J = \{J_1, \ldots, J_9\}$: A task set
  - Sorted by decreasing priorities
  - Has the precedence constraints in next slide
- Three processors
- Optimal schedule σ^*
  - in Figure next slide
  - Global completion time is 12

Task precedence constraints



priority(J_i) > priority(J_j) for all $i < j$

Optimal schedule



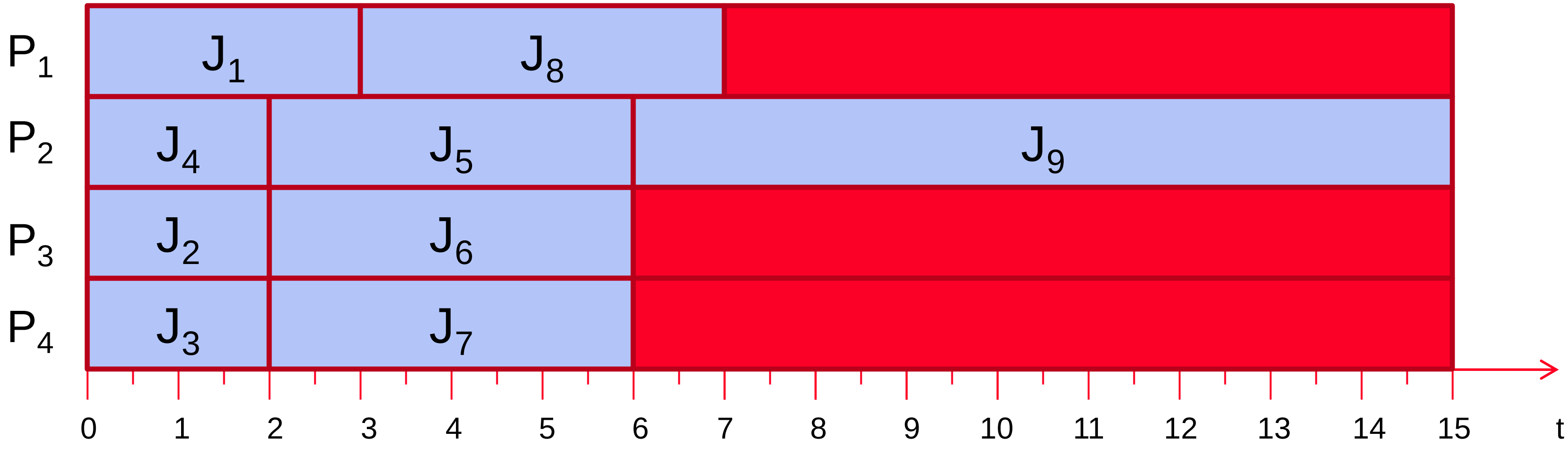
optimal schedule of task set J on a three-processor machine

➔ Find the global completion time if

- Extra processors are added
- Tasks execution time are reduced
- Precedence constrain is weakened

Examples of Graham's theorem(2)

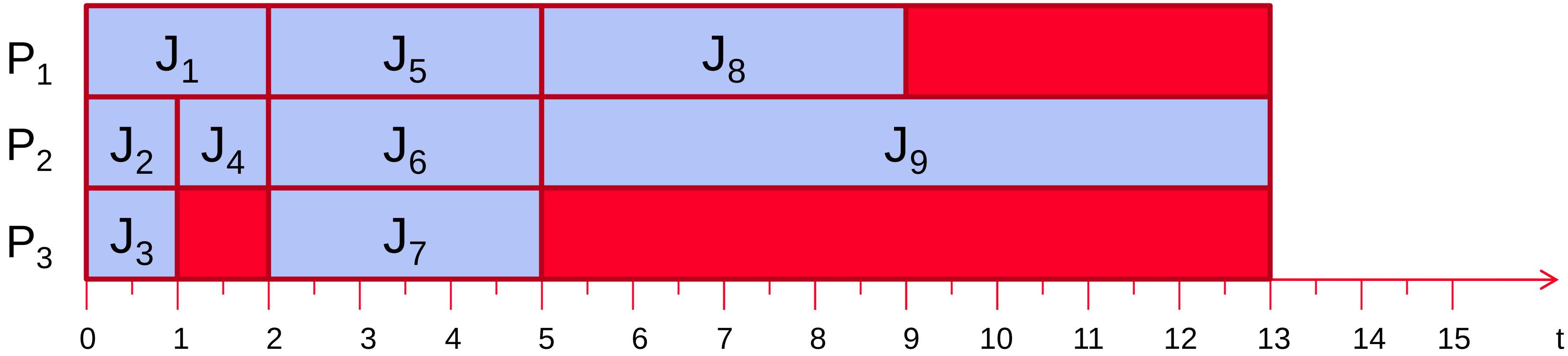
- Increasing the number of processors
  - Global completion time is 15



Schedule of task set J on a four-processor machine

Examples of Graham's theorem(3)

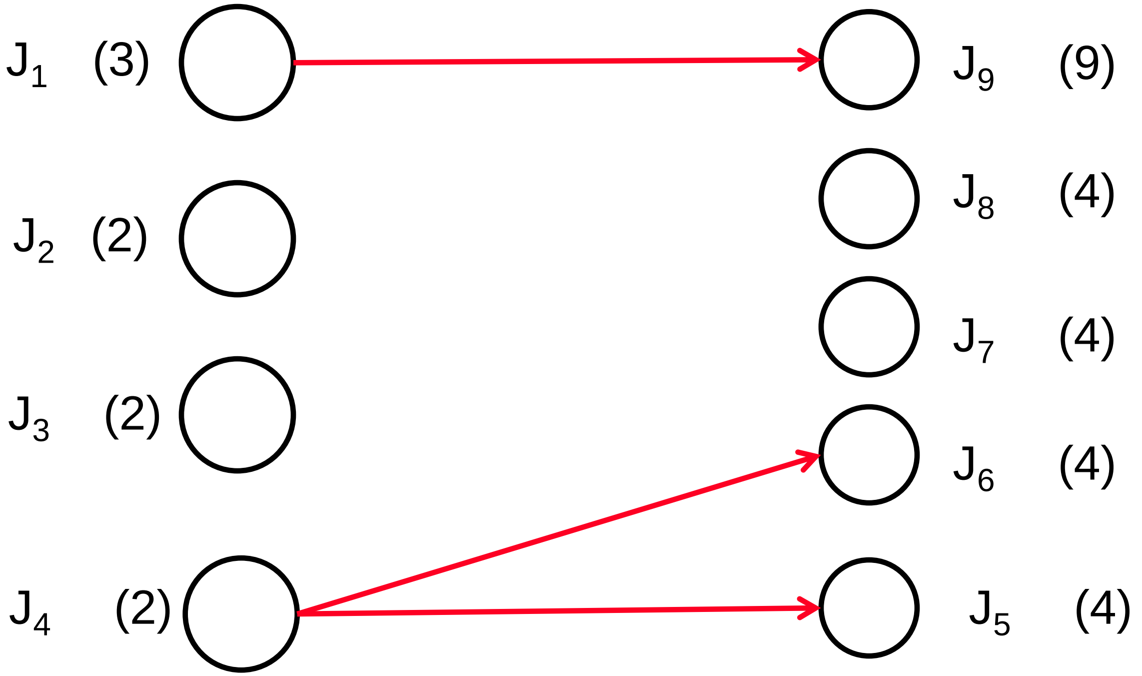
- Reducing computation time
  - Global completion time is 13



Schedule of task set J on a three-processor machine with computation times reduced by one unit of time

Examples of Graham's theorem(4)

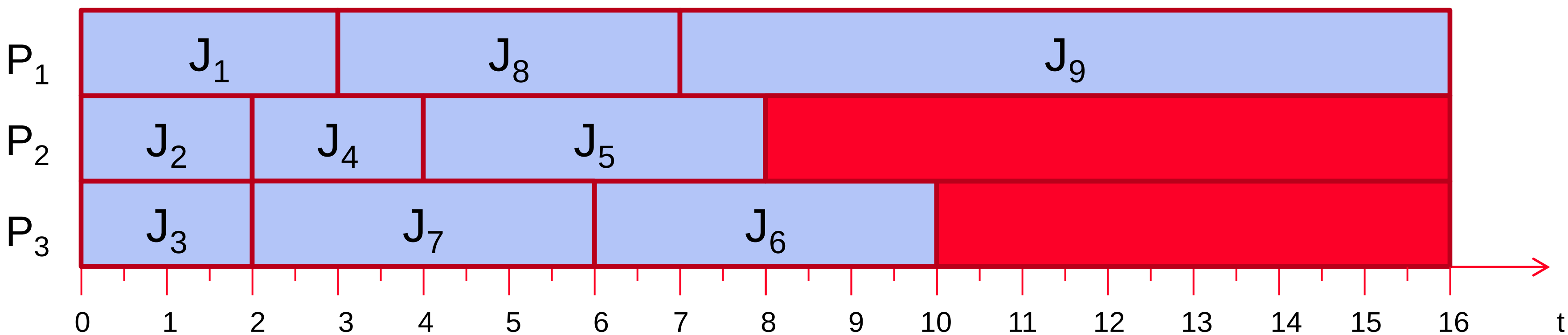
- Weakening precedence constraints
  - Global completion time is 16



Precedence graph of task set J obtained by removing the constraints on task J_5 and J_6

Examples of Graham's theorem(4)

- Weakened precedence constraints
  - Global completion time is 16



Schedule of task set J on a three-processor machine with precedence constraints weakened

Examples of Graham's theorem(5)

- Anomalies under resource constrains
  - In real-time tasks with shared resource
  - Global completion time is increased by reducing the computation time.