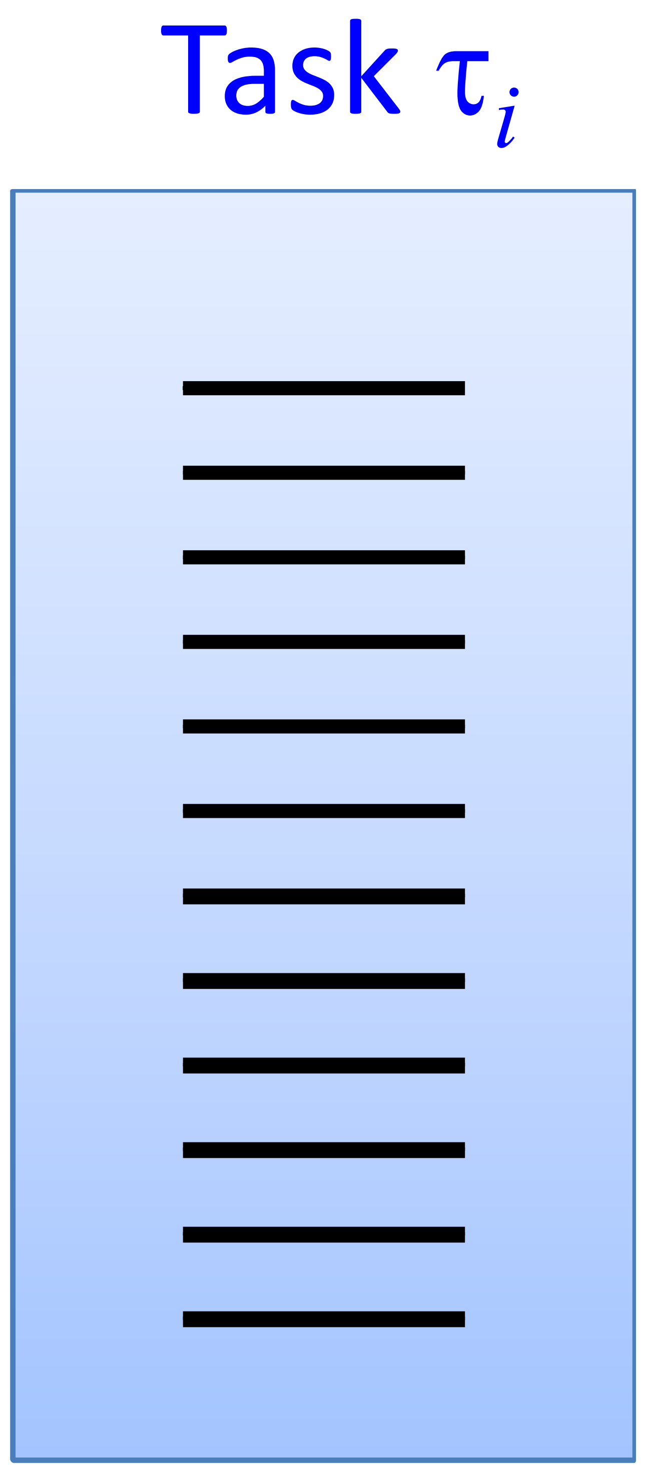
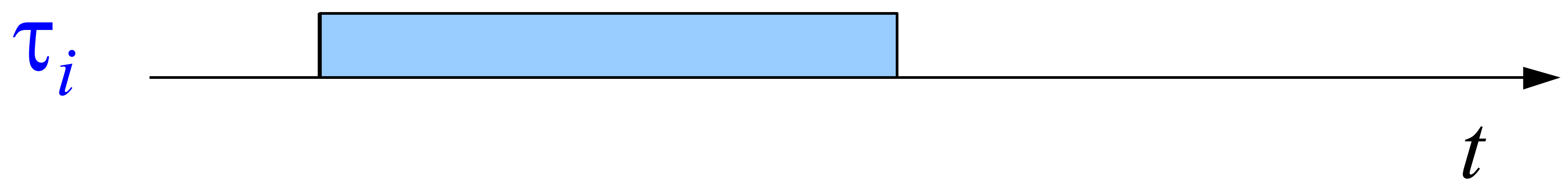
Task Scheduling

Slides source: Prof. Giorgio Buttazzo

Definition: Task

A **task** (or **thread**) is a sequence of instructions that in the absence of other activities is continuously executed by the processor until completion.

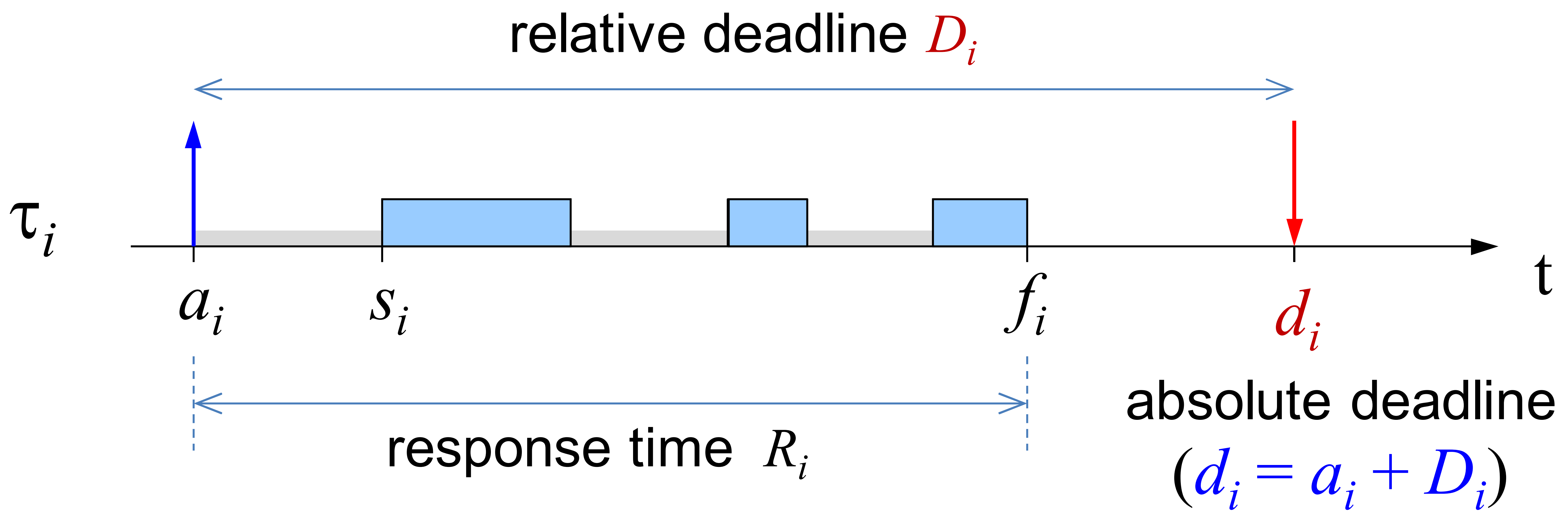




What are the important variables that characterize a computation if we want to perform a timing analysis?

Definition: Real-Time Task

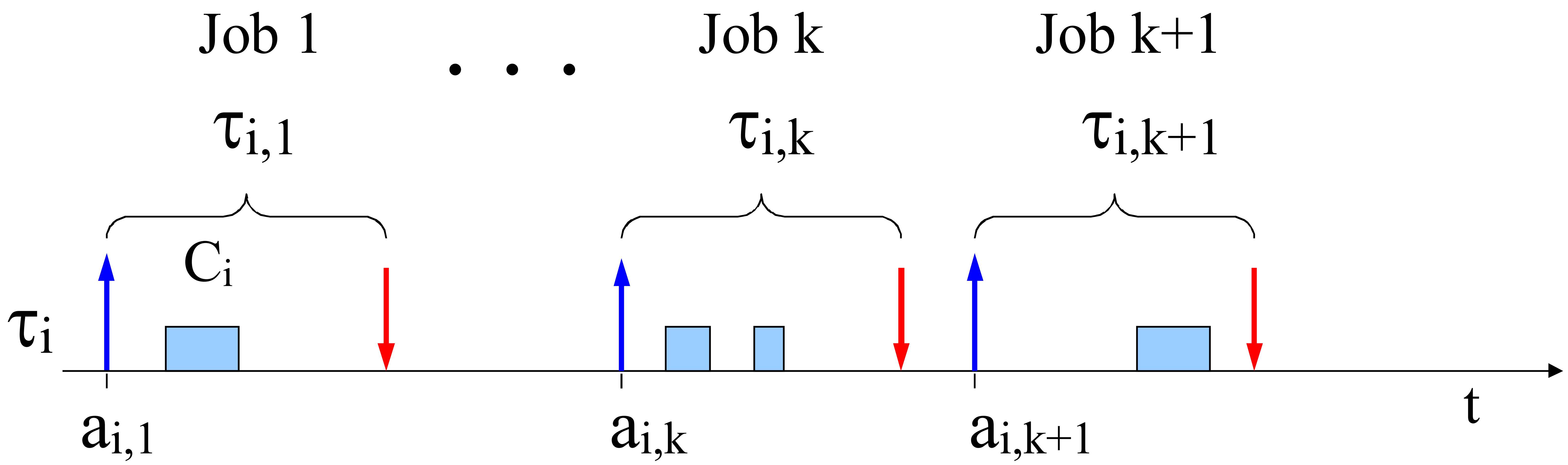
It is a task with a timing constraint on its response time, called deadline:



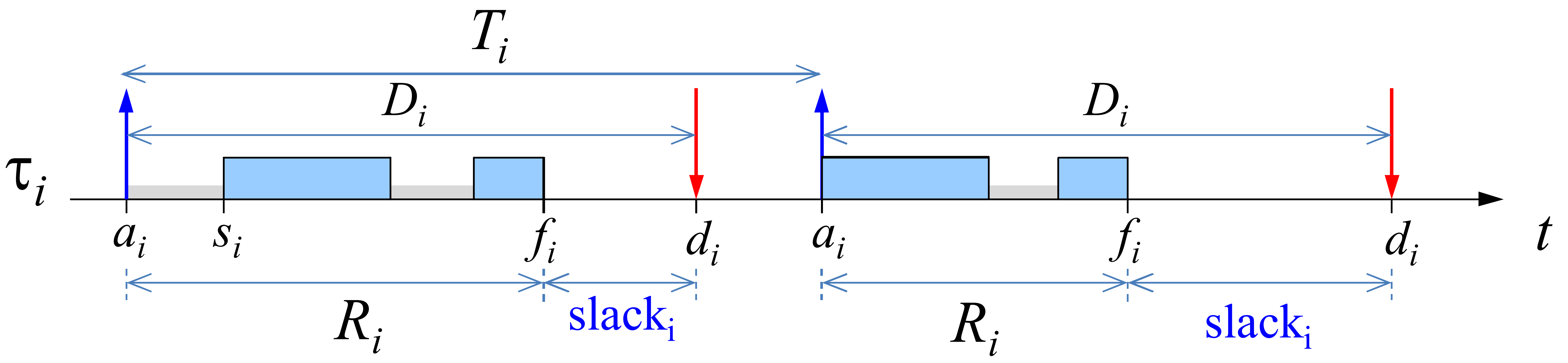
A real-time task τ_i is said to be feasible if it is guaranteed to complete within its deadline, that is, if $f_i \leq d_i$ (or $R_i \leq D_i$).

Tasks and jobs

A task running several times on different input data generates a sequence of instances (or jobs):



Parameters summary



- Computation time (C_i)
- Period (T_i)
- Relative deadline (D_i)

These parameters are specified by the programmer and are known off-line

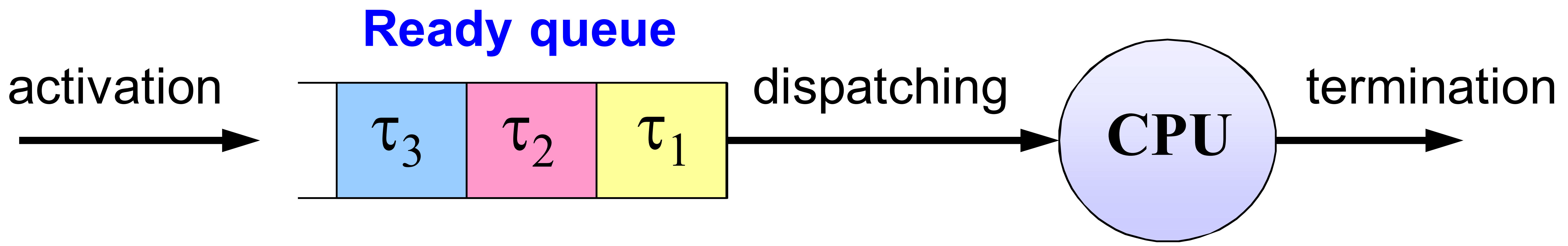
- Arrival time (a_i)
- Start time (s_i)
- Finishing time (f_i)
- Response time (R_i)
- Slack and Lateness
- Jitter

These parameters depend on the scheduler and on the actual execution, and are known at run time.

Ready queue

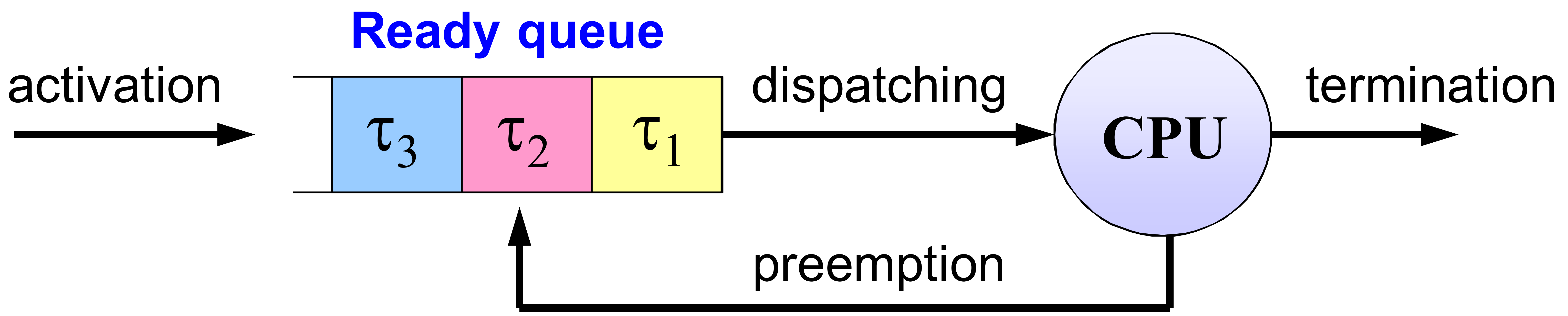
In a concurrent system, more tasks can be simultaneously active, but only one can be in execution (running).

- An active task that is not in execution is said to be ready.
- Ready tasks are kept in a ready queue, managed by a scheduling policy.
- The processor is assigned to the first task in the queue through a dispatching operation.



Preemption

It is a kernel mechanism that allows to suspend the execution of the running task in favor of a more important task. The suspended task goes back in the ready queue.



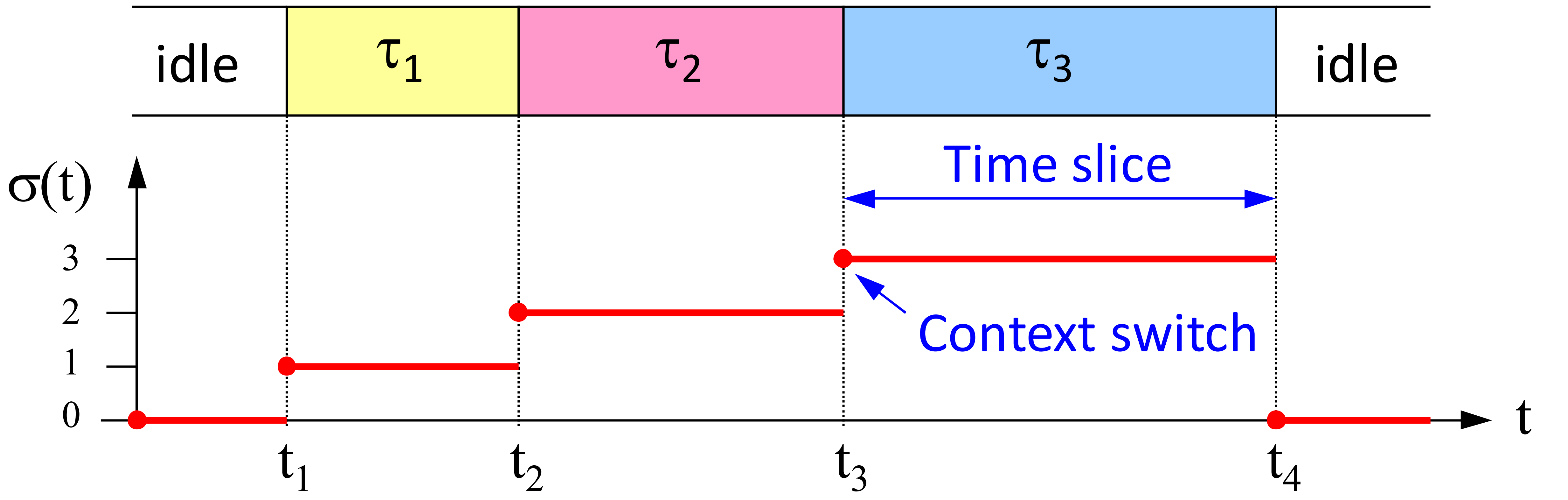
- Preemption enhances concurrency and allows reducing the response times of high priority tasks.
- It can be disabled (completely or temporarily) to ensure the consistency of certain critical operations.

Schedule

A schedule is a specific allocation of tasks to the processor, which determines the corresponding execution sequence.

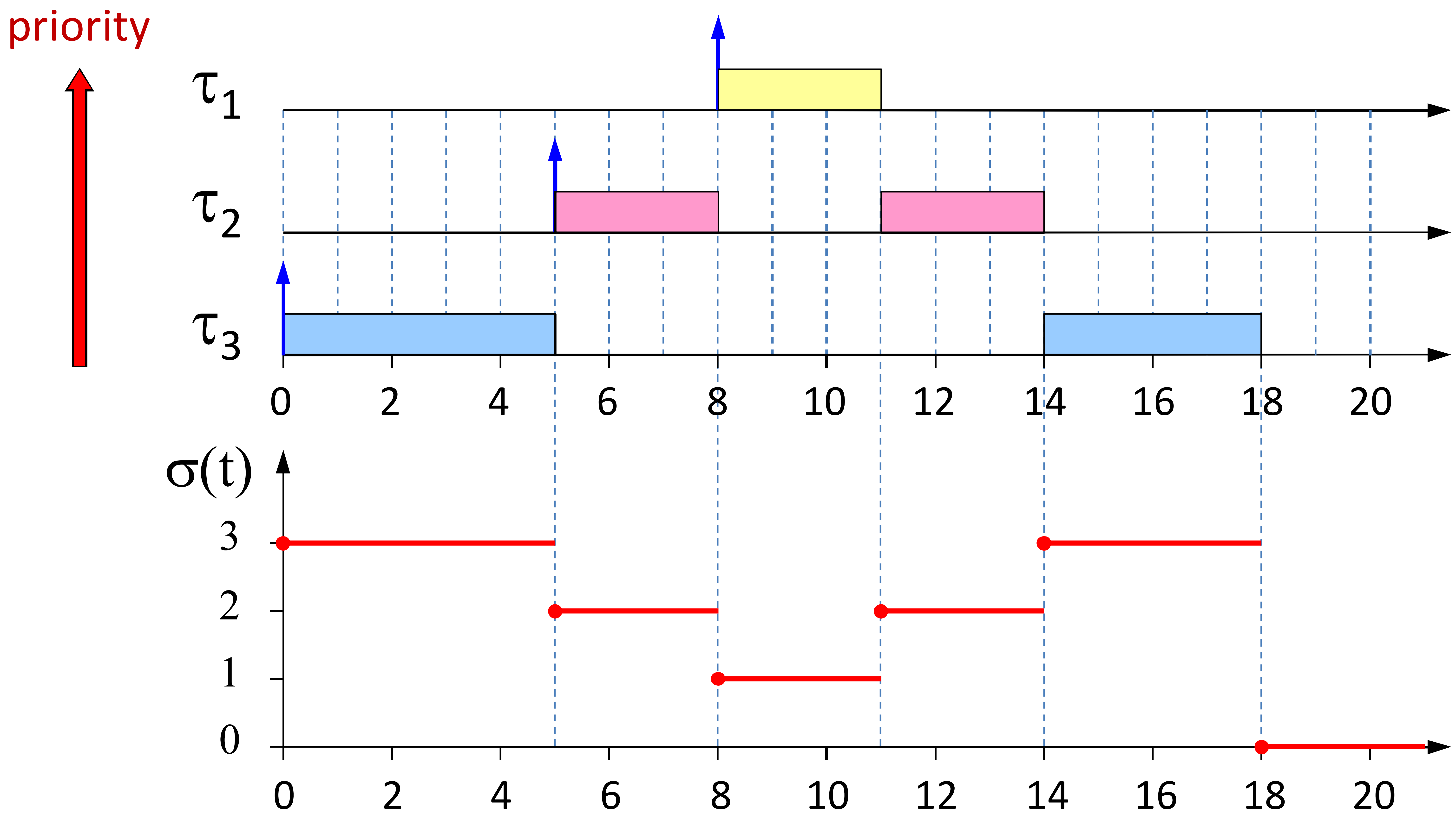
Formally, given a task set $\Gamma = \{\tau_1, \ldots, \tau_n\}$, a schedule is a function $\sigma: R^+ \rightarrow N$ that associates an integer k to each interval of time $[t, t+1)$ with the following meaning:

$$\begin{cases} k = 0 & \Rightarrow \text{ in } [t, t+1) \text{ the processor is IDLE} \\ k > 0 & \Rightarrow \text{ in } [t, t+1) \text{ the processor executes } \tau_k \end{cases}$$

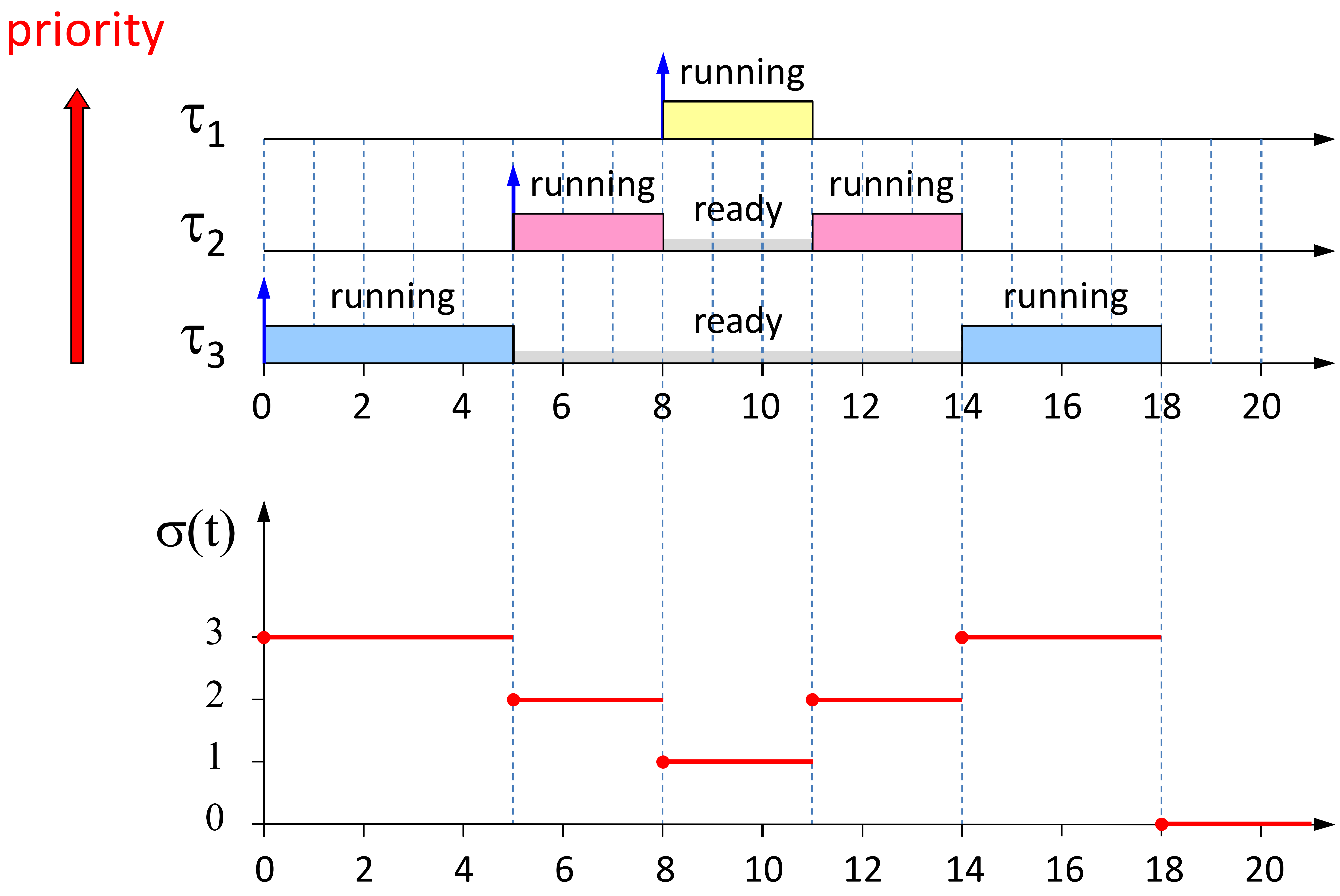


Preemptive schedule

A schedule is said to be preemptive if a task can be interrupted at any time in favor of another task and then resumed later:

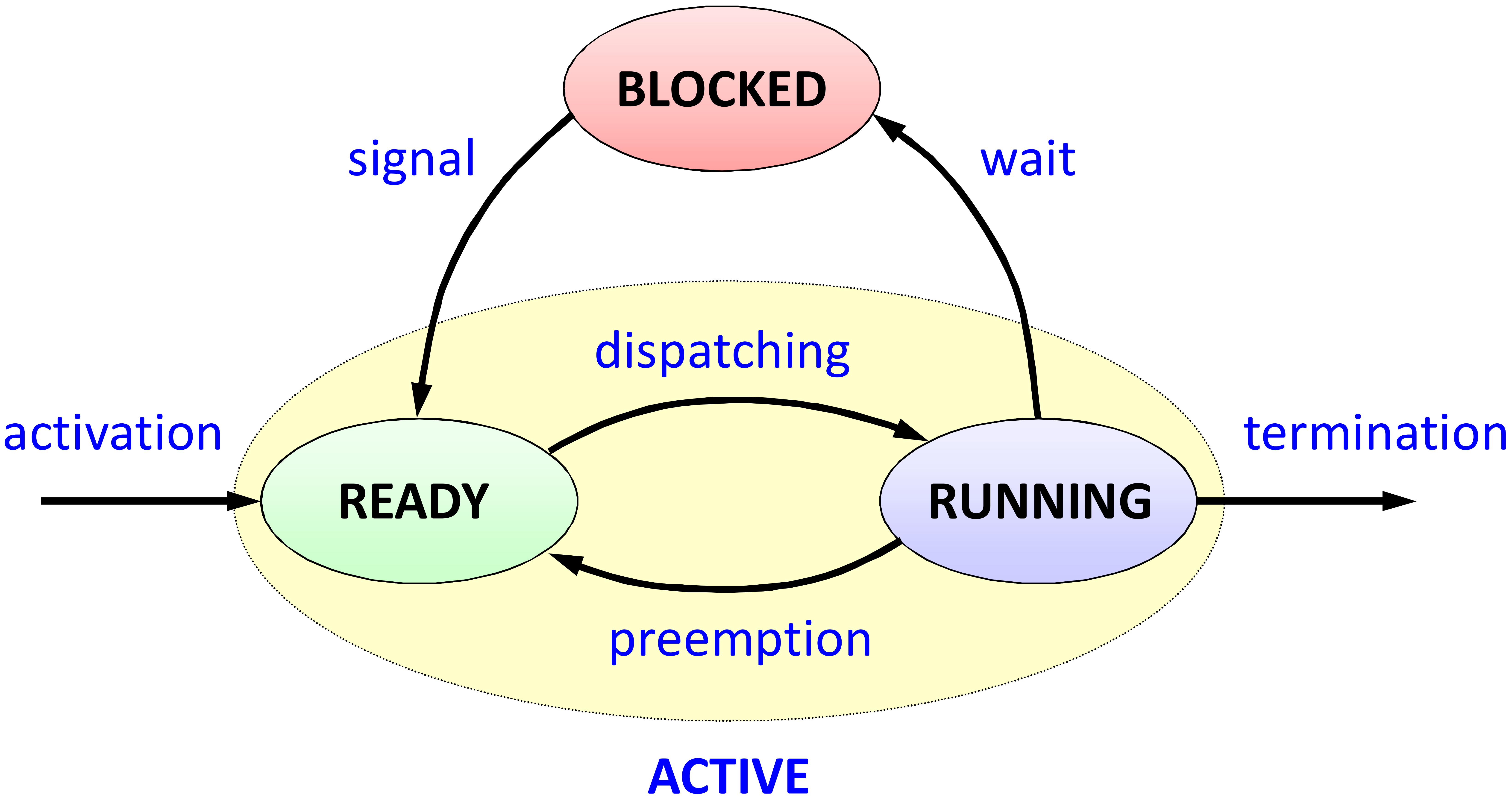


Task states



priority
running
τ_1
running
ready
running
τ_2
running
ready
running
τ_3
0 2 4 6 8 10 12 14 16 18 20
$\sigma(t)$
3
2
1
0
0 2 4 6 8 10 12 14 16 18 20

Task states


BLOCKED
signal
wait
dispatching
activation
READY
RUNNING
termination
preemption
ACTIVE

Activation modes

- **Periodic** (time-driven activation)

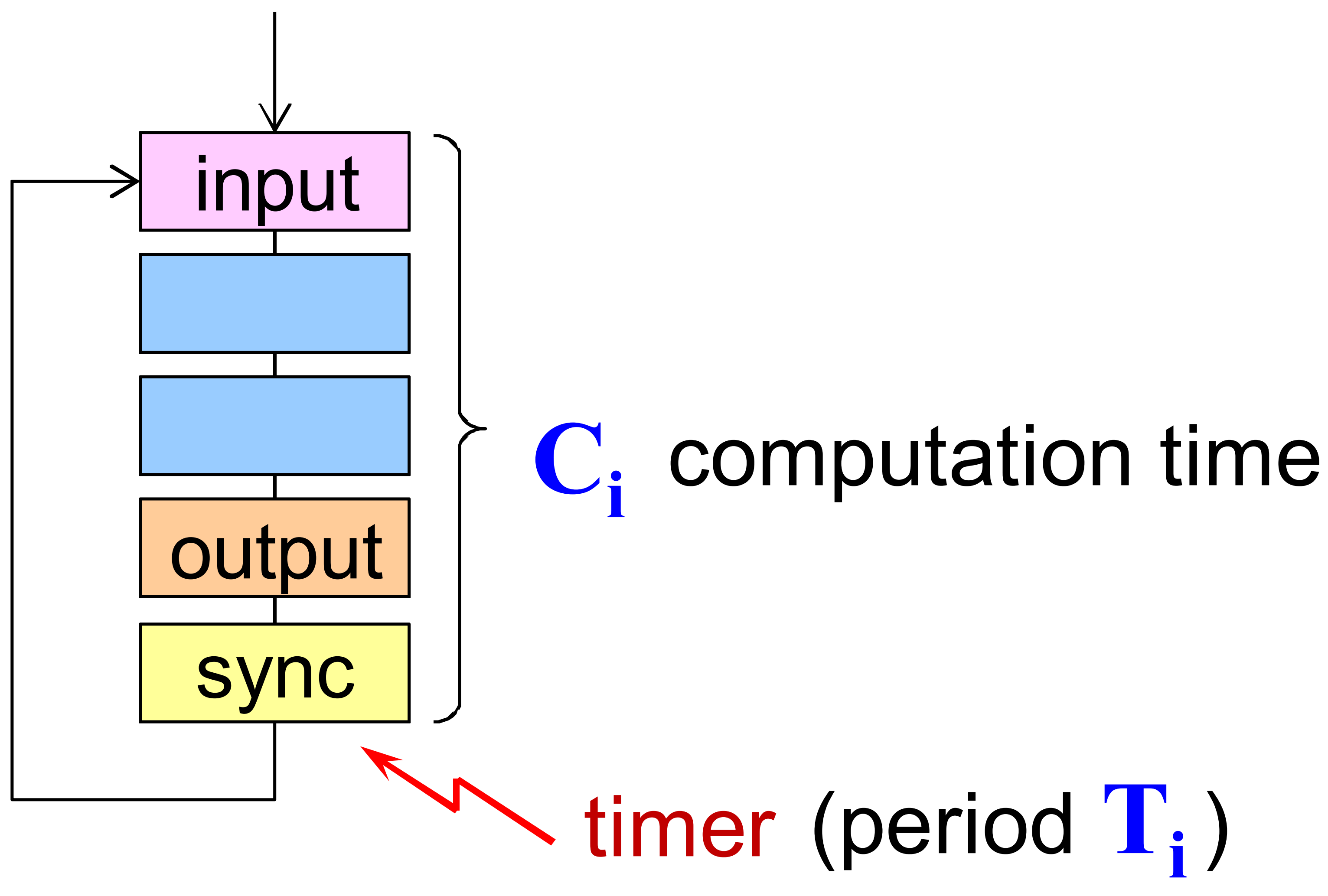
  A task is said to be periodic if its jobs are automatically activated by the operating system at predefined time instants.

- **Aperiodic** (event-driven activation)

  A task is said to be aperiodic if its jobs are activated at the arrival of an event (by interrupt or by another task through an explicit system call).

  If the activation interval between consecutive jobs cannot be smaller than a given quantity, the task is said to be **sporadic**.

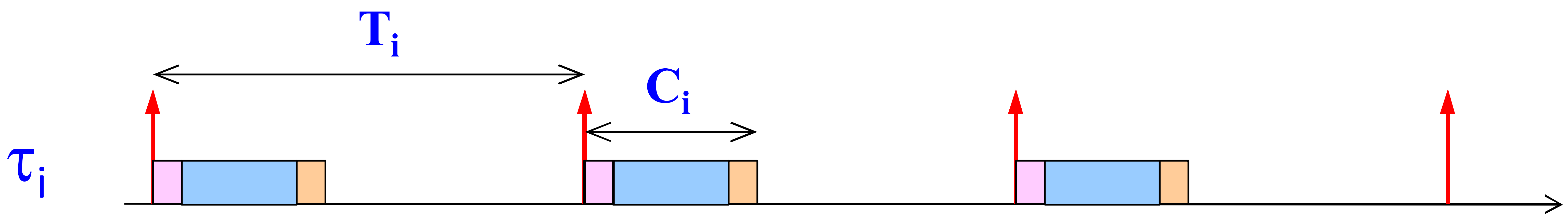
Periodic task



$$U_i = \frac{C_i}{T_i}$$

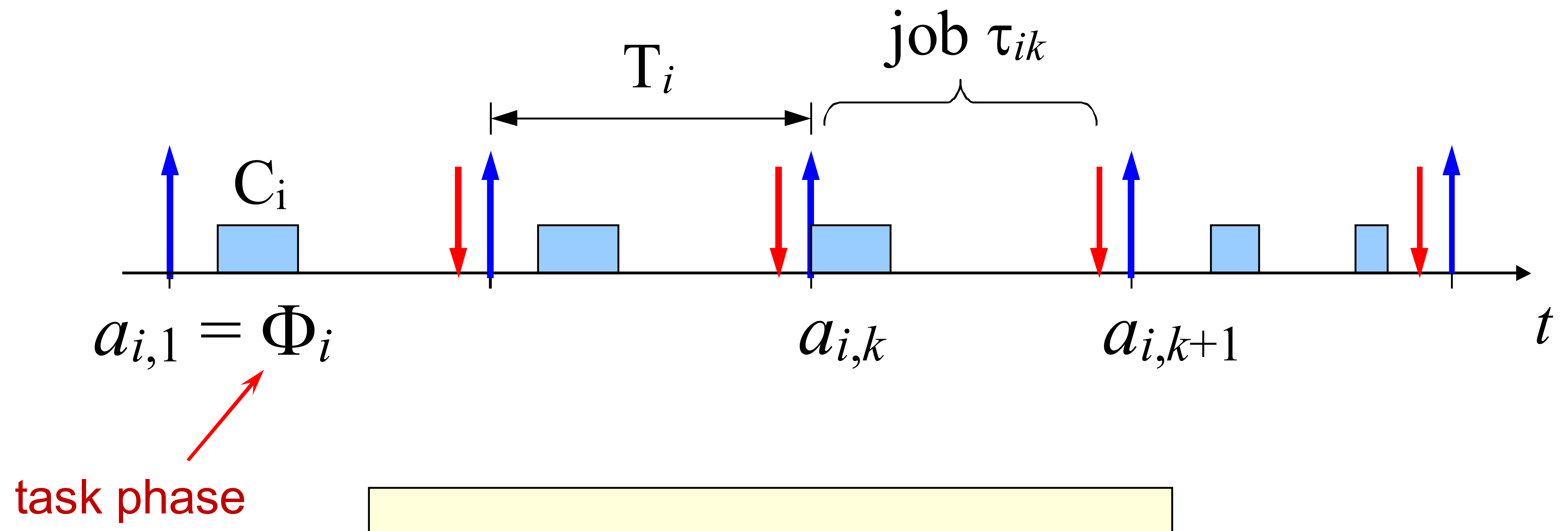
utilization factor

- A periodic task τ_i generates an infinite sequence of jobs: τ_{i1}, τ_{i2}, …, τ_{ik} (same code on different data):



Periodic task

A periodic task can be fully described by four parameters only: phase (Φ_i), worst-case computation time (C_i), period (T_i), and relative deadline (D_i).



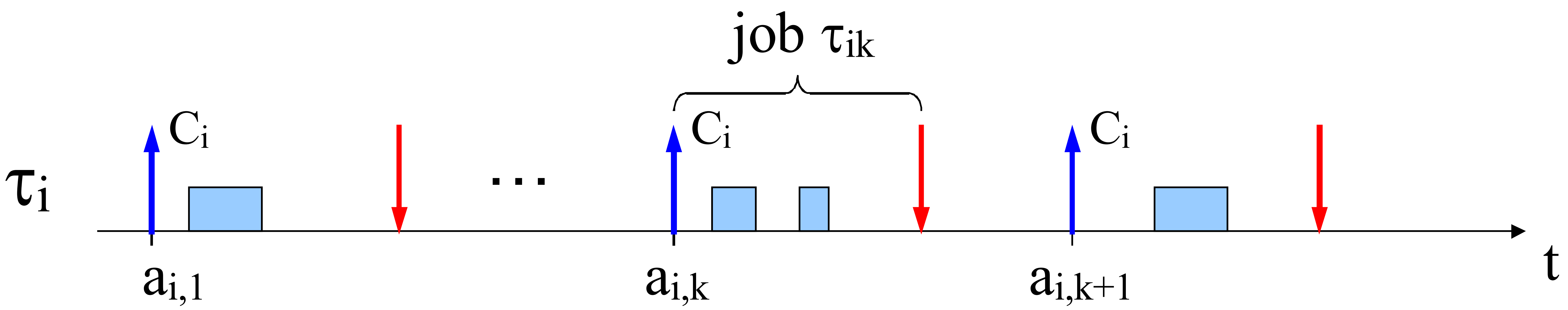
$$a_{i,k} = \Phi_i + (k-1)\, T_i$$

$$d_{i,k} = a_{i,k} + D_i$$

Aperiodic task

- **Aperiodic:** $a_{i,k+1} > a_{i,k}$
- **Sporadic:** $a_{i,k+1} \geq a_{i,k} + T_i$

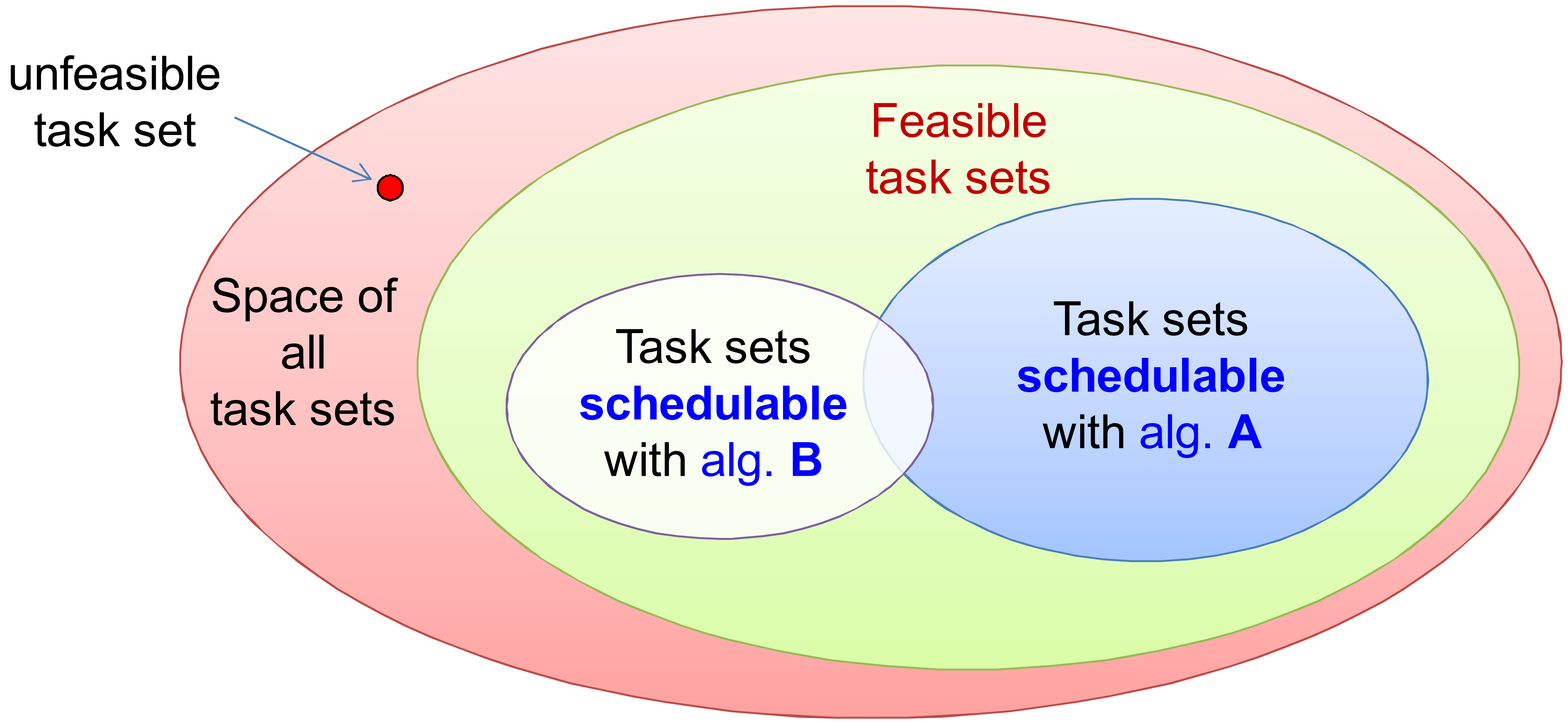
minimum interarrival time



Definitions

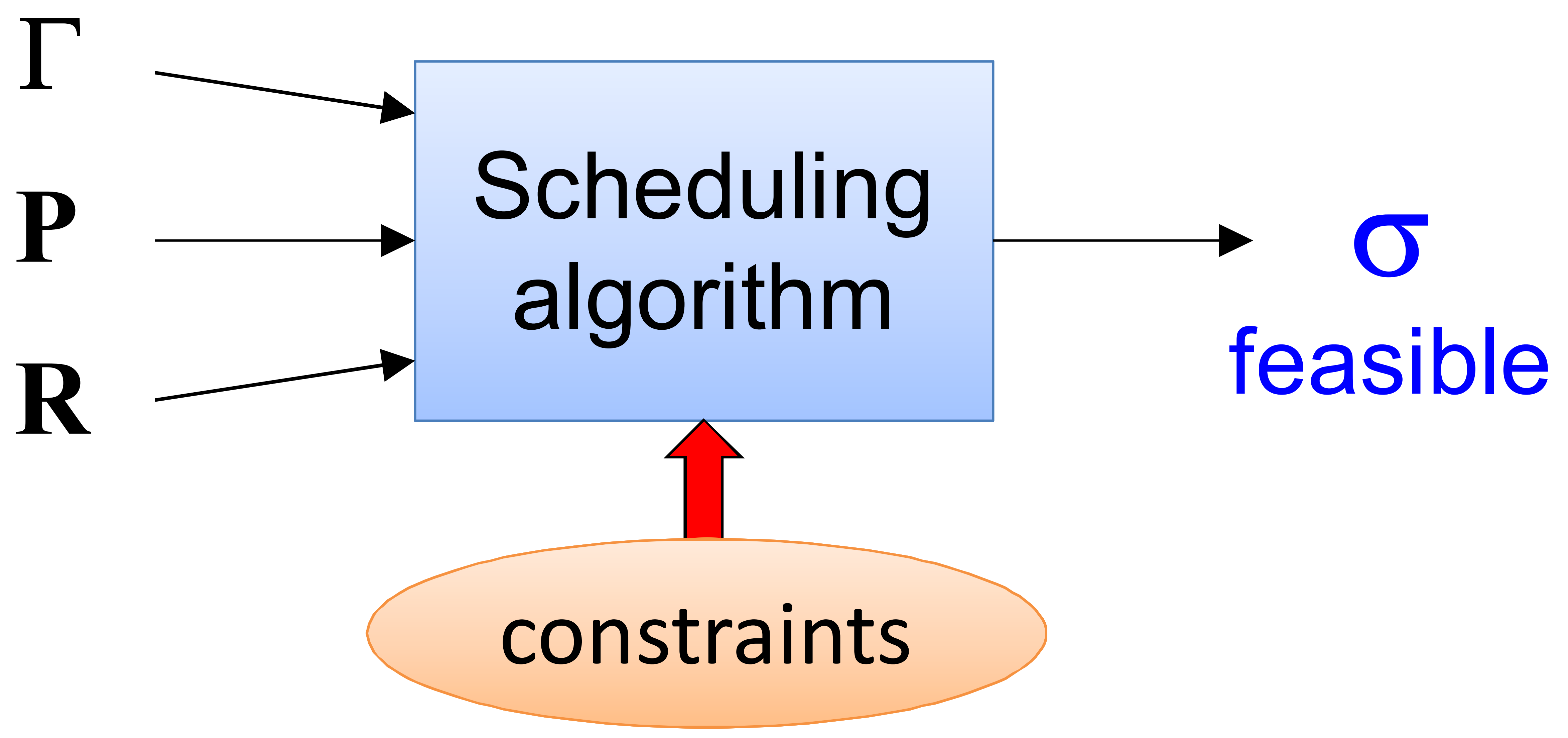
A schedule σ is said to be **feasible** if it satisfies a set of constraints.

A task set Γ is said to be **schedulable** with an algorithm A, if A generates a feasible schedule.



The scheduling problem

Given a set Γ of n tasks, a set **P** of p processors, and a set **R** of r resources, find an assignment of **P** and **R** to Γ that produces a feasible schedule under a set of constraints.



Complexity

- In 1975, Garey and Johnson showed that the general scheduling problem is **NP hard**.

  In practice, it means that the time for finding a feasible schedule grows exponentially with the number of tasks.

Fortunately, polynomial time algorithms can be found under particular conditions.

Why do we care about complexity?

- Let's consider an application with n **= 30** tasks on a processor in which the elementary step takes **1 μs**

- Consider 3 algorithms with the following complexity:

A_1: $O(n)$ → **30 μs**

A_2: $O(n^8)$ → **182 hours**

A_3: $O(8^n)$ → **40.000 billion years**

Simplifying assumptions

- Single processor
- Homogeneous task sets
- Fully preemptive tasks
- Simultaneous activations
- No precedence constraints
- No resource constraints

Algorithm taxonomy

- Preemptive vs. Non Preemptive
- Static vs. dynamic
- On line vs. Off line
- Optimal vs. Heuristic

Static vs. Dynamic

Static

scheduling decisions are taken based on fixed parameters, statically assigned to tasks before activation.

Dynamic

scheduling decisions are taken based on parameters that can change with time.

Off-line vs. On-line

Off-line

all scheduling decisions are taken before task activation: the schedule is stored in a table (**table-driven scheduling**).

On-line

scheduling decisions are taken at run time on the set of active tasks.

Optimal vs. Heuristic

Optimal

They generate a schedule that minimizes a cost function, defined based on an optimality criterion.

Heuristic

They generate a schedule according to a heuristic function that tries to satisfy an optimality criterion, but there is no guarantee of success.

Optimality criteria

- **Feasibility**: Find a feasible schedule if there exists one.
- Minimize the **maximum lateness**
- Minimize the **number of deadline miss**
- Assign a value to each task, then maximize the **cumulative value** of the feasible tasks

Task set assumptions

We consider algorithms for different types of tasks:

- **Single-job tasks (one shot)**

  tasks with a single activation (not recurrent)

- **Periodic tasks**

  recurrent tasks regularly activated by a timer (each task potentially generates infinite jobs)

- **Aperiodic/Sporadic tasks**

  recurrent tasks irregularly activated by events (each task potentially generates infinite jobs)

- **Mixed task sets**

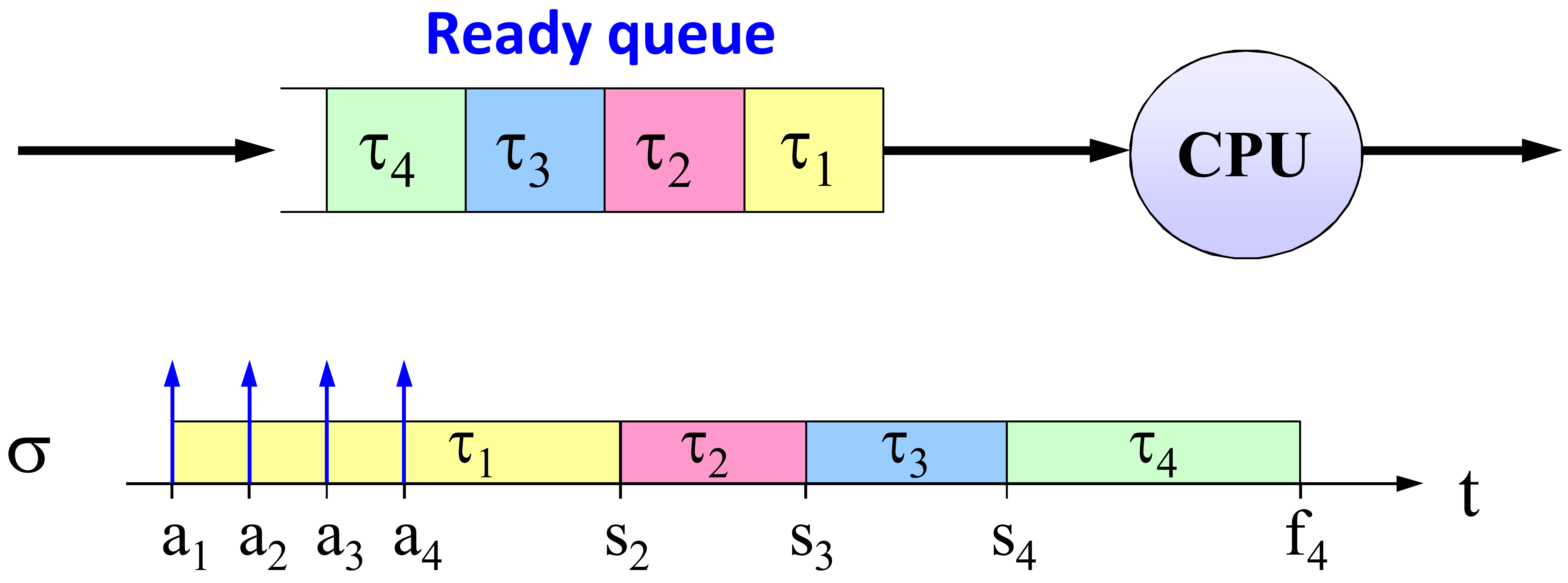
Classical scheduling policies

- First Come First Served
- Shortest Job First
- Priority Scheduling
- Round Robin

Not suited for real-time systems

First Come First Served

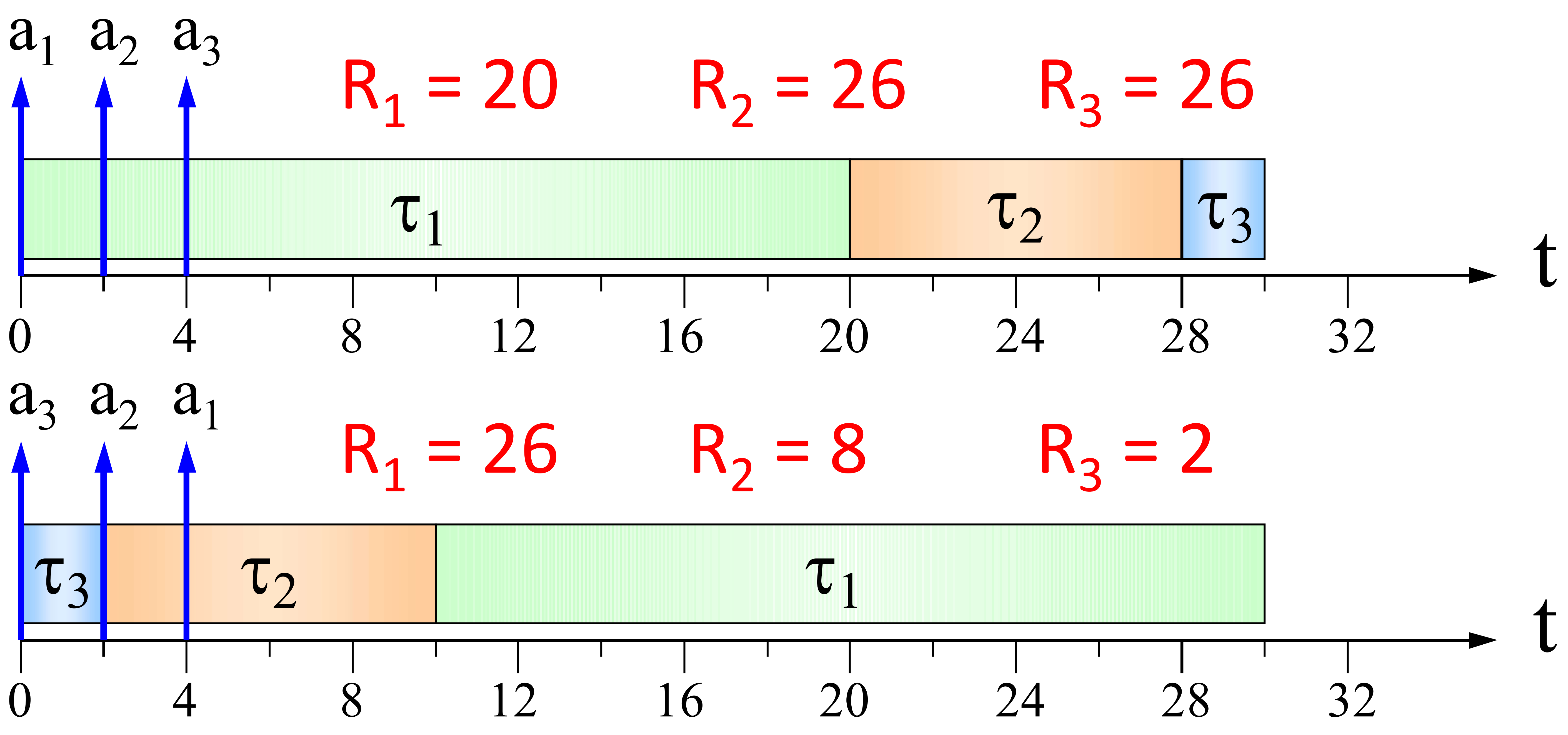
It assigns the CPU to tasks based on their arrival times (intrinsically non preemptive):



First Come First Served

- Very unpredictable

response times strongly depend on task arrivals:



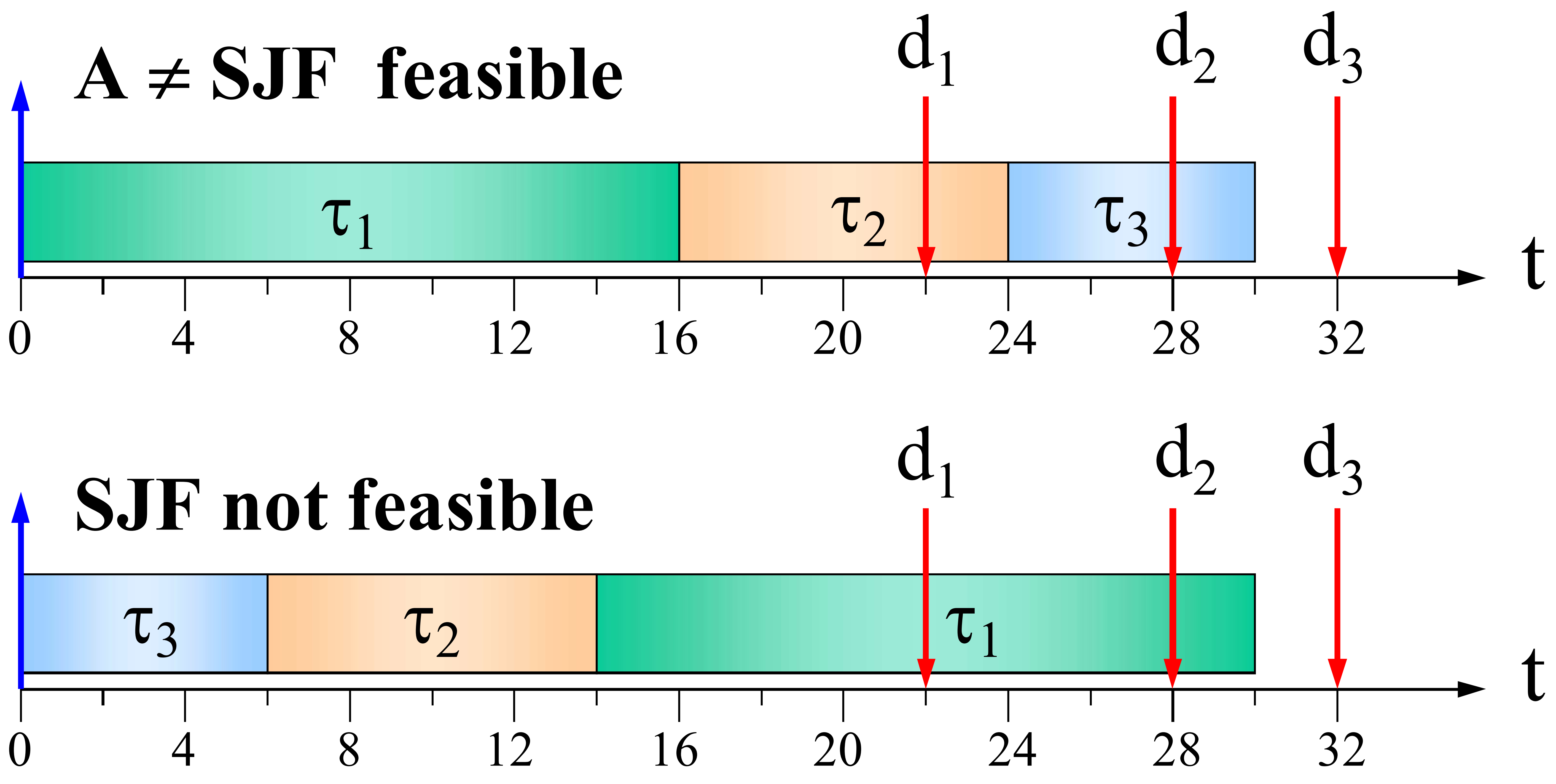
Shortest Job First (SJF)

It selects the ready task with the shortest computation time.

- Static (C$_i$ is a constant parameter)
- It can be used on line or off-line
- Can be preemptive or non preemptive
- It minimizes the average response time

Is SUF suited for Real-Time?

- It is not optimal in the sense of feasibility



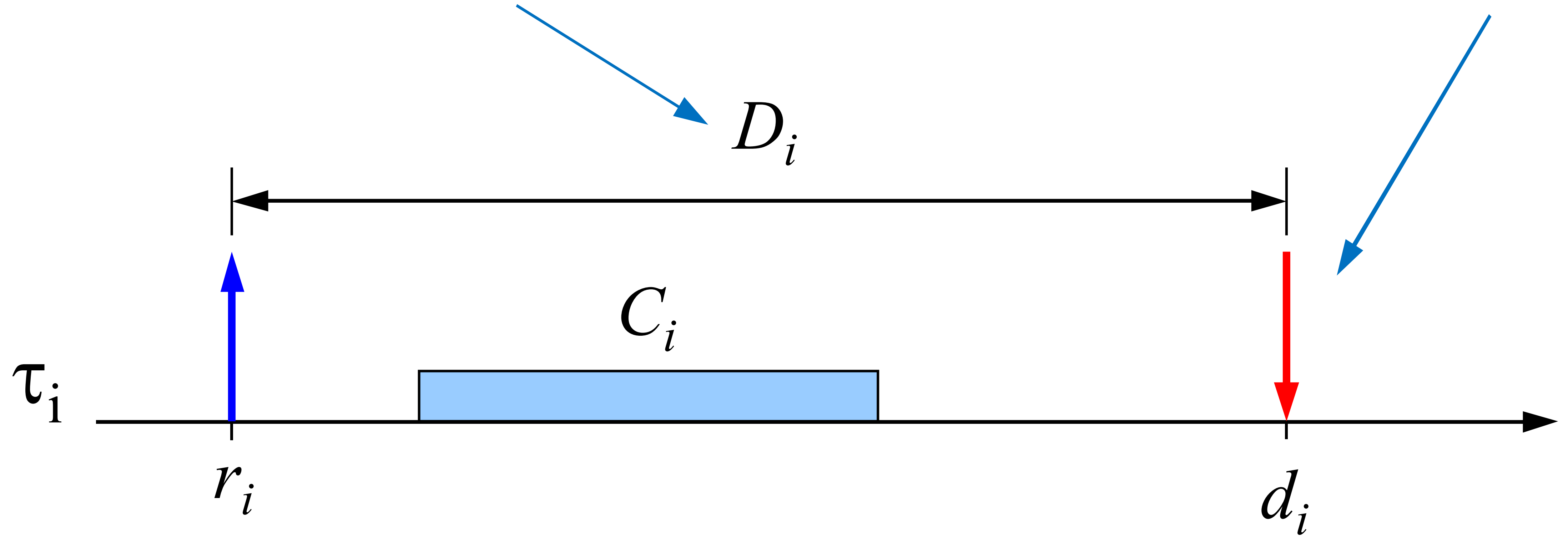
Real-Time Scheduling Algorithms

Real-Time Algorithms

Real-time scheduling algorithm can take scheduling decisions base on:

relative deadlines D_i
(**static algoritms**)

absolute deadlines d_i
(**dynamic algorithms**)



Some consider synchronous arrivals: $\forall i \; r_i = 0$ (**off-line algorithms**)

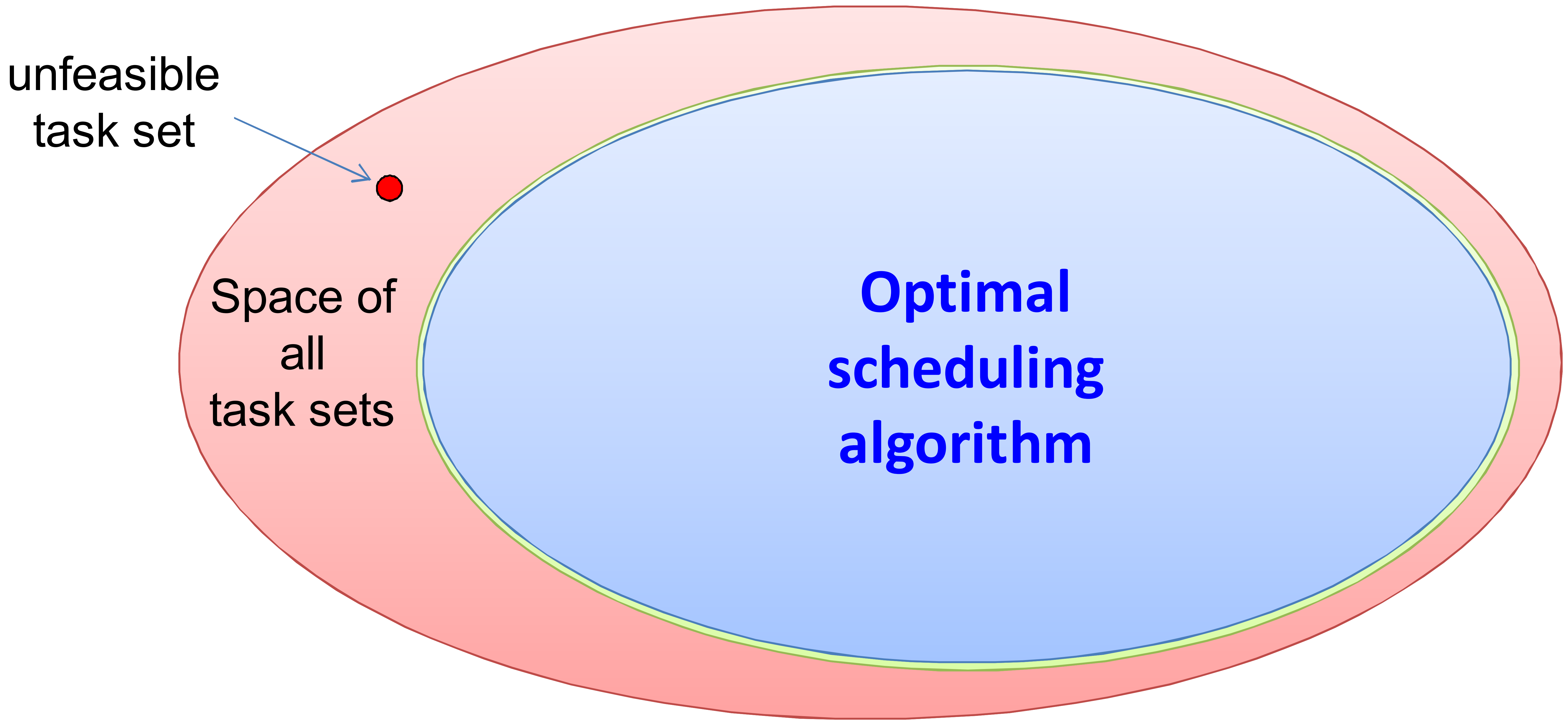
Some consider asynchronous arrivals: $\forall i \; r_i \geq 0$ (**on-line algorithms**)

On-line algorithms can be **preemptive** or **non preemptive**

(Preemption is not an issue if all tasks are ready at time $t = 0$).

Definitions

An optimal algorithm is able to generate a feasibile schedule for all feasible task sets.



A property of optimal algorithms

If a task set Γ is not schedulable by an optimal algorithm, then Γ cannot be scheduled by any other algorithm.

If an algorithm A minimizes L_{max} then A is also optimal in the sense of feasibility. The opposite is not true.