Embedded Software Engineering

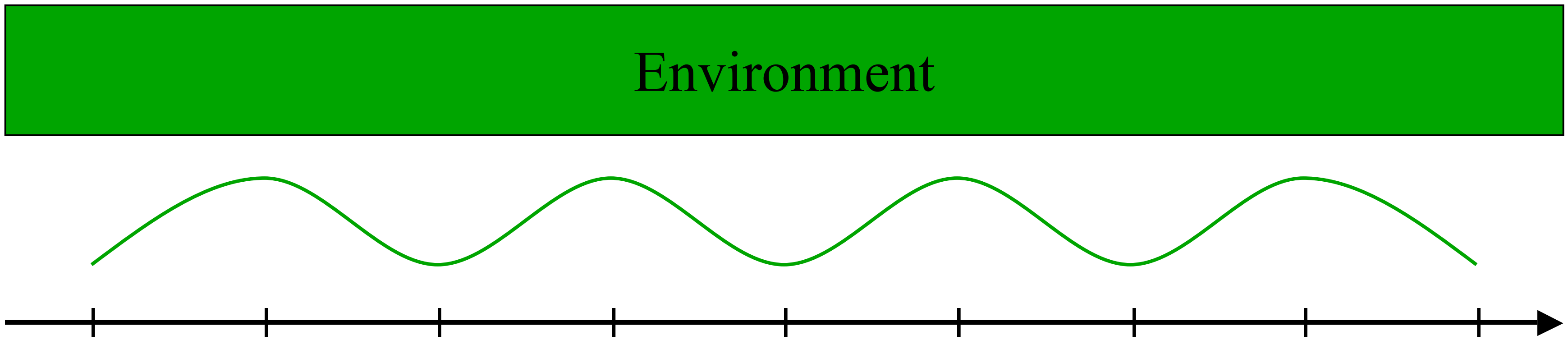
3 Unit Course, Winter 2004
CS Department, Univ. of Salzburg

Chapter 2: RT Scheduling

Christoph Kirsch

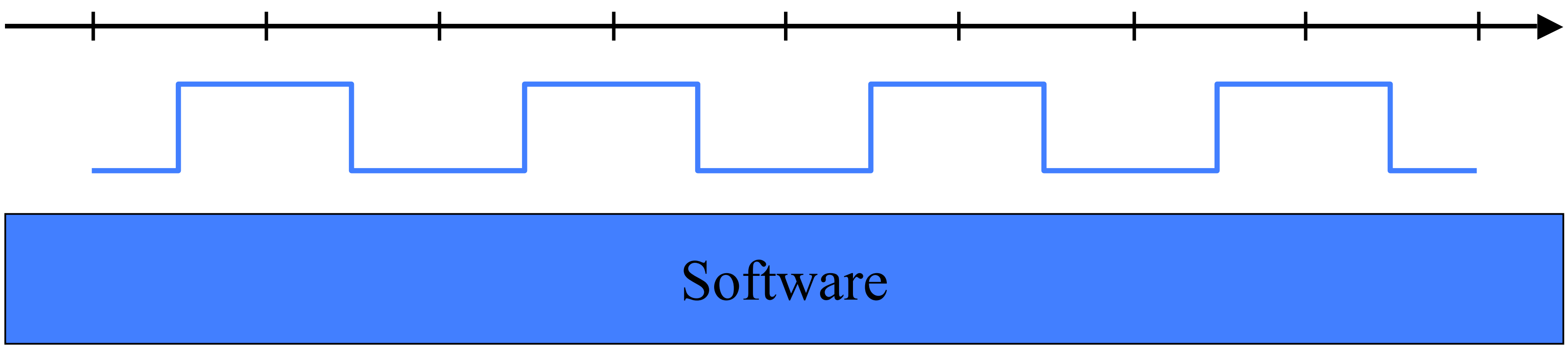
www.cs.uni-salzburg.at/~ck/teaching/ESE-Winter-2004

Platform Time is Platform Memory

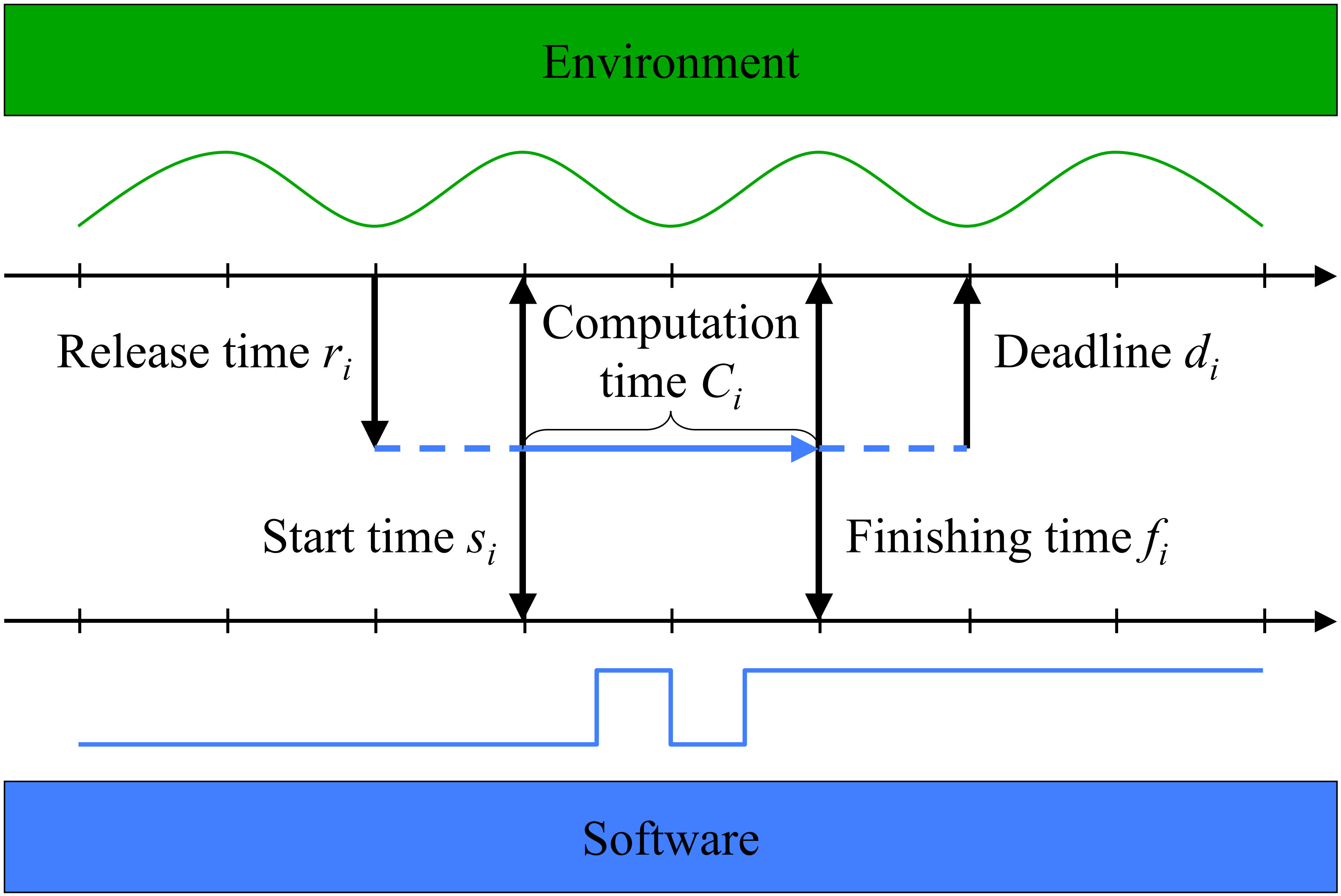


- Programming as if there is enough platform time

- Implementation checks whether there is enough of it

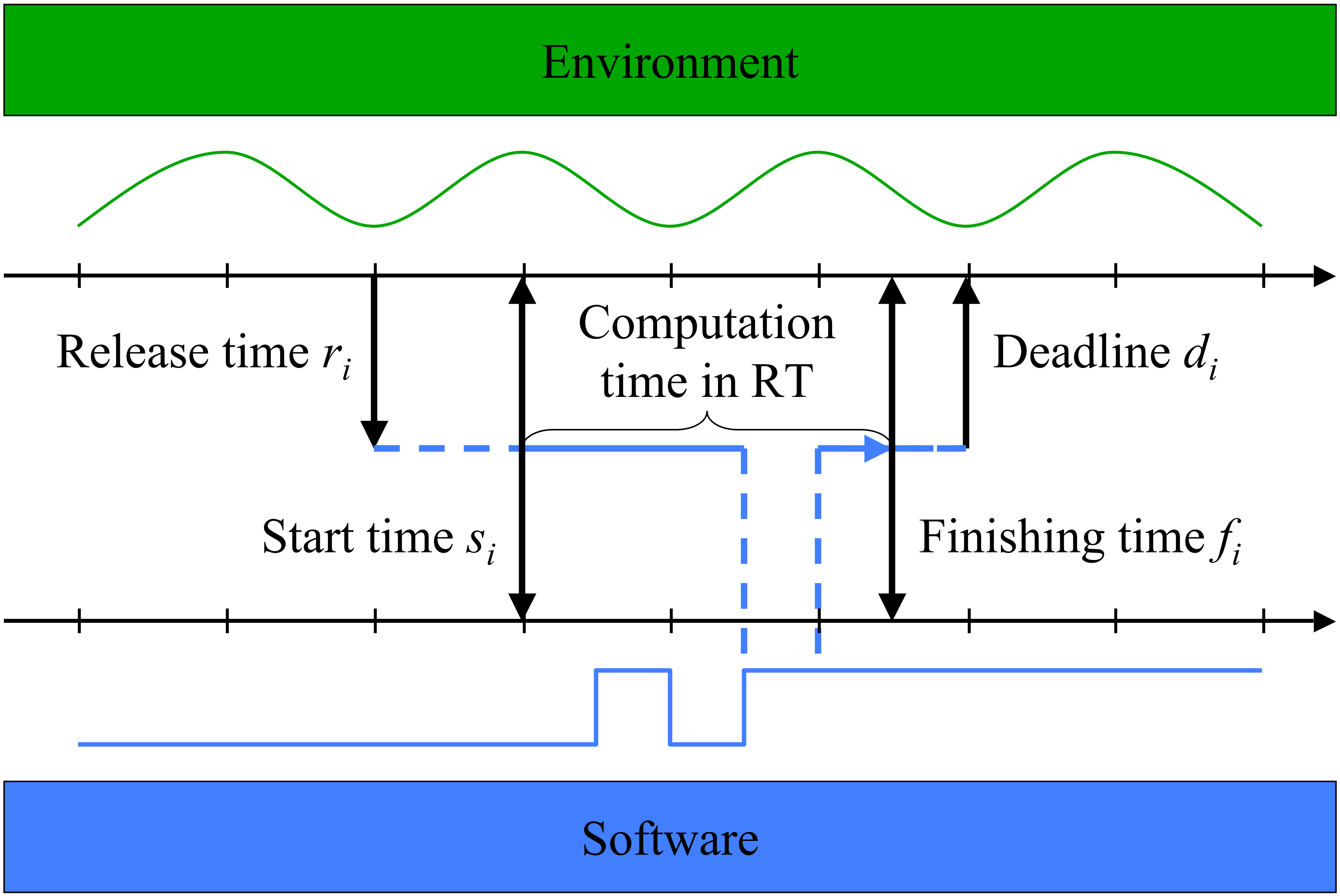


A Task T_i

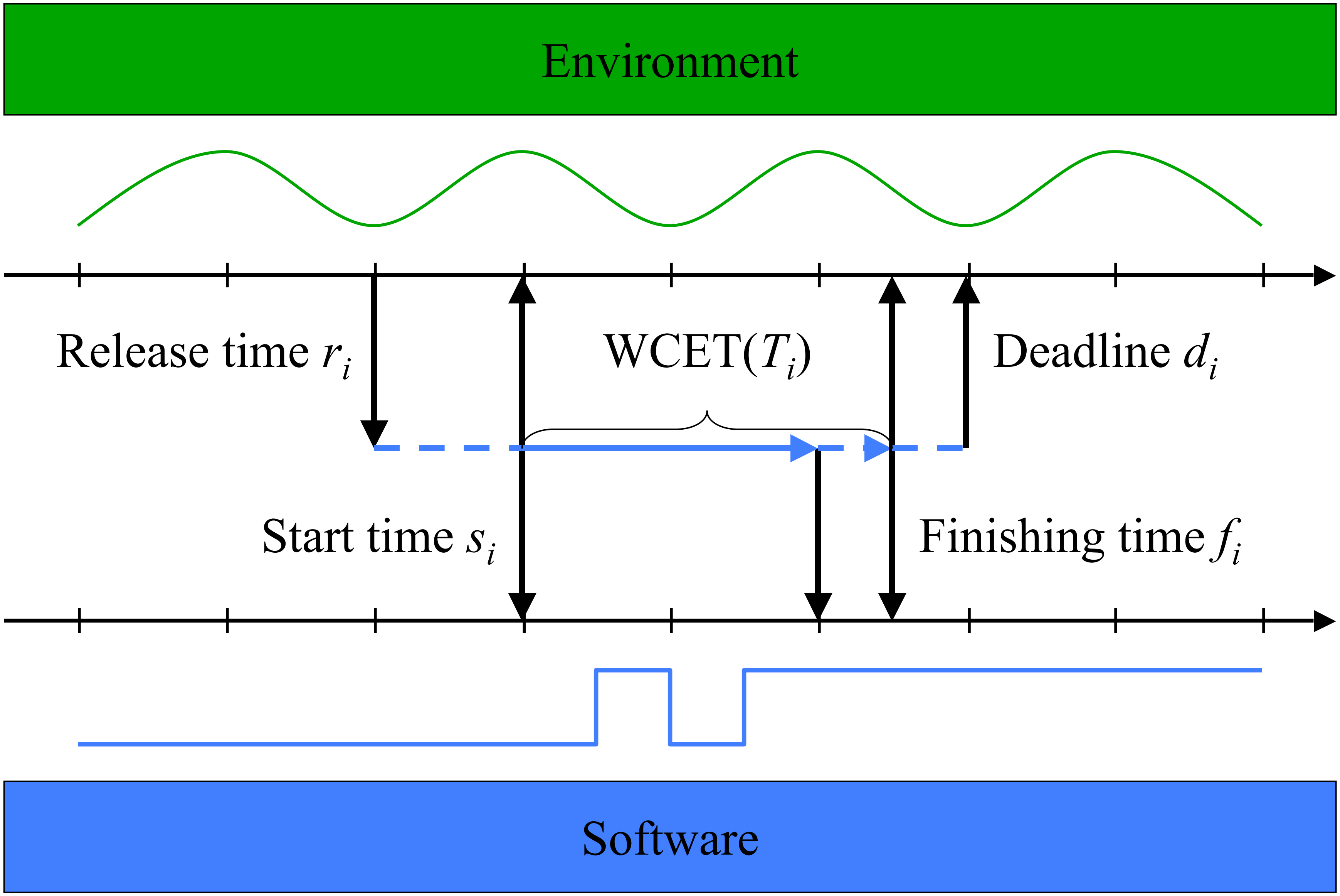


Environment
Release time r_i
Computation time C_i
Deadline d_i
Start time s_i
Finishing time f_i
Software

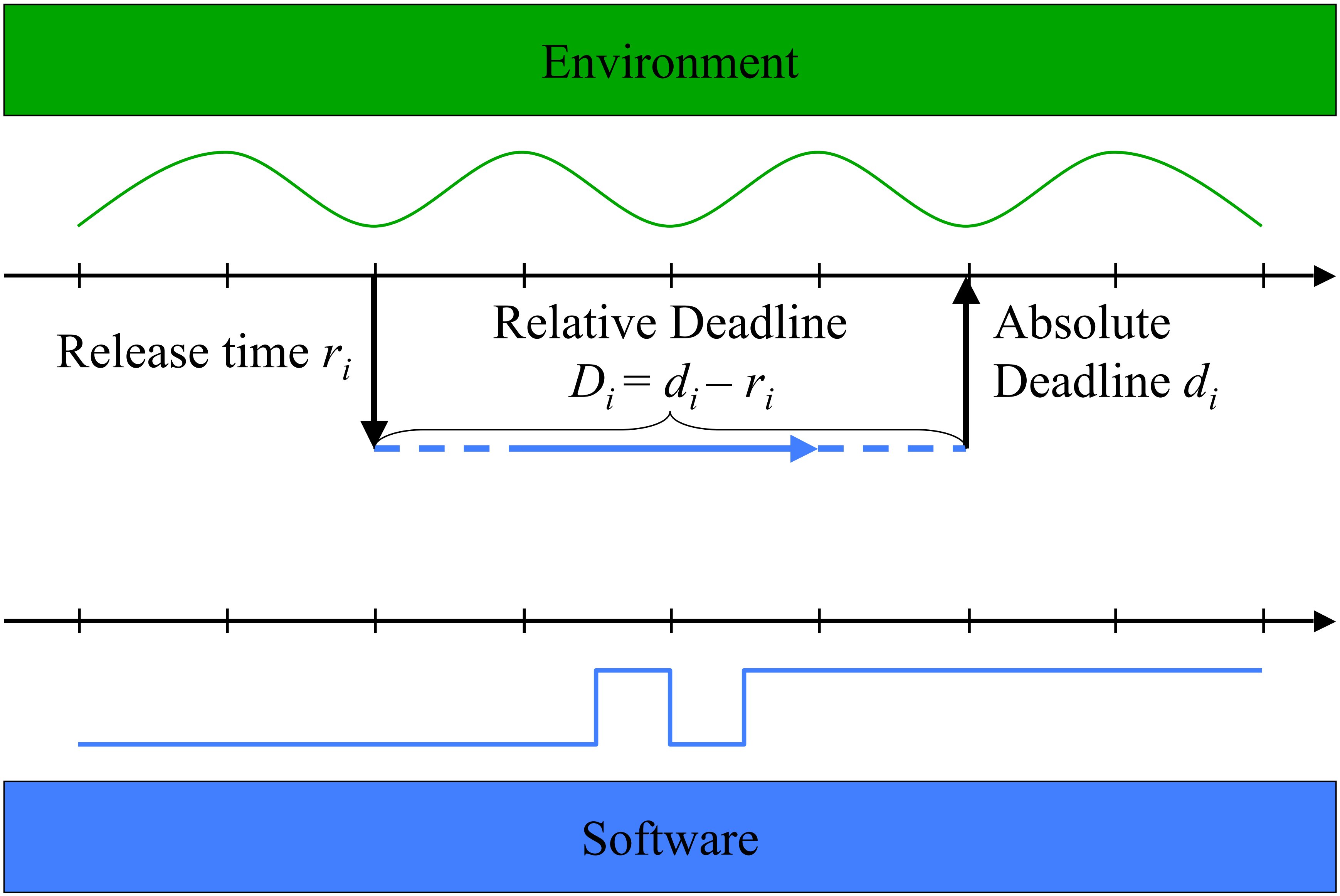
Preemption


Environment
Release time r_i
Computation time in RT
Deadline d_i
Start time s_i
Finishing time f_i
Software

Worst-Case Execution Time: WCET(T_i)



Relative Deadline D_i



Some Vocabulary for a Task T_i

- *Lateness*: $L_i = f_i - d_i$ is the delay of T_i's completion with respect to its deadline; negative L_i mean early completion
- *Laxity* (*Slack time*): $X_i = D_i - C_i$ is the maximum time T_i can be delayed on its start to complete within its deadline

Triggering a Task T_i

- *Periodically*: A *periodic task* T_i is a task with a-priori known release times regularly activated at a constant rate P_i
  - The first release time r_i is called the *phase* ϕ_I
  - The release time of the n-th instance is given by $r_i + (n - 1)\, P_i$
  - P_i is called the *period* of T_i

- *Sporadically*: A *sporadic task* T_i is a task with a minimum (*interarrival*) time between any two release times

- *Aperiodically*: An *aperiodic task* T_i is a task without any constraints on the release times

Definition: Schedule

- A *schedule* for a set T of tasks and a set S of *shared resources* is a function that maps a shared resource $s \in S$
  for any given (discrete) time instant to
  a possibly empty subset of T (Non-Determinism)

- A *feasible* schedule is a schedule in which each task can complete within its deadline

Schedulability Test vs. Scheduling Algorithm

- A *schedulability test* determines the existence of a feasible schedule for a given set of tasks and shared resources
- A schedulability test can be an *exact*, *sufficient*, or *necessary* condition for the existence of a feasible schedule

---

- A *scheduling algorithm* computes a (possibly infeasible) schedule
- A scheduling algorithm is called *optimal* with respect to a *cost function* if it minimizes that cost function
- A scheduling algorithm is called *optimal* with respect to *feasibility* if it always computes a feasible schedule provided that schedule exists

Earliest Due Date (EDD)

- The schedulability test for the *earliest due date* algorithm holds for a given set of n tasks, if:
  - $\forall i \in \{1,\ldots,n\} \,.\, f_i \leq d_i$ where $f_i = \sum_{k=1}^{i} C_k$

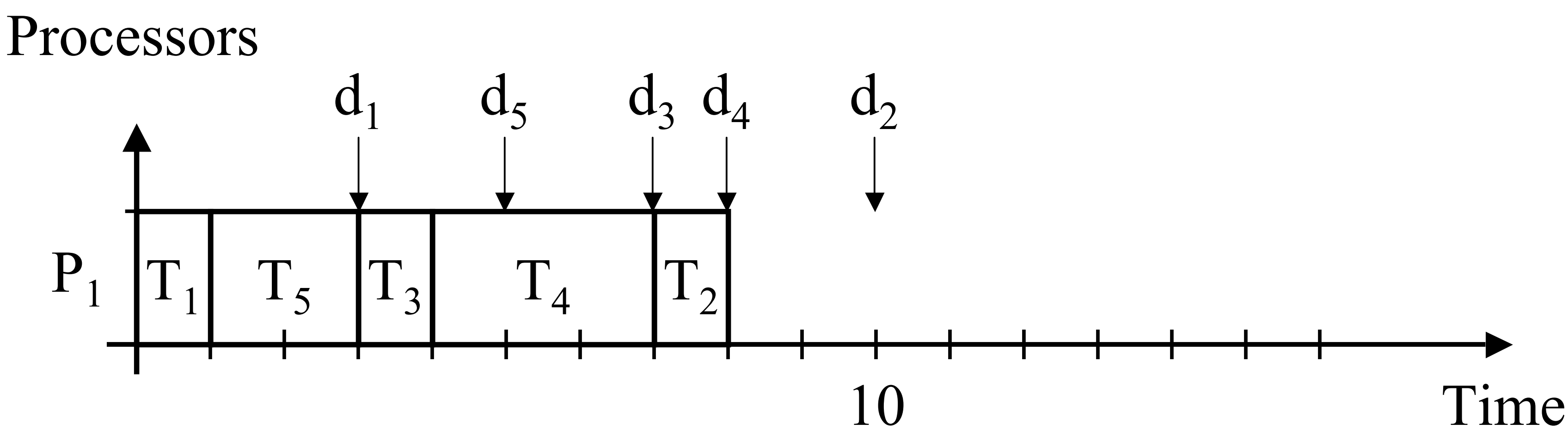
- The test is *exact*

- The *earliest due date* algorithm executes all tasks in a given set of n tasks in the order of non-decreasing deadlines

EDD Example

	T_1	T_2	T_3	T_4	T_5
C_i	1	1	1	3	2
d_i	3	10	7	8	5

Buttazzo97



Assume, then Guarantee

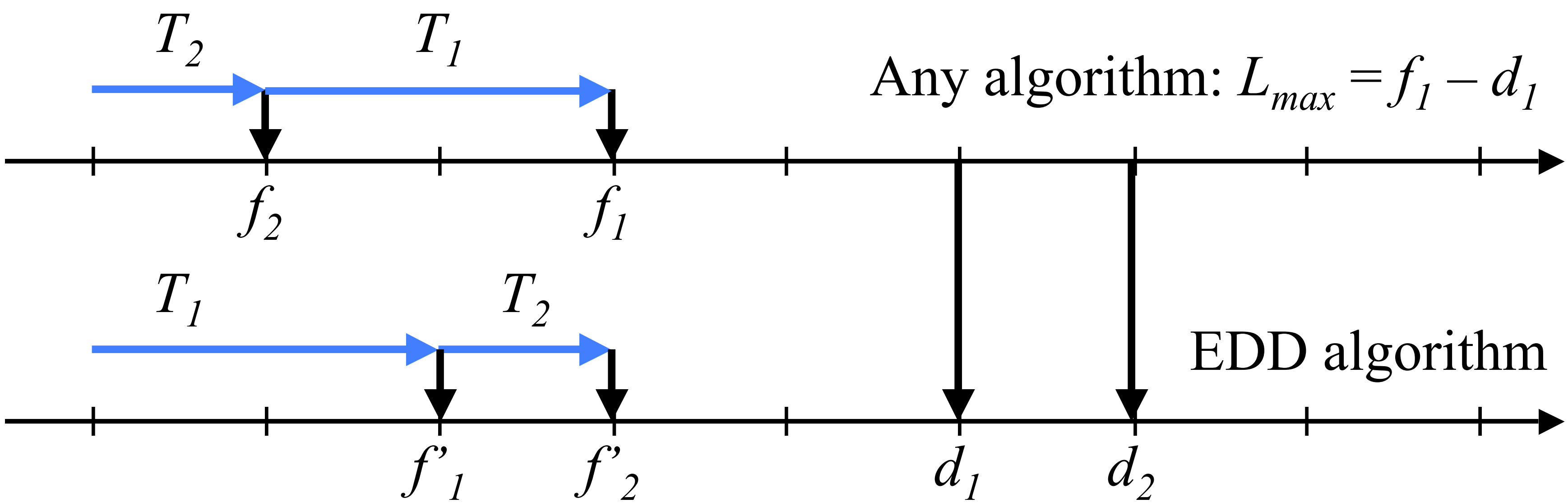
- *Resource* assumptions:
  - single processor
  - no administrative overhead

- *Task* assumptions:
  - *independent*, i.e., no *precedence* constraints
  - release times are equal for all tasks
  - WCET(T_i) = C_i given
  - absolute deadlines given

- *Optimality* guarantee:
  - EDD is optimal wrt. feasibility
  - EDD is optimal wrt. maximum lateness

Proof

- *Interchange argument*:
  In a non-EDD schedule $\exists T_1, T_2$ with $d_1 \leq d_2$
  but T_2 executes before T_1



- Exchanging does not increase maximum lateness
- There are only finitely many transpositions

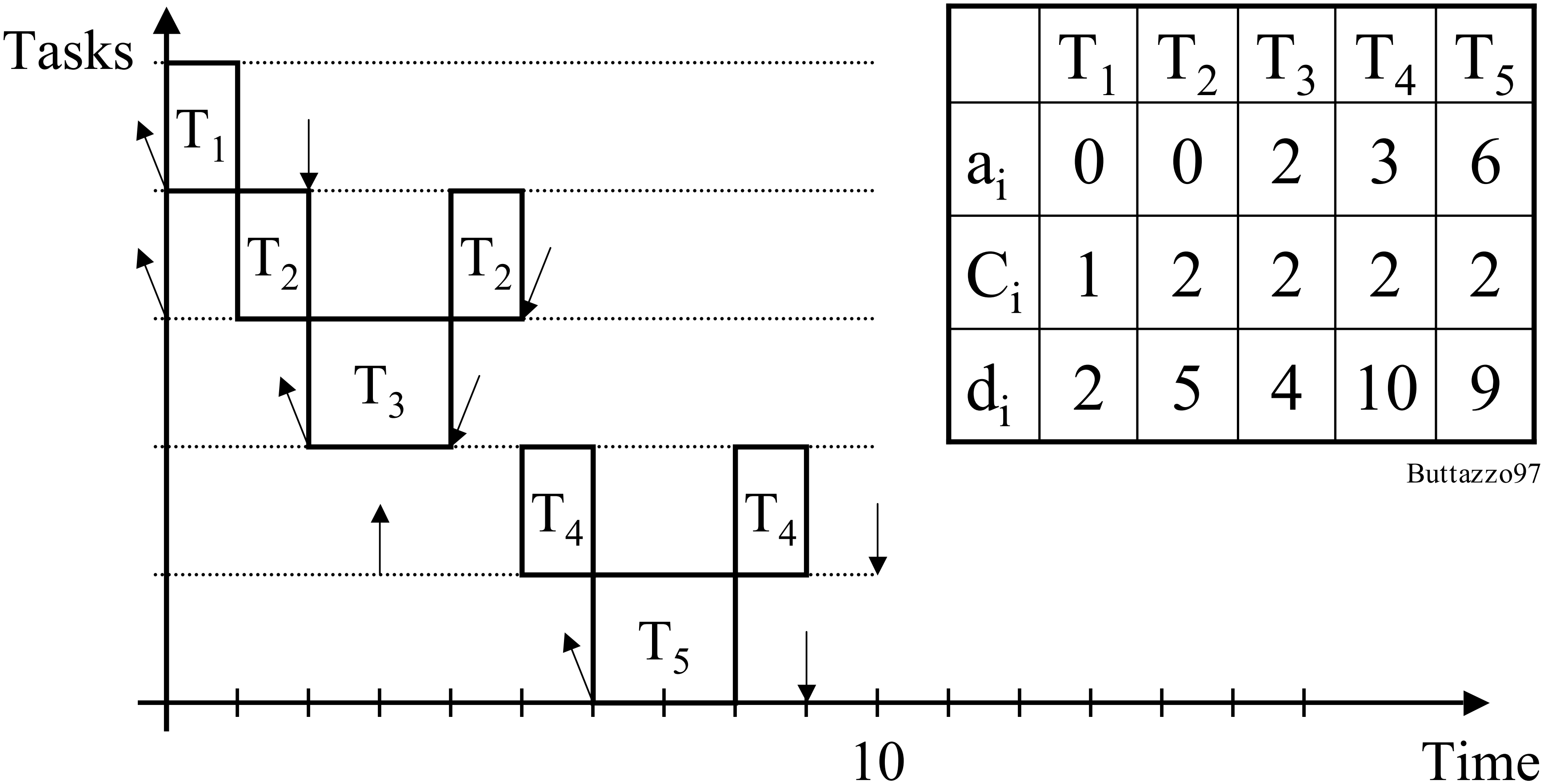
Earliest Deadline First (EDF)

- The schedulability test for the *earliest deadline first* algorithm holds for a given set of n tasks, if:
  - At any instant t where a task is released $\forall \mathrm{i} \in \{1,\ldots,n\}.\, f_i \leq d_i$ where $f_i = \sum_{k=1}^{i} c_k(t)$ and $c_k(t)$ is the remaining WCET of T_i at t

- The test is *exact*

- The *earliest deadline first* algorithm executes at any instant, given a set of n tasks, the task with the earliest deadline: dynamic priority assignment algorithm

EDF Example



	T_1	T_2	T_3	T_4	T_5
a_i	0	0	2	3	6
C_i	1	2	2	2	2
d_i	2	5	4	10	9

Buttazzo97

Assume, then Guarantee for EDF

- *Resource* assumptions:
  - single processor
  - no administrative overhead

- *Task* assumptions:
  - *preemptive*
  - *independent*, i.e., no *precedence* constraints
  - release times given
  - WCET(T_i) = C_i given
  - relative deadlines given

- *Optimality* guarantee:
  - EDF is optimal wrt. feasibility
  - EDF is optimal wrt. maximum lateness

Proof for EDF

- Based on the interchange argument for EDD:
  - Exchange time slices instead of tasks because of possible preemptions

Rate Monotonic Analysis (RMA)

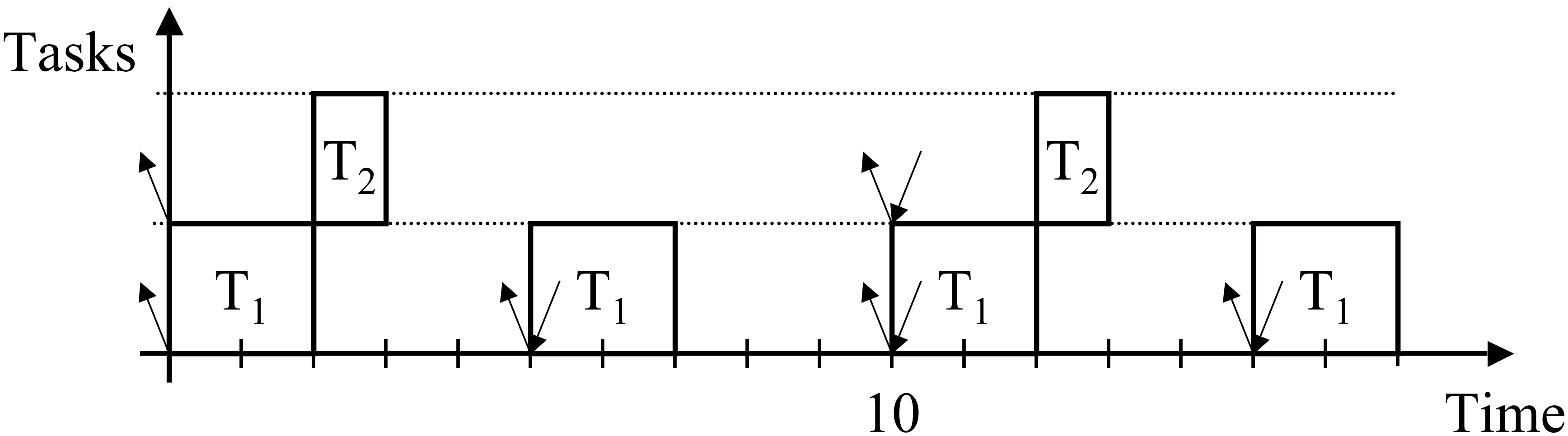
- The schedulability test for the *rate monotonic scheduling* algorithm holds for a given set of *n* tasks, if:
  - $\sum^{n}_{i=1} C_i / P_i < n * (2^{1/n} - 1)$

- The test is a utilization-based schedulability test
- The test is only *sufficient*

- The *rate monotonic scheduling* algorithm assigns a fixed priority to each task in a set of *n* tasks proportional to the task's frequency: fixed-priority assignment algorithm

RMA Example

	T_1	T_2
C_i	2	1
p_i	5	10



Assume, then Guarantee for RMA

- *Resource* assumptions:
  - single processor
  - no administrative overhead

- *Task* assumptions:
  - *preemptive*
  - *independent*, i.e., no *precedence* constraints
  - *periodic*
  - WCET(T_i) = C_i given
  - deadlines equal to periods

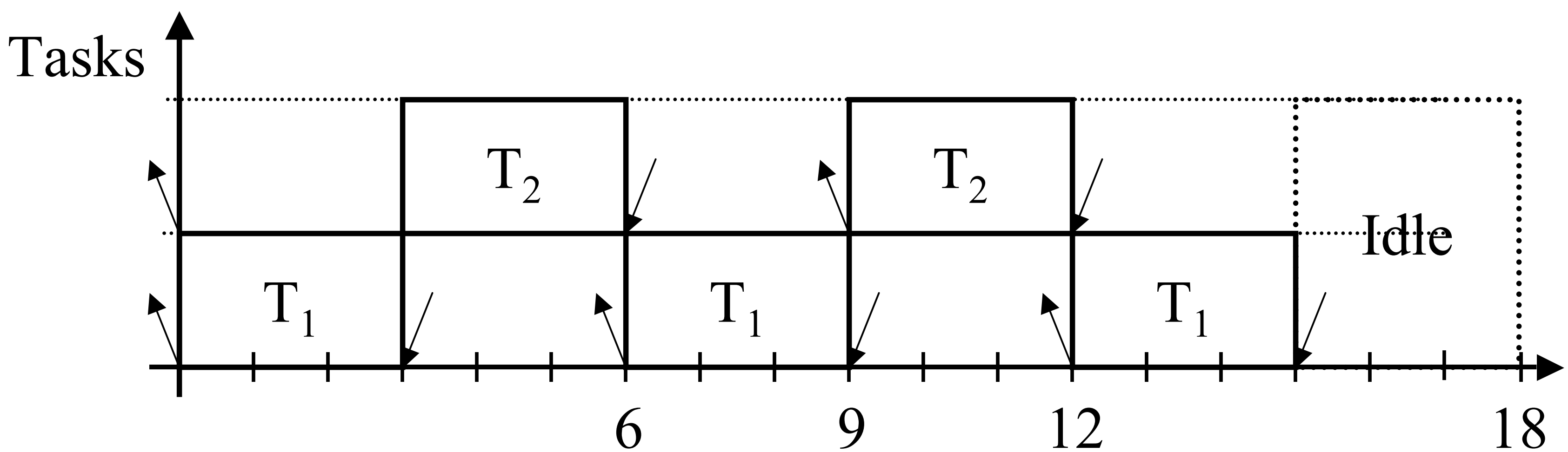
- *Optimality* guarantee:
  - RMA is optimal wrt. *fixed-priority* feasibility

Utilization-Based Schedulability Tests

- EDF:
  - $\sum^{n}_{i=1} C_i / P_i \leq 1$
  - exact, but cannot be extended to more complex task models
- RMA:
  - $\sum^{n}_{i=1} C_i / P_i < n * (2^{1/n} - 1)$
  - sufficient but not necessary (for non-harmonic task sets)

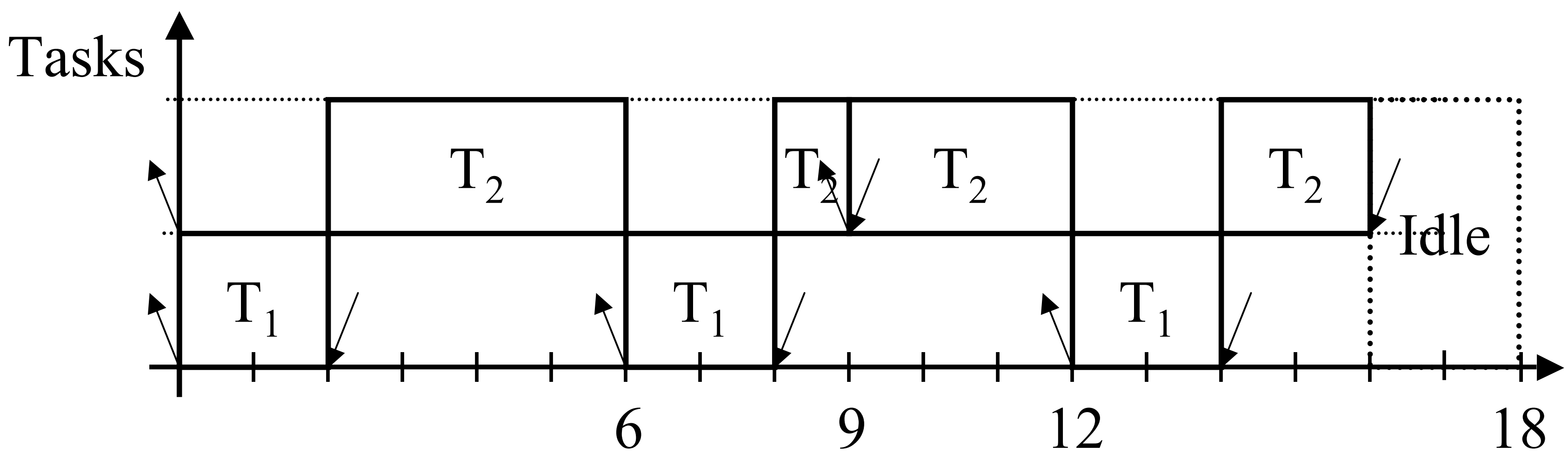
RMA: 84% Utilization (Test: < 82.8%)

	T_1	T_2
C_i	3	3
p_i	6	9



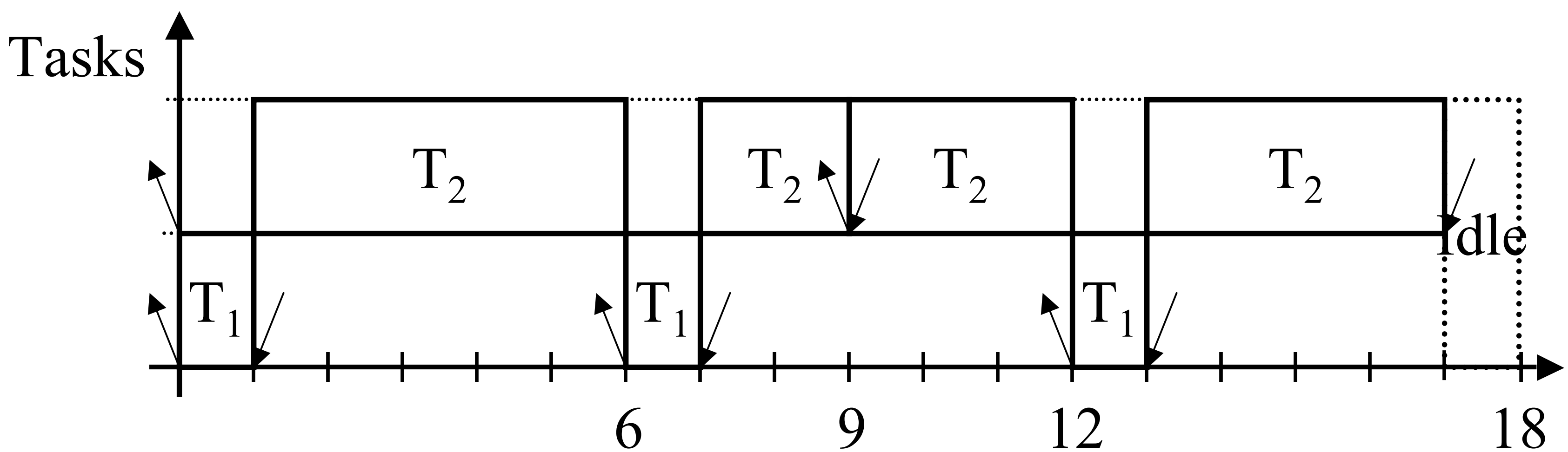
RMA: 89% Utilization

	T_1	T_2
C_i	2	5
p_i	6	9



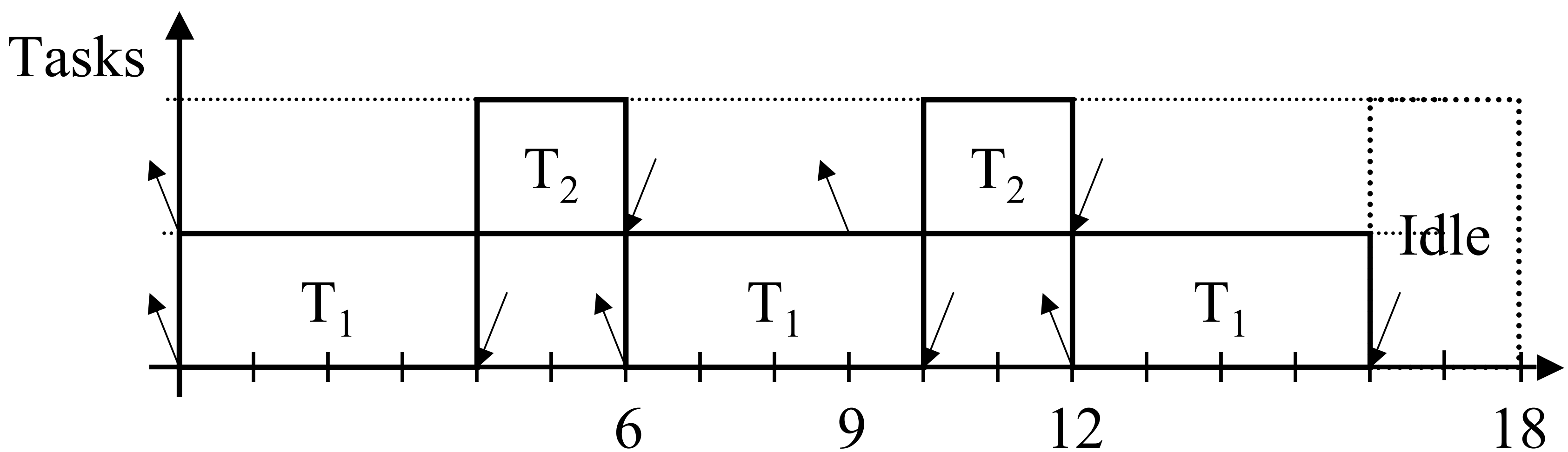
RMA: 95% Utilization

	T_1	T_2
C_i	1	7
p_i	6	9



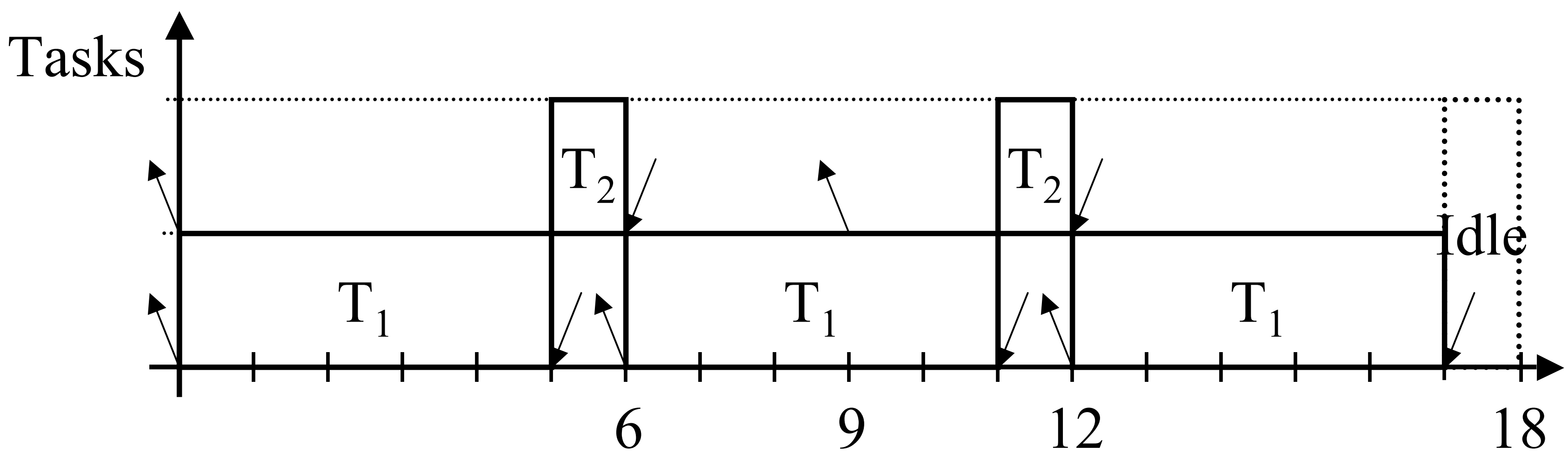
RMA: 89% Utilization

	T_1	T_2
C_i	4	2
p_i	6	9



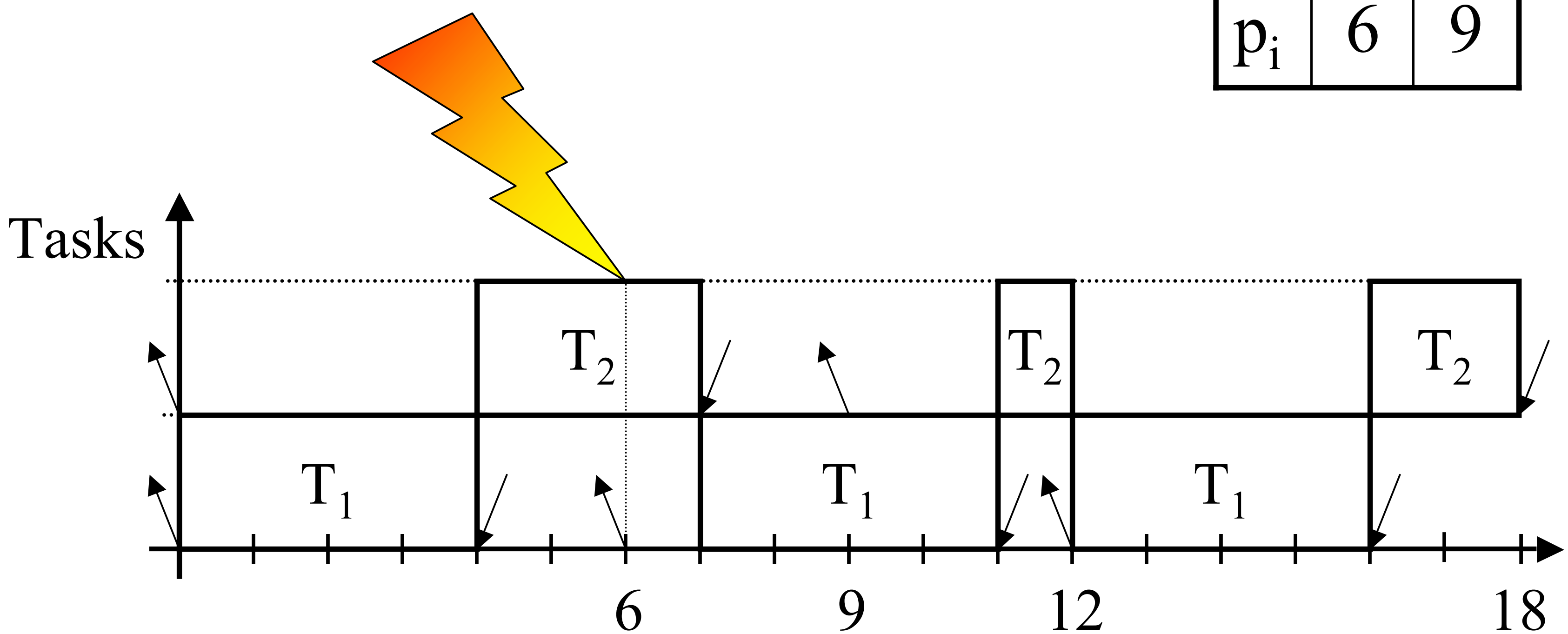
RMA: 95% Utilization

	T_1	T_2
C_i	5	1
p_i	6	9



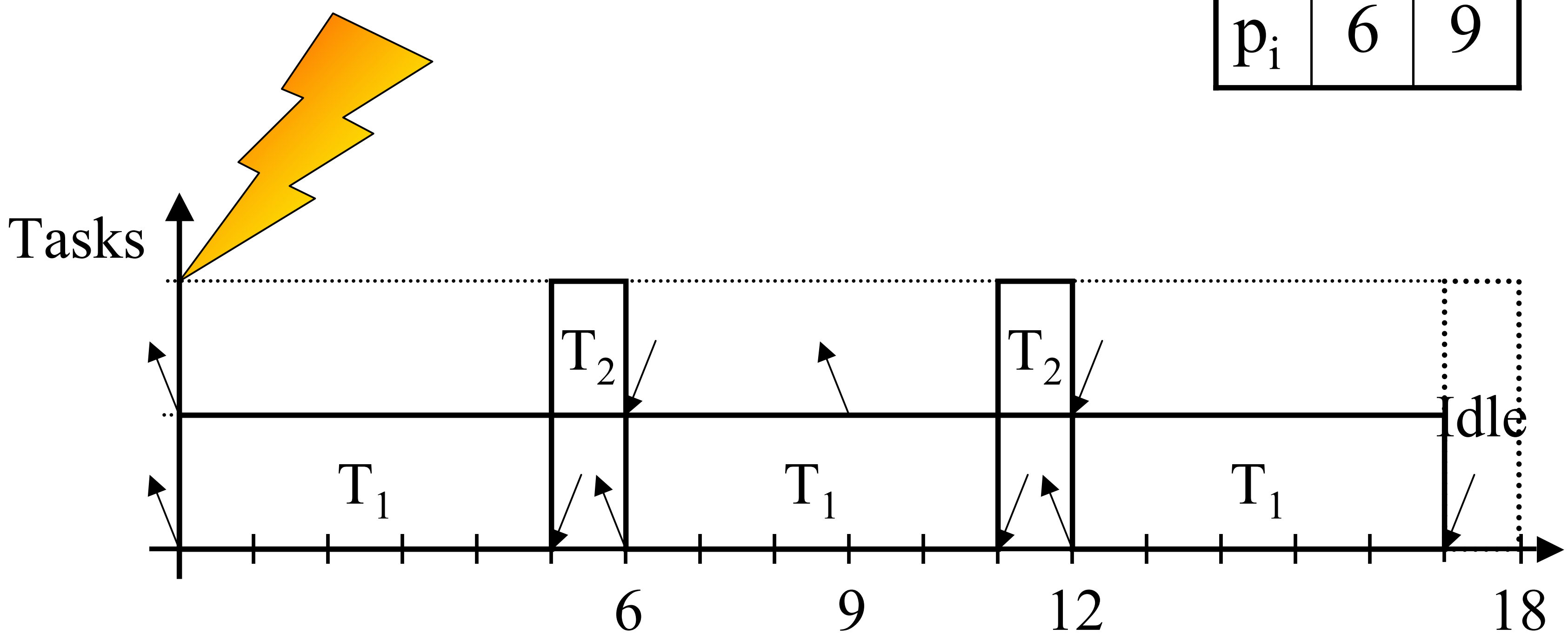
EDF: 100% Utilization

	T_1	T_2
C_i	4	3
p_i	6	9



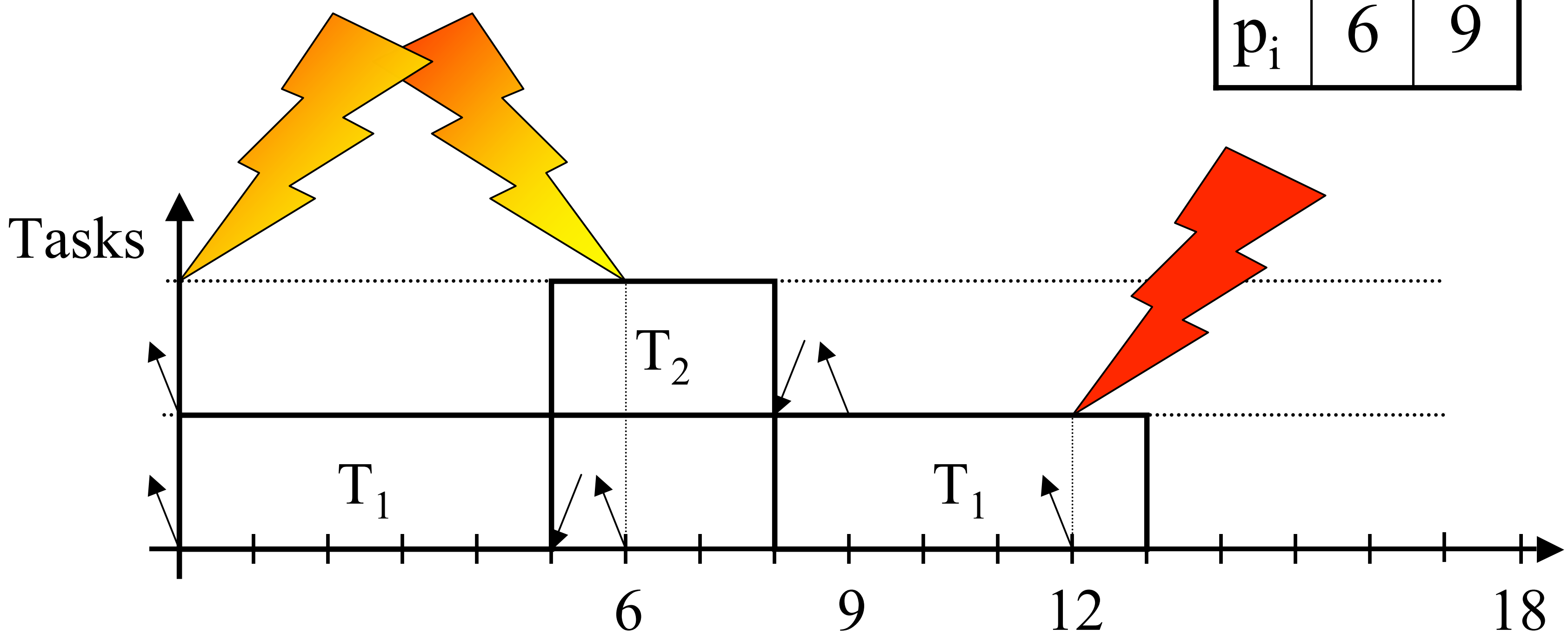
RMA: The Critical Instant

	T_1	T_2
C_i	5	1
p_i	6	9



EDF: Response Times

	T_1	T_2
C_i	5	3
p_i	6	9



Response Time Analysis

- *Response time*: $R_i = f_i - r_i$ is the time it takes T_i to complete
- The *critical instant* of a task T is the time instant at which a release of T produces the largest response time
- *Response time analysis* is done in two stages:
  - Compute the worst-case response times for all tasks T_i: $R_i = C_i - I_i$ where I_i is the maximum interference T_i can experience in any time interval $[t, t + R_i)$
  - Check if the worst-case response times are shorter than the deadlines

Response Time Analysis

- Maximum interference occurs when all higher-priority tasks are released at the same time as T_i
- $Number\_of\_releases = \lceil R_i / P_j \rceil$
  where T_j is a higher-priority task than T_i
- $Maximum\_interference = \lceil R_i / P_j \rceil * C_j$
- $I_i = \sum_{j \in \mathrm{hp}(i)} \lceil R_i / P_j \rceil * C_j$
  where hp(i) is the set of higher-priority tasks than T_i
- Fixed-point computation: $R_i = C_i + \sum_{j \in \mathrm{hp}(i)} \lceil R_i / P_j \rceil * C_j$

Busy Period

- Compute recurrence relation: $w_i^{n+1} = C_i + \sum_{j \in \mathrm{hp}(i)} \lceil w_i^n / P_j \rceil * C_j$
- Solution is found when $w_i^{n+1} = w_i^n$
- From the time a task T_i is released until T_i completes the processor is said to execute (continuously) a p_i–*busy period* where p_i is the priority of T_i
- Time window starts with $w_i^1 = C_i + \sum_{j \in \mathrm{hp}(i)} C_j$ and may have to be pushed out further