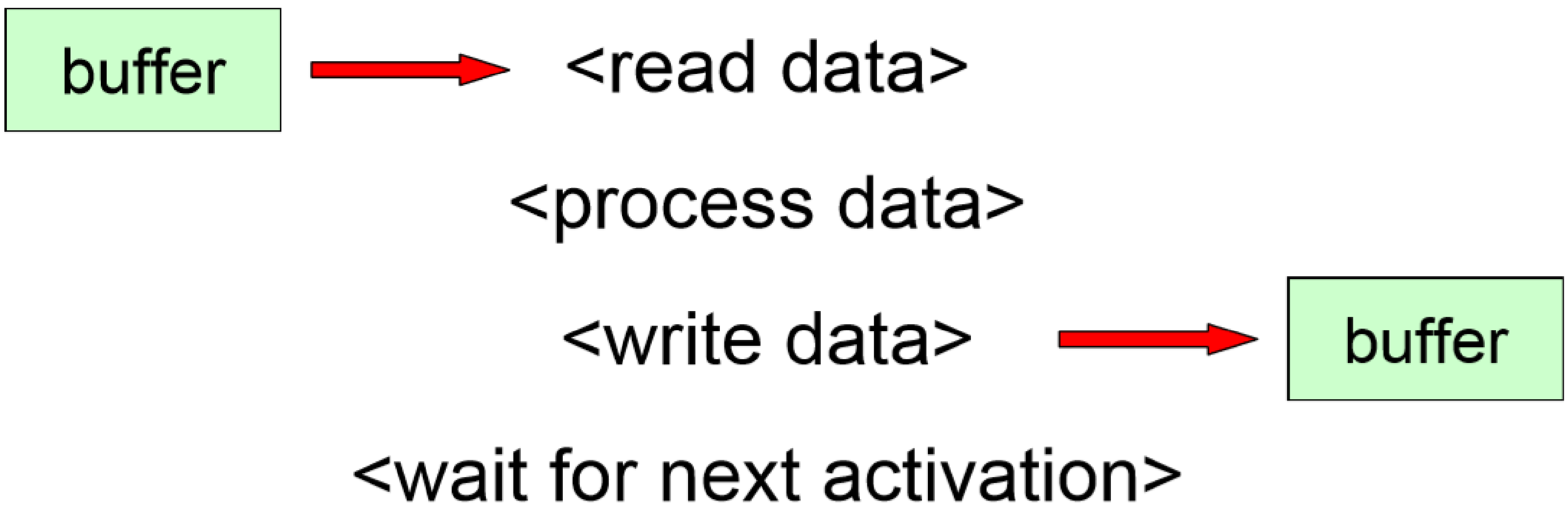
ИНФОРМАЦИОННО-УПРАВЛЯЮЩИЕ СИСТЕМЫ РЕАЛЬНОГО ВРЕМЕНИ

Лекция 3:
Динамическое планирование вычислений и оценка планируемости – 2

Кафедра АСВК,
Лаборатория Вычислительных Комплексов
Балашов В.В.

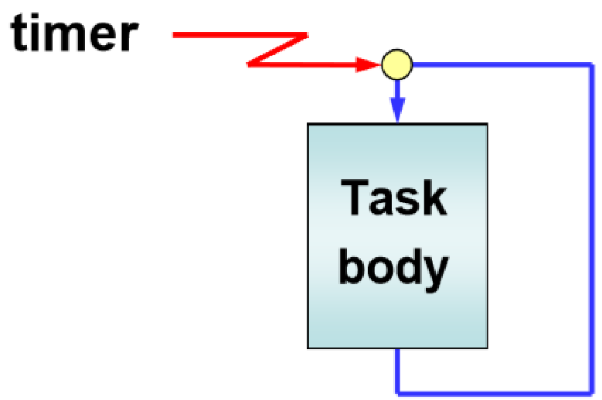
Typical task structure



Activation modes

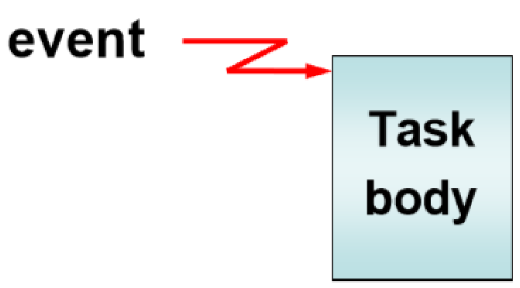
Periodic task (time driven)

A task is automatically activated by the kernel at regular time intervals



Aperiodic task (event driven)

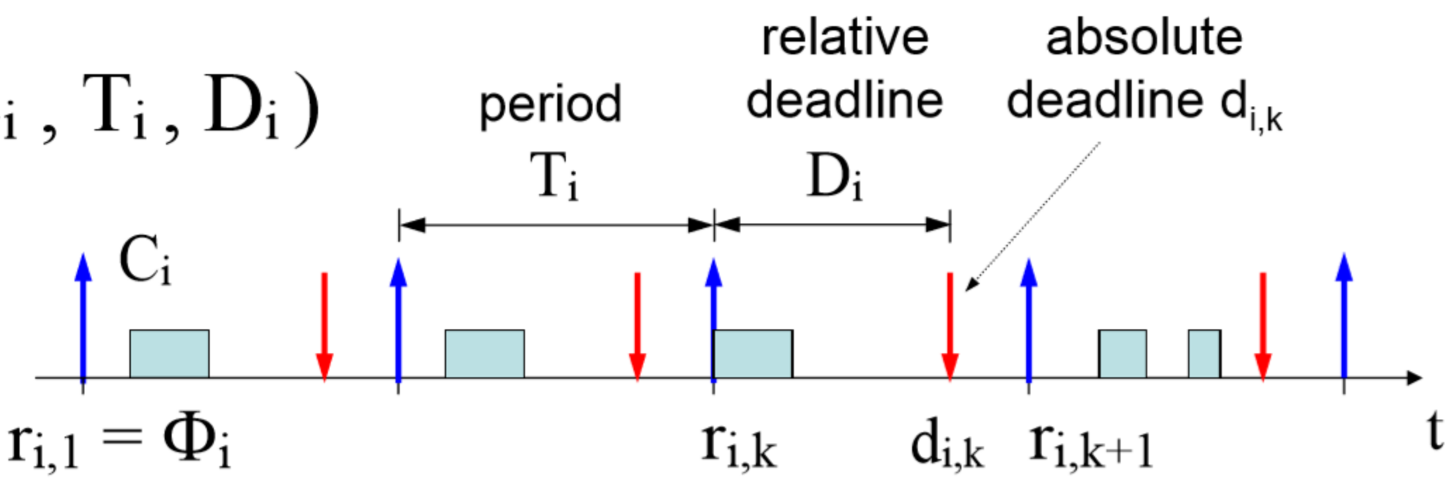
A task is activated upon the arrival of an event (interrupt or explicit activation)



Periodic Task Scheduling

We have ***n*** periodic tasks: $\{\tau_1, \tau_2 \ldots \tau_n\}$

$\tau_i (C_i, T_i, D_i)$



Goal

- Execute all tasks within their deadlines
- Verify feasibility before runtime

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

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

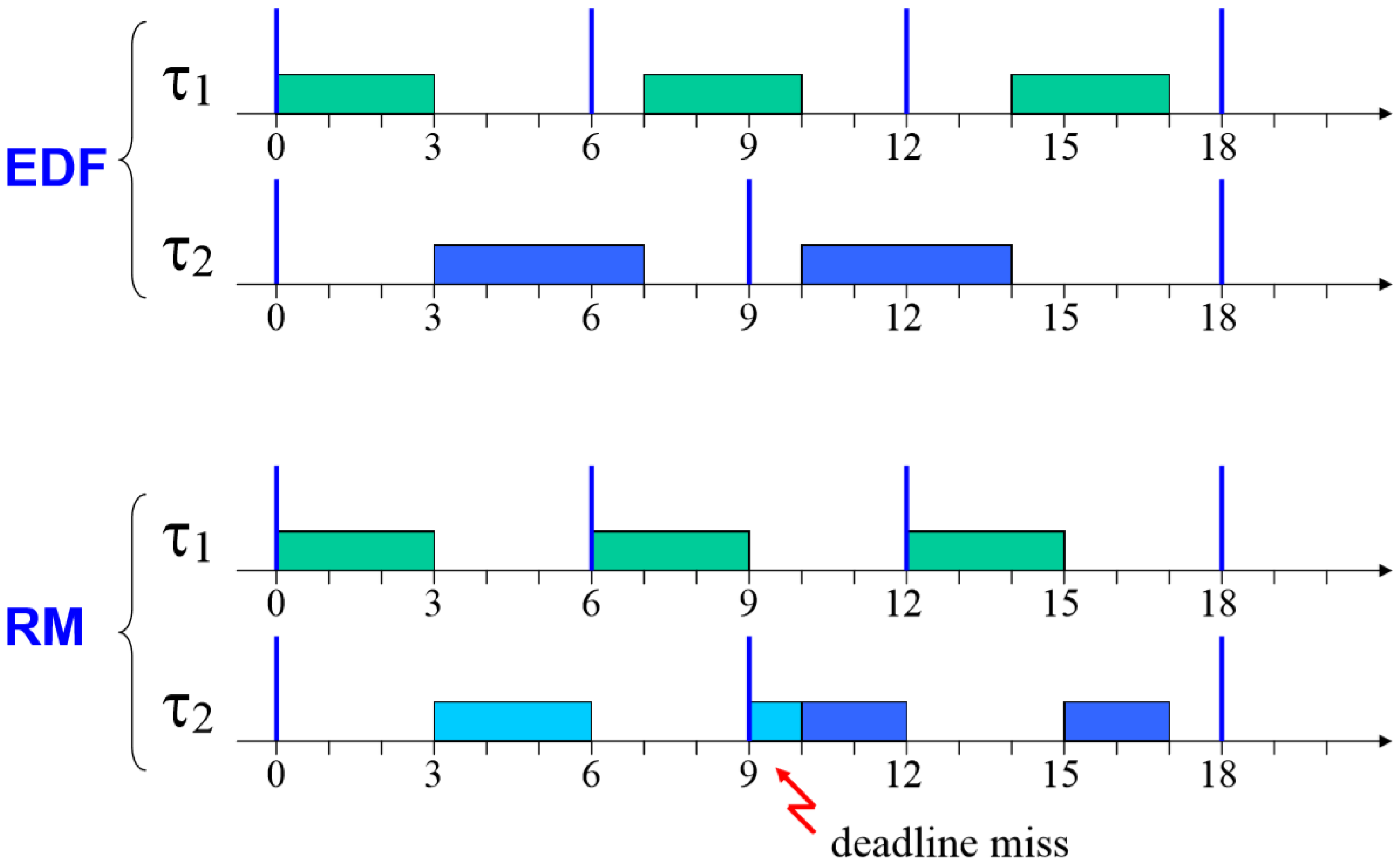
Fixed-Priority Scheduling (FPS)

- This is the most widely used approach
- Each task has a fixed, static, priority which is computer pre-run-time
- The runnable tasks are executed in the order determined by their priority
- In real-time systems, the "priority" of a task is derived from its temporal requirements, not its importance to the correct functioning of the system or its integrity

Earliest Deadline First (EDF)

- The runnable tasks are executed in the order determined by the absolute deadlines of the tasks
- The next task to run being the one with the shortest (nearest) deadline
- Although it is usual to know the relative deadlines of each task (e.g. 25ms after release), the absolute deadlines are computed at run time and hence the scheme is described as dynamic

EDF vs. RM Schedule



Response Time Analysis

[Audsley, 1990]

- For each task τ_i compute the interference due to higher priority tasks:

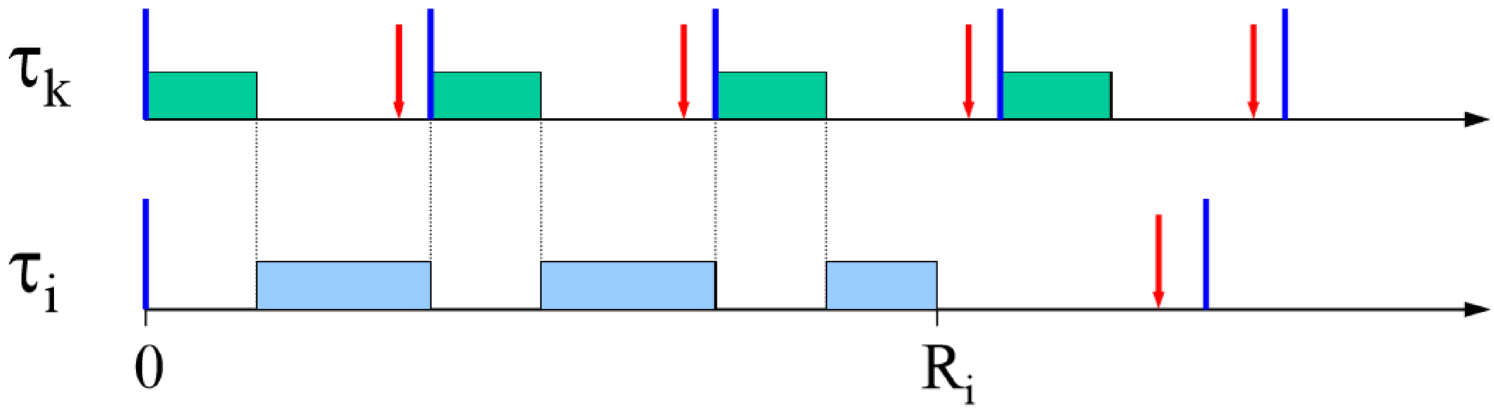
$$I_i = \sum_{D_k < D_i} C_k$$

- Compute its response time as

$$R_i = C_i + I_i$$

- Verify if $R_i \leq D_i$

Computing the interference



Interference of τ_k on τ_i in the interval $[0, R_i]$:

$$I_{ik} = \left\lceil \frac{R_i}{T_k} \right\rceil C_k$$

Interference of high priority tasks on τ_i:

$$I_i = \sum_{k=1}^{i-1} \left\lceil \frac{R_i}{T_k} \right\rceil C_k$$

Response Time Equation

$$R_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

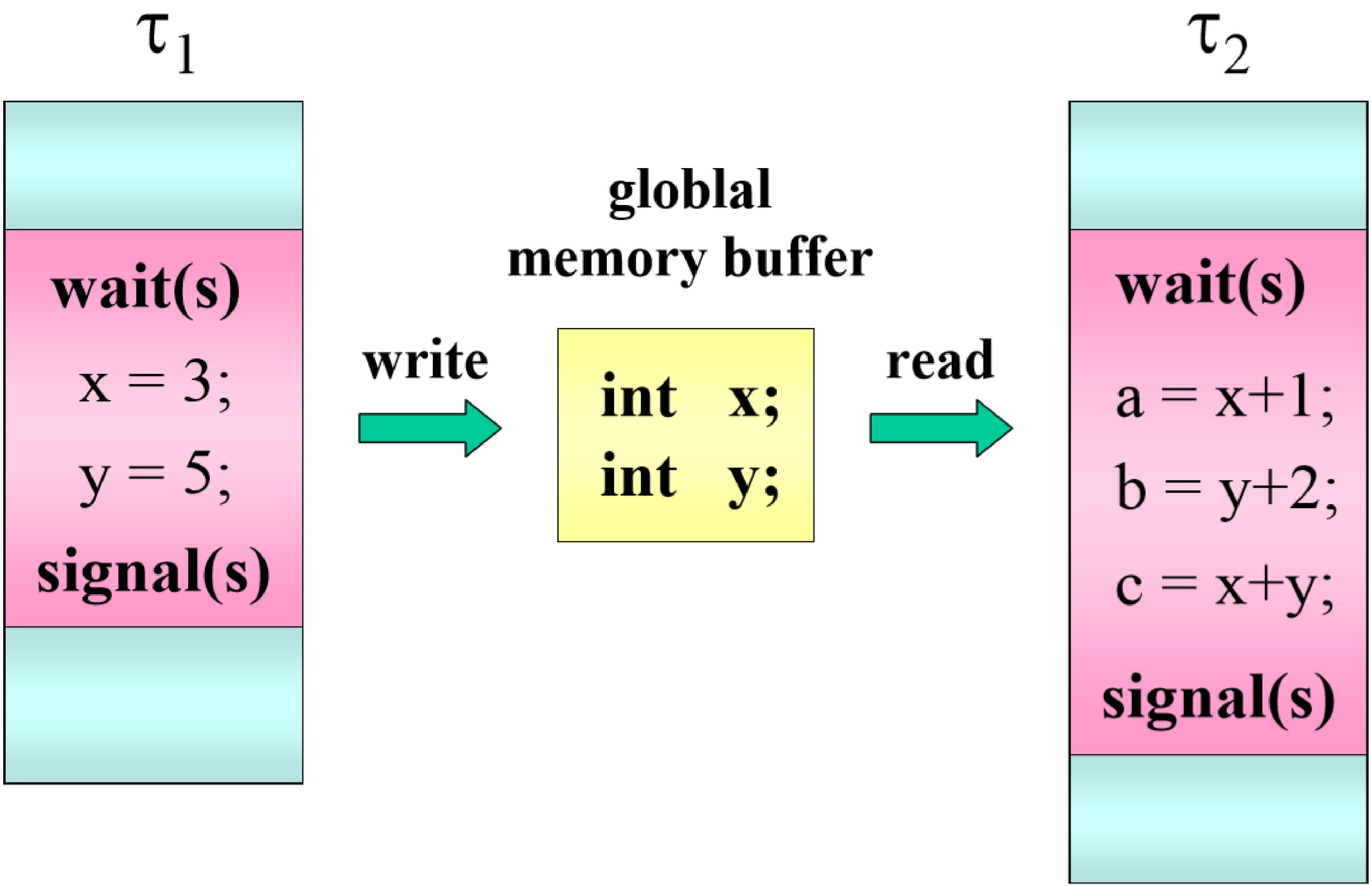
Where $hp(i)$ is the set of tasks with priority higher than task i

Solve by forming a recurrence relationship:

$$w_i^{n+1} = C_i + \sum_{j \in hp(i)} \left\lceil \frac{w_i^n}{T_j} \right\rceil C_j$$

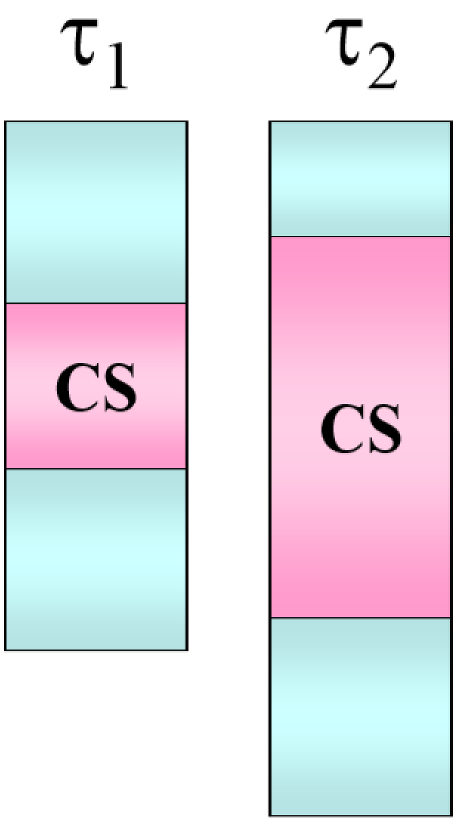
The set of values $w_i^0, w_i^1, w_i^2, \ldots, w_i^n, \ldots$ is monotonically non decreasing.
When $w_i^n = w_i^{n+1}$ the solution to the equation has been found; w_i^0 must not be greater that R_i (e.g. 0 or C_i)

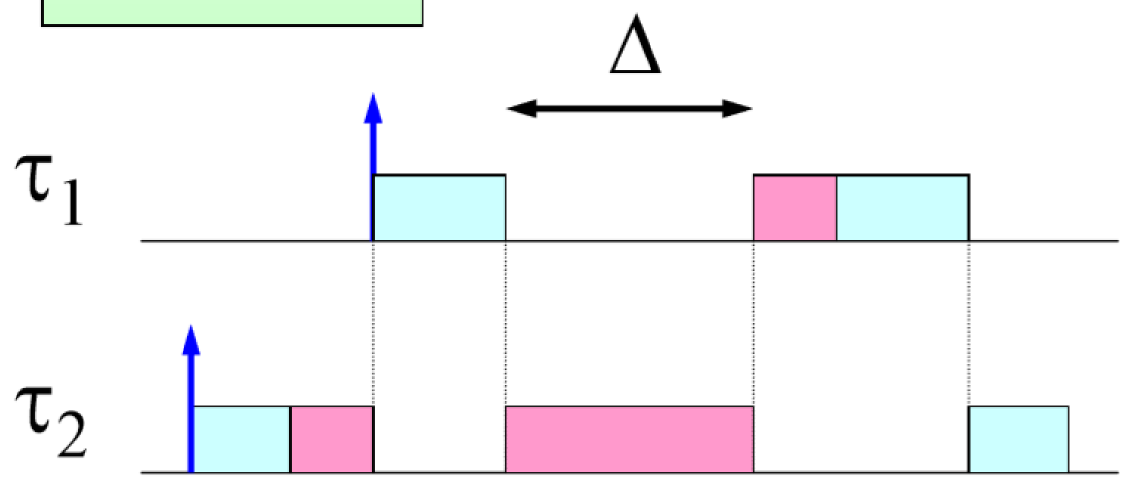
Critical sections



Blocking on a semaphore

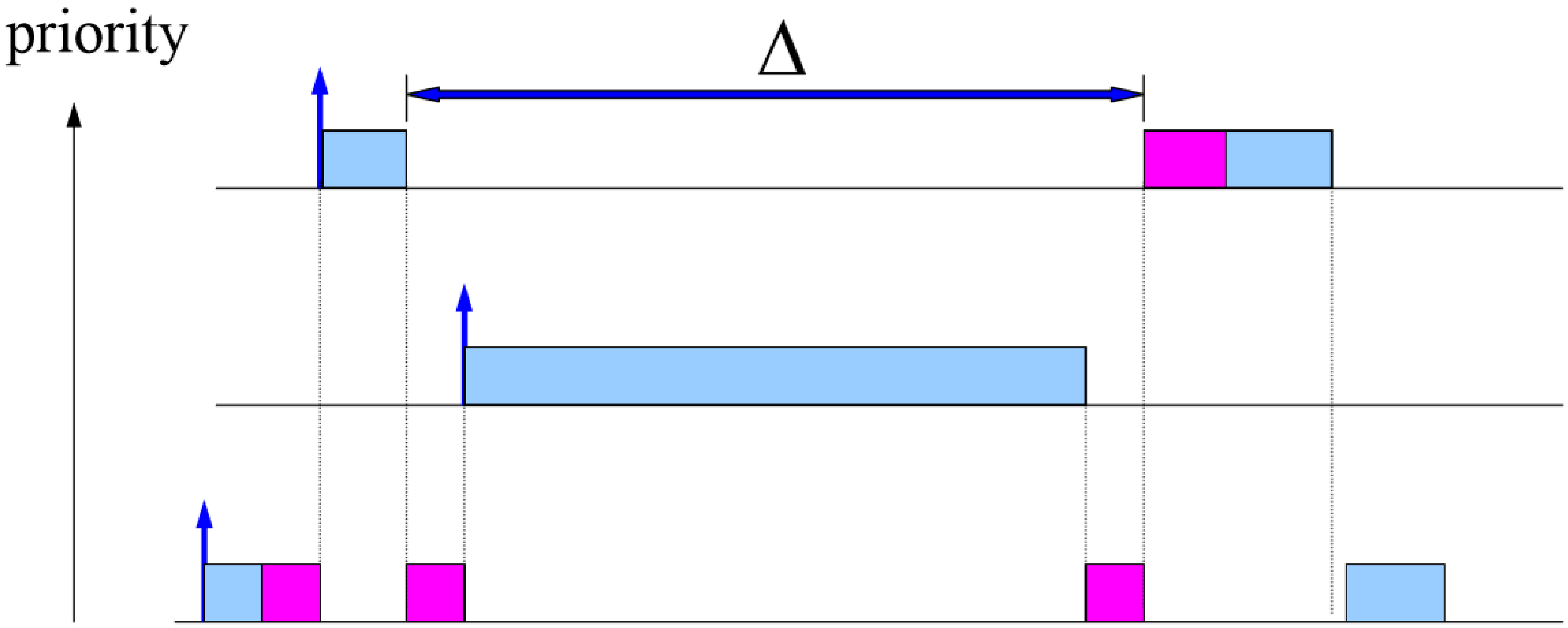
$p_1 > p_2$





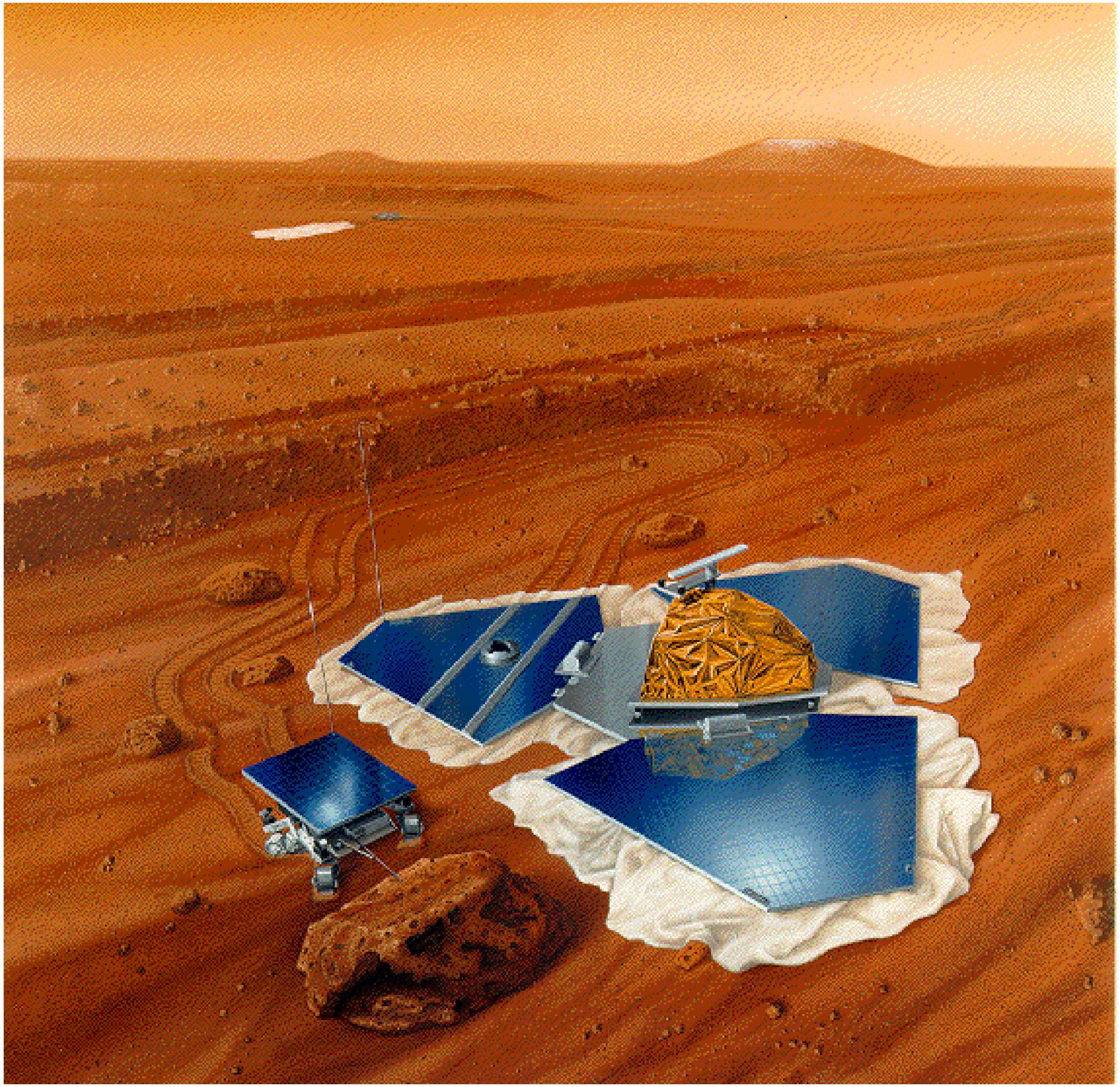
It seems that the maximum blocking time for τ_1 is equal to the length of the critical section of τ_2, but …

Priority Inversion



Occurs when a high priority task is blocked by a lower-priority task a for an unbounded interval of time.

Mars Pathfinder



The MARS Pathfinder problem

“VxWorks provides preemptive priority scheduling of threads. Tasks on the Pathfinder spacecraft were executed as threads with priorities that were assigned in the usual manner reflecting the relative urgency of these tasks.”

“Pathfinder contained an "information bus", which you can think of as a shared memory area used for passing information between different components of the spacecraft.”

- A bus management task ran frequently with high priority to move certain kinds of data in and out of the information bus. Access to the bus was synchronized with mutual exclusion locks (mutexes).”

The MARS Pathfinder problem

- The meteorological data gathering task ran as an infrequent, low priority thread, … When publishing its data, it would acquire a mutex, do writes to the bus, and release the mutex. ..
- The spacecraft also contained a communications task that ran with medium priority."

☞

High priority: retrieval of data from shared memory
Medium priority: communications task
Low priority: thread collecting meteorological data

The MARS Pathfinder problem

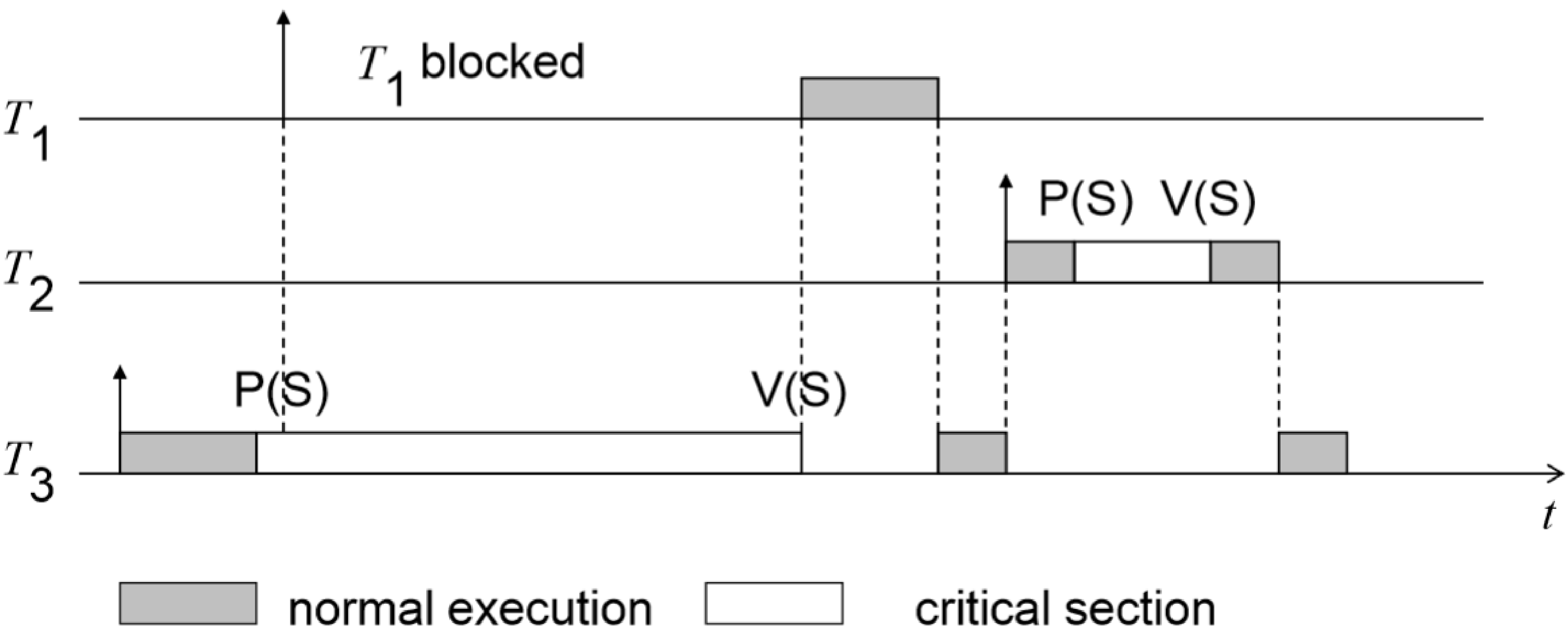
"… However, very infrequently it was possible for an interrupt to occur that caused the (medium priority) communications task to be scheduled during the short interval while the (high priority) information bus thread was blocked waiting for the (low priority) meteorological data thread.

In this case, the long-running communications task, having higher priority than the meteorological task, would prevent it from running, consequently preventing the blocked information bus task from running.

After some time had passed, a watchdog timer would go off, notice that the data bus task had not been executed for some time, conclude that something had gone drastically wrong, and initiate a total system reset."

Solutions

Disallow preemption during the execution of all critical sections. Simple, but creates unnecessary blocking as unrelated tasks may be blocked.

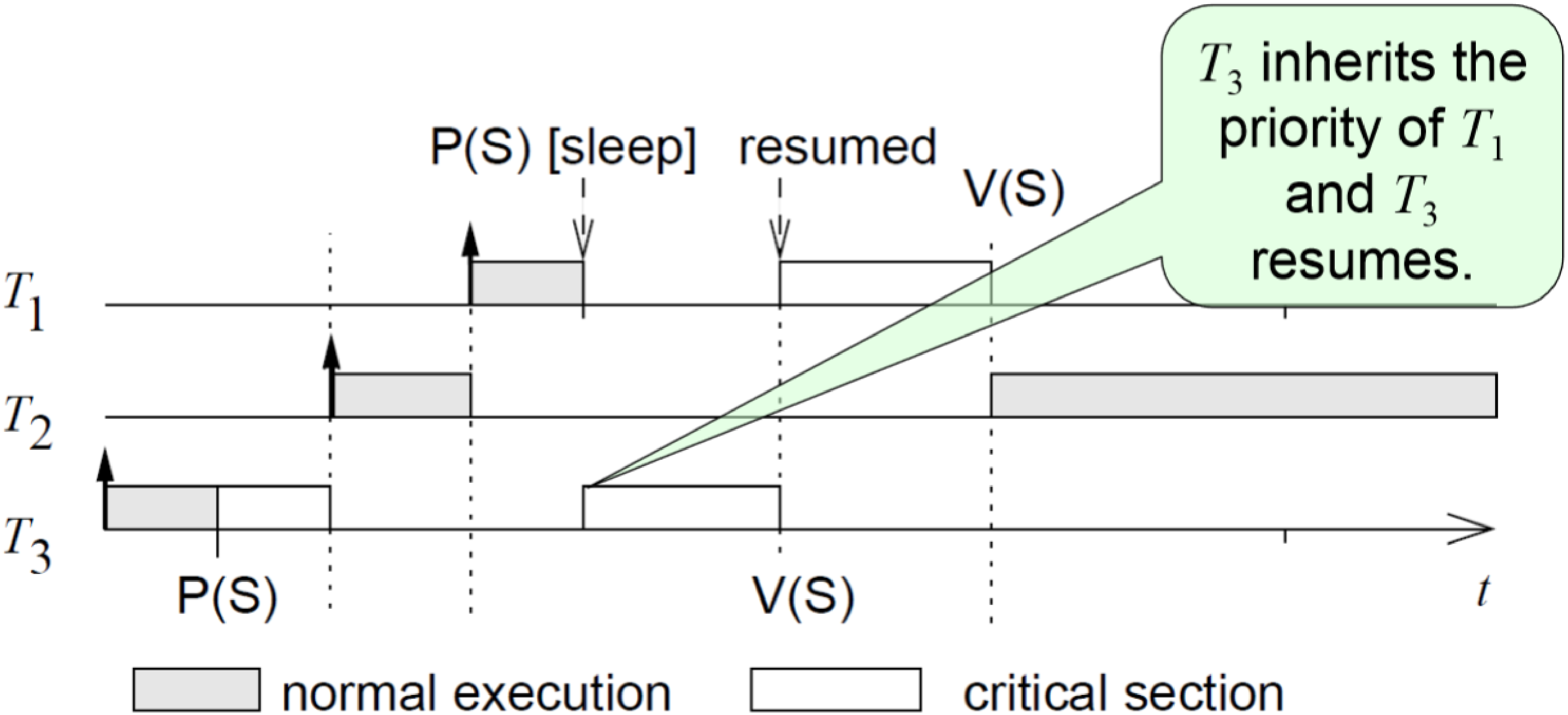


Coping with priority inversion: the priority inheritance protocol

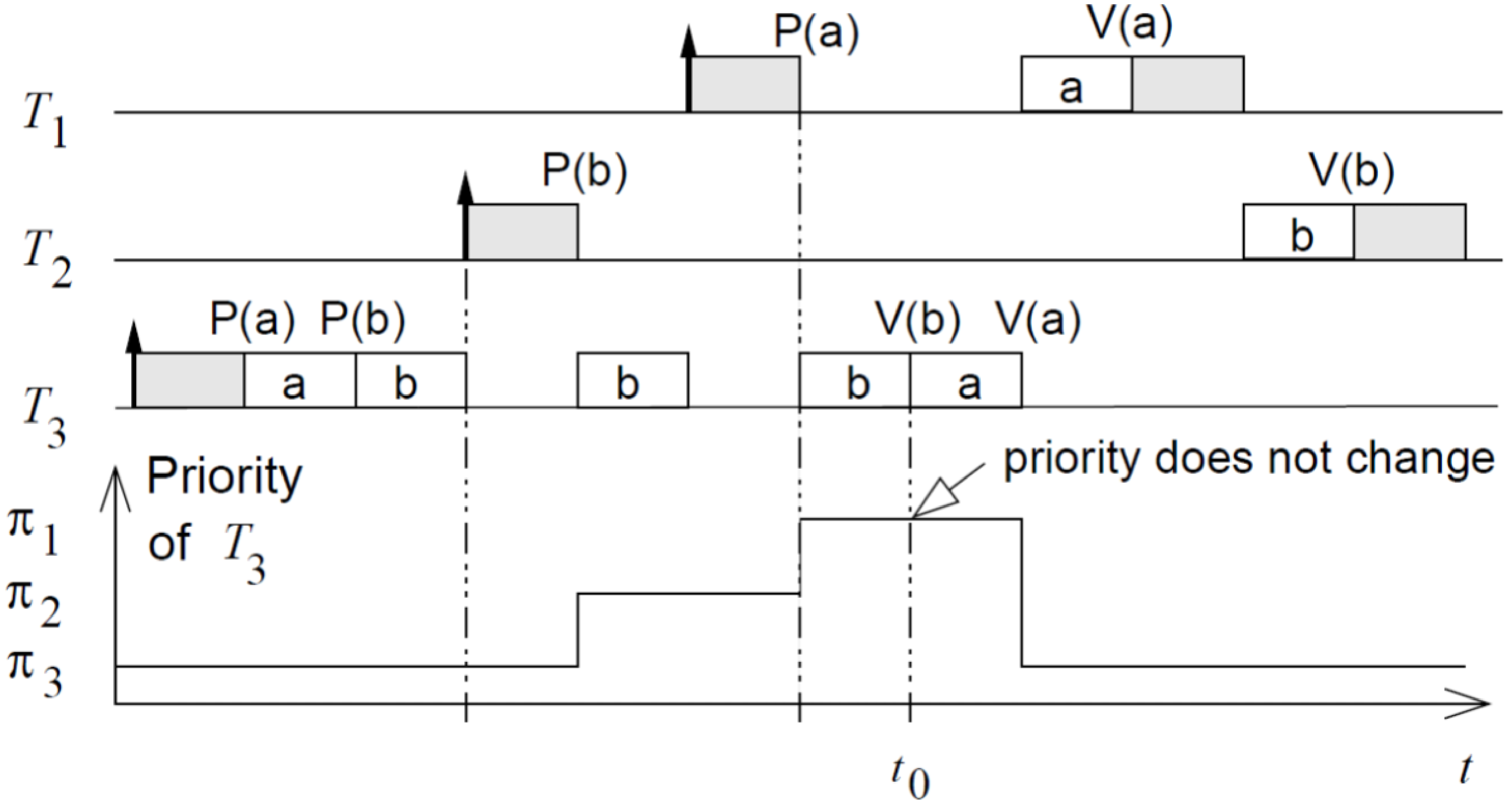
- Tasks are scheduled according to their active priorities. Tasks with the same priorities are scheduled FCFS.
- If task T_1 executes P(S) & exclusive access granted to T_2: T_1 will become blocked.
  If priority(T_2) < priority(T_1): T_2 inherits the priority of T_1.
  ☞ T_2 resumes.
  Rule: tasks inherit the highest priority of tasks blocked by it.
- When T_2 executes V(S), its priority is decreased to the highest priority of the tasks blocked by it.
  If no other task blocked by T_2: priority(T_2):= original value.
  Highest priority task so far blocked on S is resumed.
- Transitive: if T_2 blocks T_1 and T_1 blocks T_0, then T_2 inherits the priority of T_0.

Example

How would priority inheritance affect our example with 3 tasks?



Nested critical sections



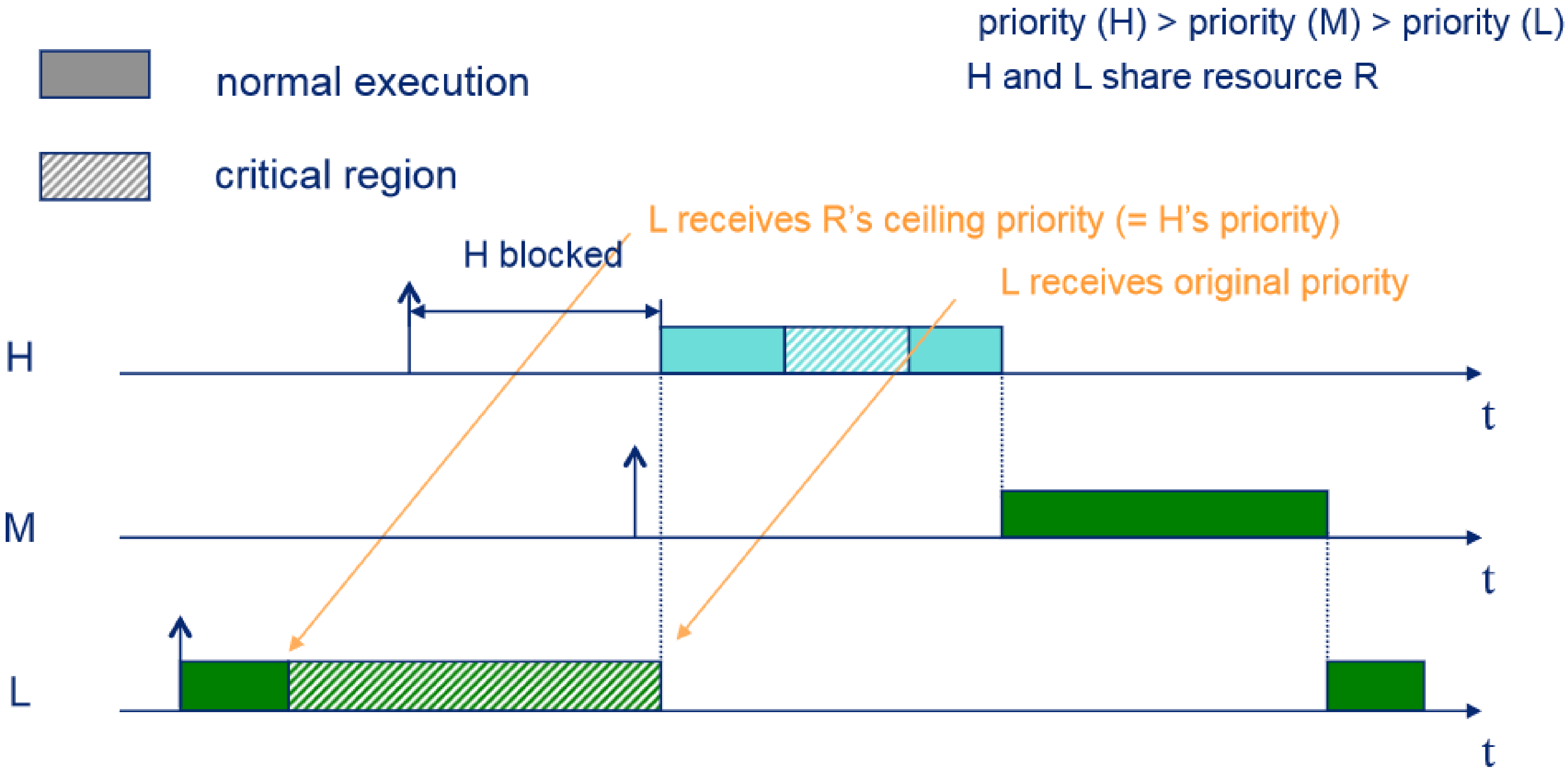
Priority Ceiling Protocol

- Each task has a static default priority assigned (perhaps by the deadline monotonic scheme)
- Each resource has a static ceiling value defined, this is the maximum priority of the tasks that use it
- A task has a dynamic priority that is the maximum of its own static priority and any it inherits due to it blocking higher-priority tasks
- A task can only lock a resource if its dynamic priority is higher than the ceiling of any currently locked resource (excluding any that it has already locked itself)

Priority Ceiling Protocol

- A high-priority task can be blocked at most once during its execution by lower-priority tasks
- Deadlocks are prevented
- Transitive blocking is prevented
- Mutual exclusive access to resources is ensured (by the protocol itself)

Immediate Ceiling Priority Protocol



Response Time and Blocking

$$R_i = C_i + B_i + I_i$$

$$R_i = C_i + B_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

$$w_i^{n+1} = C_i + B_i + \sum_{j \in hp(i)} \left\lceil \frac{w_i^n}{T_j} \right\rceil C_j$$

Insufficient Priorities

- If insufficient priorities then tasks must share priority levels
- If task *a* shares priority with task *b*, then each must assume the other interferes
- Priority assignment algorithm can be used to pack tasks together
- Ada requires 31, RT-POSIX 32 and RT-Java 28

Processor Demand Criterion

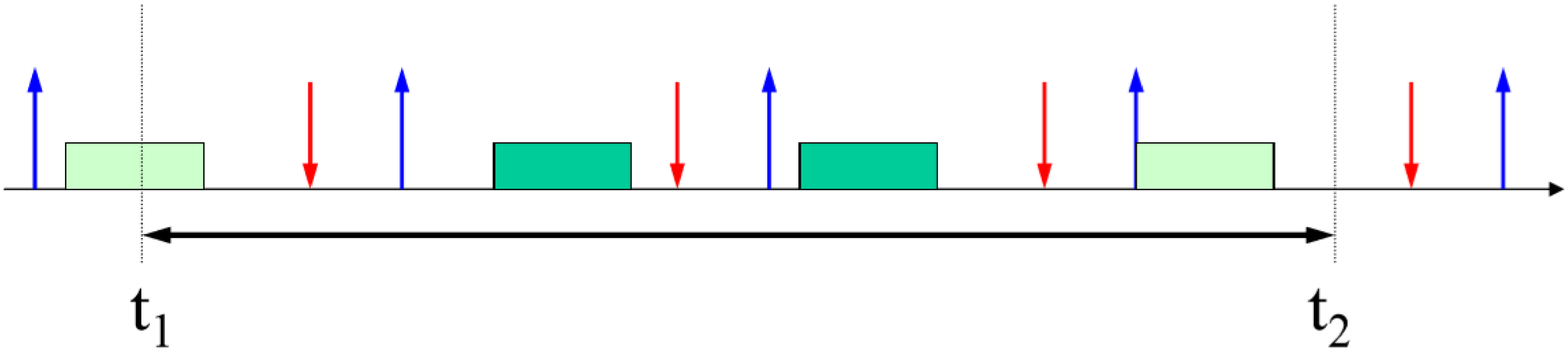
[Baruah, Howell, Rosier 1990]

For checking the existence of a feasibile schedule under **EDF**

In any interval of time, the computation demanded by the task set must be no greater than the available time.

$$\forall t_1, t_2 > 0, \qquad g(t_1, t_2) \leq (t_2 - t_1)$$

Processor Demand

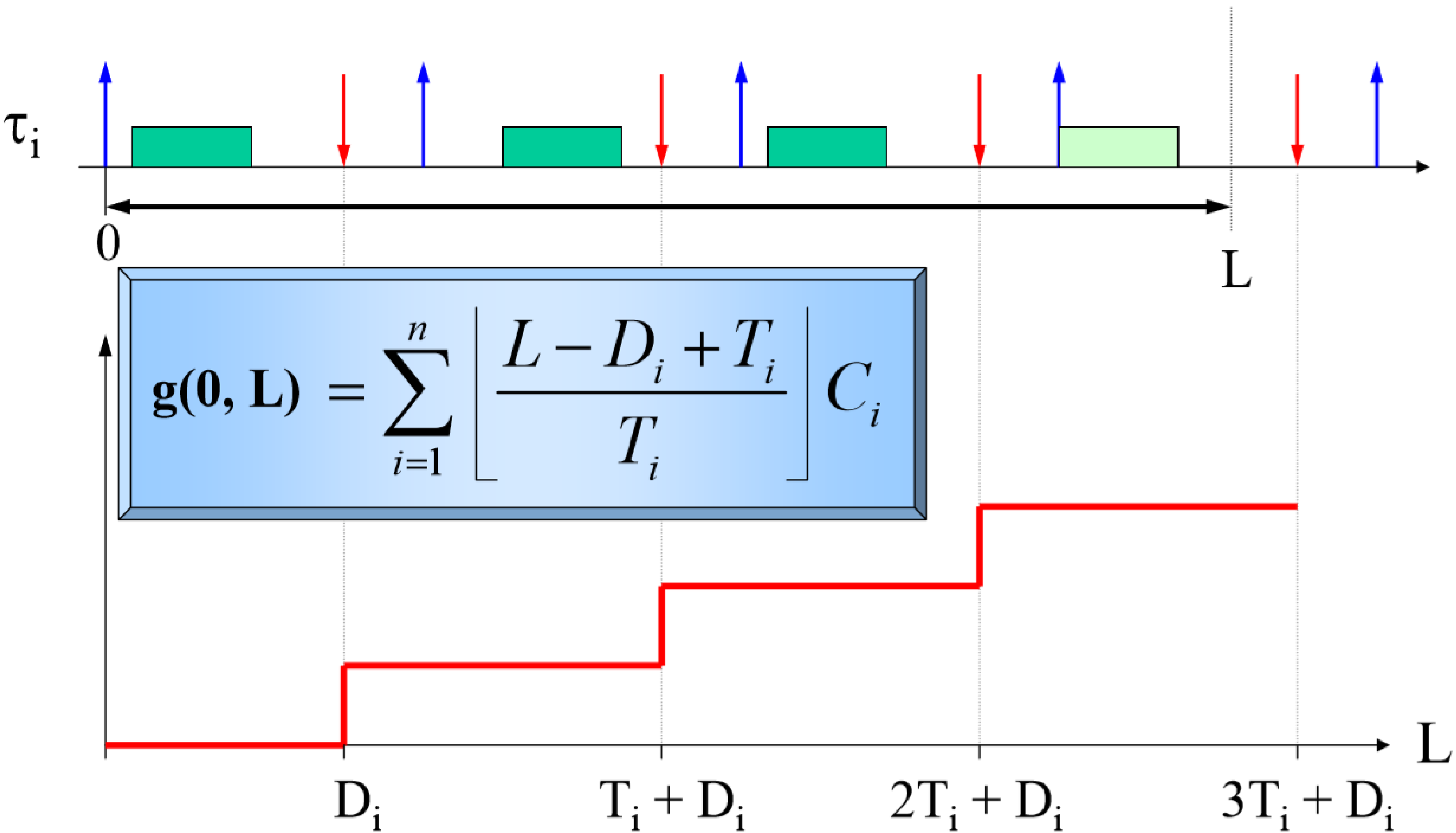


The demand in $[t_1, t_2]$ is the computation time of those jobs started at or after t_1 with deadline less than or equal to t_2:

$$g(t_1, t_2) = \sum_{r_i \geq t_1}^{d_i \leq t_2} C_i$$

Processor Demand

For synchronous task sets we can only analyze intervals [0,L]

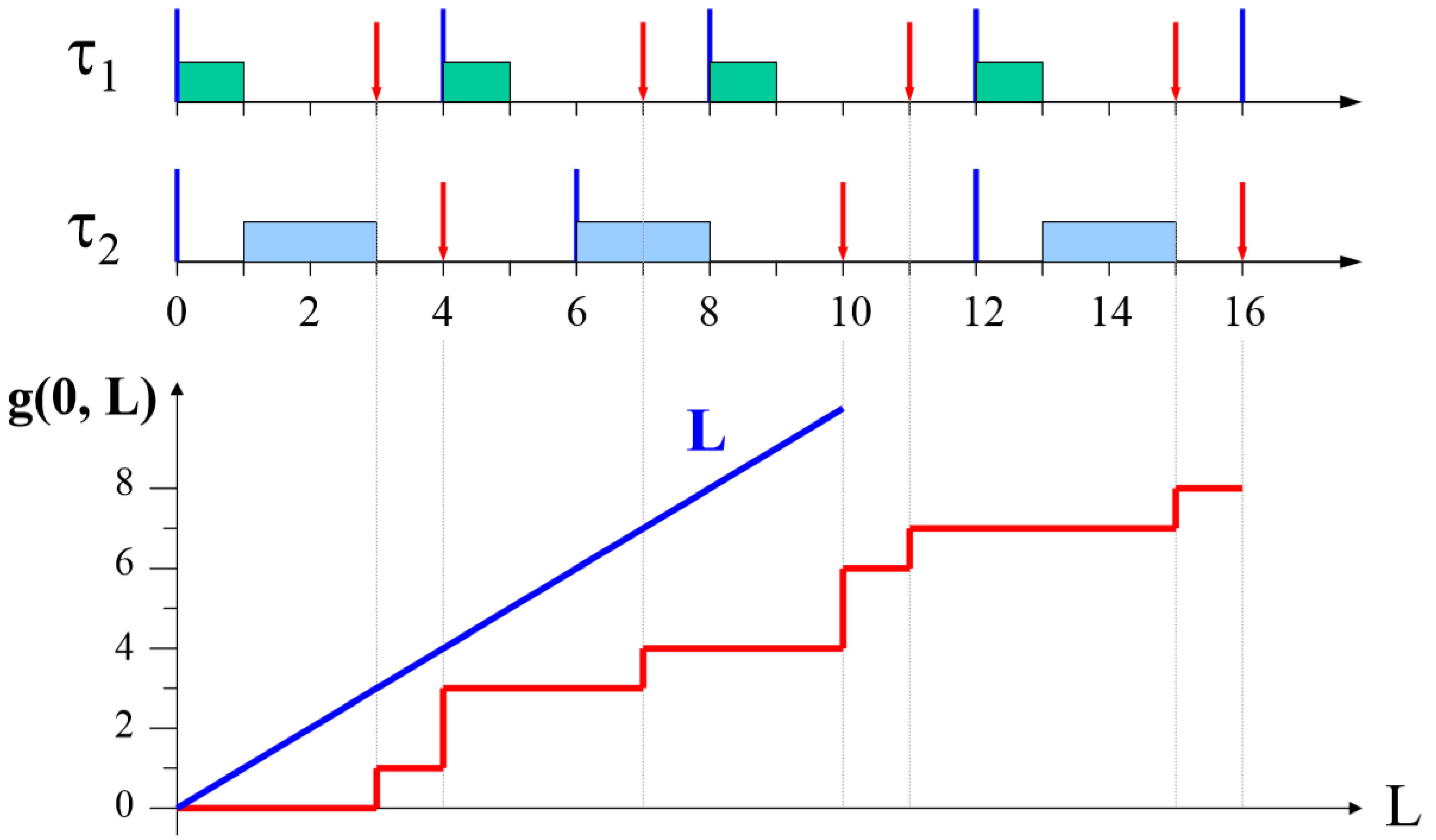


$$\mathbf{g(0, L)} = \sum_{i=1}^{n} \left\lfloor \frac{L - D_i + T_i}{T_i} \right\rfloor C_i$$

Processor Demand Test

$$\forall L > 0 \quad \sum_{i=1}^{n} \left\lfloor \frac{L - D_i + T_i}{T_i} \right\rfloor C_i \leq L$$

Example



Upper Bound for PD Test

$$L_a = \max\left\{ D_1, \ldots, D_N, \frac{\sum_{i=1}^{N} (T_i - D_i) C_i / T_i}{1 - U} \right\}$$

U is the utilisation of the task set, note upper bound not defined for U=1

[U.C. Devi. An Improved Schedulability Test for Uniprocessor Periodic Task Systems]

PD Test with Blocking

- Compute the maximum blocking time for each task
- Inflate C_i by B_i

EDF **D = T**

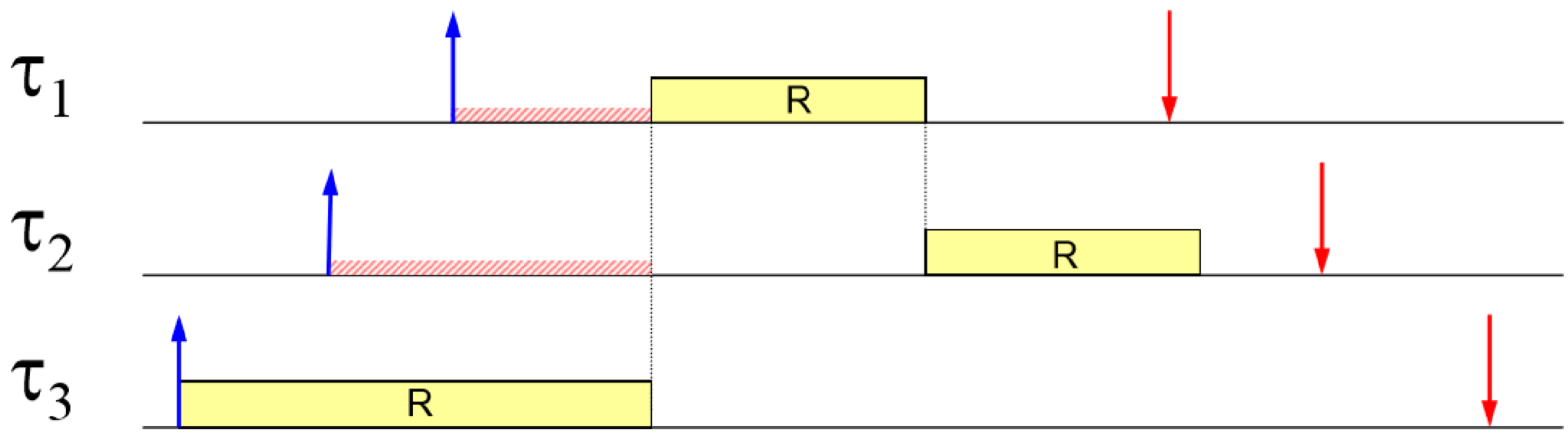
$$\forall i \qquad \sum_{k=1}^{i-1} \frac{C_k}{T_k} + \frac{C_i + B_i}{T_i} \leq 1$$

EDF **D ≤ T** task set is schedulable if U < 1 and

$$\forall i \ \forall L \qquad B_i + \sum_{k=1}^{n} \left\lfloor \frac{L + T_k - D_k}{T_k} \right\rfloor C_k \leq L$$

Non-preemtive scheduling

It is a special case of preemptive scheduling where all tasks share a single resource for their entire duration.



The max blocking time for task τ_i is given by the largest C_k among the lowest priority tasks:

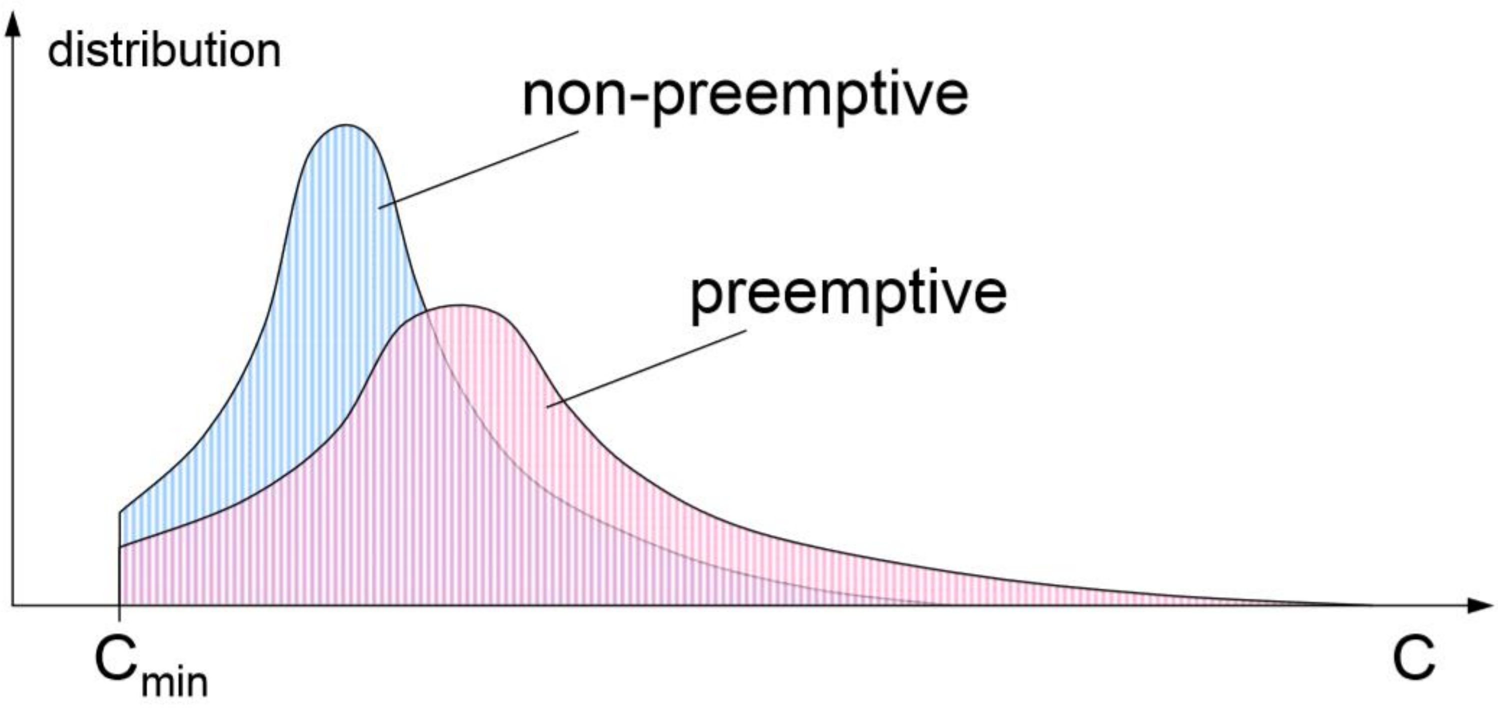
$$B_i = \max\{C_k : P_k < P_i\}$$

Advantages of NP scheduling

- It reduces runtime overhead
  - Less context switches
  - No semaphores are needed for critical sections
- It reduces stack size, since no more than one task can be in execution.
- It preserves program locality, improving the effectiveness of
  - Cache memory
  - Pipeline mechanisms
  - Prefetch queues

Advantages of NP scheduling

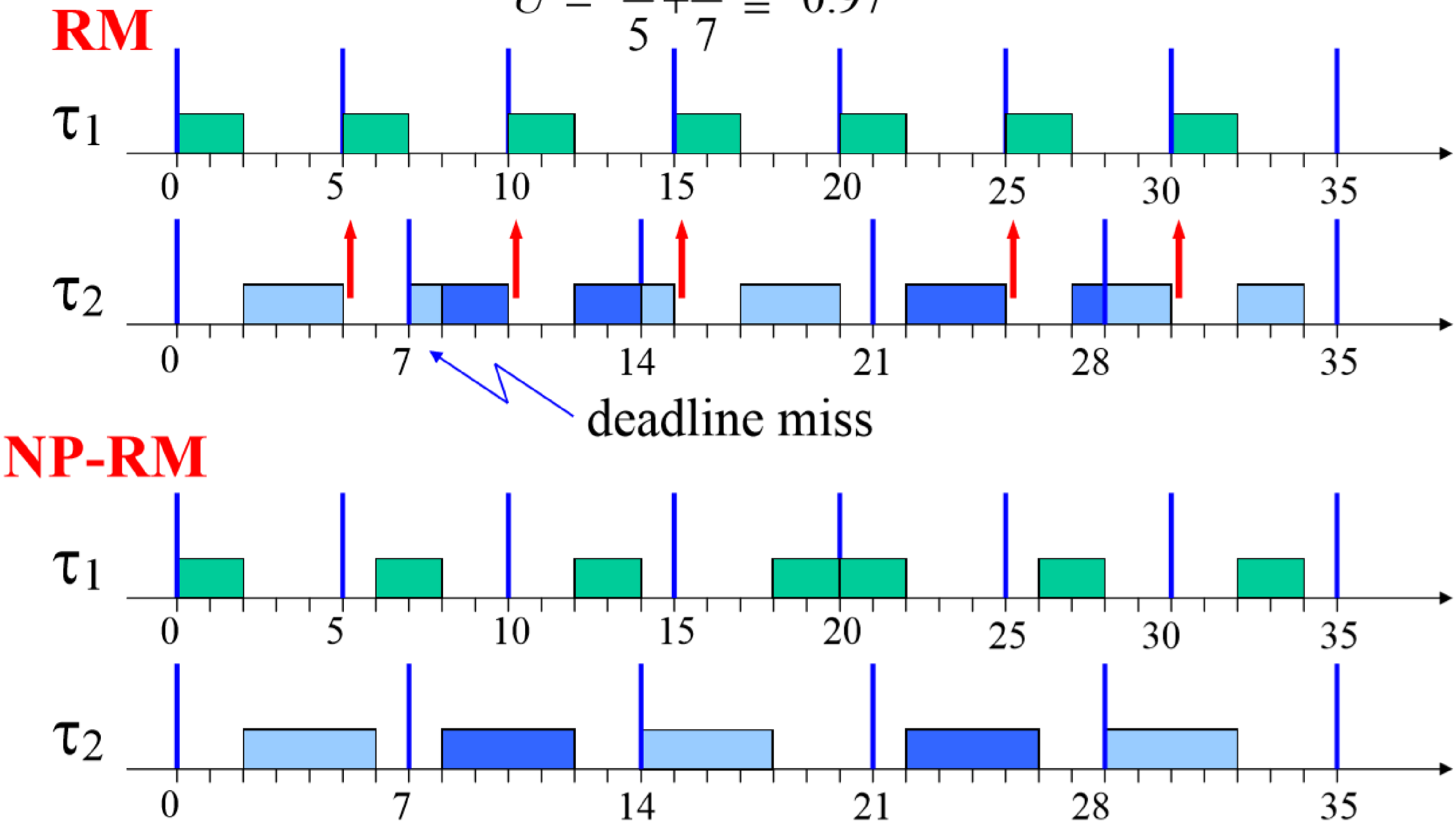
- As a consequence, task execution times are
  - Smaller
  - More predictable



Advantages of NP scheduling

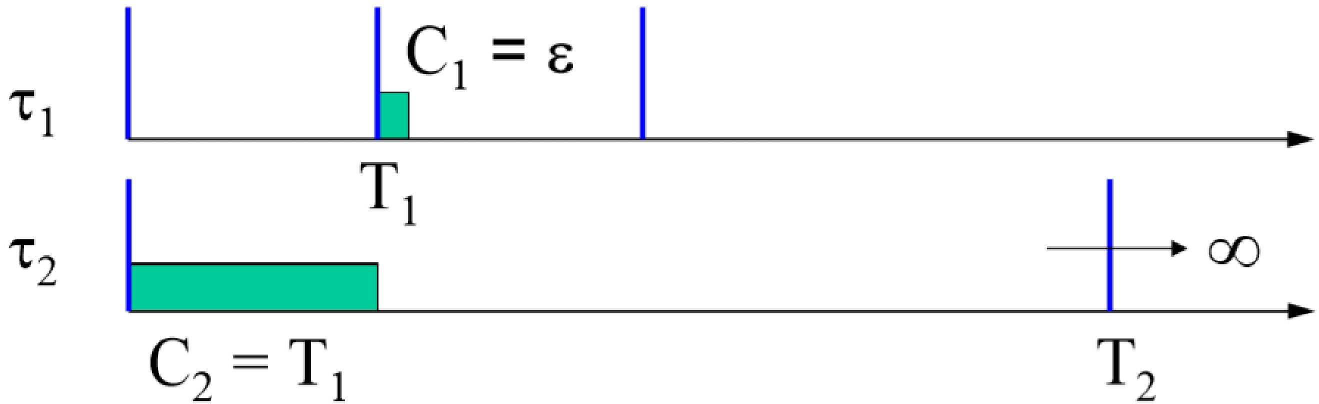
In fixed priority systems can improve schedulabiilty:

$$U = \frac{2}{5} + \frac{4}{7} \cong 0.97$$



Disadvantages of NP scheduling

- In general, NP scheduling reduces schedulability.
- The utilization bound under non preemptive scheduling drops to zero:

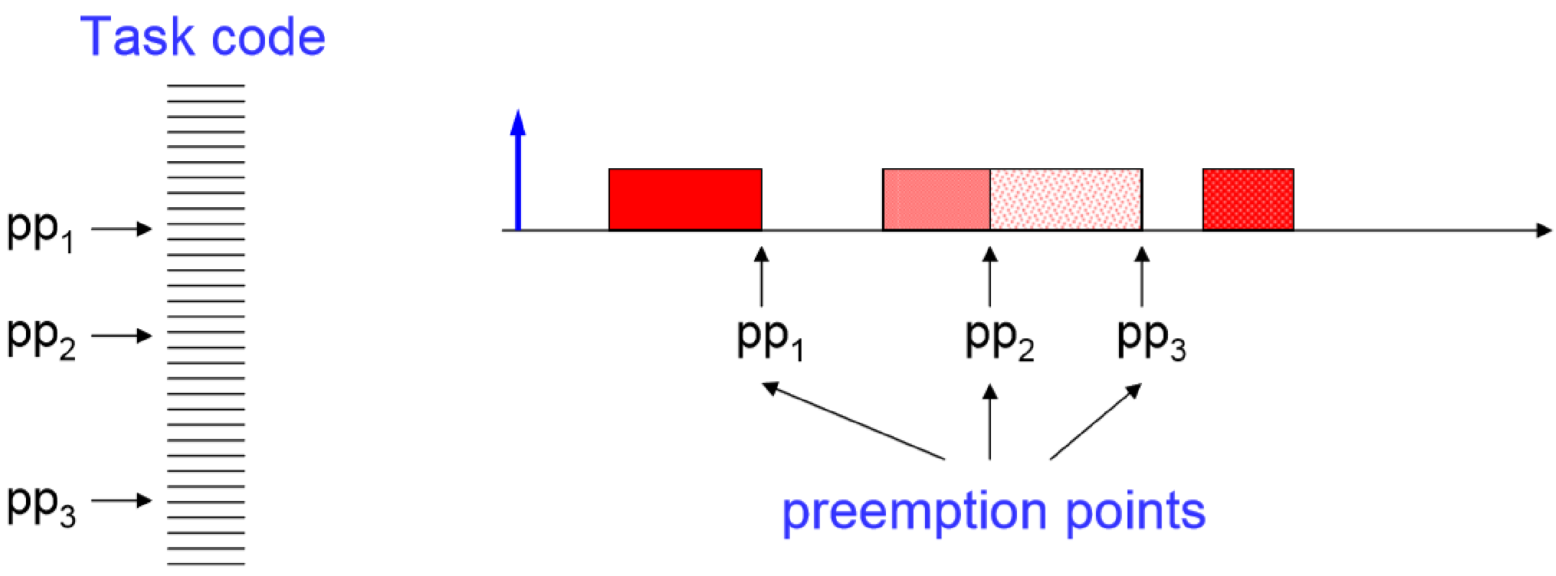


$$U = \frac{\varepsilon}{T_1} + \frac{C_2}{\infty} \rightarrow 0$$

Trade-off solutions

Tunable Preemptive Systems

- Compute the longest non-preemptive section that allows a feasible schedule
- Allow preemption only in certain points in the code.



Handling Jitter & Delay

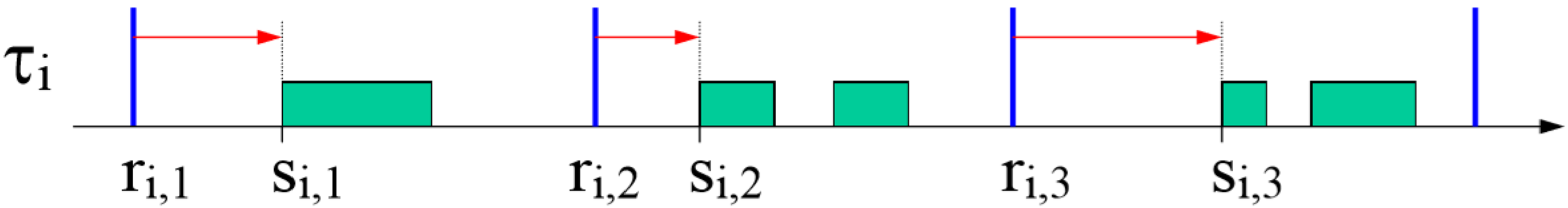
Jitter for an event

The maximum time variation in the occurrence of a particular event in two consecutive jobs.

In many control applications, delay and jitter can cause instability or jerky behavior

Definitions

Start time delay (Input Latency): $INL_{i,k} = s_{i,k} - r_{i,k}$



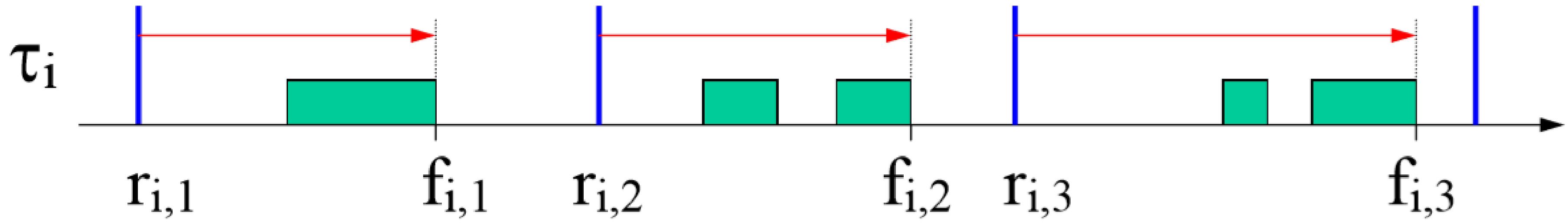
Start time Jitter (Input Jitter):

Absolute: $$INJ_i^{abs} = \max_k (s_{i,k} - r_{i,k}) - \min_k (s_{i,k} - r_{i,k})$$

Relative: $$INJ_i^{rel} = \max_k \left| (s_{i,k} - r_{i,k}) - (s_{i,k-1} - r_{i,k-1}) \right|$$

Definitions

Response Time (Output Latency): $R_{i,k} = f_{i,k} - r_{i,k}$



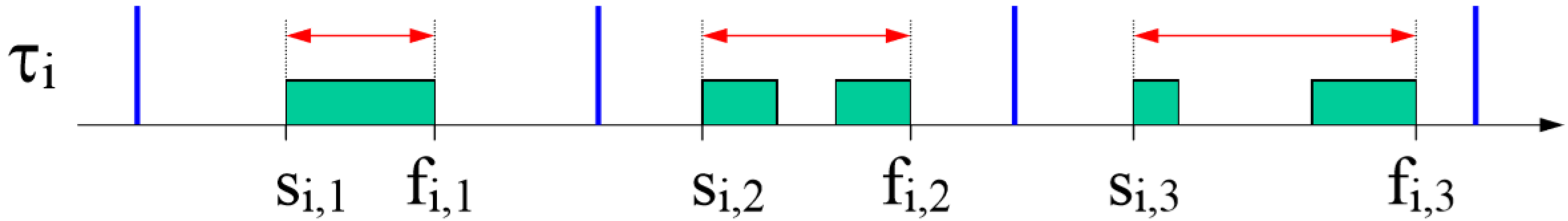
Response Time Jitter (Output Jitter):

Absolute: $RTJ_i^{abs} = \max_k (f_{i,k} - r_{i,k}) - \min_k (f_{i,k} - r_{i,k})$

Relative: $RTJ_i^{rel} = \max_k \left| (f_{i,k} - r_{i,k}) - (f_{i,k-1} - r_{i,k-1}) \right|$

Definitions

Input-Output Latency: $IOL_{i,k} = f_{i,k} - s_{i,k}$

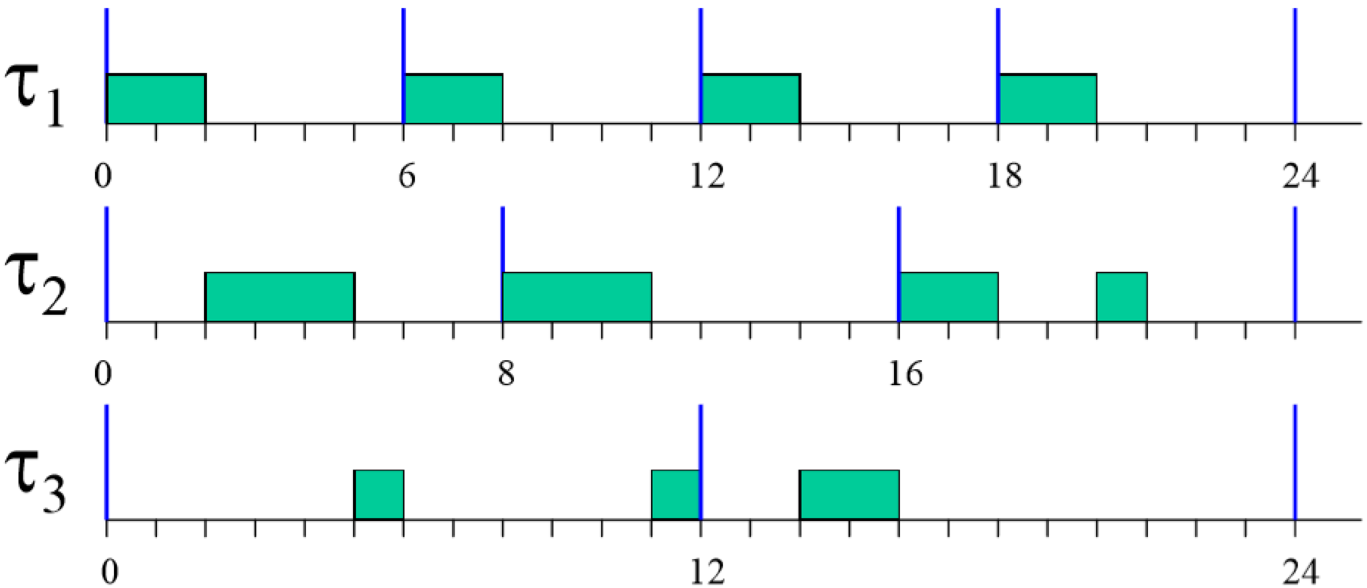


Input-Output Jitter:

Absolute: $IOJ_i^{abs} = \max_k (f_{i,k} - s_{i,k}) - \min_k (f_{i,k} - s_{i,k})$

Relative: $IOJ_i^{rel} = \max_k \left| (f_{i,k} - s_{i,k}) - (f_{i,k-1} - s_{i,k-1}) \right|$

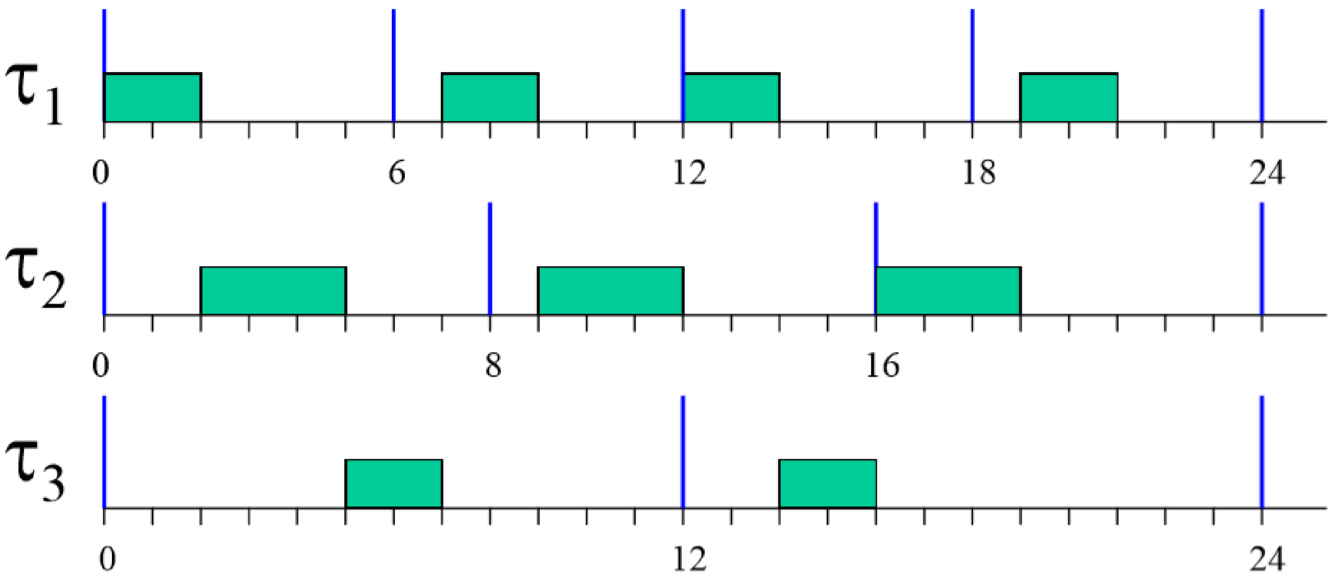
Jitter under RM



	INJ	RTJ	IOJ
τ_1	0	0	0
τ_2	2	2	2
τ_3	3	8	5

Low priority tasks experience very high delay and jitter

Jitter under EDF

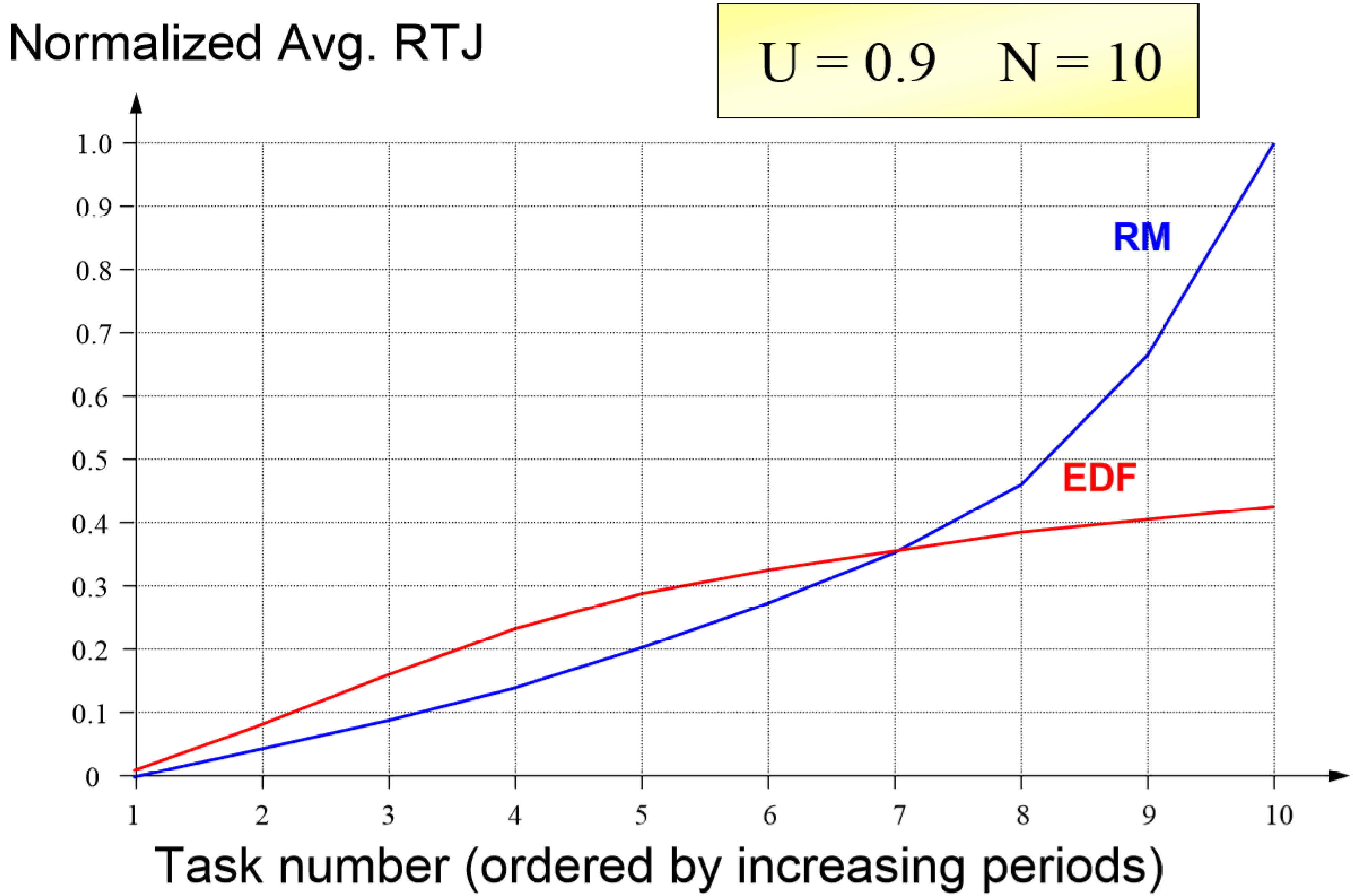


	INJ	RTJ	IOJ
τ_1	1	1	0
τ_2	2	2	0
τ_3	3	3	0

For a little increase of RTJ_1, RTJ_3 decreases a lot

IOJ = 0 for all the tasks

Jitter under RM and EDF



How to handle delay and jitter

Two main methods can be used to reduce the effect of delay and jitter:

1. compensate them by proper control actions;
2. reduce them as much as possible.

Even when compensation is used, reducing delay and jitter improves system performance

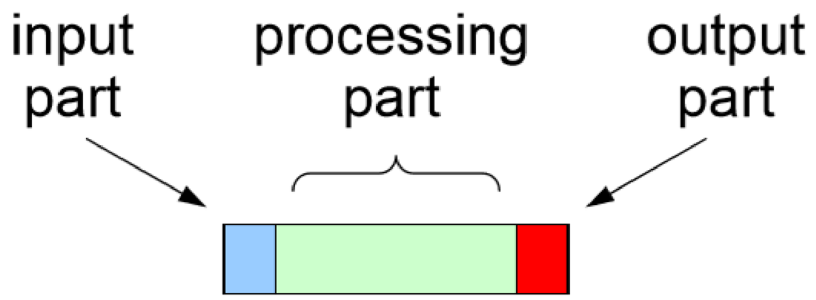
Hence we concentrate on reduction methods

Jitter Reduction methods

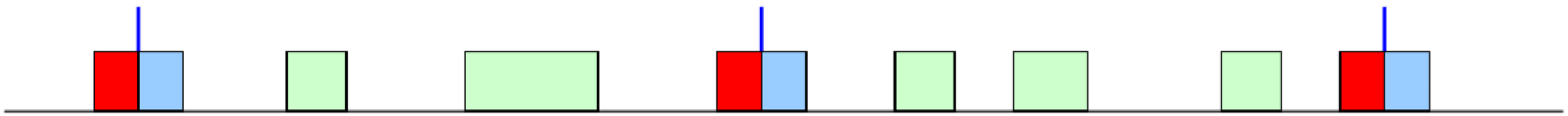
Three methods can be used to reduce the jitter caused by task interference:

1. Task Splitting
2. Advancing Deadlines
3. Non Preemptive Scheduling

Reducing Jitter by Task Splitting



The idea is to force input and output parts to execute in a time-triggered fashion, using timers:



Reducing Jitter by Task Splitting

Advantages

1. Jitter is reduced at the minimum possible value;

2. If input and output parts are small, this method is effective for any task, independently of the scheduler and task parameters.

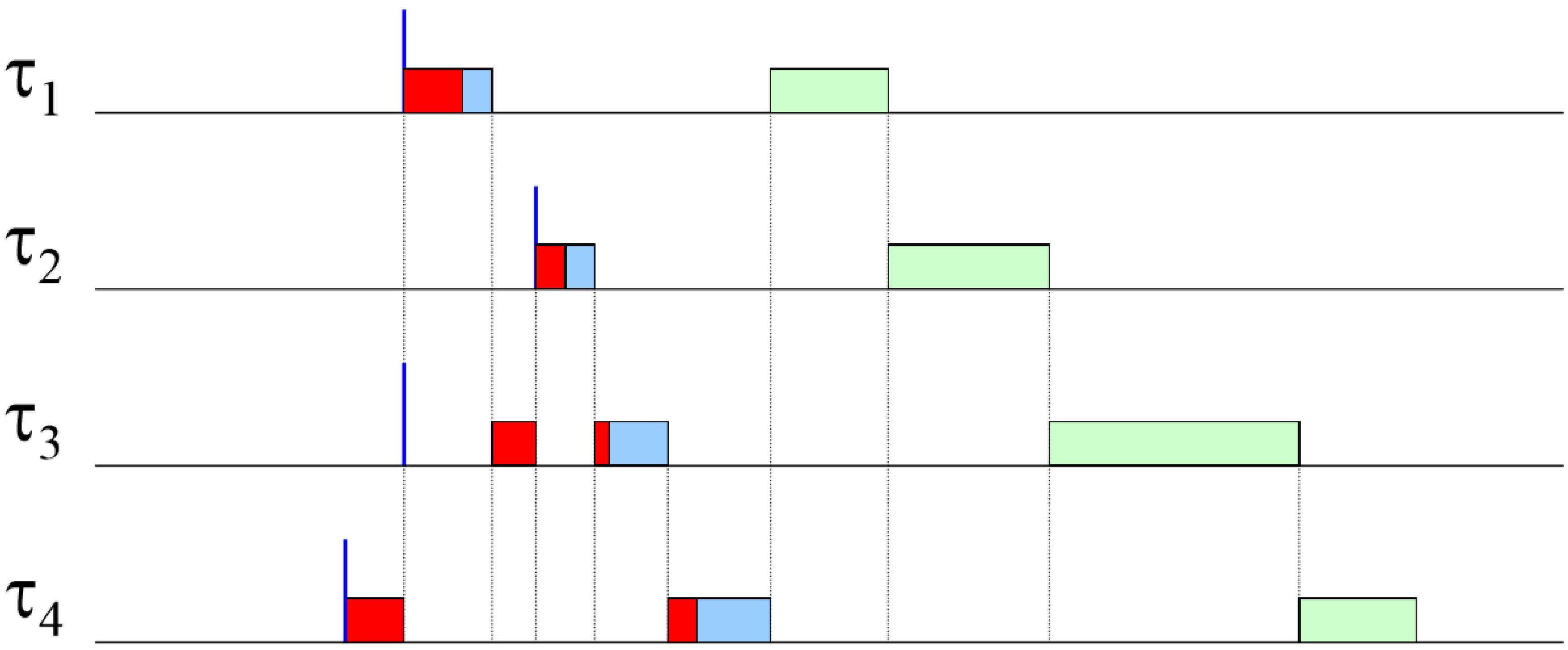
Reducing Jitter by Task Splitting

Disadvantages

1. Extra effort to be implemented;
2. Jitter is reduced at the expense of delay;
3. Input and output parts create extra interference which complicates the analysis and reduces schedulability;
4. Input and output parts may compete and need to be scheduled with some policy.

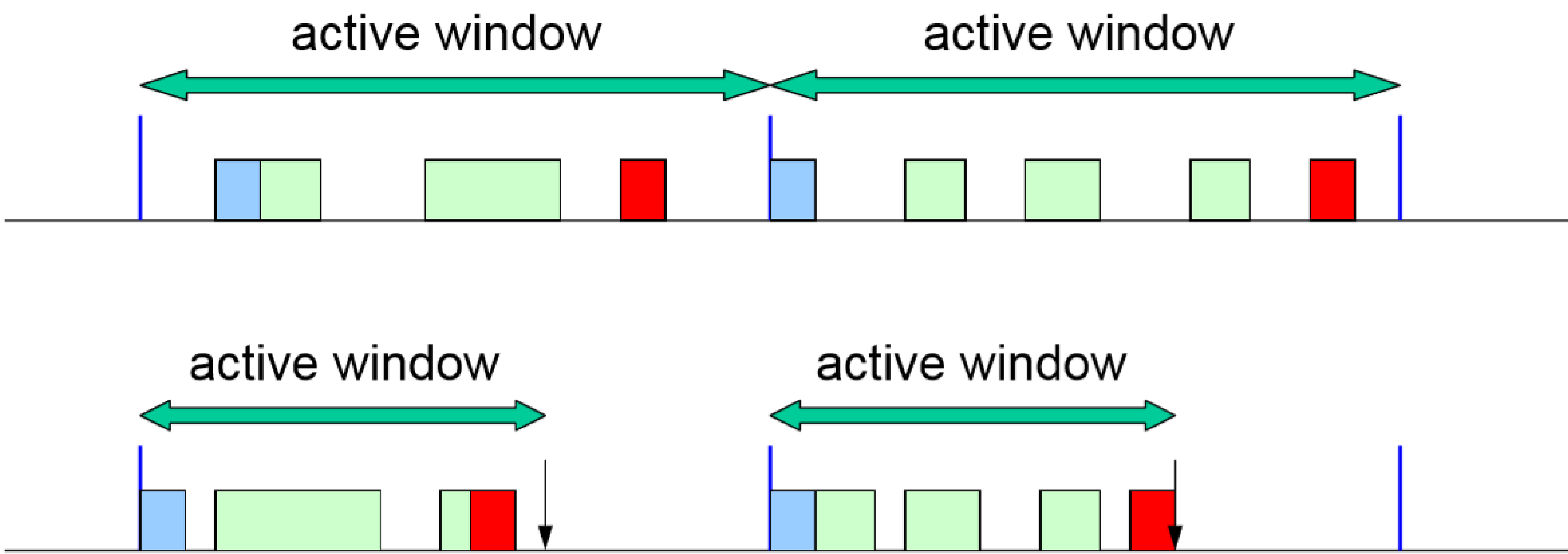
Reducing Jitter by Task Splitting

Interfering I/O parts



Reducing Jitter by Advancing Deadlines

The idea is to advance task deadlines to reduce the active window in which jobs can be executed:



Reducing Jitter by Advancing Deadlines

Advantages

1. Easy to implement (no special support is required from the OS);
2. No extra interference caused by additional timer interrupts;
3. Both delay and jitter are reduced!!

Reducing Jitter by Advancing Deadlines

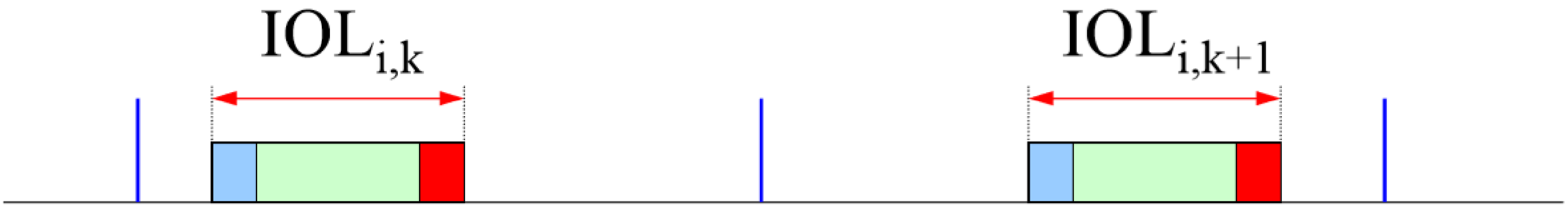
Disadvantages

1. Not all tasks can reduce jitter to zero. A further reduction can be achieved by proper offsets, but the analysis requires exponential complexity.

2. Advancing deadlines reduces system schedulability.

Reducing Jitter by Non Preemption

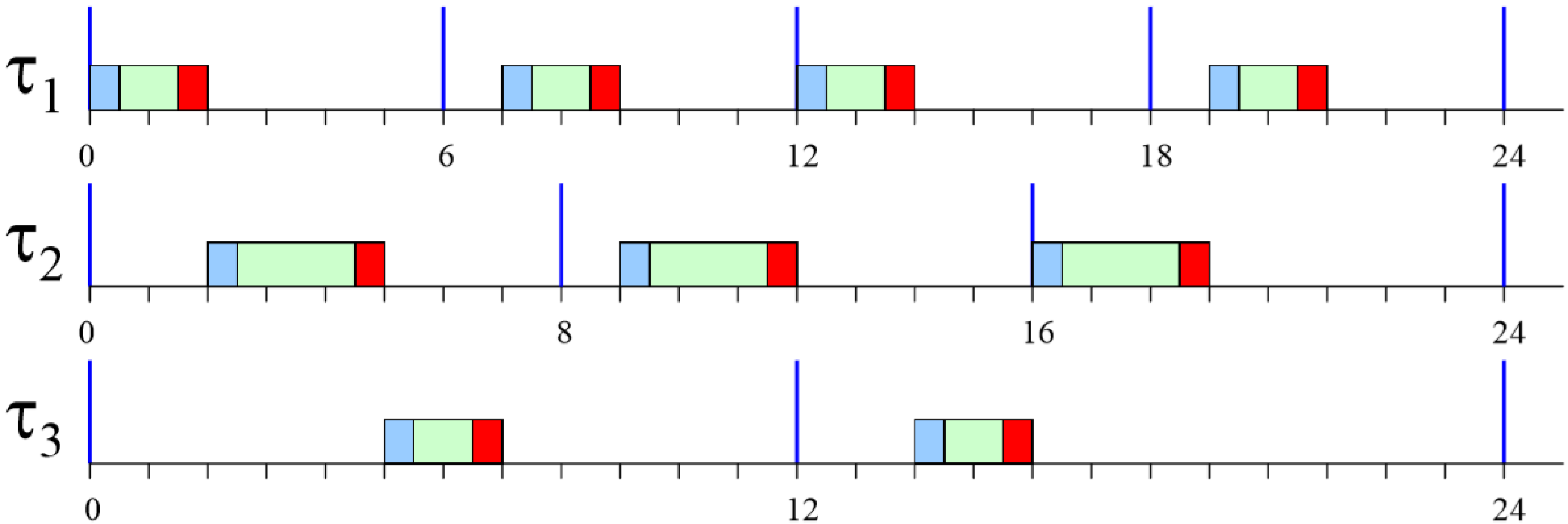
Disabling preemtions a task can be delayed, but once started cannot be interrupted:



$$\forall k \begin{cases} IOL_{i,k} = C_i \\ IOJ_{i,k} = 0 \end{cases}$$

Reducing Jitter by Non Preemption

Example with 3 tasks



Reducing Jitter by Non Preemption

Advantages

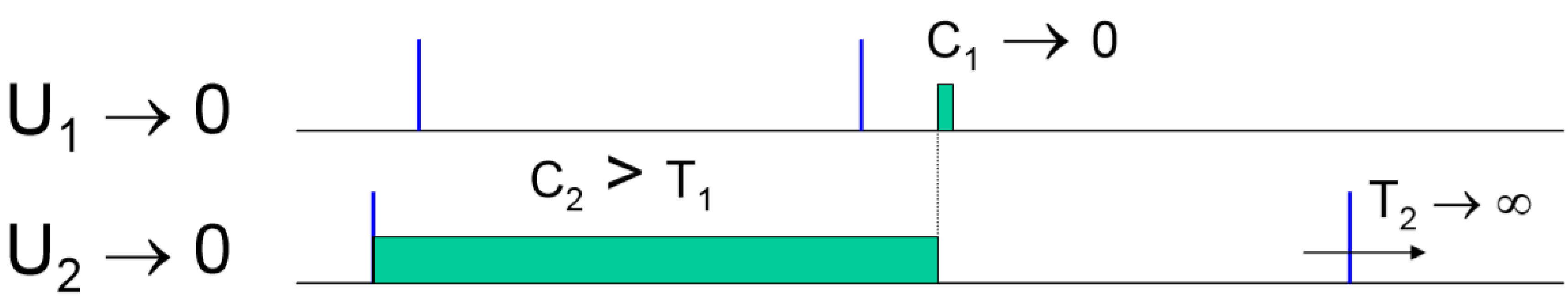
1. $IOJ_i = 0$ for all tasks;
2. $IOL_i = C_i$ for all tasks, simplifying the use of delay compensation techniques;
3. Non preemptive execution also simplifies resource management (there is no need to protect critical sections).
4. Non preemptive execution allows stack sharing.

Reducing Jitter by Non Preemption

Disadvantages

1. Non preemption reduces schedulability (analysis must take blocking times into account);

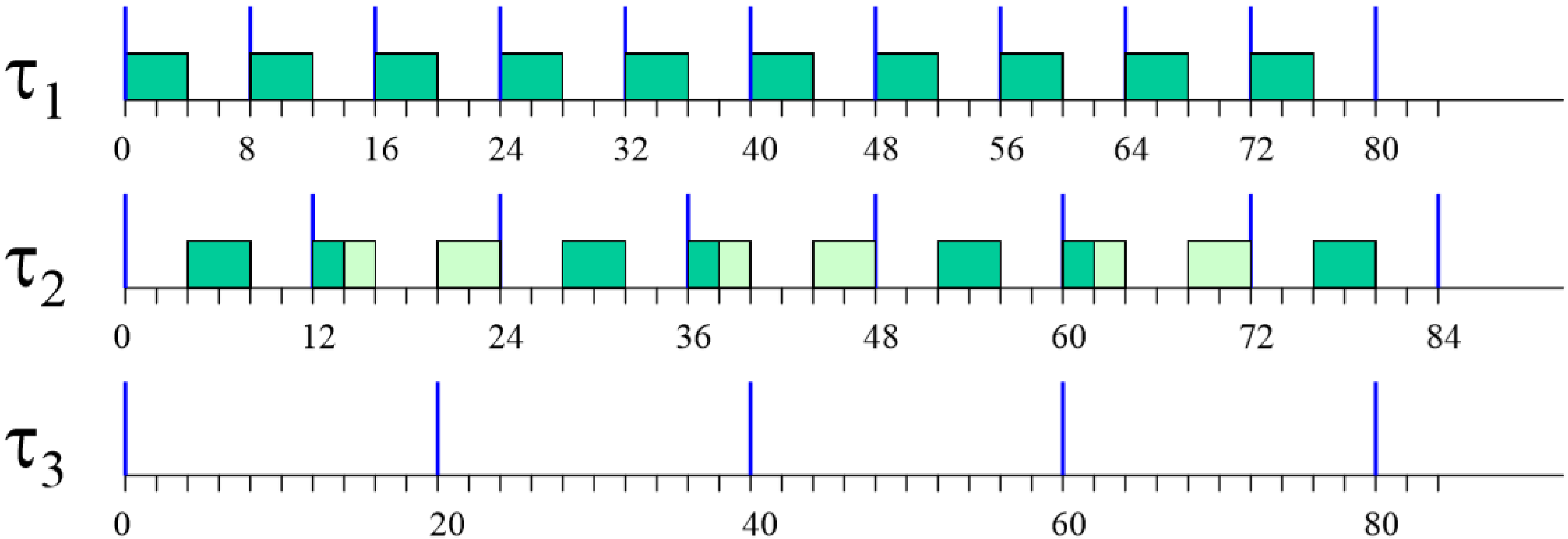
2. The utilization upper bound drops to zero:



Scheduling under overload conditions

RM under overloads

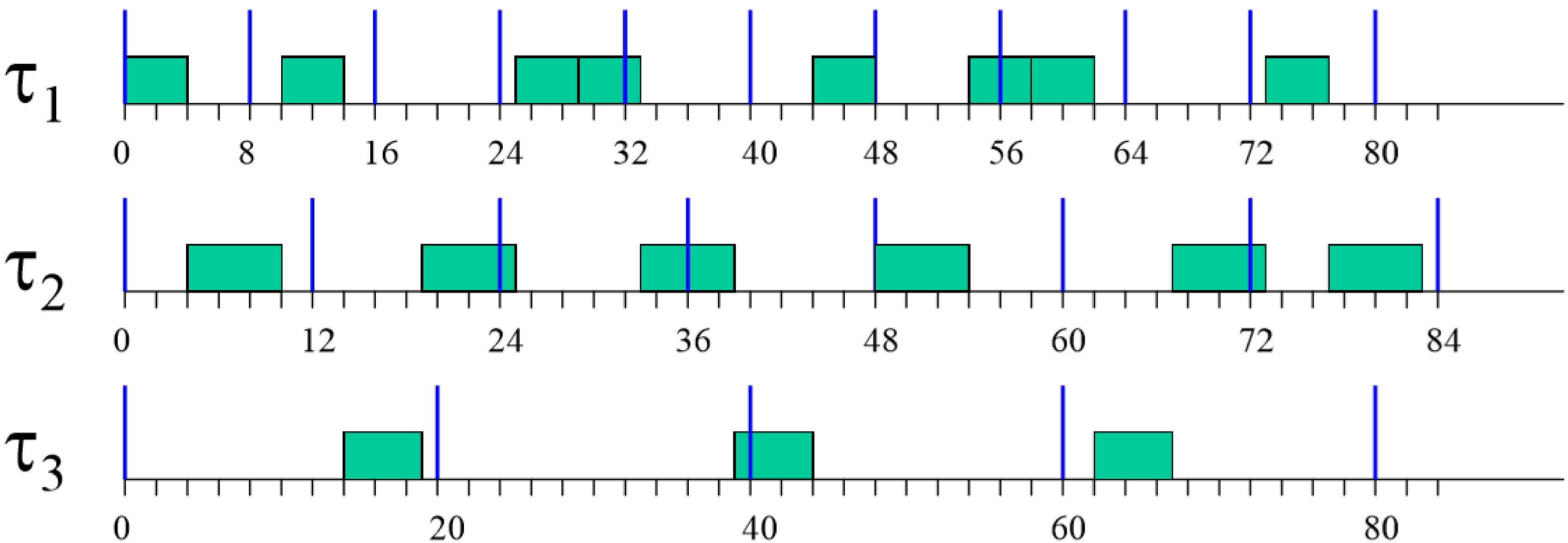
$$U = \frac{4}{8} + \frac{6}{12} + \frac{5}{20} = 1.25$$



- High priority tasks execute at the proper rate
- Low priority tasks are completely blocked

EDF under overloads

$$U = \frac{4}{8} + \frac{6}{12} + \frac{5}{20} = 1.25$$



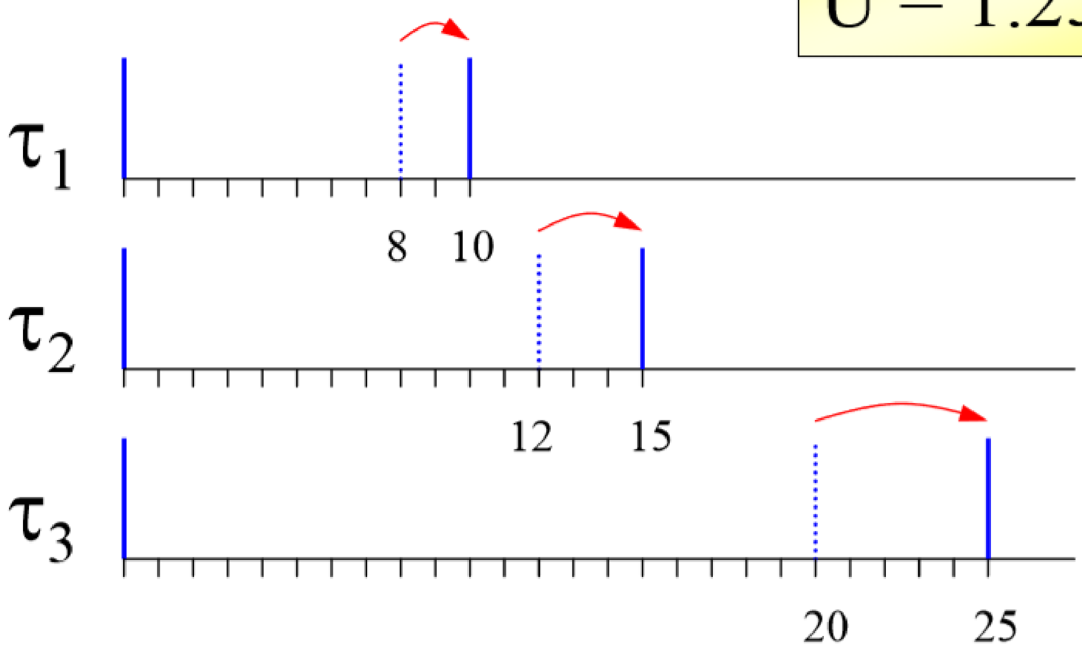
- All tasks execute at a slower rate
- No task is blocked

EDF under overloads

Theorem (Cervin '03)

If $U > 1$, EDF executes tasks with an average period $T'_i = T_i U$.

$U = 1.25$



	T_i	T'_i
τ_1	8	10
τ_2	12	15
τ_3	20	25

Exploiting control flexibility

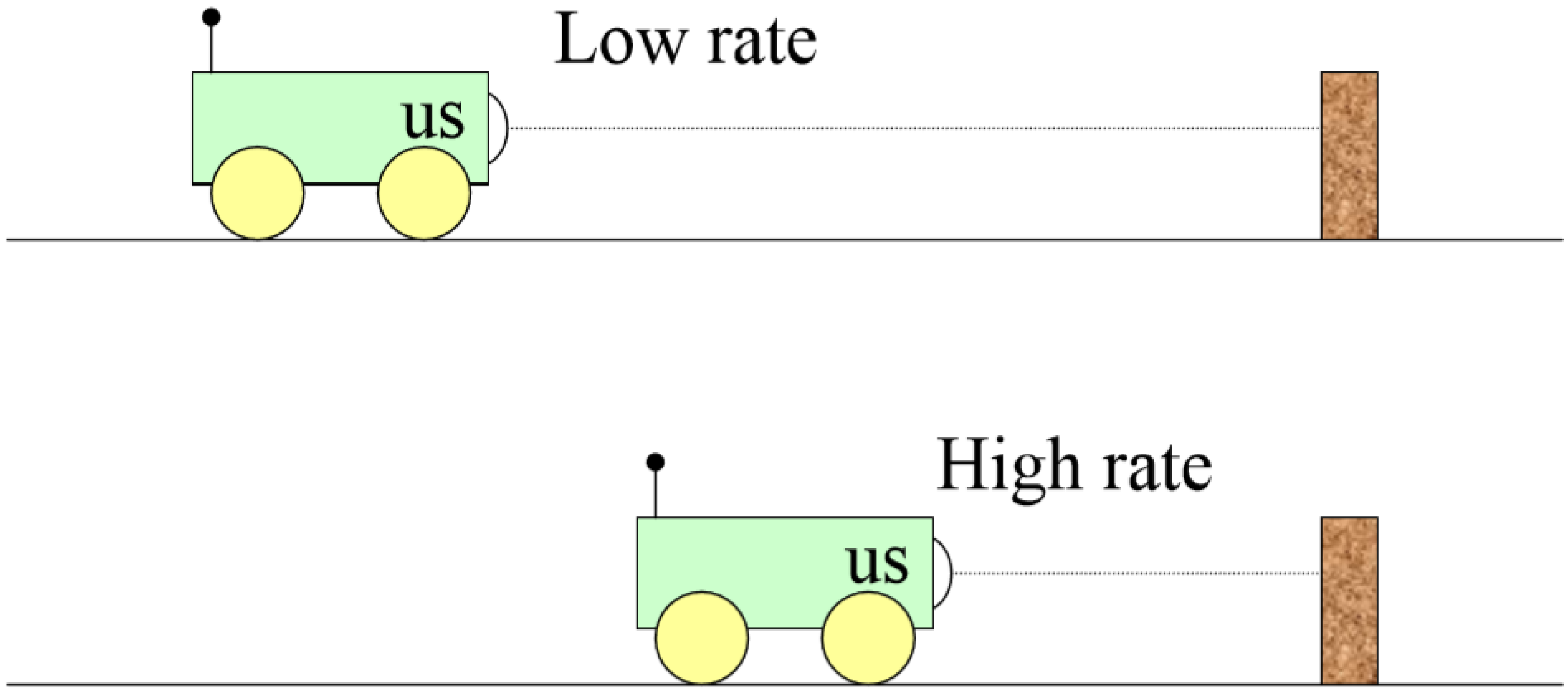
Relaxing timing constraints

- The idea is to reduce the load by increasing deadlines and/or periods.
- Each task must specify a range of values in which its period must be included.
- Periods are increased during overloads, and reduced when the overload is over.

Many control applications allow timing flexibility

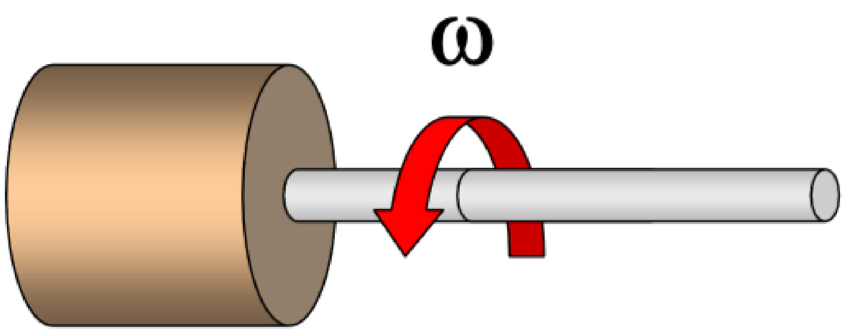
Obstacle avoidance

- The closer the obstacle, the higher the acquisition rate:



Engine control

- Some tasks need to be activated at specific angles of the motor axis:
  - ⇒ the higher the speed, the higher the rate.



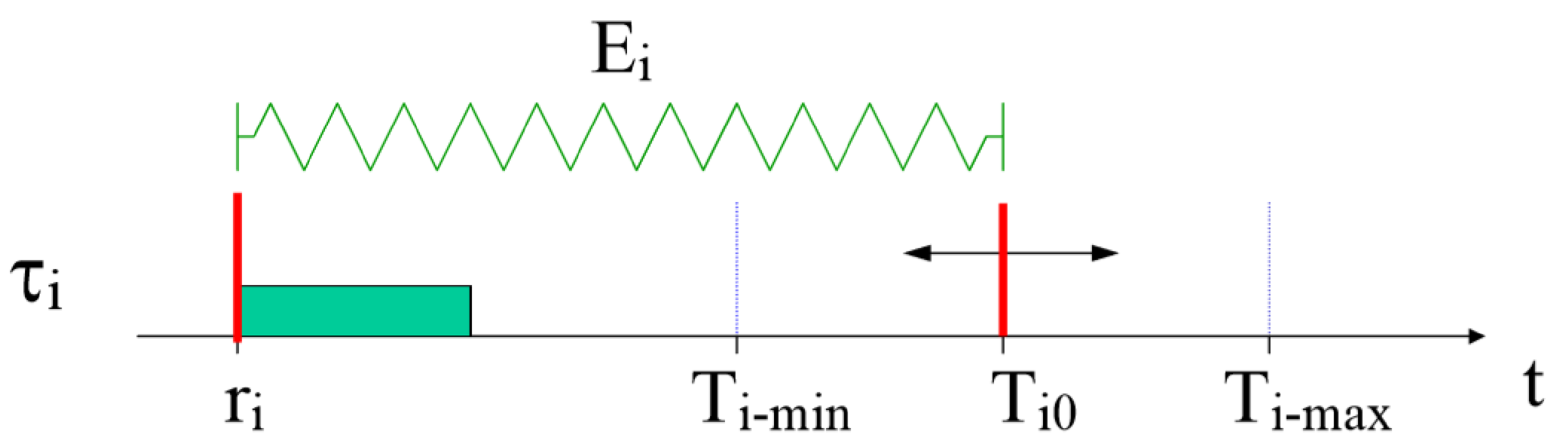
Elastic task model

- A periodic task τ_i is characterized by:

  $(C_i, T_{i\text{-}min}, T_{i\text{-}max}, E_i)$

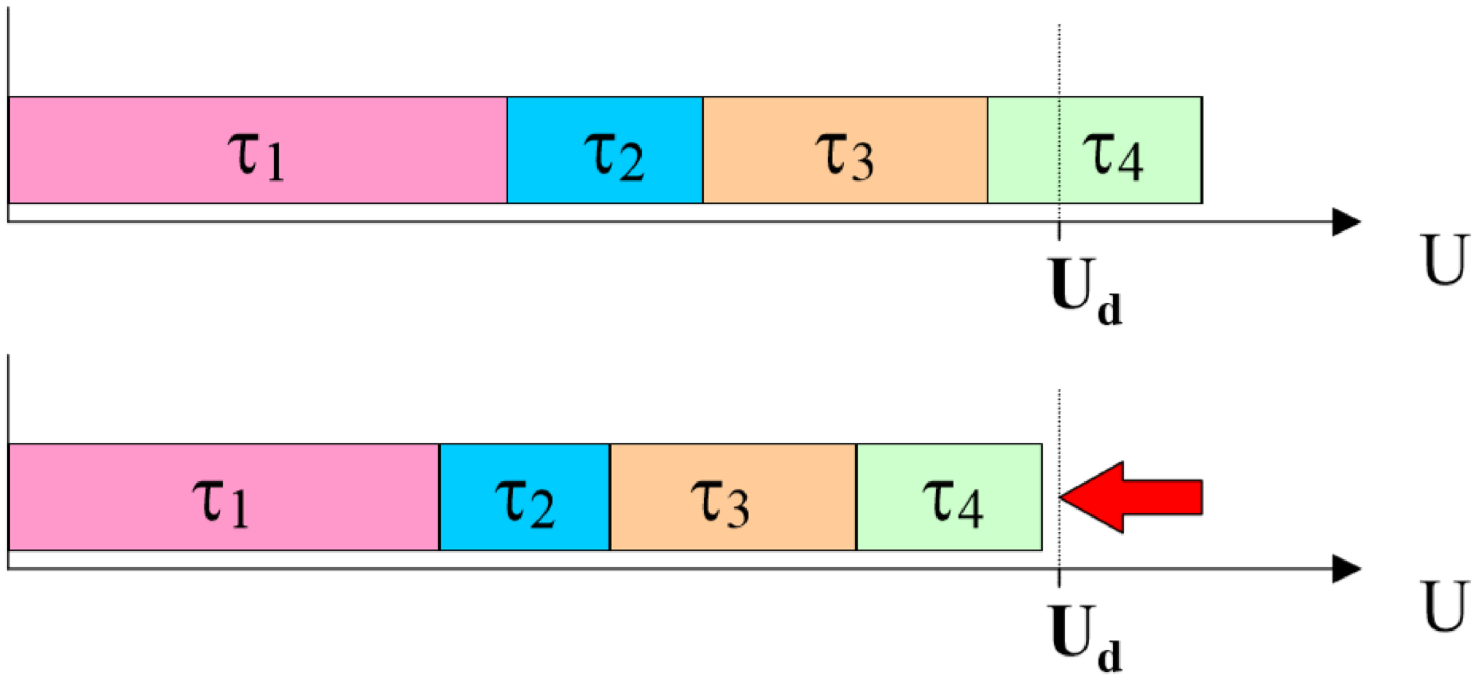
- Tasks' utilizations are treated as elastic springs

- The resistance of a task to a period variation is controlled by an **elastic coefficient E_i**



Compression algorithm

During overloads, utilizations must be compressed to bring the load below a desired value U_d.



Solution for tasks

$$U_i = U_{io} - (U_0 - U_d)\frac{E_i}{E_s}$$

then:

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

Conclusions

Estimate worst-case computation times of tasks, using specific tools and testing.

Select an appropriate scheduling algorithm and a suitable resource access protocol.

Estimate maximum blocking times due non preemptive sections or mutually exclusive resources.

Apply schedulability analysis to verify feasibility.

Exploit system flexibility defining admissible ranges of parameters to cope with overloads.

СПАСИБО ЗА ВНИМАНИЕ

hbd@cs.msu.su