artist

ARTIST2 Summer School 2008 in Europe

Autrans (near Grenoble), France

September 8-12, 2008

Real-Time Scheduling and Resource Management

Real-Time Systems Laboratory

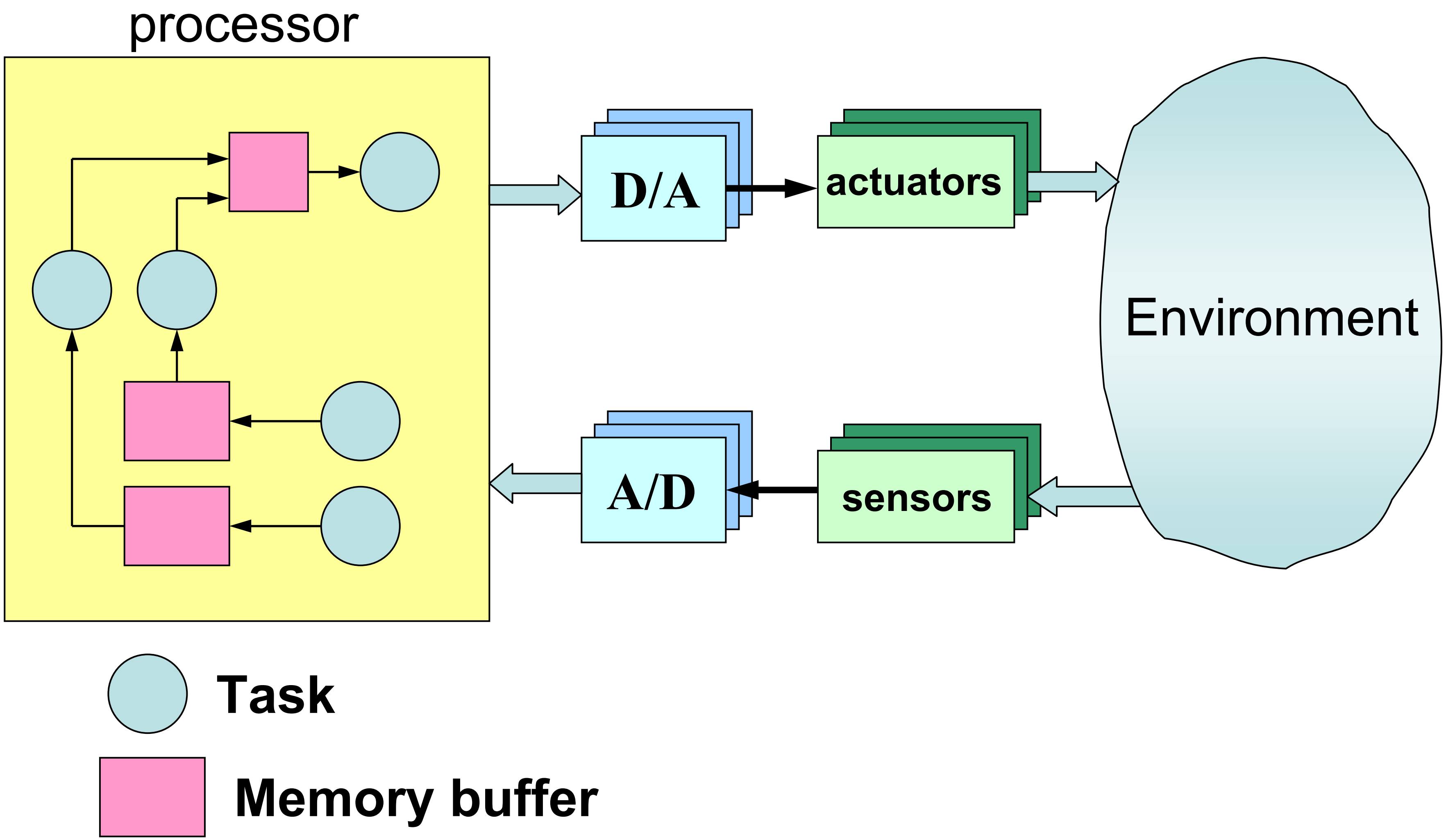
Lecturer: Giorgio Buttazzo
Full Professor
Scuola Superiore Sant'Anna

Information Society Technologies

Outline

- Importance of scheduling in embedded systems
- Review of main scheduling algorithms
- Schedulability analysis
- Taking into account shared resources
- Preemptive vs. Non preemptive scheduling
- Bounding delays and jitter
- Managing oveloads
- Design issues: integrating RT and control theory

Software Control Systems



Typical task structure

buffer → <read data>

<process data>

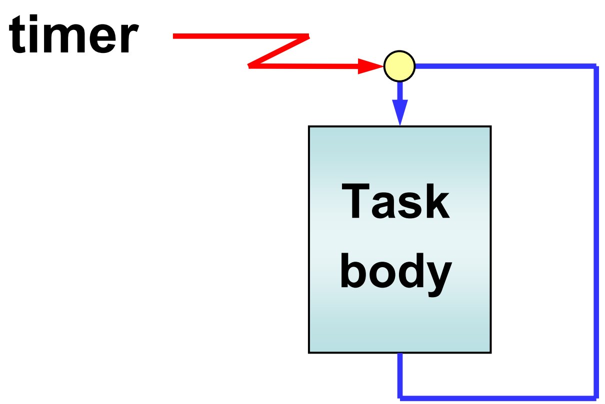
<write data> → buffer

<wait for next activation>

Activation modes

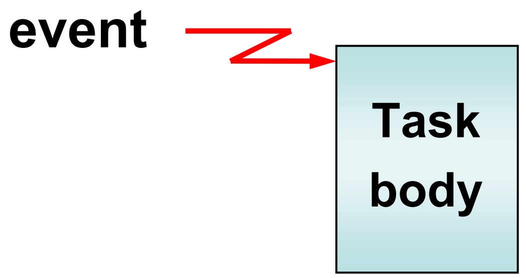
Periodic task **(time driven)**

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



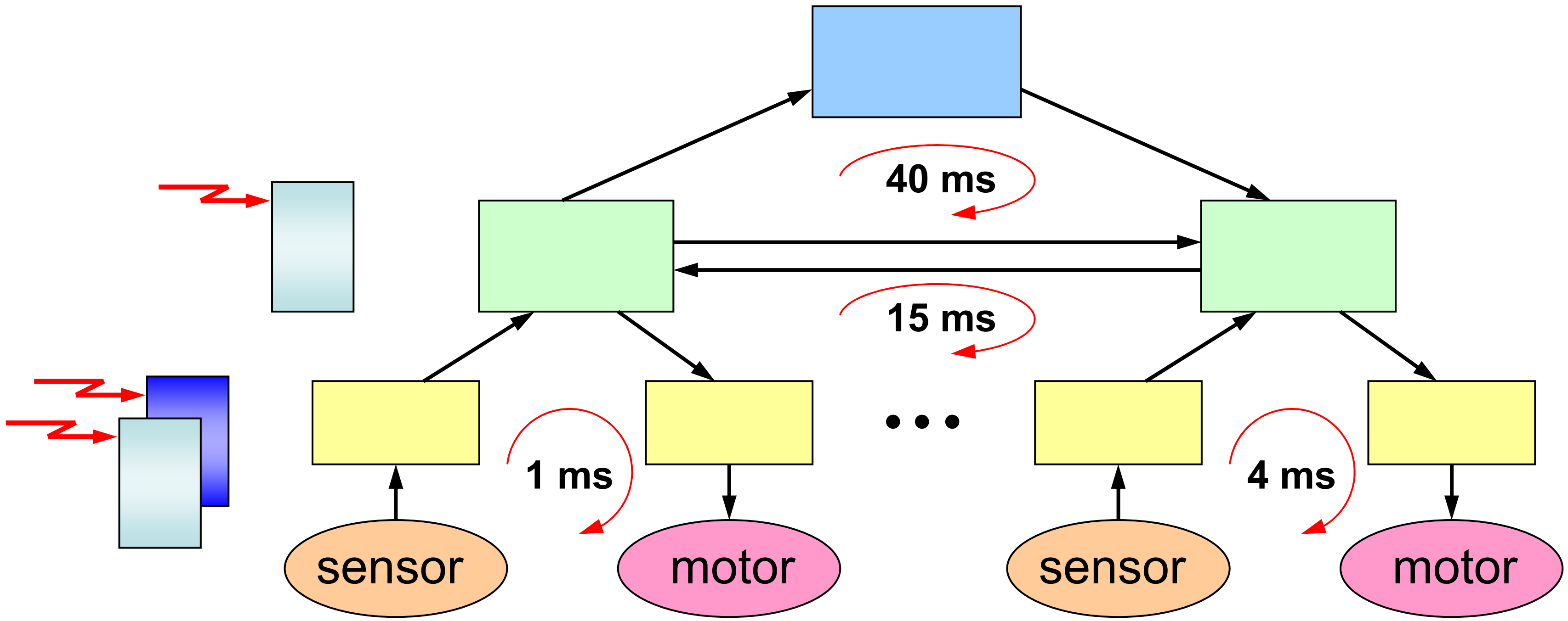
Aperiodic task **(event driven)**

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



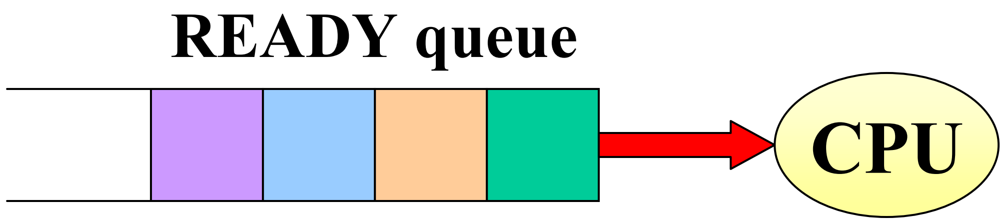
Complex control applications

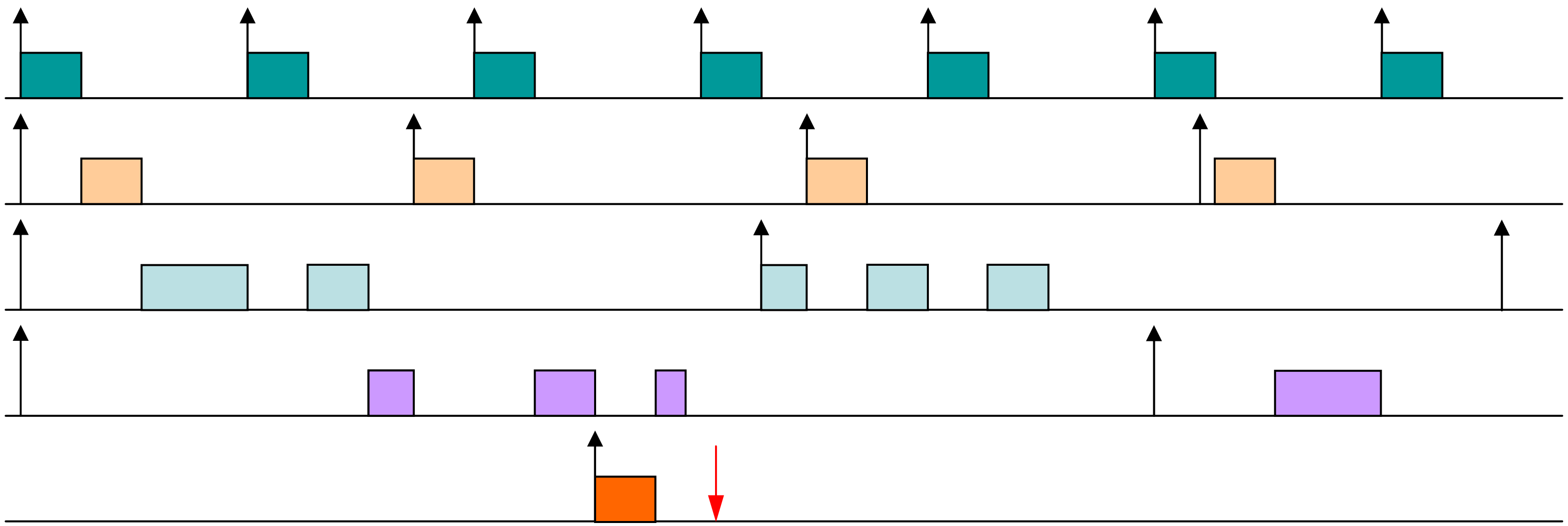
- Hierarchical design
- Many periodic activities running a different rates
- Many event-driven routines



Task scheduling

When more tasks are ready to execute, the order of execution is decided by the scheduler:





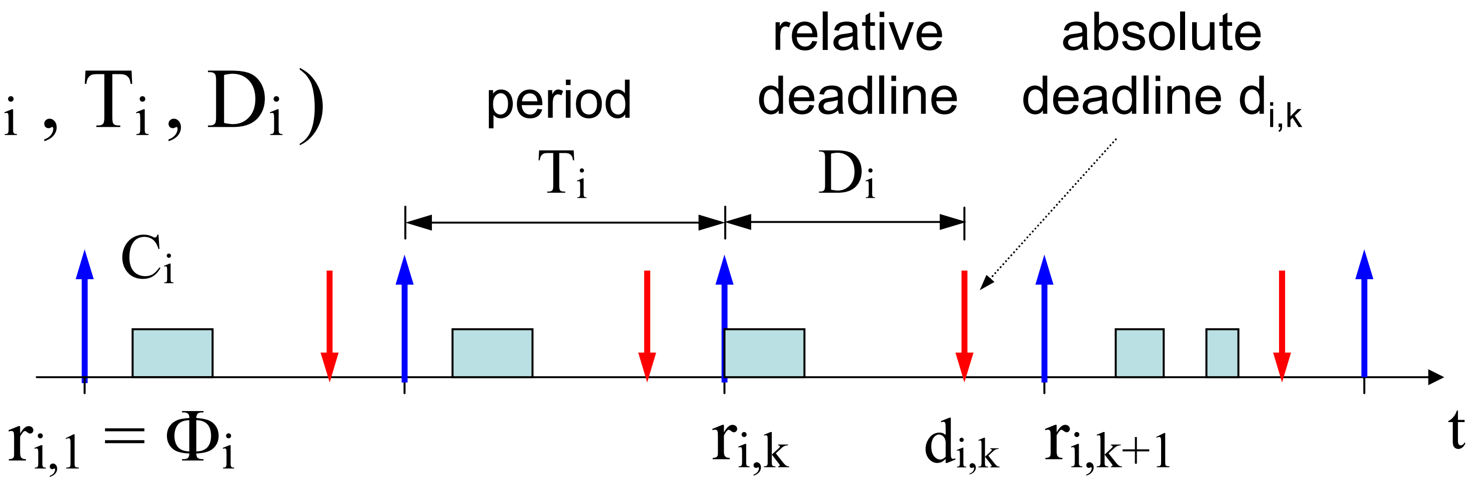
Importance of scheduling

- It affects task response times
- It affects delay and jitter in control loops
- It affects execution times (preemptions destroy cache data and prefetch queues)
- It can be used to cope with overload conditions
- It can be used to optimize resource usage
- It can be used to save energy in processors with voltage scaling (energy-aware scheduling)

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

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

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

Optimal Priority Assignments

- **Rate Monotonic (RM):**

  $p_i \propto 1/T_i$ (static) $D_i = T_i$

- **Deadline Monotonic (DM):**

  $p_i \propto 1/D_i$ (static) $D_i \neq T_i$

- **Earliest Deadline First (EDF):**

  $p_i \propto 1/d_i$ (dynamic) $d_{i,k} = r_{i,k} + D_i$

Basic results

Assumptions:
- Independent tasks
- $\Phi_i = 0$ $D_i = T_i$

In 1973, Liu & Layland proved that a set of ***n*** periodic tasks can be feasibly scheduled

under RM **if**

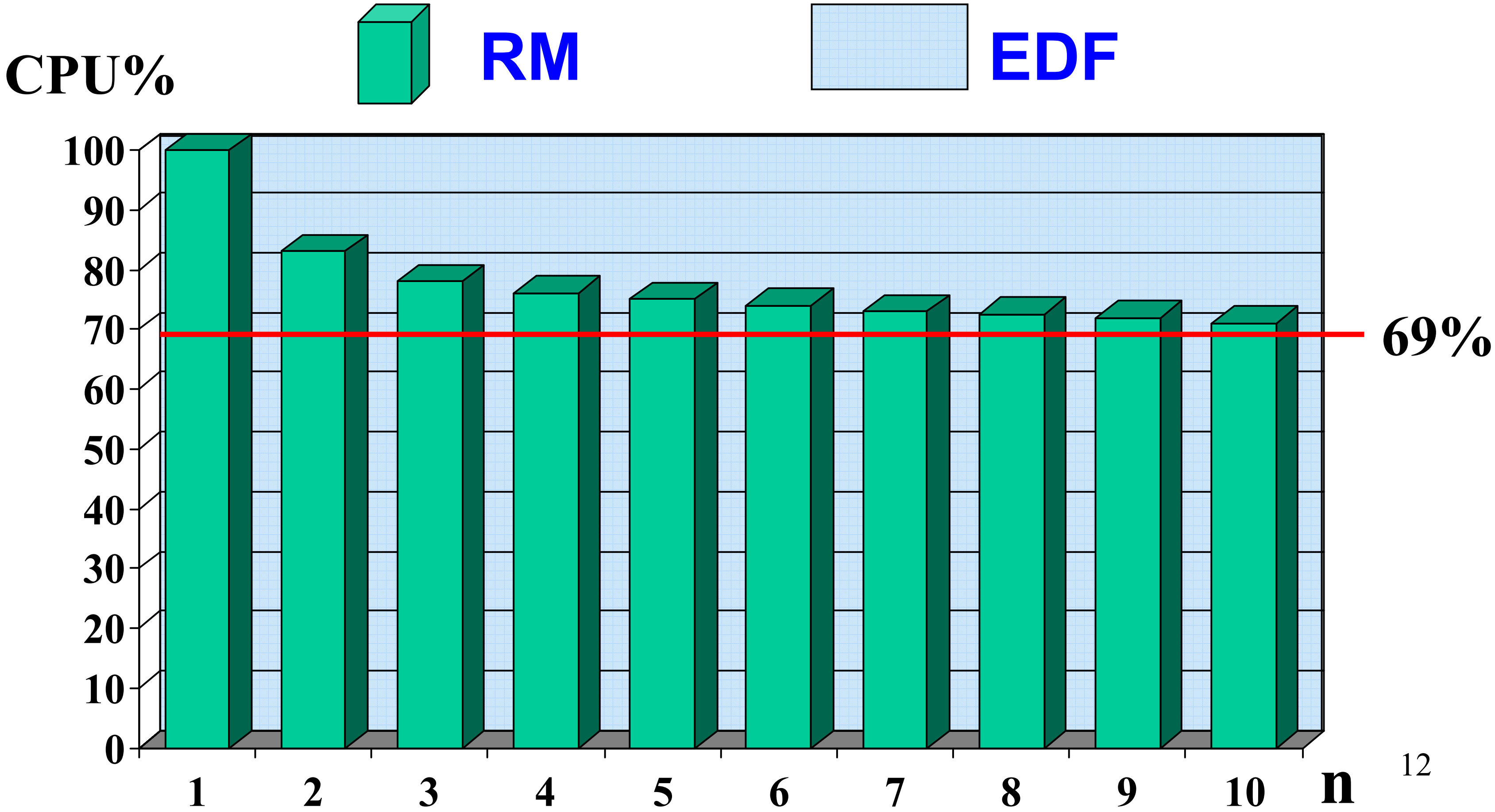
$$\sum_{i=1}^{n} \frac{C_i}{T_i} \le n\left(2^{1/n} - 1\right)$$

under EDF **if and only if**

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \le 1$$

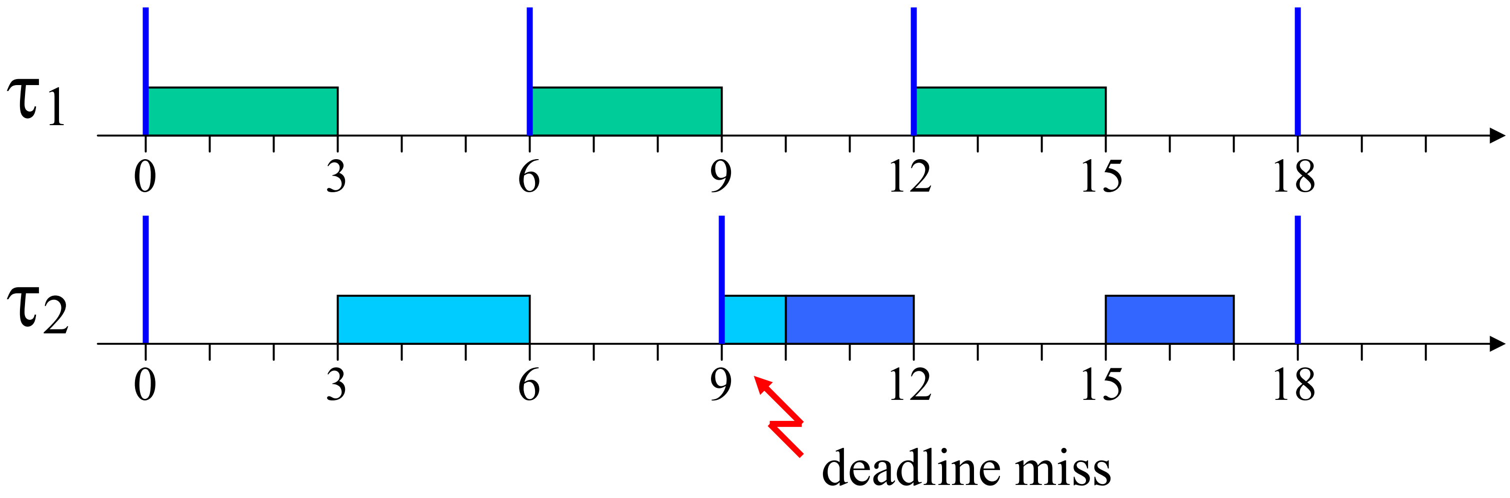
Schedulability bound

for $n \to \infty$ $\quad U_{lub} \to \ln 2 \cong 0.69$

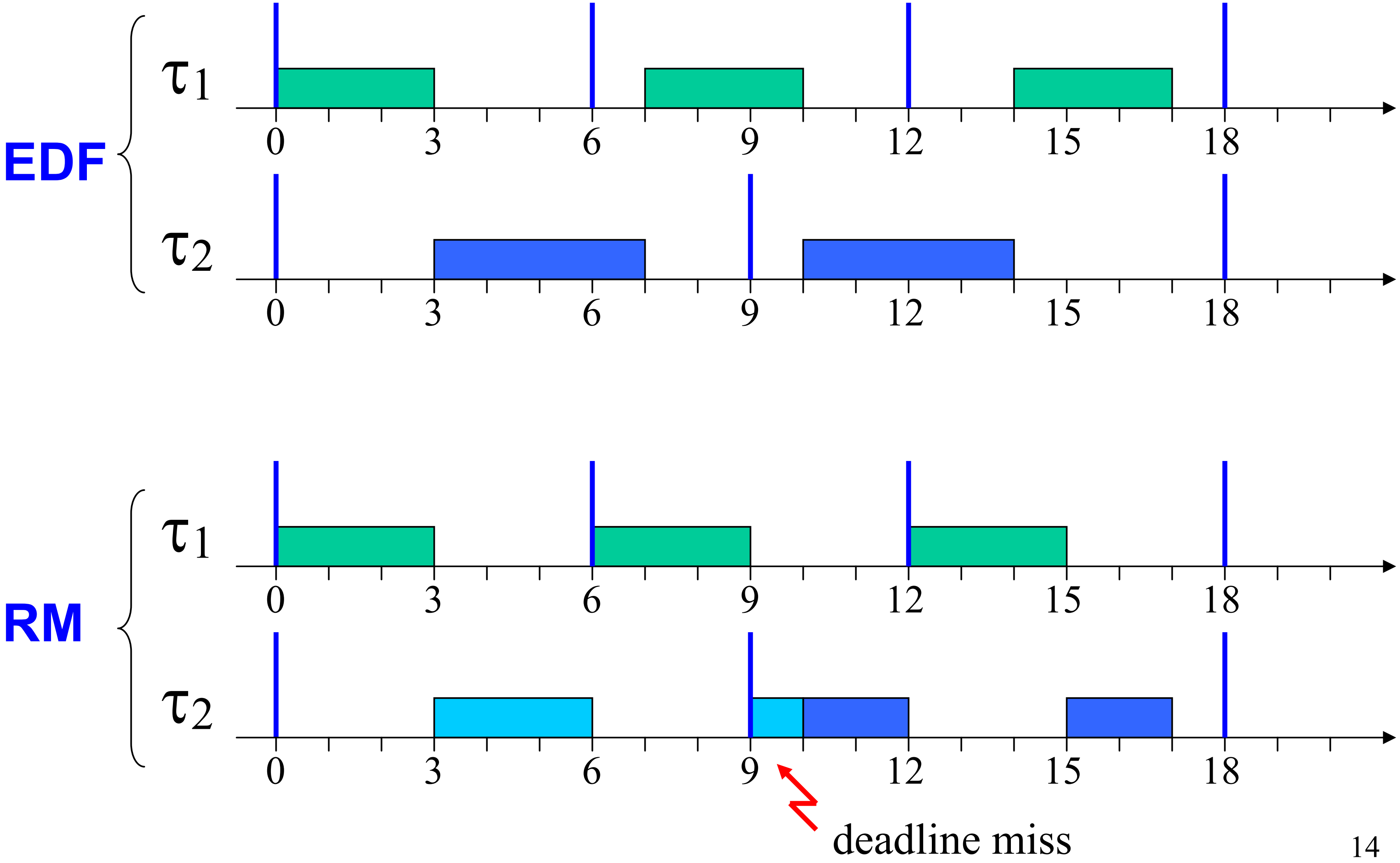


An unfeasible RM schedule

$$U_p = \frac{3}{6} + \frac{4}{9} = 0.944$$

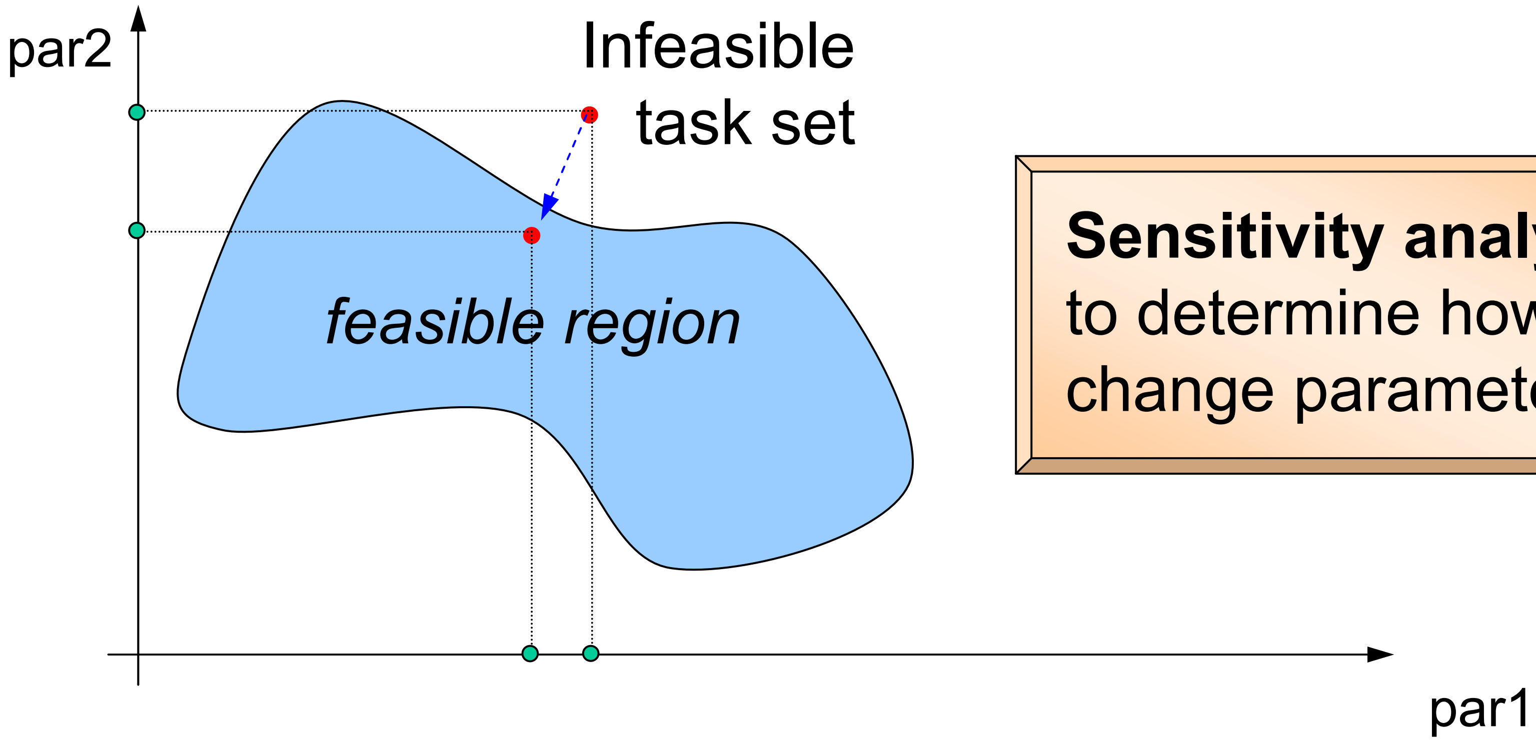


EDF vs. RM Schedule



Schedulability region

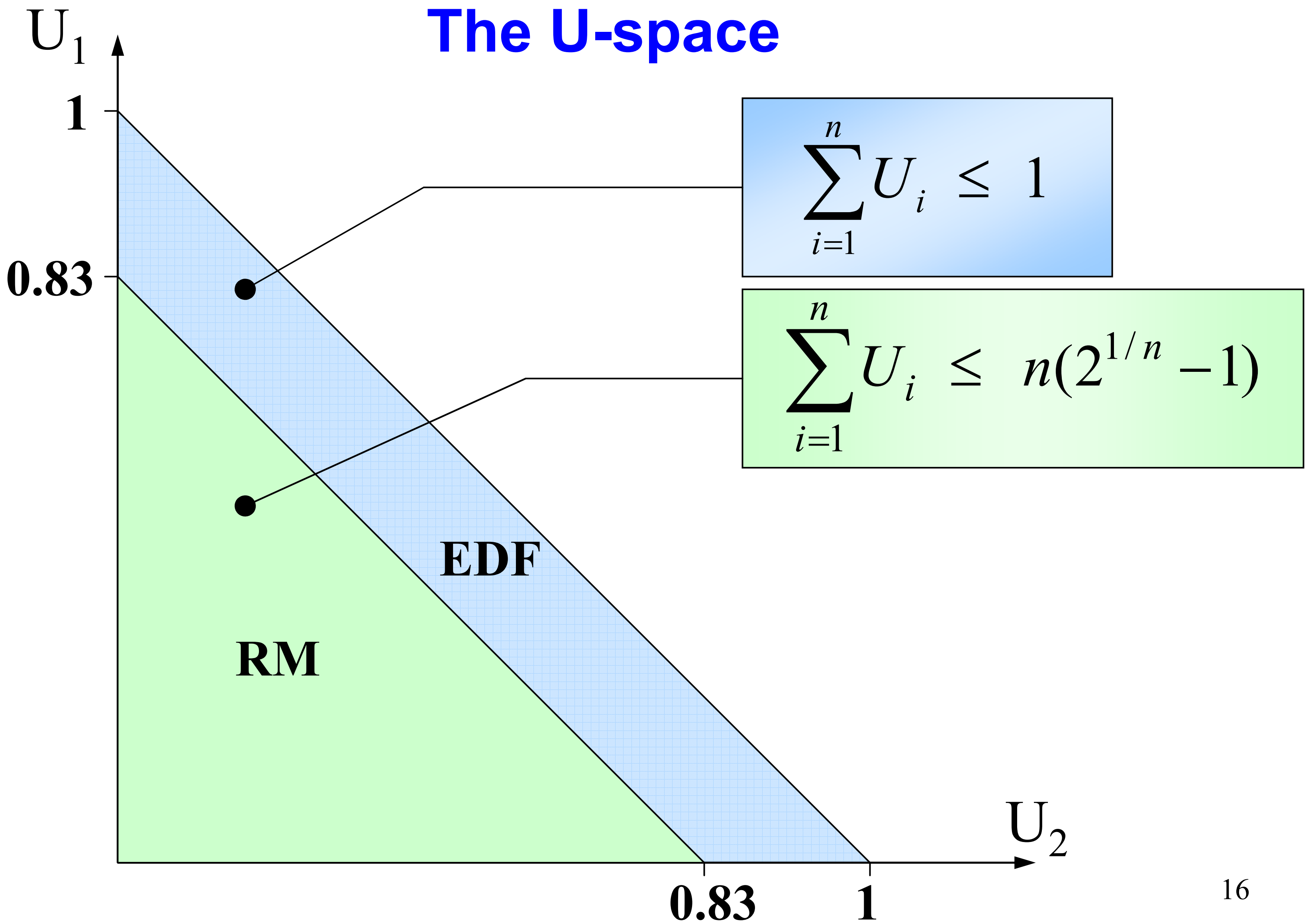
A more useful approach is to identify a region in the space of task parameters where the system is schedulable by an algorithm.



Sensitivity analysis
to determine how to change parameters

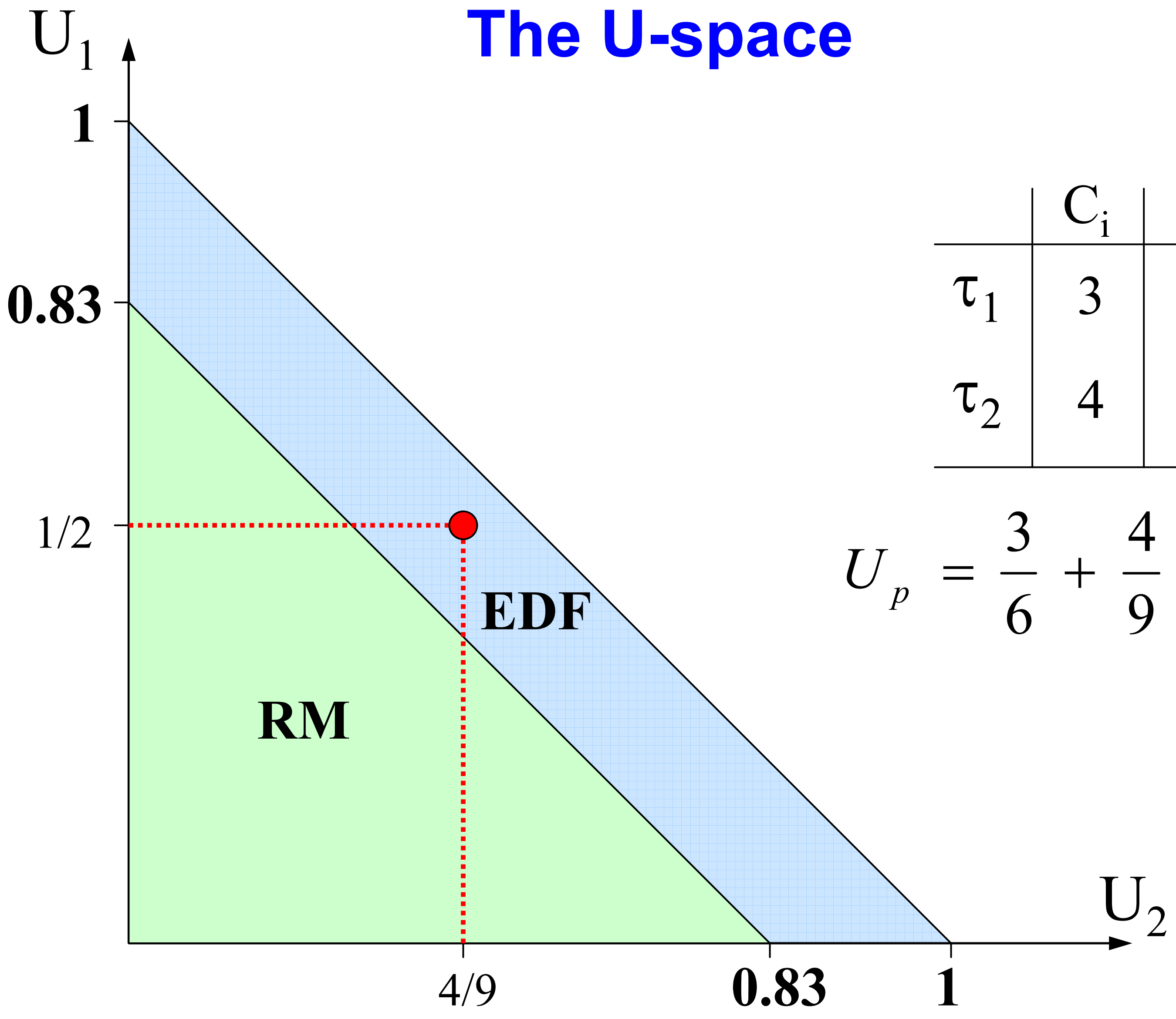
Schedulability region

The U-space



Schedulability region

The U-space



	C_i	T_i
τ_1	3	6
τ_2	4	9

$$U_p = \frac{3}{6} + \frac{4}{9} = 0.94$$

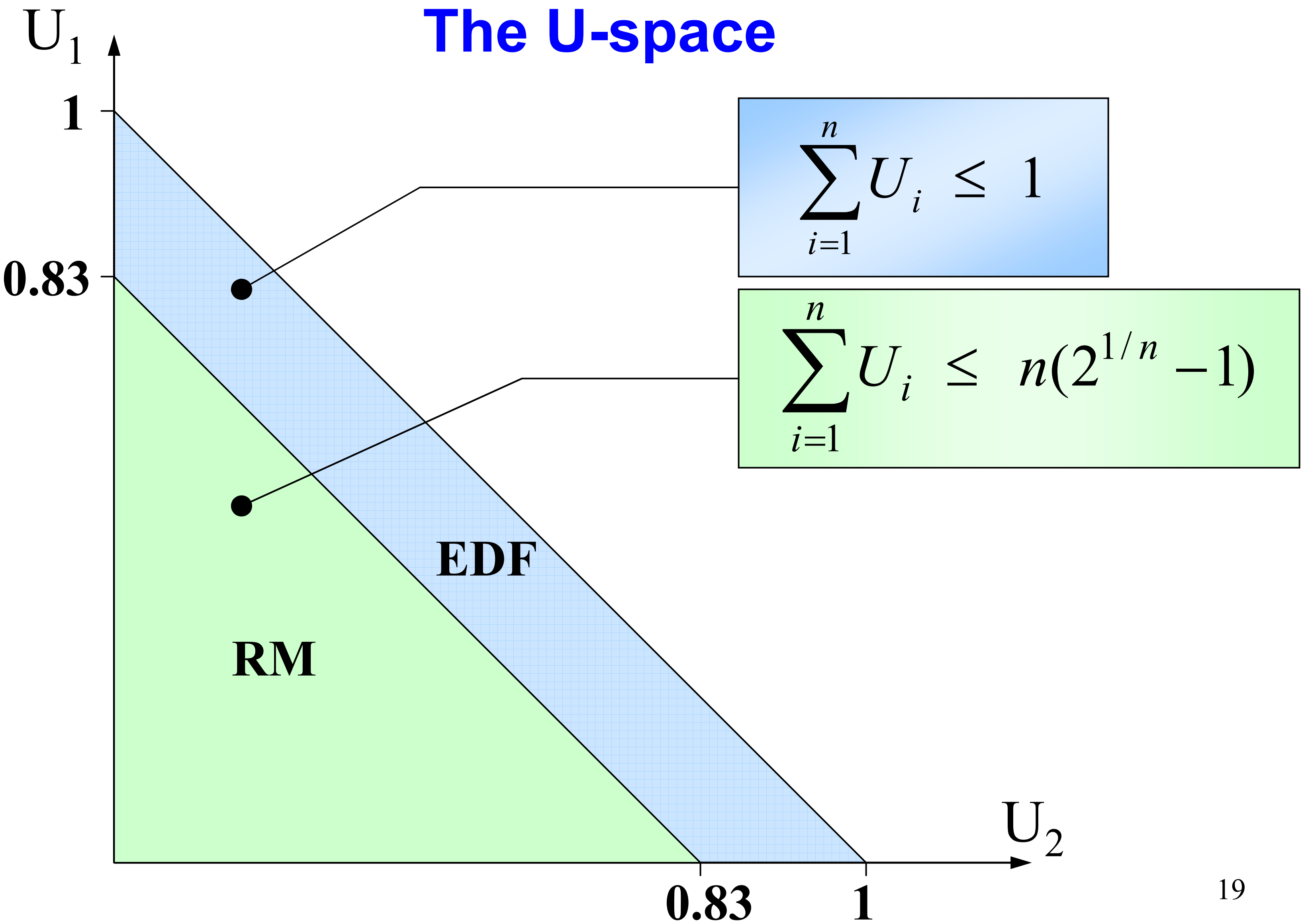
The Hyperbolic Bound

- In 2000, **Bini et al.** proved that a set of *n* periodic tasks is schedulable with RM if:

$$\prod_{i=1}^{n} (U_i + 1) \leq 2$$

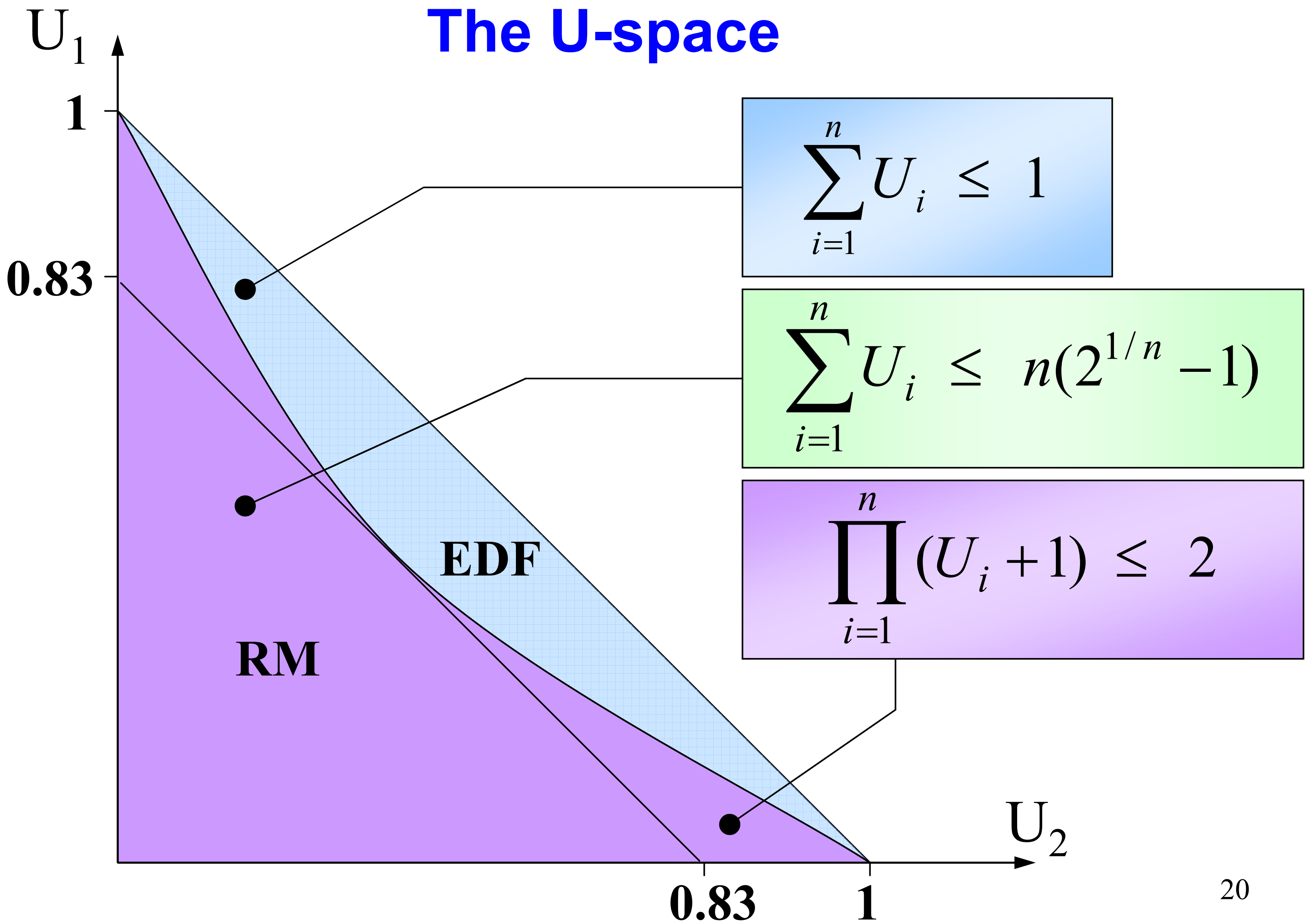
Schedulability region

The U-space

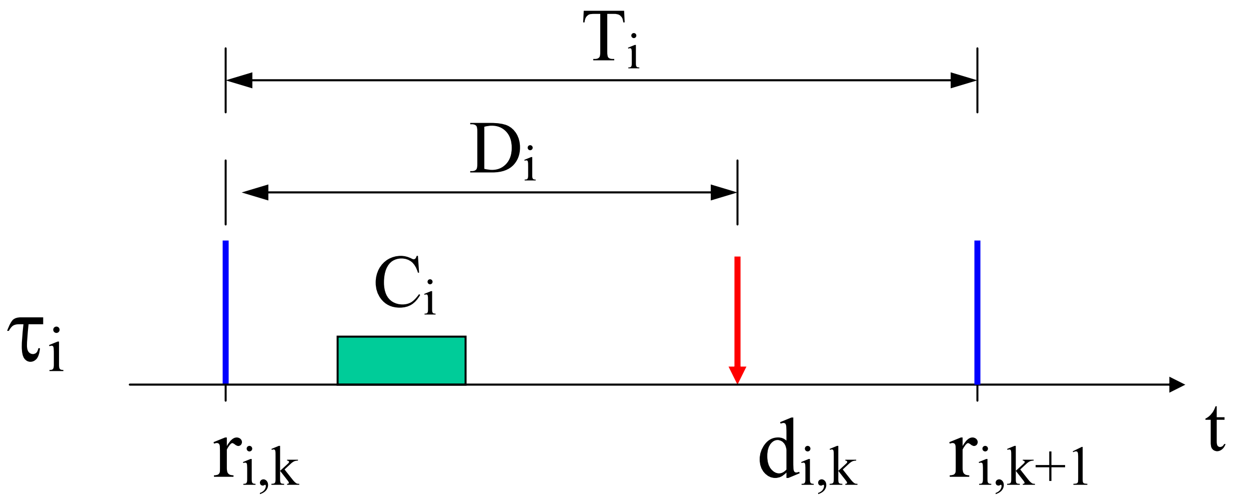


Schedulability region

The U-space



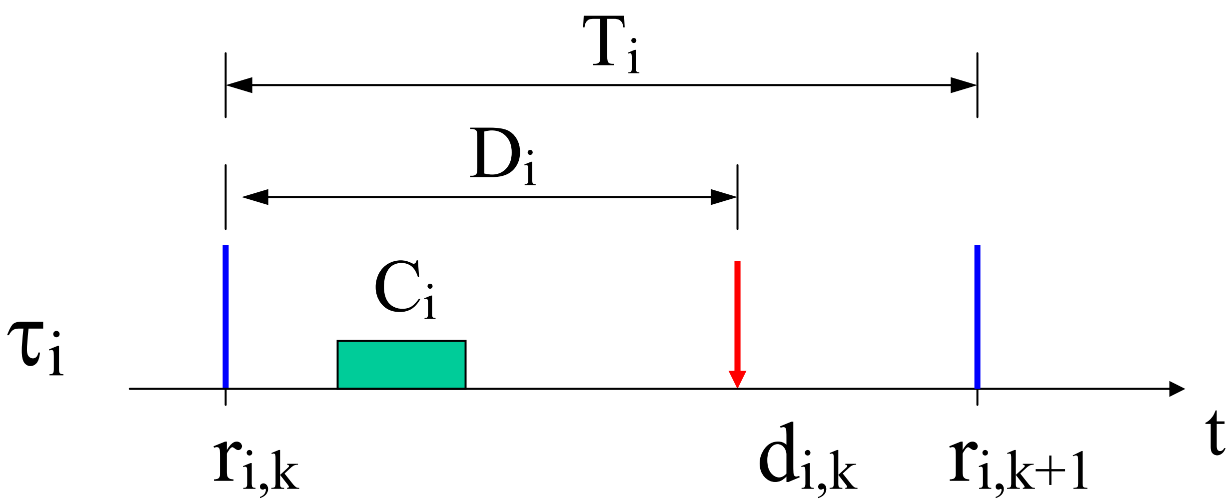
Handling tasks with D < T



Scheduling algorithms

- Deadline Monotonic: $\mathbf{p_i \propto 1/D_i}$ (static)
- Earliest Deadline First: $\mathbf{p_i \propto 1/d_i}$ (dynamic)

How to guarantee feasibility?



- **Fixed priority**: Response Time Analysis (RTA)
- **EDF**: Processor Demand Criterion (PDC)

Response Time Analysis

[Audsley, 1990]

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

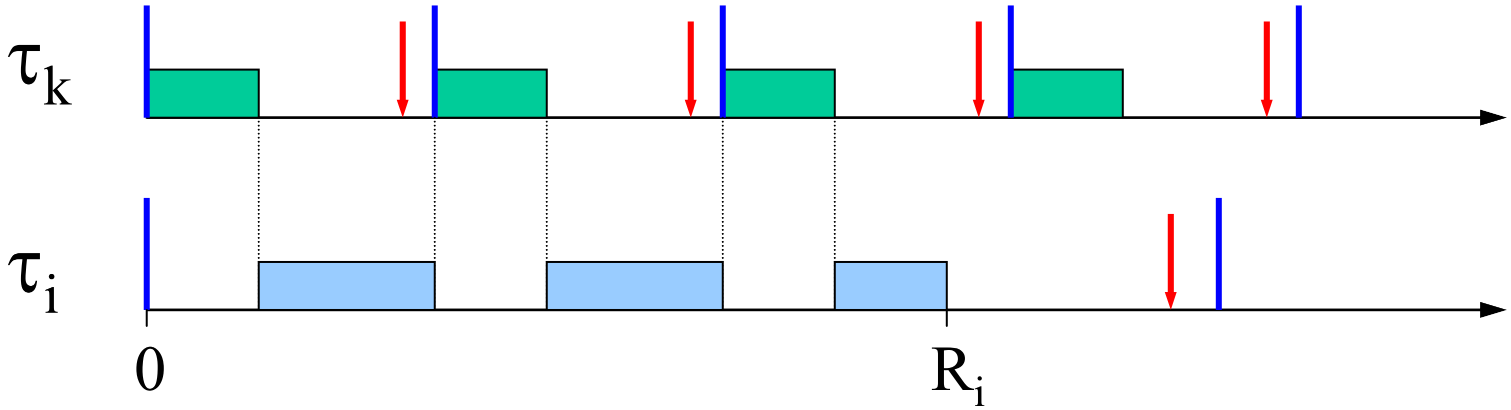
$$I_i = \sum_{D_k < D_i} C_k$$

- Compute its response time as

$$R_i = C_i + I_i$$

- Verify if $R_i \le D_i$

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:

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

Computing the response time

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

Iterative solution:

$$\begin{cases} R_i^0 = C_i \\ R_i^s = C_i + \sum_{k=1}^{i-1} \left\lceil \frac{R_i^{(s-1)}}{T_k} \right\rceil C_k \end{cases}$$

iterate until

$$R_i^s > R_i^{(s-1)}$$

Processor Demand Criterion

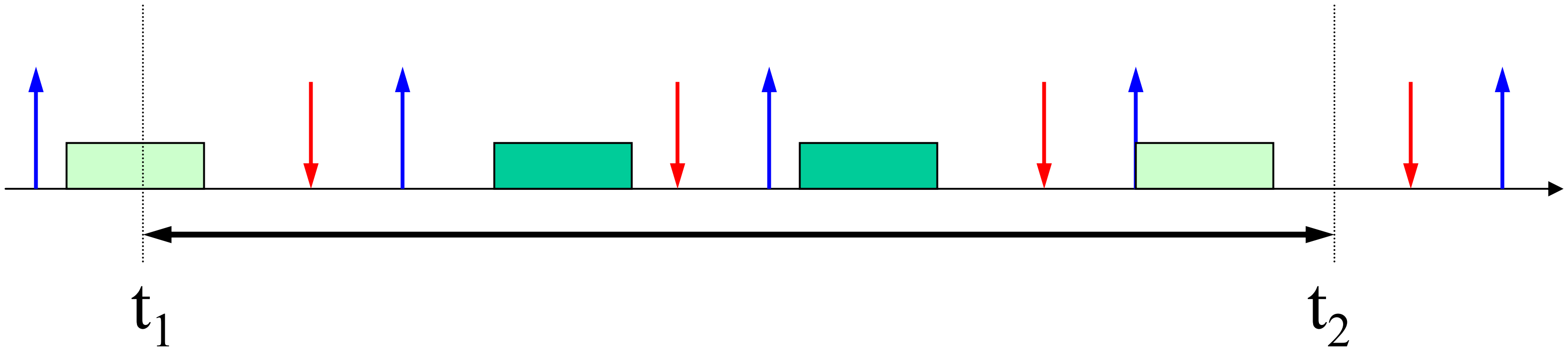
[Baruah, Howell, Rosier 1990]

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.

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

Processor Demand

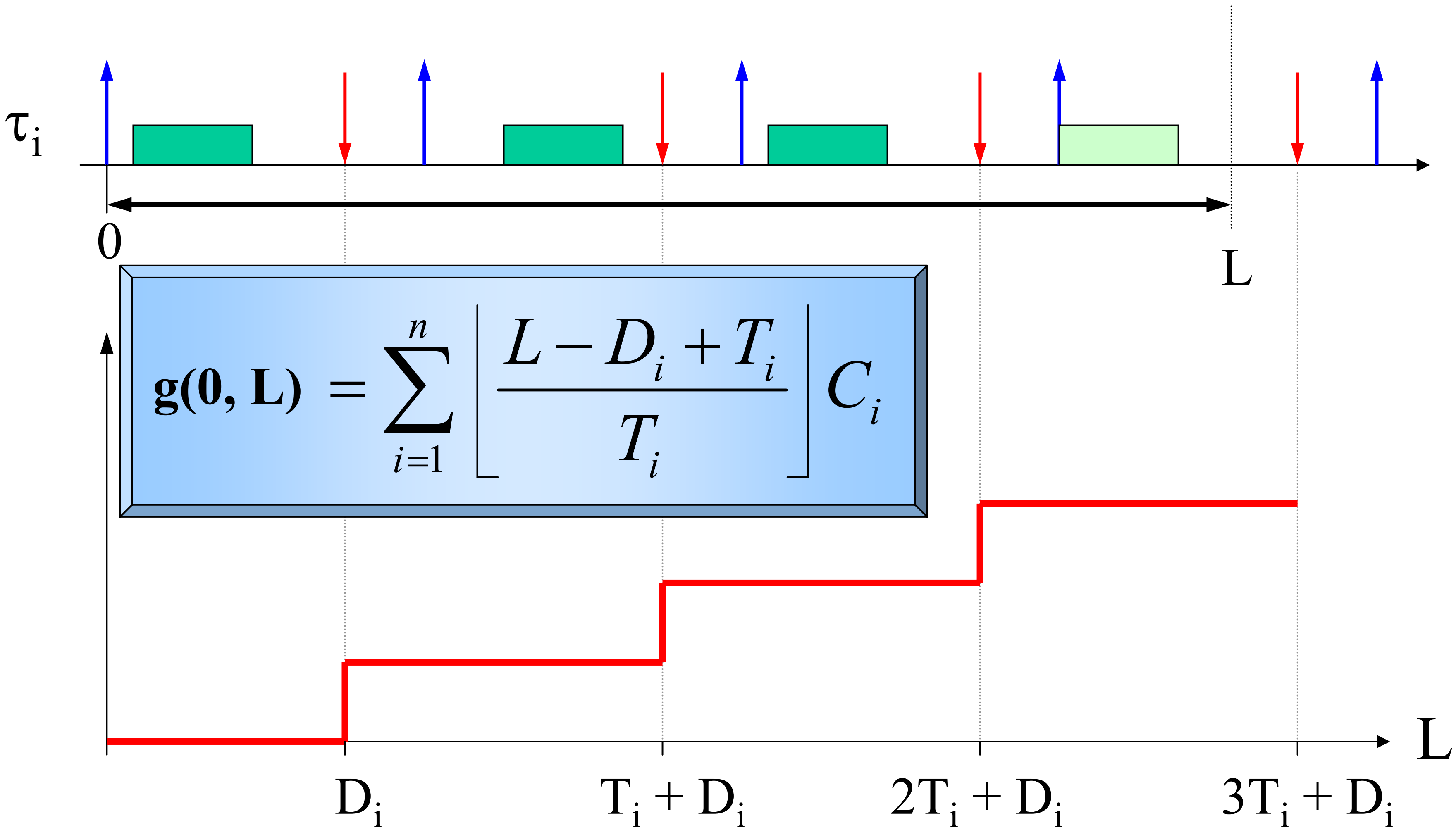


The demand in $[t_1, t_2]$ is the computation time of those jobs started at or after t_1 with deadline less than or equal to t_2:

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

Processor Demand

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

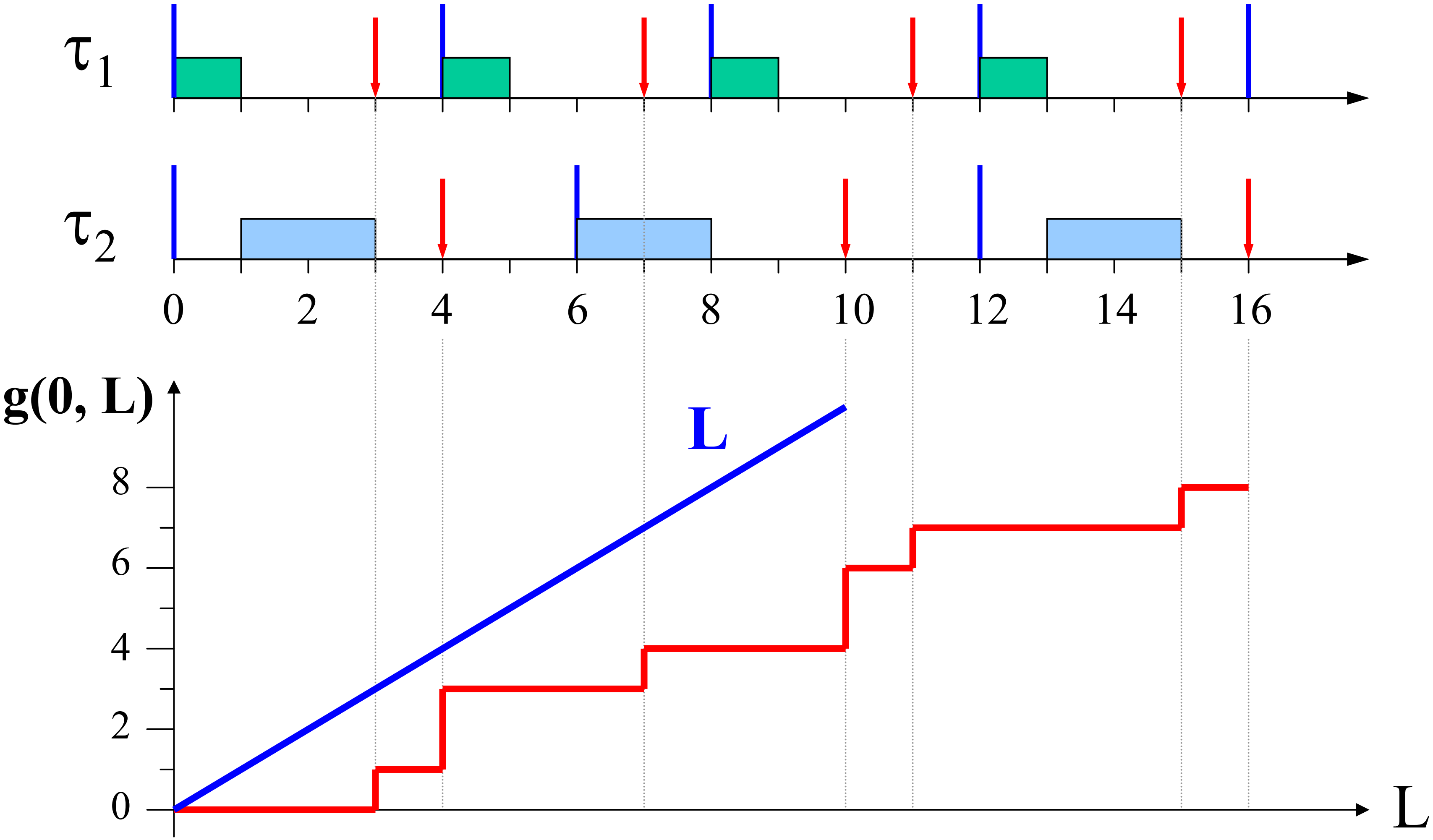


$$g(0, L) = \sum_{i=1}^{n} \left\lfloor \frac{L - D_i + T_i}{T_i} \right\rfloor C_i$$

Processor Demand Test

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

Example



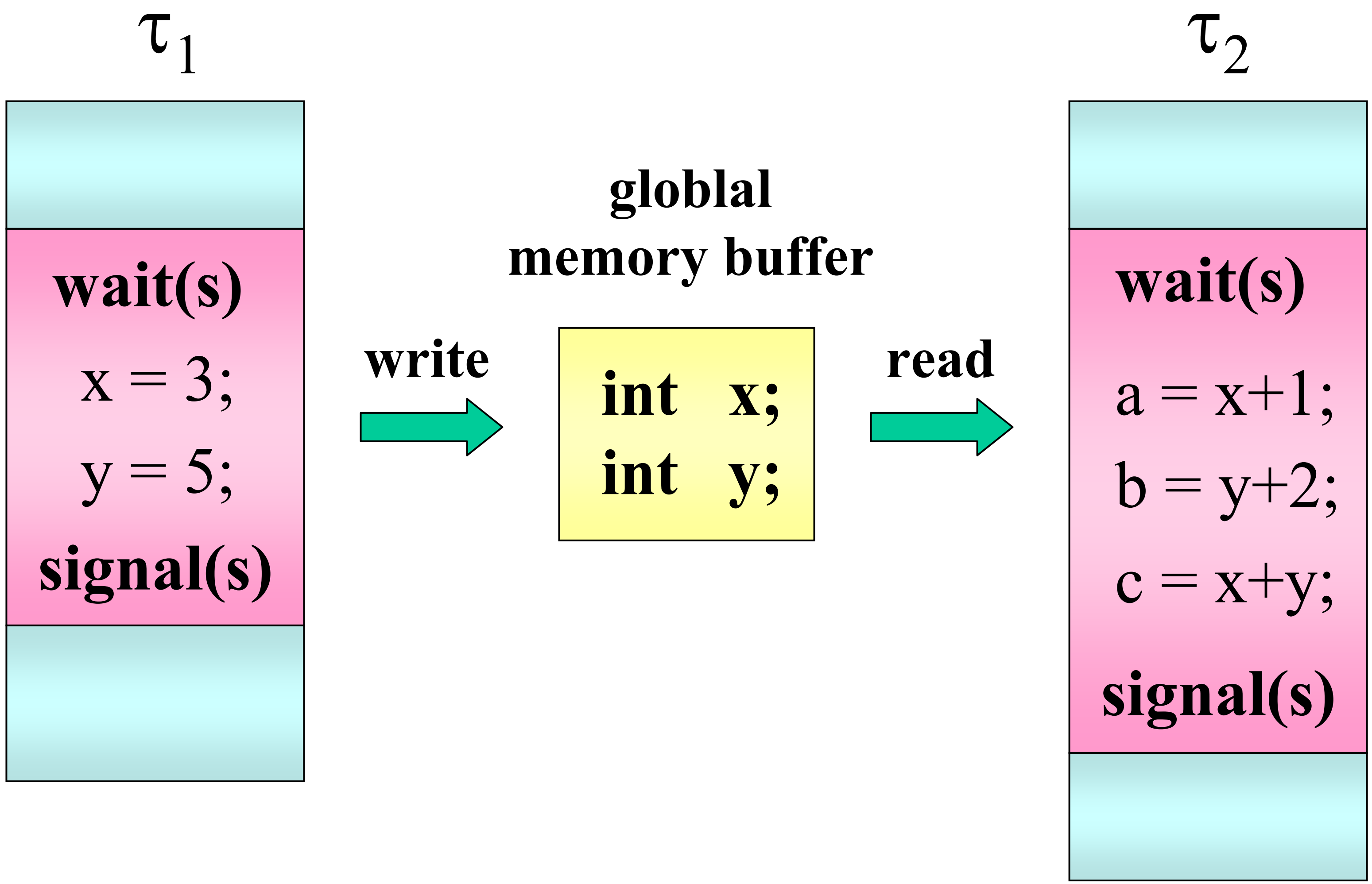
Summarizing: RM vs. EDF

	$D_i = T_i$	$D_i \le T_i$
RM	***Suff.: polynomial*** $O(n)$ LL: $\Sigma U_i \le n(2^{1/n} - 1)$ HB: $\Pi(U_i+1) \le 2$ ***Exact pseudo-polynomial*** RTA	***pseudo-polynomial*** Response Time Analysis $\forall i \quad R_i \le D_i$ $R_i = C_i + \sum_{k=1}^{i-1} \left\lceil \frac{R_i}{T_k} \right\rceil C_k$
EDF	***polynomial:*** $O(n)$ $\Sigma U_i \le 1$	***pseudo-polynomial*** Processor Demand Analysis $\forall L > 0, \quad g(0, L) \le L$

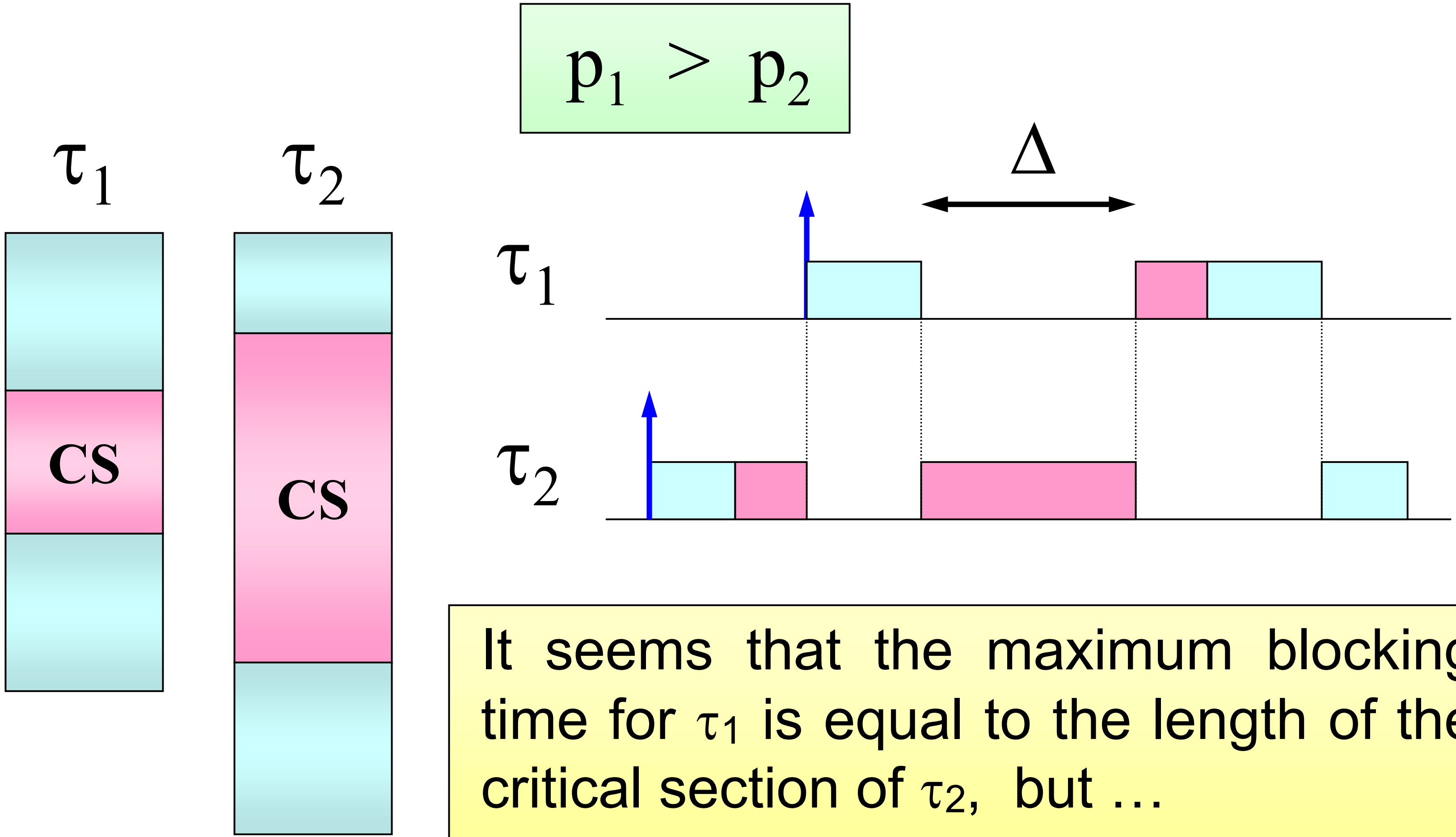
Handling shared resources

Problems caused by mutual exclusion

Critical sections

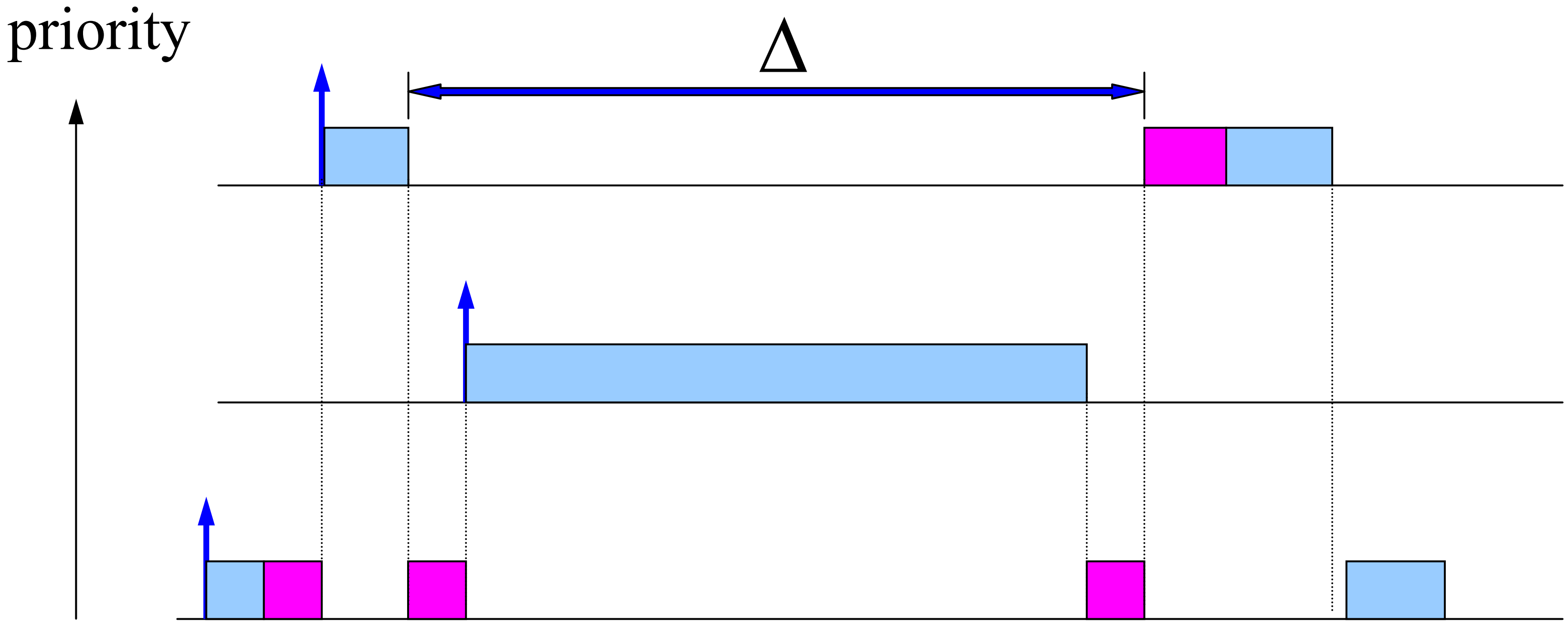


Blocking on a semaphore



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.

Resource Access Protocols

Under fixed priorities

- Non Preemptive Protocol (NPP)
- Highest Locker Priority (HLP)
- Priority Inheritance (PIP) [Sha-Rajkumar-Lehoczcky, 90]
- Priority Ceiling (PCP) [Sha-Rajkumar-Lehoczcky, 90]

Under EDF

- Non Preemptive Protocol (NPP)
- Dynamic Priority Inheritance (D-PIP) [Spuri, 98]
- Dynamic Priority Ceiling (D-PCP) [Chen-Lin, 90]
- Stack Resource Policy (SRP) [Baker, 90]

Guarantee when D = T

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

Extended LL test:

RM

$$\forall i \quad \sum_{k=1}^{i-1} \frac{C_k}{T_k} + \frac{C_i + B_i}{T_i} \leq i\left(2^{1/i} - 1\right)$$

EDF

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

Guarantee when D ≤ T

Under DM a task set is schedulable if $\forall i \;\; R_i \leq D_i$

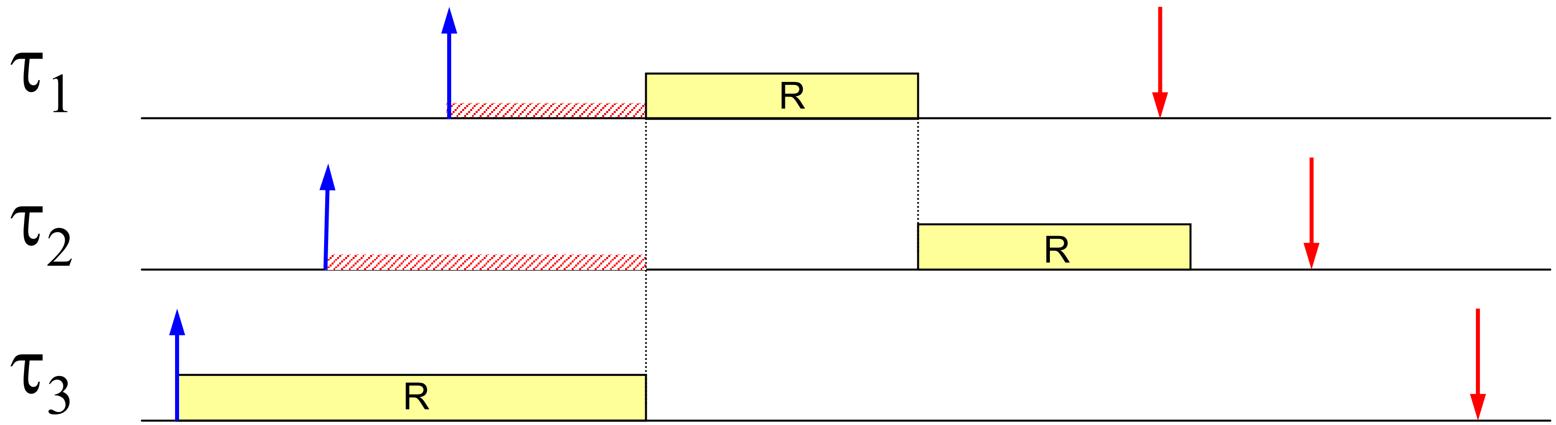
$$R_i = C_i + B_i + \sum_{k=1}^{i-1} \left\lceil \frac{R_i}{T_k} \right\rceil C_k$$

Under EDF a task set is schedulable if U < 1 and

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

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:

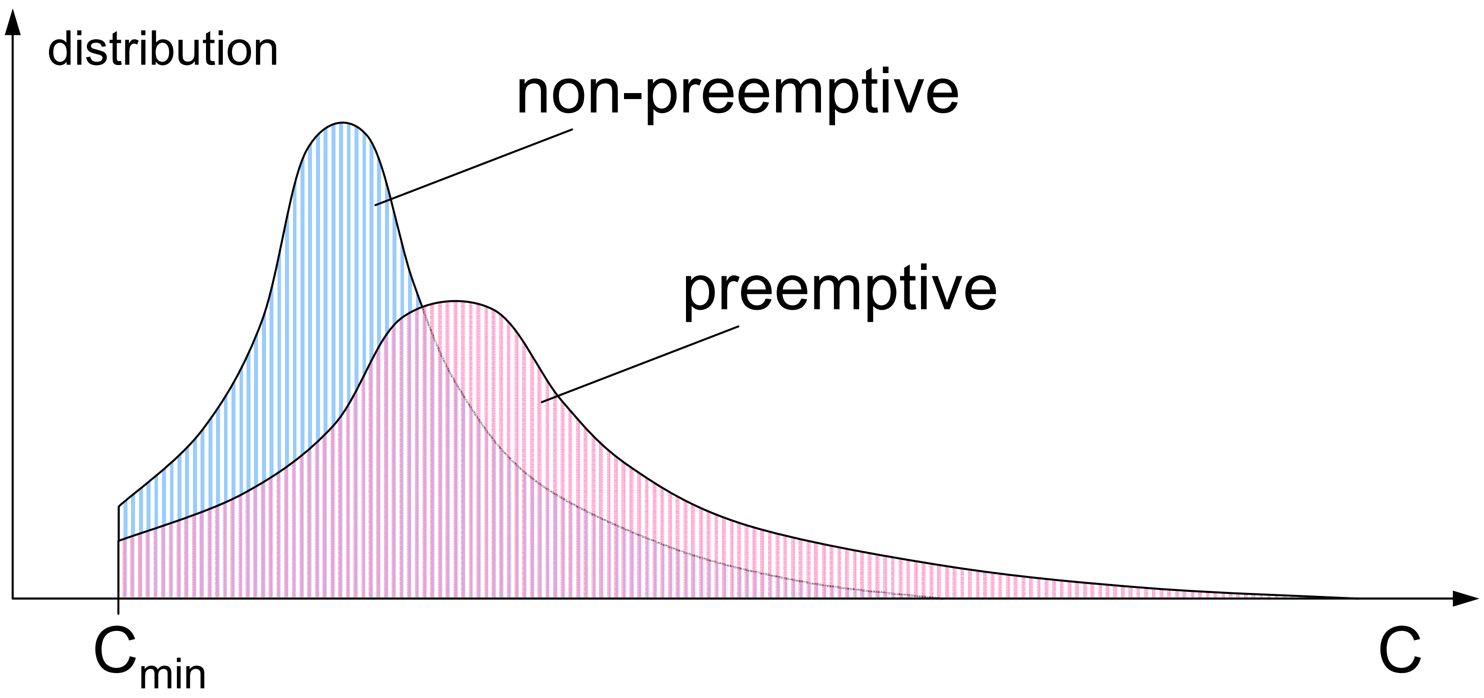
$$B_i = \max\{C_k : P_k < P_i\}$$

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

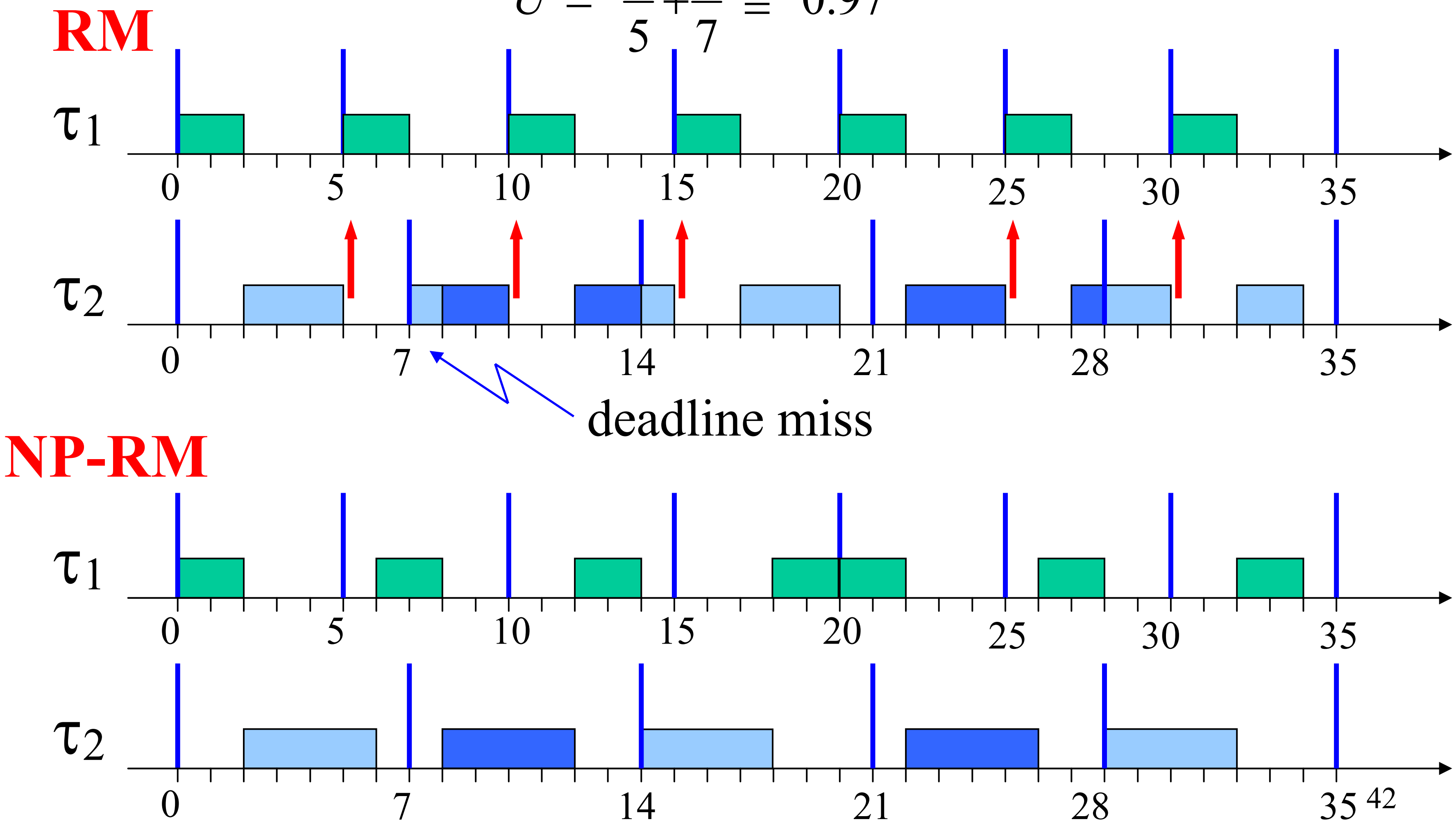
- As a consequence, task execution times are
  - Smaller
  - More predictable



Advantages of NP scheduling

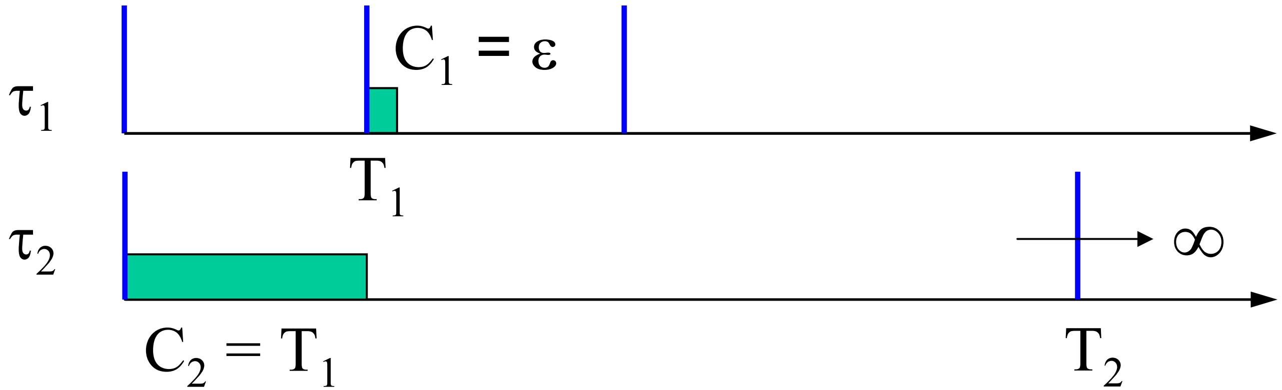
In fixed priority systems can improve schedulabiilty:

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



Disadvantages of NP scheduling

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

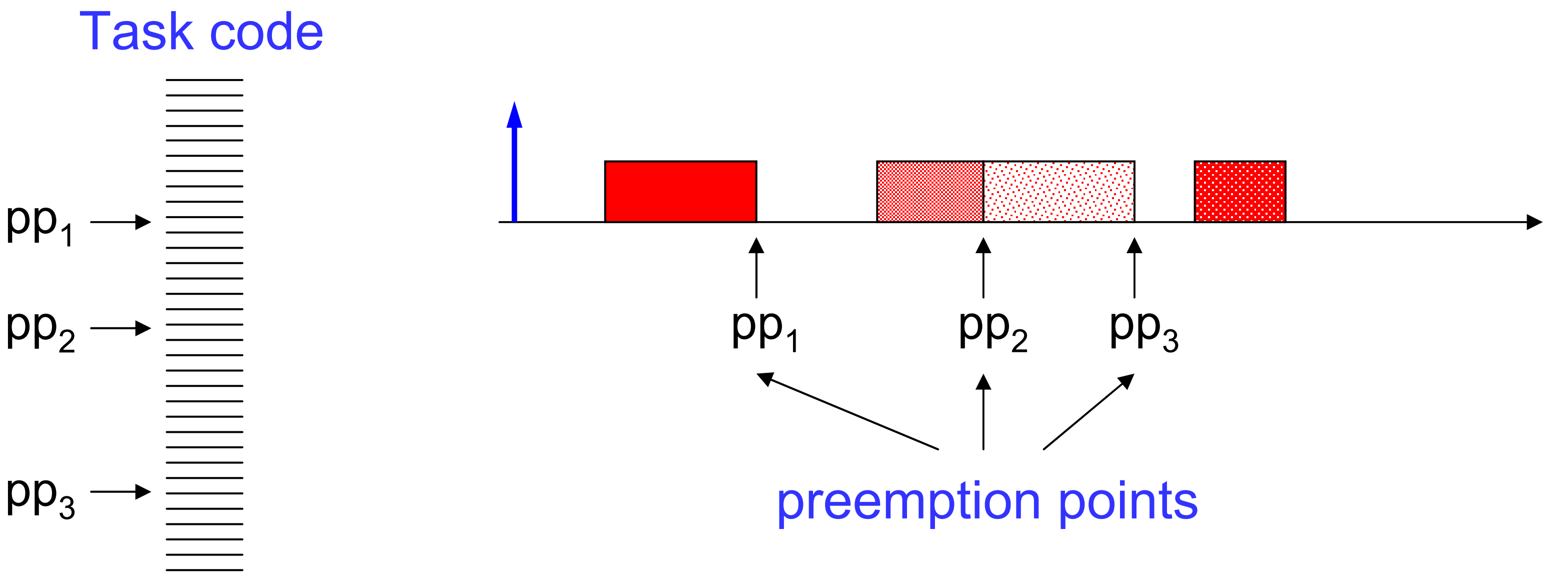


$$U = \frac{\varepsilon}{T_1} + \frac{C_2}{\infty} \longrightarrow 0$$

Trade-off solutions

Tunable Preemptive Systems

- Compute the longest non-preemptive section that allows a feasible schedule [Baruah-Bertogna, 08].
- Allow preemption only in certain points in the code.



Handling Jitter & Delay

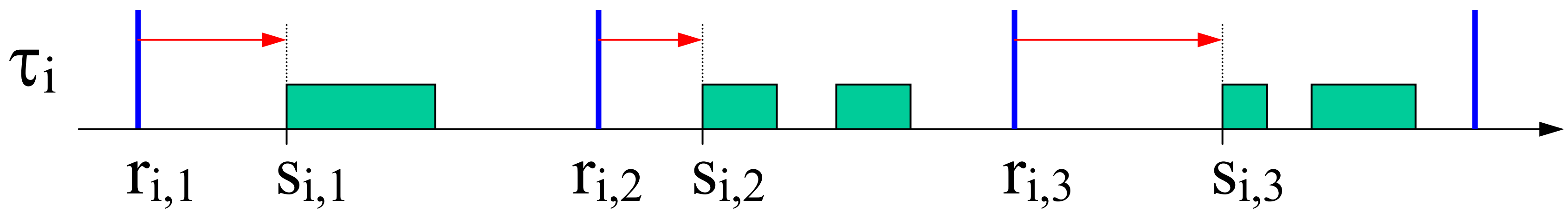
Jitter for an event

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): $INL_{i,k} = s_{i,k} - r_{i,k}$



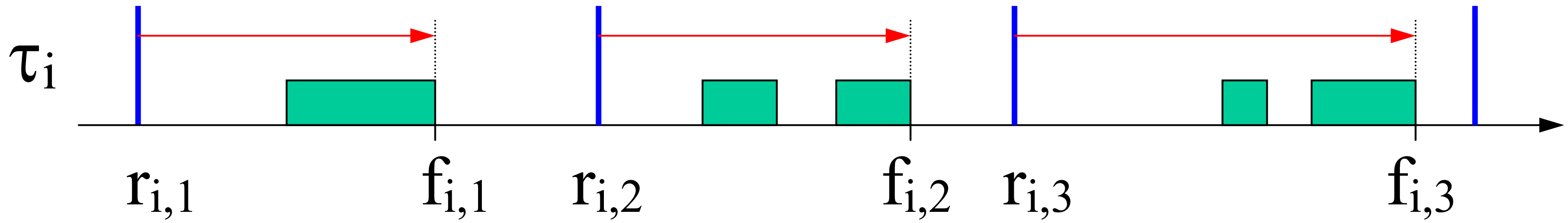
Start time Jitter (Input Jitter):

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): $R_{i,k} = f_{i,k} - r_{i,k}$



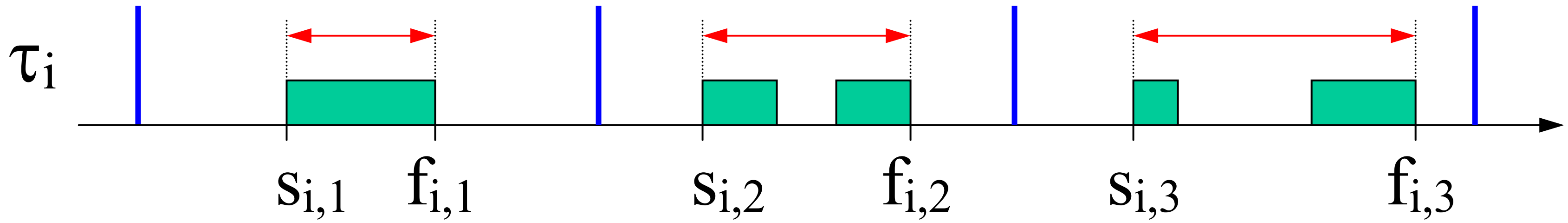
Response Time Jitter (Output Jitter):

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

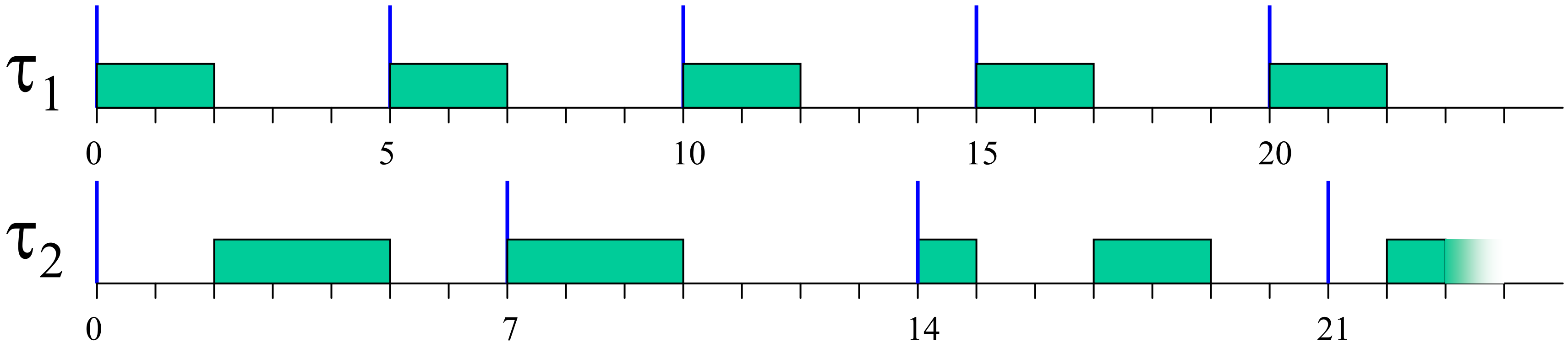


Input-Output Jitter:

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

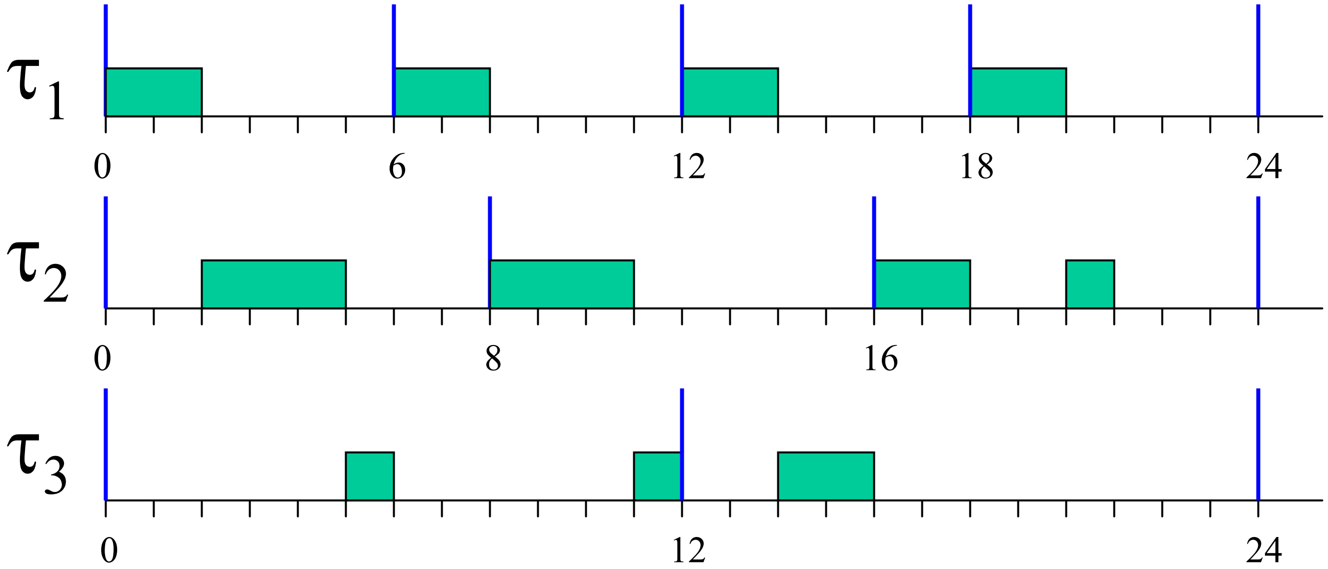
Relative: $$IOJ_i^{rel} = \max_k \left| (f_{i,k} - s_{i,k}) - (f_{i,k-1} - s_{i,k-1}) \right|$$

Cause of delays and jitter



- task parameters
- number of tasks
- total load
- activation phases
- scheduling algorithm

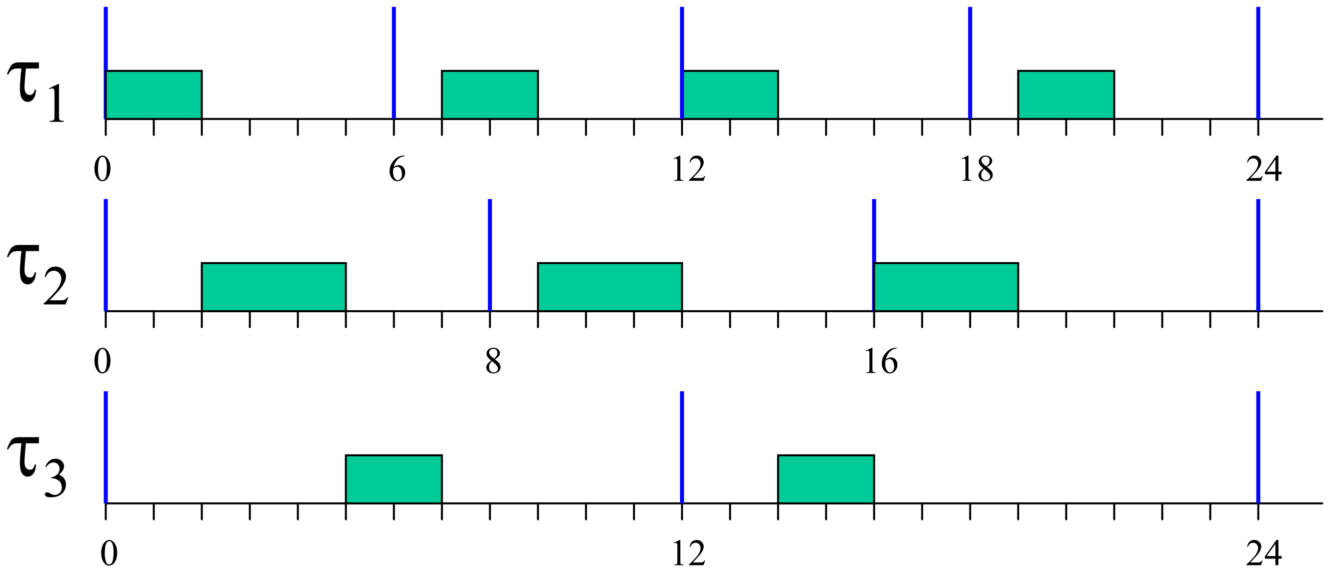
Jitter under RM



INJ	RTJ	IOJ
0	0	0
2	2	2
3	8	5

Low priority tasks experience very high delay and jitter

Jitter under EDF

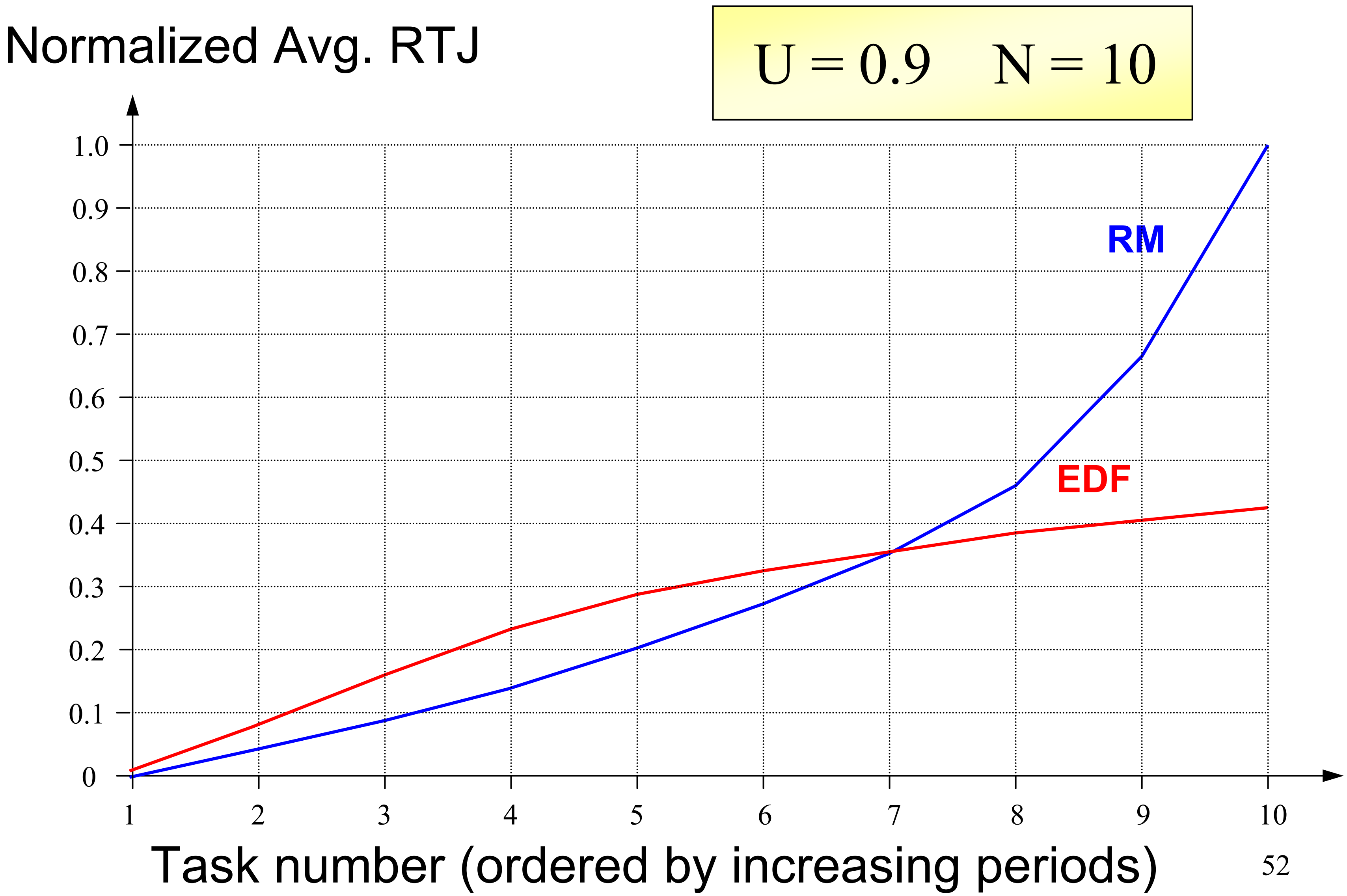


INJ	RTJ	IOJ
1	1	0
2	2	0
3	3	0

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

↓

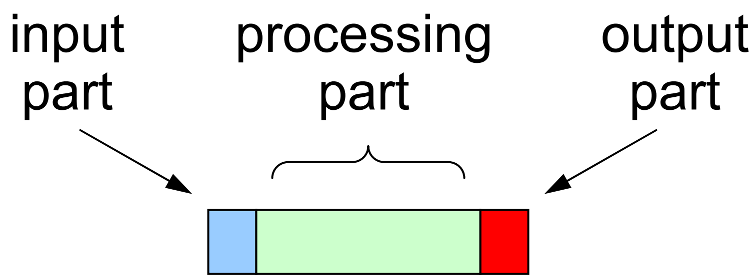
Hence we concentrate on reduction methods

Jitter Reduction methods

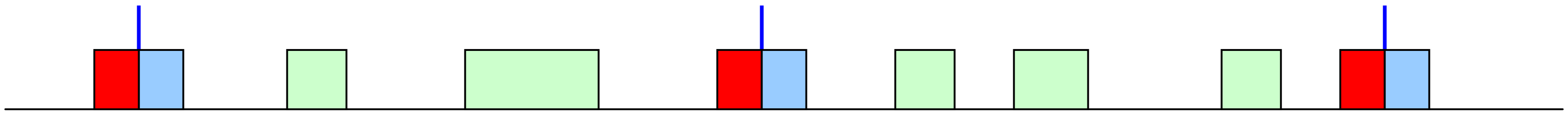
Three methods can be used to reduce the jitter caused by task interference:

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.

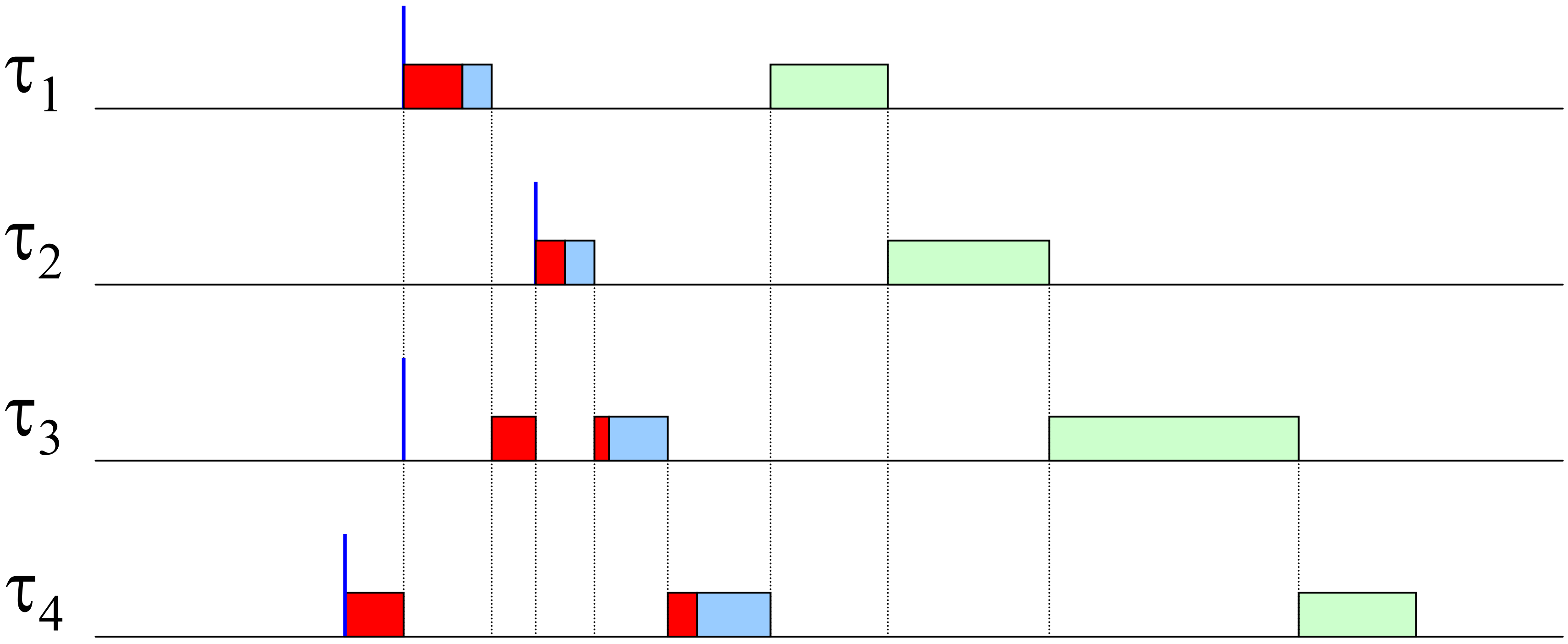
Reducing Jitter by Task Splitting

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.

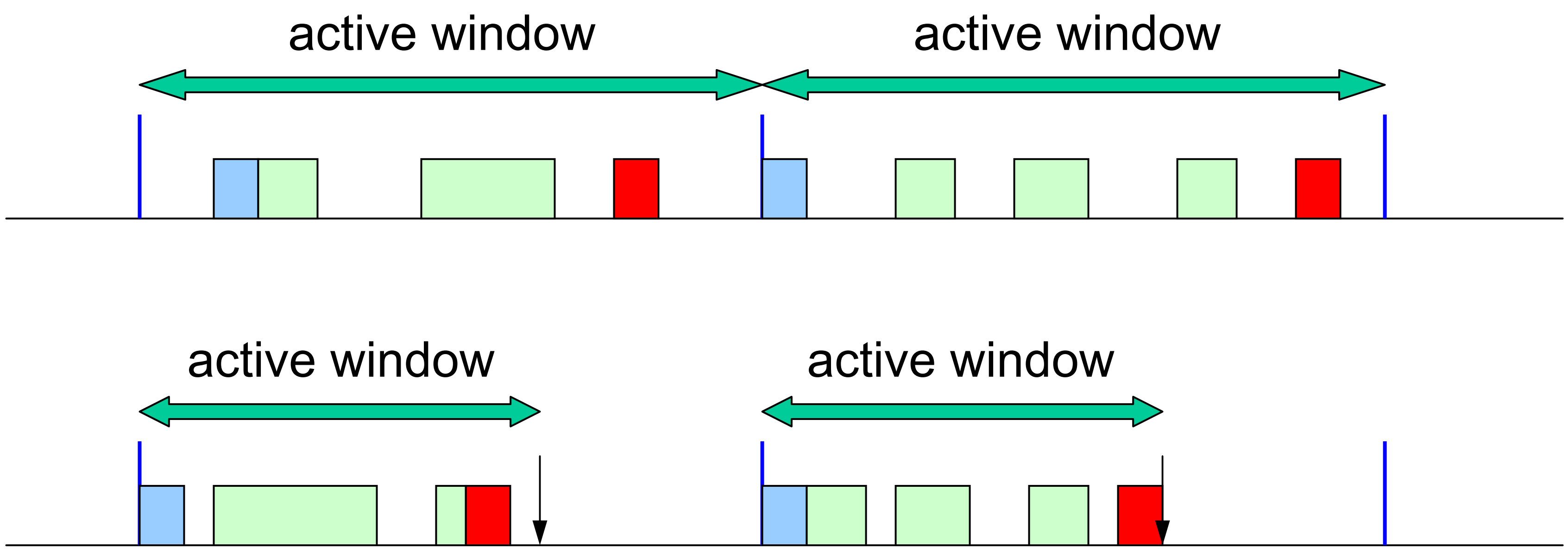
Reducing Jitter by Task Splitting

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

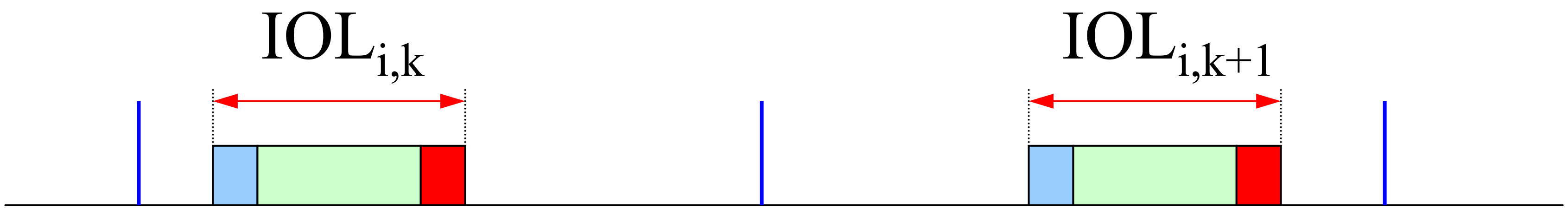
Disadvantages

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

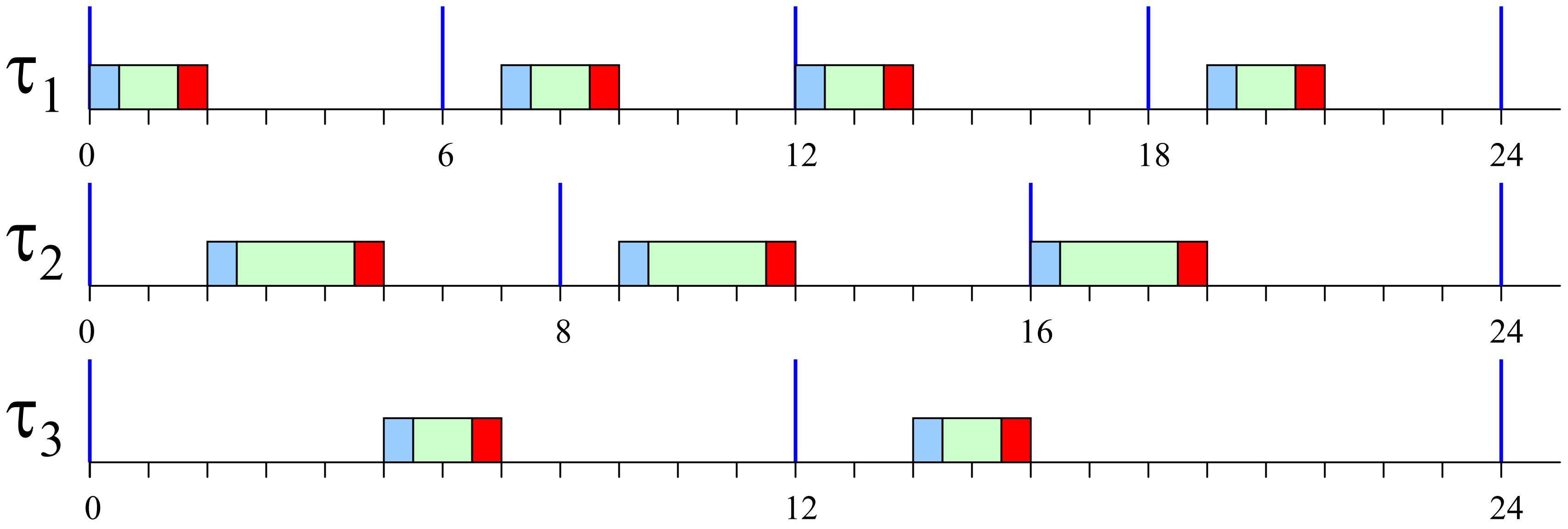
Disabling preemtions a task can be delayed, but once started cannot be interrupted:



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

Reducing Jitter by Non Preemption

Example with 3 tasks



Reducing Jitter by Non Preemption

Advantages

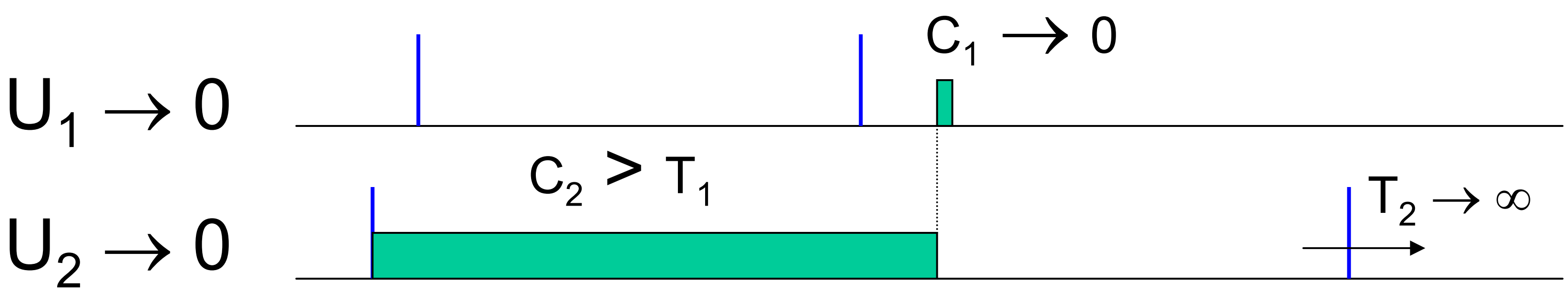
1. $IOJ_i = 0$ for all tasks;
2. $IOL_i = C_i$ for all tasks, simplifying the use of delay compensation techniques;
3. Non preemptive execution also simplifies resource management (there is no need to protect critical sections).
4. Non preemptive execution allows stack sharing.

Reducing Jitter by Non Preemption

Disadvantages

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

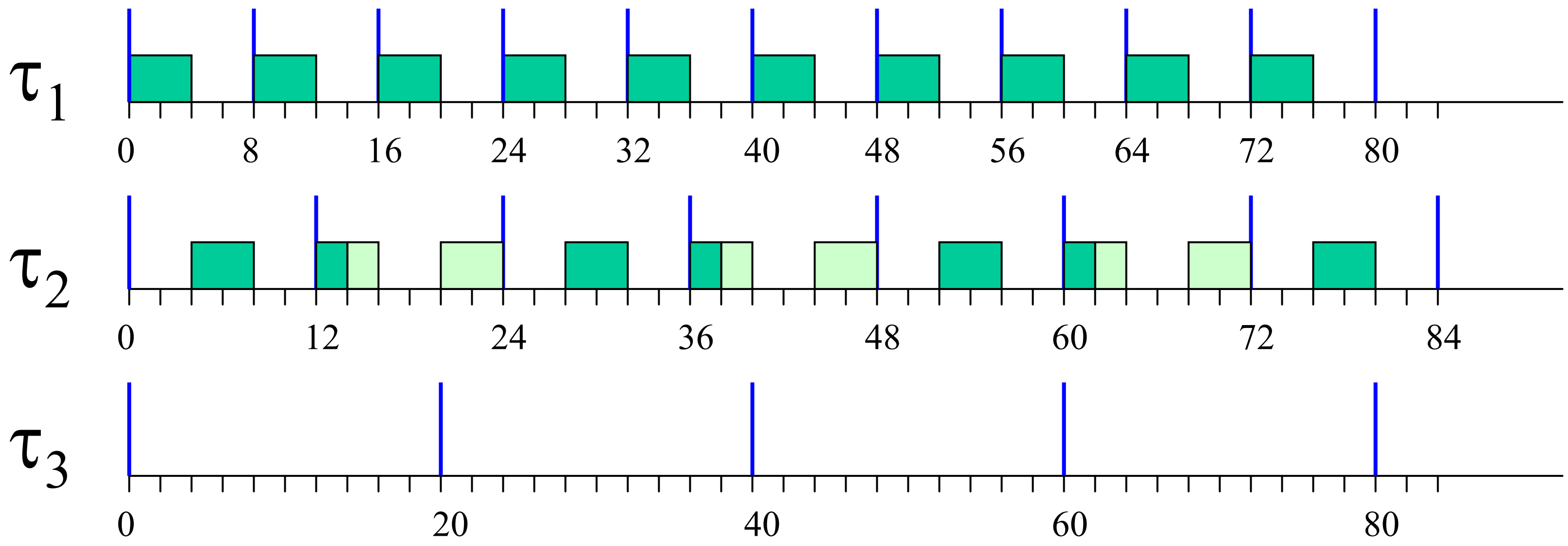
2. The utilization upper bound drops to zero:



Scheduling under overload conditions

RM under overloads

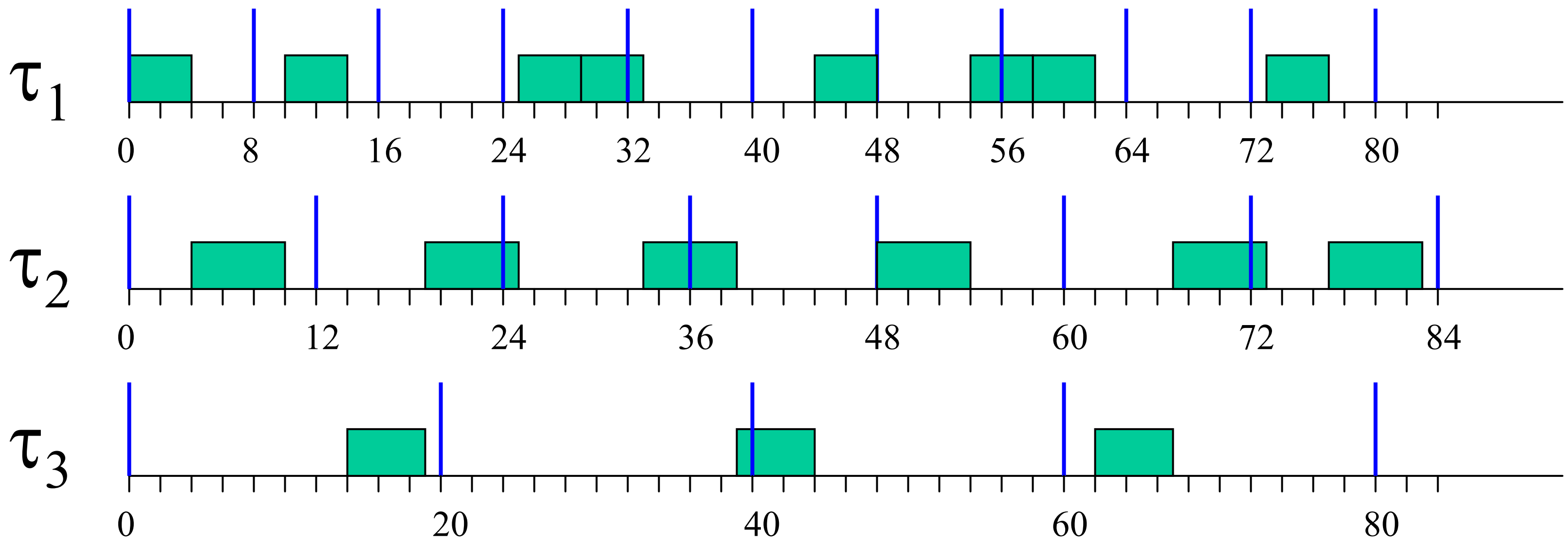
$$U = \frac{4}{8} + \frac{6}{12} + \frac{5}{20} = 1.25$$



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

EDF under overloads

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



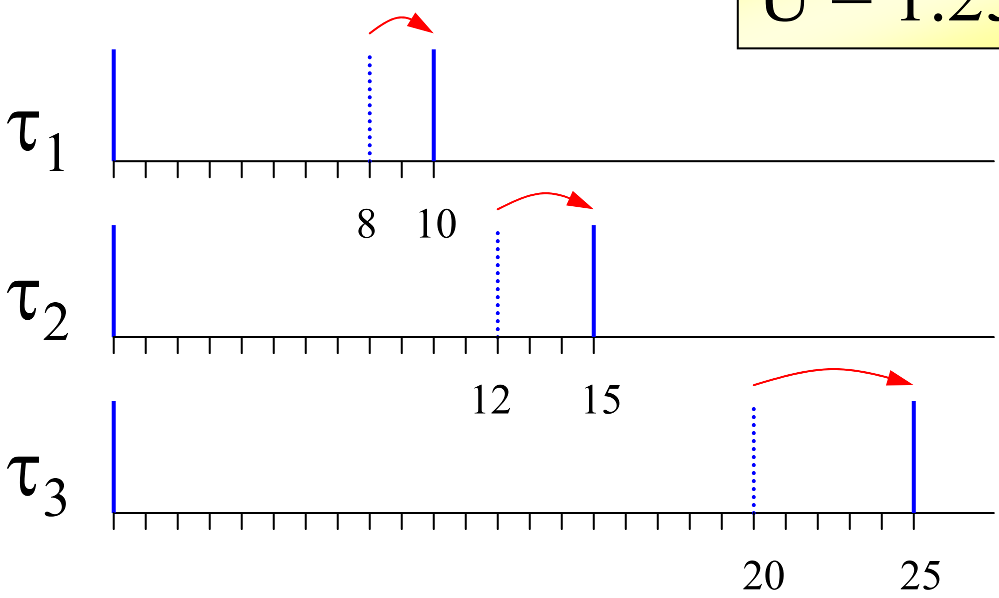
- All tasks execute at a slower rate
- No task is blocked

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

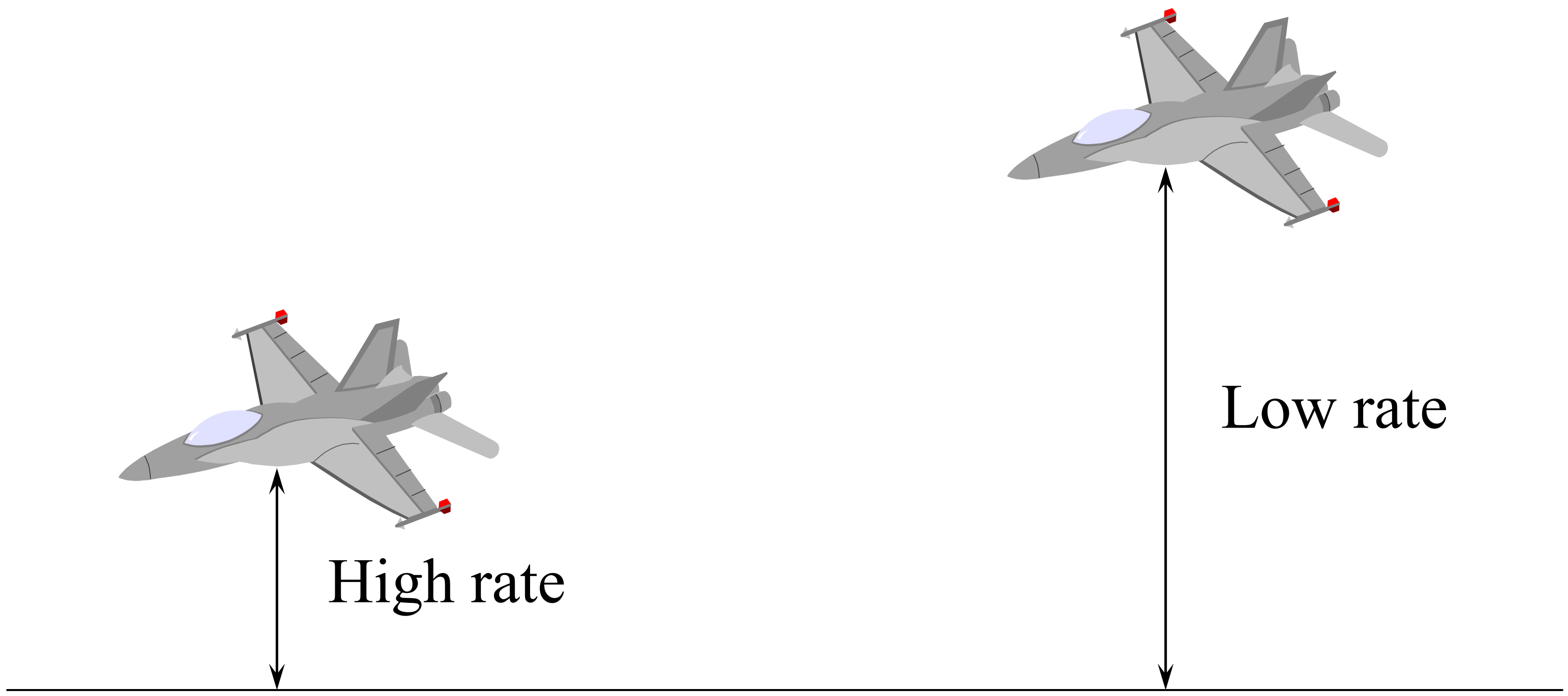
Relaxing timing constraints

- 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

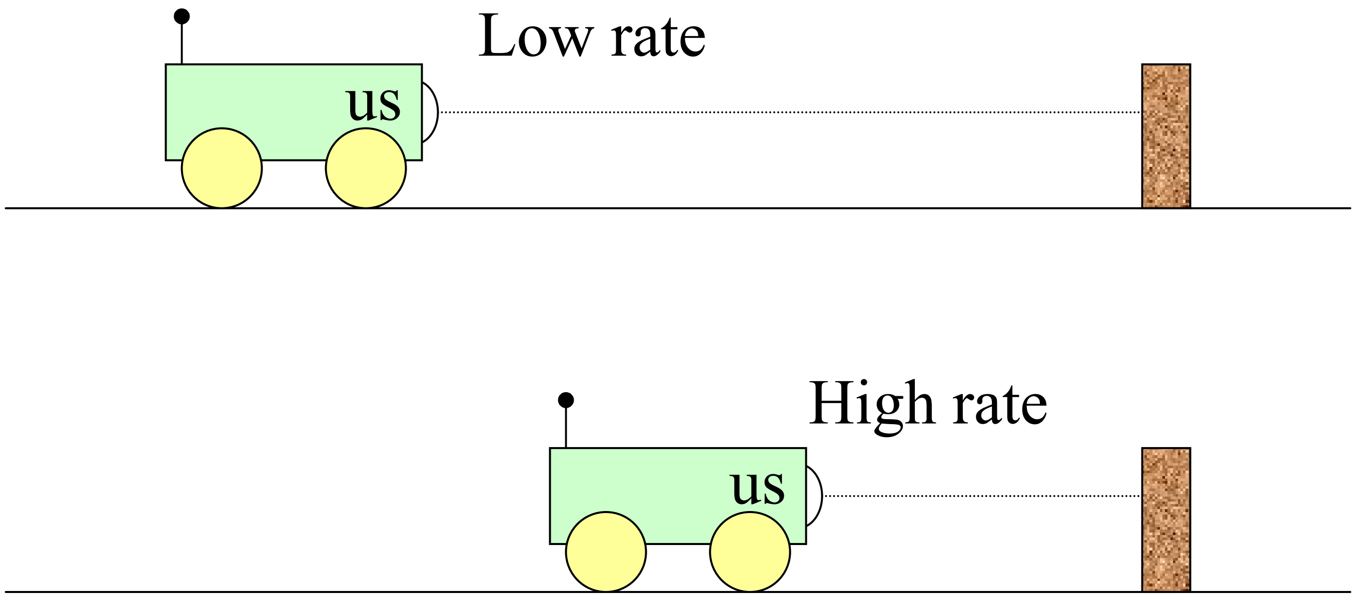
Examples: altimeter reading

- The smaller the altitude, the higher the acquisition rate:



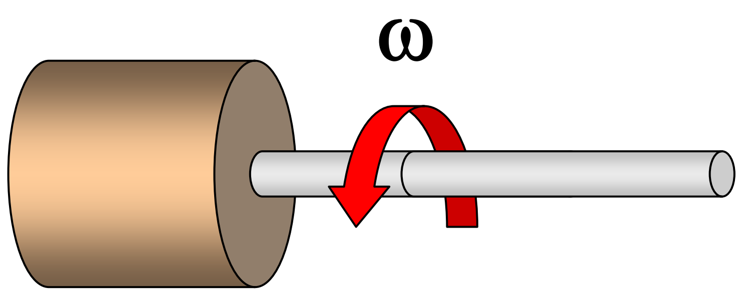
Obstacle avoidance

- 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.

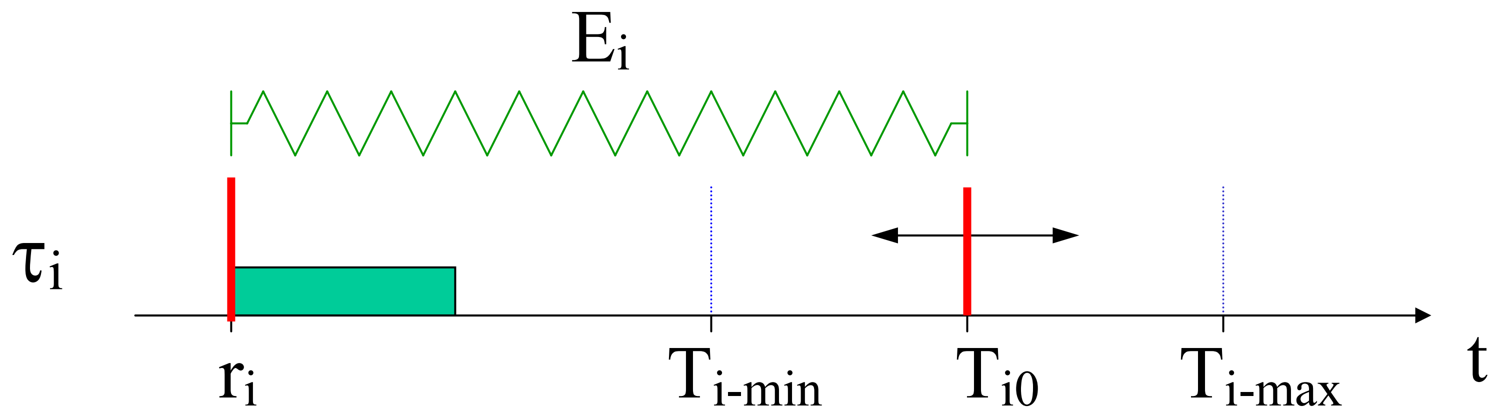


Elastic task model

- A periodic task τ_i is characterized by:

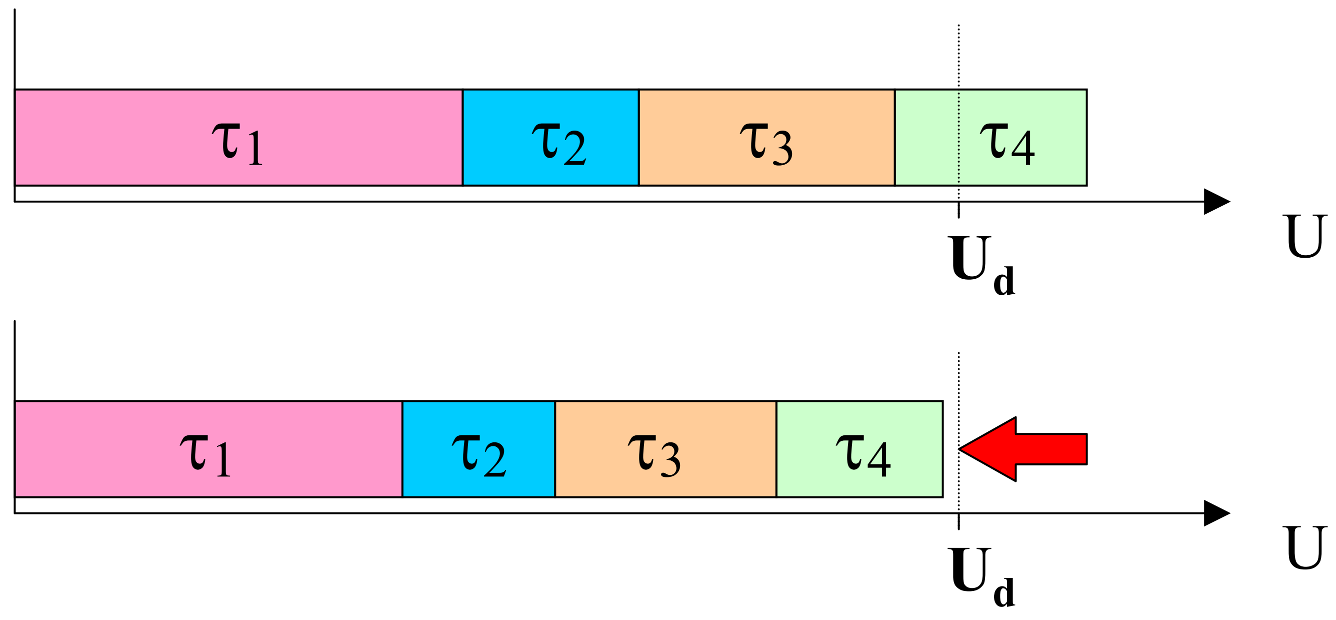
$$(C_i,\ T_{i\text{-}min},\ T_{i\text{-}max},\ E_i)$$

- 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

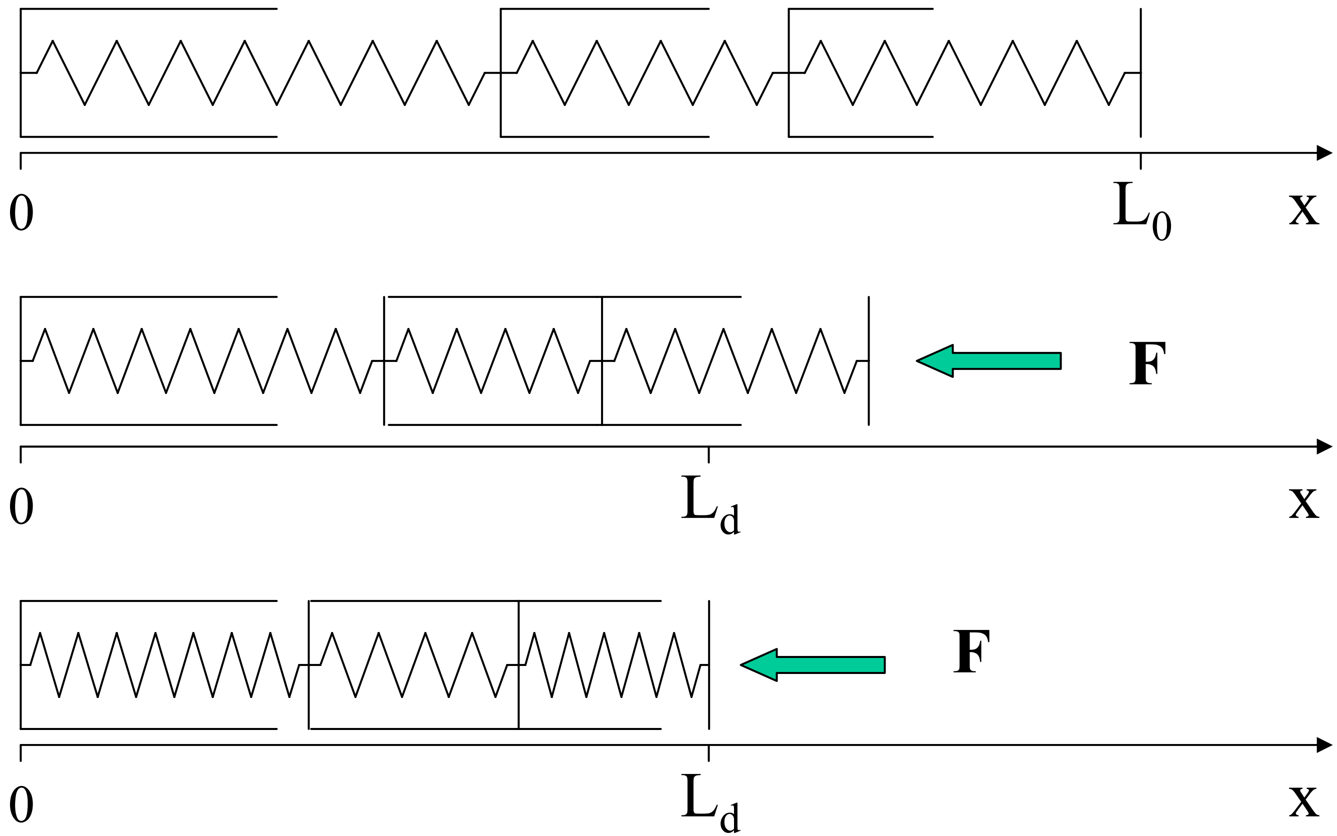
$$U_i = U_{io} - (U_0 - U_d)\frac{E_i}{E_s}$$

then:

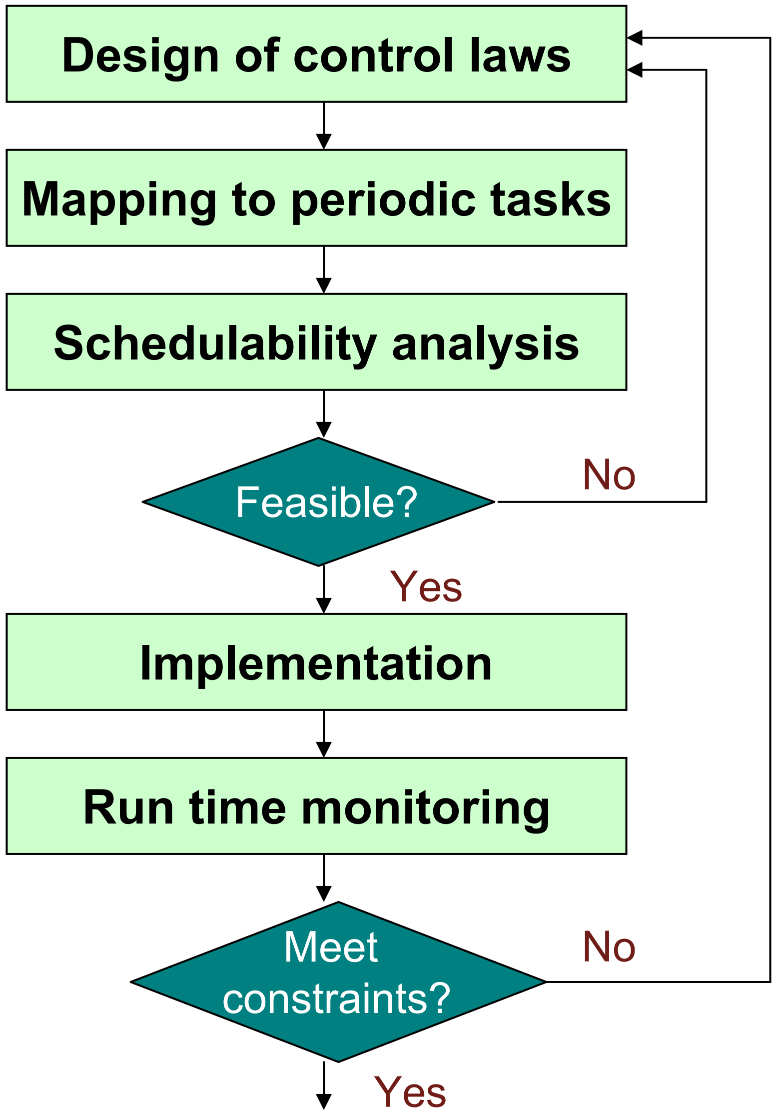
$$T_i = \frac{C_i}{U_i}$$

Solution with constraints

Iterative solution $O(n^2)$:



Control design issues

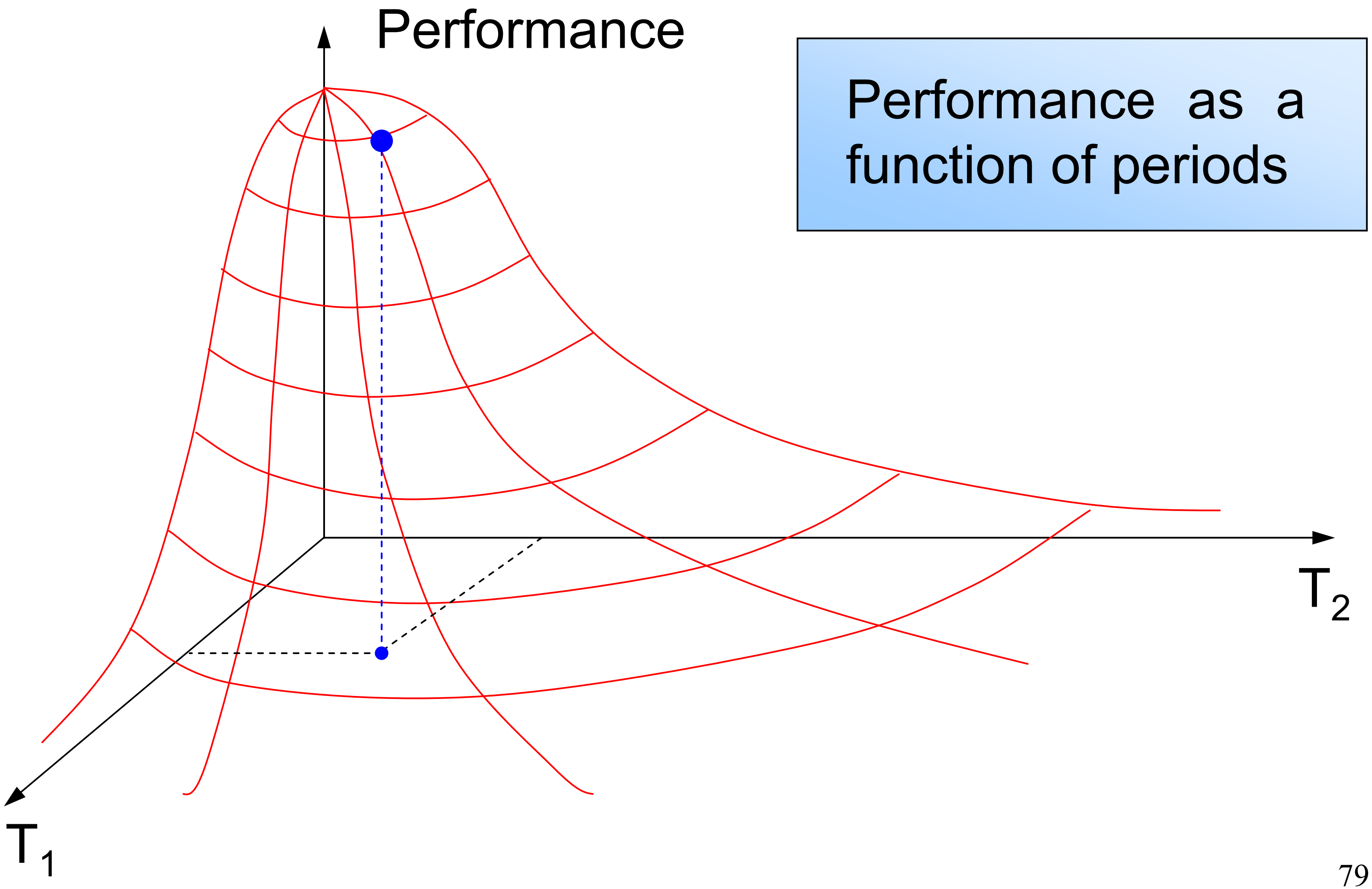


Traditional approach

Disadvantages

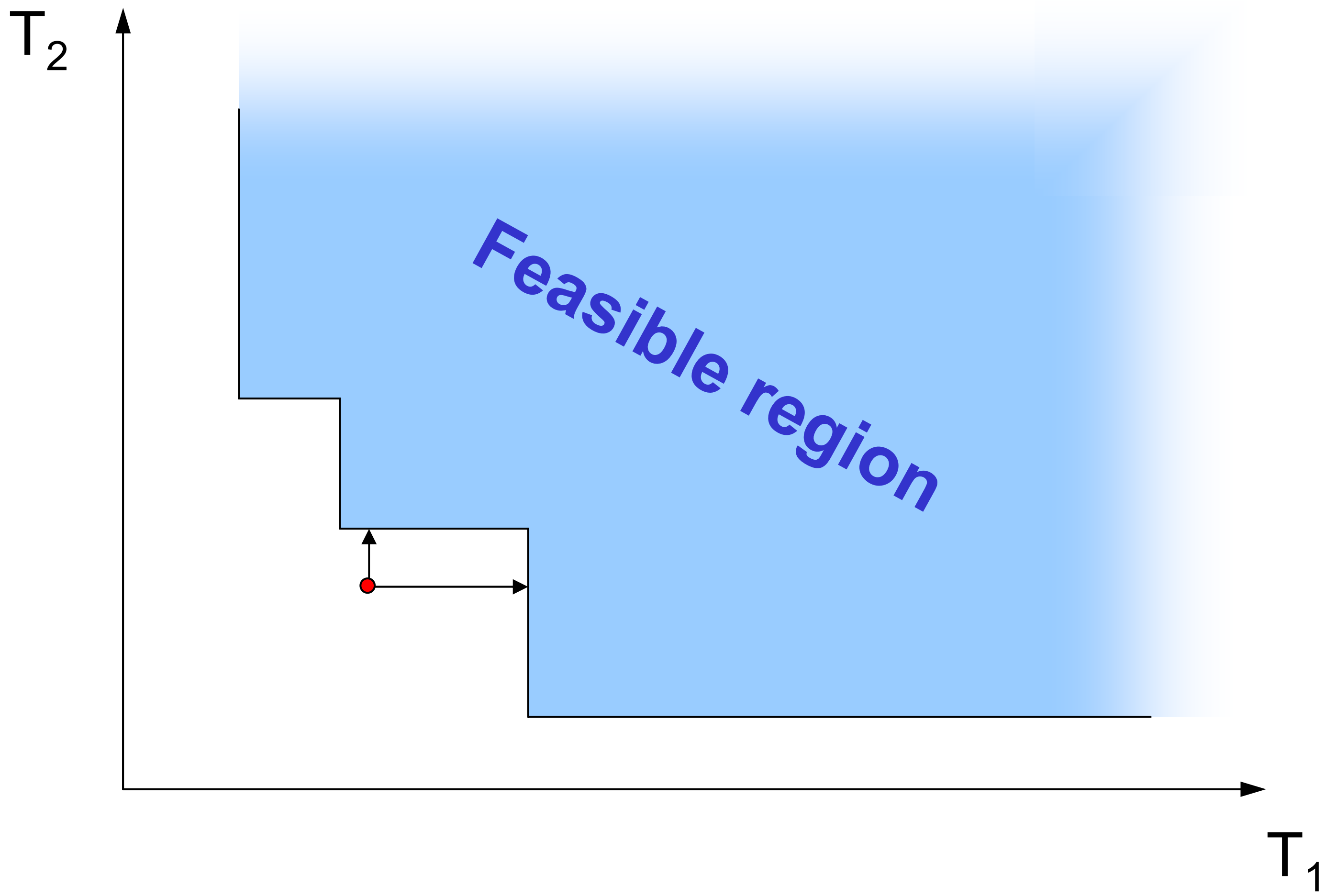
- Repetitive development process
- Suboptimal performance
- Suboptimal use of the resources

Control performance

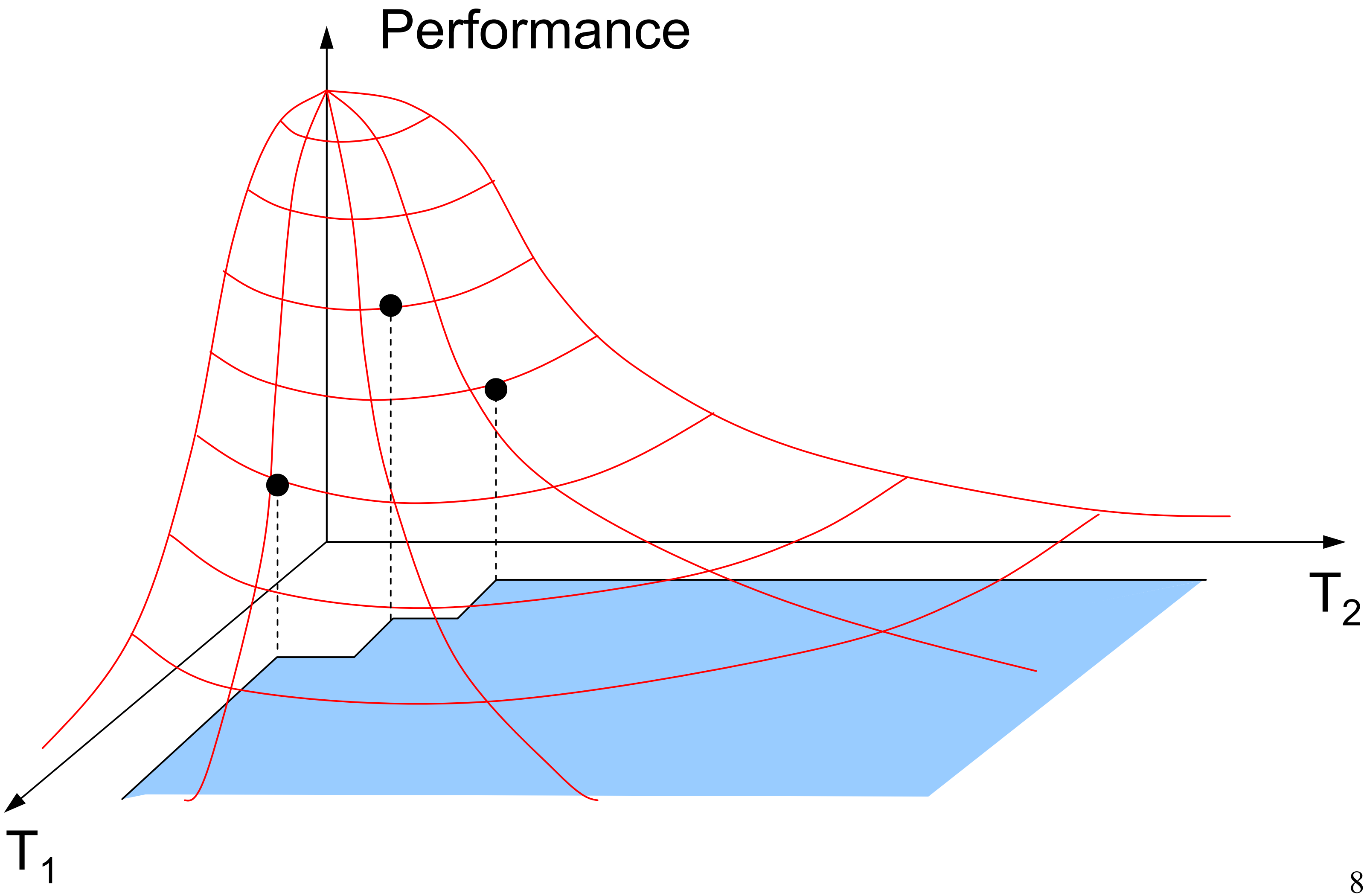


Performance as a function of periods

Sensitivity Analysis



RT & Control codesign



Codesign as optimization

Generic control law ← Sampling period, Delay, Jitter

↓

System performance characterization

↓

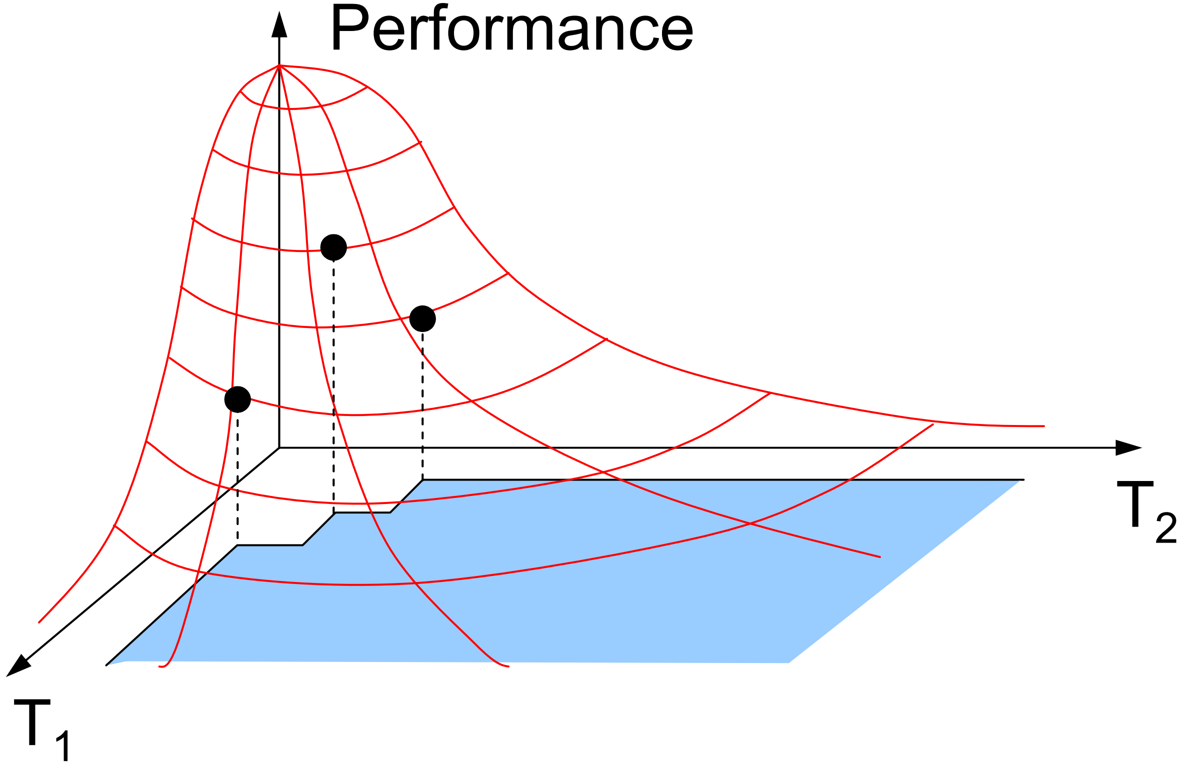
Resource constraints characterization

↓

Optimization process

↓

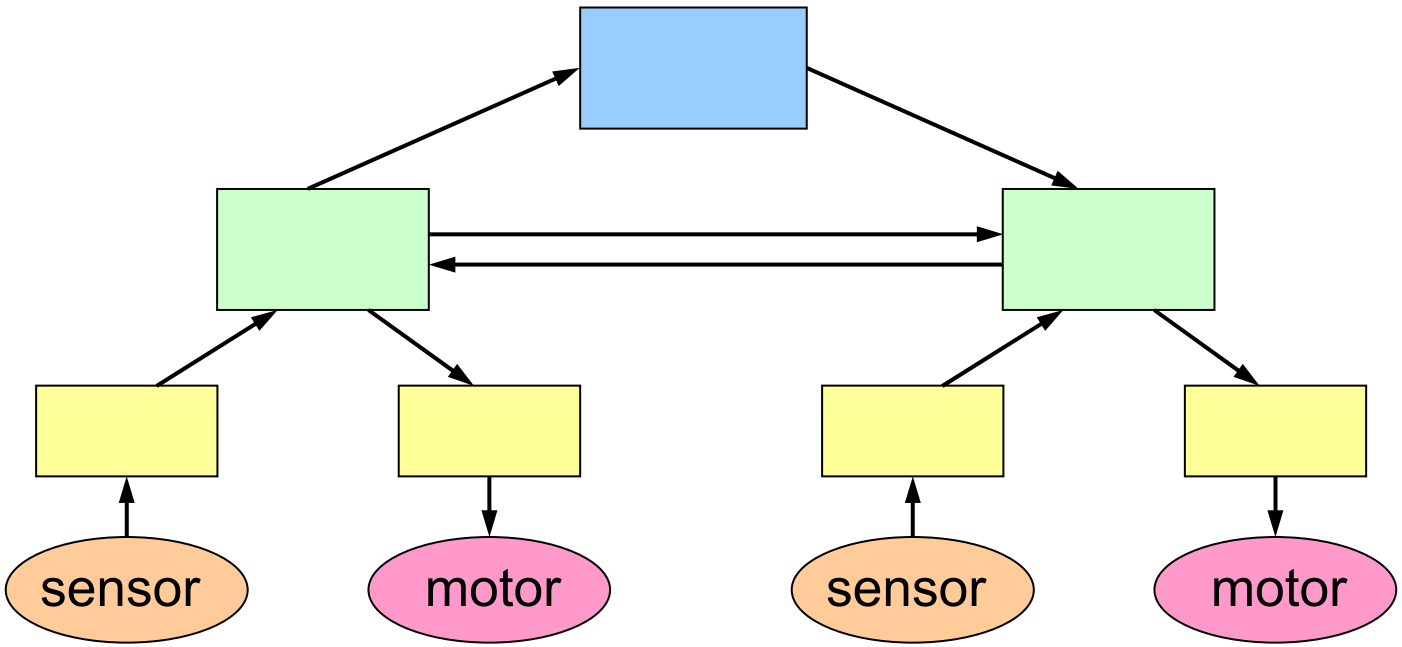
Task parameters



Conclusions

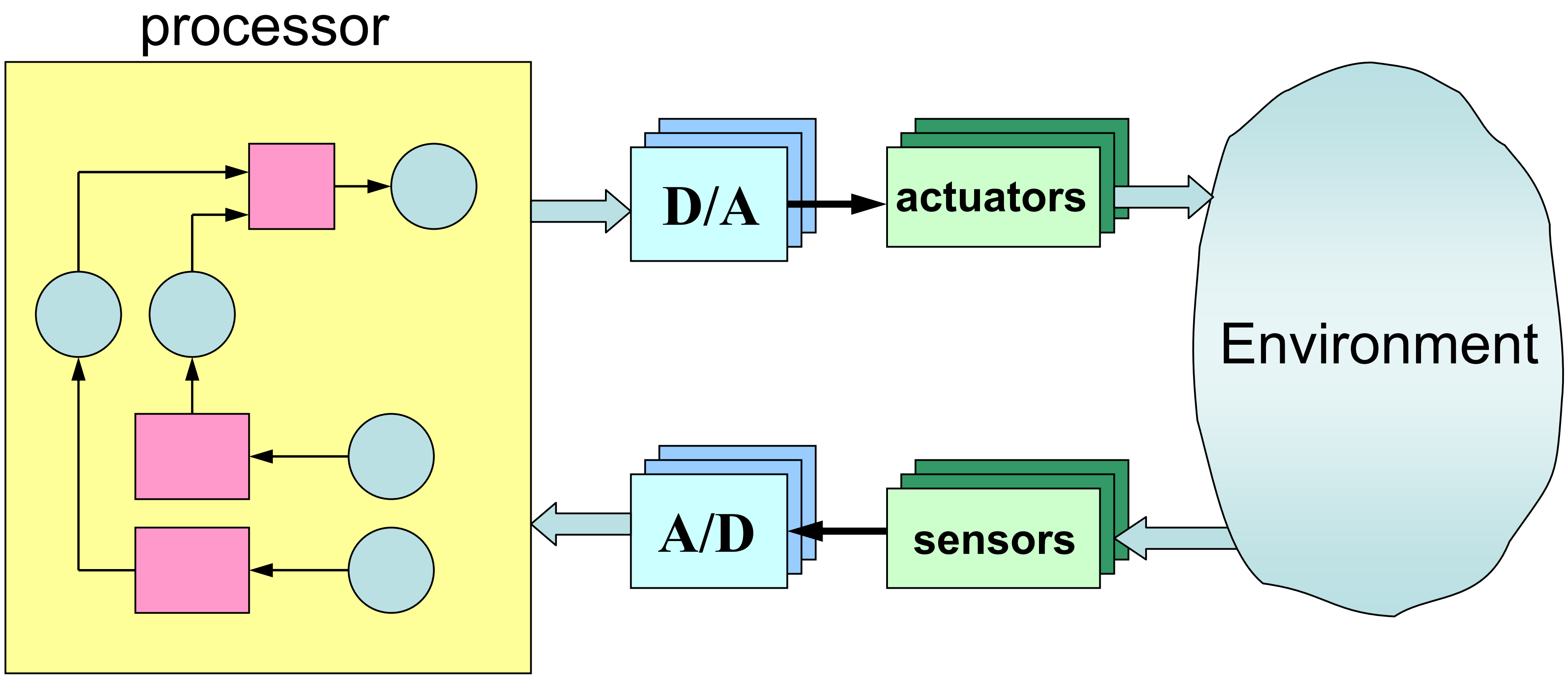
When designing complex embedded systems:

1. Split your system in components, follow a modular design, with hierarchical control levels with precise interface:



Conclusions

2. Organize software as a set of control tasks with precise timing and resource constraints:



Conclusions

3. Estimate worst-case computation times of tasks, using specific tools and testing.
4. Select an appropriate scheduling algorithm and a suitable resource access protocol.
5. Estimate maximum blocking times due non preemptive sections or mutually exclusive resources.
6. Apply schedulability analysis to verify feasibility.

Conclusions

7. If possible, use sensitivity analysis and integrate real-time with control issues at early design stages.

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

9. Perform extensive testing and simulation at each implementation step under worst-case scenarios.