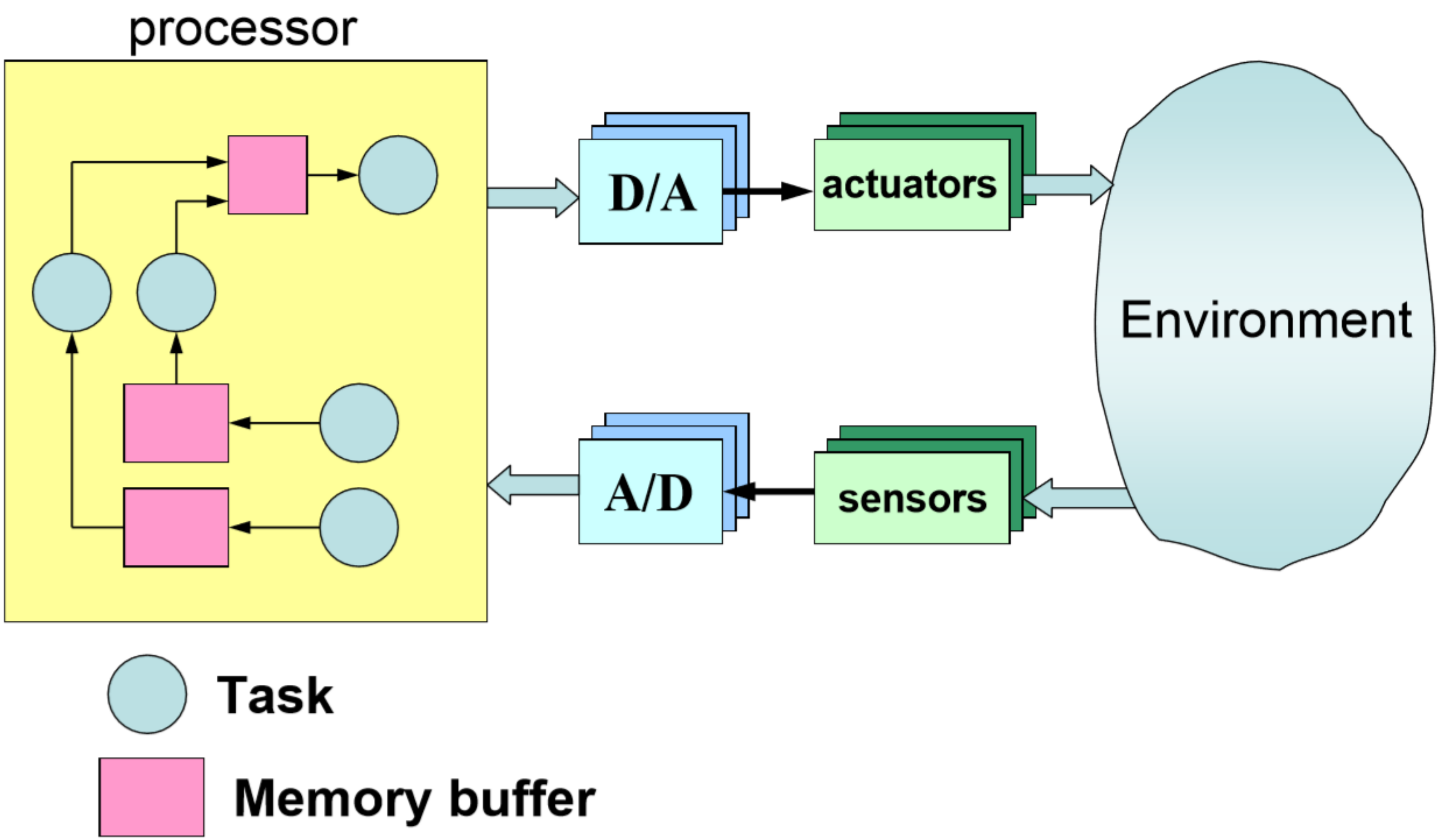
ИНФОРМАЦИОННО-УПРАВЛЯЮЩИЕ СИСТЕМЫ РЕАЛЬНОГО ВРЕМЕНИ

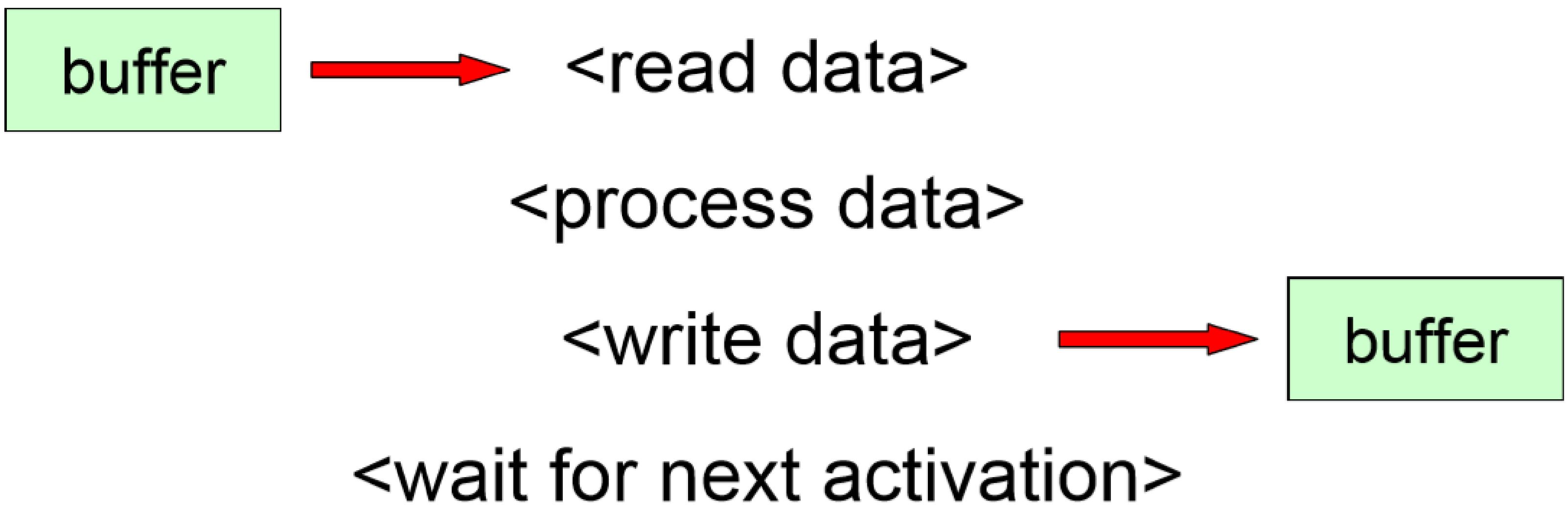
Лекция 2:
Динамическое планирование вычислений и оценка планируемости - 1

Кафедра АСВК,
Лаборатория Вычислительных Комплексов
Балашов В.В.

Software Control Systems



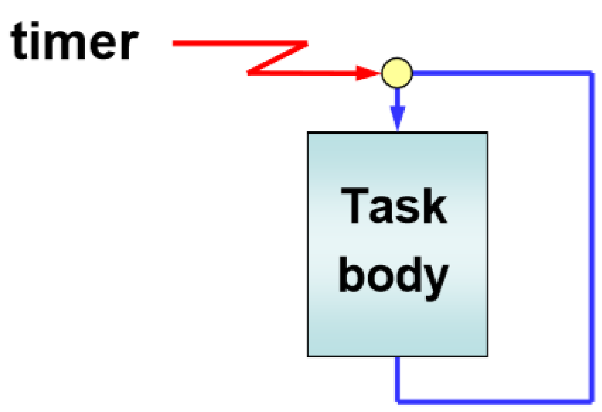
Typical task structure



Activation modes

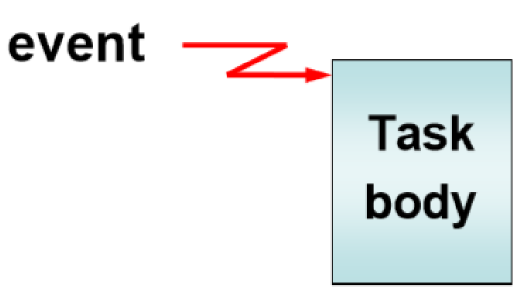
Periodic task (time driven)

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



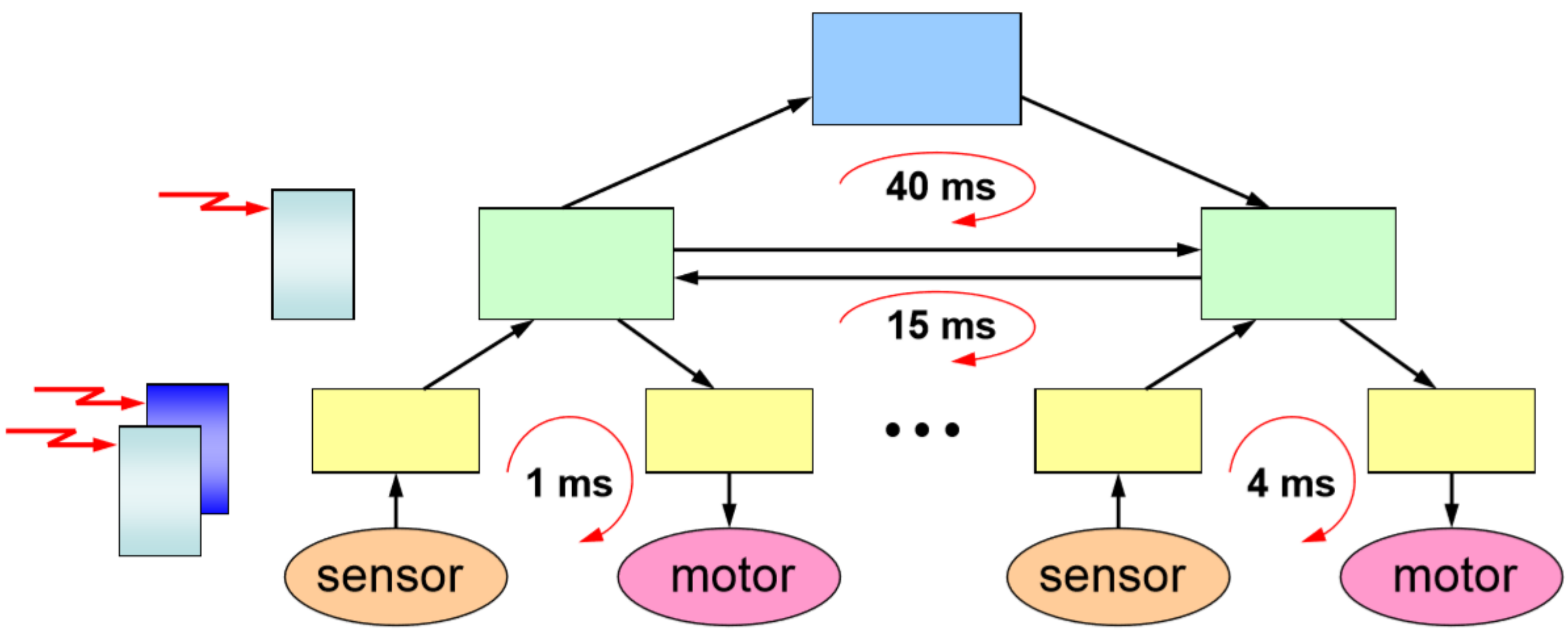
Aperiodic task (event driven)

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



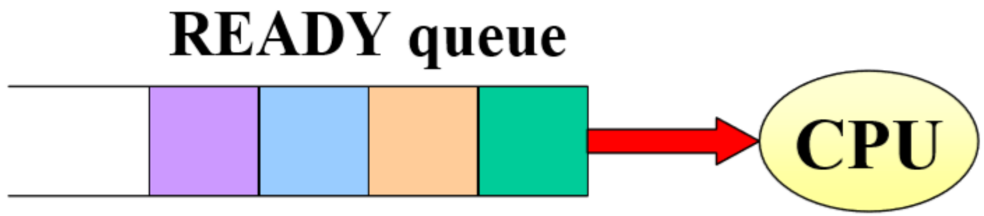
Complex control applications

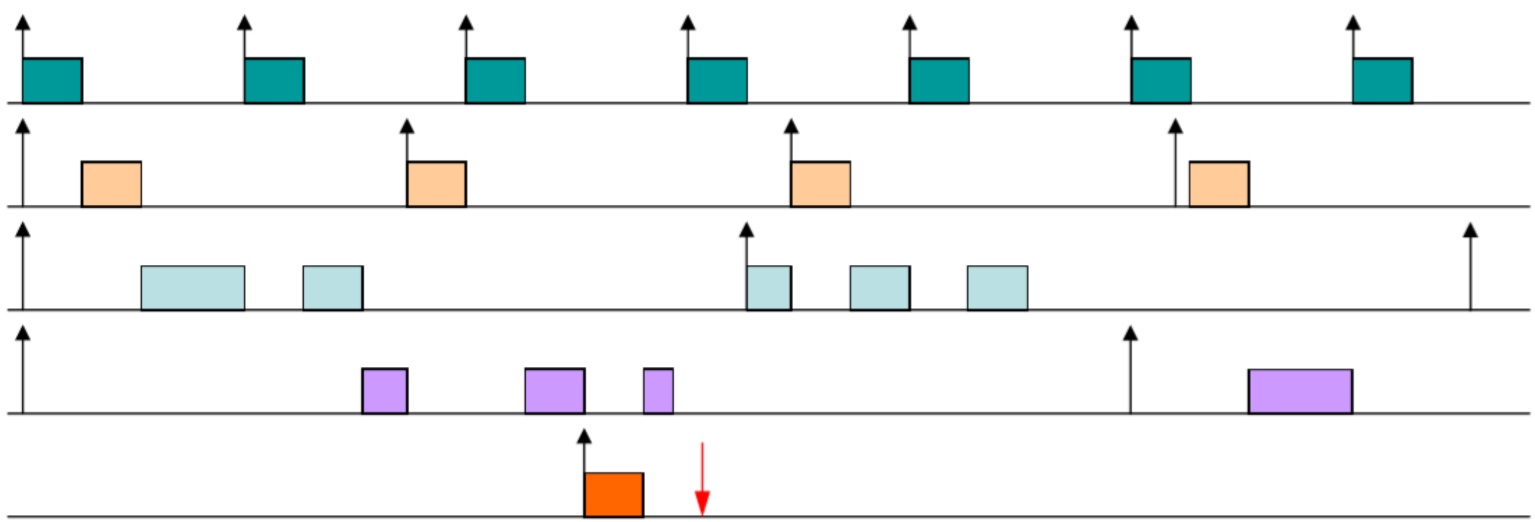
- Hierarchical design
- Many periodic activities running a different rates
- Many event-driven routines



Task scheduling

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

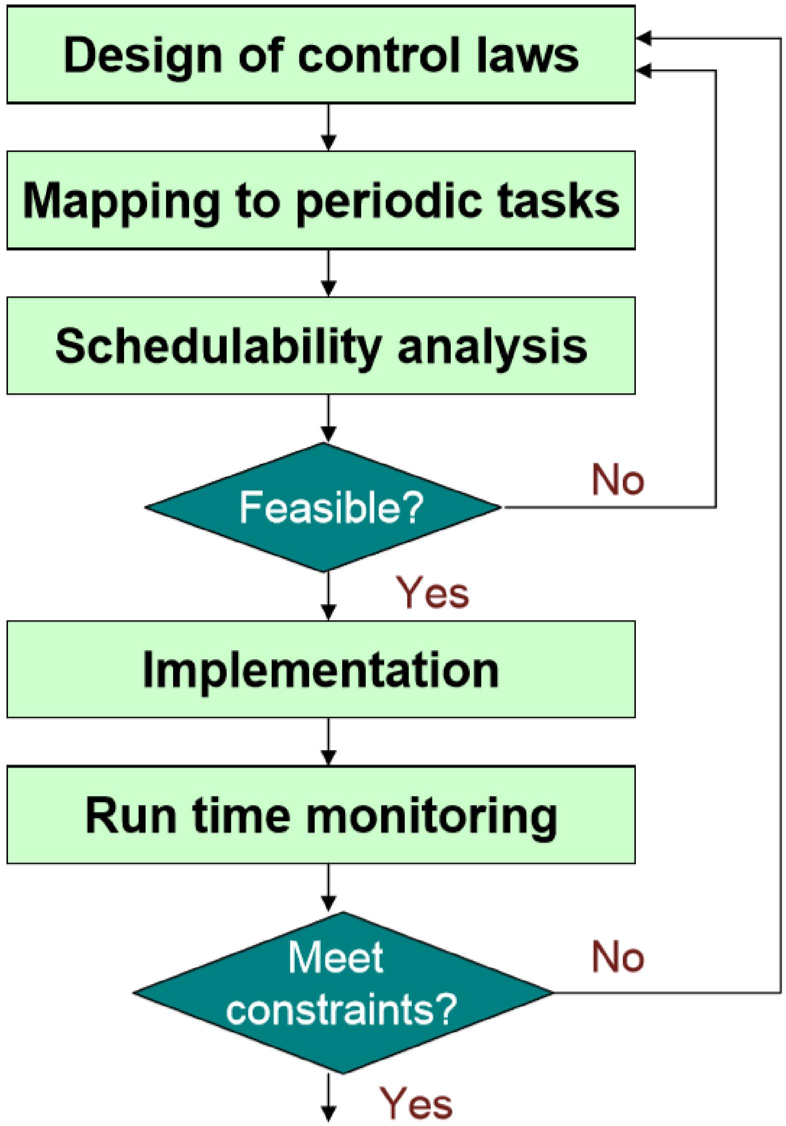




Importance of scheduling

- 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)

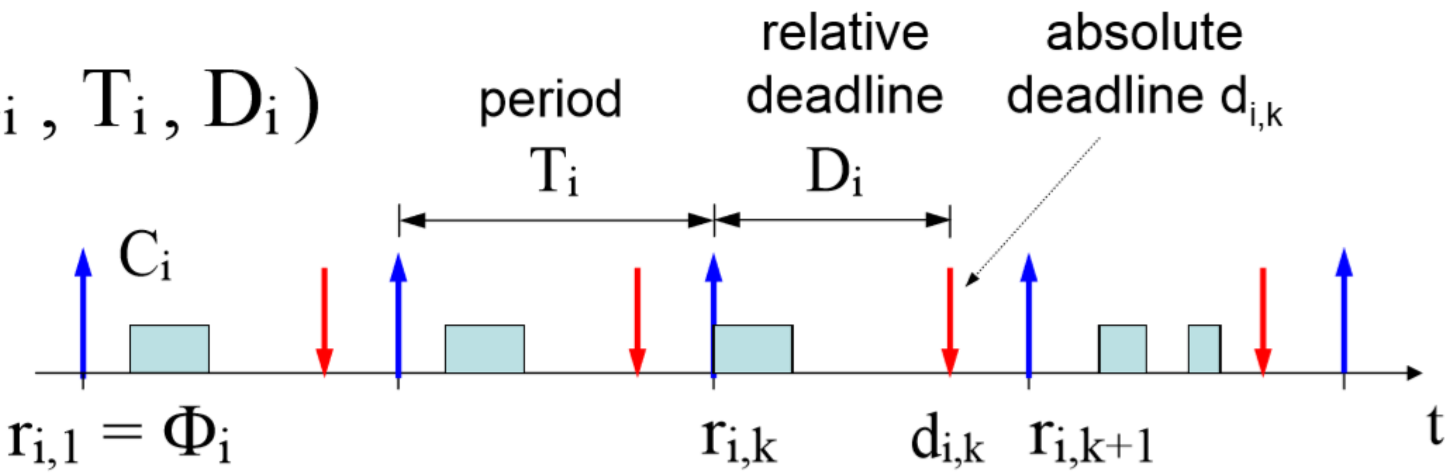
Control design



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

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

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

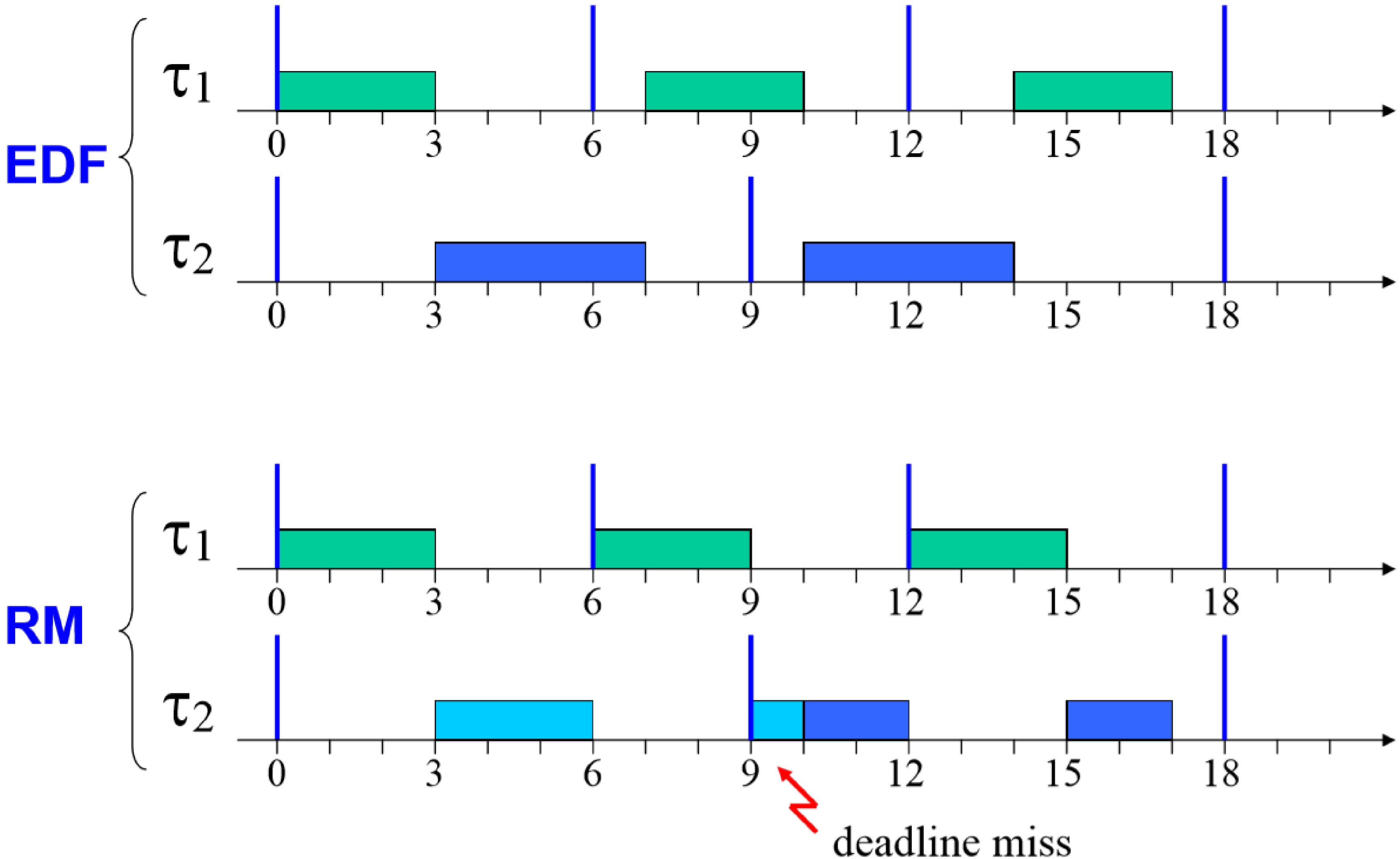
Fixed-Priority Scheduling (FPS)

- This is the most widely used approach
- Each task has a fixed, static, priority which is computed 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



FPS v EDF

- FPS is easier to implement as priorities are static
- EDF is dynamic and requires a more complex run-time system which will have higher overhead
- It is easier to incorporate tasks without deadlines into FPS; giving a task an arbitrary deadline is more artificial
- It is easier to incorporate other factors into the notion of priority than it is into the notion of deadline

FPS v EDF

- During overload situations
  - FPS is more predictable; Low priority process miss their deadlines first
  - EDF is unpredictable; a domino effect can occur in which a large number of processes miss deadlines
- But EDF gets more out of the processor!

Preemption

- With priority-based scheduling, a high-priority task may be released during the execution of a lower priority one
- In a preemptive scheme, there will be an immediate switch to the higher-priority task
- With non-preemption, the lower-priority task will be allowed to complete before the other executes
- Preemptive schemes enable higher-priority tasks to be more reactive, and hence they are preferred

Scheduling Characteristics

- Sufficient – pass the test will meet deadlines
- Necessary – fail the test will miss deadlines

- Exact – necessary and sufficient

- Sustainable – system stays schedulable if conditions 'improve'

Simple Task Model

- The application is assumed to consist of a fixed set of tasks
- All tasks are periodic, with known periods
- The tasks are completely independent of each other
- All system's overheads, context-switching times and so on are ignored (i.e, assumed to have zero cost)
- All tasks have a deadline equal to their period (that is, each task must complete before it is next released)
- All tasks have a fixed worst-case execution time

Standard Notation

B	Worst-case blocking time for the task (if applicable)
C	Worst-case computation time (WCET) of the task
D	Deadline of the task
I	The interference time of the task
N	Number of tasks in the system
P	Priority assigned to the task (if applicable)
R	Worst-case response time of the task
T	Minimum time between task releases, jobs, (task period)
U	The utilization of each task (equal to C/T)

Rate Monotonic Priority Assignment

- Each task is assigned a (unique) priority based on its period; the shorter the period, the higher the priority
- i.e, for two tasks `i` and `j`,

$$T_i < T_j \Rightarrow P_i > P_j$$

- This assignment is optimal in the sense that if any task set can be scheduled (using pre-emptive priority-based scheduling) with a fixed-priority assignment scheme, then the given task set can also be scheduled with a rate monotonic assignment scheme
- Note, priority 1 is the lowest (least) priority

Example Priority Assignment

Process	Period, T	Priority, P
a	25	5
b	60	3
c	42	4
d	105	1
e	75	2

Basic results

Assumptions:

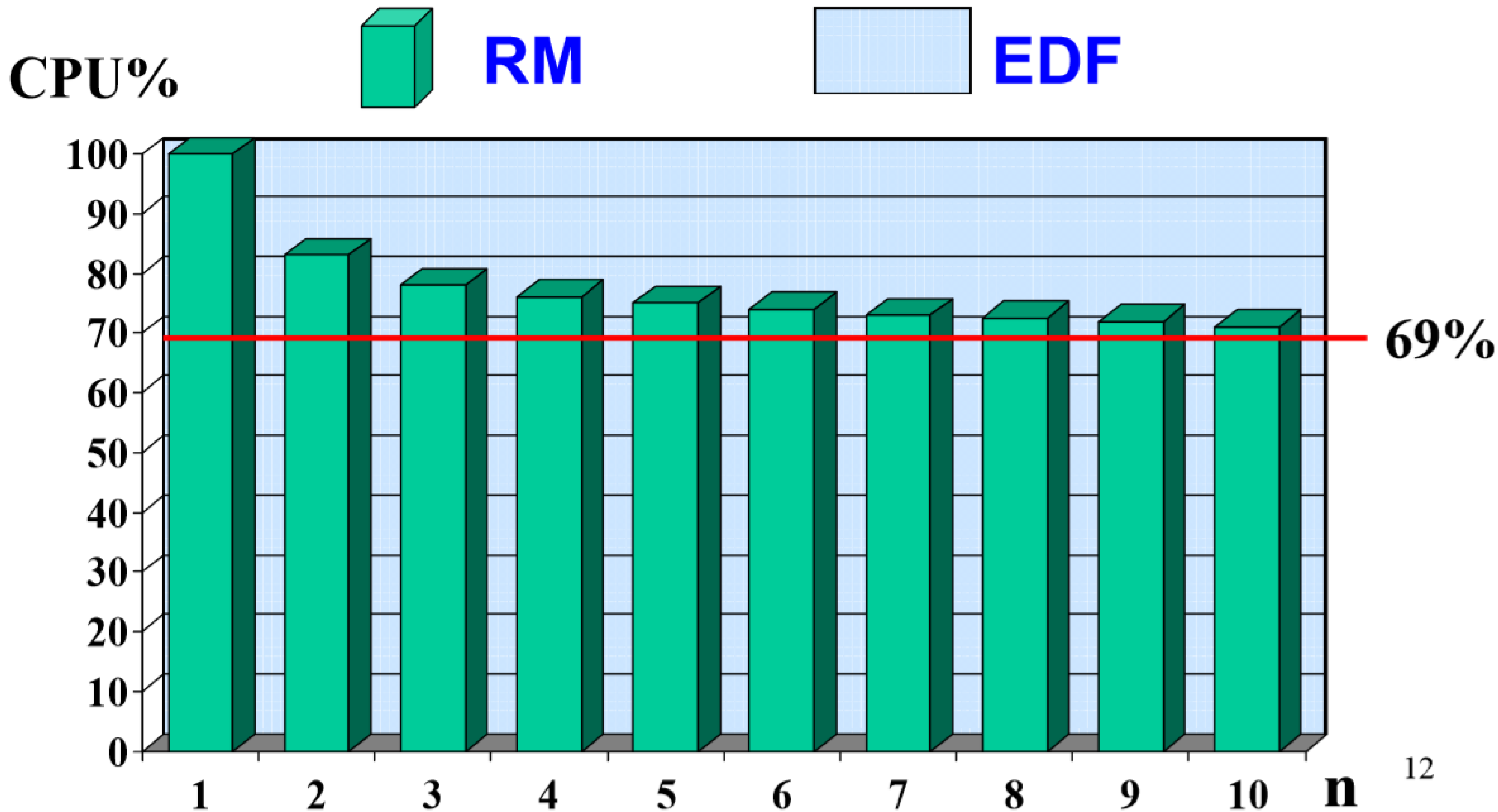
- Independent tasks
- $\Phi_i = 0$
- $D_i = T_i$

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

- **under RM** **if** $$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq n\left(2^{1/n} - 1\right)$$
- **under EDF** **if and only if** $$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1$$

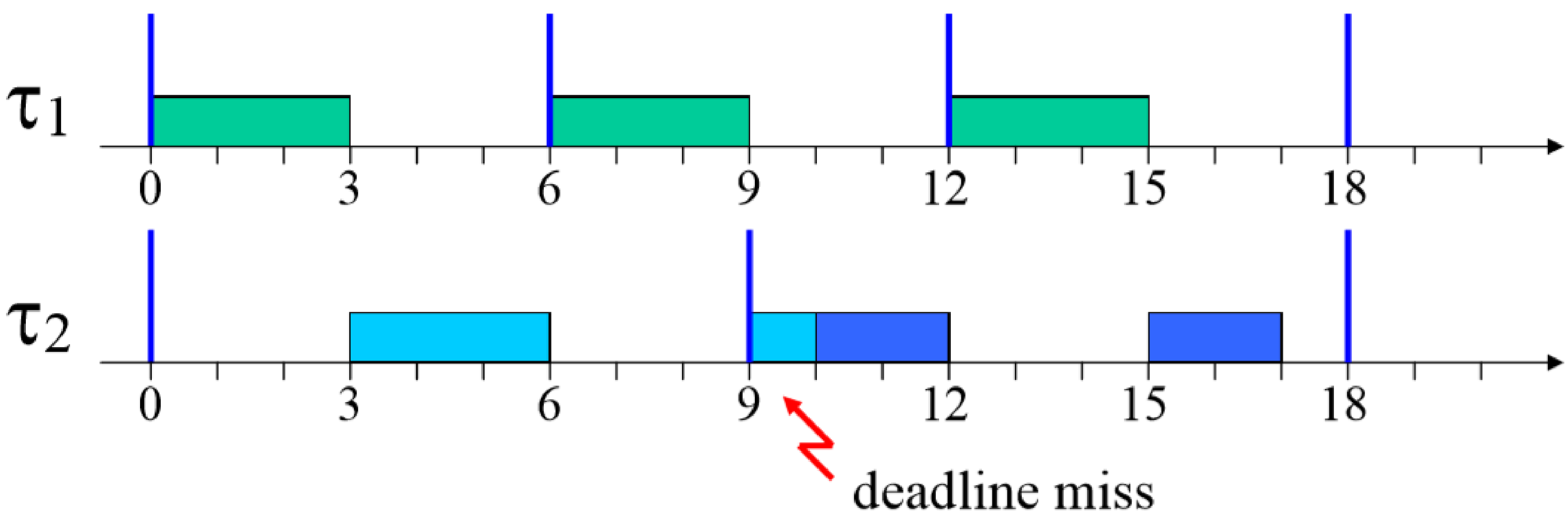
Schedulability bound

for n → ∞ $U_{lub} \to \ln 2 \cong 0.69$

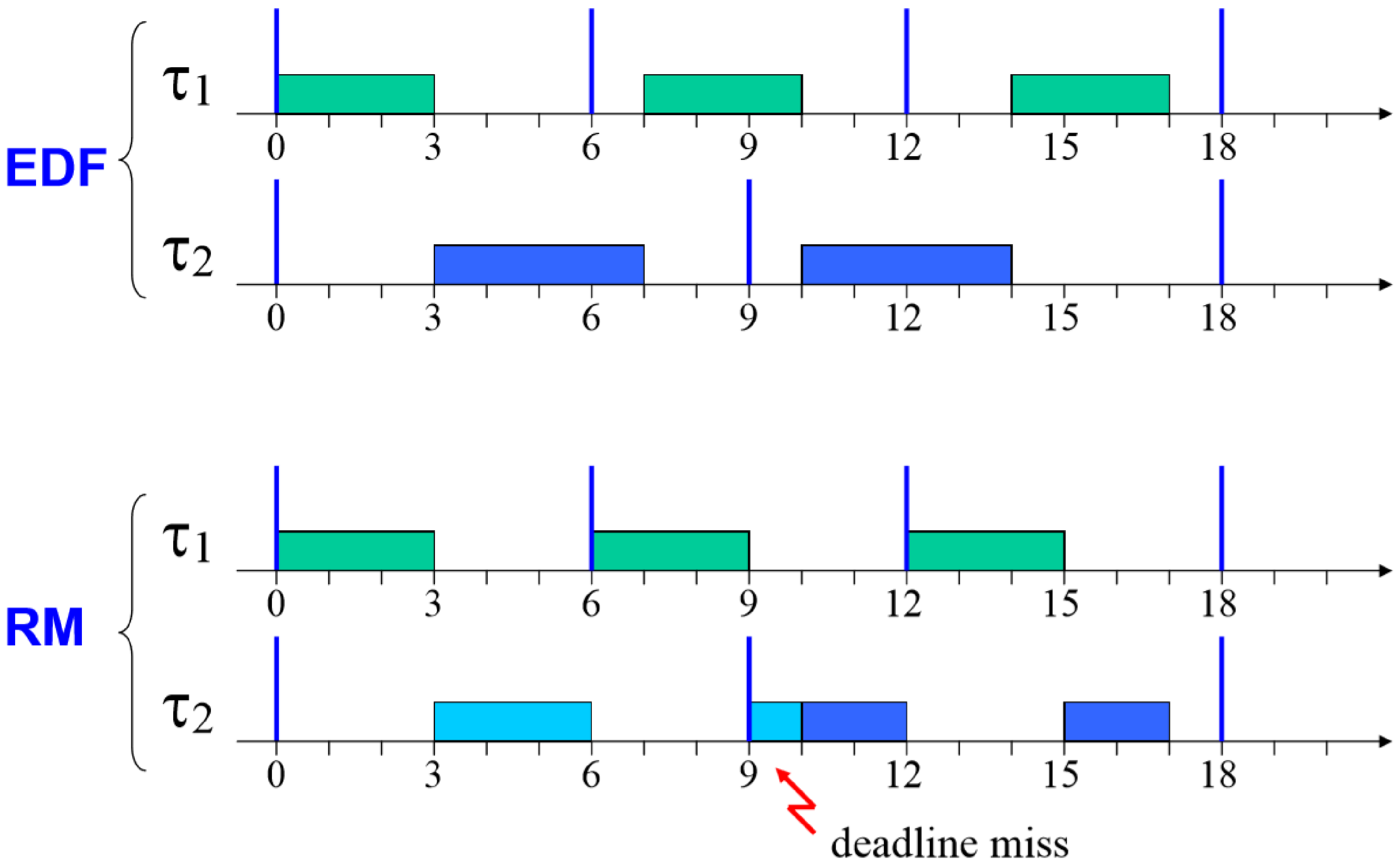


An unfeasible RM schedule

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

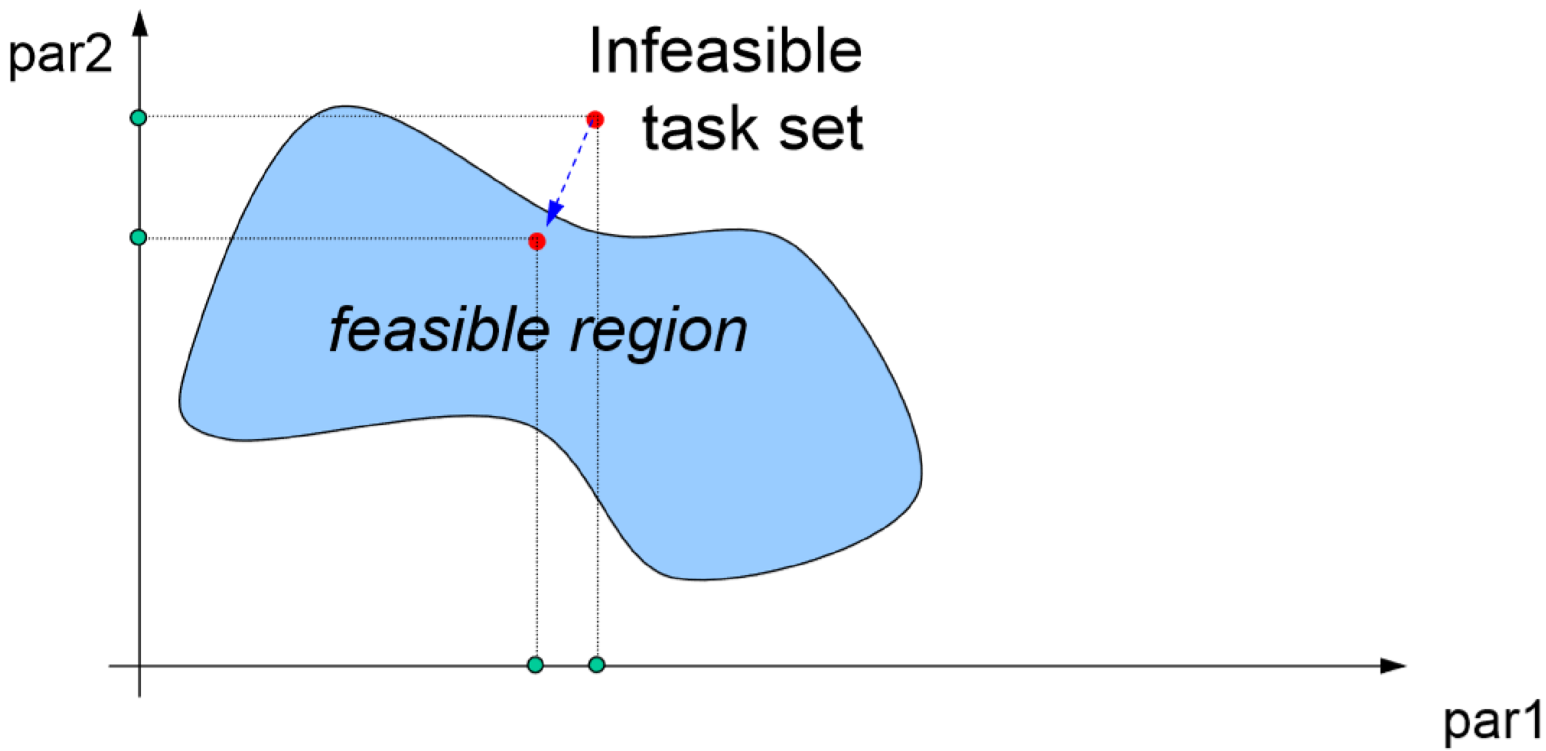


EDF vs. RM Schedule



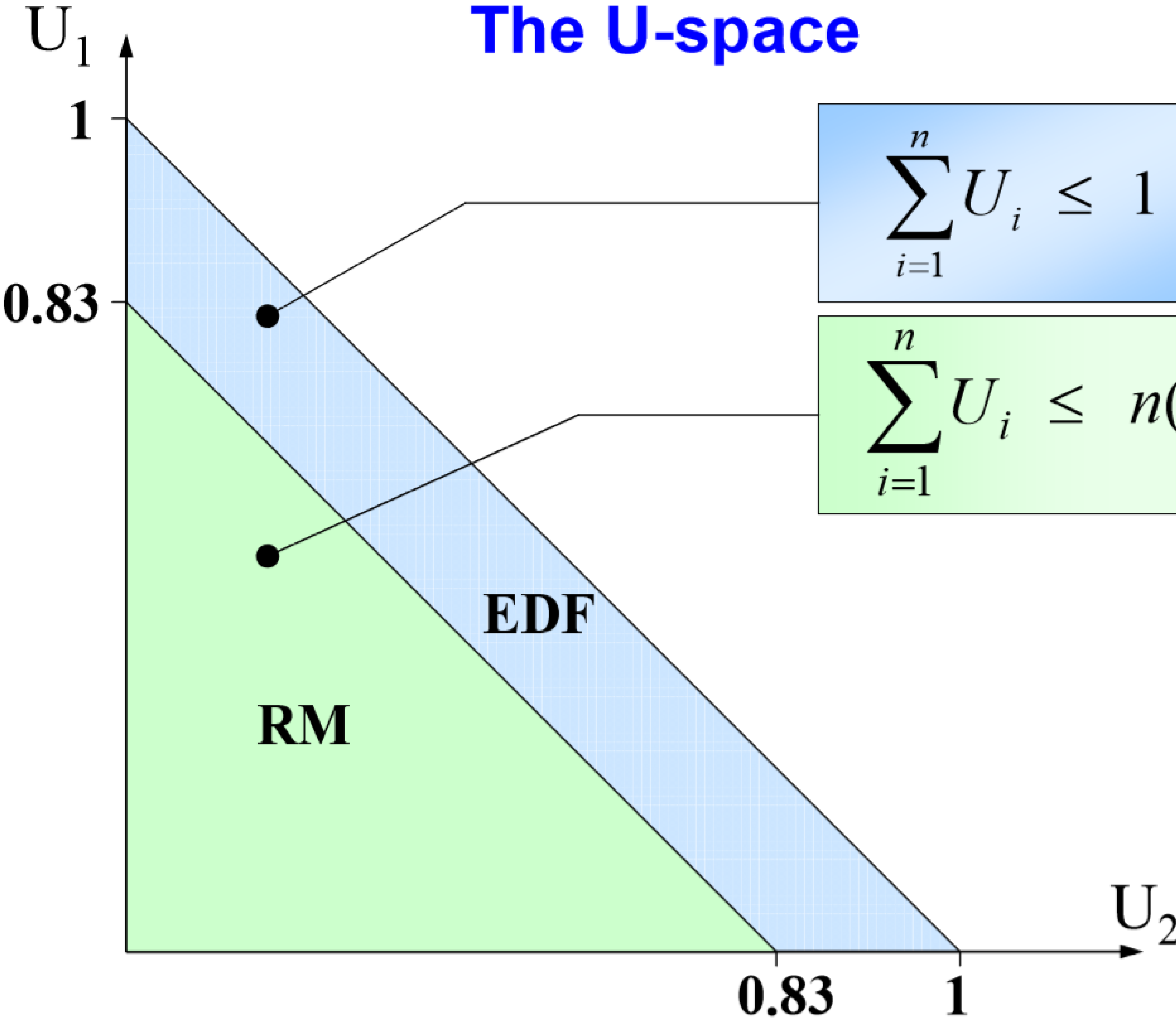
Schedulability region

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



Schedulability region

The U-space

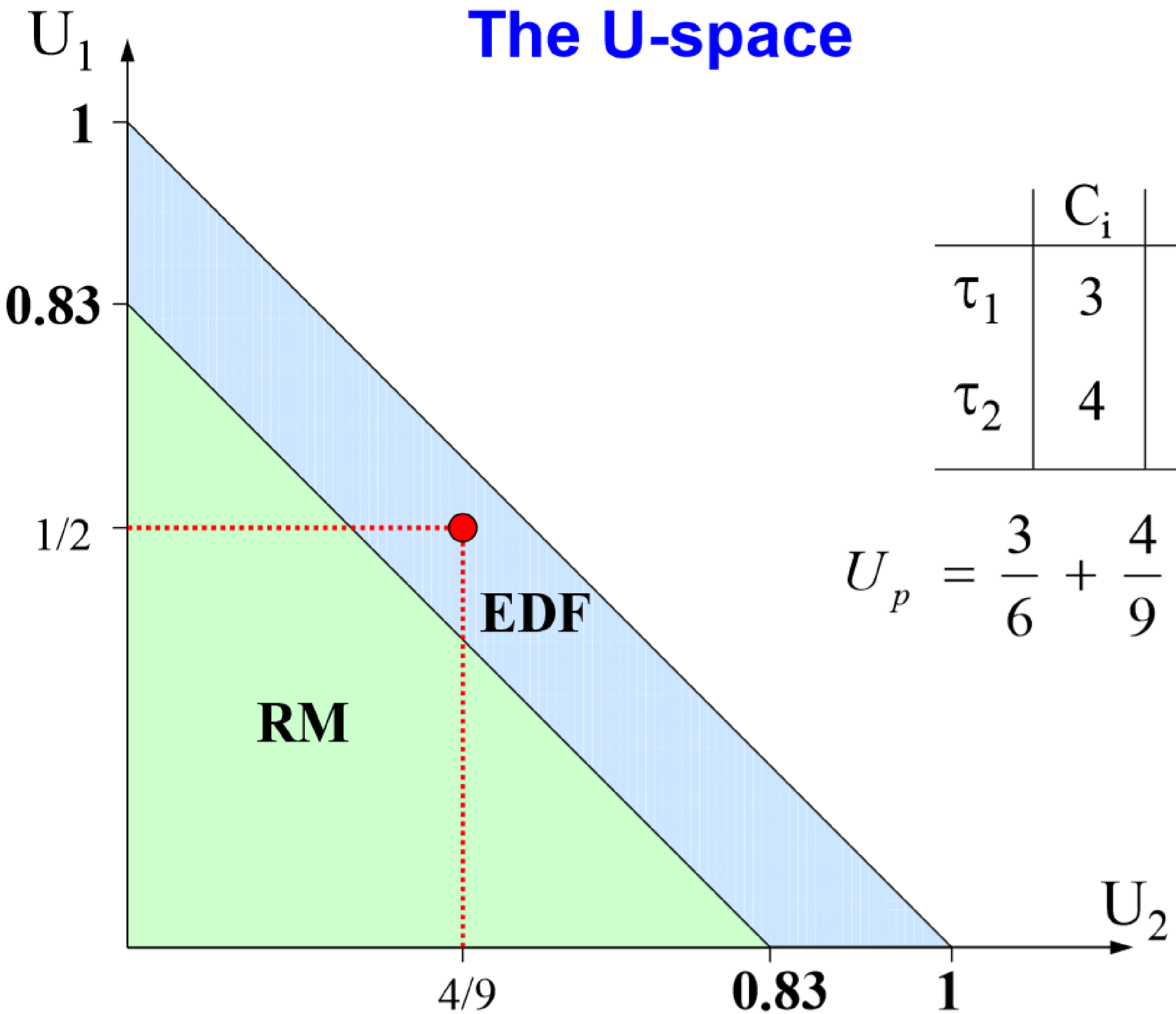


$$\sum_{i=1}^{n} U_i \le 1$$

$$\sum_{i=1}^{n} U_i \le n(2^{1/n} - 1)$$

Schedulability region

The U-space



	C_i	T_i
τ_1	3	6
τ_2	4	9

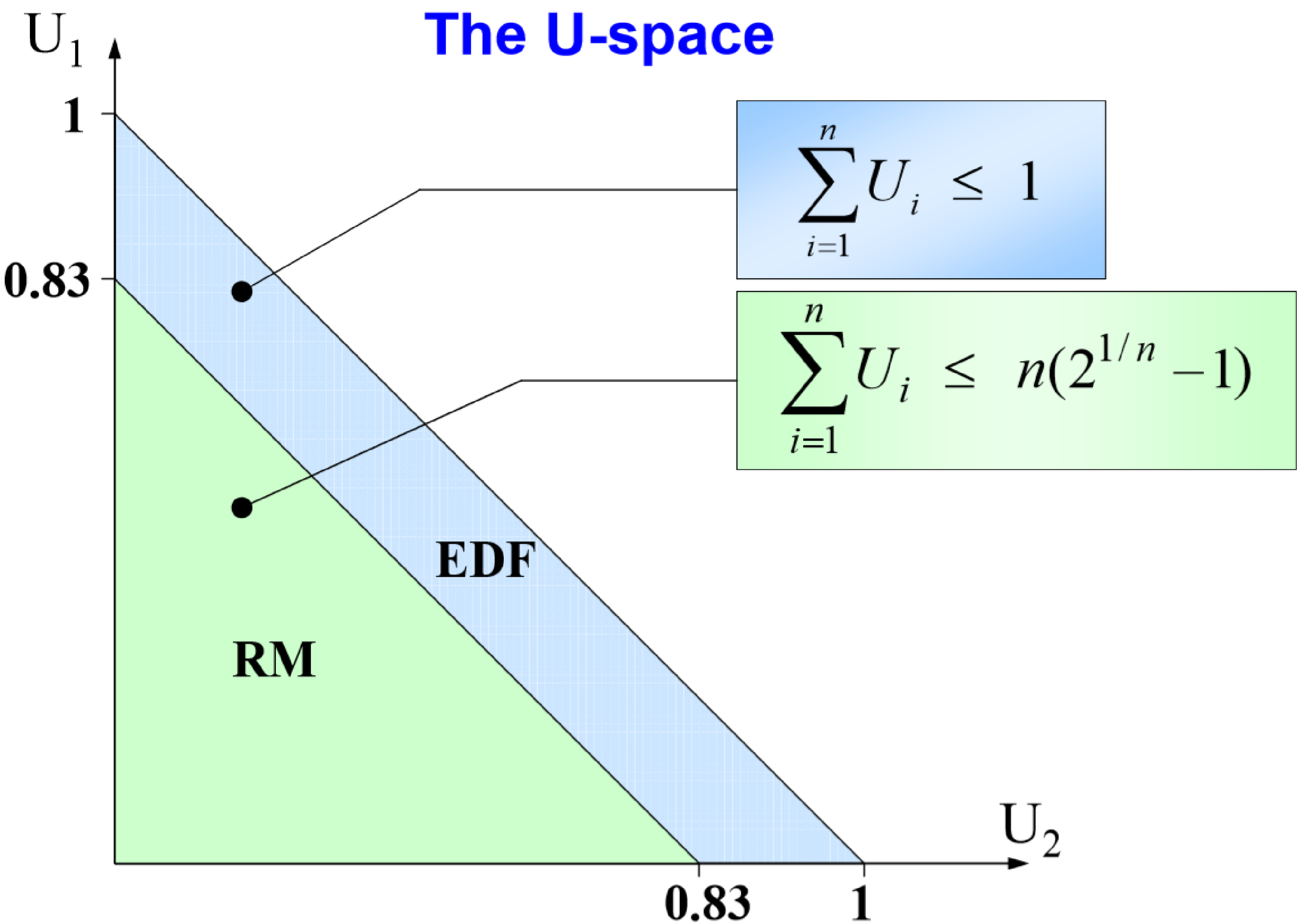
$$U_p = \frac{3}{6} + \frac{4}{9} = 0.94$$

The Hyperbolic Bound

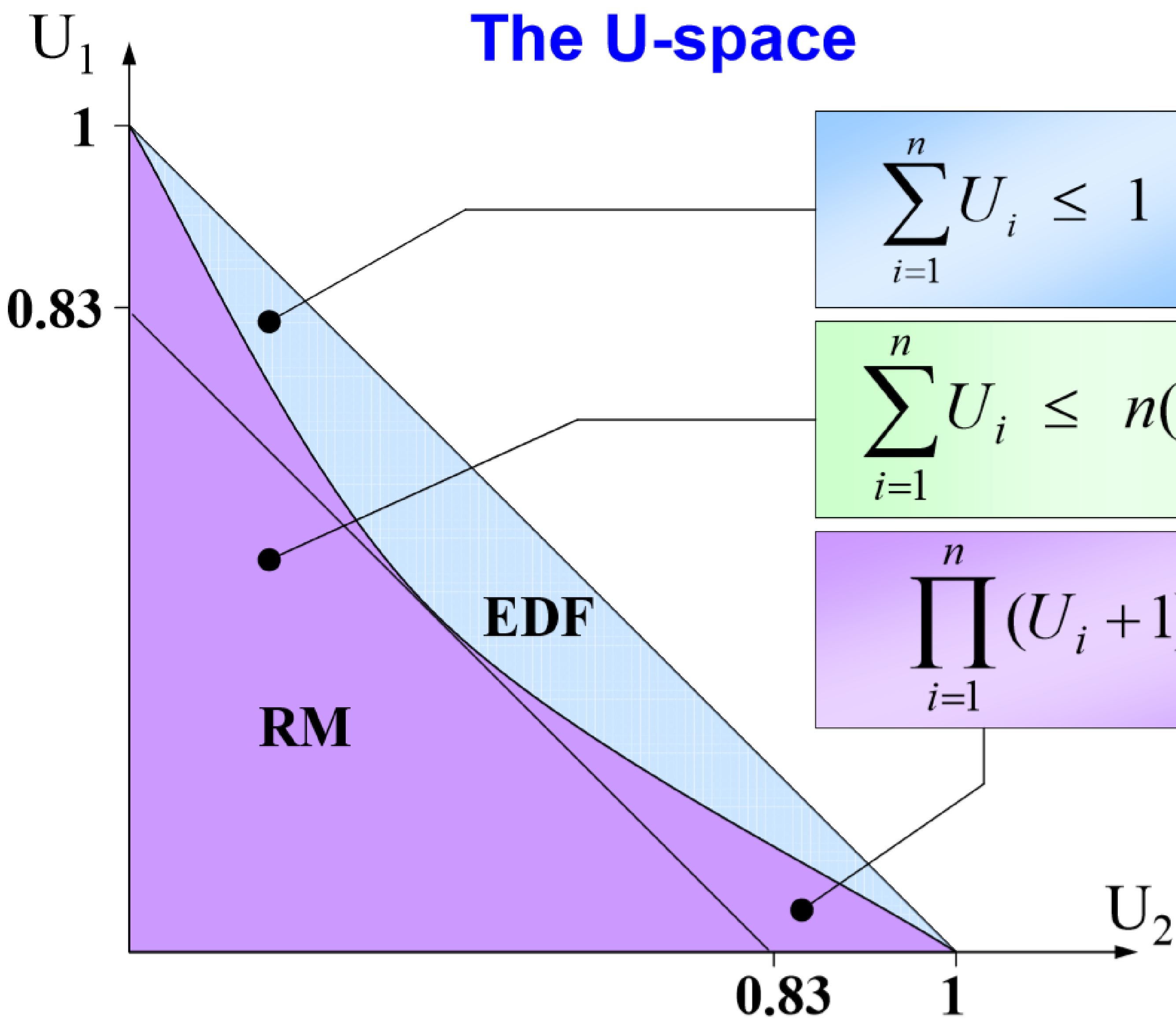
- In 2000, **Bini et al.** proved that a set of *n* periodic tasks is schedulable with RM if:

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

Schedulability region



Schedulability region

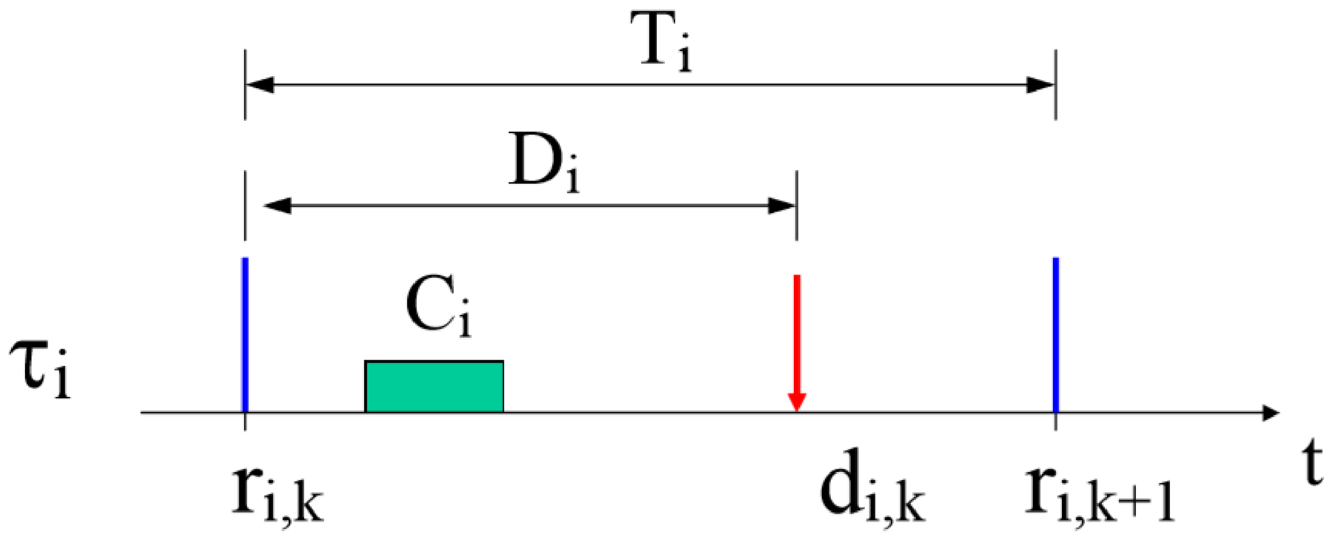


$$\sum_{i=1}^{n} U_i \leq 1$$

$$\sum_{i=1}^{n} U_i \leq n(2^{1/n} - 1)$$

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

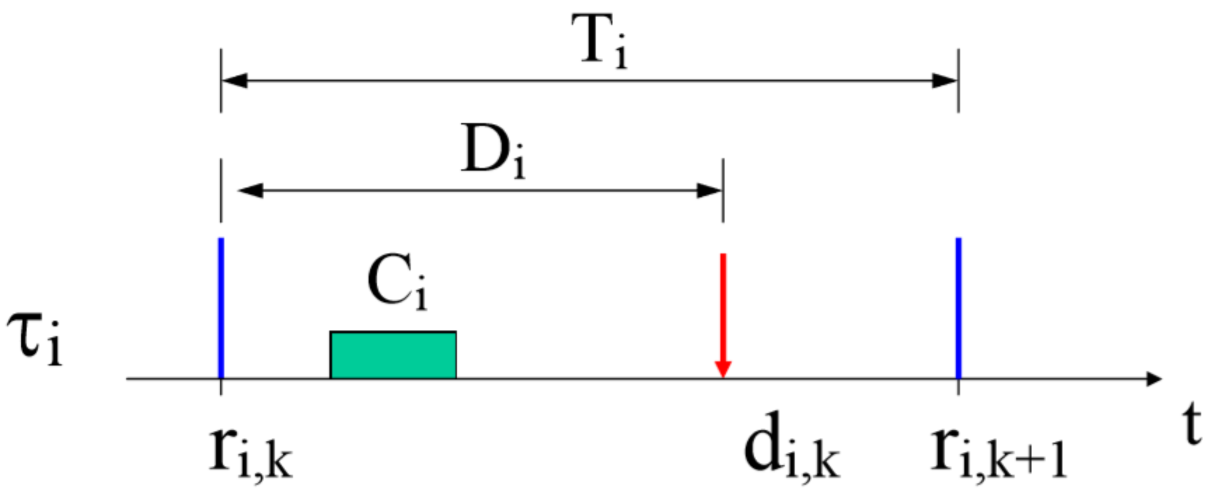
Handling tasks with D < T



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:

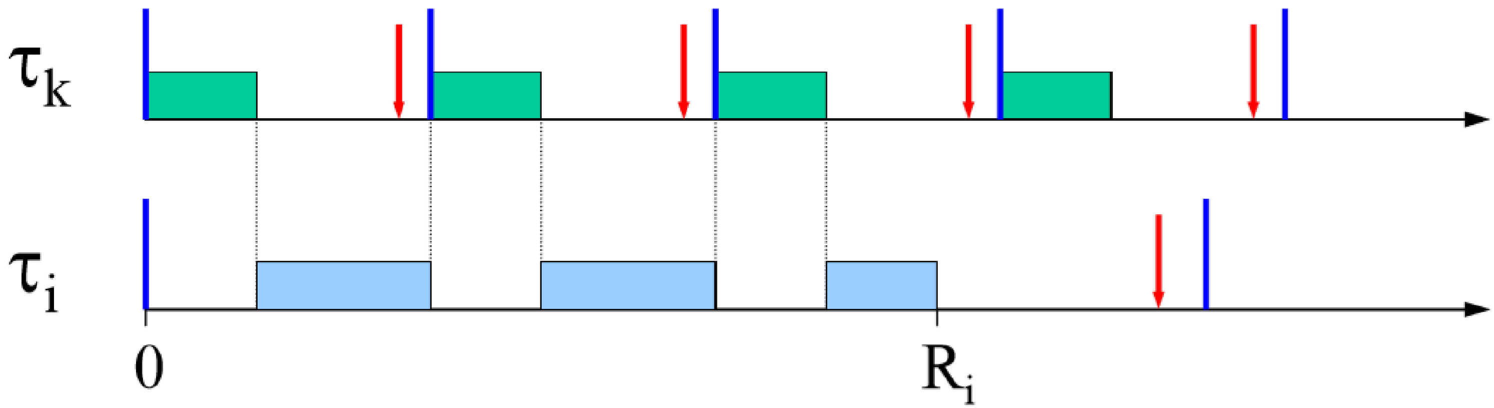
$$I_i = \sum_{D_k < D_i} C_k$$

- Compute its response time as

$$R_i = C_i + I_i$$

- Verify if $R_i \leq D_i$

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)

Response Time Calculation Algorithm

```
for i in 1..N loop -- for each process in turn
  n := 0
  $w_i^n := C_i$
loop
    calculate new $w_i^{n+1}$
    if  $w_i^{n+1} = w_i^n$ then
      $R_i = w_i^n$
      exit value found
    end if
    if  $w_i^{n+1} > T_i$  then
      exit value not found
    end if
    n := n + 1
  end loop
end loop
```

Task Set A

Task	Period T	ComputationTime C	Priority P
a	7	3	3
b	12	3	2
c	20	5	1

$$w_b^0 = 3$$

$$R_a = 3 \qquad w_b^1 = 3 + \left\lceil \frac{3}{7} \right\rceil 3 = 6$$

$$w_b^2 = 3 + \left\lceil \frac{6}{7} \right\rceil 3 = 6$$

$$R_b = 6$$

$$w_c^0 = 5$$

$$w_c^1 = 5 + \left\lceil \frac{5}{7} \right\rceil 3 + \left\lceil \frac{5}{12} \right\rceil 3 = 11$$

$$w_c^2 = 5 + \left\lceil \frac{11}{7} \right\rceil 3 + \left\lceil \frac{11}{12} \right\rceil 3 = 14$$

$$w_c^3 = 5 + \left\lceil \frac{14}{7} \right\rceil 3 + \left\lceil \frac{14}{12} \right\rceil 3 = 17$$

$$w_c^4 = 5 + \left\lceil \frac{17}{7} \right\rceil 3 + \left\lceil \frac{17}{12} \right\rceil 3 = 20$$

$$w_c^5 = 5 + \left\lceil \frac{20}{7} \right\rceil 3 + \left\lceil \frac{20}{12} \right\rceil 3 = 20$$

$$R_c = 20$$

Task Set B

Process	Period T	ComputationTime C	Priority P	Response time R
a	80	40	1	80
b	40	10	2	15
c	20	5	3	5

- The combined utilization is 1.0
- This was above the utilization threshold for three tasks (0.78), therefore it failed the test
- The response time analysis shows that the task set will meet all its deadlines

Response Time Analysis

- Is sufficient and necessary (exact)
- If the task set passes the test they will meet all their deadlines; if they fail the test then, at run-time, a task will miss its deadline (unless the computation time estimations themselves turn out to be pessimistic)

Sporadic Tasks

- Sporadics tasks have a minimum inter-arrival time
- They also require D<T
- The response time algorithm for fixed priority scheduling works perfectly for values of D less than T as long as the stopping criteria becomes

$$W_i^{n+1} > D_i$$

- It also works perfectly well with any priority ordering — hp(i) always gives the set of higher-priority tasks

Aperiodic Tasks

- These do not have minimum inter-arrival times
- Can run aperiodic tasks at a priority below the priorities assigned to hard processes, therefore, they cannot steal, in a pre-emptive system, resources from the hard processes
- This does not provide adequate support to soft tasks which will often miss their deadlines
- To improve the situation for soft tasks, a server can be employed

Execution-time Servers

- A server:
  - Has a capacity/budget of C that is available to its client tasks (typically aperiodic tasks)
  - When a client runs it uses up the budget
  - The server has a replenishment policy
  - If there is currently no budget then clients do not run
  - Hence it protects other tasks from excessive aperiodic activity

Periodic Server (PS)

- Budget C
- Replenishment Period T, starting at say 0
- Client ready to run at time 0 (or T, 2T etc) runs while budget available, is then suspended
- Budget ‘idles away’ if no clients

- Analyzed as a periodic task

Deferrable Server (DS)

- Budget C
- Period T – replenished every T time units (back to C)
  - For example 10ms every 50ms
- Anytime budget available clients can execute
- Client suspended when budget exhausted
- DS is referred to as *bandwidth preserving*
  - Retain capacity as long as possible
- PS is not bandwidth preserving

Task Sets with D < T

- For D = T, Rate Monotonic priority ordering is optimal
- For D < T, Deadline Monotonic priority ordering is optimal

$$D_i < D_j \Rightarrow P_i > P_j$$

- Response time analysis is applicable “as is” to task sets with D ≤ T

D < T Example Task Set

Task	Period T	Deadline D	ComputationTime C	Priority P	Response time R
a	20	5	3	4	3
b	15	7	3	3	6
c	10	10	4	2	10
d	20	20	3	1	20

Proof that DMPO is Optimal

- Deadline monotonic priority ordering (DMPO) is optimal if any task set, `Q`, that is schedulable by priority scheme, `W`, is also schedulable by DMPO

- The proof of optimality of DMPO involves transforming the priorities of `Q` (as assigned by `W`) until the ordering is DMPO
- Each step of the transformation will preserve schedulability

DMPO Proof Continued

- Let `i` and `j` be two tasks (with adjacent priorities) in `Q` such that under `W`: $P_i > P_j \wedge D_i > D_j$
- Define scheme `W'` to be identical to `W` except that tasks `i` and `j` are swapped

Consider the schedulability of `Q` under `W'`

- All tasks with priorities greater than P_i will be unaffected by this change to lower-priority tasks
- All tasks with priorities lower than P_j will be unaffected; they will all experience the same interference from `i` and `j`
- Task `j`, which was schedulable under `W`, now has a higher priority, suffers less interference, and hence must be schedulable under `W'`

DMPO Proof Continued

- All that is left is the need to show that task i, which has had its priority lowered, is still schedulable
- Under `W`

$$R_j < D_j, D_j < D_i \; and \; D_i \le T_i$$

- Hence task i only interferes once during the execution of j
- It follows that:

$$R'_i = R_j \le D_j < D_i$$

- It can be concluded that task `i` is schedulable after the switch
- Priority scheme `W'` can now be transformed to `W"` by choosing two more tasks that are in the wrong order for DMP and switching them

СПАСИБО ЗА ВНИМАНИЕ

hbd@cs.msu.su