Real-time Systems

Week 6:
Periodic real time scheduling

Ngo Lam Trung

Dept. of Computer Engineering

Contents

- Notation of periodic real-time tasks
- Periodic scheduling algorithms
  - Timeline Scheduling
  - Earliest Deadline First
  - Rate Monotonic
  - Deadline Monotonic
  - Earliest Deadline First (modified)

Example

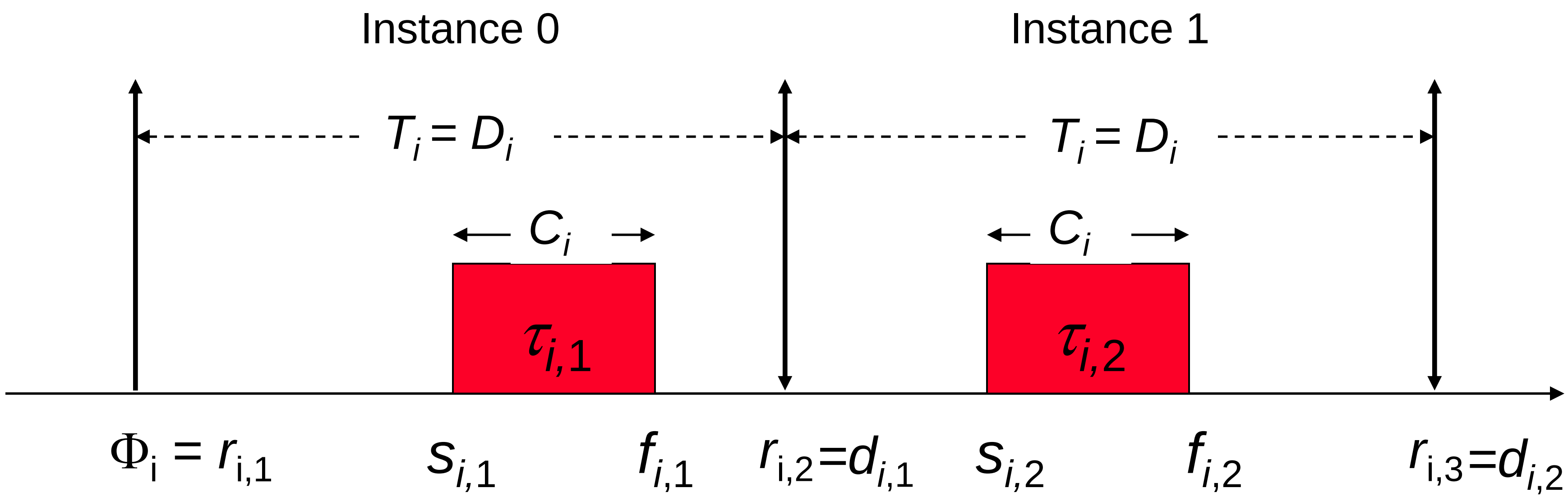
- 3 tasks:
  - Task 1: period 200 ms, computation time 50 ms
  - Task 2: period 100 ms, computation time 50 ms
  - Task 3: period 400 ms, computation time 50 ms
  - Is it schedulable?
- If task 4 is added
  - Task 4: period 200 ms, computation time 30 ms
  - Is it schedulable?

Notation of periodic task set

- Γ: a set of periodic tasks
- τ_i: a generic periodic task
- $\tau_{i,j}$: the j-th instance of task τ_i
- $r_{i,j}$: the release time of $\tau_{i,j}$
- $\Phi_i = r_{i,1}$: the phase of τ_i
- D_i: the relative deadline of τ_i
- $d_{i,j}$: the absolute deadline of $\tau_{i,j}$
  - $d_{i,j} = \Phi_{i,j} + (j\text{-}1)T_i + D_i$
- $s_{i,j}$: the start time of $\tau_{i,j}$
- $f_{i,j}$: the finishing time of $\tau_{i,j}$

Periodic task notations

- Task τ_i's timing parameters



- Task τ_i's timing parameters is ***feasible*** if all its instances finish within their absolute deadline

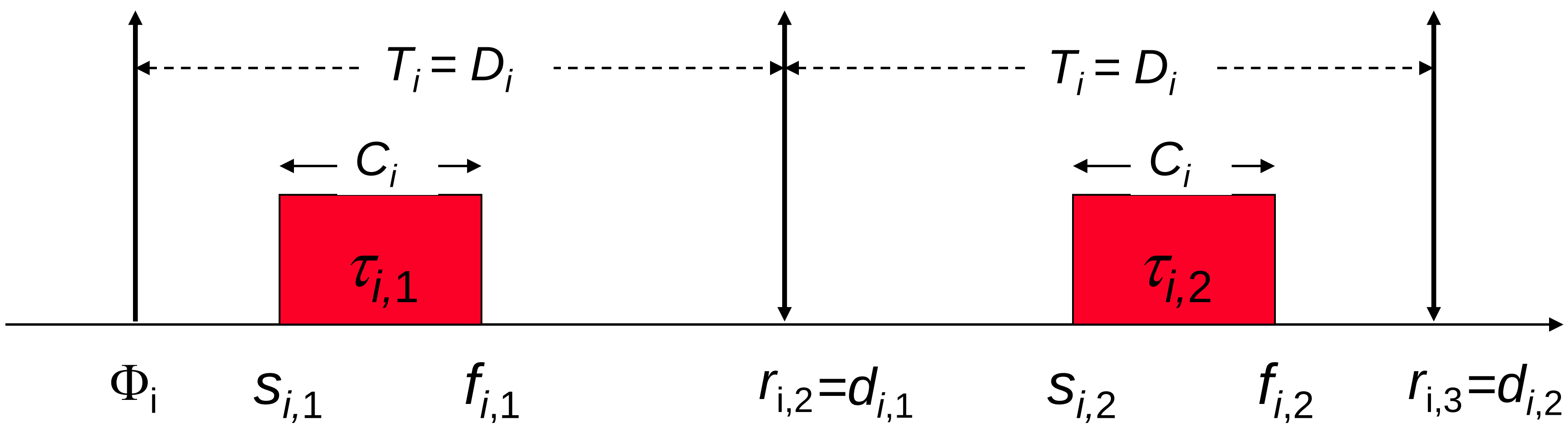
- A set Γ of periodic tasks is ***schedulable*** if all tasks in Γ are feasible

Assumptions

- **A1**. The instance of τ_i is regularly activated at a constant rate. (Period T_i)
- **A2**. All instances of a task have the same worst-case execution time C_i.
- **A3**. All instances of a task have the same relative deadline D_i and D_i=T_i.
- **A4**. All tasks are independent; no precedence & resource constraints
- **A5**. No task can suspend itself, for example on I/O operations
- **A6**. All tasks are fully pre-emptible.
- **A7**. All overheads in the kernel are ignored.

Simplified task parameters

- A task under assumptions **A1**-**A4** can be characterized by 3 parameters.
  - Task set: $\Gamma = \{\tau_i(\Phi_i, T_i, C_i),\ i=1,\ldots,n\}$
  - Release time: $r_{i,k} = \Phi_i + (k-1)T_i$
  - Absolute deadline: $d_{i,k} = \Phi_i + kT_i$

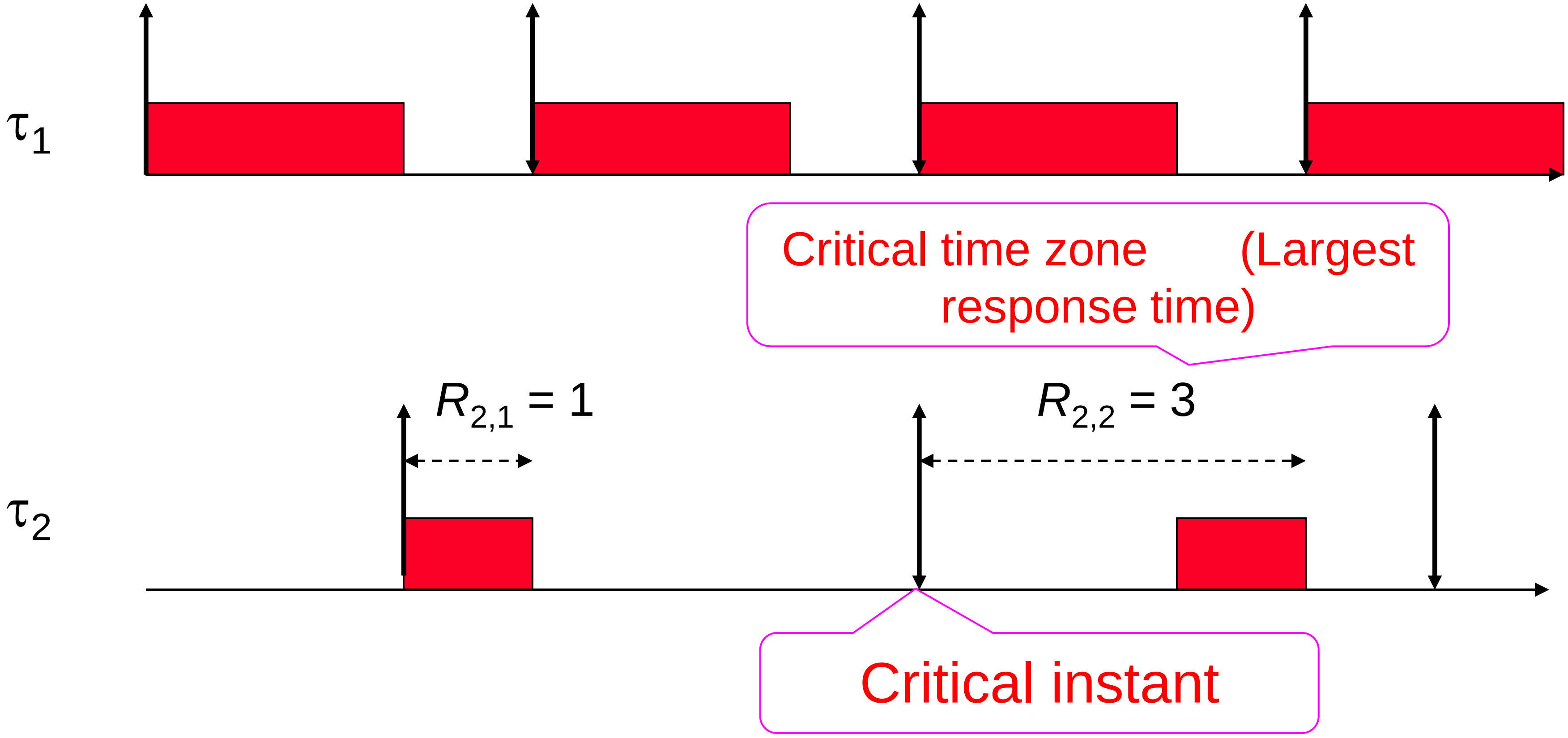


Periodic task parameters

- Response time:
  - Duration from the release time to finishing time
  - $R_{i,k} = f_{i,k} - r_{i,k}$
- Critical instant:
  - The time at which the release of a task will produce the largest response time
- Critical time zone:
  - Response time with respect to the critical instant

An example of critical instance

- $\Gamma = \{ \tau_1(0,3,2), \tau_2(2,4,1)\}$
- Assume that τ_2 has lower priority than τ_1.
- When is the critical instant of τ_2?

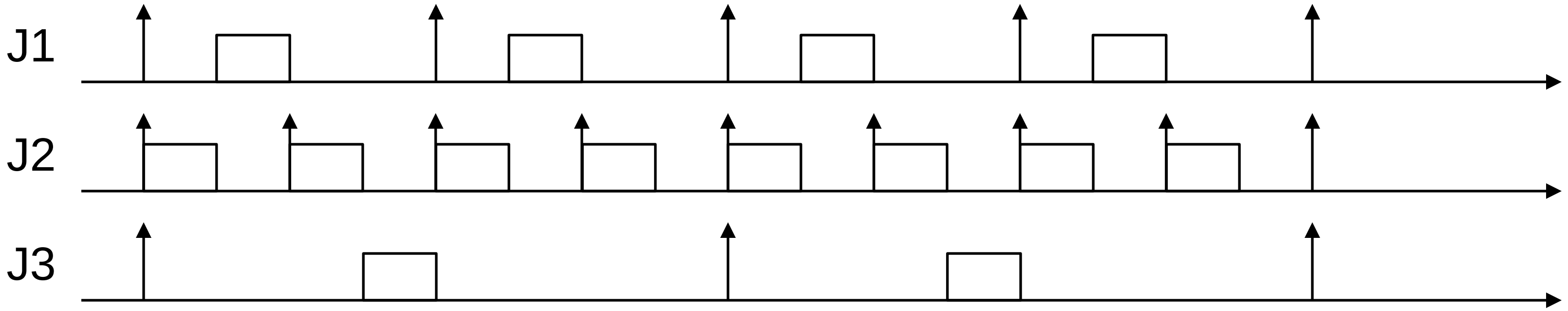


Hyperperiod

- Given a set of 3 tasks, all activate at $t = 0$:
  - Task 1: period 200 ms, computation time 50 ms
  - Task 2: period 100 ms, computation time 50 ms
  - Task 3: period 400 ms, computation time 50 ms
- How long will the schedule repeat itself?

$$H = lcm(T_1, \ldots, T_n)$$

lcm: least common multiply

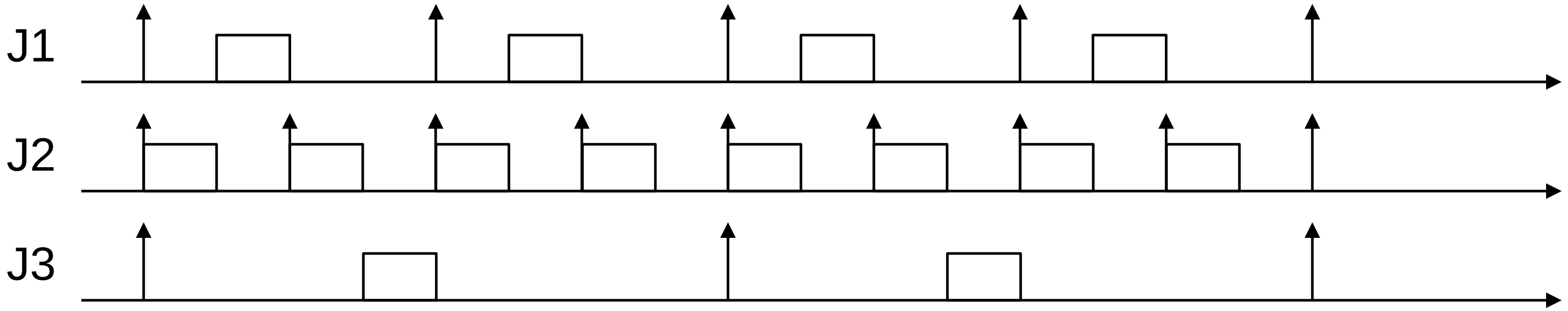


Hyper period

- Given a set of 3 tasks, all activate at *t = 0*:
  - Task 1: period 200 ms, computation time 50 ms
  - Task 2: period 100 ms, computation time 50 ms
  - Task 3: period 400 ms, computation time 50 ms
- How long will the schedule repeat itself?

$$H = lcm(T_1, \dots, T_n)$$

lcm: least common multiply



Processor utilization factor

- Processor utilization for n tasks

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

- *U* represents how many percent of processor resource is utilized by a given task set.

- Example 1:
  - U3 = 50/200+50/100+50/400 = 87.5%
  - U4 = 50/200+50/100+50/400+30/200 = 102.5%

Utilization factor vs schedulability?

- If U > 1:
  - Let H be the hyperperiod

$$U > 1 \Rightarrow UH > H$$

$$\Rightarrow \sum_{i=1}^{n} \frac{H}{T_i} C_i > H$$

  - $(H/T_i)Ci$: total CPU time requested by T_i during H

  ➔total requestime during $[0, H)$ is bigger than H

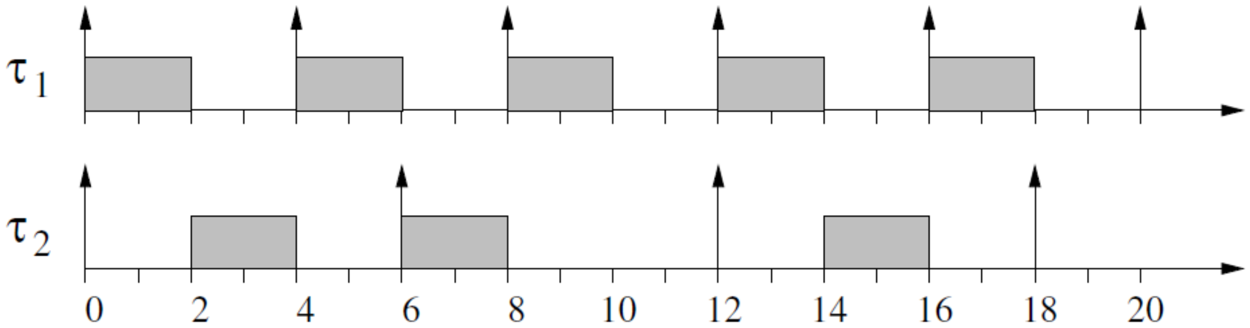
  ➔the task set is not schedulable

- What if U < 1: the task set is schedulable?

  ➔not sure!

Utilization factor vs schedulability?

- Consider two tasks T_1, T_2 (T1 has higher priority)



  - Schedulable?
  - U = ?
  - What if C1 or C2 increase by epsilon?

  ➔ U < 1 does not guarantee schedulability

- Given a task set Γ, its schedulability depends on
  - The parameters of the tasks
  - The scheduling algorithm

Processor utilization factor

- Given a scheduling algorithm A and a task set Γ, there will be a upper bound value of *U*

$$\boldsymbol{U_{ub}(\Gamma, A)}$$

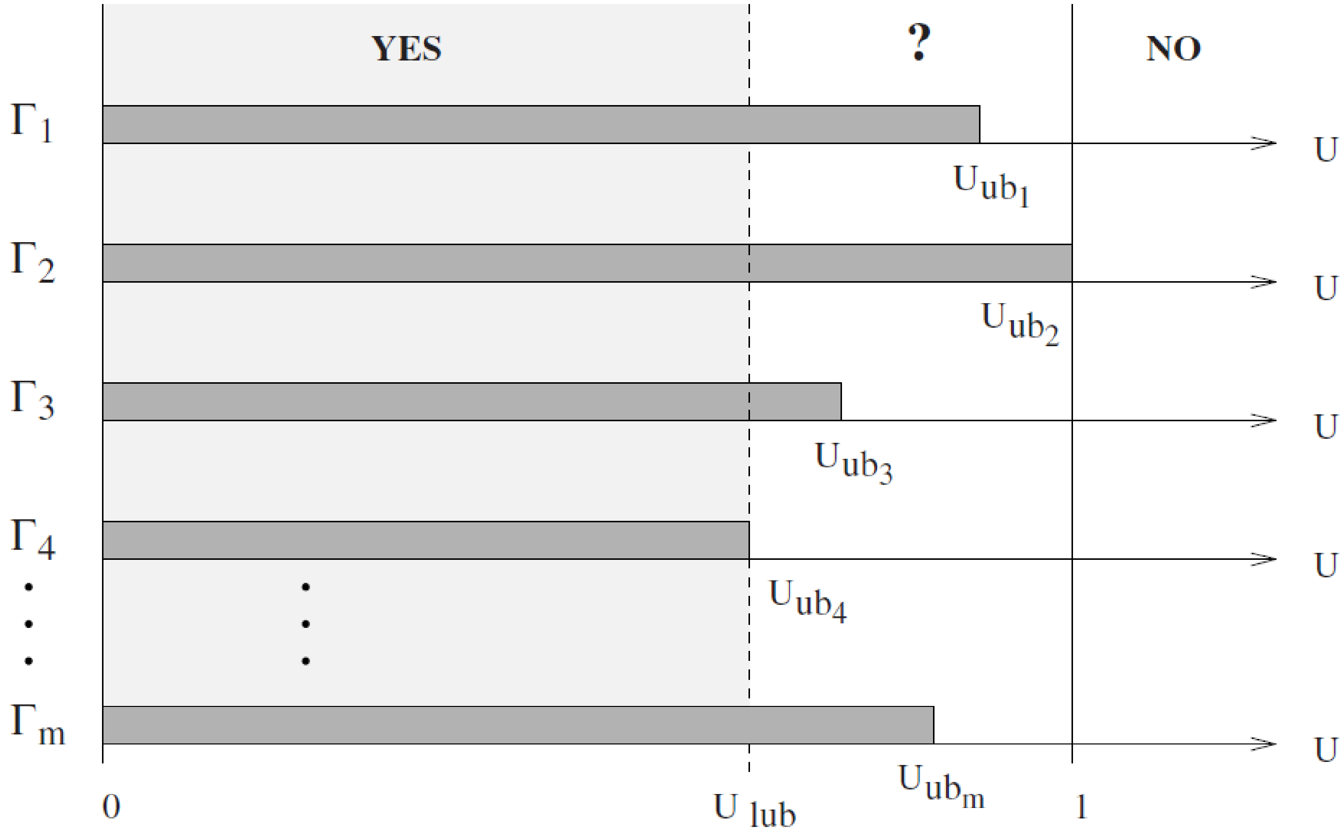
  - $U > U_{ub}(\Gamma, A)$: Γ is not schedulable by A
  - $if\ U = U_{ub}(\Gamma, A)$: Γ fully utilizes the processor

- For a given algorithm A, let

$$U_{lub}(A) = \min_{\Gamma} U_{ub}(\Gamma, A)$$

  - All task set having $U \leq U_{lub}(A)$ will be schedulable by A
  - $\text{if } 1 > U > U_{lub}(A)$, schedulability depends on actual tasks parameters (activation time, period…)

Processor utilization factor



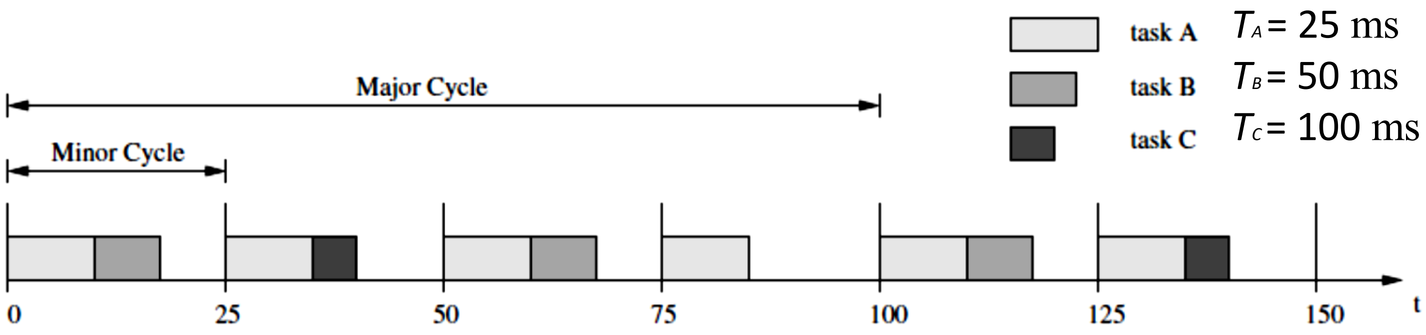
Utilization vs schedulability

Algorithms for periodic scheduling

- Timeline Scheduling ($D = T$)
- Earliest Deadline First ($D = T$)
- Rate Monotonic ($D = T$)
- Deadline Monotonic ($D \leq T$)
- Earliest Deadline First ($D \leq T$)

Algorithm 1: Timeline Scheduling

- Divide the timeline into Minor Cycles and Major Cycles
  - Major Cycle = $lcm(T_i)$ = H (least common multiply)
  - Minor Cycle = $\gcd(T_i)$ (greatest common divisor)
- Scheduling and implementation:
  - Schedule the task execution in each minor cycle of a major cycle
  - Set up a timer with period equal to minor cycle
  - The main function synchronize task execution with timer event



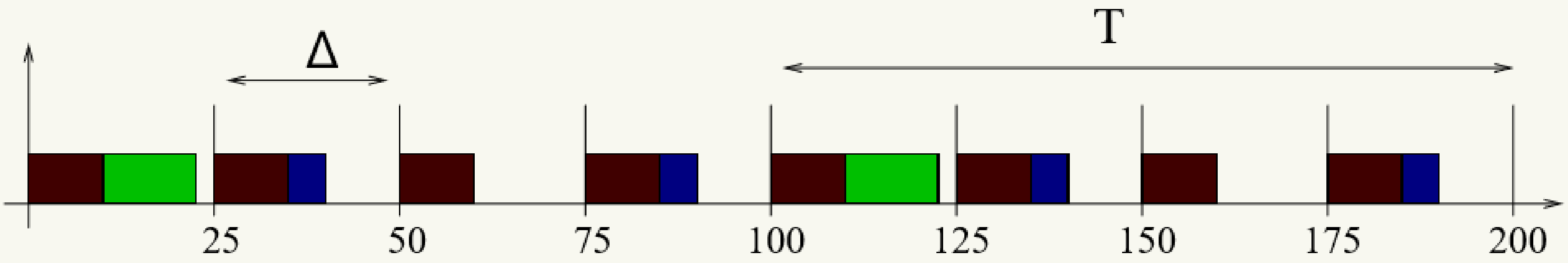
Example

- Consider a taskset $\Gamma = \{\tau_1, \tau_2, \tau_3\}$
  - Periodic tasks $\tau_i = (C_i, D_i, T_i)$, $D_i = T_i$
  - $T_1 = 25ms$, $T_2 = 50ms$, $T_3 = 100ms$

1. Minor Cycle $\Delta = gcd(25, 50, 100) = 25ms$
2. Major Cycle $T = lcm(25, 50, 100) = 100ms$
3. Compute a schedule that respects the task periods
   - Allocate tasks in slots of size $\Delta = 25ms$
   - The schedule repeats every $T = 100ms$
   - τ_1 must be scheduled every $25ms$, τ_2 must be scheduled every $50ms$, τ_3 must be scheduled every $100ms$
   - In every minor cycle, the tasks must execute for less than $25ms$

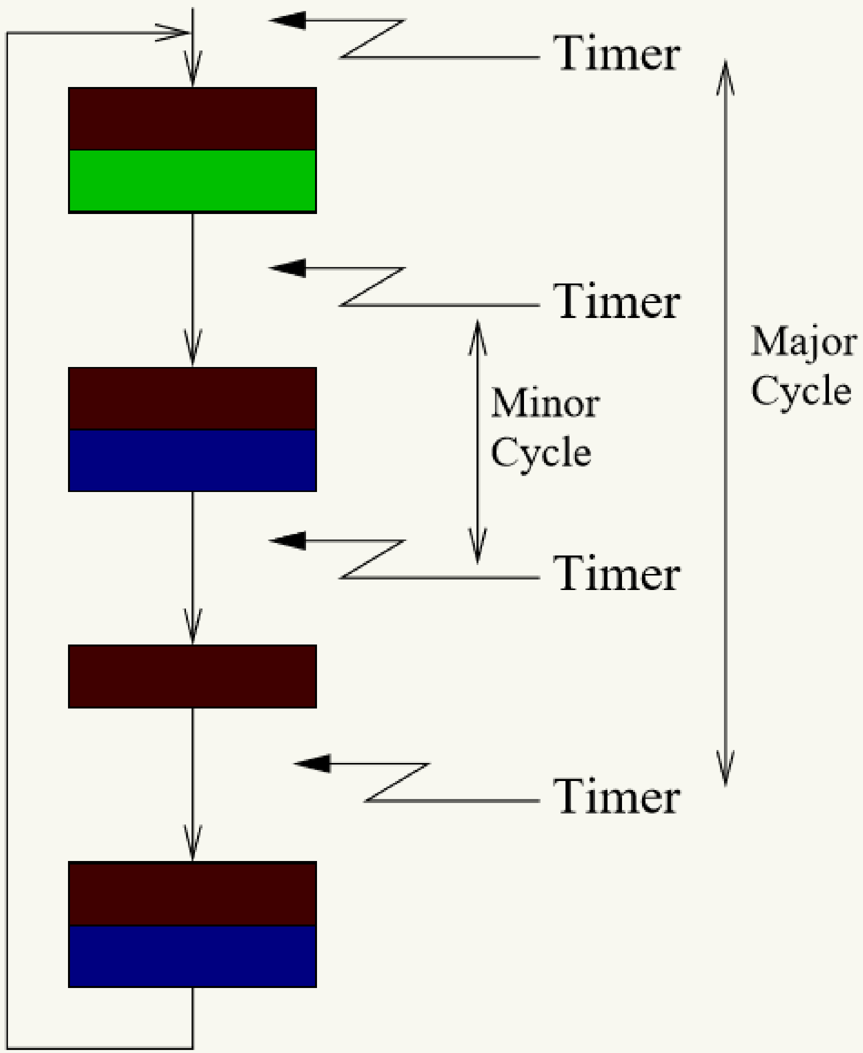
Example

- The schedule repeats every 4 minor cycles
  - τ_1 must be scheduled every $25ms \Rightarrow$ scheduled in every minor cycle
  - τ_2 must be scheduled every $50ms \Rightarrow$ scheduled every 2 minor cycles
  - τ_3 must be scheduled every $100ms \Rightarrow$ scheduled every 4 minor cycles



- First minor cycle: $C_1 + C_3 \leq 25ms$
- Second minor cycle: $C_1 + C_2 \leq 25ms$

Example



- Periodic timer firing every minor cycle
- Every time the timer fires...
- ...Read the scheduling table and execute the appropriate tasks
- Then, sleep until next minor cycle

Algorithm 1: Timeline Scheduling

- Advantage:
  - Simple, does not require RTOS
  - No context switching, minimal run-time overhead.
- Disadvantages:
  - Domino effect if task does not terminate on time
  - May need to divide task in to small pieces
  - Difficult to handle aperiodic and long tasks
  - Sensitive to task parameter changes (period, execution time…)

Algorithm 2: Ealiest Deadline First (EDF)

- Pre-emptible task set, dynamic priorities
- All tasks instances are consider aperiodic tasks
- Priority and scheduling of task is based on the instances' absolute deadline:

$$d_{i,j} = \Phi_i + (j-1)T_i + Di$$

- Proof of optimality is the same as with aperiodic tasks
- How to analyze schedulability/feasibility?

Schedulability analysis of EDF

- Theorem: a set of periodic tasks is schedulable with EDF if and only if

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

- Proof:

Schedulability analysis of EDF

- Theorem: a set of periodic tasks is schedulable with EDF if and only if

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

- Proof:
  - If U > 1: not enough CPU resource → not schedulable
  - If U <= 1: show that the task set is schedulable

  Contradiction: provided the task set is not schedulable

  Let t_2: time that time-overflow happens

  t_1: starting of **continuous utilization** $[t_1, t_2]$

  Total processor computation time demanded in $[t_1, t_2]$

$$C_p(t_1, t_2) = \sum_{r_k \geq t_1, d_k \leq t_2} C_k = \sum_{i=1}^{n} \left\lfloor \frac{t_2 - t_1}{T_i} \right\rfloor C_i$$

Schedulability analysis of EDF

- If the task set is not schedulable

$$C_p(t_1, t_2) > t_2 - t_1$$

- However

$$C_p(t_1, t_2) \leq \sum_{i=1}^{n} \frac{t_2 - t_1}{T_i} C_i = (t_2 - t_1)U$$

- Then we have

$$(t_2 - t_1)U > t_2 - t_1$$

$$\rightarrow U > 1$$

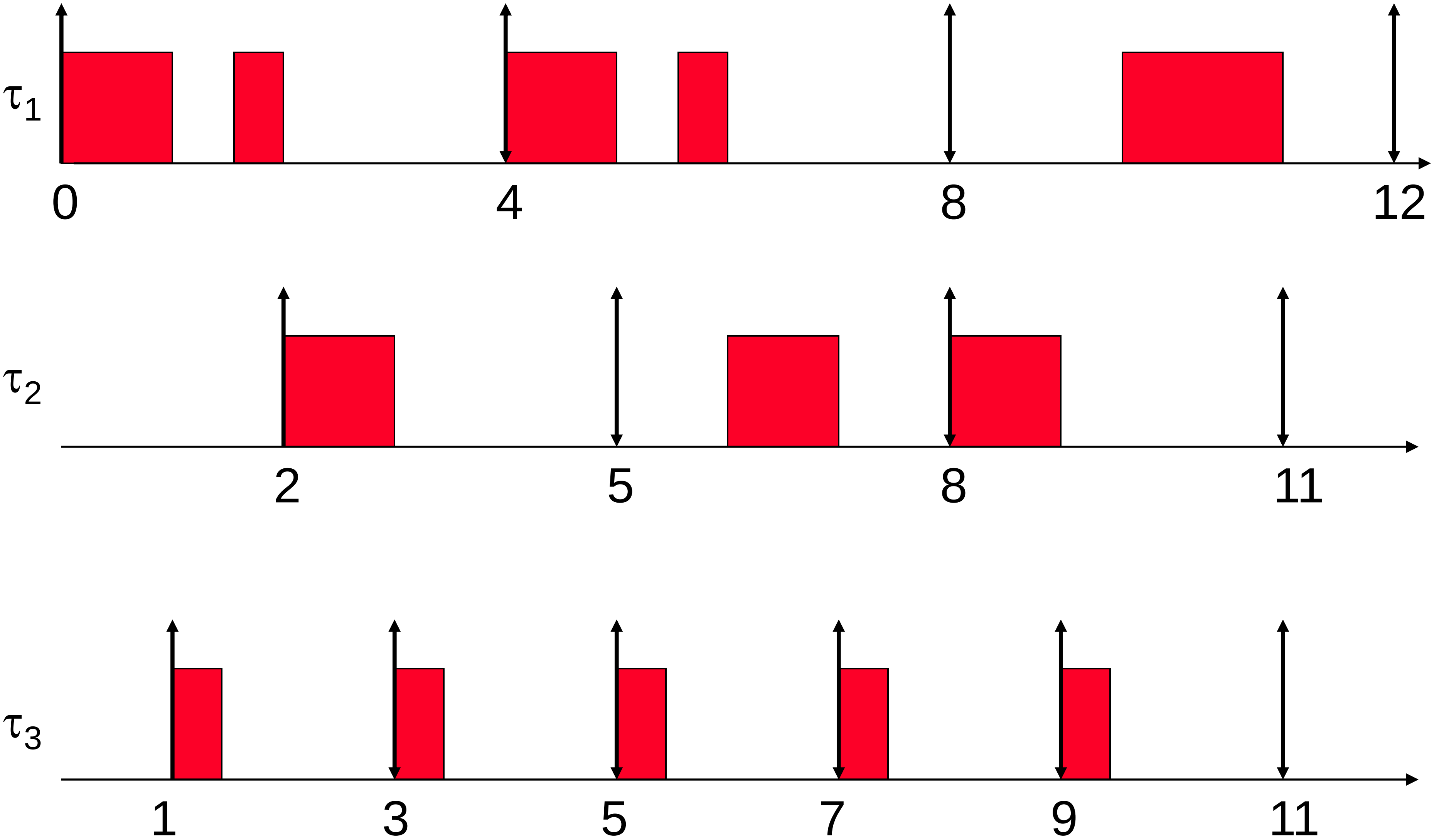
→ *contradiction*

An example of EDF scheduling

Task	1	2	3
ϕ_i	0	2	1
C_i	1.5	1	0.5
T_i	4	3	2

- Assume that $T_i = D_i$
- Preemptive task set

An example of EDF scheduling

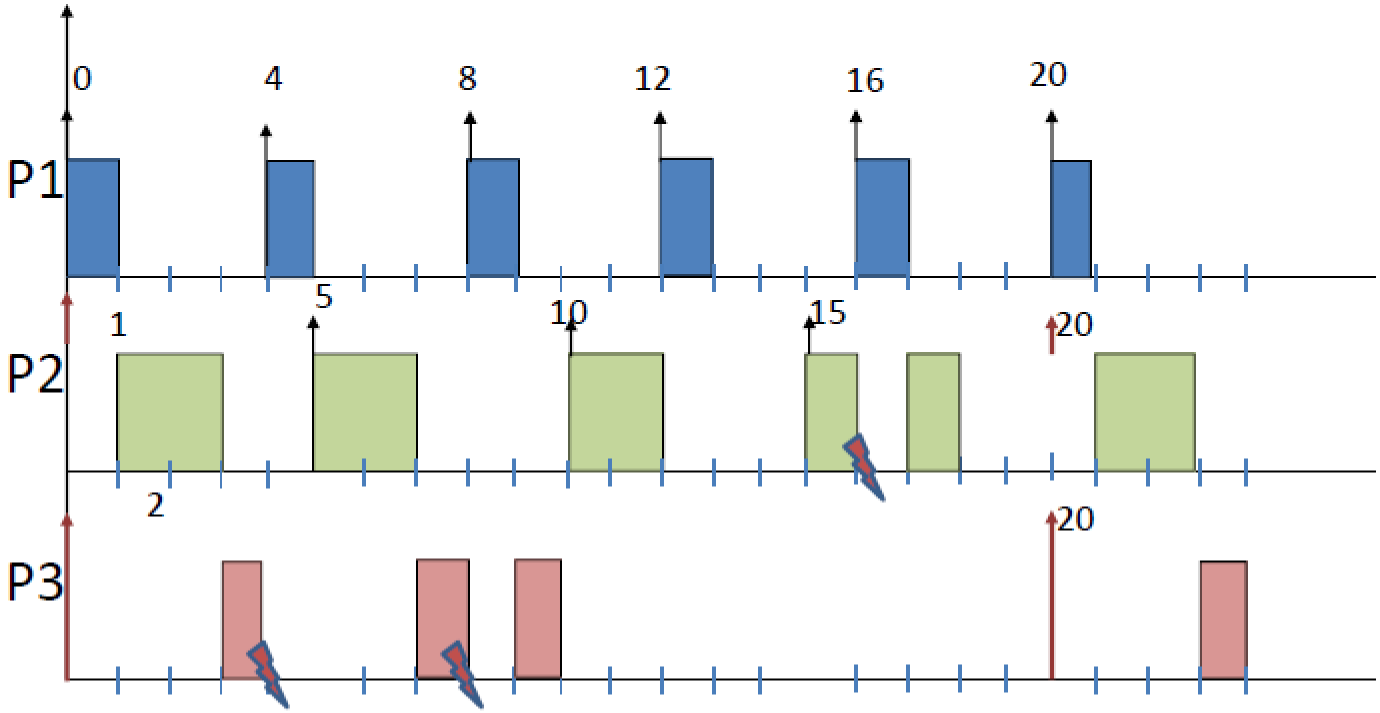


Algorithm 3: Rate Monotonic (RM)

- Pre-emptible task set, static scheduling with fixed priorities
- Priority of task is based on the task's request rate: **higher rates (shorter periods) correspond to higher priorities**
- Optimality: RM is optimal among all fixed-priority algorithms
- Schedulability/feasibility analysis: U_{lub}

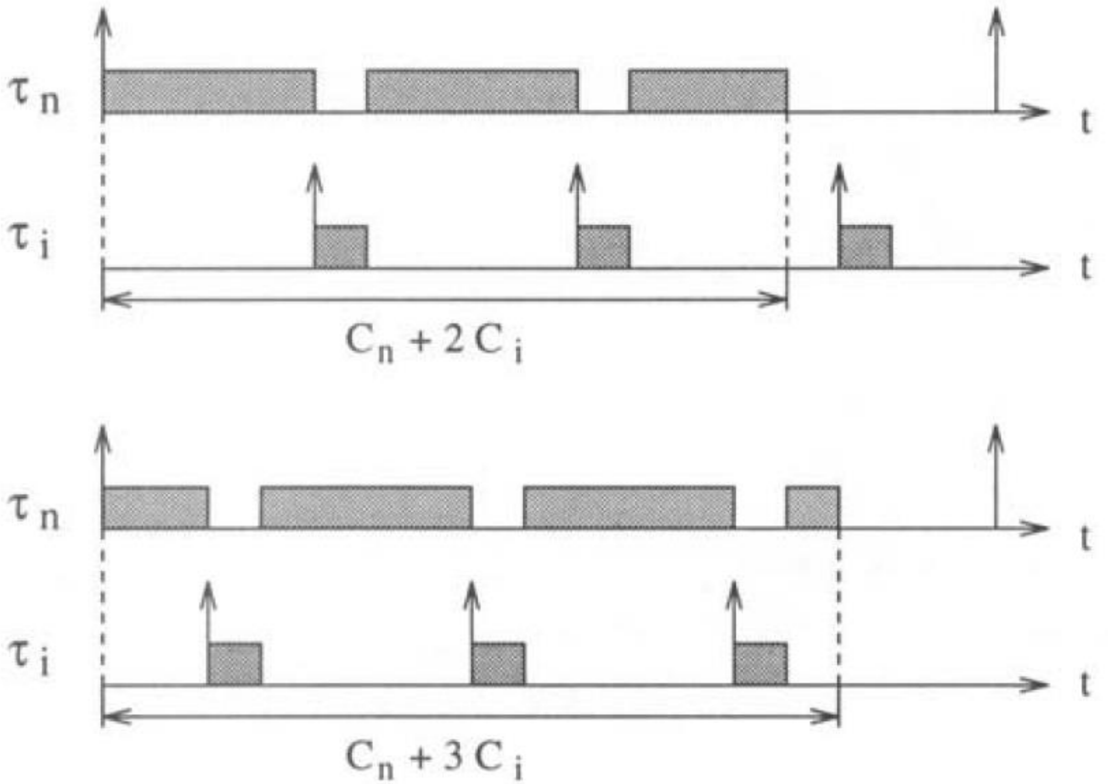
Example

- T1: c=1, p=d=4
- T2: c=2, p=d=5
- T3: c=3, p=d=20



Proof of optimality (1)

- For any task T, the critical instance occurs when it is released simultaneously with all higher-priority tasks



τ_n: task with lowest priority

τ_i: any other task

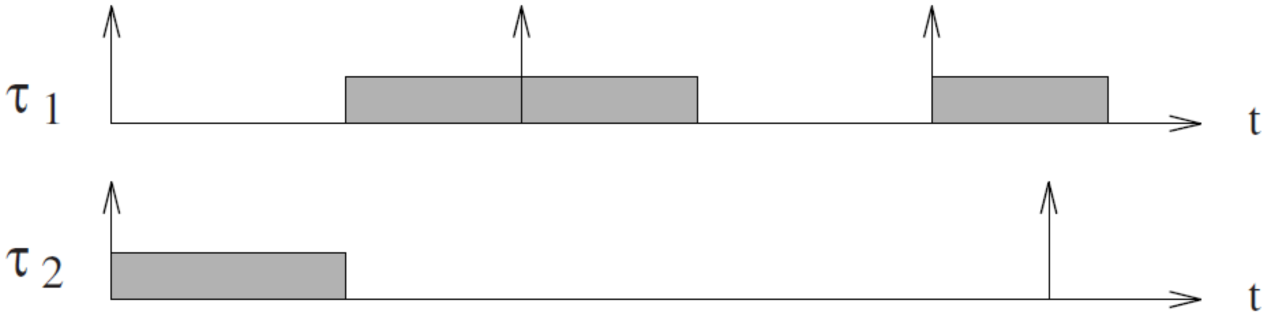
➔ Task schedulability can be checked at its critical instance

Proof of optimality (2)

- If a task set Γ is schedulable by any fixed priority algorithm, it will be schedulable by RM
  - Given two tasks τ_1, τ_2 with T1 < T2, in critical instants
  - Provided the scheduled violates RM ➔ τ_2 has higher priority

    → the schedule is feasible if

$$C_1 + C_2 \leq T_1$$

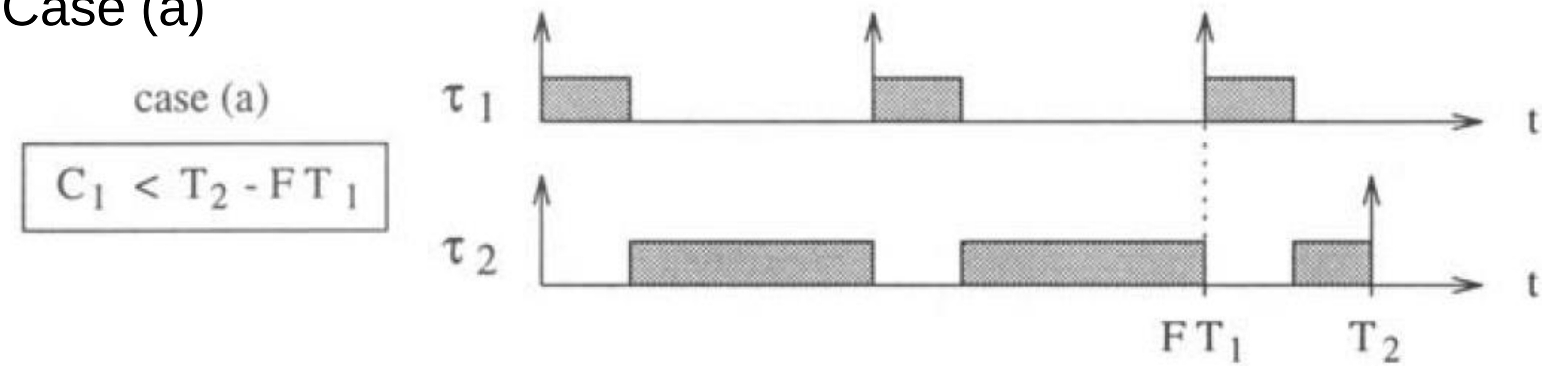


- Show that exchanging priority of T1 and T2 will result in feasible schedule

  ➔ *Homework*

Proof of optimality (3)

- Consider if τ_1, τ_2 are scheduled by RM, τ_1 has higher priority
- Let $F = \lfloor T_2/T_1 \rfloor$: the number of T1 contained entirely in T2
- Case (a)



As $$C_1 + C_2 \le T_1 \quad \rightarrow \quad FC_1 + FC_2 \le FT_1$$

Since F ≥ 1 $$FC_1 + C_2 \le FC_1 + FC_2 \le FT_1$$

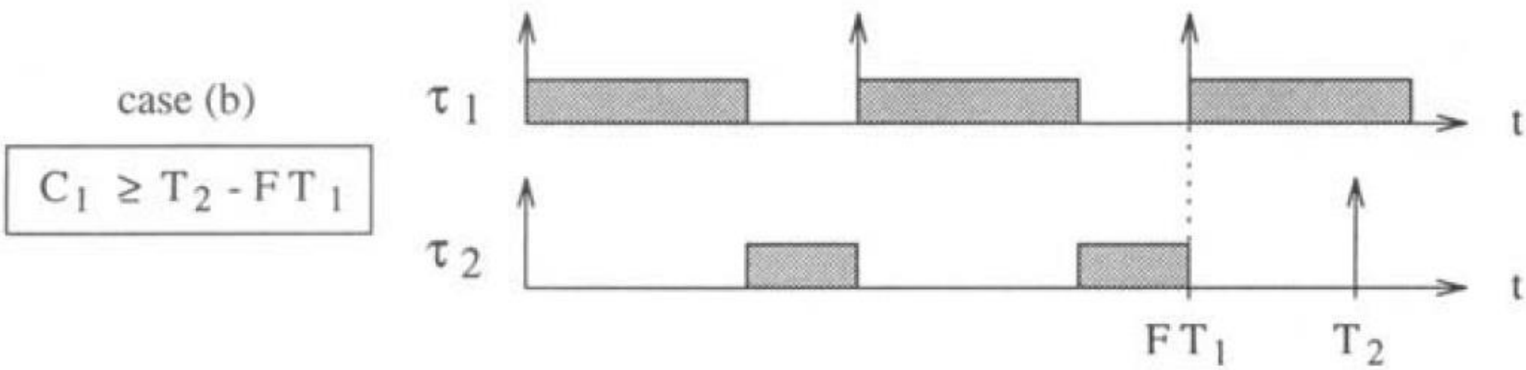
$$(F+1)C_1 + C_2 \le FT_1 + C_1$$

$$(F+1)C_1 + C_2 \le FT_1 + C_1 \le T_2$$

➔ The schedule by RM is feasible

Proof of optimality (4)

- Case (b)



As $C_1 + C_2 \leq T_1$ ➔ $FC_1 + FC_2 \leq FT_1$

Since F ≥ 1 $FC_1 + C_2 \leq FC_1 + FC_2 \leq FT_1$

➔ The schedule by RM is feasible

- Given τ_1, τ_2 if they are scheduled by any fixed priority algorithm, then they are schedulable by RM

➔ RM is optimal

RM schedulability: using U

- Necessary but not sufficient

$$U \leq 1$$

- Sufficient but not necessary (LL-bound)

$$U \leq n(2^{1/n} - 1)$$

As the number of tasks n increases to infinite

$$U \rightarrow ln2 = 0.69$$

n	U_{lub}
1	1.000
2	0.828
3	0.780
4	0.757
5	0.743

n	U_{lub}
6	0.735
7	0.729
8	0.724
9	0.721
10	0.718

Example

- T1(c=1,p=d=4), T2(c=1, p=d=5), T3(c=1, p=d=10)
- Is this tasks set schedulable by RM?

$$U = 1/4 + 1/5 + 1/10 = 0.55$$

$$n(2^{1/n} - 1) = 3(2^{1/3} - 1) =\approx 0.78$$

- We have

$$U \leq n(2^{1/n} - 1)$$

➔ Schedulable tasks set

Schedulability analysis of RM

- If $n(2^{1/n} - 1) < U \leq 1$ the tasks set might of might not be schedulable

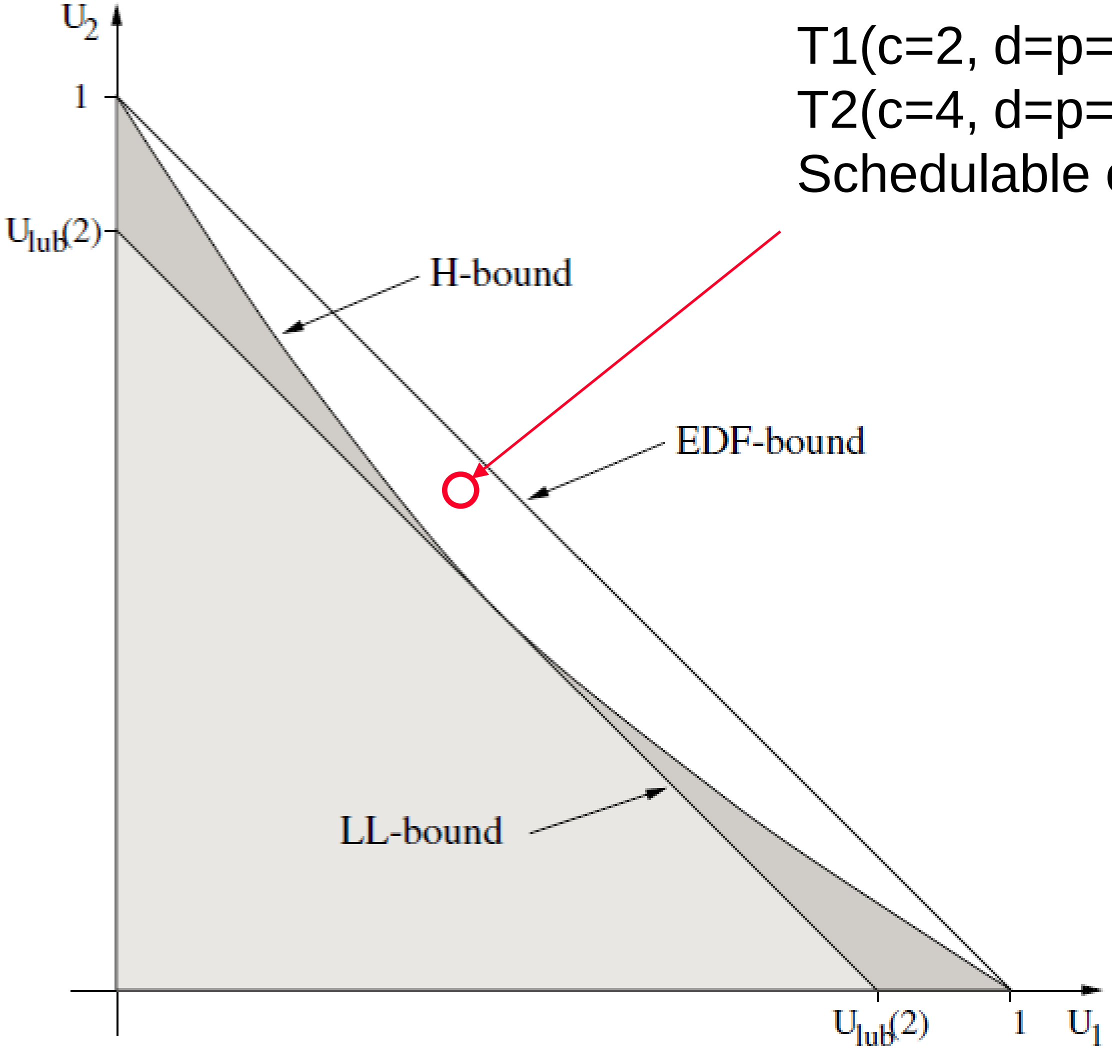
➔ Need to check manually

RM schedulability: using hyperbolic bound

- Given a set of periodic task with utilization factors U_i the tight bound for schedubility with RM is

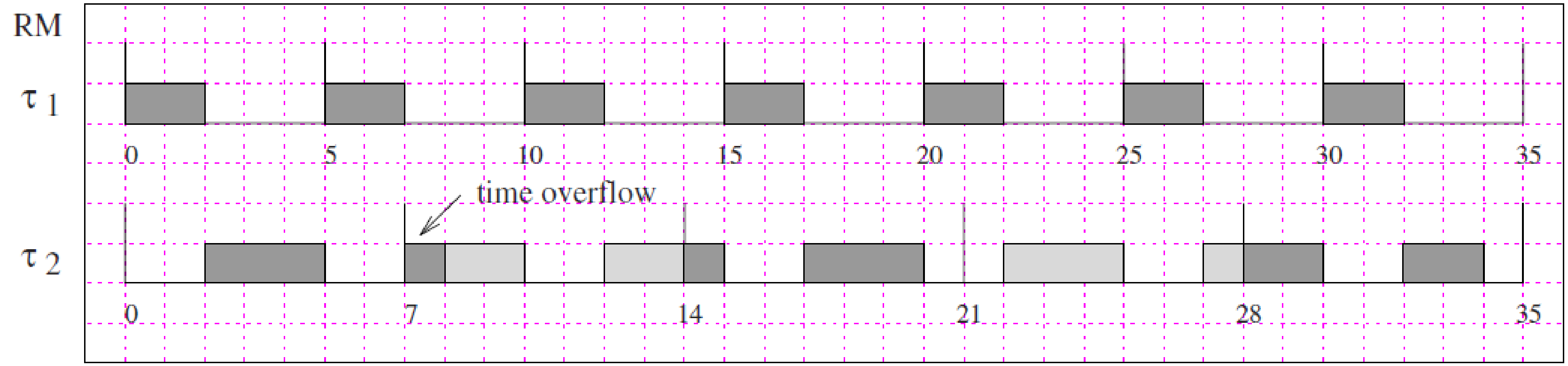
$$\prod_{i=1}^{n}(U_i + 1) \leq 2.$$

EDF vs RM

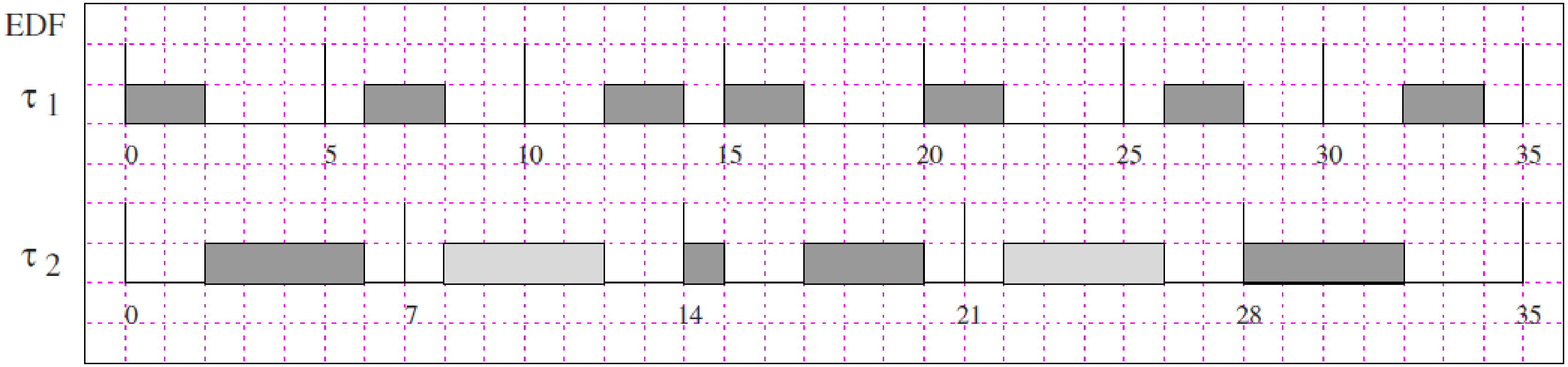


T1(c=2, d=p=5)
T2(c=4, d=p=7)
Schedulable or not?

EDF vs RM



(a)



(b)

EDF is dynamic algorithm ➔ able to produce feasible schedule when RM fails

EDF and RM comparison

- EDF: large overhead
  - Calculate time to deadline for all ready tasks
  - Assign priorities
  - Schedule based on new priorities
- RM is simpler to implement, requires less overhead

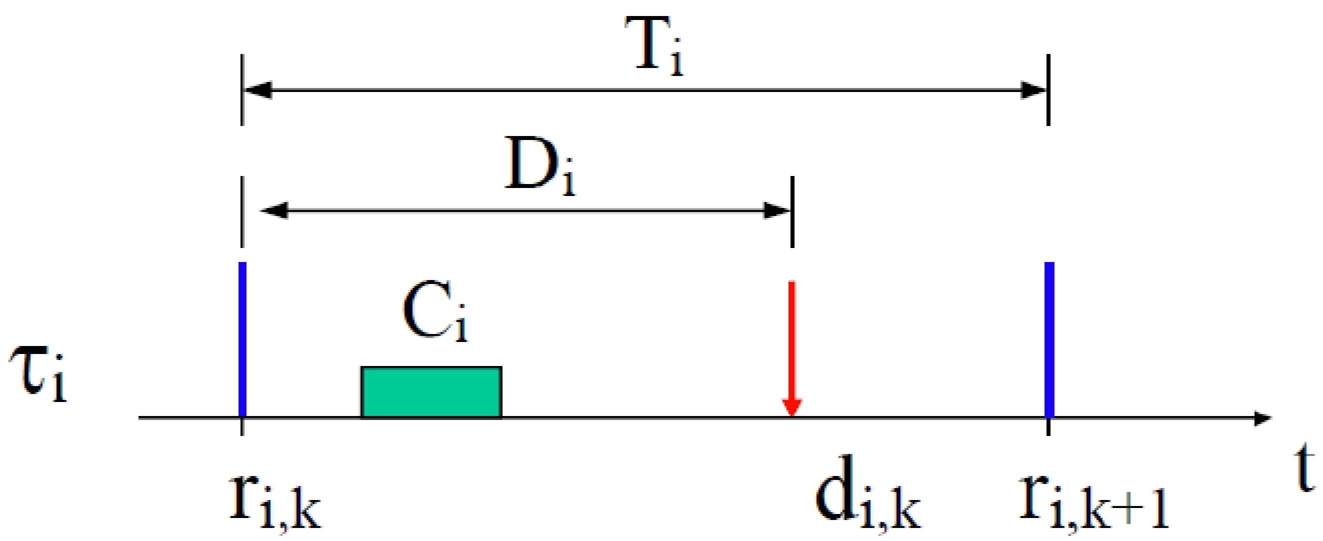
Assumptions for EDF and RM

- A3: Relative deadline equals to period

$$D_i = T_i$$

- Relax the assumption for more practical problems

$$D_i < T_i$$



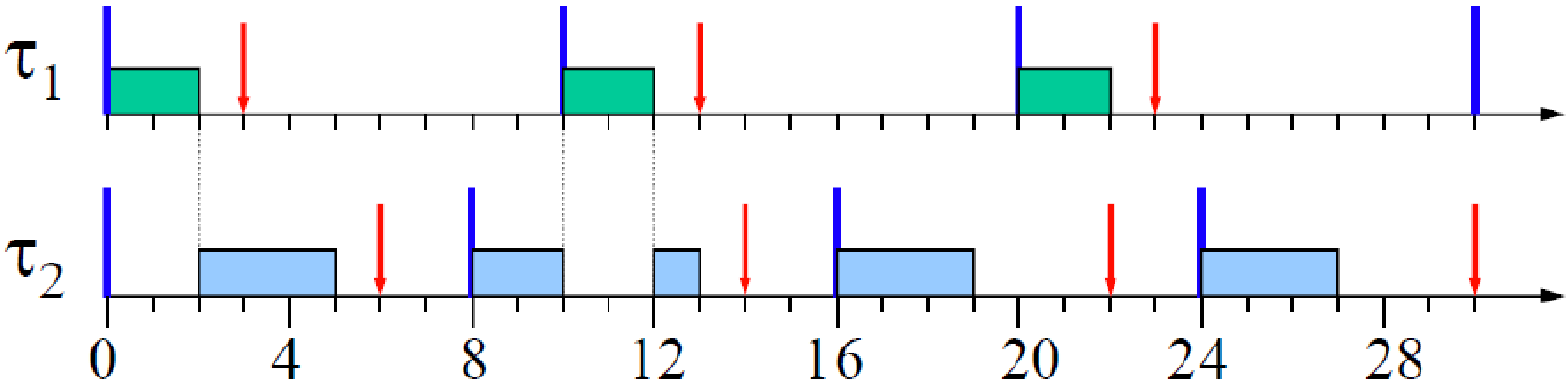
→ modified algorithms

Algorithm 4: Deadline Monotonic (DM)

- Each task is assigned a priority inversely proportional to its relative deadline

- Shorter deadlines imply higher priorities

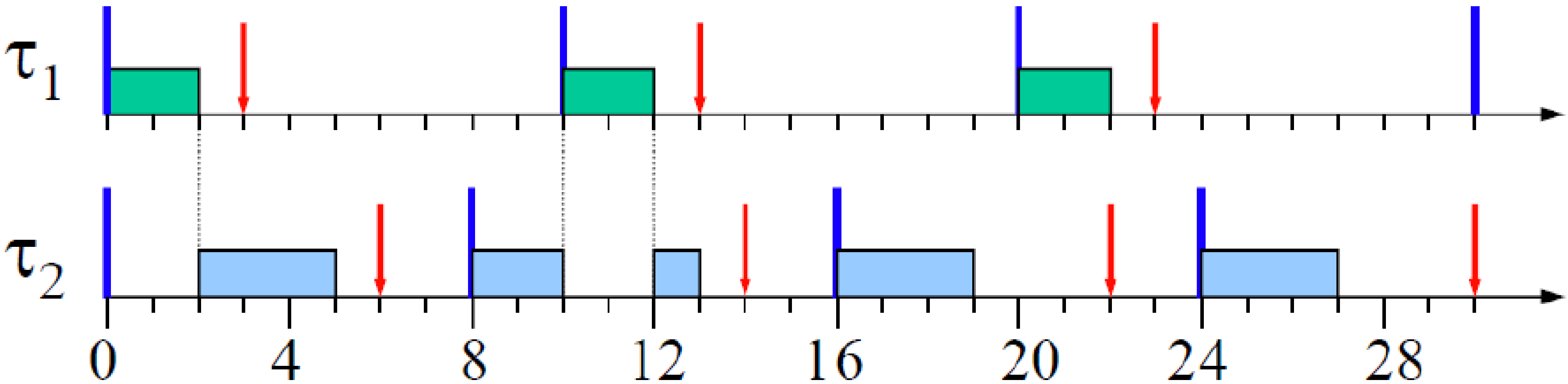
T1 (2,10,3) T2(3,8,6)
(C_i, T_i, D_i)



➔ Feasible schedule

Schedulability analysis of DM

T1 (2,10,3) T2(3,8,6)
(C_i, T_i, D_i)



- Processor utilization

$$U = 2/3 + 3/6 = 1.16 > 1$$

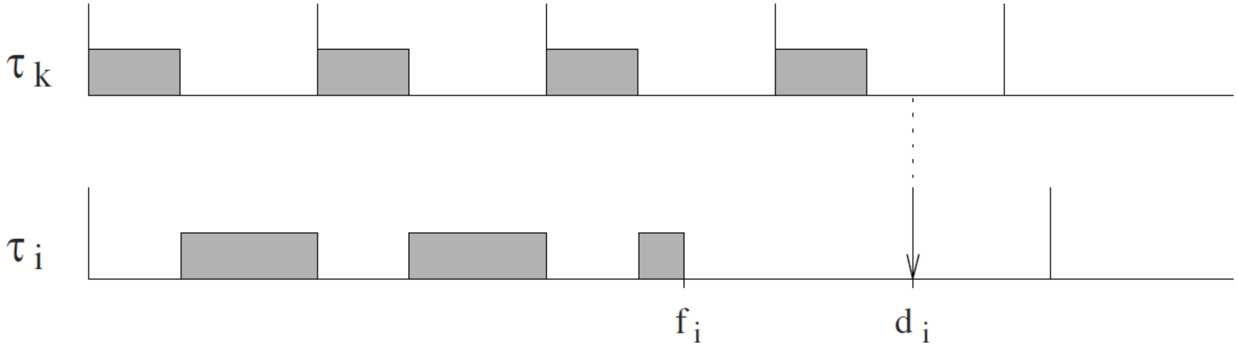
➔cannot be used for schedulability analysis

Schedulability analysis of DM

- The worst-case processor demand (at critical instances) must be met

- In the worst case: for each task τ, the sum of its processing time and the interference (preemption) imposed by higher priority tasks must be less than or equal to its relative deadline

$$\forall i : 1 \leq i \leq n \quad C_i + I_i \leq D_i$$



Schedulability based on response time

- Response time of task i

$$R_i = C_i + I_i,$$

- Interference by higher priority tasks

$$I_i = \sum_{j=1}^{i-1} \left\lceil \frac{R_i}{T_j} \right\rceil C_j.$$

- Then

$$R_i = C_i + \sum_{j=1}^{i-1} \left\lceil \frac{R_i}{T_j} \right\rceil C_j.$$

- R_i is calculated recursively until converged

Example 1

- Test the schedulability of the tasks set, present a feasible schedule if available

	C_i	T_i	D_i
τ_1	1	4	3
τ_2	1	5	4
τ_3	2	6	5
τ_4	1	11	10

Example 1

Step 0: $R_4^{(0)} = \sum_{i=1}^{4} C_i = 5,$ but $I_4^{(0)} = 5$ and $I_4^{(0)} + C_4 > R_4^{(0)}$
hence τ_4 does not finish at $R_4^{(0)}$.

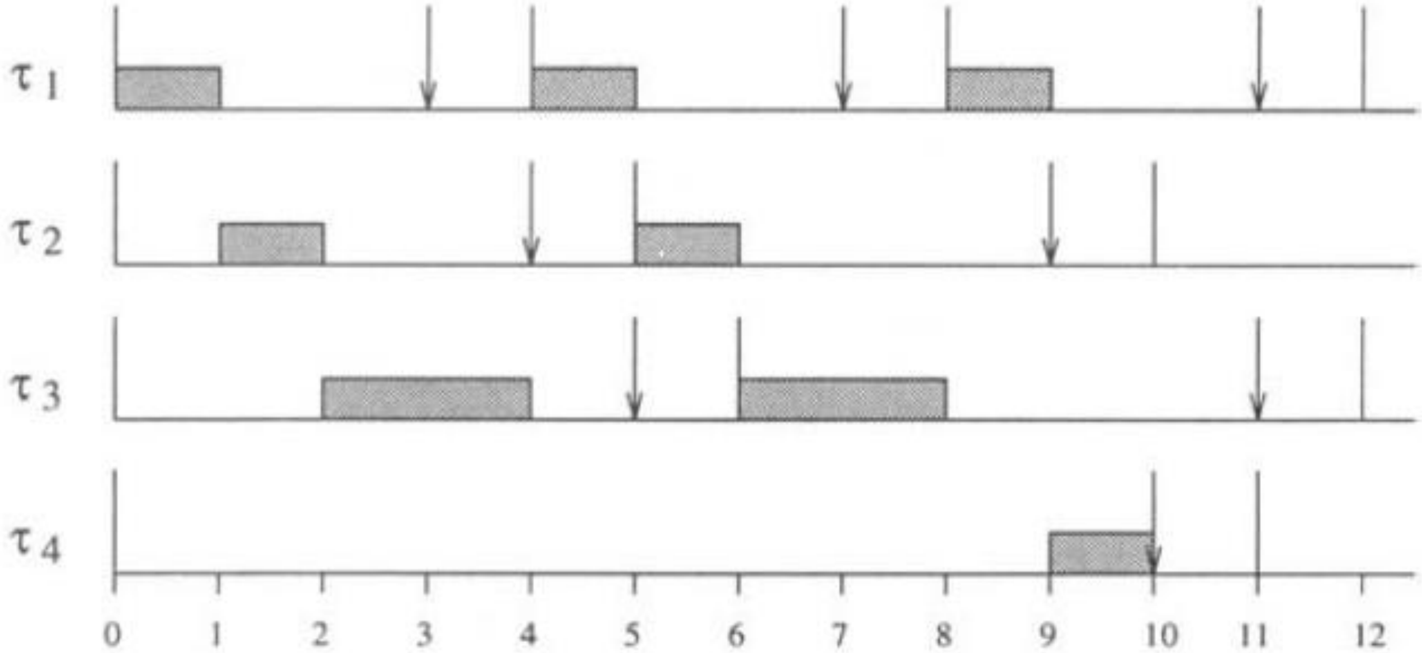
Step 1: $R_4^{(1)} = I_4^{(0)} + C_4 = 6,$ but $I_4^{(1)} = 6$ and $I_4^{(1)} + C_4 > R_4^{(1)}$
hence τ_4 does not finish at $R_4^{(1)}$.

Step 2: $R_4^{(2)} = I_4^{(1)} + C_4 = 7,$ but $I_4^{(2)} = 8$ and $I_4^{(2)} + C_4 > R_4^{(2)}$
hence τ_4 does not finish at $R_4^{(2)}$.

Step 3: $R_4^{(3)} = I_4^{(2)} + C_4 = 9,$ but $I_4^{(3)} = 9$ and $I_4^{(3)} + C_4 > R_4^{(3)}$
hence τ_4 does not finish at $R_4^{(3)}$.

Step 4: $R_4^{(4)} = I_4^{(3)} + C_4 = 10,$ but $I_4^{(4)} = 9$ and $I_4^{(4)} + C_4 = R_4^{(4)}$
hence τ_4 finishes at $R_4 = R_4^{(4)} = 10.$

Example 1



Example 2

- Analyze the schedulability of task T3

Task	T	C	D
1	250	5	10
2	10	2	10
3	330	25	50

Example 2

- Analyze the schedulability of task T3

Task	T	C	D
1	250	5	10
2	10	2	10
3	330	25	50

Iteration	R^s (for Task T3)	I	R^{s+1}
1	25	5+3x2=11	36
2	36	5+4x2=13	38
3	38	5+4x2=13	38

→T3 is schedulable

Algorithm 5: EDF with D < T

- Dynamic scheduling
- Utilization bound does not work!!!

→The processor demand approach

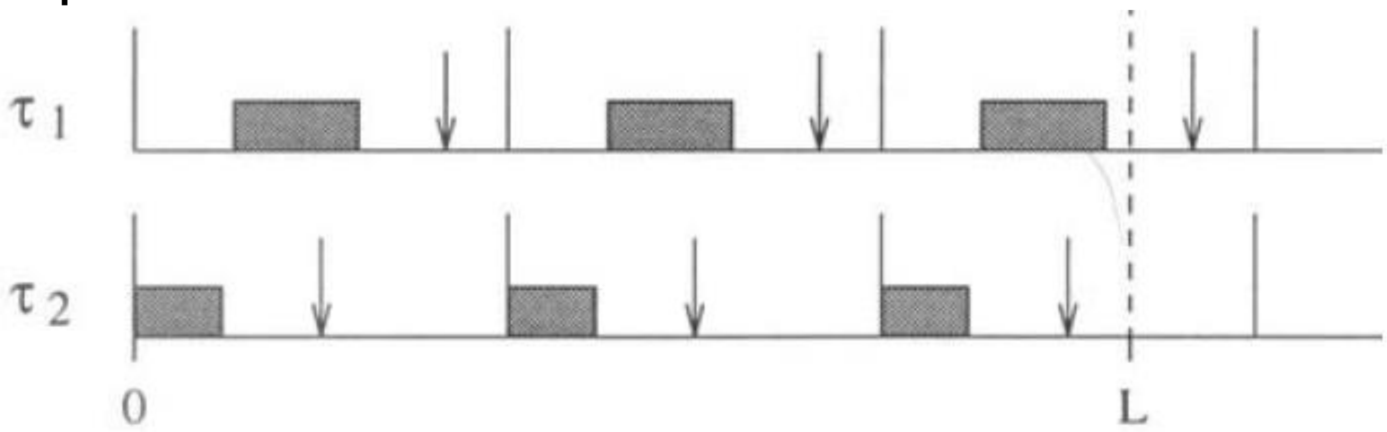
"During any time interval, the total processor demand of the whole tasks set must be no greater than the available time"

Processor demand

- Given time interval [0,L], total processor demand for task τ_i is

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

- Example



$$C_1(0,L) = \left\lfloor \frac{L}{T_1} \right\rfloor C_1$$
$$C_2(0,L) = \left(\left\lfloor \frac{L}{T_2} \right\rfloor + 1 \right) C_2$$

Processor demand

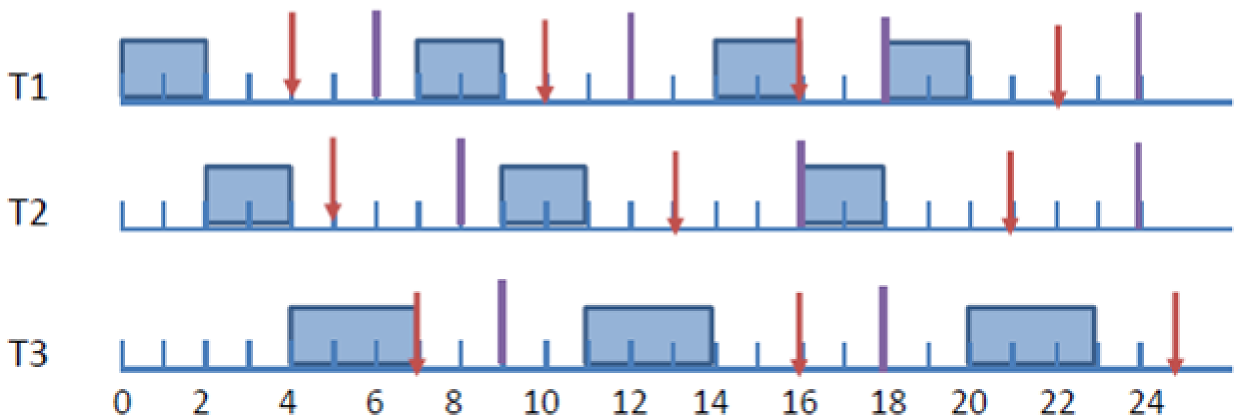
- Total processor demand for the whole task set

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

- Example

	C_i	D_i	T_i
T1	2	4	6
T2	2	5	8
T3	3	7	9

L	C(0,L)
4	2
5	4
7	7



Schedulability analysis

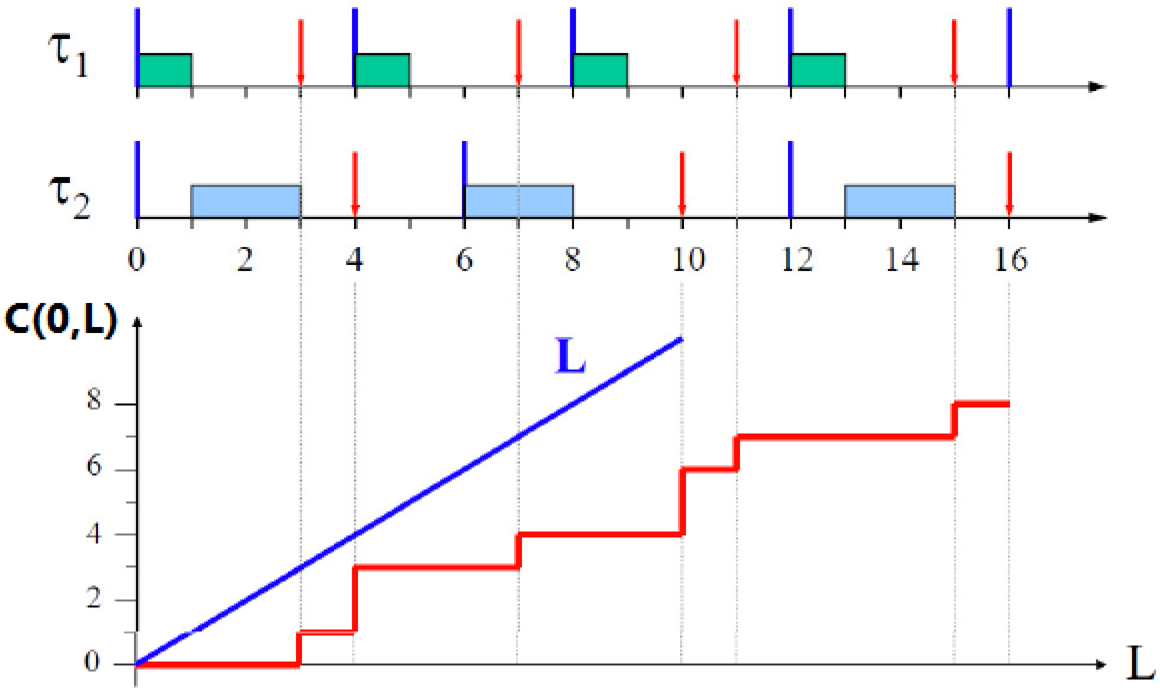
- Condition on processor demand
- For all L > 0 task set is schedulable by EDF if and only if

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

- Problem: how to check this condition?
  - Too many value of L

Schedulability analysis: Check at deadlines

- C(0, L) is a step function so we can check the schedulability condition on deadlines
- The number of values to check is still large



Schedulability analysis: Bounding L

- Observe

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

- Let

$$G(0,L)=\sum_{i=1}^{n}\frac{L+T_i-D_i}{T_i}\times C_i$$

- We have

$$C(0,L)\leq G(0,L)$$

Schedulability analysis: Bounding L

- Rewrite

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

$$= \sum_{i=1}^{n} L \frac{C_i}{T_i} + \sum_{i=1}^{n} (T_i - D_i) \frac{C_i}{T_i}$$

$$= LU + \sum_{i=1}^{n} (T_i - D_i) U_i$$

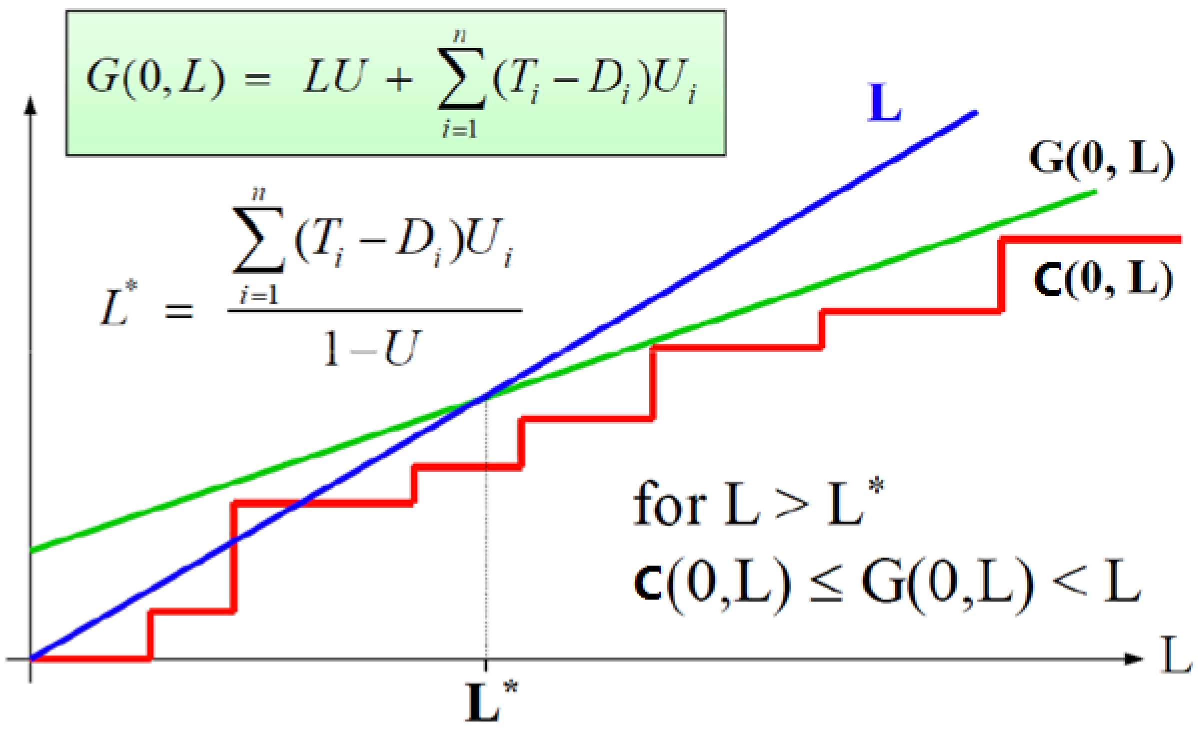
- then

$$\begin{cases} C(0,L) \le G(0,L) \\ C(0,L) \le L \end{cases}$$

Schedulability analysis: Bounding L

$G(0, L)$ is a straight line with slope U

L represents the line with slope 1. When $U < 1$, there exists $L = L^*$, where $G(0, L) = L$



L*: bounding value of L to check for schedulability

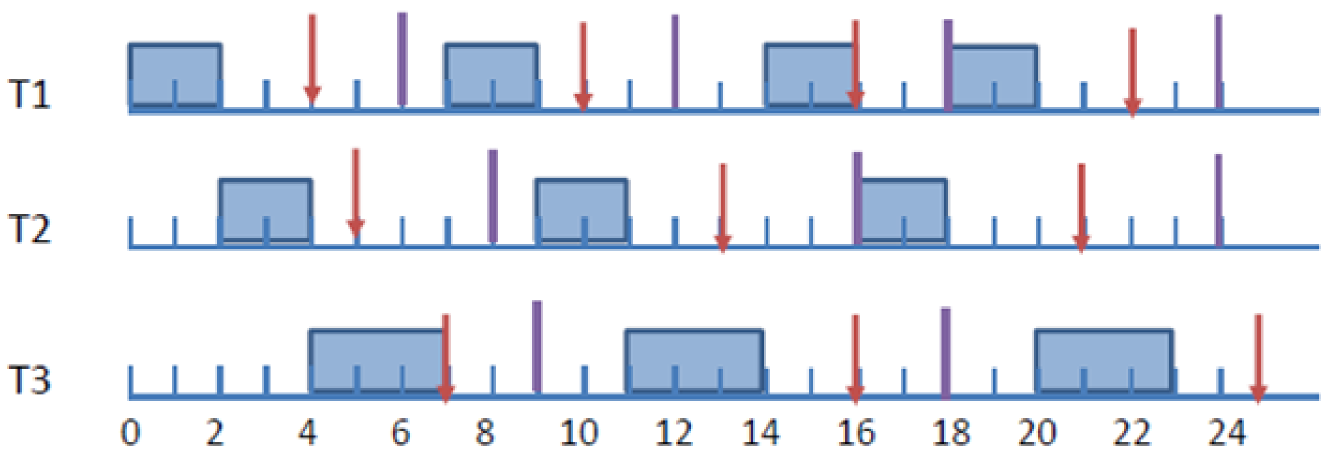
Example

- Calculate L* and verify schedulability

$$L^* = \frac{\sum_{i=1}^{n}(T_i - D_i)U_i}{1-U}$$

	C_i	D_i	T_i
T1	2	4	6
T2	2	5	8
T3	3	7	9

L	C(0,L)
4	2
5	4
7	7



Example

	C_i	D_i	T_i
T1	2	4	6
T2	2	5	8
T3	3	7	9

U = 2/6 + 2/8 + 3/9

$$L^* = \frac{\sum_{i=1}^{n} (T_i - D_i) U_i}{1 - U}$$

$L^* = 25$

L	C(0,L)	
4	2	OK
5	4	OK
7	7	OK
10	9	OK
13	11	OK
16	16	OK
21	18	OK
22	20	OK

Exercise

- Construct the schedule for this task set using RM and EDF

	C_i	T_i
τ_1	1	4
τ_2	2	6
τ_3	3	8

- Verify the schedulability and construct the schedule for the following task set using DM and EDF

	C_i	D_i	T_i
τ_1	2	5	6
τ_2	2	4	8
τ_3	4	8	12