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

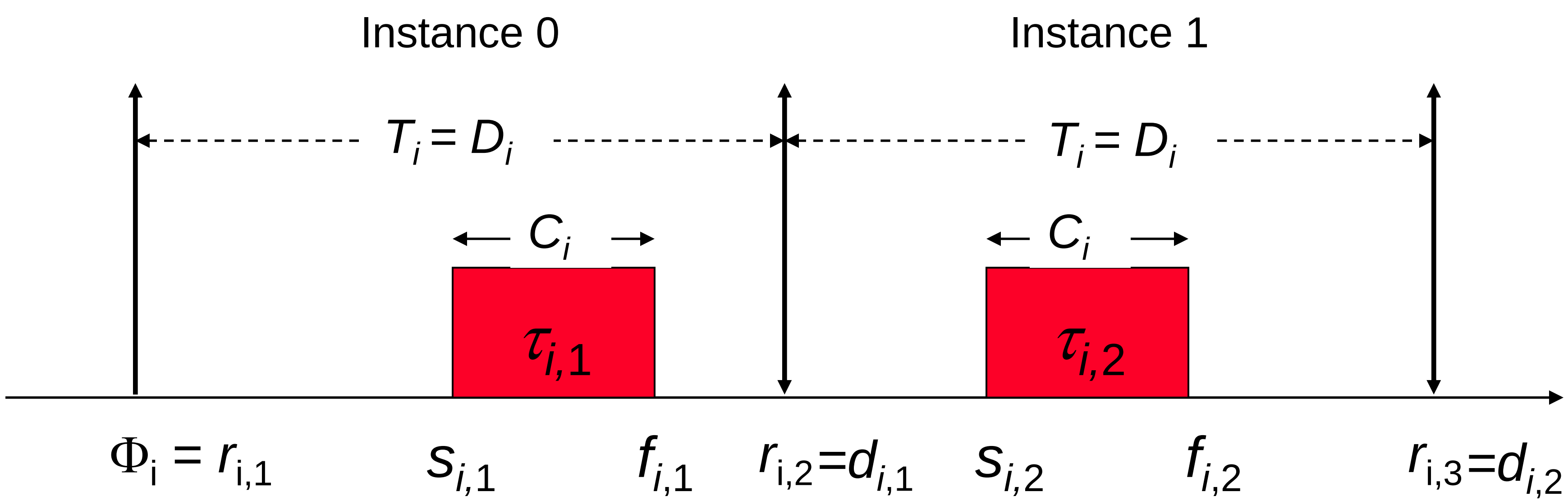
- 3 tasks:
  - Task 1: period 200 ms, computation time 50 ms
  - Task 2: period 100 ms, computation time 50 ms
  - Task 3: period 400 ms, computation time 50 ms
  - Is it schedulable?
- If task 4 is added
  - Task 4: period 200 ms, computation time 30 ms
  - Is it schedulable?

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

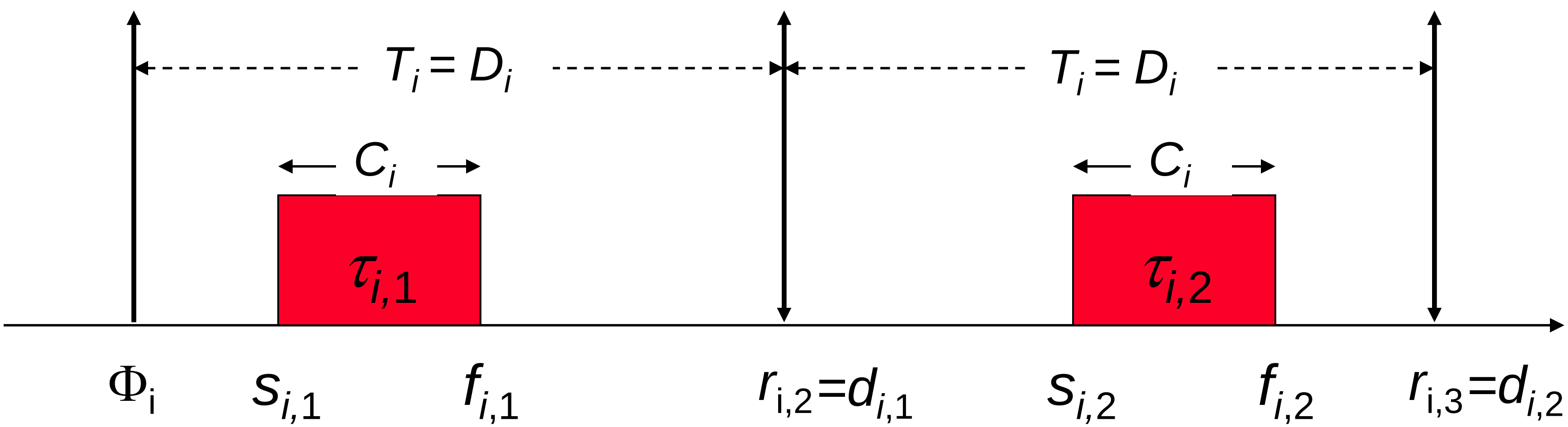
- A set Γ of periodic tasks is ***schedulable*** if all tasks in Γ are feasible

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: $\Gamma = \{\tau_i(\Phi_i, T_i, C_i),\ i=1,\dots,n\}$
  - Release time: $r_{i,k} = \Phi_i + (k-1)T_i$
  - Absolute deadline: $d_{i,k} = \Phi_i + kT_i$

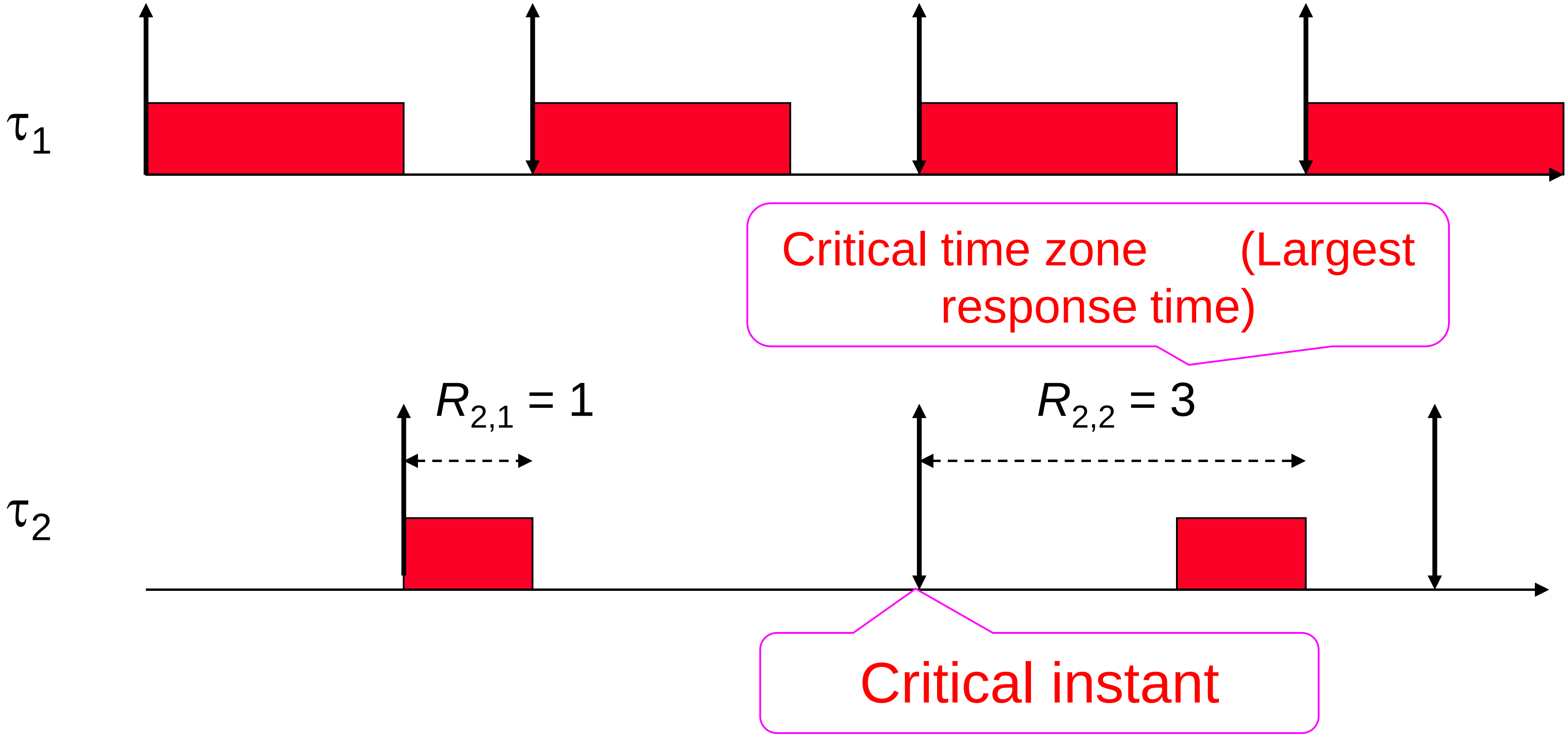


Periodic task parameters

- 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?

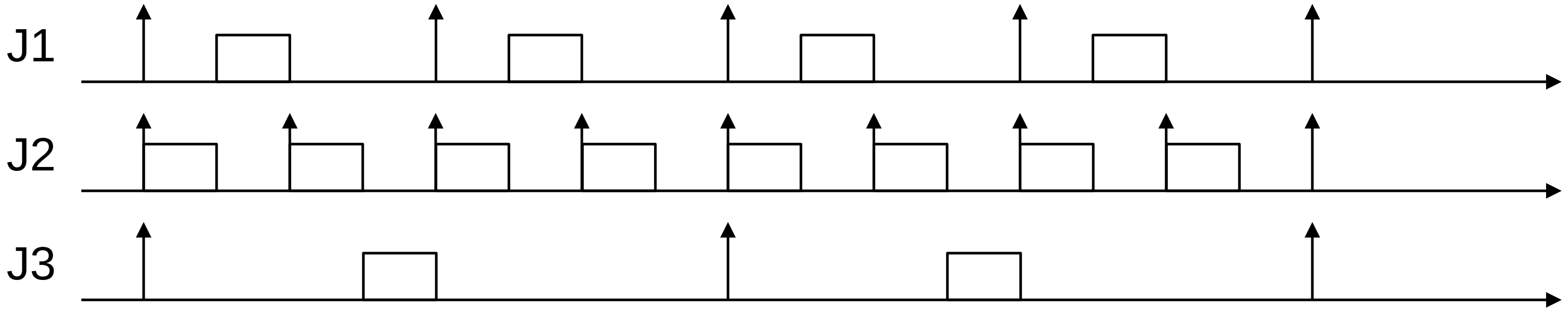


Hyperperiod

- 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?

$$H = lcm(T_1, . . . , T_n)$$

lcm: least common multiply

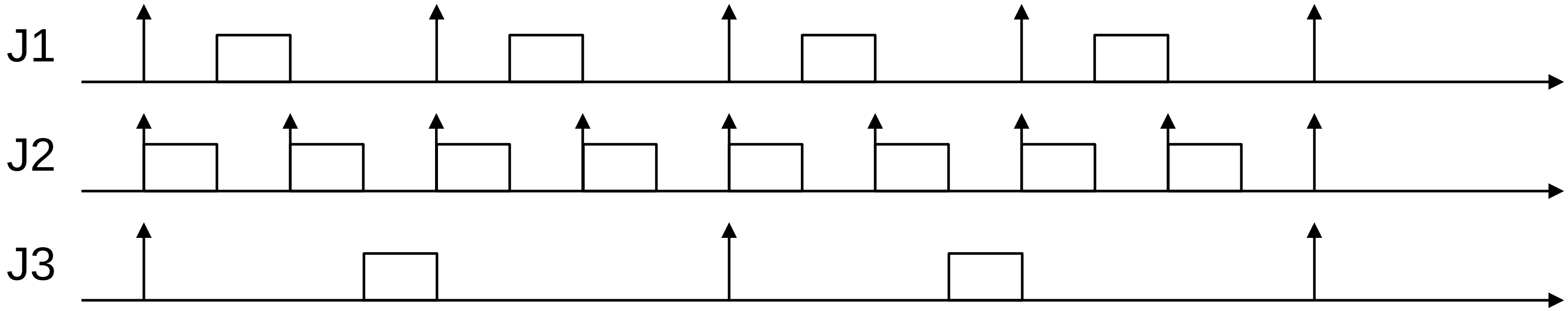


Hyper period

- 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?

$$H = lcm(T_1, . . . , T_n)$$

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

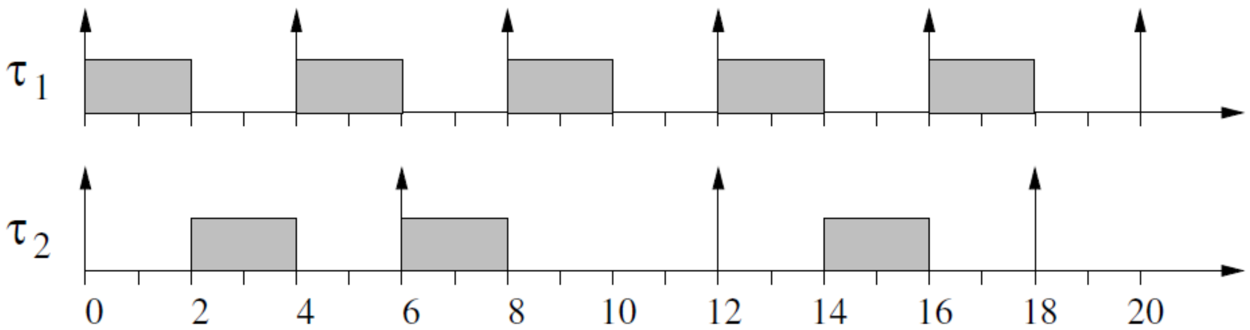
  ➔the task set is not schedulable

- 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*

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

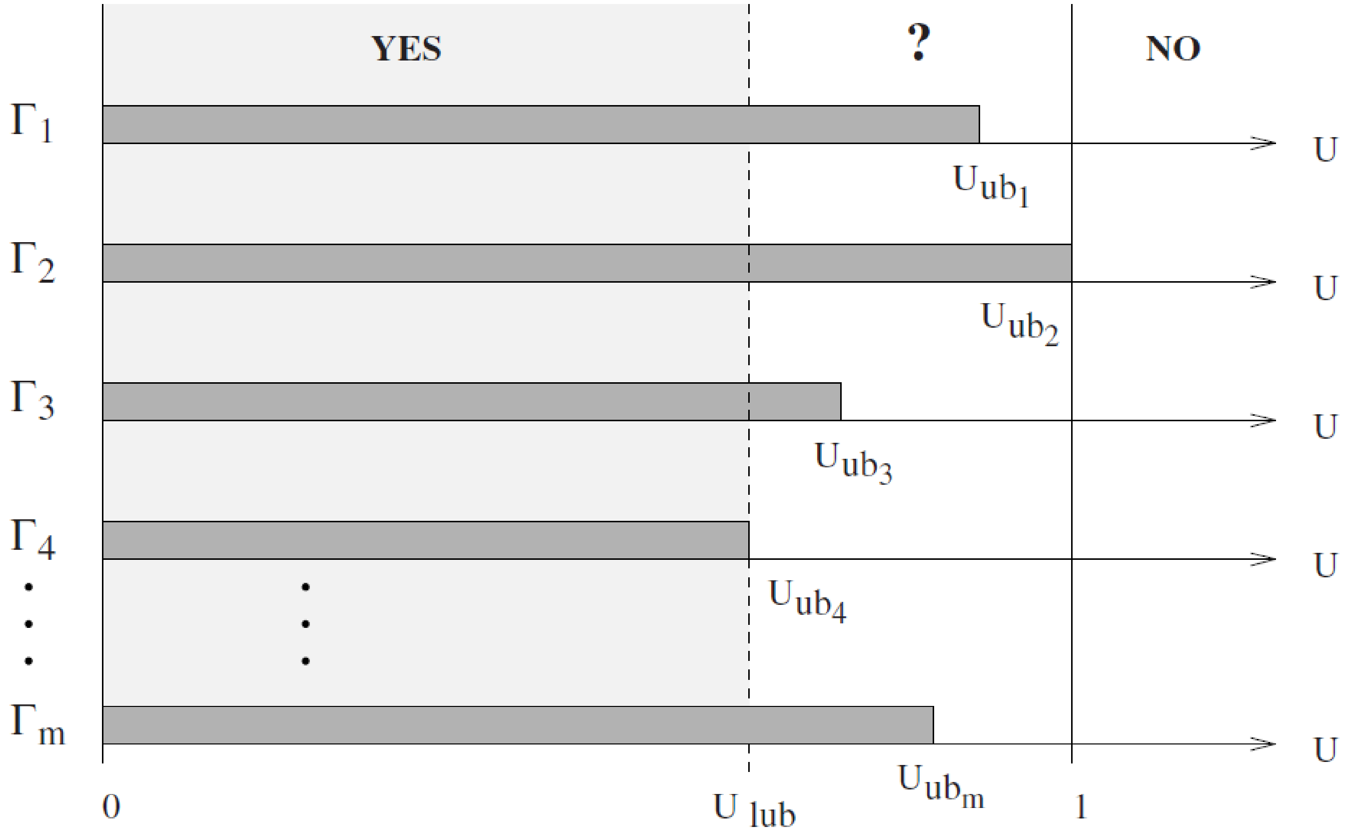
  - $U > U_{ub}(\Gamma, A)$: Γ is not schedulable by A
  - $if\ U = U_{ub}(\Gamma, A)$: Γ fully utilizes the processor

- 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



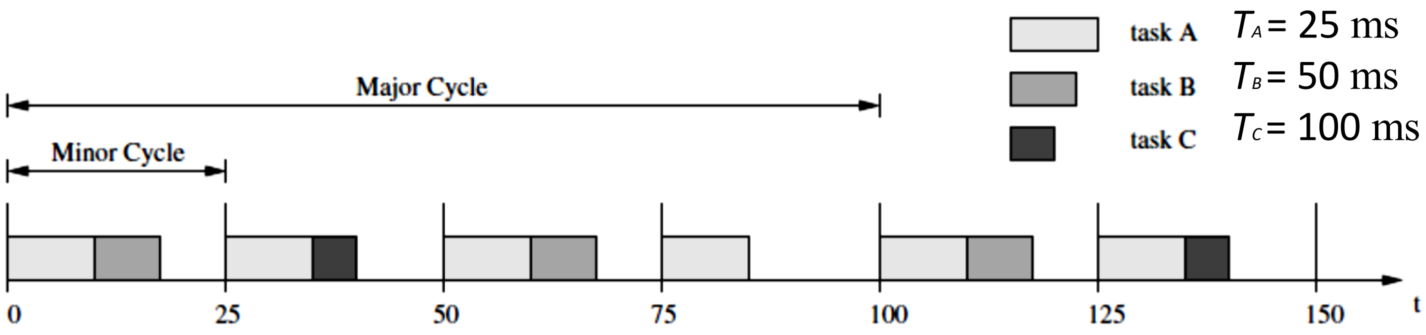
Utilization vs schedulability

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



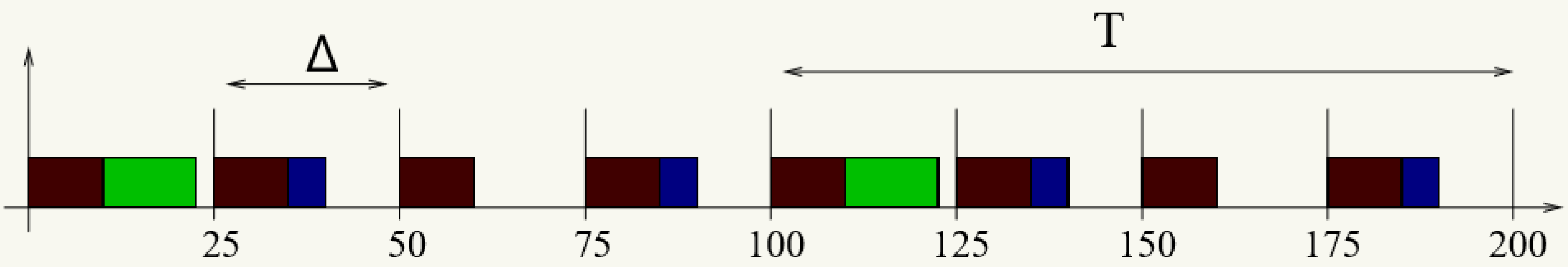
Example

- 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 $25ms$

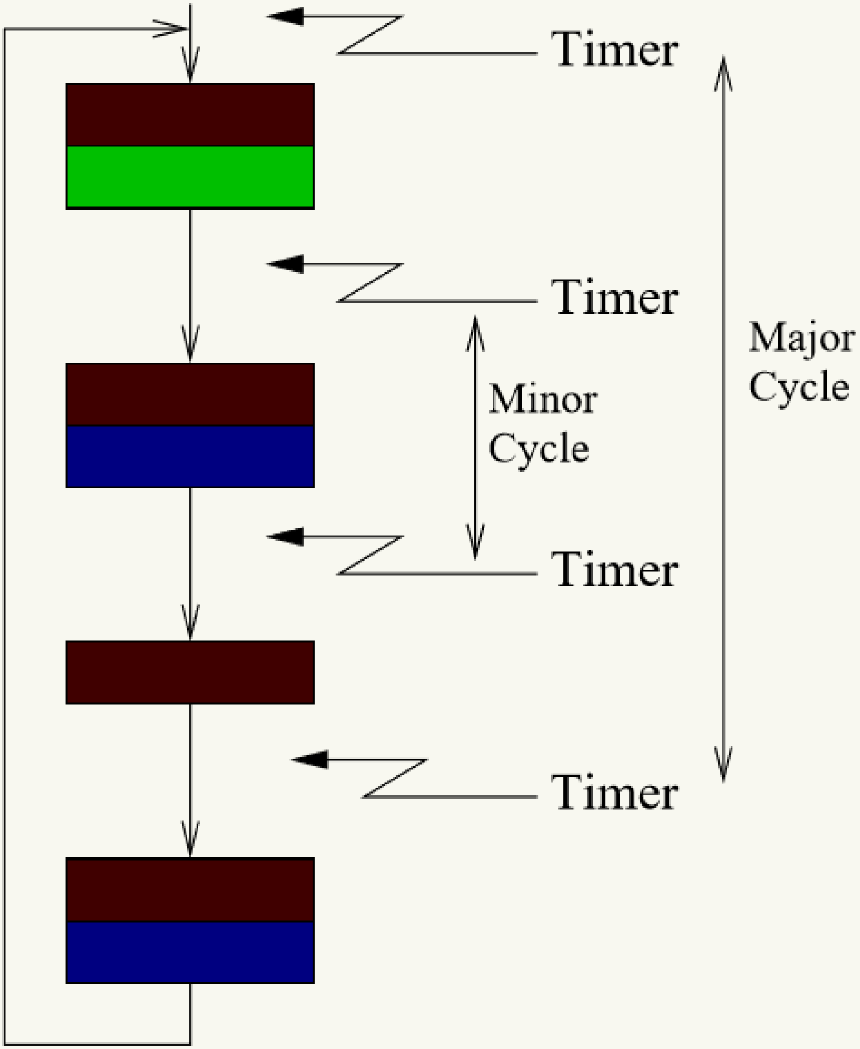
Example

- 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

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

- Then we have

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

$$\rightarrow U > 1$$

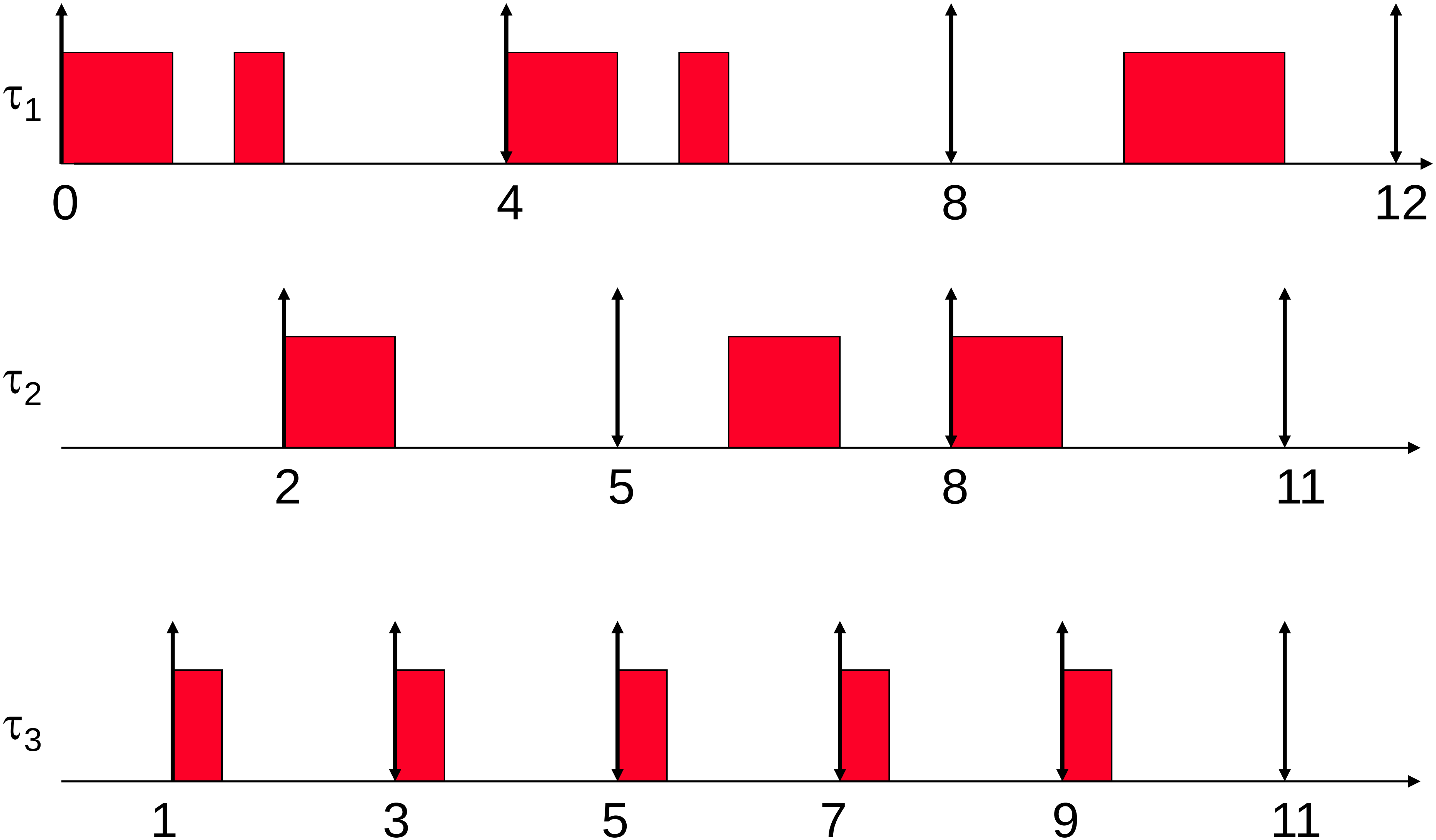
→ *contradiction*

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

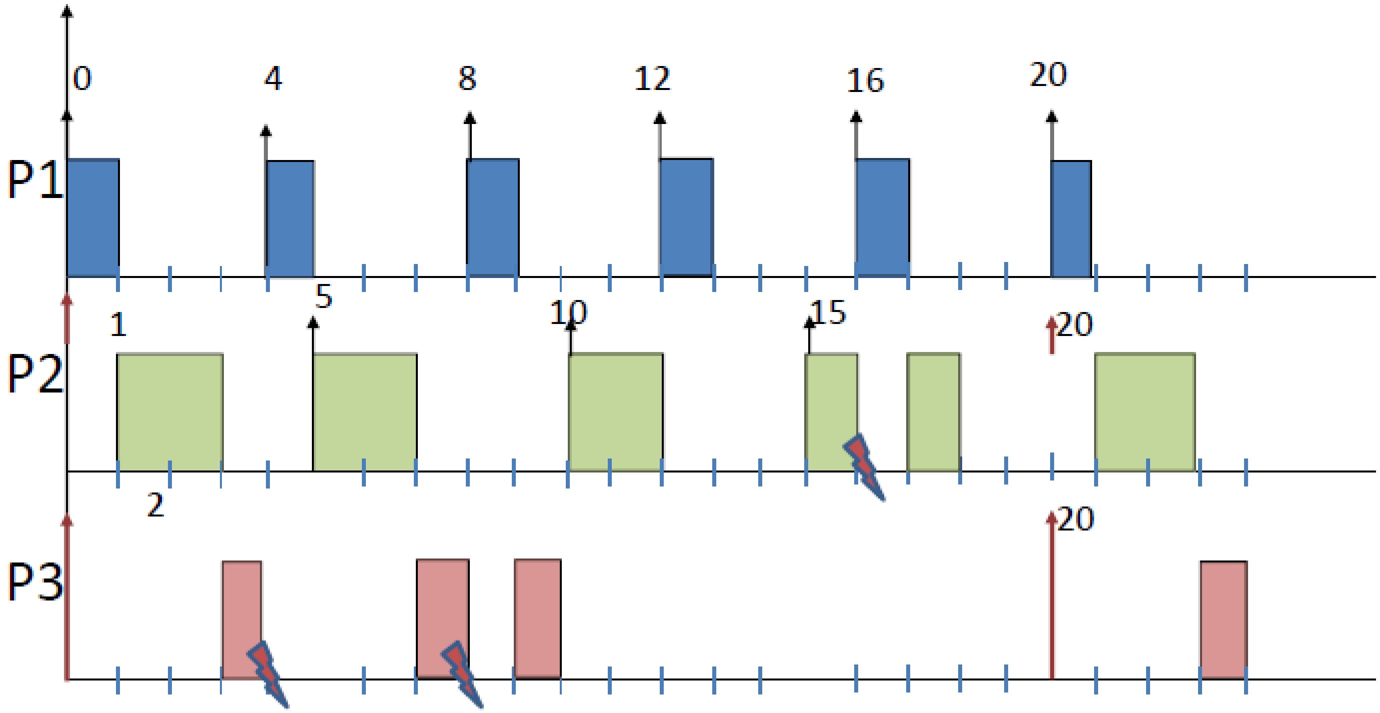


Algorithm 3: Rate Monotonic (RM)

- 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}

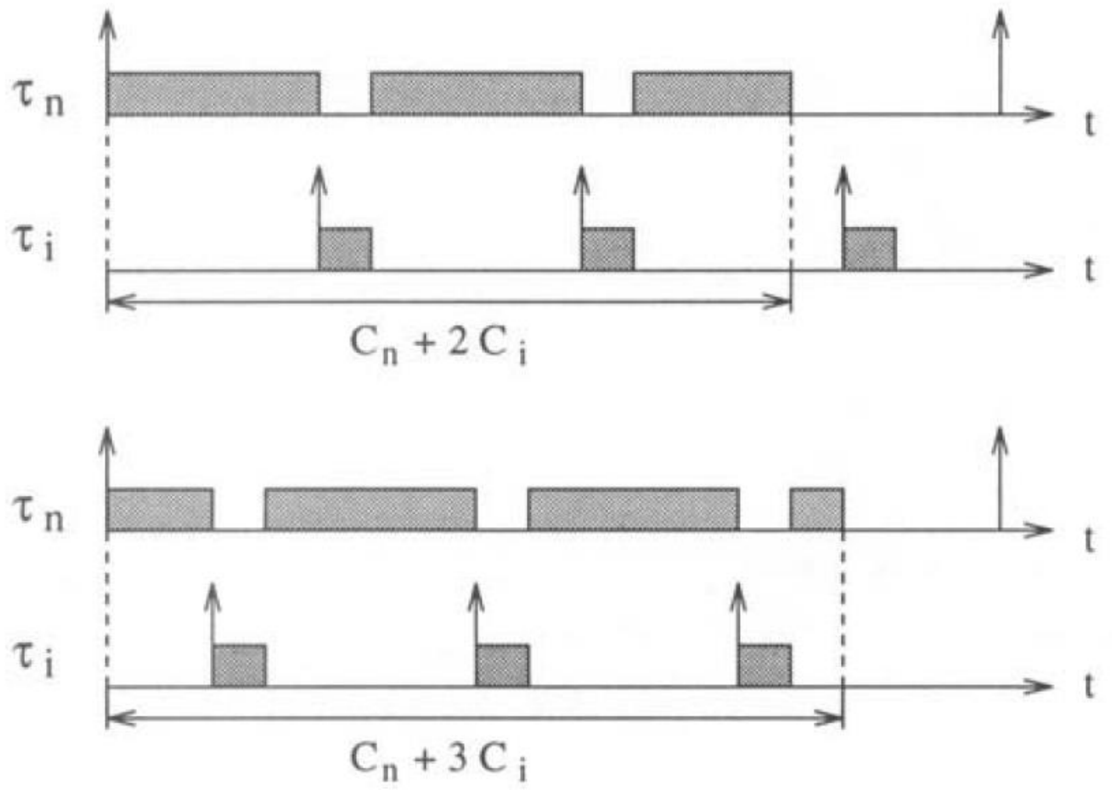
Example

- 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

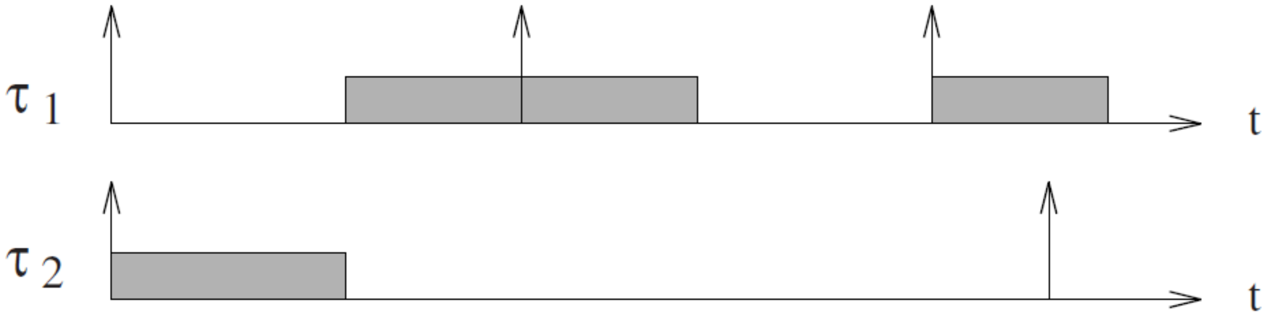
➔ Task schedulability can be checked at its critical instance

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

$$C_1 + C_2 \leq T_1$$

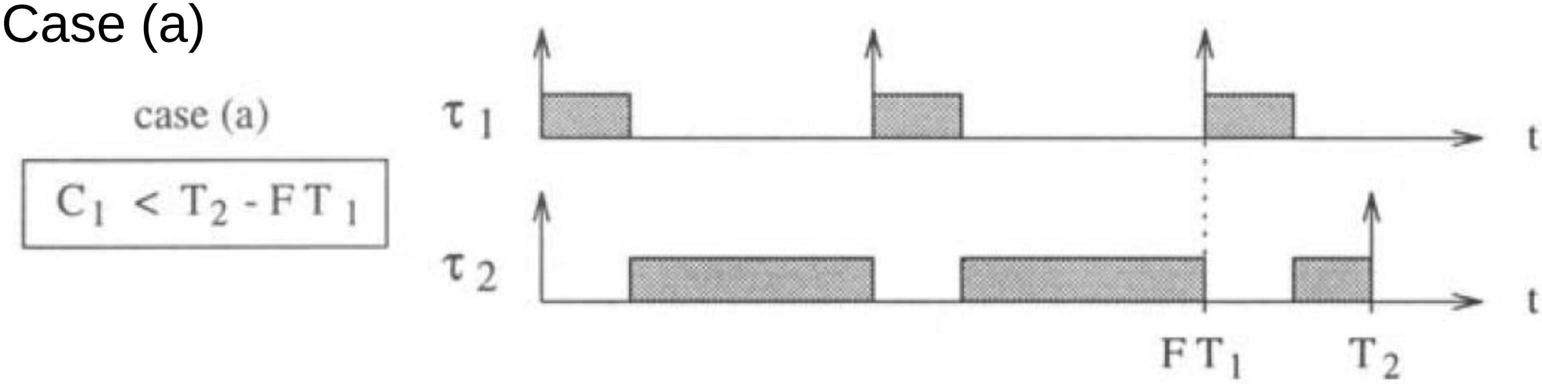


- Show that exchanging priority of T1 and T2 will result in feasible schedule

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

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

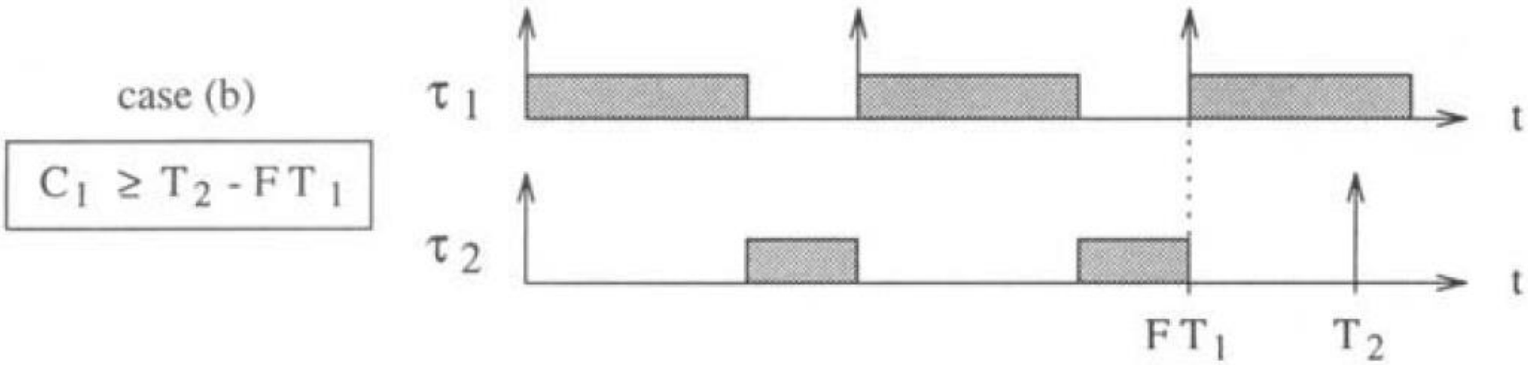
$$(F+1)C_1 + C_2 \leq FT_1 + C_1$$

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

➔ The schedule by RM is feasible

Proof of optimality (4)

- Case (b)



As $$C_1 + C_2 \leq T_1 \quad \Rightarrow \quad FC_1 + FC_2 \leq FT_1$$

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

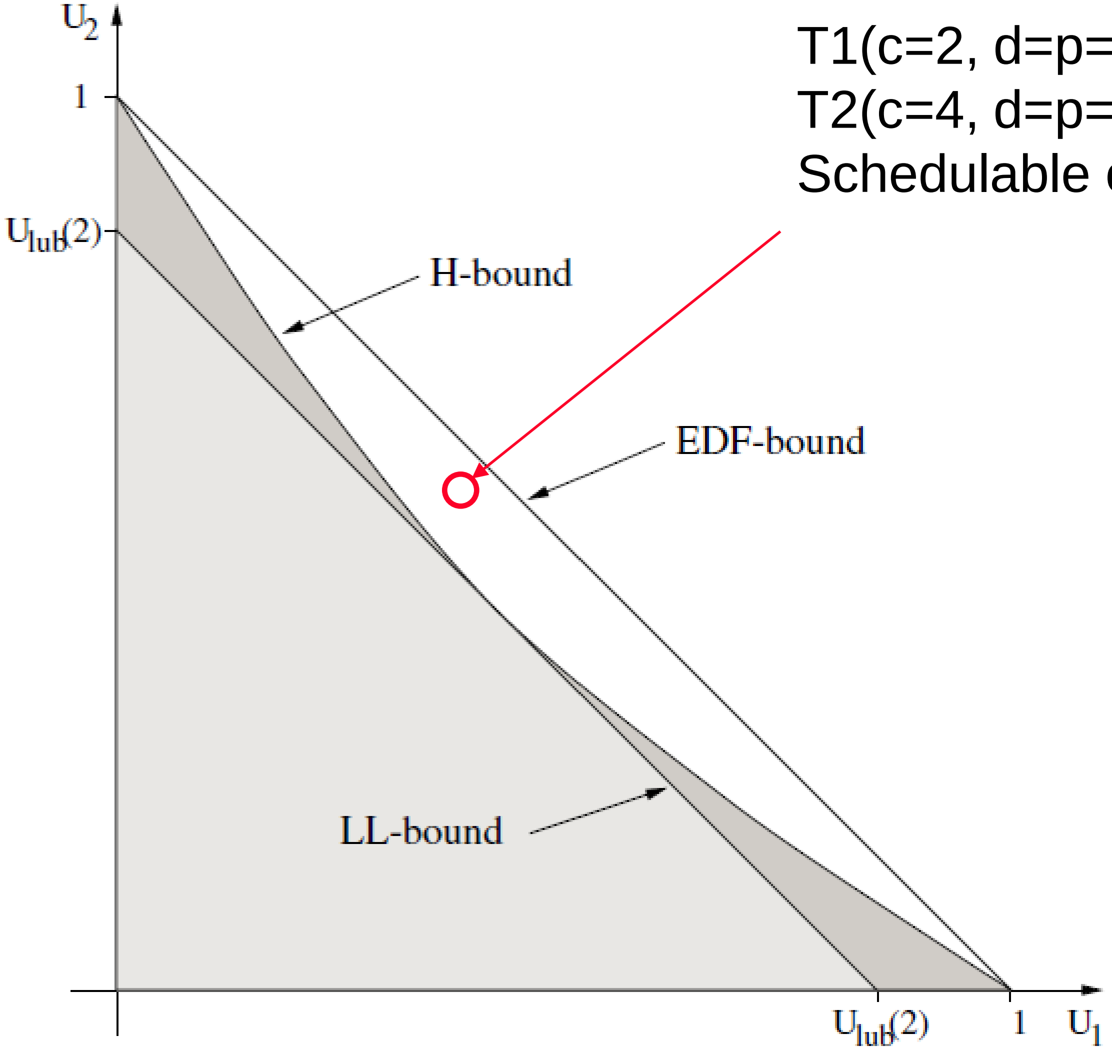
➔ Need to check manually

RM schedulability: using hyperbolic bound

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

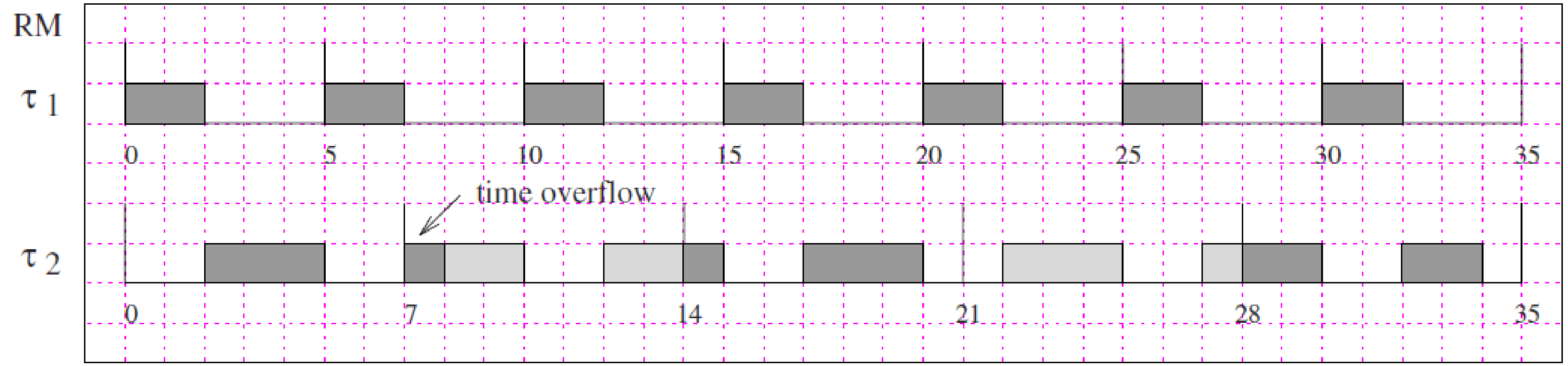
$$\prod_{i=1}^{n}(U_i + 1) \leq 2.$$

EDF vs RM

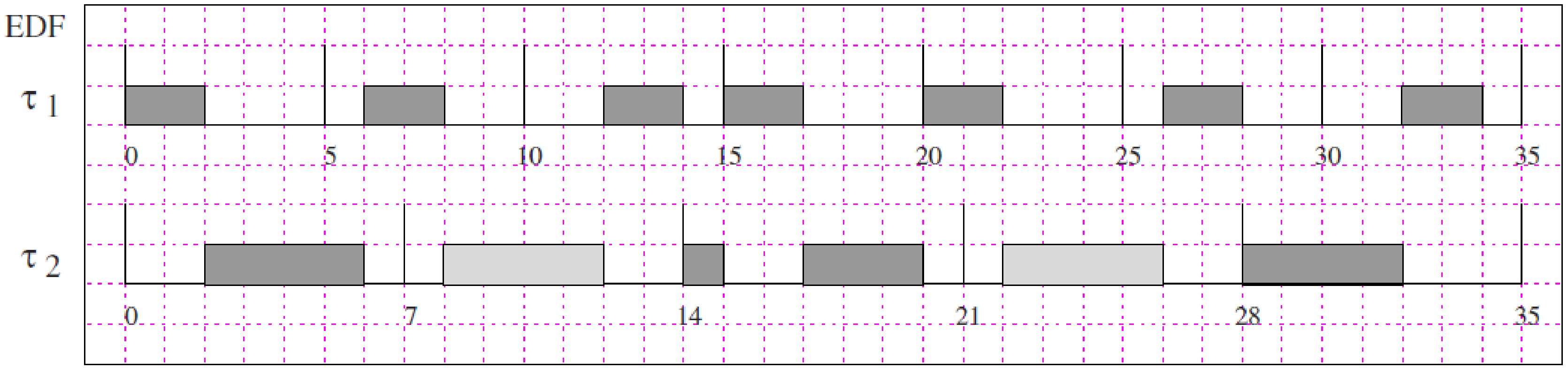


T1(c=2, d=p=5)
T2(c=4, d=p=7)
Schedulable or not?

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

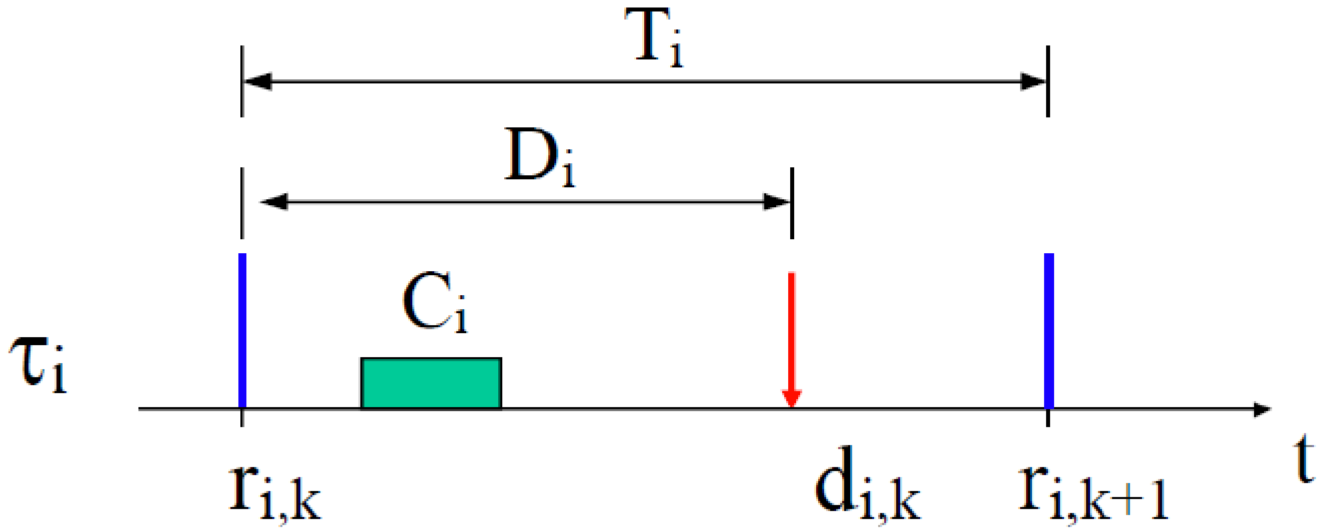
Assumptions for EDF and RM

- A3: Relative deadline equals to period

$$D_i = T_i$$

- Relax the assumption for more practical problems

$$D_i < T_i$$

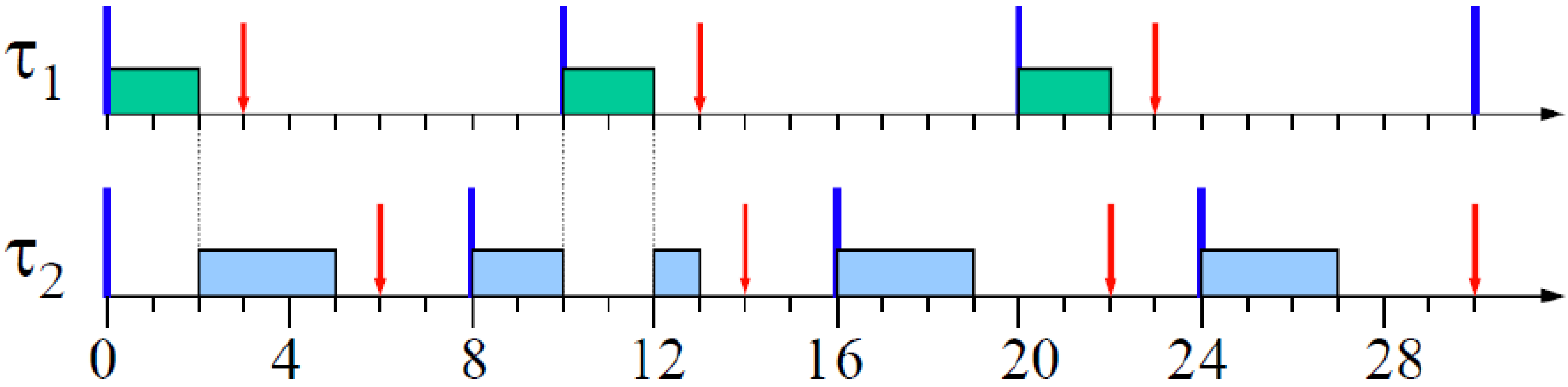


→ modified algorithms

Algorithm 4: Deadline Monotonic (DM)

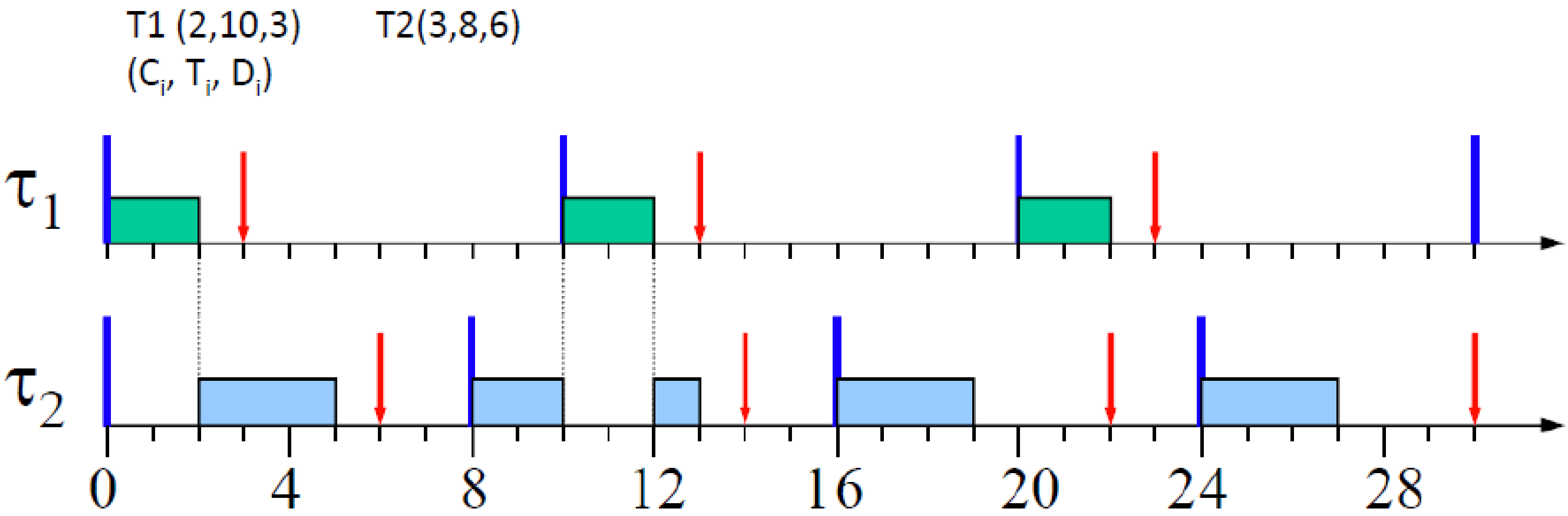
- Each task is assigned a priority inversely proportional to its relative deadline
- Shorter deadlines imply higher priorities

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



➔ Feasible schedule

Schedulability analysis of DM



- Processor utilization

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

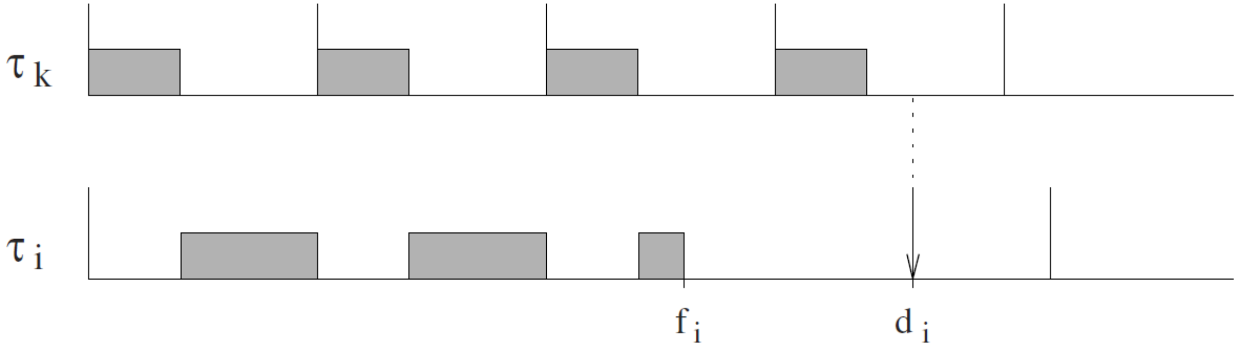
➔cannot be used for schedulability analysis

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

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



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 $R_4^{(0)}$.

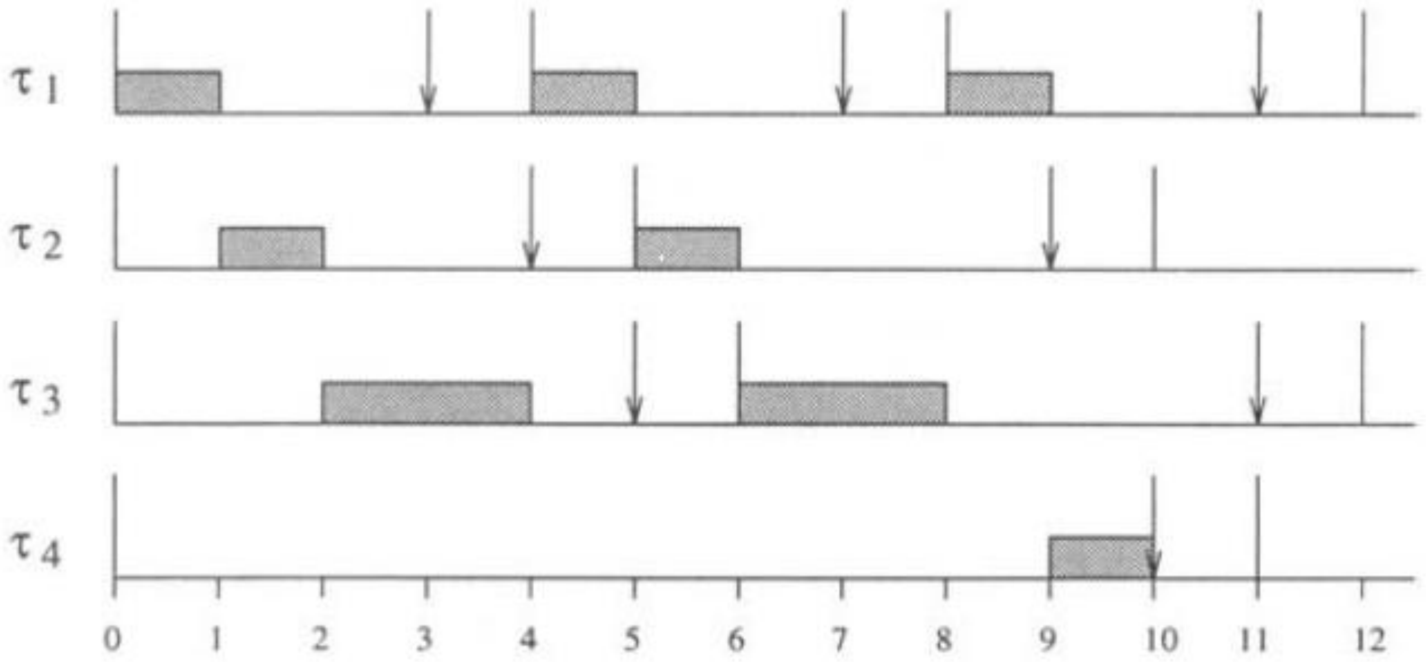
Step 1: $R_4^{(1)} = I_4^{(0)} + C_4 = 6,$ but $I_4^{(1)} = 6$ and $I_4^{(1)} + C_4 > R_4^{(1)}$
hence τ_4 does not finish at $R_4^{(1)}$.

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

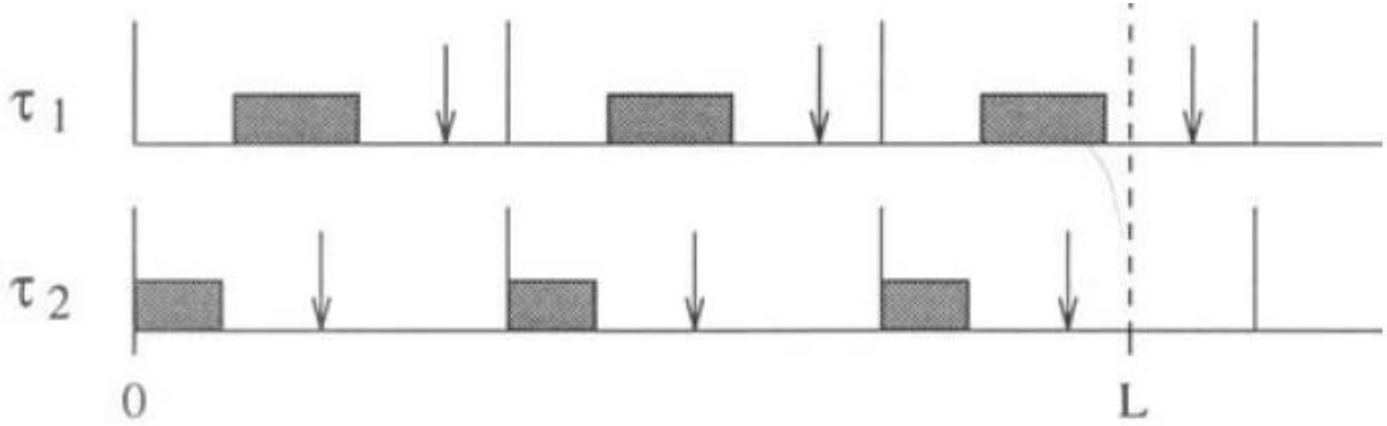
"During any time interval, the total processor demand of the whole tasks set must be no greater than the available time"

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



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

Processor demand

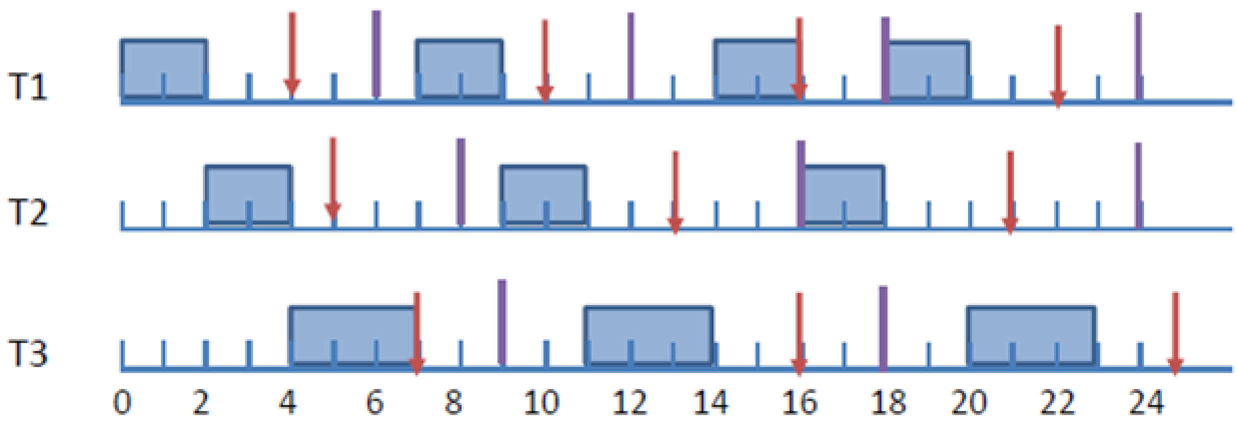
- Total processor demand for the whole task set

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

- 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

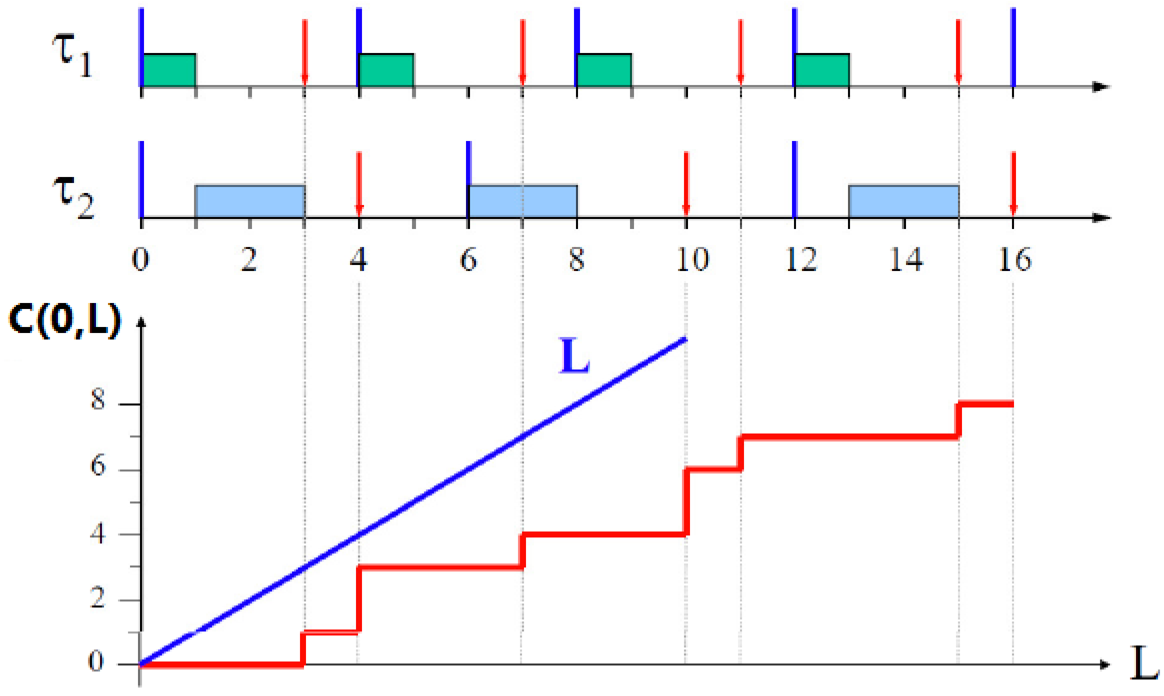
- Condition on processor demand
- For all L > 0 task set is schedulable by EDF if and only if

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

- 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

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

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

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

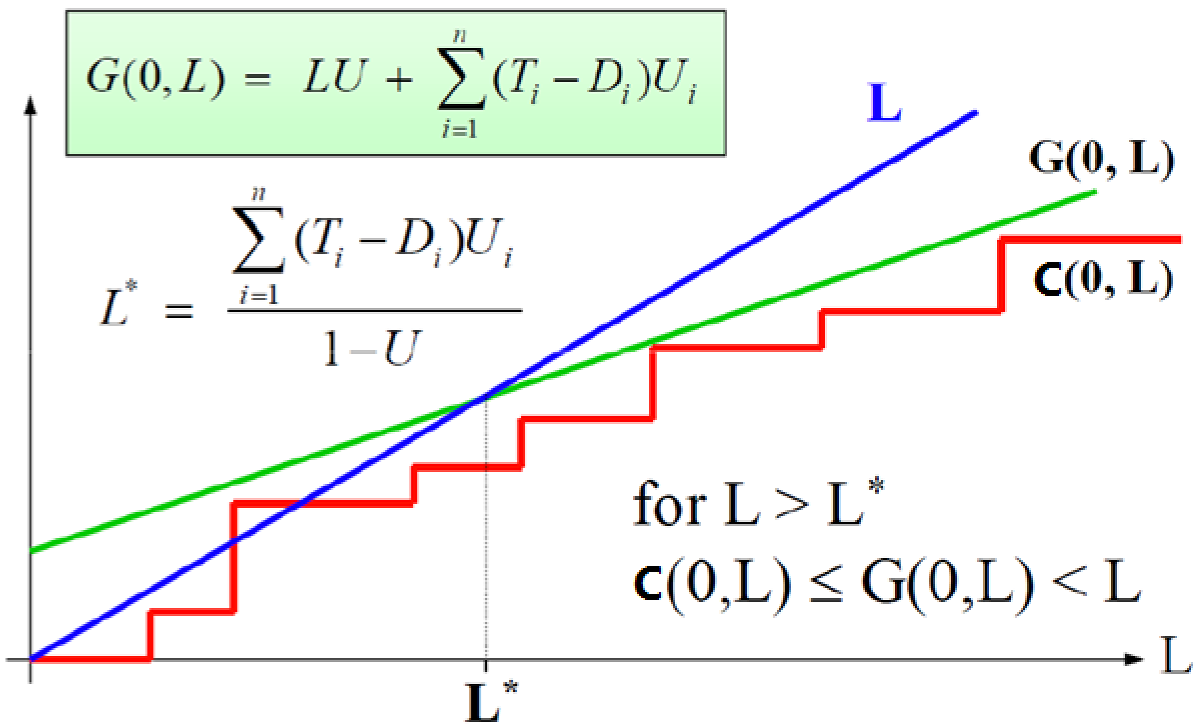
- then

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

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

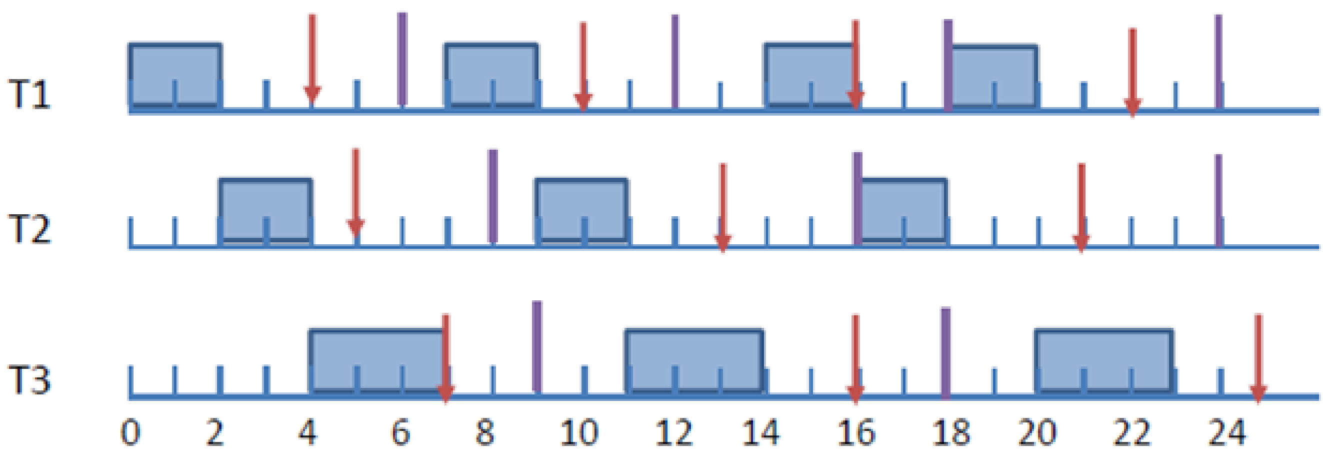
Example

- 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

Real-time Systems

Chapter 7:
Fixed Priority Servers

Ngo Lam Trung

Dept. of Computer Engineering

Contents

- Introduction
- Background scheduling
- The basic algorithm: Polling Server
- Improve response time: Deferrable Server
- Improve schedulability bound: Priority Exchange
- DS to equivalent periodic task: Sporadic Server

Introduction

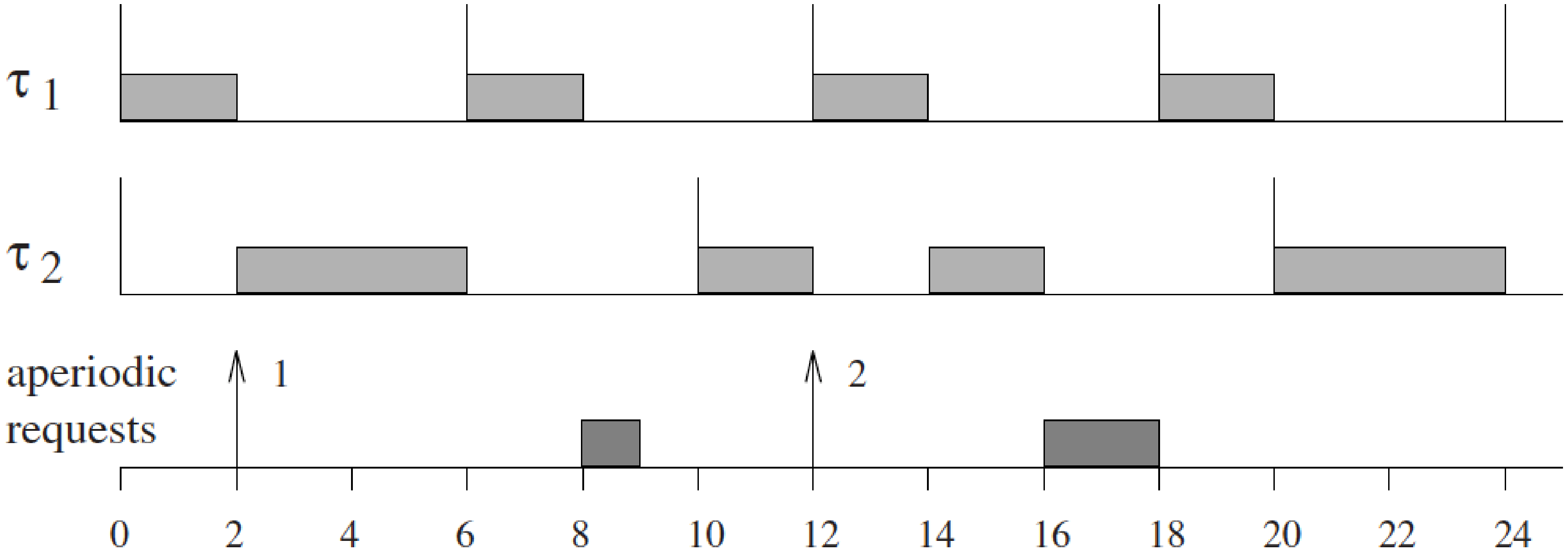
- Practical systems contain different types of task
  - Periodic tasks for critical activities: time driven, usually with hard timing constrain
  - Aperiodic tasks: event driven, may be hard/soft or non-real time.
  - ➔ Hybrid task set
- Problem: How to produce a schedule that
  - Guarantee the schedulability of critical (periodic) tasks
  - Provide acceptable response time of soft and non-real time tasks
- How about critical aperiodic tasks?
  - Assuming a maximum arrival rate → change to periodic task

Assumption

- All periodic task start at $t = 0$ and their deadline and period are equal.
- Periodic task are scheduled by RM (fixed priority).
- Arrival times of aperiodic requests are unknown.
- The minimum inter-arrival time of a sporadic task is assumed to be equal to its deadline.
- All tasks are fully preemptible

The simplest method: Background scheduling

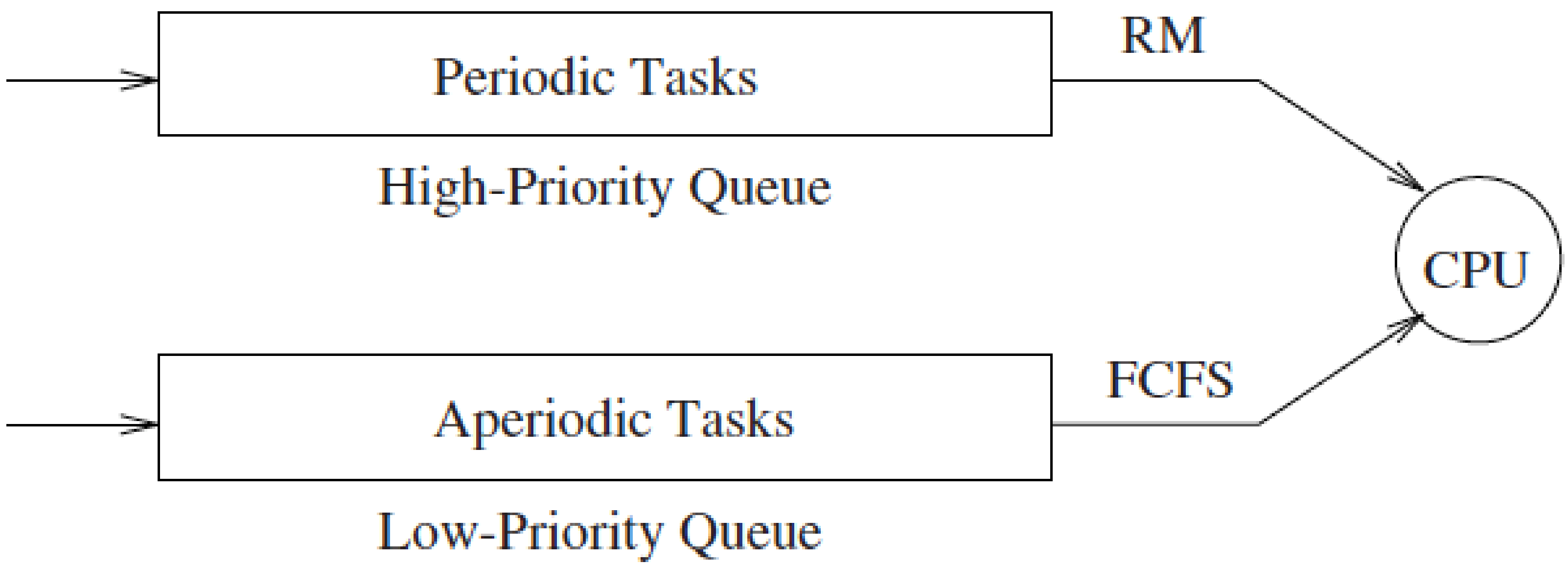
- Schedule periodic tasks with RM as usual
- Aperiodic tasks are scheduled at background: run when there is no periodic load.



$U_{periodic}$ = ?
Schedulability of periodic task will change of not?

Background scheduling

- Two task queues in background scheduling



- Pros: simple method
- Cons: response time of aperiodic tasks may be low in high periodic load

Polling Server

- Improve average aperiodic tasks response time
- Create an additional periodic task
  - Called Polling Server (PS)
  - PS serves aperiodic load asap upon request
- Server task parameter
  - Period T_S
  - Computation time C_S: server capacity
- PS is scheduled together with other periodic tasks
- When PS is activated
  - Select a waiting aperiodic task and execute it with server capacity
  - If there is no aperiodic task waiting, server suspends itself and gives up its capacity

Polling Server

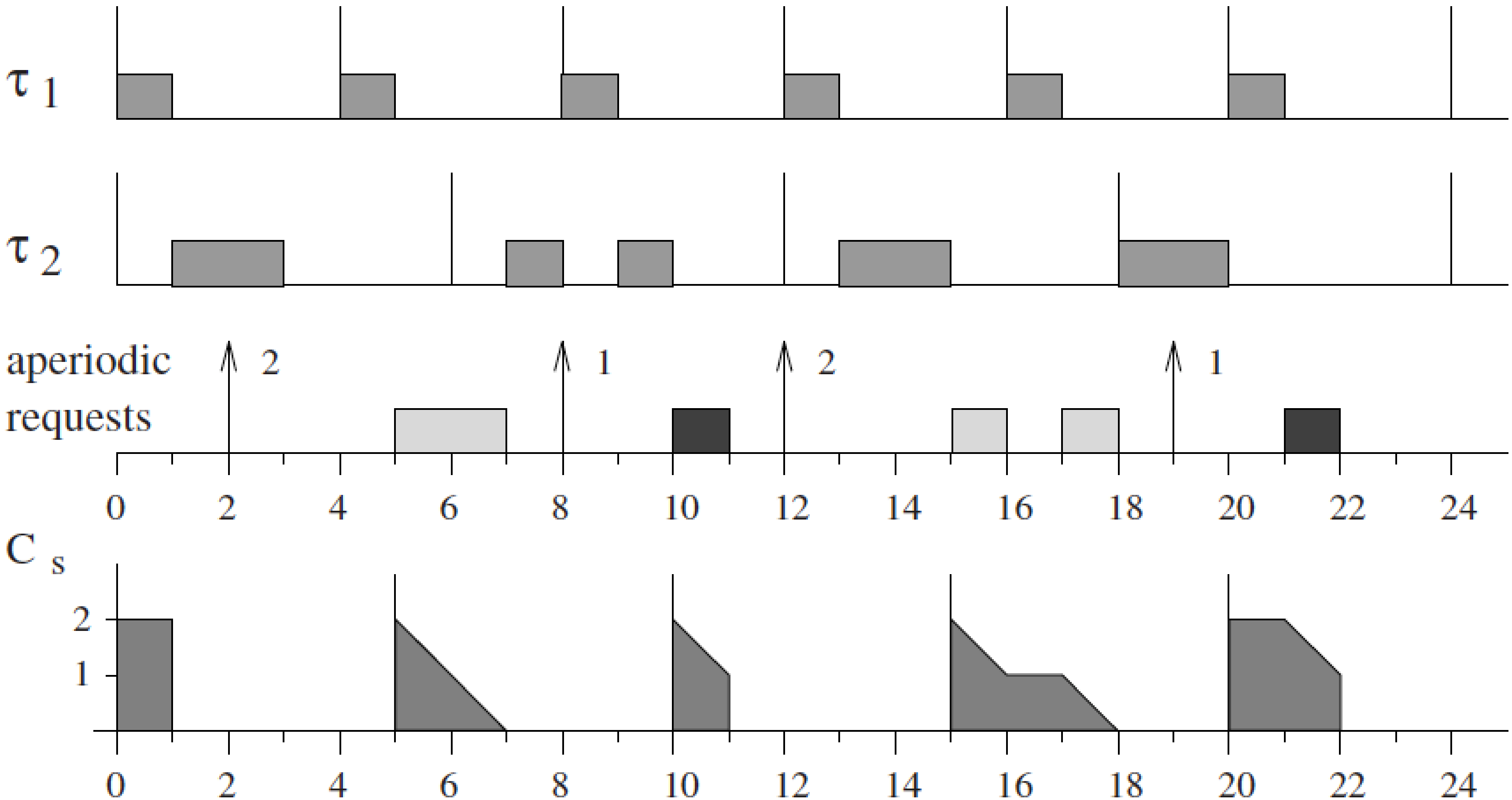
- Example

	C_i	T_i
τ_1	1	4
τ_2	2	6

Server

$C_s = 2$

$T_s = 5$



Polling Server: Schedulability analysis

- With RM, the task set including the server must be schedulable

$$U_p + Us \le Ulub(n+1).$$

- Or

$$U_p \le n\left[\left(\frac{2}{U_s+1}\right)^{1/n} - 1\right]$$

U_p = total CPU utilization of original periodic tasks

- Or

$$\prod_{i=1}^{n}(U_i+1) \le \frac{2}{U_s+1}$$

Dimensioning the PS

- What are appropriate values of Ts, Ps that guarantee feasible schedule?
- Define

$$P \stackrel{\text{def}}{=} \prod_{i=1}^{n} (U_i + 1)$$

- From schedulability condition

$$P \leq \frac{2}{U_s + 1}$$

- So we need the server satisfies

$$U_s \leq \frac{2 - P}{P}$$

Dimensioning the PS

- Let

$$U_s^{max} = \frac{2-P}{P}$$

- Server utilization can be selected so $U_s < Umax$
- Then select the smallest server period as possible

$$T_s = T_1$$

- Finally

$$C_s = U_s T_s$$

Polling Server

- Exercise 1

Consider two periodic tasks with computation times $C_1 = 1$, $C_2 = 2$ and periods $T_1 = 5$, $T_2 = 8$, handled by Rate Monotonic. Show the schedule produced by a Polling Server, having maximum utilization and intermediate priority, on the following aperiodic jobs:

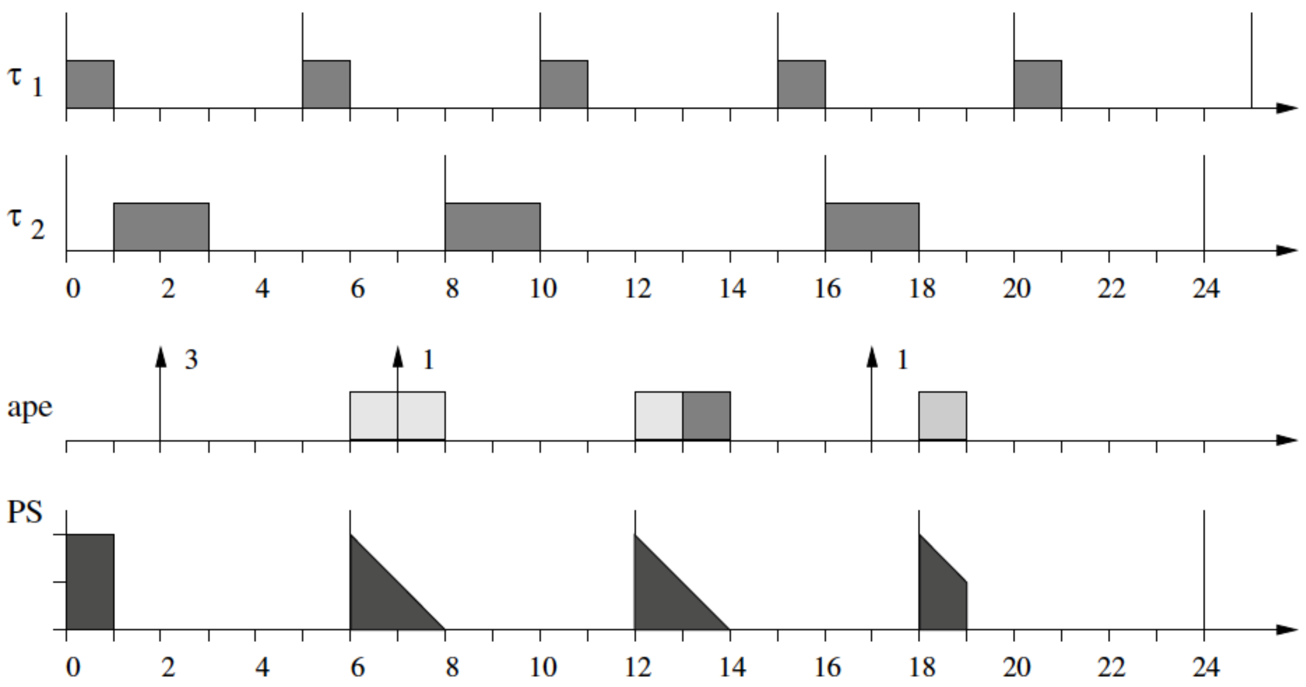
	a_i	C_i
J_1	2	3
J_2	7	1
J_3	17	1

Polling Server

- Sizing the PS

$$U_{PS}^{max} = \frac{2-P}{P} = \frac{1}{3}$$

- So we can set $T_s = 6$ (intermediate priority) and $C_s = 2$



Polling Server

- Problem: if an aperiodic request arrives after the server suspends itself, the request must wait until the next server period

  ➔ lowering average response time

- How to improve:
  - Server will not suspend
  - → Deferrable Server (DS)

Deferable Server (DS)

- Improves responsiveness of aperiodic tasks (compare to PS)
- Algorithm
  - Create high priority periodic task to serve aperiodic tasks
  - Server replenishes its capacity at the beginning of each period
  - If no aperiodic load are pending upon server invocation, the server preserves its capacity

    →aperiodic load can be served at anytime (as opposed to PS)

Example 1

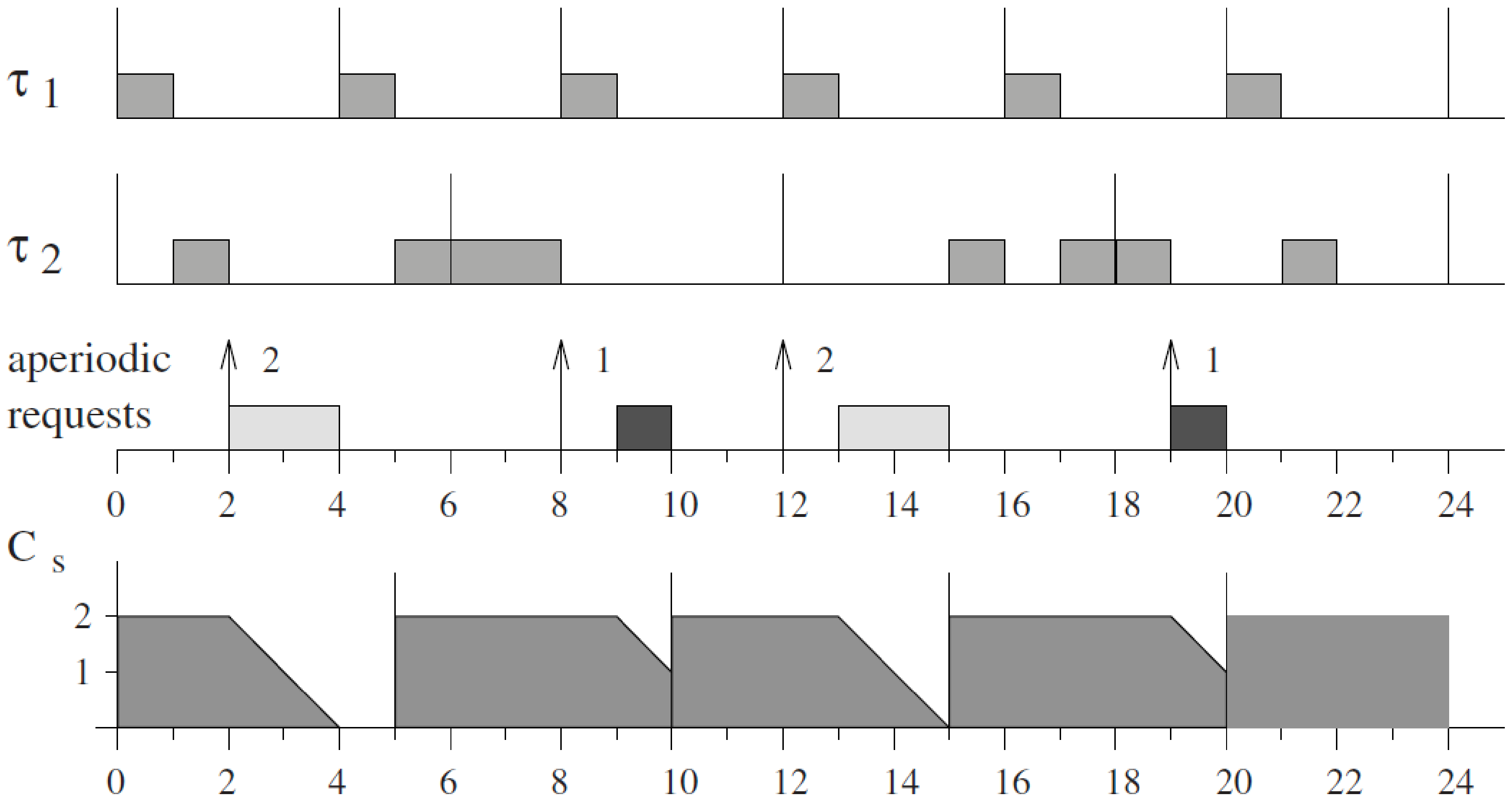
Periodic tasks' priorities are calculated by RM

	C_i	T_i
τ_1	1	4
τ_2	2	6

Server

$C_s = 2$

$T_s = 5$



Example 2

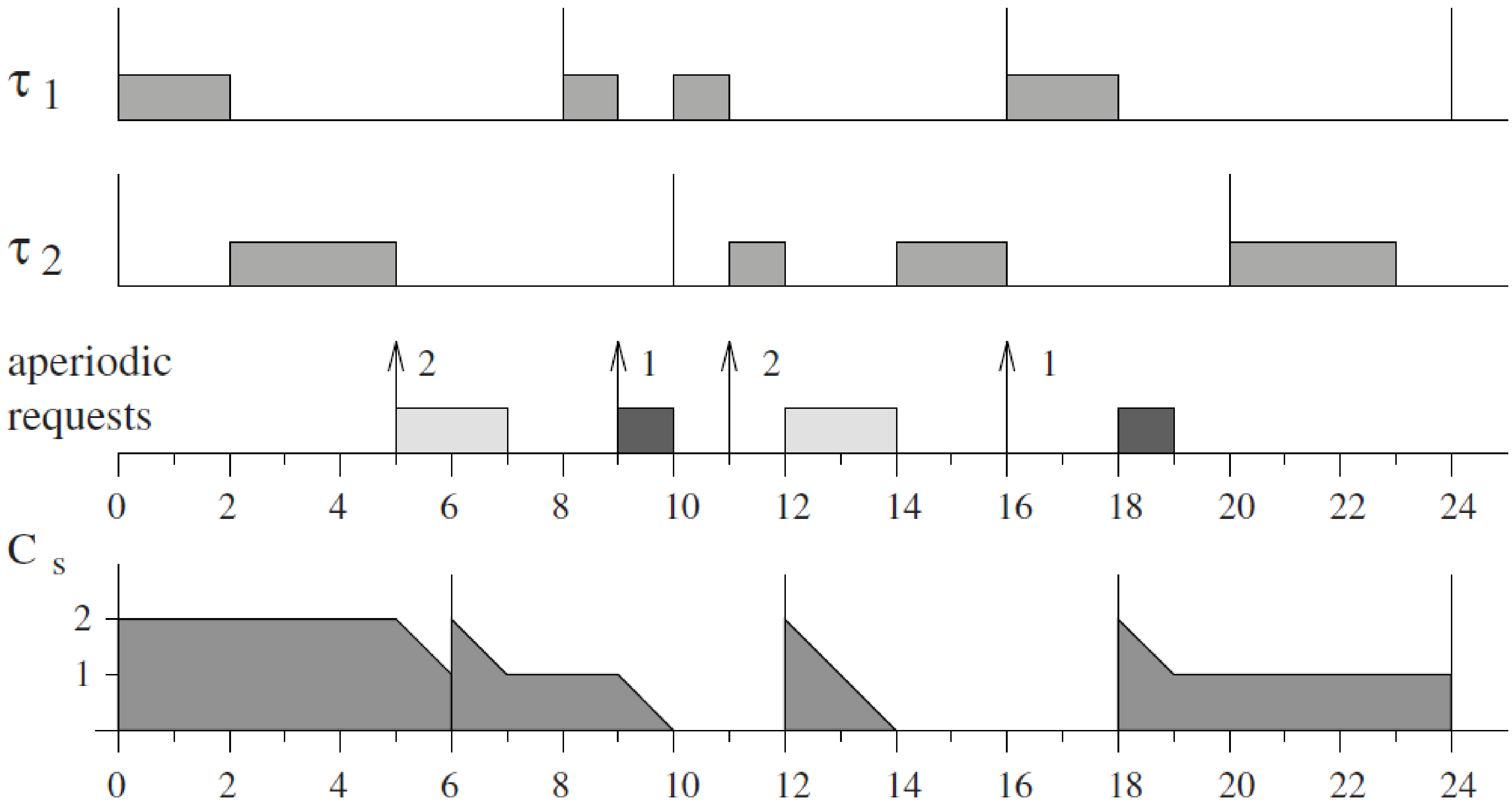
DS is the highest priority task

	C_i	T_i
τ_1	2	8
τ_2	3	10

Server

$C_s = 2$

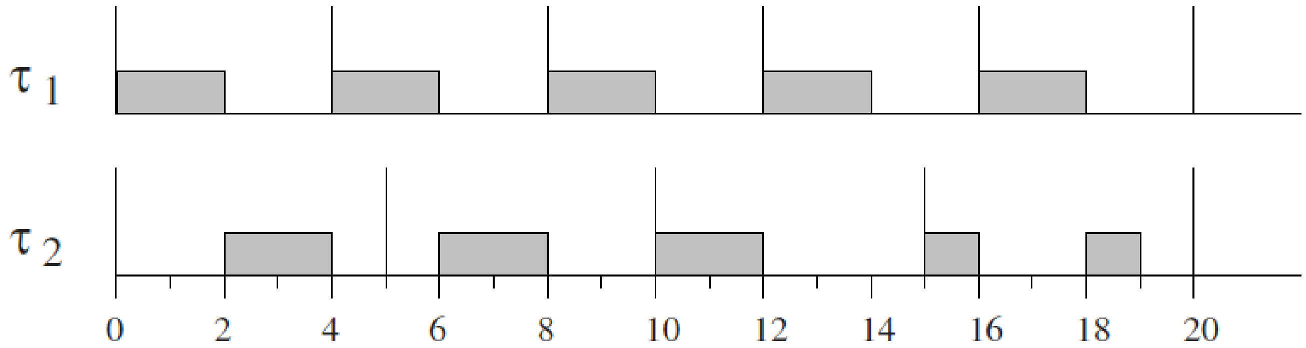
$T_s = 6$



Example 3

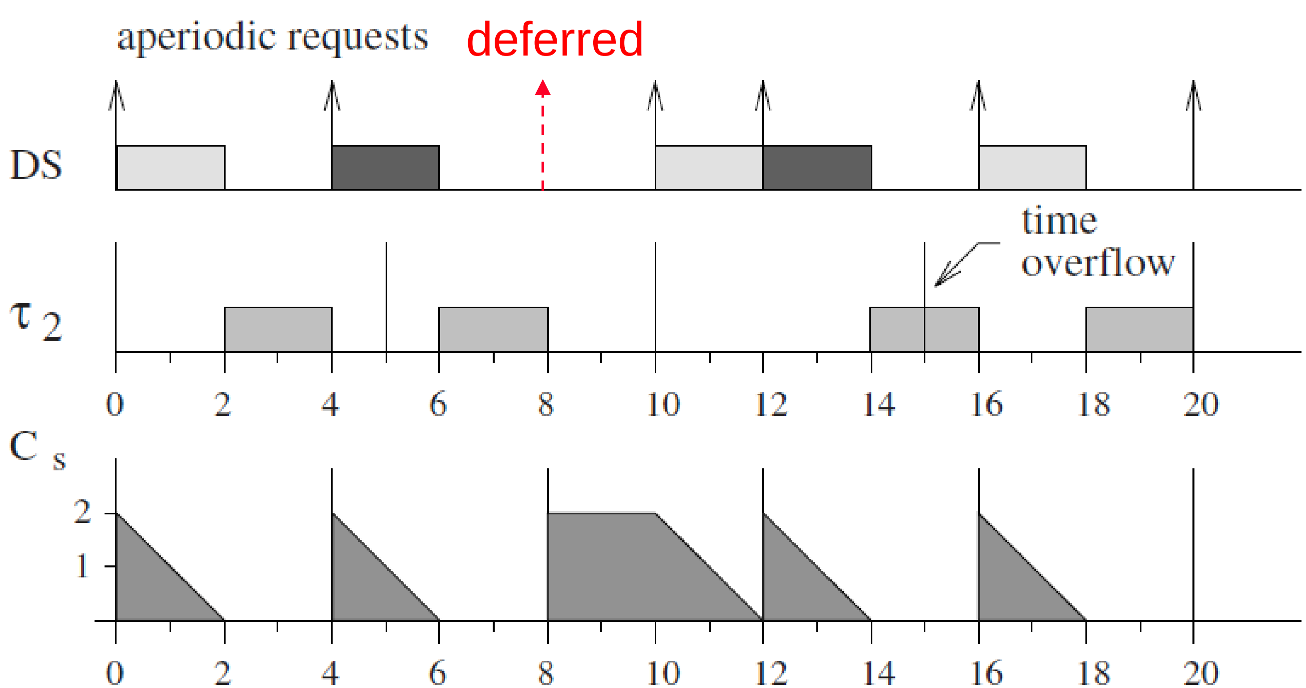
DS is not equivalent to a periodic task in RM → difficult schedulability analysis

Schedulable original task set



(a)

Replace T1 by DS → Not schedulable



DS schedulability analysis

- Given a periodic task set with total utilization U_p and a DS with utilization U_s
- The schedulability is guaranteed if

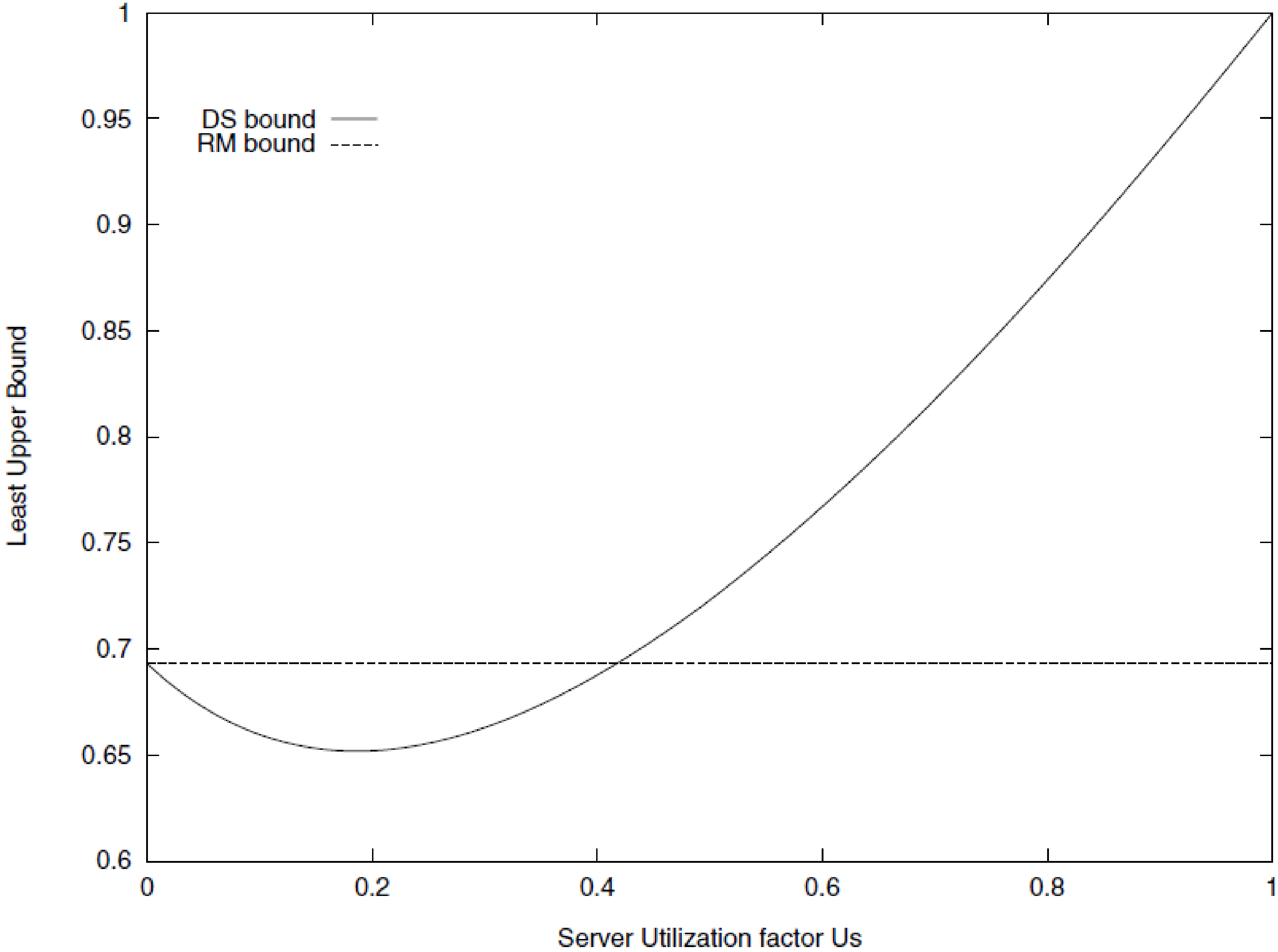
$$U_p \leq n\left[\left(\frac{U_s + 2}{2Us + 1}\right)^{\frac{1}{n}} - 1\right]$$

- Therefore the whole system bound is

$$U_{lub} = U_s + n\left[\left(\frac{U_s + 2}{2U_s + 1}\right)^{1/n} - 1\right]$$

$$\lim_{n\to\infty} U_{lub} = U_s + \ln\left(\frac{U_s + 2}{2U_s + 1}\right)$$

DS schedulability analysis



DS schedulability analysis

- Given a set of n periodic tasks with utilization U_i and a DS with utilization Us,

- The periodic task set is schedulable under RM if

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

Dimensioning a DS

- Find Us, Ts, Cs?
- Similar to PS, let

$$P \stackrel{\text{def}}{=} \prod_{i=1}^{n} (U_i + 1)$$

- Then from guarantee condition we have

$$U_s \leq \frac{2 - P}{2P - 1}$$

- So the max utilization for server is

$$U_s^{max} = \frac{2 - P}{2P - 1}$$

→choose $T_s = \min(T_i)$ and $C_s = Us * Ts$

Exercise 2

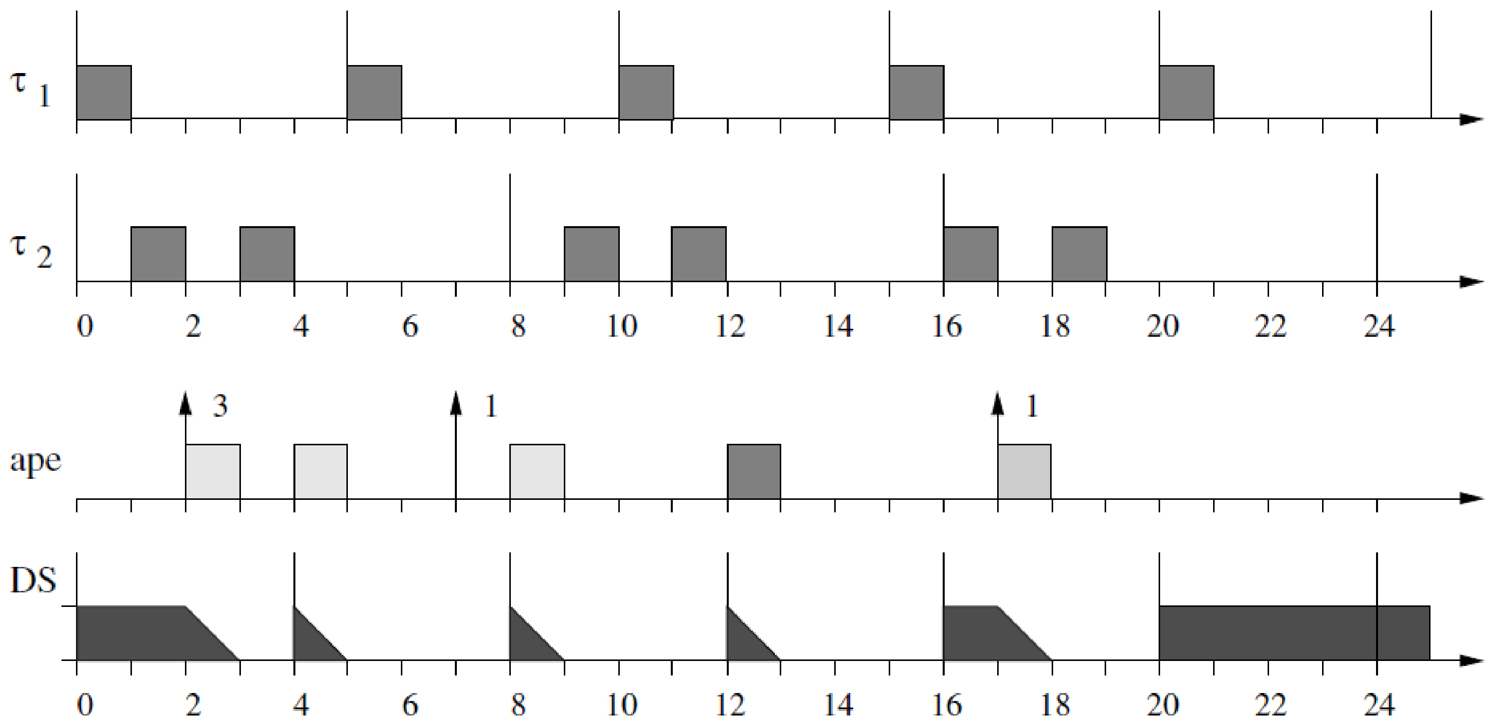
- From Ex1:
  - Periodic tasks: $C1$ = 1, T1=5, $C2$ = 2, T2=8 (scheduled by RM)
  - Aperiodic tasks:

	a_i	C_i
J_1	2	3
J_2	7	1
J_3	17	1

- Solve the scheduling problem based on DS, with highest possible priority and maximum utilization

Ex 2

- Maximum utilization: $U_{max} = 1/4$
- Highest priority with RM: $T_s = 4$
- ➔ $C_s = 1$



Priority Exchange (PE)

- Similar to DS with
  - Better schedulability bound
  - Worse aperiodic responsiveness
- PE algorithm
  - Create a periodic task (PE) with high priority for aperiodic load
  - PE preserves capacity by exchanging for lower priority tasks' execution time.
  - Upon PE activation: if there is no aperiodic load, lower priority tasks can execute and PE accumulates capacity at the corresponding priorities.
  - If there is no task waiting, PE capacity resolves to 0

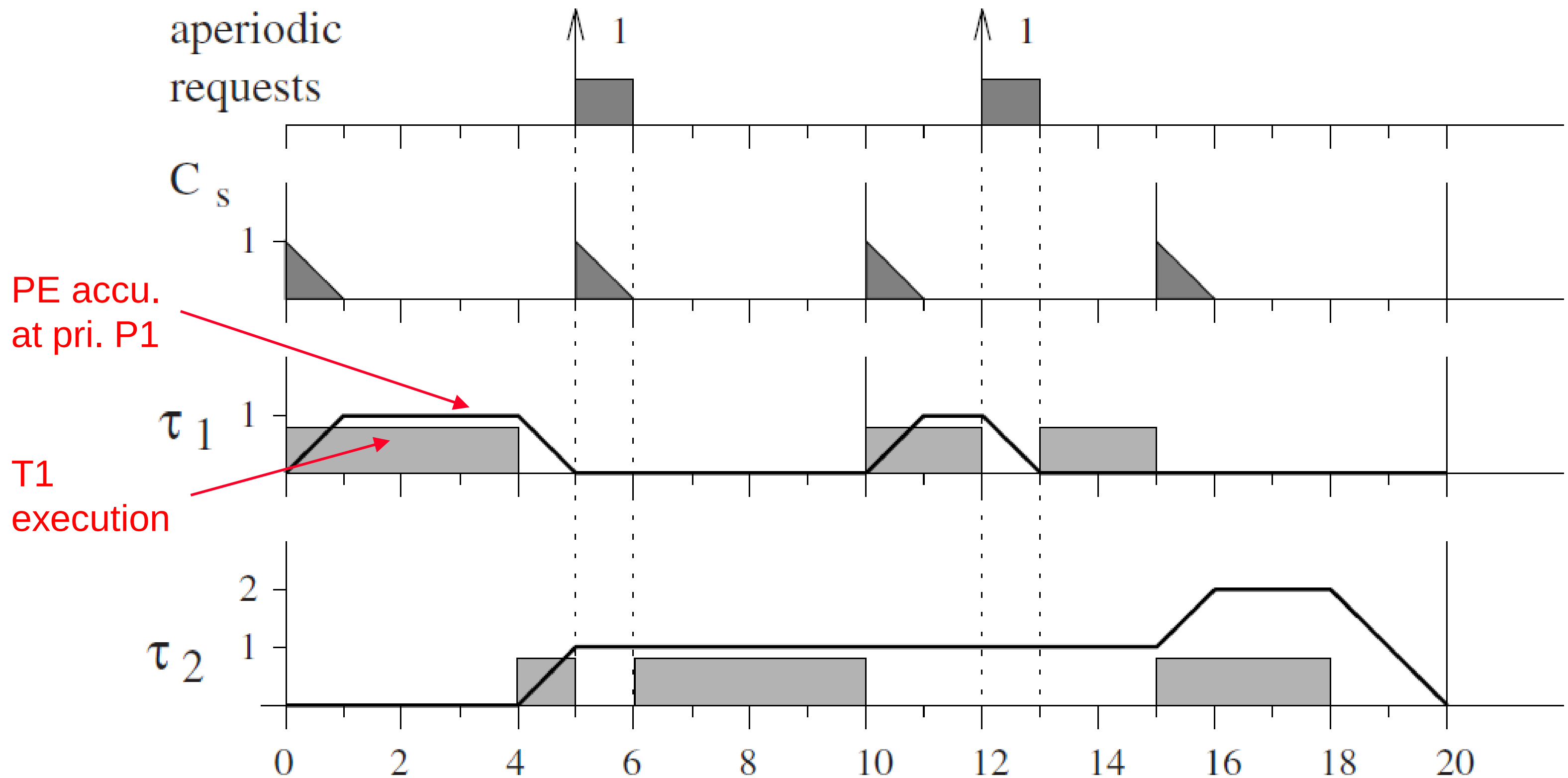
Example

	C_i	T_i
τ_1	4	10
τ_2	8	20

Server

$C_s = 1$

$T_s = 5$



Exercise 4

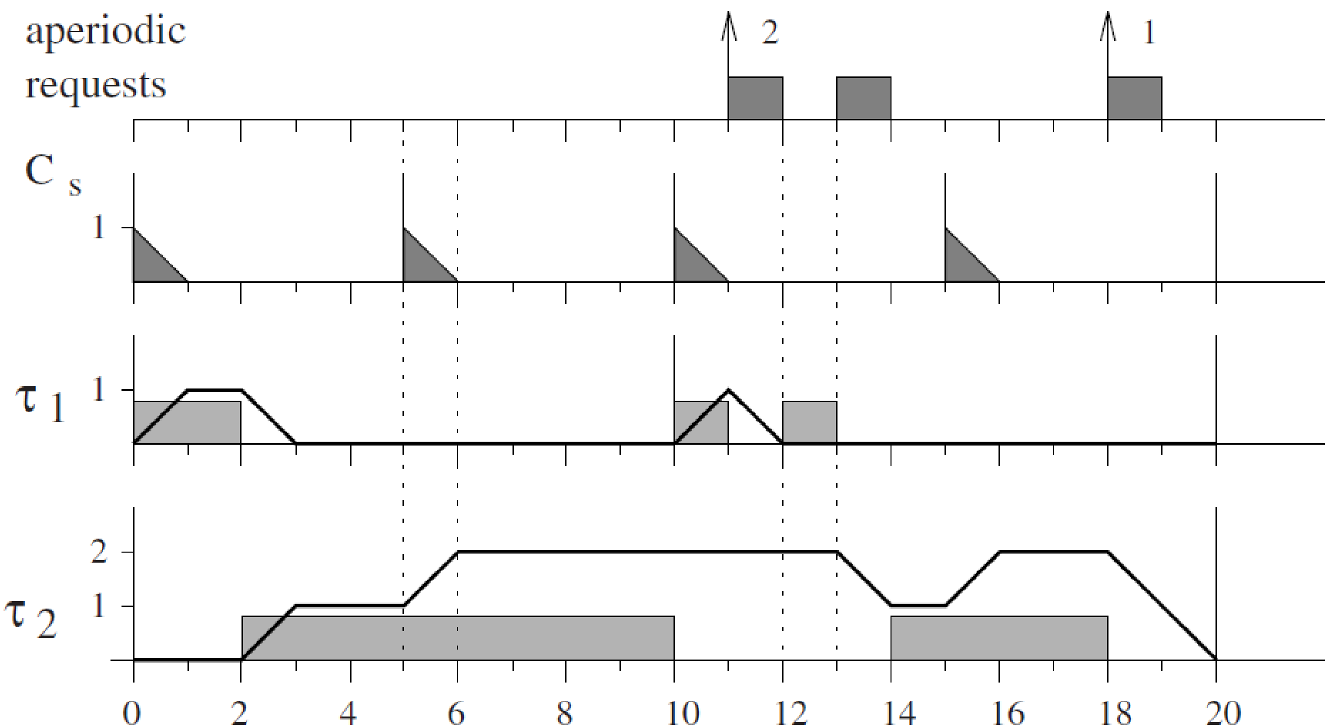
- Why in this case do the PE and T1 preempt each other?

	C_i	T_i
τ_1	2	10
τ_2	12	20

Server

$C_s = 1$

$T_s = 5$



Schedulability bound

- Given a periodic task set with total utilization U_p and a PE server with utilization U_s
- The schedulability is guaranteed if

$$U_p \leq n\left[\left(\frac{2}{U_s+1}\right)^{1/n} - 1\right]$$

- Sizing PE server

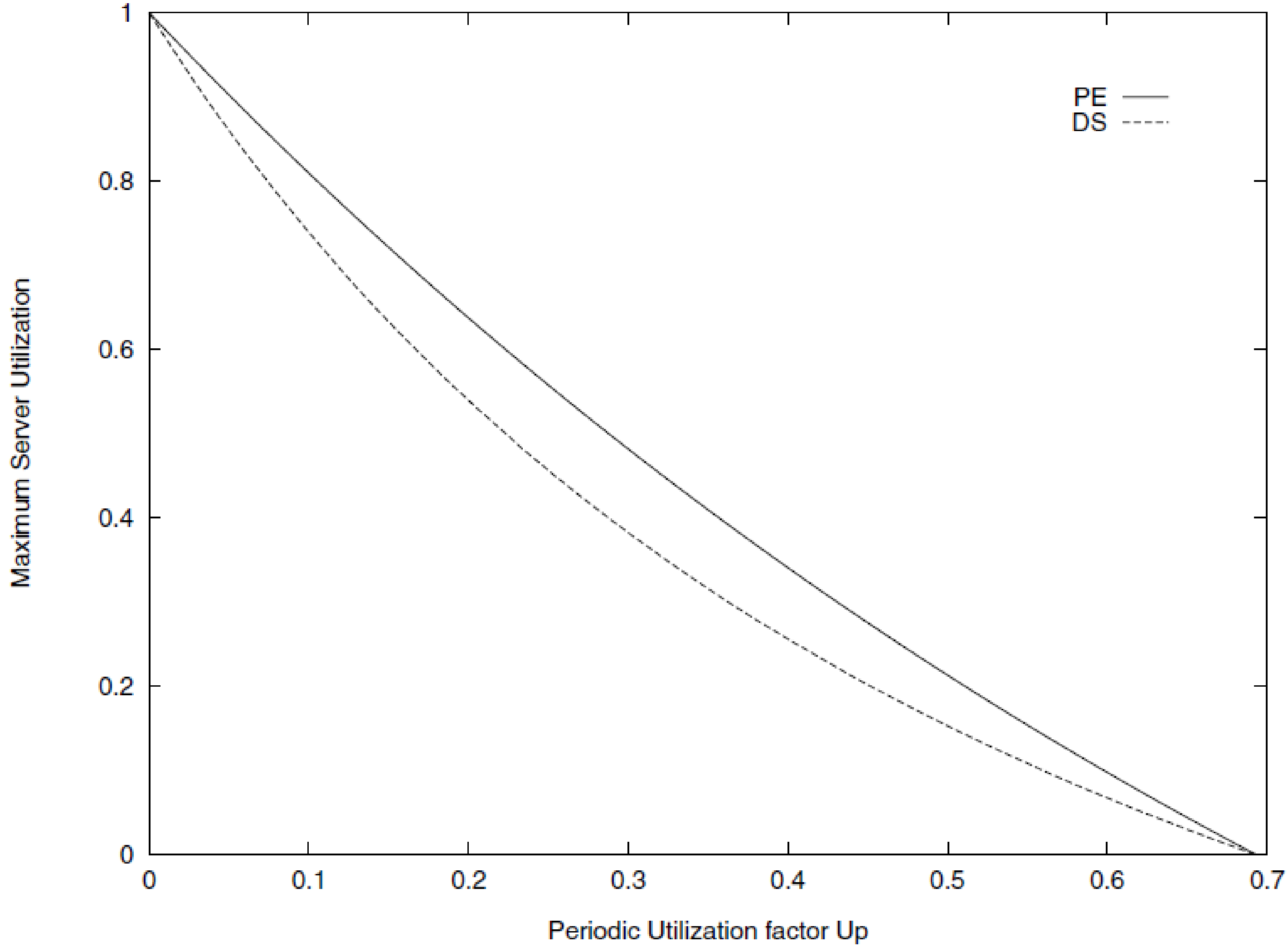
$$U_{PE}^{max} = \frac{2-P}{P}$$

where

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

Comparing Up between DS and PE

Which is better in term of periodic utilization?



Sporadic Server

- Similar to DS
- Delay the replenishing time of server → server becomes equivalent to a normal periodic task
- Idea:
  - Divide the timeline of SS to active and inactive time slices
    - Active: server serves or may serve periodic task
    - Inactive: server does not serve periodic task
  - Start of active time slice: mark delayed replenishing time
  - End of active time slice: calculate replenishing amount

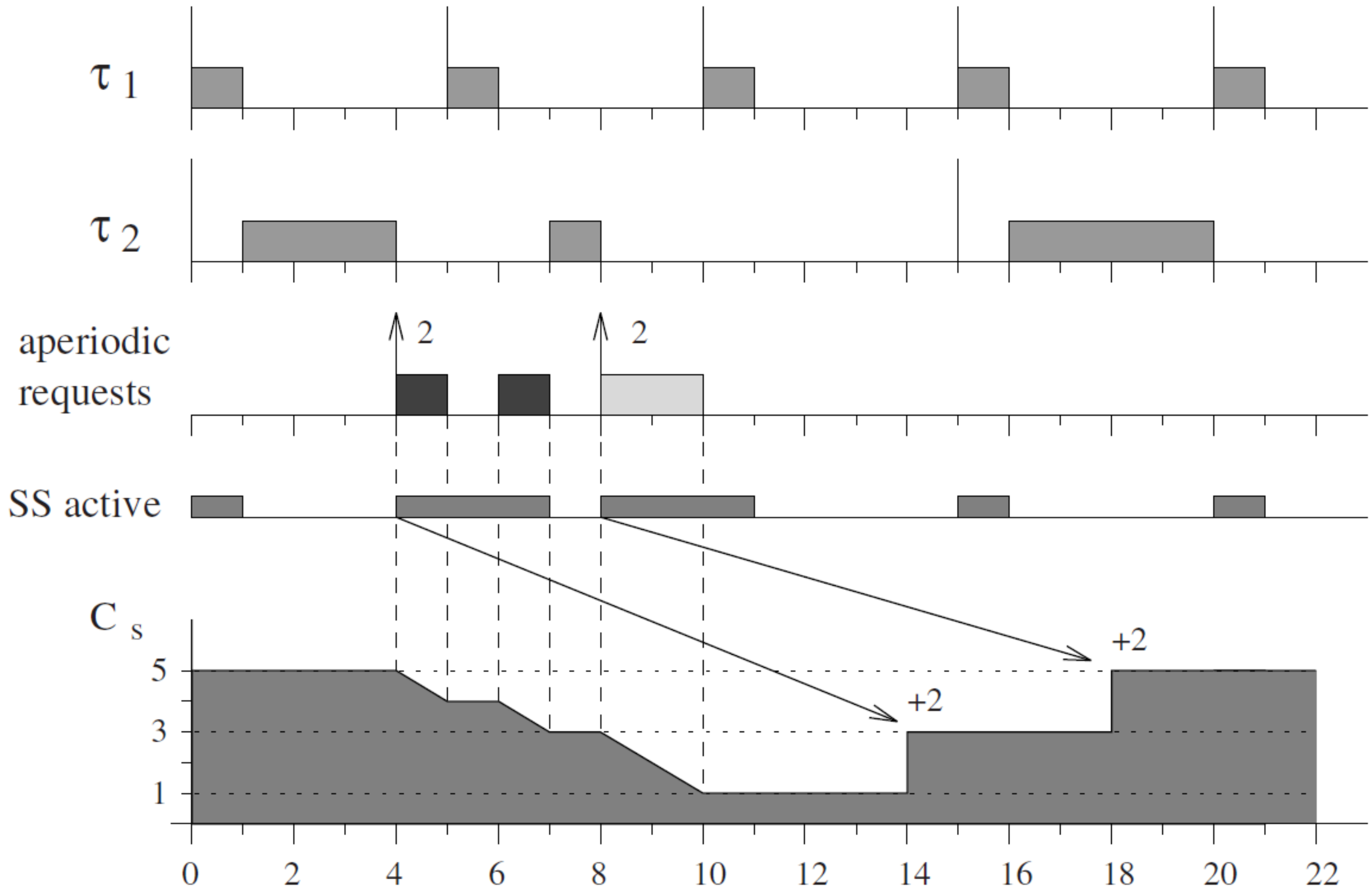
Example: intermediate SS

	C_i	T_i
τ_1	1	5
τ_2	4	15

Server

$C_s = 5$

$T_s = 10$



Sporadic Server

P_{exe}	It denotes the priority level of the task that is currently executing.
P_s	It denotes the priority level associated with SS.
Active	SS is said to be *active* when $P_{exe} \geq P_s$.
Idle	SS is said to be *idle* when $P_{exe} < P_s$.
RT	It denotes the *replenishment time* at which the SS capacity will be replenished.
RA	It denotes the *replenishment amount* that will be added to the capacity at time RT.

$$RT = Start\_of\_Active + Ts$$
$$RA = Consume\_capacity$$

Exercise 5:

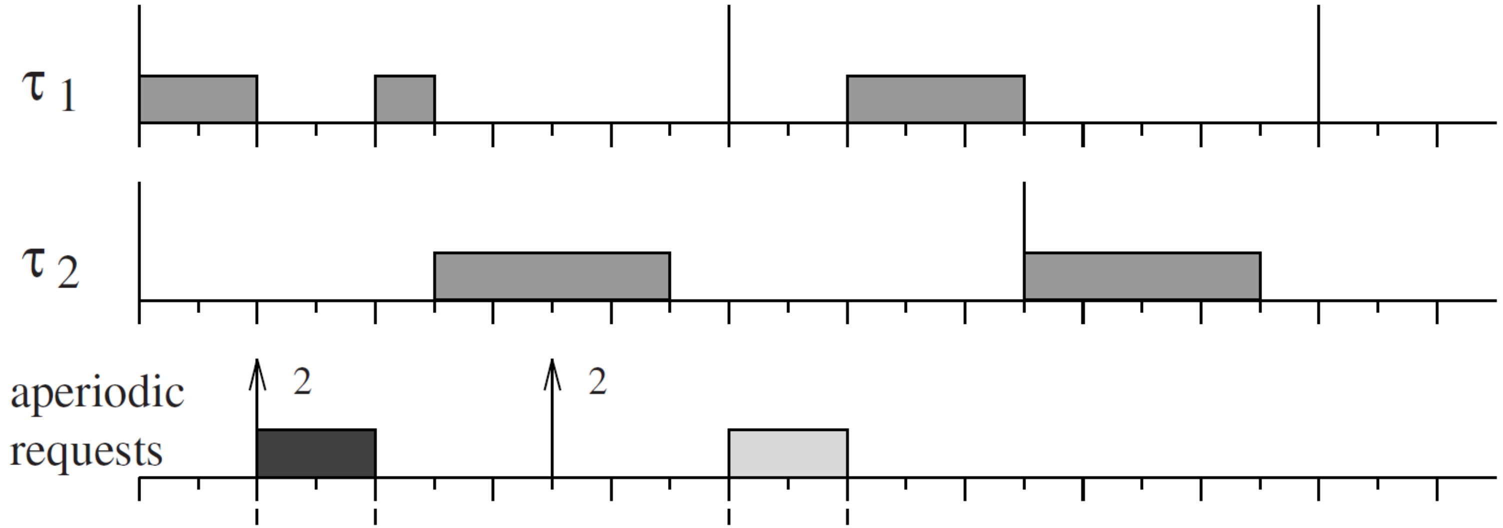
- Find response time of aperiodic tasks

	C_i	T_i
τ_1	3	10
τ_2	4	15

Server

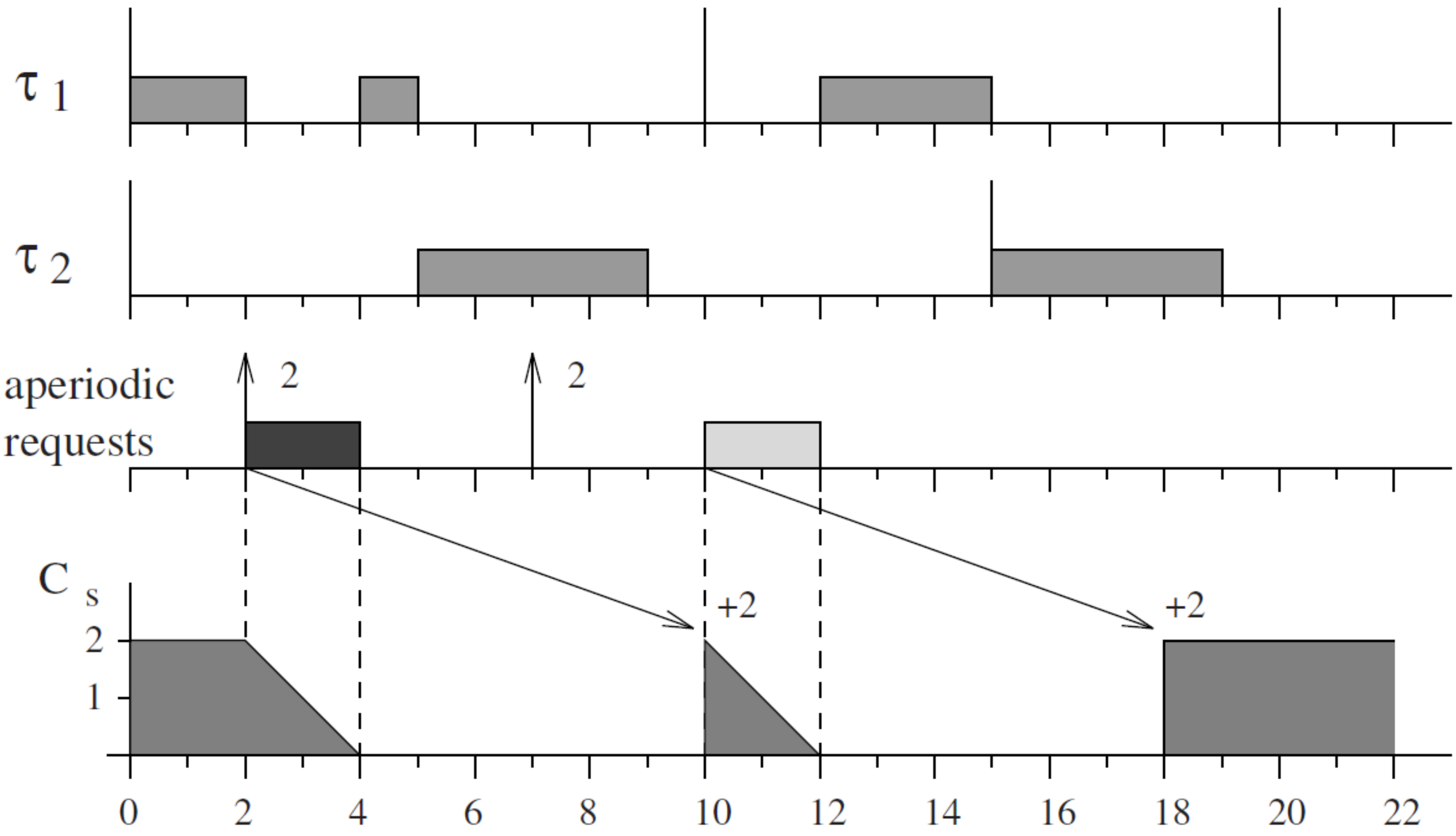
$C_s = 2$

$T_s = 8$



Exercise 5

- Server is with highest priority



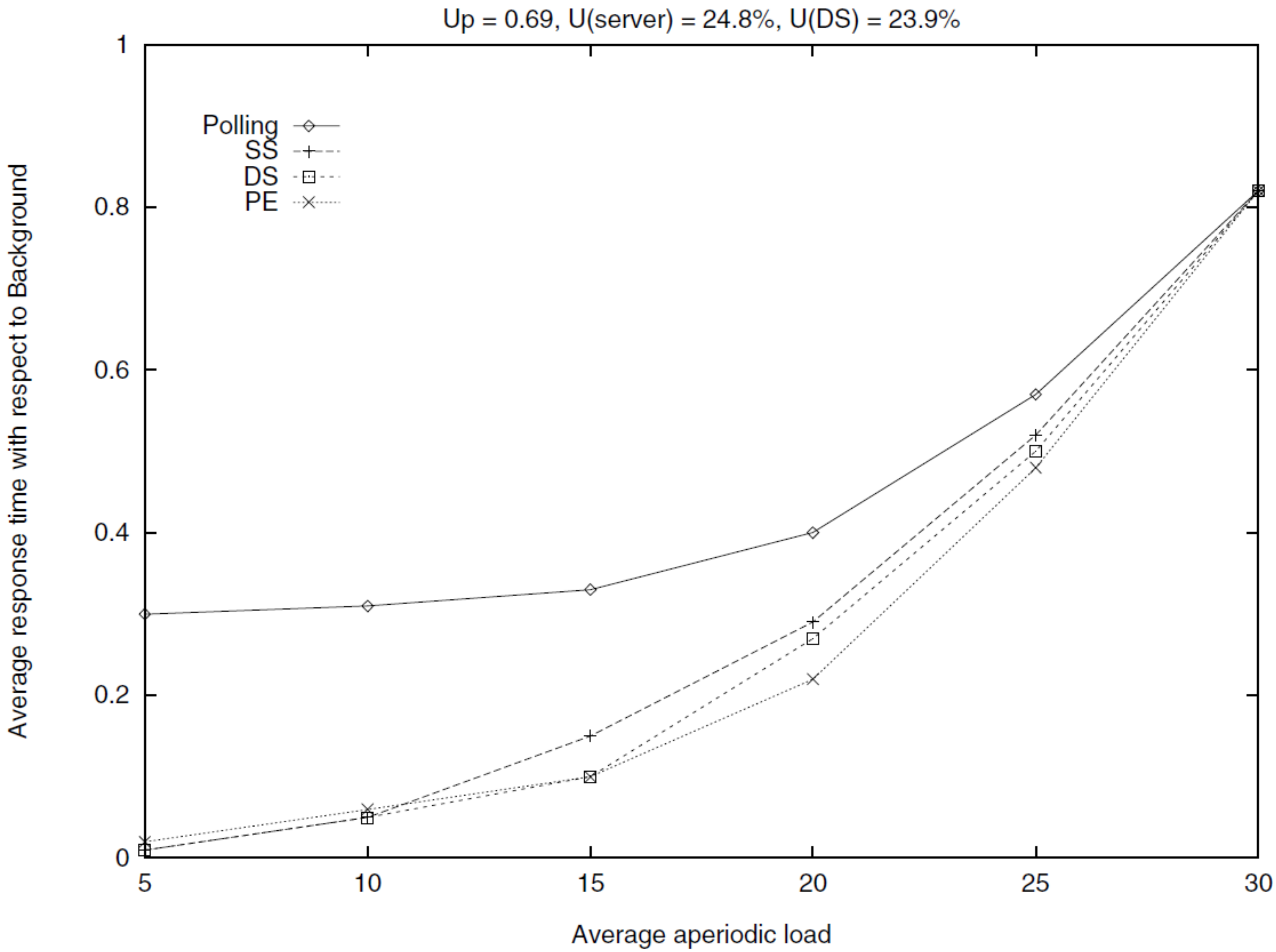
Periodic task equivalent

- SS (like DS) violates assumption: task must not suspend itself and reactivate later
- SS is different from DS: replenishing time is delayed
- **Theorem 5.1 (Sprunt, Sha, Lehoczky)** *A periodic task set that is schedulable with a task τi is also schedulable if τi is replaced by a Sporadic Server with the same period and execution time*
- Therefore, schedulability analysis of SS is similar to Polling Server with RM

$$U_p \leq n\left[\left(\frac{2}{U_s+1}\right)^{1/n} - 1\right]$$

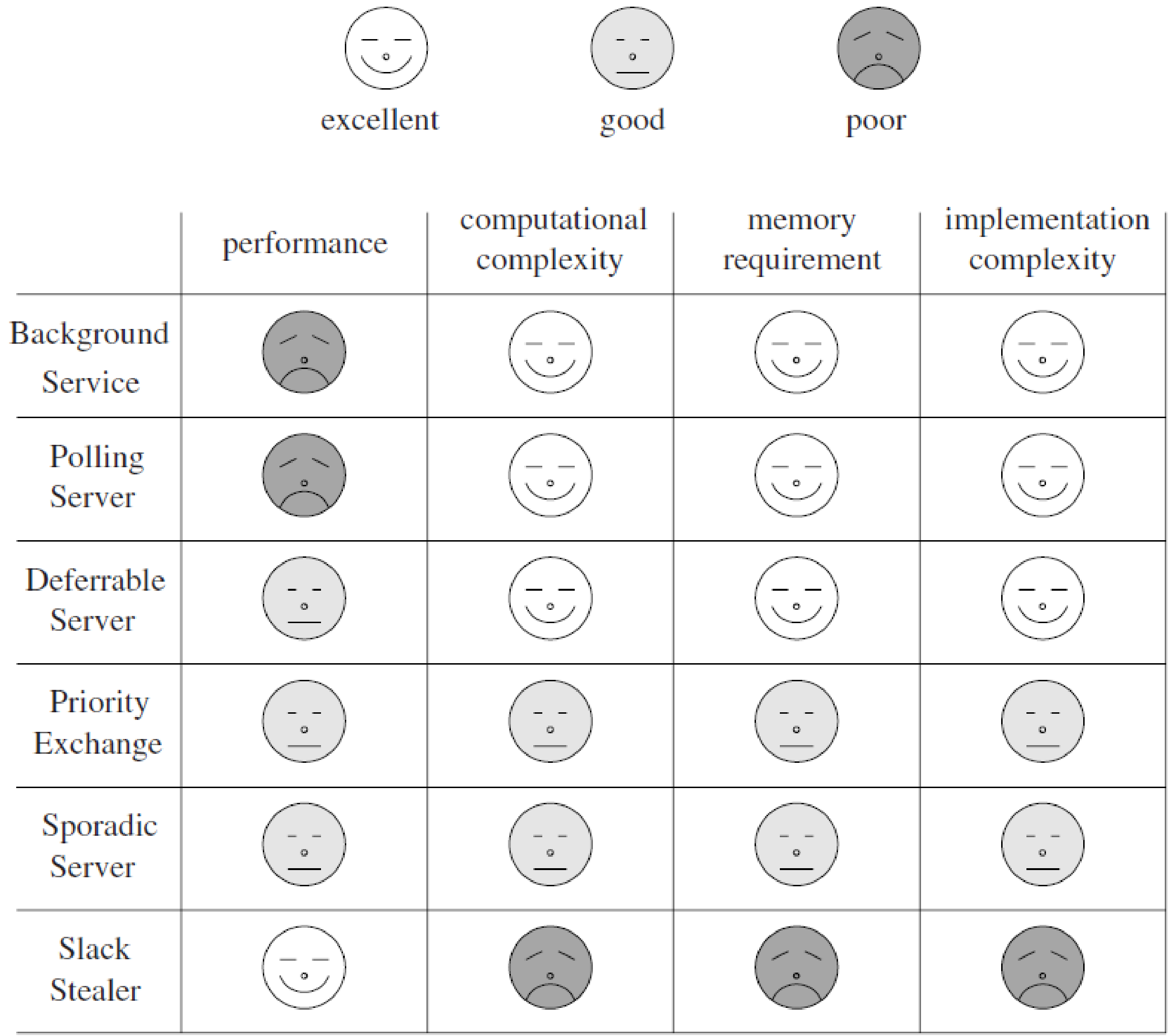
$$U_{SS}^{max} = \frac{2-P}{P}.$$

Performance evaluation



Performance results of PS, DS, PE, and SS

Comparison



Note: Slack Stealer can be read from textbook

Real-time Systems

Week 8: Dynamic Priority Servers

Ngo Lam Trung

Dept. of Computer Engineering

How to further increase U_{lub}?

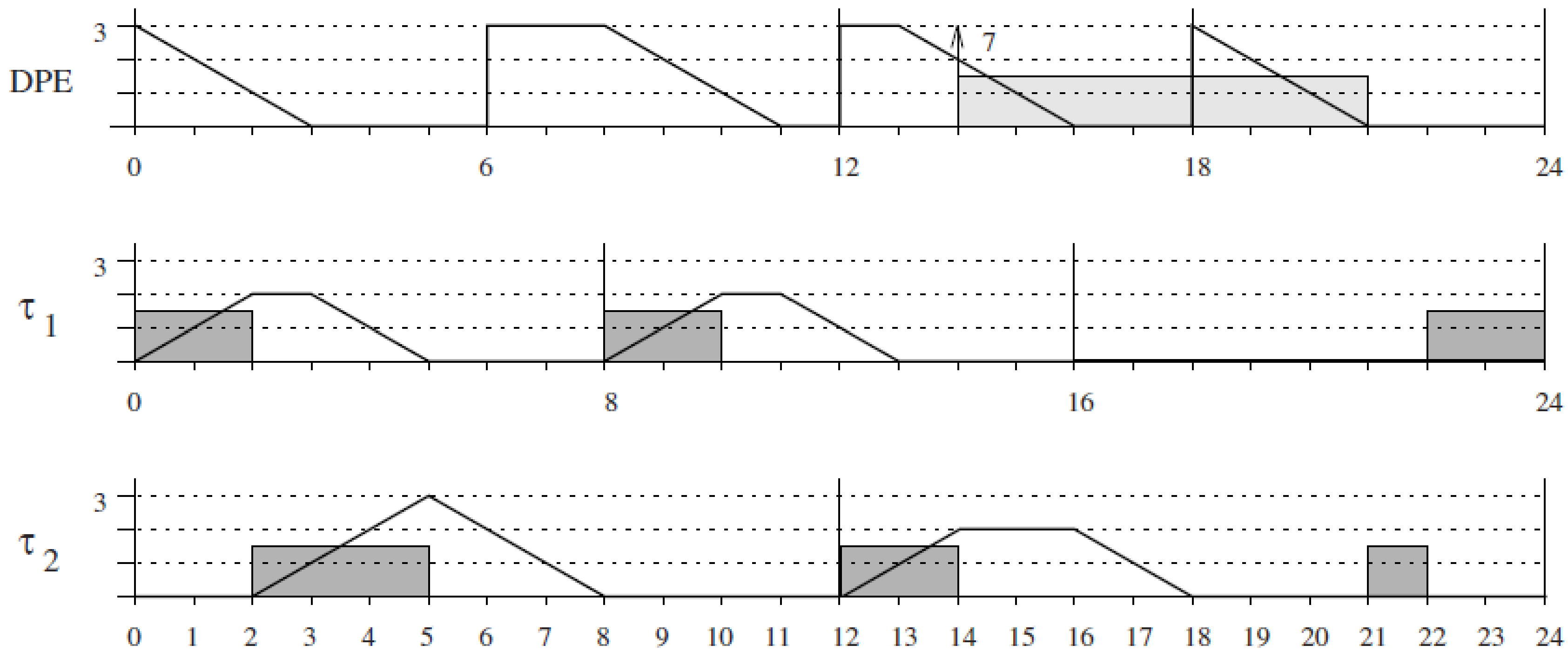
- Fixed priority server uses fixed priority algo.
  - Simple
  - Small U_{lub}
- How to increase U_{lub}?

➔uses the same approach: create periodic task to serve aperiodic task (the server)

→apply dynamic priority scheduling algorithm (EDF) to increase utilization bound

Dynamic Priority Exchange Server

- Similar to fixed priority exchange server
  - Server can exchange capacity with other tasks that have longer deadline at the scheduling time
  - Server accumulate capacity time with the new deadline
  - Server capacity will be consumed until it is exhausted

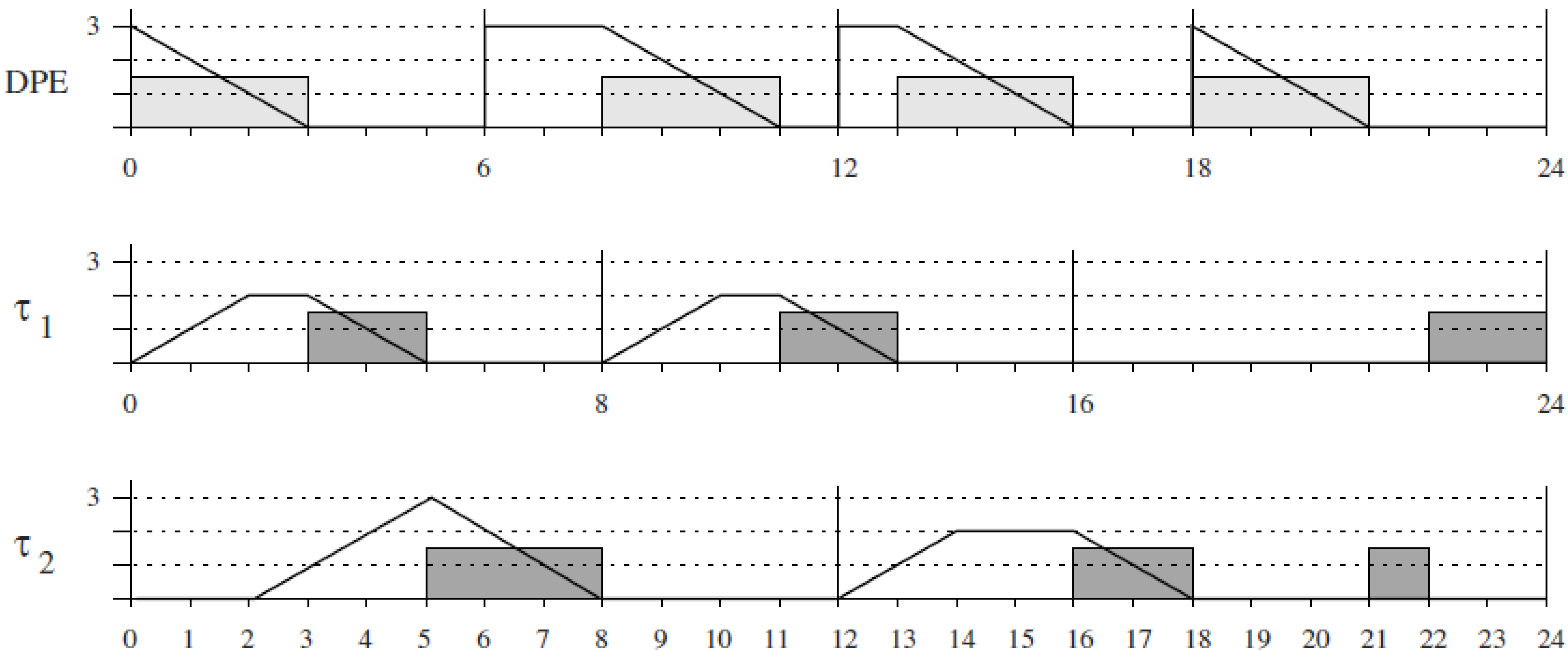


$\tau_1(2, 8)$, $\tau_2(3, 12)$, DPE(3,6)

Schedulability analysis

Theorem 6.1 (Spuri, Buttazzo) *Given a set of periodic tasks with processor utilization U_p and a DPE server with processor utilization U_s, the whole set is schedulable by EDF* if *and* only if

$$U_p + U_s \leq 1.$$

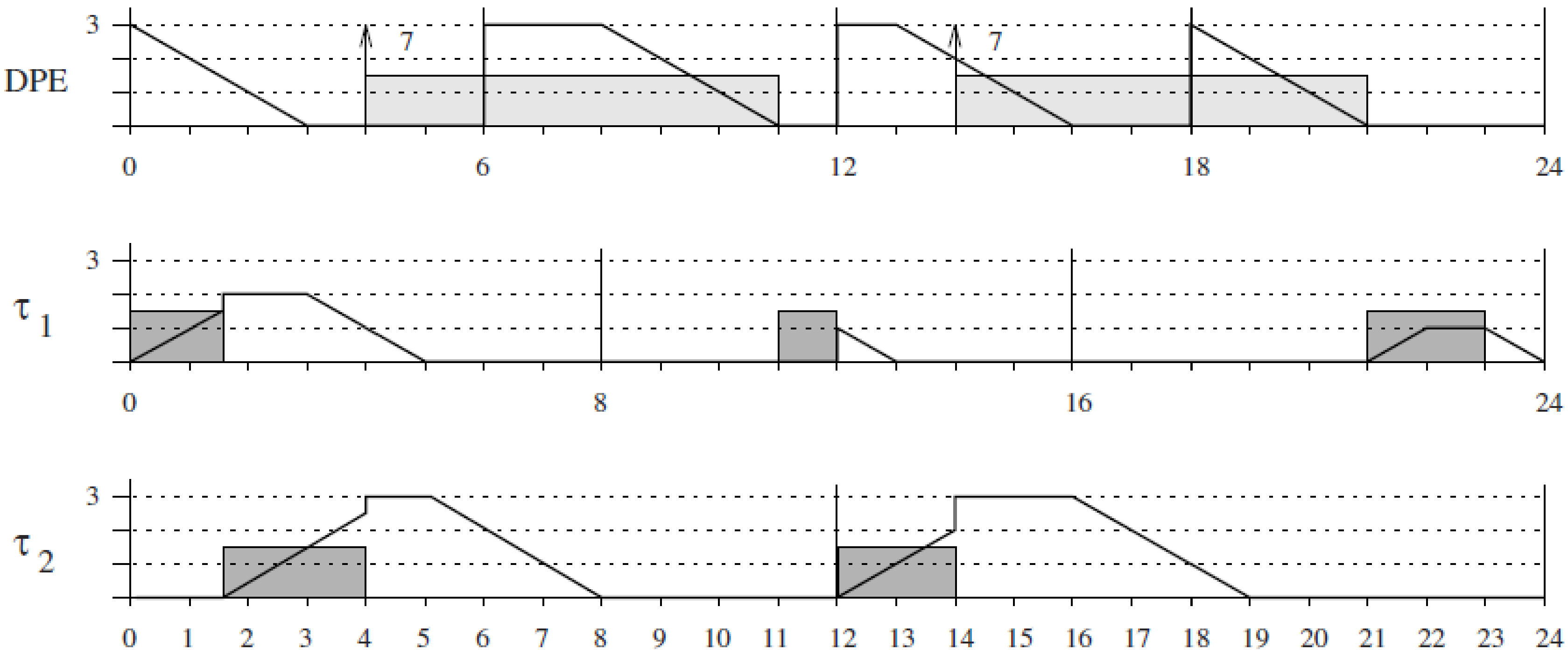


$U = \frac{3}{6} + \frac{2}{8} + \frac{3}{12} = 1$ ➔ schedulable task set

Reclaiming spare capacity

- What if the real C is smaller than worst case C?

→ Spare capacity from can be reclaimed and transfer to aperiodic capacity.



Real-time Systems

Chapter 9:
Resource Access Protocol

Ngo Lam Trung

Dept. of Computer Engineering

Contents

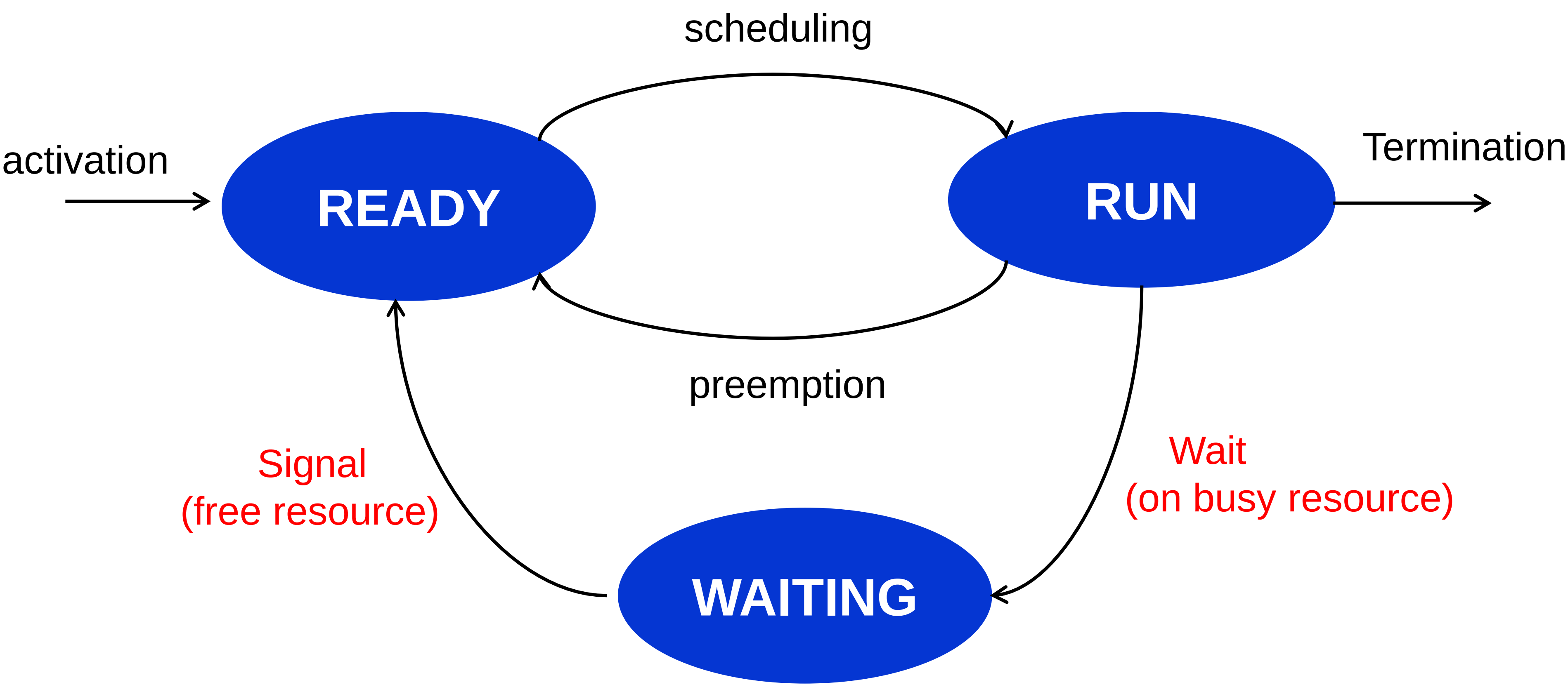
- Introduction
- The priority inversion phenomenon
- Solutions for priority inversion

Resource constraint

- Resource
  - Any software structure that can be used by the process to advance its execution
  - Ex: data structure, variables, main memory area, a file, a piece of program, a set of registers of a peripheral device
- Many shared resources do not allow simultaneous access

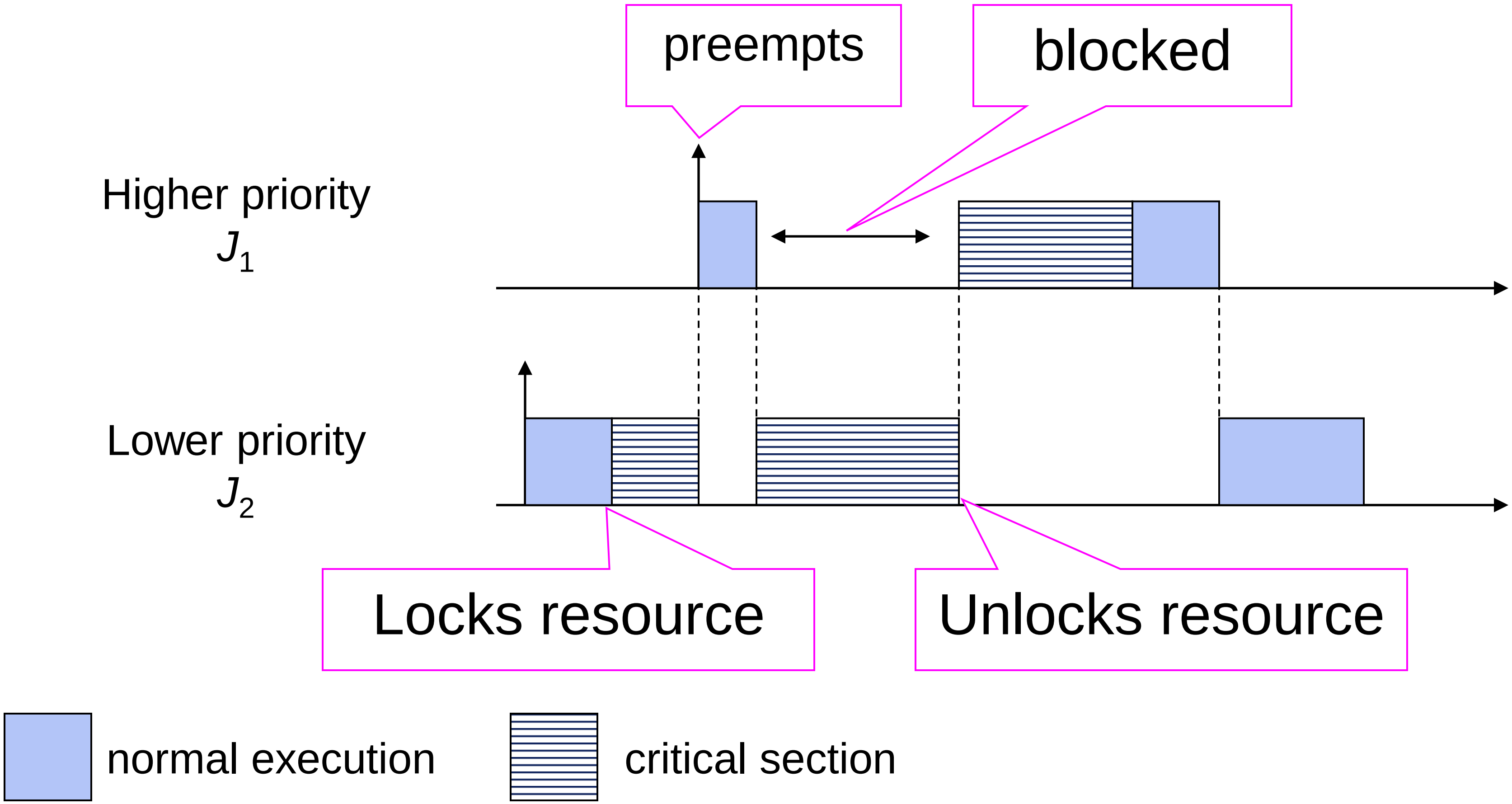
  → require mutual exclusion
- Critical section
  - A piece of code under mutual exclusion constraints
  - Tasks entering critical section have to wait until no other task is holding the resource

Waiting state caused by resource constraint



Example of blocking on exclusive resource

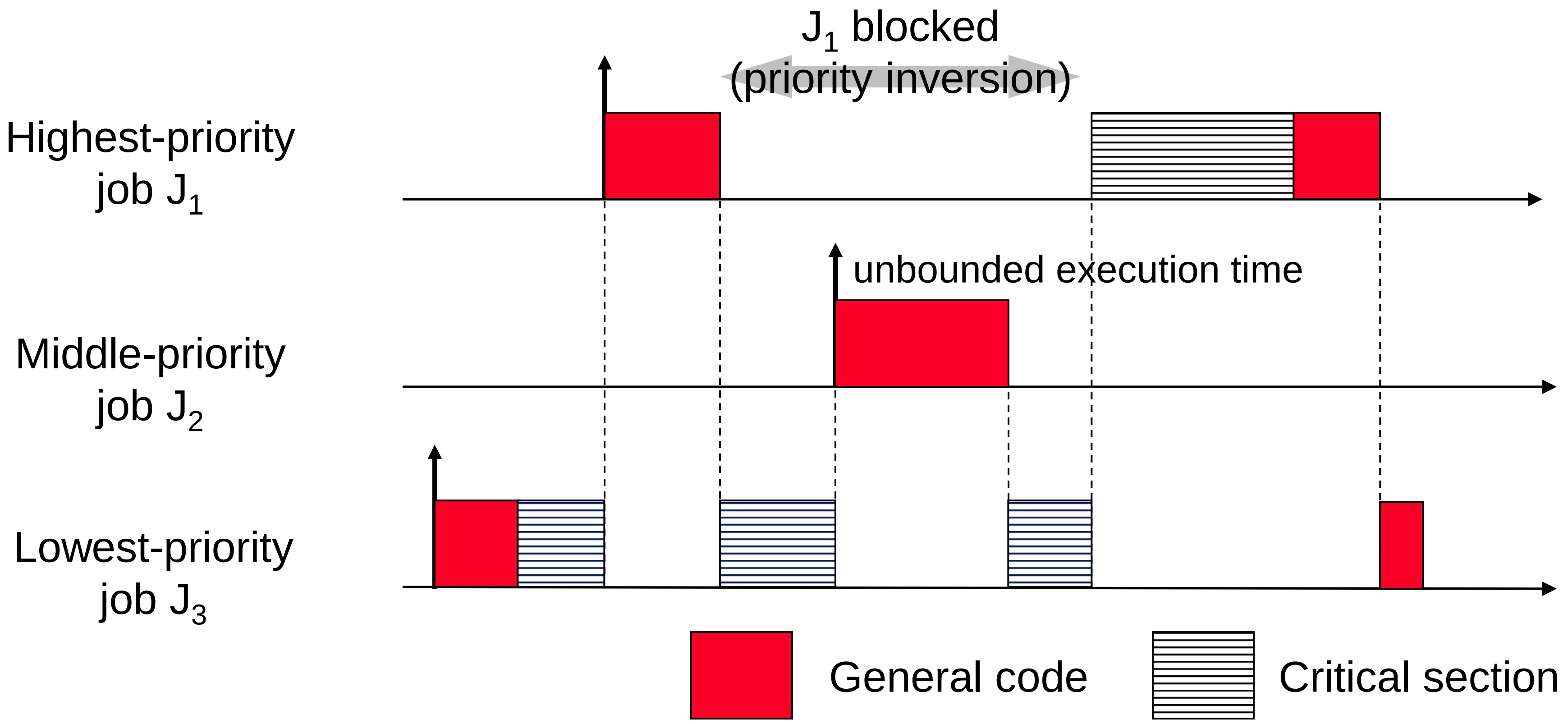
- Scheduling with preemption



Problem: the task with higher priority has to wait for the task with lower priority

The priority inversion phenomenon

- J3 enters critical section first
- J1 is blocked, has to wait until J3 signal the resource
- J2 preempts J3 ➔ J1 has to wait for J2



Problems

- The task with higher priority has to wait for the task with lower priority
- Blocking time is unbounded → the system is not predictable.
- Example of priority inversion: Mars Pathfinder 1997
  - CPU: RAD6000 20MHz ($200K-$300K)
  - OS: VxWork
  - Experienced CPU reset upon touching down on Mars, debugging on Earth detected priority inversion, fixed by new firmware upload.

Problems

- Solutions
  - Non-preemptive Protocol
  - Highest Locker Priority Protocol
  - Priority Inheritance Protocol
  - Priority Ceiling Protocol
  - Stack Resource Policy

Terminology & assumptions(1)

- Periodic task set $\Gamma = \{\tau_1, \tau_2, \ldots, \tau_n\}$
  - $\tau_i = (C_i, T_i)$
  - Relative deadline $D_i = T_i$
- Resources $R_1, \ldots, R_m$
  - Each R_k is guarded by semaphore S_k

- J_i: a job of τ_i
- P_i: nominal priority of τ_i
- $p_i \geq P_i$: active priority of τ_i (initially set to P_i)
- $z_{i,j}$: j-th critical section of J_i
- $d_{i,j}$: duration of $z_{i,j}$
- $S_{i,j}$: the semaphore guarding $z_{i,j}$
- $R_{i,j}$: the resource used in $z_{i,j}$
- Notation $z_{i,j} \subset z_{i,k}$ means $z_{i,j}$ is entirely contained in $z_{i,k}$.

Terminology & assumptions (2)

- Assumptions
  - $J_1,\ldots,J_n$ are listed in decreasing order of P_i
  - Jobs don't suspend themselves.
  - The critical sections used by any task are properly nested.

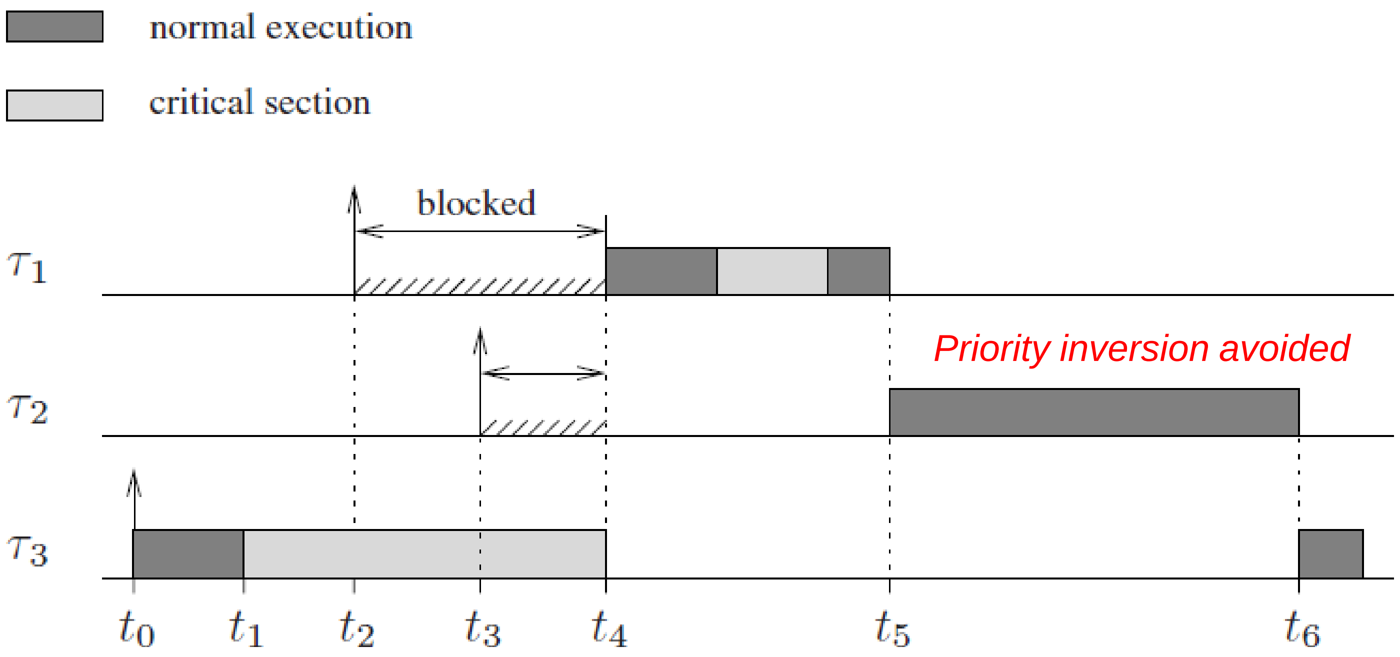
$$z_{i,j} \subset z_{i,k} \text{ or } z_{i,k} \subset z_{i,j} \text{ or } z_{i,j} \cap z_{i,k} = 0$$

  - Critical sections are guarded by binary semaphores.

The simplest: Non-preemptive Protocol

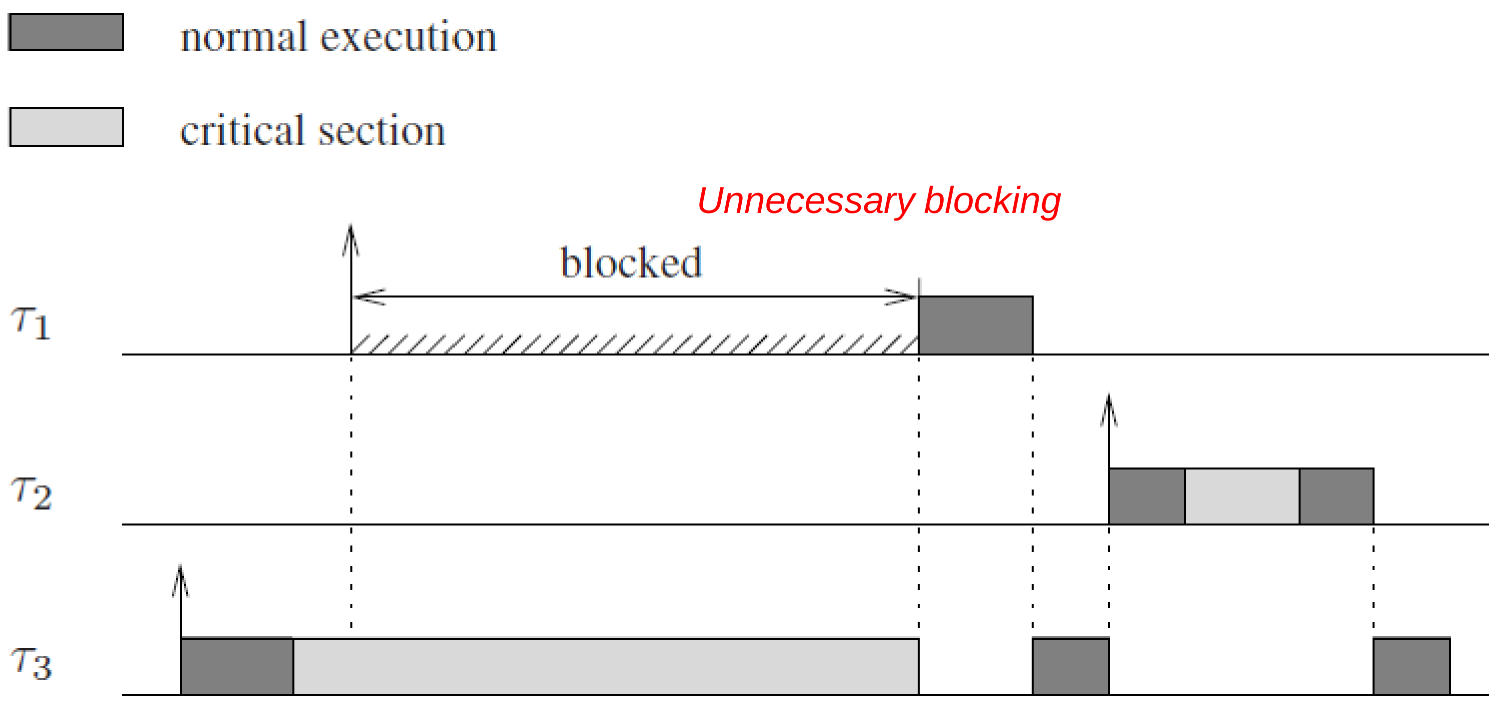
- Block all other tasks whenever a task enters a critical section
- The dynamic priority of the running task is raised to the highest level

$$p_i(R_k) = \max_h \{P_h\}$$



The simplest: Non-preemptive Protocol (NPP)

- Pros: simple
- Cons: unnecessary blocking



Blocking time of Non-preemptive Protocol

- Given the task T_i, the set of critical sections that can block T_i

$$\gamma_i = \{Z_{j,k} \mid P_j < Pi, k = 1, \ldots, m\}$$

- The maximum blocking time is

$$B_i = \max\{d_{j,k} - 1 \mid Z_{j,k} \in \gamma i\}.$$

➔ Duration of the longest critical section that can block T_i

Highest Locker Priority Protocol (HLP)

- Improves NPP: raising the priority of the task entering a critical section to the highest priority among the tasks sharing that resource.
- When a task enters resource R_k, its dynamic priority is raised to

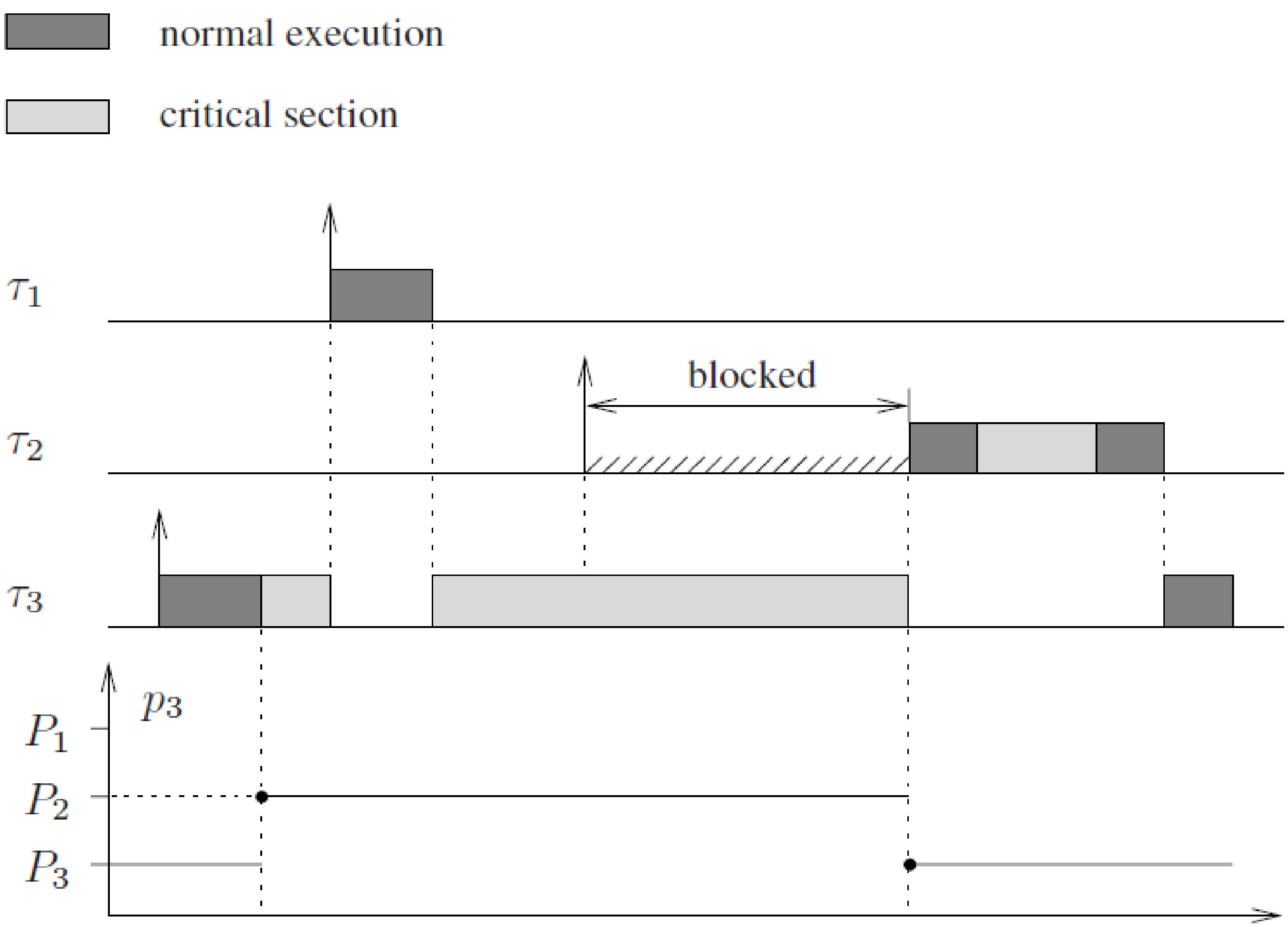
$$p_i(R_k) = \max_h \{P_h \mid \tau_h \text{ uses } R_k\}$$

- When the task exits the resource, its dynamic priority is reset to the nominal value P_i
- Priority ceiling can be computed offline

$$C(R_k) \stackrel{\text{def}}{=} \max_h \{P_h \mid \tau_h \text{ uses } R_k\}$$

Highest Locker Priority Protocol

- Example



HLP Blocking time

- The set of critical instants that can block task T_i

$$\gamma_i = \{Zj_{,k} \mid (P_j < Pi) \text{ and } C(R_k) \geq Pi\}$$

- Hence, maximum blocking time is

$$B_i = \max_{j,k}\{\delta_{j,k} - 1 \mid Z_{j,k} \in \gamma_i\}$$

- Problem: what if critical section is access in only one branch of a conditional statement?

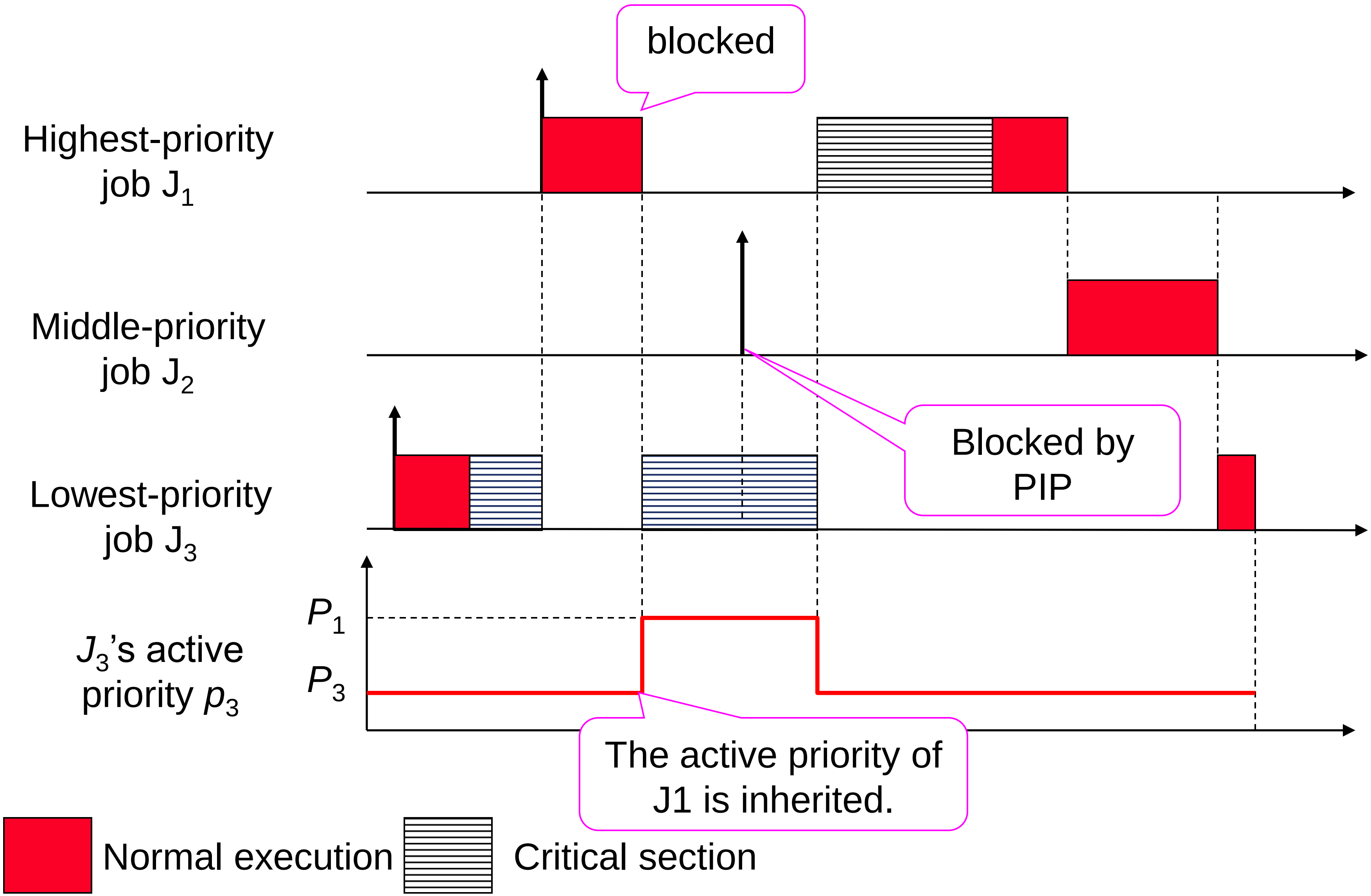
Priority Inheritance Protocol

- Modify the priority of tasks in critical sections
- When a task blocks higher-priority tasks, it temporarily ***inherits*** the highest priority of the blocked tasks.
  - Prevents preemption of medium-priority tasks

Protocol definition

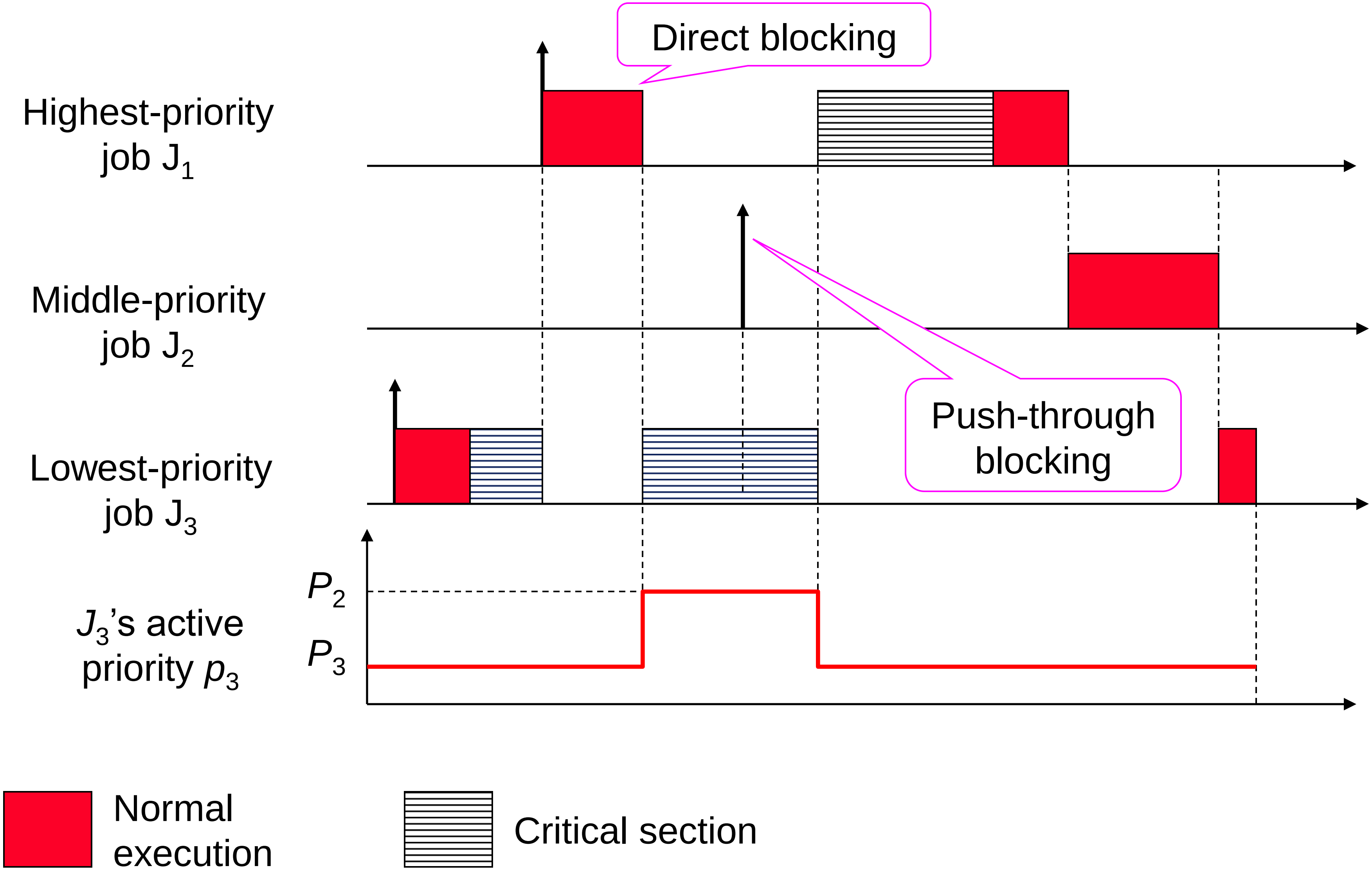
- Jobs are scheduled based on their active priorities
- When the higher-priority job J_{high} is blocked on a semaphore because the lower-priority job J_{low} is in execution of its critical section, **the active priority p_{high} of J_{high} is inherited** to that of J_{low}.
- The rest of the critical section of J_{low} is executed with the active priority p_{high}.
- In case the medium-priority job J_{medium} activates, it cannot preempt the execution of J_{low} → Unbounded priority inversion is avoided.
- Priority inheritance is transitive; if a job J_3 blocks a job J_2, and J_2 blocks a job J_1, then J_3 inherits the priority of J_1 via J_2.

Example



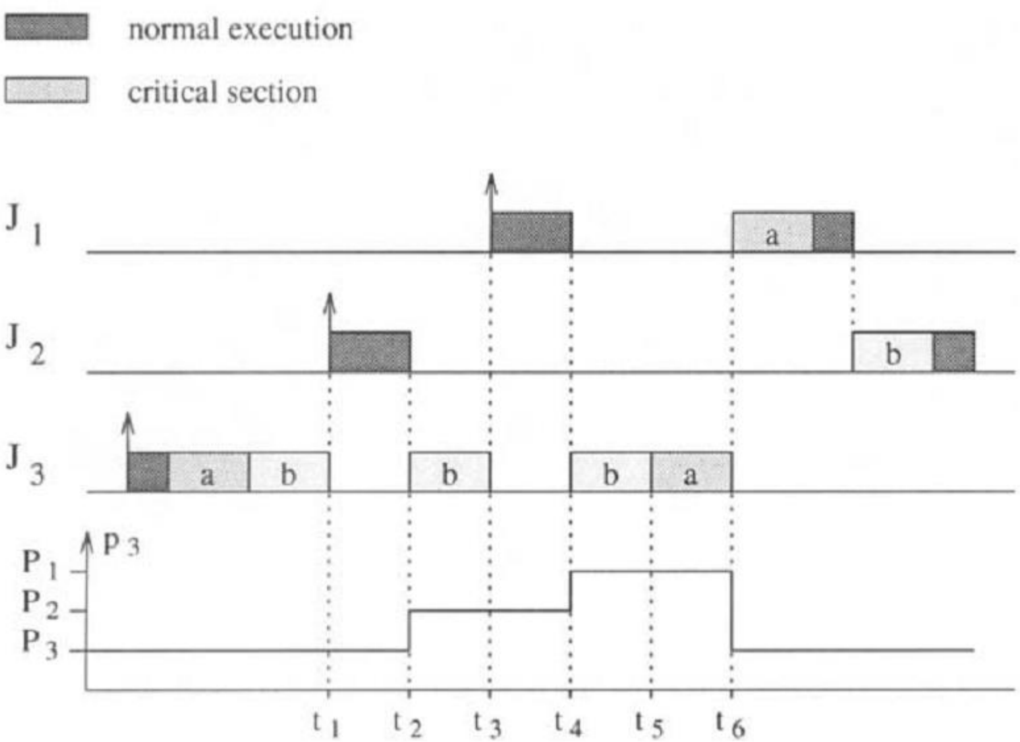
blocked
Highest-priority job J_1
Middle-priority job J_2
Blocked by PIP
Lowest-priority job J_3
J_3's active priority p_3
P_1
P_3
The active priority of J1 is inherited.
Normal execution
Critical section

Direct blocking & Push-through blocking



PIP with nested critical sections

- When the blocking job J_k exits the critical section, the blocked job with the highest priority is awakened.
- J_k replaces its active priority p_k by nominal priority P_k if no other jobs are blocked by J_k, or by the highest priority of the tasks blocked by J_k



Transitive priority inheritance

Properties

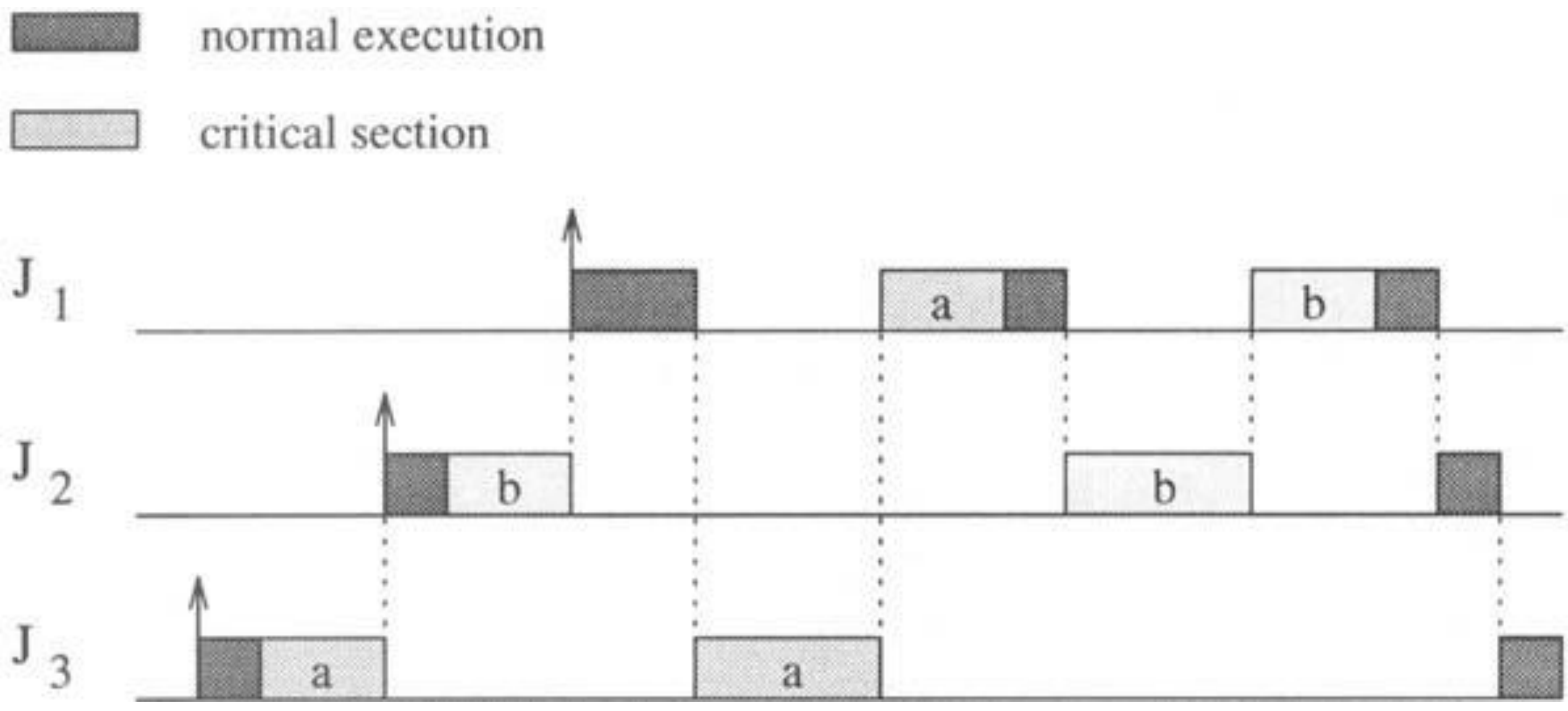
- Push-through blocking to job J_i occurs only if the semaphore is accessed by a job J_{low} with $p_{\text{low}}<p_i$ and by a job J_{high} with p_{high} that can be equal or higher than p_i
- Transitive priority inheritance can occur only in the presence of nested critical sections.
- If there are n lower-priority jobs that can block a job $\mathrm{J_i}$, then J_i can be blocked at most the duration of n critical sections.
- If there are m distinct semaphores that can block a job J_i, then J_i can be blocked for at most the duration of m critical sections.

Properties

- Under the priority inheritance protocol, a job *J* can be blocked for at most the duration of min(*n*,*m*) critical sections.
  - n is the number of lower-priority jobs that could block *J*
  - m is the number of distinct semaphores that can be used to block *J*

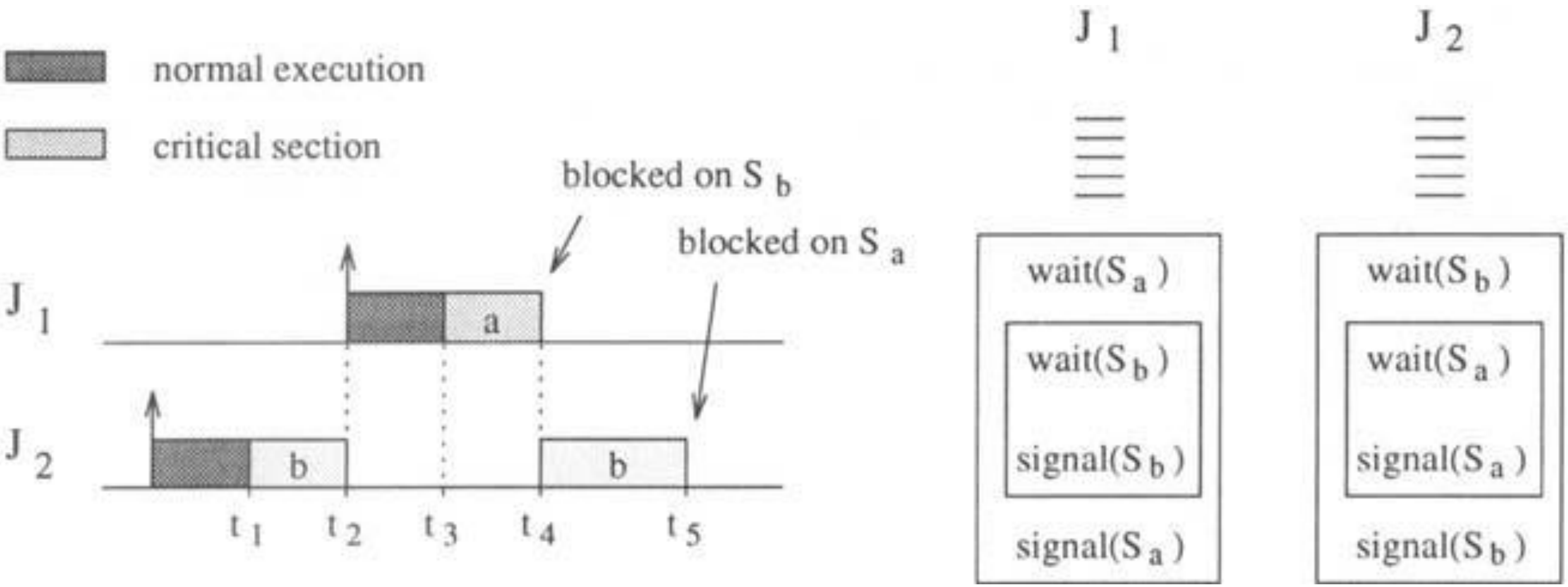
➔ *The maximum blocking time for any task J is bounded*

Remaining problem 1: Chained blocking



➔ J1 can be blocked several times

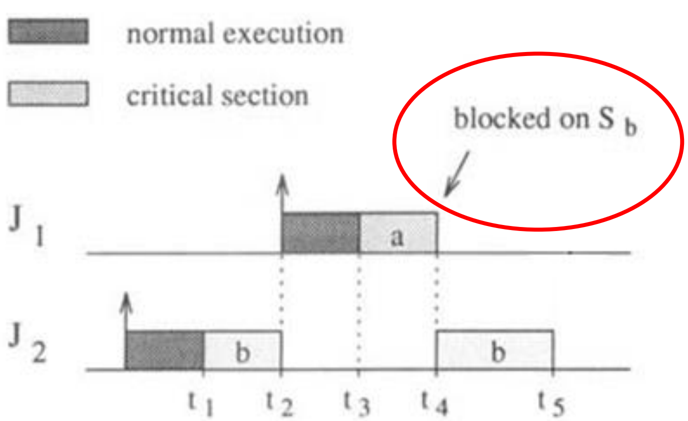
Remaining problem (2): Deadlock

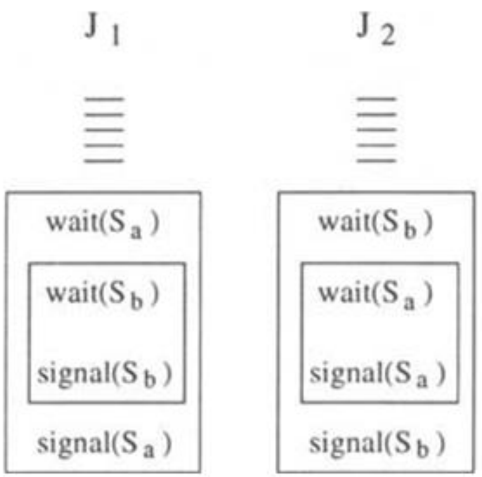


➔ Deadlock caused as J2 enters the nested critical session

Priority Ceiling Protocol

- Extends the Priority Inheritance Protocol
- Assign each semaphore a ceiling priority, equal to the priority of the highest-priority task that can lock it.
- Provided a critical section contains several semaphores, a job J can enter the critical section only when its priority is higher than all priority ceilings of the semaphores already locked by other jobs.





Protocol definition (1)

- S_k: an arbitrary semaphore
- $C(S_k)$: priority ceiling of S_k

$$C(S_k) \stackrel{\text{def}}{=} \max_i \{P_i \mid S_k \in \sigma_i\}$$

This value can be computed offline

- J_i: the job with the highest priority in ready queue
- P_i: the priority of J_i
- S^*: semaphore with the highest priority ceiling among all the semaphores currently locked by jobs other than J_i

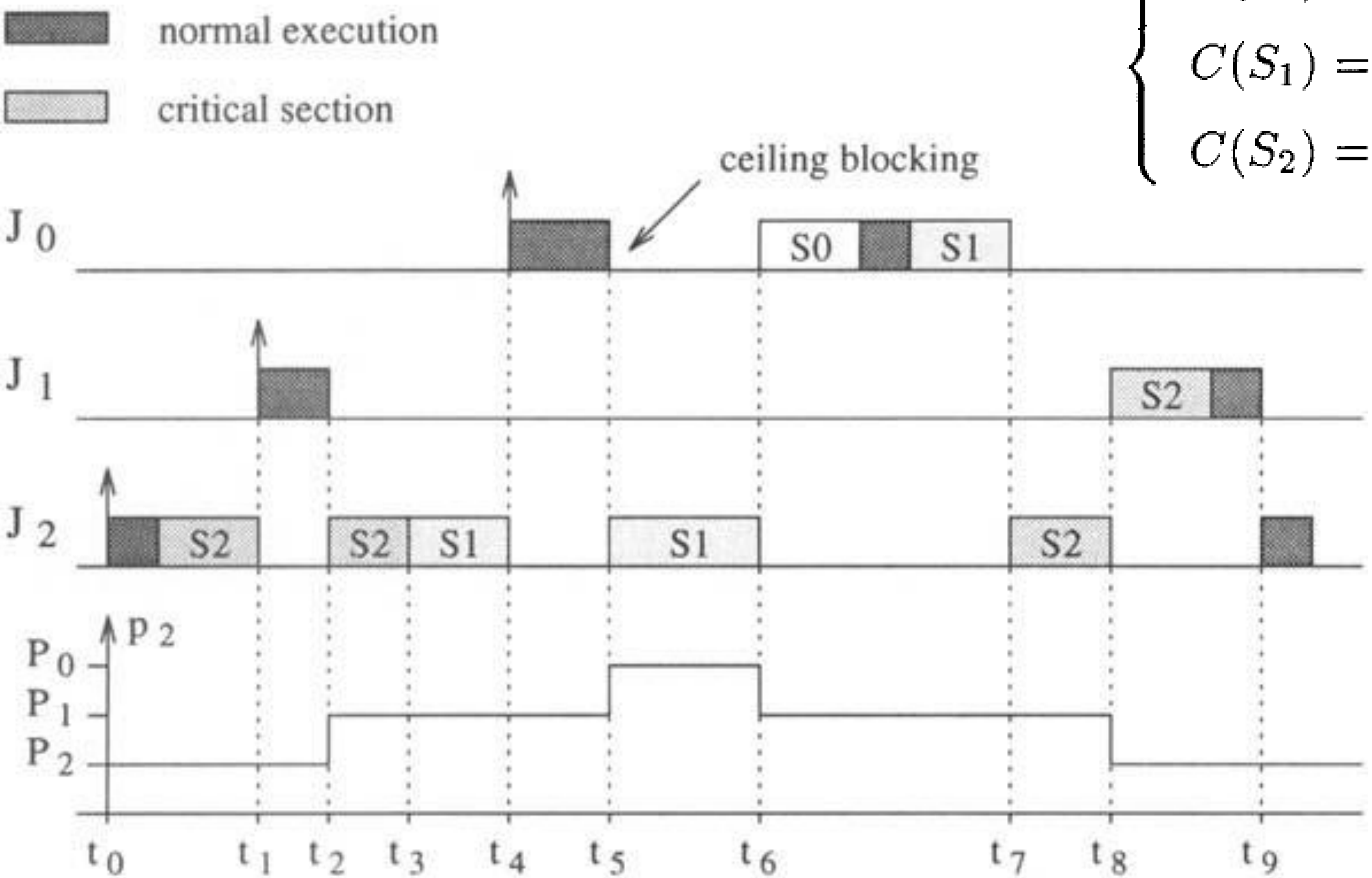
Protocol definition (2)

- When J_i is about to enter a critical section guarded by semaphore S_k,
  - If $P_i \leq C(S^*)$
    - locking on S_k is denied, &
    - J_i is blocked on semaphore S^* by the job holding the lock on S^*.
  - If $P_i > C(S^*)$
    - J_i locks on S_k *and continue execution*
- When J_i is blocked on a semaphore S,
  - The job J_k locking on S inherits the priority p_i
  - Generally, a task inherits the highest priority of the jobs blocked by it.
- When J_k exits a critical section & unlocks the semaphore,
  - If there are blocked jobs, then p_k is the highest active priority of the jobs blocked by J_k
  - Otherwise, p_k is restored to P_k

Example

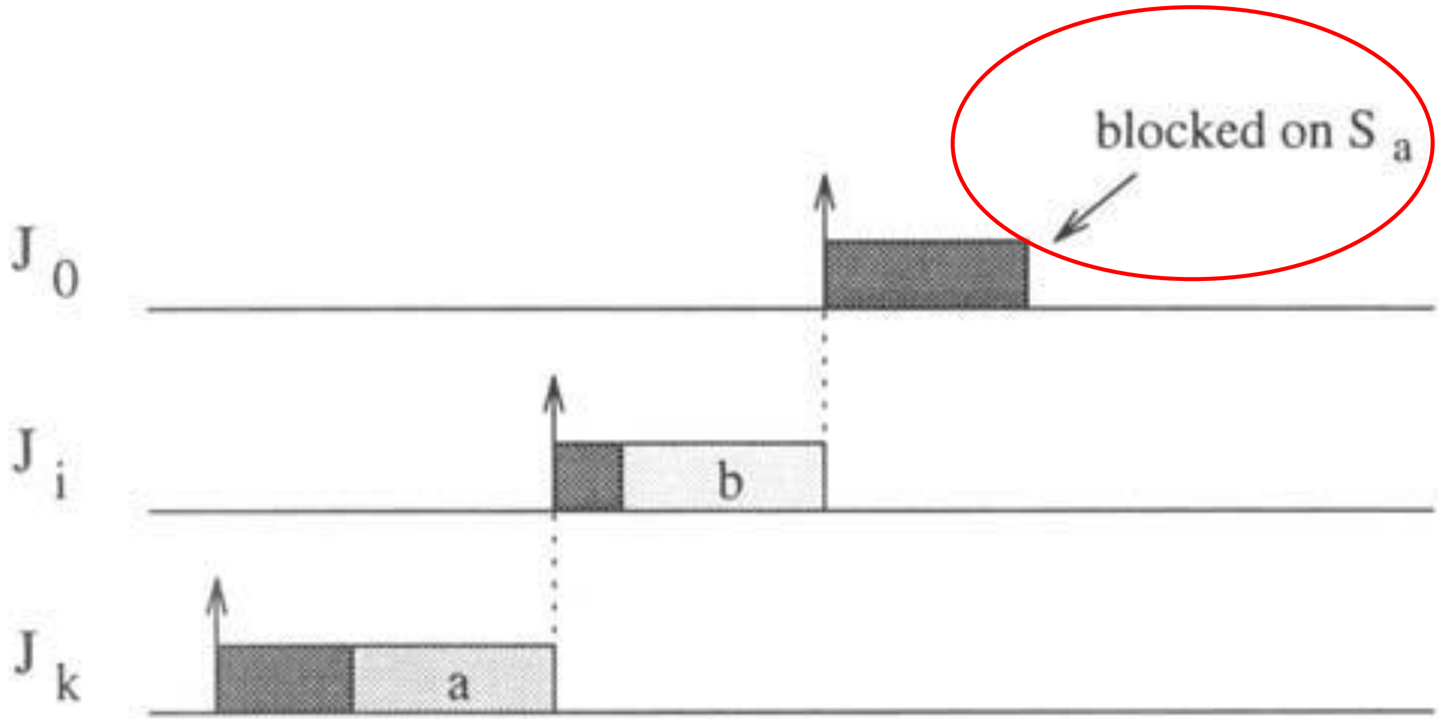
- Ceiling blocking

$$\begin{cases} C(S_0) = P_0 \\ C(S_1) = P_0 \\ C(S_2) = P_1 \end{cases}$$



Ceiling blocking

- A task is blocked by the protocol because of the priority ceiling condition
- Necessary to avoid chained blocking and deadlock



This will never happen with PCP

Properties of the protocol (2)

- The Priority Ceiling Protocol prevents deadlocks.
- Under the Priority Ceiling Protocol, a job J_i can be blocked for at most the duration of one critical section.

➔Reduce blocking time
➔Avoid unnecessary high-priority tasks blocking
➔Avoid deadlock

Comparison

	priority	Num. of blocking	pessimism	blocking instant	transpa-rency	deadlock preven-tion	implem-entation
NPP	any	1	high	on arrival	YES	YES	easy
HLP	fixed	1	medium	on arrival	NO	YES	easy
PIP	fixed	α_i	low	on access	YES	NO	hard
PCP	fixed	1	medium	on access	NO	YES	medium
SRP	any	1	medium	on arrival	NO	YES	easy

SRP (Stack Resource Protocol): for student's further reading