國立高雄大學
National University of Kaohsiung

漫談即時系統研究議題

國立高雄大學 資工系

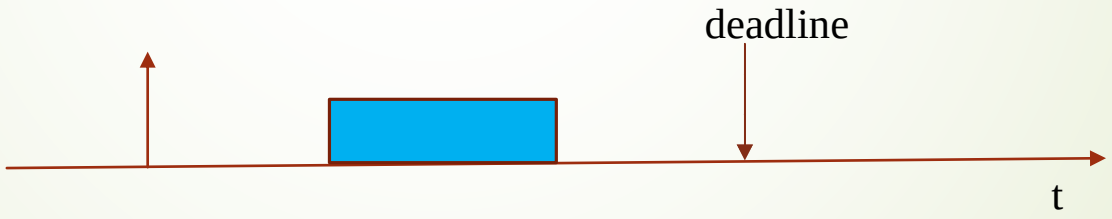
郭錦福

Agenda

- What are Real-time systems?
- System model
- Uniprocessor scheduling
  - Scheduling of Independent Periodic Tasks
  - Scheduling of Hybrid Task Set
  - Scheduling of Dependent Tasks
- Multiprocessor scheduling
- Other task models
- Conclusion

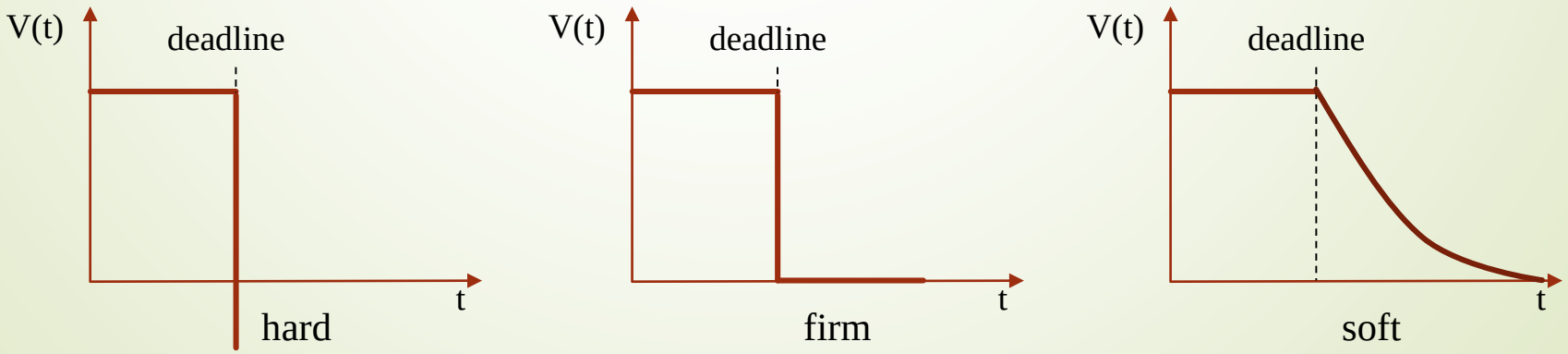
Real-Time Systems

- Tasks must be completed in time instead of early.
- Real-time systems are not high-performance systems.
- Real-time systems must provide performance guarantees with low hardware cost.



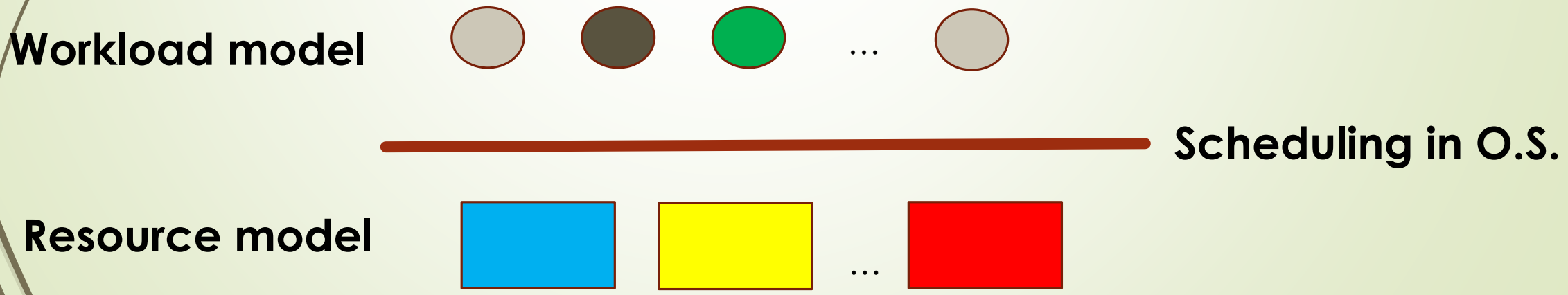
Three Kinds of Real-Time Deadline Constraints

- Hard real-time: deadline misses have catastrophic consequences
- Firm real-time: deadline misses must be limited, e.g., some quality of service must be defined
- Soft real-time (or not real-time): deadline misses are not important



Problem Abstraction

- Application system → System model
- System model
  - Workload model
  - Resource model



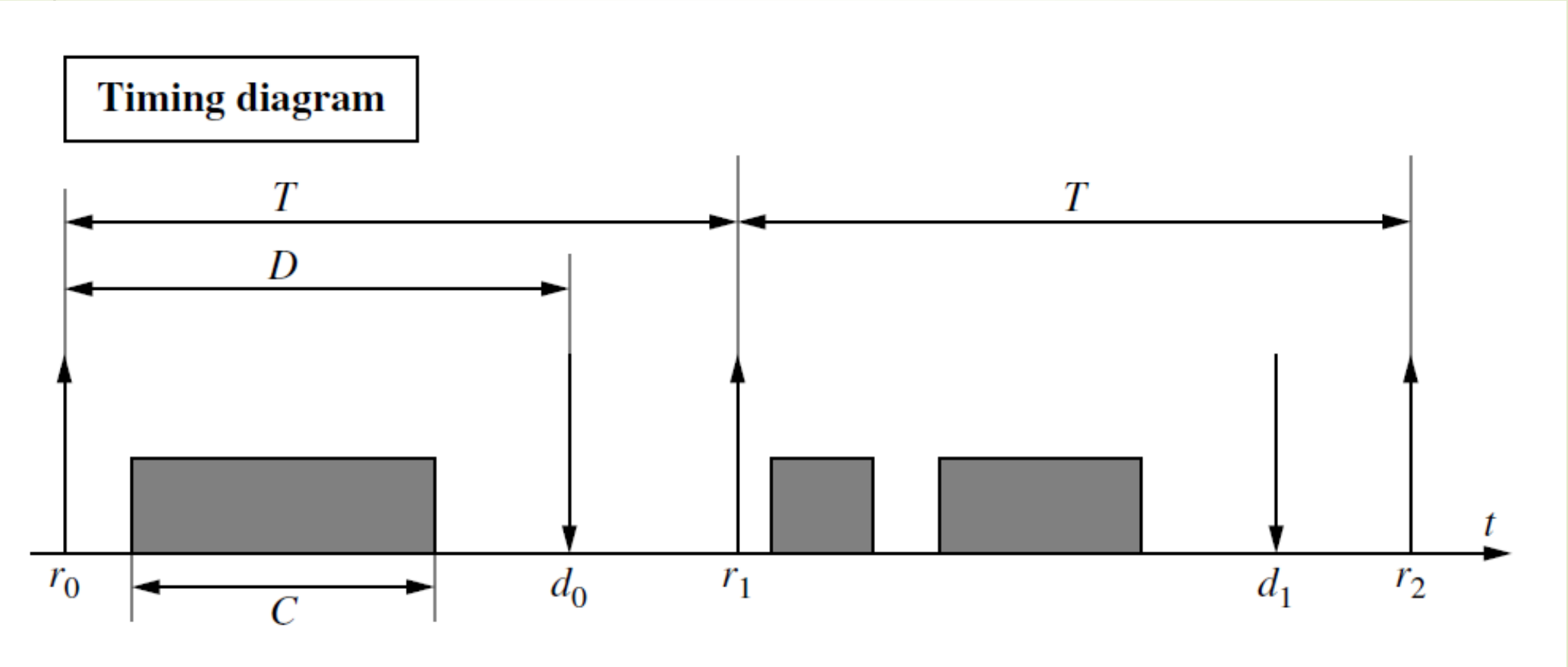
Workload Model

- Timing parameters of tasks
- Precedence constraints and dependencies among tasks
- Functional parameters of tasks
  - Preemptivity of jobs
  - Criticality of jobs
  - Optional executions
  - ….

Task Model

- Periodic, aperiodic, or sporadic task
- Independent or dependent task
- Tasks with varying timing parameters, such as execution time, period or deadline
  - Task with worst-case execution time and actual execution time
  - Task with variable execution time
    - Multiframe model: I frames, B frames, and P frames
- Preemptive or non-preemptive task

Periodic Task Model



Periodic Task Model

- A task model is defined with the main timing parameters.
- The model includes primary and dynamic parameters
  - r, task release time, i.e. the triggering time of the task execution request.
  - C: task worst-case computation time, when the processor is fully allocated to it.
  - D, task relative deadline, i.e. the maximum acceptable delay for its processing.
  - T, task period (valid only for periodic tasks).
  - When the task has hard real-time constraints, the relative deadline allows computation of the absolute deadline $d = r + D$.
    - Transgression of the absolute deadline causes a timing fault.
- Utilization of the task: $u = C/T$

Aperiodic Task and Sporadic Task

- Aperiodic tasks
  - Assigned to either no deadlines or soft deadlines
  - Response time
- Sporadic tasks
  - Assigned to hard deadlines
  - Admission control

Job vs. Task

- Job: Each unit of work scheduled and executed by the system
- Task: A set of related jobs which jointly provides a certain system function.
  - A task is defined as a sequence of related jobs.

Resource Model

- Processor
- Memory
- Network bandwidth
- Disk
- …

Processor Model

- Uniprocessor or multiprocessor
- Dynamic voltage scaling (DVS) or non-DVS processor
- Ideal or non-ideal processor
- Homogeneous or heterogeneous multiprocessor

Uniprocessor Scheduling

Uniprocessor Scheduling

- Scheduling of Independent Periodic Tasks
- Scheduling of Hybrid Task Set
- Scheduling of Dependent Tasks

Scheduling of Independent Periodic Tasks

Scheduling Algorithms

- Fixed-priority algorithm
- Dynamic-priority algorithm

Fixed-Priority Algorithms

- The priorities are assigned to tasks before execution and do not change over time.
  - Rate monotonic algorithm
  - Deadline monotonic algorithm

Rate Monotonic (RM) Algorithm

- For a set of periodic tasks, assigning the priorities according to the rate monotonic (RM) algorithm means that tasks with shorter periods (higher request rates) get higher priorities.
- RM is an optimal fixed-priority assignment algorithm.
  - If the schedule is feasible by an arbitrary priority assignment, then it is also feasible by applying the RM algorithm.

Schedulability Test of RM

- A sufficient schedulability condition
  - The relative deadline of each task is equal to its period.
  - $\sum_{i=1}^{n} \frac{C_i}{T_i} \leq n \cdot (2^{1/n} - 1)$
  - The upper bound converges to ln(2) = 0.69 for high values of n.
  - If the condition isn't satisfied, the task set "may" be not schedulable.

Deadline Monotonic (DM) Algorithm

- The task with the shortest relative deadline is assigned the highest priority.
- DM is an optimal fixed-priority assignment algorithm.

Schedulability Test of DM

- A sufficient schedulability condition
  - A task can have a relative deadline shorter than its period.
  - $\sum_{i=1}^{n} \frac{C_i}{T_i} \leq n \cdot (2^{1/n} - 1)$
  - The upper bound converges to ln(2) = 0.69 for high values of n.
  - If the condition isn't satisfied, the task set "may" be not schedulable.

Dynamic-Priority Algorithms

- Priorities are assigned to tasks based on dynamic parameters that may change during task execution.
  - Earliest deadline first (EDF)
  - Least laxity first (LLF)

Earliest Deadline First (EDF) Algorithm

- The earliest deadline first (EDF) algorithm assigns priority to jobs according to their absolute deadline.
  - The job with the earliest deadline will be executed at the highest priority.
- EDF is an optimal dynamic-priority assignment algorithm.
  - If there exists a feasible schedule for a task set, then the EDF algorithm is able to find it.

Schedulability Test of EDF

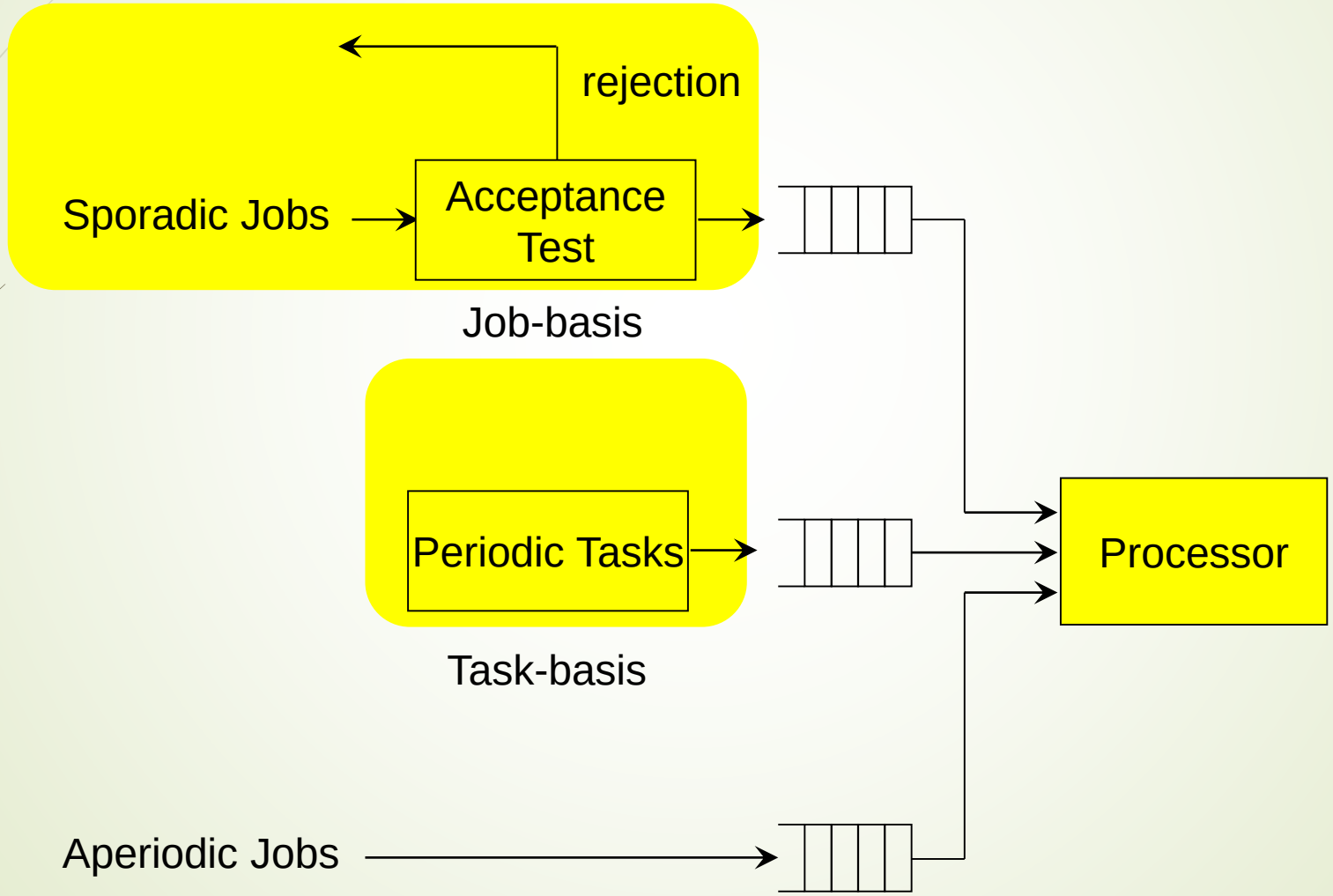
- A necessary and sufficient schedulability condition
  - The relative deadline of each task is equal to its period.
    - A set of periodic tasks is schedulable with the EDF algorithm if and only if $\sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1$
    - If the condition isn't satisfied, the task set must be not schedulable.
- A sufficient schedulability condition
  - The relative deadline of a task may be less than its period.
    - A set of periodic tasks is schedulable with the EDF algorithm if $\sum_{i=1}^{n} \frac{C_i}{D_i} \leq 1$
  - If the condition is not satisfied, the task set "may" be not schedulable.

Least Laxity First (LLF) algorithm

- The least laxity first (LLF) algorithm assigns priority to jobs according to their relative laxity.
  - The job with the smallest laxity will be executed at the highest priority.
  - The laxity of a job J_i
    - $d_i - t - rem_i$
- When the laxity of the tasks is computed only at arrival times, the LLF schedule is equivalent to the EDF schedule.
- However if the laxity is computed at every time, more context-switching will be necessary.
- LLF is optimal and the schedulability of a set of tasks can be guaranteed using the EDF schedulability test.

Scheduling of Hybrid Task Set

Hybrid Task Set



Handling Aperiodic Jobs

- Approaches
  - Background execution
  - Interrupt-driven execution
  - Polled execution
  - Bandwidth-preserving servers

Background Execution

- Aperiodic job are scheduled and executed only at times when
  - There is no periodic or sporadic job ready for execution.
- Pros:
  - Simple to implement.
  - Always produces correct schedules.
- Cons:
  - The execution of aperiodic jobs may be delayed.
  - Response times of aperiodic jobs prolonged unnecessarily.
    - Bad responses

Interrupt-Driven Execution

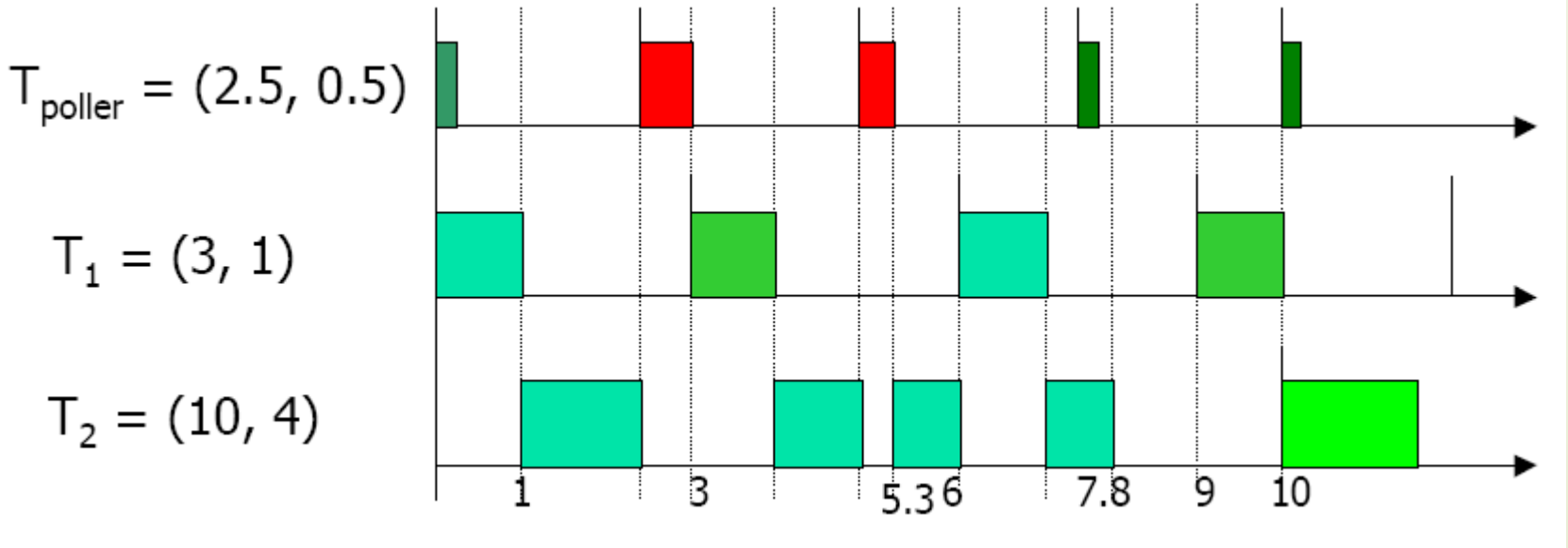
- Whenever an aperiodic job arrives
  - The execution of periodic tasks are interrupted (paused).
  - The aperiodic job is executed.
- The aperiodic job has a the shortest possible response time.
- Periodic tasks may miss some deadlines.

Polled Execution

- Polling server (poller)
  - Poller task (p_s, e_s) with polling period p_s and execution time (or budget) e_s.
  - Examines the aperiodic job queue.
    - If empty (idle), suspend immediately.
    - If not empty (backlogged), execute until
      - Executed for an interval e_s (consumes budget and exhausted), or
      - Queue become empty.
- Size of server: $u_s = {}^{e_s}/_{p_s}$.

Example of Polled Execution

- $W_A = 5.3 - 0.1 = 5.2$
- Aperiodic job arriving after exam queue empty: must wait until the next polling period.
  - Preserving the execution budget desirable.



TA = (0.1, -, 0.8)

Bandwidth Preserving Server Algorithm

- Improve the polling approach.
- Defined by a set of consumption and replenishment rules.
  - A backlogged bandwidth-reserving sever is ready for execution when it has budget.
    - The scheduler keeps track of the consumption of the server budget and suspends the server when the server budget is exhausted or the server becomes idle.
    - The scheduler moves the server back to the ready queue once it replenishes the server budget if the server is still backlogged at the time.
  - The server suspends itself whenever it finds the aperiodic job queue empty (becomes idle).
    - When the server becomes backlogged again upon arrival of an aperiodic job, the scheduler puts the server back to the ready queue if the server has budget at this time.

Three Bandwidth Preserving Servers

- *Deferrable servers*
- Sporadic servers
- Constant utilization/total bandwidth servers and weighted fair-queueing servers

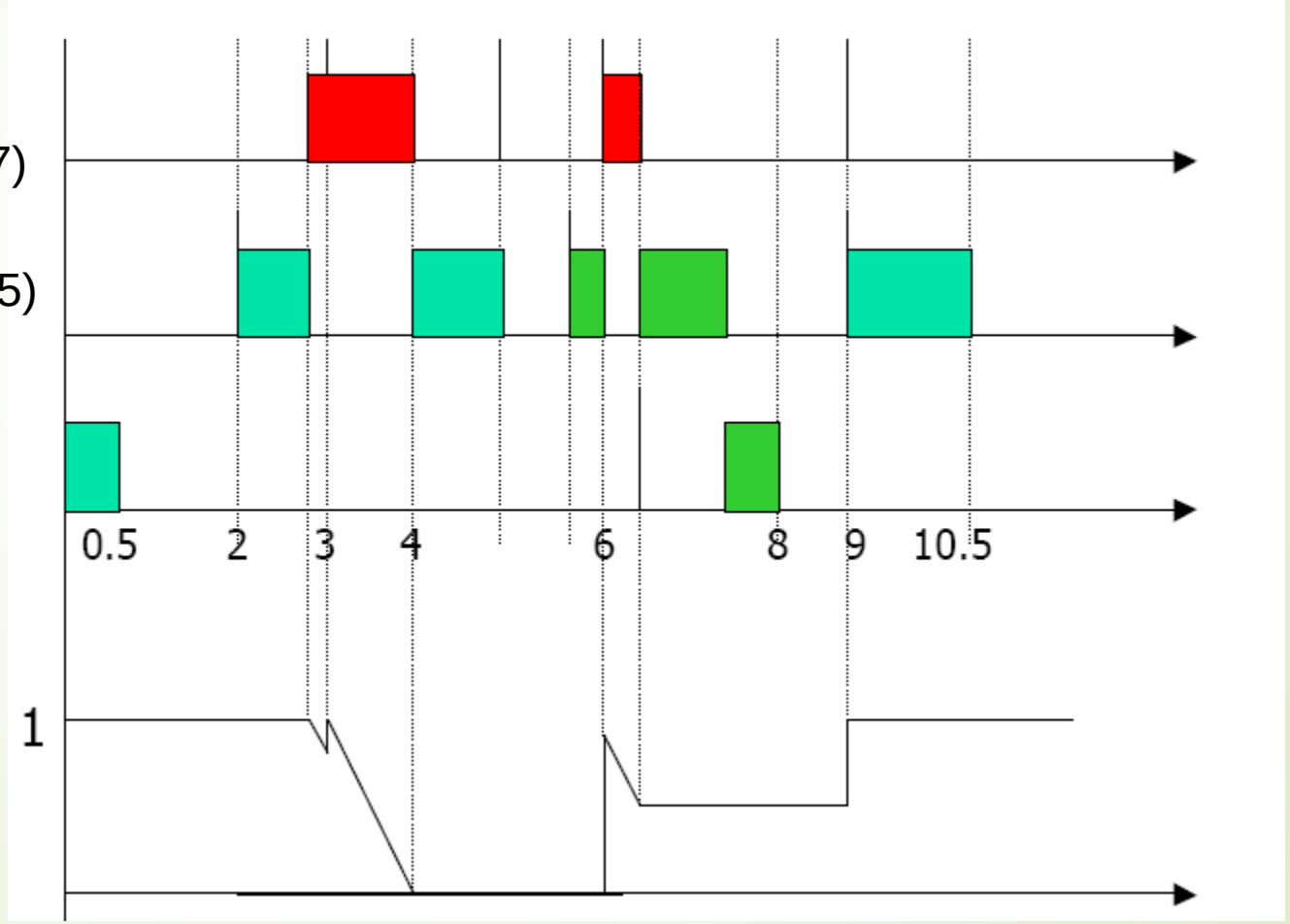
Deferrable Server

- Simplest bandwidth-preserving servers.
  - Execution budget is replenished periodically with period p_s.
  - When a deferrable server finds no aperiodic job ready for execution, it **preserves the budget**.
- Consumption rule
  - The execution budget of the server is consumed whenever the server executes.
- Replenishment rule
  - The execution budget of the server is set to e_s at the time instant $k \times p_s$, for $k = 0, 1, 2, \ldots$.
- Notes
  - The sever is not allowed to cumulate its budget from period to period.
  - Any budget held by the server immediately before each replenishment time is lost.

Example of Deferrable Server

- RM schedule

當有非週期性工作等待服務且還有budget時，
依據DS的週期排程。



Example of Deferrable Server

- EDF schedule

DS = (3, 1)

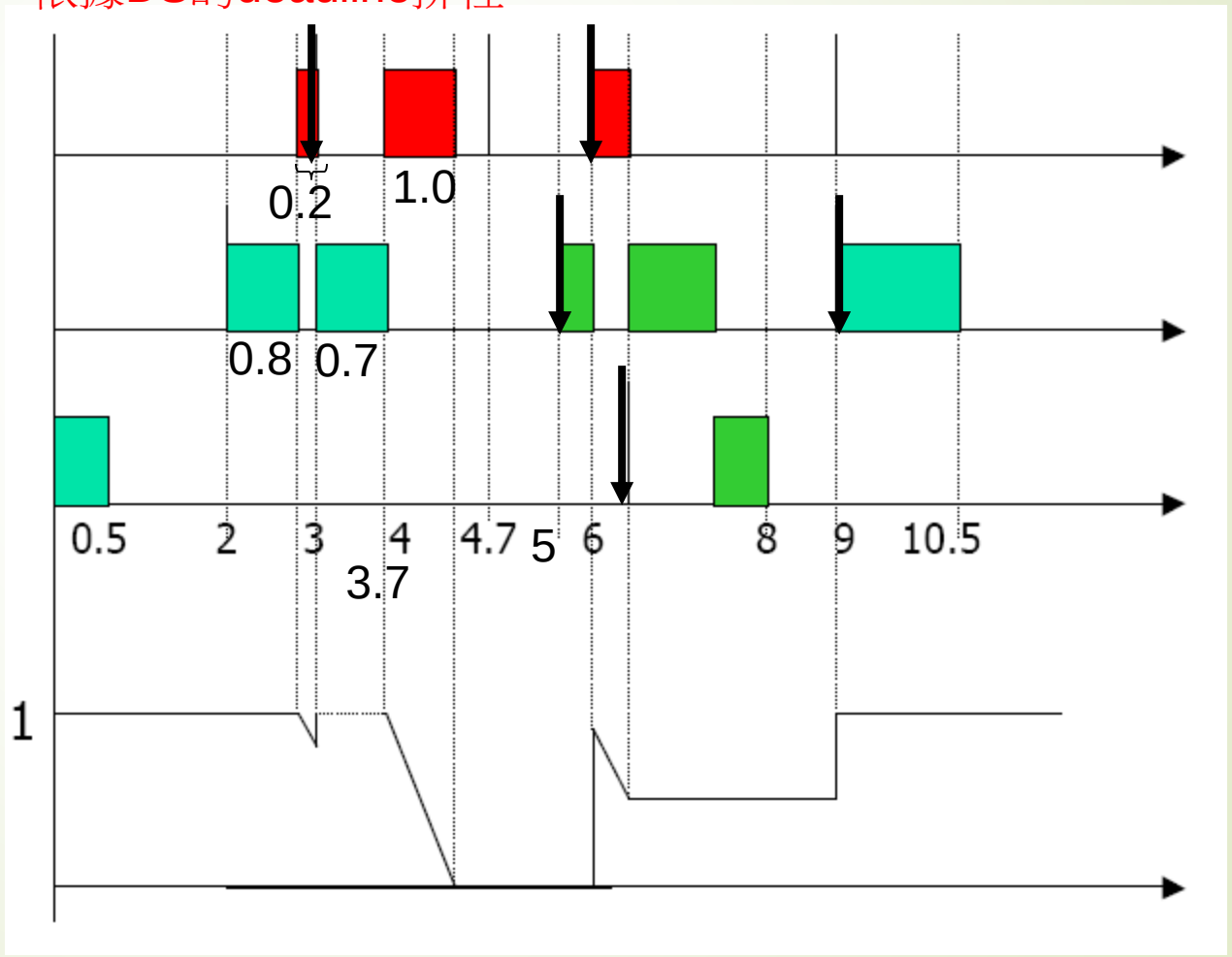
J_A = (2.8, -, 1.7)

T_1 = (2, 3.5, 1.5)

T_2 = (6.5, 0.5)

Budget

當有非週期性工作等待服務且還有budget時，
依據DS的deadline排程。



Deferrable Server

- Notes
  - Deferrable server: virtual periodic task.
  - Scheduling overhead: no higher than the poller.
- Additional background server
  - Whenever the budget is exhausted AND the processor is idle, the server is scheduled.
  - Responsiveness of the system can be further improved.
  - Ex) J_A can be executed in [4.7, 5.2].

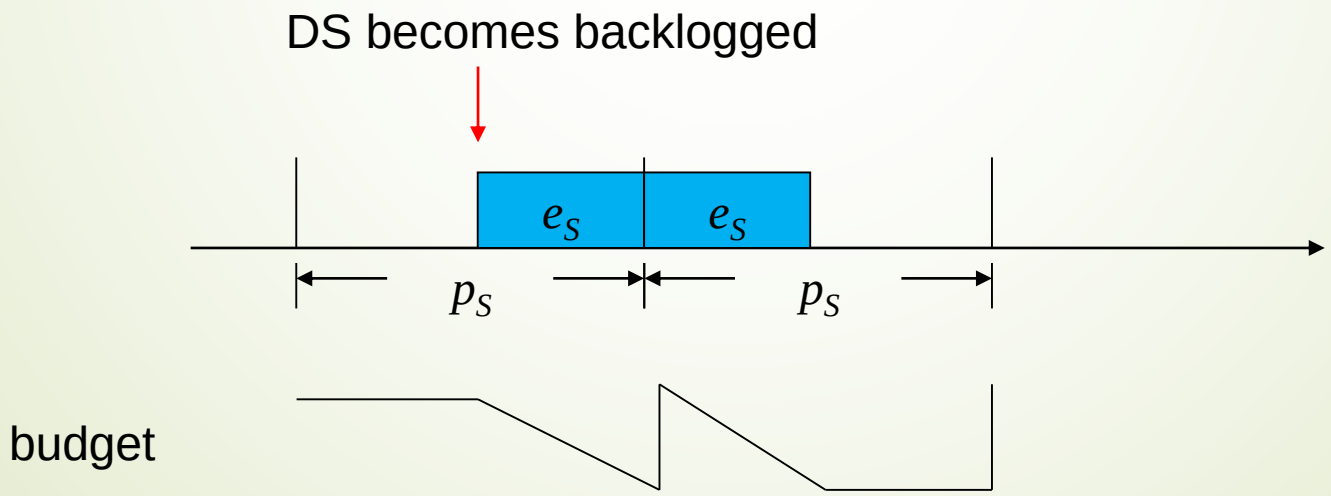
Deferrable Servers

- Because budget is preserved, aperiodic jobs may conflict with any periodic jobs at any time
  - Differently, jobs of a purely periodic task are ready at the beginning of every period

Deferrable Servers

- The below figure shows a scenario in which a deferrable server interferes low-priority periodic tasks at most

最糟情況下，會有連續的兩段budget長度的執行，這樣的影響不可讓週期性工作miss deadline

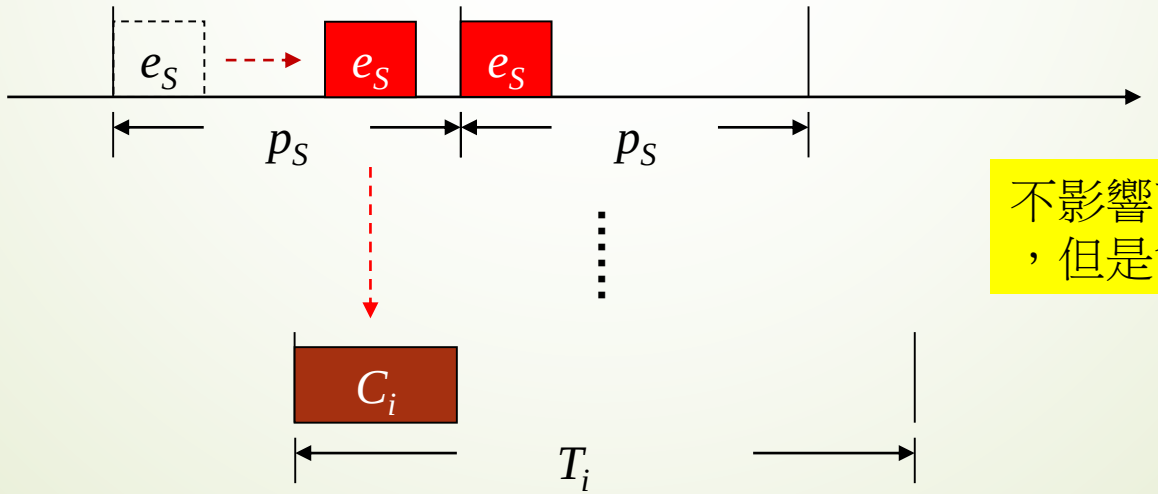


Schedulability of Systems Containing Deferrable Server

- Making the execution budget bigger may make unschedulable.
  - For RM, the schedulability test $\sum_{i=1}^{n} \frac{C_i}{T_i} + u_s \leq U(n+1)$ can not be applied.
  - For EDF, the schedulability test $\sum_{i=1}^{n} \frac{C_i}{T_i} + u_s \leq 1$ can not be applied.

Deferrable Server

- If the deferrable server τ_{DS} is of an arbitrary priority, it may impose up to one additional e_s on feasible intervals of jobs of some low-priority task τ_i



不影響高優先權的任務，但是會影響低優先權者

Deferrable Server (Fixed-Priority)

- When the server's period is arbitrary.
  - Server behaves just like a periodic task (p_s,e_s).
  - An additional e_s in the feasible interval: additional blocking time of τ_i.
- Task set $\{\tau_1,\ldots, \tau_m, \tau_{DS}, \tau_{m+1},\ldots, \tau_n\}$
  - $\{\tau_1,\ldots, \tau_m\}$ is schedulable if $\sum_{j=1}^{m} \frac{C_j}{T_j} \leq U(m)$
  - For i=m, $\sum_{j=1}^{m} \frac{C_j}{T_j} + \frac{e_s}{p_s} \leq U(i+1)$
  - For i=m+1~n, the reminder of the task set is schedulable if

$$\sum_{j=1}^{i} \frac{C_j}{T_j} + \frac{e_s}{p_s} + \frac{e_s}{T_i} \leq U(i+1)$$

Deferrable Server (Deadline-Driven)

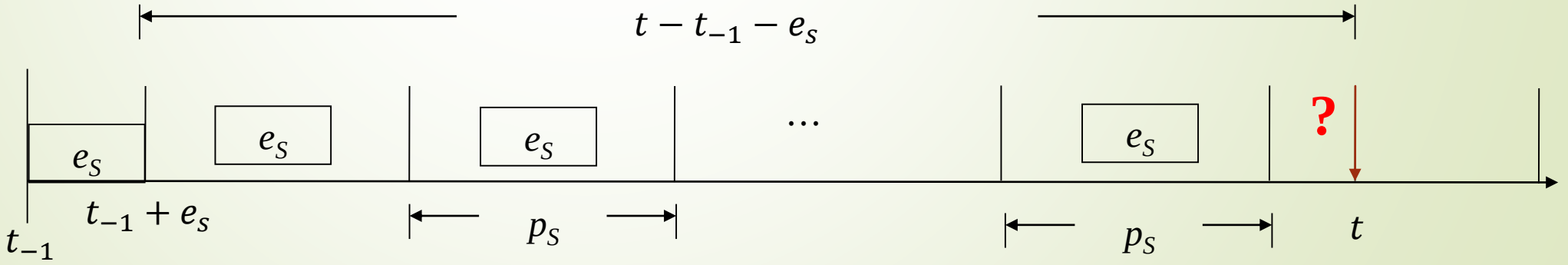
- Schedulability of a *deadline-driven* system with a deferrable server
  - Consider task set $\{\tau_1, \ldots, \tau_n\} \cup (p_s, e_s)$
    - Let t be the deadline of a job of a periodic task
    - Let t_{-1} $(< t)$ be the latest time instant *when the CPU is idle or is executing jobs with deadline later than t*
    - Consider time interval $(t_{-1}, t]$, we are interested in the maximum computation time demand from periodic tasks and the deferrable server

Schedulability of Deferrable Server

- Total amount $w_{DS}(t - t_{-1})$ of processor time consumed by deferrable server in $(t_{-1}, t]$is at most equal to

$$e_s + \left\lfloor \frac{t - t_{-1} - e_s}{p_s} \right\rfloor \times e_s$$

Why floor?



The last job of e_s does not affect jobs of deadline no later than t due to EDF discipline.

Schedulability of Deadline-Driven Systems (cont'd)

- Since floor(x) <= x and $u_s = \frac{e_s}{p_s}$:

$$w_{DS}(t - t_{-1}) = e_s + \left\lfloor \frac{t - t_{-1} - e_s}{p_s} \right\rfloor \times e_s$$

$$\leq e_s + \frac{t - t_{-1} - e_s}{p_s} e_s = u_s(t - t_{-1} + p_s - e_s)$$

$e_s = p_s * u_s$

Schedulability of Deadline-Driven Systems (cont'd)

- **Thm.** A periodic task τ_i in a system of n independent, preemptive periodic tasks is schedulable with a deferrable server with period p_s, execution budget e_s, and utilization u_s, according to the EDF algorithm **if**

$$\sum_{k=1}^{n} \frac{e_k}{T_k} + u_s \left(1 + \frac{p_s - e_s}{D_i}\right) \leq 1$$

- Notes
  - Deferrable server behaves like a periodic task (p_s, e_s).
  - Extra amount of time as blocking: only $(p_s - e_s)u_s$.

Sporadic Servers

- **Deferrable server**
  - May delay lower-priority tasks for more time than a periodic task with the same period and execution time.
    - *they may introduce that periodic tasks won't meet their deadlines*
      - Both in priority-driven and deadline-driven scheduling
  - Schedulability test for periodic systems with deferrable servers becomes quite complicated
- **Sporadic server**
  - Improved: Each sporadic server with period p_s and budget e_s never demands more processor time than the periodic task (p_s, e_s) in any time interval.
  - For the schedulability, we can treat the sporadic server like a periodic task (p_s, e_s).
  - Extends schedulability of deferrable server.
  - However, nothing always wins. Sporadic servers have ***complicated consumption and replenishment rules***.
  - Different sporadic servers with different consumption and replenishment rules.

Sporadic Servers

- Design guideline: a sporadic server never demand processor time more than a periodic task (p_s, e_s) does in any time interval
  - Sporadic servers can be replaced with corresponding periodic tasks for schedulability test
  - For RM, use the schedulability test $\sum_{i=1}^{n} \frac{C_i}{T_i} + u_s \leq U(n+1)$
  - For EDF, use the schedulability test $\sum_{i=1}^{n} \frac{C_i}{T_i} + u_s \leq 1$
- A system maybe schedulable with a sporadic server when it is unschedulable with a deferrable server

Sporadic Servers

- A sporadic server can also be characterized by (p_s, e_s)
  - Note again that a sporadic server is not a periodic task!

- Different designs of sporadic servers exist, complicated rules for sporadic servers aim at
  - *preserving server budget for a longer period of time and*
  - *replenishing server budget more aggressively (i.e., earlier)*

Three Bandwidth Preserving Servers

- Deferrable servers
- Sporadic servers
- *Constant utilization/total bandwidth servers*

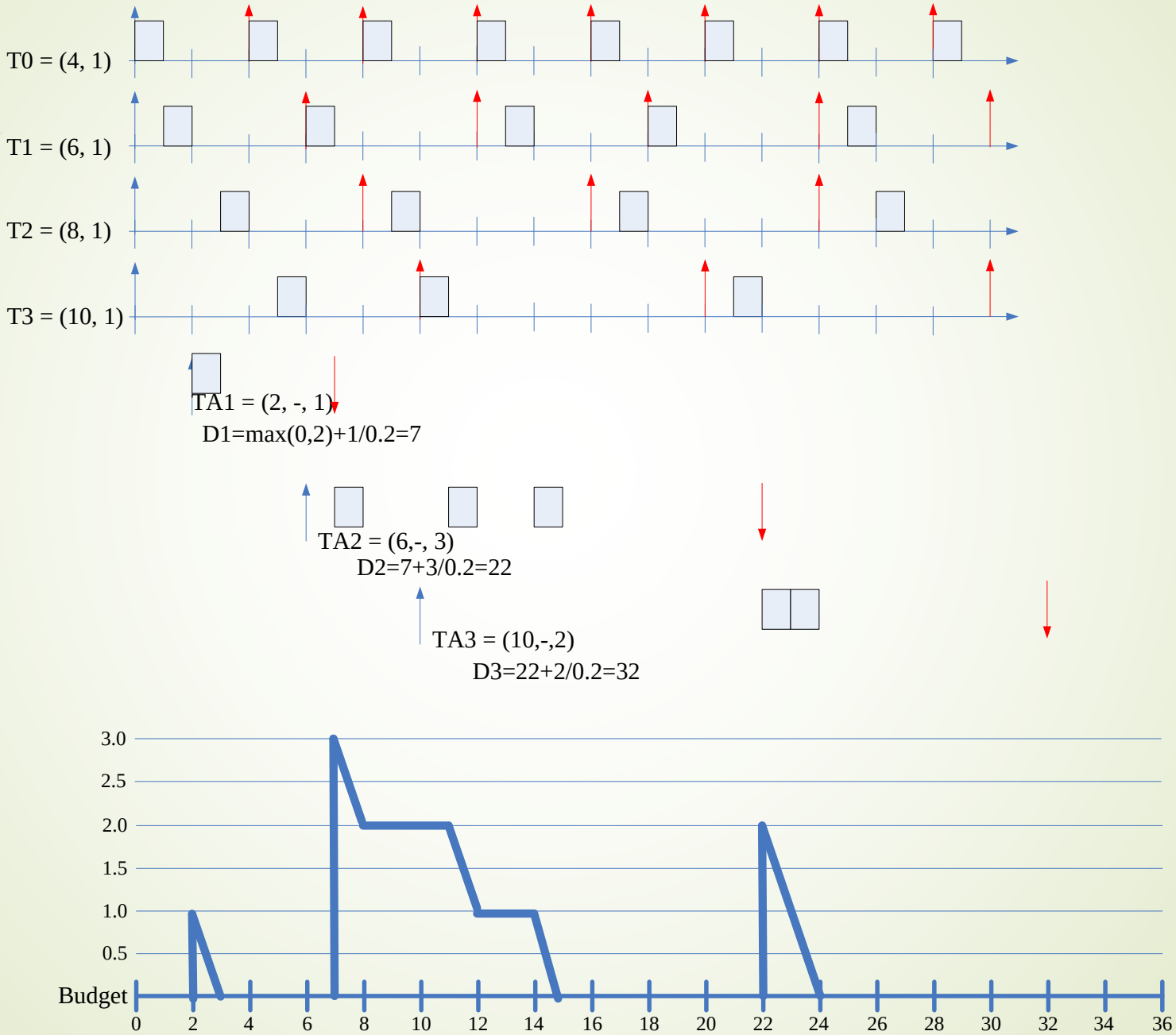
Constant Utilization Server Algorithm

- Scheduling aperiodic jobs amid periodic tasks in a deadline-driven system.
- Size of server: instantaneous utilization u_s
  - Fraction of processor time reserved for the execution of aperiodic jobs.
- **Constant utilization server**
  - The deadline d of server is always defined.
  - The server is eligible and ready for execution only when its budget is nonzero.
  - When the server is ready, it is scheduled with periodic tasks on the EDF basis.
  - Emulates a sporadic task with a constant instantaneous utilization.
    - Schedulability test $\sum_{i=1}^{n} \frac{C_i}{T_i} + u_s \leq 1$

Consumption and Replenishment Rules of Constant Utilization Server

- Consumption rules of CU server
  - A server consumes its budget only when it executes.
    - Never has any budget when there is no ready aperiodic job.
- Replenishment rules of CU server of size u_s
  - **R1**. Initially, $e_s = 0$, and $d = 0$.
  - **R2**. When an aperiodic job with execution time e arrives at time t to an empty aperiodic job queue,
    - (a) if $d > t$, do nothing;
    - (b) if $t \geq d$, $d = t + \frac{e}{u_s}$, and $e_s = e$.
  - **R3**. At the deadline d of server,
    - (a) if the server is **backlogged**, set the server deadline to $d + \frac{e}{u_s}$, and $e_s = e$.
    - (b) if the server is idle, do nothing.
- Always given enough budget to complete the ready job each time its budget is replenished.

CUS


T0 = (4, 1)
T1 = (6, 1)
T2 = (8, 1)
T3 = (10, 1)
TA1 = (2, -, 1)
D1=max(0,2)+1/0.2=7
TA2 = (6,-, 3)
D2=7+3/0.2=22
TA3 = (10,-,2)
D3=22+2/0.2=32
3.0
2.5
2.0
1.5
1.0
0.5
Budget
0
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
32
34
36

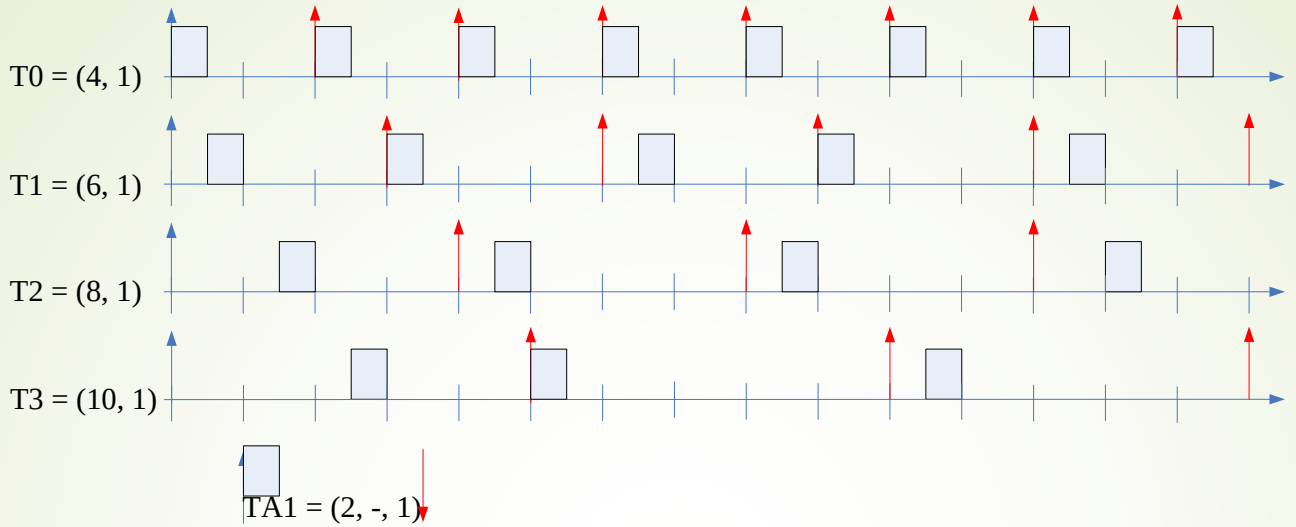
Total Bandwidth Servers

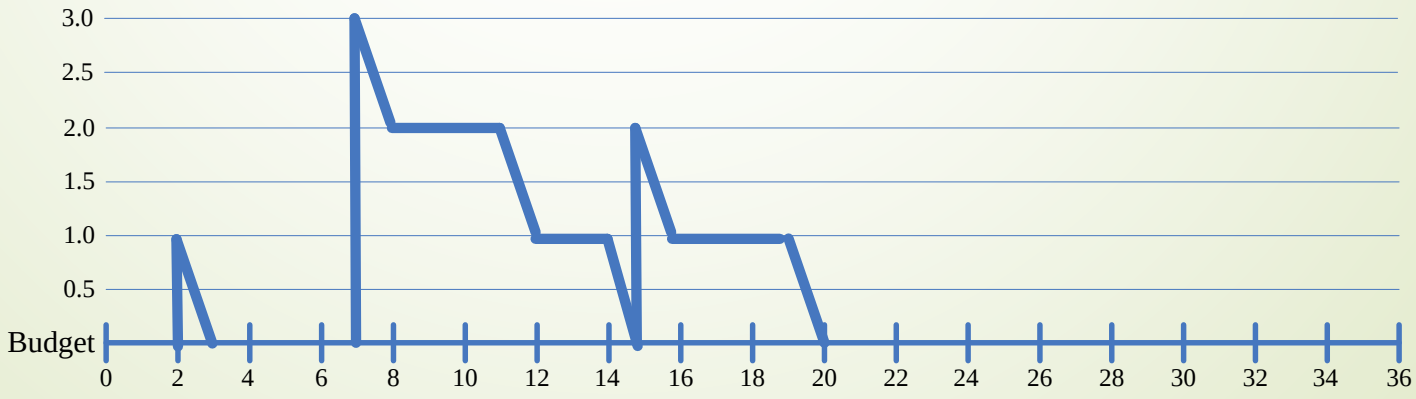
- CUS replenish budget on deadlines of jobs only
  - Aperiodic jobs arrive before an upcoming server deadline will be pended until budget is replenished at the deadline
- Is it possible to advance the replenishment?
  - Yes: total bandwidth servers
- The total bandwidth server algorithm
  - Improves the responsiveness of a constant utilization server by allowing the server to claim the background time not used by periodic tasks.
  - Aggressive algorithm.

Consumption and Replenishment Rules of Total Bandwidth Server

- Replenishment rules of TB server of size u_s
  - **R1**. Initially, $e_s = 0$, and $d = 0$.
  - **R2**. When an aperiodic job with execution time e arrives at time t to an empty aperiodic job queue, set d to $\max(d, t) + \frac{e}{u_s}$, and $e_s = e$.
  - **R3**. When the server completes the current aperiodic job,
    - (a) if the server is backlogged, set the server deadline to $d + \frac{e}{u_s}$, and $e_s = e$.
    - (b) if the server is idle, do nothing.

TBS


T0 = (4, 1)
T1 = (6, 1)
T2 = (8, 1)
T3 = (10, 1)
TA1 = (2, -, 1)
D1=max(0,2)+1/0.2=7
TA2 = (6,-, 3)
D2=7+3/0.2=22
TA3 = (10,-,2)
D3=22+2/0.2=32


3.0
2.5
2.0
1.5
1.0
0.5
Budget
0
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
32
34
36

Scheduling of Dependent Tasks

Tasks with Dependences

- There are two kinds of typical dependencies that can be specified on real-time tasks:
  - Precedence constraints that correspond to synchronization or communication among tasks;
  - Mutual exclusion constraints to protect shared resources.
    - These critical resources may be data structures, memory areas, external devices, registers, etc.

Tasks with Precedence Relationships

- A precedence constraint between two tasks τ_i and τ_j, is denoted by $\tau_i \rightarrow \tau_j$, if
  - The execution of task τ_i precedes that of task τ_j.
  - In other words, task τ_j must await the completion of task τ_i before beginning its own execution.

RM Algorithm for Precedence Constraints

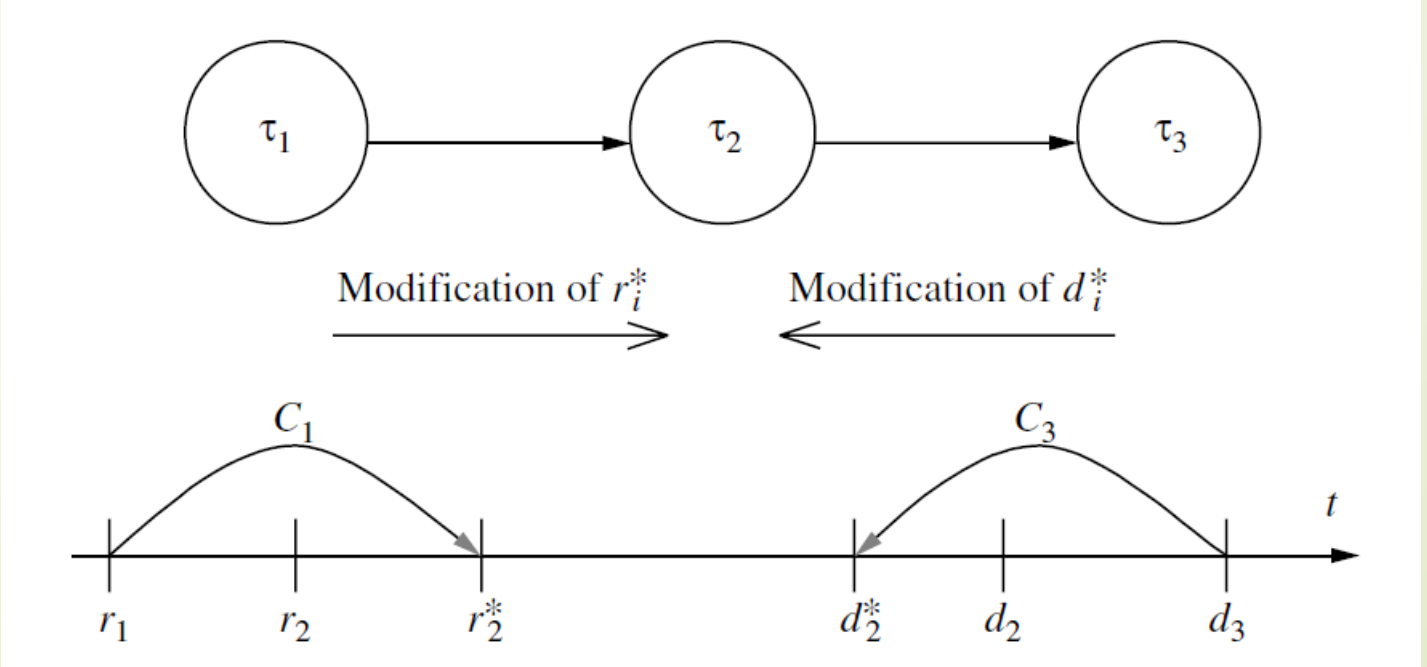
- Simple precedence constraint:
  - The execution of the kth instance of task τ_i precedes the execution of the k th instance of task τ_j.
    - These tasks have the same rate (i.e. $T_i = T_j$).
- Transform precedence constraint $\tau_i \rightarrow \tau_j$ into timing constraint for RM
  - the task parameters must be in accordance with the following rules:
    - $r_j^* \geq \max(r_j, r_i^*)$, r_i^* is the modified release time of task τ_i
    - $Prio_i \geq Prio_j$ in accordance with the RM algorithm

DM Algorithm for Precedence Constraints

- Simple precedence constraint:
  - The execution of the kth instance of task τ_i precedes the execution of the k th instance of task τ_j.
    - These tasks have the same rate (i.e. $T_i = T_j$).
- Transform precedence constraint $\tau_i \rightarrow \tau_j$ into timing constraint for DM
  - the task parameters must be in accordance with the following rules:
    - $r_j^* \geq \max(r_j, r_i^*)$, r_i^* is the modified release time of task τ_i
    - $D_j^* \geq \max(D_j, D_i^*)$, D_i^* is the modified relative deadline of task τ_i
    - $Prio_i \geq Prio_j$ in accordance with the DM algorithm

EDF Algorithm for Precedence Constraints

- If there exists $J_i \rightarrow J_j$ between jobs J_i and J_j,
  - $r_j^* \geq \max(r_i^* + C_i, r_j)$
  - $d_i^* \geq \min(d_j^* - C_j, d_i)$

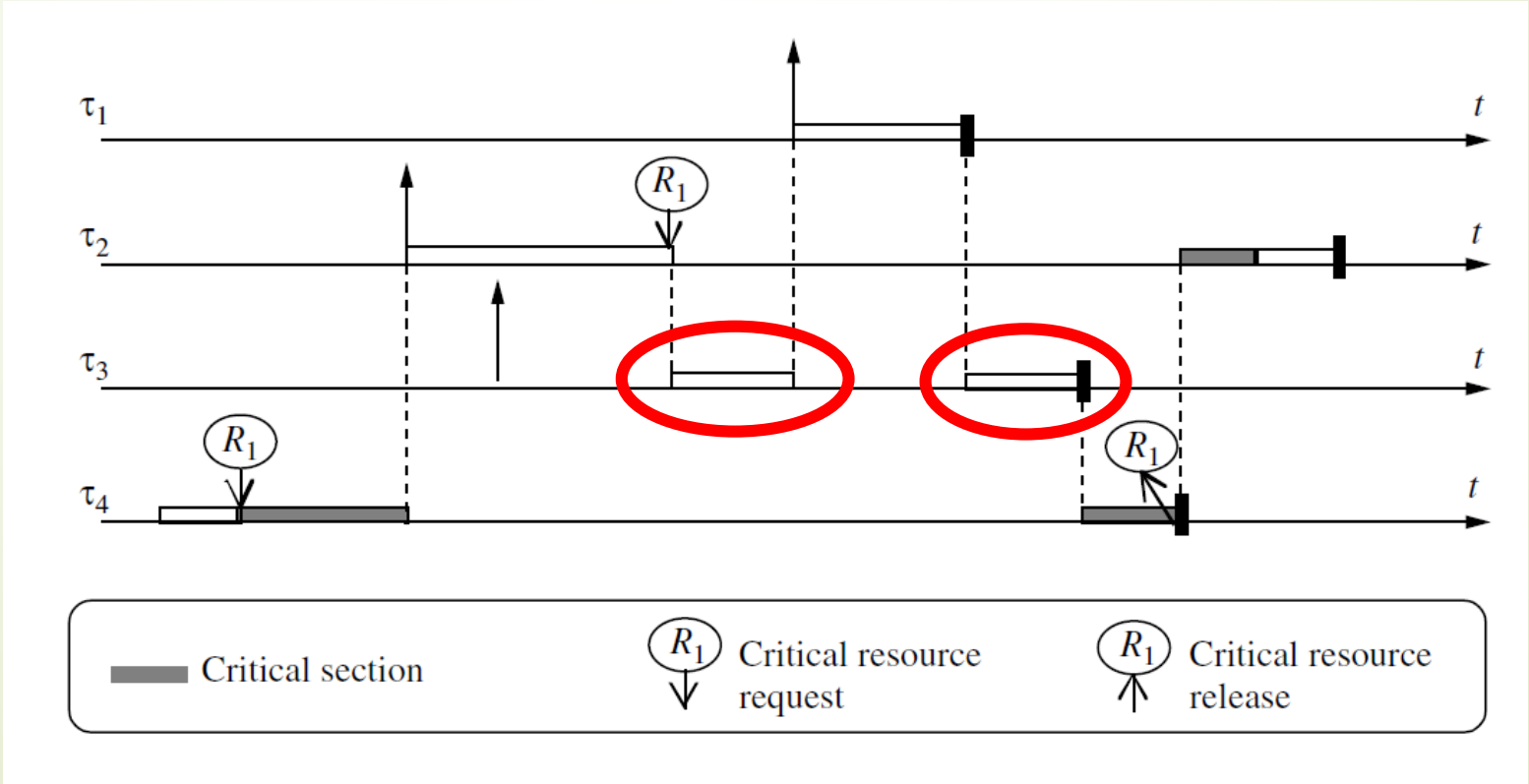


Tasks Sharing Critical Resources

- When tasks are allowed to access shared resources, their access needs to be controlled in order to maintain data consistency.
- Assume a critical resource, called R, shared by two tasks τ_1 and τ_2.
  - Ensure that the sequences of statements of τ_1 and τ_2, which perform on R, are executed under mutual exclusion.
  - These pieces of code are called critical sections.
  - It is important to note that the execution is in a non-preemptive context.

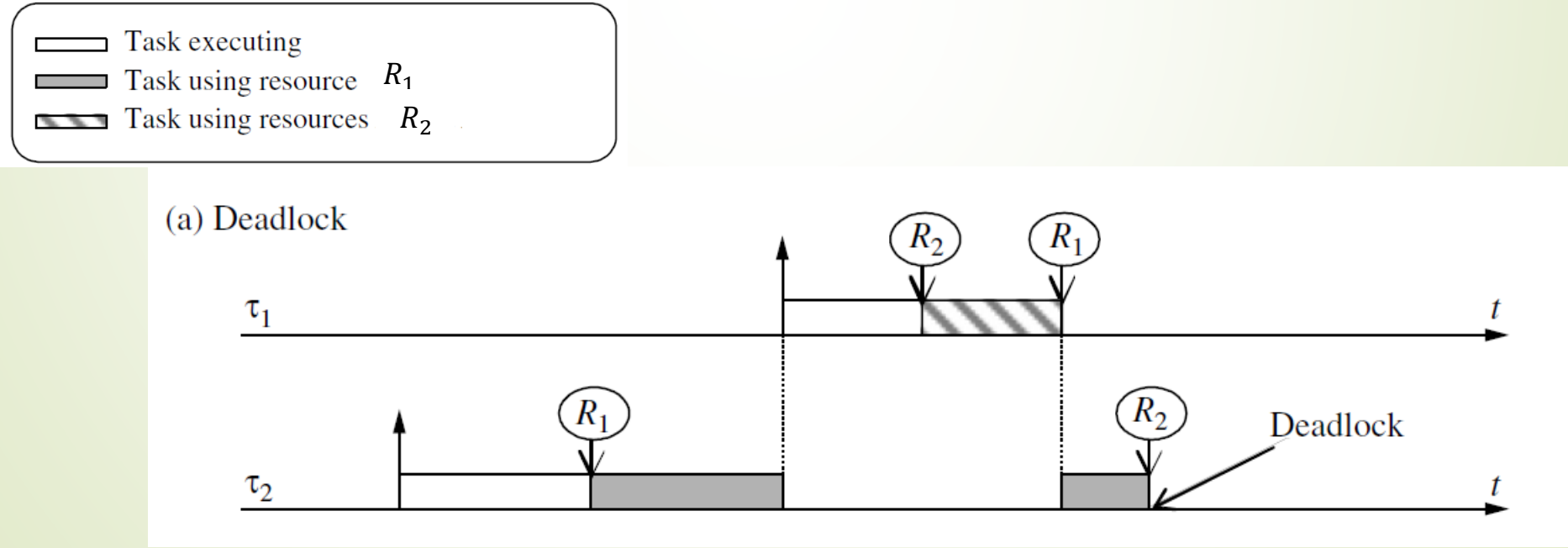
Priority Inversion Phenomenon

- A higher-priority job can be blocked by a lower-priority job to which the resource is granted.



Deadlock Phenomenon

- When tasks share the same set of two or more critical resources, then a deadlock situation can occur and, as a consequence, the real-time application fails.



Resource Access-Control Protocols

- Non-Preemptible Critical Section
- Basic Priority-Inheritance Protocol
- Basic Priority-Ceiling Protocol

Non-Preemptive Critical Sections

- Nonpreemptive Critical Section (NPCS) protocol
  - Simplest control of resource access.
  - Schedule all critical sections nonpreemptively.
  - No deadlock.
  - Blocking time due to resource conflict in a fixed-priority/deadline-driven system:
    - $B = Max_{i+1 \leq k \leq n} CR$

Non-Preemptive Critical Sections

- Adv:
  - Simple: No prior knowledge of resource requirements.
  - Simple to implement for RM/EDF systems.
  - Useful when short critical sections and much conflicts.
- Disadv:
  - Every job without resource request can be blocked by lower-priority jobs with resource requirements.

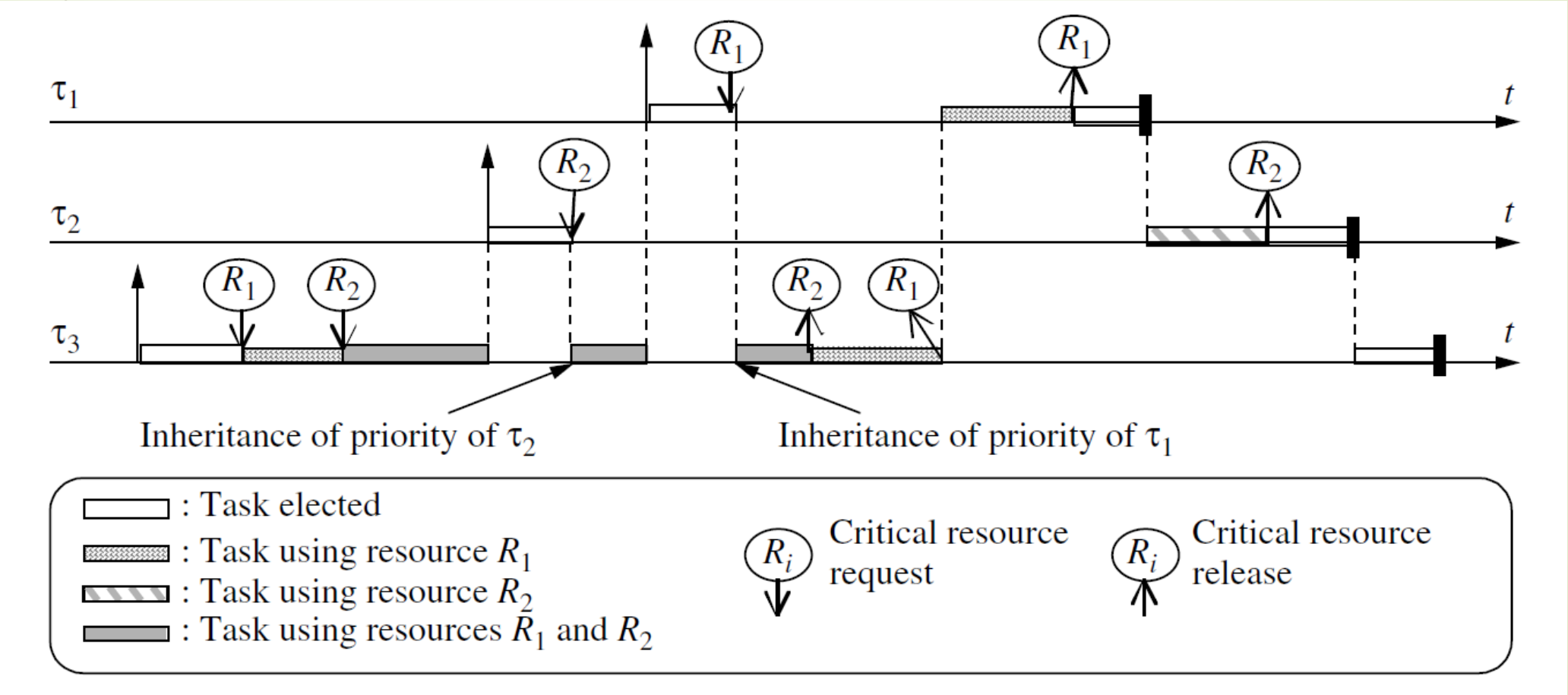
Basic Priority-Inheritance Protocol

- Features:
  - Works with any preemptive, priority-driven scheduling algorithm.
  - Does not require prior knowledge on resource requirements.
  - Does not prevent deadlock.
  - Uncontrolled priority inversion cannot occur.
  - Basic version: Only 1 unit for each resource.
- Notations
  - Assigned priority: Job priority assigned by the scheduling algorithm.
  - Inherited priority: Priority $Prio_l$ of J_l raised to $Prio_h$.
  - Current priority: time varying priority at the current time.

Rules of Basic Priority-Inheritance Protocol

1. Scheduling Rule:
   - Ready jobs are scheduled on the processor preemptively in a priority driven manner according to their current priorities.
   - At its release time t, the current priority $Prio(t)$ of every job J is equal to its assigned priority.
   - The job remains at this priority except under the condition of Rule 3.
2. Allocation Rule:
   - When a job J requests a resource R at time t :
     - If R is free, R is allocated to J until J releases the resource.
     - If R is not free, the request is denied and J is blocked.
3. Priority-Inheritance Rule:
   - When the requesting job J becomes blocked, the job J_l which blocks J inherits the current priority $Prio(t)$ of J.
   - The job J_l executes at its inherited priority $Prio(t)$ until it releases R; at that time, the priority of J_l returns to its priority $Prio(t')$ at the time t' when it acquires the resource R.

Example of PIP



Blocking Time of PIP

- Under PIP, a task τ_1 can be blocked at most by n critical sections of lower priority tasks or by m critical sections corresponding to resources shared with lower priority tasks.
  - $B = \max(m, n) \times CR_{max}$

Basic Priority-Ceiling Protocol

- Two assumptions on priority–ceiling protocol
  - The assigned priorities of all jobs are fixed.
  - The resource required by all jobs are known a priori before the execution of any job begins.
- Terms
  - Priority ceiling $\pi(R_i)$ of resource R_i:
    - The highest priority of all the jobs that require R_i.
  - The [current priority] ceiling $\hat{\pi}(t)$:
    - The highest priority ceiling of the resources that are in use at the time t.
    - Ω (non-existing lowest priority) when all the resources are free.

Definition of the Basic Priority-Ceiling Protocol

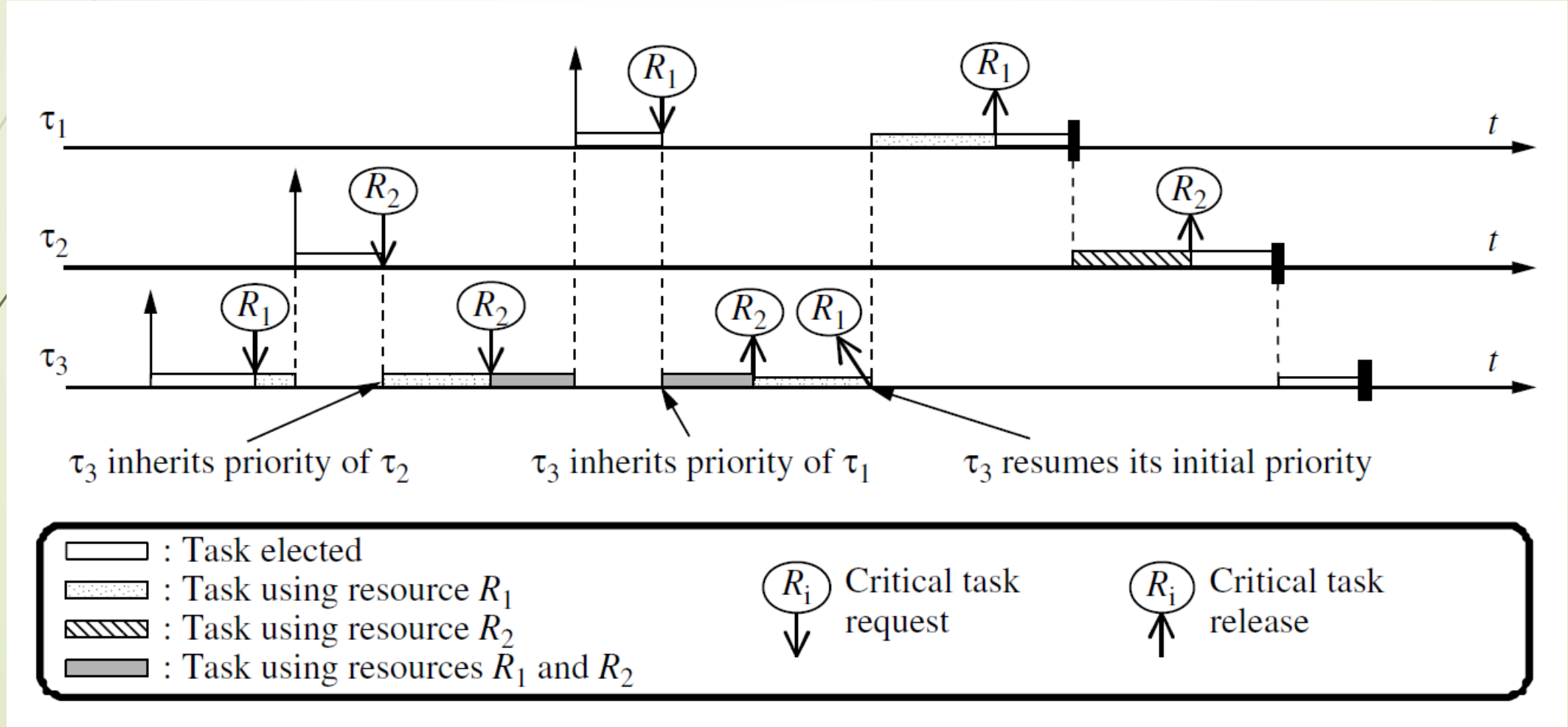
- Rules of Basic Priority-Ceiling Protocol (I)
  1. Scheduling Rule:
     - At its release time t, the current priority $Prio(t)$ of every job J is equal to its assigned priority. The job remains at this priority except under the condition in rule 3.
     - Every ready job J is scheduled preemptively and in a priority-driven manner at its current priority $Prio(t)$.
  2. Allocation Rule: Whenever a job J requests a resource R at time t, one of the following two conditions occurs:
     - R is held by another job. J's request fails and J becomes blocked.
     - R is free.
       - If J's priority $Prio(t)$ is higher than the current priority ceiling $\hat{\pi}(t)$, R is allocated to J.
       - If J's priority $Prio(t)$ is not higher than the current priority ceiling $\hat{\pi}(t)$, R is allocated to J only if J is the job holding the resource whose priority ceiling is equal to $\hat{\pi}(t)$; otherwise, J's request is denied, and J becomes blocked.

Basic Priority-Ceiling Protocol (II)

- Rules of Basic Priority-Ceiling Protocol (II)

3. Priority-Inheritance Rue:
   - When J becomes blocked, the job J_l which blocks J inherits the current priority $\pi(t)$ of J.
   - J_l executes at its inherited priority until the time when it releases every resource whose priority ceiling is equal to or higher than $\hat{\pi}(t)$; at that time, the priority J_l returns to its priority $Prio(t')$ at time t' when it was granted the resources.

Example of PCP



Differences between the Priority-Inheritance and Priority-Ceiling Protocols

- Fundamental difference:
  - PIP is greedy:
    - Allocation rule of PIP lets the requesting job have a resource whenever the resource is free.
  - PCP is not greedy:
    - A job may be denied its requested resource even when the resource is free at the time.
    - It is possible for *J* to be blocked by a lower-priority job which does not hold the requested resource.

Multiprocessor Scheduling

Comparison with Uniprocessor Scheduling

- An on-line algorithm which builds a feasible schedule for any set of tasks with deadlines within m processors ($m \geq 2$), cannot exist.
- The necessary condition of schedulability referring to the maximum load U_j of each processor j
  - $U = \sum_{j=1}^{m} U_j = \sum_{i=1}^{n} U_i = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq m$
- Most multiprocessor scheduling problems are NP-Hard.

Scheduling Algorithms

- Partitioned algorithm
  - Without migration
- Global scheduling
  - With job-level migration
- Scheduling with restricted migration

Partitioning Algorithms

- In a Partitioned Scheduling, at design time each task is assigned to a processor, at run time task (or job) migration are forbidden.
- The partition problem, i.e., the assignment of tasks to processors at design time problem is a variant of the popular Bin-Packing optimization problem.
  - NP-Hard
- Most heuristics are greedy algorithms (never goes back and reverses the decision) and are based on two steps:
  1. Sorting
  2. Allocation

Allocation Heuristics

- First-Fit (FF): Assign the task τ_i on the first processor able to accept it, in the order of their indexes.
- Best-Fit (BF): Assign the task τ_i on the first processor with the highest utilization factor able to accept it.
- Worst-Fit (WF): Assign the task τ_i on the processor with the lowest utilization factor able to accept it.
- Almost Worst-Fit (AWF): Is a variant of WF in which the chosen processor has the second smallest utilization factor.
- Next-Fit (NF): Only the last processor created can receive tasks. When it is not possible to place the task τ_i, the current processor is closed (it will no longer be able to receive new tasks). The next processor then becomes the new current processor.

Global Scheduling

- All the jobs ready to be executed are placed in the single global queue and the m highest priority jobs are allocated on the m processors of the platform.
- The scheduler is allowed to interrupt job execution on a processor at any time, and may resume its execution on a different processor.
- Global scheduling is more complex to understand, study, and implement.

Maximum Utilization Bound

- Global EDF: $m - (m - 1)U_{max}$
- Fixed-job priority scheduler: $\frac{m+1}{2}$
- EDF-US[λ]: $m^2/(2m + 1)$ when $\lambda = m/(2m+1)$
  - Providing the highest priority to the tasks whose utilization is greater than the parameter λ

Other Task Models

- Multiframe model: the execution times of successive instances of a task are specified by a finite array of integer numbers rather than a single number which is the worst-case execution time commonly assumed in the classical model.
- (m, k)-constrained model: m jobs out of any and each overlapped window of k jobs of the task meet their deadlines
- Parallel tasks
  - Each job may be executed on different processors simultaneously
  - A job is said to be
    - rigid: if the number of processors assigned to this job is specified externally to the scheduler a priori, and does not change throughout its execution;
    - moldable: if the number of processors assigned to this job is determined by the scheduler, and does not change throughout its execution;
    - malleable if the number of processors assigned to this job can be changed by the scheduler during the job's execution.

Other Task Models

- Dynamic task model
  - An old task may depart and a new task may arrive in.
- Fork-Join task model
  - A task begins as a single master thread that executes sequentially until it encounters the first fork construct, where it splits into multiple parallel threads which synchronize/join on their termination
- Mode change
  - Temporal parameters of tasks may change during the run time
- Dag task model
  - Each task is represented as a directed acyclic graph, which is a set of precedence-constrained sequential jobs
- Resource model
  - Local resource and global resource

Other Considered Objectives

- Minimize energy consumption of the system
- Minimize reject rate of sporadic jobs
- Minimize miss rate of periodic jobs
- Minimize average response time of aperiodic jobs

Conclusion

- The system model of a real-time system consists of the workload model and the resource model.
- A task model includes many attributes and parameters for the tasks.
- A processor model indicates the feature of the processors in the system.
- Scheduling algorithms are proposed for the combination of a workload model and a resource model. The schedulability test is also important.

Thanks for your attention.

Q&A