Priority-Driven Scheduling of Periodic Tasks

Pei-Hsuan Tsai

CPS LAB

- Assumptions made in this chapter are that
  - **1.** the tasks are independent and
  - **2.** there are no aperiodic and sporadic tasks.
- Limitations of these algorithms, every job
  - is ready for execution as soon as it is released
  - can be preempted at any time
  - never suspends itself.
- Scheduling decisions
  - made immediately upon job releases and completions.
- Context switch overhead
  - is negligibly small compared with execution times of the tasks
- The number of priority levels
  - is unlimited.

- When an application creates a new task, the application
  - first requests the scheduler to add the new task
  - providing the scheduler with relevant parameters of the task, including its period, execution time, and relative deadline.
- Acceptance test on the new periodic task.
  - Accept when the new task and all other existing tasks can be feasibly scheduled,
  - Otherwise, the scheduler rejects the new task.

6.1 Static assumption

6.1 STATIC ASSUMPTION

- Focus on uniprocessor systems. Why?
- A multiprocessor priority-driven system is either dynamic or static.
- In a static system,
  - Tasks are partitioned into subsystems.
  - Each subsystem is assigned to a processor, and tasks on each processor are scheduled by themselves.
- In a dynamic system,
  - one common priority queue
  - Jobs are dispatched to processors for execution as the processors become available.
- The dynamic approach performs well most of the time.
- However, in the worst case, the performance of priority-driven algorithms can be unacceptably poor.

A simple example demonstrates worst-case.

- The application system contains $m + 1$ independent periodic tasks.
- T_i, for $i = 1, 2, \ldots m$, are identical.
  - periods are equal to 1, and
  - execution times are equal to 2ε, where ε is a small number.
- T_{m+1} :
  - periods is $1+\varepsilon$
  - execution time is 1
- The tasks $T_1 \sim T_{m+1}$ are in phase.
- Their relative deadlines are equal to their periods.
- EDF basis
- The first job $J_{m+1,1}$ in T_{m+1} has the lowest priority because it has the latest deadline.
- Figure 6–1 shows an EDF schedule of these jobs if the jobs are dispatched and scheduled dynamically on m processors.

- $J_{m+1,1}$ does not complete until $1 + 2\varepsilon$ and, hence, misses its deadline.

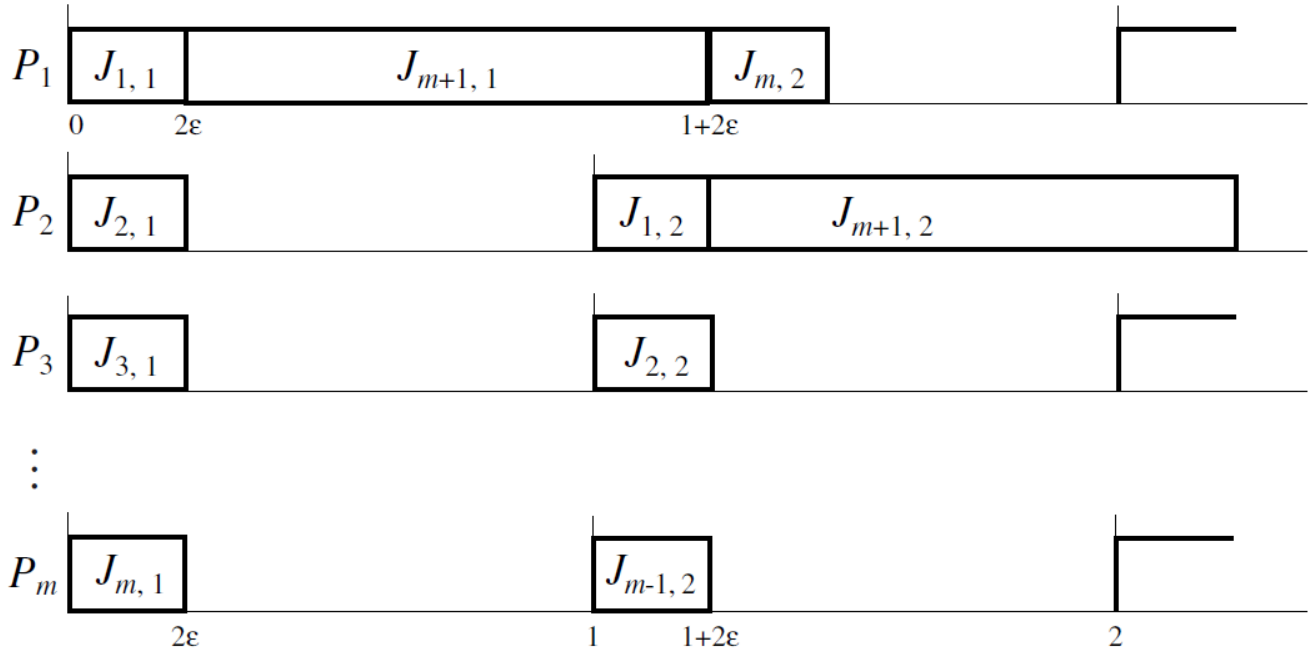


FIGURE 6–1 A dynamic EDF schedule on m processors.

- The total utilization U of these m +1 periodic tasks is $2m\varepsilon + 1/(1 + \varepsilon)$.
- This system can be feasibly scheduled statically.
- As long as $2m\varepsilon \leq 1$, this system can be feasibly scheduled on two processors.
  - put T_{m+1} on one processor and the other tasks on the other processor
  - Schedule the task(s) on each processor according to either of these priority-driven algorithms

Critical problem of dynamic system

- Do not know how to determine worst-case and best-case performance of it.
- The only way to validate a dynamic system is by simulating and testing the system.
- Most hard real-time systems built and in use to date and in the near future are static.
- When tasks in a static system are independent, we can consider the tasks on each processor independently of the tasks on the other processors.
- The problem of scheduling in multiprocessor and distributed systems is reduced to that of uniprocessor scheduling.

6.2 Fixed-priority vs. dynamic-priority algorithms

- A priority-driven scheduler is an on-line scheduler.
- It does not precompute a schedule of the tasks.
- It assigns priorities to jobs after they are released.
- It places the jobs in a ready job queue in priority order.
- A scheduling decision is made whenever a job is released or completed.
- At each scheduling decision time,
  - the scheduler updates the ready job queue.
  - schedules and executes the job at the head of the queue.

Priority-driven algorithms

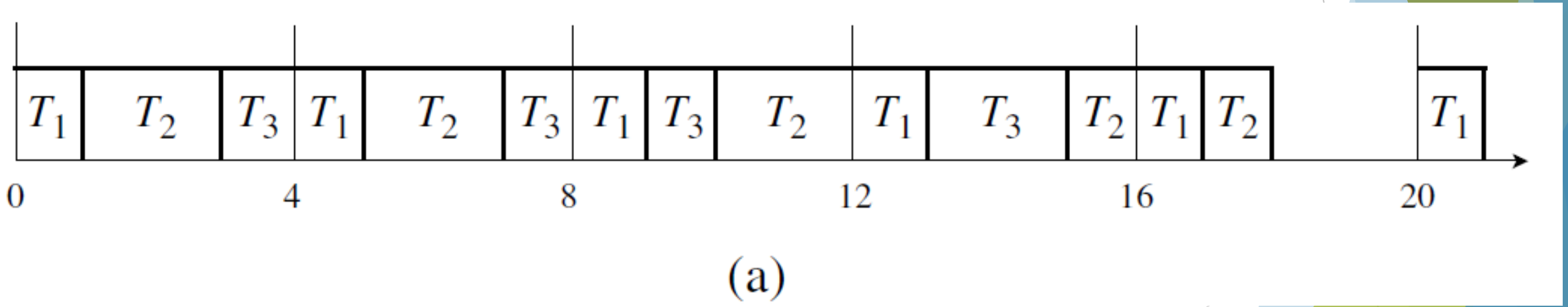
- How priorities are assigned to jobs.
- Two types: fixed priority and dynamic priority.
- A *fixed-priority* algorithm assigns the same priority to all the jobs in each task.
- A *dynamic-priority* algorithm assigns different priorities to the individual jobs in each task.
- Three categories of algorithms:
  - Fixed-priority algorithms: Rate-monotonic, Deadline-monotonic
  - Task-level dynamic-priority algorithms: EDF, FIFO, LIFO
  - Job-level dynamic-priority algorithms: LST

6.2 Rate-Monotonic Algorithms

- A well-known fixed-priority algorithm.
- The shorter the period, the higher the priority.
- The *rate*: the inverse of task period.
- The higher its rate, the higher its priority.

An example of RM schedule with 3 periodic tasks

- This system contains three tasks: $T_1 = (4, 1)$, $T_2 = (5, 2)$, and $T_3 = (20, 5)$.
- T_1 is the highest, T_2 has the next highest priority, T_3 has the lowest.
- $J_{2,1}$in T_2 is delayed until $J_{1,1}$ in $T1$ completes, and the $J_{2,3}$ in T_2 is preempted at time 16 when $J_{1,4}$ in $T1$ is released.

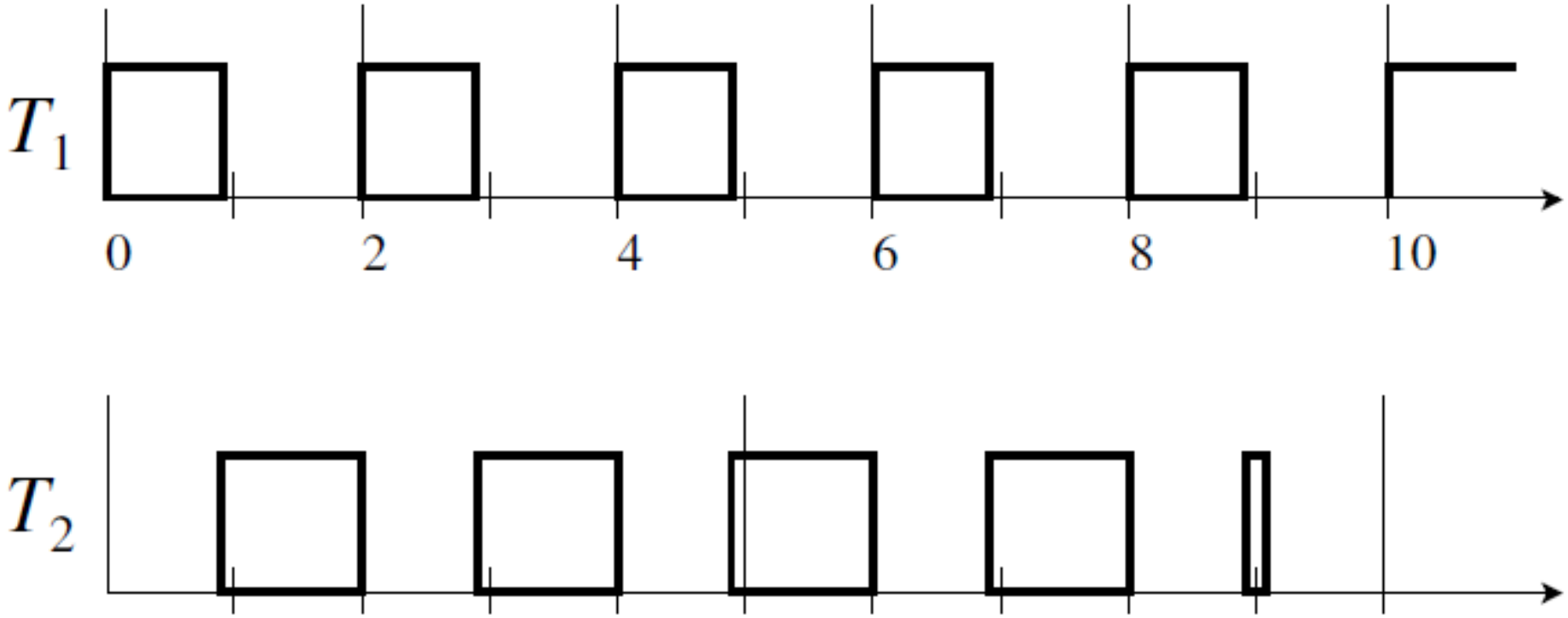


(a)

CPS LAB

Another form of RM schedule with 2 periodic tasks

- The schedule for the tasks $T_1 = (2, 0.9)$ and $T_2 = (5, 2.3)$.
- According to the RM algorithm, task T_1 has a higher-priority than task T_2.
- Every job in T_1 is scheduled and executed as soon as it is released.
- The jobs in T_2 are executed in the background of T_1.

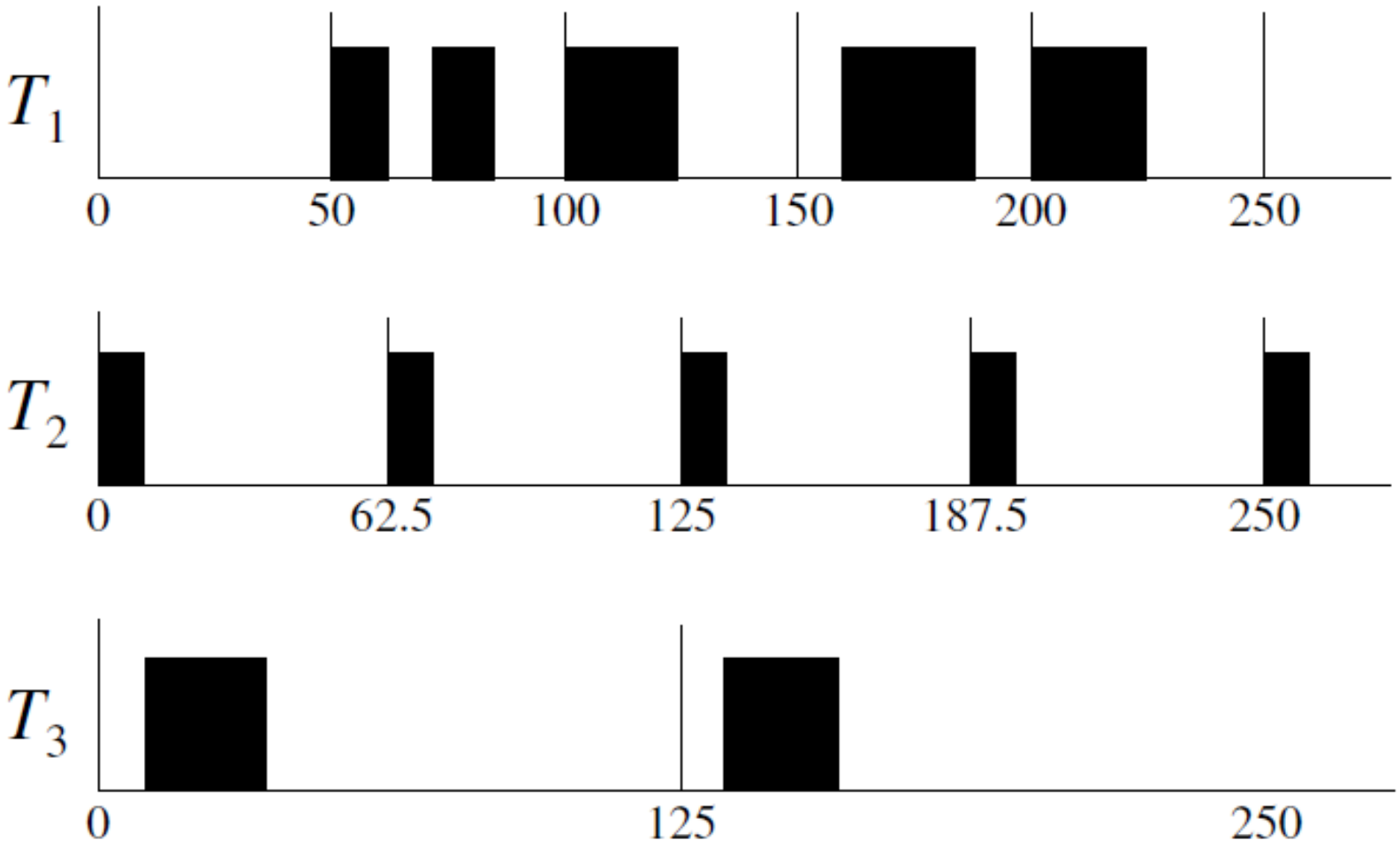


CPS LAB

Deadline-monotonic algorithm (DM)

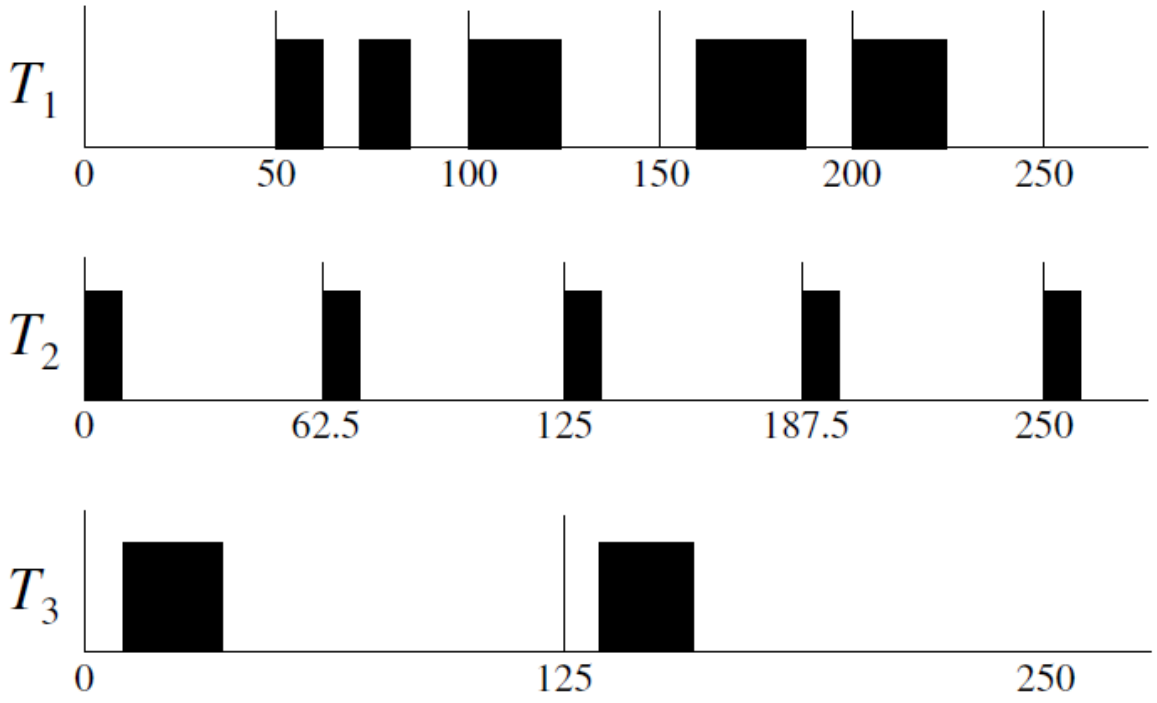
- Assign priorities to tasks according to relative deadlines: the shorter the relative deadline, the higher the priority.
- The system consists of three tasks, T_1 = (50, 50, 25, 100), T_2 = (0, 62.5, 10, 20), and T_3 = (0, 125, 25, 50).
- Their utilizations are 0.5, 0.16, and 0.2, respectively.
- The total utilization is 0.86.
- According to the DM algorithm, priority = $T_2 > T_3 > T_1$.

Deadline-monotonic Schedule

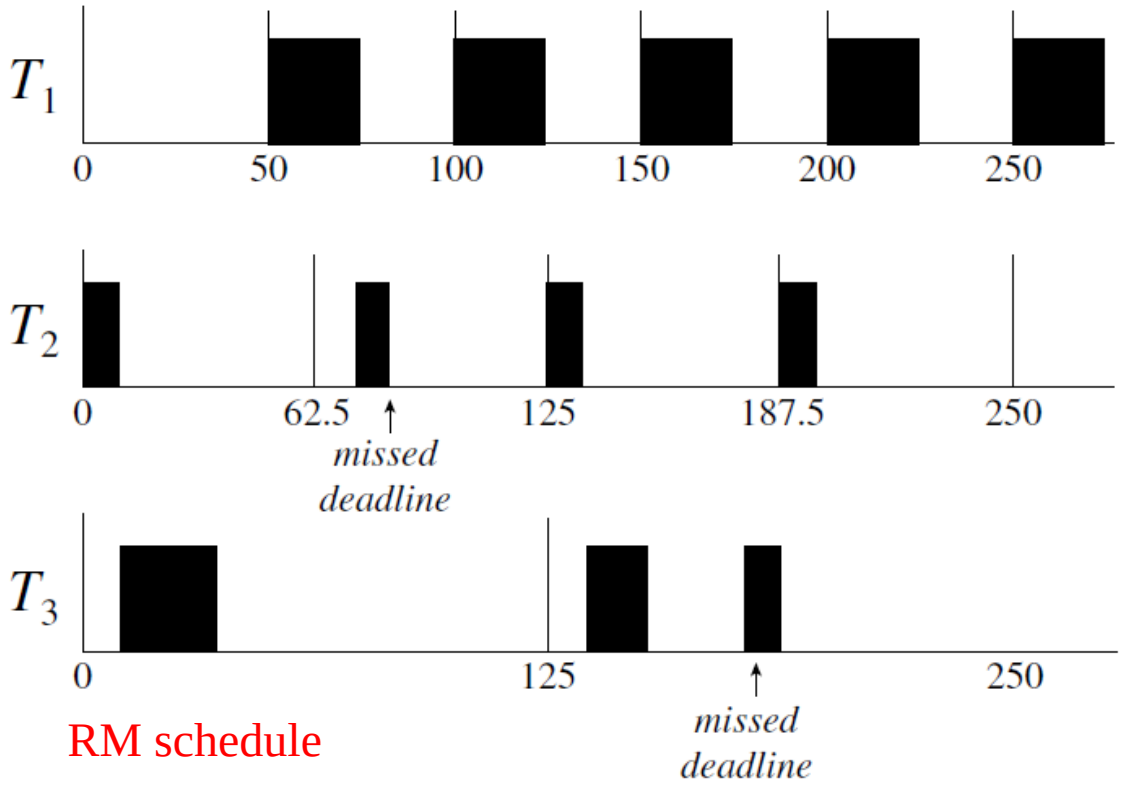


(a)

- According to the DM algorithm, T_2 has the highest priority while T_1 has the lowest priority. [$T_2 > T_3 > T_1$]
- According to the RM algorithm, $T1$ has the highest priority, and $T3$ has the lowest priority. [$T_1 > T_2 > T_3$]
- We see that because the priorities of the tasks with short relative deadlines are too low, these tasks cannot meet all their deadlines.



DM schedule



RM schedule

6.2.2 Well-known dynamic algorithm

- The EDF algorithm assigns priorities to individual jobs in the tasks according to their absolute deadlines.
- The EDF schedule for the tasks $T_1 = (2, 0.9)$ and $T_2 = (5, 2.3)$.

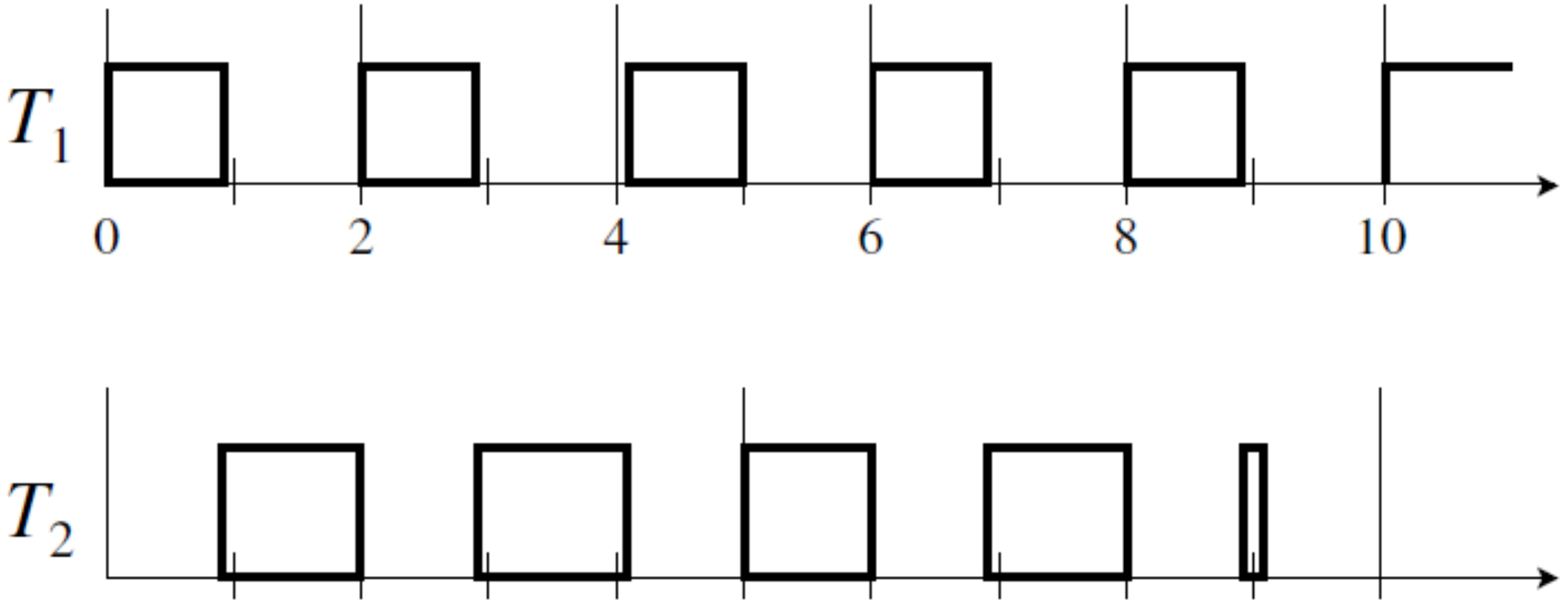


FIGURE 6–4 An earliest-deadline-first schedule of (2, 0.9) and (5, 2.3).

CPS LAB

Well-Known Dynamic Algorithms -LST

- Another well-known dynamic-priority algorithm is the *Least-Slack-Time-First* (LST) algorithm.
- The slack time of job at time time t , is equal to $d - t - x$
  - Where remaining execution time (i.e., the execution of its remaining portion) is x and deadline is d.
- The smaller the slack, the higher the priority
- The slacks of all the ready jobs are checked each time a new job is released.
- Coincidentally, the schedule of T_1 and T_2 in the above example produced by the LST algorithm happens to be identical to the EDF schedule in Figure 6–4.
- In general, however, the LST schedule of a system may differ in a fundamental way from the EDF schedule.

Example of LST schedule

- Three tasks: T_1 = (2, 0.8), T_2 = (5, 1.5), and T_3 = (5.1, 1.5).
- At time 0,
  - Slack of $J_{1,1}$, = 2 - 0.8 =1.2; $J_{2,1}$ = 5-1.5 = 3.5; $J_{3,1}$ = 5.1-1.5 = 3.6, respectively.
  - LST: $J_{1,1} > J_{2,1} > J_{3,1}$. $J_{1,1}$ has the highest priority, and $J_{3,1}$ has the lowest.
  - EDF: $J_{1,1} > J_{2,1} > J_{3,1}$.
- At time 0.8,
  - $J_{1,1}$ completes, and $J_{2,1}$ executes.
- At time 2,
  - $J_{1,2}$ is released.
  - Slack of $J_{1,2}$ = 2-0.8 = 1.2; $J_{2,1}$ = 5-2-0.3 = 2.7; and $J_{3,1}$ = 5.1-2-1.5 = 1.6, respectively.
  - LST: $J_{1,2} > J_{3,1} > J_{2,1}$
  - EDF: $J_{1,1} > J_{2,1} > J_{3,1}$.
- LST algorithm is a job-level dynamic-priority algorithm.
- EDF algorithm is a job-level fixed-priority algorithm.

- Non-strict LST algorithm:
  - scheduling decisions are made only at the times when jobs are released or completed.
- Strictly LST algorithms:
  - monitor the slacks of all ready jobs and compare them with the slack of the executing job.
  - reassign priorities to jobs whenever their slacks change relative to each other.
- $T_1 = (2, 0.9)$ and $T_2 = (5, 2.3)$ as an example:
  - at time 2.7, the slack of $J_{2,1}$ becomes $(5 - 2.7 - 1.2) = 1.1$, the same as that of $J_{1,2}$.
  - schedule the two ready jobs in a round-robin manner until $J_{1,2}$ completes.
- The run-time overhead:
  - the time required to monitor and compare the slacks of all ready jobs as time progresses.
  - extra context switches.
- Strictly LST algorithm is an unattractive alternative.

Other dynamic algorithms- FIFO & LIFO

- Three tasks: $T_1 = (0, 3, 1, 3)$, $T_2 = (0.5, 4, 1, 1)$, and $T_3 = (0.75, 7.5, 2, 7.5)$.
- FIFO basis:
  - $J_{1,1} > J_{2,1} > J_{3,1}$.
  - $T_1 > T_2 > T_3$ initially.
- Later, $J_{1,4}$, $J_{2,3}$, and $J_{3,2}$ are released at the times 9, 8.5, and 8.25, respectively,
  - $T_3 > T_2 > T_1$.

6.2.3 Relative Merits

- Algorithms (FIFO, LIFO) that do not take into account the urgencies of jobs in priority assignment usually perform poorly.
- T_1 = (2, 0.9) and T_2 = (5, 2.3) on the FIFO basis, most of the jobs in T_1 would miss their deadlines.
- Fixed-priority algorithms usually assigns priorities to tasks on the basis of their functional criticality: the more critical the task, the higher the priority.
  - Ex:T_1 is a video display task. T_2 is a task which monitors and controls a patient's blood pressure.
  - $T_2 > T_1$
  - T_1 would miss most of its deadlines.
  - Unnecessary, T_1 and T_2 can be feasibly scheduled.
- Assign priorities to jobs based on temporal parameters of jobs.
- Have either optimal or reasonably good performance: RM, DM, EDF, and the LST algorithm.

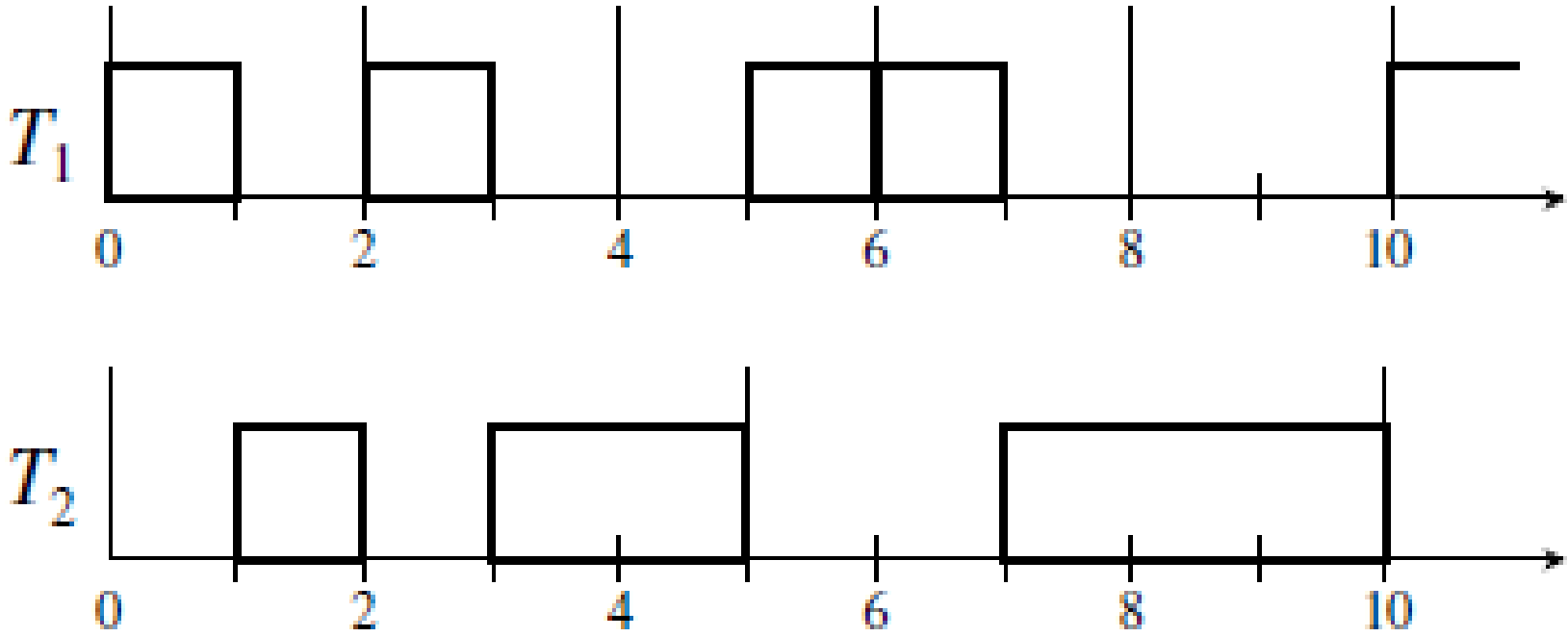
- *A scheduling algorithm can feasibly schedule any set of periodic tasks on a processor if the total utilization of the is equal to or less than the schedulable utilization of the algorithm.*
- The higher the schedulable utilization of an algorithm, the better the algorithm.
- An algorithm whose schedulable utilization is equal to 1 is an optimal algorithm.
- EDF algorithm is optimal.
- RM and DM algorithms are not optimal.

Performance vs. predictability

- Optimal dynamic-priority algorithms outperform fixed-priority algorithms based on the criterion of schedulable utilization,
- An advantage of fixed-priority algorithms is predictability.
- When tasks have fixed priorities, overruns of jobs in a task can never affect higher-priority tasks.
- In contrast, when the tasks are scheduled according to a dynamic algorithm, it is difficult to predict which tasks will miss their deadlines during overloads.

Figure 6-5 Unpredictability and instability of the EDF.

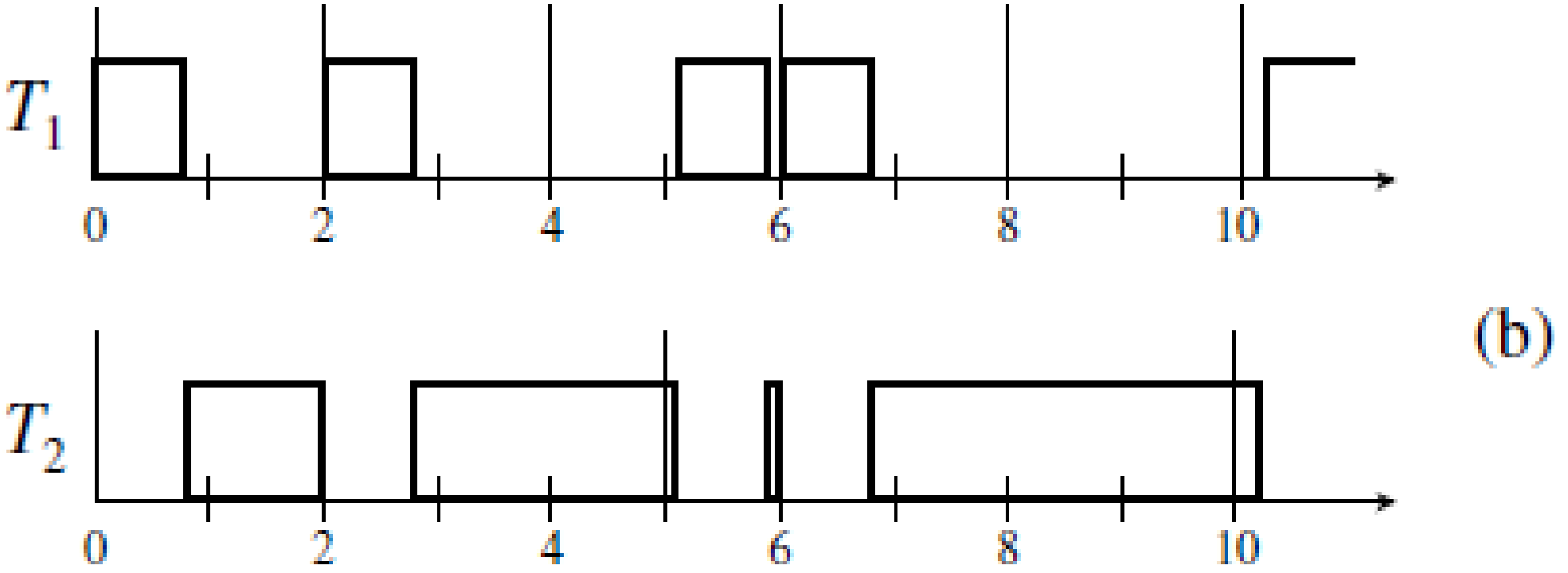
- (a) $T_1 = (2,1)$ and $T_2 = (5, 3)$ with $U = 1.1$
- The tasks shown in Figure 6–5(a) have a total utilization of 1.1.
- According to their EDF schedule shown here, the job $J_{1,5}$ in $T_1 = (2, 1)$ is not scheduled until 10 and misses its deadline at 10.
- Deadlines of all jobs in T_2 are met in the schedule segment.



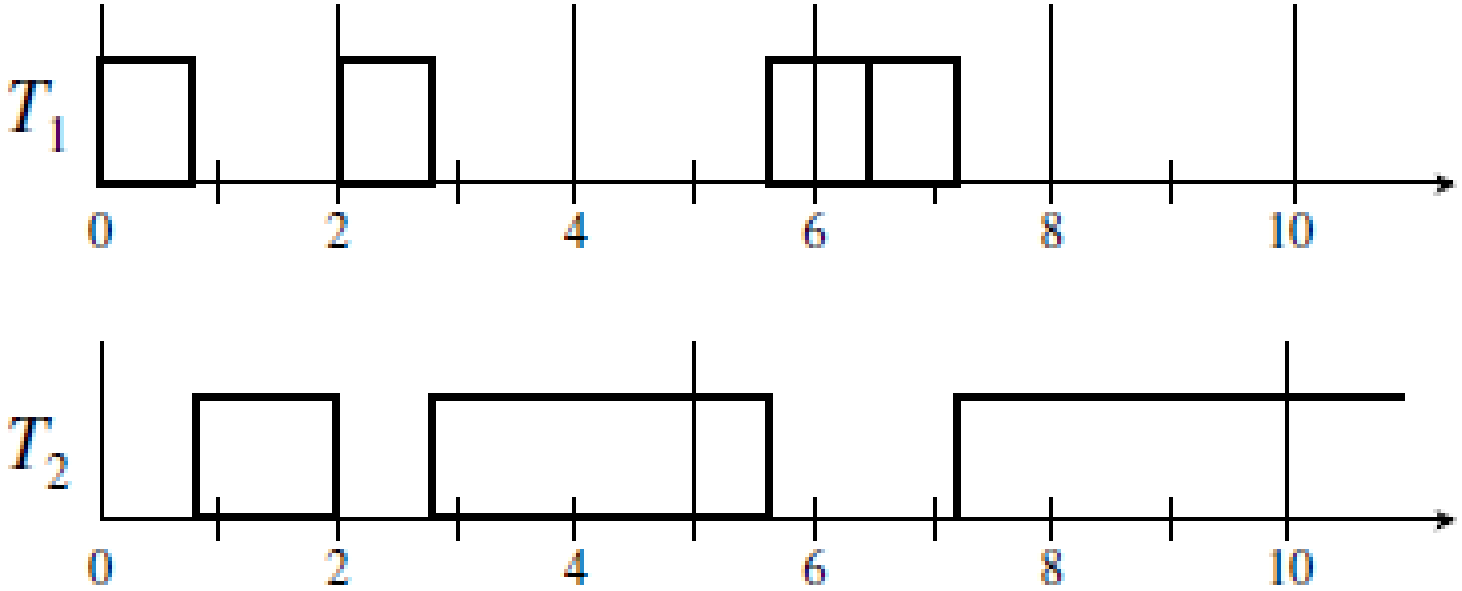
(a)

CPS LAB

- (b) T_1 = (2,0.8) and T_2 = (5, 3.5) with U = 1.1. (0.8/2 + 3.5/5 = 11/10)
- According to the EDF schedule in Figure 6–5(b), $J_{1,5}$ in T_1 = (2, 0.8), as well as every job in T_2, cannot complete on time.
- There is no easy test to determine which tasks will miss their deadlines and which tasks will not.



- The EDF algorithm has another serious disadvantage.
- A late job has a higher-priority than a job whose deadline is still in the future.
- Consequently, if the execution of a late job is allowed to continue, it may cause some other jobs to be late.
- (c) $T_1 = (2, 0.8)$ and $T_2 = (5, 4.0)$ with $U = 1.2$ $(0.8/2 + 4/5 = 12/10)$
- At time 5, $J_{2,1}$ becomes late, it continues to execute because its priority is higher.
- As a consequence, $J_{1,3}$ is late also and after $J_{1,4}$, every job in both tasks will be late.
- EDF is unsuitable for systems where overload conditions are unavoidable.



CPS LAB

6.3 Maximum schedulable utilization

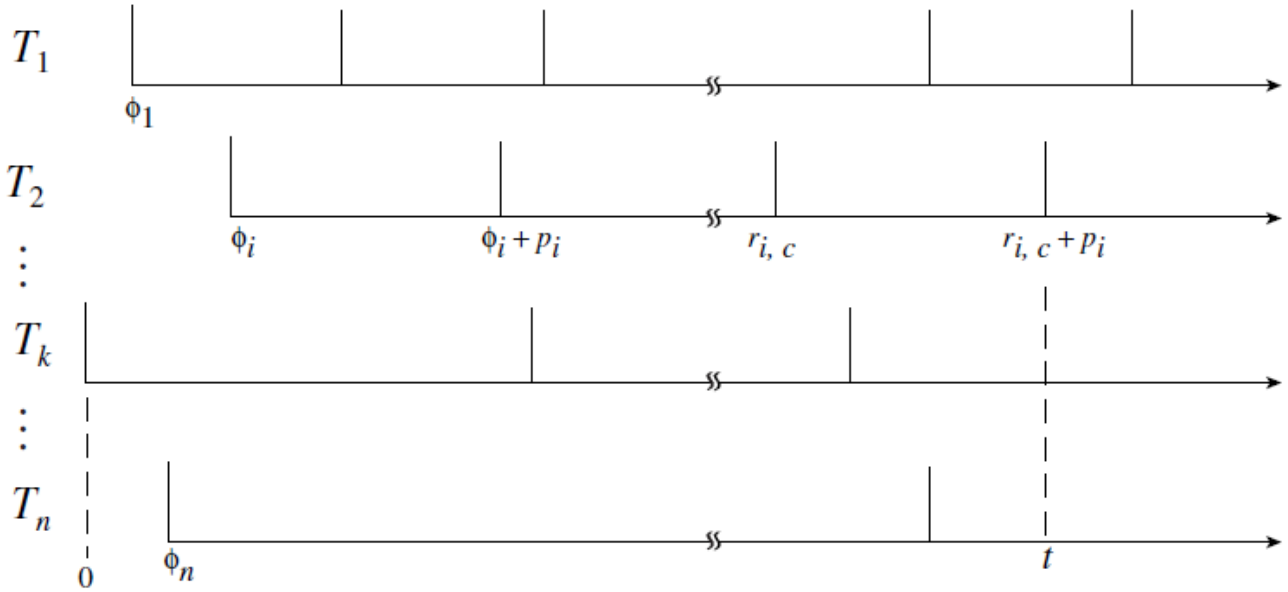
6.3.1 Schedulable Utilizations of the EDF Algorithm

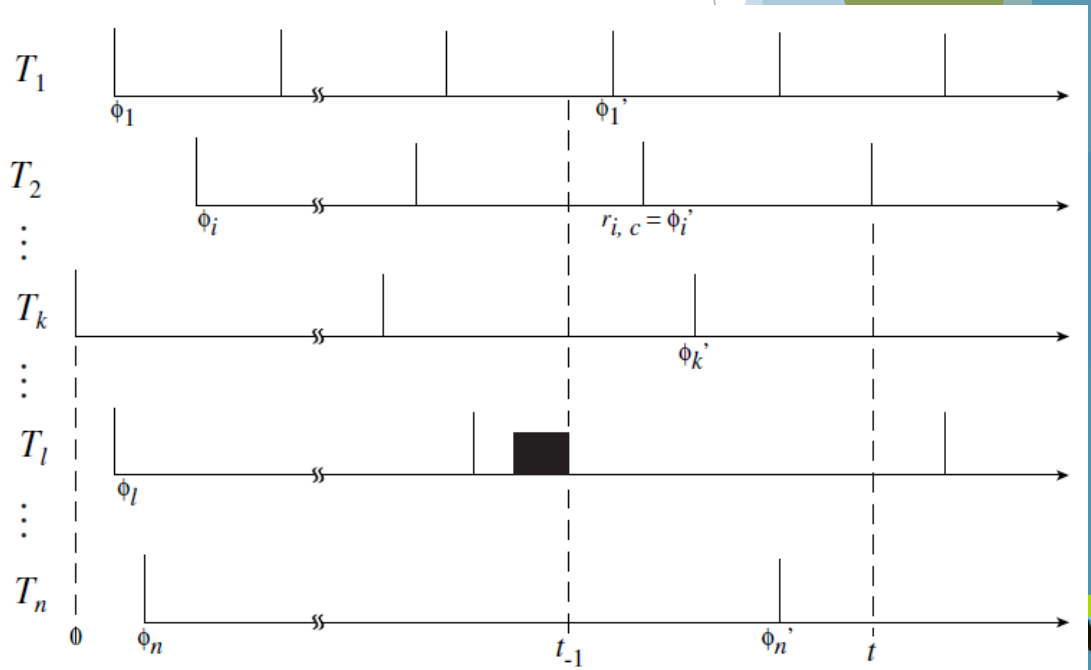
- **THEOREM 6.1.** A system *T* of independent, preemptable tasks with relative deadlines equal to their respective periods can be feasibly scheduled on one processor if and only if its total utilization is equal to or less than 1.
- Proof: read the book ◦
- The following facts follow straightforwardly from this theorem.
  - **1.** *A system of independent, preemptable periodic tasks with relative deadlines longer than their periods can be feasibly scheduled on a processor as long as the total utilization is equal to or less than 1.*
  - **2.** *The schedulable utilization UEDF(n) of the EDF algorithm for n independent, preemptable periodic tasks with relative deadlines equal to or larger than their periods is equal to 1.*

proof.

- Focus on "if its total utilization is larger than 1". <the system is not feasible>
- As stated in Theorem 4.1, the EDF algorithm is optimal in the sense that it can surely produce a feasible schedule of any feasible system.
- the EDF algorithm can surely produce a feasible schedule of any system with a total utilization equal to 1.
- According to an EDF schedule, the system fails to meet some deadlines, then its total utilization is larger than 1.
- Suppose that the system begins to execute at time 0 and at time t, the job $J_{i,c}$ of task T_i misses its deadline.

- Assume that prior to t the processor never idles.
- Two cases :
- (1) The current period of every task begins at or after $r_{i,c}$, the release time of the job that misses its deadline
- (2) the current periods of some tasks begin before $r_{i,c}$.
- The two cases are illustrated in Figure 6–6.
- In this figure, we see that the current jobs of all tasks T_k , for all k≠ i, have equal or lower priorities than $J_{i,c}$ because their deadlines are at or after t.





- Case (1): The fact that $J_{i,c}$ misses its deadline at t tells us that any current job whose deadline is after t is not given any processor time to execute before t and that the total processor time required to complete $J_{i,c}$ and all the jobs with deadlines at or before t exceeds the total available time t.
- In other words,

$$t < \frac{(t - \phi_i)e_i}{p_i} + \sum_{k \neq i} \left\lfloor \frac{t - \phi_k}{p_k} \right\rfloor e_k$$

- The first term on the right-hand side of the inequality is the time required to complete all the jobs in T_i with deadlines before t and the job $J_{i,c}$.
- Each term in the sum gives the total amount of time before t required to complete jobs that are in a task T_k other than T_i and have deadlines at or before t.
- Combining this inequality with the one in Eq. (6.1), we have U > 1.

$$\frac{(t - \phi_i)e_i}{p_i} + \sum_{k \neq i} \left\lfloor \frac{t - \phi_k}{p_k} \right\rfloor e_k \leq t\frac{e_i}{p_i} + t\sum_{k \neq i} \frac{e_k}{p_k} = t\sum_{k=1}^{n} u_k = tU$$

- Case (2):
- Let T' denote the subset of T containing all the tasks whose current jobs were released before $r_{i,c}$ and have deadlines after t.
- It is possible that some processor time before $r_{i,c}$ was given to the current jobs of some tasks in T'.
- In the figure, T_l is such a task.
- Let t_{-l} be the end of the latest time interval I (shown as a black box in Figure 6–6(b)) that is used to execute some current job in T'.
- . In this segment starting from t_{-l}, none of the current jobs with deadlines after t are given any processor time.

$$t - t_{-1} < \frac{(t - t_{-1} - \phi'_i)e_i}{p_i} + \sum_{T_k \in \mathbf{T}-\mathbf{T}'} \left\lfloor \frac{t - t_{-1} - \phi'_k}{p_k} \right\rfloor e_k$$

- This inequality is the same as the one in Eq. (6.1), which in turn implies that U > 1

Facts follow straightforwardly from theorem

- **1.** *A system of independent, preemptable periodic tasks with relative deadlines longer than their periods can be feasibly scheduled on a processor as long as the total utilization is equal to or less than 1.*
- **2.** *The schedulable utilization UEDF(n) of the EDF algorithm for n independent, preemptable periodic tasks with relative deadlines equal to or larger than their periods is equal to 1.*

Another algorithm which schedulable utilization is 1: LST

- *The schedulable utilization of the LST algorithm is also 1.*
- BUT!When the relative deadlines of some tasks are less than their respective periods, the system may no longer be feasible, even when its total utilization is less than 1.
- As an example, the task with period 5 and execution time 2.3 in Figure 6–4 would not be schedulable if its relative deadline were 3 instead of 5.

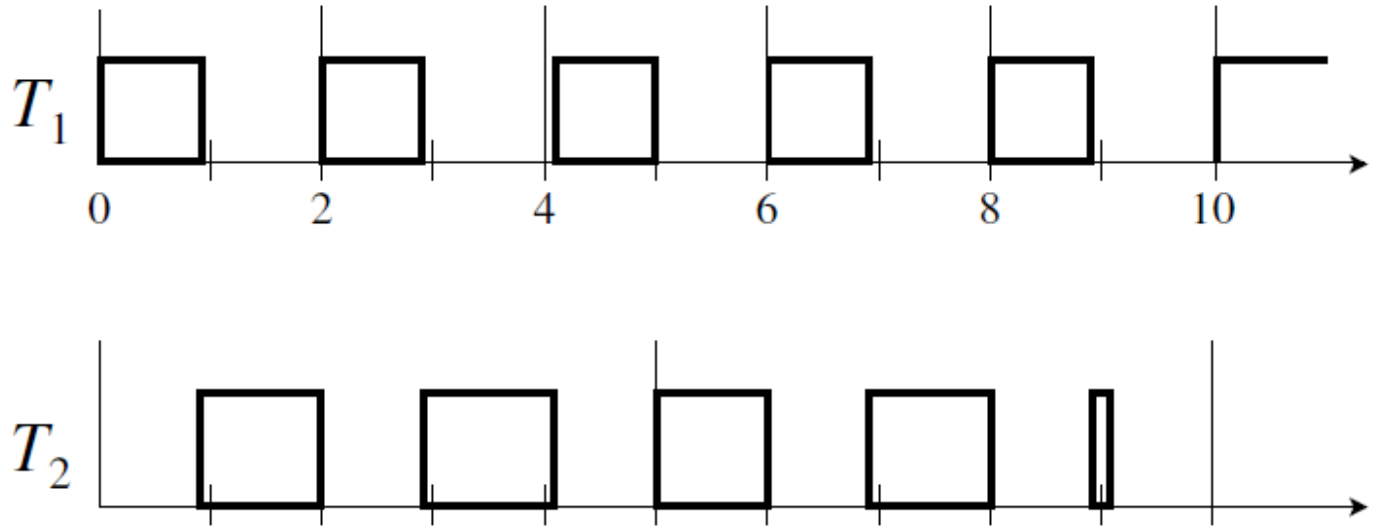


FIGURE 6–4 An earliest-deadline-first schedule of (2, 0.9) and (5, 2.3).

Density of Tasks

- We call the ratio of the execution time e_k of a task T_k to the minimum of its relative deadline D_k and period p_k the *density* δ_k of the task.
- In other words, the density of T_k is $e_k/\min(D_k , p_k)$.
- The sum of the densities of all tasks in a system is the *density* of the system and is denoted by Δ.
- When $D_i < p_i$ for some task T_i , $\Delta > U$.
- If the density of a system is larger than 1, the system may not be feasible.
- For example, this sum is larger than 1 for (2, 0.9) and (5, 2.3, 3), and the tasks are not schedulable by any algorithm.
- On the other hand, any system is feasible if its density is equal to or less than 1.

- **THEOREM 6.2.** A system **T** of independent, preemptable tasks can be feasibly scheduled on one processor if its density is equal to or less than 1.
- The condition given by this theorem is not necessary for a system to be feasible.
- A system may be feasible when its density is greater than 1.
- The system consisting of (2, 0.6, 1) and (5, 2.3) is an example.
- Its density is larger than 1, but it is schedulable according to the EDF algorithm.

6.3.2 Schedulability test for the EDF algorithm

- a *schedulability test :*
  - To validate whether all the jobs meet their hard deadlines in the given application system.
  - If a schedulability test is efficient, it can be used as an on-line acceptance test.
- The validation problem that can be stated as follows:
  - **1. Workload:** the period p_i , execution time e_i , and relative deadline D_i of every task T_i in a system $\mathbf{T} = \{T_1, T_2, \ldots T_n\}$ of independent periodic tasks, and
  - **2. Algorithm:** a priority-driven algorithm used to schedule the tasks in **T** preemptively on one processor.
- To determine whether all the deadlines of every task T_i , for every $1 \leq i \leq n$, are always met.

Schedulability Test for the EDF Algorithm

- To determine whether the given system of n independent periodic tasks surely meets all the deadlines when scheduled according to the preemptive EDF algorithm on one processor, we check whether the inequality (6.2) is satisfied.

$$\sum_{k=1}^{n} \frac{e_k}{\min(D_k, p_k)} \leq 1$$

-
- (6.2)is the *schedulability condition* of the EDF algorithm.
- If it is satisfied, the system is schedulable according to the EDF algorithm.
- When Eq. (6.2) is not satisfied, the conclusion we may draw from this fact depends on the relative deadlines of the tasks.
- If $D_k \geq p_k$ for all k from 1 to n, then Eq. (6.2) reduces to $U \leq 1$, which is both a necessary and sufficient condition for a system to be feasible. (充分必要條件，假如6.2不滿足，那系統一定無法被EDF排。)
- On the other hand, if $D_k < p_k$ for some k, Eq. (6.2) is only a sufficient condition; therefore we can only say that the system may not be schedulable when the condition is not satisfied. (充分條件，意思是當6.2不被滿足時，系統有可能無法被排程，但不是一定無法被排程。)

Schedulability test can help to design system

- Eq. (6.2) can guide the choices of the periods and execution times of the tasks while we design the system.
- An example: a digital robot controller.
- A control-law computation
  - takes 8 milliseconds on the chosen processor to complete.
  - The control-law task executes once every 10 milliseconds.
  - Utilization : 0.8
- Add a Built-In Self-Test (BIST) task.
  - The maximum execution time of this task is 50 milliseconds.
  - By equation (6.2), we can execute the BIST task every 250 milliseconds.
  - Utilization : 0.2
- Add a telemetry task.
  - Execute 15 milliseconds.
  - By equation (6.2), reduce the BIST task to once a second. (50/1000 = 0.05)
  - The relative deadline of the telemetry task as short as 100 milliseconds. (0.05 +0.8 + 0.15)
- Total utilization: control-law computation (0.8)+BIST (0.05)+telemetry (0.15) ≤ 1

$$\sum_{k=1}^{n} \frac{e_k}{\min(D_k, p_k)} \leq 1$$

CPS LAB

6.4 Optimality of the RM and DM algorithms

Fixed priority algorithm can not be optimal (schedulable utilization =1)

- Priority: $T_i > T_k$ if $i < k$.
- π_i ：the priority of a task T_i as priority.
- total utilization of the subset of tasks with equal or higher priority than T_i = $U_i = \sum_{k=1}^{i} u_k$
- Fixed-priority algorithms cannot be optimal.
- Example:
  - Two tasks: T_1 = (2, 1) and T_2 = (5, 2.5).
  - Total utilization = 1, the tasks are feasible.
  - $J_{1,1}$ and $J_{1,2}$ can complete in time only if they have a higher priority than $J_{2,1}$.
  - In the time interval (0, 4], T_1 must have a higher-priority than T_2.
  - However, at time 4 when $J_{1,3}$ is released, $J_{2,1}$ can complete in time only if T_2 (i.e., $J_{2,1}$) has a higher priority than T_1 (i.e., $J_{1,3}$).
- This change in the relative priorities of the tasks is not allowed by any fixed priority algorithm.

RM is optimal with simple periodic tasks

- A system of periodic tasks is *simply periodic* if for every pair of tasks T_i and T_k in the system and $p_i < p_k$, p_k is an integer multiple of p_i . (兩兩成倍數關係)
- An example of a simply periodic task system is the flight control system.
- In that system, the shortest period is 1/180 seconds.
- The other two periods are two times and six times 1/180 seconds.
- **THEOREM 6.3.** A system of simply periodic, independent, preemptable tasks whose relative deadlines are equal to or larger than their periods is schedulable on one processor according to the RM algorithm if and only if its total utilization is equal to or less than 1.

Proof of Theorem 6.3

- The tasks are in phase and the processor never idles before the task T_i misses a deadline for the first time at t .
- t is an integer multiple of p_i.
- t is also an integer multiple of the period p_k of every higher-priority task T_k, for $k = 1, 2, \ldots, i - 1$.
- The total time required to complete all the jobs at t is equal to $\sum_{k=1}^{i} (e_k t / p_k)$ which is equal to t times the total utilization of the i highest priority tasks.

  t *($U_i = \sum_{k=1}^{i} u_k$)
- That T_i misses a deadline at t means that this demand for time exceeds t .
- In other words, $U_i > 1$.

DM algorithm is the optimal fixed-priority algorithm.

- **THEOREM 6.4.** A system **T** of independent, preemptable periodic tasks that are in phase and have relative deadlines equal to or larger than their respective periods can be feasibly scheduled on one processor according to the DM algorithm whenever it can be feasibly scheduled according to any fixed-priority algorithm.

Proof

- Transform a feasible fixed-priority schedule into DM schedule.
- Starting from task T_1 with the shortest relative deadline in order of increasing relative deadlines.
- When we find two tasks T_i and T_{i+1} which are such that D_i is less than D_{i+1} but T_i has a lower priority than T_{i+1} according to this schedule, we switch the priorities of these two tasks and modify the schedule of the two tasks accordingly.
- After the switch, the priorities of the two tasks are assigned on the DM basis relative to one another.
- By repeating this process, we can transform the given schedule into a DM schedule.

RM and DM are optimal in some systems.

- DM and RM algorithms are the same when the relative deadline of every task T_i is equal to δp_i for some constant $0 < \delta$.
- *The RM algorithm is optimal among all fixed-priority algorithms whenever the relative deadlines of the tasks are proportional to their periods.*

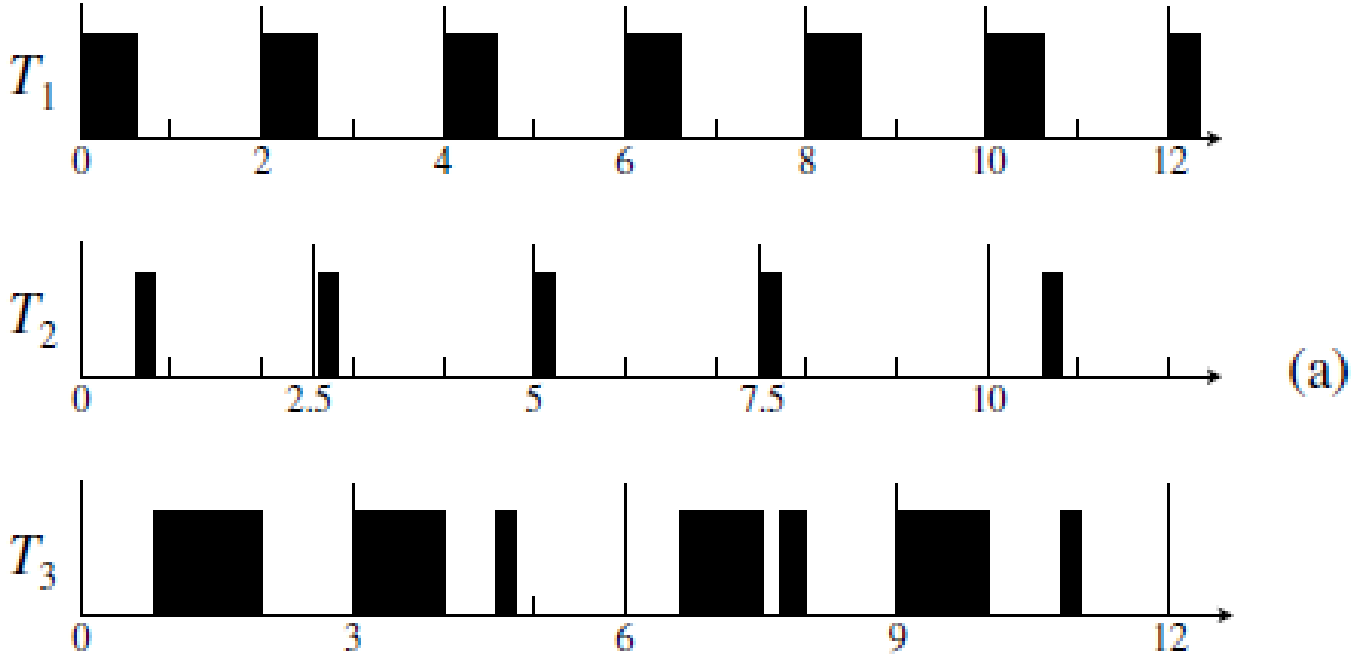
6.5 A schedulability test for fixed-priority tasks with short response times

6.5.1 Critical Instants

- The schedulability test checks one task T_i at a time to determine whether the response times of all its jobs are equal to or less than its relative deadline D_i .
- Use worst-case combination of release times of any job $J_{i,c}$ in T_i and all the jobs that have higher priorities than $J_{i,c}$.
- We call the response time of a job in T_i released at a critical instant the *maximum* (*possible*) *response time* of the task and denote it by W_i .
- **THEOREM 6.5.** In a fixed-priority system where every job completes before the next job in the same task is released, a critical instant of any task T_i occurs when one of its job $J_{i,c}$ is released at the same time with a job in every higher-priority task, that is, $r_{i,c} = r_{k,l_k}$ for some l_k for every $k = 1, 2, \ldots, i - 1$.
- Proof: Skip.

Example of critical instant

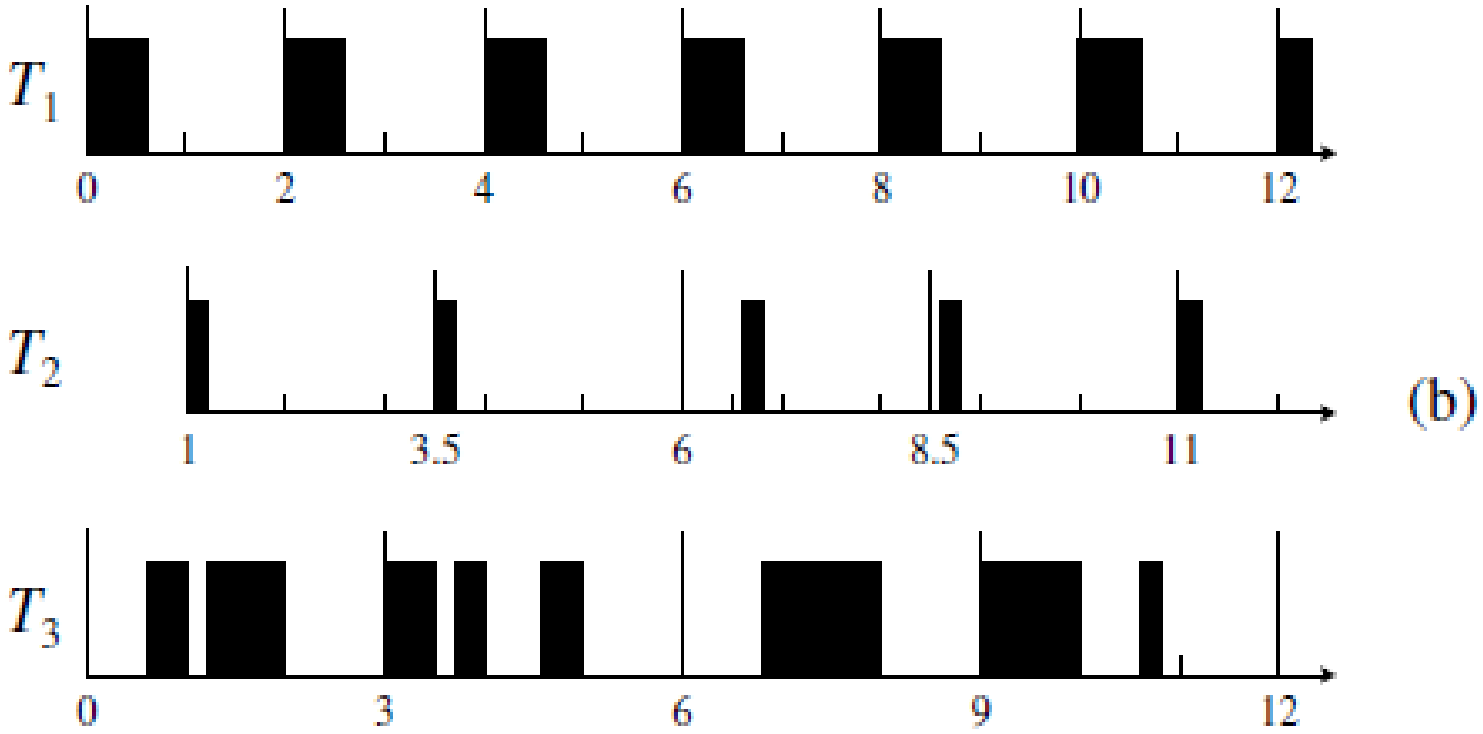
- Figure 6–8(a) shows an RM schedule of the three tasks, T_1=(2, 0.6), T_2=(2.5, 0.2), and T_3=(3, 1.2) when they are in phase.
- $T_1 > T_2 > T_3$.
- Time 0 is a critical instant of both lower-priority tasks.
- The response times of the jobs in T_2(2.5, 0.2) are 0.8, 0.3, 0.2, 0.2, 0.8, and so on.
- The response times of the jobs in T_3(3, 1.2) are 2, 1.8, 2, 2, and so on.



(a)

Another example

- RM schedule of T_1=(2, 0.6), T_2=(2.5, 0.2), and T_3=(3, 1.2), when the phase of the task T_2 is 1, while the phases of the other tasks are 0.
- We see that 6 is a critical instant of the two lower-priority tasks.
- The jobs of these tasks released at this instant have the maximum possible response times of 0.8 and 2, respectively.



(b)

6.5.2 Time-Demand Analysis

- Compute the total demand for processor time as a function of time from the critical instant by
  - a job released at a critical instant
  - all the higher-priority tasks
- Check whether this demand can be met before the deadline of the job.
- Starting from the task T_1 with the highest priority in order of decreasing priority.
- At time $t_0 + t$ for $t \geq 0$, the total (processor) time demand $w_i(t)$ of this job and all the higher-priority jobs released in $[t_0, t]$ is given by e_k ,

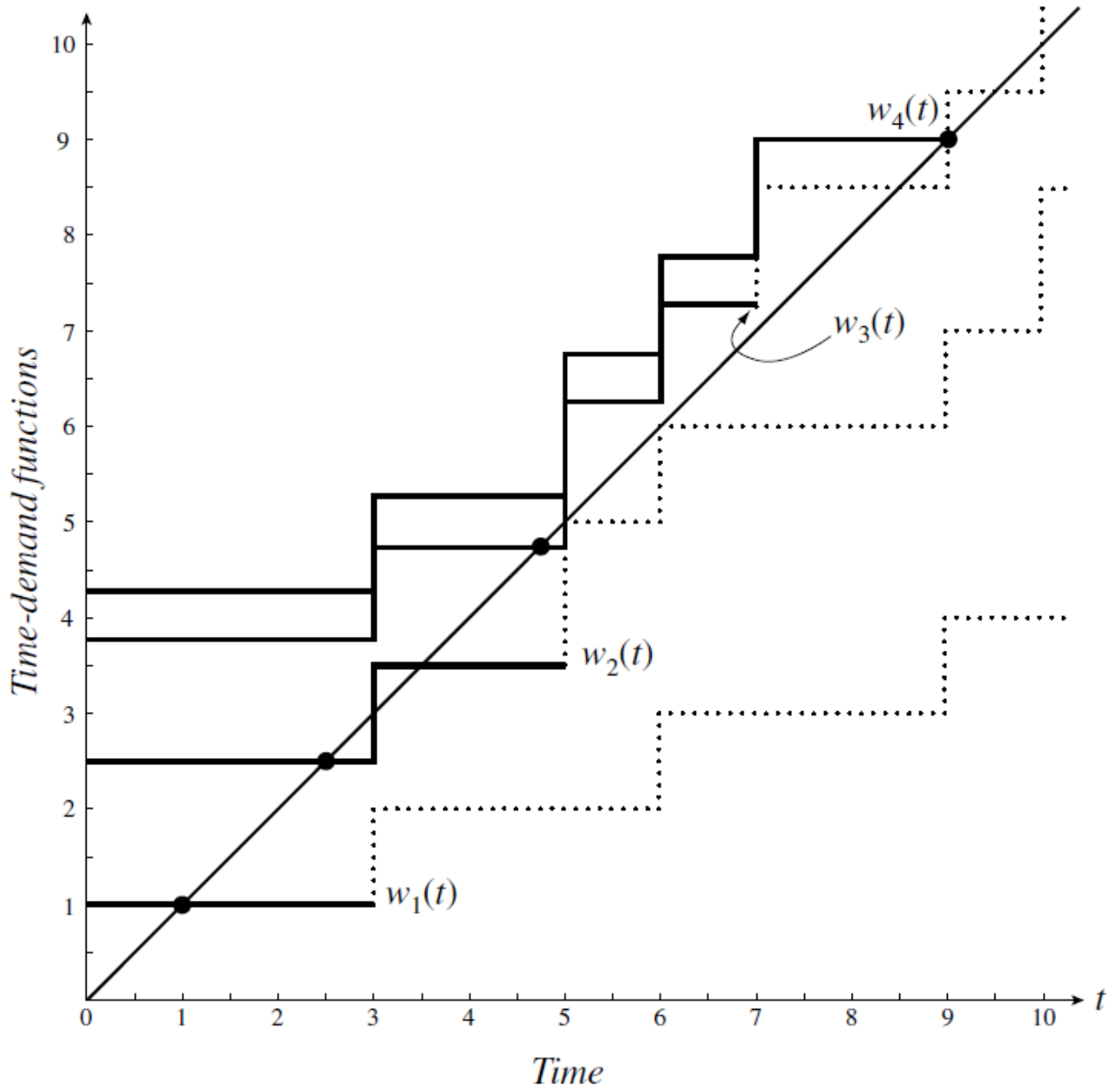
$$w_i(t) = e_i + \sum_{k=1}^{i-1} \left\lceil \frac{t}{p_k} \right\rceil e_k, \qquad \text{for } 0 < t \leq p_i \qquad (6.5)$$

- This job of T_i can meet its deadline $t_0 + D_i$ if
  - $w_i(t) \leq t$ for some $t \leq D_i$, where D_i is equal to or less than p_i .

CPS LAB

Example of Time-Demand Analysis

- $T_1 = (3, 1)$, $T_2 = (5, 1.5)$, $T_3 = (7, 1.25)$, $T_4 = (9, 0.5)$.
- Total utilization is 0.87
- RM algorithm.
- The solid lines:
  - the time-demand functions of these four tasks.
- The dotted lines:
  - the total contributions of higher-priority tasks
- Every task in this system is schedulable.
- The maximum possible response times of the tasks are 1, 2.5, 4.75, and 9, respectively.



Additional task

- A fifth task T_5= (10, 1).
- The time-demand function of this task lies entirely above the supply function *t* from 0 to 10.
- This task cannot be feasibly scheduled by RM.
- The time-demand function of any task T_i is a staircase function.
- The rises in the function occur at time instants which are integer multiples of periods of higher-priority tasks.
- The shortage $w_i(t) - t$ between the processor-time demand and the supply is the smallest for all *t* in the interval from the previous rise.
- Whether a task is schedulable,
  - the time-demand function of the task is equal to or less than the supply at these instants.

Summary

- To determine whether T_i can be feasibly scheduled by the given scheduling algorithm using the *time-demand analysis method*
  - **1.** compute the time-demand function $w_i(t)$ according to Eq. (6.5), and
  - **2.** check whether the inequality

$$w_i(t) \leq t \qquad (6.6a)$$

  is satisfied for values of t that are equal to

$$t = jp_k; \qquad k = 1, 2, \ldots i; \; j = 1, 2, \ldots, \lfloor \min(p_i, D_i)/p_k \rfloor \qquad (6.6b)$$

- If this inequality is satisfied at any of these instants, T_i is schedulable.

6.6 Schedulability test for sixed-priority tasks with arbitrary response times

6.6.1 Busy intervals, level-π_i

- A *level-π_i busy interval* (t_0, t] begins at an instant t_0 when

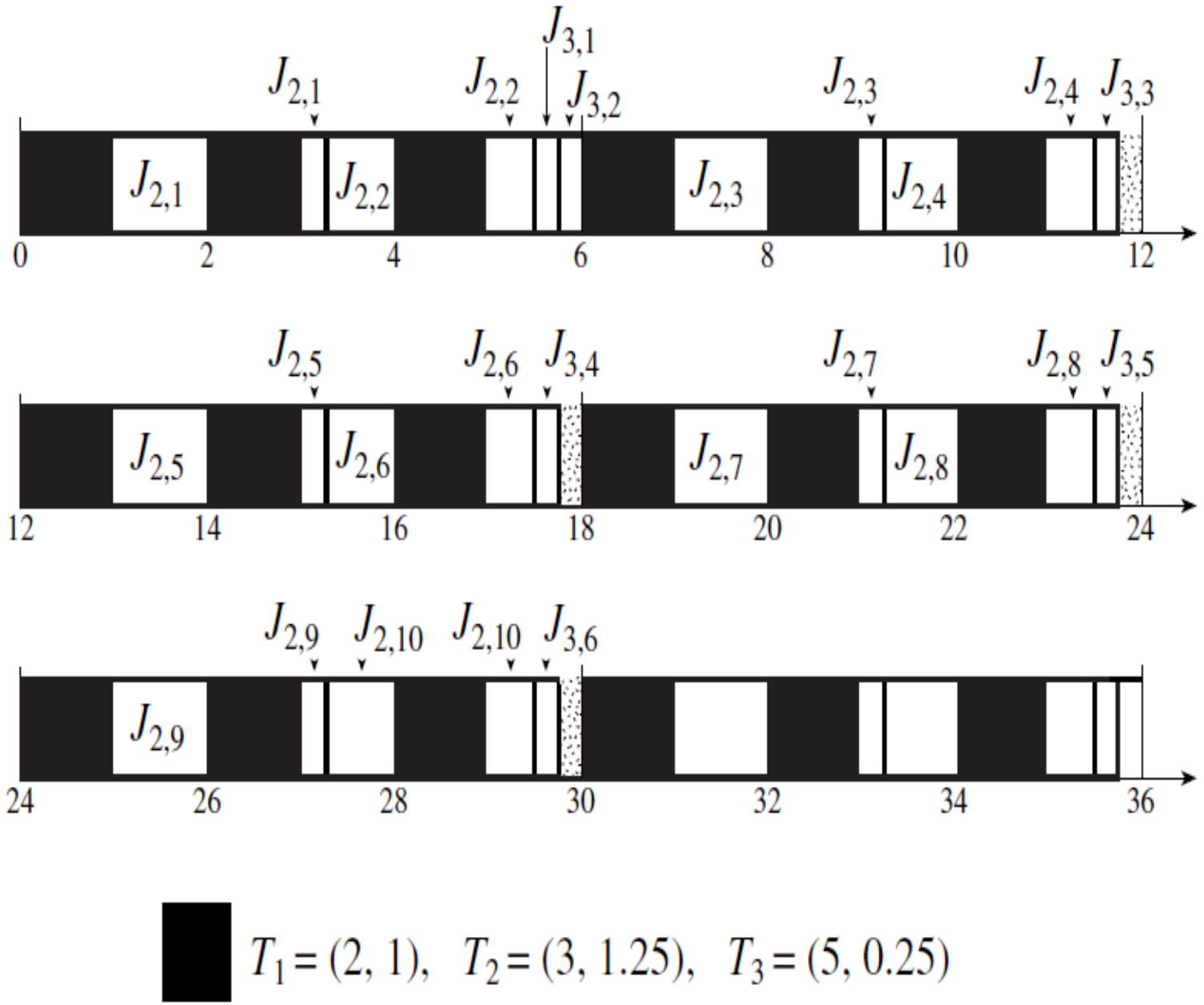
  (1) all jobs in $\mathbf{T}_i$ released before the instant have completed and

  (2) a job in $\mathbf{T}_i$ is released.

- All the jobs in $\mathbf{T}_i$ released since t_0 are complete before instant t.
- In the interval (t_0, t], the processor is busy all the time executing jobs with priorities π_i or higher.
- All the jobs executed in the busy interval are released and completed in the interval.
- A level-π_i busy interval is *in phase* if the first jobs of all tasks that have priorities equal to or higher than priority π_i and are executed in this interval have the same release time.
- Otherwise, we say that the tasks have arbitrary phases in the interval.

Example of busy interval

- Level-1 busy interval
  - always ends 1 unit time after it begins
- Level-2 busy interval
  - all the level-2 busy intervals are in phase.
  - The lengths of these intervals are all equal to 5.5.
- Level-3 busy interval
  - At 5.5, the first job in T_3 is scheduled.
  - The second job in T_3 is scheduled at 5.75.
  - The first level-3 busy interval ends at time 6.
  - The second level-3 busy interval begins at time 6.
  - This level-3 busy interval is not in phase.
  - The length of this level-3 busy interval is only 5.75. (11.75)
  - All the subsequent level-3 busy intervals in the hyperperiod have arbitrary phases.



6.6.2 General Schedulability Test

- For tasks with arbitrary relative deadlines
- The first job $J_{i,1}$ may no longer have the largest response time among all jobs in T_i .
- Examine all the jobs of T_i that are executed in the first level-π_i busy interval.
- If the response times of all these jobs are no greater than the relative deadline of T_i ,
  - T_i is schedulable; otherwise,
  - T_i may not be schedulable.

General Time-Demand Analysis Method

- Test one task at a time starting from the highest priority task T_1 in order of decreasing priority.
- Assume that all the tasks are in phase and the first level-π_i busy interval begins at time 0.
- While testing whether T_i is schedulable, consider the subset $\mathbf{T}_i$ of tasks with priorities π_i or higher.
- **(i)** If the first job of every task in $\mathbf{T}_i$ completes by the end of the first period of the task(relative deadline ≤ period), check whether the first job $J_{i,1}$ in T_i meets its deadline.
  - T_i is schedulable if $J_{i,1}$ completes in time.
  - Otherwise, T_i is not schedulable.
- **(ii)** If the first job of some task in $\mathbf{T}_i$ does not complete by the end of the first period of the task (relative deadline > period), do the following:
  - **(a)** Compute the length of the in phase level-π_i busy interval by solving the equation $t = \sum_{k=1}^{i} \lceil \frac{t}{p_k} \rceil e_k$ iteratively, starting from

    $t^{(1)} = \sum_{k=1}^{i} e_k$ until $t^{(l+1)} = t^{(l)}$ for some $l \geq 1$.

    The solution $t^{(l)}$ is the length of the level-π_i busy interval.
  - **(b)** Compute the maximum response times of all $\lceil t^{(l)}/p_i \rceil$ jobs of T_i in the in-phase level-πi busy interval in the manner described below and determine whether they complete in time.
    - T_i is schedulable if all these jobs complete in time;
    - otherwise T_i is not schedulable.

- It is easy to compute the response time of the first job $J_{i,1}$ of T_i in the first in-phase level-π_i busy interval.
- The time-demand function $w_{i,1}(t)$ is still given by the expression in the right-hand side of Eq. (6.5).
- An important difference is that the expression remains valid for all $t > 0$ before the end of the level-π_i busy interval.
- For the sake of convenience, we copy the expression here.

$$w_{i,1}(t) = e_i + \sum_{k=1}^{i-1} \left\lceil \frac{t}{p_k} \right\rceil e_k, \qquad \text{for } 0 < t \le w_{i,1}(t) \tag{6.8}$$

- The maximum possible response time $W_{i,1}$ of $J_{i,1}$ is equal to the smallest value of t that satisfies the equation $t = w_{i,1}(t)$.
- To obtain $W_{i,1}$, we solve the equation iteratively and terminate the iteration only when we find $t\,(l+1)$ equal to $t\,(l)$.
- Because U_i is no greater than 1, this equation always has a finite solution, and the solution can be found after a finite number of iterations.
- Let $W_{i,j}$ denote the maximum possible response time of the j th job in a level-π_i busy interval.
- The following lemma tells us how to compute $W_{i,j}$.
- **LEMMA 6.6.** The maximum response time $W_{i,j}$ of the j th job of T_i in an in-phase level-π_i busy period is equal to the smallest value of t that satisfies the equation

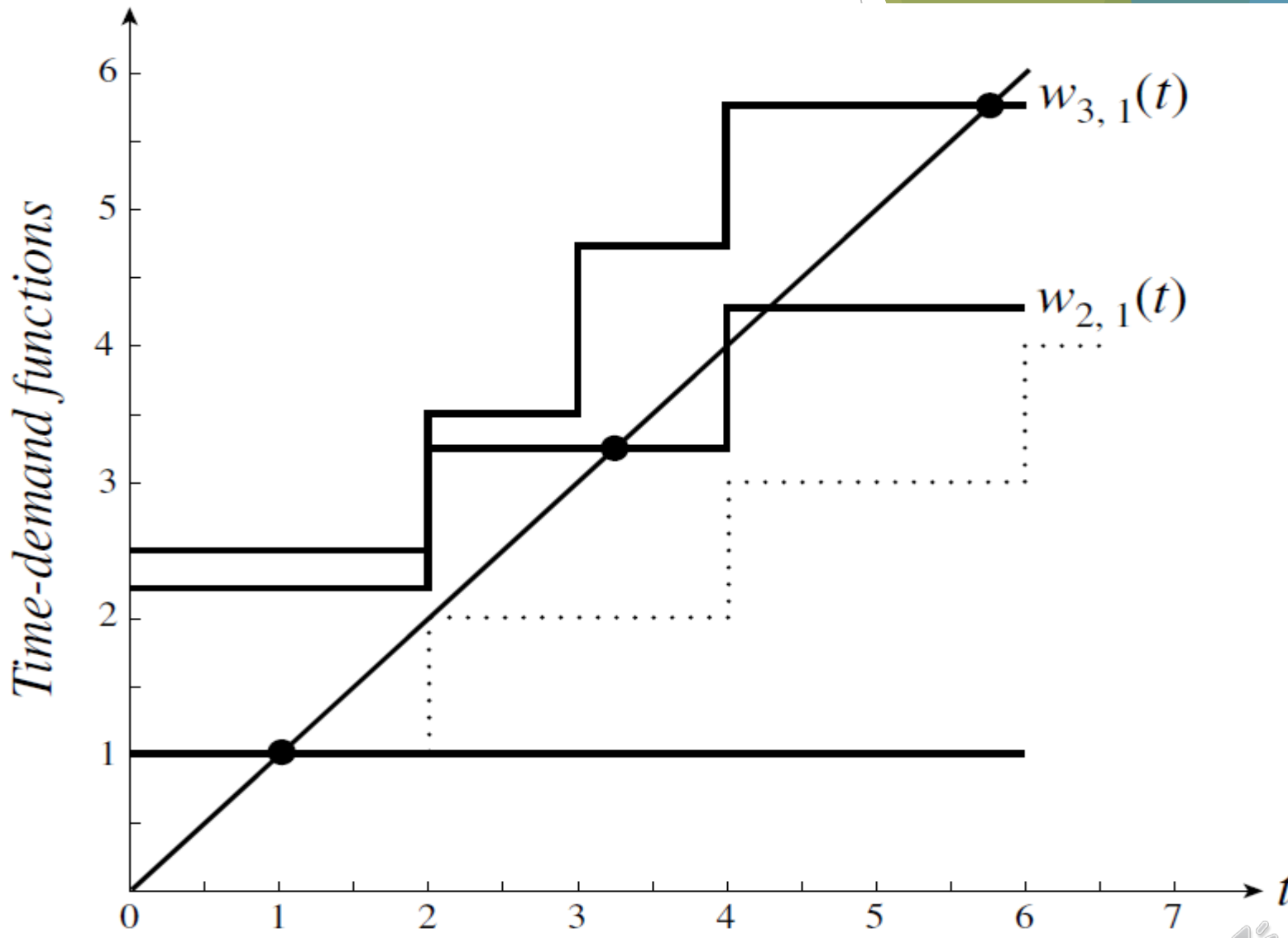
$$t = w_{i,j}(t + (j-1)p_i) - (j-1)p_i \tag{6.9a}$$

where

$$w_{i,j}(t) = je_i + \sum_{k=1}^{i-1} \left\lceil \frac{t}{p_k} \right\rceil e_k, \qquad \text{for } (j-1)p_i < t \le w_{i,j}(t) \tag{6.9b}$$

Example of time-demand analysis

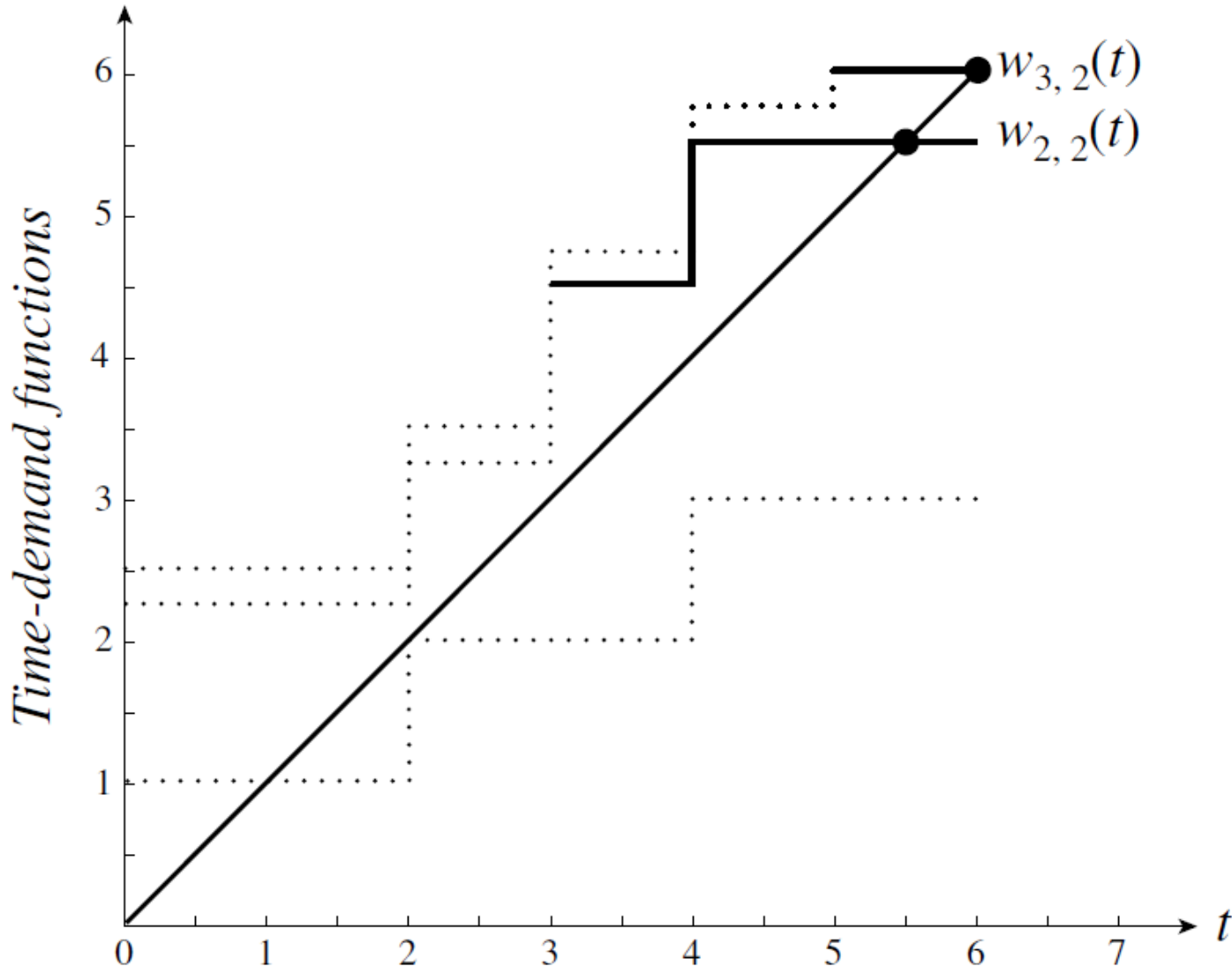
- As an example, Figure 6–13(a) shows the time-demand functions $w_{1,1}(t)$, $w_{2,1}(t)$ and $w_{3,1}(t)$ of the first jobs $J_{1,1}$, $J_{2,1}$.
- 3.25 is the response time of $J_{2,1}$;
  - $w_{2,1}(t)$ lies entirely above the supply t line from 0 to 3, $J_{2,1}$ does not complete at 3.
  - $W_{2,1}$ equals 3.25.
- 5.75 is the response time of $J_{3,1}$;
  - The first intersection of $w_{3,1}(t)$ with the supply line t.



Time-demand functions of tasks $T_1 = (2, 1)$, $T_2 = (3, 1.25)$ and $T_3 = (5, 0.25)$.

Example of time-demand analysis

- The time-demand functions $w_{2,2}(t)$ and $w_{3,2}(t)$ of the second jobs of T_2 and T_3 in the first busy intervals, respectively.
- We see that these functions are equal to t when t is equal to 5.5 and 6, respectively.
- Therefore, $J_{2,2}$ completes at $t = 5.5$, and $J_{3,2}$ completes at $t = 6$.
- Subtracting their release times, we find that $W_{2,2}$ is 2.5 and $W_{3,2}$ is 1.



Time-demand functions of tasks $T_1 = (2, 1)$, $T_2 = (3, 1.25)$ and $T_3 = (5, 0.25)$.

LEMMA 6.6. The maximum response time $W_{i,j}$ of the jth job of T_i in an in-phase level-π_i busy period is equal to the smallest value of t that satisfies the equation

$$t = w_{i,j}(t + (j-1)p_i) - (j-1)p_i \tag{6.9a}$$

where $w_{i,j}(\,)$ is given by

$$w_{i,j}(t) = je_i + \sum_{k=1}^{i-1} \left\lceil \frac{t}{p_k} \right\rceil e_k, \qquad \text{for } (j-1)p_i < t \le w_{i,j}(t) \tag{6.9b}$$

$T_1 = (2, 1)$, $T_2 = (3, 1.25)$ and $T_3 = (5, 0.25)$.

- Computationally, it is more efficient to find $W_{i,j}$ by solving Eq. (6.9) iteratively in a manner similar to Eq. (6.7).
- For example, to find $W_{2,2}$, we substitute the parameters of the tasks into Eq. (6.9) to obtain

  $t = 2 \times 1.25 + (t + 3)/2 - 3$
- Begin with the initial guess t (1) of 1.25, the execution time of $J_{2,2}$.
- Substituting t on the right-hand side of the equation by this value, we obtain

  $t^{(2)} = 2.5$ {from $2 \times 1.25 + (1.25 + 3)/2 - 3$}.
- Substitute t on the right-hand side by 2.5, we obtain $t^{(3)} = 2.5$. {from $2 \times 1.25 + (2.5 + 3)/2 - 3$}
- This allows us to terminate the iteration and conclude that $W_{2,2}$ is 2.5.
- Similarly, $W_{3,2} = 1$ is the minimum solution of the equation

  $t = 2 \times 0.25 + (t + 5)/2 + 1.25(t + 5)/3 - 5$

Alternative is the worst-case simulation approach

- Generate the schedule of the tasks in $\mathbf{T}_i$ according to the given fixed-priority algorithm and assume that the tasks are in phase.
- If there is no missed deadlines within the first level-πi busy interval, all the tasks in $\mathbf{T}_i$ are schedulable.
- Otherwise, we cannot guarantee that all tasks in $\mathbf{T}_i$ always meet their deadlines.
- To determine whether the j^{th} job in an in-phase level-π_i busy interval completes in time, check whether the inequality $w_{i,j}(t) \leq t$ is ever satisfied for some instant t in the range from $(j-1)p_i$ to $(j-1)p_i + D_i$.
- As with Eq. (6.6), only need to check at time instants which are integer multiples of p_k, for $k = 1, 2, \ldots, i$, in this range of time.

End