INFO-F404, Operating Systems II

Introduction & Uniprocessor real-time scheduling
Academic year 2013–2014

Joël Goossens

Université libre de Bruxelles

Introduction

A) Introduction

The Speaker

Joël Goossens
Faculté des Sciences
Département d'Informatique
Campus de la Plaine
Building NO, 8th floor, room 2N8.114
E-mail: Joel.Goossens@ulb.ac.be
URL: parts.ulb.ac.be/goossens

Teaching organization

- Lectures: 2 hours/week
- Exercise sessions: Mr Nikita Veshchikov
- Assignments (A_1&A_2)
- Oral exam (O)
- Final note

$$\begin{cases} O & \text{if } O < 10 \\ 3 * A_1/20 + 2 * A_2/20 + 15 * O/20 & \text{else.} \end{cases}$$

Objectives of the course

- Introduction to the theoretical foundations, algorithms and tools for the design of real-time and embedded systems
- Introduction to parallel and distributed programming

Uniprocessor real-time systems

B) Uniprocessor real-time systems

Course material

- Slides available on `http://uv.ulb.ac.be`
- For the first lesson: `http://goo.gl/WyXUWM`
- Important references
  - Jane W.S. Liu, "*Real-Time Systems*", Prentice Hall, 2000
  - Joseph Y-T. Leung, "*Handbook of Scheduling: Algorithms, Models, and Performance analysis*", Chapman Hall, 2004. Specifically Chapter 28, "Scheduling Real-Time Tasks: Algorithms and Complexity" (Sanjoy Baruah & J. Goossens).
  - J. Goossens, "*Scheduling of Hard Real-Time Periodic Systems with various kinds of deadline and offset constraints*", PhD Thesis, ULB, 1999.
- Reference in French:
  - N. Navet, "*Systèmes temps réel, volume 2*", Hermès, 2006. Specifically Chapter 1, "Ordonnancement temps réel monoprocesseur" (P. Richard & F. Ridouart) and Chapter 2, "Ordonnancement temps réel multiprocesseur" (J. Goossens).

Real-time systems — A first definition I

- Real-time systems are defined as systems in which the correctness of the system depends not only on the logical result of computations, but also on the **time** at which the results are produced
- Typically, job parameters include a deadline
- Applications include air traffic control, power plant control, multimedia communication, robotics, embedded systems, real-time stock quotes...

Real-time systems — A first definition II

We can distinguish between at least three kinds of real-time constraints:

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

Existing Real-Time Operating Systems (RTOS)

Modeling applications I

Applications are modeled using the concept of **jobs** and/or the concept of **tasks**.

Definition 1 (Job)

A **Job** j is characterized by the tuple (a, e, d), symbolizing a release time a, a computation requirement e and an absolute deadline d with the interpretation that in the interval $[a, d)$ the job must receive e CPU units. A real-time instance is a collection of jobs $J = \{j_1, j_2, \ldots\}$.

Very often, the critical portion of the software is characterized by **recurrent** operations. More specifically, we shall consider periodic and/or sporadic tasks.

Modeling applications II

Definition 2 (Periodic Task)

A periodic task is characterized by the tuple (O_i, T_i, D_i, C_i), where

- O_i (the offset) corresponds to the release time of the first job of the task
- C_i (the computation time) corresponds to the worst-case execution requirement of the task
- D_i (the relative deadline) corresponds to the time-delay between a job release and the corresponding deadline of the task. A job released at time t must be completed before or at time $t + D_i$.
- T_i (the period) corresponds to the duration between two consecutive task releases

Modeling applications III

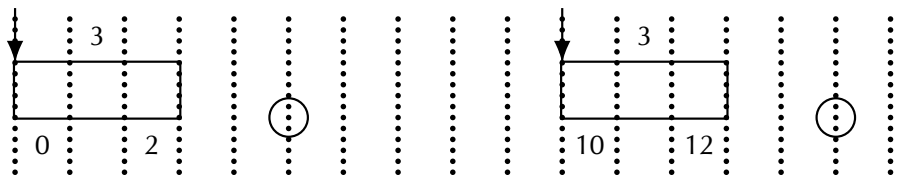
Job $J_{i,k}$

- A periodic task $\tau_i = (O_i, T_i, D_i, C_i)$ induces an infinite collection of jobs: $J_{i,1}, J_{i,2}, \ldots$
- $J_{i,k}$ represents the k^{th} job of τ_i
- Each job $J_{i,k}$ has the same (worst-case) computation requirement C_i
- $J_{i,k}$ is released at instant $O_i + (k-1) \cdot T_i$
- with an absolute deadline equal to $O_i + (k-1) \cdot T_i + D_i$

Modeling applications IV

A periodic task

$$\tau_1 = (O_1 = 0, T_1 = 10, C_1 = 3, D_1 = 5)$$



$$U(\tau_1) = 3/10$$

Modeling applications V

Definition 3 (Sporadic Task)

Sporadic tasks are quite similar to periodic tasks, the only difference being that the period of a sporadic task denotes the **minimum** inter-arrival time instead of the exact one.

A periodic/sporadic system is composed of a **finite** set of periodic/sporadic tasks $\tau = \{\tau_1, \ldots, \tau_n\}$.

Definition 4 (Utilization)

The task **utilization** is defined by $U(\tau_i) \stackrel{\text{def}}{=} C_i/T_i$. The system utilization is the sum of all task utilizations: $U(\tau) \stackrel{\text{def}}{=} \sum_{\tau_i \in \tau} U(\tau_i)$

Different kinds of periodic tasks

From a theoretical as well as a practical point of view, we distinguish the following kinds of tasks:

implicit-deadline the deadline corresponds to the period, i.e. $D_i = T_i \ \forall i$. Each job must finish before the next arrival of the task.

constrained-deadline the deadline is explicit and not larger than the period, i.e. $D_i \leqslant T_i \ \forall i$.

arbitrary-deadline the general case: no constraint exists between the period and the deadline.

synchronous the first job of every task is released simultaneously, i.e. $O_i = 0 \ \forall i$.

asynchronous not synchronous.

Scheduling I

- Scheduling disciplines are algorithms used for distributing resources among parties which simultaneously and asynchronously request them. Scheduling disciplines are used in routers (to handle packet traffic) as well as in operating systems (to share CPU time among both threads and processes), disk drives (I/O scheduling), printers (print spooler), most embedded systems, etc.
- The main purposes of scheduling algorithms are to minimize resource starvation and to ensure fairness amongst the parties utilizing the resources. Scheduling deals with the problem of deciding which of the outstanding requests is to be allocated resources. There are many different scheduling algorithms.
- In real-time environments, such as embedded systems for automatic control in industry (e.g. robotics), the scheduler must also ensure that processes meet deadlines; this is crucial for keeping the system stable or safe.

Scheduling II

- The CPU scheduler decides which of the ready, in-memory processes are to be executed (allocated a CPU) next following a clock interrupt (an I/O interrupt, an operating system call or another form of signal).
- This scheduler can be preemptive, implying that it is capable of forcibly removing processes from a CPU when it decides to allocate that CPU to another process. It may also be non-preemptive.

FTP, FJP & DP Schedulers

We distinguish between three kinds of scheduling algorithms:

- **Fixed Task Priority** (FTP): the scheduler does an offline assignment of a unique priority to each task. During system execution, the priority of each job is inherited from its task.
- **Fixed Job Priority** (FJP): a fixed and unique priority is assigned to each job during system execution. Two jobs of a same task may have distinct priorities.
- **Unrestricted Dynamic Priority** (DP): no restrictions are placed on the priorities that may be assigned to jobs. A job may have different priority levels during its lifetime.

Chapter Hypotheses

Unless we explicitly state the opposite, we make the following assumptions in this chapter:

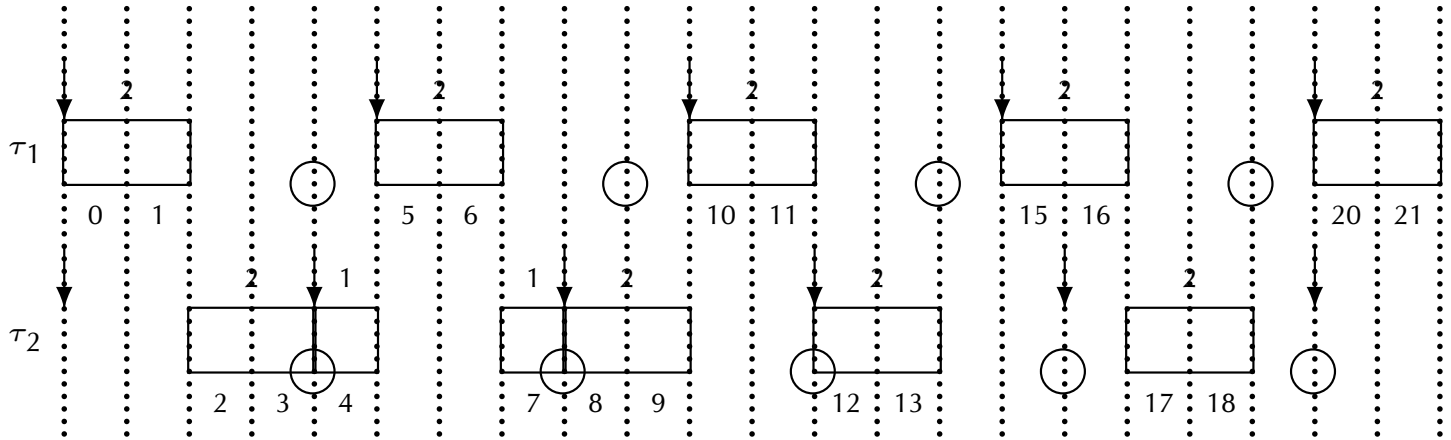
- We consider uniprocessor systems
- We consider hard real-time systems
- We consider preemptible tasks/jobs
- Preemption delays are negligible
- The scheduler is work-conserving: the CPU cannot be idle while there exist active jobs
- Tasks are independent: save for the CPU, there is no other common resource shared among tasks, nor any kind of precedence constraint between tasks

An FTP Scheduler $\tau_1 \succ \tau_2$

2 periodic tasks:

$$\tau = \{\tau_1 = (T_1 = 5, D_1 = 4, C_1 = 2), \tau_2 = (T_2 = 4, D_2 = 4, C_2 = 2)\}$$

$\tau_1 \succ \tau_2$ (τ_1 has the highest priority), $U(\tau) = 2/5 + 1/2 = 9/10$



Additional Definition & Observations

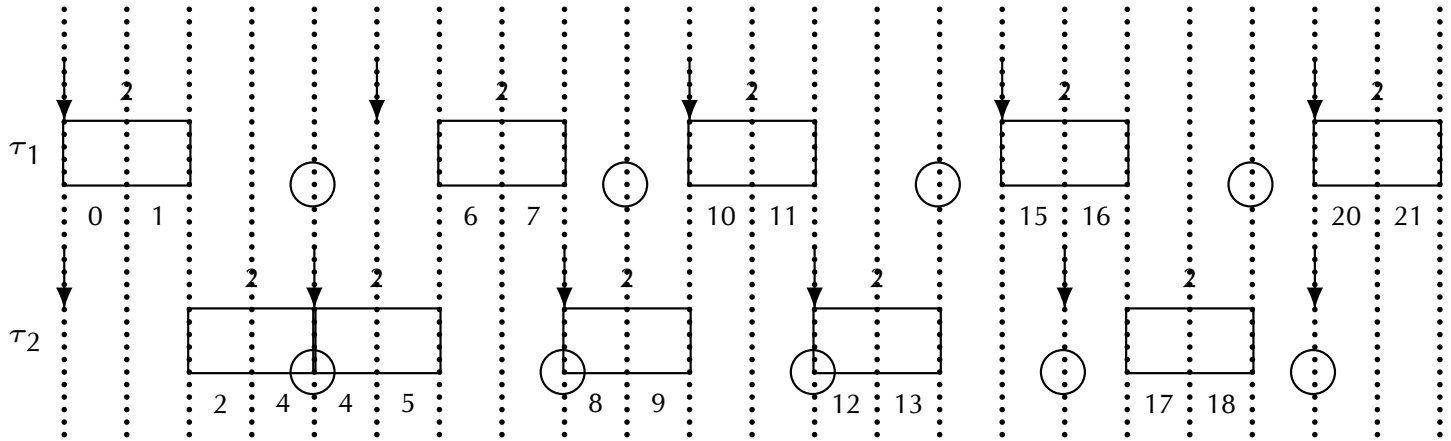
Definition 5 (Job Response Time)

The response time of a job is the completion time minus the release time

- priorities are fixed at task level
- tasks are preemptible
- the schedule is periodic
- the system is schedulable
- the response times of τ_2: 4,4,2,2,3

An FJP Scheduler: EDF

Priority is based on the **absolute** deadline of the job: earliest deadline first (EDF)



Observations

- priorities are dynamic at task level, fixed at job level
- the schedule is periodic
- the system is schedulable
- Many EDF schedules exist for this task set

A first necessary condition

There is no hope to schedule a task set with a utilization larger than 1.

Lemma 6

*Let τ be a periodic (or sporadic) task set, then $U(\tau) \leqslant 1$ is a **necessary** condition for system schedulability.*

FTP Scheduling — Basics

- We will consider the scheduling of periodic (or sporadic) tasks using a Fixed Task Priority Scheduler
- $\tau_i \succ \tau_j$ means that τ_i has a higher priority than τ_j.
- We assume that each job generated by τ_i receives the priority of τ_i and remains unchanged during the job's execution.
- WLOG we consider the following total order: $\tau_1 \succ \tau_2 \succ \cdots \succ \tau_n$

FTP-Priority Assignment RM I

- The RM (Rate Monotonic) priority assignment is defined for synchronous and implicit deadline systems.
- The priority rule is the following: the lower the period, the higher the priority. Ties can be resolved arbitrarily but in a consistent (deterministic) manner.

FTP-Priority Assignment RM II

- The offline time-complexity of the assignment is equivalent to sorting the task set according to the periods: $\mathcal{O}(n \log n)$
- We will show that RM is optimal, but first we will give some basic definitions and properties.

Definition 7 (FTP-feasibility)

A periodic task set is said to be **FTP-feasible** if a schedulable FTP-priority assignment exists.

Definition 8 (FTP-optimality)

An FTP-priority assignment is FTP-optimal if the assignment schedules **all** FTP-feasible task sets.

Critical Instant I

Definition 9 (Critical instant)

A critical instant of a task τ_i is an instant such that: the job of τ_i released at this instant has the **maximum response time** of all the jobs in τ_i.

Informally, a critical instant of τ_i represents a worst-case scenario from the standpoint of τ_i.

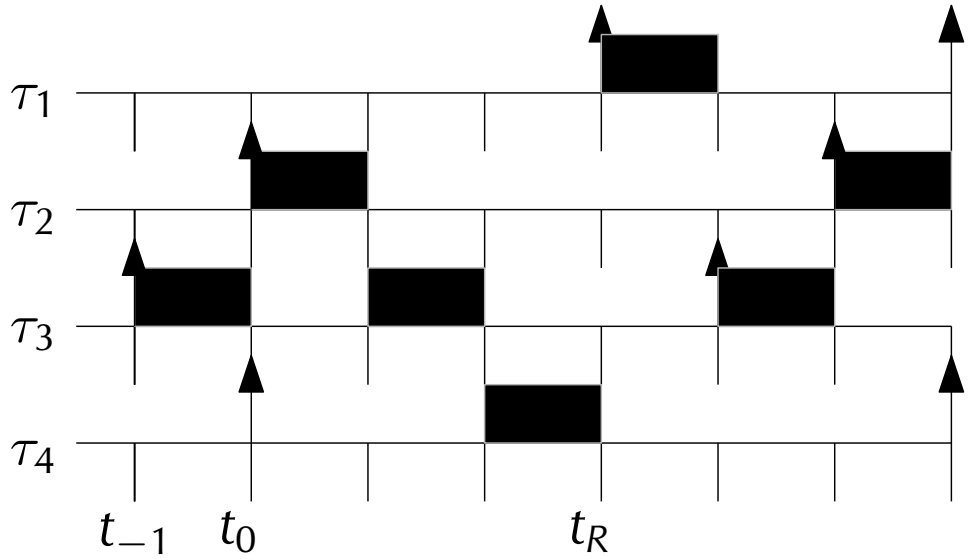
Lemma 10 ([Liu, 2000])

A critical instant of any task τ_i occurs when one of its jobs $J_{i,k}$ is released at the same time as a job of every higher priority task.

Proof. (This proof corrects a flaw published in [Liu, 2000])

- Let t' be the release time of $J_{i,k}$
- Let t_{-1} be the latest idle instant for $\tau_1, \ldots, \tau_i$ at or before t'
- Let t_R denote the instant when $J_{i,k}$ completes

Critical Instant II



- If we (artificially) redefine the release time of $J_{i,k}$ to be t_{-1}, then t_R remains unchanged (but the response time of $J_{i,k}$ may increase)

Critical Instant III

- Assume τ_i releases a job at t_{-1} and some higher priority task τ_j does not. If we "left-shift" τ_j so that its first job is released at time t_{-1}, then the response time of $J_{i,k}$ is not decreased
- Starting with τ_1, let us "left-shift" any task whose first job is released after t_{-1} so that its first job released at t_{-1}
- With each shift, the response time of τ_i does not decrease
- We have constructed a portion of the schedule that is identical to the one which occurs at time t_{-1} (when $\tau_1, \ldots, \tau_i$ all release jobs synchronously)
- Moreover, the response time of τ_i is at least the one of τ_i's job released at t'
- This shows that asynchronous releases cannot cause larger response time than synchronous releases
- Thus, if a job is released at the same time as jobs of all higher priority tasks, that moment is a critical instant

Critical Instant IV

- It remains to show that the left-shift does not impact on t_{-1}:
  - No job shifts past t_{-1}
  - Every job that started after t_{-1} continues to do so after the shifts, by construction
  - Similarly, any job that started before t_{-1} continues to do so after the shifts
  - Consider any interval $[x, t_{-1})$, $x < t_{-1}$. No job shifts into this interval, as no left-shifts cross t_{-1}. Therefore, the total demand during $[x, t_{-1}]$ cannot increase. Therefore, the last completion time prior to t_{-1} cannot be delayed ■

Work-Conservation I

Lemma 11

Let τ be a periodic task system, and let S_1 and S_2 be two schedules of work-conserving schedulers for τ. Schedule S_1 is idle at instant t iff schedule S_2 is idle at instant t, $\forall t \geqslant 0$.

Work-Conservation II

Proof. The proof is made by induction. Assume schedules S_1 and S_2 have an idle unit at instant $t_0 - \varepsilon$ (infinitesimal ε —arbitrarily close to, but greater than zero) and have idle units identical for all instants before t_0. Let t_1 be the first idle unit (WLOG in S_1) strictly after t_0; assume the CPU idles during $[t_1, t_2)$. Since S_1 is work-conserving, we know that all jobs released before t_2 have completed their execution. Since S_2 is work-conserving as well, we must have that the same work is executed before t_1 (possibly in a different order). Consequently, the CPU must be idle in $[t_1, t_2)$ in S_2 as well. ■

RM Optimality I

Theorem 12

RM *is an FTP-optimal priority assignment for synchronous and implicit deadline tasks.*

Proof. Let $\tau = \{\tau_1, \ldots, \tau_n\}$ be a periodic (or sporadic) implicit deadline synchronous task set. We must prove that if a feasible FTP-priority assignment exists, then the RM priority assignment is feasible as well. Suppose the feasible FTP-assignment corresponds to $\tau_1 \succ \tau_2 \succ \cdots \succ \tau_n$. Let τ_i and τ_{i+1} be two adjacent priorities with $T_i \geqslant T_{i+1}$. Let us exchange the priorities of τ_i and τ_{i+1}: if the task set is still schedulable, since any rate monotonic priority assignment can be obtained from any priority ordering by a sequence of such priority exchanges, we may deduce that any rate monotonic priority assignment is also feasible (i.e. using the bubble sort algorithm).
To show that such a priority exchange is feasible, we have 4 cases to consider.

RM Optimality II

1. The priority exchange does not modify the schedulability of the tasks with a higher priority than τ_i (i.e. $\tau_1, \ldots, \tau_{i-1}$).
2. Obviously, task τ_{i+1} remains schedulable after the priority exchange, since it may use all the free slots left by $\{\tau_1, \tau_2, \ldots, \tau_{i-1}\}$ instead of only those left by $\{\tau_1, \tau_2, \ldots, \tau_{i-1}, \tau_i\}$.
3. Assuming that the requests of τ_i remain schedulable, from Lemma 11 we have that the scheduling of each task τ_k, for $k = i+2, i+3, \ldots, n$ is not altered since the idle periods left by higher priority tasks are the same.
4. Hence, we must only verify that τ_i also remains schedulable. Using Lemma 10, we can restrict this question to the first request of task τ_i. Let r_{i+1} be the response time of the first request of τ_{i+1} before the priority exchange. Feasibility implies that $r_{i+1} \leqslant D_{i+1}$. It is not difficult to see that during the interval $[0, r_{i+1})$, the CPU (when left free by higher priority tasks) is assigned first to the (first) request of τ_i then to the (first) request of τ_{i+1} (the latter is not interrupted by

RM Optimality III

subsequent requests of τ_i since $T_i > T_{i+1} = D_{i+1} \geqslant r_{i+1}$). Hence, after the priority exchange, the CPU allocation is exchanged between τ_i and τ_{i+1}, and it follows that τ_i ends its computation at time r_{i+1} and thus meets its deadline since $r_{i+1} \leqslant D_{i+1} = T_{i+1} \leqslant T_i = D_i$. ■

Response Time of the First Task Request I

From Lemma 10, for synchronous constrained deadline systems, only the response time of the **first** request of each task must be considered to conclude feasibility.
In the following, r_i^1 denotes the response time of the first request of τ_i.

Theorem 13

A synchronous constrained deadline system is FTP-schedulable iff

$$r_i^1 \leqslant D_i \qquad \forall i$$

Audsley & Tindell [Audsley, 1991] proved that this value is the smallest positive solution to the equation:

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

Indeed, the interval $[0, r_i^1)$ includes $\left\lceil \frac{r_i^1}{T_j} \right\rceil$ higher priority requests of τ_j.

Response Time of the First Task Request II

Notice that r_i^1 occurs on both sides of the equation. The minimal solution can be found by fixed-point iteration:

$$\begin{cases} w_0 & \stackrel{\text{def}}{=} & C_i, \text{ (initialization)} \\ w_{k+1} & \stackrel{\text{def}}{=} & C_i + \sum_{j=1}^{i-1} \left\lceil \frac{w_k}{T_j} \right\rceil C_j \text{ (iteration).} \end{cases}$$

The iteration proceeds until $w_k = w_{k+1}$ or $w_k > D_i$.

Schedulable Utilization of RM I

Theorem 14 ([Leung, 2004])

A set of n implicit deadline synchronous periodic tasks can be feasibly scheduled with RM *if*

$$U(\tau) \leqslant n(\sqrt[n]{2} - 1)$$

Notice that $0.69 < \ln 2 < \cdots < 0,75 < 0,77 < 0.83$, hence:

Corollary 15

A set of n implicit deadline synchronous periodic tasks can be feasibly scheduled with RM *if*

$$U(\tau) \leqslant 0.69$$

DM FTP-Priority Assignment

- The DM(Deadline Monotonic) FTP-priority assignment is defined/used for synchronous **constrained** deadline task sets.
- The priority rule is the following: the lower the relative deadline, the higher the priority. Ties can be resolved arbitrarily but in a consistent (deterministic) manner.
- Notice that RM is a particular case of DM.

FTP-Optimality of DM

Theorem 16

DM *is an FTP-optimal priority assignment for synchronous and constrained deadline tasks.*

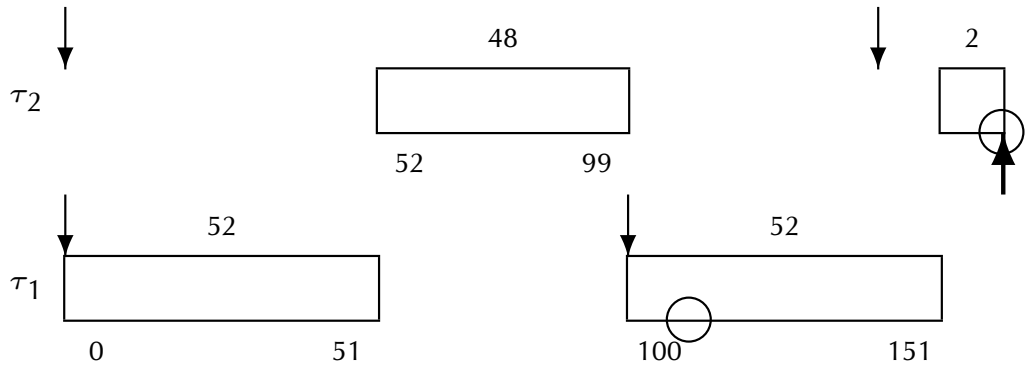
Proof. The same argument used to prove RM FTP-optimality (Theorem 12) can be used here. ■

Note that Lemma 10 concerns synchronous constrained deadline tasks. Consequently, the lemma can be applied and we can check the schedulability of DM by computing the response time of the first request of each task.

$$r_i^1 \leqslant D_i \quad \forall i$$

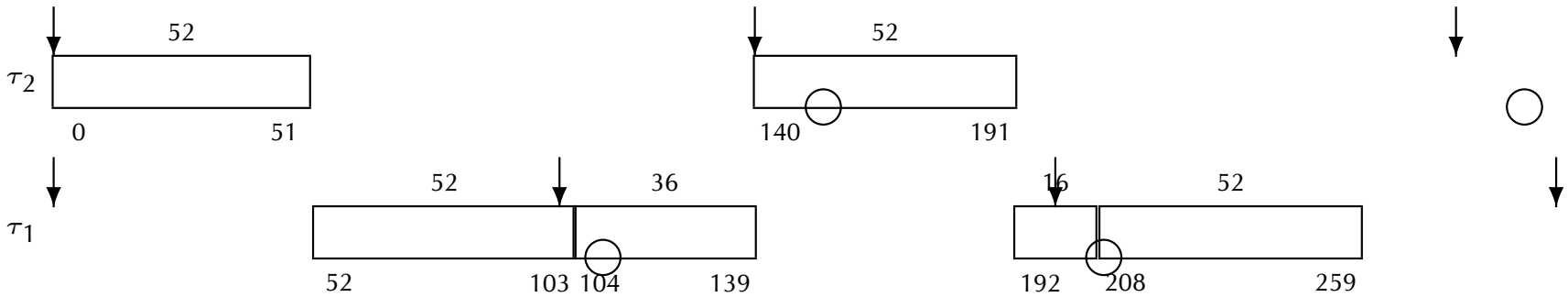
Arbitrary Deadlines I

- To summarize what we've seen so far, RM concerns (synchronous) implicit deadline tasks, whereas DM concerns (synchronous) constrained deadline tasks.
- We will now consider (synchronous) **arbitrary** deadline tasks.
- First, we show that neither RM nor DM are optimal for such systems. For instance, consider the system $\tau = \{\tau_1 = \{C_1 = 52, T_1 = 100, D_1 = 110\}, \tau_2 = \{C_2 = 52, T_2 = 140, D_2 = 154\}\}$. In this case, DM and RM are identical: $\tau_1 \succ \tau_2$. Using that FTP-priority assignment, the first job of τ_2 misses its deadline at instant 154:



Arbitrary Deadlines II

- However, the system is FTP-feasible since the other FTP-assignment ($\tau_2 \succ \tau_1$) is schedulable. We will see that for such systems, is not sufficient to consider the first request of each task.



Arbitrary Deadlines — Important Phenomena

- Several jobs of the **same** task can be active simultaneously. In that case, the scheduler considers only the older job, i.e. FIFO is used at task level.
- In our example, this is the case during the time-interval [100, 102)
- Second phenomenon: the response time of the first request is not necessarily the maximum (see the previous schedule where the response time of the first job is 104 while the second one is larger **108**)

Feasibility Interval I

Definition 17 (Feasibility Interval)

A **feasibility interval** is a **finite** interval of time such that, if no deadline is missed considering only the jobs released within this interval, then we can conclude that no deadline will ever be missed during the system's lifetime (i.e. the task set is feasible).

Feasibility Interval II

Theorem 18

For synchronous constrained deadline tasks, $[0, \max\{D_i \mid i = 1, \ldots, n\})$ *is a feasibility interval*

Proof. A direct consequence of Lemma 10 ■

- For (synchronous) arbitrary deadline tasks, we will show that the first busy period is a feasibility interval.
- But first, a few additional definitions.

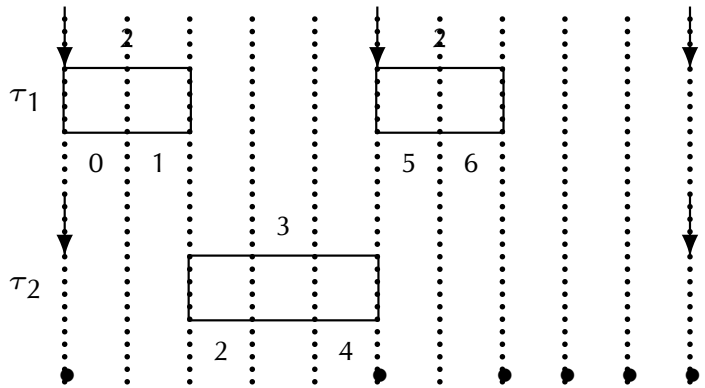
Definition 19 (Idle Point)

$x \in \mathbb{N}$ is an idle processor point of the schedule of a system if all requests occurring strictly before x have completed their execution before or at time x.

- If the CPU is idle in the interval $[a, b)$, all the points in that interval are idle points.

Feasibility Interval III

- Consider the system $\{\tau_1 = \{C_1 = 2, T_1 = 5\}, \tau_2 = \{C_2 = 3, T_2 = 10\}\}$ (idle points are pictured using $\bullet$). The system is idle in the interval $[7, 10)$; $7, 8, 9, 10$ are idle points ; 10 is an idle point which coincides with new job arrivals.



Feasibility Interval IV

Definition 20 (Elementary Busy Period)

An **elementary busy period** is a time-interval $[a, b)$ such that a and b are idle points and the interval (a, b) does not contain any idle points.

Definition 21 (Level-i Busy Period)

A level-i busy period is an elementary busy period considering the scheduling of tasks $\{\tau_1, \ldots, \tau_i\}$ where τ_i executes at least one job.

Theorem 22

*The largest response time for a request of task τ_i occurs during the **first** level-i busy period $[0, \lambda_i)$ in the synchronous case, λ_i being the first idle point (after 0) in the synchronous schedule of the task subset $\{\tau_1, \ldots, \tau_i\}$, and $[0, \lambda_i)$ is the largest level-i busy period.*

Proof. Let $[a, b)$ be a level-i busy period considering the scheduling of the task subset $\{\tau_1, \ldots, \tau_i\}$. Let $a + \Delta_j$ be the instant of the first request of τ_j after instant a ($\Delta_j \geqslant 0$). By definition, $\Delta_k = 0$ for at least one k. Suppose first that $\Delta_i > 0$. Only tasks with a higher priority than τ_i

Feasibility Interval V

execute in $[a, a + \Delta_i)$. Thus, if we decrease Δ_i, each job of τ_i in the interval $[a, b)$ will be completed at the same instant, thus increasing the response time. Therefore, the maximum case corresponds to $\Delta_i = 0$. Now, if $\Delta_j > 0$ ($j < i$), then decreasing Δ_j increases (or does not change) the demand of higher priority jobs in $[a, b)$. This fact increases (or does not change) the length of the busy period. Consequently, the worst case corresponds to $\Delta_1 = \Delta_2 = \cdots = \Delta_i = 0$. ■

Feasibility Interval VI

Theorem 23

For synchronous arbitrary deadline systems, $[0, \lambda_n)$ *is a feasibility interval.*

Proof. A direct consequence of $\lambda_1 < \lambda_2 < \cdots < \lambda_n$. ■

λ_n is the smallest solution to the equation

$$\lambda_n = \sum_{i=1}^{n} \left\lceil \frac{\lambda_n}{T_i} \right\rceil C_i \tag{2}$$

and can be computed through fixed-point iteration.

Feasibility Interval VII

$$w_0 \stackrel{\text{def}}{=} \sum_{i=1}^{n} C_i,$$

$$w_{k+1} \stackrel{\text{def}}{=} \sum_{i=1}^{n} \left\lceil \frac{w_k}{T_i} \right\rceil C_i.$$

Feasibility Interval for asynchronous constrained deadline systems I

Unfortunately, feasibility intervals are larger for asynchronous systems. We first introduce a few additional definitions.

$$P \stackrel{\text{def}}{=} \text{lcm}\{T_i \mid i = 1, \ldots, n\} \qquad \textbf{hyper-period}$$

and

$$O_{\max} \stackrel{\text{def}}{=} \max\{O_i \mid i = 1, \ldots, n\}$$

Theorem 24 ([Leung and Whitehead, 1982])

For asynchronous constrained deadline systems, $[O_{\max}, O_{\max} + 2P)$ *is a feasibility interval.*

Feasibility Interval for asynchronous constrained deadline systems II

We will now present an improved interval. First, observe that (feasible) schedules are periodic:

Theorem 25

Feasible schedules of periodic tasks are periodic.

Proof. For any natural instant time $t > O_{\max}$, we denote the configuration of a feasible schedule by the tuple $\{(\varepsilon_i, \delta_i) | 1 \leqslant i \leqslant n\}$, where ε_i is the cumulative CPU time used by the current job of τ_i, and δ_i is the time elapsed since its last request. Since the configuration of the schedule S at time $t+1$ is unequivocally determined by the configuration at time $t \geqslant 0$ and the fact that $\forall i \in [1, n]\ 0 \leqslant \varepsilon_i(t) \leqslant C_i$, $0 \leqslant \delta_i(t) < T_i$, there are finitely many possible configurations and we may find two instants $t_1, t_2 (t_1 < t_2)$ with the same configuration. Hence, from t_1, the schedule will repeat periodically (with a period dividing $t_2 - t_1$). ∎

Feasibility Interval for asynchronous constrained deadline systems III

Theorem 26

For any asynchronous constrained deadline system ordered by decreasing priorities, let S_i be inductively defined by $S_1 = O_1$, $S_i = O_i + \lceil \frac{(S_{i-1}-O_i)^+}{T_i} \rceil T_i$ $(i = 2, 3, \ldots, n)$; then, if the schedule is feasible up to $S_n + P$, with $x^+ = \max\{x, 0\}$, it is feasible and periodic from S_n with a period of P.

Proof. The proof is made by induction on n. The property holds in the trivial case where $n = 1$: the schedule for τ_1 is periodic of period T_1 from the first release of τ_1 ($S_1 = O_1$). Let us now assume that the property is true up to $i - 1$ and the schedule of the first i tasks is feasible up to $S_i + P_i$, with $P_i = \text{lcm}\{T_j \mid j = 1, \ldots, i\}$. S_i is the first release of task τ_i after (or at) S_{i-1}; hence $S_i \geqslant S_{i-1}$, $P_i \geqslant P_{i-1}$, $S_i + P_i \geqslant S_{i-1} + P_{i-1}$ and by induction hypothesis, the schedule for the task subset $\{\tau_1, \ldots, \tau_{i-1}\}$ is feasible and periodic from S_{i-1} with period P_{i-1}. Since tasks are ordered by priority, the periodicity of the first ones is unchanged by the

Feasibility Interval for asynchronous constrained deadline systems IV

requests of task τ_i, and the schedule repeats at time $S_i + \text{lcm}\{P_{i-1}, T_i\}$. Hence, for the task set $\{\tau_1, \ldots, \tau_i\}$, the schedule is feasible and repeats from S_i with period P_i. ■

Feasibility Interval for asynchronous constrained deadline systems V

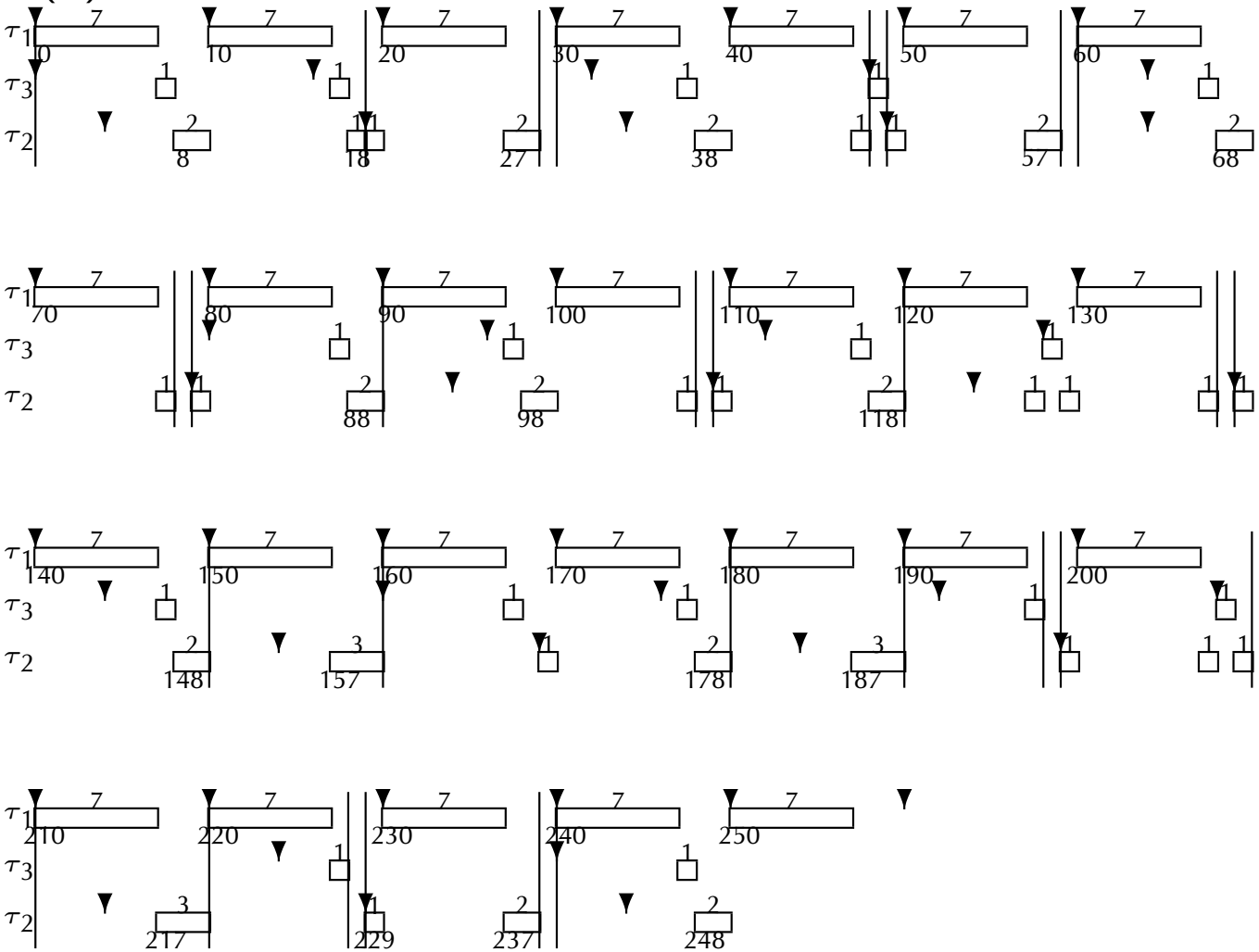
It follows that $[0, S_n + P)$ is a feasibility interval. The speedup in comparison with Theorem 24 is 2.

Illustrating S_n I

3 periodic tasks
$(T_1 = 10, C_1 = 7, O_1 = 0; T_2 = 15, C_2 = 1, O_2 = 4; T_3 = 16, C_3 = 3, O_3 = 0)$,

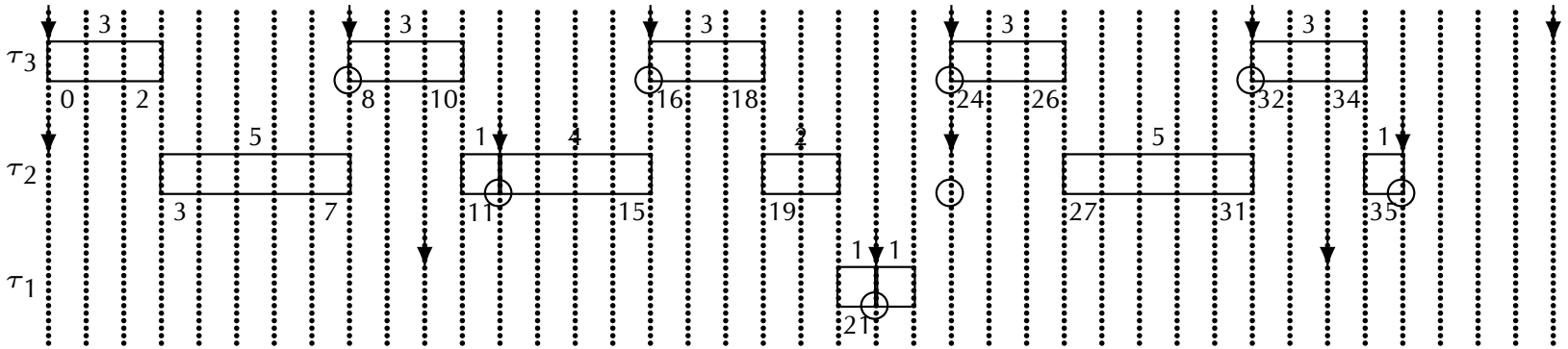
Illustrating S_n II

$U(\tau) = 0.95417$



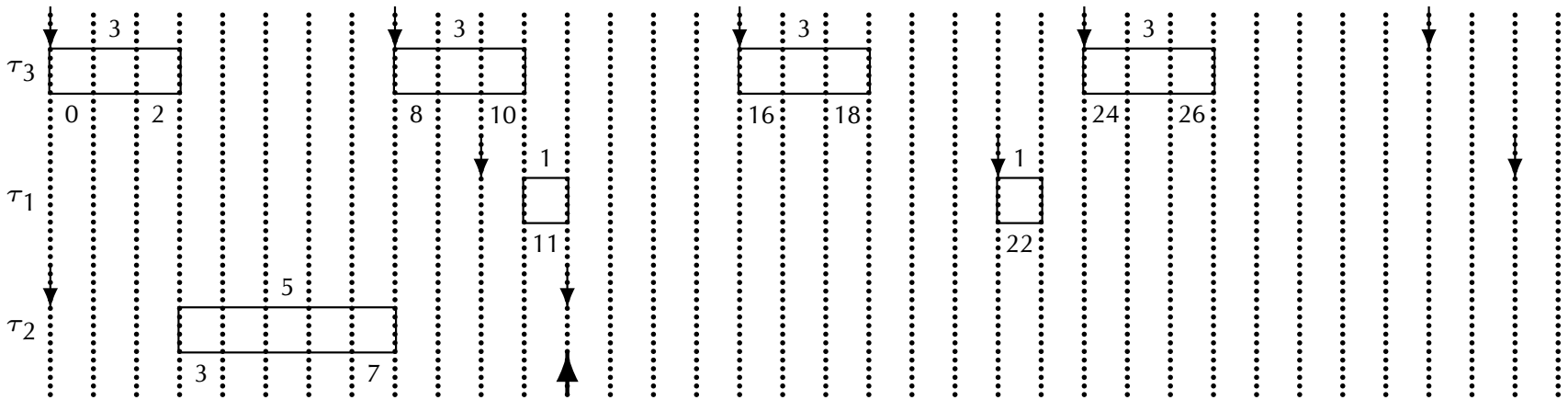
Optimal FTP assignment for asynchronous constrained deadline systems I

Unfortunately, neither DM nor RM are optimal. Consider the system $\tau_1 = \{C_1 = 1, T_1 = D_1 = 12, O_1 = 10\}$, $\tau_2 = \{C_2 = 6, T_2 = D_2 = 12, O_2 = 0\}$, $\tau_3 = \{C_3 = 3, T_3 = D_3 = 8, O_3 = 0\}$. The system is FTP-schedulable with $\tau_3 \succ \tau_2 \succ \tau_1$.



Optimal FTP assignment for asynchronous constrained deadline systems II

The system is not schedulable with RM (nor, equivalently, DM) $\tau_3 > \tau_1 > \tau_2$.



Audsley Algorithm I

A first naive but optimal priority assignment consists in considering all possible ($n!$) FTP-priority assignments.
Audsley in [Audsley et al., 1993] proposed an efficient algorithm which considers, in the worst-case, $\mathcal{O}(n^2)$ FTP-priority assignments. The idea is based upon the following property.

Audsley Algorithm II

Definition 27 (Lowest-priority viable)

A task τ_i is said to be **lowest-priority viable** iff all its jobs meet their deadline when:

- task τ_i has the lowest priority
- the other tasks have any higher priority
- when scheduling tasks $\tau_j \neq \tau_i$, the scheduler considers deadlines as soft, i.e. continues to schedule until completion

Audsley Algorithm III

Theorem 28

Suppose that τ_i is lowest-priority viable. Then, there exists a feasible FTP-assignment for τ iff there exists an FTP-priority assignment for $\tau \setminus \{\tau_i\}$.

Proof. First, we denote the task/priority assignment with $p(\tau_a) = b$, meaning that task τ_a has priority b.
Suppose that the following priority assignment p is feasible for τ:

$$p(\tau_1) = 1, p(\tau_2) = 2, \ldots, p(\tau_i) = i, p(\tau_{i+1}) = i + 1, \ldots, p(\tau_n) = n$$

then a second priority assignment is feasible as well:

$$p'(\tau_1) = 1, \ldots, p'(\tau_{i+1}) = i, p'(\tau_{i+2}) = i+1, \ldots, p'(\tau_n) = n-1, p'(\tau_i) = n$$

Since τ_i is lowest-priority viable, we can assign the lowest priority to τ_i $p'(\tau_i) = n$. The tasks assigned to levels $i + 1, \ldots, n$ are promoted and

Audsley Algorithm IV

remain schedulable. Obviously, the tasks assigned to levels $1, \ldots, i-1$ remain schedulable. ■

Theorem 28 suggests a procedure to assign task priorities:

1. First, determine a lowest-priority task
2. Apply recursively the same technique to the subset $\tau \setminus \{\tau_i\}$ of cardinality $n-1$

Audsley Algorithm V

Audsley's priority assignment:

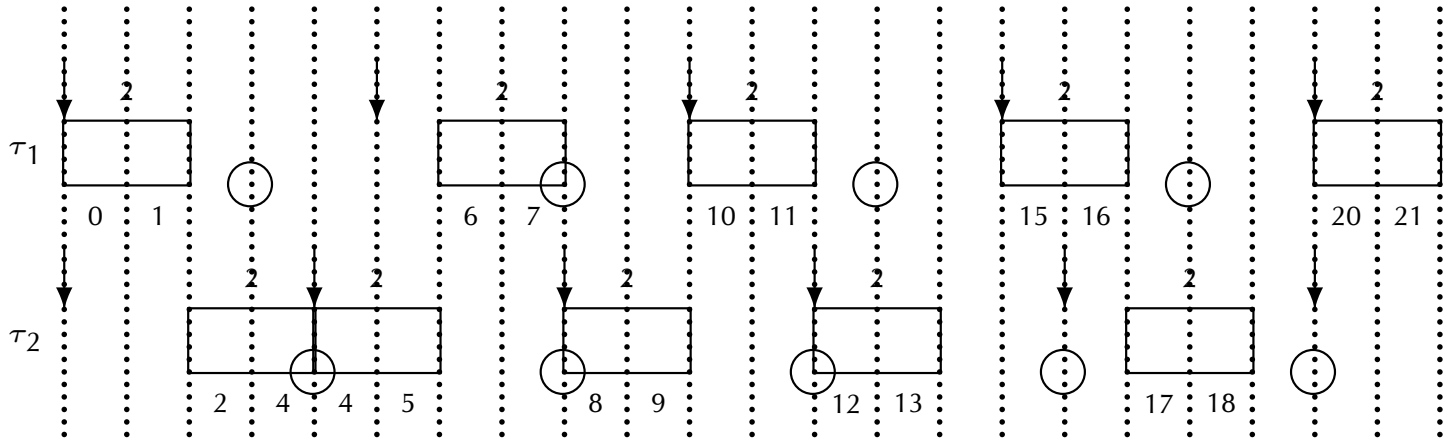
procedure **Audsley**(τ)
if (there is no lowest-priority viable task)
    return infeasible
else {
    let τ_i be a lowest-priority viable task in τ
    assign the lowest priority to τ_i
    Audsley($\tau \setminus \{\tau_i\}$)
}
end **Audsley**

The procedure requires checking if a task is lowest-priority viable. This can be checked by considering the interval $[0, S_n + P_n)$.
Notice that Audsley's algorithm will consider at most
$n + (n-1) + \cdots + 2 + 1 = \mathcal{O}(n^2)$ distinct priority assignments.

FJP-Priority assignment EDF

- EDF is a Fixed Job Priority scheduler
- The priority rule is the following: the lower the **absolute** deadline, the higher the priority. Ties can be resolved arbitrarily but in a consistent (deterministic) manner.

An EDF-Schedule



EDF Optimality I

In the following, EDF.J denotes the schedule produced by EDF when scheduling the job set J.

Definition 29 (Feasible Job Set)

A job set J is said to be feasible if there exists a schedule which meets all the job deadlines of J.

EDF Optimality II

Theorem 30

If a job set J is feasible, then EDF *feasibly schedules that set.*

Proof. The proof is made by induction. Let $\Delta > 0$ be an infinitesimal value. Consider a feasible schedule S for J, and let $[t_0, t_0 + \Delta)$ be the first time where S differs from EDF.J. Suppose that the job $j_1 = (a_1, e_1, d_1)$ is scheduled in S in that interval while $j_2 = (a_2, e_2, d_2)$ is scheduled in EDF.J. Since S meets all the deadlines, it feasibly schedules j_2 in $[a_2, d_2)$ (possibly preemptively) and j_2 completes before d_2. This implies that S schedules j_2 for a duration of Δ before d_2. By definition of EDF, $d_2 \leqslant d_1$; thus S schedules j_2 for a duration of Δ before d_1 as well. The schedule S^1 — obtained by swapping the executions of j_1 and j_2 for a duration of Δ in S — is identical to EDF.J in the interval $[0, t + \Delta)$. EDF optimality follows by induction on t: $S^\infty =$ EDF. ∎

In other words, EDF schedules everything that is schedulable!

Strong EDF Optimality I

Notice that the optimality of EDF is stronger than the one of RM (or DM). EDF optimality concerns sets of **jobs**, whereas optimality concerns sets of tasks for FTP-schedulers.
Consequently, EDF is also optimal for

- asynchronous systems
- arbitrary deadline systems

Implicit deadlines and EDF I

We have seen (Lemma 6) that $U(\tau) \leqslant 1$ is a **necessary** condition for feasibility. We will see that for EDF and periodic implicit deadline tasks, the condition is also **sufficient**.

Implicit deadlines and EDF II

Theorem 31

$U(\tau) \leqslant 1$ is a necessary and sufficient condition for the feasibility of periodic implicit deadline tasks.

Proof. Let τ be a task set of n implicit deadline periodic tasks $\tau_1, \ldots, \tau_n$. Consider a "processor sharing" schedule S obtained by partitioning the timeline into infinitesimal slots and by scheduling the task τ_i for a fraction of $U(\tau_i)$ in each slot. Since $U(\tau) \leqslant 1$, such a schedule can be constructed. That schedule contains, for each job τ_i, exactly $U(\tau_i) \times T_i = C_i$ CPU units between the job's arrival and its deadline. ■

Implicit deadlines and EDF III

Corollary 32

An implicit deadline periodic task set is EDF*-feasible iff* $U(\tau) \leqslant 1$.

Proof. This is a direct consequence of EDF optimality (Theorem 30) and Theorem 31.

Synchronous and constrained deadlines and EDF I

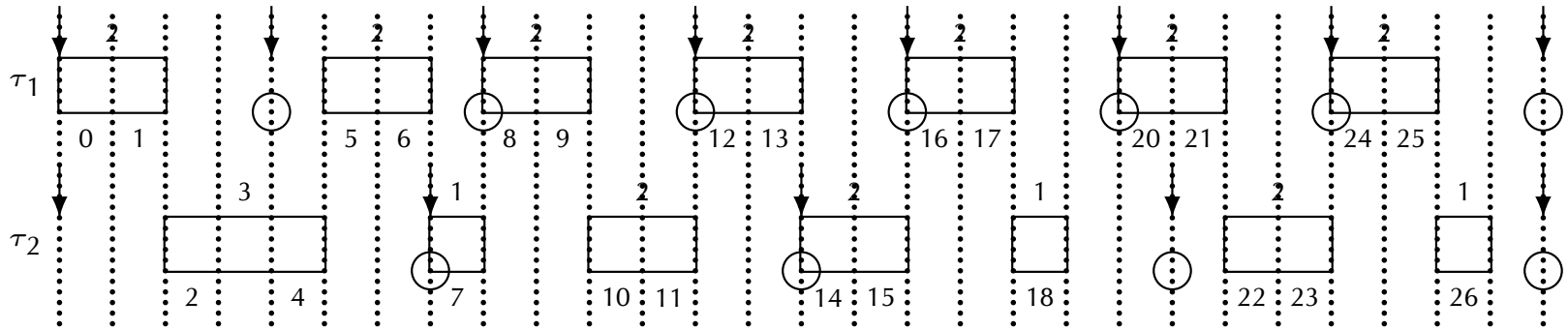
First, notice that the synchronous periodic case remains the worst-case. We have the following property as well:

Theorem 33

Let τ be a synchronous periodic arbitrary deadline system. If τ is EDF*-feasible, then all corresponding asynchronous periodic task sets and all corresponding sporadic task sets are* EDF*-feasible as well.*

Unfortunately, for EDF scheduling the response time of the first job in the synchronous configuration is not the largest one anymore.
Consider the system
$S = \{\tau_1 = \{C_1 = 2, D_1 = T_1 = 4\}, \{C_2 = 3, D_2 = T_2 = 7\}\}$, the response time of the first jobs of τ_2 in the synchronous case are: 5, 5, 5, **6**

Synchronous and constrained deadlines and EDF II



Liu & Layland have "shown" (the property is correct but the proof is flawed) that we cannot miss a deadline **after** an idle unit.

Theorem 34 ([Liu and Layland, 1973])

When EDF *schedules a synchronous periodic constrained deadline system, there are no idle units in the schedule before a deadline miss.*

Synchronous and constrained deadlines and EDF III

That property can be extended by considering the notion of idle instant (see Definition 19) and arbitrary deadlines.

Theorem 35 ([Goossens, 1999])

When EDF *schedules a synchronous periodic arbitrary deadline system, there are no idle instants in the schedule before a deadline miss.*

Consequently, the interval $[0, L)$ is a feasibility interval, where L corresponds to the first idle instant (origin excepted) in the schedule.

Synchronous and constrained deadlines and EDF IV

L is the smallest positive solution to

$$L = \sum_{i=1}^{n} \left\lceil \frac{L}{T_i} \right\rceil C_i$$

and can be computed by fixed-point iteration:

$$w_0 \stackrel{\text{def}}{=} \sum_{i=1}^{n} C_i,$$
$$w_{k+1} \stackrel{\text{def}}{=} \sum_{i=1}^{n} \left\lceil \frac{w_k}{T_i} \right\rceil C_i.$$

Notice this is exactly the same value as λ_n introduced for FTP-schedulers (see Equation (2)–slide 51)

Asynchronous Periodic Systems and EDF I

Definition 36 (Configuration)

Considering a periodic system R and a schedule S, we define the configuration at instant t as follows:

$$C_S(R,t) = ((\gamma_1(t), \alpha_1(t), \beta_1(t)), \ldots, (\gamma_n(t), \alpha_n(t), \beta_n(t)))$$

where

- $\gamma_i(t) = (t - O_i) \mod T_i$ is the time elapsed since the last request of τ_i, if $t \geqslant O_i$; $\gamma_i(t) = t - O_i$ otherwise.
- $\alpha_i(t)$ is the number of active jobs of τ_i.
- $\beta_i(t)$ is the cumulative CPU time used by the oldest active job of τ_i. If $\alpha_i(t) = 0$, we let $\beta_i(t) = 0$.

Asynchronous Periodic Systems and EDF II

Theorem 37

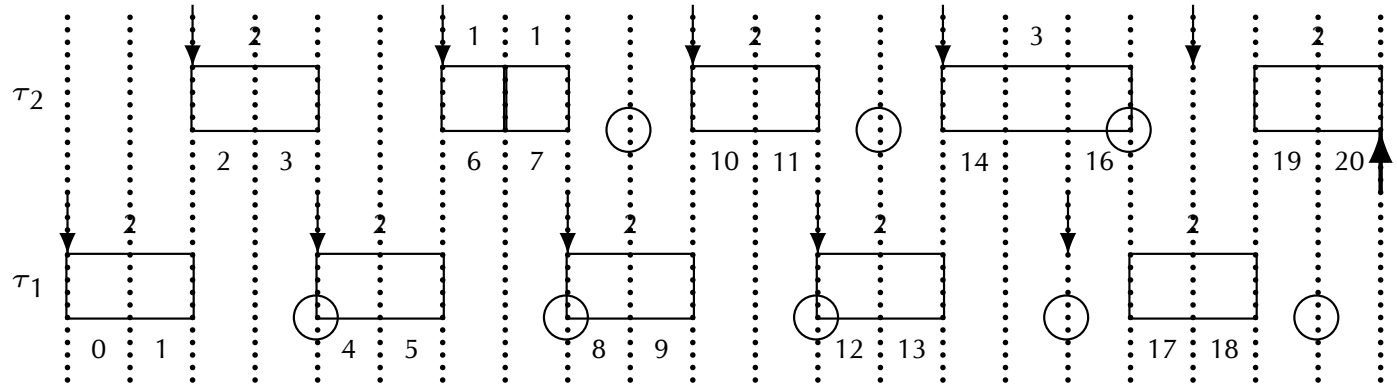
Let S be the EDF*-schedule for the periodic asynchronous arbitrary deadline system R. R is feasible iff (1) every deadline occurring in $[0, O_{\max} + 2P]$ is met and (2) $C_s(R, O_{max} + P) = C_s(R, O_{max} + 2P)$.*

Notice that both conditions set out by Theorem 37 are necessary, as shown by the following system.

	T_i	D_i	C_i	O_i
τ_1	4	4	2	0
τ_2	4	7	3	2

We have $P = 4$ and $O_{\max} = 2$; as we can see, all deadlines in the interval $[0, 10)$ are met but $C_S(R, 6) \neq C_S(R, 10)$, since $\beta_2(6) = 2$ and $\beta_2(10) = 1$. Actually, τ_2 misses its deadline at time 21.

Asynchronous Periodic Systems and EDF III



Formally (see Definition 17), $[0, O_{\max} + 2P)$ is **not** a feasibility interval!
Notice that the condition $U(\tau) \leqslant 1$ is not satisfied by our system ($U = \frac{5}{4}$).
In fact, if $U(\tau) \leqslant 1$ holds, only the first condition of Theorem 37 must be satisfied to conclude feasibility.

Corollary 38

$[0, O_{\max} + 2P)$ *is a feasibility interval for periodic asynchronous arbitrary deadline systems where* $U(\tau) \leqslant 1$.

DP-Priority assignment LLF I

The Least Laxity First (LLF) is a job-level **dynamic** priority scheduler (DP, see slide 19). LLF is also called the slack-time algorithm.
At each instant, the priority of each job depends on its **laxity**.

Definition 39 (Laxity)

Let J be a job characterized by the tuple (a, e, d) (see Definition 1, slide 11). Let $\varepsilon_J(t)$ represent the cumulative CPU time used by J since its release. The **laxity** (denoted by $\ell_J(t)$) at instant t is defined as follows:

$$\ell_J(t) \stackrel{\text{def}}{=} d - t - (e - \varepsilon_J(t))$$

DP-Priority assignment LLF II

The job laxity represents its degree of urgency:

- $\ell_J(t) > 0$: the job can be feasibly delayed (with a maximum of $\ell_J(t)$ time-units) before being scheduled
- $\ell_J(t) = 0$: it is mandatory to schedule the job immediately and until its deadline to ensure feasibility
- $\ell_J(t) < 0$: the job will miss (or misses) its deadline for sure
- the laxity of a running job remains constant

Definition 40 (LLF)

At each instant, LLF gives the CPU to the job with the smallest laxity. Ties can be resolved arbitrarily but in a consistent (deterministic) manner.

Mok has shown, in the context of his PhD thesis, the optimality of LLF:

Theorem 41 ([Mok, 1983])

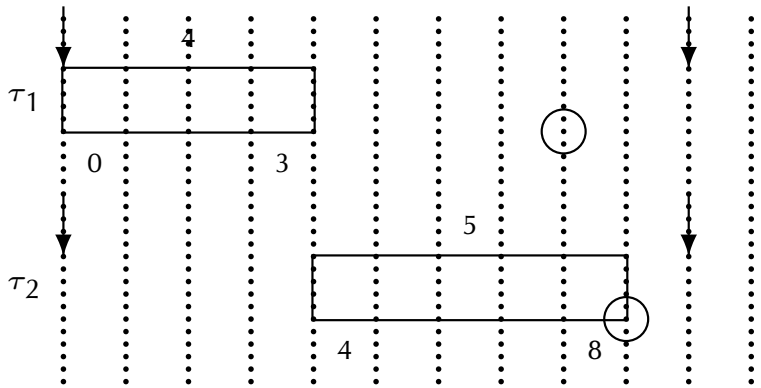
If a job set J is feasible, then LLF *feasibly schedules that set.*

EDF vs. LLF— number of preemptions I

LLF and EDF are both optimal schedulers: they schedule any feasible set of jobs. However, they differ with regards to the number of preemptions that occur during the schedule.
Consider the following system:

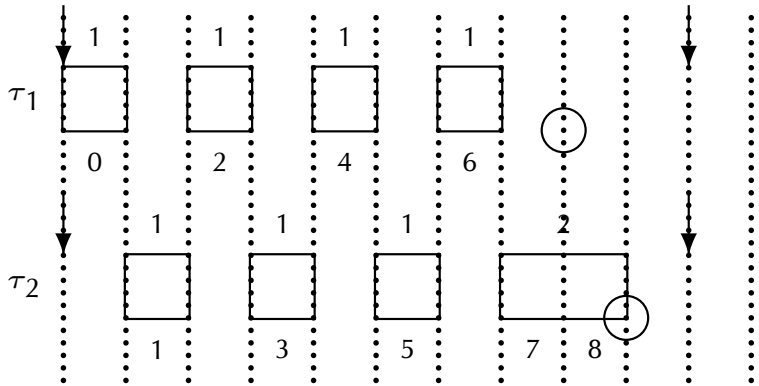
	C_i	T_i	D_i
τ_1	4	10	8
τ_2	5	10	9

The EDF-schedule is preemption-free:



EDF vs. LLF— number of preemptions II

The LLF-schedule, however, has 6 preemptions every 10 time-units.



In fact, a **thrashing** situation (situation where large amounts of computer resources are used to do a minimal amount of work) can occur: Suppose that at time t, two active jobs (J_1 and J_2) have the same and minimal laxity $\ell_{J_1}(t) = \ell_{J_2}(t)$. Assuming that LLF schedules J_1 first, then at instant $t+1$ we have: $\ell_{J_1}(t+1) = \ell_{J_2}(t+1) + 1$. LLF must thus preempt J_1 and schedule J_2. At instant $t+2$, both laxities are identical once more: $\ell_{J_1}(t+2) = \ell_{J_2}(t+2)$. This scenario repeats while J_1 and J_2 are active and have the minimal laxity of the system.

EDF vs. LLF — scheduler classes

To conclude about LLF, for the uniprocessor scheduling problem of periodic real-time tasks, there are not benefits to consider the DP scheduler class since they induce a lot of preemptions with no additional properties (optimality exists for a “simpler” class — FJP).

References I

[Audsley et al., 1993] Audsley, N., Tindell, K., and Burns, A. (1993).
The end of the line for static cyclic scheduling?
In *Proceedings of the EuroMicro Conference on Real-Time Systems*, pages 36–41.

[Audsley, 1991] Audsley, N. C. (1991).
Optimal priority assignment and feasibility of static priority tasks with arbitrary start times.
Technical report.

[Goossens, 1999] Goossens, J. (1999).
Scheduling of hard real-time periodic systems with various kinds of deadline and offsets contraints.
PhD thesis, Université Libre de Bruxelles.

References II

[Leung, 2004] Leung, J., editor (2004).
Handbook of Scheduling: Algorithms, Models, and Performance Analysis.
Chapman Hall/CRC Press.
Chapter: Scheduling Real-time Tasks: Algorithms and Complexity.

[Leung and Whitehead, 1982] Leung, J. Y.-T. and Whitehead, J. (1982).
On the complexity of fixed-priority scheduling of periodic, real-time tasks.
Performance Evaluation, 2:237–250.

[Liu and Layland, 1973] Liu, C. L. and Layland, J. W. (1973).
Scheduling algorithms for multiprogramming in a hard-real-time environment.
Journal of the Association for Computing Machinery, 20(1):46–61.

References III

[Liu, 2000] Liu, J. W. S. (2000).
Real-Time Systems.
Prentice-Hall.

[Mok, 1983] Mok, A. K.-L. (1983).
Fundamental design problems of distributed systems for the hard-real-time environment.
PhD thesis, Massachusetts Institute of Technology.