La recherche en Ordonnancement Temps-réel

Introduction

Damien Masson[1]

[1]Université Paris-Est, LIGM UMR CNRS 8049, UPEM, ESIEE Paris, ENPC, France

Mars 2016

Introduction à l'ordonnancement temps-réel

Introduction à l'ordonnancement temps-réel

Définition

"En informatique temps réel, le comportement correct d'un système dépend, non seulement des résultats logiques des traitements, mais aussi du temps auquel les résultats sont produits".

John Stankovic. *Misconceptions about real-time computing*. IEEE Computer, October 1988.

Déterminisme

Garantir qu'un Système Temps Réel (STR) respectera ses spécifications pendant toute sa durée de vie.

- Déterminisme logique : les mêmes entrées appliquées au système produisent les mêmes résultats.
- Déterminisme temporel : respect des contraintes temporelles (ex : échéance).
- Fiabilité : le système répond à des contraintes de disponibilité (fiabilité du logiciel et du matériel).
- Système prédictible : on cherche à déterminer a priori si le système va répondre aux exigences temporelles.

Temps-réel n'est pas vitesse

Un système temps réel n'est pas un système rapide mais un système qui satisfait à des contraintes temporelles.

Exemples de grandeur :

- la milliseconde pour les systèmes radar
- la seconde pour les systèmes de visualisation avec interraction humaine
- quelques heures pour le contrôle de production impliquant des réactions chimiques
- 24 heures pour les prévisions météo
- plusieurs mois ou années pour les systèmes de navigation de sonde spatiale

Systèmes embarqués (SE)

Systèmes informatiques dans lequel le processeur/calculateur est englobé dans un système plus large.

- le logiciel est souvent entièrement dédié à une application spécifique
- exemple : une sonde spatiale, un pacemaker, ...
- intervention humaine directe difficile voire impossible

Avionique

Système temps réel critique

- Plusieurs systèmes embarqués : commandes de vol, radars, moteurs, ..., systèmes multimédia passagers !
- Contraintes temporelles fortes (les conséquences en cas de non respect = mort)
- Dimensionnement au pire cas et réservation des ressources
- Utilisation de redondance matérielle et logicielle
- Matériel et logiciel dédiés
- Système fermé, validé a priori

Multimédia sur Internet

Système temps réel souple

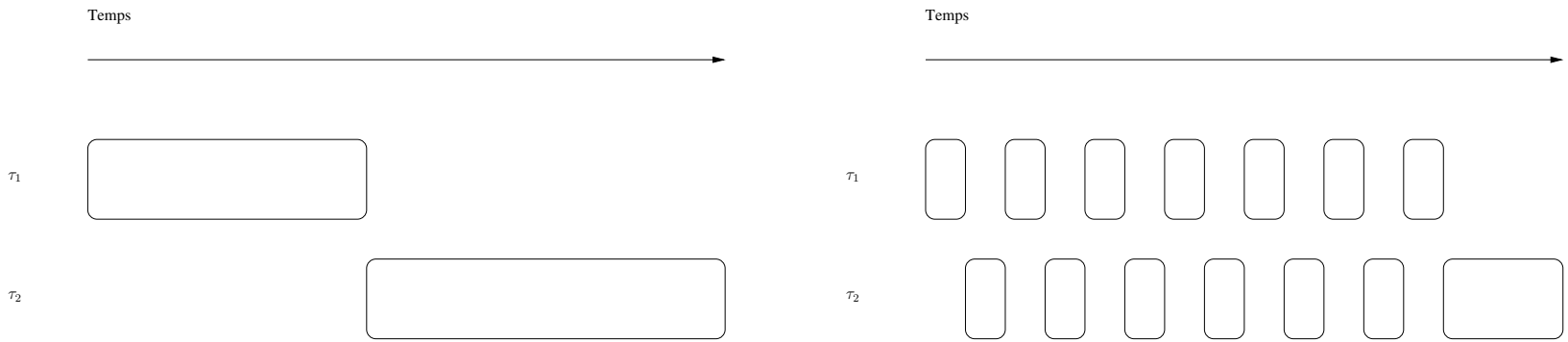
- Contraintes temps-réel moins fortes (conséquences = QoS)
- Contraintes temporelles : gigue, délais de bout en bout, temps de réponse
- Application interactive
- Débits variables et difficiles à estimer hors ligne.

Autres exemples

- Transports : ferroviaire, aérospatiale, système d'information géographique (SIG), systèmes de régulation automobile, ...
- Médias (décodeurs numériques)
- Communications (téléphone mobile, routeurs, satellites, ...)
- Supervision médicale, écologique
- Système de production industriel : centrale nucléaire, chaîne de montage, usine chimique
- Robotique
- ...

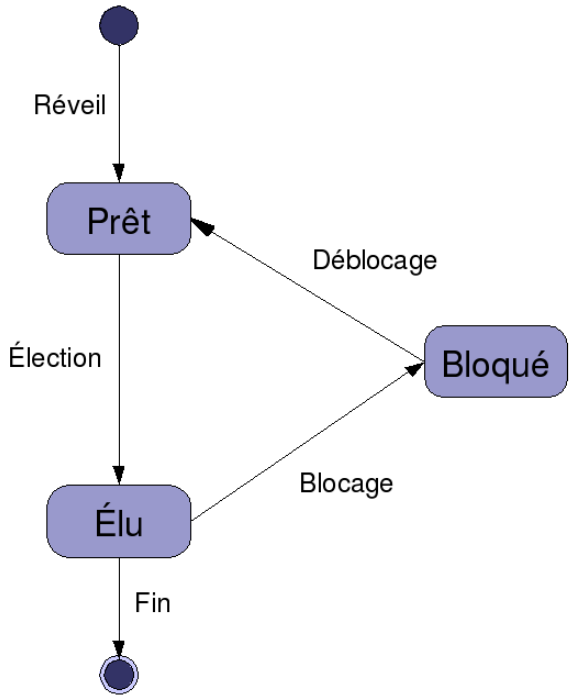
Introduction à l'ordonnancement temps-réel

Système temps partagé



- Facilite l'accès aux ressources
- Masque les ressources (multiprocesseur, disques RAID, mémoire)
- Recherche de l'équité
- Absence de famine
- Maximise le débit global

Ordonnancement Classique (ex : Linux)



- Priorité = proportionnelle au temps d'attente dans l'état "prêt"
- Équité, pas de prise en compte de l'urgence ou de contrainte temporelle
- Politique généralement opaque
- Temps de réponse inconnu

Modèle de tâches récurrentes (sporadiques ou périodiques)

C.L. Liu and J.W. Layland, *Scheduling algorithms for multiprogramming in a hard real-time environment*, Journal of the Association for Computing Machinery 20 (1973), no. 1, p. 46-61.

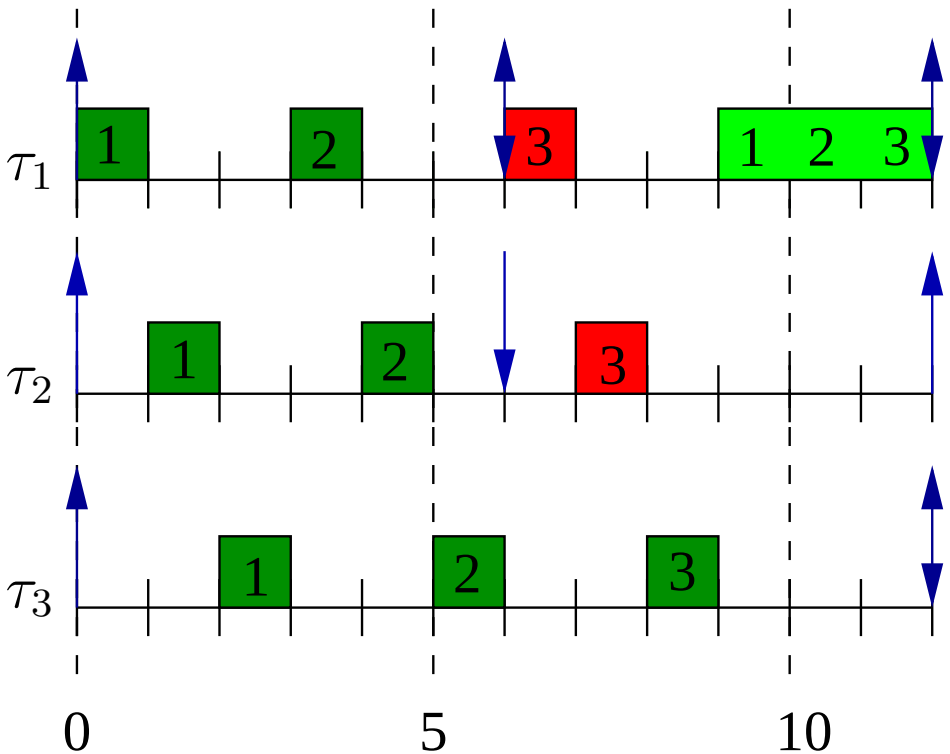
Une tâche τ_i est définie par :

- date de première activation (release) : r_i
- pire temps d'exécution (WCET) : C_i
- période : T_i
- échéance relative : D_i
- échéance absolue de l'instance k : $d_{i,k}$
- convention : majuscules pour les durées, minuscules pour les dates
- ... (Modèle extensible)

Exemple

	Coût C_i	Période T_i	Échéance D_i
τ_1	3	6	6
τ_2	3	12	6
τ_3	3	12	12

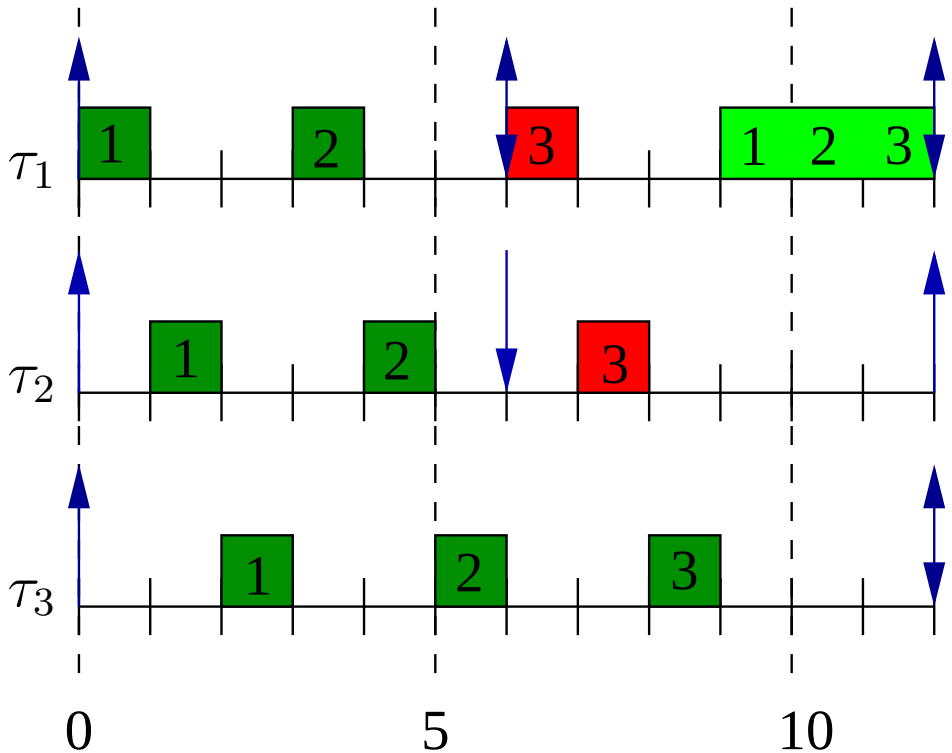
Round Robin



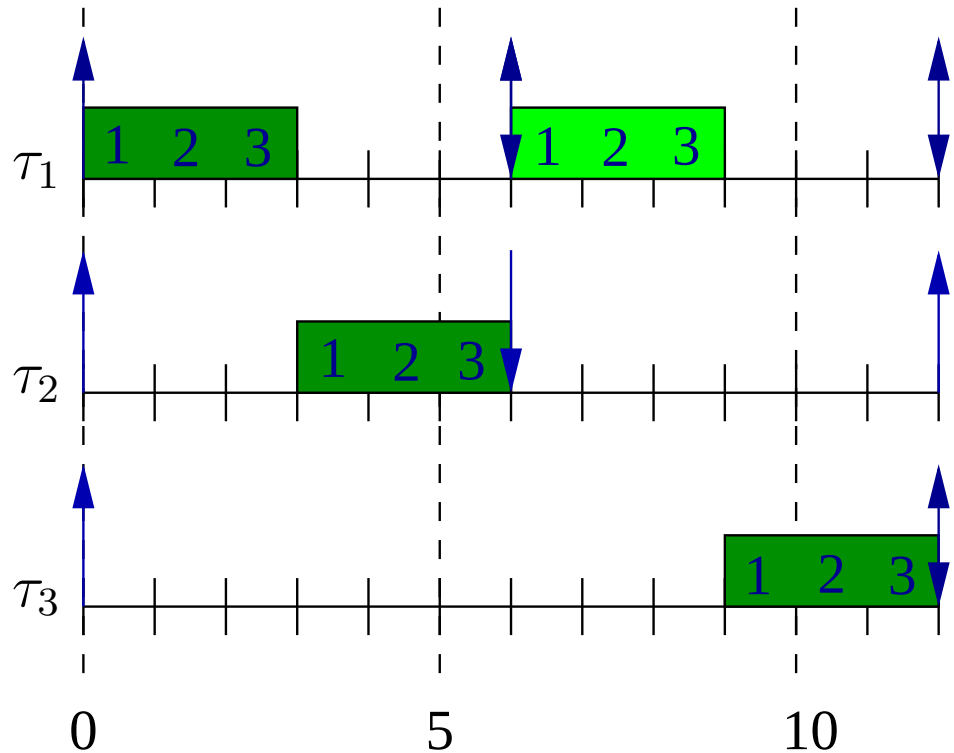
Exemple

	Coût C_i	Période T_i	Échéance D_i
τ_1	3	6	6
τ_2	3	12	6
τ_3	3	12	12

Round Robin



Priorités fixes



Ordonnancement Temps-réel

- algorithme d'ordonnancement : c'est l'algorithme utilisé pour décider quelle tâche doit s'exécuter
- ordonnancement : c'est le résultat de l'algorithme d'ordonnancement
- ordonnanceur : c'est la tâche chargée d'appliquer l'algorithme d'ordonnancement

- Deux familles : préemptif, non préemptif
- Deux méthodes : hors ligne ou en ligne

Ordonnancement Temps-réel

- algorithme d'ordonnancement : c'est l'algorithme utilisé pour décider quelle tâche doit s'exécuter
- ordonnancement : c'est le résultat de l'algorithme d'ordonnancement
- ordonnanceur : c'est la tâche chargée d'appliquer l'algorithme d'ordonnancement

- Deux familles : préemptif, non préemptif
- Deux méthodes : hors ligne ou en ligne

On s'intéresse aujourd'hui aux algorithmes préemptifs en ligne.

Ordonnancement temps-réel en ligne préemptif

(Mono Processeur)

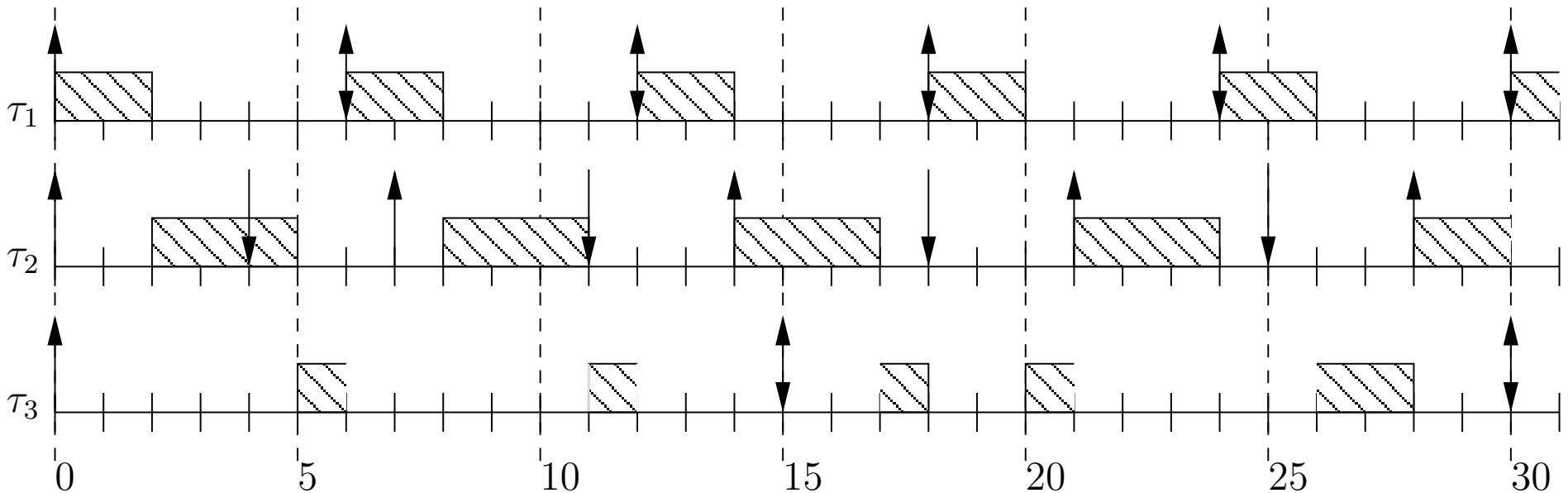
Les instances des tâches (job) sont classées par priorité. À chaque instant, l'ordonnanceur attribue le processeur à la tâche la plus prioritaire.

- priorités fixes : les priorités des tâches sont fonction d'une constante (période, échéance, importance...) ;
- priorités dynamiques : les priorités sont fonction d'une variable (prochaine échéance, laxité, ...).

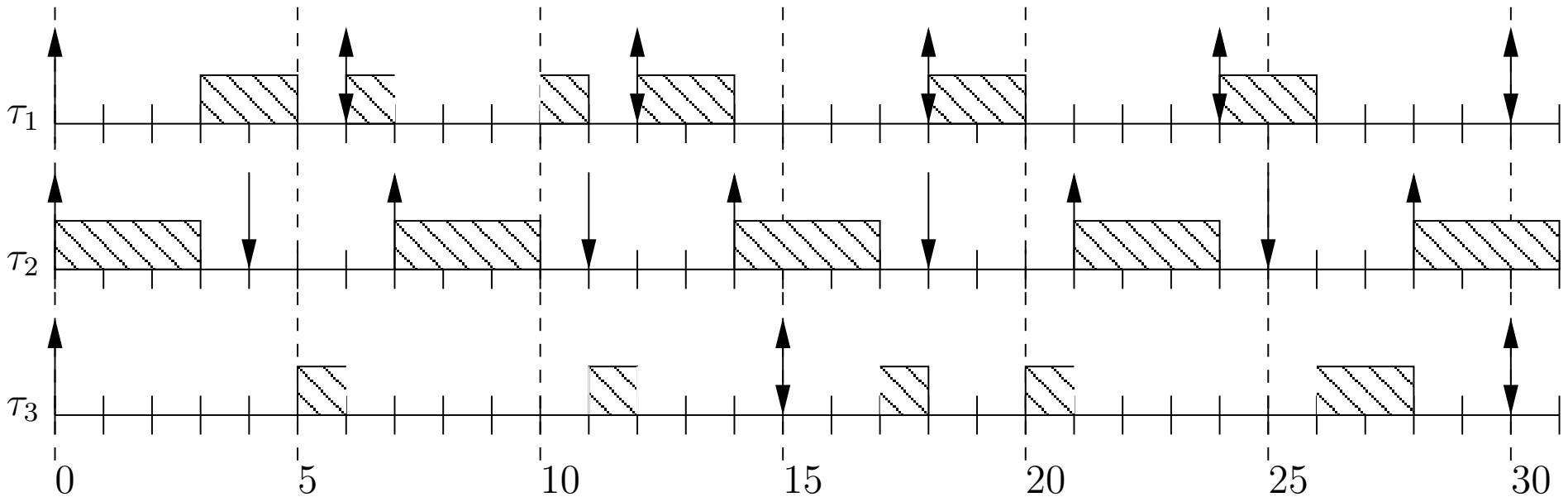
Les principaux algorithmes

- Rate Monotonic (RM) : priorité à la plus petite période
- Deadline Monotonic (DM) : priorité à la plus petite échéance relative
- EDF : priorité à l'échéance absolue la plus proche

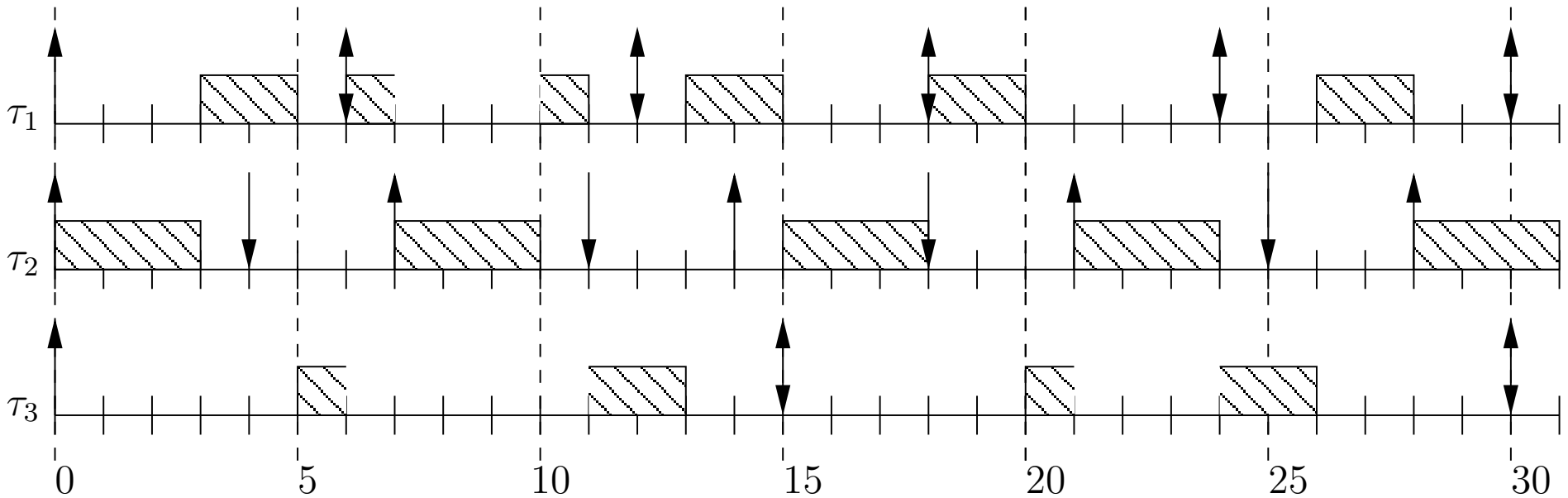
RM



DM



EDF



Introduction à l'ordonnancement temps-réel

Optimalité

- RM est optimal pour les algorithmes à priorité fixes pour les tâches à échéances sur requêtes
- DM est optimal pour les algorithmes à priorité fixes pour les tâches à échéances contraintes
- OPA est une méthode optimale pour assigner les priorités
- EDF est optimal **par rapport à l'ordonnançabilité**

Ordonnançabilité

≠ Faisabilité

- **Faisabilité :** Soit un jeu de tâches, existe-t-il un ordonnançement qui respecte toutes les contraintes ?
- **Ordonnançabilité :** Soit un jeu de tâches et un algorithme d'ordonnancement, l'ordonnançement produit respecte-t-il toutes les contraintes ?

Plusieurs approches selon la criticité du système étudié et les moyens disponibles :

- condition suffisante mais pas nécessaire pour un contrôle d'admission en ligne,
- détection de fautes ou de surcharge,
- vérification exacte à l'aide de la théorie de l'analyse de faisabilité/ordonnançabilité.

Étude de la charge

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

l'étude de la charge peut permettre de conclure sur la faisabilité et/ou l'ordonnançabilité dans certains cas :

- $U \leq n(2^{\frac{1}{n}} - 1)$ est une condition suffisante d'ordonnançabilité pour un système à échéances sur requêtes ordonnancé en priorités fixes. Attention, ce n'est pas une condition nécessaire.
- $U \leq 1$ est une condition nécessaire et suffisante pour un système à échéances sur requêtes ordonnancé en EDF.

Malheureusement, lorsque $D_i \leq T_i$, ou pire lorsqu'on autorise $D_i > T_i$ (tâches dites ré-entrantes), les choses se compliquent !

Étude de l'ordonnançabilité d'un système à échéances sur requêtes

(priorités fixes)

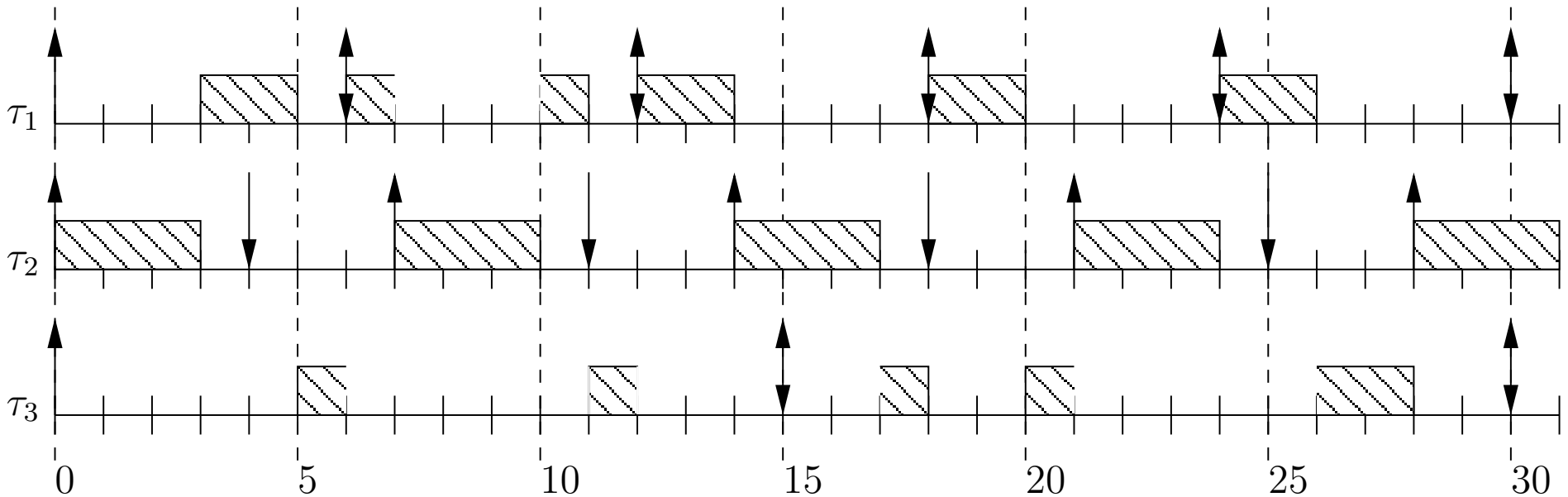
- on l'a vu, $U \leq n(2^{\frac{1}{n}} - 1)$ est une condition suffisante d'ordonnançabilité, et $U > 1$ est une condition suffisante (évidente ?) de non ordonnançabilité
- cas $1 > U > n(2^{\frac{1}{n}} - 1)$?
  - étude de la demande : on cherche un instant inférieur à l'échéance où la demande cumulée est plus faible que le temps écoulé
  - étude du pire temps de réponse : le pire temps de réponse est-il plus petit que l'échéance ?
    - temps de réponse d'une instance R_i^j : temps entre la requête et la fin du traitement
    - pire temps de réponse d'une tâche ($WCRT_i$) : maximum des temps de réponse de chaque instance

Étude de la demande

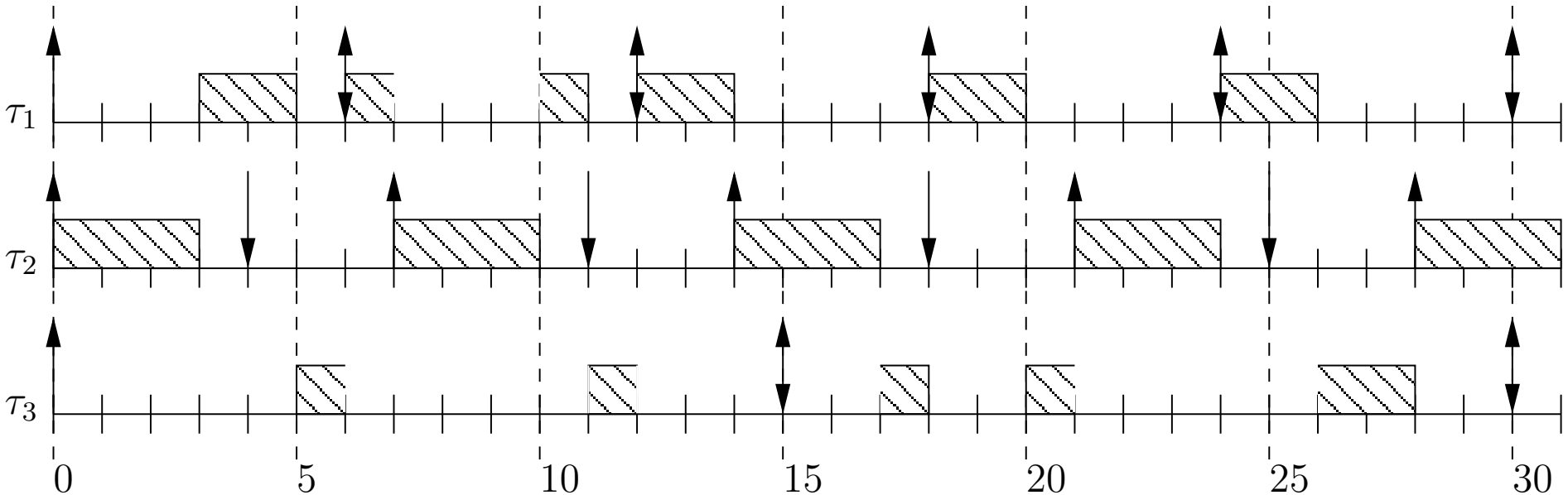
Un système où les tâches réentrantes sont interdites ($\forall i, D_i \leq T_i$) est ordonnançable en priorités fixes ssi il existe un instant t dans l'intervalle $]0, Di]$ tel que $t = w_i(t)$

- avec $w_i(t) = \sum_{k \leq i} \left\lceil \frac{t}{T_k} \right\rceil C_k$
- algorithme récursif : calcul de $t_1 = w_i(0)$, puis $t_2 = w_i(t_1)$, ..., $t_n = w_i(t_{n-1})$
- on s'arrête si l'on dépasse l'échéance ou si l'on trouve t tel que $t = w_i(t)$.

DM



DM



t	8	13	15
$w_3(t)$	13	15	18

Limites

- Ce test ne permet que de conclure sur l'ordonnançabilité, il ne donne pas d'autres informations.
- Il peut être intéressant de calculer les temps de réponses, pour avoir une idée du comportement de la tâche (gigue, temps de réponse moyen...)
- Ce test ne marche plus pour le cas $D_i > T_i$ (tâches ré-entrantes)

Calcul de temps de réponse

- calcul récursif très similaire à l'étude de la demande
- une tache n'est retardée que par les plus prioritaires
- on cherche à calculer le tdr de l'instance j de la tâche τ_i, en numérotant les instances à partir de 1. Sa date de terminaison, notée F_i^j, est donnée par l'équation :

$$F_i^j = \min_{t>0}\{t = w_{i-1}(t) + j * C_i\} \quad (1)$$

- Son tdr, R_i^j, est alors la différence entre sa date de terminaison et sa date d'activation. Il est donné par l'équation :

$$R_i^j = F_i^j - (r_i + (j-1)T_i) \quad (2)$$

Busy Period

période occupée

- une période occupée de niveau i est un intervalle entre deux instants d'inactivité au niveau de priorité i du processeur ($w_i(t) = t$)
- pour calculer celle qui commence à $t = 0$ (la pire dans le scénario synchrone) on réalise l'étude de la demande, mais on ne s'arrête plus si l'échéance est dépassée

Il suffit d'étudier une tâche durant sa busy period synchrone pour rencontrer son pire temps de réponse.

Faisabilité sous EDF

- $D_i = T_i$: ordonnancable ssi $\forall t \geq 0, t \geq \sum_{i=1}^{n} \left\lfloor \frac{t}{T_i} \right\rfloor C_i$
- équivalent au test necessaire et suffisant $\sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1$
- $D_i \leq T_i$: ordonnancable ssi $\forall t \geq 0, t \geq \sum_{i=1}^{n} \left(\left\lfloor \frac{t - D_i}{T_i} \right\rfloor + 1 \right)_0 C_i$
- on se limite à vérifier la propriété lors des échéances contenues dans la busy-period synchrone

Autres modèles de tâches

- Jusqu'à présent, on a fait l'hypothèse de tâches indépendantes les unes des autres, mais il peut exister d'autres contraintes :
  - relations de précédence entre les tâches
  - partage de ressources avec protection par sémaphores entre les tâches
- Des tâches non périodiques doivent également pouvoir être traitée :
  - temps d'inter arrivée de deux instances minimal et étude pire cas (modèle sporadique)
  - traitements encapsulés dans un serveur avec des ressources réservées
  - traitement sur le temps libre du système (background ou slack stealing)