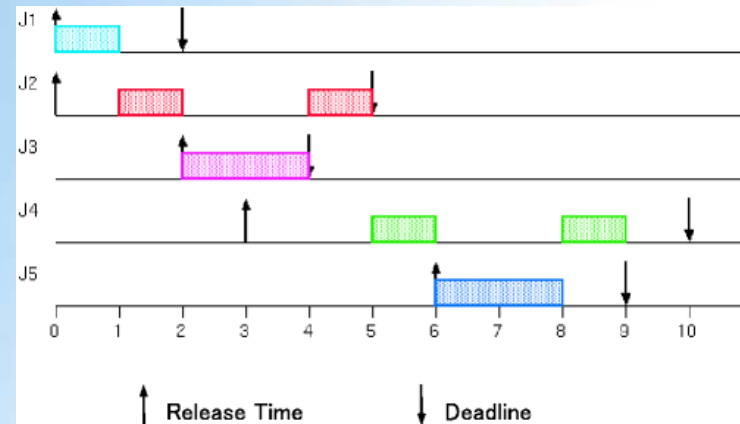


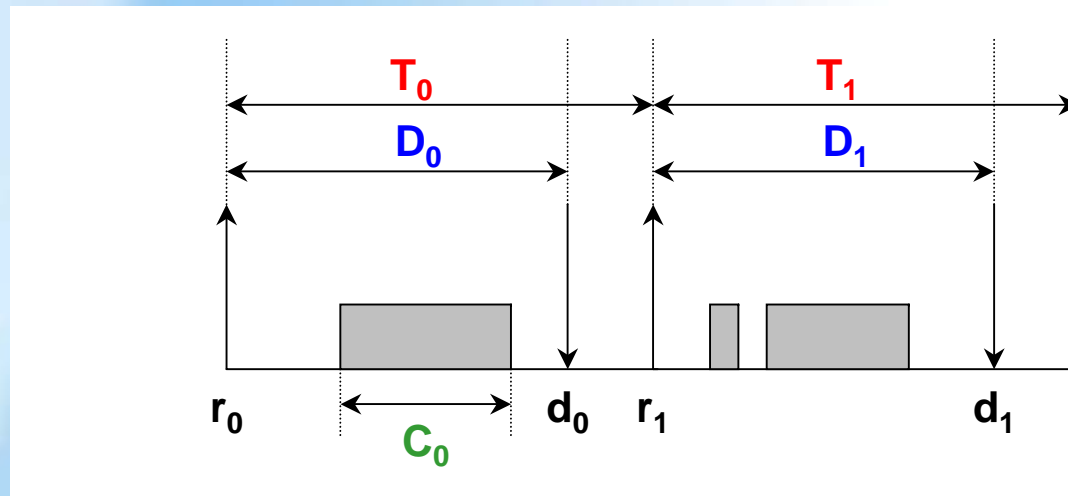
Problématique de l'ordonnancement temps-réel



Plan du cours

- ④ **La notion de tâche temps-réel**
- ④ **Le mécanisme d'ordonnancement**
- ④ **L'ordonnancement de tâches périodiques**
- ④ **L'ordonnancement de tâches apériodiques**

Le modèle canonique des tâches temps-réel $\tau_i(C_i, T_i, D_i)$



- r_i : date de réveil
- T_i : période d'exécution
- C_i : durée d'exécution maximale
- D_i : délai critique
- $d_i = r_i + D_i$: date d'échéance
- $L = D_i - C_i$: laxité

Les niveaux de contraintes temporelles

- Temps-réel **strict/dur** (*hard real-time*) (*ex : Airbag*)
 - toutes les instances de tâches doivent être exécutées dans le respect de leurs échéances

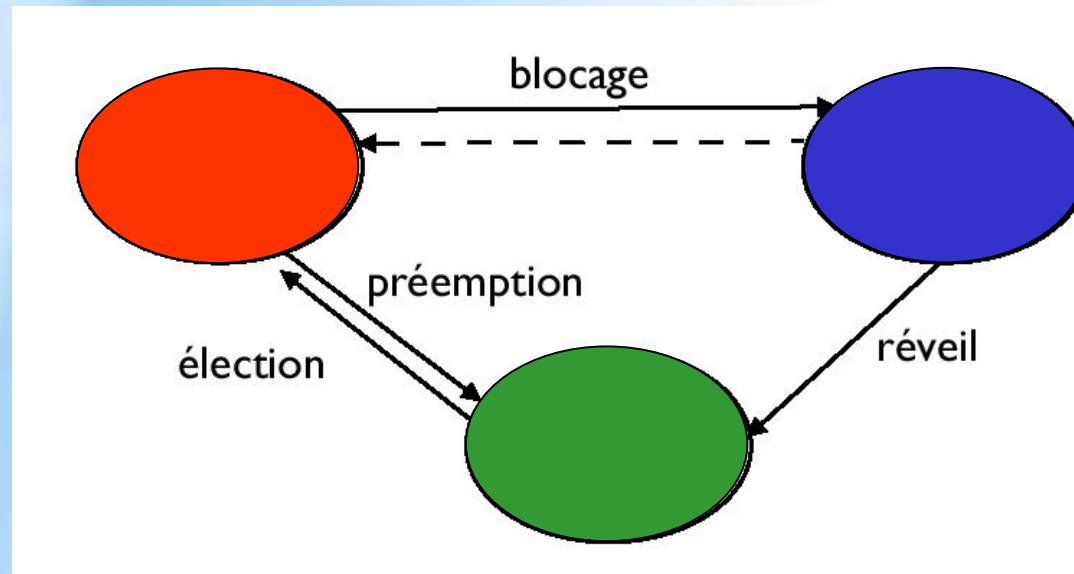
- Temps-réel **souple/mou** (*soft real-time*) (*ex : Distributeur automatique*)
 - les tâches peuvent manquer des échéances

- Temps-réel **ferme** (*firm real-time*) (*ex : Systèmes multimédias*)
 - les tâches sont autorisées à ne pas respecter *occasionnellement* leurs échéances

Les types de tâches temps-réel

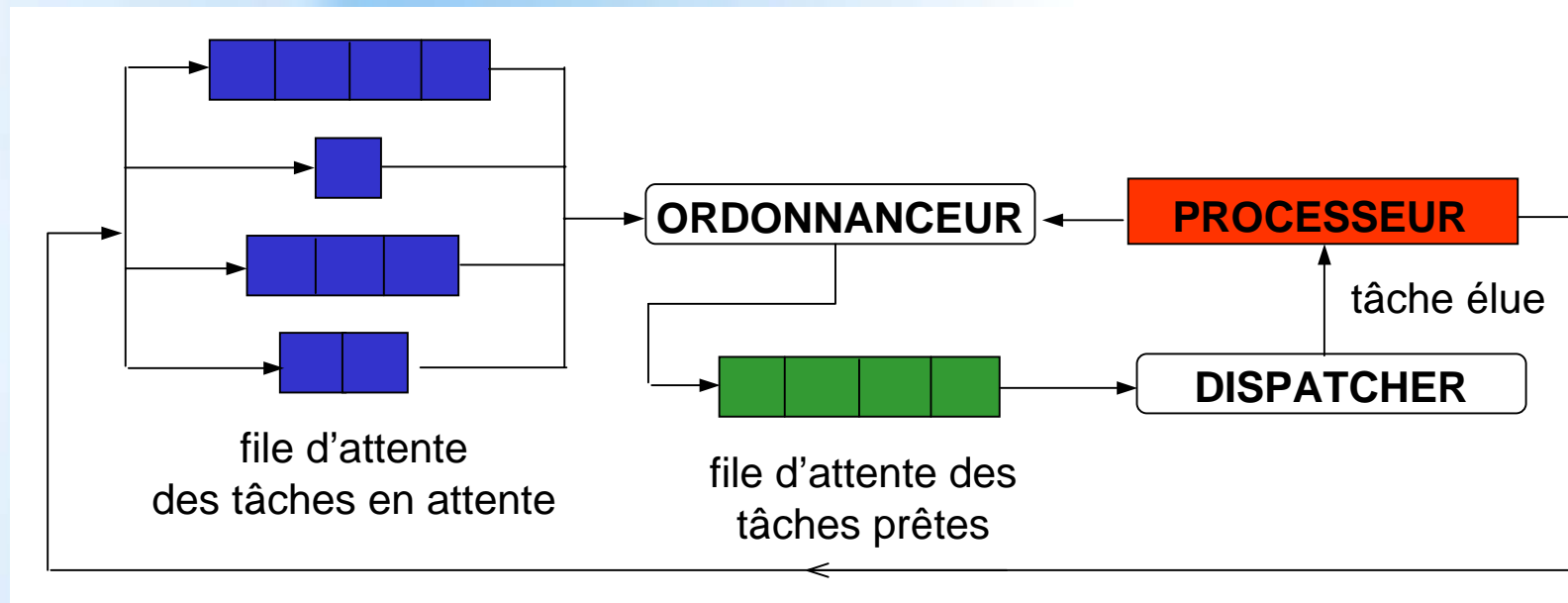
- Les tâches **périodiques** (*ex : Relevé de température*)
 - activation à intervalles réguliers,
 - à échéances sur requêtes si $D_i = T_i$
- Les tâches **apériodiques/sporadiques**
 - activation à des instants irréguliers,
 - intervalle de temps entre deux activations borné (sporadiques) ou non (apériodiques),
 - à contraintes relatives, (*ex : Requête sur une borne de service*)
 - à contraintes strictes (*ex : Arrêt d'urgence*)

Diagramme d'états d'une tâche temps-réel



- **courant** : la tâche s'exécute sur le processeur
- **prêt** : la tâche est prête à s'exécuter mais n'a pas le processeur
- **en-attente** : il manque une ressource (en plus du processeur) à la tâche pour qu'elle puisse s'exécuter

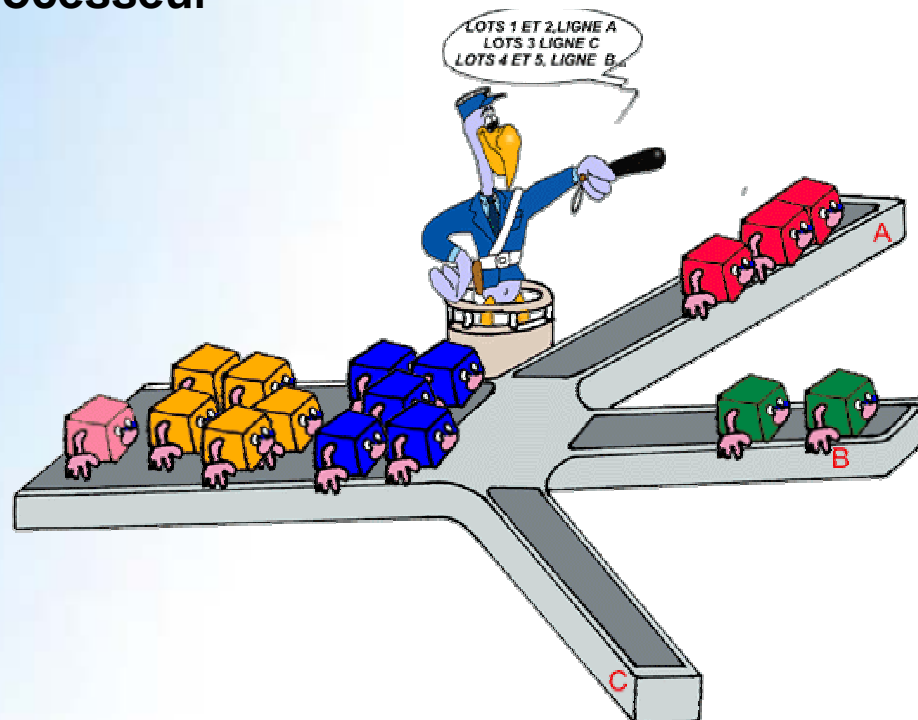
L'ordonnancement des tâches temps-réel



- **ORDONNANCEUR** : alloue le processeur aux différentes tâches
- **DISPATCHER** : implémente l'ordonnanceur (élection des tâches prêtes)

Typologie des algorithmes d'ordonnancement

- **Monoprocasseur / multiprocasseur**
- **En-ligne / Hors-ligne**
- **Préemptif / Non préemptif**
- **Oisif / Non oisif**
- **Centralisé / Réparti**



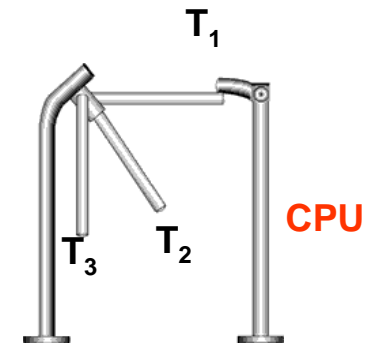
L'ordonnancement non préemptif

- Ordonnancement **selon l'ordre d'arrivée** :
 - premier arrivé, premier servi (First Come First Serve - FCFS)

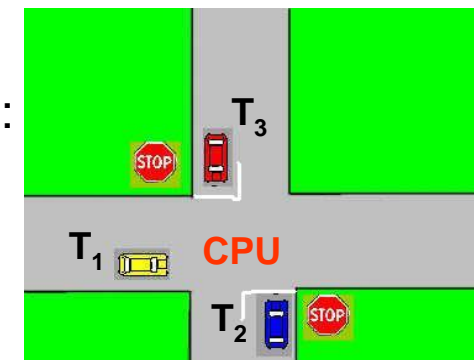
- Ordonnancement **selon la durée de calcul** :
 - le plus court d'abord (Shortest Job First – SJF)

L'ordonnancement préemptif

- Ordonnancement **sans notion de priorité** :
 - temps-partagé avec politique du tourniquet (round-robin)

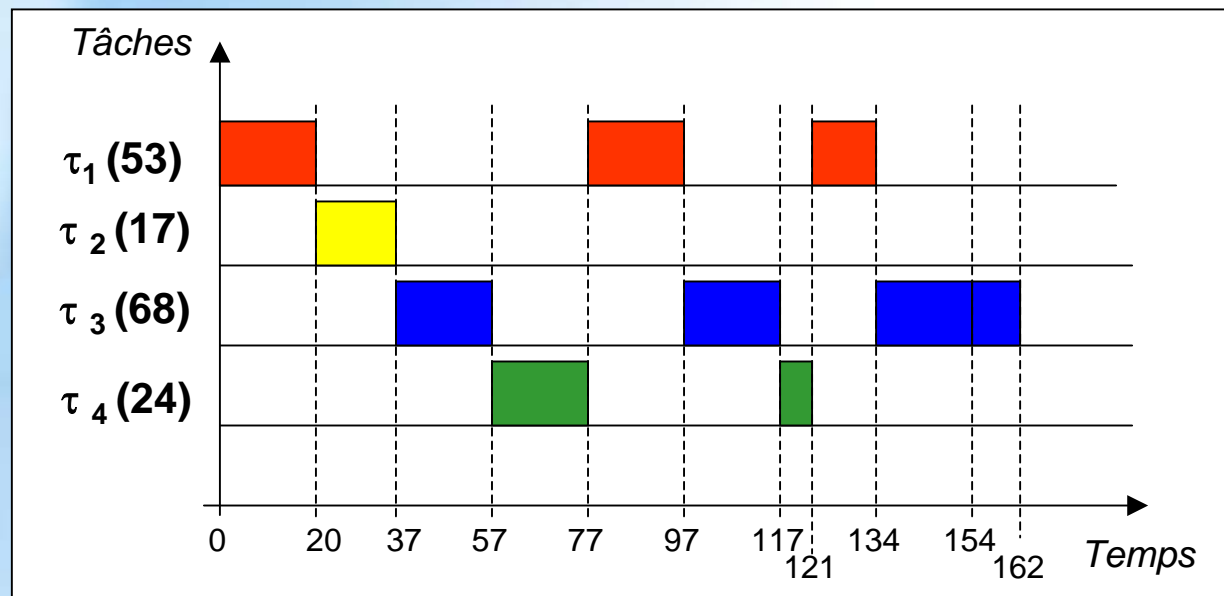


- Ordonnancement **à priorités** (statiques ou dynamiques) :
 - la tâche la plus prioritaire obtient le processeur



L'ordonnancement temps-partagé (Round-Robin)

- **Principe** : allocation du processeur par tranche (quantum) de temps
- **Exemple** ($q=20$, $n=4$) :



- Chaque tâche obtient le processeur au bout de $(n-1)*q$ unités de temps au plus

L'ordonnancement temps-partagé (Round-Robin)

- Intérêts :

- Equité de l'attribution du processeur entre toutes les tâches
- Mise en œuvre simple

- Inconvénients :

- Pas de prise en compte de l'importance relative des tâches
- Difficulté du choix de la tranche de temps
 - Si q est trop grand, round-robin devient équivalent à FIFO
 - Si q est trop petit, il y a augmentation du nombre de changements de contexte !

L'ordonnancement préemptif à priorités

- Ordonnancement à priorités fixes (statiques)

- Rate Monotonic (RM)
- Deadline Monotonic (DM)

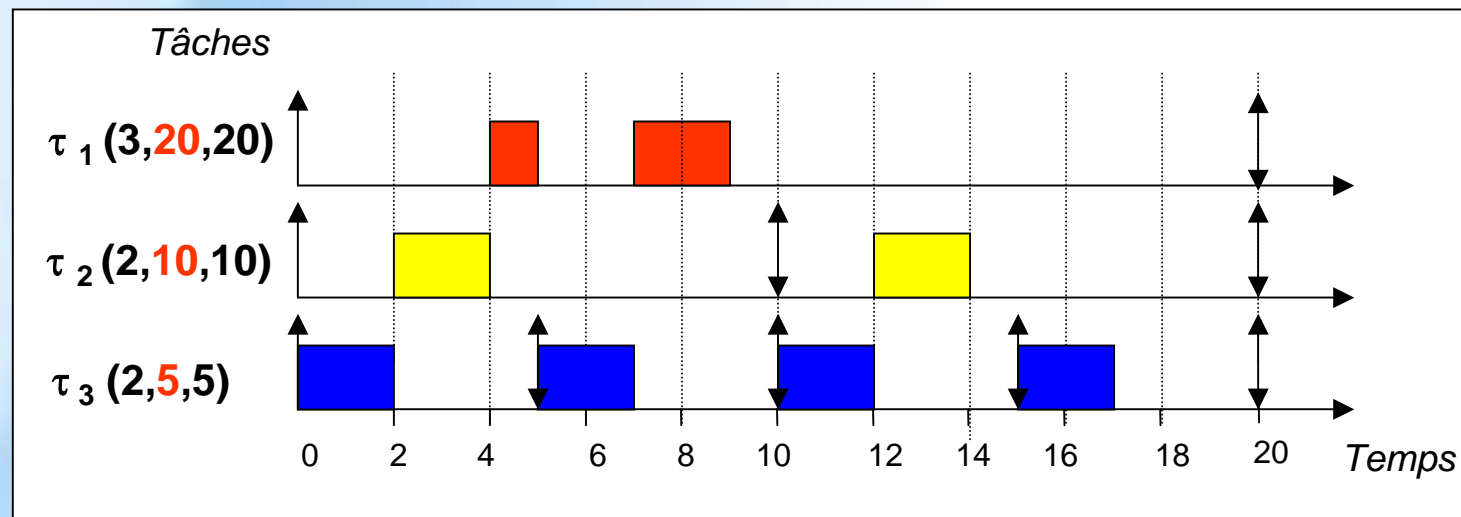
- Ordonnancement à priorités dynamiques

- Earliest Deadline First (EDF)
- Least Laxity First (LLF)

Rate Monotonic (RM)

- **Principe** : une tâche est d'autant plus prioritaire que sa période d'activation T_i est petite

- **Exemple** :



- **Propriété** : RM est **optimal** dans la classe des algorithmes à priorités fixes pour des tâches périodiques indépendantes à échéances sur requêtes ($D_i=T_i$)

Rate Monotonic (RM)

- Conditions de faisabilité :

- Condition nécessaire :
$$U = \sum_{i=1}^n \frac{C_i}{T_i} \leq 1$$

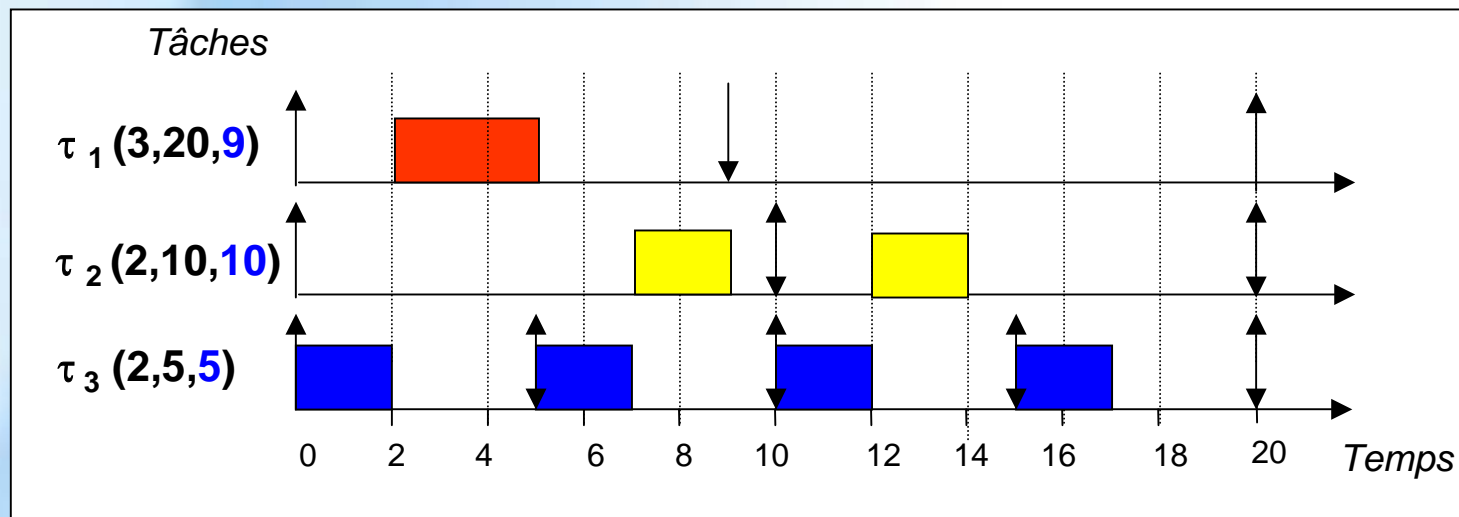
- Condition suffisante :
$$U = \sum_{i=1}^n \frac{C_i}{T_i} \leq n(2^{1/n} - 1)$$

$$U(1) = 1; \quad U(2) = 0.828; \quad U(3) = 0.779; \quad U(\infty) = \ln 2 \approx 0.693$$

Deadline Monotonic (DM)

- **Principe** : une tâche est d'autant plus prioritaire que son délai critique D_i est petit

- **Exemple** :



- **Propriété** : DM est **optimal** dans la classe des algorithmes à priorités fixes pour des tâches périodiques indépendantes telles que $D_i \leq T_i$

Deadline Monotonic (DM)

- Conditions de faisabilité :

- Condition nécessaire :
$$U = \sum_{i=1}^n \frac{C_i}{T_i} \leq 1$$

- Condition suffisante :
$$U = \sum_{i=1}^n \frac{C_i}{D_i} \leq n(2^{1/n} - 1)$$

$$U(1) = 1; \quad U(2) = 0.828; \quad U(3) = 0.779; \quad U(\infty) = \ln 2 \approx 0.693$$

L'ordonnancement à priorités fixes

- **Intérêts** : 

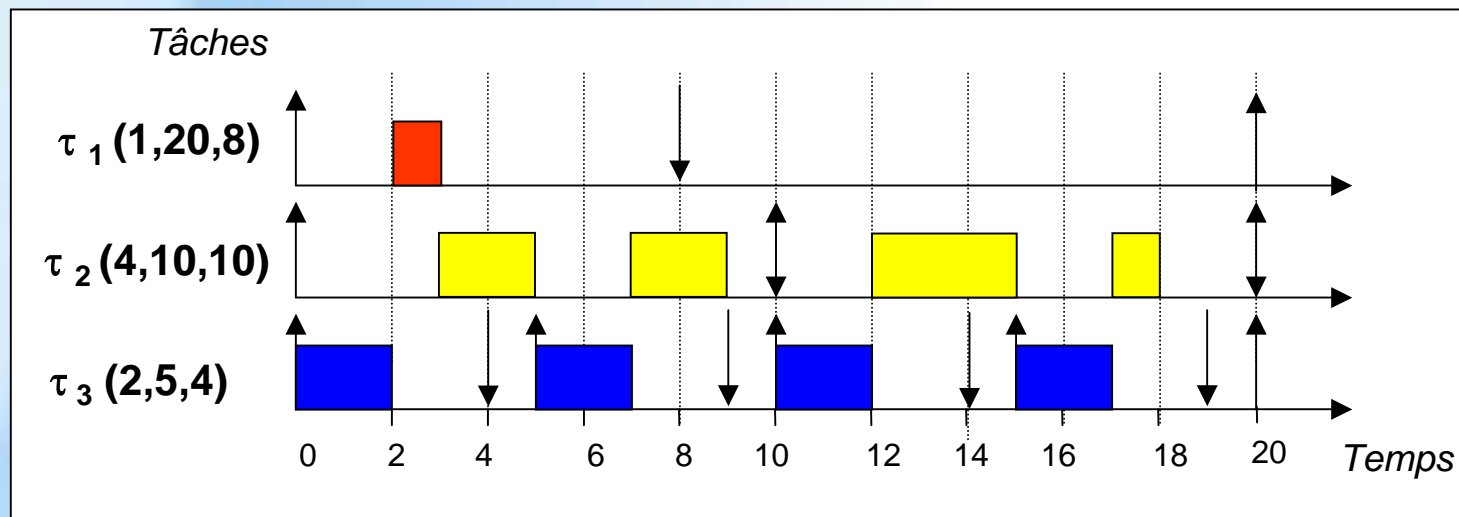
- Mécanisme simple
- S'implante naturellement dans les OS du marché

- **Inconvénients** : 

- Hypothèses restrictives
- Indépendance des tâches impérative pour l'utilisation des conditions de faisabilité
- Borne supérieure pour le facteur d'utilisation du processeur

Earliest Deadline First (EDF)

- **Principe** : à chaque instant, la tâche la plus prioritaire est celle dont l'échéance absolue d_i est la plus proche
- **Exemple** :



- **Propriété** : EDF est **optimal** dans la classe des algorithmes préemptifs pour des tâches périodiques indépendantes telles que $D_i \leq T_i$

Earliest Deadline First (EDF)

- Condition de faisabilité :

- si $D_i = T_i$:

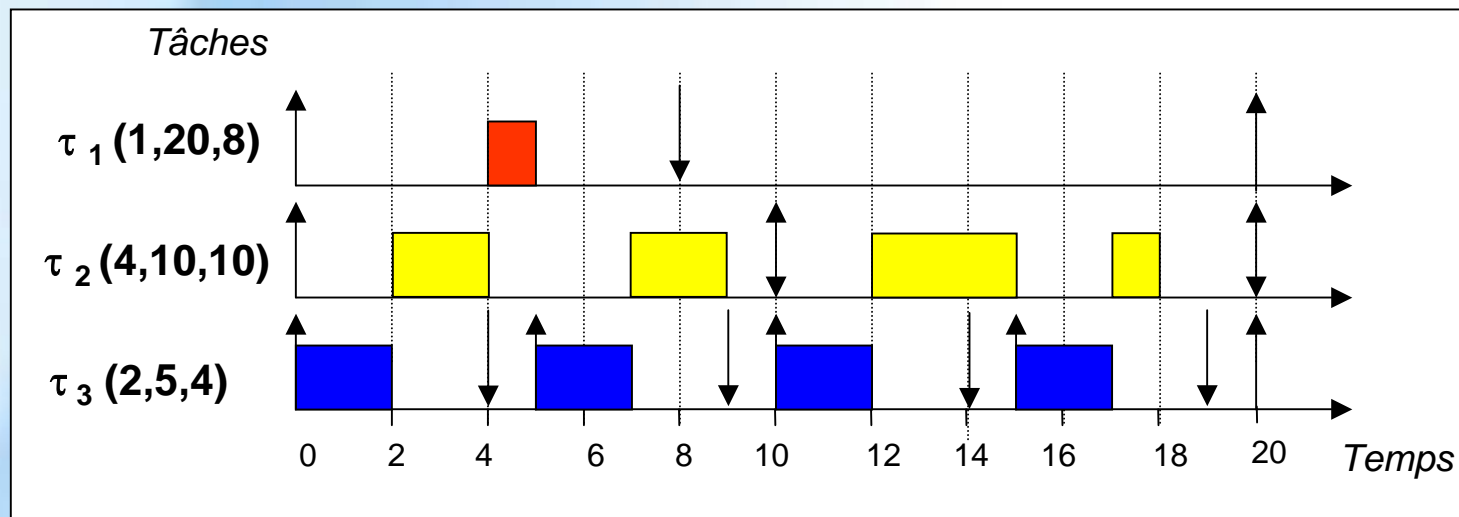
- Condition nécessaire et suffisante :
$$U = \sum_{i=1}^n \frac{C_i}{T_i} \leq 1$$

- si $D_i \leq T_i$:

- Condition suffisante :
$$U = \sum_{i=1}^n \frac{C_i}{D_i} \leq 1$$

Least Laxity First (LLF)

- **Principe** : à chaque instant, la tâche la plus prioritaire est celle dont la laxité $L(t) = r_i + D_i - (t + C_i(t))$ est la plus petite
- **Exemple** :



- **Propriété** : LLF est **optimal** dans la classe des algorithmes préemptifs pour des tâches périodiques indépendantes telles que $D_i \leq T_i$

Least Laxity First (LLF)

- Condition de faisabilité :

- si $D_i = T_i$:

- Condition nécessaire et suffisante :
$$U = \sum_{i=1}^n \frac{C_i}{T_i} \leq 1$$

- si $D_i \leq T_i$:

- Condition suffisante :
$$U = \sum_{i=1}^n \frac{C_i}{D_i} \leq 1$$

L'ordonnancement à priorités dynamiques

- Intérêts :

- Simplicité de mise en oeuvre
- Optimisation de l'usage des ressources
- Bien adapté aux tâches périodiques à courtes échéances

- Inconvénients :

- Indépendance des tâches impératives pour l'utilisation des conditions de faisabilité
- Instabilité en cas de surcharge (EDF)
- Nombreux changements de contexte dans certains cas (LLF)
- Difficilement implantable dans les OS actuels

L'ordonnancement de tâches apériodiques

- Objectif :

- cas de tâches apériodiques à **contraintes relatives**

→ minimiser le temps de réponse des tâches



- cas de tâches apériodiques à **contraintes strictes**

→ garantir le respect d'un maximum de tâches



L'ordonnancement de tâches apériodiques

- La gestion des tâches **en arrière-plan** :

- **Background scheduling (BG)**

- La gestion des tâches **par un serveur** :

à priorités fixes :

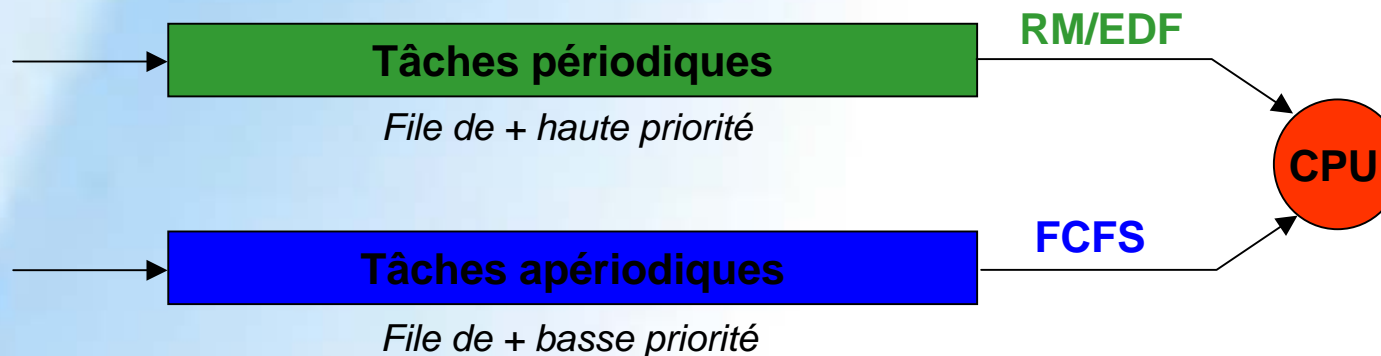
- **Polling Server (PS)**
- **Deferrable Server (DS)**
- Priority Exchange Server
- Sporadic Server
- Slack Stealer Server

à priorités dynamiques :

- Dynamic Sporadic Server (DSS)
- Dynamic Priority Exchange Server (DPE)
- Improved Priority Exchange Server (IPE)
- **Total Bandwidth Server (TBS)**
- **Earliest Deadline as Late as possible (EDL)**

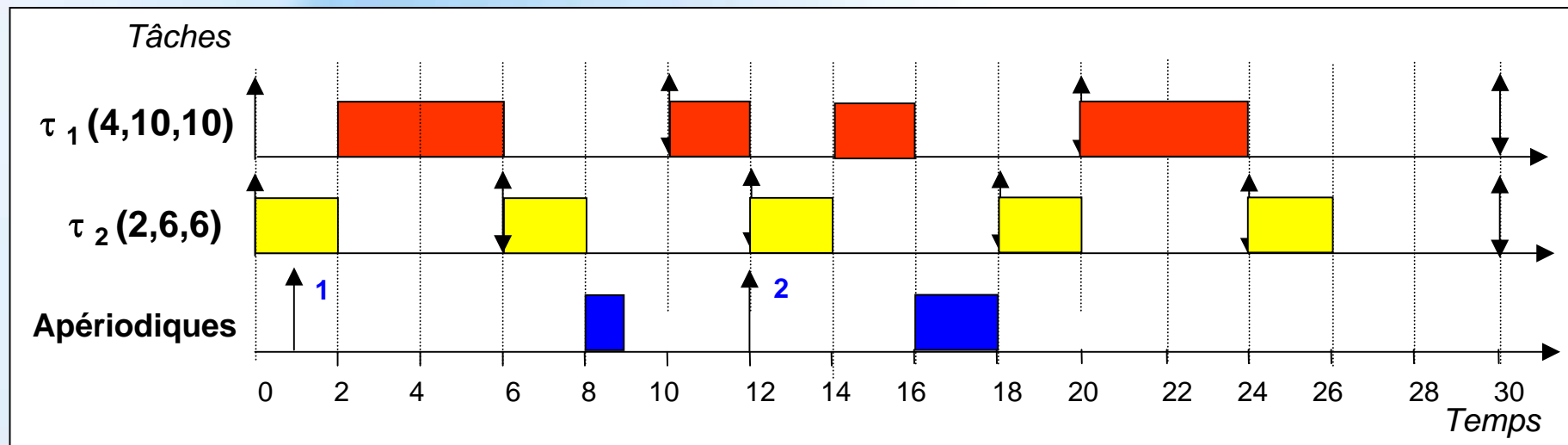
Background Scheduling (BG)

- **Principe** : les tâches apériodiques sont exécutées en tâches de fond, lorsqu'il n'y a pas de requêtes périodiques à l'état prêt
- **Mécanisme de fonctionnement** :



Background Scheduling (BG)

- Exemple (RM-BG):



Background Scheduling (BG)

- Intérêts : 

- Simplicité de mise en oeuvre
- Pas d'impact sur les tâches périodiques

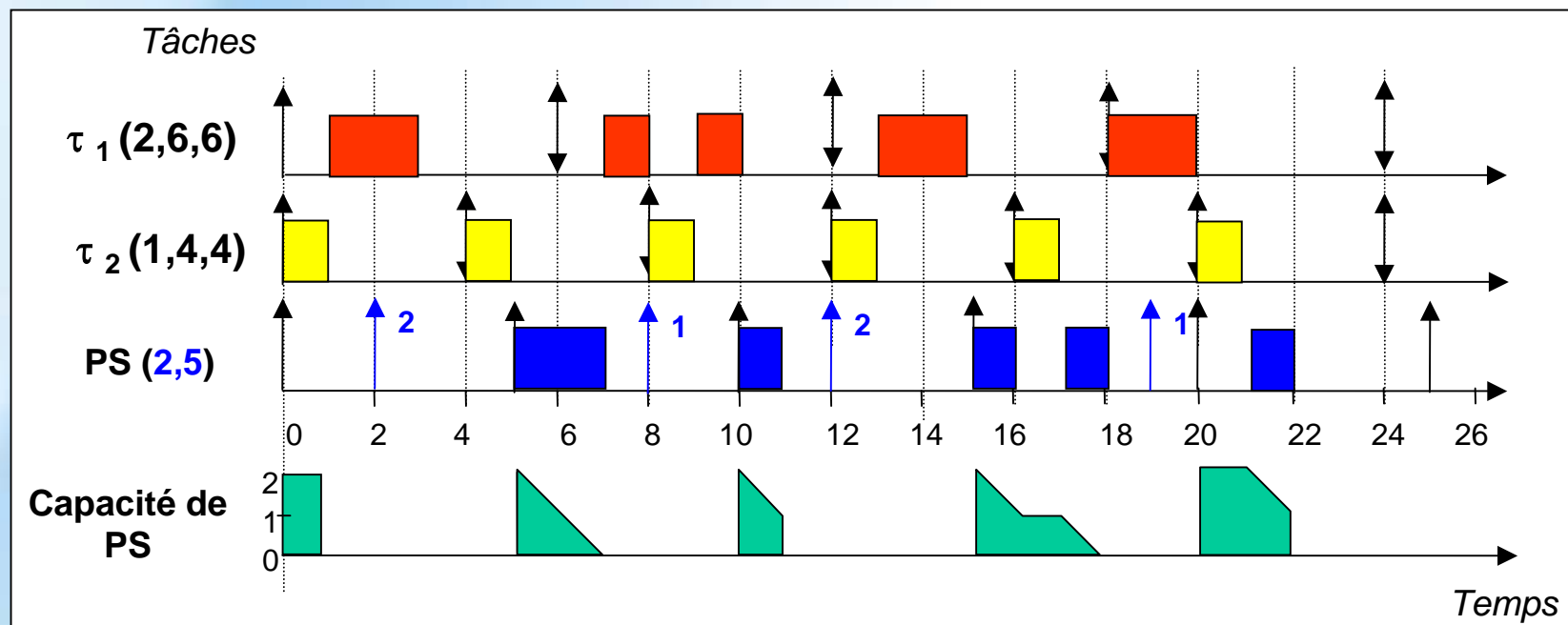
- Inconvénients : 

- Le temps de réponse des tâches apériodiques peut être élevé
- Applicable uniquement pour des tâches apériodiques à contraintes relatives
- Utilisable principalement dans des systèmes à charge modérée

Polling Server (PS)

- **Principe** : Une tâche périodique (C_s, P_s) appelée **serveur apériodique** active les tâches apériodiques dans son temps d'exécution appelé **capacité du serveur**

- **Exemple (RM-PS, $C_s=2; T_s=5$):**



Polling Server (PS)

- **Intérêts :** 

- Meilleures performances que celles obtenues avec la gestion des tâches apériodiques en tâches de fond
- Faibles complexités de calcul et d'implémentation

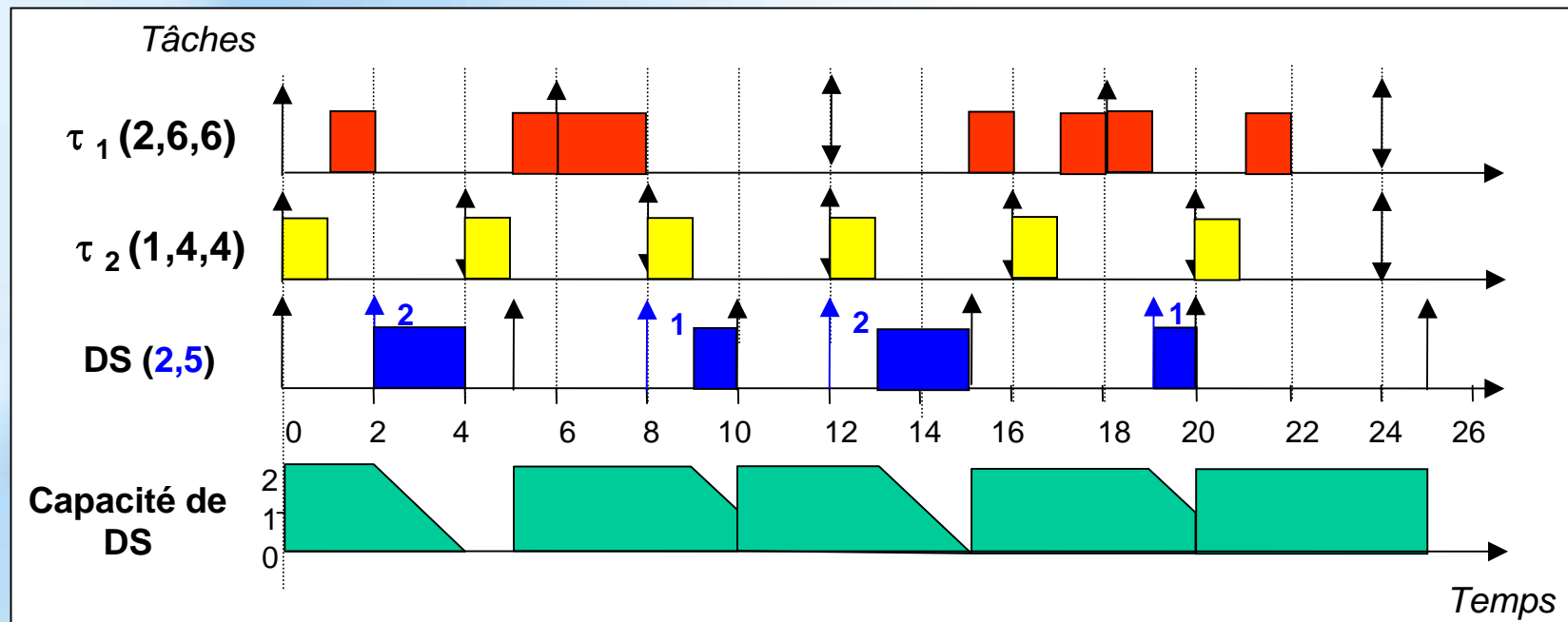
- **Inconvénients :** 

- La capacité du serveur est perdue en cas d'absence de tâche apériodique en attente lors du réveil du serveur
- Hypothèses restrictives de l'algorithme RM

Deferrable Server (DS)

- **Principe** : identique à celui du Polling server à l'exception que DS **conserve sa capacité courante** jusqu'à la fin de sa période d'activation

- **Exemple (RM-DS, $C_s=2$; $T_s=5$):**



Deferrable Server (DS)

- Intérêts : 

- Meilleures performances que celles obtenues avec le server Polling
- Faibles complexités de calcul et d'implémentation

- Inconvénients : 

- Hypothèses restrictives de l'algorithme RM



Total Bandwidth Server (TBS)

- **Principe** : Lorsque la $k^{\text{ème}}$ requête apériodique arrive au temps $t=r_k$, elle reçoit une **échéance fictive** calculée comme suit :

Date de réveil de la requête occurrente

Durée d'exécution de la requête occurrente

$$d_k = \max(r_k, d_{k-1}) + \frac{C_k^a}{U_s}$$

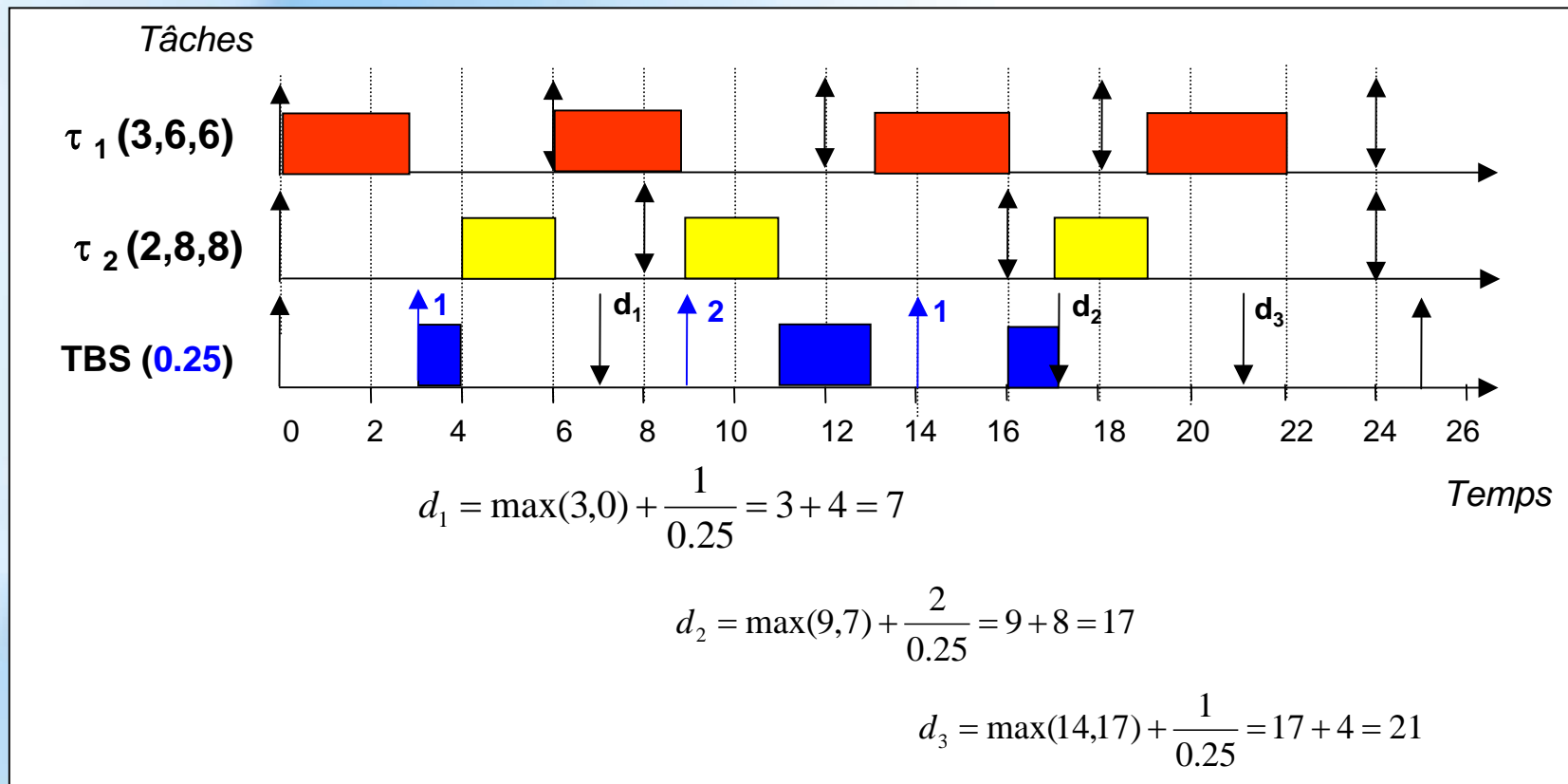
Largeur de bande CPU allouée au serveur

Échéance fictive de la requête précédente



Total Bandwidth Server (TBS)

- Exemple (EDF-TBS, $U_p=0.75$; $U_s=0.25$):





Total Bandwidth Server (TBS)

- Intérêts : 😊

- Mise en œuvre assez simple
- Faibles complexités de calcul et d'implémentation

- Inconvénients : 😞

- Performances assez médiocres pour des systèmes fortement chargés
- Nécessité de connaître les durées d'exécution des tâches apériodiques
occurrentes



Earliest Deadline as Late as possible (EDL)

- **Principe** selon l'algorithme suivant :

début

si (tâches apériodiques présentes) alors

- exécuter les tâches apériodiques au + tôt
- exécuter les tâches périodiques au + tard

sinon

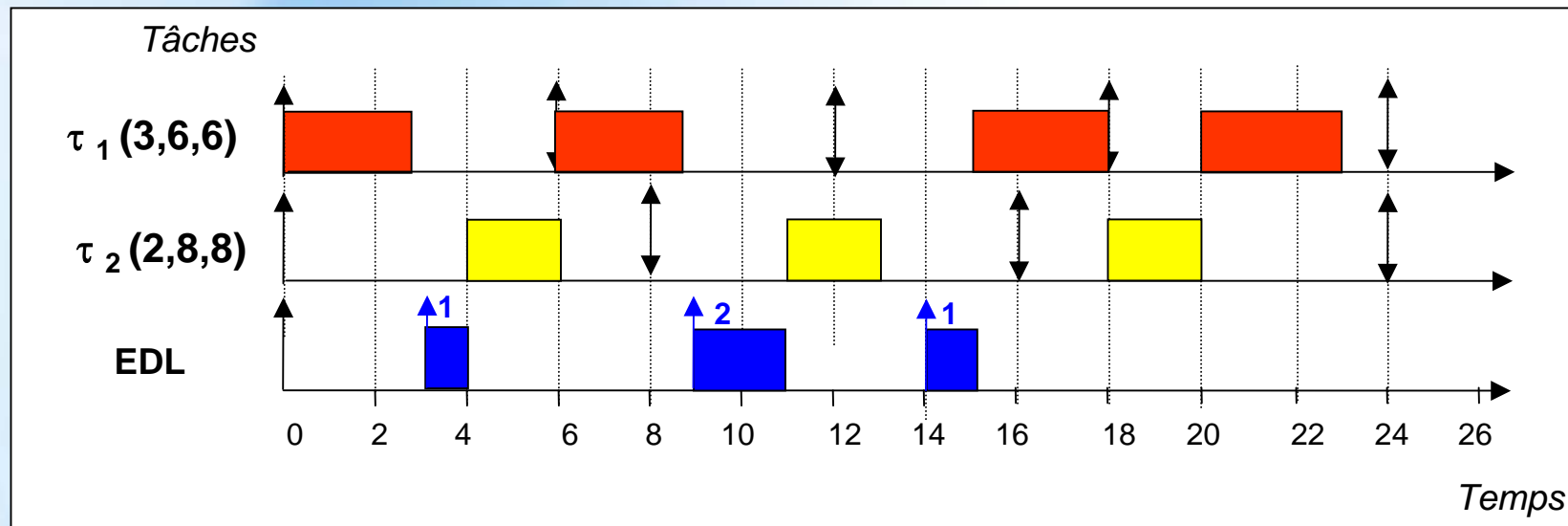
- exécuter les tâches périodiques au + tôt

fin



Earliest Deadline as Late as possible (EDL)

- Exemple (EDF-EDL) :





Earliest Deadline as Late as possible (EDL)

- Intérêts : 😊

- Serveur optimal
- Pas de nécessité de connaître les durées d'exécution des tâches apériodiques occurrentes

- Inconvénients : 😞

- Complexités de calcul et d'implémentation élevées
- Besoins mémoire importants

Conclusion (1)

- En bref :

- L'ordonnanceur a pour rôle d'allouer le processeur aux différentes tâches
- L'ordonnancement temps-réel
 - est régi par une politique spécifique basé sur la priorité
 - peut être hors-ligne ou en-ligne
 - doit être déterministe
- Compromis simplicité / performances des algorithmes d'ordonnancement
- Quantification de l'overhead d'ordonnancement

Conclusion (2)

- Programmation synchrone / asynchrone

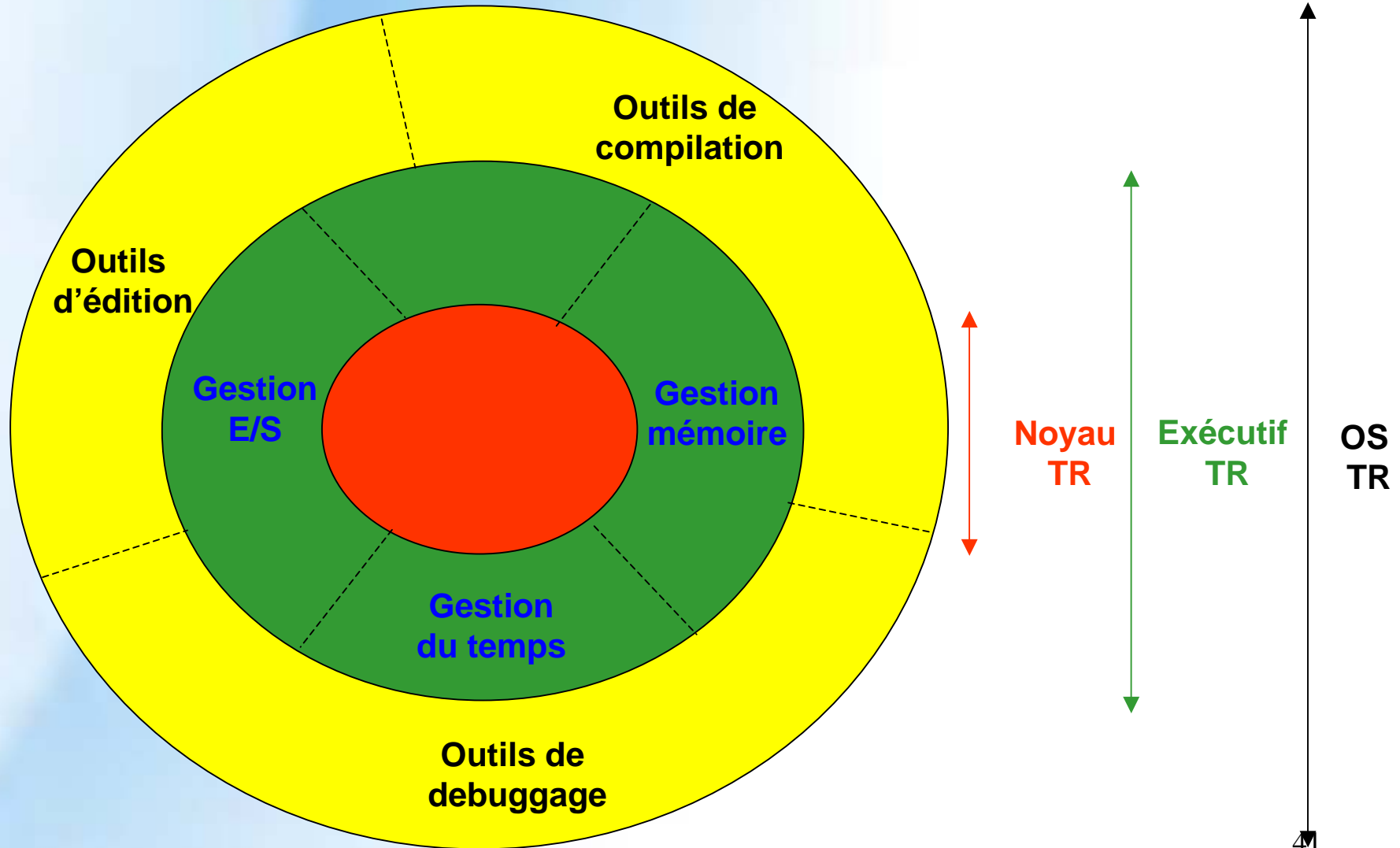
- **Modèle de temps synchrone** – **Cadre d'étude**

- Hypothèse des temps de calculs et des temps de communications *nuls*
- *Simultanéité* possible pour les événements
- *Contrôle logique et temporel* de l'application → vérification formelle

- **Modèle de temps asynchrone** – **Monde réel**

- Temps de calculs éventuellement longs
- *Non-simultanéité* des événements
- Difficultés voire impossibilité de preuves de correction de l'application

Extension à la notion d'exécutif temps-réel



Références

1. Giorgio Buttazzo, ***HARD REAL-TIME COMPUTING SYSTEMS: Predictable Scheduling Algorithms and Applications***, Second Edition, Springer, 2004.
2. J. Stankovic, K. Ramamritham, M. Spuri, and G. Buttazzo, ***Deadline Scheduling for Real-Time Systems***, Kluwer Academic Publishers, Boston, 1998.
3. J. W. S. Liu, ***Real-Time Systems***, Prentice Hall, New Jersey, 2000.