Plan du cours

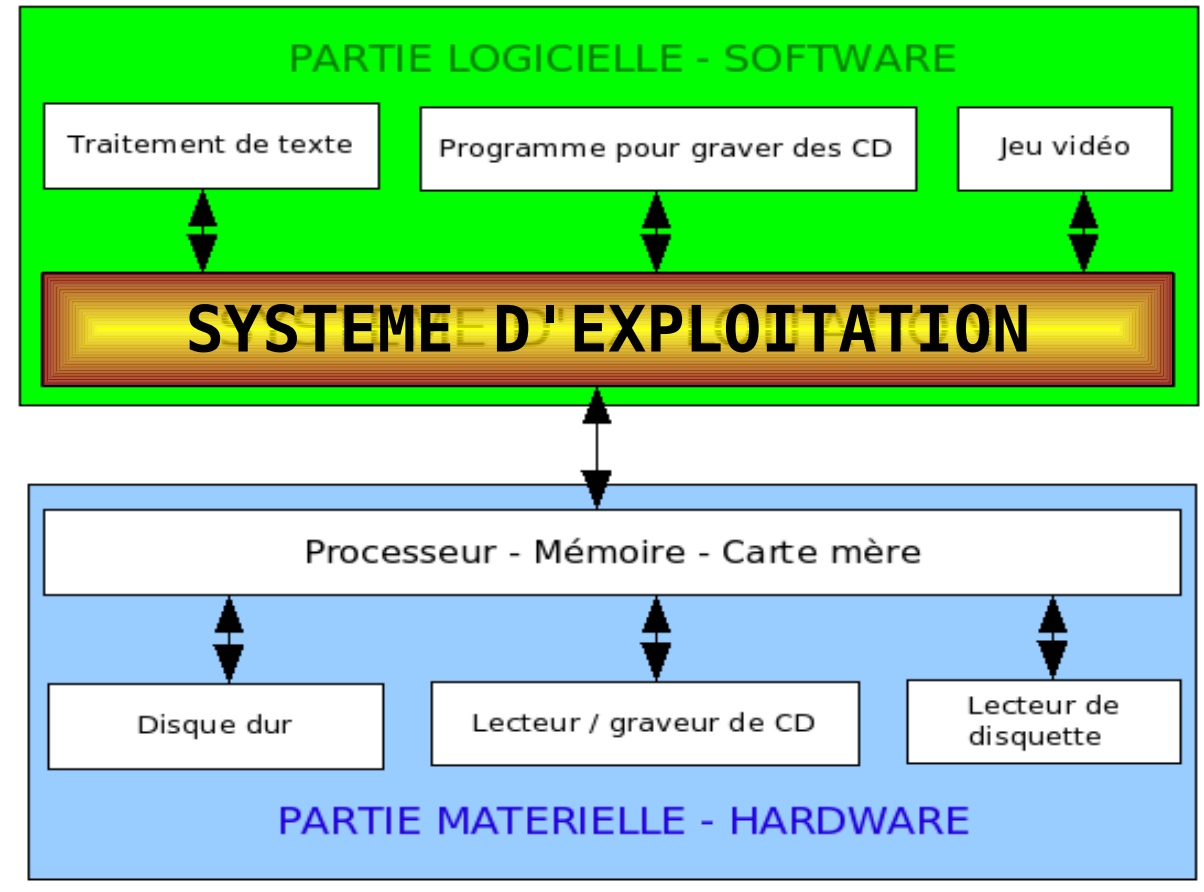
- Taxonomie des systèmes informatiques
- Systèmes temps réel
- **Spécificités des OS pour le temps réel**
- L'OS Xenomai pour le temps réel
- Systèmes embarqués
- Linux pour l'embarqué
- Marché des OS pour le temps réel et l'embarqué
- Modélisation d'applications temps réel avec UML 2.x

Rappels sur les OS (1)

Applications

Matériel

Rappels sur les OS (2)



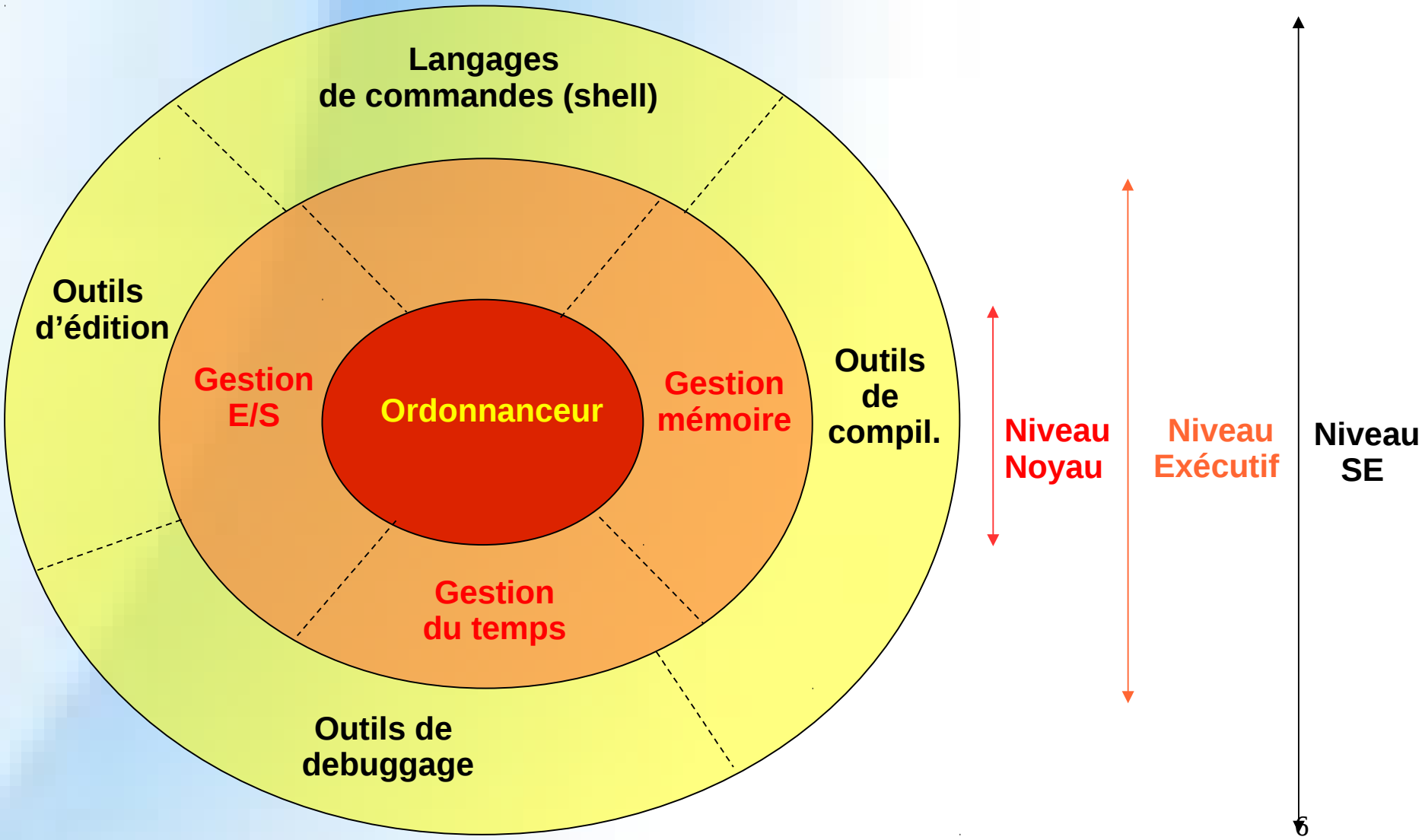
Rappels sur les OS (3)

- **Logiciel** destiné à faciliter et simplifier l'utilisation d'un ordinateur
- **Interface** entre l'utilisateur et le matériel (= abstraction des spécificités d'accès complexes du matériel)
- Le SE gère les ressources matérielles
- Le SE réalise 4 grands types de tâches :
  - La gestion des processus
  - La gestion de la mémoire
  - La gestion du système de fichiers
  - La gestion des périphériques d'E/S

Systèmes d'exploitation temps réel (1)

- Noyau ou moniteur temps réel
- Exécutif temps réel
- Système d'exploitation temps réel

Systèmes d'exploitation temps réel (2)



Systèmes d'exploitation temps réel (3)

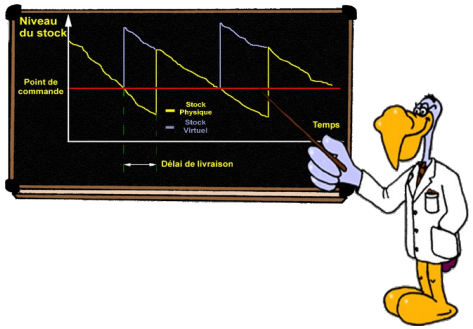
- **Principaux composants influant sur les performances d'un RTOS** :
  - Horloge temps réel
  - Gestionnaire des tâches
  - Gestionnaire d'interruptions
  - Gestionnaire de la mémoire
  - Gestionnaire d'E/S

Problématiques pour le temps réel (1)

- Quelles architectures logicielles sont à considérer ?
  - Noyaux monolithiques
  - Micro-noyaux
  - Micro-noyaux hybrides ou extensibles
  - Nano-noyaux
  - Micro-noyaux temps réel hybrides

Problématiques pour le temps réel (2)

- Quelles fonctionnalités sont à adapter ?
  - La gestion des **tâches** (ordonnancement)
  - Les **protocoles d'accès aux ressources** (synchronisation)
  - La gestion des **interruptions**
  - La gestion de la **mémoire**

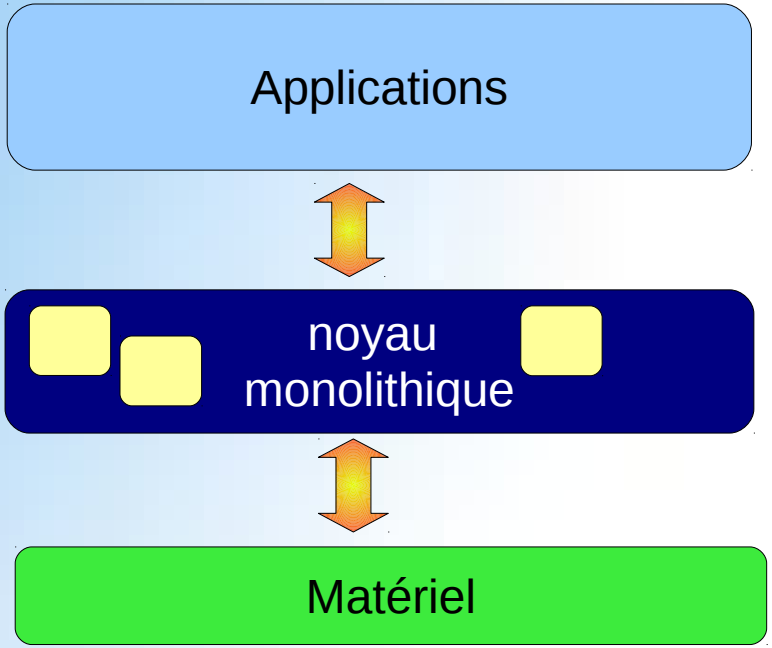


Problématique temps réel n°1

Les architectures logicielles...

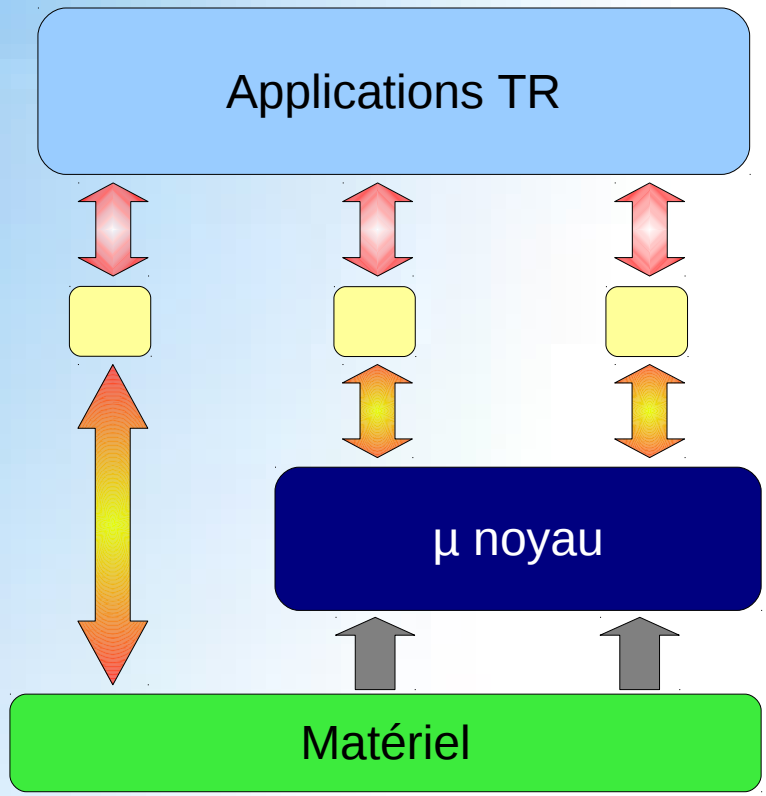
Architectures logicielles pour le temps réel (1)

- Noyaux monolithiques



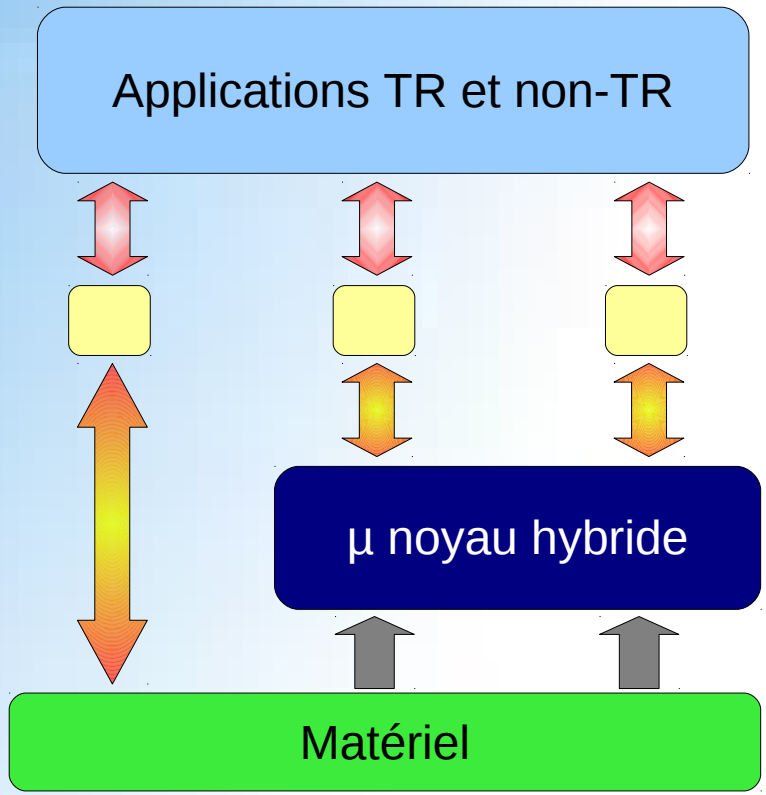
Architectures logicielles pour le temps réel (2)

- Micro-noyaux



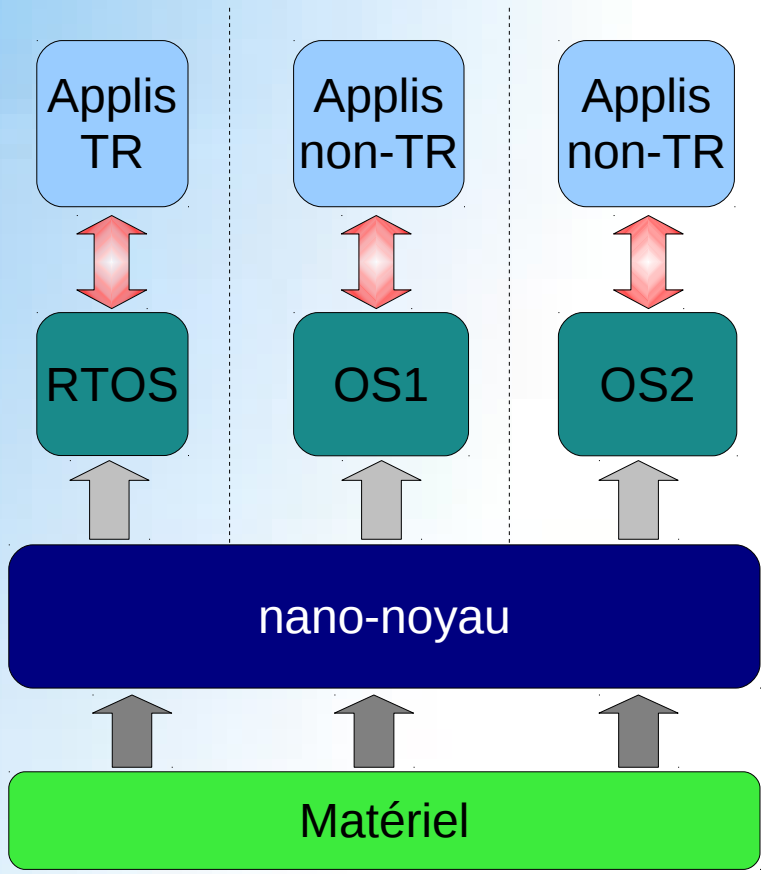
Architectures logicielles pour le temps réel (3)

- Micro-noyaux hybrides ou extensibles



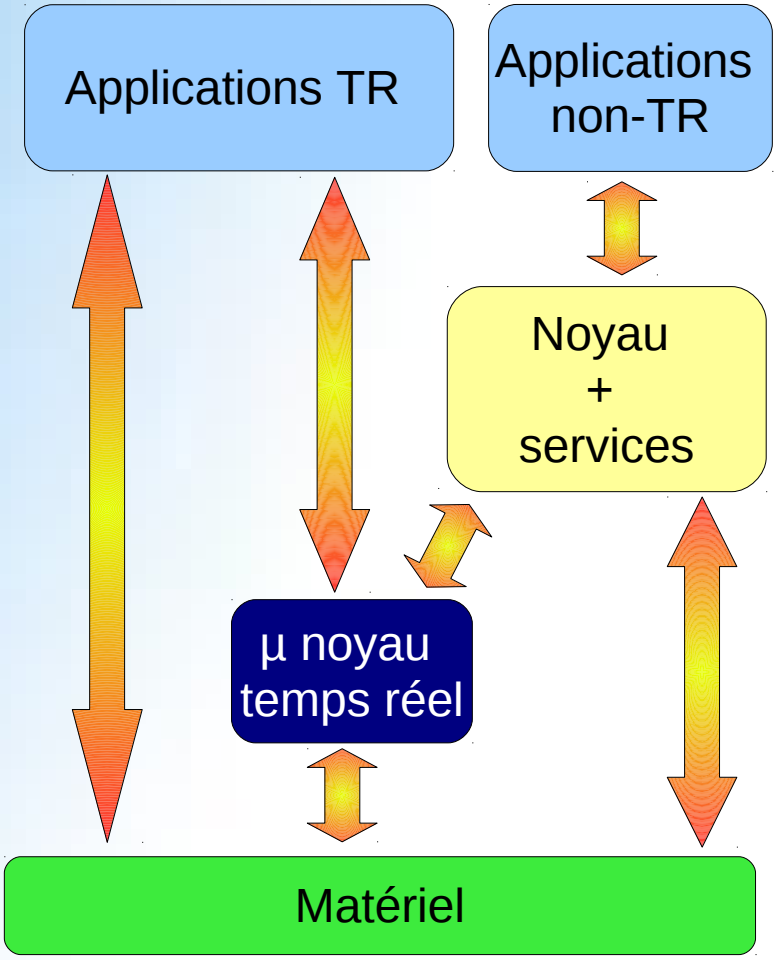
Architectures logicielles pour le temps réel (4)

- Nano-noyaux



Architectures logicielles pour le temps réel (5)

- Micro-noyaux temps réel hybrides



Problématique temps réel n°2

La gestion des tâches...

La gestion des tâches (1)

- Types de stimuli → tâches temps réel périodiques et/ou apériodiques

La gestion des tâches (2)

- **Tâches temps réel périodiques T_i**

r_i : date de réveil

D_i : délai critique

P_i : période d'activation

$d_i = r_i + D_i$: date d'échéance

C_i : durée d'exécution maximale

$L_i = D_i - C_i$: laxité

La gestion des tâches (3)

- **Tâches temps réel apériodiques non critiques R_i**

  r_i : date de réveil

  C_i : durée d'exécution maximale

- **Tâches temps réel apériodiques critiques S_i**

  r_i : date de réveil

  C_i : durée d'exécution maximale

  D_i : délai critique

  d_i : date d'échéance

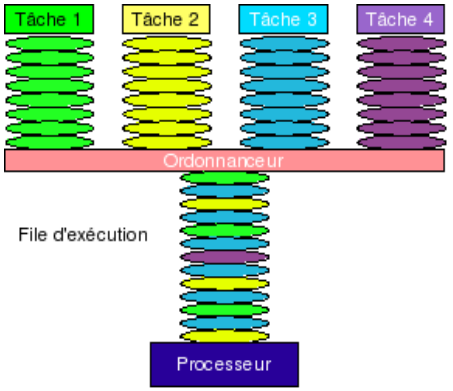
La gestion des tâches (4)

- **Niveaux de contraintes temporelles des tâches**
  - Temps-réel **strict/dur** (*hard real-time*)
  - Temps-réel **souple/mou** (*soft real-time*)
  - Temps-réel **ferme** (*firm real-time*)

L'ordonnancement des tâches (1)

- L'ordonnancement consiste à **choisir** le processus à exécuter à un instant *t* et à **déterminer** le temps durant lequel le processeur lui sera alloué

- L'**objectif de l'ordonnanceur** est d'optimiser certains aspects des performances du système.

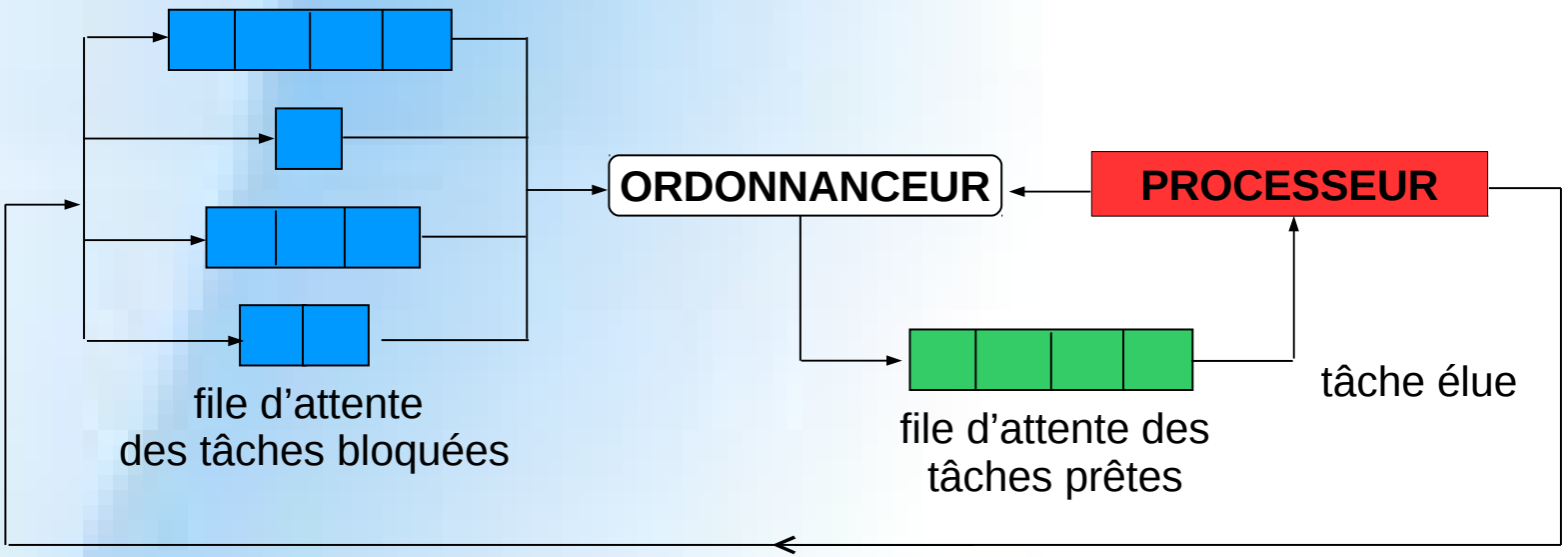


- **Compromis** entre :
  - Temps de traitement moyen du système
  - Utilisation efficace du processeur
  - Temps de réponse moyen/max du système
  - Satisfaction des conditions d'échéance pour les processus
  - Bonne utilisation des autres ressources du système

L'ordonnancement des tâches (2)

- **Systèmes de traitement par lots**
  - optimiser le nombre de tâches à l'heure
  - réduire le délai entre la soumission et l'achèvement
  - faire en sorte que le processeur soit occupé en permanence
- **Systèmes interactifs**
  - répondre rapidement aux requêtes
  - répondre aux attentes des utilisateurs
- **Systèmes temps réel**
  - respecter les délais
  - eviter de perdre des données
  - eviter la dégradation de la qualité au niveau des résultats produits

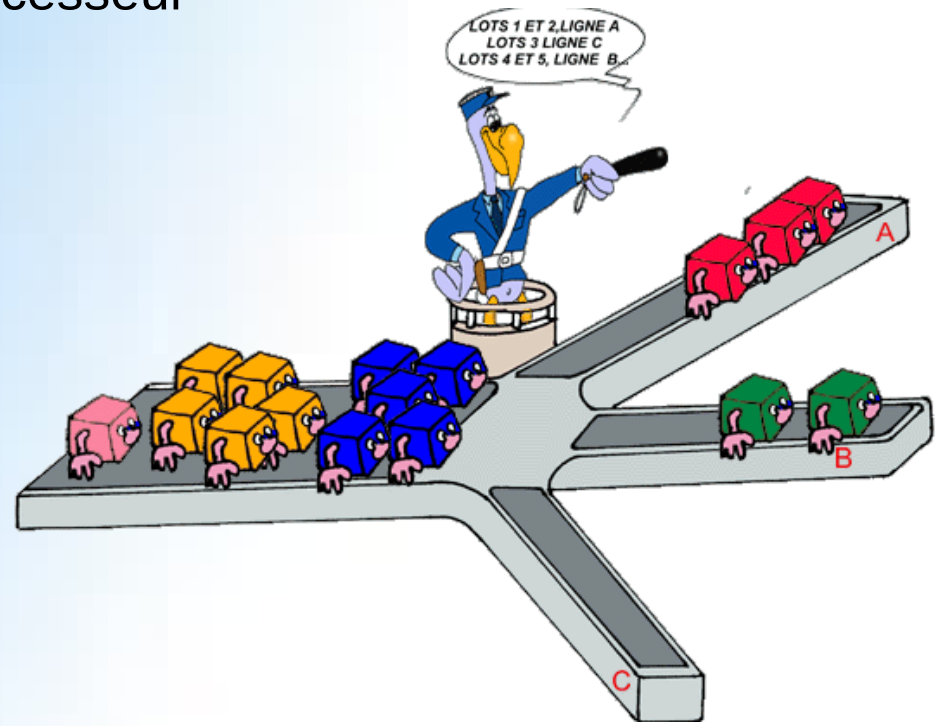
L'ordonnancement des tâches (3)



- **ORDONNANCEUR** : alloue le processeur aux différentes tâches selon un algorithme d'ordonnancement donné

Typologie des algorithmes d'ordonnancement

- Monoprocesseur / multiprocesseur
- En-ligne / Hors-ligne
- Préemptif / Non préemptif
- Oisif / Non oisif
- Centralisé / Réparti



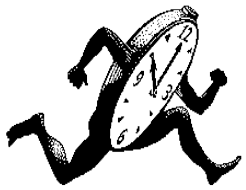
Ordonnancement non préemptif (non temps réel)

- Ordonnancement **selon l'ordre d'arrivée** :
  - premier arrivé, premier servi

  First Come First Serve (FCFS)

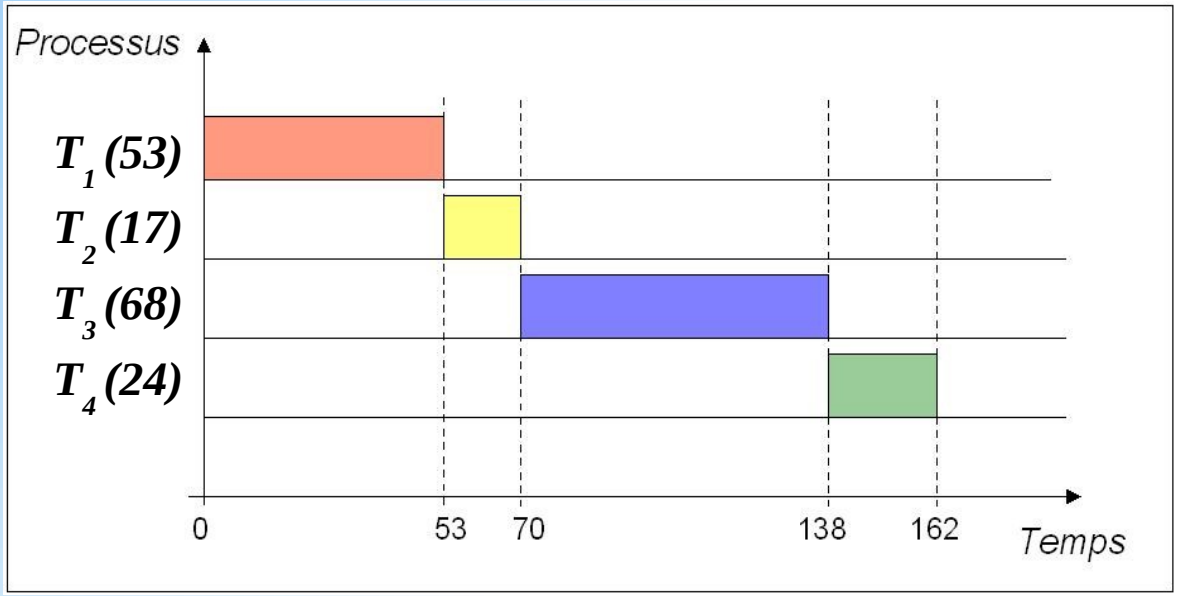
- Ordonnancement **selon la durée de calcul** :
  - travail le plus court d'abord

  Shortest Job First (SJF)



Ordonnancement First Come First Served (FCFS)

- **Principe** : Les tâches sont ordonnancées selon leur ordre d'arrivée
- **Exemple** (T_1, T_2, T_3 et T_4 arrivent dans cet ordre à t = 0) :



Ordonnancement First Come First Served (FCFS)

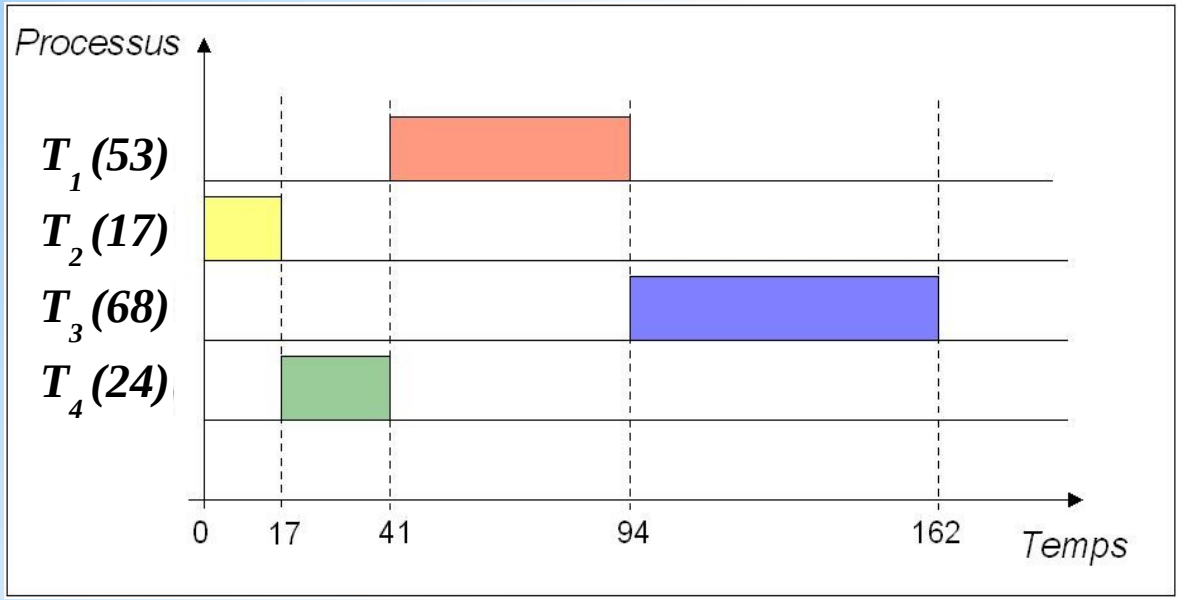
- **Intérêts** : 
  - Algorithme facile à comprendre
  - Faible complexité d'implémentation (une seule liste chaînée)

- **Inconvénients** : 
  - Pas de prise en compte de l'importance relative des tâches
  - Temps d'attente du processeur généralement important

Ordonnancement Shortest Job First (SJF)

- **Principe** : La tâche dont le temps d'exécution est le plus court est ordonnancée en priorité

- **Exemple** (T_1, T_2, T_3 et T_4 arrivent à t = 0) :



Ordonnancement Shortest Job First (SJF)

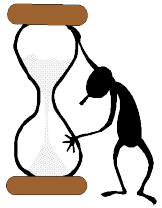
- **Intérêts** : 
  - SJF réduit le temps d'attente des processus
  - Utilisation limitée à des environnements et à des applications spécifiques

- **Inconvénients** : 
  - Pas de prise en compte de l'importance relative des tâches
  - Algorithme optimal uniquement dans le cas où toutes les tâches sont
  - disponibles simultanément

Ordonnancement préemptif (non temps réel)

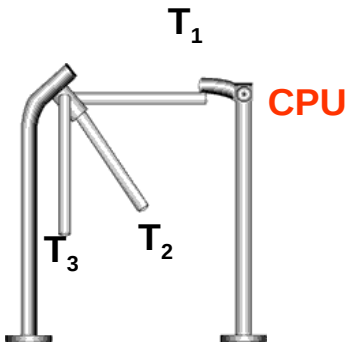
- Ordonnancement **selon la durée de calcul restante** :
  - temps restant le plus court d'abord

  Shortest Remaining Time (SRT)



- Ordonnancement **sans notion de priorité** :
  - temps-partagé avec politique du tourniquet

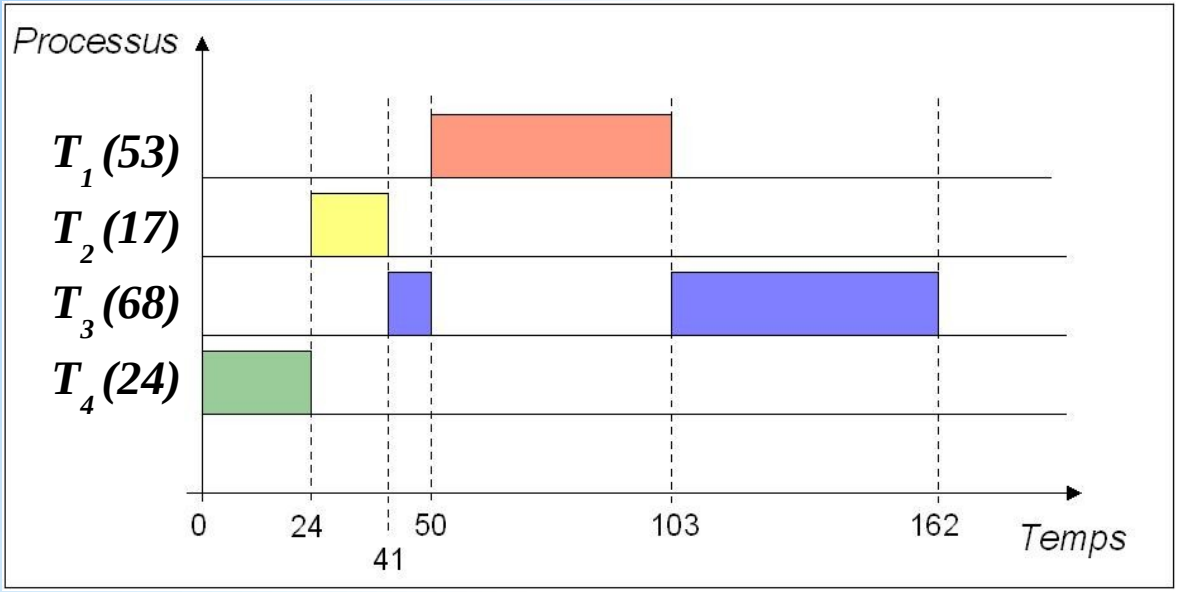
  Round-Robin (RR)



Ordonnancement Shortest Remaining Time (SRT)

- **Principe** : La tâche dont le temps d'exécution restant est le plus court parmi ceux qui restent à exécuter, est ordonnancée en premier

- **Exemple** (T_3 et T_4 arrivent à $t = 0$; T_2 à $t = 20$; T_1 à $t = 50$) :



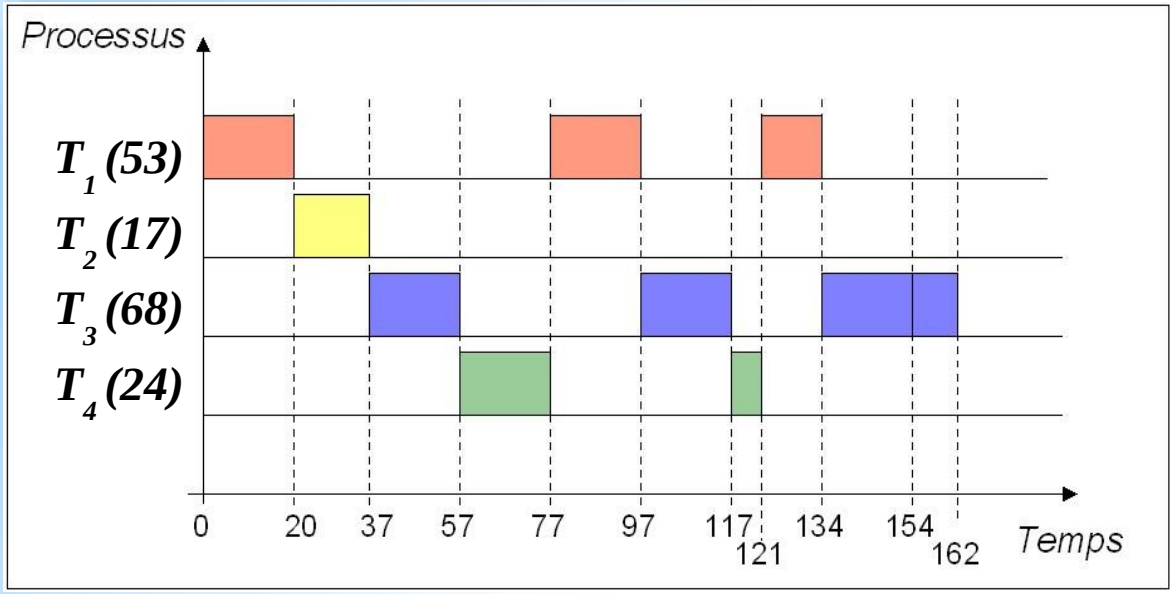
Ordonnancement Shortest Remaining Time (SRT)

- **Intérêts** : 
  - SRT minimise le temps d'attente moyen des tâches les plus courtes
  - Utilisation limitée à des environnements et à des applications spécifiques

- **Inconvénients** : 
  - Pas de prise en compte de l'importance relative des tâches
  - Non équité de service : SRT pénalise les tâches longues
  - Possibilité de famine pour les tâches longues

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

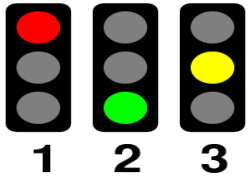
Ordonnancement temps-partagé (Round-Robin)

- **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 !

Ordonnancement temps réel

- Ordonnancement **préemptif à priorités** :
  - la tâche la plus prioritaire obtient le processeur



- Ordonnancement **à priorités fixes (statiques)** :
  - Rate Monotonic (RM)
  - Deadline Monotonic (DM)
- Ordonnancement **à priorités dynamiques** :
  - Earliest Deadline First (EDF)
  - Least Laxity First (LLF)

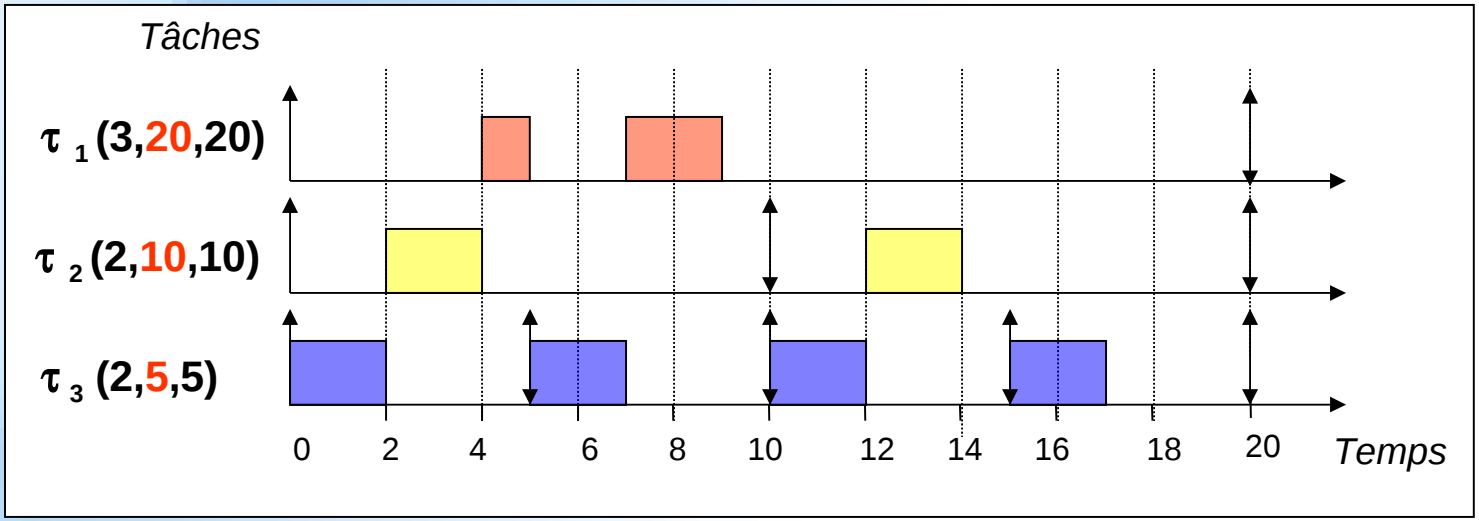
Métriques usuelles en ordonnancement temps réel

- Ordonnançabilité
  - Tests exacts
  - Tests suffisants
  - Tests nécessaires
- Optimalité
- Overhead

Rate Monotonic (RM)

- **Principe** : une tâche est d'autant plus prioritaire que sa période d'activation P_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 ($D_i=P_i$)

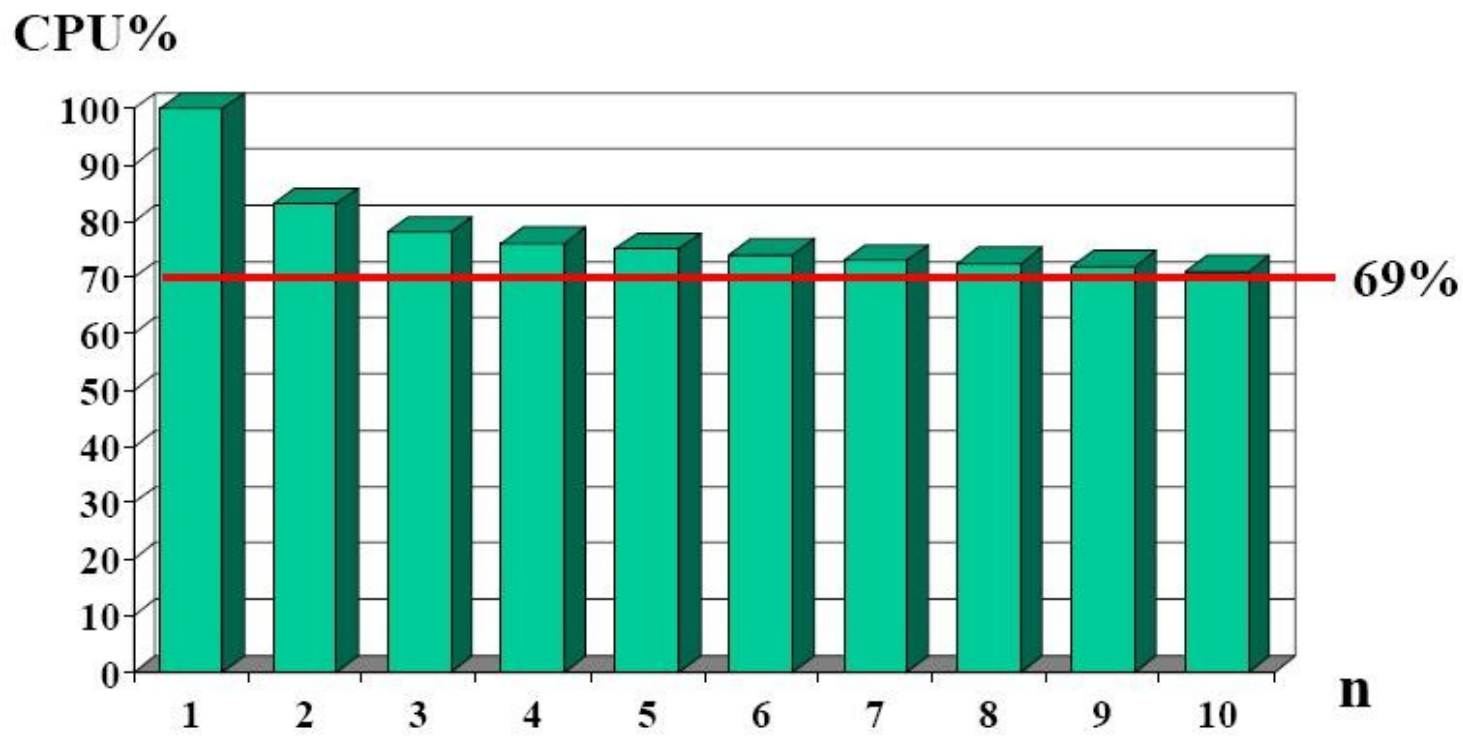
Rate Monotonic (RM)

- **Conditions d'ordonnançabilité :**
  - Condition nécessaire : $$U = \sum_{i=1}^{n} \frac{C_i}{P_i} \leq 1$$
  - Condition suffisante : $$U = \sum_{i=1}^{n} \frac{C_i}{P_i} \leq n(2^{1/n} - 1)$$

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

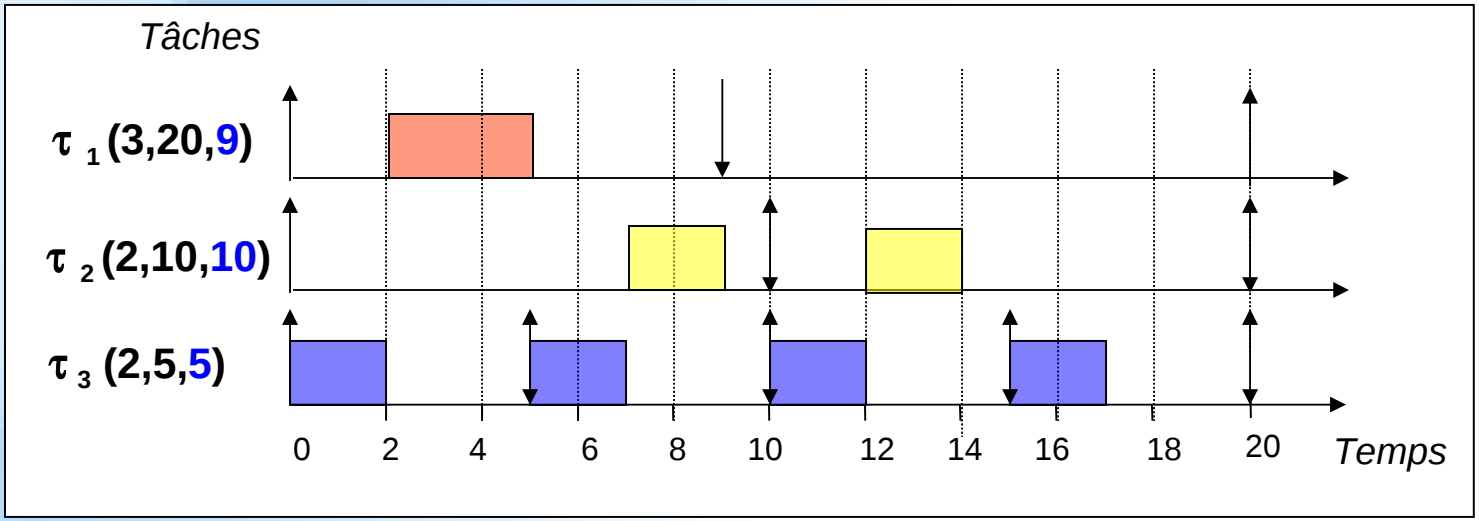
Rate Monotonic (RM)

- **Limite d'ordonnançabilité :**



Deadline Monotonic (DM)

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



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

Deadline Monotonic (DM)

- **Conditions d'ordonnançabilité :**
  - Condition nécessaire : $$U = \sum_{i=1}^{n} \frac{C_i}{P_i} \leq 1$$
  - Condition suffisante : $$U = \sum_{i=1}^{n} \frac{C_i}{D_i} \leq n(2^{1/n} - 1)$$

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

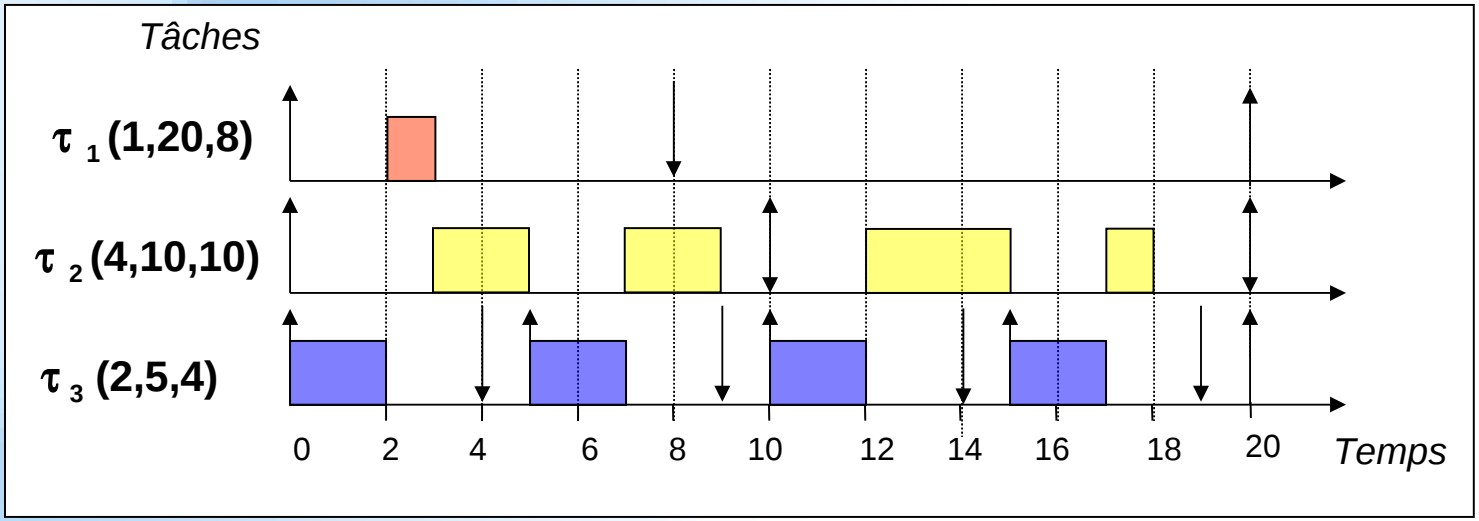
L'ordonnancement à priorités fixes

- **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 La date d'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 $D_i \leq P_i$

Earliest Deadline First (EDF)

- **Conditions d'ordonnançabilité :**
  - **si $D_i=P_i$:**

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

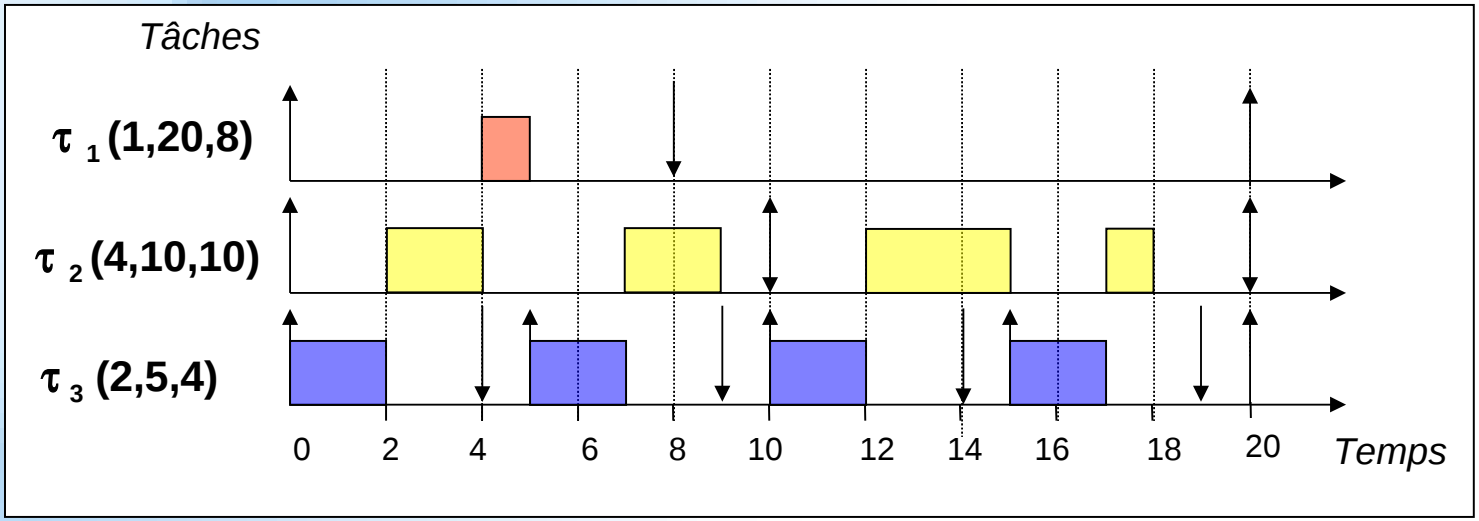
  - **si $D_i<P_i$:**

    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 $L(t) = d_i - t - C_i(t)$ est la plus petite

- **Exemple** :



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

Least Laxity First (LLF)

- **Conditions d'ordonnançabilité :**
  - **si $D_i=P_i$:**

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

  - **si $D_i \leq P_i$:**

    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 **non critiques**

    → minimiser le temps de réponse des tâches

  - cas de tâches apériodiques **critiques**

    → garantir le respect d'un maximum de tâches

L'ordonnancement de tâches apériodiques

- La gestion des tâches **en arrière-plan** :
  - Background scheduling (BG)

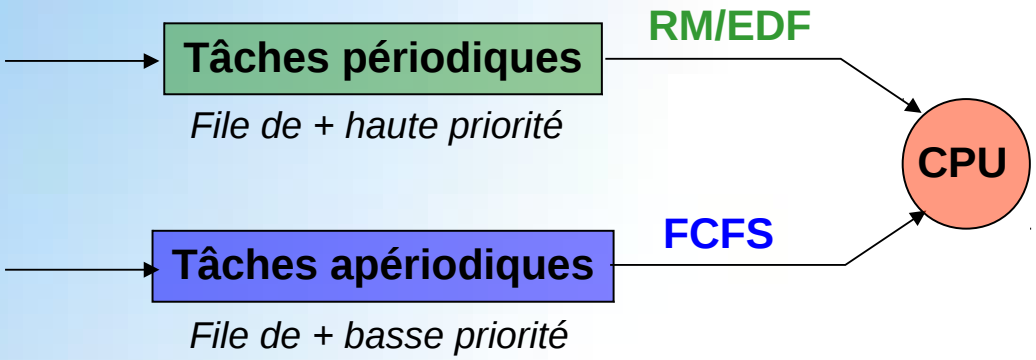
- La gestion des tâches **par un serveur** :

à priorités fixes :	à priorités dynamiques :
- Polling Server (PS)	- Dynamic Sporadic Server (DSS)
- Deferrable Server (DS)	- Dynamic Priority Exchange Server (DPE)
- Priority Exchange Server	- Improved Priority Exchange Server (IPE)
- Sporadic Server (SS)	- Total Bandwidth Server (TBS)
- Slack Stealer Server	- Earliest Deadline as Late as possible (EDL)

Background Scheduling (BG)

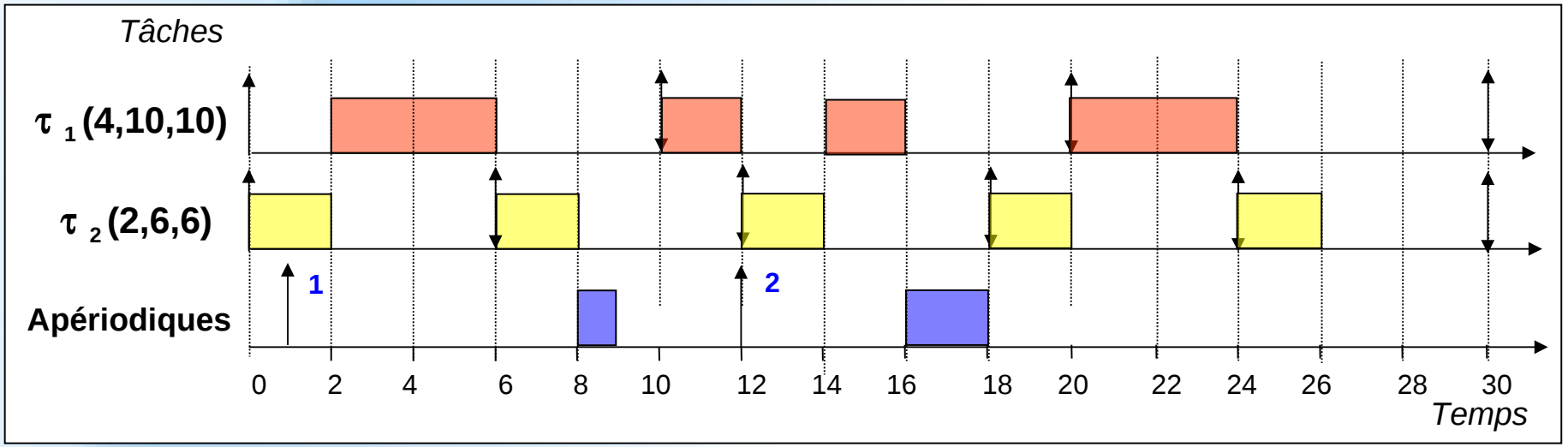
- **Principe** : les tâches apériodiques sont exécutées en tâches de fond, lorsqu'il n'y a pas de requêtes périodiques à l'état prêt

- **Mécanisme de fonctionnement :**



Background Scheduling (BG)

- **Exemple (RM-BG):**



Background Scheduling (BG)

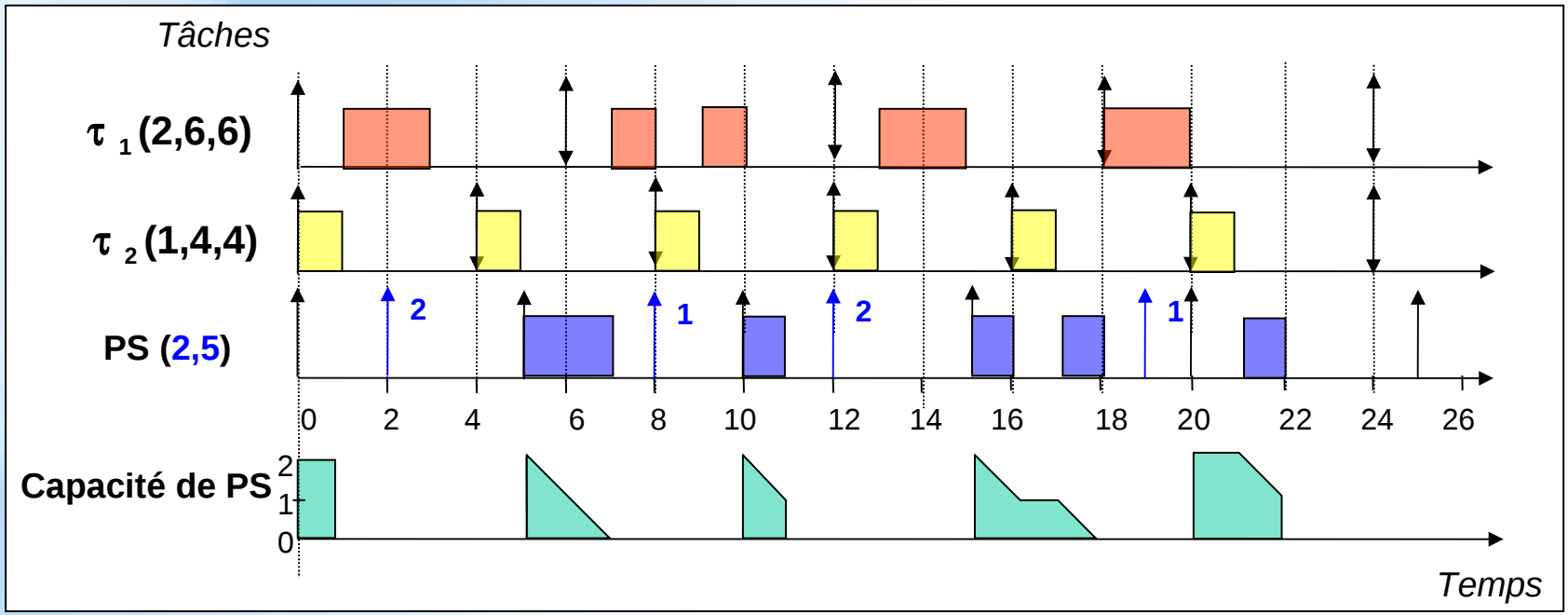
- **Intérêts** : 
  - Simplicité de mise en oeuvre
  - Pas d'impact sur les tâches périodiques

- **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, T_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)

- Condition d'ordonnançabilité sous RM-PS :
  - Condition suffisante :

$$\frac{C_s}{T_s} + \sum_{i=1}^{n} \frac{C_i}{P_i} \leq (n+1)\left(2^{1/(n+1)} - 1\right)$$

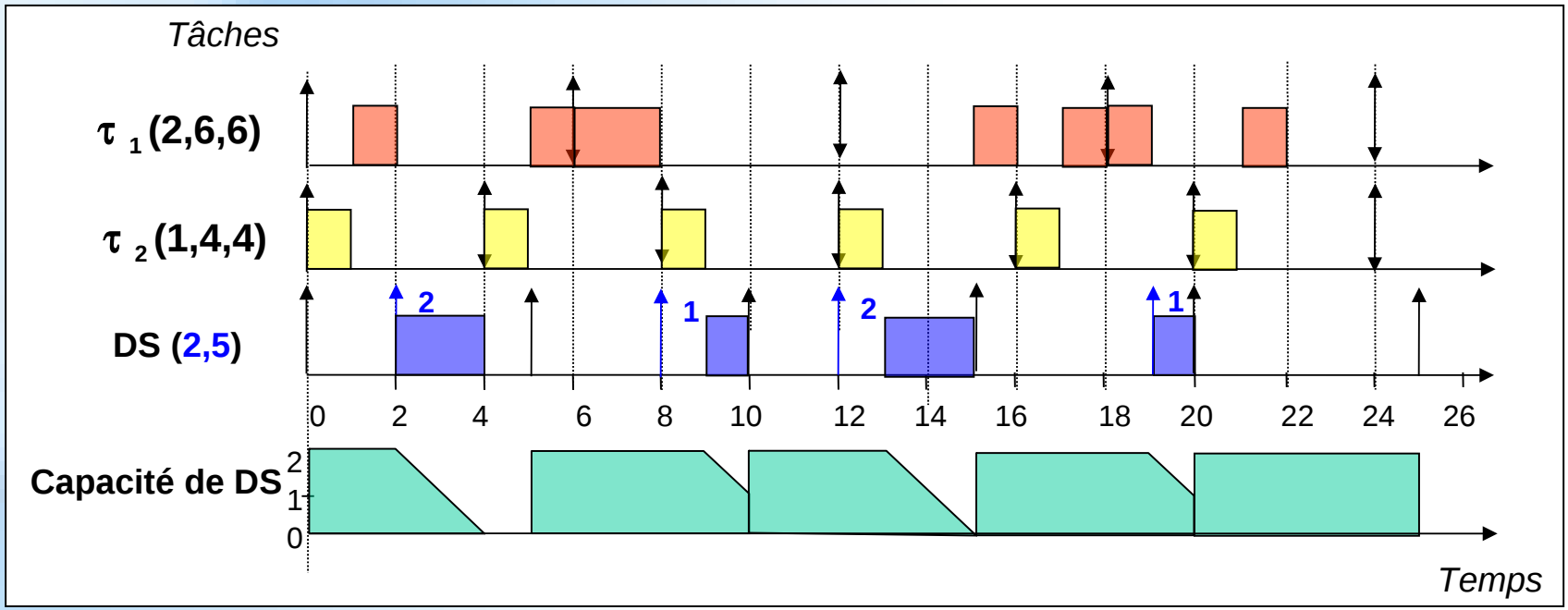
Polling Server (PS)

- **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)

- Condition d'ordonnançabilité sous RM-DS :
  - Condition suffisante :

$$Up \leqslant n\left[\left(\frac{Us+2}{2\mathrm{Us}+1}\right)^{1/n}-1\right]$$

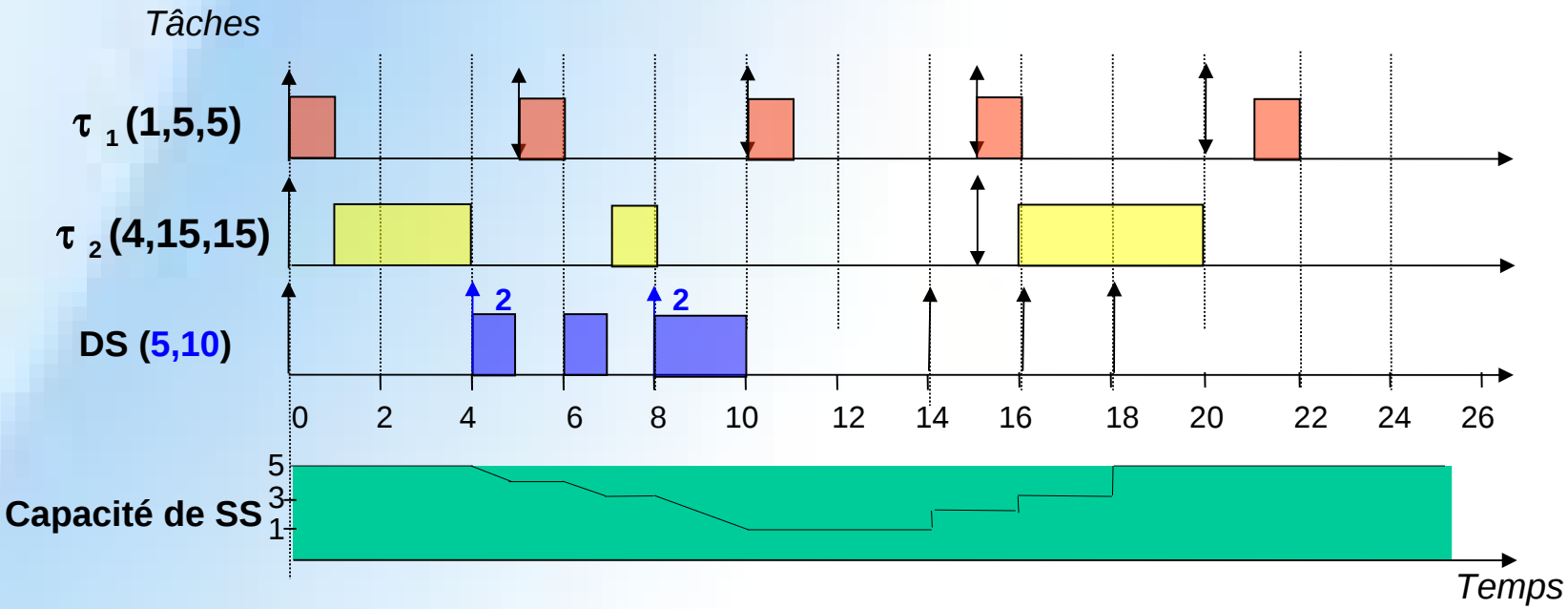
Deferrable Server (DS)

- **Intérêts** : 
  - Meilleures performances que celles obtenues avec le server Polling
  - Faibles complexités de calcul et d'implémentation
- **Inconvénients** : 
  - Hypothèses restrictives de l'algorithme RM

Sporadic Server (SS)

- **Principe** : identique à celui de DS à l'exception que SS « reremplit » sa capacité seulement après que celle-ci ait été consommée par une tâche apériodique

- **Exemple (RM-SS, C_s=5; T_s=10):**



Sporadic Server (SS)

- Condition d'ordonnançabilité sous RM-SS :
  - Condition suffisante :

$$Up \leqslant n\left[\left(\frac{2}{Us+1}\right)^{1/n} - 1\right]$$

Sporadic Server (SS)

- **Intérêts** : 
  - Résolution du problème de possibles dépassements d'échéances pour les tâches périodiques sous DS
  - Faibles complexités de calcul et d'implémentation
- **Inconvénients** : 
  - Hypothèses restrictives de l'algorithme RM

Total Bandwidth Server (TBS)

- **Principe** : Lorsque la $k^{ième}$ 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

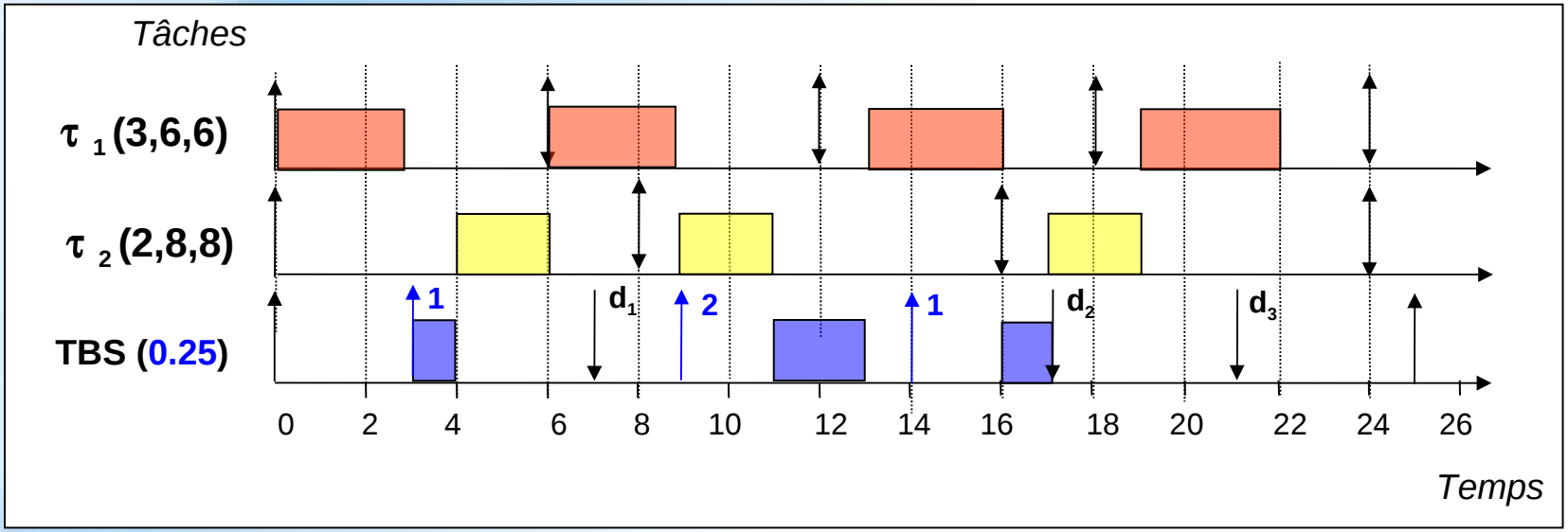
$$d_k = \max(r_k, d_{k-1}) + \frac{C_k^a}{U_s}$$

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)

- Condition d'ordonnançabilité sous EDF-TBS :
  - Condition **nécessaire et suffisante** :

$$U_p + U_s \leq 1$$

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

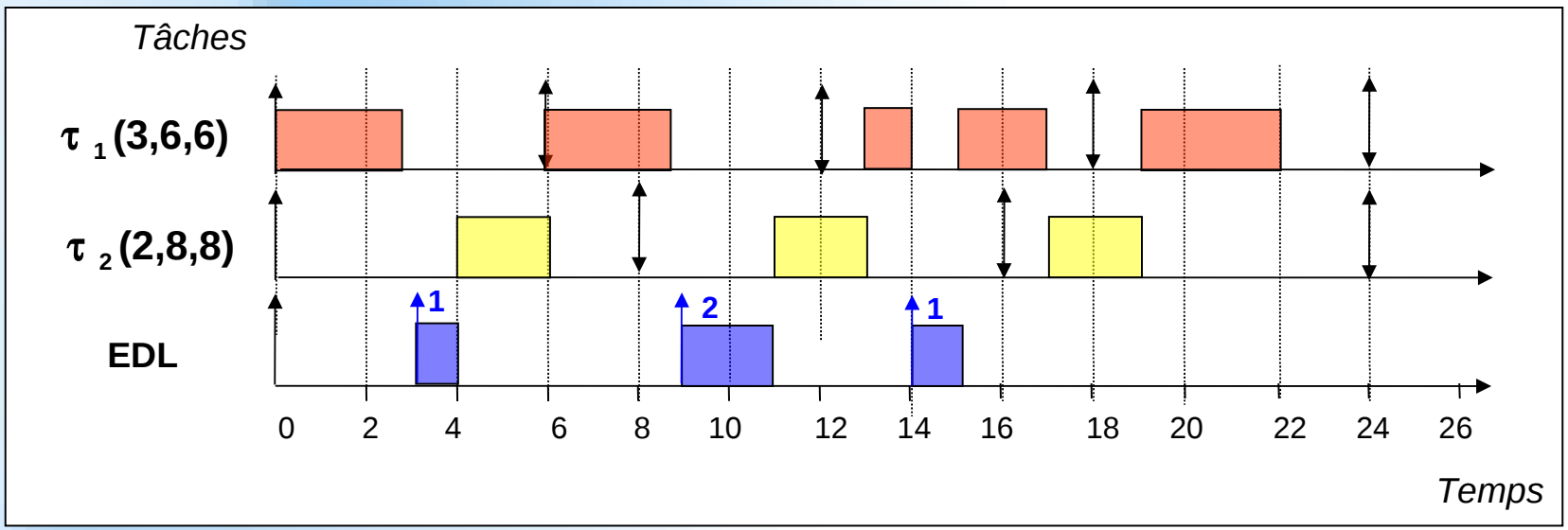
sinon

- exécuter les tâches périodiques au + tôt

fin

Earliest Deadline as Late as possible (EDL)

- **Exemple (EDF-EDL) :**



Earliest Deadline as Late as possible (EDL)

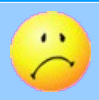
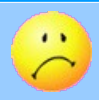
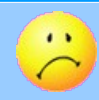
- Condition d'ordonnançabilité sous EDL :
  - Condition **nécessaire et suffisante** :

$$U_p \leq 1$$

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

Synthèse des serveurs de tâches apériodiques

	Performance	Complexité calcul	Besoins mémoire	Complexité implémentation
BG				
PS				
DS				
SS				
TBS				
EDL				

Conclusion sur l'ordonnancement temps réel

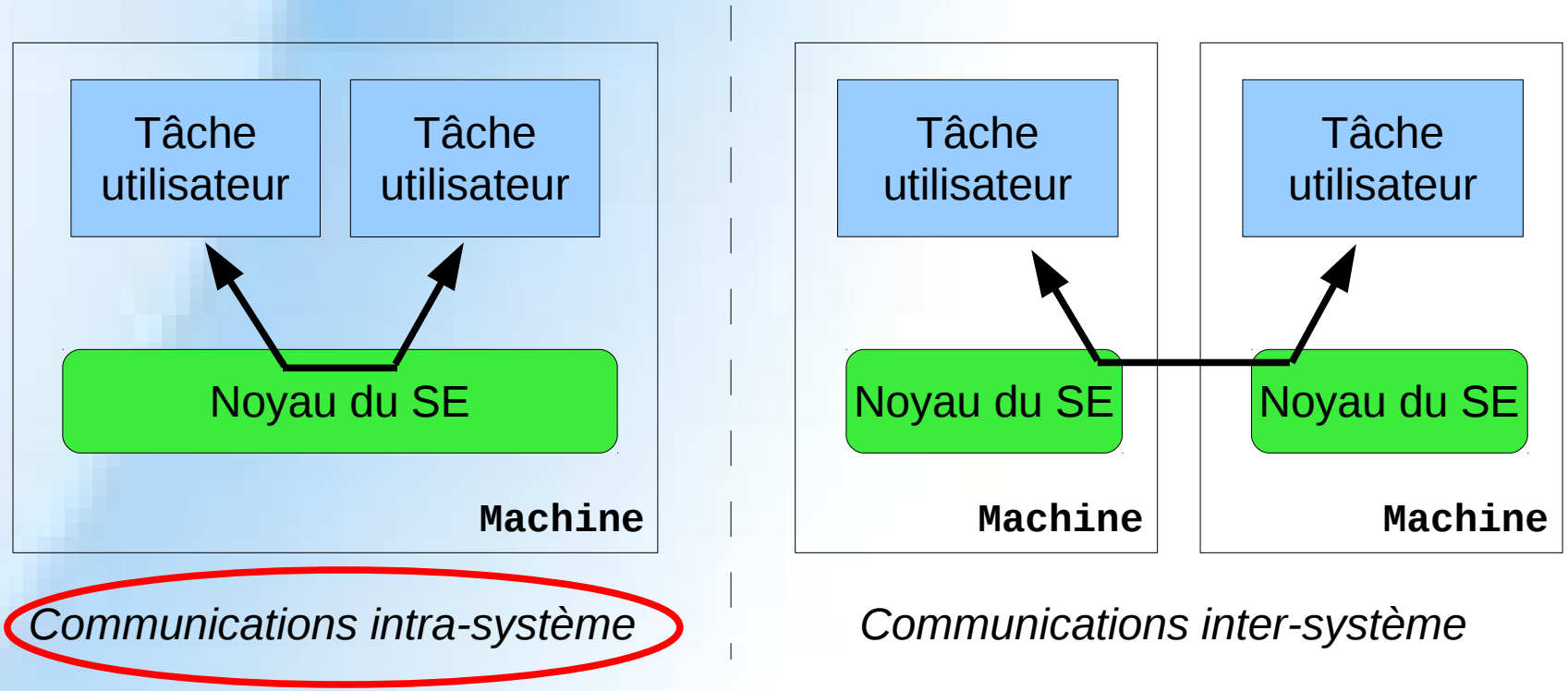
- L'ordonnanceur doit sélectionner et activer les tâches dans une fenêtre de temps fixée (*déterminisme*)
- L'ordonnancement temps-réel
  - est régi par une politique préemptive spécifique basé sur la priorité
  - peut être hors-ligne ou en-ligne
  - doit être déterministe
- Compromis simplicité / performances des algorithmes d'ordonnancement
- Quantification de l'overhead d'ordonnancement
- Limites : affecter des priorités n'est pas toujours une garantie de maîtrise des contraintes temporelles (cas de partage des ressources)

Problématique temps réel n°3

Les protocoles d'accès aux ressources...

Contexte (1)

- Tâches **concurrentes** vs. Tâches **distants**



Contexte (2)

- Plusieurs tâches → **accès concurrents** aux ressources

- Une **ressource** désigne toute entité dont a besoin une tâche pour s'exécuter :
  - ressource **matérielle** (processeur, périphérique, etc.)
  - ressource **logicielle** (variable)

- 3 phases pour l'exploitation d'une ressource par une tâche :
  - sollicitation de la ressource
  - utilisation de la ressource
  - libération de la ressource

Contexte (3)

- Une ressource est dite **critique** lorsque des accès concurrents à cette ressource peuvent mener à un état incohérent

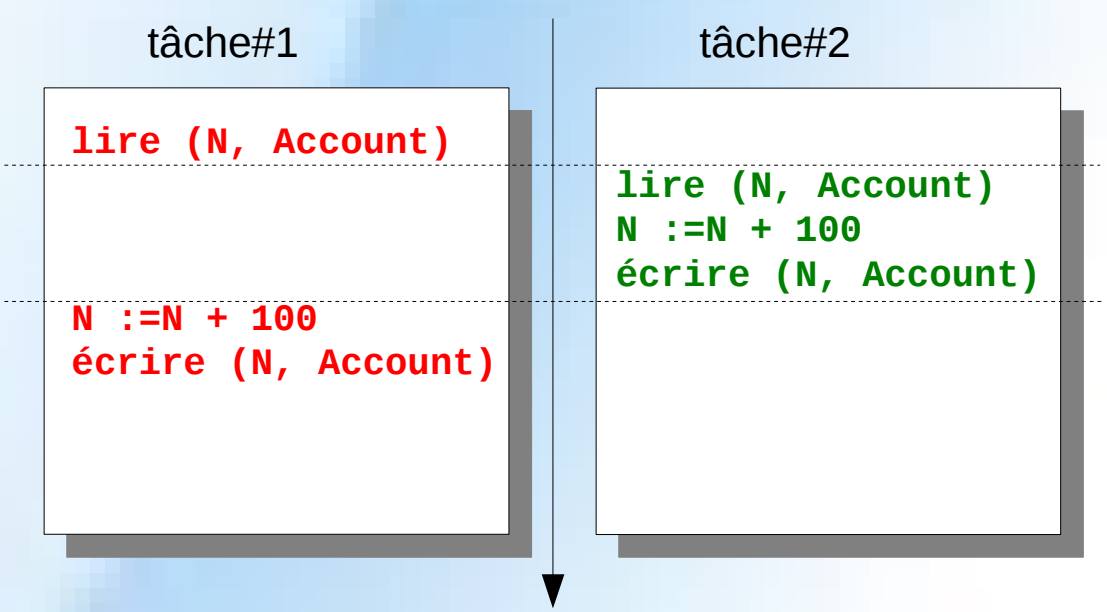
- On parle aussi de **situation de compétition** (*race condition*) pour décrire une situation dont l'issue dépend de l'ordre dans lequel les opérations sont effectuées

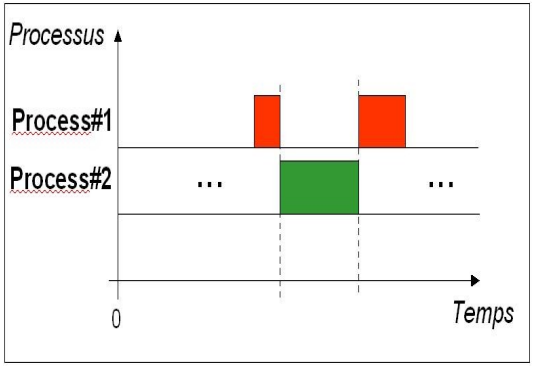
- Une **section critique** est une section de programme manipulant une ressource critique

- Un mécanisme d'**exclusion mutuelle** sert à assurer l'atomicité des sections critiques relatives à une ressource critique

Accès aux ressources critiques

- Cas d'**incohérence de données** : problème de la synchronisation relative de l'exécution des tâches

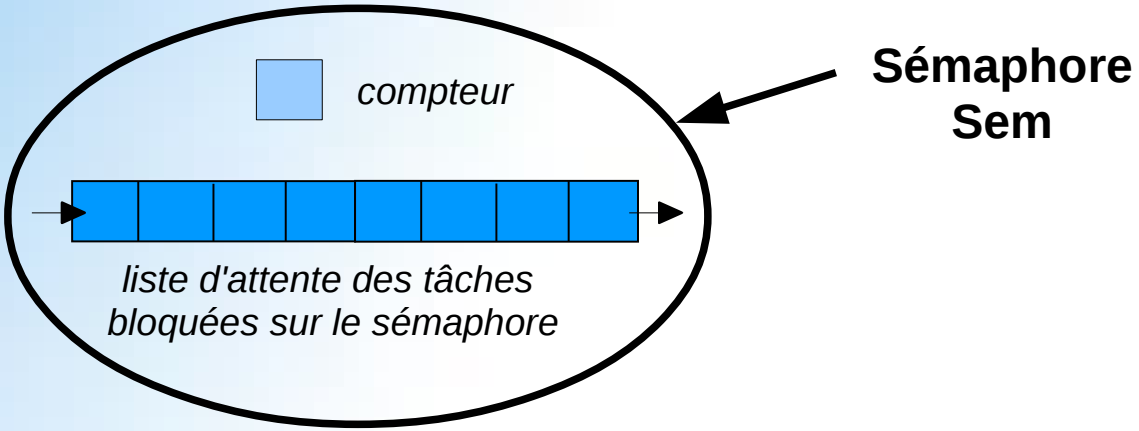




- tâche#2 ne doit pas accéder à N tant que tâche#1 l'utilise !

Principe des sémaphores : rappels (1)

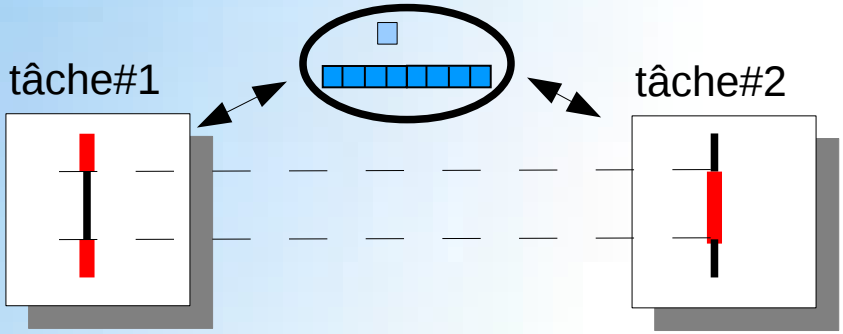
- Un sémaphore est une **structure de données**
  - contenant un compteur (valeur entière non négative)
  - gérant une file d'attente de tâches attendant qu'advienne une condition particulière propre au sémaphore



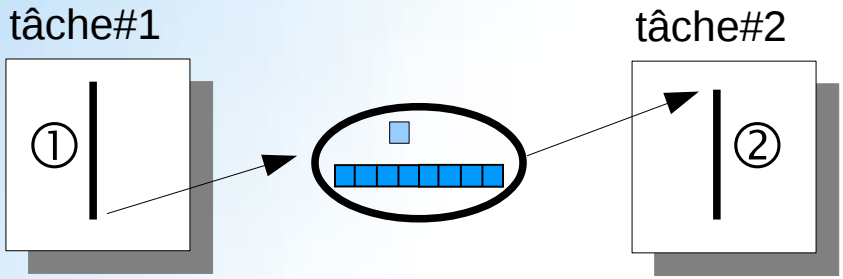
Principe des sémaphores : rappels (2)

- Types de synchronisation possibles :

- **Exclusion mutuelle**



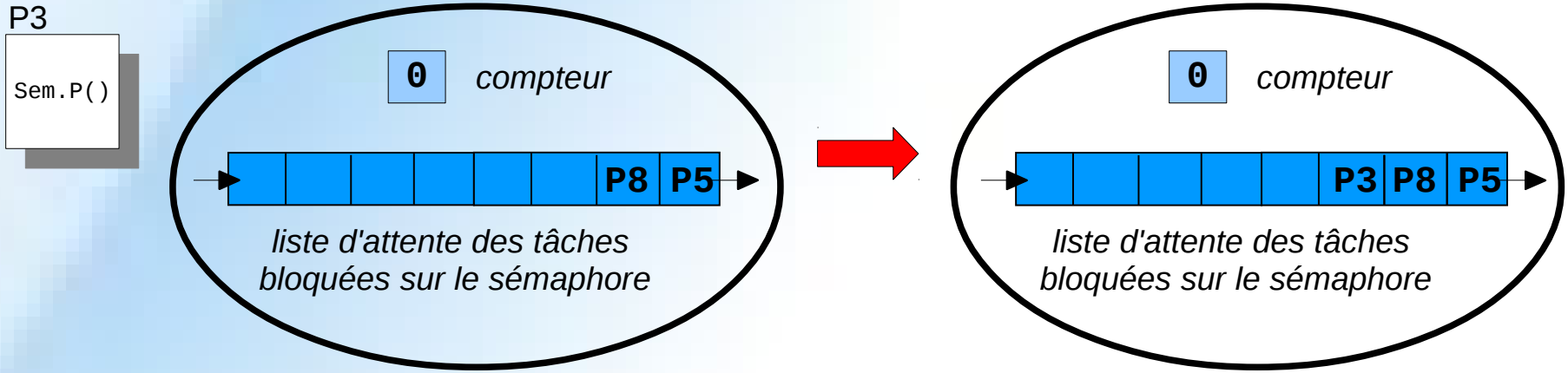
- **Barrière de synchronisation**



Principe des sémaphores : rappels (3)

- 1ère opération : **test de prise** du sémaphore (« Puis-je ?)

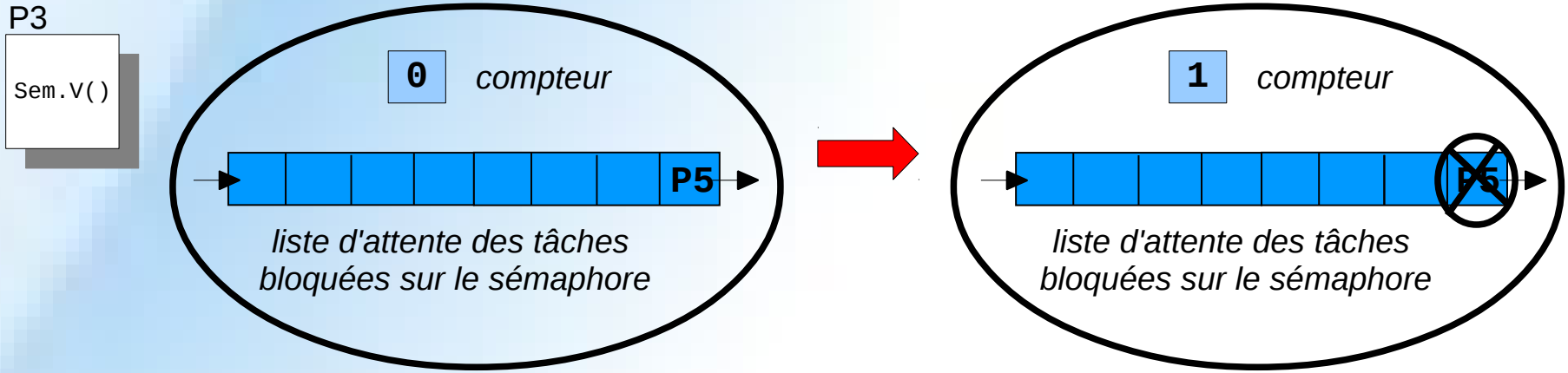
```
Sem.P() :     si (Sem.compteur>0)
                    alors Sem.compteur = Sem.compteur-1
                    sinon insère_ce_processus(Sem.file)
              finsi
```



Principe des sémaphores : rappels (4)

- 2ème opération : **libération** du sémaphore (« Vas-y ! »)

```
Sem.V() : Sem.compteur = Sem.compteur+1
          si (Sem.compteur > 0)
             alors extrait_un_processus(Sem.file)
          finsi
```



Principe des sémaphores : rappels (5)

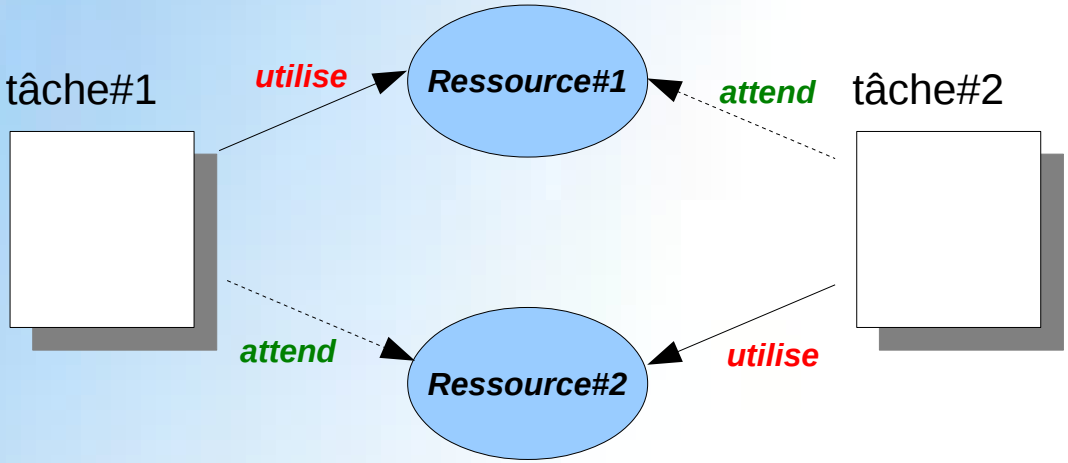
- Types de sémaphores :

  - **Sémaphore binaire** : implémentation de l'exclusion mutuelle
    - le compteur ne peut admettre que les valeurs 0 ou 1
    - le compteur doit être initialisé à 1

  - **Sémaphore compteur** : implémentation de la barrière de synchronisation
    - Le compteur doit être initialisé à 0

Problèmes liés à l'utilisation de sémaphores (1)

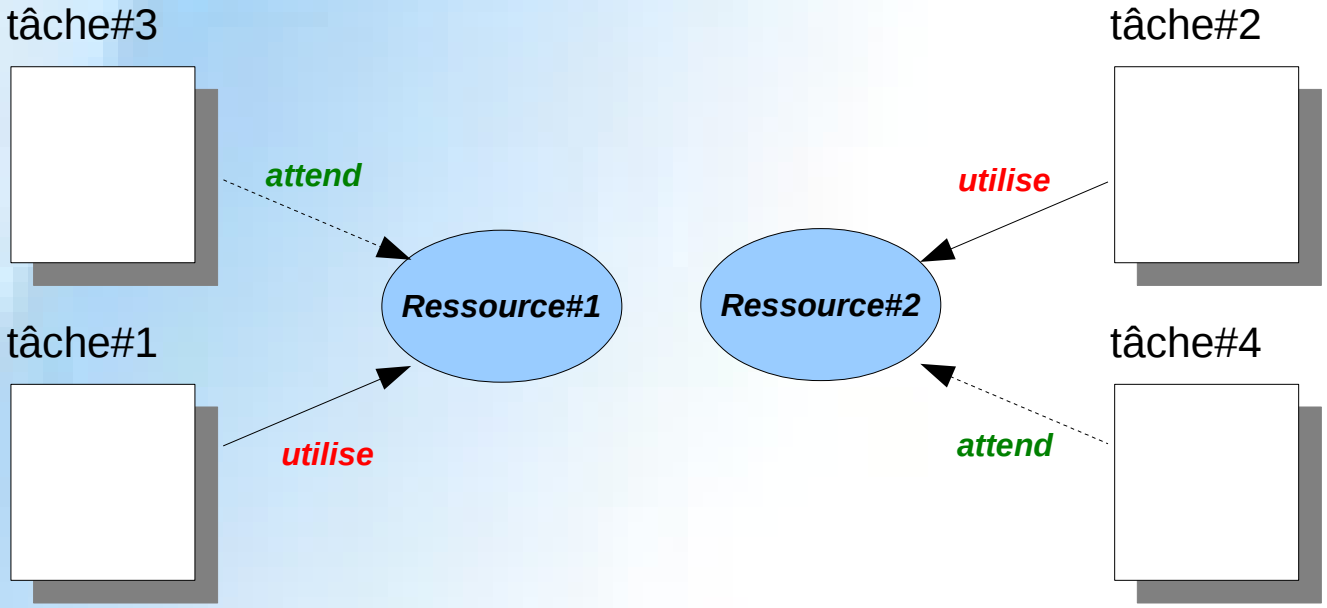
- Cas d'**interblocage (*deadlock*)** : ensemble de tâches attendant chacune une ressource déjà possédée par une tâche de l'ensemble



- L'attente est **infinie**

Problèmes liés à l'utilisation de sémaphores (2)

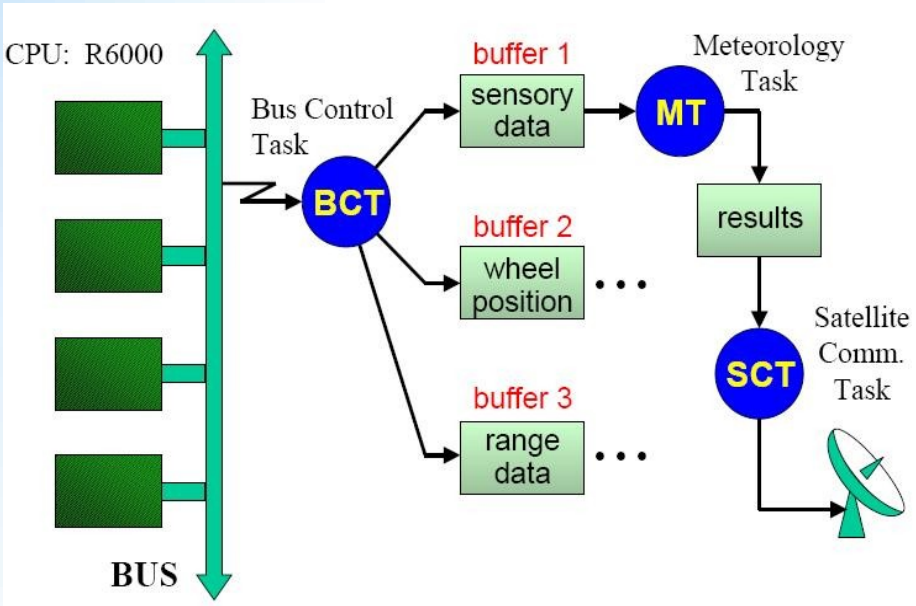
- Cas de **coalition et famine (*livelock*)** : ensemble de tâche monopolisant des ressources au détriment d'autres tâches.



- L'attente est **indéfinie**

Problèmes liés à l'utilisation de sémaphores (3)

- **Un exemple réel** : la mission Mars Pathfinder, 4 juillet 1997
  - Système ordonnancé par RM
  - RTOS VxWorks
  - **Ressources partagées** :
    le bus du système
    ...
  - **Tâches du système** :
    tâche de gestion du bus
    tâche de communication avec le sol
    ...
    tâche de recueil d'informations météorologiques



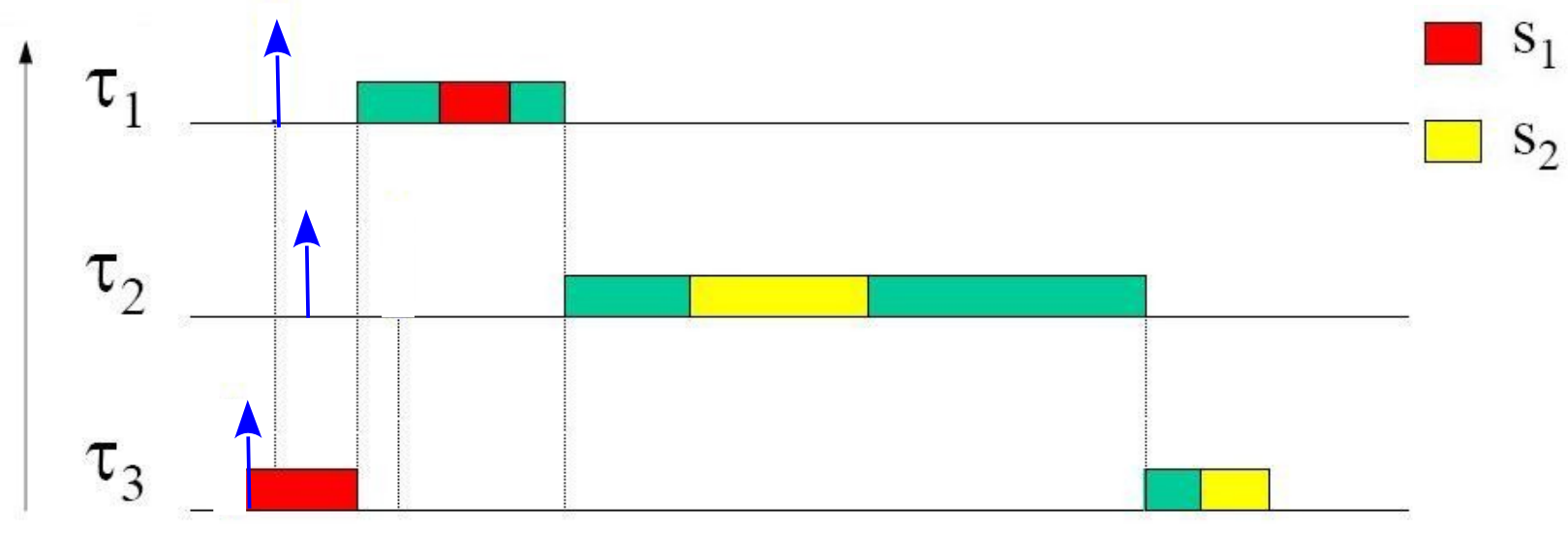
Pb d'inversion de priorités !

Protocoles de résolution des inversions de priorités

- Pour les algorithmes d'ordonnancement **à priorités fixes** :
  - Non Preemptive Protocol (NPP)
  - Highest Locker Priority (HLP)
  - Priority Inheritance Protocol (PIP)
  - Priority Ceiling Protocol (PCP)
- Pour les algorithmes d'ordonnancement **à priorités dynamiques** :
  - Non Preemptive Protocol (NPP)
  - Dynamic Priority Inheritance Protocol (D-PIP)
  - Dynamic Priority Ceiling Protocol (D-PCP)
- Pour les 2 types :
  - Stack Resource Policy (SRP)

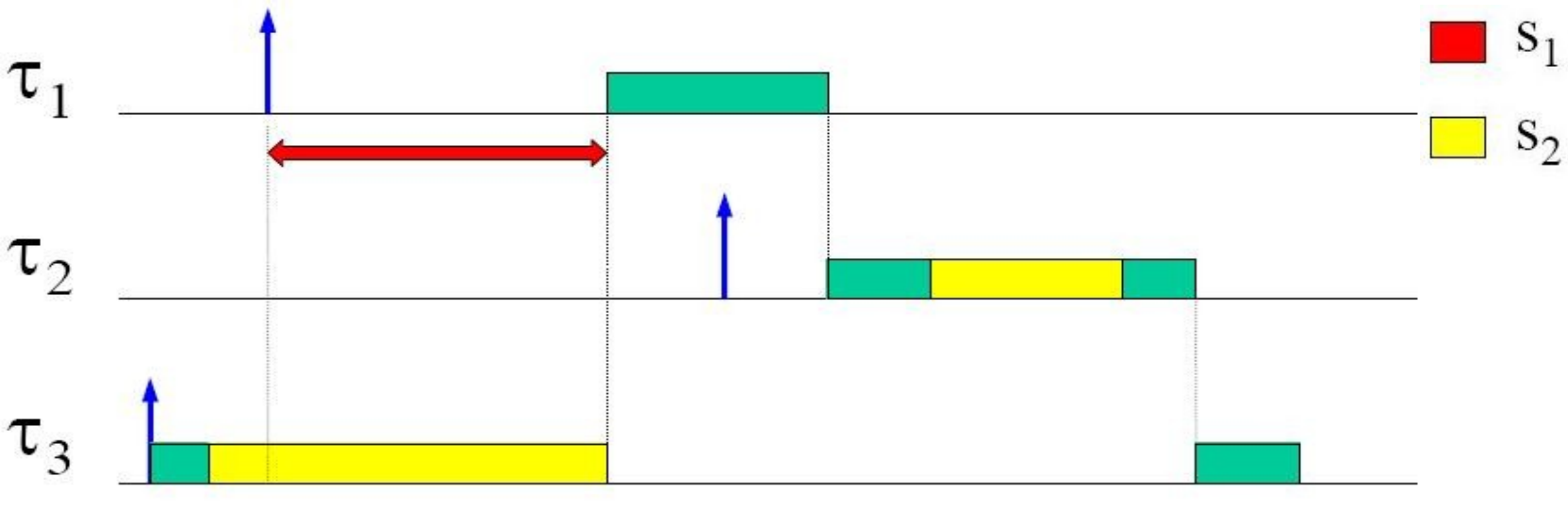
Non Preemptive Protocol (NPP)

- **Principe** : une tâche en section critique obtient la plus haute priorité parmi **toutes** les tâches



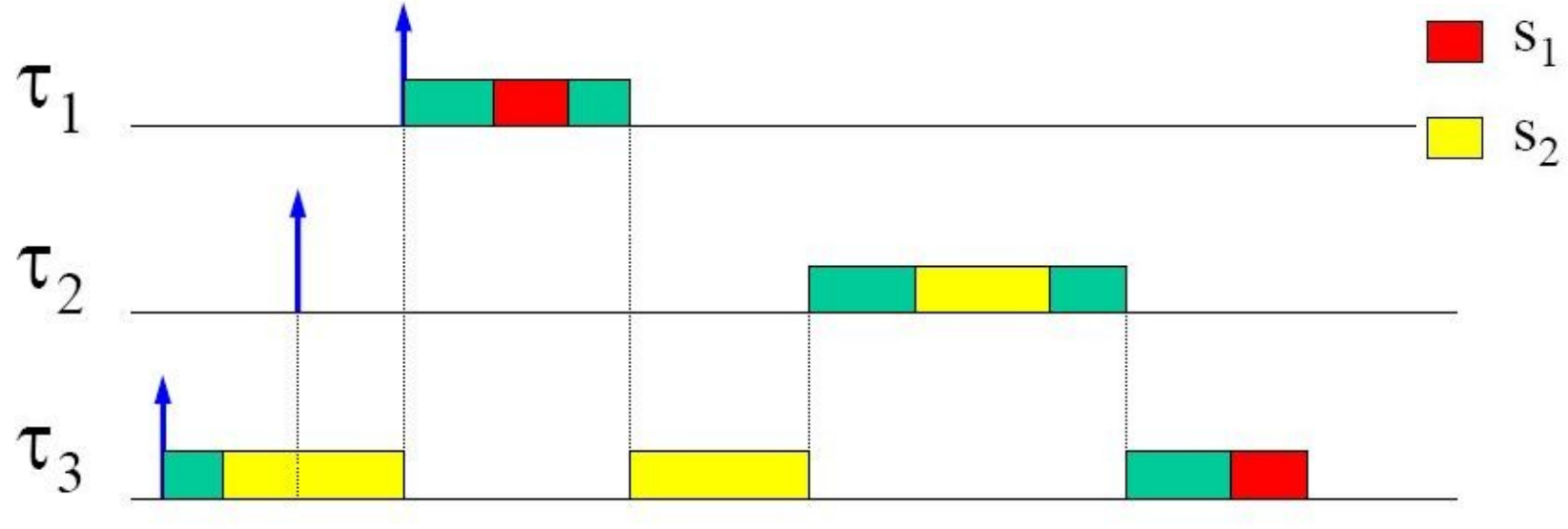
Non Preemptive Protocol (NPP)

- **Problème** : blocages superflus



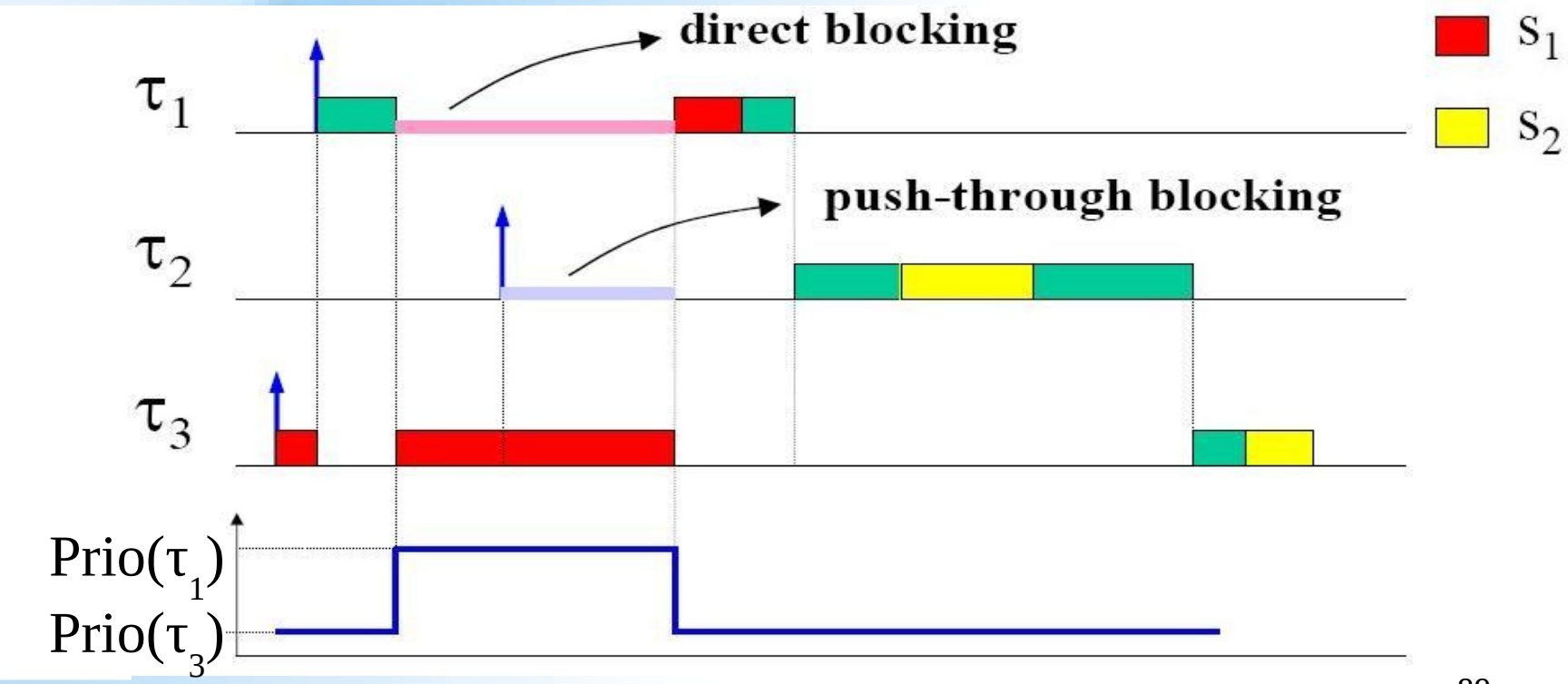
Highest Locker Priority (HLP)

- **Principe** : une tâche en section critique obtient la plus haute priorité parmi les tâches utilisant le **même** sémaphore



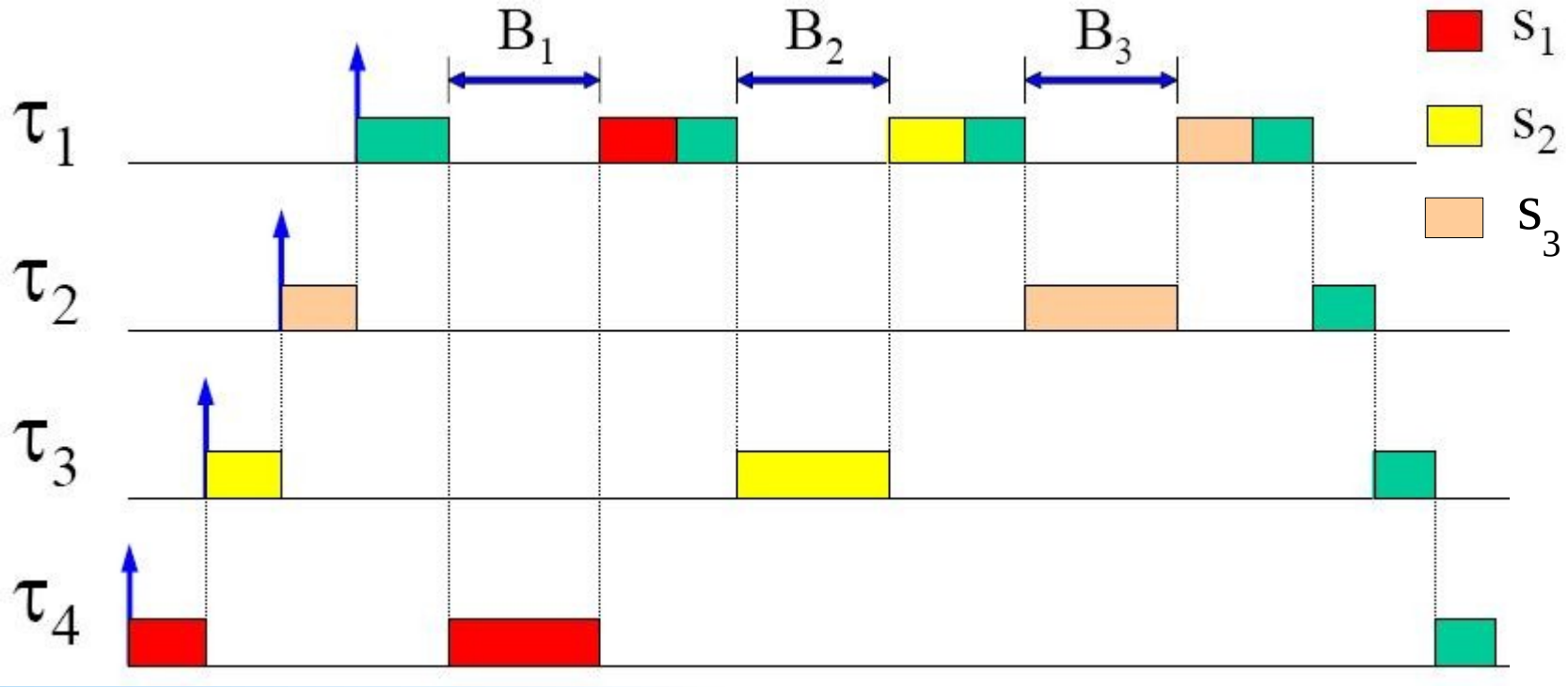
Priority Inheritance Protocol (PIP)

- **Principe** : une tâche en section critique augmente sa priorité **uniquement lorsqu'elle bloque d'autres tâches** (la priorité héritée correspond à la plus haute priorité parmi les tâches bloquées)



Priority Inheritance Protocol (PIP)

- **Problème** : blocages en chaîne



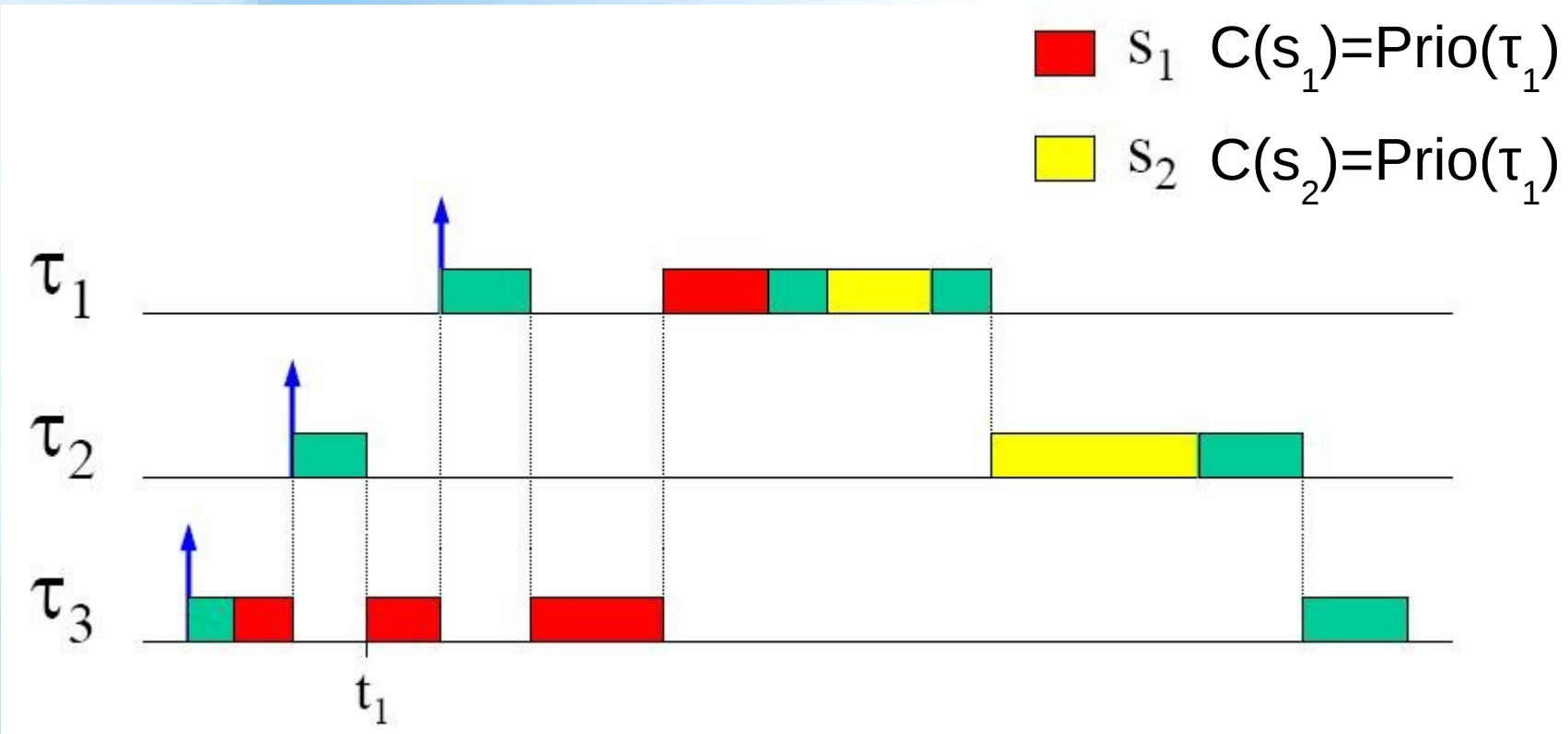
Priority Ceiling Protocol (PCP)

- **Intérêts** : 
  - Le temps de blocage est réduit à une seule section critique
  - PCP prévient les situations d'interblocages (deadlocks)

- **Inconvénient** : 
  - Utilisation non transparente pour l'utilisateur : allocation de valeurs plafonds

Priority Ceiling Protocol (PCP)

- **Principe** : une tâche peut entrer en section critique uniquement si celle-ci est libre et qu'il n'y a pas de risque de blocage en chaîne



Stack ressource Policy (SRP)

- SRP satisfait les **mêmes propriétés** que PCP :
  - inversion de priorités non bornée
  - interblocages et blocages en chaîne supprimés
  - chaque tâche est bloqué au plus une fois

et **en plus** :

  - utilisation avec des algorithmes à priorités fixes ou dynamiques, sans adaptations préalables
  - une tâche ne se bloquera jamais une fois son exécution démarrée

- **Principe** : une tâche se bloque au moment où elle tente de préempter une autre tâche (et non plus au moment où elle tente d'accéder à une ressource)

Garanties d'ordonnançabilité en présence de ressources partagées ?

- **Méthodologie :**
  - Sélection d'un algorithme d'ordonnancement
  - Sélection d'un protocole d'accès aux ressources
  - Calcul du temps de blocage maximum B_i de chaque tâche
  - Test de l'ordonnançabilité de l'ensemble des tâches en incluant les temps de blocage des tâches

Calcul du temps de blocage B_i des tâches sous PIP

- **Définition**

Le temps de blocage maximum pour chaque tâche T_i peut être déterminé de la manière suivante :

$$B_i = \min (B^l_i, B^s_i)$$

avec :

$$B^l_i = \sum_{j=i+1}^{n} \max_k [D_{j,k} : C(S_k) \geq P_i]$$

$$B^s_i = \sum_{k=1}^{m} \max_{j>i} [D_{j,k} : C(S_k) \geq P_i]$$

n *: le nb de tâches ordonnées par priorités décroissantes*

m *: le nb de ressources*

Calcul du temps de blocage B_i des tâches sous PIP

- **Méthodologie :**

1) Pour chaque tâche T_j de plus faible priorité que T_i, identifier l'ensemble $\beta_{i,j}$ de toutes les sections critiques qui peuvent bloquer T_i

2) Pour chaque sémaphore S_k, identifier l'ensemble $\gamma_{i,k}$ de toutes les sections critiques gardées par S_k qui peuvent bloquer T_i

3) Effectuer la somme B^l_i des plus longues sections critiques dans chaque $\beta_{i,j}$, pour toute tâche de priorité plus faible que T_i

4) Effectuer la somme B^s_i des plus longues sections critiques dans chaque $\gamma_{i,k}$, pour tout sémaphore S_k

5) Calculer B_i comme le minimum entre B^l_i et B^s_i

Exemple de calcul de B_i sous PIP

	s_1	s_2	s_3
T_1	1	2	0
T_2	0	9	3
T_3	8	7	0
T_4	6	5	4

B_1, B_2, B_3 et B_4 ?

Calcul du temps de blocage B_i des tâches sous PIP

- **Complexité :**
  - $O(mn^2)$
- **Limites du calcul**
  - Ne représente qu'une borne supérieure sur les temps de blocage
  - Le calcul n'exclut pas les cas suivants (impossibles par hypothèse) :
    - 2 tâches peuvent utiliser une même ressource *simultanément*
    - Une tâche peut utiliser 2 ressources *simultanément*

Calcul du temps de blocage B_i des tâches sous PCP

- **Définition**

B_i correspond à la plus longue section critique parmi celles :

- appartenant aux tâches de plus faible priorité que T_i et,

- gardées par un sémaphore de valeur plafond supérieure ou égale à Prio (T_i)

Exemple de calcul de B_i sous PCP

	$C(s_1)=Prio(\tau_1)$	$C(s_2)=Prio(\tau_1)$	$C(s_3)=Prio(\tau_2)$
T_1	1	2	0
T_2	0	9	3
T_3	8	7	0
T_4	6	5	4

B_1, B_2, B_3 et B_4 ?

Garanties d'ordonnançabilité en présence de ressources partagées ?

- **Méthodologie :**
  - Sélection d'un algorithme d'ordonnancement
  - Sélection d'un protocole d'accès aux ressources
  - Calcul du temps de blocage maximum B_i de chaque tâche
  - Test de l'ordonnançabilité de l'ensemble des tâches en incluant les temps de blocage des tâches

Tests d'ordonnançabilité avec temps de blocage

- **Sous RM avec PIP ou PCP :**

Condition suffisante : $$U = \sum_{k=1}^{i} \frac{C_k}{P_k} + \frac{B_i}{P_i} \leq i\,(2^{1/i} - 1)$$

- **Exemple :**

	C_i	P_i	B_i
T_1	1	2	1
T_2	1	4	1
T_3	2	8	0

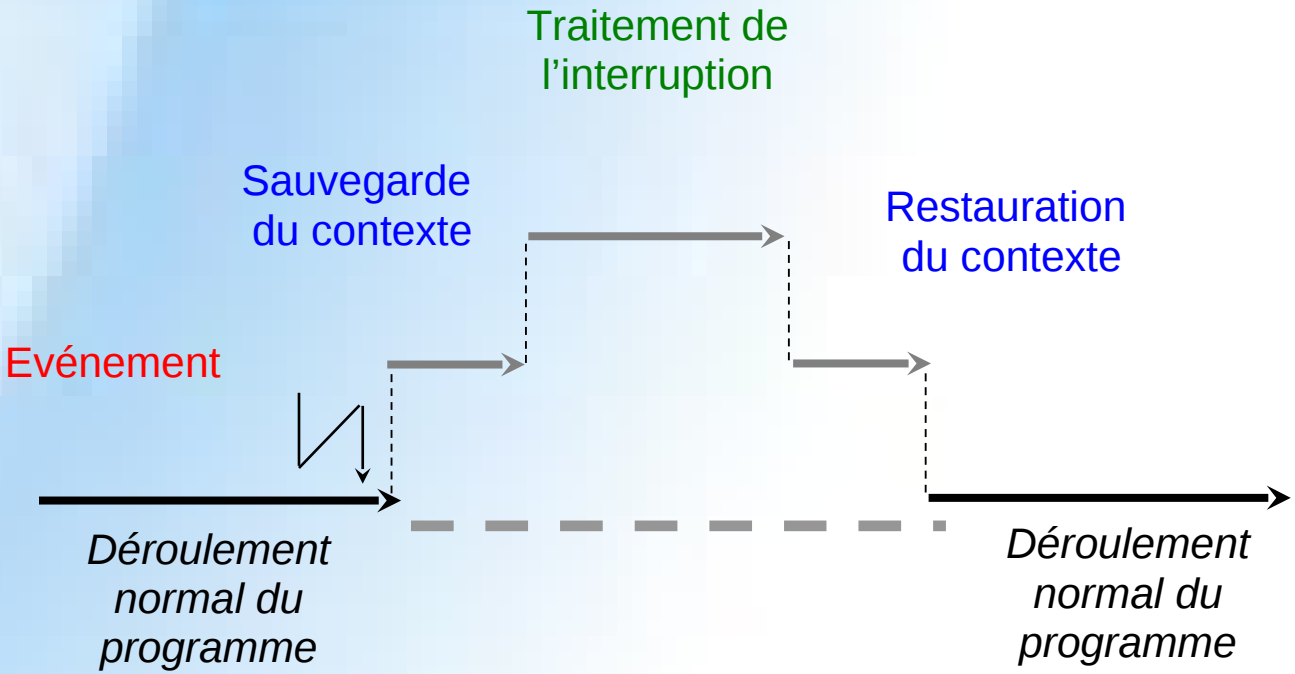
Système ordonnançable ?

Problématique temps réel n°4

La gestion des interruptions...

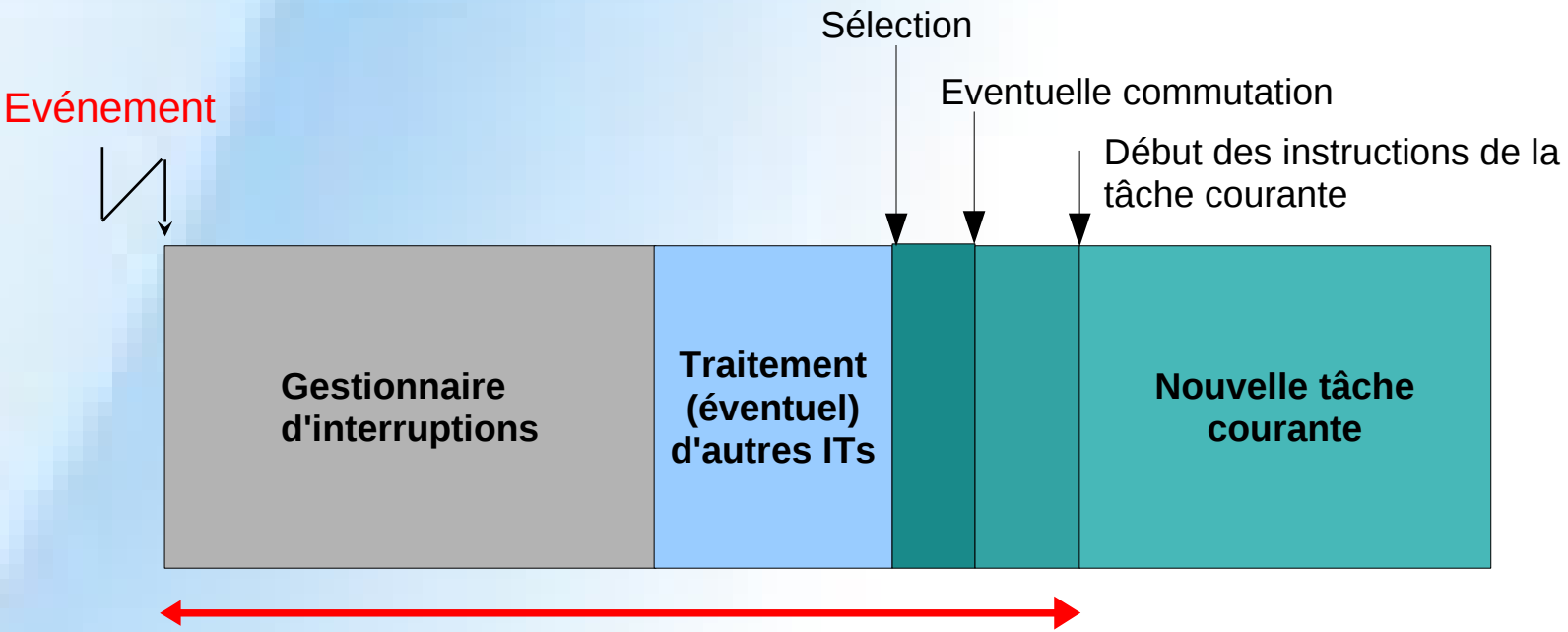
Le service d'interruption (1)

- **Délai dû au gestionnaire d'interruptions :**



Le service d'interruption (2)

- **Délai dû au système d'exploitation :**



Le service d'interruption (3)

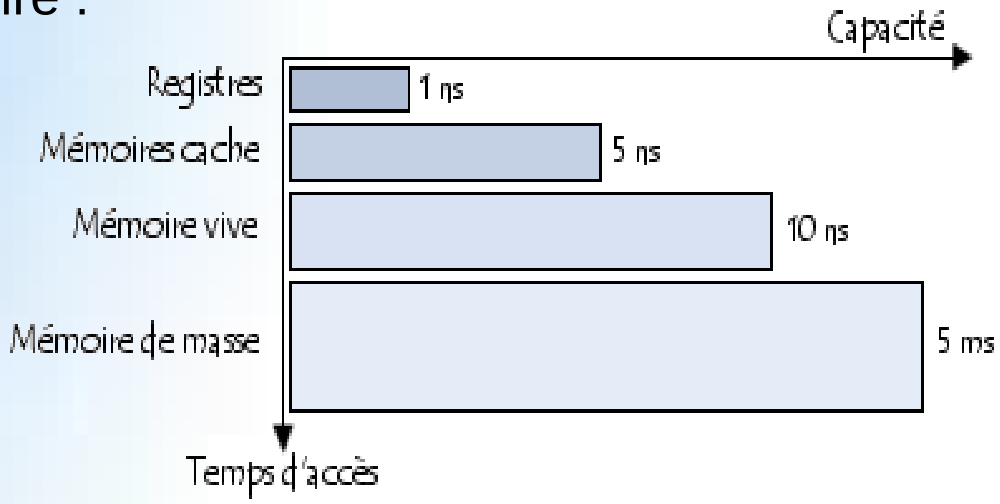
- Les OS classiques
  - ils **n'offrent pas** de garanties sur le temps de latence aux interruptions
- Les OS temps réel
  - ils offrent des niveaux de priorités suffisants pour répartir les degrés d'urgence des interruptions
  - le gestionnaire d'interruptions de l'OS est réduit au strict minimum

Problématique temps réel n°5

La gestion de la mémoire...

Les concepts de base

- 2 niveaux de gestion de la mémoire :



  - **Niveau matériel**

    les registres du processeur

    la mémoire cache

  - **Niveau système d'exploitation**

    la mémoire principale (appelée également *mémoire centrale ou interne*)

    la mémoire secondaire (appelée également *mémoire de masse ou physique*)

Rôle du gestionnaire de mémoire du SE

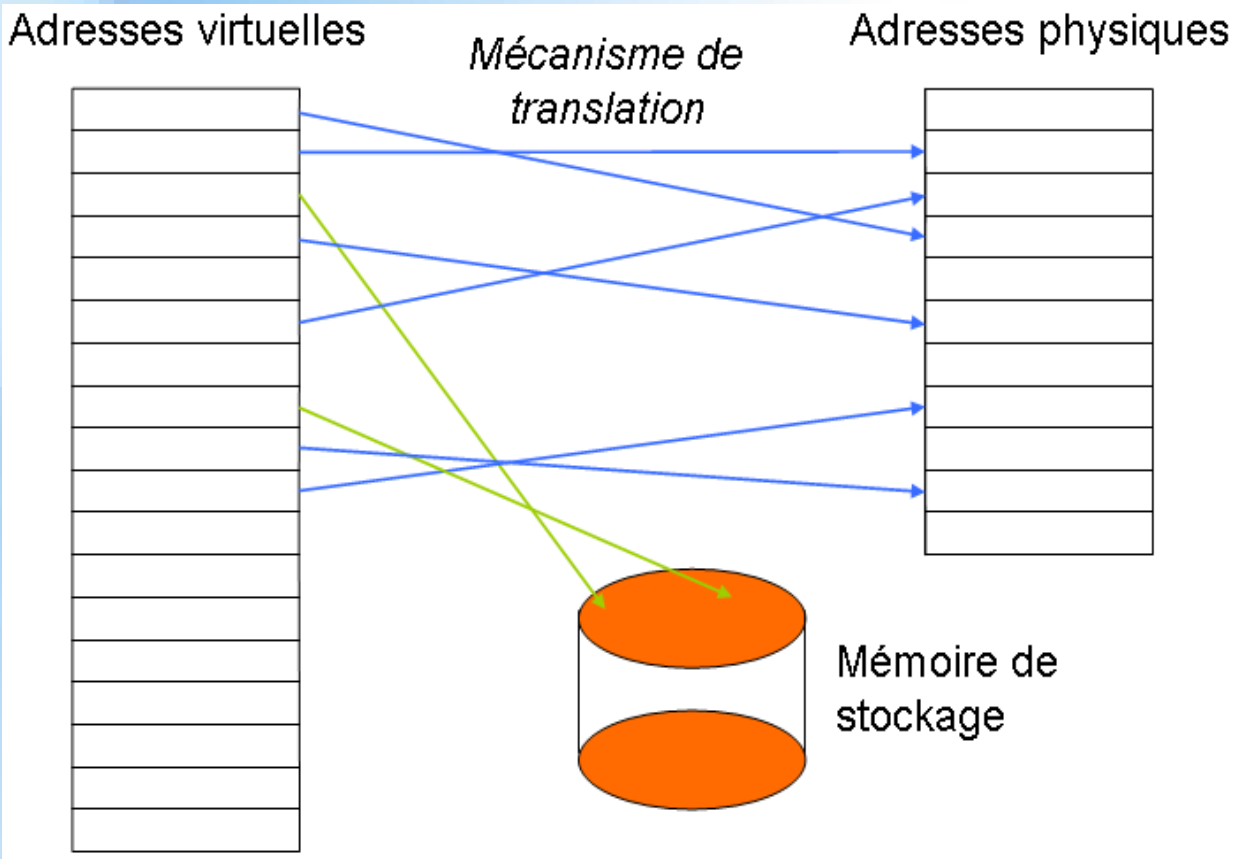
Le gestionnaire de mémoire du SE

- **Objectifs du gestionnaire de mémoire** du système d'exploitation :
  - Partager la mémoire
  - Allouer des blocs de mémoire aux différents processus
  - **Protéger les espaces mémoire utilisés**
  - Optimiser la quantité de mémoire disponible

Mécanisme de mémoire virtuelle + Mécanismes de découpage de la mémoire

+ rapidité et déterminisme pour le temps réel

La mémoire virtuelle

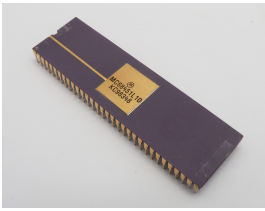


- Problème du swapping de pages appartenant à des tâches temps réel

Le verrouillage des pages

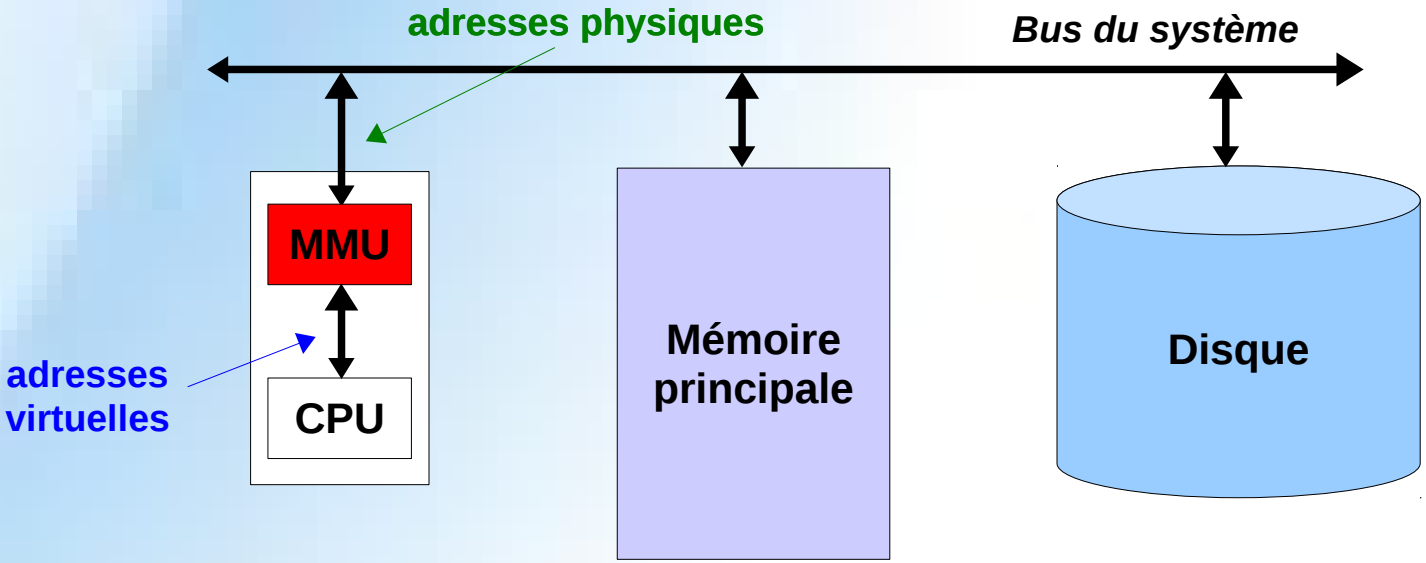
- La **table des pages virtuelles** (TPV) est composée de plusieurs champs :
  - le numéro de cadre
  - un bit de présence
  - un bit de modification (M)
  - un bit de référence (R)
  - un bit de protection

Implémentation du verrouillage : la MMU



- La traduction des adresses virtuelles en adresses physiques, ainsi que le verrouillage des pages est assurée par un circuit matériel spécifique pour la gestion de la mémoire :

la MMU (*Memory Management Unit*)



Les différentes approches pour le temps réel

- Systèmes d'exploitation temps réel à **mémoire virtuelle**
  - Présence d'une MMU
  - Verrouillage des pages des tâches temps réel dans la RAM physique
  - Suppression de l'overhead de pagination
- Systèmes d'exploitation temps réel à **mémoire linéaire**
  - Pas de MMU
  - Déterminisme temporel
  - Développement multitâche difficile (pas de protection mémoire)