# Ordonnancement temps réel

*Pierre-Yves Duval (cppm)*





Ecole d'informatique temps réel - La Londes les Maures 7-11 Octobre 2002

# Problématique

Satisfaire les contraintes de temps des transactions temps réel en allouant de façon efficace:

-ressources CPU
-ressources de communication
-un accès aux données

L’ordonnancement concerne les méthodes d’allocations de ces ressources.

# Classification des algorithmes d'ordonnancement

-A contraintes strictes ou souples

-Respect des contraintes garanties à 100% ou tolérance

-Statiques ou dynamiques

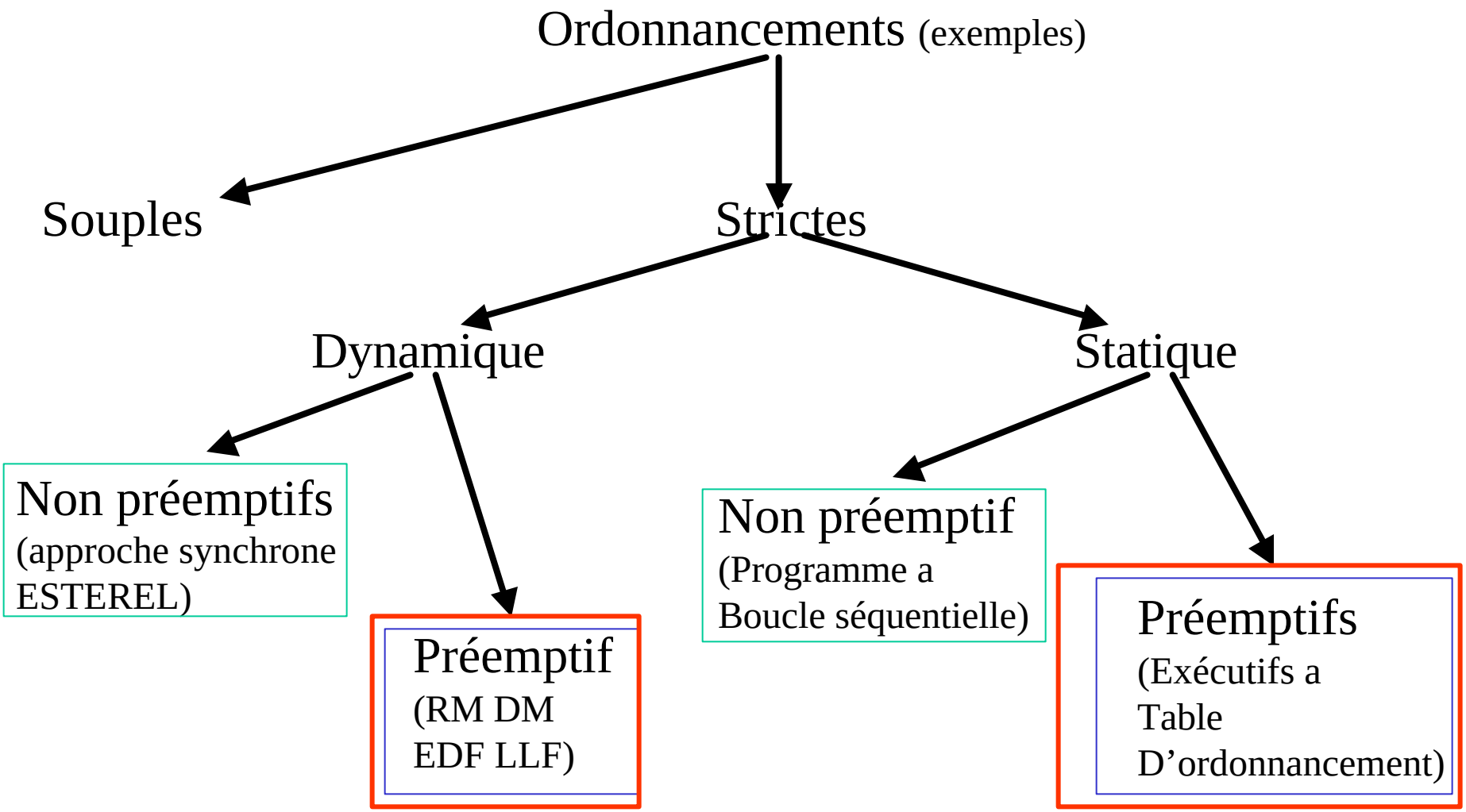
-Ordonnancement fixe ou réévaluation en ligne

-Préemptifs ou non préemptifs

-Tâches pouvant être interrompues avant terme ou pas

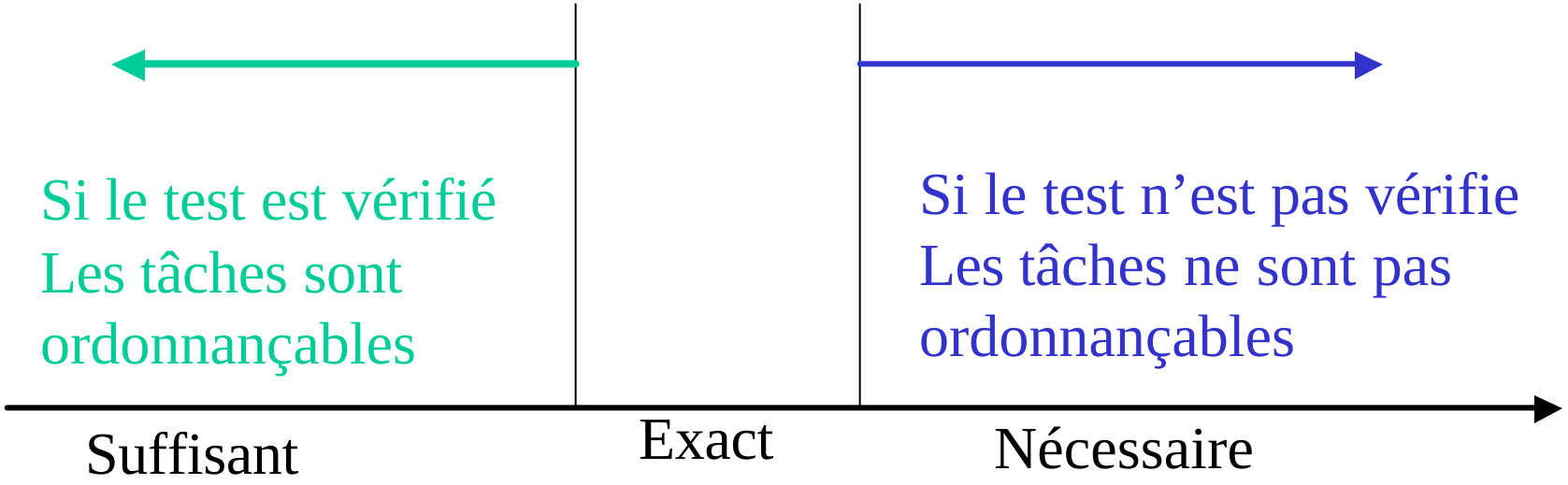
-Centralisés ou répartis

# Types d'ordonnancement



# Test d'ordonnancement

Déterminer si il existe un algorithme qui satisfasse les contraintes temporelles d'un système.



# Test d'ordonnancement de tâches périodiques

Un test nécessaire pour qu'un ensemble de tâches {Ti} périodiques soit ordonnançable consiste à vérifier que:

$$m = \sum c_i/t_i <= n$$

avec m = ci/ti taux d'occupation du processeur par le processus i,
ci temps de calcul nécessaire à la tâche i,
ti période de la tâche i,
n nombre de processeurs disponibles.

# Ordonnancement dynamique

# Ordonnancement dynamique

Ordonnanceur en ligne qui réévalue à chaque nouvelle demande la tâche qui doit être exécutée

Dans les modèles ci-après on fera l'hypothèse que le temps pris par l'ordonnanceur et les temps de changement de tâche sont nuls.

L'implantation de ces ordonnanceurs doit donc être très efficace.

# RM Tâches indépendantes périodiques

**Rate Monotonic** [Liu73] (optimal pour 1 CPU)
Dynamique et préemptif basé sur des priorités fixes

Hypothèses:
-Tâches périodiques (tâche i période Ti)
(mais pas obligées de s'exécuter à chaque période)
-Echéance en fin de période (pas obligatoire)
-Tâches préemtibles et indépendantes
-Tâches ne peuvent se suspendre ou se bloquer
-Le temps d'exécution $C_i$ connu à priori et constant
(peut ne pas être constant, on prend le maxi)

# RM Tâches indépendantes périodiques

**Rate Monotonic RM**

Parti statique:

- Classement des tâches par périodes croissantes
- Allocation de la priorité en fonction inverse de la période
- Une tâche et une seule par priorité

Parti dynamique:

- A chaque demande de CPU allocation à la tâche la plus prioritaire

# RM Tâches indépendantes périodiques

**Rate Monotonic RM**

Condition nécessaire

$C_i$ temps calcul, $T_i$ période

$$\sum_{i=1}^{n}\left(C_i/T_i\right)<=1$$

Condition suffisante

$$\sum_{i=1}^{n}\left(C_i/T_i\right)<=n(2^{1/n}-1)$$

Evaluation: quand n→infini tend vers Ln2 0.7 (70% occupation)

Cas particulier: toutes périodes multiples de la plus petite on peut Atteindre 100% occupation

# RM Tâches indépendantes périodiques

**Rate Monotonic RM**

Analyse exacte (Joseph et Pandya [Jos86])

Calcul itératif du temps de réponse en cas pire

$$R_i = C_i + \sum_{\forall j \in hp(i)} \left\lceil \left( R_i / T_j \right) \right\rceil C_j$$

hp(i) tâches plus prioritaires que i

$R_i$ obtenu par calcul itératif

$$R^{n+1}_i = C_i + \sum_{\forall j \in hp(i)} \left\lceil \left( R^n_i / T_j \right) \right\rceil C_j$$

# DM Tâches indépendantes périodiques

## **Deadline Monotonic DM** **(mêmes hypothèses que RM)**

Même principe que RM mais on affecte les priorités aux tâches selon l'ordre inverse de leurs échéances

**Intérêt** : RM pénalise les tâches rares mais urgentes. Cet algorithme est meilleur pour les tâches dont l'échéance est très inférieure à la période

La condition suffisante d'ordonnançabilité devient:

$$\sum_{i=1}^{n}(C_i/D_i) <= n(2^{1/n} - 1)$$

# RM/DM Tâches indépendantes périodiques

## Evaluations de RM et DM

Pour:

- mécanisme simple
- s'implante naturellement avec les OS du marché
- peut-être utilisé pour affecter des priorités d’interruptions

Contre:

- hypothèses restrictives
- uniquement avec préemption
- indépendance impérative

# EDF Tâches indépendantes périodiques

**Earliest Deadline first EDF**[Der74] (optimal pour 1 CPU)
Dynamique et préemptif base sur des priorités dynamiques

**Hypothèses:**
-échéances quelconques mais connues à l'activation
-Tâche préemtibles et indépendantes
-Tâche ne peuvent se suspendre ou se bloquer
-Le pis temps d'exécution $C_i$ connu à priori et constant
(peut ne pas être constant on prend le maxi)

**Algorithme:** (totalement dynamique)
A chaque instant la tâche qui s'exécute est celle qui a l'échéance la plus proche

# EDF Tâches indépendantes périodiques

## Earliest Deadline First EDF

**Si T=D** (période = échéance) **Test exact**

$$\sum_{i=1}^{n} \left( {}^{Ci}/_{Pi} \right) <= 1$$

Ci temps calcul, Pi période

On peut donc utiliser le CPU à 100%

**Si D quelconque** **Test suffisant**

$$\sum_{i=1}^{n} \left( {}^{Ci}/_{Di} \right) <= 1$$

Ci temps calcul, Di échéance

# EDF Tâches indépendantes périodiques

**Evaluation de EDF:**

Pour:

- simple
- optimise l'usage des ressources
- bien adapté aux tâches périodiques et sporadiques à courtes échéances

Contre:

- uniquement avec préemption
- indépendance impérative
- mauvais comportement en cas de surcharge
- pas implantable facilement dans les OS actuels (commence à apparaître dans les nouveaux OS)

# LLF Tâches indépendantes périodiques

**Least Laxity First LLF**

Dynamique et préemptif base sur des priorités dynamiques

Même hypothèses qu'EDF
Optimal sur 1 CPU et meilleur qu'EDF en multiprocesseur

Difficile à implanter car nécessite de maintenir à jour le temps de calcul déjà consommé par chaque tâche

**Algorithme:** (totalement dynamique)
A chaque instant la tâche qui s'exécute est celle qui à la marge la plus Courte
Marge = Echéance – Temps de calcul restant

# Gestion des tâches apériodiques

## Prise en compte de tâches apériodiques

**(dans un contexte de tâches périodiques)**

**But:**
Accepter certaines tâches apériodiques à contraintes souples dans un contexte de tâches périodiques à contraintes strictes.

-traitement arrière plan

- serveur de tâches

# Tâches apériodiques en arrière plan

**Prise en compte de tâches apériodiques : arrière plan**

**Principe:**
Tâches apériodiques ordonnancées en mode FIFO quand aucune tâches périodiques n'est activée

**Implantation avec RM:**
Donner aux tâches apériodiques la priorité la plus faible.

**Inconvénient:**
Très mauvais temps de réponse si charge périodique importante.

# Serveurs de tâches apériodiques

**Prise en compte de tâches apériodiques : serveurs**

**Principe:**
Une tâche périodique appelée serveur apériodique active dans son temps d'exécution, appelé capacité du serveur, les tâches apériodiques en attente selon sa politique propre.

C'est une forme de réservation garantie de temps CPU consacré aux tâches apériodiques

# Serveurs de tâches apériodiques

**Prise en compte de tâches apériodiques :**

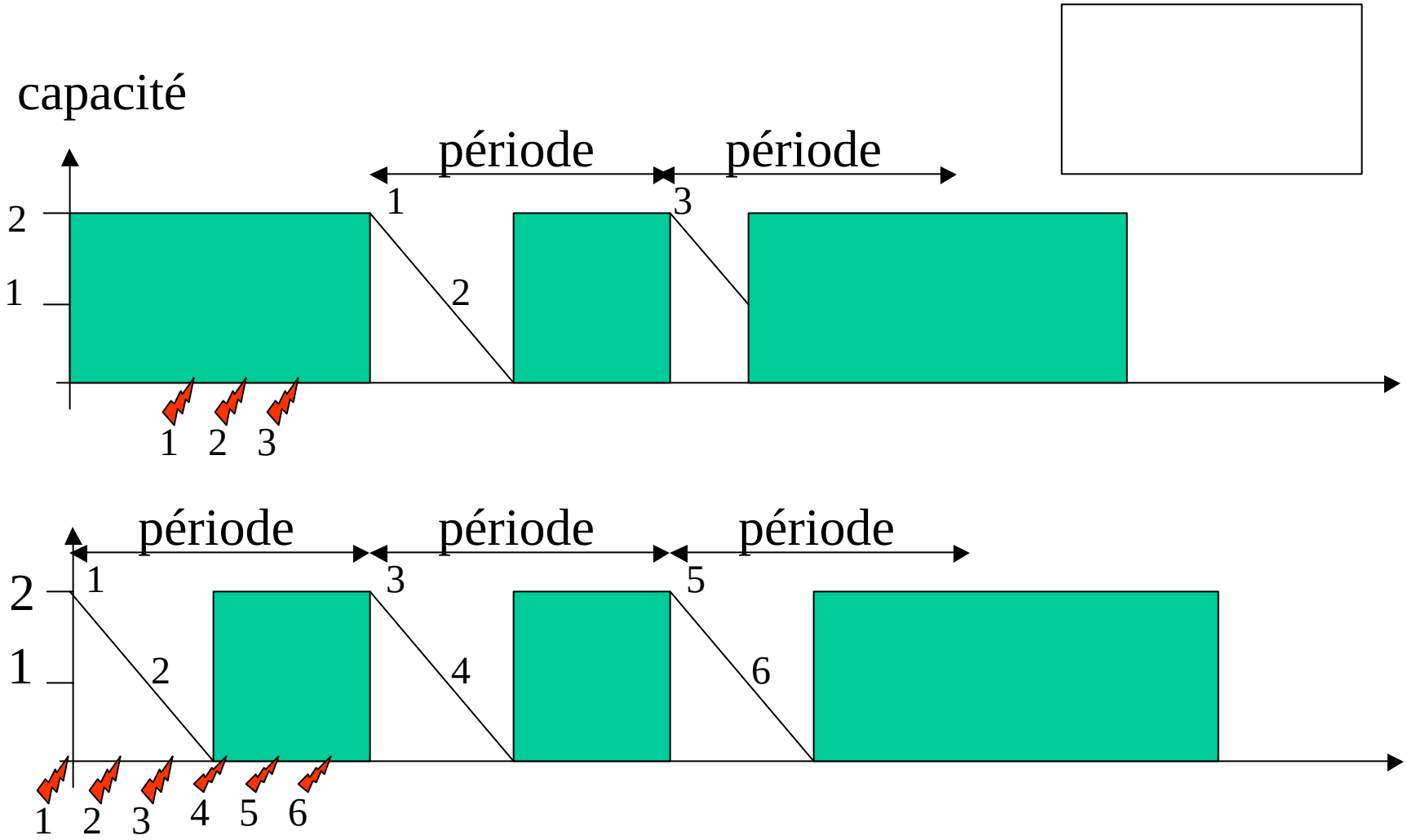
**Exemple du serveurs de scrutation**

Quand il est activé, le serveur exécute les tâches en attente jusqu'à épuisement de celles-ci ou de sa capacité.

Dès qu'il n'y a plus de tâche en attente, il se suspend jusqu'à sa prochaine exécution périodique.

Sa capacité est réinitialisée â chaque nouvelle exécution

# Serveur à scrutation



# Serveur sporadique

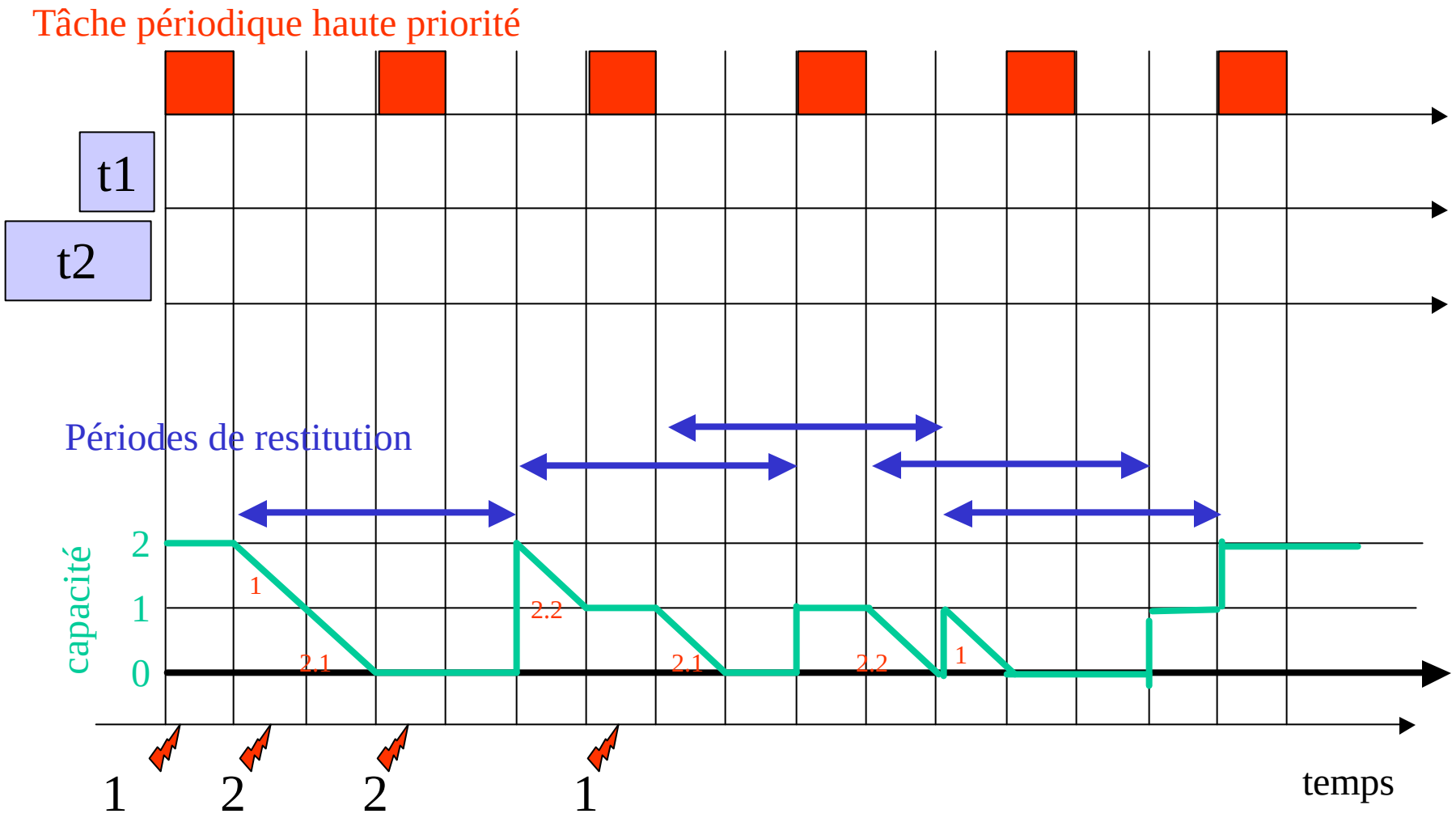
Le serveur sporadique s'exécute chaque fois qu'il y a une demande de tâche sporadique **et** qu'il a une capacité non nulle **et** que sa priorité est éligible (pas de tâche plus prioritaire).

Il s'arrête dès qu'il a épuisé sa capacité ou servit toutes les demandes en attente.

Une certaine capacité lui est restituée, une période après le début de son chaque exécution.

La quantité restituée est égale au CPU consommé entre le début et la fin de son exécution.

# Serveur sporadique



# Insertion de tâches apériodiques

Il existe d'autres méthodes pour prendre en compte des tâches apériodiques dans un contexte périodique.

Certaines calculent le retard maximum qu'elles peuvent imposer aux tâches périodiques pour insérer les tâches apériodiques dans ces trous sans créer de retard sur les tâches périodiques.

-si les contraintes sur les tâches apériodiques sont souples elles utilisent « au mieux » ces espaces sans garanti.
-si les contraintes sur les tâches apériodiques sont strictes un test d'acceptation est fait et la tâche n'est acceptée que si une nouvelle séquence d'ordonnancement qui respecte toutes les échéances peut être construite/calculée (dynamiquement).

# Tâches dépendantes

Dépendances:
-contraintes de précédences
-contraintes d'exclusions mutuelles

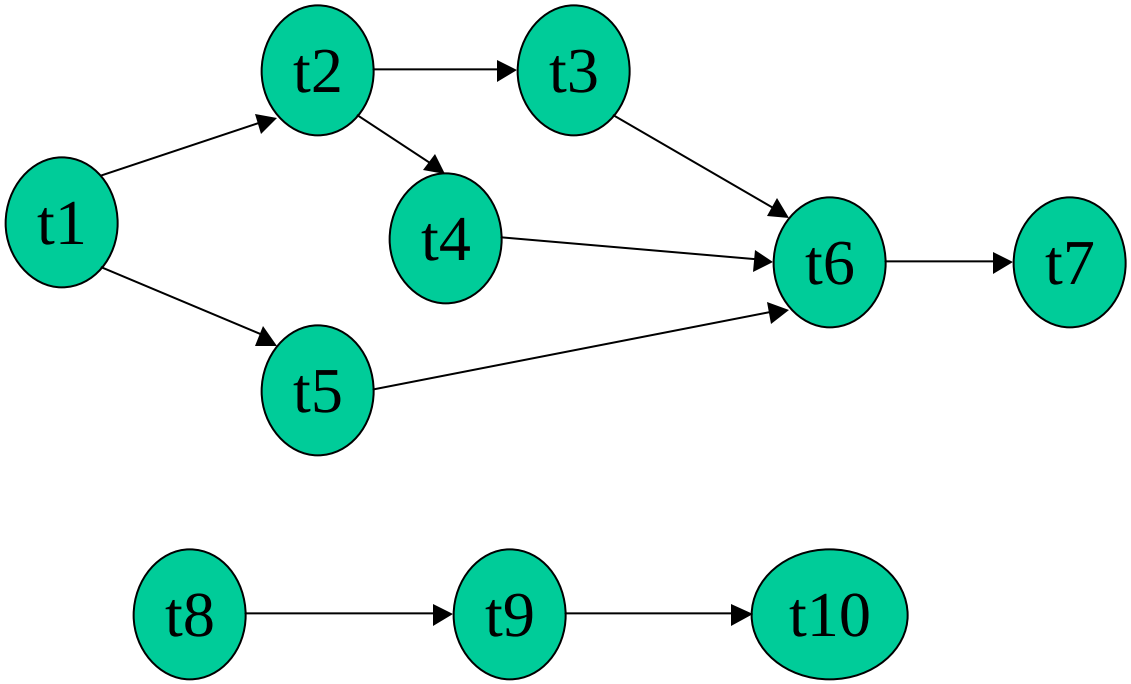
Problème d'ordonnancement de tâches dépendantes NP complet donc non solvable efficacement en ligne.

3 approches de contournement:
-tests suffisants et réserver des ressources pour l'ordonnanceur
-problème en deux phases: une traitée hors ligne et l'autre en ligne
-hypothèses restrictives sur la régularité des tâches

# Précédences

Certaines tâches ne peuvent s'exécuter que si d'autres tâches se sont exécutées avant.

Arbre de précédences:



# Précédences

Contraintes prises en compte dans le modèle des tâches périodiques avec un ordonnanceur préemptif par transformation.

Impose la même période aux tâches

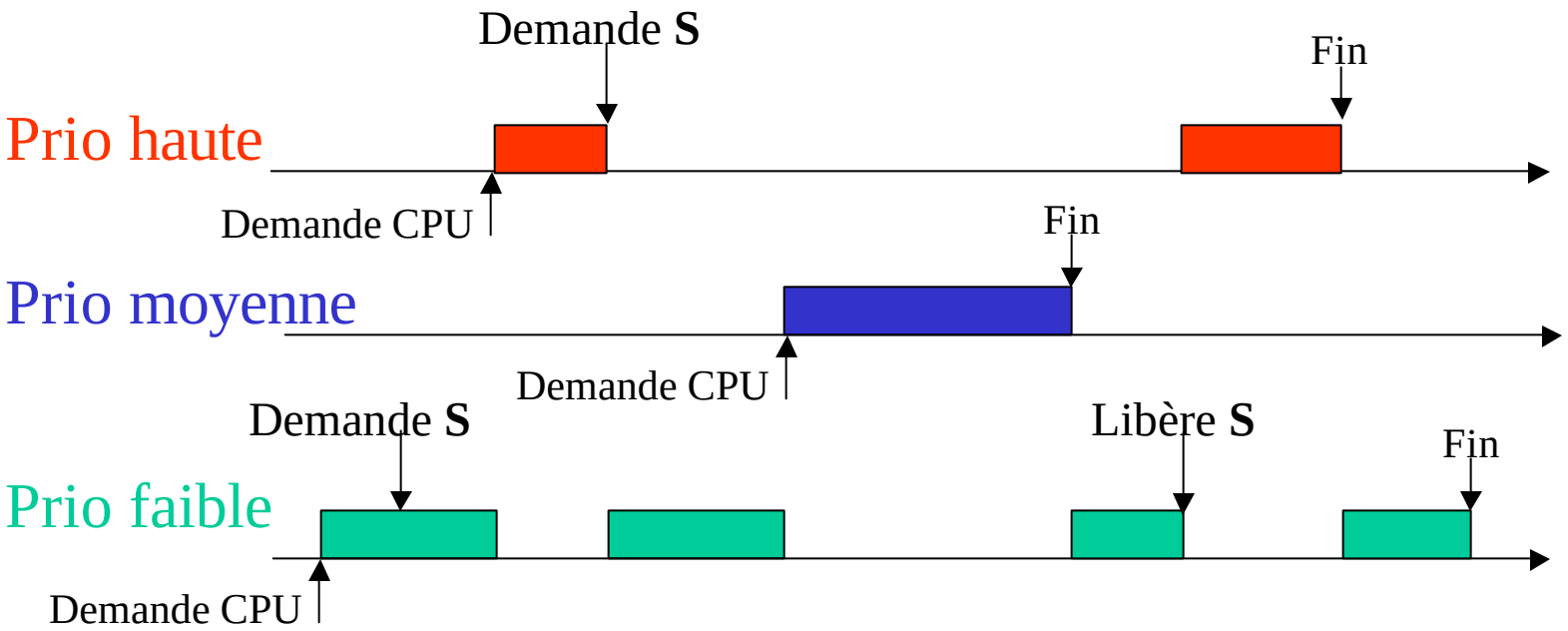
Pour RM si $t_i \rightarrow t_j$ forcer
$r_i <= r_j$ ( r date d'activation )
Prio $t_i$ > Prio $t_j$ en respectant les règles relatives aux périodes/priorités

Pour DM si $t_i \rightarrow t_j$ forcer
$r_i <= r_j$ ( r date d'activation )
$D_i <= D_j$ ce qui impliquera naturellement
Prio $t_i$ > Prio $t_j$ en respectant les règles relatives aux périodes/priorités

# Problèmes liés à l'exclusions

**Problème de l'inversion de priorité:**

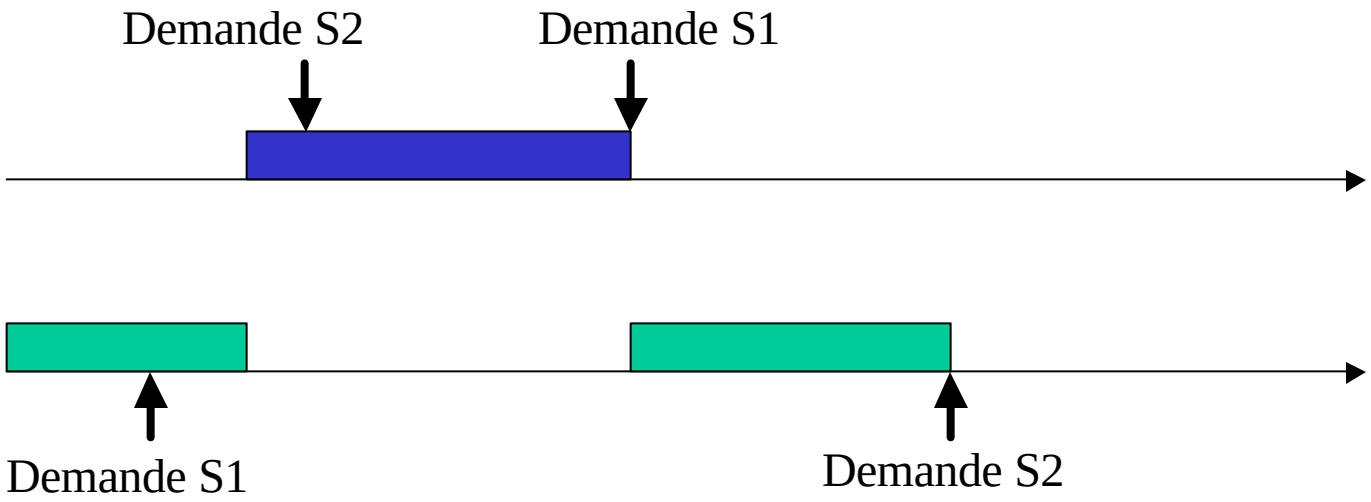
S'il y a exclusion mutuelle, l'approche préemptive basée sur les priorités conduit à des situations où une tâche plus prioritaire ne peut s'exécuter à cause d'une tâche moins prioritaire. Ceci invalide une analyse de type RM.



# Problèmes liés à l’exclusions

**Etreinte fatale (deadlock)**

Situation qui amène le système à se bloquer définitivement



# Solutions pour l’exclusion

## L’héritage de priorité simple

**Principe**: rehausser la priorité du processus bloquant à celle du processus qu’il bloque.

**Effet**: Limitation du temps de blocage, plus de possibilité de chaînes de blocage

**Limites**:

Etreintes fatales toujours possibles

Beaucoup d’interactions entre tâches entraîneront une diminution du nombre de priorités effectives et un regroupement des tâches sur quelques priorités hautes

# Solutions pour l'exclusion

**Protocole avec priorité plafonnées**

Principe:

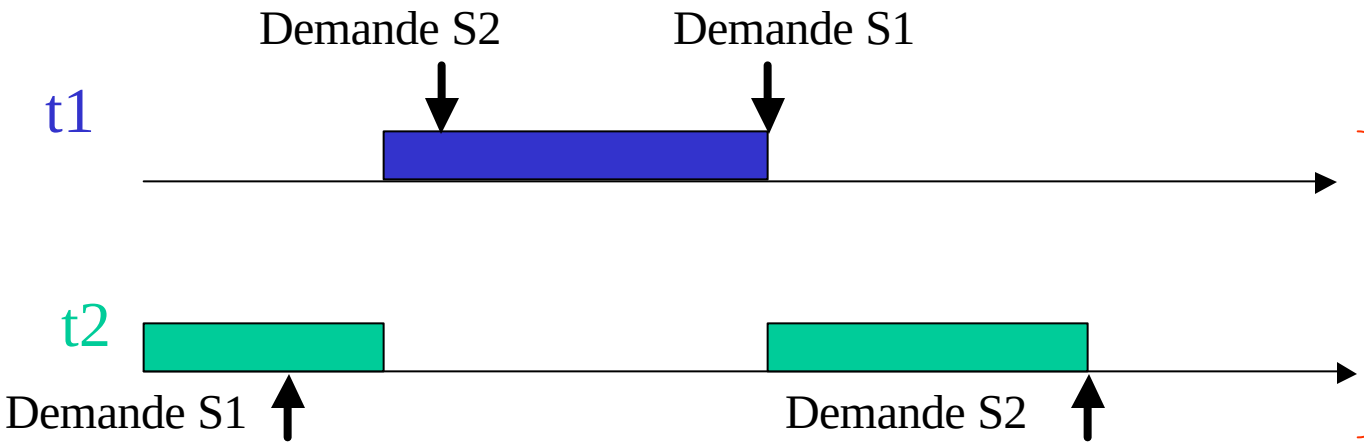
- On définit une valeur statique associée à un sémaphore qui est la valeur de priorité maxi des tâches qui l'utilisent

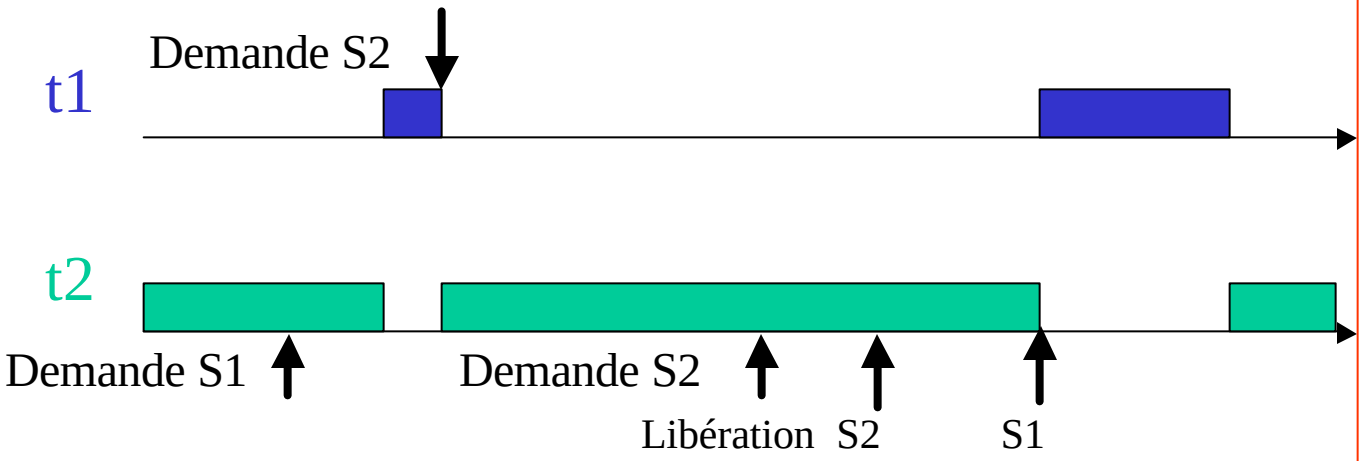
- Une tâche n'entre en section critique (obtient le sémaphore) que si sa priorité courante est supérieure à toutes les priorités des sémaphores en cours d'utilisation lors de sa demande.

- En section critique la tâche hérite de la priorité de la tâche la plus prioritaire qu'elle bloque.

# Solutions pour l'exclusion

**Protocole avec priorité plafonnées:** Evitement de l'étreinte fatale



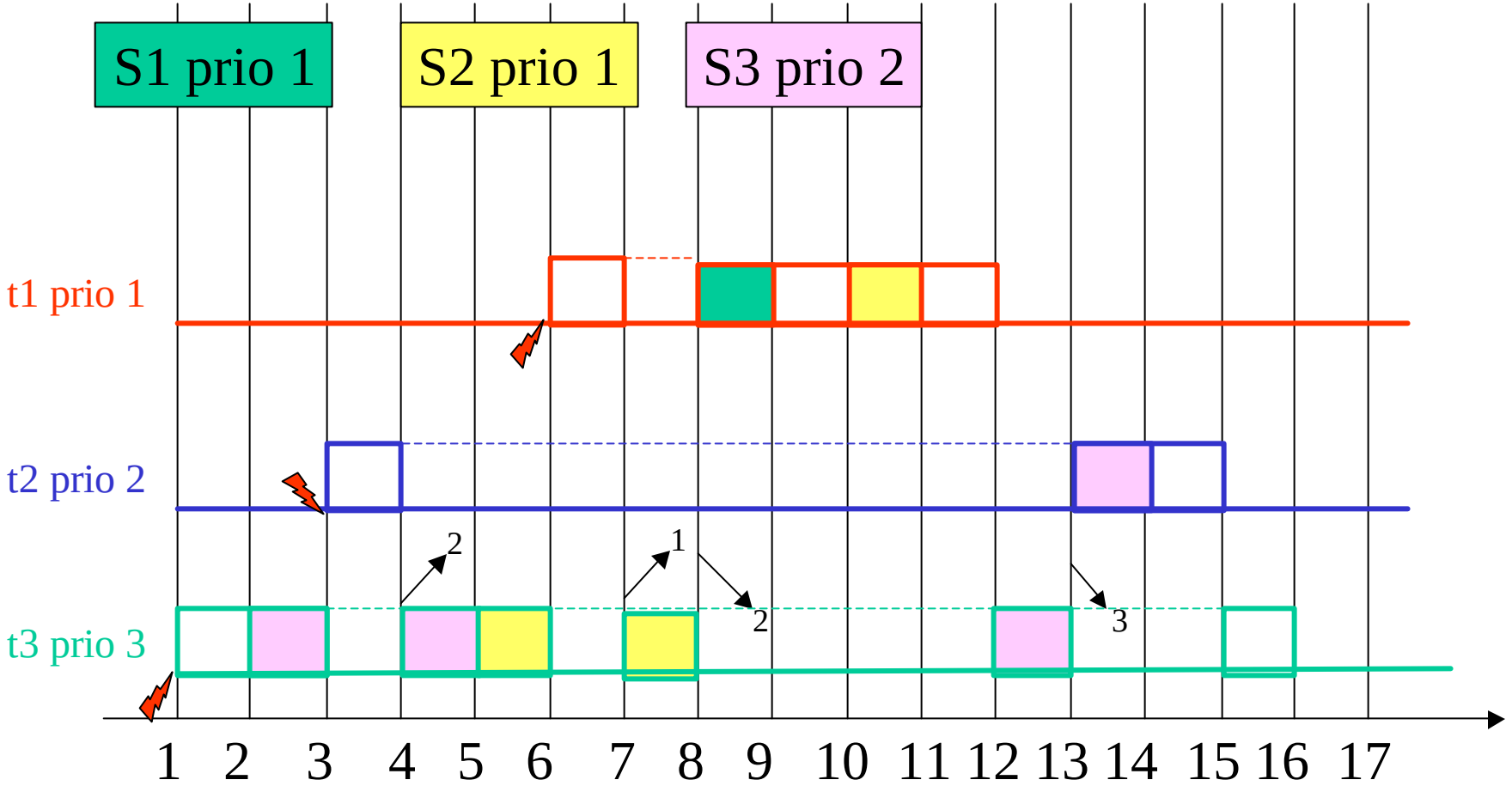




Priorités
t1 =1, t2 =2
S1=1, S2=1

Quand t1 demande S2 accès est refuse car Prio t1 pas supérieure à celle de S2

# Solutions pour l'exclusion

## **Protocole avec priorité plafonnées:** Exemple



# Solutions pour l'exclusion

## **Protocole avec priorité plafonnées:** Test

Un test d'ordonnancement suffisant avec le protocole à plafonnement de priorités pour l'ensemble de tâches $\{t_i\}$ de périodes $p_i$ et durée $c_i$ est:

$\forall i, 1 <= i <= n$

$$B_i/p_i + \sum_{i=1}^{n} (c_i/p_i) <= i\,(\,2^{(1/i)} - 1\,)$$

$B_i/p_i$ est le cas pis de blocage d'une tâche par une tâche de plus faible priorité dans l'ensemble.

Si ce test échoue une analyse exacte doit être faite (voir TD).

# Solutions pour l’exclusion

**Protocole avec priorité plafonnées:** Conditions d’application

Ce protocole marche sous les conditions suivantes:

- les sémaphores doivent être relâchés entre deux exécutions d’une même tâche

-le verrouillage des sémaphores doit se faire en ordre pyramidal (sans entrelacement)

OK pour P1 … P2 … V2 … V1

KO pour P1 … P2 … V1 … V2

# Solutions pour l’exclusion

**Héritage immédiat des priorités (Highest Locker Protocol)**

**Problème:** L’algorithme avec priorités plafonnes est lourd à implanter

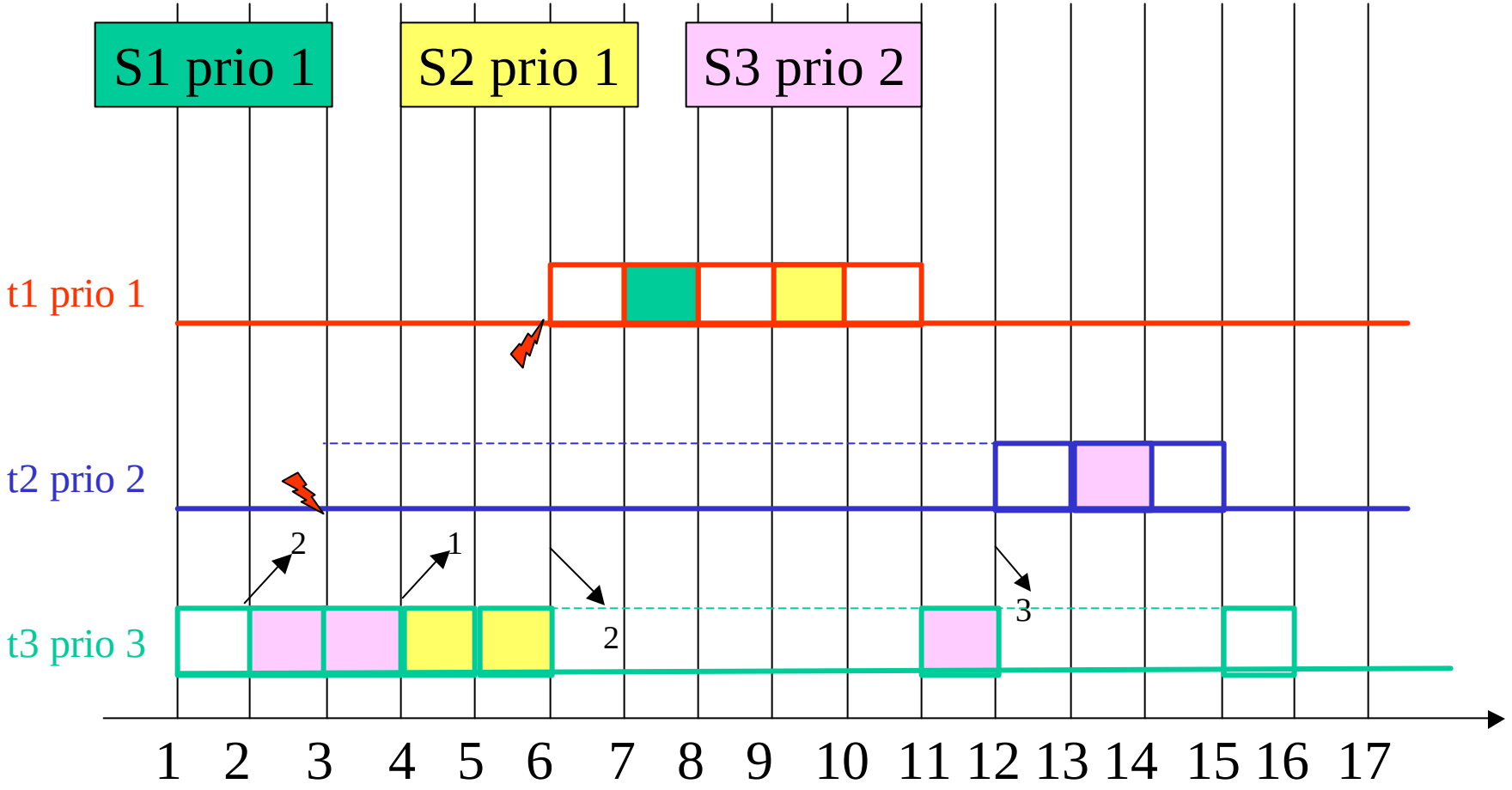
**Solution:** Variante beaucoup plus utilisée en pratique, le HLP

**Principe:**

- priorité associée à un sémaphore = plus haute priorité des tâches utilisatrices
- quand une tâche prend un sémaphore elle hérite immédiatement de la priorité immédiatement supérieure au sémaphore

# Solutions pour l'exclusion

## **Héritage immédiat des priorités HLP :** Exemple



# Solutions pour l'exclusion

## Allocation de ressources basée sur une pile

**(Stack Resources Policy SRP [BAK])**

Utilisable avec des priorités dynamiques par exemple EDF

Nouveau paramètre de tâche: **le niveau de préemption** $\Upsilon$ selon la règle

Si $\pi_i > \pi_k$ et $r_i > r_k$ alors $\Psi_i > \Psi_k$

$\pi$ priorité et r date d'activation

Remarque: Si les priorités sont statiques $\Psi$ peut être pris égal a $\pi$

Protocole:

Ressource a un **plafond dynamique de priorité** = $\Psi$ max des tâches qui l'utilisent à cet instant

Le **plafond courant du système** S = maxi des plafonds des ressources en cours d'utilisation

Quand une tâche i demande une ressource elle ne l'obtient que si son $\Psi i > S$
ou si elle détient la ressource qui fixe le plafond courant.

# Ordonnancement dynamique dans les systèmes répartis

Il n’existe pas de solution connue à ce jour pour résoudre le problème de l’ordonnancement dynamique avec garanti des contraintes de temps dans un système répartis.

# Ordonnancement statique

# Définition

**Objet:**
Construction hors ligne d'une séquence complète de planification sur la base de tous les paramètres temporels des tâches.

**Conséquences:**
- hypothèse de régularité temporelle du monde externe
- événements asynchrones ne sont vus/observés qu'à certaines dates précises (approche synchrone)

L'ordonnancement à l'exécution est piloté par une
**table d'ordonnancement cyclique**
produit de l'algorithme hors ligne.

# Organisation temporelle

## Système séquence par le temps

-L’ordonnancement est périodique
-Le temps découpe en unité de base insécable : cycle de base (tick)
-Le cycle de base est incrémente par une IT de l’horloge temps réelle

Si le système est réparti, les horloges doivent être synchronisées avec une précision inférieure au cycle de base.

# Table d’ordonnancement

Contient la séquence des activations des tâches à exécuter à chaque IT d’horloge au cours d’une période/cycle du système.

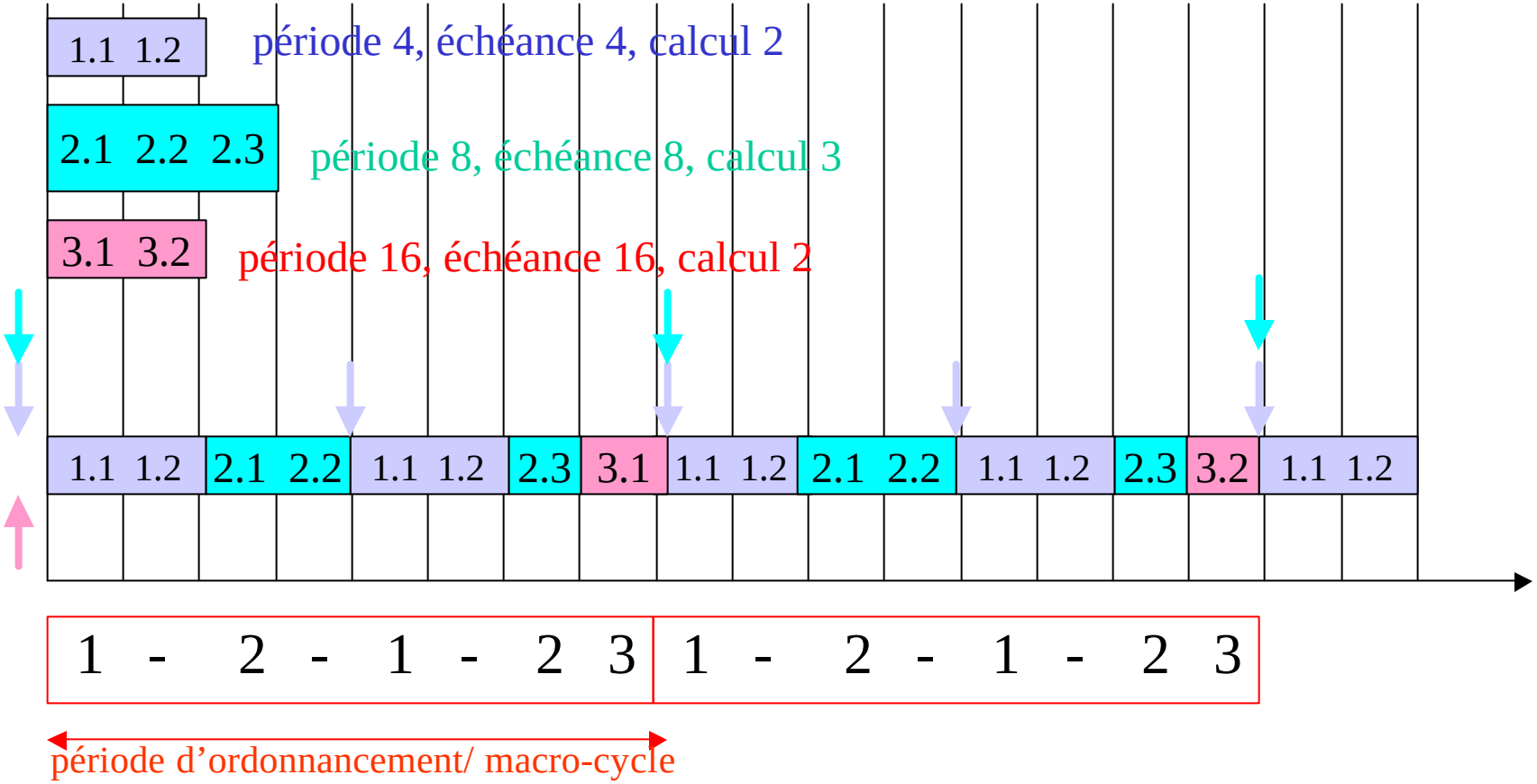
La période du cycle (macro-cycle) définit la taille de la table

Chaque transaction est périodique et dure un multiple du cycle de base

La table d’ordonnancement cyclique a un cycle dont la durée est le PPCM des taches périodiques qui composent le système.

# Table d’ordonnancement

Exemple: simulation du rate monotonic à partir du cas pire



# Point de vue sur les systèmes critiques

Xu et Parnas dans :On satisfying constraints in hard real time systems » in Proceeding of the ACM Sigsoft 91 (Conference on software for Critical Systems)

« Pour satisfaire les contraintes de temps dans les systèmes temps réel durs, le souci premier doit être la prédétermination du comportement du système.
L'ordonnancement statique hors ligne est le plus souvent le seul moyen pratique d'atteindre un comportement prévisible dans un système complexe »