

Ordonnancement temps réel

Laurent Pautet

Laurent.Pautet@enst.fr

Version 1.5



Problématique de l'ordonnancement temps réel

- En fonctionnement normal, respecter les contraintes temporelles spécifiées par toutes les tâches
- En fonctionnement anormal, limiter les effets des débordements temporels et assurer le respect des contraintes temporelles des tâches les plus critiques

Dans la suite, on veillera à ce que les temps nécessaires à la mise en œuvre de l'algorithme d'ordonnancement ainsi que celui de changement de contexte soient négligeables ce qui implique une complexité faible et une implémentation efficace



Définitions

- **Tâches dépendantes ou indépendantes**
 - Les tâches indépendantes ne partagent que le processeur
 - Les tâches dépendantes partagent d'autres ressources ou sont reliées par des contraintes de précédence
- **Ordonnancement préemptif ou non**
 - Un ordonnanceur préemptif peut interrompre une tâche au profit d'une tâche plus prioritaire
 - Un ordonnanceur non préemptif n'arrête pas l'exécution de la tâche courante
- **Ordonnancement hors ligne ou en ligne**
 - Un ordonnancement hors ligne est pré-calculé avant exécution puis est exécuté avec ou sans préemption
 - Un ordonnancement en ligne décide dynamiquement avec ou sans préemption de l'exécution des tâches



Définitions

- Ordonnancement optimal
 - Algorithme qui produit un ordonnancement pour tout ensemble de tâches ordonnançables (si un algorithme le fait, lui le fait aussi)
- Test d'ordonnancement
 - Formule qui fournit une condition nécessaire et/ou suffisante pour qu'un algorithme satisfasse les contraintes temporelles d'un ensemble de tâches
- Test d'admission
 - Formule qui vérifie que l'ajout d'une nouvelle tâche préserve le respect des contraintes temporelles de l'ensemble de tâches précédent



Notations

- Caractéristiques de la tâche τ_i
 - C_i : durée de calcul de τ_i
 - S_i : date d'activation de τ_i
 - D_i : date d'échéance de τ_i
 - T_i : période de τ_i
 - $U_i = C_i / T_i =$ utilisation du processeur pour τ_i
- Caractéristiques du système
 - N : nombre de tâches périodiques
 - $U = \sum U_i =$ utilisation globale du processeur
- Opérateurs
 - Plafond $\lceil x \rceil$ (x si x est entier sinon l'entier supérieur)
 - Plancher $\lfloor x \rfloor$ (x si x est entier sinon l'entier inférieur)



Panorama des algorithmes

- Ordonnancements de tâches périodiques
 - Ordonnement non-préemptif à base de table
 - Ordonnements préemptif à priorité fixe
 - Rate Monotonic Scheduling
 - Deadline Monotonic Scheduling
 - Ordonnements préemptif à priorité dynamique
 - Earliest Deadline First
 - Least Laxity First
- Ordonnements de tâches apériodiques
 - Serveur à scrutation
 - Serveur différé
 - Serveur sporadique
- Partage de ressources
 - *Priority Inheritance Protocol*
 - *Priority Ceiling Protocol*
 - *Highest Locker Protocol*



Tester l'ordonnançabilité

- Soit un système constitué de n tâches à échéance contrainte et à départ simultané
- Il **suffit** de tester l'ordonnancement jusqu'à l'hyper-période ie la période égale au ppcm des périodes des tâches du système.

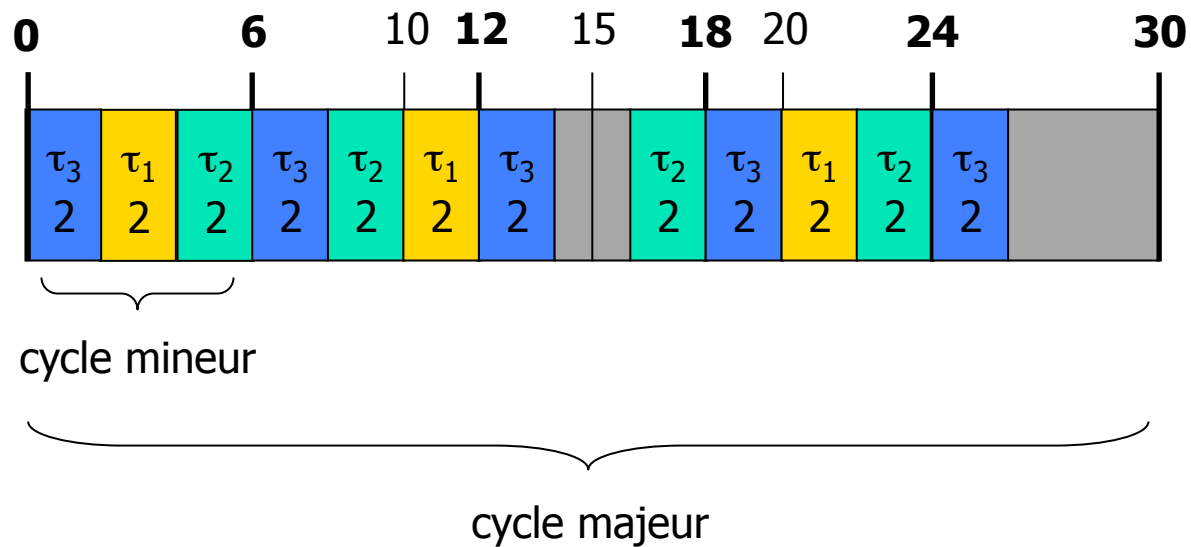


Ordonnancement piloté par une table

- Hypothèses
 - Tâches périodiques
- Principe
 - Cycle majeur = PPCM des périodes
 - Cycle mineur = bloc non-préemptible
 - Le cycle mineur divise le cycle majeur
 - Un ordonnanceur cyclique boucle sur le cycle majeur en exécutant la séquence de cycles mineurs
 - Le cycle mineur correspond à un point de contrôle permettant de vérifier le respect des contraintes

Ordonnancement piloté par une table

	période	échéance	calcul	utilisation
τ_1	10	10	2	0,200
τ_2	15	15	4	0.267
τ_3	6	6	2	0.333





Ordonnancement piloté par une table

- Avantages
 - Mise en œuvre efficace
 - Pas besoin d'exclusion mutuelle entre tâches
- Inconvénients
 - Manque de flexibilité
 - Impact d'une tâche supplémentaire
 - Traitement des tâches apériodiques
 - Construction difficile de la table
 - La découpe constitue un problème complexe dès lors qu'il faut prendre en compte les contraintes temporelles, les ressources partagées, les tâches apériodiques

Ordonnancement par priorité statique

Rate Monotonic Scheduling

■ Hypothèses

- Tâches périodiques, préemptibles et indépendantes
- Activation en début de période ($s_i = 0$)
- Échéance en fin de période ($D_i = T_i$)
- Pire cas C_i connu *a priori*

■ Principe

- Une activation de tâche réveille l'ordonnanceur
- Il choisit la tâche éligible de plus courte période

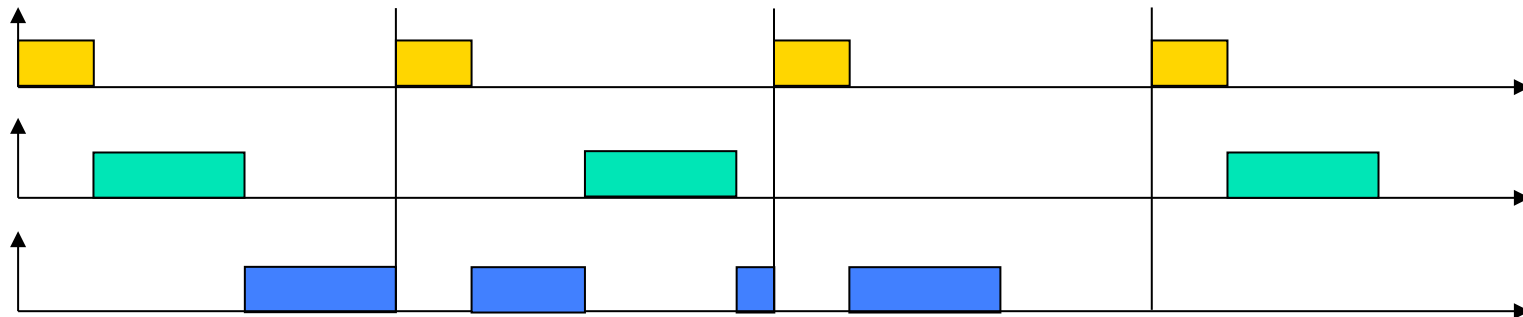
■ Test d'ordonnancement

- Condition nécessaire : $U \leq 1$
- Condition suffisante : $U \leq n \left(2^{1/n} - 1 \right)$
 $\lim_{n \rightarrow +\infty} n \left(2^{1/n} - 1 \right) = \log(2) = 69\%$

Ordonnancement par priorité statique

Rate Monotonic Scheduling

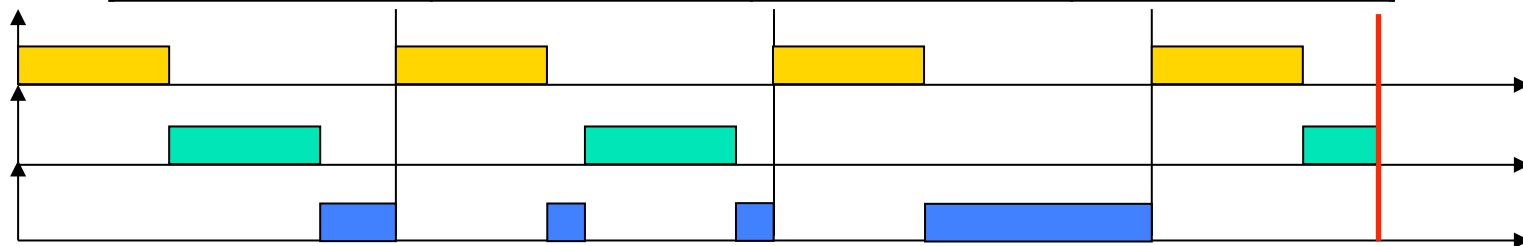
$3 \times (2^{1/3} - 1) \approx 0.78$	période	calcul	utilisation
τ_1	10	2	0.200
τ_2	15	4	0.267
τ_3	36	12	0.333



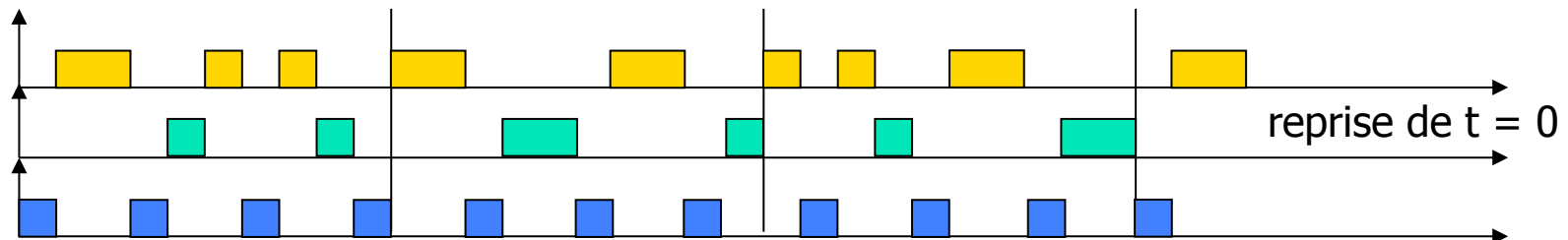
Ordonnancement par priorité statique

Rate Monotonic Scheduling

$3x(2^{1/3}-1)=0.78$	période	calcul	utilisation
τ_1	10	4	0.400
τ_2	15	4	0.267
τ_3	36	12	0.333



$3x(2^{1/3}-1)=0.78$	période	calcul	utilisation
$\tau'_1 = \tau_1 / 2$	5	2	0.400
τ'_2	15	4	0.267
$\tau'_3 = \tau_3 / 12$	3	1	0.333



Ordonnancement par priorité statique

Rate Monotonic Scheduling

■ Avantages

- Simplicité de mise en œuvre
- Optimal pour les ordonnancements à priorité statique
- Répandu dans les exécutifs classiques
- Bon comportement en cas de surcharge

■ Inconvénients

- Surdimensionnement possible du système

■ Condition nécessaire et suffisante en $O(n^2)$

$$\forall i, 1 \leq i \leq n, \min_{(k,l) \in R_i} \left(\sum_{j=1}^{j=i-1} \frac{C_j}{T_j} \frac{T_j}{lT_k} \left\lceil \frac{lT_k}{T_j} \right\rceil + \frac{C_i}{lT_k} \right) \leq 1 \text{ avec } R_i = \left\{ (k,l) \mid 1 \leq k \leq i, l = 1, \dots, \left\lfloor \frac{T_i}{T_k} \right\rfloor \right\}$$

Théorème de la zone critique

Rate Monotonic Scheduling

- Toutes les échéances sont respectées quel que soit l'instant d'arrivée des tâches **ssi** arrivant initialement simultanément elles respectent leur première échéance

$$\forall i, 1 \leq i \leq n, \exists t \leq D_i, W_i(t) = \sum_{j \leq i} C_j * \lceil t/T_j \rceil \leq t$$

- Activer simultanément les tâches de plus fortes priorités que τ_i pour maximiser le retard dans l'exécution de τ_i
- Pour trouver t , on applique une méthode itérative :
 $t = C_i$ et tant que $t \neq W_i(t)$ et $W_i(t) < T_i$, $t' = W_i(t)$
- Il s'agit de démarrer à $t = 0$ puis de calculer le temps nécessaire pour traiter toutes les requêtes activées pendant t . Si t dépasse T_i , il y a dépassement. Sinon, on augmente (strictement) t en lui affectant $W_i(t)$.

Théorème de la zone critique

Rate Monotonic Scheduling

```
function Workload (T, I : Natural)
  return Natural is
    S : Natural := 0;
  begin
    for J in 1 .. I loop
      S := S + C(J) * Ceiling (T, P(J));
    end loop;
    return S;
  end Workload;
```

```
procedure Assert (I : Natural) is
  T : Natural := C (I);
  W : Natural := Workload (T, I);
  begin
    while T < P (I) and then T < W loop
      T := Workload (T, I);
    end loop;
    if T > P (I) then
      raise Overflow;
    end if;
  end Assert;
```

	P	C
τ_1	3	1
τ_2	5	2
τ_3	15	4

1. Assert (1) (avec τ_1)
 1. Workload (1, 1) = 1
2. Assert (2) (avec τ_1 et τ_2)
 1. Workload (2, 2) = 1+2
 2. Workload (3, 2) = 1+2
3. Assert (3) (avec τ_1 , τ_2 et τ_3)
 1. Workload (4, 3) = 2+2+4
 2. Workload (8, 3) = 3+4+4
 3. Workload (11, 3) = 4+6+4
 4. Workload (14, 3) = 5+6+4
 5. Workload (15, 3) = 5+6+4



Ordonnancement par priorité statique

Deadline Monotonic Scheduling

- Hypothèses
 - Identiques à Rate Monotonic Scheduling
 - L'échéance est inférieure à la période ($D_i \leq T_i$)
- Principe
 - Proche de celui de RMS
 - Classement des tâches par échéances croissantes
 - $T_i = D_i$ devient un cas particulier de RMS
- Test d'ordonnancement
 - La condition nécessaire et suffisante existe
 - Condition suffisante : $\sum \frac{C_i}{D_i} \leq n (2^{1/n} - 1)$



Ordonnancement par priorité statique

Deadline Monotonic Scheduling

- **Avantages**
 - Voir RMS
 - Rate Monotonic Scheduling pénalise les tâches peu fréquentes (longue période) mais urgentes (faible échéance).
 - Deadline Monotonic Scheduling s'avère meilleur dans ce cas.
- **Inconvénients**
 - Voir RMS
- **Remarques**
 - À ne pas confondre avec EDF



Ordonnement par priorité dynamique

Earliest Deadline First

■ Hypothèses

- Tâches périodiques, préemptibles et indépendantes
- Activation en début de période ($s_i = 0$)
- Échéance en fin de période ($D_i = T_i$)
- Pire cas C_i connu à priori

■ Principe

- Une activation de tâche réveille l'ordonnanceur
- Il choisit la tâche éligible de plus proche échéance

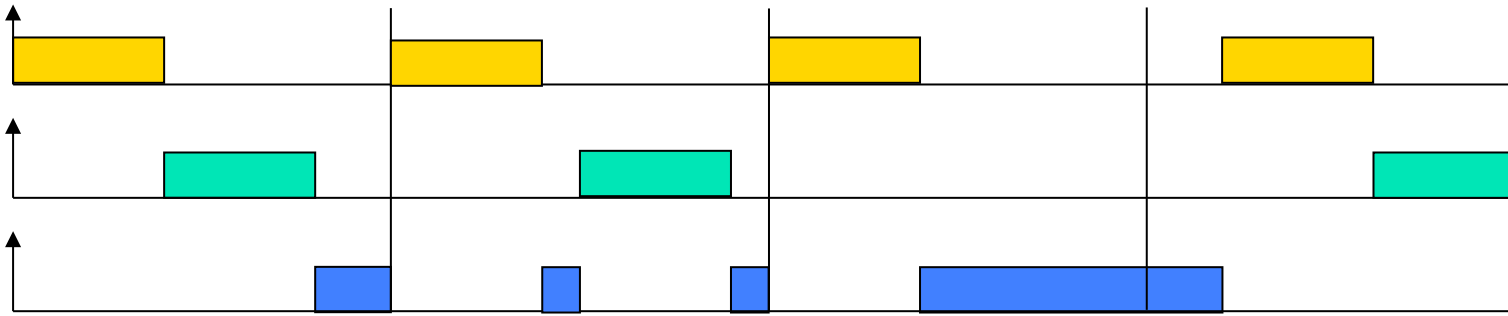
■ Test d'ordonnement

- Condition nécessaire et suffisante : $\sum C_i / T_i \leq 1$

Ordonnancement par priorité dynamique

Earliest Deadline First

	période	calcul	utilisation
τ_1	10	4	0.400
τ_2	15	4	0.267
τ_3	36	12	0.333





Ordonnancement par priorité dynamique

Earliest Deadline First

- **Avantages**

- Utilisation possible de 100% du processeur
- Meilleur que RMS pour les tâches de courte échéance
- Optimal pour les ordonnancements à priorité dynamique si les échéances sont inférieures aux périodes

- **Inconvénients**

- Légère complexité de la mise en œuvre
- Moins répandu dans les exécutifs que RMS
- Mauvais comportement en cas de surcharge

- **Remarques**

- Si D_i est quelconque comparée à T_i , la condition nécessaire et suffisante n'est plus que suffisante



Ordonnancement par priorité dynamique

Least Laxity First

- Hypothèses

- Similaires à celles d'EDF

- Principe

- Une activation de tâche réveille l'ordonnanceur
- Il choisit la tâche éligible de moindre marge
- marge = échéance – temps de calcul restant – temps courant

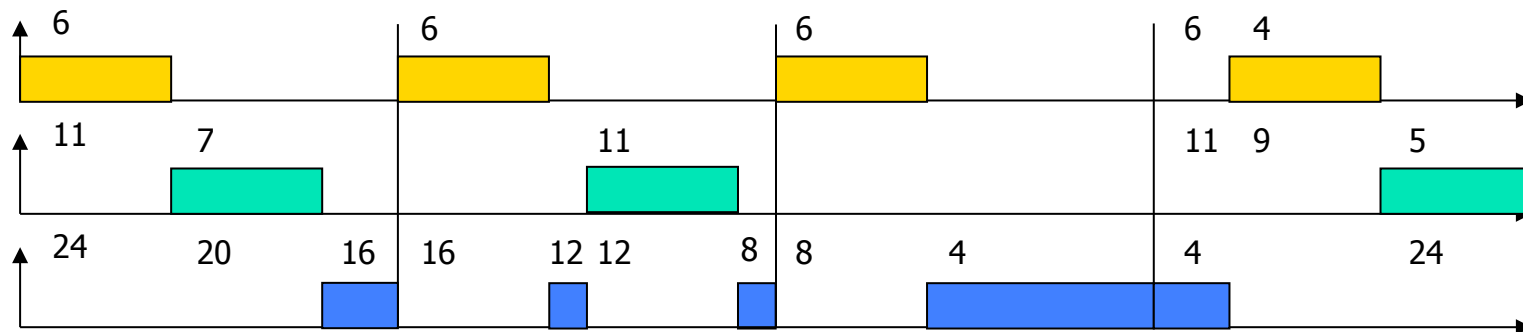
- Test d'ordonnancement

- Condition nécessaire et suffisante : $\sum C_i / T_i \leq 1$

Ordonnancement par priorité dynamique

Least Laxity First

	période	calcul	utilisation
τ_1	10	4	0.400
τ_2	15	4	0.267
τ_3	36	12	0.333





Ordonnancement par priorité dynamique

Least Laxity First

- Avantages
 - Meilleur qu'EDF dans le cas de multi-processeur
- Inconvénients
 - Forte complexité de la mise en œuvre
 - Difficulté du calcul du temps de calcul restant
 - Mauvais comportement en cas de surcharge
 - Nombre de préemptions engendrées supérieur



Ordonnancement des tâches apériodiques

- Principes

- Des tâches apériodiques doivent être intégrées dans un ordonnancement de tâches périodiques
- Les tâches périodiques doivent respecter leurs échéances

- Définitions

- Les tâches apériodiques sont activées à des instants aléatoires
- Les tâches sporadiques se caractérisent par un délai minimum entre deux activations



Ajout de tâches apériodiques

- Principes

- Traiter les tâches apériodiques comme les tâches périodiques lorsque l'ordonnancement le permet
- Le test d'ordonnancement sert de test d'admission dès lors que les tâches apériodiques viennent avec des caractéristiques plus ou moins précises

- Hypothèses

- Le test d'ordonnancement (pas l'algorithme d'ordonnancement) doit être de faible complexité pour que l'admission s'effectue dynamiquement



Ordonnancement des tâches apériodiques

Serveur en arrière plan

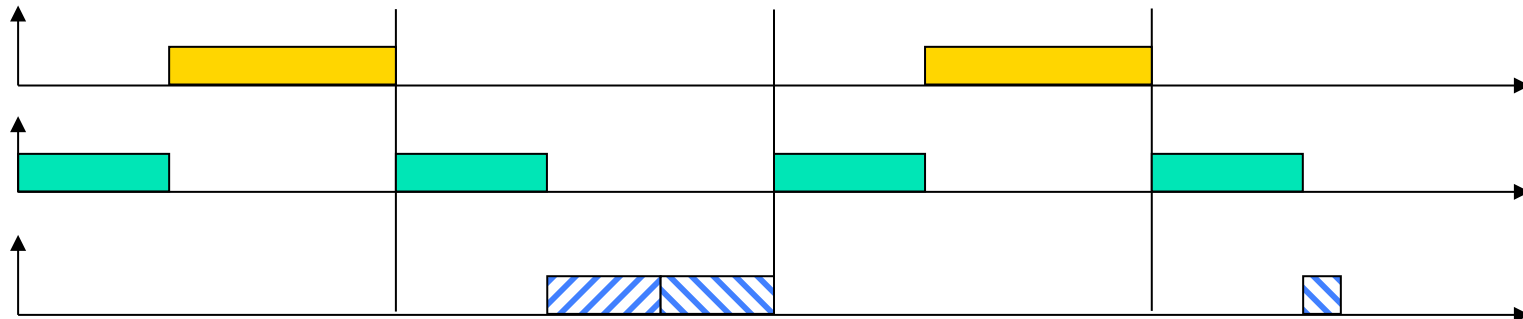
- Principes

- Les tâches apériodiques sont traitées séquentiellement par un serveur de faible priorité
- Le serveur n'a pas de temps de calcul associé ou de capacité (faible priorité)
- L'absence de capacité vient du fait que le serveur remplit les trous dans l'ordonnancement

Ordonnancement des tâches apériodiques

Serveur en arrière plan

	période/activation	calcul	priorité	utilisation/réponse
événement ε_1	7	3		17
événement ε_2	11	4		35
tâche τ_1	20	6	3	0,300
tâche τ_2	10	4	2	0,400
serveur en arrière plan	*	*	10	*



Ordonnancement des tâches apériodiques

Serveur en arrière plan

- Avantages

- Simplicité de mise en œuvre

- Inconvénients

- On ne peut prédire les échéances de traitement des tâches apériodiques
- Les tâches apériodiques peuvent être critiques
- Le mauvais temps de réponse en cas de charge importante est pénalisant



Ordonnancement des tâches apériodiques

Serveur à scrutation

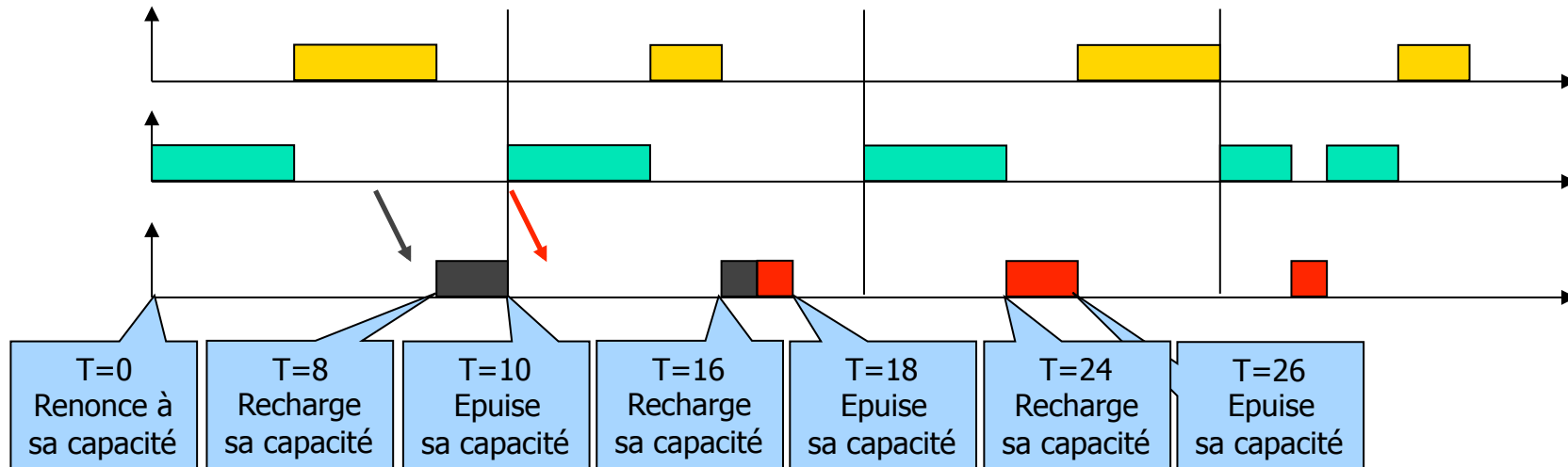
■ Principes

- Les tâches apériodiques sont traitées séquentiellement par un serveur de forte priorité
- Le serveur dispose d'une capacité et d'une période
- La capacité est réallouée toutes les périodes
- Le temps consommé à traiter une tâche apériodique est débité sur la capacité
- Le serveur devenu actif traite toutes les tâches apériodiques dans la limite de sa capacité
- Le serveur devenu inactif (absence de tâche) perd sa capacité jusqu'à la prochaine ré-allocation

Ordonnancement des tâches apériodiques

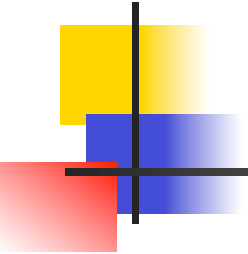
Serveur à scrutation

	période/activation	calcul	priorité	utilisation/réponse
événement ε_1	7	3		17
événement ε_2	11	4		33
tâche τ_1	20	6	3	0,300
tâche τ_2	10	4	2	0,400
serveur à scrutation	8	2	1	0,250



Ordonnancement des tâches apériodiques

Serveur à scrutation

- 
- Avantages
 - Simplicité de mise en oeuvre
 - Inconvénients
 - En relâchant sa capacité, le serveur épuise le temps alloué pour des tâches à venir
 - Mauvais temps de réponse
 - Le traitement peut s'étendre sur plusieurs périodes même pour une tâche de courte durée



Ordonnancement des tâches apériodiques

Serveur différé

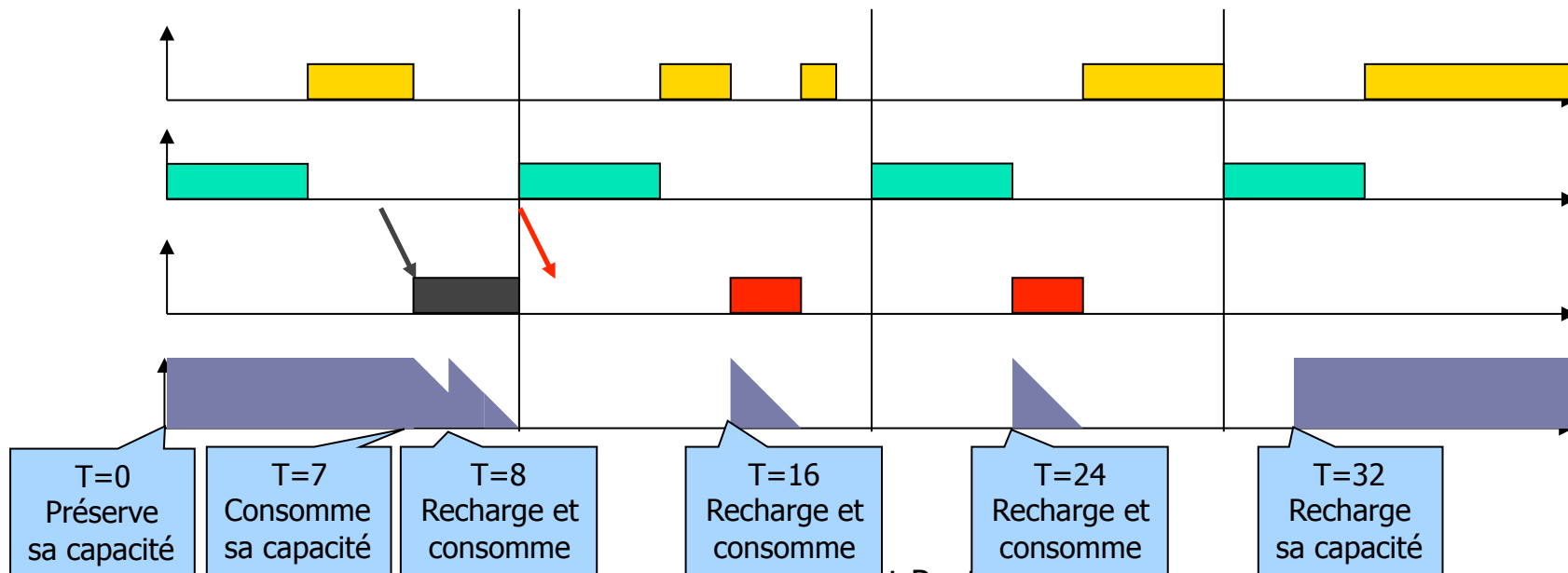
■ Principes

- Les tâches apériodiques sont traitées séquentiellement par un serveur de forte priorité
- Le serveur dispose d'une capacité et d'une période
- La capacité est réallouée toutes les périodes
- Le temps consommé à traiter une tâche apériodique est débité sur la capacité
- Le serveur devient actif lorsqu'une tâche apériodique est à traiter et que sa capacité n'est pas épuisée

Ordonnancement des tâches aperiodiques

Serveur différé

	période/activation	calcul	priorité	utilisation/réponse
événement ε_1	7	3		10
événement ε_2	11	4		26
tâche τ_1	20	6	3	0,300
tâche τ_2	10	4	2	0,400
serveur différé	8	2	1	0,250



Ordonnancement des tâches apériodiques

Serveur différé

- Avantages

- Il consomme le temps du processeur ainsi que sa capacité qu'en présence de tâches apériodique

- Inconvénients

- La consommation de la capacité et la ré-allocation peuvent survenir concomitamment et rallonger artificiellement le temps autorisé au serveur
- Il peut donc se comporter comme une tâche ayant le double de son budget et rendre le système difficilement analysable



Ordonnancement des tâches apériodiques

Serveur sporadique

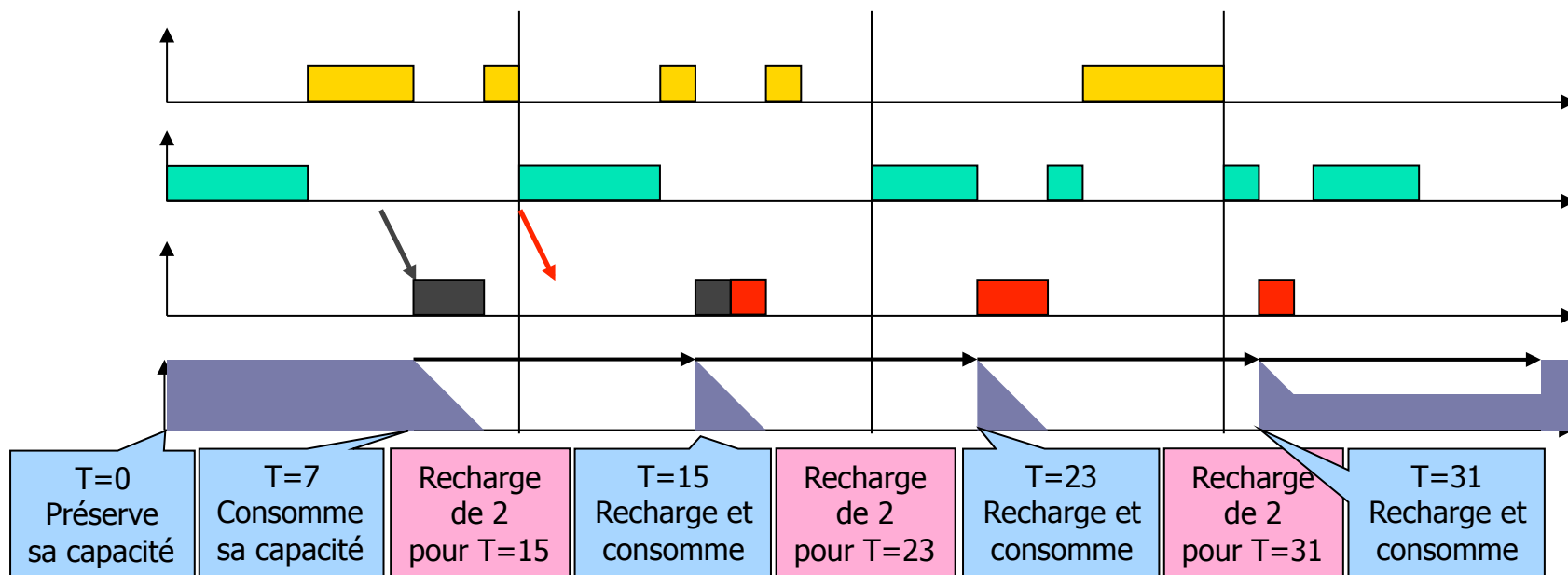
■ Principe

- Les tâches apériodiques sont traitées séquentiellement par un serveur de forte priorité
- Le serveur dispose d'une capacité et d'une période
- Le temps consommé à traiter une tâche apériodique est débité sur la capacité
- Le montant exact de temps consommé est crédité après un délai d'une période
- Le serveur devient actif lorsqu'une tâche apériodique est à traiter et que sa capacité n'est pas épuisée

Ordonnancement des tâches apériodiques

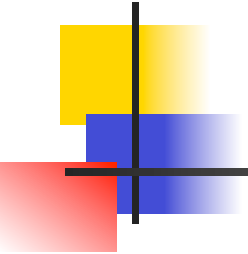
Serveur sporadique

	période/activation	calcul	priorité	utilisation/réponse
événement ε_1	7	3		17
événement ε_2	11	4		32
tâche τ_1	20	6	3	0,300
tâche τ_2	10	4	2	0,400
serveur apériodique	8	2	1	0,250



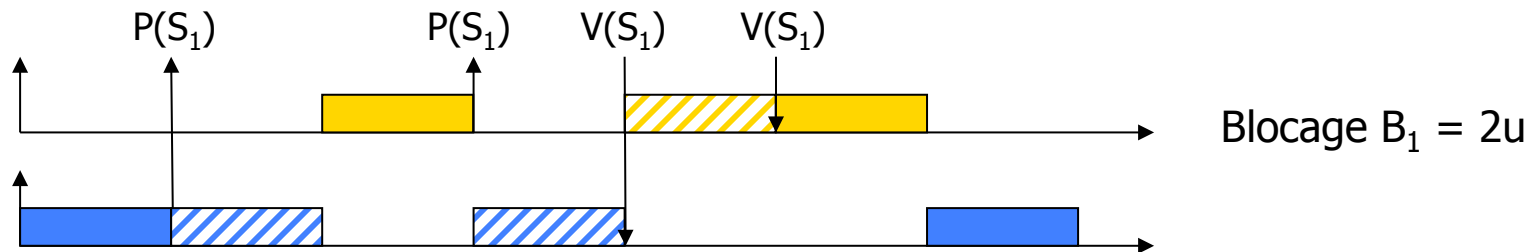
Ordonnancement des tâches apériodiques

Serveur sporadique

- 
- Avantages
 - Meilleur que les serveurs à préservation de bande passante
 - Inconvénients
 - Forte complexité comparée à celle du serveur différé
 - Remarques
 - Une variante consiste à transformer le serveur sporadique en serveur en arrière plan (faible priorité) lorsque sa capacité est épuisée afin de récupérer le temps non-alloué

Partage de ressources

Blocage et Ordonnancement



- Analogie avec les cas précédents
 - B_i la durée la plus longue de blocage potentiel de la tâche t_i par une tâche de priorité inférieure
 - Analogie avec un scénario où pour la tâche t_i , le temps de calcul C_i deviendrait $C_i + B_i$
- L'objectif consiste alors à réduire B_i en instaurant des politiques de partage de ressources adéquates

Partage de ressources

Tests d'ordonnancement

- Condition suffisante d'ordonnancement avec RMS

$$\forall i, 1 \leq i \leq n, \sum_{j=1}^{j=i} \frac{C_j}{T_j} + \frac{B_i}{T_i} \leq n \left(2^{\frac{1}{n}} - 1\right)$$

- Le théorème de la zone critique devient

$$\forall i, 1 \leq i \leq n, \exists t \leq D_i, W_i(t) = \sum_{j=1}^i C_j \left\lceil \frac{t}{T_j} \right\rceil + B_i \leq t$$

- Condition suffisante d'ordonnancement avec EDF

$$\forall i, 1 \leq i \leq n, \sum_{j \leq i} C_j / T_j + B_i / T_i \leq 1$$



Partage de ressources

Priority Inheritance Protocol

- Problèmes

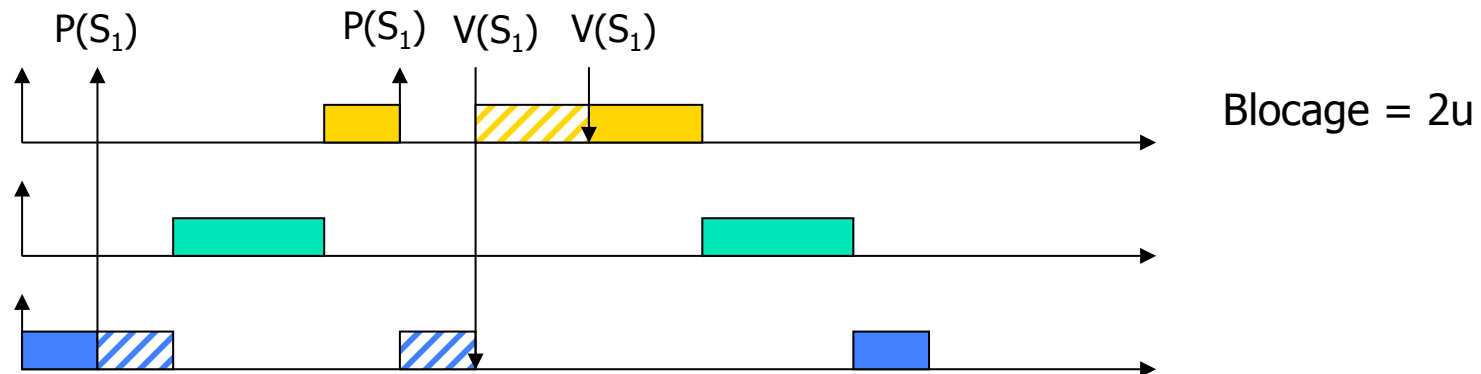
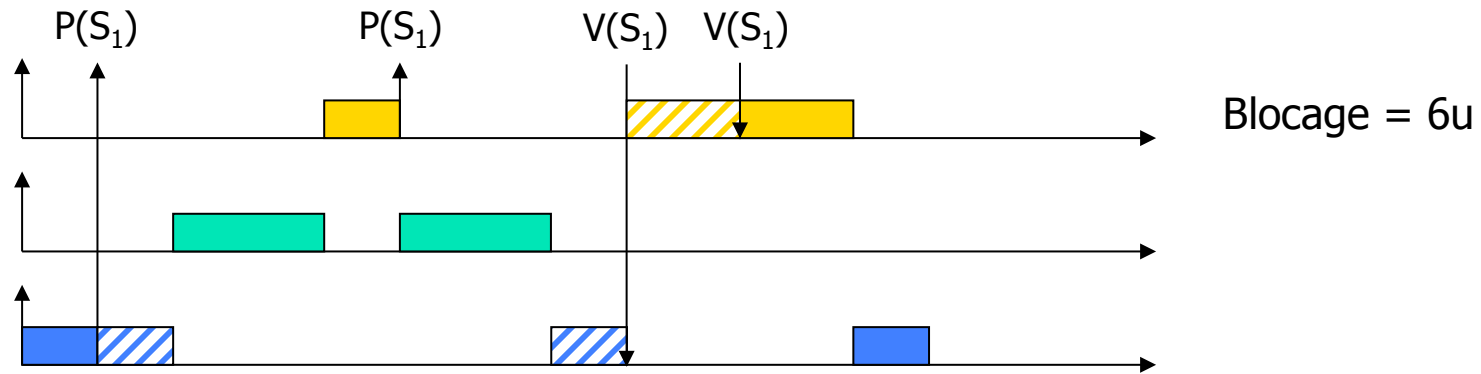
- Un ordonnancement préemptif peut conduire à une situation d'inversion de priorité où une tâche de faible priorité bloque une tâche de forte priorité pendant un temps supérieur à celui de l'exclusion mutuelle
- On ne peut évaluer la borne supérieure de ce temps

- Solution

- L'héritage de priorité monte la priorité de la tâche bloquante au niveau de celle bloquée
- Une fois le sémaphore libéré, la tâche bloquée retrouve sa priorité initiale

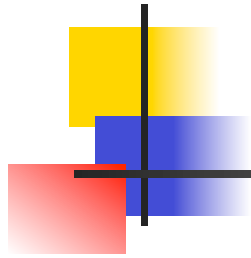
Partage de ressources

Priority Inheritance Protocol



Partage de ressources

Priority Inheritance Protocol



■ Avantages

- Le temps de blocage dû à la situation d'inversion de priorité se limite à l'utilisation de sémaphore par la tâche de faible priorité

■ Inconvénients

- En cas d'accès à plusieurs sémaphores, la tâche de forte priorité va enchaîner les temps de blocage
- Dans le cas d'accès à un sémaphore par plusieurs tâches, les élévations de priorité vont s'enchaîner
- Les interblocages restent possibles (ordre)
- Il y a blocage de tâches intermédiaires qui ne partagent pas le sémaphore en question



Partage de ressources

Priority Ceiling Protocol

- Problèmes

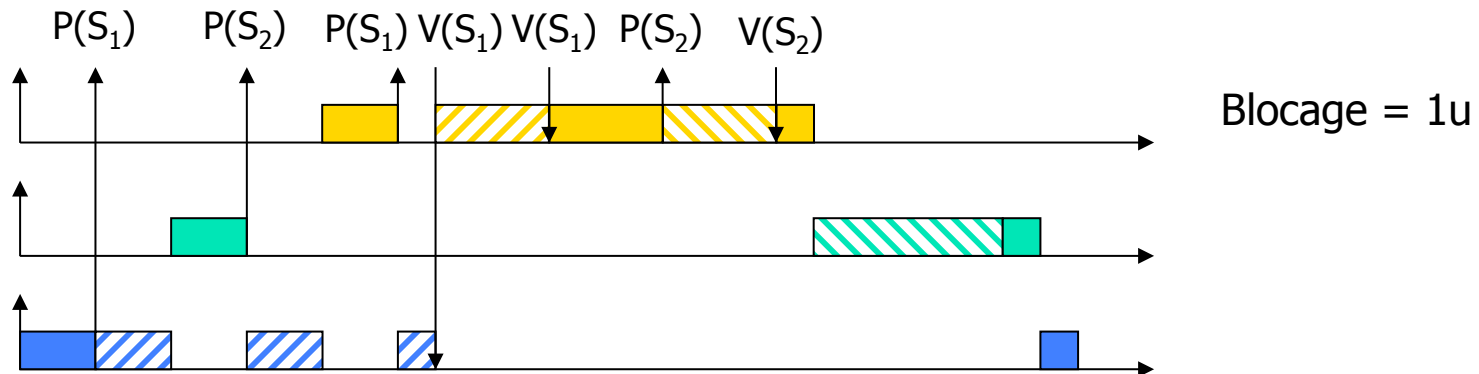
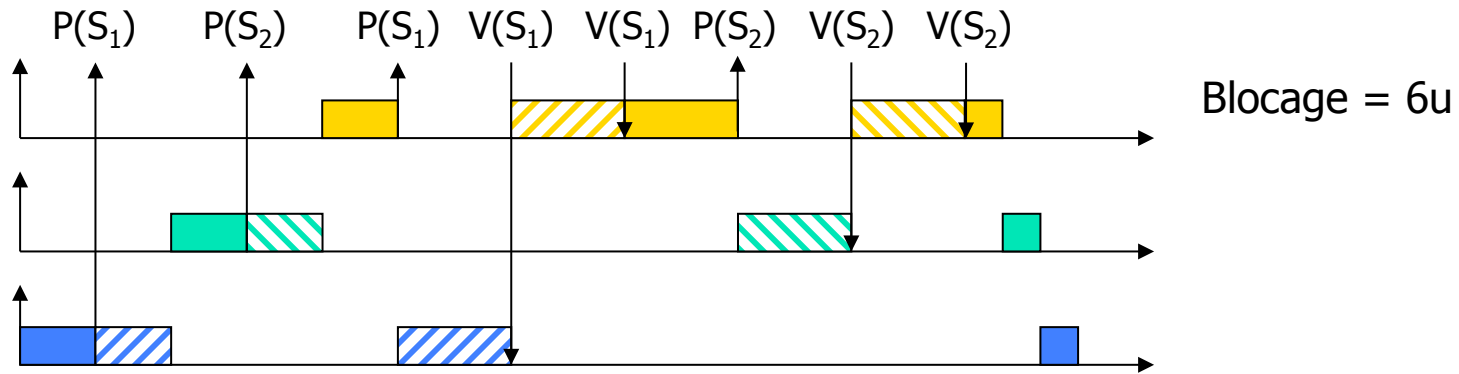
- Les temps de blocage peuvent s'enchaîner
- Les élévations de priorité peuvent s'enchaîner
- Les interblocages restent possibles

- Solution

- La priorité plafonnée (statique) représente la priorité maximum des tâches qui l'utilisent
- Une tâche accède à un sémaphore lorsque sa priorité est strictement supérieure à toutes les priorités plafonnées des sémaphores utilisés
- La tâche bloquante hérite de la priorité de la tâche bloquée la plus prioritaire

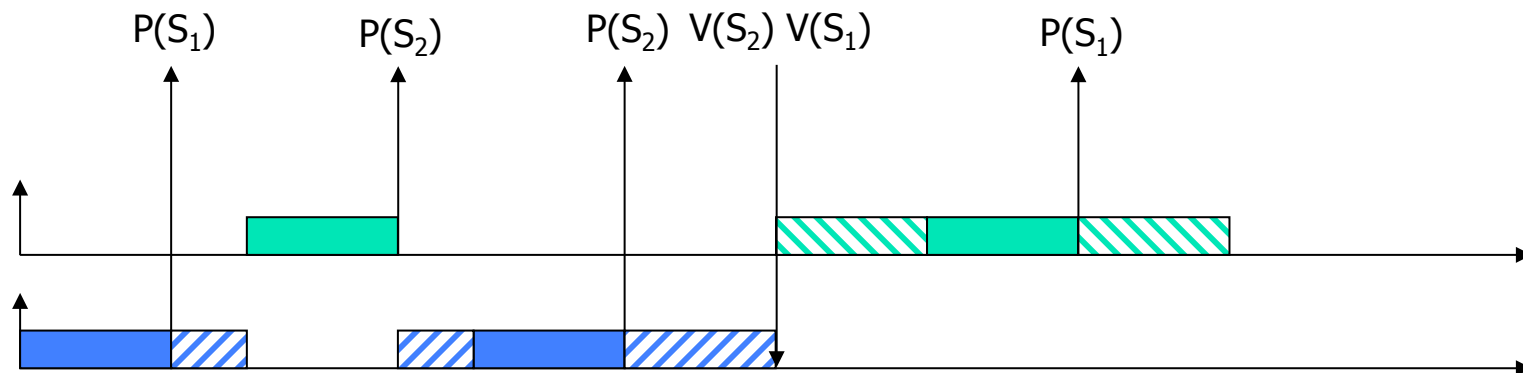
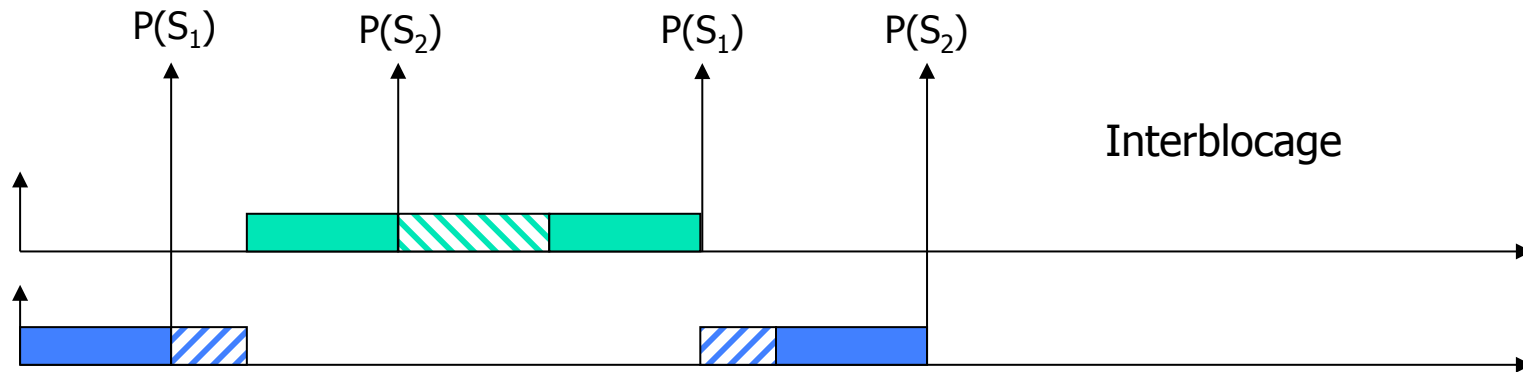
Partage de ressources

Priority Ceiling Protocol



Partage de ressources

Priority Ceiling Protocol





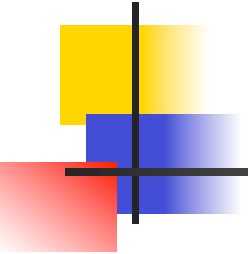
Partage de ressources

Priority Ceiling Protocol

- Avantages
 - Pas d'enchaînement de temps de blocage
 - Pas d'enchaînement d'élévations de priorité
 - Pas de risques d'interblocages
- Inconvénients
 - Forte complexité de la mise en oeuvre

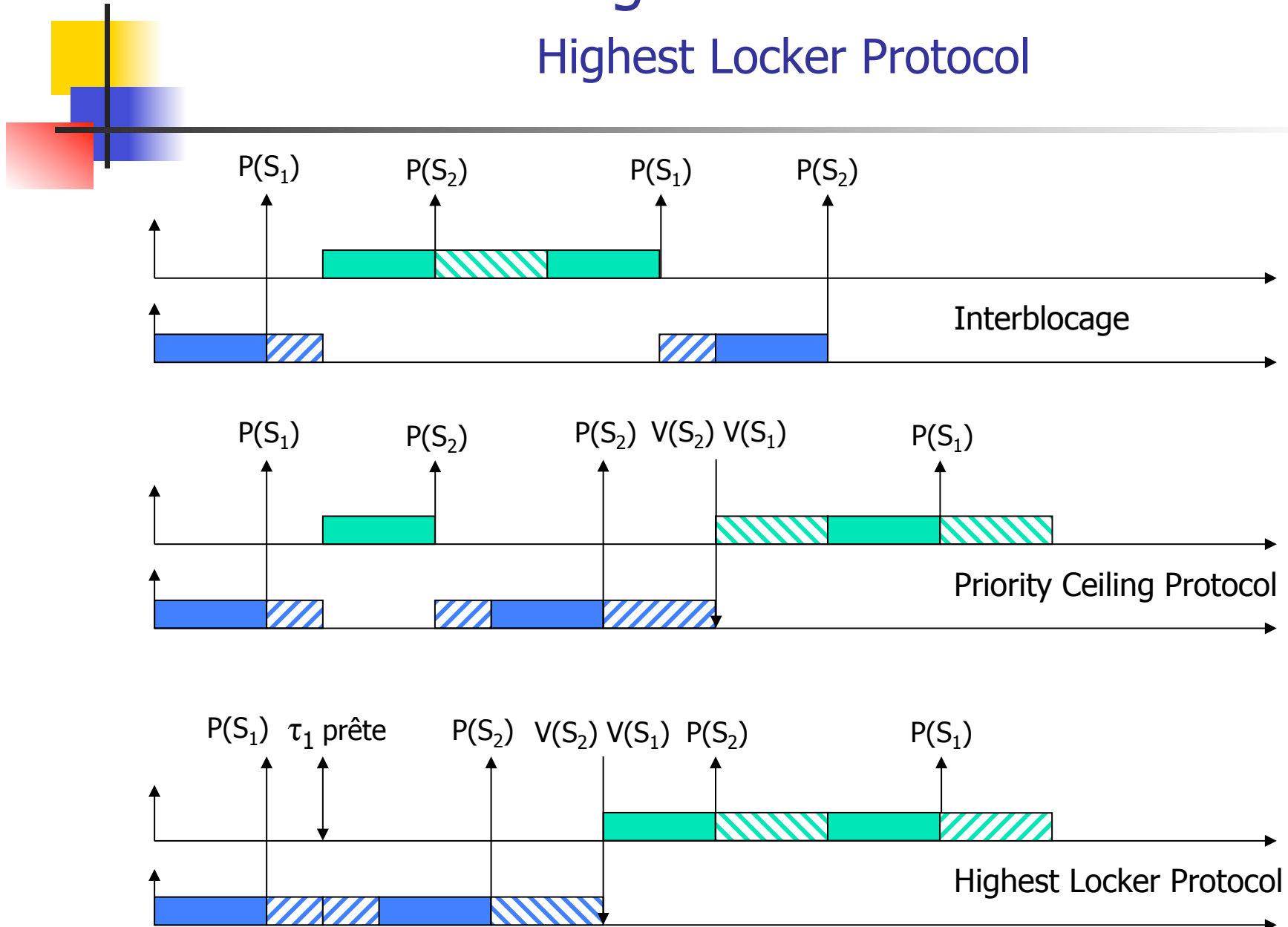
Partage de ressources

Highest Locker Protocol

- 
- Problèmes
 - Forte complexité de la mise en œuvre de PCP
 - Solution
 - La priorité plafonnée (statique) représente la priorité maximum des tâches qui l'utilisent
 - Quand une tâche accède à un sémaphore, elle hérite d'une priorité strictement supérieure à la priorité plafonnée

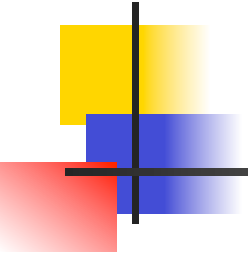
Partage de ressources

Highest Locker Protocol



Partage de ressources

Highest Locker Protocol

- 
- Avantages
 - Moindre complexité comparée à celle de PCP
 - Inconvénients
 - Le temps de blocage en cas d'attente sur un sémaphore se trouve allongé



Conclusions

- 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



Lectures

- C. Bonnet et I. Demeure, Introduction aux systèmes temps réel, Hermes
- F. Cottet, J. Delacroix, C. Kaiser et Z. Mameri, Ordonnement temps réel, Hermès
- G. Buttazzo. Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications Kluwer academic Publishers, Boston, 1997.
- C. Liu and J. Layland. Scheduling Algorithms for Multiprogramming in a Hard Real-time Environment. Journal of the ACM, 20(1):46--61, Jan. 1973.
- L. Sha, R. Rajkumar and J. Lehoczky, "*Priority Inheritance Protocol*": An Approach to real-time synchronisation," IEEE Transaction on Computers 39(9), pp.1175-1185, 1993.
- Min-Ih Chen and Kwei-Jay Lin. Dynamic priority ceilings: A concurrency control protocol for real-time systems. Journal of Real-Time Systems, 2:325--346, 1990.
- T. Baker. Stack-based scheduling of real-time processes. Real-Time Systems, 3(1):67--99, March 1991.
- B. Sprunt, L. Sha, and J. P. Lehoczky, "Aperiodic scheduling for hard real-time system". The Journal of Real-Time Systems, 1, pp. 27-60, 1989.