Ordonnancement online sur plate-forme maître-esclave

Frédéric Vivien

16 décembre 2005

Plan de l'exposé

Notion d'ordonnancement

Les processeurs

- Parallèles
  - Identiques
  - Uniformes

Notion d'ordonnancement

Les processeurs

- Parallèles
  - Identiques
  - Uniformes

Les tâches

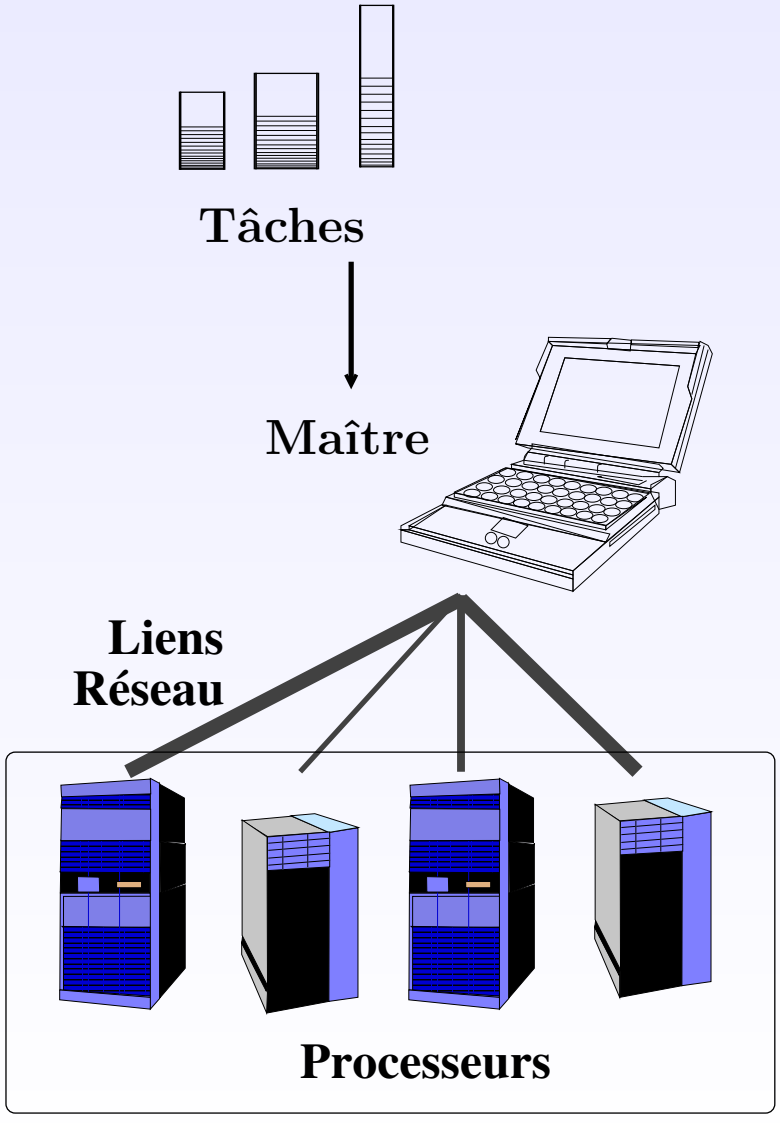
Décrites par :

- leur volume de calcul
- leur volume de communication
- leur date d'arrivée

Notion d'ordonnancement

L'ordonnanceur

- Récupère les tâches
- Les envoie aux processeurs



Notion d'ordonnancement

Le But

Distribuer les tâches aux processeurs, afin de traiter ces tâches

- En respectant les contraintes du système,
  - au niveau des processeurs
  - au niveau des tâches
- En optimisant une fonction objective.

Notion d'ordonnancement

Formellement

- n tâches, m processeurs
- $p_{i,j}$: temps de traitement de la tâche i sur le processeur j
- $c_{i,j}$: temps d'envoi de la tâche i du maître jusqu'à l'esclave j
- r_i : date d'arrivée
- C_i : date de fin de calcul
- Les fonctions objectives principales :
  - makespan : $\max C_i$
  - flot maximal : $\max C_i - r_i$
  - somme des flots : $\sum(C_i - r_i)$

Notion d'ordonnancement

Simplifions un peu le problème

- tâches identiques indépendantes,

Notion d'ordonnancement

Simplifions un peu le problème

- tâches identiques indépendantes,

Indispensable pour avoir des algorithmes performants.

Notion d'ordonnancement

Simplifions un peu le problème

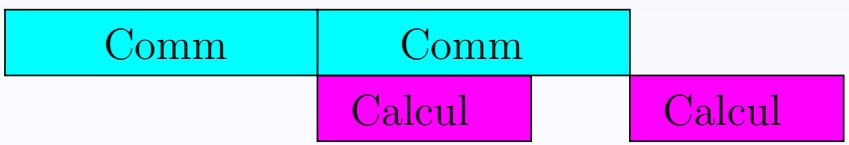
- tâches identiques indépendantes,
- Communications rapides.

Notion d'ordonnancement

Simplifions un peu le problème

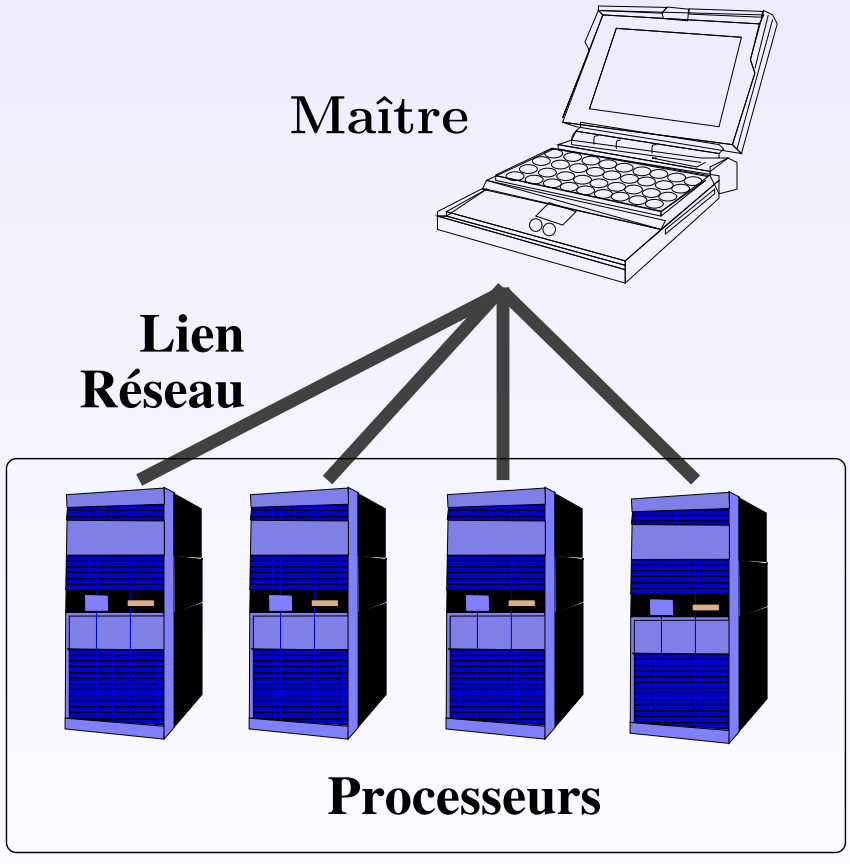
- tâches identiques indépendantes,
- Communications rapides.

Si $c_{j_0} = min\ c_j$ et $c_{j_0} > p_{j_0}$, alors l'ordonnancement optimal est trivial.



Plan de l'exposé

Notre plate-forme



Notre problème

Le but

Minimiser :

- le *makespan*
- le flot maximal
- la somme des flots

Notre problème

Le but

Minimiser :

- le *makespan*
- le flot maximal
- la somme des flots

Et les 3 à la fois pour un problème à la volée, si possible !

Notre problème

Le but

Minimiser :

- le *makespan*
- le flot maximal
- la somme des flots

Et les 3 à la fois pour un problème à la volée, si possible !

C'est possible, *Round-Robin* le fait.

Notre problème

Le but

Minimiser :

- le *makespan*
- le flot maximal
- la somme des flots

Et les 3 à la fois pour un problème à la volée, si possible !

C'est possible, *Round-Robin* le fait.

Round-Robin

Allocation des tâches de façon cyclique :
il envoie la tâche i au processeur $i \ mod \ m$ dès qu'il peut.

L'ordonnanceur optimal : Round-Robin

Idée de la preuve

- Exhiber un algorithme optimal *ASAP* (*As Soon As Possible*) ayant un comportement proche de celui de *Round-Robin*,
- Montrer que *ASAP* et *Round-Robin* terminent l'exécution des tâches en même temps.

L'ordonnanceur optimal : Round-Robin

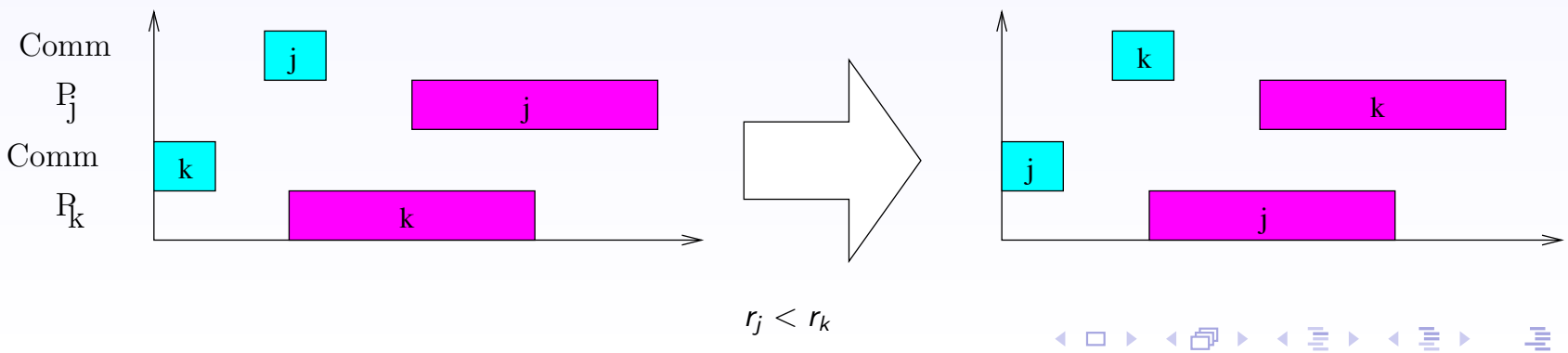
- Soit S un ordonnancement optimal pour minimiser $\sum(C_i - r_i)$

L'ordonnanceur optimal : Round-Robin

- Soit S un ordonnancement optimal pour minimiser $\sum(C_i - r_i)$
- Construisons un ordonnancement *ASAP* à partir de S, optimal lui aussi pour cette norme, qui :
  - envoie des tâches aux esclaves par ordre de date d'arrivée croissante

L'ordonnanceur optimal : Round-Robin

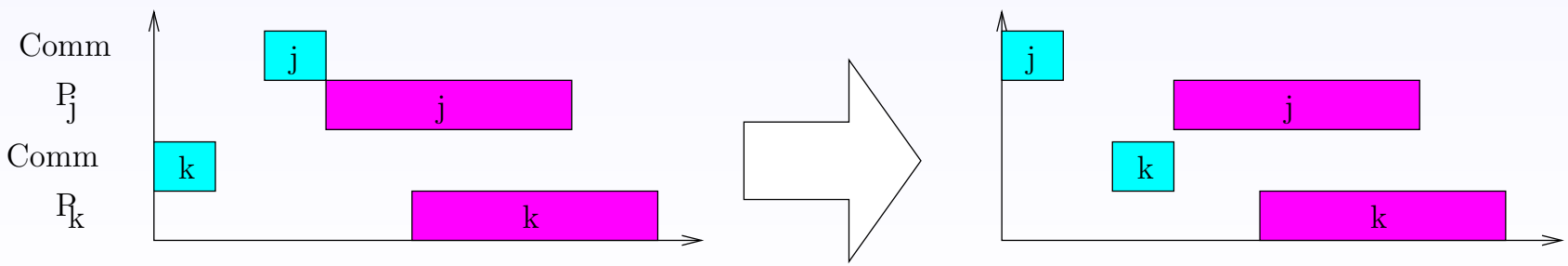
- Soit S un ordonnancement optimal pour minimiser $\sum(C_i - r_i)$
- Construisons un ordonnancement *ASAP* à partir de S, optimal lui aussi pour cette norme, qui :
  - envoie des tâches aux esclaves par ordre de date d'arrivée croissante



$r_j < r_k$

L'ordonnanceur optimal : Round-Robin

- Soit S un ordonnancement optimal pour minimiser $\sum(C_i - r_i)$
- Construisons un ordonnancement *ASAP* à partir de S, optimal lui aussi pour cette norme, qui :
  - envoie des tâches aux esclaves par ordre de date d'arrivée croissante



$r_j < r_k$

L'ordonnanceur optimal : Round-Robin

- Soit S un ordonnancement optimal pour minimiser $\sum(C_i - r_i)$
- Construisons un ordonnancement *ASAP* à partir de S, optimal lui aussi pour cette norme, qui :
  - envoie des tâches aux esclaves par ordre de date d'arrivée croissante
  - fait traiter les tâches par les esclaves par ordre de date d'arrivée croissante

L'ordonnanceur optimal : Round-Robin

- Soit S un ordonnancement optimal pour minimiser $\sum(C_i - r_i)$
- Construisons un ordonnancement *ASAP* à partir de S, optimal lui aussi pour cette norme, qui :
  - envoie des tâches aux esclaves par ordre de date d'arrivée croissante
  - fait traiter les tâches par les esclaves par ordre de date d'arrivée croissante
  - envoie des tâches aux esclaves dès que possible

L'ordonnanceur optimal : Round-Robin

- Soit S un ordonnancement optimal pour minimiser $\sum(C_i - r_i)$
- Construisons un ordonnancement *ASAP* à partir de S, optimal lui aussi pour cette norme, qui :
  - envoie des tâches aux esclaves par ordre de date d'arrivée croissante
  - fait traiter les tâches par les esclaves par ordre de date d'arrivée croissante
  - envoie des tâches aux esclaves dès que possible

On démontre par récurrence que *ASAP* et *Round-Robin* finissent leurs tâches au même instant.

L'ordonnanceur optimal : Round-Robin

- Soit S un ordonnancement optimal pour minimiser $\sum(C_i - r_i)$
- Construisons un ordonnancement *ASAP* à partir de S, optimal lui aussi pour cette norme, qui :
  - envoie des tâches aux esclaves par ordre de date d'arrivée croissante
  - fait traiter les tâches par les esclaves par ordre de date d'arrivée croissante
  - envoie des tâches aux esclaves dès que possible

On démontre par récurrence que *ASAP* et *Round-Robin* finissent leurs tâches au même instant.

Raisonnement identique en prenant pour S un algorithme optimal pour minimiser le *makespan* et le flot maximal.

L'ordonnanceur optimal : Round-Robin

Conclusion

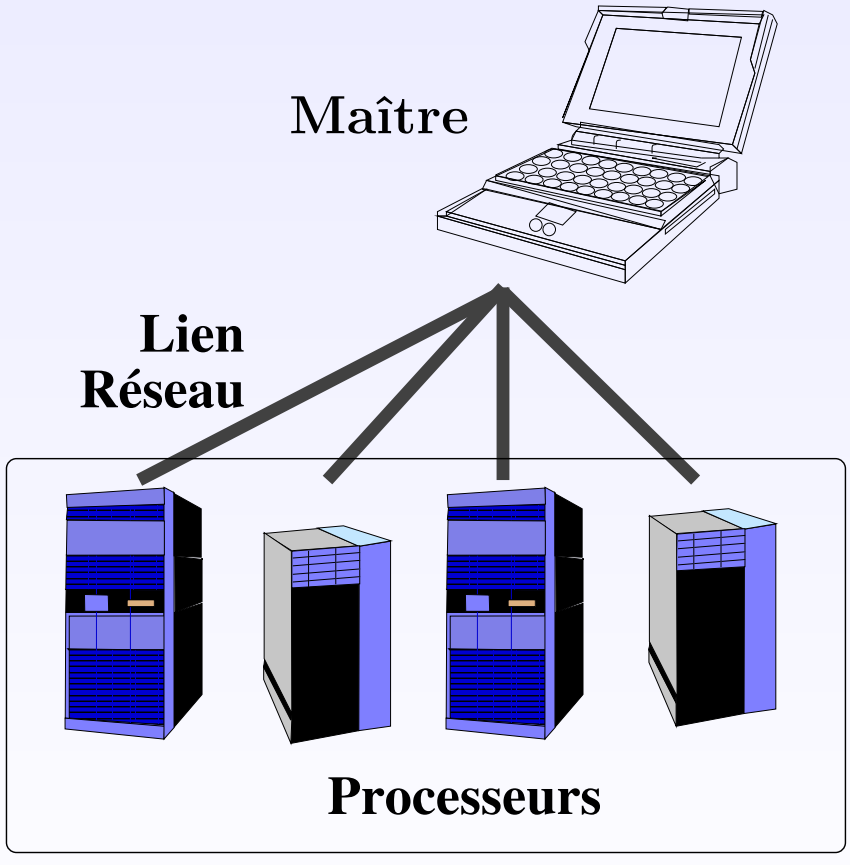
Au final, *Round-Robin* est un algorithme optimal pour minimiser **à la fois**

- le *makespan*,
- la somme des flots,
- et le flot maximal,

pour un problème à la volée avec dates d'arrivée.

Plan de l'exposé

La plate-forme



L'ordonnanceur optimal ...

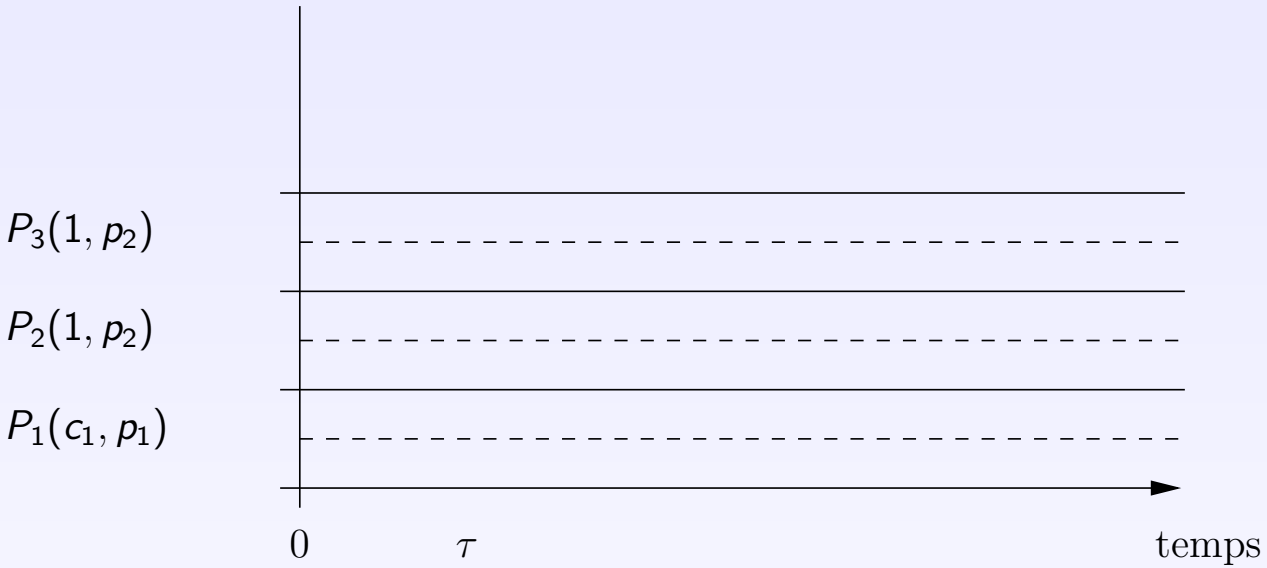
n'existe pas ! !

pour aucune des 3 normes usuelles, pour le problème à la volée, du moins.

Bornes de compétitivité sur les algorithmes on-lines

Platform type	Objective function		
	Makespan	Max-flow	Sum-flow
Communication homogeneous	$\frac{5}{4} = 1.250$	$\frac{5-\sqrt{7}}{2} \approx 1.177$	$\frac{2+4\sqrt{2}}{7} \approx 1.093$
Computation homogeneous	$\frac{6}{5} = 1.200$	$\frac{5}{4} = 1.250$	$\frac{23}{22} \approx 1.045$
Heterogeneous	$\frac{1+\sqrt{3}}{2} \approx 1.366$	$\sqrt{2} \approx 1.414$	$\frac{\sqrt{13}-1}{2} \approx 1.302$

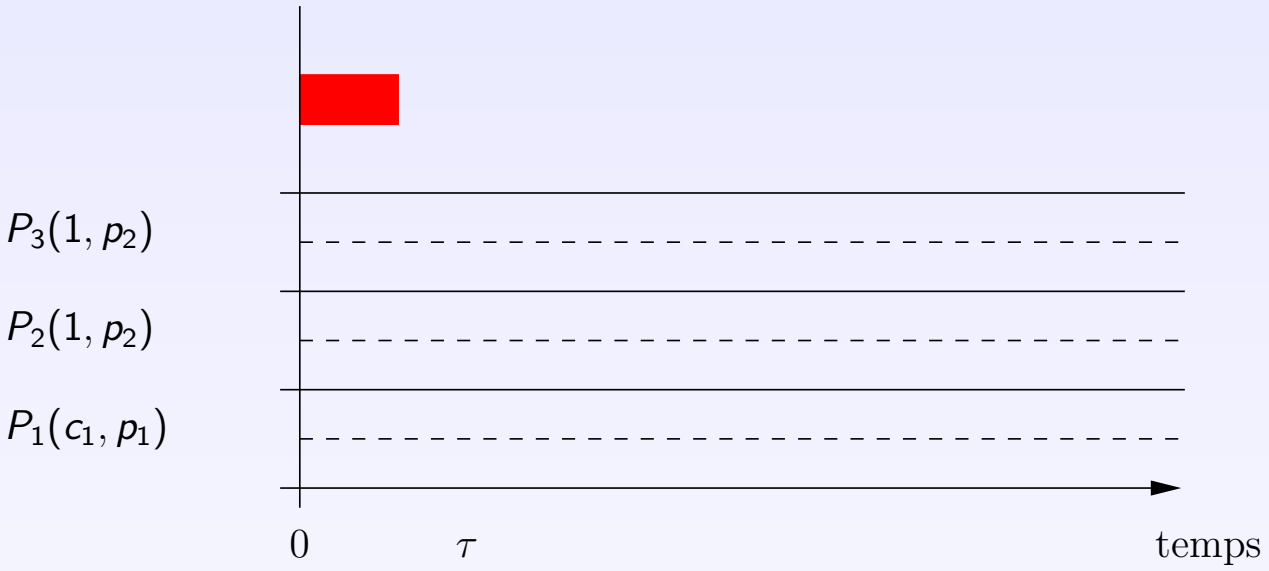
Obtention de la borne inférieure de compétitivité (1)



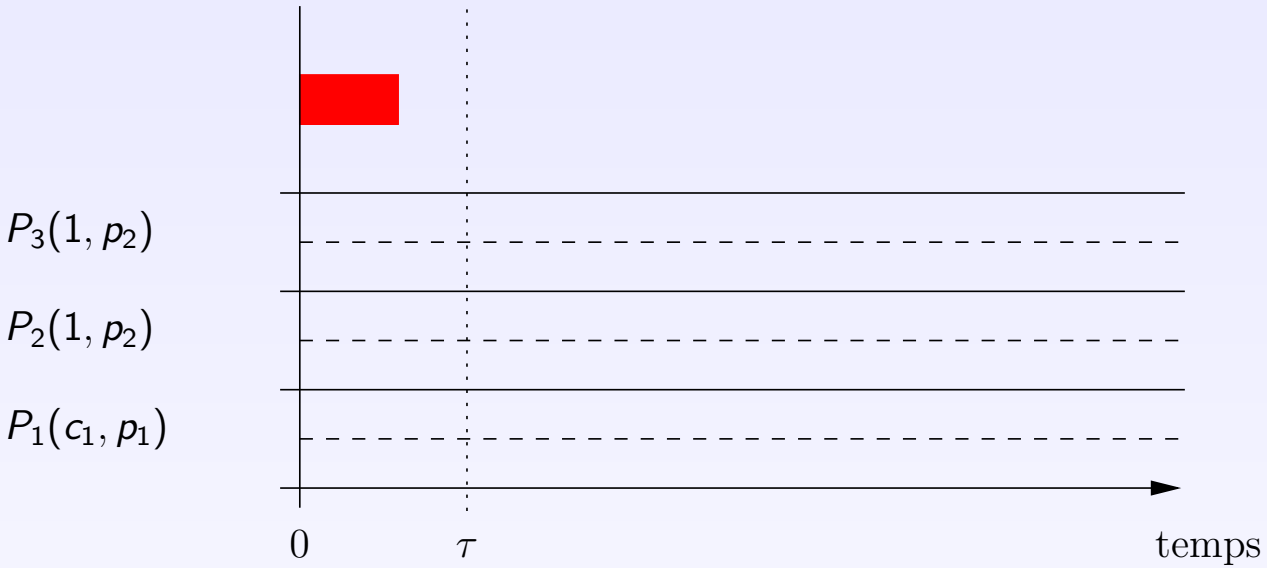
Idée :

- un processeur rapide avec communications lentes $(c_1 > 1)$;
- deux processeurs identiques et lents, mais avec communications rapides ;
- si une seule tâche, il faut choisir le processeur rapide $(c_1 + p_1 < 1 + p_2)$.

Obtention de la borne inférieure de compétitivité (1)

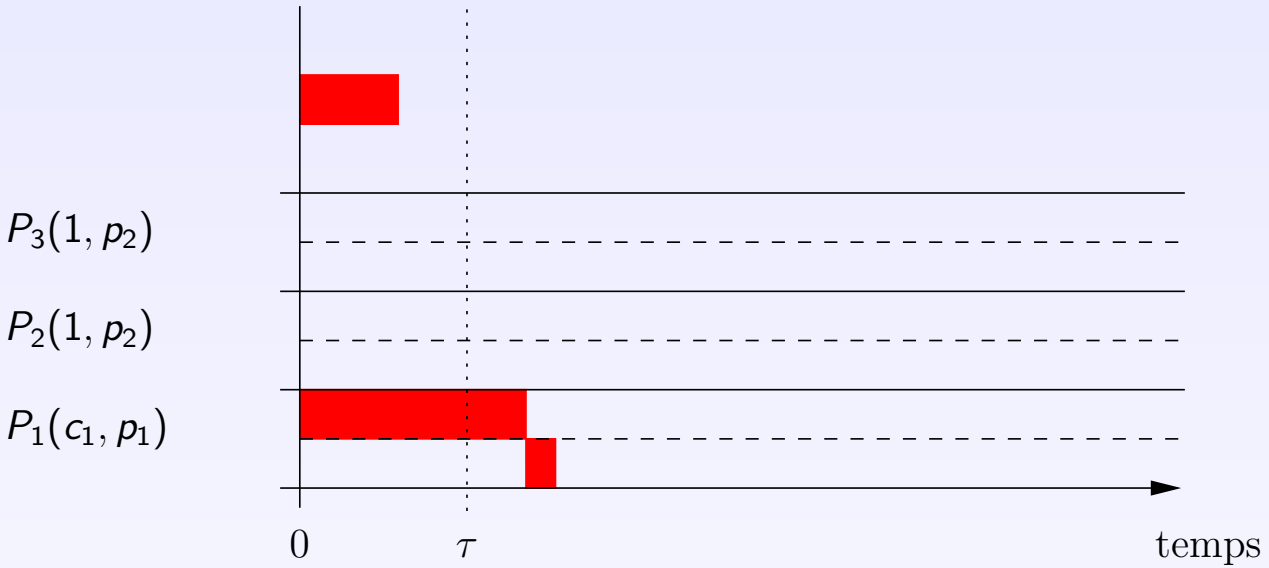


Obtention de la borne inférieure de compétitivité (1)



On regarde au temps $\tau \geq 1$ ce qui s'est passé. Trois possibilités :
On oblige l'algorithme à exécuter la première tâche sur P_1.

Obtention de la borne inférieure de compétitivité (1)

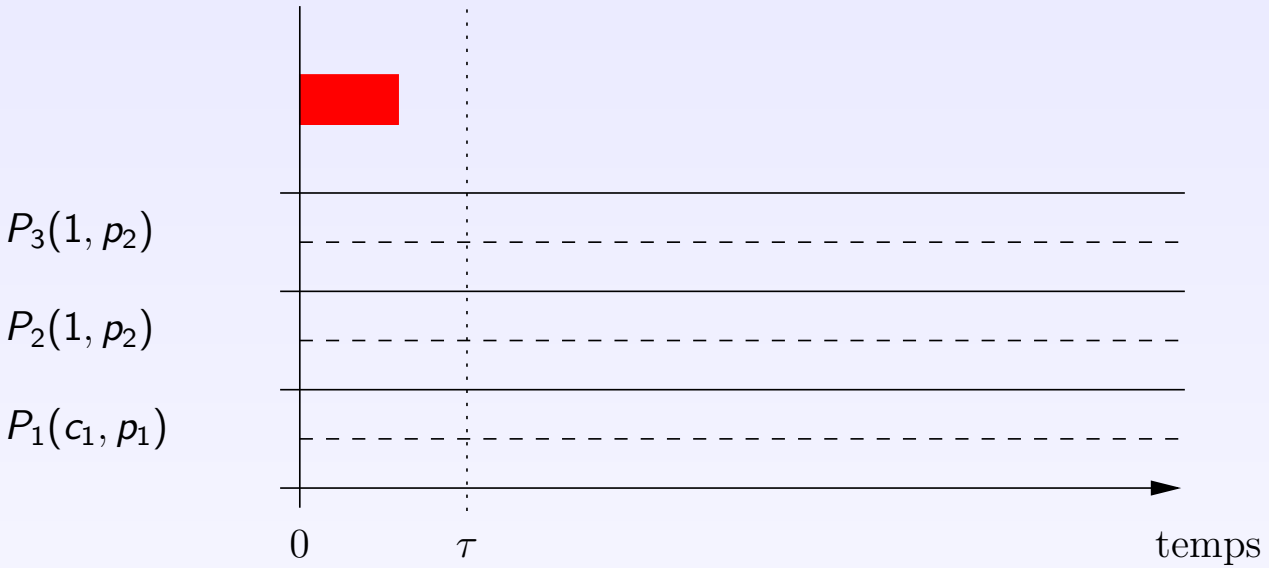


On regarde au temps $\tau \geq 1$ ce qui s'est passé. Trois possibilités :

1. Optimal : tâche sur P_1, max-flot $\geq c_1 + p_1$.

On oblige l'algorithme à exécuter la première tâche sur P_1.

Obtention de la borne inférieure de compétitivité (1)

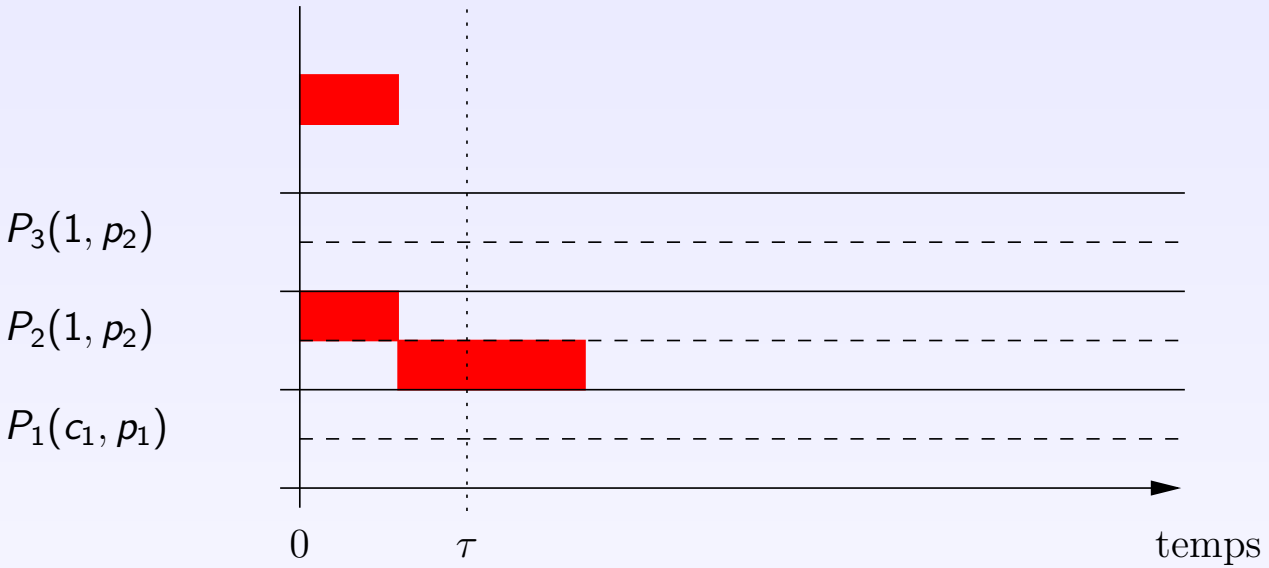


On regarde au temps $\tau \geq 1$ ce qui s'est passé. Trois possibilités :

1. Optimal : tâche sur P_1, max-flot $\geq c_1 + p_1$.
2. Rien fait : max-flot $\geq \tau + c_1 + p_1$, ratio $\geq \frac{\tau + c_1 + p_1}{c_1 + p_1}$.

On oblige l'algorithme à exécuter la première tâche sur P_1.

Obtention de la borne inférieure de compétitivité (1)

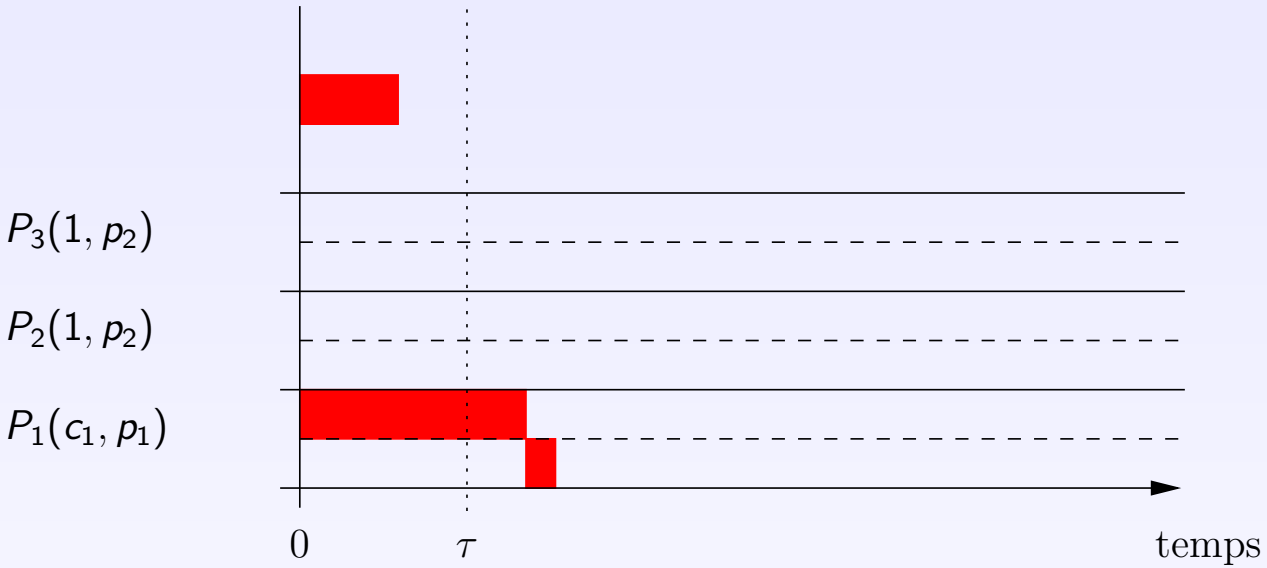


On regarde au temps $\tau \geq 1$ ce qui s'est passé. Trois possibilités :

1. Optimal : tâche sur P_1, max-flot $\geq c_1 + p_1$.
2. Rien fait : max-flot $\geq \tau + c_1 + p_1$, ratio $\geq \frac{\tau + c_1 + p_1}{c_1 + p_1}$.
3. Tâche envoyée sur P_2, max-flot $\geq 1 + p_2$. Ratio $\geq \frac{1 + p_2}{c_1 + p_1}$.

On oblige l'algorithme à exécuter la première tâche sur P_1.

Obtention de la borne inférieure de compétitivité (1)

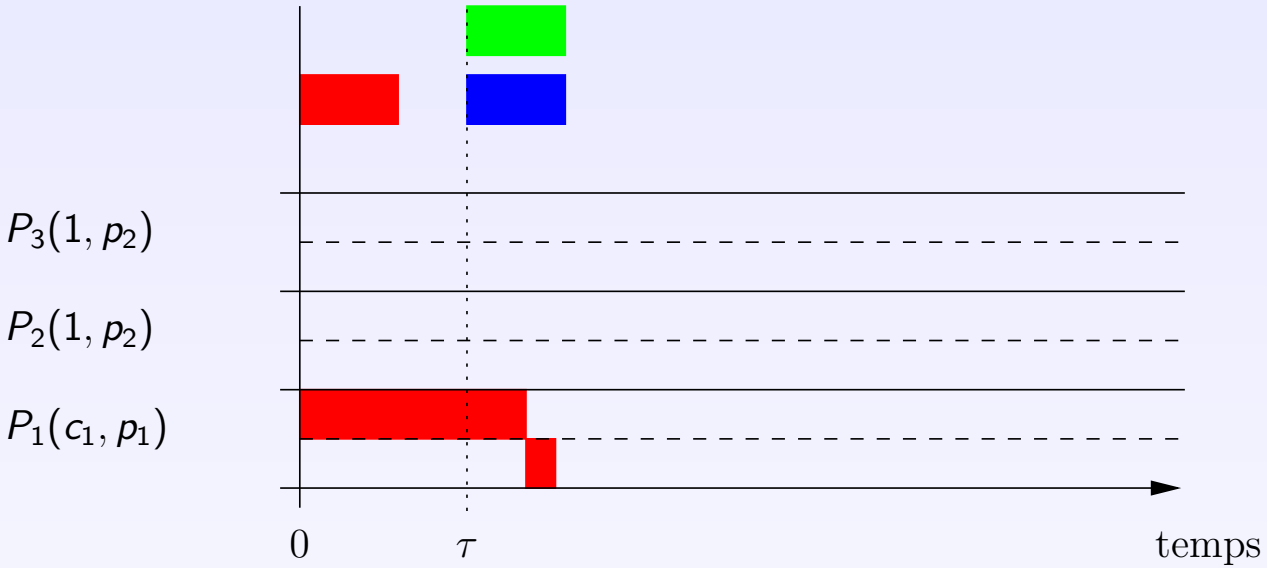


On regarde au temps $\tau \geq 1$ ce qui s'est passé. On va choisir τ, c_1, p_1 et p_2 pour avoir :

$$\min \left\{ \frac{1 + p_2}{c_1 + p_1}, \frac{\tau + c_1 + p_1}{c_1 + p_1} \right\} \geq \rho$$

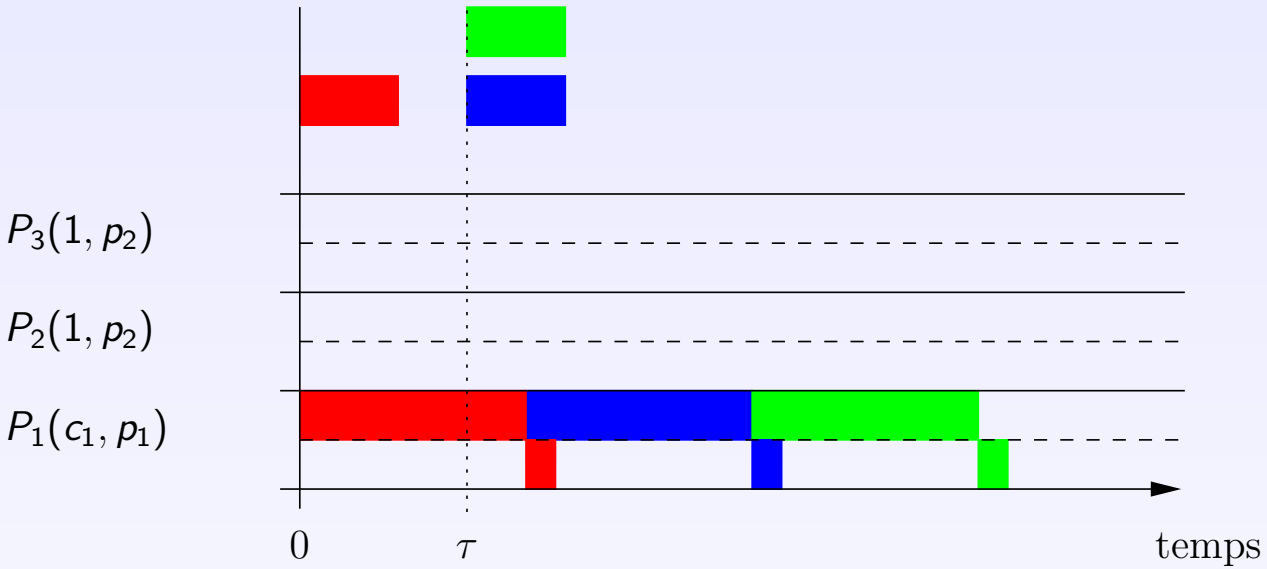
On oblige l'algorithme à exécuter la première tâche sur P_1.

Obtention de la borne inférieure de compétitivité (1)



Au temps τ on envoie deux nouvelles tâches.
On considère tous les cas possibles.

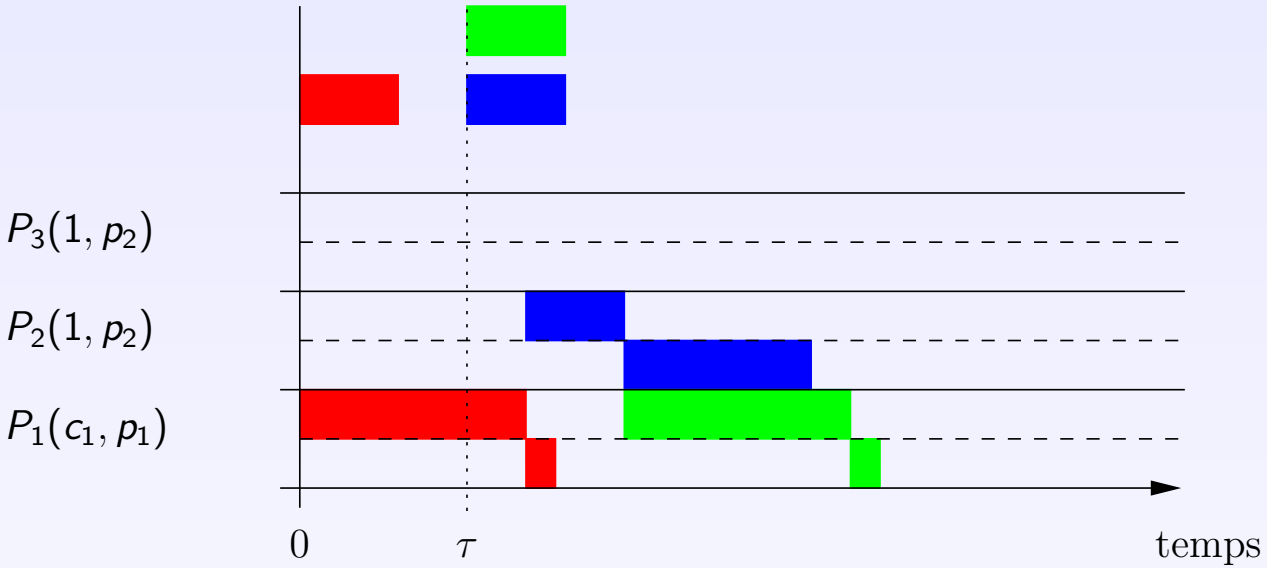
Obtention de la borne inférieure de compétitivité (1)



Au temps τ on envoie deux nouvelles tâches.
Les deux tâches sont exécutées sur P_1 :

$$\max\{c_1 + p_1,$$
$$\max\{\max\{c_1, \tau\} + c_1 + p_1, c_1 + 2p_1\} - \tau,$$
$$\max\{\max\{c_1, \tau\} + c_1 + p_1 + \max\{c_1, p_1\}, c_1 + 3p_1\} - \tau\}$$

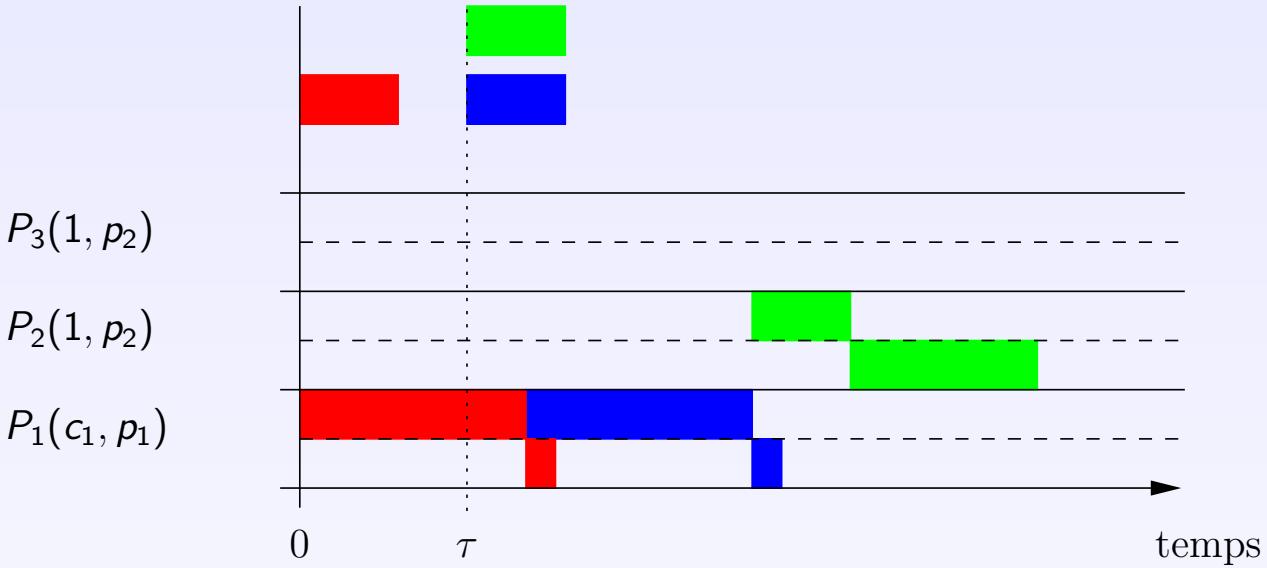
Obtention de la borne inférieure de compétitivité (1)



Au temps τ on envoie deux nouvelles tâches.
La première des deux tâches est exécutée sur P_2 (ou p_2), et l'autre sur P_1.

$$\max\{c_1 + p_1,$$
$$(\max\{c_1, \tau\} + c_2 + p_2) - \tau,$$
$$\max\{\max\{c_1, \tau\} + c_2 + c_1 + p_1, c_1 + 2p_1\} - \tau\}$$

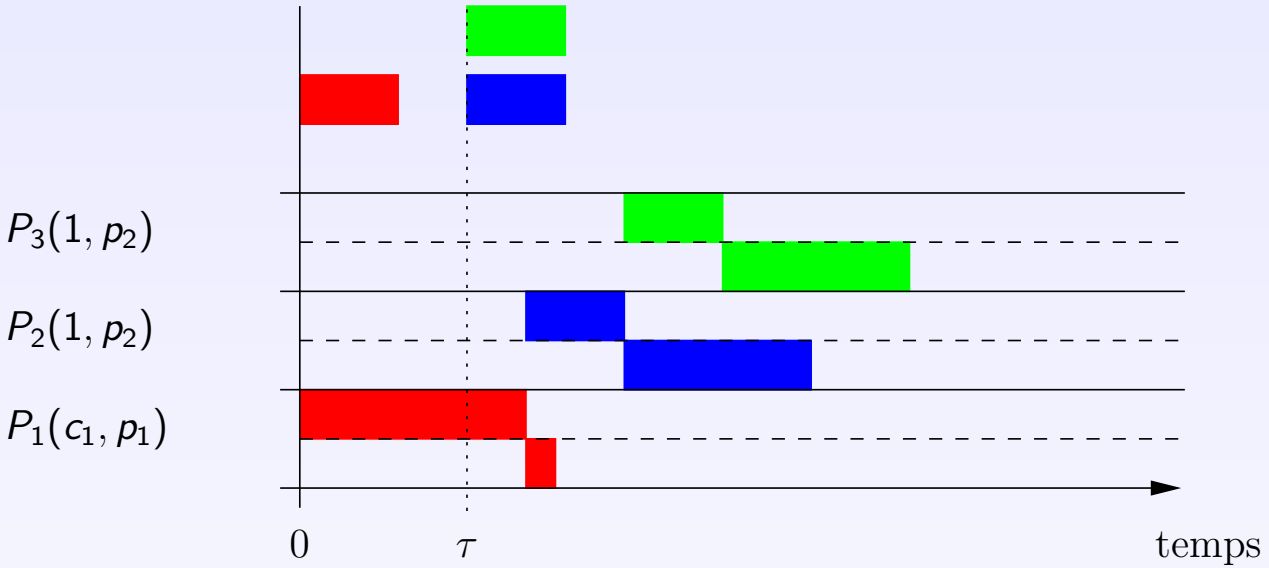
Obtention de la borne inférieure de compétitivité (1)



Au temps τ on envoie deux nouvelles tâches.
La première des deux tâches est exécutée sur P_1, et l'autre sur P_2 (ou p_2).

$$\max\{c_1 + p_1,$$
$$\max\{\max\{c_1, \tau\} + c_1 + p_1, c_1 + 2p_1\} - \tau,$$
$$(\max\{c_1, \tau\} + c_1 + c_2 + p_2) - \tau\}$$

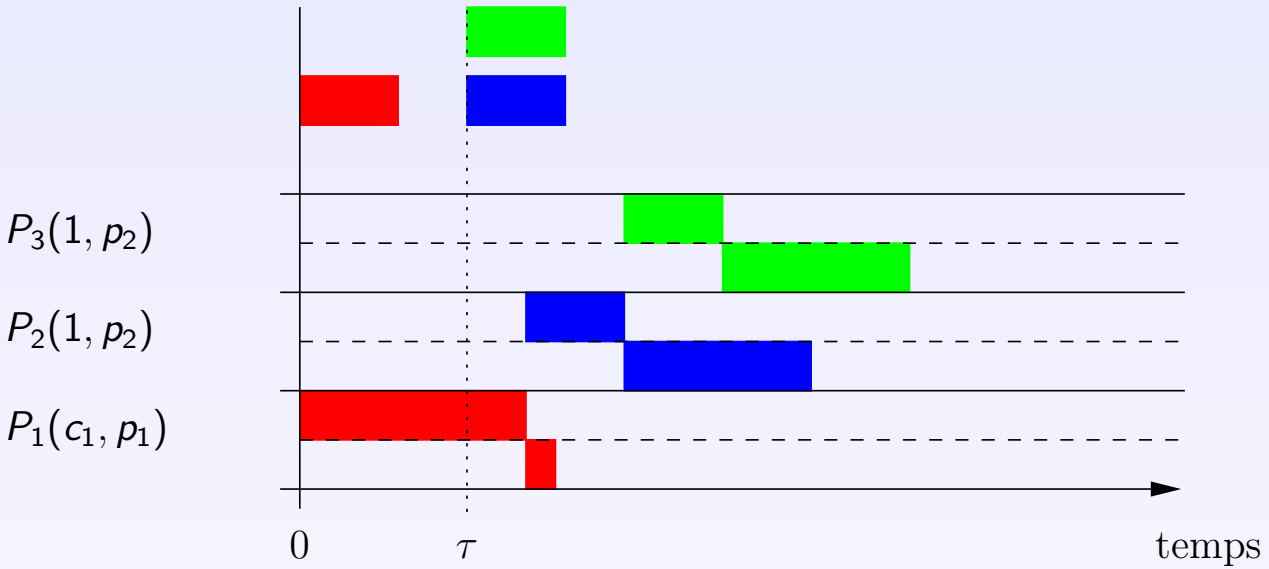
Obtention de la borne inférieure de compétitivité (1)



Au temps τ on envoie deux nouvelles tâches.
Une des deux tâches est exécutée sur P_2 et l'autre sur p_2.

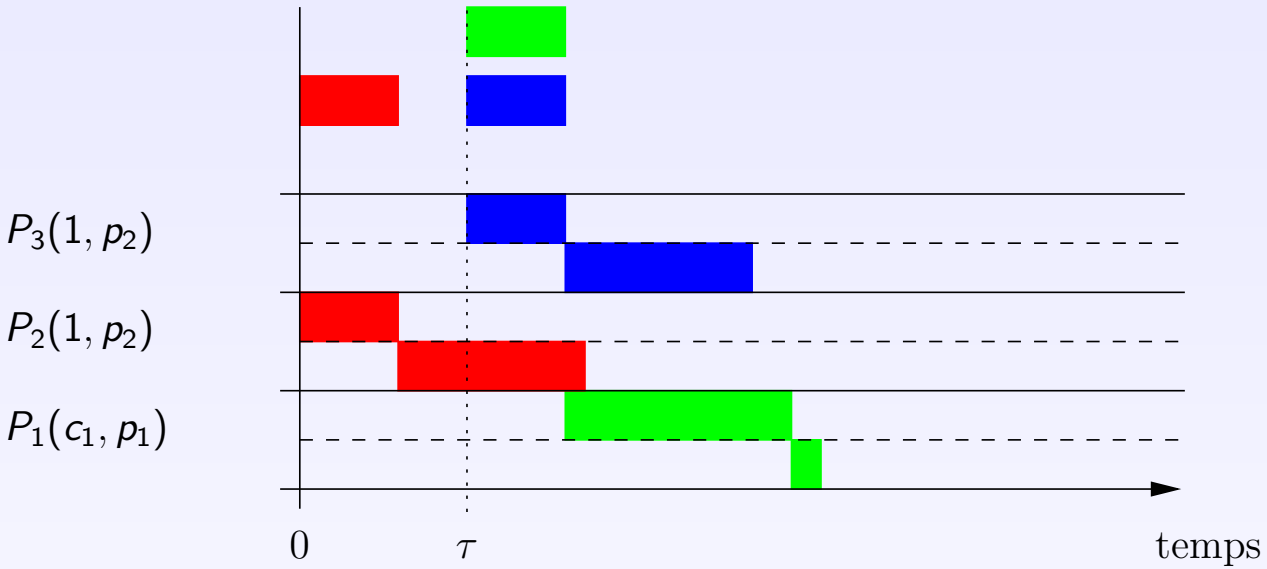
$$\max\{c_1+p_1, (\max\{c_1,\tau\}+c_2+p_2)-\tau, (\max\{c_1,\tau\}+c_2+c_2+p_2)-\tau\}$$

Obtention de la borne inférieure de compétitivité (1)



Au temps τ on envoie deux nouvelles tâches.
Le cas où les deux tâches sont exécutées sur P_2 (ou toutes les deux p_2) est plus mauvais que le précédent, on n'a donc pas besoin de l'étudier.

Obtention de la borne inférieure de compétitivité (1)



Au temps τ on envoie deux nouvelles tâches.
L'optimal (souhaité) : la 1re tâche sur P_2, la 2e sur p_2 et la 3e sur P_1.

$$\max\{c_2+p_2, (\max\{c_2,\tau\}+c_2+p_2)-\tau, (\max\{c_2,\tau\}+c_2+c_1+p_1)-\tau\}$$

Obtention de la borne inférieure de compétitivité (2)

Borne inférieure sur la compétitivité :

$$\min \begin{cases} \frac{\tau+c_1+p_1}{c_1+p_1}, \\ \frac{1+p_2}{c_1+p_1}, \\ \min \begin{cases} \max\{c_1+p_1, \max\{\max\{c_1,\tau\}+c_1+p_1, c_1+2p_1\}-\tau, \max\{\max\{c_1,\tau\}+c_1+p_1+\max\{c_1,p_1\}, c_1+3p_1\} \\ \max\{c_1+p_1, (\max\{c_1,\tau\}+c_2+p_2)-\tau, \max\{\max\{c_1,\tau\}+c_2+c_1+p_1, c_1+2p_1\}-\tau\} \\ \max\{c_1+p_1, \max\{\max\{c_1,\tau\}+c_1+p_1, c_1+2p_1\}-\tau, (\max\{c_1,\tau\}+c_1+c_2+p_2)-\tau\} \\ \max\{c_1+p_1, (\max\{c_1,\tau\}+c_2+p_2)-\tau, (\max\{c_1,\tau\}+c_2+c_2+p_2)-\tau\} \end{cases} \end{cases}$$
$$\overline{\max\{c_2+p_2, (\max\{c_2,\tau\}+c_2+p_2)-\tau, (\max\{c_2,\tau\}+c_2+c_1+p_1)-\tau\}}$$

Problème : trouver τ, c_1, p_1 et p_2 (car $c_2 = 1$) qui maximisent cette borne inférieure.

Contraintes : $c_1 + p_1 < p_2$.

Obtention de la borne inférieure de compétitivité (3)

1. Résolution numérique

Obtention de la borne inférieure de compétitivité (3)

1. Résolution numérique
2. Caractérisation de la forme de l'optimal : $\tau < c_1$, $p_1 = 0$, etc.

Obtention de la borne inférieure de compétitivité (3)

1. Résolution numérique
2. Caractérisation de la forme de l'optimal : $\tau < c_1$, $p_1 = 0$, etc.
3. Nouveau système :

$$\min \left\{ \begin{array}{l} \frac{\tau+c_1}{c_1}, \\ \frac{1+p_2}{c_1}, \\ \frac{\min \left\{ \begin{array}{l} 3c_1 - \tau, \\ c_1 + 1 - \tau + p_2, \\ 2c_1 - \tau + 1 + p_2 \\ c_1 + 2 + p_2 - \tau \end{array} \right.}{1+p_2} \end{array} \right. = \min \left\{ \begin{array}{l} \frac{\tau+c_1}{c_1}, \\ \frac{1+p_2}{c_1}, \\ \frac{c_1+1-\tau+p_2,}{1+p_2} \end{array} \right.$$

Obtention de la borne inférieure de compétitivité (3)

1. Résolution numérique
2. Caractérisation de la forme de l'optimal : $\tau < c_1$, $p_1 = 0$, etc.
3. Nouveau système :

$$\min \left\{ \begin{array}{l} \frac{\tau+c_1}{c_1}, \\ \\ \frac{1+p_2}{c_1}, \\ \\ \min \left\{ \begin{array}{l} 3c_1 - \tau, \\ c_1 + 1 - \tau + p_2, \\ 2c_1 - \tau + 1 + p_2 \\ c_1 + 2 + p_2 - \tau \end{array} \right. \\ \hline 1+p_2 \end{array} \right. = \min \left\{ \begin{array}{l} \frac{\tau+c_1}{c_1}, \\ \\ \frac{1+p_2}{c_1}, \\ \\ \frac{c_1+1-\tau+p_2}{1+p_2} \end{array} \right.$$

4. Solution : $c_1 = 2(1+\sqrt{2})$, $p_2 = \sqrt{2}c_1 - 1$, $\tau = 2$, $\rho = \sqrt{2}$.

Minimisation online du makespan

On peut montrer que si un ordonnanceur X ne connaît :

- ni le nombre de tâches à venir,
- ni leurs dates d'arrivée,

il ne peut pas trouver l'ordonnancement optimal pour minimiser une des trois fonctions objectives usuelles dans tous les cas.

Minimisation online du makespan

On peut montrer que si un ordonnanceur X ne connaît :

- ni le nombre de tâches à venir,
- ni leurs dates d'arrivée,

il ne peut pas trouver l'ordonnancement optimal pour minimiser une des trois fonctions objectives usuelles dans tous les cas.

On obtient même une borne inférieure de compétitivité de $\frac{5}{4}$ pour le *makespan*.

Minimisation online du makespan

On peut montrer que si un ordonnanceur X ne connaît :

- ni le nombre de tâches à venir,
- ni leurs dates d'arrivée,

il ne peut pas trouver l'ordonnancement optimal pour minimiser une des trois fonctions objectives usuelles dans tous les cas.

On obtient même une borne inférieure de compétitivité de $\frac{5}{4}$ pour le *makespan*.

Donnons lui donc plus de connaissances.

Scheduling the Last Jobs First

Algorithme optimal...

lorsque l'on connaît le nombre total de tâches, pour minimiser le *makespan*.

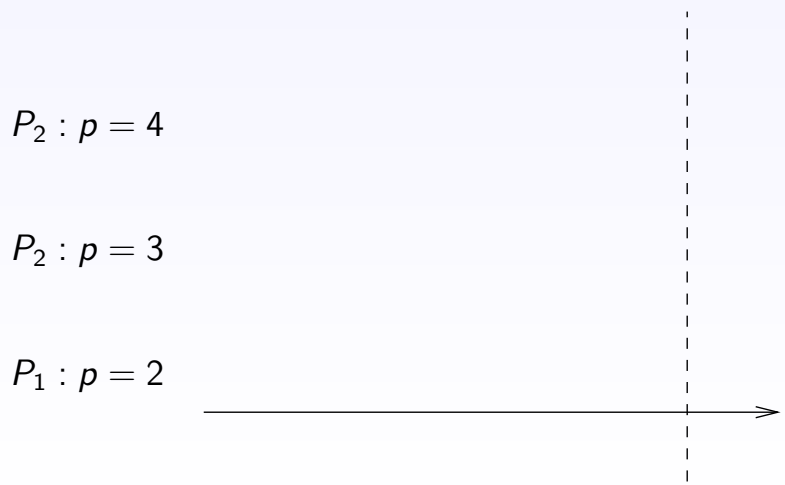
Scheduling the Last Jobs First

Principe de fonctionnement

Scheduling the Last Jobs First

Principe de fonctionnement

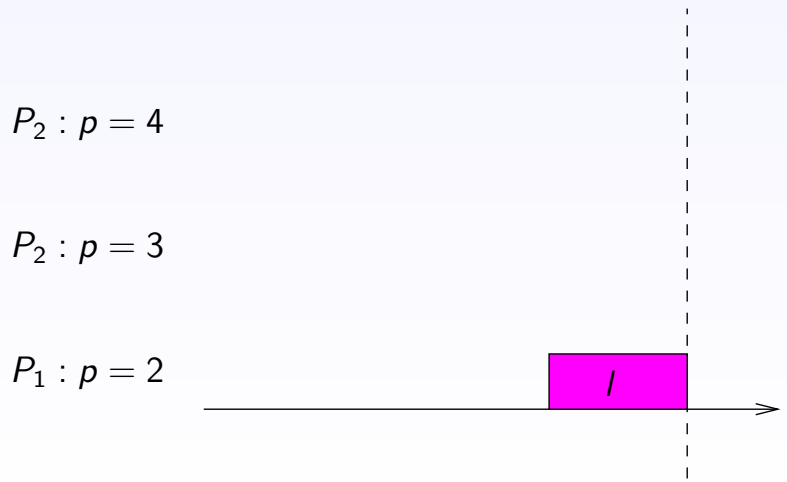
- Fixer une date virtuelle de fin



Scheduling the Last Jobs First

Principe de fonctionnement

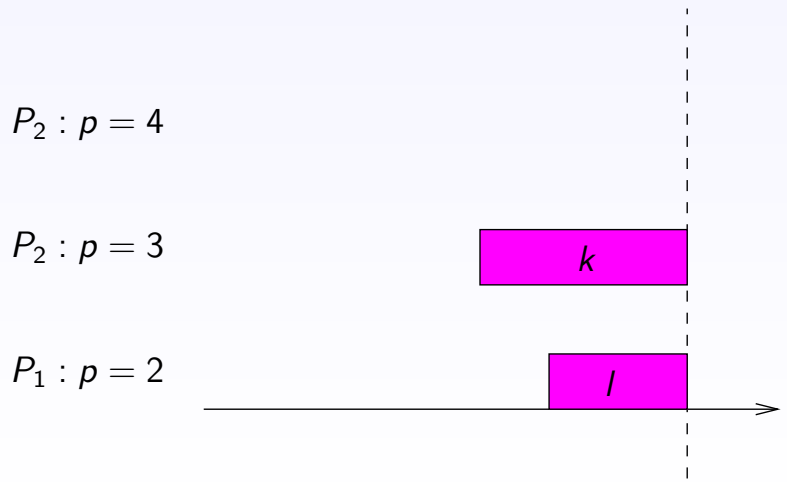
- Fixer une date virtuelle de fin
- Allouer les tâches au plus tard, en tenant compte de l'ordonnancement déjà défini



Scheduling the Last Jobs First

Principe de fonctionnement

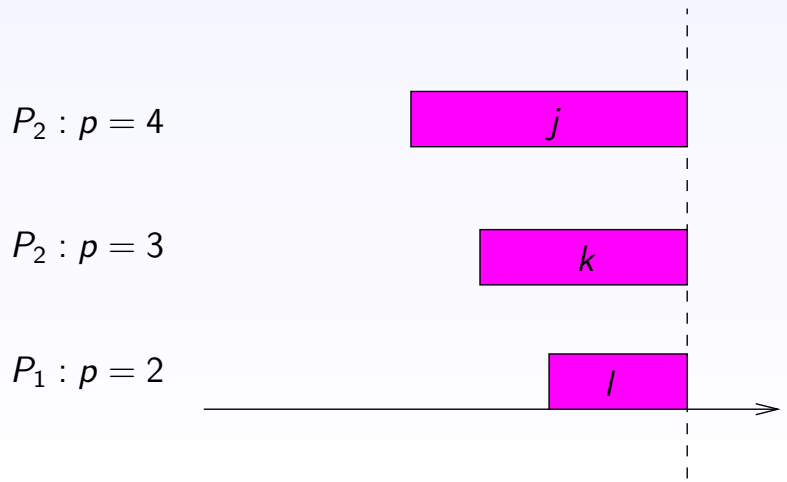
- Fixer une date virtuelle de fin
- Allouer les tâches au plus tard, en tenant compte de l'ordonnancement déjà défini



Scheduling the Last Jobs First

Principe de fonctionnement

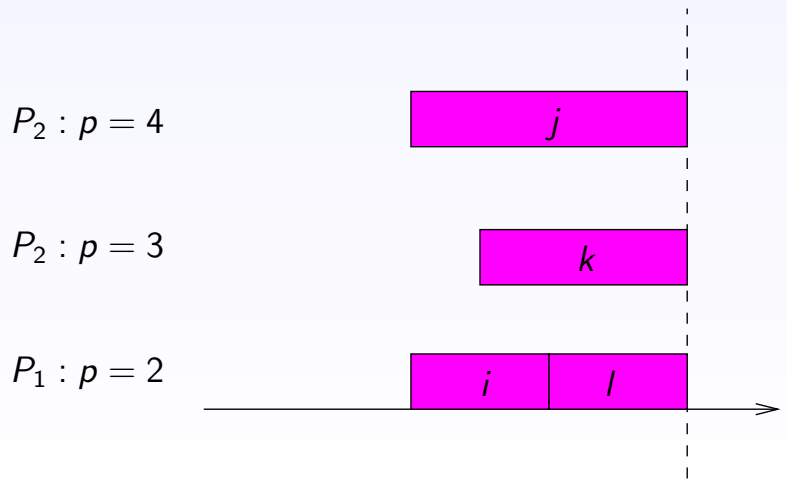
- Fixer une date virtuelle de fin
- Allouer les tâches au plus tard, en tenant compte de l'ordonnancement déjà défini



Scheduling the Last Jobs First

Principe de fonctionnement

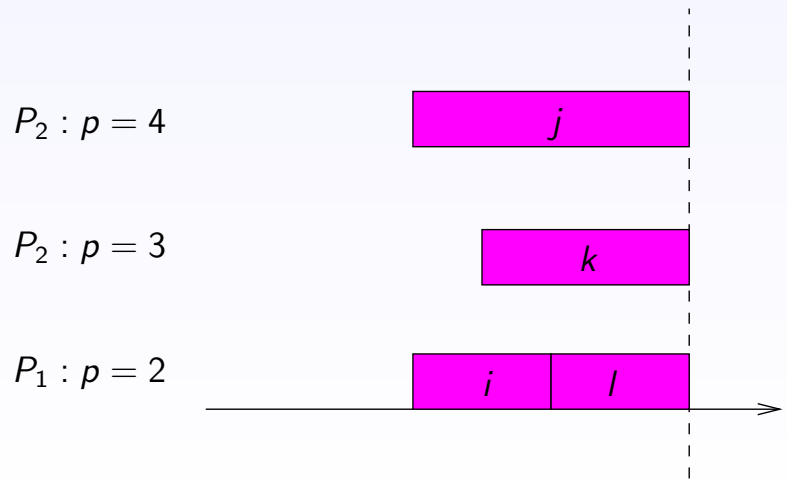
- Fixer une date virtuelle de fin
- Allouer les tâches au plus tard, en tenant compte de l'ordonnancement déjà défini



Scheduling the Last Jobs First

Principe de fonctionnement

- Fixer une date virtuelle de fin
- Allouer les tâches au plus tard, en tenant compte de l'ordonnancement déjà défini
- Mémoriser l'attribution des tâches aux processeurs



Scheduling the Last Jobs First

Principe de fonctionnement

- Fixer une date virtuelle de fin
- Allouer les tâches au plus tard, en tenant compte de l'ordonnancement déjà défini
- Mémoriser l'attribution des tâches aux processeurs
- Distribuer les tâches au plus tôt