Ordonnançement temps réel pour tâches périodiques Ordonnancement temps réel

- Partant d'un ensemble de tâches Γ
  - Les tâches ayant:
- Un temps d'arrivée
- Un temps d'exécution
- **Une échéance**
- L'ordonnancement vise à trouver un moyen d'allouer le processeur aux différentes tâches de manière à respecter les contraintes de temps

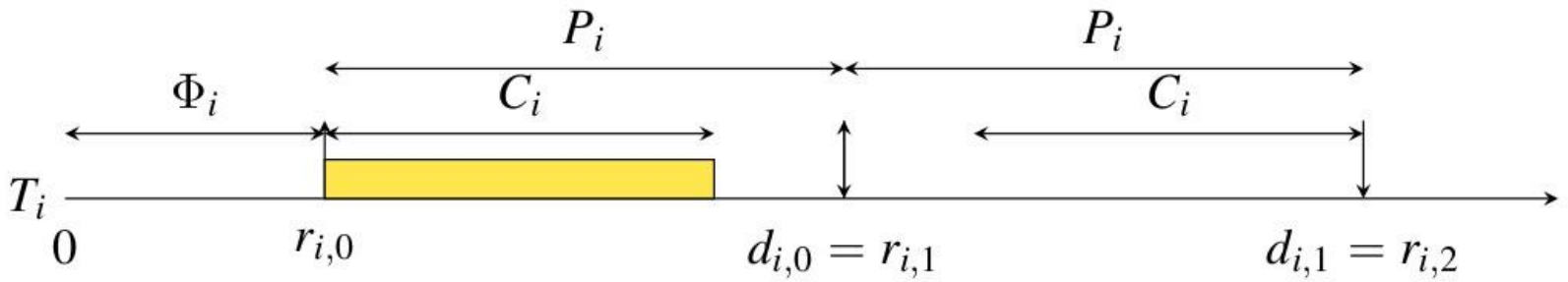
Tâches périodiques

Variable	Description
Γ	Un ensemble de tâches périodiques
τ_i	Une tâche périodique générique
$\tau_{i,j}$	La j$^{\text{ème}}$ instance de la tâche périodique τ_i
$r_{i,j}$	Le temps d'arrivée de la j$^{\text{ème}}$ instance de la tâche périodique τ_i
Φ_i	Le déphasage de la tâche τ_i; il s'agit du temps d'arrivée de $\tau_{i,0}$ ($\Phi_i = r_{i,0}$)
D_i	Echéance relative de la tâche τ_i
$d_{i,j}$	Echéance absolue de la j$^{\text{ème}}$ instance de la tâche τ_i ($d_{i,j} = \Phi_i + (j-1)P_i + D_i$)
$s_{i,j}$	Temps de début d'exécution de la j$^{\text{ème}}$ instance de la tâche τ_i
$f_{i,j}$	Temps de fin d'exécution de la j$^{\text{ème}}$ instance de la tâche τ_i

Ordonnancement temps réel

- Pour les tâches à échéances sur requête, nous pouvons calculer les temps d'arrivée des tâches, ainsi que leurs échéances respectives de cette manière:

$$r_{i,j} = \Phi_i + (j-1)P_i$$
$$d_{i,j} = r_{i,j} + P_i = \Phi_i + jP_i$$



Hypothèse

- Les instances d’une tâche périodique sont activées avec une période constante
- Toutes les instances d’une tâche ont le même pire temps d’exécution C_i
- Toutes les instances d’une tâche ont la même échéance relative D_i
- Toutes les tâches sont indépendantes. Il n’y a pas de dépendances entre tâches
- Une tâche ne peut se suspendre elle-même
- La surcharge liée aux opérations du noyau est négligée

Période d’étude

- Les ordonnancements sont dédiés aux ensembles de tâches périodiques.
- Alors le fonctionnement complet du système est donc cyclique.
- L’algorithme d’ordonnancement doit trouver une séquence de tâches, valable sur toute la durée de fonctionnement du système.
- Cette période de longueur L est appelée période d’étude, ou période de base.

Période d'étude

$$L = [0, PPCM(P_i)]$$

Où le *PPCM* dénote le plus petit multiple commun des périodes des tâches.

- Si les tâches sont ordonnançables sur la période d'étude, nous pouvons garantir qu'elles le seront pour un temps infini.
- Pour des tâches asynchrones, ne débutant donc pas au même instant, la période d'étude est :

Période d'étude

$$L = [\min\{r_{i,0}\}, 2 \times PPCM(P_i) + \max\{r_{i,0}]\}$$

- en présence de tâches apériodiques, la période d'étude devient :

$$L = [\min\{r_{i,0}\}, 2 \times PPCM(P_i) + \max\{r_{i,0}, r_j + D_j\}]$$

Où r_j est la date d'activation de la tâche apériodique T_j , et où D_i est son échéance relative.

Exemple

Tâche	Coût	Période
T_1	10	25
T_2	5	50
T_3	30	100

- Pour une tâche τ_i , le facteur d'occupation est défini par:

Taux d'utilisation du processeur

$$u_i = \frac{C_i}{P_i}$$

- Pour l'ensemble des tâches Γ, le facteur d'utilisation est dès lors:

$$U = \sum_{i=1}^{n} \frac{C_i}{P_i}$$

- Une condition nécessaire (mais pas suffisante) pour qu'un ensemble de tâches soit ordonnançable:

Taux d'utilisation du processeur

$$U = \sum_{i=1}^{n} \frac{C_i}{P_i} \leq 1$$

- Mais il dépend des caractéristiques des tâches et de l'algorithme appliqué
- Il existe un $U_{ub}(\Gamma, A)$ (Upper Bound) au-delà duquel les tâches ne sont pas ordonnançables
- Lorsque $U = U_{ub}(\Gamma, A)$, l'ensemble Γ utilise entièrement le processeur
- L'augmentation du temps d'exécution d'une des tâches rend l'ensemble non ordonnançable

Taux d'utilisation du processeur

- Pour un algorithme A donné, il existe une valeur minimale de U_{ub} (Least Upper Bound)

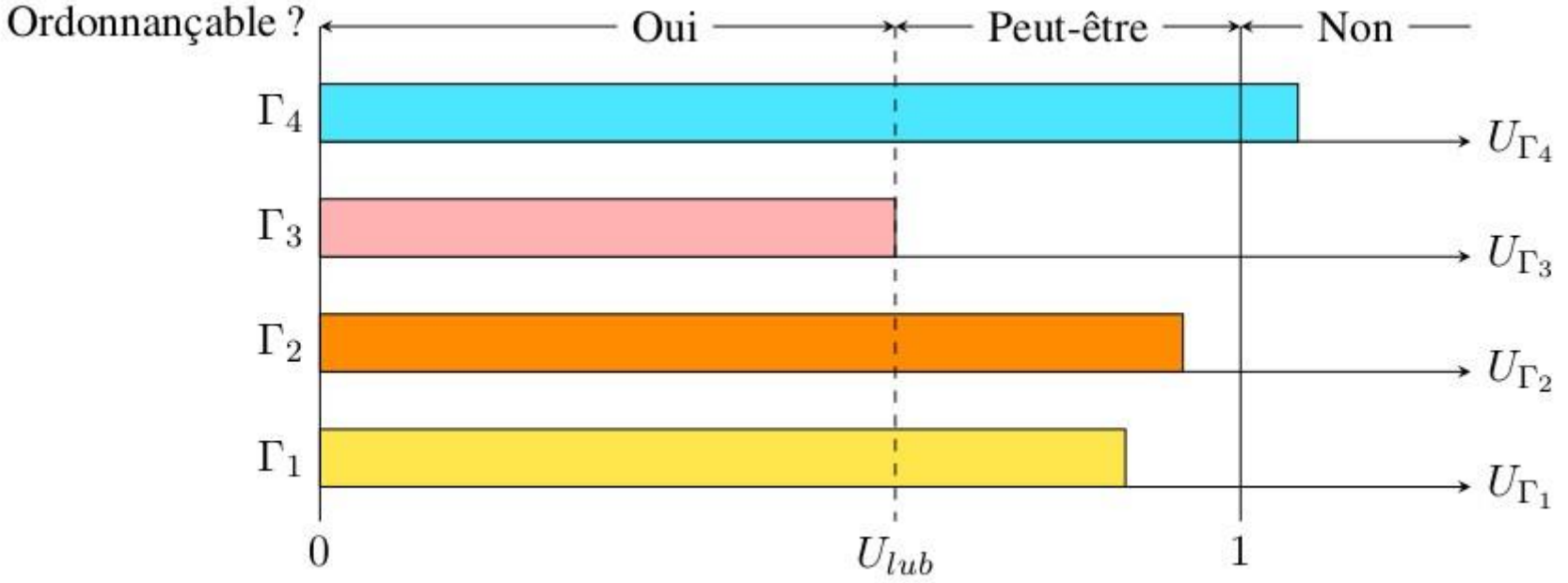
$$U_{lub}(A) = \min_{\Gamma} U_{ub}(\Gamma, A)$$

- De ce fait, un ensemble de tâches Γ est ordonnançable si son facteur d'utilisation du processeur est inférieur à cette limite minimale

Taux d’utilisation du processeur

$$U_\Gamma = \sum_{i=1}^{n} \frac{C_i}{P_i} \leq U_{lub}(A) = \min_{\Gamma} U_{ub}(\Gamma, A) \Rightarrow \Gamma \text{est ordonnançable}$$

Taux d'utilisation du processeur



Rate Monotonic

- L’algorithme Rate Monotonic est un algorithme statique, applicable à un ensemble de tâches à échéance sur requête, où les priorités des tâches sont fixes et décidée avant le lancement du système.
- Plus une tâche a une petite période d’activation, plus sa priorité sera haute.

Analyse d’ordonnançabilité (Least Upper Bound)

- Une condition suffisante

$$\sum_{i=1}^{n} \frac{C_i}{P_i} \leq U_{lub_{RM}}(n) = n(2^{\frac{1}{n}} - 1)$$

n	U_{lub}
1	1.000
2	0.828
3	0.780
4	0.757
5	0.743

d'ordonnançabilité est :

- Nous pouvons calculer U_{lub} pour de grandes valeurs de n :

$$U_{lub} = \lim_{n \to \infty} n(2^{\frac{1}{n}} - 1) = ln2 \simeq 0.69$$

Analyse d’ordonnançabilité (Hyperbolic Bound)

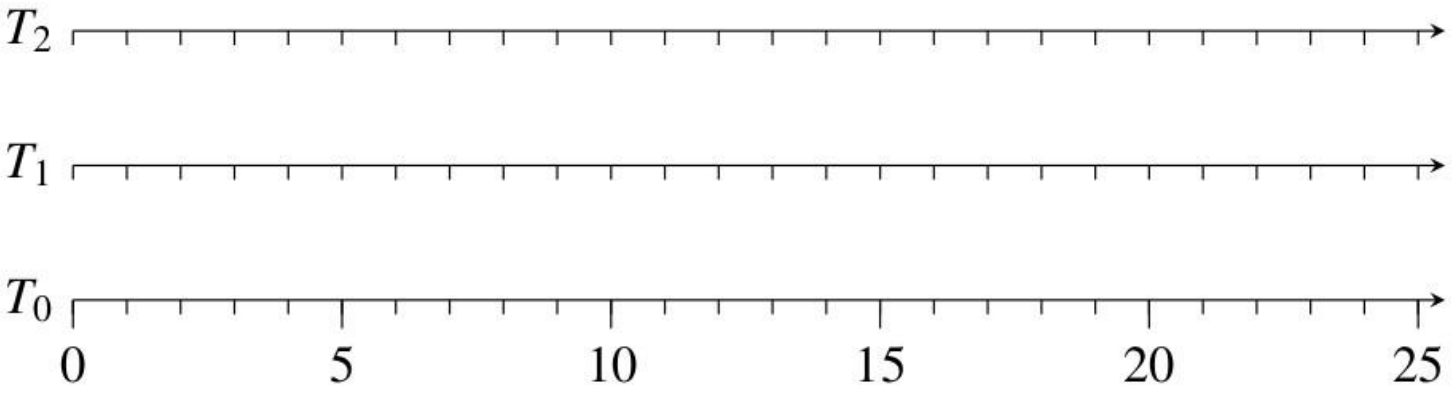
- En 2001, une nouvelle condition d’ordonnançabilité, appelée Hyperbolic Bound, a été proposée. Il s’agit également d’une condition suffisante mais pas nécessaire :

$$\prod_{i=1}^{n}(U_i + 1) \leq 2$$

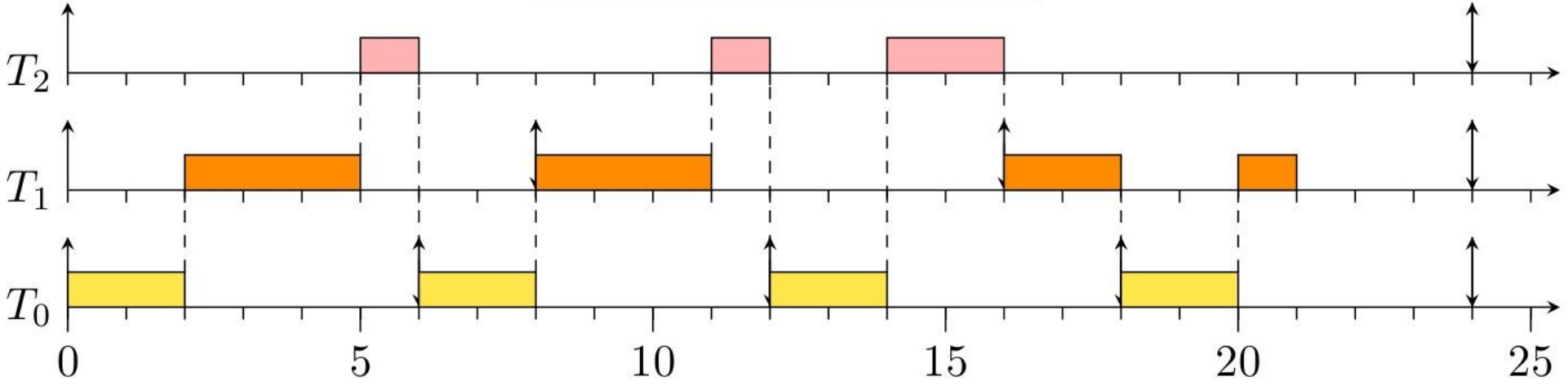
- Cette condition est moins restrictive que la précédente, et est donc intéressante.

Exercice 1

Tâche	Coût	Période
T_0	2	6
T_1	3	8
T_2	4	24



Tâche	Coût	Période
T_0	2	6
T_1	3	8
T_2	4	24



Exercice 1

Exercice 2

Process	Execution time	Period
P1	1	8
P2	2	5
P3	2	10

Exercice 2

- Least Upper Bound
  - L'utilisation sera : $\frac{1}{8} + \frac{2}{5} + \frac{2}{10} = 0.725.$
  - La condition suffisante pour les processus, sous laquelle nous pouvons conclure que le système est planifiable est :

$$U_{lub} = 3(2^{\frac{1}{3}} - 1) = 0.77976 > 0.693$$

  - 0.693 utilisation de RMS pour n=∞
  - Puisque 0.725 < 0.77976 , il est déterminé que le système est garanti comme étant ordonnançable.

Exercice 3

Process	Execution time	Period
P1	3	16
P2	2	5
P3	2	10

- Least Upper Bound
  - L'utilisation sera : $\frac{3}{16} + \frac{2}{5} + \frac{2}{10} = 0.7875.$

Exercice 3

- La condition suffisante pour les processus, sous laquelle nous pouvons conclure que le système est planifiable est :

$$U_{lub} = 3(2^{\frac{1}{3}} - 1) = 0.77976 > 0.693$$

- 0.693 utilisation de RMS pour n=∞
- Puisque 0.7875 > 0.77976 , il est déterminé que le système n'est pas garanti comme étant ordonnançable.

- Hyperbolic Bound
  - En utilisant la limite hyperbolique :

Exercice 3

$$(\frac{3}{16}+1)*(\frac{2}{5}+1)*(\frac{2}{10}+1)=1.995\leq 2$$

- on constate que l'ensemble de tâches est ordonnançable.

Deadline Monotonic

- L’algorithme Rate Monotonic est basé sur l’hypothèse que les tâches sont à échéance sur requête.
- échéances pourront être plus petites que les périodes.
- Dans ce cas, l’algorithme Deadline Monotonic est un algorithme optimal dans le cas des algorithmes à priorité statique avec échéances plus petites que les périodes. Comme RM, il est préemptif.

Analyse d’ordonnançabilité

- Plus son échéance est petite, plus sa priorité est grande.
- Une condition suffisante d’ordonnançabilité est :

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

- la charge du processeur y est surestimée.
- Il est possible de déduire un autre test, en se basant sur les observations suivantes :

Analyse d'ordonnançabilité

1)Le pire cas au niveau des demandes d'utilisation du processeur se trouve au moment où toutes les tâches sont activées simultanément

2)Le pire temps de réponse d'une tâche correspond à la somme de son temps d'exécution et des interférences des tâches de priorité supérieure.

- Si nous supposons les tâches ordonnées selon l'ordre ascendant de leurs échéances
- Ordonnançables si:

$$\forall i : 1 \leq i \leq n \quad C_i + I_i \leq D_i$$

Analyse d’ordonnançabilité

- Où I_i est l’interférence mesurée sur la tâche τ_i :

$$I_i = \sum_{j=1}^{i-1} \left\lceil \frac{D_i}{P_j} \right\rceil C_j$$

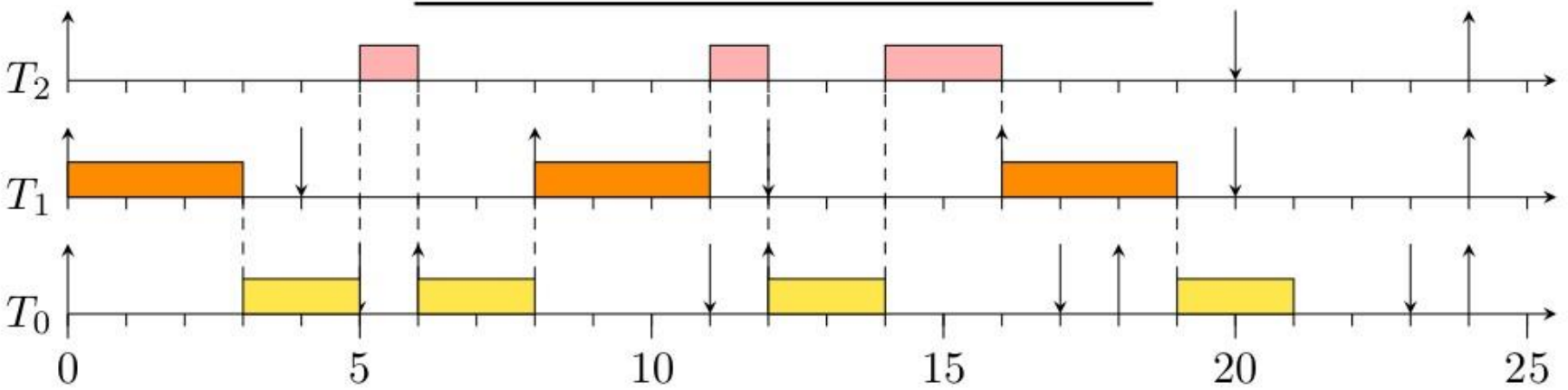
- Test suffisant, mais pas nécessaire

Exemple

Tâche	Coût	Période	Echéance
T_0	2	6	5
T_1	3	8	4
T_2	4	24	20

Exemple

Tâche	Coût	Période	Echéance
T_0	2	6	5
T_1	3	8	4
T_2	4	24	20



Earliest Deadline First

- L’algorithme Earliest Deadline First (EDF) donne la priorité à la tâche ayant l’échéance la plus proche.

Analyse d’ordonnançabilité

- L’algorithme EDF est optimal dans le cas préemptif.
  La condition nécessaire et suffisante d’ordonnançabilité est :

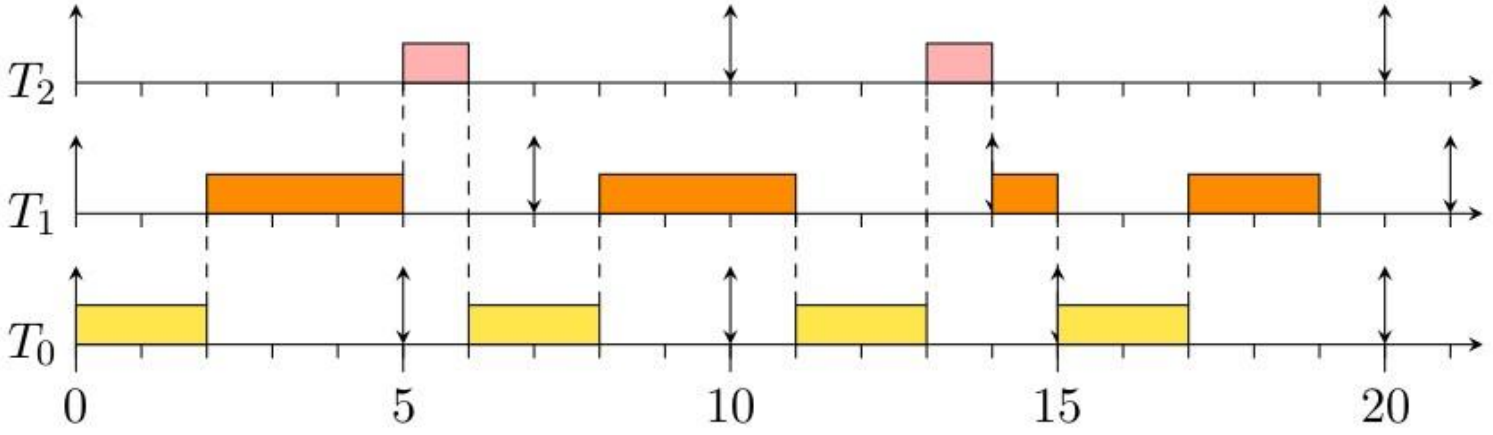
$$\sum_{i=1}^{n} \frac{C_i}{P_i} \leq 1$$

Exemple

Tâche	Coût	Période
T_0	2	5
T_1	3	7
T_2	1	10

Exemple

Tâche	Coût	Période
T_0	2	5
T_1	3	7
T_2	1	10



Least Laxity First

- L’algorithme Least Laxity First (LLF) est un algorithme à priorité dynamique. Il traite des tâches périodiques pour lesquelles la préemption est autorisée.

Least Laxity First

- L = D – C: sa laxité nominal. Indique le retard maximum que peut prendre la tâche sans dépasser son échéance
- D(t) = d – t: son délai critique résiduel au temps t
- C(t): sa durée d’exécution résiduelle au temps t
- L(t) = D(t) – C(t): sa laxité résiduelle au temps t

Analyse d’ordonnançabilité

- La condition d’ordonnançabilité est identique à celle d’EDF.

$$\sum_{i=1}^{n} \frac{C_i}{P_i} \leq 1$$

Exemple