Ordonnancement online de requêtes divisibles

Frédéric Vivien

9 & 16 décembre 2005

Plan de l'exposé

Plan de l'exposé

La problématique

- Une plate-forme hétérogène.
- Un ensemble de bases de données.
  - fichiers textes de quelques Mo à quelques Go ;
  - chaque base est potentiellement repliquée ;
  - une base n'est pas forcément disponible sur tous les processeurs.
- Des requêtes : comparaison d'un motif et d'une base
  - les requêtes sont indépendantes ;
  - un très grand nombre de requêtes ;
  - les motifs sont de tailles différentes.

Application : GriPPS de l'Institut de Biologie et Chimie des Protéines

Analyse de l'application : tâches divisibles

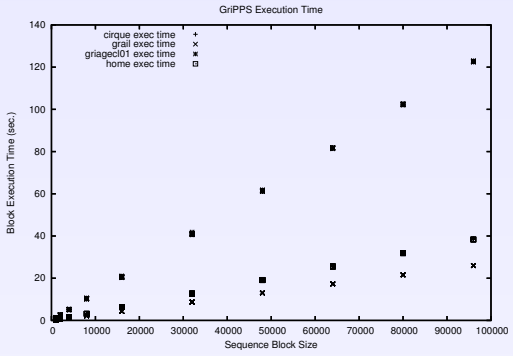


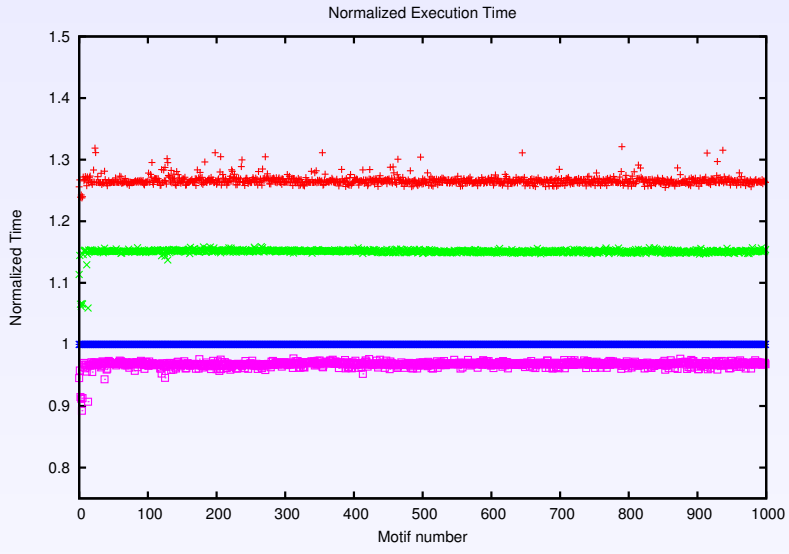
Schéma expérimental :

- Benchmark : ensemble fixé de motifs, comparé avec des sous-ensembles de taille variable d'une base de données.
- Chaque benchmark est exécuté 10 fois.

Conclusions :

- Le temps d'exécution d'une requête est linéaire en la taille de la base de données.
- Représentation compacte des motifs $\Rightarrow$ temps de communications négligeables. (autre expérience)

Analyse de l'application : machines uniformes



- Détermination des vitesses de processeur relatives
  - ensemble de motifs comparés individuelleemnt sur un ensemble de machines
  - moyenne sur 40 itérations
  - temps d'exécution moyen comparé à une machine de référence

Notations

- Tâches J_1, ..., J_n
  Tâche J_j arrive dans le système à la date r_j.
  Tâche J_j a un poids (ou une priorité) w_j.

- Machines M_1, ..., M_m
  La machine M_i prend un temps $c_{i,j}$ pour traiter la tâche J_j.
  $c_{i,j}$ est infini si la tâche J_j a besoin d'une base de données qui n'est pas présente sur la machine M_i.

- Date de fin d'exécution de J_j : C_j.
  Flot de la tâche J_j : $F_j = C_j - r_j$ (temps passé dans le système)

Divisibilité

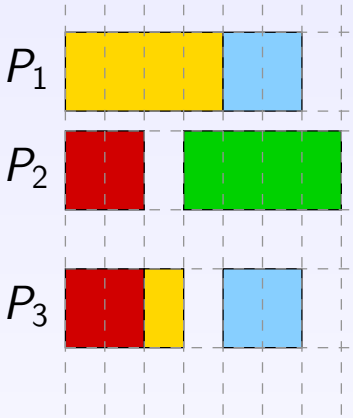
- Chaque tâche est par nature divisible : à un instant donné des processeurs différents peuvent comparer un motif avec des portions différentes d'une même base de données.

- $\alpha_{i,j}$: fraction de J_j traitée par la machine M_i :

$$\forall j, \sum_i \alpha_{i,j} = 1.$$

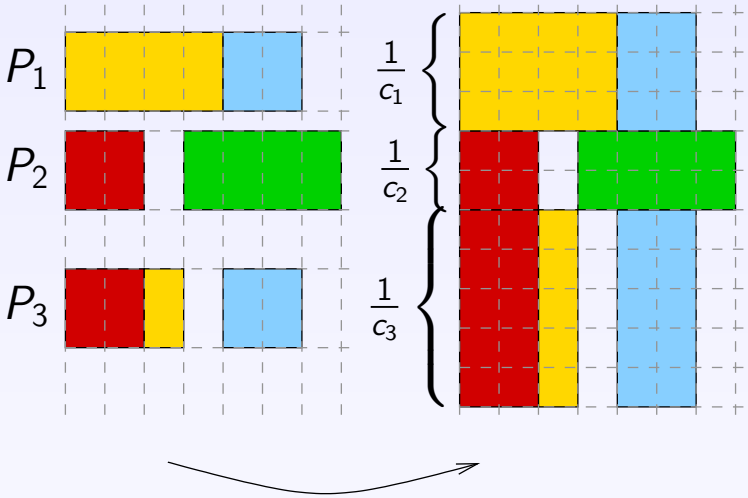
Des tâches divisibles au cas mono-processeur

Transformation géométrique d'un problème uniforme et divisible en un problème mono-processeur.



Des tâches divisibles au cas mono-processeur

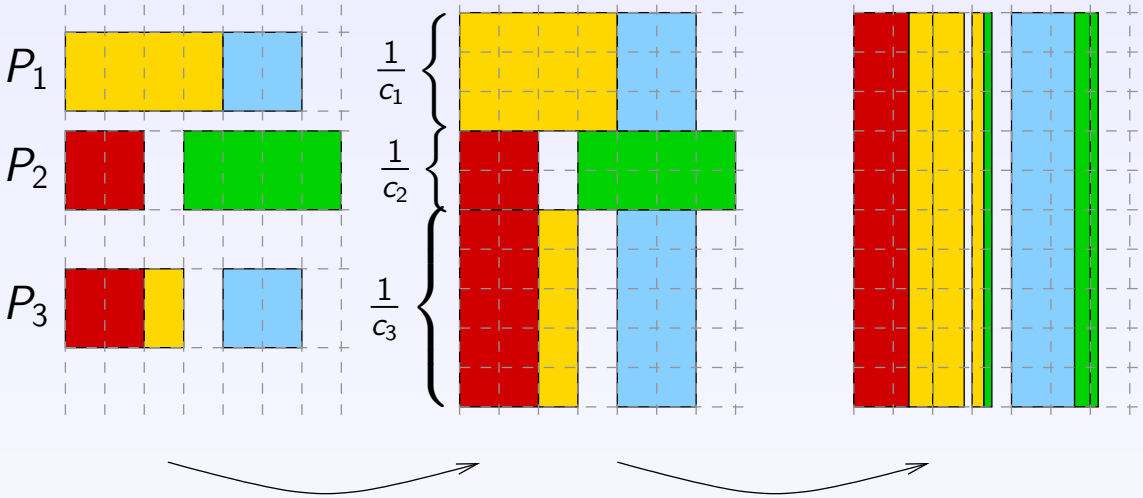
Transformation géométrique d'un problème uniforme et divisible en un problème mono-processeur.



Représentation géométrique de l'hétérogénéité

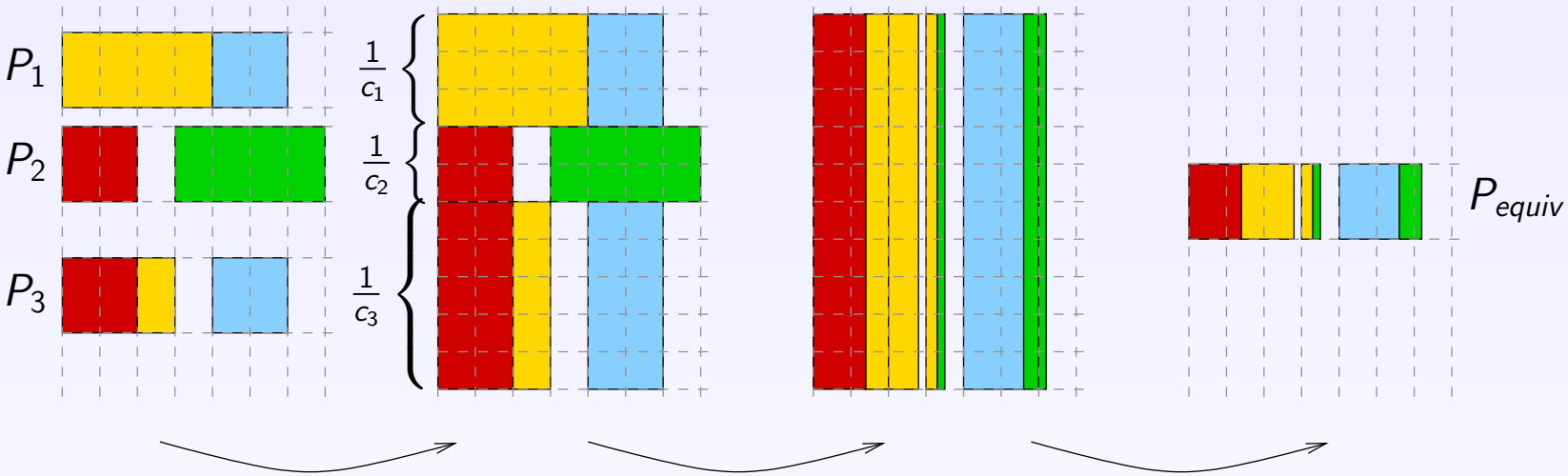
Des tâches divisibles au cas mono-processeur

Transformation géométrique d'un problème uniforme et divisible en un problème mono-processeur.



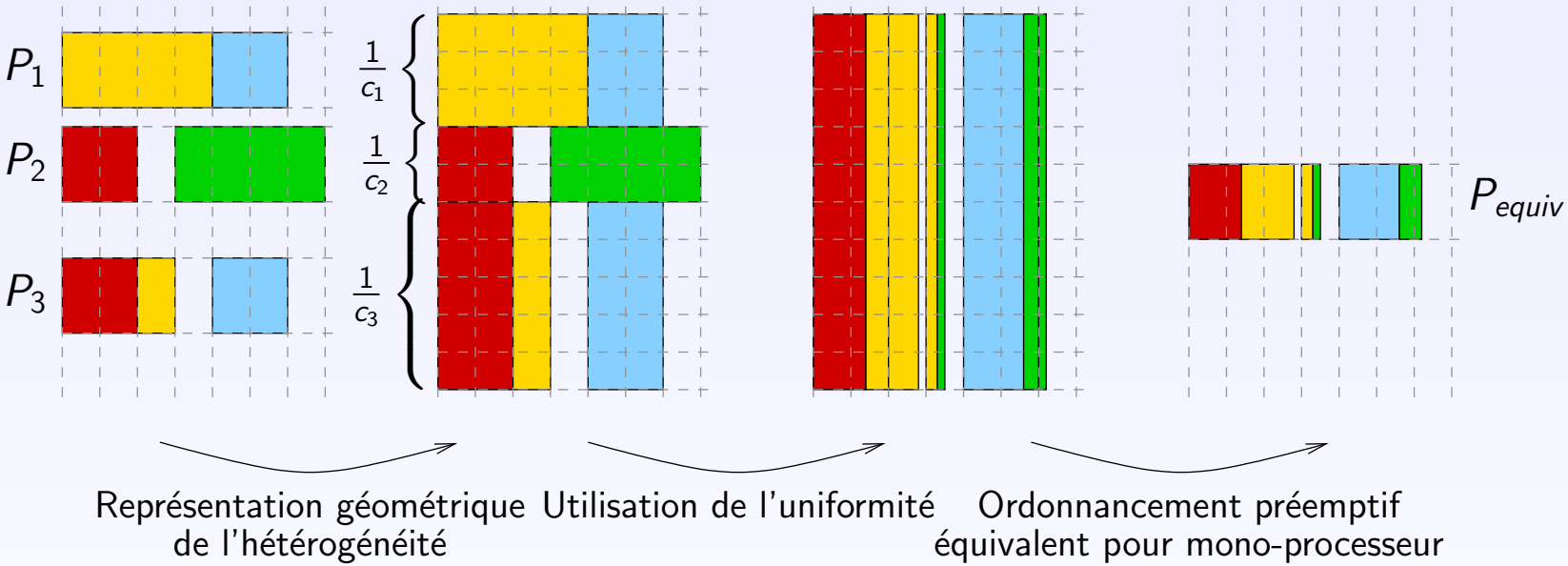
Des tâches divisibles au cas mono-processeur

Transformation géométrique d'un problème uniforme et divisible en un problème mono-processeur.



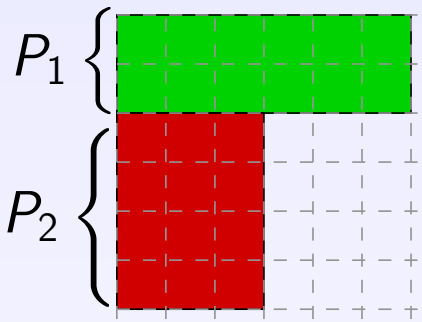
Des tâches divisibles au cas mono-processeur

Transformation géométrique d'un problème uniforme et divisible en un problème mono-processeur.

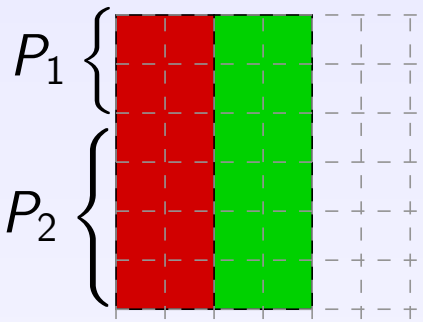


Nous pouvons ne considérer que le cas mono-processeur...
... sauf que nous sommes dans le cas mono-processeur avec disponibilités.

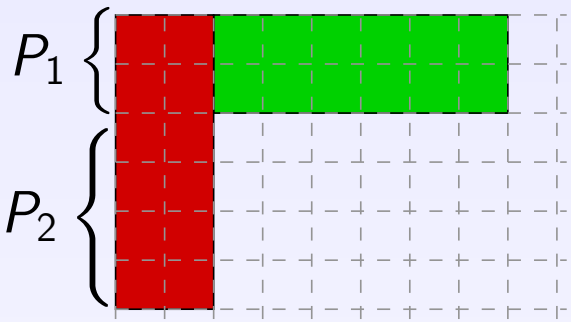
Du cas mono-processeur aux tâches divisibles



A : ordonnancement originel

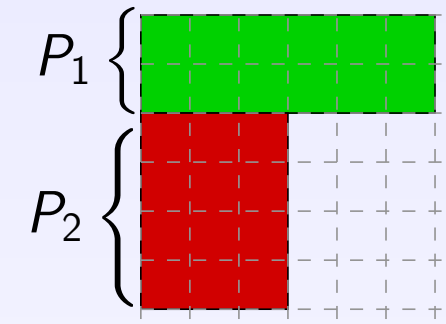


B : machines uniformes

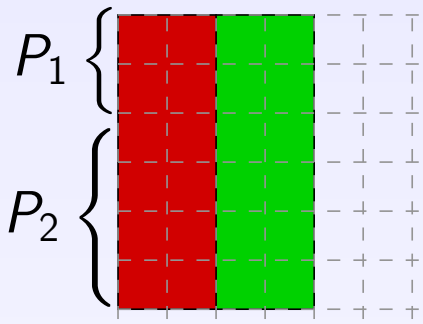


C : disponibilités restreintes

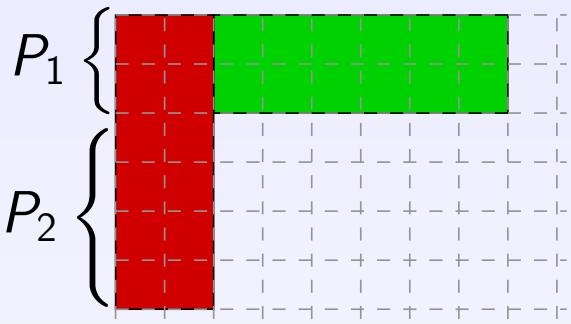
Du cas mono-processeur aux tâches divisibles



A : ordonnancement originel



B : machines uniformes



C : disponibilités restreintes

Règle simple pour étendre les ordonnancements définis pour le cas mono-processeur :

1: **while** des processeurs sont inactifs **do**
2: Sélectionner la tâche de plus grand priorité et la distribuer sur tous les processeurs disponibles qui peuvent l'exécuter.

Que doit-on optimiser ?

- Makespan : $\max_j C_j$.
  Optimisation de l'utilisation de la machine.
  Dates d'arrivée ignorées.

Que doit-on optimiser ?

- Makespan : $\max_j C_j$.
  Optimisation de l'utilisation de la machine.
  Dates d'arrivée ignorées.
- Flot ou temps de réponse moyen : $\sum_j (C_j - r_j)$.
  Optimisation du point de vue de l'utilisateur.

Que doit-on optimiser ?

- Makespan : $\max_j C_j$.
  Optimisation de l'utilisation de la machine.
  Dates d'arrivée ignorées.
- Flot ou temps de réponse moyen : $\sum_j (C_j - r_j)$.
  Optimisation du point de vue de l'utilisateur.
  Inconvénient : possibilité de famine.

Que doit-on optimiser ?

- Makespan : $\max_j C_j$.
  Optimisation de l'utilisation de la machine.
  Dates d'arrivée ignorées.
- Flot ou temps de réponse moyen : $\sum_j (C_j - r_j)$.
  Optimisation du point de vue de l'utilisateur.
  Inconvénient : possibilité de famine.
- Flot ou temps de réponse maximal : $\max_j (C_j - r_j)$.
  Pas de famine. Favorise les tâches longues. Optimisation du pire cas.

Que doit-on optimiser ?

- Makespan : $\max_j C_j$.
  Optimisation de l'utilisation de la machine.
  Dates d'arrivée ignorées.
- Flot ou temps de réponse moyen : $\sum_j (C_j - r_j)$.
  Optimisation du point de vue de l'utilisateur.
  Inconvénient : possibilité de famine.
- Flot ou temps de réponse maximal : $\max_j (C_j - r_j)$.
  Pas de famine. Favorise les tâches longues. Optimisation du pire cas.
- Flot maximal pondéré : $\max_j w_j (C_j - r_j)$.
  Permet de redonner de l'importance aux tâches courtes.

Que doit-on optimiser ?

- Makespan : $\max_j C_j$.
  Optimisation de l'utilisation de la machine.
  Dates d'arrivée ignorées.
- Flot ou temps de réponse moyen : $\sum_j (C_j - r_j)$.
  Optimisation du point de vue de l'utilisateur.
  Inconvénient : possibilité de famine.
- Flot ou temps de réponse maximal : $\max_j (C_j - r_j)$.
  Pas de famine. Favorise les tâches longues. Optimisation du pire cas.
- Flot maximal pondéré : $\max_j w_j (C_j - r_j)$.
  Permet de redonner de l'importance aux tâches courtes.
  Cas particulier du *stretch* : w_j=1/taille de la tâche.

Évaluation de la qualité d'un ordonnancement online

Un algorithme online a un facteur de compétitivité ρ si et seulement si :

Quelque soit l'ensemble de tâches T_1, ..., T_n :

$$\text{Coût ordo online}(T_1, ..., T_N) \leq$$
$$\rho \times \text{Coût ordo optimal offline}(T_1, ..., T_N).$$

Minimisation du *stretch*

On considère la minimisation du *stretch* maximal ou moyen.

Minimisation du *stretch*

On considère la minimisation du *stretch* maximal ou moyen.

Théorème

Δ : ratio des tailles de la plus grande et de la plus petite tâche. Soit un algorithme online de ratio de compétitivité $\rho(\Delta) < \Delta$ pour la minimisation du stretch moyen.

Il existe pour cet algorithme une série de tâches entraînant une famine, et pour laquelle le stretch maximal atteint est arbitrairement éloigné du stretch maximum optimal.

Plan de l'exposé

Minimisation du flot maximal ou moyen

- Le flot maximal est minimisé par la politique *premier arrivé, premier servi*.

- Le flot moyen est minimisé par la politique *plus petit temps de travail restant* (*Shortest Remaining Processing Time* ou SRPT).

Minimisation du stretch moyen

- Complexité du problème : ouverte.

Minimisation du stretch moyen

- Complexité du problème : ouverte.
- Polynomial Time Approximation Schemes (PTAS).

Minimisation du stretch moyen

- Complexité du problème : ouverte.
- Polynomial Time Approximation Schemes (PTAS).
- Le ratio de compétitivité de tout algorithme online vaut au moins : 1.19485.

Minimisation du stretch moyen

- Complexité du problème : ouverte.
- Polynomial Time Approximation Schemes (PTAS).
- Le ratio de compétitivité de tout algorithme online vaut au moins : 1.19485.
- *Shortest Remaining Processing Time* est 2-competitive.

Minimisation du stretch moyen

- Complexité du problème : ouverte.
- Polynomial Time Approximation Schemes (PTAS).
- Le ratio de compétitivité de tout algorithme online vaut au moins : 1.19485.
- *Shortest Remaining Processing Time* est 2-competitive.
- Amélioration évidente : *Shortest Weighted Remaining Processing Time* (plus court temps de calcul restant, pondéré).
  À tout instant t, SWRPT ordonnances la tâche J_j qui minimises $p_j \rho_t(j)$.
  SWRPT est *au mieux* 2-compétitive.

Minimisation du stretch moyen

- Complexité du problème : ouverte.
- Polynomial Time Approximation Schemes (PTAS).
- Le ratio de compétitivité de tout algorithme online vaut au moins : 1.19485.
- *Shortest Remaining Processing Time* est 2-competitive.
- Amélioration évidente : *Shortest Weighted Remaining Processing Time* (plus court temps de calcul restant, pondéré).
  À tout instant t, SWRPT ordonnances la tâche J_j qui minimises $p_j \rho_t(j)$.
  SWRPT est *au mieux* 2-compétitive.

Le cas offline semble difficile... mais il existe un algorithme d'approximation simple dans le cas online.

Plan de l'exposé

Cadre offline : règles du jeu

- On connaît à l'avance les dates d'arrivée des tâches et leurs durées (cadre *offline*).
- On cherche à minimiser le flot maximal pondéré (poids non spécifiés).
- Les tâches sont divisibles.

Existence d'un ordonnancement de max-stretch donné

Existence d'un ordonnancement de max-stretch $\mathcal{S}$:

Pour chaque tâche J_j, $\quad \frac{C_j - r_j}{p_j} \leq \mathcal{S}$

Équivalent à un problème d'ordonnancement avec dates butoirs où $d_j(\mathcal{S}) = r_j + p_j \times \mathcal{S}$

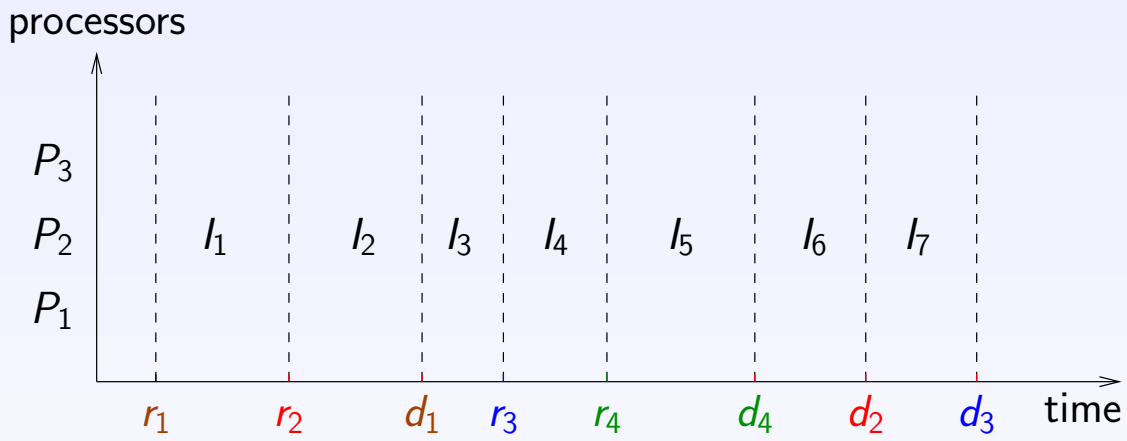
Ordonnancement avec dates butoir : définition

- Chaque job J_j a une date butoir (*deadline*) d_j en plus de sa date d'arrivée r_j.

- Question : existe-t-il un ordonnancement qui exécute chaque tâche J_j dans son intervalle $[r_j, d_j]$?

Ordonnancement avec dates butoir : notations

Ensemble des dates d'arrivée et butoirs : $\{r_1, ..., r_n, d_1, ..., d_n\}$.



Ces dates, triées, définissent un ensemble de n_{int} intervalles $I_1, ..., I_{n_{\text{int}}}$, avec $1 \leq n_{\text{int}} \leq 2n - 1$.

$I_t = [\inf I_t, \sup I_t[$

$\alpha_{i,j}^{(t)}$: fraction de J_j traitée par M_i pendant l'intervalle I_t.

Ordonnancement avec dates butoir : résolution

1. *Dates d'arrivées* :

$$\forall i, \forall j, \forall t, \quad r_j \geq \sup I_t \Rightarrow \alpha_{i,j}^{(t)} = 0$$

Ordonnancement avec dates butoir : résolution

1. *Dates d'arrivées* :

$$\forall i, \forall j, \forall t, \quad r_j \geq \sup I_t \Rightarrow \alpha_{i,j}^{(t)} = 0$$

2. *Dates butoirs* :

$$\forall i, \forall j, \forall t, \quad d_j \leq \inf I_t \Rightarrow \alpha_{i,j}^{(t)} = 0$$

Ordonnancement avec dates butoir : résolution

1. *Dates d'arrivées* :

$$\forall i, \forall j, \forall t, \quad r_j \geq \sup I_t \Rightarrow \alpha_{i,j}^{(t)} = 0$$

2. *Dates butoirs* :

$$\forall i, \forall j, \forall t, \quad d_j \leq \inf I_t \Rightarrow \alpha_{i,j}^{(t)} = 0$$

3. *Contraintes de ressources* :

$$\forall t, \forall i, \quad \sum_j \alpha_{i,j}^{(t)} . c_{i,j} \leq \sup I_t - \inf I_t$$

Ordonnancement avec dates butoir : résolution

1. *Dates d'arrivées* :

$$\forall i, \forall j, \forall t, \quad r_j \geq \sup I_t \Rightarrow \alpha_{i,j}^{(t)} = 0$$

2. *Dates butoirs* :

$$\forall i, \forall j, \forall t, \quad d_j \leq \inf I_t \Rightarrow \alpha_{i,j}^{(t)} = 0$$

3. *Contraintes de ressources* :

$$\forall t, \forall i, \quad \sum_j \alpha_{i,j}^{(t)} . c_{i,j} \leq \sup I_t - \inf I_t$$

4. *Complétion des tâches* :

$$\forall j, \quad \sum_t \sum_i \alpha_{i,j}^{(t)} = 1$$

Flot pondéré maximal et dates butoirs

- Existe-t-il un ordonnancement réalisant un flot pondéré maximal $\mathcal{F}$?

Flot pondéré maximal et dates butoirs

- Existe-t-il un ordonnancement réalisant un flot pondéré maximal $\mathcal{F}$?
- $\max_j w_j(C_j - r_j) \leq \mathcal{F} \quad \Leftrightarrow \quad \forall j, w_j(C_j - r_j) \leq \mathcal{F} \quad \Leftrightarrow$
  $\forall j, C_j \leq r_j + \mathcal{F}/w_j$.

Flot pondéré maximal et dates butoirs

- Existe-t-il un ordonnancement réalisant un flot pondéré maximal $\mathcal{F}$?
- $\max_j w_j(C_j - r_j) \leq \mathcal{F} \quad \Leftrightarrow \quad \forall j, w_j(C_j - r_j) \leq \mathcal{F} \quad \Leftrightarrow$
  $\forall j, C_j \leq r_j + \mathcal{F}/w_j$.
- Équivalent à résoudre le problème d'ordonnancement avec dates butoirs où $d_j(\mathcal{F}) = r_j + \mathcal{F}/w_j$ est le *deadline* de la tâche J_j.

Flot pondéré maximal et dates butoirs

- Existe-t-il un ordonnancement réalisant un flot pondéré maximal $\mathcal{F}$?
- $\max_j w_j(C_j - r_j) \leq \mathcal{F} \quad \Leftrightarrow \quad \forall j, w_j(C_j - r_j) \leq \mathcal{F} \quad \Leftrightarrow$
  $\forall j, C_j \leq r_j + \mathcal{F}/w_j$.
- Équivalent à résoudre le problème d'ordonnancement avec dates butoirs où $d_j(\mathcal{F}) = r_j + \mathcal{F}/w_j$ est le *deadline* de la tâche J_j.
- Il suffit de faire une dichotomie sur $\mathcal{F}$!

Flot pondéré maximal et dates butoirs

- Existe-t-il un ordonnancement réalisant un flot pondéré maximal $\mathcal{F}$?
- $\max_j w_j(C_j - r_j) \leq \mathcal{F} \quad \Leftrightarrow \quad \forall j, w_j(C_j - r_j) \leq \mathcal{F} \quad \Leftrightarrow$
  $\forall j, C_j \leq r_j + \mathcal{F}/w_j$.
- Équivalent à résoudre le problème d'ordonnancement avec dates butoirs où $d_j(\mathcal{F}) = r_j + \mathcal{F}/w_j$ est le *deadline* de la tâche J_j.
- Il suffit de faire une dichotomie sur $\mathcal{F}$!

  On préférerait une solution qui termine...

Résolution sur un intervalle d'objectifs (1)

- Soient $\mathcal{F}_1$ et $\mathcal{F}_2$, $\mathcal{F}_1 < \mathcal{F}_2$.

  On suppose que l'ordre relatif des dates d'arrivées et des dates butoirs, $r_1, \ldots, r_n, d_1(\mathcal{F}), \ldots, d_n(\mathcal{F})$, est indépendant de $\mathcal{F} \in]\mathcal{F}_1; \mathcal{F}_2[$.

Résolution sur un intervalle d'objectifs (1)

- Soient $\mathcal{F}_1$ et $\mathcal{F}_2$, $\mathcal{F}_1 < \mathcal{F}_2$.

  On suppose que l'ordre relatif des dates d'arrivées et des dates butoirs, $r_1, \ldots, r_n, d_1(\mathcal{F}), \ldots, d_n(\mathcal{F})$, est indépendant de $\mathcal{F} \in]\mathcal{F}_1; \mathcal{F}_2[$.

- On définit à partir de l'ensemble $r_1, \ldots, r_n, d_1(\mathcal{F}), \ldots, d_n(\mathcal{F})$ des intervalles comme précédemment.

  Les extrémités d'un intervalle sont des fonctions affines de $\mathcal{F}$.

Résolution sur un intervalle d'objectifs (1)

- Soient $\mathcal{F}_1$ et $\mathcal{F}_2$, $\mathcal{F}_1 < \mathcal{F}_2$.

  On suppose que l'ordre relatif des dates d'arrivées et des dates butoirs, $r_1, \ldots, r_n, d_1(\mathcal{F}), \ldots, d_n(\mathcal{F})$, est indépendant de $\mathcal{F} \in]\mathcal{F}_1; \mathcal{F}_2[$.

- On définit à partir de l'ensemble $r_1, \ldots, r_n, d_1(\mathcal{F}), \ldots, d_n(\mathcal{F})$ des intervalles comme précédemment.

  Les extrémités d'un intervalle sont des fonctions affines de $\mathcal{F}$.

- On adapte le système précédent.

Résolution sur un intervalle d'objectifs (2)

$$\forall i, \forall j, \forall t, \quad r_j \geq \sup I_t(\mathcal{F}) \Rightarrow \alpha_{i,j}^{(t)} = 0$$

$$\forall i, \forall j, \forall t, \quad d_j(\mathcal{F}) \leq \inf I_t(\mathcal{F}) \Rightarrow \alpha_{i,j}^{(t)} = 0$$

$$\forall t, \forall i, \quad \sum_j \alpha_{i,j}^{(t)}.c_{i,j} \leq \sup I_t(\mathcal{F}) - \inf I_t(\mathcal{F})$$

$$\forall j, \quad \sum_t \sum_i \alpha_{i,j}^{(t)} = 1$$

Résolution sur un intervalle d'objectifs (2)

$$\forall i, \forall j, \forall t, \quad r_j \geq \sup I_t(\mathcal{F}) \Rightarrow \alpha_{i,j}^{(t)} = 0$$

$$\forall i, \forall j, \forall t, \quad d_j(\mathcal{F}) \leq \inf I_t(\mathcal{F}) \Rightarrow \alpha_{i,j}^{(t)} = 0$$

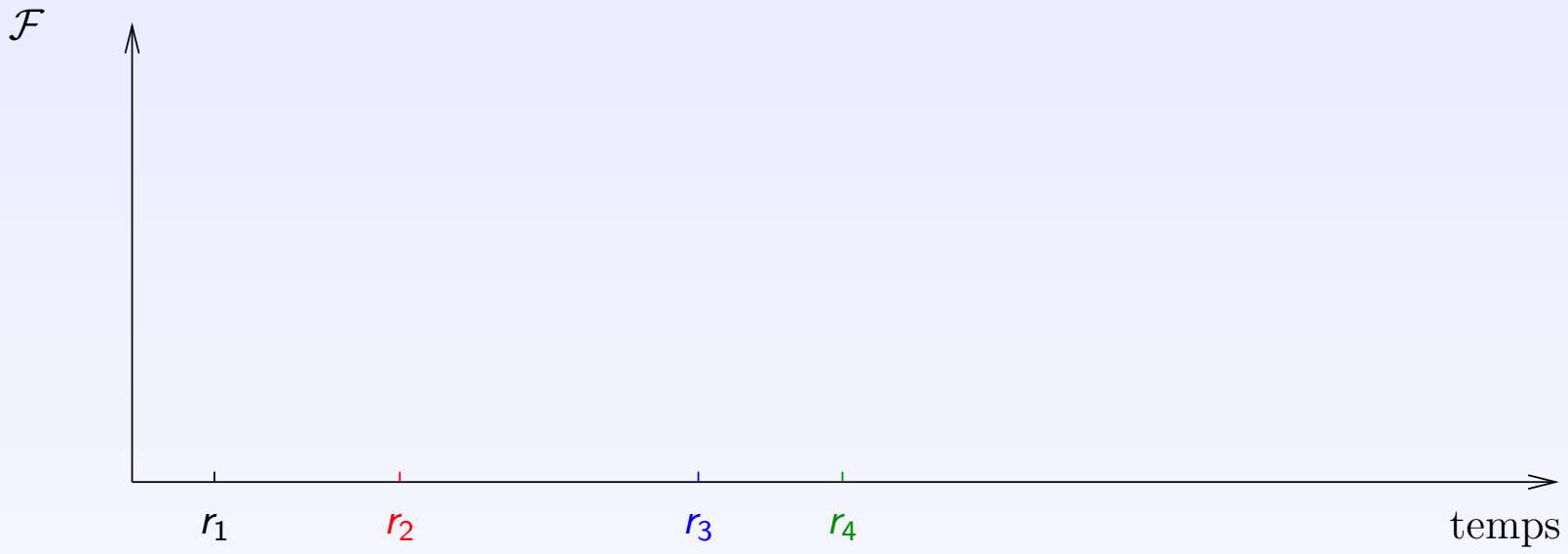
$$\forall t, \forall i, \quad \sum_j \alpha_{i,j}^{(t)}.c_{i,j} \leq \sup I_t(\mathcal{F}) - \inf I_t(\mathcal{F})$$

$$\forall j, \quad \sum_t \sum_i \alpha_{i,j}^{(t)} = 1$$

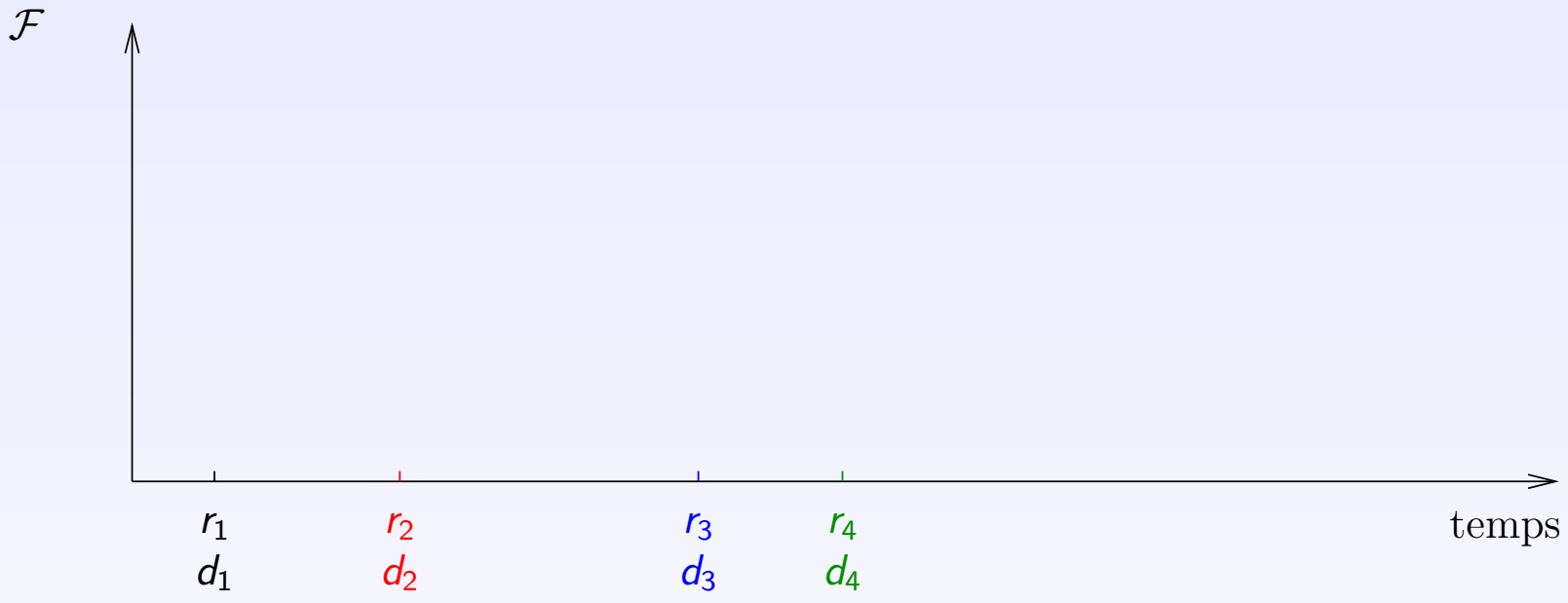
$$\mathcal{F}_1 \leq \mathcal{F} \leq \mathcal{F}_2$$

$$\min \mathcal{F}$$

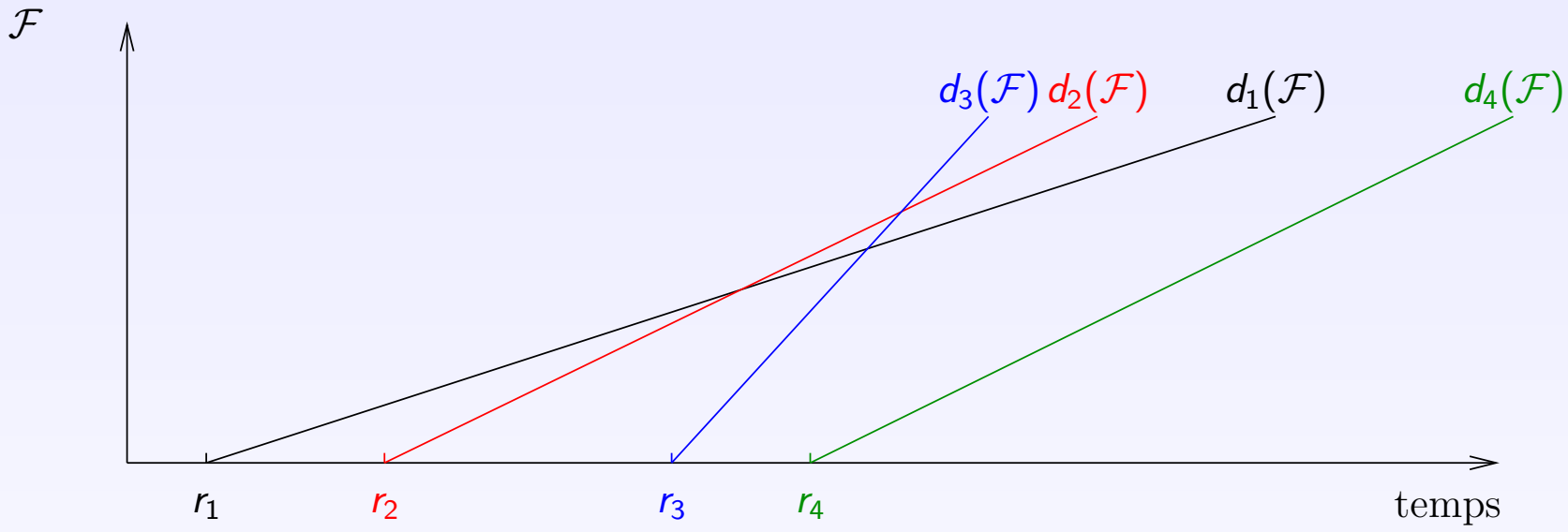
Ordre relatif des dates clefs



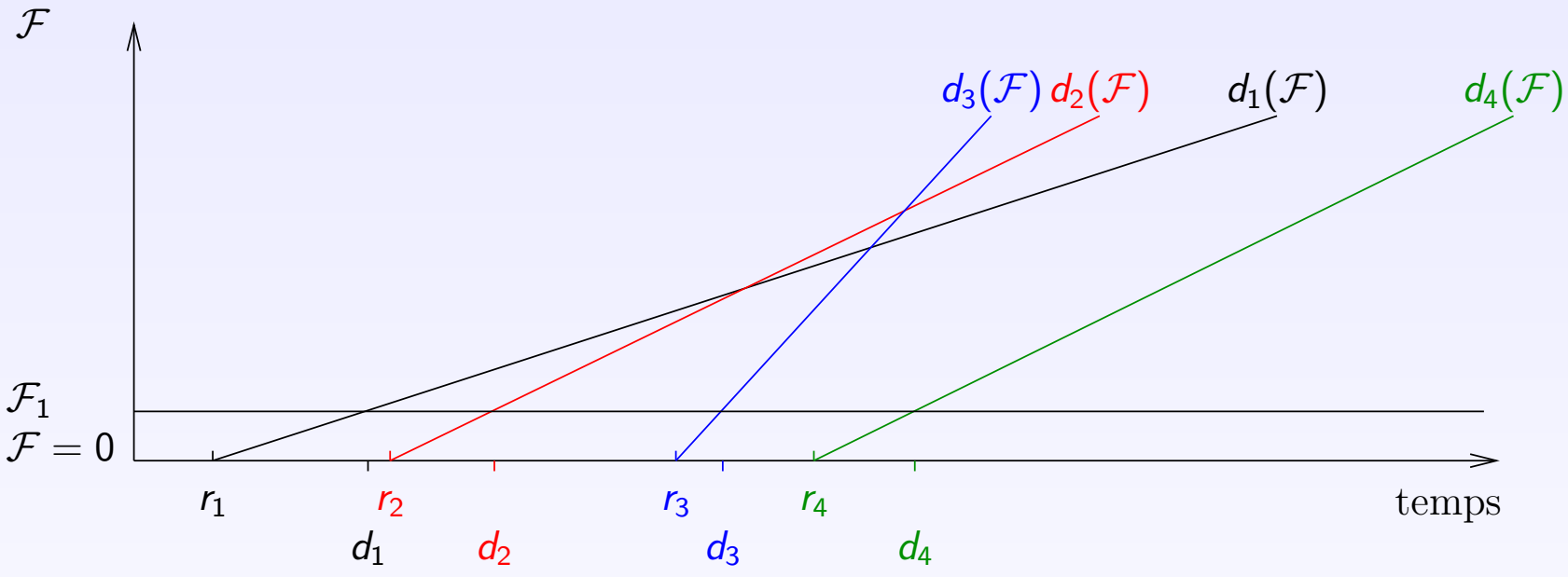
Ordre relatif des dates clefs



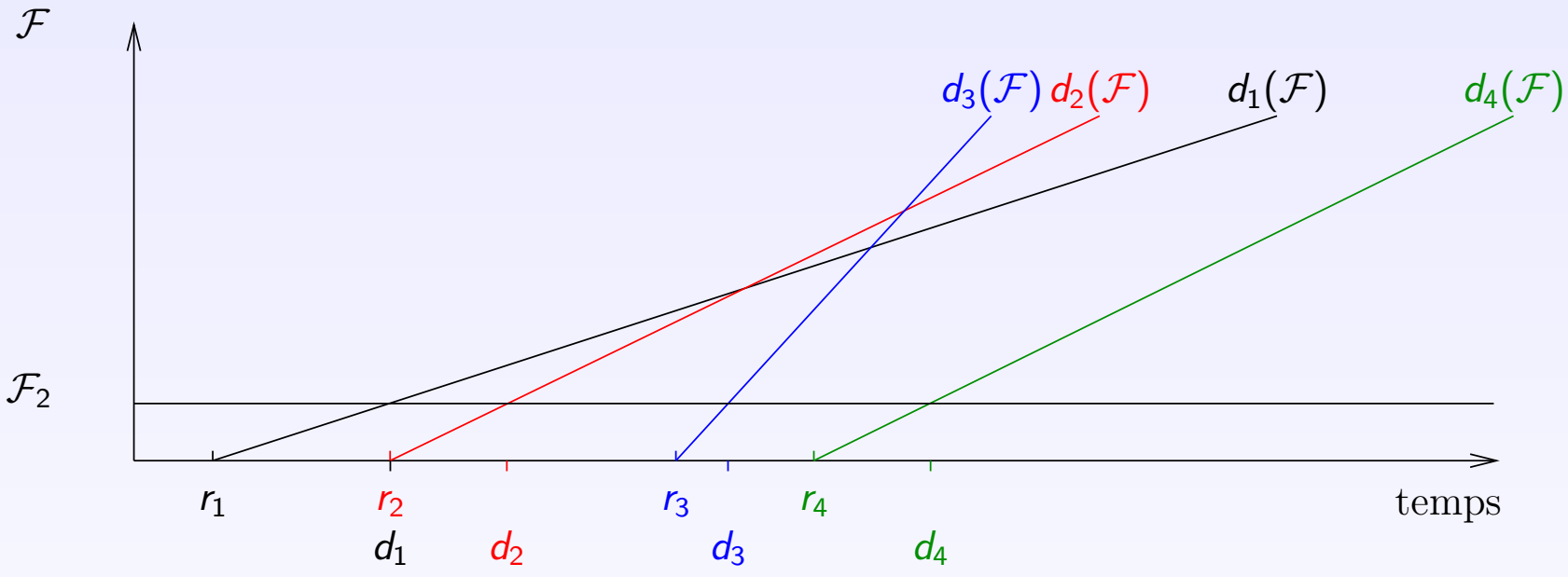
Ordre relatif des dates clefs



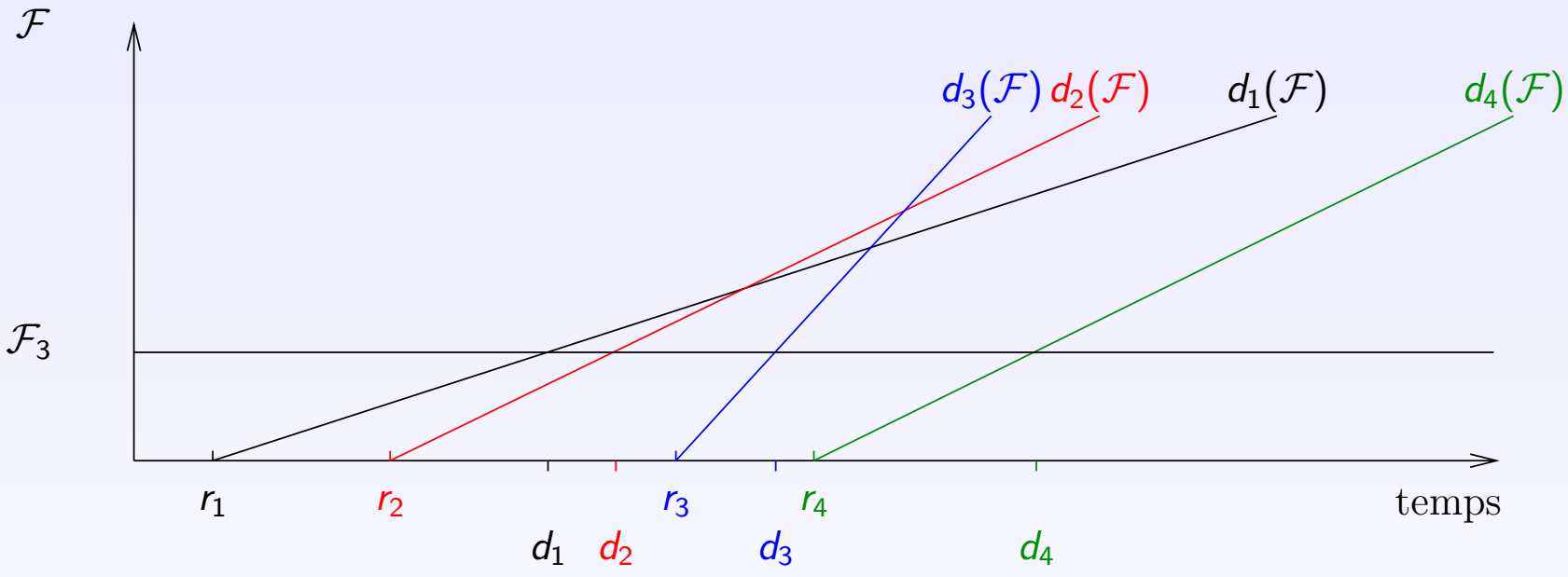
Ordre relatif des dates clefs



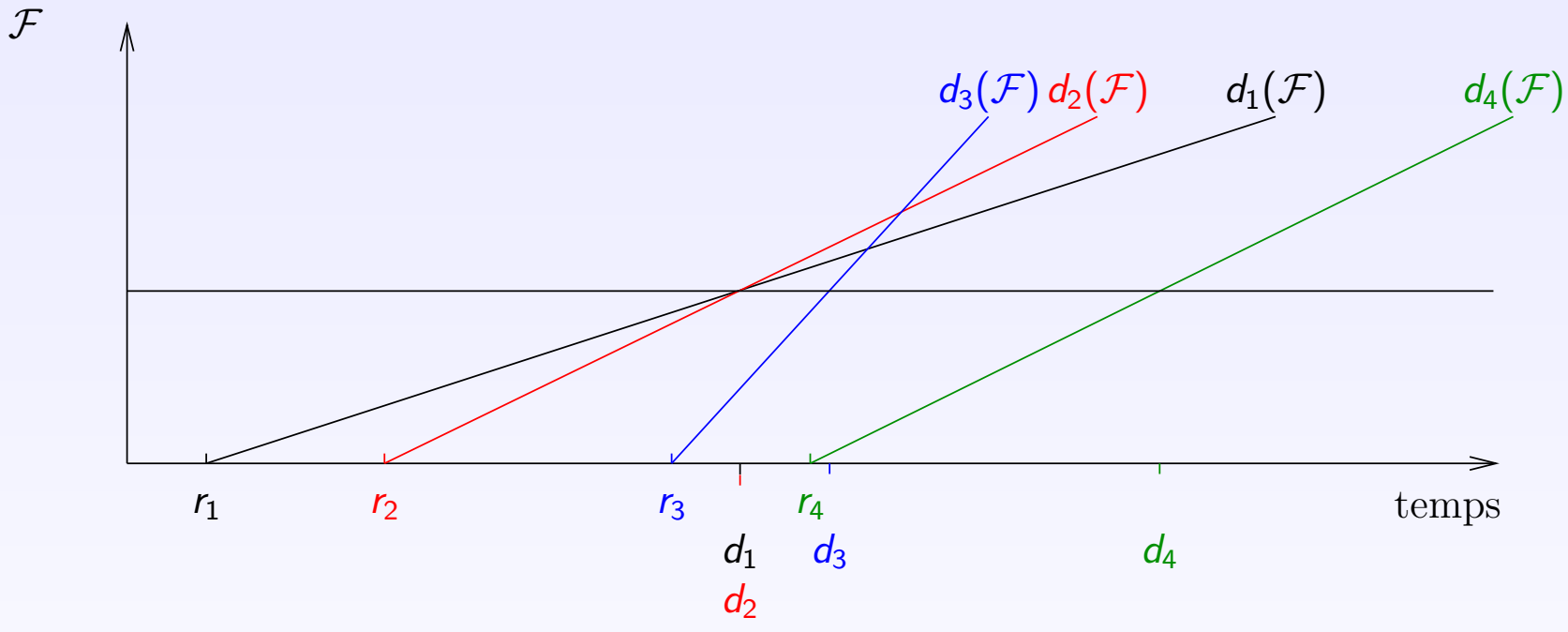
Ordre relatif des dates clefs



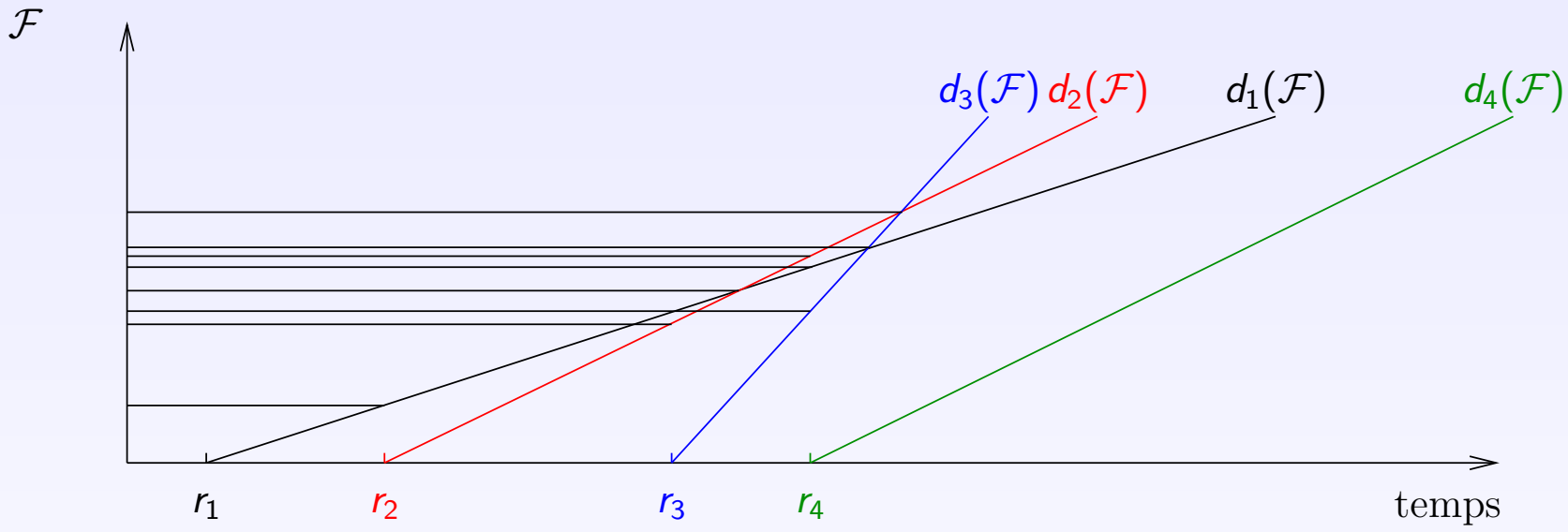
Ordre relatif des dates clefs



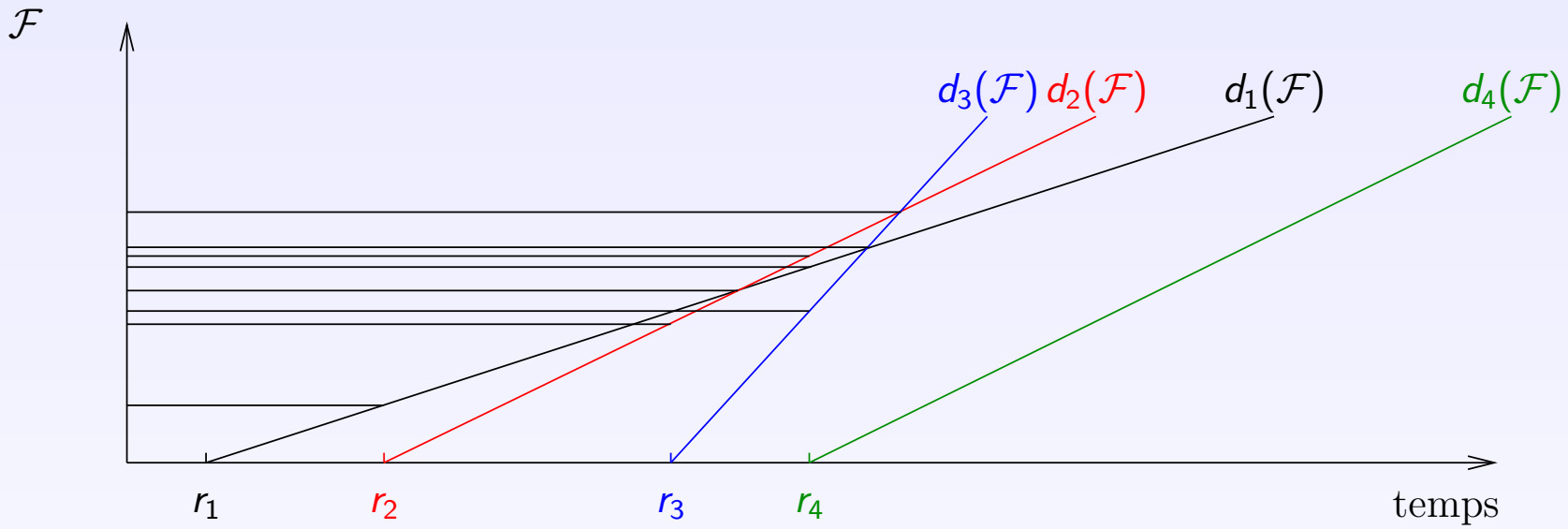
Ordre relatif des dates clefs



Ordre relatif des dates clefs



Ordre relatif des dates clefs



Il y a au plus $n_q \leq n^2 - n$ intersections possibles.

Résolution

- On calcule des $n_q \leq n^2 - n$ intersections possibles.

Résolution

- On calcule des $n_q \leq n^2 - n$ intersections possibles.
- Soient $\mathcal{F}_1, \mathcal{F}_2, ..., \mathcal{F}_{n_q}$ les valeurs particulières de l'objectif.

  On effectue une recherche dichotomique sur l'ensemble des valeurs particulières $\mathcal{F}_i$, en recherchant à chaque fois une solution dans l'intervalle $[\mathcal{F}_i, \mathcal{F}_{i+1}]$.

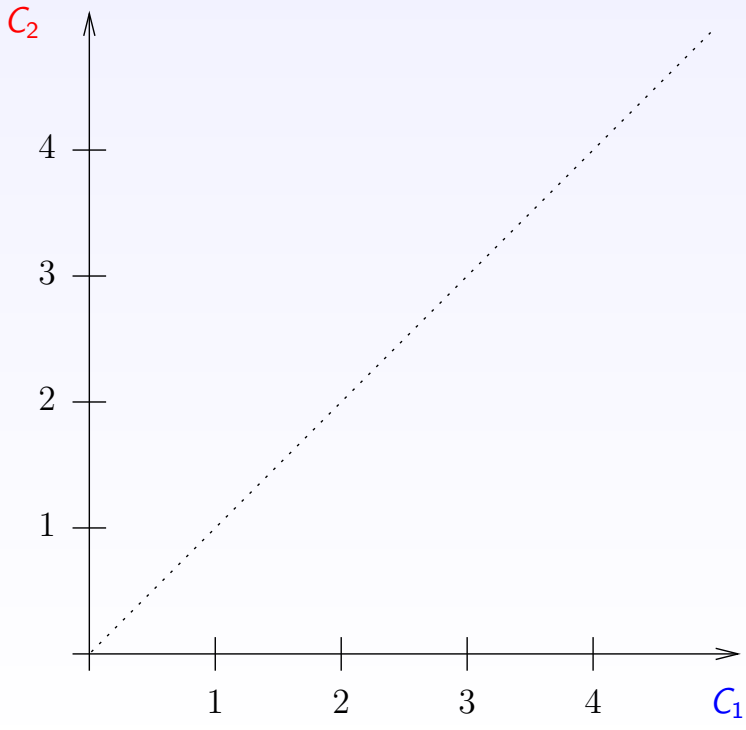
Résolution

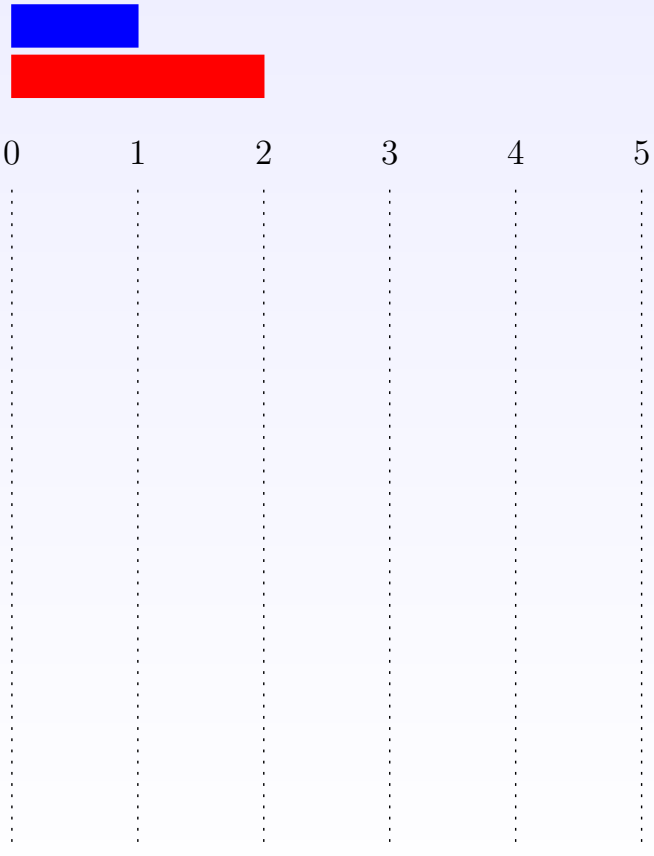
- On calcule des $n_q \leq n^2 - n$ intersections possibles.
- Soient $\mathcal{F}_1, \mathcal{F}_2, ..., \mathcal{F}_{n_q}$ les valeurs particulières de l'objectif.

  On effectue une recherche dichotomique sur l'ensemble des valeurs particulières $\mathcal{F}_i$, en recherchant à chaque fois une solution dans l'intervalle $[\mathcal{F}_i, \mathcal{F}_{i+1}]$.
- Algorithme polynomial de résolution.

Pareto optimalité

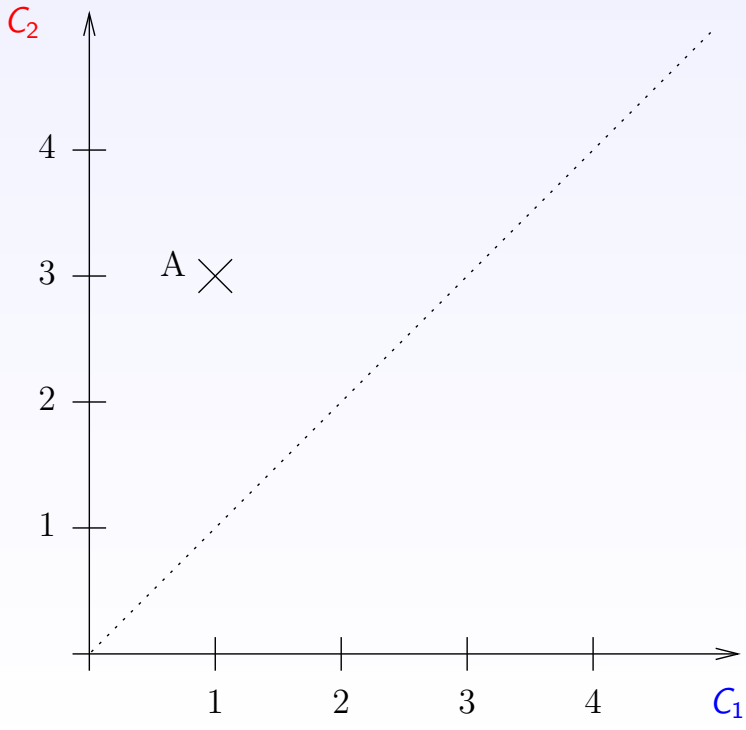
Le coût (ici le temps de complétion) d'un utilisateur ne peut pas être amélioré sans dégrader le coût d'un autre utilisateur.

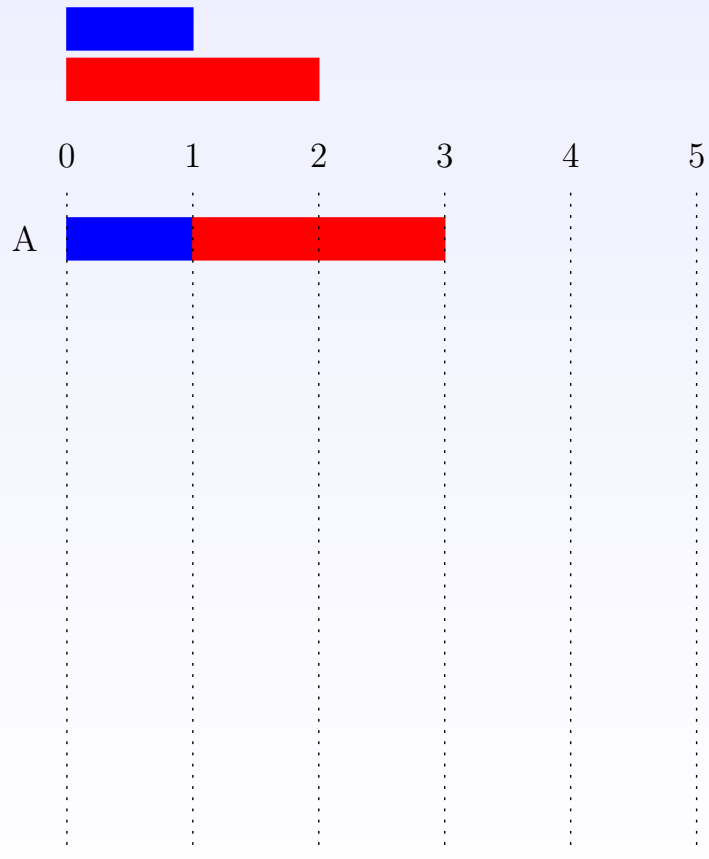




Pareto optimalité

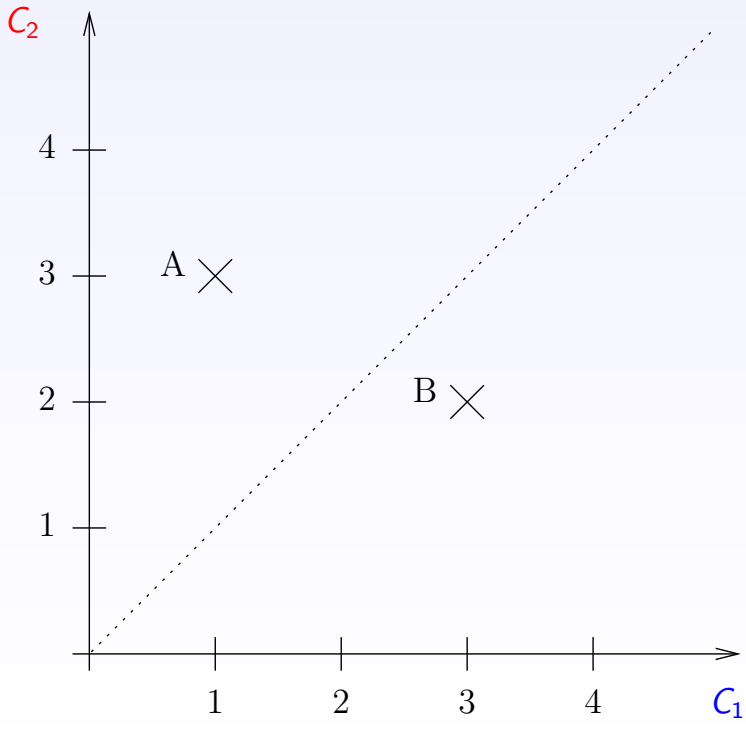
Le coût (ici le temps de complétion) d'un utilisateur ne peut pas être amélioré sans dégrader le coût d'un autre utilisateur.

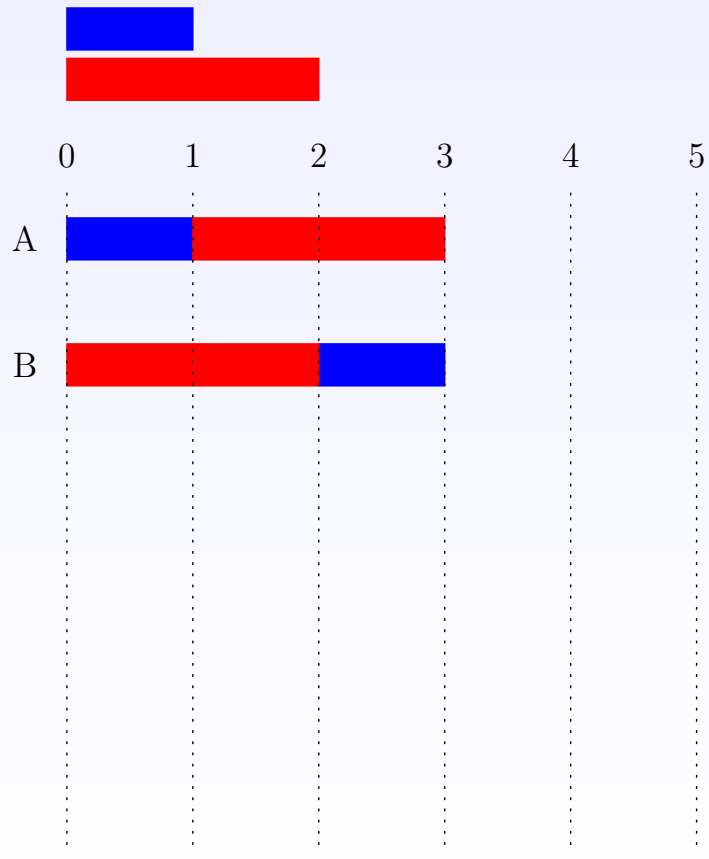




Pareto optimalité

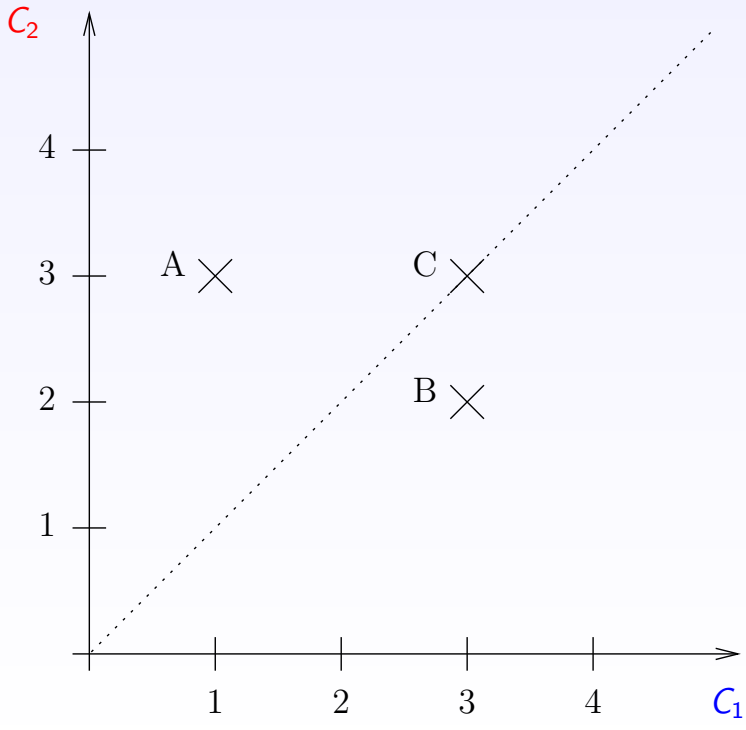
Le coût (ici le temps de complétion) d'un utilisateur ne peut pas être amélioré sans dégrader le coût d'un autre utilisateur.

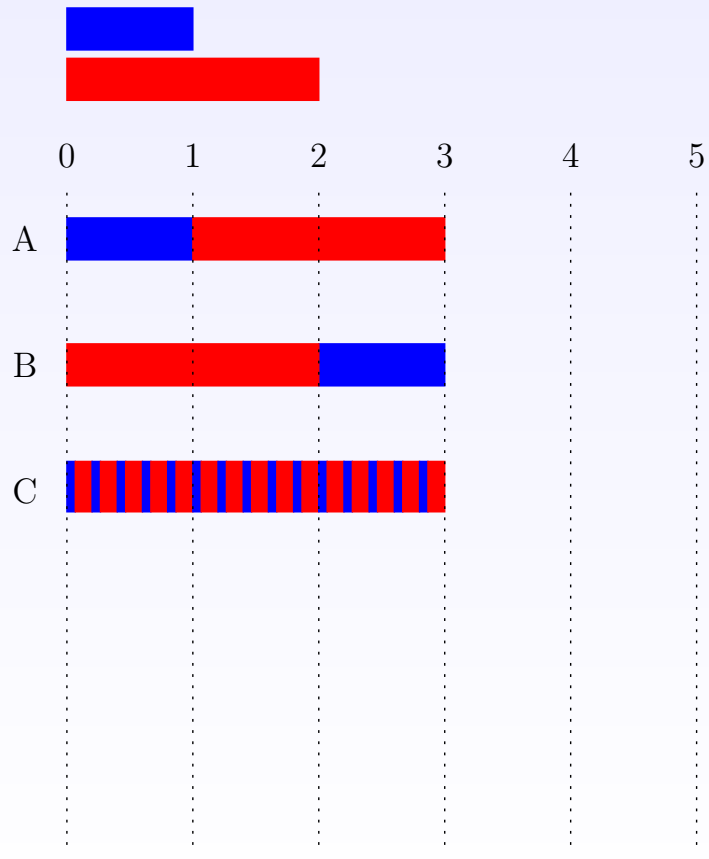




Pareto optimalité

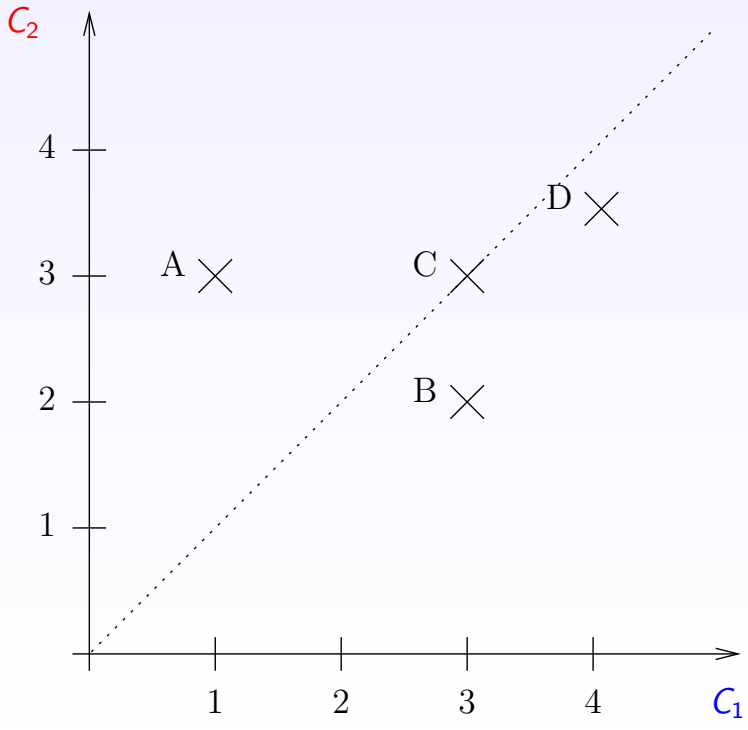
Le coût (ici le temps de complétion) d'un utilisateur ne peut pas être amélioré sans dégrader le coût d'un autre utilisateur.

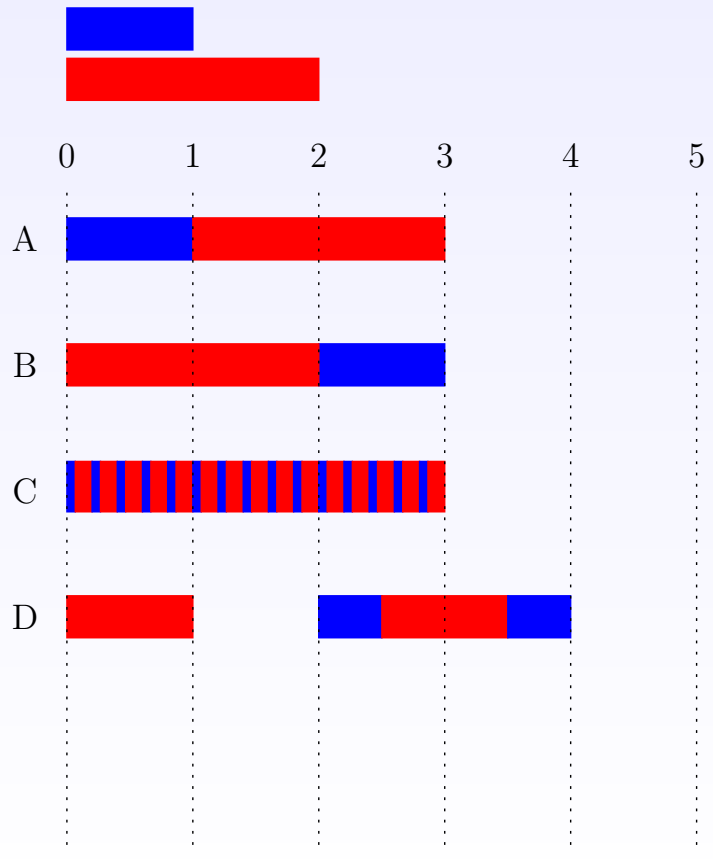




Pareto optimalité

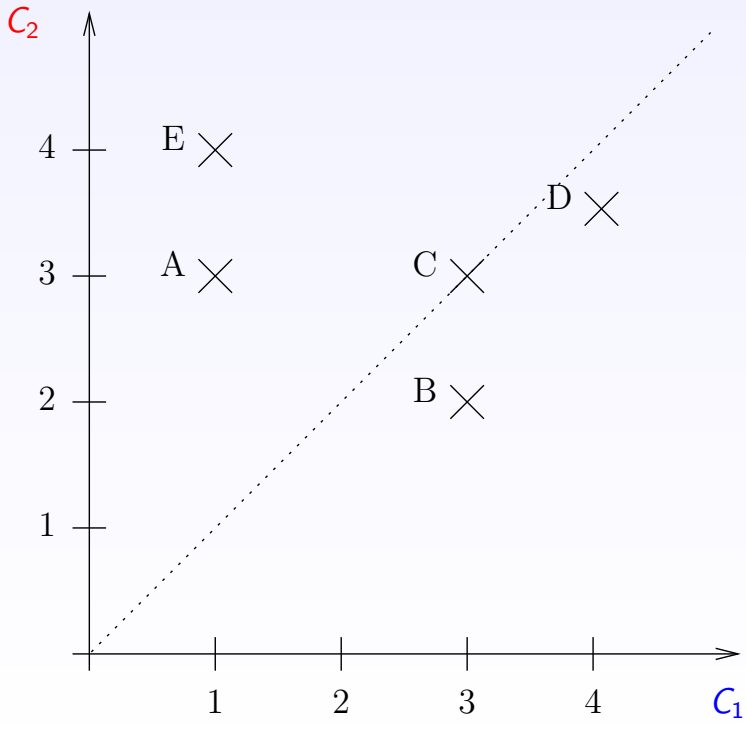
Le coût (ici le temps de complétion) d'un utilisateur ne peut pas être amélioré sans dégrader le coût d'un autre utilisateur.

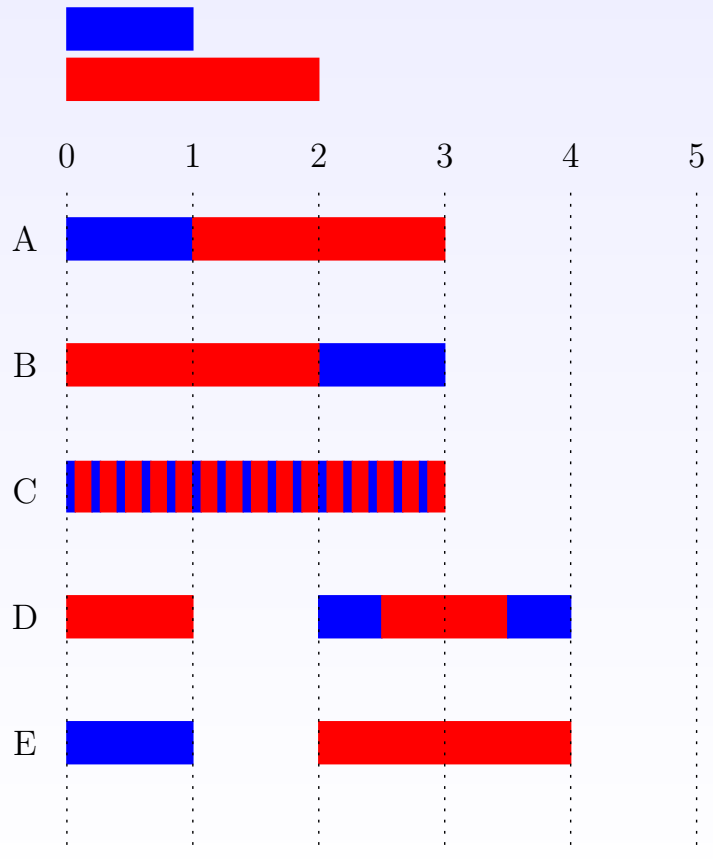




Pareto optimalité

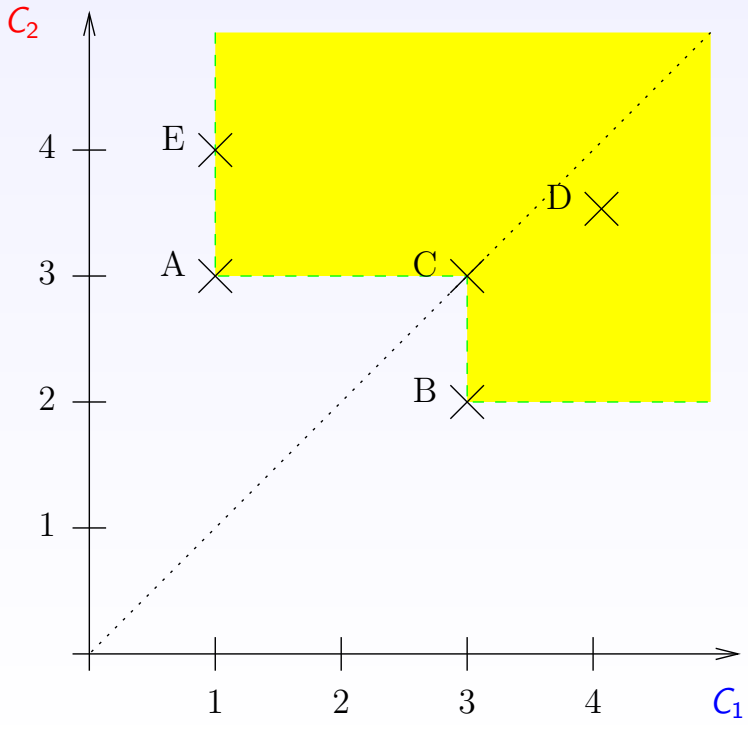
Le coût (ici le temps de complétion) d'un utilisateur ne peut pas être amélioré sans dégrader le coût d'un autre utilisateur.

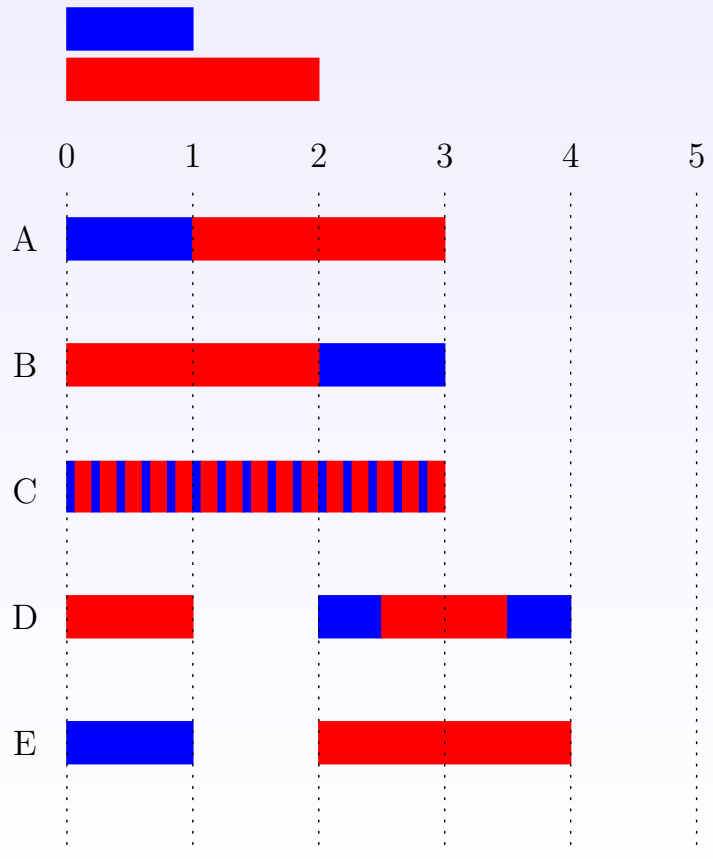




Pareto optimalité

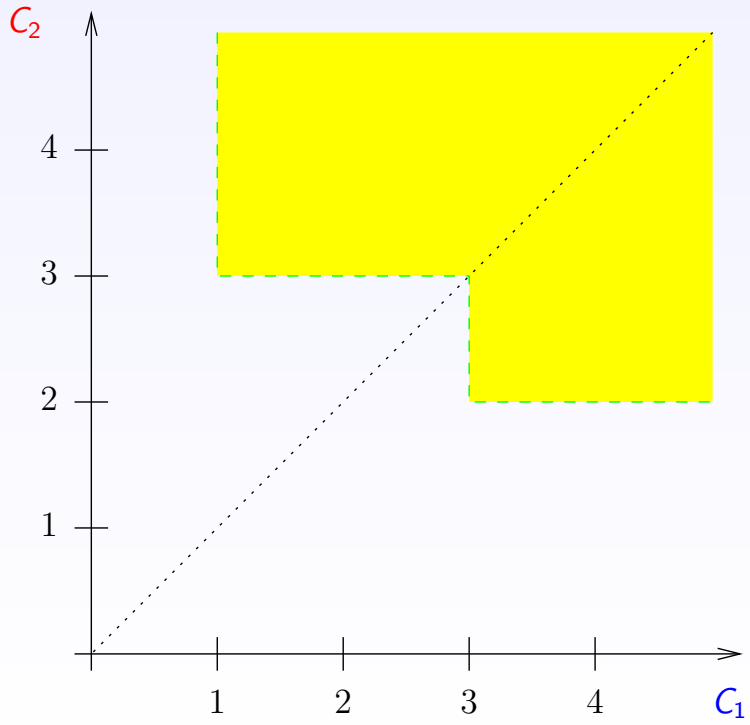
Le coût (ici le temps de complétion) d'un utilisateur ne peut pas être amélioré sans dégrader le coût d'un autre utilisateur.





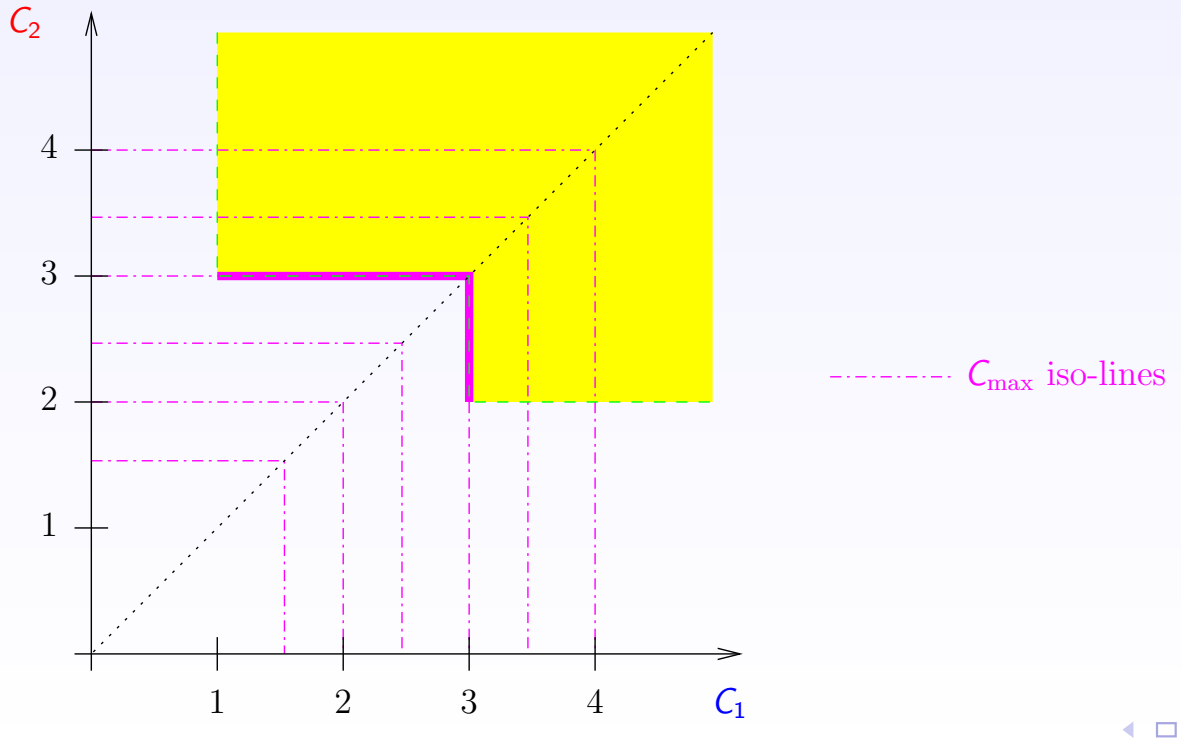
Pareto optimalité

Le coût (ici le temps de complétion) d'un utilisateur ne peut pas être amélioré sans dégrader le coût d'un autre utilisateur.



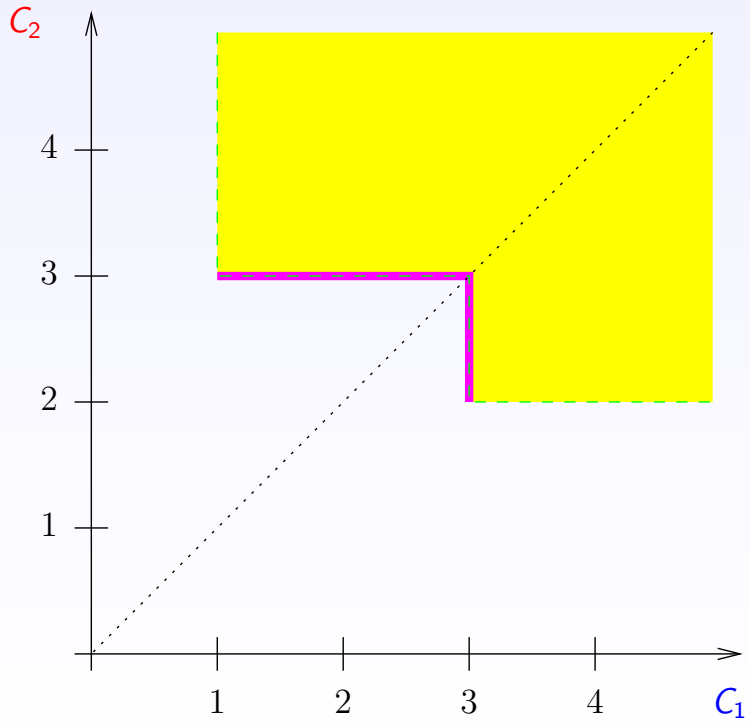
Pareto optimalité

Le coût (ici le temps de complétion) d'un utilisateur ne peut pas être amélioré sans dégrader le coût d'un autre utilisateur.



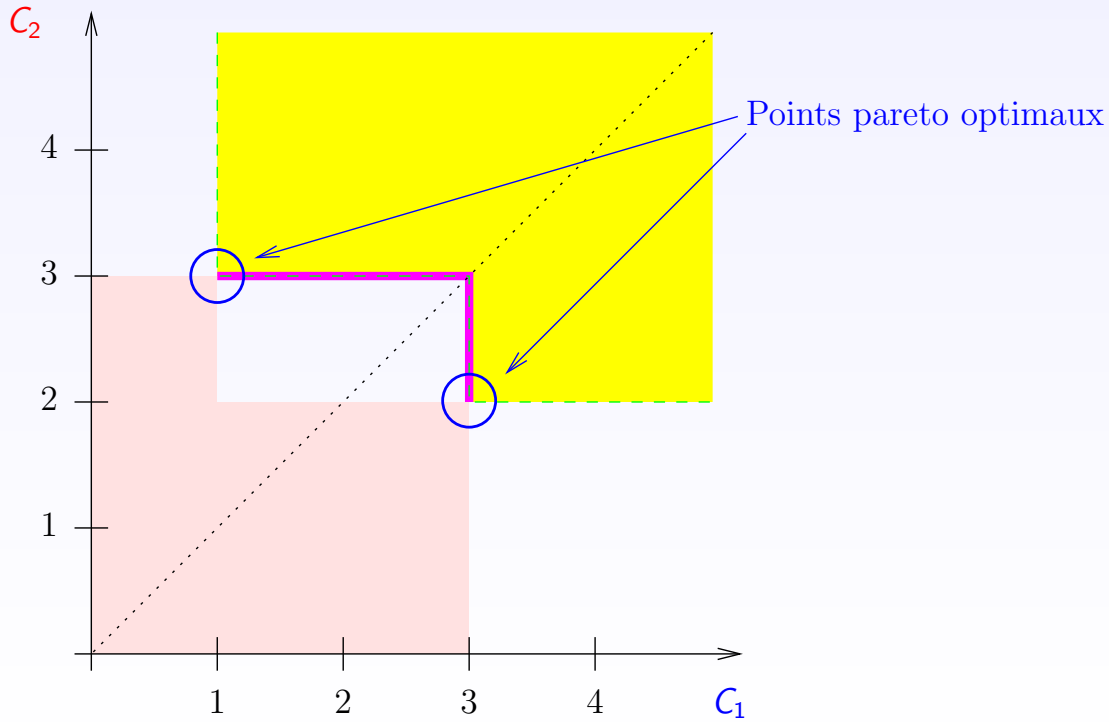
Pareto optimalité

Le coût (ici le temps de complétion) d'un utilisateur ne peut pas être amélioré sans dégrader le coût d'un autre utilisateur.



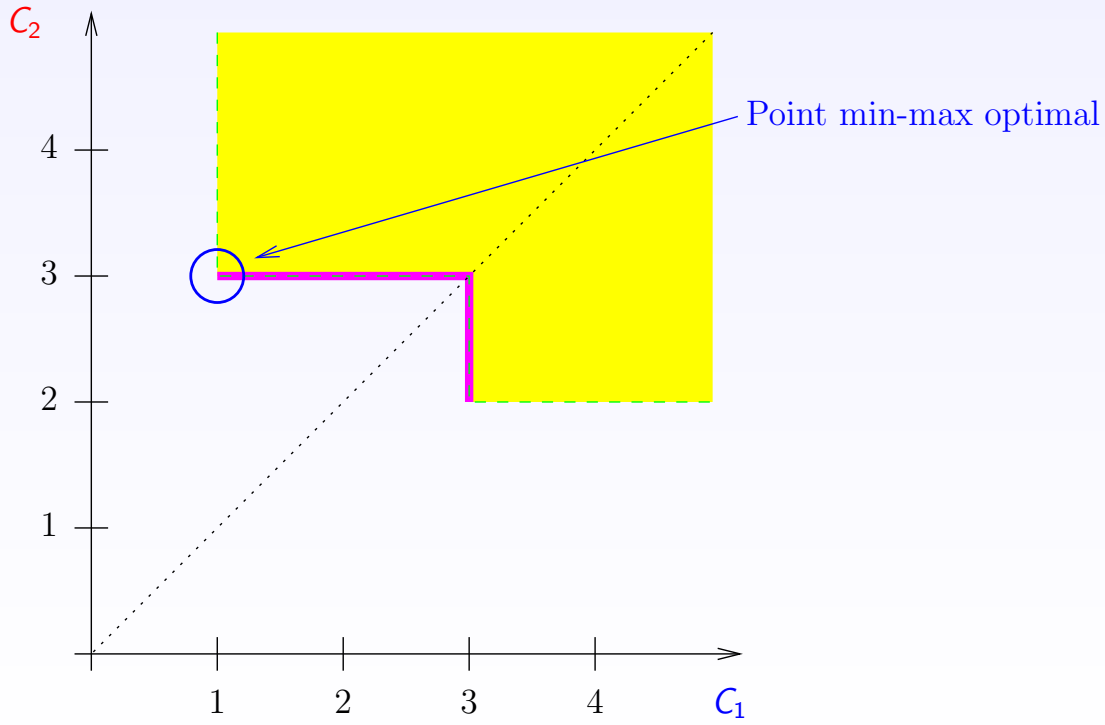
Pareto optimalité

Le coût (ici le temps de complétion) d'un utilisateur ne peut pas être amélioré sans dégrader le coût d'un autre utilisateur.



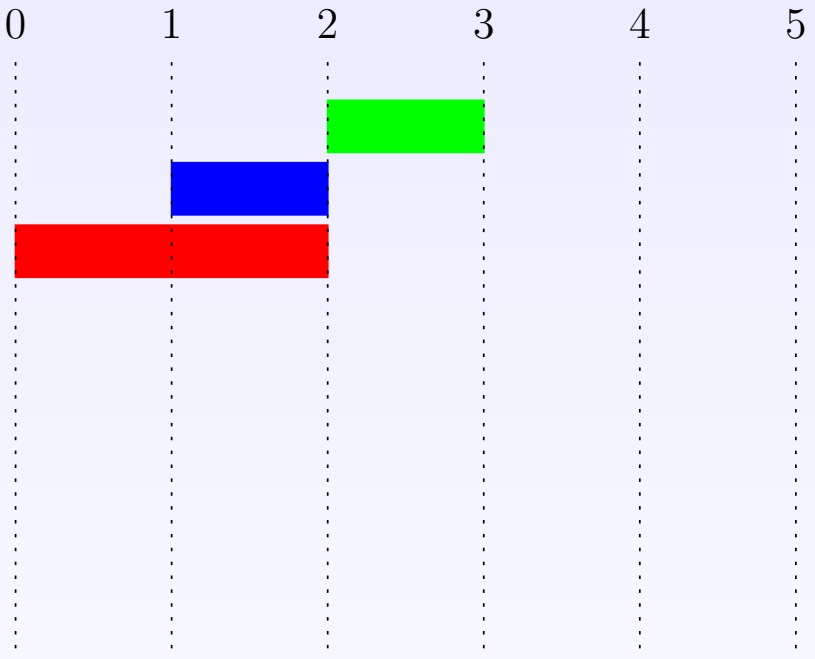
Pareto optimalité

Le coût (ici le temps de complétion) d'un utilisateur ne peut pas être amélioré sans dégrader le coût d'un autre utilisateur.

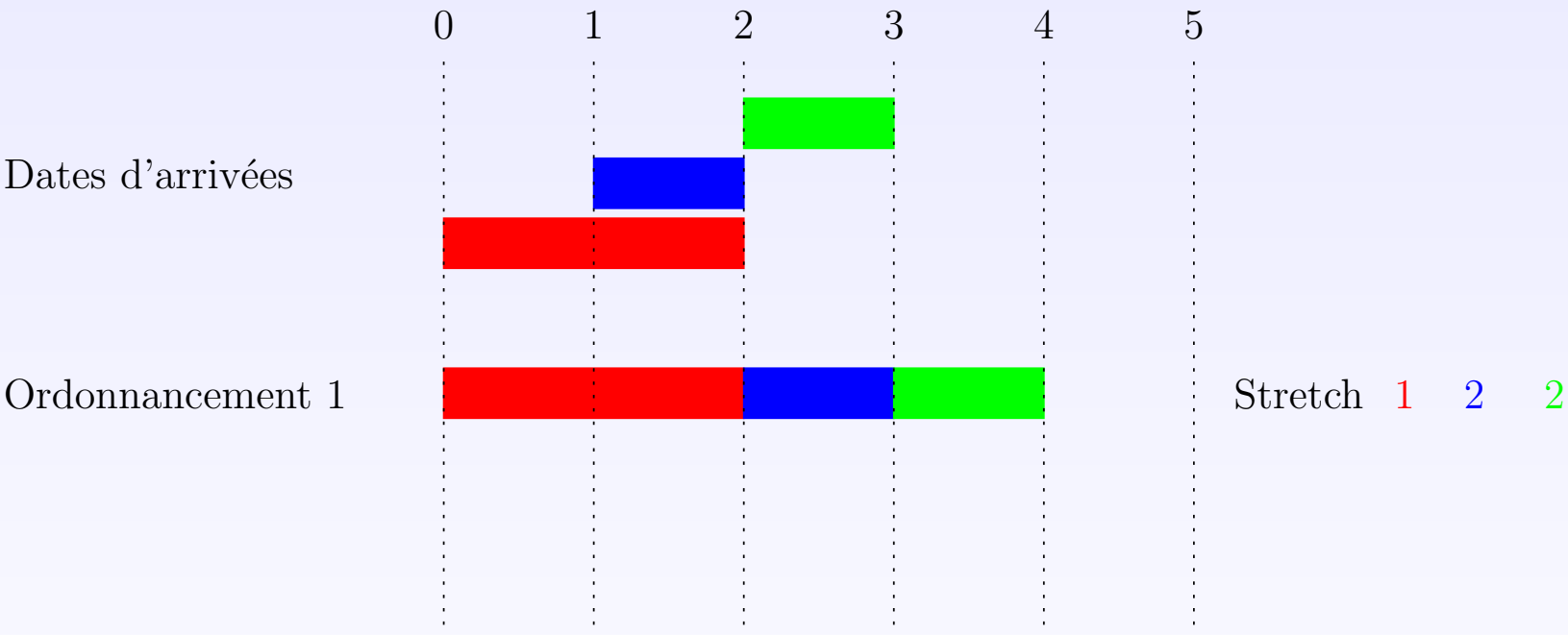


Cas du max-stretch (1)

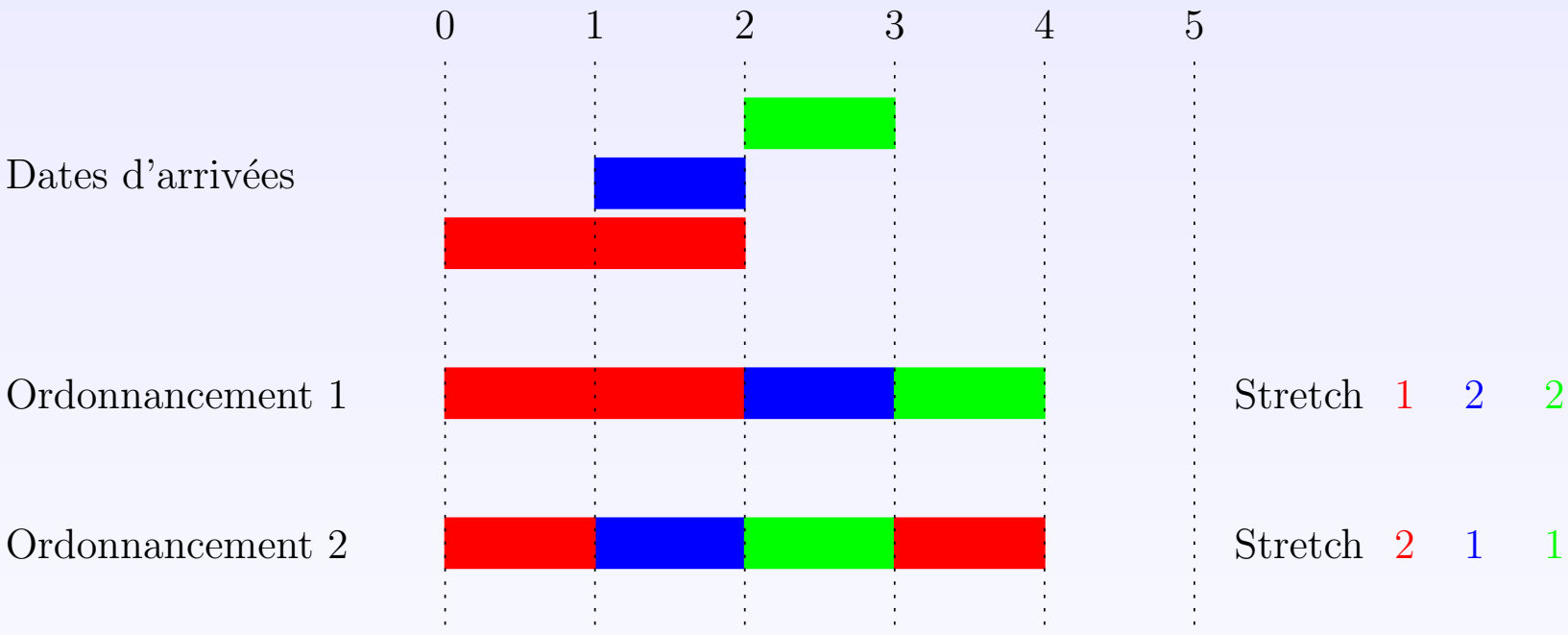
Dates d'arrivées



Cas du max-stretch (1)



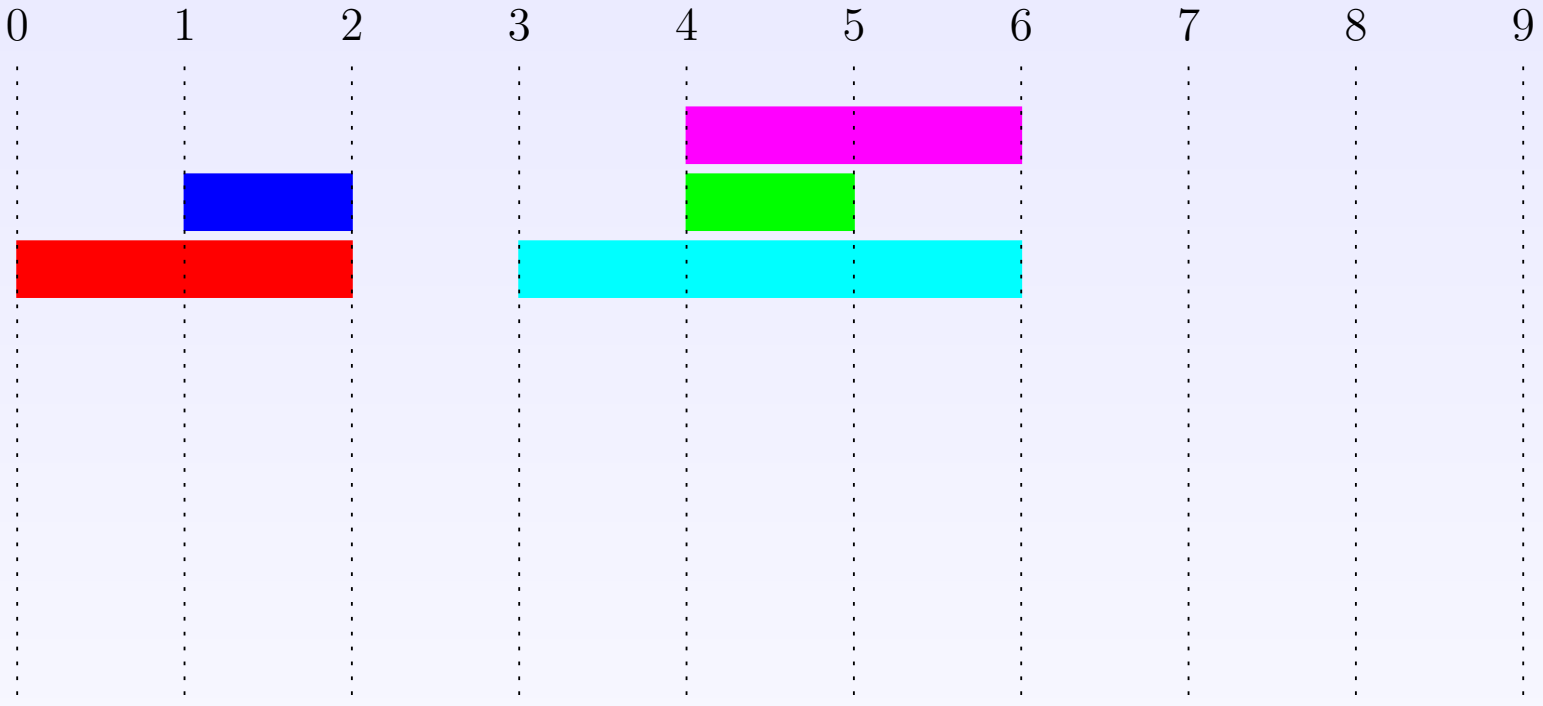
Cas du max-stretch (1)



Ordonnancement 1 et Ordonnancement 2 sont pareto optimaux

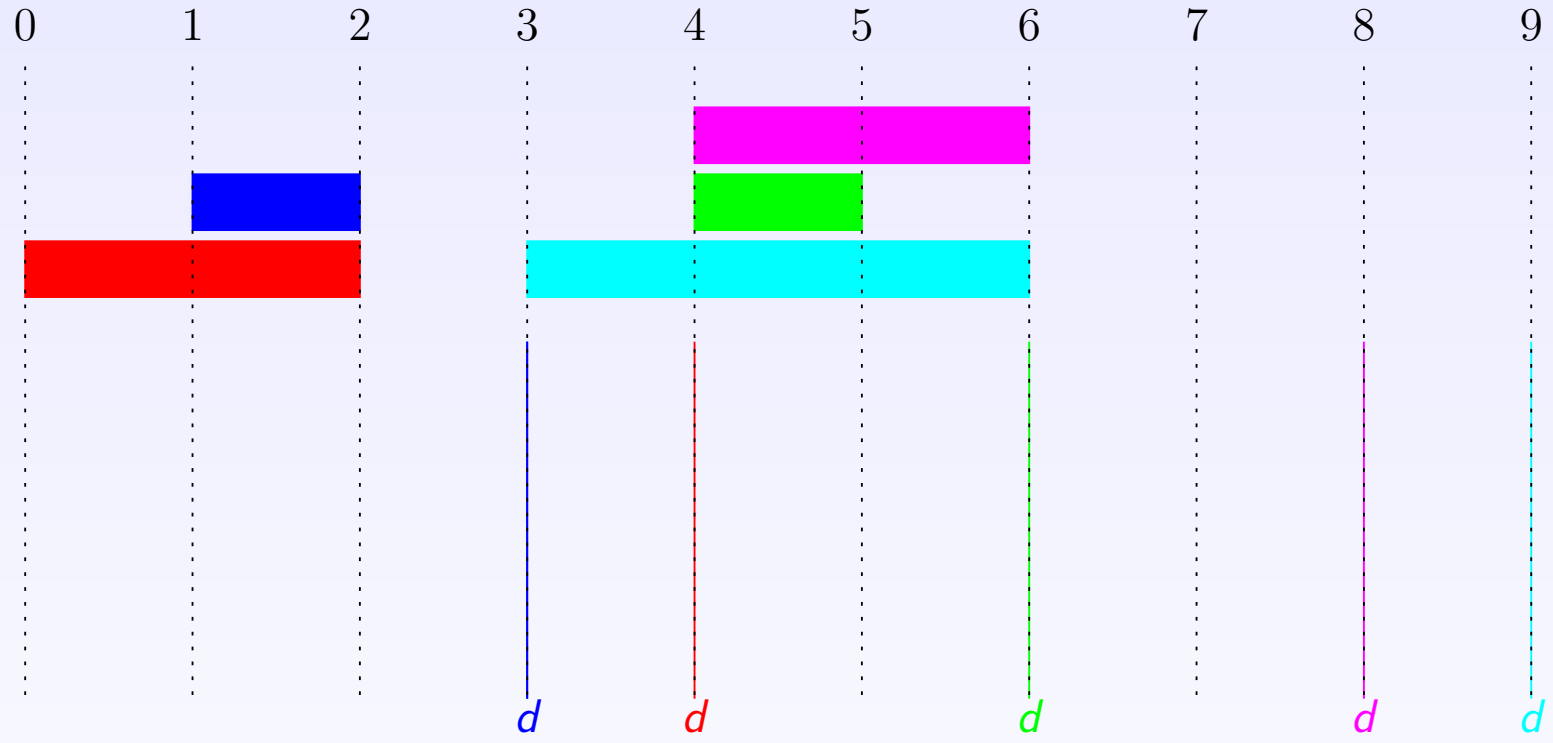
Ordonnancement 2 est la solution de max-min

Cas du max-stretch (2)



Calcul du max-stretch optimal : 2.

Cas du max-stretch (2)

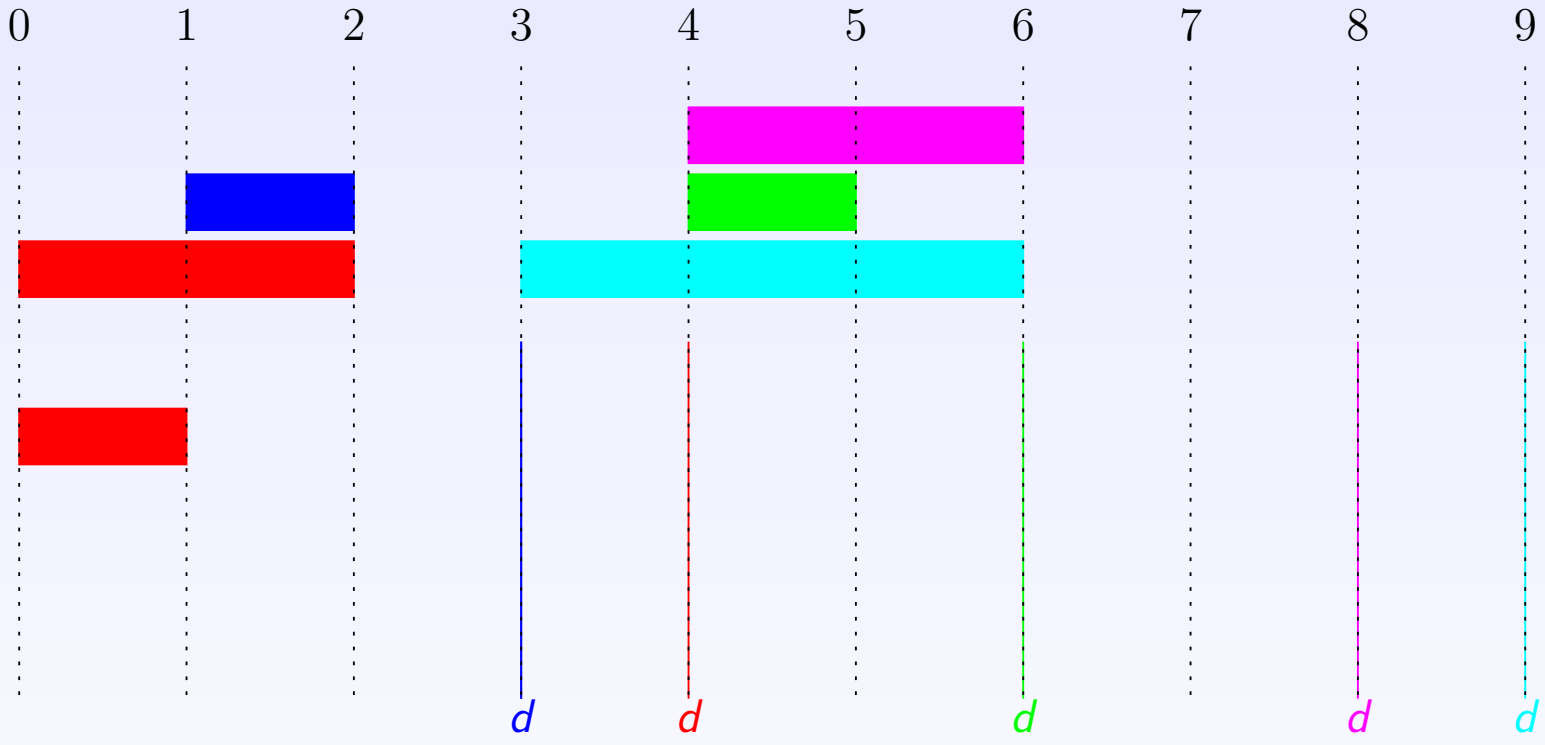


Calcul du max-stretch optimal : 2.

Définition d'une date butoir par tâche.

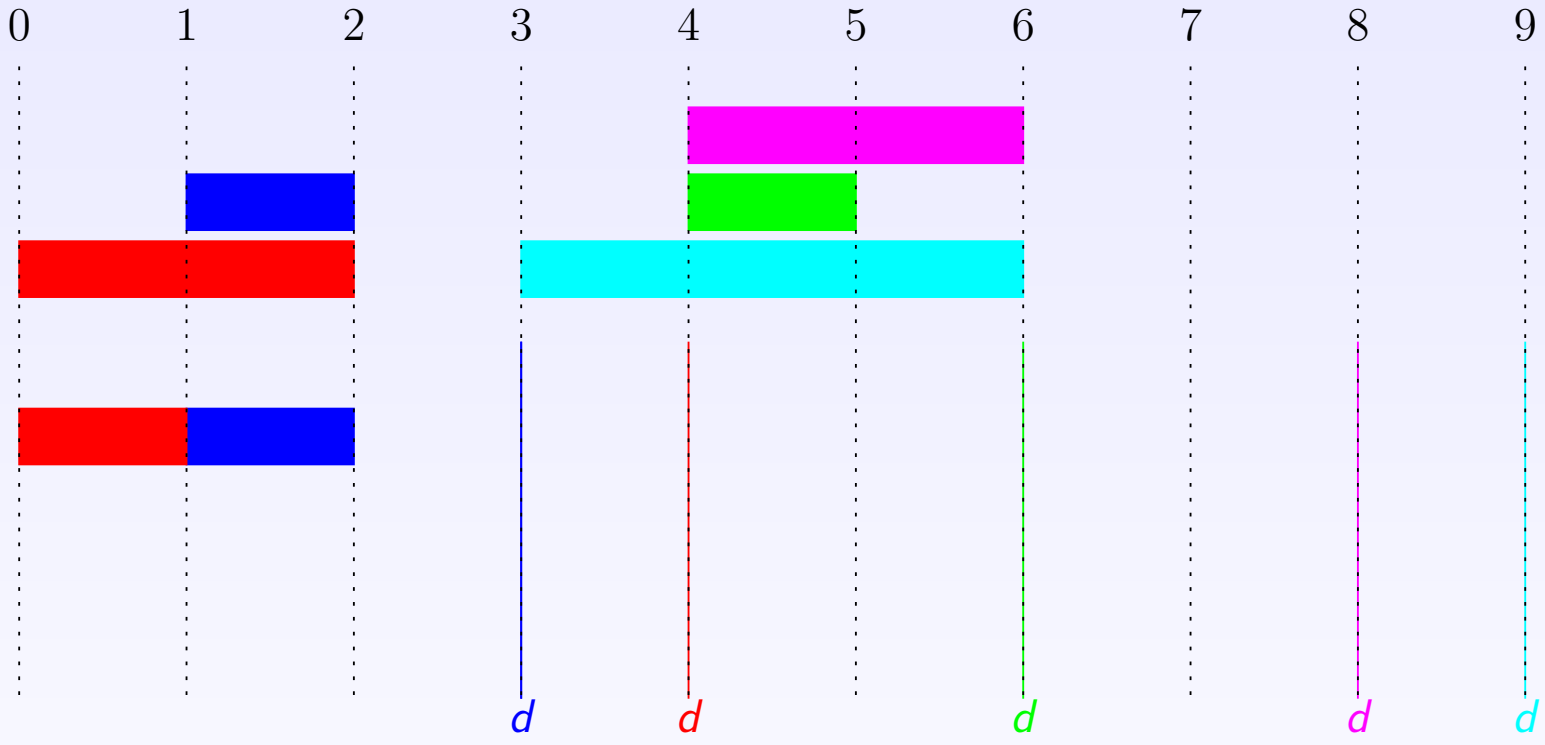
Ordonnancement des tâches *Earliest deadline first*.

Cas du max-stretch (2)



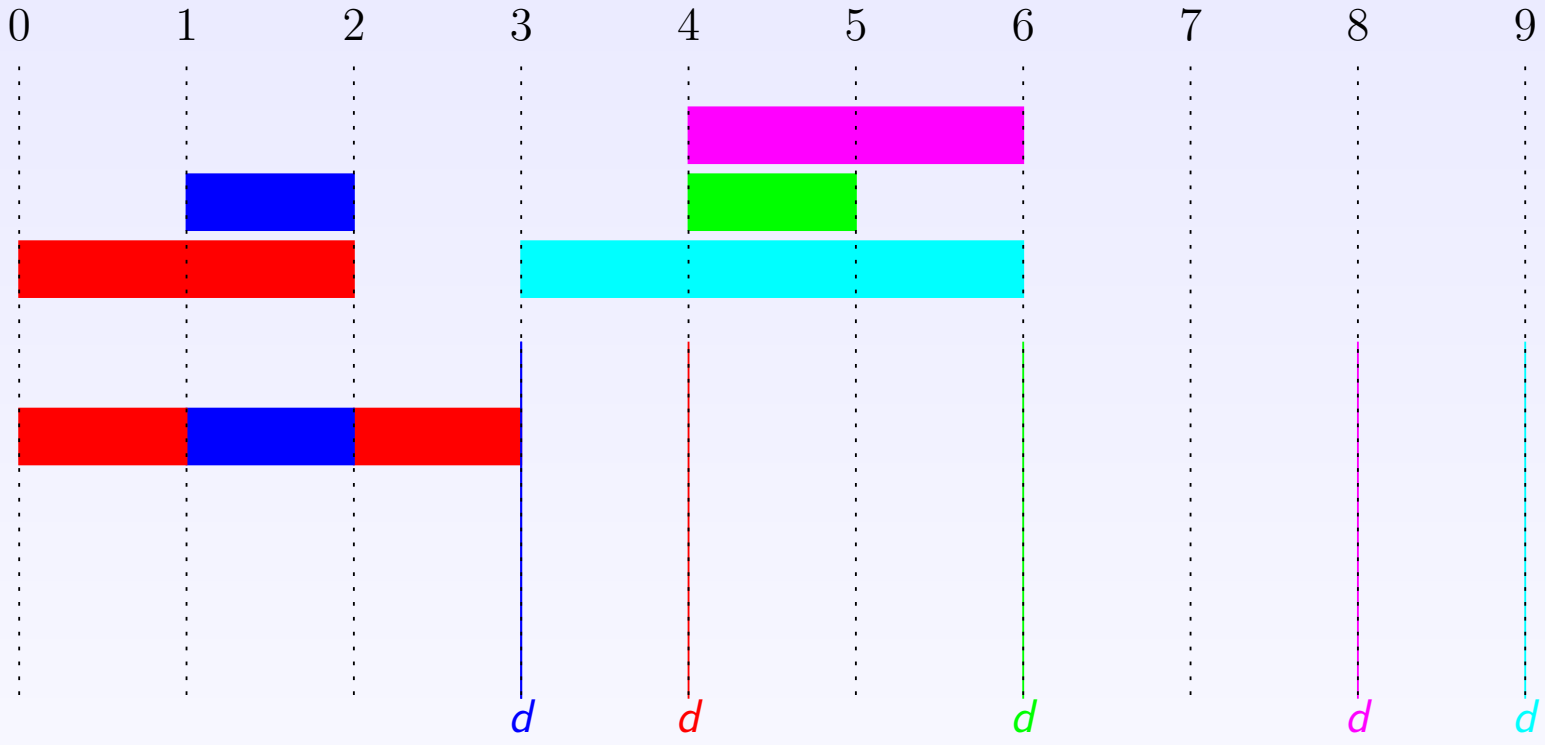
Ordonnancement des tâches *Earliest deadline first*.

Cas du max-stretch (2)



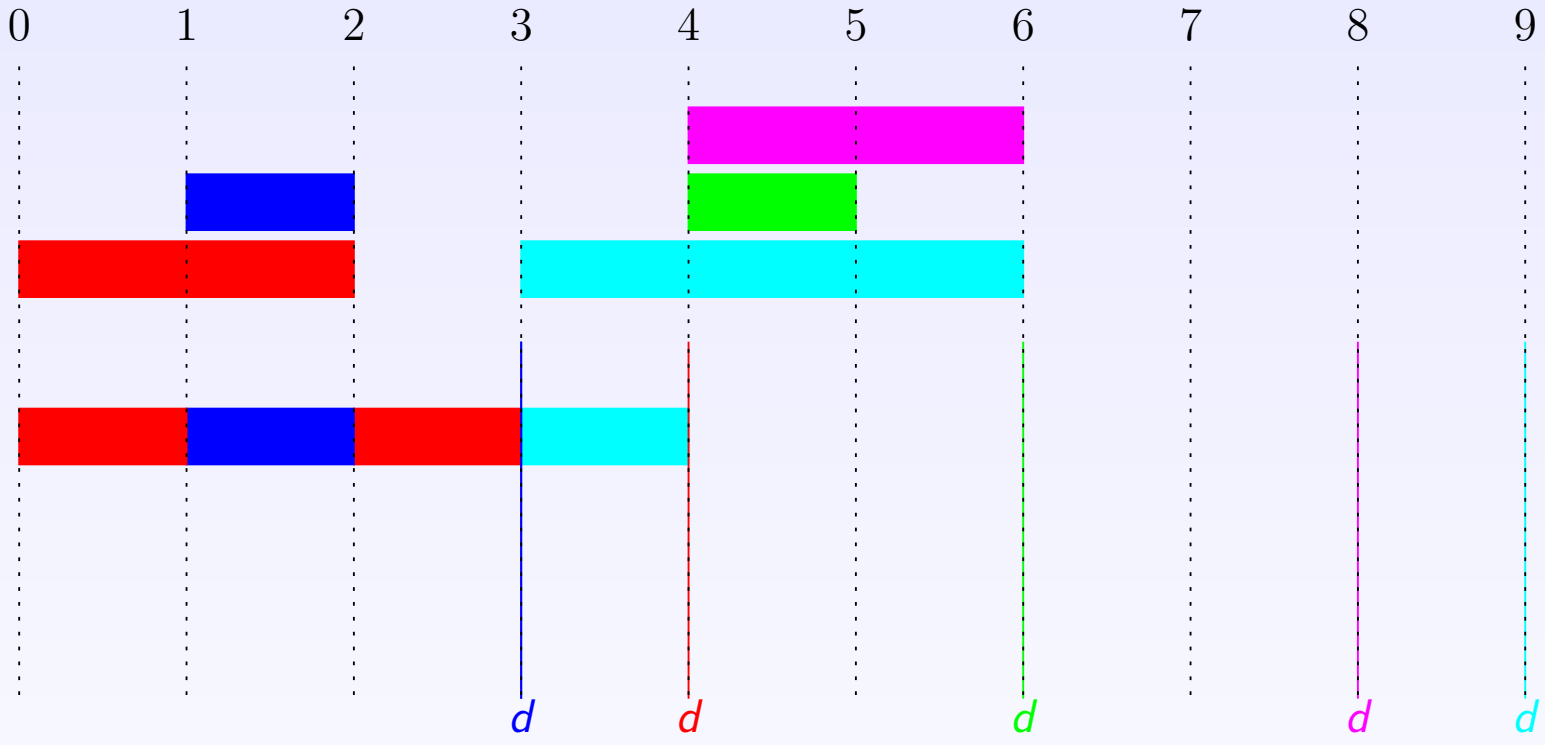
Ordonnancement des tâches *Earliest deadline first*.

Cas du max-stretch (2)



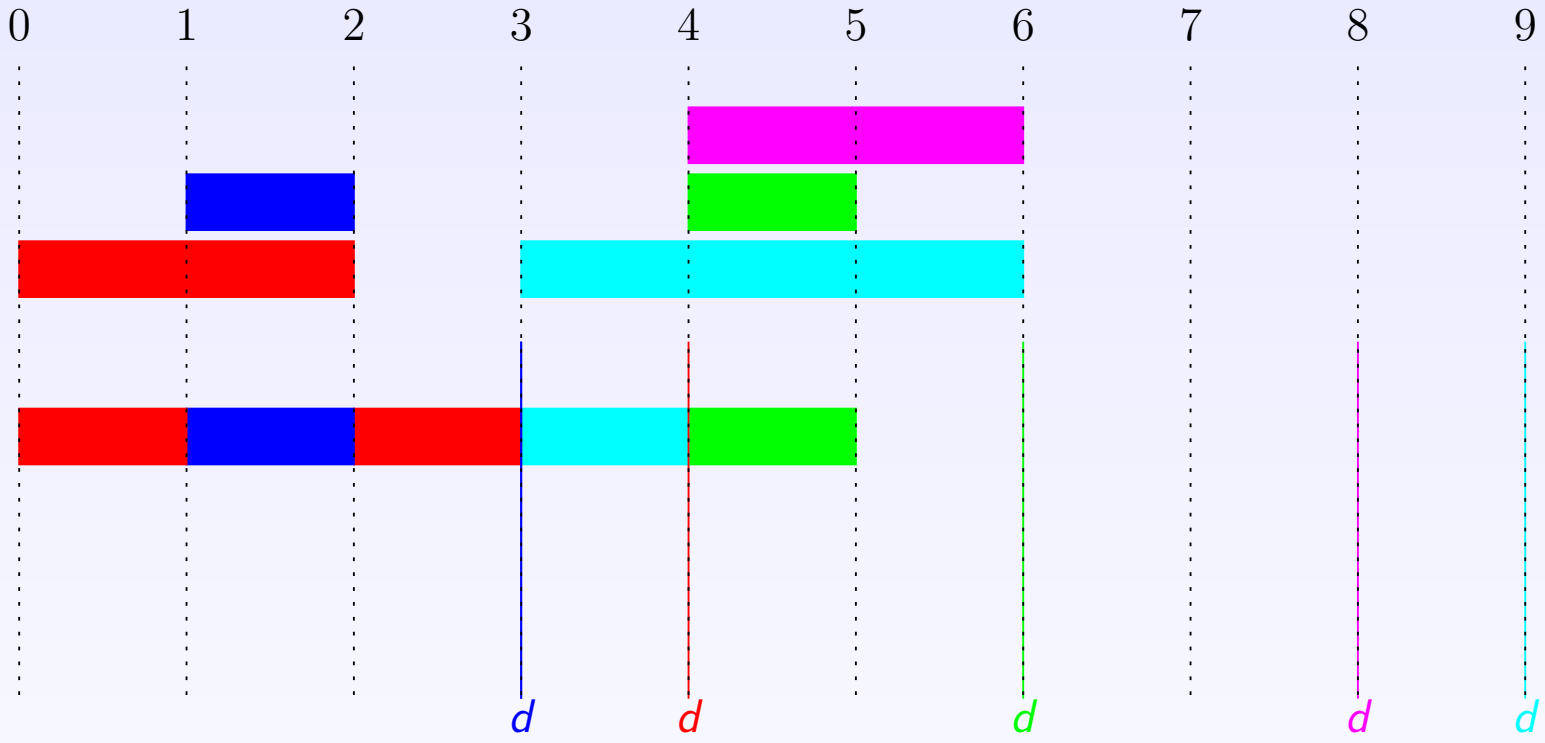
Ordonnancement des tâches *Earliest deadline first*.

Cas du max-stretch (2)



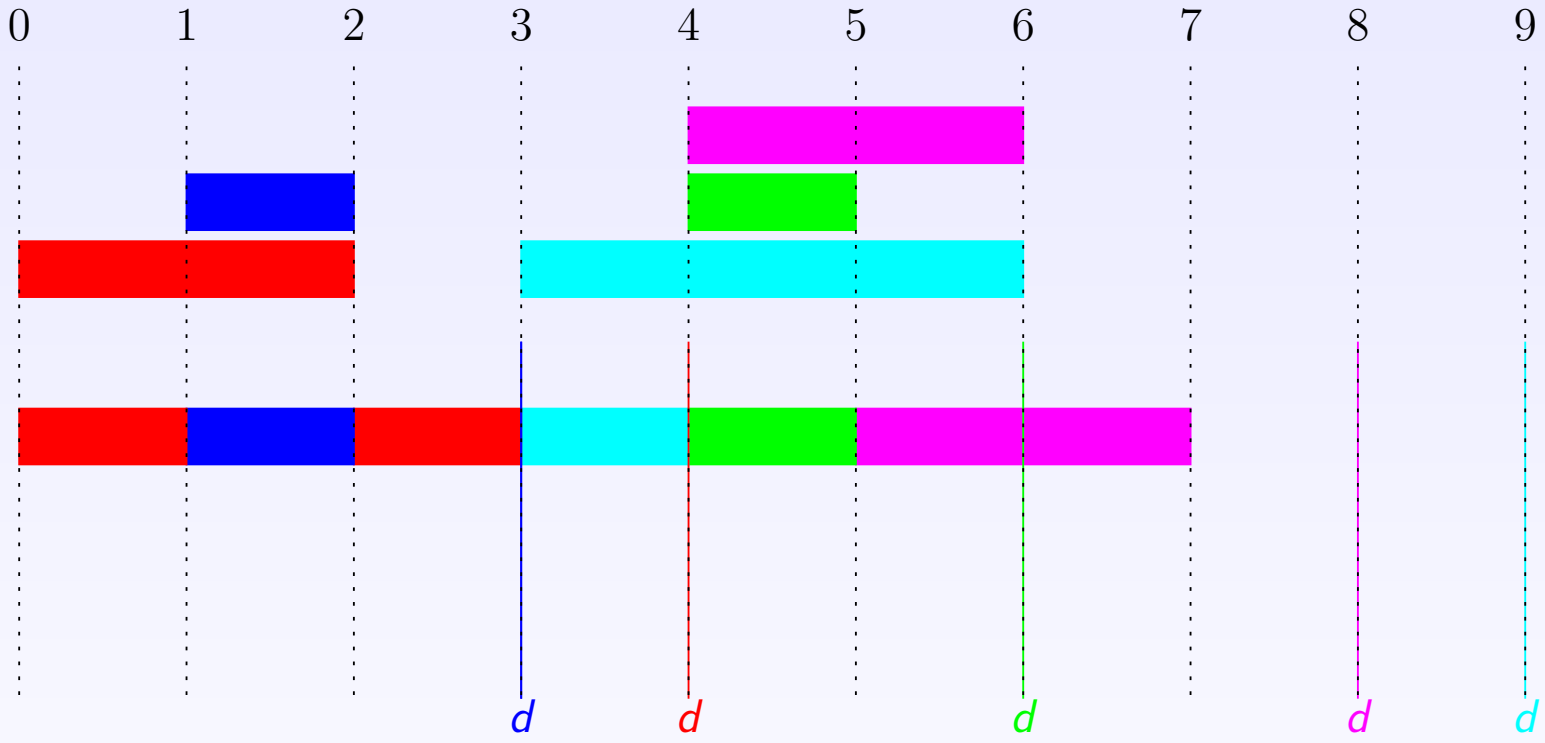
Ordonnancement des tâches *Earliest deadline first*.

Cas du max-stretch (2)



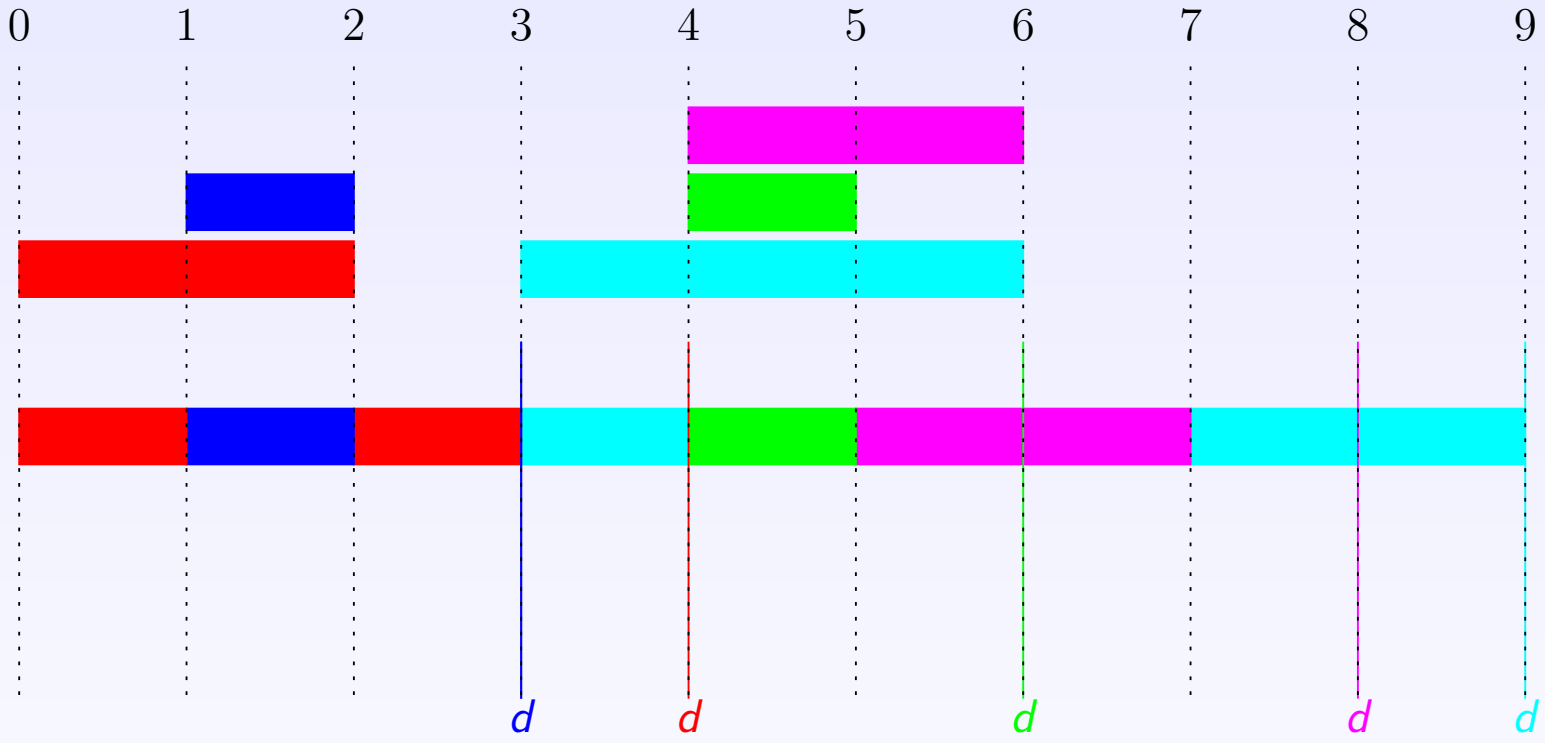
Ordonnancement des tâches *Earliest deadline first*.

Cas du max-stretch (2)



Ordonnancement des tâches *Earliest deadline first*.

Cas du max-stretch (2)

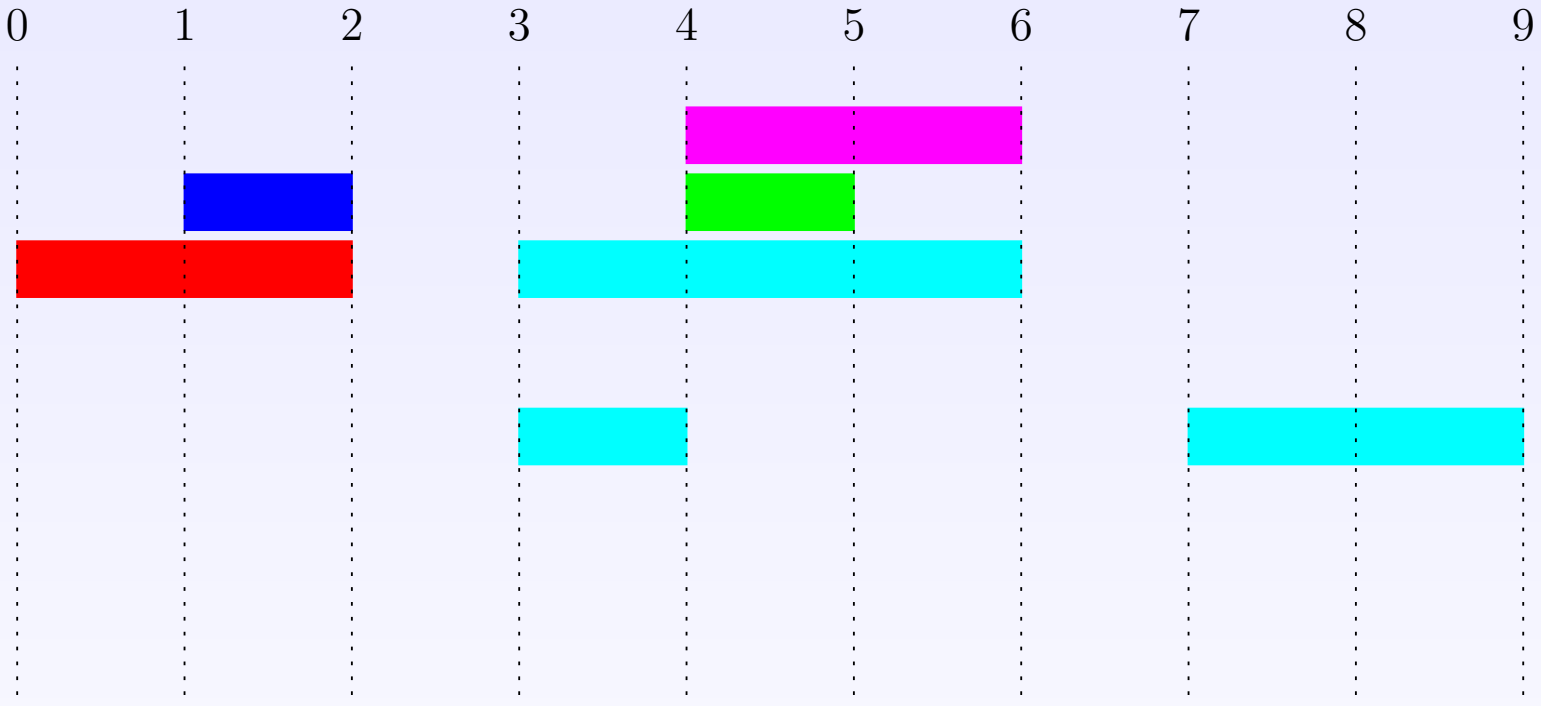


Ordonnancement des tâches *Earliest deadline first*.

Si date de complétion = date butoir alors, quel que soit l'ordonnancement, le stretch de cette tâche est égal au stretch maximal.

On fixe les tâches non optimisables et on recommence récursivement.

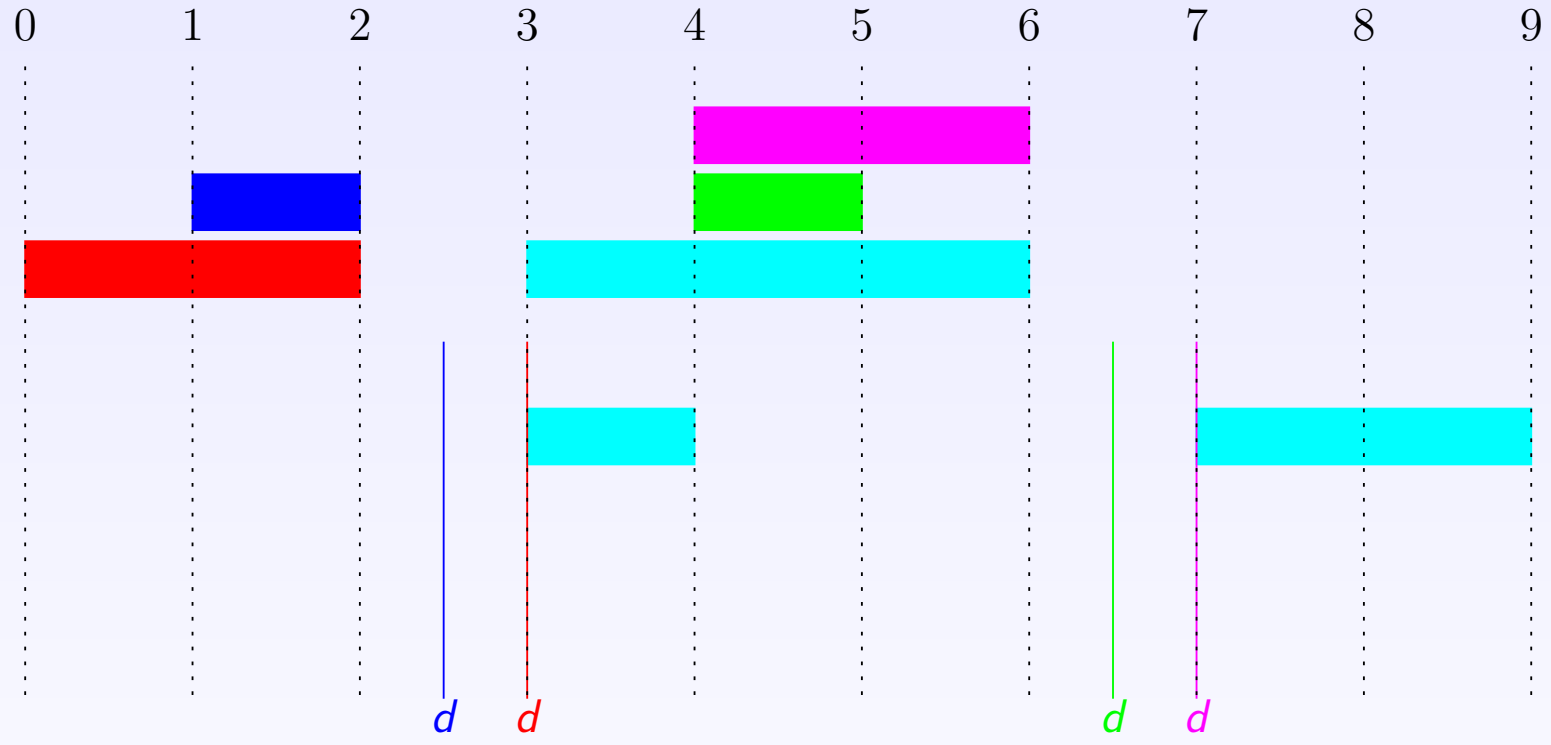
Cas du max-stretch (2)



On fixe les tâches non optimisables et on recommence récursivement.

Max-stretch des tâches restantes : 1,5.

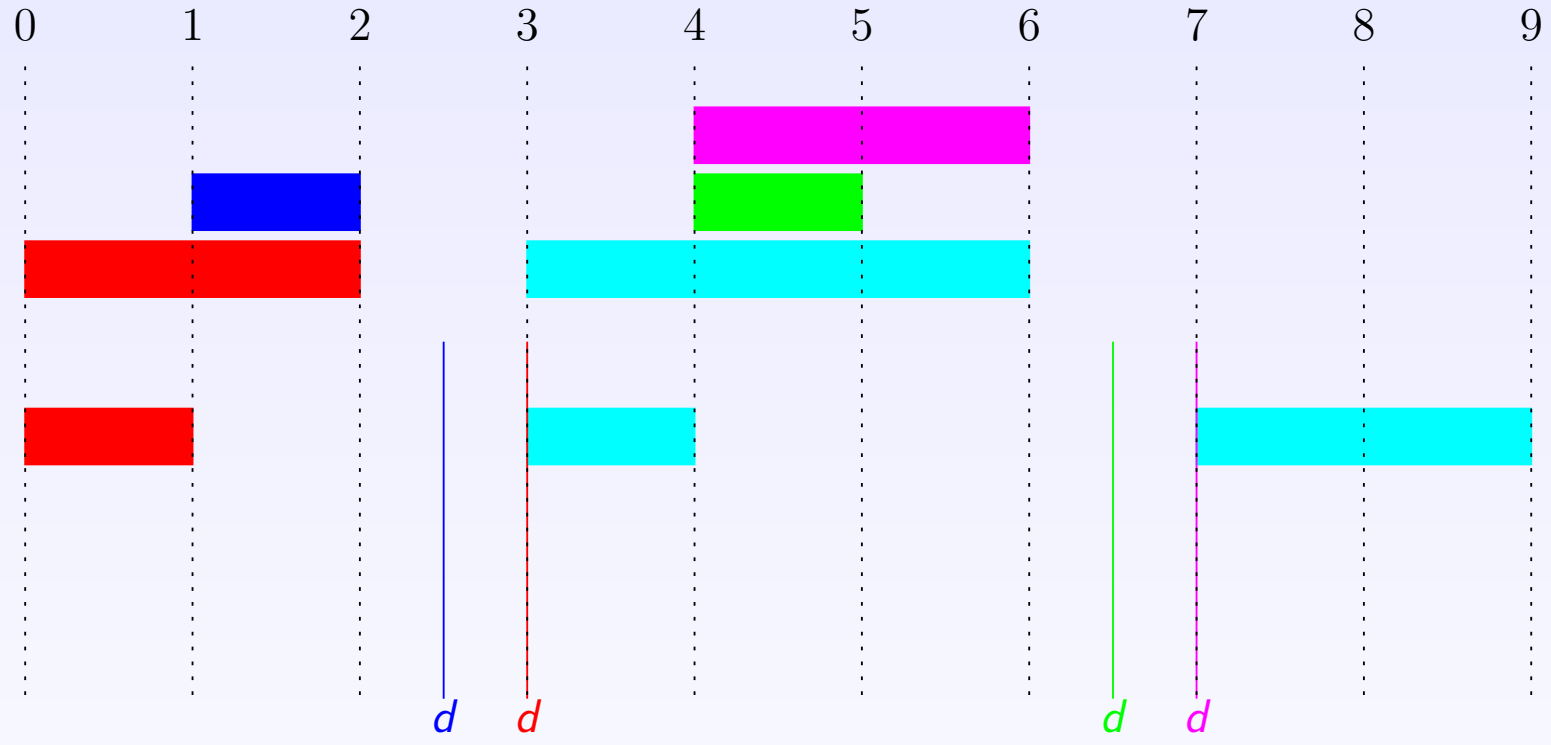
Cas du max-stretch (2)



On fixe les tâches non optimisables et on recommence récursivement.

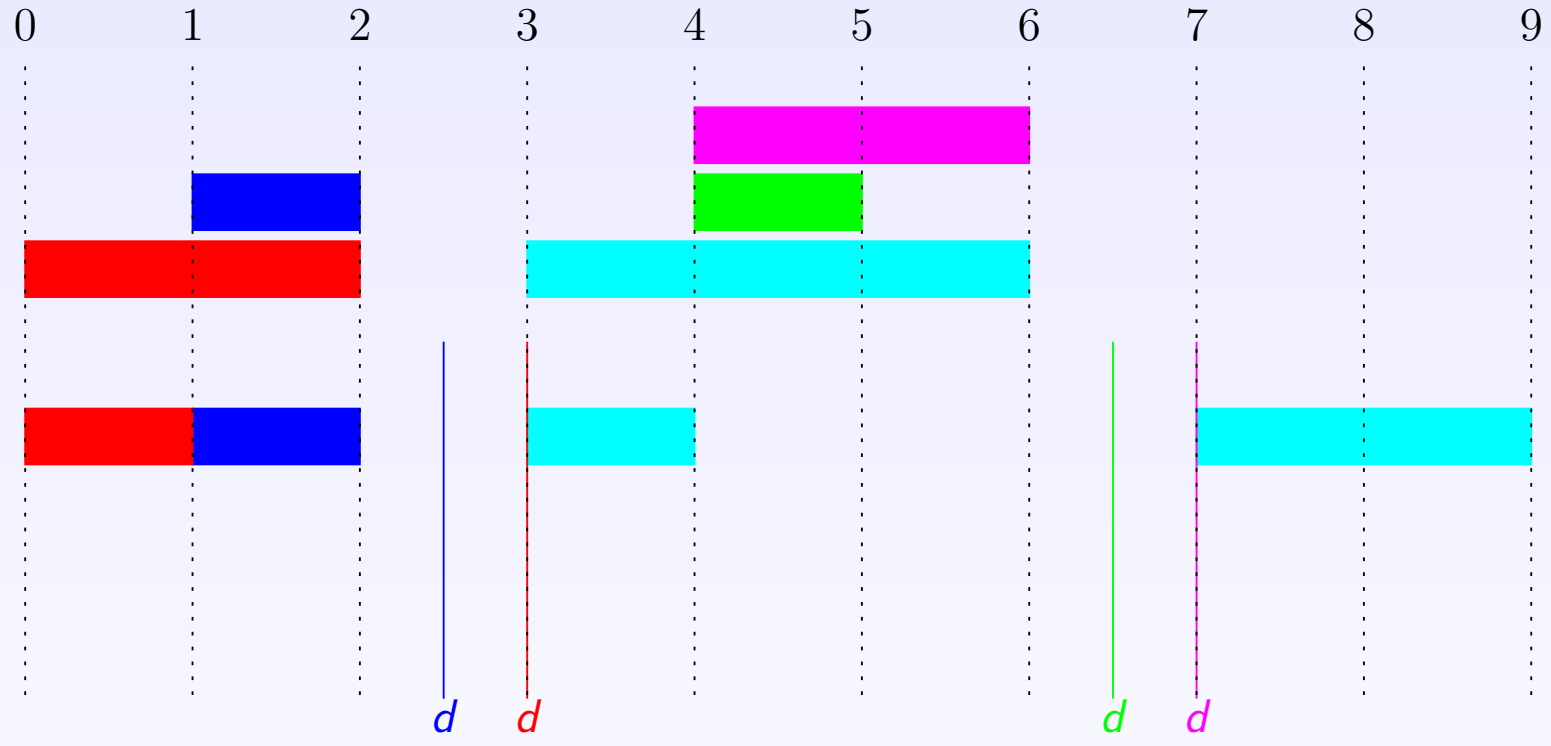
Max-stretch des tâches restantes : 1,5.

Cas du max-stretch (2)



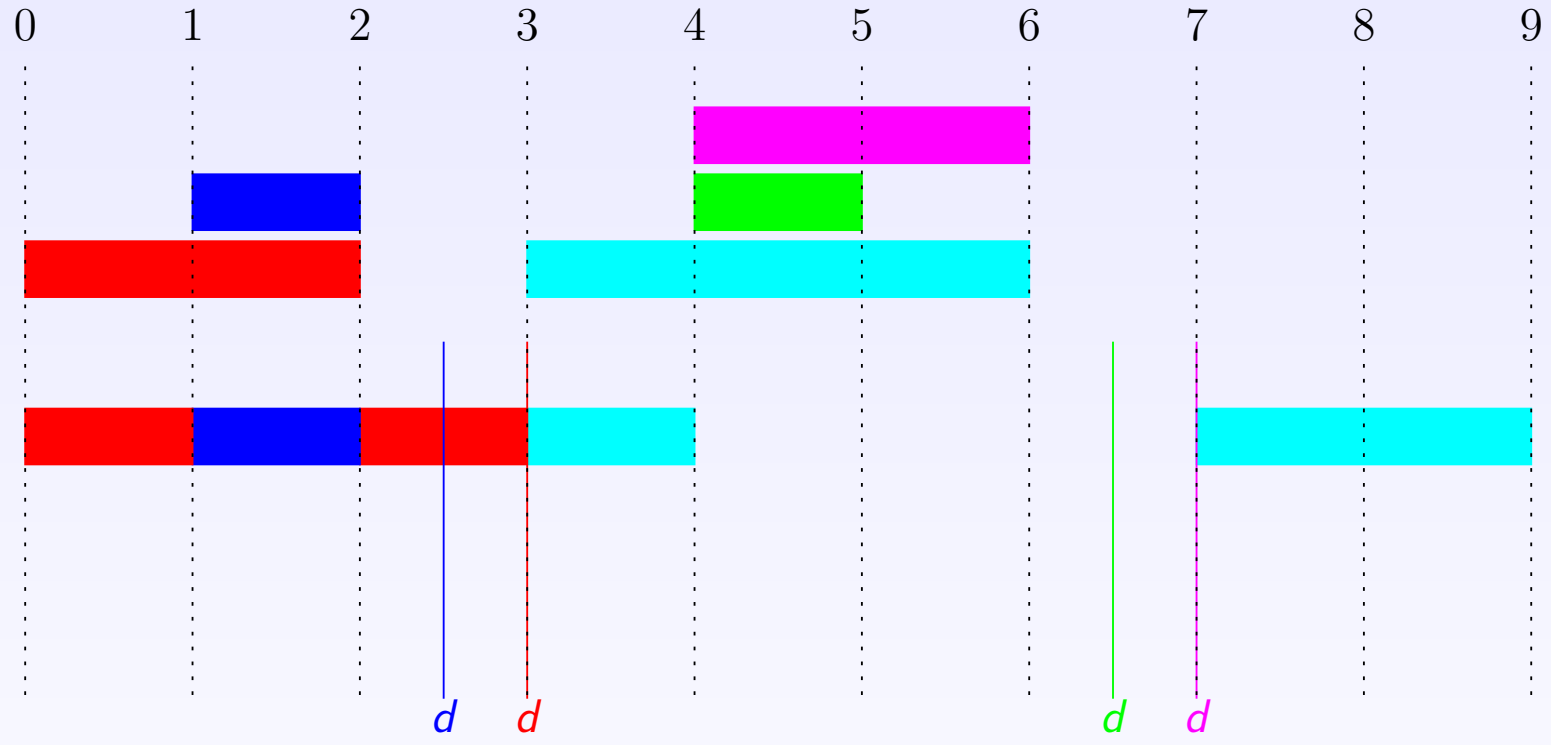
Ordonnancement des tâches *Earliest deadline first*.

Cas du max-stretch (2)



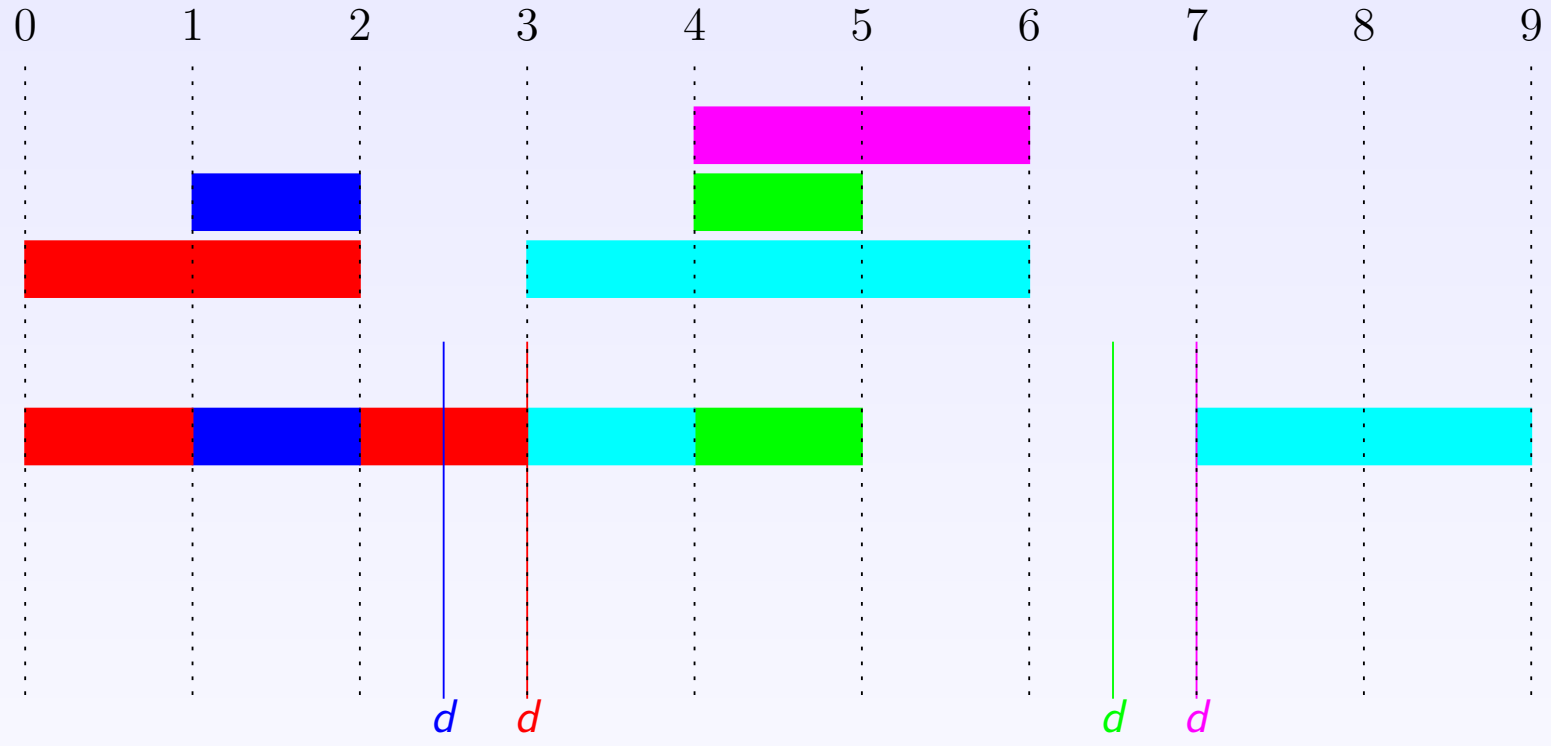
Ordonnancement des tâches *Earliest deadline first*.

Cas du max-stretch (2)



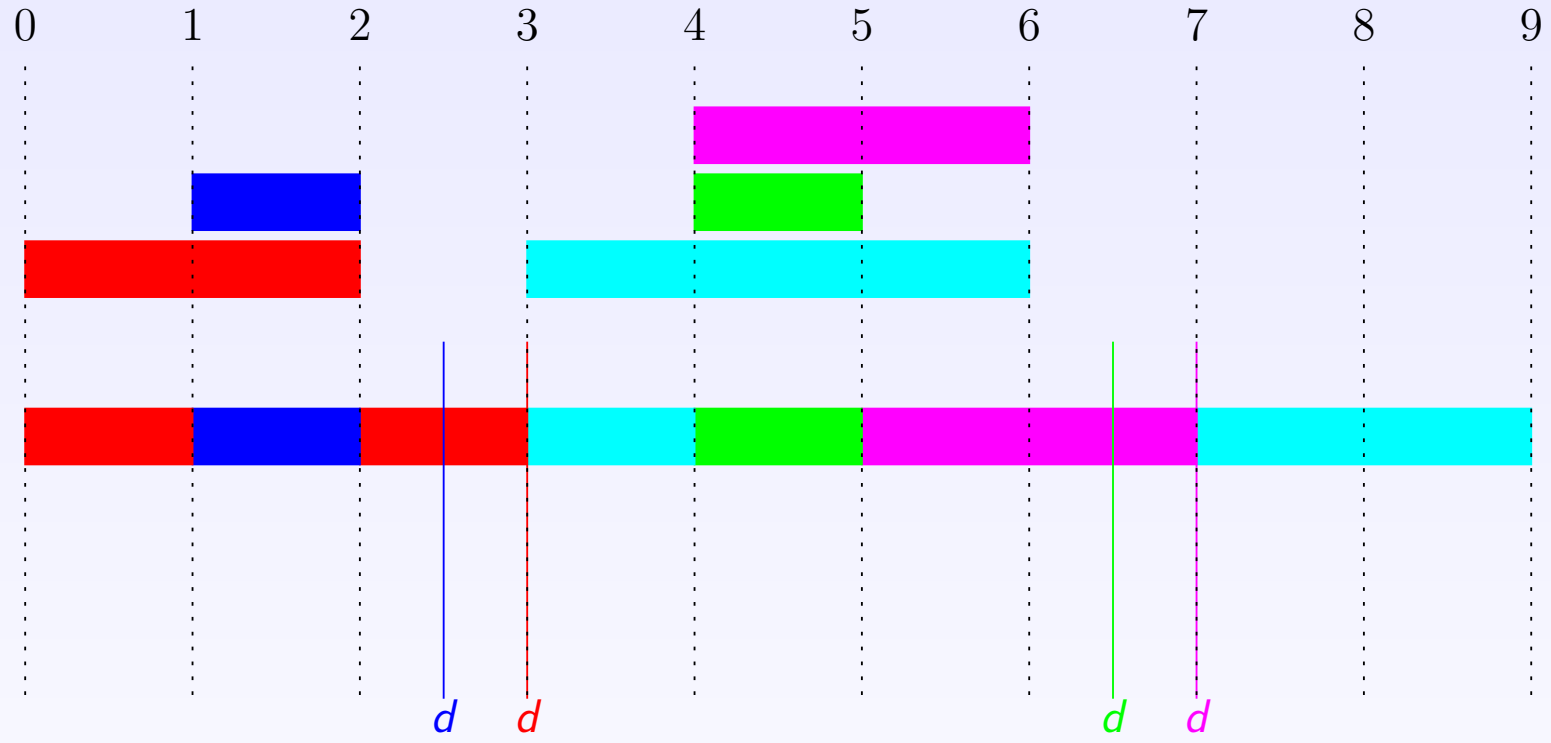
Ordonnancement des tâches *Earliest deadline first*.

Cas du max-stretch (2)



Ordonnancement des tâches *Earliest deadline first*.

Cas du max-stretch (2)

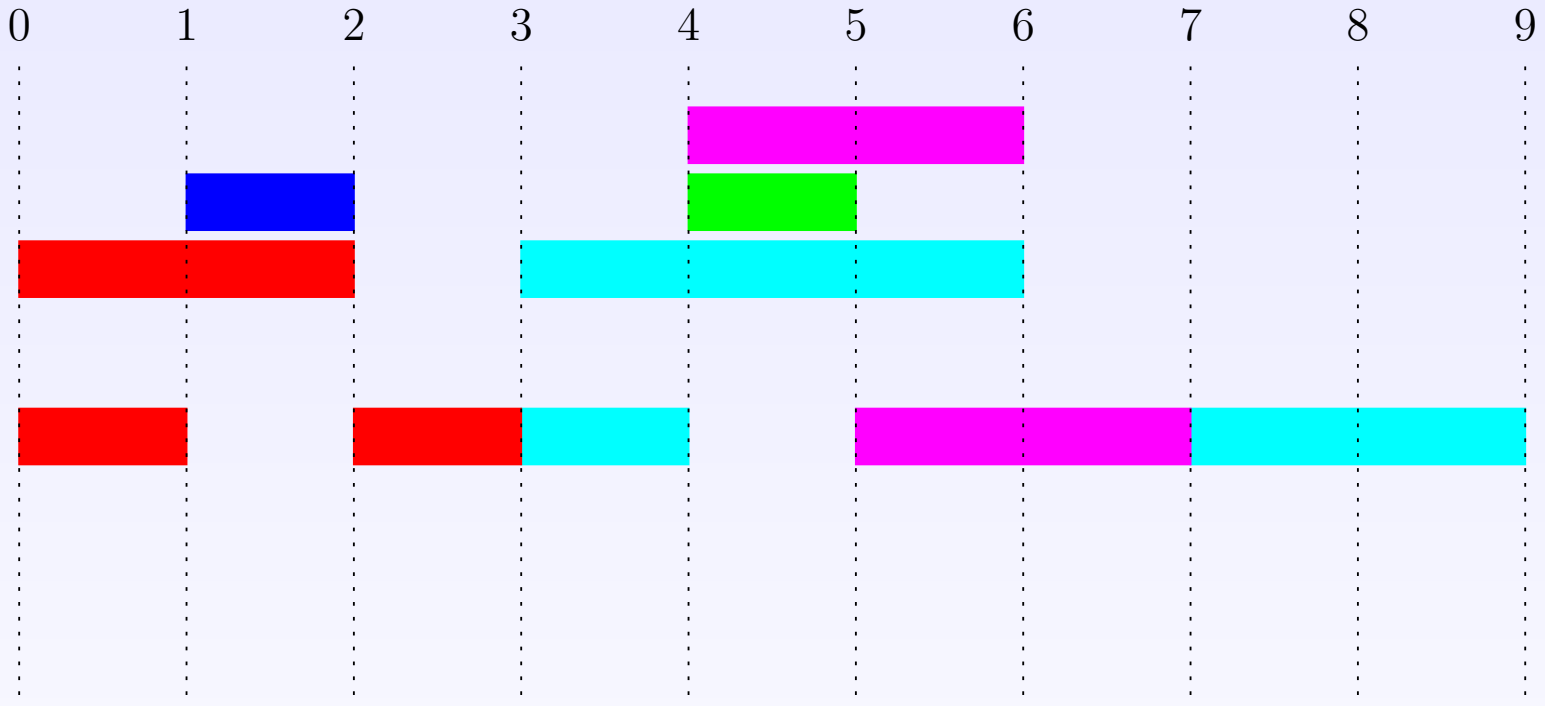


Ordonnancement des tâches *Earliest deadline first*.

Si date de complétion = date butoir alors, quel que soit l'ordonnancement, le stretch de la tâche est égal au stretch maximal.

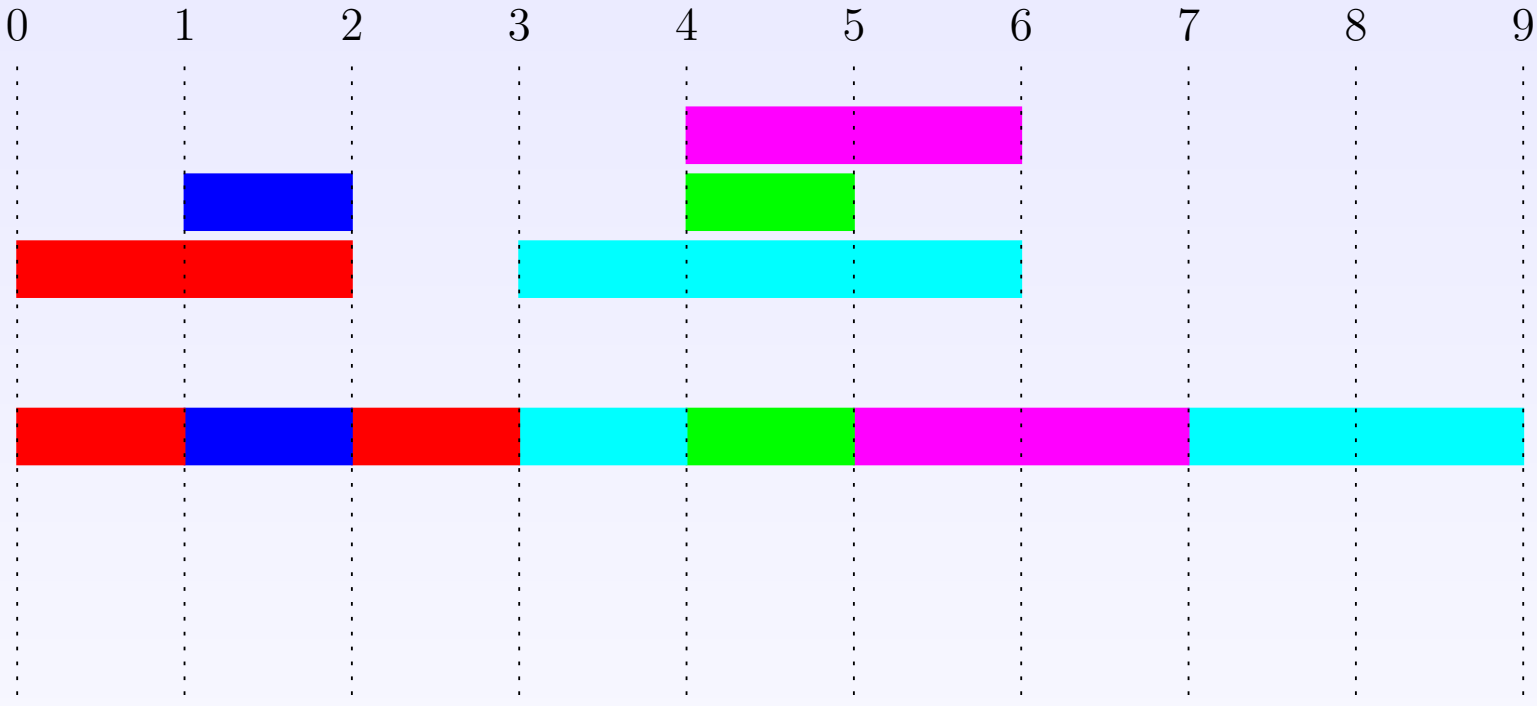
On fixe les tâches non optimisables et on recommence récursivement.

Cas du max-stretch (2)

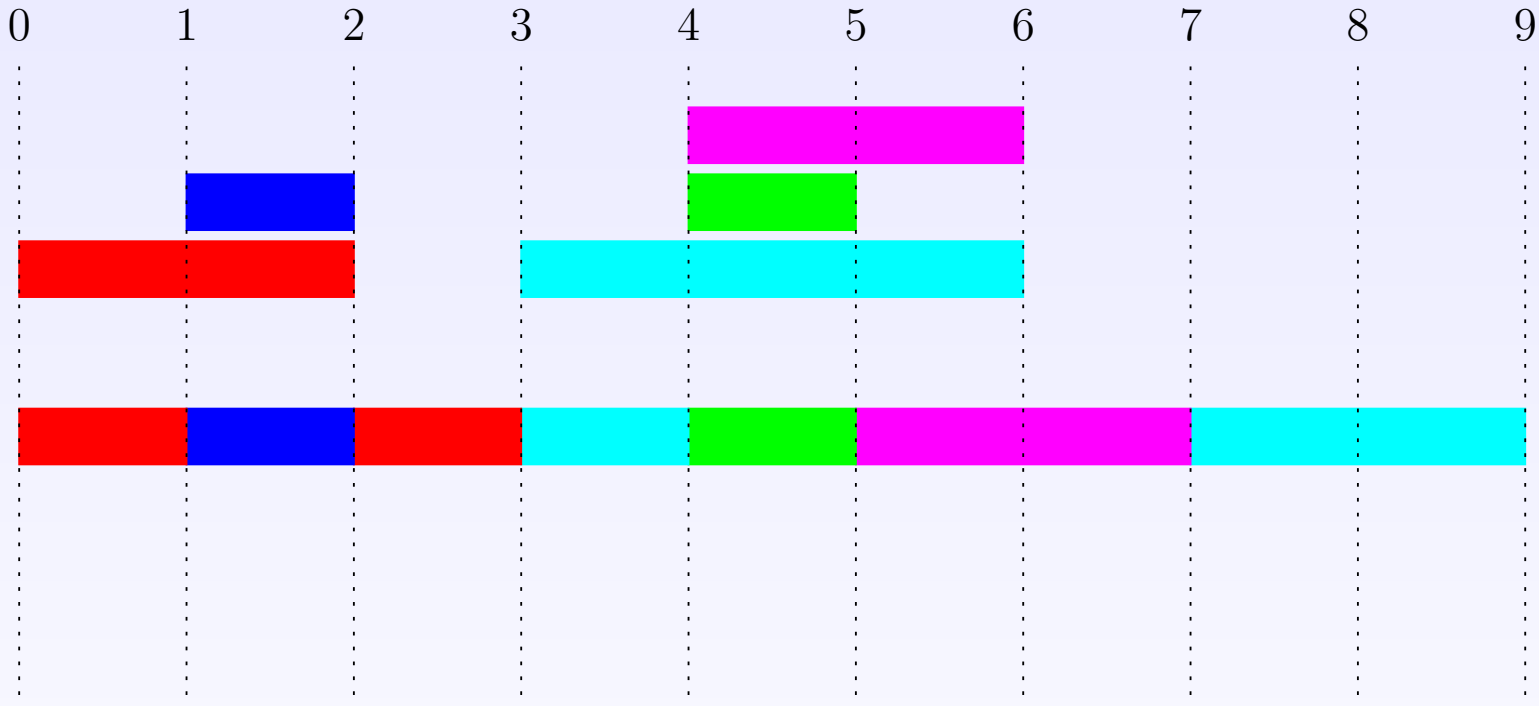


On fixe les tâches non optimisables et on recommence récursivement.

Cas du max-stretch (2)



Cas du max-stretch (2)



Si deux tâches ont même date butoir : on choisit arbitrairement (ça n'a pas d'importance).

Plan de l'exposé

Règles du jeu

- Les caractéristiques d'une tâche ne sont connues que lors de l'arrivée de la tâche dans le système.
- On cherche à minimiser le flot maximal pondéré (poids non spécifiés).
- À un instant donné une seule machine peut exécuter une fraction d'une tâche.

Évaluation de la qualité d'un ordonnancement online

Un algorithme online a un facteur de compétitivité ρ si et seulement si :

Quelque soit l'ensemble de tâches T_1, ..., T_n :

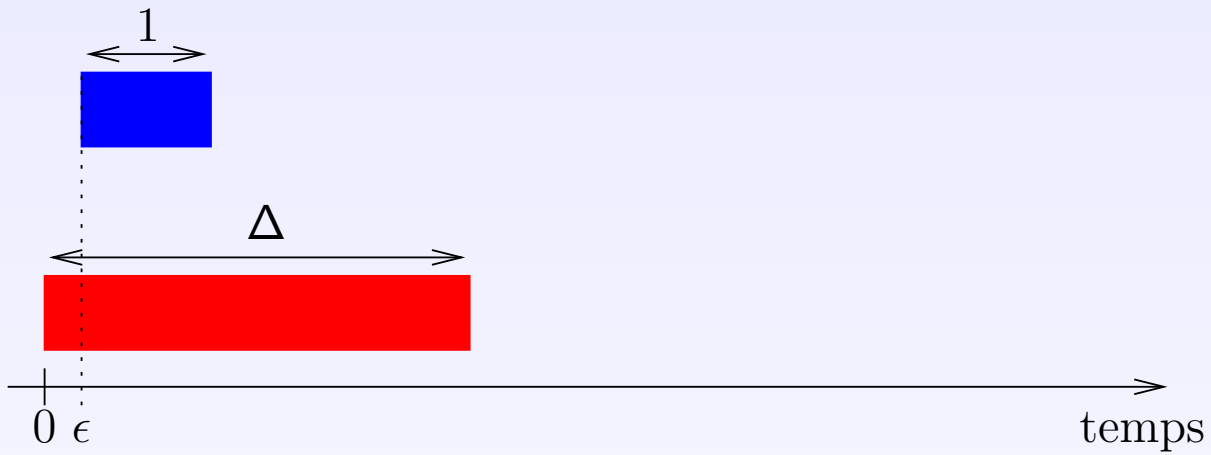
$$\text{Coût ordo online}(T_1, ..., T_N) \leq$$
$$\rho \times \text{Coût ordo optimal offline}(T_1, ..., T_N).$$

Compétitivité de FIFO (1)

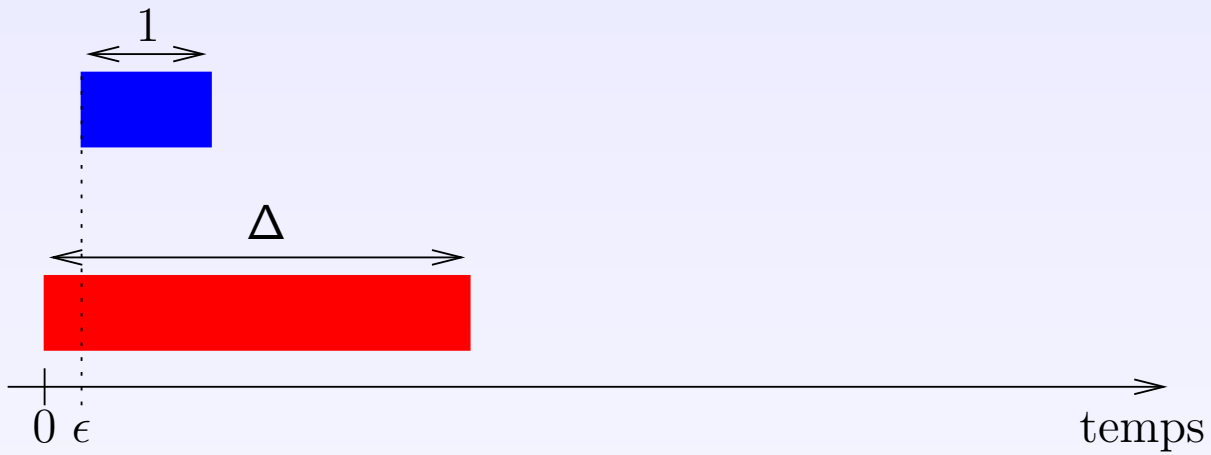
Théorème

FIFO est Δ compétitive pour la minimisation de max-stretch.

Compétitivité de FIFO (2) : au *mieux* Δ compétitive



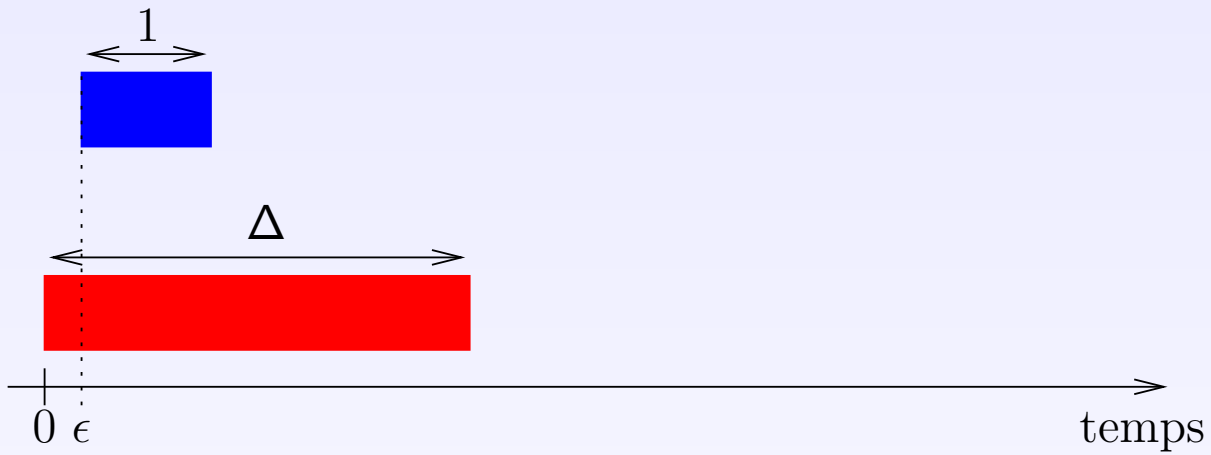
Compétitivité de FIFO (2) : au *mieux* Δ compétitive



FIFO

Max-stretch $= 1 + \Delta - \epsilon$

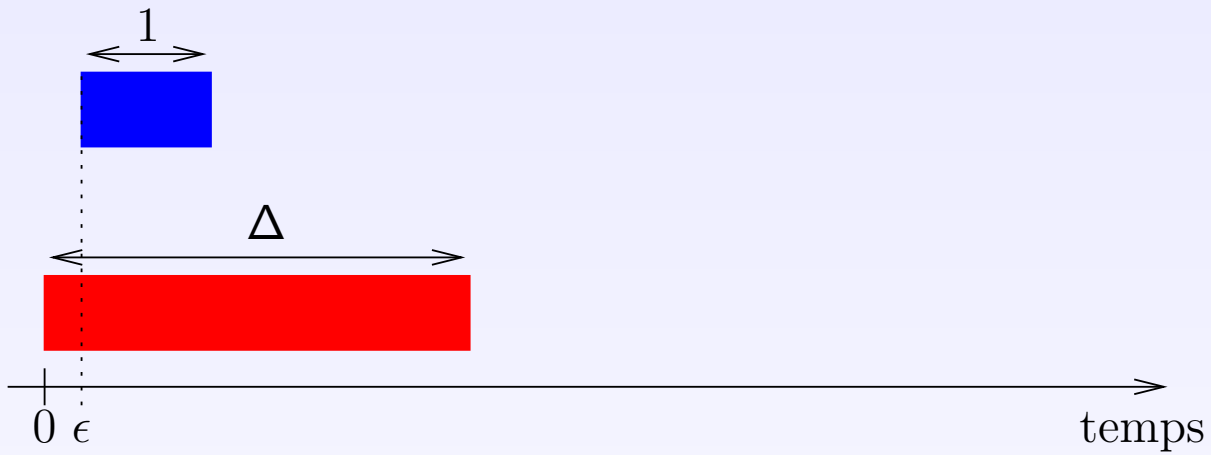
Compétitivité de FIFO (2) : au *mieux* Δ compétitive



FIFO Max-stretch $= 1 + \Delta - \epsilon$

Optimal Max-stretch $= \frac{1+\Delta}{\Delta}$

Compétitivité de FIFO (2) : au *mieux* Δ compétitive



FIFO Max-stretch $= 1 + \Delta - \epsilon$

Optimal Max-stretch $= \frac{1+\Delta}{\Delta}$

Ratio de compétitivité : $\frac{1+\Delta-\epsilon}{\frac{1+\Delta}{\Delta}} = \Delta \frac{1+\Delta-\epsilon}{1+\Delta} = \Delta - \epsilon \frac{\Delta}{1+\Delta} \geq \Delta - \epsilon.$

Compétitivité de FIFO (3) : au *pire* Δ compétitive

Borne sur la compétitivité

Théorème

Sur un seul processeur, aucun algorithme d'ordonnancement on-line avec préemption, minimisant le stretch, n'a un facteur de compétitivité inférieur ou égal à $\frac{1}{2}\Delta^{\sqrt{2}-1}$ *si le système reçoit des tâches d'au moins trois tailles différentes et si* Δ *est le rapport entre la taille de la plus grande et de la plus petit tâche.*

Borne sur la compétitivité

Théorème

Sur un seul processeur, aucun algorithme d'ordonnancement on-line avec préemption, minimisant le stretch, n'a un facteur de compétitivité inférieur ou égal à $\frac{1}{2}\Delta^{\sqrt{2}-1}$ *si le système reçoit des tâches d'au moins trois tailles différentes et si* Δ *est le rapport entre la taille de la plus grande et de la plus petit tâche.*

Principe de la preuve : par l'absurde on suppose qu'il existe un tel algorithme et on construit une séquence de tâches qui l'envoie dans le décor.

L'adversaire

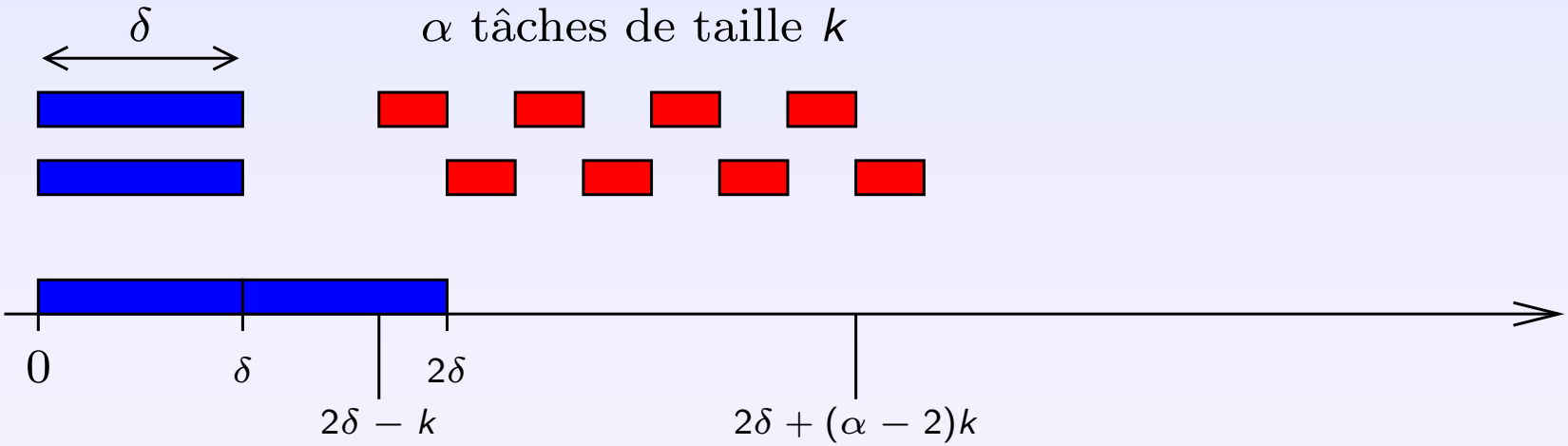
L'adversaire

L'adversaire

Stretch réalisable : $\dfrac{2\delta - 0}{\delta} = 2.$

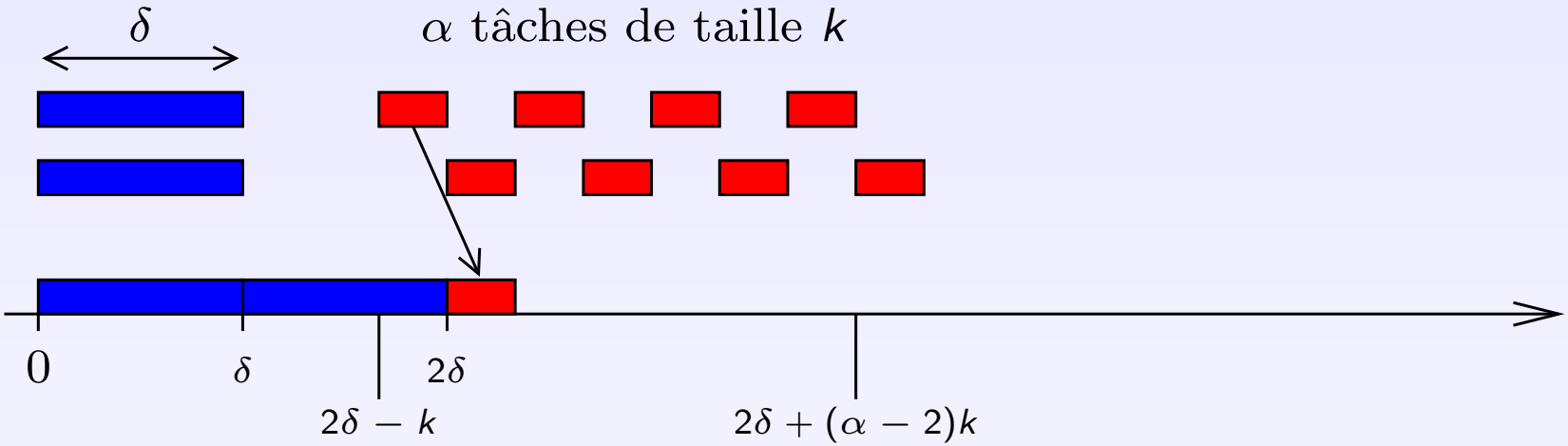
L'adversaire

L'adversaire



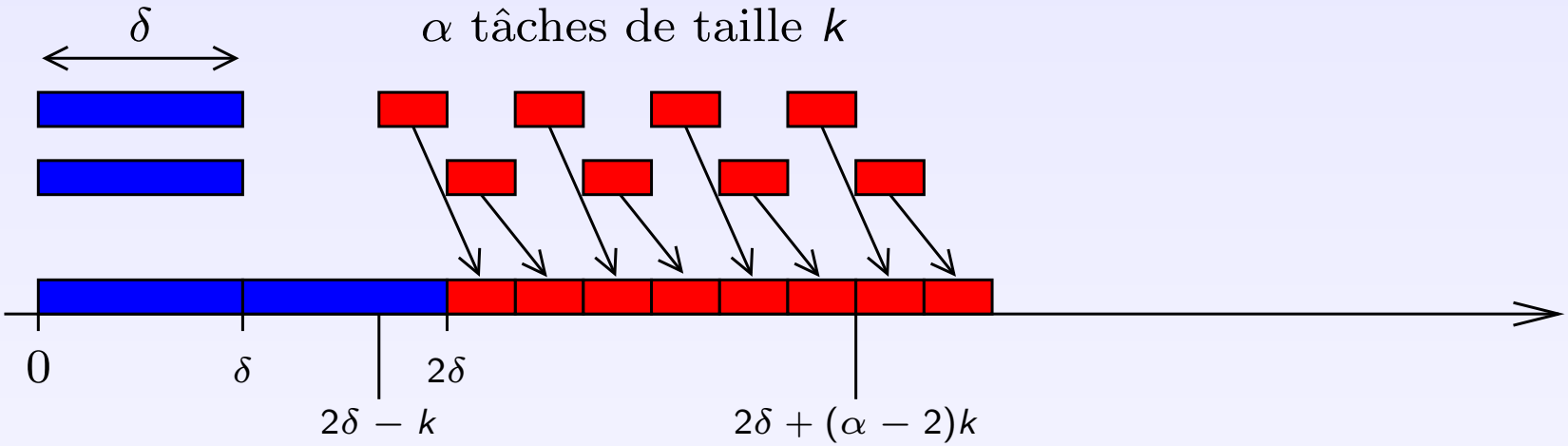
La tâche T_{2+j} arrive à la date $2\delta + (j-2)k$.

L'adversaire



La tâche T_{2+j} arrive à la date $2\delta + (j-2)k$.

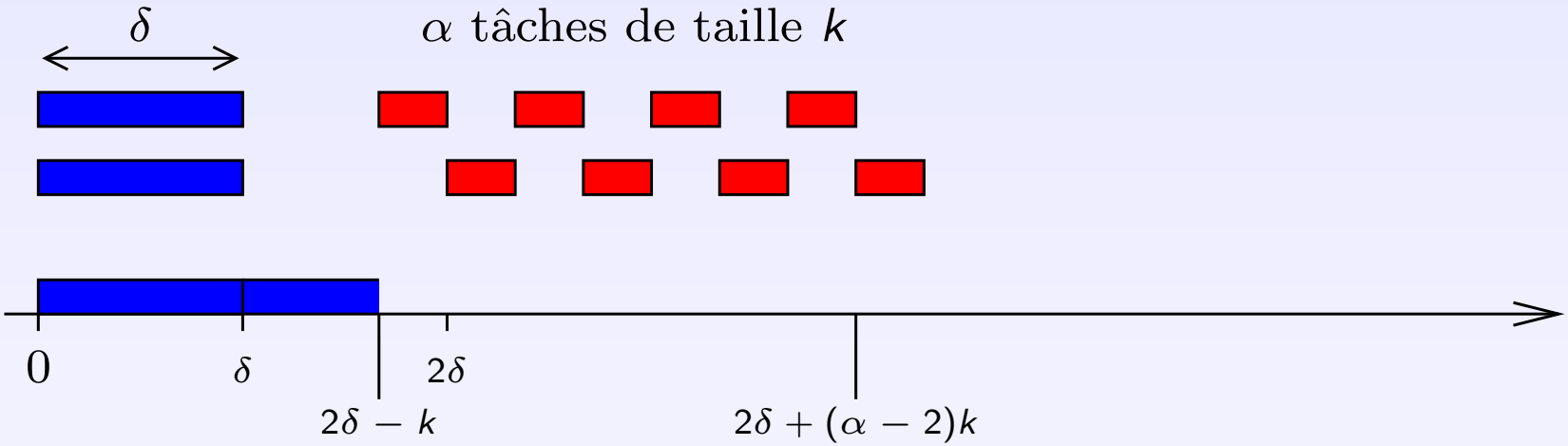
L'adversaire



La tâche T_{2+j} arrive à la date $2\delta + (j-2)k$.

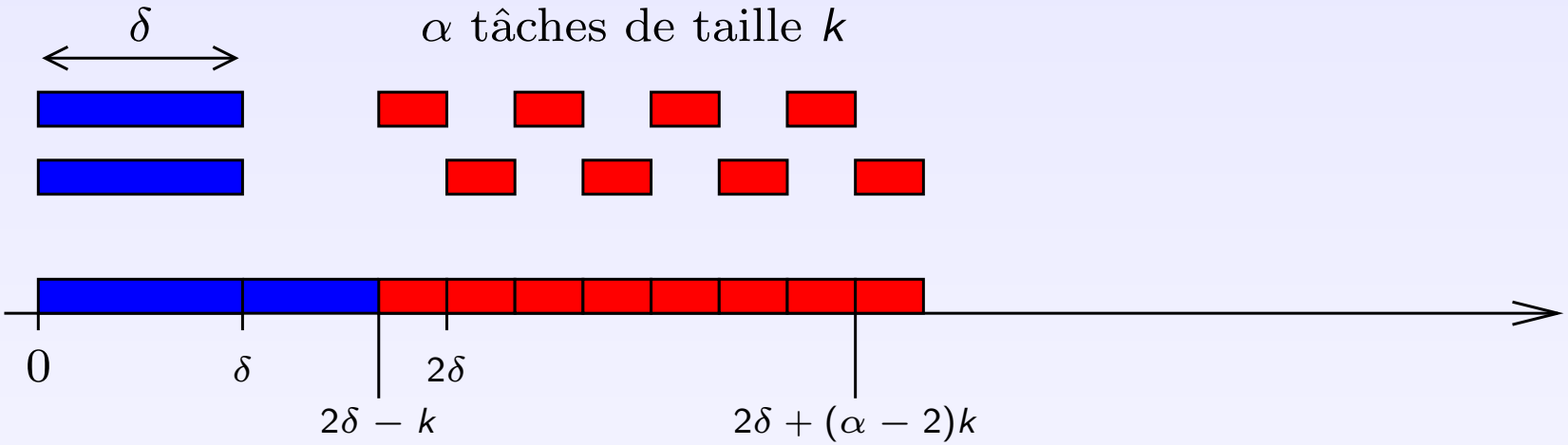
Stretch réalisable : $\dfrac{(2\delta + jk) - (2\delta + (j-2)k)}{k} = 2.$

L'adversaire



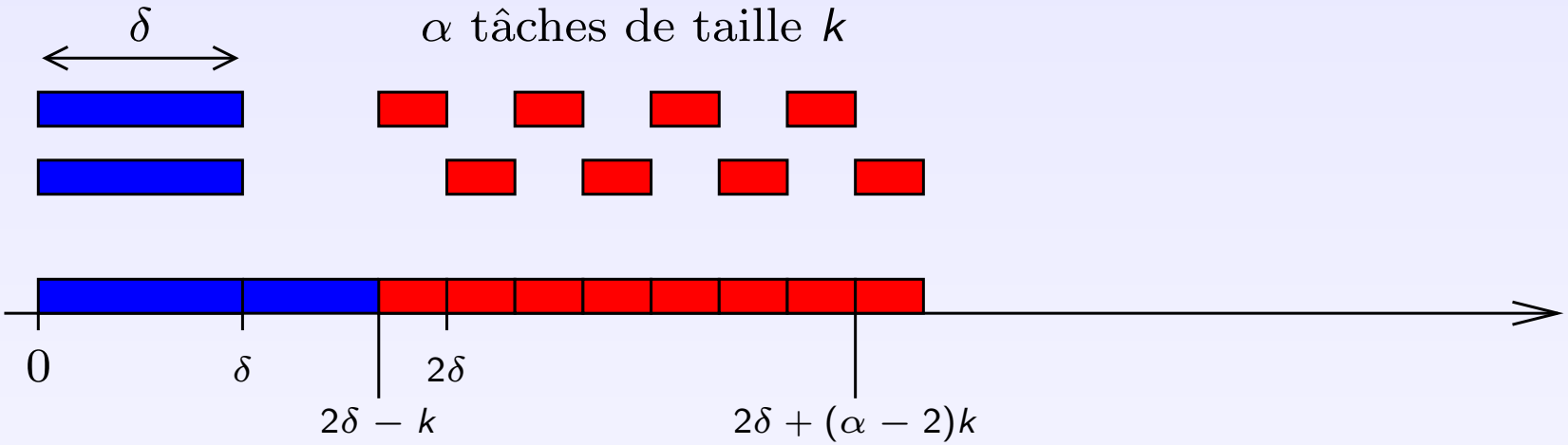
En pratique : on ne sait pas ce qui se passe après $2\delta - k$.

L'adversaire



On veut empêcher ce cas de figure (chaque tâche de taille k est exécutée dès son arrivée).

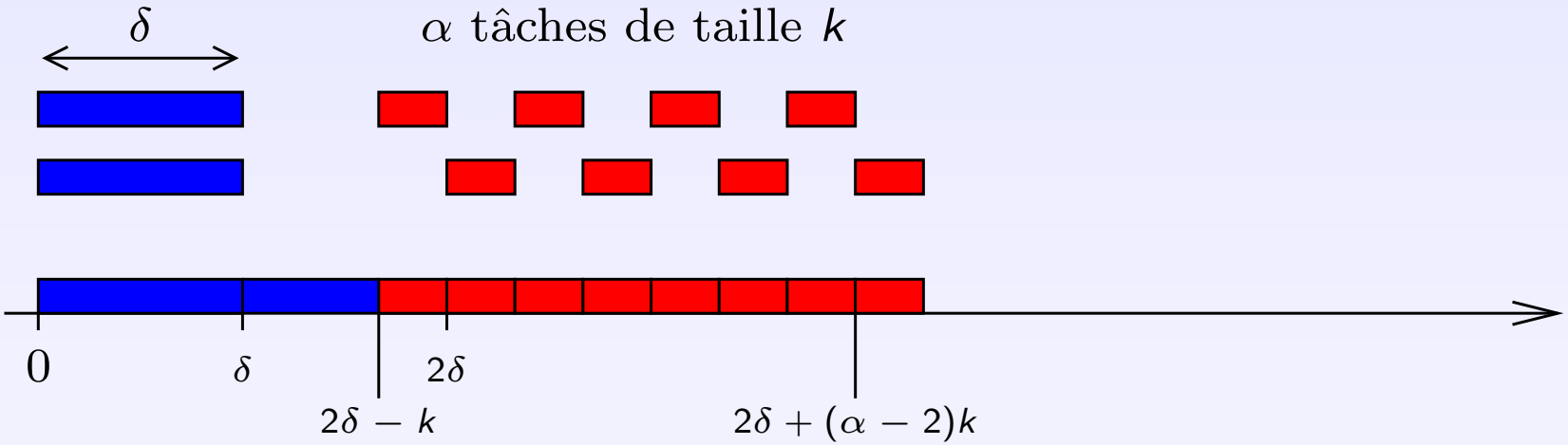
L'adversaire



On veut empêcher ce cas de figure (chaque tâche de taille k est exécutée dès son arrivée).

L'algorithme étant $\frac{1}{2}\Delta^{\sqrt{2}-1}$ compétitif, l'exécution de T_1 et de T_2 doit finir au plus tard à la date : $2 \cdot \frac{1}{2}\Delta^{\sqrt{2}-1} \cdot \delta = 2 \cdot \frac{1}{2}\left(\frac{\delta}{k}\right)^{\sqrt{2}-1} \cdot \delta$

L'adversaire

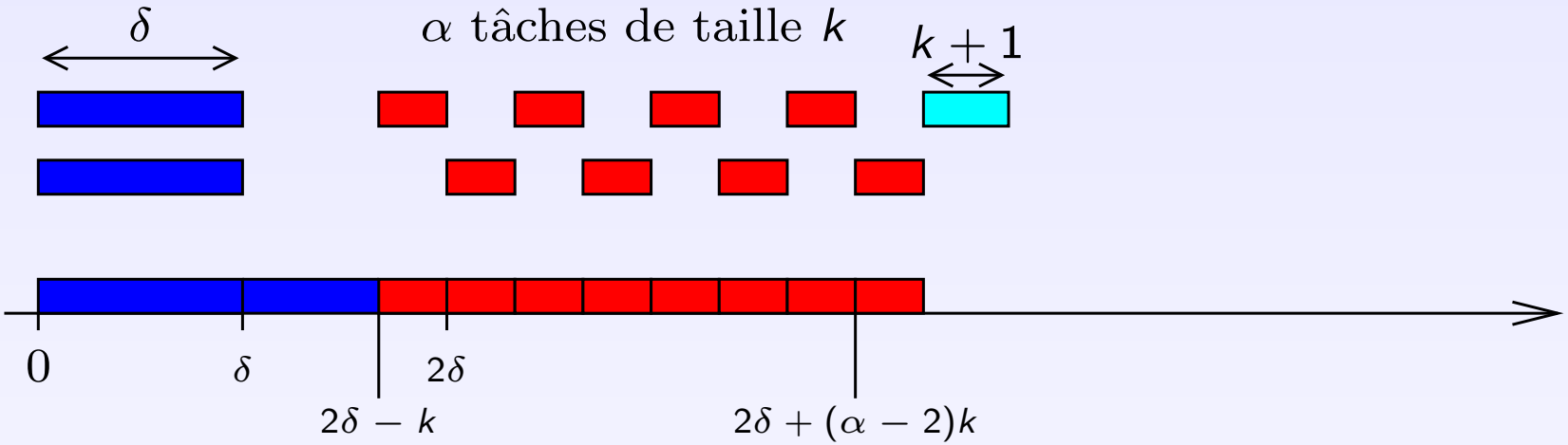


On veut empêcher ce cas de figure (chaque tâche de taille k est exécutée dès son arrivée).

L'algorithme étant $\frac{1}{2}\Delta^{\sqrt{2}-1}$ compétitif, l'exécution de T_1 et de T_2 doit finir au plus tard à la date : $2 \cdot \frac{1}{2}\Delta^{\sqrt{2}-1} \cdot \delta = 2 \cdot \frac{1}{2}\left(\frac{\delta}{k}\right)^{\sqrt{2}-1} \cdot \delta$

On pose $\alpha = \lceil 1 + k - \frac{2\delta}{k} \rceil$ et alors $2\delta + (\alpha - 1)k \geq 2 \cdot \frac{1}{2}\left(\frac{\delta}{k}\right)^{\sqrt{2}-1} \cdot \delta$.

L'adversaire

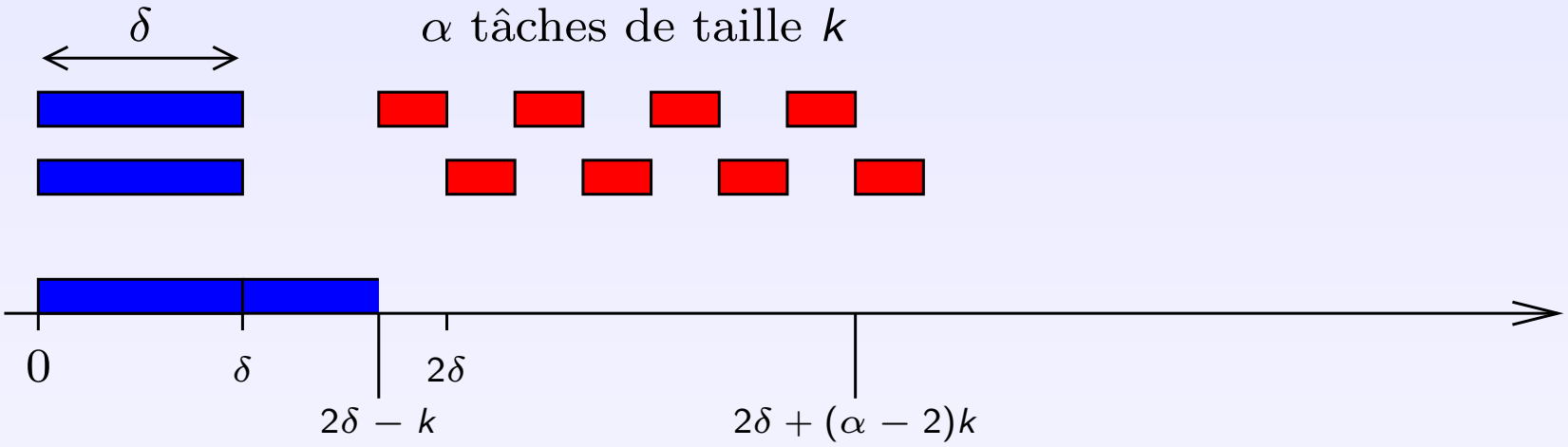


On veut empêcher ce cas de figure (chaque tâche de taille k est exécutée dès son arrivée).

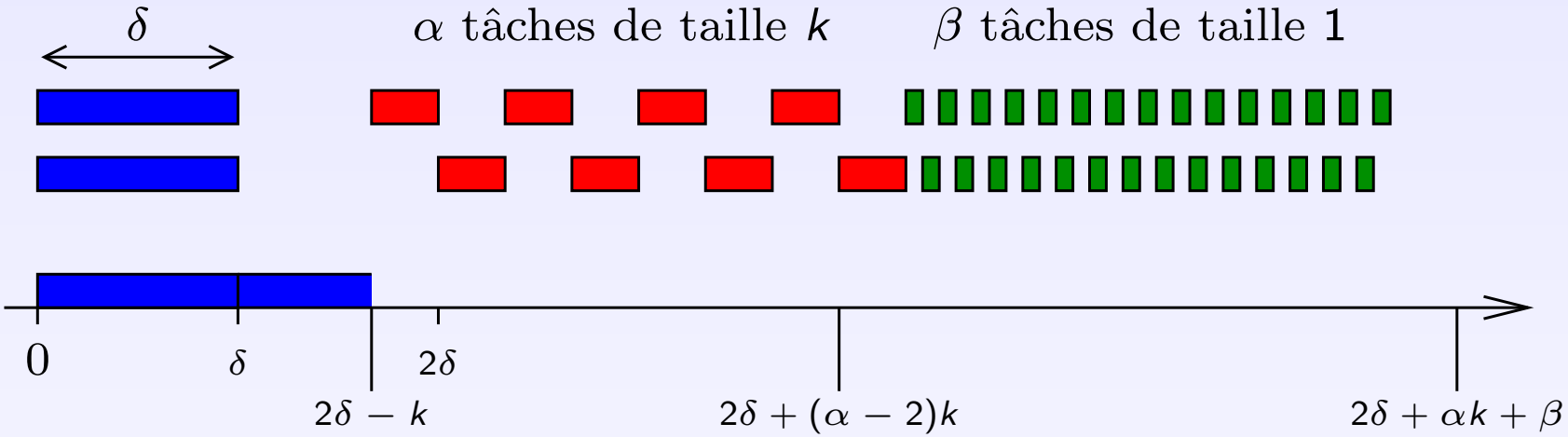
L'algorithme étant $\frac{1}{2}\Delta^{\sqrt{2}-1}$ compétitif, l'exécution de T_1 et de T_2 doit finir au plus tard à la date : $2 \cdot \frac{1}{2}\Delta^{\sqrt{2}-1} \cdot \delta = 2 \cdot \frac{1}{2}\left(\frac{\delta}{k}\right)^{\sqrt{2}-1} \cdot \delta$

On pose $\alpha = \lceil 1 + k - \frac{2\delta}{k} \rceil$ et alors $2\delta + (\alpha - 1)k \geq 2 \cdot \frac{1}{2}\left(\frac{\delta}{k}\right)^{\sqrt{2}-1} \cdot \delta$.

L'adversaire

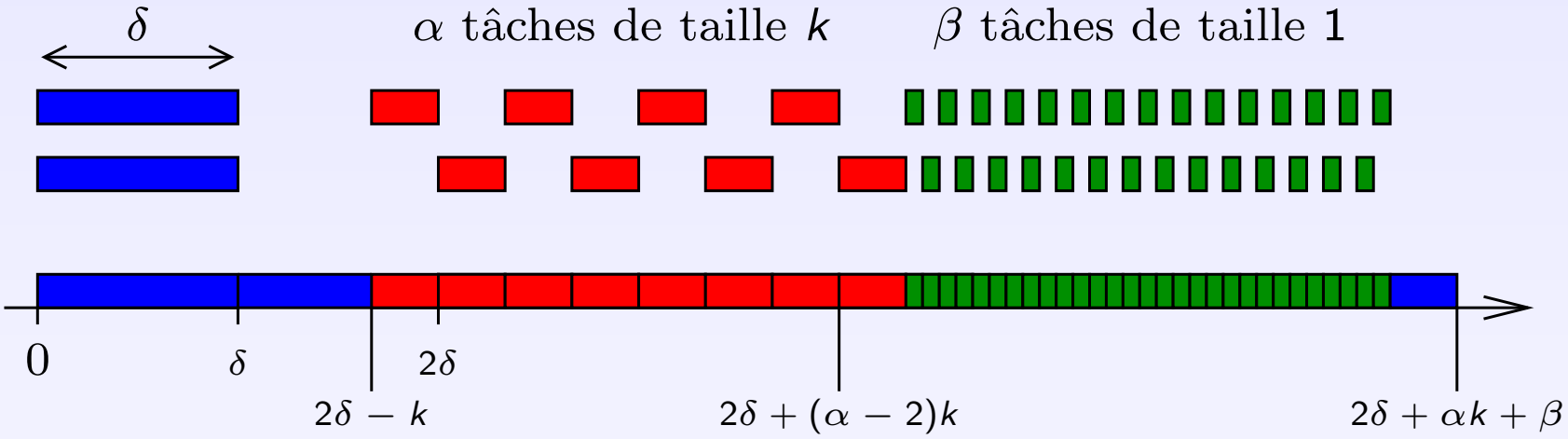


L'adversaire



La tâche $T_{2+\alpha+j}$ arrive à la date $2\delta + (\alpha - 1)k + (j - 1)$.

L'adversaire



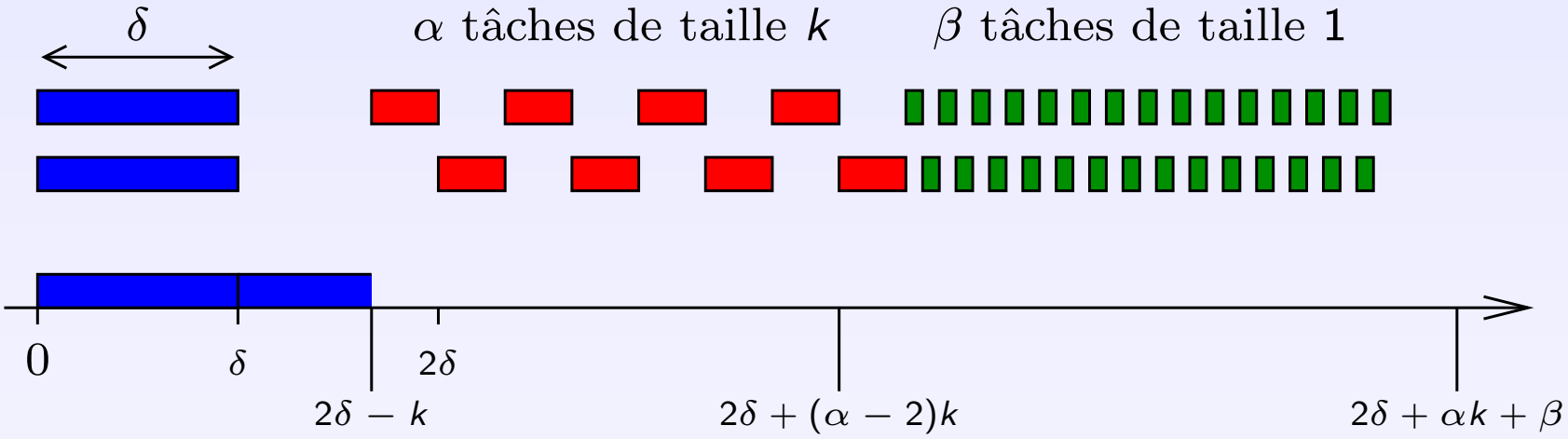
Stretch réalisable (en offline)

Stretch de chaque tâche de taille k ou 1 : 1.

Stretch de T_1 ou T_2 : $\dfrac{2\delta + \alpha k + \beta}{\delta}$

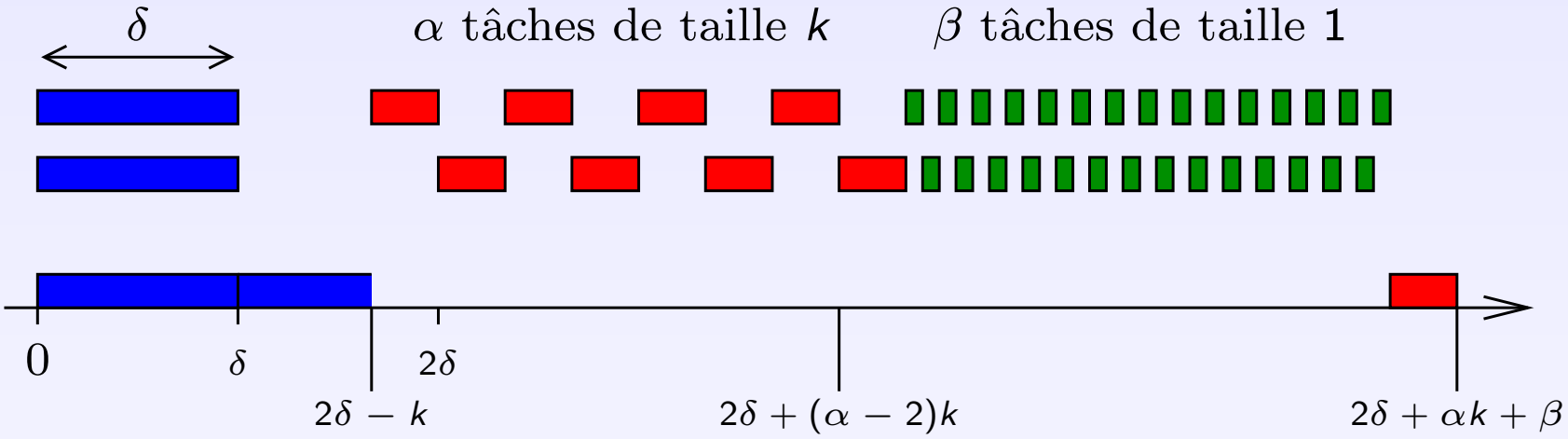
Stretch optimal $\leq \dfrac{2\delta + \alpha k + \beta}{\delta}$

L'adversaire



Stretch atteignable en online

L'adversaire

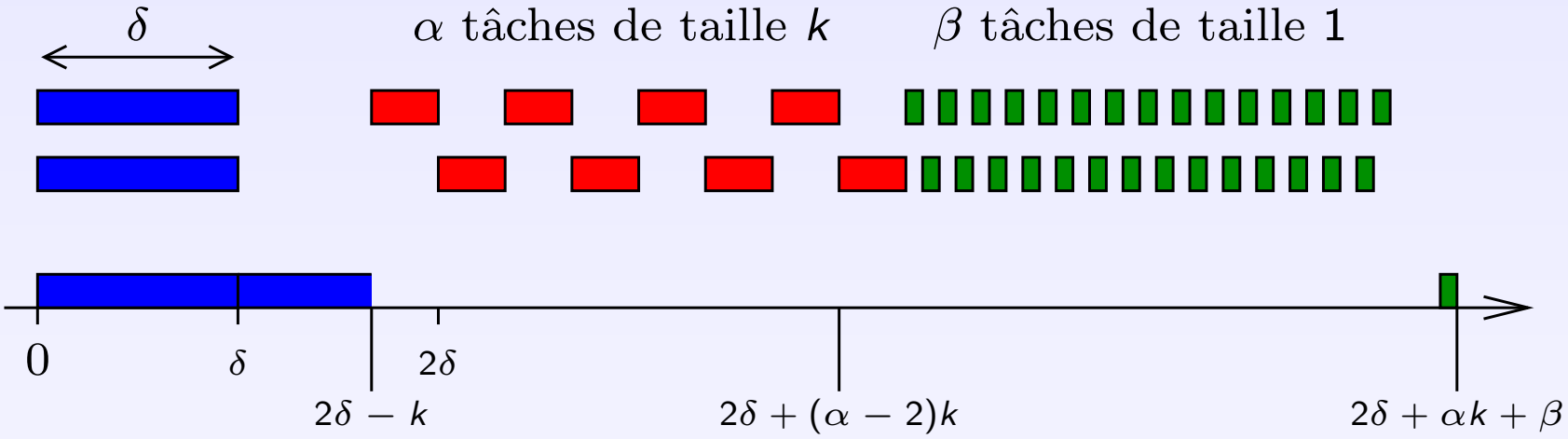


Stretch atteignable en online

On termine par une tâche de taille k.

$$\text{Stretch} \geq \frac{(2\delta + \alpha k + \beta) - (2\delta + (\alpha - 2)k)}{k} = 2 + \frac{\beta}{k}.$$

L'adversaire

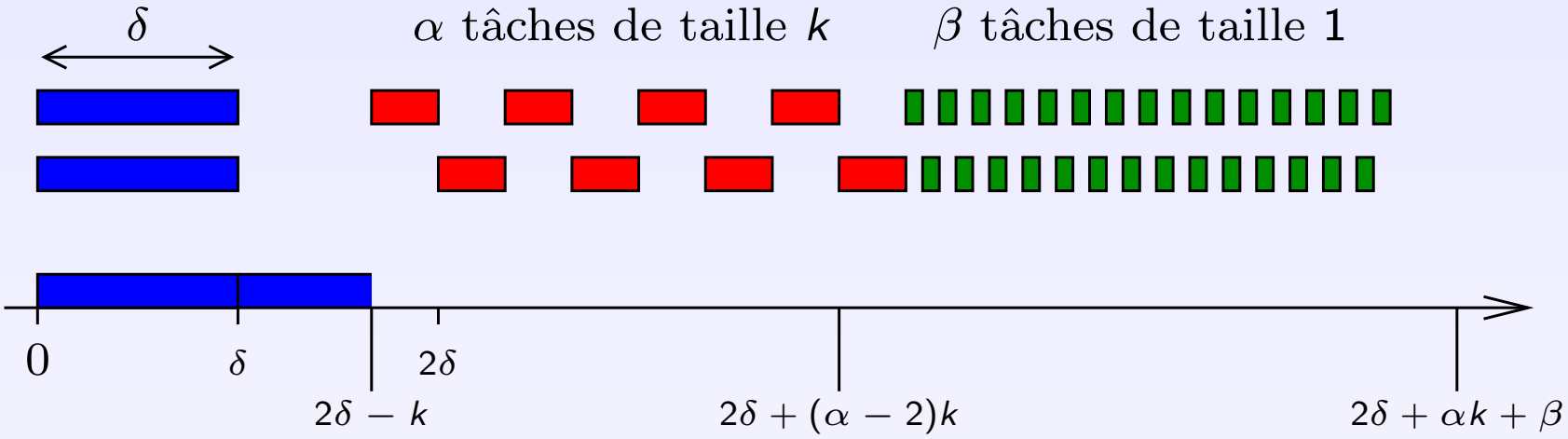


Stretch atteignable en online

On termine par une tâche de taille 1.

$$\text{Stretch} \geq \frac{(2\delta + \alpha k + \beta) - (2\delta + (\alpha - 1)k + (\beta - 1))}{1} = k + 1.$$

L'adversaire



Stretch atteignable en online

$$\text{Stretch} \geq \min\left\{2 + \frac{\beta}{k}, k + 1\right\}$$

On pose : $\beta = \lceil k(k-1) \rceil$

Alors : stretch $\geq k + 1$.

L'adversaire : un résumé

$$\alpha = \left\lceil 1 + k - \frac{2\delta}{k} \right\rceil$$

$$\beta = \lceil k(k-1) \rceil$$

$$\text{Stretch optimal} \leq \frac{2\delta + \alpha k + \beta}{\delta}$$

Stretch réalisé $\geq k + 1$.

L'adversaire : un résumé

$$\alpha = \left\lceil 1 + k - \frac{2\delta}{k} \right\rceil$$

$$\beta = \lceil k(k-1) \rceil$$

$$\text{Stretch optimal} \leq \frac{2\delta + \alpha k + \beta}{\delta}$$

Stretch réalisé $\geq k + 1$.

On pose $k = \delta^{2-\sqrt{2}}$

L'adversaire : un résumé

$$\alpha = \left\lceil 1 + k - \frac{2\delta}{k} \right\rceil$$

$$\beta = \lceil k(k-1) \rceil$$

Stretch optimal $\leq \dfrac{2\delta + \alpha k + \beta}{\delta}$

Stretch réalisé $\geq k + 1$.

On pose $k = \delta^{2-\sqrt{2}}$

On a alors $k + 1 > \left(\dfrac{1}{2}\delta^{\sqrt{2}-1}\right)\left(\dfrac{2\delta + \alpha k + \beta}{\delta}\right)$

Algorithmes d'approximations existants

Deux algorithmes gloutons d'approximations $\sqrt{\Delta}$-compétitif :

Algorithmes d'approximations existants

Deux algorithmes gloutons d'approximations $\sqrt{\Delta}$-compétitif :

1. Bender, Muthukrishnan, et Rajaraman (2002)
   Pour chaque tâche J_j, on définit un pseudo-stretch $\widehat{S}_j(t)$:

$$\widehat{S}_j(t) = \begin{cases} \frac{t-r_j}{\sqrt{\Delta}} & \text{si } 1 \leq p_j \leq \sqrt{\Delta}, \\ \frac{t-r_j}{\Delta} & \text{si } \sqrt{\Delta} < p_j \leq \Delta. \end{cases}$$

   Les tâches sont ordonnancées par pseudo-stretch décroissants.

Algorithmes d'approximations existants

Deux algorithmes gloutons d'approximations $\sqrt{\Delta}$-compétitif :

1. Bender, Muthukrishnan, et Rajaraman (2002)
   Pour chaque tâche J_j, on définit un pseudo-stretch $\widehat{\mathcal{S}}_j(t)$:

$$\widehat{\mathcal{S}}_j(t) = \begin{cases} \frac{t-r_j}{\sqrt{\Delta}} & \text{si } 1 \leq p_j \leq \sqrt{\Delta}, \\ \frac{t-r_j}{\Delta} & \text{si } \sqrt{\Delta} < p_j \leq \Delta. \end{cases}$$

   Les tâches sont ordonnancées par pseudo-stretch décroissants.

2. Bender, Chahrabarti, et Muthukrishnan (1998).
   À chaque fois qu'une nouvelle tâche arrive :
   - Calcul du max-stretch offline $\mathcal{S}$.
   - Les tâches sont ordonnancées dans l'ordre *earliest deadline first* avec les deadlines définit par $\sqrt{\Delta} \times \mathcal{S}$.

   Problème : ne cherche à optimiser que les tâches les plus contraignantes.

Une heuristique non-garantie

À chaque fois qu'une nouvelle tâche arrive :

1. Préempte la tâche en cours d'exécution (s'il y en a).
2. Calcul du meilleur max-stretch atteignable, $\mathcal{S}$, étant données les décisions déjà prises.
3. Avec les dates butoirs et les intervalles définis par le max-stretch $\mathcal{S}$, résoudre :

$$\text{MINIMIZE} \quad \sum_{j=1}^{n} \sum_{t} \left(\sum_{i=1}^{m} \alpha_{i,j}^{(t)} \right) \frac{\sup I_t(\mathcal{S}) + \inf I_t(\mathcal{S})}{2} \quad \text{, WHILE}$$

$$\begin{cases} \text{(1a)} \quad \forall i, \forall j, \forall t, \quad r_j \geq \sup I_t(\mathcal{S}) \Rightarrow \alpha_{i,j}^{(t)} = 0 \\ \text{(1b)} \quad \forall i, \forall j, \forall t, \quad d_j(\mathcal{S}) \leq \inf I_t(\mathcal{S}) \Rightarrow \alpha_{i,j}^{(t)} = 0 \\ \text{(1c)} \quad \forall t, \forall i, \quad \sum_{j} \alpha_{i,j}^{(t)} . c_{i,j} \leq \sup I_t(\mathcal{S}) - \inf I_t(\mathcal{S}) \\ \text{(1d)} \quad \forall j, \quad \sum_{t} \sum_{i} \alpha_{i,j}^{(t)} = 1 \end{cases} \tag{1}$$

(Pseudo-approximation d'une relaxation rationnelle du stretch

Conclusion

Minimisation du stretch moyen

- Cas offline : apparemment difficile.
- Cas online : plutôt facile.

Minimisation du stretch maximal

- Cas offline : en temps polynomial.
- Cas online : très difficile.

et en pratique ?

Plan de l'exposé

Paramètres de simulation

- **plates-formes** contenant 3, 10, ou 20 clusters homogènes de 10 processeurs ;
- **applications** avec 3, 10, ou 20 bases de données distinctes ;
- **disponibilité des bases de données** de 30%, 60%, ou 90% chacune ;
- **charge de travail** de 0.75, 1.0, 1.25, 1.5, 2.0, ou 3.0.

Résultats des simulations

	Max-stretch			Sum-stretch		
	Mean	SD	Max	Mean	SD	Max
Offline	1.0000	0.0003	1.0167	1.6729	0.3825	4.4468
Online	1.0025	0.0127	2.0388	1.0806	0.0724	2.0343
Online-EDF	1.0024	0.0127	2.0581	1.0775	0.0708	2.0392
Online-EGDF	1.0781	0.1174	2.4053	1.0021	0.0040	1.0707
Bender98 [1]	1.0798	0.1315	2.0978	1.0024	0.0044	1.0530
SWRPT	1.0845	0.1235	2.5307	1.0002	0.0012	1.0458
SRPT	1.0939	0.1299	2.3741	1.0044	0.0055	1.0907
SPT	1.1147	0.1603	2.8295	1.0027	0.0054	1.1195
Bender02	3.4603	3.0260	28.4016	1.2053	0.2417	5.2022
FIFO-Div	6.3385	7.4375	73.4019	1.3732	0.5628	11.0440
MCT	27.0124	20.1083	129.6119	50.9840	36.9797	157.8909

Tab.: Statistiques aggrégées sur 162 configurations plate-forme/application.

Plan de l'exposé

Conclusions

Minimisation du stretch moyen

- Cas offline : apparemment difficile.
- Cas online : plutôt facile.

Minimisation du stretch maximal

- Cas offline : en temps polynomial.
- Cas online : très difficile.

En pratique

- Les algorithmes d'approximations sont dans les choux.
- SWRPT et nos onlines très performantes.
- SWRPT peut entraîner de la famine.
- Sum-stretch ne semble pas être une mesure pertinente.

Et plus globalement...

1. Étude du problème (théorique) offline car la solution online ne sera jamais meilleure que l'optimal offline.

2. Comparaison des solutions online et offline pour quantifier la qualité des solutions online.

3. Transposition des résultats d'un modèle à un autre (du modèle des tâches divisibles au modèle avec préemption, et réciproquement).