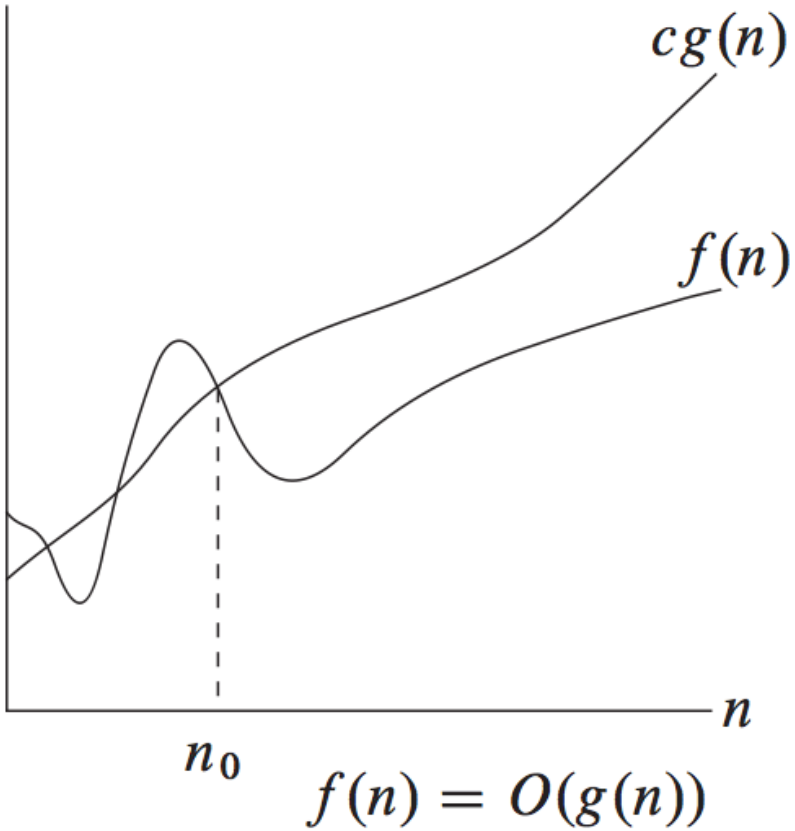
Algorithmes gloutons

Johanne Cohen

LISN-CNRS

Plan

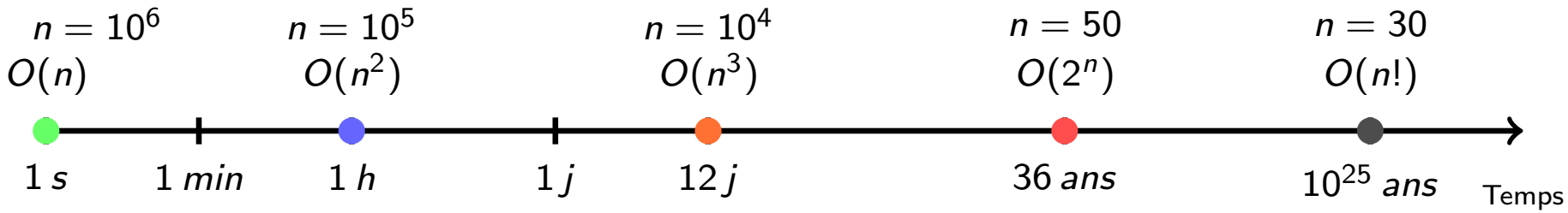
Notation $\mathcal{O}$



(image scannée dans le livre *Introduction to Algorithms* de Thomas H. Cormen Charles E. Leiserson Ronald L. Rivest Clifford Stein)

Ordres de grandeur en algorithmique

Hypothèse: sur un processeur effectuant 10^6 d'instructions de haut niveau par seconde ($\infty = 10^{25}$ans).



n	$n \log_2 n$	n^2	n^3	1.5^n	2^n	$n!$
10	< 1 s	< 1 s	< 1 s	< 1 s	< 1 s	4 s
30	< 1 s	< 1 s	< 1 s	< 1 s	18 min	10^{25} ans
50	< 1 s	< 1 s	< 1 s	11 min	36 ans	∞
100	< 1 s	< 1 s	1 s	12 892 ans	10^{17} ans	∞
10^3	< 1 s	1 s	18 min	∞	∞	∞
10^4	< 1 s	2 min	12 jours	∞	∞	∞
10^5	2 s	3 h	32 ans	∞	∞	∞
10^8	20 s	12 jours	31 710 ans	∞	∞	∞

Un premier exemple : problème du stockage.

Considérons n fichiers $P_1, P_2, \ldots, P_n$ qui peuvent être stocker sur un disque dur de capacité D gigabytes.

- Chaque fichier P_i a besoin s_i gigabytes pour être stocké.
- Tous les fichiers ne peuvent pas être stockés sur le disque :

$$(\sum_{i=1}^{n} s_i > D)$$

Objectif:

Concevoir un algorithme qui permet de maximiser le nombre de fichiers stockés sur le disque.

Plan

Problème de stockage : approche générale

1. Modéliser le problème

   à l'aide d'outils mathématiques, en le formulant comme un problème d'optimisation.

2. Concevoir un algorithme capable de fournir une solution optimale.

Problème d'optimisation

Données :

$\mathcal{X}$ un ensemble fini,
v une valuation des éléments de $\mathcal{X}$: $\mathcal{X} \to \mathbb{R}^+$,
$\mathcal{F}$ un ensemble de parties de $\mathcal{X}$ (ensemble des solutions faisables)

Objectif: Trouver $S \in \mathcal{F}$ qui maximise la quantité $\sum_{e \in S} v(e)$

Remarque: Si $\mathcal{F}$ est l'ensemble des parties de $\mathcal{X}$, alors l'algorithme "*force brute*" nécessite au moins $\mathcal{O}(2^{|\mathcal{X}|})$ opérations.

Un algorithme résolvant le problème du stockage.

Rappel: Trouver un sous-ensemble de $\{P_1, \ldots, P_n\}$ de cardinal maximum qui peut être stockés sur le disque de capacité D.

1. Classer les fichiers $P_1, P_2, \ldots, P_n$ en fonction de la taille du fichier s_i:

$$s_1 \leq s_2 \leq \cdots \leq s_n$$

2. Initialiser la recherche avec $S = \emptyset$
3. Pour $i = 1$ à n faire:
   Si $\sum_{P_j \in S} s_j + s_i \leq D$ alors $S \leftarrow S \cup \{P_i\}$
4. Retourner S

Exemple: Considérons qu'on dispose 4 fichiers de tailles 3, 5, 7, 10, et un disque dur de capacité $D = 17$.

L'algorithme retourne $S = \{P_1, P_2, P_3\}$.

Un algorithme résolvant le problème du stockage.

Rappel: Trouver un sous-ensemble de $\{P_1, \ldots, P_n\}$ de cardinal maximum qui peut être stockés sur le disque de capacité D.

1. Classer les fichiers $P_1, P_2, \ldots, P_n$ en fonction de la taille du fichier s_i: $\mathcal{O}(n \log n)$ opérations
   $$s_1 \leq s_2 \leq \cdots \leq s_n$$
2. Initialiser la recherche avec $S = \emptyset$ $\mathcal{O}(1)$ *opérations*
3. Pour $i = 1$ à n faire: *la boucle est exécutée n fois*
   Si $\underbrace{\sum_{P_j \in S} s_j + s_i \leq D}_{\mathcal{O}(1) \text{ opérations}}$ alors $\underbrace{S \leftarrow S \cup \{P_i\}}_{\mathcal{O}(1) \text{ opérations}}$
4. Retourner S

Complexité (au pire): $\mathcal{O}(n \log n)$

Théorème

Théorème

L'algorithme retourne une solution optimale.

Preuve :

- Il suffit de prouver par récurrence qu'une solution optimale S contient les $|S|$ premiers fichiers ayant les plus petites tailles.
- Soit S une solution optimale
  - si S contient le fichier P_1, alors c'est bon.

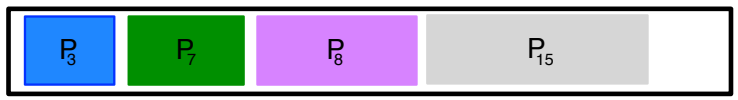
Théorème

Théorème

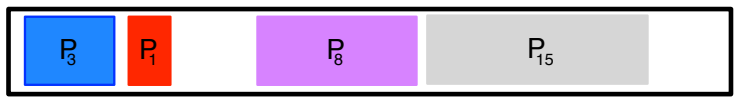
L'algorithme retourne une solution optimale.

Preuve :

- Il suffit de prouver par récurrence qu'une solution optimale S contient les $|S|$ premiers fichiers ayant les plus petites tailles.
- Soit S une solution optimale
  - si S contient le fichier P_1, alors c'est bon.
  - si S ne contient pas le fichier P_1, alors



en remplacant un élément de S par P_1,



la solution obtenue reste optimale.

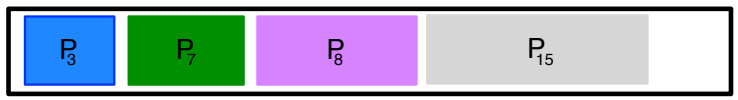
Théorème

Théorème

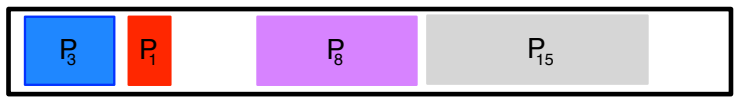
L'algorithme retourne une solution optimale.

Preuve :

- Il suffit de prouver par récurrence qu'une solution optimale S contient les $|S|$ premiers fichiers ayant les plus petites tailles.
- Soit S une solution optimale
  - si S contient le fichier P_1, alors c'est bon.
  - si S ne contient pas le fichier P_1, alors



en remplacant un élément de S par P_1,



la solution obtenue reste optimale.

- et ainsi de suite en considérant les fichiers $P_2, P_3 \ldots, P_{|S|}$.

Plan

Un algorithme glouton : principe général

- Pour un problème d'optimisation,

  on construit la solution de façon séquentielle, en faisant à chaque étape le meilleur choix local.

- Pas de retour en arrière

  On ne remet en cause les décisions.

- Progression descendante = choix puis résolution d'un problème plus petit.

Un algorithme glouton générique

1. Classer les éléments de $\mathcal{X}$ par valuations décroissantes:
$$v(e_1) \geq v(e_2) \geq \cdots \geq v(e_n)$$

2. Initialiser la recherche avec $S = \emptyset$;
3. Pour $i = 1$ à n faire:
   Si $S \cup \{e_i\} \in \mathcal{F}$ alors $S \leftarrow S \cup \{e_i\}$;

4. Retourner S

Un algorithme glouton générique

1. Classer les éléments de $\mathcal{X}$ par valuations décroissantes:
$$v(e_1) \geq v(e_2) \geq \cdots \geq v(e_n)$$
$\mathcal{O}(n \log n)$ opérations
2. Initialiser la recherche avec $S = \emptyset$; $\mathcal{O}(1)$ *opérations*
3. Pour $i = 1$ à n faire: *la boucle est exécutée n fois*
   Si $\underbrace{S \cup \{e_i\} \in \mathcal{F}}_{\mathcal{O}(T) \text{ opérations}}$ alors $\underbrace{S \leftarrow S \cup \{e_i\}}_{\mathcal{O}(1) \text{ opérations}}$;
4. Retourner S

Complexité: $\mathcal{O}(n \log n + nT)$

où T correspond au coût du test $(S \cup \{e_i\} \in \mathcal{F})$

Plan

Gestion d'un cache

- Un système de mémoire composé de deux types de mémoire :

  la **mémoire principale** et la **mémoire cache**.

- Pour accéder aux données de la mémoire principale, le système doit les placer dans le cache.
- Lorsque la page demandée est déjà dans le cache, elle est directement lue.
- Sinon, la page demandée est extraite de la mémoire principale, puis mise dans le cache

  ce qui ralentira la vitesse d'extraction des données.

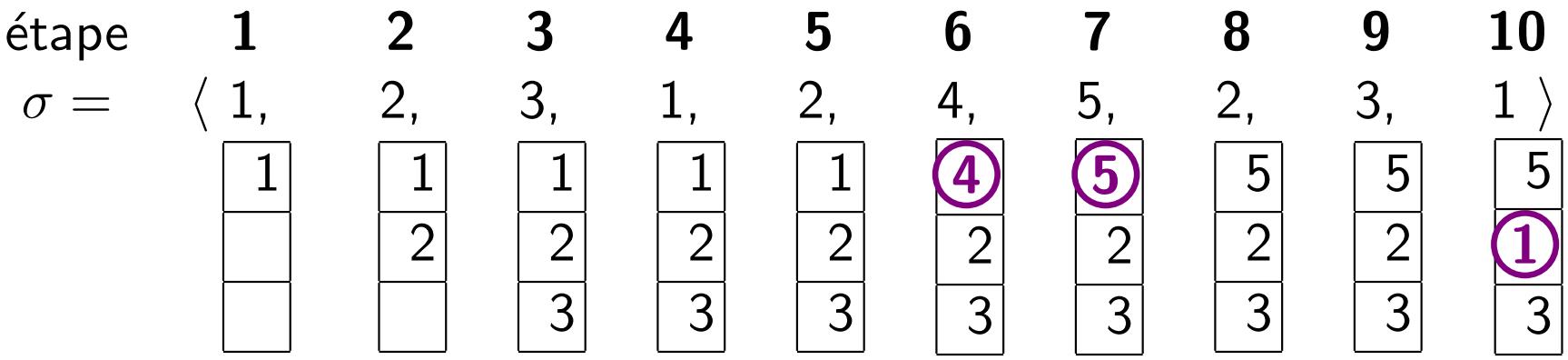
Notation:

- k le nombre de pages que peut stocker la mémoire cache;
- N le nombre de pages contenues dans la mémoire principale.

Objectif

Exemple:

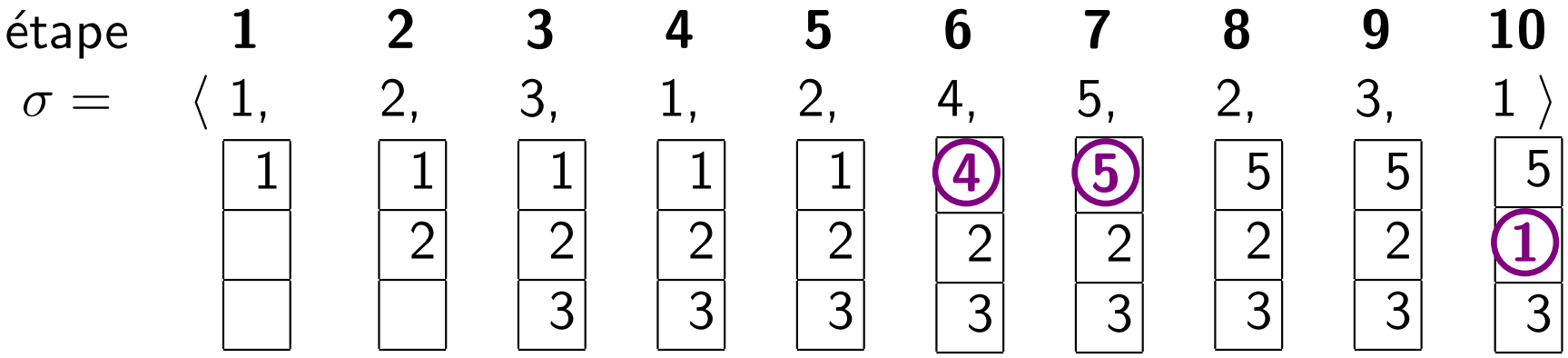
- Le cache peut contenir au plus $k = 3$ pages.
- Considérons la séquence de demandes: $\langle 1, 2, 3, 1, 2, 4, 5, 2, 3, 1 \rangle$



Objectif

Exemple:

- Le cache peut contenir au plus $k = 3$ pages.
- Considérons la séquence de demandes: $\langle 1, 2, 3, 1, 2, 4, 5, 2, 3, 1 \rangle$



Objectif: Concevoir un algorithme de chargement de pages qui doit traitées dans l'ordre des demandes de pages

- **Contrainte de présence:** la page doit être dans le cache.
- **Contrainte de capacité:** pas au plus k pages dans le cache.

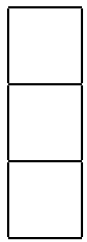
Plus précisément

Autre exemple : chargement de pages

Algorithme **générique** de chargement de pages

Un algorithme **générique** de chargement de pages

En entrée: une séquence de demandes σ.
Algorithme: A chaque étape i, faire

1. décider si $\sigma[i]$ est dans le cache ($\sigma[i]$ dans  ?)
2. si oui, alors aller à l'étape suivante
3. sinon:
   - 3.1 si le cache n'est pas plein alors insérer $\sigma[i]$ dans le cache
   - 3.2 sinon
     - 3.2.1 déterminer la page e à supprimer et son emplacement dans le cache
     - 3.2.2 insérer la page $\sigma[i]$ dans le cache à la place de e

Complexité de l'algorithme

L'instruction décider si $\sigma[i]$ est dans le cache peut se faire en

- $\mathcal{O}(1)$ opérations :
  - il faut un tableau B de booléens stockant si la page i est, ou pas, dans le cache
  - S'il existe une telle structure, il faut $\mathcal{O}(N)$ opérations pour l'initialiser.

- $\mathcal{O}(\log k)$ opérations :
  - il faut parcourir un arbre binaire de recherche équilibré qui maintient quelles sont les pages dans le cache.

Complexité de l'algorithme

L'instruction décider si $\sigma[i]$ est dans le cache peut se faire en

- $\mathcal{O}(1)$ opérations :
  - il faut un tableau B de booléens stockant si la page i est, ou pas, dans le cache
  - S'il existe une telle structure, il faut $\mathcal{O}(N)$ opérations pour l'initialiser.

Ce n'est pas raisonnable si N est grand.

(Peut-être utiliser une table de hachage)

- $\mathcal{O}(\log k)$ opérations :
  - il faut parcourir un arbre binaire de recherche équilibré qui maintient quelles sont les pages dans le cache.

Complexité de l'algorithme

Le test *le cache n'est pas plein* peut se faire en

- $\mathcal{O}(1)$ opérations:

  Il suffit de stocker le nombre de pages que contient le cache à chaque instant.

L'instruction *déterminer la page e dans le cache* dépend

- de la stratégie adopté pour choisir la page à supprimer.

Différentes stratégies adoptées pour choisir la page à supprimer.

- **LRU (Least Recently Used)** : on évince l'élément le moins récemment utilisé

  $\Rightarrow$ privilégie la récence d'accès.

- **LIFO (Last-In/First-Out)** : on évince le plus récent inséré

  $\Rightarrow$ fonctionne comme une pile (stack).

- **LFD (Longest Forward Distance)** : on supprime la page dont la demande suivante est la plus éloignée dans la séquence

  $\Rightarrow$ les décisions prises dépentent du futur.

Exemple : stratégie LFD sur l'instance $\sigma = \langle 4, 1, 2, 3, 1, 2, 4, 1, 2, 3 \rangle$

Exemple : stratégie LFD sur l'instance $\sigma = \langle 4, 1, 2, 3, 1, 2, 4, 1, 2, 3 \rangle$

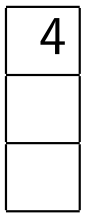
étape	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
$\sigma =$	$\langle$ 4,	1,	2,	3,	1,	2,	4,	1,	2,	3 $\rangle$

cache

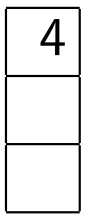
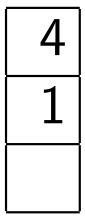
Exemple : stratégie LFD sur l'instance $\sigma = \langle 4, 1, 2, 3, 1, 2, 4, 1, 2, 3 \rangle$

étape	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
$\sigma =$	$\langle$ 4,	1,	2,	3,	1,	2,	4,	1,	2,	3 $\rangle$

cache



Exemple : stratégie LFD sur l'instance $\sigma = \langle 4, 1, 2, 3, 1, 2, 4, 1, 2, 3 \rangle$

étape	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
$\sigma =$	$\langle$ 4,	1,	2,	3,	1,	2,	4,	1,	2,	3 $\rangle$
cache										

Exemple : stratégie LFD sur l'instance $\sigma = \langle 4, 1, 2, 3, 1, 2, 4, 1, 2, 3 \rangle$

étape	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
$\sigma =$	$\langle$ 4,	1,	2,	3,	1,	2,	4,	1,	2,	3 $\rangle$
cache	4	4	4							
		1	1							
			2							

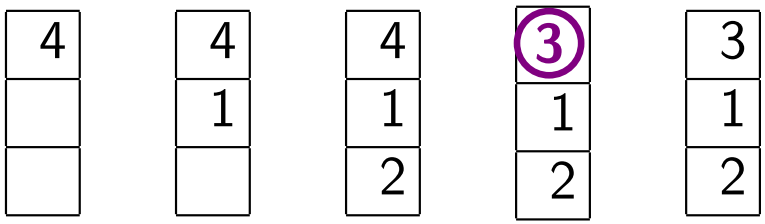
Exemple : stratégie LFD sur l'instance $\sigma = \langle 4, 1, 2, 3, 1, 2, 4, 1, 2, 3 \rangle$

étape	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
$\sigma =$	$\langle$ 4,	1,	2,	3,	1,	2,	4,	1,	2,	3 $\rangle$
cache	4	4	4	③						
		1	1	1						
			2	2						

Exemple : stratégie LFD sur l'instance $\sigma = \langle 4, 1, 2, 3, 1, 2, 4, 1, 2, 3 \rangle$

étape	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
$\sigma =$	$\langle$ 4,	1,	2,	3,	1,	2,	4,	1,	2,	3 $\rangle$

cache



Exemple : stratégie LFD sur l'instance $\sigma = \langle 4, 1, 2, 3, 1, 2, 4, 1, 2, 3 \rangle$

étape	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
$\sigma =$	$\langle$ 4,	1,	2,	3,	1,	2,	4,	1,	2,	3 $\rangle$
cache	4	4	4	③	3	3
		1	1	1	1	1
			2	2	2	2

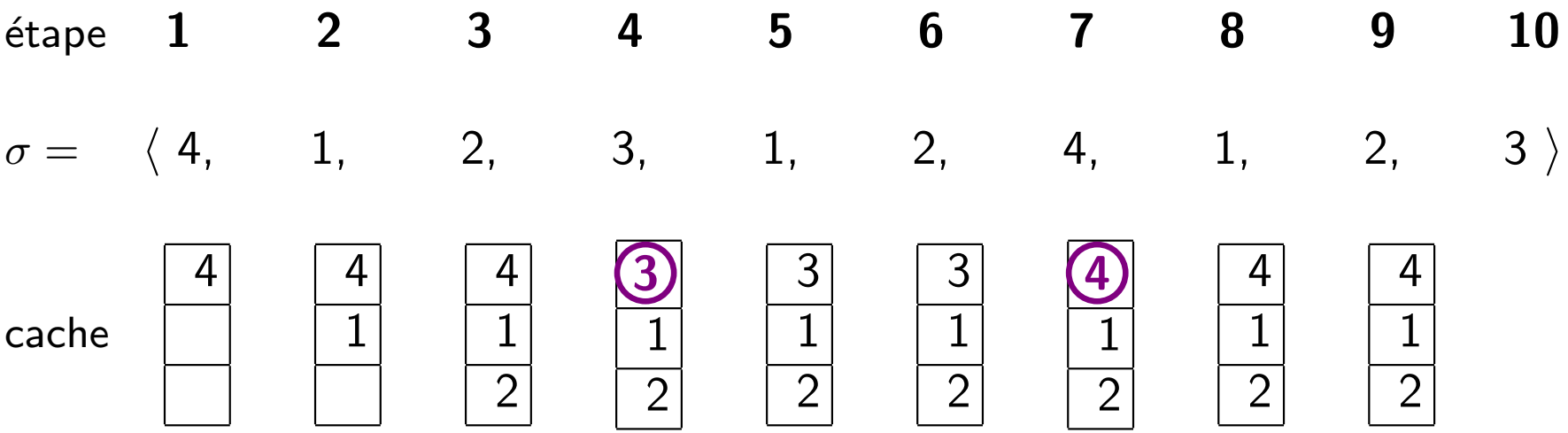
Exemple : stratégie LFD sur l'instance $\sigma = \langle 4, 1, 2, 3, 1, 2, 4, 1, 2, 3 \rangle$

étape	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
$\sigma =$	$\langle$ 4,	1,	2,	3,	1,	2,	4,	1,	2,	3 $\rangle$
cache	4	4	4	③	3	3	④			
		1	1	1	1	1	1			
			2	2	2	2	2			

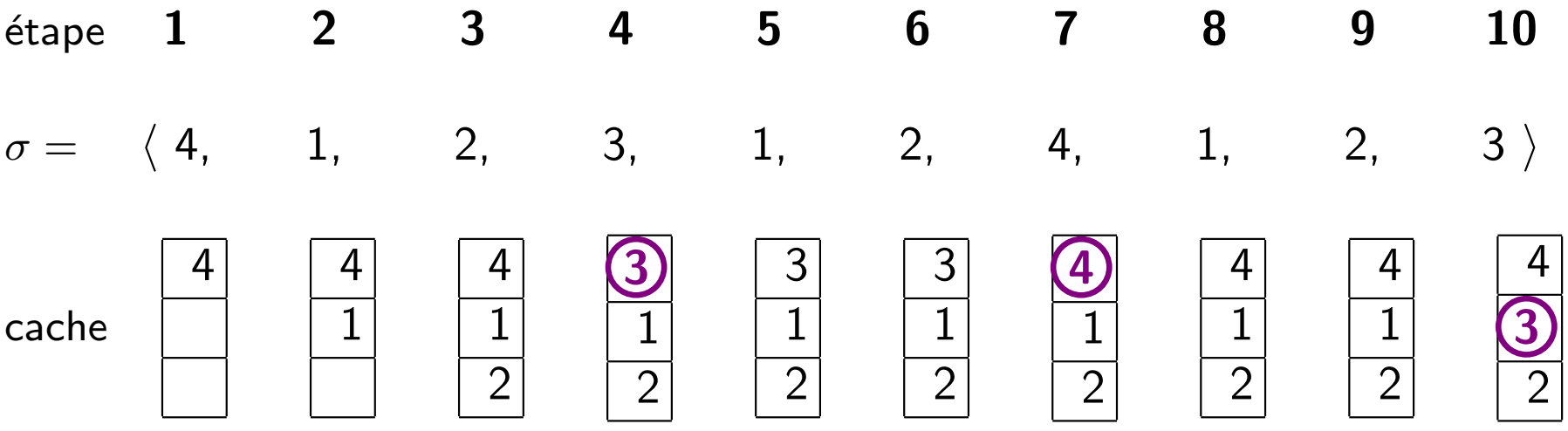
Exemple : stratégie LFD sur l'instance $\sigma = \langle 4, 1, 2, 3, 1, 2, 4, 1, 2, 3 \rangle$

étape	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
$\sigma =$	$\langle$ 4,	1,	2,	3,	1,	2,	4,	1,	2,	3 $\rangle$
cache	4	4	4	③	3	3	④	4		
		1	1	1	1	1	1	1		
			2	2	2	2	2	2		

Exemple : stratégie LFD sur l'instance $\sigma = \langle 4, 1, 2, 3, 1, 2, 4, 1, 2, 3 \rangle$



Exemple : stratégie LFD sur l'instance $\sigma = \langle 4, 1, 2, 3, 1, 2, 4, 1, 2, 3 \rangle$



Idée de la preuve

Notons $\mathrm{cout}_{\mathcal{S}}(\sigma)$: le nombre de pages supprimées par la stratégie $\mathcal{S}$ sur la séquence σ.

Soit σ une séquence de longueur $> i$, soit $\mathcal{A}$ une stratégie.

Il faut prouver que

Il existe une stratégie $\mathcal{B}$ telle que :

1. pendant les $i-1$ premières étapes, $\mathcal{B}$ se comporte comme $\mathcal{A}$;
2. à l'étape i, $\mathcal{B}$ utilise la stratégie LFD pour remplacer la page supprimée ;
3. $\mathrm{cout}_{\mathcal{B}}(\sigma) \leq \mathrm{cout}_{\mathcal{A}}(\sigma)$.

Illustration de l'argument d'échange

À l'étape i, $\begin{cases} \mathcal{A} \text{ évince } p, \\ \mathcal{B} \text{ évince } q \text{ (dont la prochaine demande est la plus tardive).} \end{cases}$

Ensuite, $\begin{cases} \mathcal{B} \text{ suit exactement le comportement de } \mathcal{A}, \\ \text{sauf si } \mathcal{A} \text{ choisit d'évincer } q \text{ à l'étape } t \text{ alors } \mathcal{B} \text{ évince } p \end{cases}$

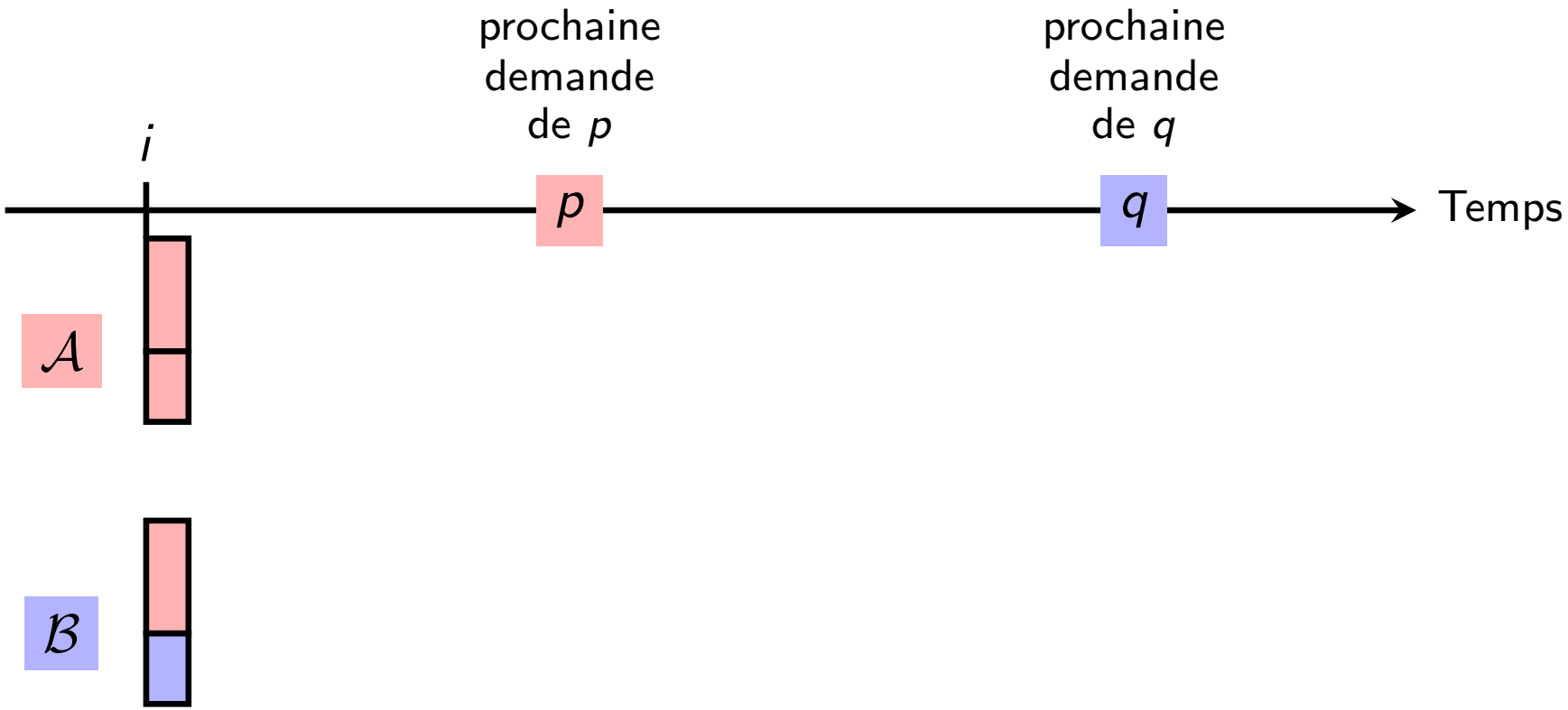


Illustration de l'argument d'échange

À l'étape i, $\begin{cases} \mathcal{A} \text{ évince } p, & \textbf{la page q est dans le cache pour } \mathcal{A} \\ \mathcal{B} \text{ évince } q & \textbf{la page p est dans le cache pour } \mathcal{B}. \end{cases}$

Ensuite, $\begin{cases} \mathcal{B} \text{ suit exactement le comportement de } \mathcal{A}, \\ \text{sauf si } \mathcal{A} \text{ choisit d'évincer } q \text{ à l'étape } t \text{ alors } \mathcal{B} \text{ évince } p \end{cases}$

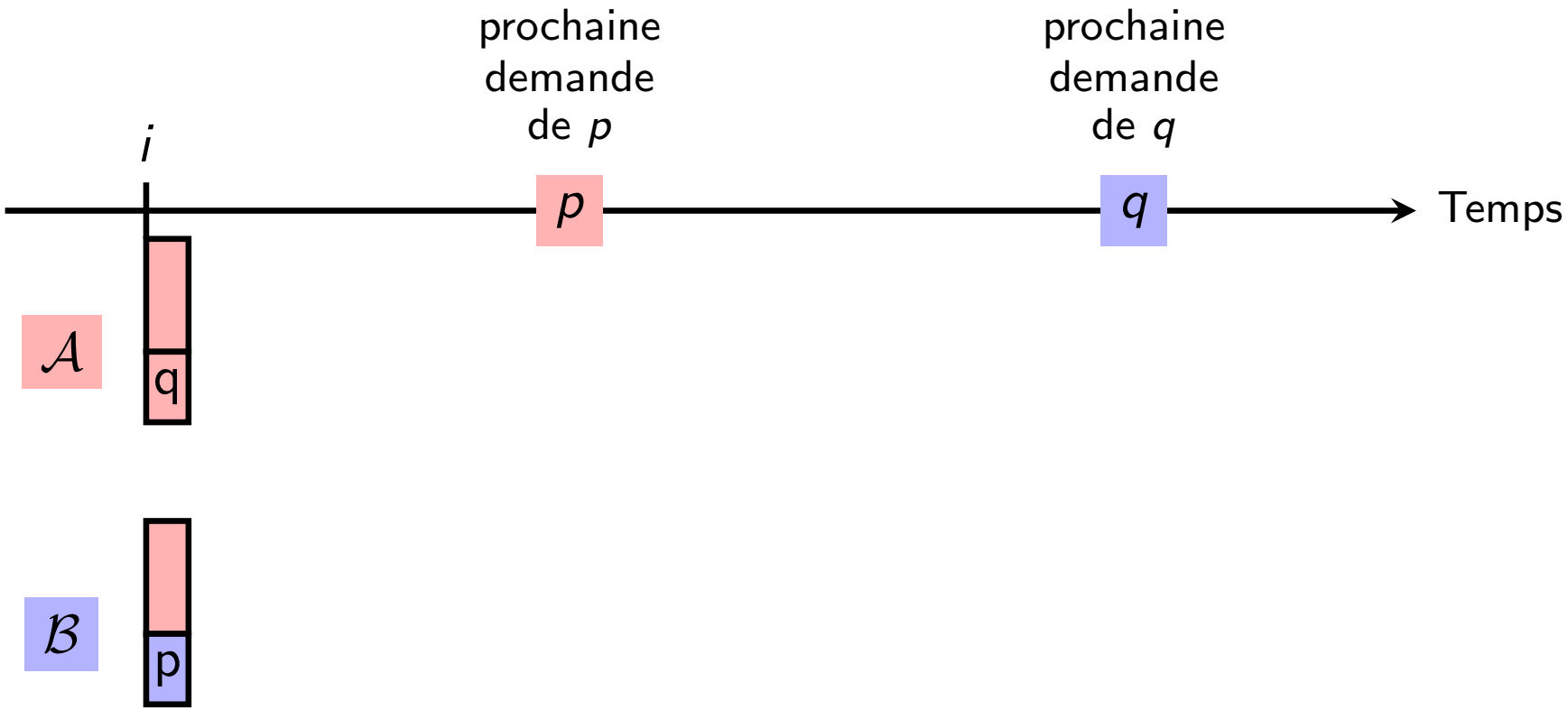


Illustration de l'argument d'échange

À l'étape i, $\begin{cases} \mathcal{A} \text{ évince } p, & \textbf{la page q est dans le cache pour } \mathcal{A} \\ \mathcal{B} \text{ évince } q & \textbf{la page p est dans le cache pour } \mathcal{B}. \end{cases}$

Ensuite, $\begin{cases} \mathcal{B} \text{ suit exactement le comportement de } \mathcal{A}, \\ \text{sauf si } \mathcal{A} \text{ choisit d'évincer } q \text{ à l'étape } t \text{ alors } \mathcal{B} \text{ évince } p \end{cases}$

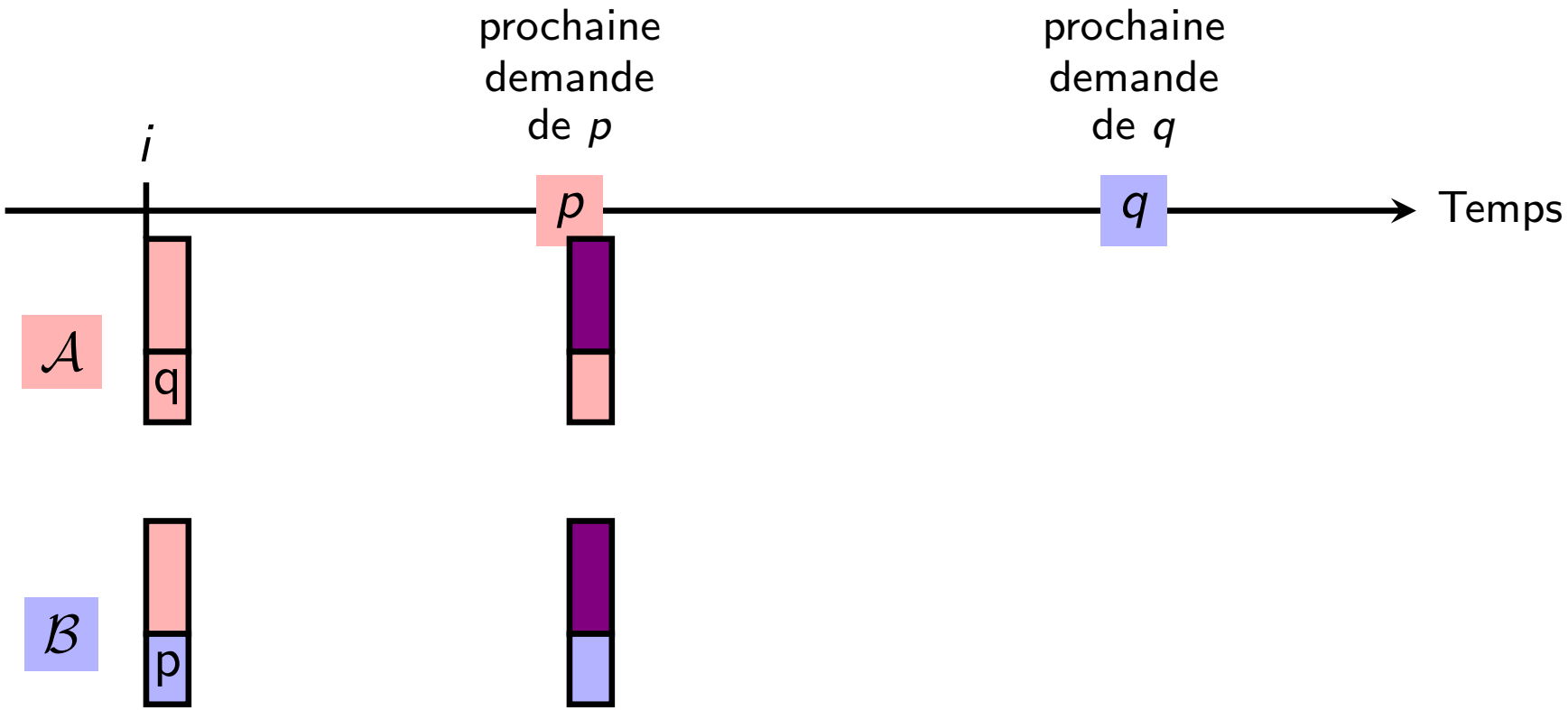
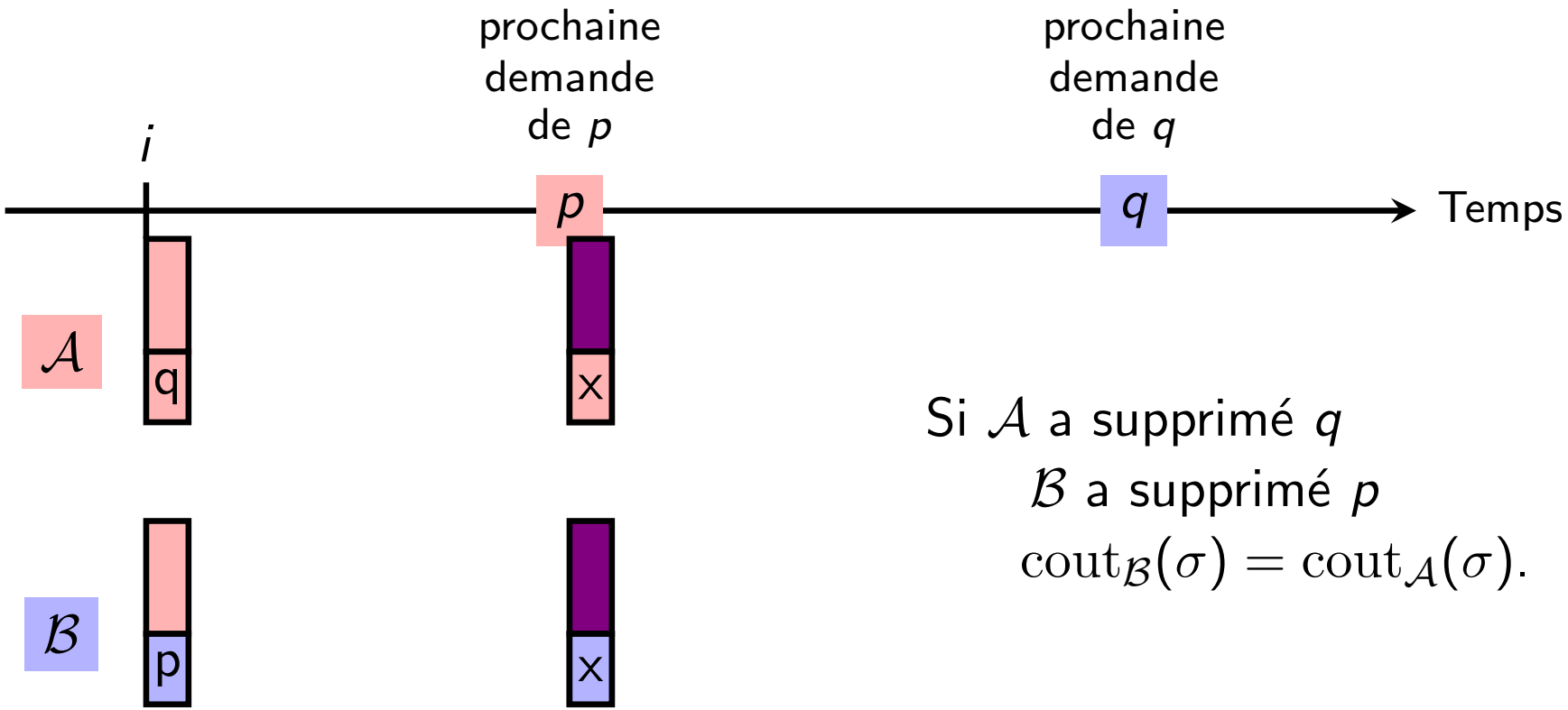


Illustration de l'argument d'échange

À l'étape i, $\begin{cases} \mathcal{A} \text{ évince } p, & \textbf{la page q est dans le cache pour } \mathcal{A} \\ \mathcal{B} \text{ évince } q & \textbf{la page p est dans le cache pour } \mathcal{B}. \end{cases}$

Ensuite, $\begin{cases} \mathcal{B} \text{ suit exactement le comportement de } \mathcal{A}, \\ \text{sauf si } \mathcal{A} \text{ choisit d'évincer } q \text{ à l'étape } t \text{ alors } \mathcal{B} \text{ évince } p \end{cases}$



Si $\mathcal{A}$ a supprimé q
$\mathcal{B}$ a supprimé p
$\text{cout}_{\mathcal{B}}(\sigma) = \text{cout}_{\mathcal{A}}(\sigma)$.

Illustration de l'argument d'échange

À l'étape i, $\begin{cases} \mathcal{A} \text{ évince } p, \\ \mathcal{B} \text{ évince } q \text{ (dont la prochaine demande est la plus tardive).} \end{cases}$

Ensuite, $\begin{cases} \mathcal{B} \text{ suit exactement le comportement de } \mathcal{A}, \\ \text{sauf si } \mathcal{A} \text{ choisit d'évincer } q \text{ à l'étape } t \text{ alors } \mathcal{B} \text{ évince } p \end{cases}$

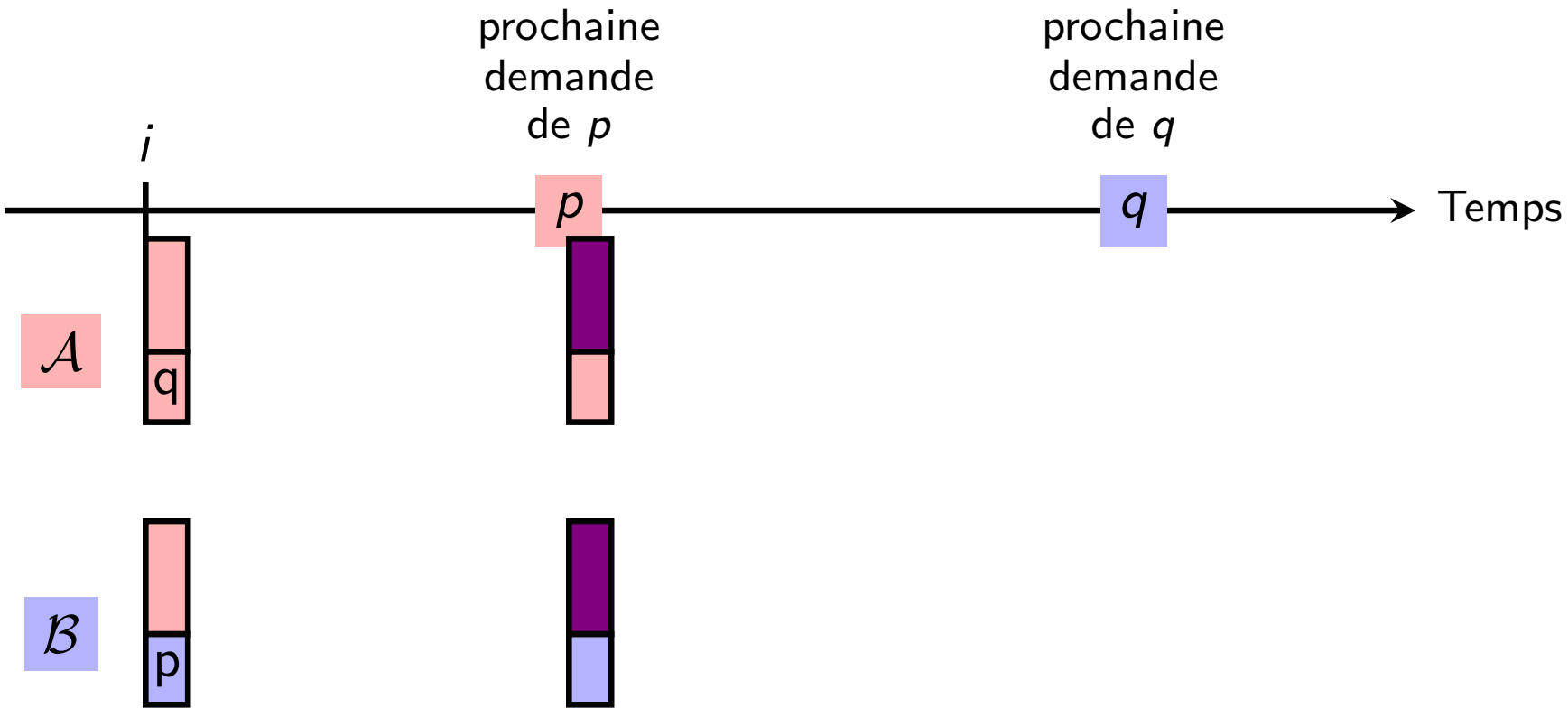
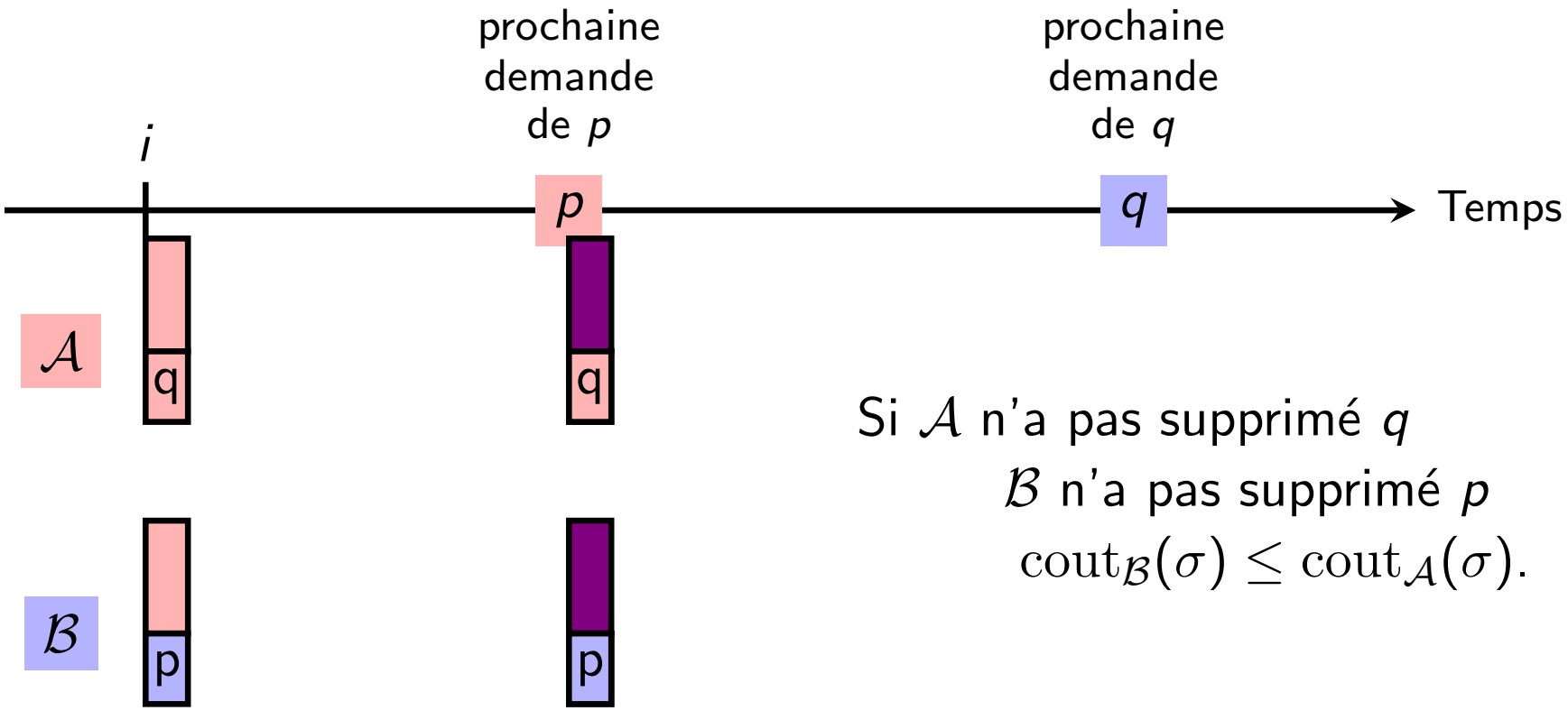


Illustration de l'argument d'échange

À l'étape i, $\begin{cases} \mathcal{A} \text{ évince } p, \\ \mathcal{B} \text{ évince } q \text{ (dont la prochaine demande est la plus tardive).} \end{cases}$

Ensuite, $\begin{cases} \mathcal{B} \text{ suit exactement le comportement de } \mathcal{A}, \\ \text{sauf si } \mathcal{A} \text{ choisit d'évincer } q \text{ à l'étape } t \text{ alors } \mathcal{B} \text{ évince } p \end{cases}$



Si $\mathcal{A}$ n'a pas supprimé q
$\mathcal{B}$ n'a pas supprimé p
$\text{cout}_{\mathcal{B}}(\sigma) \leq \text{cout}_{\mathcal{A}}(\sigma)$.

Idée de la preuve

Soit σ une séquence de longueur $> i$, soit $\mathcal{A}$ une stratégie,
il existe une stratégie $\mathcal{B}$ telle que :

1. pendant les $i-1$ premières étapes, $\mathcal{B}$ se comporte comme $\mathcal{A}$;
2. à l'étape i, $\mathcal{B}$ utilise la stratégie LFD pour remplacer la page supprimée;
3. $\mathrm{cout}_{\mathcal{B}}(\sigma) \leq \mathrm{cout}_{\mathcal{A}}(\sigma)$.

On peut en déduire que la stratégie qui consiste à utiliser LFD pour choisir la page à remplacer est optimale. Car,

Idée de la preuve

Soit σ une séquence de longueur $> i$, soit $\mathcal{A}$ une stratégie,
il existe une stratégie $\mathcal{B}$ telle que :

1. pendant les $i-1$ premières étapes, $\mathcal{B}$ se comporte comme $\mathcal{A}$;
2. à l'étape i, $\mathcal{B}$ utilise la stratégie LFD pour remplacer la page supprimée;
3. $\mathrm{cout}_{\mathcal{B}}(\sigma) \leq \mathrm{cout}_{\mathcal{A}}(\sigma)$.

On peut en déduire que la stratégie qui consiste à utiliser LFD pour choisir la page à remplacer est optimale. Car,

toute solution optimale
qui ne suit pas la
stratégie LFD pendant
x étapes

Idée de la preuve

Soit σ une séquence de longueur $> i$, soit $\mathcal{A}$ une stratégie,
il existe une stratégie $\mathcal{B}$ telle que :

1. pendant les $i-1$ premières étapes, $\mathcal{B}$ se comporte comme $\mathcal{A}$;
2. à l'étape i, $\mathcal{B}$ utilise la stratégie LFD pour remplacer la page supprimée;
3. $\mathrm{cout}_{\mathcal{B}}(\sigma) \leq \mathrm{cout}_{\mathcal{A}}(\sigma)$.

On peut en déduire que la stratégie qui consiste à utiliser LFD pour choisir la page à remplacer est optimale. Car,

toute solution optimale qui ne suit pas la stratégie LFD pendant **x étapes**

peut être transformée en

Idée de la preuve

Soit σ une séquence de longueur $> i$, soit $\mathcal{A}$ une stratégie,
il existe une stratégie $\mathcal{B}$ telle que :

1. pendant les $i-1$ premières étapes, $\mathcal{B}$ se comporte comme $\mathcal{A}$;
2. à l'étape i, $\mathcal{B}$ utilise la stratégie LFD pour remplacer la page supprimée;
3. $\mathrm{cout}_{\mathcal{B}}(\sigma) \leq \mathrm{cout}_{\mathcal{A}}(\sigma)$.

On peut en déduire que la stratégie qui consiste à utiliser LFD pour choisir la page à remplacer est optimale. Car,

toute solution optimale qui ne suit pas la stratégie LFD pendant **x étapes**

peut être transformée en

une solution optimale qui ne suit pas la stratégie LFD pendant **x − 1 étapes**.

Récapitulatif

Type d' instructions	complexité en nbre d'opérations	complexité en place mémoire
décider si $\sigma[i]$ est dans le cache		
tester le cache n'est pas plein		
Déterminer la page à supprimer (LIFO)		
Déterminer la page à supprimer (LRU)		
Déterminer la page à supprimer (LFD)		

Attention :

Récapitulatif

Type d' instructions	complexité en nbre d'opérations	complexité en place mémoire
décider si $\sigma[i]$ est dans le cache	$\mathcal{O}(1)$	$\mathcal{O}(N)$
tester le cache n'est pas plein		
Déterminer la page à supprimer (LIFO)		
Déterminer la page à supprimer (LRU)		
Déterminer la page à supprimer (LFD)		

Attention :

Récapitulatif

Type d' instructions	complexité en nbre d'opérations	complexité en place mémoire
décider si $\sigma[i]$ est dans le cache	$\mathcal{O}(1)$	$\mathcal{O}(N)$
tester le cache n'est pas plein	$\mathcal{O}(1)$	$\mathcal{O}(1)$
Déterminer la page à supprimer (LIFO)		
Déterminer la page à supprimer (LRU)		
Déterminer la page à supprimer (LFD)		

Attention :

Récapitulatif

Type d' instructions	complexité en nbre d'opérations	complexité en place mémoire
décider si $\sigma[i]$ est dans le cache	$\mathcal{O}(1)$	$\mathcal{O}(N)$
tester le cache n'est pas plein	$\mathcal{O}(1)$	$\mathcal{O}(1)$
Déterminer la page à supprimer (LIFO)	$\mathcal{O}(1)$	$\mathcal{O}(1)$
Déterminer la page à supprimer (LRU)		
Déterminer la page à supprimer (LFD)		

Attention :

Récapitulatif

Type d' instructions	complexité en nbre d'opérations	complexité en place mémoire
décider si $\sigma[i]$ est dans le cache	$\mathcal{O}(1)$	$\mathcal{O}(N)$
tester le cache n'est pas plein	$\mathcal{O}(1)$	$\mathcal{O}(1)$
Déterminer la page à supprimer (LIFO)	$\mathcal{O}(1)$	$\mathcal{O}(1)$
Déterminer la page à supprimer (LRU)	$\mathcal{O}(1)$	$\mathcal{O}(k)$
Déterminer la page à supprimer (LFD)		

Attention :

Récapitulatif

Type d' instructions	complexité en nbre d'opérations	complexité en place mémoire
décider si $\sigma[i]$ est dans le cache	$\mathcal{O}(1)$	$\mathcal{O}(N)$
tester le cache n'est pas plein	$\mathcal{O}(1)$	$\mathcal{O}(1)$
Déterminer la page à supprimer (LIFO)	$\mathcal{O}(1)$	$\mathcal{O}(1)$
Déterminer la page à supprimer (LRU)	$\mathcal{O}(1)$	$\mathcal{O}(k)$
Déterminer la page à supprimer (LFD)	$\mathcal{O}(\log k)$	$\mathcal{O}(k)$

Attention :

Récapitulatif

Type d' instructions	complexité en nbre d'opérations	complexité en place mémoire
décider si $\sigma[i]$ est dans le cache	$\mathcal{O}(1)$	$\mathcal{O}(N)$
tester le cache n'est pas plein	$\mathcal{O}(1)$	$\mathcal{O}(1)$
Déterminer la page à supprimer (LIFO)	$\mathcal{O}(1)$	$\mathcal{O}(1)$
Déterminer la page à supprimer (LRU)	$\mathcal{O}(1)$	$\mathcal{O}(k)$
Déterminer la page à supprimer (LFD)	$\mathcal{O}(\log k)$	$\mathcal{O}(k)$

Attention : les stratégies LIFO et LRU ne sont pas optimales pour le nombre de suppressions de pages.

Aujourd'hui

- Problèmes d'optimisation
- Algorithmes gloutons
  - problème du stockage (problème du sac-à-dos)..
  - Gestion de cache

La semaine prochaine :

Programmation dynamique.