Systèmes de fichiers

Thomas Lavergne
lavergne@lisn.fr

Partie visible de l'OS

- Stockage sur support physique
- Accès aux données et programmes
- Accès à l'ensemble du système

Partie visible de l'OS

- Stockage sur support physique
- Accès aux données et programmes
- Accès à l'ensemble du système

Définition

Un fichier est une collection nommée d'informations accessibles via un périphérique.

Partie visible de l'OS

- Stockage sur support physique
- Accès aux données et programmes
- Accès à l'ensemble du système

Définition

Un fichier est une collection nommée d'informations accessibles via un périphérique.

Unité *logique*

- Indépendante du support physique
- Abstraction des propriétés physiques

Type de fichier

- Code source
- Données
- Bibliothèque
- Fichier exécutable
- Point d'accès à un périphérique
- etc

$\rightarrow$ À chaque type de fichier correspond une structure spécifique !

Généralement indiqué par son extension.

Structure d'un fichier : exemples

Fichiers texte : `.txt`

Données textuelles à l'usage de l'utilisateur humain

$\longrightarrow$ Succession de caractères, organisés en lignes séparées par un caractère spécial (`\n`, `^M`, `...` `selon l'OS`)

Structure d'un fichier : exemples

Fichiers texte : `.txt`

Données textuelles à l'usage de l'utilisateur humain

$\longrightarrow$ Succession de caractères, organisés en lignes séparées par un caractère spécial (`\n`, `^M`, ... selon l'OS)

Fichiers source : `.c`, `.java`, ...

Fourni par un utilisateur humain pour être traité par la machine

$\longrightarrow$ succession de fonctions et sous-programmes, composés d'instructions séparées par des symboles spécifiques

Structure d'un fichier : exemples

Bibliothèques : `.o`, `.dll`, ...

Construits par un compilateur à partir d'un fichier source

$\longrightarrow$ Succession d'octets organisés en blocs interprétables par l'éditeur de lien.

Structure d'un fichier : exemples

Bibliothèques : `.o`, `.dll`, ...

Construits par un compilateur à partir d'un fichier source

$\longrightarrow$ Succession d'octets organisés en blocs interprétables par l'éditeur de lien.

Exécutable : `.exe`, ...

$\longrightarrow$ Succession d'instructions que l'OS peut charger en mémoire pour exécuter un programme.

File Control Block

Définition

Structure de données de l'OS pour stocker les informations nécessaires à la gestion des fichiers

File Control Block

Définition

Structure de données de l'OS pour stocker les informations nécessaires à la gestion des fichiers

Contenu

Identifiant : numérique, unique, pour l'OS

Emplacement : pointeur sur un périphérique

Taille : en octets ou en blocs

Protection : lecture, écriture, exécution...

Date(s) : création, modification, accès...

Utilisateur : propriétaire du fichier

Définition

Rendre accessible à un utilisateur B un fichier de l'utilisateur A

Exemple : lecture de `/bin/sh`, accès à `/dev/mouse0`, ...

Définition

Rendre accessible à un utilisateur B un fichier de l'utilisateur A

`Exemple : lecture de /bin/sh, accès à /dev/mouse0, ...`

Politique de protection

Définir qui peut accéder à quel(s) fichier(s)

- Identifiant utilisateur $\longrightarrow$ identifiant processus
- Contrôle d'accès dans le FCB

Liste de contrôle d'accès (ACL)

Utilisateur $\longrightarrow$ droits

Problèmes :

- X les utilisateurs doivent être connus a priori
- X taille du FCB ! (grossit avec le nombre d'utilisateurs)

Liste de contrôle d'accès (ACL)

Utilisateur $\longrightarrow$ droits

Problèmes :

- ✗ les utilisateurs doivent être connus a priori
- ✗ taille du FCB ! (grossit avec le nombre d'utilisateurs)

Mot de passe

1 mot de passe par fichier $\times$ type d'accès (lecture, écriture, ...)

- ✗ Pas très pratique $\longrightarrow$ peu utilisé

Classes d'utilisateurs

Exemple : Propriétaires vs Autres

$\longrightarrow$ quelques bits par fichier

Classes d'utilisateurs

Exemple : Propriétaires vs Autres

$\longrightarrow$ quelques bits par fichier

Notion de groupe

✓ Ensemble de groupes définis a priori

Ex : `admin, dev-disque, user-disque, dev-ram,user-ram`

✓ FCB : 1 utilisateur + 1 groupe (propriétaires)

Ex : `toto.c u=batman, g=dev-disque`

✓ Utilisateur $\longrightarrow$ liste de groupes

Ex : `robin, g=[dev-ram,user-disque]`

$\longrightarrow$ `robin` n'a pas accès à `toto.c`

Exemple : Unix

Classes + groupes

- 3 classes : utilisateur, groupe, autres
- N groupes
- 3 droits : read, write, execute

$\rightarrow$ 3×3 bits par fichier

Exemple : Unix

Classes + groupes

- 3 classes : utilisateur, groupe, autres
- N groupes
- 3 droits : read, write, execute

$\longrightarrow$ 3×3 bits par fichier

Exemple :

```
$ ls -l
drwx------ 6 tom prof 4096 nov.  2 12:06 private
-rwx------ 1 tom prof 2356 déc.  5 15:31 notes.tex
drwxrwx--- 8 tom prof 4096 déc.  4 17:30 doc
-rwxrwxr-x 1 joe stud 9234 jan.  1 11:12 prog.c
```

Notion de répertoire

Définition

Le répertoire est la structure de stockage des informations des fichiers (les FCB) sur le périphérique.

Notion de répertoire

Définition

Le répertoire est la structure de stockage des informations des fichiers (les FCB) sur le périphérique.

Structure

- Entrée du répertoire = id et/ou nom du fichier
  $\rightarrow$ deux méthodes d'accès
- Contenu du répertoire = FCB des fichiers
  $\rightarrow$ plusieurs ko de données

Principe

L'OS récupère l'information sur les fichiers dans le répertoire

Interface

Opérations sur un fichier

Appels systèmes de base

- Création : allocation espace + entrée répertoire
- Lecture : pointeur de lecture
- Écriture : pointeur d'écriture
- Repositionnement : déplacer le pointeur
- Suppression : retrait de l'entrée dans le répertoire
- Troncature : changement de taille

Opérations sur un fichier

Appels systèmes de base

- Création : allocation espace + entrée répertoire
- Lecture : pointeur de lecture
- Écriture : pointeur d'écriture
- Repositionnement : déplacer le pointeur
- Suppression : retrait de l'entrée dans le répertoire
- Troncature : changement de taille

Opérations composées

Ex : copie, renommage

$\longrightarrow$ effectuées à partir des appels systèmes de base

Problème

- Nécessité d'accéder au FCB à chaque opération sur le fichier
- Le FCB est stocké dans le répertoire

$\longrightarrow$ Très coûteux en accès disque (donc en temps) !

Ouverture de fichier

Problème

- Nécessité d'accéder au FCB à chaque opération sur le fichier
- Le FCB est stocké dans le répertoire

$\longrightarrow$ Très coûteux en accès disque (donc en temps) !

Définition

L'appel système `open` permet de charger le FCB en mémoire

Ouverture de fichier

Problème

- Nécessité d'accéder au FCB à chaque opération sur le fichier
- Le FCB est stocké dans le répertoire

$\longrightarrow$ Très coûteux en accès disque (donc en temps) !

Définition

L'appel système `open` permet de charger le FCB en mémoire

Accès à un fichier

L'OS impose que tout accès à un fichier soit précédé d'une ouverture.

Stockage des FCB en RAM

La table des fichiers ouverts de l'OS contient les FCB des fichiers ouverts.

- Ouverture $\rightarrow$ chargement du FCB
- Fermeture $\rightarrow$ retrait de la table
- Pas d'impact sur le fichier !

Stockage des FCB en RAM

La table des fichiers ouverts de l'OS contient les FCB des fichiers ouverts.

- Ouverture $\rightarrow$ chargement du FCB
- Fermeture $\rightarrow$ retrait de la table
- Pas d'impact sur le fichier !

Gestion par l'OS

- Implicite : open implicite au premier accès
- Explicite : exception si le fichier n'a pas été ouvert avant
- Une table globale + une table par processus

Système de fichiers

Différence avec la RAM

- Grande quantité de données
- Accès lent (rapport 10^3 à 10^6)

Différence avec la RAM

- Grande quantité de données
- Accès lent (rapport 10^3 à 10^6)

Notion de bloc

- Unité de base du support physique
  - Les données à stocker sont découpées en bloc
- Taille fixe (de 32o à 16Mo selon support)

Exemples : disques dur années 2000 = 512o, SSD actuel = 1Mo

Différence avec la RAM

- Grande quantité de données
- Accès lent (rapport 10^3 à 10^6)

Notion de bloc

- Unité de base du support physique
  - Les données à stocker sont découpées en bloc
- Taille fixe (de 32o à 16Mo selon support)

Exemples : disques dur années 2000 = 512o, SSD actuel = 1Mo

Système de fichiers *(FileSystem)*

Organisation des fichiers en blocs (*cf. mémoire paginée*)

Système logique

- Structure de répertoires
- FCB + gestion de la protection

Structure d'un système de fichiers

Système logique

- Structure de répertoires
- FCB + gestion de la protection

Système physique

- Fichiers $\longrightarrow$ ensemble de blocs logiques
- Bloc logique $\longrightarrow$ blocs physique *cf. mémoire paginée*
- Identification des blocs physiques selon support

Structure d'un système de fichiers

Système logique

- Structure de répertoires
- FCB + gestion de la protection

Système physique

- Fichiers $\longrightarrow$ ensemble de blocs logiques
- Bloc logique $\longrightarrow$ blocs physique *cf. mémoire paginée*
- Identification des blocs physiques selon support

Lien : pilote de périphérique

ex : chargement bloc 456 $\longrightarrow$ instruction matérielle

Montage de répertoire

Le système de fichiers associe...

... des noms à des blocs logiques (fichiers)

... des noms à des répertoires venus du disque !

Le système de fichiers associe...

... des noms à des blocs logiques (fichiers)

... des noms à des répertoires venus du disque !

Montage de répertoire

Le montage consiste à positionner un répertoire dans le système de fichier

Le système de fichiers associe...

... des noms à des blocs logiques (fichiers)

... des noms à des répertoires venus du disque !

Montage de répertoire

Le montage consiste à positionner un répertoire dans le système de fichier

Principe

- Chargement du FCB par l'OS (disque → RAM)
- Association dans le MFD au niveau logique

→ Les fichiers deviennent accessibles

Accès à un fichier

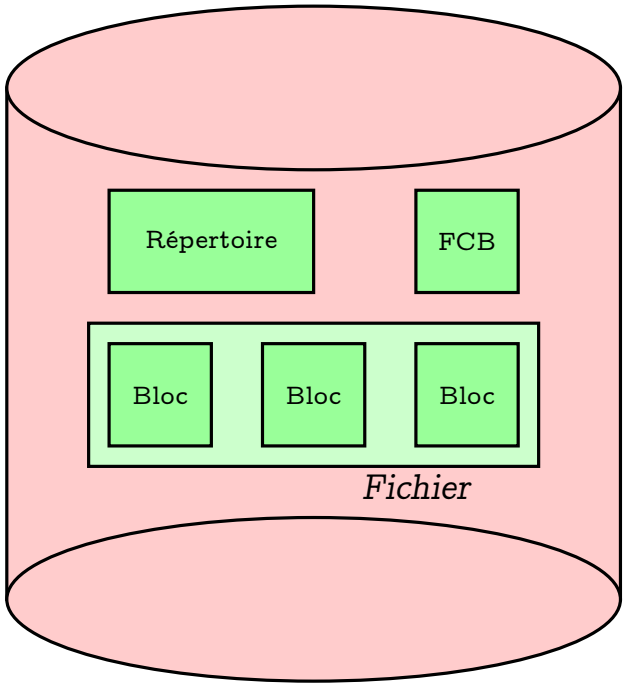
Application

RAM (OS)

Cache répertoires

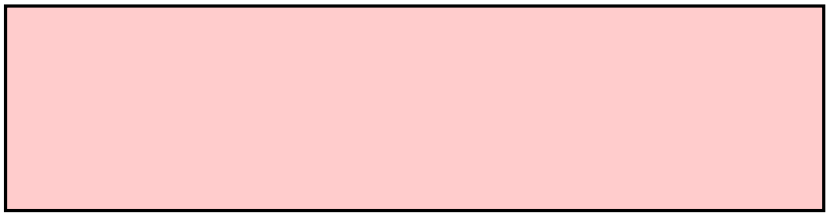
Table fichiers ouverts

Disque

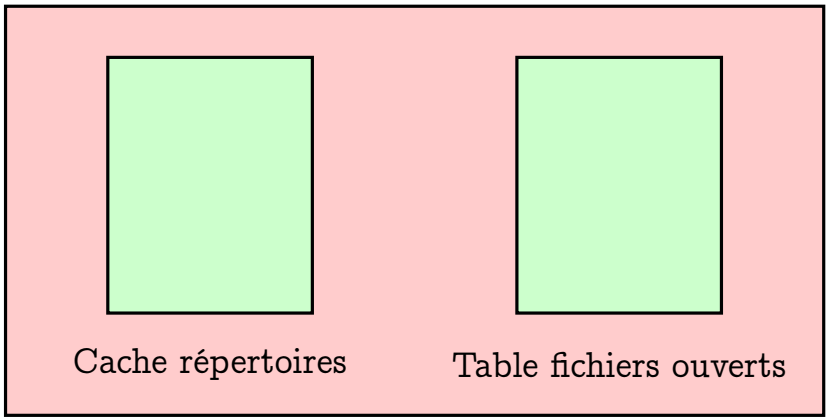


Accès à un fichier

Application

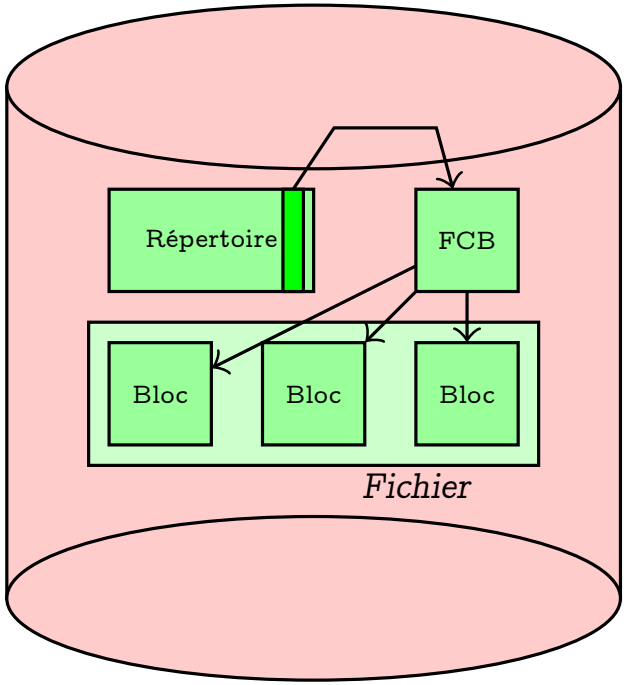


RAM (OS)



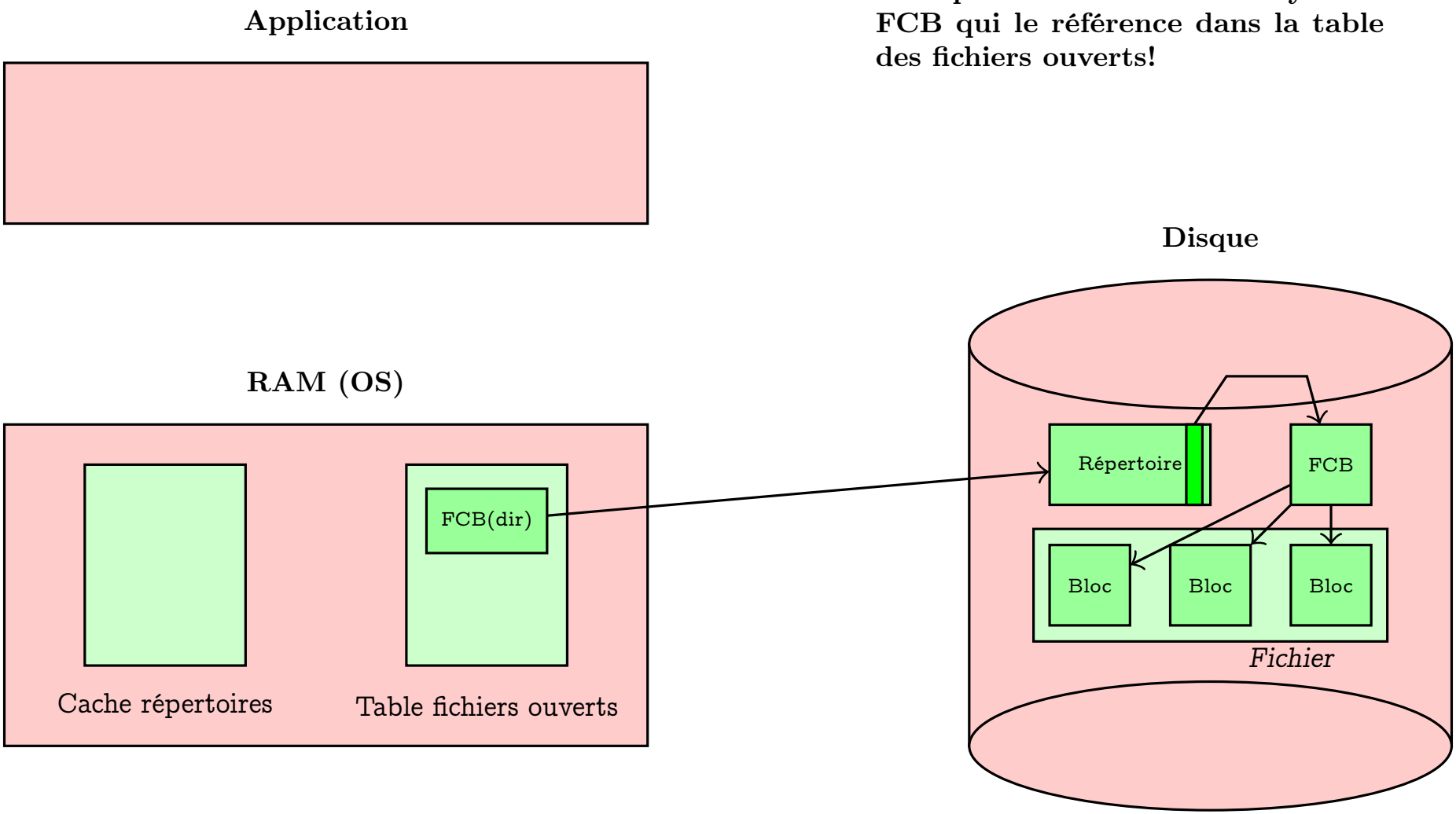
Le système de fichier (sur le disque) définit la structuration des données

Disque



Accès à un fichier

Le répertoire est monté : il y a un FCB qui le référence dans la table des fichiers ouverts!



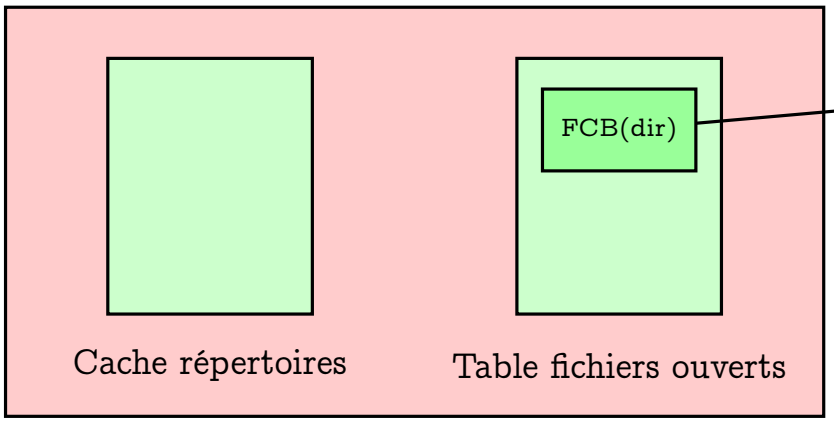
Accès à un fichier

Lors du premier accès à un fichier...

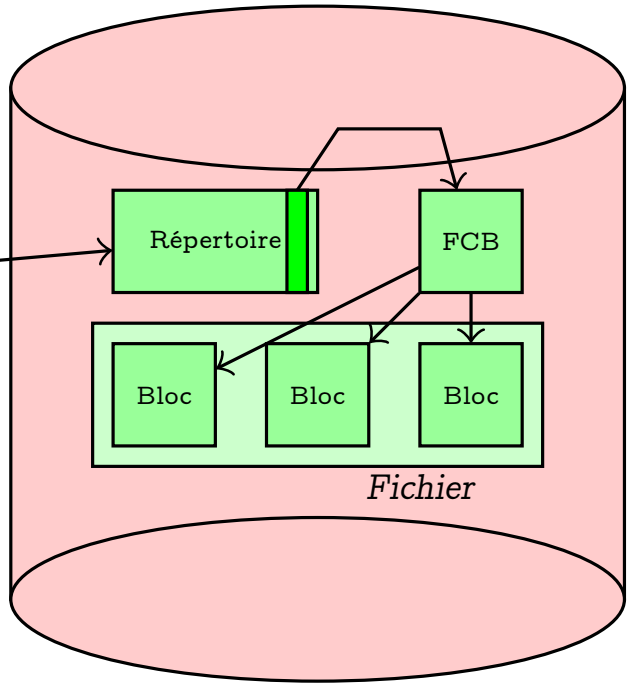
Application

```
open(dir/file.ext)
```

RAM (OS)

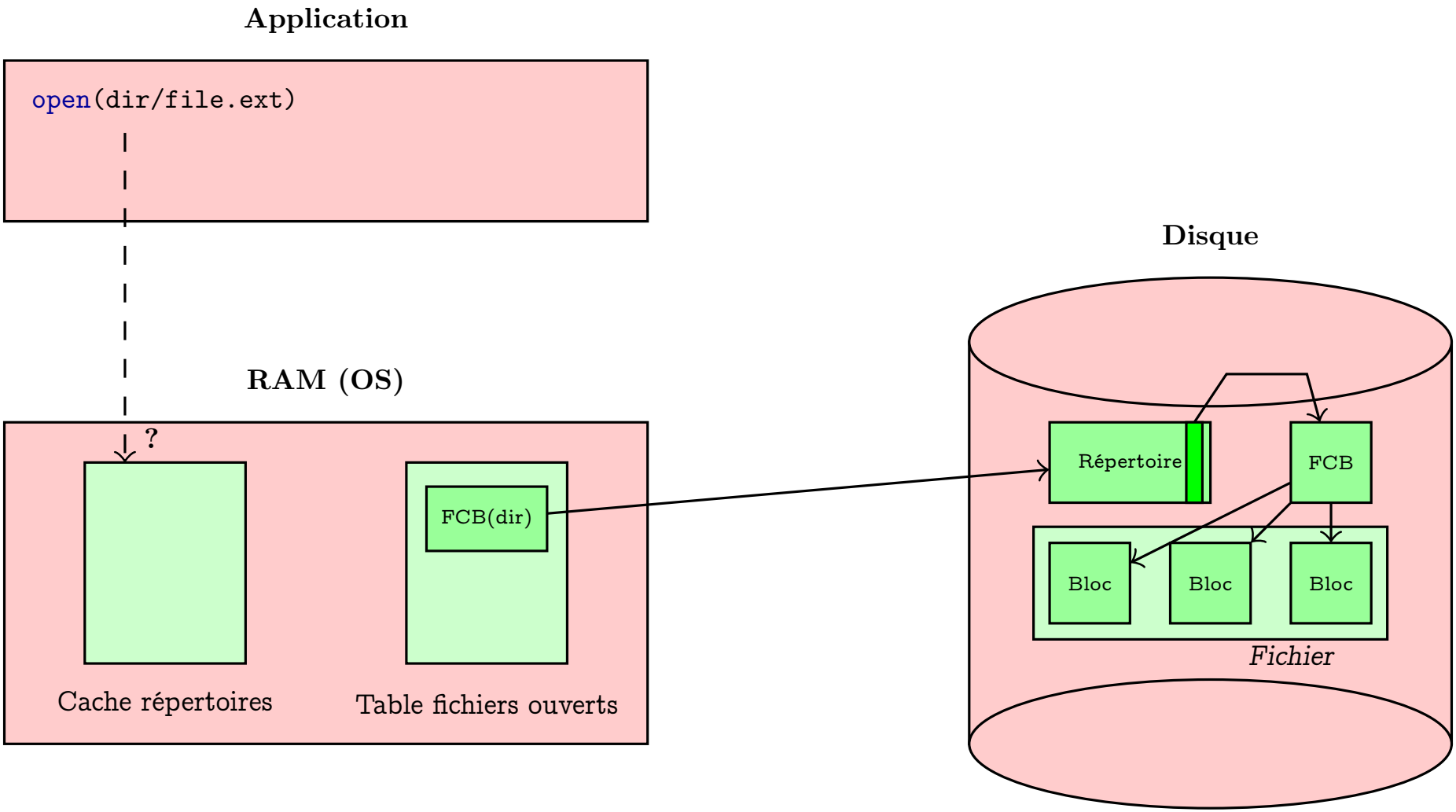


Disque



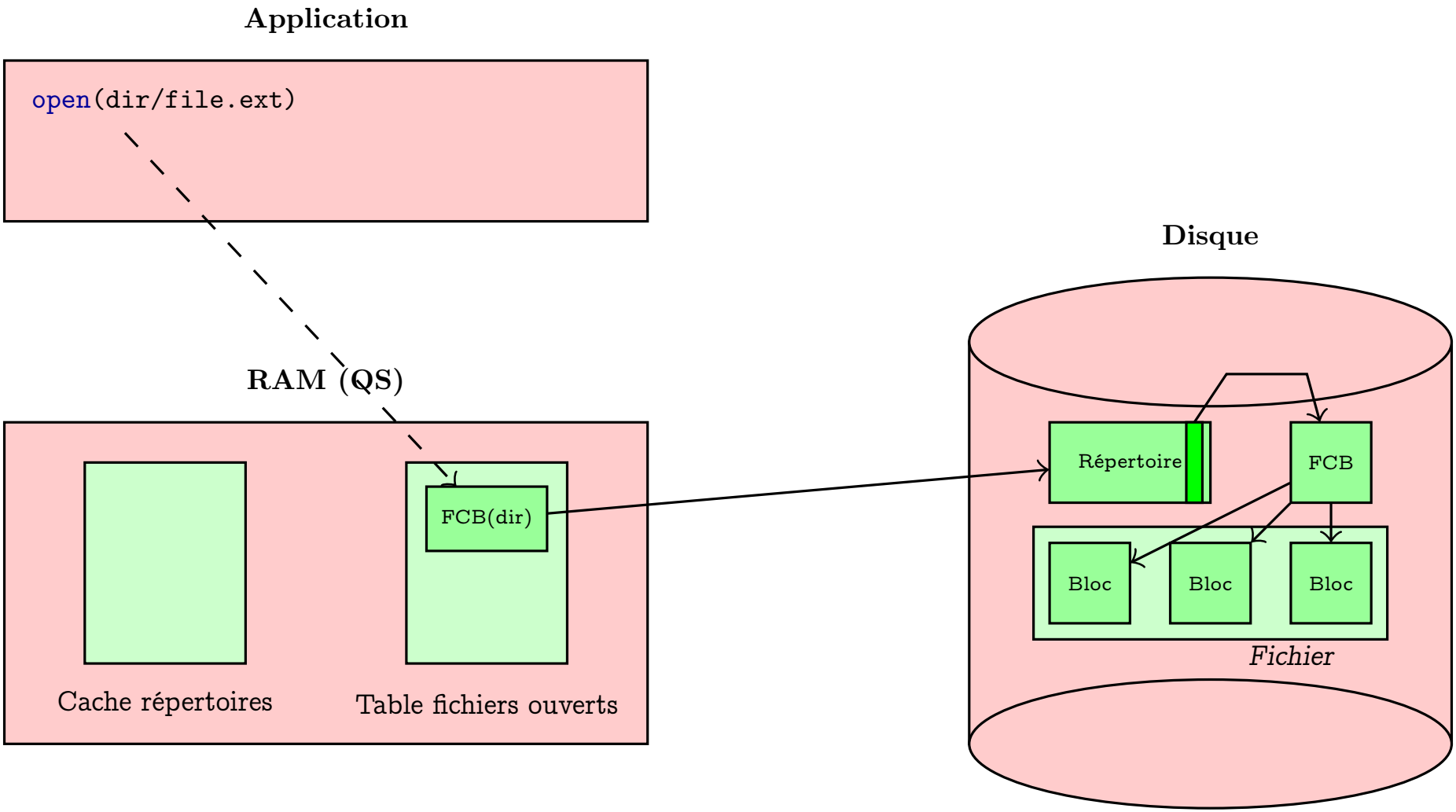
Accès à un fichier

Lors du premier accès à un fichier...



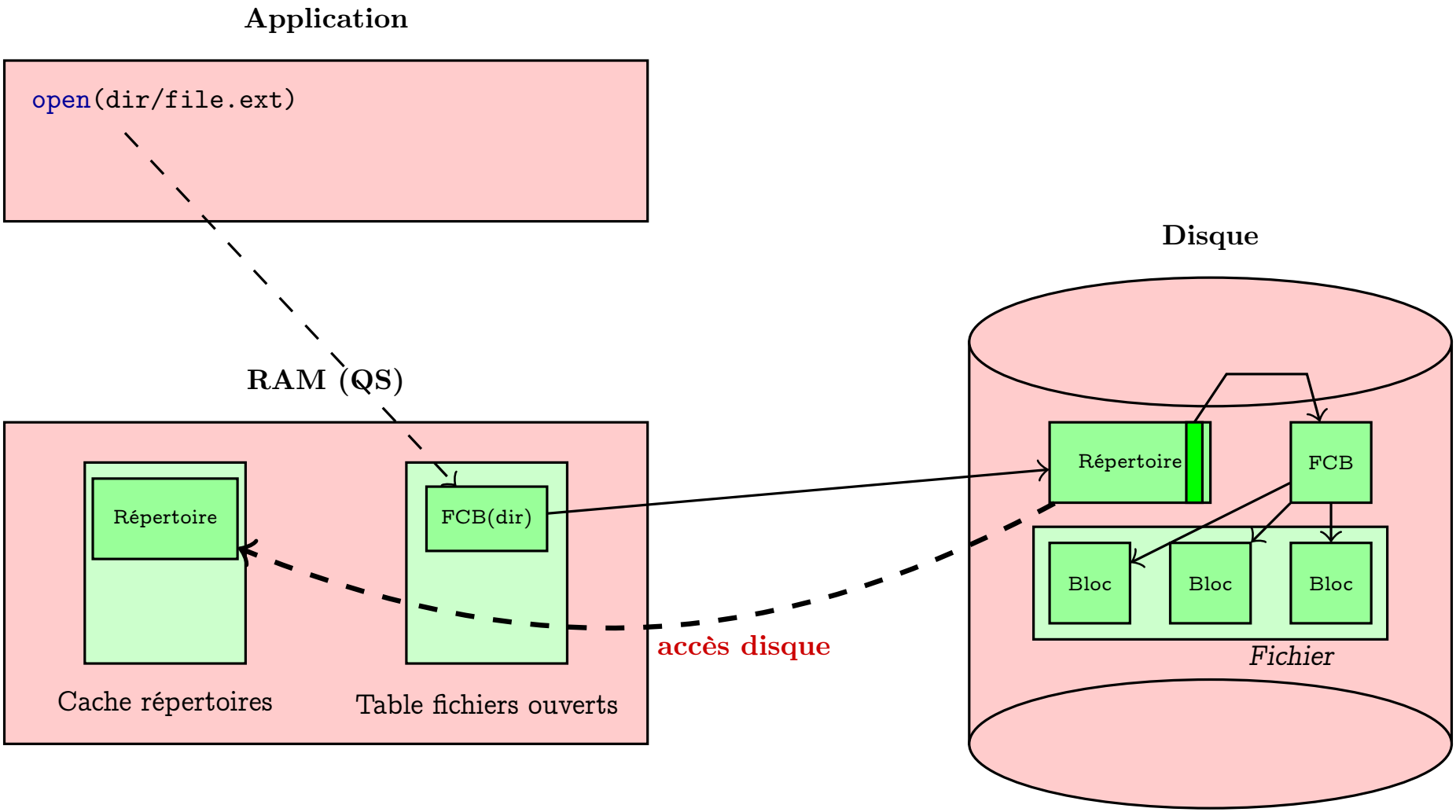
Accès à un fichier

Lors du premier accès à un fichier...



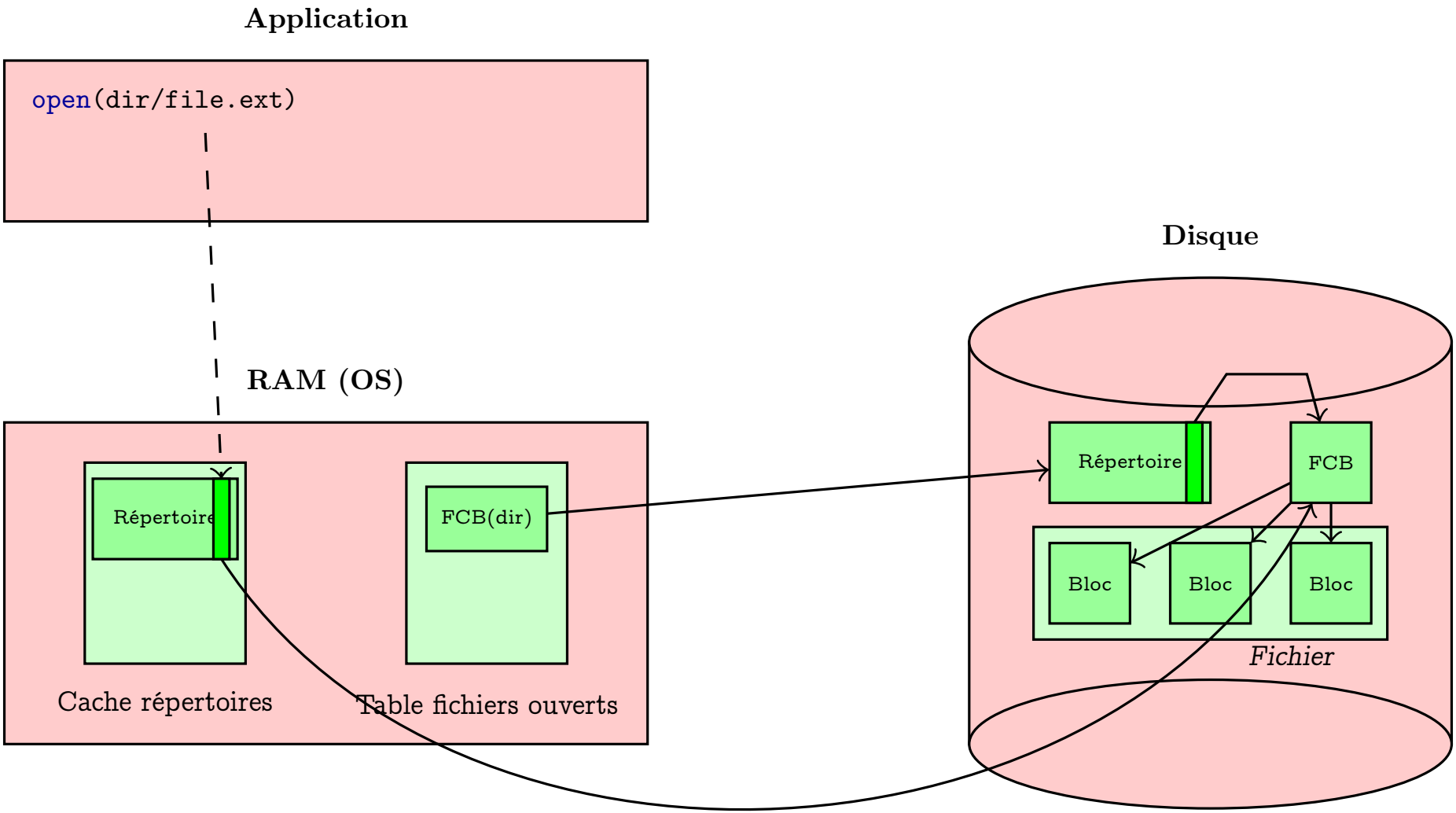
Accès à un fichier

Lors du premier accès à un fichier...



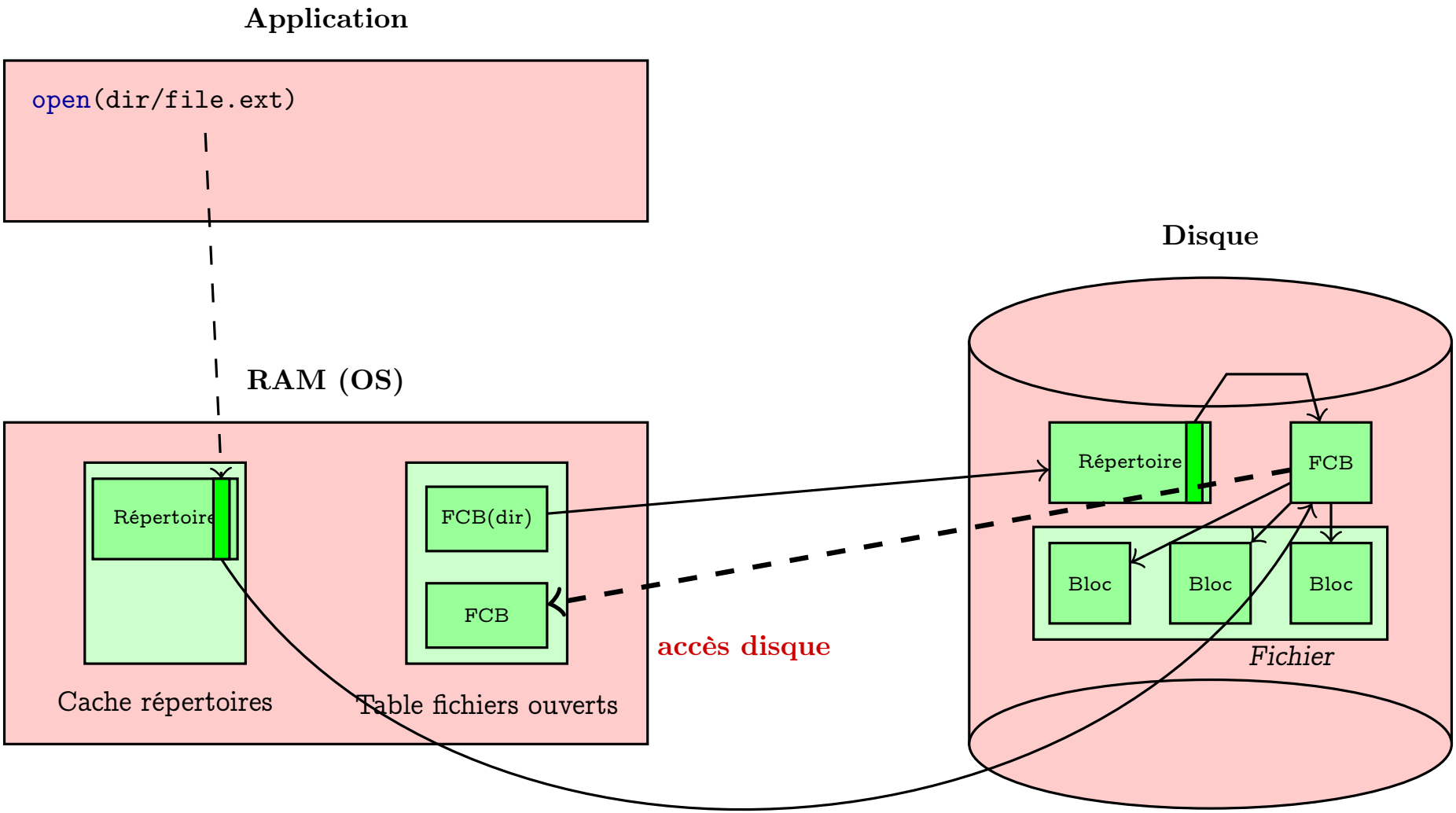
Accès à un fichier

Lors du premier accès à un fichier...



Accès à un fichier

Lors du premier accès à un fichier...

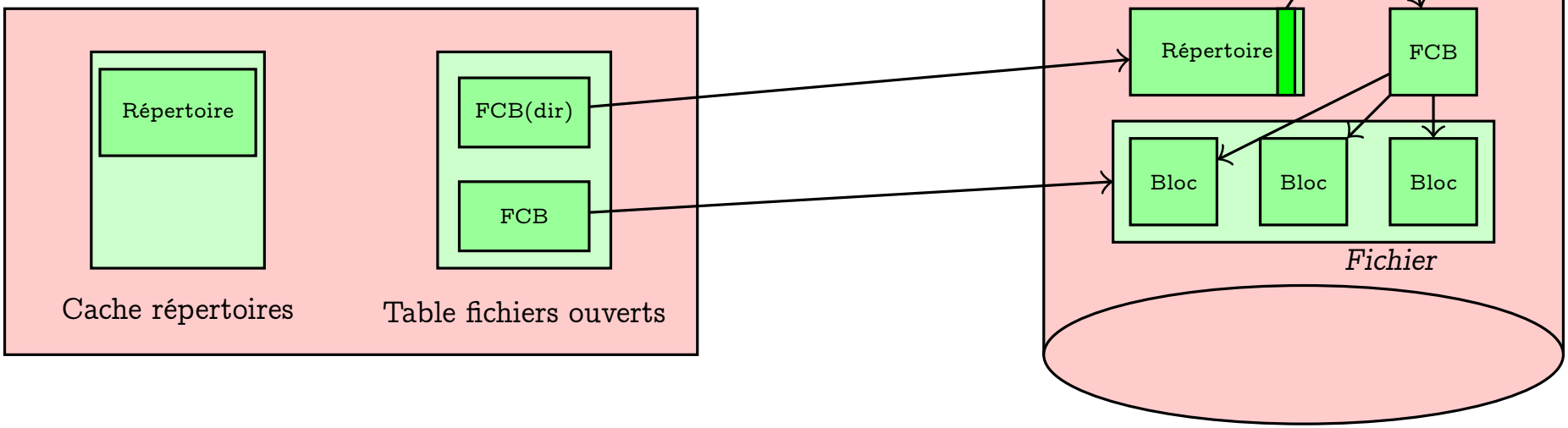


Accès à un fichier

Lors du premier accès à un fichier...

Application

```
open(dir/file.ext)
```



Accès à un fichier

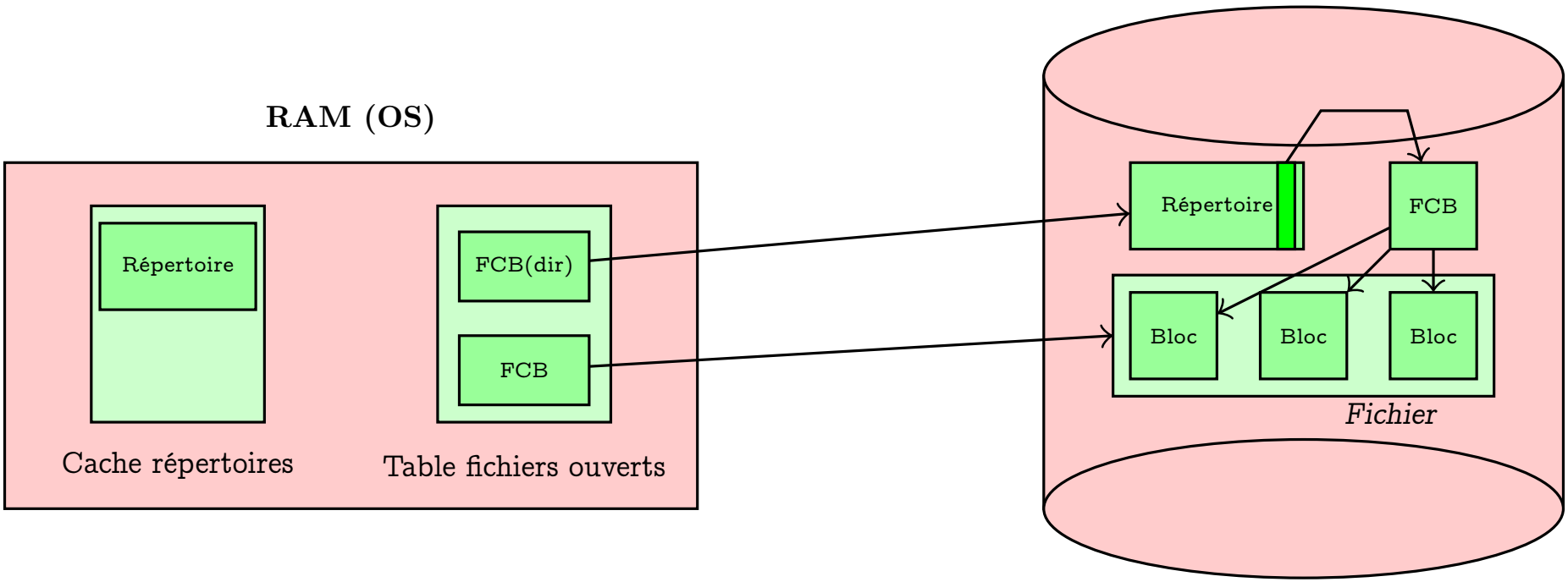
Accès au fichier ouvert...

Application

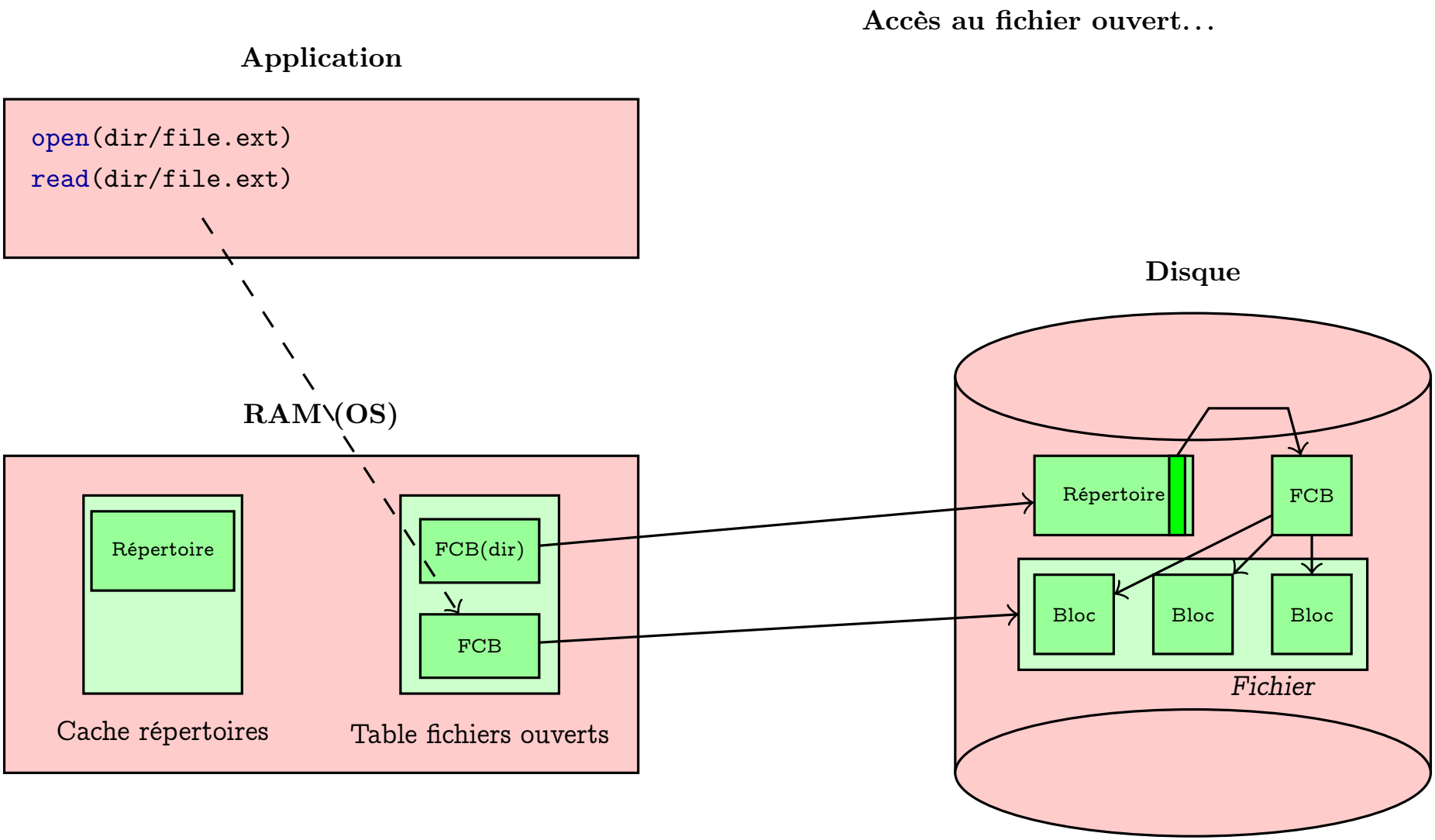
```
open(dir/file.ext)
read(dir/file.ext)
```

Disque

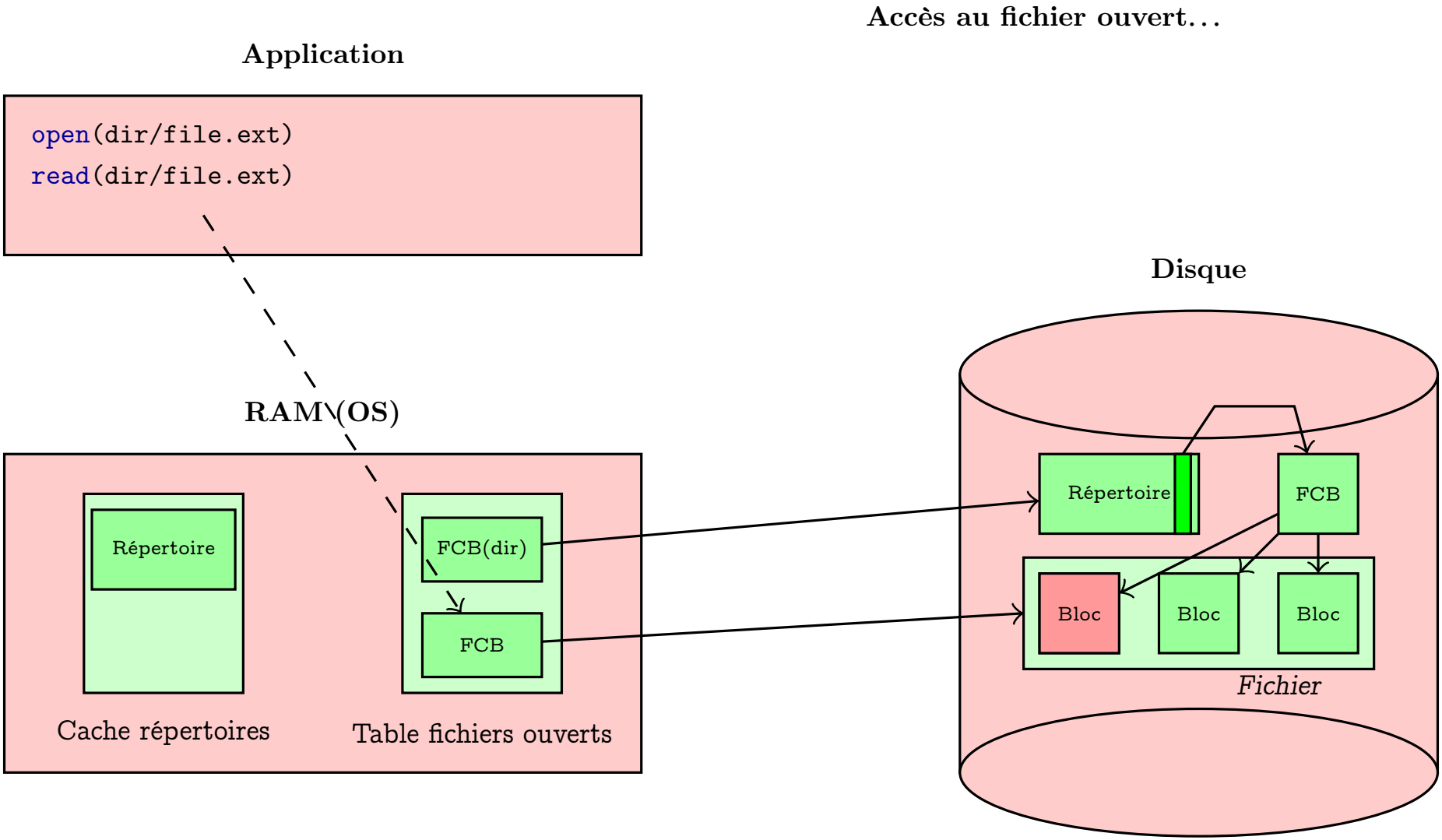
RAM (OS)



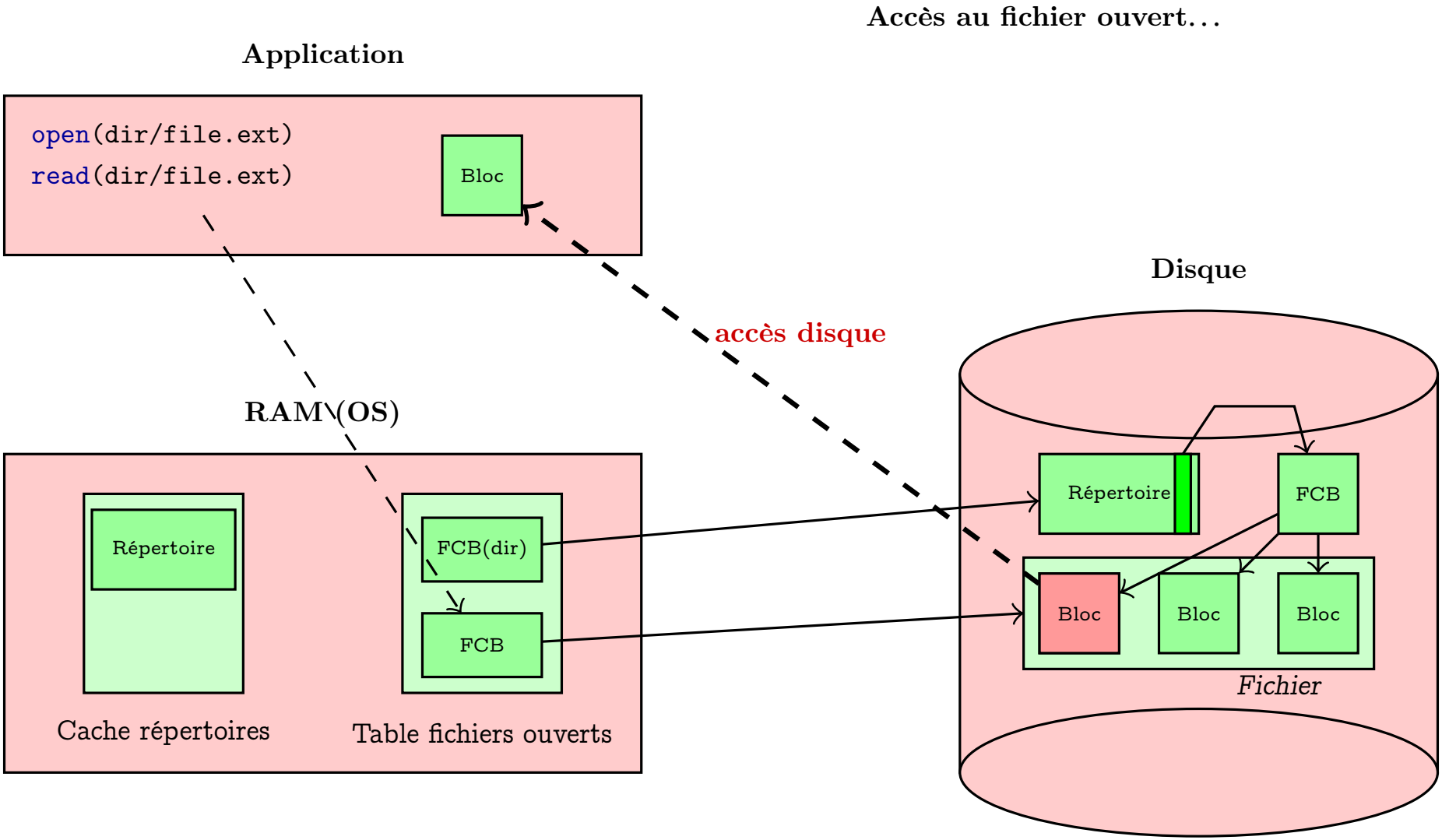
Accès à un fichier



Accès à un fichier



Accès à un fichier

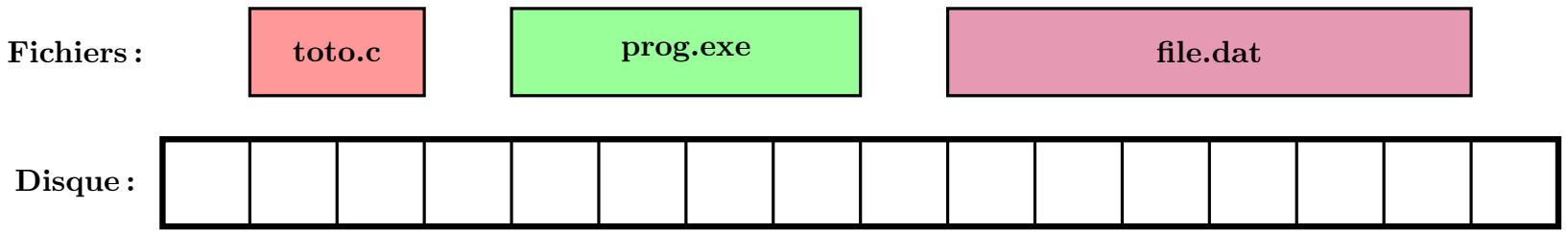


Allocation

Problème

Fichiers $\longrightarrow$ blocs logiques $\longrightarrow$ blocs physiques

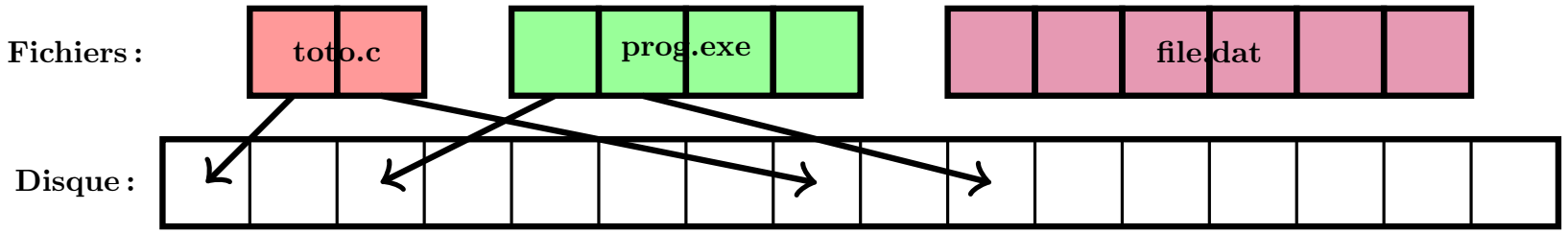
- Choix des blocs physiques pour 1 fichier donné



Problème

Fichiers $\longrightarrow$ blocs logiques $\longrightarrow$ blocs physiques

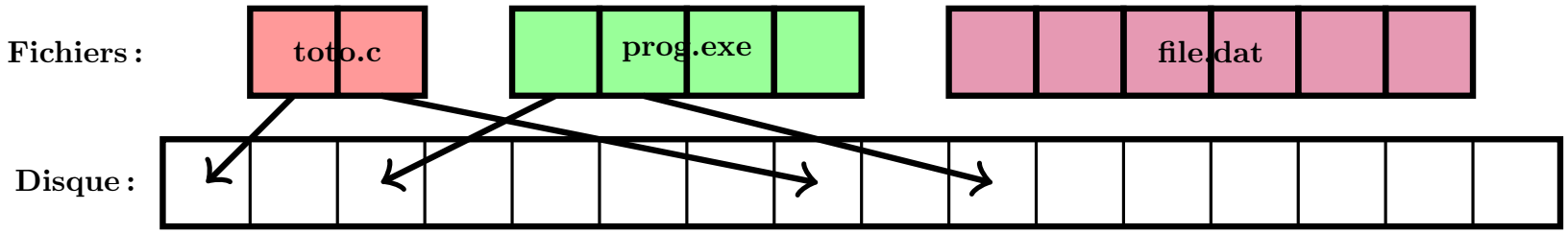
- Choix des blocs physiques pour 1 fichier donné



Problème

Fichiers $\longrightarrow$ blocs logiques $\longrightarrow$ blocs physiques

- Choix des blocs physiques pour 1 fichier donné



3 principales méthodes possibles

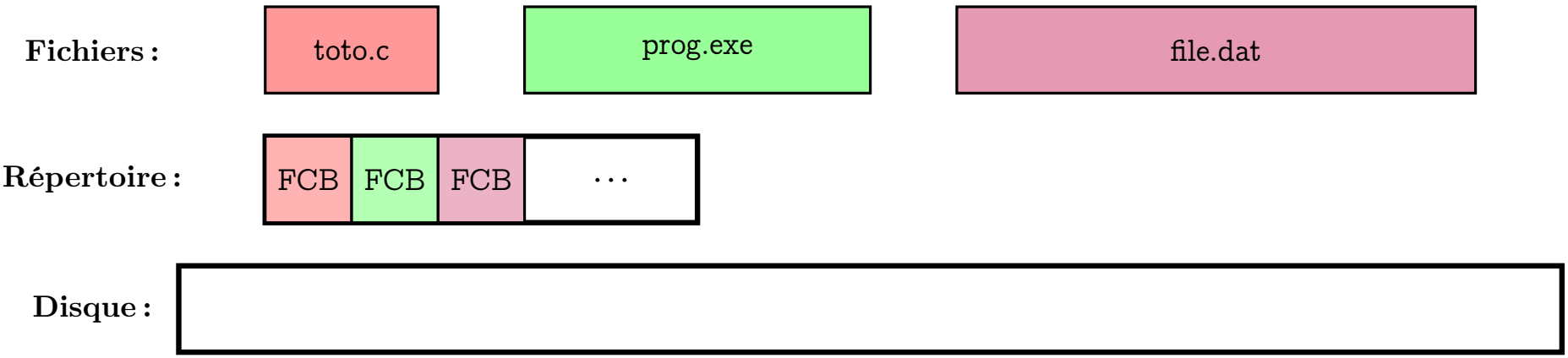
- Allocation contiguë
- Allocation chaînée
- Allocation indexée

Chaque méthode a ses avantages et inconvénients !

Allocation contiguë

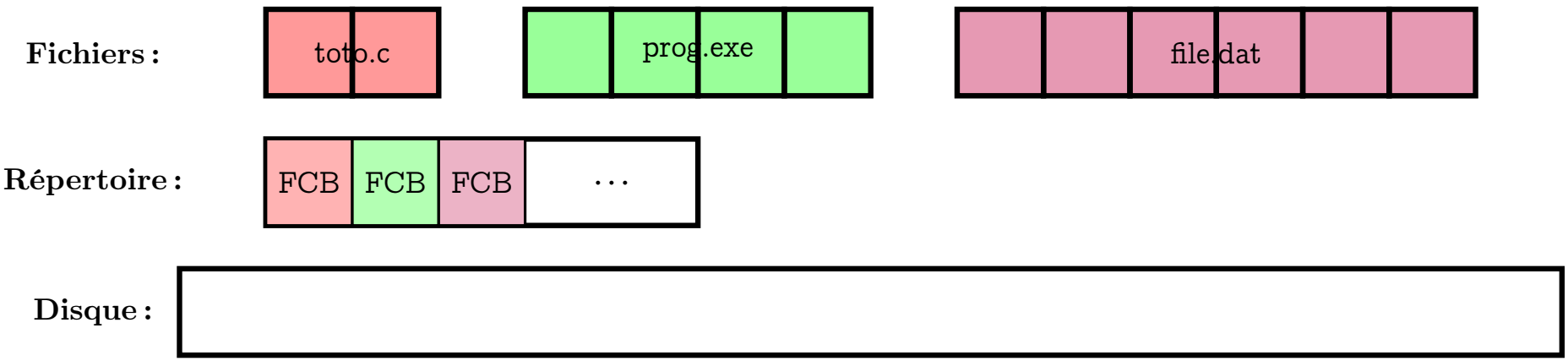
Principe

Ranger les blocs les uns derrière les autres



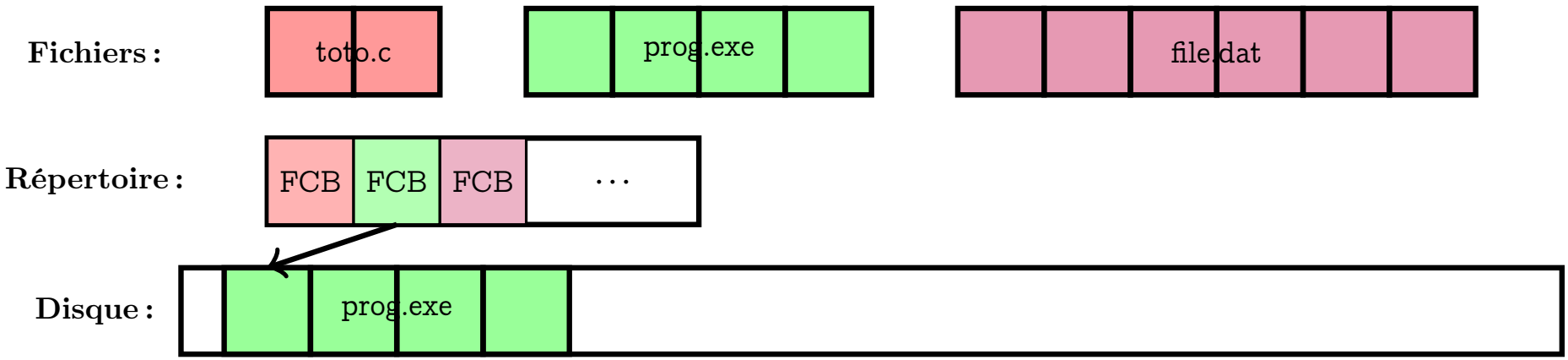
Principe

Ranger les blocs les uns derrière les autres



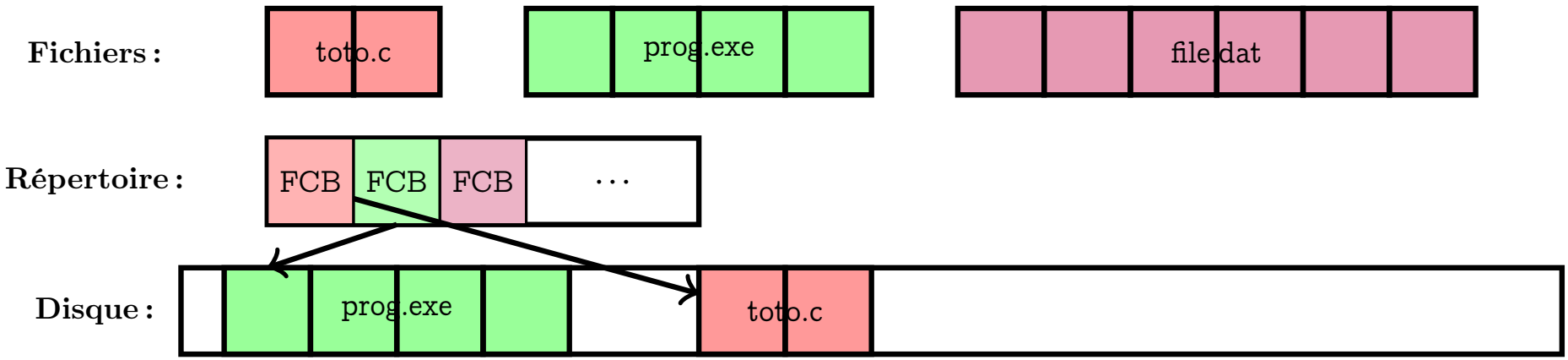
Principe

Ranger les blocs les uns derrière les autres



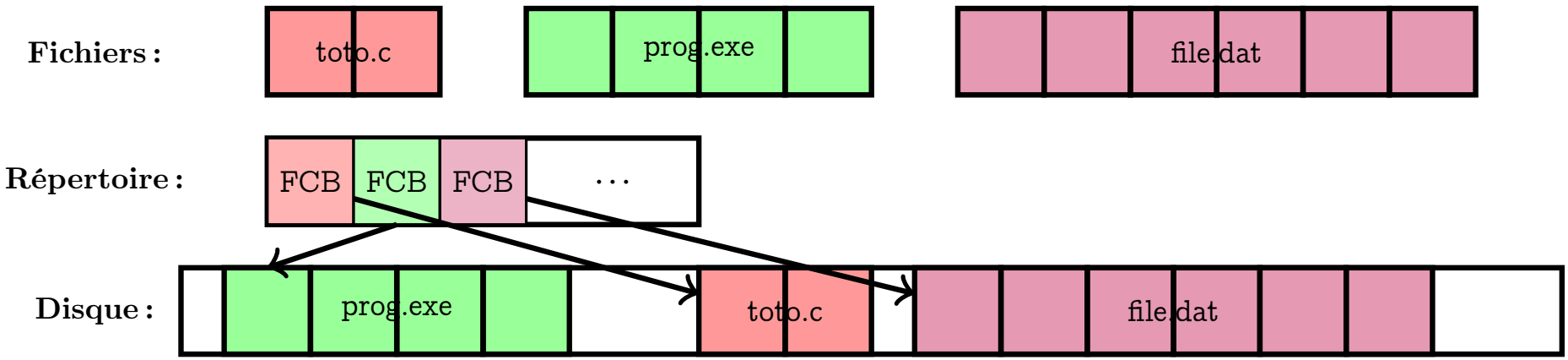
Principe

Ranger les blocs les uns derrière les autres



Principe

Ranger les blocs les uns derrière les autres



Allocation contiguë

Principe

Ranger les blocs les uns derrière les autres

Avantages

- ✓ Accès au bloc suivant : aucun coût
- ✓ FCB : adresse bloc départ + taille

Allocation contiguë

Principe

Ranger les blocs les uns derrière les autres

Avantages

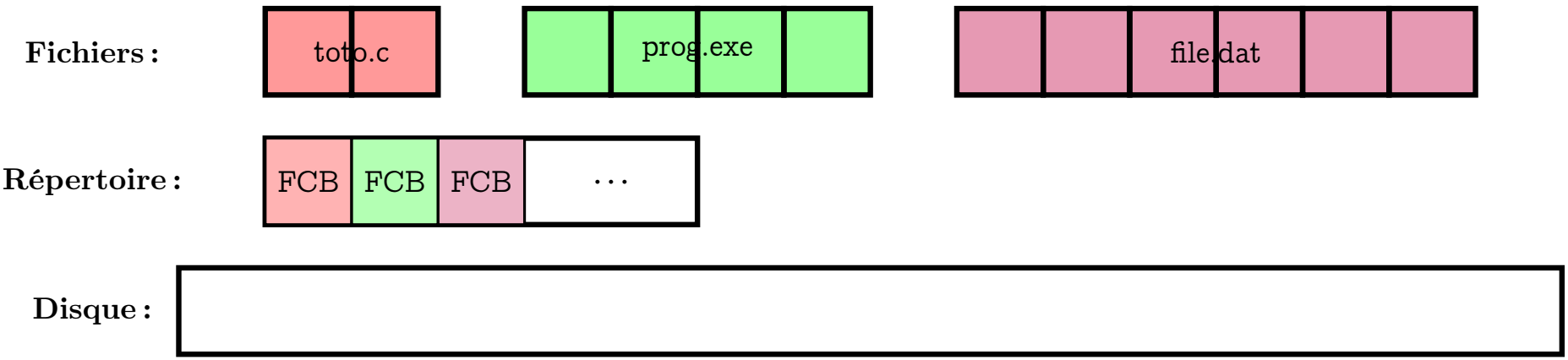
- ✓ Accès au bloc suivant : aucun coût
- ✓ FCB : adresse bloc départ + taille

Inconvénients

- X Fragmentation (compactage coûteux)
- X Connaître à l'avance la taille des fichiers
- X Recherche d'espace libre coûteux
- X Stratégies d'allocation à définir

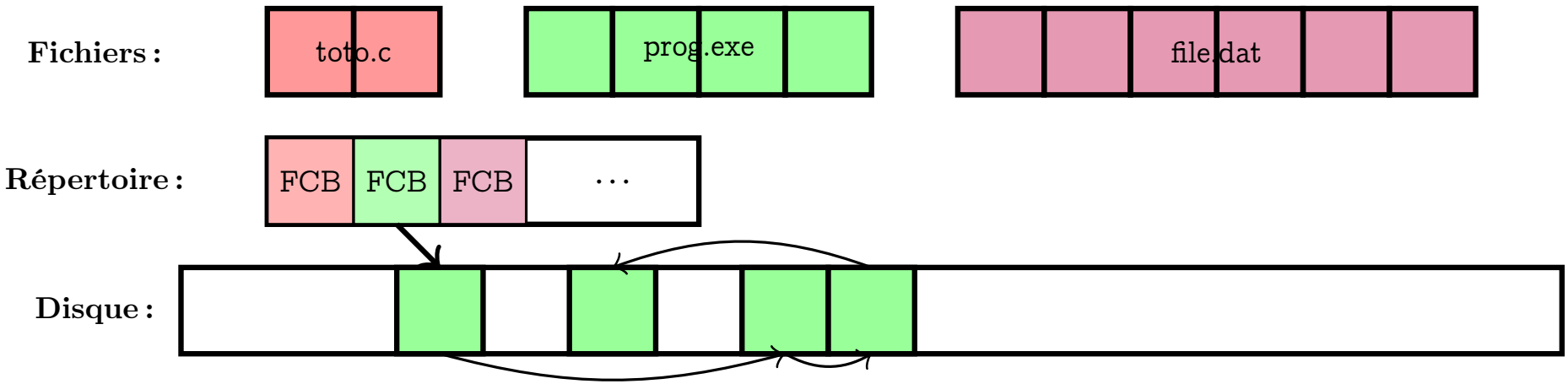
Principe

Fichier = liste chaînée de blocs



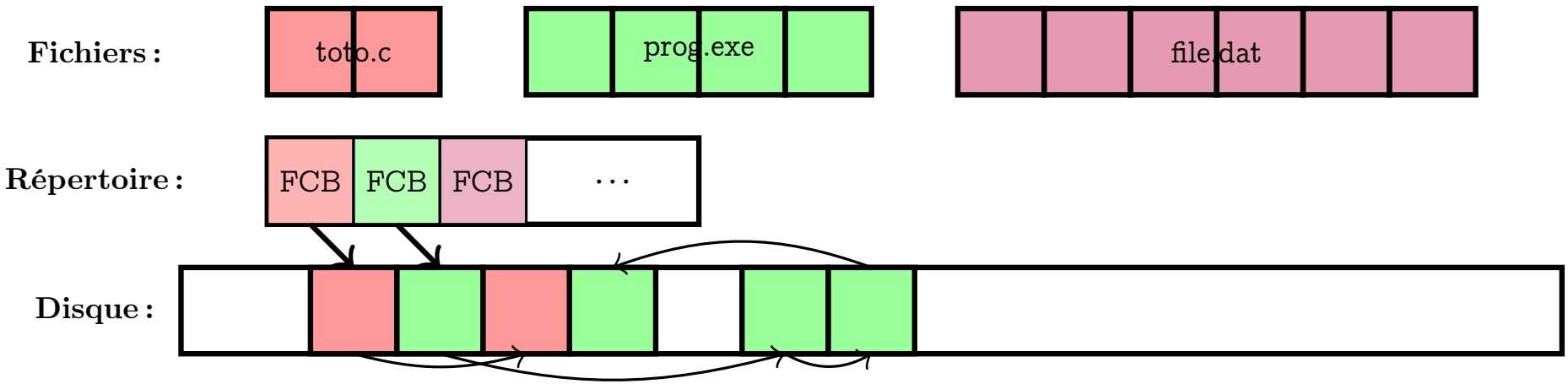
Principe

Fichier = liste chaînée de blocs



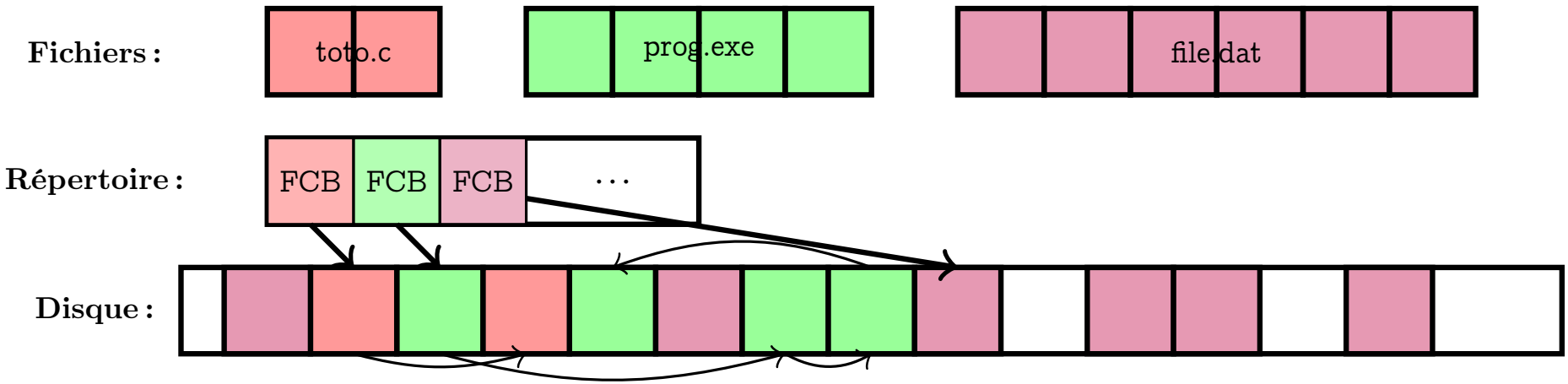
Principe

Fichier = liste chaînée de blocs



Principe

Fichier = liste chaînée de blocs



Allocation chaînée

Principe

Fichier = liste chaînée de blocs

Avantages

- ✓ FCB : adresse premier et dernier blocs
- ✓ Pas de fragmentation
- ✓ Fichiers taille quelconque

Allocation chaînée

Principe

Fichier = liste chaînée de blocs

Avantages

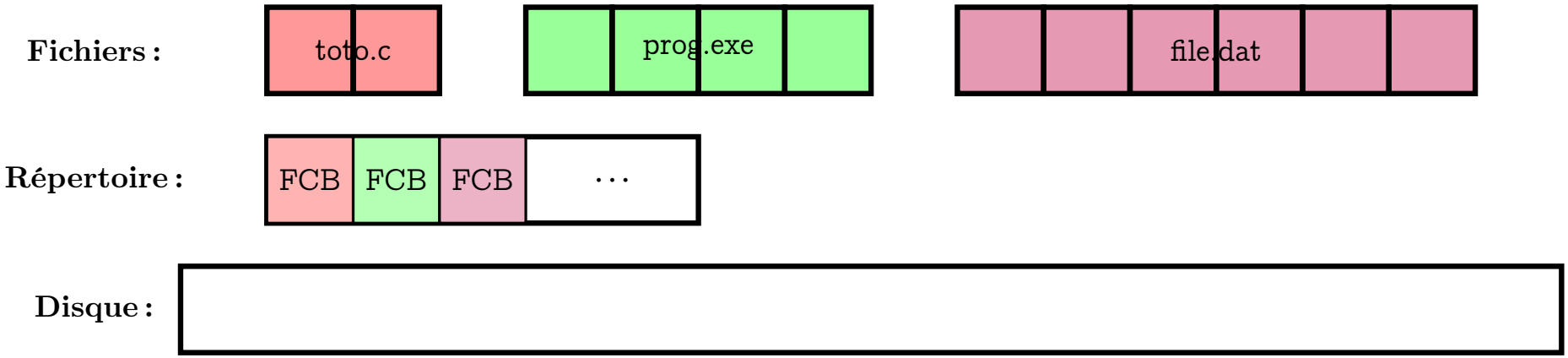
- ✓ FCB : adresse premier et dernier blocs
- ✓ Pas de fragmentation
- ✓ Fichiers taille quelconque

Inconvénients

- ✗ Accès séquentiel : N^e bloc $\rightarrow$ N accès disques !
- ✗ Fiabilité : 1 bloc endommagé $\rightarrow$ fichier perdu
- ✗ Espace utilisé par les pointeurs

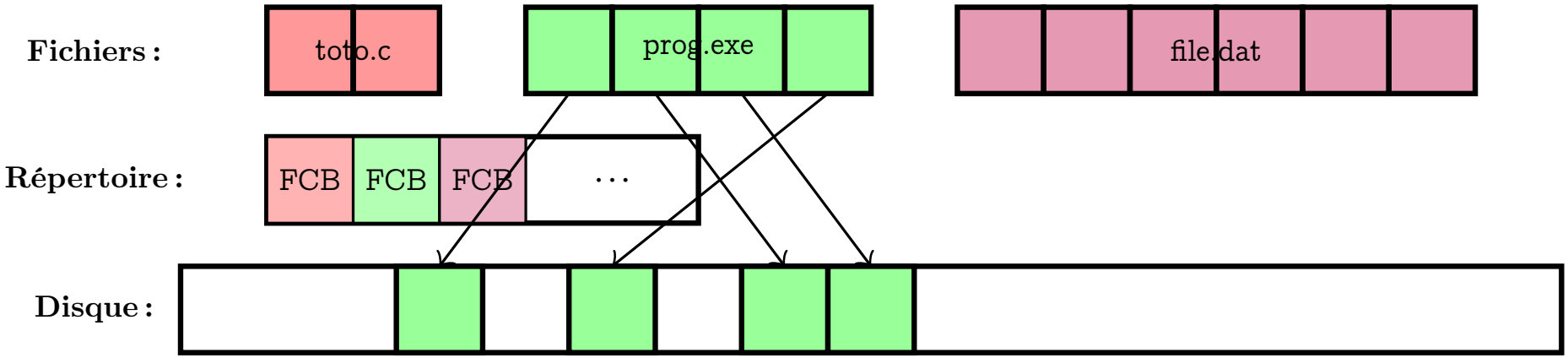
Principe

Rassembler tous les pointeurs dans un bloc d'index



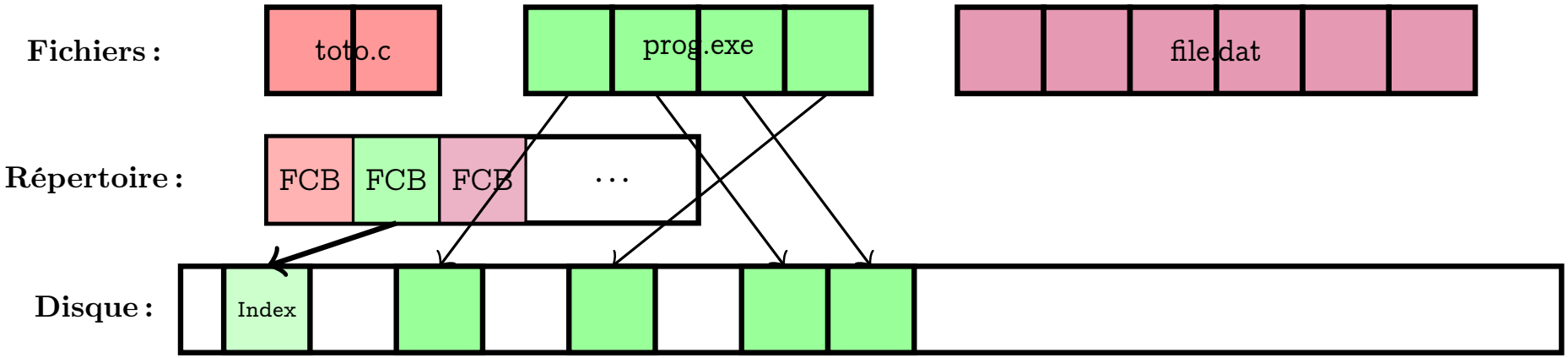
Principe

Rassembler tous les pointeurs dans un bloc d'index



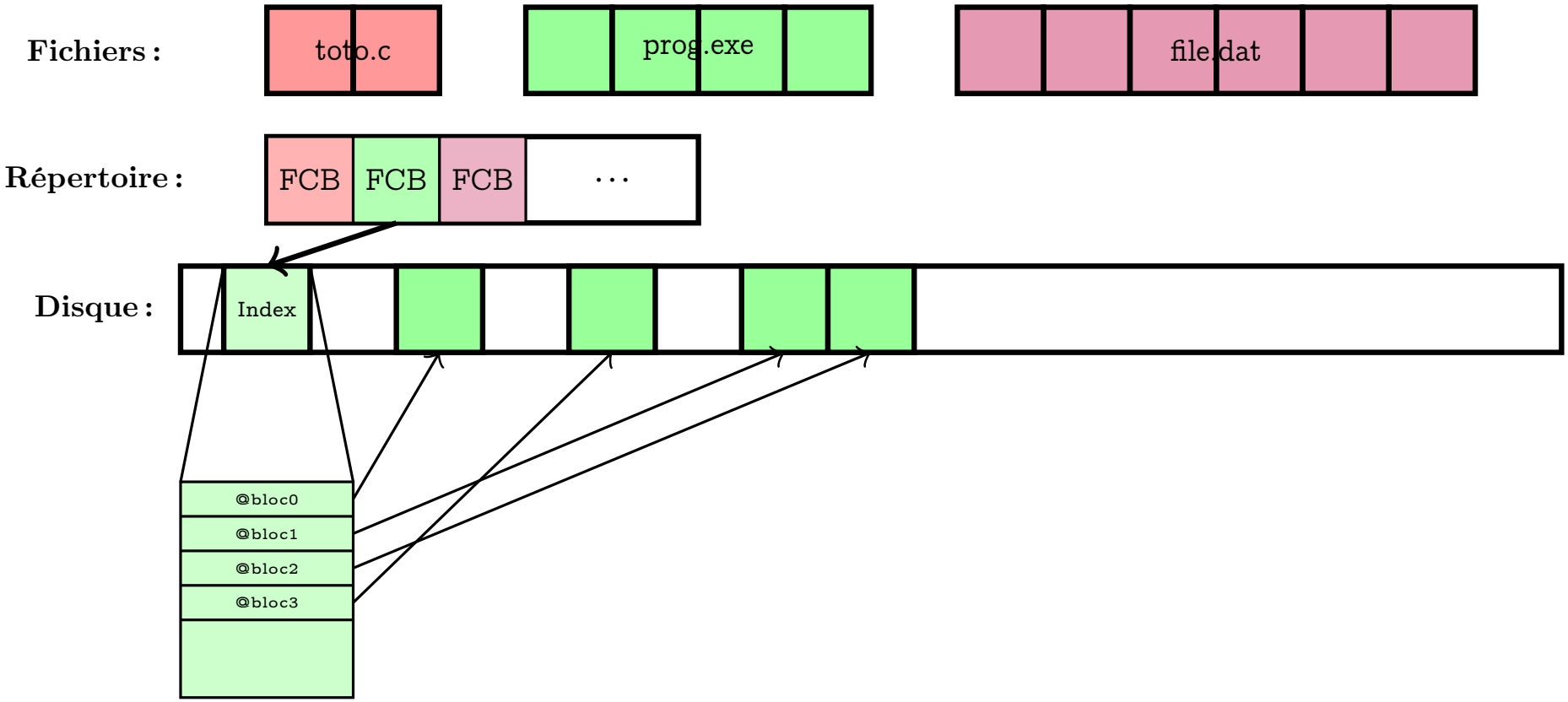
Principe

Rassembler tous les pointeurs dans un bloc d'index



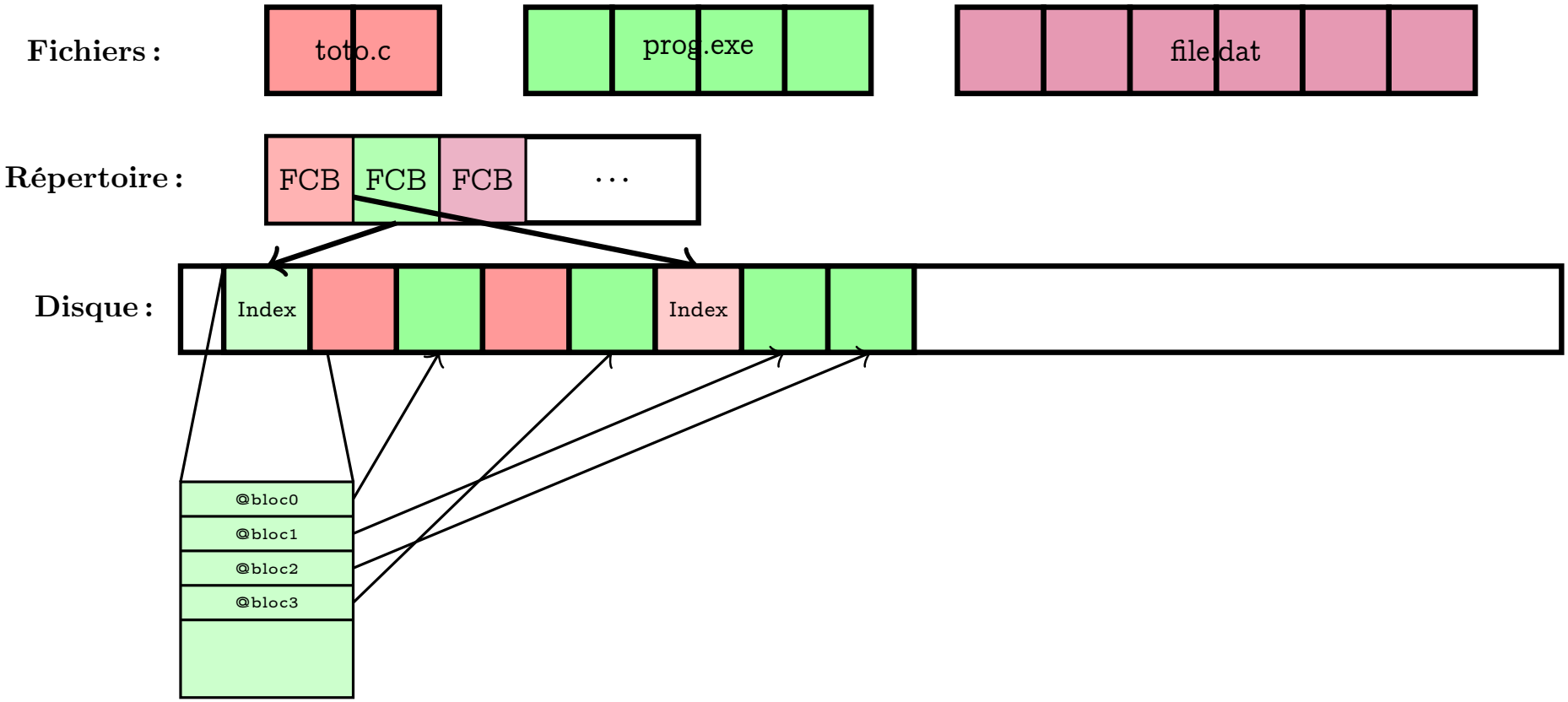
Principe

Rassembler tous les pointeurs dans un bloc d'index



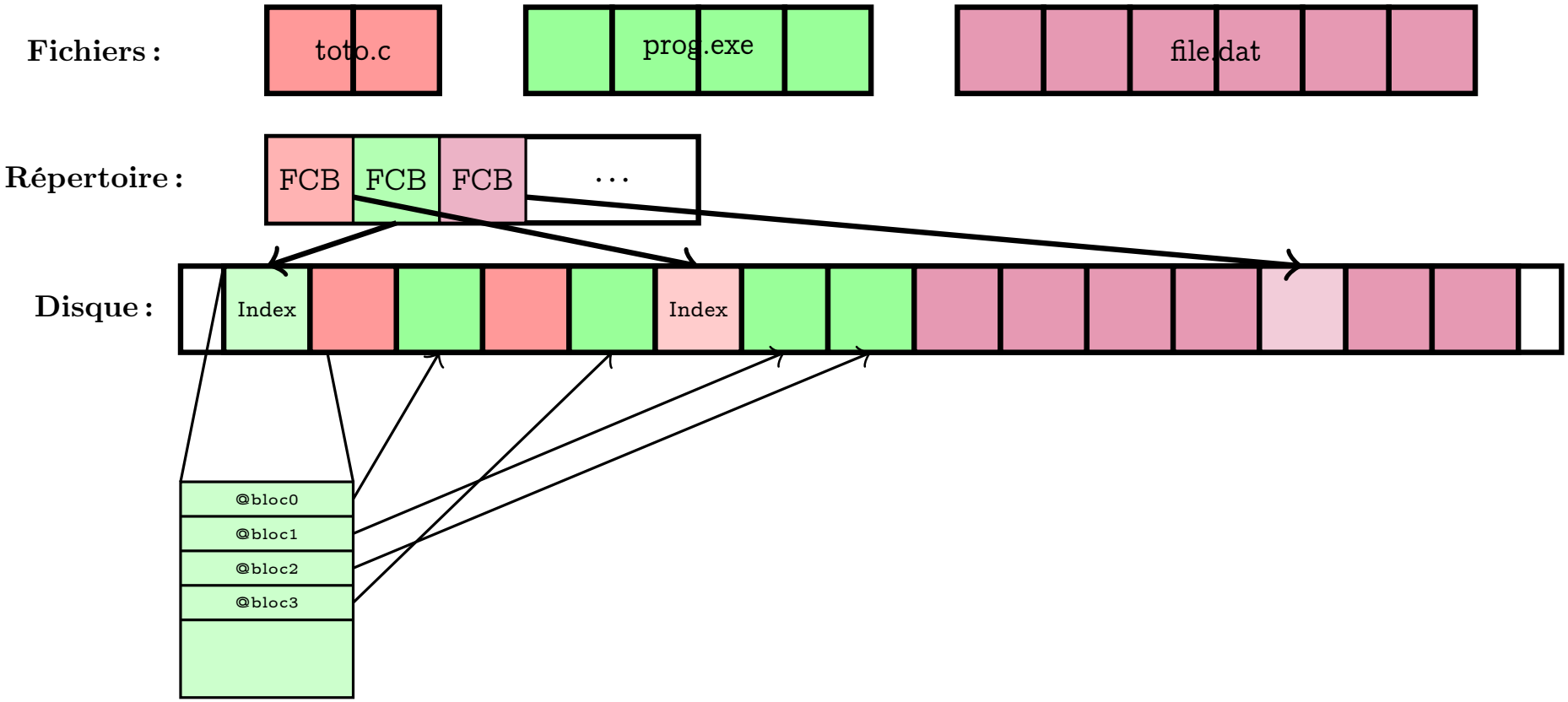
Principe

Rassembler tous les pointeurs dans un bloc d'index



Principe

Rassembler tous les pointeurs dans un bloc d'index



Allocation indexée

Principe

Rassembler tous les pointeurs dans un bloc d'index

Avantages

- ✓ Pas de fragmentation
- ✓ Accès direct (2 accès disque)

Allocation indexée

Principe

Rassembler tous les pointeurs dans un bloc d'index

Avantages

- ✓ Pas de fragmentation
- ✓ Accès direct (2 accès disque)

Inconvénients

- X Taille fichier limitée par taille bloc

$$64\ bits \times 64\ blocs = 512o \quad \Rightarrow max = 64 \times 512o = 32\ Ko$$

Allocation indexée

Problème

Fichiers nécessitant plus d'un bloc d'index

Exemple :

- 512ko de disque, blocs de 8o (exemple non réaliste !)
- Taille de l'adresse = ?

Problème

Fichiers nécessitant plus d'un bloc d'index

Exemple :

- 512ko de disque, blocs de 8o (exemple non réaliste !)
- Taille de l'adresse $= 2^{19} \div 2^3 = 2^{16} =$ 16 bits
- Nombre max d'index/bloc = ?

Problème

Fichiers nécessitant plus d'un bloc d'index

Exemple :

- 512ko de disque, blocs de 8o (exemple non réaliste !)
- Taille de l'adresse $= 2^{19} \div 2^3 = 2^{16} =$ 16 bits
- Nombre max d'index/bloc $= 8 \div 2 =$ 4 blocs
- Taille max d'un fichier $=$?

Problème

Fichiers nécessitant plus d'un bloc d'index

Exemple :

- 512ko de disque, blocs de 8o (exemple non réaliste !)
- Taille de l'adresse $= 2^{19} \div 2^3 = 2^{16} =$ 16 bits
- Nombre max d'index/bloc $= 8 \div 2 =$ 4 blocs
- Taille max d'un fichier $=$ 4 blocs $=$ 32o

Problème

Fichiers nécessitant plus d'un bloc d'index

Exemple :

- 512ko de disque, blocs de 8o (exemple non réaliste !)
- Taille de l'adresse $= 2^{19} \div 2^3 = 2^{16} =$ 16 bits
- Nombre max d'index/bloc $= 8 \div 2 =$ 4 blocs
- Taille max d'un fichier $=$ 4 blocs $=$ 32o

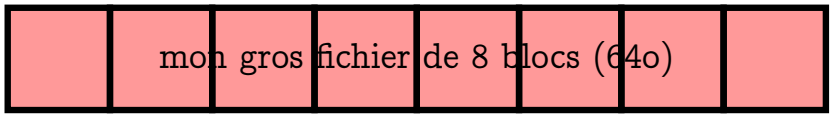
Un peu petit...

Solution 1 : liste chaînée de blocs d'index

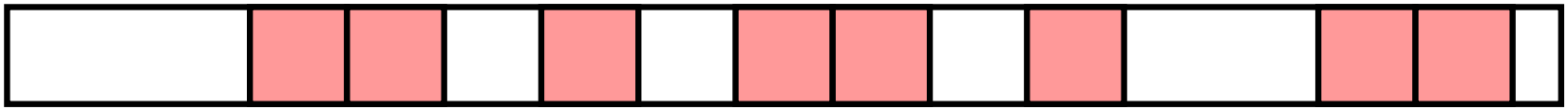
Principe

- Le répertoire pointe vers un bloc d'index
- Chaque bloc d'index se termine par un pointeur vers un autre bloc d'index

Fichier :



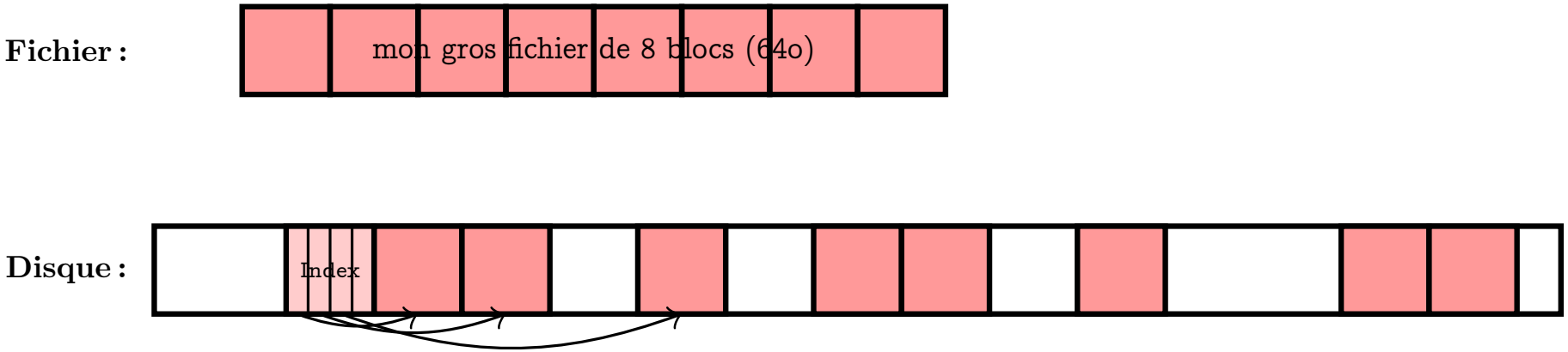
Disque :



Solution 1 : liste chaînée de blocs d'index

Principe

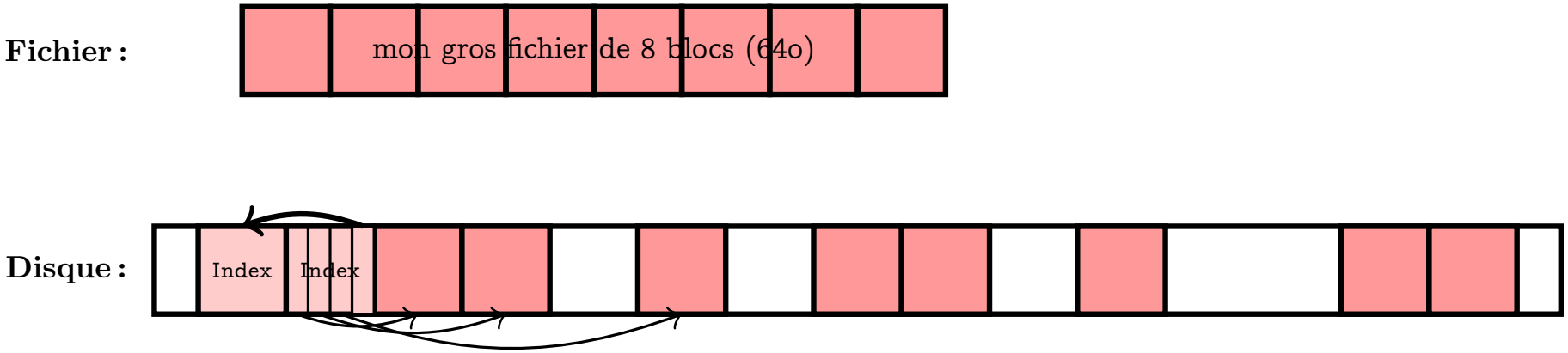
- Le répertoire pointe vers un bloc d'index
- Chaque bloc d'index se termine par un pointeur vers un autre bloc d'index



Solution 1 : liste chaînée de blocs d'index

Principe

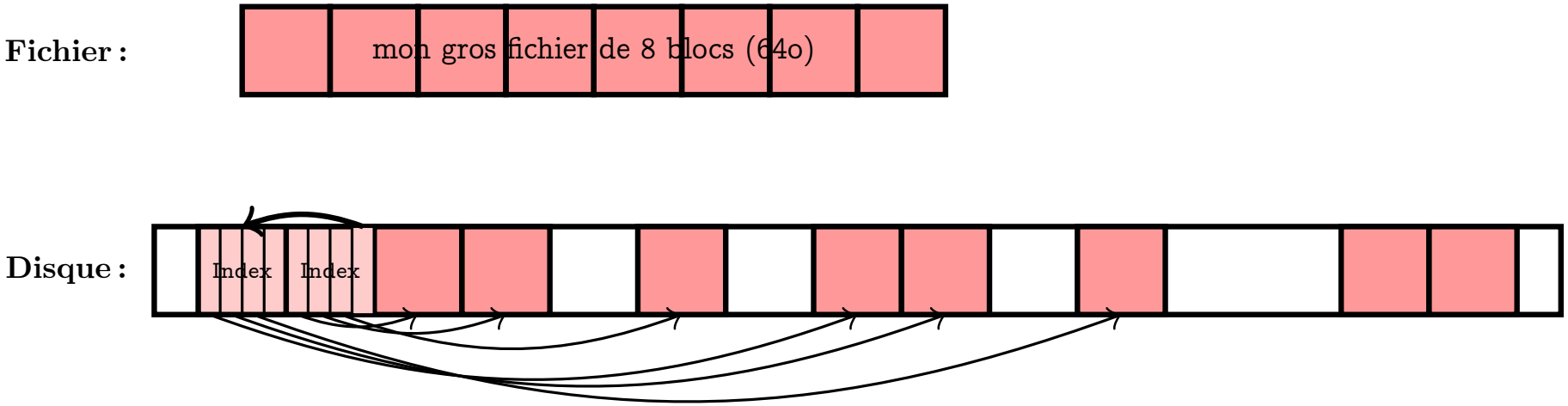
- Le répertoire pointe vers un bloc d'index
- Chaque bloc d'index se termine par un pointeur vers un autre bloc d'index



Solution 1 : liste chaînée de blocs d'index

Principe

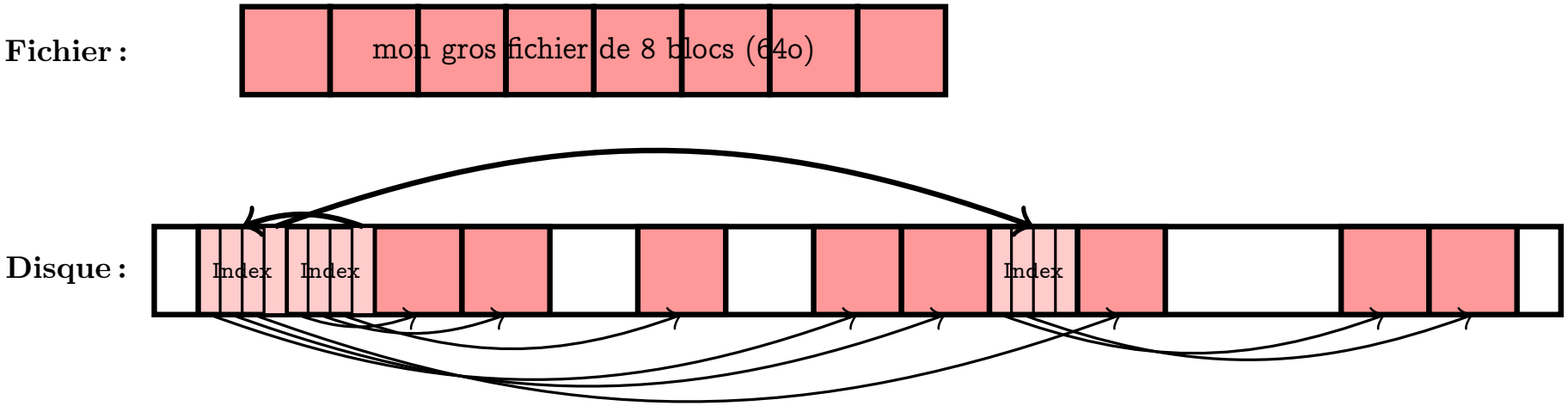
- Le répertoire pointe vers un bloc d'index
- Chaque bloc d'index se termine par un pointeur vers un autre bloc d'index



Solution 1 : liste chaînée de blocs d'index

Principe

- Le répertoire pointe vers un bloc d'index
- Chaque bloc d'index se termine par un pointeur vers un autre bloc d'index



Solution 1 : liste chaînée de blocs d'index

Principe

- Le répertoire pointe vers un bloc d'index
- Chaque bloc d'index se termine par un pointeur vers un autre bloc d'index

Avantages

✓ Pas de fragmentation

✓ Taille de fichier quelconque

Solution 1 : liste chaînée de blocs d'index

Principe

- Le répertoire pointe vers un bloc d'index
- Chaque bloc d'index se termine par un pointeur vers un autre bloc d'index

Avantages

✓ Pas de fragmentation

✓ Taille de fichier quelconque

Inconvénients

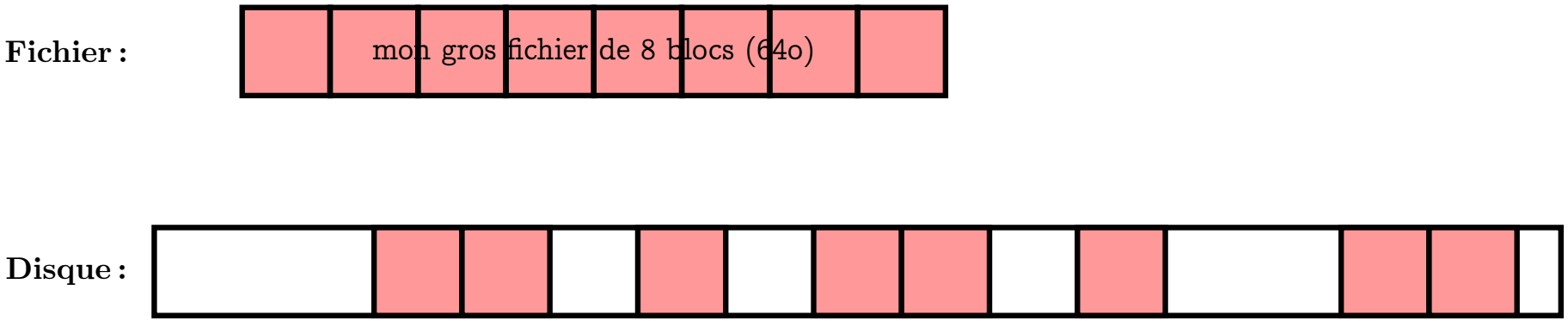
X N blocs perdus par fichier

X Accès indirect ($\geqslant$2 accès disque) mais plus rapide que la liste chaînée de blocs !

Solution 2 : indexation à plusieurs niveaux

Principe

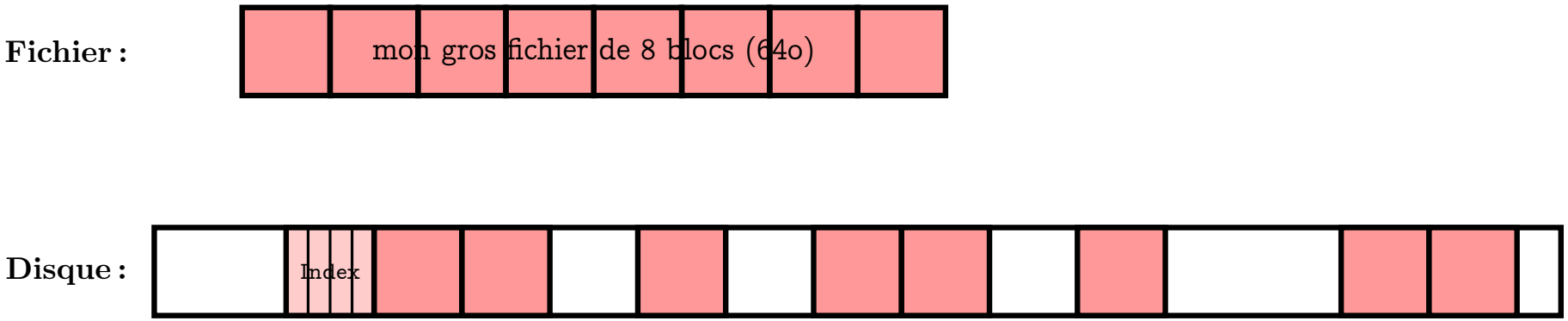
- Le répertoire pointe vers un bloc d'index *maître*
- Le bloc maître pointe vers des blocs d'index
- Éventuellement, indexation sur 3 niveaux (n^3)



Solution 2 : indexation à plusieurs niveaux

Principe

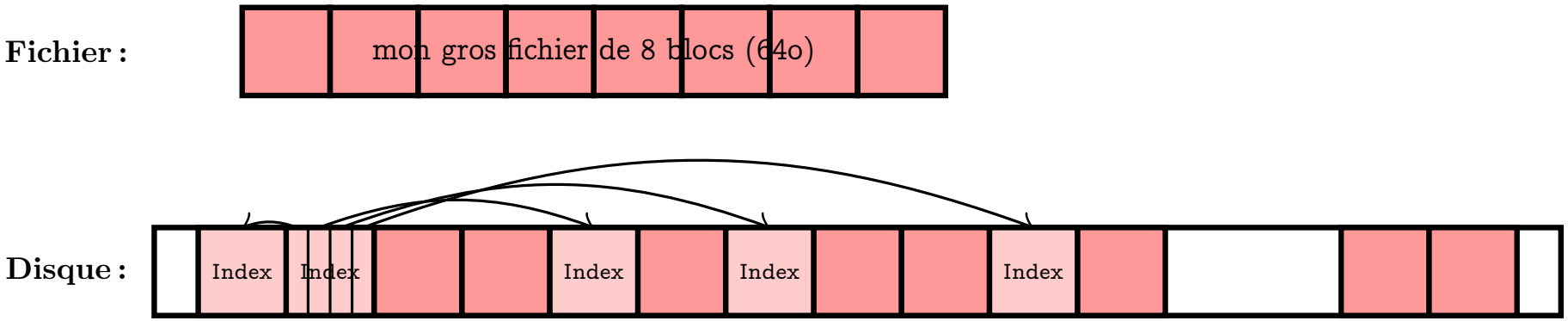
- Le répertoire pointe vers un bloc d'index *maître*
- Le bloc maître pointe vers des blocs d'index
- Éventuellement, indexation sur 3 niveaux (n^3)



Solution 2 : indexation à plusieurs niveaux

Principe

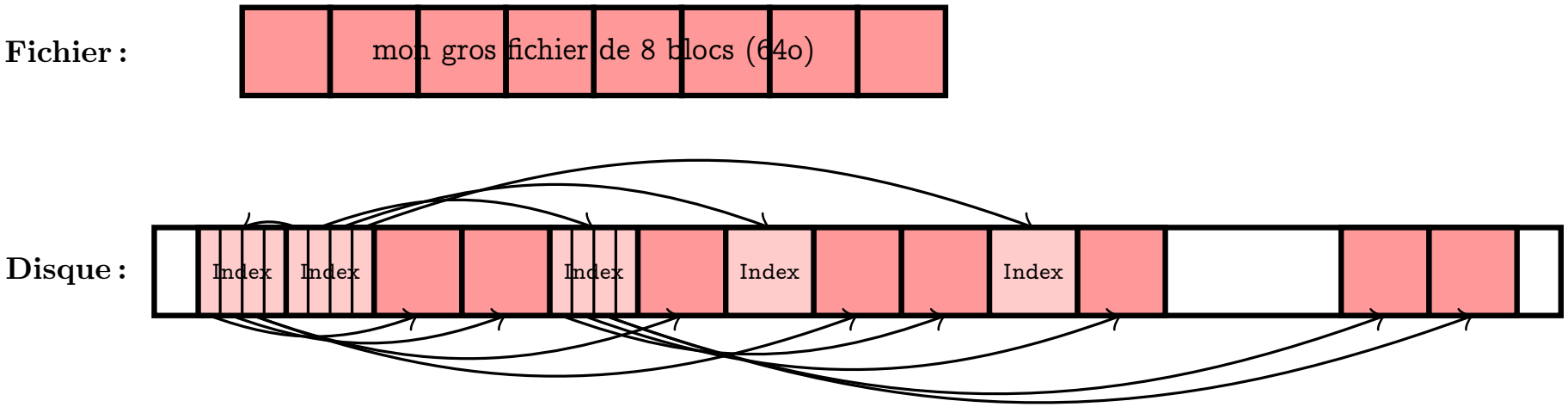
- Le répertoire pointe vers un bloc d'index *maître*
- Le bloc maître pointe vers des blocs d'index
- Éventuellement, indexation sur 3 niveaux (n^3)



Solution 2 : indexation à plusieurs niveaux

Principe

- Le répertoire pointe vers un bloc d'index *maître*
- Le bloc maître pointe vers des blocs d'index
- Éventuellement, indexation sur 3 niveaux (n^3)



Solution 2 : indexation à plusieurs niveaux

Principe

- Le répertoire pointe vers un bloc d'index *maître*
- Le bloc maître pointe vers des blocs d'index

Avantages

✓ Pas de fragmentation

✓ Taille de fichier quelconque

✓ Accès direct (3 accès disque max + possibilité index en cache)

Solution 2 : indexation à plusieurs niveaux

Principe

- Le répertoire pointe vers un bloc d'index *maître*
- Le bloc maître pointe vers des blocs d'index

Avantages

✓ Pas de fragmentation

✓ Taille de fichier quelconque

✓ Accès direct (3 accès disque max + possibilité index en cache)

Inconvénients

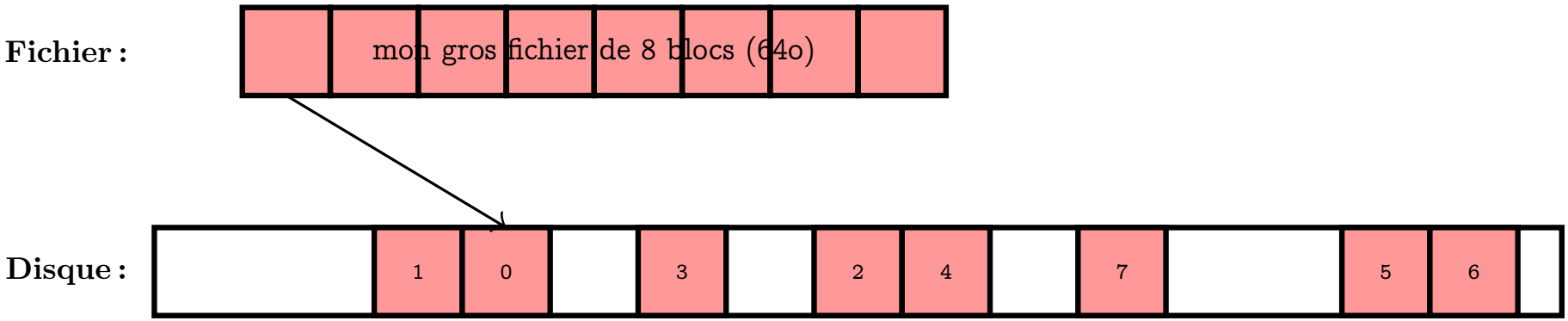
X $\geqslant$ 2 blocs perdus par fichier

(même si on ne crée pas les index en trop)

File Allocation Table (FAT)

Principe

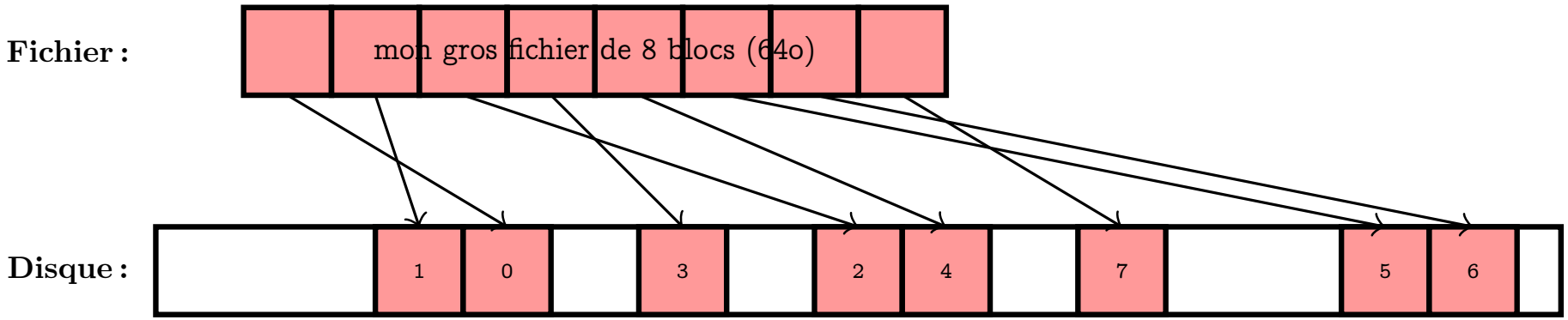
- Utilisé sous MSDOS (Intel) et OS/2 (IBM)
- Allocation indexée
- Liste chaînée des index en début de partition



File Allocation Table (FAT)

Principe

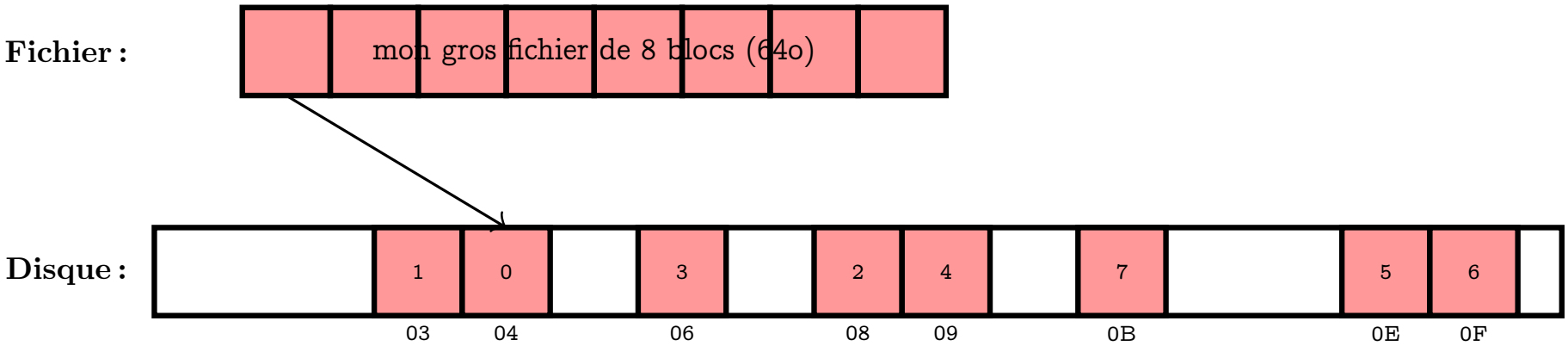
- Utilisé sous MSDOS (Intel) et OS/2 (IBM)
- Allocation indexée
- Liste chaînée des index en début de partition



File Allocation Table (FAT)

Principe

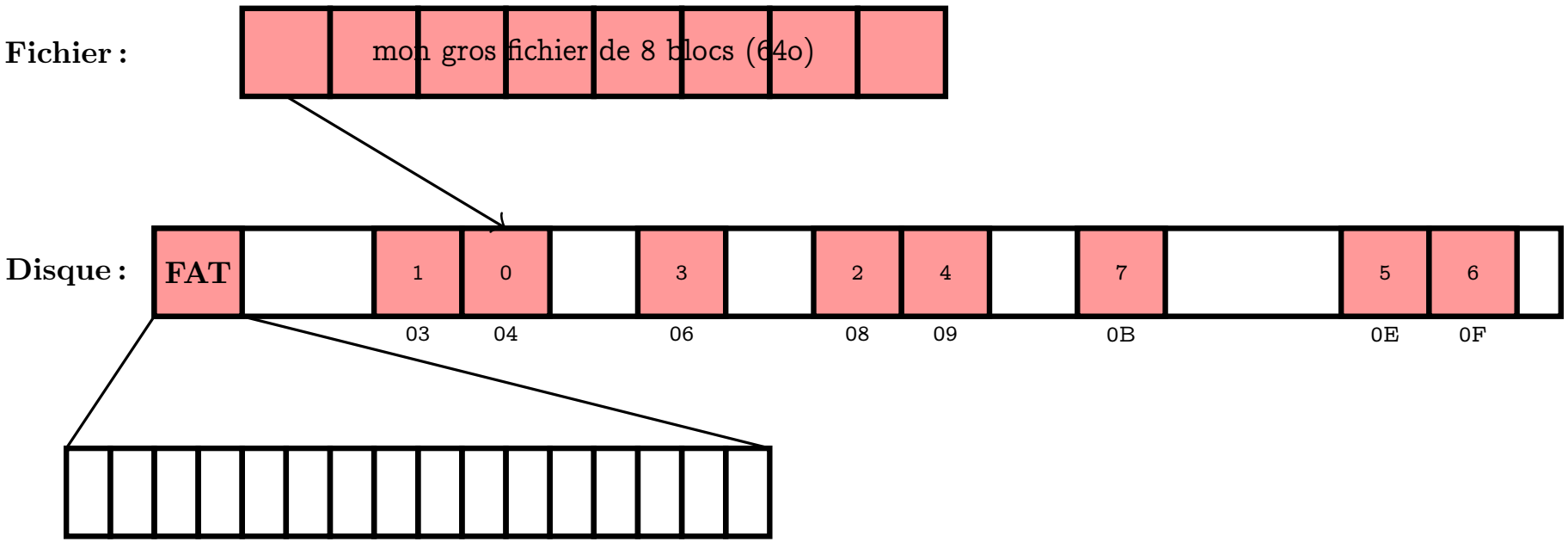
- Utilisé sous MSDOS (Intel) et OS/2 (IBM)
- Allocation indexée
- Liste chaînée des index en début de partition



File Allocation Table (FAT)

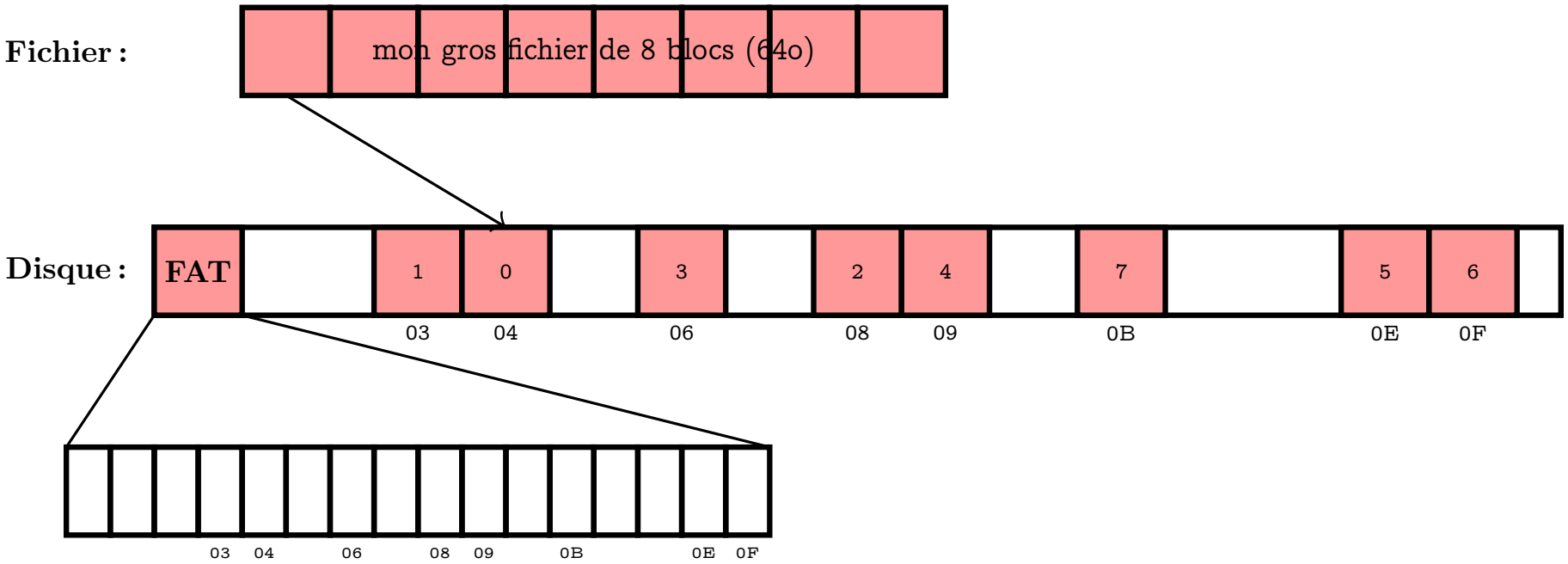
Principe

- Utilisé sous MSDOS (Intel) et OS/2 (IBM)
- Allocation indexée
- Liste chaînée des index en début de partition



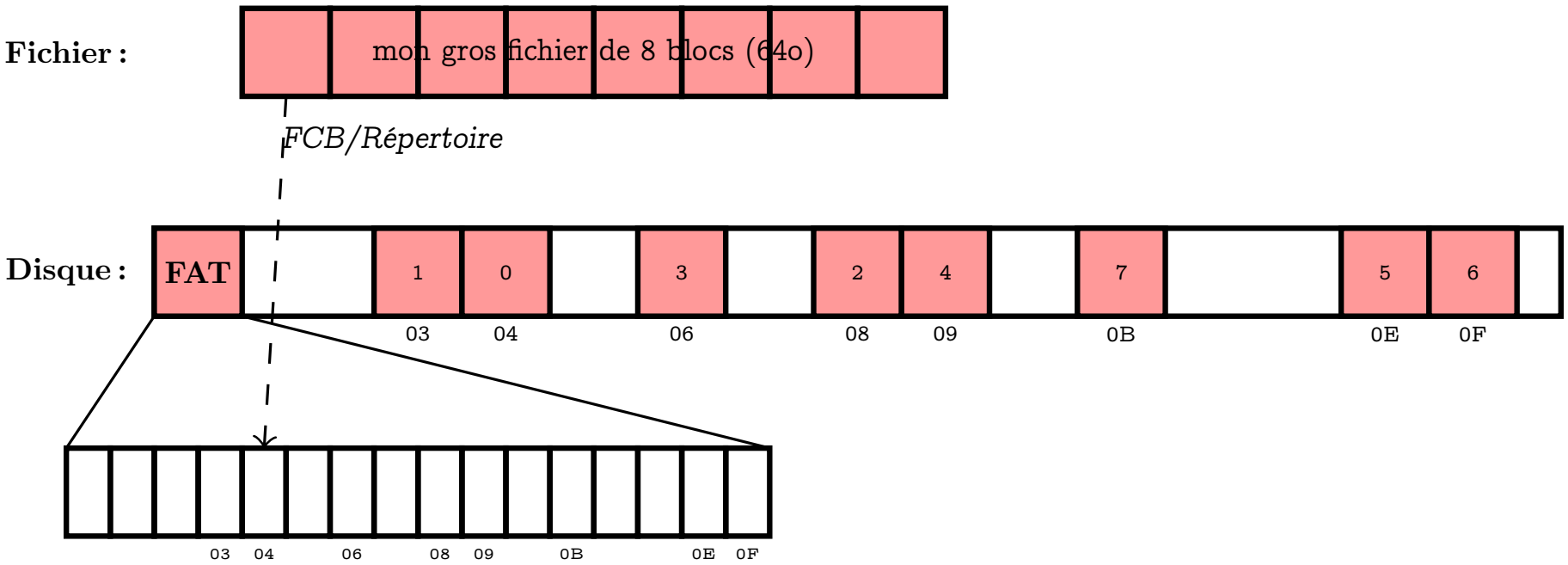
Principe

- Utilisé sous MSDOS (Intel) et OS/2 (IBM)
- Allocation indexée
- Liste chaînée des index en début de partition



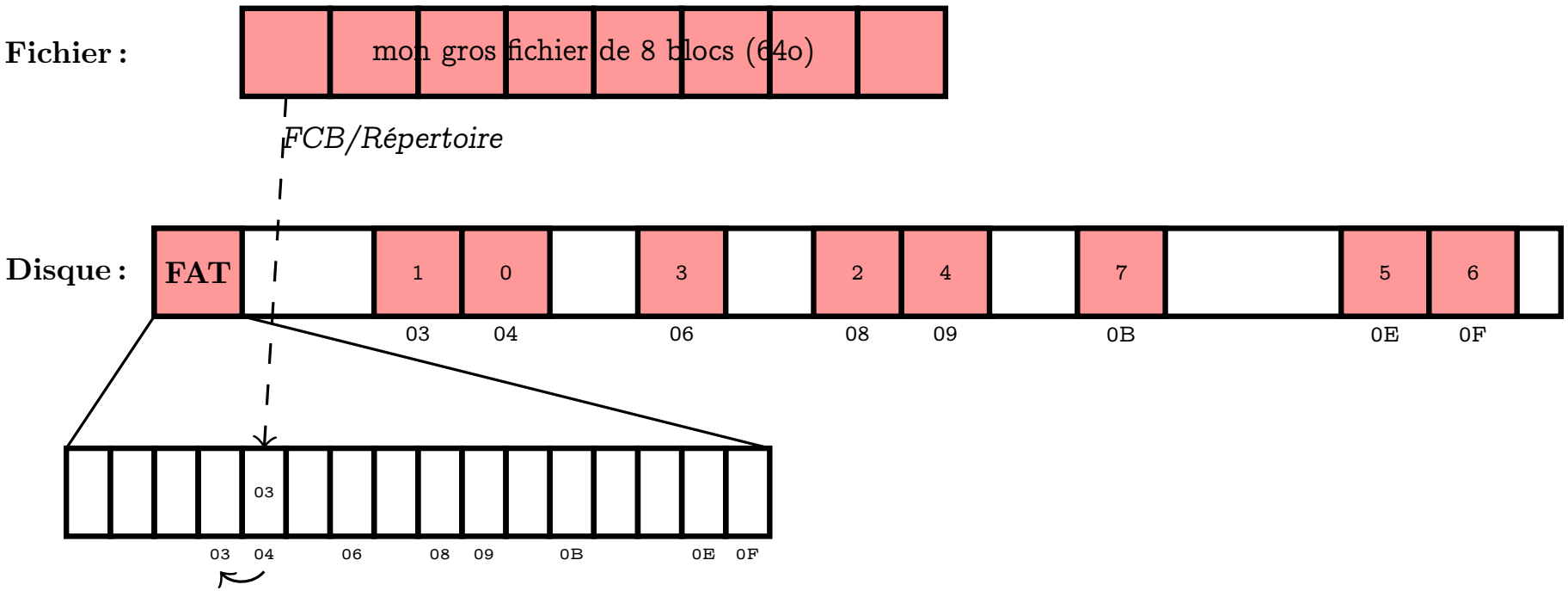
Principe

- Utilisé sous MSDOS (Intel) et OS/2 (IBM)
- Allocation indexée
- Liste chaînée des index en début de partition



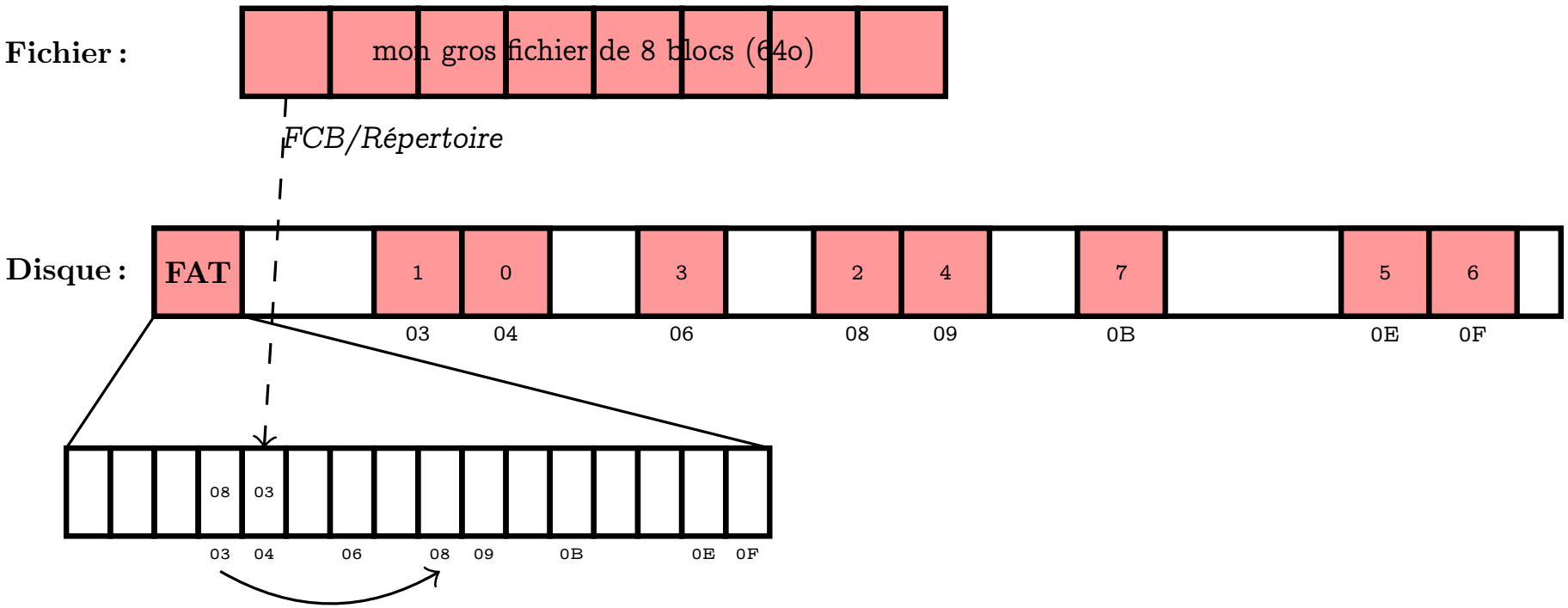
Principe

- Utilisé sous MSDOS (Intel) et OS/2 (IBM)
- Allocation indexée
- Liste chaînée des index en début de partition



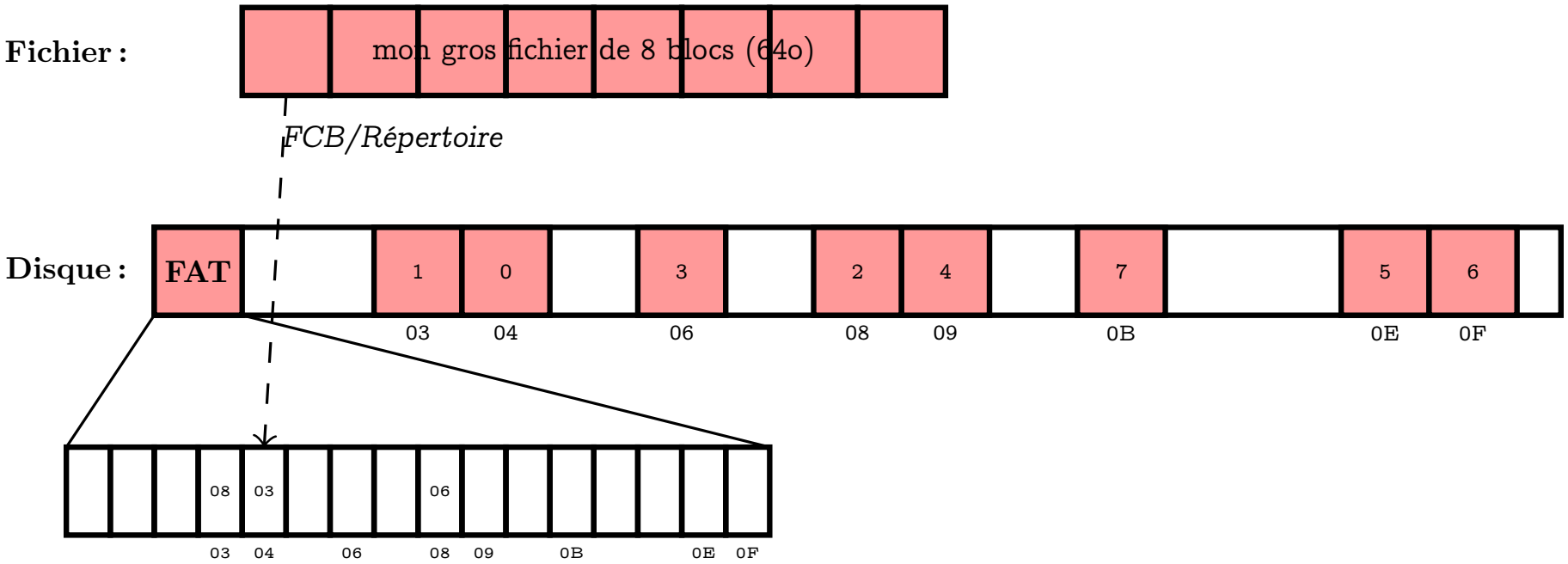
Principe

- Utilisé sous MSDOS (Intel) et OS/2 (IBM)
- Allocation indexée
- Liste chaînée des index en début de partition



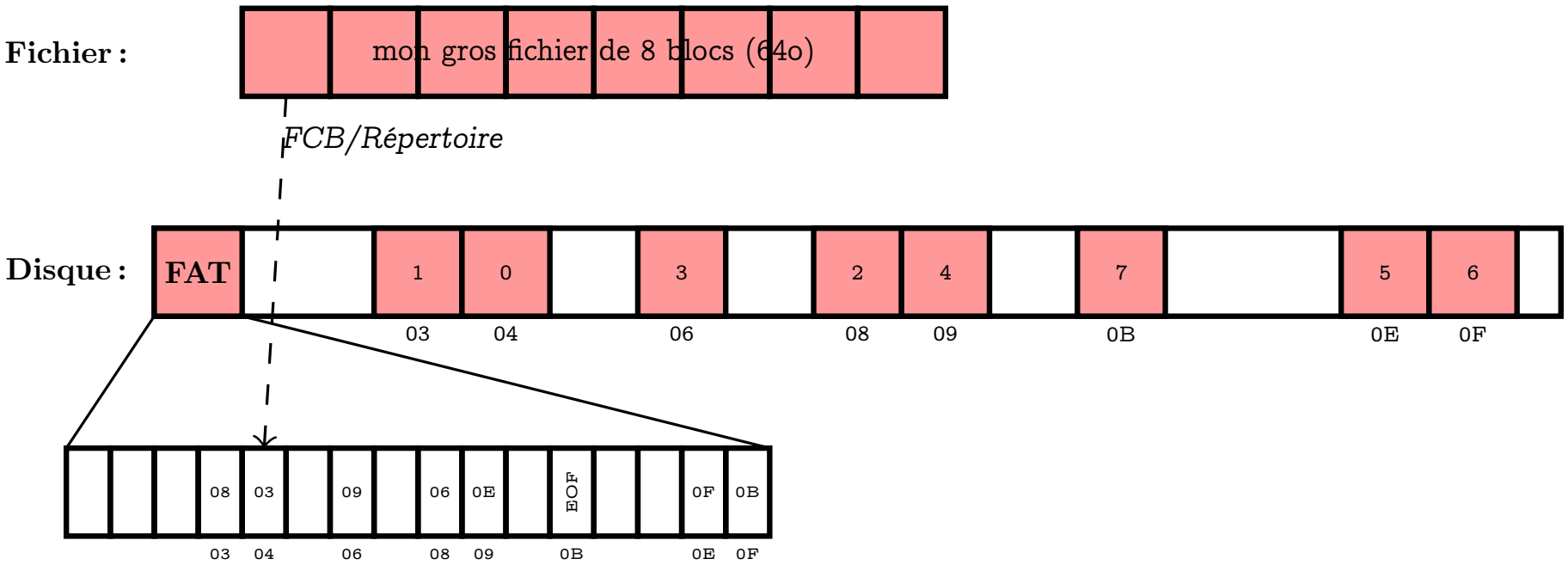
Principe

- Utilisé sous MSDOS (Intel) et OS/2 (IBM)
- Allocation indexée
- Liste chaînée des index en début de partition



Principe

- Utilisé sous MSDOS (Intel) et OS/2 (IBM)
- Allocation indexée
- Liste chaînée des index en début de partition



File Allocation Table (FAT)

Principe

- Allocation indexée
- Liste chaînée des index en début de partition

Avantages

✓ FCB : adresse premier bloc = premier index
✓ Pas de fragmentation (allocation indexée)
✓ Allocation bloc simple
✓ Accès rapide (FAT chargée en cache)

File Allocation Table (FAT)

Principe

- Allocation indexée
- Liste chaînée des index en début de partition

Avantages

✓ FCB : adresse premier bloc = premier index
✓ Pas de fragmentation (allocation indexée)
✓ Allocation bloc simple
✓ Accès rapide (FAT chargée en cache)

Inconvénients

- Fiabilité : FAT perdue $\rightarrow$ disque foutu !

File Allocation Table (FAT)

Principe

- Allocation indexée
- Liste chaînée des index en début de partition

Avantages

✓ FCB : adresse premier bloc = premier index
✓ Pas de fragmentation (allocation indexée)
✓ Allocation bloc simple
✓ Accès rapide (FAT chargée en cache)

Inconvénients

- Fiabilité : FAT perdue $\rightarrow$ disque foutu !

doubler la FAT (sur 2 blocs distincts)

Support physiques

Système de fichiers

- Découpage des fichiers en blocs logiques
- Allocation des blocs sur le support physique

Système de fichiers

- Découpage des fichiers en blocs logiques
- Allocation des blocs sur le support physique

Problèmes

- Accès aux blocs
  - Minimiser le temps de réponse du matériel
  - Dépend du matériel et des algorithmes
- Garantir l'intégrité des données
  - Vérifier les blocs

Principe

Le disque est beaucoup plus lent que la RAM

- Ne pas bloquer le processeur pendant le chargement des blocs

$\rightarrow$ tampon et transfert par bloc

Principe

Le disque est beaucoup plus lent que la RAM

- Ne pas bloquer le processeur pendant le chargement des blocs

$\rightarrow$ tampon et transfert par bloc

Tampon

Espace de stockage plus petit mais plus rapide conservant les données les plus utilisées

Principe

Le disque est beaucoup plus lent que la RAM

- Ne pas bloquer le processeur pendant le chargement des blocs

$\rightarrow$ tampon et transfert par bloc

Tampon

Espace de stockage plus petit mais plus rapide conservant les données les plus utilisées

Utilisation

- Gestion : *cf.* algos remplacement de pages
- Utilisé pour pages et blocs disques

Détection des secteurs défectueux

Secteur défectueux $\longrightarrow$ relecture $\neq$ écriture

Détection des secteurs défectueux

Secteur défectueux $\longrightarrow$ relecture $\neq$ écriture

Code d'erreur

Fonction de l'ensemble des données du bloc

- Stocké sur le secteur
- Comparé avec $\phi(\textit{données\_secteur})$

Détection des secteurs défectueux

Secteur défectueux $\rightarrow$ relecture $\neq$ écriture

Code d'erreur

Fonction de l'ensemble des données du bloc

- Stocké sur le secteur
- Comparé avec $\phi(\textit{données\_secteur})$

Exemple : Somme de contrôle

2 bits de données $\rightarrow$ 1 bit de parité de la somme

- 0001 1011 0110 1100 $\rightarrow^{\Phi}$ 0110 1100
- Vérification : 000 $\rightarrow$ ok, 010 $\rightarrow$ erreur

Détection des secteurs défectueux

Secteur défectueux $\longrightarrow$ relecture $\neq$ écriture

Code d'erreur

Fonction de l'ensemble des données du bloc

- Stocké sur le secteur
- Comparé avec $\phi(\textit{données\_secteur})$

Exemple : Somme de contrôle

2 bits de données $\longrightarrow$ 1 bit de parité de la somme

- 0001 1011 0110 1100 $\longrightarrow^{\Phi}$ 0110 1100
- Vérification : 000 $\longrightarrow$ ok, 010 $\longrightarrow$ erreur

mais on ne peut pas savoir lequel des 3 bits a été modifié...

Disques durs

Fiches perforées (années 50)

- X Capacité (nombre de trous)

Fiches perforées (années 50)

- ✗ Capacité (nombre de trous)

Bandes magnétiques (années 60)

- ✓ Capacité
- ✗ Allocation contiguë

Fiches perforées (années 50)

✗ Capacité (nombre de trous)

Bandes magnétiques (années 60)

✓ Capacité

✗ Allocation contiguë

Disques durs/disquettes (années 70)

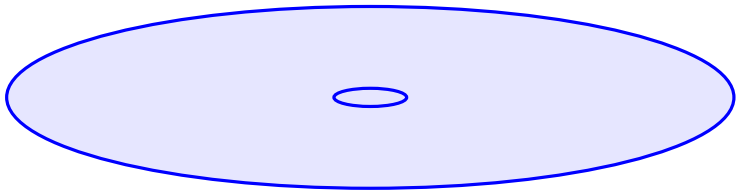
✓ Capacité

✓ Allocation libre

✗ Fragilité

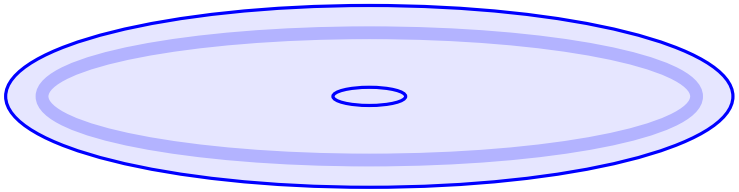
Disque

- Plaque circulaire



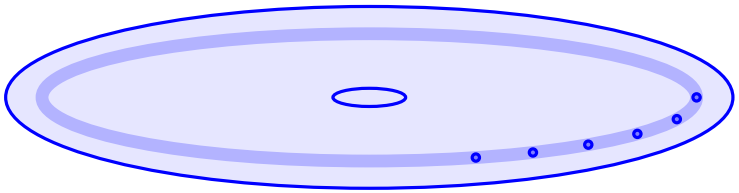
Disque

- Plaque circulaire $\longrightarrow$ n pistes concentriques



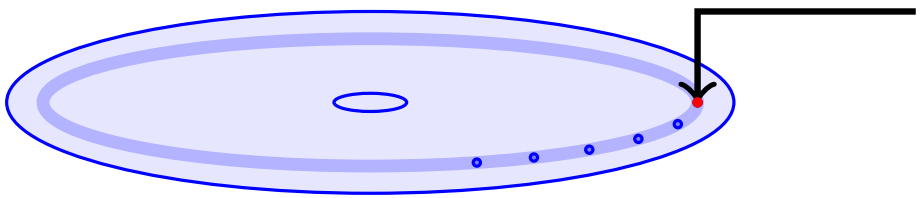
Disque

- Plaque circulaire $\longrightarrow$ n pistes concentriques
- m secteurs (cadre de blocs) par piste



Disque

- Plaque circulaire $\longrightarrow$ n pistes concentriques
- m secteurs (cadre de blocs) par piste
- Tête de lecture mobile

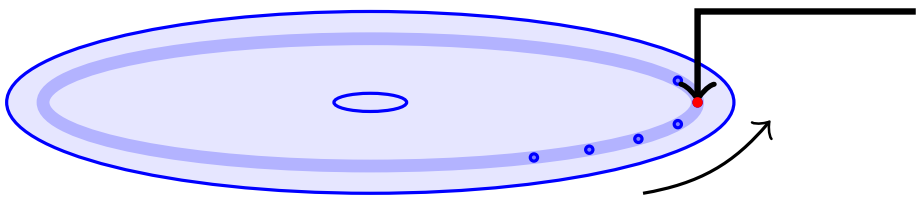


Fonctionnement

- La tête lit 1 secteur à la fois

Disque

- Plaque circulaire $\longrightarrow$ n pistes concentriques
- m secteurs (cadre de blocs) par piste
- Tête de lecture mobile

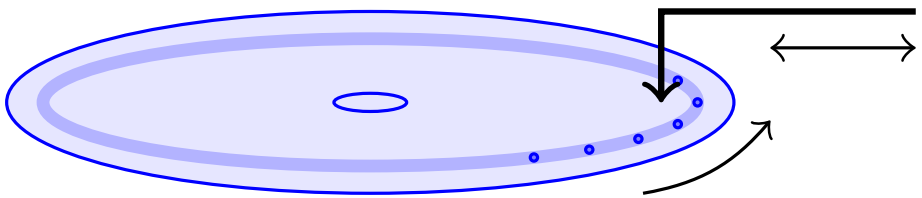


Fonctionnement

- La tête lit 1 secteur à la fois
- Rotation disque $\longrightarrow$ lecture des secteurs de la piste

Disque

- Plaque circulaire $\longrightarrow$ n pistes concentriques
- m secteurs (cadre de blocs) par piste
- Tête de lecture mobile



Fonctionnement

- La tête lit 1 secteur à la fois
- Rotation disque $\longrightarrow$ lecture des secteurs de la piste
- Déplacement tête $\longrightarrow$ lecture des autres pistes

Nombre de secteurs par piste

Les pistes concentriques sont toutes de tailles différentes !

- Combien de secteurs par piste ?

Nombre de secteurs par piste

Les pistes concentriques sont toutes de tailles différentes !

- Combien de secteurs par piste ?

Nombre de secteurs variable ?

- Difficile à gérer pour le contrôleur !

Ex : conversion bloc logique $\longrightarrow$ adresse physique (piste,secteur)

- Temps d'accès variable selon piste !

Nombre de secteurs par piste

Les pistes concentriques sont toutes de tailles différentes !

- Combien de secteurs par piste ?

Nombre de secteurs variable ?

- Difficile à gérer pour le contrôleur !

Ex : conversion bloc logique $\longrightarrow$ adresse physique (piste,secteur)

- Temps d'accès variable selon piste !

Le nombre de secteurs par pistes est constant

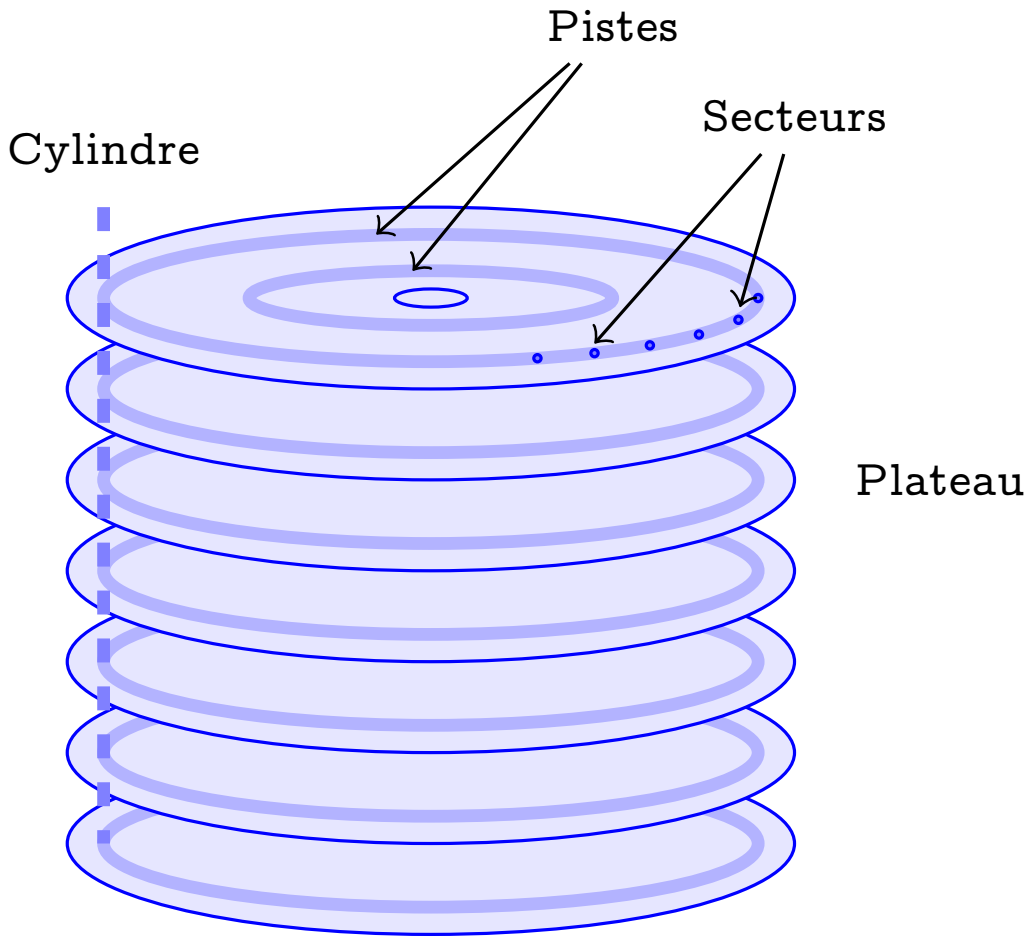
(ils sont plus ou moins étalés)

Structure d'un disque dur

Cylindres

Plusieurs disques empilés, appelés plateaux

Les pistes de même rayon forment un cylindre



Structure d'un disque dur

Disque dur

- Un disque dur est composé de n cylindres
- Chaque cylindre est composé de m pistes
- Chaque piste est composée de k secteurs

Disque dur

- Un disque dur est composé de n cylindres
- Chaque cylindre est composé de m pistes
- Chaque piste est composée de k secteurs

Tête de lecture

La tête de lecture est composée de :

- Un bras mobile en rateau
- m têtes fixes aux extrémités
- Un multiplexeur pour sélectionner la piste à lire

$\rightarrow$ Chaque tête s'insère au dessus d'une piste

Secteurs

Un secteur peut contenir un seul bloc de données

Secteurs

Un secteur peut contenir un seul bloc de données

Numérotation

- Le bloc 0 est sur le premier secteur de la première piste du cylindre superieur
- Par secteur croissant, puis par piste croissante, puis par cylindre

Secteurs

Un secteur peut contenir un seul bloc de données

Numérotation

- Le bloc 0 est sur le premier secteur de la première piste du cylindre superieur
- Par secteur croissant, puis par piste croissante, puis par cylindre

Bloc logique

- Adresse physique = (cylindre,piste,secteur)
- Bloc logique $\rightarrow$ adresse physique

Secteur défectueux

Dans tout support physique, certains secteurs deviennent inutilisables avec le temps

Gestion des secteurs défectueux

Secteur défectueux

Dans tout support physique, certains secteurs deviennent inutilisables avec le temps

Table des blocs

- Marquer le secteur inutilisable
- Modifier l'adresse physique du bloc logique

$\longrightarrow$ Il faut une table des blocs

Gestion des secteurs défectueux

Secteur défectueux

Dans tout support physique, certains secteurs deviennent inutilisables avec le temps

Table des blocs

- Marquer le secteur inutilisable
- Modifier l'adresse physique du bloc logique

$\rightarrow$ Il faut une table des blocs

Méthodes de gestion

- par l'OS + marquer secteurs défectueux dans la FAT
- par le contrôleur du périphérique

Structures de données

- Table des secteurs défectueux
- Réserver un ensemble de secteurs pour des remplacement lors du formatage du disque

Gestion des secteurs défectueux

Structures de données

- Table des secteurs défectueux
- Réserver un ensemble de secteurs pour des remplacement lors du formatage du disque

Contrôleur de périphérique

- Vérification à l'écriture
- Prévient l'OS d'un secteur défectueux
- L'OS demande un remplacement (glissement)
- Accès transparent pour l'OS qui ne voit que des blocs logiques

Ordonnancement

Accès à un secteur

- Positionnement de la tête de lecture sur la piste

  donc sur le bon cylindre...

- Rotation du disque (maximum un tour d'attente)

Accès à un secteur

- Positionnement de la tête de lecture sur la piste

  donc sur le bon cylindre...

- Rotation du disque (maximum un tour d'attente)

Temps d'accès

- Le disque tourne en permanence
- Vitesse de rotation = caractéristique matérielle
- Action sur les déplacements entre cylindres

Problème

Accès à un secteur

- Positionnement de la tête de lecture sur la piste

  donc sur le bon cylindre...
- Rotation du disque (maximum un tour d'attente)

Temps d'accès

- Le disque tourne en permanence
- Vitesse de rotation = caractéristique matérielle
- Action sur les déplacements entre cylindres

Principe

Minimiser le temps de déplacement de la tête en parcourant les cylindres dans un ordre intelligent

Exemple

Caractéristique disque

- Vitesse de rotation fixée
- 256 cylindres

Exemple

Caractéristique disque

- Vitesse de rotation fixée
- 256 cylindres

Requêtes

Au temps 0, la tête est sur le cylindre 53.

Le contrôleur reçoit les demandes suivantes :

98, 183, 37, 122, 14, 124, 65, 67

Exemple

Caractéristique disque

- Vitesse de rotation fixée
- 256 cylindres

Requêtes

Au temps 0, la tête est sur le cylindre 53.

Le contrôleur reçoit les demandes suivantes :

98, 183, 37, 122, 14, 124, 65, 67

Ordonnancement optimal

Pas d'autre demande $\longrightarrow$ trier et partir du plus proche

En pratique : la file est dynamique

First Come, First Served

Principe

Prendre les cylindres dans l'ordre

Principe

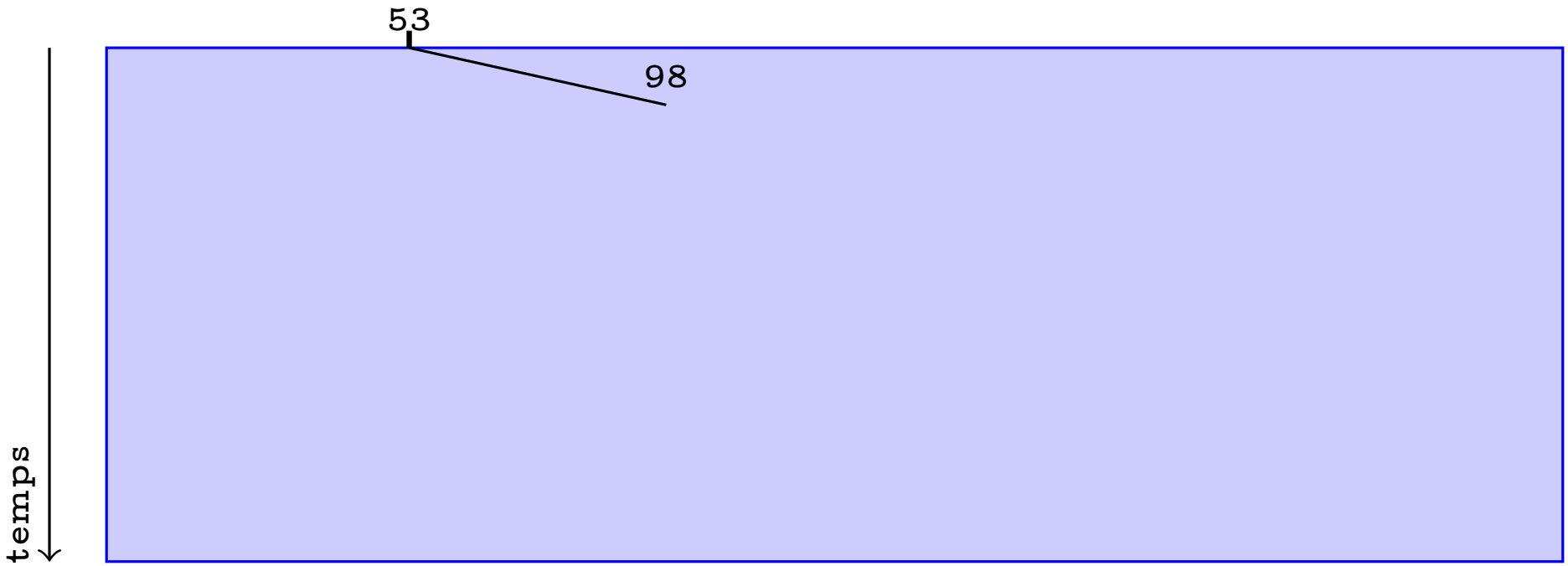
Prendre les cylindres dans l'ordre

Cylindre 53 + { 98, 183, 37, 122, 14, 124, 65, 67 }

Principe

Prendre les cylindres dans l'ordre

Cylindre 53 + { 98, 183, 37, 122, 14, 124, 65, 67 }

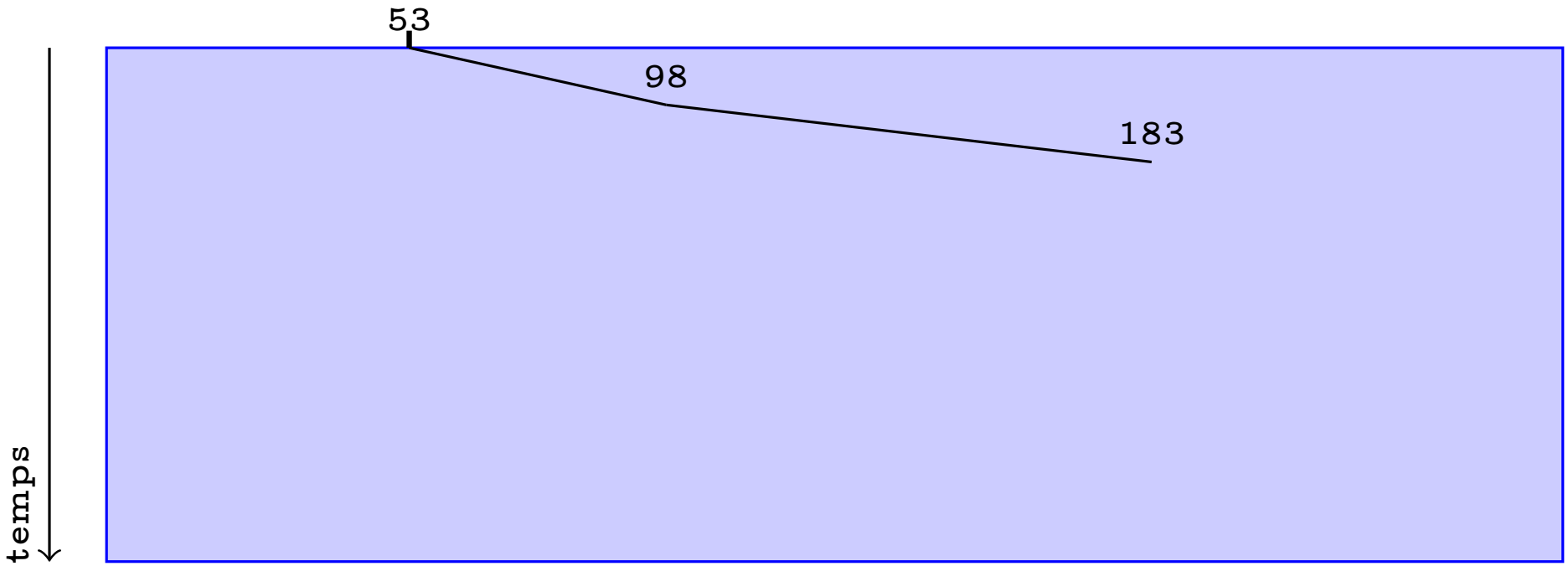


45+

Principe

Prendre les cylindres dans l'ordre

Cylindre 53 + { 98, 183, 37, 122, 14, 124, 65, 67 }

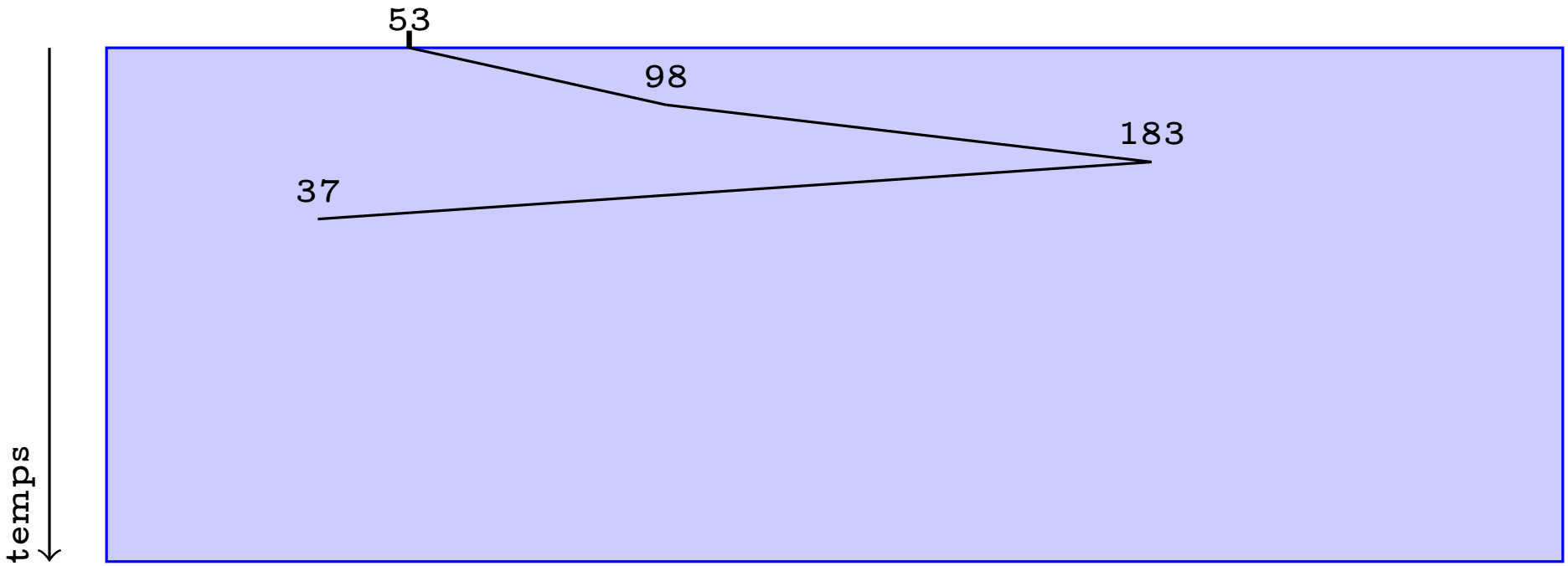


45+85+

Principe

Prendre les cylindres dans l'ordre

Cylindre 53 + { 98, 183, 37, 122, 14, 124, 65, 67 }

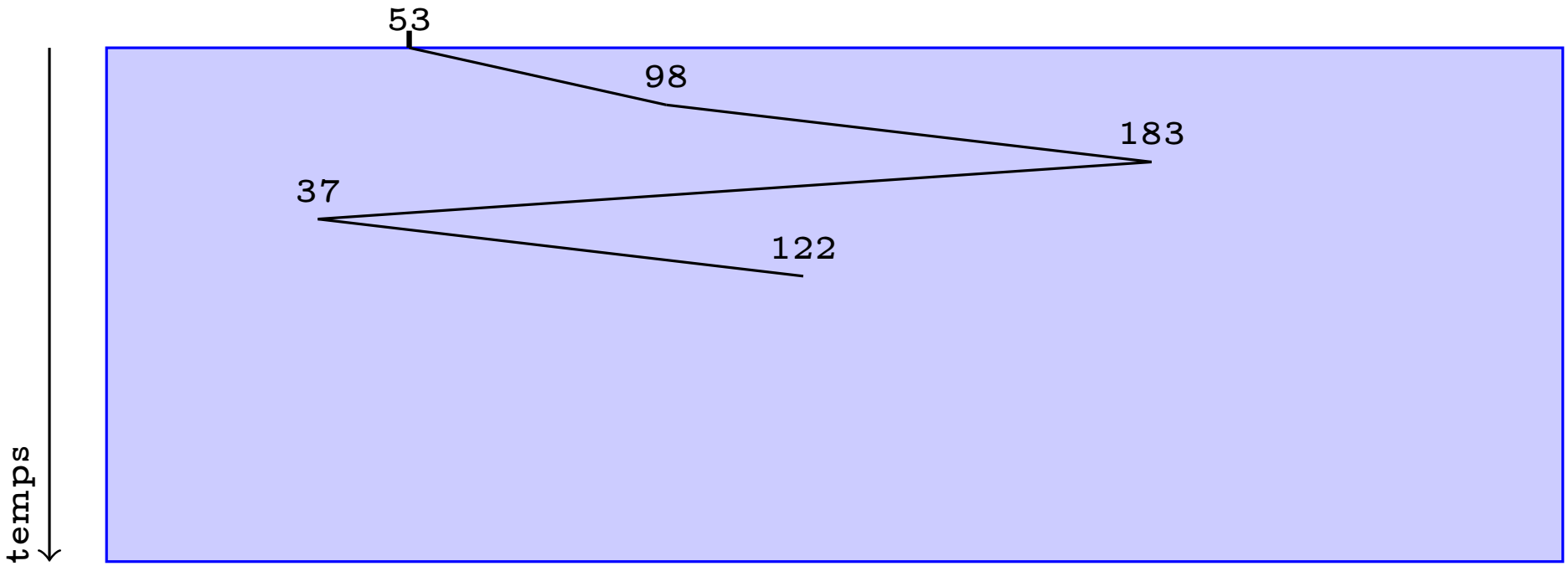


45+85+146+

Principe

Prendre les cylindres dans l'ordre

Cylindre 53 + { 98, 183, 37, 122, 14, 124, 65, 67 }

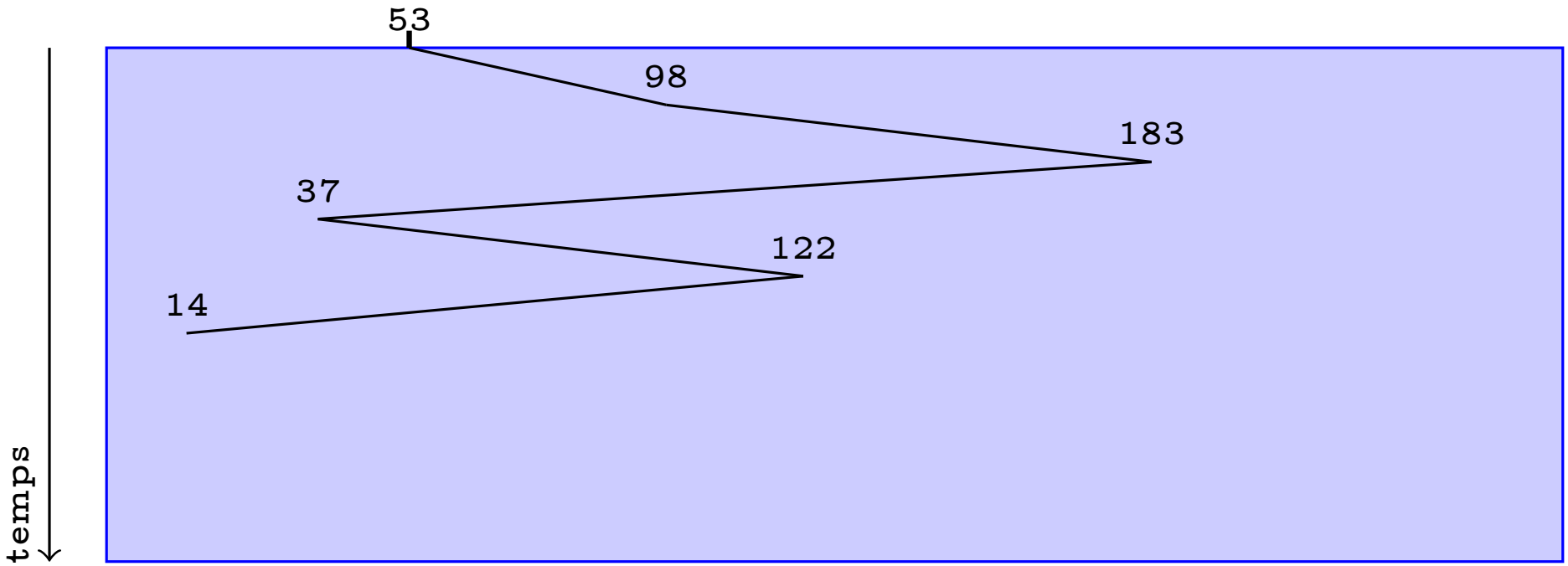


45+85+146+85+

Principe

Prendre les cylindres dans l'ordre

Cylindre 53 + { 98, 183, 37, 122, 14, 124, 65, 67 }

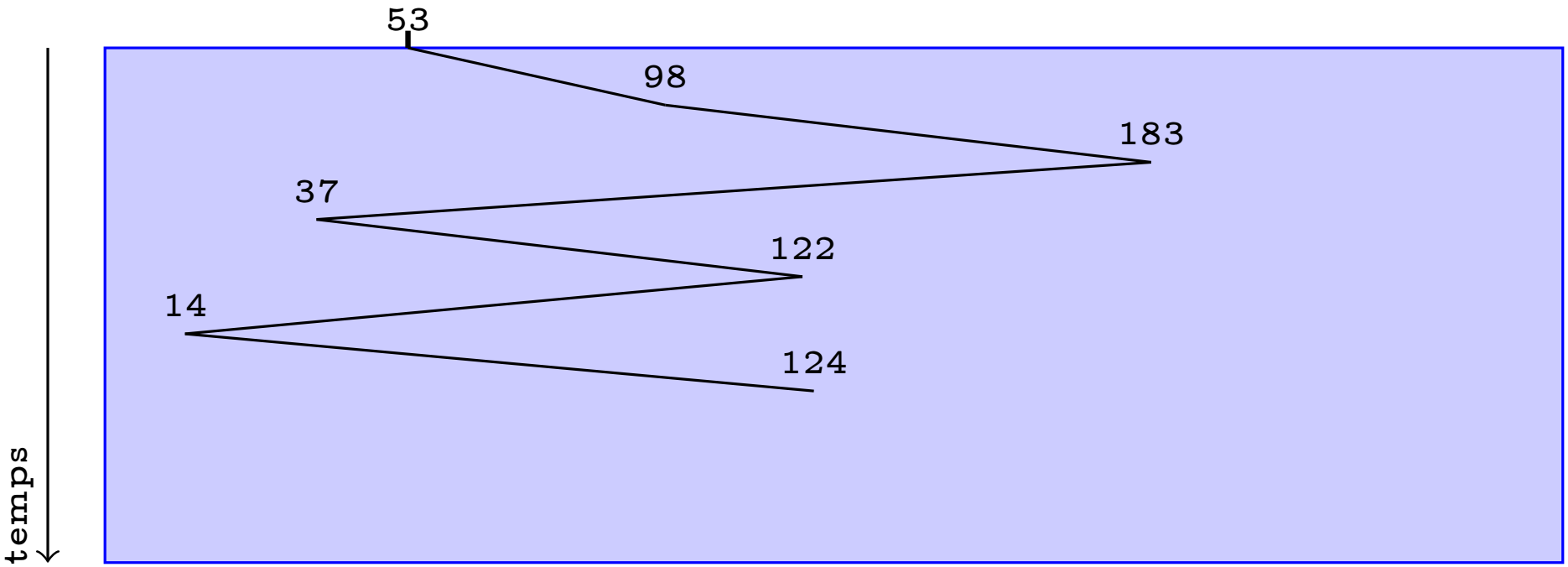


45+85+146+85+108+

Principe

Prendre les cylindres dans l'ordre

Cylindre 53 + { 98, 183, 37, 122, 14, 124, 65, 67 }

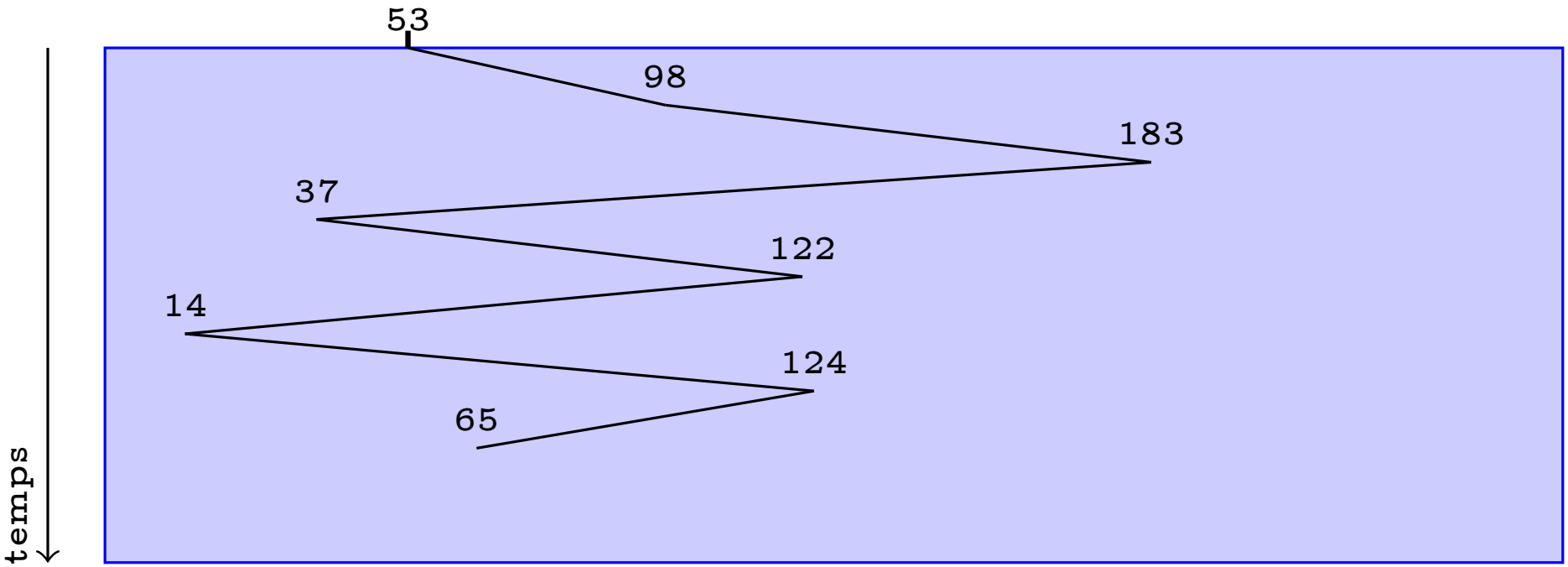


45+85+146+85+108+110+

Principe

Prendre les cylindres dans l'ordre

Cylindre 53 + { 98, 183, 37, 122, 14, 124, 65, 67 }

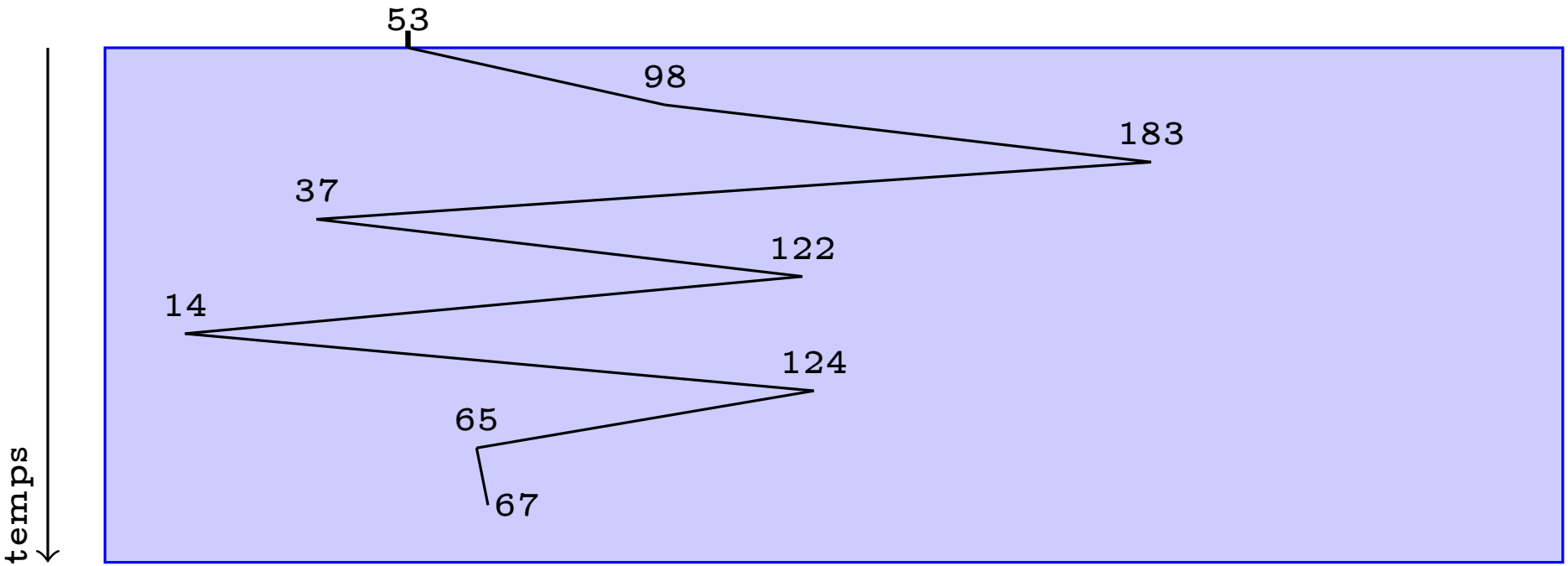


45+85+146+85+108+110+59+

Principe

Prendre les cylindres dans l'ordre

Cylindre 53 + { 98, 183, 37, 122, 14, 124, 65, 67 }



45+85+146+85+108+110+59+2=640 cylindres parcourus

Shortest Seek Time First

Principe

Aller vers le cylindre le plus proche

Shortest Seek Time First

Principe

Aller vers le cylindre le plus proche

Cylindre 53 + { 98, 183, 37, 122, 14, 124, 65, 67 }

Shortest Seek Time First

Principe

Aller vers le cylindre le plus proche

Cylindre 53 + { 98, 183, 37, 122, 14, 124, 65, 67 }

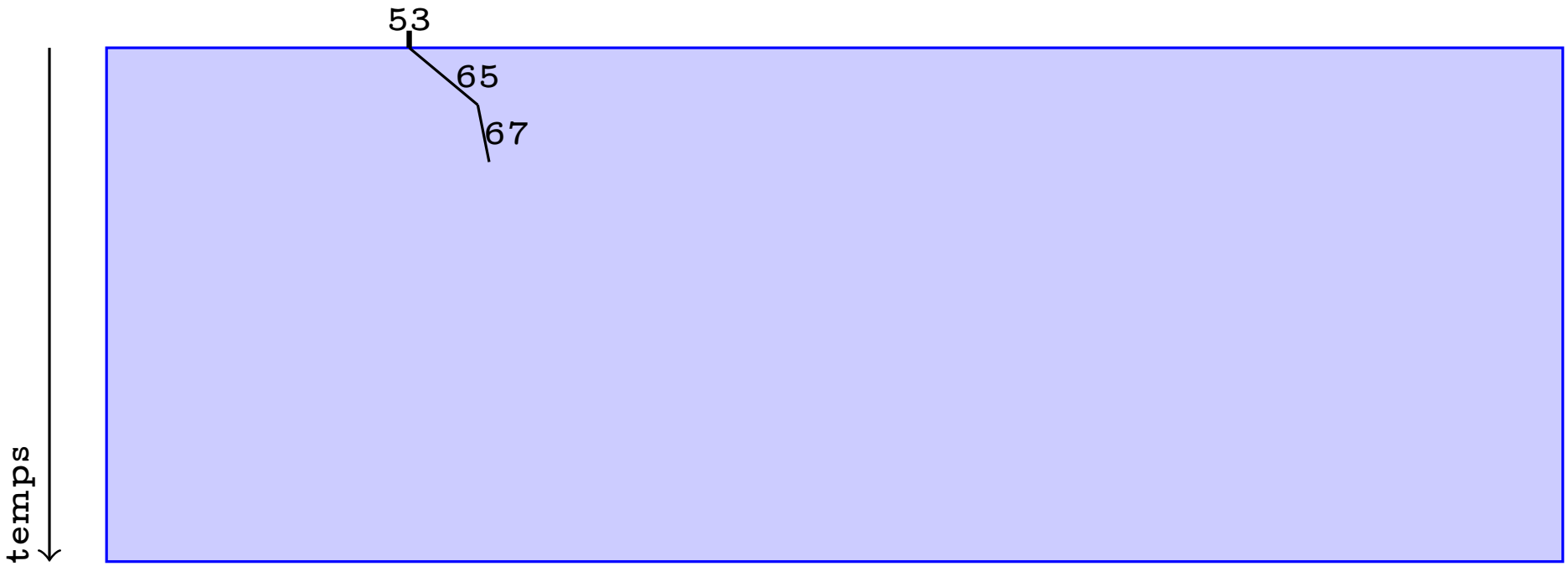
12+

Shortest Seek Time First

Principe

Aller vers le cylindre le plus proche

Cylindre 53 + { 98, 183, 37, 122, 14, 124, 65, 67 }

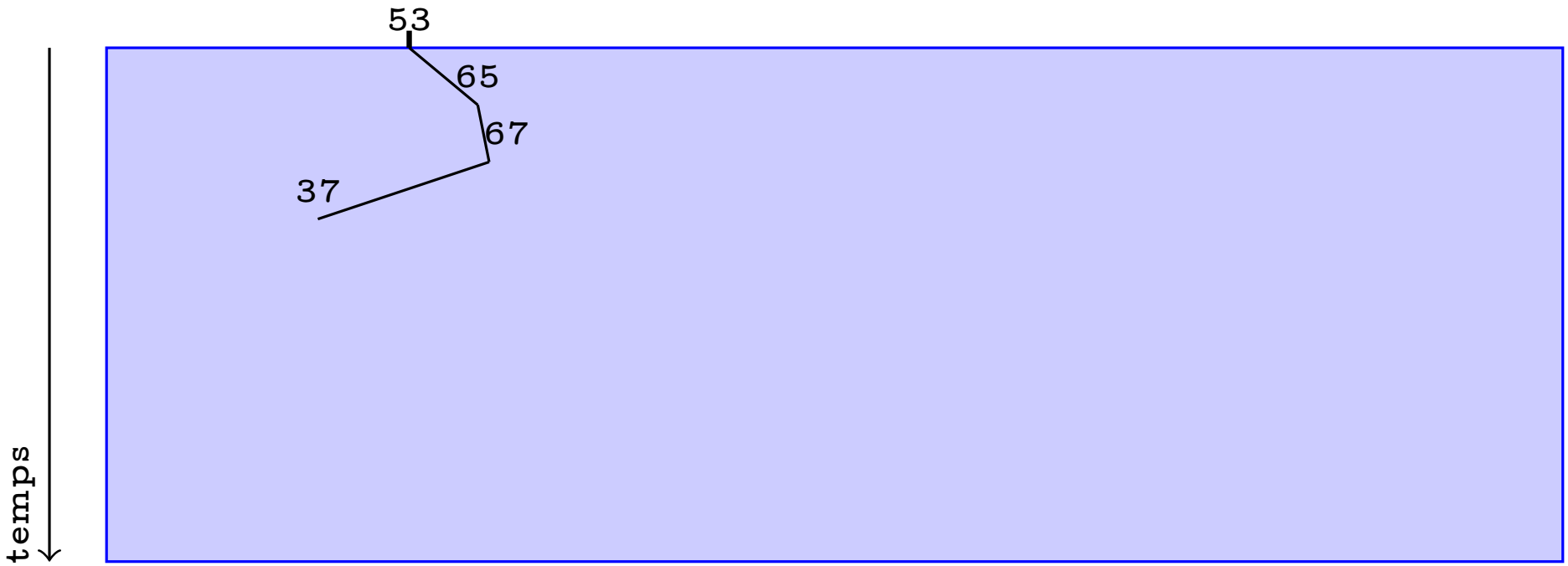


12+2+

Principe

Aller vers le cylindre le plus proche

Cylindre 53 + { 98, 183, 37, 122, 14, 124, 65, 67 }

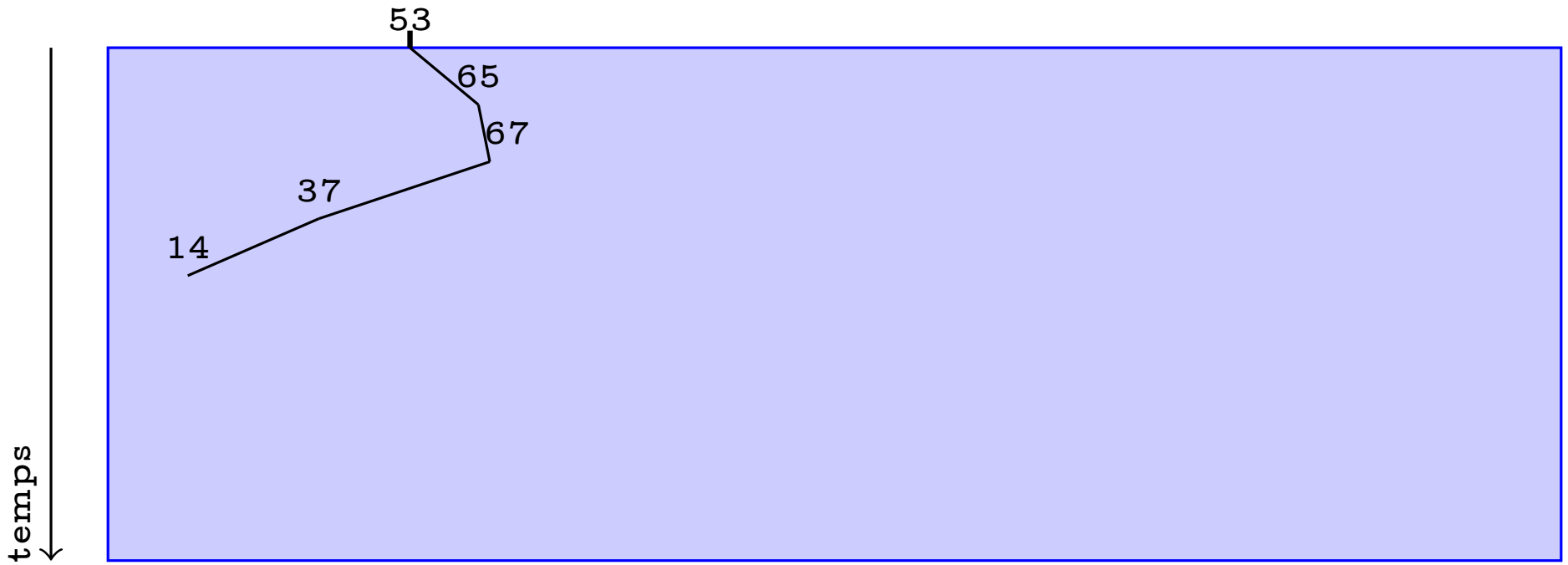


12+2+30+

Principe

Aller vers le cylindre le plus proche

Cylindre 53 + { 98, 183, 37, 122, 14, 124, 65, 67 }

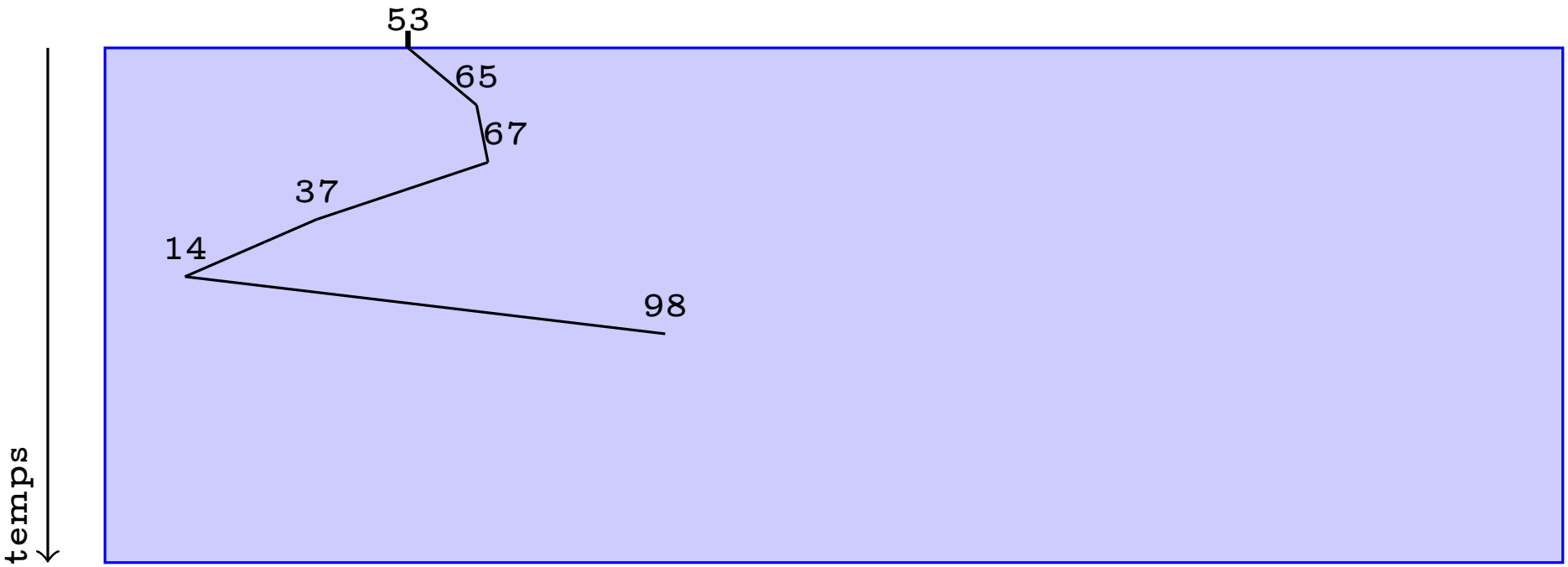


12+2+30+23+

Principe

Aller vers le cylindre le plus proche

Cylindre 53 + { 98, 183, 37, 122, 14, 124, 65, 67 }

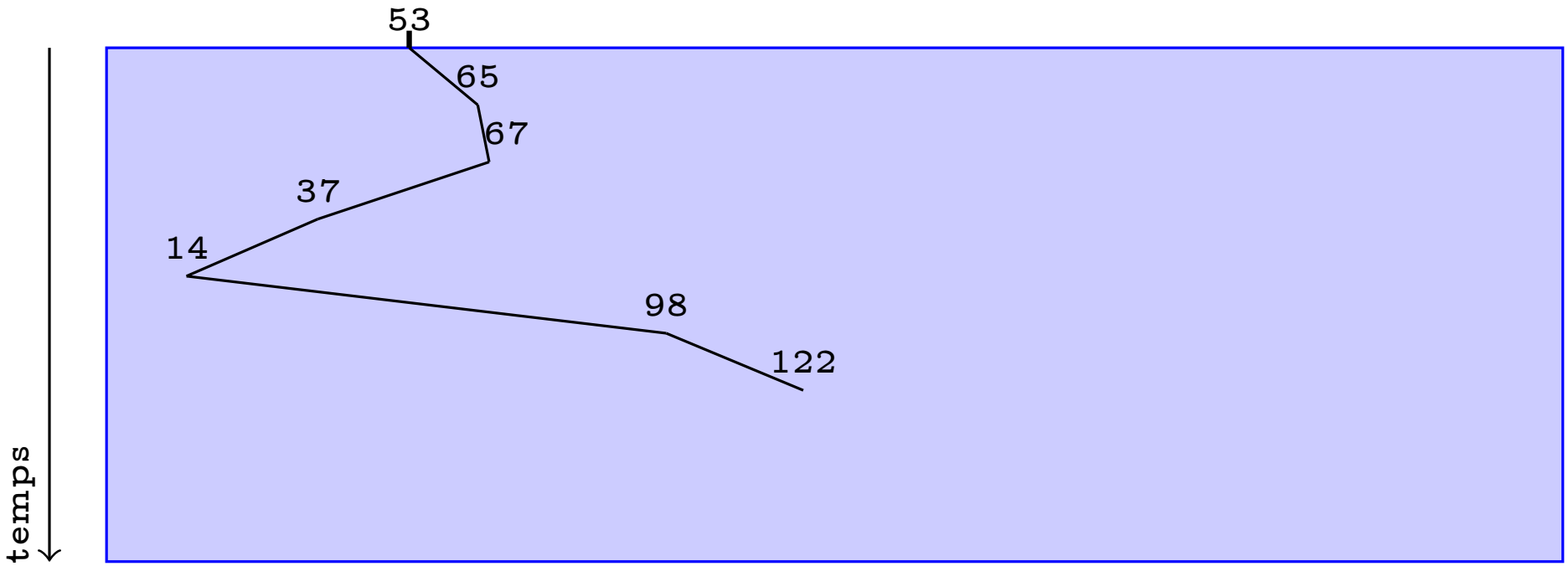


12+2+30+23+84+

Principe

Aller vers le cylindre le plus proche

Cylindre 53 + { 98, 183, 37, 122, 14, 124, 65, 67 }

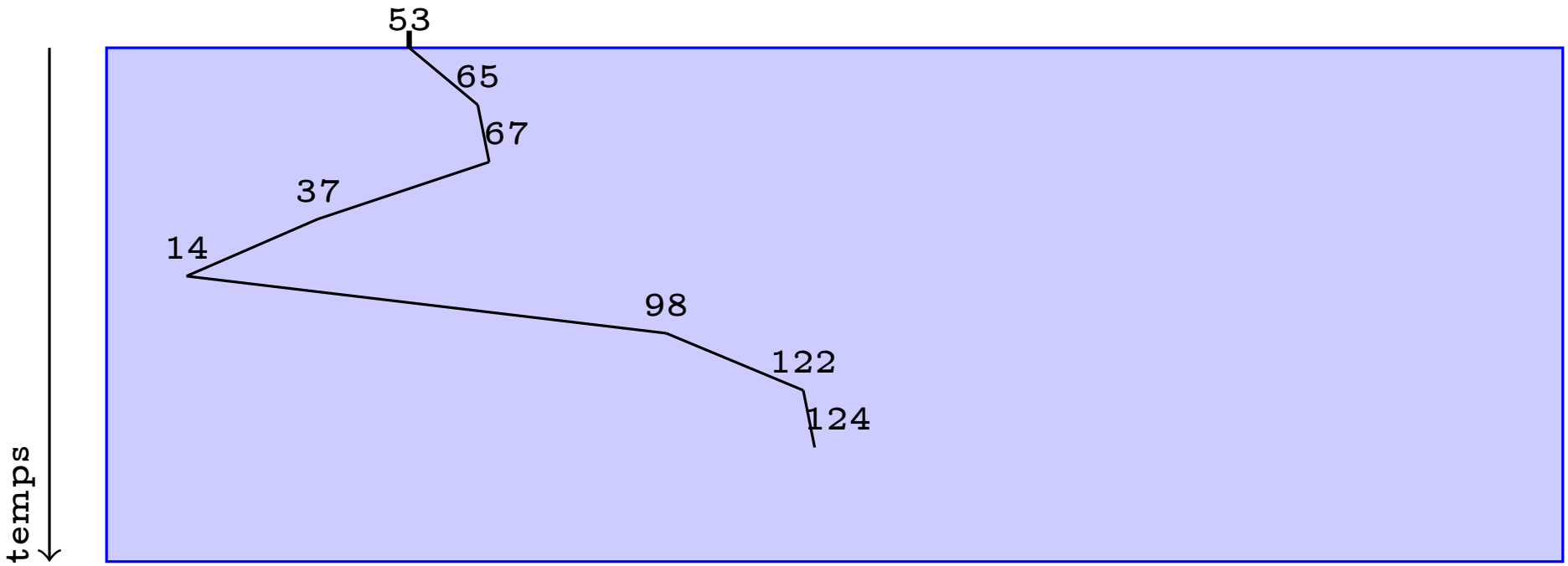


12+2+30+23+84+24+

Principe

Aller vers le cylindre le plus proche

Cylindre 53 + { 98, 183, 37, 122, 14, 124, 65, 67 }



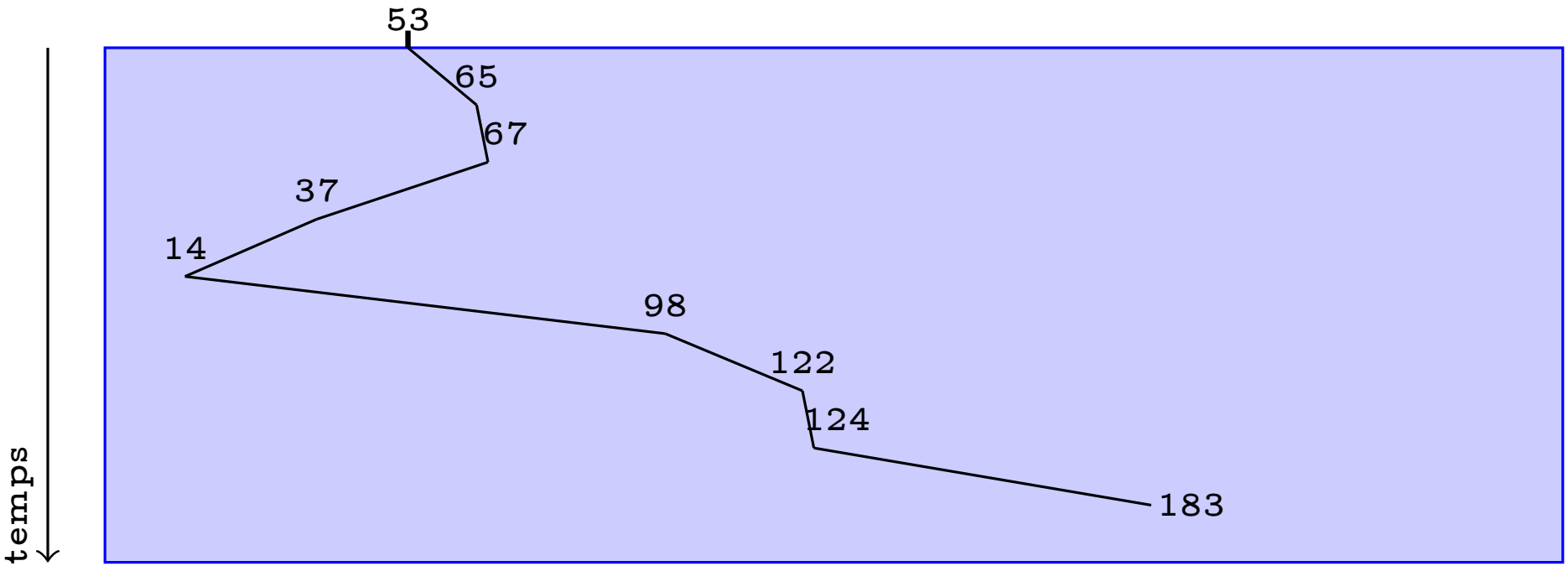
12+2+30+23+84+24+2+

Shortest Seek Time First

Principe

Aller vers le cylindre le plus proche

Cylindre 53 + { 98, 183, 37, 122, 14, 124, 65, 67 }



12+2+30+23+84+24+2+59=236 cylindres parcourus

Shortest Seek Time First

Avantages

✓ Temps de traitement souvent très bon

Shortest Seek Time First

Avantages

✓ Temps de traitement souvent très bon

Limites

X Pas forcément optimal...

→ Il faut tenir compte des nouvelles arrivées

Shortest Seek Time First

Avantages

✓ Temps de traitement souvent très bon

Limites

✗ Pas forcément optimal...

→ Il faut tenir compte des nouvelles arrivées

✗ Risque de famine !

Tant qu'il arrive des cylindres proches, on reste dans la zone et les autres cylindres ne sont pas servis !

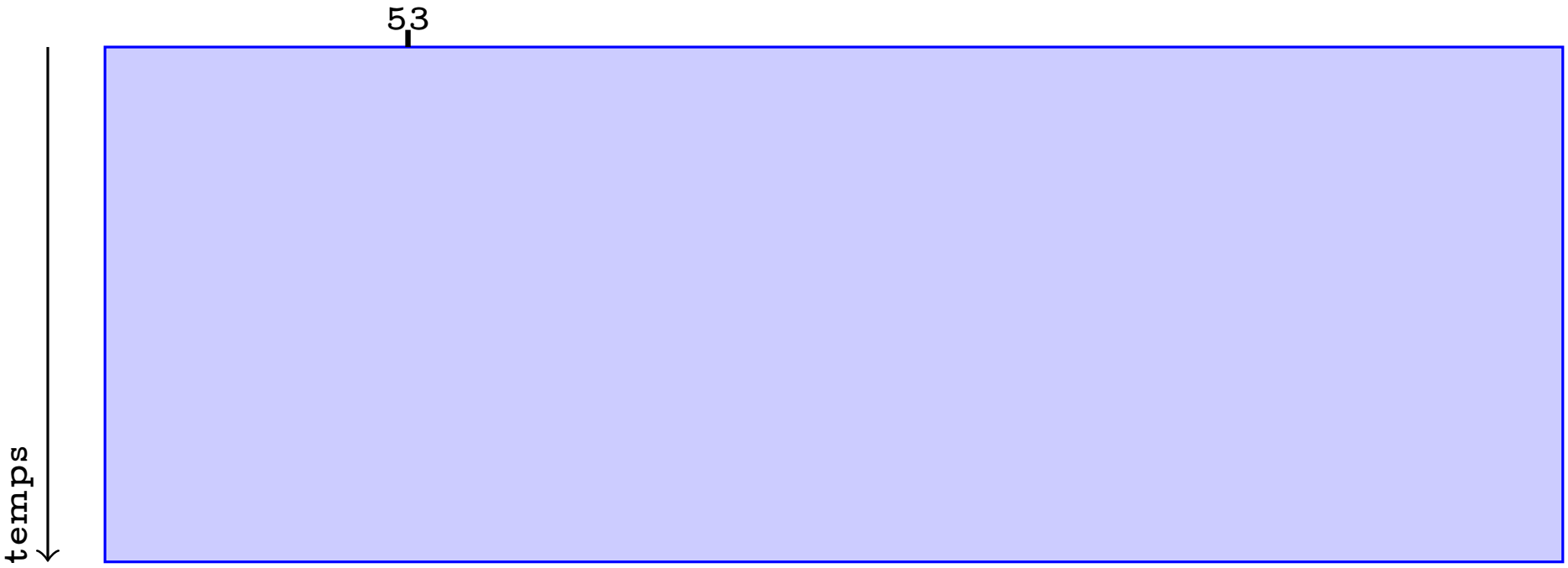
Principe

Balayer dans un sens puis dans l'autre

Principe

Balayer dans un sens puis dans l'autre

Cylindre 53 + { 98, 183, 37, 122, 14, 124, 65, 67}



Principe

Balayer dans un sens puis dans l'autre

Cylindre 53 + { 98, 183, 37, 122, 14, 124, 65, 67}

16+

Principe

Balayer dans un sens puis dans l'autre

Cylindre 53 + { 98, 183, 37, 122, 14, 124, 65, 67}

16+23+

Principe

Balayer dans un sens puis dans l'autre

Cylindre 53 + { 98, 183, 37, 122, 14, 124, 65, 67}

16+23+14+

Principe

Balayer dans un sens puis dans l'autre

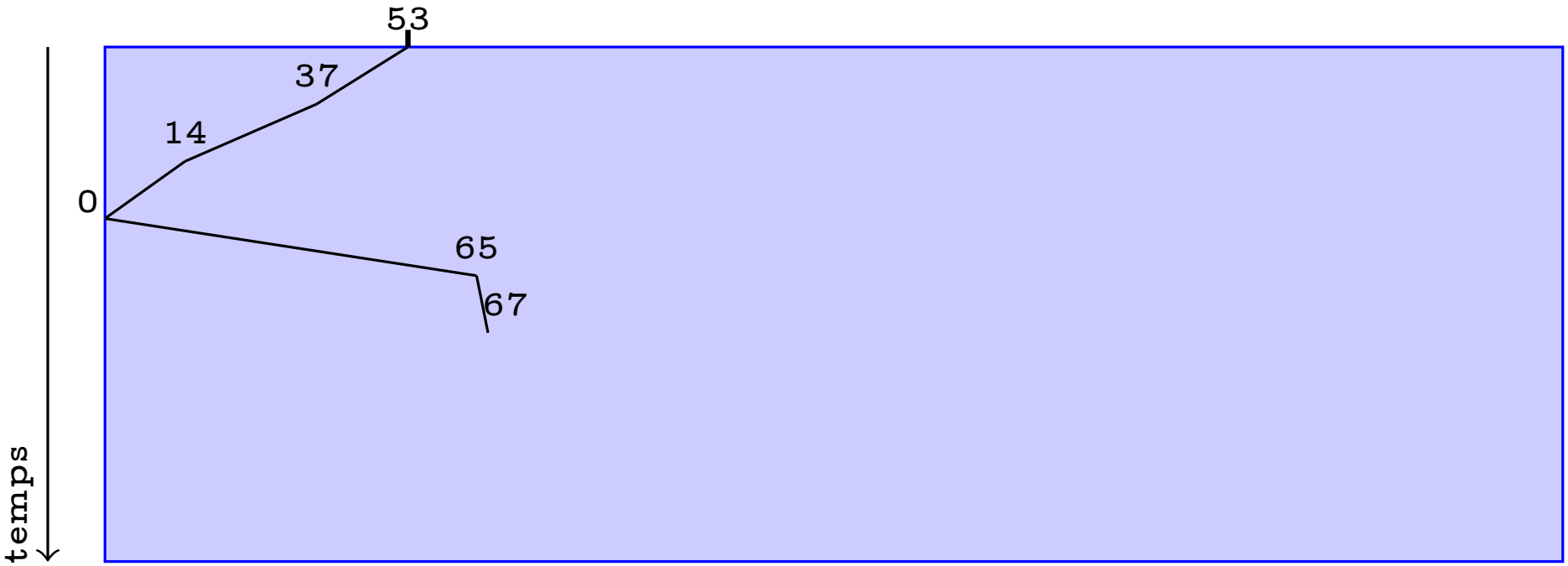
Cylindre 53 + { 98, 183, 37, 122, 14, 124, 65, 67}

16+23+14+65+

Principe

Balayer dans un sens puis dans l'autre

Cylindre 53 + $\{$ 98, 183, 37, 122, 14, 124, 65, 67$\}$

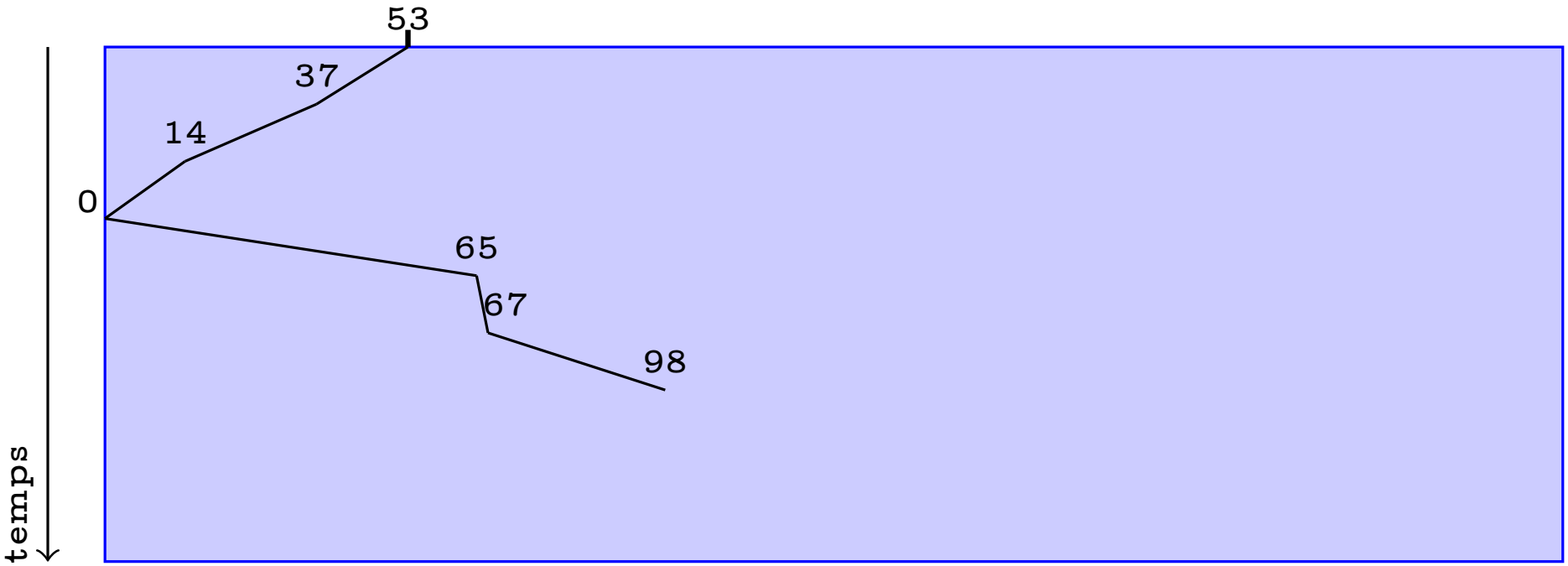


16+23+14+65+2+

Principe

Balayer dans un sens puis dans l'autre

Cylindre 53 + { 98, 183, 37, 122, 14, 124, 65, 67}

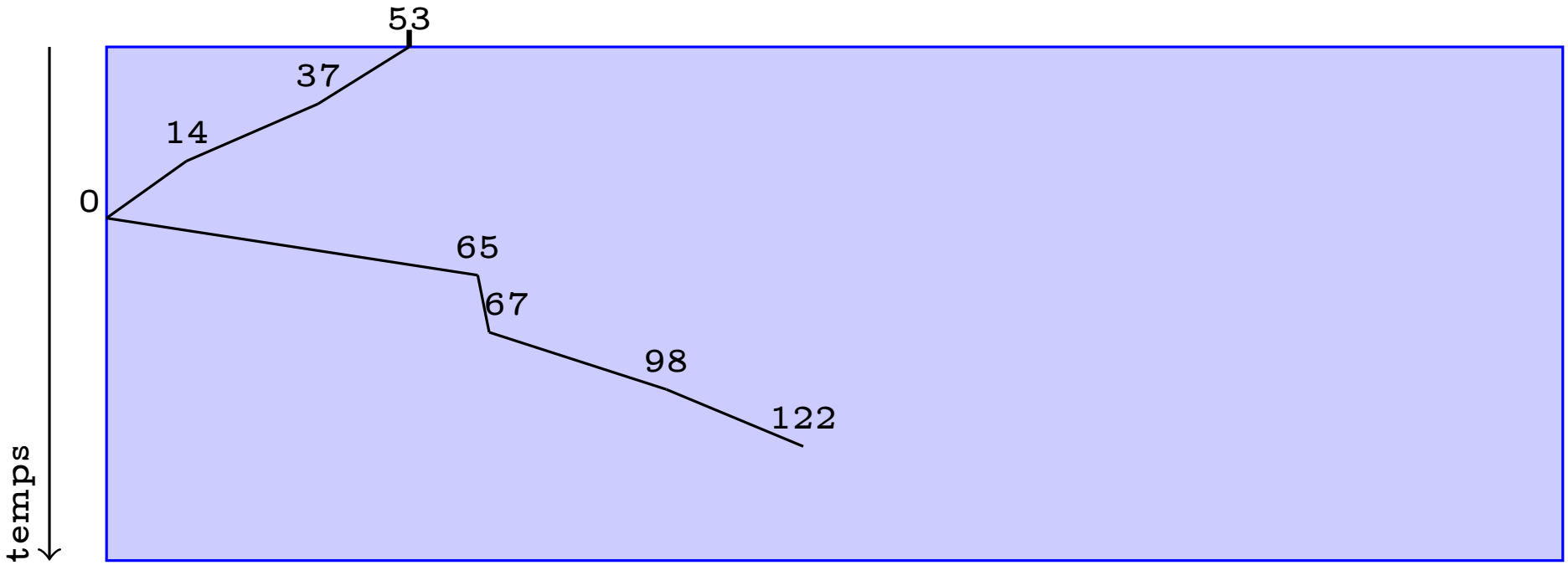


16+23+14+65+2+31+

Principe

Balayer dans un sens puis dans l'autre

Cylindre 53 + { 98, 183, 37, 122, 14, 124, 65, 67}

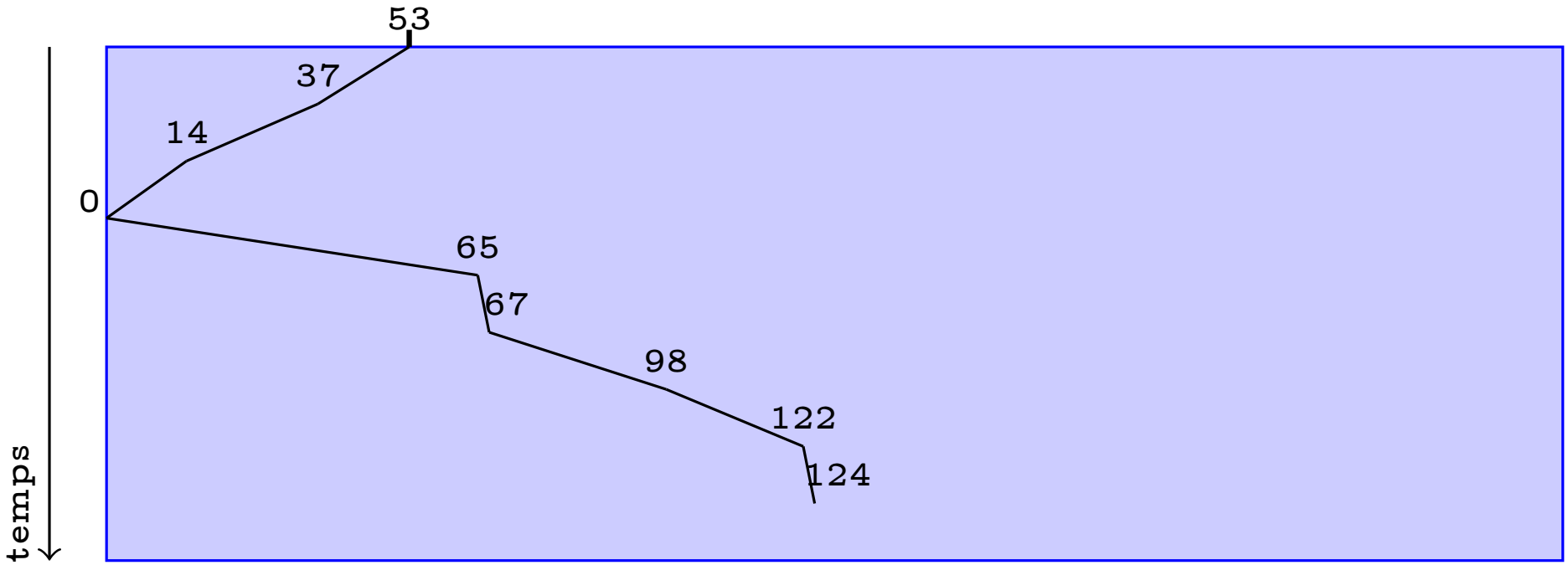


16+23+14+65+2+31+24+

Principe

Balayer dans un sens puis dans l'autre

Cylindre 53 + { 98, 183, 37, 122, 14, 124, 65, 67}

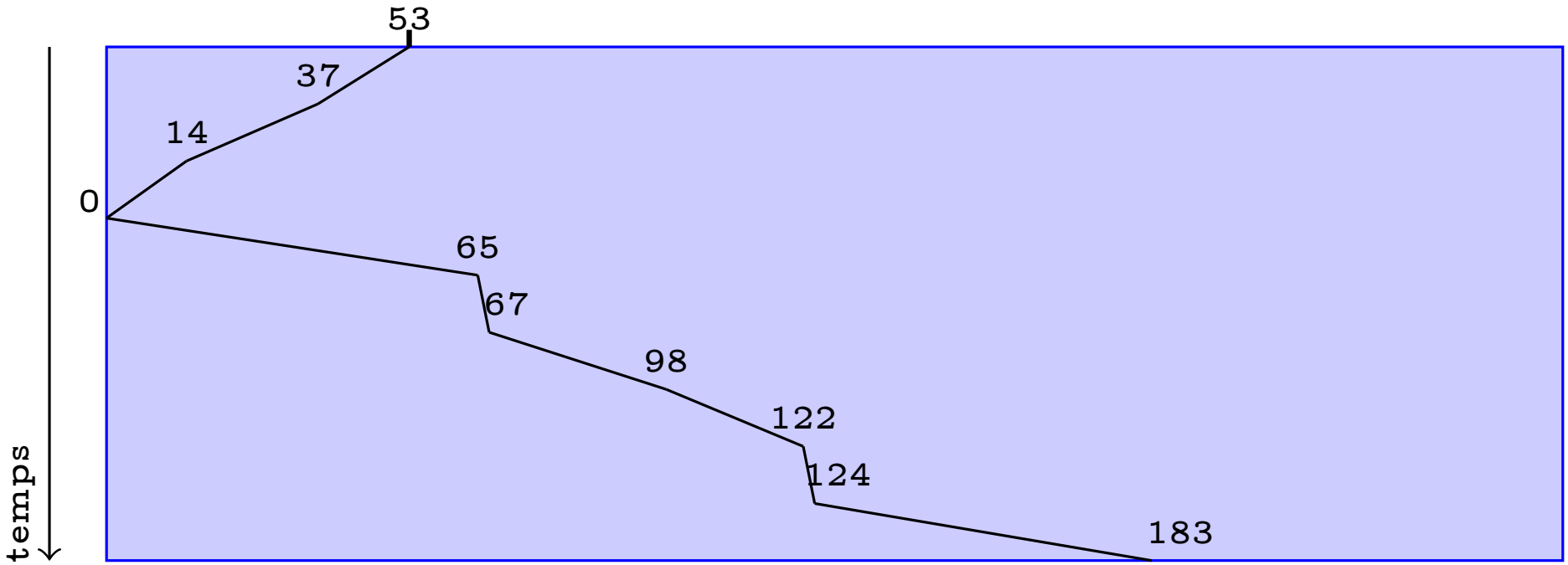


16+23+14+65+2+31+24+2+

Principe

Balayer dans un sens puis dans l'autre

Cylindre 53 + $\{$ 98, 183, 37, 122, 14, 124, 65, 67$\}$



16+23+14+65+2+31+24+2+59 = 53 + 183 = 236

Avantages

- ✓ Temps de traitement souvent très bon
- ✓ Pas de famine

Avantages

✓ Temps de traitement souvent très bon

✓ Pas de famine

Limites

- Parcours inutiles vers les bords
- Lorsqu'on fait demi-tour, on vient de servir les cylindres près du bord $\rightarrow$ il est peu probable d'en avoir beaucoup à traiter par ici...

même en tenant compte de nouvelles arrivées !

Principe

Repartir lorsqu'on a atteint le plus petit cylindre demandé

Principe

Repartir lorsqu'on a atteint le plus petit cylindre demandé

Cylindre 53 + { 98, 183, 37, 122, 14, 124, 65, 67 }

Principe

Repartir lorsqu'on a atteint le plus petit cylindre demandé

Cylindre 53 + { 98, 183, 37, 122, 14, 124, 65, 67 }

16+

Principe

Repartir lorsqu'on a atteint le plus petit cylindre demandé

Cylindre 53 + { 98, 183, 37, 122, 14, 124, 65, 67 }

16+23+

Principe

Repartir lorsqu'on a atteint le plus petit cylindre demandé

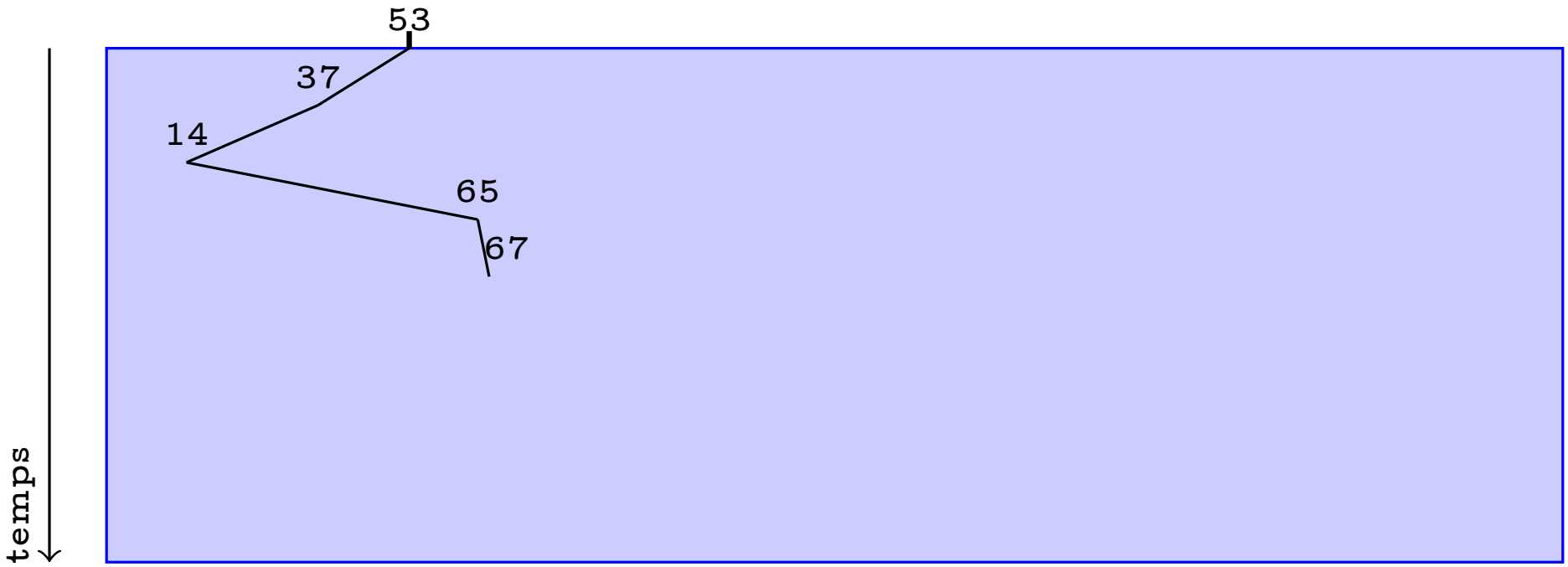
Cylindre 53 + { 98, 183, 37, 122, 14, 124, 65, 67 }

16+23+51+

Principe

Repartir lorsqu'on a atteint le plus petit cylindre demandé

Cylindre 53 + { 98, 183, 37, 122, 14, 124, 65, 67 }

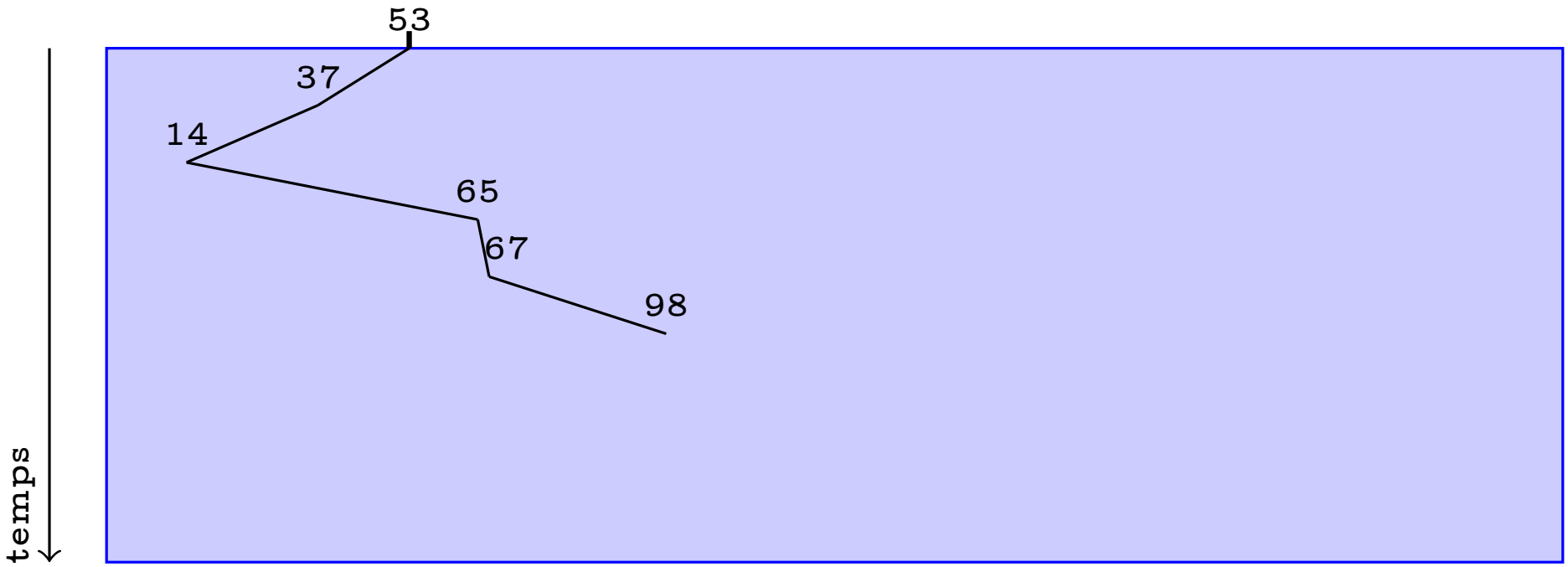


16+23+51+2+

Principe

Repartir lorsqu'on a atteint le plus petit cylindre demandé

Cylindre 53 + { 98, 183, 37, 122, 14, 124, 65, 67 }

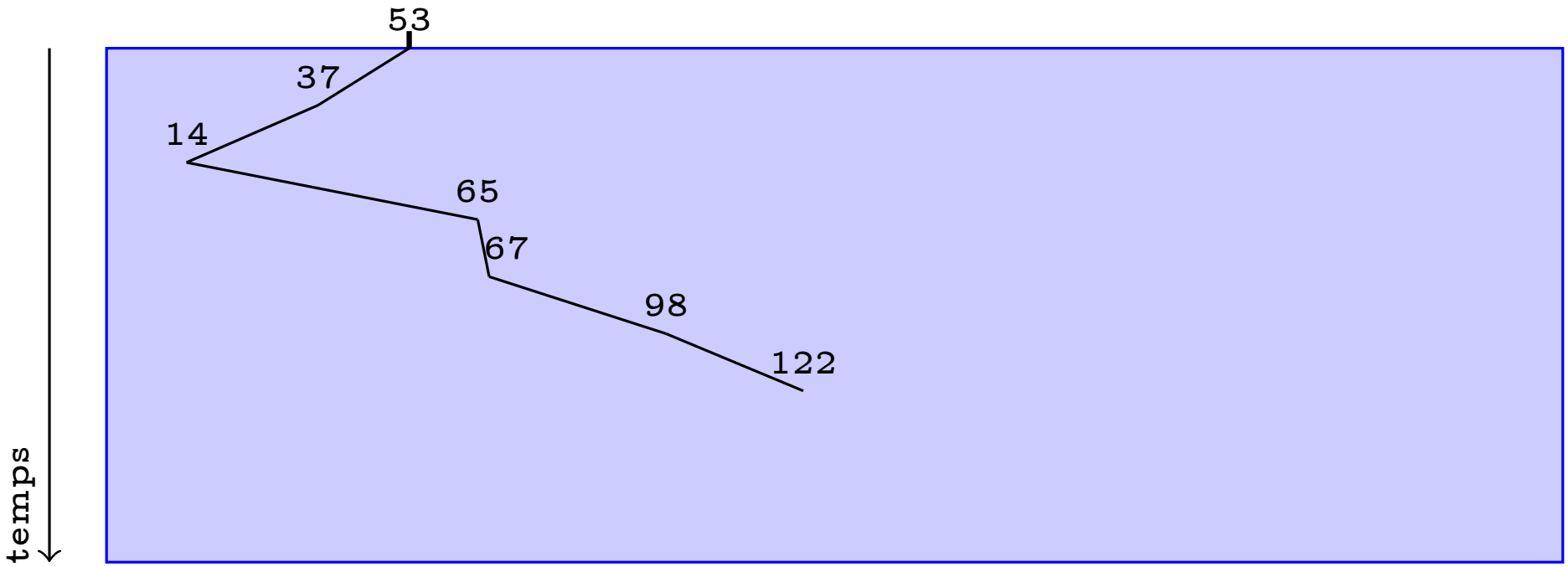


16+23+51+2+31+

Principe

Repartir lorsqu'on a atteint le plus petit cylindre demandé

Cylindre 53 + { 98, 183, 37, 122, 14, 124, 65, 67 }

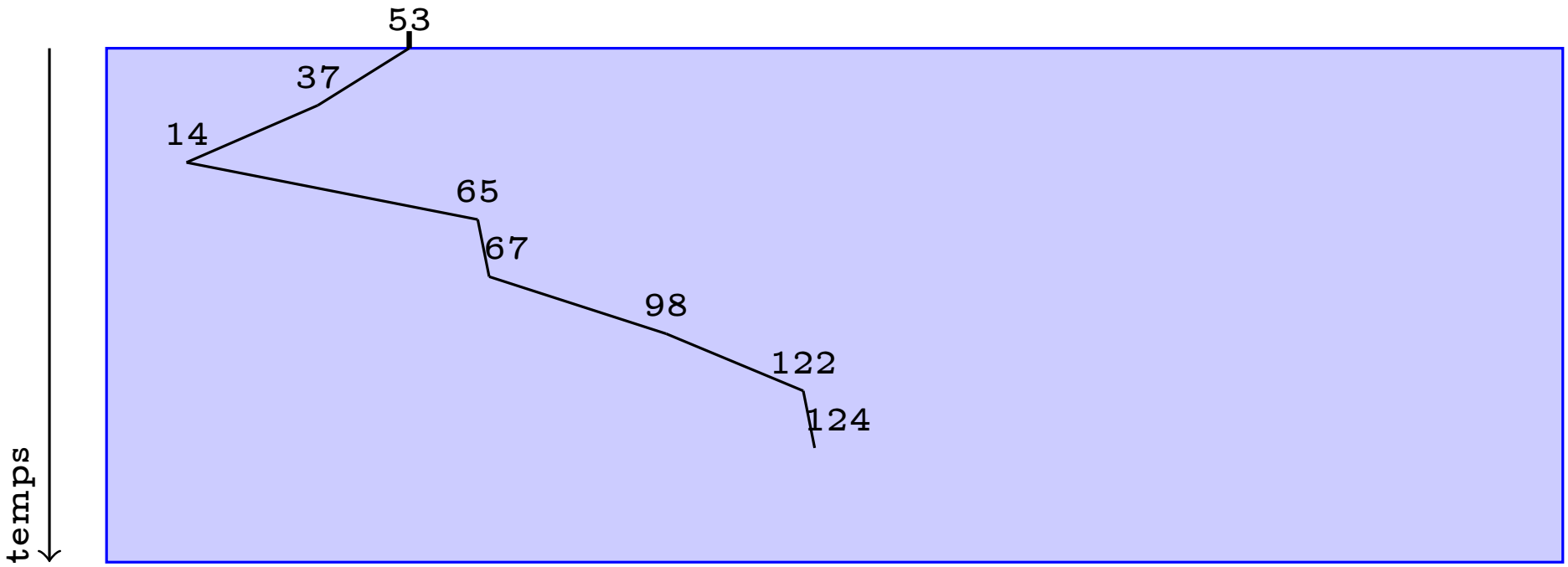


16+23+51+2+31+24+

Principe

Repartir lorsqu'on a atteint le plus petit cylindre demandé

Cylindre 53 + { 98, 183, 37, 122, 14, 124, 65, 67 }

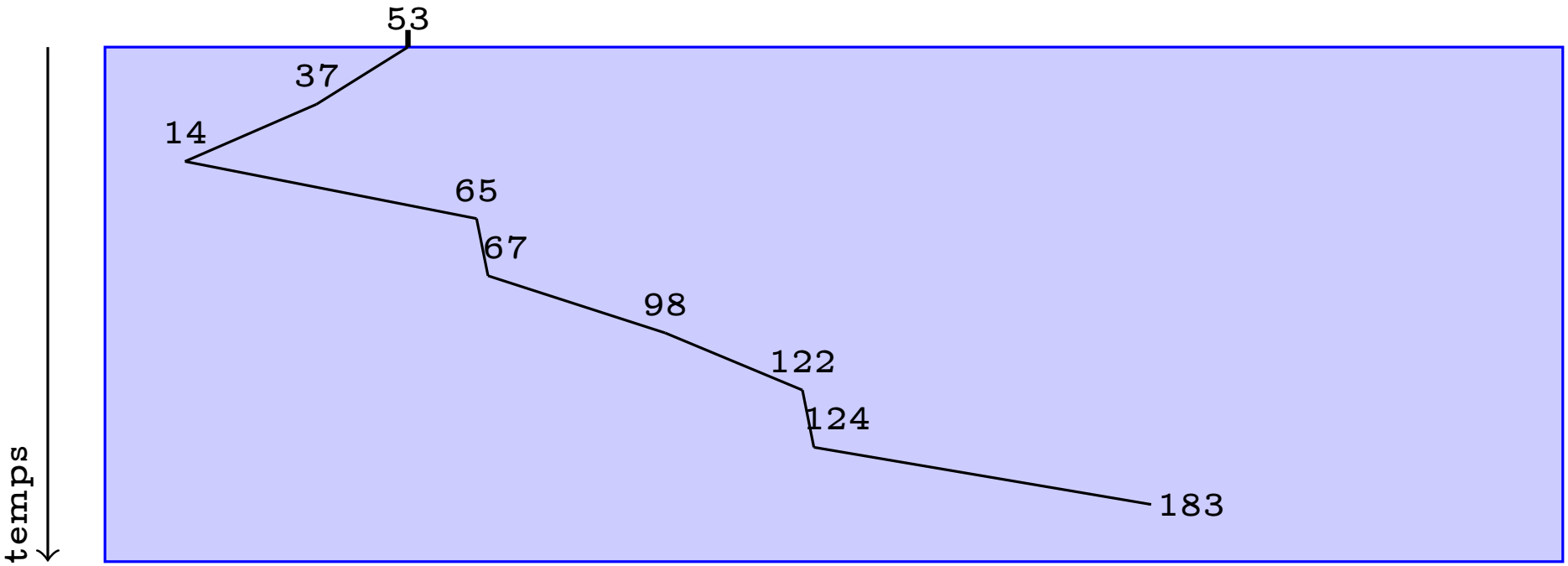


16+23+51+2+31+24+2+

Principe

Repartir lorsqu'on a atteint le plus petit cylindre demandé

Cylindre 53 + { 98, 183, 37, 122, 14, 124, 65, 67 }



16+23+51+2+31+24+2+59 = (53-14) + (183-14) = 208

C-Scan

Principe

Balayage circulaire : toujours dans le même sens

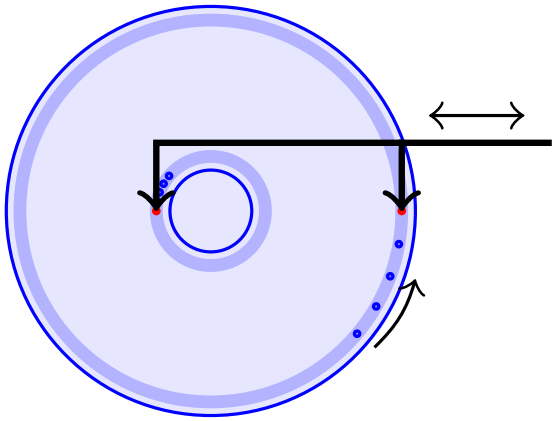
Principe

Balayage circulaire : toujours dans le même sens

Implémentation

Deux têtes de lecture espacées du rayon du disque

- Tête 1 lit cylindre 0 pendant que tête 2 lit le secteur n



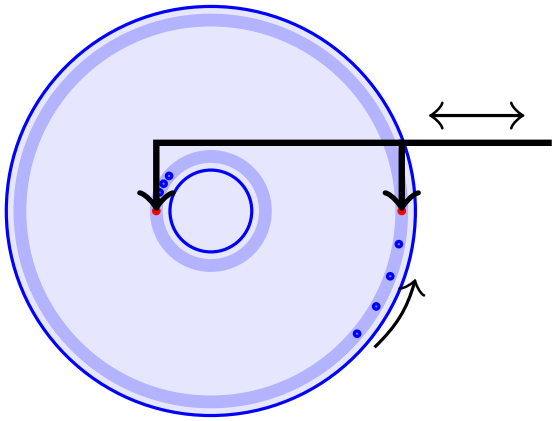
Principe

Balayage circulaire : toujours dans le même sens

Implémentation

Deux têtes de lecture espacées du rayon du disque

- Tête 1 lit cylindre 0 pendant que tête 2 lit le secteur n



- Extérieur $\longrightarrow$ intérieur

Tête 1 balaye de 0 à n-1

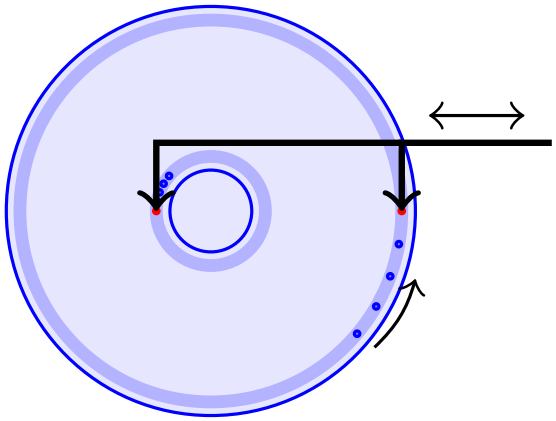
Principe

Balayage circulaire : toujours dans le même sens

Implémentation

Deux têtes de lecture espacées du rayon du disque

- Tête 1 lit cylindre 0 pendant que tête 2 lit le secteur n



- Extérieur $\longrightarrow$ intérieur

  Tête 1 balaye de 0 à n-1

- Tête 1 = secteur n

  $\longrightarrow$ Tête 2 = secteur 0

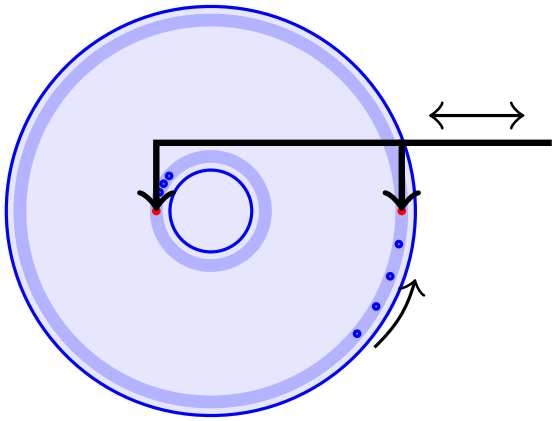
Principe

Balayage circulaire : toujours dans le même sens

Implémentation

Deux têtes de lecture espacées du rayon du disque

- Tête 1 lit cylindre 0 pendant que tête 2 lit le secteur n



- Extérieur $\longrightarrow$ intérieur

  Tête 1 balaye de 0 à n-1

- Tête 1 = secteur n

  $\longrightarrow$ Tête 2 = secteur 0

- Intérieur $\longrightarrow$ extérieur

  Tête 2 balaye de 0 à n-1

C-Scan

Principe

Balayage circulaire (ici, descendant)

Principe

Balayage circulaire (ici, descendant)

Cylindre 53 + { 98, 183, 37, 122, 14, 124, 65, 67 }

Principe

Balayage circulaire (ici, descendant)

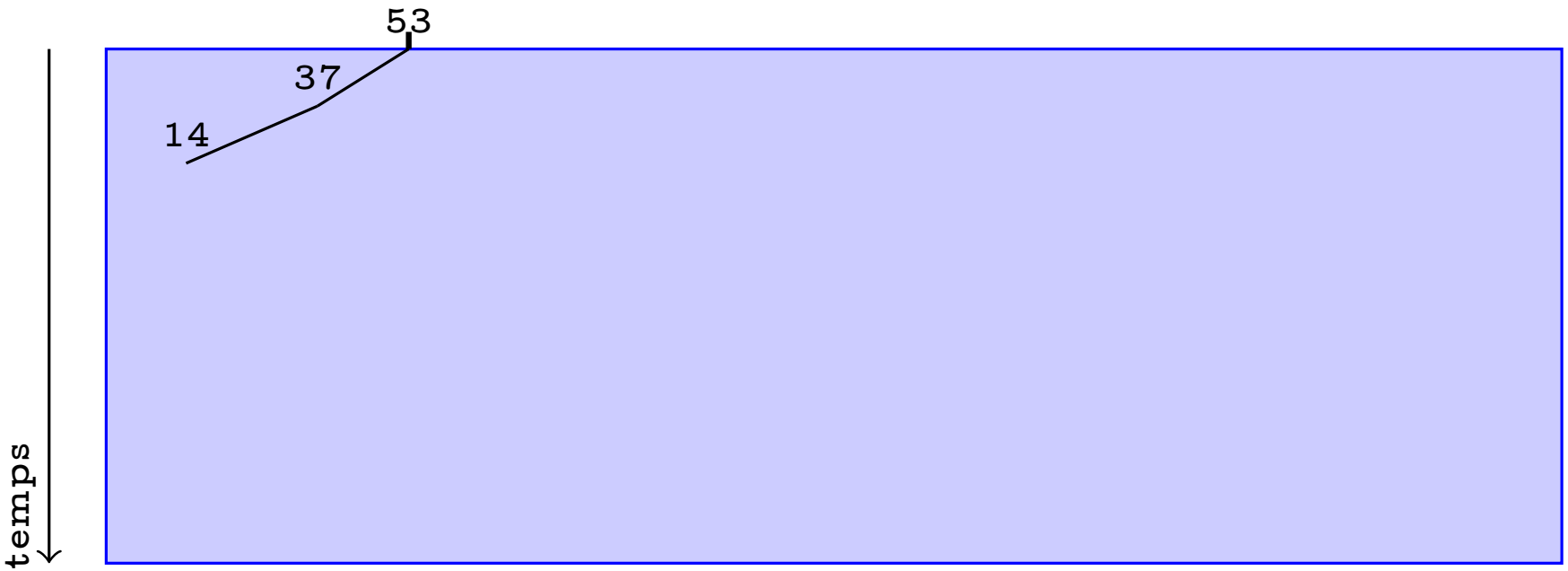
Cylindre 53 + { 98, 183, 37, 122, 14, 124, 65, 67 }

16+

Principe

Balayage circulaire (ici, descendant)

Cylindre 53 + { 98, 183, 37, 122, 14, 124, 65, 67 }

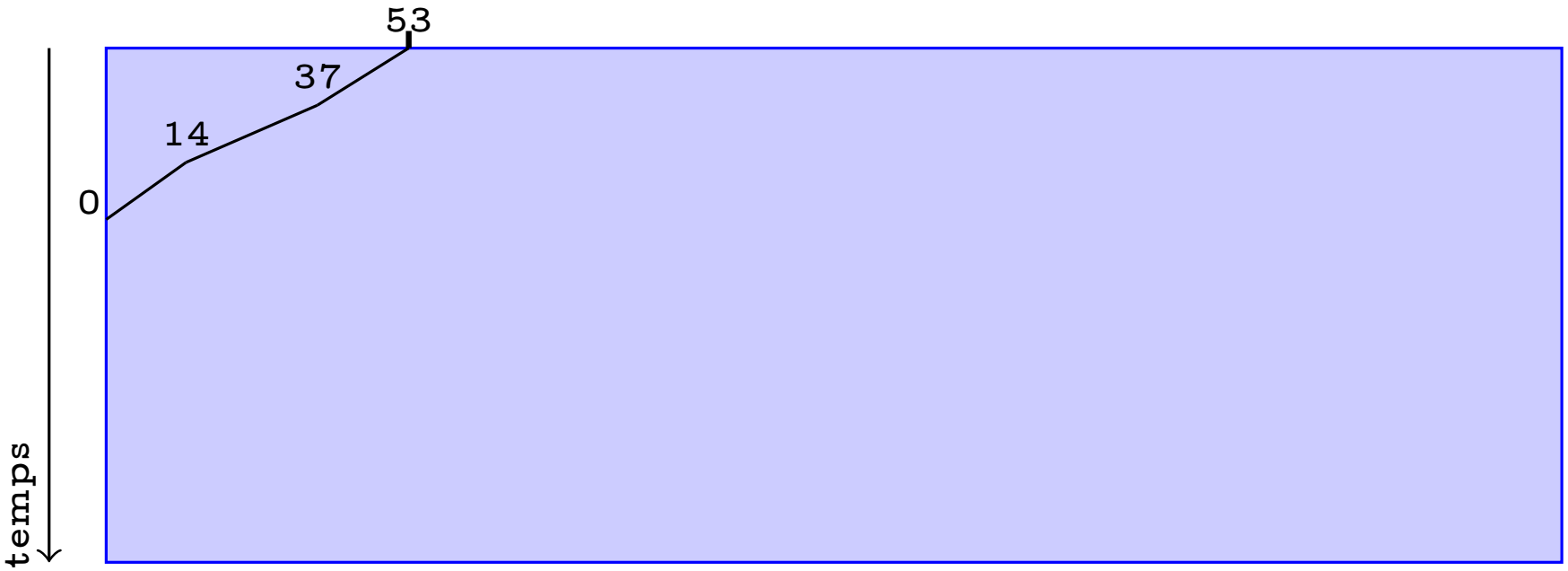


16+23+

Principe

Balayage circulaire (ici, descendant)

Cylindre 53 + { 98, 183, 37, 122, 14, 124, 65, 67 }

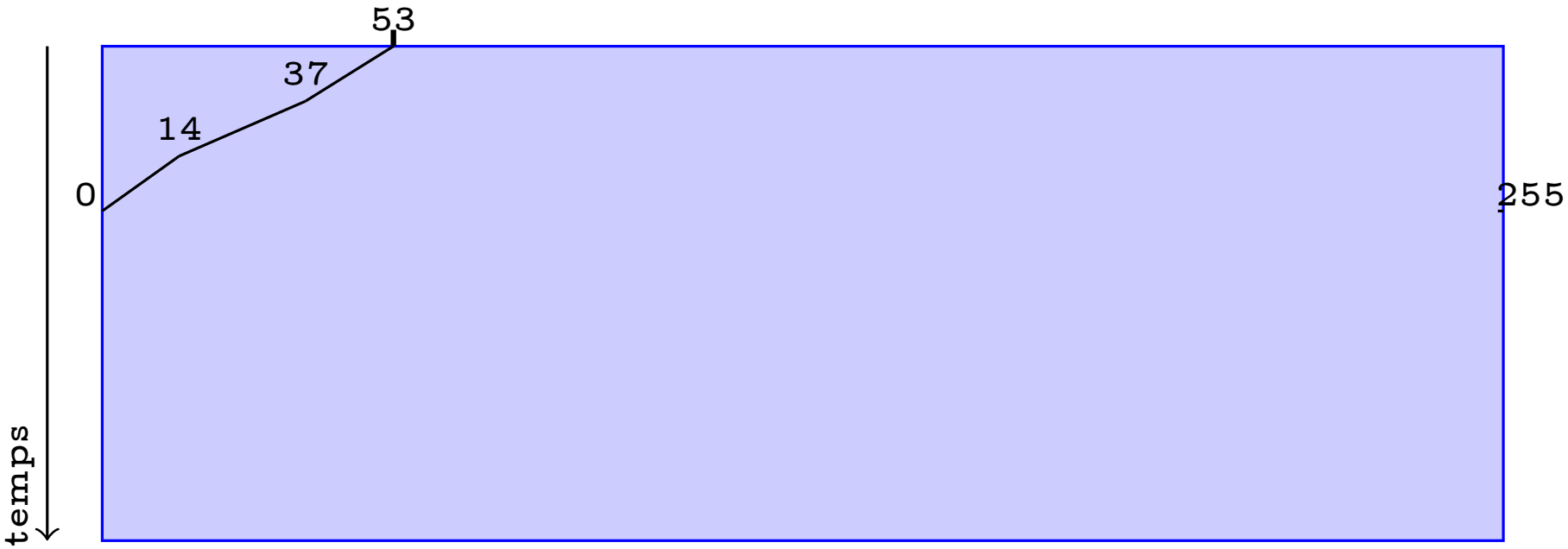


16+23+14+

Principe

Balayage circulaire (ici, descendant)

Cylindre 53 + { 98, 183, 37, 122, 14, 124, 65, 67 }



16+23+14+

Principe

Balayage circulaire (ici, descendant)

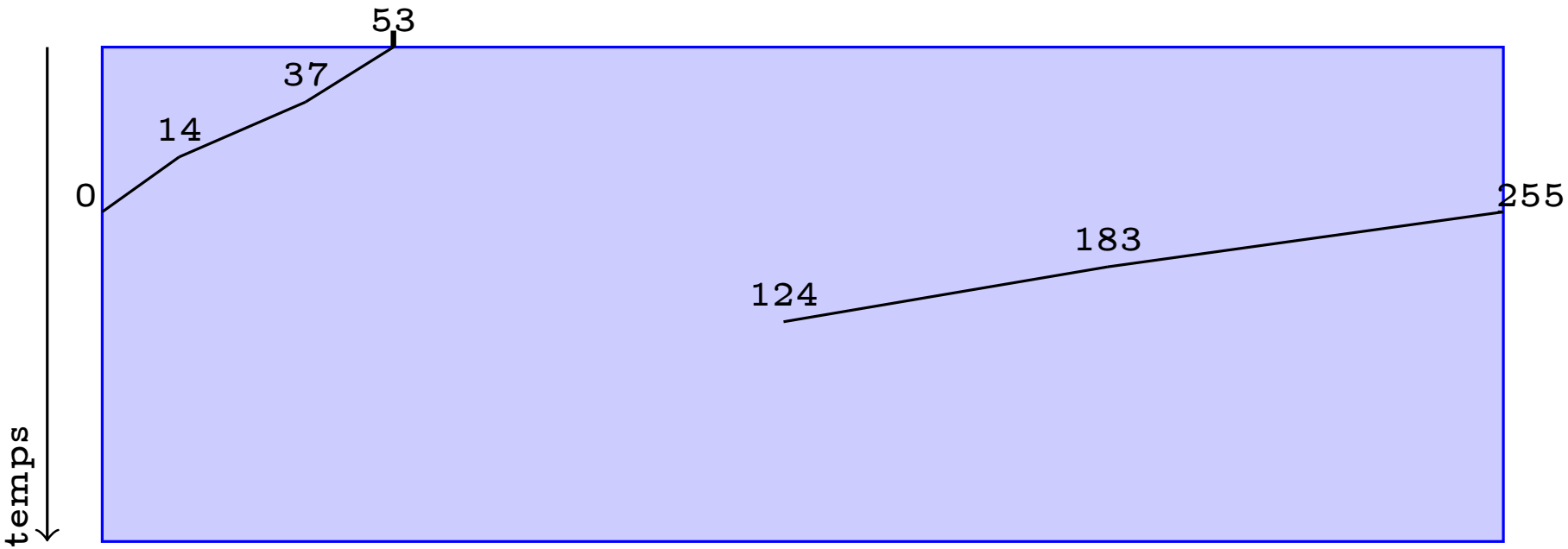
Cylindre 53 + { 98, 183, 37, 122, 14, 124, 65, 67 }

16+23+14+72+

Principe

Balayage circulaire (ici, descendant)

Cylindre 53 + { 98, 183, 37, 122, 14, 124, 65, 67 }

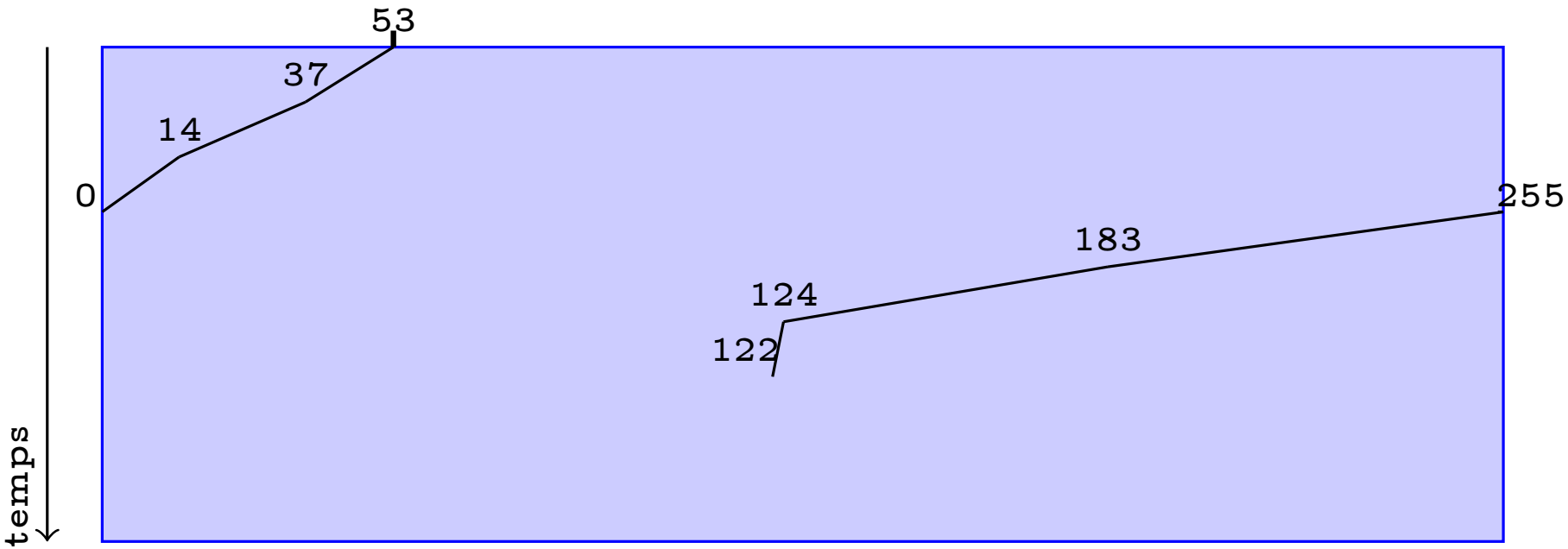


16+23+14+72+59+

Principe

Balayage circulaire (ici, descendant)

Cylindre 53 + { 98, 183, 37, 122, 14, 124, 65, 67 }

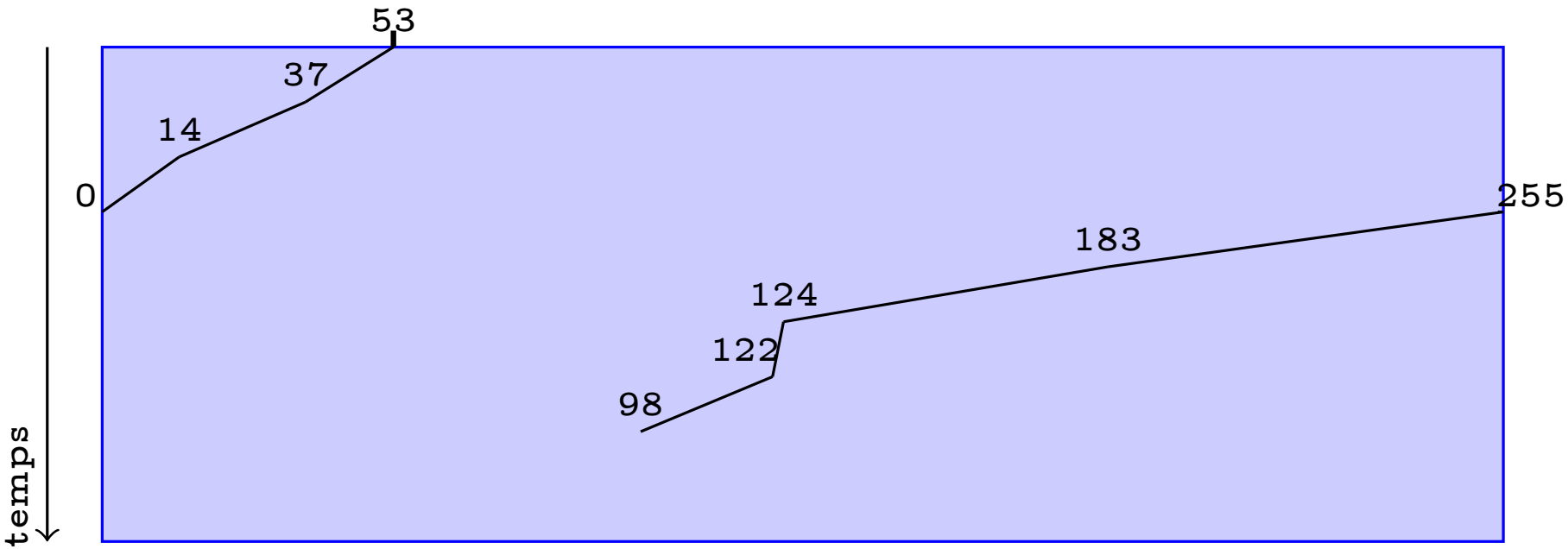


16+23+14+72+59+2+

Principe

Balayage circulaire (ici, descendant)

Cylindre 53 + { 98, 183, 37, 122, 14, 124, 65, 67 }

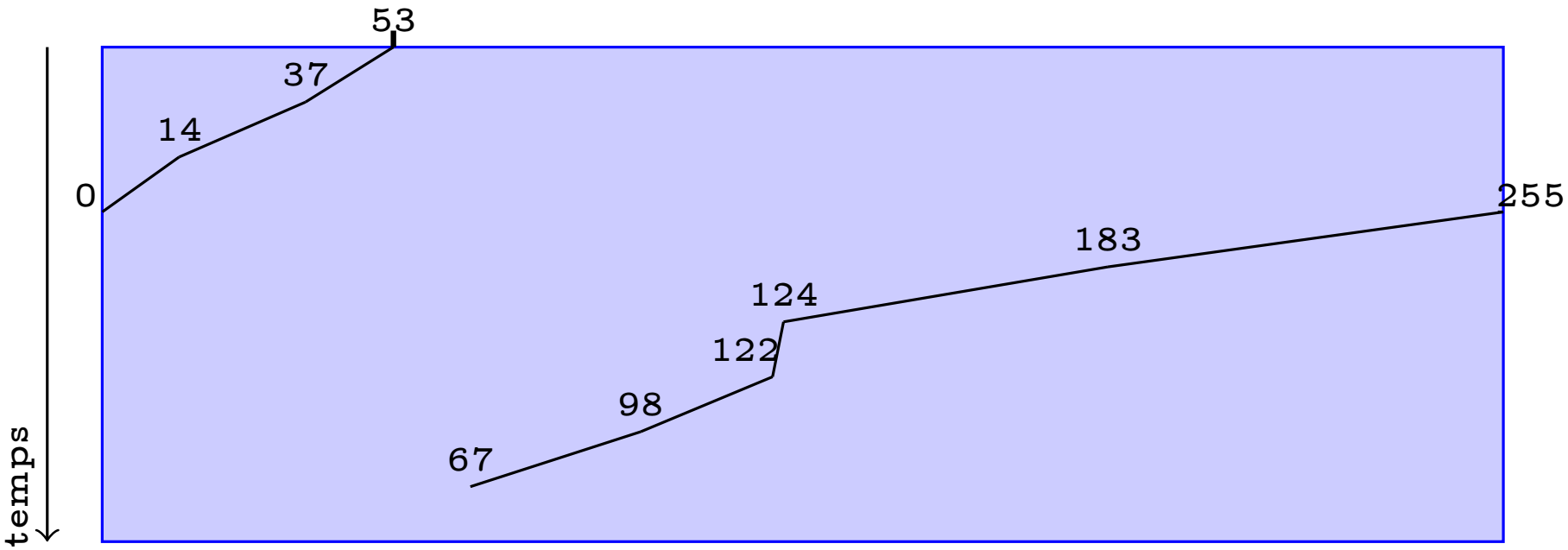


16+23+14+72+59+2+24+

Principe

Balayage circulaire (ici, descendant)

Cylindre 53 + { 98, 183, 37, 122, 14, 124, 65, 67 }

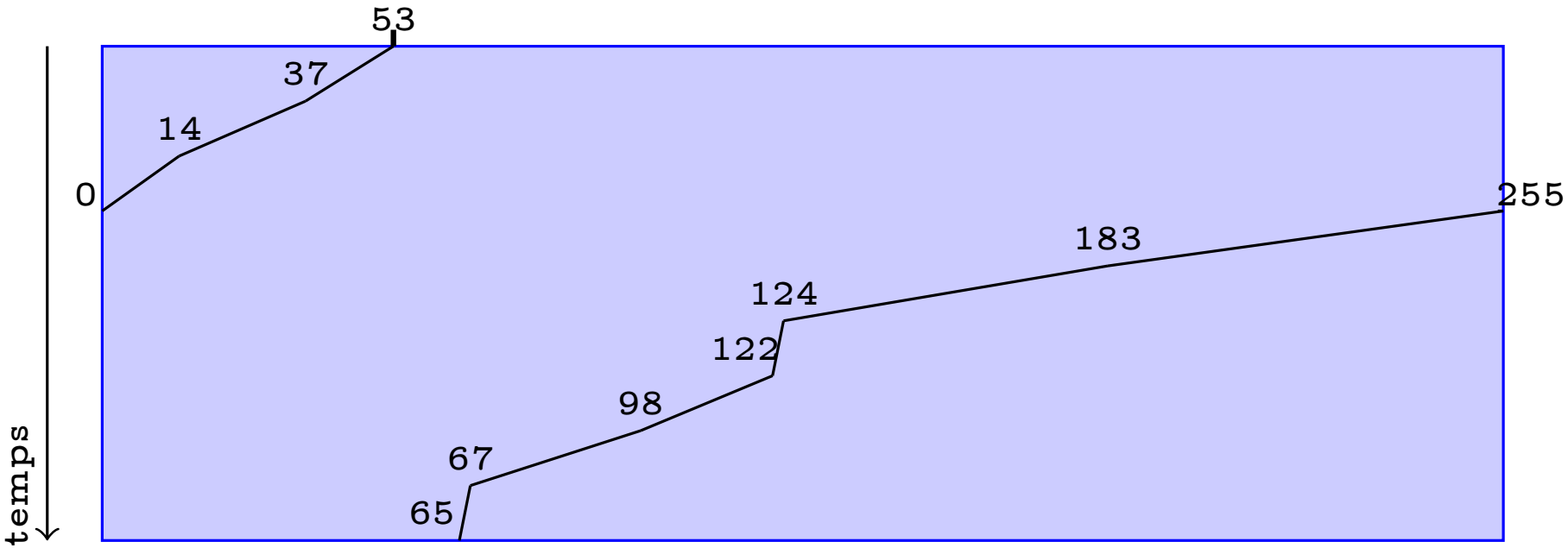


16+23+14+72+59+2+24+31+

Principe

Balayage circulaire (ici, descendant)

Cylindre 53 + { 98, 183, 37, 122, 14, 124, 65, 67 }



16+23+14+72+59+2+24+31+2 = 243

C-Look

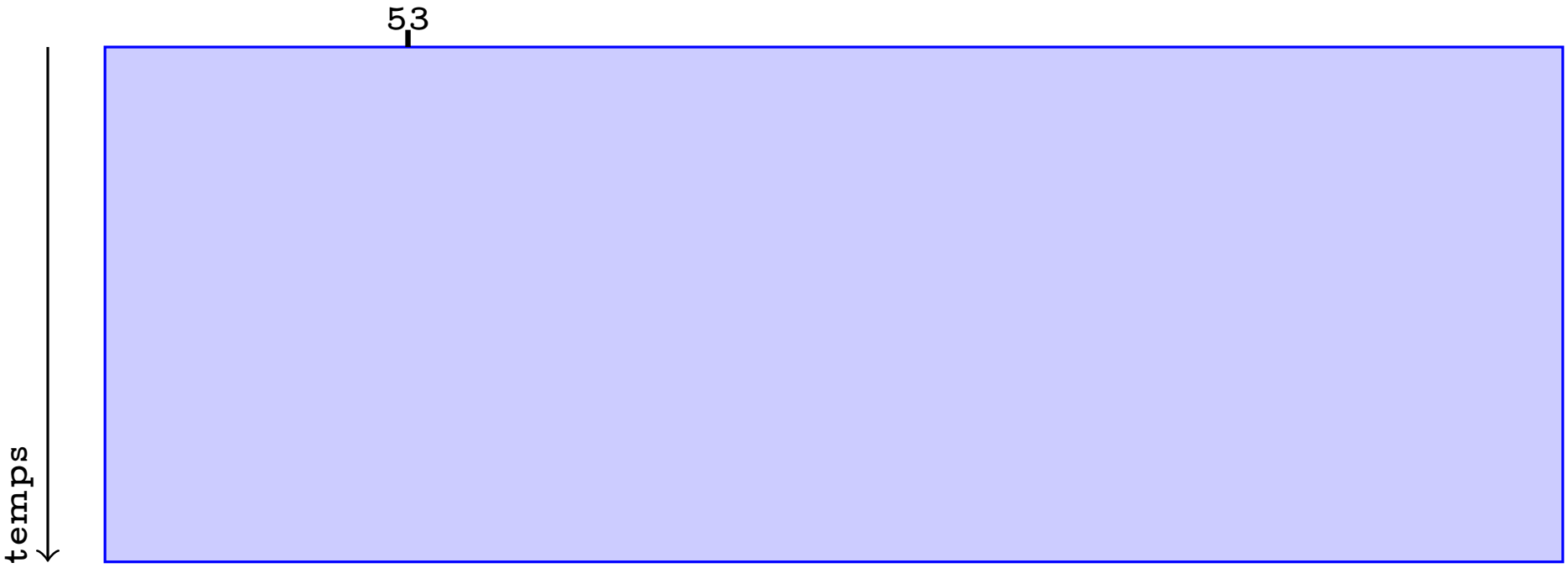
Principe

Même principe sans aller jusqu'au bord

Principe

Même principe sans aller jusqu'au bord

Cylindre 53 + { 98, 183, 37, 122, 14, 124, 65, 67 }



Principe

Même principe sans aller jusqu'au bord

Cylindre 53 + { 98, 183, 37, 122, 14, 124, 65, 67 }

16+

Principe

Même principe sans aller jusqu'au bord

Cylindre 53 + { 98, 183, 37, 122, 14, 124, 65, 67 }

16+23+

Principe

Même principe sans aller jusqu'au bord

Cylindre 53 + { 98, 183, 37, 122, 14, 124, 65, 67 }

16+23+

Principe

Même principe sans aller jusqu'au bord

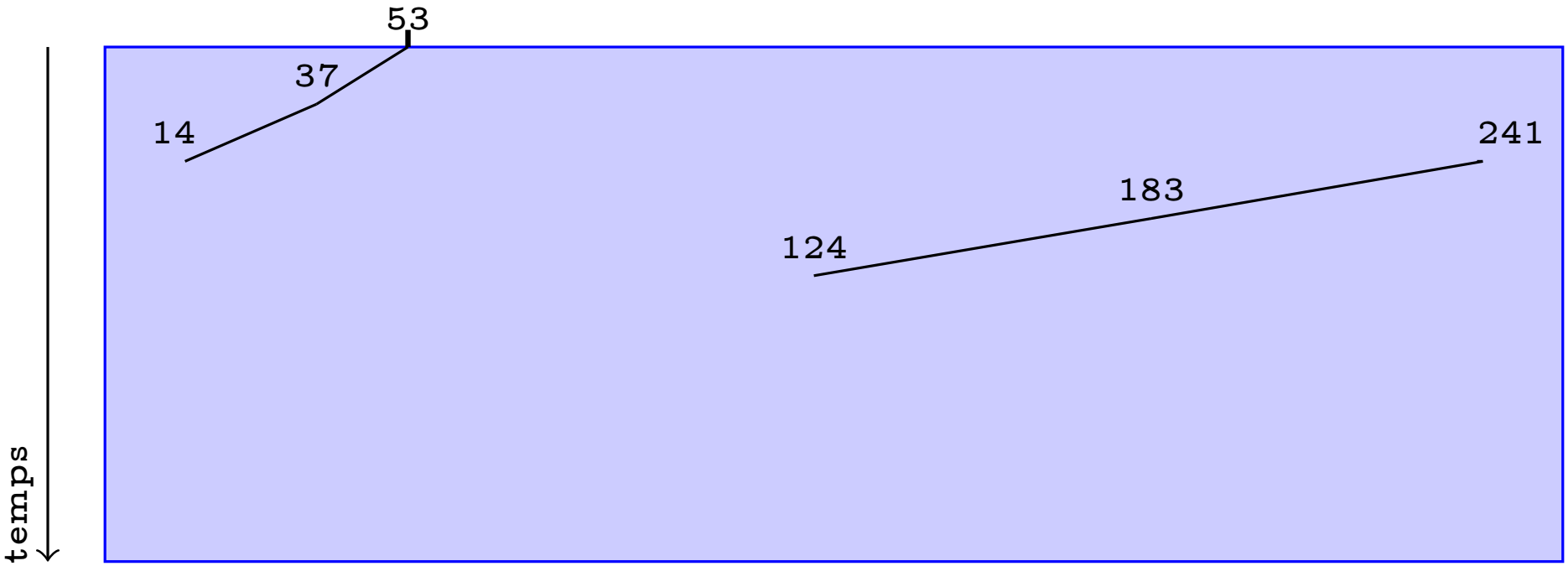
Cylindre 53 + { 98, 183, 37, 122, 14, 124, 65, 67 }

16+23+58+

Principe

Même principe sans aller jusqu'au bord

Cylindre 53 + { 98, 183, 37, 122, 14, 124, 65, 67 }

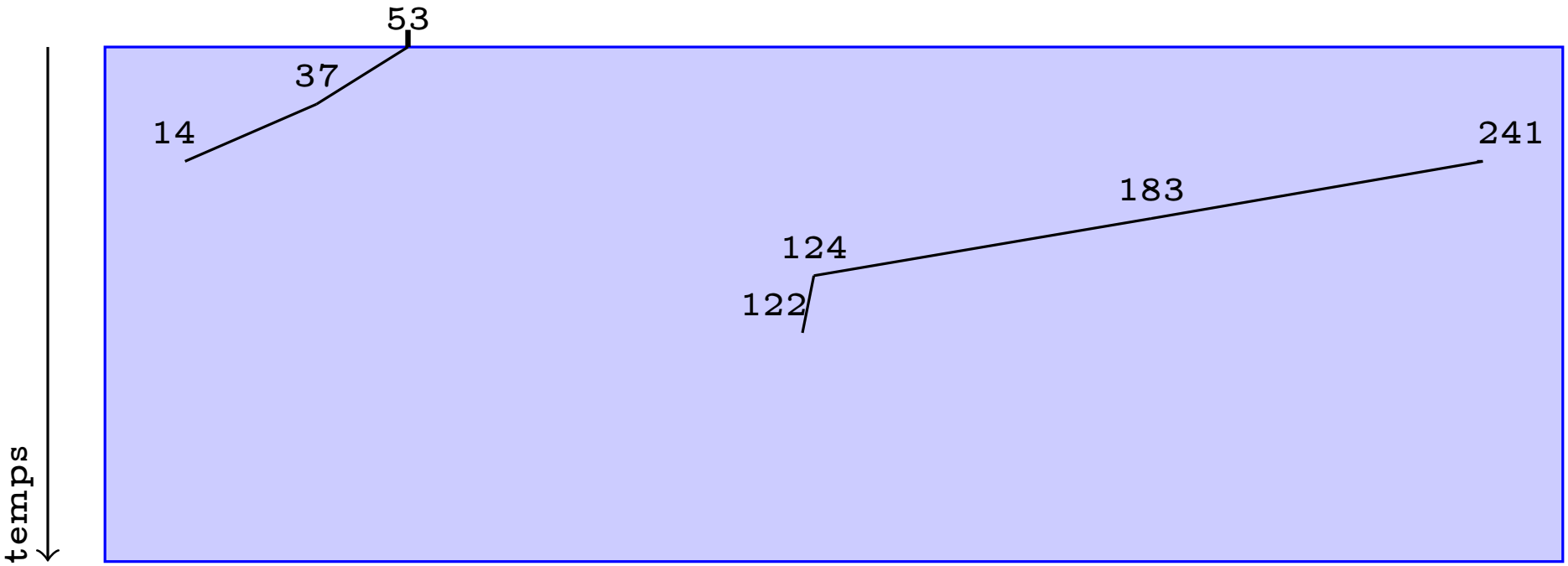


16+23+58+59+

Principe

Même principe sans aller jusqu'au bord

Cylindre 53 + { 98, 183, 37, 122, 14, 124, 65, 67 }

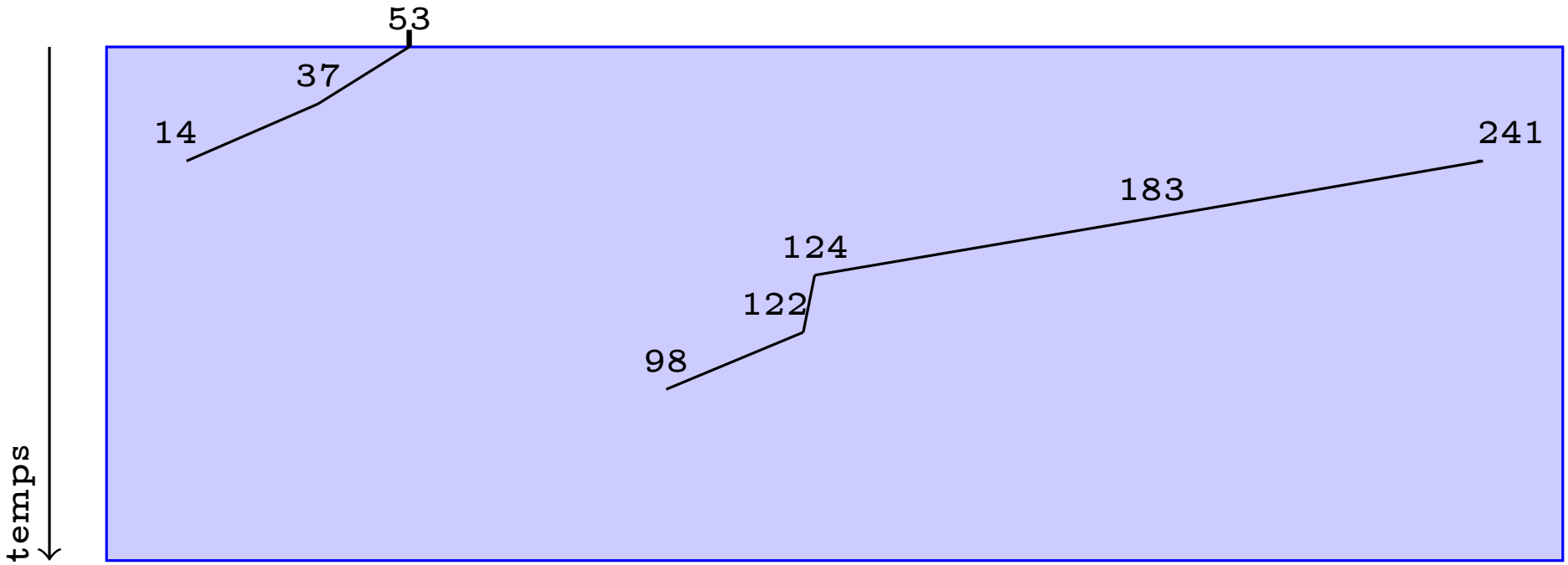


16+23+58+59+2+

Principe

Même principe sans aller jusqu'au bord

Cylindre 53 + { 98, 183, 37, 122, 14, 124, 65, 67 }

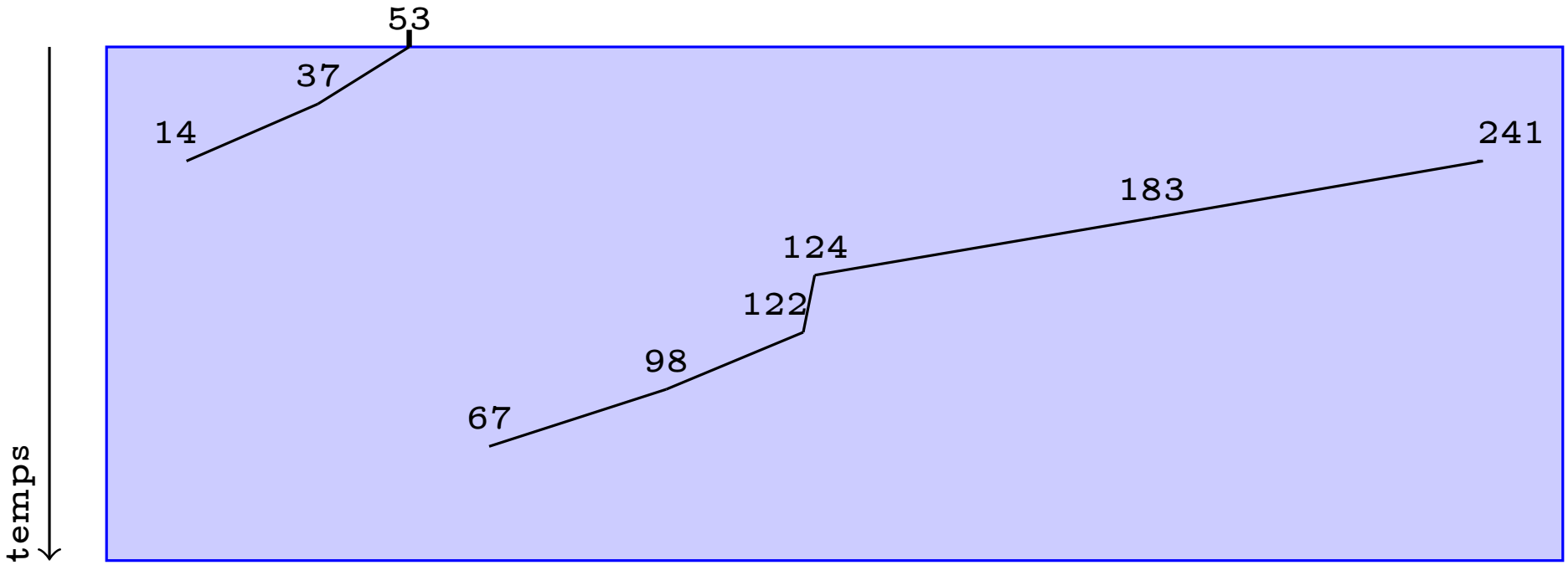


16+23+58+59+2+24+

Principe

Même principe sans aller jusqu'au bord

Cylindre 53 + { 98, 183, 37, 122, 14, 124, 65, 67 }

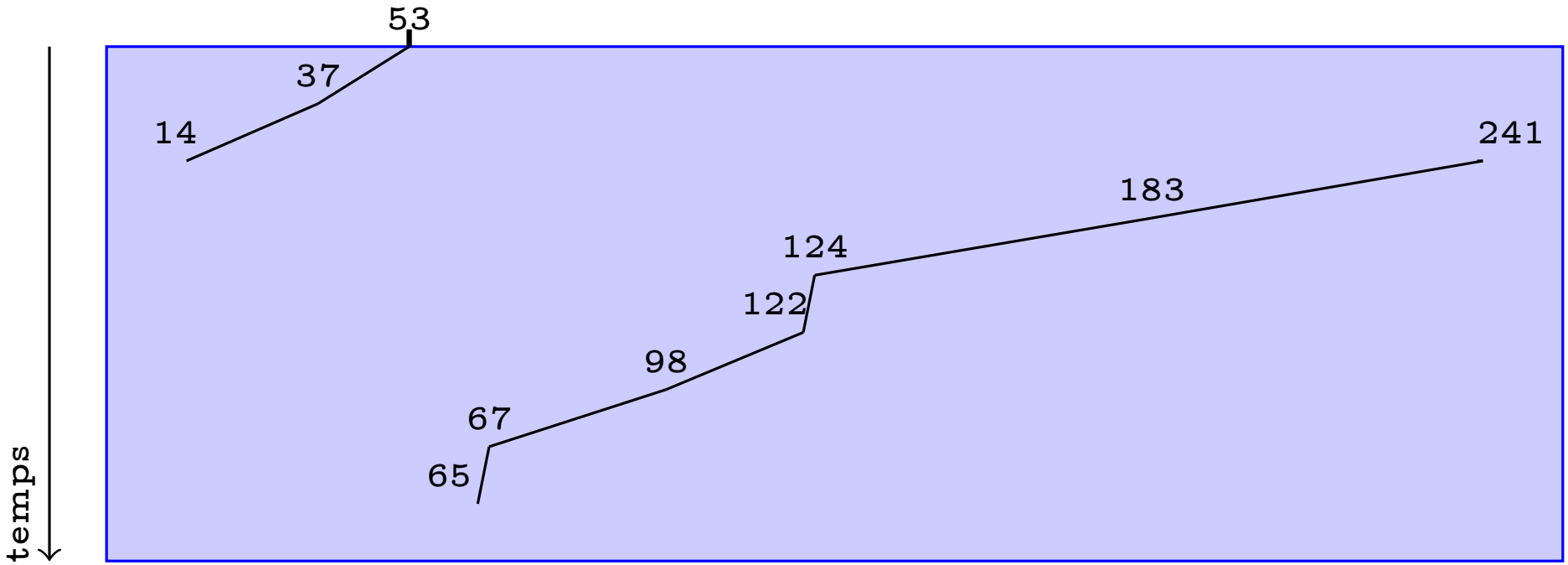


16+23+58+59+2+24+31+

Principe

Même principe sans aller jusqu'au bord

Cylindre 53 + { 98, 183, 37, 122, 14, 124, 65, 67 }



16+23+58+59+2+24+31+2 = (53-14) + (241-65) = 215

En pratique

- La plupart des OS utilisent SSTF
- Forte charge d'E/S : C-LOOK

En pratique

- La plupart des OS utilisent SSTF
- Forte charge d'E/S : C-LOOK

Ordonnancement optimal

Possible à calculer à chaque pas de temps mais très coûteux

En pratique

- La plupart des OS utilisent SSTF
- Forte charge d'E/S : C-LOOK

Ordonnancement optimal

Possible à calculer à chaque pas de temps mais très coûteux

Rappel

Le temps de réponse dépend aussi de :

- La méthode d'allocation de fichiers
- La position des répertoires et des blocs d'index
- Le temps de rotation du disque
- La priorité au niveau OS (pagination *vs* E/S)

Bandes magnétiques

Stockage sur bande magnétique

Avantages

- ✓ Coût : très peu cher au Tio ! (facteur 5 à 10)
  - Stockage préventif de données brutes
  - Ex : 15 To pour 40 EUR en 2020
- ✓ Coût de maintenance quasi nul
- ✓ Durée de vie (sans perte de données) plus élevée

(CD = 5 ans, DD = 5 à 10 ans, Bande = 20 à 30 ans)

- ✓ Volume (ex : LTO-8 $\longrightarrow$ 150 Go/cm^3)

Stockage sur bande magnétique

Avantages

✓ Coût : très peu cher au Tio ! (facteur 5 à 10)
- Stockage préventif de données brutes
- Ex : 15 To pour 40 EUR en 2020

✓ Coût de maintenance quasi nul

✓ Durée de vie (sans perte de données) plus élevée

(CD = 5 ans, DD = 5 à 10 ans, Bande = 20 à 30 ans)

✓ Volume (ex : LTO-8 $\longrightarrow$ 150 Go/cm^3)

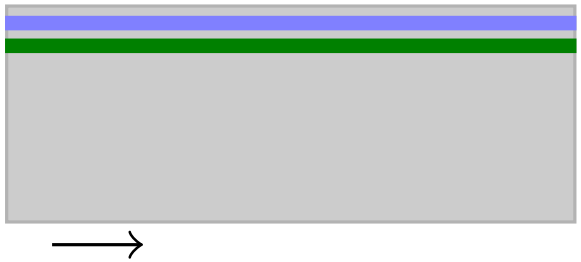
Inconvénients

X Temps d'accès aléatoire et très élevé

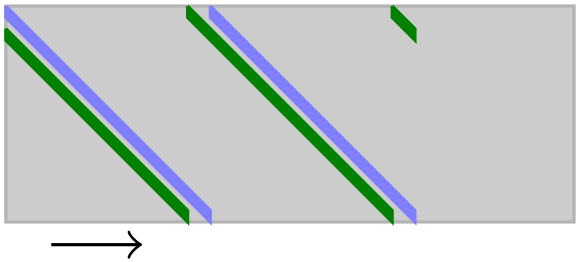
Réservé au stockage à froid ou application spécifiques

Pistes

- 9 pistes : 8 données + 1 parité



pistes linéaires



pistes hélicoïdales

Pistes

- 9 pistes : 8 données + 1 parité

Blocs

- Secteurs taille fixe
- Intervalles inter-enregistrement (IRG)
  - entre les secteurs
  - On s'arrête uniquement sur les IRG
  - Interruption sur secteur
    $\rightarrow$ rembobiner à l'IRG précédent

Pistes

- 9 pistes : 8 données + 1 parité

Blocs

- Secteurs taille fixe
- Intervalles inter-enregistrement (IRG)
  - entre les secteurs
  - On s'arrête uniquement sur les IRG
  - Interruption sur secteur
    $\rightarrow$ rembobiner à l'IRG précédent

Avantages

- Lecture possible dans les 2 sens
- Stockage en baies