Organisation et stockage des données

Plan

1. Introduction
2. Rappels (Fichiers et Disques)
3. Gestion des n-uplets
4. Echange disque-mémoire
5. Index
6. Conclusion

1. Introduction

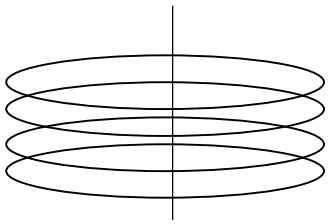
- Une BD est constituée d'un ensemble de relations ayant chacune une extension (un ensemble de n-uplets)
- Physiquement, ces n-uplets sont stockés dans un ou plusieurs fichiers qui peuvent être répartis sur un ou plusieurs sites
- Le format de stockage choisi doit permettre :
  - Une utilisation optimale de la mémoire
  - Un accès rapide
  - Des mises à jour faciles

1. Introduction

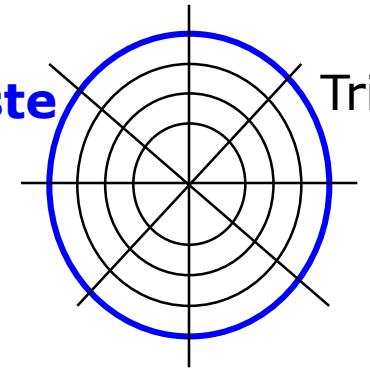
- Les n-uplets manipulés par une application doivent être préalablement transférés en mémoire centrale
- Ce transfert est réalisé via la zone «tampon » ou «cache » de la mémoire centrale
- Pour accéder rapidement à un ensemble de n-uplets vérifiant certaines conditions, toute BD relationnelle est munie d'index qui permettent d'associer chaque valeur d'un constituant à l'ensemble des n-uplets qui possèdent cette valeur pour ce constituant

2. Rappels (Fichiers et Disques)

- Un fichier est stocké sur un disque et peut s'étendre sur un ou plusieurs secteurs du disque
- Un disque est composé de plusieurs plateaux empilés «disk pack», présente une importante capacité de stockage, tourne à une vitesse très importante
- Après formatage, les disques sont divisés en cylindres et chaque cylindre en pistes et secteurs de même taille (512 octets ou multiple), ils forment l'unité de transfert UC/disque

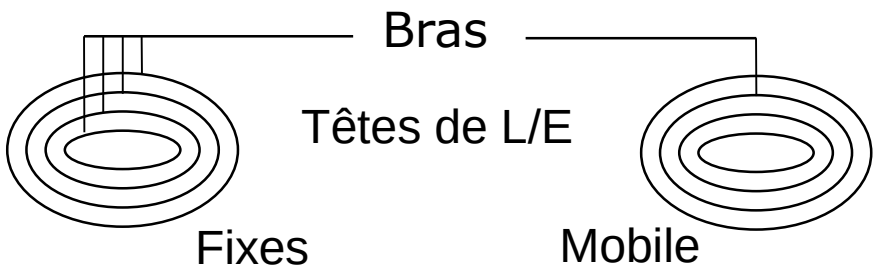


Disk pack



2. Rappels (Fichiers et Disques)

- Un disk pack de **n** disques présente **2n** faces
- Il existe des disques à tête unique (mobile) par face, et d'autres à têtes multiple (fixes) par face, il y aura autant de têtes qu'il y a de pistes



2. Rappels (Fichiers et Disques)

Caractéristiques d'un disque :

- Nombre de plateaux ou de faces utilisées
- Nature des têtes (fixes ou mobiles)
- Capacité de stockage
- Nombre de cylindres ou de pistes
- Taille d'un secteur
- Temps d'accès (caractéristique du lecteur), inclut
  - Le délai de rotation, pour que l'information soit disponible au niveau de la tête de L/E
  - Le délai de positionnement, sur la piste concernée
  - Le délai de transfert

2. Rappels (Fichiers et Disques)

- Un fichier possède un nom et est constitué d'une suite de **blocs**
- L'espace occupé par un fichier varie en fonction du temps, on distingue 4 phases :
  1. **Allocation initiale**
  2. **Expansion**: des blocs supplémentaires alloués au fur et à mesures des besoins.
  3. **Contraction**: les blocs qui ne sont plus utilisés sont restitués
  4. **Réorganisation**

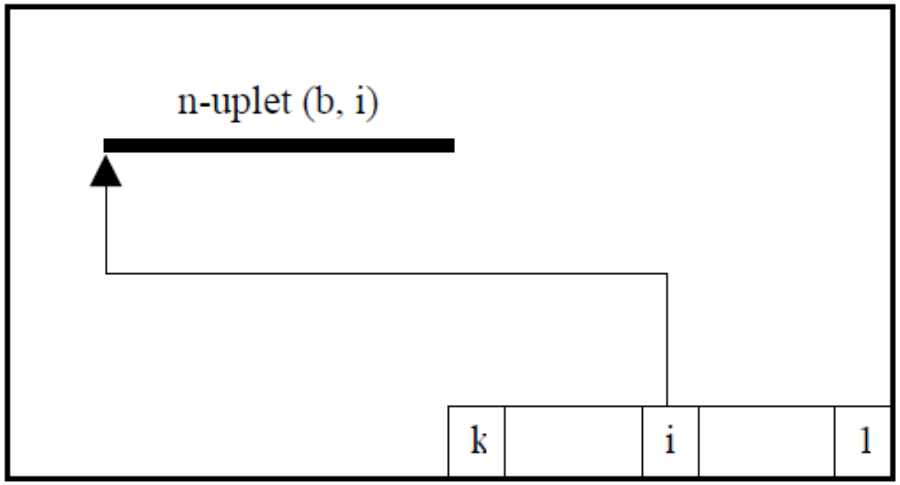
3. Gestion des n-uplets

- Les n-uplets d'une BD relationnelle sont rangés dans des pages
- Une **page** est stockée dans un bloc
- Nous supposerons qu'un bloc contient une seule page
- L'adresse physique d'un n-uplet est un quadruplet (s,f,b,n), elle désigne le **n** ième n-uplet de la page enregistrée dans le **b** ième bloc du fichier **f** du site **s**

3. 1. Organisation d'une page

Une page est découpée en deux espaces:

1. La zone des n-uplets: les n-uplets sont implantés dans un ordre quelconque
2. Un répertoire qui est implanté à la fin de la page. La i ième case de ce répertoire indique le déplacement dans la page du n-uplet de rang i



page stockée dans un bloc b et contenant k n-uplets

Cette organisation permet une certaine évolution de la taille des n-uplets d'une page sans qu'il soit nécessaire de modifier leurs adresses

3. 2. Identification d'un n-uplet

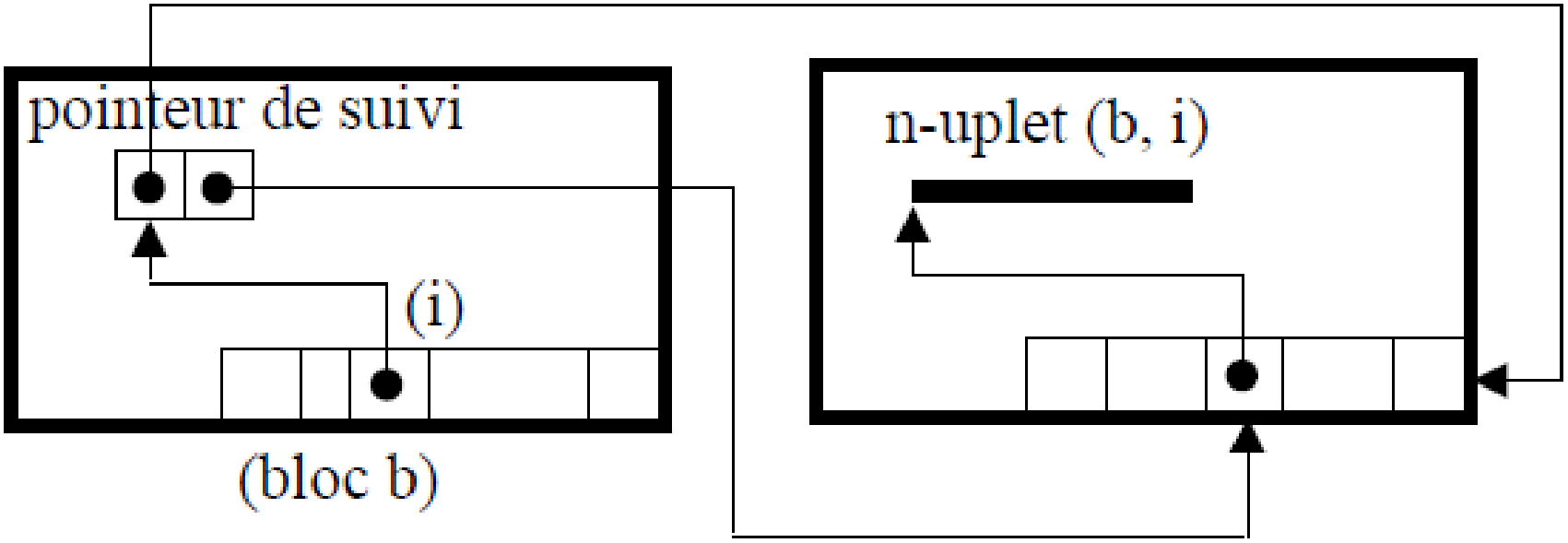
Il est nécessaire d'identifier chaque n-uplet afin :

- de les distinguer des autres n-uplets
- de pouvoir y accéder à partir des index

- Trois méthodes d'identification peuvent être utilisées :

1. Par **adressage direct** : l'identificateur d'un n-uplet est son adresse physique; cette méthode impose la mise en place d'un pointeur de suivi dans le cas où un n-uplet doit changer de place à la suite d'une mise à jour augmentant sa taille

3. 2. Identification d'un n-uplet



Identification d'un n-uplet par adressage direct

3. 2. Identification d'un n-uplet

2. Par **adressage indirect** : l'identificateur d'un n-uplet est un nombre entier attribué à sa création; cette méthode nécessite la mise en place d'une table de correspondance pour passer d'un identificateur d'un n-uplet à son adresse physique.
3. Par **la clé primaire** : un index primaire permet d'assurer la correspondance entre la clé et l'adresse physique d'un n-uplet

3. 2. Identification d'un n-uplet

Performance :

La 1ière méthode est la plus efficace :

- Elle nécessite 1 accès disque pour accéder à 1 n-uplet s'il n'y a pas de pointeur de suivi, et 2 accès disque s'il y en a un
- Elle ne nécessite pas de table de correspondance

3. 3. Représentation d'un n-uplet

Valeurs d'attributs

- Représentées sous leur forme externe ou bien sous les formes utilisées par les programmes d'application

n-uplets courts (taille inférieure à une page)

Les deux formats possibles sont :

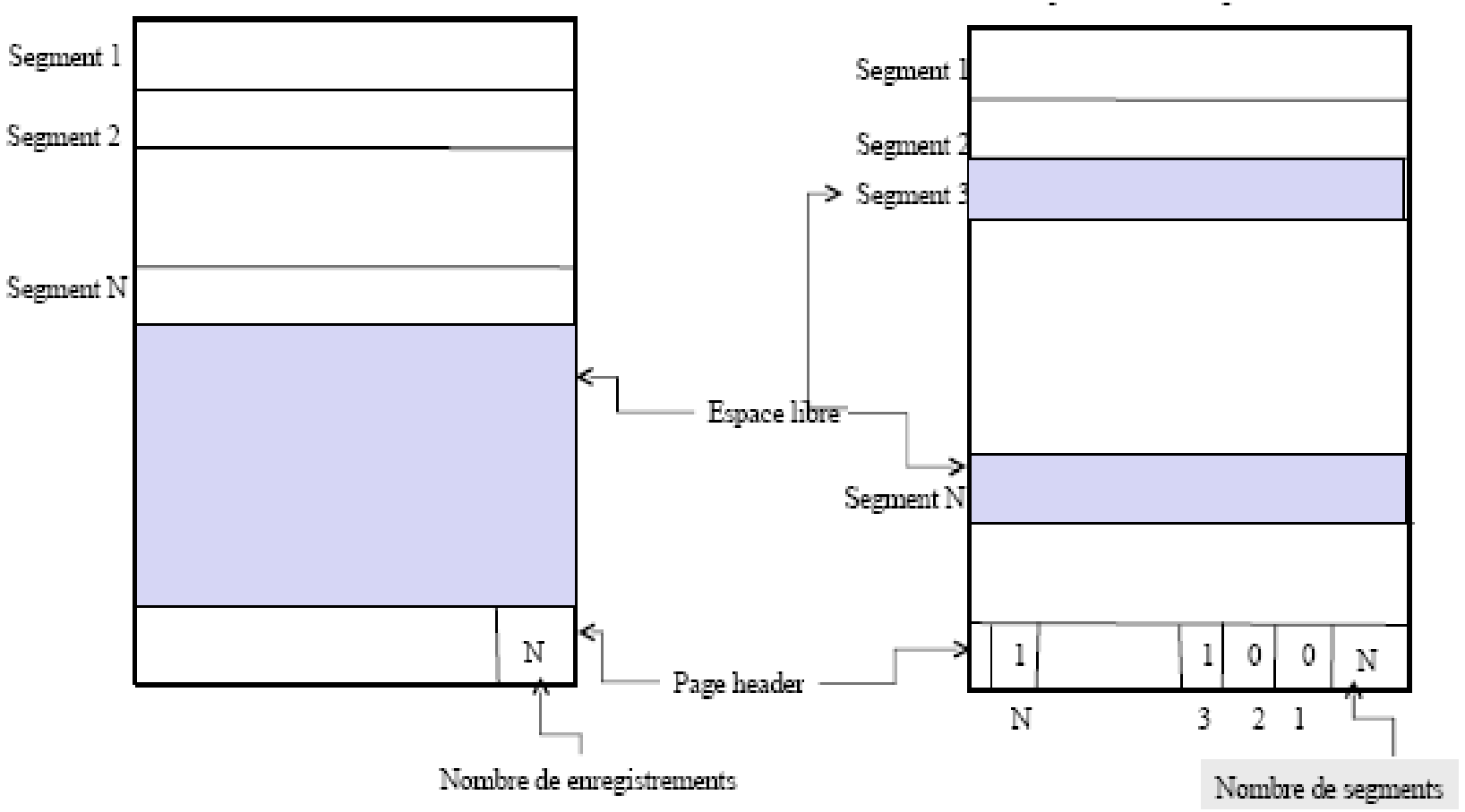
- Format fixe
  - Valeurs d'attributs enregistrées dans des champs de longueur fixe
- Format variable
  - Valeurs d'attributs précédés de leur longueur
  - Valeurs d'attributs précédés du nom de l'attribut

3. 3. Représentation d’un n-uplet

Le choix d’un format dépend de :

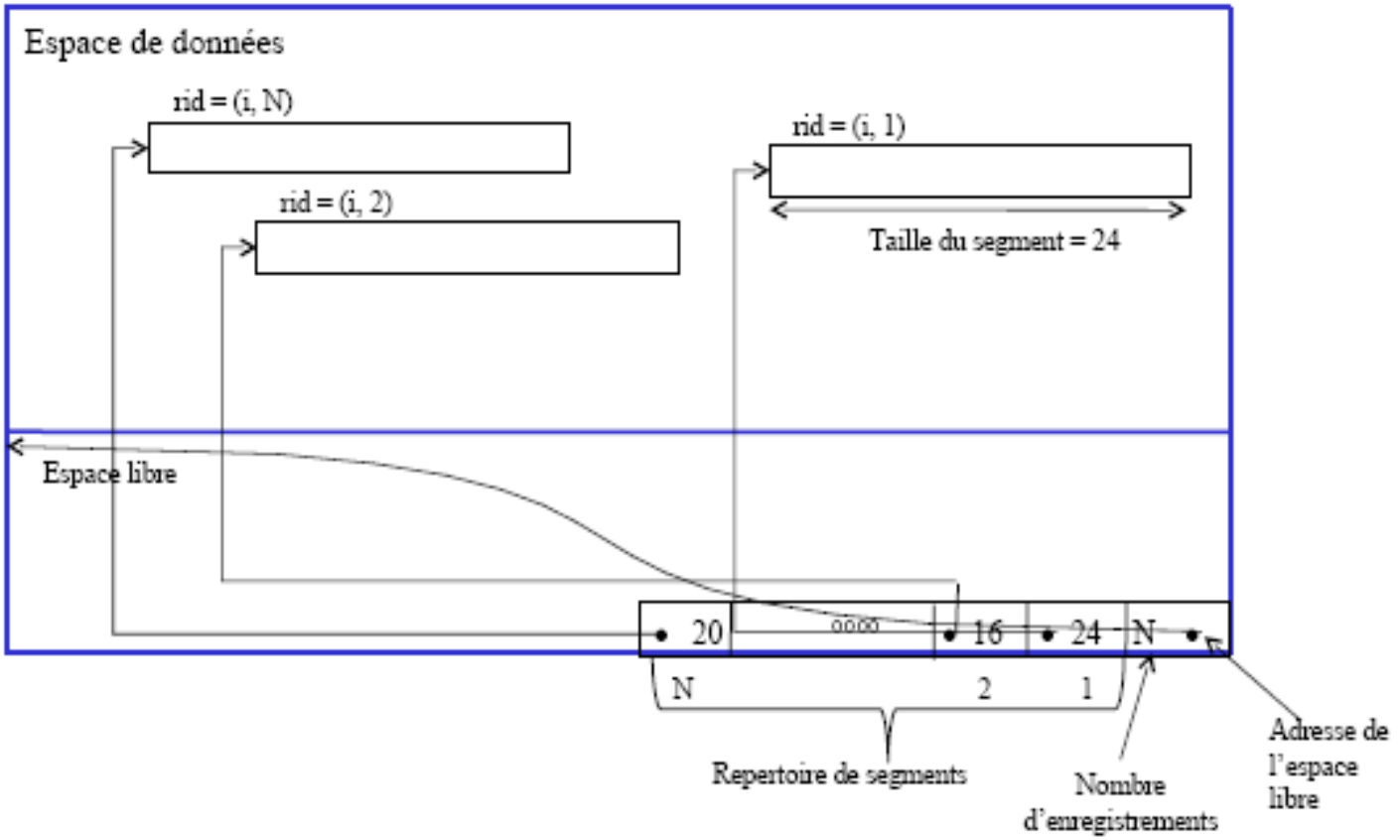
- la possibilité de représenter des attributs de longueur variable
- l’évolutivité des schémas de relation par ajout ou suppression d’attributs
- la rapidité d’accès aux valeurs d’attributs

3. 3. Représentation d'un n-uplet



Placement des n-uplets de taille fixe

3. 3. Représentation d'un n-uplet

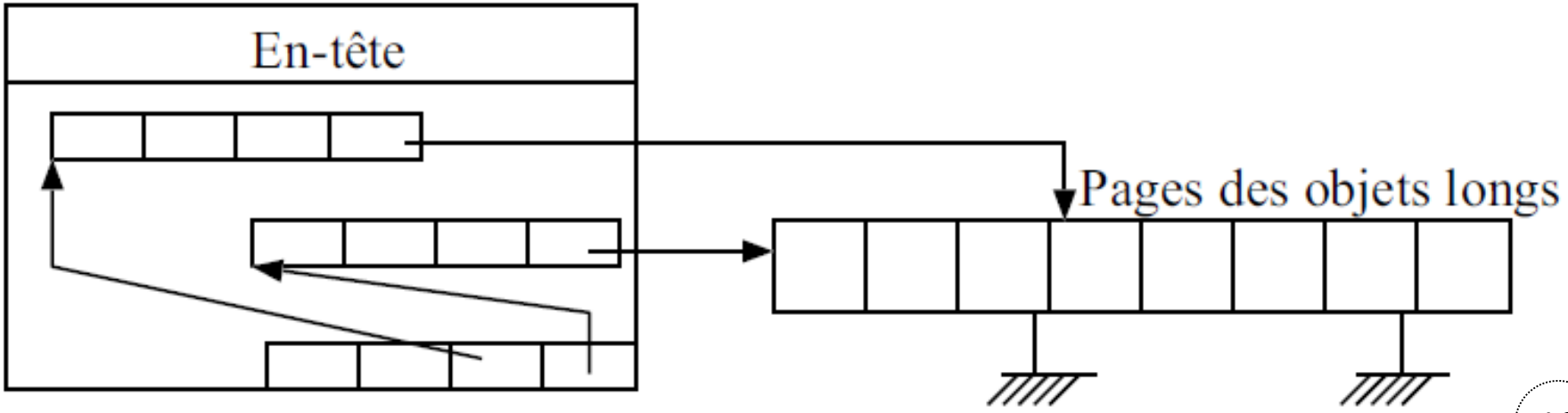


Placement des n-uplets de taille variable

3. 3. Représentation d'un n-uplet

n-uplets longs (taille supérieure à une page)

- Ils sont fragmentés en plusieurs pages
- Cette fragmentation peut être réalisée en séparant les attributs longs stockés chacun dans une page différente, des attributs courts stockés dans la même page.

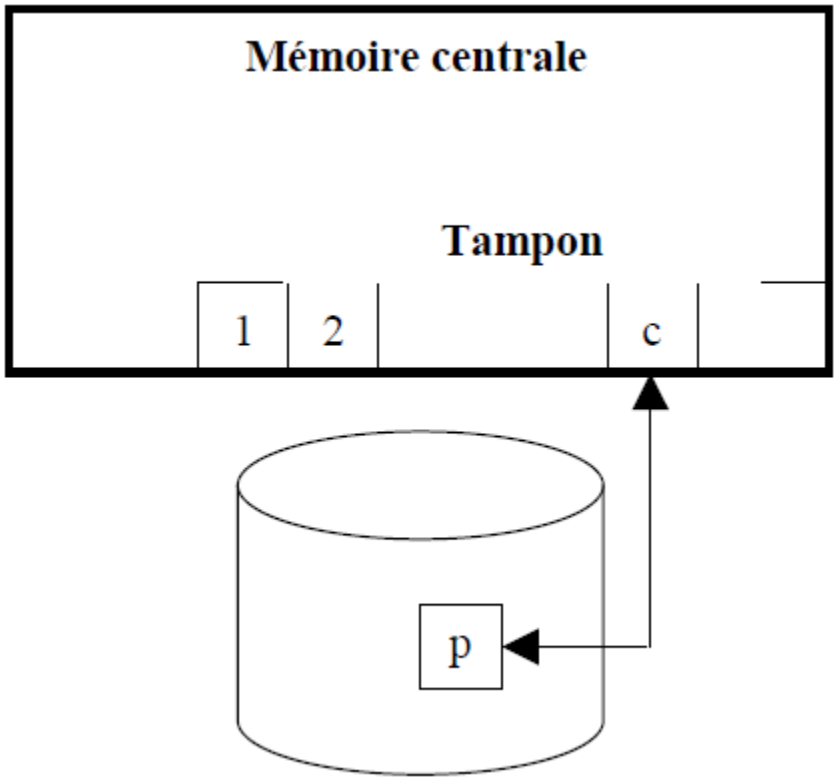


4. Échange disque-mémoire

- Réalisé au travers d'une zone **tampon** (buffer)
- Les principaux objectifs du gestionnaire de tampon sont :
  - Rendre les pages et les n-uplets qu'elles contiennent accessibles en mémoire centrale
  - Coordonner l'écriture des pages sur le disque en coopération avec le gestionnaire de transactions
  - Minimiser le nombre d'accès disques

4. 1. Structure d'un tampon

- Un tampon est formé d'une suite de **cases** contigües, chacune peut contenir une page.
- En mémoire centrale une page est donc repérée par le numéro de la case du tampon dans laquelle elle est rangée.



4. 2. Recherche d'une page

L'opération de recherche d'une page

- a pour argument l'adresse p de la page cherchée
- retourne le numéro c de la case du tampon dans laquelle cette page est rangée

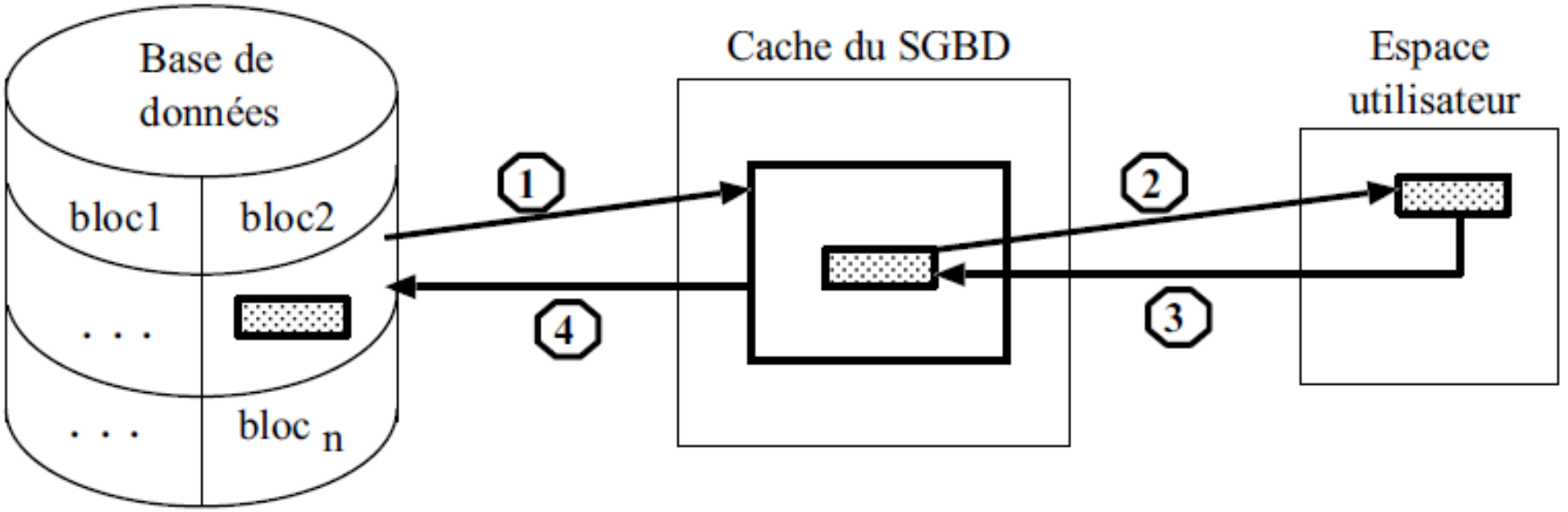
Cette opération se déroule selon les étapes suivantes:

1. Si la page p est dans la case c du tampon , retourner c; **on économise un accès disque.**
2. Si la page p n'est pas dans le tampon, il faut la lire sur le disque. On teste s'il existe une case libre pour la recevoir. Si oui, soit c cette case.

4. 2. Recherche d'une page

3. Sinon, il faut libérer une case et donc renvoyer une page du tampon sur le disque. Deux stratégies sont possibles : renvoyer la page la moins récemment utilisée ou la dernière page utilisée. Soit c la case contenant cette page.
4. Si la page à rejeter a été modifiée pendant son séjour en mémoire centrale, il faut la réécrire sur le disque. Ce travail est pris en compte par le gestionnaire de transactions.
5. Transférer la page p du disque dans la case c et retourner c.

4. 2. Recherche d'une page



Transfert et chargement des pages entre support de stockage permanant, mémoire centrale et espace utilisateur

5. Index

- Un **index** est un fichier qui contient un ensemble d'enregistrements formés d'une clé et d'une information associée qui permet un accès direct à l'enregistrement identifié par la clé.
- Dans une BD relationnelle, l'information associée est en général un identificateur de n-uplet ou bien un ensemble d'identificateurs de n-uplets, selon que l'index est primaire ou secondaire.

5. Index

- Un **index primaire** est construit sur la clé primaire d'une relation. Il donne accès à l'identificateur de n-uplet de cette clé.
- Un **index secondaire** est construit sur un attribut quelconque d'une relation. Il donne accès aux identificateurs des n-uplets pour une valeur de cet attribut.

5. 1. Typologie des index

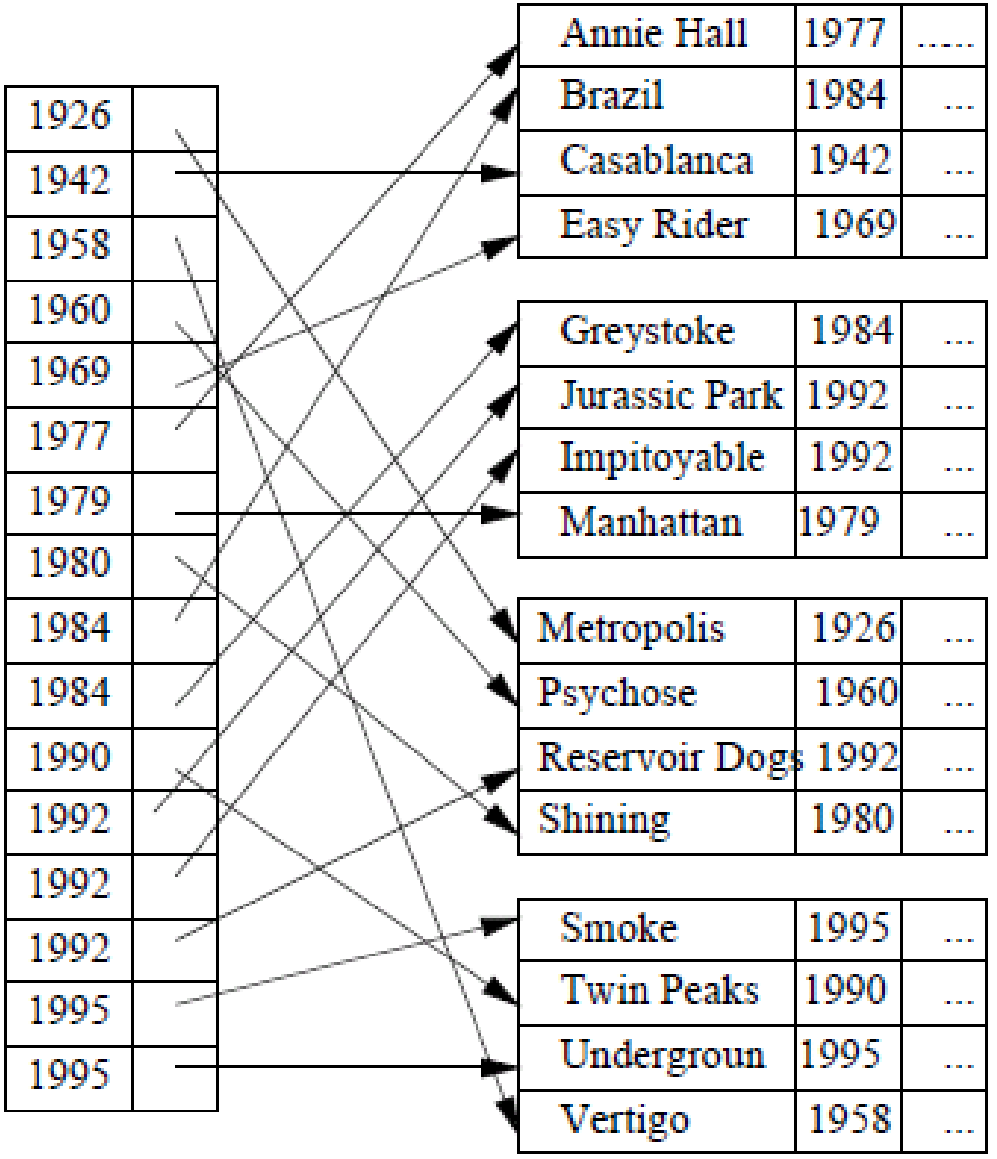
Index dense

- Utilisé pour indexer des données non triées sur la clé de recherche
- Une entrée dans l'index par valeur de la clé
- Entrée: <V*i*,P*i*> *où*
  - V*i :* valeur d'une clé d'index C
  - P*i :* tête d'une liste d'adresses de n-uplets
- Si clé d'index = clé de la relation : pas de chaînage
- Rangement des n-uplets pas nécessairement contigu
- Basé sur toutes les valeurs qui existent dans la relation en les associant à chaque enregistrement et non à chaque adresse de bloc

5. 1. Typologie des index

Exemple :

Index dense sur les années



5. 1. Typologie des index

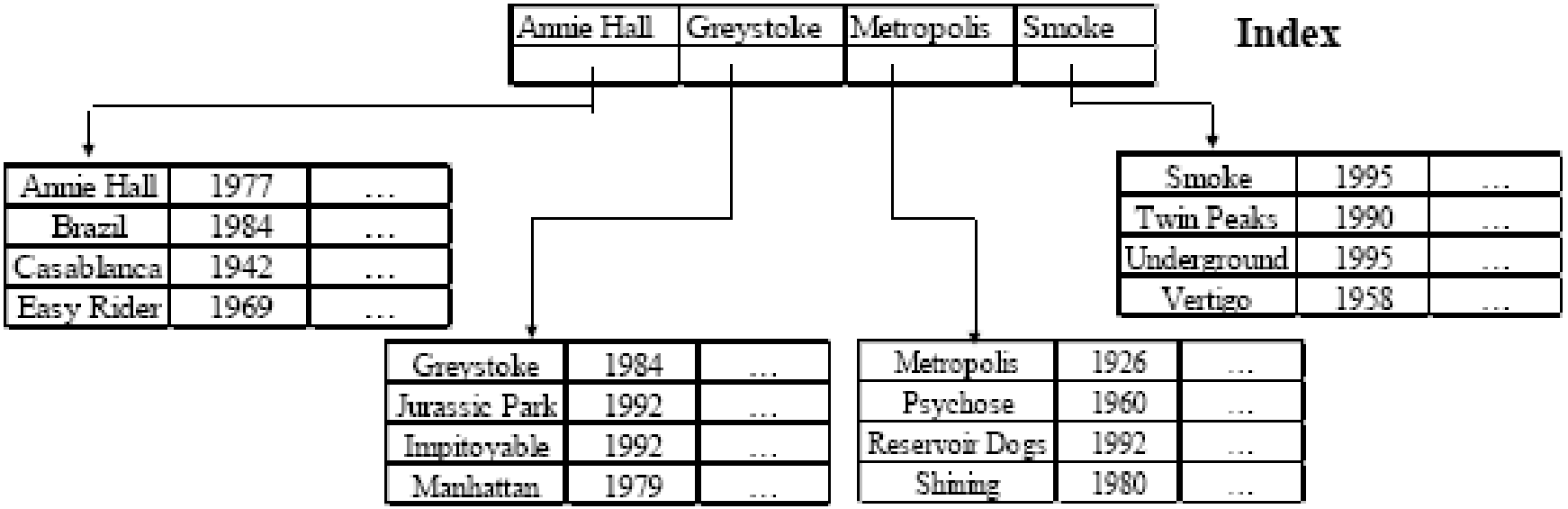
Index non dense (creux)

Moins d'entrées que de valeurs de la clé

L'index ne comprend qu'un enregistrement par bloc. Il est trié, on peut donc y accéder par dichotomie

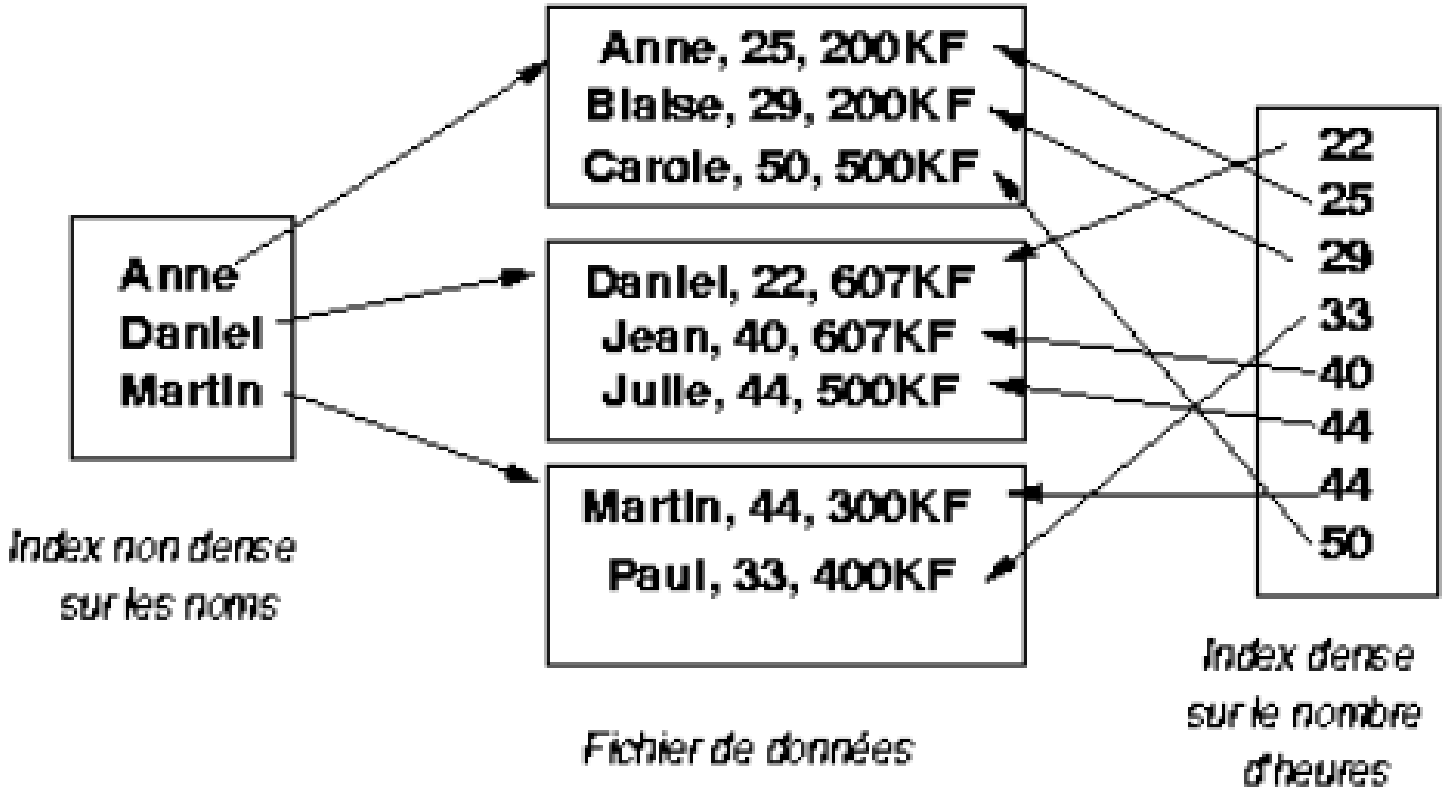
Très efficace pour les recherches

Problèmes : maintien de l'ordre suite à des insertions, destructions, maintien de la correspondance avec l'index



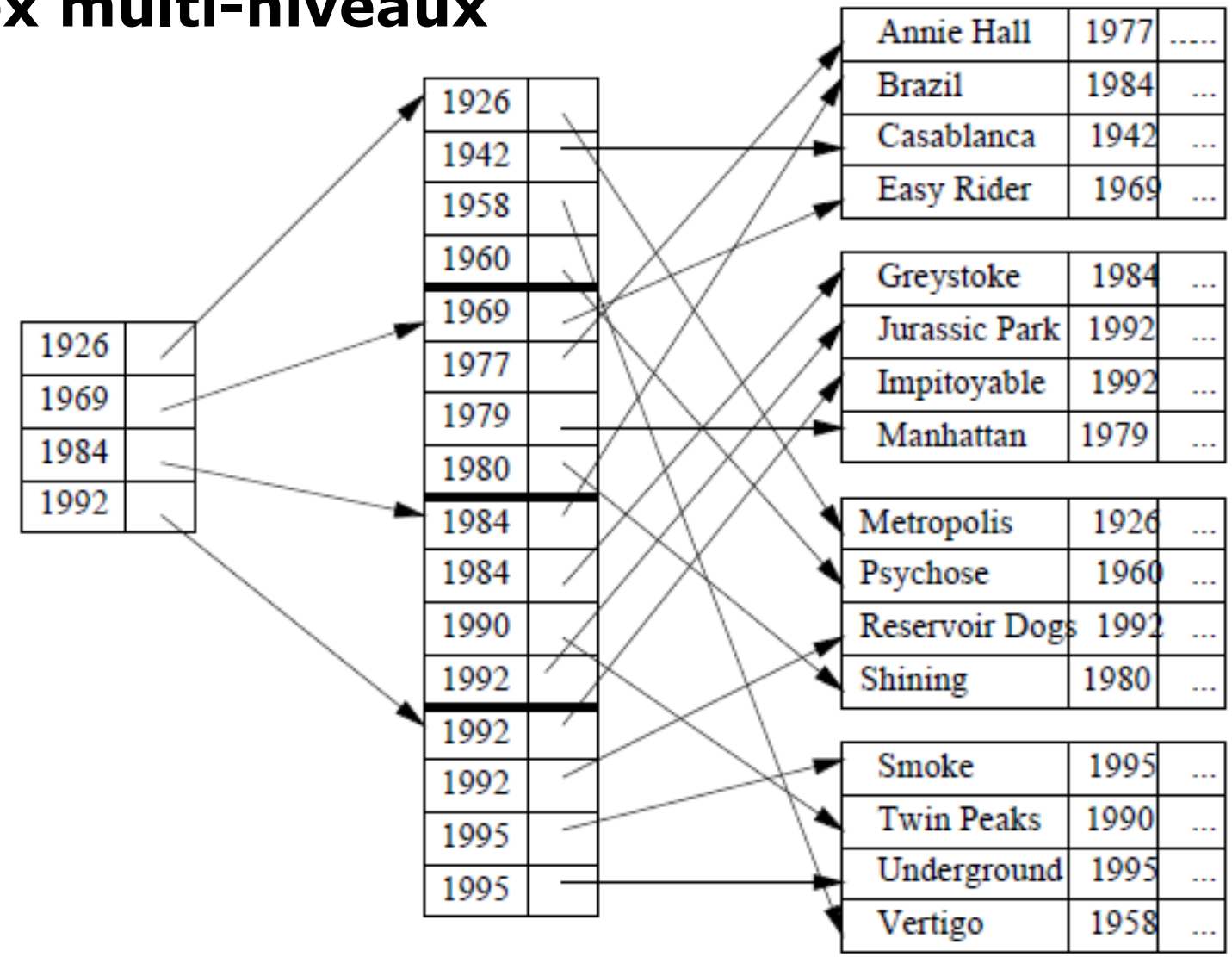
5. 1. Typologie des index

Index dense/non dense (sparce)



5. 1. Typologie des index

Index multi-niveaux



5. 2. Classification des index

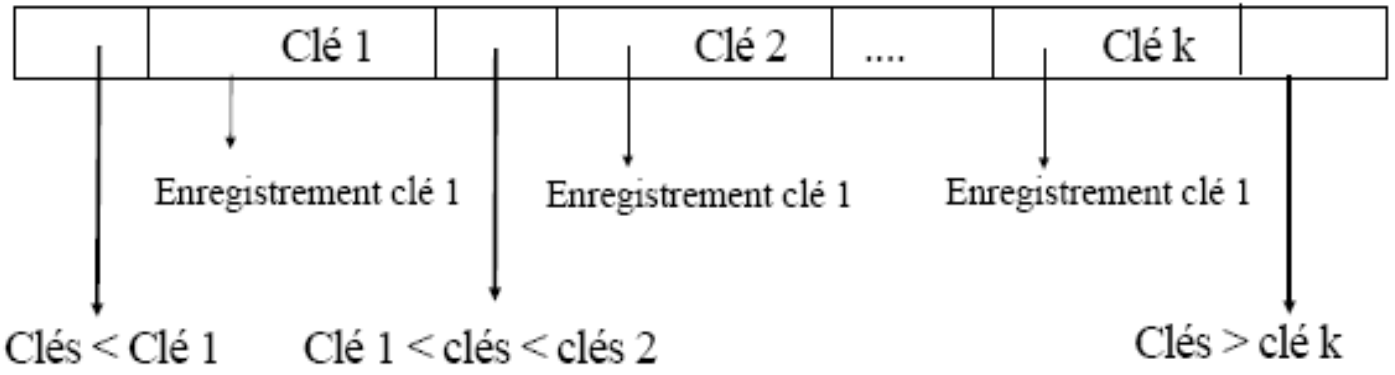
- Accès mono-critère :
  - Arbres :
    - B-Arbres
  - Accès par hachage
    - Statique
    - Dynamique

5. 3. Représentation des index

B-arbres (b-tree : balanced tree)

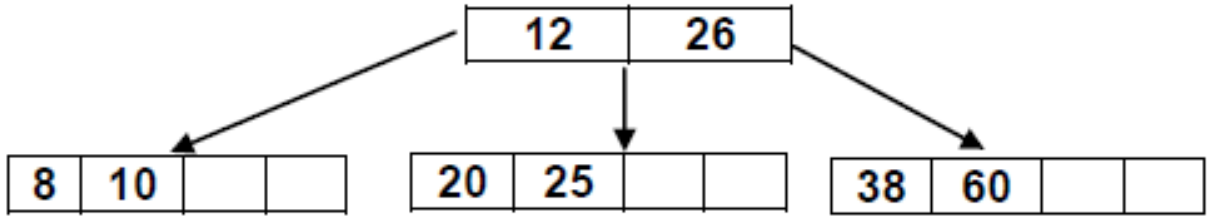
Un B-arbre d'ordre m est un arbre tel que

- *chaque nœud contient k clés triées, avec $m \leq k \leq 2m$ sauf la racine pour laquelle k vérifie $1 \leq k \leq 2m$.*
- *tout nœud non feuille a (k+1) fils. Le $i^{ème}$ fils a des clés comprises entre les $(i-1)^{ème}$ et $i^{ème}$ clés du père.*
- *l'arbre est équilibré.*



5. 3. Représentation des index

B-arbres



- Les nœuds qui ont m clés ont m+1 fils
- 1er fils : clés inférieures à la 1ère clé
- Autres fils : clés supérieures à une clé et inférieures à la clé suivante (si elle existe)

5. 3. Représentation des index

B-arbres

- Ils permettent de réduire fortement le nombre d'accès disque
- On peut parcourir l'arbre dans l'ordre des clés
- Les caractéristiques physiques principales :
  - toutes les branches ont la même profondeur
  - chaque nœud a toujours un nombre minimum et un nombre maximum de clés

5. 3. Représentation des index

B-arbres : *Implémentation*

- Principe des arbres binaires mais chaque nœud a beaucoup plus que 2 fils (plusieurs dizaines le plus souvent)
- B-arbre d'ordre n : les nœuds ont au moins n et au plus 2*n clés (sauf, la racine qui peut n'avoir qu'une seule clé)
- Nombre de fils d'un nœud = nombre de clés + 1
- Toutes les feuilles sont au même niveau

5. 3. Représentation des index

B-arbres : *Insertion*

Recherche de la feuille d'insertion

- **Si** la feuille n'est pas pleine **Alors**
- insérer la clé "à sa place" **Sinon**
- /* la feuille est pleine 2m clés */
  - laisser les m plus petites clés dans le nœud
  - allouer un nouveau nœud et y placer les m plus grandes clés
  - remonter la clé médiane dans le nœud père
  - application récursive de ce principe éventuellement jusqu'à la racine
- **Finsi**

5. 3. Représentation des index

Variante B+ des B-arbres

- La structure d'index la plus utilisée (➔Un composant essentiel des SGBD)
- Est un B-arbre ou les feuilles contiennent toutes les clés.
- Les feuilles pointent sur le fichier
- Nœuds et feuilles correspondent à des blocs disque
- Contenu d'un nœud : $P_0K_1P_1...P_{N-1}K_NP_N$
- K_i : valeurs des clés $i \leq j \rightarrow K_i \leq K_j$; P_i : pointeurs
- Contenu d'une feuille : Bloc de données avec des clés voisines

5. 3. Représentation des index

Variante B+ des B-arbres

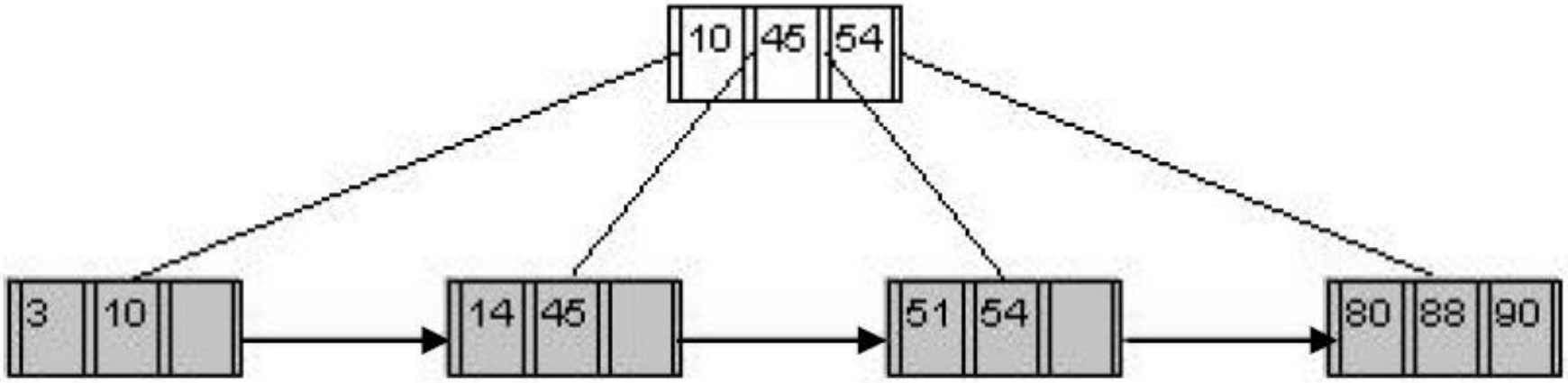
- Arbre B+ c'est un index organisé en B-arbre plus un ensemble de feuilles contenant les clés

- Conséquences :
  - Éclatement d'une feuille : une copie de la clé médiane est remontée
  - Suppression d'une clé uniquement dans une feuille
  - Recherche jusqu'aux feuilles

5. 3. Représentation des index

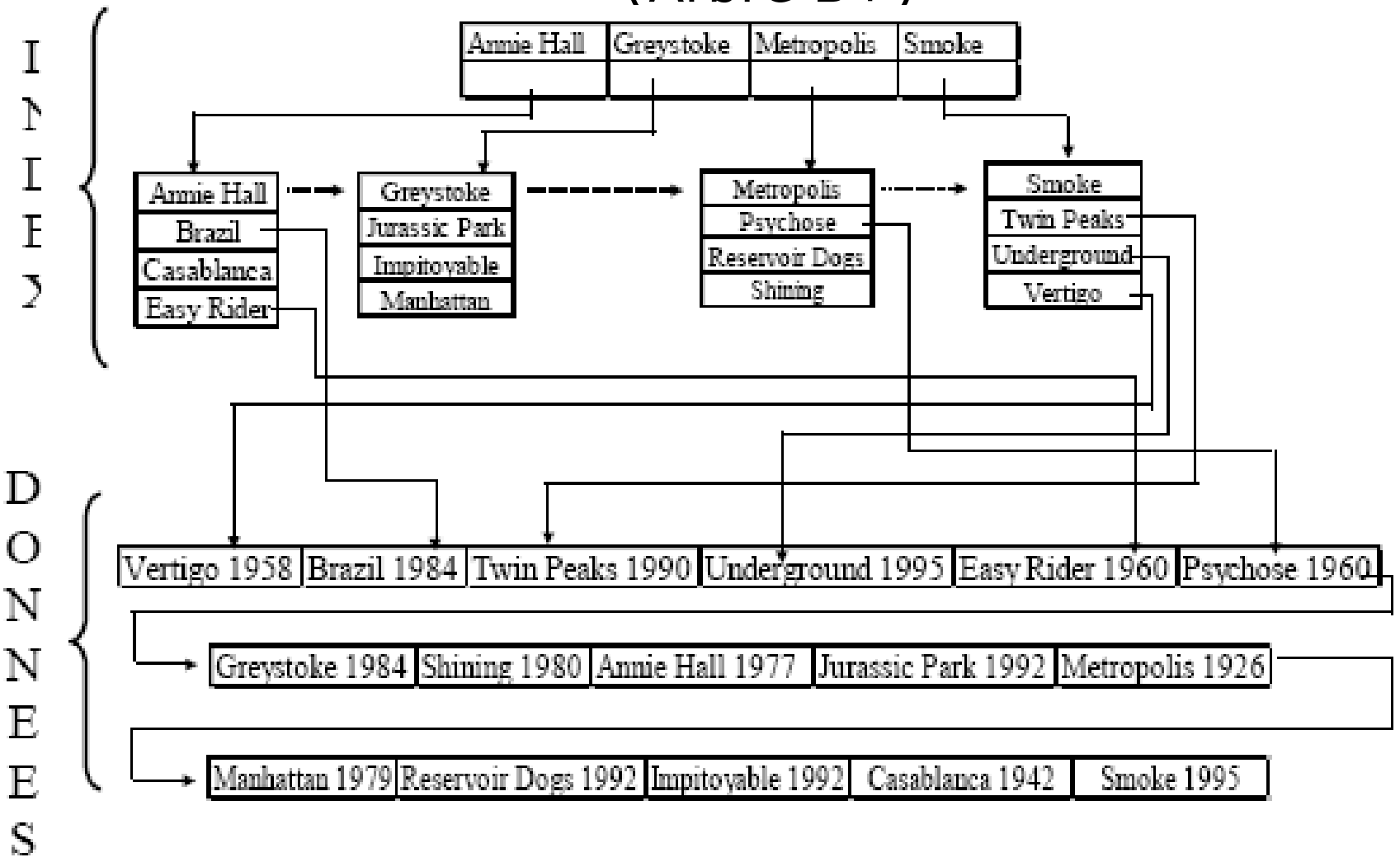
Variante B+ des B-arbres

- Les clés sont toutes rangées dans les feuilles et les feuilles sont chaînées
- Le parcours de toutes les clés dans l'ordre de l'index est ainsi très rapide (on parcourt la liste chaînée des feuilles)



5. 3. Représentation des index

Exemple d'index basé sur les structures arborescentes (Arbre B+)



5. 3. Représentation des index

Index bitmap

- Considère toutes les valeurs possibles pour un attribut. Pour chaque valeur, on stocke un tableau de bits (dit bitmap) avec autant de bits qu'il y a de lignes dans la table.
- Très utile pour les colonnes qui ne possèdent que quelques valeurs distinctes.
- Utile lorsque les données de la table ne sont presque jamais modifiées
- Peut être très performants sur des combinaisons («et», «ou») de critères (reviennent à faire des opérations sur des tableaux de bits)

5. 3. Représentation des index

Index bitmap : *Exemple 1*

Identificateur	F	M
8999887	0	1
8999885	0	1
8999878	1	0
8999865	0	1
8999889	1	0
8999899	0	1

Index bitmap sur le sexe des employés

5. 3. Représentation des index

Index bitmap : *Exemple 2*

	Titre	Genre
1	Vertigo	Suspense
2	Brazil	Science-fiction
3	Twin Peaks	Fantastique
4	Underground	Drame
5	Easy Rider	Drame
6	Psychose	Drame
7	Greystoke	Aventures
8	Shining	Fantastique
9	Annie Hall	Comédie
10	Jurassic Park	Science-fiction
11	Metropolis	Science-fiction
12	Manhattan	Comédie
13	Reservoir Dogs	Policier
14	Impitoyable	Western
15	Casablanca	Drame
16	Smoke	Comédie

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Drame	0	0	0	1	1	1	0	0	0	0	0	0	0	0	1	0
Science-fiction	0	1	0	0	0	0	0	0	0	1	1	0	0	0	0	0
Comédie	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	1

Index bitmap sur
le genre des films

5. 3. Représentation des index

Index bitmap : *Implémentation*

- Très différente de celle des B-arbres
- Tableau de bits avec autant de colonnes (ou lignes) que de valeurs distinctes de l'attribut indexé et autant de lignes (ou colonne) que la table
- Un bit est à 1 si la ligne de la table a la valeur correspondant à la colonne
- Il ne faut pas trop de valeurs distinctes ni trop de modifications

5. 3. Représentation des index

Index bitmap : *Recherche*

nb colonnes = nb valeurs distinctes

Index bitmap sur le sexe des employés

Identificateur	F	M
8999887	0	1
8999885	0	1
8999878	1	0
8999865	0	1
8999889	1	0
8999899	0	1

Pour trouver les **employées**, il suffit de parcourir la colonne F et de récupérer les 1

5. 3. Représentation des index

Le Hachage :

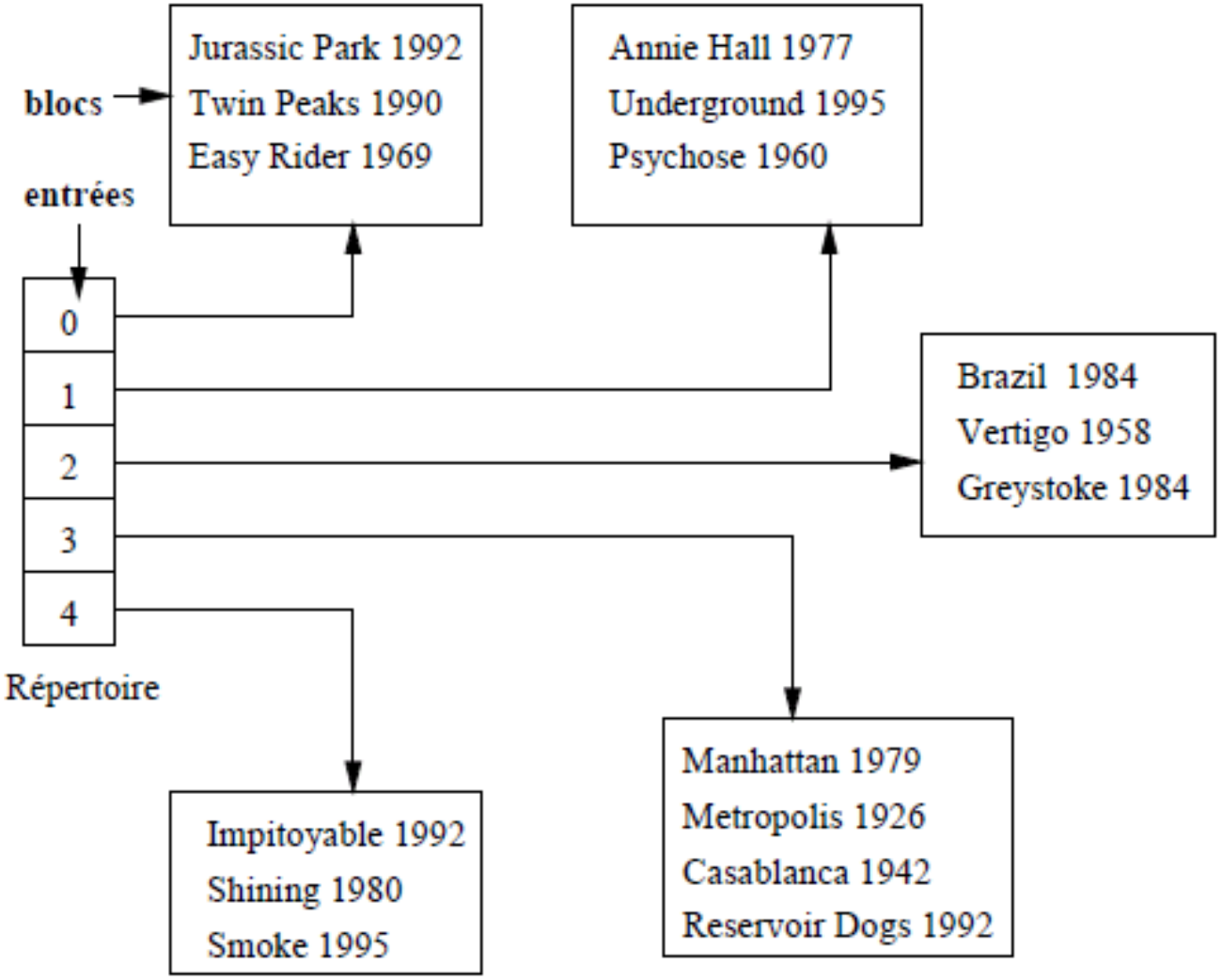
- Les tables de hachage sont des structures utilisées en mémoire centrale pour organiser des ensembles et fournir un accès performant à ses éléments.
- **Principe :**

Il s'agit d'organiser un ensemble d'éléments d'après une clé, et d'utiliser une fonction (dite de *hachage*) qui, pour chaque valeur de clé ***c***, donne l'adresse ***f(c)*** d'un espace de stockage où l'élément doit être placé.

5. 3. Représentation des index

Le Hachage :

Exemple d'une table de hachage



5. 4. Comparaison

- Les B-arbres sont très souples et offrent des recherches rapides en fonction d'une valeur de la clé ou d'une plage de valeurs. En général, on crée un tel index pour les colonnes ayant un très grand nombre de valeurs différentes ou qui interviennent dans des critères de recherche ou de jointure. C'est ainsi qu'en pratique, on en définit toujours pour les clés primaires et étrangères.

5. 4. Comparaison

- Les index hachés permettent en une seule E/S de retrouver un n-uplet dont on donne la valeur de la clé.
- Les index à matrices binaires (bitmap) peuvent être utiles pour les requêtes de comptage en fonction de critères portant sur plusieurs colonnes comportant que peu de valeurs différentes. En pratique, ces index sont utilisés pour de grandes tables.

➔ *La maintenance d'un index représente toujours une surcharge de travail pour le SGBD. Dès lors, on évite de créer des index sur les colonnes fortement modifiées.*

6. Conclusion : Règles à respecter

- Ne pas créer d'index pour des tables de moins de 200-300 lignes, l'accès séquentiel est plus rapide.
- Ne pas créer d'index sur une colonne qui ne possède que quelques valeurs différentes.
- Indexer les colonnes qui interviennent souvent dans les clauses WHERE et ORDER BY
- Indexer les colonnes de jointure.