Université Paris Nanterre

Master MIAGE – Première année

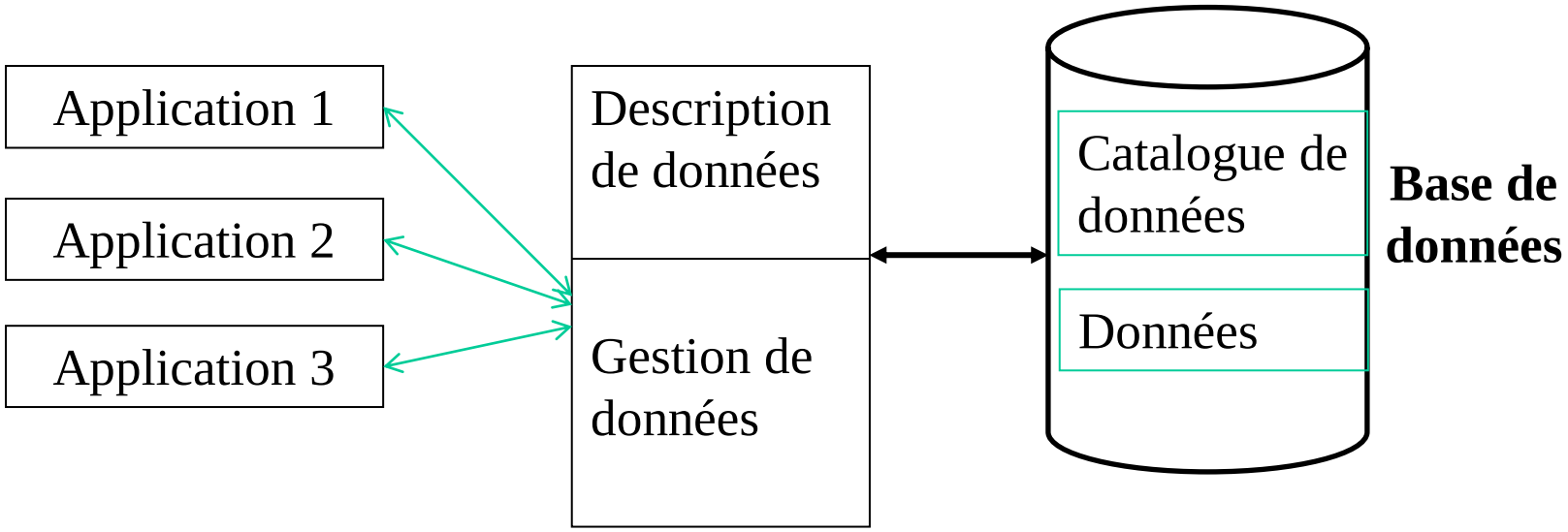
Bases de données avancées

P1 –Stockage et Indexation

Marta Rukoz

mcastill@parisnanterre.fr

Système de gestion de base de données.
Stockage de données



- Regroupement de données communes à plusieurs applications
- Diminution de la redondance

BD relationnelles (review)

- Basées sur la théorie relationnelle : Domaines, Relations, Opérateurs relationnels…
- Mis en œuvre sous forme des tableaux à deux dimensions contenant des lignes et des colonnes.
- Requêtes sont écrit en Structured Query Language (SQL).
- Les données sont typées et peuvent être numériques, *string*, *dates*, blobs **non interprétés** ou autres types. Les types sont appliquées par le système.
- Les tableaux peuvent se joindre et se transformer en nouveaux et plus complexes tableaux, en raison de leur fondement mathématique dans la théorie relationnelle (ensemble).

BD relationnelles

Base de données : Agences
Agence(NAgence, Nom, Ville, Directeur)
Compte(NCompte, NClient)
Localisation(NCompte, Nagence, Solde)
Client(Nclient, Nom, Adresse)

Agence

NAgence	Nom	Ville	Directeur
A1	Paris1	Paris	Keller
A2	Lyon1	Lyon	Martin
A3	Nantes3	Nantes	Sens
A2	Marseille	Marseille	Heirs

Compte

NCompte	Nagence	Solde
310	A1	1500
315	A1	20000
330	A1	50000
440	A2	600
500	A3	70000
320	A4	40000
650	A4	8000
530	A4	700

Compte-Client

Ncompte	Nclient
310	C1
310	C2
315	C1
330	C3
440	C4

Client

Nclient	Nom	Adresse
C1	Peters	X Av.
C2	Martin	Ad2
C3	Germain	Ad44
C4	Boit	Ad3
C5	Seull	AD5

Supprimer le client C4

Supprimer le Compte-Client (440, C4) et le compte 440 (car seule client sur ce compte) et le client.

Système de Gestion de Base de Données (SGBD)

Objectifs principaux :

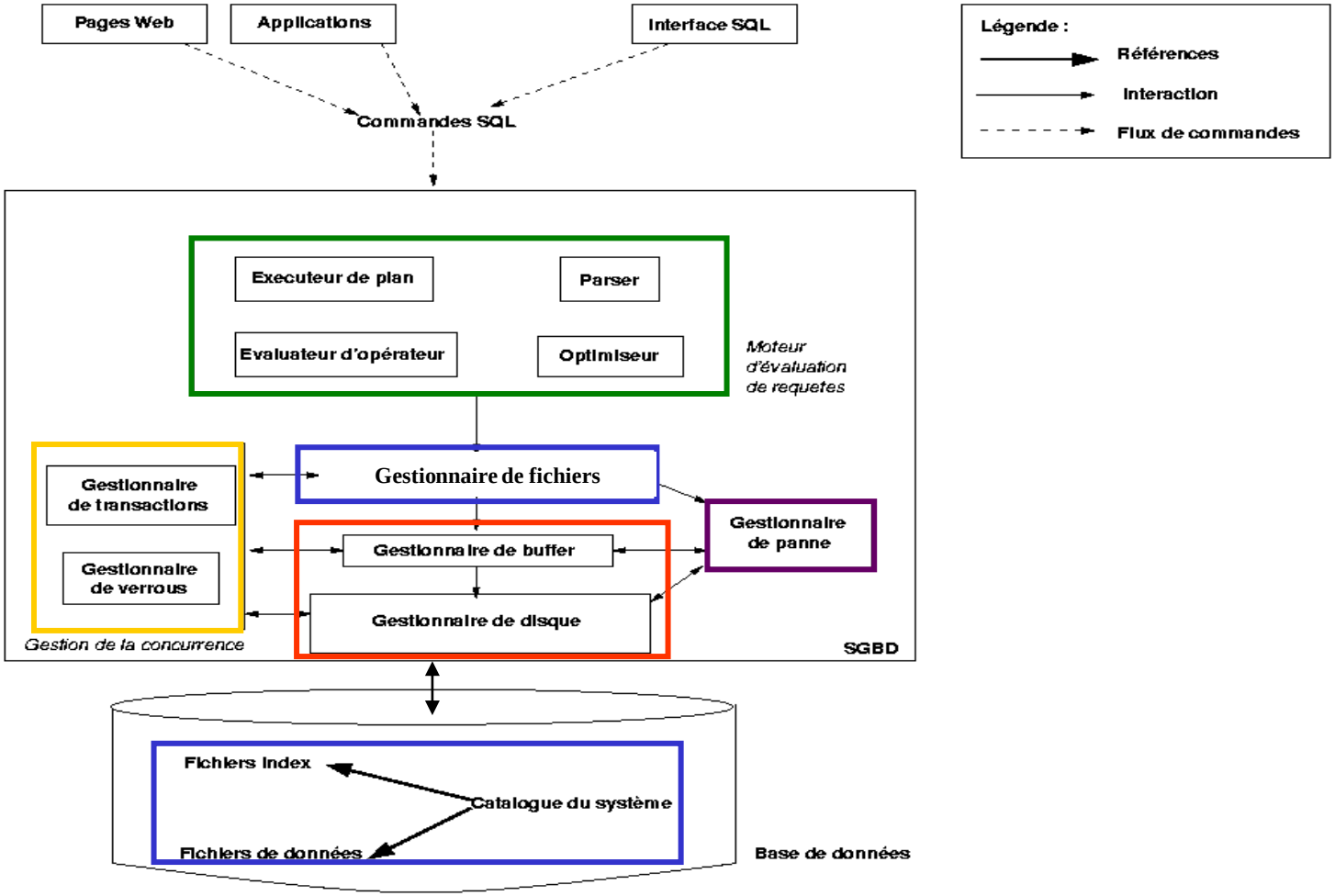
- Indépendance physique et logique
- Accès aux données par l'intermédiaire d'un Langage de Manipulation de Données (LMD)
- Administration centralisée des données (intégration)
- Non-redondance des données
- Partage, Cohérence, et Sécurité des données
- Résistance aux pannes :

Un SGDBD prendre en charge la gestion de données pour toutes les applications, il garantit :

- **La reprise sur panne.**
- **La cohérence de données.**
- **La persistance**

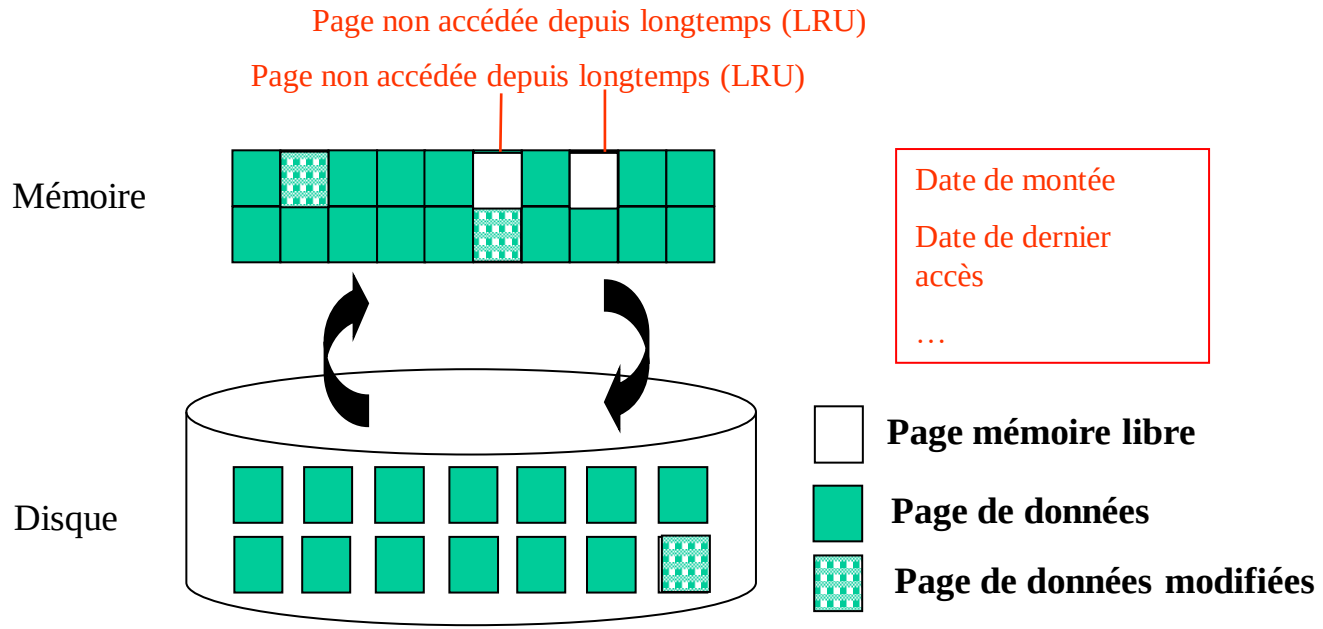
Peux s'appuyer sur le Système de Gestion de Fichier (SGF) système d'exploitation

Architecture d'un SGBD relationnel



Gestion de Stockage (buffer et disque)

Afin de gérer le stockage de données, les espaces de stockage, mémoire vive et disque, sont en générale découpés en unités d'accès(E/S) d'une taille fixe, pages (niveau logique, contenu) ou blocs (niveau physique).



- Un SGBD doit ranger les données sur disque pour les faire persistantes (survivre à un arrêt du système).
- Un SGB doit amener les données en mémoire pour les traiter.

La performance *d'une application bases de données* ***dépend*** *fortement de la capacité à gérer efficacement les* ***transferts disque-mémoire***

Notation

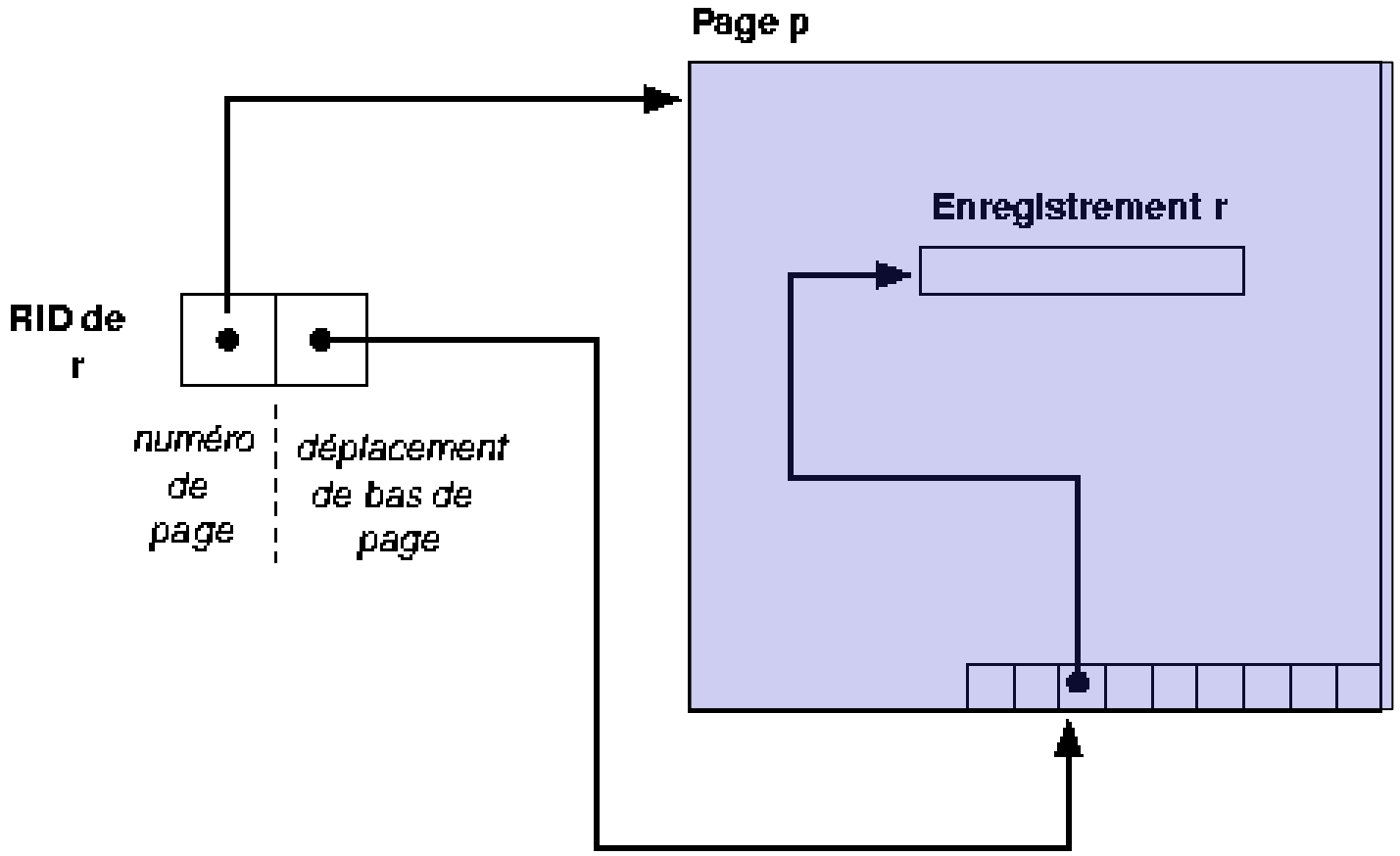
Base de données

- **Attribut :** valeurs
- **Nuplet** : collection de taille fixe ou variable d'attributs
- **Table ou relation** : collection de **Nuplets**

Système de fichiers

- **champs** : séquences d'octets de taille fixe ou variable représentant la valeur d'un attribut sur le disque ou en mémoire
- **enregistrement** : collection de taille fixe ou variable de champs
- **Fichier** : collection de **pages(logique)** ou **blocs(physique)** disque ou mémoire

Identification des enregistrements



Répartition des enregistrements dans les pages

- Taille d'une page : **B**
- Taille d'un enregistrement : **R**
  - Si B > R, une page peut contenir plusieurs enregistrements
    - Facteur de blocage Bfactor = B/R
  - Si B/R ≠ entier, espace libre = B – (Bfator * R) octets

Exemple :

- **B**=4096, **R**=84, ⌊4096/84⌋ = **48** enregistrements par page;
- L'enregistrement 550 est dans la page ⌊550/48⌋ + 1= 12ème;
- La page 12ème contient les enregistrements ((11 x 48) +1) **529 à 576** (12 x 48) ;
- L'enregistrement 550 est le 22ème enregistrement de la 12ème page.

On peut aussi le référencer par le fichier + la page + le numéro interne.

Soit F1.12.22

Répartition des enregistrements dans les blocs

Si espace libre (B – (Bfator * R) ≠ 0), alors :

- Stockage d'une partie d'un enregistrement, un pointeur vers le bloc contenant le reste de l'enregistrement
  - **stockage étendu**

- Si un enregistrement ne peut être réparti dans plusieurs blocs
  - **stockage non-étendu**
  - **Perte d'espace**

Stockage et adressage des enregistrements

En générale un enregistrement est stocké dans une seule page. L'adresse d'un enregistrement est le ROWID :

- Numéro du fichier.
- Numéro de la page dans le fichier.
- Numéro du n-uplet dans la page.

Exemple : 001.00000FF4.002. est l'adresse du deuxième n-uplet, de la page FF4 dans le fichier 001

Organisation de fichiers

Une base de données = un ou plusieurs fichiers

Un fichier = un ou plusieurs blocs (espace physique) contenant de pages (entité logique)

Comment organiser les fichiers d'une base de données ?

- l'espace est-il bien utilisé ?
- Est-il facile et efficace de faire une recherche ?
- Est-il facile de faire une mise à jour ?
- Les données sont-elles correctement représentées et en sécurité ?

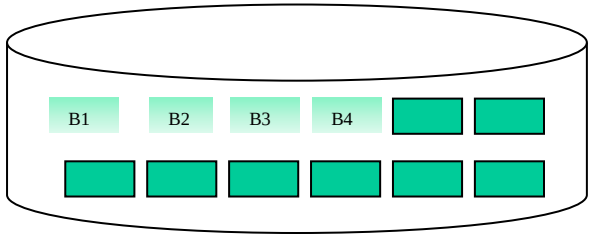
On Intégra ou non des fonctionnalités du SGF du système d'exploitation :

- A chaque relation correspond un fichier
  - $\Rightarrow$ liaison forte du SGBD et du SGF
- Stockage de toute la base de données dans un seul fichier
  - $\Rightarrow$ le SGF donne accès aux différentes pages
  - $\Rightarrow$ le SGBD contrôle tout

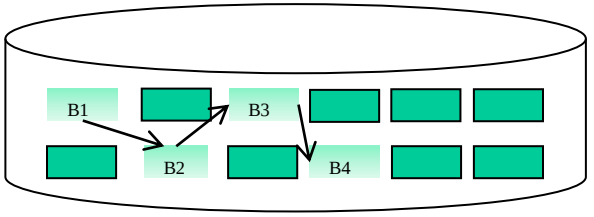
Les pages doivent être connues du SGBD

Placement de blocs aux fichiers

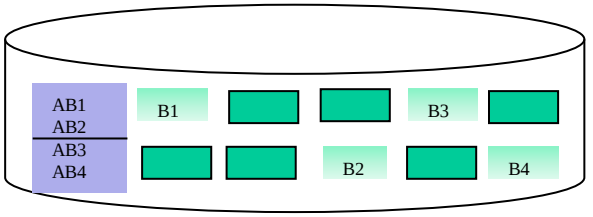
- **Allocation contiguë** : blocs du fichier dans des blocs consécutifs → lecture du fichier rapide



- **Allocation chaînée** : un bloc du fichier pointe vers le bloc suivant



- **Allocation indexée** : des blocs d'index contiennent des pointeurs vers les blocs du fichier



Organisation de fichier

Fait référence à l'**organisation** d'un fichier en **enregistrements, blocs et structures, et méthodes d'accès**, inclut la manière dont les enregistrements sont distribués sur le support de stockage.

•**Objectif** : rendre efficaces les opérations suivantes

- Localisation d'un enregistrement
- Lecture d'un enregistrement
- Localisation de l'enregistrement suivant
- Suppression d'un enregistrement
- Modification d'un enregistrement
- Insertion d'un nouvel enregistrement

Méthodes d'accès : ensemble de programmes permettant d'effectuer de manière optimale les opérations précédentes

- Plusieurs méthodes d'accès peuvent être utilisées pour une organisation de fichier
- Certaines méthodes d'accès ne peuvent être utilisées pour certaines organisations de fichiers.

  •Ex : méthode d'accès indexées pour un fichier dépourvu d'index

Organisation de fichier

- Tout fichier a une organisation dite primaire
  - Elle définit le **mode de placement des enregistrements** dans le fichier
  - Elle est déterminante pour l'insertion et la suppression d'enregistrements.

- Un fichier peut avoir une ou plusieurs organisations secondaires
  - Associées à une structure de données supplémentaire (des index), facilitant l'accès aux données
  - Elles améliorent les autres opérations

Organisation des fichiers

Organisation séquentielle :

- **Fichier aléatoire :** Les enregistrements sont placés de manière aléatoire dans les pages du fichier. La recherche d'un enregistrement est réalisée par son RID.
- **Fichier ordonné :** Les enregistrements sont ordonnés selon la valeur d'un **champ (clé de recherche).** Attention la clé de recherche n'est pas nécessairement la clé de la relation .

Organisation directe :

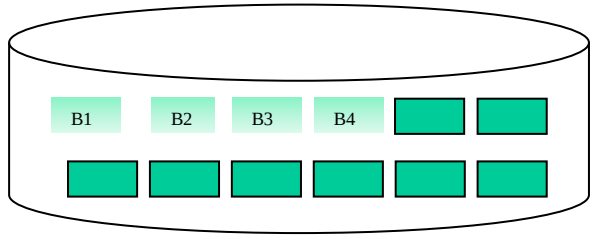
Une fonction de hachage calcule le paquet où doit être stocké l'enregistrement en fonction de la valeur d'un ou plusieurs champs

- Fichier de hachage statique (taille fixe)
- Fichier de hachage : extensible

Organisation séquentielle indexée :

- Index primaire ou index plaçant
  - Arbres B et B+
  - Index de groupement (clustering index)
- Index secondaire

Organisation séquentielle



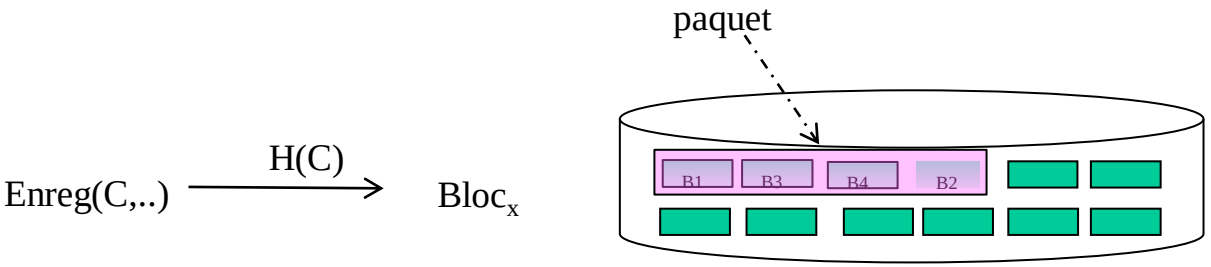
- Si le fichier est trié sur le champ servant de critère de recherche, il est possible d'effectuer une recherche par dichotomie qui est beaucoup plus efficace.

- Effacement d'enregistrement : lecture + réécriture
- Gestion de l'espace vide :
  - Marquer les enregistrements à effacer + compacter l'espace régulièrement
  - Insérer des enregistrements à la place des enregistrements à effacer

- Fichier peut être étendue/non étendu, les enregistrements peuvent être de taille fixe ou variable

Organisation directe : hachage

- Distribution non séquentielle.
- Pages groupées en paquet (un ou plusieurs blocs contiguës)
- Une fonction de hachage calcule le paquet où doit être stocké l'enregistrement en fonction de la valeur d'un ou plusieurs champs.

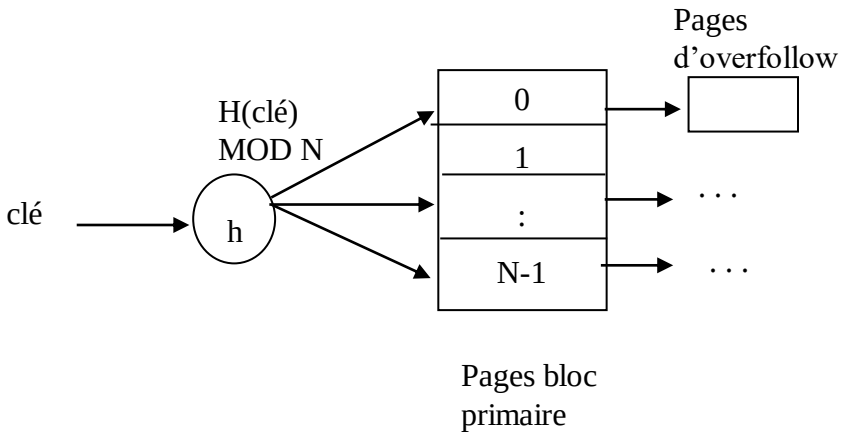


+ Recherche d'enregistrement par sa clé facile
+ Cette structure n'occupe aucun espace disque, il permet des recherches par clé par accès direct (calculé) au bloc susceptible de contenir l'enregistrement.

– Impossible d'optimiser les recherches par intervalle car l'organisation des enregistrements ne s'appuie pas sur l'ordre des clés.
– Il n'existe pas de garantie d'adresse unique (le nombre de valeurs possibles est généralement plus grand que le nombre d'adresses)
– Quand une même adresse est générée pour deux ou plusieurs enregistrements, on parle de ***collisions***.

Organisation directe : hachage statique (taille fixe)

- Le nombre de blocs du fichier est fixe.
- Si bloc primaire est plein lors insertion, on utilise des pages **d'overflow**, liées au bloc.



OVERFLOW

- ***Unchained overflow*** : Maintien d'une zone d'overflow
- ***Chained overflow*** : Maintien, pour chaque bloc, d'un pointeur vers une zone d'overflow
- ***Multiple hashing*** : Utilisation d'une deuxième fonction de hachage pour placer les enregistrements dans la zone d'overflow

- Si le nombre d'enregistrements est inférieur à celui prévu → gaspillage de l'espace disque
- Si le nombre d'enregistrements est supérieur à celui prévu → collision qui ralentissent l'exploitation du fichier

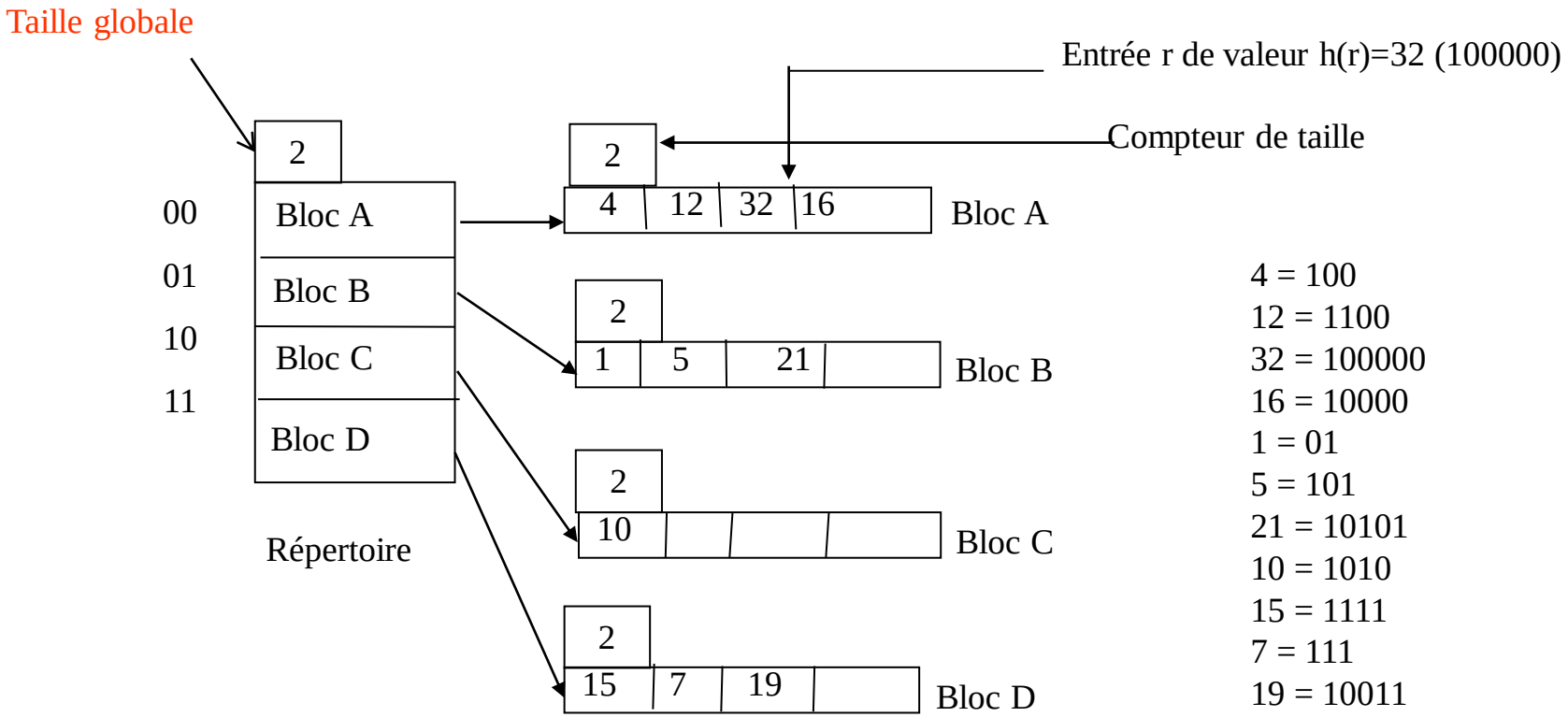
Solution : modifier le nombre de blocs de manière dynamique

Hachage extensible

- Structure d'index dynamique
- Introduit une indirection par un *répertoire de blocs*
- Dédouble le fichier en doublant la taille du répertoire et en découpant uniquement les blocs pleins.
- Généralement, taille du répertoire est 2^d où ***d*** est la taille globale de l'index.
- L'entrée dans le répertoire est déterminée par les ***d*** derniers bits du résultat de la fonction de hachage.
- L'entrée dans le répertoire pointe vers des pages de disque qui contiennent les données.
- Quand une page est pleine, on redistribue les données en regardant les ***d*** derniers bits de la valeur de hachage.

Hachage extensible : exemple (1)

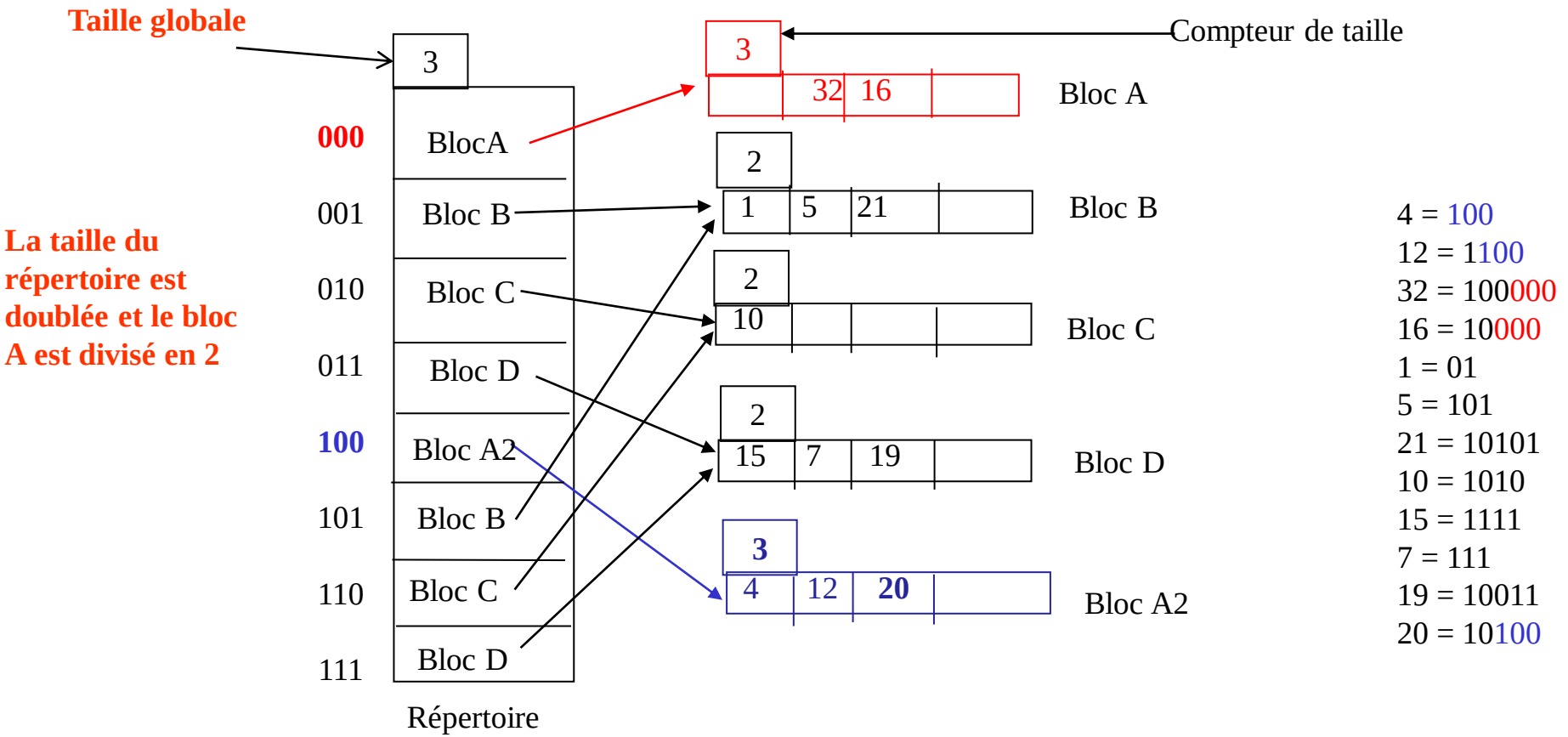
Fonction de hachage qui prendre en compte les deux derniers chiffres d'une représentation binaire d'un nombre



Hachage extensible : exemple (2)

Insertion de l'enregistrement h(r) = 20 (10100)

Il faut diviser le bloc A en deux et redistribuer les données du bloc, en considérant, cette fois ci, les trois derniers chiffres du nombre binaire



Organisation directe opérations

• Recherche :

+ Si le répertoire tient en mémoire principale, on peut effectuer une recherche en ne lisant qu'un bloc.

- Impossible d'optimiser les recherches par intervalle car l'organisation des enregistrements ne s'appuie pas sur l'ordre des clés.

• Mise à jour :

Mal adapté. L'espace alloué risque d'être insuffisant s'il y a des insertions. On chaîne alors les blocs.

Organisation séquentielle indexée

- Tire parti des avantages de l'organisation séquentielle et de l'organisation directe
- Améliore la recherche d'enregistrements par rapport à l'organisation séquentielle
- **Utilise une structure additionnelle : l'index**
  - À utiliser pour les fichiers dynamiques (insertions et suppressions fréquentes)
  - Fichier trié, beaucoup plus petit que le fichier de données
  - Efficacité des accès
  - Inconvénient : la mise à jour de de la table peut entraîner celle de l'index

TD 1 : Stockage de données

Les index

- Un index sur une table :
  - semblable à l'index à la fin d'un livre.
  - permet de répondre aux requêtes beaucoup plus rapidement en créant des chemins d'accès aux enregistrements beaucoup plus directs.
- Un index est formé des enregistrements avec des **clés** qui permettent ensuite de lire directement les données correspondantes.
- **3 alternatives pour les données des index :**
  1) Les **entrées** de **clé de recherche** *k* sont les enregistrements mêmes
  2) Les entrées sont des couples (*k*,*rid*)
  3) Les entrées sont des couples (*k*,*liste_rid*)

Index

- **Index primaire ou index plaçant :** les enregistrements sont stockés ordonnées suivant l'index

  Clé de recherche = clé primaire de la relation

- **Index secondaire :** construit à partir des champs qui n'ordonnent pas le fichier. Il peut y avoir plusieurs par fichier.

  → **(*clé de recherche*, *pointeur(s) vers les pages du fichier*)**

  → **(*clé de recherche*, *valeur(s) de clé primaire*)**

  ⇒ **l'index primaire doit être lu après l'index secondaire**

- Pas du tout nécessaire pour manipuler les informations d'une base de données en SQL.
- Requêtes SQL transparentes au fait qu'il existe ou non un index.

Structures d’index

- Arbre B
- Arbre B+
- Hachage
- Multi niveaux

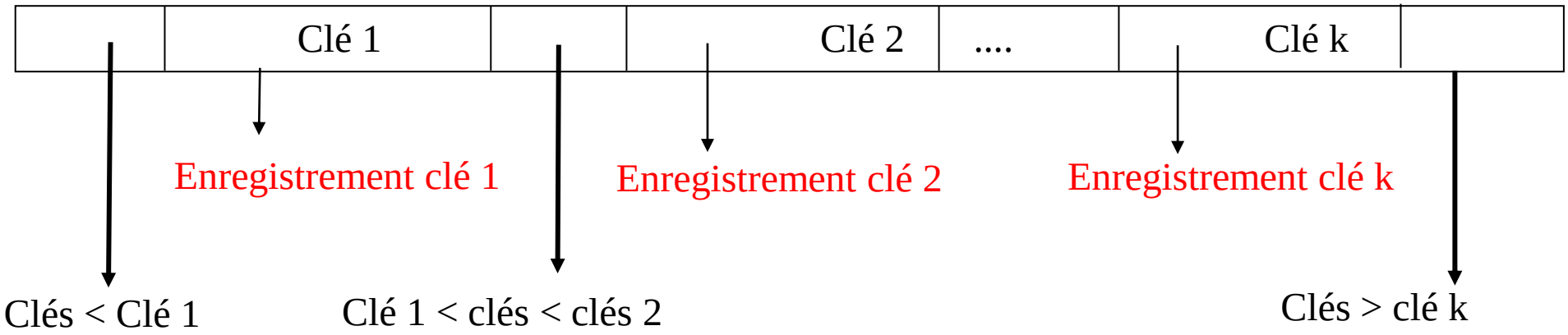
Arbre B d'ordre m

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

(i) chaque nœud contient **k** clés triées, avec $m \leq \mathbf{k} \leq 2m$ sauf la racine pour laquelle **k** vérifie $1 \leq k \leq 2m$.

(ii) tout nœud non-feuille a (k+1) fils. Le $i^{ème}$ fils a des clés comprises les $(i-1)^{ème}$ et $i^{ème}$ clés du père.

(iii) l'arbre est équilibré.



Insertion dans un arbre B d'ordre *m*

- 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

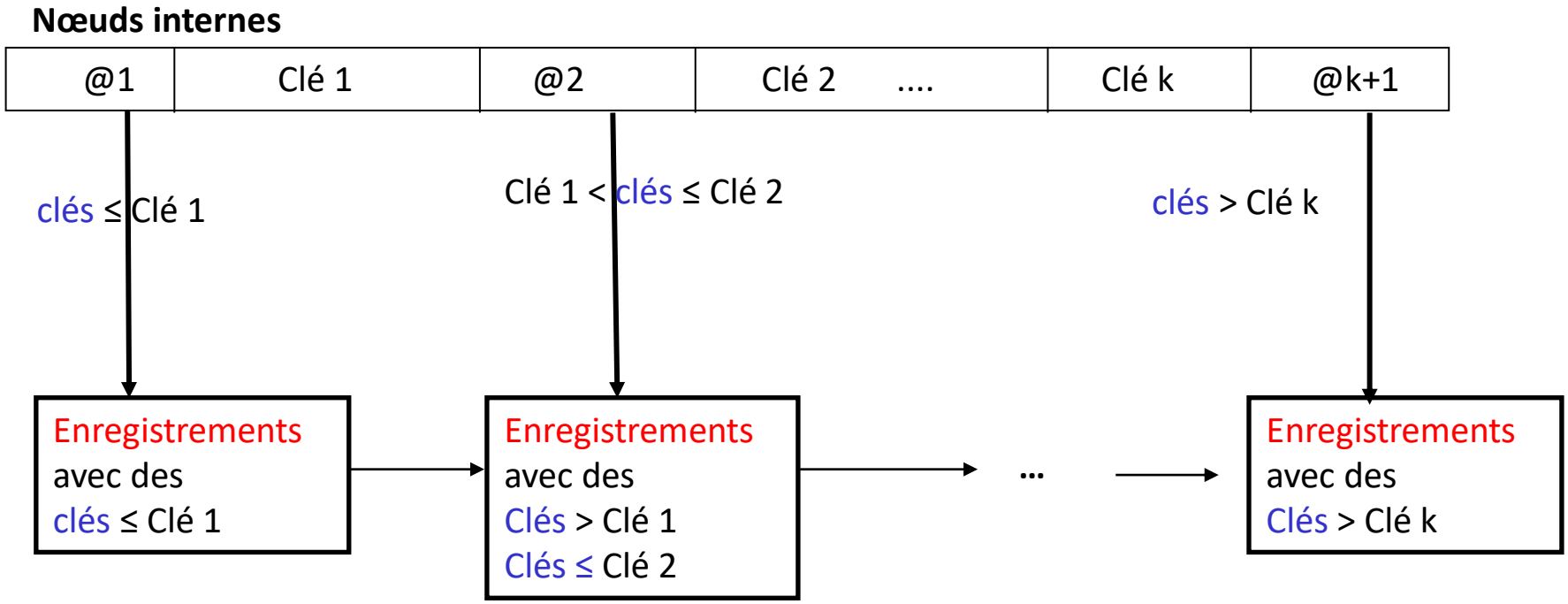
Arbre B^+ d'ordre m

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

(i) chaque nœud contient **k** clés triées, avec **k** ≤ 2m

(ii) tout nœud non-feuille a (**k+1**) fils.

(iii) l'arbre est équilibré.



Arbre B+ d'ordre *m*

1. Est un arbre B ou les feuilles contiennent toutes les clés
2. Suppression d'une clé uniquement dans une feuille

3. Index organise en arbre B + ensemble de feuilles contenant les clés
4. Conséquences :
    1. Éclatement d'une feuille : une copie de la clé médiane est remontée
    2. Suppression d'une clé : uniquement dans une feuille
    3. Recherche jusqu'aux feuilles

Index dense/non dense

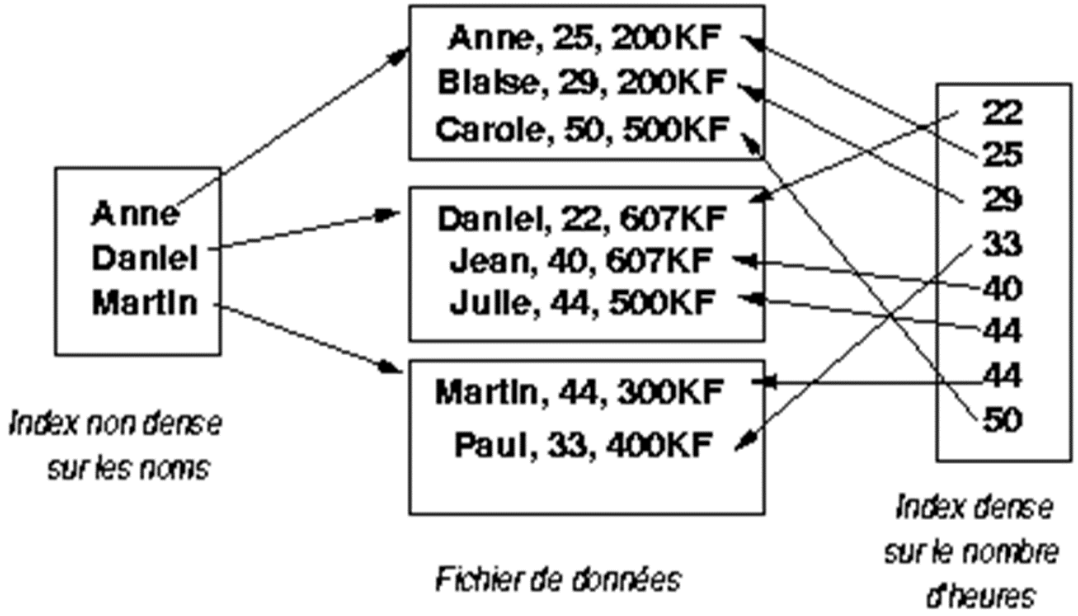
Dense

L'index contient une entrée pour chaque un enregistrement. Il est trié.

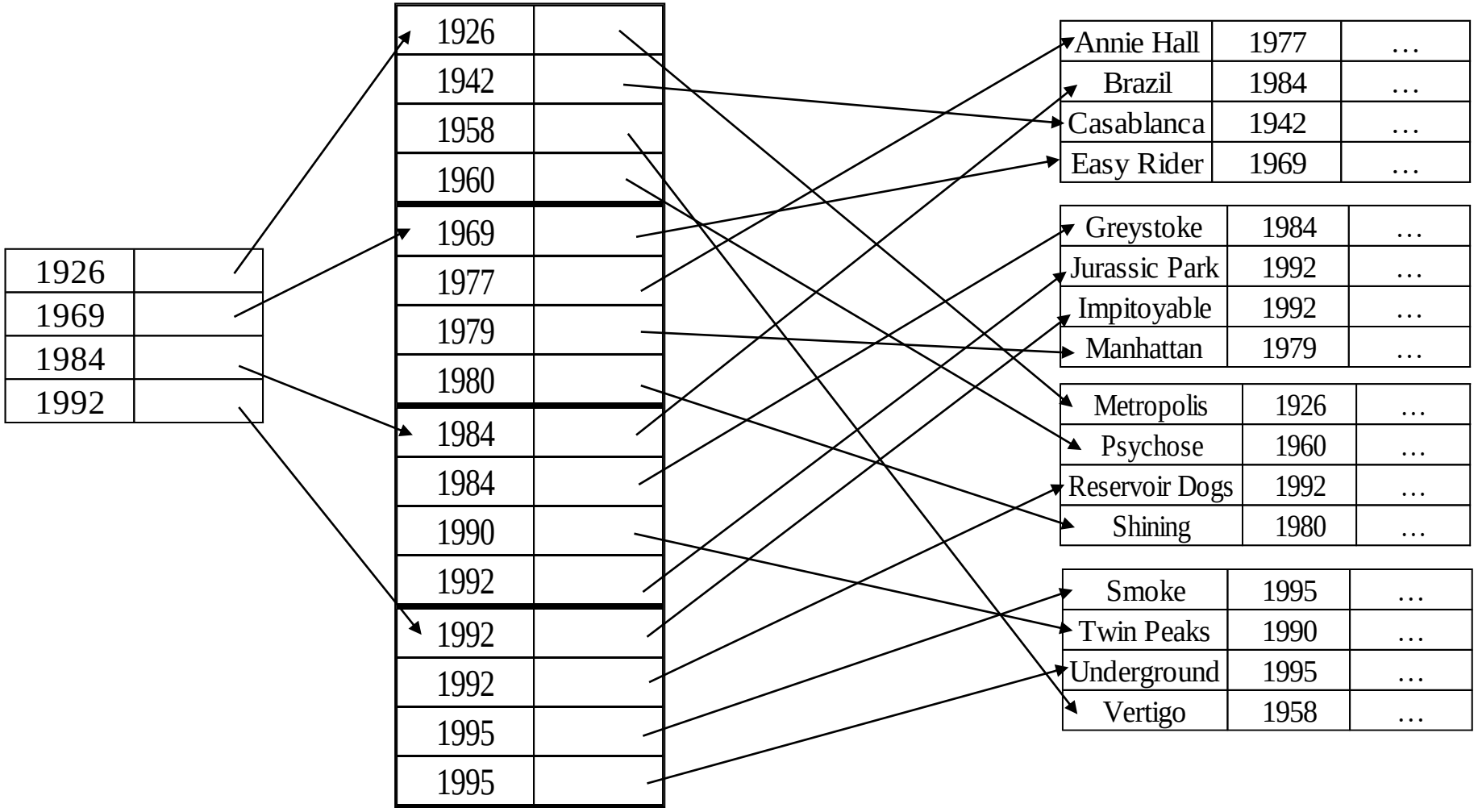
Non dense

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

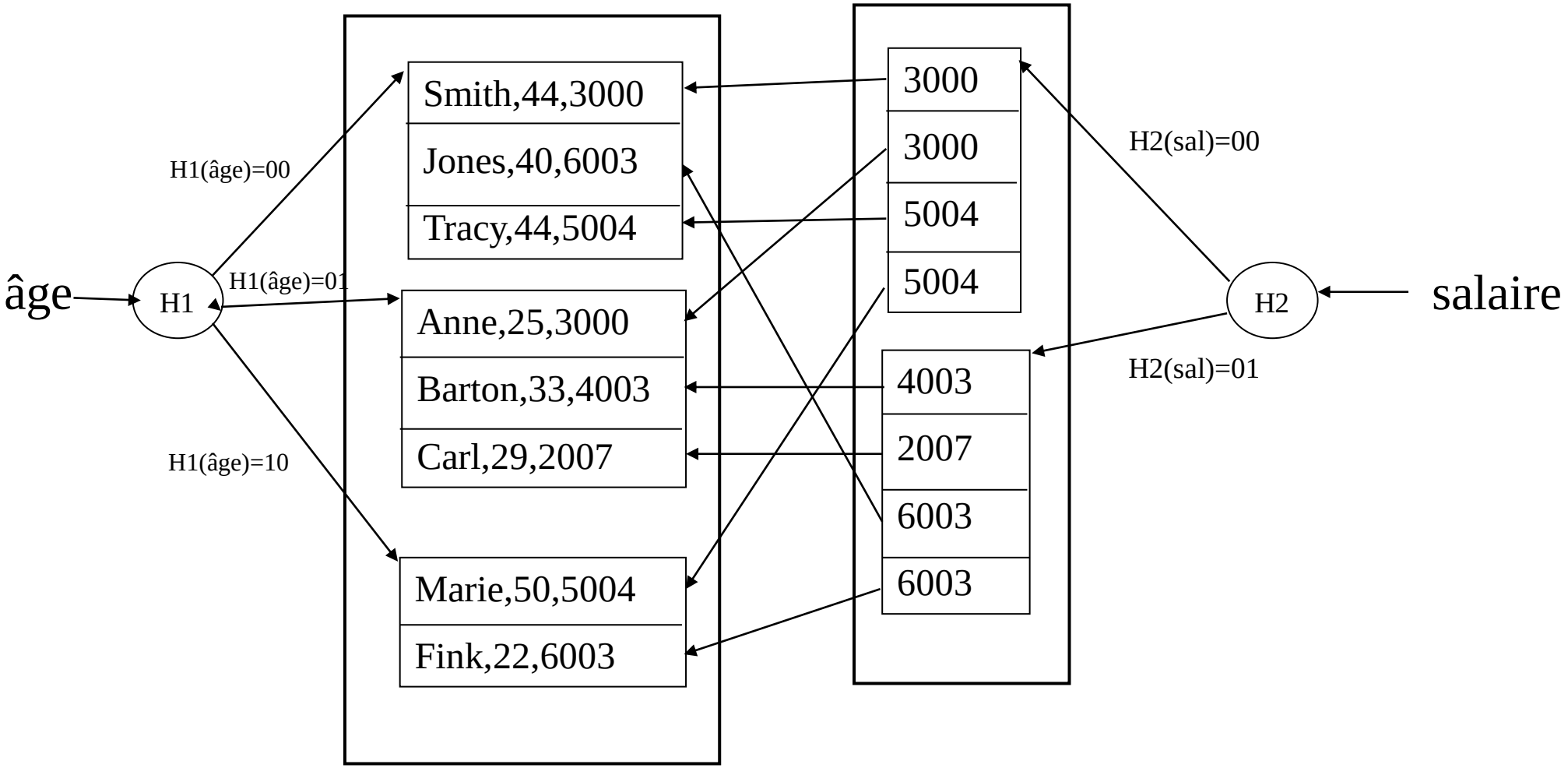
- Très efficace pour les recherches
- Problèmes : maintien de l'ordre suite à des insertions, destructions, maintien de la correspondance avec l'index.



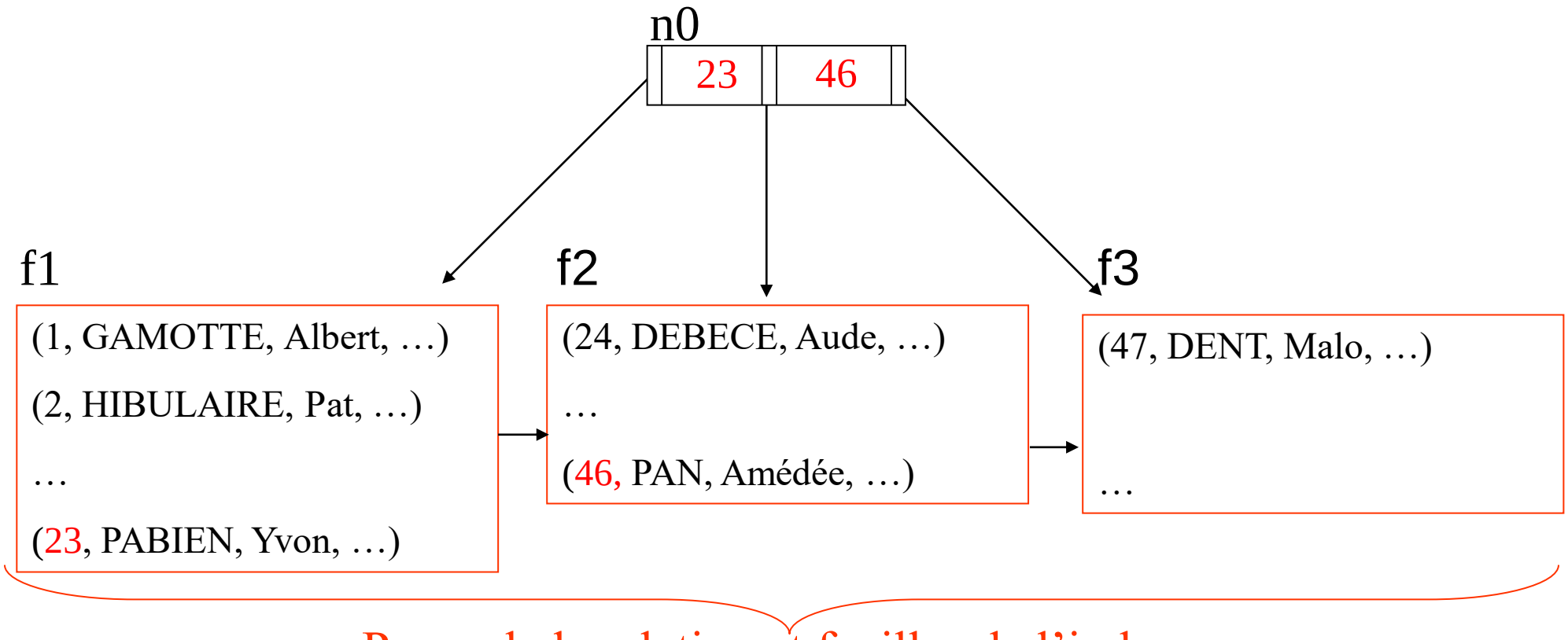
Index : *index multi-niveaux*



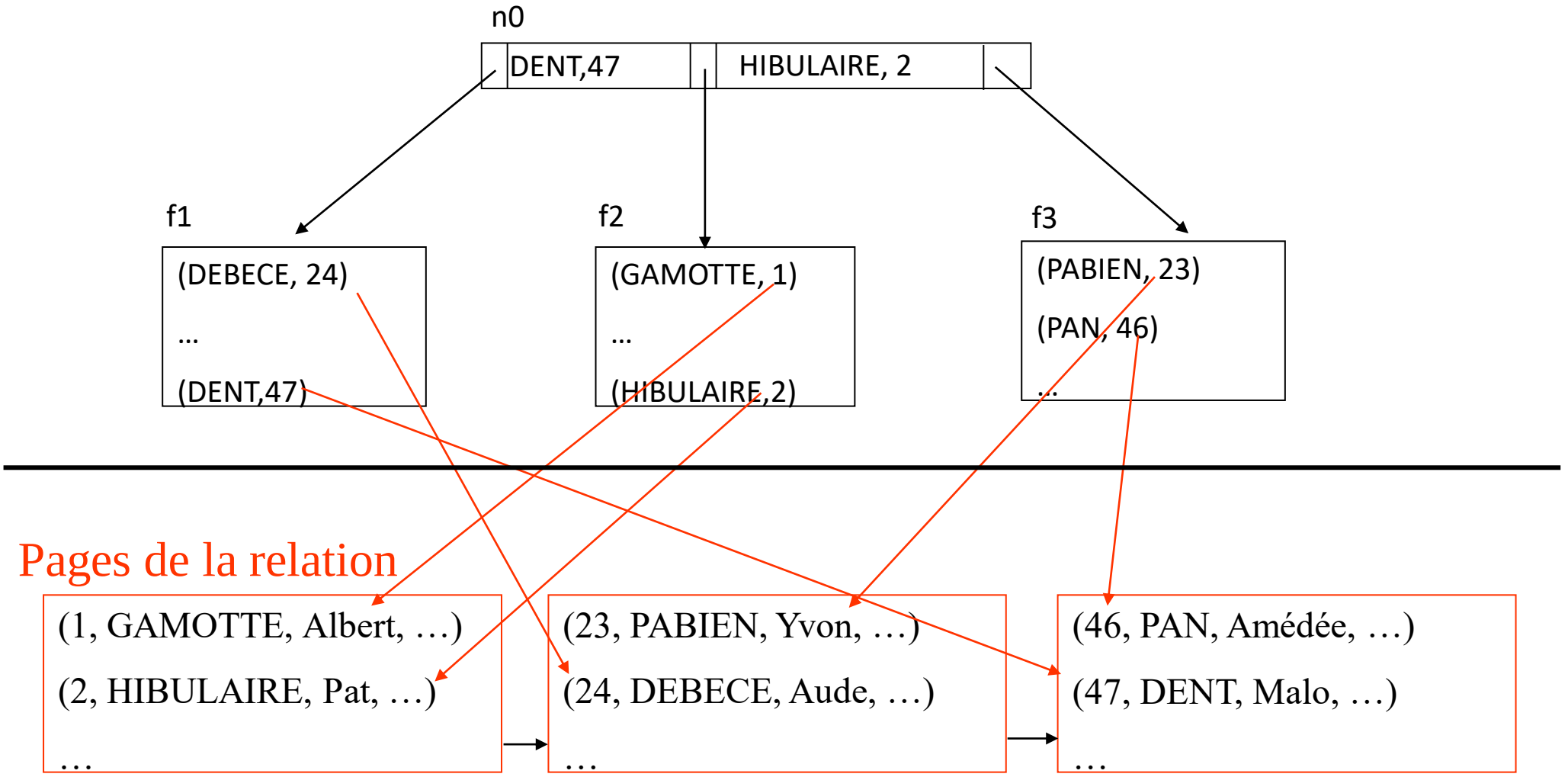
Fichier de hachage et un index sur un fichier de hachage



Arbre B+ : Exemple d'index primaire



Arbre B+ : Exemple d'index secondaire



Comparaison

- Les arbres-B 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.

- Les index hachés permettent en une seule E/S de retrouver un tuple dont on donne la valeur de la clé.

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.

Indexation dans Oracle

- Oracle propose plusieurs techniques d'indexation : arbres-B, arbres-B+ (par défaut), tables de hachage. Oracle maintient automatiquement les index, de manière transparente pour l'utilisateur.
- Par défaut, un arbre-B+ est créé sur la clé primaire de chaque table.
- Si l'utilisateur effectue des recherches sur des critères qui ne font pas partie de la clé, les recherches peuvent être optimisées en créant un nouvel index.

CREATE [UNIQUE] INDEX nom_index
ON nom_table (nom_colonne [**ASC | DESC**] [, nom_colonne …];

UNIQUE : une seule valeur de champ de clé

Exemples :

CREATE INDEX nomcli_ind **ON** client(nomcli);
CREATE INDEX com_ind **ON** commandes (datecom **DESC**, numcli **ASC**);

Indexation dans Oracle

- Pour créer une table en arbre-B, il faut l'indiquer à la création :

```
CREATE TABLE nom_table
        (Att1 typeAtt1,
        …
        PRIMARY KEY (Att1),
        ORGANIZATION INDEX);
```

- Tables de hachage :

```
CREATE CLUSTER HashFilms (id INTEGER)
        SIZE 500
        HASHKEYS 500;

CREATE TABLE Film (idFilm INTEGER, …) CLUSTER HashFilms (idFilm);
```

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.

Catalogue du système (1/2)

- Appelé également **dictionnaire de données**
- Contient la description des données de la base
  - ♦ Pour chaque relation :
    - – nom de la relation, identificateur du fichier et structure du fichier
    - – nom et domaine de chaque attribut
    - – nom des index
    - – contraintes d'intégrité
  - ♦ Pour chaque index :
    - – nom et structure de l'index
    - – attribut appartenant à la clé de recherche
  - ♦ Pour chaque vue :
    - – nom de la vue
    - – définition de la vue

Catalogue du système (2/2)

- Contient également des données statistiques
  - **Cardinalité de chaque relation**
  - **Nombre de pages de chaque relation**
  - **Nombre de valeurs distinctes de clé de recherche pour chaque index**
  - **Hauteur des index de structures arborescente**
  - **Valeur minimum et valeur maximum de chaque clé de recherche dans chaque index**
- Exemple sous Oracle : USER_ALL_TABLES, USER_CONSTRAINTS etc.

TD 2 : Indexation de données

BIBLIOGRAPHIE

Ouvrages de référence utilisés pour le cours :

- R. Ramakrishnan et J. Gehrke, *Database Management Systems*, Third Edition; McGraw-Hill, 2003
- H. Garcia Molina, J.D. Ullman et J. Widom, *Database System Implementation*, Prentice Hall, 2000
- H. Garcia Molina, J.D. Ullman et J. Widom, *Database Systems - The Complete Book*, Prentice Hall, 2002
- T. Connoly, C. Begg et A. Strachan, *Database Systems A Pratical Approach to Desigh, Implementation and Management*, 1998
- A. Silberschatz, H.F. Korth et S. Sudarshan, *Database System Concepts*, McGraw-Hill, 2002, version de 1996
- C.J. Date, *An Introduction aux bases de données*, 6ème édition, Thomson publishing, 1998
- R.A. El Masri et S.B. Navathe, *Fundamentals of Database Systems*, Prentice Hall
- G. Gardarin, *Bases de Données - objet/relationnel*, Eyrolles, 1999, + *Le client - serveur*, Eyrolles, 1996
- https://www.lamsade.dauphine.fr/~manouvri/
- https://laurent-audibert.developpez.com/Cours-BD/

Recherche par dichotomie (Brève révision)

Algorithme (pour un fichier de *n* blocs)

1. On lit le bloc situé au milieu du fichier.
2. Si on trouve l'enregistrement : terminé.
3. Sinon on sait qu'il est soit à droite (si la clé cherchée est supérieure à celles du bloc) ou à gauche : on recommence en (1).

•Coût (au pire) nombre de fois où on peut diviser *n* en 2.

•Soit le plus petit *m* tel que $2^m \geq n$, donc $m = \lceil \log_2 n \rceil$