

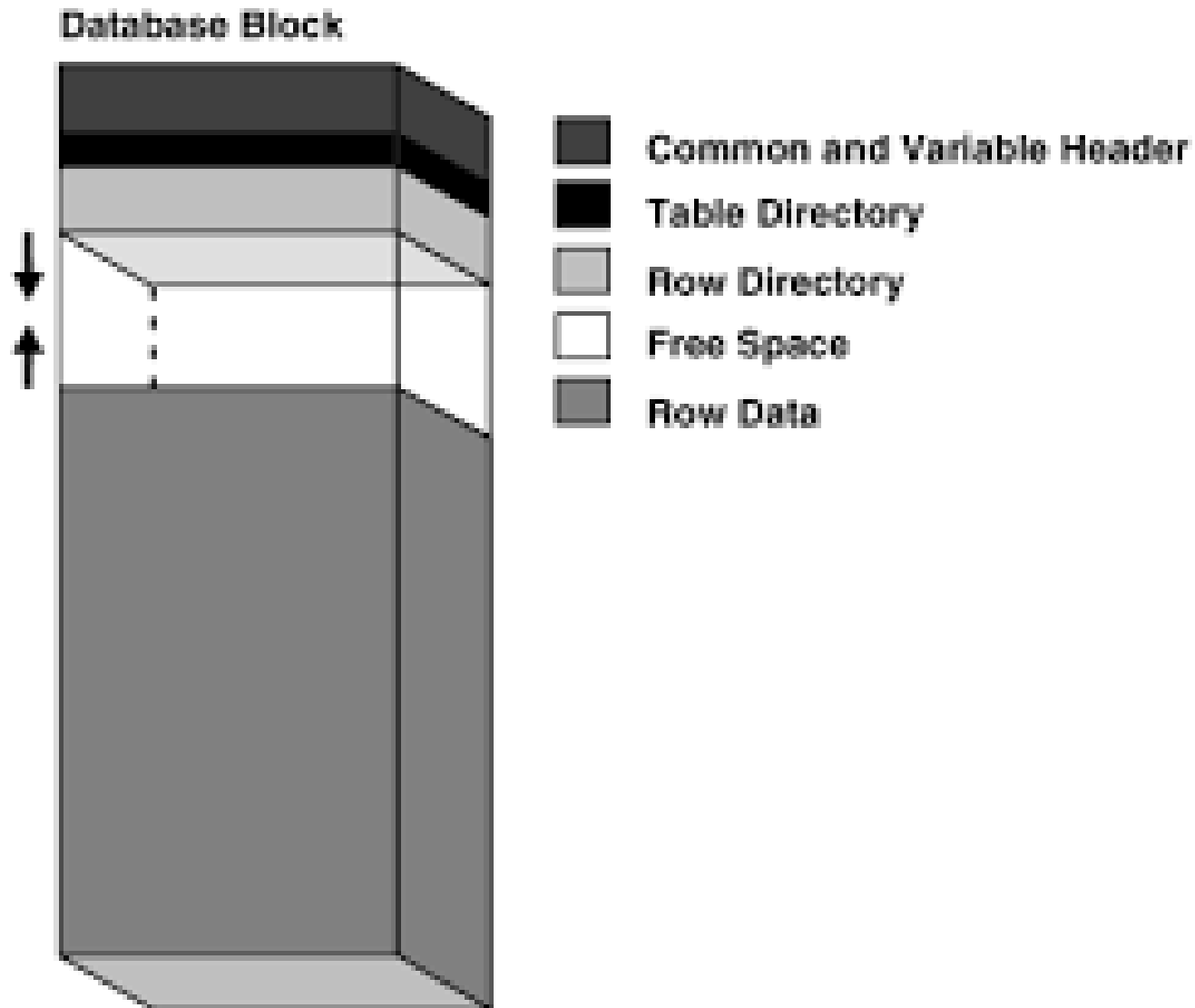
Efficacité des accès aux données et mécanismes d'index

Nombreux emprunts au livre “Database Systems”
(Hector Garcia-Molina)

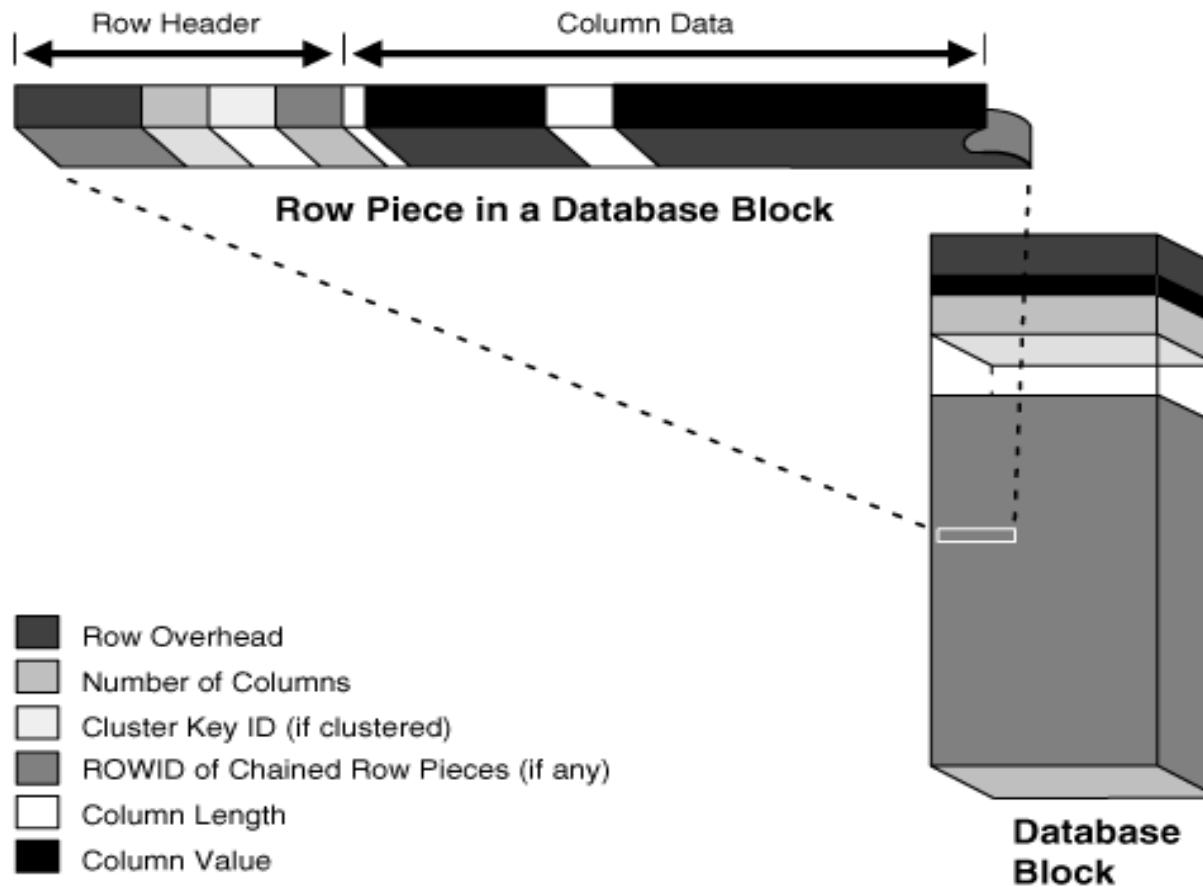
Plan global

- Retour sur le bloc Oracle
- Organisations séquentielles indexées (ISAM)
- **Arbres balancés (B-trees et variantes)**
- Table de hachage
- Index bitmap

Bloc de données Oracle

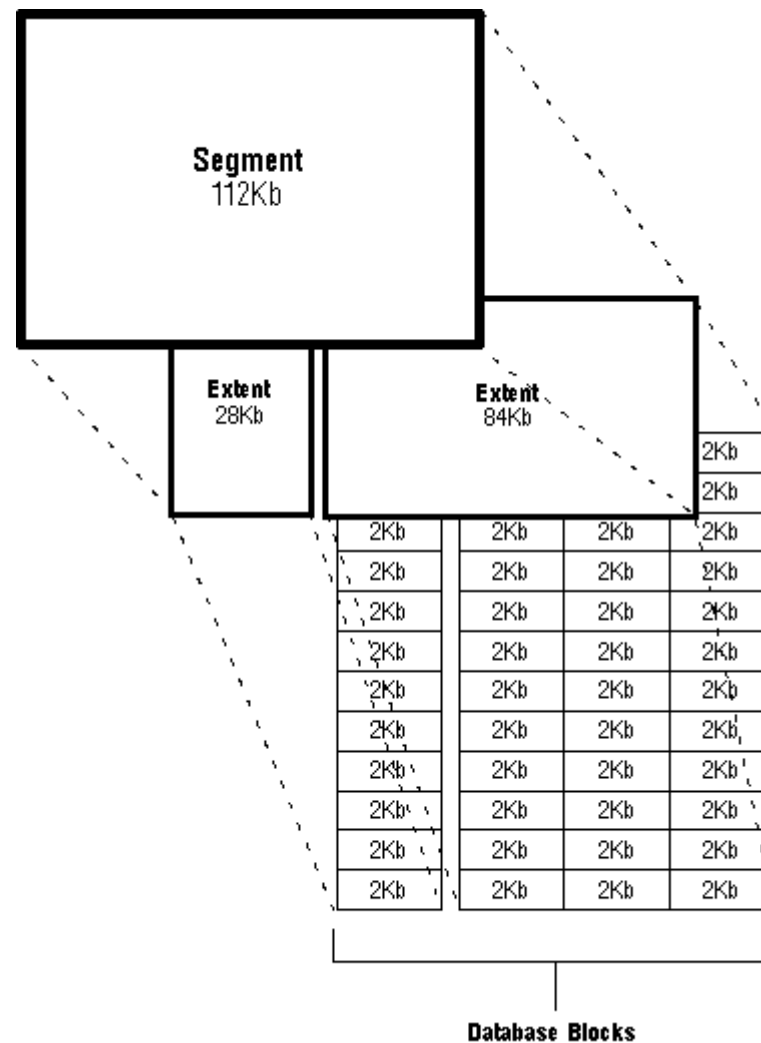


Structure d'un enregistrement

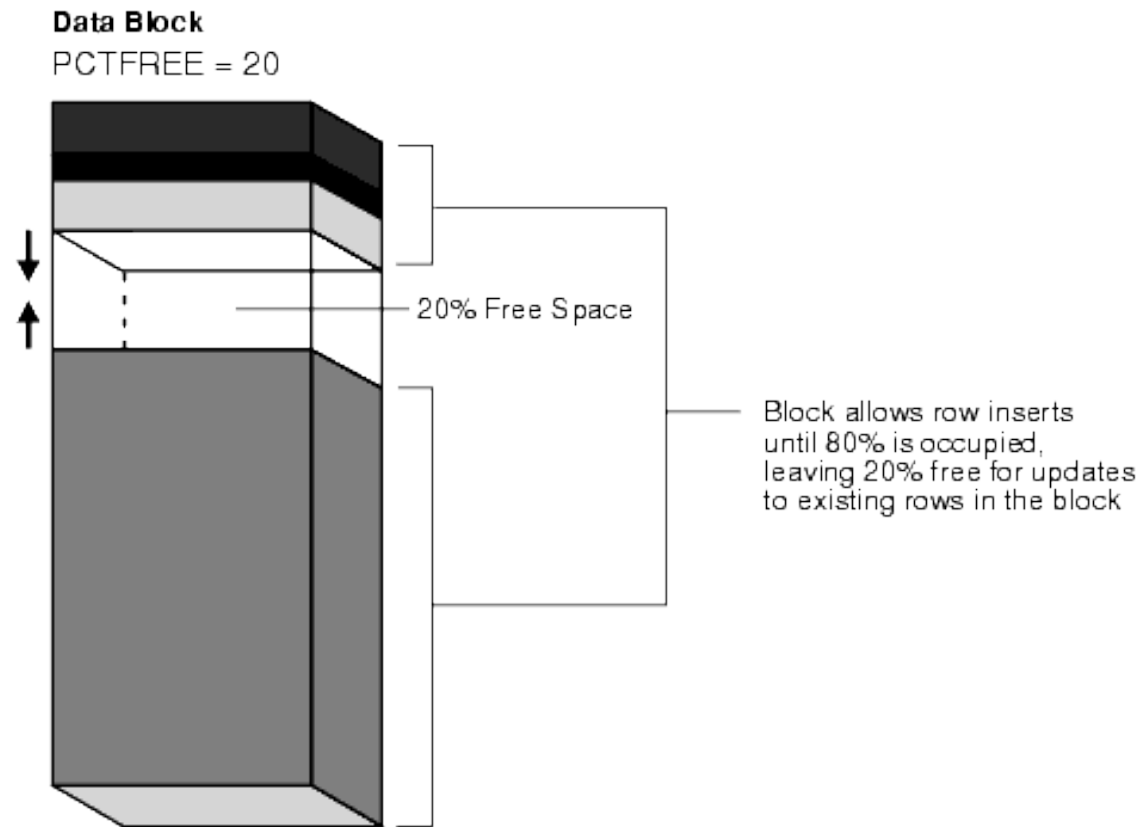


Rowid : object_number.relative_file_number.block_number.row_number
(infos accessibles avec le paquetage dbms_rowid)

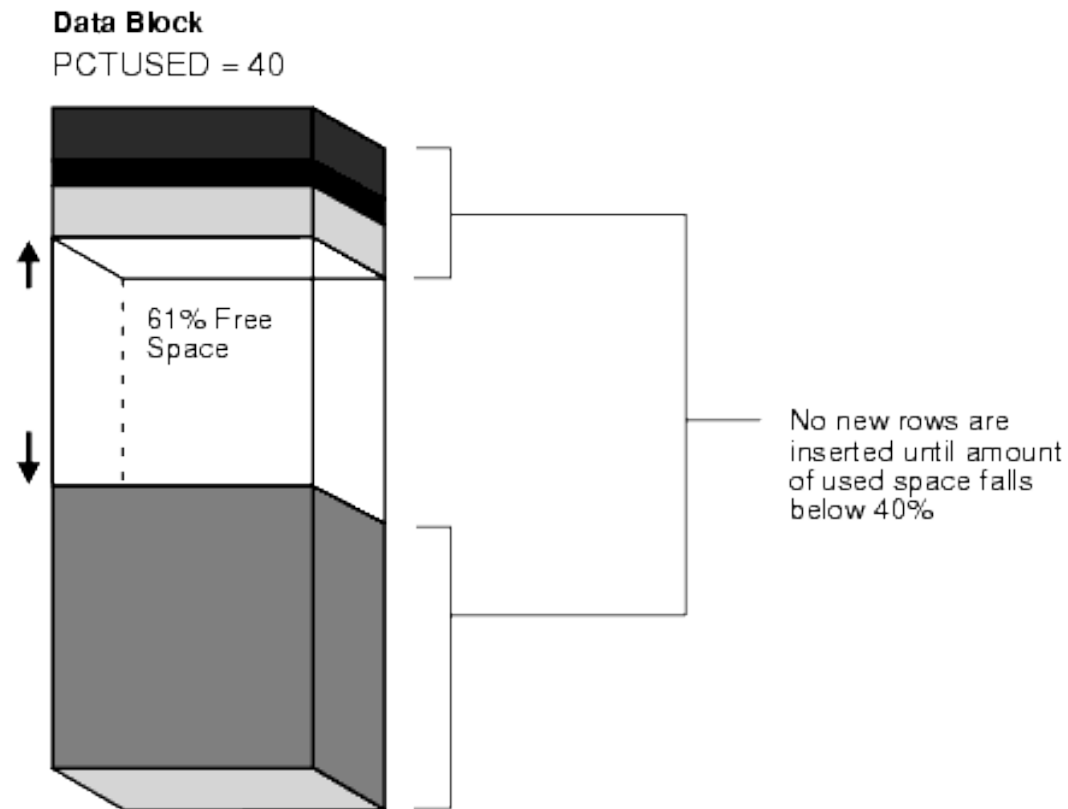
Organisation logique : bloc/extent/segment/tablespace



De la place pour les mises à jour



Avant toute nouvelle insertion



User_tables et stockage physique

```
isa@gandalf: ~  
Fichier Édition Affichage Rechercher Terminal Aide  
-Heure de la dernière connexion réussie : Lun. Oct. 05 2020 11:49:28 +02:00  
  
Connecte a :  
Oracle Database 12c Enterprise Edition Release 12.2.0.1.0 - 64bit Production  
  
>SQL> desc user_tables  
  Nom                                NULL ?    Type  
-----  
TABLE_NAME                          NOT NULL  VARCHAR2(128)  
TABLESPACE_NAME                     VARCHAR2(30)  
CLUSTER_NAME                        VARCHAR2(128)  
IOT_NAME                            VARCHAR2(128)  
STATUS                              VARCHAR2(8)  
PCT_FREE                            NUMBER  
PCT_USED                            NUMBER  
INI_TRANS                           NUMBER  
MAX_TRANS                           NUMBER  
INITIAL_EXTENT                       NUMBER  
NEXT_EXTENT                         NUMBER  
MIN_EXTENTS                         NUMBER  
MAX_EXTENTS                         NUMBER  
PCT_INCREASE                        NUMBER  
FREELISTS                           NUMBER  
FREELIST_GROUPS                     NUMBER  
LOGGING                             VARCHAR2(3)  
BACKED_UP                           VARCHAR2(1)  
NUM_ROWS                            NUMBER  
BLOCKS                              NUMBER  
EMPTY_BLOCKS                        NUMBER  
AVG_SPACE                            NUMBER  
CHAIN_CNT                           NUMBER  
TRANS                               NUMBER
```


Structures d'accès facilitatrices

Comment ne parcourir qu'une fraction de blocs au regard de la requête ?

Comment ne parcourir qu'une fraction de blocs au regard de la requête ?

Exemple – recherche linéaire

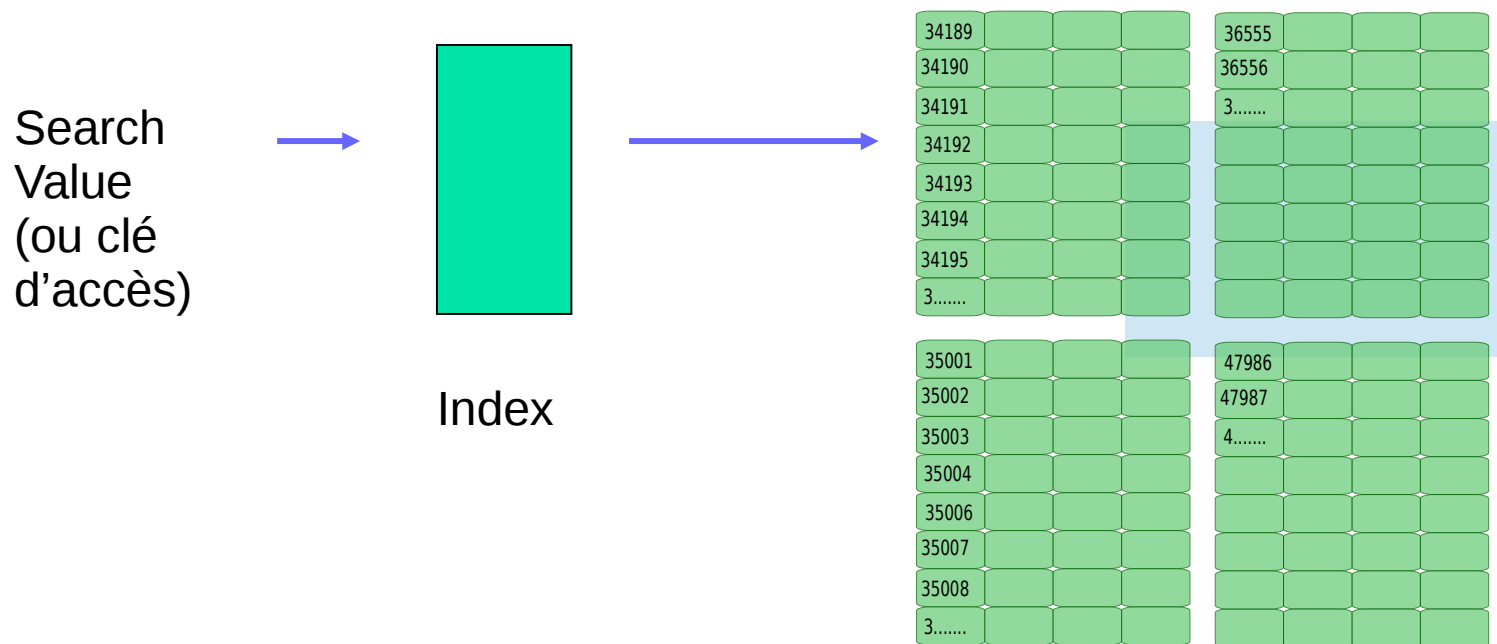
```
select * from commune where code_Insee = '34192';
```

- Lire tuple après tuple
- En moyenne lecture de 50 % des tuples et donc de 50% des blocs ($B/2$)
- 100% si la valeur n'existe pas
- ***$B/2$ coûteux surtout si B élevé***

codeInsee				
34189				
34190				
34191				
34192				
34193				
34194				
34195				
3.....				

10

Idée : une structure complémentaire pour accélérer la localisation des tuples cibles = INDEX



Index

- Définition : structure de données, avec en entrée une propriété (search key), et qui permet de retrouver rapidement les enregistrements possédant cette propriété
- Un index est construit au travers de champs spécifiés dans un fichier
 - **search key** : chaque valeur possible pour cette clé est triée et associée à une liste de pointeurs vers les tuples corrélés
 - Rechercher avec un index a pour résultat de retrouver une liste d'adresses
- Il restera nécessaire de parcourir des blocs et des enregistrements mais :
- enregistrements d'index plus petits et donc plus aisés à monter en mémoire vive
- clés triées donc une recherche dichotomique est possible (et non plus linéaire) : complexité logarithmique

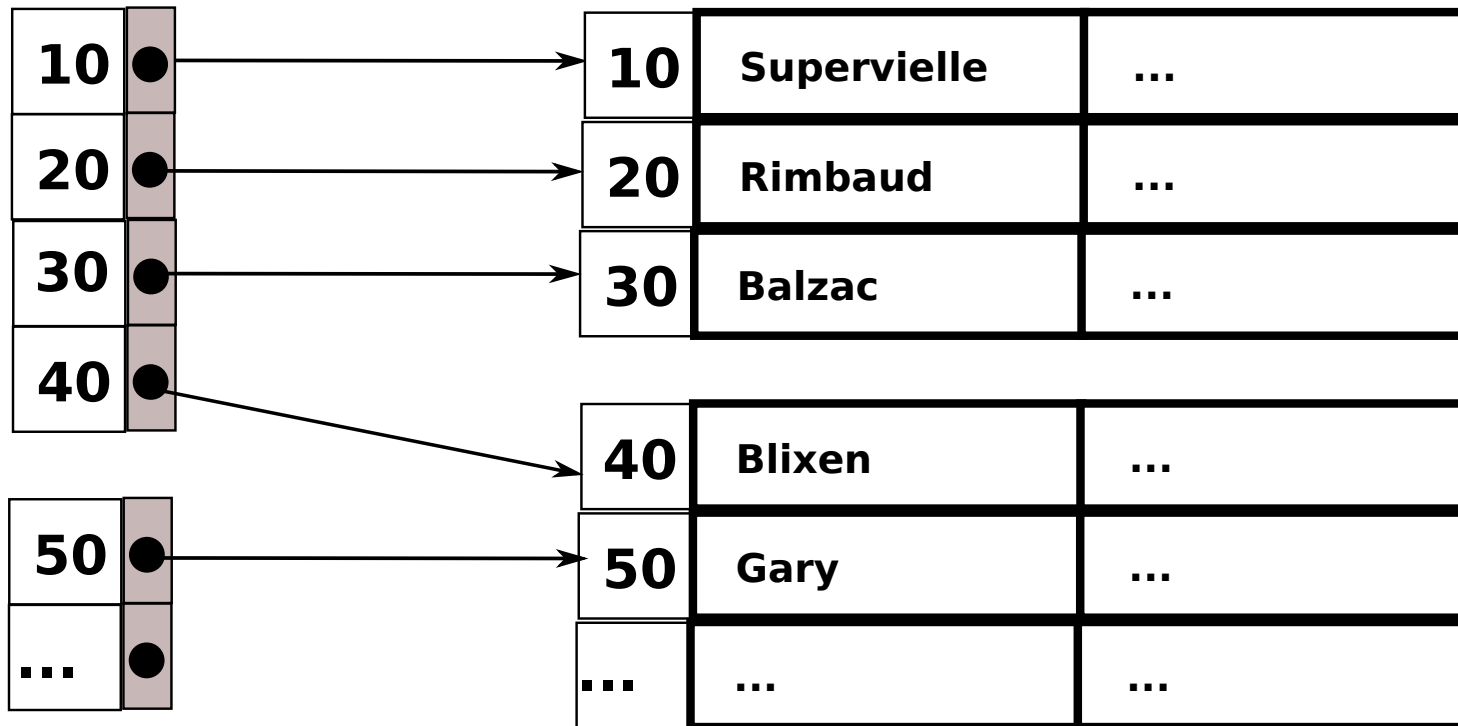
Organisations Séquentielles indexées

ISAM (Indexed Sequential Access Method) IBM 1966

Séquentiel indexé : index dense

blocs d'index (index trié et dense)

blocs de données (trié)



<clé/pointeur>

Dense : toutes les valeurs de clé représentées

Exemple 1 avec index dense

Table de 1 000 000 de tuples avec 10 tuples (de 400 octets) par bloc de 4 Ko (**100 000 blocs**)

Espace mémoire pour la table : **400 Mo** (100 000 * 4 Ko)

Espace mémoire pour l'index : taille de la clé 30 octets et taille du pointeur 10 octets : **40 Mo** (1 000 000 * 40) et **10 000 blocs**

Si blocs d'index en mémoire vive

- Recherche sur la valeur d'une clé :

$\log_2(10\,000) = \ln(10\,000)/\ln(2) = 13.28..$ et donc 14 blocs à

parcourir + une opération d'entrée / sortie pour aller chercher le bloc de l'enregistrement recherché

Séquentiel indexé : index creux

blocs d'index (index trié et creux)

10	●
40	●
70	●
100	●

110	●
...	●

<clé/pointeur>

blocs de données (trié)

10	Supervielle	...
20	Rimbaud	...
30	Balzac	...

40	Blixen	...
50	Gary	...
...

Creux (sparse) : certaines valeurs de clé représentées => en général une valeur par bloc

Exemple 1 avec index creux

Table de **1 000 000** de tuples

avec 10 tuples par bloc de 4 Ko (100 000 blocs)

Espace mémoire pour la table : **400 Mo** (100 000 * 4 Ko)

Espace mémoire pour l'index creux : taille de la clé 30 octets et taille du pointeur 10 octets : 1 seule entrée par bloc pour les 100 000 blocs 4 Mo (100 000 * 40) et **1 000 blocs** => gain en terme de place pour la RAM

Si blocs d'index en mémoire vive

- Recherche sur la valeur d'une clé :

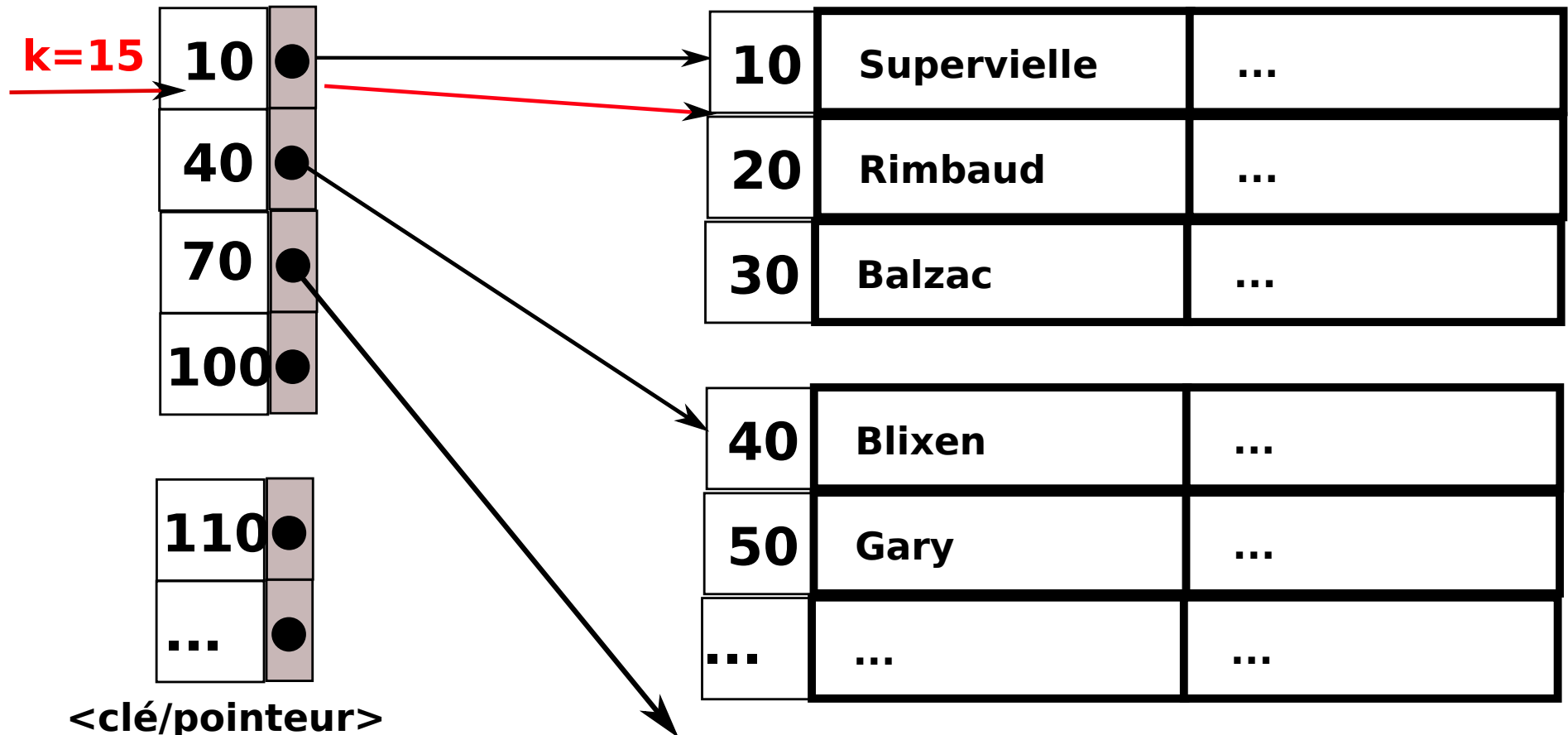
$\log_2(1\,000) = \ln(1\,000)/\ln(2) = 9.96..$ et donc 10 blocs à

parcourir + une opération d'entrée / sortie pour aller chercher le bloc de l'enregistrement recherché

Séquentiel indexé : recherche

recherche sur valeur de clé = 15

blocs de données (trié)



Ex : search key = 15

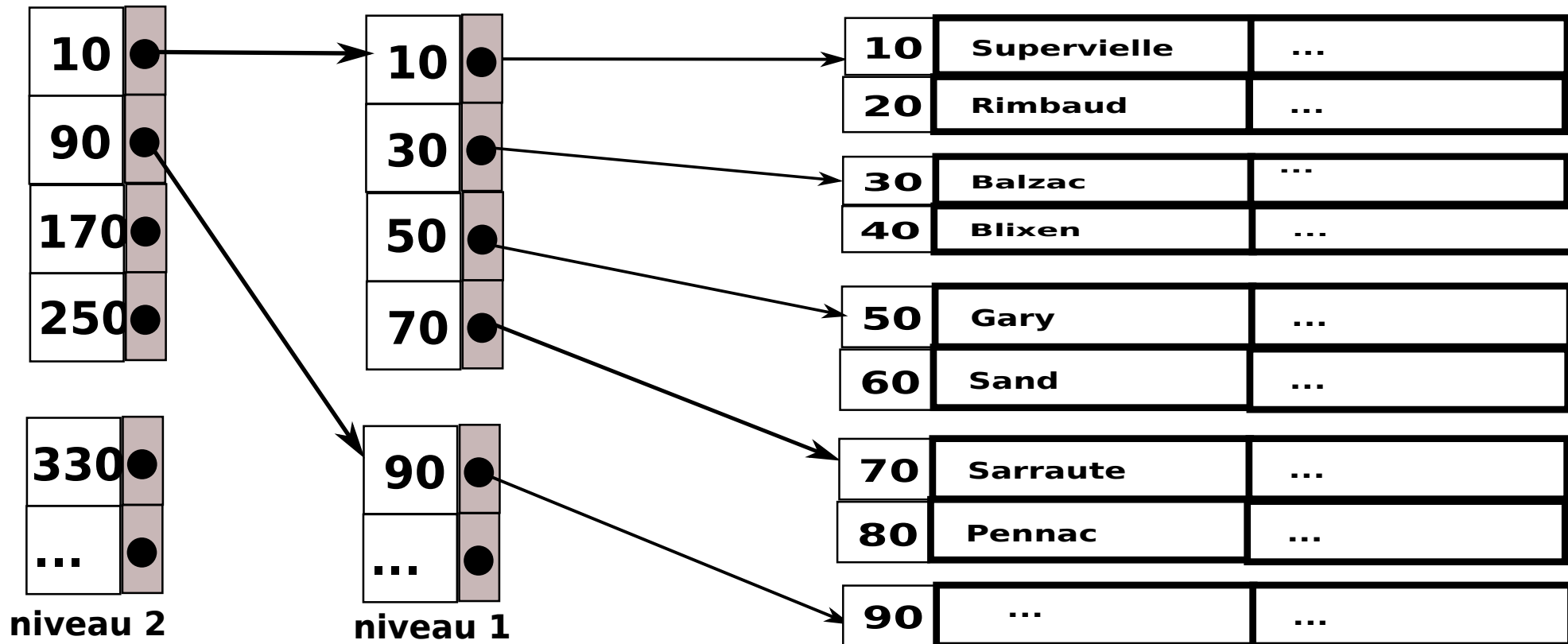
Parcours : recherche de la plus grande valeur ≤ 15 ici 10

Désavantage : si la valeur n'existe pas, il faut quand même parcourir le bloc d'index et effectuer l'opération d'entrée/sortie

Séquentiel indexé : multi-niveaux

blocs d'index à deux niveaux

blocs de données (trié)



Idée : poser un index sur un index

Le niveau 1 d'index peut être dense, par contre le niveau 2 doit être creux, sinon sans intérêt

Exemple 1 avec index multi-niveaux

Table de 1 000 000 de tuples avec 10 tuples par bloc de 4 Ko octets (100 000 blocs)

Espace mémoire pour la table : 400 Mo (1 000 000 * 400)

Espace mémoire pour l'index creux niveau 1 : 1 000 blocs

Espace mémoire pour l'index creux niveau 2 : 10 blocs => gain accru en terme de place pour la RAM

Recherche sur la valeur d'une clé :

$\log_2(10) = \ln(10)/\ln(2) = 3,32..$ et donc 4 blocs à parcourir + deux opérations d'entrée / sortie pour aller chercher le bloc de l'index niveau 1 et de l'enregistrement recherché

Arbres équilibrés (B-Tree)

A plus de deux niveaux : le choix se porte sur les B-Tree (B pour Balanced) Bayer, R & McCreight, E. (1971)

ISAM nécessite que le fichier de données soit trié, ce qui rend les insertions coûteuses (blocs de débordement si nécessaires et nécessité de réorganisation fréquente)

Notions autour du B-Tree

Structure de données maintenant dynamiquement un ensemble d'éléments afin que l'arbre soit équilibré :

Equilibre important pour les opérations usuelles sur une table : recherche, insertion, suppression

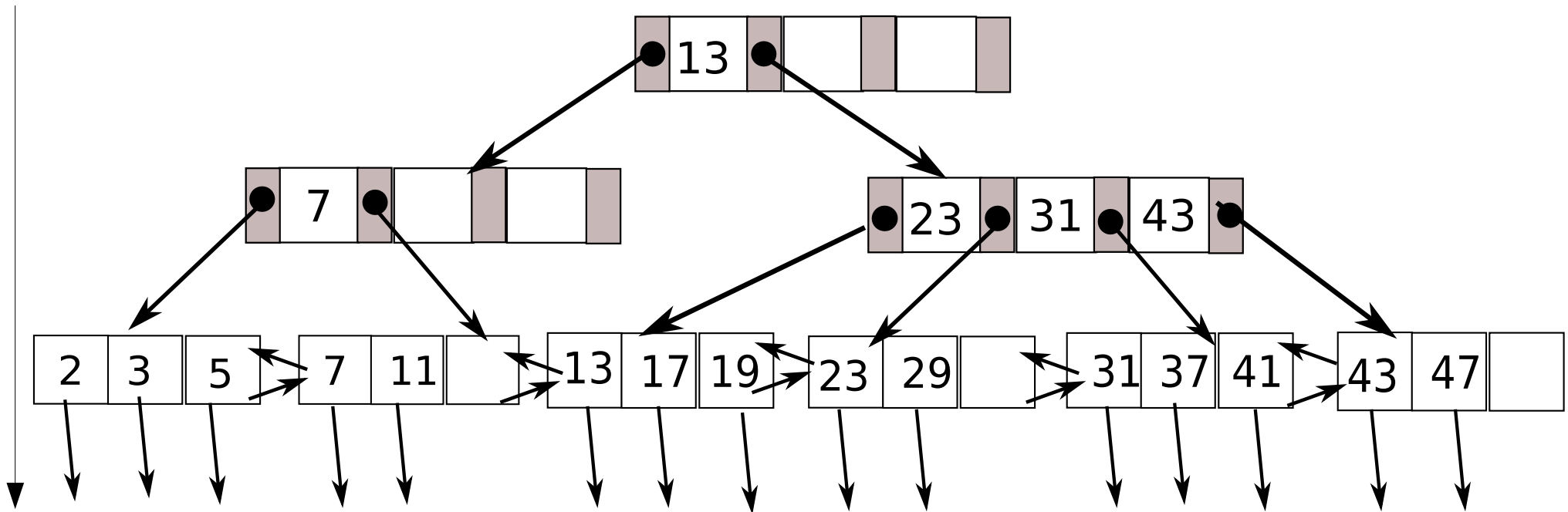
- chaque noeud branche de l'arbre : **n clés et n+1 pointeurs**,
- un noeud (sauf la racine) est **de à moitié plein à plein**
- pour les noeuds feuilles : au même niveau et contiennent les clés et les pointeurs sur les données

Plusieurs variants d'arbres – **B⁺-tree et B*tree**

- B-arbre : noeuds intermédiaires à même de contenir des pointeurs sur des données
- **B+arbre, seules les feuilles (doublement chaînées) contiennent des pointeurs sur les données**
- B*arbre les noeuds sont au moins à 2/3 plein

Arbre de hauteur 2 ou 3 (avec ou sans le niveau feuilles)

Hauteur constante

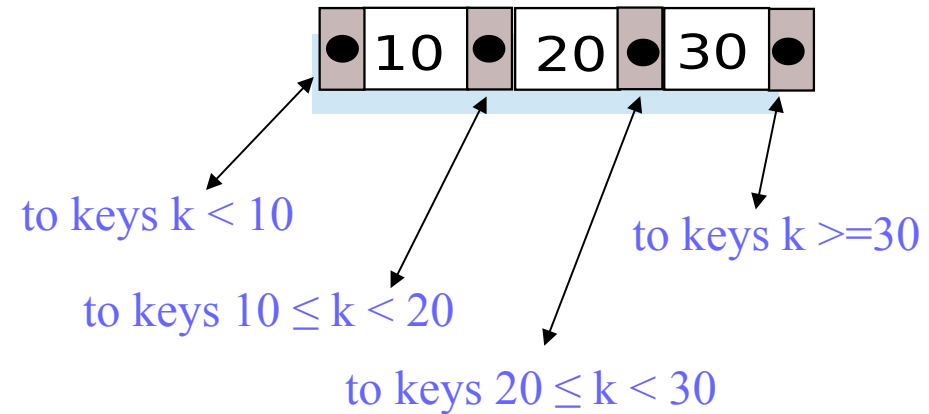


Des détails sur B+Tree

Noeuds branches

(pointeurs vers les noeuds fils)

- à gauche pointeur vers un fils avec une valeur de **clé <**
- à droite pointeur vers un fils avec une valeur de **clé >=**

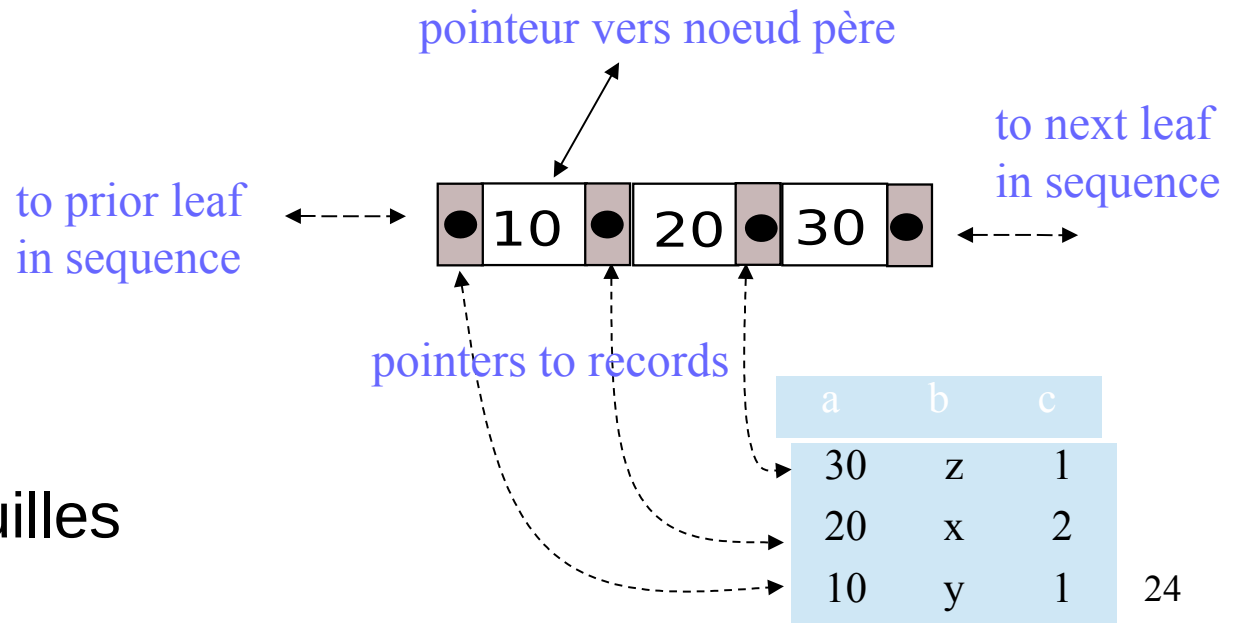


Noeuds feuilles

(n pointeurs de données,

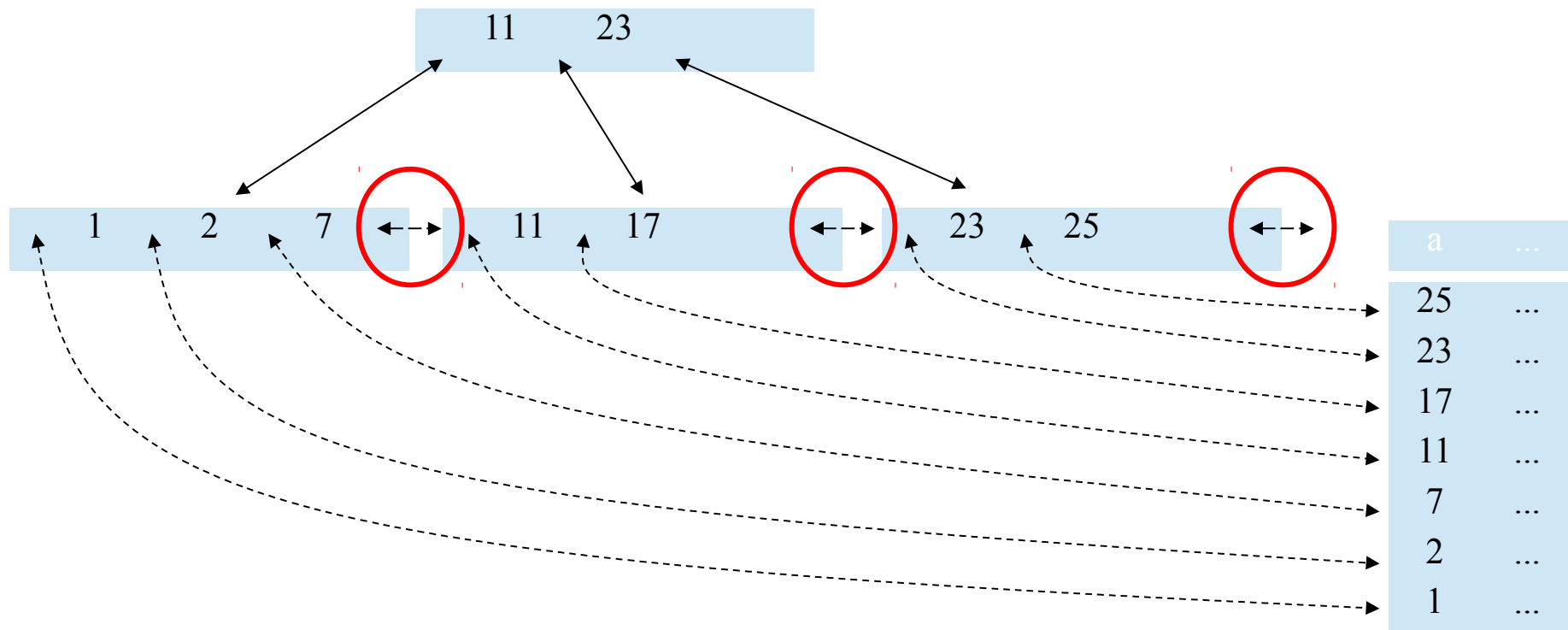
1 pointeur gauche,

1 pointeur droit vers les feuilles voisines)



Exemple d'arbre à 2 niveaux

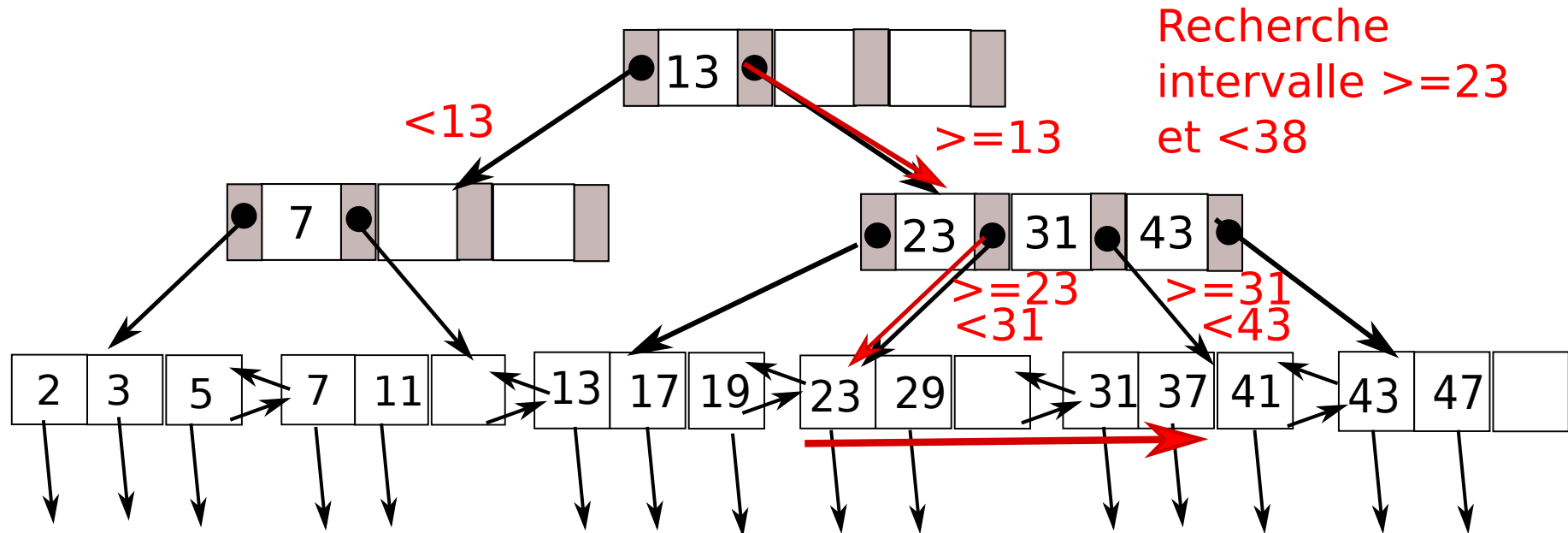
Remarque : les feuilles chaînées vont permettre la lecture séquentielle et donc la la recherche sur intervalle (utile aussi pour les inégalités)



Opérations sur l'index

- ✓ Recherche : parcours top-down et comparaison
 - Branches : les pointeurs sont exploités de branche en branche :
 - Si clé = $K \rightarrow$ choix du pointeur gauche si $K < K' \rightarrow$ choix du pointeur droit si $K \geq K'$
 - Feuilles :
 - Index dense :
 - Si la n ème clé = K alors le n ème pointeur pointe sur l'enregistrement recherché
 - Si la valeur de clé K est absente alors l'enregistrement recherché n'existe pas
 - Index creux :
 - Trouver la plus grande des valeurs juste inférieure ou égale à K
 - Retrouver le bloc feuille pointé par cette valeur
 - Rechercher dans le bloc de données pour cet enregistrement

Exemple recherche sur intervalle



Taille de l'arbre

- Exemple
 - nombre de valeurs n max qu'un bloc de 4 Ko peut héberger si la clé est sur 4 octets et les pointeurs sur 8
- $4(n) + 8(n+1) \leq 4096 \Rightarrow 12(n) = 4088 \rightarrow n \text{ (nbre valeurs clés)} = 340$

Si l'on considère que chaque noeud est à 2/3 plein (254 valeurs et 255 pointeurs) et que la racine comme chaque noeud fils a 255 fils :

Au niveau 2 on a $(255)^2$ enregistrements possibles et

$(255)^3$ enregistrements soit plus de 16 millions pour un arbre d'une hauteur de 3

Ordre m (si le noeud contient n entrées : $n / 2$)

Hauteur h :

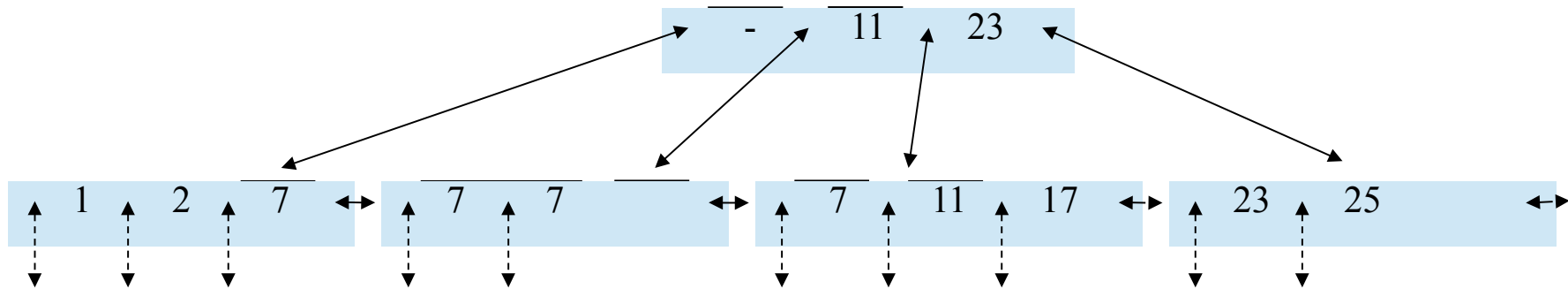
$$\log_{2m}(N) \leq h \leq \log_m(N)$$

N étant le nombre de tuples de l'index

Efficacité de l'arbre

- Recherche, Insertion, Délétion :
- Parcours de l'arbre de la racine aux feuilles :
pour une hauteur de 3 :
3 entrées/sorties + 1 I/O de plus (lookup) ou 2 (insertion/délétion)
- si l'arbre est totalement en mémoire vive :
parcours de 3 blocs et 1 à 2 entrées/sorties
- Meilleur cas : $\log_m(n) + 1$ avec m nombre de clés/2 (ordre) et n nombre d'enregistrements

Index non unique : duplication de valeurs au niveau feuilles



Remarque 1:

noeud branche pointe sur la première occurrence de la clé dupliquée et ensuite les autres occurrences sont lues séquentiellement

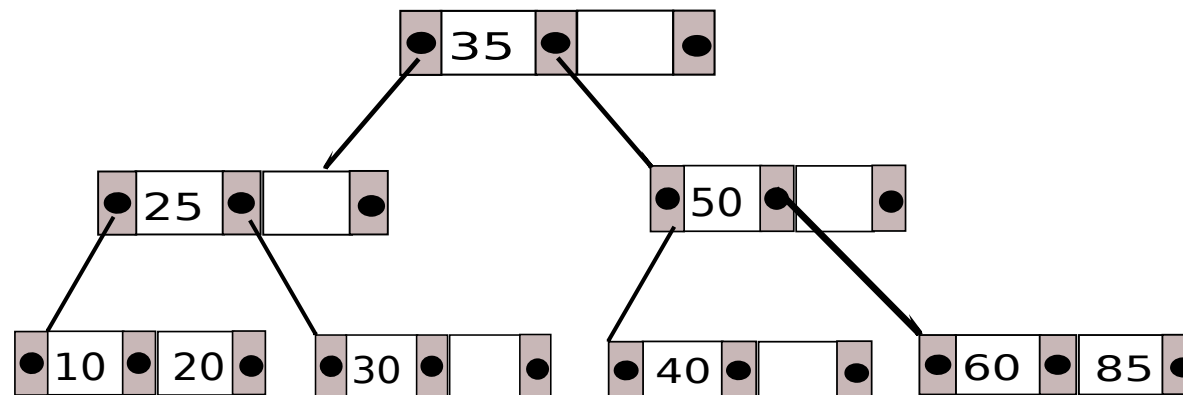
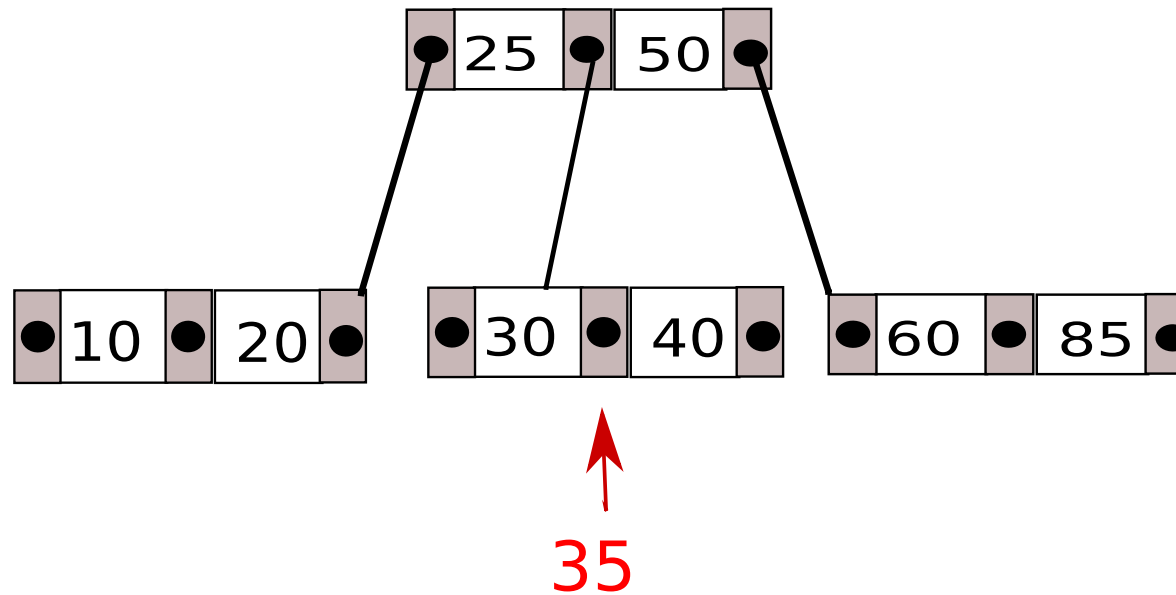
Remarque 2:

Dans certains cas, la valeur du noeud branche est non renseignée (null) : ici par exemple pour 7

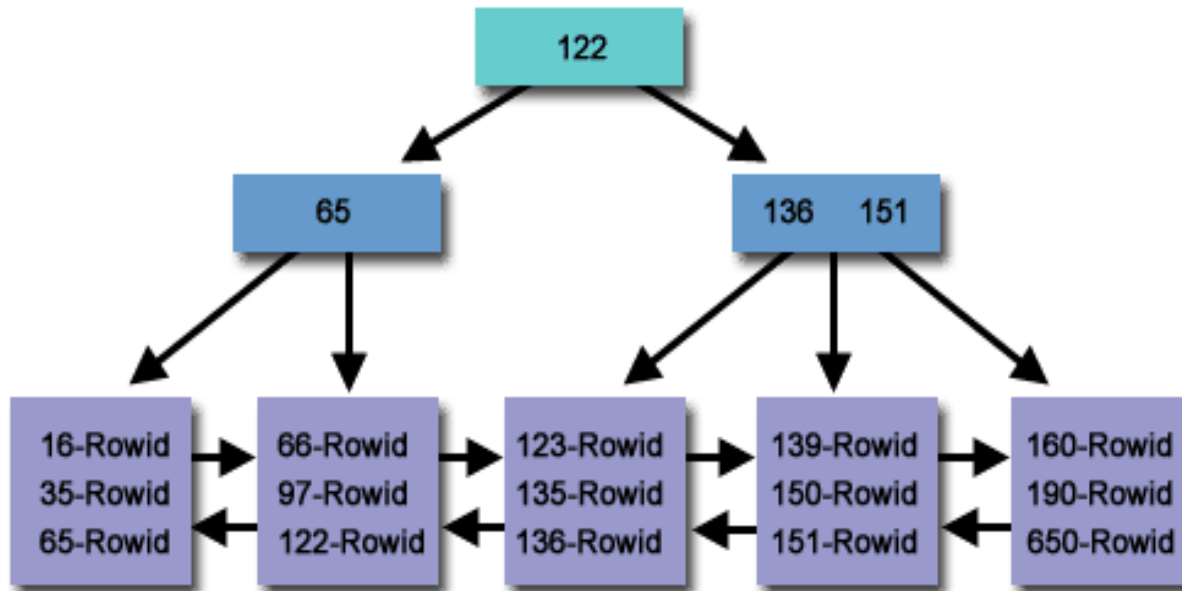
Insertion : Algorithme d'ajout

1. Trouver dans l'arbre la feuille où l'élément pourrait être ajouté.
 2. Si le noeud contient moins de valeurs que le nombre maximum autorisé par noeud, alors ajouter l'élément en respectant le tri
 3. Sinon, la feuille est alors éclatée :
 - (a) L'élément médian est choisi (nouveau père du sous arbre) parmi tous les éléments présents y compris le nouveau: élément médian = $(k+1)/2^{\text{ème}}$.
 - (b) Les valeurs $<$ au médian \rightarrow fils gauche, et les valeurs $>$ \rightarrow fils droit.
 - (c) L'élément médian (père du sous-arbre) est ajouté au noeud parent .
- Un nouvel éclatement peut alors en résulter (continuer ainsi jusqu'à la racine)

Exemple d'insertion



B+Tree Oracle

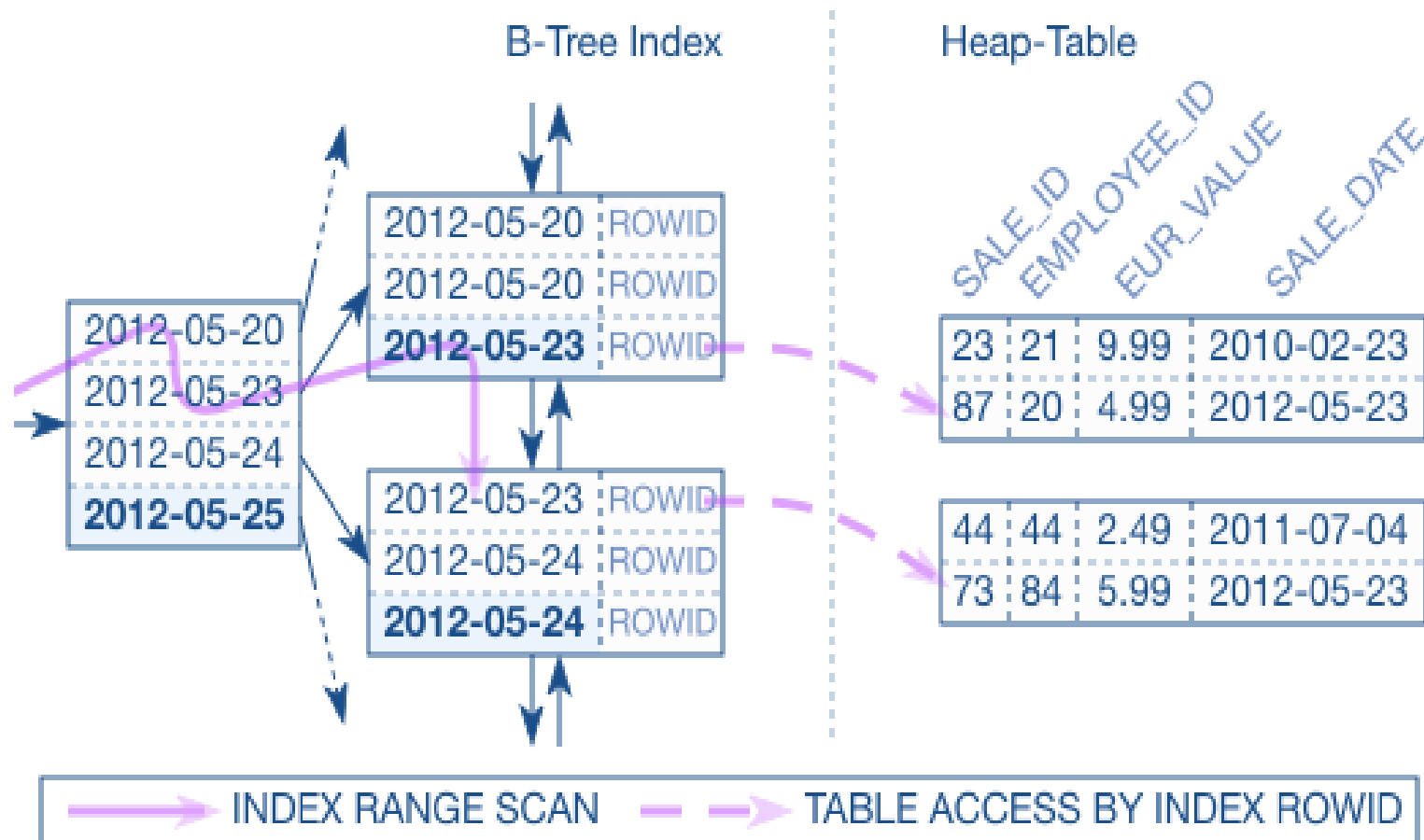


Remarques:

Index dense et fichier de données non trié (structure en tas ou heap file)

Enregistrement feuille = clé d'index + rowid

B+Tree Oracle (exemple index secondaire et non plaçant)



Définition implicite d'index B+-Tree

Lors de la création d'une contrainte

- De **clé primaire** (unique et not null)
- d'**unicité** (unique)
- L'index unique se nomme de la même manière que la contrainte
- Notion d'**index plaçant** : quand l'index préside aussi l'organisation des blocs de données

Définition explicite d'index B-Tree(Oracle)

```
create unique index com_idx on commune
(code_insee);

create index com_idx on commune(lower(nom_com));

alter index com_idx disable; (que les index sur
fonction)

drop index com_idx;

- rendre un index inutilisable

alter index commune_pk unusable;

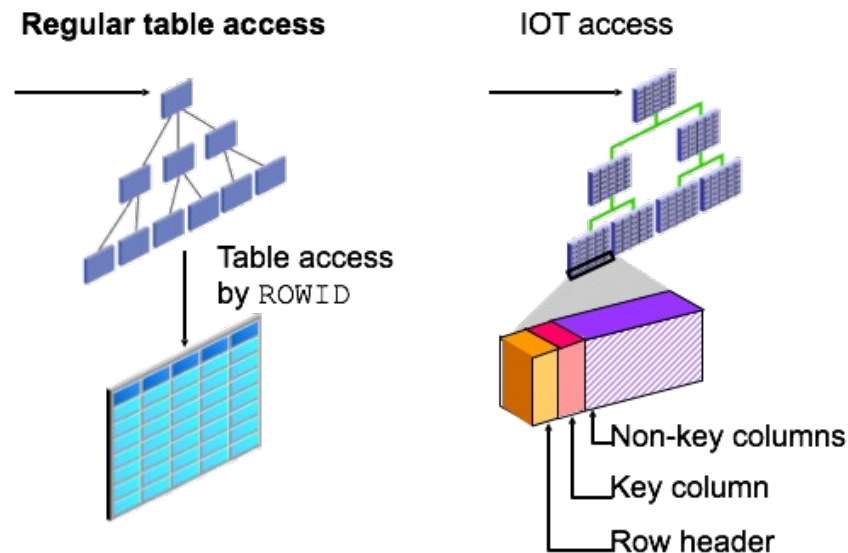
-- le reconstruire pour le rendre à nouveau
valide

    alter index commune_pk rebuild ;
```

Table organisée en index B-Tree(Oracle)

```
create table nomTable  
(attr1 datatype1, attr2 datatype2, ...)  
segment creation immediate  
organization index;
```

Table Types
Partition
> IOT
Cluster
DBA Tasks



Index Bitmap

Quand le domaine de valeurs d'un attribut se réduit à quelques valeurs

Souvent mobilisé au sein des entrepôts de données

Exemple : Genre sur EMP

Num	Nom	Genre	Fonction
1	Martin	M	ingénieur
2	Dupond	F	président
3	Dupont	M	commercial
4	Dubois	F	ingénieur

	F	M
1	0	1
2	1	0
3	0	1
4	1	0


Index bitmap sur Genre = matrice de bits (0/1)

Cardinalité : nombre valeurs distinctes
attributs (select distinct genre from emp)

Vecteurs de bits pour F et M

Exemple Genre sur EMP

Num	Nom	Genre	Fonction
1	Martin	M	ingénieur
2	Dupond	F	président
3	Dupont	M	commercial
4	Dubois	F	ingénieur



	F	M
1	0	1
2	1	0
3	0	1
4	1	0

Espace mémoire nécessaire plus que limité

Taille d'un vecteur en octets : $\text{nbre tuples} / 8$
codage du vecteur 8 bits par 8 bits et

Si taille du bloc de 8Ko :

Taille de l'index en blocs :
 $\text{cardinalité attribut} * (\text{nbre de tuples} / (8 * 8192))$

Autre exemple Fonction sur EMP

Num	Nom	Genre	Fonction
1	Martin	M	ingénieur
2	Dupond	F	président
3	Dupont	M	commercial
4	Dubois	F	ingénieur

	ingénieur	président	commercial
1	1	0	0
2	0	1	0
3	0	0	1
4	1	0	0

Create bitmap index on Emp (fonction) ;

Requêtes agrégats

```
SELECT count(*) FROM EMP  
WHERE fonction = 'président' ;
```

Revient à compter les 1 dans le vecteur qui correspond à président

Autres exemples de requêtes

Num	Nom	Genre	Fonction
1	Martin	M	ingénieur
2	Dupond	F	président
3	Dupont	M	commercial
4	Dubois	F	ingénieur

	ingénieur	président	commercial
1	1	0	0
2	0	1	0
3	0	0	1
4	1	0	0

Requêtes agrégats

```
SELECT count(*) FROM EMP  
WHERE fonction in  
(‘commercial’,‘président’) ;
```

```
SELECT max(count(*)) FROM EMP  
group by fonction ;
```

```
SELECT avg(salaire) FROM EMP  
group by fonction ;
```

Hash Index

Recherche sélective exacte

Index et table de hachage

Clé (Num) = 31

$h(\text{Num}) = 31$
 $\text{mod } 3 = 1$

Fonction h qui appliquée
à la clé
Fournit l'adresse du tuple

Problème des collisions
sur les adresses

Recherche uniquement sur des valeurs
exactes (valeurs identiques possibles)
(exclut donc recherche par intervalle et
opérateur !=)

partition				
0	3	Dupont	M	commercial
	6	Dulac	F	designer
	9	Duchemin	F	graphiste
1	1	Martin	M	ingénieur
	10	Dupond	F	président
	31	Doré	M	ingénieur
2	2	Duvivier	F	administrateur
	11	Ducarre	M	chef projet

Index et table de hachage

Différents modes de partitionnement :

Hachage

Intervalles de valeurs

Liste de valeurs

Exemple

```
CREATE TABLE EMP1 (Num integer, Nom varchar(10), Genre varchar(1),  
Fonction varchar(15))  
PARTITION BY HASH (Num) PARTITIONS 20 ;
```

En Oracle fonction avec une distribution uniforme ORA_HASH

Index et table de hachage

Autre Exemple

```
CREATE CLUSTER emp_cluster (num NUMBER(5,0))  
  SIZE 500  
  HASH IS num HASHKEYS 1500;
```

```
CREATE TABLE employe (  
  num NUMBER(5,0) PRIMARY KEY,  
  name varchar(15))  
  CLUSTER emp_cluster (num);
```

Création préalable d'un cluster de tables
SIZE = taille estimée d'un tuple
HASHKEYS = nombre de tuples