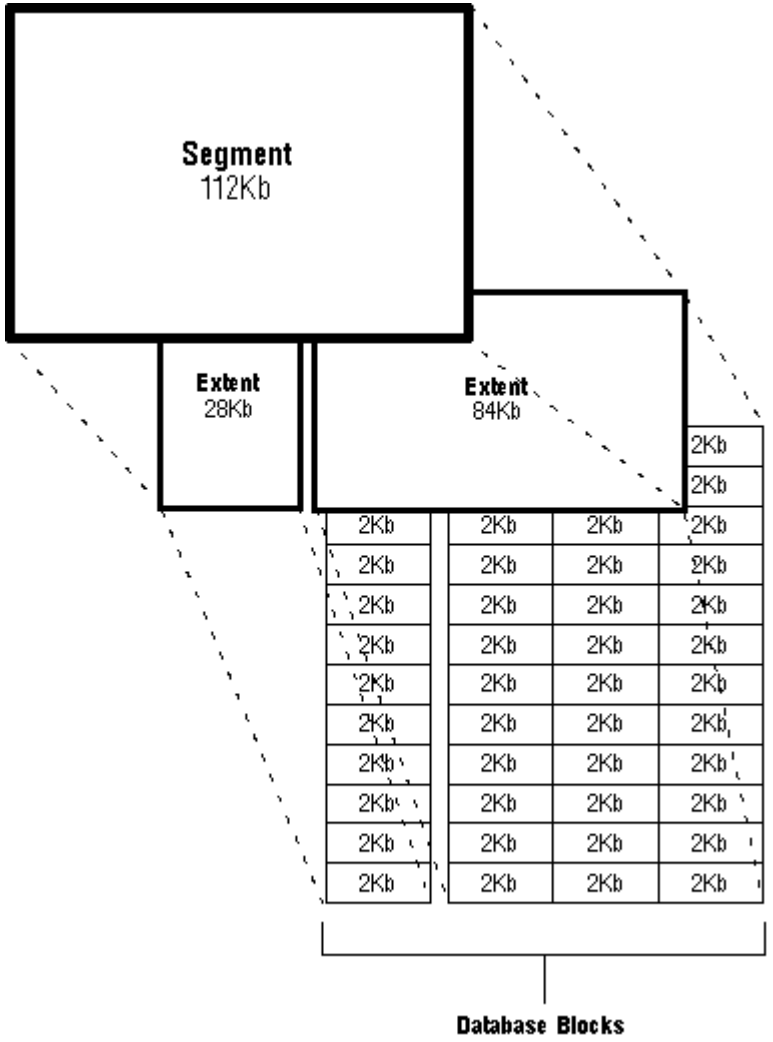
Efficacité des accès aux données et mécanismes d'index

Nombreux emprunts à Database Systems (Hector Garcia-Molina)

Plan global

- Organisations séquentielles indexées (ISAM)
- Arbres balancés (B-trees et variantes)
- [Tables de hachage
- Index bitmap]

Organisation logique : bloc/extent/segment/tablespace

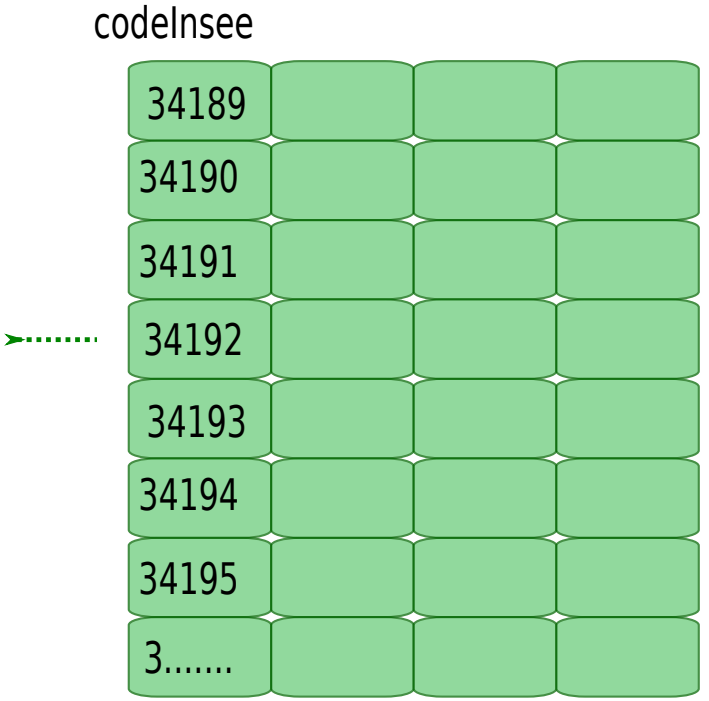


Comment ne parcourir qu'une fraction de blocs au regard de la requête

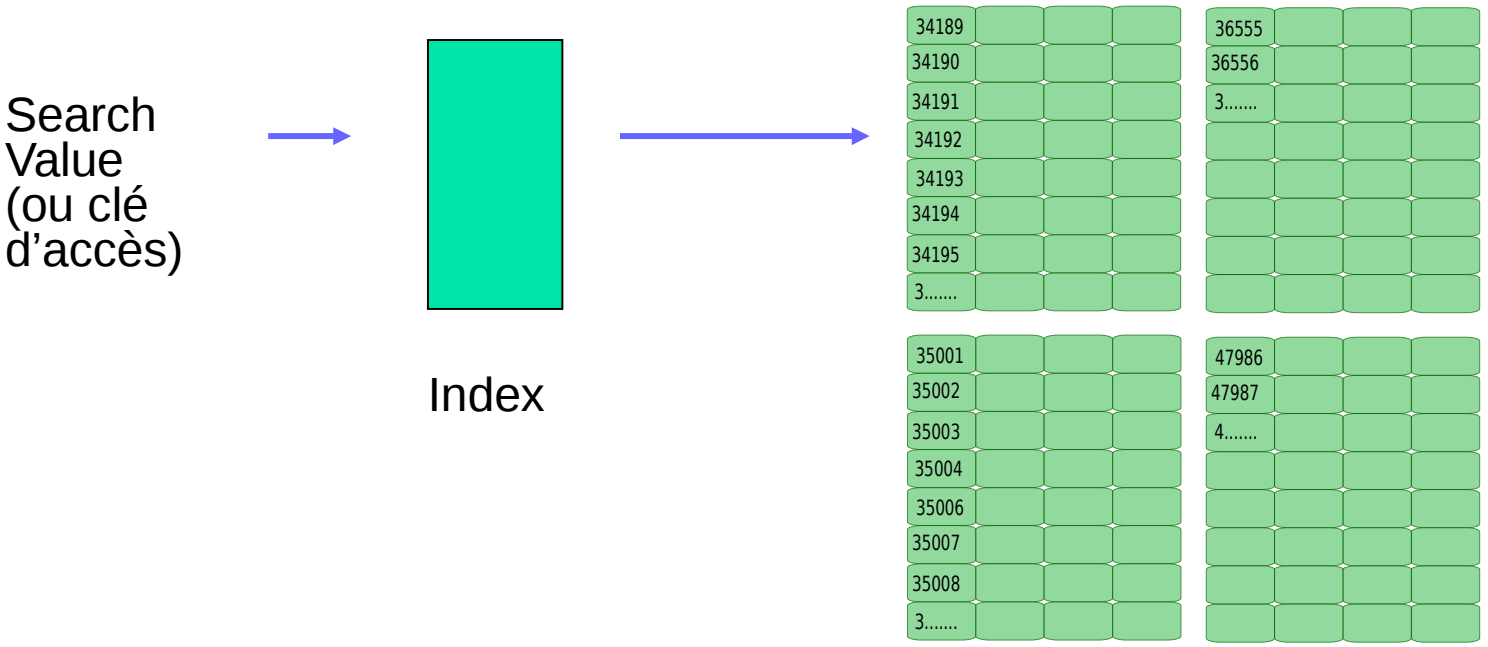
- Exemple – recherche linéaire

```
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***



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



Index

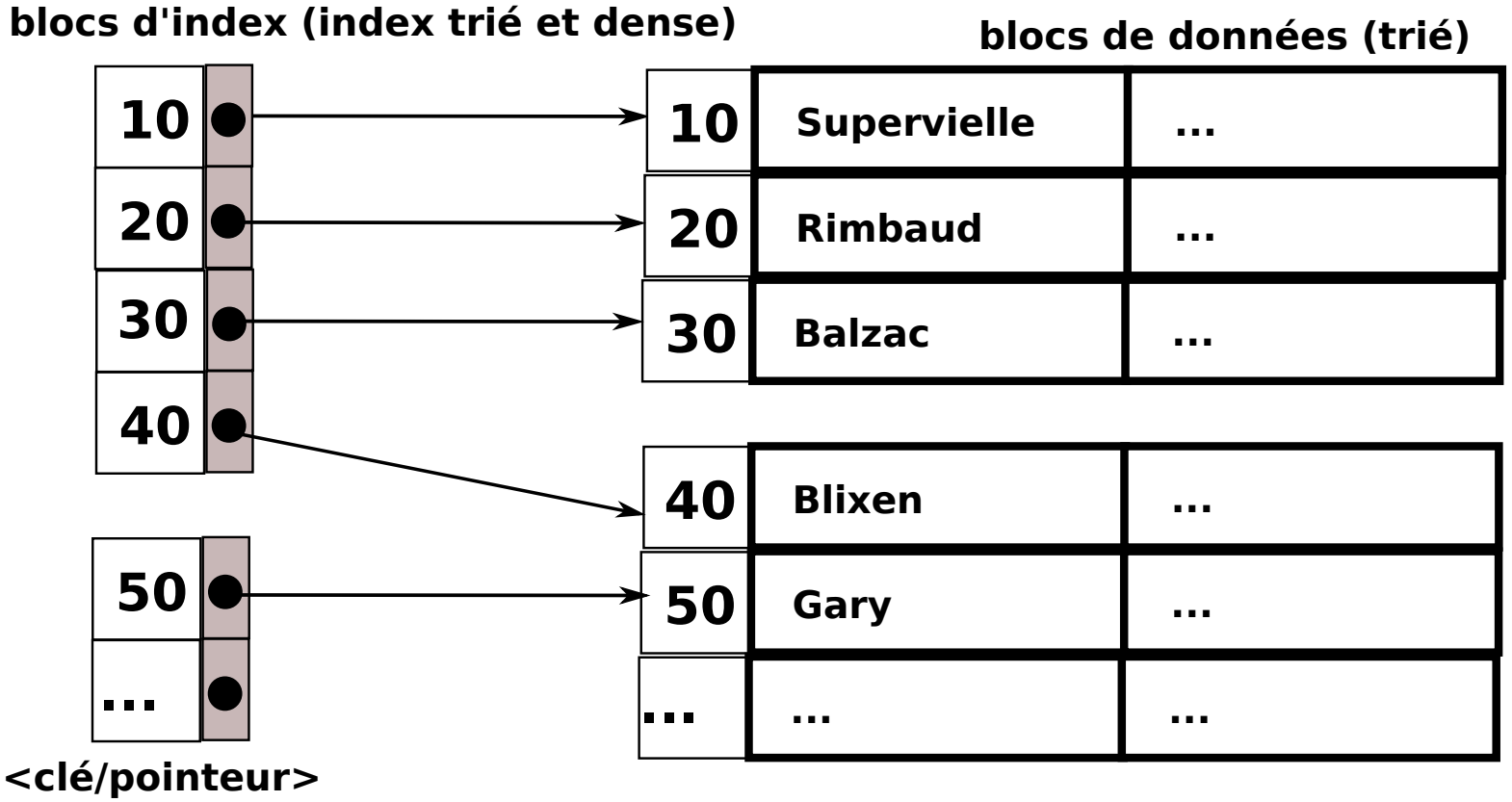
- Définition : structure de données avec en entrée une propriété (search key), et qui permet de retrouver rapidement les enregistrements possédant cette propriété

- 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



Dense : toutes les valeurs de clé représentées

Exemple 1 avec index dense

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)

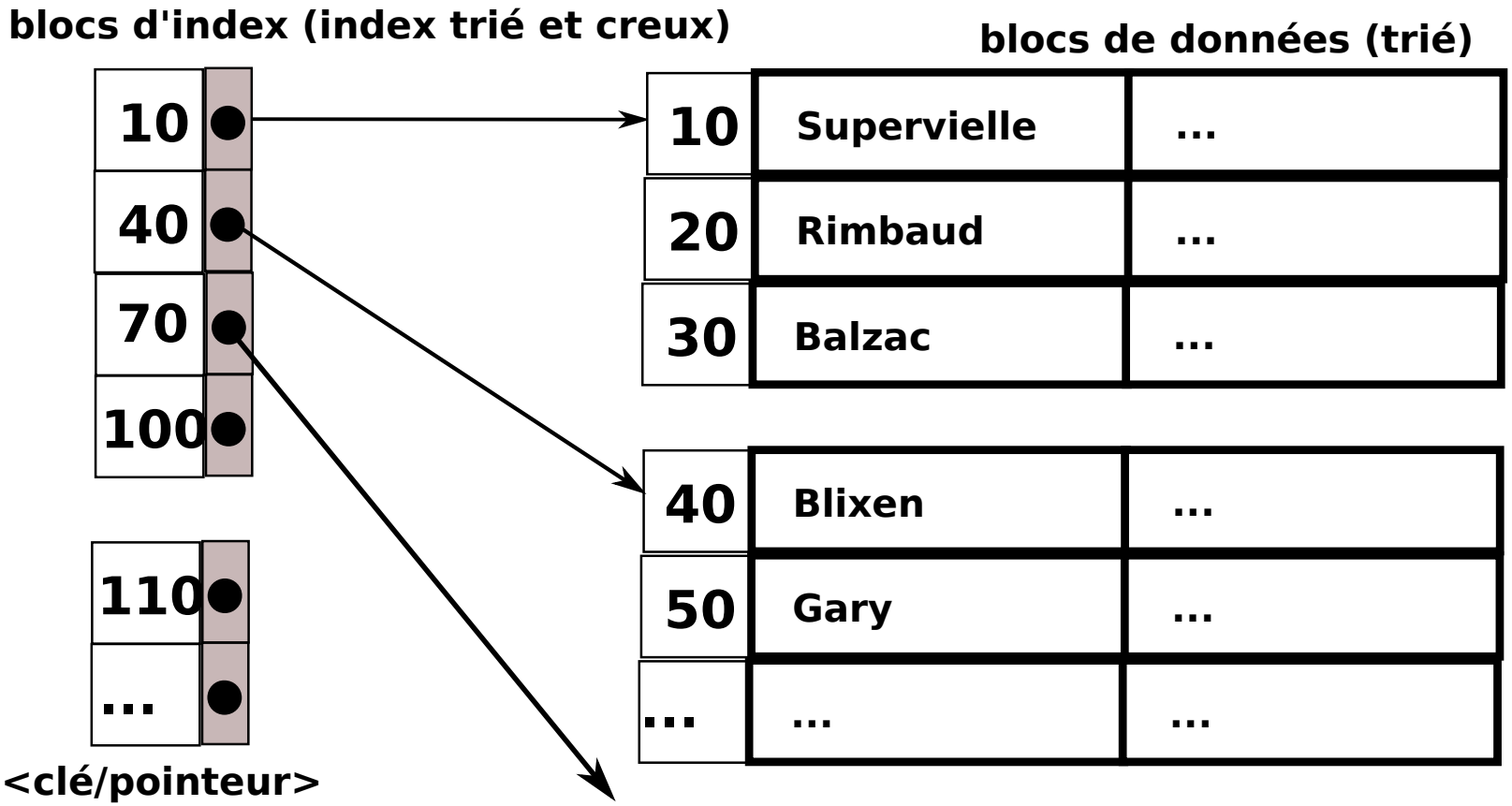
Espace mémoire pour l'index : taille de la clé 30 octets et taille du pointeur 10 octets : 40 Mo (1 000 000 * 40) et 10 000 blocs

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



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)

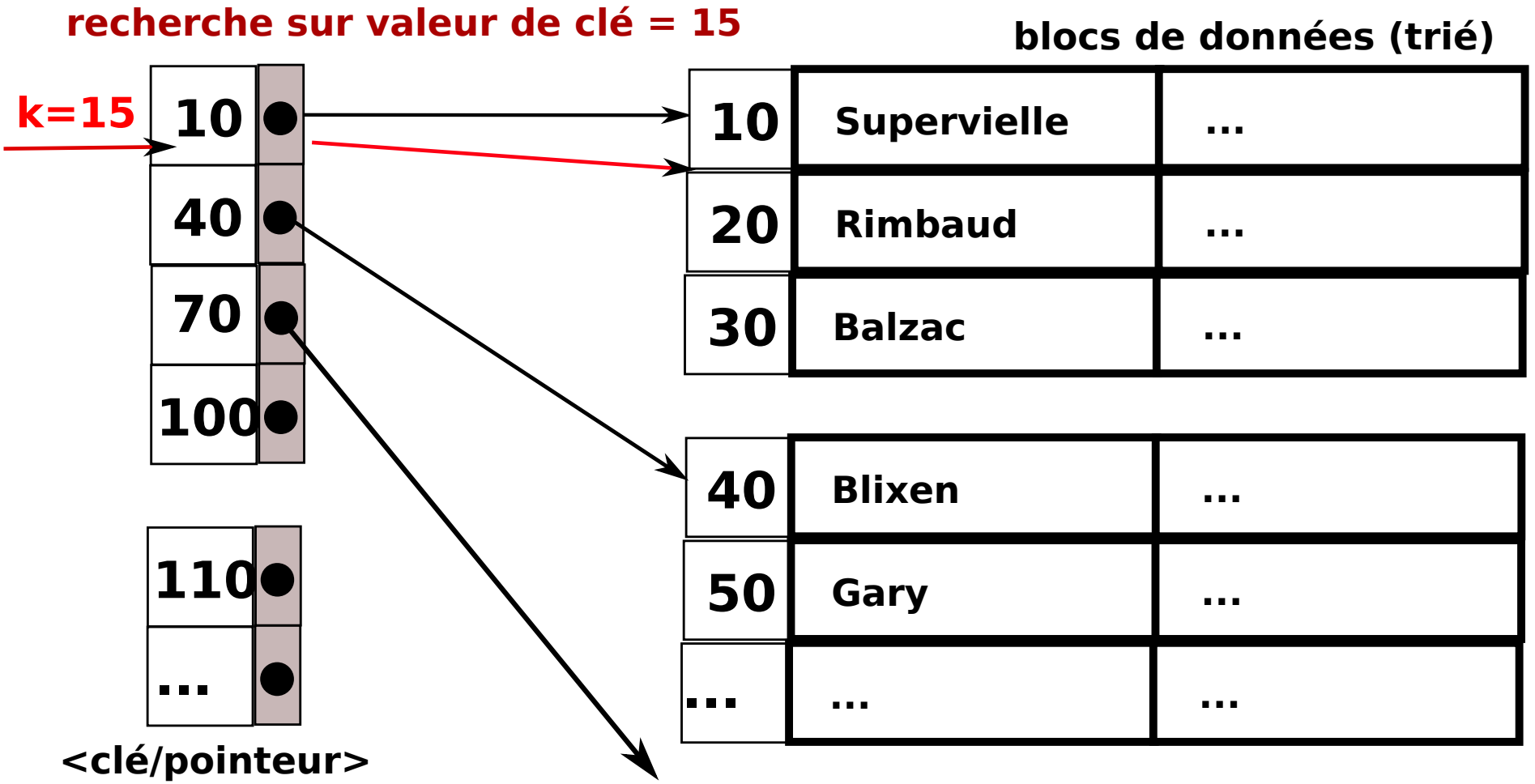
Espace mémoire pour la table : 400 Mo (100 000 * 4 Ko)

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

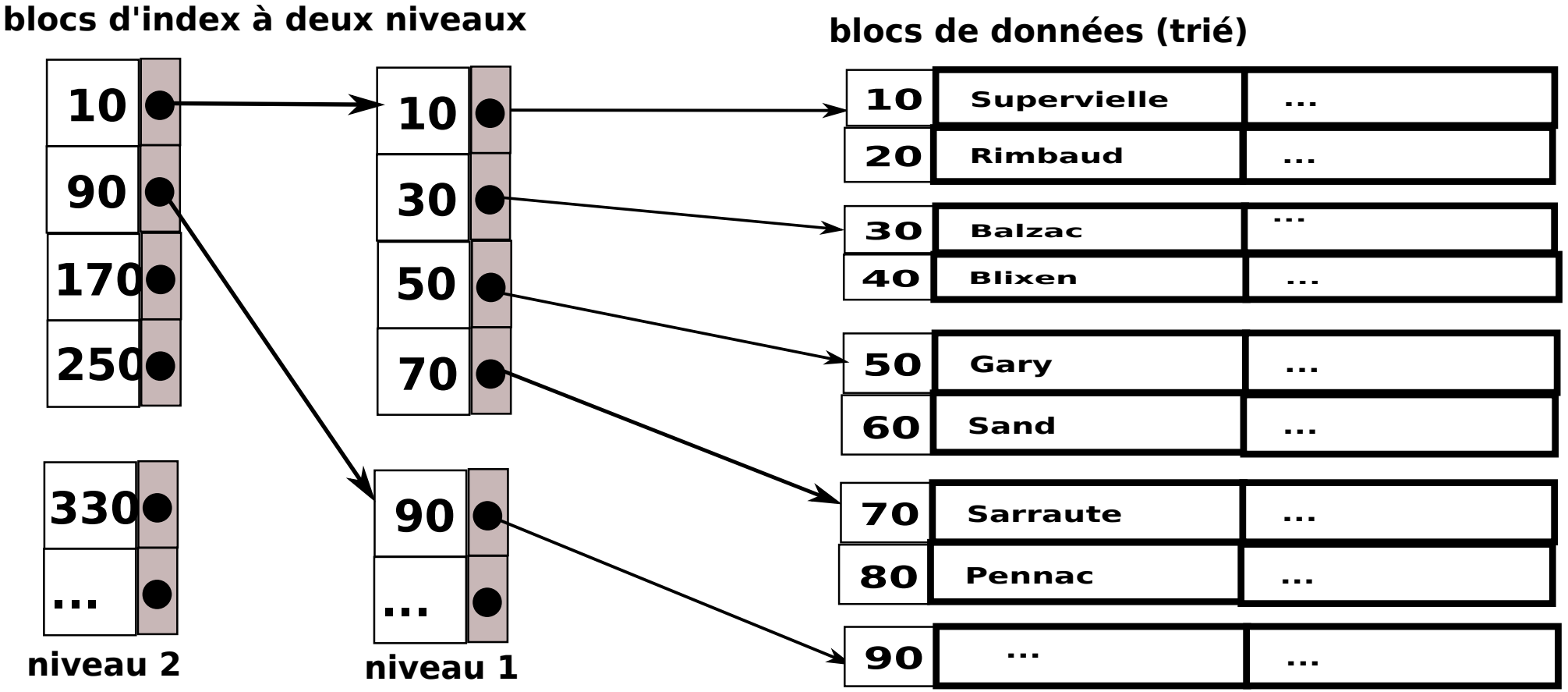
- Recherche sur la valeur d'une clé : $\log_2(1\,000) = \ln(1\,000)/\ln(2) = 9.96..$ et donc 10 blocs à parcourir + une opération d'entrée / sortie pour aller chercher le bloc de l'enregistrement recherché

Séquentiel indexé : recherche



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



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 sous forme d'arbre qui maintient dynamiquement un ensemble d'éléments de manière à ce que l'arbre soit équilibré :

L'équilibre est important pour toutes les opérations usuelles sur une table : recherche, insertion, suppression

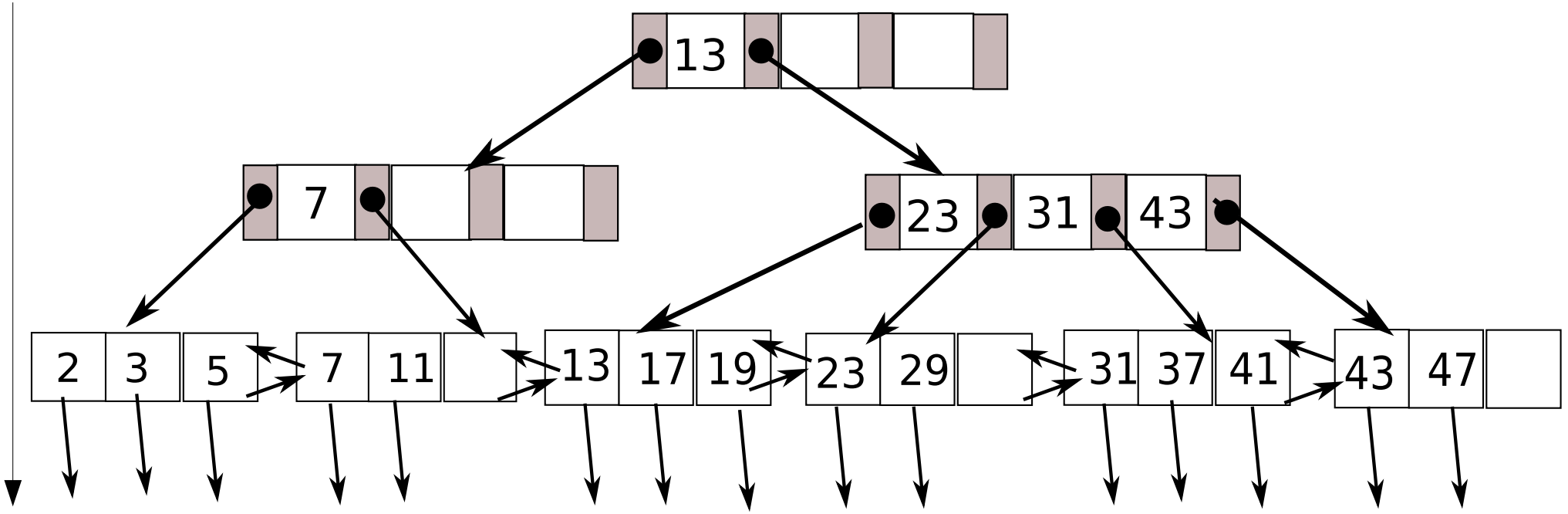
- pour chaque noeud branche de l'arbre : n clés et n+1 pointeurs,
- un noeud (sauf la racine) est de à moitié plein à plein
- pour les noeuds feuilles : elles sont toutes au même niveau et elles contiennent les clés et les pointeurs sur les données

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

- Dans un B-arbre, les noeuds intermédiaires peuvent contenir des pointeurs sur des données
- Dans un B+arbre, seules les feuilles contiennent des pointeurs sur les données et les noeuds sont doublement chaînés
- Dans un B*arbre les noeuds sont au moins à 2/3 plein

Arbre d'ordre 4 (nbre clés+1) et de hauteur 3

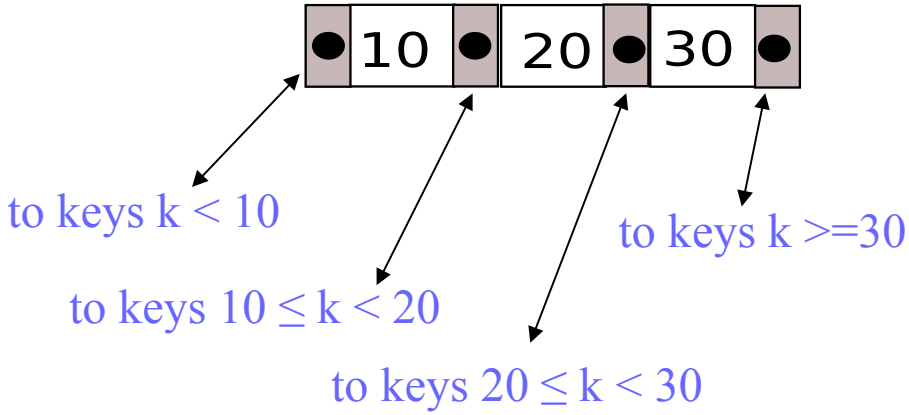
Hauteur constante



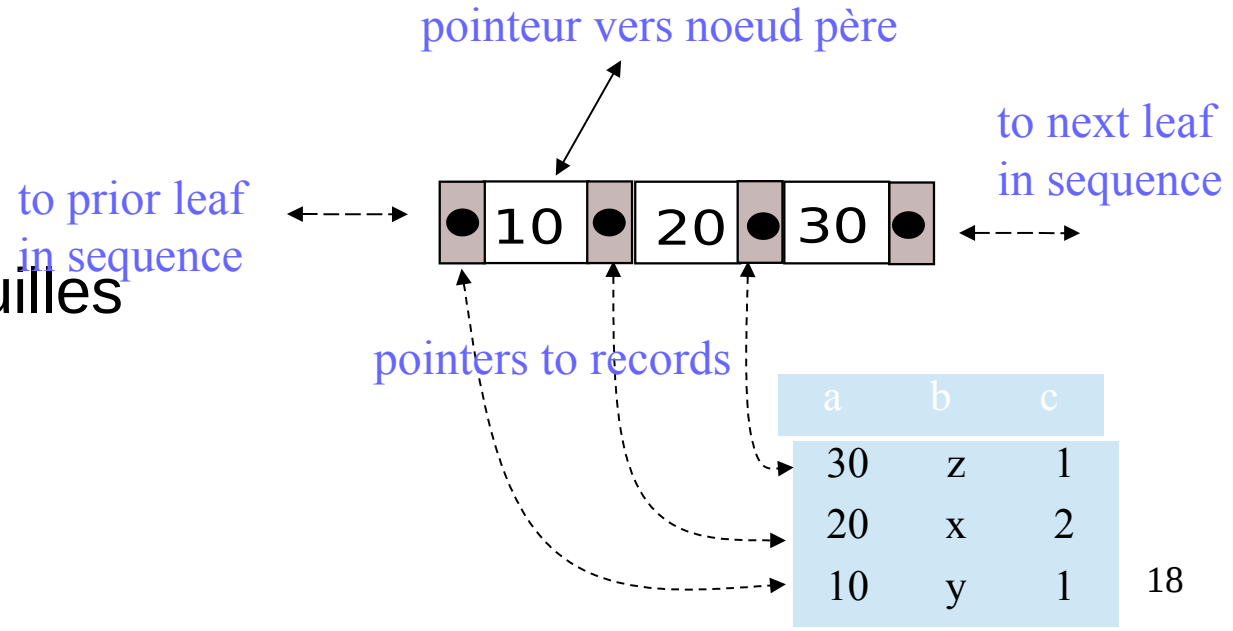
Des détails sur B+Tree

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 >=

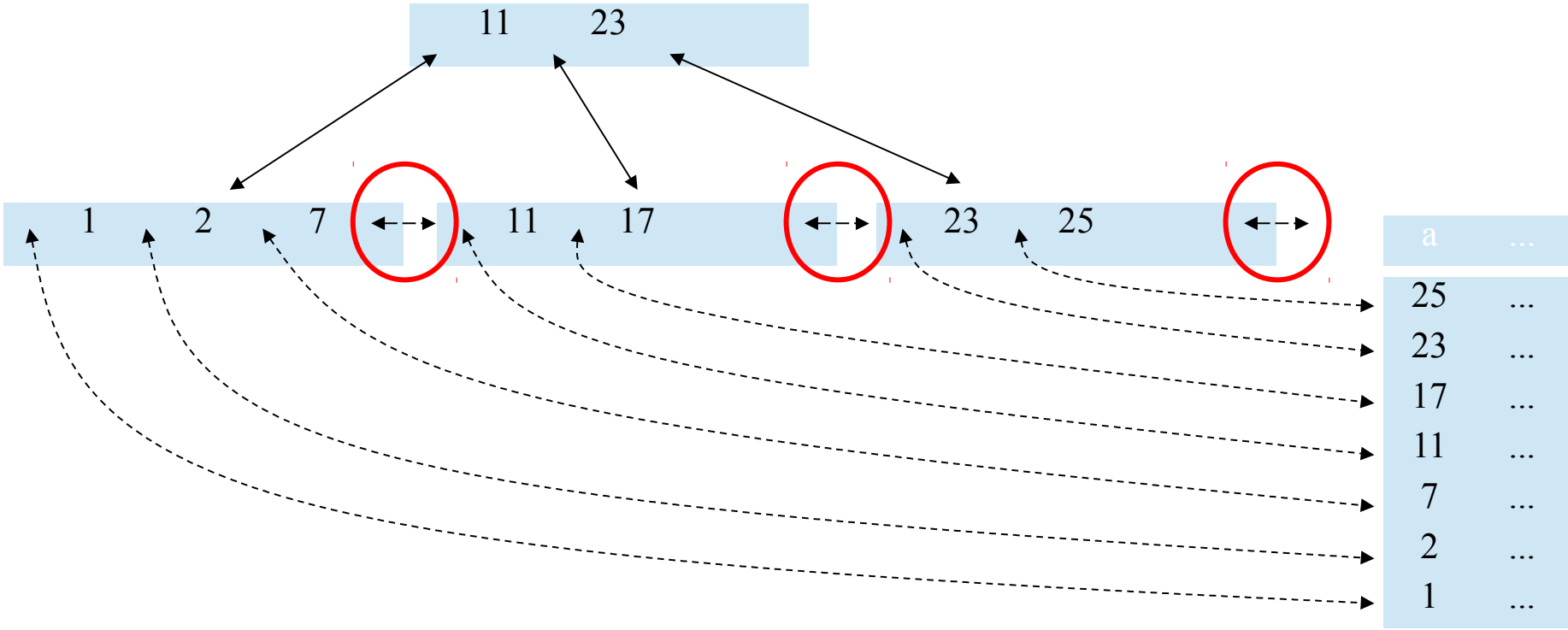


Noeuds feuilles
(n pointeurs de données,
1 pointeur gauche,
1 pointeur droit vers les feuilles voisines)



Exemple d'arbre de hauteur 2

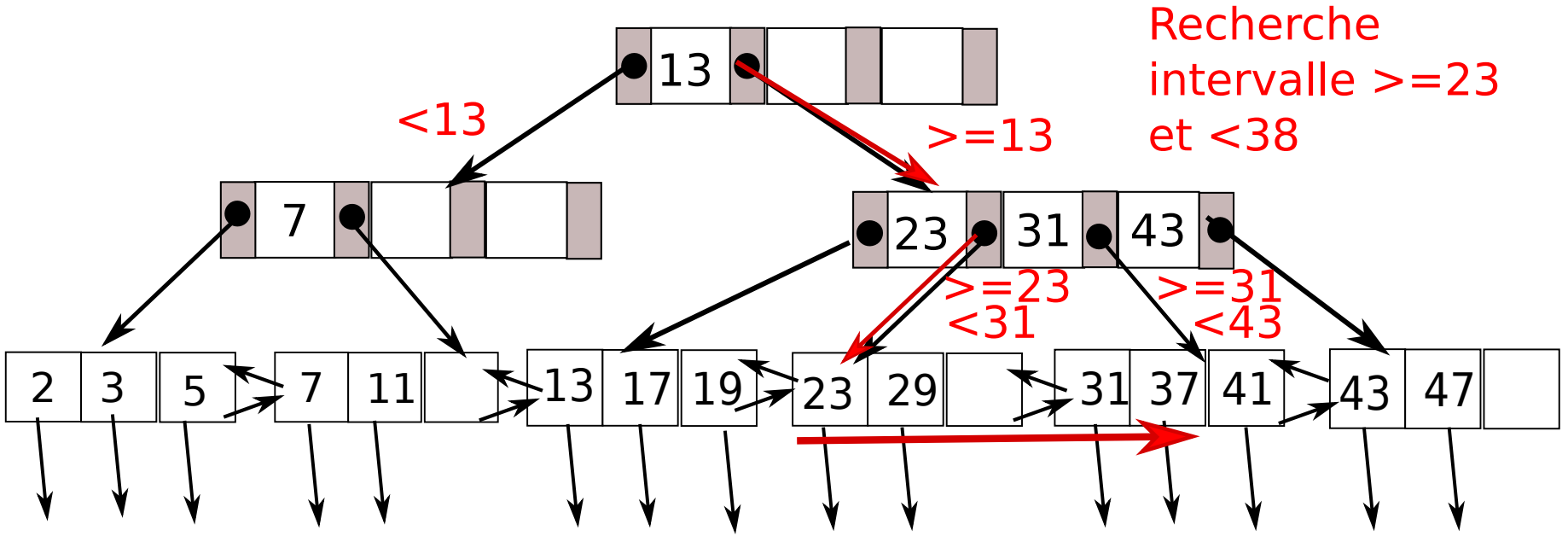
Remarque : les feuilles chaînées vont permettre la lecture séquentielle et donc la la recherche sur intervalle



Opérations sur l'index

- Recherche : parcours top-down et comparaison
  - Branches : les pointeurs sont exploités de branche en branche :
    - Si clé = K → choix du pointeur gauche si K < K' → choix du pointeur droit si K >= K'
  - Feuilles :
    - Index dense :
      - Si la nième clé = K alors le nième 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) <= 4096 => 12(n) = 4088 -> n (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 : on a $(255)^2$ noeuds feuilles et

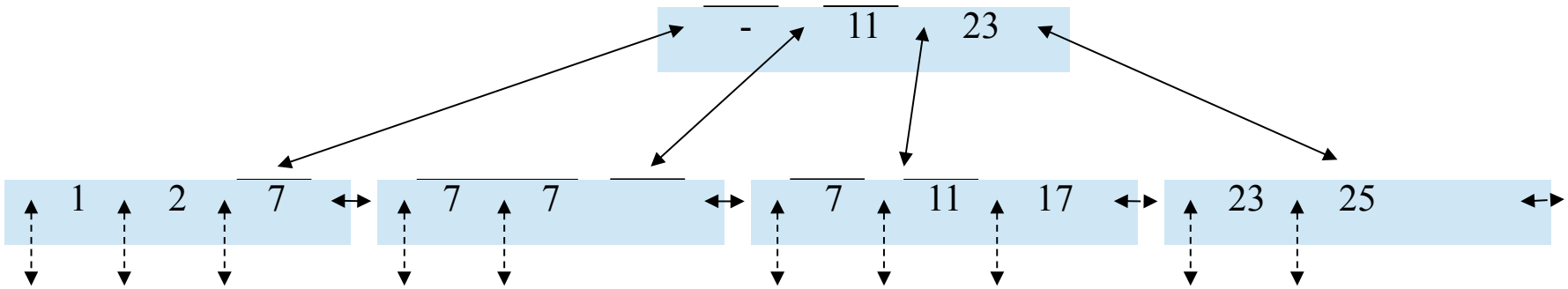
$(255)^3$ enregistrements soit plus de 16 millions pour un arbre d'une hauteur de 3

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 et n nombre d'enregistrements

Index non unique : duplication de valeurs



Remarque 1:
noeud branche pointe sur la première occurrence de la clé dupliquée et ensuite les autres occurences sont lues séquentiellement

Remarque 2:
Dans certains cas, la valeur du noeud branche est à 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 :

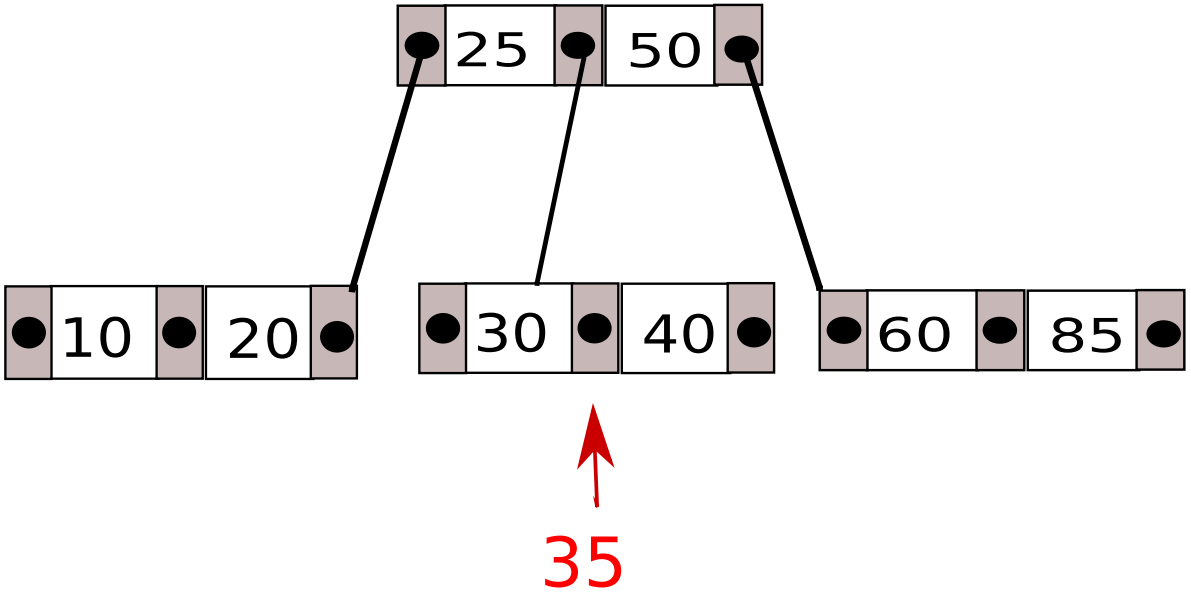
(a) L’élément médian est choisi (nouveau père du sous arbre) parmi tous les éléments présents y compris le nouveau: élément médian = (k+1)/2 ème.

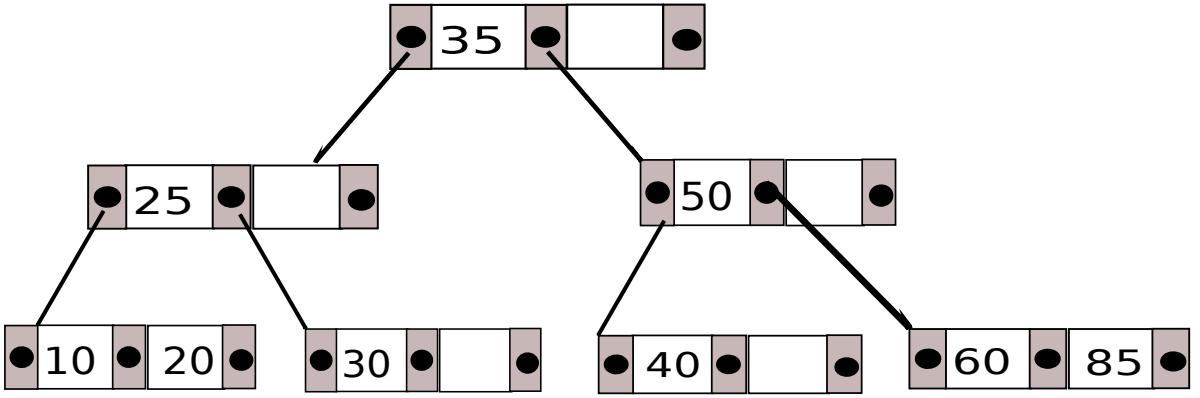
(b) Les valeurs < au médian -> fils gauche, et les valeurs > -> fils droit.

(c) L’élément médian (père du sous-arbre) est ajouté au noeud parent .

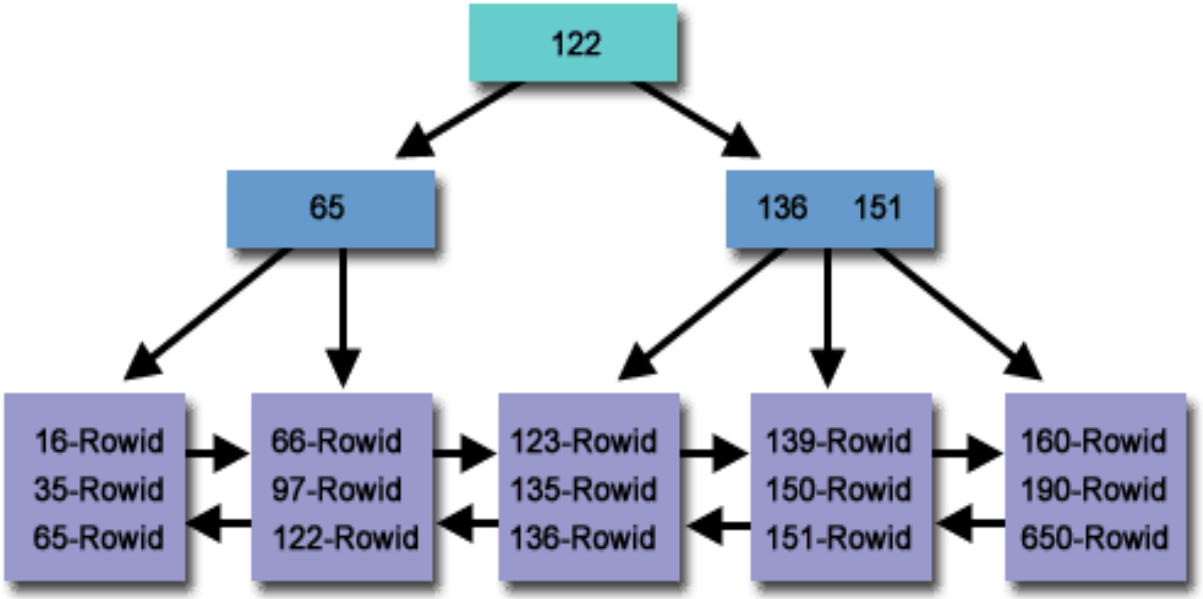
Un nouvel éclatement peut alors en résulter (continuer ainsi jusqu'à la racine)

Exemple d'insertion





B+Tree Oracle



Remarques:

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

Définition 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;

– inutilisable

alter index commune_pk unusable;

-- le reconstruire pour le rendre à nouveau valide

 alter index commune_pk rebuild ;
```