Bases de données

Polytech Paris-Sud

Apprentis 4ème année

Cours 3 : Indexation

kn@lri.fr
http://www.lri.fr/~kn

Plan

Le fichier comme abstraction du support physique

Le système et la couche physique exposent plusieurs notions de bas niveau:

- Secteurs, pistes, plateau, ...
- Pages/blocs
- *Buffer* de lecture/écriture

On souhaite proposer des structures de données et des algorithmes génériques permettant d'implanter un moteur d'exécution de requêtes (SQL).

On abstrait les concepts bas niveau par la notion de fichier. Un fichier un ensemble d'enregistrements. Un enrigstrement contient des données structurées en champs. En pratique, les fichiers sont paginés

Opérations sur les fichiers

La couche physique propose plusieurs opérations sur les fichiers

- Création/suppression de fichier
- Insertion/supression d'enregistrement
- Accès au $i^{ème}$ enregistrement du fichier
- Accès au $i^{ème}$ champ d'un enregistrement

(cf. cours 2 pour les différentes manières d'implanter ces opérations).
Une table est généralement représenté par un fichier (au sens collection d'enregistrements)

Fichier simple

La structure la plus simple pour un fichier est la structure en tas (à ne pas confondre avec la structure de donnée du même nom). La couche physique du SGBD connait la liste des pages d'un fichier ainsi que l'espace libre sur chaque page, les enregistrements sont ajoutés ou supprimé dans les pages que le SGBD considère comme meilleures (du point de vue des E/S) sans considération sur les données.

Rechercher un enregistrement particulier, en fonction d'un critère sur son contenu (ex: `id='5'`) nécessite un parcours séquentiel du fichier (scan)

Plan

Index

Un index est un fichier auxiliaire, structuré qui va rendre plus efficace l'accès à certaines données, en fonction d'une clé d'index.

Un enregistrement d'un index s'appelle une entrée d'index. L'entrée associée à la clé *k* est notée *k** et contient suffisament d'information pour retrouver le ou les enregistrements (de la table indexée) associés à *k*. On distingue trois types d'indexes:

- Index de type I : *k** représente directement un enregistrement de la table originale (la table indexée).
- Index de type II : une entrée est un couple <k,rid> où k est la clé et rid est un pointeur vers l'enregistrement ayant cette clé
- Index de type III : un couple <k,rid-list> où k est la clé et rid-list est une liste de pointeurs vers les enregistrements ayant cette clé

Indexes groupants

Un index est dit groupant si ses entrées sont triées dans le même ordre que les enregistrement du fichier indexé. Il est dit non-groupant dans le cas contraire

Les enregistrements de la table initiale ne sont jamais dupliqués. On ne garde donc jamais un indexe de type I en plus de la table initiale (on supprime cette dernière)

Indexes denses

Un index **dense** s'il contient une clé par enregistrement et non dense sinon

Si une table possède un index **non-dense**, alors le fichier de cette table est **toujours** trié selon la clé d'index (sinon l'index ne sert à rien).

Autres propriétés

Un index pour lequel la clé d'index contient la clé primaire de la relation est appelé un index primaire. Les autres indexes sont appelés indexes secondaires

Deux entrées d'index possédant la même clé sont des doublons. Un index primaire ne contient pas de doublon.

Plan

Différentes structures pour différents usages

Outre le fichier tas un index peut avoir les représentations internes suivantes:

- Hash-index : l'index est organisé comme une table de hashage
- Tree-index : l'index est organisé comme un arbre de recherche
- Fichier trié : le fichier est trié selon une clé particulière (i.e. l'ordre des enregistrements sur le disque coïncide avec l'ordre de la clé d'index)

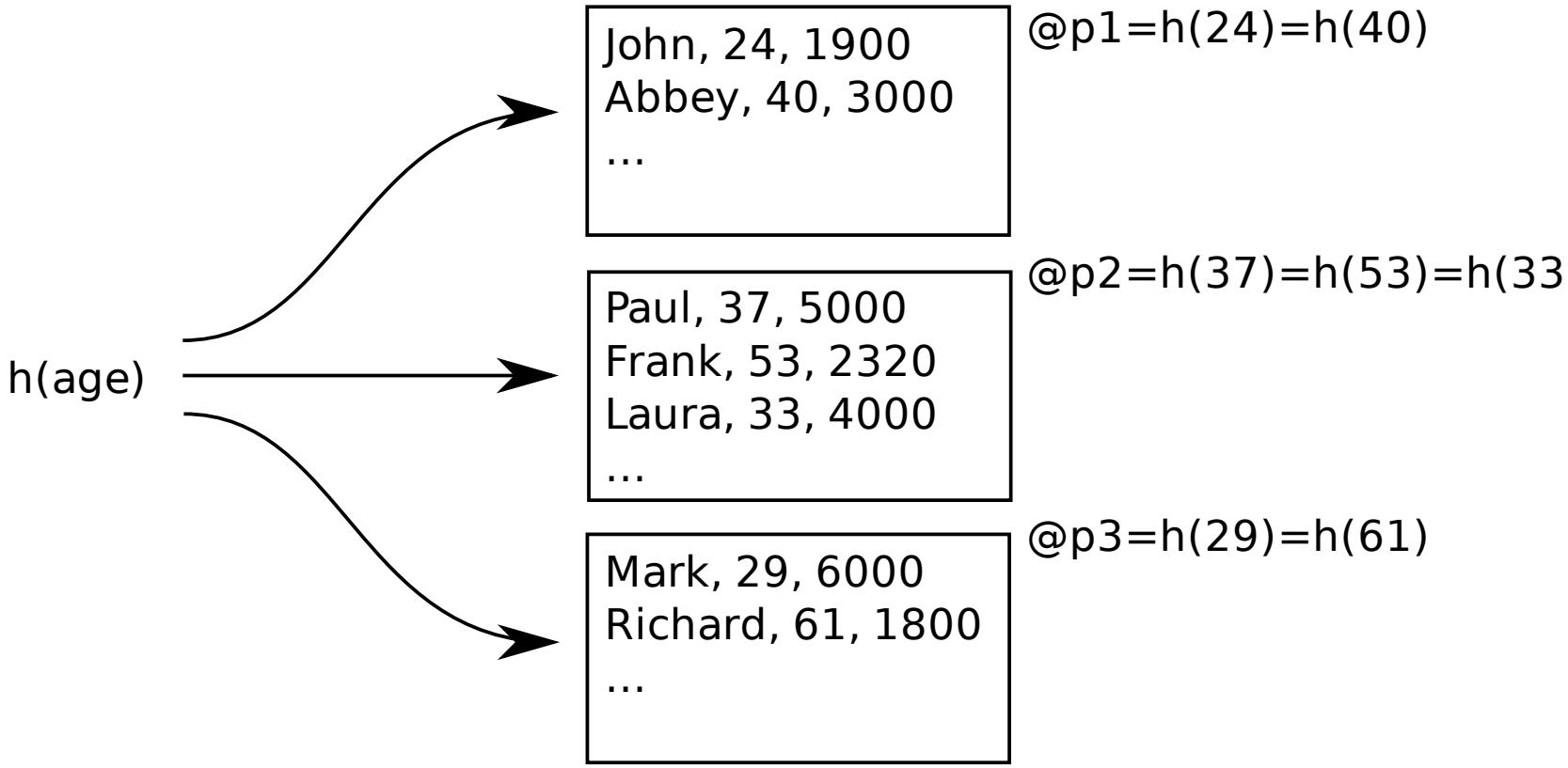
On illustre sur une relation ayant le schéma `Emp(Nom, Age, Salaire)`

Hash index (informel)

Un hash index repose sur une fonction de hash h **telle que:**

$$h(k) = @page$$

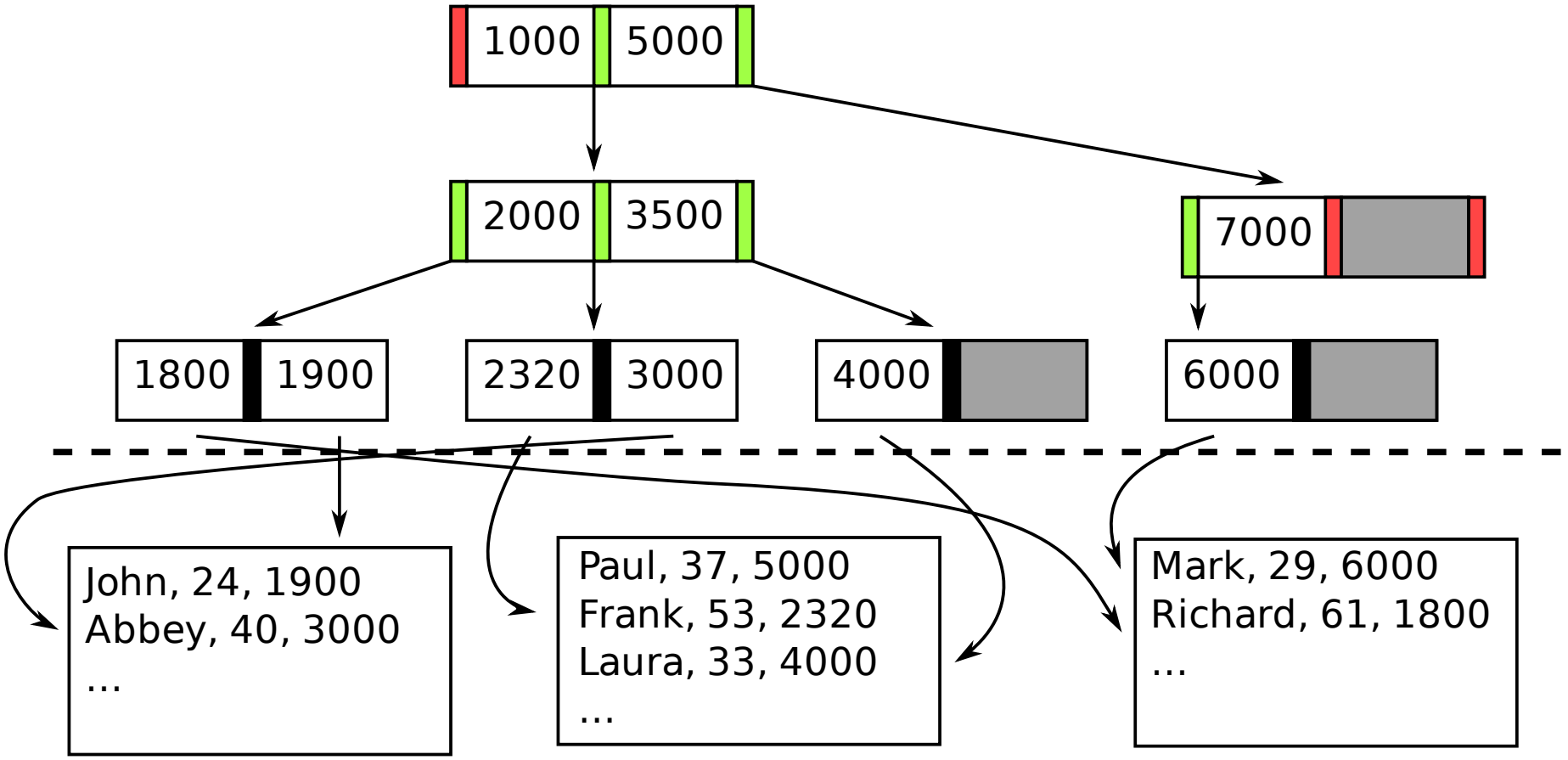
où k **est la clé d'index et** @page **est l'adresse (sur le disque) de la page où se trouve l'enregistrement associé à** k



Remarque: index de type I

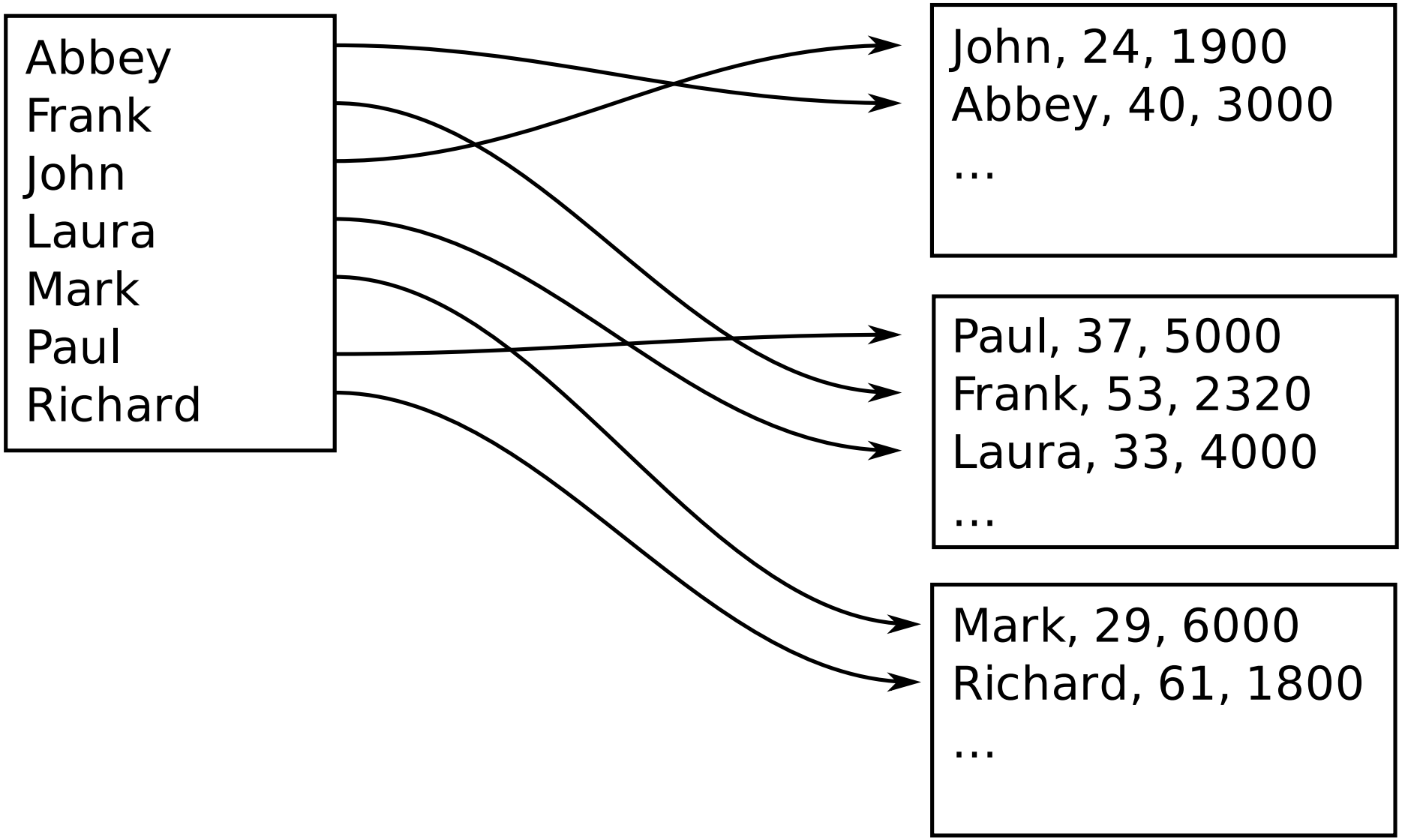
Arbre B+ (informel)

Arbre «n-aire» de recherche (avec n grand, généralement 100)



Remarque: index de type II, non groupant

Fichier trié



Remarque: index de type II, non groupant

Cas d'utilisation

- **Hash-index : condition d'égalité (`age=28 OR age=46`)**
- **Arbre B+ : condition d'intervale (`sal > 1000 AND sal < 3000`), condition de préfixe (`nom LIKE 'Jo%'`)**
- **Fichier trié : idem (plus compact mais insertion/suppression plus difficile)**

Coût des opérations 1/2

On pose: `B` **: nombre de pages de données**
`R` **: nombre d'enregistrements par page**
`D` **: temps moyen de transfert d'une page**
On suppose des indexes de type I

Fichier tas

- **Parcours** `B * D`
- **Recherche**
  - **0 réponse** `B * D`
  - **1 réponse** `0.5 * B * D` **(en moyenne)**
  - **n réponses** `B * D`

Coût des opérations 2/2

Fichier trié

- **Recherche sur autre que clé de tri:** $B * D$
- **Recherche sur clé de tri :** $D*\log_2 B + (\#réponses/R)$

Hash index

- **Recherche sur autre que clé de tri, ou recherche autre qu'égalité:** $B * D$
- **Recherche (egalité) sur clé de tri :** $2 + (\#réponses/R)$

Arbre B+

- **Recherche sur autre que clé de tri :** $D * \log_F B + B*D$
- **Recherche sur clé de tri :** $D*\log_F B + (\#réponses/R)$

Clé composée

Une clé d'index peut être composée de plusieurs champs de la relation originale. Si une clé est composée de $(c_1, \ldots, c_n)$**, on peut l'utiliser pour toute requête dont la valeur dépend de tous les** `i <= k`**, pour** `k <=n`**.**

Supposons une clé composée sur `(age,salaire)`**. On peut l'utiliser pour** `age > 30` **ou pour** `age >30 AND salaire <4000` **mais pas pour** `salaire <4000` **ou** `age >30 OR salaire <4000`

Plan

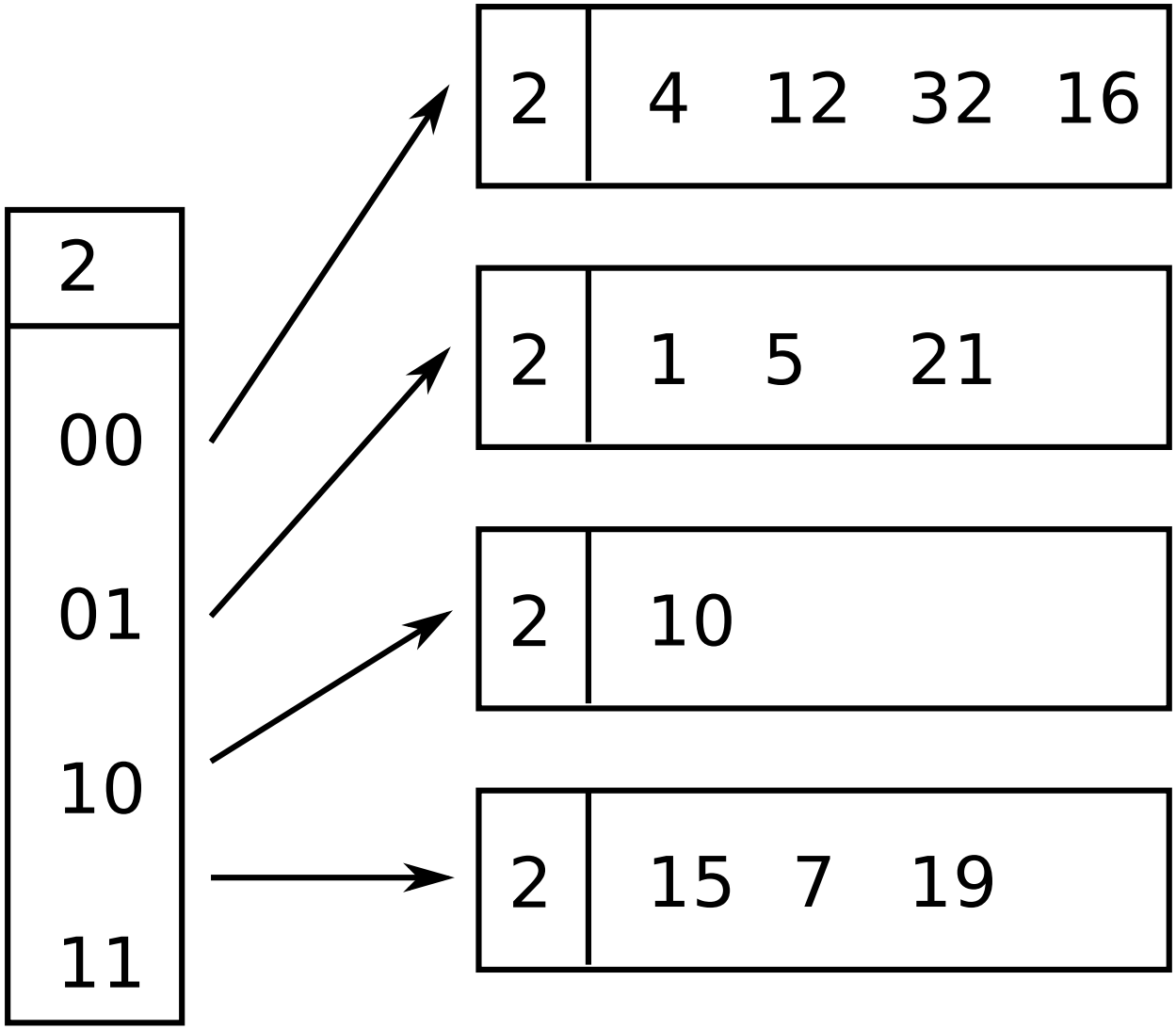
Tables de hash classique (rappel)

Table de hash classique = tableau de taille `N` contenant des listes chaînées de valeurs (*bucket*). On calcule `h(k) mod N` pour voir dans quel *bucket* insérer la nouvelle valeur. Si un *bucket* devient trop grand, on double la taille du tableau et on redistribue tous les contenus des *bucket*.

La redistribution est trop couteuse ici : on doit scanner/écrire tout l'index. On souhaite ne toucher que le tableau et le *bucket* trop grand, et pas les autres.

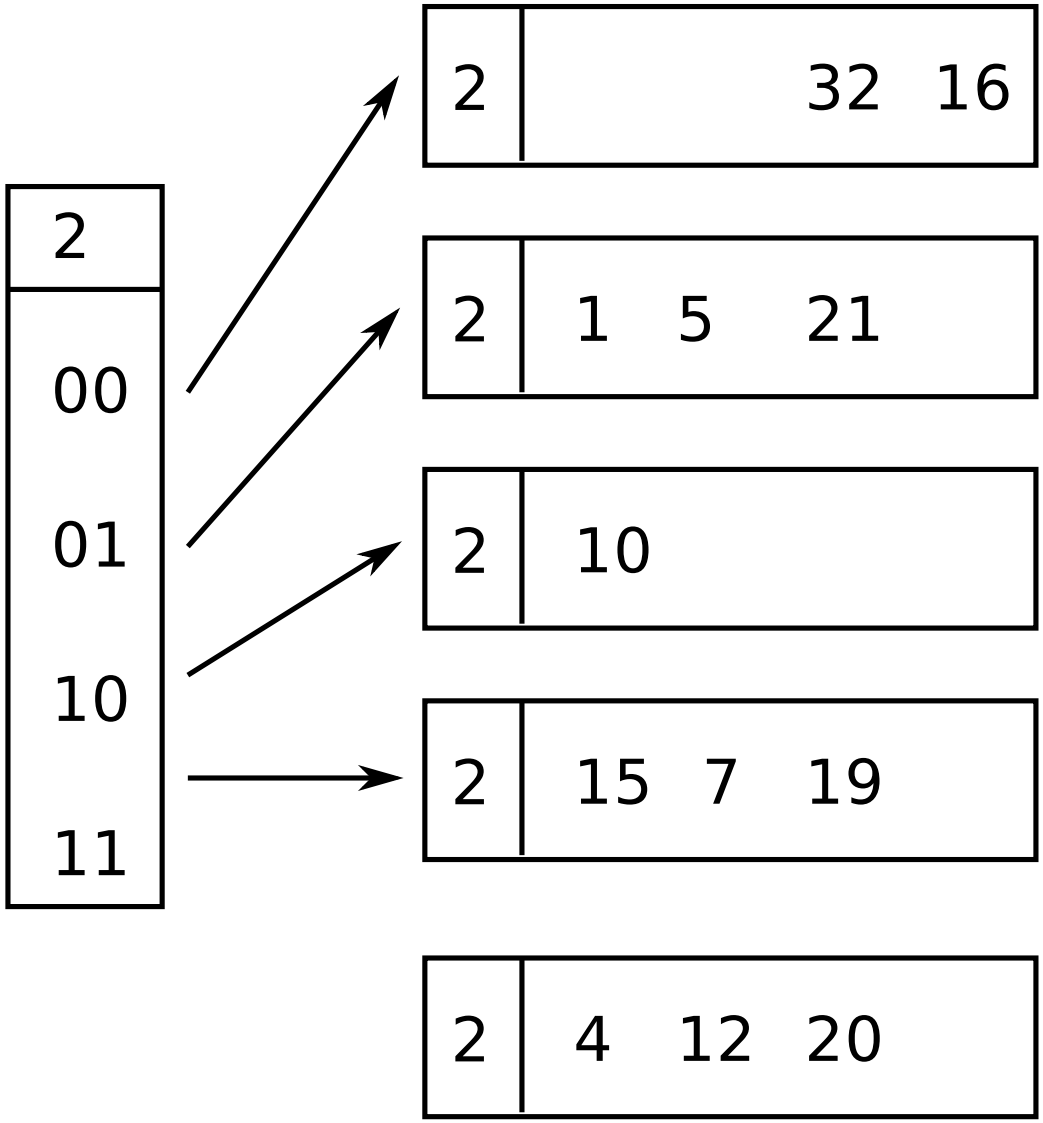
Hash-index avec répertoire

On n'utilise pas un simple tableau mais un tableau contenant un masque binaire. On garde aussi un compteur de profondeur globale et un compteur de profondeur local, pour chaque *bucket*



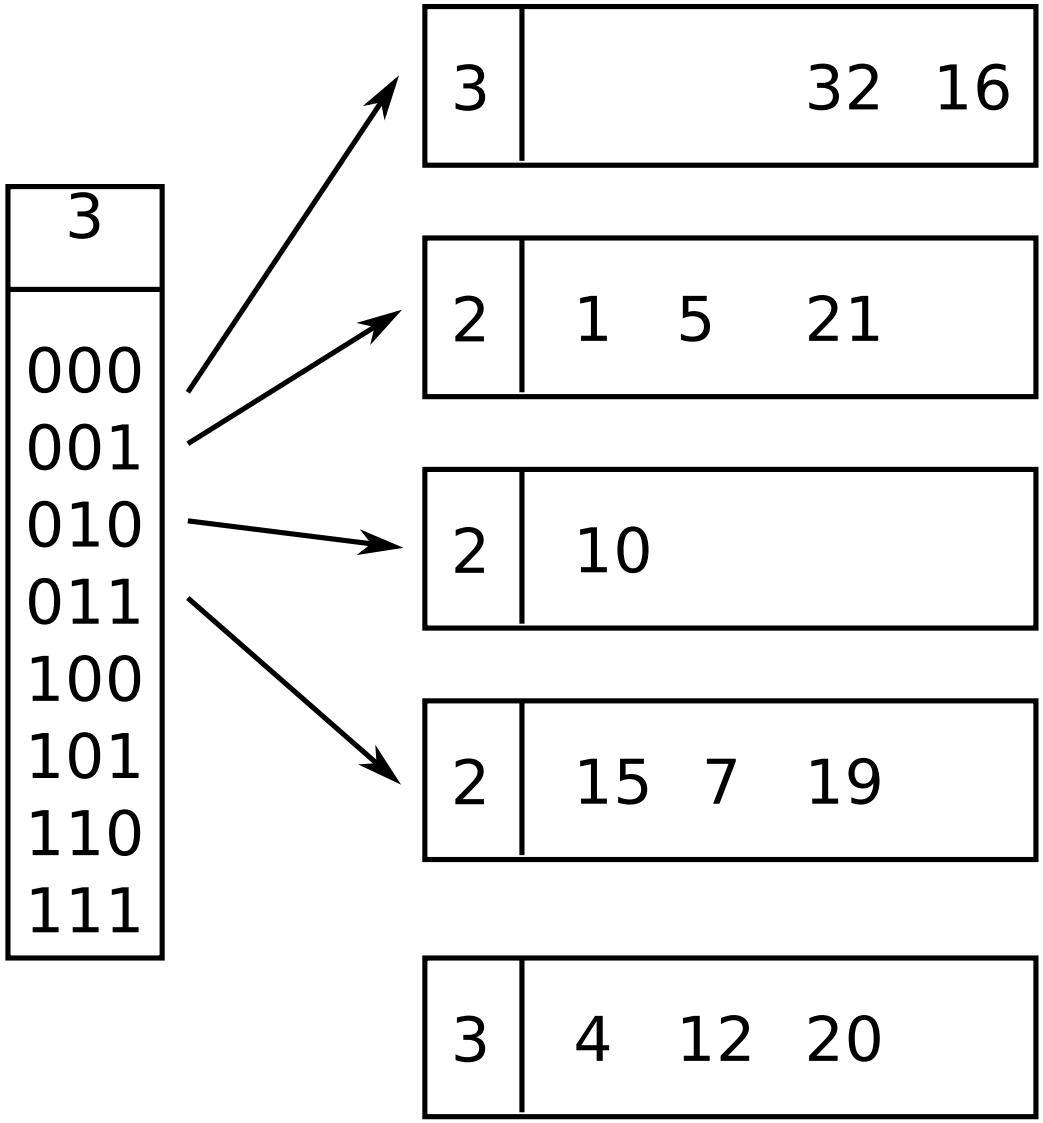
Insertion de 20

On sépare le *bucket* plein en deux (100 vs 000)



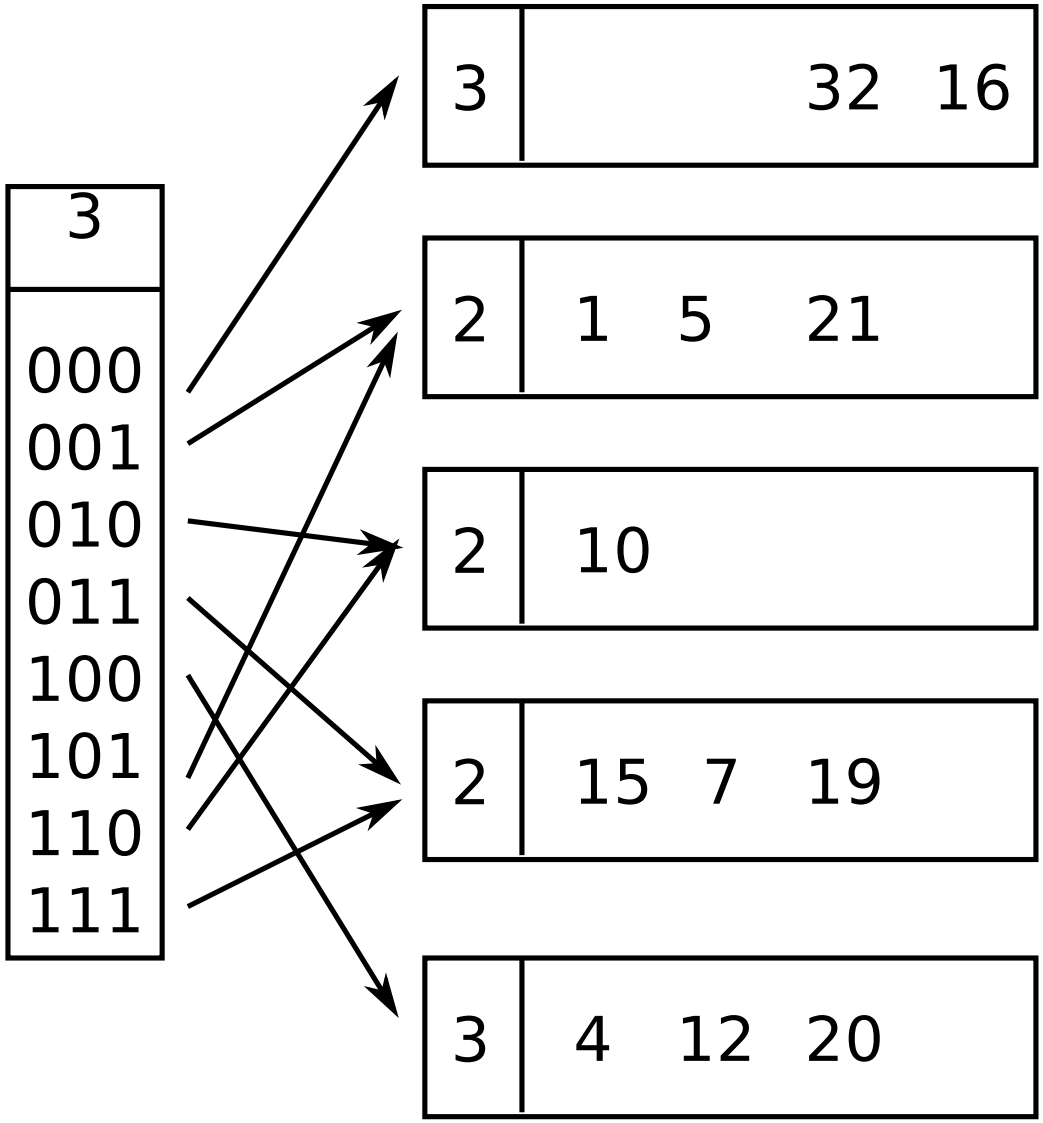
Insertion de 20

On double la taille du répertoire et on augmente les compteurs globaux et locaux



Insertion de 20

Le pointeur de 100 va vers le nouveau *bucket* les autres vont vers les anciens



Plan

Arbre B+

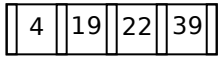
Un Arbre B+ est un arbre «n-aire» dont les nœuds peuvent contenir entre M et 2M valeurs (sauf la racine, qui a entre 1 et 2M valeurs).

Les noeuds internes contiennent des valeurs et des pointeurs vers les fils.

Les feuilles contiennent les valeurs finales, un pointeur vers les données indexées et sont chaînées entre-elles

Exemple sur un noeud d'ordre 4 (i.e. entre 2 et 4 valeurs, 5 pointeurs vers les fils).

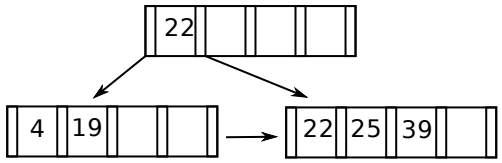
Arbre vide: 

Insertion 4,19,22,39 

(noeud plein)

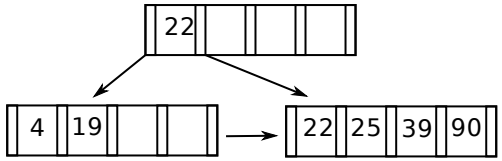
Arbre B+ partage des feuilles

Insertion 25

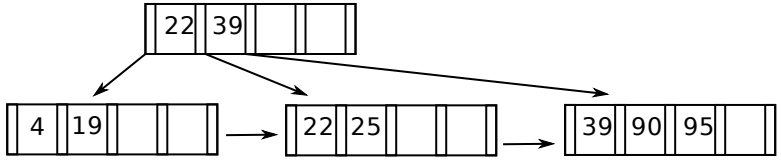


(partage du noeud, insertion de 25, report de la plus petite valeur dans le parent, chaînage des feuilles)

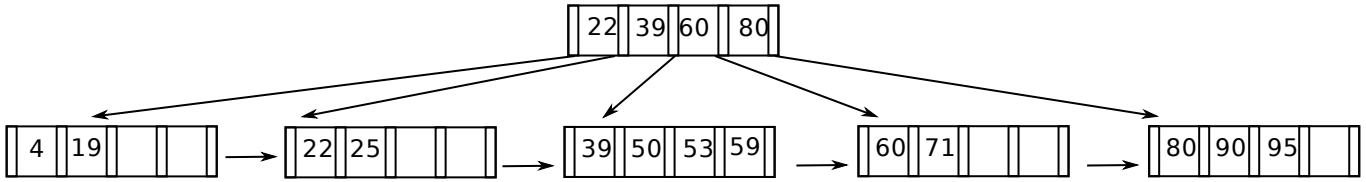
Insertion 90



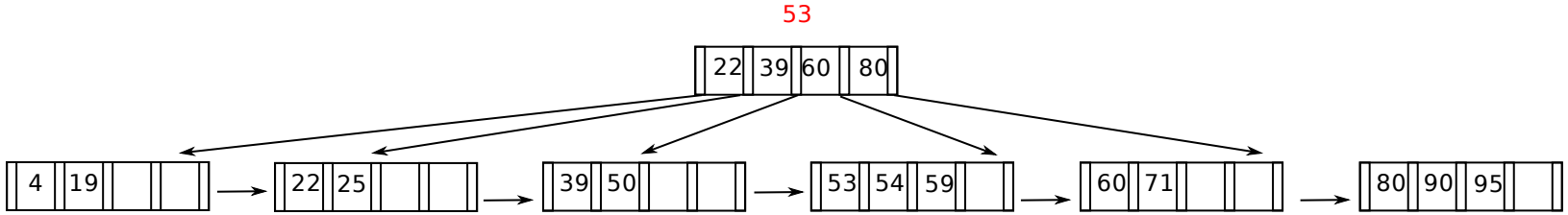
Insertion 95



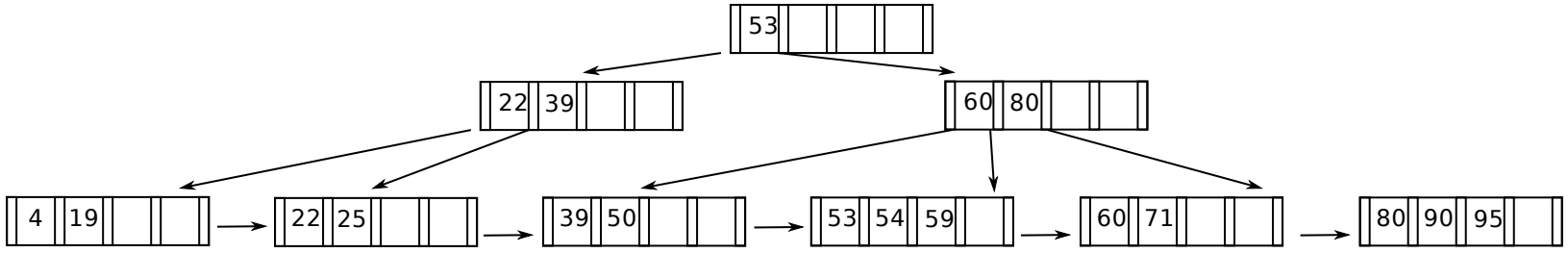
Arbre B+ partage des noeuds internes



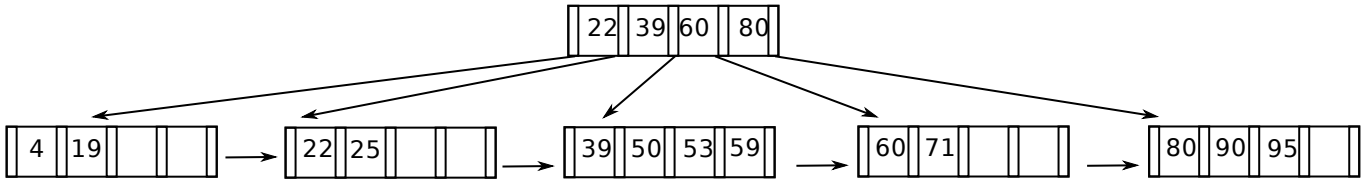
insertion de 54



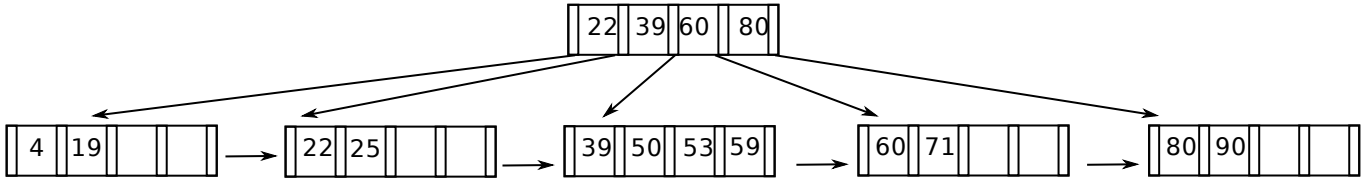
Partage en 2 et insertion de la valeur mediane dans un nouveau parent



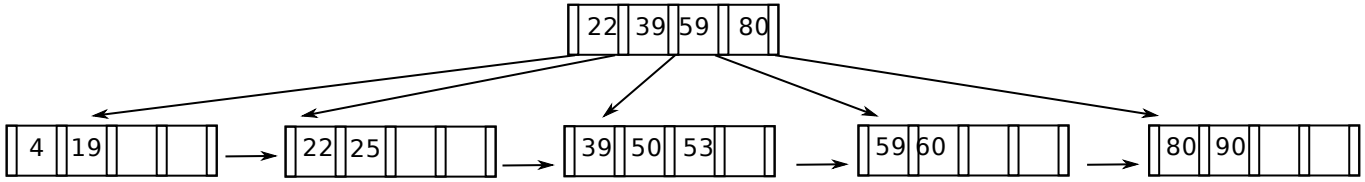
Arbre B+ suppression



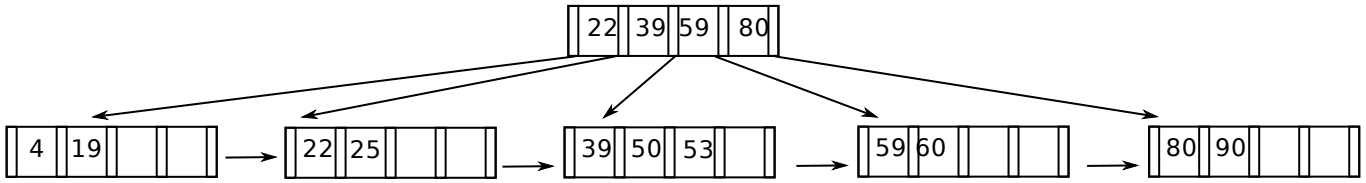
suppression 95 (simple)



suppression 71 (utilisation des voisins)



Arbre B+ suppression (suite)



suppression 19 (fusion des voisins, suppression de l'entrée dans le parent)