

Chapitre 5

Indexation

Pourquoi l'indexation?

- ❖ Si les tables sont très grandes, la recherche séquentielle est très coûteuse

```
CINEMA(NomCinema, Adresse, Gerant)
SALLE(NomCinema, NoSalle, Capacite)
```

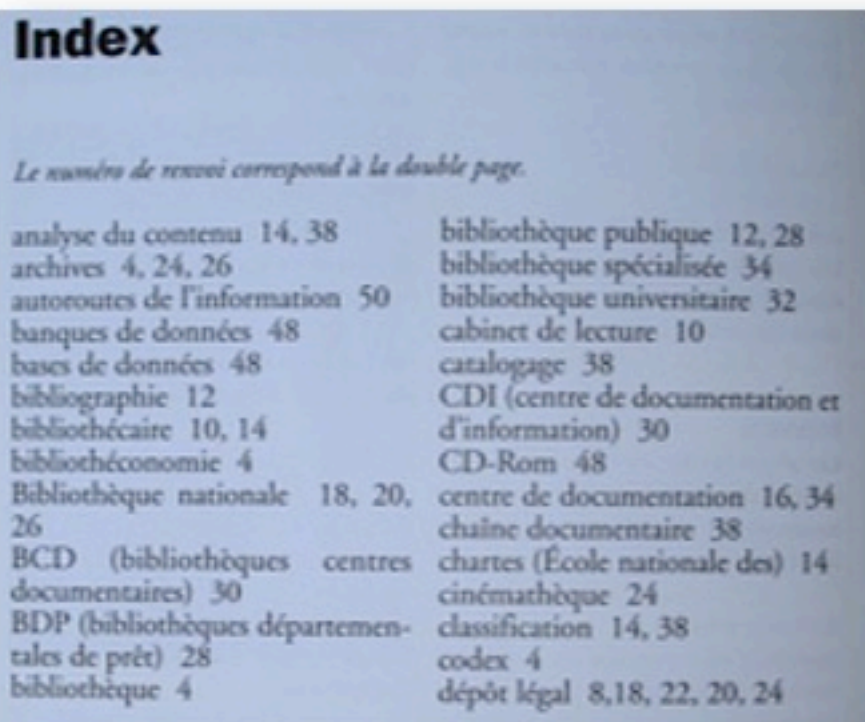
- Comment obtenir « vite » les enregistrements satisfaisant un prédicat ?
Supposant qu'uniquement 20% des enregistrements satisfont la requête :

```
SELECT Adresse
FROM CINEMA, SALLE
WHERE capacite > 150 AND CINEMA.NomCinema = SALLE.NomCinema;
```

- Comment faire « vite » les jointures ?

Pourquoi l'indexation?

- ❖ Prenons un livre à contenu technique. Il contient (au moins) un index
 - L'index présente les termes importants, classés par ordre alphabétique
 - À chaque terme sont associés les numéros de page où on trouve le terme
 - En parcourant l'index (par dichotomie !) on trouve la ou les pages qui nous intéressent.

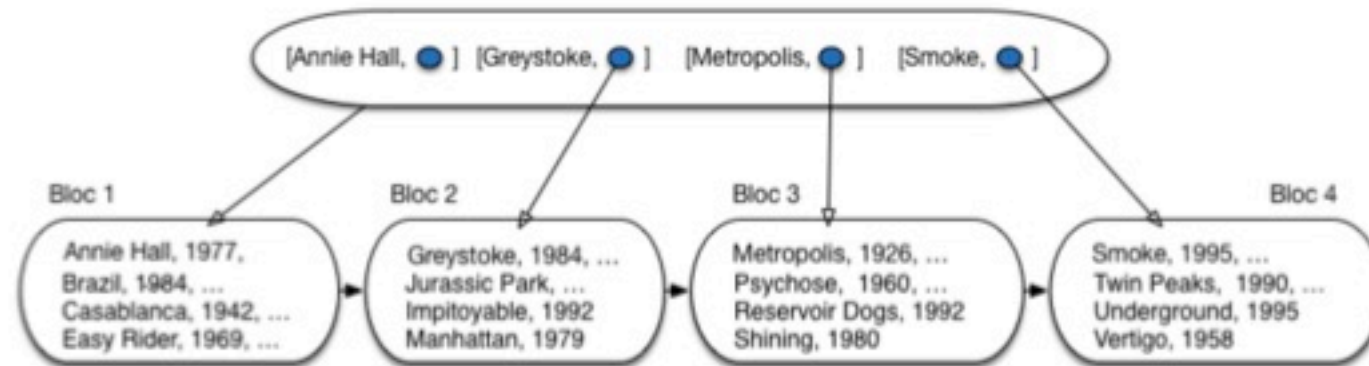


| Index | |
|---|---|
| <i>Le numéro de renvoi correspond à la double page.</i> | |
| analyse du contenu 14, 38 | bibliothèque publique 12, 28 |
| archives 4, 24, 26 | bibliothèque spécialisée 34 |
| autoroutes de l'information 50 | bibliothèque universitaire 32 |
| banques de données 48 | cabinet de lecture 10 |
| bases de données 48 | catalogage 38 |
| bibliographie 12 | CDI (centre de documentation et d'information) 30 |
| bibliothécaire 10, 14 | CD-Rom 48 |
| bibliothéconomie 4 | centre de documentation 16, 34 |
| Bibliothèque nationale 18, 20, 26 | chaîne documentaire 38 |
| BCD (bibliothèques centres documentaires) 30 | chartes (École nationale des) 14 |
| BDP (bibliothèques départementales de prêt) 28 | cinémathèque 24 |
| bibliothèque 4 | classification 14, 38 |
| | codex 4 |
| | dépôt légal 8, 18, 22, 20, 24 |

Pourquoi l'indexation?

- ❖ En informatique, un index est une **structure de données** qui associe à une valeur d'un ensemble d'attributs, l'adresse (ou les adresses/ pointeurs) des enregistrements contenant cette valeur
- ❖ C'est un fichier qui permet de trouver un enregistrement dans une table
 - **Clé d'indexation** = une liste d'un ou plusieurs attributs
 - Une **adresse** : adresse de bloc ou une adresse d'enregistrement
 - **Structure d'index** : enregistrements de la forme [valeur, addr], valeur est la clé
 - L'index est trié sur valeur

❖ Exemple



Fichier de données trié sur le titre

Sans index : le parcours séquentiel du fichier

Avec un index : accès direct à l'enregistrement, amélioration très importante des temps de réponse

Types d'index : Index Primaire

- ❖ Index formé des valeurs des clés primaires d'une table
- ❖ L'unicité fait qu'un index retourne soit une adresse (correspondant à l'enregistrement) soit rien

❖ Exemple

| Id | | Id | Espèce | Sexe | Date de naissance | Nom | Commentaires |
|----|---|----|--------|---------|---------------------|----------------|------------------------|
| 1 |  | 2 | chat | NULL | 2010-03-24 02:23:00 | Roucky | NULL |
| 2 |  | 1 | chien | male | 2010-04-05 13:43:00 | Rox | Mordille beaucoup |
| 3 |  | 3 | chat | femelle | 2010-09-13 15:02:00 | Schtroumpfette | NULL |
| 4 |  | 6 | tortue | femelle | 2009-06-13 08:17:00 | Bobosse | Carapace bizarre |
| 5 |  | 9 | tortue | NULL | 2010-08-23 05:18:00 | NULL | NULL |
| 6 |  | 4 | tortue | femelle | 2009-08-03 05:12:00 | NULL | NULL |
| 7 |  | 7 | chien | femelle | 2008-12-06 05:18:00 | Caroline | NULL |
| 8 |  | 8 | chat | male | 2008-09-11 15:38:00 | Baghera | NULL |
| 9 |  | 5 | chat | NULL | 2010-10-03 16:44:00 | Choupi | Né sans oreille gauche |

Types d'index : Index Secondaire

- ❖ Index sur les attributs qui ne sont pas des clés. Intéressants si
 - L'attribut est presque une clé (peu d'enregistrements ont la même valeur pour cet attribut)
 - Si les enregistrements sont clustérisés : groupage des tuples avec une même valeur pour un attribut dans le même bloc si possible

❖ Exemple

| nom | id | nom | prenom | init_2e_prenom | email |
|------------|----|------------|-----------|----------------|--------------------------|
| Boulian | 1 | Dupont | Charles | T | charles.dupont@email.com |
| Caramou | 2 | François | Damien | V | fdamien@email.com |
| Dupont | 3 | Vandenbush | Guillaume | A | guillaumevdb@email.com |
| Dupont | 4 | Dupont | Valérie | C | valdup@email.com |
| Dupont | 5 | Dupont | Valérie | G | dupont.valerie@email.com |
| François | 6 | François | Martin | D | mdmartin@email.com |
| François | 7 | Caramou | Arthur | B | leroiarthur@email.com |
| Loupiot | 8 | Boulian | Gérard | M | gebou@email.com |
| Sunna | 9 | Loupiot | Laura | F | loulau@email.com |
| Vandenbush | 10 | Sunna | Christine | I | chrichrisun@email.com |

Types d'index : Index Dense et Clairsemé

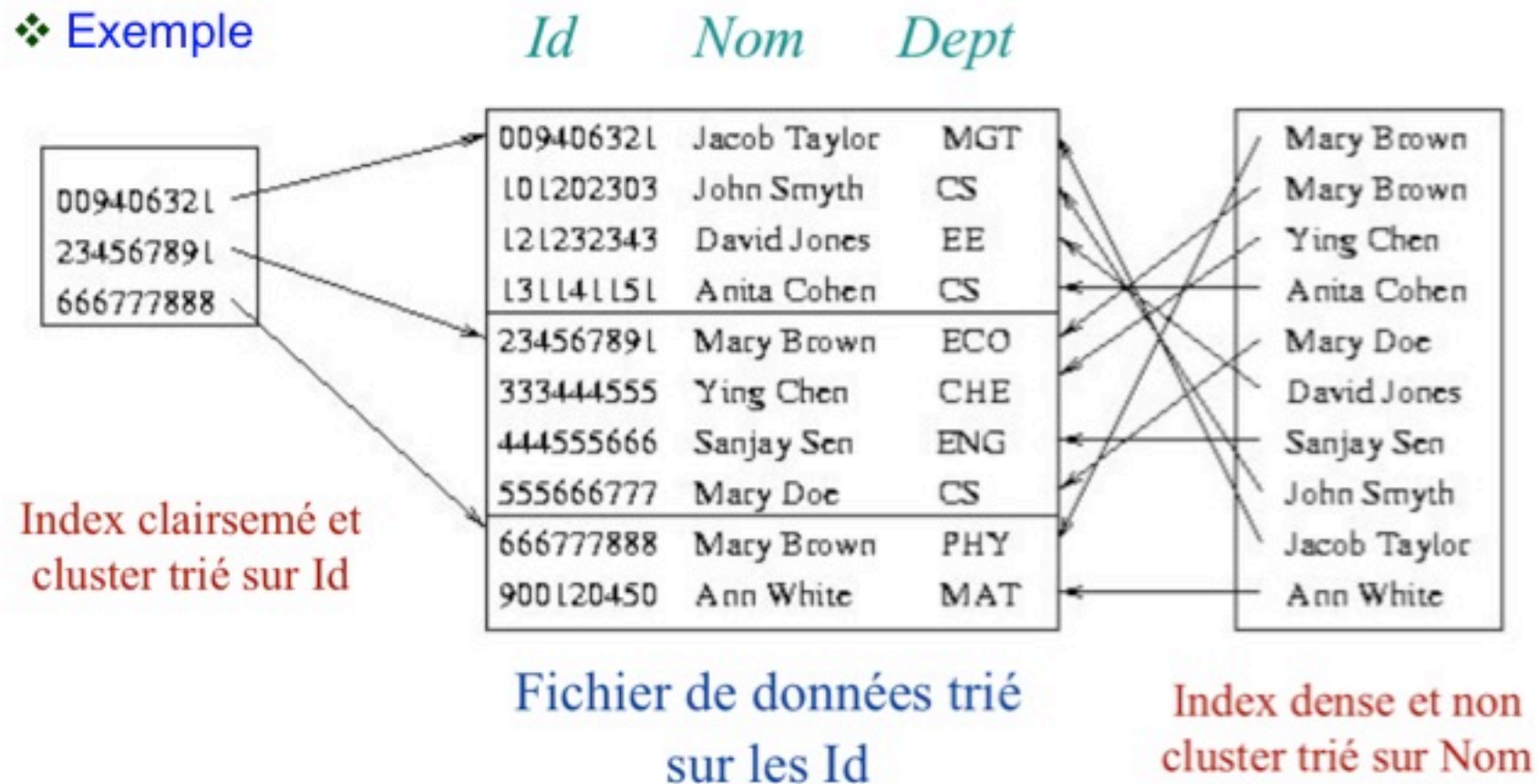
❖ Index dense

- Séquence de blocs avec toutes les clés des enregistrements et leurs pointeurs

❖ Index clairsemé

- Séquence de blocs avec une clé et pointeur par bloc indexé
- Atout : si relation très grande l'index peut tenir dans la mémoire principale

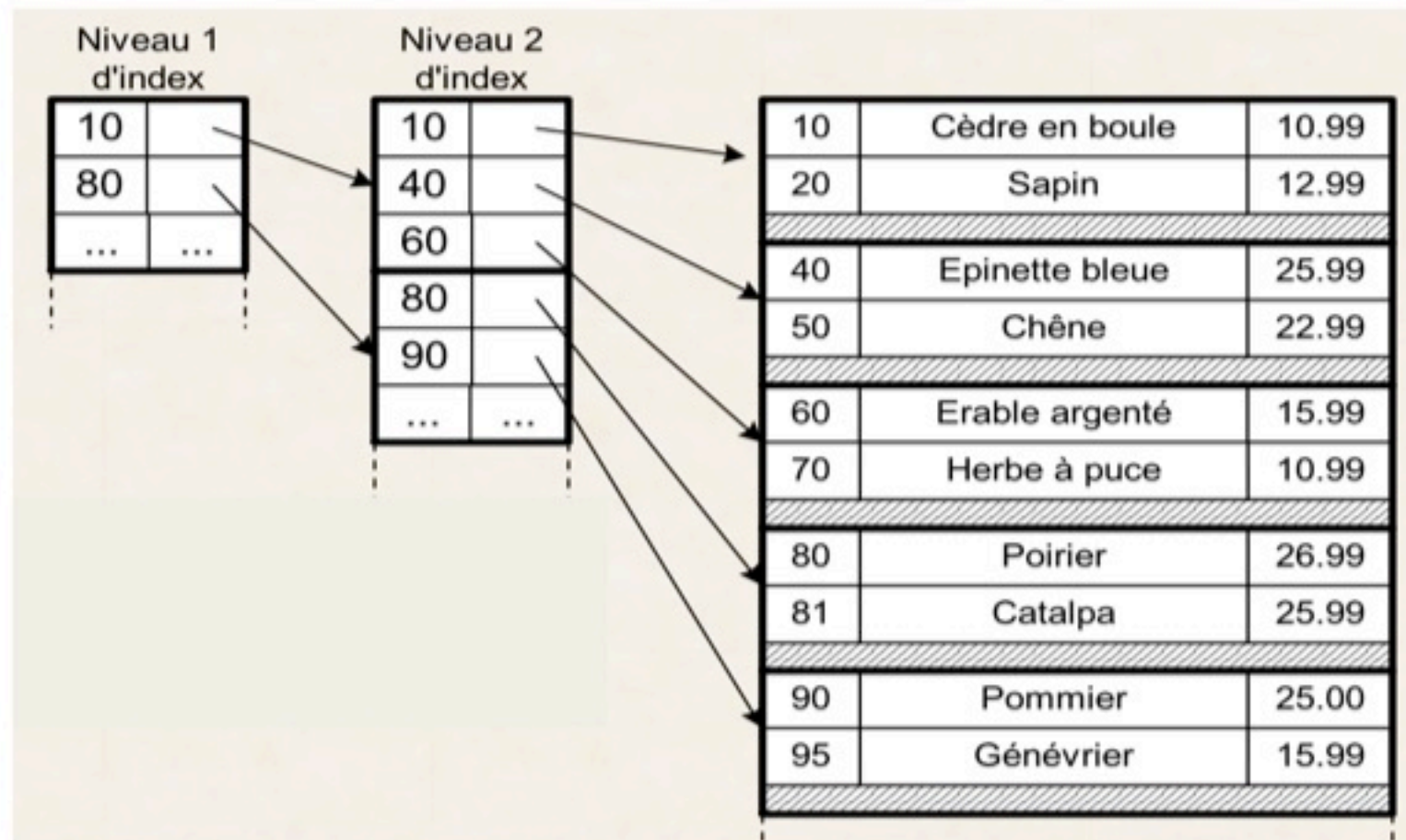
❖ Exemple



Types d'index : index Multi-niveaux

❖ Index sur index pour accélérer la recherche

❖ Exemple



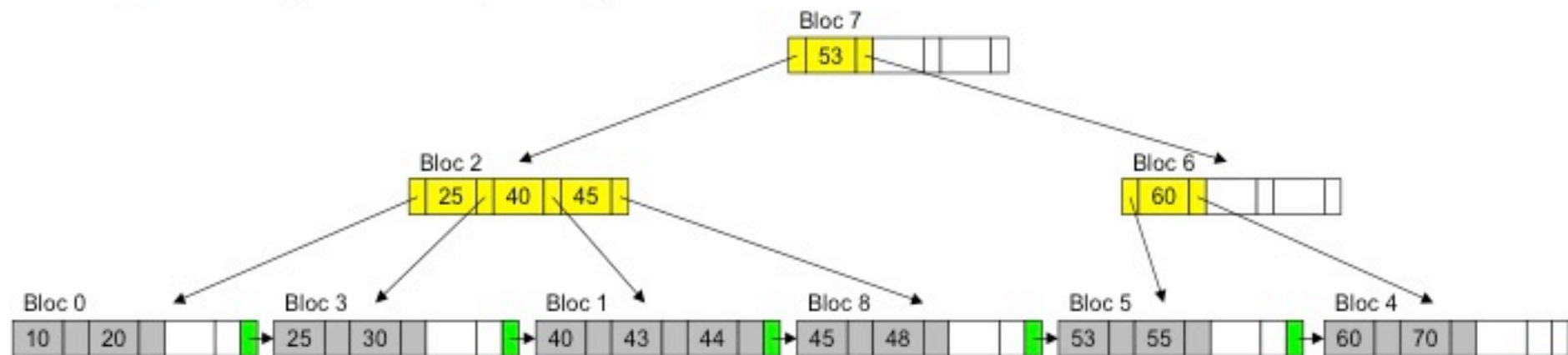
Types d'index : Index Arbres B+

❖ Comme un index multi-niveaux

- Pas besoin d'avoir d'un fichier trié
- La variante Arbre B+ est la plus implémentée dans les SGBD

❖ Les arbres B+

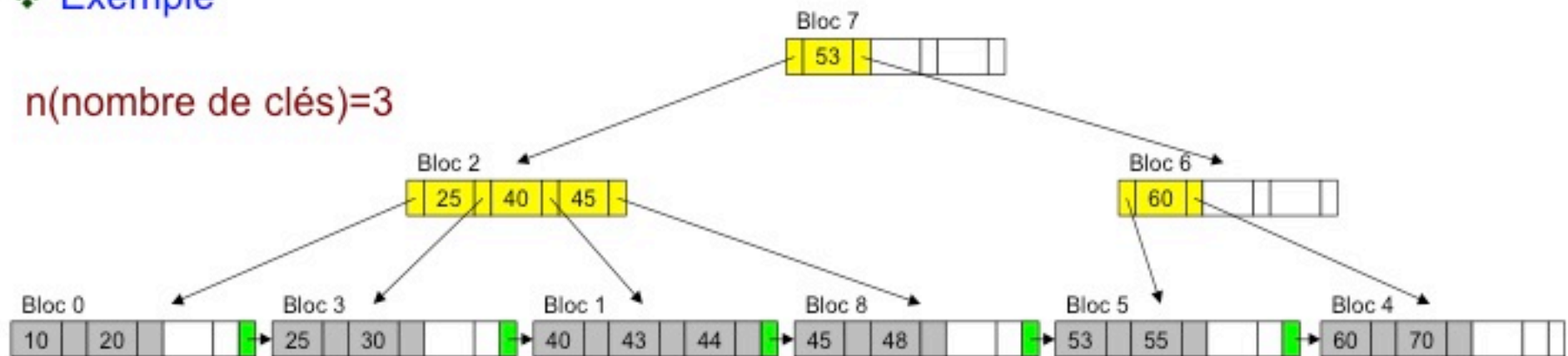
- Sont équilibrés (B : Balanced)
- Maintiennent autant de niveaux que nécessaire pour le fichier à indexer
- Chaque nœud est un (sous) index
- Chaque niveau est soit rempli à moitié soit rempli complètement
- Se réorganisent dynamiquement



Index Arbres B+ : Concepts

❖ Exemple

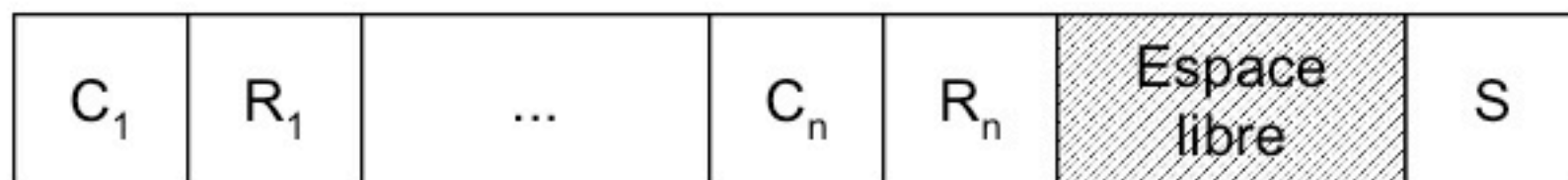
$n(\text{nombre de clés})=3$



- ❖ Chaque nœud est stocké dans un bloc, la capacité des blocs (n comme nombre de clés) détermine la forme de l'arbre
- ❖ La racine a au moins 2 pointeurs (et une clé)
- ❖ Les nœuds intérieurs pointent vers les blocs du niveau suivant et au moins $\lfloor (n+1)/2 \rfloor$ pointeurs doivent être utilisés
- ❖ Toutes les clés sont dans les feuilles de l'arbre et chaque clé a un pointeur vers l'enregistrement correspondant
- ❖ Les clés dans les feuilles sont réparties et ordonnées de gauche à droite et au moins $\lfloor (n+1)/2 \rfloor$ pointeurs doivent être utilisés
- ❖ Dans chaque feuille, un pointeur supplémentaire (le dernier) pointe vers la feuille suivante à droite (vers le bloc avec la suite de clés)

Structure d'une feuille

1. Remplie à moitié au minimum
2. Clés triées : $i < j \Rightarrow C_i < C_j$
3. Clés d'une feuille < clés de la suivante
4. Au même niveau (équilibré)



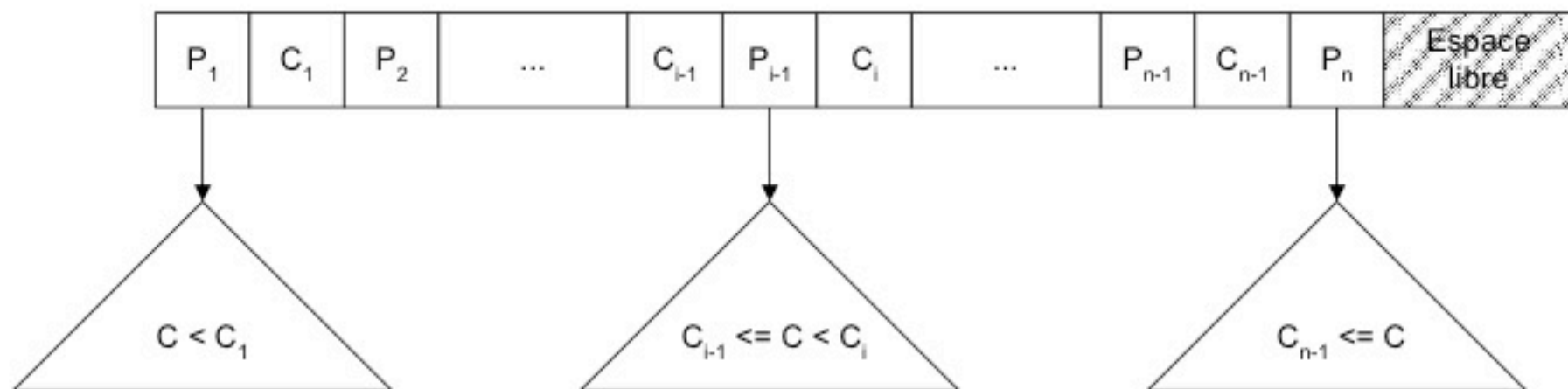
C_i : Clé

R_i : reste de l'enregistrement ou référence

S : Pointeur sur le bloc suivant dans la liste des feuilles

Structure d'un bloc interne

1. Rempli à moitié au minimum
2. Clés triées : $i < j \Rightarrow C_i < C_j$
3. $C_{i-1} \leq$ Clés pointées par $P_{i-1} < C_i$



Index Arbres B+ : Taille d'un arbre B+

- ❖ Un paramètre n est associé à chaque arbre, il détermine la taille et forme des blocs
- ❖ Chaque **bloc** a espace pour **n clés** et **$n+1$ pointeurs**
- ❖ n doit être aussi grand que possible selon la taille d'un bloc

❖ Exemple



- la taille d'un **bloc est de 4096 octets**
- les **clés** sont des entiers de **4 octets**
- les **pointeurs** des entiers de **8 octets**

Nombre maximal de clés à stocker dans un bloc ?

on a $4n + 8(n+1) \leq 4096$

ça fait **$n=340$**

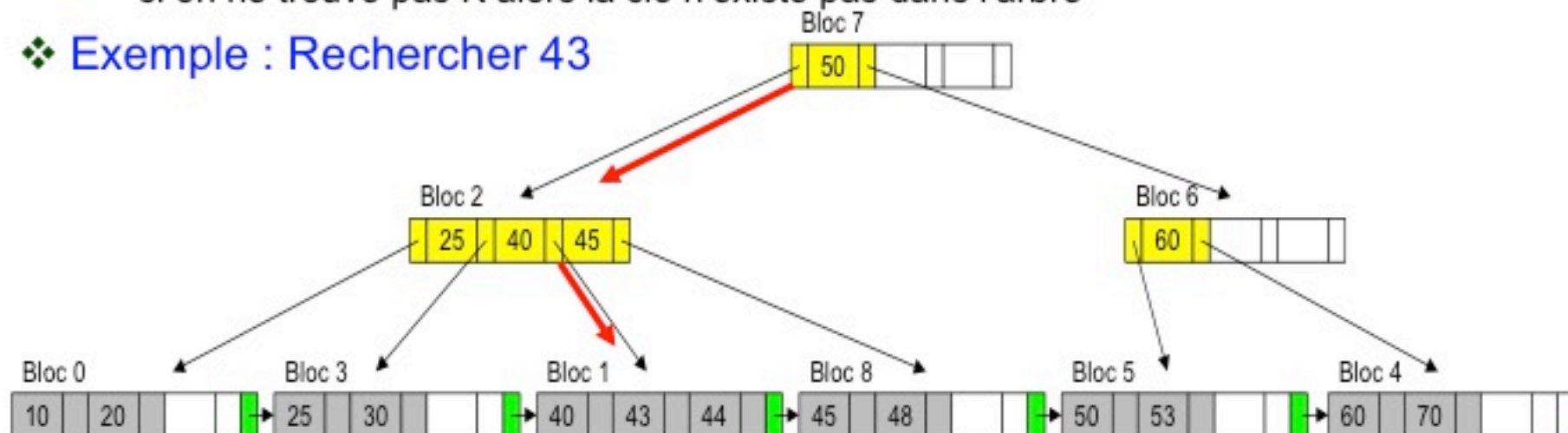
Index Arbres B+ : Taille d'un arbre B+

- ❖ Nombre de clés : n
- ❖ Nombre de pointeurs : $n+1$
- ❖ Nombre de pointeurs au minimum
 - Racine : 2
 - Intermédiaire : $\lfloor (n+1)/2 \rfloor$ (partie entière inférieure)
 - Feuille : $\lceil (n+1)/2 \rceil$ (partie entière supérieure ou par excès)
- ❖ Nombre de feuilles (Clés en total indexées) : $n \cdot (n+1)^H$; H est la hauteur de l'arbre ou nombre de niveaux
- ❖ Clés au minimum : $2 \cdot (n/2+1)^H - 1$
- ❖ Clés au maximum : $(n+1)^{H+1} - 1$

La recherche dans un Arbre B+

- ❖ L'enregistrement recherché est K
- ❖ La recherche récursive commence à la racine et quand on arrive aux feuilles on trouve le pointeur de l'enregistrement
- ❖ Sur tous les nœuds, on fait la même comparaison pour savoir quel nœud fils doit être examiné ensuite
- ❖ Supposons un nœud avec clés K_1, K_2, \dots, K_n
 - si $K < K_1$, le fils à examiner est le premier
 - Autrement si $K_1 \leq K < K_2$, le fils à examiner est le second fils, et ainsi de suite
 - Lorsque le nœud est une feuille, si on trouve K alors on obtient le pointeur correspondant, si on ne trouve pas K alors la clé n'existe pas dans l'arbre

❖ Exemple : Rechercher 43



Insertion dans un arbre B+

1. D'abord on fait une recherche pour trouver la place de la nouvelle clé
2. Itérativement, on essaie d'insérer la nouvelle clé dans le nœud (feuille si la première fois) correspondante s'il y a de la place (éventuellement décaler les clés)
3. S'il n'y a pas de place, le nœud est découpé en 2 et les clés sont réparties entre les 2 nœuds (chaque nœud est maintenant à moitié plein)
 - Ce découpage fait que le nœud parent du nœud découpé ait besoin d'avoir un nouveau pointeur
4. Et on recommence à 2, et ainsi de suite jusqu'à la racine

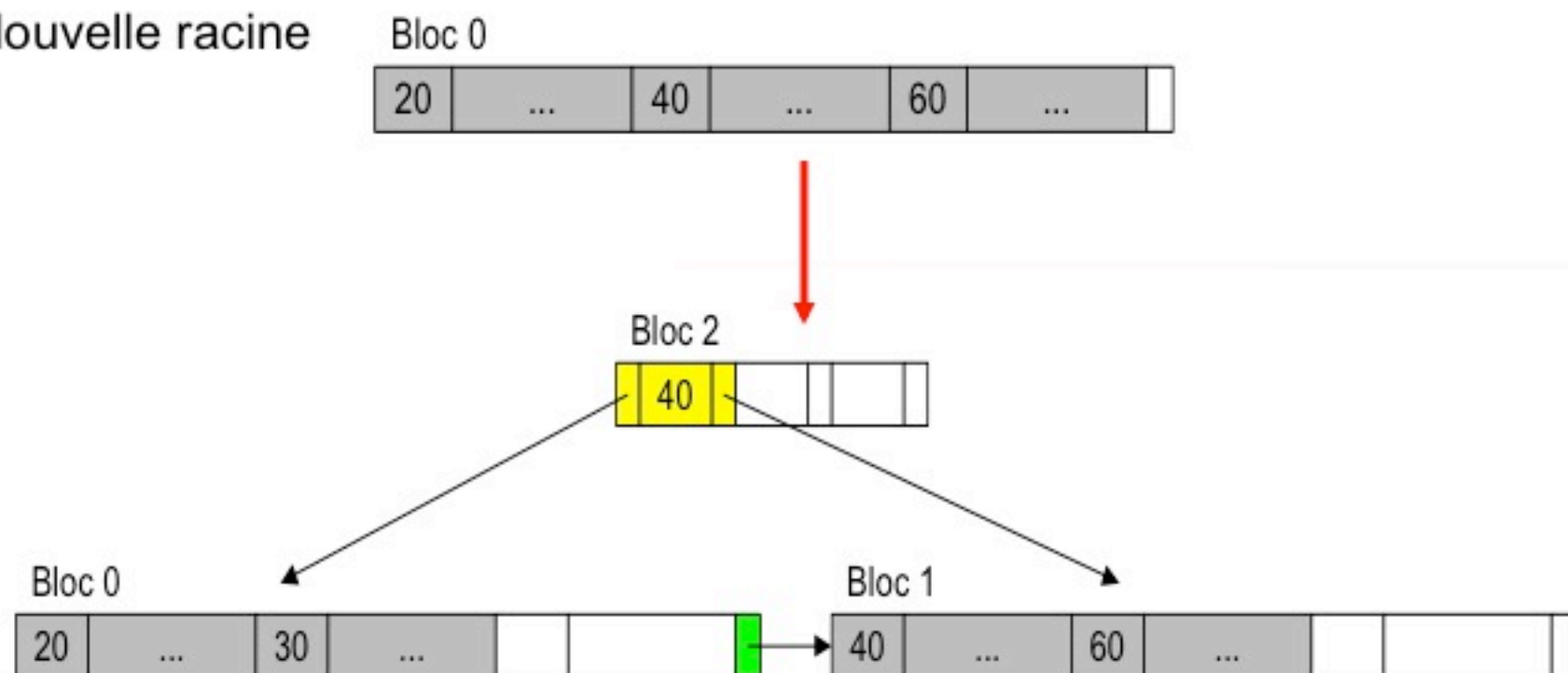
Insertion dans un arbre B+ : Exemples

❖ *Insérer 20 ensuite 60*



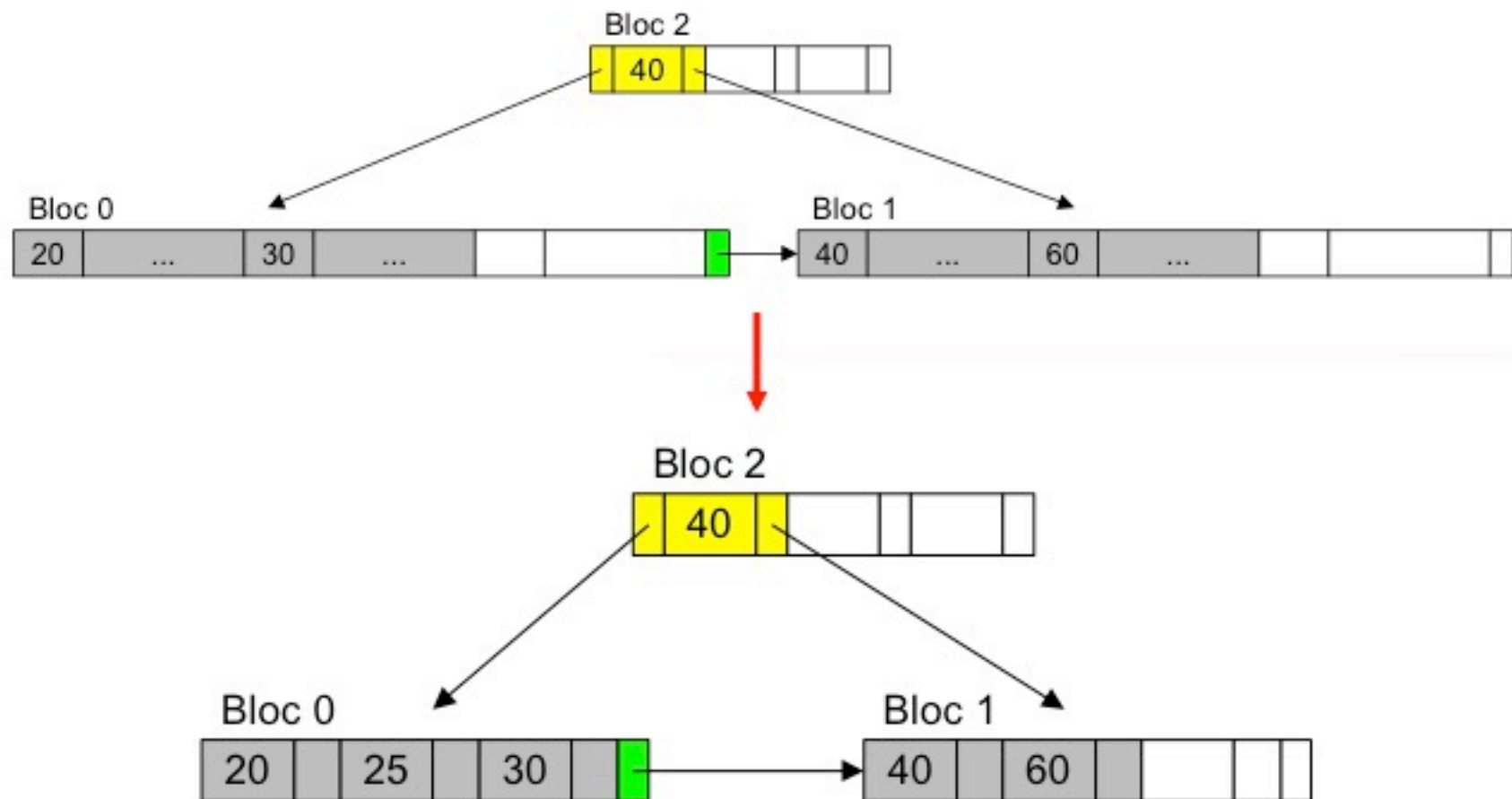
Insertion dans un arbre B+ : Débordement et division

- ❖ Insertion de 30
- ❖ Débordement et la division du bloc 0
- ❖ 40 est promue
- ❖ Nouvelle racine



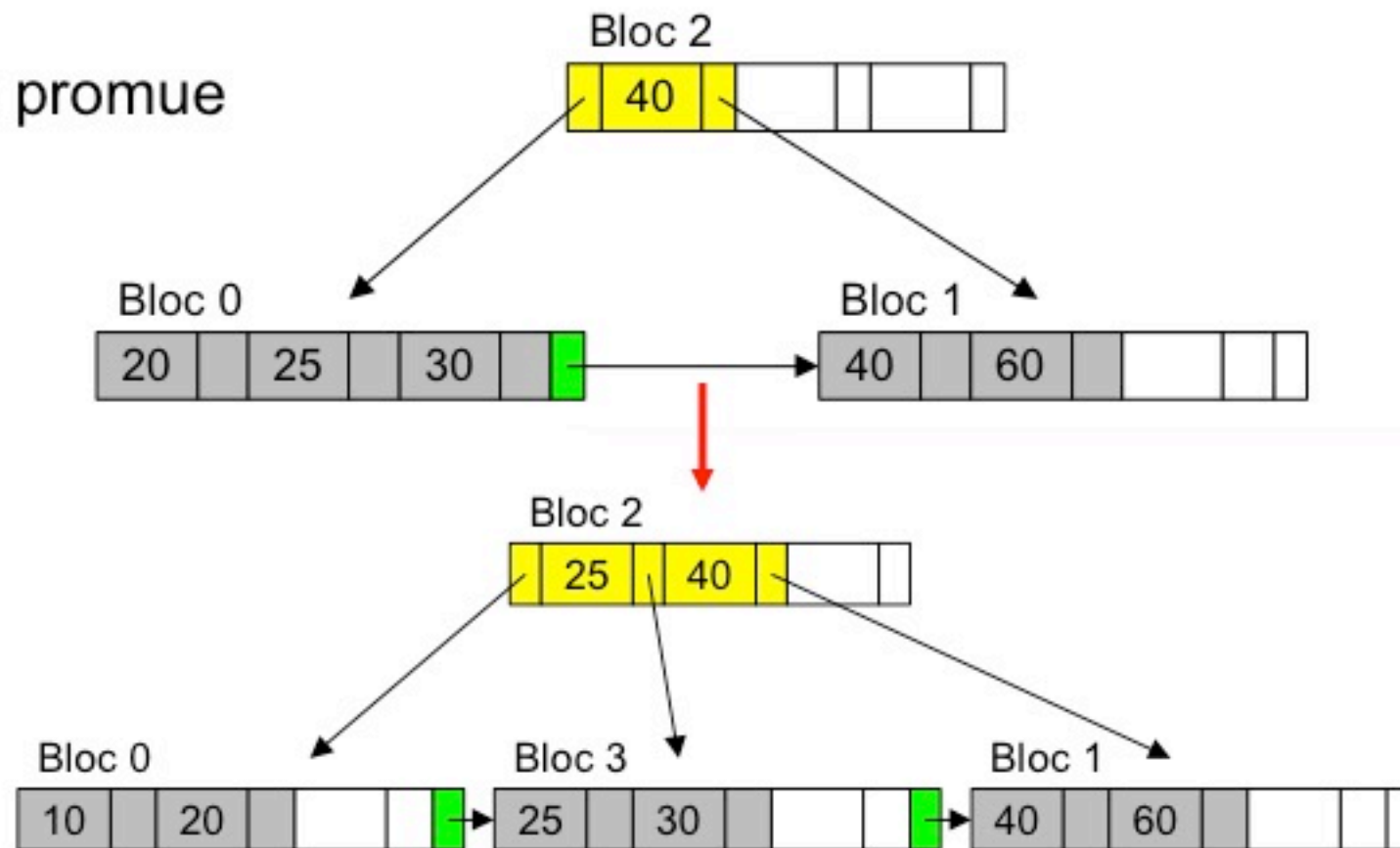
Insertion dans un arbre B+

- ❖ Insérer 25
- ❖ Décalage de 30



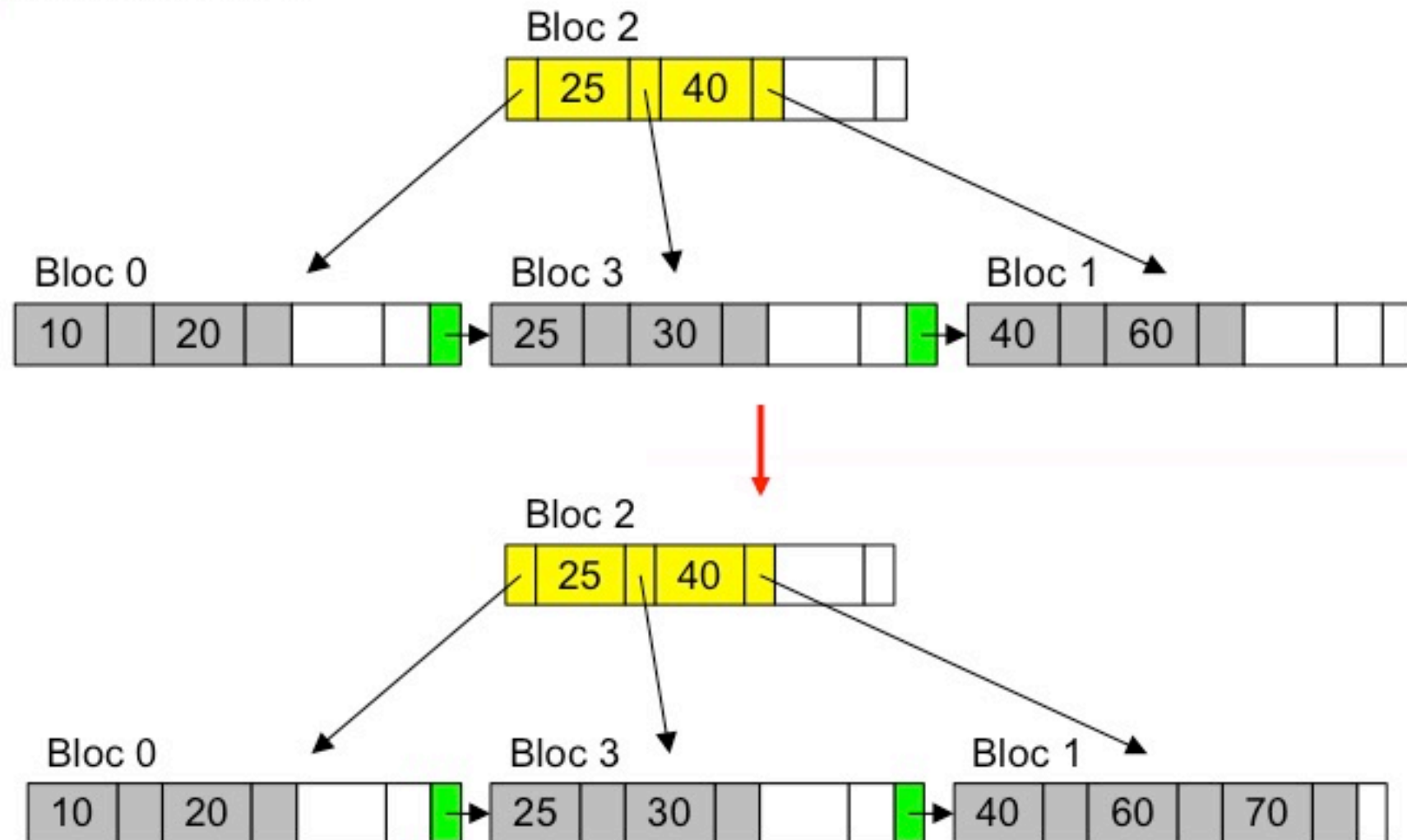
Insertion dans un arbre B+

- ❖ Insertion de 10
- ❖ Débordement et la division du bloc 0
- ❖ 25 est promue



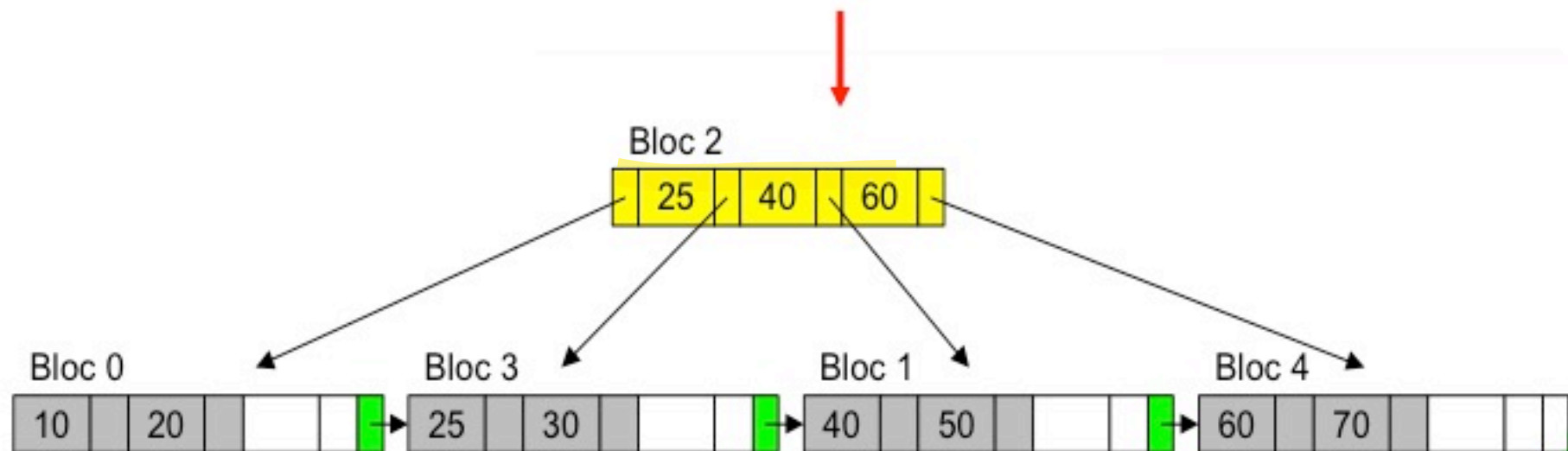
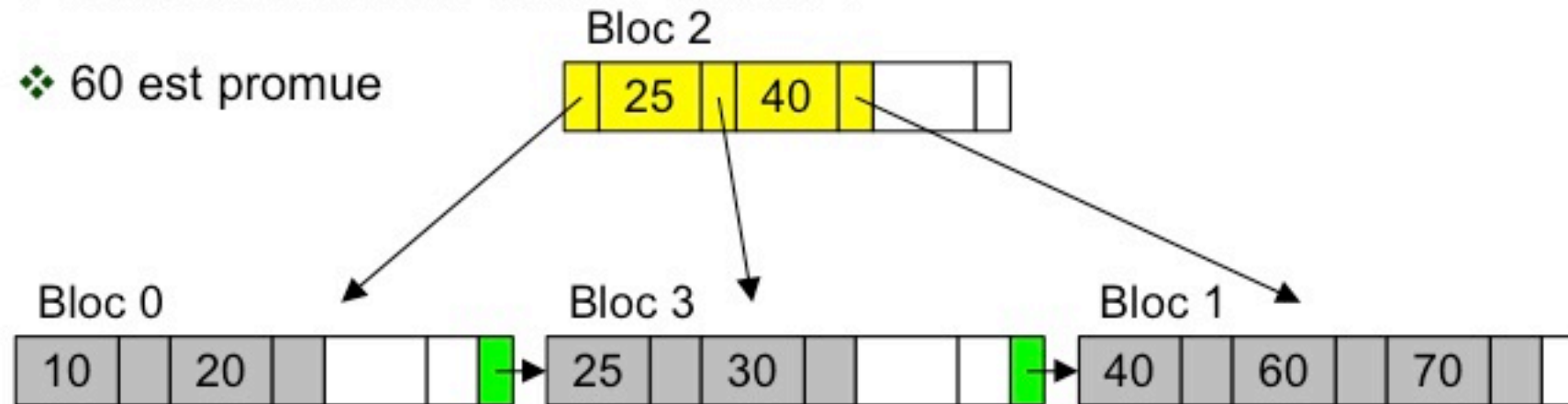
Insertion dans un arbre B+

❖ Insertion de 70



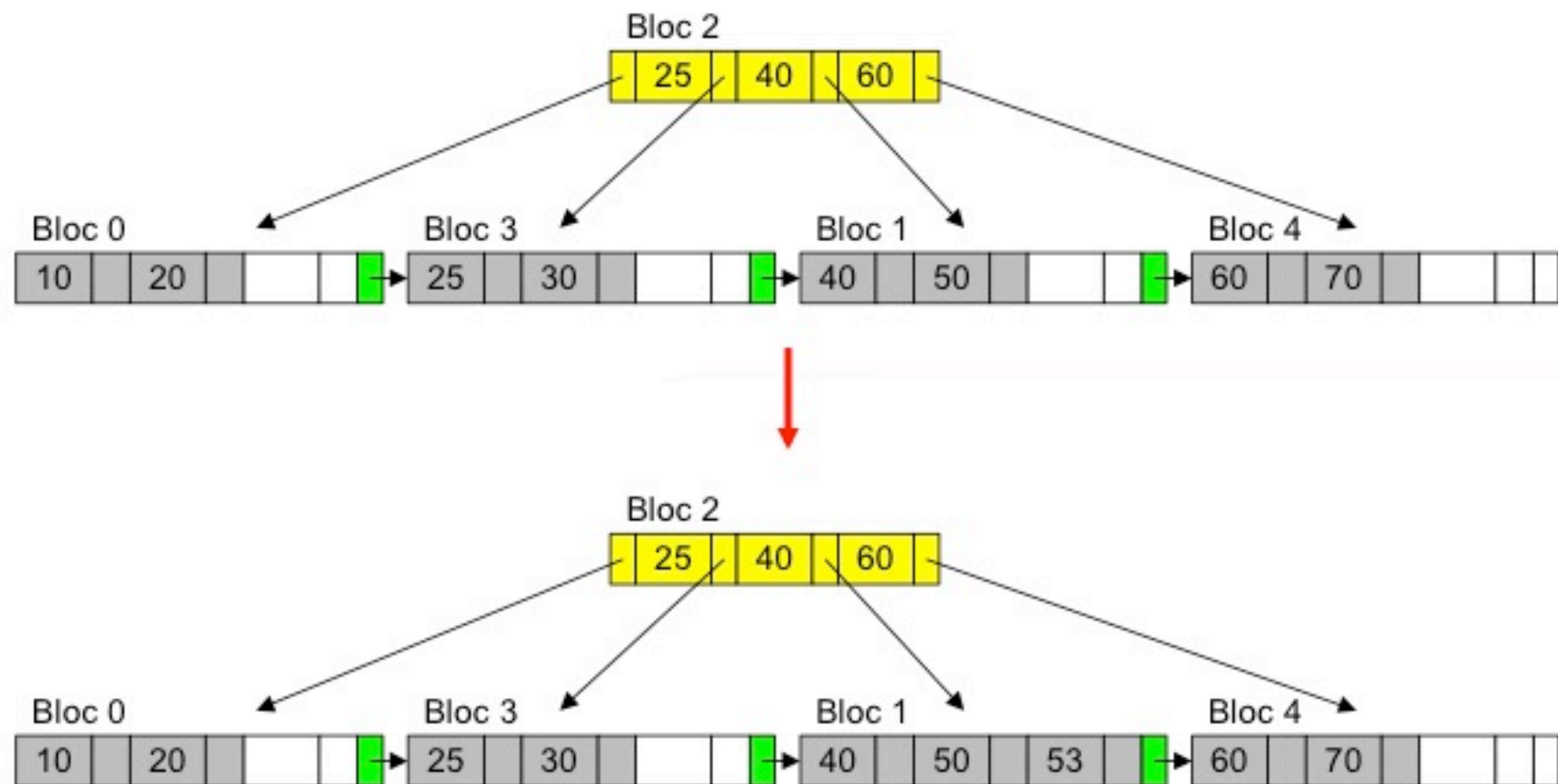
Insertion dans un arbre B+

- ❖ Insertion de 50
- ❖ Débordement et la division du bloc 1
- ❖ 60 est promue



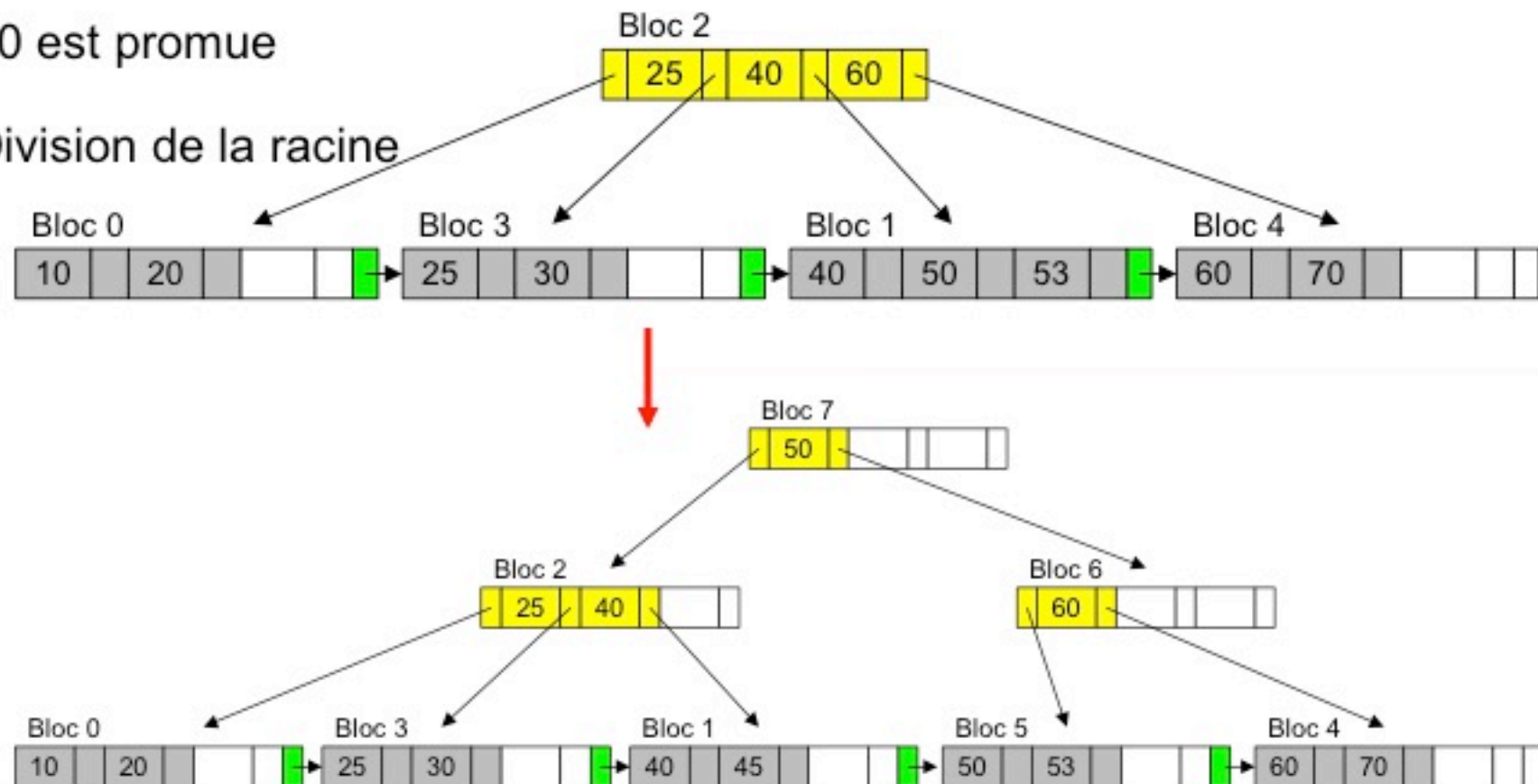
Insertion dans un arbre B+

❖ Insertion de 53



Insertion dans un arbre B+

- ❖ Insertion de 45
- ❖ Division du bloc 1
- ❖ 50 est promue
- ❖ Division de la racine

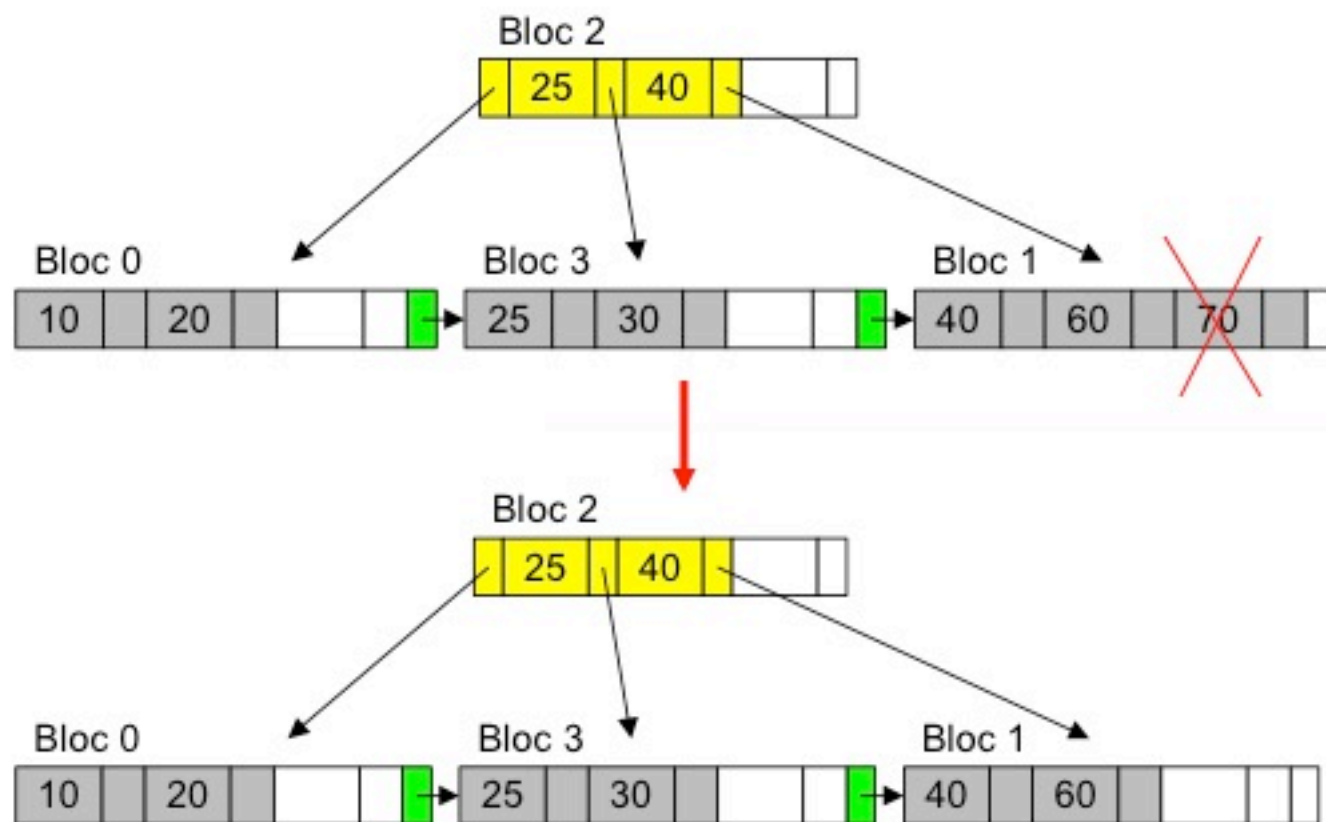


Supprimer dans un arbre B+

1. D'abord on fait la recherche du nœud qui a la clé à supprimer, supposons que c'est le nœud N
2. Ensuite, on supprime la clé, son pointeur et l'enregistrement correspondant
3. Si le nœud N a encore le nombre minimal de clés obligatoires $\lfloor (n+1)/2 \rfloor$ la suppression est terminée
4. Autrement, si un nœud frère adjacent de N, supposons M, a plus de clés que le minimal, alors on déplace une clé de M vers N et le nœud parent de M doit possiblement réajuster ses clés et la suppression est terminée
5. Si M est au nombre minimal de clés alors on fusionne N et M, les clés du parent doivent être réajustées et une clé doit être supprimée. Ensuite on recommence à 3 avec le nœud parent comme N

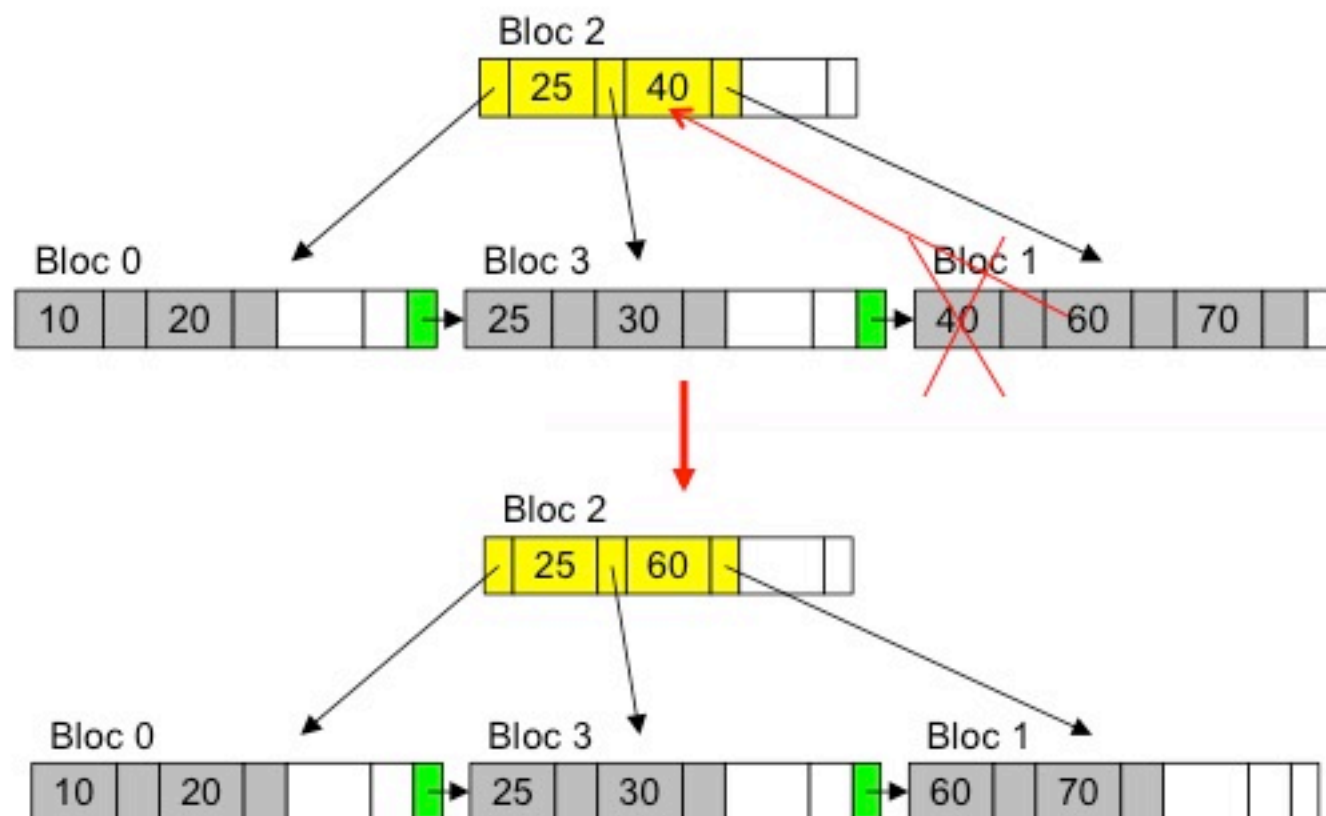
Suppression dans un arbre-B+

- ❖ Supprimer 70
 - minimum préservé



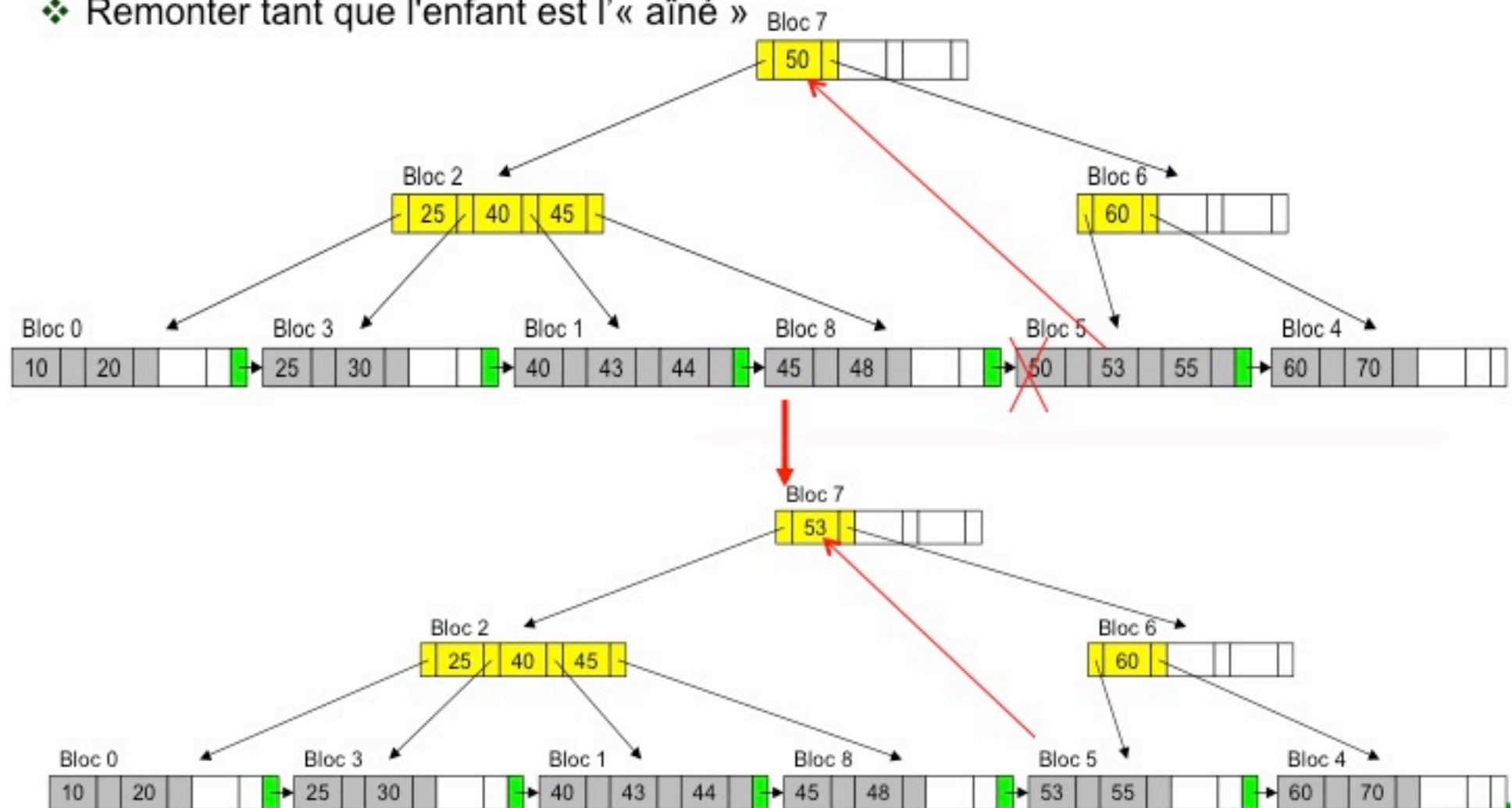
Première clé du bloc et pas la première feuille

- ❖ Supprimer 40
- ❖ Remplacer dans le parent



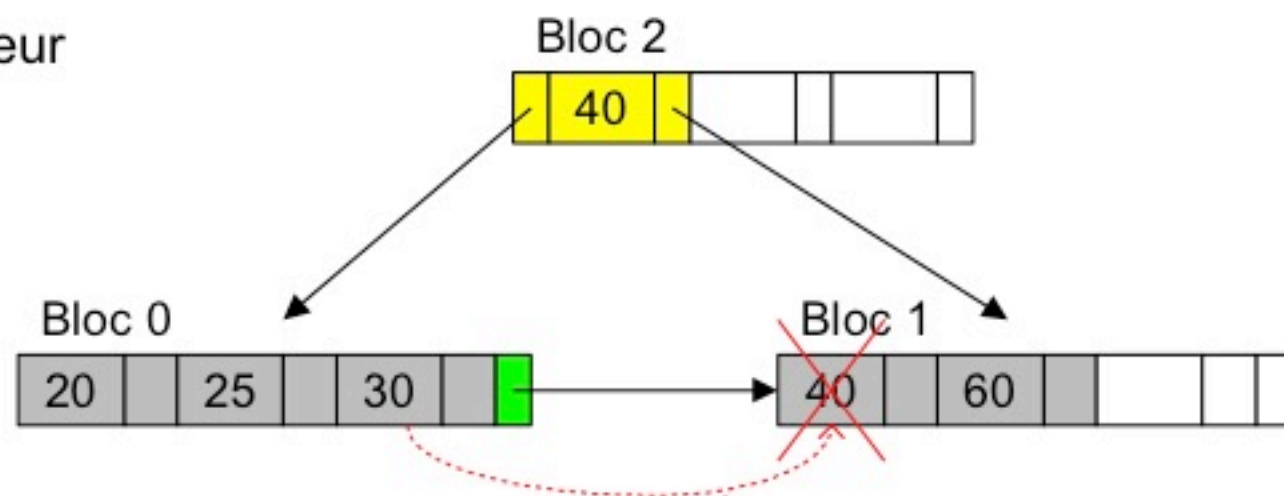
Première clé du bloc et pas la première feuille

- ❖ Supprimer 50
- ❖ Remonter tant que l'enfant est l'« aîné »



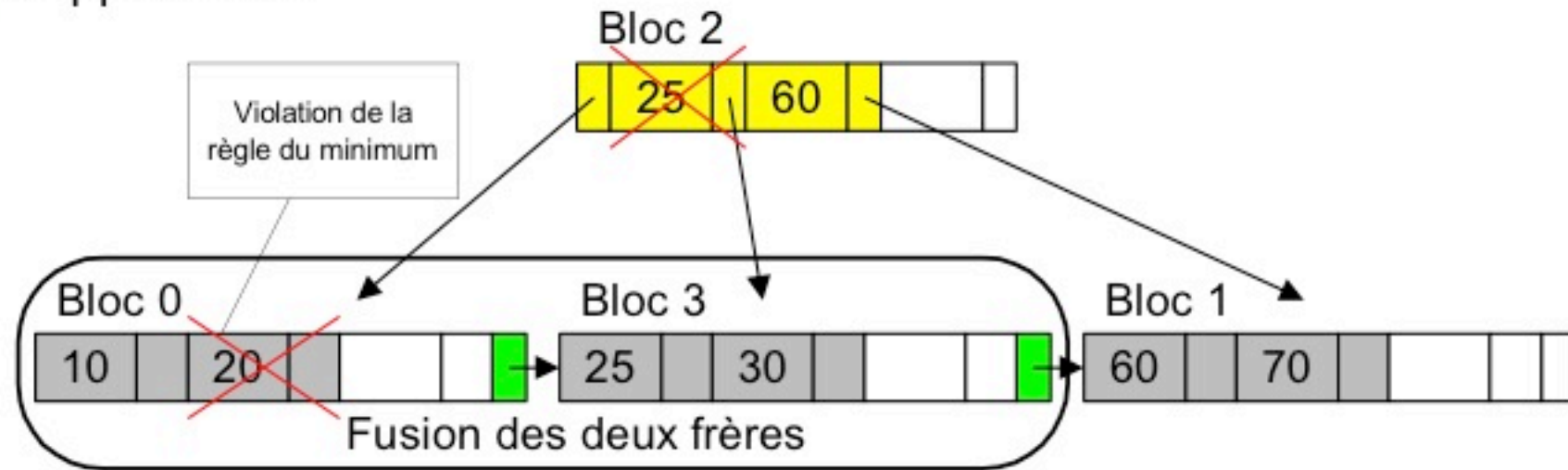
Violation du minimum : redistribution si possible

- ❖ Supprimer 40
- ❖ Ajuster séparateur



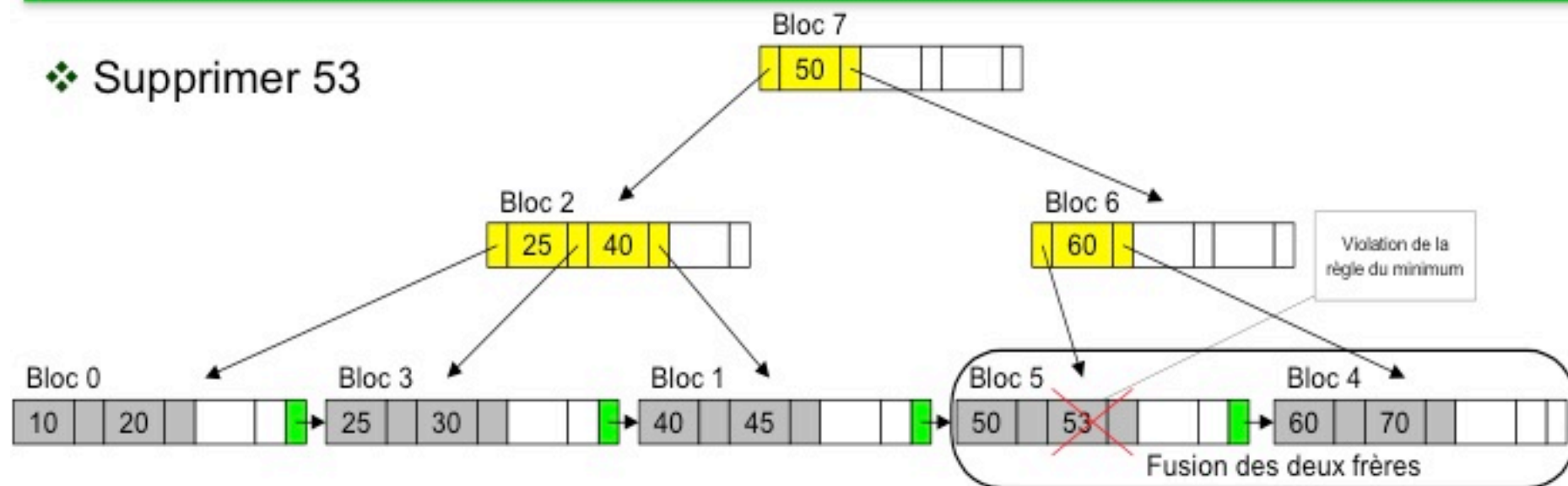
Violation du minimum : fusion

❖ Supprimer 20

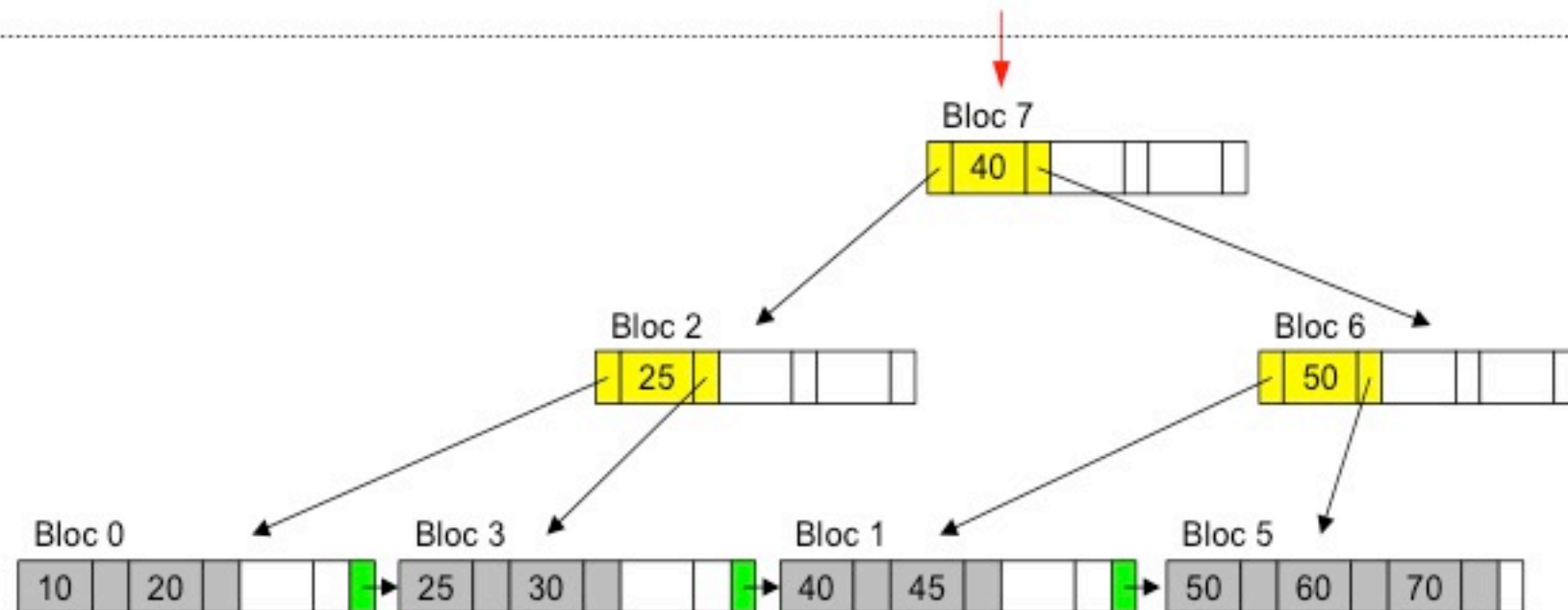
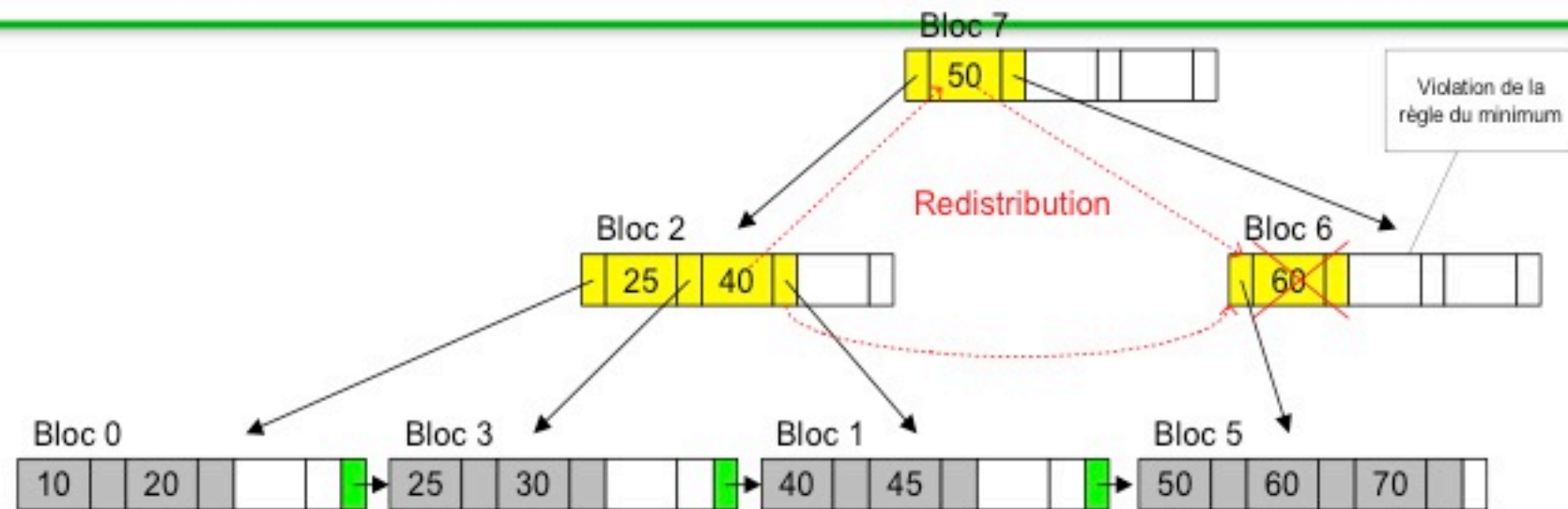


Cas de fusion de feuilles et de redistribution au niveau du parent

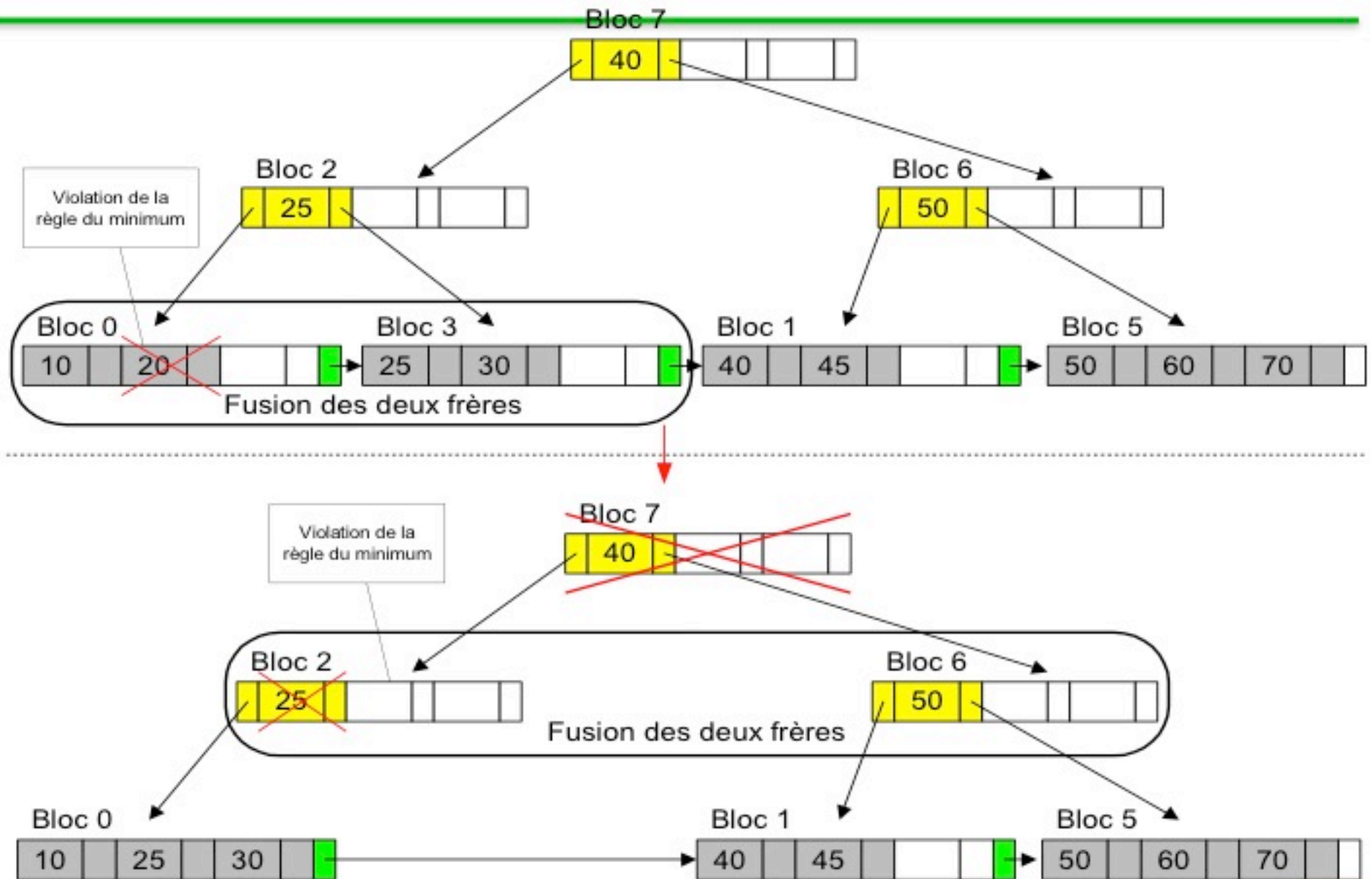
❖ Supprimer 53



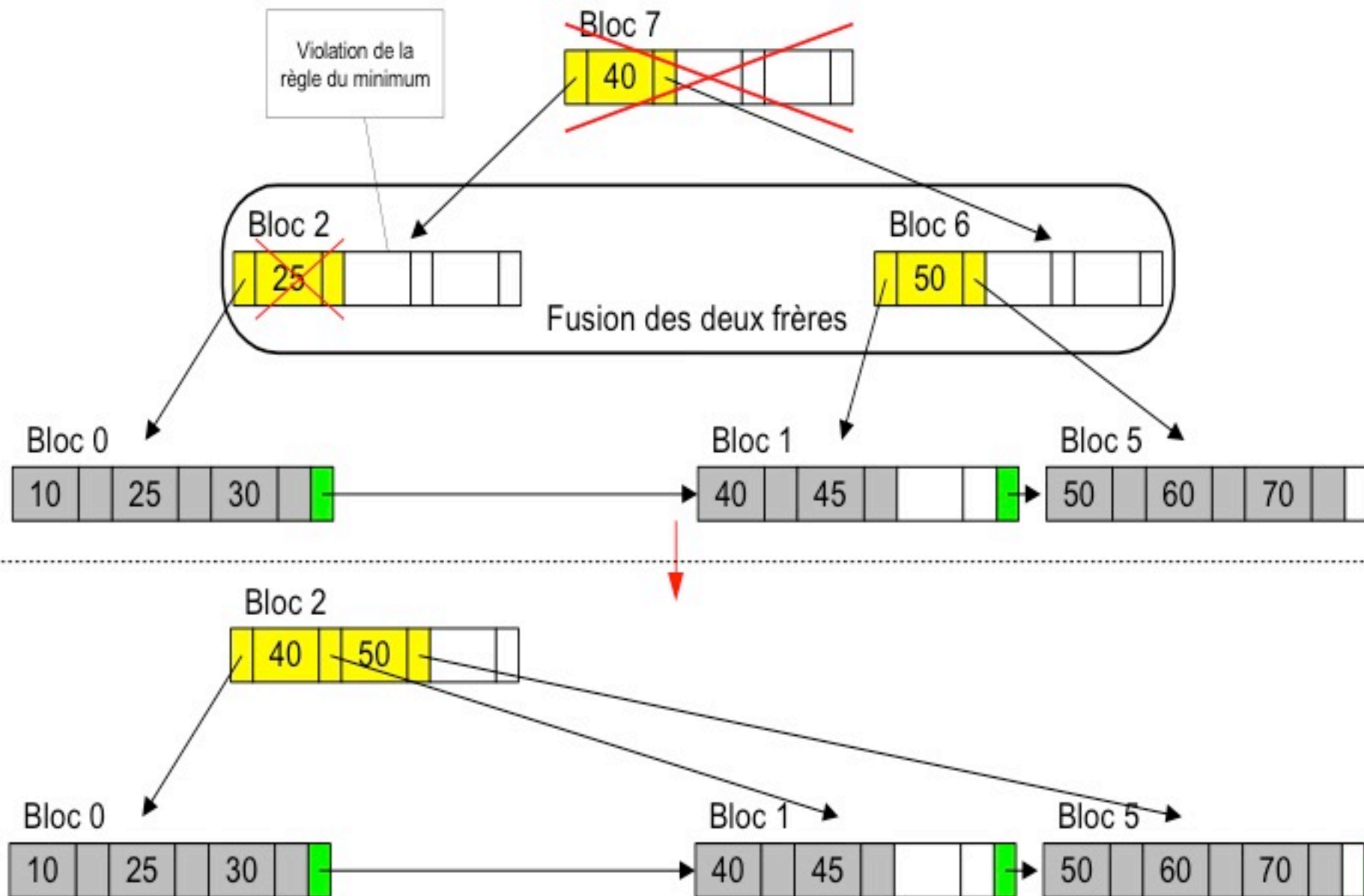
Cas de fusion de feuilles et de redistribution au niveau du parent (suite)



Cas de fusion en cascade



Cas de fusion en cascade (suite) : réduction de la hauteur



Avantages des Arbres B+

- ❖ Restent équilibrés
- ❖ Les blocs sont partiellement vides pour accélérer les insertions et les suppressions
- ❖ Si n est suffisamment large, la réorganisation de nœuds (découpage et merge) est rare et souvent cela reste au niveau des feuilles
- ❖ Dans chaque bloc une recherche binaire peut être faite
- ❖ Excellentes performances pour
 - l'interrogation et l'update : nombre de I/O = nombre de niveaux + 1
 - insert et delete : coût d'une interrogation + le coût de réorganisation
- ❖ Les performances ne se dégradent pas lorsque les tables grossissent
- ❖ Efficace pour recherches sur un rang de valeurs ($<$, $>$) et l'égalité

Création d'un index : syntaxe

```
CREATE INDEX <nomi> ON <nom_table>  
(<attr1> [ASC,DESC],  
...  
<attrn> [ASC,DESC]) [propriétés de l'index]
```

❖ Par défaut l'index est un arbre B+

❖ Exemple

```
CINEMA(NomCinema, Adresse, Gerant)  
SALLE(NomCinema, NoSalle, Capacite)
```

```
CREATE INDEX index_cinema ON CINEMA(Adresse desc);
```

Les mises à jour et les indexes

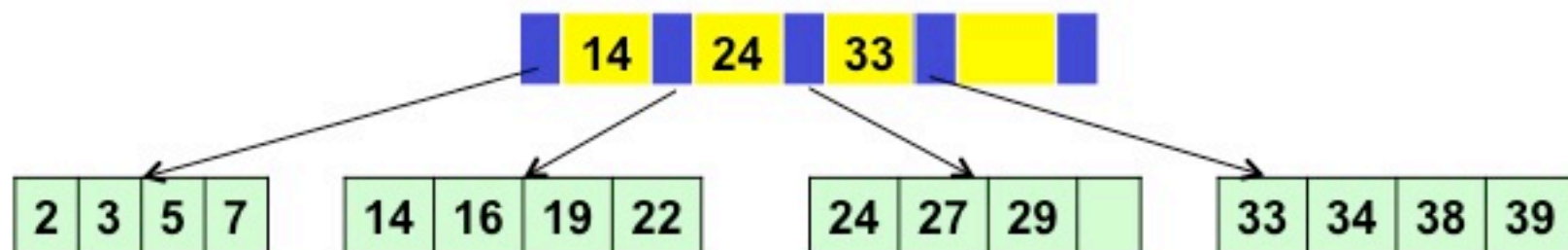
- ❖ Chaque modification de l'ensemble d'enregistrements implique une mise à jour des indexes
- ❖ Les indexes servent à optimiser, mais il faut vérifier que les performances ne sont pas dégradées par la présence de nombreux indexes (ou/et mal définis)
- ❖ L'existence d'un index sur un attribut peut accélérer l'exécution de requêtes portant sur cet attribut (valeurs/rang de valeurs et jointures)
- ❖ MAIS les indexes rendent les insertions, suppressions et modifications plus complexes et chères en temps

Exercice

- ❖ Supposez un arbre a une taille de bloc de 8192 octets, les clés et les pointeurs sont des entiers de 4 octets. Quel est le nombre de clés que peut stocker un bloc ?

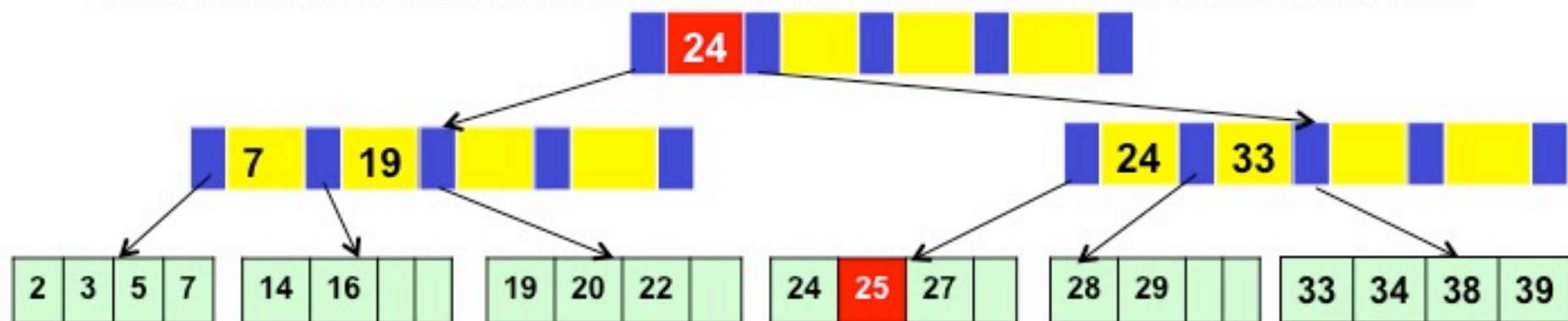
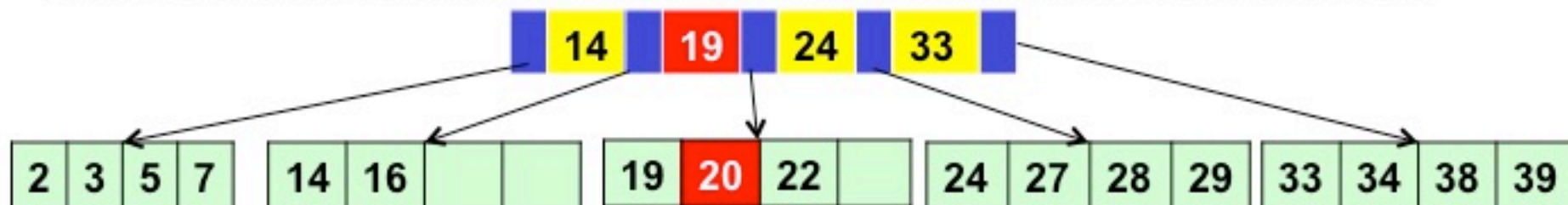
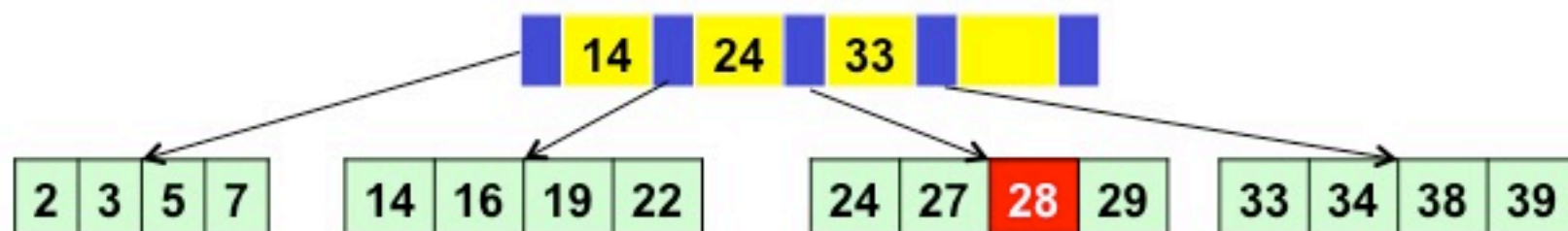
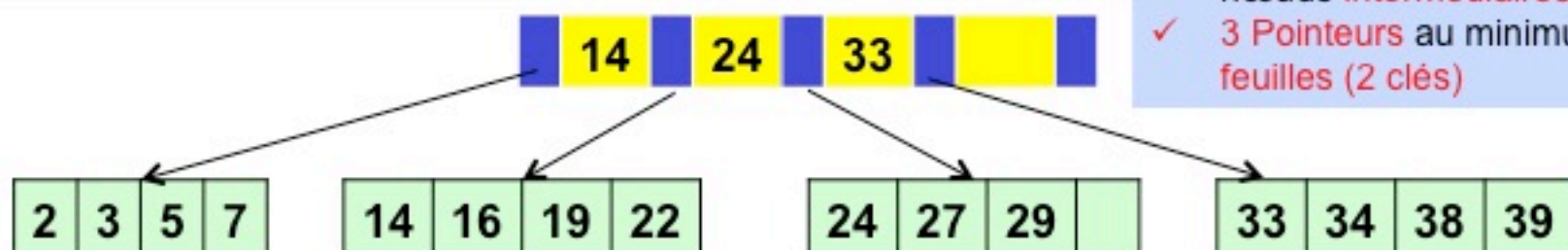
$$n \times 4 + (n+1) \times 4 < 8192$$
$$n \leq 1023 \rightarrow \text{Max}(n) = 1023$$

- ❖ Donnez le résultat de l'arbre après les insertions suivantes : 28, 20, 25
- ❖ Supprimer 29

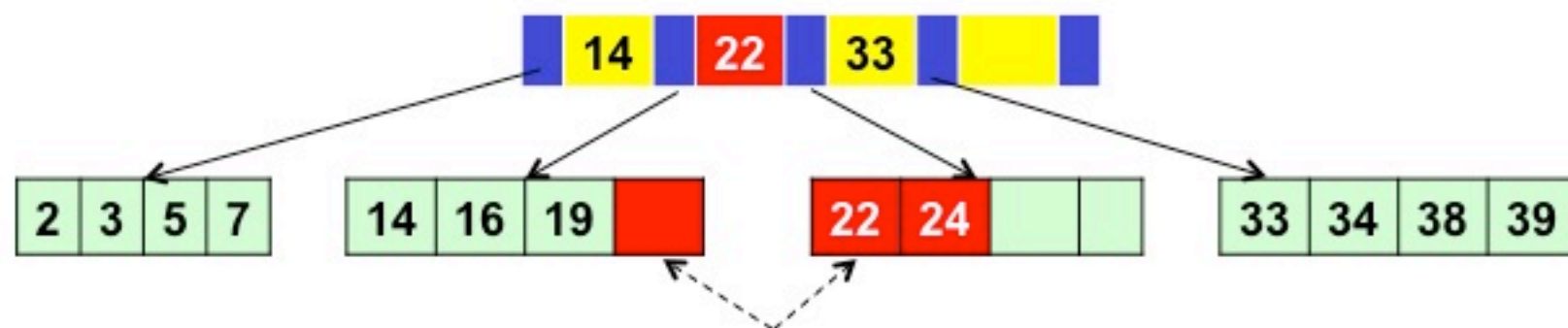
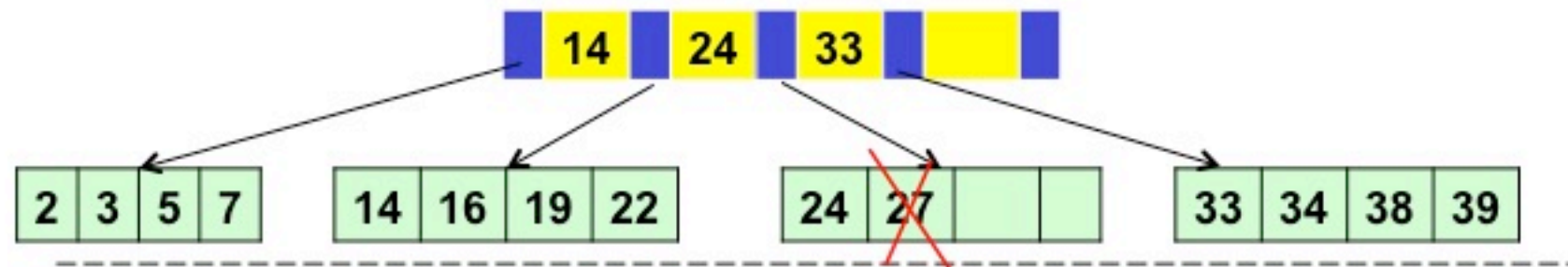
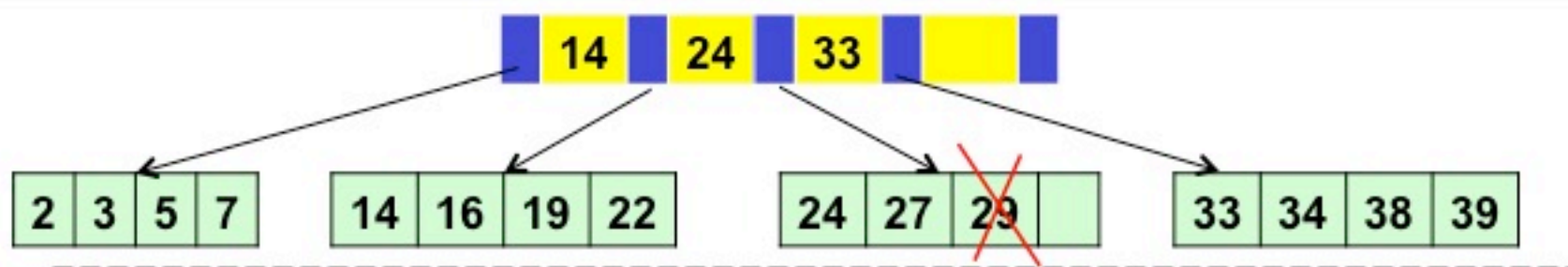


Ajout de 28, 20, 25

- ✓ $n=4$
- ✓ 2 Pointeurs au minimum sur les nœuds intermédiaires (1 clé)
- ✓ 3 Pointeurs au minimum sur les feuilles (2 clés)



Supprimer 29, 27



Redistribution de clés