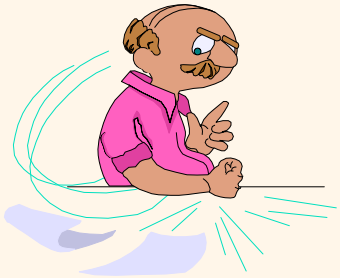
Bases de Données Avancées

Ioana Ileana

Université Paris Descartes

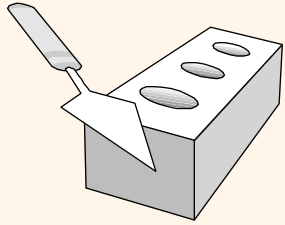
LIPADE
Laboratoire d'Informatique PAris DEscartes

Cours 5: Index de type arbre (tree indexes) 2ème partie

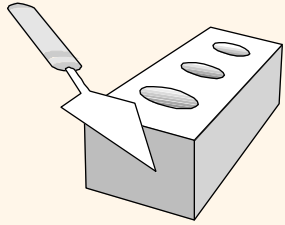


- Diapos traduites et adaptées du matériel fourni en complément du livre Database Management Systems 3ed, par Ramakrishnan et Gehrke ; un grand merci aux auteurs pour la réalisation et la disponibilité de ce matériel !
- Les diapos originales (en anglais) sont disponibles ici :
  http://pages.cs.wisc.edu/~dbbook/openAccess/thirdEdition/slides/slides3ed.html
- Plus particulièrement, ce cours touche aux éléments dans les Chapitre 10 du livre ci-dessus; lecture conseillée! ;)

ISAM : le problème des pages overflow

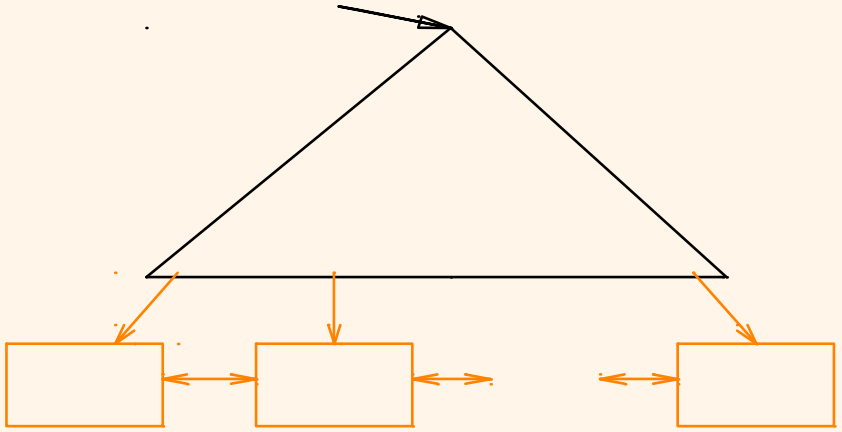


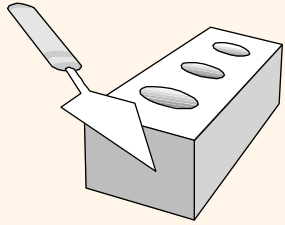
- ISAM : structure statique, les pages primaires feuilles sont en nombre fixe et allouées « au tout début » (à la construction de la structure)
- Pour accommoder les insertions, des pages overflow sont créées et remplies avec les clés / records insérés ! Pour minimiser le coût, souvent ces pages ne sont pas triées (les entrées y sont donc « dans le désordre »)
- → Si « longue chaîne de pages overflow », cela pénalise le temps de recherche (et d'insertion / suppression) !
  - Pour une feuille, on peut avoir à examiner en réalité plein de pages ; de plus, comme il n'y a pas eu de tri, dans ces pages on ne peut pas utiliser la recherche dichotomique !



B+ Tree : l'index le plus utilisé

- ISAM : structure statique ; problème avec les insertions
- B+ Tree : structure dynamique, qui s'adapte beaucoup mieux aux insertions et suppressions !
- Comme dans ISAM, dans le B+ Tree les nœuds intermédiaires « dirigent la recherche » et les nœuds feuilles contiennent les entrées de données
  - Avec l'avantage que les opérations d'insertion et suppression gardent l'arbre équilibré ! Ce qui rend efficace toute recherche / insertion / suppression !
  - Mais, comme il y a création et suppression dynamique des pages feuilles, elles ne peuvent plus être allouées de manière séquentielle ! Il faut donc les ***chaîner*** (pour faire un parcours dans le cas d'une recherche par intervalle...)





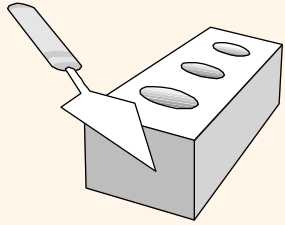
B+ Tree: ordre de l'arbre

- Comme pour ISAM, dans un nœud intermédiaire on a m clés et m+1 pointeurs / fils ; cela nous fait m entrées d'index !
- Dans les nœuds feuilles, on a m entrées de données
- Paramètre **d** : l'*ordre* de l'arbre = mesure de la capacité maximale en entrées d'un noeud: **maximum 2*d entrées dans chaque noeud!**
- Minimum 50% de "remplissage" pour chaque nœud :

  d<= nombre d'entrées <=2*d

  - … sauf pour la racine où on autorise moins de remplissage:

    1<=nombre d'entrées racine <=2*d
- On obtient que le nombre maximal de fils d'un nœud est égal à 2*d+1 !

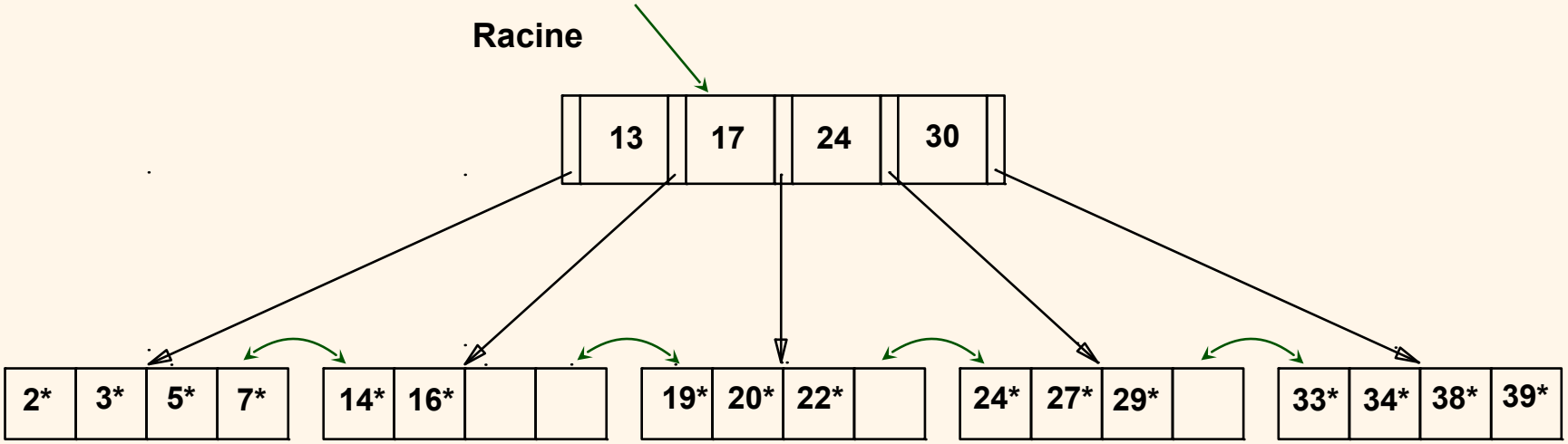


Les B+ Trees en pratique

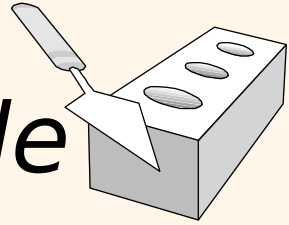
- Ordre typique (d) : 100.
- Taux de remplissage typique: 2/3 (66.5%)
  - Fanout (nb de fils) moyen = 133
- Capacité totale typique:
  - Hauteur 3: 133^3 = 2,352,637 records
  - Hauteur 4: 133^4 = 312,900,700 records
  - Hauteur 5: 133^5 = 41,615,795,893 records

Recherche dans un B+ Tree

- Démarrer la recherche à la racine
- Comme dans ISAM , les nœuds intermédiaires dirigent la recherche, et on a les même propriétés de « séparateurs » sur les clés des nœuds intermédiaires !
- Si intervalle: parcourir les pages suivantes grâce au chaînage
- Rechercher 5*? 15*? toutes les entrées >= 24* ?

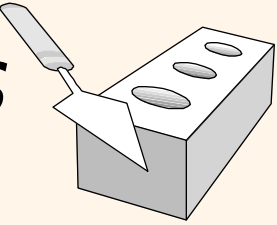


B+Tree: insertion d'une entrée de données



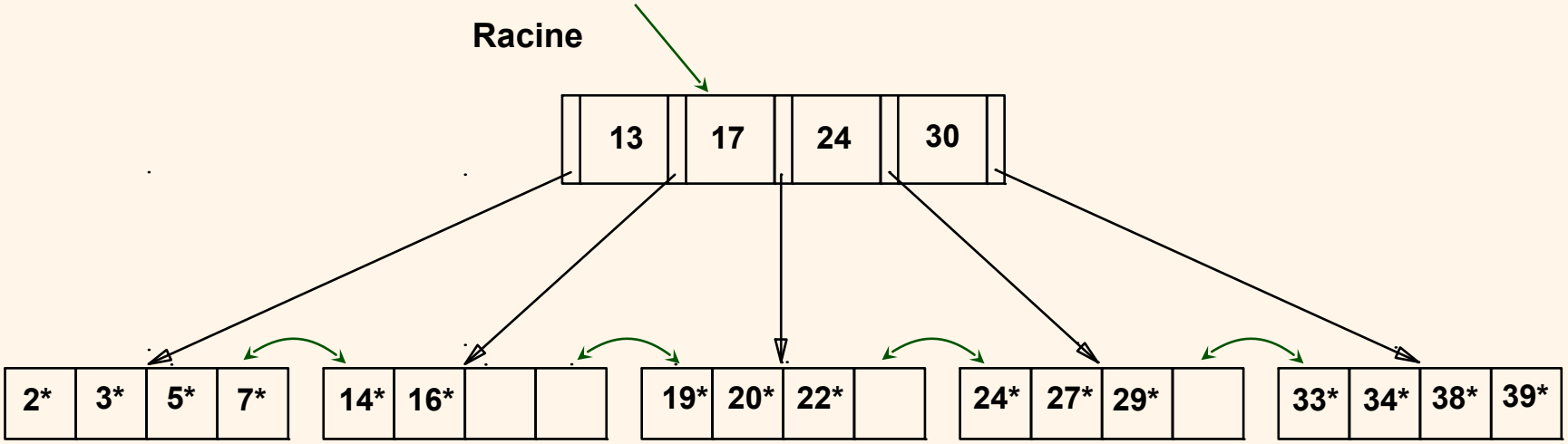
- Trouver la feuille N qui « devrait contenir l'entrée » par recherche à partir de la racine
- Rajouter l'entrée à la liste des entrées dans N (au bon endroit; garder les entrées triées)
  - Si on n'a pas dépassé la capacité de N (donc 2*d entrées), c'est fini!
  - Sinon, nous devons *diviser (split) le nœud N:*
    - Nous construisons un nouveau nœud N2 et le chaînons dans la liste des feuilles après N
    - « Clé du milieu » = la (d+1) ème clé dans N
    - Nous redistribuons les entrées entre N et N2: les d premières entrées restent dans N; les d+1 dernières vont dans N2
    - Nous insérons, dans le parent de N, après le pointeur qui dirigeait vers N, une nouvelle entrée d'index qui contient comme clé la clé du milieu et qui pointe vers de N2!
- *Les splits peuvent se propager vers le haut de l'arbre*: parce qu'un split cause une insertion dans le nœud parent, donc un potentiel besoin de split sur le nœud parent!

B+Tree: propagation des effets d'une insertion

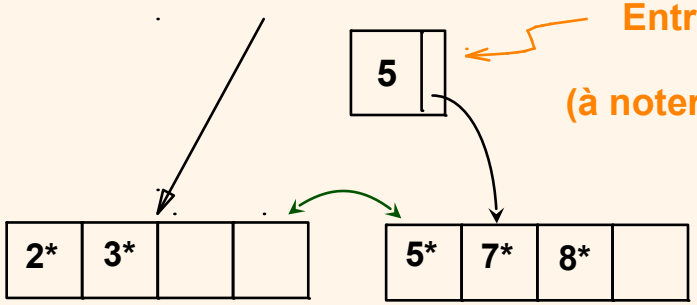


- *Les splits peuvent se propager vers le haut de l'arbre*: parce qu'un split cause une insertion dans le nœud parent, donc un potentiel besoin de split sur ce nœud!
  - Pour diviser un nœud intermédiaire: similaire au split d'une feuille mais pas besoin de chaîner et *N2 ne garde pas l'entrée du milieu*!
  - La raison de la différence avec les feuilles c'est que toutes les entrées de données doivent se trouver dans les feuilles
    - Nous sommes donc obligés de garder, dans le cas des feuilles, l'entrée du milieu !
- Les splits vont augmenter la taille de l'arbre:
  - Il devient plus "large" mais aussi plus "haut" si split de la racine = rajout d'un niveau!
- A noter: la manière de faire les splits nous garantit que les « taux de remplissage minimal et maximal» sont respectés!

Exemple insertion : arbre initial

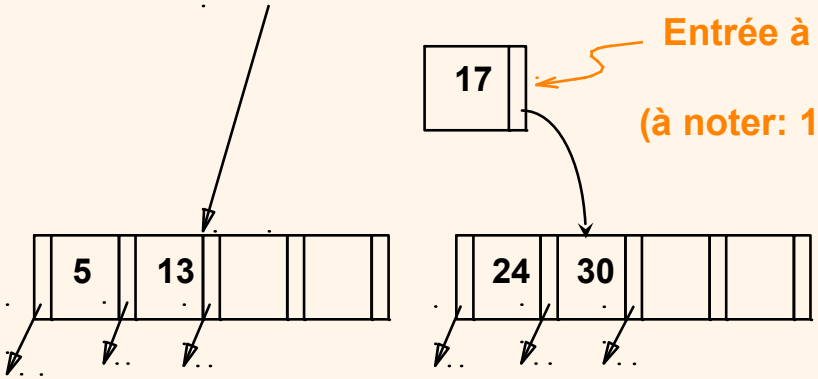


*Exemple insertion: insertion de l'entrée 8**



Entrée à insérer dans le nœud parent

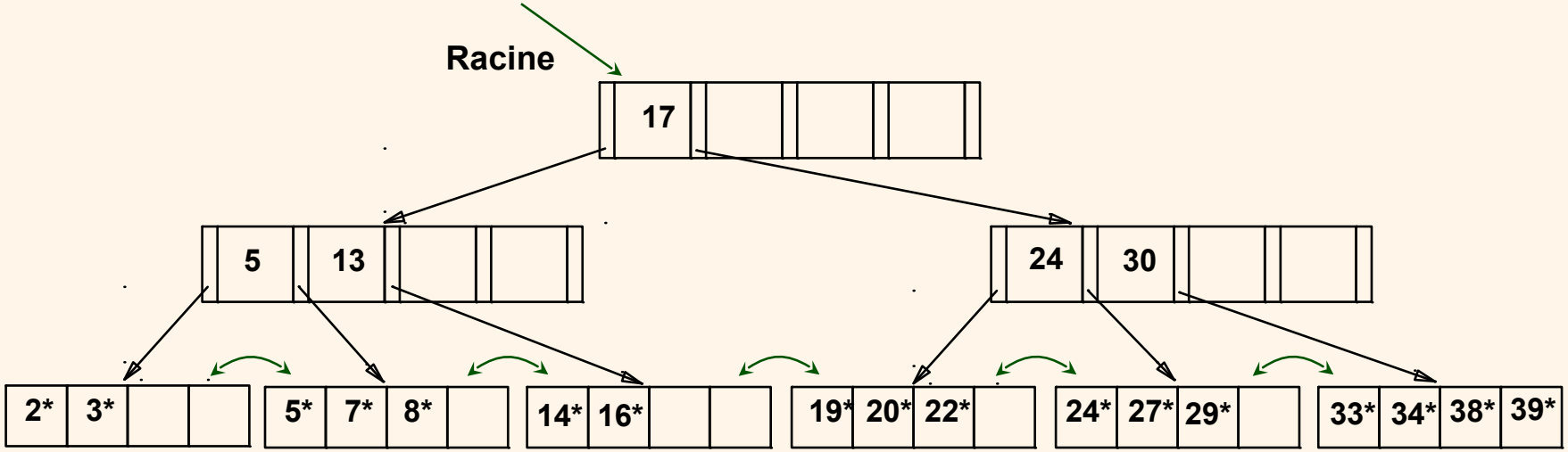
(à noter: 5 reste dans la nouvelle feuille)



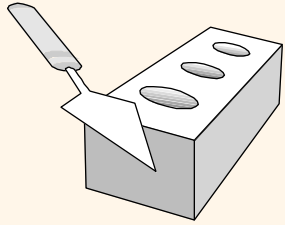
Entrée à insérer dans le nœud parent

(à noter: 17 ne reste plus dans le nouveau noeud)

Example insertion: état final de l'arbre



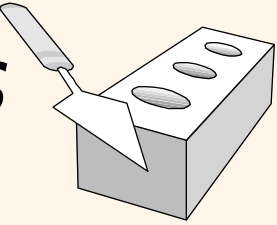
- A noter: la racine a été divisée, ce qui a rajouté un niveau à l'arbre!



B+Tree: suppression d'une entrée de données

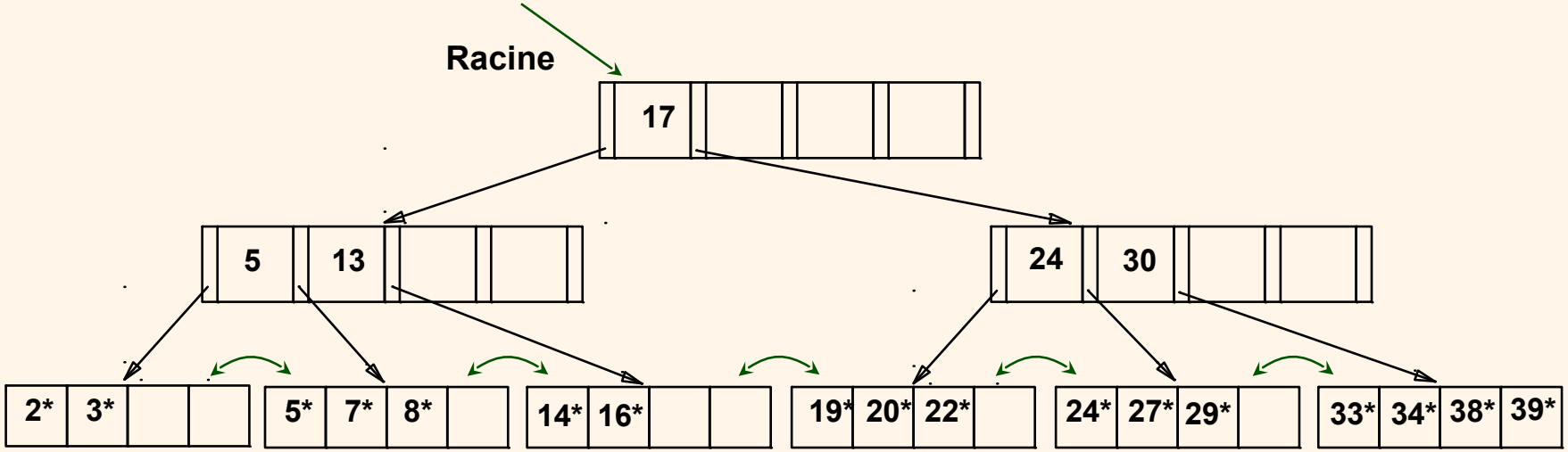
- Trouver la feuille N qui contient l'entrée par recherche depuis la racine
- Supprimer l'entrée
  - Si N reste "suffisamment rempli" (>=d), c'est fini!
  - Sinon*:*
    - Essayer de *redistribuer* les entrées entre N et un de ses « frères directs » (les nœuds "avant ou après N" qui ont le même parent!) ; dans ce cas, on actualise la clé « qui sépare N et son frère » avec « la clé du milieu » (même système que lors de l'insertion!)
    - Si cela n'est pas possible (trop peu d'entrées dans les frères), *unifier* (*merge*) N et un de des frères directs et supprimer l'entrée avec « la clé séparatrice » dans le parent
- *Comme pour l'insertion, les effets de la suppression peuvent se propager dans l'arbre, à cause des suppressions dans les nœuds parents*

B+Tree: propagation des effets d'une suppression

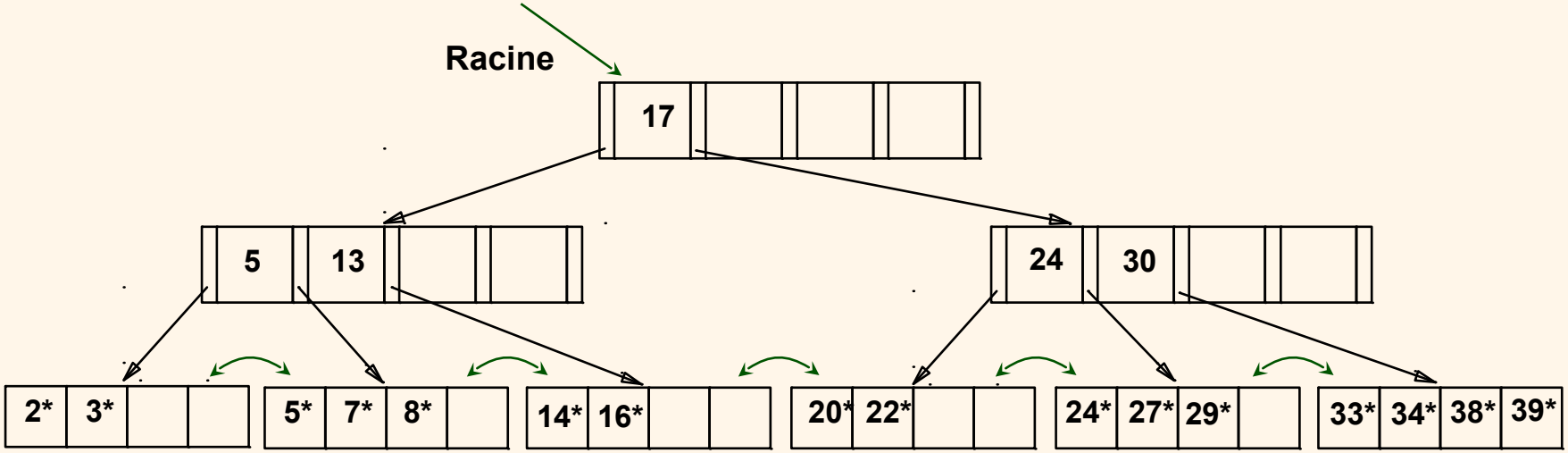


- *Les suppressions peuvent se propager vers le haut de l'arbre, car un merge provoque une suppression dans le parent*:
  - Dans le cas du merge ou de la redistribution de deux nœuds intermédiaires, la clé correspondant à l'entrée séparatrice est « redescendue »
    - Elle formera une nouvelle entrée d'index avec le « premier pointeur » du deuxième nœud
    - Dans le cas du merge, l'entrée séparatrice dans le parent est simplement effacée ensuite
    - Dans le cas de la redistribution, l'entrée séparatrice sera actualisée avec la clé du milieu ET on enlève l'entrée de cette clé ensuite dans le nœud fils concerné!
      - Même principe que pour les insertions !
- Les suppressions peuvent se propager jusqu'à la racine, ce qui peut faire diminuer la hauteur de l'arbre (si racine vide)
- A noter: la manière de faire les unifications et redistributions nous garantit que les « taux de remplissage minimal et maximal» sont respectés!

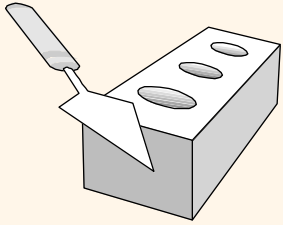
Example suppression: arbre initial



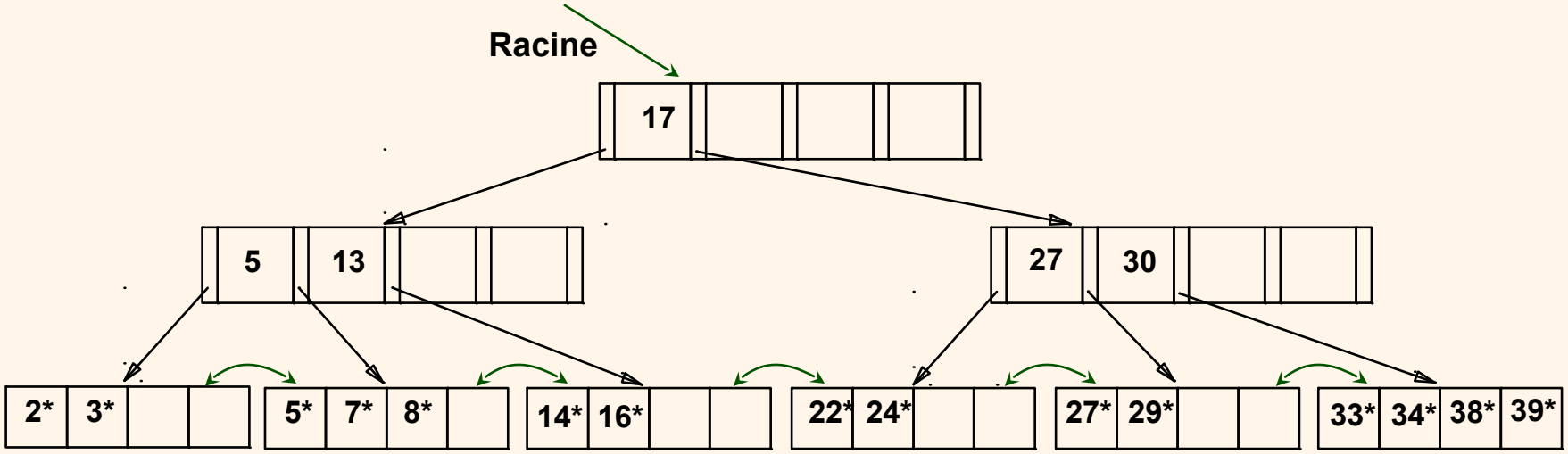
*Example suppression: suppression de 19**



- Facile car cela ne mène pas au sous-remplissage
- Mais si on veut ensuite supprimer 20* ?

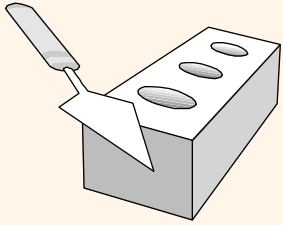


*Example suppression: suppression de 20**

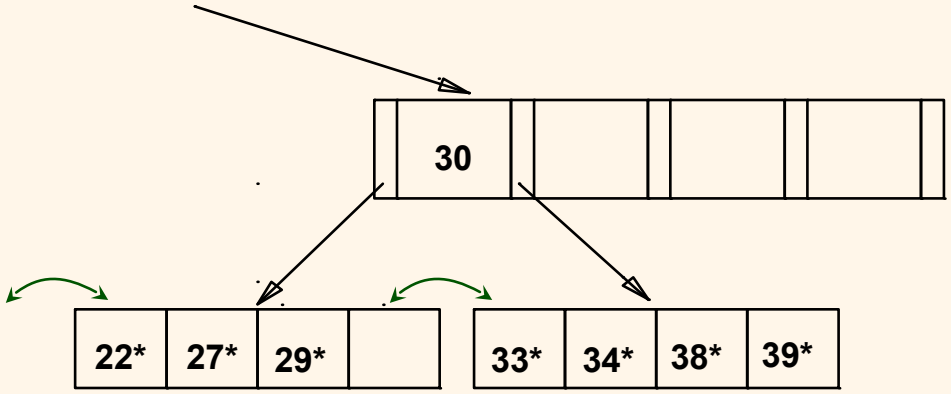


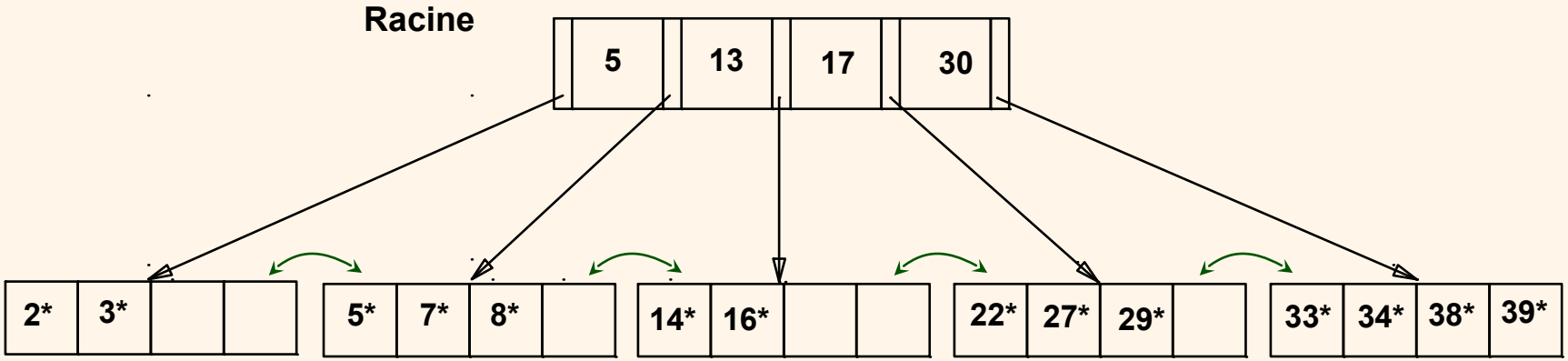
- Redistribution avec la feuille suivante: 24* « bouge » dans la feuille sous-remplie
- La clé « séparatrice » du parent est actualisée à 27!
- Et si ensuite on veut supprimer 24* ?

*Example suppression: suppression de 24**



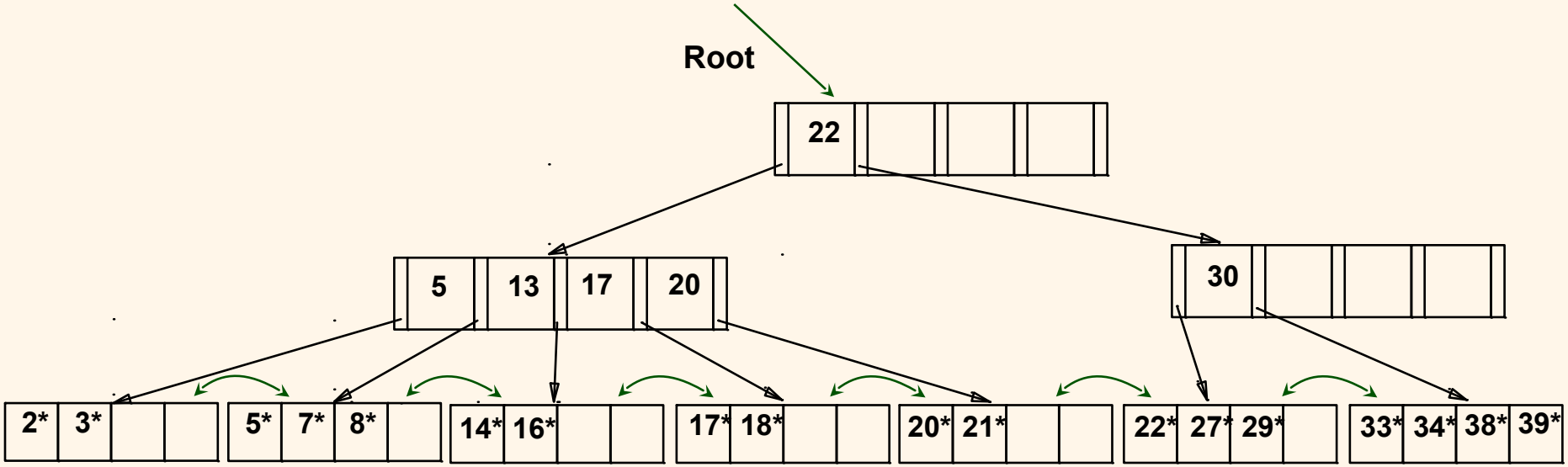
- Besoin de merge; l'entrée séparatrice (avec clé 27) est effacée dans le parent
- Ci-dessous: le merge se propage vers le haut ; on « redescend » 17 et on efface son entrée ds la racine → et le nœud « unifié » devient racine !





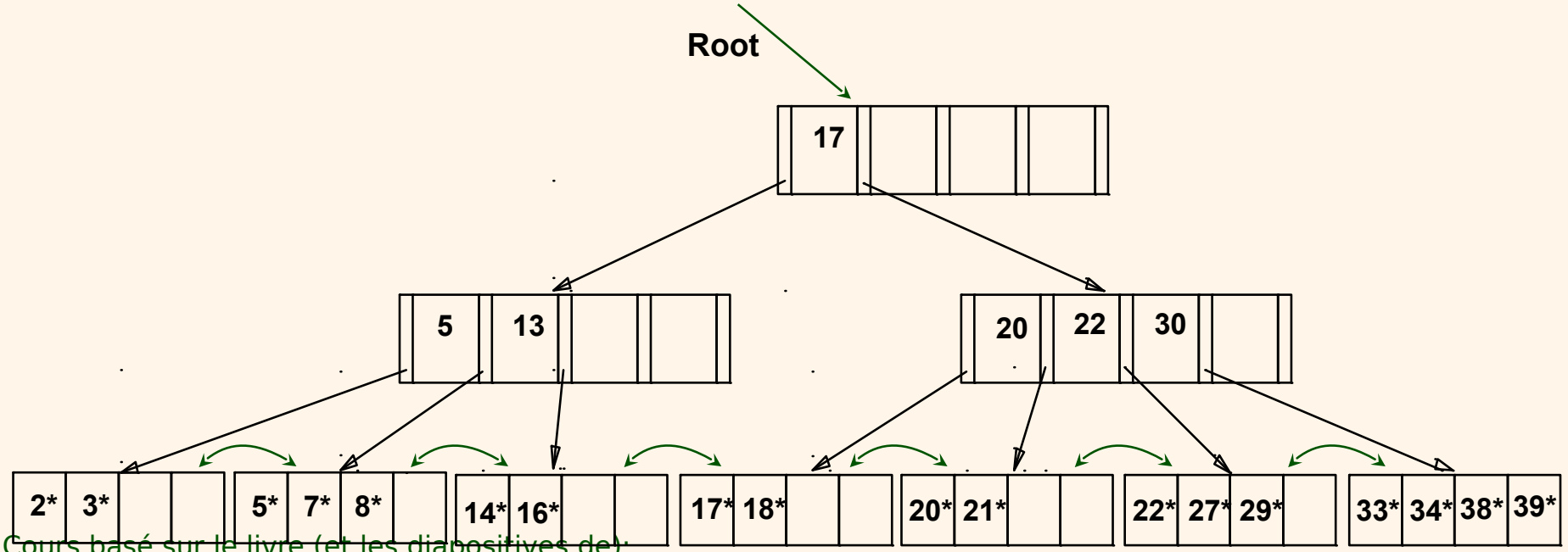
*Exemple 2 suppression: état pendant la suppression de 24**

- Ci-dessous: état de l'arbre pendant la suppression de 24*
  - Q: quel aurait pu être l'état de « début de suppression »?
- Ici, on peut faire de la redistribution entre les deux fils du nœud racine!
- Pour cela, on « redescend 22 »...

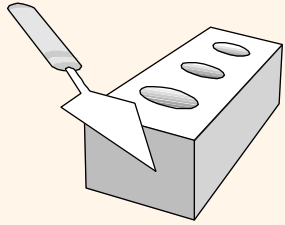


Exemple 2 suppression: après redistribution

- … puis on remonte 17 (il remplace donc 22 comme clé séparatrice!)
  - On aurait pu remonter 20 aussi – pourquoi?

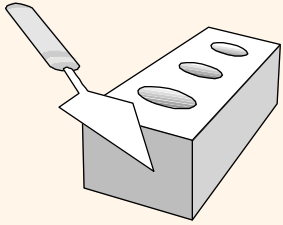


B+ Tree: chargement ”par lot” (bulk loading)

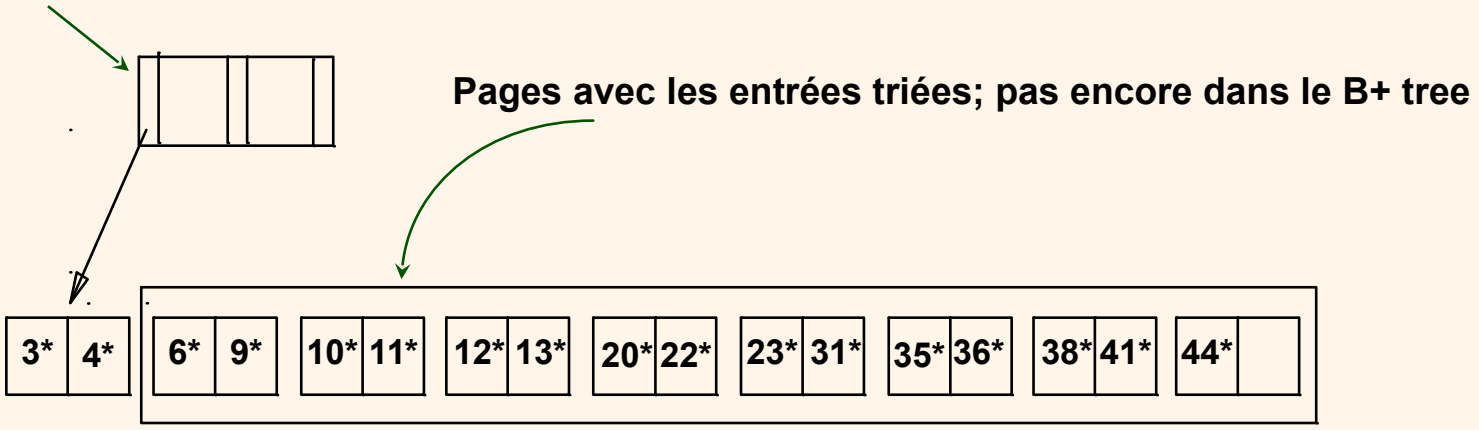


- Si on dispose déjà d’une collection de records et on veut construire un B+ Tree, les insertions “un à un” ne sont pas la meilleure manière de faire!
- *Le chargement « par lot » (bulk loading)* peut en effet être beaucoup plus efficace!
  - Feuilles allouées de manière séquentielle
  - Moins de I/O pendant la construction
  - Meilleur contrôle du « taux de remplissage » des pages
  - Avantages aussi pour les stratégies de vérouillage...

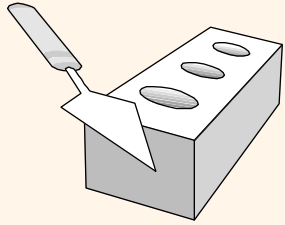
Bulk loading sur un exemple: initialisation



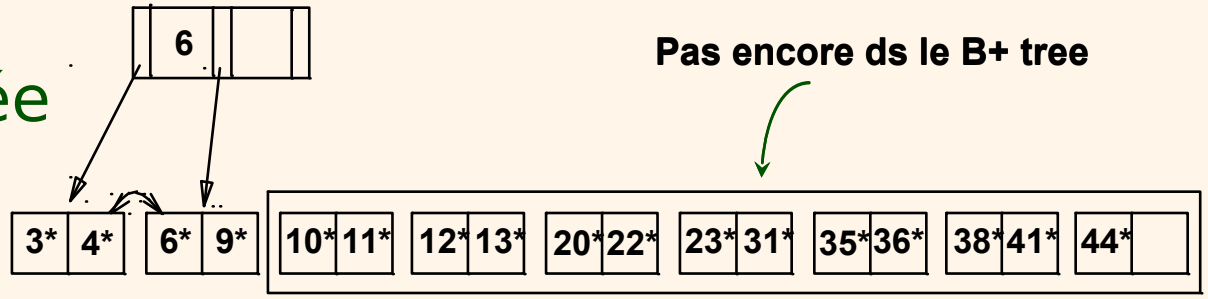
- *Initialisation*: Trier les entrées, les regrouper dans des feuilles (comme dans ISAM!), rajouter un nœud au dessus avec le premier pointeur pointant vers la première feuille
  - Avantage (comme dans ISAM) : allocation séquentielle des pages feuilles !

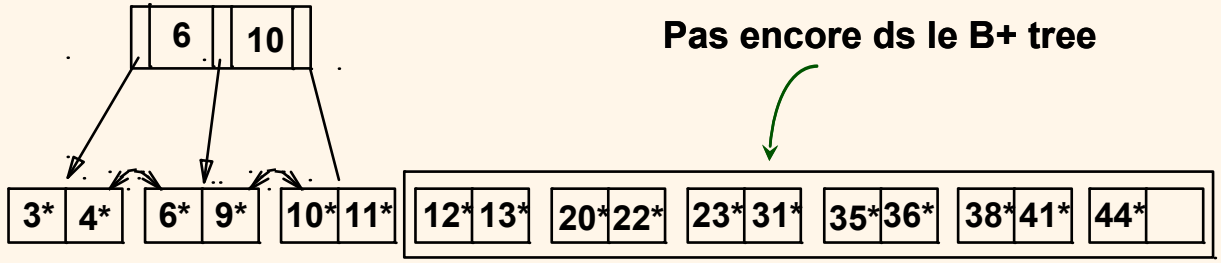


Bulk loading sur un exemple: prise en compte successive des pages feuilles

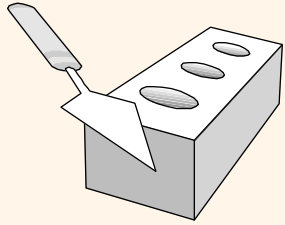


- A partir de la deuxième page, insérer une entrée d'index qui lui correspond (première clé sur la page), toujours dans le nœud "le plus à droite" dans la couche juste au-dessus des feuilles !

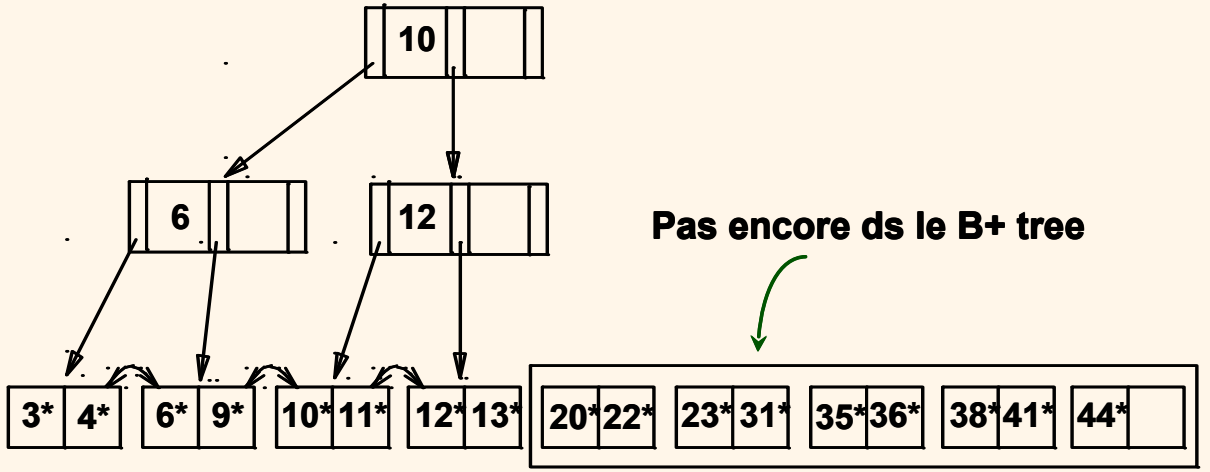


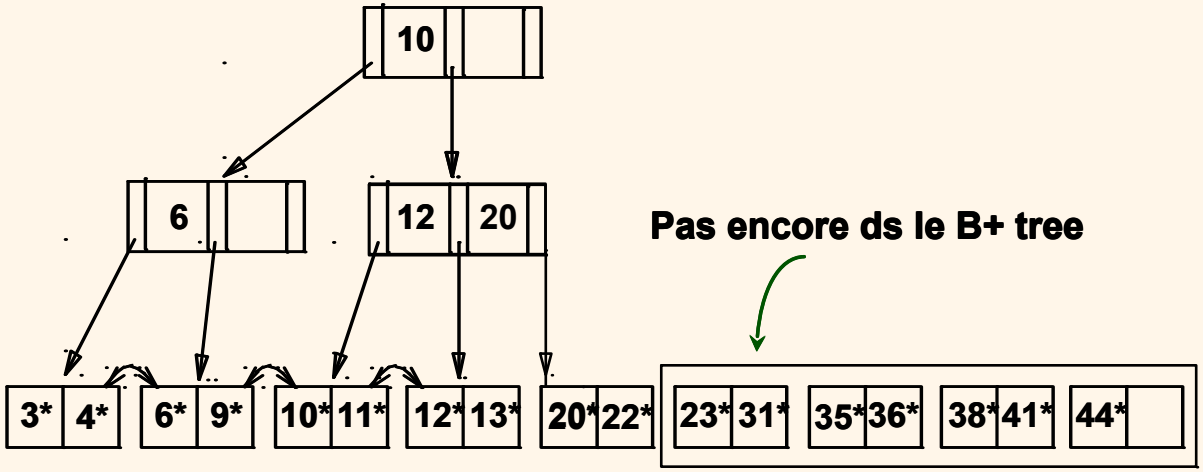


Bulk loading sur un exemple: prise en compte successive des pages feuilles

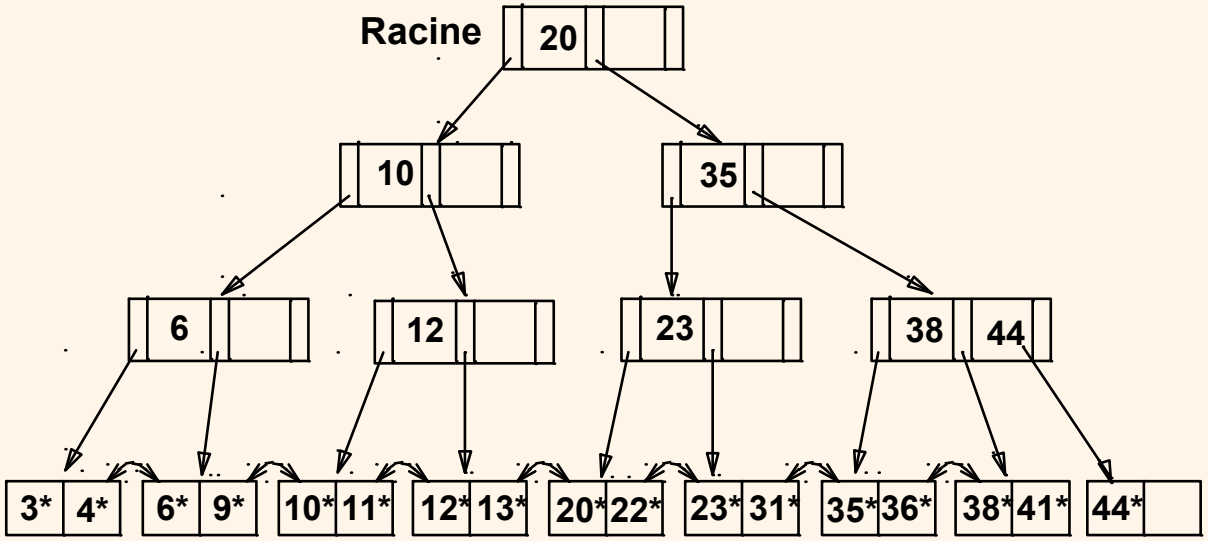


- A partir de la deuxième page, insérer une entrée d'index qui lui correspond (première clé sur la page), toujours dans le nœud "le plus à droite" dans la couche juste au-dessus des feuilles !
- Si nœud trop rempli, faire un split ! Cela peut mener à la création de niveaux supplémentaires dans l'arbre !

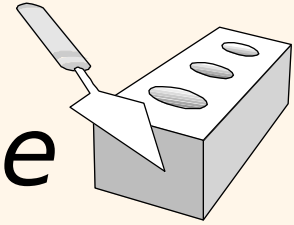




Bulk loading sur un exemple: état final de l'arbre



Synthèse / résumé index de type arbre



- Un index de type arbre est efficace pour les recherches / sélections de type intervalle ou égalité
  - À condition que ces recherches portent sur le(s) champ(s) qui forment la clé de recherche de l'index
- ISAM est une structure statique – adaptée aux situations où à la suite de la construction il y a peu d'insertions / suppressions
  - Les pages feuilles sont les seules modifiées, et il y a besoin de pages « overflow »
  - Les accumulations de pages overflow pénalisent les performances!
- B+ Tree est une structure dynamique – les insertions / suppressions sont rapides et ne pénalisent pas par la suite les performances car l'arbre reste équilibré
  - Pour ces raisons, très souvent préférable à ISAM; parmi les éléments les plus utilisés et optimisés d'un SGBD
  - Si besoin de construire un B+ Tree pour une collection de records déjà existante, le bulk loading est plus efficace que les insertions successives!