

Module BDR
Master d'informatique
spécialité SAR
Année 2011

site Web :

<http://www-bd.lip6.fr/ens/bdr2011/>

Stéphane Gancarski

Stephane.Gancarski@lip6.fr

Remerciements : Anne Doucet

Objectifs

Présenter les architectures des systèmes de gestion de bases de données (réparties) et les techniques permettant de les implémenter.

Techniques d'implémentation des SGBD relationnels et objet.
Architecture des bases de données réparties et du Web.
Conception, interrogation et manipulation de données réparties.

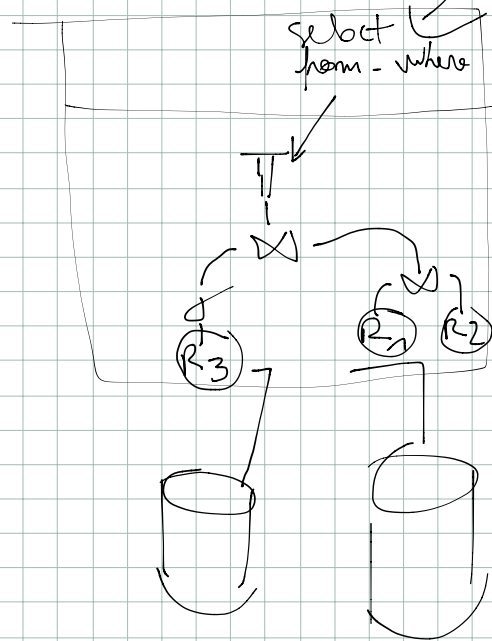
Mise en pratique : chaque séance comporte 2h de TME Oracle

SQL


select . . .

create . . .

update . . .



Plan

- Méthodes d'accès (hachage, index)
- Optimisation de requêtes *optimiser le nombre de pages que l'on va charger.*
- ODBMS et ORDBMS
- Techniques d'implémentation objet
- Bases de données réparties *conception*
- Interrogation de bases de données réparties 
- Reprise sur pannes
- Transactions réparties *quels sont les problèmes liés aux BDR*
- SGBD parallèles *planifier les données pour y accéder en parallèle*
- Gestion de données hétérogènes et réparties

Bibliographie

- G. Gardarin : *Bases de Données – objet et relationnel*, Eyrolles, 1999.
- H. Garcia-Molina, J.D.Ullman, J. Widom : *Database System Implementation*, Prentice Hall, 2000.
- M.T.Özsu, P. Valduriez : *Principles of Distributed Database Systems*, 2nd edition, Prentice Hall, 1999. 3è edition en avril
- R. Ramakrishnan : *Database Management Systems*, Mc-Graw Hill, 1997.
- S. Abiteboul, P. Buneman, D. Suciu : *Data on the Web : from relations to semistructured data and XML*, Morgan Kaufmann, 1999.

BDR
Master d'Informatique (SAR)
Niveau M1

Cours 1- Méthodes d'accès

Stéphane Gançarski
Stephane.Gancarski@lip6.fr

Plan

- Fonctions et structure des SGBD
- Structures physiques
 - Stockage des données
 - Organisation de fichiers et indexations
 - hachage
 - index
 - arbres B

Objectifs des SGBD

- Contrôle intégré des données

- Cohérence (transaction) et intégrité (CI)

- partage

- performances d'accès

- sécurité

contrôler les accès
aux données

tout se passe comme si
on était tout seul

autoriser l'accès à certains utilisateurs

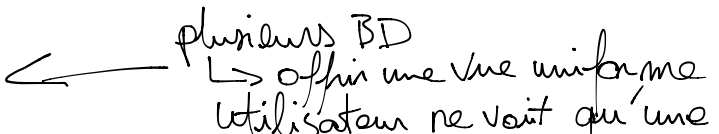
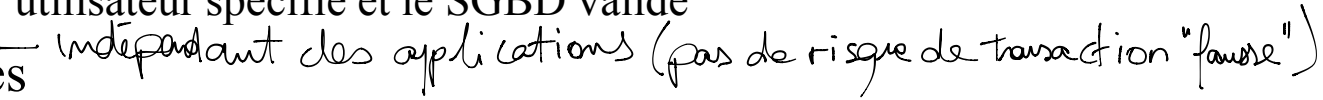
- Indépendance des données

- logique : cache les détails de l'organisation conceptuelle des données (définir des vues)

- physique : cache les détails du stockage physique des données (accès relationnel vs chemins d'accès physiques)

une requête qui définit une portion d'une BD

Fonctions

- Schéma intégré 
– vue uniforme des données, par ex. sous formes de relations (ou tables)
– Grâce à des adaptateurs par exemple
- Intégrité déclarative et cohérence
– $24000 \leq \text{Salaire} \leq 250000$
– l'utilisateur spécifie et le SGBD valide
- Vues 
– réorganisation de relations pour certaines classes d'utilisateurs
- Accès déclaratif
– avec un langage de requête (SQL), l'utilisateur spécifie ce qu'il veut obtenir et non comment l'obtenir.

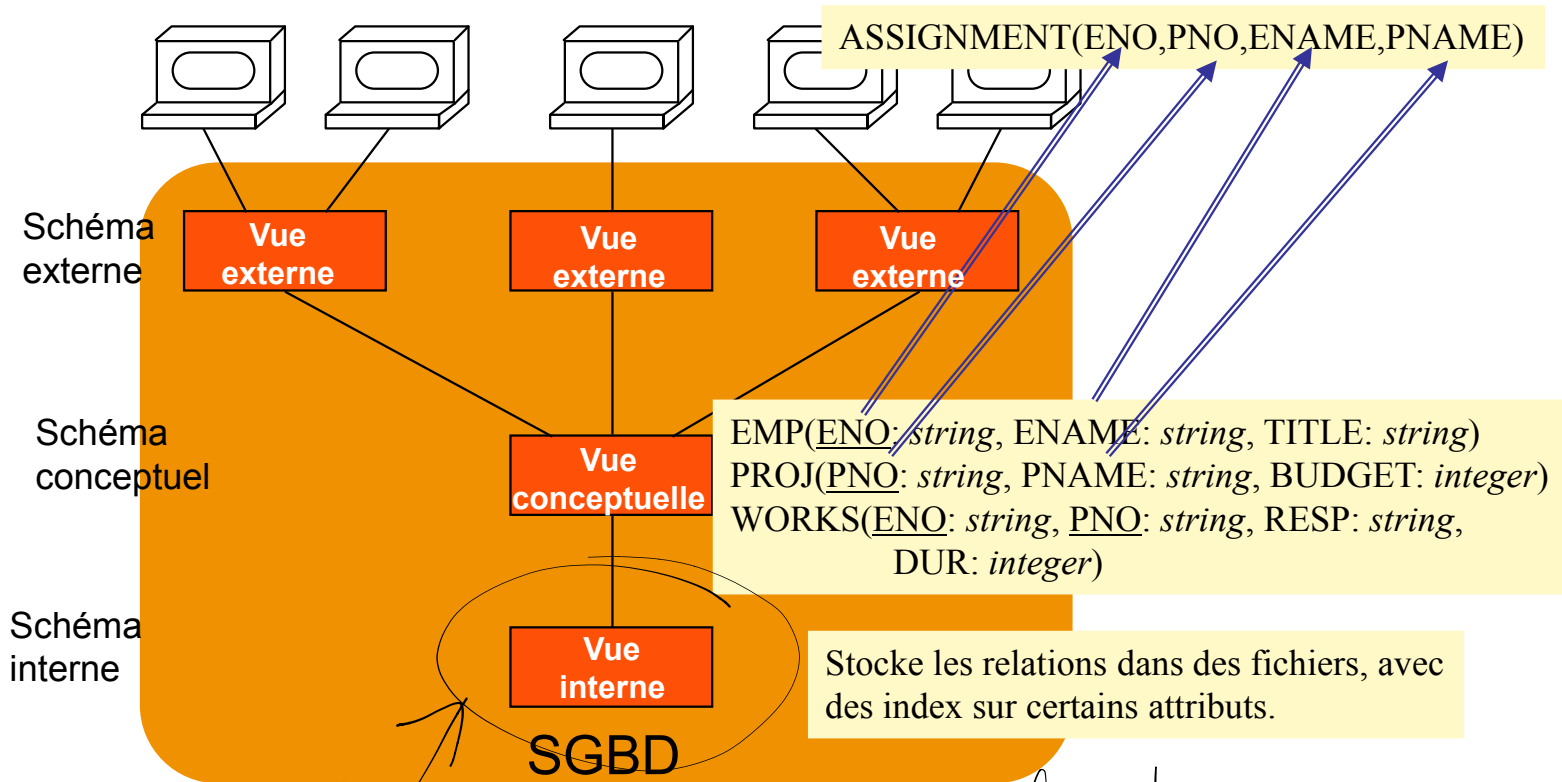
Fonctions

- Traitement et optimisation de requêtes
 - performances obtenues automatiquement. Evaluation de coût
- Transactions
 - exécution des requêtes par des unités atomiques
 - « indépendance » à la concurrence multi-utilisateurs et aux pannes

permet d'avoir une BD fiable pour une transaction
- Conception d'applications BD
 - conception visuelle des schémas de BD (E/A, OMT, UML, ...)
 - conception des traitements et des interfaces graphiques
 - **conception de la répartition** des données (répartition des traitements en principe transparente)

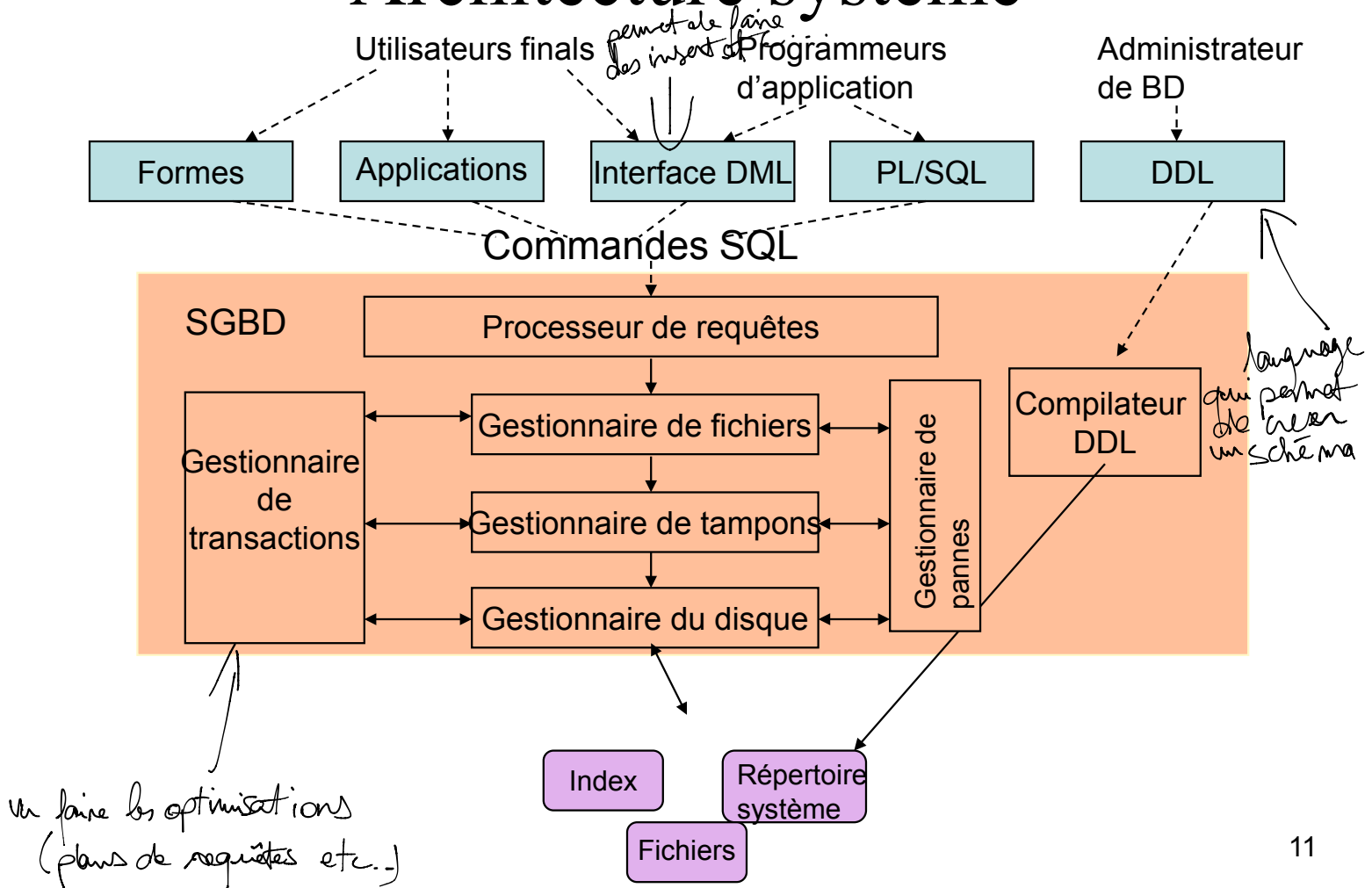
*SGBD: - génère tout ou partie d'un plan d'exéc
- calcule la fonction coût
- choisit la meilleure*
- Administration système
 - outils d'audit et de réglage (tuning)
 - visualisation des plans d'accès
 - Abordé en BDR, approfondi en ABDR M2

Architecture ANSI/SPARC



implémenter une base de relation

Architecture système



Stockage des données

- Les données sont stockées en mémoire secondaire (disques, bandes magnétiques, flash?). *de plus en plus les données sont externalisées*
- L'unité de stockage, et de transfert de données (disque-MC) est le bloc (coût incompressible). *pas de taille finie. → HS*
- Le *Nb accès disque.* coût d'une opération de la BD (en client/serveur) est principalement fonction du nombre d'accès disque nécessaires pour accéder aux données. Il dépend donc fortement de la façon dont les données sont organisées sur le disque.
- Le gérant du disque gère l'espace disque. L'unité de gestion est la page, qui correspond à un bloc. La taille de la page (4K ou 8K, voire plus) est un paramètre du SGBD.
- Le gestionnaire de tampons gère l'occupation de la mémoire centrale (plus efficace que l'OS).

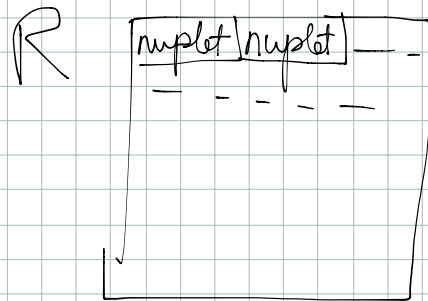
Organisation des données en fichiers

- Les données sont stockées dans des fichiers (ensemble d'enregistrements). Chaque enregistrement a un identificateur unique (ex. idpage+offset).
- Les fichiers sont stockés sur plusieurs pages. La page sur laquelle se trouve un enregistrement est déterminée par le gestionnaire de fichiers.
- La façon d'organiser les enregistrements dans un fichier a un impact important sur les performances. Elle dépend du type de requêtes. *Dépend aussi du type de mémoire (Flash très lente écriture)*
- Un SGBD offre en général plusieurs méthodes d'accès. Le choix de la meilleure méthode est du ressort de l'administrateur de la base (aussi choix de la clé primaire).

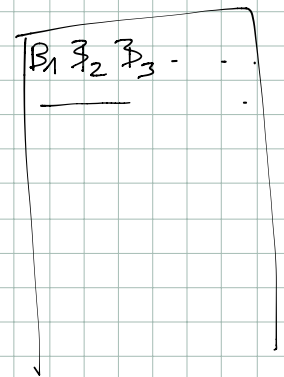
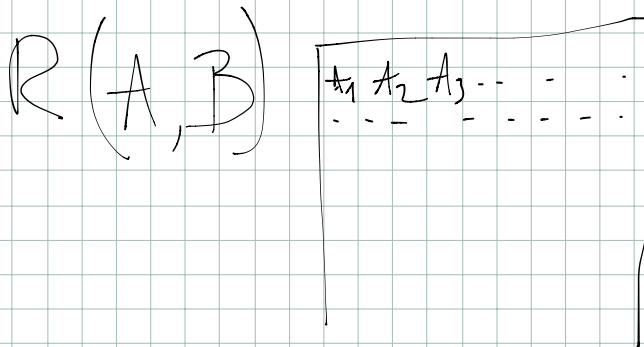
SGBD (INRIA) sur carte à puce (permet la sécurité des données)

↳ jamais de fuite des données
↳ Pb : C. à puce : mémoire Flash → lent écriture .

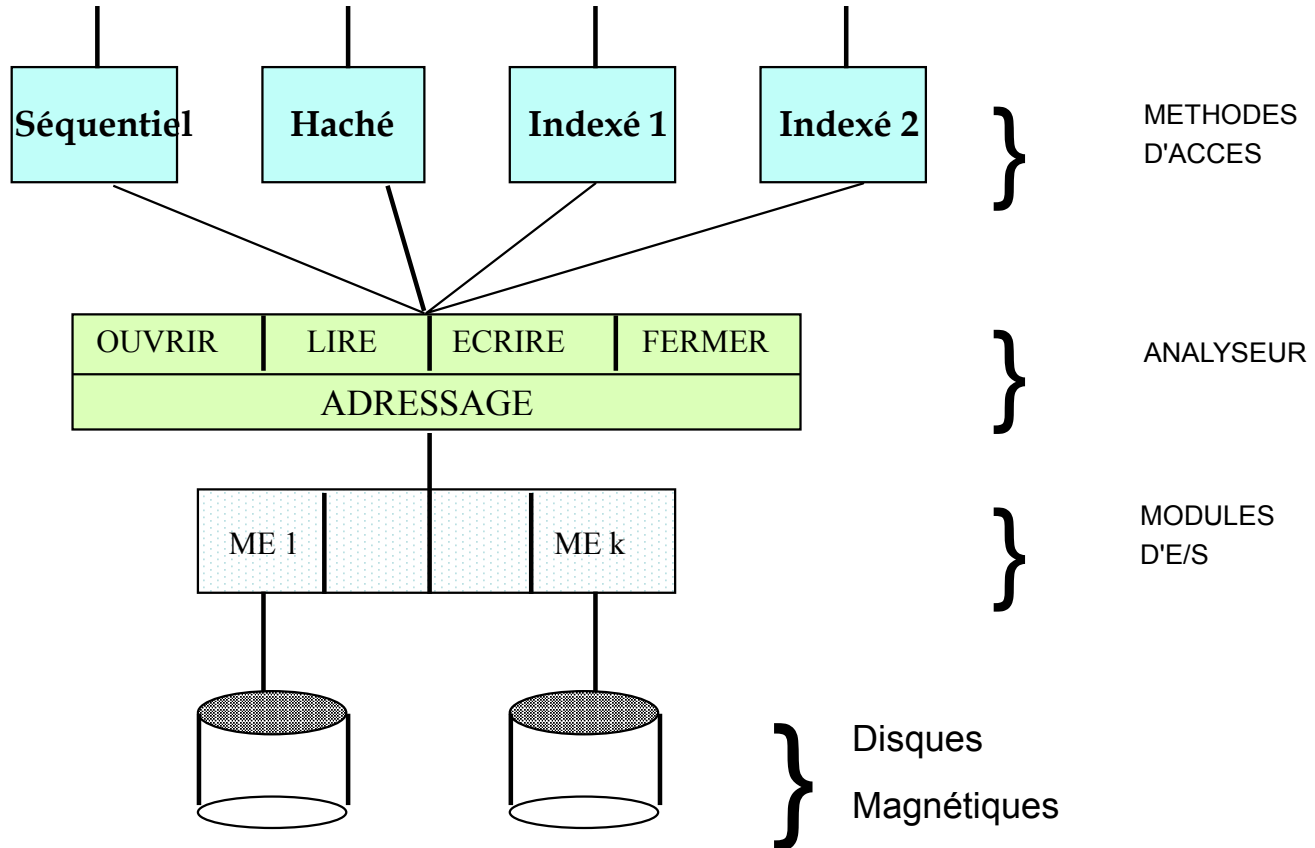
OLTP: beaucoup de requêtes assez courtes
↳ sur nuplets individuels.



OLAP: requêtes longues avec agrégats



Architecture d'un SGF



Opérations des SGBD / fichiers

- Parcourir tous les enregistrements du fichier (*scan*)
- Rechercher les enregistrements satisfaisant une condition.
 - Exact: *employés habitant Paris*
 - Intervalle : *employés dont le salaire > 5000.*
- Insérer un enregistrement
- Supprimer un enregistrement

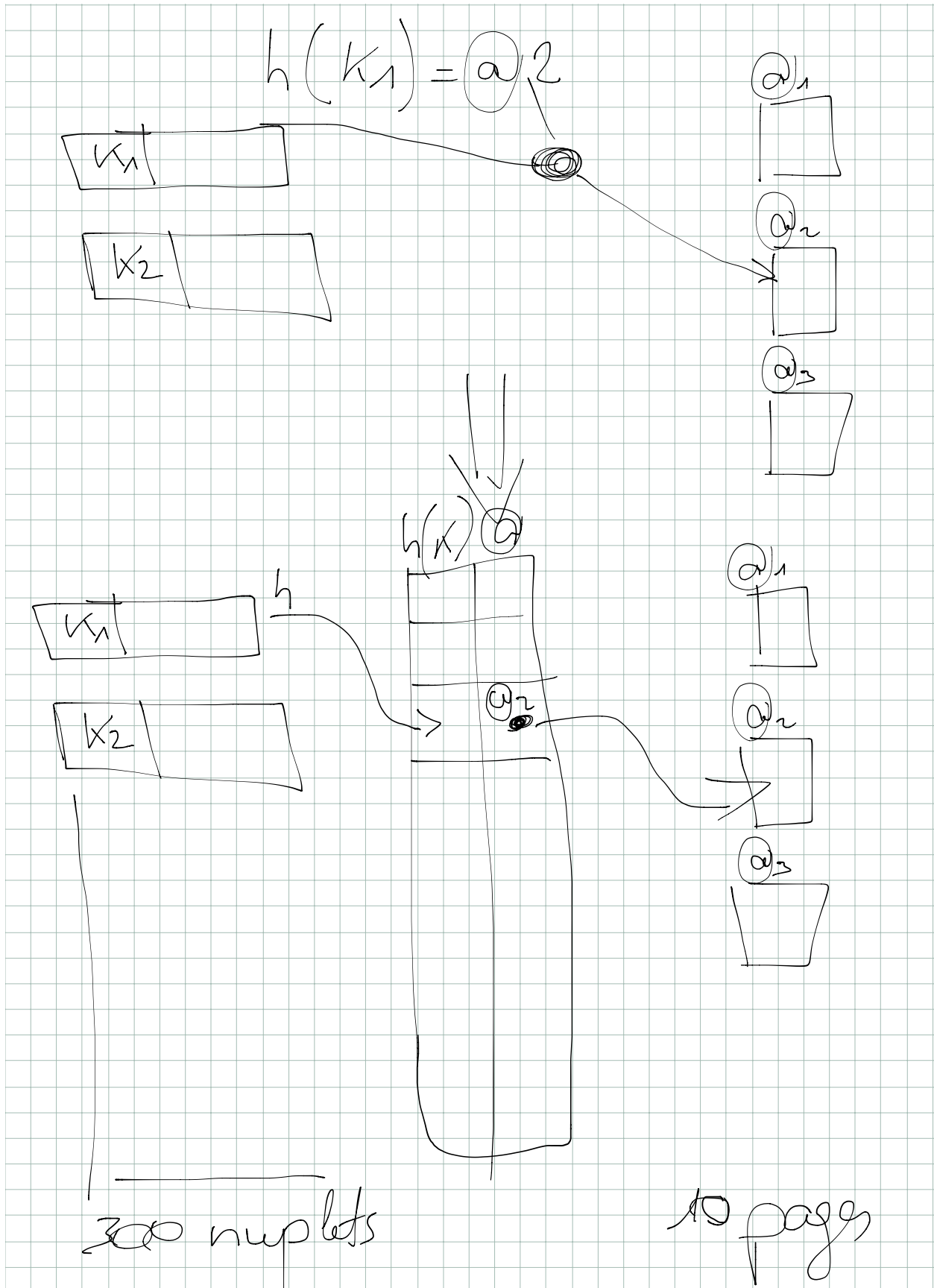
Organisation séquentielle.

Avantage : - très facile à maintenir en +1 aj

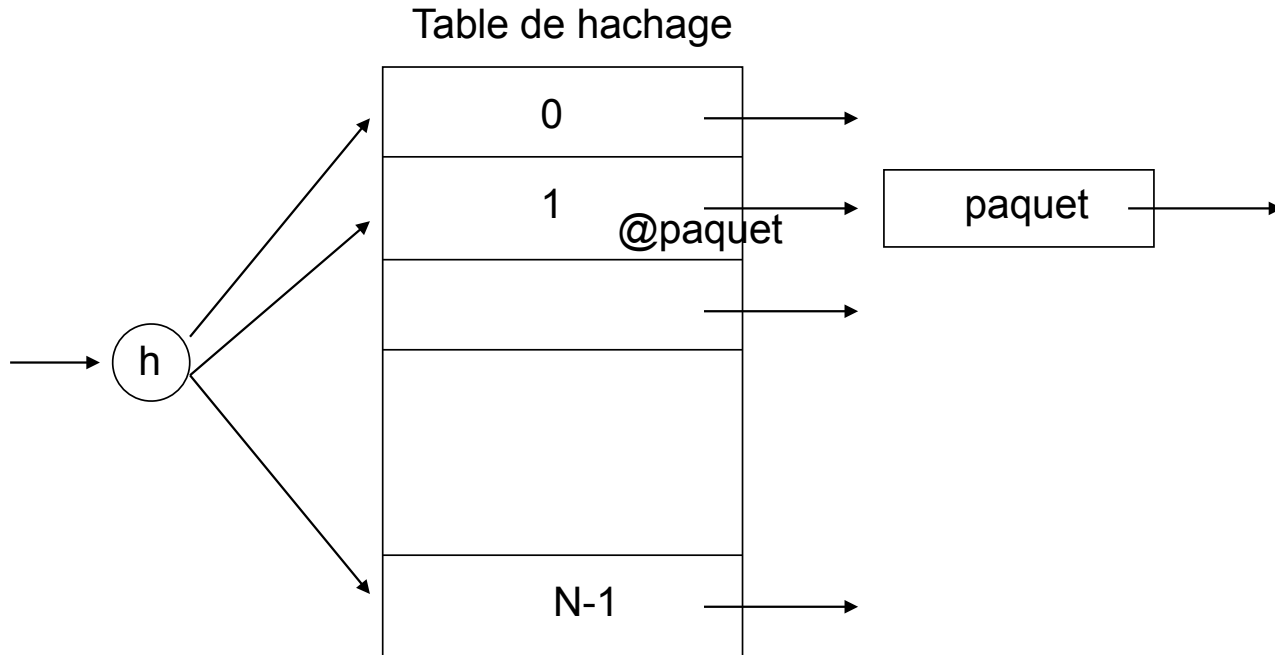
Organisations par Hachage

- Fichier haché statique (Static hashed file)
 - Fichier de taille fixe dans lequel les articles sont placés dans des paquets dont l'adresse est calculée à l'aide d'une fonction de hachage fixe appliquée à la clé.
 - On peut rajouter une indirection : table de hachage.
 - $H(k)$ donne la position d'une cellule dans la table.
 - Cellule contient adresse paquet
- Différents types de fonctions :
 - Conversion en nb entier
 - Modulo P
 - Pliage de la clé (combinaison de bits de la clé)
 - Peuvent être composées
- But :
 - Obtenir une distribution uniforme pour éviter les collisions (saturation)

Déjà : Obtenir une distribution uniforme pour éviter les collisions (saturation) 16



Hachage statique



Attention à l'uniformité du tableau de hachage !

Hachage statique

- Très efficace pour la recherche (condition d'égalité) : on retrouve le bon paquet en une lecture de bloc.
- Bonne méthode quand il y a peu d'évolution
- Choix de la fonction de hachage :
 - Mauvaise fonction de hachage ==> Saturation locale et perte de place
 - Solution : autoriser les débordements

Techniques de débordement

- l'adressage ouvert
 - place l'article qui devrait aller dans un paquet plein dans le premier paquet suivant ayant de la place libre; il faut alors mémoriser tous les paquets dans lequel un paquet plein a débordé.
- le chaînage
 - constitue un paquet logique par chaînage d'un paquet de débordement à un paquet plein.
- le rehachage
 - applique une deuxième fonction de hachage lorsqu'un paquet est plein, puis une troisième, etc..., toujours dans le même ordre.

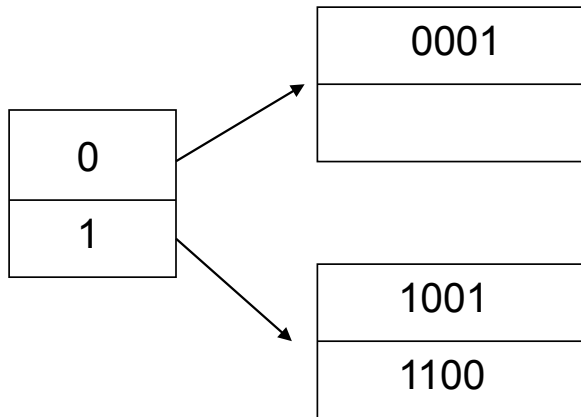
Le chaînage est la solution la plus souvent utilisée. Mais si trop de débordement, on perd tout l'intérêt du hachage (séquentiel)

Hachage dynamique

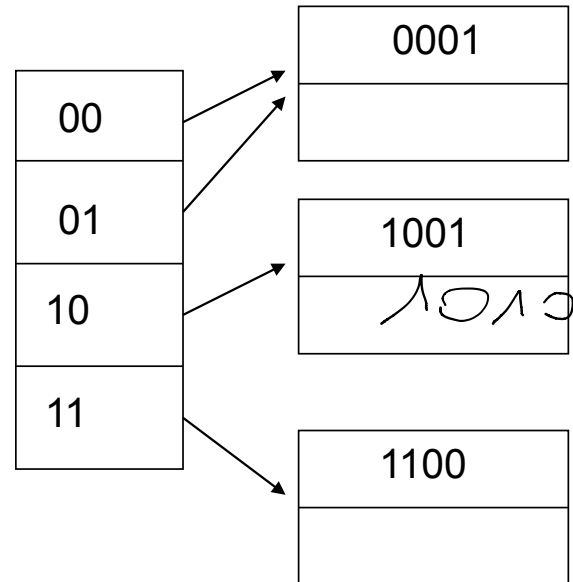
- Hachage dynamique :
 - techniques permettant de faire grandir progressivement un fichier haché saturé en distribuant les enregistrements dans de nouvelles régions allouées au fichier.
- Deux techniques principales
 - Hachage extensible
 - Hachage linéaire

Hachage extensible

- Ajout d'un niveau d'indirection vers les paquets (tableau de pointeurs), qui peut grandir : le répertoire
- Jamais de débordement



1010

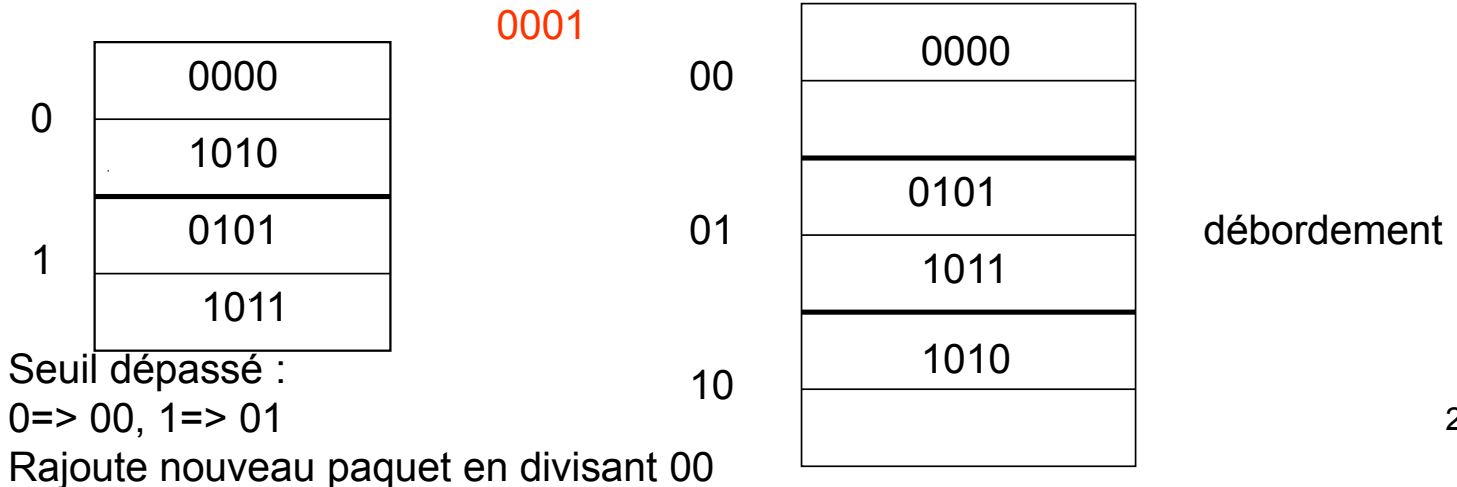


Hachage extensible (suite)

- Avantage : accès à un seul bloc (si le répertoire tient en mémoire)
- Inconvénient :
 - interruption de service lors du doublement du répertoire.
 - Peut ne plus tenir en mémoire.
 - Si peu de record par page, le répertoire peut être inutilement gros

Hachage linéaire

- le nombre moyen d'enregistrements par paquet ne dépasse pas un certain seuil (ex. 80%). Nouveaux paquets au fur et à mesure besoins, un par un.
 - Débordements quand le seuil n'est pas atteint et que le paquet est plein.
 - Pas besoin de répertoire. Mais il faut une suite de fonction de hachage qui double le nombre de paquets à chaque fois :
 - Ex : N paquets initialement, $h(x)$ fonction de hachage initiale
 - $h_i(x) = h(x) \bmod (2^i N)$
- Si on cherche 1011, on sait que le paquet 11 (=3) n'existe pas car > nombre de paquets. Donc paquet 01**



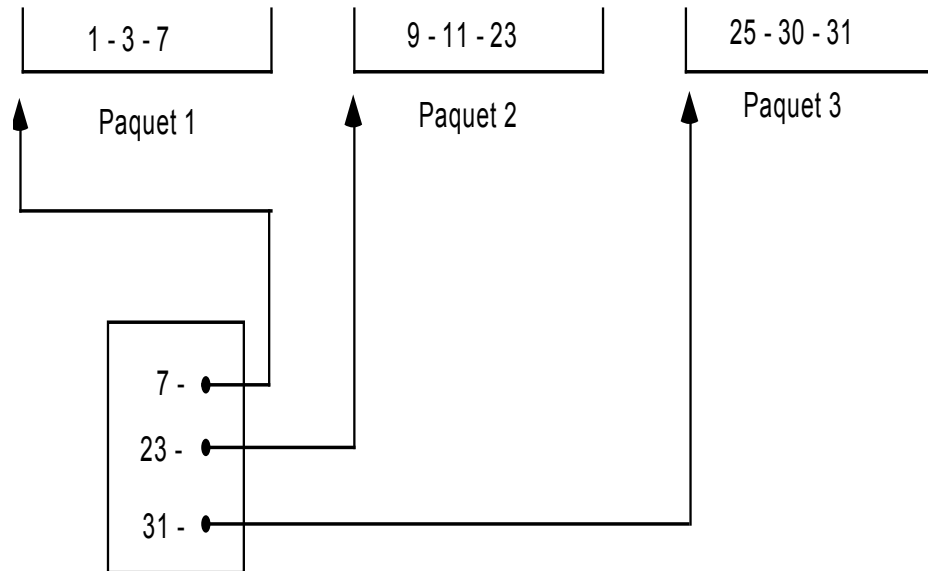
Organisations Indexées

- Objectifs :
 - Accès rapide à partir d'une clé
 - Accès séquentiel trié ou non
- Moyens :
 - Utilisation de tables permettant la recherche de l'adresse de l'enregistrement à partir de la clé
- Index
 - Table (ou plusieurs tables) permettant d'associer à une clé d'enregistrement l'adresse relative de cet enregistrement.

Différents Types d'Index

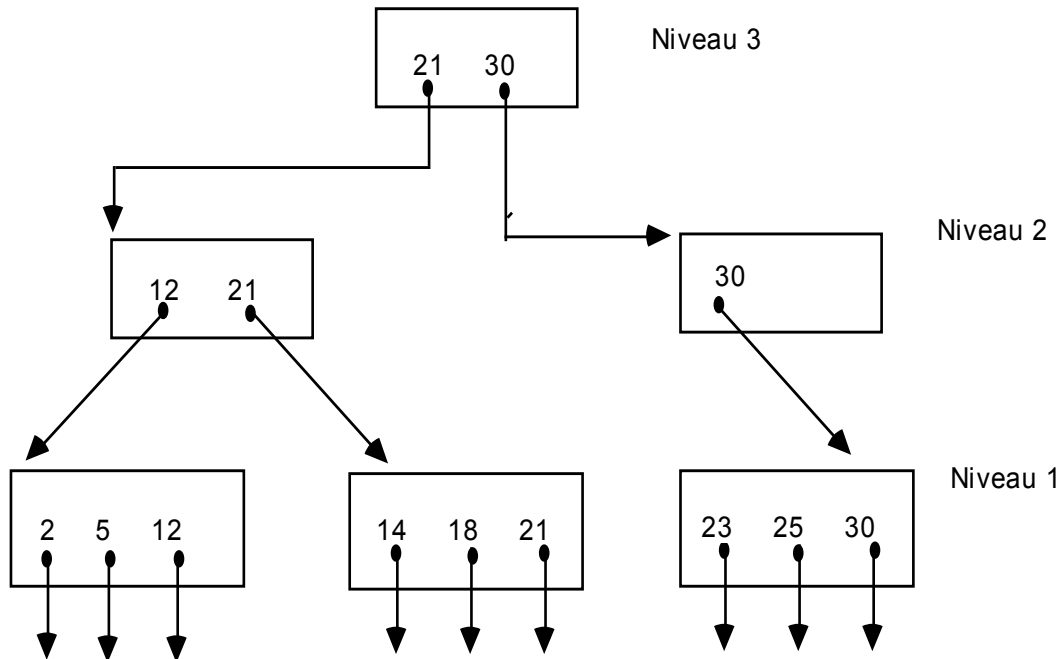
- Un index contenant toutes les clés est dense
- Un index non dense est possible si le fichier est trié
 - Il contient alors la plus grande clé de chaque bloc avec l'adresse relative du bloc.
- Il est possible de construire des index hiérarchisés
 - Chaque index possède alors un index qui permet d'accélérer la recherche.
 - Il est ainsi possible de gérer efficacement de gros fichiers.

Exemple d'index non dense



Exemple d'index hiérarchisé

Index à n niveaux, le niveau k étant un index trié divisé en paquets, possédant lui-même un index de niveau $k+1$, la clé de chaque entrée de ce dernier étant la plus grande du paquet.



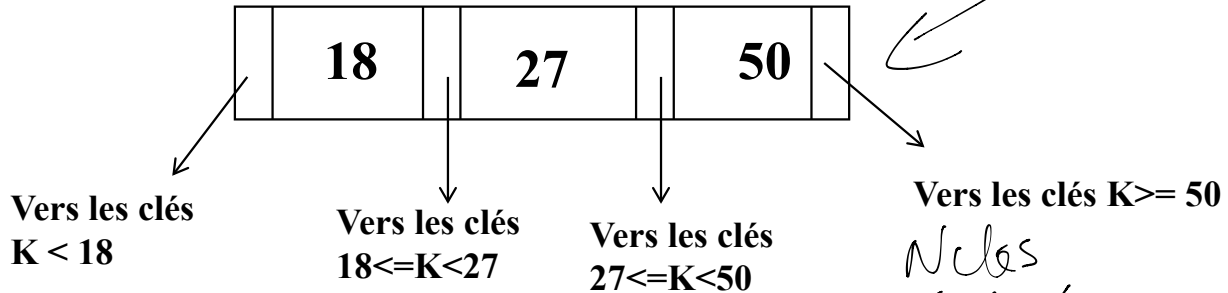
Arbre-B+

- Les arbres-B+ sont des indexs hiérarchiques. Ils fournissent des outils de base pour construire des index équilibrés.
- Ils permettent d'améliorer l'efficacité des recherches
 - En maintenant automatiquement le bon niveau d'index tout en gardant un arbre équilibré (tous les chemins de la racine aux feuilles ont même longueur) :
 - important pour estimer le coût
 - En gérant la place dans les feuilles de façon à ce que chaque ~~feuille~~ ^{noeud (sauf racine)} soit au moins à moitié plein.
 - Garder un index suffisamment compact (tenir en mémoire par ex.)

$h \rightarrow \text{hauteur}$

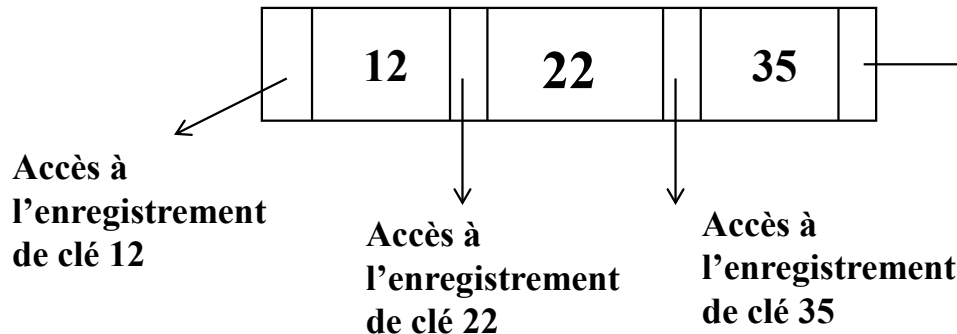
Arbre B+

- Les nœuds internes servent à orienter la recherche



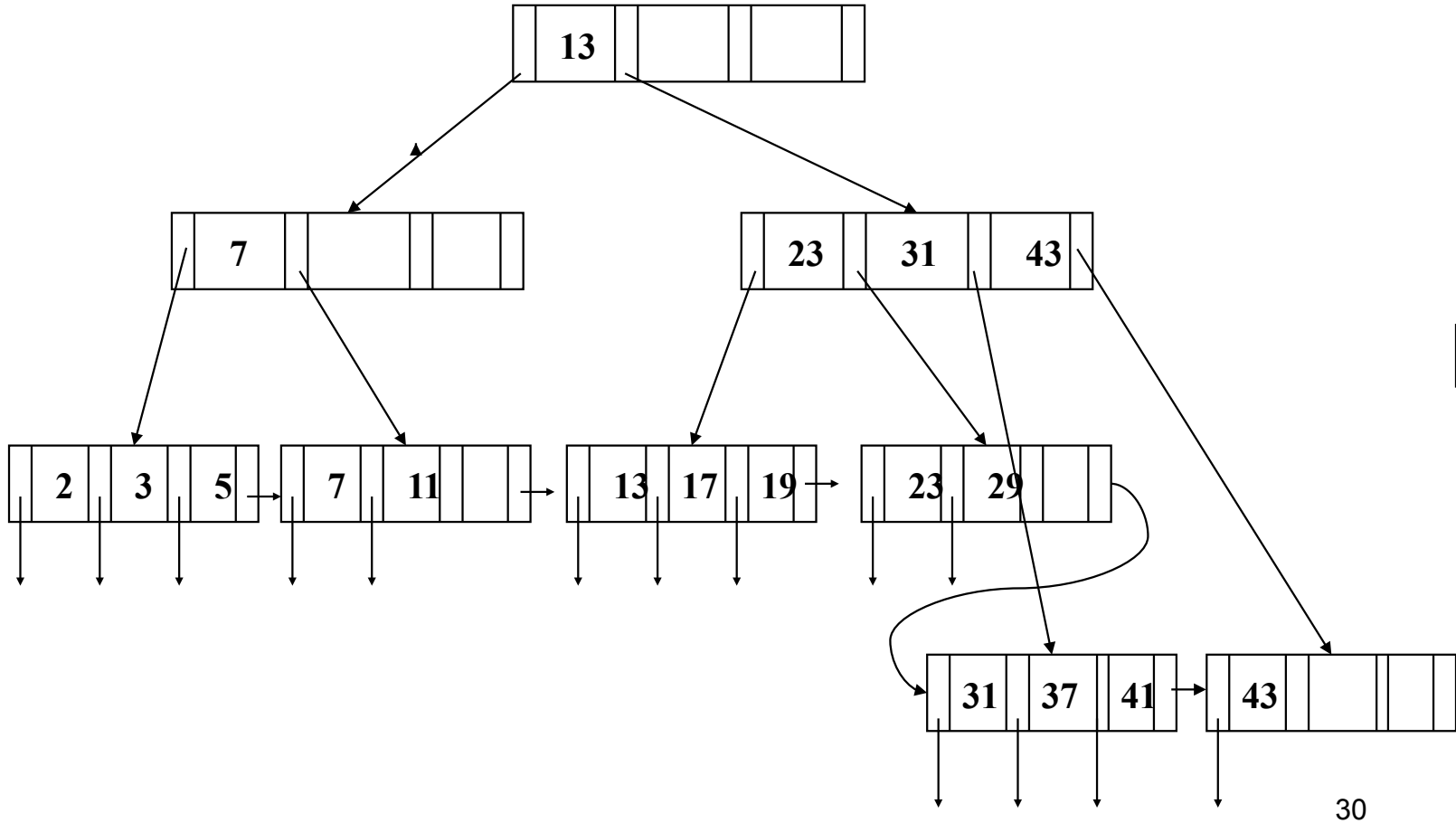
*4 pointeurs
3 clés*

- Les feuilles donnent accès aux enregistrements



*Nœuds
K clé/page
→ N/K feuilles
 N/K^2 nœud interne
Accès à la feuille suivante
 $N/K^h = 1$
(racine)*

Example



Arbre- B+

- Chaque bloc contient n valeurs de clés et $n+1$ pointeurs.
- Un arbre-B+ est d'ordre d si $d \leq n < 2d$.
 - d est la mesure de la capacité d'un nœud de l'arbre.
- En général, un arbre B+ a rarement plus de 4 niveaux.

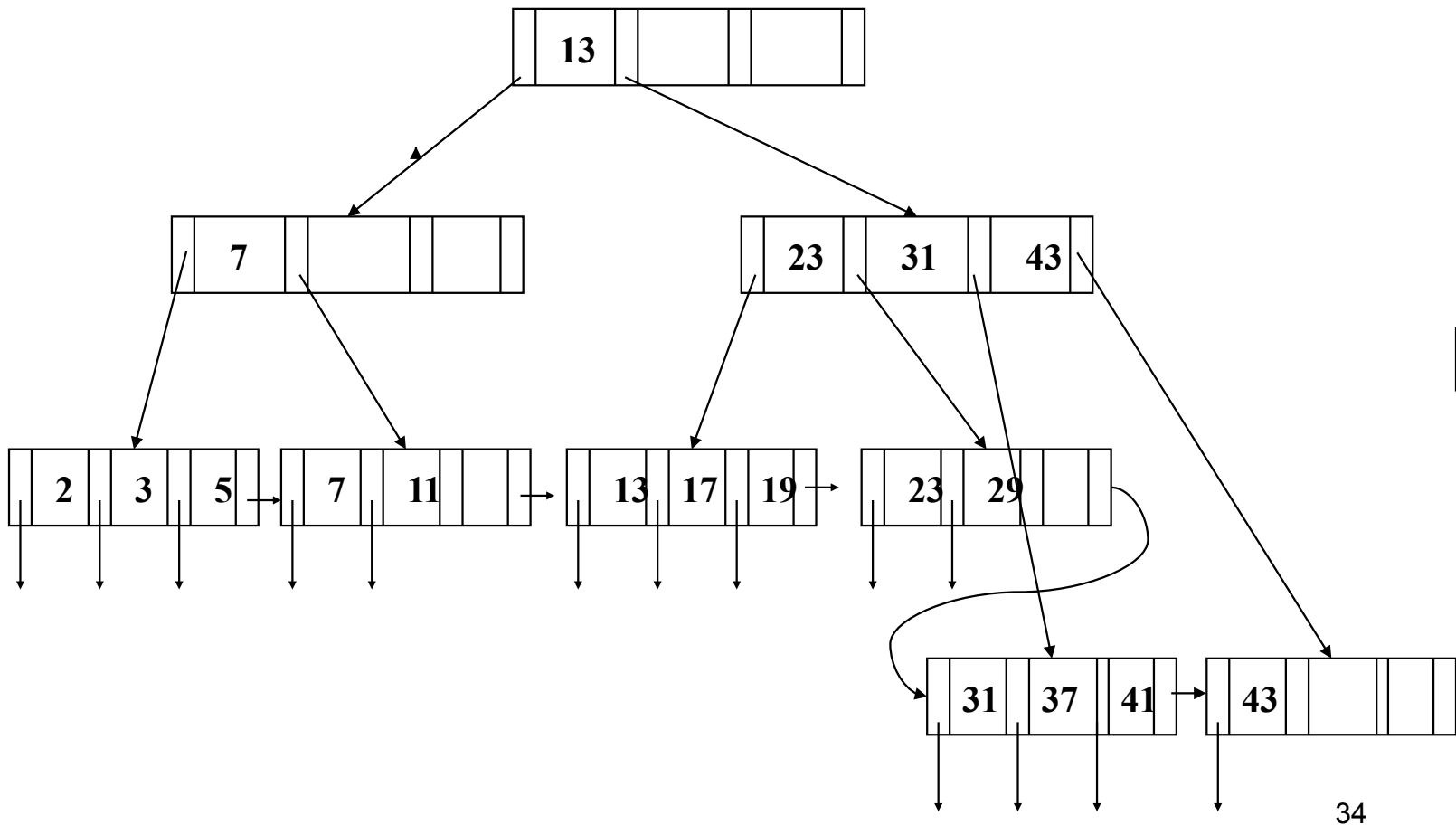
Insertion

- Rechercher la feuille où insérer la nouvelle valeur.
- Insérer la valeur dans la feuille en ordre de tri s'il y a de la place.
- Si la feuille est pleine, il y a débordement. Il faut créer un nouveau nœud :
 - Insérer les $(n+1)/2$ premières valeurs dans le nœud original, et les autres dans le nouveau nœud (à droite du premier).
 - La plus petite valeur du nouveau nœud doit être insérée dans le nœud parent, ainsi qu'un pointeur vers ce nouveau nœud.
 - Remarque : les deux feuilles ont bien un nombre correct de valeurs (elles sont au moins à moitié pleines)

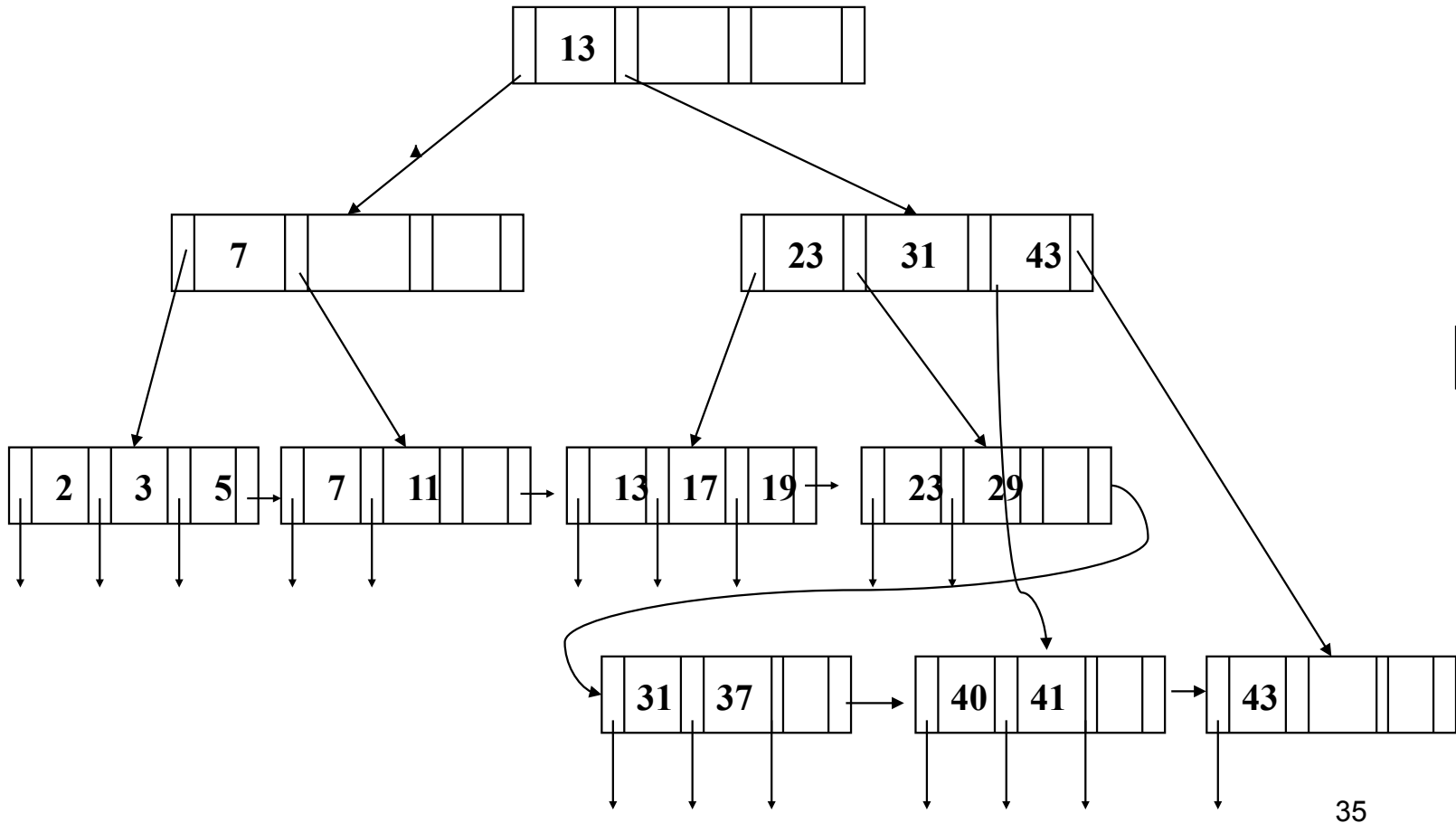
Insertion (cont.)

- S'il y a débordement dans le parent, il faut créer un nouveau nœud frère, à droite du premier:
 - Les $(n+1)/2$ premières paires(clé-pointeur) restent dans premier nœud(N), les autres se trouvent dans le nouveau nœud (M).
 - Chaque nœud devant contenir un pointeur de plus que de clé, il y a toujours une clé ne trouvant pas de place. Il s'agit de la clé indiquant la plus petite valeur accessible via le premier fils de M. Cette clé est utilisée par le parent de N et M pour la recherche entre ces deux nœuds.
- Remarque : les divisions peuvent se propager jusqu'à la racine et créer un nouveau niveau pour l'arbre.

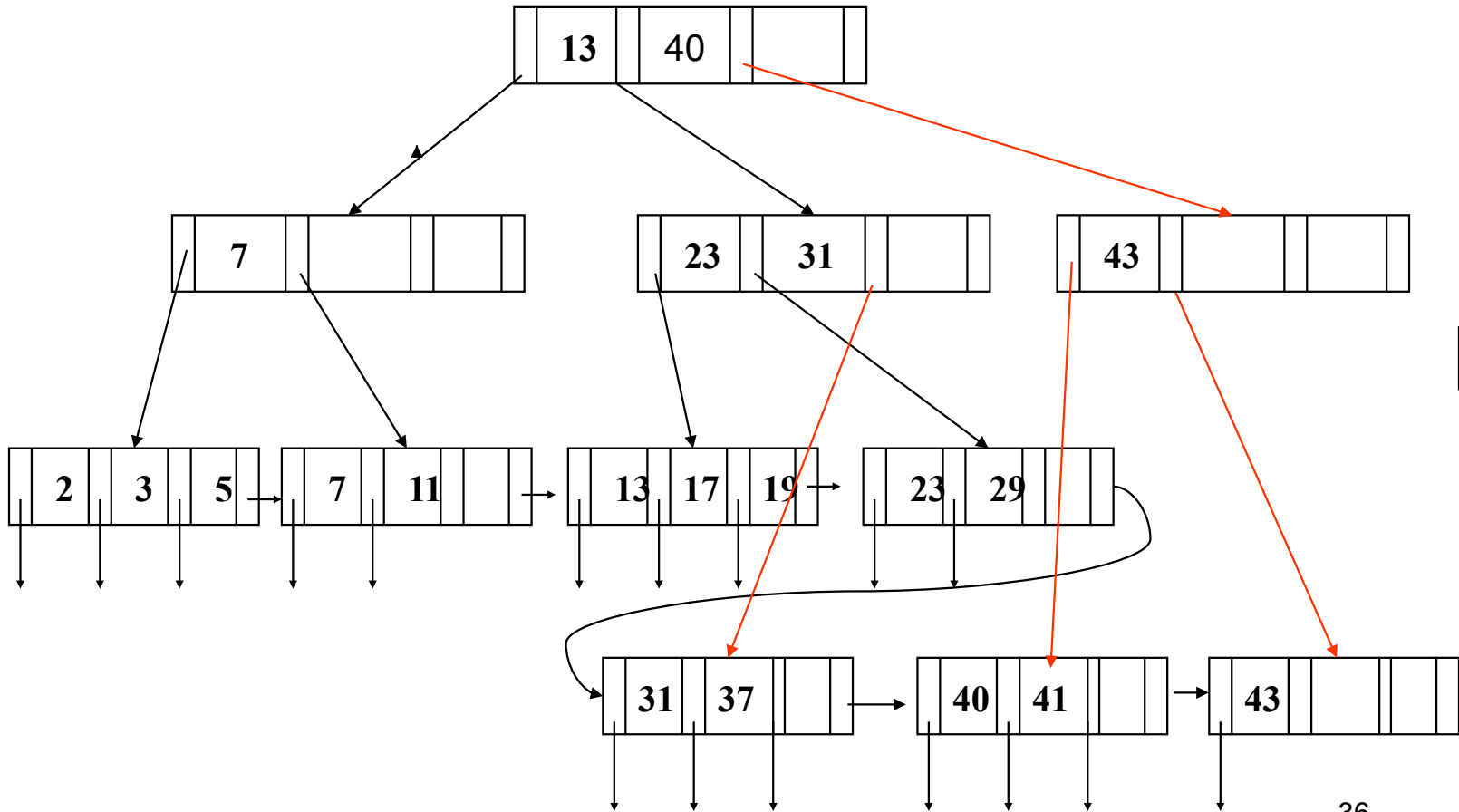
Example



Exemple (insertion de la clé 40) éclatement d'une feuille



Exemple (insertion de la clé 40) éclatement d'un noeud



Suppression

- Supprimer la valeur (et le pointeur vers l'enregistrement) de la feuille où elle se trouve.
- Si la feuille est encore suffisamment pleine, il n'y a rien d'autre à faire. Sinon, il faut
 - redistribuer les valeurs avec une feuille ayant le même parent, afin que toutes les feuilles aient le nombre minimum de valeur requis.
 - répercuter ces modifications au niveau des parents.
 - Si la redistribution est impossible, il faut fusionner des feuilles, et ajuster les valeurs au niveau des parents.
 - Si le parent n'est pas suffisamment plein, appliquer récursivement l'algorithme de suppression.
- Remarque : la propagation peut entraîner la suppression d'un niveau.

Avantages et Inconvénients

- Avantages des organisations indexées par arbre-b ($b+$) :
 - Régularité = pas de réorganisation du fichier nécessaires après de multiples mises à jour.
 - Lecture séquentielle rapide: possibilité de séquentiel physique et logique (trié)
 - Accès rapide en 3 E/S au plus pour des fichiers de 1 M d'articles
- Inconvénients :
 - Les suppressions génèrent des trous difficiles à récupérer
 - Dans le cas d'index non plaçant, la localité est mauvaise pour des accès séquentiels ou sur clés secondaires, ce qui conduit à de nombreux déplacement de bras.
 - Taille de l'index pouvant être importante.

Conclusion

- Les fichiers séquentiels sont efficaces pour le parcours rapide, l'insertion et la suppression, mais lents pour la recherche.
- Les fichiers triés sont assez rapides pour les recherches (très bons pour certaines sélections), mais lents pour l'insertion et la suppression.
- Les fichiers hachés sont efficaces pour les insertions et les suppressions, très rapides pour les sélections avec égalité, peu efficaces pour les sélections ordonnées.
- Les index permettent d'améliorer certaines opérations sur un fichier. Les Arbres-B⁺ sont les plus efficaces.