Stockage et indexe

Réda DEHAK reda@lrde.epita.fr

Création et suppression d'indexe

Création d'indexe :

CREATE [UNIQUE] [CLUSTERED | NONCLUSTERED] INDEX index_name **ON** table_name (column_name [ASC|DESC])

Suppression d'indexe :

DROP INDEX index_name

Architecture d'un SGBD

Requêtes SQL

Évaluation des requêtes

Plan d'exécution

Parser

Évaluation des opérateurs

Optimisation

Gestionnaire des transactions

Gestion de la concurrence

Gestionnaire des verrous

Méthodes d'accès et fichiers

Gestionnaire du tampon

Gestionnaire de l'espace disque

Récupération

SGBD

Base de données

Index, données

Support des données

- **Disques :** Permet un accès direct à n'importe quelle page avec un coût fixe.
  - L'accès à des pages consécutives est plus rapide qu'un accès aléatoire.
- **Bandes magnétiques :** lecture des pages en séquence.
  - Moins chère que les disques
  - Utilisées pour l'archivage et les sauvegardes.

Organisation des données

- **Fichier :** ensemble d'enregistrements de données sur le support externe (plusieurs pages).
  - Un enregistrement est identifié par un Record id (RID)
  - Indexes sont des structures de données qui permettent de retrouver les enregistrements correspondants aux RID à partir de la clé de l'indexe.
- **Le gestionnaire du tampon :** copie les pages de la mémoire physique vers les tampons de la mémoire vive.
  - Les accès aux fichiers et indexes passent par le gestionnaire du tampon (buffer manager)

Organisation des fichiers

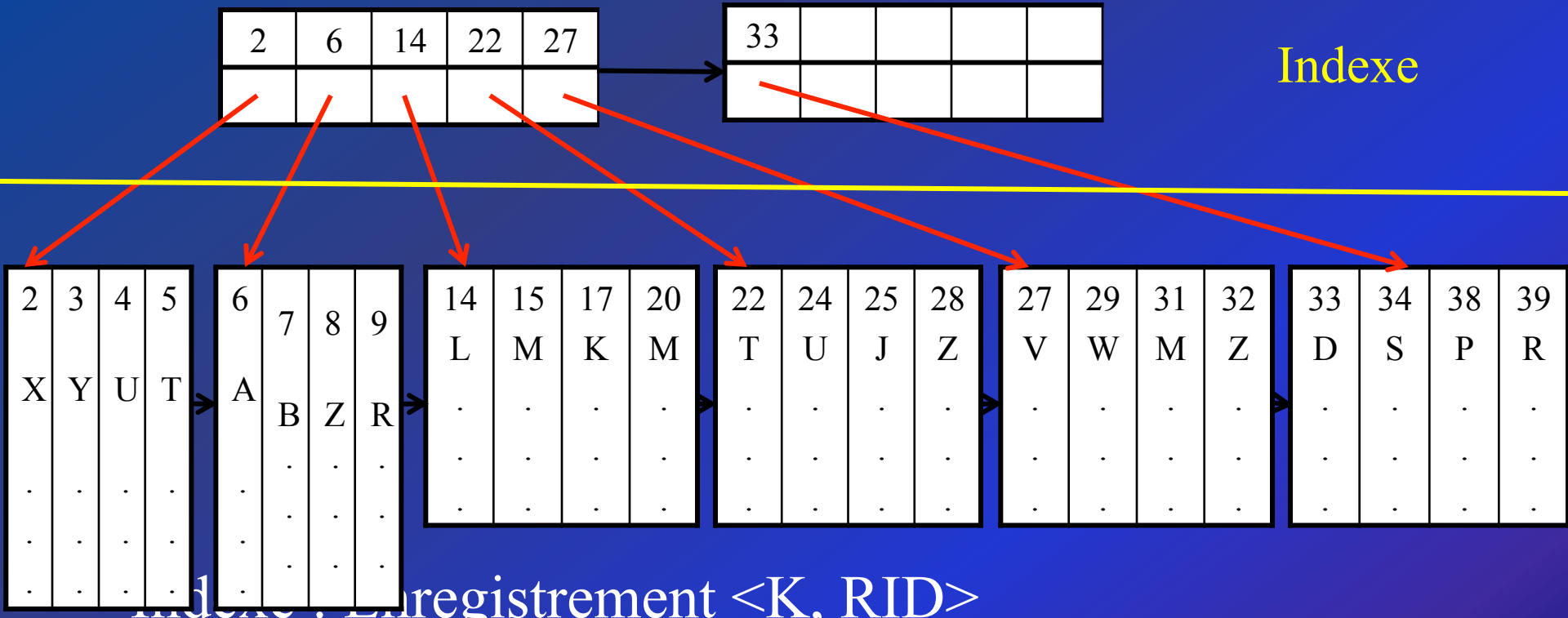
Plusieurs méthodes d'organisation sont possibles :

- **Tas (ordre aléatoire) :**
  - Utile pour un accès global à l'ensemble des données (scanner)
- **Fichier trié :**
  - Utile si les données doivent être extraites dans un ordre particulier.
  - Accès rapide à un enregistrement particulier du fichier à partir de la valeur de la clé du tri
- **Fichier indexé :** structure de données pour organiser les enregistrement en arbre ou à l'aide d'une fonction de hachage.
  - Comme les fichiers triés pour la vitesse d'accès.
  - Les modifications de la base sont plus rapide par rapport au cas des fichiers triés.

Indexes

- Un indexe augmente **la vitesse d'accès** au fichier à partir de la clé de l'indexe :
  - Une **clé d'indexe** est constituée de n'importe quel sous ensemble d'attributs de la relation.
  - La **clé d'indexe** est différente de la clé de la relation (ensemble minimal permettant d'identifier un enregistrement de la relation).

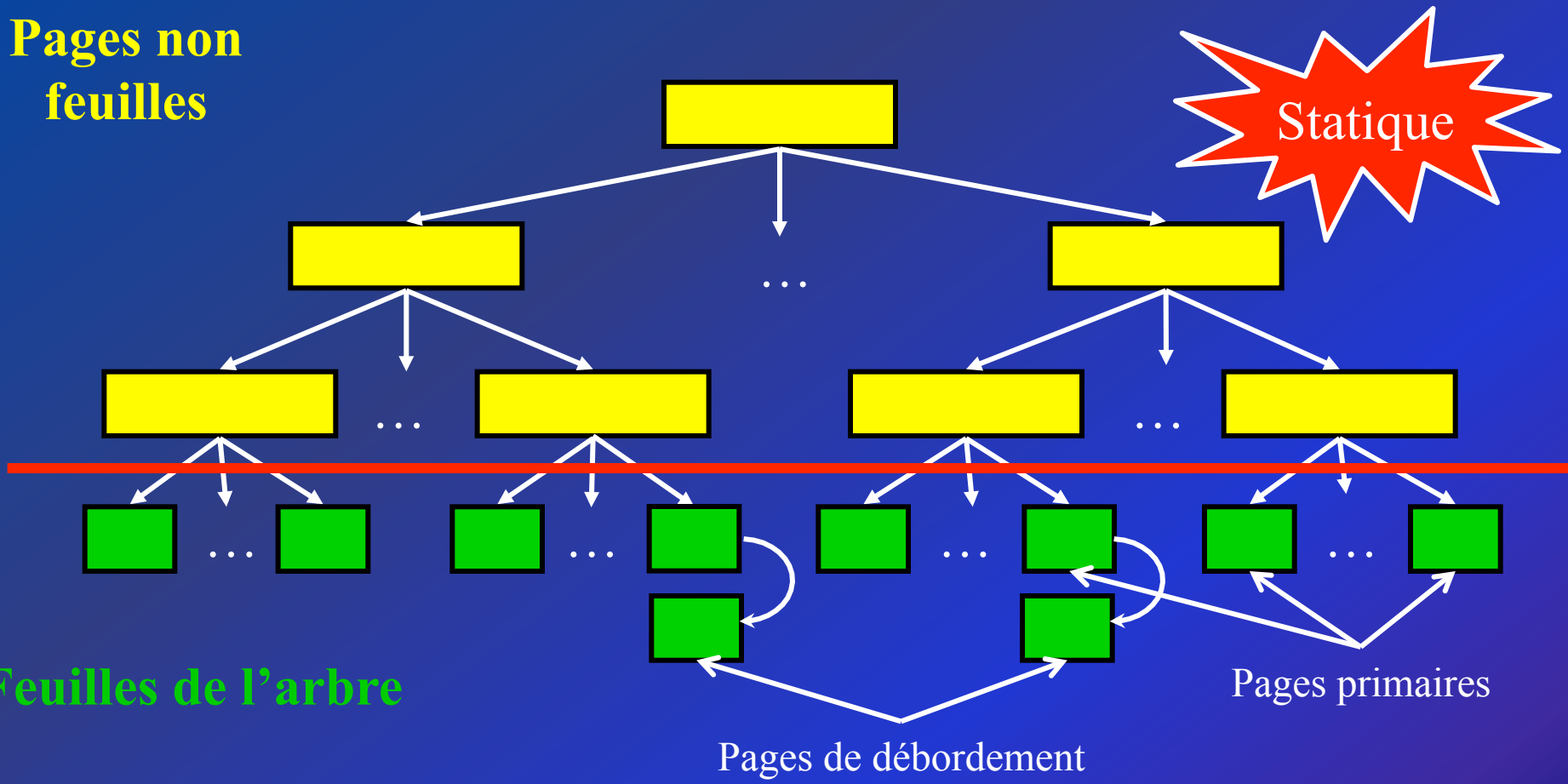
Indexes



- Index : Enregistrement <K, RID>
- Fichier : Enregistrement K*

Indexed Sequential Access Method (ISAM)

Pages non feuilles



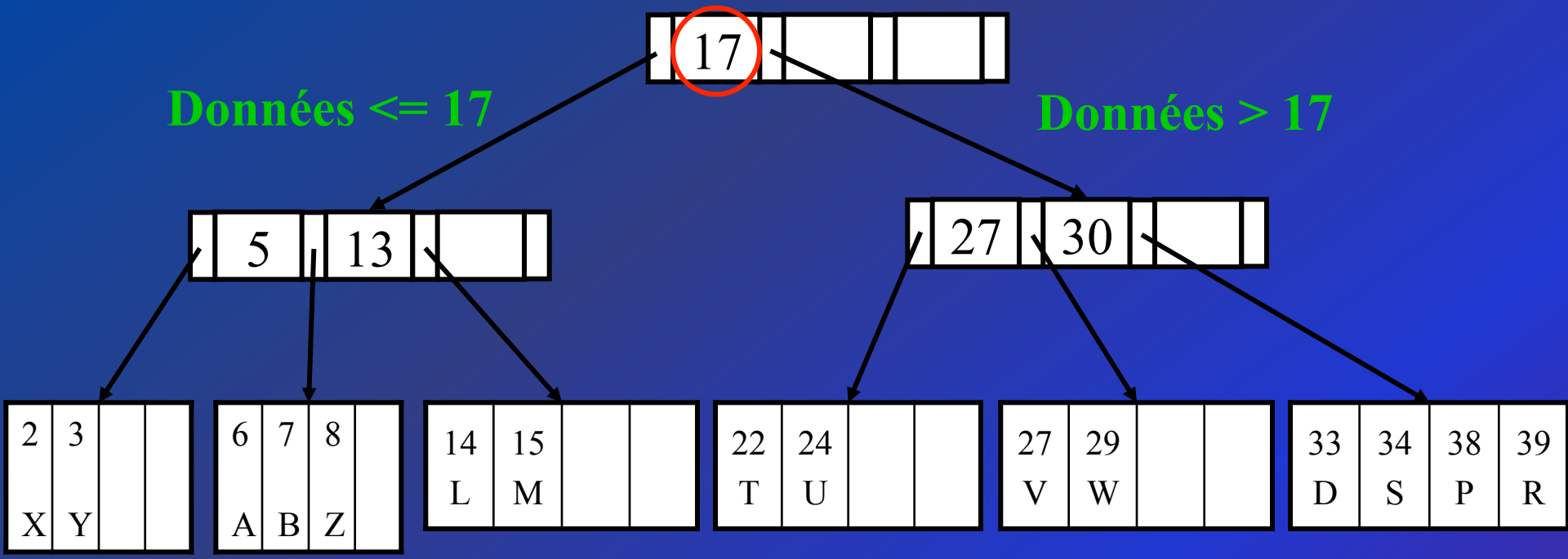
Indexed Sequential Access Method (ISAM)

- Pages feuilles et de débordements : contiennent les données triées suivant la clé de recherche.
- Pages non feuilles : contiennent les clés de recherche et les n° des pages pour diriger la recherche.
- Recherche : Commencer à partir de la racine et descendre vers les feuilles de l'arbre

  Coût : Hauteur de l'arbre
- Insertion : chercher la feuille qui doit contenir la nouvelle entrée, et insérer la clé. En cas ou il n'y a plus de place réserver une page de débordement
- Suppression : Supprimer la clé de la feuille. Si la page de débordement est vide, la libérer.

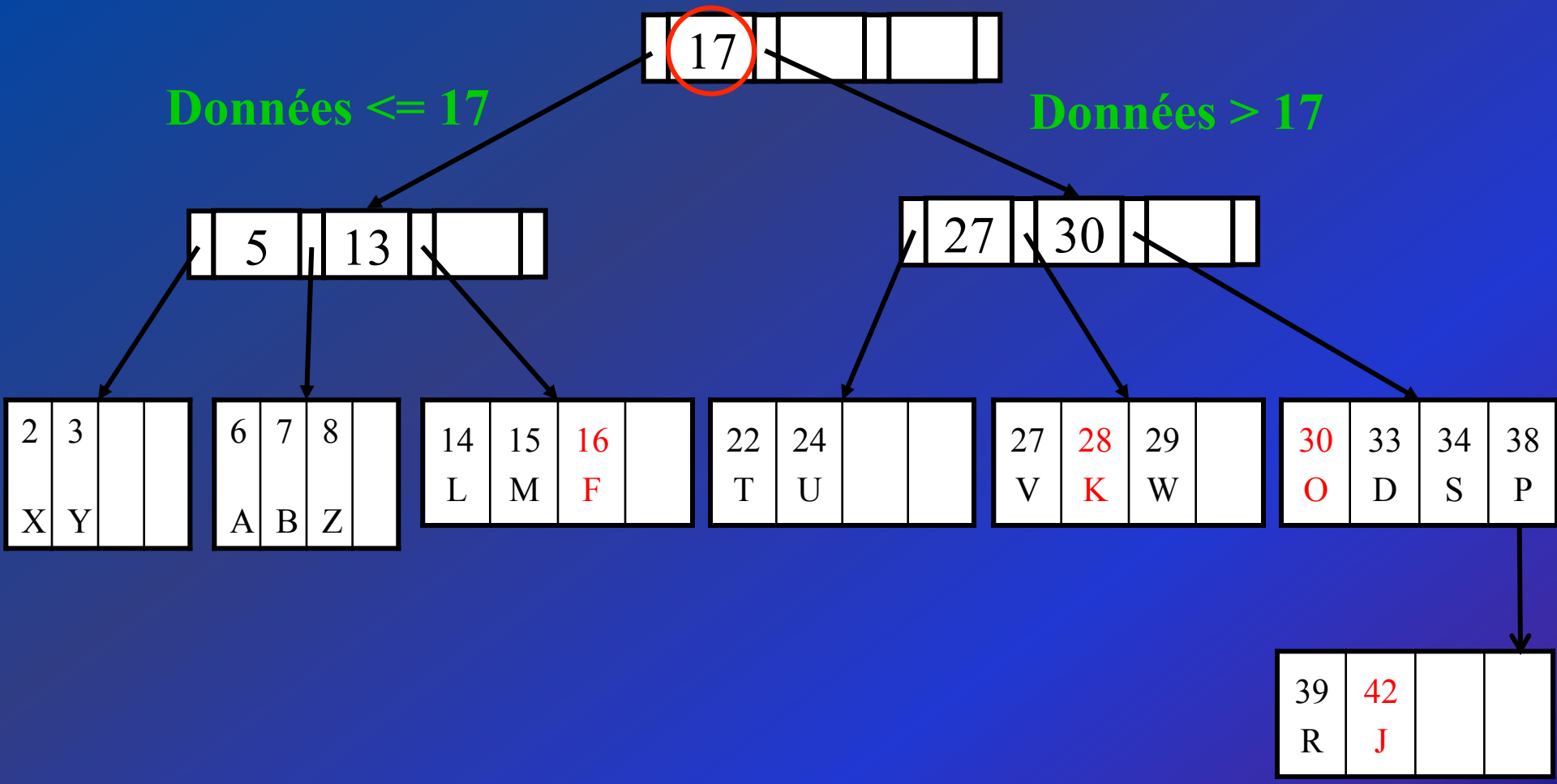
- La structure de l'indexe est statique pas de changement.

Exemple

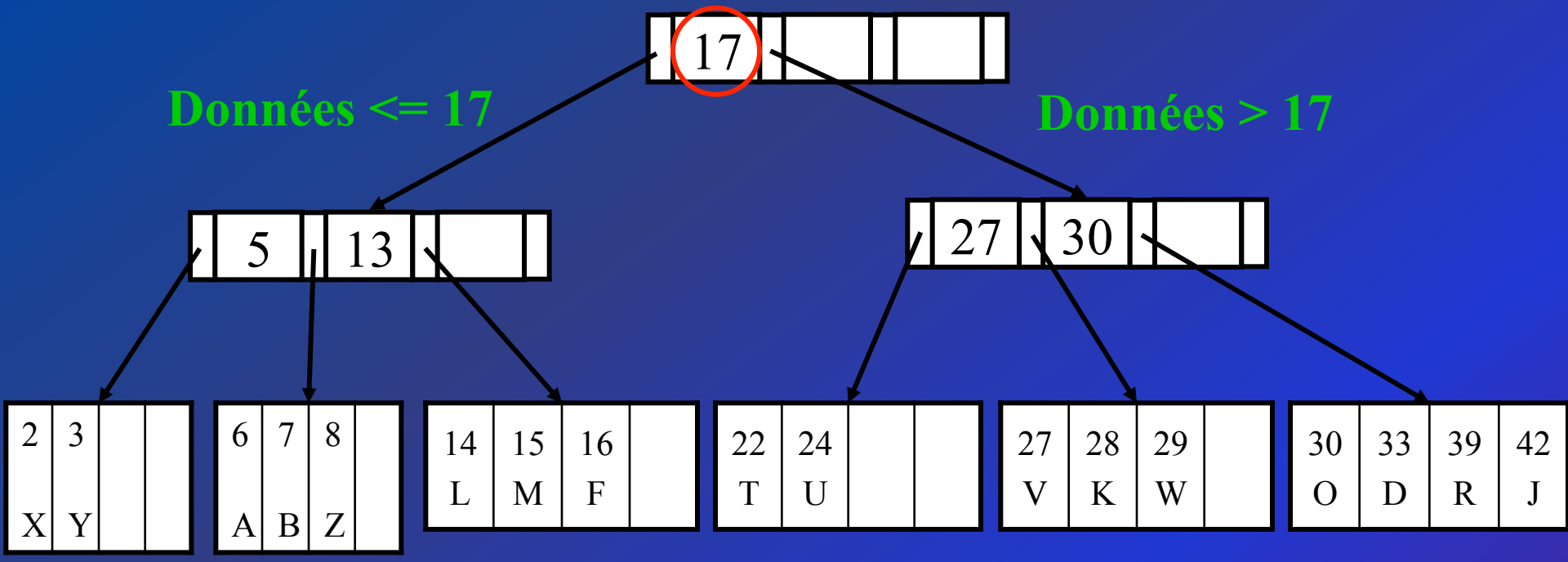


- Rechercher 28*? 29*?
- Rechercher les tuples pour qui vérifient 15*< clé < 30*?
- INSERT / DELETE : rechercher la page concernée et insérer (ou supprimer) la nouvelle données*
  - Dans certain cas, il est nécessaire de rajouter ou bien libérer une page de débordement

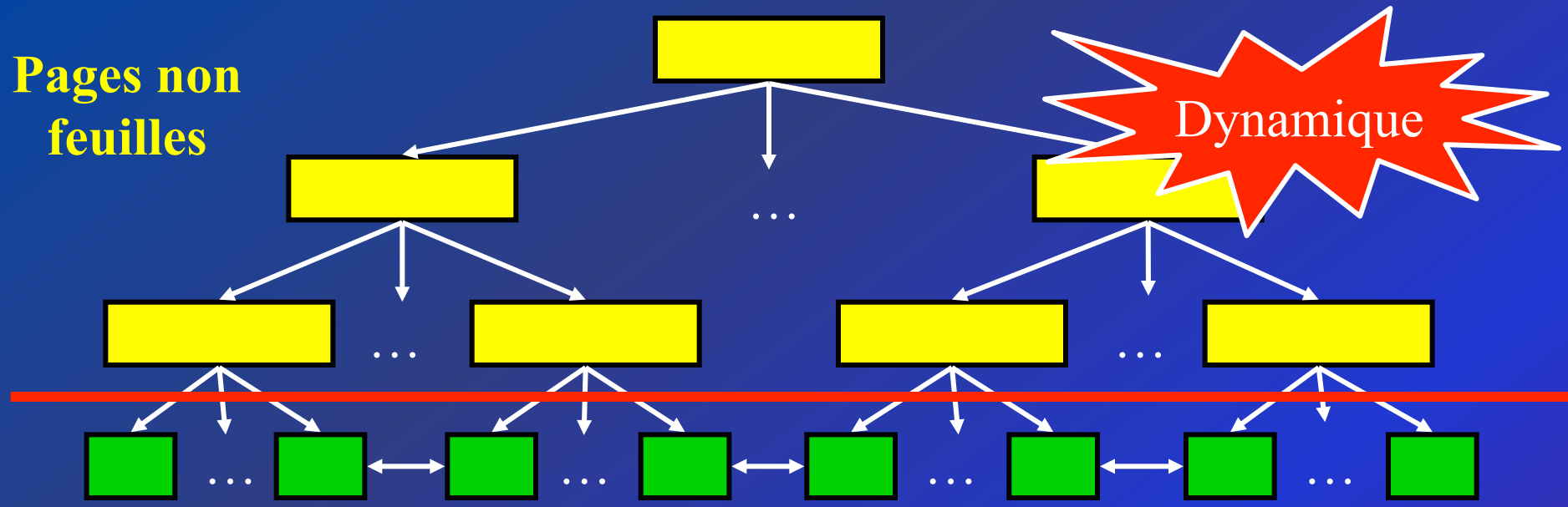
Insertion 28, 30, 42, 16



Suppression 34, 38

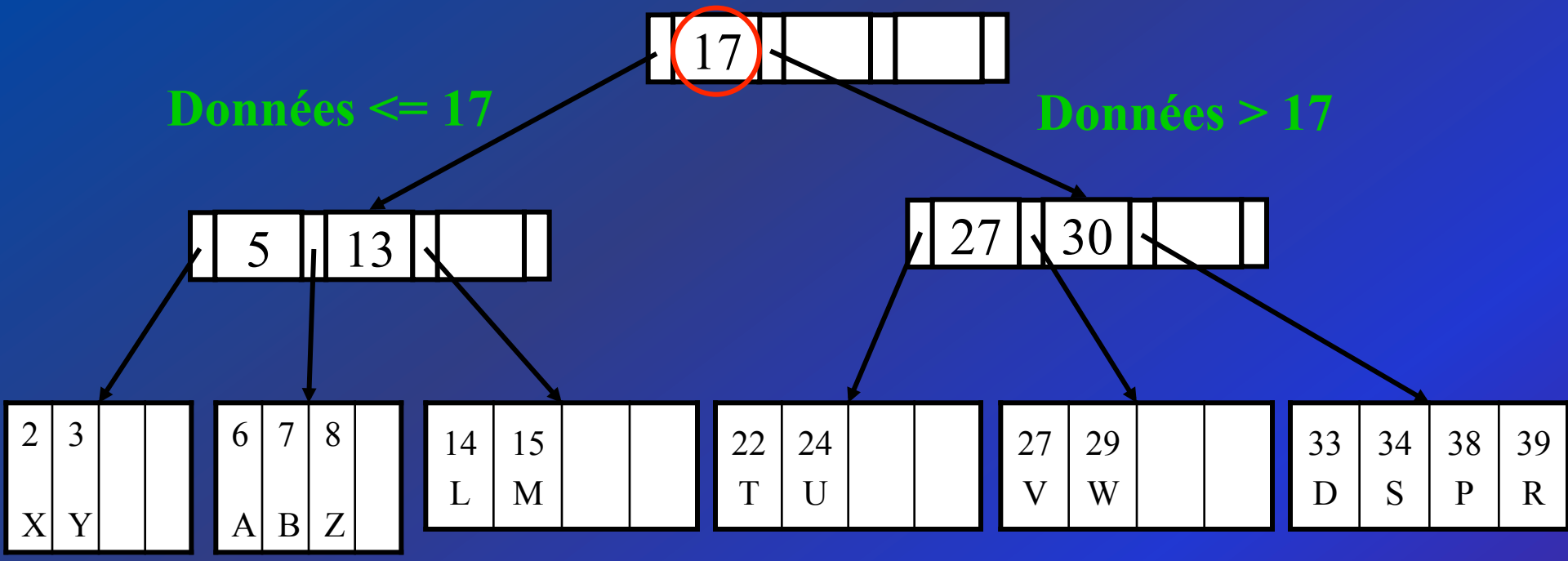


Indexes : Arbre B+



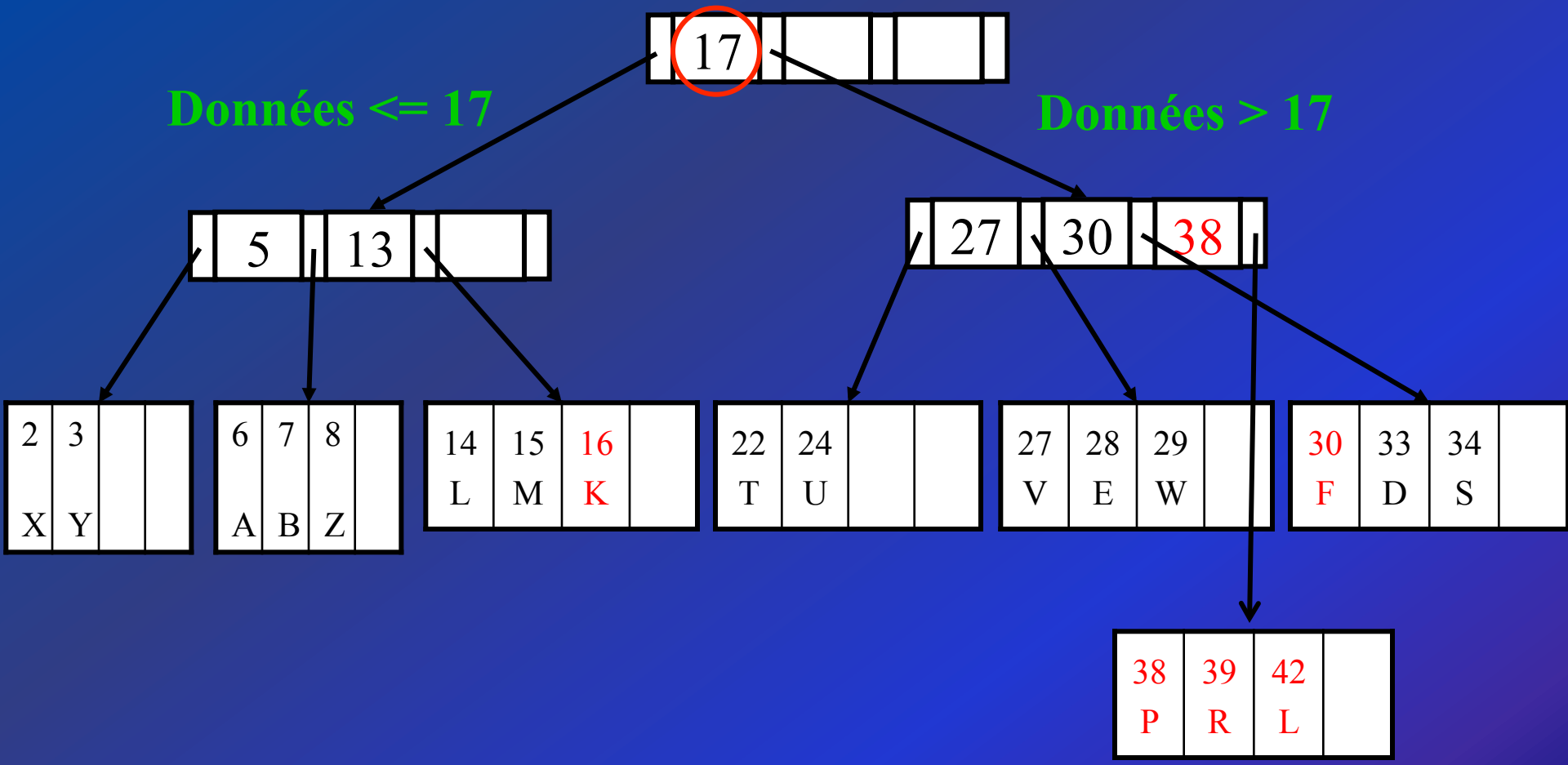
- Les pages feuilles contiennent les données et sont doublement chaînées (avant arrière)
- Les pages non feuilles contiennent les clés de l'indexe, elles sont utilisées pour diriger la recherche.

Exemple



- Rechercher 28*? 29*?
- Rechercher les tuples pour qui vérifient 15*< clé < 30*?
- INSERT / DELETE : rechercher la page concernée et insérer (ou supprimer) la nouvelle données*
  - Dans certain cas, il est nécessaire de modifier le nœud parent.

Insertion 28, 30, 42, 16



Hauteur/nbre de page indexées

- Généralement un ordre de 100 avec un taux de remplissage de 67% → soit 133 fils possibles pour chaque nœud
  - Niveau 0 : 133^0 = 1 pages : 8ko
  - Niveau 1 : 133^1= 133 pages : 1Mo
  - Niveau 2 : 133^2= 17689 pages : 133Mo
  - Niveau 3 : 133^3 = 2352637 pages : 17To
  - Niveau 4 : 133^4 = 312900721 pages : 2Oo

Indexe : méthodes arborescentes

- Méthodes idéales pour accélérer les recherches avec critère d'égalité ou bien Between
- Méthode ISAM : structure statique
  - Seules les feuilles de l'indexe peuvent être modifiées
  - Nécessite des pages de débordement
  - Les performances peuvent se dégrader avec l'augmentation du nombre des pages de débordement.
  - Avantage pour le verrouillage
- Méthode B+-arbre : structure dynamique
  - Les insertions/suppressions gardent l'arbre équilibré.
  - L'importance de l'ordre F implique une hauteur faible dépassant rarement 3 ou 4 niveaux

Indexe : méthodes arborescentes

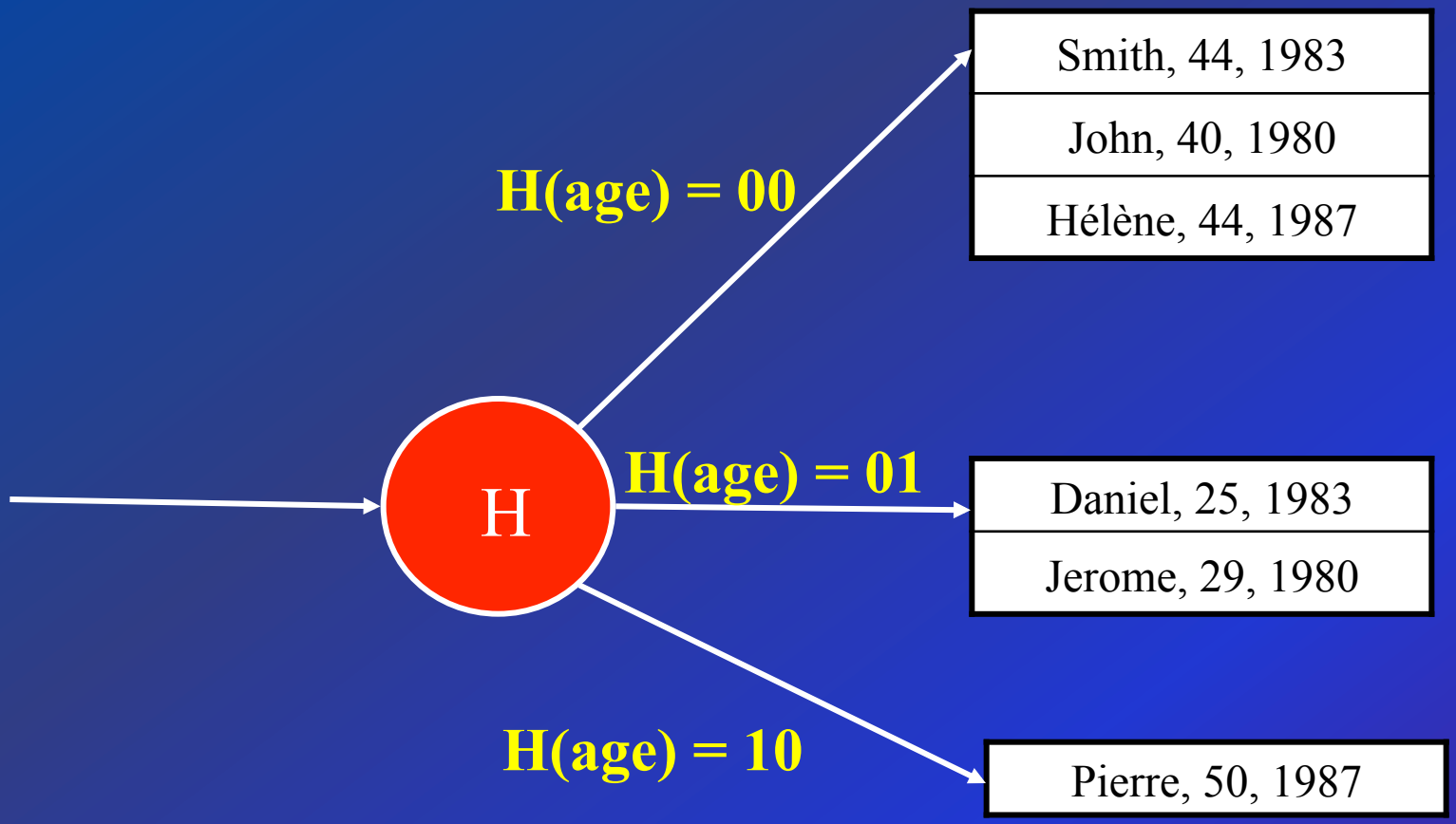
- Méthode B+-arbre : structure dynamique
  - En moyenne un taux de remplissage de 67%
  - Préférable à la méthode ISAM (modulo verrouillage)
  - Si les feuilles de l'arbre contiennent les données, les splits engendre un changement des rid.

- La compression des clés permet d'augmenter l'ordre.
- Les indexes les plus utilisés dans la conception des bases de données

Indexe : hachage

- Le meilleur pour une sélection avec un critère d'égalité.
- Un indexe est une collection de paquets :
  - Un paquet = une page primaire + 0 ou n pages de débordement.
  - Les paquets contiennent les données.
- Fonction de hachage H :
  - H(k) = l'adresse du paquet qui doit contenir les données correspondant à l'enregistrement contenant la clé k.

Exemple


H(age) = 00
H(age) = 01
H(age) = 10
H
Smith, 44, 1983
John, 40, 1980
Hélène, 44, 1987
Daniel, 25, 1983
Jerome, 29, 1980
Pierre, 50, 1987

Les données de l'indexe

Les entrées k* peuvent correspondre à :

1. Les données avec la clé de valeur k.

   Indexe Plaçant (Clustered Index)

2. <k, RID de l'enregistrement correspondant à la clé k>

   Indexe non plaçant (Unclustered Index)

3. <k, liste des RID correspondants à la clé k>

   Indexe non plaçant sans unicité de la clé k

Les données de l'indexe

1. **Les données avec la clé de valeur k :**
   - L'indexe est une structure pour organiser le fichier de données (enregistrement)
   - Un seul indexe peut être du type 1, sinon il faut dupliquer les données)
2. **Pour les autres cas des données pour rediriger la recherche sont stockées dans les feuilles de l'indexe.**

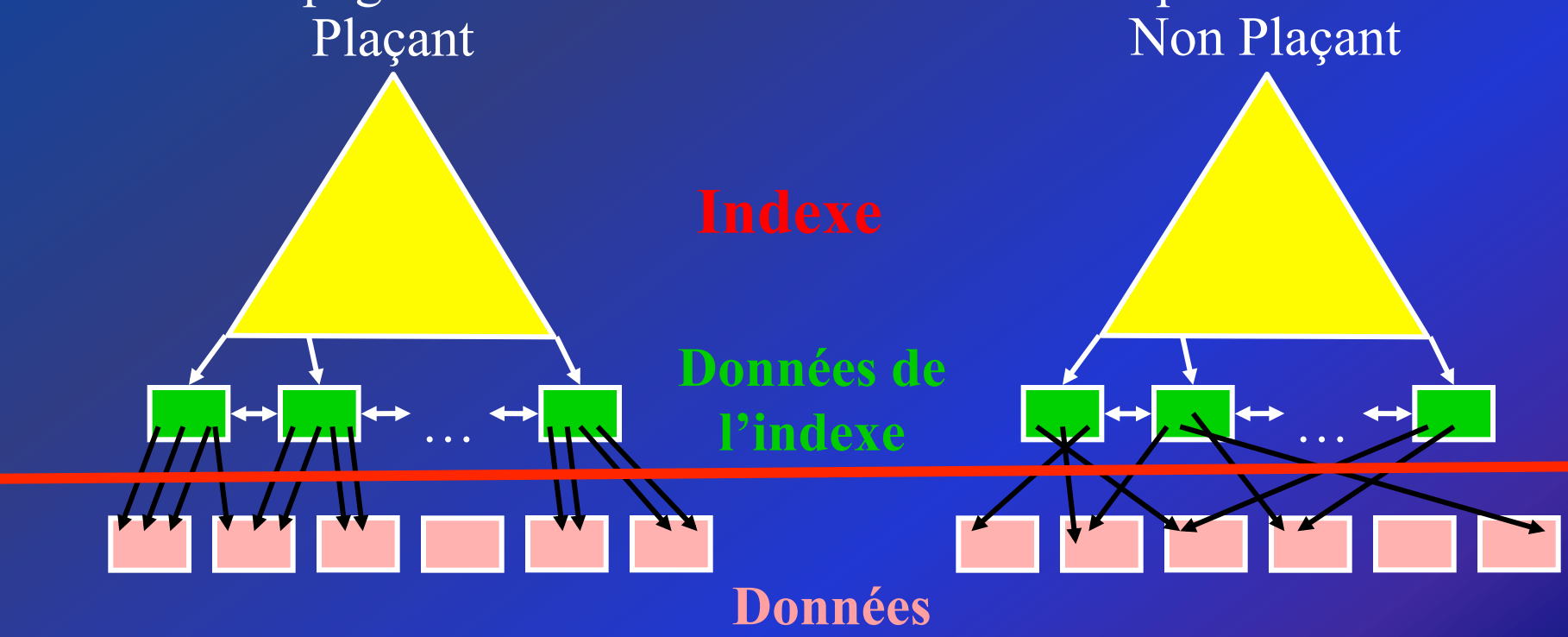
Classification des indexes

- **Primaire / Secondaire**
  - Indexe primaire : Si la clé de recherche contient la clé primaire de la relation.
  - Indexe secondaire : Si la clé primaire de la relation n'est pas incluse dans la clé de l'indexe.
- Plaçant / Non Plaçant (Clustered/Unclustered)
  - Si les données sont stockées dans le même ordre que l'indexe alors l'indexe est plaçant.
  - La méthode 1 implique un indexe plaçant.
  - En pratique un indexe plaçant implique la méthode 1 (les fichiers triés ne sont jamais implémentés)
  - Les coûts d'accès varient selon que l'indexe est plaçant ou non.

Indexe Plaçant / Non Plaçant

Construction d'un indexe plaçant :

- Trier le fichier tas en ajoutant des places libres dans chaque page
- Les pages de débordement sont nécessaire pour les insertions.



Modèle pour l'évaluation des coûts d'accès

On s'intéresse aux coûts de lecture / écriture :

- P : le nombre de pages de la base de données.
- E : Le nombre d'enregistrements par page.
- D : temps moyen de lecture écriture d'une page disque.

Remarque :

On ignore les gains liés à une lecture séquentielle des pages consécutives

Comparaison

Un fichier contenant les informations d'un employé : nom, prénom, age, salaire.

1. Tas : Ordre aléatoire des enregistrements dans le fichier.
2. Fichier Trié sur (age)
3. Fichier indexé (plaçant) (age) avec un arbre B+ (méthode 1).
4. Un tas avec un indexe (age) non plaçant en utilisant un arbre B+.
5. Un tas avec un indexe (age) non plaçant utilisant les fonctions de hash.

Opérations de comparaison

1. Scanner : extraire tous les enregistrement de la table.

   SELECT * FROM Emp;

2. Recherche avec critère d'égalité.

   SELECT * FROM Emp WHERE age = 29;

3. Sélection avec un critère BETWEEN.

   SELECT * FROM Emp WHERE age BETWEEN 20 AND 40;

4. Insertion.

   INSERT INTO Emp VALUES(42, 'Dubois', 'Alain', 42, 2500)

5. Suppression.

   DELETE FROM Emp WHERE age = 42;

Hypothèse

- Tas :
  - Le critère de sélection avec égalité aboutit à un seul résultat.
- Fichier trié :
  - Compacter le fichier après suppression.
- Indexes :
  - Méthode (2 et 3) : taille de la clé de l'indexe = 10% de la taille d'un enregistrement.
  - Hash : pas de page de débordement
  - Taux de remplissage des pages avec B-TREE : 67%.
  - Taux de remplissage des pages avec méthode de HASH : 80%.
- Scanner :
  - Les feuilles de l'indexe sont doublement chaînées.

Comparaison

	Scan	Egalité	Between	Insert	Delete
Tas					
Trié					
Indexe plaçant					
Non plaçant, arbre B+					
Non plaçant, hash					

Scan

SELECT * FROM Emp;

1. **Tas :**
   - Il faut lire toutes les pages de la tables Emp

Nombre de page à lire : P

Coût : DP

2. **Fichier trié :**
   - Il faut lire toutes les pages de la tables Emp

Nombre de page à lire : P

Coût : DP

   - **Avantage :** le résultat est trié selon les valeurs de la clé de tri.

Scan

3. **Indexe plaçant :**
   - Il faut lire toutes les pages de la tables Emp

Nombre de page à lire : 1.5 * P

Coût : 1.5DP

   - **Avantage :** le résultat est trié selon les valeurs de la clé de tri.

4. **Indexe B-TREE non plaçant :**
   - Il faut lire toutes les pages de données de l'indexe

Nombre de page à lire : 0.15 * P

   - Pour chaque tuple, charger une page, donc il faut lire une page pour chaque tuple de la table employé :

Nombre de page à lire : PR

Coût : DP (R + 0.15)

   - **Avantage :** le résultat est trié selon les valeurs de la clé de tri.

Scan

5. Indexe hash non plaçant :

- Il faut lire toutes les pages de données de l'indexe (les paquets)

Nombre de page à lire : 0.125 * P

- Pour chaque tuple, charger une page, donc il faut lire une page pour chaque tuple de la table employé :

Nombre de page à lire : PR

Coût : DP (R + 0.125)

- **Avantage :** le résultat est trié selon les valeurs de la clé de tri.

Comparaison

	Scan	Egalité	Between	Insert	Delete
Tas	DP				
Trié	DP				
Indexe plaçant	1.5DP				
Non plaçant, arbre B+	DP * (R + 0.15)				
Non plaçant, hash	DP * (R + 0.125)				

Égalité

SELECT * FROM Emp WHERE age = 29;

1. **Tas :**
   - Dans le pire des cas, on doit lire toutes les pages de la table employé,
   - Dans le meilleur des cas, on lit une seule page :

Nombre de page à lire en moyenne : 0.5 P

Coût : 0.5 DP

2. **Fichier trié :**
   - On utilise la dichotomie :

Nombre de page à lire : $Log_2(P)$

Coût : D $Log_2(P)$

Égalité

3. **Indexe plaçant :**
   - Il faut lire toutes les pages des nœuds internes du B-TREE de la racine vers les feuilles pour trouver la page concernée :

Nombre de page à lire : $H = Log_F(1.5 P)$

Coût : $H D = D Log_F(1.5 P)$

   - F le nombre de fils pour chaque nœud du B-TREE.

4. **Indexe B-TREE non plaçant :**
   - Il faut lire toutes les pages des nœuds internes du B-TREE de la racine vers les feuilles pour trouver la page concernée de l'indexe:

Nombre de page à lire : $Log_F (0.15 * P)$

   - Lire la page qui contient le tuple concerné :

Nombre de page à lire : 1

Coût : $D (Log_F (0.15 * P) + 1)$

Égalité

5. Indexe hash non plaçant :

- En utilisant la fonction de hash, lire la page des données de l'indexe concernée

Nombre de page à lire : 1

- Déterminer et charger la page de données concernée

Nombre de page à lire : 1

Coût : 2D

Comparaison

	Scan	Egalité	Between	Insert	Delete
Tas	DP	0.5DP			
Trié	DP	$D\log_2(P)$			
Indexe plaçant	1.5DP	$D\log_F(1.5P)$			
Non plaçant, arbre B+	DP (R + 0.15)	$D(1+\log_F(0.15P))$			
Non plaçant, hash	DP (R + 0.125)	2D			

BETWEEN

SELECT * FROM Emp WHERE age BETWEEN 20 AND 40;

1. **Tas :**
   - Il faut lire toutes les pages

Nombre de page à lire : P

Coût : DP

2. **Fichier trié :**
   - On utilise la dichotomie pour trouver le premier élément ensuite continuer en séquence sur les pages suivante:

Nombre de page à lire : $Log_2(P) + T_0 / R$

Coût : $D (Log_2(P) + T_0 / R)$

- T_0 : nombre de tuples qui vérifient la condition du BETWEEN

BETWEEN

3. Indexe plaçant :

– Il faut lire toutes les pages des nœuds internes du B-TREE de la racine vers les feuilles pour trouver la page qui contient le premier tuple et continuer en séquence sur les autres pages:

Nombre de page à lire :

$$H + 1.5\ T_0 / R = Log_F(1.5\ P) + 1.5\ T_0 / R$$

$$\text{Coût : } D\ (Log_F(1.5\ P) + 1.5\ T_0 / R)$$

4. Indexe B-TREE non plaçant :

– Il faut lire toutes les pages des nœuds internes du B-TREE de la racine vers les feuilles pour trouver la première page de l'indexe du premier tuple qui vérifie la condition du between et continuer en séquence :

$$\text{Nombre de page à lire : } Log_F\ (0.15 * P) + 0.15\ T_0 / R$$

– Pour chaque tuple, lire une page

$$\text{Nombre de page à lire : } T_0$$

$$\text{Coût : } D\ (Log_F\ (0.15 * P) + 0.15\ T_0 / R + T_0)$$

BETWEEN

5. Indexe hash non plaçant :

- Il faut utiliser le scan, en particulier dans le cas où l'age est un attribut réel :
- Il faut lire toutes les pages de données de l'indexe (les paquets)

Nombre de page à lire : 0.125 * P

- Pour chaque tuple, charger une page, donc il faut lire une page pour chaque tuple de la table employé :

Nombre de page à lire : PR

Coût : DP (R + 0.125)

Comparaison

	Scan	Egalité	Between	Insert	Delete
Tas	DP	0.5DP	DP		
Trié	DP	$D\log_2(P)$	$D (\log_2(P) + T_0 / R)$		
Indexe plaçant	1.5DP	$D\log_F(1.5P)$	$D (\log_F(1.5P) + 1.5\ T_0 / R)$		
Non plaçant, arbre B+	$DP (R + 0.15)$	$D(1+ \log_F(0.15P))$	$D (\log_F(1.5P) + 0.15\ T_0 / R + T_0)$		
Non plaçant, hash	$DP (R + 0.125)$	2D	DP(R+0.125)		

INSERT

INSERT INTO Emp VALUES(42, ‘Dubois’, ‘Alain’, 42, 2500);

1. **Tas :**
   - Il faut lire la dernière page du tas :

     Nombre de page à lire : 1
   - Il faut écrire la page mémoire après modification

     Nombre de page à écrire : 1

     Coût : 2D
2. **Fichier trié :**
   - On cherche l’endroit où il faut insérer le tuple (coût identique à une recherche).
   - On insère le tuple en décalant tous les autres d’une case sur la page courante et sur les pages consécutives.

     Nombre de page traiter en moyenne : P/2

     Coût : Recherche + DP

BETWEEN

3. Indexe plaçant :

- On cherche l'endroit où il faut insérer le tuple (coût identique à une recherche).
- On insère le tuple et on écrit la nouvelle page :

Coût : Recherche + D

4. Indexe B-TREE non plaçant :

- On cherche l'endroit où il faut insérer le tuple (coût identique à une recherche).
- On insère le tuple dans l'indexe et dans la page de données :

Coût : Recherche + 2D

BETWEEN

5. Indexe hash non plaçant :

- On cherche l'endroit où il faut insérer le tuple (coût identique à une recherche).
- On insère le tuple dans l'indexe et dans la page de données :

Coût : Recherche + 2D

Comparaison

	Scan	Egalité	Between	Insert	Delete
Tas	DP	0.5DP	DP	2D	
Trié	DP	$D\log_2(P)$	$D (\log_2(P) + T_0 / R)$	Search + DP	
Indexe plaçant	1.5DP	$D\log_F(1.5P)$	$D (\log_F(1.5P) + 1.5\ T_0 / R)$	Search +D	
Non plaçant, arbre B+	$DP (R + 0.15)$	$D(1+ \log_F(0.15P))$	$D (\log_F(1.5P) + 0.15\ T_0 / R + T_0)$	Search +2D	
Non plaçant, hash	$DP (R + 0.125)$	2D	$DP(R+0.125)$	Search + 2D	

Comparaison

	Scan	Egalité	Between	Insert	Delete
Tas	DP	0.5DP	DP	2D	Search + D
Trié	DP	$D\log_2(P)$	$D (\log_2(P) + T_0 / R)$	Search + DP	Search + DP
Indexe plaçant	1.5DP	$D\log_F(1.5P)$	$D (\log_F(1.5P) + 1.5\ T_0 / R)$	Search +D	Search + D
Non plaçant, arbre B+	$DP (R + 0.15)$	$D(1+ \log_F(0.15P))$	$D (\log_F(1.5P) + 0.15\ T_0 / R + T_0)$	Search +2D	Search + 2D
Non plaçant, hash	$DP (R + 0.125)$	2D	$DP(R+0.125)$	Search + 2D	Search + 2D

Conclusion

- Les méthodes d'indexation ont des avantages et des inconvénients.
- Il faut choisir le type d'indexes qui est le plus favorable pour les traitement qu'on veut faire.

- Question :
  - faut-il rajouter un indexe pour chaque requête exécutée?
  - Quels sont les inconvénients d'un indexe