

# Bases de données avancées

---

**Jean-Yves Antoine**

LI - Université François Rabelais de Tours

Jean-Yves.Antoine@univ-tours.fr



UFR Sciences et Techniques  
IUP GMI Blois – IUP3

# Bases de données avancées

---

**Performances : indexation et optimisation**

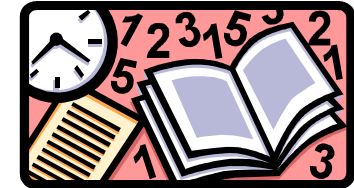


UFR Sciences et Techniques  
IUP GMI Blois – IUP3

# Problématique

## Bases de données : très grosses masses de données

- Entrées / sorties : temps d'accès à un enregistrement physique
- Requête SQL : combinatoire des calculs relationnels (jointure,...)



## Exemples

- Restriction

```
SELECT nom, prenom FROM base_insee WHERE ville='Blois';
```

**insee**

50 000 000  
tuples

- Jointure

```
SELECT c.nom FROM client c, pays p  
WHERE p.pays = c.pays  
AND p.continent = 'Europe';
```

**client**

10000  
tuples

**pays**

200  
tuples

*(5000 clients Européens ; 25 pays Européens)*



# Optimisation de performances

---

- **Accès aux données** : indexation, hachage
- **Jointure physique entre tables** : clusters
- **Optimisation des requêtes** : optimisation logique et physique
- **Tuning de l'organisation physique** : *block size*

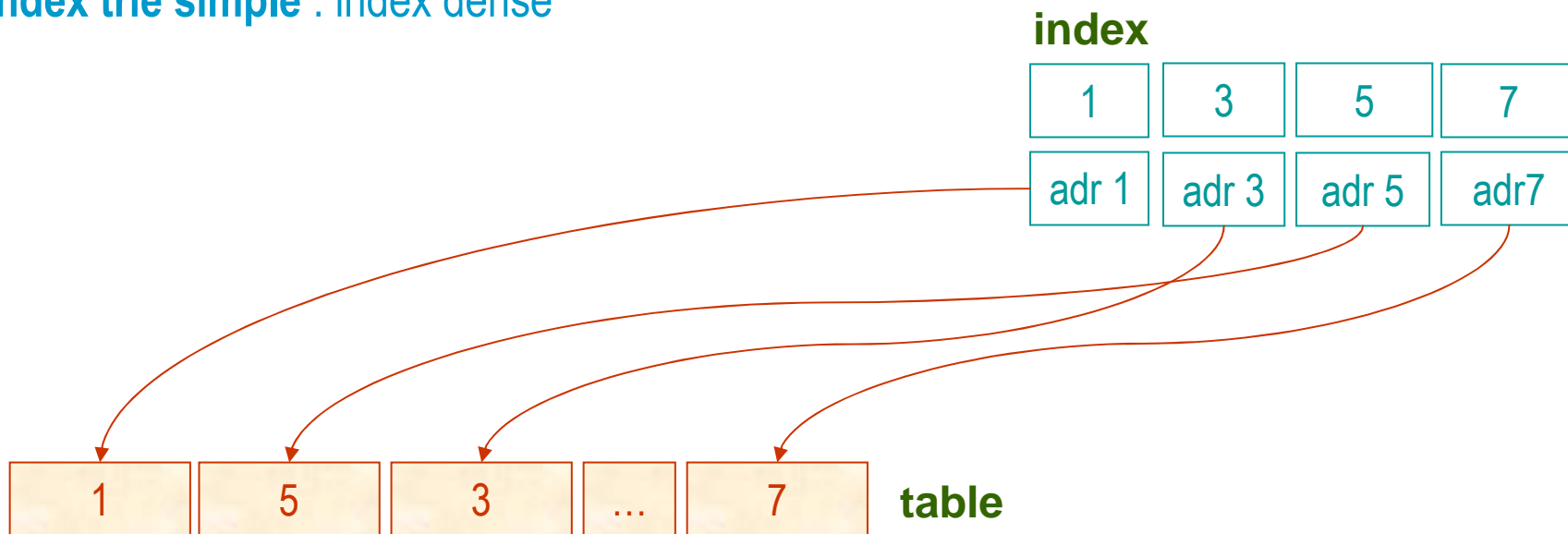


# Indexation

## Principe

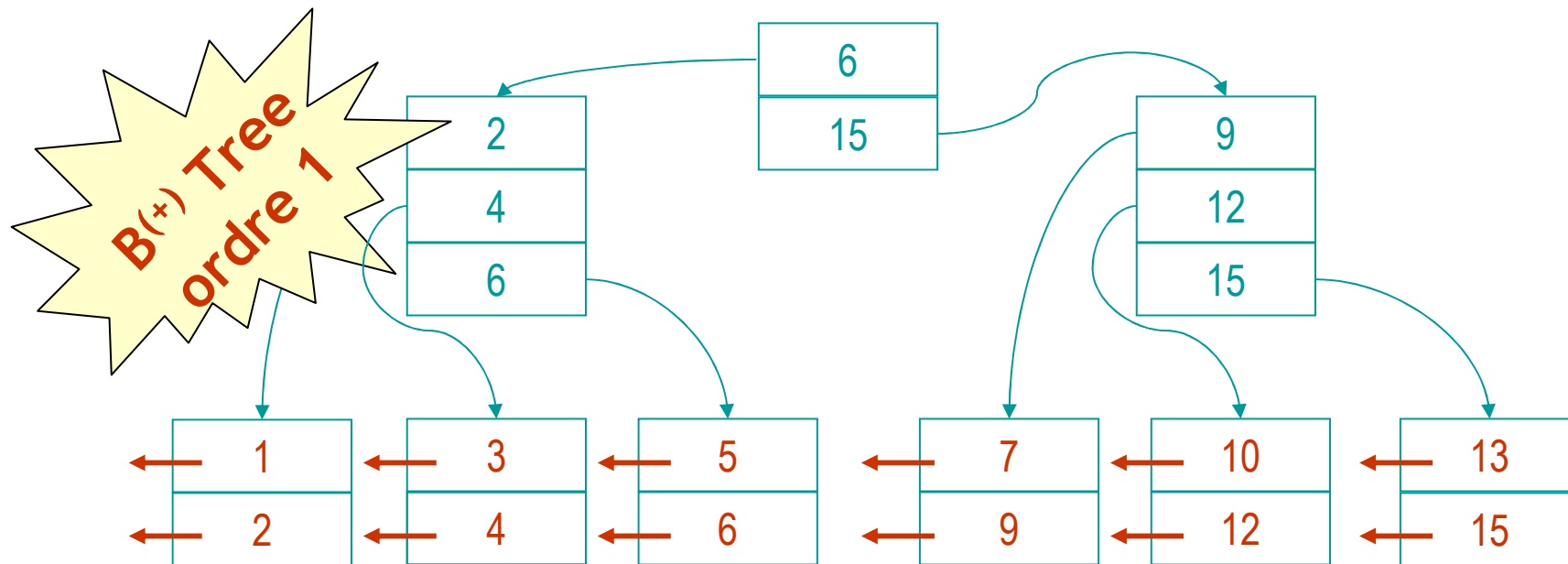
- **Index** : table associant directement à une clé l'adresse du tuple considéré
- Organisation d'index : brut, trié, hiérarchisé, avec hachage
- Index sur clé primaire mais également sur champ très utilisés (« clé » secondaire)

## Index trié simple : index dense



# Index trié et hiérarchisé : arbre B

- Index non dense au premier niveau : tests de valeur à chaque niveau



- **B Tree** : arbre trié et équilibré pour optimiser le temps d'accès à l'information

**Définition formelle** : un arbre B d'ordre  $m$  est un arbre tq :

- toutes les feuilles sont au même niveau
- tout nœud intermédiaire a un nombre de feuilles compris dans  $[m+1, 2m+1]$
- le nœud racine a un nombre de feuilles compris dans  $[0, 2m+1]$  (ou 0 si arbre vide)

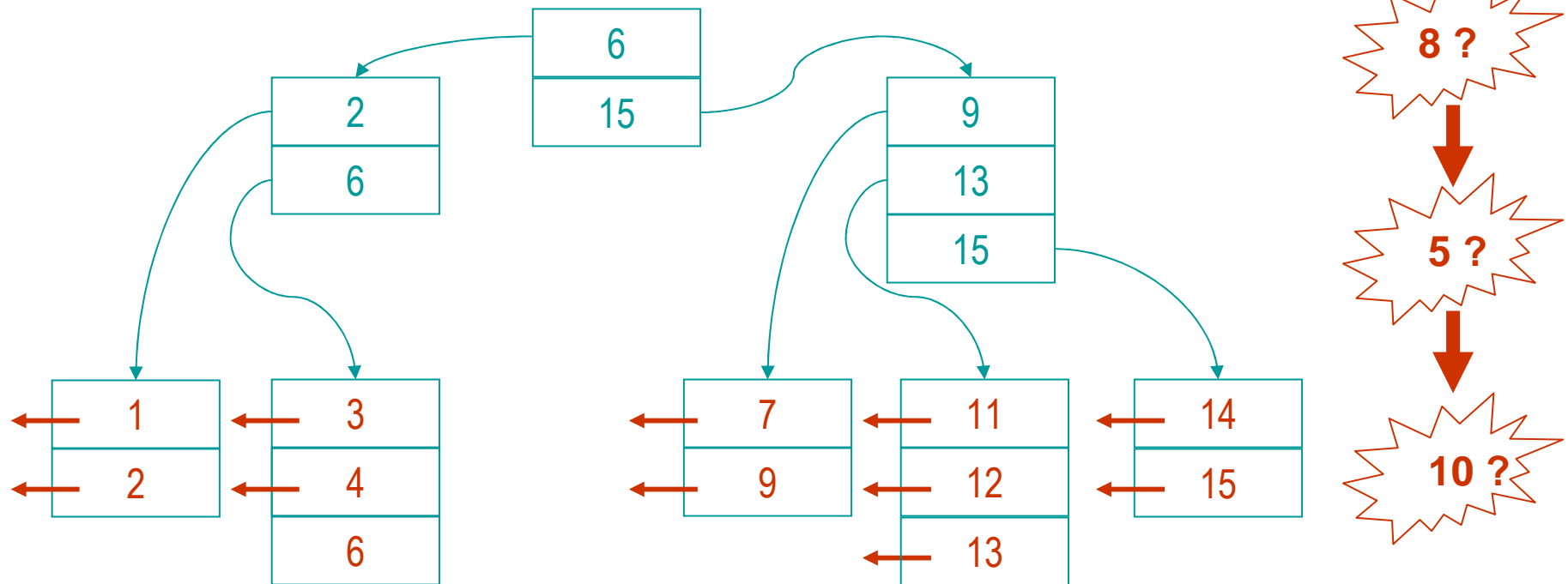


# Index hiérarchisé : arbre B

## Insertion dans un arbre B :

- a) recherche du nœud terminal où doit se situer la clé
- b) si nœud non saturé, insertion, sinon migration (éventuelle récursive) au niveau supérieur

Exemple : arbre ordre 1

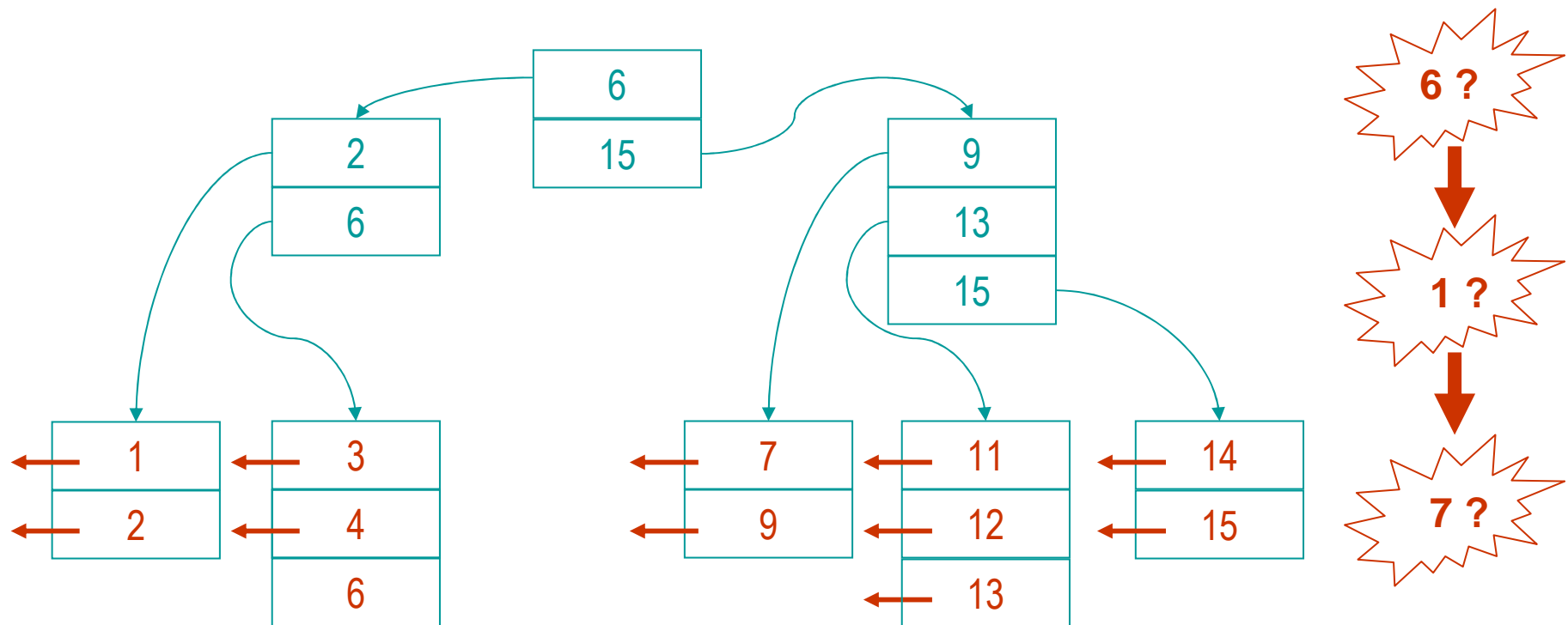


# Index hiérarchisé : arbre B

## Suppression dans un arbre B :

problème identique : fusion de nœuds éventuelle

**Exemple** : arbre ordre 1



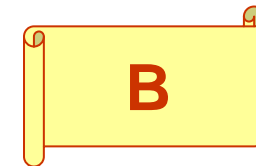
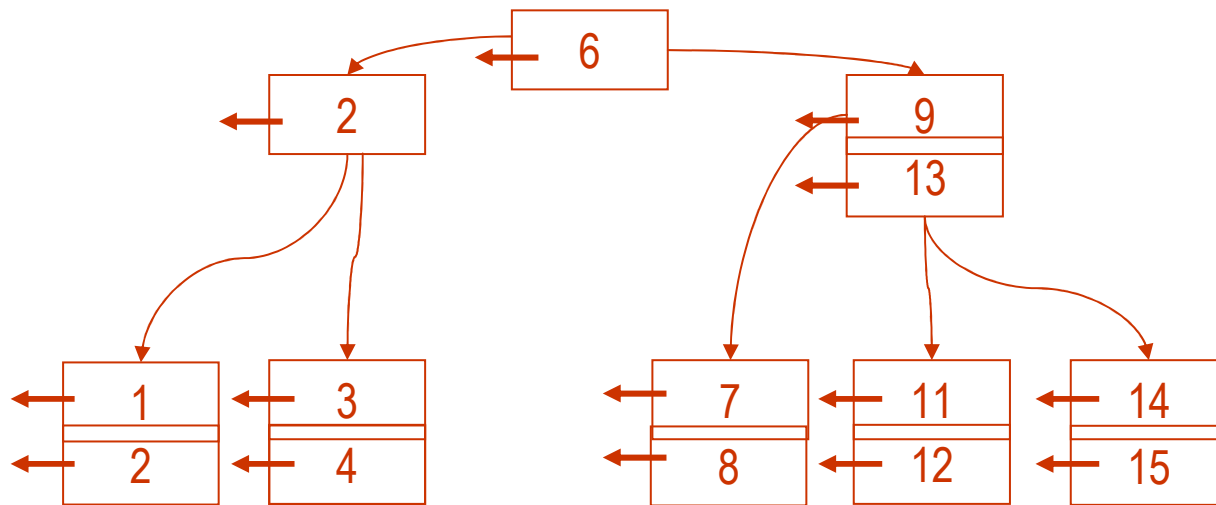
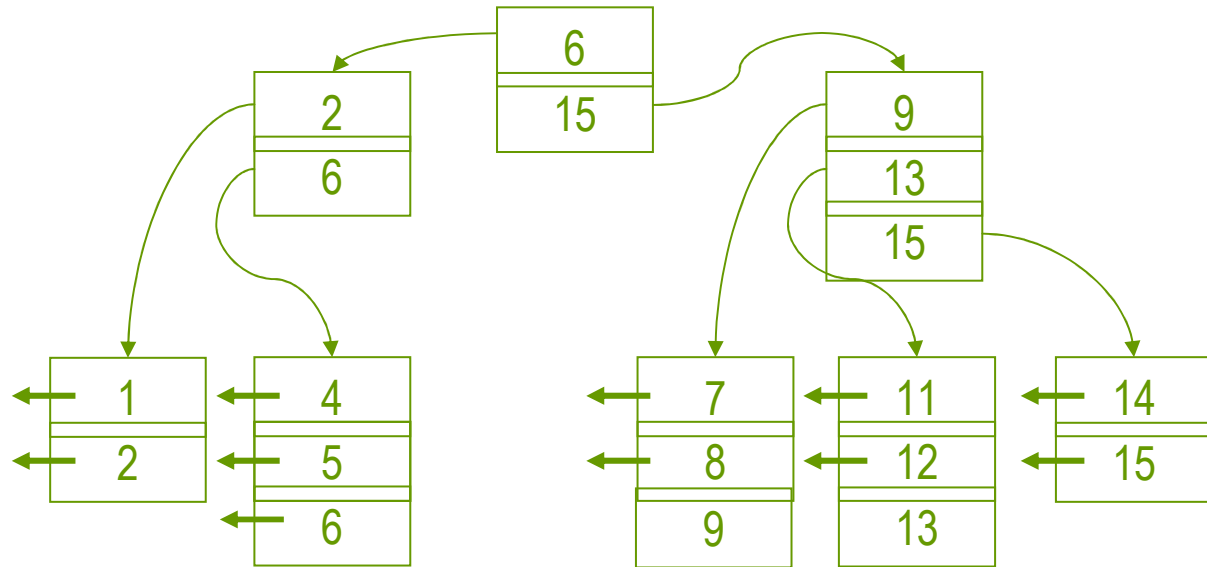
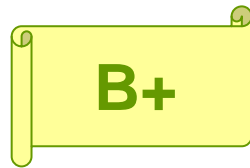
**Modification d'index** : opération lourde car re-calcul de l'arbre équilibré





# Index hiérarchisé

## Arbre B / Arbre B+



# Indexation et hachage

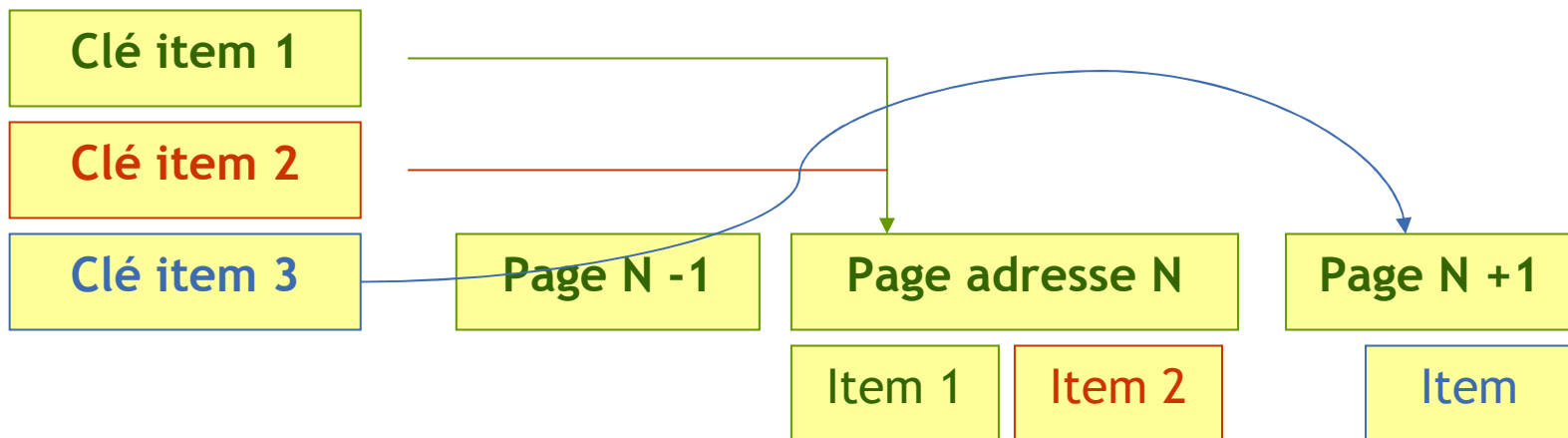
Technique générale (SGF, SGBD...) d'accès aux données par une clé



## Hachage statique

- Taille de fichier fixe

N adresses disponibles



- Fonction de hachage

modulo ou autres

- Débordement

paquets de débordement



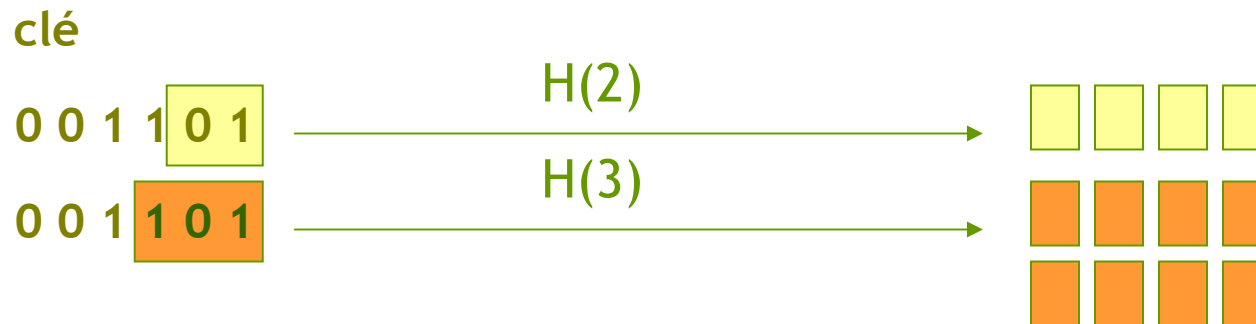
# Hachage dynamique

## Hachage statique

- ✓ excellentes performances ... tant qu'il n'y a pas de débordement
- ✓ inadapté aux données en nombre variable : indexation SGBD

## Hachage dynamique : principe

- ✓ doublement de l'espace d'adressage (page) en cas de saturation
- ✓ hachage : utilisation progressive des bits de la clé en fonction des besoins



## Hachage dynamique : méthodes

- ✓ hachage extensible
- ✓ hachage linéaire

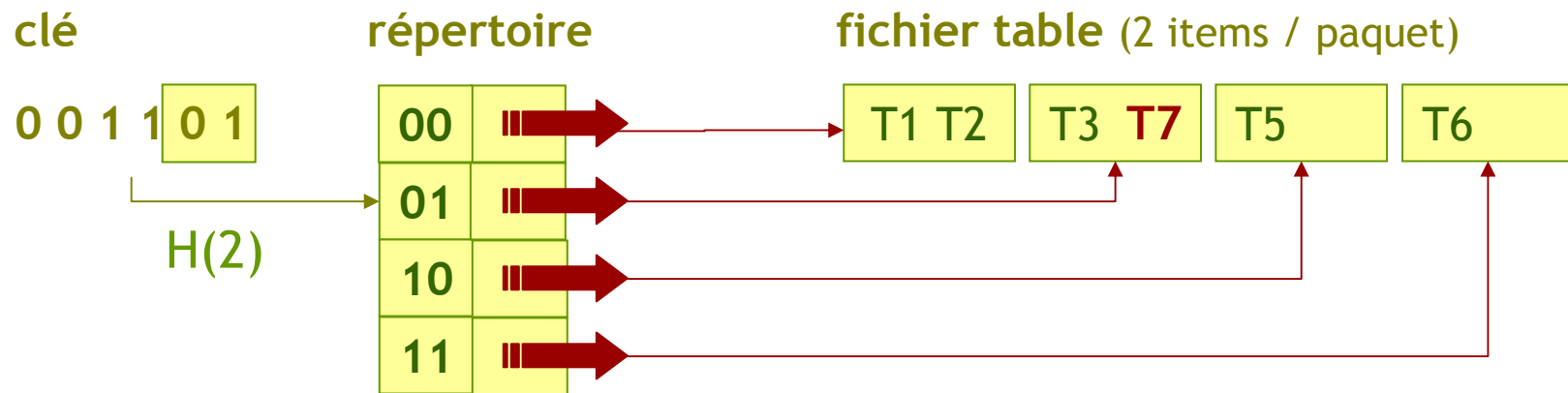


# Hachage dynamique

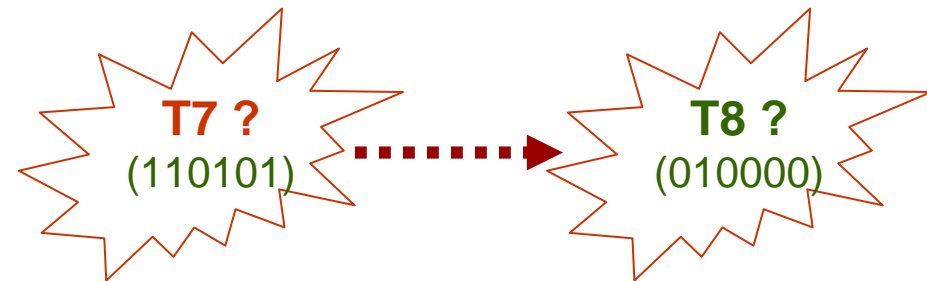
## Hachage extensible

[Fagin 1979]

- ✓ Duplication d'une page dès sa saturation
- ✓ Répertoire adressant ces pages en fonction des M premiers bits de la clé
- ✓ Doublement du répertoire si utilisation d'un bit de clé supplémentaire

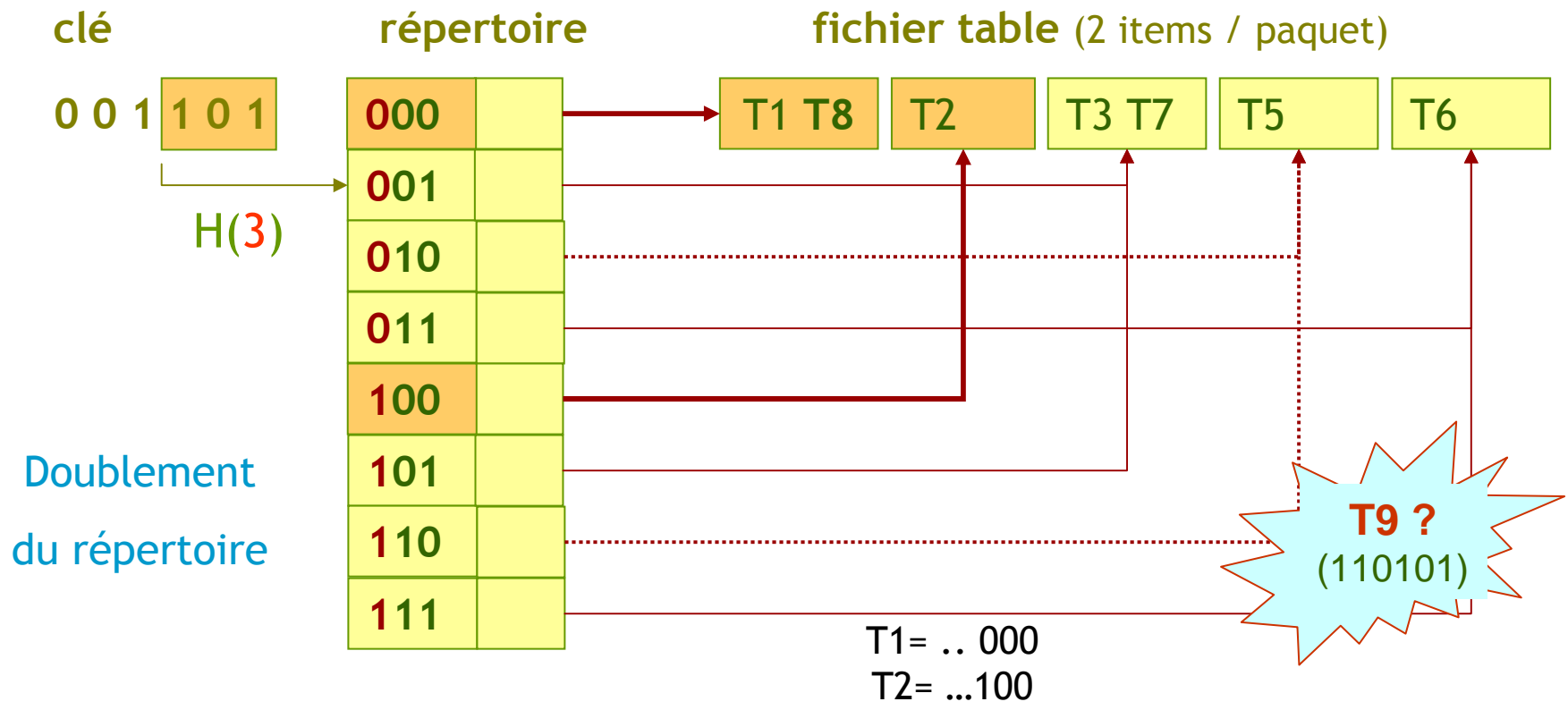


## Insertion



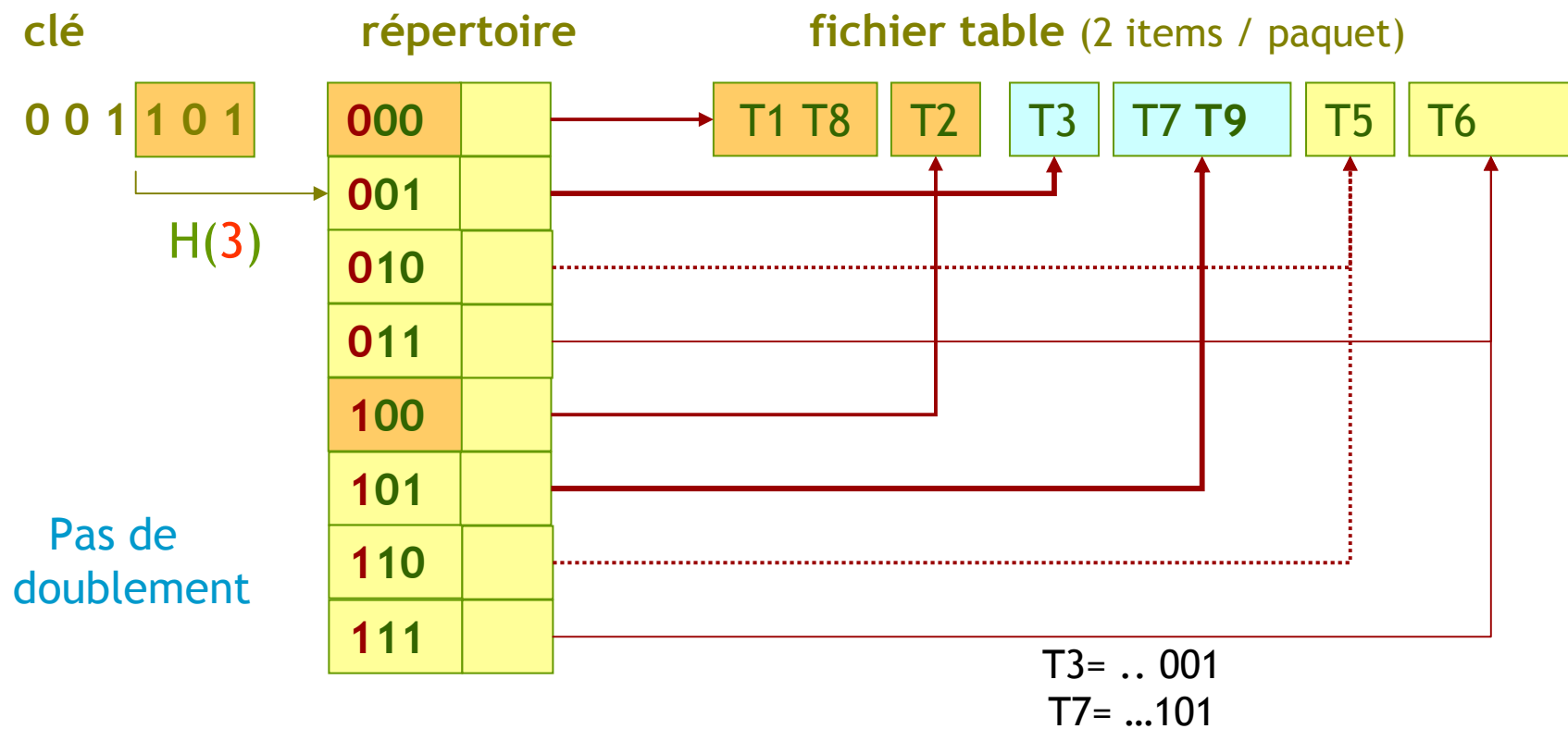
# Hachage extensible

## Insertion



# Hachage extensible

## Insertion

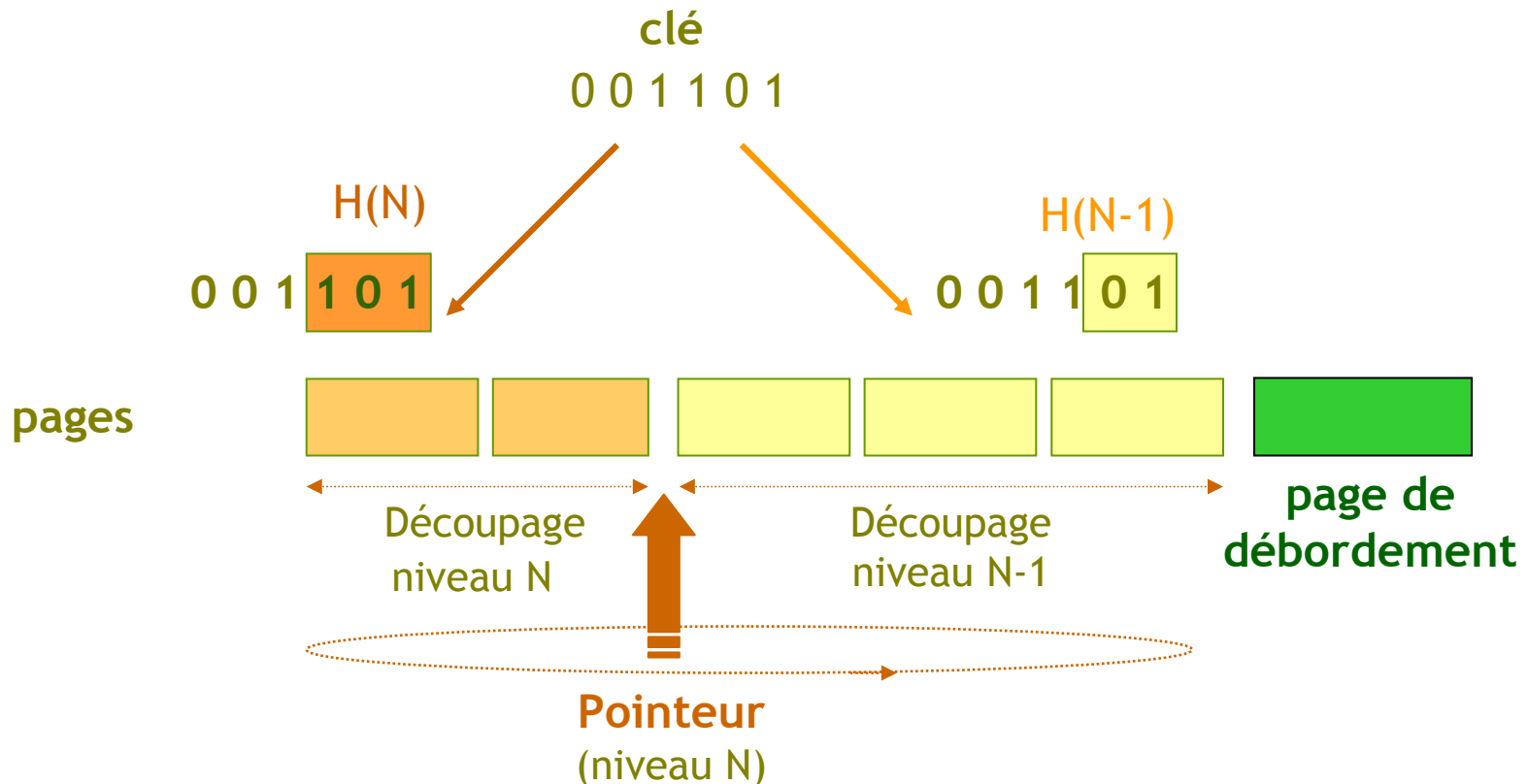


# Hachage dynamique

## Hachage linéaire

[Litwin 1980]

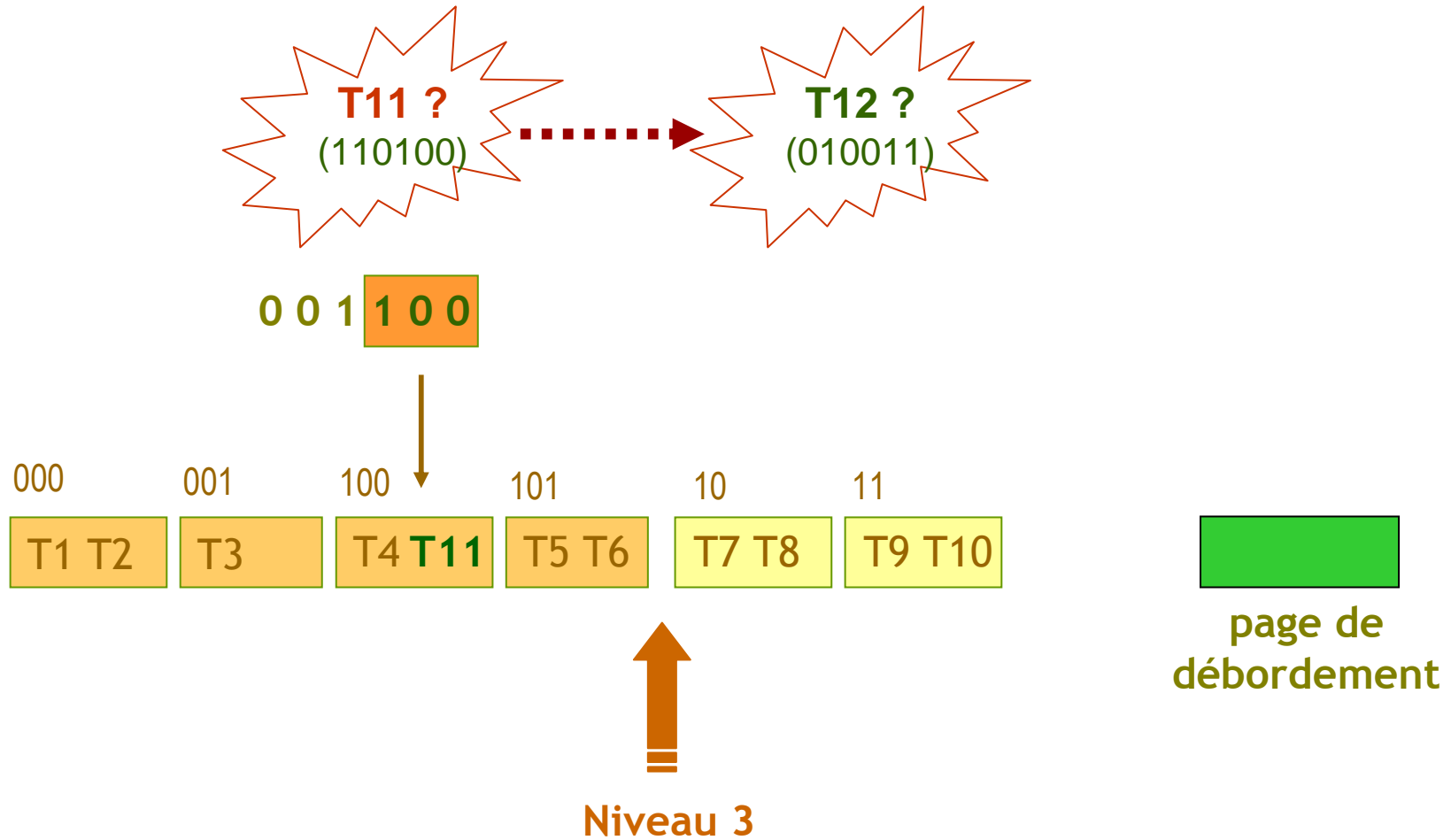
- ✓ Saturation d'une page : duplication d'une page courante
- ✓ Pointeur cyclique sur page courante + niveau d'éclatement
- ✓ Page de débordement : mise en attente d'éclatement du paquet concerné



# Hachage linéaire

Insertion

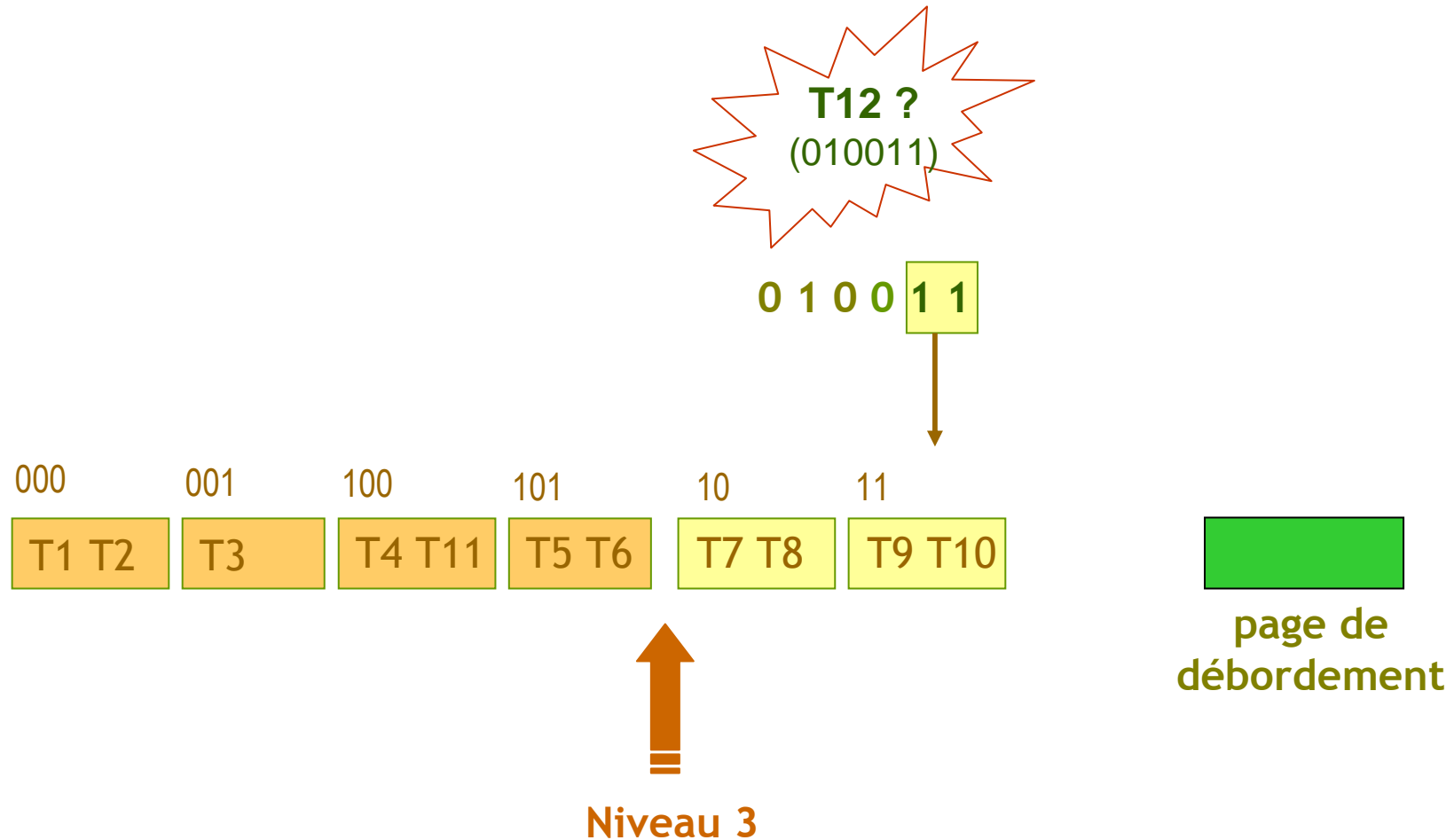
Pages de 2 articles maximum





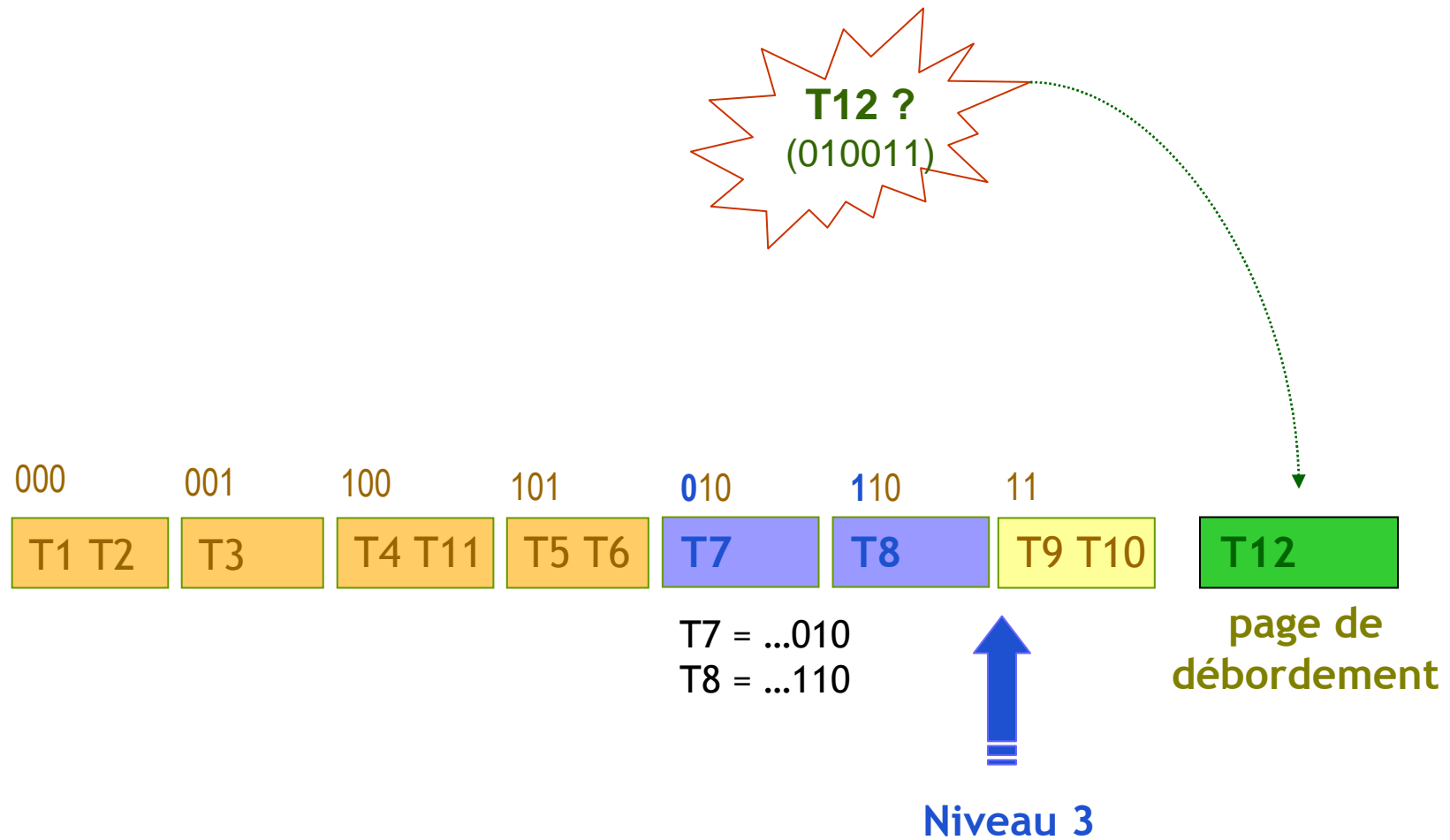
# Hachage linéaire

## Insertion avec débordement



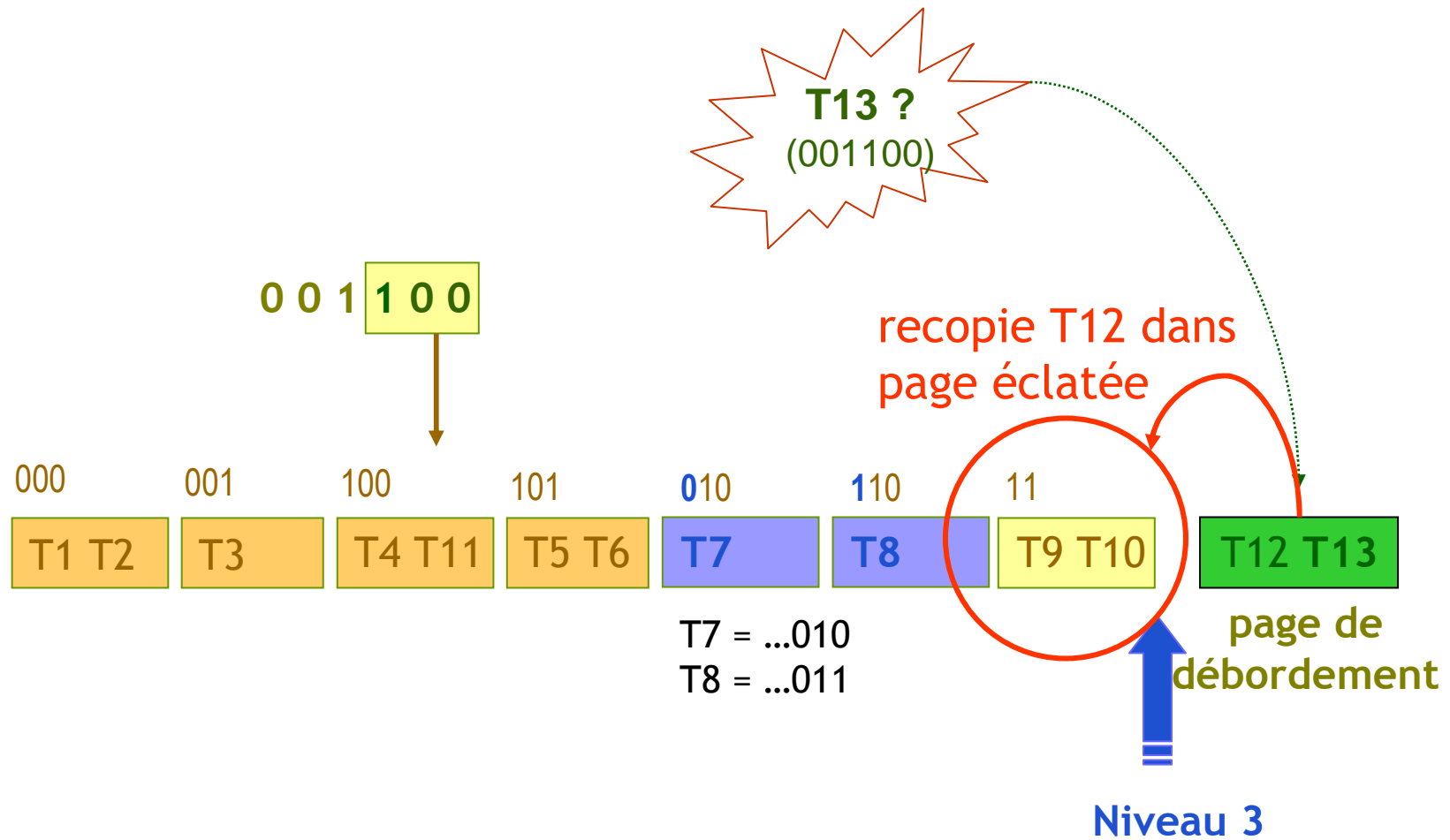
# Hachage linéaire

Insertion avec débordement : éclatement du paquet courant



# Hachage linéaire

## Insertion avec débordement



# Index secondaire

---

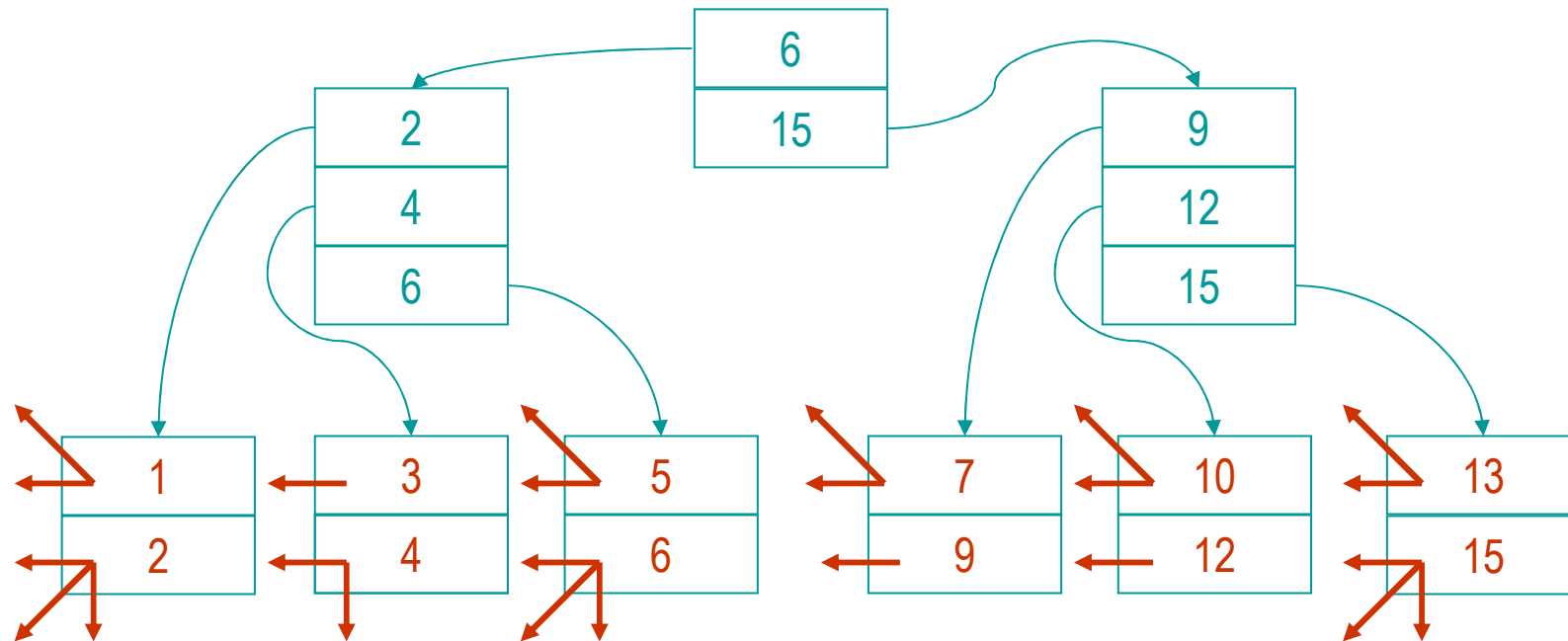
- Index sur un **attribut non clé** fréquemment utilisé dans une requête
- Index **non discriminant**
- Intégré ou à part (fichier inverse)
- Index à **adressage** direct ou indirection sur l'index primaire
  - ✓ temps mise à jour vs. temps d'accès
- **Sélection / restriction** : combinaison d'index secondaires
  - ✓ travail sur les index séparés puis intersection
  - ✓ travail sur un index en premier
  - ✓ index multi-attributs


```
SELECT nom, prenom  
FROM base_insee  
WHERE ville='Blois'  
AND sociopro ='ouvrier';
```



# Index secondaire

Index secondaire hiérarchique (arbre B+)



 Plusieurs pointeurs pour une valeur de clé secondaire  
Adressage direct ou sur index primaire

Index secondaire haché : pas de modification (plusieurs adresses par page)

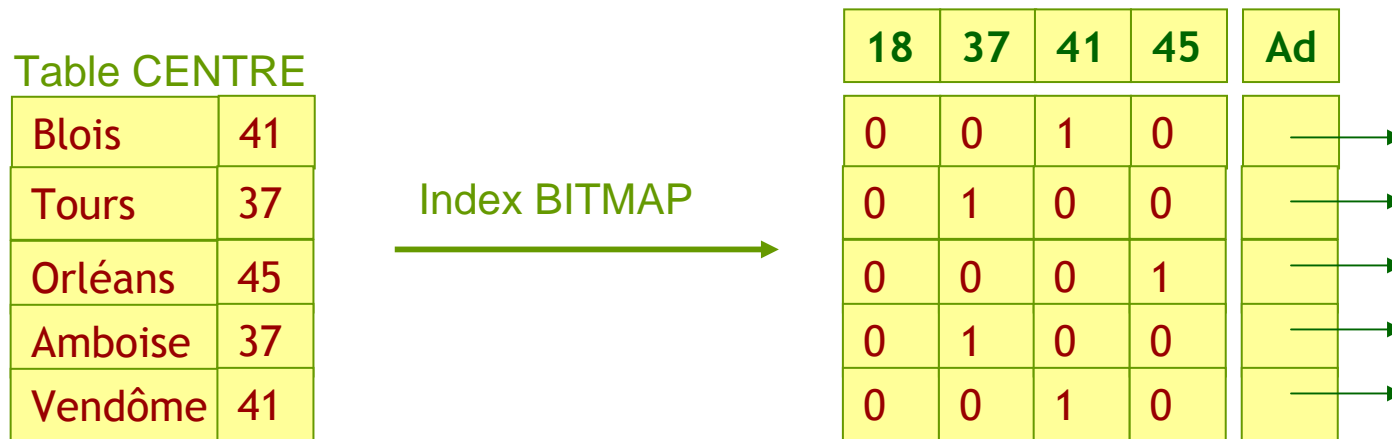


# Index bitmap

Index sur champ ayant peu de valeurs

[O'Neil 1997]

Table de bijection entre toutes les valeurs possibles et les tuples



## Utilisation

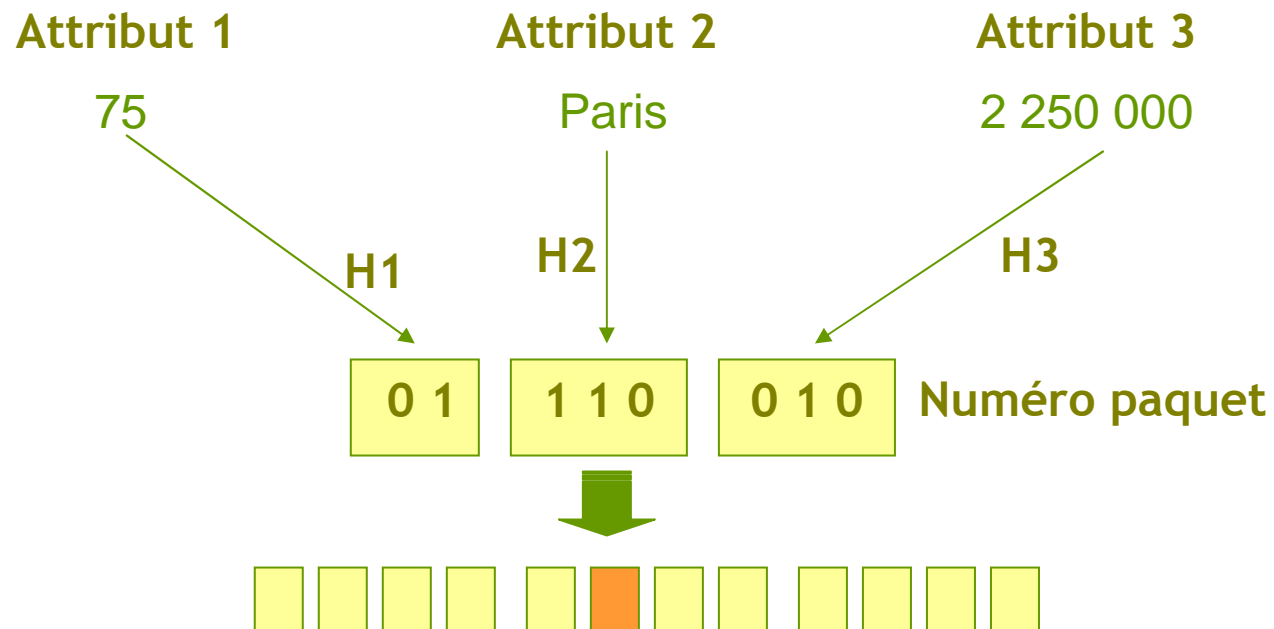
- Champ à domaine de valeurs restreintes
- Indexation de champs à **valeurs continues** : plages de valeurs
- Très efficace pour l'accès à des attributs **multiples** ou des valeurs multiples d'un même attribut : union ou intersection des vecteurs de bits



# Hachage multi-attributs

## Hachage multi-attribut statique

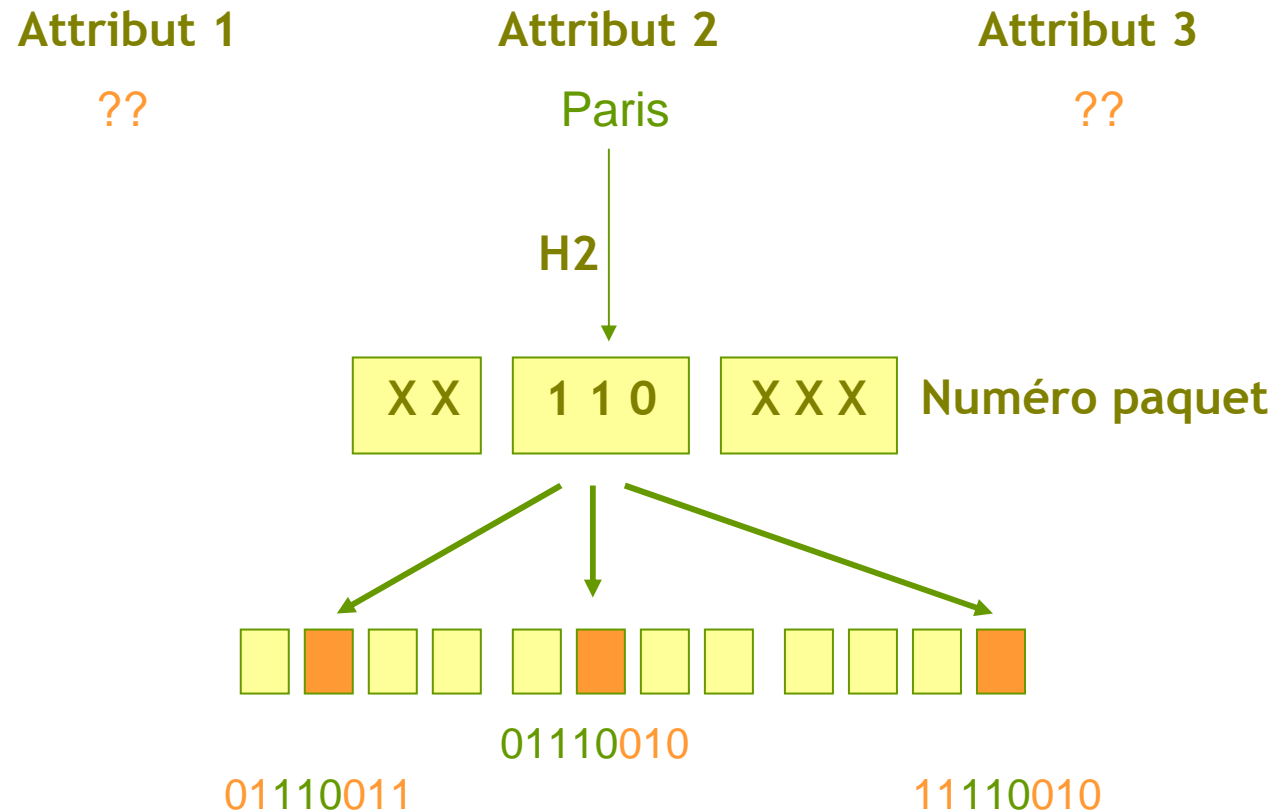
- Concaténation de fonctions de hachage travaillant sur chaque attribut
- Nombre fixe de bits alloués à chaque fonction : indexation statique



# Hachage multi-attributs

Hachage multi-attribut statique : accès sur un seul attribut

```
SELECT * FROM Table_Ville VERS Ville = 'Paris';
```

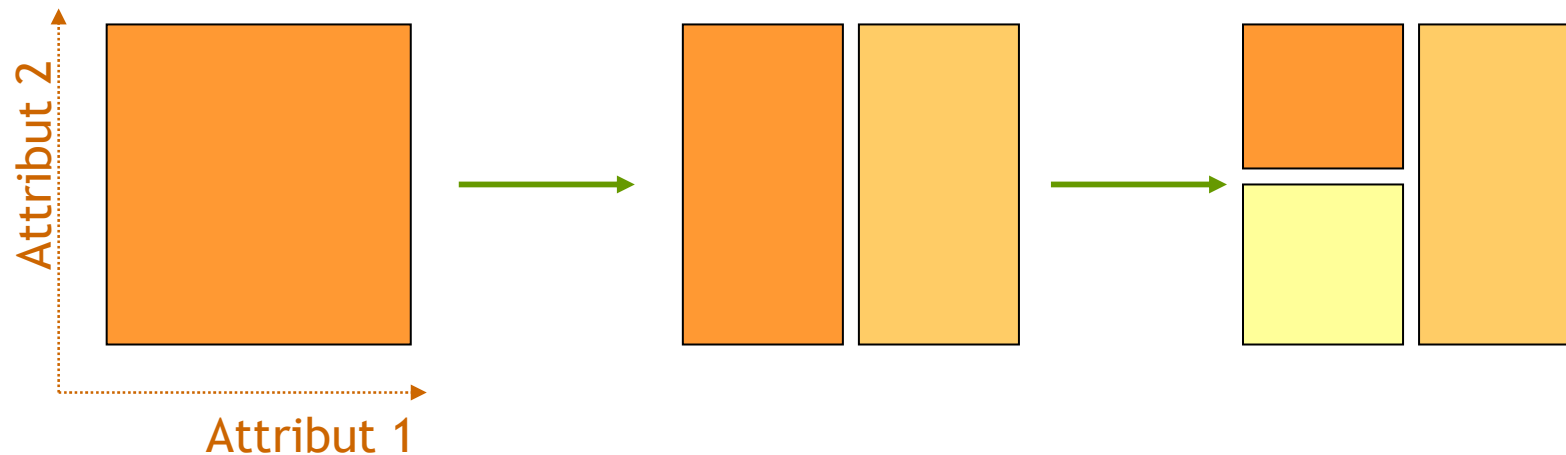




# Hachage multi-attributs

## Hachage multi-attribut dynamique

- Hachage extensible sur plusieurs dimensions (une par attribut) : éclatement successifs suivant une dimension puis une autre...
- Répertoire multidimensionnel



Problème : choix de la dimension d'éclatement



# Indexation : conclusion

---

## Exemple

```
SELECT nom, prenom FROM base_insee WHERE n_insee='06612....';  
SELECT * FROM base_insee WHERE ville='Blois' AND sociopro='ouvrier';
```

**insee**

50 000 000  
tuples

## Intérêt et limitations

- Accès très efficace aux données pour des clés discriminantes
- Mise à jour ralentie : calcul arbre B, éclatement répertoire de hachage..
- Différentes techniques : adaptation à chaque type d'attribut c

## Comparaison des méthodes d'indexation

- **Hachage** : efficace en accès mais pas adapté pour les accès ordonnées ou par plages de valeurs, à la différence des index triés (arbres B)
- **Hachage extensible** plus robuste que le **hachage linéaire** face à des distribution de données mal équilibrées
- Le répertoire utilisé en hachage extensible peut devenir volumineux (hachage linéaire : simple pointeur)



# Indexation : SQL

---

**Index simple** (attribut unique) ou **composé** (plusieurs attributs)

**Oracle 8i** : 16 attributs maximum dans un index composé

**Oracle 10g** : 32 attributs maximum dans un index composé

**Index sur table** ou sur plusieurs tables réunies par un **cluster**

Possibilité de définir des **index multiples** sur un objet : Oracle choisira le meilleur index pour une requête donnée

**Index unique** : un tuple unique par valeur d'index

**Remarque**      contrainte d'unicité définie sur les attributs correspondant à l'index

**Indexation implicite / explicite**

- Indexation automatique des attributs clés ou soumis à une contrainte d'unicité
- Indexation explicite (création d'index) possible dans les autres cas



# Indexation : SQL

---

## Création (syntaxe simplifiée)

```
CREATE INDEX [UNIQUE] [CLUSTERED] index  
ON nom_table [ USING {btree | hash} ]  
[ ( nom_colonne [ ASC | DESC ], ... ) ] [ { IN | ON } nom_dbSPACE ] ;
```

## Création (syntaxe simplifiée : Oracle)

```
CREATE INDEX [UNIQUE | BITMAP] index  
ON [ nom_table ( nom_colonne, ... )  
[ NOSORT ] ;
```

```
CREATE INDEX index  
ON CLUSTER nom_cluster  
[ NOSORT ] ;
```

## Suppression

```
DROP INDEX <nom_index> ;
```

## Table indexée

Association implicite avec un index basé sur la clé primaire

Définition : clause **ORGANIZATION INDEX** en fin de **CREATE TABLE**;



# Indexation : SQL

---

## Effets de l'indexation

- Temps d'accès aux données réduit si la requête porte sur le(s) attribut(s) indexé(s)
- Accélération fonction de la taille de la table (ou du cluster) concerné
- Ralentissement des mises à jour (re-calculation d'index)

## Optimisation de l'indexation sur les critères de sélection

- Accès accéléré uniquement sur comparaison des valeurs d'index à des valeurs  
... WHERE parti >= 0 et non pas .. WHERE parti IS NOT NULL
- Pas d'accélération si calcul sur un attribut indexé  
WHERE date\_S = to\_date(' 01/01/03 ') et non WHERE to\_char(date\_S) = ' 01/01/03 '
- Dans certains SGBD, possibilité de créer des champs calculés indexés pour précisément gérer ce type de requête



# Jointure physique : Cluster

## Problème

Temps perdu à vérifier les conditions de jointure lors d'une requête multi-tables

## Cluster

Regroupement physique des tables autour des clés de jointure : **clé de cluster**

Utilisation limitée aux jointures très fréquemment sollicitées

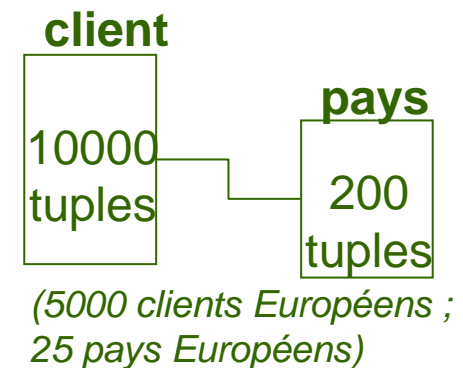
## Exemple

```
SELECT c.nom FROM client c, pays p
WHERE p.pays = c.pays
AND p.continent = 'Europe';
```

- **Oracle : type de cluster**

**Cluster à index**                      indexation sur la clé du cluster

**Cluster à hachage**                      fonction de hachage appliquée à la clé du cluster



# Cluster

---

## Création (syntaxe simplifiée)

```
CREATE CLUSTER nom_cluster  
    (<nom_cle1 type_cle1> [, <nom_clé type_cle2...> ] )  
[SIZE taille_bloc]  
{ INDEX | HASHKEYS integer } ;
```

## Suppression

```
DROP CLUSTER <nom_cluster>  
[INCLUDING TABLES [CASCADE CONSTRAINTS]]
```

- **INCLUDING TABLES** suppression des tables liées au cluster (DROP TABLE préalable sinon)
- **CASCADE CONSTRAINTS** suppression des contraintes d'intégrité référentielles pointant sur les tables du cluster



# Optimisation

---

## Optimisation

- Temps d'accès : indexation
- Calcul relationnel : optimisation **automatique** des ordres SQL

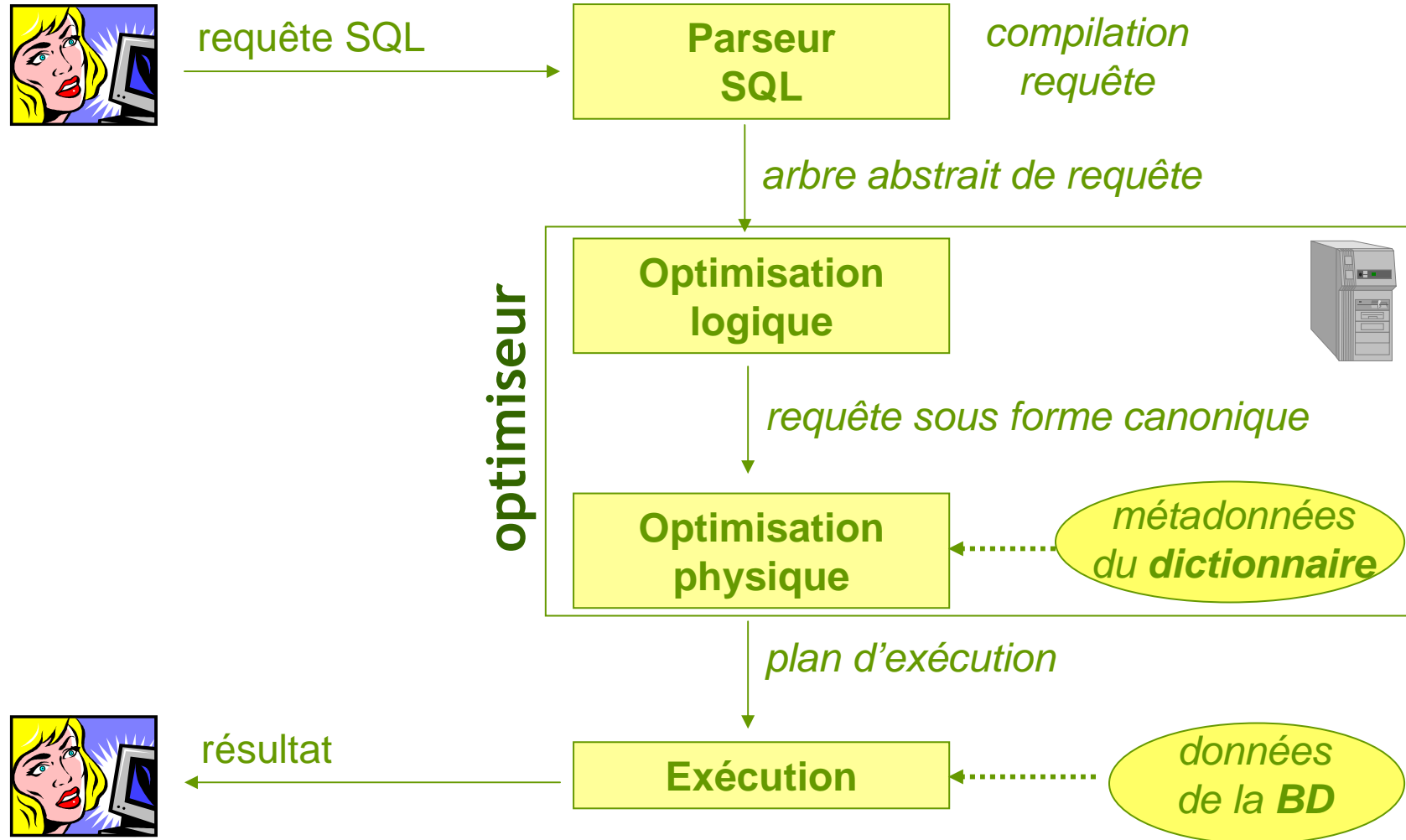
## SGBD : module d'optimisation automatique

- Plan d'exécution : SQL langage sémantique de haut niveau
- Optimisation dynamique : interprétation





# Optimisation

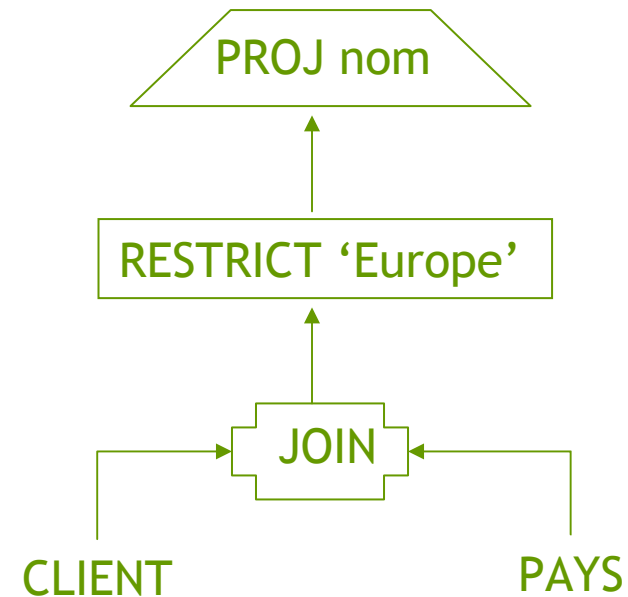


# Parseur SQL

## Arbre abstrait de requête

```
SELECT c.nom FROM client c, pays p  
WHERE p.pays = c.pays  
AND p.continent = 'Europe';
```

→ Algèbre relationnelle



## Vues

Intégration de la définition des vues



# Optimisation logique

## Exemple

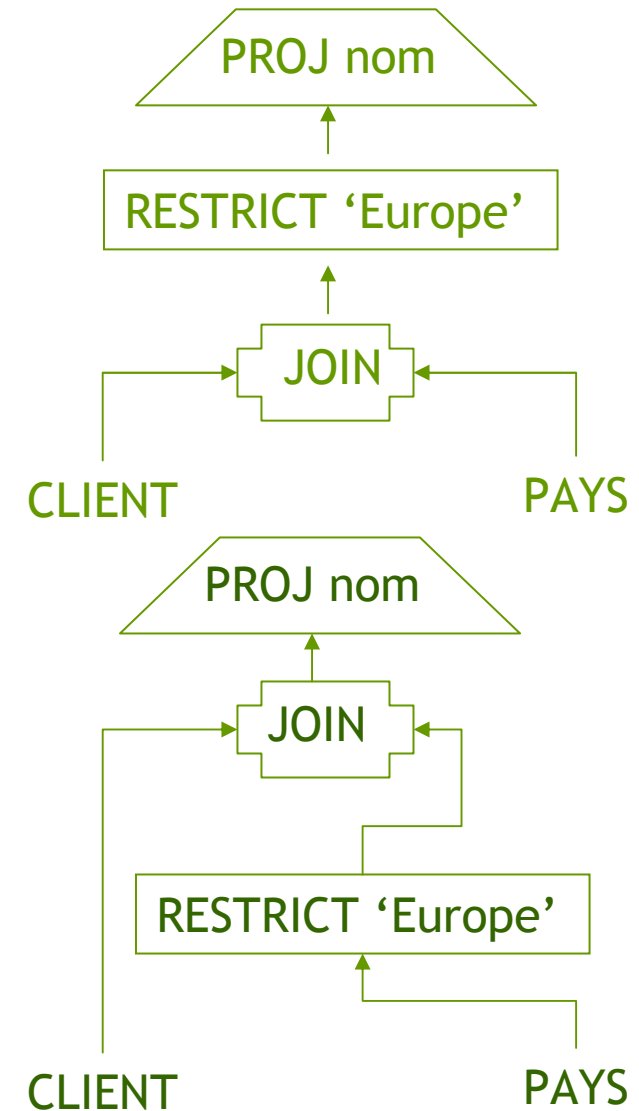
```
SELECT c.nom FROM client c, pays p
WHERE p.pays = c.pays
AND p.continent = 'Europe';
```

- plusieurs écritures SQL possibles
- plusieurs arbres de requête possibles

## Mise sous forme canonique de la requête

- Optimisation indépendante de la structure et du contenu de la BD
- Ré-ordonnancement de l'arbre de requête (ordre des opérations) pour arriver à une forme canonique efficace

## ↳ Règles de transformation de requête



# Optimisation logique : transformations

## Restrictions et projections

- Regroupement des restrictions

$$- \Pi_{\text{attribut}} (\sigma_{\text{condition}}(\text{Table})) \equiv \sigma_{\text{condition}} (\Pi_{\text{attribut}} (\text{Table}))$$

Si condition porte  
sur attributs



## Distributivité de la restriction

- $\sigma$  distributive sur  $\cup, \cap, \text{DIFF}$
- $\sigma$  (quasi) distributive sur  $\text{JOIN}$

$$\text{Exemple : } \text{JOIN} (T1, \sigma_{\text{condition}}(T2)) \equiv \sigma_{\text{condition}} (\text{JOIN} (T1, T2))$$



# Optimisation logique : transformations

---

## Distributivité de la projection

- $\Pi$  distributif sur  $\cup, \cap$  mais pas sur la différence DIFF
- $\Pi$  distributif sur JOIN

**Exemple** :  $\Pi_C (\text{JOIN} (T1, T2)) \equiv \text{JOIN} (\Pi_{C \rightarrow T1} (T1), \Pi_{C \rightarrow T2} (T2))$



## Commutativité et associativité

- $\cup, \cap, \text{JOIN}$  sont commutatives et associatives
- DIFF et DIV non commutatives, non associatives

**Exemple** : choix de la plus petite table comme relation source de jointure



# Optimisation logique : transformations

## Expressions arithmétiques et logiques

- transitivité, distributivité, commutativité des opérateurs arithmétiques ou booléens

**Exemple :**

$\sigma_{A > B \text{ AND } B > 3} \equiv$



$\sigma_{A > 3 \text{ AND } B > 3 \text{ AND } A > B}$



**Exemple :**

$\sigma_{Cd1 \text{ OR } (Cd2 \text{ AND } Cd3)} \equiv \sigma_{(Cd1 \text{ OR } Cd2) \text{ AND } (Cd1 \text{ OR } Cd3)}$



# Optimisation logique

## Stratégie générale d'optimisation

1. Descendre les restrictions aussi bas que possible (pour les exécuter en premier) et regrouper les restrictions portant sur la même table
2. Descendre ensuite les projections et les regrouper également
3. Réaliser les unions
4. Réaliser les jointures

op.  
unaires  
op.  
binaires

## Détachement

Décomposition de la requête en restriction suivies de semi-jointures

### Exemple

```
SELECT c.nom FROM client c, pays p  
WHERE p.pays = c.pays  
AND p.continent = 'Europe';
```



```
SELECT nom FROM clients WHERE pays IN  
(SELECT pays FROM pays  
WHERE continent = 'Europe');
```



# Optimisation physique

---

## Objectif

- Étude de la structure de la base de données (implantation physique)
- Choix des procédures de niveau inférieur les plus adaptées pour la réalisation de chaque opérateur : plusieurs méthodes pour un même opérateur

**Exemple** : restriction

- Fonction de coût pour chaque opération
  - Accès mémoire
  - Utilisation du processeur

- Coût : interdépendances des opérations

**Exemple** : `SELECT DISTINCT...`

- Calcul du meilleur chemin d'exécution :
  - Plan de requête minimisant le coût global de réalisation de la requête
  - Espace de recherche trop important : heuristiques





# Administration : mesures de performances

---

## Dictionnaire Oracle

**Index**            USER\_INDEXES , ALL\_INDEXES, DBA\_INDEXES

**Clusters**        USER\_CLUSTERS, ALL\_CLUSTERS, DBA\_CLUSTERS  
USER\_CLUSTERS\_HASH\_EXPRESSIONS, etc...

## Vues dynamiques de performance

V\$SYSSTAT, V\$ROWCACHE, V\$LIBRARYCACHE

## Mesure de la durée des ordres SQL

```
SQL> set timing on
```

```
SQL> votre commande SQL
```

```
SQL> set timing off
```



# Bibliographie

---

## Ouvrages d'entrée

- G. GARDARIN, *Bases de données objet et relationnel*, Eyrolles, Paris. Chapitres III et X
- DATE C. J. (2000) *Introduction aux bases de données* (7<sup>o</sup> édition), Vuibert, Paris, ISBN 2-7117-8664-1. Chapitre XVII

## Travaux cités

- R. FAGIN *et al.* (1979) Extensible hashing : a fast access method for dynamic files. *ACM TODS*, 4(3), 315:344.
- W. LITWIN (1980) Linear hashing : a now tool for file and table adresssing. *6th Very Large Data Bases Conference*, Montréal, Canada, 517:523.

## Oracle 8i

- R. CHAPPUIS (2001) *Les bases de données Oracle 8i : développement, administration, optimisation*, Dunod, Paris, ISBN 2-10-005330-2
- G. BRIARD (2000) *Oracle 8i sous Linux*, Eyrolles, Paris, ISBN 2-212-09135-4

## Oracle 9i

- DELEGLISE D. (2004) *Guide du Développeur Oracle*, Supinfo, ISBN 2-914835-00-0
- G. BRIARD (2001) *Oracle 9i sous Linux*, Eyrolles, Paris, ISBN 2-212-11026-X

