École Doctorale 2008/2009

Systèmes de Bases de Données Avancés

2. Traitements et Stockage

École nationale Supérieure d'Informatique

Plan

1. Organisations physiques
   - Supports de stockage et types d'index
2. Opérations de base
   - Sélection, Tri et Jointures
3. Traitement des requêtes

2. Traitements et Stockage

2.1. Supports de Stockage

2.1 Organisations physiques

- La hiérarchie des supports de stockage -

- Le cache (processeur – RAM)
- Mémoire centrale
- EEPROM (mémoire Flash)
- Disques magnétiques
- Disques optiques
- Bandes magnétiques

2.1 Organisations physiques

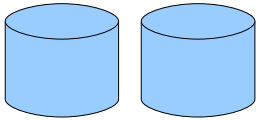
- structure des disques magnétiques -

- Composants physiques
  - plateaux (surfaces de disques)
  - bras (têtes de lecture/écritures)
  - contrôleur
- Critères de performances
  - temps d'accès
  - temps de déplacement (du bras)
  - temps de rotation
  - taux de transfert

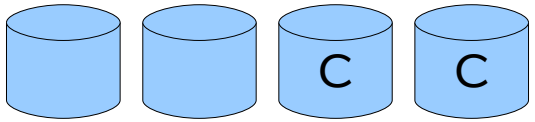
2.1 Organisations physiques

- RAID -

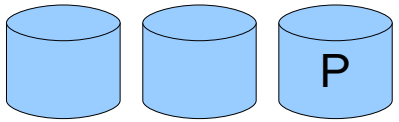
« Redundant Array of Independent Disks »



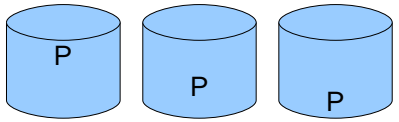
raid 0 : pas de redondance



raid 1 : avec disques miroirs



raid 2,3,4 : disques de parité

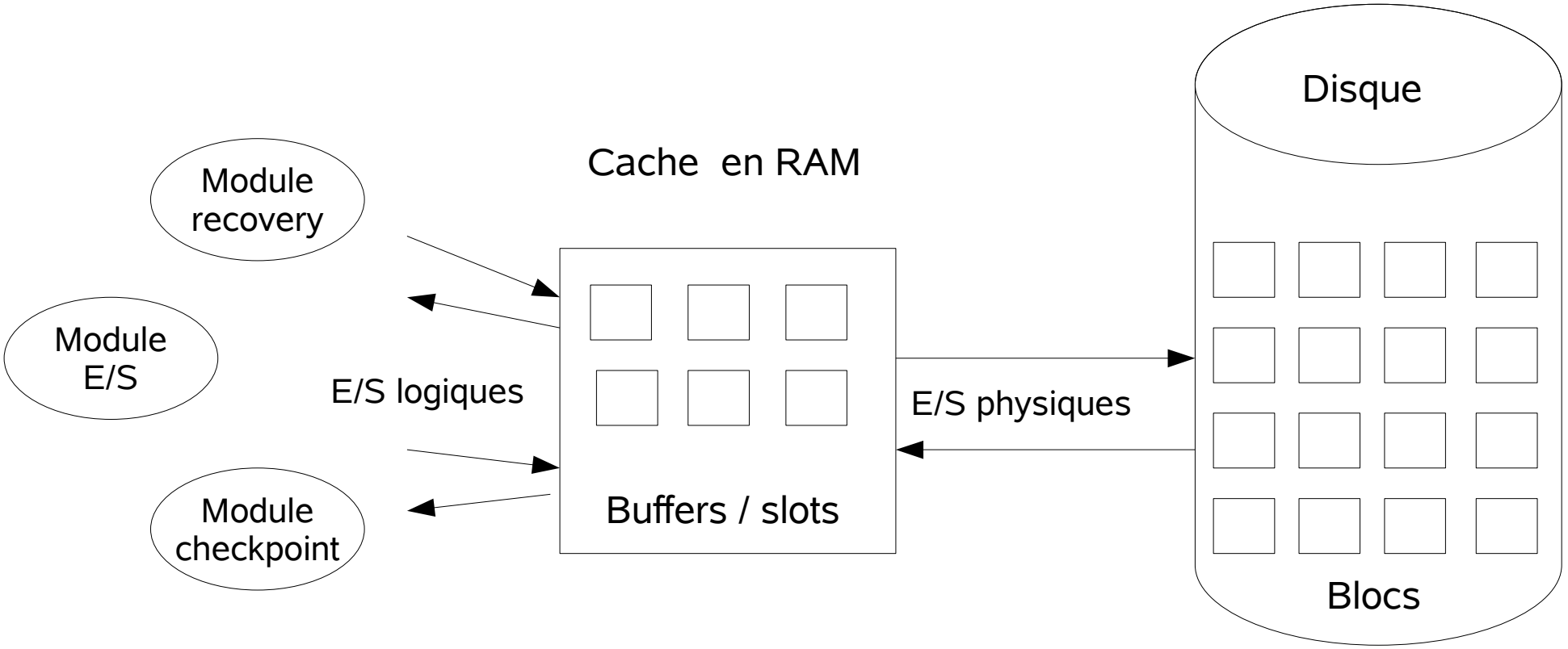


raid 5 et 6 : info de parité répartie

- Arrangement de plusieurs disques donnant l'abstraction d'un unique, grand disque.
- Buts:
  - Augmenter les performances et la disponibilité.
- Deux techniques:
  - « **Data striping** »: Les données sont partitionnées et distribuées sur plusieurs disques.
  - **Redondance** : (Plus de disques →Plus de pannes.) Une information redondante permet la reconstruction de données en cas de panne.

2.1 Organisations physiques

- Le gestionnaire du cache -



Chargement d'un bloc b:

si le bloc b est dans le cache, retourner son numéro de slot.
sinon s'il y a un slot libre dans le cache, chargé le bloc du disque
sinon choisir un slot à vider (l'écrire sur disque) et le remplacer par b

Stratégies de remplacement:

LRU, FIFO, ...

2.1 Organisations physiques

- structures de fichiers -

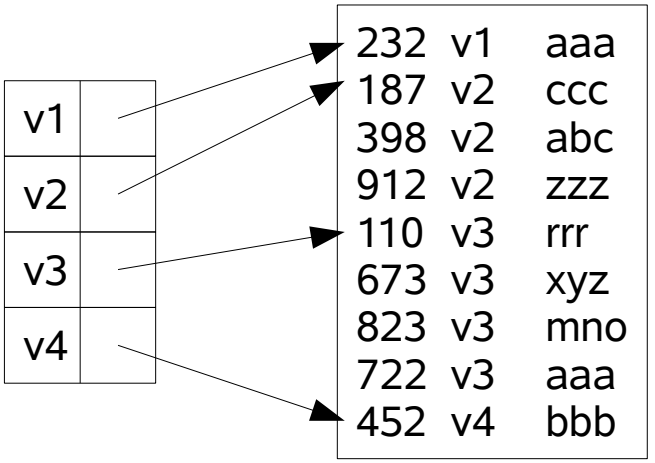
- Fichier = ensemble de blocs
- chaque bloc renferme un ensemble d'enregistrements
- un enregistrement peut être de taille fixe ou variable (longueur de champs, nombre de champs)
- organisation de fichier
  - non ordonné (heap file)
  - ordonné (sur un ou + attributs) : fichier séquentiel
- Clustering
  - rassembler dans le même bloc, les enreg susceptibles d'être traités ensembles.

2.1 Organisations physiques

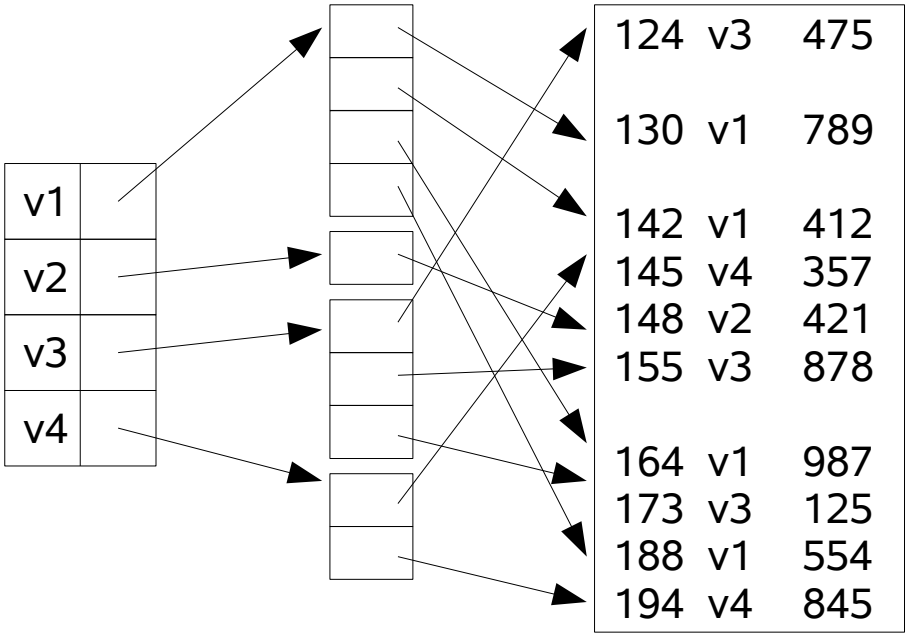
- Index -

(maintient l'ordre des enreg suivant un ou + attr)

- structuré en tables ou arbres (B-Arbres)
- Primaire (dense ou non dense)
  - si le fichier est ordonné suivant l'attribut indexé
- Secondaire (dense)
  - si le fichier n'est pas ordonné



index primaire dense (contient toutes les valeurs)



index secondaire

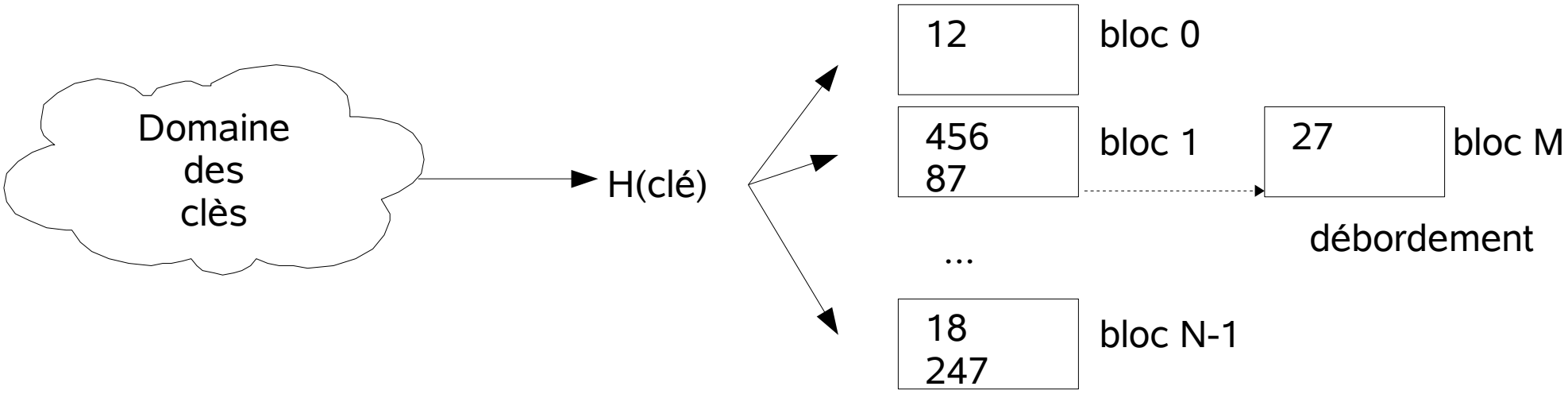
2.1 Organisations physiques

- Hachage -

utilisation d'une fonction H pour le calcul du n° de bloc

propriétés souhaitables:

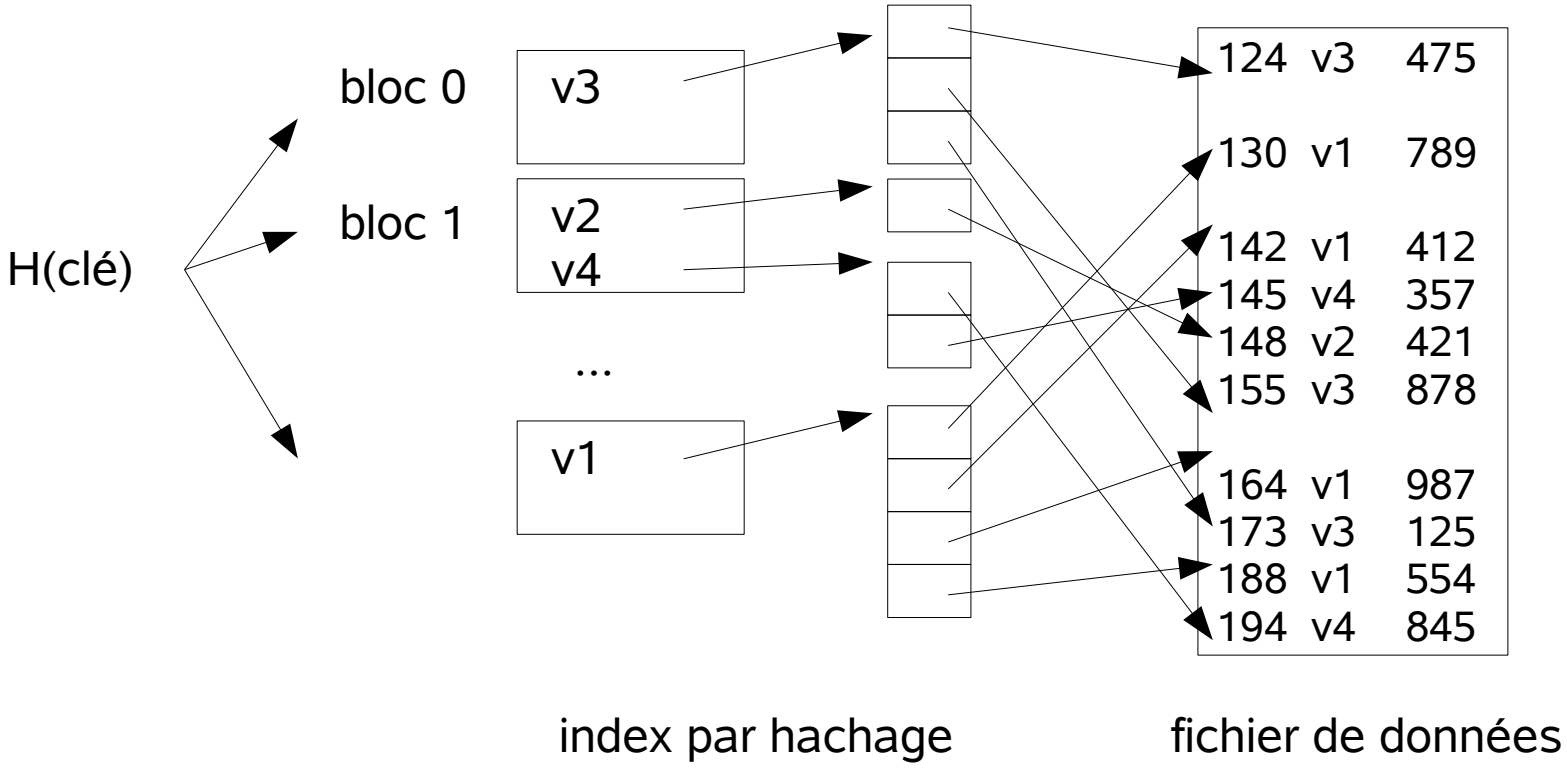
- distribution uniforme : la fct H affecte le même nb de données à chaque bloc, si l'ensemble des valeurs du domaine était utilisées
- H apparait aléatoire : pas de corrélation « visible » entre les n° de blocs et les clés.



2.1 Organisations physiques

- Hachage comme index -

- On peut construire une table de hachage jouant le rôle d'index sur un fichier de données existant



2.1 Organisations physiques

- Autres types d'index -

- Hachage dynamique
  - la fonction de hachage H évolue (change dynamiquement) avec la taille du fichier de données
- Accès Multi-clés
  - la structure d'index est organisée autour de plusieurs attributs
    - R-Trees, Grid Files, Hachage par interpolation, ...
- Bitmaps
  - pour des attributs ayant peu de valeurs
    - pour chaque valeur, on a une chaine de bits (de longueur le nb d'enreg de la relation) indiquant les enreg ayant cette valeur.

ex: attr booleen V | 0 | 1 | 0 | 0 | 1 | 1 | F | 1 | 0 | 1 | 1 | 0 | 0 |

2.2 Opérations de base

2.2 Opérations de base

- Sélection
  - avec et sans index
- Tri
  - « par fusion »
- Jointure
  - boucles imbriquées
  - « par fusion »
  - « par hachage »
- Autres opérateurs (intersection, différence, ...)
  - Utilisent des algorithmes similaires à ceux de la jointure

2.2 Opérations de base

Sélection avec conditions simples (attribut <opérateur> valeur)

- Séquentiel : Cas général
- Dichotomie : Fichier trié + condition d'égalité
- Avec Index Primaire :
  - Cond d'égalité sur clé
  - Cond d'égalité sur non clé
  - Cond >, >=
- Avec Index Secondaire (en cas de faible sélectivité)
  - Cond d'égalité
  - Cond >, >=, <, <=

2.2 Opérations de base

Sélection avec conditions complexes

- Formes complexes
  - $cond_1$ AND $cond_2$ AND ... $cond_n$
  - $cond_1$ OR $cond_2$ OR ... $cond_n$
  - NOT (cond)
    - not x < v : x >= v
    - not x = v : x < v OR x > v
    - ...etc
- Conjonctive avec un seul index (sur une des condition)
- Conjonctive avec un index multi-clés (sur une ou + condition '=')
- Conjonctive avec intersection (utilisation de plusieurs index)
- Disjonctive avec union (s'il y a des index pour chaque condition)

2.2 Opérations de base

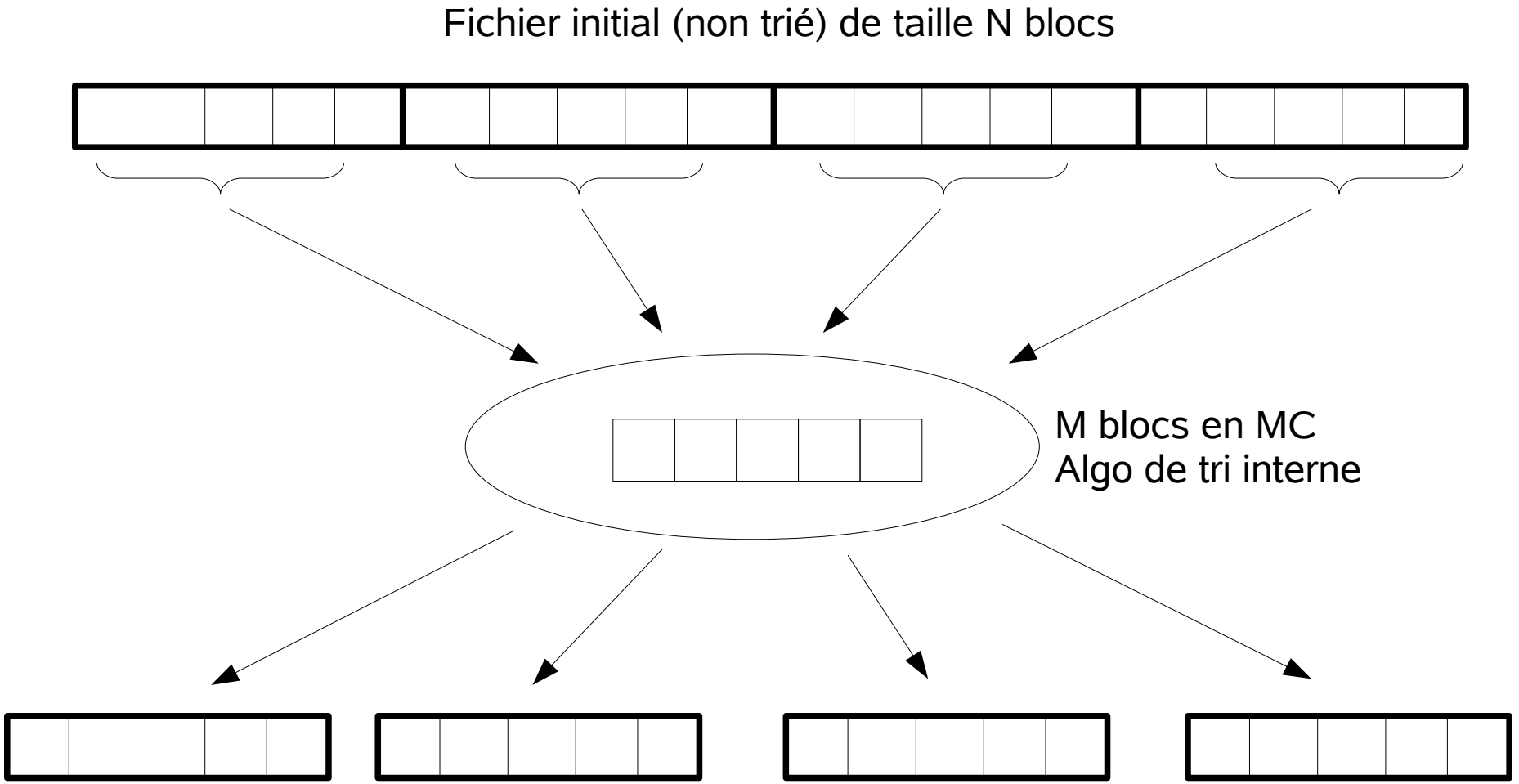
Tri externe « par fusion »

- Utilisé pour :
  - Algorithmes de jointures
  - Elimination des doubles (clause 'distinct')
  - Groupement ('Group By')
  - Filtre 'Order By'

- Soit M le nombre de buffers disponibles en MC
- L'algorithme se déroule en 2 principales étapes:
  - Construction de fragments initiaux triés
  - Fusion des fragments

Tri : 1ere Phase - construction des fragments -

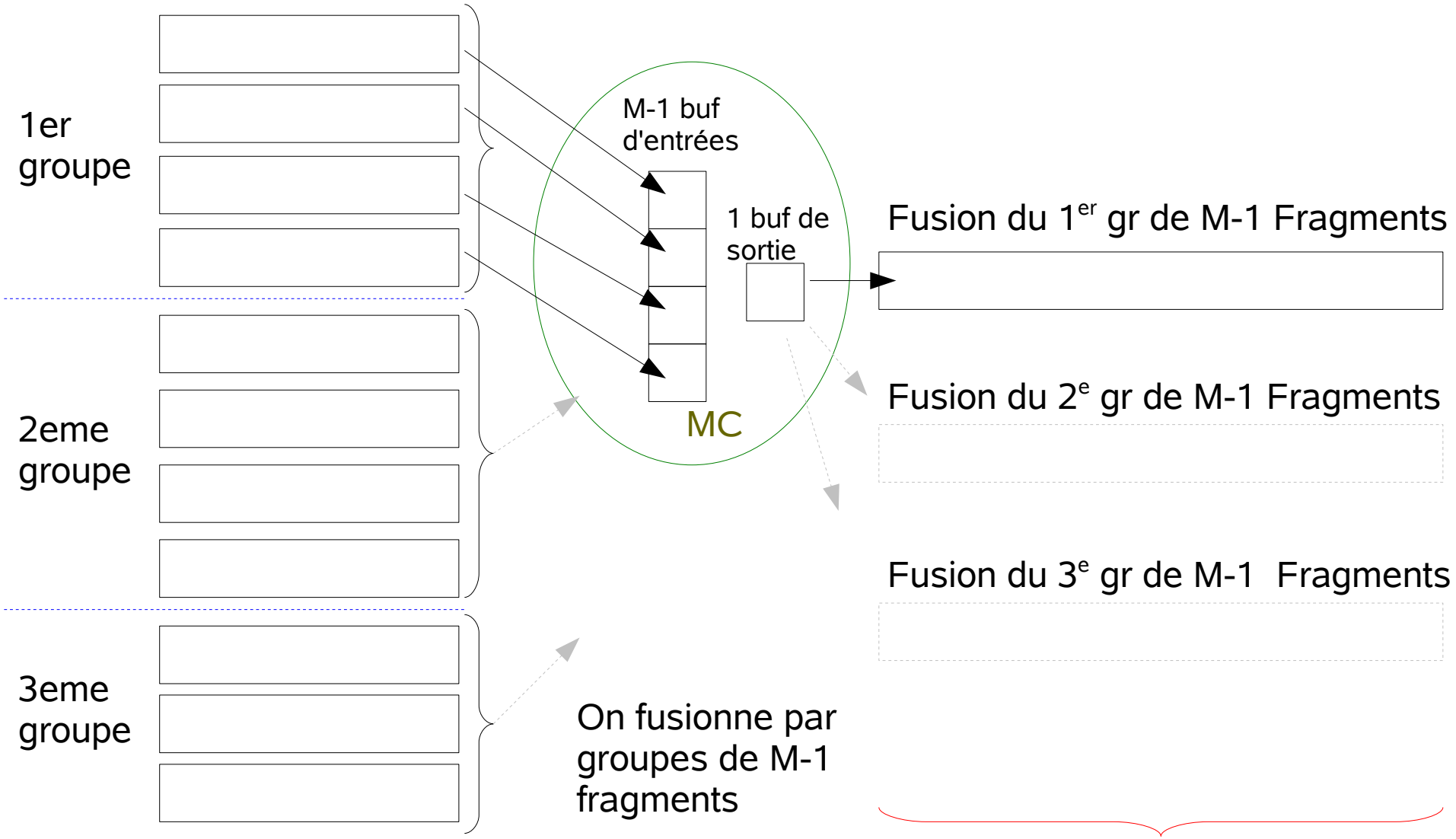
coût pour un fichier de N blocs : N lectures + N écritures (2N accès)



[N/M] Fragments initiaux triés, de taille M blocs chacun

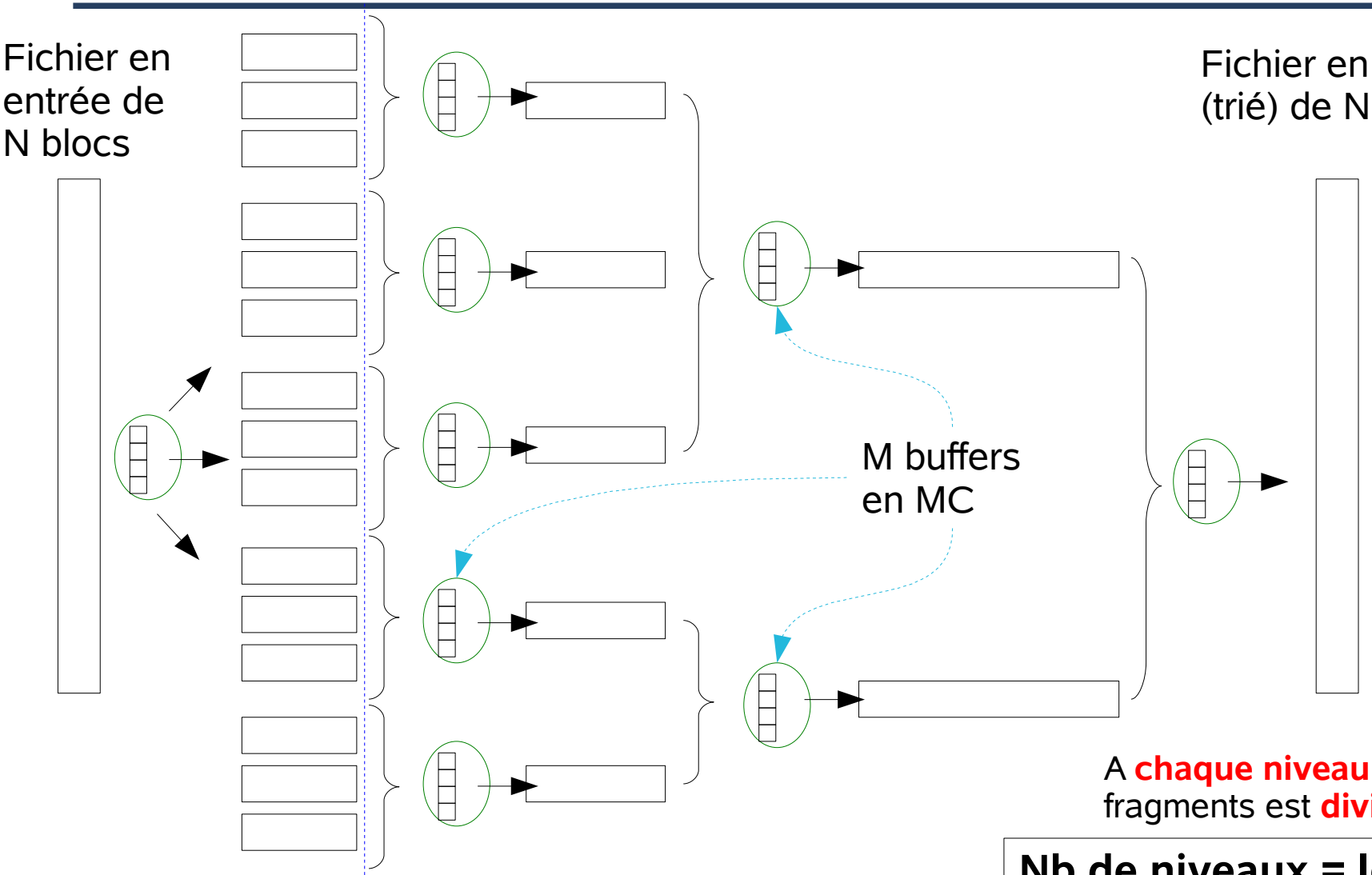
Tri : 2eme Phase - Fusion des fragments -

coût : lire et écrire N blocs autant de fois que de niveaux de fusions



Tri externe : Résumé

Coût total = Fragmentation + Fusions = $2N + 2N \log_{M-1}[N/M]$



A **chaque niveau** le nb de fragments est **divisé par M-1**

Nb de niveaux = $\log_{M-1}[N/M]$

à chaque niv, il faut 2N accès

2.2 Opérations de base

Algorithmes de jointure

- Par boucles imbriquées (Nested-loop join)
  - cas général
- Jointure par fusion (Merge join)
  - équi-jointure
- Jointure par hachage (Hash join)
  - équi-jointure
- Jointure par index (Indexed Nested-loop join)
  - équi-jointure

Jointure par boucles imbriquées

R (Relation externe : N_R blocs) x S (Relation interne : N_S blocs – doit être Stockée)

- Utiliser le **max d'espace** mémoire pour la **relation externe** (la plus petite)
  M-2 buffers pour R, 1 buffer pour S et 1 buffer écrire le résultat.
- Lire **R** par fragments de **M-2 blocs**. Pour chaque fragment de **R**, lire **S**

Pour chaque **fragment Fr** de **R** (lecture de M-2 blocs de **R**)

Pour chaque **bloc Bs** de **S** (lecture d'un bloc de **S**)

Pour chaque tuple tr dans Fr

Pour chaque tuple ts dans Bs

Si la condition de jointure est vérifiée entre tr et ts
rajouter la concaténation tr.ts au résultats B
Si B est plein, le vider sur disque

Fr x Bs
en MC

- Coût (sans compter l'écriture du résultat) :

S est lue $[N_R/(M-2)]$ fois. **R** est lue une seule fois

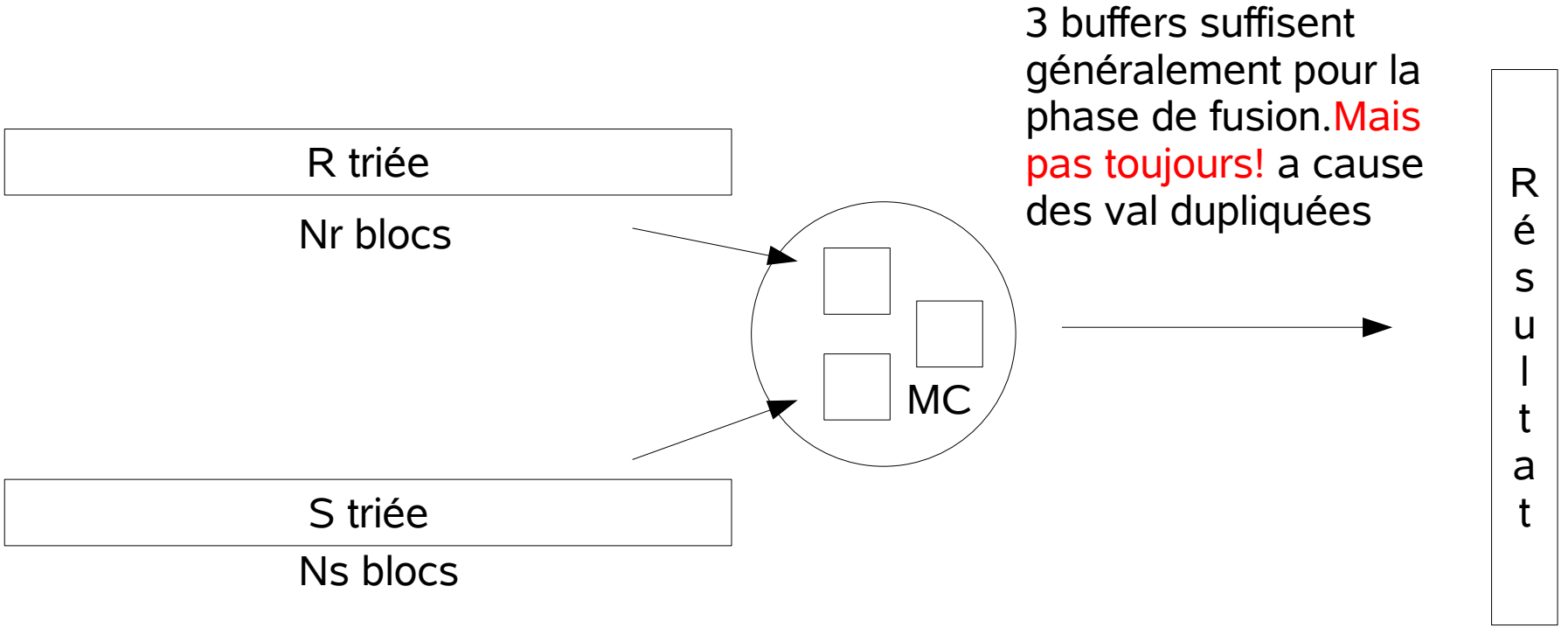
= $\mathbf{N_R + N_S\ [N_R/(M-2)]}$ accès disque

Jointure par fusion

Coût = 2 Tris + 1 fusion

$2Nr(1+\log_{M-1}[Nr/M]) + 2Ns(1+\log_{M-1}[Ns/M]) + Nr + Ns$

- En 2 étapes :
  - Trier les 2 relations R et S si elles ne le sont pas déjà
  - Parcourir les 2 relations en même temps (comme la fusion) pour calculer l'équi-jointure



Jointure par fusion

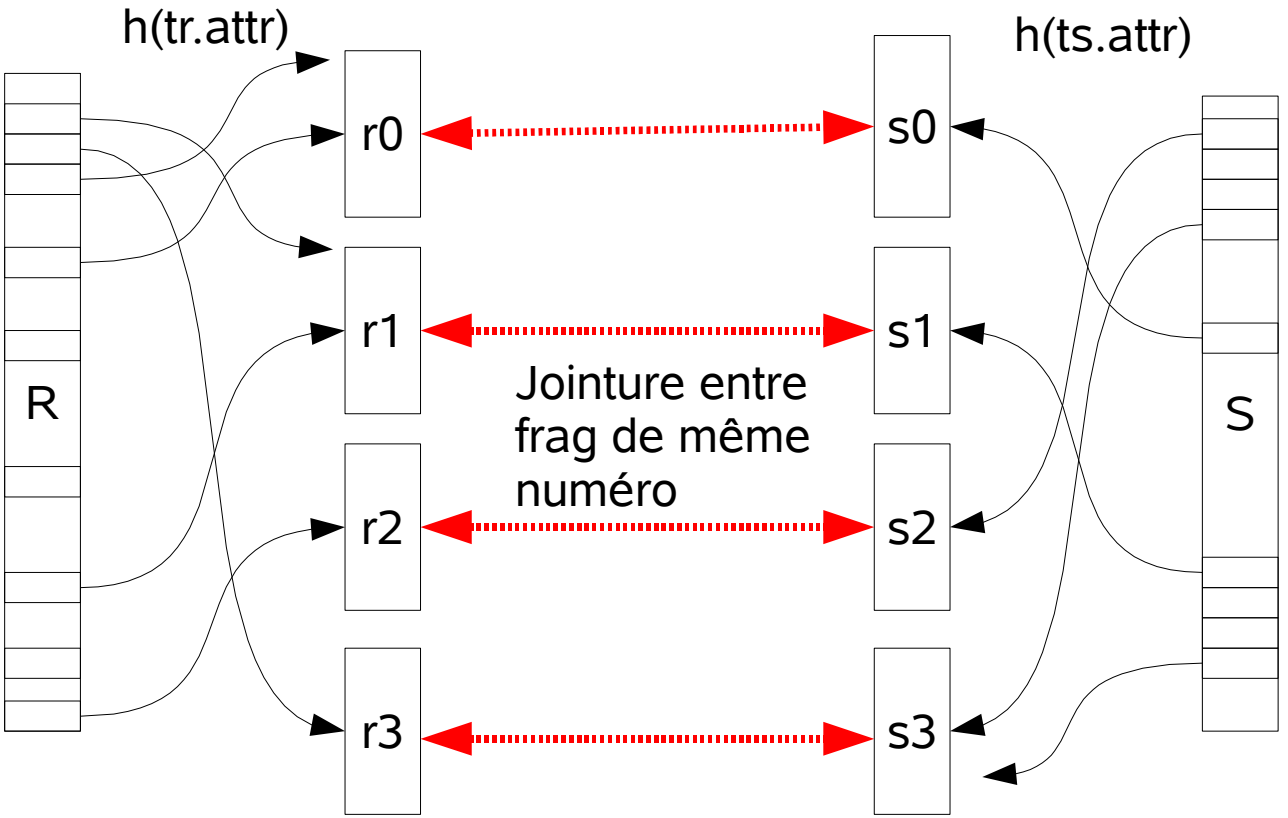
Algorithme : phase de fusion

```
Pr = adr du 1er tuple de R;  Ps = adr du 1er tuple de S;
TQ  ( Pr <> 0 && Ps <> 0 )
    ts = tuple(Ps); E = {ts}; Ps = suiv( Ps ); stop = faux;
    TQ ( non stop && Ps <> 0 )
        t = tuple(Ps);
        SI (t.attr == ts.attr)  E = E U {t}; Ps = suiv(Ps)  SINON  stop = vrai FSI;
    tr = tuple(Pr);
    TQ ( Pr <> 0 && tr.attr < ts.attr )
        Pr = suiv(Pr); tr = tuple(Pr);
    TQ ( Pr <> 0 && tr.attr == ts.attr )
        POUR chaque ts dans E
            ajouter <tr.ts> dans le résultat;
        Pr = suiv(Pr); tr = tuple(Pr);
```

Jointure par hachage

la table de hachage d'un fragment doit résider en MC

- L'idée est de partitionner R et S par la même fonction de hachage h, puis de calculer les jointures entre fragments équivalents



Jointure par hachage

pseudo algorithme

Partitionner **S** en s0, s1, ... s_{n-1};

Partitionner **R** en r0, r1, ... r_{n-1};

POUR i = 0,n-1

    /* Build */

    lire le frag '$\mathbf{s}_i$' et construire un index par hachage en MC (table de hachage)

    /* Probe */

    POUR chaque tuple tr dans le frag '$\mathbf{r}_i$'

        récupérer avec l'index en MC les tuples ts ayant la même val d'attr que tr

        POUR chaque ts récupéré

            rajouter <ts.tr> dans le résultat

Estimation du coût : **3 (Nr + Ns) + 4n**

2(Nr+Ns) : partitionnement de S et R et Nr+Ns+4n pour le 'build' et le 'probe'

4n estimation des débordements (2 blocs supplémentaires pour chaque fragment)

Jointure avec index

- index sur S -

- Boucles imbriquées
  - avec utilisation de l'index au lieu du parcours de la table interne (S)
  - Pour chaque tuple tr de R
    - Utiliser l'index sur S pour rechercher les tuples ts vérifiant (ts.attr==tr.attr)
- Coût
  - un parcours de R : Nr
  - une recherche dans l'index pour chaque tuple : n * c
  - Total = **Nr + n * c**

Nr : nb de blocs de R, **n** : nb de tuples dans R, **c** : coût d'une selection dans S

Jointure Complexe

- Condition de jointure quelconque
  - par « Boucles imbriquées »
- Condition conjonctive : $R \ x_{c1 \text{ et } c2 \text{ et } ...cn} \ S$
  - On calcule la jointure R x S avec une condition ci
  - les autres conditions cj seront vérifiées juste avant l'écriture de <tr.ts> dans le résultat.
- Condition disjonctive : $R \ x_{c1 \text{ ou } c2 \text{ ou...cn}} \ S$
  - prendre l'union des résultats des n jointures :
  - $R \ x_{c1} S \ U \ R \ x_{c2} S \ U \ ... \ R \ x_{cn} S$

2.3. Traitement des requêtes

2.3 Traitement des requêtes

- Les 3 phases du traitement
  - Traduction : SQL ==> Expression Algébrique
    (Arbre d'opérateurs)
  - Optimisations (heuristiques - logique / coûts - physique)
    - Transformation d'expressions algébriques
    - Choix des algorithmes pour les opérateurs de l'exp
    - Choix des types de connections entre opérateurs (matérialisation des résultats temporaires / pipeline)
  - Évaluation du plan global d'exécution choisi

Traduction SQL ==> Exp. Algébrique

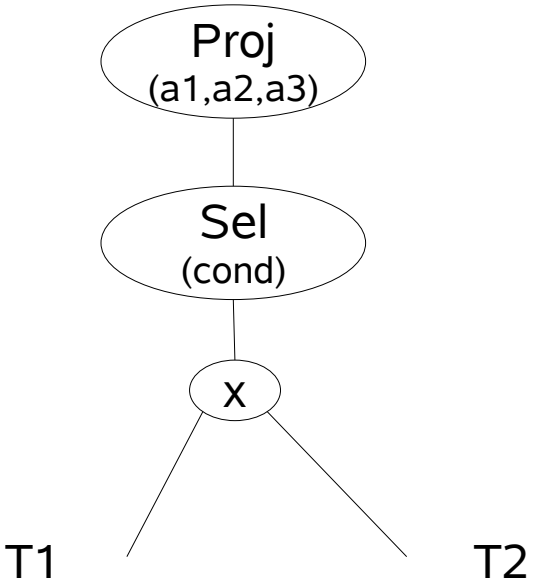
Chaque bloc SELECT est traduit et optimisé indépendamment des autres blocs ==> Projection(... Selection(... Produit-cartésien))

- Minimise les choix pour l'optimiseur

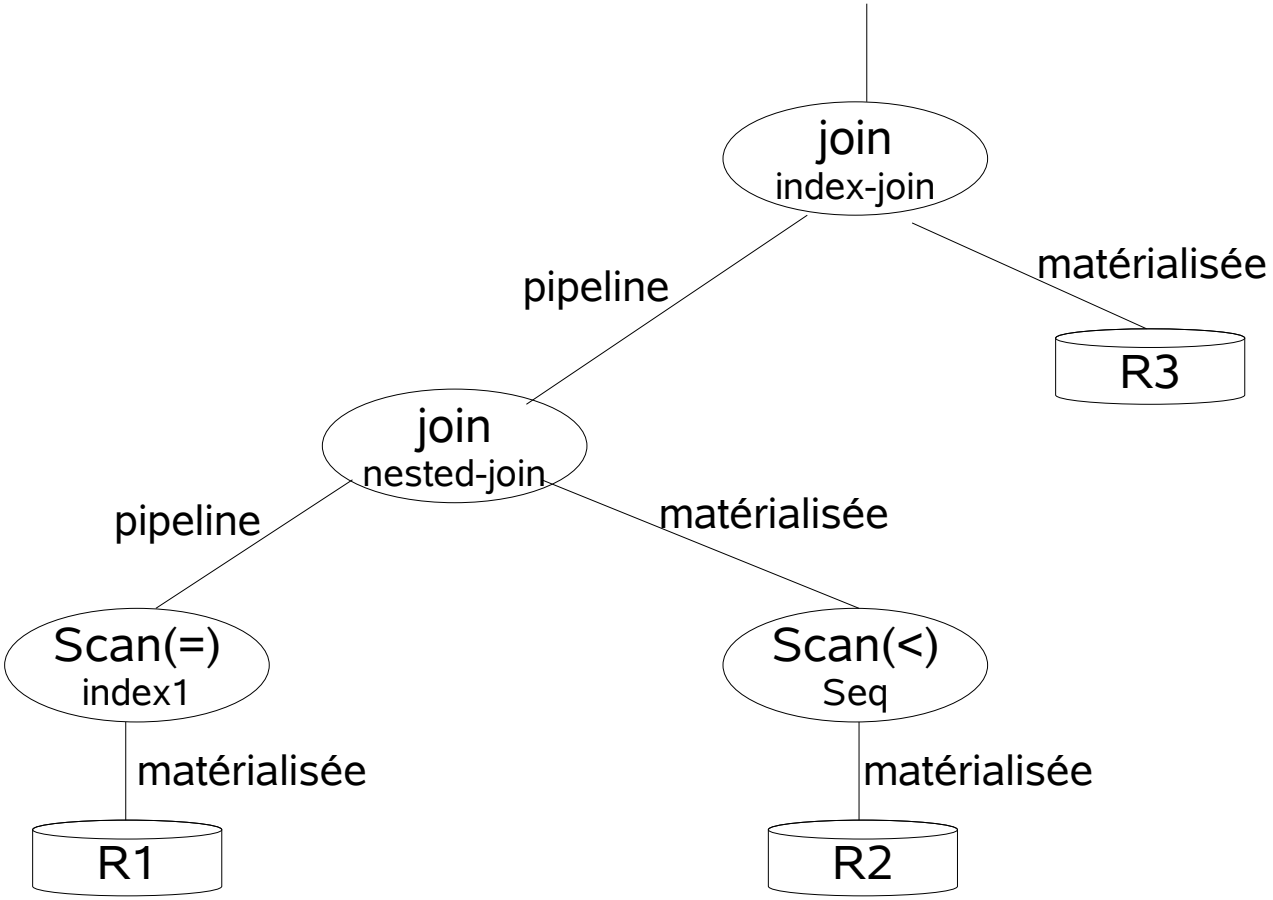
SELECT a1, a2, a3
FROM T1, T2
WHERE cond

-->

Proj(<a1, a2, a3>, Sel(<cond>, (T1xT2)))



Exemple de plan d'exécution prêt à être évalué



Optimisation logique

Propriétés des opérateurs relationnels

Associativité / Commutativité de la sélection

Sel(c1 & c2, R) = Sel(c2, Sel (c1, R))

Absorption des projections

Proj(A, (Proj B, (Proj C, R))) = Proj(A, R)

Distributivité sélection/projection

Proj(Y, Sel(c(X), R)) = Proj(Y, Sel(c(X), Proj(XUY, R)))

Distributivité sélection/jointure

Sel(c1 & c2, Join(c,R,S))) = Join(c, Sel(R,c1) , Sel(S,c2))

Distributivité projection/jointure

Proj(Y1 U Y2, Join(c(X1,X2),R,S)) = /* 1:cas général, 2:cas où les Xi ⊆ Yi */

1. Proj(Y1 U Y2, Join(c(X1,X2), Proj(Y1 U X1, R), Proj(Y2 U X2, S)))
2. Join(c(X1,X2), Proj(Y1, R), Proj(Y2, S))

Commutativité jointures : Join(c, R,S) = Join(c, S,R)

Associativité jointures : la condition c2 ne porte que sur les attr de R2 et R3

Join(c2 et c3, Join(c1,R1,R2), R3) = Join(c1 et c3, R1, Join(c2,R2, R3))

Optimisation logique

- un algorithme simple par heuristiques -

1. Convertir les Sélections avec conditions conjonctives en une cascade de Sélections avec conditions simples
2. Faire descendre le plus possible les Sélections en utilisant la commutativité et la distributivité
   les sélections qui ne traversent pas les produits seront transformées en Join
3. *Faire descendre le plus possible les Projections en utilisant la commutativité et la distributivité (option)*
4. Combiner des cascades de Sel et Proj dans une Sélection seule, une Projection seule ou une Sélection suivie par une Projection
5. Regrouper les noeuds de l'arbre. Chaque noeud interne binaire (⋂, U, x) constitue un groupe. A chaque groupe sont ajoutés tous les noeuds internes unaires ascendants et descendants se terminant par des feuilles
6. Chaque groupe sera transformé en un plan d'exécution.

Exemple

Soient les relations:

Livre(code, titre, ...) , Prêt(ncarteP, codeP, date), Lecteur(ncarte, nom, ...)

Soit la requête:

« Lister les noms des lecteurs et des titres des livres pour tous les prêts d'avant le '15/06/2004' »

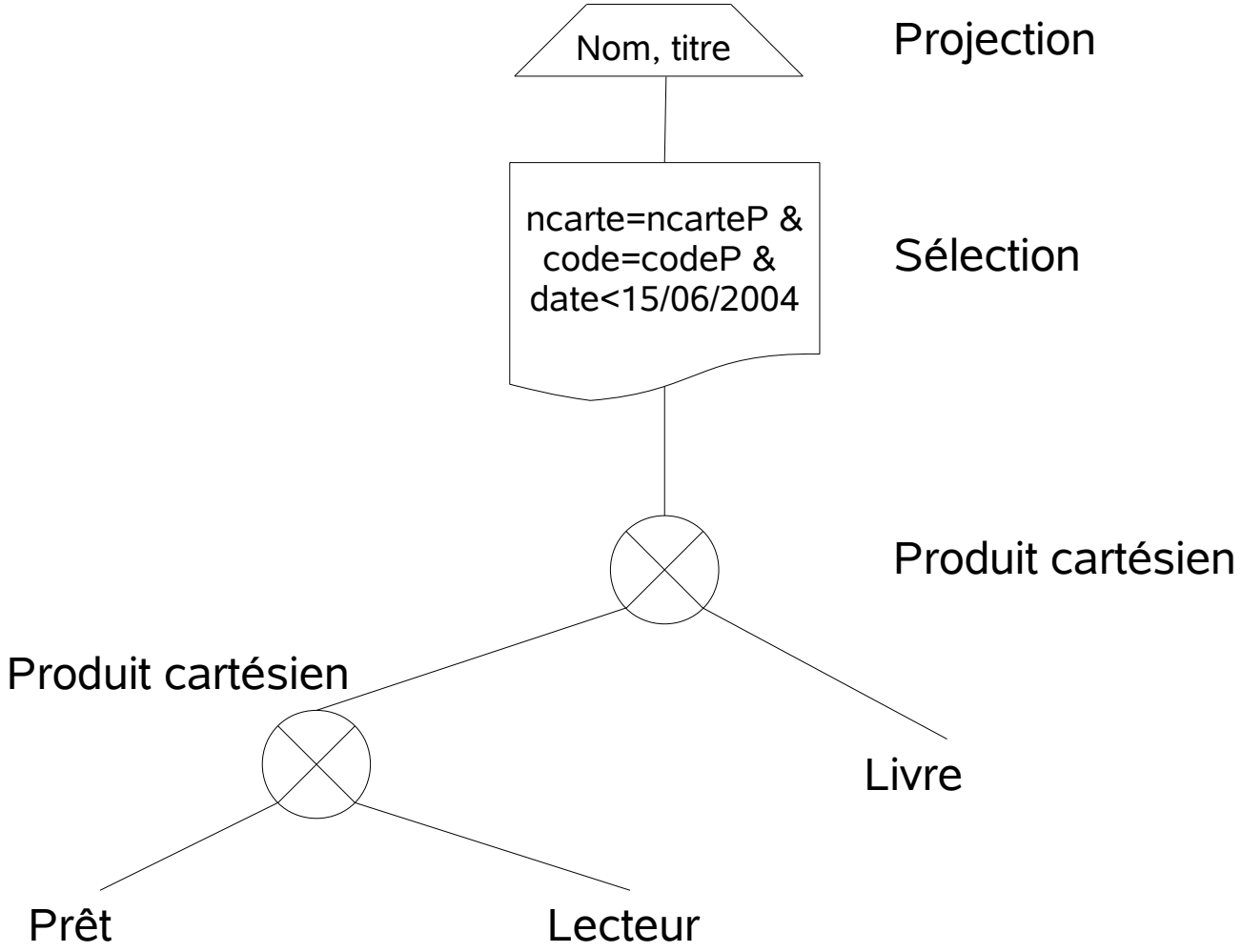
En SQL:

```
SELECT nom, titre
FROM Prêt, Lecteur , Livre
WHERE
      ncarte = ncarteP      AND
      code = codeP          AND
      date < '15/06/2004';
```

Traduction algébrique :

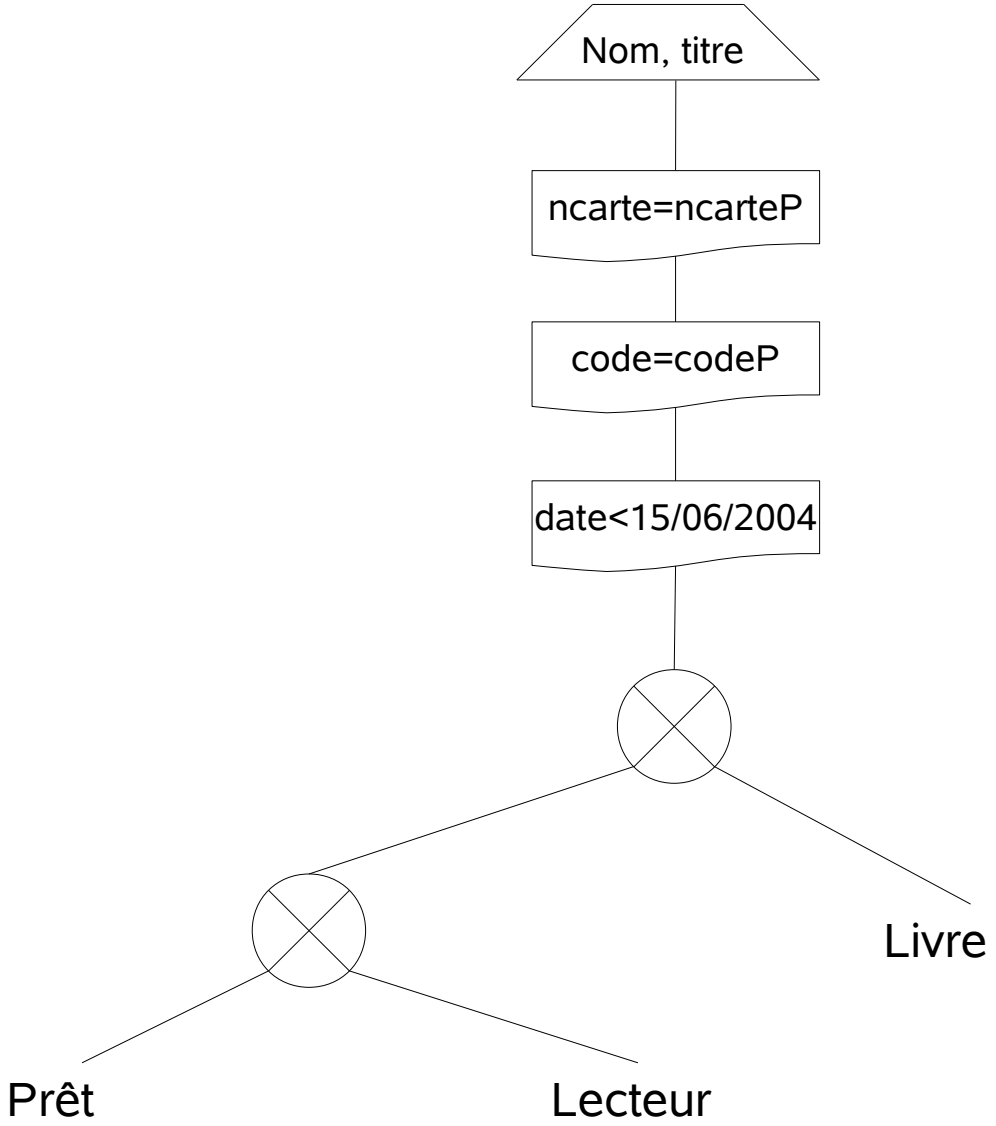
Proj(nom,titre, Sel(ncarte=ncarteP&code=codeP&date<'15/06/2004', (Prêt X Lecteur X Livre)

Arbre de la requête
non optimisé

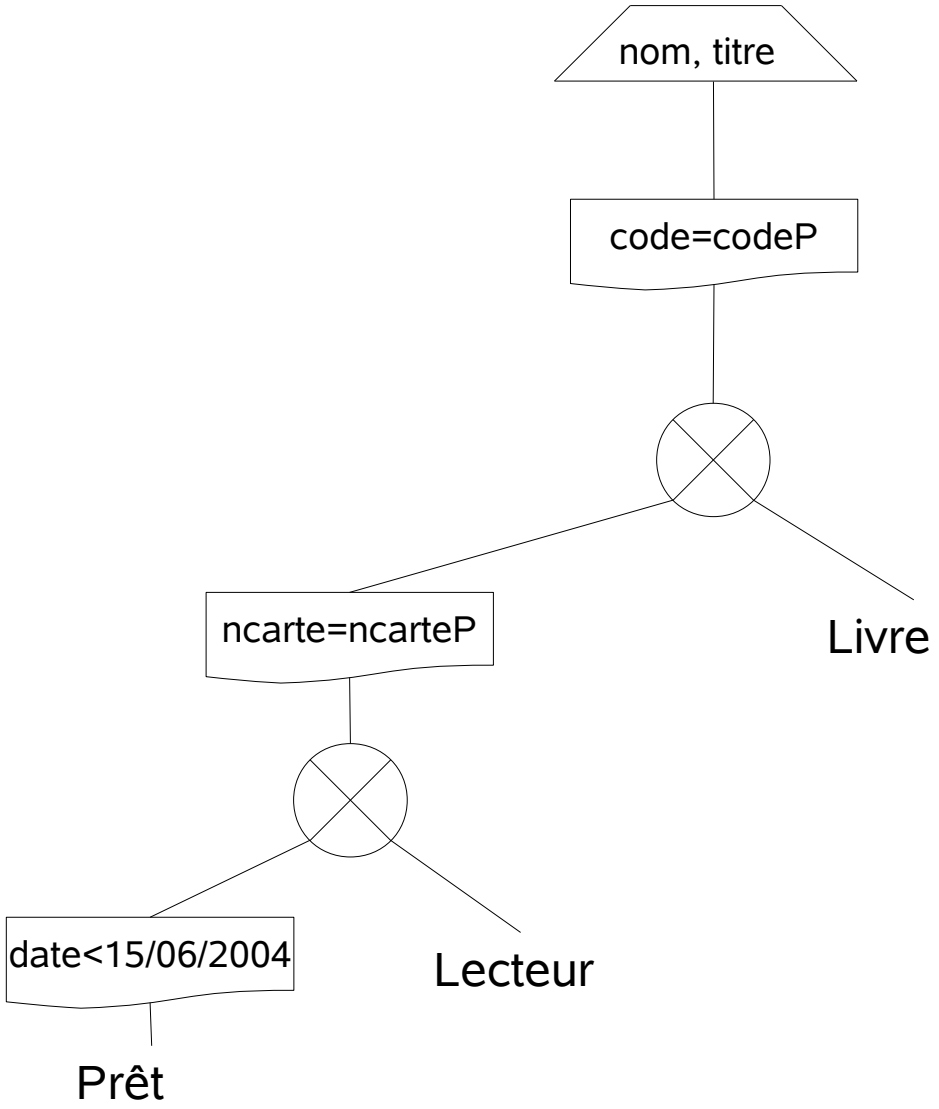


Arbre de la requête

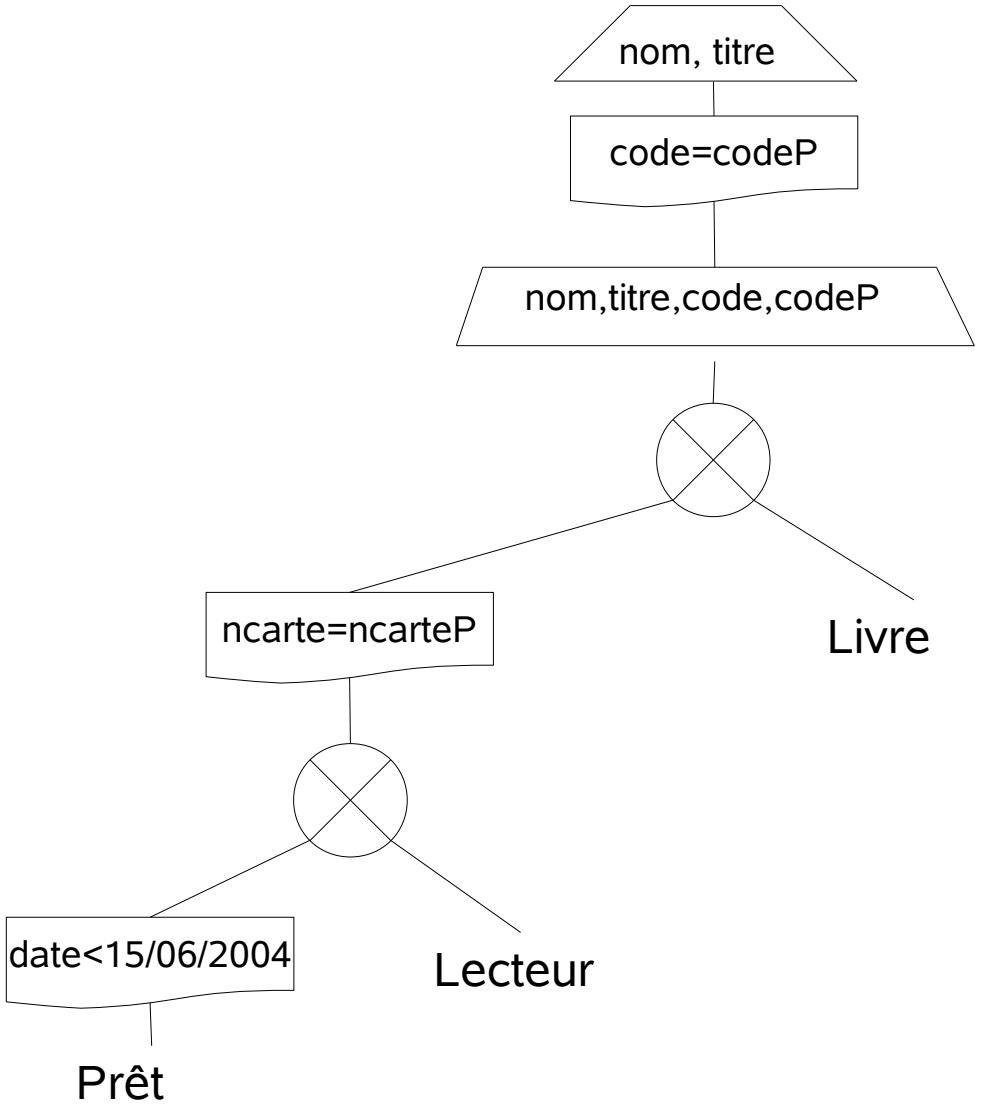
après décomposition des sélections en cascade



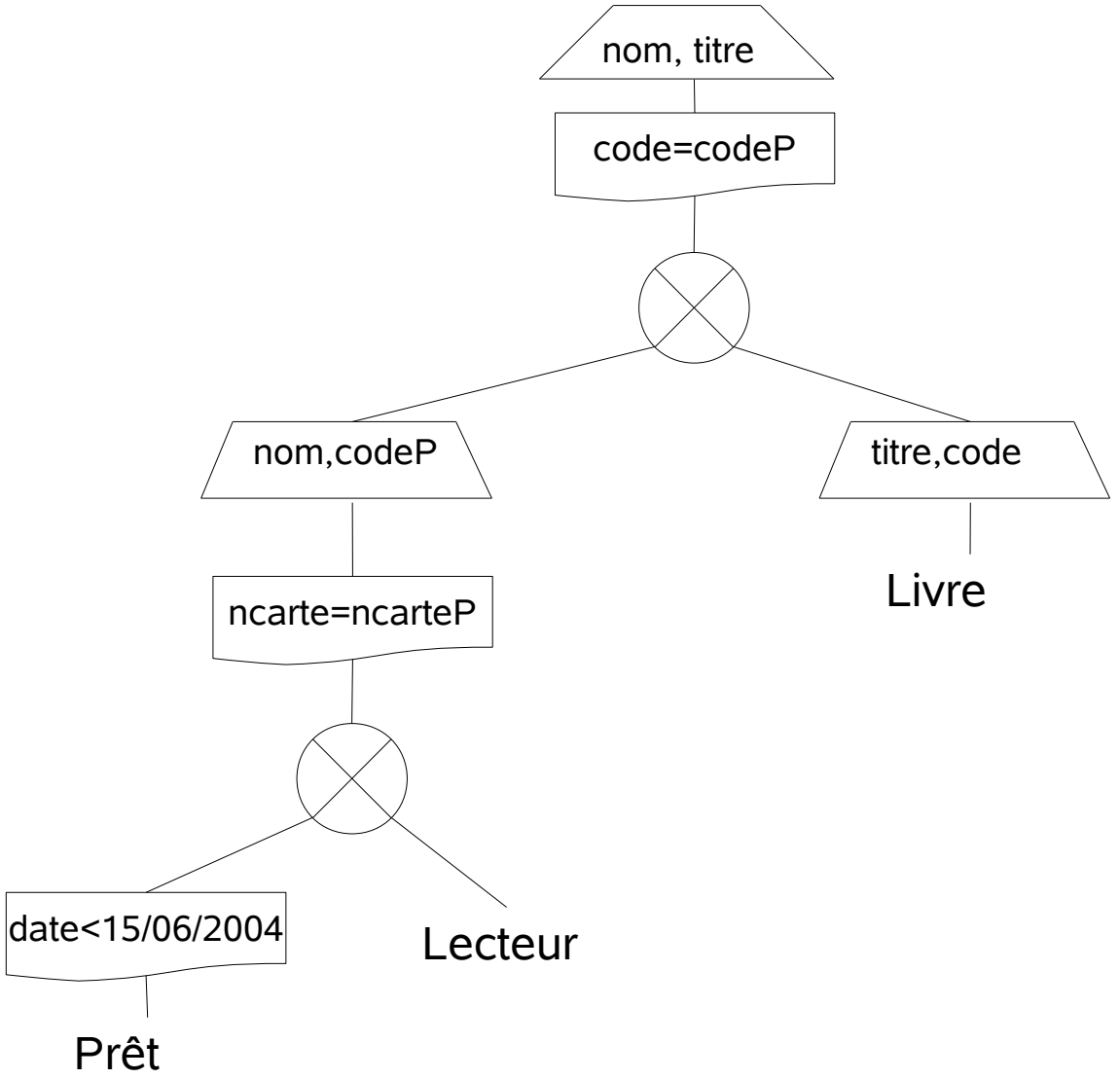
Arbre de la requête
après descente des sélections



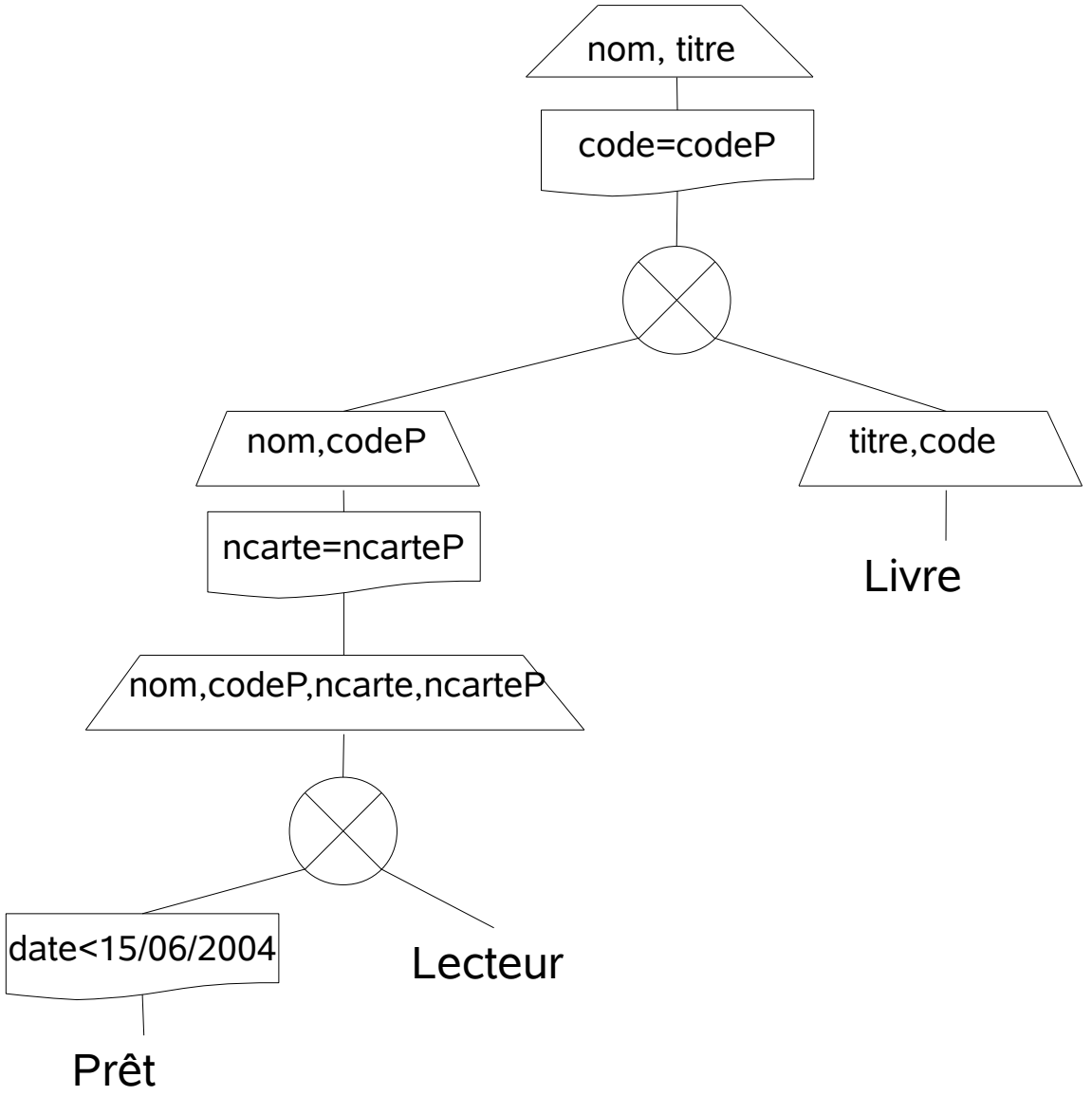
Arbre de la requête
descente des projections 1/4



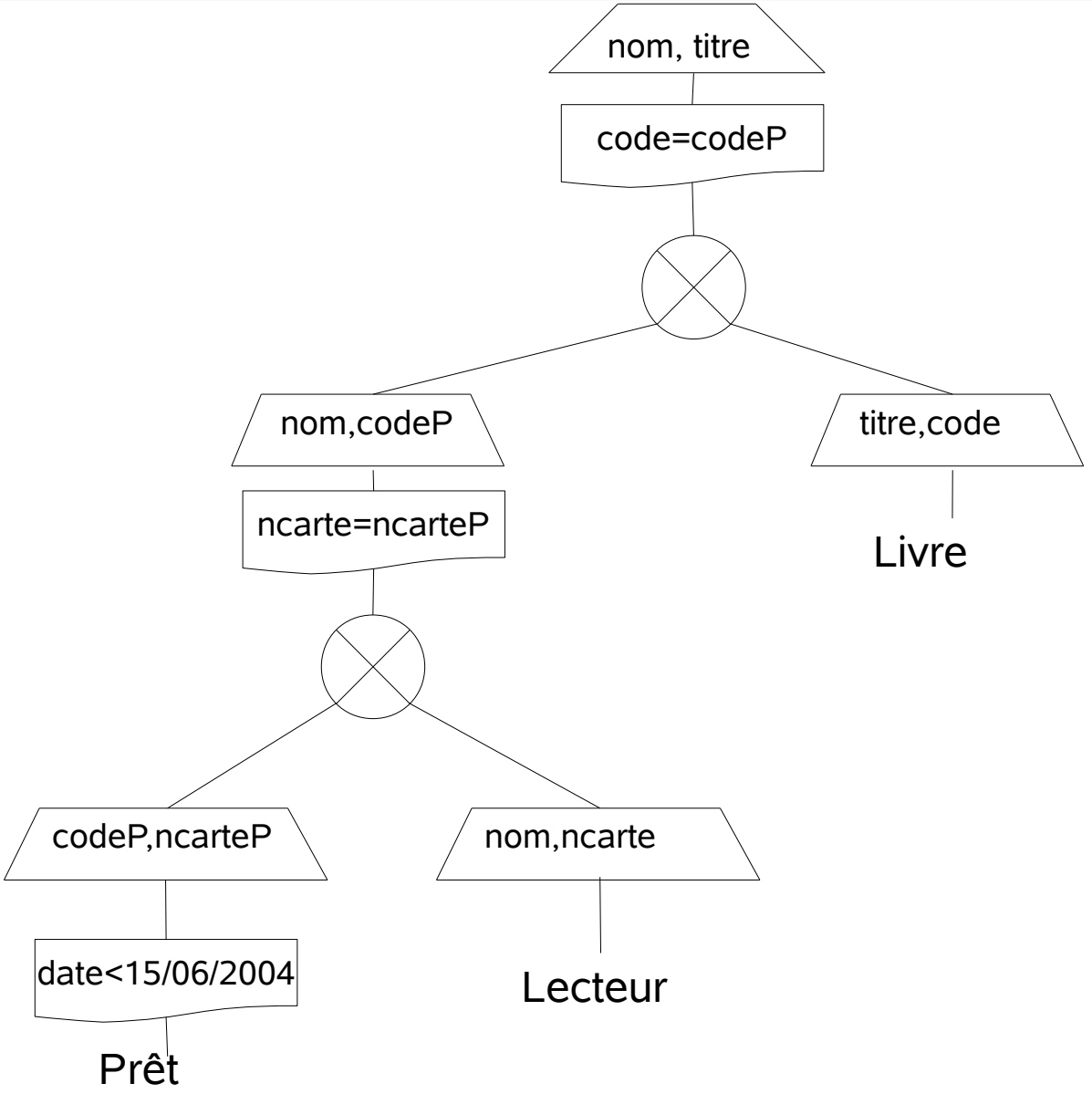
Arbre de la requête
descente des projections 2/4



Arbre de la requête
descente des projections 3/4



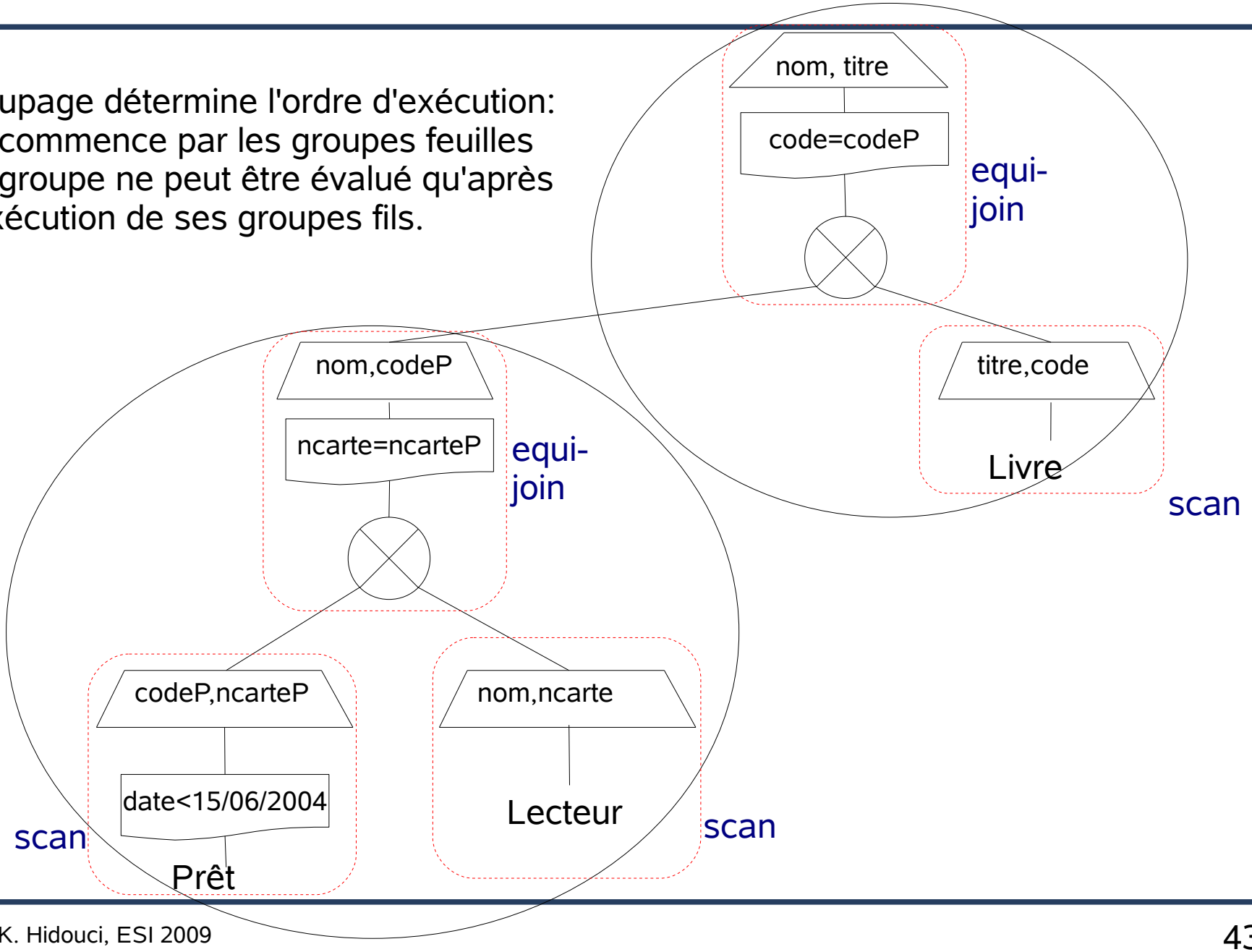
Arbre de la requête
descente des projections 4/4



Arbre de la requête

groupage des opérateurs unaires autour des op. binaires

Le groupage détermine l'ordre d'exécution:
- on commence par les groupes feuilles
- un groupe ne peut être évalué qu'après exécution de ses groupes fils.



Optimisation Physique
Statistiques et estimations

- Choix des algorithmes à utiliser pour les opérateurs, en fonction :
  - Des chemins d'accès existant
  - De la taille de la mémoire disponible
  - De la taille des tables en entrée (existantes et temporaires)
  - De statistiques sur les données (min,max, nb de valeur distinctes, ...)
    - Issues principalement des catalogues systèmes
      ==> coût de maintenance élevé
- Choix des connections entre les opérateurs
  - Par résultats matérialisés (Single/Double buffering)
  - Par utilisation des pipelines (Demand/Producer driven)

Quelques données statistiques dans les catalogues

Nr : le nombre de tuples dans la table R

Br : le nombre de blocs occupés par la table R

Lr : le nombre d'octets d'un tuple de R

CI(index) : le coût d'une recherche dans l'index
= hateur de l'arbre si l'index est un B-Arbre

V(A,R) : le nombre de valeurs distinctes de l'attribut A dans R
Si A est une clé, V(A,R) = Nr

min(A,R) : la valeur minimale de A dans R

max(A,R) : la valeur miximale de A dans R

histogramme(A,a,b,R) : le nombre de valeurs distinctes entre a et b de l'attribut A dans la table R
==> si cette information n'est pas disponible, on suppose alors que les valeurs de A sont uniformément distribuées.

Estimation sur les résultats temporaires

Sélectivité d'une opération de sélection de la forme : Sel(c,R)

Probabilité qu'un tuple de R vérifie la condition c

= Taille(Sel(c,R)) / Nr

Estimation de **V(A, Sel(c,R))**

= 1 si c est de la forme (A=v)

= n si c est de la forme (A=v1 ou A=v2 ou ... A=vn)

= V(A,R) * Sélectivité de la sélection si c est de la forme A op v
avec v un opérateur relationnel (<, >, <= ou >=)

= min(V(A,R), taille(Sel(c,R))) dans le cas général

Estimation de **V(A, Join(c,R,S))**

min(V(A,R), taille(Join(c,R,S))) si A provient de R seulement
min(V(A,S), taille(Join(c,R,S))) si A provient de S seulement
min(V(A,R), V(A,S), taille(Join(c,R,S))) si A est dans R et dans S

Estimation de la taille d'une sélection

taille (Sel(cond , R))

Taille = Nr * Probabilité qu'un tuple de R vérifie 'cond'

Condition de la forme **A=v**

taille = Nr/V(A,R)

Condition de la forme **A <= v ou bien A < v**

taille = Nr * (v – min(A,R)) / (max(A,R) – min(A,R))

Condition de la forme **A >= v ou bien A > v**

taille = Nr * (max(A,R) - v) / (max(A,R) – min(A,R))

Condition de la forme **Not(c)**

taille = Nr – taille(Sel(c,R)) (s'il n'y a pas de NULL)

= le nombre de tuples de R moins le nombre de tuples qui vérifient c

= Nr * (1 - la sélectivité de c)

Taille d'une sélection complexe

Taille(Sel(c1 et c2 et ... et cn , R))

Posons Si l'estimation de la taille de Sel(ci,R)
alors la probabilité qu'un tuple vérifie Sel(ci,R) = Si/Nr
la probabilité qu'un tuple vérifie c1 et c2 ... et cn est :
(S1/Nr) * (S2/Nr) * ... *(Sn/Nr)
donc la taille de la sélection = **$Nr * S1 * S2 * ... Sn / Nr^n$**

Taille(Sel(c1 ou c2 ou ... ou cn , R))

La probabilité qu'un tuple vérifie la disjonction est égale à :
1 - la probabilité qu'il ne vérifie aucune condition ci
or cette dernière probabilité représente la sélectivité de :
Sel(not c1 et not c2 et ... et not cn , R)
qui est donnée par : (1-S1/Nr) * (1-S2/Nr) * ... (1-Sn/Nr)
donc la taille = **$Nr * (1 - (1-S1/Nr) * (1-S2/Nr) * ... (1-Sn/Nr))$**

Taille d'une jointure

Taille de RxS = Nr * Ns

Taille de équi-Join(R, S) avec attr de jointure une clé

< Ns si attr de jointure est une clé de R,
car chaque tuple de S peut se joindre avec au plus un tuple de R

< Nr si attr de jointure est une clé de S
car chaque tuple de R peut se joindre avec au plus un tuple de S

= Ns si attr de jointure est une clé étrangère dans S ref. R

= Nr si attr de jointure est une clé étrangère dans R ref. S

Taille de équi-Join(R, S) avec attr de jointure quelconque

On estime que chaque tuple de R va générer Ns/V(A,S) tuples dans le résultat, soit donc une taille de Nr*Ns/V(A,S)
De manière analogue, en considérant chaque tuple de S, on aurait eu un résultat de Ns*Nr/V(A,R)
En combinant les 2 estimations on aura alors :
taille = Nr*Ns / min(V(A,S) , V(A,R))

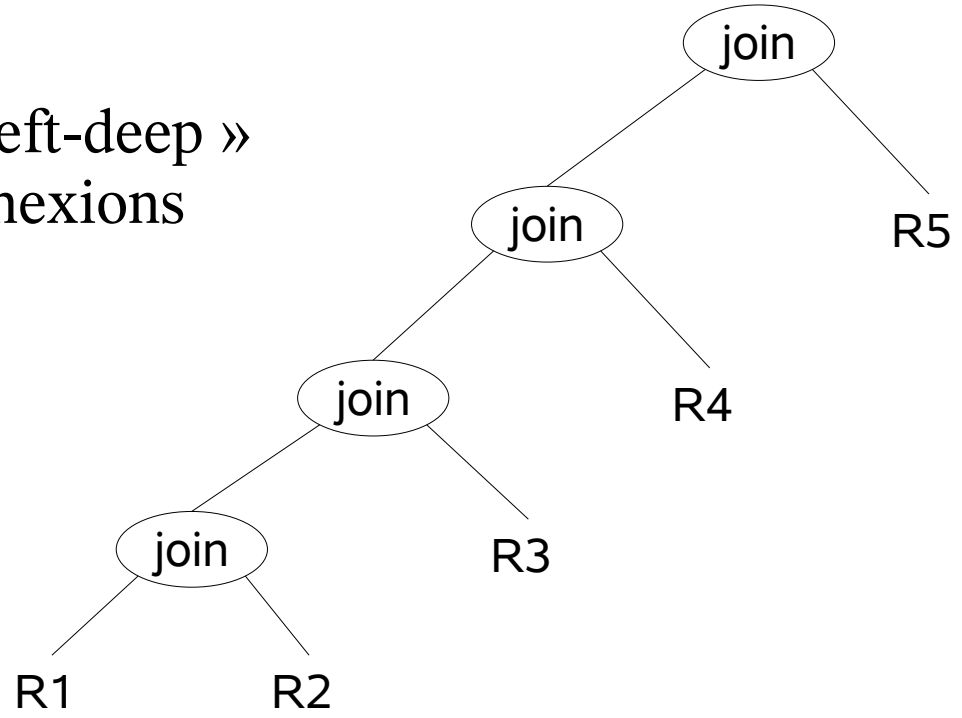
Taille de Join(c, R, S) = Taille de Sel(c, RxS)

Le « join ordering »

L'estimation des tailles des jointures permet de sélectionner l'ordre optimal d'exécution d'une cascade de jointures

Souvent on ne considère l'ordre « left-deep » qui favorise l'utilisation des connexions « en pipeline »

Left-deep join tree



Taille d'autres opérations

Union(R, S)

l'ensemble des tuples de R plus ceux de S moins les valeurs en double

$< Nr + Ns$

Intersect(R, S)

difficile à estimer généralement, mais dans le pire des cas une des relation est totalement incluse dans l'autre

$\min(Nr, Ns)$

Moins(R, S)

même raisonnement, le pire des cas : R incluse dans S

$< Nr$

Structure d'un optimiseur physique

- Générer les expressions algébriques équivalentes
  - Parcours en profondeur
- Enrichir l'espace de recherche en transformant chaque expression de l'étape précédente en un ou plusieurs plans physique
  - Choisir pour chaque opération un ou + algorithmes
  - Choisir pour chaque connexion les 2 possibilités (matérialisée et « en pipeline »)
- Associer un coût à chaque plan (en utilisant les statistiques)
- Sélectionner le meilleur plan ou presque
  - par prog. dynamique ou par heuristique