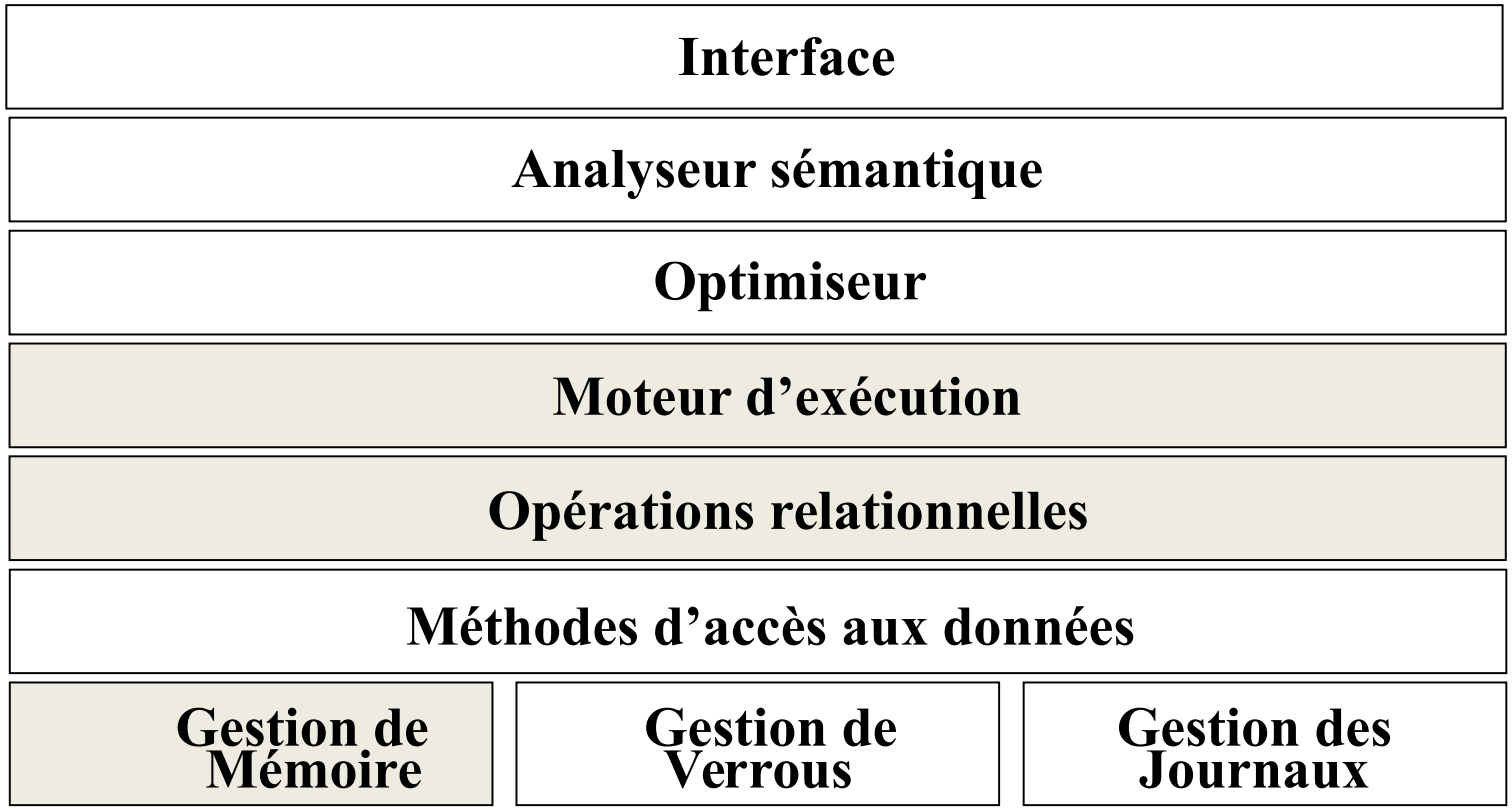
Algèbre Relationnelle : Implémentation

- Rappel de l'algèbre
- Structures d'indexation et placement
- Algorithme de sélection
- Algorithme de jointure

1. RAPPELS


Interface
Analyseur sémantique
Optimiseur
Moteur d'exécution
Opérations relationnelles
Méthodes d'accès aux données
Gestion de Mémoire
Gestion de Verrous
Gestion des Journaux
Système opérationnel

Langages de requêtes

- Deux langages de requêtes à fondements mathématiques sont à la base de SQL :
  - Calcul relationnel:
    - Permet aux utilisateurs de décrire ce qu'ils veulent, plutôt que la manière dont ce qu'ils veulent doit être calculé. (Non opérationnel, déclaratif.)
  - Algèbre relationnelle:
    - Plus opérationnelle, très utile pour représenter les plans d'exécution.

Calcul Relationnel

- Dérivé de la logique
- Requêtes de la forme
  - {X | Formule(X, Y) }
  - X est un vecteur de variables non quantifiées (x1, x2, …)
  - Y est un vecteur de variables quantifiées ((∀/∃) y1, y2, …)
  - Chaque variable représente une colonne de table
- {(x1,x2) | ∃ y1 (Vins(x1,x2,.,y1) AND y1 = 1999) }

Algèbre relationnelle

- Opérations de base:
  - Sélection (σ)
    - Sélectionne un sous-ensemble des lignes d'une relation.
  - Projection (π)
    - Efface des colonnes d'une relation [et élimine les doubles].
  - Produit Cartésien (X)
    - Permet de combiner deux relations.
  - Différence (-)
    - Elimine les tuples de R1 contenus dans R2
  - Union ($\cup$)
    - Constitue une relation R avec les tuples de R1 et ceux de R2

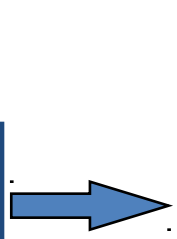
Algèbre relationnelle

- Opérations additionnelles:
  - Jointure (|▯|)
    - Combinaison de produit cartésien et sélection sur colonne Θ comparables (=, <, >, ...)
  - Intersection
    - Constitue une relation R avec les tuples appartenant à la fois à R1 et R2
- Chaque opération retournant une relation, les opérations peuvent être composées!
  - L'algèbre est fermée.

Division

- C= A / B
- C(X) est la division de A (X, Y) par B (Y) ssi C contient tous les tuples (x) tels que $\forall(y) \in B$, $\exists(x, y) \in A$

Quels sont les fournisseurs de toutes les pièces



S#	P#
S1	P1
S1	P2
S2	P1
S2	P3

A

P#
P1
P2

B

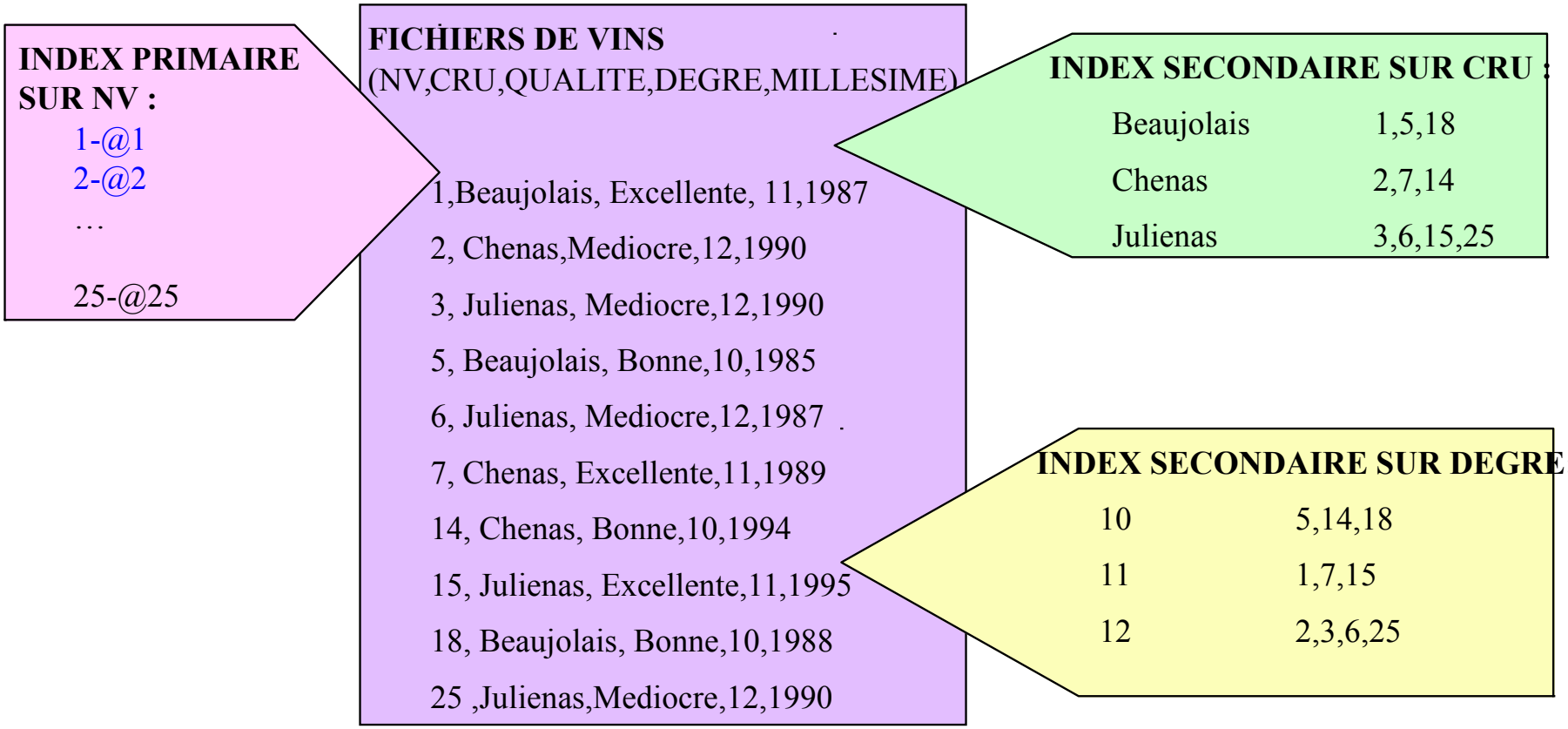
S#
S1

C

2.STRUCTURE PHYSIQUE DES TABLES

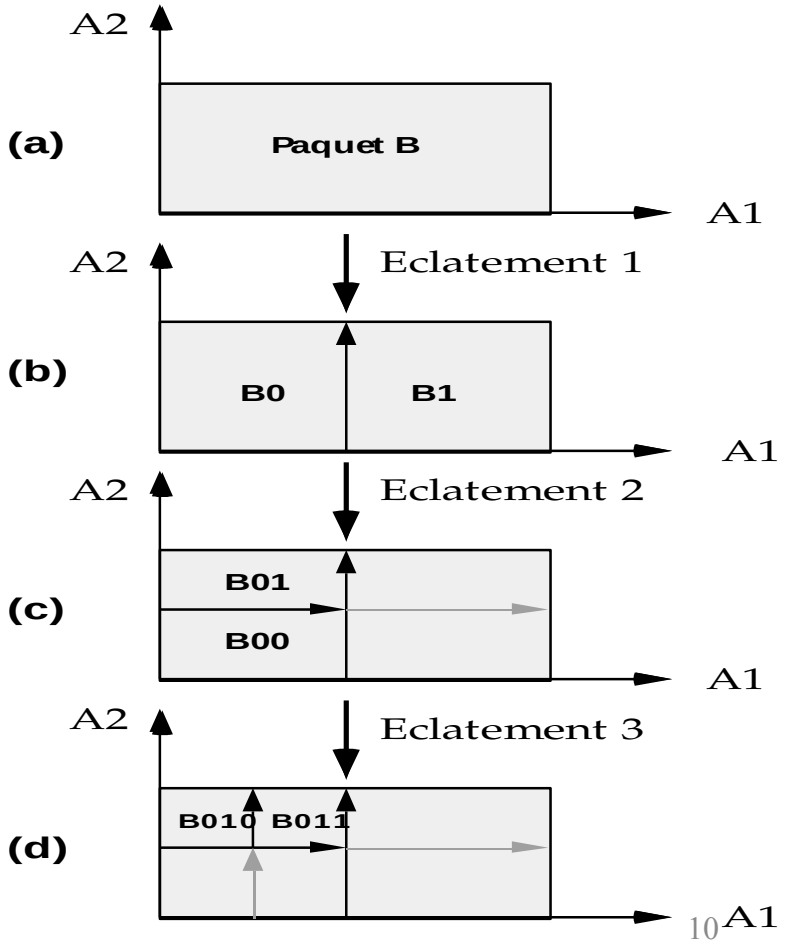
- Cas typique
  - Index plaçant sur clé primaire
    - Fichier type VSAM (B+ tree, trié sur clé primaire)
      - Valeur de clé ➔ adresse tuple
    - Plusieurs index non plaçants sur clé secondaire
      - Valeur de clé ➔ liste d'adresses de tuples
- Variantes
  - Hachage sur clé primaire
  - Table partitionnée
    - Hachage sur clé secondaire, index par partition
  - Placement multi-attributs

Exemple


INDEX PRIMAIRE SUR NV :
1-@1
2-@2
…
25-@25
FICHIERS DE VINS
(NV,CRU,QUALITE,DEGRE,MILLESIME)
1,Beaujolais, Excellente, 11,1987
2, Chenas,Mediocre,12,1990
3, Julienas, Mediocre,12,1990
5, Beaujolais, Bonne,10,1985
6, Julienas, Mediocre,12,1987
7, Chenas, Excellente,11,1989
14, Chenas, Bonne,10,1994
15, Julienas, Excellente,11,1995
18, Beaujolais, Bonne,10,1988
25 ,Julienas,Mediocre,12,1990
INDEX SECONDAIRE SUR CRU :
Beaujolais 1,5,18
Chenas 2,7,14
Julienas 3,6,15,25
INDEX SECONDAIRE SUR DEGRE:
10 5,14,18
11 1,7,15
12 2,3,6,25

Placement Multi-dimensionnel

- 1. Insertion
  - Eclatement progressif du fichier selon les bits des fonctions de hachage sur différents attributs
- 2. Recherche
  - Codage du critère par les fonctions de hachage
  - Balayage de p << N régions
- Exemple
  - A1 = CRU Hachage selon 1e lettre
  - A2 = DEGRE Hachage selon entier
  - CRU = CHABLIS ET DEGRE = 12
  - ➔ Région B101



Index Bitmap

- Cas des index peu discriminants
- Exemple
  - 10**6 lignes de commandes
  - 100 produits
  - Index sur produit ➔ 10**4 adresses par entrées !!
- Index lourd à manipuler !
- Variante : Index bitmap
- Index bitmap
  - Table de bits telle que
  - Bit [i,j] = 1 si le i^{e} tuple référence la valeur codée j
- Accélérateur d'accès très utile et compacte pour attribut peu discriminant

Exemple

Ménagère	Produit	Coût	P1	P2	P3	P4	P5
1	P2	120	0	1	0	0	0
2	P1	80	1	0	0	0	0
3	P3	150	0	0	1	0	0
4	P2	120	0	1	0	0	0
5	P4	70	0	0	0	1	0
6	P4	70	0	0	0	1	0

- Qui a acheté P2 ?
- Qui a acheté P2 ou P4 ?
- Qui a acheté P2 ou P4 et pas P3 ?

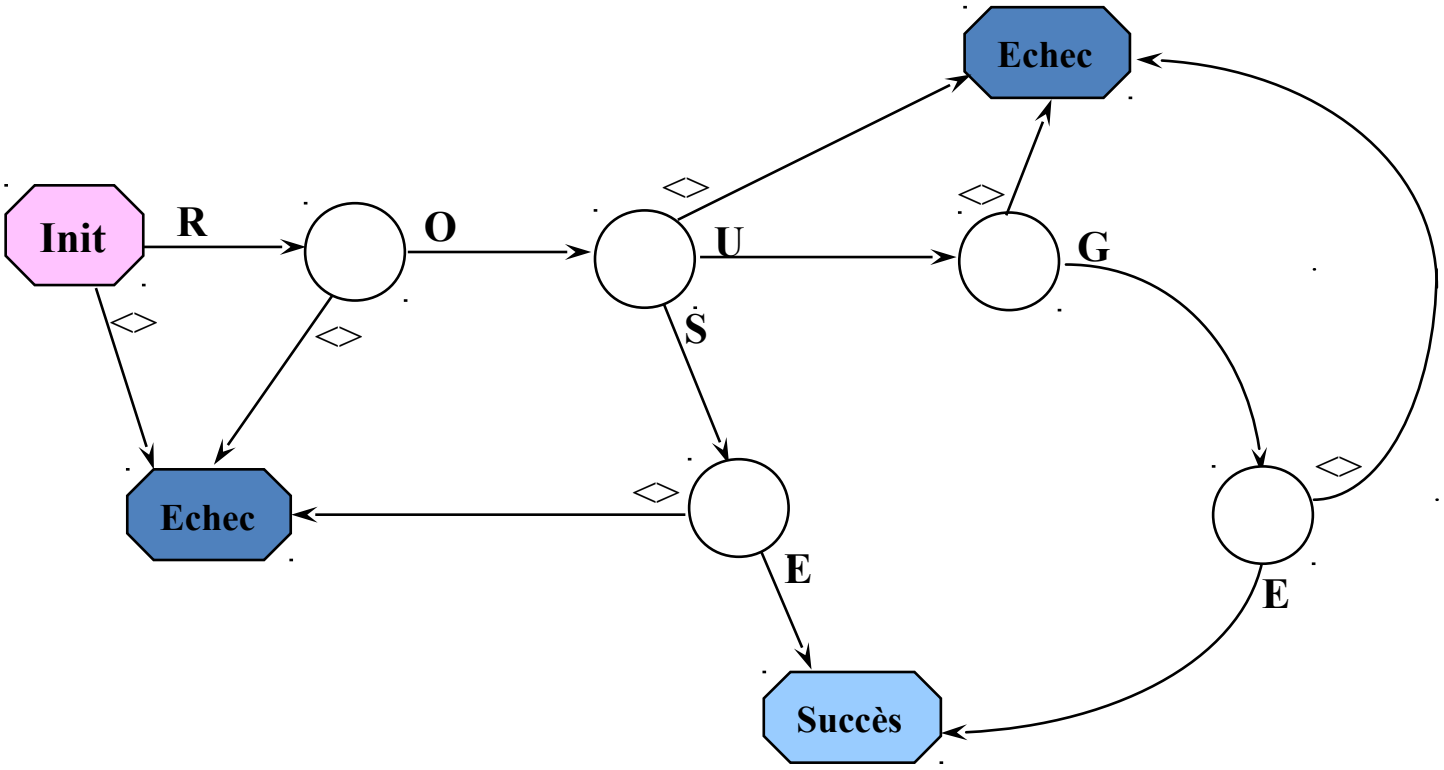
3. EXECUTION DES SELECTIONS

- CAS SANS INDEX
  - Filtrage séquentiel de la relation
  - Peut être réalisé par un automate matériel
- Algorithme Select (R, Q) :

```
Foreach page p de R do {
    Read (p);
    Foreach tuple t de p
        { if  CheckS (t, Q) then result = result ∪ t ;}
  ;
}
```

Exemple d'automate

- CheckS (Couleur = "ROUGE" ou "ROSE")



Sélection via index

- **Cas avec index**
  - unidimensionnel (Isam, Arbre-B)
  - multi-dimensionnel (Grid file, Arbre de prédicat)
- **Variante : Cas avec hachage**
  - unidimensionnel
  - multi-dimensionnel (réduction du balayage)

Utilisation d'Index B-tree

- **Intersection et union de listes d'adresses de tuples**
  - Accès aux tuples dont les identifiants sont retenus
  - Vérification du reste du critère
- **Exemple : Utilisation de tous les index**
  - (CRU = "CHABLIS") AND (MILL > 1999) AND (DEGRE = 12)
  - $L = C \cap (M1 \cup M2 \cup ... \cup Mp) \cap D$
  - Accéder aux tuples de L
- **Exemple : Choix du meilleur index**
  - Accès via l'index (CRU = "CHABLIS")
  - Vérification du reste du critère sur les tuples

Utilisation d'index Bitmap

- Détermination des adresses des tuples candidats
- Possibilité de mixer avec des index B-Tree
- Très utile pour les agrégats
  - Somme des coûts dépensés pour produit P2
- Et le data mining …

4. EXECUTION DES JOINTURES

- Opération essentielle des SGBDR
- De multiples algorithmes
  - Sans index :
    - réduire les balayages
    - Optimiser les comparaisons
    - Créer dynamiquement des index ou tables de hachage
  - Avec index
    - Profiter au mieux des index
  - Toujours : Réduire I/O et temps CPU

Jointure et opérateurs similaires

- Les mêmes types d'algorithmes peuvent êtres utilisés pour les autres opérateurs binaires
- Variations :
  - Nested Loop Join
  - Block Nested Loop Join
  - Index Join
  - Sort Merge Join
  - Hash Join
  - Hybrid Hash Join
  - Pipelined Hash Join

Évaluations avec Mémoire

- On veut joindre les tables CLIent et COMmandes
- |CLI| = Nbre de pages occupées par CLI
- ||CLI|| = Nbre de tuples dans CLI
- |CLI| << |COM|
- M = Nbre de pages mémoires disponibles
  - (quelques approximations pour simplifier les formules)
- Coût = Nbre d'I/O
  - (on ne distingue pas I/O séquentielle et aléatoire)

Jointure avec un index (Index Join)

- Cas avec 1 index sur R1

  Pour chaque page q de R2

  Lire (q)

  Pour chaque tuple t de q

  Accéder a R1 via index

  Joindre si succès

- Coût = |R2| + ||R2||*(n+1) avec $n \leq 2$

Prise en compte de la mémoire

- Hypothèse : index sur la clé primaire de CLI
  - L'index a n niveaux et tient sur p pages
- Principe : (équijoins)
  - pour chaque commande, récupérer le n° de client, parcourir l'index et retrouver le client
- Brut Force (ou M = 0)
  - Coût = $||COM|| \times (n+1) + |COM|$
- M = $|CLI| + p$
  - Coût = $|COM| + p + |CLI|$
- Intérêt de la technique ???

Jointure avec 2 index

- Les indexes sont triés
  - B-Tree, B+ Tree
  - Les articles ne sont pas forcément triés
- Fusion des index
  - Couples (@t1, @t2) des tuples joignant
  - Appelé index de jointure
  - Trié sur @t1, @t2
  - Accès aux tuples et concaténation
- Remarque :
  - Possibilité de maintenir l'index de jointure

Jointure sans index (Nested Loop)

- Cas sans index ➔ balayages multiples
- Par boucle imbriquée
  - Foreach page p de R1
  - Foreach page q de R2
  - Foreach tp de p
  - Foreach tq de q
  - { if CheckS (t, Q) then result = result ∪ (tp||tq) ;} ;
- Coût = page(R1) * page(R2) + page(R1)

Prise en compte de la mémoire

- Principe :
  - Pour chaque client, lire ttes les commandes et comparer
- Brut Force :
  - coût = ||CLI|| $\times$ |COM| + |CLI|
- M = 0 : aucune bufferisation
  - ➔ coût = |CLI| $\times$ |COM| + |CLI|
- M >= |CLI| : on cache CLI dans M
  - ➔ coût = |CLI| + |COM|
- Entre les deux : on cache M pages de CLI
  - ➔ coût = (|CLI|/M) $\times$ |COM| + |CLI| (B.N.L.)

Tri-Fusion (Sort-Merge Join)

- Par tri-fusion
  - Trier (R1, AttJ)
  - Trier (R2, AttJ)
  - Fusionner (R1, R2)

- Coût = page(R1) LOG page(R1) +
  page(R2) LOG page(R2) +
  page(R1) + page(R2)

Prise en compte de la mémoire

- Principe : (équijoins)
  - Trier les deux relations sur l'attribut de jointure
  - Parcourir simultanément les deux relations et joindre
- Si M est trop petite ➔ Trôp coûteux !
- Si M >= sqr(|COM|) ➔ coût = $3 \times (|CLI| + |COM|)$
- Si une des relations est déjà triée (ex. CLI est placée sur la clé primaire) on économise la phase de tri (ici $2 \times |CLI|$)

Jointure par Hachage

- On crée un fichier haché (ou indexé)

  Hacher (R1) (plus petite relation)

  Pour chaque page p de R2

    Lire (p)

    Pour chaque tuple t de p

      Accéder au fichier aléatoire

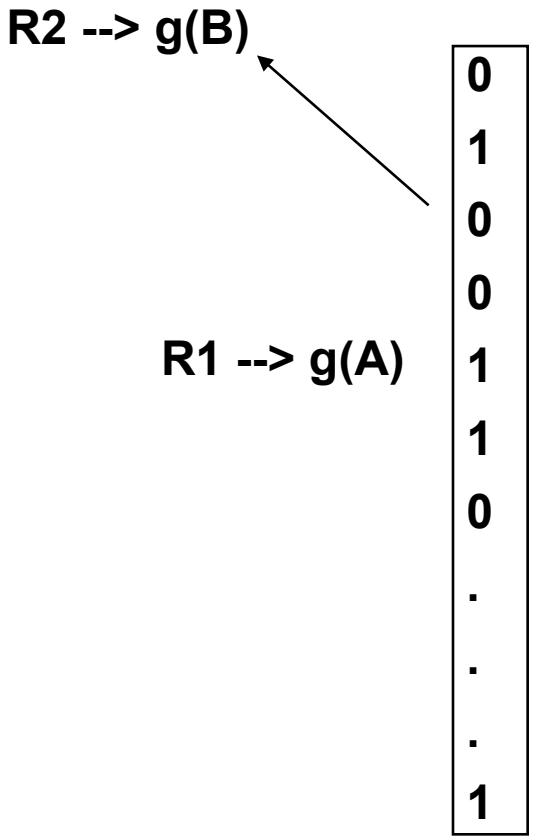
      Ajouter la concaténation au résultat si succes

- Coût = page(R1) + A page(R1) +

  page(R2) + A page(R2)

Amélioration par Tableaux de Bits

- Tableau de bits par hachage de R1 en mémoire pour éliminer les accès sans chance de jointure lors de la lecture de R2.
- Utilisation d'une fonction g distribuant sur un vecteur de bits assez long ...

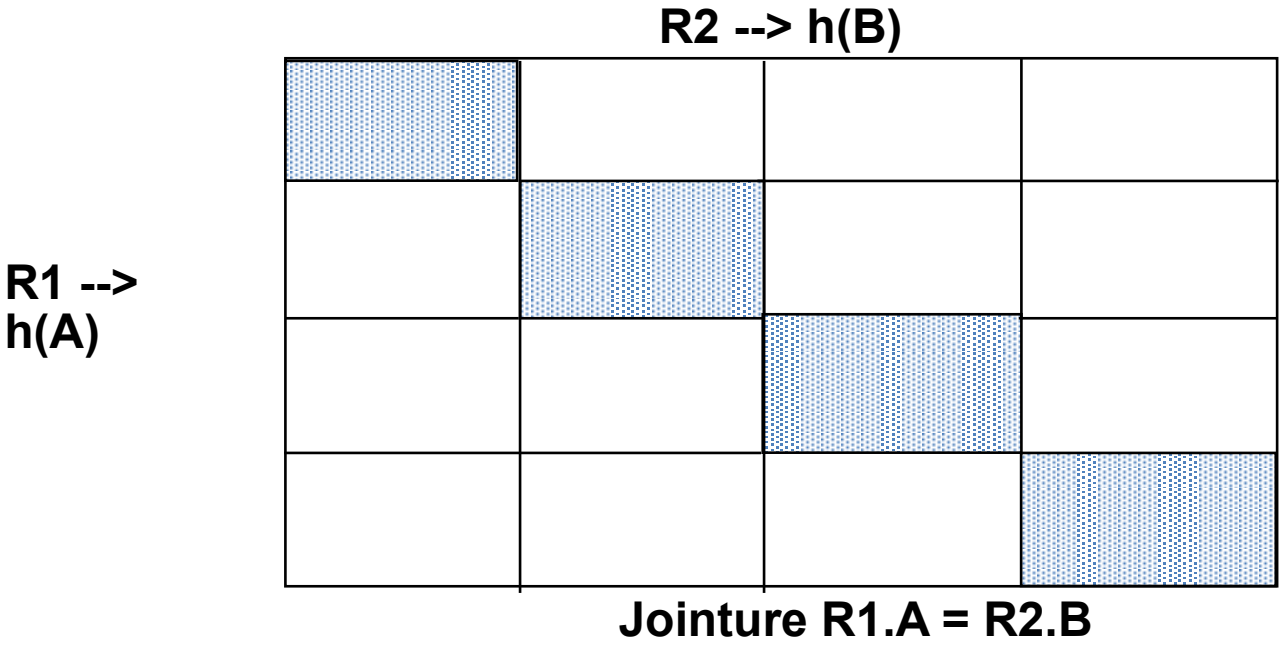


Prise en compte de la mémoire

- Hypothèse M >=|CLI|
- Principe (équijoins) :
  - Charger CLI et hacher la relation en mémoire en n paquets (n du même ordre que ||CLI||)
  - Pour chaque tuple de COM, tester la table de hachage
- Coût = |CLI| + |COM|
- Différence avec le Nested Loop ?

Jointure par Bi-hachage

- Hachage des deux tables en paquets en mémoire (virtuelle) par une même fonction h
- Jointure des paquets de même rang



Grace Hash Join

- **Hypothèse :** M >= sqr(|COM|)
- **Principe** :
  - Hacher sur le disque CLI et COM en N paquets. N est calculé de façon à ce que |paquet| < M
  - Joindre chaque paquet de CLI avec chaque paquet de COM
  - coût = $3 \times (|CLI| + |COM|)$

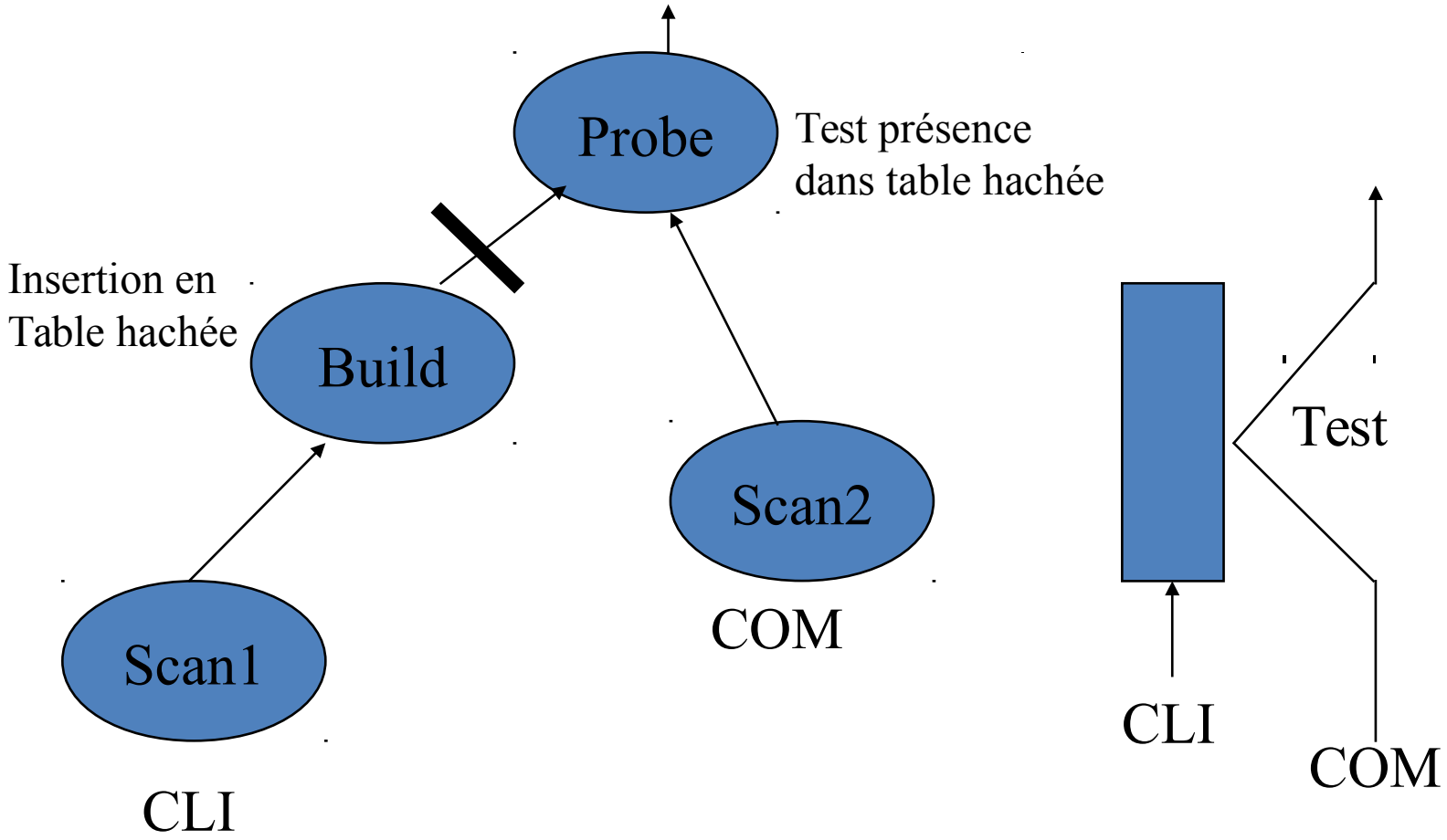
Hybrid Hash Join

- Exemple M = |CLI|/n
- Principe :
  - Hacher CLI en n paquets. Le paquet 1 est conservé en mémoire les paquets 2 à n sont réécris sur disque.
  - Hacher COM en n paquets. Le paquet 1 est testé immédiatement en mémoire avec le paquet 1 de CLI. Les autres sont réécris sur disque
- Coût : $(|CLI|+|COM|) \times (1 + 2(n-1)/n)$
- A.N. : n = 2 ➔ Coût = $2 \times (|CLI|+|COM|)$

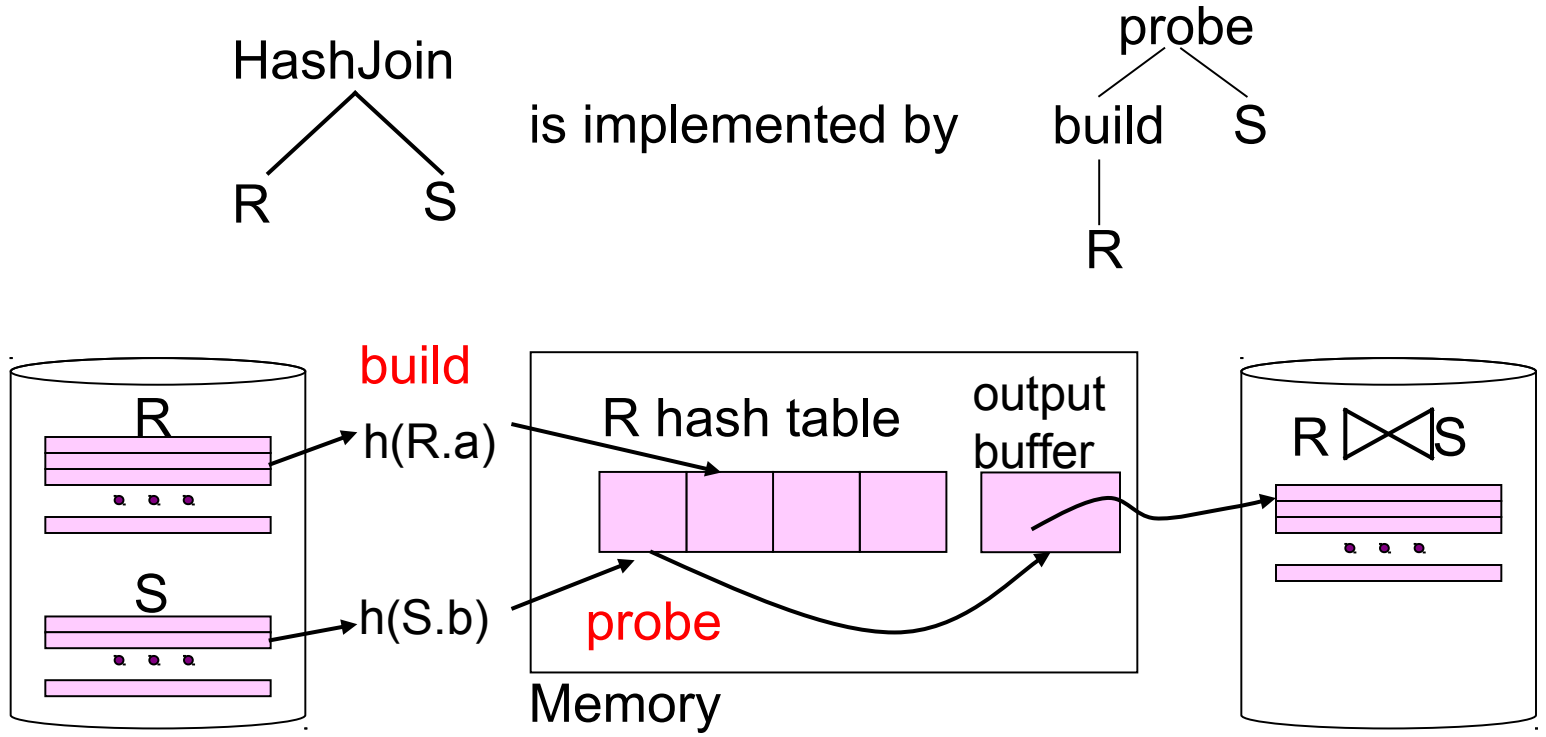
Mode pipeline ou bloquant

- Le pipeline sort un tuple résultat dès qu'il est calculé
- Pas nécessaire d'attendre le dernier tuple résultat pour débloquer l'utilisateur
- Plus efficace vu de l'utilisateur
- Opérateur bloquant
  - Tri
- Opérateur non bloquant
  - Sélection, hachage

Hachage et Pipeline



HashJoin: Build-Probe



5. CONCLUSION

- Problème essentiel des SGBD
- De multiples algorithmes possibles
- Pas d'algorithme toujours meilleur !
- Nécessite d'un modèle de coût pour choisir le meilleur algorithme
- Choix souvent fait à la compilation