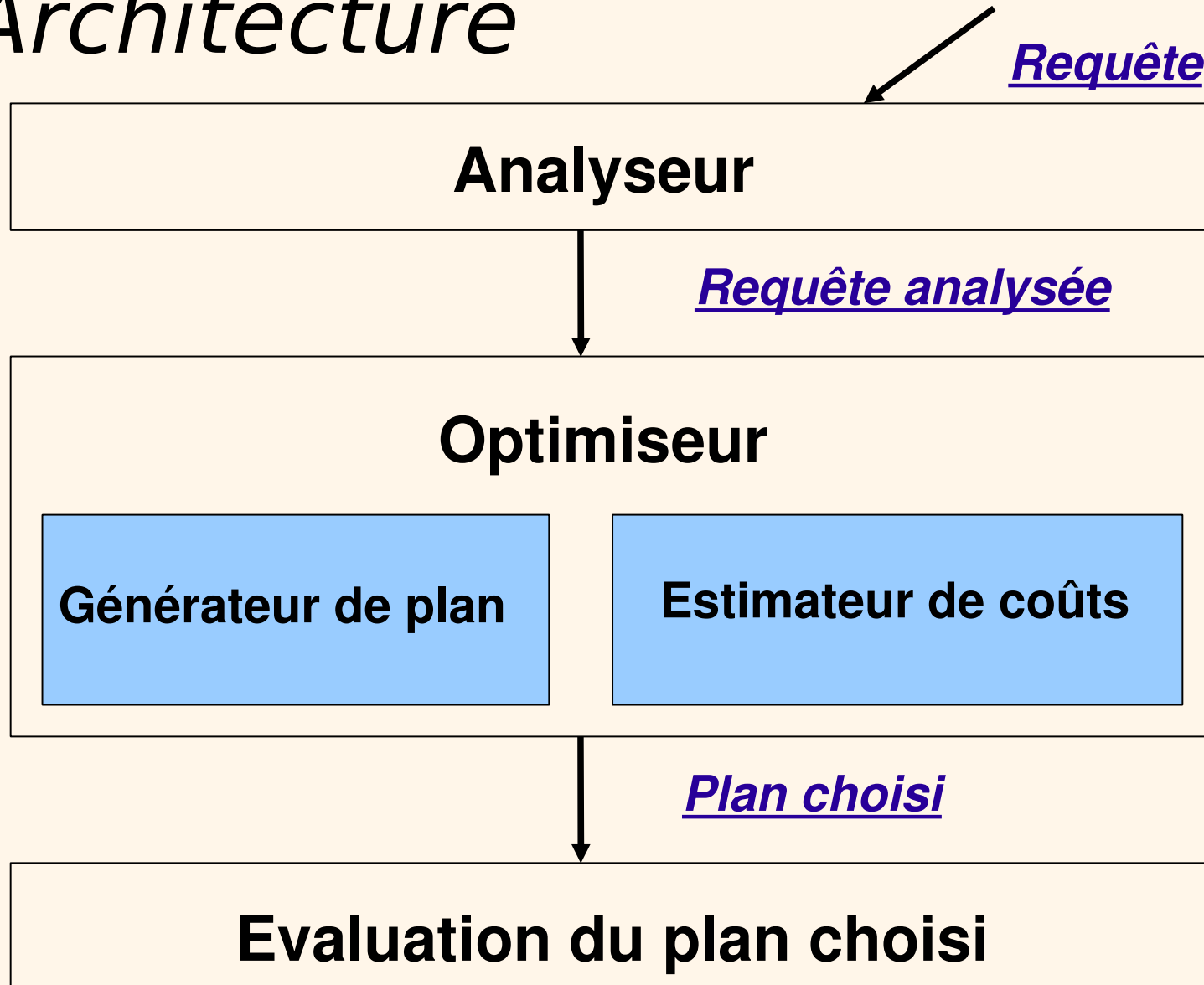


Optimisation de Requêtes

Architecture



Histoire : System R

Impact : Très répandu,
opérationnel < 10 jointures.

Enumeration des plans

Estimation des coûts:

Statistiques, catalogue système,

Estimation du coût des opérateurs

Combine les temps CPU et coût I/O

Coût des opérateurs **et**

de la taille des résultats.

Eviter les produits cartesiens

Schéma pour les exemples

Sailors(*sid*: integer, *sname*: string, *rating*: integer, *age*: real)

Reserves (*sid*: integer, *bid*: integer, *day*: dates, *rname*: string)

Boats(*bid*: integer, *bname*: string, *color*: string)

Reserves:

- Chaque n-uplet 40 bytes, 100 n-uplets par page, 1000 pages.

Sailors:

- Chaque n-uplet 50 bytes, 80 n-uplets par page, 500 pages.

L' Analyseur

Correction syntaxique de la requête

Isole les différents blocs

Génère l'arbre syntaxique passé à l'optimiseur

Exemple

Bloc externe

```
SELECT S.sname  
FROM Sailors S  
WHERE S.age IN  
    (SELECT MAX (S2.age)  
     FROM Sailors S2  
     GROUP BY S2.rating)
```

Bloc imbriqué

Exemple plus général

```
SELECT S.sid, Min(R.day)
FROM Sailors S Reserves R Boats B
WHERE S.sid = R.sid AND R.bid = B.bid AND B.color = 'red'
      AND S.rating = (SELECT MAX (S2.rating)
                      FROM Sailors S2)
GROUP BY S.sid
HAVING COUNT(*) > 1
```

Arbre Syntaxique généré

Π S.sid, MIN(R.day)(

HAVING COUNT(*) > 2(

GROUP BY S.sid (

σ S.sid = R.sid \wedge R.bid = B.bid \wedge B.color = 'red'

\wedge S.rating = *valeur bloc imbriqué*

(S X R X B))))

Optimiseur

Une requête SQL est une collection de **blocs (query blocks)**,

Optimise un bloc à la fois.

Les blocs imbriqués seront traités ultérieurement (dans le cours) .

Optimiseur : génération de plans

Pour chaque bloc l'optimiseur génère un ensemble de plans.

Qu'est ce qu'un plan ?

Programme combinant des méthodes d'évaluation des opérateurs.

Indication de l'enchaînement des méthodes.

Représentation d'un plan = Arbre

Plans : Représentation graphique

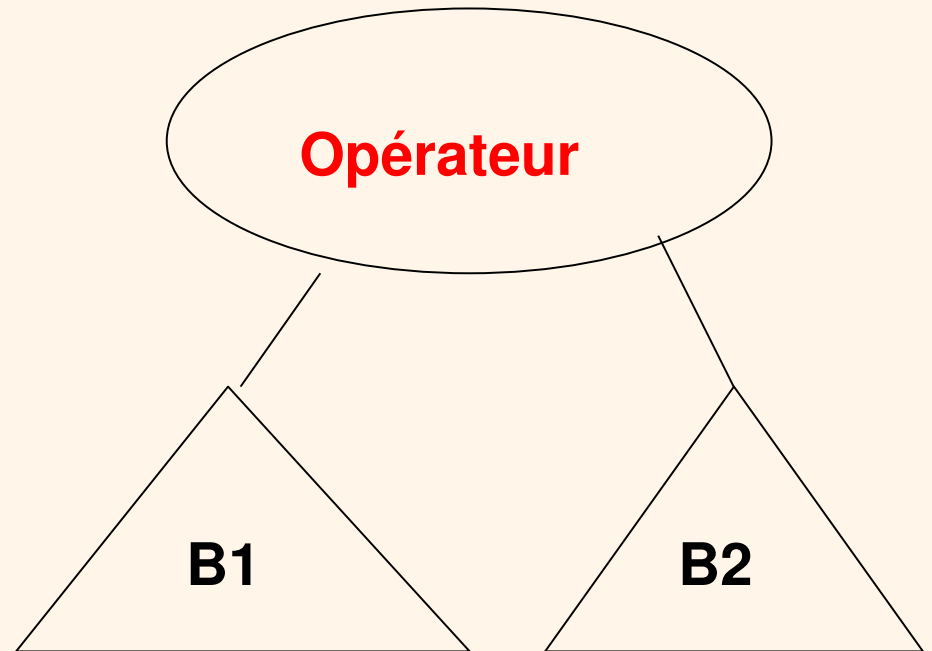
Méthode

Noeud interne:

méthode pour opérateur

entrée : résultats de B1 et B2

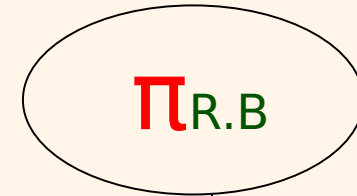
Feuille = table



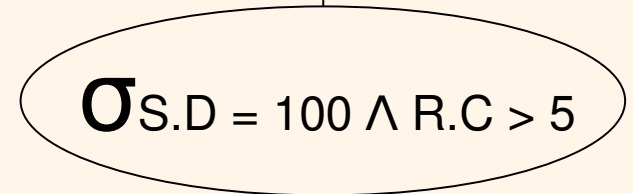
Exécution du plan \approx Parcours en profondeur

Un exemple de plan

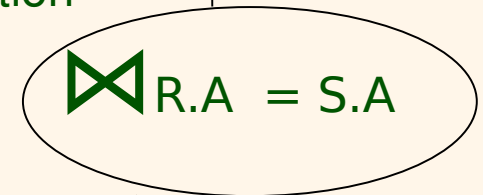
A la volée



A la volée



Double iteration



**SELECT R.B FROM R, S
WHERE R.A = S.A AND
S.D = 100 AND R.C > 5**

Plans

❖ Deux problèmes :

- Etant donnée une requête, **quels plans considérer ?**
 - Algorithme pour trouver dans l'espace de recherche le plan de moindre coût (estimé)
- Comment estimer le **coût d'un plan ?**

❖ **Idéalement** : Trouver le meilleur plan.

❖ **En pratique** : Eviter le pire !

Nous présenterons l'approche de System R

Estimation du coût

- ❖ Estimer le coût de chaque opération dans l'arbre.
- ❖ Estimer la taille des résultats intermédiaires *pour chaque* opération dans l'arbre.
 - information sur relations en entrée.
 - Pour sélections et jointures, suppose indépendance des conditions.

Estimations pour un plan mono-relation

Index I sur la clé primaire “matchant” la condition de sélection:

- *Coût est $H(I)+1$ pour un B+ tree, de l'ordre de 1.2 pour un hash index.*

Index groupant I “matchant” une ou plusieurs sélections

- *$(NPages(I)+NPages(R)) * \text{produits de sélectivité des selects.}$*

Parcours séquentiel du fichier :

$Npages(R).$

ETC... Déjà vu en cours ...

Estimation pour des plans multi-relations

Syntaxe générale

```
SELECT attribute list  
FROM relation list  
WHERE term1 AND ... AND termk
```

Estimation pour des plans multi-relation

Pour un bloc:

Le nombre max. de n-uplets, N , dans le résultat est le produit des cardinalités des relations de la clause FROM.

Le facteur de réduction (RF) associé à chaque *term* reflète l'impact de *term* dans la réduction de la taille du résultat.

*Cardinalité du résultat = N * produit des RF.*

Facteurs de réduction

Comment les estimer ?

En exploitant les statistiques du catalogue

Et la forme du terme.

Forme : attribut = valeur

Si l'on dispose d'un index I sur l'attribut,
On peut approcher RF par

$$1/N_{\text{clés}}(I)$$

En supposant une distribution uniforme des
n-uplets (hypothèse usuelle)

Forme : attribut = valeur

Si l'on ne dispose pas d'index en général le système attribue une valeur arbitraire
(System R : 1/10)

Suivant le système, des statistiques peuvent être maintenues comme le nombre de valeurs distinctes

Forme attribut1 = attribut2

Si l'on dispose d'index I1 et I2 sur attribut1 et attribut2, on peut approcher RF par :

$$1/\text{MAX}(\text{Nclés}(I1), \text{Nclés}(I2))$$

Forme attribut1 = attribut2

Si un seul index est disponible on approche
par $1/N_{cles(I)}$

Si aucun index n'est présent le système
utilise :

$1/10$

Forme : attribut > valeur

Si index I et que la valeur est arithmétique
alors on approche RF par

$$(\text{Maxval}(I) - \text{valeur}) / (\text{Maxlval}(I) - \text{Minval}(I))$$

Sinon le système choisit une valeur
arbitraire inférieure à :

$$1/2$$

Énumération des Plans

Exploiter les équivalences de l'algèbre relationnelle

et considérer deux cas :

Plans Mono-relation

Plans Multi-relation

Algèbre Relationnelle : équivalences

Permettent de réordonnancer les jointures
descendre les projections et les sélections
joins.

Selections:

$$\sigma_{c_1 \wedge \dots \wedge c_n}(R) \equiv \sigma_{c_1}(\dots \sigma_{c_n}(R))$$

$$\sigma_{c_1}(\sigma_{c_2}(R)) \equiv \sigma_{c_2}(\sigma_{c_1}(R))$$

Projections:

$$\pi_{a_1}(R) \equiv \pi_{a_1}(\dots (\pi_{a_n}(R)))$$

Jointures:

$$(R \bowtie S) \bowtie S \equiv R \bowtie (S \bowtie T)$$

$$R \bowtie S \equiv S \bowtie R$$

Autres Equivalences

Une projection commute avec une sélection si elle porte sur les attributs projetés.

Sélection entre des attributs de deux arguments d' un produit cartésien transforme celui ci en jointure.

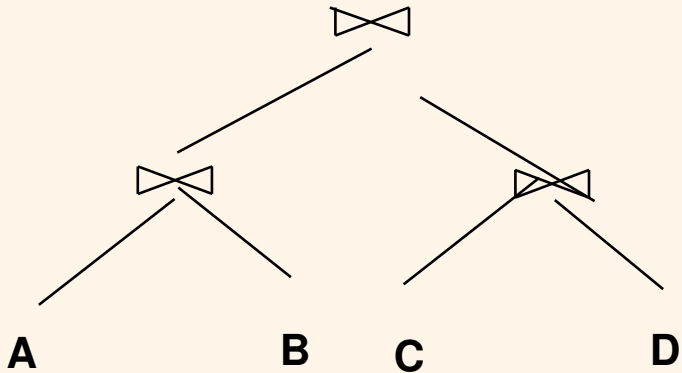
Etc ...

Plans mono-relation

Pour des requêtes mono-relation consiste en une combinaison de sélections, projections, et agrégats

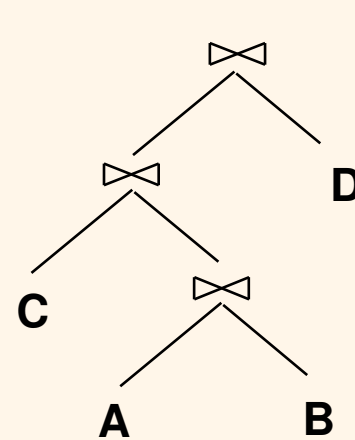
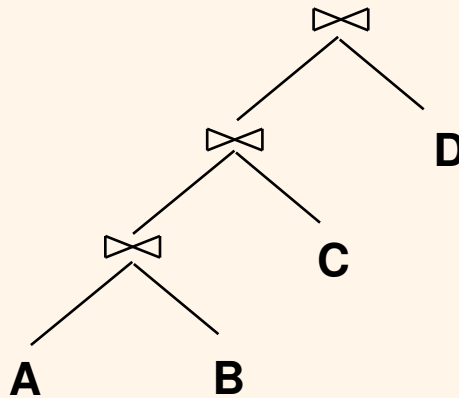
- Chaque chemin d'accès (parcours / index) est considéré et celui de moindre coût estimé est sélectionné..
- Les différentes opérations sont enchaînées (par ex., si un index est utilisé pour une sélection, la projection est effectuée pour chaque n-uplet et les n-uplets résultats sont traités en *pipeline ou à la volée*).

Plans multi-relations



**Plan
touffu**

**Plan en
peigne**



Plan linéaire

Plans multi-relations

Le nombre de plans augmente
**exponentiellement avec le
nombre de jointures.**

*Il faut restreindre l'espace de
recherche*

Plans multi-relations

Décision fondamentale de System R:
*seuls les plans en peigne sont
considérés.*

Les plans en peigne permettent de
générer tous les plans évaluables *à
la volée (pipeline)*

Les résultats intermédiaires ne sont pas
écrits dans des fichiers temporaires.

- Tous les plans en peigne ne sont pas
évaluables en pipeline

Enumeration des plans en peigne

- ❖ Les plans diffèrent seulement par :
 - L'ordre des relations,
 - les méthodes d'accès pour chaque relation et
 - l'algorithme de jointure pour chaque jointure.

Énumération des plans en peigne (une stratégie)

Énumérés en N passes (si N relations jointes):

- **Passé 1:** trouver le meilleur plan 1-relation pour chaque relation.
- **Passé 2:** trouver la meilleure manière de joindre le résultat de chaque plan 1-relation (externe) avec une autre relation (*tous les plans 2-relation.*)
- **Passé N :** trouver la meilleure manière de joindre le résultat des plans $(N-1)$ -relation (externe) à la N ème relation. (*tous les plans N -relation.*)

Énumération des plans en peigne (une stratégie)

Pour chaque sous ensemble de relation
retenir

- Celui de moindre coût, et éventuellement
- Celui de moindre coût parmi les plans produisant des résultats ordonnés.

Enumération des plans suite

- ❖ **ORDER BY, GROUP BY, aggregates** pris en compte à la fin en utilisant un plan ordonné ou un tri externe.
- ❖ Un plan à N-1 relations n'est pas combiné avec une autre relation à moins qu'il y ait une condition de jointure et que tous les prédicats de la clause WHERE aient été exploités.
 - i.e., **éviter les produits Cartésien si possible.**
- ❖ Malgré l'élagage de l'espace de recherche cette approche demeure **exponentielle** en fonction du nombre de tables.

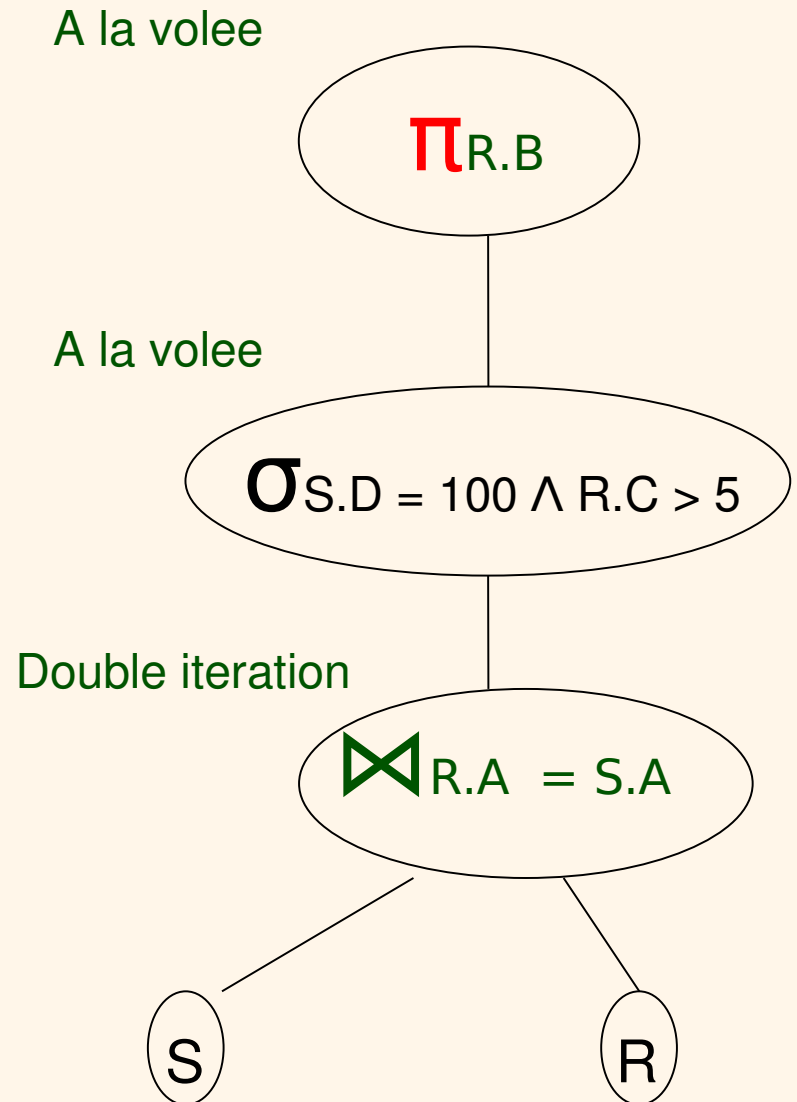
Exemple

**SELECT R.B FROM R, S
WHERE R.A = S.A AND
S.D = 100 AND R.C > 5**

Coût = 500 + 500 x 1000

pas le meilleur plan

pas le plus mauvais



Exemple : un autre plan

**SELECT R.B FROM R, S
WHERE R.A = S.A AND
S.D = 100 AND R.C > 5**

Coût : (5 pages de buffer)

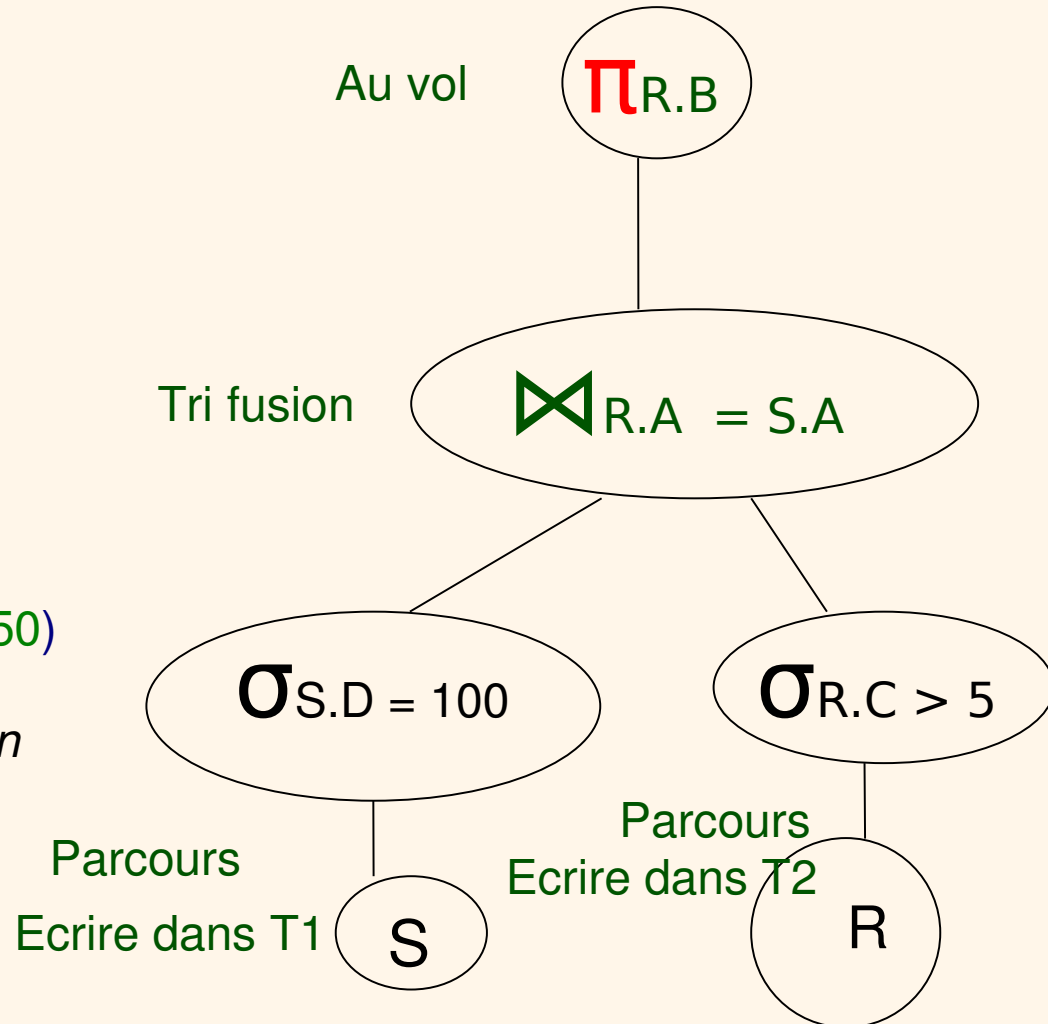
(500+250) + (1000+10) +

selection selection

(2 x 3 x 250) + (2 x 2 x 10) + (10 + 250)

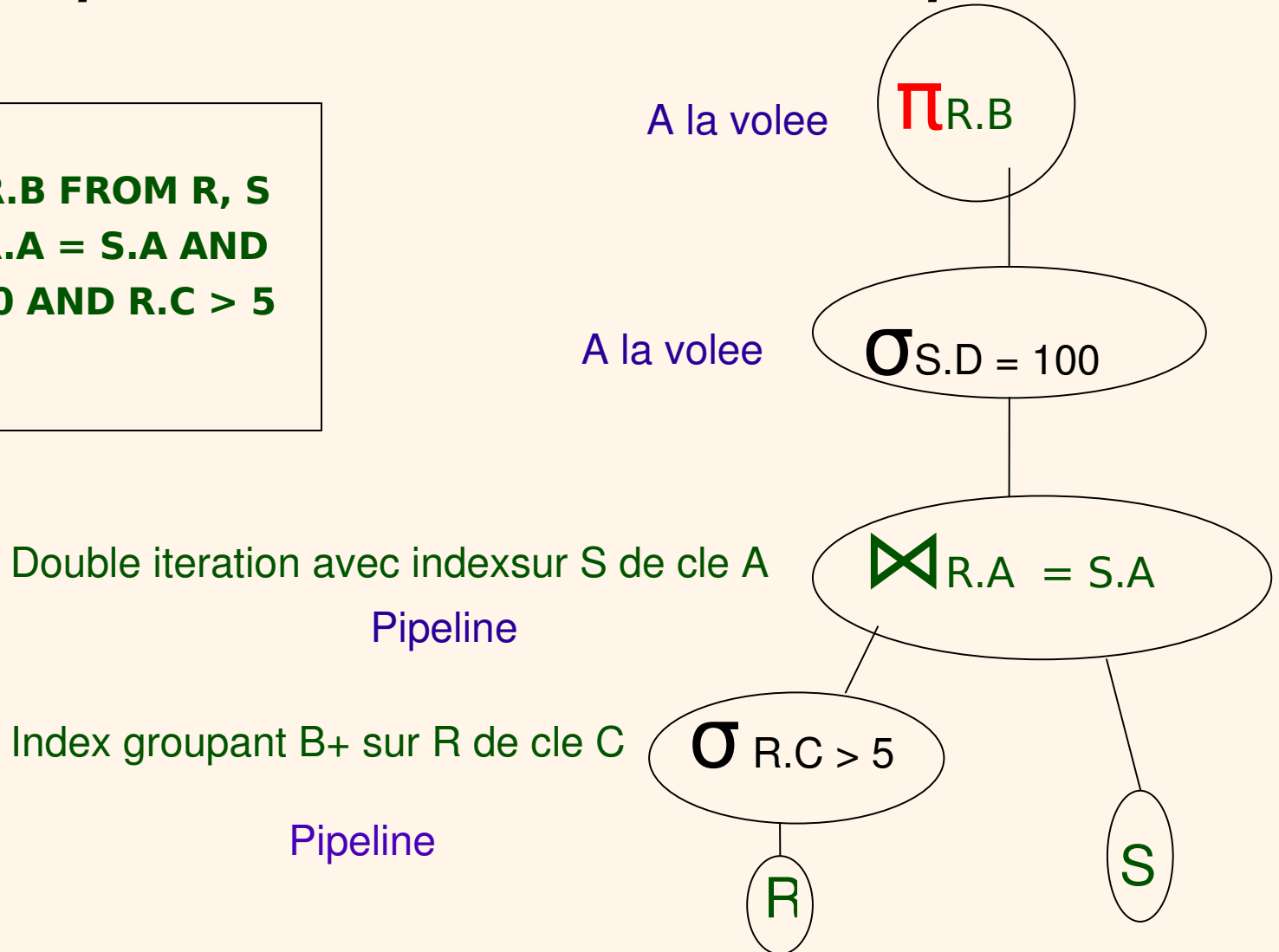
tri tri fusion

3569 E/S



Exemple : Un troisième plan

**SELECT R.B FROM R, S
WHERE R.A = S.A AND
S.D = 100 AND R.C > 5**



Exemple en détail

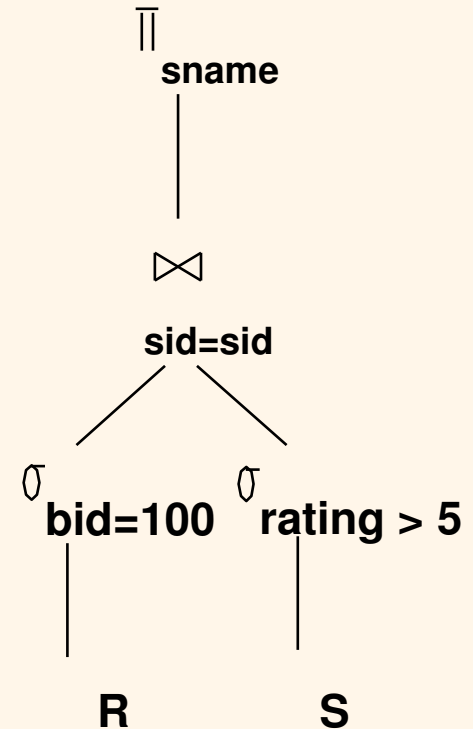
Passel:

- **S:** L'arbre B+ matche $rating > 5$, et est probablement moins cher. Cependant si l'on s'attend à ce que cette sélection retrouve beaucoup de n-uplets et si l'index n'est pas groupant un parcours peut s'avérer moins cher.
 - Le plan exploitant l'arbre B+ est garde car les n-uplets sont ordonnés suivant *rating* order.
- **R:** L'arbre B+ sur *bid* matches $bid=100$. C'est moins cher

S:
B+ tree on *rating*

Hash on *sid*

R:
B+ tree on *bid*



Exemple suite

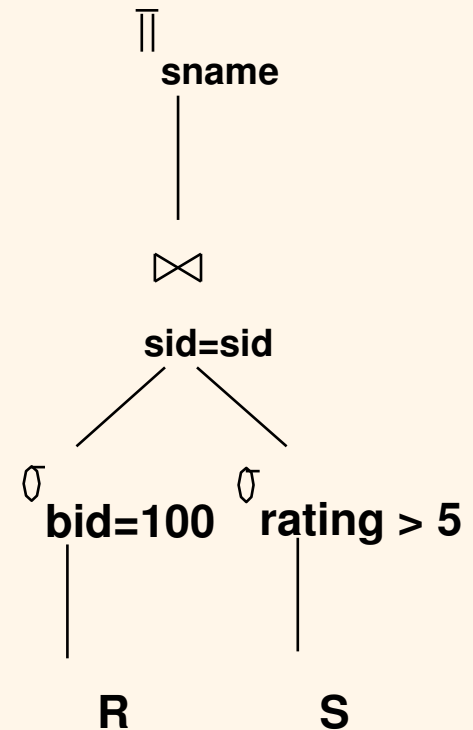
Passe 2:

Considere chaque plan garde en passe 1
comme relation (input) externe et
determine comment effectuer la jointure
avec la relation restante.

e.g., R en externe : Hash index
peut etre utilise pour obtenir les
n-uplets de S satisfaisant $sid =$
valeur externe de sid .

S:
B+ tree on
rating
Hash on *sid*

R:
B+ tree on
bid



Requêtes imbriquées

- 1) Le bloc interne est optimisé indépendamment
- 2) Le bloc externe est optimisé en intégrant le coût "d'appel" au bloc interne.
- 3) Cet ordonnancement implicite des blocs implique que de bonnes stratégies ne seront jamais considérées.

```
SELECT S.sname
FROM Sailors S
WHERE EXISTS
  (SELECT *
   FROM Reserves R
   WHERE R.bid=103
   AND R.sid=S.sid)
```

Optimisation du bloc interne

Bloc imbriqué :

```
SELECT *  
FROM Reserves R  
WHERE R.bid=103  
      AND S.sid= valeur externe
```

le bloc interne est optimisé
indépendamment en considérant le n-uplet
externe comme valeur d'une condition de
sélection.

Requêtes imbriquées : moralité

Equivalent non imbriqué:

```
SELECT S.sname  
FROM Sailors S,  
Reserves R  
WHERE S.sid=R.sid  
AND R.bid=103
```

***La version non imbriquée de la requête
est toujours mieux optimisée !!!.***

Conclusion

- ❖ L' optimisation de requetes est une tache importante pour tout SGBD relationnel.
- ❖ Necessaire d'en comprendre les mecanismes afin d'etre en mesure d'appréhender l'impact sur les performances d'une conception donnee(relations, index) pour une charge de travail (workload – ensemble de requetes) determine.

Conclusion

❖ Deux aspects :

- Considerer l'ensemble des differents plans.
 - Elagage (plans en peigne)
- Estimer les couts de chaque plan.
 - Taille des resultats intermediaires et couts des algorithmes utilises.
 - *Points cless*: Statistiques, index, implementation des methodes.

Conclusion (Fin)

❖ Requetes mono-relation:

- Tous les chemins d'accès sont consideres, le moins couteux est choisi.
- *Point cle*: Selections qui *match un* index, que la cle de l'index ait tous les champs requis et/ou fournisse les n-uplets dans un ordre souhaite.

❖ Requetes multi-relation:

- Tous les plans mono relation sont enumeres.
 - Selections/projections considerees le plus tot possible.
- Puis les autres passes sont executees etc ...
- A chaque passe, pour chaque sous ensemble de relations seul le meilleur plan pour chaque ordre pertinent des n-uplets est retenu..