

# BDR

## Master d'Informatique (SAR)

Cours 2- Optimisation de requêtes

Stéphane.Gańczarski

Stephane.Gancarski@lip6.fr

# Optimisation de requêtes

- Traitement et exécution de requêtes
- Implémentation des opérateurs relationnels
- Restructuration de la requête
- Coût des opérations
- Optimisation du coût
- Espace de recherche
- Stratégie de recherche

# Exemple

EMP(ENO, ENAME, TITLE)

PROJECT(PNO, PNAME, BUDGET)

WORKS(ENO, PNO, RESP, DUR)

WORKS.(ENO) clé étrangère vers EMP

WORKS.(PNO) clé étrangère vers PROJECT

# Problème

Soit la requête

pour chaque projet de budget  $> 250$  qui emploie plus de 2 employés, donner le nom et le titre des employés

Comment l'exprimer en SQL ?

# Problème

Soit la requête

pour chaque projet de budget  $> 250$  qui emploie plus de 2 employés, donner le nom et le titre des employés

```
SELECT  Ename, Title
FROM    Emp, Project, Works
WHERE   Budget > 250000
AND     Emp.Eno=Works.Eno
AND     Project.Pno=Works.Pno
AND     Project.Pno IN
        (SELECT  w.Pno
         FROM     Works w
         GROUP BY w.Pno
         HAVING   COUNT(*) > 2)
```

Comment l'exécuter?

# Un plan d'exécution possible

$T_1 \leftarrow$  Lire la table **Project** et sélectionner les tuples de  
**Budget > 250**

$T_2 \leftarrow$  Joindre  $T_1$  avec la relation **Works** (**sur PNO**)

$T_3 \leftarrow$  Joindre  $T_2$  avec la relation **Emp** (**sur ENO**)

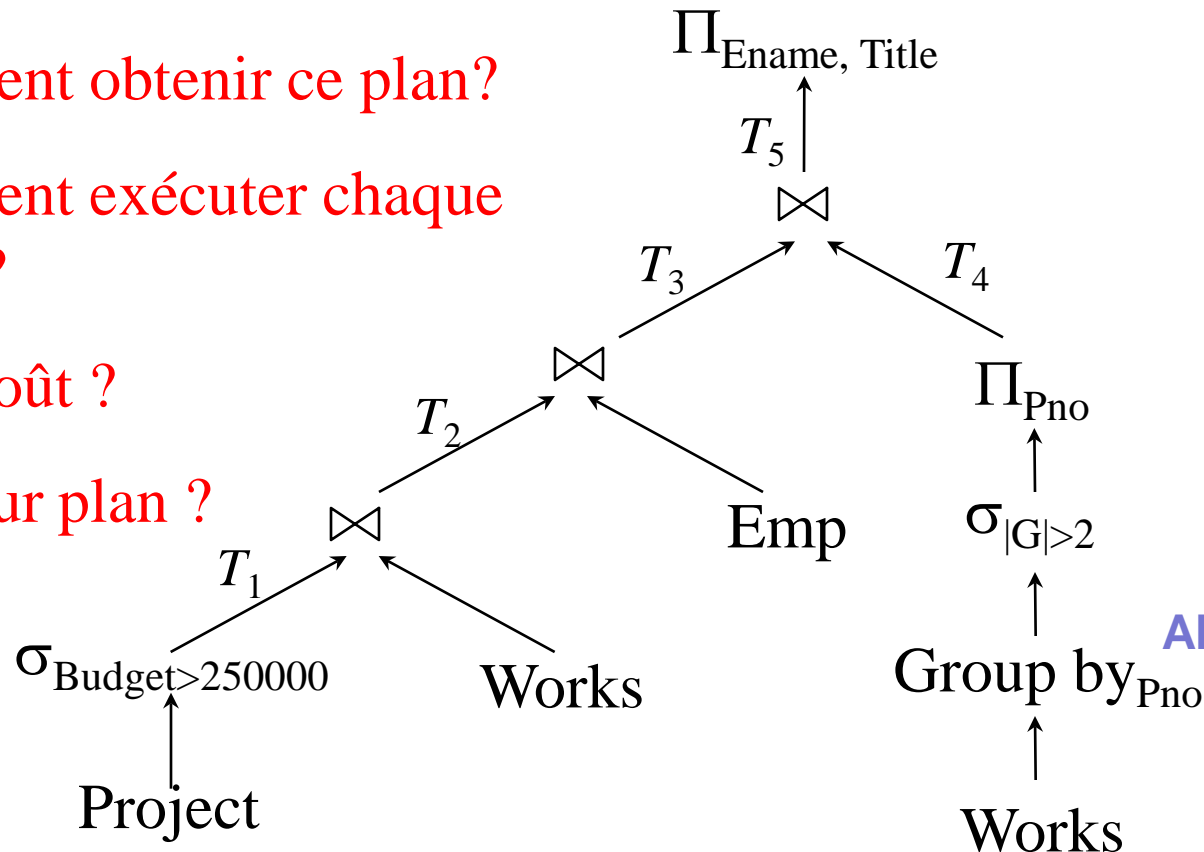
$T_4 \leftarrow$  Grouper les tuples de **Works** sur **Pno** et pour les  
groupes qui ont plus de 2 tuples, projeter sur **Pno**

$T_5 \leftarrow$  Joindre  $T_3$  avec  $T_4$

Projeter  $T_5$  sur **Ename**, **Title**

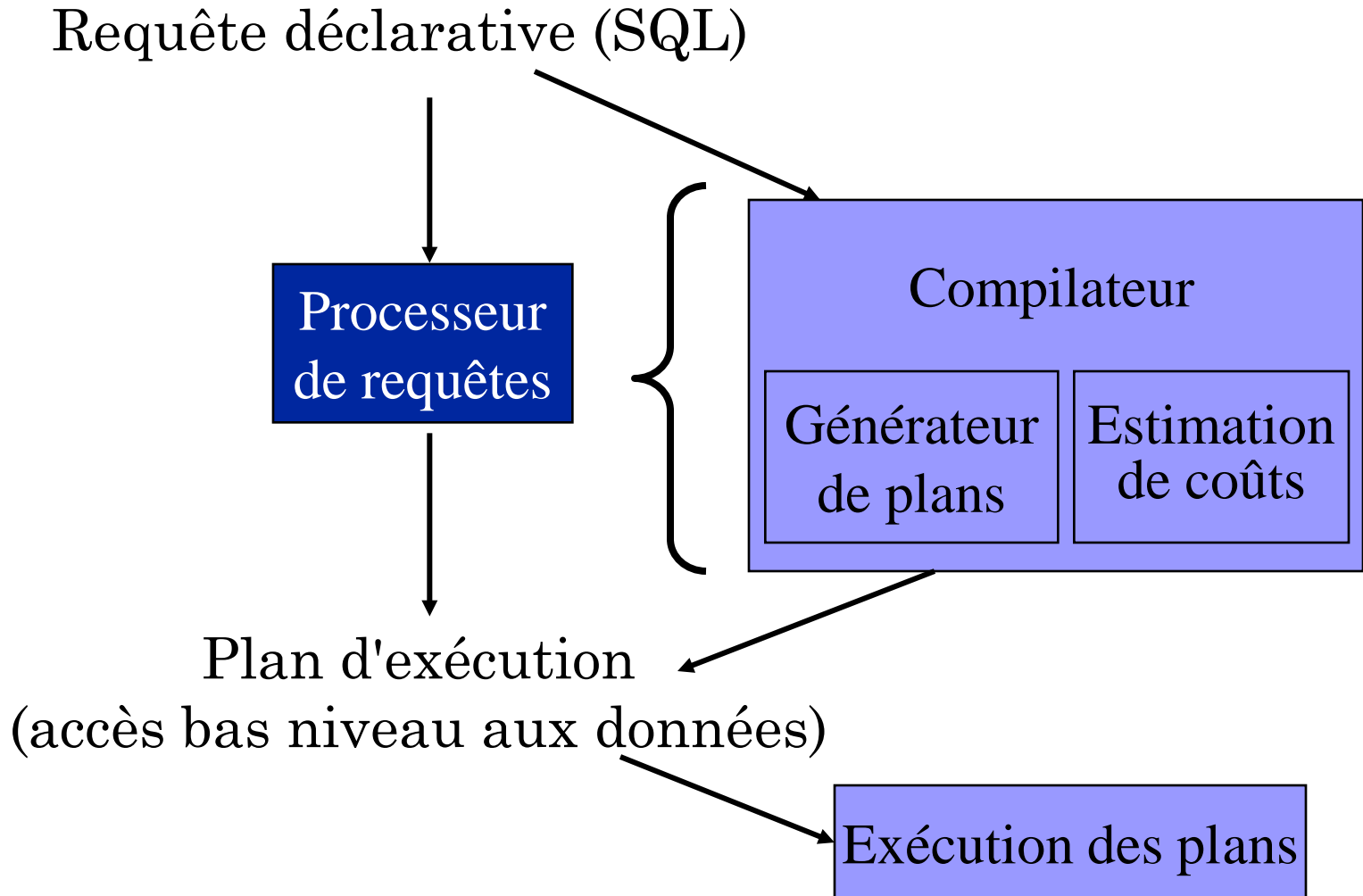
# Représentation graphique

- Comment obtenir ce plan?
- Comment exécuter chaque noeud?
- Quel coût ?
- Meilleur plan ?



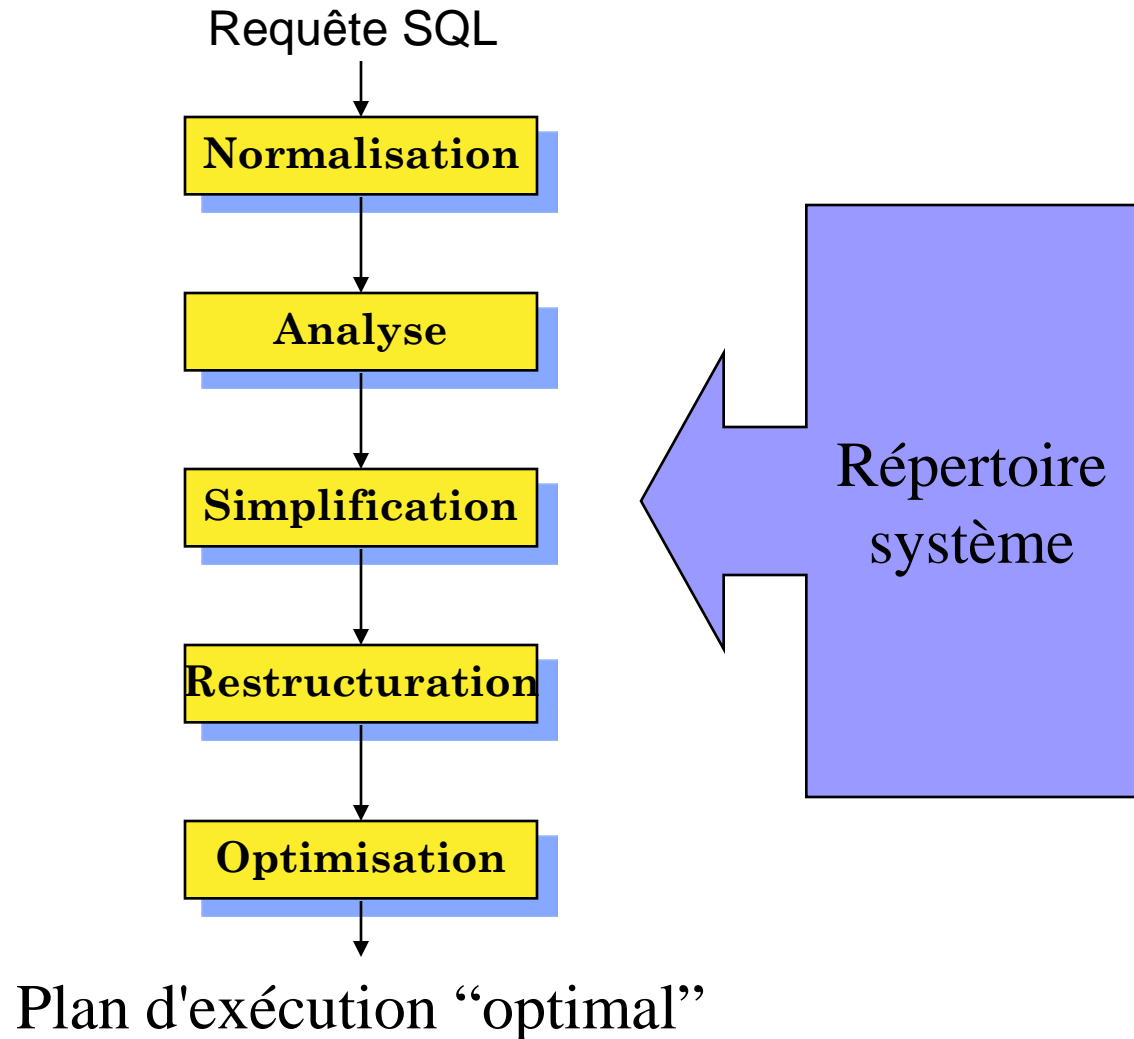
$\Pi_{\text{Ename, Title}}(\Pi_{\text{Pno}}(\sigma_{|G| > 2} \text{Group}_{\text{Pno}}(\text{Works})))$   
 $\bowtie (\text{Emp} \bowtie ((\sigma_{\text{Budget} > 250000} \text{Project}) \bowtie \text{Works})))$

# Traitement des requêtes





# Etapes du traitement des requêtes



# Normalisation de requête

- Analyse lexicale et syntaxique
  - vérification de la validité de la requête
  - vérification des attributs et relations
  - vérification du typage de la qualification
- Mise de la requête en **forme normale**
  - forme normale conjonctive
$$(p_{11} \vee p_{12} \vee \dots \vee p_{1n}) \wedge \dots \wedge (p_{m1} \vee p_{m2} \vee \dots \vee p_{mn})$$
  - forme normale disjonctive
$$(p_{11} \wedge p_{12} \wedge \dots \wedge p_{1n}) \vee \dots \vee (p_{m1} \wedge p_{m2} \wedge \dots \wedge p_{mn})$$
  - OR devient union
  - AND devient jointure ou sélection

# Simplification

- Pourquoi simplifier?
  - plus une requête est simple, plus son exécution peut être efficace
- Comment? en appliquant des transformations
  - élimination de la redondance
    - règles d'idempotence
$$p_1 \wedge \neg(p_1) \equiv \text{faux}$$
$$p_1 \wedge (p_1 \vee p_2) \equiv p_1$$
$$p_1 \vee \text{faux} \equiv p_1$$
$$\dots$$
  - application de la transitivité (att1=att2 ,att2=att3)
  - utilisation des règles d'intégrité
    - CI : att1 <100 Q: ... where att1 > 1000...

# Exemple de simplification

```
SELECT      Title
FROM        Emp
WHERE       Ename = 'J. Doe' P1
OR          (NOT(Title = 'Programmer') ¬P2)
AND         (Title = 'Programmer' P2
OR          Title = 'Elect. Eng.') P3
AND         NOT(Title = 'Elect. Eng.')) ¬P3
```



**$P1 \vee (\neg P2 \wedge (P2 \vee P3) \wedge \neg P3)$**

```
SELECT      Title
FROM        Emp
WHERE       Ename = 'J. Doe'
```

# Implémentation des opérateurs

Rappel: accès disque >> accès mémoire (négligeable)

- **Sélection** (avec  $R$  contenant  $n$  pages disques)
  - parcours séquentiel (scan)
    - le nombre d'accès disques est en  $O(n)$
  - Parcours (scan) avec index
    - index  $B^+$  –  $O(\log n)$  /\* hauteur (log base  $k$ ) + nb pages résultat
    - index haché –  $O(1)$  /\* statique en supposant une bonne répartition.  $O(2)$  hachage dynamique
- **Projection**
  - sans élimination des doubles –  $O(n)$
  - avec élimination des doubles
    - en triant –  $O(n \log n)$
    - en hachant –  $O(n+t)$  où  $t$  est le nombre de pages du fichier haché

# Implémentation des opérateurs

- Jointure

- boucle imbriquée (nested loop):  $T = R \bowtie S$

```
foreach tuple  $r \in R$  do
```

```
    foreach tuple  $s \in S$  do
```

```
        if  $r == s$  then  $T = T + \langle r, s \rangle$ 
```

- $O(n*m)$

- amélioration possible pour réduire les accès disques

- boucles imbriquées par pages ou blocs : permet de joindre chaque  $n$ -uplet de  $R$  avec non plus un seul  $n$ -uplet de  $S$ , mais avec tous (on suppose  $p$ ) ceux qui tiennent en MC :  $O(n*m/p)$

# Implémentation des opérateurs

- Jointure

- boucle imbriquée et index sur attribut de jointure de  $S$  (cas typique : jointure sur clé étrangère)

```
foreach tuple  $r \in R$  do
```

```
    accès aux tuples  $s \in S$  par index
```

```
    foreach tuple  $s$  do
```

```
         $T = T + \langle r, s \rangle$  /*  $O(n+M)$ ,  $M = \text{card}(R) * k(\text{hauteur})$ 
```

- tri-fusion
  - trier  $R$  et  $S$  sur l'attribut de jointure : tri externe  $O(n \log(n))$
  - fusionner les relations triées :  $O(n+m)$
- hachage
  - hacher  $R$  et  $S$  avec la même fonction de hachage
  - pour chaque paquet  $i$  de  $R$  et  $i$  de  $S$ , trouver les tuples où  $r=s$

# Tri externe

- Algo (de base, voir version améliorée en TD)
  - Trier des paquets de  $k$  pages tenant en mémoire disponible  
 $n/k$  paquets,  $2n$  E/S
  - Charger les premières pages de chaque paquet et trier
    - Dès qu'une page est vide, charger la suivante du même paquet
    - On obtient des paquets de  $k^2$  pages triés  
 $n/k^2$  paquets,  $2n$  E/S
  - Continuer jusqu'à obtenir un paquet de  $k^i \geq n$  pages
- Coût total
  - A chaque étape on lit et écrit toutes les données :  $2n$  E/S
  - Nombre d'étape :  $\log_k(n)$



# Optimisation

- Elaborer des plans
  - arbre algébrique, restructuration, ordre d'évaluation
- Estimer leurs coûts
  - fonctions de coût
    - en terme de temps d'exécution
    - coût I/O + coût CPU
    - poids très différents
      - par ex. coût I/O = 1000 \* coût CPU
- Choisir le meilleur plan
  - Espace de recherche : ensemble des expressions algébriques équivalentes pour un même requête
  - algorithmes de recherche:
    - parcourir l'espace de recherche
    - algorithmes d'optimisation combinatoire

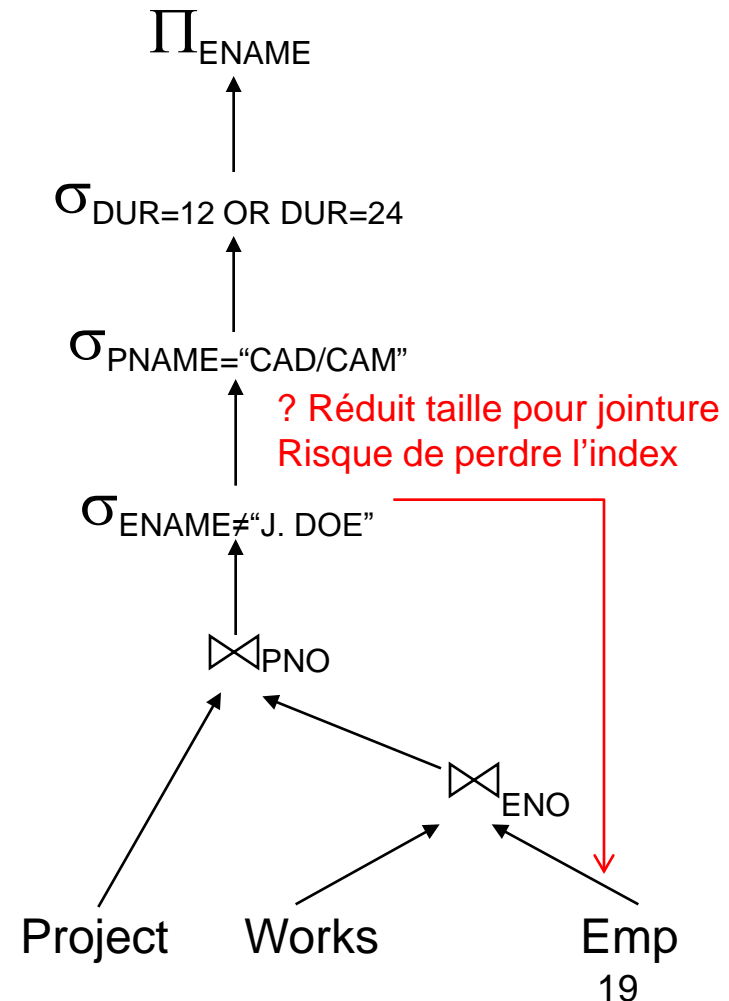
# Restructuration

- Objectif : choisir l'ordre d'exécution des opérations algébriques (élaboration du plan logique).
- Conversion en arbre algébrique
- Transformation de l'arbre (optimisation)
  - règles de transformation (équivalence algébriques),
  - estimation du coût des opérations en fonction de la taille
  - Estimation du résultat intermédiaire (taille et ordre? )
  - En déduire l'ordre des jointures

# Restructuration

- Conversion en arbre algébrique
- Exemple

```
SELECT  Ename
FROM    Emp, Works, Project
WHERE   Emp.Eno=Works.Eno
AND     Works.Pno=Project.Pno
AND     Ename NOT='J.Doe'
AND     Pname = 'CAD/CAM'
AND     (Dur=12 OR Dur=24)
```



# Calcul du coût d'un plan

- La fonction de coût donne les temps I/O et CPU
  - nombre d'instructions et d'accès disques
- Estimation de la taille du résultat de chaque noeud
  - Permet d'estimer le coût de l'opération suivante
  - sélectivité des opérations – “facteur de réduction”
  - propagation d'erreur possible
- Estimation du coût d'exécution de chaque noeud de l'arbre algébrique
  - utilisation de pipelines ou de relations temporaires importante
    - Pipeline : les tuples sont passés directement à l'opérateur suivant.
      - Pas de relations intermédiaires (petites mémoires, ex. carte à puce).
      - Permet de paralléliser (BD réparties, parallèle)
      - Intéressant même pour cas simples :  $\sigma_{F \wedge F'}(R)$ , index sur  $F' \rightarrow \sigma_F(\sigma_{F'}(R))$
    - Relation temporaire : permet de trier

# Statistiques

- Relation
  - cardinalité :  $\text{card}(R)$
  - taille d'un tuple : largeur de  $R$
  - fraction de tuples participant une jointure / attribut
  - ...
- Attribut
  - cardinalité du domaine
  - nombre de valeurs distinctes  $\text{distinct}(A,R) = \Pi_A(R)$
  - Valeur max, valeur min
- Hypothèses
  - indépendance entre différentes valeurs d'attributs
  - distribution uniforme des valeurs d'attribut dans leur domaine
  - Sinon, il faut maintenir des histogrammes
    - Equilarge : plages de valeurs de même taille
    - Equiprofond: plages de valeurs contenant le même nombre d'occurrence
    - Equiprofond meilleur pour les valeurs fréquentes (plus précis)
- Stockage :
  - Les statistiques sont des métadonnées, stockées sous forme relationnelle (voir TME « prise en main »)
  - Rafraîchies périodiquement, pas à chaque fois.

# Tailles des relations intermédiaires

- Sélection

$$taille(R) = card(R) * largeur(R)$$

$$card(\sigma_F(R)) = SF_{\sigma}(F) * card(R)$$

où

$$SF_{\sigma}(A = valeur) = \frac{1}{card(\Pi_A(R))}$$

$$SF_{\sigma}(A > valeur) = \frac{max(A) - valeur}{max(A) - min(A)}$$

$$SF_{\sigma}(A < valeur) = \frac{valeur - min(A)}{max(A) - min(A)}$$

$$SF_{\sigma}(p(A_i) \wedge p(A_j)) = SF_{\sigma}(p(A_i)) * SF_{\sigma}(p(A_j))$$

$$SF_{\sigma}(p(A_i) \vee p(A_j)) = SF_{\sigma}(p(A_i)) + SF_{\sigma}(p(A_j)) - (SF_{\sigma}(p(A_i)) * SF_{\sigma}(p(A_j)))$$

$$SF_{\sigma}(A \in valeur) = SF_{\sigma}(A = valeur) * card(\{valeurs\})$$

# Tailles des relations intermédiaires

- Projection

$$\text{card}(\Pi_A(R)) = \text{distinct}(A,R) \leq \text{card}(R)$$

- Produit cartésien

$$\text{card}(R \times S) = \text{card}(R) * \text{card}(S)$$

- Union

$$\max\{\text{card}(R), \text{card}(S)\} \leq \text{card}(R \cup S) \leq \text{card}(R) + \text{card}(S)$$

- Différence

$$0 \leq \text{card}(R-S) \leq \text{card}(R)$$

# Tailles des relations intermédiaires

- Jointure

- cas particulier:  $A$  est clé de  $R$  et  $B$  est clé étrangère de  $S$ ;

$$\text{card}(R \bowtie_{A=B} S) = \text{card}(S)$$

- plus généralement

$$\text{card}(R \bowtie S) = SF_J * \text{card}(R) * \text{card}(S)$$

Comment l'obtenir ?



# Espace de recherche

- Caractérisé par les plans “équivalents” pour une même requête
  - ceux qui donnent le même résultat
  - générés en appliquant des règles de transformation
- Le coût de chaque plan est en général différent
- L'ordre des jointures est important

# Règles de transformation

- Commutativité des opérations binaires

- $R \times S \equiv S \times R$
- $R \bowtie S \equiv S \bowtie R$
- $R \cup S \equiv S \cup R$

- Associativité des opérations binaires

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

- Idempotence des opérations unaires

- $\Pi_{A'}(\Pi_{A''}(R)) \equiv \Pi_{A' \cap A''}(R)$
- $\sigma_{p_1(A_1)}(\sigma_{p_2(A_2)}(R)) \equiv \sigma_{p_1(A_1) \wedge p_2(A_2)}(R)$

où  $R[A, A_1, A_2]$  et  $A' \subseteq A, A'' \subseteq A$

# Règles de transformation

Commutativité de la sélection et de la projection (pas toujours)

Commutativité de la sélection avec les opérations binaires

- $\sigma_{p(A)}(R \times S) \equiv (\sigma_{p(A)}(R)) \times S$
- $\sigma_{p(A_i)}(R \bowtie_{(A_j, B_k)} S) \equiv (\sigma_{p(A_i)}(R)) \bowtie_{(A_j, B_k)} S$
- $\sigma_{p(A_i)}(R \cup T) \equiv \sigma_{p(A_i)}(R) \cup \sigma_{p(A_i)}(T)$

où  $A_i$  appartient à  $R$  et  $T$

Commutativité de la projection avec les opérations binaires

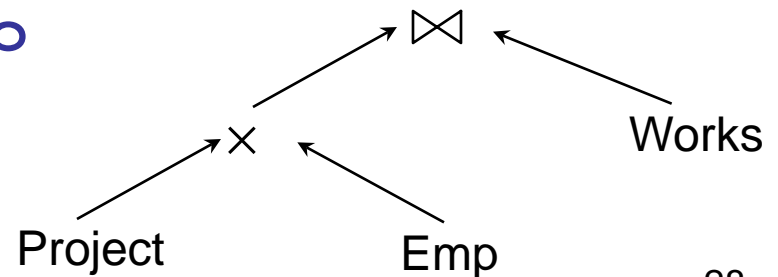
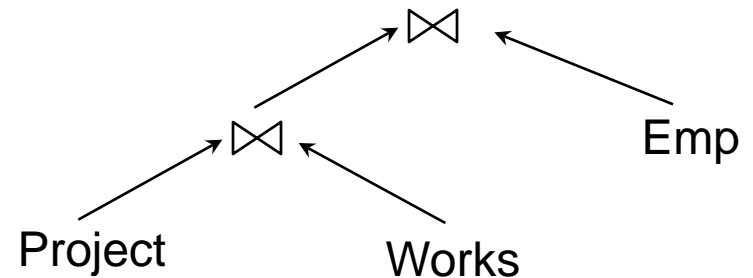
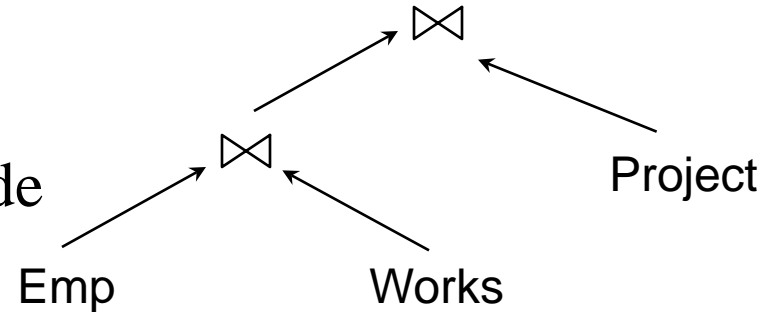
- $\Pi_C(R \times S) \equiv \Pi_{A'}(R) \times \Pi_{B'}(S)$
- $\Pi_C(R \bowtie_{(A_j, B_k)} S) \equiv \Pi_{A'}(R) \bowtie_{(A_j, B_k)} \Pi_{B'}(S)$
- $\Pi_C(R \cup S) \equiv \Pi_C(R) \cup \Pi_C(S)$

où  $R[A]$  et  $S[B]$ ;  $C = A' \cup B'$  où  $A' \subseteq A$ ,  $B' \subseteq B$ ,  $A_j \subseteq A'$ ,  $B_k \subseteq B'$

# Arbres de jointures

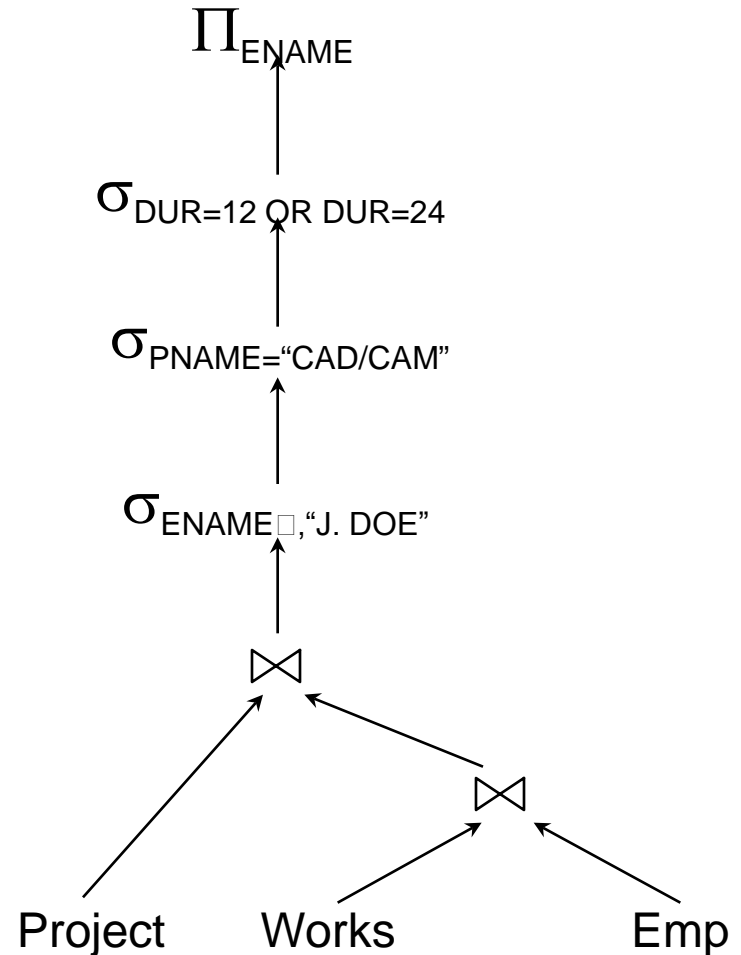
- Avec  $N$  relations, il y a  $O(N!)$  arbres de jointures équivalents qui peuvent être obtenus en appliquant les règles de *commutativité* et d'*associativité*

```
SELECT  Ename, Resp
FROM    Emp, Works, Project
WHERE   Emp.Eno=Works.Eno
AND
        Works.PNO=Project.PNO
```

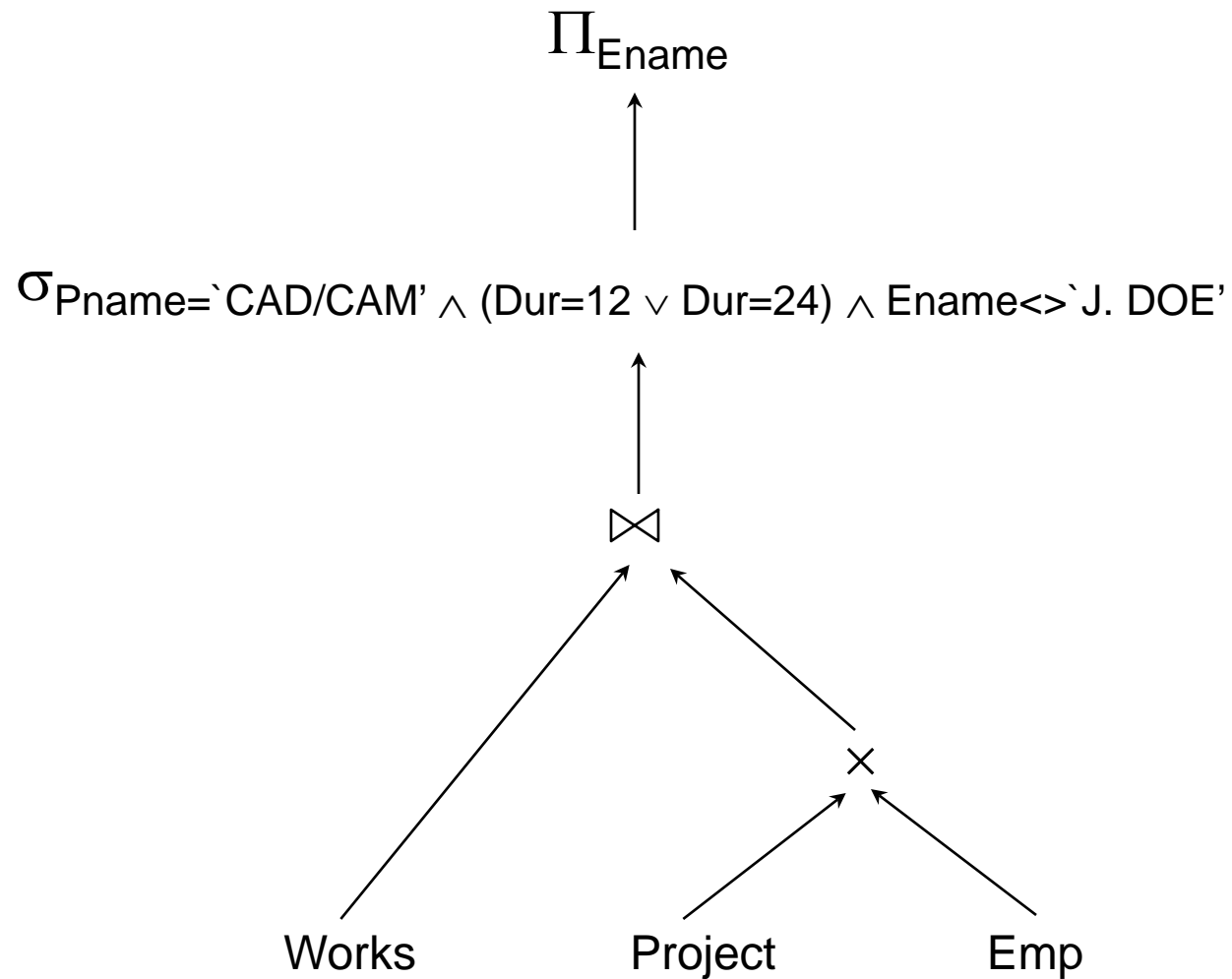


# Example

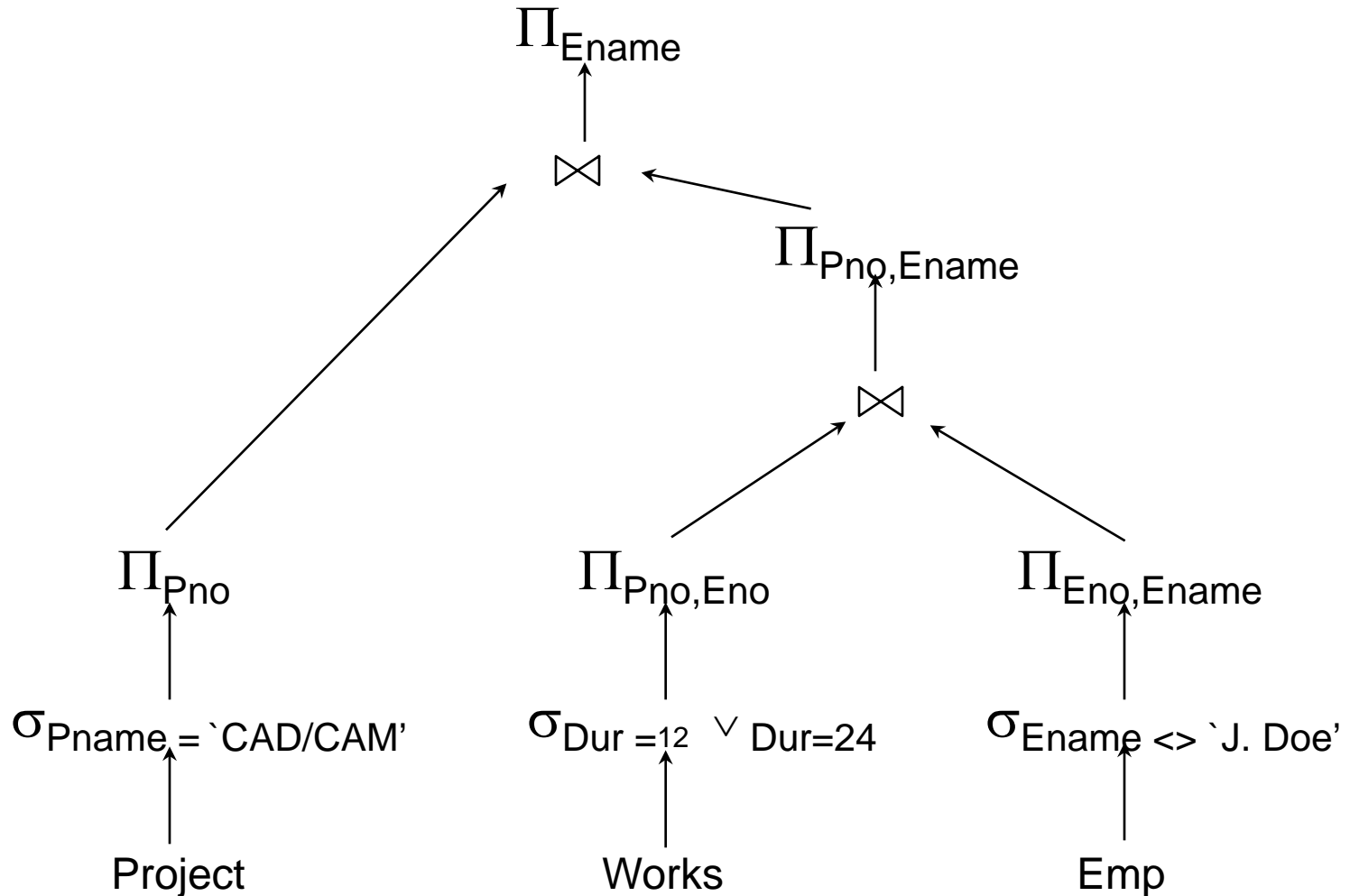
```
SELECT Ename
FROM Project p, Works w,
      Emp e
WHERE w.Eno=e.Eno
AND   w.Pno=p.Pno
AND   Ename<>`J. Doe`
AND   p.Pname=`CAD/CAM`
AND   (Dur=12 OR Dur=24)
```



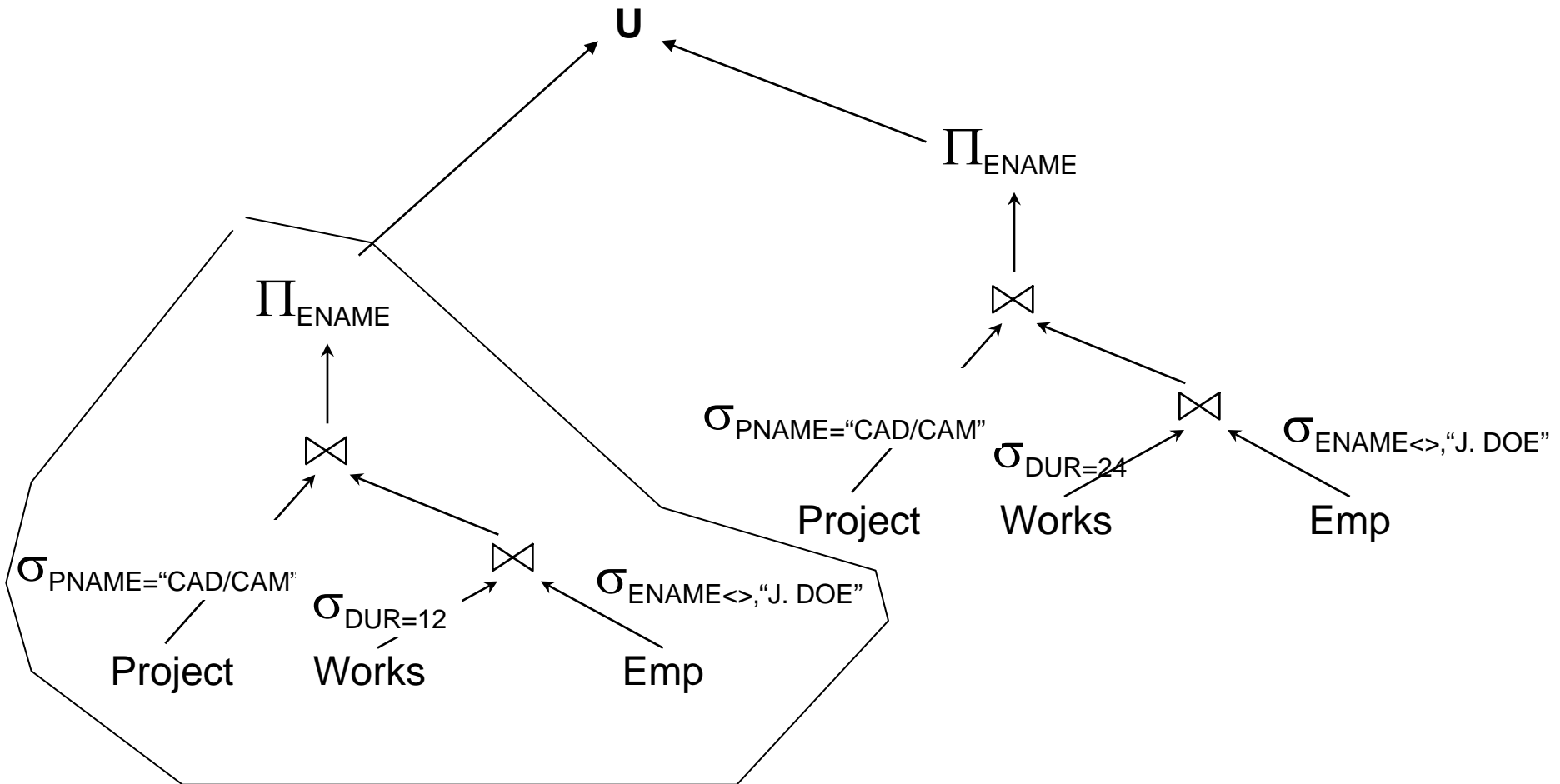
# Requête équivalente



# Autre requête équivalente



# Encore une requête équivalente



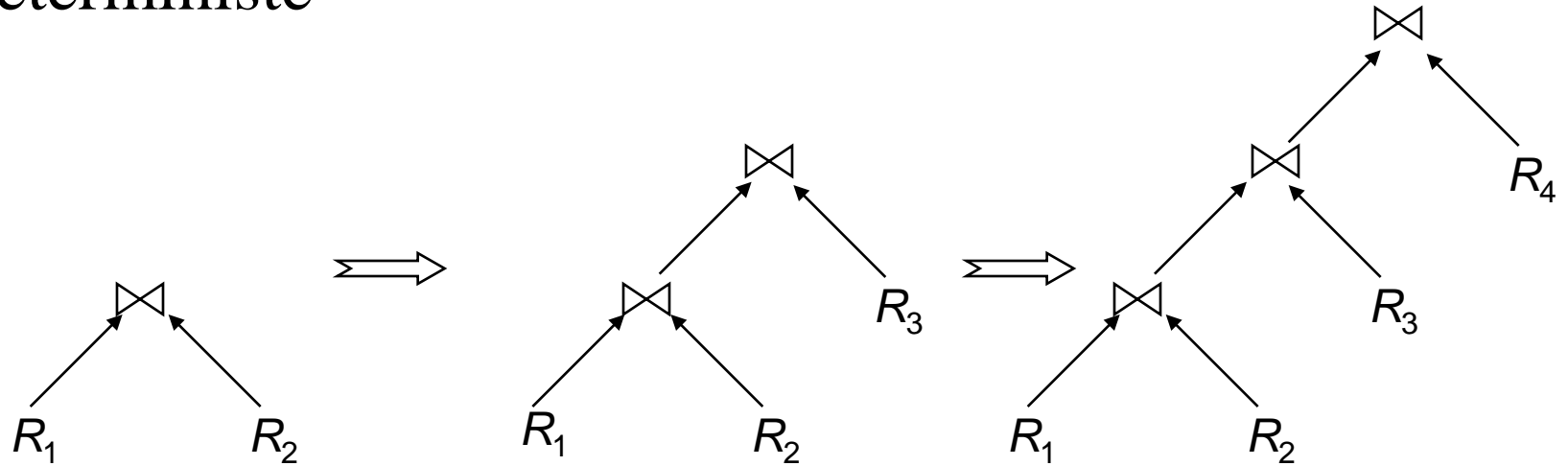


# Stratégie de recherche

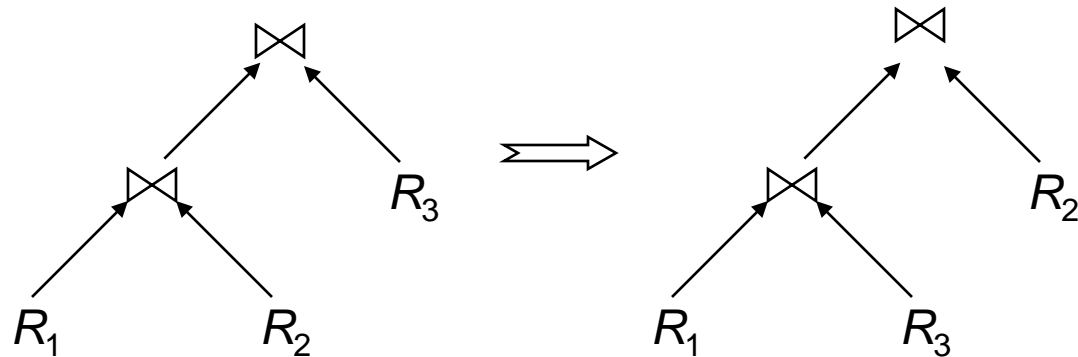
- Déterministe
  - part des relations de base et construit les plans en ajoutant une relation à chaque étape
  - programmation dynamique: largeur-d'abord
  - excellent jusqu'à 5-6 relations
- Aléatoire
  - recherche l'optimalité autour d'un point de départ particulier
  - réduit le temps d'optimisation (au profit du temps d'exécution)
  - meilleur avec  $> 5-6$  relations
    - recuit simulé (simulated annealing)
    - amélioration itérative (iterative improvement)

# Stratégies de recherche

- Déterministe



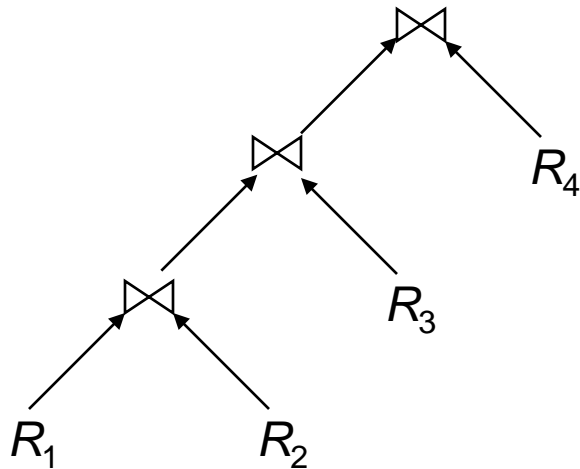
- Aléatoire



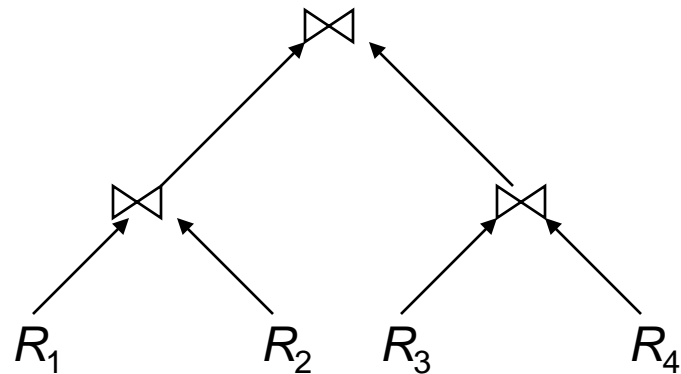
# Algorithmes de recherche

- Limiter l'espace de recherche
  - heuristiques
    - par ex. appliquer les opérations unaires avant les autres
    - Ne marche pas toujours (perte d'index, d'ordre)
  - limiter la forme des arbres

Arbre linéaire



Arbre touffu



# Génération de plan physique

- Sélection :
  - Commencer par les conditions d'égalité avec un index sur l'attribut
  - Filtrer sur cet ensemble de n-uplets ceux qui correspondent aux autres conditions
- Jointure
  - Utilisation des index, des relations déjà triées sur l'attribut de jointure, présence de plusieurs jointures sur le même attribut
- Pipelines ou matérialisation

# Conclusion

- Point fondamental dans les SGBD
- Importance des métadonnées, des statistiques sur les relations et les index, du choix des structures d'accès.
- L'administrateur de bases de données peut améliorer les performances en créant de nouveaux index, en réglant certains paramètres de l'optimiseur de requêtes (voir TME et cours ABDR en M2)