

Master d'Informatique

# **BDR - 4I803 – COURS 6**

Interrogation de bases de données réparties

# Interrogation de Bases de Données réparties

- Transparence des requêtes
- Evaluation des requêtes
- Fragmentation de requêtes
- Optimisation de requêtes

# Transparence de la répartition

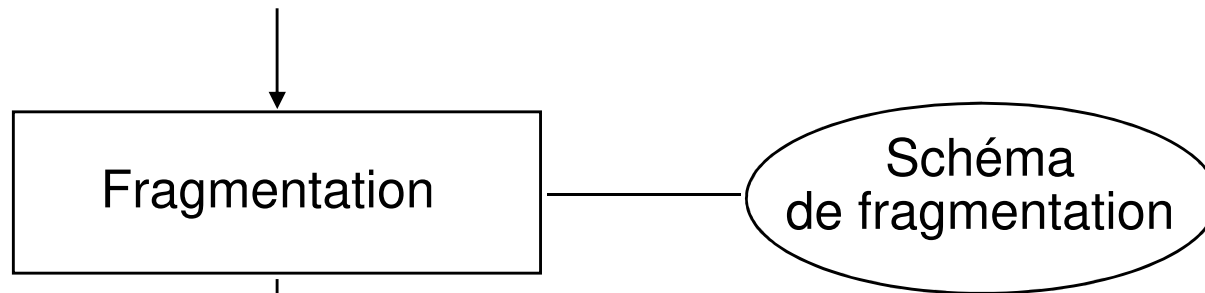
- Transparence de la répartition : degré d'intégration du schéma
  - haut niveau de transparence : pas d'information sur la fragmentation
  - bas niveau de transparence (ex. RDA) : l'utilisateur doit spécifier les fragments qu'il manipule => pb de nommage non ambigu.
- Chaque item de la BD doit avoir un nom unique
  - 1. serveur de noms : attributs, relations, utilisateurs, ...
    - goulot d'étranglement, pb en cas de panne, peu d'autonomie locale
    - Peu de màj -> répliquer
  - 2. Préfixer par le site + serveur de nom local

# Transparence des requêtes

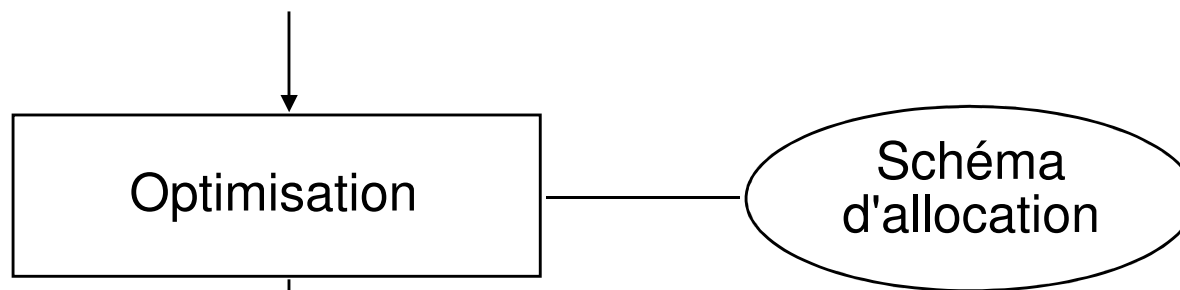
- Le système doit optimiser les lectures/écritures, effectuer automatiquement les mises à jour des répliques, optimiser les requêtes.
- Où se trouve le schéma de répartition (table des fragments, des répliques)?
  1. Chaque site a une copie de tout le schéma :
    - + : lectures rapides
    - : modifications de schémas (pb général de la répli.)
  2. Chaque site a un schéma local
    - : il faut retrouver les informations sur les autres sites (ex. inondation)
  3. Solutions mixtes : seuls certains sites ont une copie du schéma, certains sites conservent les informations concernant un item, etc. Le schéma est une (meta)base répartie

# Evaluation de Requêtes Réparties

Requête sur relations globales



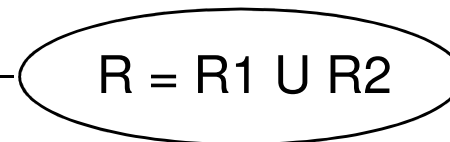
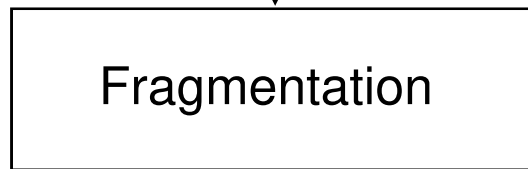
Requête sur fragments



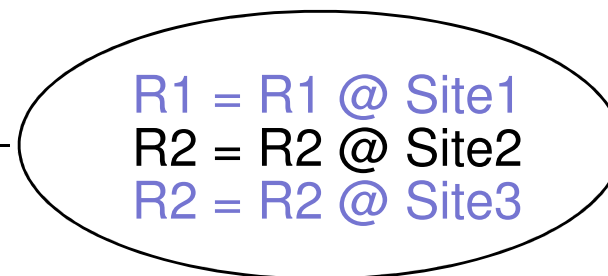
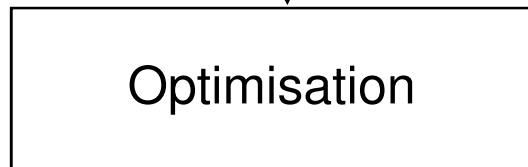
Plan d'exécution réparti

# Exemple d'évaluation simple

Select A from R where B = b



Select A from R1 where B = b  
union Select A from R2 where B = b



Select A from R1 @ Site1 where B = b  
union Select A from R2 @ Site3 where B = b

# Fragmentation

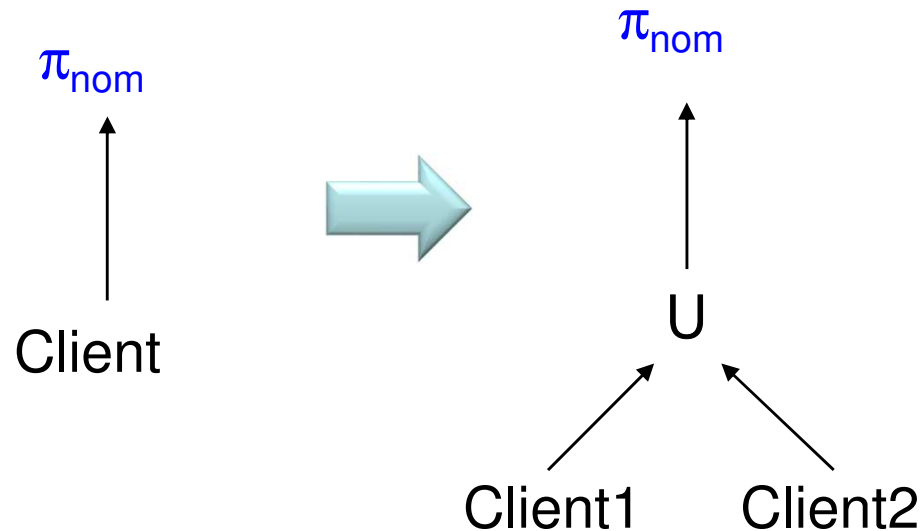
- Réécriture
  - Traduire la requête SQL en un arbre algébrique
    - feuille = relation
    - noeud = opérateur relationnel
- Reconstruction
  - Substituer chaque relations globale par sa définition en fonction des fragments
- Transformation
  - Appliquer des techniques de réduction pour éliminer les opérations inutiles

# Reconstruction

**Requête : Select distinct nom**

**From Client**

$$\left. \begin{array}{l} \text{Client}_1 = \sigma_{\text{ville} = \text{'Paris'}} \text{Client} \\ \text{Client}_2 = \sigma_{\text{ville} \neq \text{'Paris'}} \text{Client} \end{array} \right\} \text{Client} = \text{Client}_1 \cup \text{Client}_2$$



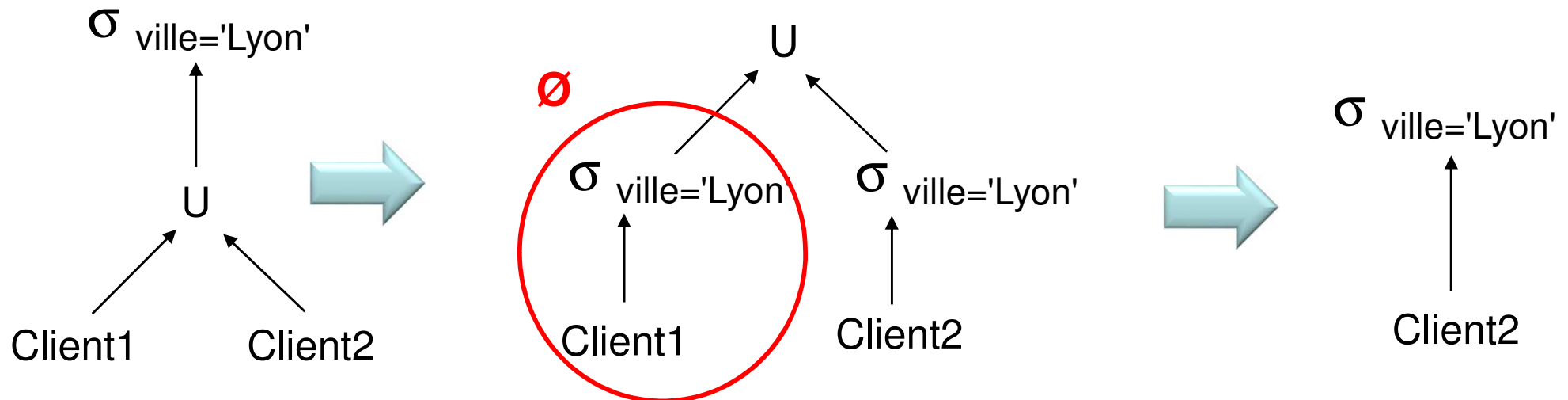


# Réduction pour la fragmentation horizontale

Règle : éliminer l'accès aux fragments inutiles

Exemple : **Requête:**  $\sigma_{ville = 'Lyon'} Client$

$$\left. \begin{array}{l} Client_1 = \sigma_{ville = 'Paris'} Client \\ Client_2 = \sigma_{ville \neq 'Paris'} Client \end{array} \right\} Client = Client_1 \cup Client_2$$

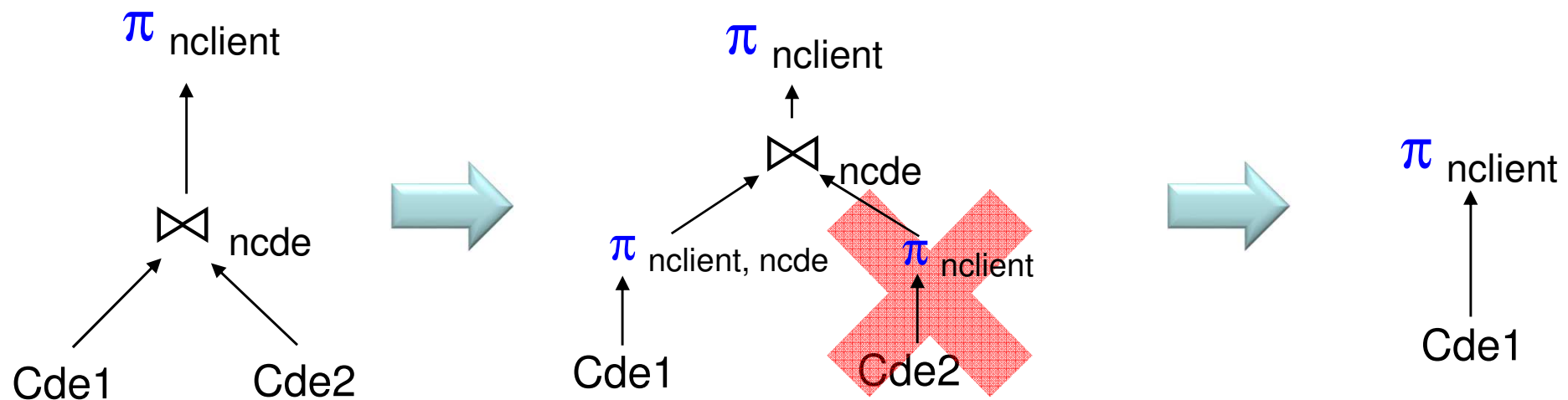


# Réduction pour la Fragmentation Verticale

Règle : éliminer les accès aux relations de base qui n'ont pas d'attributs utiles pour le résultat final

Exemple de requête: **Select distinct nclient from Cde**

$$\left. \begin{array}{l} \text{Cde1} = \pi_{\text{ncde}, \text{nclient}}(\text{Cde}) \\ \text{Cde2} = \pi_{\text{ncde}, \text{produit}, \text{qté}}(\text{Cde}) \end{array} \right\} \text{Cde} = \text{Cde}_1 \bowtie \text{Cde}_2$$



# Réduction pour la Fragmentation Horizontale Dérivée

Règle : distribuer les jointures par rapport aux unions et appliquer les réductions pour la fragmentation horizontale

Exemple

$Client_1 = \sigma_{ville = 'Paris'} Client$

$Client_2 = \sigma_{ville \neq 'Paris'} Client$

$Cde1 = Cde \bowtie Client_1$

$Cde2 = Cde \bowtie Client_2$

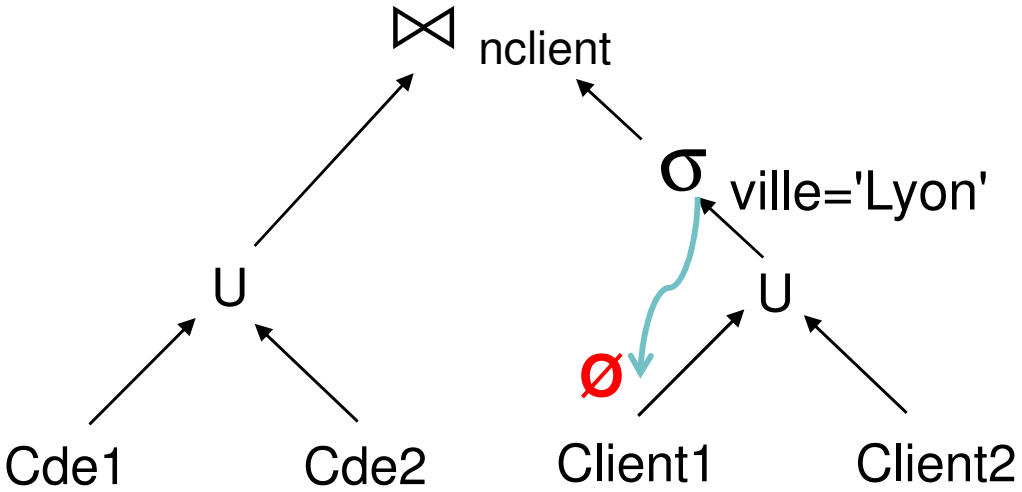
**Select \* from Client, Cde**

**where Client.nclient = Cde.nclient and ville='Lyon'**

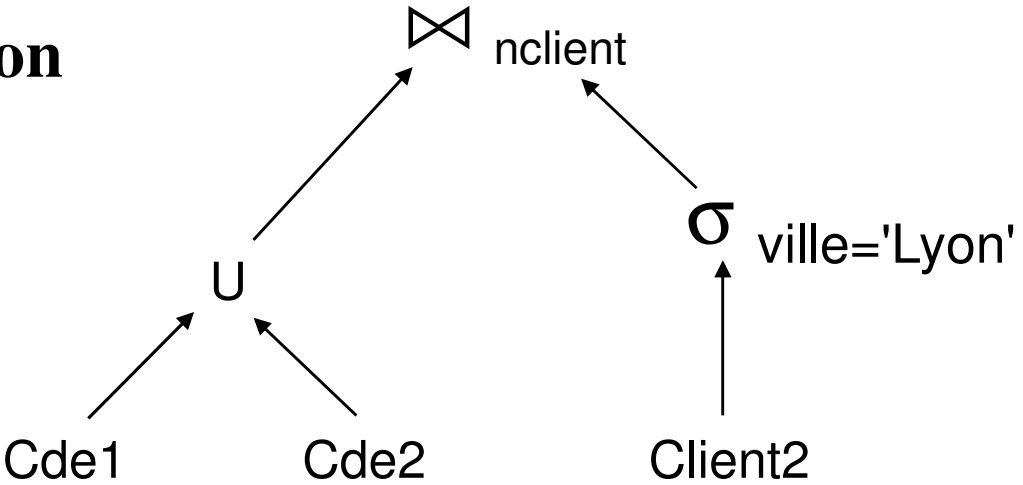
Select \* from Client, Cde  
where Client.nclient = Cde.nclient and  
ville=Lyon

# Exemple

## Requête canonique



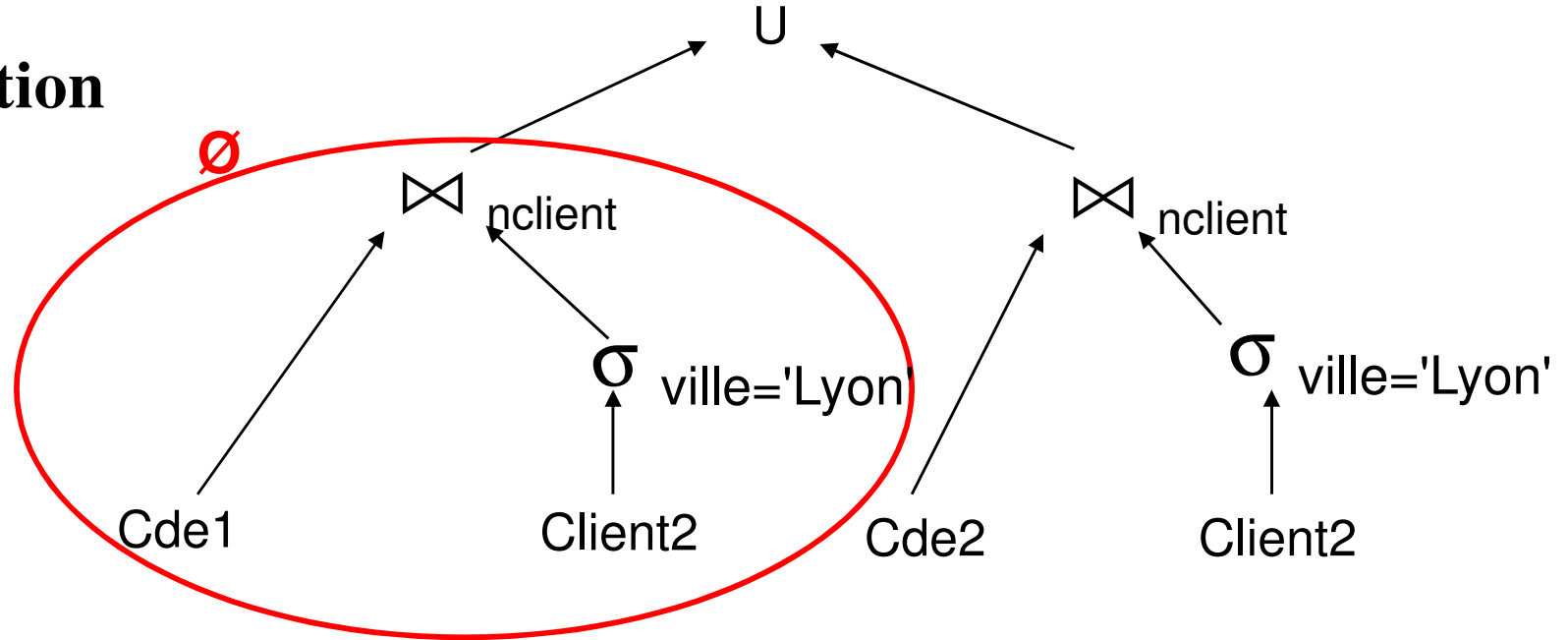
## Après 1ère réduction



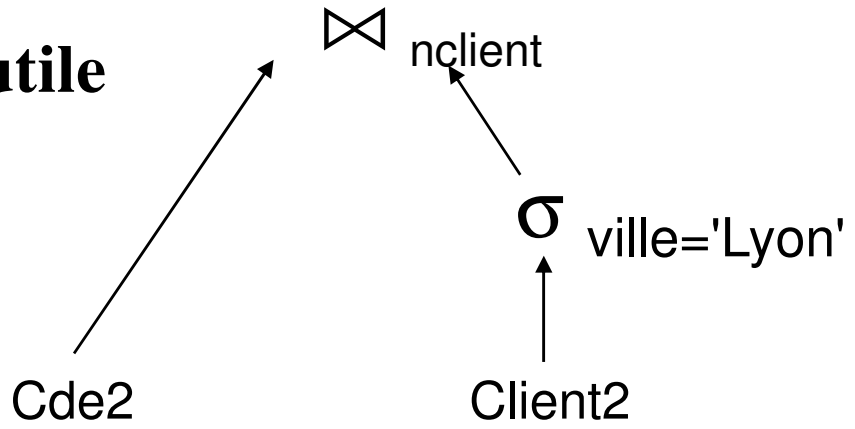
Select \* from Client, Cde  
where Client.nclient = Cde.nclient and  
ville=Lyon

# Exemple (suite)

## Distribution



## Elimination du sous-arbre inutile



# Optimisation de Requêtes Réparties

entrée : une requête simplifiée exprimée sur des fragments

sortie : un plan d'exécution réparti optimal

- Objectifs

- choisir la meilleure localisation des fragments
- minimiser une fonction de coût:  $f$  (I/O, CPU, Comm.)
- exploiter le parallélisme (temps réponse vs. utilisation ressource)
- exprimer les transferts inter-sites pour les minimiser

- Solution

- examiner le coût de chaque plan possible (par transformation) et prendre le meilleur

# Exemple

**Site 1 :**  $\text{Client}_1 = \sigma_{\text{ville} = \text{'Paris'}} \text{Client}$

**Site 2 :**  $\text{Client}_2 = \sigma_{\text{ville} \neq \text{'Paris'}} \text{Client}$

**Site 3 :**  $\text{Cde1} = \text{Cde} \bowtie \text{Client}_1$

**Site 4 :**  $\text{Cde2} = \text{Cde} \bowtie \text{Client}_2$

**Site 5 :** résultat

**Select \***

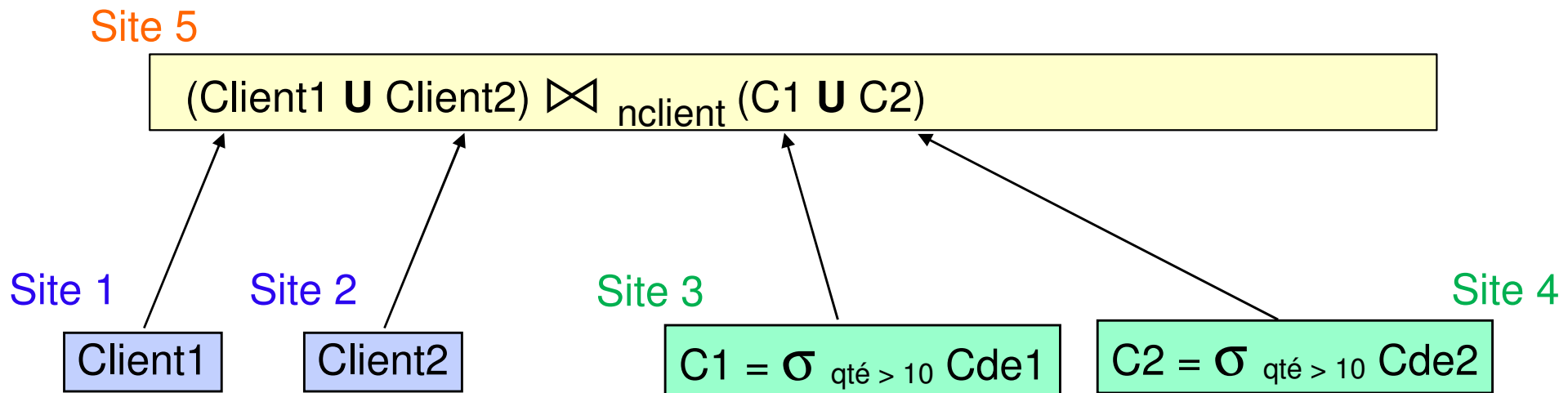
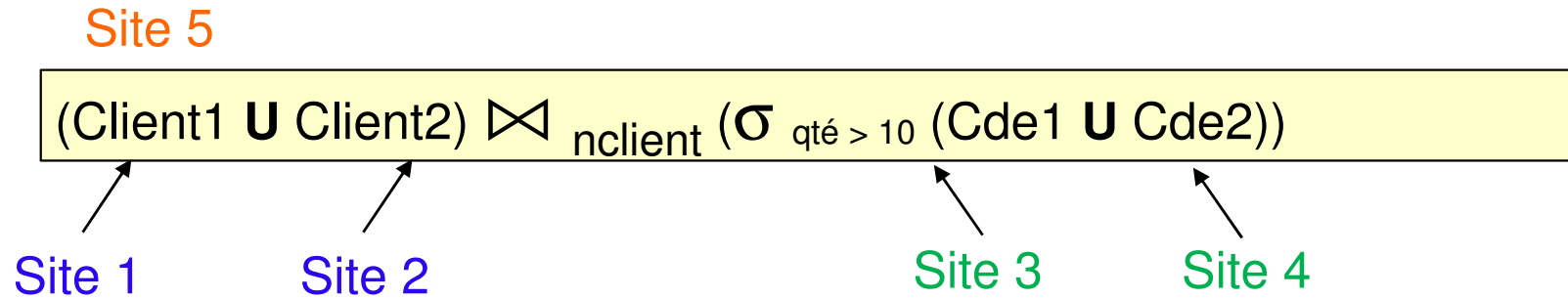
**from Client, Cde**

**where Client.nclient = Cde.nclient**

**and qté > 10**

Select \* from Client, Cde  
where Client.nclient = Cde.nclient  
and qté > 10

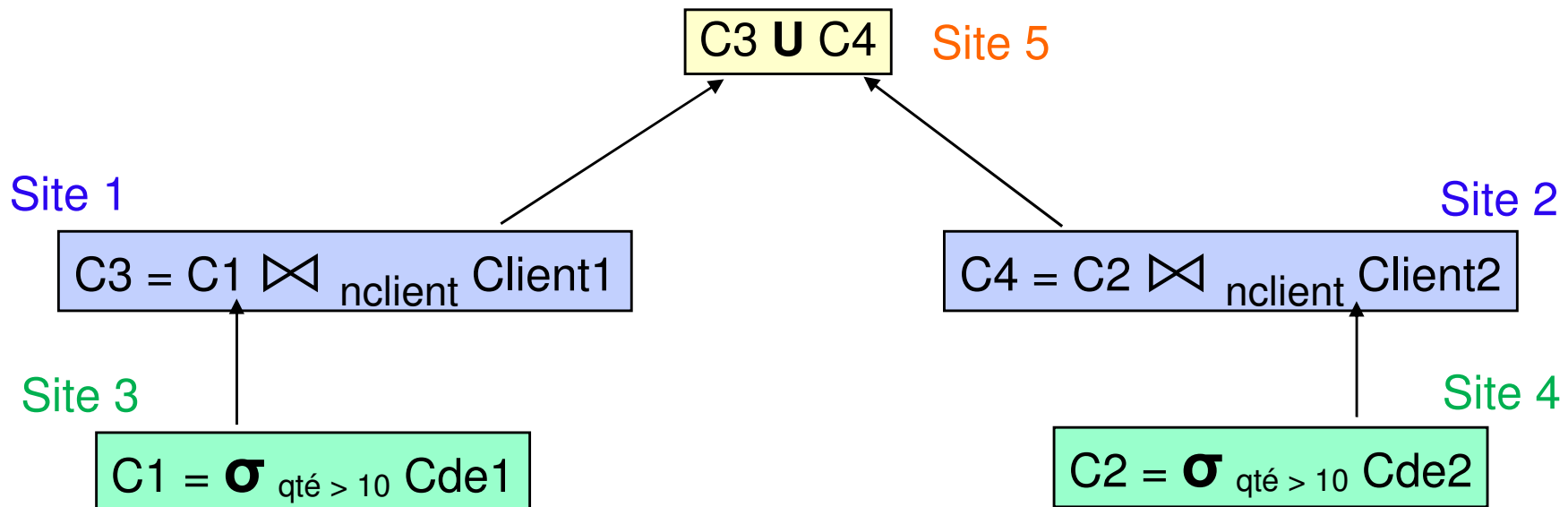
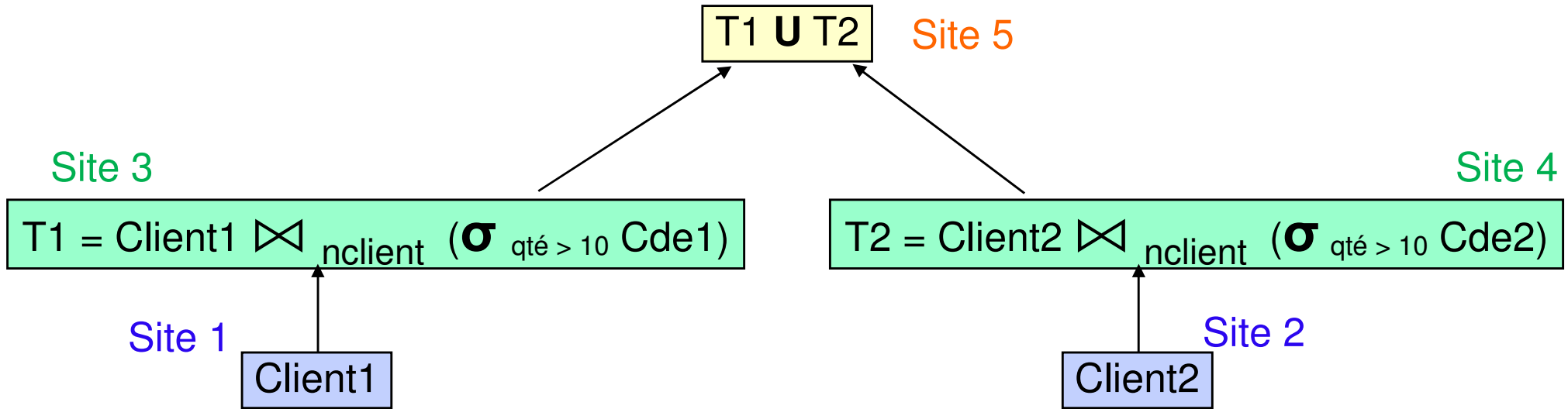
## Solution: plans 1 et 2





Select \* from Client, Cde  
where Client.nclient = Cde.nclient  
and qté > 10

## Solutions: plans 3 et 4



# Coût des Solutions

## Supposons

- taille (Cde1) = taille (Cde2) = 10 000
- taille (Client1) = taille (Client2) = 2 000
- coût de transfert de 1n-uplet = 1
- Sélectivité( $qté > 10$ ) = 1%

## Stratégie 1

- transfert de Cde1 + Cde2 = 20 000
- transfert de Client1 + Client2 = 4000

## Stratégie 2

- transfert de Client1 + Client2 = 4000
- transfert de C1 + C2 = 200

## Stratégie 3

- transfert de Client1 + Client2 = 4000
- Transfert de T1 + T2 = 200

## Stratégie 4

- transfert de C1 + C2 = 200
- transfert de C3 + C4 = 200

# Jointure

R sur S1, S sur S2, T sur S3.

Requête demandée sur le site S0 :  $R \bowtie S \bowtie T$

Plusieurs possibilités :

a) Copier tout sur S0 et faire les jointures sur S0

b) Copier R sur S2, et joindre R et S sur S2

Copier le résultat sur S3, et faire la jointure avec T sur S3

Copier le résultat sur S0

Tenir compte des index, de la taille des relations à transférer, de la taille des relations intermédiaires, de la charge des sites, de la vitesse de transmission, etc.

On peut paralléliser les jointures : grand nombre de stratégies

# Semi-jointure

Traiter la jointure entre deux relations réparties sur deux sites :

R1 sur  $S_1$  et R2 sur  $S_2$

Rappel :  $R1 \bowtie R2 = \pi_{\text{Att de R1}} (R1 \bowtie R2)$

**Requête  $R1 \bowtie_A R2 = R1 \bowtie_A (R2 \bowtie_A R1)$**  et le résultat doit être sur  $S_1$

Sur  $S_1$ :  $T1 = \pi_A (R1)$ , puis envoi de T1 sur  $S_2$

Sur  $S_2$ :  $T2 = R2 \bowtie_A T1$ , puis envoi de T2 sur  $S_1$

Sur  $S_1$ : Calcul de  $R1 \bowtie_A T2$

**Requête  $R1 \bowtie R2 = (R1 \bowtie R2) \bowtie (R2 \bowtie R1)$**  et résultat sur  $S_0$

Sur  $S_1$ : Transférer  $T1 = \pi_A (R1)$  vers  $S_2$

Sur  $S_2$ : Transférer  $T2 = \pi_A (R2)$  vers  $S_1$

Sur  $S_1$ : Transférer  $T3 = R1 \bowtie R2$  vers  $S_0$

Sur  $S_2$ : Transférer  $T4 = R2 \bowtie R1$  vers  $S_0$

Sur  $S_0$  :  $T3 \bowtie T4$

ne transférer des nuplets  
complets que s'ils joignent

# Algorithme d'optimisation R\* (System R)

- Choix de la meilleure stratégie dans la liste exhaustive des stratégies possibles.
- Le site maître (où la requête est posée) prend toutes les décisions globales :
  - sélection des sites où exécuter les requêtes
  - choix des fragments
  - choix des méthodes de transferts
- Les autres sites intervenant dans la requête prennent les décisions locales :
  - ordre des jointures locales
  - génération des plans d'accès locaux
- Fonction de coût totale incluant le coût des traitements locaux, et les coûts de transmission.

# Algorithme R\*

Input : arbre de requête, localisation des relations, statistiques

Output : stratégie d'exécution la moins chère

Pour chacune des relations

- déterminer le coût de chacun des accès possibles
  - index, parcours séquentiel, etc.
- choisir l'accès le moins coûteux

Calculer le coût de chacune des combinaisons possibles

- Choisir celle de moindre coût

Pour chaque site intervenant dans la requête

- calculer le plan d'exécution local le moins coûteux.

# Optimisation

L'optimiseur choisit (à l'aide de statistiques et de fonctions de coût)

- l'ordre des jointures
- l'algorithme de jointure
- le chemin d'accès
- les sites des résultats intermédiaires
- la méthode de transfert de données
  - Tout envoyer (bien pour les petites relations)
  - Envoyer uniquement les n-uplets utiles : on envoie la valeur de jointure sur le site de la relation interne, et on renvoie uniquement les n-uplets correspondants. (bien si bonne sélectivité)

# Stratégies de jointure

4 stratégies possibles pour effectuer  $R \bowtie S$  sur l'attribut A, avec l'algorithme des boucles imbriquées (R relation externe, S relation interne).

On note  $s = \frac{\text{Card}(S \bowtie_A R)}{\text{Card}(R)}$  le nombre moyen de n-uplets de S

joignant un n-uplet de R.

On note LC le coût du traitement local, et CC le coût de communication.

## 1. Envoyer R (relation externe) sur le site de S (relation interne)

Les n-uplets de R sont joints au fur et à mesure de leur arrivée sur le site de S

$$\begin{aligned} \text{Coût total} = & \text{LC}(\text{retrouver } \text{card}(R) \text{ n-uplets de R}) \\ & + \text{CC}(\text{size}(R)) \\ & + \text{LC}(\text{retrouver } s \text{ n-uplets de S}) * \text{card}(R) \end{aligned}$$



## Stratégies de jointure (2)

### 2. Envoyer S (relation interne non triée) sur le site de R (relation externe).

(les n-uplets internes ne peuvent pas être joints au fur et à mesure de leur arrivée sur le site, et sont stockés dans la relation temporaire T).

$$\begin{aligned} \text{Coût total} = & \text{LC (retrouver } card(S) \text{ n-uplets de S)} \\ & + \text{CC (size(S))} \\ & + \text{LC (stocker } card(S) \text{ n-uplets dans T)} \\ & + \text{LC (retrouver } card(R) \text{ n-uplets de R)} \\ & + \text{LC (retrouver } s \text{ n-uplets de T) * } card(R) \end{aligned}$$

## Stratégies de jointure (3)

### 3. Chercher les n-uplets de S (relation interne) nécessaires pour chaque n-uplet de R (relation externe).

La valeur de l'attribut de jointure de chaque n-uplet de R est envoyée sur S. Les  $s$  n-uplets de S correspondant à cette valeur sont envoyés sur le site de R et joints à la volée.

$$\begin{aligned} \text{Coût total} = & \text{LC (retrouver } \textit{card} (R) \text{ n-uplets de R)} \\ & + \text{CC} (\textit{length}(A) ) * \textit{card}(R ) \\ & + \text{LC (retrouver } s \text{ n-uplets de S) } * \textit{card} (R ) \\ & + \text{CC}(s * \textit{length}(S)) * \textit{card} (R ) \end{aligned}$$

Rmq: On suppose que l'attribut A est **unique** dans R

## Stratégie de jointure (4)

### 4. Transférer les deux relations sur un troisième site et calculer la jointure sur ce site.

S (relation interne) est transférée en premier, et stockée dans T. Les n-uplets de R sont joints au fur et à mesure de leur arrivée.

$$\begin{aligned} \text{Coût total} = & \text{LC (retrouver } \mathit{card}(S) \text{ n-uplets de } S) \\ & + \text{CC (} \mathit{size}(S) \text{)} \\ & + \text{LC (stocker } \mathit{card}(S) \text{ n-uplets dans } T) \\ & + \text{LC (retrouver } \mathit{card}(R) \text{ n-uplets de } R) \\ & + \text{CC (} \mathit{size}(R) \text{)} \\ & + \text{LC (retrouver } s \text{ n-uplets de } T) * \mathit{card}(R) \end{aligned}$$

# Conclusion

- Importance de la fragmentation horizontale pour augmenter le parallélisme inter- et intra-requête
- Utiliser la fragmentation verticale si forte affinité d'attributs et beaucoup d'attributs
- L'optimisation reposant sur un modèle de coût est critique s'il y a beaucoup de sites