

BDR 4I803 Cours 4

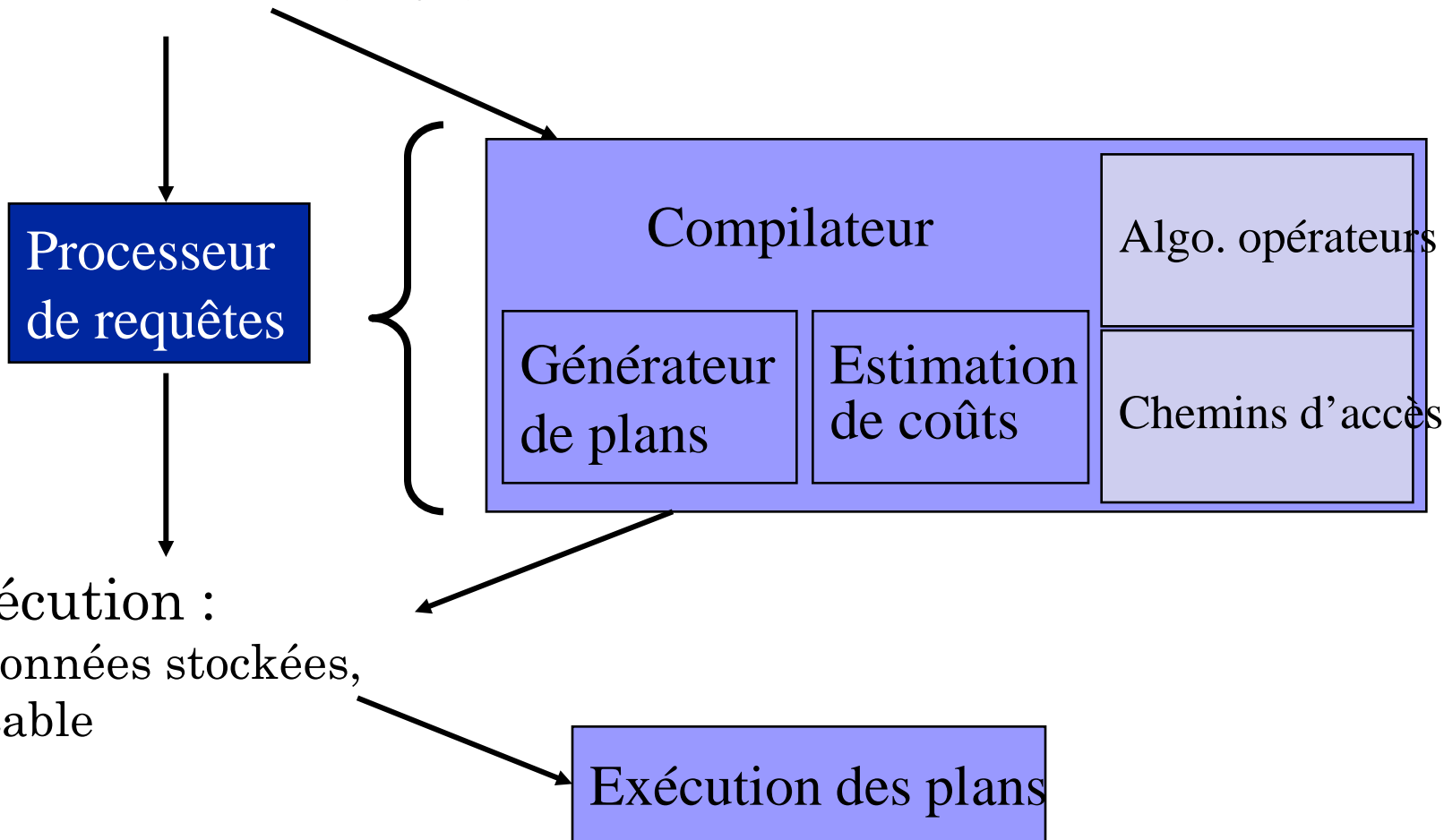
Opérateurs relationnels Implémentation et coût

Plan

- Rappel et Objectif
- Notion de pipeline
- Tri
- Sélection
- Projection
- Jointure
- Autres

Traitement des requêtes

Requête déclarative (SQL)



Objectif

- Comprendre les algorithmes qui évaluent les opérateurs relationnels
- Quantifier les accès aux données nécessaire pour évaluer une opération
 - Unité de mesure : la page
 - Les opérations principales sont :
 - sélection, projection, jointure, tri, ...
- Disposer d'un modèle (i.e., des formules) pour déterminer le coût d'une opération en termes d'accès aux données
- Hypothèses : coût **E/S** >> coût **CPU**
 - Lire un nuplet à partir d'une **page de données stockée** sur disque dure beaucoup plus longtemps que de **calculer** un nuplet à partir de **données déjà en mémoire**.

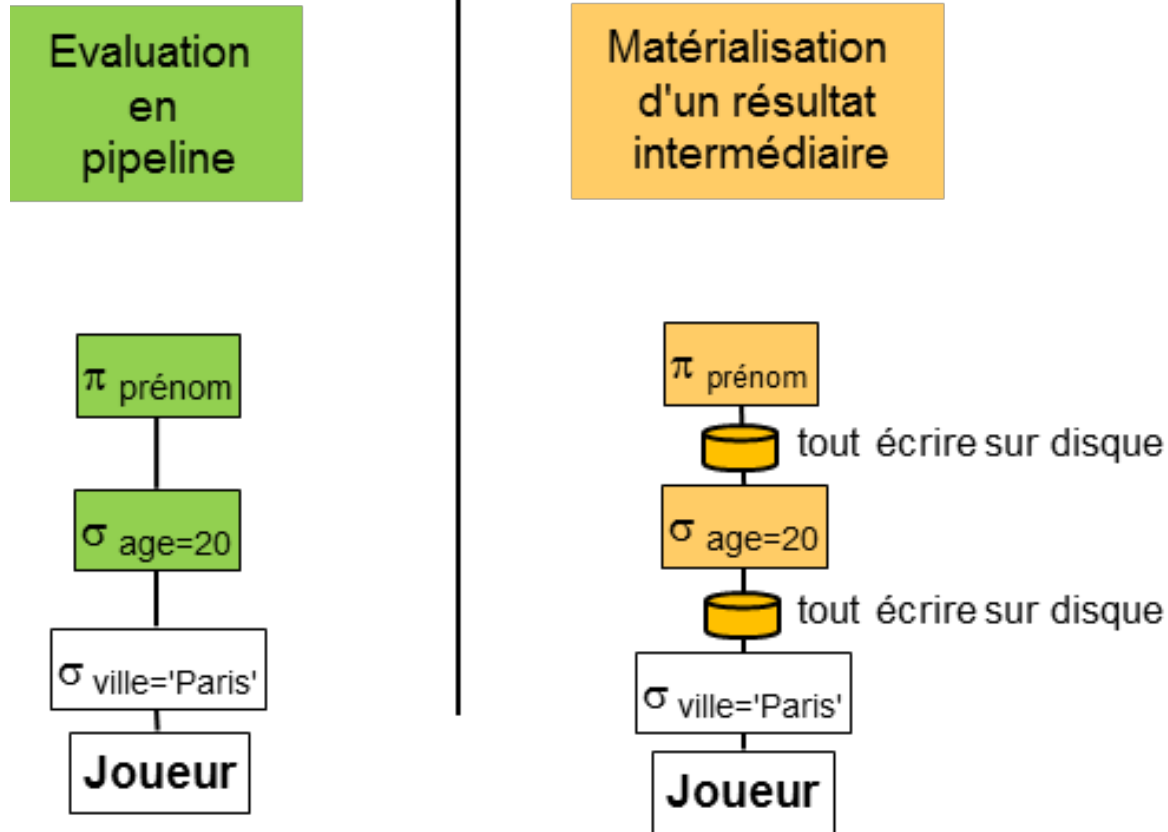
Implémentation des opérateurs

- Il existe plusieurs algorithmes *physiques* possibles pour un opérateur *logique*
 - comprendre les différentes variantes
- Détailler les **étapes** de l'algorithme physique
- Faire la distinction entre
 - Etape impliquant un accès aux données
 - Etape sans accès aux données

Evaluation en pipeline d'une opération

- Une opération est évaluée en **pipeline**
 - Si elle est évaluée **sans** lire aucune donnée stockée dans la base
 - Chaque opérande (données en entrée) doit être le résultat d'une autre opération
 - Opérande \neq table
 - Opérande **non** matérialisée : jamais stockée temporairement sur disque avant de d'évaluer l'opération
 - On « consomme » les opérandes pour « produire » la sortie progressivement
- **Pipeline = traitement à la volée**
- Avantage : le **coût** d'une opération en pipeline est **négligeable**

Pipeline vs. matérialisation



- Rmq: pas de pipeline pour la première sélection car accès aux données stockées.

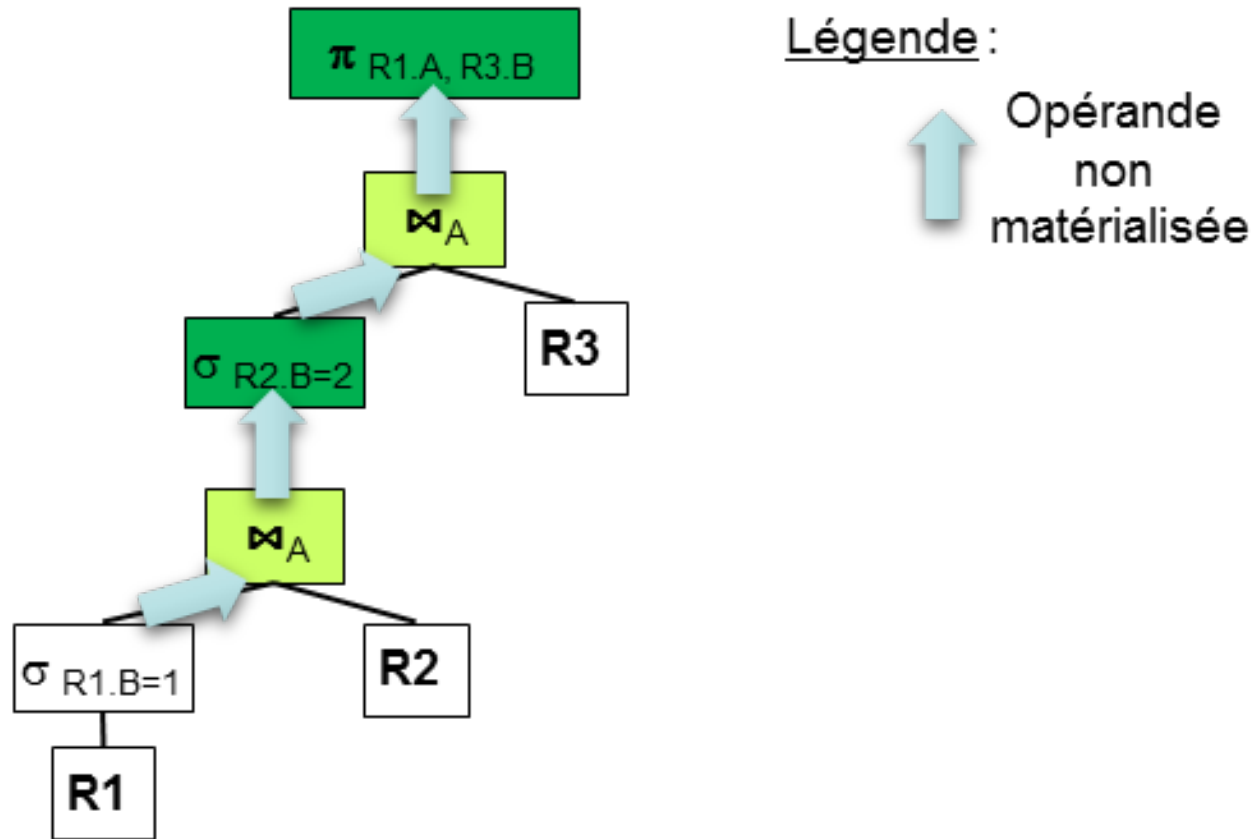
Opération unaire évaluée en pipeline

- Une opération unaire a une opérande e
 - Exples : $\sigma_{prédicat}(e)$ $\pi_{attributs}(e)$
- Si l'opérande e est une **expression composée** d'au moins une opération, c-à-d, si $e \neq \text{table}$ alors
 - $\text{coût}(\sigma_{prédicat}(e)) \simeq \text{coût}(e)$
 - $\text{coût}(\pi_{attributs}(e)) \simeq \text{coût}(e)$

Opération binaire évaluée en pipeline

- Opération binaire en pipeline
 - Union avec doublons
 - opération non relationnelle: UNION ALL en SQL
 - Fusion de listes déjà triées
 - Jointure entre une "petite" et une "grande" relation
- Coût d'une opération *n-aire* en pipeline
 - = somme du coût de ses opérandes

Opération binaire évaluée en pipeline



- "Flux" de nuplets "remontant" sur la branche principale.
- Evaluation en pipeline possible des jointures si R2 et R3 ont préalablement été "chargés" en mémoire.

Evaluation itérative en pipeline

- Une opération en pipeline est **itérative**
 - Parcours itératif des nuplets de l'opérande pour calculer, un par un, les nuplets du résultat.
 - Le 1^{er} nuplet du résultat dépend seulement des m premiers éléments de l'opérande
 - Le 2^{ème} nuplet du résultat dépend seulement des n éléments suivants de l'opérande
 - ... ainsi de suite jusqu'au dernier nuplet du résultat
- Interface itérative
 - Commune à tous les opérateurs :
 - méthodes open, nextTuple(), close
 - nextTuple() invoque récursivement nextTuple() des opérandes
- Avantage:
 - Chaque opérateur produit son résultat à la demande de son père
 - **Contrôle** du flux des nuplets intermédiaires

Algorithmes des opérateurs relationnels

- Diapos suivantes
 - Parcours séquentiel
 - Tri
 - Sélection
 - Projection
 - Jointure
 - ...

Parcours séquentiel d'une table

- Requête:
 - Select * from R
- R stockée dans $page(R)$ pages du disque
 - Taille d'une page en octets : T_{page}
 - Nombre de nuplets de R dans une page :
 - $T_{page} / largeur(R)$
 - $page(R) = card(R) / (T_{page} / largeur(R))$
- Parcours séquentiel : *Table Access Full*
 - $Coût(R) = page(R) \cdot c$
 - avec $c < 1$ si les pages à lire sont contigües
 - sinon $c = 1$

Tri externe

- Hypothèse : k pages tiennent en mémoire
- Algorithme en s étapes : tri de blocs puis fusions de blocs
- Etape n°1 : tri
 - Lire R pour créer des paquets triés de k pages chacun
 - Nombre de paquets obtenus : $\text{page}(R)/k$
 - Coût de l'étape de tri (lecture + matérialisation) : $2.\text{page}(R)$
- Puis étapes n° 2, 3, ..., s : fusion
 - Fusion
 - Charger la première page de k paquets et les fusionner
 - Dès qu'une page est vide, charger la suivante du même paquet
 - Dès que les k premiers paquets ont été fusionnés, fusionner les k paquets suivants
 - On obtient des paquets triés de taille k^2 pages
 - Coût d'une étape : lire et matérialiser toutes les données : $2.\text{page}(R)$
 - Nombre de paquets : $\text{page}(R)/k^2$
 - Continuer jusqu'à obtenir **un seul** paquet, on a donc :
 - $\text{page}(R) / k^s \leq 1$

Tri externe (suite)

- Nombre d'étape s tq : $k^s \geq \text{page}(R)$:
$$s = \lceil \log_k(\text{page}(R)) \rceil$$
- Coût total des s étapes :
 - $\text{Coût}(\text{tri}(R)) = 2 \cdot \text{page}(R) \cdot s$
- Si on ne matérialise **pas** le résultat de la dernière étape :
 - $\text{Coût}(\text{tri}(R)) = \text{page}(R) \cdot (2s - 1)$
- Si R est une expression composée ($R \neq \text{table}$)
Tri fait en pipeline, pas besoin de la première lecture de R

Sélection : σ

- Sélection : $\sigma_{p(A)}(R)$
 - avec $p(A)$ est un prédicat dépendant de l'attribut A
- Si R est une expression composée
 - $\text{Coût}(\sigma_{p(A)}(R)) = \text{coût}(R)$
- Si R est une table et l'attribut A n'est pas indexé
 - $\text{Coût}(\sigma_{p(A)}(R)) = \text{page}(R)$

Sélection par index non plaçant: : σ_{NP}

- Index **non** plaçant sur A, accès en 2 étapes
 - Etape 1: **traverser l'index**
 - **Index Scan** : atteindre une feuille de l'index pour évaluer une égalité
 - $C_{index} = 1$ pour une table de hachage linéaire
 - $C_{index} = 2$ pour une table de hachage extensible
 - Ou **Index Range Scan** : atteindre des feuilles consécutives de l'index pour évaluer une inégalité
 - $C_{index} = \text{hauteur de l'arbre} + (\text{nombre de feuilles concernées} - 1)$
 - Lire les **rowid** des nuplets (sauf si unique, stocké dans la feuille)
 - $C_{rowid} = \lceil \text{card}(\sigma_{p(A)}(R)) / \text{nombre de rowid par page} \rceil$
 - Etape 2: **Table Access By Rowid**
 - lire **une page par nuplet** du résultat
 - Coût($\sigma_{NP_{p(A)}}(R)$) = $C_{index} + C_{rowid} + \text{card}(\sigma_{p(A)}(R)) \cdot 1$

Sélection par index plaçant : σ_P

- Index **plaçant** sur A, accès en 2 étapes
 - Etape 1: **traverser l'index**
 - obtenir l'adresse de la première page indexée
 - $C_{index} = 0$ pour une table de hachage linéaire
 - $C_{index} = 1$ pour une table de hachage extensible
 - $C_{index} = \text{hauteur de l'arbre} - 1$
 - Etape 2 : lire des pages "pleines" de nuplets du résultat
 - lire **une fraction** de la table

$$\text{Coût}(\sigma_{P(A)}(R)) = C_{index} + \lceil \text{page}(R) * \text{card}(\sigma_{P(A)}(R)) / \text{card}(R) \rceil$$

- Rappel pour calculer $\text{card}(\sigma_{P(A)}(R))$
 - la distribution des attributs est uniforme. Les attributs sont tous indépendants les uns des autres.

Sélections complexes

- Sélections complexes sur une table contenant plusieurs prédicats
- Lorsque plusieurs attributs sont indexés séparément
 - Conjonction (AND)
 - Intersection d'adresses de nuplets (rowid)
 - Vecteur binaire puis ET logique
 - Disjonction (OR)
 - Union d'adresses de nuplets
 - Vecteur binaire puis OU logique


Projection : π


- Projection sans doublons : $\pi_{\text{Attributs}}(R)$
 - R est une relation
 - Correspond au "**Select distinct** Attributs"
 - si $\pi_{\text{Attr}}(R)$ tient en mémoire ou si R est sans doublons
 - $\text{Coût}(\pi_{\text{Attr}}(R)) = \text{coût}(R)$
 - si $\pi_{\text{Attr}}(R)$ ne tient **pas** en mémoire. Deux possibilités :
 - En triant
 - Lire R pour matérialiser R trié selon *Attributs*, puis lire R trié
 - OU en hachant
 - Lire R et la hacher sur disque, insérer uniquement si nouvelle valeur pour « Attributs » puis lire chaque paquet
- Projection SQL avec doublons : **Select Attributs**
 - Opération **non** relationnelle
 - $\text{Coût}(\text{proj}_{\text{Attr}}(R)) = \text{coût}(R)$

Jointures:

- Diapos suivantes :
- Boucles imbriquées
 - simple
 - par bloc
 - avec index
- Par hachage
- Par Tri fusion
- ...

Boucles imbriquées simples :

- On suppose que S est une table
- Jointure notée $R \text{ _{R.a=S.a} S$
 - Lire R pour itérer sur les nuplets résultant de R
 - La lecture se fait page par page
 - Pour chaque page le R, lire S pour obtenir les nuplets de S qui satisfont le prédicat de jointure

```
foreach page Pr de R do
  foreach nuplet s ∈ S do
    qqsoit r ∈ Pr, if r.a=s.a then ajoute <r,s> dans T
```
- $\text{Coût}(R \text{ _{R.a=S.a} S) = \text{coût}(R) + \text{page}(R) \cdot \text{page}(S)$

Boucles avec matérialisation : \bowtie_{Mat}

- Sert lorsque S est une sous-expression
 - i.e., S n'est pas une table
- Jointure notée $R \bowtie_{\text{Mat } R.a=S.a} S$
 - Commencer par **évaluer S** et **stocker** le résultat dans S_{Mat}
 - Lire R pour itérer sur les nuplets résultant de R
 - Pour chaque r dans R, lire S_{Mat} pour obtenir les nuplets de S qui satisfont le prédicat de jointure

$$\text{Coût}(R \bowtie_{\text{Mat } R.a=S.a} S) = \text{coût}(S) + \text{page}(S) + \text{coût}(R) + \text{page}(R) \cdot \text{page}(S)$$

Jointure par blocs : \bowtie_{Bloc} et \bowtie_{MatBloc}

- Hypothèse
 - On suppose que $M+2$ pages de R tiennent en mémoire
- Possibilité de **réduire** le nombre d'accès à S
 - Itération principale par **blocs** de M pages de R
 - Puis joindre chaque nuplet de S avec le **bloc** courant
- Si S est une table
 - $\text{Coût}(R \bowtie_{\text{Bloc}}_{A.a=B.a} S) = \text{coût}(R) + \text{page}(R)/M \cdot \text{page}(S)$
- Si S est une sous-expression
 - alors on **matérialise S** avant d'effectuer la jointure :
 - $\text{Coût}(R \bowtie_{\text{MatBloc}}_{A.a=B.a} S) = \text{coût}(S) + \text{page}(S) + \text{coût}(R) + \text{page}(R)/M \cdot \text{page}(S)$

Jointure par boucles avec index: \bowtie_{Ind}

- Jointure par boucle imbriquée et index sur l'attribut a de S
 - Évite de parcourir S entièrement pour chaque page de R

```
foreach tuple r ∈ R do
  accès aux tuples s ∈ S par index
  foreach tuple s do
    ajouter <r,s> dans le résultat
```

- $\text{Coût}(R \bowtie_{\text{Ind } R.a=S.a} S) = \text{coût}(R) + \text{card}(R) \cdot \text{coût}(\sigma_{a=v}(S))$

On suppose que l'index tient en mémoire

cf. diapo sur la sélection, on fixe $C_{\text{index}} = 0$ et $C_{\text{rowid}} = 0$

Le terme $\text{coût}(\sigma_{a=v}(S))$ dépend du type d'index :

- Index **non plaçant** sur $S.a$: $\text{coût}(\sigma_{a=v}(S)) = \text{card}(\sigma_{a=v}(S))$
- Index **plaçant** sur $S.a$: $\text{coût}(\sigma_{a=v}(S)) = \lceil \text{page}(S) * \text{card}(\sigma_{a=v}(S)) / \text{card}(S) \rceil$
- cas typique de la jointure naturelle
 - si l'attribut a est une clé de S , alors $\text{coût}(\sigma_{a=v}(S)) = 1$

Jointure par tri puis fusion : \bowtie_{TF}

- Tri-fusion
 - trier R et S sur l'attribut de jointure : $\text{Sort}(\text{join})$
 - voir tri externe
 - Fusionner les relations triées : Merge join
 - Amélioration: commencer la fusion de R avec S , avant la fin complète du tri
 - on peut fusionner directement les 2 relations dès que le nombre de paquets restant dans les 2 relations est inférieur à k
 - Trier R en P_R paquets, et trier S en P_S paquets, tq:
 $P_R + P_S < k$
 - Fusion des P_R paquets de R avec les P_S paquets de S en une seule étape
 - Exemple: jointure entre 2 tables R et S
 - $\text{Page}(R)=6000$, $\text{page}(S)=3000$ $k=100$ pages en mémoire
 - Tri: 2. ($\text{page}(R) + \text{page}(S)$),
 - On obtient 60 blocs de R et 30 blocs de S soit un total de 90 blocs
 - Il suffit de lire les blocs pour les fusionner : $\text{page}(R) + \text{page}(S)$
 - $\text{Coût}(R \bowtie_{TF_{R.a=S.a}} S) = 3 (\text{page}(R) + \text{page}(S))$

Jointure par hachage : \bowtie_H

- Hachage en mémoire
 - Lire S et construire une hashmap temporaire en mémoire:
 - Associer l'attribut de jointure avec les nuplets correspondants
 - $\text{Map}\langle A, \text{List}\langle \text{Nuplet} \rangle \rangle$
 - Lire R pour itérer sur les nuplets de R :
 - Pour chaque r dans R , obtenir les nuplets de S associés avec la valeur $R.a$
 - Pour chaque nuplet associé : produire un nuplet du résultat
 - $\text{Coût}(R \bowtie_H R.a=S.a S) = \text{coût}(S) + \text{coût}(R)$
- Hachage externe
 - Hacher R et S avec la **même** fonction de hachage : $h(a)$
 - Les données doivent être hachées suffisamment "finement" de telle sorte que :
 - un paquet de R avec un paquet de S tiennent en mémoire
 - Lecture : $\text{coût}(R) + \text{coût}(S)$
 - Ecriture: $\text{page}(R) + \text{page}(S)$
 - pour chaque paquet i de R et i de S , associer les nuplets r et s tq $r.a = s.a$
 - $\text{page}(R) + \text{page}(S)$
 - $\text{Coût}(R \bowtie_{H\text{Ext}} R.a=S.a S) = \text{coût}(R) + \text{coût}(S) + 2 \cdot (\text{page}(R) + \text{page}(S))$

Order by

- Voir tri externe

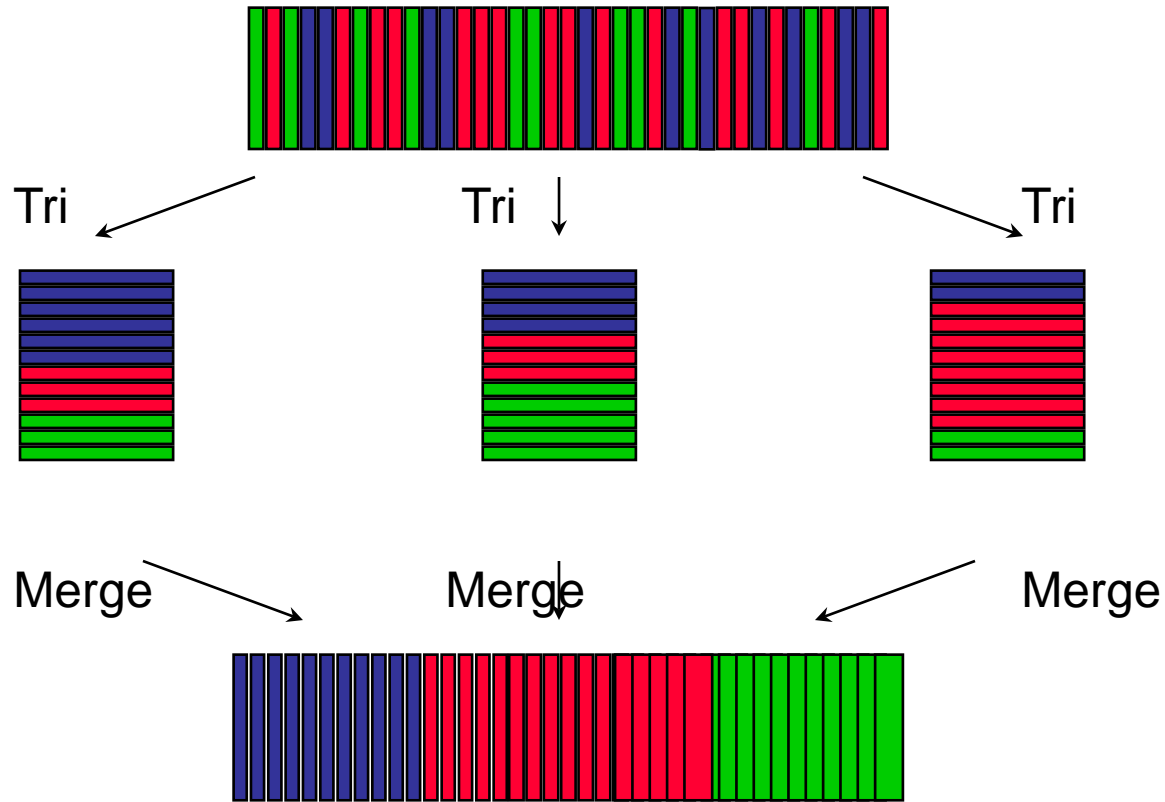
Autres opérations

- Group By avec agrégation
 - Hachage ou tri
- a IN (*sous-requête*)
 - Evaluer la sous-requête pour chaque valeur de a
 - Algorithme général de type "boucles imbriquées"
 - Lorsque la sous-requête ne dépend **pas** de la requête principale
 - Evaluer une (semi) jointure : Requête principale \bowtie sous-requête
 - Charger la sous requête en mémoire : voir algo de type \bowtie_H
 - Ou
 - Matérialiser la sous-requête : voir algo de type \bowtie_{Mat}

Perspectives

- Nombreuses autres variantes pour implémenter les opérateurs relationnels
- Evaluation en parallèle d'un opérateur
- Tri externe en parallèle
 - Sort benchmark : sortbenchmark.org
 - 2009: 500 GO/mn, 2013: 1,4 TO/mn, 2014: 4,3 TO/mn
 - 2015: **16** TO/mn <http://sortbenchmark.org/FuxiSort2015.pdf>

Tri externe : Fusion en 1 seule étape



Fusion en plusieurs étapes

