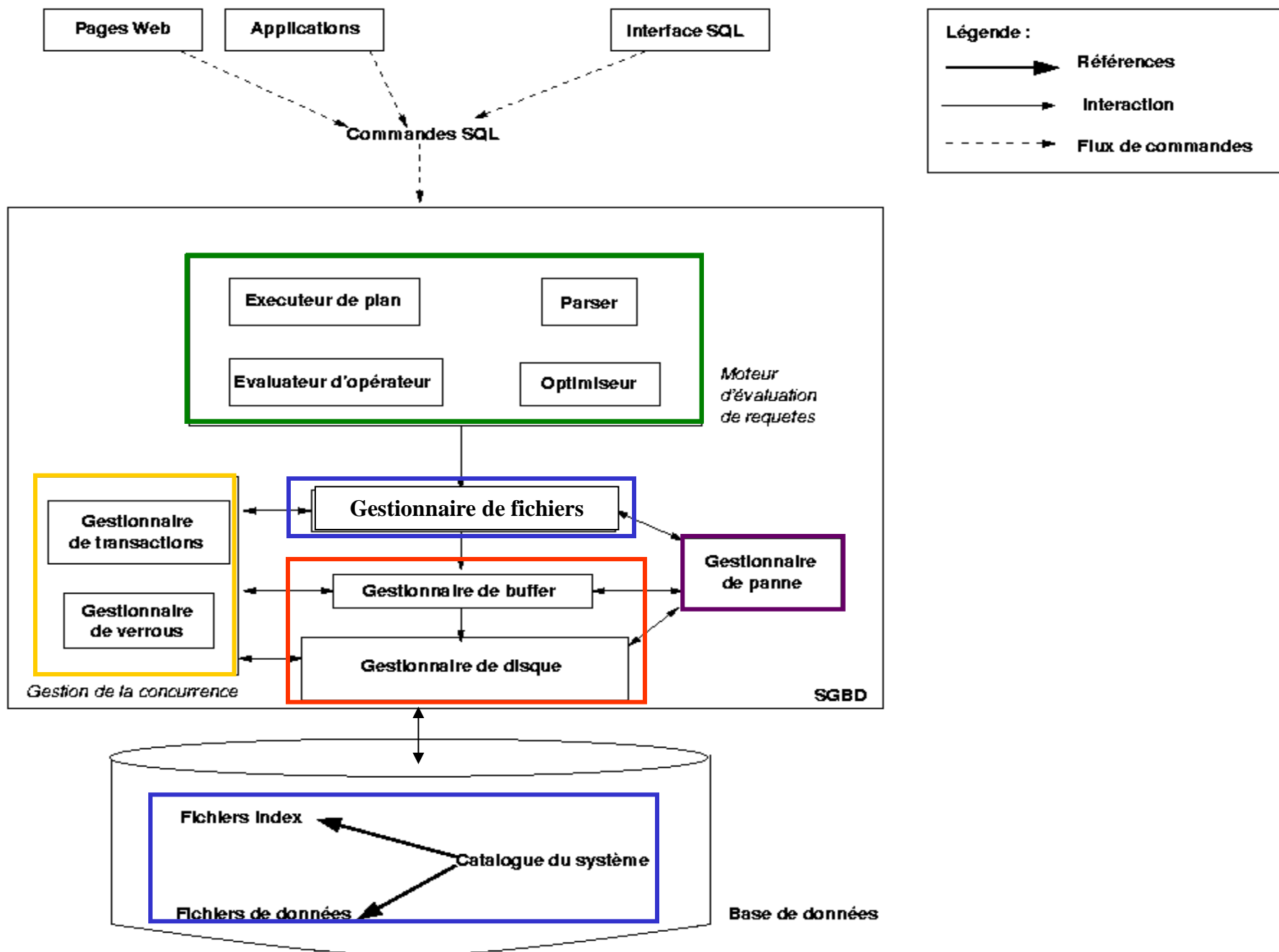


Architecture d'un SGBD
Optimisation des requêtes
Marta Rukoz

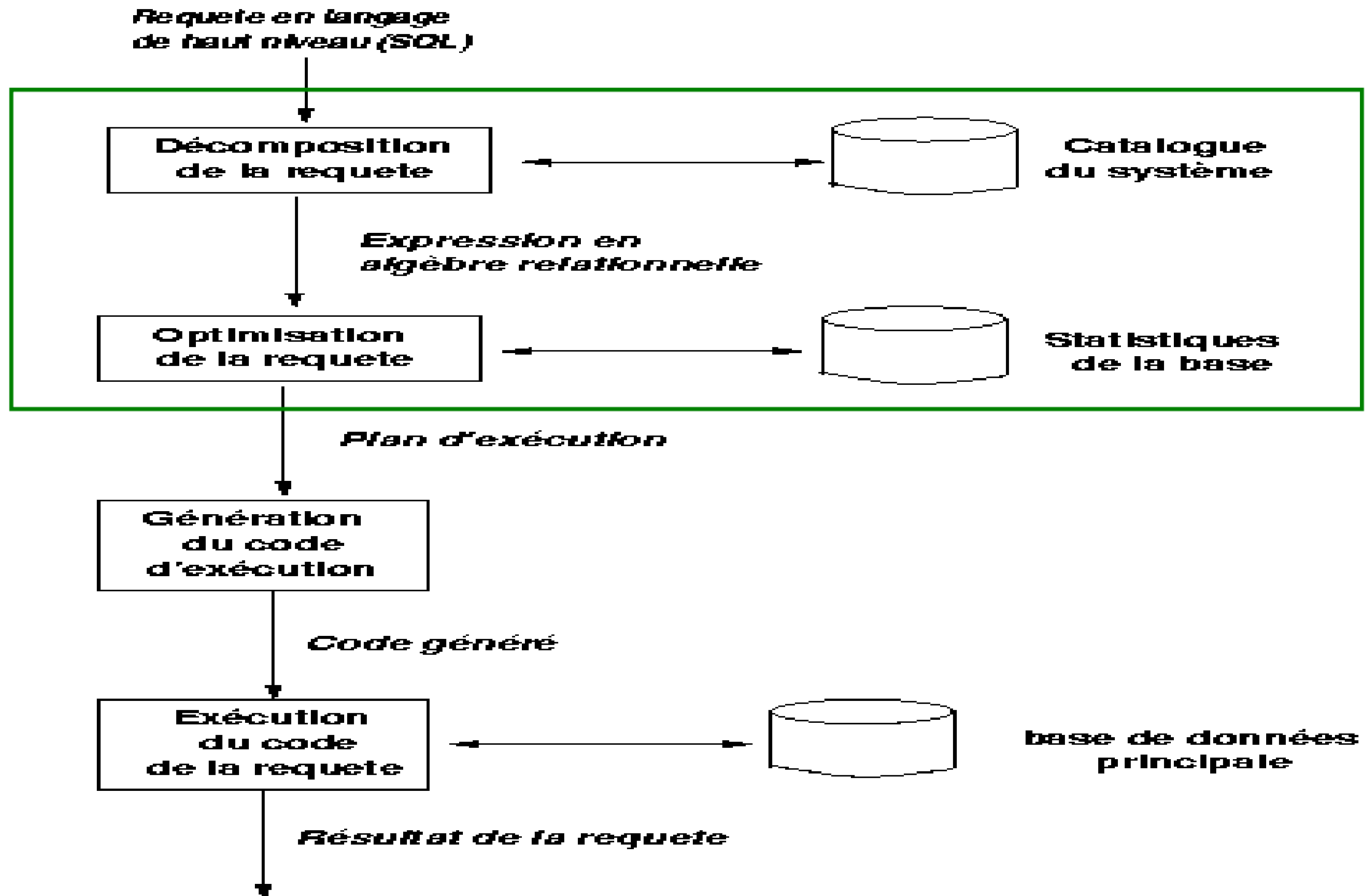
- Décomposition d'une requête
- Optimisation de la requête
- Évaluation de la requête



Optimisation de requêtes

- **Exécution de requête** : séries d'opérations permettant d'extraire des données de la base
- **Optimisation de requête** : activité permettant de choisir la meilleure stratégie d'exécution d'une requête

Phases d'exécution d'une requête



Exemple (1/5)

"Quels sont les noms des commerciaux basés dans les filiales de Londres ? »

```
SELECT e.Nom  
FROM Employe e, Filiale f  
WHERE e.#Filiale=f.#Filiale  
AND e.Position = 'Commercial'  
AND f.Ville='Londres'
```

Supposons :

- *Chaque page ne contient qu'un nuplet*
- *Employe contient 1000 nuplets, Filiale en contient 50*
- *Il y a 50 commerciaux et 5 filiales à Londres*

- **Trois requêtes possibles en algèbre relationnelle**
- **Calcul du coût de chaque requête en terme E/S**

Exemple (2/5)

$\sigma_{(\text{position}='Commercial') \wedge (\text{Ville}='Londres') \wedge (\text{Employe.Num_filiale}=\text{Filiale.Num_filiale})}(\text{Employe} \times \text{Filiale})$

$\sigma_{(\text{position}='Commercial') \wedge (\text{Ville}='Londres')}$
 $(\text{Employe} \bowtie_{\text{Employe.Num_filiale}=\text{Filiale.Num_filiale}} \text{Filiale})$

$\sigma_{(\text{position}='Commercial')} \text{Employe} \bowtie_{\text{Employe.Num_filiale}=\text{Filiale.Num_filiale}}$
 $\sigma_{(\text{Ville}='Londres')} \text{Filiale})$

Exemple (3/5)

Employe 1000 nuplets, Filiale 50 Il y a 50 commerciaux et 5 à Londres

$\sigma_{(position='Commercial') \wedge (Ville='Londres') \wedge (Employe.Num_filiale=Filiale.Num_filiale)}(Employe \times Filiale)$

- Le produit cartésien impose la lecture des deux relations : **1000 + 50 E/S**
- La relation résultat du produit cartésien contient : **1000 * 50** nuplets.
Tous ces nuplets doivent être lus pour vérifier les prédicats de la sélection :
1000 * 50 E/S

$$(1000 + 50) + 2 * (1000 * 50) = 101050 E/S$$

Exemple (4/5)

Employe 1000 nuplets, Filiale 50 Il y a 50 commerciaux et 5 à Londres

$\sigma_{(\text{position}=\text{'Commercial'}) \wedge (\text{Ville}=\text{'Londres'})}$

$(\text{Employe} \bowtie \text{Employe.Num_filiale}=\text{Filiale.Num_filiale}) \text{ Filiale}$

- La jointure entre les deux relations impose une lecture des deux relations : **1000 + 50 E/S**

- Le résultat de la jointure est une relation contenant : **1000** nuplets. Ces nuplets doivent être lus pour vérifier les prédicats de la sélection : **1000 E/S**

$$(1000 + 50) + (2 * 1000) = 3050 \text{ E/S}$$

Exemple (5/5)

Employe 1000 nuplets, Filiale 50 Il y a 50 commerciaux et 5 à Londres

$\sigma_{(\text{position}=\text{'Commercial'})}$ **Employe** \bowtie **Employe**.Num_filiale=Filiale.Num_filiale)

$\sigma_{(\text{Ville}=\text{'Londres'})}$ **Filiale)**

- Première sélection \Rightarrow la lecture de *Employe* et la création d'une relation de 50 nuplets : **1000 + 50 E/S**
- La deuxième sélection \Rightarrow la lecture de Filiale et la création d'une relation de 5 nuplets : 50 + 5 E/S
- La jointure \Rightarrow la lecture des deux relations : (50 + 5) E/S

$$1000 + 50 + 2 * (50 + 5) = 1160 \text{ E/S}$$

Phase 1 : Décomposition

Transformation de la requête SQL en une requête en algèbre relationnelle

- Vérification syntaxique et sémantique de la requête
- Utilisation du **catalogue** du système
- Représentation de la requête par un **arbre d'opérateurs algébriques**

Arbre algébrique (1/2)

Représentation des :

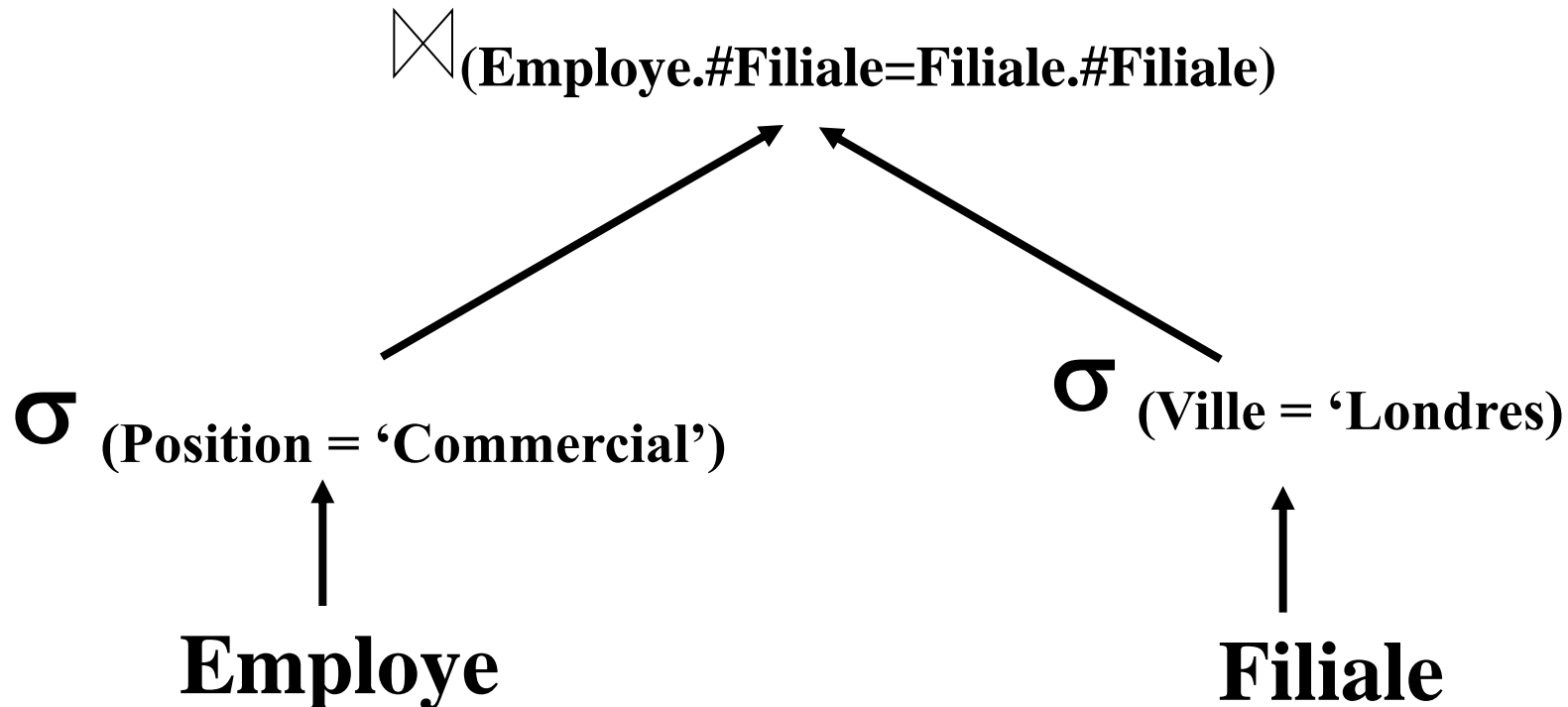
- relations impliquées dans la requête par les nœuds feuille de l'arbre
- résultats intermédiaires par des nœuds non feuille
- résultat de la requête par le nœud racine de l'arbre

*Ordre des séquences d'opérations : des
feuilles vers la racine*

Arbre algébrique (2/2)

$\sigma_{(\text{position}=\text{'Commercial'})}$ **Employe** \bowtie **Employe**.Num_filiale=Filiale.Num_filiale)

$\sigma_{(\text{Ville}=\text{'Londres'})}$ **Filiale**)



Phase 2 : Optimisation

- Équivalences d'expressions
- Transformation d'un arbre algébrique
- Évaluation de requête
 - Statistiques de la base de données
 - Coût des opérations

Équivalences d'expressions (1/6)

1) Cascade de sélections : $\sigma_{p \wedge q \wedge r} (R) = \sigma_p(\sigma_q(\sigma_r(R)))$

2) Commutativité des sélections : $\sigma_p(\sigma_q(R)) = \sigma_q(\sigma_p(R))$

3) Séquence de projections : $\Pi_L(\Pi_M(\dots \Pi_N(R))) = \Pi_L(\Pi(R))$

4) Commutativité des sélections et des projections :

$$\Pi_{A_1 \dots A_n}(\sigma_p(R)) = \sigma_p(\Pi_{A_1 \dots A_n}(R))$$

Équivalences d'expressions (2/6)

5) Commutativité des jointures : $\mathbf{R} \bowtie_p \mathbf{S} = \mathbf{S} \bowtie_p \mathbf{R}$ et

$$\mathbf{R} \times \mathbf{S} = \mathbf{S} \times \mathbf{R}$$

6) Commutativité des jointures et des sélections

Si p s'applique à la relation \mathbf{R} alors

$$\sigma_p(\mathbf{R} \bowtie_r \mathbf{S}) = (\sigma_p(\mathbf{R})) \bowtie_r \mathbf{S} \quad \text{et}$$

$$\sigma_p(\mathbf{R} \times \mathbf{S}) = (\sigma_p(\mathbf{R})) \times \mathbf{S}$$

Équivalence d'expressions (4/6)

7) Commutativité des jointures et des projections

Si la projection se fait sur l'ensemble d'attributs $L=L_1 \cup L_2$ avec L_1 s'appliquant à R et L_2 s'appliquant à S alors

$$\Pi_{L_1 \cup L_2} (R \bowtie_p S) = \Pi_{L_1} (R) \bowtie_p \Pi_{L_2} (S)$$

8) Commutativité des unions et des intersections

$$(R \cup S) = (S \cup R) \text{ et}$$

$$(R \cap S) = (S \cap R)$$

Équivalence d'expressions (3/6)

9) Commutativité des unions, intersections, différences et des sélections

$$\sigma_p(\mathbf{R} \cup \mathbf{S}) = \sigma_p(\mathbf{R}) \cup \sigma_p(\mathbf{S})$$

$$\sigma_p(\mathbf{R} \cap \mathbf{S}) = \sigma_p(\mathbf{R}) \cap \sigma_p(\mathbf{S})$$

$$\sigma_p(\mathbf{R} - \mathbf{S}) = \sigma_p(\mathbf{R}) - \sigma_p(\mathbf{S})$$

Équivalence d'expressions (5/6)

10) Commutativité des projections et des unions

$$\Pi_L(R \cup S) = \Pi_L(R) \cup \Pi_L(S)$$

11) Associativité des jointures

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

12) Associativité des unions et des intersections

$$(R \cup S) \cup T = R \cup (S \cup T)$$

$$(R \cap S) \cap T = R \cap (S \cap T)$$

Transformation d'un arbre algébrique

- ① Division des conjonctions de sélections
- ② Ré-ordonnancement des sélections en utilisant les règles 2^* et 4^+
- ③ Application des sélections les plus sélectives en premier
- ④ Transformation des produits cartésiens en jointure
- ⑤ Ré-ordonnancement des équi-jointures en utilisant la règle 11
- ⑥ Déplacement des projections et création de nouvelles projections en utilisant les règles 4^+ et 7^{++}

* Commutativité des sélections

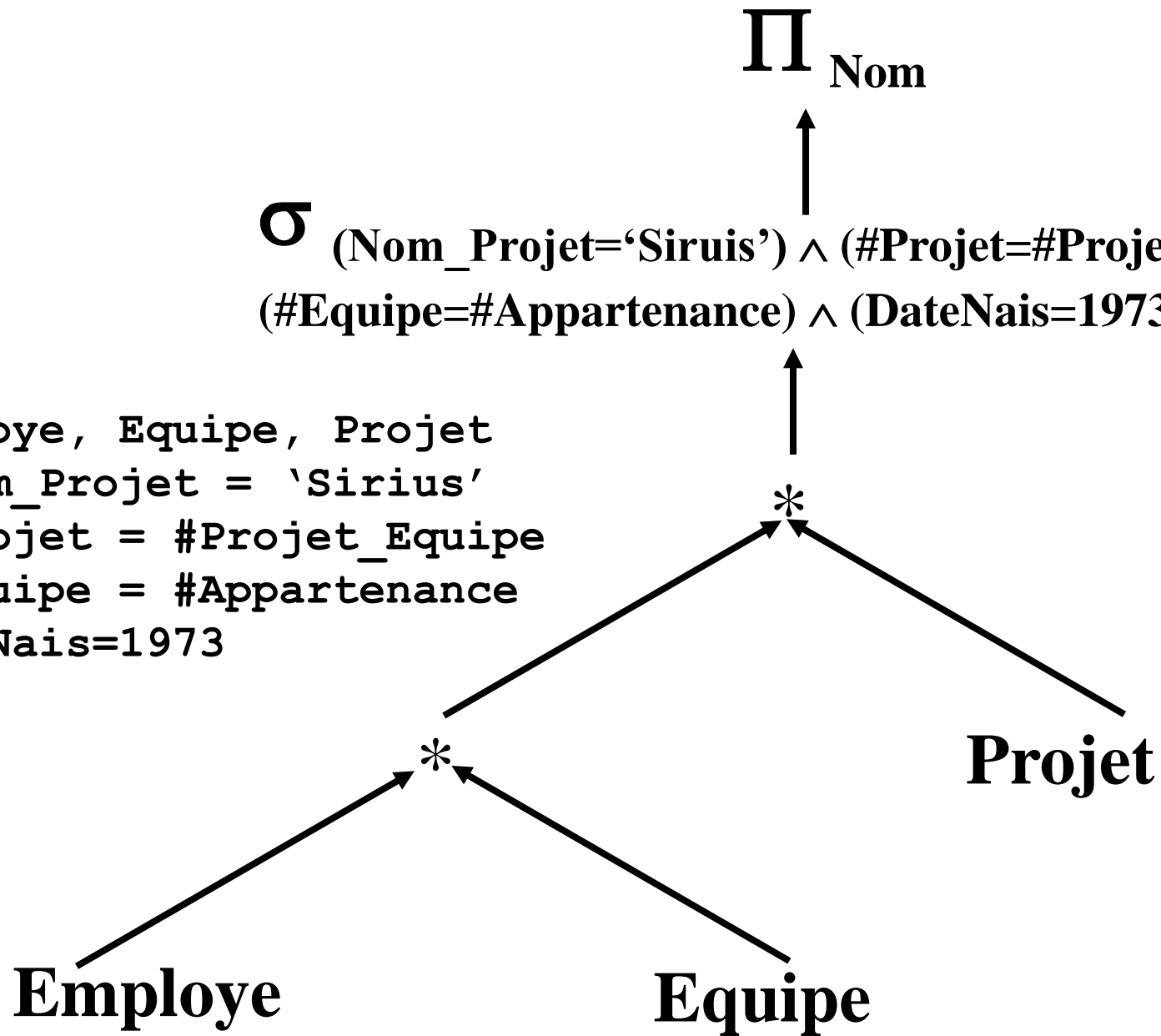
+Commutativité des sélections et des projections)

++Commutativité des jointures et des projections

```

SELECT Nom
FROM Employe, Equipe, Projet
WHERE Nom_Projet = 'Sirius'
AND #Projet = #Projet_Equipe
AND #Equipe = #Appartenance
AND DaeNais=1973

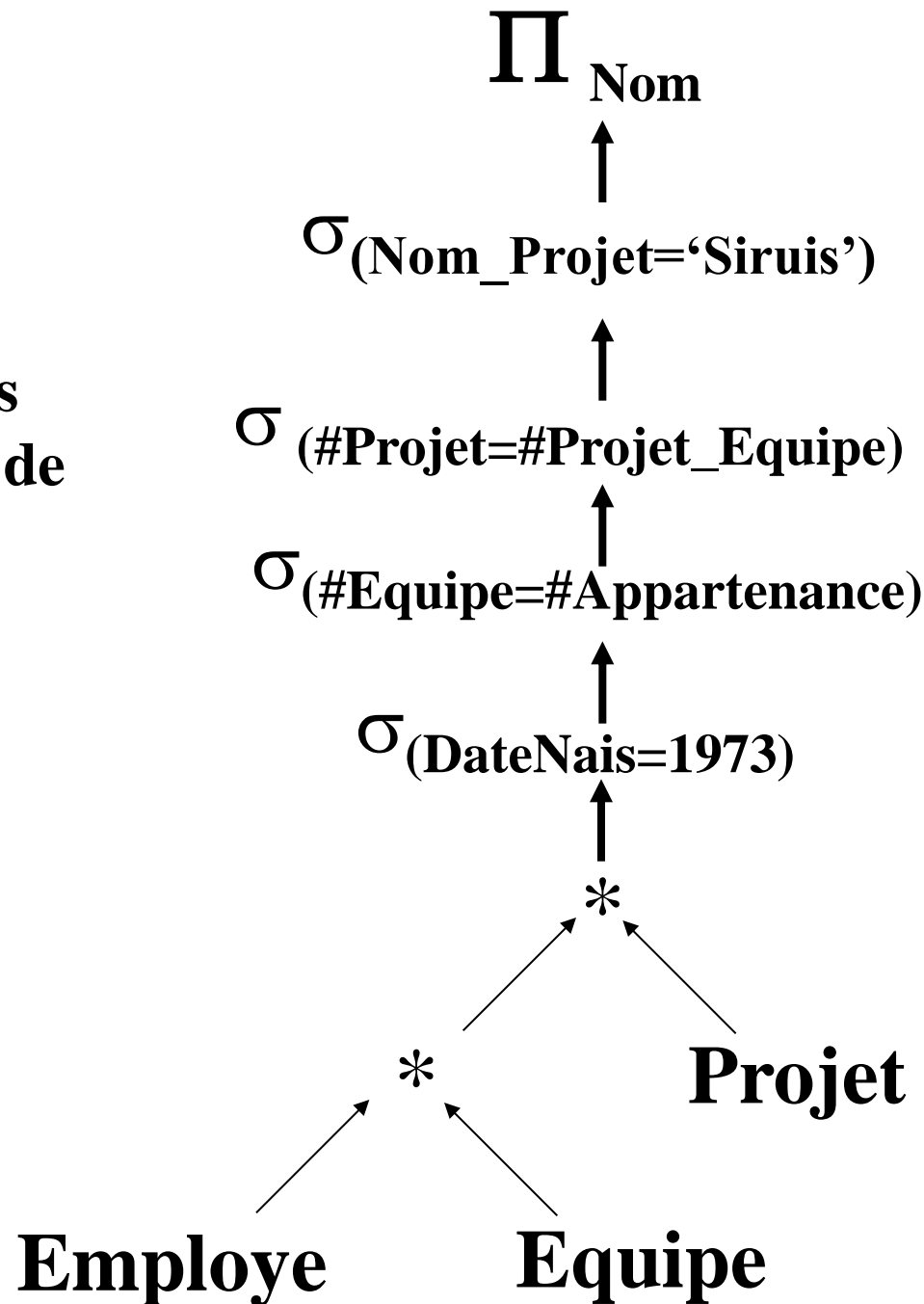
```



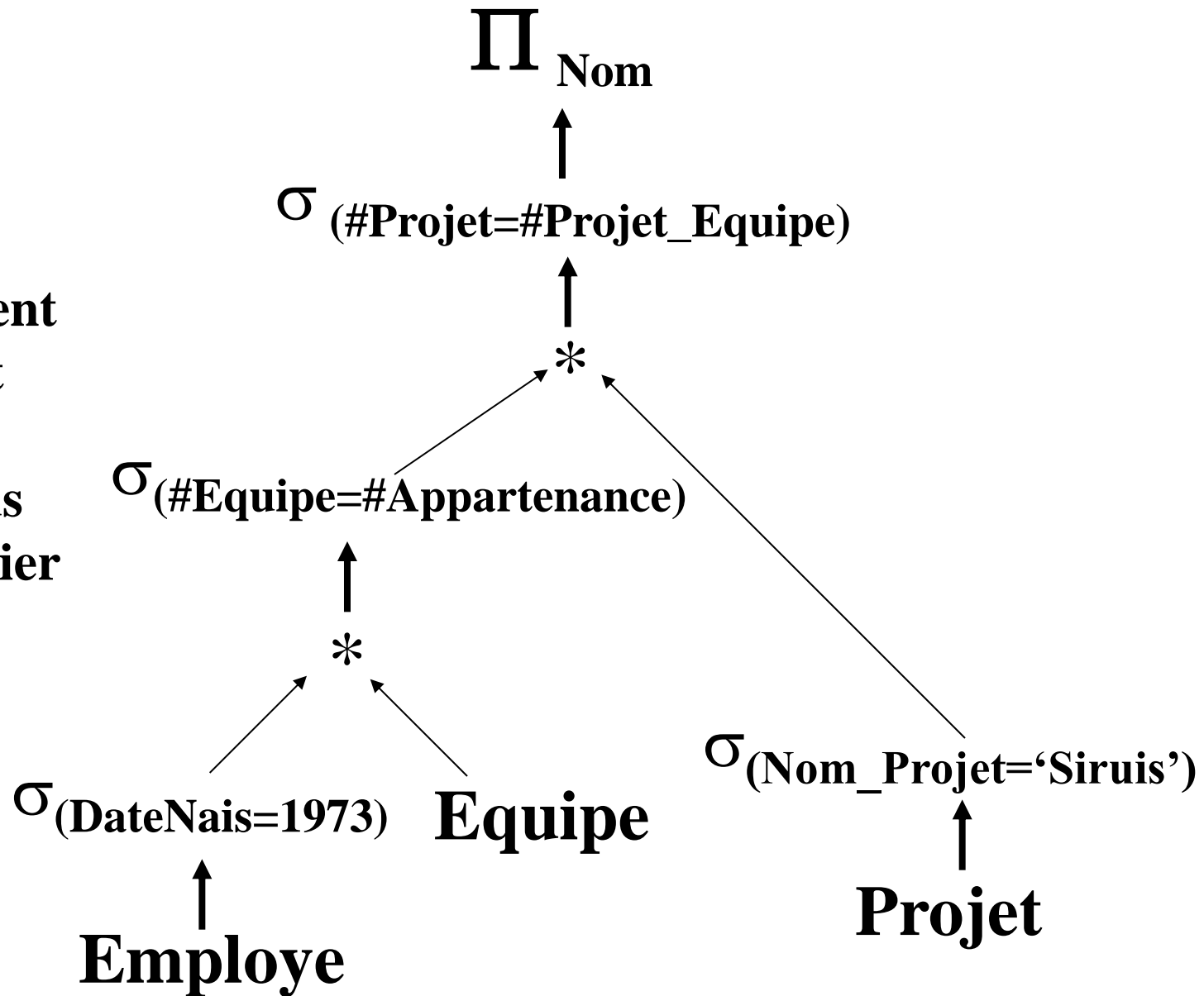
$$\sigma_{(\text{Nom_Projet}='Sirius') \wedge (\#Projet=\#Projet_Equipe) \wedge (\#Equipe=\#Appartenance) \wedge (\text{DateNais}=1973)}$$

$$\Pi_{\text{Nom}}$$

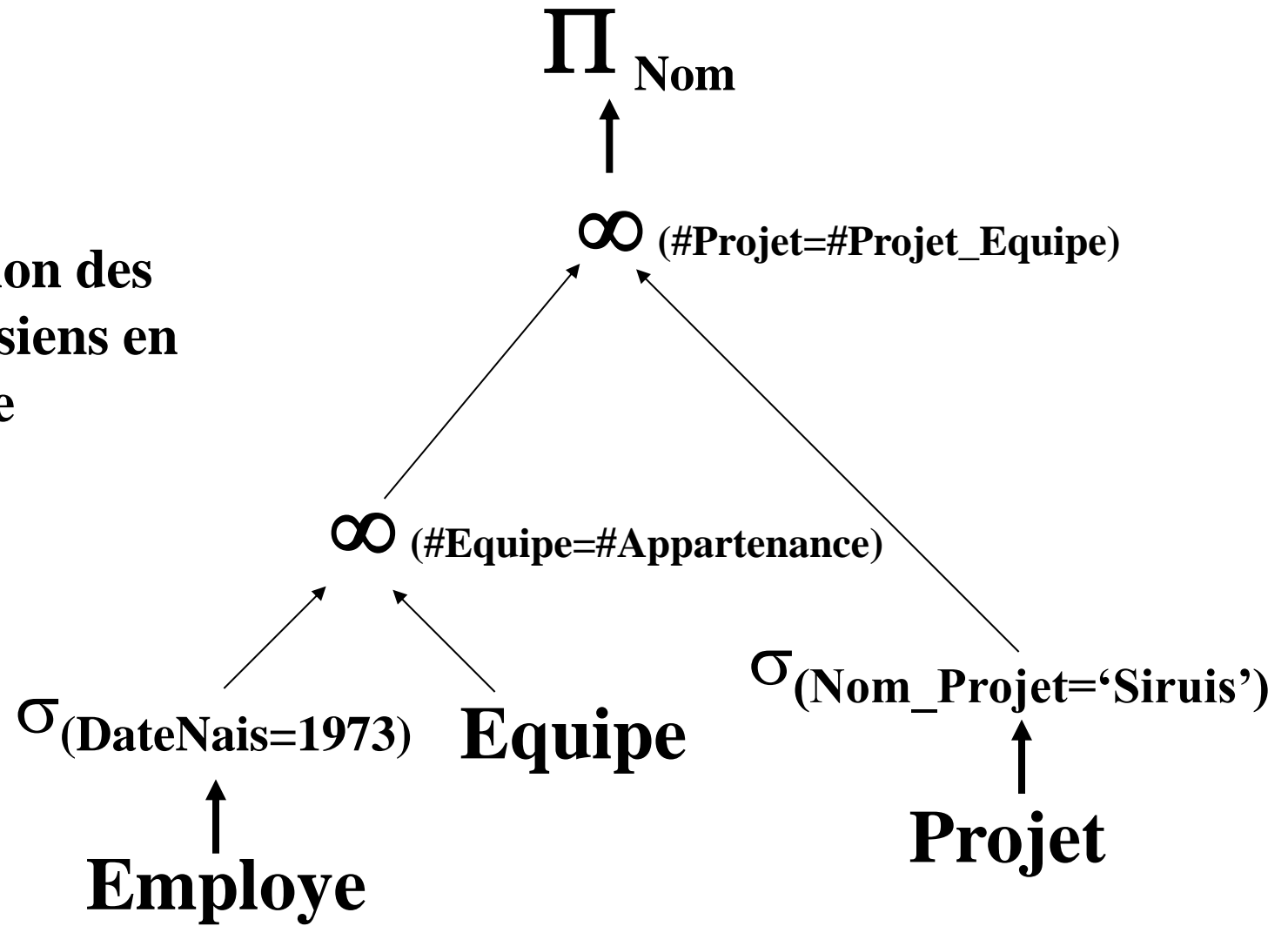
**Division des
conjonctions de
sélections**



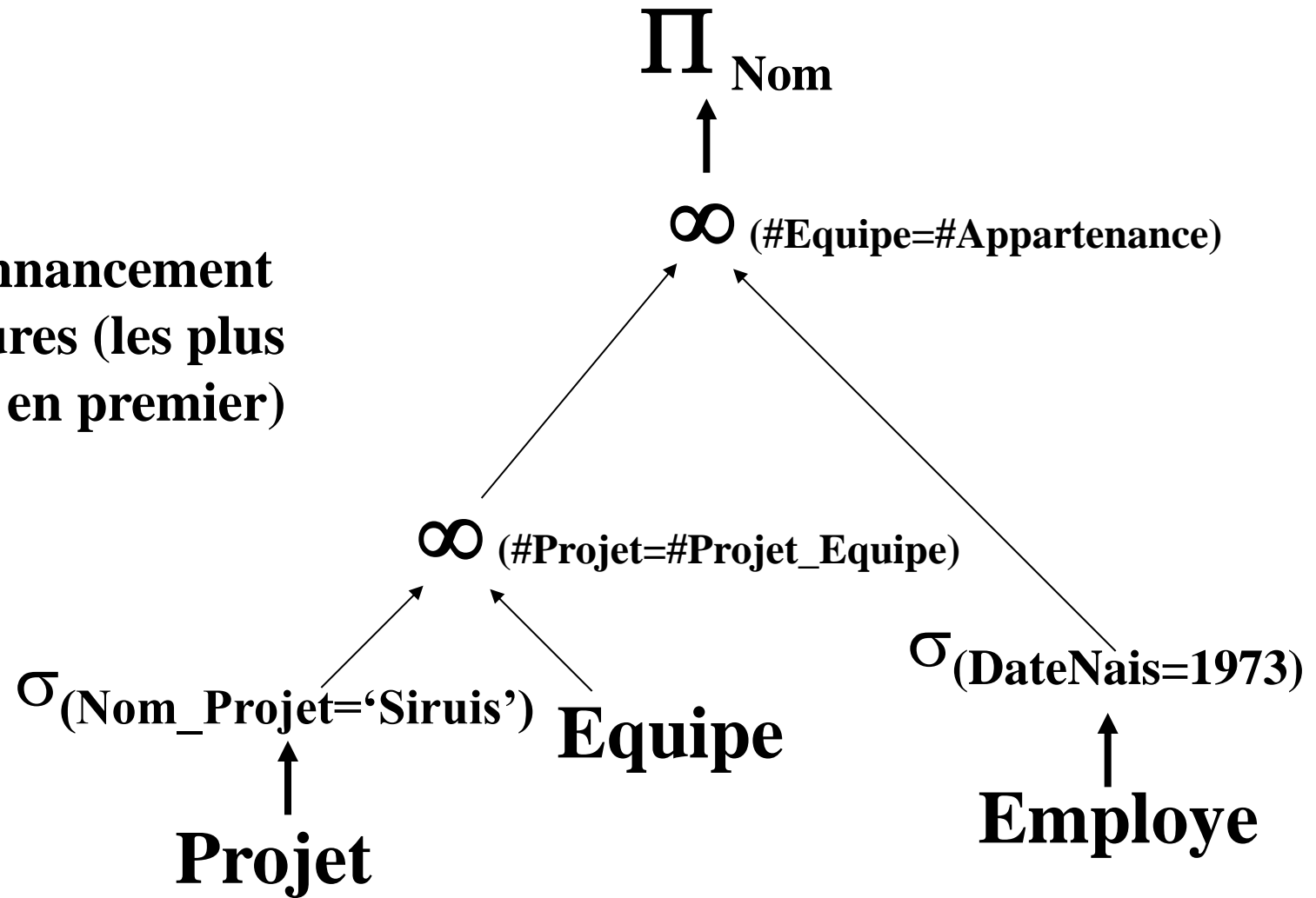
**Ré-ordonnancement
des sélections et
application des
sélections les plus
sélectives en premier**



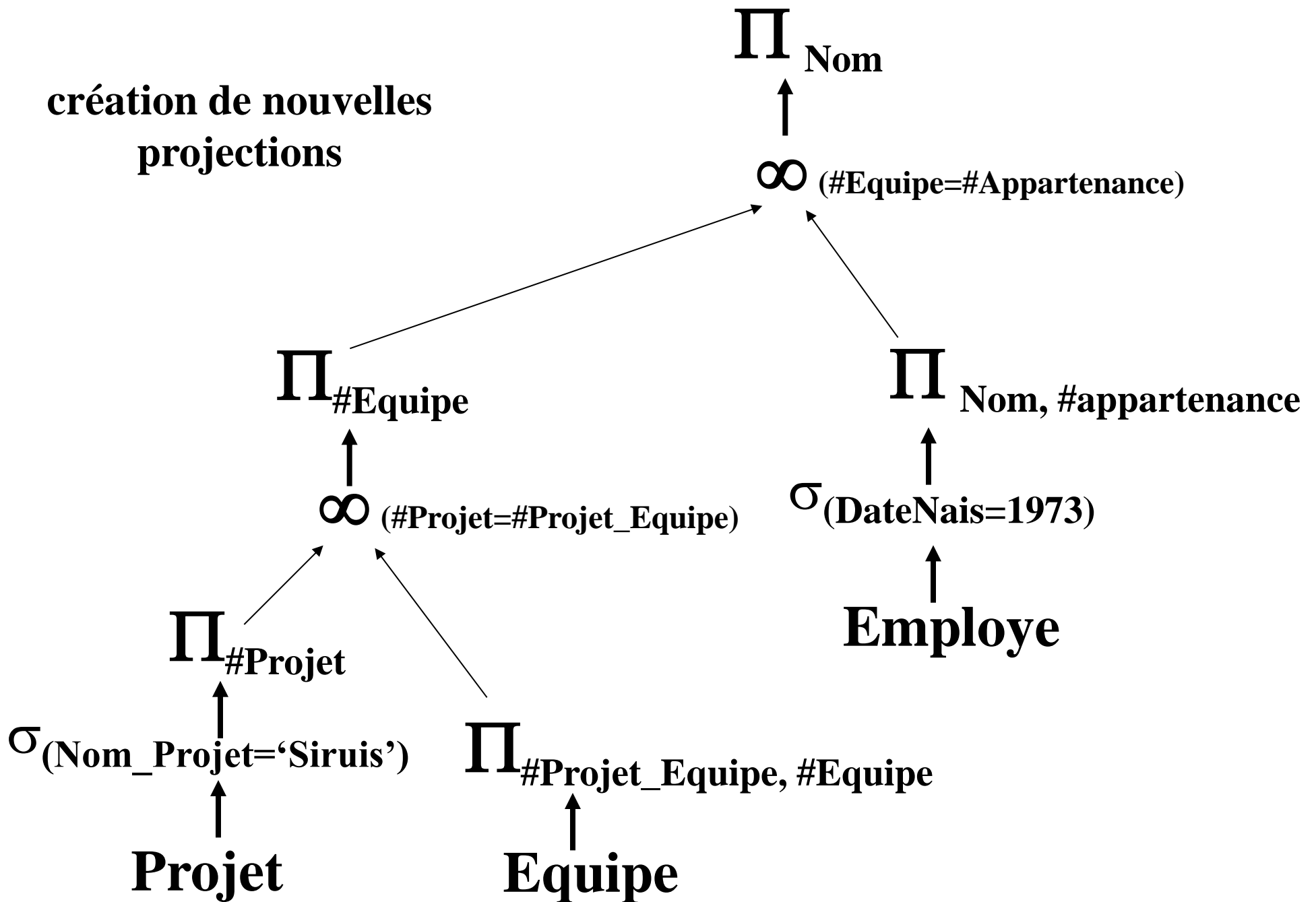
**Transformation des
produits cartésiens en
jointure**



**Ré-ordonnancement
des jointures (les plus
sélectives en premier)**



création de nouvelles
projections



Évaluation de requête

- **Statistiques de la base de données**
- **Coût des opérations**

Statistiques (1/3)

- Pour chaque relation R [CBS98] :
 - ◆ $NTuples(R)$: nombre de nuplets
 - ◆ $Bfactor(R)$: nombre de nuplets par bloc
 - ◆ $NBlocks(R)$: nombre de blocs pour la relation
$$NBlocks(R) = NTuples(R) / Bfactor(R)$$
- Pour chaque attribut A de R :
 - ◆ $NDistinct_A(R)$: nombre de valeurs distinctes de A
 - ◆ $Min_A(R)$ et $Max_A(R)$: valeurs min et max de A
 - ◆ $SC_A(R)$: nombre moyen de nuplets satisfaisant un prédicat sur A ,
en prenant l'hypothèse que les nuplets de R sont répartis
uniformément en fonction des valeurs de A

Statistiques (2/3)

- ◆ $SC_A(R)$ dépend du prédicat
 - Egalité : **1** si A est clé de R, ou $(NTuplets(R) / NDistinct_A(R))$ sinon
 - $A > c$: $NTuplets(R) * ((Max_A(R) - c) / (Max_A(R) - Min_A(R)))$
 - $A < c$: $NTuplets(R) * ((c - Min_A(R)) / (Max_A(R) - Min_A(R)))$
 - $A \in \{c_1, ..., c_n\}$: $NTuplets(R) / (NDistinct_A(R) * n)$
 - $A \wedge B$: $(SCA(R) * SCB(R)) / NTuples(R)$
 - $A_1 \wedge A_2 ... \wedge A_n$: $(SCA_1(R) * ... * SCA_n(R)) / (NTuples(R))^{n-1}$
 - $A \vee B$: $(SCA(R) + SCB(R)) - ((SCA(R) * SCB(R)) / NTuples(R))$

Statistiques (3/3)

- Pour index I de R sur un attribut A :
 - ◆ $NLevels_A(I)$: nombre de niveaux pour I
 - ◆ $NLBlocks_A(I)$: nombre de blocs utilisés pour I

Attention : dans le cas d'index de type 1 (où on a accès directement aux enregistrements du fichier), ce nombre ne tient pas compte du niveau des nœuds feuille

Coût des opérations

- **Coût d'une opération de sélection**
- **Coût d'une opération de jointure**
- **Coût d'une opération de projection**

Coût d'une opération de sélection

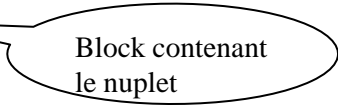
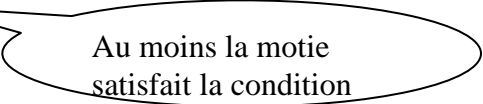
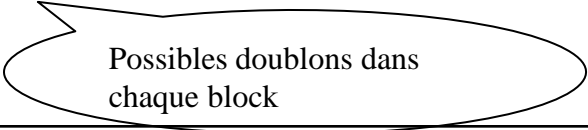
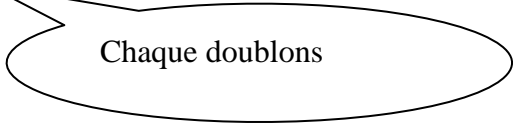
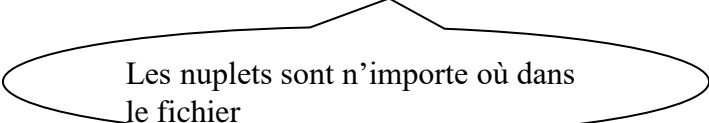
Stratégies

- La recherche linéaire (sur un fichier non ordonné sans index)
- La recherche binaire (sur un fichier ordonné sans index)
- L'égalité sur une clé de hachage
- Une condition d'égalité sur la clé primaire
- Une condition d'inégalité sur la clé primaire
- Une condition d'égalité sur un index secondaire clustered
- Une condition d'égalité sur un index secondaire unclustered
- Une condition d'inégalité sur un index secondaire en arbre B^+

Coût d'une opération de sélection

Stratégie	Coût
recherche linéaire (sur un fichier non ordonné sans index)	$\mathbf{NBlocks(R)}$
recherche binaire (sur un fichier ordonné sans index) si la condition porte sur les attributs sur lesquels le fichier est ordonné	$\mathbf{log2(NBlocks(R))}$ si la condition d'égalité porte sur la clé de la relation ou $\mathbf{log2(NBlocks(R)) + \overbrace{((SCA(R)/BFactor(R)) - 1}^{\text{Doublons possibles}}}$ sinon
L'égalité sur une clé de hachage	$\mathbf{1}$ s'il n'y a pas de page d'overflow

Coût d'une opération de sélection

Stratégie	Coût
recherche par égalité sur la clé primaire	$NLevels_A(I) + 1$  <p>Block contenant le nuplet</p>
Recherche par inégalité sur la clé primaire	$NLevels_A(I) + (NBlocks(R) / 2)$  <p>Au moins la moitié satisfait la condition</p>
Recherche par égalité sur la clé de recherche avec un index secondaire clustered	$NLevels_A(I) + (SC_A(R) / BFactor(R))$  <p>Possibles doublons dans chaque block</p>
Recherche par égalité sur la clé de recherche avec un index secondaire unclustered	$NLevels_A(I) + SC_A(R)$  <p>Chaque doublons</p>
Recherche par inégalité sur la clé de recherche dans un arbre B ⁺	$NLevels_A(I) + (NLBlocks_A(I) / 2) + (NTuples(R) / 2)$  <p>Les nuplets sont n'importe où dans le fichier</p>

Coût d'une opération de jointure

Stratégies d'implantation

- Algorithme des boucles imbriquées (*Block nested loop join*)
- Algorithme des boucles imbriquées en utilisant un index
- Algorithme de tri-fusion (*Sort-merge Join*)
- Algorithme de hachage (*Hash Join*)

Cardinalité de la jointure

La cardinalité du produit cartésien de deux relations R et S est :

$$NTuples(R) * NTuples(S)$$

Il est plus difficile d'estimer la cardinalité du résultat d'une jointure, car cela dépend de la distribution des attributs de jointure.

$$NTuples(R \bowtie S) \leq NTuples(R) * NTuples(S)$$

Cardinalité de la jointure

Lorsque le prédicat est $R.A = S.B$, l'estimation de la cardinalité est :

- Si A est clé de la relation R : $NTuples(R \bowtie S) \leq NTuples(S)$
- Si B est clé de S : $NTuples(R \bowtie S) \leq NTuples(R)$
- Si ni A ni B ne sont clé, alors :

$$NTuples(R \bowtie S) = SCA(R) * NTuples(S)$$

$$NTuples(R \bowtie S) = SCB(S) * NTuples(R)$$

Algorithme des boucles imbriquées

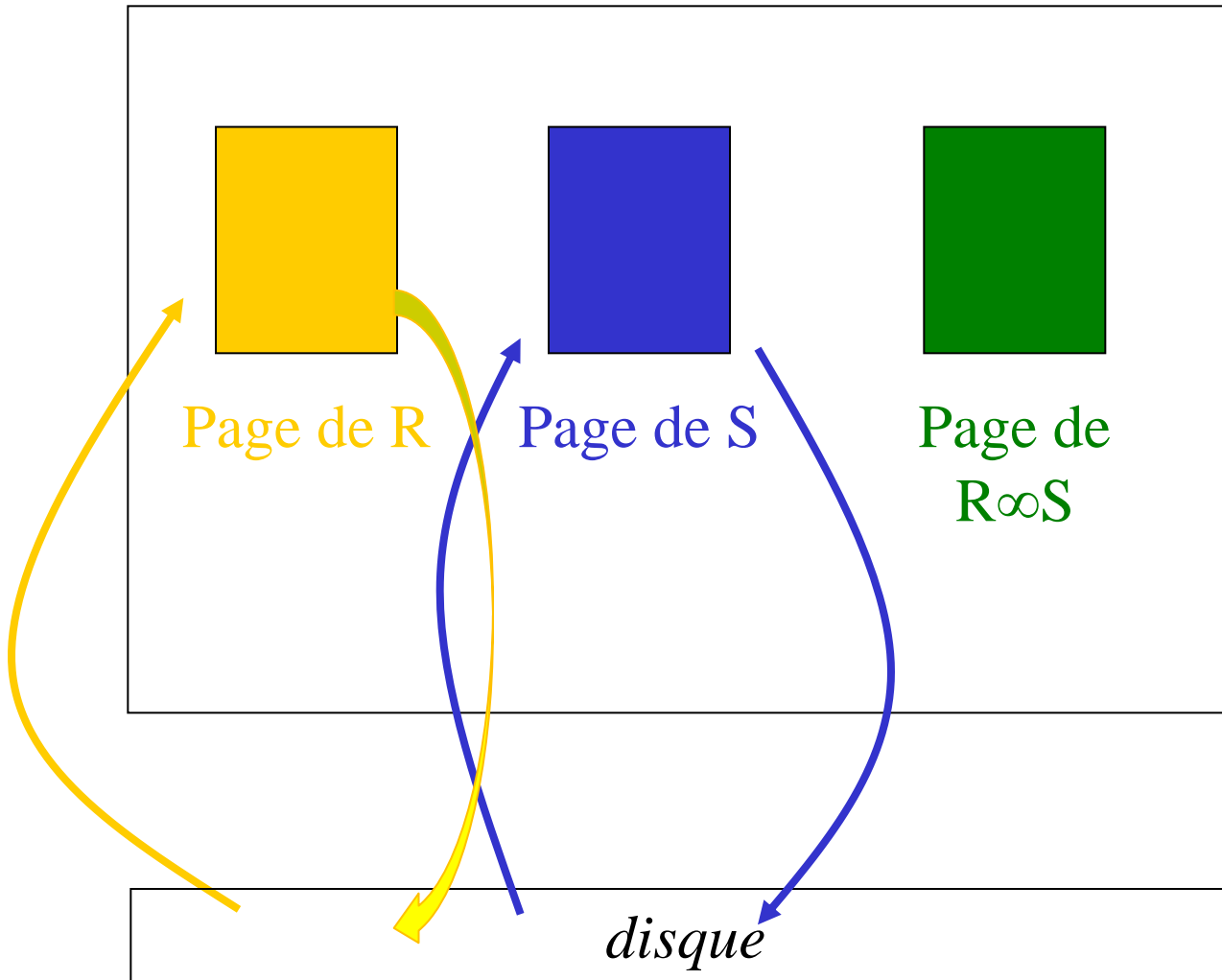
Consiste à monter en mémoire une page de la relation R et d'effectuer la jointure des enregistrements de cette page avec tous les enregistrements de la relation S **en montant une à une les pages de S.**

$$\mathbf{NBlocks(R) + (NBlocks(R) * NBlocks(S))}$$

Les blocs de S sont montés en mémoire autant de fois qu'il y a de pages pour R. Cette valeur dépend donc de la première relation choisie pour la jointure.

Boucles imbriquées

Zone mémoire du buffer



Les blocs de S sont montés en mémoire autant de fois qu'il y a de blocs pour R

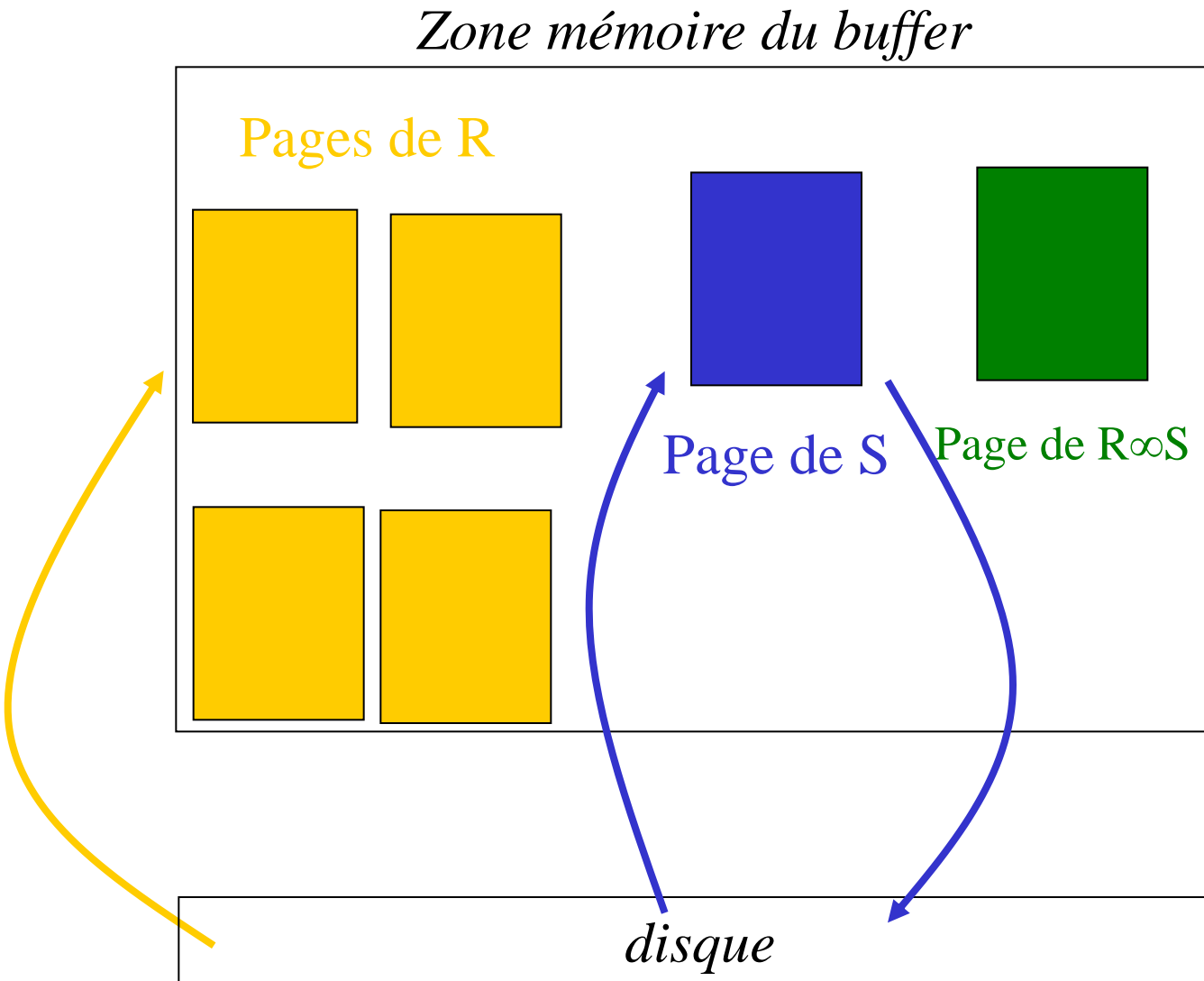
Algorithme des boucles imbriquées

Il est possible d'améliorer le coût **en utilisant au maximum l'espace du buffer de mémoire**. Si le buffer peut contenir *nbuffer* blocs en mémoire, on place (*nbuffer* - 2) blocs de la plus petite relation en mémoire (supposons R), un bloc de la deuxième relation et un bloc pour la relation résultat.

$$\mathbf{NBlocks(R) + [(NBlocks(R)/(nbuffer - 2)) * NBlocks(S)]}$$

Si R tient entièrement en mémoire : **$NBlocks(R) + NBlocks(S)$**

Boucles imbriquées



S'il y a $NBuffer$ places en mémoire, on monte les pages de R en mémoire par paquets de $(NBuffer - 2)$ pages

S'il existe un index sur l'attribut de jointure d'une des deux relations, on l'utilise pour trouver les nuplets participant à la jointure

Algorithme des boucles imbriquées avec index

S'il existe un index (arbre B+ ou fichier de hachage) sur l'attribut de jointure pour une des deux relations, il est préférable d'utiliser pour trouver les enregistrements participant à la jointure

Si l'attribut A est la clé primaire de S, le coût est :

- $NBlocks(R) + NTuples(R) * (NLevelsA(S) + 1)$

Si l'attribut de jointure est indexé par un index clustered sur l'attribut A, le coût est :

$$NBlocks(R) + NTuples(R) * (NLevelsA(S) + (SCA(R) / BFactor(R)))$$

Algorithme de tri-fusion

① **Tri des relations sur l'attribut de jointure**

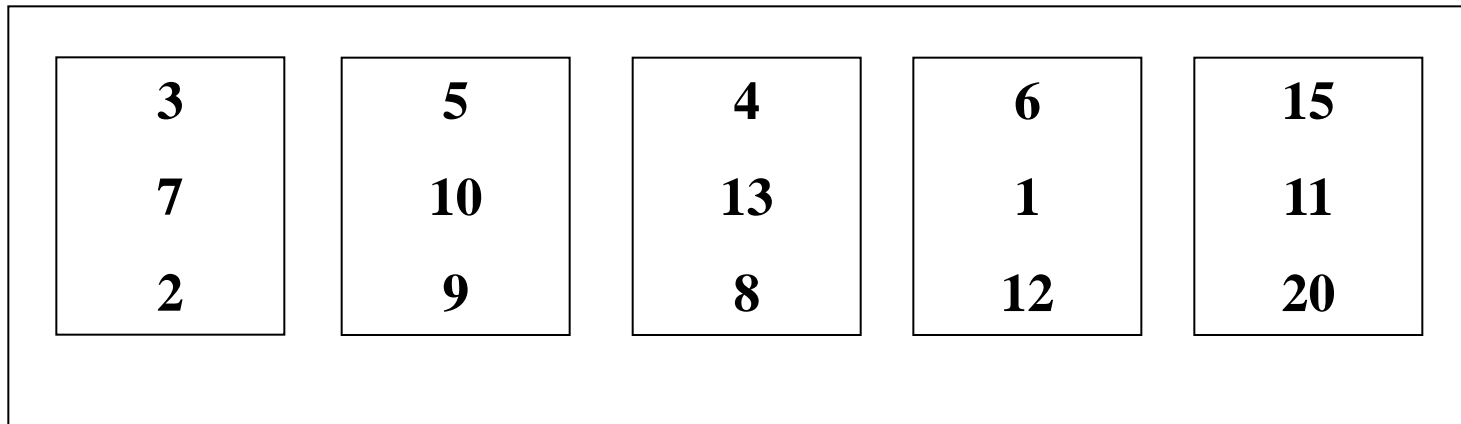
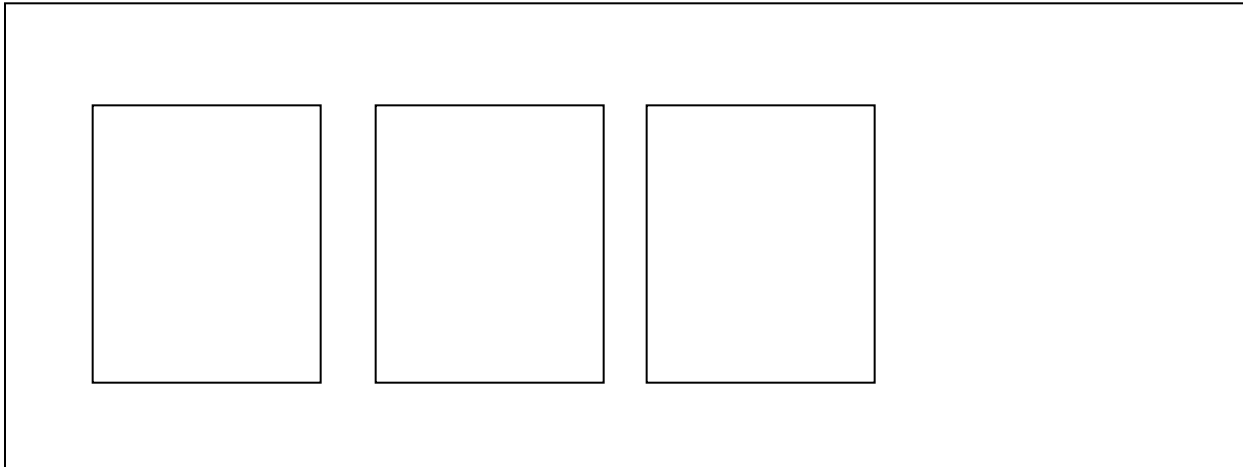
Ex. Tri externe ou tri par séries monotones

② **Equi-jointure des deux relations triées**

En parcourant simultanément les deux relations pages par pages

Tri externe : 1ère étape

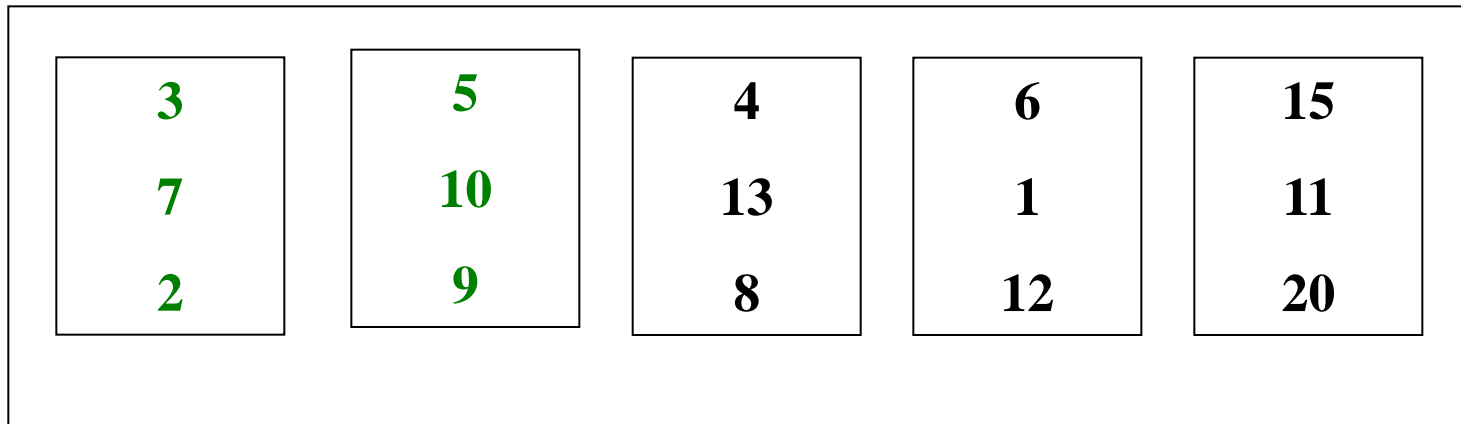
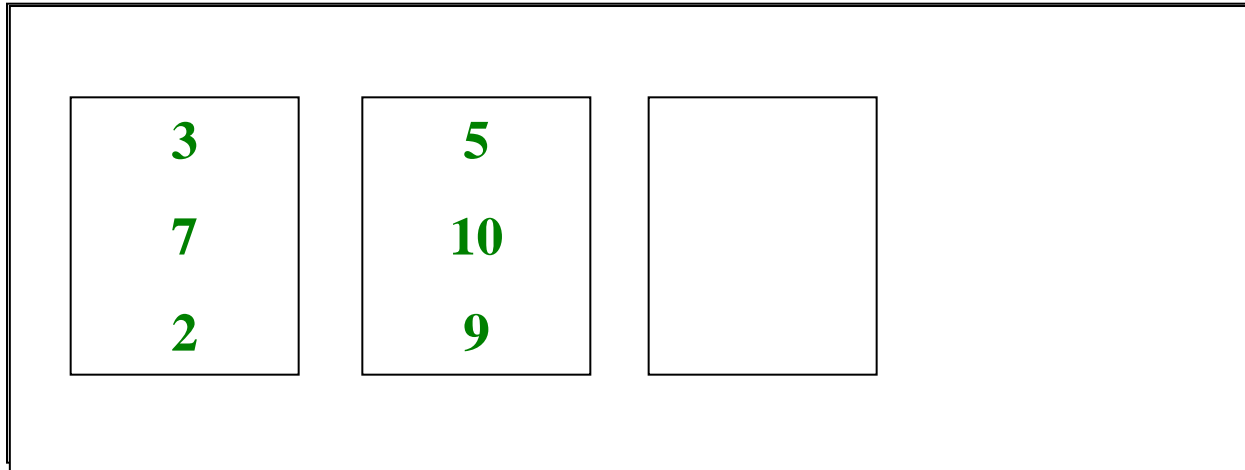
*La mémoire
contient 3
emplacements*



disque

Tri externe : 1ère étape

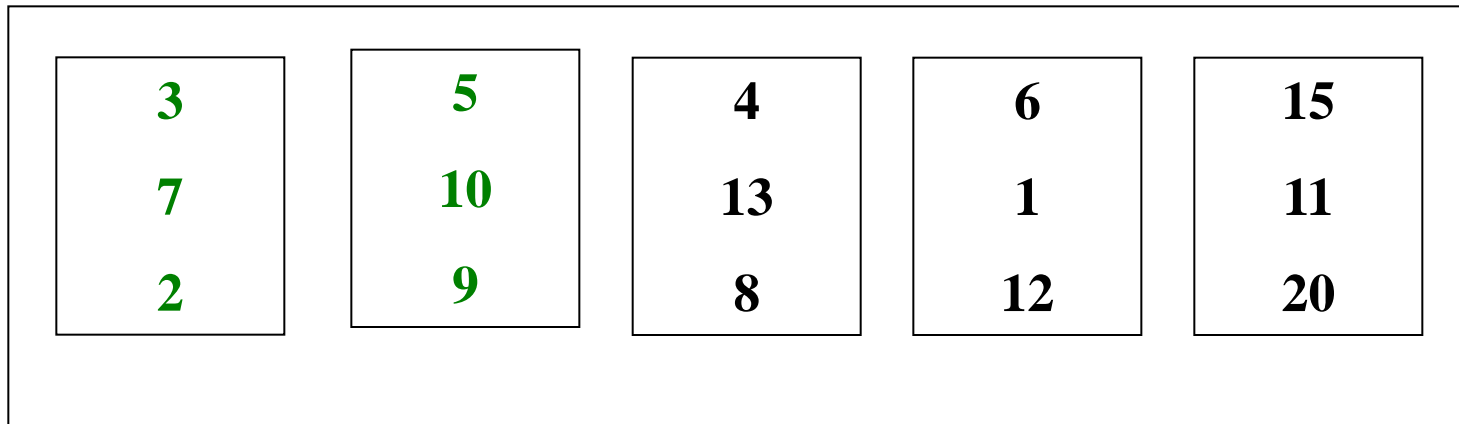
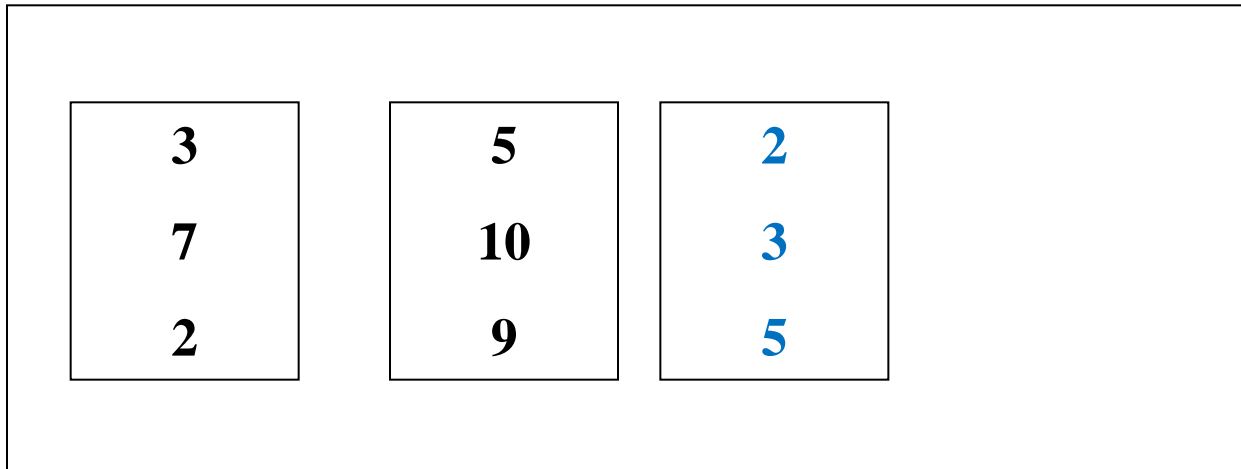
*La mémoire
contient 3
emplacements*



disque

Tri externe : 1ère étape

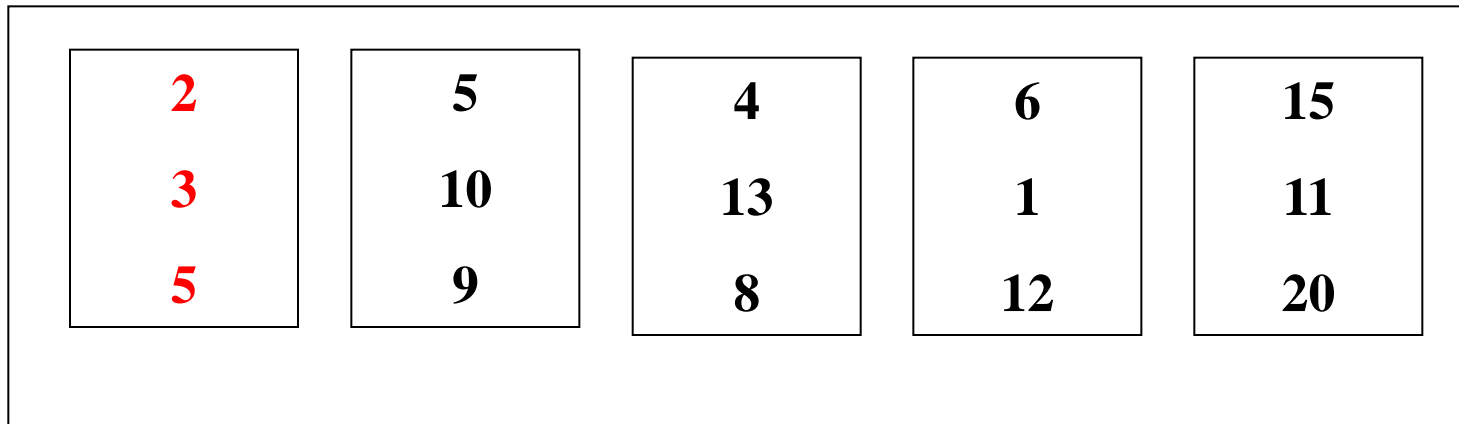
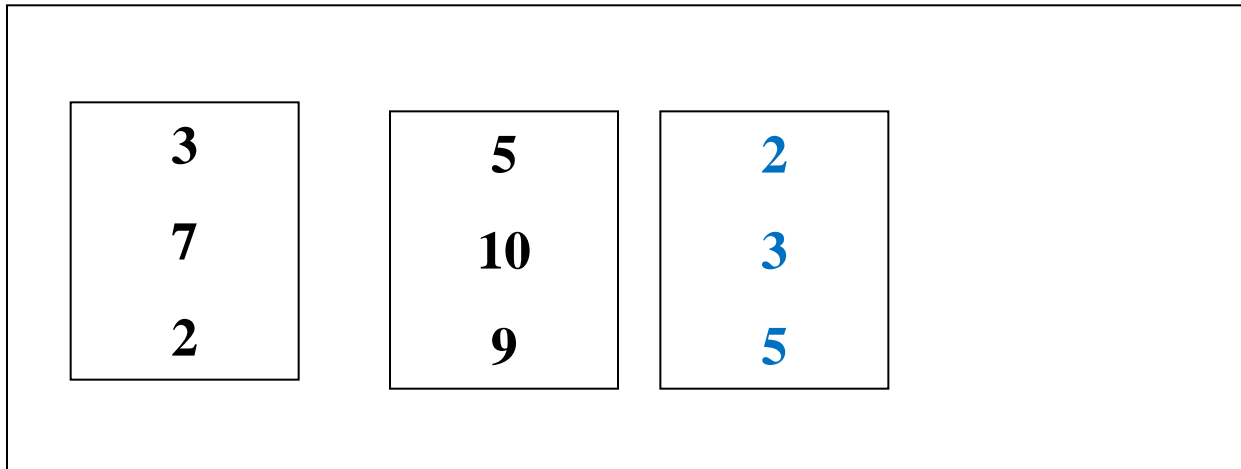
*La mémoire
contient 3
emplacements*



disque

Tri externe : 1ère étape

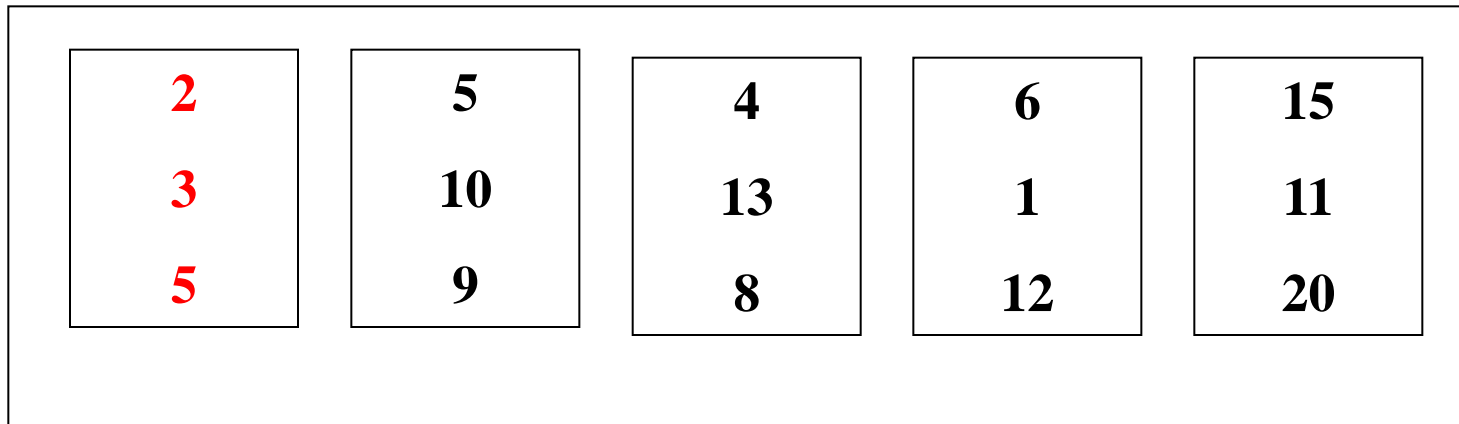
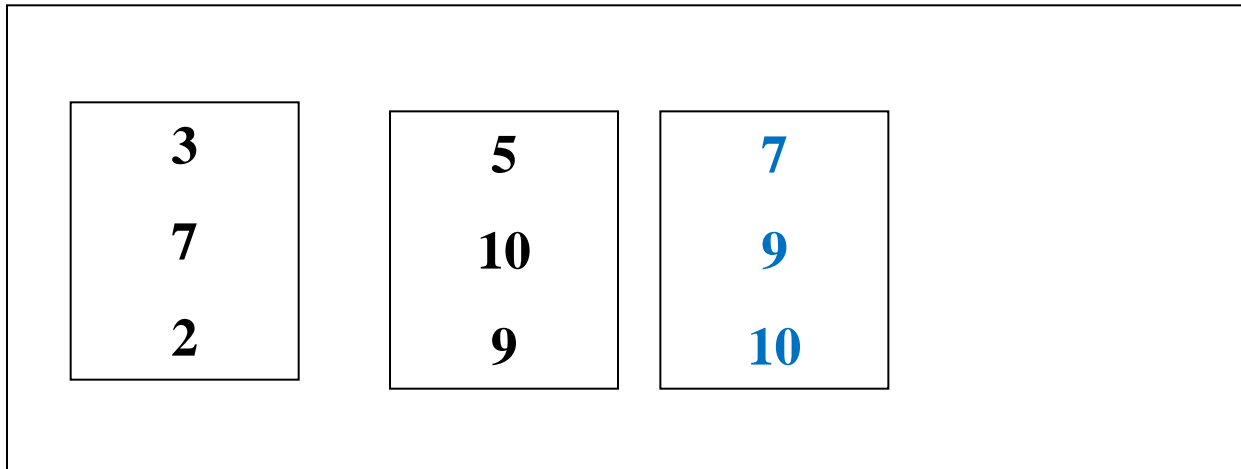
*La mémoire
contient 3
emplacements*



disque

Tri externe : 1ère étape

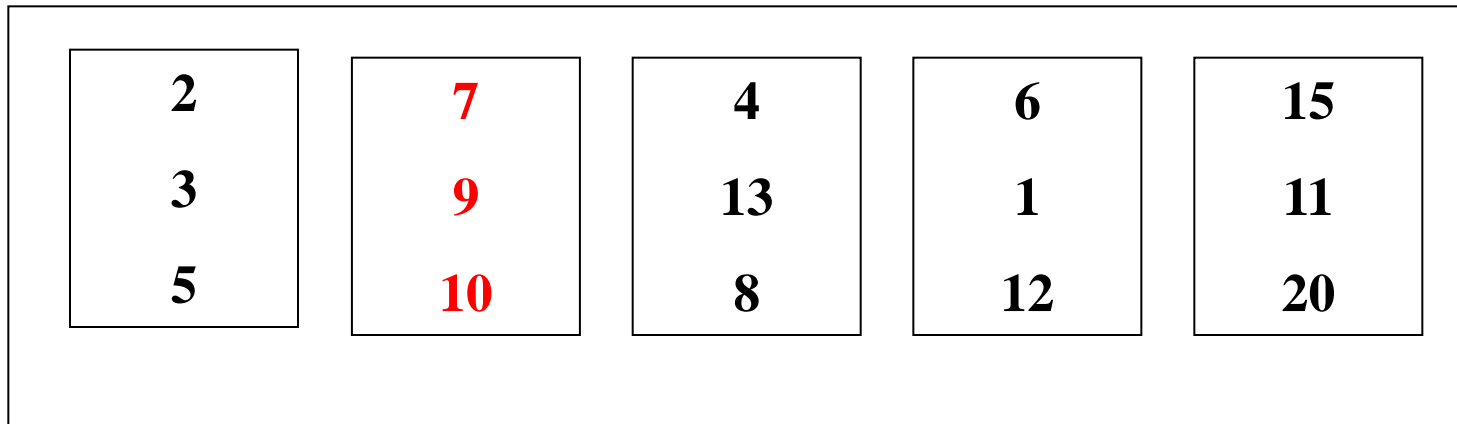
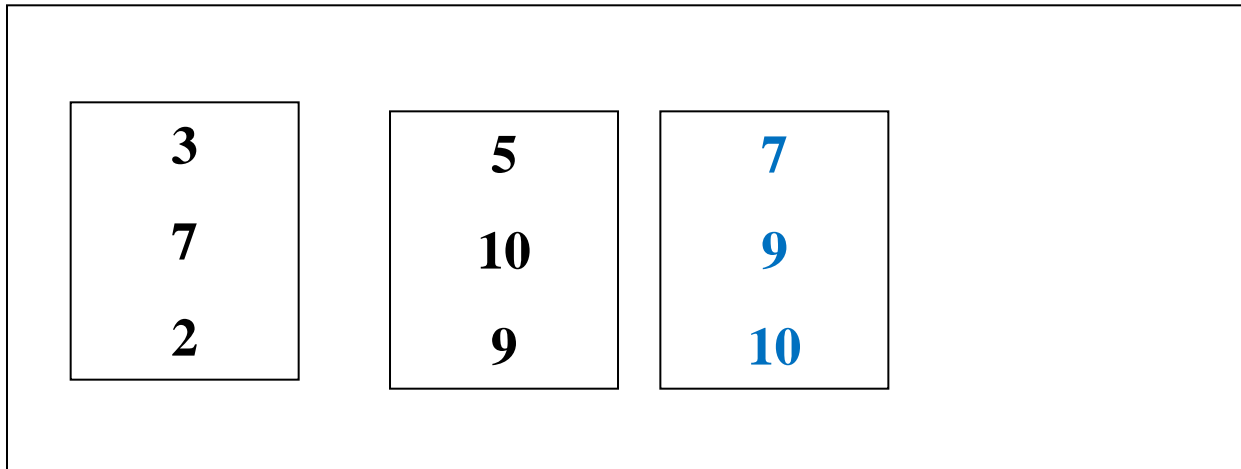
*La mémoire
contient 3
emplacements*



disque

Tri externe : 1ère étape

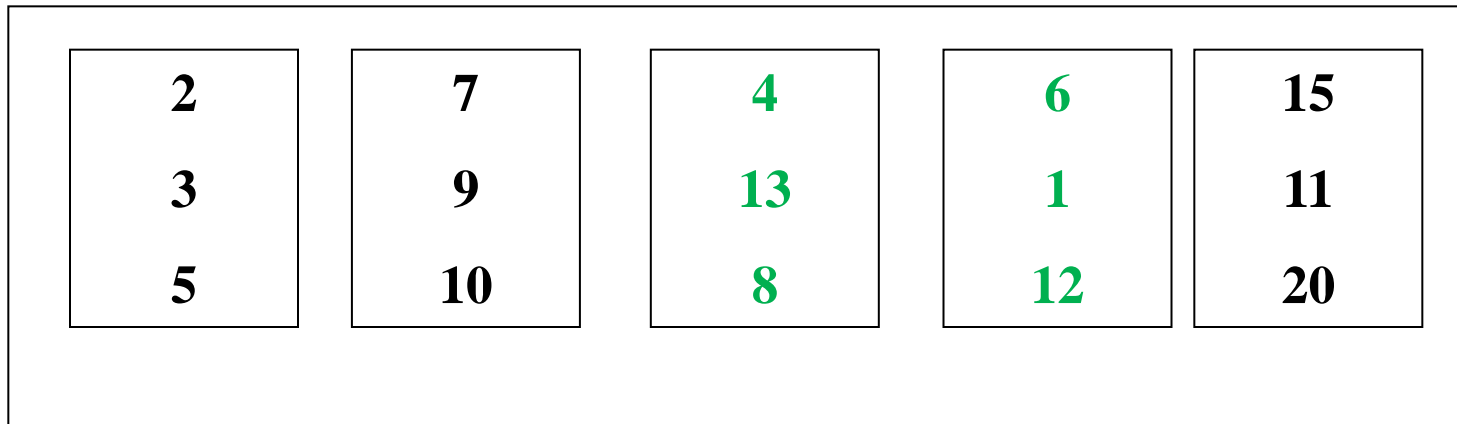
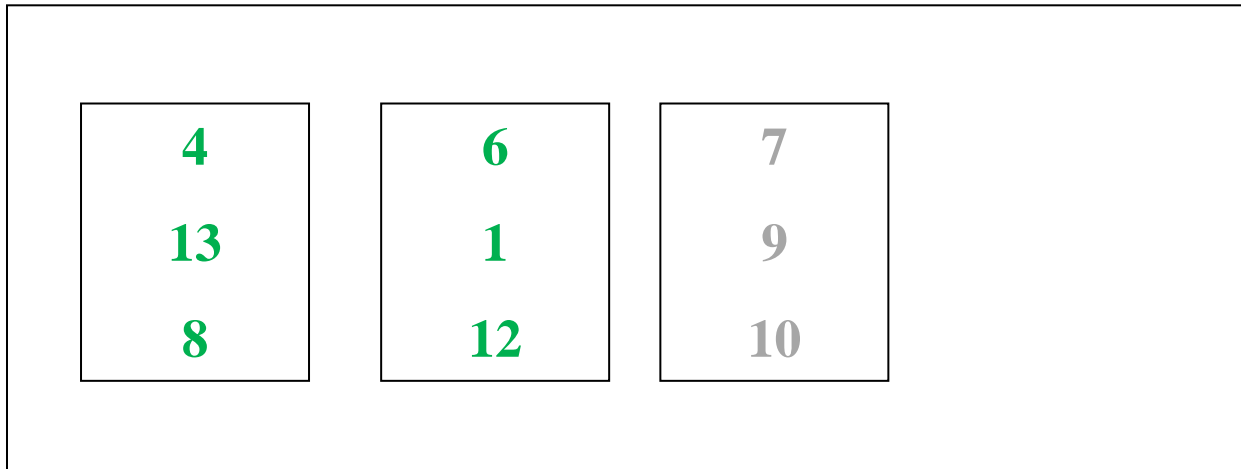
*La mémoire
contient 3
emplacements*



disque

Tri externe : 1ère étape

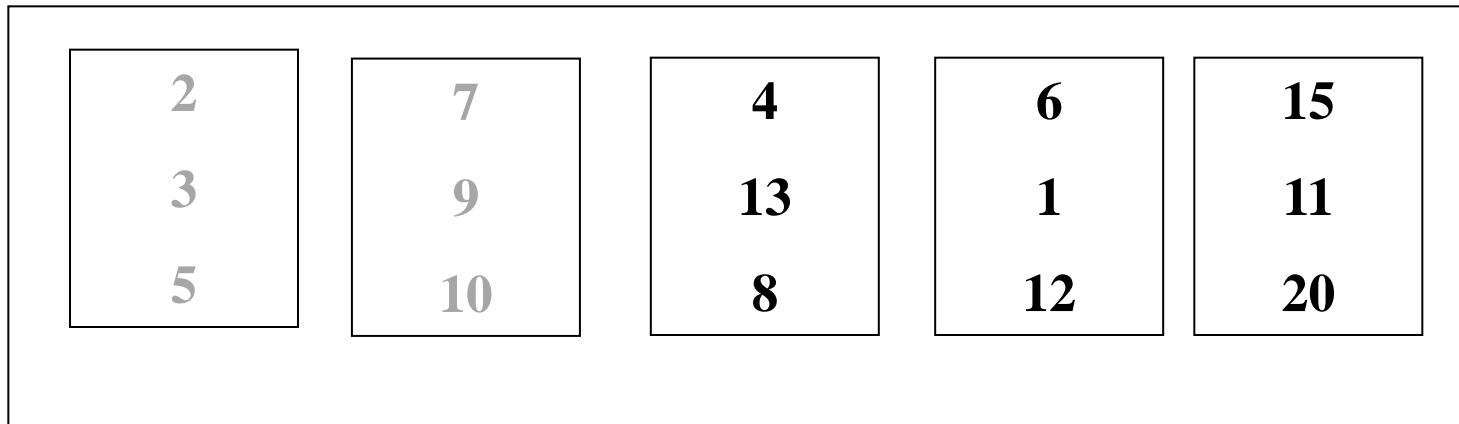
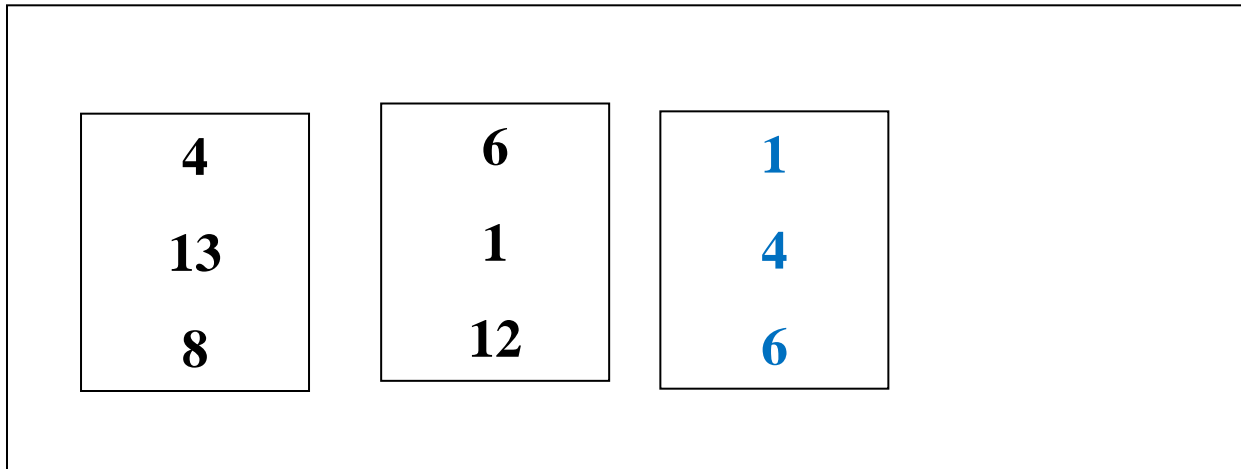
*La mémoire
contient 3
emplacements*



disque

Tri externe : 1ère étape

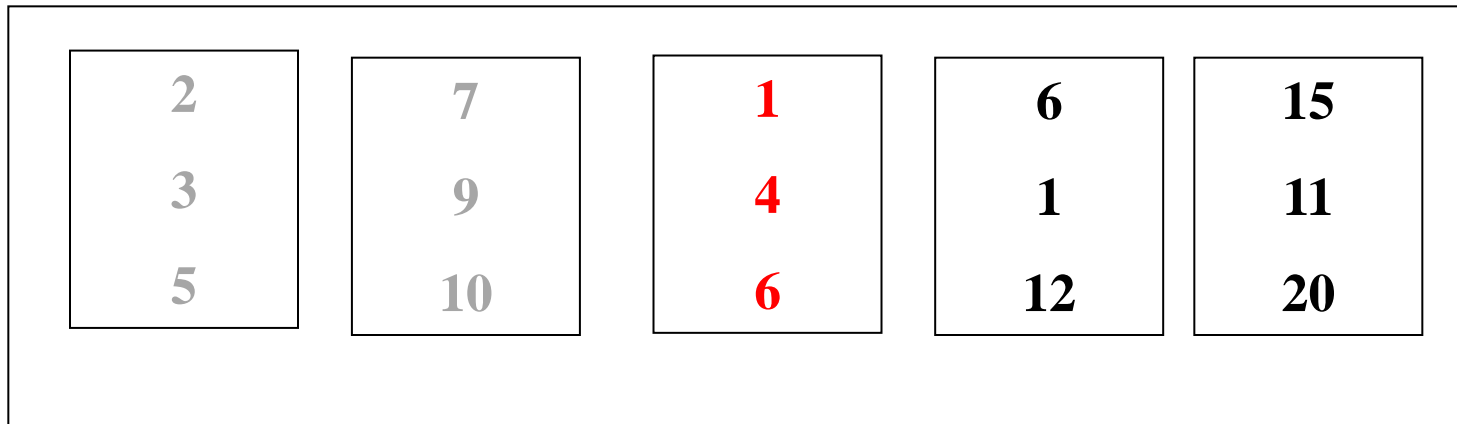
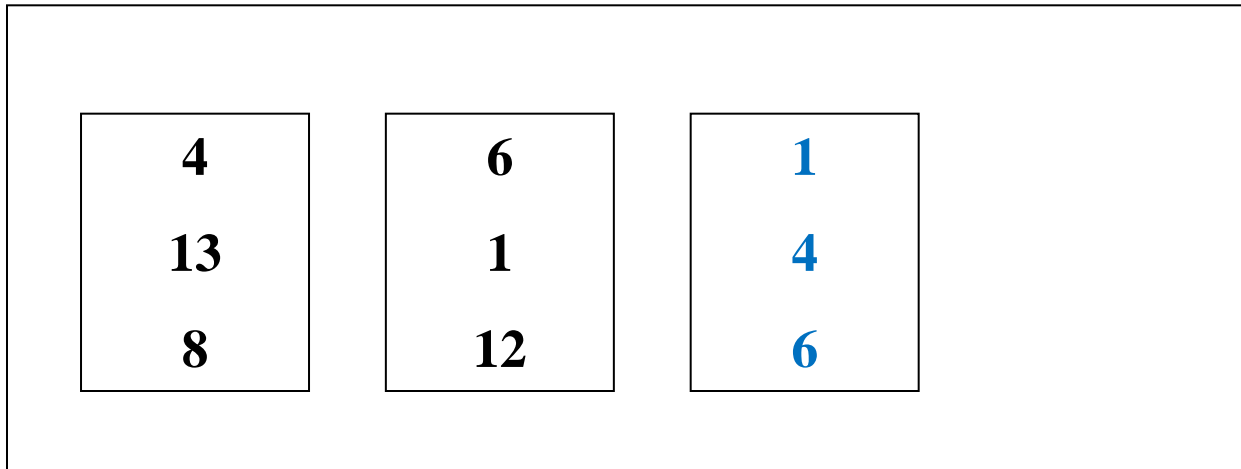
*La mémoire
contient 3
emplacements*



disque

Tri externe : 1ère étape

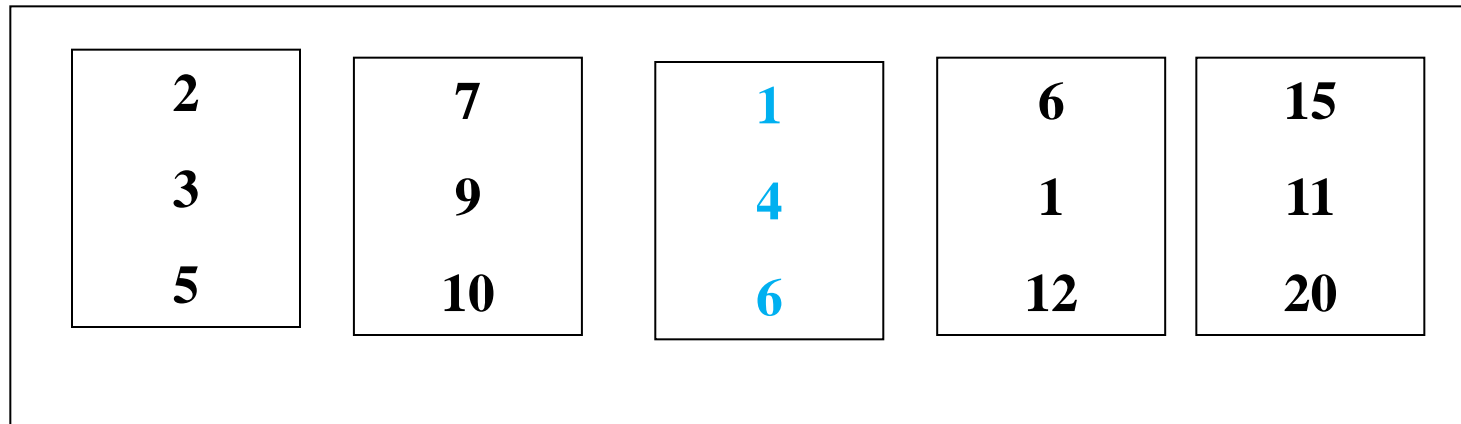
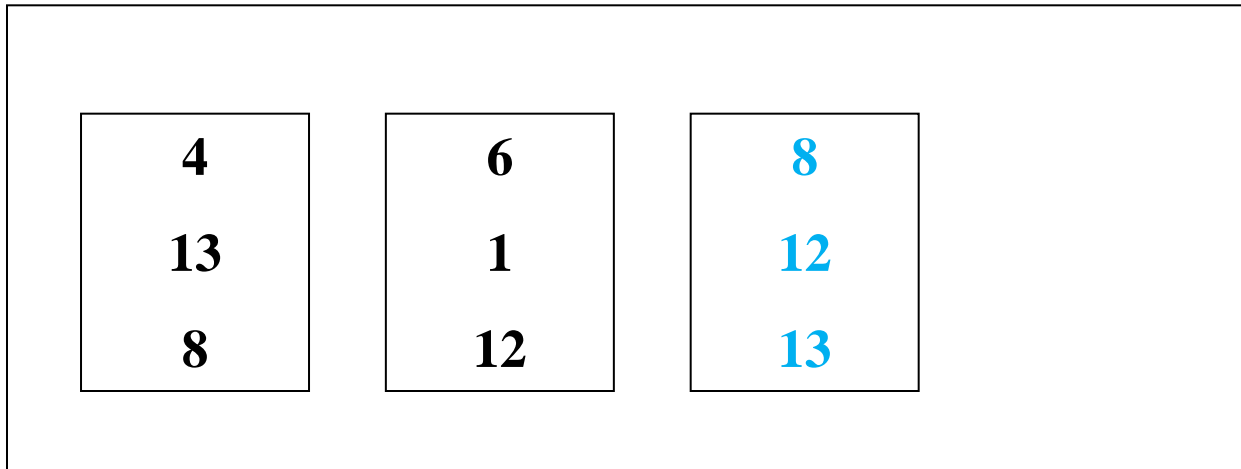
*La mémoire
contient 3
emplacements*



disque

Tri externe : 1ère étape

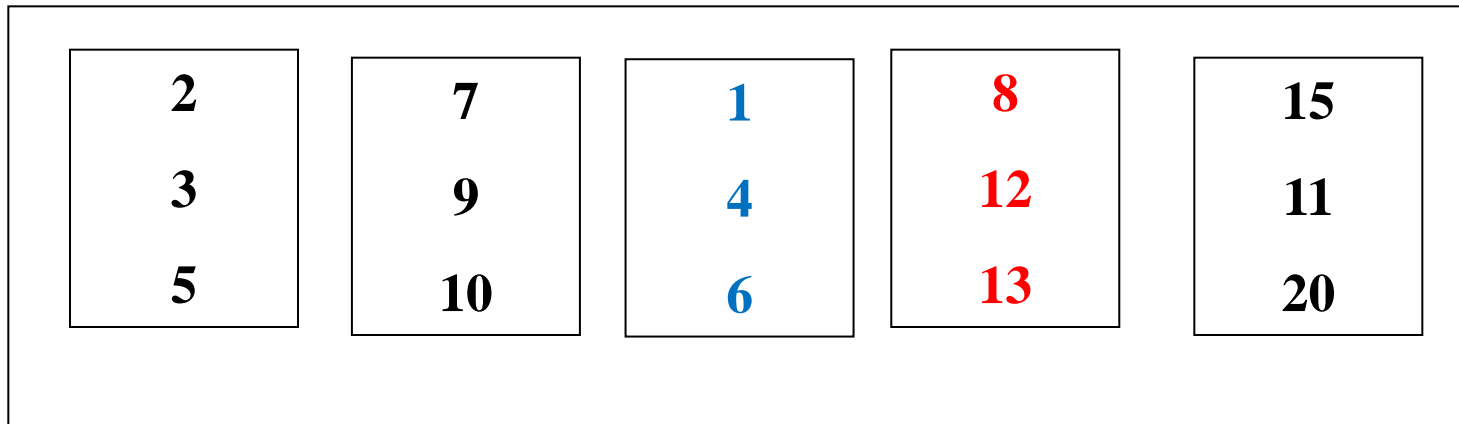
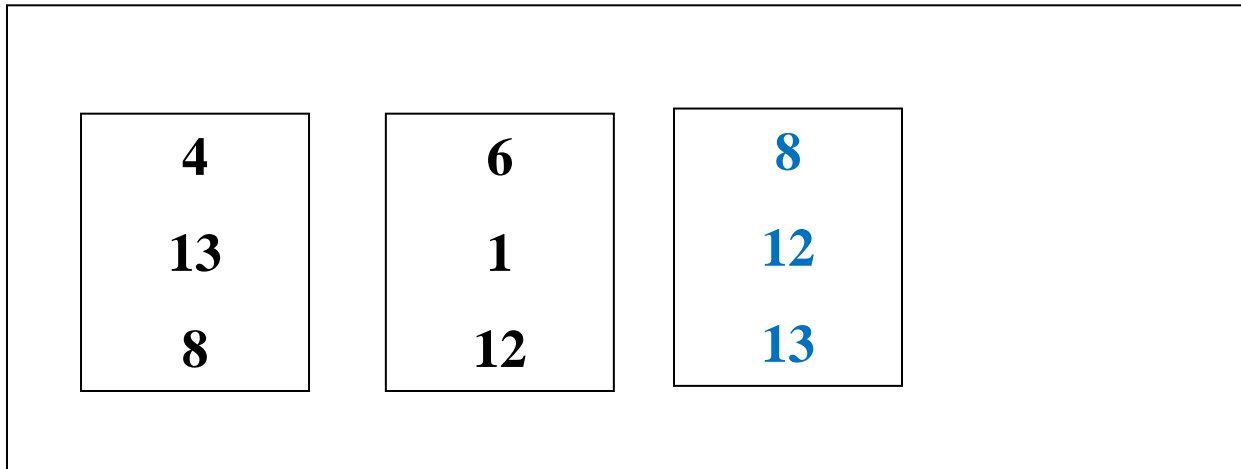
*La mémoire
contient 3
emplacements*



disque

Tri externe : 1ère étape

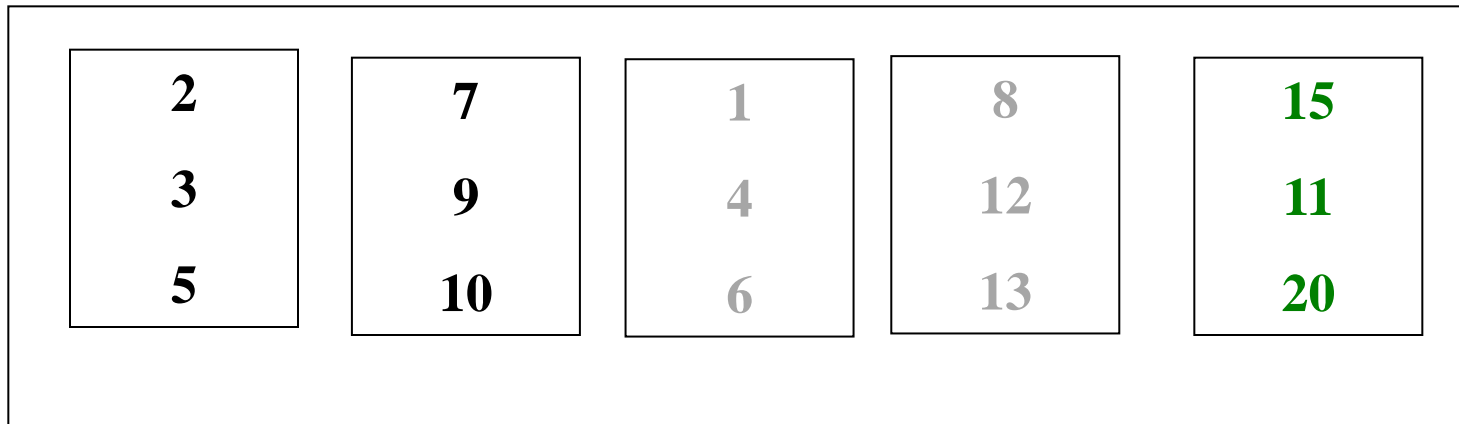
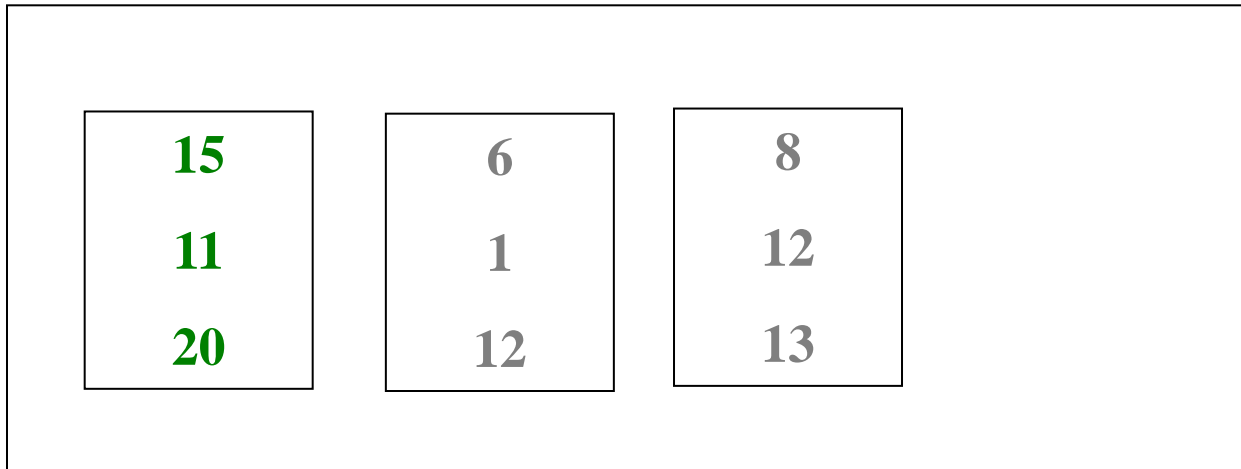
*La mémoire
contient 3
emplacements*



disque

Tri externe : 1ère étape

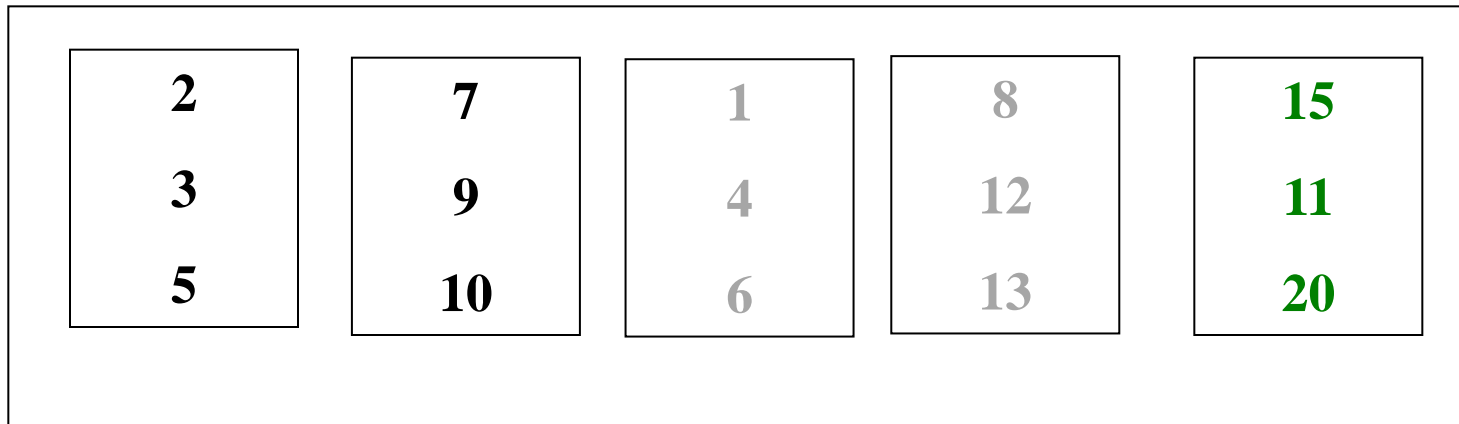
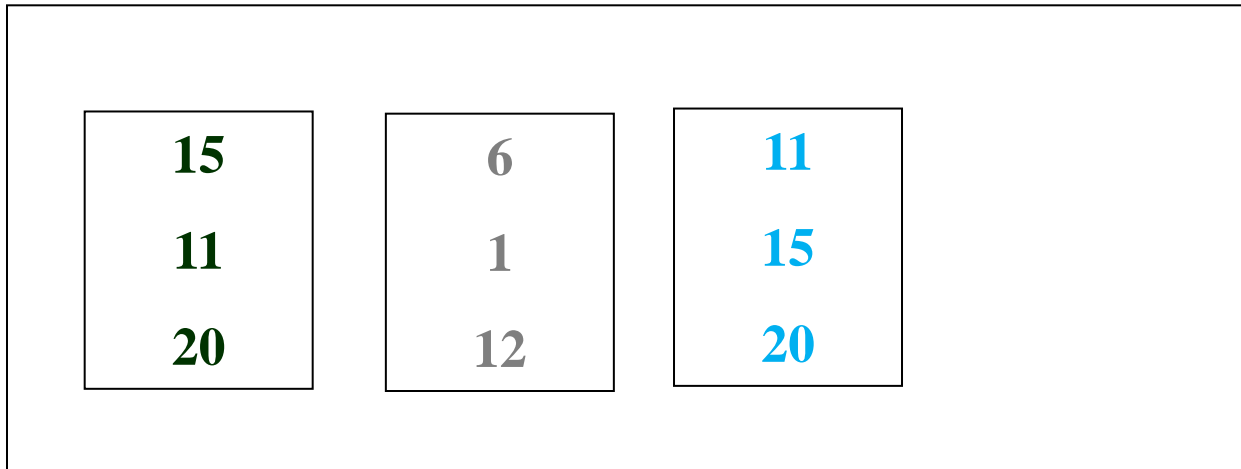
*La mémoire
contient 3
emplacements*



disque

Tri externe : 1ère étape

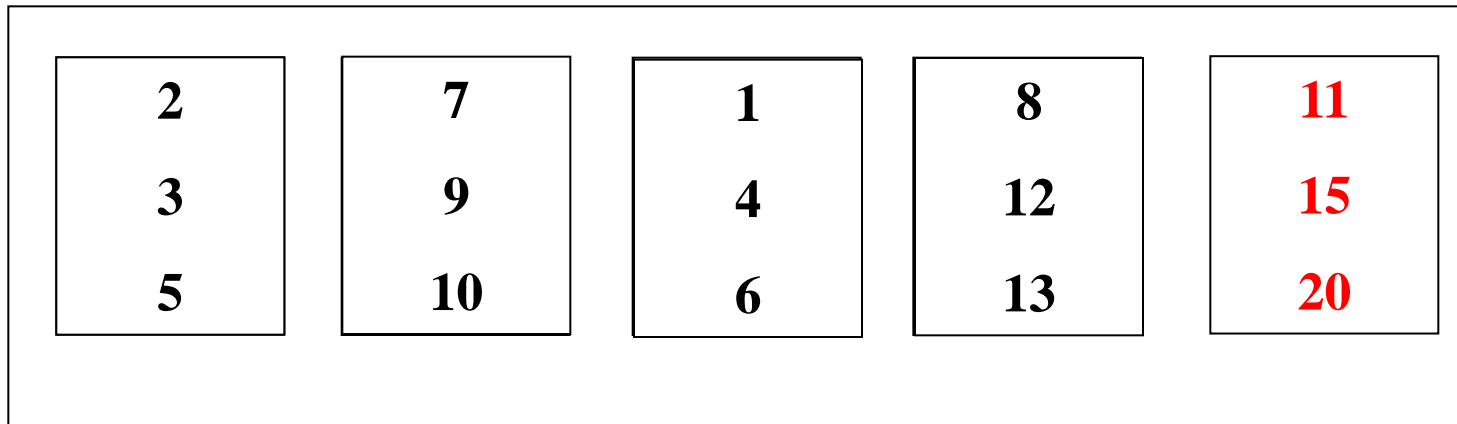
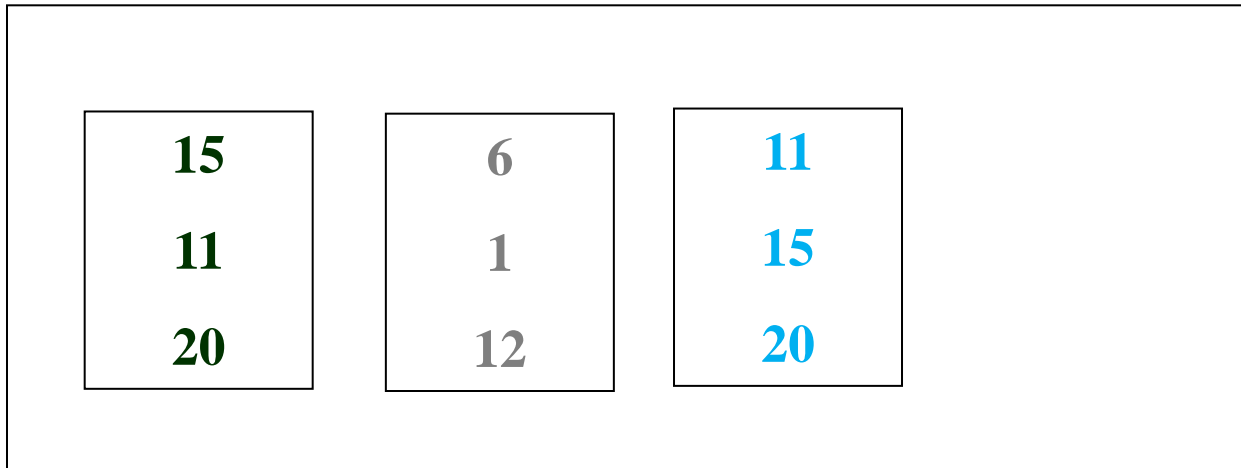
*La mémoire
contient 3
emplacements*



disque

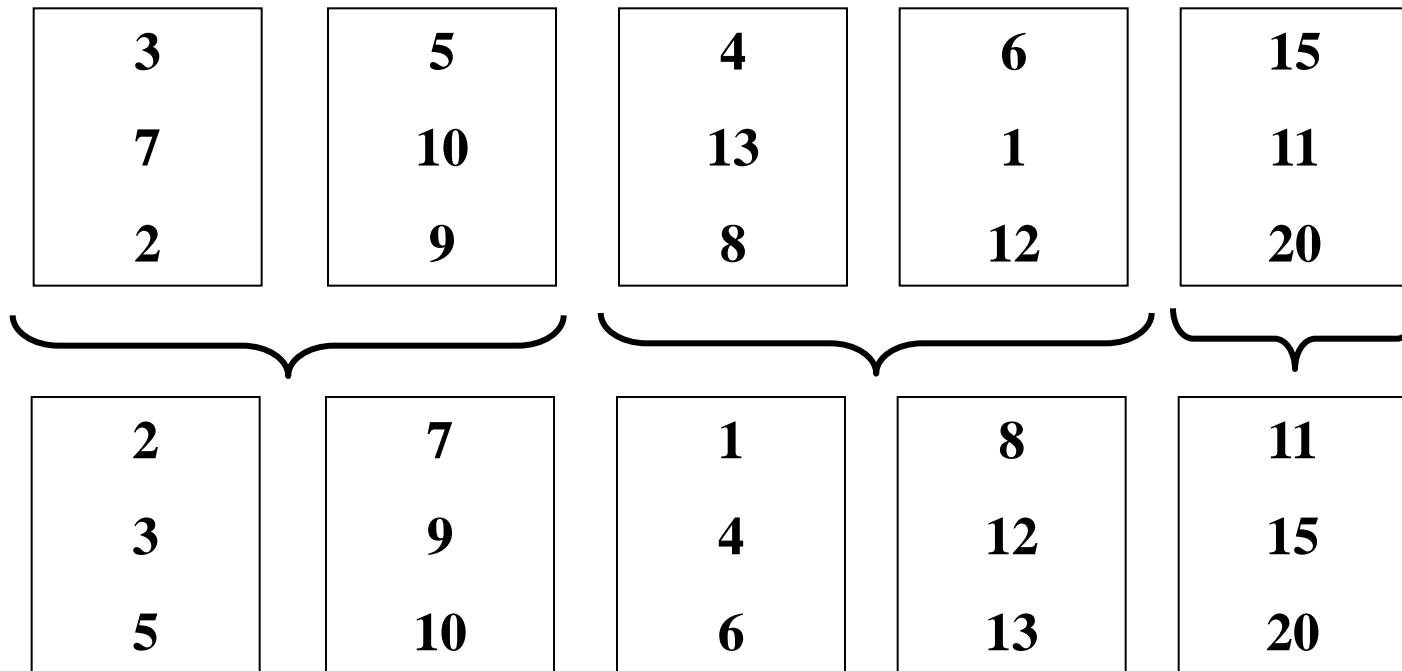
Tri externe : 1ère étape

*La mémoire
contient 3
emplacements*



disque

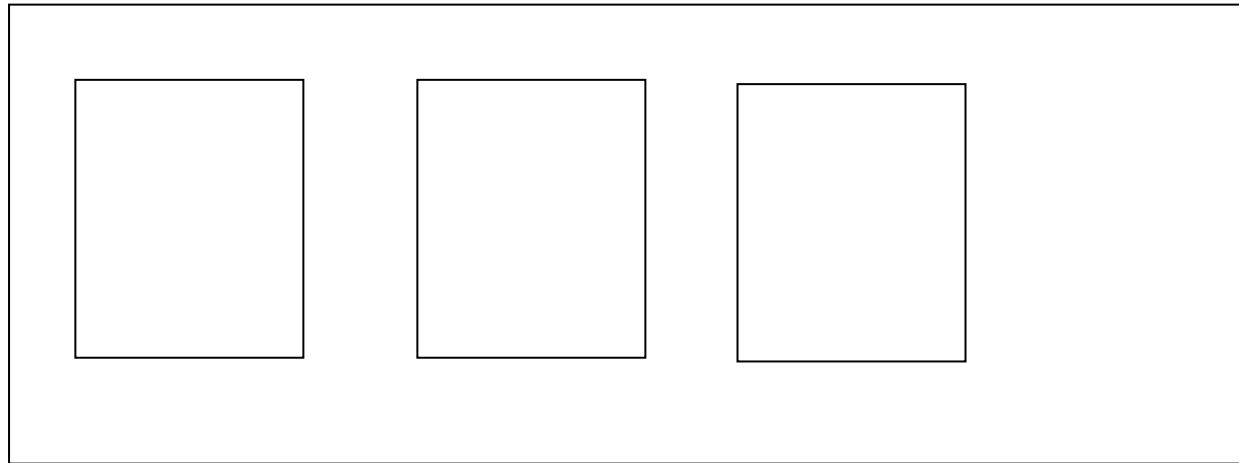
Tri externe



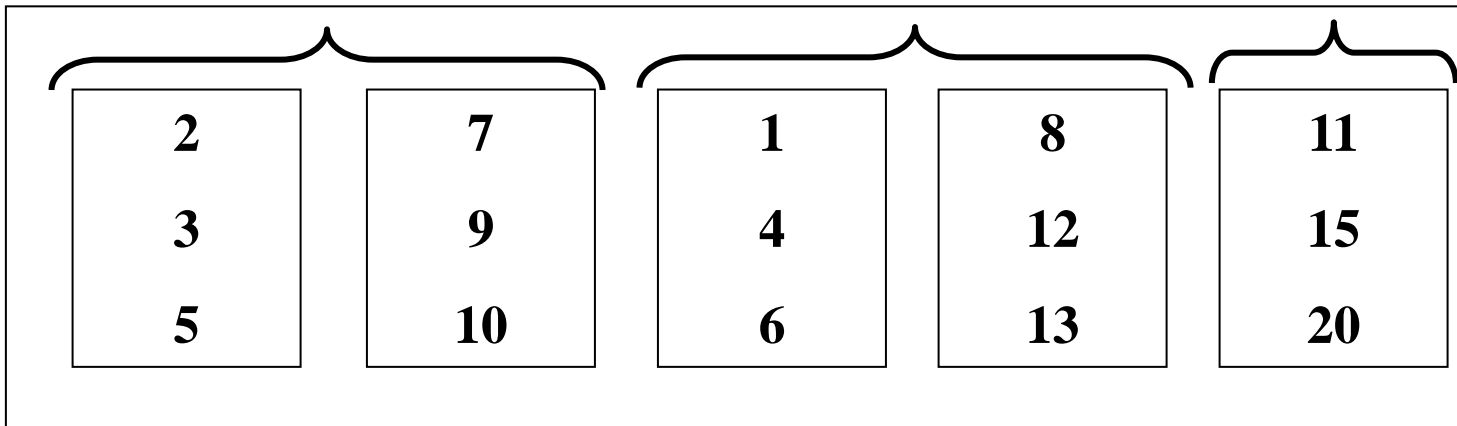
*Ensemble de
départ*

Après la 1ère étape de
tri : Les pages sont
triées par paquets de 2
 \Rightarrow 3 paquets triés de 2
pages soit 2×5 E/S

Tri externe : 2ème étape

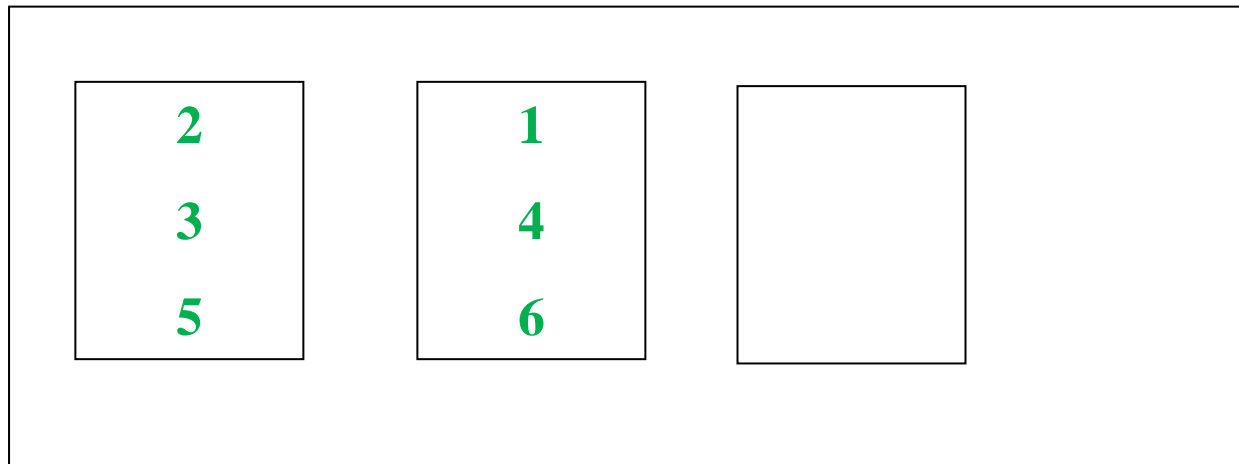


*La mémoire
contient 3
emplacements*

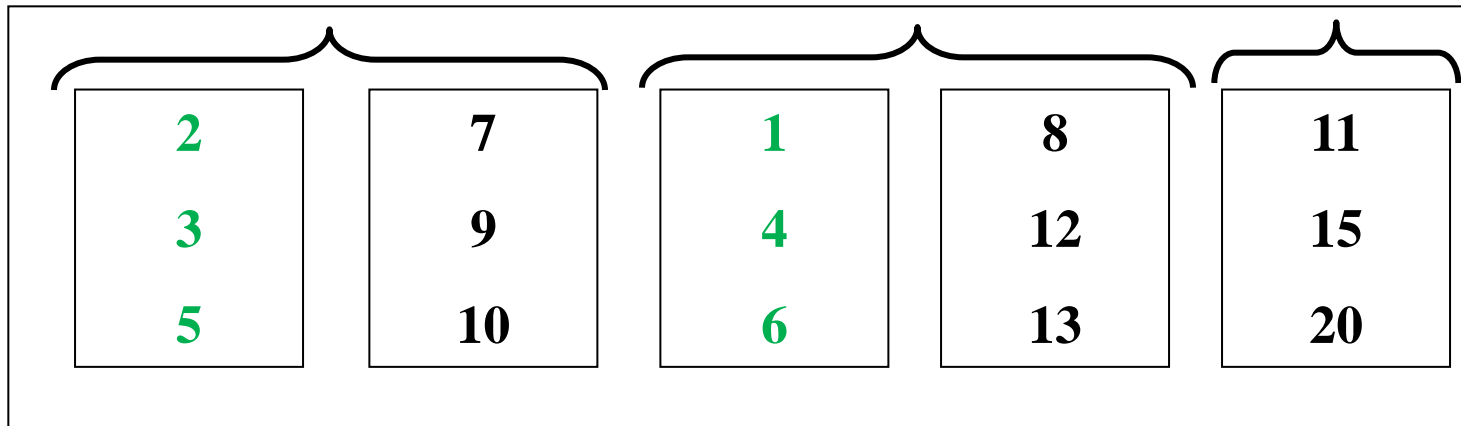


disque

Tri externe : 2ème étape

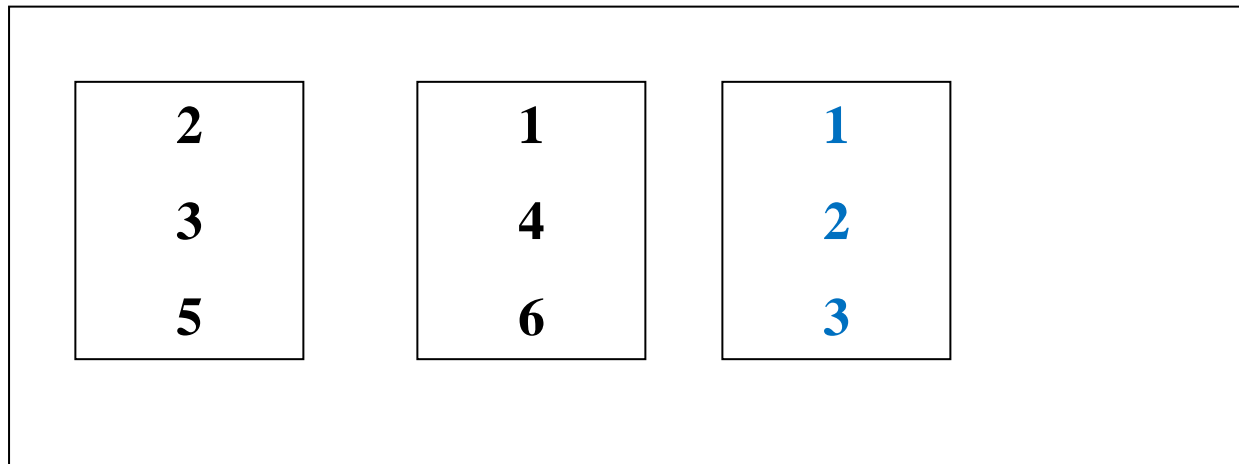


*La mémoire
contient 3
emplacements*

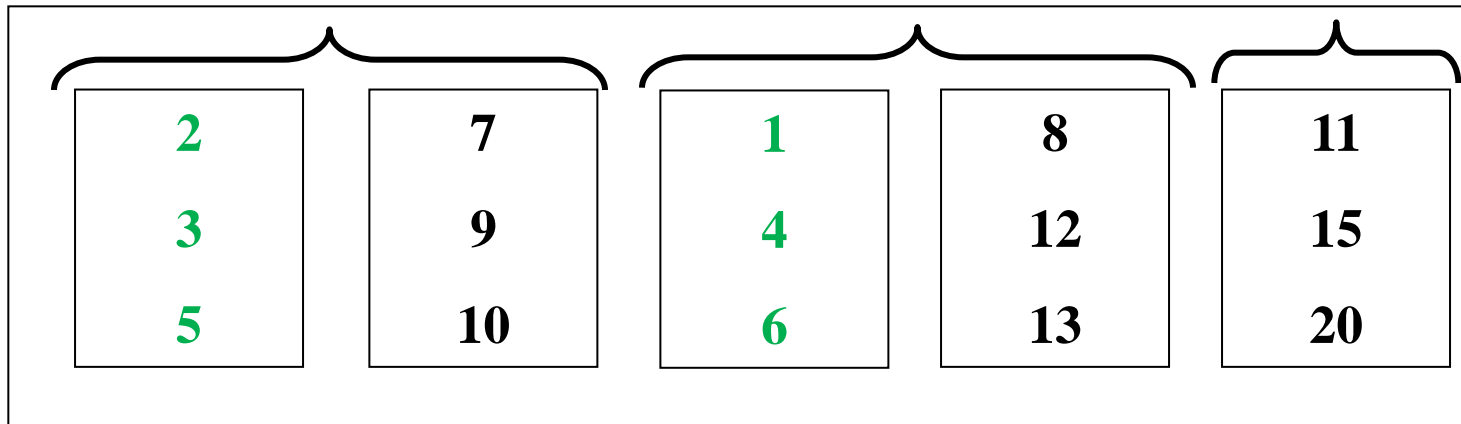


disque

Tri externe : 2ème étape

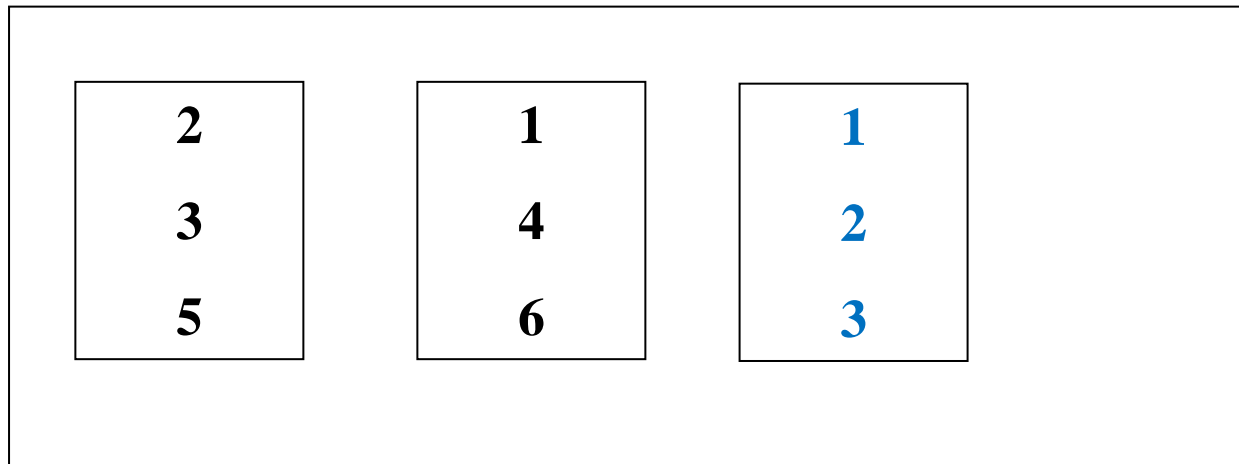


*La mémoire
contient 3
emplacements*

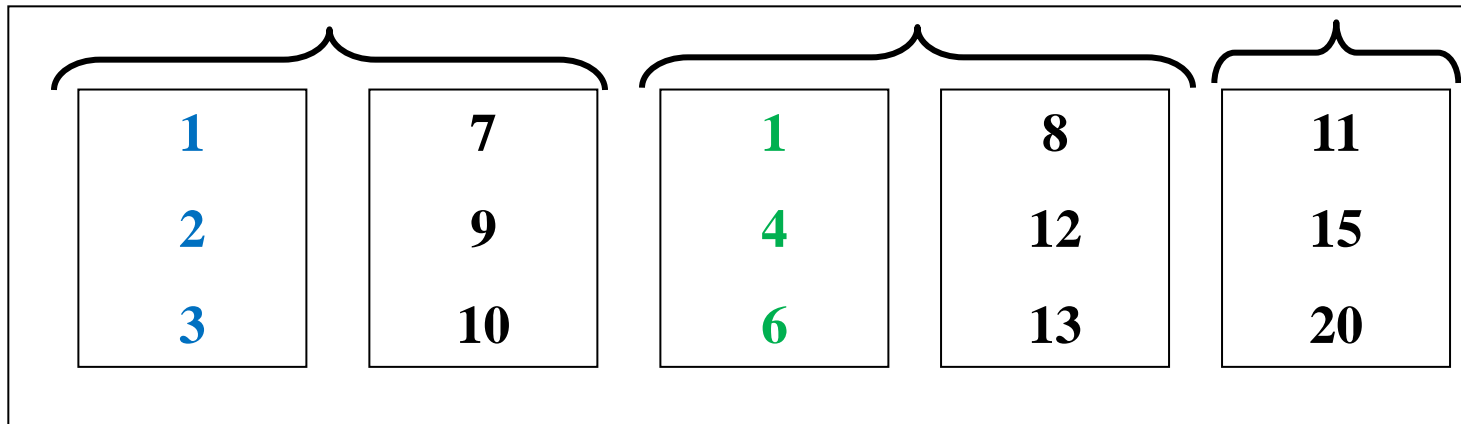


disque

Tri externe : 2ème étape

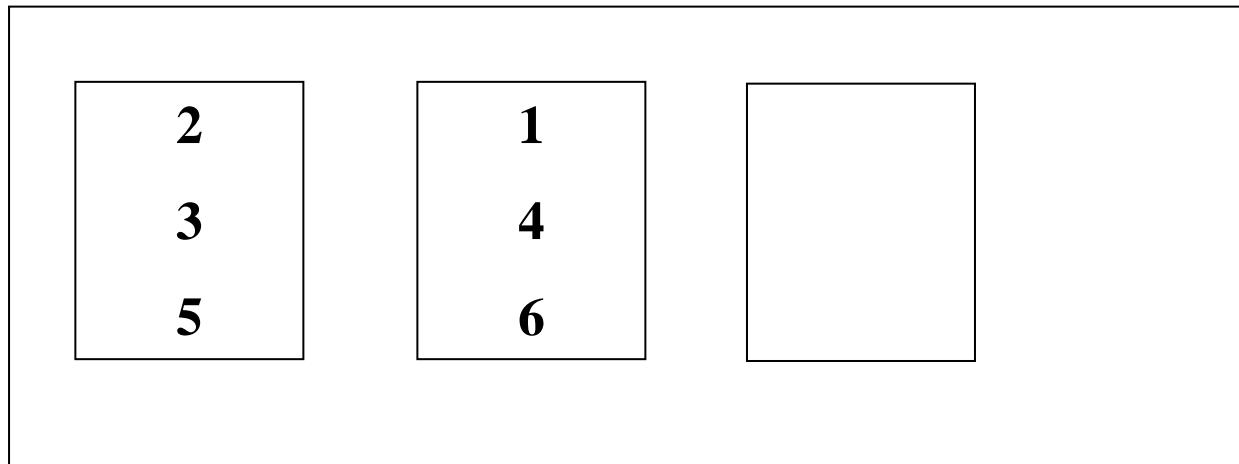


*La mémoire
contient 3
emplacements*

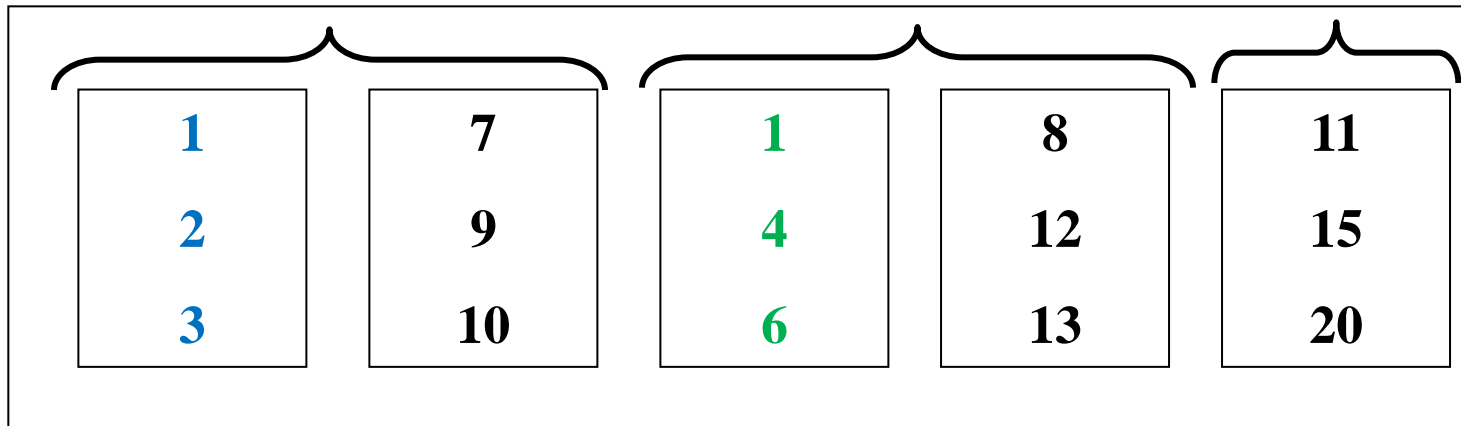


disque

Tri externe : 2ème étape

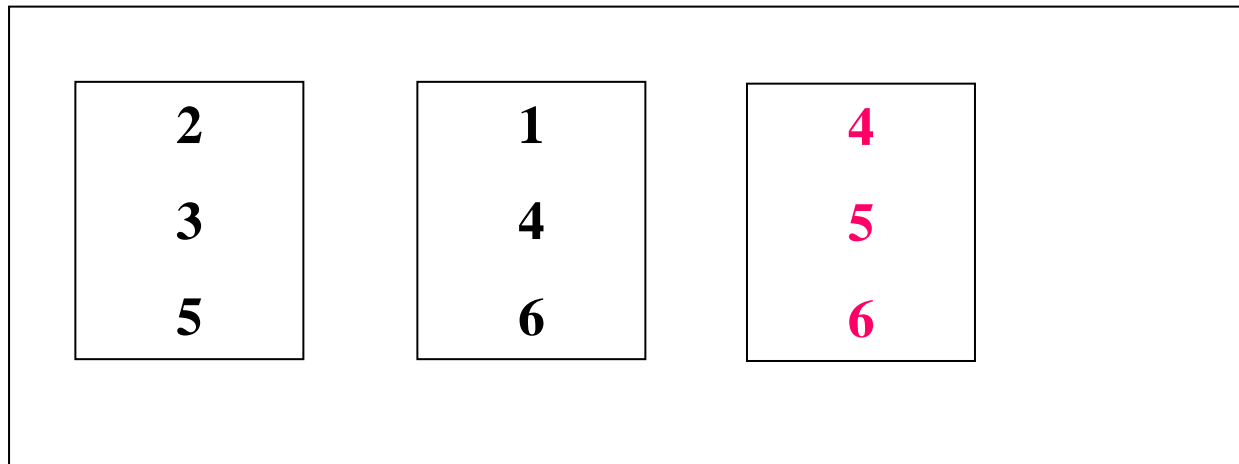


*La mémoire
contient 3
emplacements*

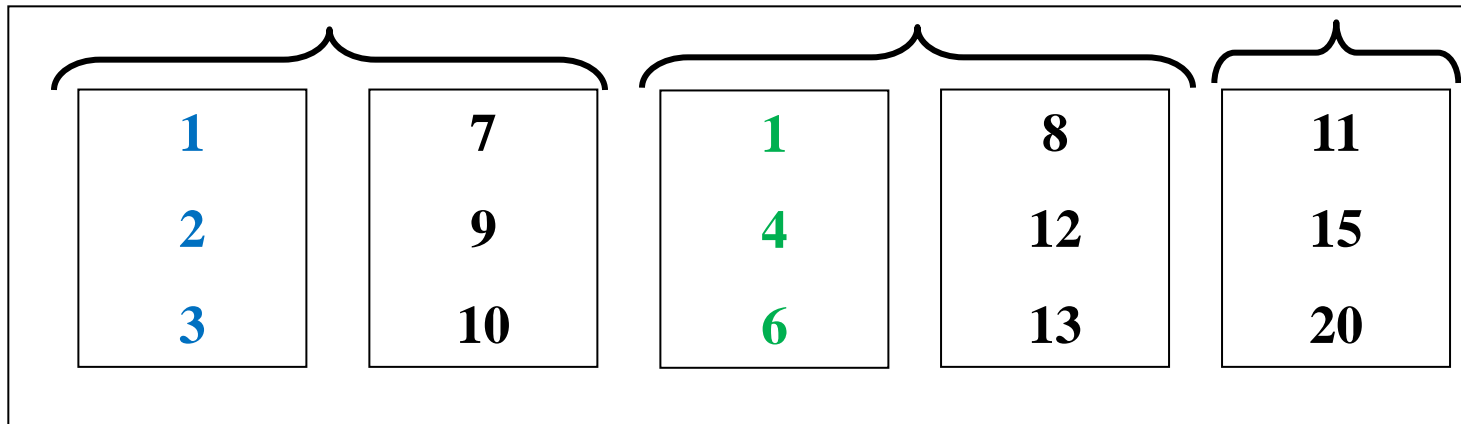


disque

Tri externe : 2ème étape

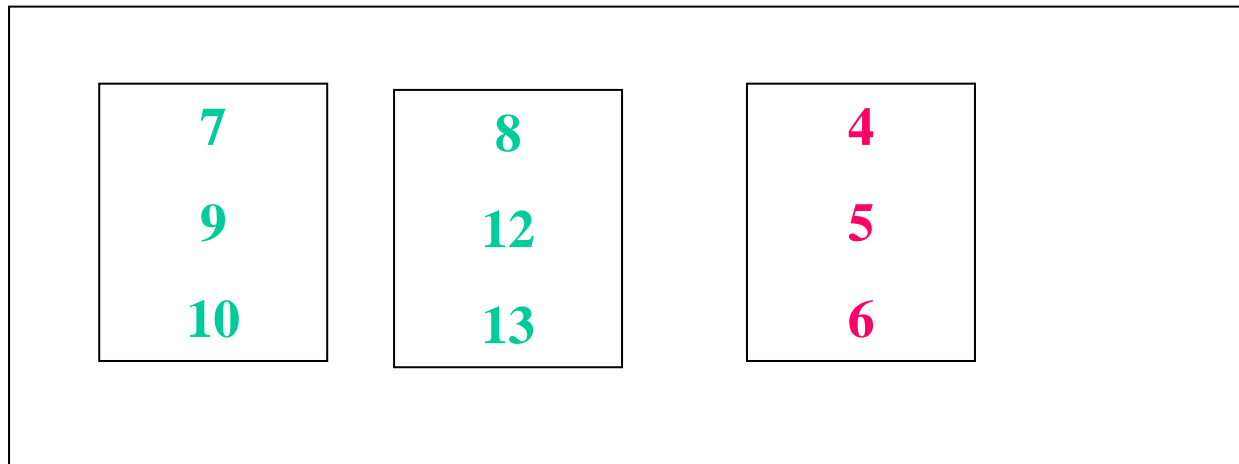


*La mémoire
contient 3
emplacements*

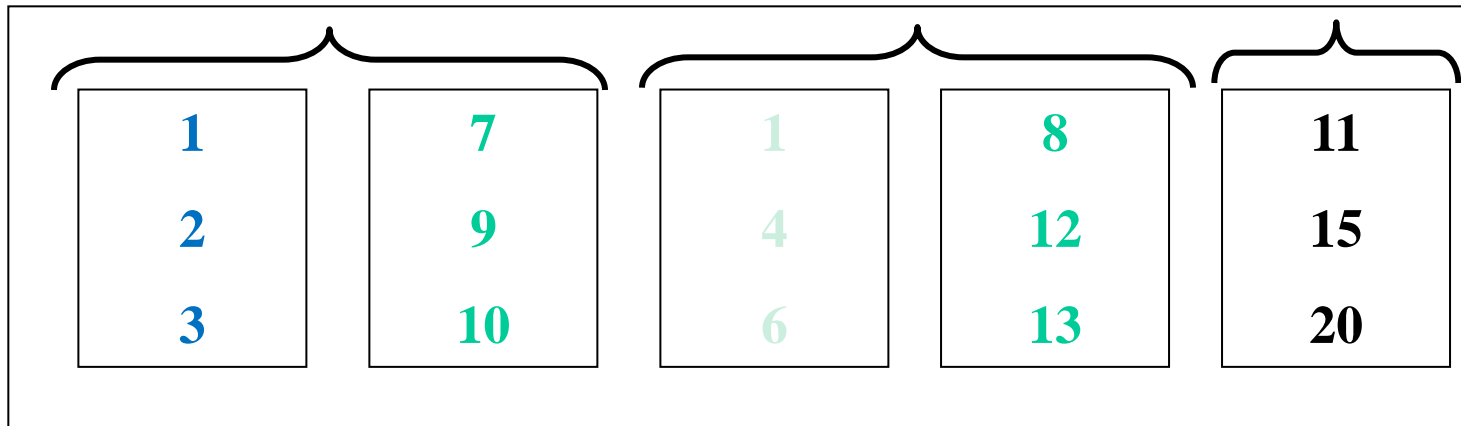


disque

Tri externe : 2ème étape

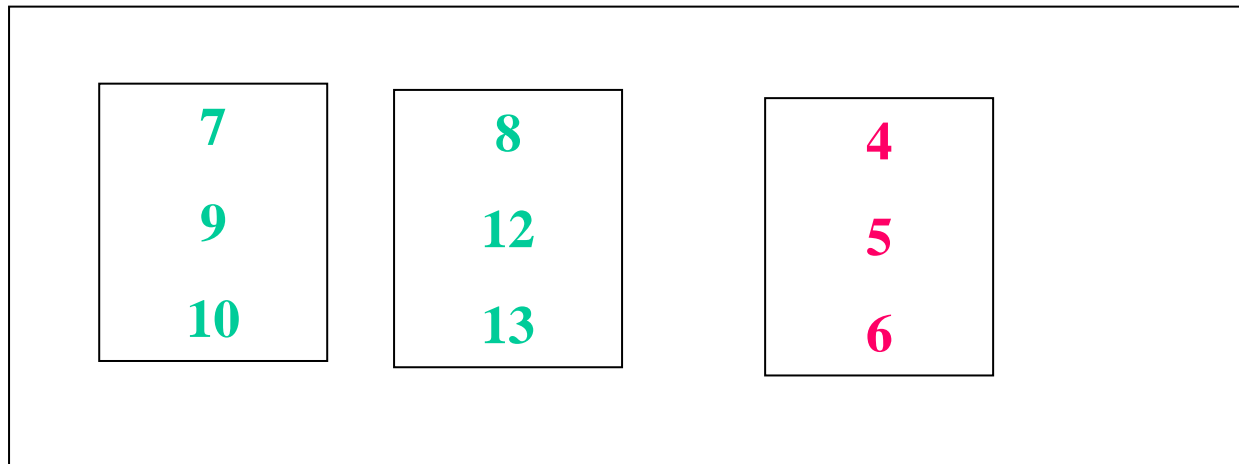


*La mémoire
contient 3
emplacements*

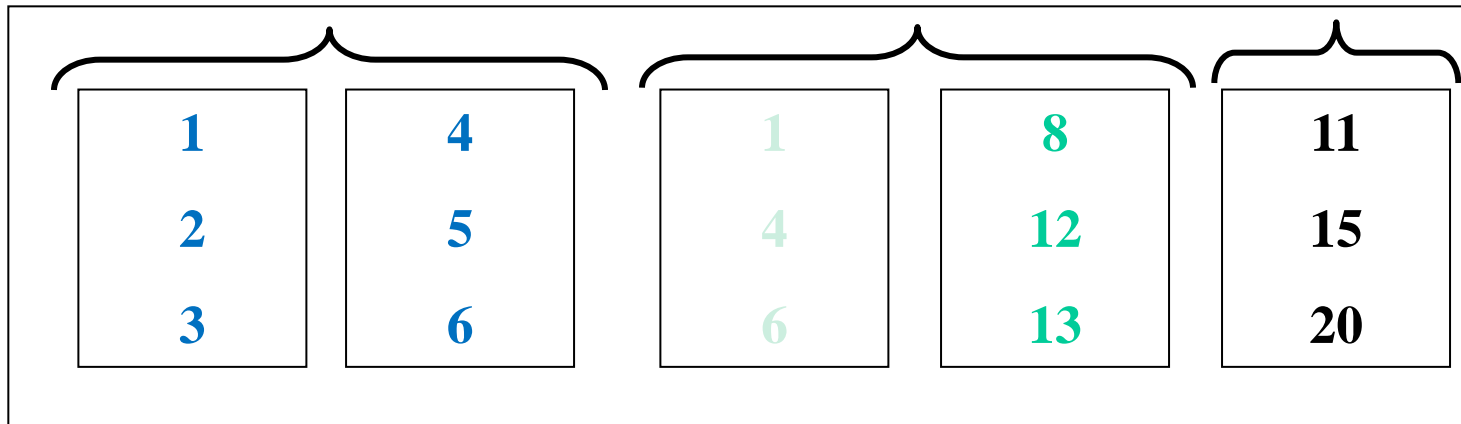


disque

Tri externe : 2ème étape

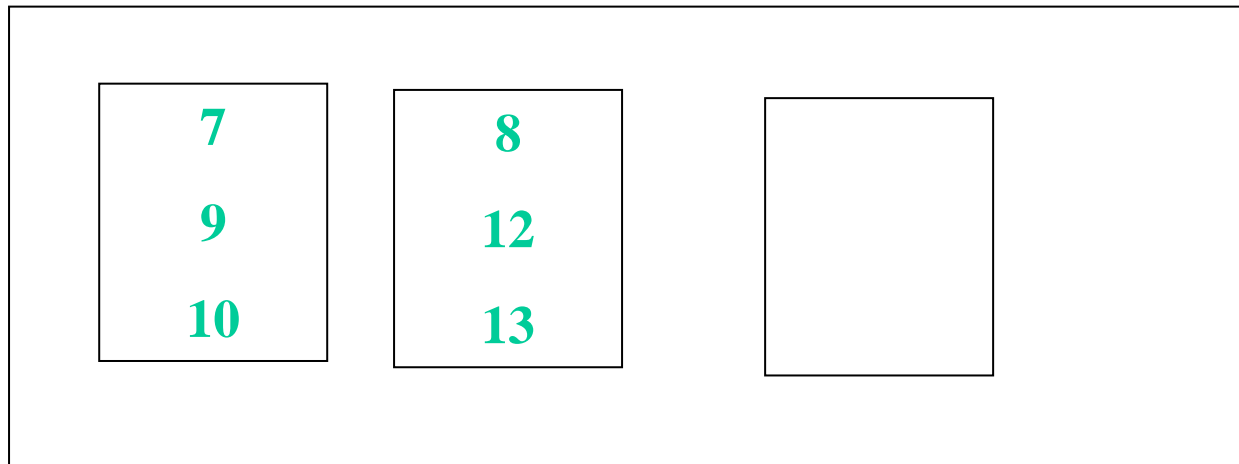


*La mémoire
contient 3
emplacements*

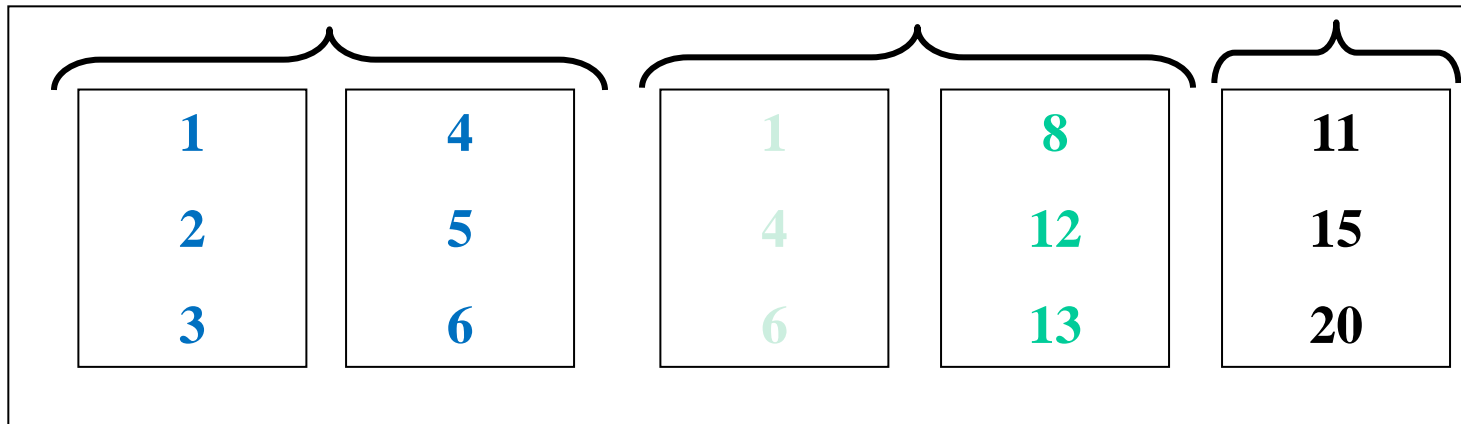


disque

Tri externe : 2ème étape

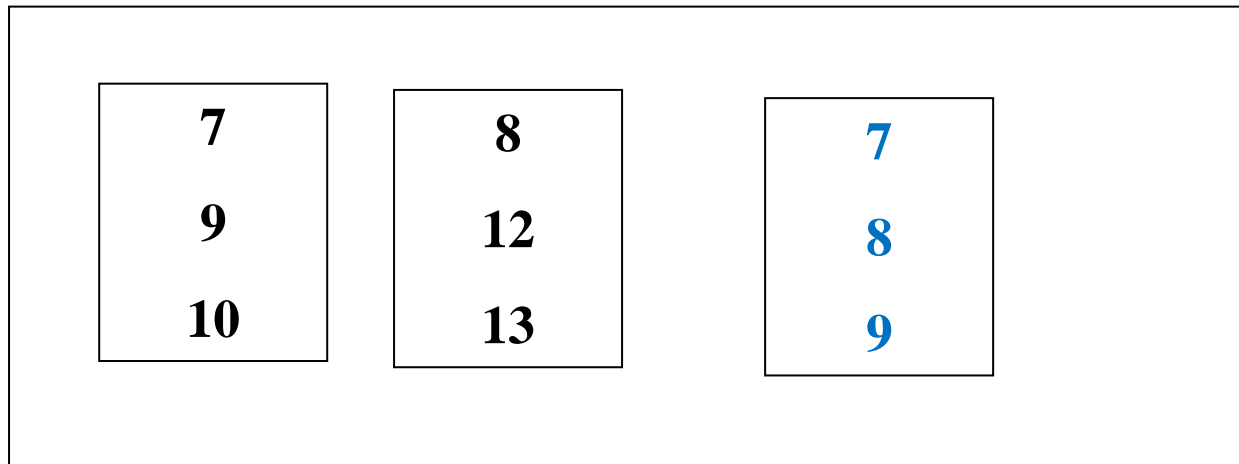


*La mémoire
contient 3
emplacements*

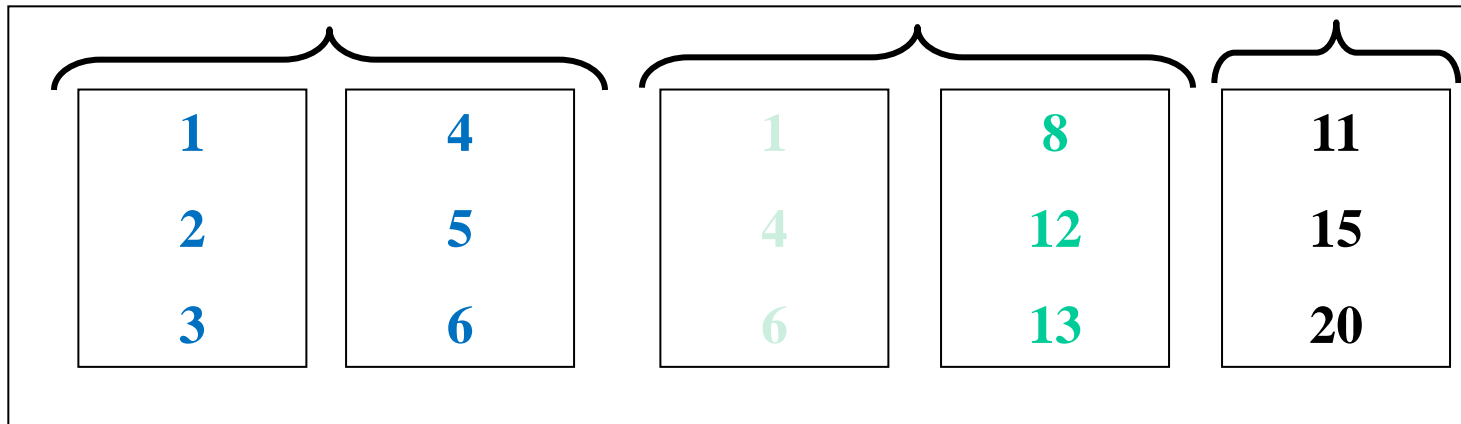


disque

Tri externe : 2ème étape

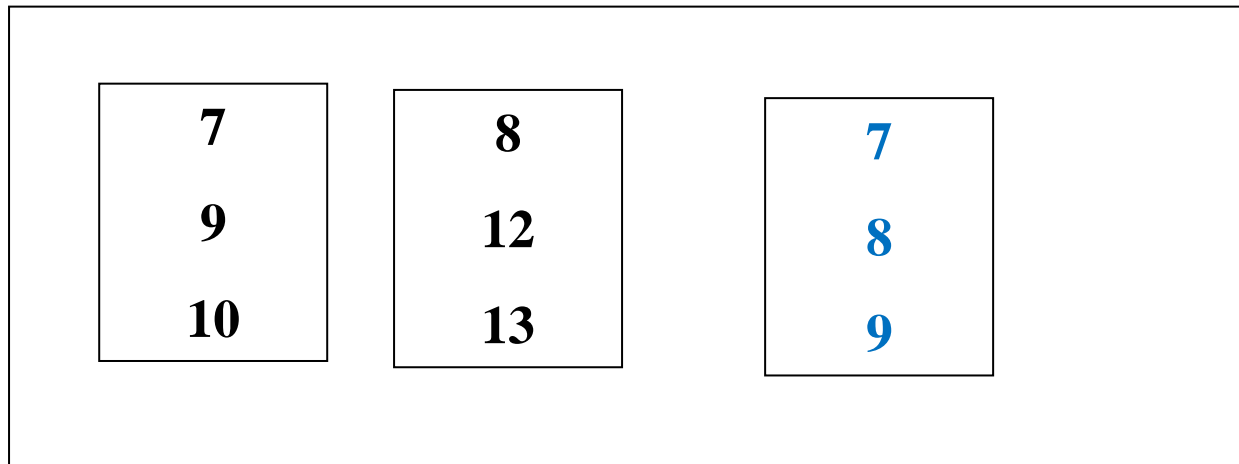


*La mémoire
contient 3
emplacements*

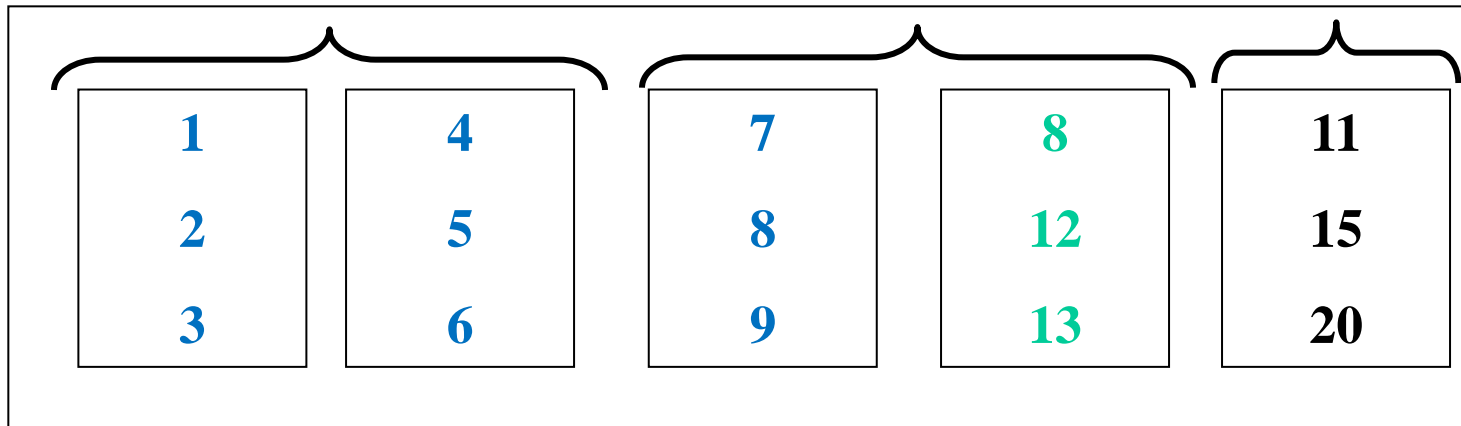


disque

Tri externe : 2ème étape

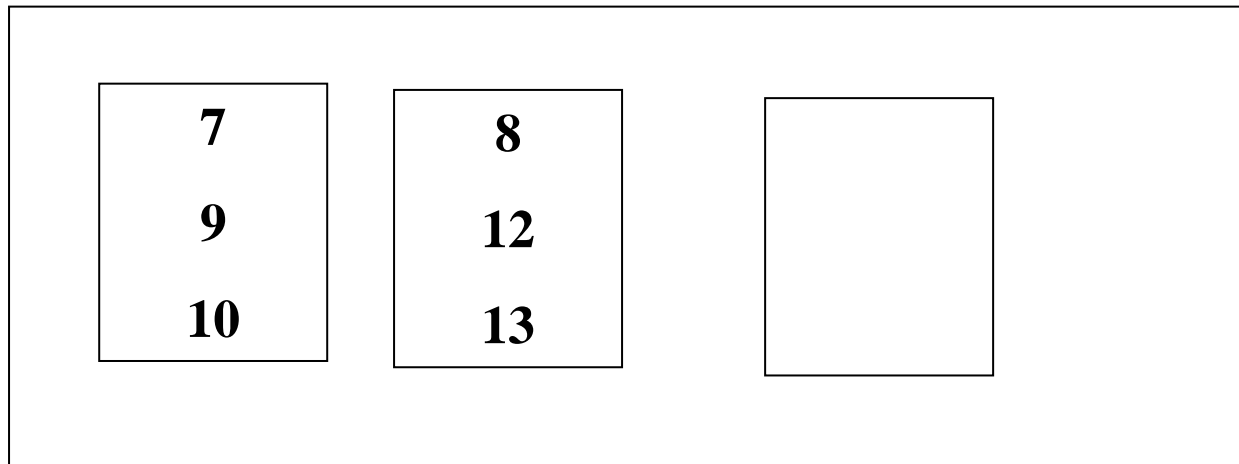


*La mémoire
contient 3
emplacements*

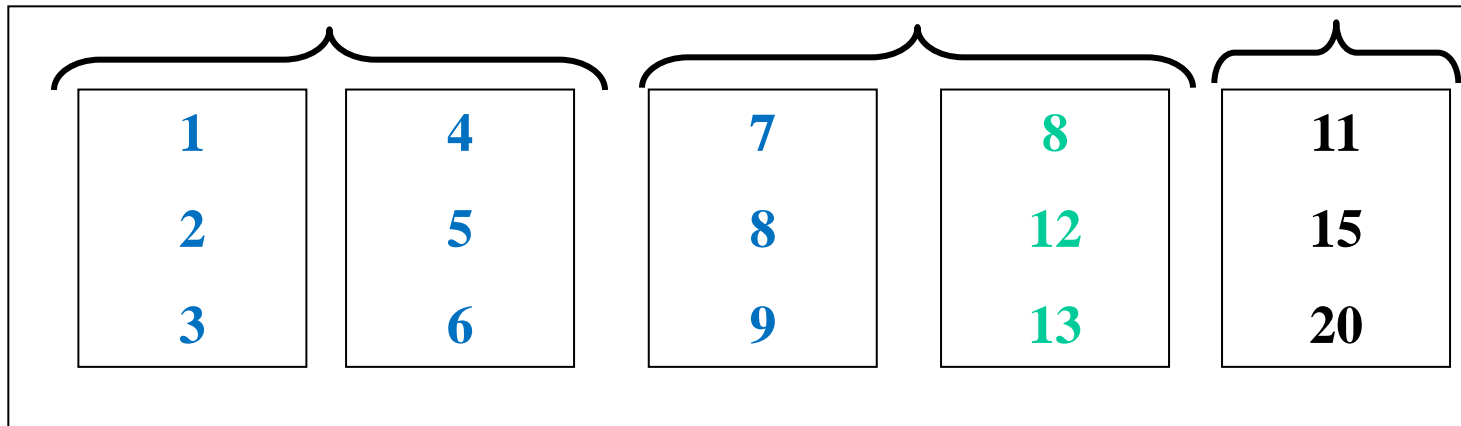


disque

Tri externe : 2ème étape

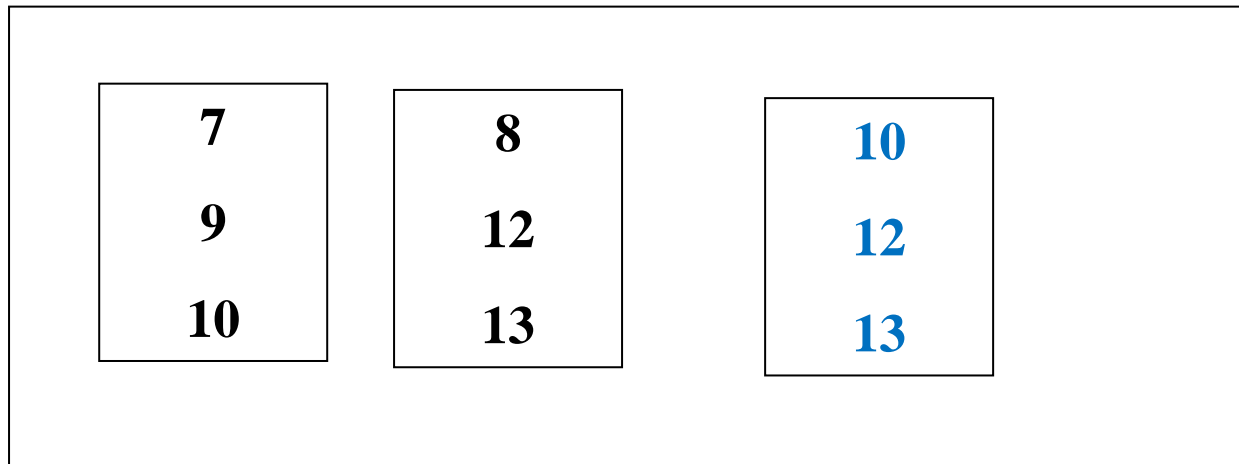


*La mémoire
contient 3
emplacements*

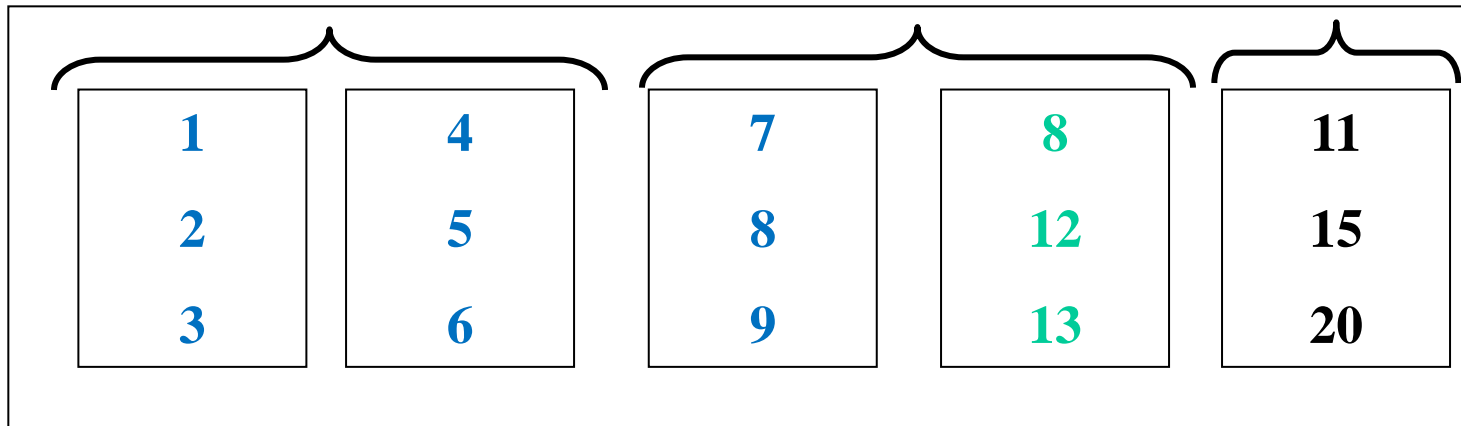


disque

Tri externe : 2ème étape

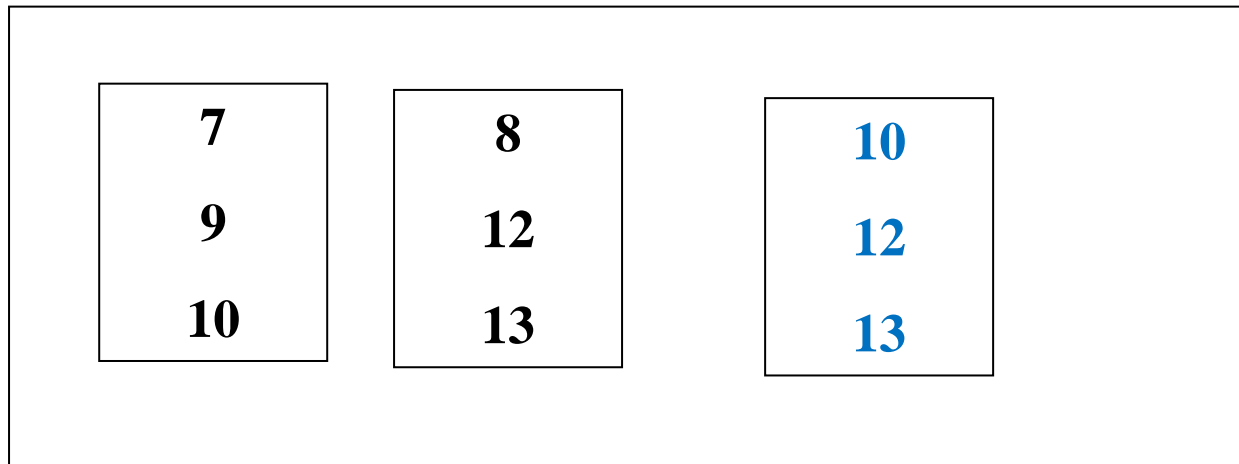


*La mémoire
contient 3
emplacements*

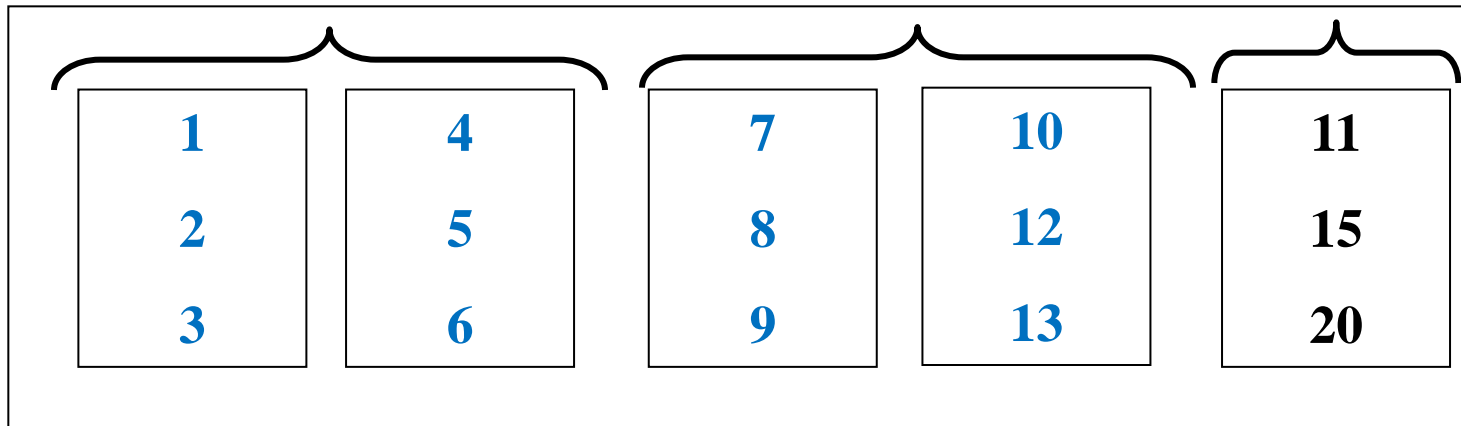


disque

Tri externe : 2ème étape

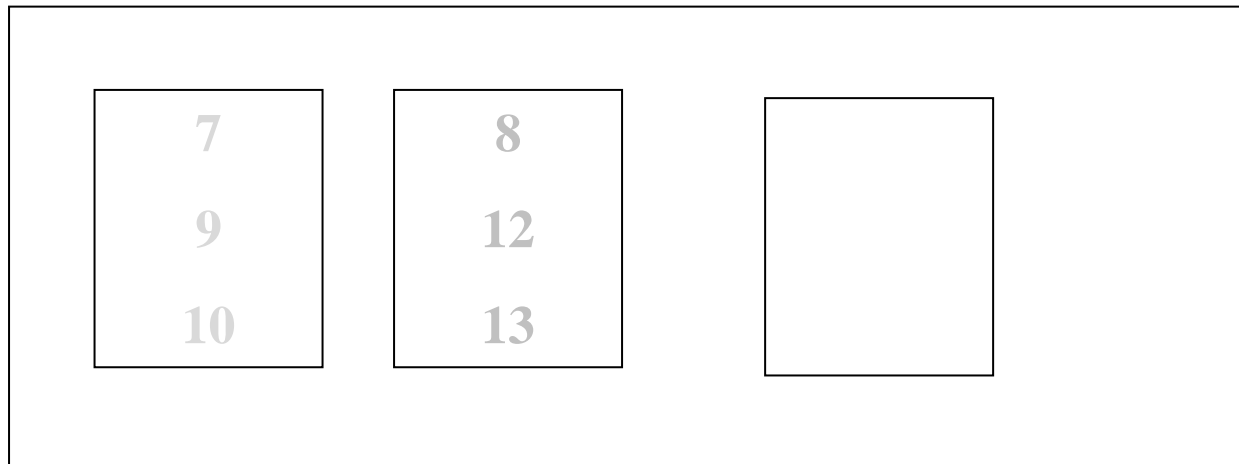


*La mémoire
contient 3
emplacements*

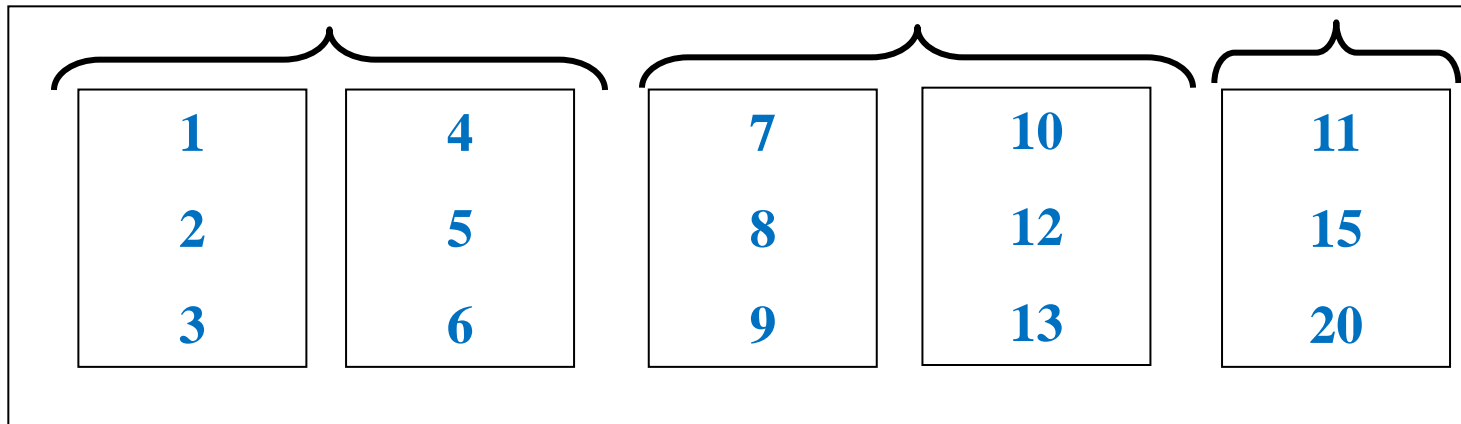


disque

Tri externe : 2ème étape



*La mémoire
contient 3
emplacements*



disque

Tri externe

3	5	4	6	15
7	10	13	1	11
2	9	8	12	20

*Ensemble de
départ*

2	7	1	8	11
3	9	4	12	15
5	10	6	13	20

Après la 1ère étape de
tri : Les pages sont
triées par paquets de 2
 \Rightarrow 3 paquets triés de 2
pages soit $2*5$ E/S

1	4	7	10	11
2	5	8	12	15
3	6	9	13	20

Après la 2ème étape
de tri : Les paquets
sont triés 2 par 2 \Rightarrow
1 paquets trié de 4
pages et 1 paquet de
1 pages soit $2*5$ E/S

Tri externe

Passe 1 : On obtient des paquets triés de deux pages

Passe 2 : on obtient des paquets triés de quatre pages.

Passe k : on obtient des paquets triés de 2^k pages

L'algorithme s'arrête lorsque $2^k \geq \text{Nbblocks}(R) \Rightarrow$

$$k \geq \log_2 (\text{Nbblocks}(R))$$

Si la mémoire peut contenir $B+1$ pages

Le coût de tri d'une relation est :

- $\text{NBLOCKS}(R) * \log B(\text{NBLOCKS}(R))$

Fusion

RID	Nom	Heure	Salaire	SID	Département	Jour
22	Daniel	7	145KF	28	R&D	02/02/2001
28	Jeanne	9	175KF	28	R&D	05/02/2001
31	Paul	12	200KF	31	Compta	03/02/2001
36	Pierre	6	120KF	31	Direction	05/02/2001
				31	Compta	06/02/2001

RID	Nom	Heure	Salaire	Département	Jour
28	Jeanne	9	175KF	R&D	02/02/2001
28	Jeanne	9	175KF	R&D	05/02/2001
31	Paul	12	200KF	Compta	03/02/2001
...

Algorithme de tri-fusion

Le coût de tri d'une relation est :

- $NBlocks(R) * \log B(NBlocks(R))$

Le coût de la fusion(après le tri) est :

$$NBlocks(R) + NBlocks(S)$$

Le coût de la jointure est

$$NBlocks(R) * \log_B(NBlocks(R)) + NBlocks(S) * \log_B(NBlocks(S)) + NBlocks(R) + NBlocks(S)$$

Jointure par hachage

Lorsque la **jointure est naturelle**, il est possible d'utiliser une fonction de hachage pour exécuter la jointure. L'idée est de créer des partitions des deux relations à partir d'une même fonction de hachage. Chaque partition i de R et de S contient des enregistrements de même valeur pour les attributs de jointure.

Si l'index de hachage peut tenir en mémoire, le coût est :

- $3 * (NBlocks(R) + (NBlocks(S)))$

Projection $\Pi_{A_1, \dots, A_n}(R)$

La projection sur une relation R comporte deux sous opérations :

- Élimination des attributs n'apparaissant pas dans la projection
- Élimination des doublons

Cardinalité de la relation résultante :

Lorsqu'il n'y a pas de doublons à éliminer : **NTuples(R)**

Si on élimine les doublons pour la projection de R sur un attribut A :

SCA(R)

Projection $\Pi_{A_1, \dots, A_n}(R)$

Élimination des doublons par tri

L'idée est de trier les enregistrements afin de supprimer les doublons :

- Lire toute la relation **R**
- Éliminer les attributs non souhaités
- Trier **R**
- Enlever les doublons

$$\text{NBlocks}(R) + \text{NBlocks}(R) * (\text{LogB}(\text{NBlocks}(R)))$$

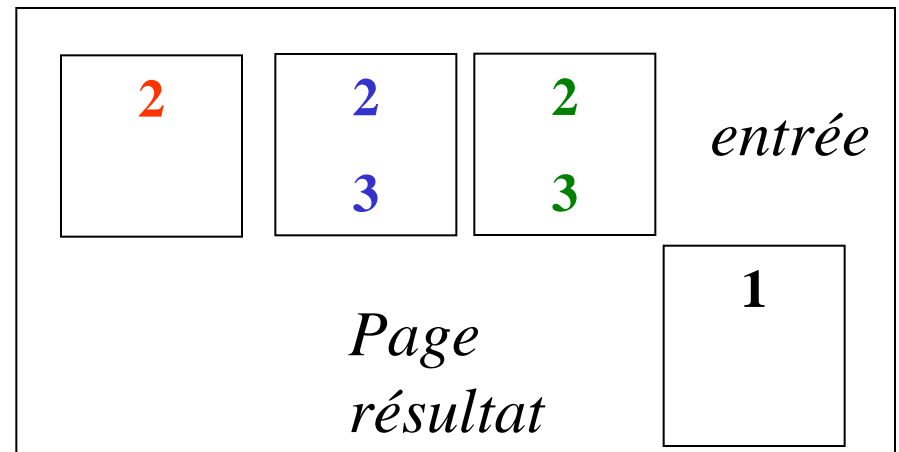
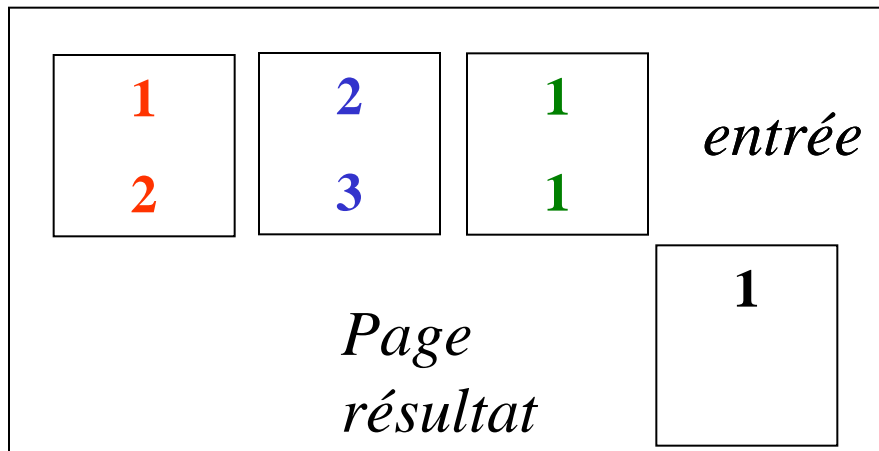
Élimination des doublons par tri

2, 5, 2, 1, 2, 2, 4, 5, 4, 3, 4, 2, 1, 5, 2, 1, 3

17 nuplets 2 nuplets / bloc
4 pages en mémoire

1, 2, 2, 2, 2, 5, 2, 3, 4, 4, 4, 5, 1, 1, 2, 3, 5

*Étape 1 : Tri des blocs par
paquets de 3 en mémoire*



*Étape 2 : la valeur 1 est la plus petite
valeur, on l'écrit dans le résultat et on
supprime les doublons*

*Étape 3 : la valeur 1 est la plus petite
valeur, on l'écrit dans le résultat et on
supprime les doublons ...*

Projection $\Pi_{A1,...,An}(R)$

Élimination des doublons par hachage, deux étapes :

- *Partitionnement* : R est lue page par page. Si la mémoire a une capacité de B pages, on utilise $B-1$ pages pour la partition.

Pour chaque enregistrement de R , les attributs n'appartenant pas à la projection sont éliminés et une **fonction de hachage** est utilisée sur les attributs de la projection pour placer l'enregistrement dans une des $B-1$ pages.

$$NBlocks(R) + T \text{ où } T < NBlocks(R)$$

T étant dépendant du nombre d'attributs de la projection

Projection $\Pi_{A_1, \dots, A_n}(R)$

- **Élimination** : Une fonction de hachage (différent de la première) partitionne les enregistrements d'une même page dans une page de hachage. Si un enregistrement a la même valeur qu'un précédent déjà haché, les deux enregistrements sont comparés et s'il y a des doublons ils sont éliminés. Le coût est donc : ***T***

Les SGBD IBM DB2 et Oracle utilisent le tri pour éliminer les doublons

SQL Server et Sybase implantent les deux algorithmes

Unions, intersections et différence de R et S

Pour toutes ces opérations il faut trier les relations, puis les lire simultanément pour créer le résultat en éliminant les doublons, le coût est :

$$[\text{NBlocks}(\mathbf{R}) * (\text{Log}_B(\text{NBlocks}(\mathbf{R}))) + [\text{NBlocks}(\mathbf{S}) * (\text{Log}_B(\text{NBlocks}(\mathbf{S}))) + \text{NBlocks}(\mathbf{R}) + \text{NBlocks}(\mathbf{S})]$$