Bases de données Distribuées

Mooc Bador
Serge Abiteboul, Benjamin Nguyen, Philippe Rigaux

informatiques mathématiques

C018SA-W6-S1

SEMAINE 6 : Bases de données distribuées

1. **Introduction**
2. Différentes architectures
3. Fragmentation
4. Optimisation de requête
5. Réplication
6. Concurrence
7. Conclusion : cinq tendances

Bases de données distribuées

- Les données d'une application sont de plus en plus souvent distribuées entre plusieurs SGBD/serveurs
- Dans cette séquence, on explique
  1. pourquoi on distribue des données, et
  2. comment on réalise une telle distribution.

Système distribué

Un système distribué est une application qui coordonne les actions de plusieurs ordinateurs pour réaliser une tâche particulière

Le calcul distribué a une composante importante de gestion de données :

- Les données de l'application
- Les échanges de données entre les ordinateurs (communications)
- Les états du système et des sessions des utilisateurs
- Les profils des utilisateurs…

Base de données distribuées

Une base de données distribuées : une grande quantité de données résidant sur plusieurs machines

Un système de gestion de données distribuées :

un logiciel qui permet d'avoir un point d'entrée unique

sur une base de données distribuées

Contradiction apparente

- Les calculs et données sont décentralisés
- Un point d'entrée unique pour accéder
  à toutes les données de manière transparente

Base de données distribuées (suite)

Fragmentation

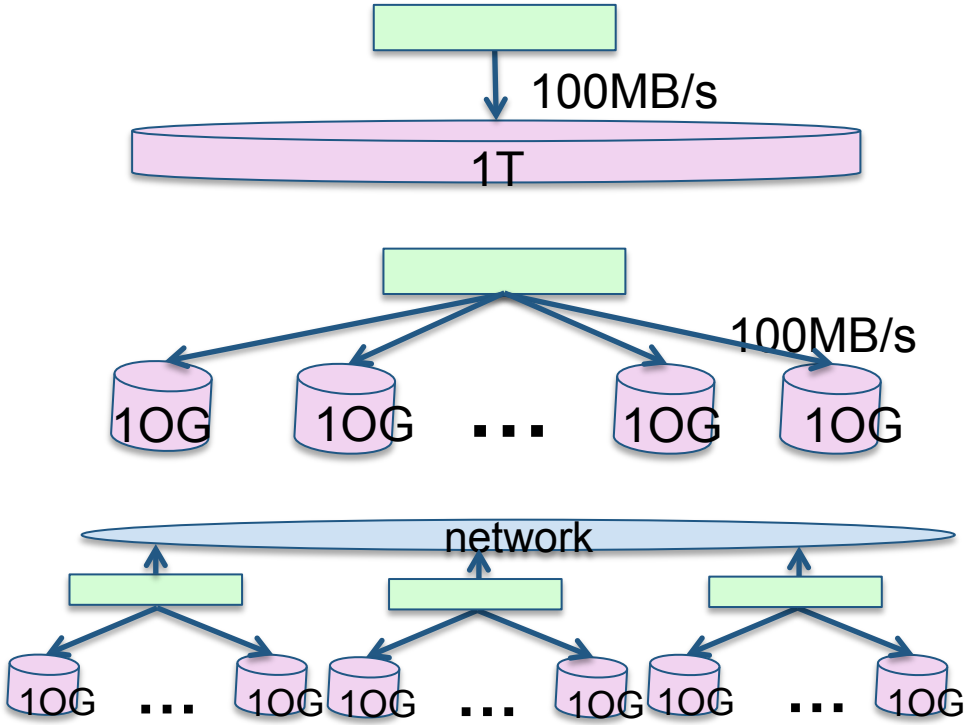
- On part d'une grosse base de données
- On la distribue en fragments pour améliorer les performances
- On obtient une base de données distribuées

Intégration

- On part de bases de données existantes (autonomes)
- On les intègre pour obtenir un point d'entrée unique
- On obtient une base de données distribuées

Réplication de données

Pourquoi fragmenter : performance



Un ordinateur et un disque

- 1 téraoctet
- Scan séquentiel
- 166 minutes (> 2.5 heures)

100 disques en parallèle

- moins de 2mn

100 ordinateurs distribués

- Chacun son propre CPU
- Chacun son disque
- **Ça passe à l'échelle**

Pourquoi intégrer : conserver l'autonomie

Autonomie des différents systèmes

Permet

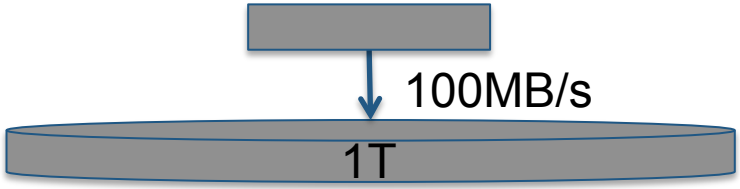
- L'intégration de données provenant d'organismes indépendants (par ex, entreprises indépendantes)
- L'intégration de données provenant de différentes branches d'un même organisme mais en laissant plus de liberté à chaque branche

Tendance dans l'industrie : des structures de moins en moins hiérarchiques, de plus en plus d'autonomes

Pourquoi répliquer : sureté

Mono-serveur

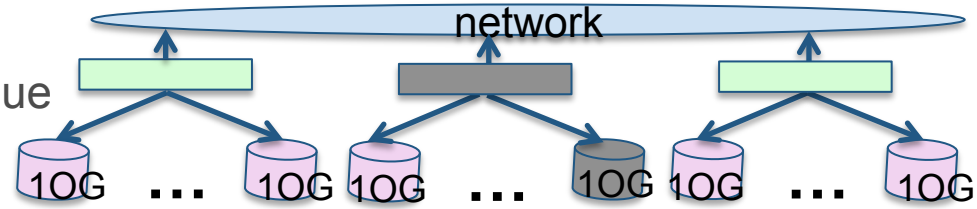
- Panne du processeur ou du disque
- Arrêt du service
- Perte des données



Multiserveurs

- Panne d'un processeur ou d'un disque
- Le service continue
- Grâce à la réplication,

  pas de perte des données



Enormément utilisé

- Réplication par 2 : banque
- Réplication par 3 : airbus

Concept essentiel : transparence

de localisation
- l'utilisateur n'a pas savoir où sont les données

de réseau/système
- il n'a pas à connaître les aspects techniques du réseau

de fragmentation
- il n'a pas à connaître comment les données sont fragmentées

de réplication
- il n'a pas à savoir comment/si elles sont répliquées

Chaque entité a un nom unique

Pas seulement des avantages

Complexité

- Rajoute des couches de logiciel
- Communication
- Distribution du contrôle
- Conception de la base de données
- Gestion des contraintes d'intégrité

Absence de standards

Moins d'expérience

Incohérences

- Que fait-on quand deux systèmes donnent des informations contradictoires
- Exemple : réservation de place
- Exemple : collision d'avion

C018SA-W6-S2

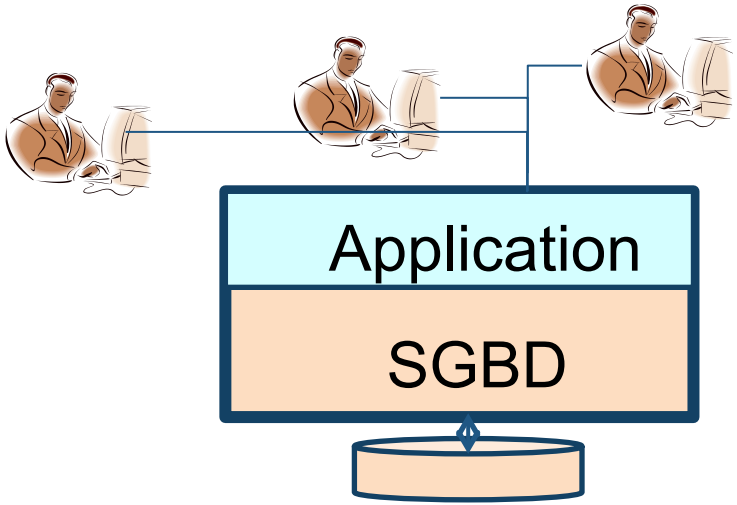
SEMAINE 6 : Bases de données distribuées

1. Introduction
2. **Différentes architectures**
3. Fragmentation
4. Optimisation de requête
5. Réplication
6. Concurrence
7. Conclusion : cinq tendances

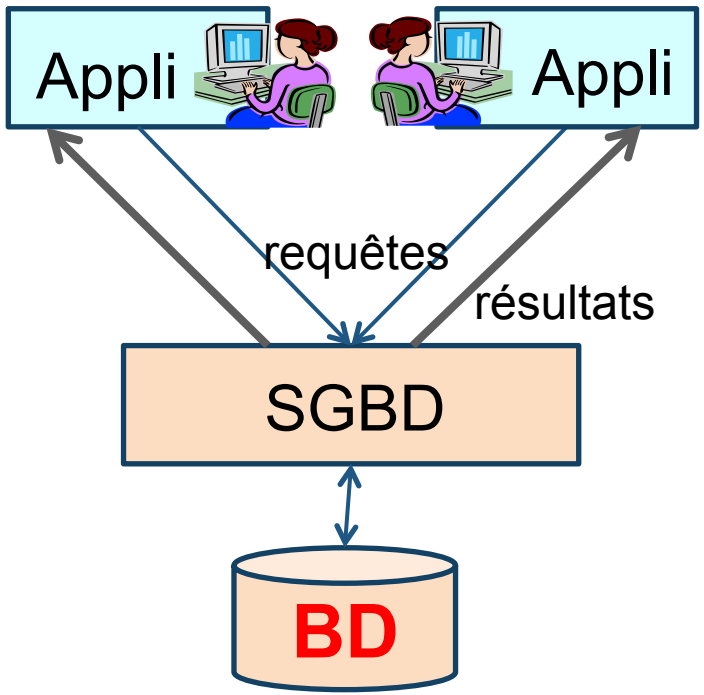
Mono-machine : pas de distribution

Les SGBD des débuts : **centralisés**

- Pluri-utilisateurs
- Terminaux



Architecture Client-Serveur



Client

- Application
- Interface graphique

Serveur

- Gestion des requêtes
- Gestion des transactions
- Gestion des pannes…

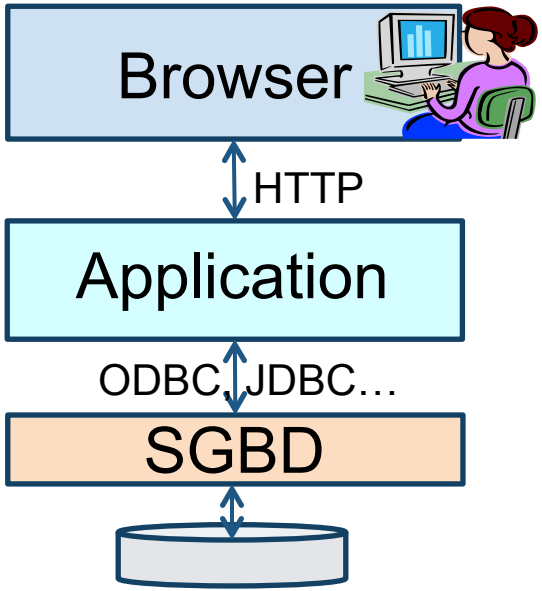
Architecture 3 niveaux

3-tier en anglais

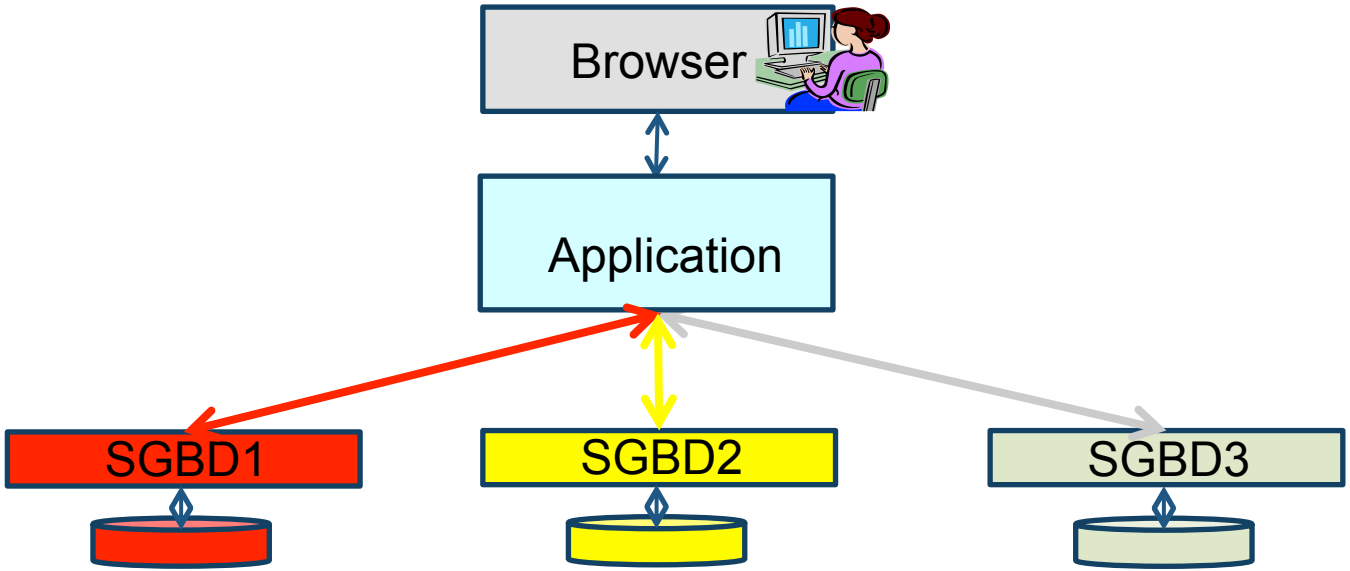
Le niveau client : un navigateur Web
- Présente du contenu, par ex HTML

Le niveau intermédiaire
- Communique avec le SGBD
- Gère l'application (Java, C++, C# …)
- Génère le contenu pour le client

Le niveau serveur : base de données



Vers la distribution : plusieurs serveurs de données



Typologie : hétérogénéité des SGBD locaux

- Modèles différents : relationnel, XML, objet…
- Organisations différentes : schémas, ontologies…
- Langages de requêtes différents : SQL, Xquery…
- Operating systems différents : Linux, Windows
- SGBD différents : Oracle, mySQL, DB2…

L'hétérogénéité introduit de la complexité

Typologie suivant le degré d'autonomie

SGBD distribuées : intégration forte

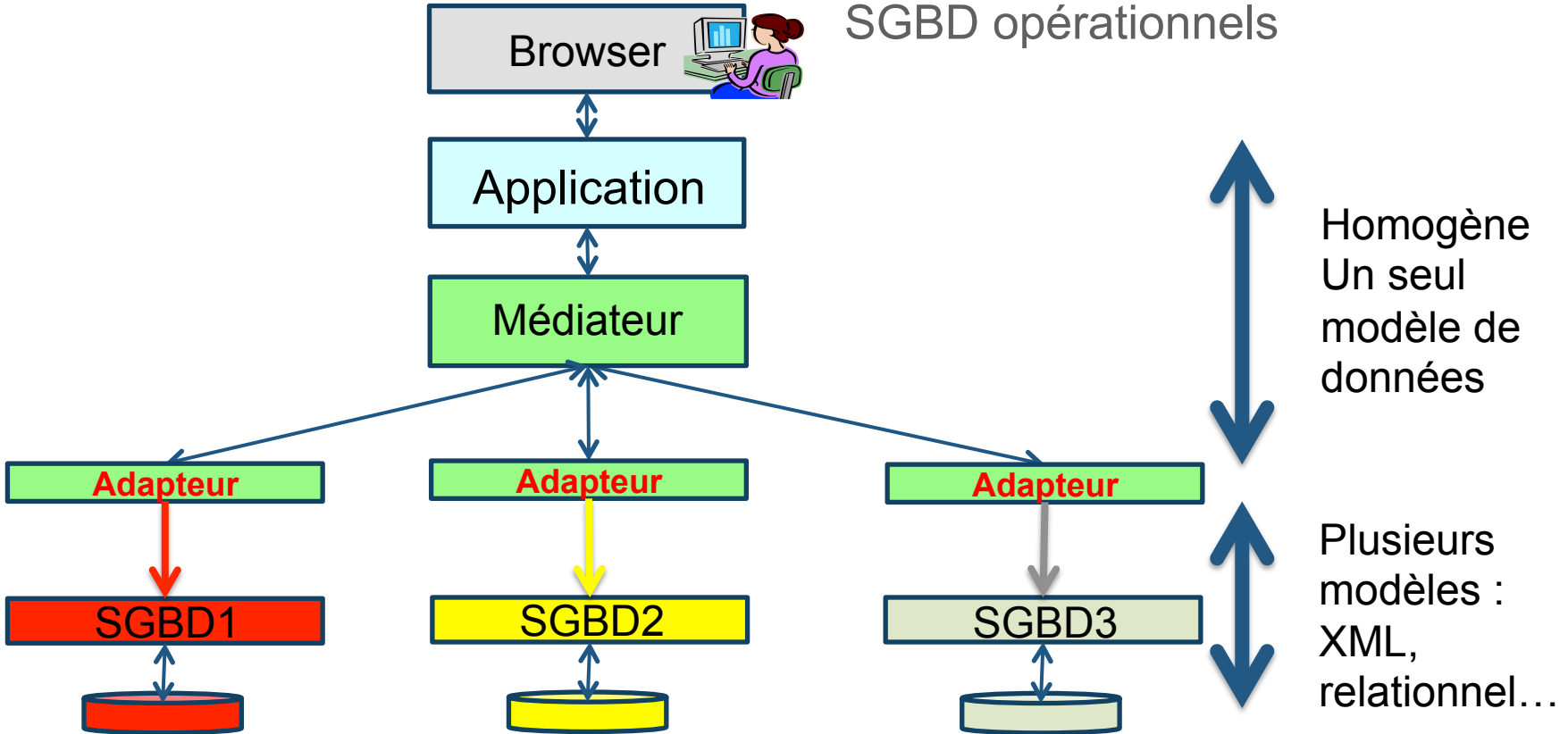
- Exemple : Oracle
- Local Oracle servers avec réseau Net8 logiciel et distributed SQL

Fédération de systèmes : autonomie forte

- Très hétérogène
- Modèles, organisations, langages, systèmes…

Une approche : La médiation

- Permet d'accéder simplement à des données hétérogènes
- Sans perturber le fonctionnement des SGBD opérationnels



SEMAINE 6 : Bases de données distribuées

1. Introduction
2. Différentes architectures
3. **Fragmentation**
4. Optimisation de requête
5. Réplication
6. Concurrence
7. Conclusion : cinq tendances

Exemple

Base de données

- Relation Employé E(enum, nom, site, salaire,...)
- Un employé n'appartient qu'à un seul site
- 40 000 employés à Paris et Lyon, 20 000 à Marseille

Charge de travail (workload)

- 90% des requêtes à Paris/Lyon/Marseille
  sont sur des employés locaux

Pour illustrer

- Accès disque = 1 unité & Accès réseau = 10 unités

Fragmentation : performance

Sans fragmentation (un SGBD à Paris)

Coût

- à Paris = 1 unité
- à L/M = 10
- moyen = 0.4+0.6*10= 6.4 unités

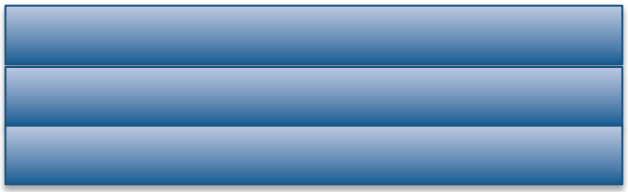
Avec fragmentation (3 SGBD : BD-P, BD-L, BD-M)

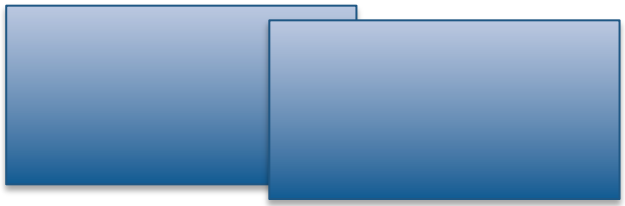
Coût

- à P/L/M = 0.9+0.1*10= 1.9
- moyen = 1.9 unités
- Evite le risque de saturer le SGBD de Paris

Fragmentation

Horizontale	Verticale
$\sigma_{site=Paris}(E)$ et $\sigma_{site\neq Paris}(E)$	$\Pi_{enum,nom}(E)$ et $\Pi_{enum,adresse\ldots}(E)$
$\sigma_{site=Paris}(E) \cup \sigma_{site\neq Paris}(E)$	$\Pi_{enum,nom}(E) \bowtie \Pi_{enum,adresse\ldots}(E)$





Fragmentation

Obligation : sans perte d'information

- $\sigma_{site=Paris}(E) \cup \sigma_{site\neq Paris}(E) = E$
- $\Pi_{enum,nom}(E) \bowtie \Pi_{enum,adresse\ldots}(E) = E$
  - Sans perte d'information si enum est une clé

Préférable : sans redondance quand c'est possible

- $\sigma_{salaire>10K}(E)$ et $\sigma_{salaire<15K}(E)$
  - Les deux fragments ne sont pas disjoints

Et des fragmentations plus complexes ?

Rappel : semi-jointure

Pour
- R sur U
- S sur V

$R \ltimes S$ est une relation sur U

$$R \ltimes S = \Pi_U(R \bowtie S)$$

Exemple de fragmentation plus complexe

Employé E(enum, nom, site, salaire, ...)

Projet P(enum, projet)

- enum clé d'Employé et
- clé étrangère de projet

Question fréquente :

donner toutes les informations de X

ainsi que ses projets

Exemple de fragmentation plus complexe (2)

Employé E(enum, nom, site, salaire, ...)

Projet P(enum, nomprojet, budget, …)

$E1@site_1 = \sigma_{site=Paris}(E)$ $P1@site_2 = P \ltimes E1$

$E2@site_3 = \sigma_{site\neq Paris}(E)$ $P2@site_4 = P \ltimes E2$

Le nuplet d'un projet qui a des employés à Paris et sur un autre site est répliqué

Fragmentation & SGBD

Les deux grandes classes de fragmentation

(horizontale et verticale)

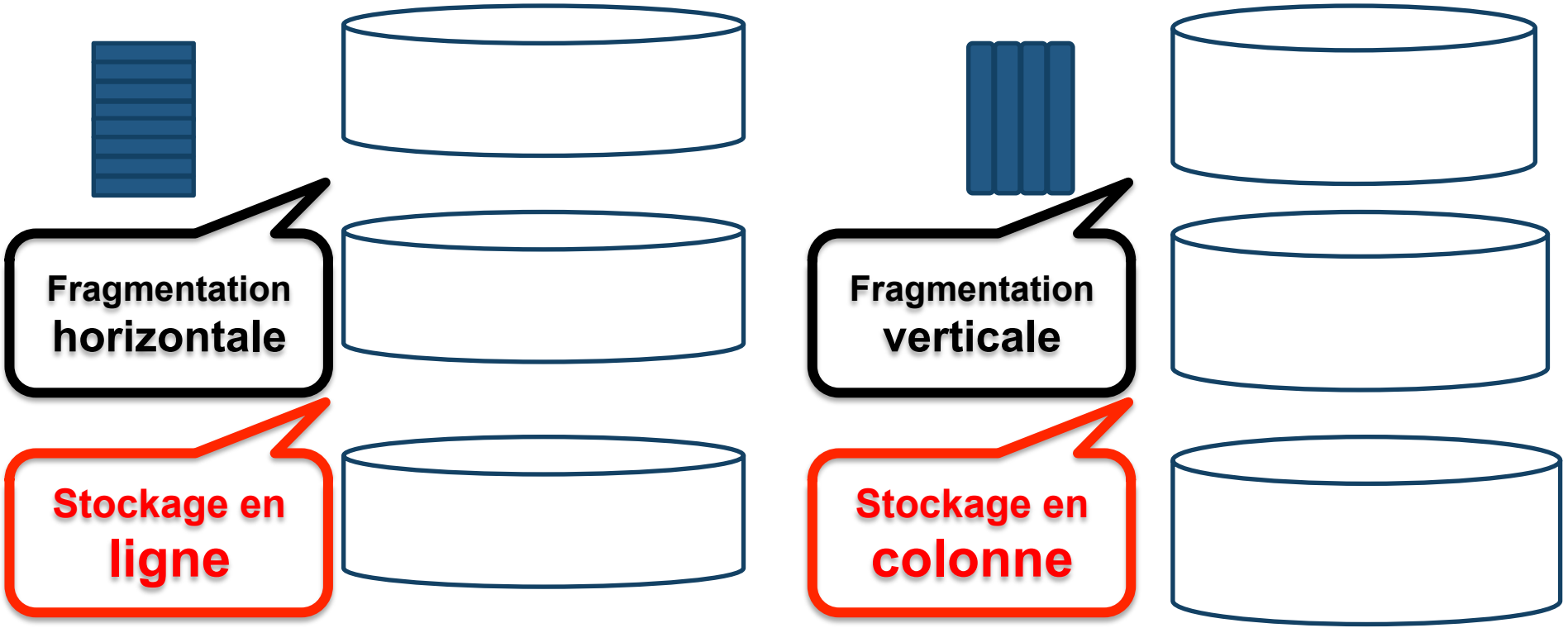
correspondent à deux grandes classes de SGBD relationnels

- Les SGBD à stockage par ligne
  - Ex d'unité de stockage

    [nom : Martin, âge : 45, adresse : Paris]
- Les SGBD à stockage par colonne
  - Ex d'unité de stockage

    {nom : Durand, nom : Doe, nom : Martin, … }

Parallélisme massif et fragmentation



Lignes ou colonnes…

Bases de données	Stockage ligne	Stockage colonne
	Oracle Database, MySQL, IBM DB2, MS SQL Server…	MonetDB, Google BigTable, SAP Hana…
	Plus grande partie du marché des BD	Grandit
Lire/écrire un nuplet	rapide	
Lire/écrire un attribut		rapide
Comprimer		excellent
Typologie d'applications	OLTP (transactionnel)	OLAP (décisionnel)

C018SA-W6-S4

SEMAINE 6 : Bases de données distribuées

1. Introduction
2. Différentes architectures
3. Fragmentation
4. **Optimisation de requête**
5. Réplication
6. Concurrence
7. Conclusion : cinq tendances

Que faut-il optimiser ?

Bases de données classiques

- I/O : Accès disque (millisec)

  >> Opération du processeur (microsec)

Bases de données distribuées

- Communication (jusqu'à 1sec sur Internet)

  >> Accès disque (millisec)

Attention : basé sur des communications lentes

- par ex, kilo-octets par second
- LAN : bande passante

  même ordre de grandeur qu'un disque

Quel est le problème ? Exemple

E1@site1 = $\sigma_{site=Paris}(E)$ P1@site2 = P ⋉ E1
E2@site3 = $\sigma_{site\neq Paris}(E)$ P2@site4 = P ⋉ E2

- Une requête arrive à site5

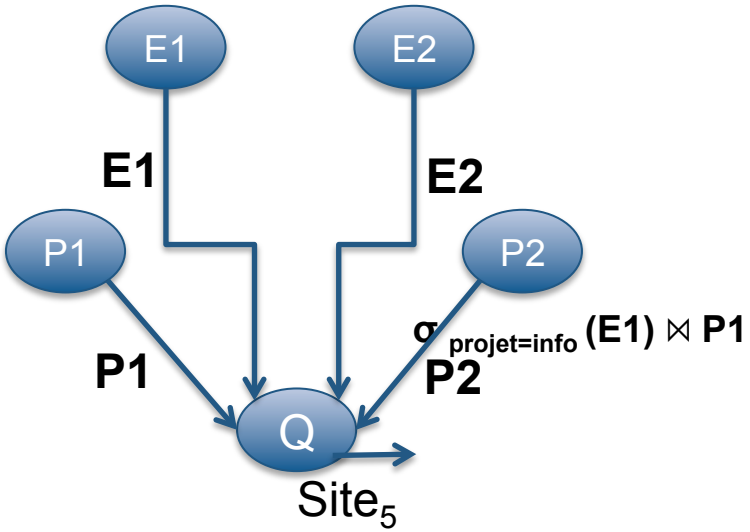
select enom
from E, P
where E.enum = P.enum
and projet = "info"

Q = Π_{enom} ($\sigma_{projet="info"}$ (E ⋈ P))
= Π_{enom} ([$\sigma_{projet=info}$ (E1 ⋈ P1)] ∪
[$\sigma_{projet=info}$ (E2 ⋈ P2)])

Plusieurs plans possibles : centralisation brutale

$E1@site1 = \sigma_{site=Paris}(E)$
$E2@site3 = \sigma_{site\neq Paris}(E)$

$P1@site2 = P \ltimes E1$
$P2@site4 = P \ltimes E2$

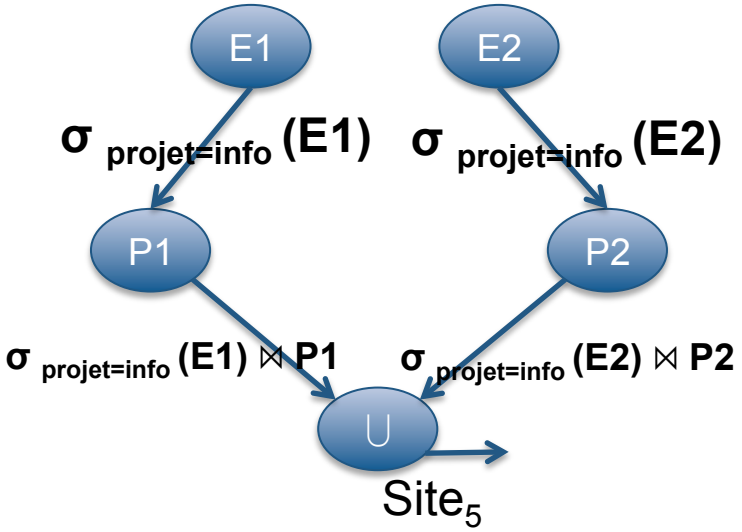


Plusieurs plans possibles : solution optimisée

$E1@site1 = \sigma_{site=Paris}(E)$ $\quad P1@site2 = P \ltimes E1$

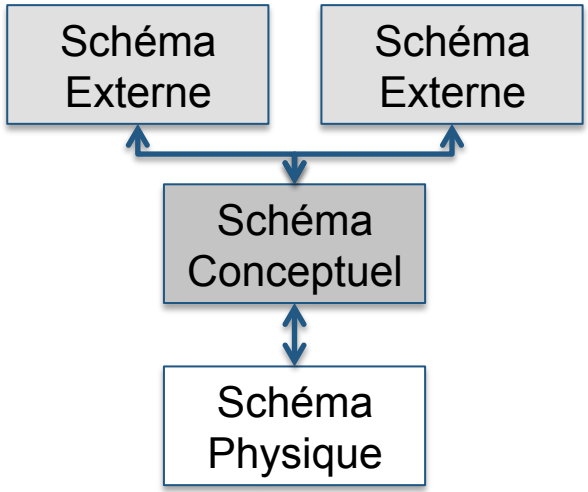
$E2@site3 = \sigma_{site\neq Paris}(E)$ $\quad P2@site4 = P \ltimes E2$

Optimisation

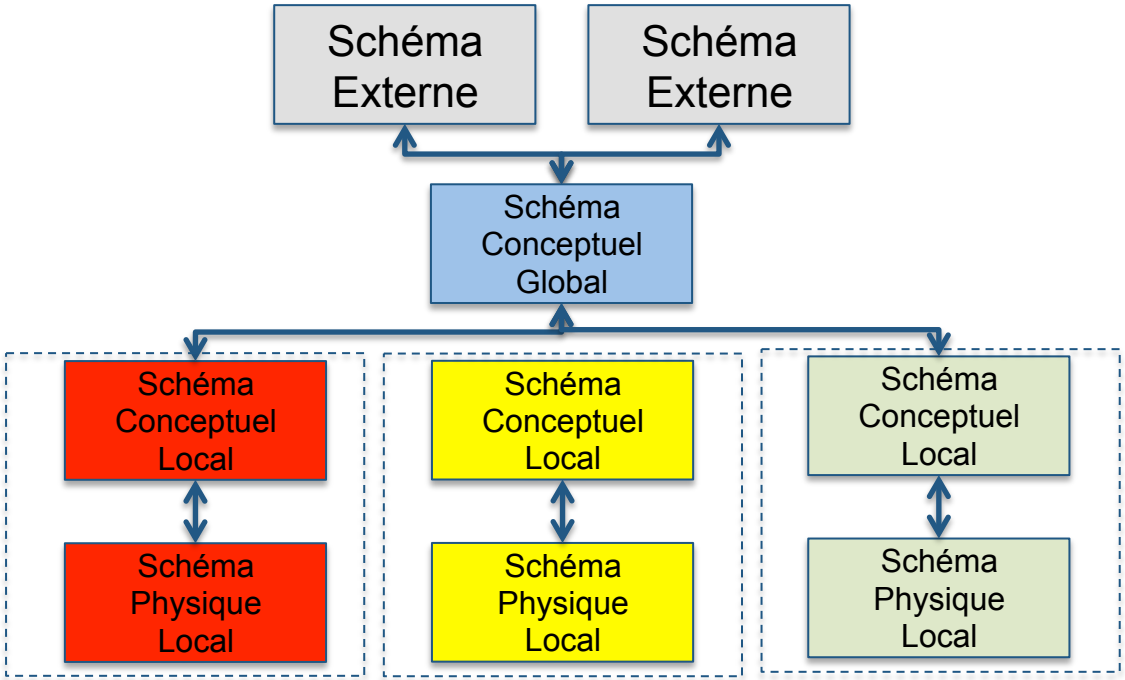


Architecture logique : intégration de données

- BD centralisé – 3 niveaux



- BD distribuée – 4 niveaux



Séparer les problèmes pour simplifier

Analyse syntaxique

- SQL Q on SCG ⇒ requête algébrique H sur SCG

Localisation

- requête algébrique H sur SCG ⇒ requête algébrique $G(H_1, H_2)$
  avec H_1,H_2 sur SCL_1, SCL_2

Optimisation globale

- Minimise globalement les communications
- Obtient $G(H_1, H_2)$ en $G'(H_1', H_2')$
  avec H_1', H_2' sur SCL_1, SCL_2

Optimisation locale

- Minimise localement les I/O (et aussi les calculs)
- Localement chaque H_i' est optimisé en H_i''
- On évalue

$$G'(H_1'', H_2'') = G'(H_1', H_2') = H = Q$$

C018SA-W6-S5

SEMAINE 6 : Bases de données distribuées

1. Introduction
2. Différentes architectures
3. Fragmentation
4. Optimisation de requête
5. **Réplication**
6. Concurrence
7. Conclusion : cinq tendances

Réplication de données : Fiabilité

Faire qu'il y ait plusieurs copies de la même donnée

Si une machine n'est plus accessible, on peut quand même accéder à la donnée

Si une machine perd une donnée, la donnée existe encore

Réplication de données : Performance

Rendre les données plus disponibles

- Réplication d'une relation
- Matérialisation d'une vue

Economiser des communications

Réplication de données : trade-off

Trade-off requêtes vs. mises-à-jour

- Classique en BD

Avantage : Requête

- Interroger une des copies
- Interroger une vue

Désavantage : Mise-à-jour

- Mise-à-jour d'une copie – propager aux autres
- Mise-à-jour d'une vue – traduire en MàJ des relations de bases – ambiguïté

Les copies peuvent ralentir les requêtes (verrous)

Exemple de réplication

Base de données

- Relation Employé E(enum, nom, site, salaire,...)
- Un employé appartient à deux sites en moyennes
- 10 sites d'à peu près la même taille
- 4 à Paris, 4 à Lyon, 2 à Marseille

Charge de travail (*workload*)

- 90% des requêtes à Paris/Lyon/Marseille sont sur des employés locaux

Réplication possible

- 3 SGBD avec les employés locaux

Performance

- Relation Employé E(enum, nom, site, salaire,...)
- Un employé n'appartient qu'à un seul site
- 40 000 employés à Paris et Lyon, 20 000 à Marseille
- 90% des requêtes sur des employés locaux
- Accès disque = 1 unité & Accès réseau = 10 unités

- Sans fragmentation 6.4 unités
- Avec fragmentation 1.9 unités
- Les données répliquées à P/L/M

  1 unité

Choisir la réplication

Que doit-on répliquer et où ?

- Problème complexe d'optimisation

Utilise une approche « gloutonne »

Faire jusqu'au point fixe

Pour chaque réplication d'un fragment sur un site

- Quel est le gain
- Quel est le coût

Répliquer le fragment tel que

- (gain > coût) et (gain – coût) maximal

C018SA-W6-S6

SEMAINE 6 : Bases de données distribuées

1. Introduction
2. Différentes architectures
3. Fragmentation
4. Optimisation de requête
5. Réplication
6. **Concurrence**
7. Conclusion : cinq tendances

Sans réplication – 2PL

Etendre ce qui marche dans le cas centralisé

- (voir Concurrence et Transaction)

La notion de sérialisabilité s'étend facilement

Le verrouillage à deux phases (2PL) aussi

- Ordonnanceur local sur chaque machine
- Verrous locaux

Difficulté – Détection de deadlocks

Détection de cycles dans le graphe « WaitFor »

- Gestion centralisée ou pas de ce graphe

Timeout

L’extension d’estampillage au cas distribué est aussi possible

Et avec réplication ?

- Méthode incorrecte mais efficace
- Méthode correcte mais couteuse

Réplication asynchrone (incorrecte)

Les copies ne sont mises-à-jour que de temps en temps

Les copies peuvent avoir des valeurs différentes – pas de maintien de la cohérence

Bien meilleures performances

Permet beaucoup plus d'autonomie entre les sites

Et une méthode correcte

- Sur une copie, l'ordonnancement doit être équivalent à un ordonnancement sériel sur une base de données à une seule copie, et
- Toutes les copies doivent être identiques

Problème avec la réplication

Entité x dupliquée sur $site_1$ et $site_2$

Deux transactions :

- T1: read(x); x:=x+5; write(x); commit
- T2: read(x); x:=x*10; write(x); commit

Deux ordonnancements

- $Site_1$: R1(x),W1(x),C1,R2(x),W2(x),C2
- $Site_2$: R2(x),W2(x),C2,R1(x),W1(x),C1

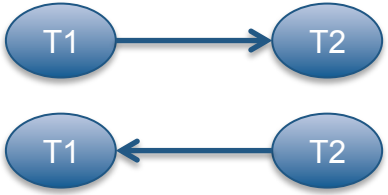
Supposons que au départ

- Avant x@s1 = x@s2 = 1
- Après x@s1 = 60 and x@s2 =15
- Incohérence entre les deux copies

La sérialisabilité ne fonctionne plus

Chacun des deux ordonnancements est sérialisable (sériel même)

- $Site_1$: R1(x),W1(x),C1,R2(x),W2(x),C2
- $Site_2$: R2(x),W2(x),C2,R1(x),W1(x),C1



Problème : deux opérations conflictuelles apparaissent dans des ordres différents dans les deux sites

Sérialisabilité revisitée

1. L'ordonnancement doit être sérialisable sur chaque site
2. Deux opérations conflictuelles doivent apparaître dans le même ordre sur tous les ordonnanceurs où elles apparaissent

Question : comment garantir ça ?

Une technique ROWA

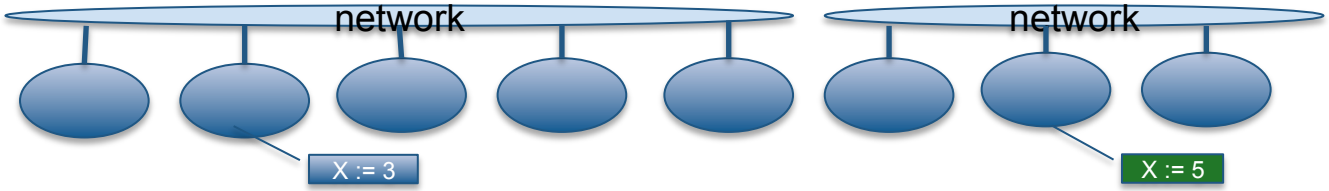
ROWA : Read-once/write-all

- Read(x) traduit en read(x@s) sur une copie x@s de x
- Write(x) traduit en write(x@s) pour toutes les copies x@s de x

Si une copie échoue, cela bloque la transaction

Alternative

- Write sur toutes les copies disponibles
- À condition d'avoir une majorité de copies disponibles



Se protéger contre les partitionnements du réseau

Pourquoi ROWA marche ?

Deux opérations conflictuelles doivent inclure au moins un *write*

Les *write* sont dans le même ordre (l'ordre défini par la majorité)

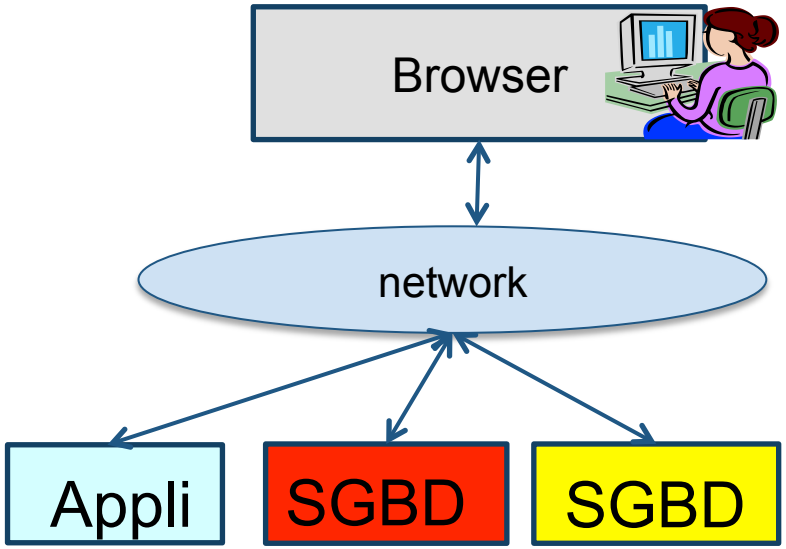
Très efficace quand il y a beaucoup plus de *read* que de *write*

C018SA-W6-S7

SEMAINE 6 : Bases de données distribuées

1. Introduction
2. Différentes architectures
3. Fragmentation
4. Optimisation de requête
5. Réplication
6. Concurrence
7. **Conclusion : cinq tendances**

Le cloud



- L’utilisation de la puissance de calcul ou de stockage de serveurs informatiques distants par l'intermédiaire d’Internet
- L’appli est dans le cloud
- Les données sont dans le cloud

NoSQL

Systèmes de gestion de données spécialisés

pour des applications « extrêmes »

- OLTP : millions de transactions par seconde
- OLAP : Analyse de téraoctets de données

Eviter la lourde surcharge que paient les SGBD relationnels pour leur universalité

- Langage plus simple (pas SQL)
- Modèle plus simple (clé-valeur)
- Gestion de la concurrence plus limitée (pas ACID)

Le pair-à-pair

Chaque machine est à la fois un serveur et un client

Autonomie totale

Mieux utiliser les ressources disponibles sur le réseau

- Les CPU inoccupés
- Les espaces libres sur disques et mémoire
- Les réseaux de communications disponibles

Exemple

- Serveur de musique ou vidéos

Big data

De plus en plus de données disponibles

- Web, réseaux sociaux, Internet des objets, téléphones intelligents…

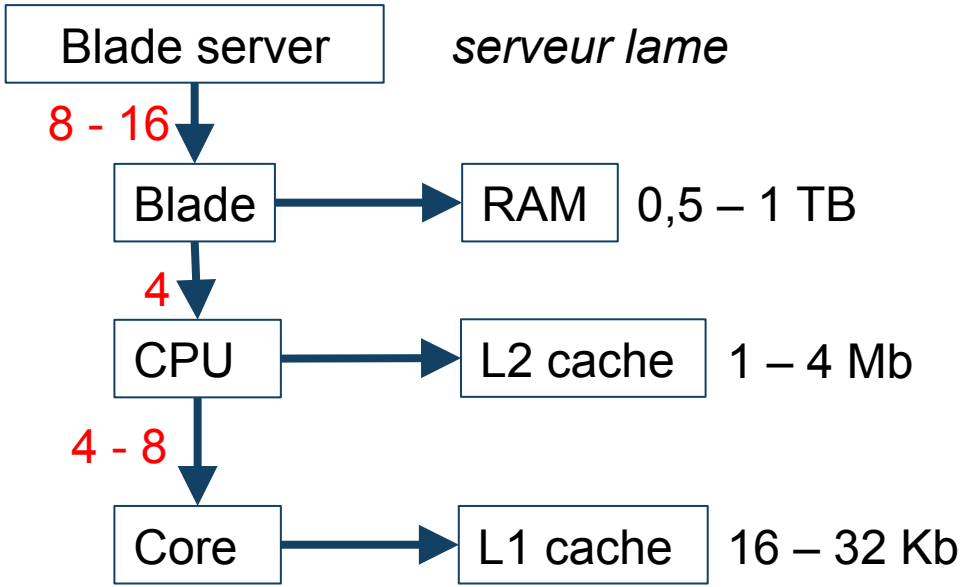
Analyse de ces données pour en tirer de la valeur

- Techniques d'apprentissage (machine learning)

Calcul massivement parallèle comme Hadoop

Systèmes NoSQL

Base de données en mémoire



Des serveurs avec tellement de mémoire que la BD tient dedans

On retrouve les hiérarchies de mémoires du cours Indexation

Base de données en mémoire

- **Au delà de 100 *cores***
- **Au delà de 10 Tb de mémoire**
- **MaJ : écriture sur disque pour fiabilité**

- Problèmes
  - ✓ Programmation complexe pour utiliser tout le parallélisme
  - ✓ Dissipation d'énergie

Parallélisme contre distribution

- Machines parallèles
  - ✓ Homogènes, couplage fort
  - ✓ But premier : performance
- Machines distribuées
  - ✓ Hétérogènes, couplage faible, indépendance
  - ✓ But premiers : sureté et performance

Merci

Serge Abiteboul, Benjamin Nguyen, Phillippe Rigaux