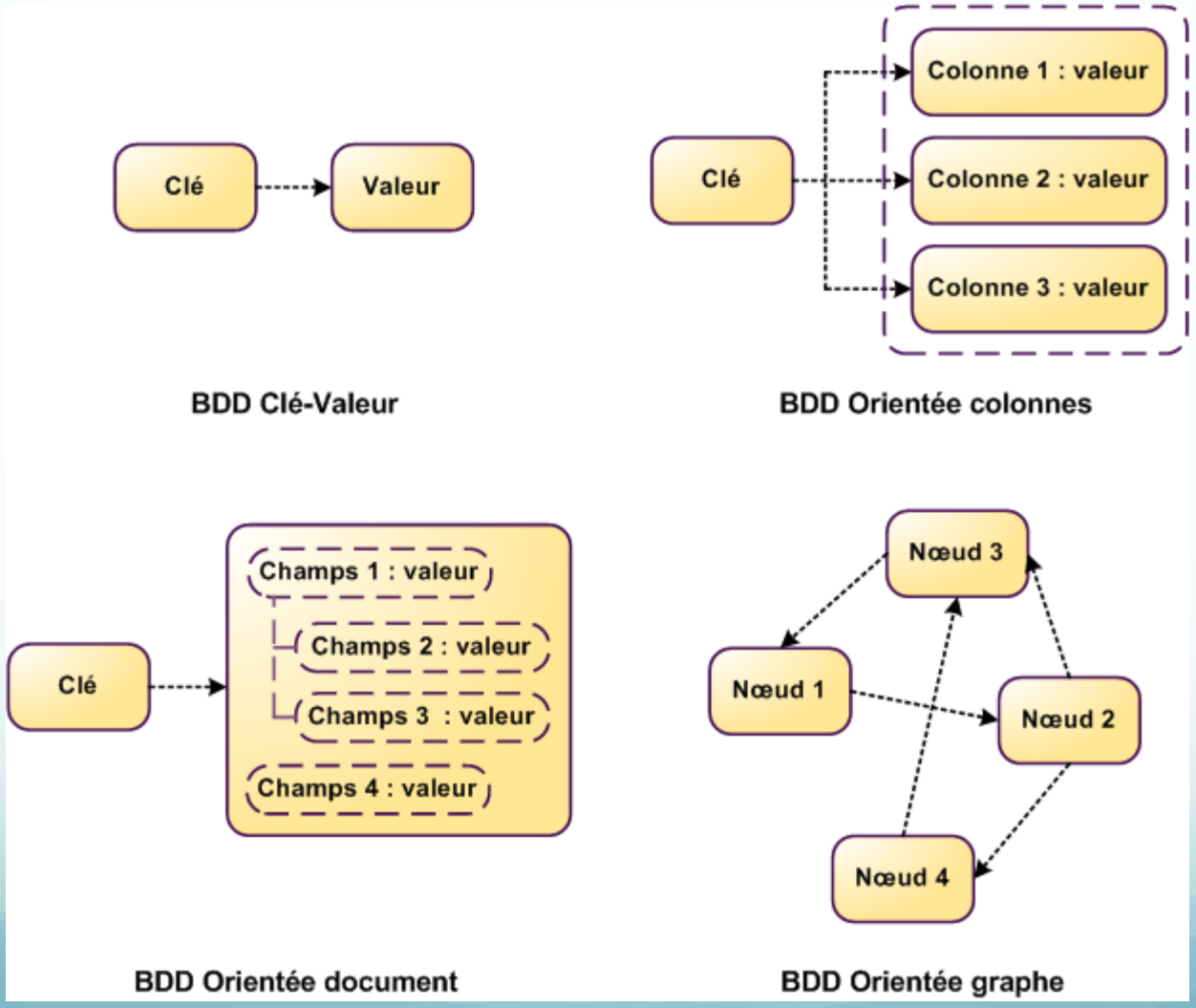
Persistance

Michel Gagnon, Nikolay Radoev
École Polytechnique de Montréal

SQL vs NoSQL

- Les BD relationnelles ne sont pas nécessairement adaptées aux grands volumes de données hétérogènes
- Les BD NoSQL visent à favoriser les données distribuées
- Les BD NoSQL utilisent des modèles de données plus simples, donc moins structurées
  - Schémas statiques (BD relationnelles) vs schémas dynamiques (BD NoSQL)
- Les BD relationnelles assurent l'atomicité des transactions et la cohérence des données (propriétés ACID)
  - Certaines BD NoSQL les assurent aussi

Familles de NoSQL



BD Clé-Valeur

- La plus simple des familles
- Information sauvegardé dans une collection de paires clé-valeur
  - Chaque clé est unique
  - La valeur est souvent représentée par un *string*
  - Pas de schéma défini : chaque paire est indépendante
- Adapté pour l'accès rapide à de l'information simple
- Moins pratique pour des données complexes
- Exemples :
  - Riak : BD distribuée avec une forte tolérance aux pannes
  - **Redis** : BD en-mémoire (conservé en RAM) très populaire
  - Voldemort : Créé et utilisé par LinkedIn jusqu'à 2018 (remplacé par Espresso, une BD orientée document)

BD Orientées document

- Adapté au monde du web
  - Le format JSON (ou des dérivés) est très utilisé pour représenter l'information
- Extension du concept clé-valeur qui représente la valeur sous-forme de document
  - La clé est souvent un élément du document
  - Chaque document peut avoir son propre schéma
  - Un document peut contenir d'autres documents (similiare à JSON)
- Exemple :
  - CouchDB : BD basée sur une API REST qui peut être déployée sur des appareils mobiles
  - **MongoDB** : Très populaire avec un offre de services supplémentaires au dela de la sauvegarde de documents

BD orientées colonnes

- S'oppose à la représentation des tables dans les BD relationnelles.
  - Les BD relationnelles manipulent les colonnes d'une ligne d'une table de manière statique
  - Chaque colonne = nouveau fichier, donc il n'est pas nécessaire de reconstruire la table si on modifie le schéma des données
- Très pratique pour des requêtes qui traitent seulement un sous-ensemble de chaque donnée (une ou plusieurs colonnes)
- Stocke des informations avec une clé unique par rangée.
- **BigTable** : BD derrière les produits de Google. Encodage en 3D (colonne + rangée + *timestamp*) pour la sauvegarde.
  - Exemple : une vidéo YouTube où colonne = URL, rangées = propriétés et *timestamp* = versions différentes de la vidéo

BD orientées graphe

- Modélisation, stockage et manipulation de données complexes liées par des relations non-triviales ou variables.
- Cas d'utilisation : informations sur les réseaux sociaux. Ex : comptes Facebook liés à des « amis »
- Cas d'utilisation : bases de connaissance avec des données structurées. Utile pour des agents intelligents (Alexa, Siri, etc)
- Exemples :
  - **Neo4j** : utilisé pour des systèmes de recommandation pour des sites de commerce
  - Knowledge Graph : Base de connaissance de Google derrière les boîtes d'info dans les recherches de Google ainsi que Google Assistant.

NoSQL vs SQL

- Il ne s'agit pas de remplacer les SGBDR.
- NoSQL est plus récent, mais gagne en popularité.
  - Outillage et communautés en pleine croissance.
  - Rattrapage de la maturité de MySQL ou PostgreSQL
- Intérêt de NoSQL
  - Préférence de représentation moins rigide
  - Adapté pour un grand volume de données (ex: Cassandra)
- Pas une solution miracle pour tout type de stockage de données.
  - Conversion d'une représentation habituellement offert par une BD SQL vers NoSQL peu aboutir à une solution peu efficace.
  - Pour des donnés avec des schémas stables, une BD SQL peut être une meilleure solution.

MongoDB

- Choix de BD NoSQL pour le cours LOG4420
  - S'intègre bien avec des serveurs utilisant NodeJS/Express
  - Utilise un format similaire au JSON (BSON)
- Basé sur le concept de documents
- Un serveur contient des bases de données
  - Une machine peut héberger plusieurs instances
- Une base de données contient des collections
- Une collection contient des documents
  - Chaque document peut avoir son propre schéma
  - Utilise le format BSON : Binary JSON

MongoDB
Structure d'un document

- MongoDB utilise le format BSON qui est un objet JSON encodé en binaire

- BSON contient les types de JSON ainsi que quelques types supplémentaires:
  - Double, Object, Binary data, ObjectId,Date, Null, Regular Expression, JavaScript, Symbol, 32-bit integer, Timestamp, 64-bit integer, Min key, Max key, etc.

- Attribut **_id** réservé pour définir une clé primaire
  - Si ce n'est pas fourni lors de la création du document, MongoDB en génère un automatiquement
  - Toujours le premier élément du document et forcément unique

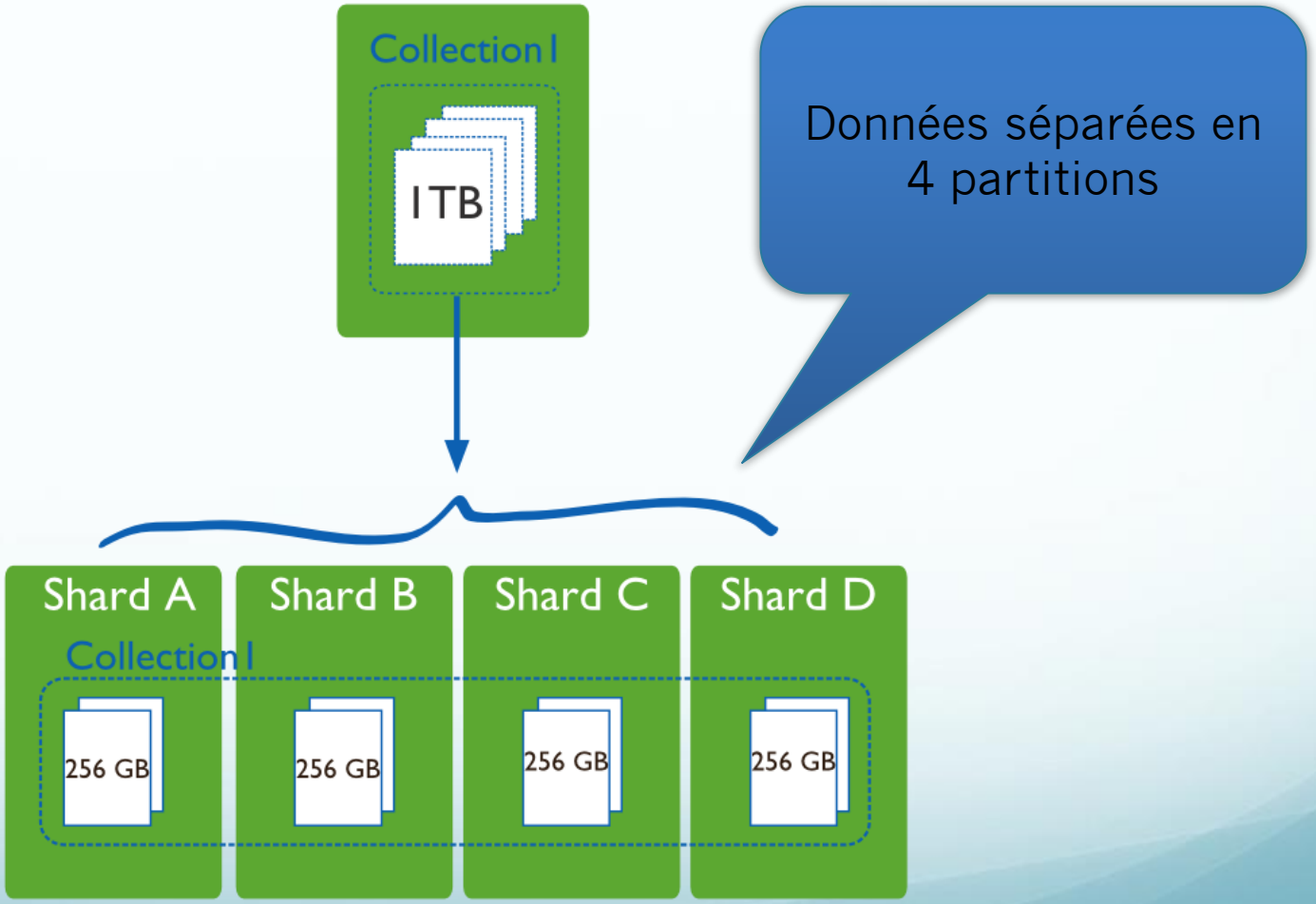
MongoDB

- Distribution horizontale des données (*sharding*)
  - Attention: ne pas confondre avec la redondance des données, il s'agit ici de partitionner les données et placer les partitions sur différents serveurs
- Depuis la version 4.0 (2018), MongoDB offre un support limité pour des transactions multi-documents ACID.
  - Depuis 4.2 (2019), ceci s'applique pour des documents sur plusieurs shards
- MongoDB peut être utilisé comme un système de fichiers distribués ou trop grands (>16 MB) avec la spécification GridFS

MongoDB
Distribution de données

- La distribution de données (*sharding*) est une méthode pour distribuer les données d'une BD sur plusieurs machines différentes.
  - Une très grosse BD avec beaucoup d'accès en écriture peut surutiliser la capacité d'un seul serveur
- Horizontalement évolutif (horizontal scaling)
  - Combinaison de plusieurs machines moins puissantes
- MongoDB distribue les lectures/écritures sur toutes les partitions d'une grappe.

MongoDB
Distribution des données

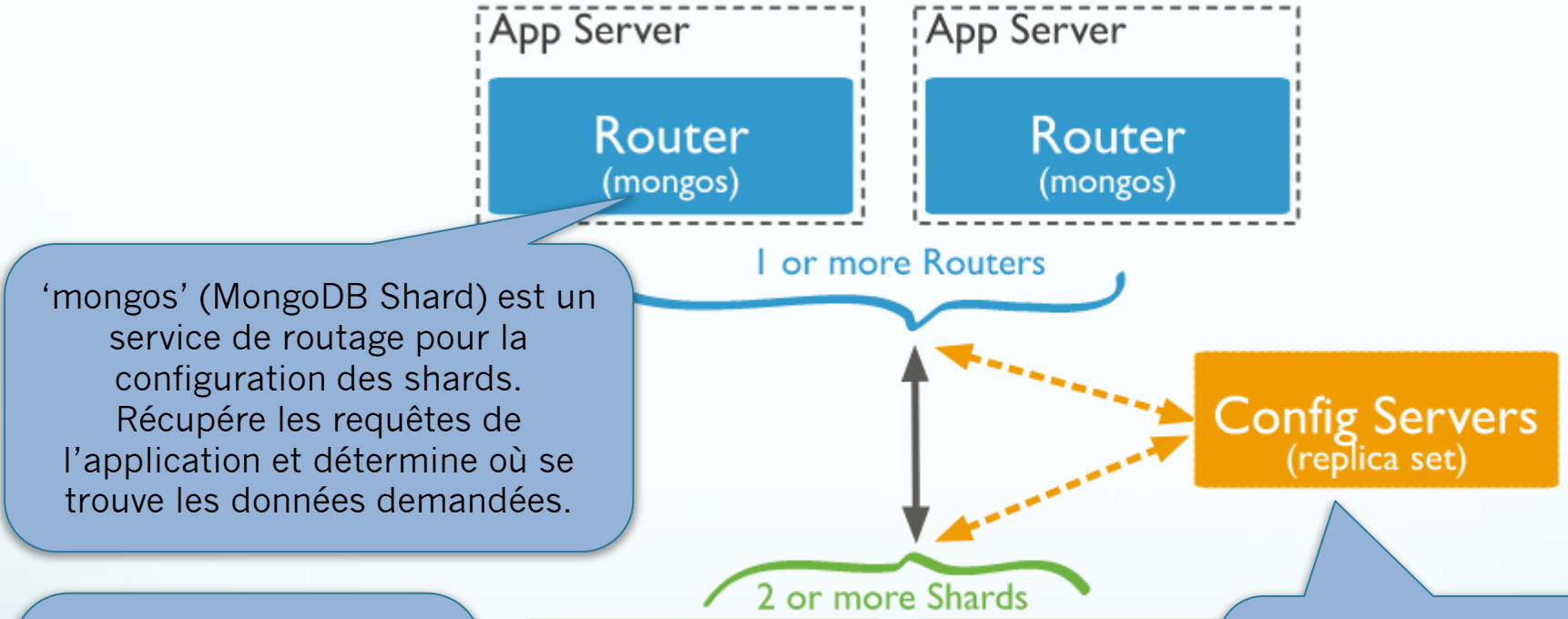


Extrait de http://docs.mongodb.org/manual/core/sharding-introduction/

MongoDB
Distribution de données

- Pour distribuer les documents d'une collection, MongoDB partitionne la collection en utilisant un *shard key*.
  - Cette clé est générée par MongoDB et se trouve dans chaque document d'une collection
  - On peut spécifier notre propre clé si on veut distribuer les documents d'une manière spécifique
- Plusieurs types de clé sont possibles pour une meilleure distribution
  - Le choix de la clé affecte la performance et l'évolutivité d'une grappe.

MongoDB
Distribution des données



‘mongos’ (MongoDB Shard) est un service de routage pour la configuration des shards. Récupére les requêtes de l’application et détermine où se trouve les données demandées.

Chaque partition contient un sous-ensemble des données. Chacune des partitions peut être dupliquée

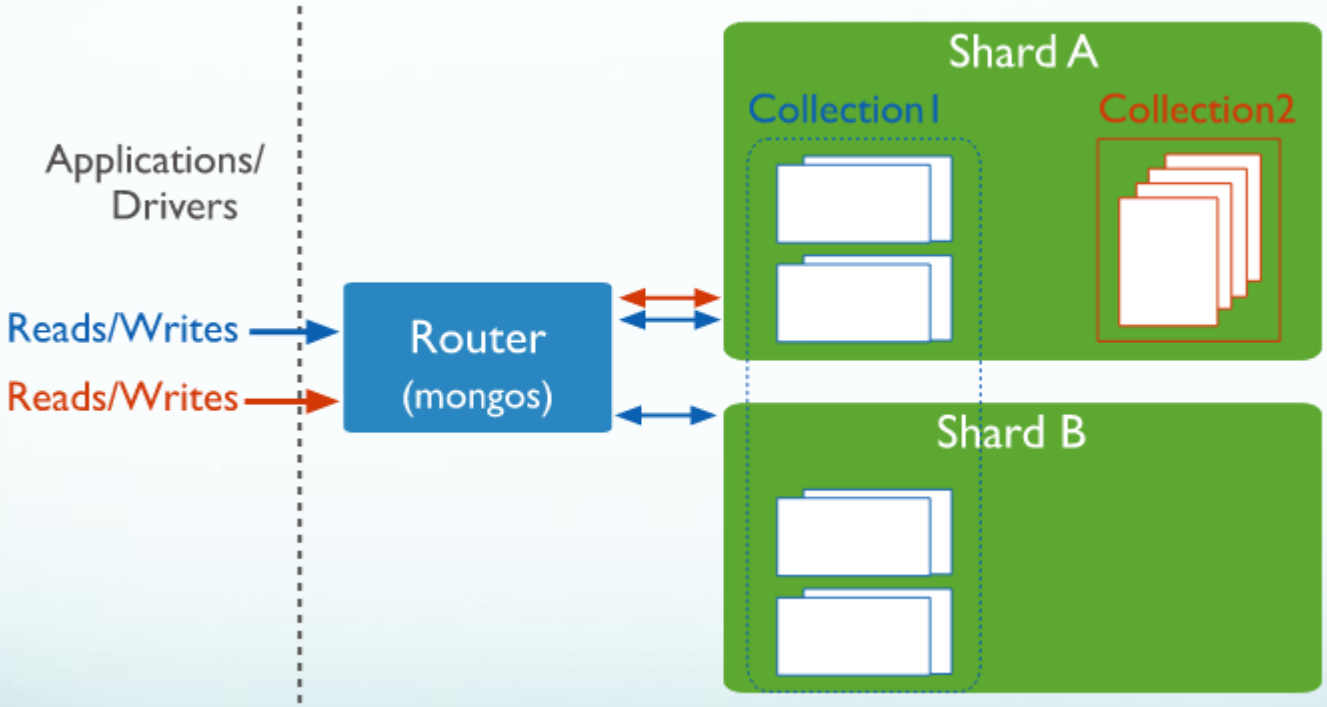
Contiennent les méta-données.
Sont consultés par le routeur pour savoir où envoyer la requête

Extrait de http://docs.mongodb.org/manual/core/sharding-introduction/

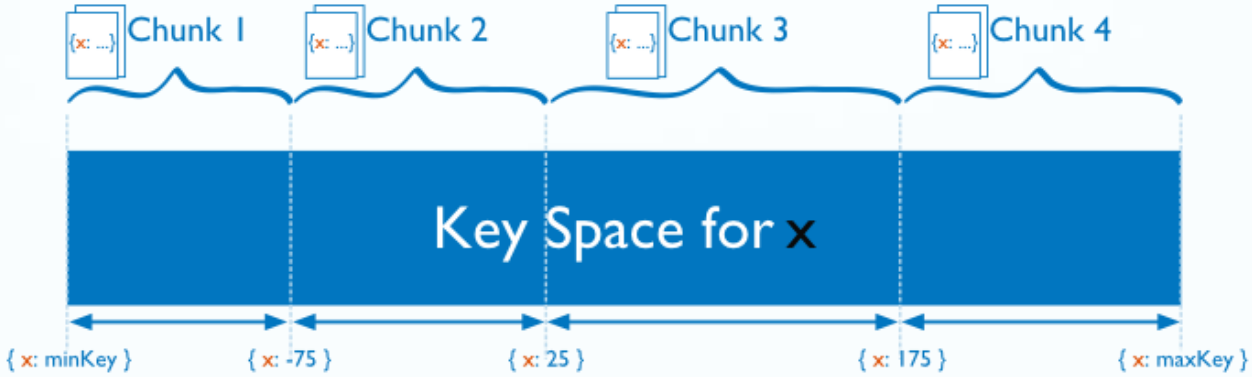
MongoDB
Distribution de données

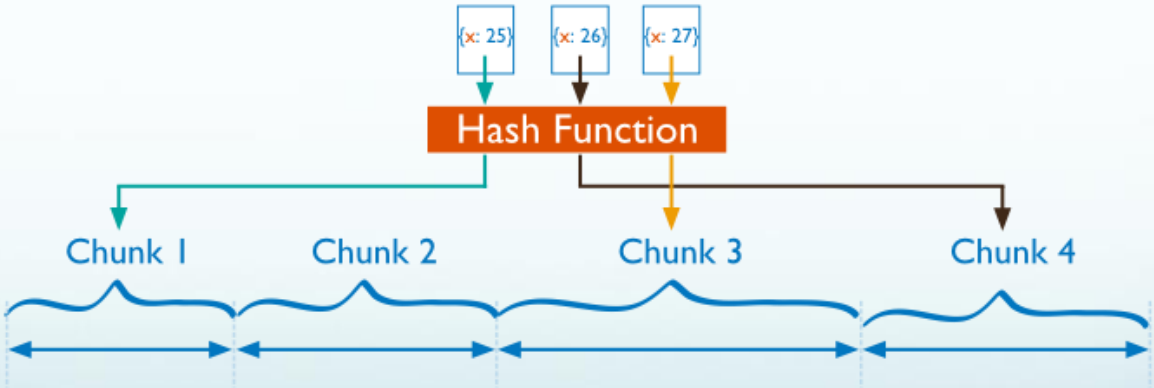
- MongoDB distribue les lectures/écritures sur toutes les partitions d'une grappe.
  - Permet à chaque partition de traiter une partie des données de façon parallèle.
- Si la taille des données augmente, on peut simplement augmenter le nombre de partition (donc augmenter le nombre de machines en conséquence)
- Si une machine tombe en panne, MongoDB peut tout de même faire une lecture/écriture partielle sur les machines qui sont en service
  - Dans ce cas, il est préférable d'avoir de la redondance de données de MongoDB sur différentes machines.

MongoDB
Distribution de données



MongoDB - Deux manières de partitionner

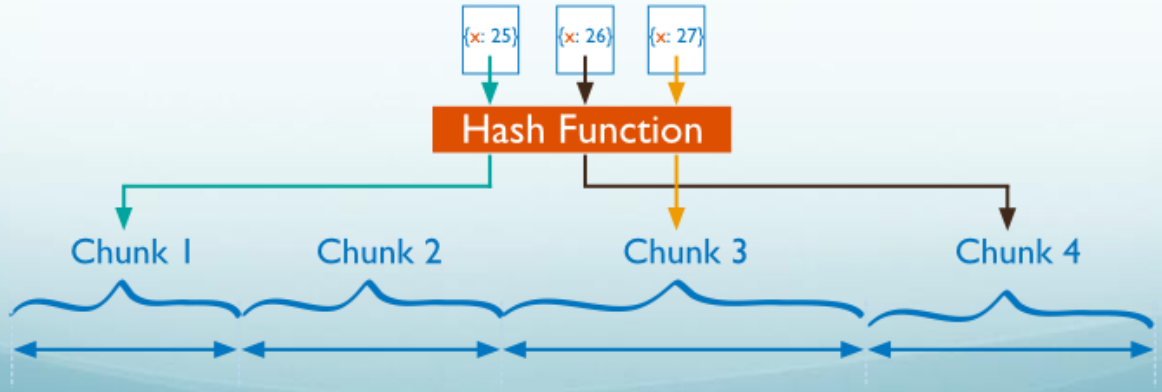




Extrait de http://docs.mongodb.org/manual/core/sharding-introduction/

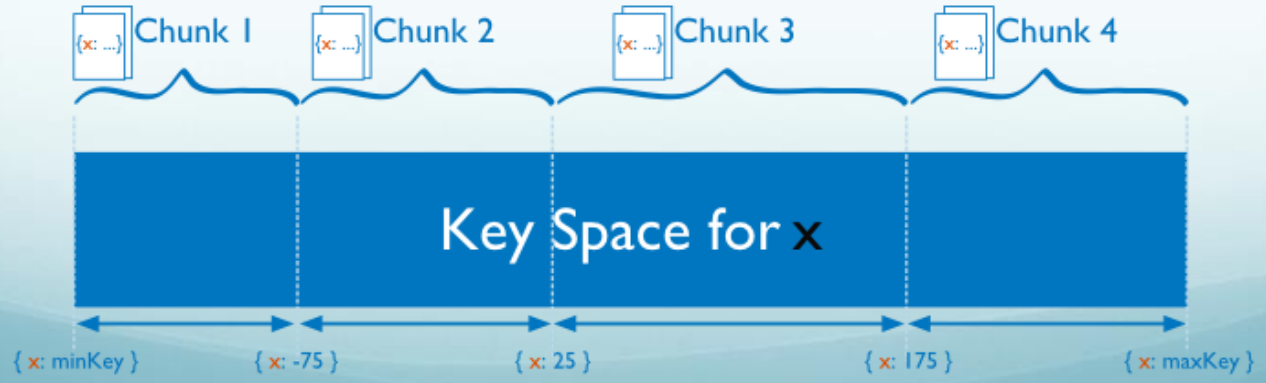
MongoDB
Paritionnement par clé hashée

- À partir du *shard key* d'un document, MongoDB calcule une clé hashée.
- Chaque partition est donc assigné un intervalle basé sur les clé hashées
- Des *shard key* proches ne seront pas nécessairement dans la même partition.
- Ce type de partitionnement facilite la distribution uniforme des données

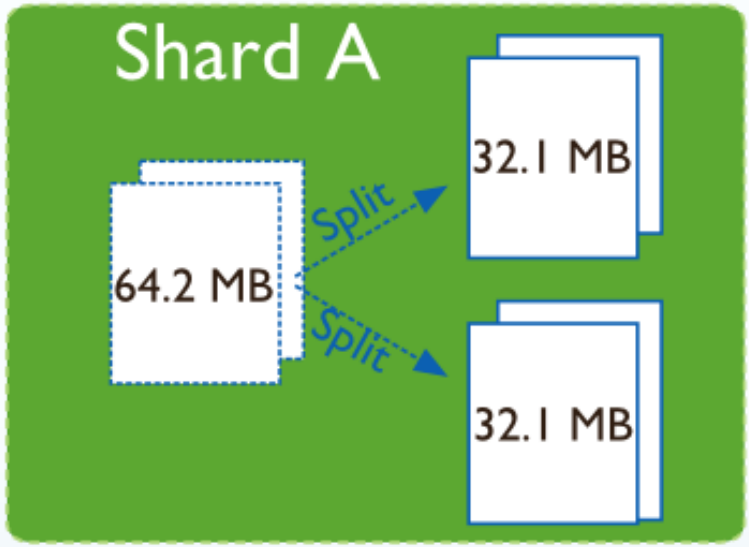


MongoDB
Paritionnement par intervalle

- On calcule des intervalles en fonction des *shard key* de chaque document.
- Chaque partition est assignée un intervalle de valeur
- Les clés proches ont une forte chance de se retrouver dans une même partition
- Permet de mieux cibler les partitions qui risquent de contenir les données lors d'une lecture
- Une clé mal choisie risque de déséquilibrer la partition des données



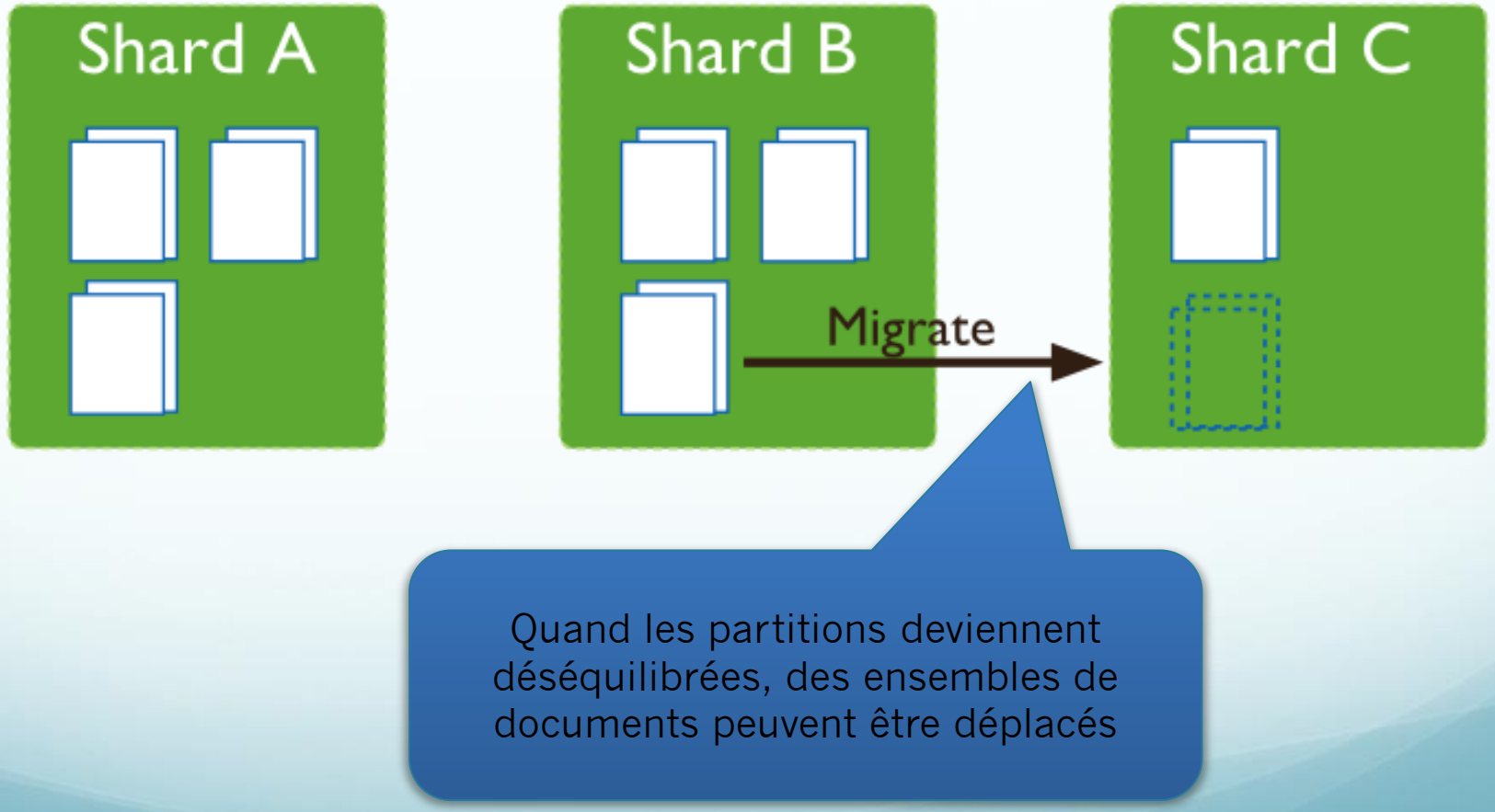
MongoDB - Partitionnement des groupes de données



Quand un ensemble de documents devient trop gros, il est divisé en deux

Extrait de http://docs.mongodb.org/manual/core/sharding-introduction/

MongoDB
Équilibrage des données



Extrait de http://docs.mongodb.org/manual/core/sharding-introduction/

MongoDB
Redondance des données

- Une instance primaire et une ou plusieurs instances secondaires
- L'instance primaire est la seule qui accepte les requêtes d'écriture
- Par défaut, on lit toujours sur l'instance primaire
- Les instances secondaires répètent sur leurs données (de manière asynchrone) les opérations effectuées sur l'instance primaire
- Si l'instance primaire tombe en panne, les autres instances élisent une nouvelle instance primaire
- Une instance peut être un arbitre (pas de données, mais participe aux élections)
- Une instance secondaire de priorité 0 ne peut pas devenir primaire
- Une instance secondaire peut être non votante

MongoDB Requêtes

- MongoDB utilise son propre langage de requêtes contrairement au SQL
  - En pratique, on accède à une instance Mongo à travers un *driver* spécifique à notre langage de programmation qui implémente l'API de Mongo.
    - Exemple : le *driver* pour NodeJS
  - L'utilisation d'un ODM (Object Data Model) comme mongoose permet de manipuler les documents de MongoDB comme des objets
    - Impose un schéma à chaque document
    - Simplifie la syntaxe de MongoDB
    - Une alternative intéressante utilisée pour vos TPs

MongoDB
Recherche de données

- Méthodes **find()**, **findOne() et findById()**
- S'appliquent toujours sur une collection spécifique
- On lui passe comme premier argument un critère pour la recherche
- On peut aussi lui passer un deuxième argument, qui spécifie les champs qu'on veut extraire (la projection)
- Un critère ressemble à un document, mais peut contenir des opérateurs spéciaux
- Exemples:
  - **db.prod.find({ qty: { $gt: 25 } })** (tous les documents qui spécifient une quantité > 25)
  - **db.prod.find({_id: { $in: [5, ''alsl1231"] })** (document dont l'identificateur est 5 ou ''alsl1231"
- La méthode **find()** retourne un curseur (un type d'itérateur) pour parcourir l'ensemble de documents obtenus

MongoDB
Syntaxe des opérateurs

- La syntaxe des requêtes de MongoDB rassemble beaucoup à du code JS, mais avec quelques différences
  - Les opérateurs de MongoDB commencent par **$**

- Opérateurs logiques :
  - $and : permets de joindre plusieurs requêtes ensemble et retourne seulement les documents qui satisfont toutes les requêtes.
    - Exemple : **find($and : [{ sigle : /^LOG/ }, { credits : {$lt : 4} }])** qui cherche tous les cours qui ont un sigle qui commence par **LOG** ET qui ont moins que 4 crédits.
  - $not : inverse le sense d'une requête et retourne les documents qui ne correspondent pas à la requête
  - $or : permets de joindre plusieurs requêtes ensemble et retourne les documents qui satisfont au moins une requête.
    - Exemple : find($or : [{ sigle : /^LOG/ }, { credits : {$lt : 4} }]) qui cherche tous les cours qui ont un sigle qui commence par **LOG** OU qui ont moins que 4 crédits.

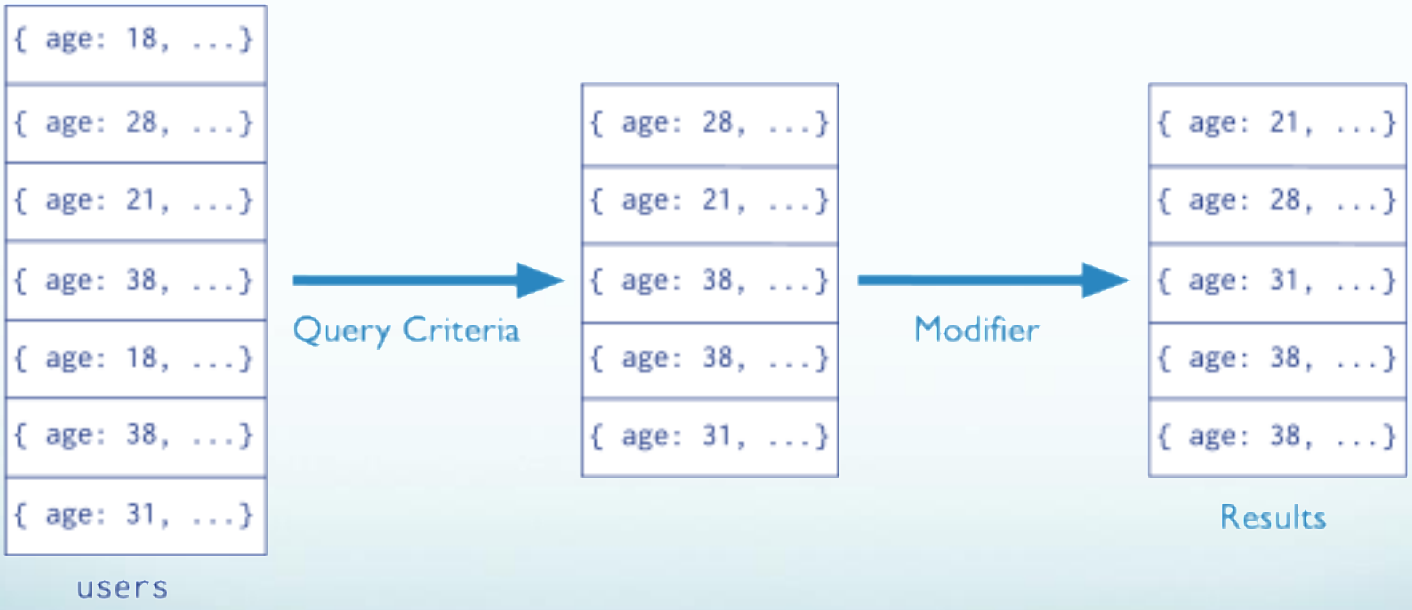
MongoDB
Syntaxe des opérateurs

- Opérateurs logiques :
  - $eq : effectue une comparaison d'égalité
  - $gt : effectue une comparaison de type strictement plus grand
  - $gte : effectue une comparaison de type plus grand ou égal
  - $lt : effectue une comparaison de type strictement plus petit
  - $lte : effectue une comparaison de type plus petit ou égal
  - $in : effectue une comparaison avec les éléments fournis dans un tableau.
    - Ex : find(sigle :{$in: ["LOG8480","LOG4420"] }) retourne seulement les cours dont le sigle est LOG8480 ou LOG4420.
  - $nin : l'inverse de $in et retourne les documents qui ne correspondent pas aux critères fournis dans le tableau.

Exemple - Requête find() avec projection

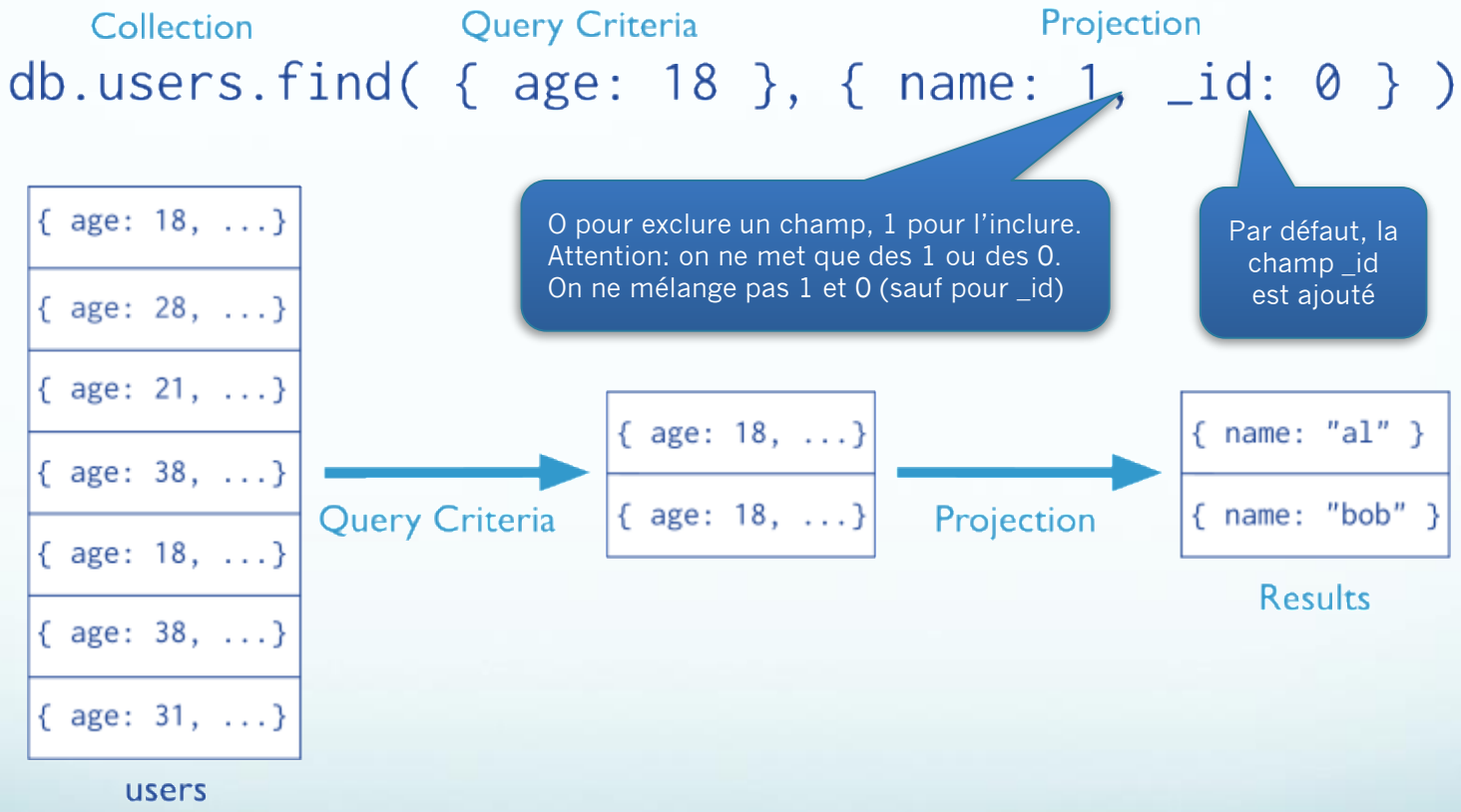
Collection — Query Criteria — Modifier

```
db.users.find( { age: { $gt: 18 } } ).sort( {age: 1 } )
```



Extrait de https://docs.mongodb.org/manual/core/crud-introduction/

Exemple - Requête find() avec projection



Extrait de https://docs.mongodb.org/manual/core/read-operations-introduction/

MongoDB Curseur

- Le curseur est la structure de documents retournée suite à une requête sur une collection de MongoDB
  - Le curseur est une référence vers l'information sur l'instance MongoDB.
  - La méthode *toArray()* permet d'obtenir un tableau JS, mais perd la flexibilité des méthodes du curseur
  - Il est toujours plus efficace d'appliquer des modificateurs/manipulations sur le curseur que de le transformer en *array*

MongoDB Curseur

- Méthodes utiles:
  - next() et hasNext()
  - toArray()
  - forEach()
  - sort()
  - skip()
  - limit()
- Les méthodes peuvent être appelées en cascade:
  - **db.prod.find().sort({nom: 1, prenom : -1}).skip(10).limit(5)**
  - Dans le cas de *sort*, le tri s'applique en ordre ascendant (1) ou descendant (-1) de gauche à droite.
    - L'exemple précédant est équivalent à :

    db.prod.find().sort({nom:1}).sort({prenom:-1})

MongoDB
Modification de données

- Méthodes **insertOne()** et **insertMany()**
  - Prend en argument le(s) document(s) à ajouter et une fonction de *callback*
  - Si on ne spécifie pas de valeur à **_id**, un identificateur unique sera généré automatiquement
- Méthodes **updateOne()** et **updateMany()**
  - À moins de le spécifier dans les options, un seul document est mis à jour
  - L'option **upsert** permet d'ajouter un nouveau document si aucun n'est trouvé
- Méthodes **deleteOne()** et **deleteMany()**
  - Prend en argument le critère identifiant le(s) document(s) à retirer
- ***Many()** retourne le ObjectId des documents modifiés (insertion, modification ou suppession)

Modification de données – Exemple d'insertion

```
const cours = db.collection("cours");
const log4420 = { sigle : "LOG4420", credits : 3};
cours.insertOne(log4420, (error,res) => {
    if (err) throw err;
})

const nouveauxCours = [
    { sigle : "LOG8480", credits : 3},
    { sigle : "INF1010", credits : 2},
    { sigle : "LOG4900", credits : 6} ];
cours.insertMany(nouveauxCours, (error,res) => {
        if (err) throw err;
})
```

Modification de données – Exemple de modification

```
const cours = db.collection("cours");
const query = { sigle : "LOG4420"}
const nouveauxCredits = {$set : {credits : 5}}

cours.updateOne( query,  nouveauxCredits, (err,res) => {
        if (err) throw err;
    })
```

Outre que **$set**, on peut utiliser d'autres opérateurs pour modifier un ou plusieurs documents :

$inc : incrémente la valeur du champ par le paramètre fourni
$mul : multiplie la valeur du champ par le paramètre fourni
$rename : renome le champ
$unset : retire le champ spécifié du document

Modification de données – Exemple de suppression

```
const cours = db.collection("cours");
const query = { sigle : "LOG4420" };
cours.deleteOne(query, (error,res) => {
    if (err) throw err;
})

// Supprime tous les cours qui ont plus que 2 et moins que 5
crédits (donc 3 ou 4)
const queryCredits = {credits : {$gt : 2, $lt : 5};
cours.deleteMany(queryCredits , (error,res) => {
        if (err) throw err;
})

// Supprime la collection au complet
cours.drop()
```