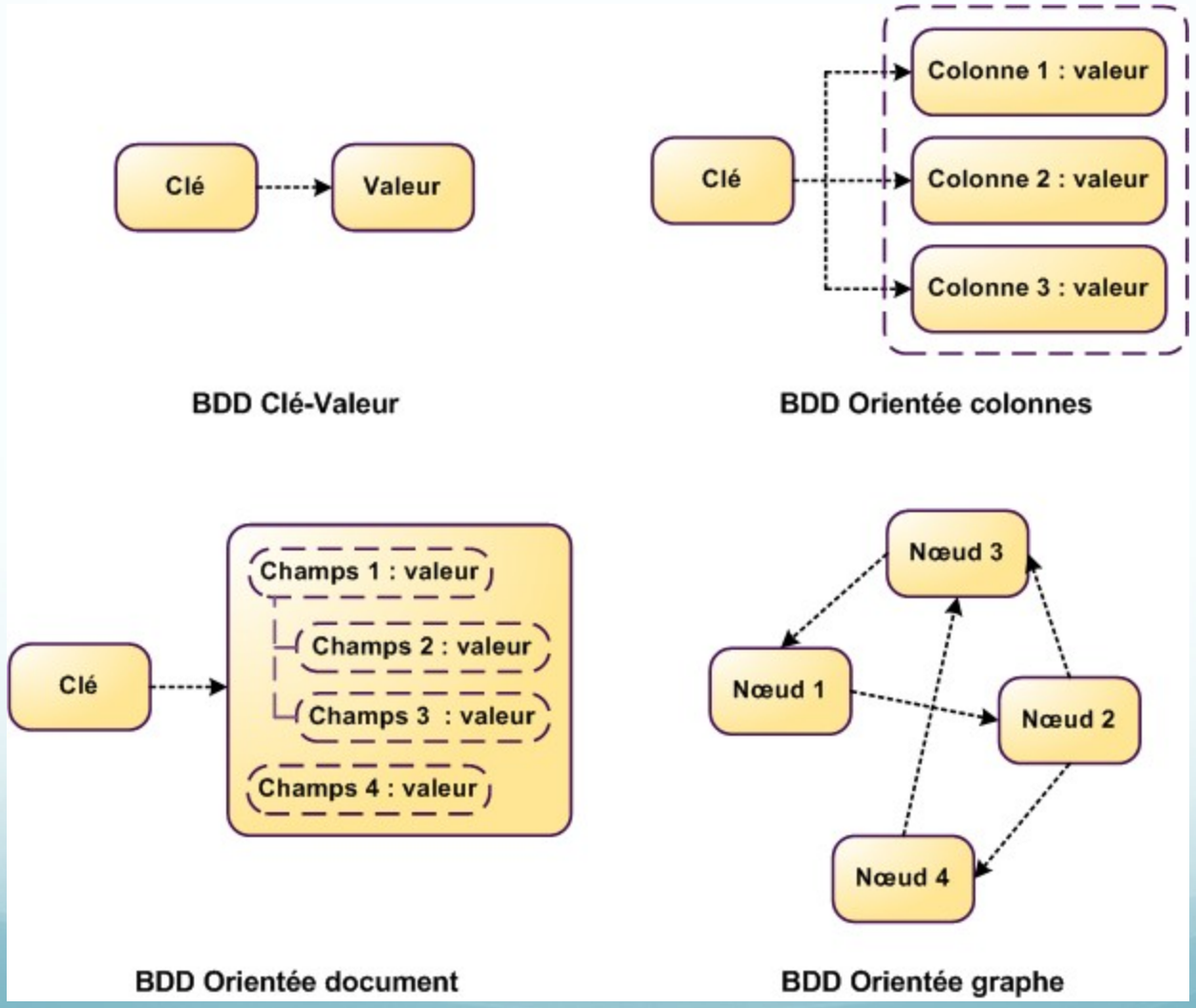
Persistance

Michel Gagnon
École polytechnique de Montréal

SQL vs NoSQL

- Les BD relationnelles pas nécessairement adaptées aux grands volumes de données
- Les BD NoSQL visent à favoriser les données distribuées
- Les BD NoSQL utilisent des modèles de données plus simples, donc moins structurées
- Les BD relationnelles reposent sur des technologies matures
- Schémas statiques (BD relationnelles) vs schémas dynamiques (BD NoSQL)
- Les BD relationnelles assurent l'atomicité des transactions et la cohérence des données

Familles de NoSQL



BD Clé-Valeur

- La plus simple des familles
- Adapté aux caches ou accès rapides aux informations
- Exemples : Riak, Redis, Voldemort

BD Orientées document

- Adapté au monde du web
- Extension du concept clé-valeur qui représente la valeur sous-forme de document
- Hiérarchique, similaire à du JSON
- Exemple : CoucheDB, **MongoDB**

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
- Vision plus flexible permettant d’avoir des colonnes différentes pour chaque ligne et de multiplier de manière conséquente le nombre de colonne par ligne
- Stocke des informations par clé.
- Exemples : HBase, Cassandra

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
- Exemples : Neo4j, HypergraphDB, FlockDB

NoSQL vs SQL - Mythes

- Il ne s'agit pas de remplacer les SGBDR.
- NoSQL relativement récent.
  - Outillage et communautés pas comparable à celles des SGDBR courants.
  - Loin de la maturité de MySQL ou PostgreSQL
- Intérêt de NoSQL ne dépend pas de la quantité de volume de données à manipuler
  - Intérêt se trouve dans la préférence de représentation
- Pas une solution miracle pour tout type de stockage de données.
  - Conversion d'une représentation habituellement offert par une BD SGBDR vers BD NoSQL peu aboutir à une solution peu efficace

MongoDB

- Choix de BD NoSQL pour le cours LOG4420
- Basé sur le concept de document
- Un serveur contient des bases de données
- Basée sur la notion de maître/esclaves
- Une base de données contient des collections
- Une collection contient documents
- 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

MongoDB
Structure d’un document

- Un document est essentiellement un objet JSON
- Type de données: Double, Object, Array, Binary data, Object id, Boolean, Date, Null, Regular Expression, JavaScript, Symbol, 32-bit integer, Timestamp, 64-bit integer, Min key, Max key
- Attribut **_id** réservé pour définir une clé primaire

MongoDB vs MySQL

- MySQL
  - Maturité : grosse communauté et stable
  - Compatibilité : Fonctionne sur toutes les plateformes principales avec des connecteurs sur plusieurs langages de programmation (NodeJs, Ruby, C#, C++, Java, Perl, Python, PHP)
  - Redondance : BD peut être dupliquée sur plusieurs nœuds, donc charge réduite et disponibilité augmentée
  - Distribution : Possible, contrairement à la majorité des BD SQL (mais reste une opération complexe)
  - Verticalement évolutif : Augmenté la charge en ajoutant plus de CPU, RAM sur un serveur.
  - Basée sur la notion de table

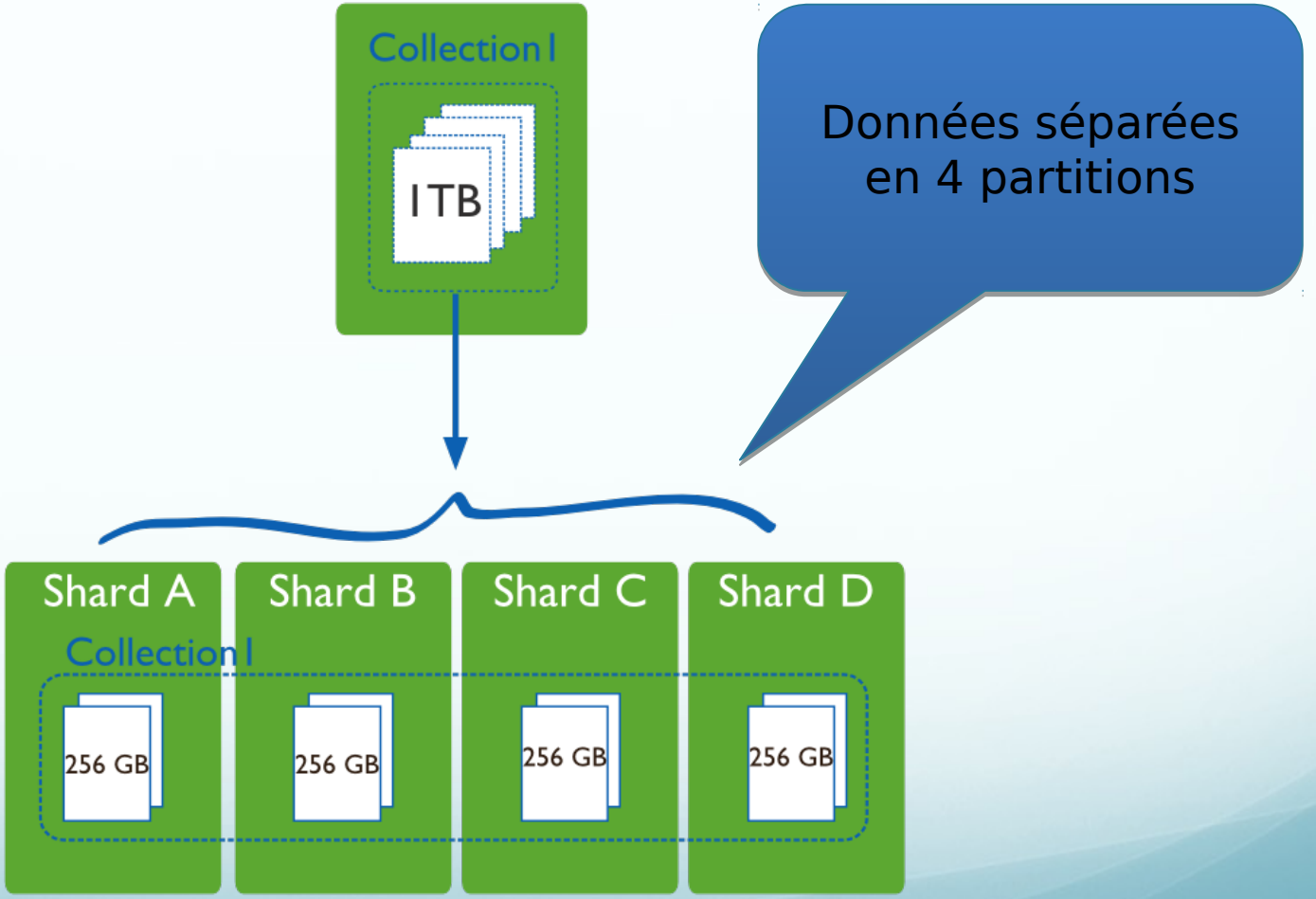
MongoDB vs MySQL

- MongoDB
  - Schéma dynamique : Possible de modifier le schéma d’une table sans modifier les données existantes
  - Évolutivité : MongoDB est horizontalement évolutif, on peut simplement ajouter des nouveaux nœuds et la charge est diminué, la redonance augmentée et on peut distribuer les données sur différents noeuds
  - Rapidité : Extrêmement performant pour des requêtes simples.
  - Basée sur la notion de documents

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 peuvent sur-utiliser 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.
  - Permet à chaque partition

MongoDB
Distribution des données

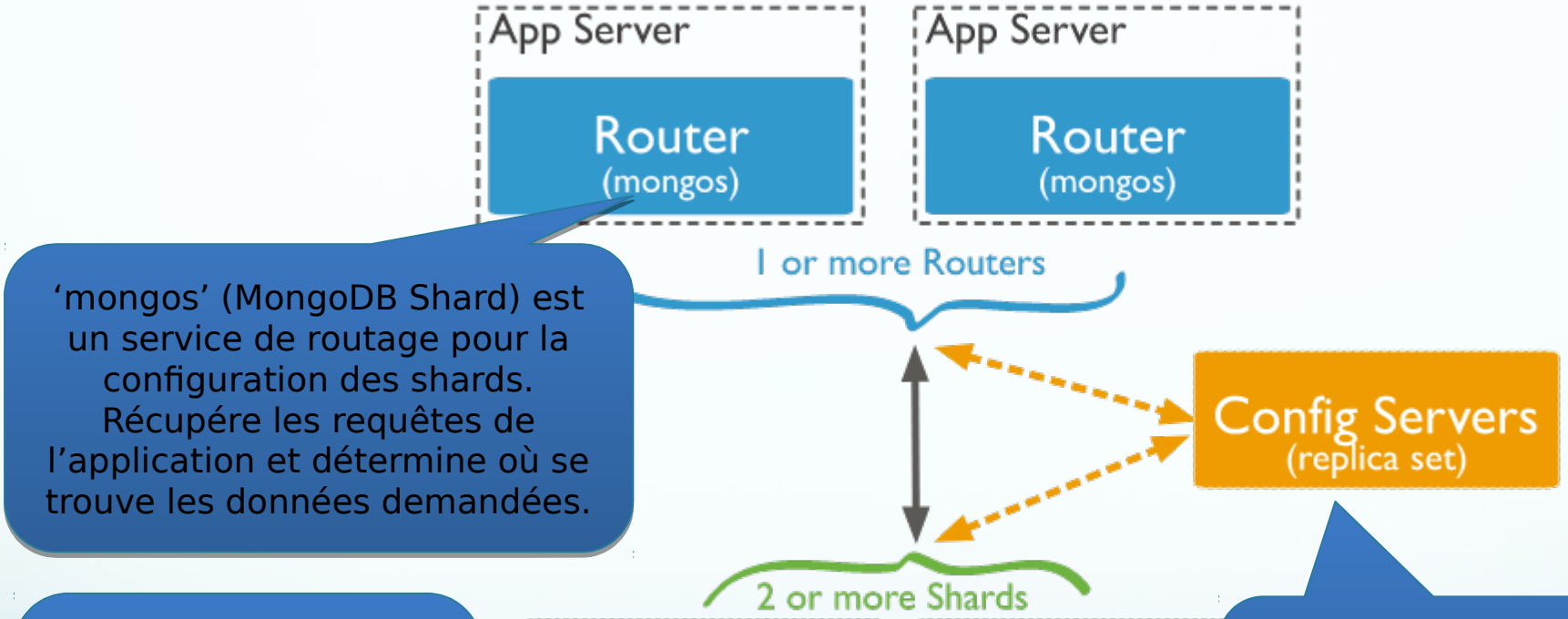


Extrait de http://docs.mongodb.org/manual/core/sharding-introduction/

MongoDB
Distribution de données

- Pour distribuer les documents d'une collection, MongoDB partionne la collection en utilisant un 'shard key'.
  - Cette clé est générée par MongoDB et se trouve dans chaque document d'une collection
- 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

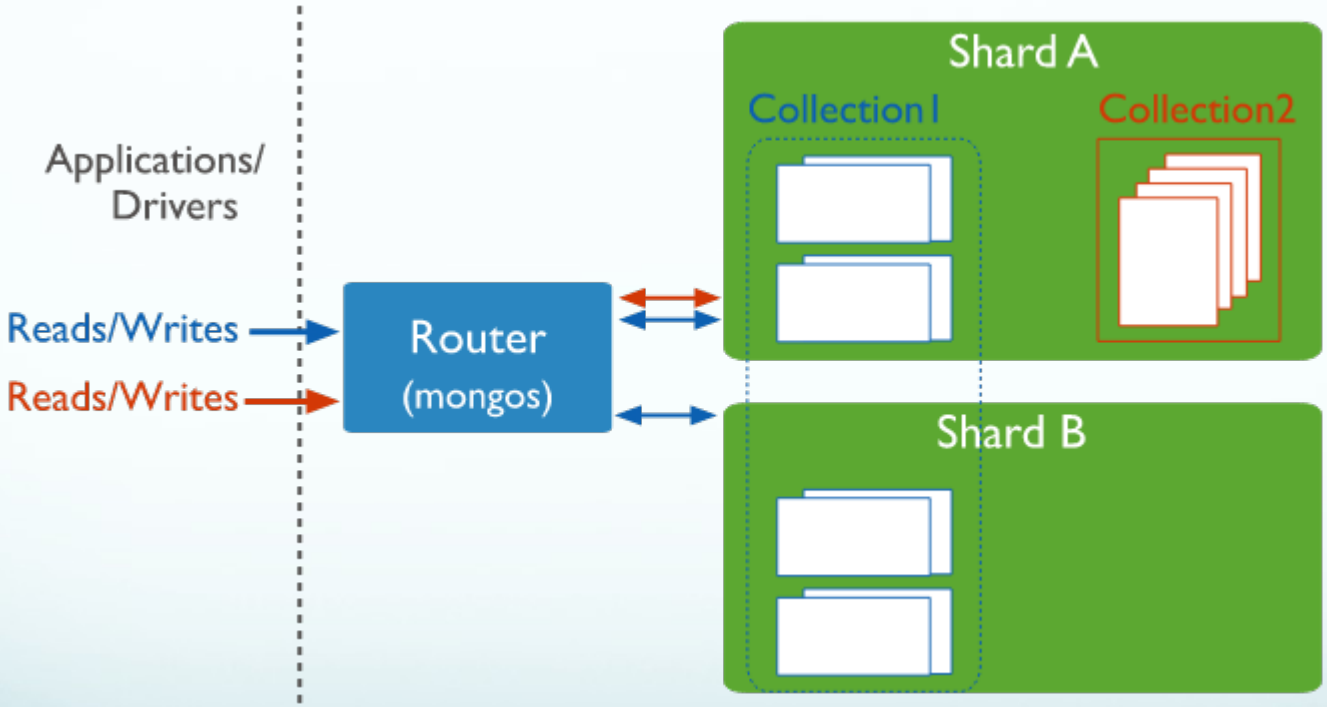
Contiennent les méta-données.
Sont consultés par le routeur pour savoir où envoyer la requête

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

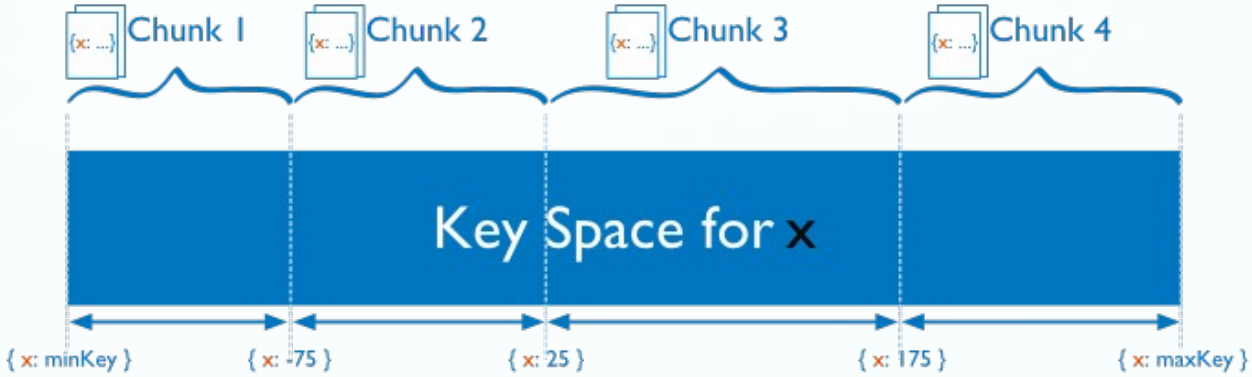
MongoDB
Distribution de données

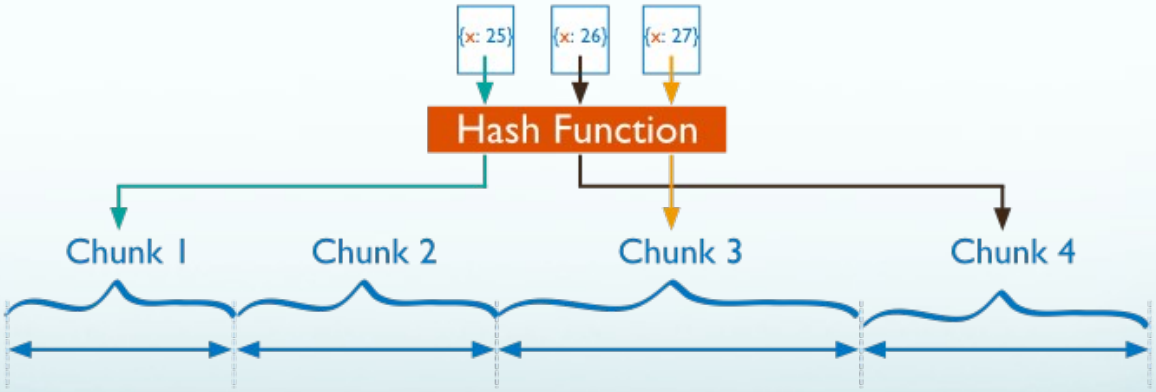
- MongoDB distribue les lectures/écritures sur toutes les partitions d'une grappe.
  - Permet à chaque partition de traiter une partie des données de façon parallèle.
- Si la taille des données augmente, on peut simplement augmenter le nombre de partition (donc augmenter le nombre de machines en conséquence)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

MongoDB
Distribution de données



MongoDB - Deux manières de partitionner

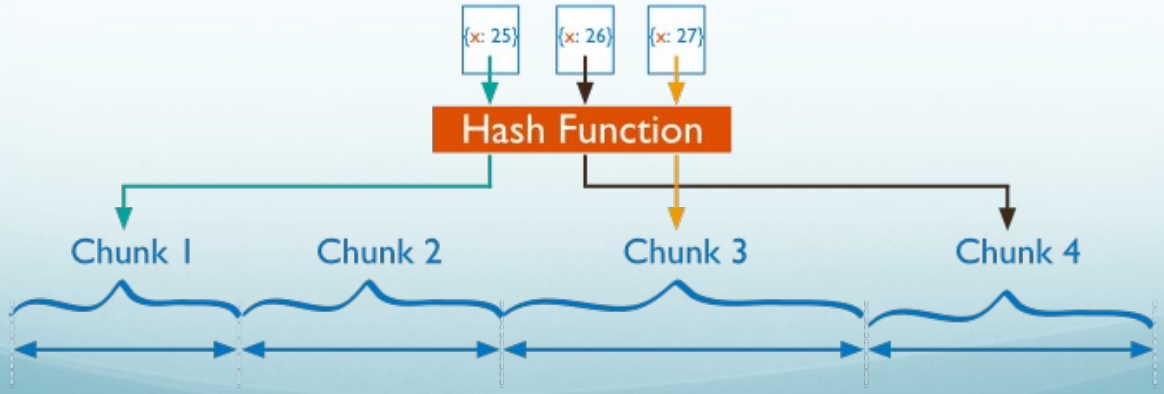




Extrait de http://docs.mongodb.org/manual/core/sharding-introduction/

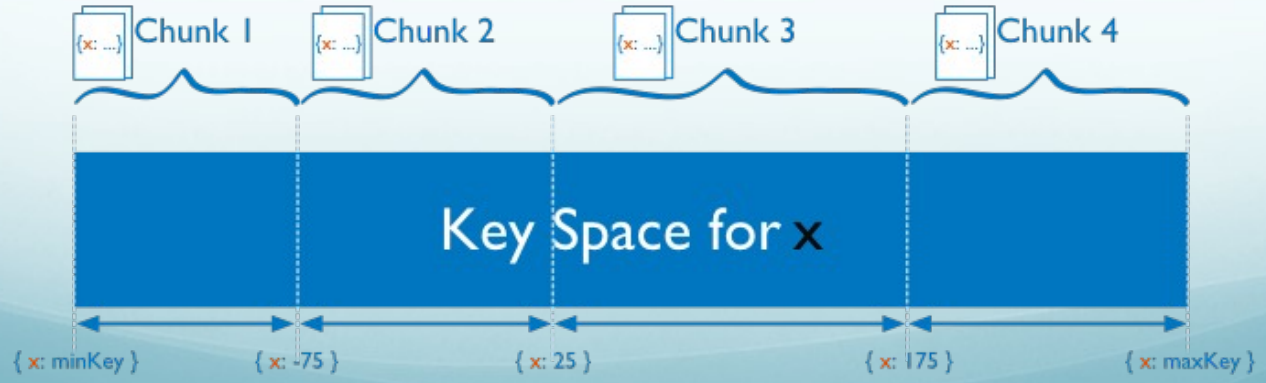
MongoDB
Paritionnement par clé hashée

- À partir du ‘shard key’ d’un document, MongoDB calcul une clé hashée.
- Chaque partition est donc assigné un interval basé sur les clé hashées
- Des ‘shard key’ proche ne seront problablement pas dans la même partition.
- Ce type de partitionnement facilite la distribution uniforme des données

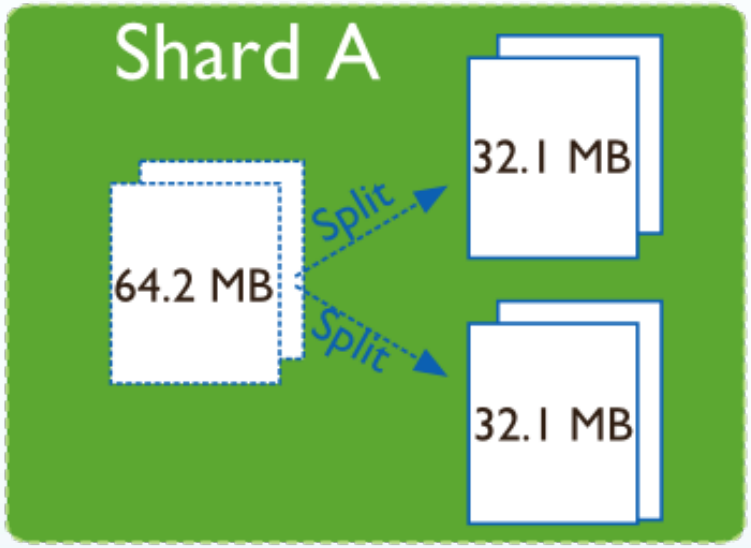


MongoDB
Paritionnement par interval

- On calcul des intervalles en fonctionne des ‘shard key’ de chaque document.
- Chaque partition est assignée une intervalle de valeur
- Les clés proches ont une forte chance de se retrouver dans une même partition
- Permet de mieux sibler les partitions qui risquent de contenir les données lors d’une lecture
- Une clé mal choisie risque de déquilibrer la partition des données



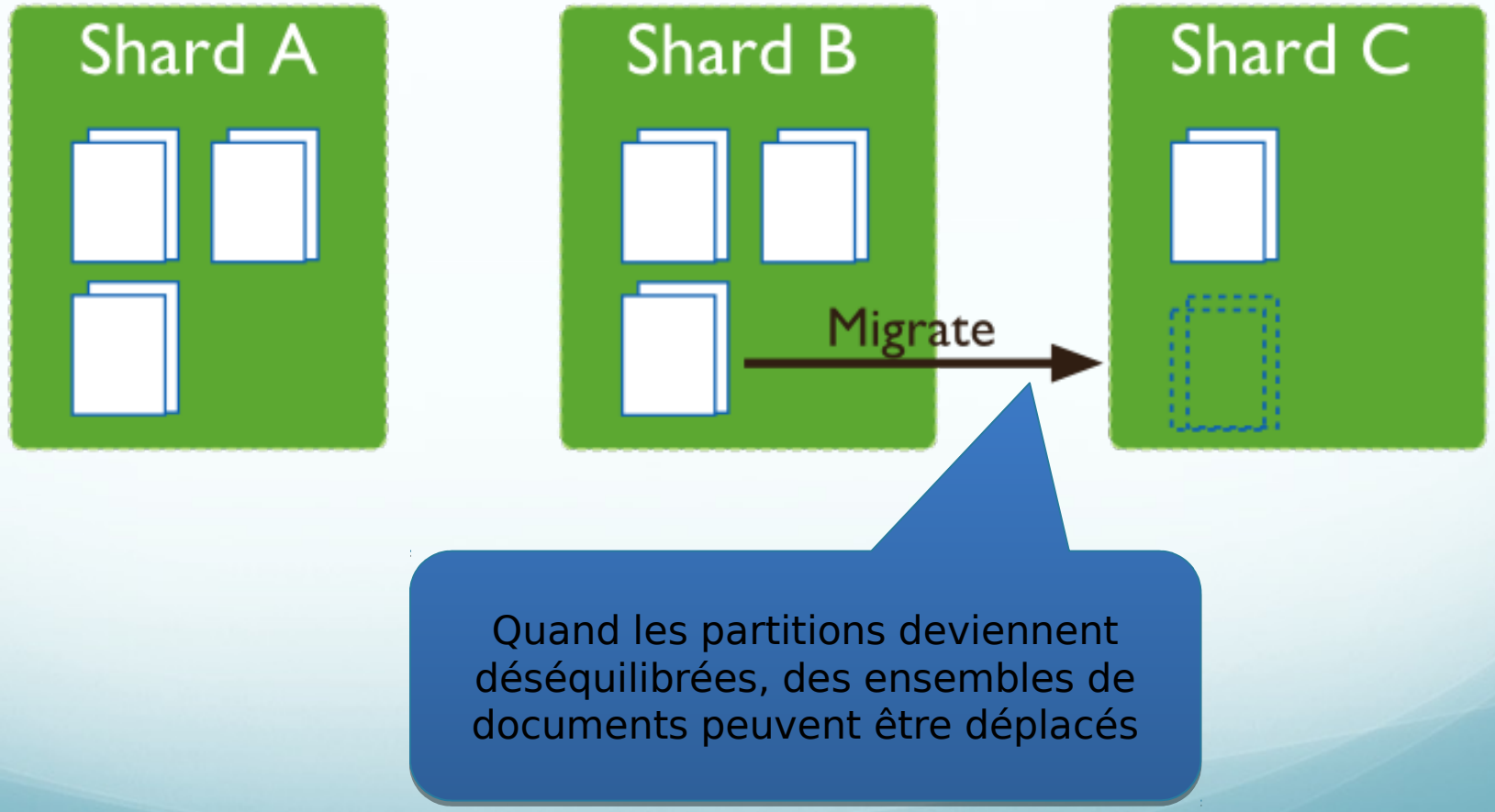
MongoDB - Partitionnement des groupes de données



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

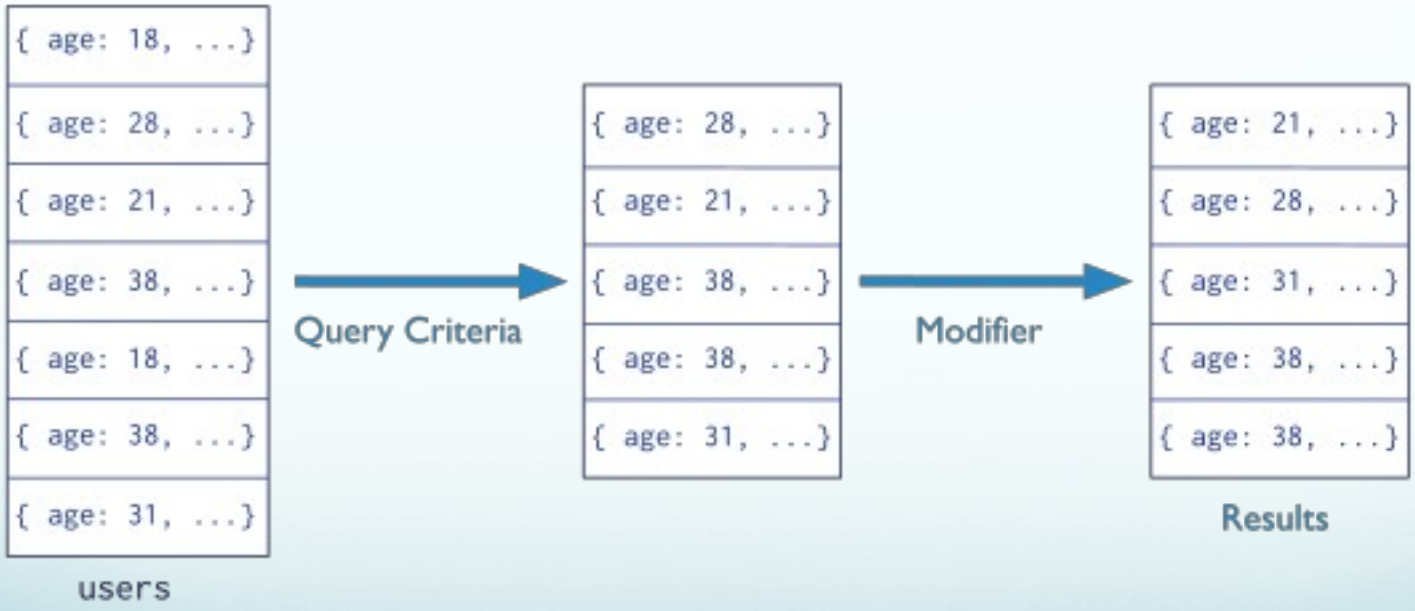
MongoDB Requête

- Méthodes **find()**, **findOne() et findById()**
- 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'idenfificateur est 5 ou "alsl1231"
- La méthode **find()** retourne un curseur (une sorte d'itérateur)

Exemple - Requête find() avec projection

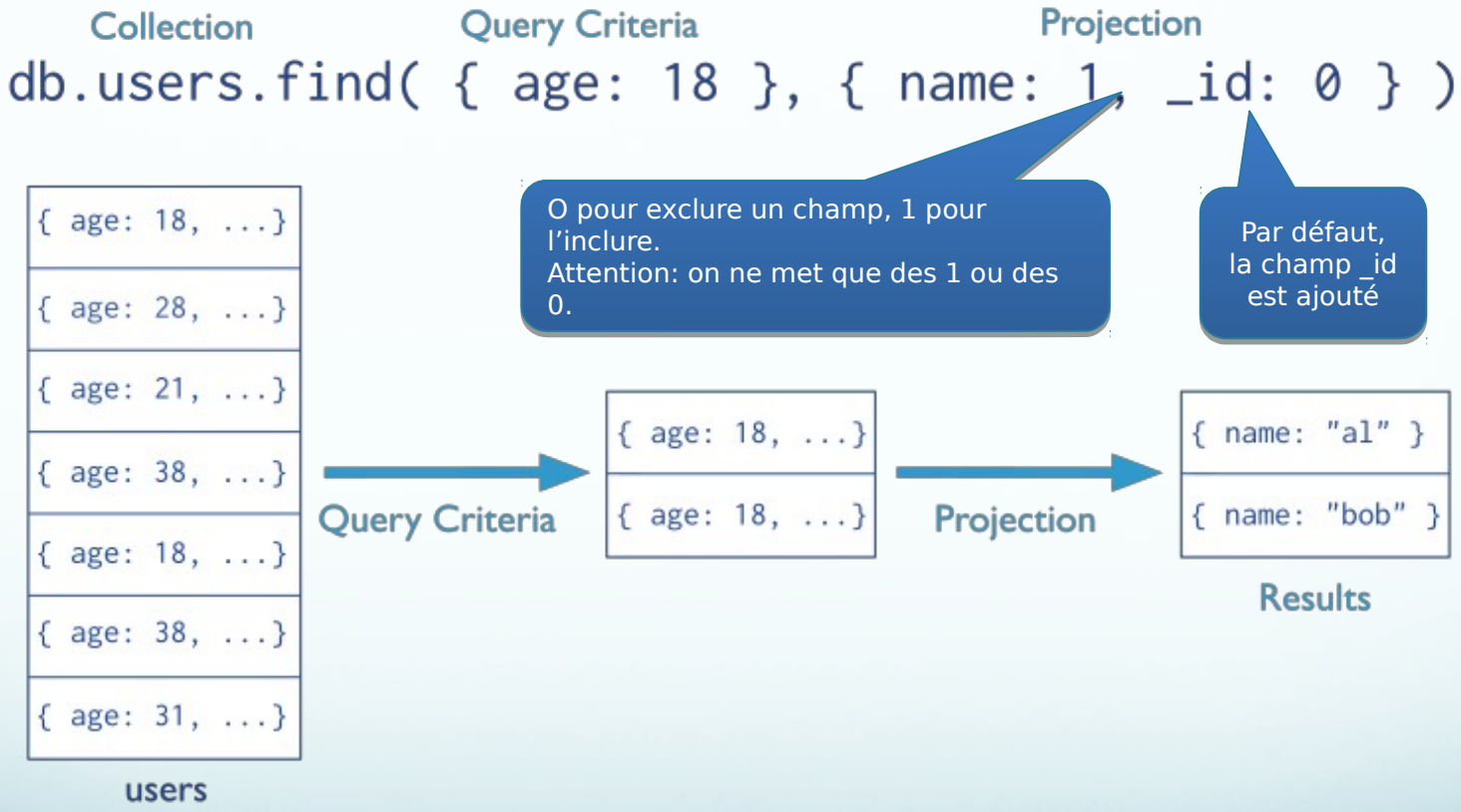
Collection — Query Criteria — Modifier

```
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

- Méthodes utiles:
  - next() et hasNext()
  - toArray()
  - forEach()
  - sort()
  - limit()
- Les méthodes peuvent être appelées en cascade:
  - **db.prod.find().sort({name: 1}).limit(5)**

MongoDB
Modification de données

- Méthodes **insertOne()** et **insertMany()**
  - Prend en argument le(s) document(s) à ajouter
  - 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 insérés