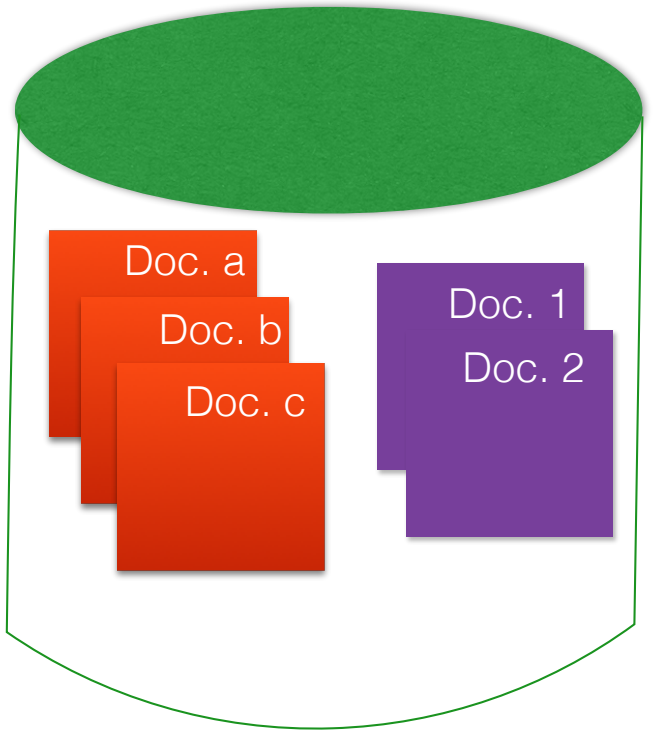
Bases de données orientées document

Introduction

- Au centre des BD documents:
  - des documents :-)
- Une BD document permet de stocker et retrouver des documents
- Document = notion abstraite
  - Ce ne sont pas des documents physiques!



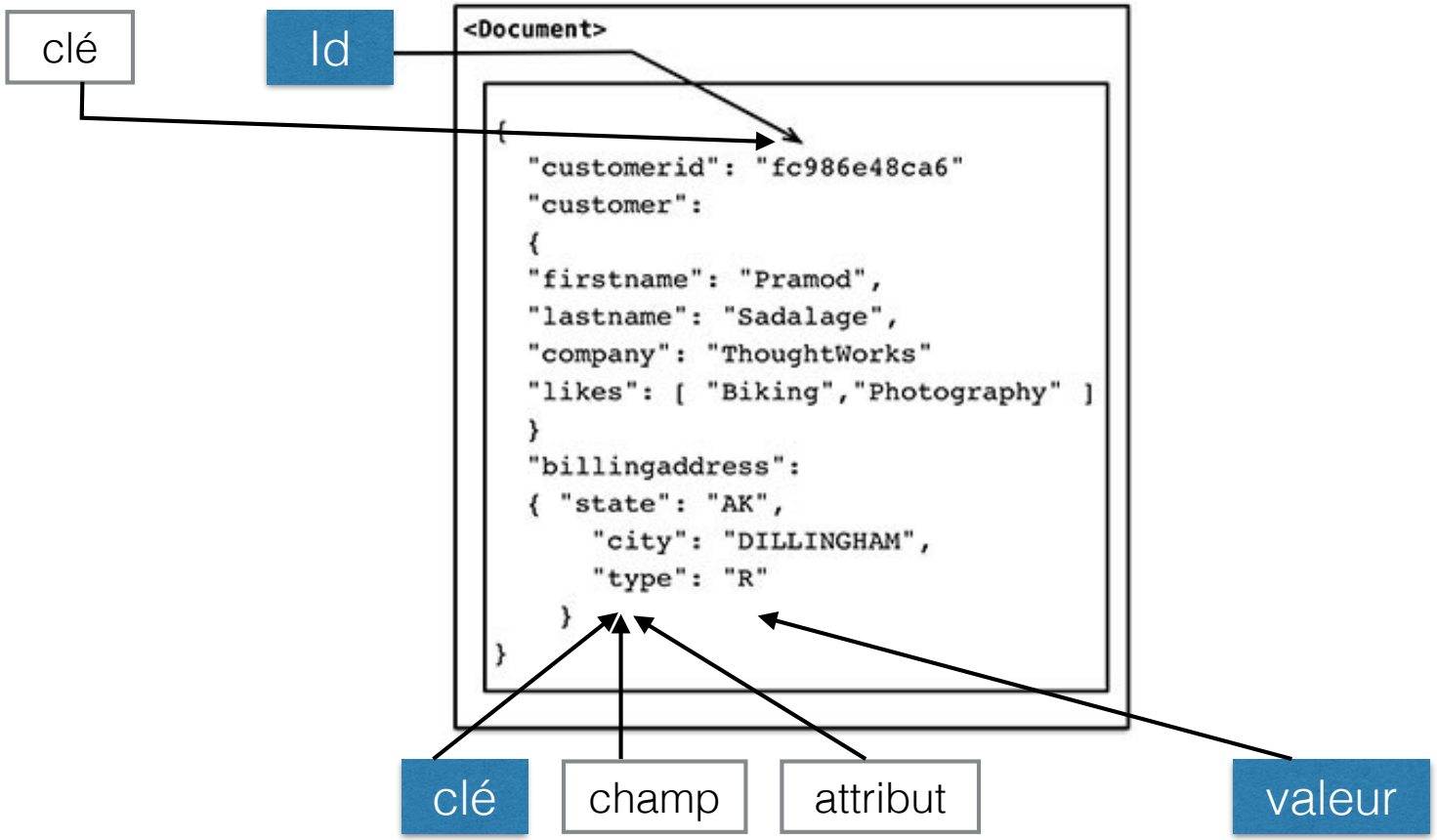
Introduction

La notion de document

- Les documents sont auto-descriptifs, avec une représentation arborescente des données
  - Le format JSON est le format privilégié
- Un document est composé de **clés** et de **valeurs** associées
  - Un document possède également un **identifiant**
- Les valeurs peuvent être requêtées
- Les valeurs sont soit :
  - d'un type simple (entier, chaîne de caractères, …)
  - elles-même composées de plusieurs couples clé/valeur.

Petit point vocabulaire

- En fonction des livres/sources/supports de cours:



Introduction

La notion de document

- Les documents n'ont pas de schéma
  - Deux instances de document dans la même base (=collection) peuvent avoir des clés différentes
  - Cela facilite le matching de documents avec des objets

```
{
    "name": "Pitarch",
    "function": "Associate
Professor",
     "age":30
 }
```

```
{
    "name": "Hubert",
    "function": "Professor",
     "status":"not available",
     "teaching":["RDB", "Python"]
}
```

[Le format JSON]

{...} : les accolades définissent un objet.

"language":"Java" : les guillemets (double-quotes) et les double-points définissent un couple clé/valeur (on parle de membre).

[...] : Les crochets définissent un tableau (ou array en anglais).

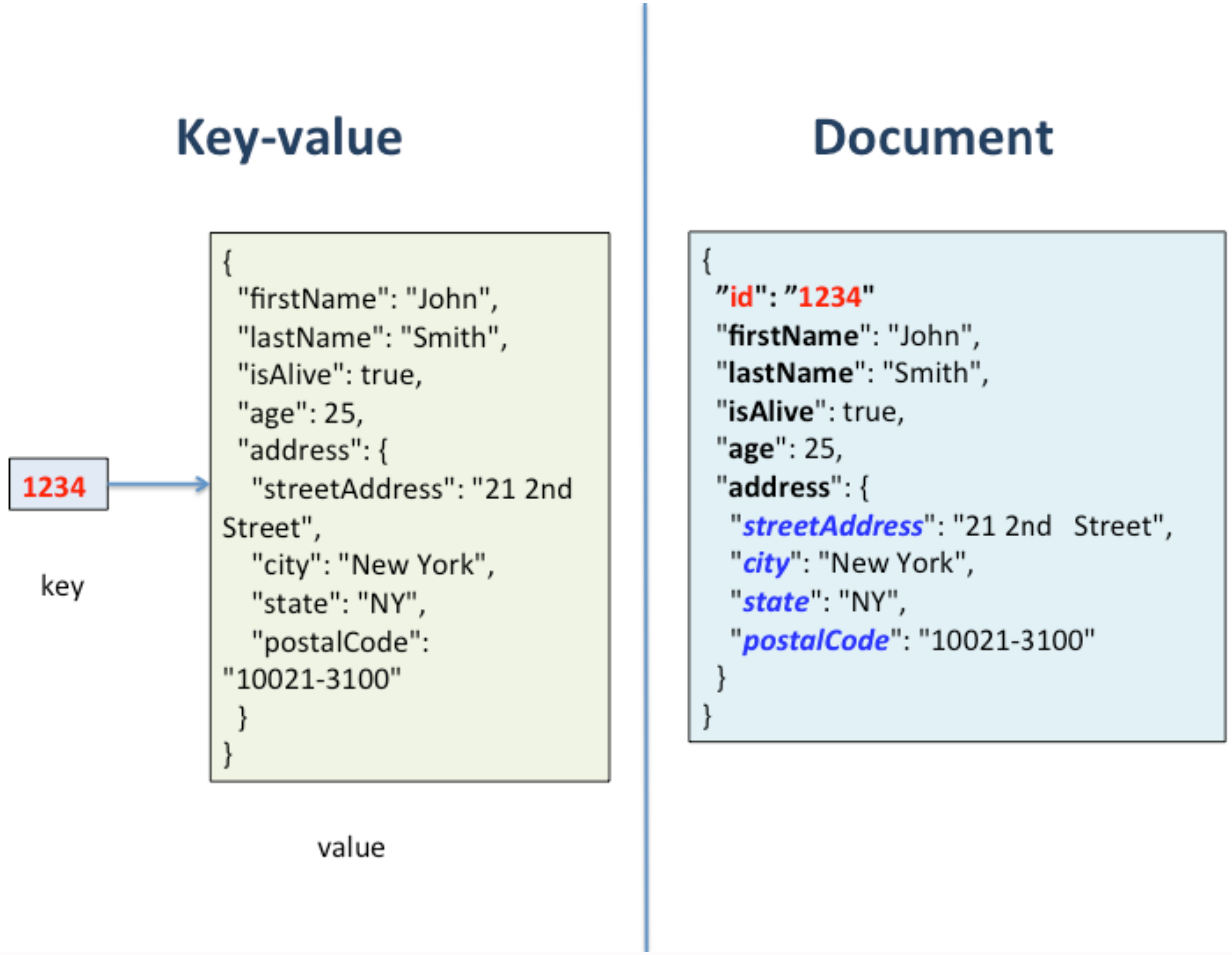
{"id":1, "language":"json", "author":"Douglas Crockford"} : Les virgules permettent de séparer les membres d'un tableau ou, comme ici, d'un objet.

```
{
    "titre_album" :"Abacab",
     "groupe" :"Genesis",
     "annee":1981,
     "genre" :"Rock"
 }
```

[Le format JSON]

```
{
  "fruits": [
    { "kiwis": 3,
      "mangues": 4
    },
    { "panier": true }
  ],
  "legumes": {
      "patates": "amandine",
      "poireaux": false
    },
    "viandes": ["poisson","poulet","boeuf"]
 }
```

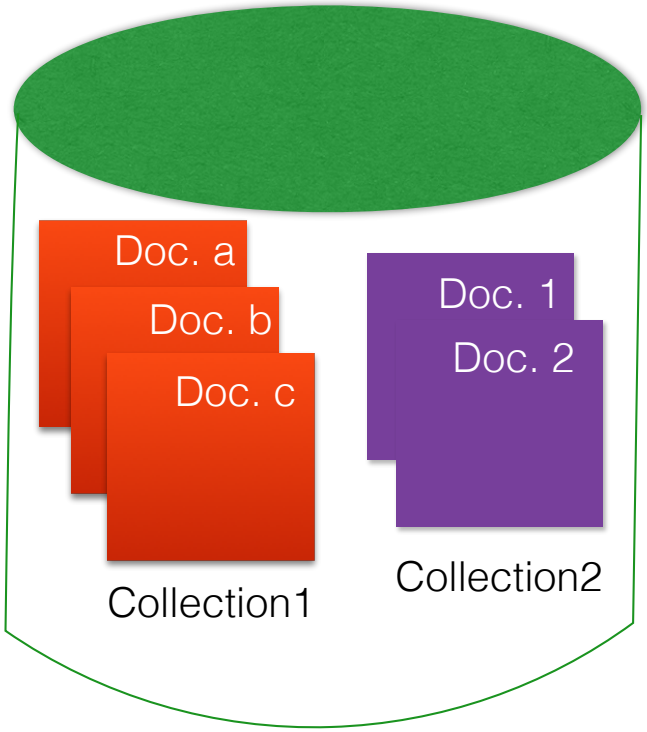
Introduction



Les BD documents peuvent être vues comme des BD clé-valeur où la valeur est examinable, c'est à dire que la valeur peut être requêtée.

Concepts généraux

- **Base de données** (~ *base de données* en relationnel)
  - Ensemble de collections
  - Espace de stockage
- **Collection** (~ *table* en relationnel)
  - Ensemble de documents qui partagent un objectif ou des similarités
  - Pas de schéma prédéfini
- **Document** (~ *ligne, tuple* en relationnel)
  - Un enregistrement dans une collection



Base de données

Concepts généraux

- Les BD orientées document supportent des opérations CRUD (comme les BD relationnelles!):
  - Creation (créations et insertions)
  - Retrieval (recherche)
  - Update (mises à jour)
  - Deletion (suppressions)

Avantages/inconvénients

- **Avantages des BD documents**
  - Modèle de données simple mais puissant
    - Grande flexibilité : facilité d'évolution de schéma
  - Bonne mise à l'échelle (surtout si sharding pris en charge)
  - Forte expressivité de requêtage
    - Recherche de documents basée sur leur contenu

- **Inconvénients**
  - Modèle de requête limité à des clés (et indexes)
    - Peut être lent pour les grandes requêtes
  - Inadapté pour des données interconnectées

Principaux domaines applicatifs

- Enregistrement d'évènements
- Systèmes de gestion de contenu (CMS)
- Web analytique ou analytique temps-réel
- Catalogue de produits
- …

Qui (quelques big names) ?

- BBC web application
- Ubuntu one (cloud storage and replication service pour Ubuntu)
- Doodle
- bit.ly
- NewYorkTimes
- Github
- …

Principaux systèmes

- CouchDB
  - stockage JSON, requêtes en Javascript via MapReduce

- MongoDB
  - Documents à la JSON, requêtes en Javascript

- Couchbase
  - Documents JSON

- ElasticSearch
  - Documents JSON

Focus

MongoDB

- SGBD OpenSource développé depuis 2007
  - Nom tiré de l'anglais humongous qui peut être traduit par « énorme »
- 4eme SGBD le plus populaire en juin 2016 (https://fr.wikipedia.org/wiki/MongoDB)
  - 1er SGBD noSQL
- Sharding et réplication
- Système CP (cohérence / résistance au morcellement)
- Journalisation
- Utilisation de MapReduce
- Utilisé par Foursquare, bit.ly, Doodle, expedia
  - (https://www.mongodb.com/who-uses-mongodb)

MongoDB

Documents

- MongoDB stocke l'information au format BSON (Binary JSON)
  - L'utilisateur lui voit du JSON

- Syntaxe d'un document en MongoDB
  - _id est un identifiant (généré ou manuel)
  - att-1 est une clé dont la valeur est une chaîne de caractères
  - att-2 est une clé dont la valeur est un entier
  - att-3 est une clé dont la valeur est une liste de valeurs
  - att-k est une clé dont la valeur est un document inclus

```
{
        _id: identifiant,
        "att-1": "val-1",
        "att-2": val2,
        "att-3": ["val-31",
        "val-32",...]
        …
        "att-k":{
            "att-k1": "val-k1",
             …
                }
 }
```

Durabilité des transactions

- Journalisation de type **write-ahead log**
  - Les instructions de création/modification/suppression des données sont écrites en mémoire (private view) puis dans un journal

- Les écritures sont inscrites sur le disque:
  - toutes les 60 secondes
  - à l'arrêt du serveur (et le journal est supprimé)

- Si mongoDB démarre et constate l'existence d'un journal, c'est que le moteur n'a pas été arrêté correctement
  - Phrase de récupération qui rejoue le journal

- Pour en savoir plus: http://blog.mongodb.org/post/33700094220/how-mongodbs-journaling-works

Gestion de la distribution

Concurrence

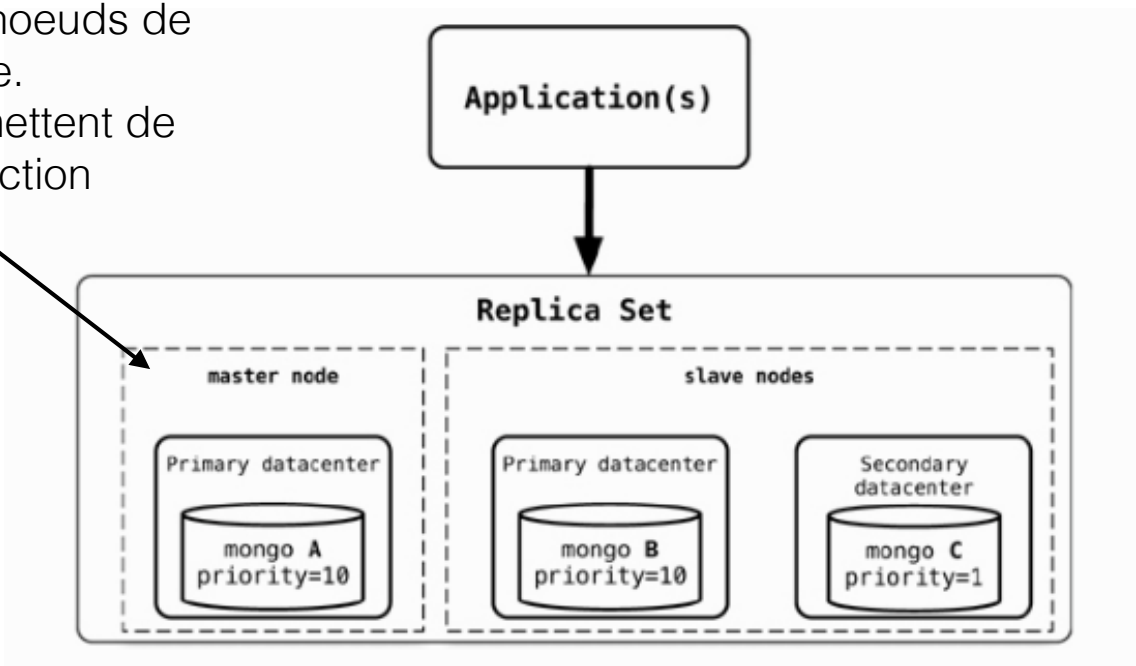
- Utilisation de verrous lecture/écriture pour beaucoup d'opérations
  - Plusieurs opérations de lecture possibles, mais une seule opération d'écriture à la fois.

Gestion de la distribution

Réplication

- Les données sont dupliquées selon une architecture maître-esclave
  - Une même donnée est accessible sur plusieurs noeuds et les données sont accessibles même si le maître ne répond plus.
- Concept de *r**eplica set*** (ensemble de réplication)

Elu par les autres noeuds de l'ensemble.
Les priorités permettent de « forcer »l'élection

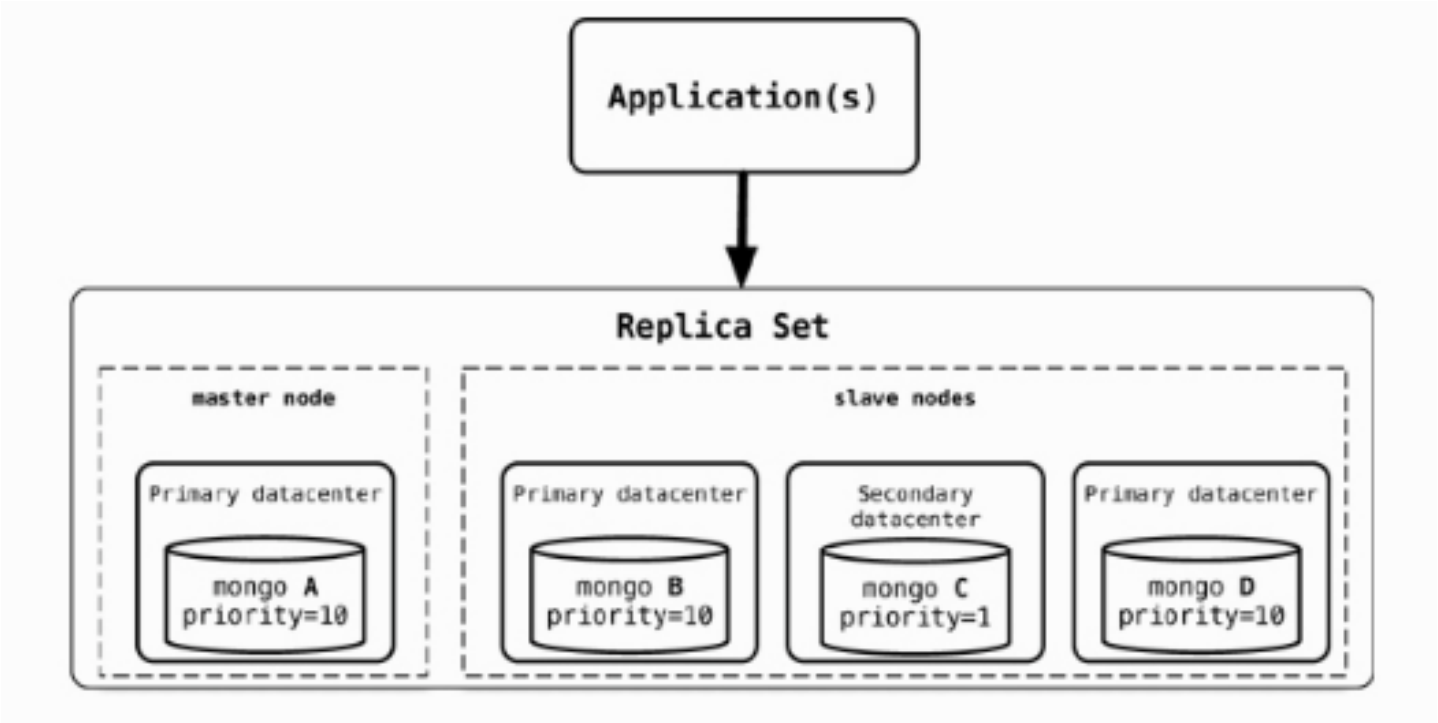


Source:Pramod J. Sadalage et al.

Gestion de la distribution

Scaling

- Montée en charge en lecture
  - Ajout d'un noeud dans le *replica set*

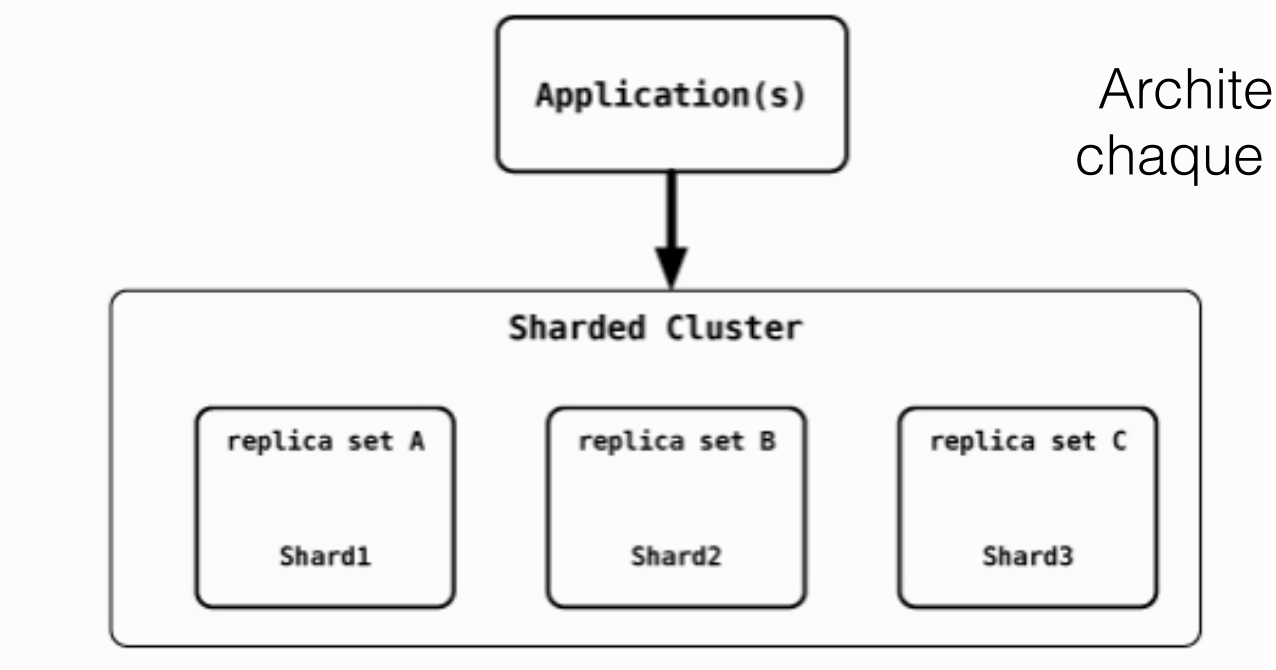


Source:Pramod J. Sadalage et al.

Gestion de la distribution

Scaling

- Montée en charge en écriture: sharding



Architecture dans laquelle chaque *shard* est un *replica set*.

- Pour en savoir plus: https://docs.mongodb.com/manual/sharding/

Source:Pramod J. Sadalage et al.

Modélisation des données

- Questions à se poser :
  - Quel est l'unité d'information (l'agrégat) de base à considérer pour mon application ?
  - Quelles sont les propriétés qui le définissent ? (clé/valeurs)?
  - y a-t'il des liens entre mes données ?

La modélisation choisie est très dépendante de l'application à mettre en place!

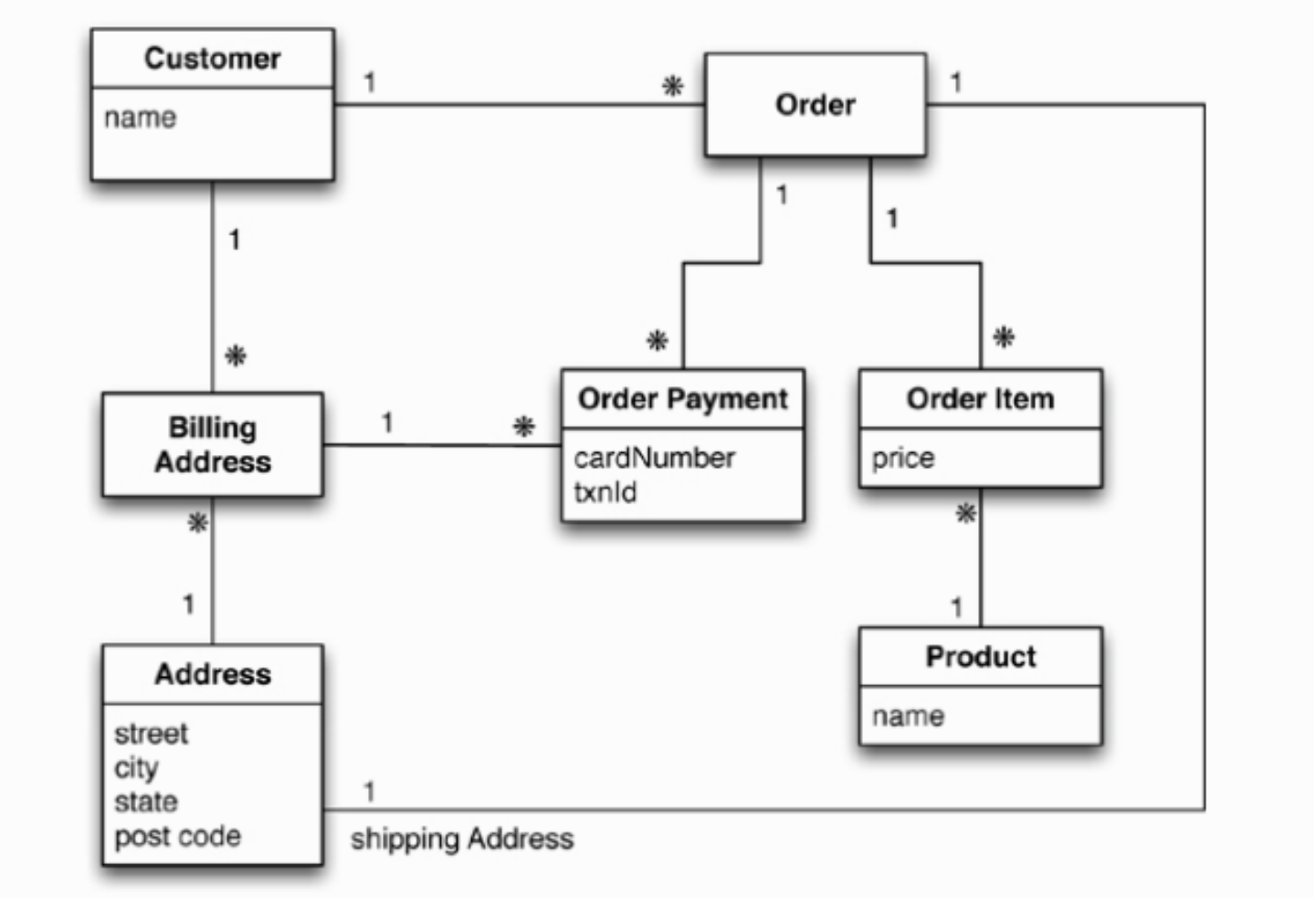
Modélisation des données

Liens

- Lorsque l'on doit modéliser des liens entre les données, 2 possibilités:
  - enchâssement (***embedding***)
    - Intégration de sous-documents
  - liaisons (***linking***)
    - gestion nécessaire du déférencement

Exemple d'application, rappel

Produits-Clients-Commande



Exemple d'application, rappel

Produits-Clients-Commande

En tant que comptable

Je veux connaître :

- les factures en un seul coup d'oeil
- les impayés et les clients associés
- les bons clients pour communication au service Client

De telle sorte à gérer efficacement mes comptes et les impayés

Modélisation des données

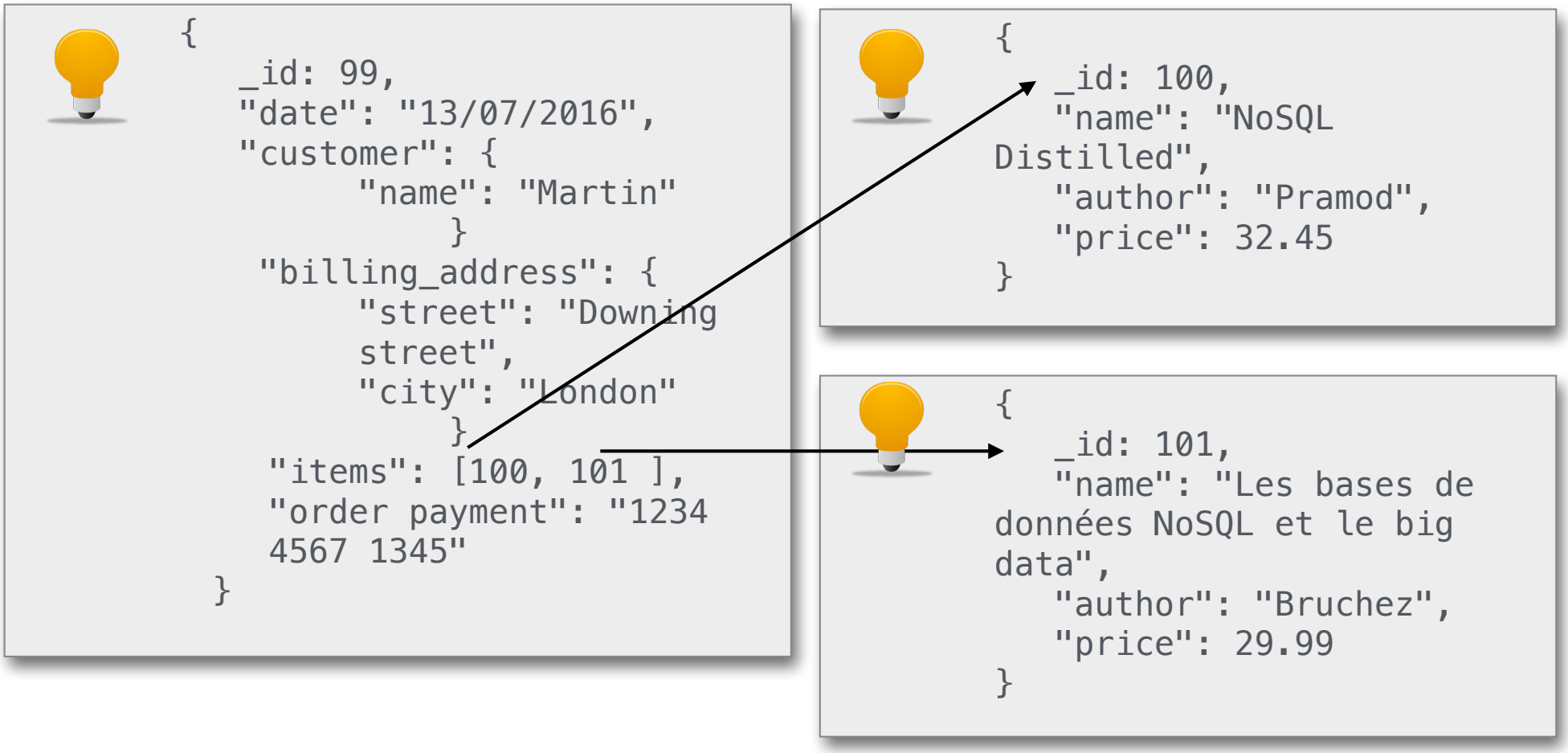
Embedding

```
{
    _id: 99,
    "date": "13/07/2016",
    "customer": {
        "name": "Martin"
            },
     "billing_address": {
        "street": "Downing street",
        "city": "London"
            },
     "items": [
        {
          "name": "NoSQL Distilled",
          "author":"Pramod",
          "price": 32.45
        }
        {
          "name": "Les bases de données NoSQL et le big
          data",
          "author":"Bruchez",
          "price": 29.99
        }
      ],
     "order payment": "1234 4567 1345"
}
```

Présent si facture payée

Modélisation des données

Linking



Modélisation des données

Embedding vs *linking*: que choisir ?

- Une gestion des relations au niveau du serveur est difficile à effectuer dans un environnement distribué

- => la voie conseillée est d'enchâsser les documents (si cela est possible!)
  - la duplication des données permet qui plus est de conserver un historique…
    - Dans notre exemple, le prix d'un livre restera enregistré pour chaque commande même s'il évolue dans le temps

Modélisation des données

Embedding vs *linking*: que choisir ?

Il est préférable de faire des ensembles d'ensembles d'ensembles, quitte à dupliquer beaucoup de données…

Taille maximale d'un document : 16 Mo

Opérations sur les documents

- Opérations possibles sur les documents:
  - insert
  - find
  - update
  - remove

- Toute opération sur un seul document est atomique
  - Il y a bien un verrouillage à l'écriture sur un document, mais il ne s'étend pas par défaut sur une mise à jour de plusieurs documents

- Par défaut, seuls des indexes sur les _id sont créés
  - Si pas d'index, lecture séquentielle des données

- Lors d'insertions et de mises à jours, la base de données et la collection sont automatiquement créées si elles n'existent pas

Opération `insert`

- Syntaxe pour insérer un document dans une collection *coll*:
  - **`<doc>`** est un document ou un tableau de documents à insérer
  - **`<option>`** est un document pouvant contenir:
    - un booléen ***`ordered`*** (insertion de plusieurs documents stoppée en cas d'erreur)
    - un document ***`writeConcern`***

```
db.coll.insert(
   <doc>,
   <options>
   )
```

Opération `insert`

- Concernant l'identifiant du document à insérer:
  - si la clé **`_id`** est présente dans le document, s'assurer de sa valeur
  - sinon MongoDB ajoute un identifiant automatiquement (type **`ObjectID`** sur 12 octets)

Les identifiants sont uniques pour une collection

```
db.orders.insert(
{
        _id: 99,
       "date": "13/07/2016",
       "customer": {
         "name": "Martin"
             },
        "billing_address": {
         "street": "Downing
         street",
         "city": "London"
             },
     "items": [
        {
         "name": "NoSQL Distilled",
         "author":"Pramod",
         "price": 32.45
        },
        {
         "name": "Les bases de
         données NoSQL et le big
         data",
         "author":"Bruchez",
         "price": 29.99
        }
      ]
} )
```

document

Opération `insert`

Insertion de plusieurs documents

```
db.orders.insert(
[
{
    _id: 1,
    name: "L'île au trésor",
    category:["novel","adventure"]
},
{
    _id: 2,
    name: "L'écume des jours",
    category:["novel"]
},
{
    _id: 3,
    name: "Vingt mille lieux sous les mers",
    category:["adventure", "novel"]
},
 {
    _id: 4,
    name: "Dune",
    category:["SF", "novel"]
}
 ]
 )
```

Opération `insert`

`writeConcern`

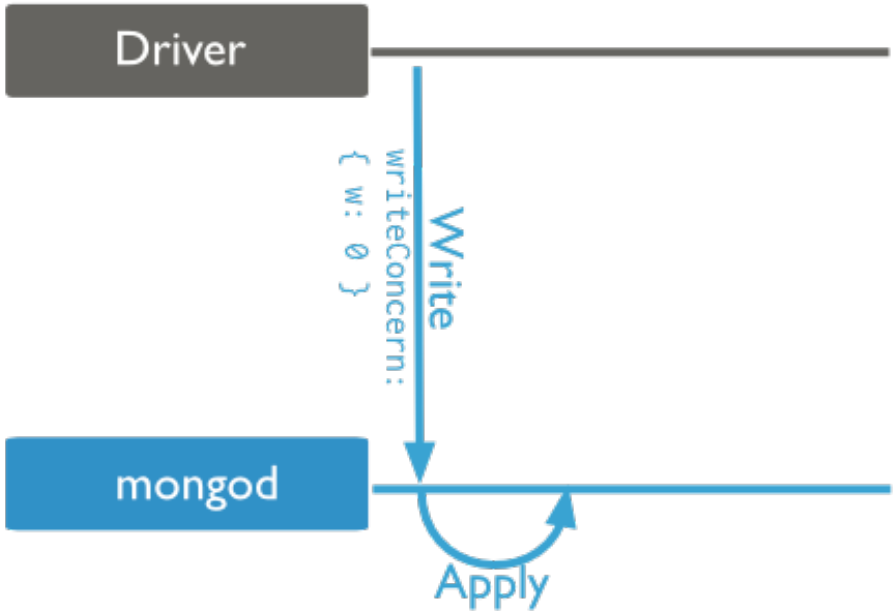
- Le document optionnel **`writeConcern`** décrit la garantie du succès d'une opération avec
  - **`w`**, nombre de confirmations d'écriture (0,1, 2, « majority »…)
  - **`j`** booléen indiquant l'écriture dans le journal
  - **`wtimeout`**, limite de temps en ms quand w>1

- Par défaut: w=1, j=true

Opération `insert`

`writeConcern`

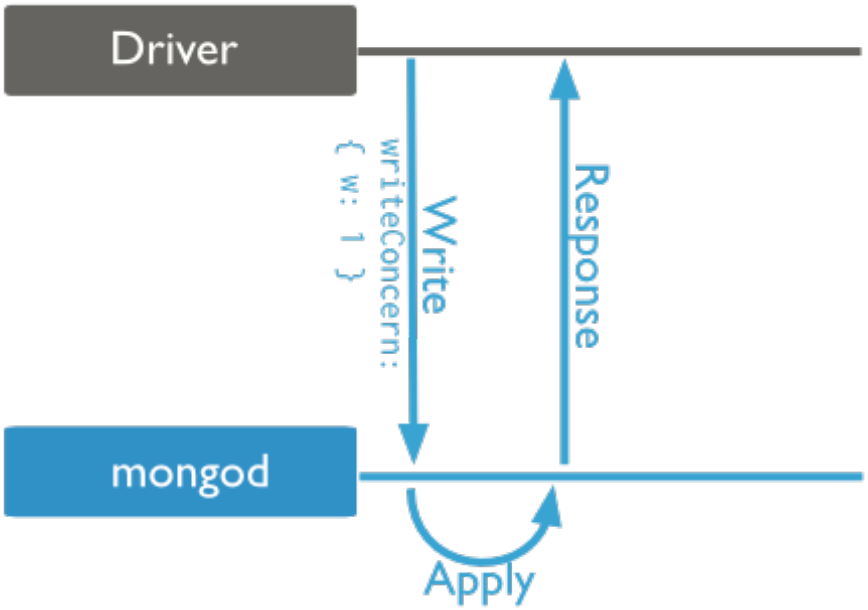
- w=0: pas d'accusé de réception



Opération `insert`

`writeConcern`

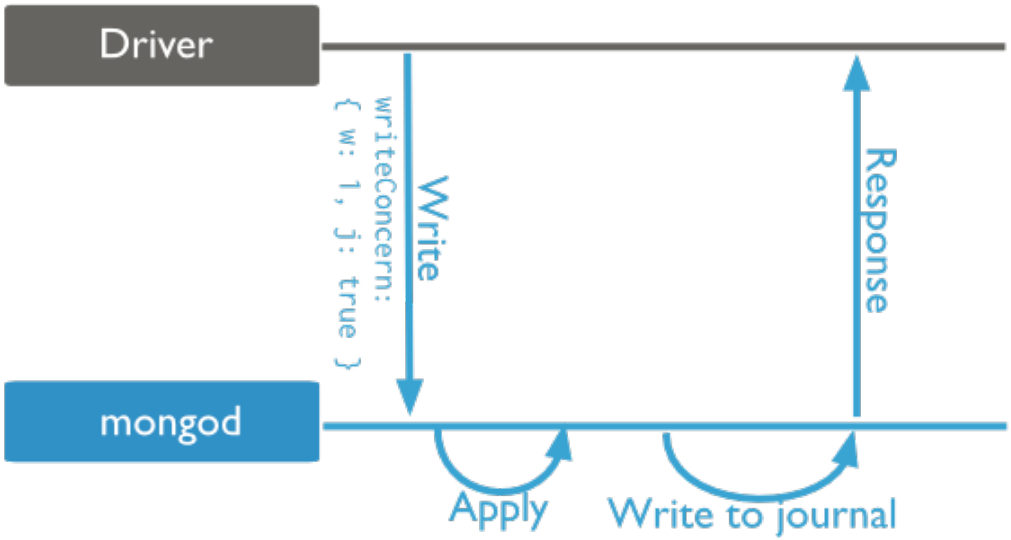
- w=1: accusé de réception



Opération `insert`

`writeConcern`

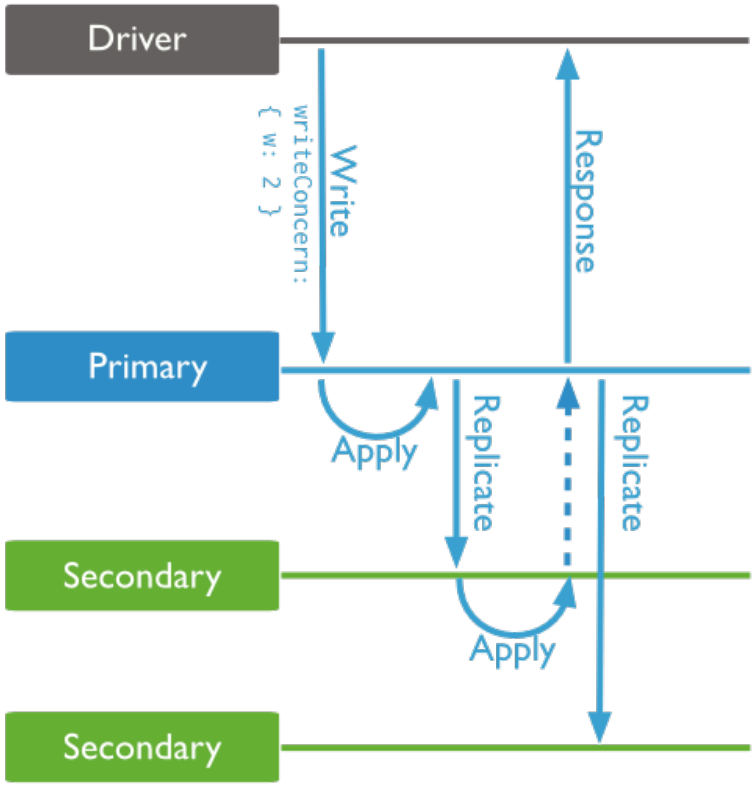
- w=1, j=true: accusé de réception une fois les données écrites dans le journal



Opération `insert`

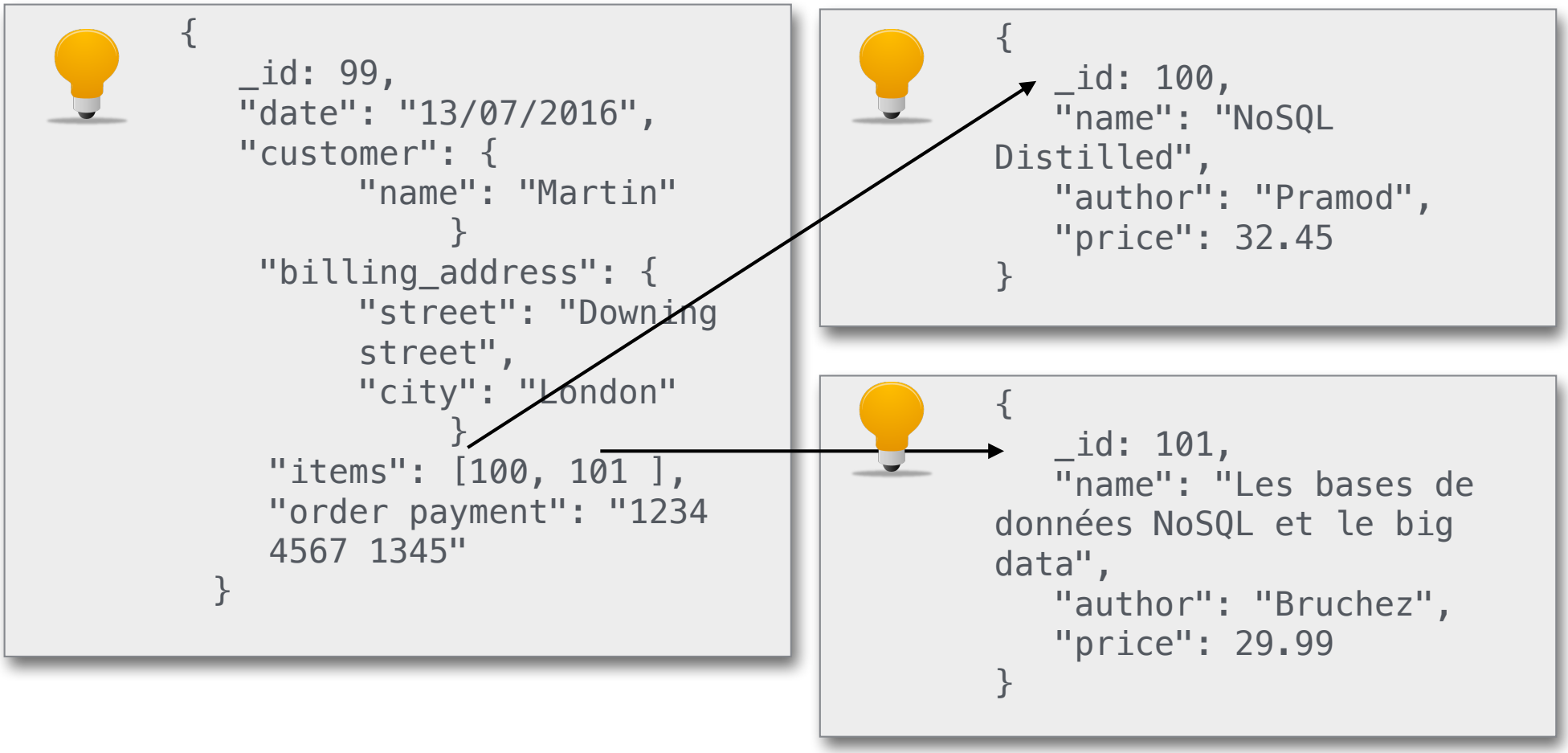
`writeConcern`

- w=2: accusé de réception après 2 écritures



Gestion des références

Solution 1: De façon manuelle



Gestion des références

Solution 2: Avec DBref

- DBrefs est une convention pour représenter un document indiquant un lien/une référence d'un document vers un autre

- Permet de lier des documents de collections différentes
- DBref est géré par de nombreux drivers
- Syntaxe:
  - `$ref:` collection du document référencé
  - `$id`: valeur d'_id du document référencé
  - `$db` (optionnel): BD du document référencé

```
{ "$ref" : <value>, "$id" : <value>, "$db" : <value> }
```

Gestion des références

Solution 2: Avec DBref

Opération `find`

- Syntaxe pour rechercher des documents dans une collection *coll*:
  - `<criteres>` est un document contenant les critères de sélection des documents
  - `<projection>` est un document contenant les clés (champs) qui seront présents dans les documents résultats

```
db.coll.find(
   <criteres>,
   <projection>
   )
```

Opération `find`

Critères

- Le document <criteres> permet de:
  - retourner tous les documents d'une collection

```
db.coll.find( {} )
```

  - retourner des documents dont la valeur d'une clé est égale à une constante

```
db.coll.find( {clé1:<const>} )
```

  - retourner des documents en utilisant des opérateurs

```
db.coll.find( {clé1:<opérateur>, …,
clék:<operateur>} )
```

Opération `find`

Opérateurs

- Comparaison de valeur (`$ne, $gt, $gte, $lt, $lte`):

Documents avec un prix supérieur à 10:

```
db.coll.find( {price:{$gt:10}} )
```

Documents avec un prix inférieur ou égal à 25:

```
db.coll.find( {price:{$lte:25}} )
```

Documents avec un prix différent de 50:

```
db.coll.find( {price:{$ne:50}} )
```

- Comparaison avec un tableau de valeurs (`$in, $nin`)

Documents avec un prix compris dans l'ensemble 10, 20, 30

```
db.coll.find( {price:{$in: [10, 20, 30]}} )
```

Opération `find`

Opérateurs

- Comparaison logique (`$and, $or, $not`):

Documents avec un prix à 10 et une quantité supérieure à 1:

```
db.coll.find( {$and:[{price:{$gt:10}, {quantity:{$gt:1}}]} )
```

Documents avec un prix qui ne soit pas supérieur à 25:

```
db.coll.find( {price:{$not{$gt:25}}} )
```

- Existence ou type d'un élément (`$exists, $type`)

Documents avec un champ prix:

```
db.coll.find( {price:{$exists: true}} )
```

Opération `find`

Opérateurs - Recherche sur le texte

- MongoDB permet des recherches plein texte sur n'importe quelle clé contenant du texte ou un tableau de texte, à condition qu'un index soit créé sur le texte

```
db.coll.createIndex( { name: "text", description: "text" } )
```

- `$search, $regex, $text`:

Documents dont l'un des textes contient NoSQL ou MongoDB:

```
db.coll.find( {$text:{$search: "NoSQL MongoDB"}} )
```

Documents avec une clé name contenant SQL:

```
db.coll.find( {name:{$regex:'SQL'}} )
```

Idem mais avec option insensible à la casse (i)

```
db.coll.find( {name:{$regex:'SQL', $options:'i'}} )
```

Opération `find`

Accès à des sous-documents

```
db.orders.insert(
{
     _id: 99,
    "date": "13/07/2016",
     "customer": {
          "name": "Martin"
              },
     "billing_address": {
          "street": "Downing street",
          "city": "London"
              },
     "items": [
        {
          "name": "NoSQL Distilled",
          "author":"Pramod",
          "price": 32.45
        },
        {
          "name": "Les bases de données
          NoSQL et le big data",
          "author":"Bruchez",
          "price": 29.99
        }
      ]
  } )
```

Documents qui ont un sous-document `customer` contenant une clé `name` ayant pour valeur « `Martin` »

```
db.orders.find({"customer.name":"Martin"})
```

Opération `find`

Accès à des tableaux

```
db.orders.insert(
[{
    _id: 1,
    name: "L'île au trésor",
    category:["novel","adventure"]
},
{
    _id: 2,
    name: "L'écume des jours",
    category:["novel"]
},
{
    _id: 3,
    name: "Vingt mille lieux sous les mers",
    category:["adventure", "novel"]
},
 {
    _id: 4,
    name: "Dune",
    category:["SF", "novel"]
},
{
    _id: 5,
    name: "Le seigneur des anneaux",
    category:["novel",
"adventure","fantasy"]
} ]
 )
```

Documents contenant un tableau `category` avec exactement les valeurs `novel` et `adventure`

```
db.orders.find( { category: [ "novel",
"adventure" ] } )
```

```
{
    _id: 1,
    name: "L'île au trésor",
    category:["novel","adventure"]
}
```

Opération `find`

Accès à des tableaux

```
db.orders.insert(
[{
    _id: 1,
    name: "L'île au trésor",
    category:["novel","adventure"]
},
{
    _id: 2,
    name: "L'écume des jours",
    category:["novel"]
},
{
    _id: 3,
    name: "Vingt mille lieux sous les mers",
    category:["adventure", "novel"]
},
 {
    _id: 4,
    name: "Dune",
    category:["SF", "novel"]
},
{
    _id: 5,
    name: "Le seigneur des anneaux",
    category:["novel", "adventure","fantasy"]
} ]
 )
```

Documents contenant un tableau `category` avec l'élément `novel`

```
db.orders.find( { category:  "novel" } )
```

Tous les documents !

Opération `find`

Accès à des tableaux

```
db.orders.insert(
[{
    _id: 1,
    name: "L'île au trésor",
    category:["novel","adventure"]
},
{
    _id: 2,
    name: "L'écume des jours",
    category:["novel"]
},
{
    _id: 3,
    name: "Vingt mille lieux sous les mers",
    category:["adventure", "novel"]
},
 {
    _id: 4,
    name: "Dune",
    category:["SF", "novel"]
},
{
    _id: 5,
    name: "Le seigneur des anneaux",
    category:["novel",
"adventure","fantasy"]
} ]
 )
```

Documents contenant un tableau `category` avec pour première valeur `novel`

```
db.orders.find({ "category.0": "novel"})
```

```
{
    _id: 1,
    name: "L'île au trésor",
    category:["novel","adventure"]
}
{
    _id: 2,
    name: "L'écume des jours",
    category:["novel"]
}
{
    _id: 5,
    name: "Le seigneur des anneaux",
    category:["novel", "adventure","fantasy"]
}
```

Opération `find`

Accès à des tableaux

```
db.orders.insert(
[{
    _id: 1,
    name: "L'île au trésor",
    prices:[10,11,15]
},
{
    _id: 2,
    name: "L'écume des jours",
    prices:[13]
},
{
    _id: 3,
    name: "Vingt mille lieux sous les mers",
    prices:[20,7]
},
 {
    _id: 4,
    name: "Dune",
    prices:[60,12]
},
{
    _id: 5,
    name: "Le seigneur des anneaux",
    prices:[15,17]
} ]
 )
```

Documents contenant un tableau `prices` avec au moins un élément >10 et <14

```
db.orders.find(
   {prices:{$elemMatch:{$gt:10, $lt:14}}})
```

```
{
    _id: 1,
    name: "L'île au trésor",
    prices:[10,11,15]
}
{
    _id: 2,
    name: "L'écume des jours",
    prices:[13]
}
{
    _id: 4,
    name: "Dune",
    prices:[60,12]
}
```

Opération `find`

Accès à des tableaux

```
db.orders.insert(
[{
    _id: 1,
    name: "L'île au trésor",
    prices:[10,11,15]
},
{
    _id: 2,
    name: "L'écume des jours",
    prices:[13]
},
{
    _id: 3,
    name: "Vingt mille lieux sous les mers",
    prices:[20,7]
},
 {
    _id: 4,
    name: "Dune",
    prices:[60,12]
},
{
    _id: 5,
    name: "Le seigneur des anneaux",
    prices:[15,17]
} ]
 )
```

Documents contenant un tableau `prices` avec au moins un élément >10 et un autre <14

```
db.orders.find({prices:{$gt:10, $lt:14}})
```

Documents 1, 2, 3 et 4

Opération `find`

Accès à des tableaux contenant des sous-documents

```
 db.orders.insert(
[{
    _id: 1,
    name: "L'île au trésor",
    prices:[{store:"amazon", price:10},
{store:"fnac", price:11},{store:"other", price:15}]
},
{
    _id: 2,
    name: "L'écume des jours",
    prices:[{store:"amazon", price:13}]
},
{
    _id: 3,
    name: "Vingt mille lieux sous les mers",
    prices:[{store:"amazon", price:20},
{sotre:"fnac", price:7}]
},
 {
    _id: 4,
    name: "Dune",
    prices:[{store:"other", price:60},{store:"fnac",
price:12}]
},
{
    _id: 5,
    name: "Le seigneur des anneaux",
    prices:[{store:"fnac", price:15},
{store:"amazon", price:17}]
} ]
 )
```

Documents dont le premier document du tableau `prices` a une clé `price` <=15

```
db.orders.find({"prices.0.price":
{$lte:15}})
```

```
{
    _id: 1,
    name: "L'île au trésor",
    prices:[{store:"amazon", price:10},
{store:"fnac", price:11},{store:"other",
price:15}]
}
{
    _id: 2,
    name: "L'écume des jours",
    prices:[{store:"amazon", price:13}]
}
{
    _id: 5,
    name: "Le seigneur des anneaux",
    prices:[{store:"fnac", price:15},
{store:"amazon", price:17}]
}
```

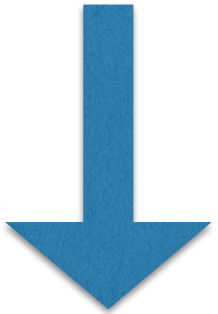
Opération `find`

Accès à des tableaux contenant des sous-documents

```
 db.orders.insert(
[{
    _id: 1,
    name: "L'île au trésor",
    prices:[{store:"amazon", price:10},
{store:"fnac", price:11},{store:"other", price:15}]
},
{
    _id: 2,
    name: "L'écume des jours",
    prices:[{store:"amazon", price:13}]
},
{
    _id: 3,
    name: "Vingt mille lieux sous les mers",
    prices:[{store:"amazon", price:20},
{sotre:"fnac", price:7}]
},
 {
    _id: 4,
    name: "Dune",
    prices:[{store:"other", price:60},{store:"fnac",
price:12}]
},
{
    _id: 5,
    name: "Le seigneur des anneaux",
    prices:[{store:"fnac", price:15},
{store:"amazon", price:17}]
} ]
 )
```

Documents dont un document du tableau `prices` a une clé `price` <=15

```
db.orders.find({"prices.price":{$lte:15}})
```



Tous les documents !

Opération `find`

Projection

- Le document `<projection>`:
  - contient soit une liste de champs projetés, soit une liste de champs exclus
  - Syntaxe:

```
{clé1:<boolean>, …, clék:<boolean>}
```

  - La valeur du booléen détermine la projection du champs
    - 1/true = champ projeté (dans les documents résultats)
    - 0/false = champ non projeté (exclu)
  - champ "_id" inclus par défaut, mais possibilité de l'exclure

Opération `find`

Projection

- Il existe des opérateurs de projection, notamment pour les tableaux:
  - `$` et `$elemMatch` permettent de projeter le contenu d'un tableau répondant à une condition de sélection.
  - `$slice` permet de récupérer une partie spécifique d'un tableau.

https://docs.mongodb.com/manual/reference/operator/projection/

Opération `find`

Projection

Tous les champs sauf price et name:

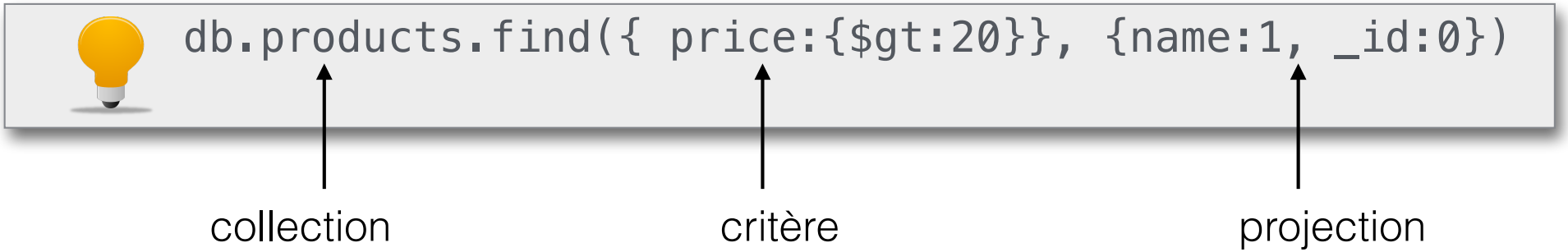
```
{price:false, name:false}
```

Uniquement le champ price:

```
{price:1, _id:0}
```

Opération `find`

Exemple complet



Retourne tous les documents ayant une clé `price` supérieure à 20 en ne gardant que le champ `name`

Opération `find`

Méthodes sur les curseurs

- L'opération find renvoie un curseur vers une liste de documents sur lequel il est possible d'appliquer des méthodes:
  - `sort` ({clé1:1|-1, …clék:1|-1})}
    - les documents sont triés (1 pour ascendant, -1 pour descendant) selon la première clé, puis la seconde en cas d'égalité, et ce jusqu'à la clé k
  - `skip(n)`
    - les n premiers documents sont supprimés du résultat
  - `limit(n)`: n documents sont retournés au maximum
  - `count()`: compte le nombre de résultats

  - et bien d'autres: https://docs.mongodb.com/manual/reference/method/js-cursor/

Opération `find`

Méthodes sur les curseurs

Tri sur les champs `price` puis `name`:

```
db.coll.find().sort({price:-1, name:1})
```

Limite de 10 documents:

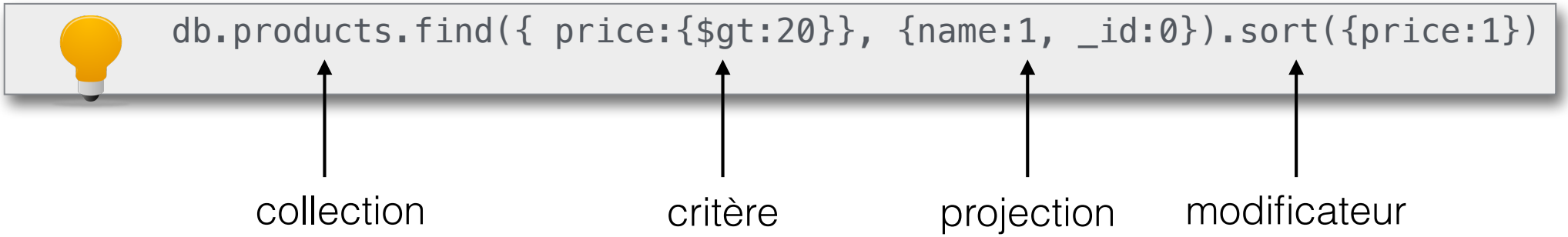
```
db.coll.find().limit(10)
```

Combinaison:

```
db.coll.find().sort({price:-1}).skip(10).limit(50)
```

Opération `find`

Exemple complet

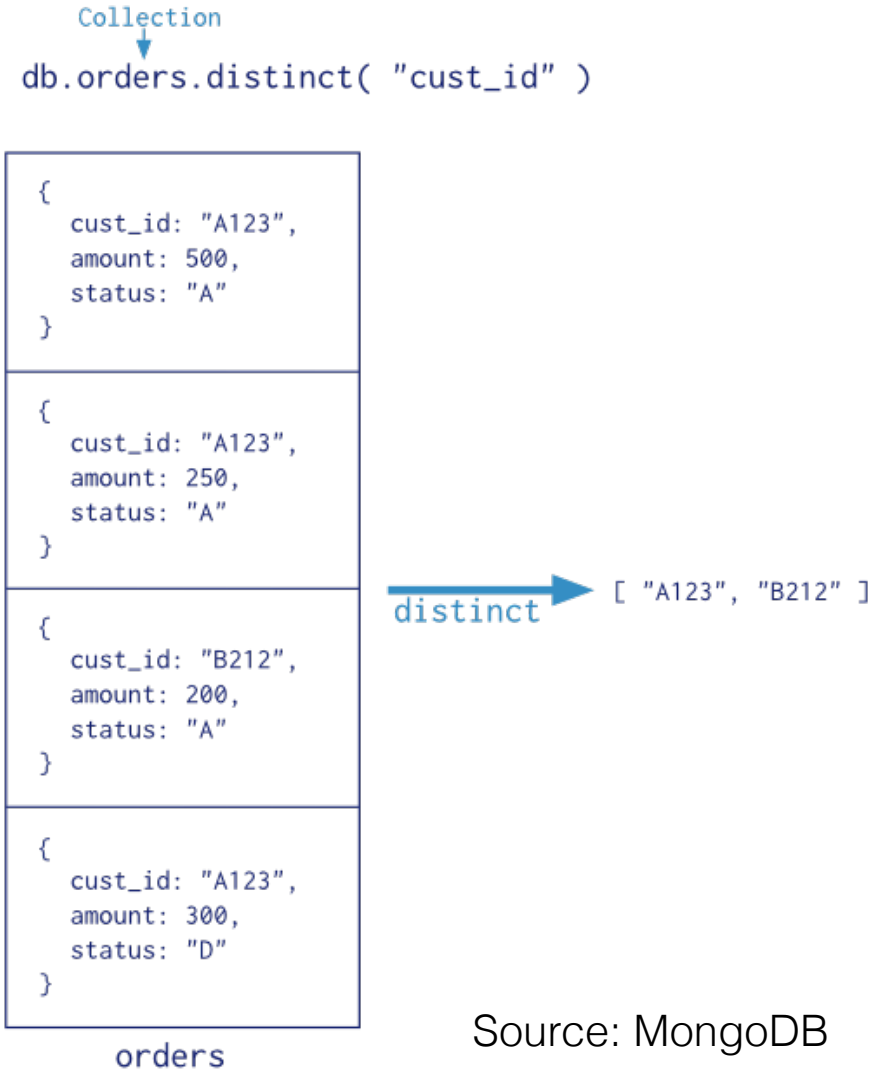


Retourne tous les documents ayant une clé `price` supérieure à 20 en ne gardant que le champ `name`, triés selon la clé `price` croissante

Agrégation

Opérations simples sur collections

- `db.collection.count()`
- `db.collection.group()`
- `db.collection.distinct()`

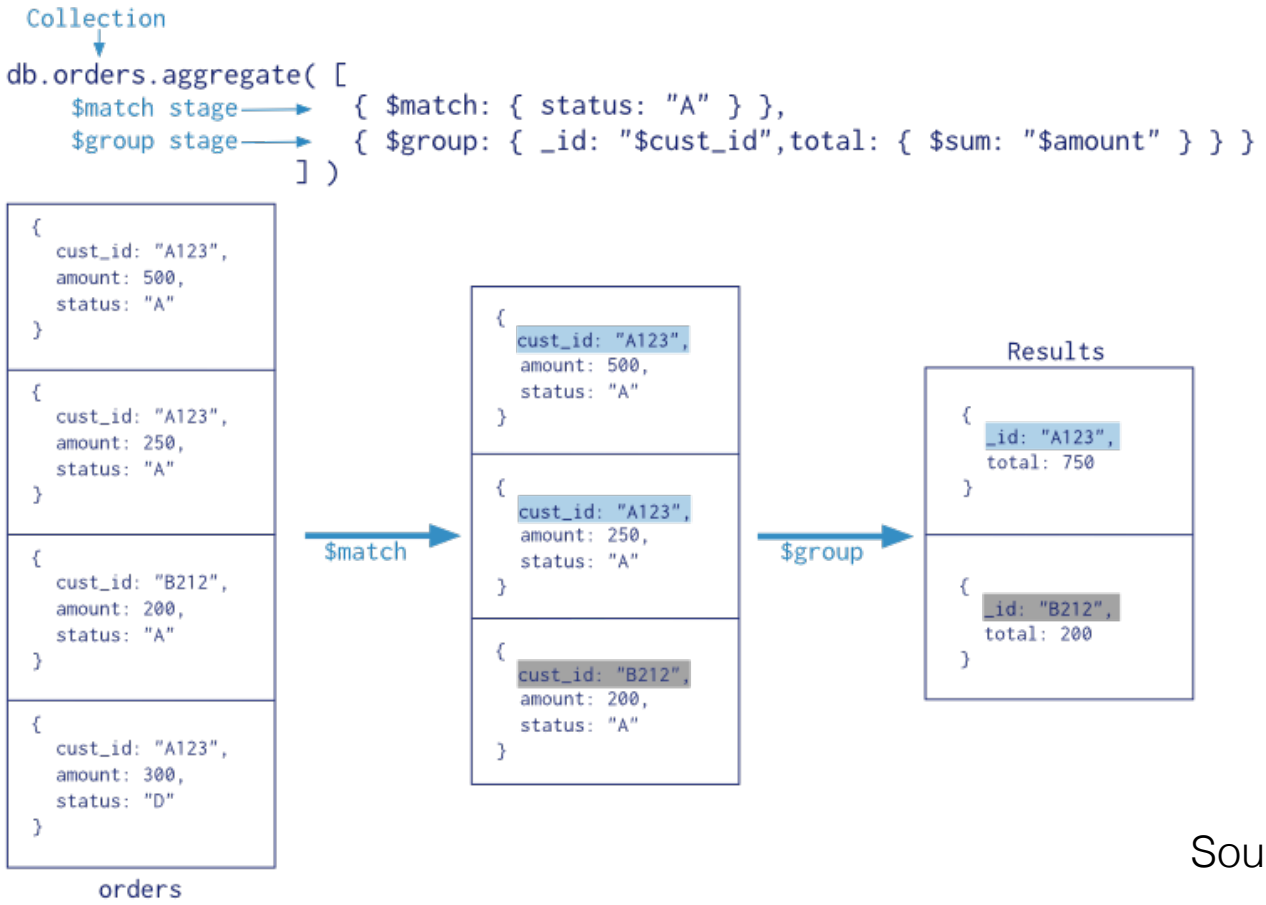


Source: MongoDB

Framework d'agrégation

Pipeline d'agrégation: `aggregate`

- Les documents entrent dans un pipeline qui les transforme en un résultat agrégé



Source: MongoDB

Framework d'agrégation

Pipeline d'agrégation: `aggregate`

- Etapes possibles dans le pipeline:
  - `$project:` projection
  - `$match`: sélection
  - `$unwind`: déconstruction d'un tableau
  - `$group`: groupement des lignes
  - `$sort`: tri
  - `$limit:` limite le nombre de lignes du résultat
  - … (https://docs.mongodb.com/manual/reference/operator/aggregation-pipeline/)
-

Framework d'agrégation

Pipeline d'agrégation: `aggregate`

- Quelques opérateurs utilisables dans le pipeline
  - `$size`: taille d'un tableau
  - `$slice`: sous-ensemble d'un tableau
  - `$setUnion, $setIntersection, $setDifference`: union, intersection et différence entre deux ensembles
  - `$ifNull`: renvoie le premier arguments si non null, le second sinon

- Opérateurs accessibles à l'étape $group
  - `$sum`
  - `$avg`
  - `$max`
  - `$min`
  - `$first` (1er document résultant de l'expression du group)
  - `$last` (dernier document)
  - `$addToSet` (valeurs uniques)

Framework d'agrégation

`aggregate`, exemple

```
{ "_id" : 1,
"item" : "abc",
"price" : 10,
"quantity" : 2,
"date" : ISODate("2014-03-01T08:00:00Z")
}
{ "_id" : 2,
 "item" : "jkl",
 "price" : 20,
 "quantity" : 1,
 "date" : ISODate("2014-03-01T09:00:00Z")
}
{ "_id" : 3,
 "item" : "xyz",
"price" : 5,
 "quantity" : 10,
 "date" : ISODate("2014-03-15T09:00:00Z")
}
{ "_id" : 4,
 "item" : "xyz",
 "price" : 5, "quantity" : 20,
"date" : ISODate("2014-04-04T11:21:39.736Z")
}
{ "_id" : 5,
"item" : "abc",
 "price" : 10,
"quantity" : 10,
"date" : ISODate("2014-04-04T21:23:13.331Z")
}
```

```
db.sales.aggregate(
   [
      {
        $group : {
           _id : { month: { $month:
"$date" }, day: { $dayOfMonth: "$date" },
year: { $year: "$date" } },
           totalPrice: { $sum: { $multiply:
[ "$price", "$quantity" ] } },
           averageQuantity: { $avg:
"$quantity" },
           count: { $sum: 1 }
        }
      }
   ]
)
```

Framework d'agrégation

`aggregate`, exemple

```
db.sales.aggregate(
   [
      {
        $group : {
           _id : { month: { $month:
"$date" }, day: { $dayOfMonth: "$date" },
year: { $year: "$date" } },
           totalPrice: { $sum: { $multiply:
[ "$price", "$quantity" ] } },
           averageQuantity: { $avg:
"$quantity" },
           count: { $sum: 1 }
        }
      }
   ]
)
```

```
{
 "_id" :
    { "month" : 3, "day" : 15, "year" : 2014 },
"totalPrice" : 50,
"averageQuantity" : 10,
"count" : 1
}
{
"_id" :
    { "month" : 4, "day" : 4, "year" : 2014 },
 "totalPrice" : 200,
"averageQuantity" : 15,
"count" : 2
}
{
"_id" :
    { "month" : 3, "day" : 1, "year" : 2014 },
 "totalPrice" : 40,
"averageQuantity" : 1.5,
"count" : 2
}
```

Opération `update`

- Syntaxe pour mettre à jour un document dans une collection *coll*:
  - `<query>` est un document utilisant des opérateurs de requête
  - `<maj>` est un document avec les mises à jour
  - `<option>` est un document pouvant contenir
    - un booléen *upsert* (création d'un document si aucun résultat pour query). False par défaut
    - un booléen *multi* (pour mettre à jour plusieurs documents)
    - un document *writeConcern* (type de garantie d'écriture)

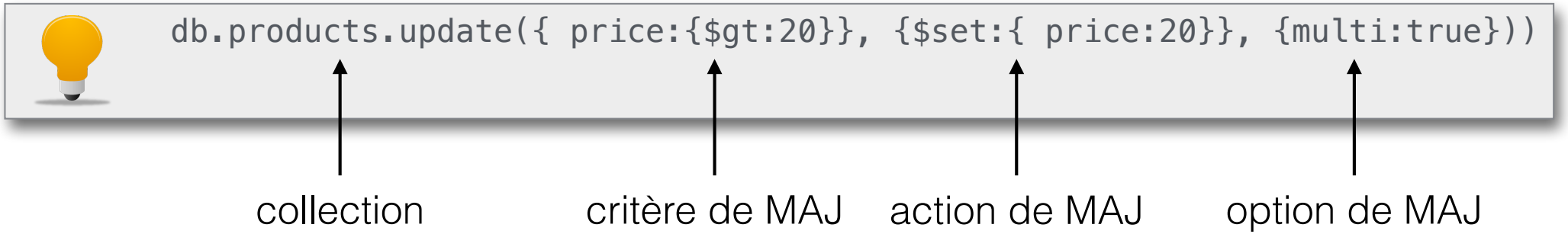
```
db.coll.update(
   <query>,
   <maj>,
   <options>
   )
```

Opération `update`

- Contenu du document `<maj>`:
  - Uniquement des opérateurs de mise à jour
    - le(s) document(s) retourné(s) par <query> ont leurs clés mises à jour
    - opérateurs sur clés (par exemple $inc, $rename, $set)
    - opérateurs sur tableaux (par exemple $pop, $push, $addToSet)
  - Uniquement des paires clé/valeur:
    - le document <maj> remplace l'intégralité du **premier document** répondant aux critères de <query>
    - la valeur de _id est conservée

Opération `update`

Exemple complet



Mise à jour de la clé `price` (nouvelle valeur: 20) pour tous les documents (option « multi ») contenant une clé `price` supérieure à 20

Opération `remove`

- Syntaxe pour supprimer une base de données db

```
db.dropDatabase()
```

- Syntaxe pour supprimer une collection coll

```
db.coll.remove({})
```

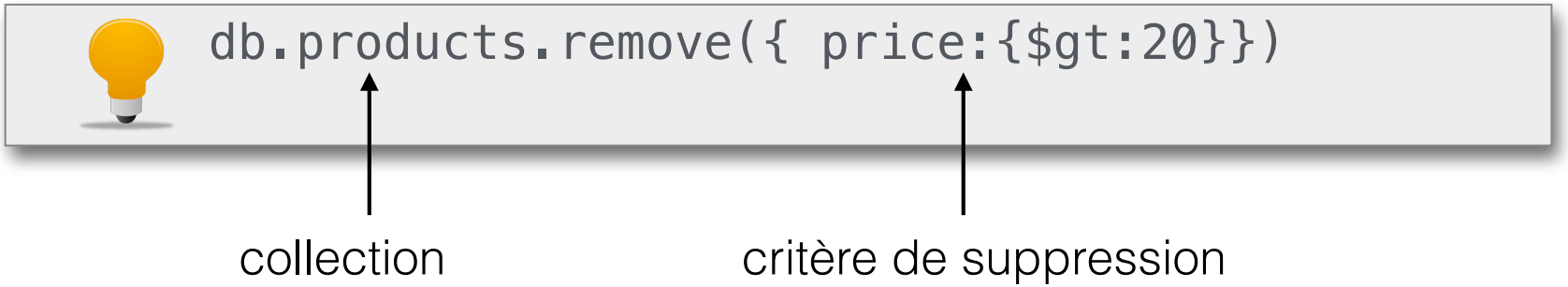
Opération `remove`

- Syntaxe pour supprimer des documents de la collection *coll*:
  - `<query>` est un document utilisant des opérateurs de requête
  - `<option>` est un document pouvant contenir
    - un booléen *justOne* (pour supprimer un seul document)
    - un document *writeConcern* (type de garantie d'écriture)

```
db.coll.remove(
   <query>,
   <options>
   )
```

Opération `remove`

Exemple

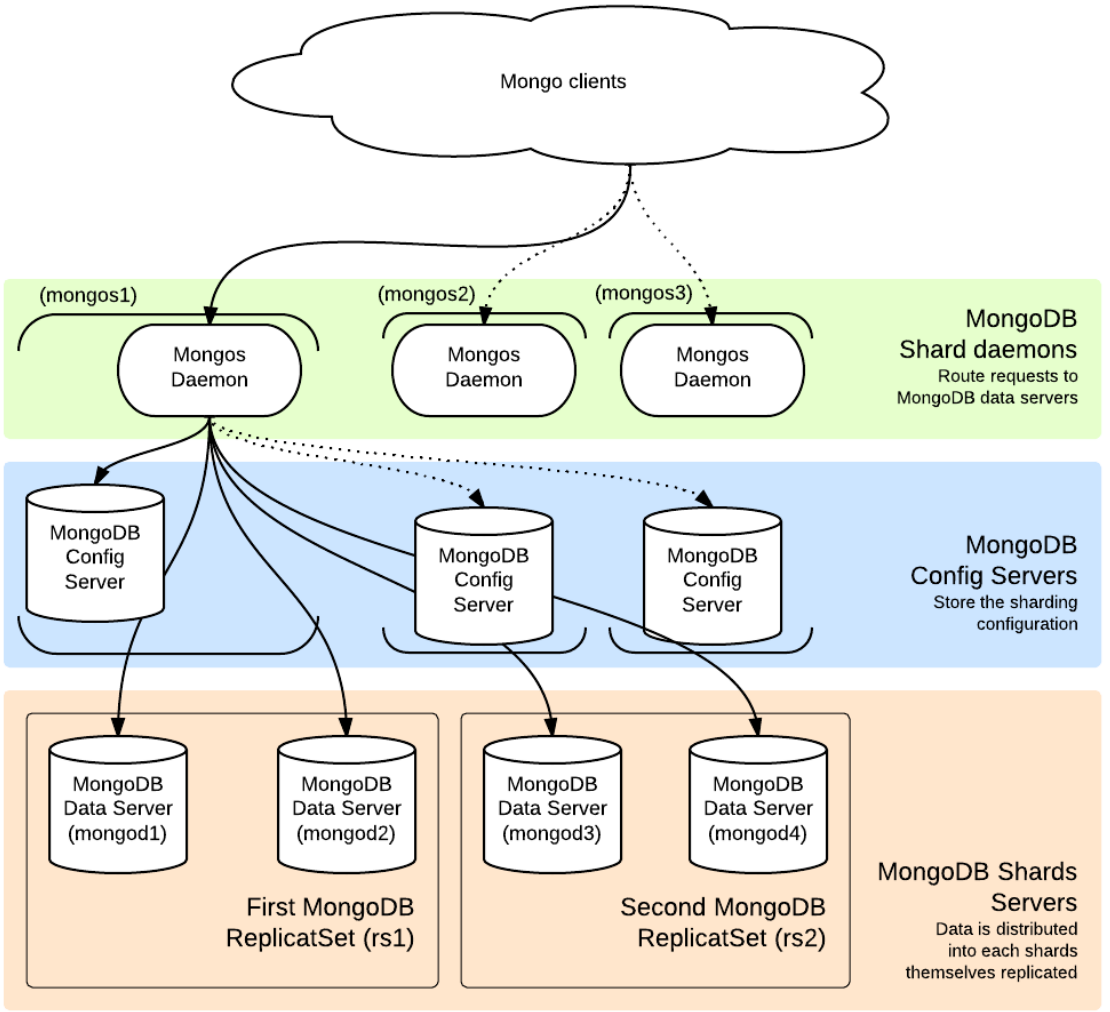


Suppression de tous les documents de la collection `products` dont la clé `price` est supérieure à 20

Mise en oeuvre

- Configurations d'un ou plusieurs serveurs **`mongod`**

- Accès client:
  - via l'invite de commandes interactive **`mongo`**
    - invite de commande javascript
  - via un langage hôte



Exemple d'architecture.
Source: https://blog.sileht.net/using-a-shardingreplicaset-mongodb-with-ceilometer.htm

MongoDB

Drivers pour programmation Client

- C
- C++
- Haskell
- Erlang
- Java en TP
- javascript
- .net
- Perl
- PHP
- Python
- Ruby
- Scala

Bases de données document

Forces et Faiblesses

Quand utiliser ?

- Recherches sur le contenu nécessaires
- Exemples d'applications:
  - Gestion de logs
  - Applications d'e-commerce pour lesquelles le schéma des produits/ commandes est susceptible d'évoluer.

- https://www.quora.com/What-is-the-use-case-to-choose-a-document-oriented-database-noSQL-over-RDBMS

Quand ne pas utiliser ?

- Besoin de croiser fréquemment des données entre les documents
- Requêtes sur des niveaux différents de granularité

- http://www.sarahmei.com/blog/2013/11/11/why-you-should-never-use-mongodb/

Des questions ?

Références

- Livres:
  - La révolution Big Data: les données au coeur de la transformation de l'entreprise. J.C. Cointot. Junod, 2014
  - Bases de données noSQL et Big Data. P. Lacomme, S. Aridhi, R. Phan, Ellipse, 2014.
  - NOSQL Distilled: a brief guide to the emerging world of polyglot persistence. Pramod J. Sadalage, Martin Fowler. Addison-Wesley, 2013
  - Les bases de données et le Big Data: Comprendre et mettre en oeuvre. Rudi Bruchez, Eyrolles, 2015
- Supports de cours / présentations:
  - http://fr.slideshare.net/mongodb/the-spring-data-mongodb-project
  - http://fr.slideshare.net/mongodb/introduction-to-mongodb-56807822
  - Introduction aux systèmes NoSQL, Bernard Espionnasse, mai 2016
  - Les SGBD non relationnels, Fabien Duchateau (accédé en juin 2016)

Quelques liens supplémentaires

- Recherche plein texte avec MongoDB:
  - http://code.tutsplus.com/fr/tutorials/full-text-search-in-mongodb--cms-24835

- Comparaison des performances de MongoDB avec d'autres systèmes de bases de données:
  - http://www.planetcassandra.org/nosql-performance-benchmarks/
  - https://www.arangodb.com/2015/10/benchmark-postgresql-mongodb-arangodb/
  - et bien d'autres…