

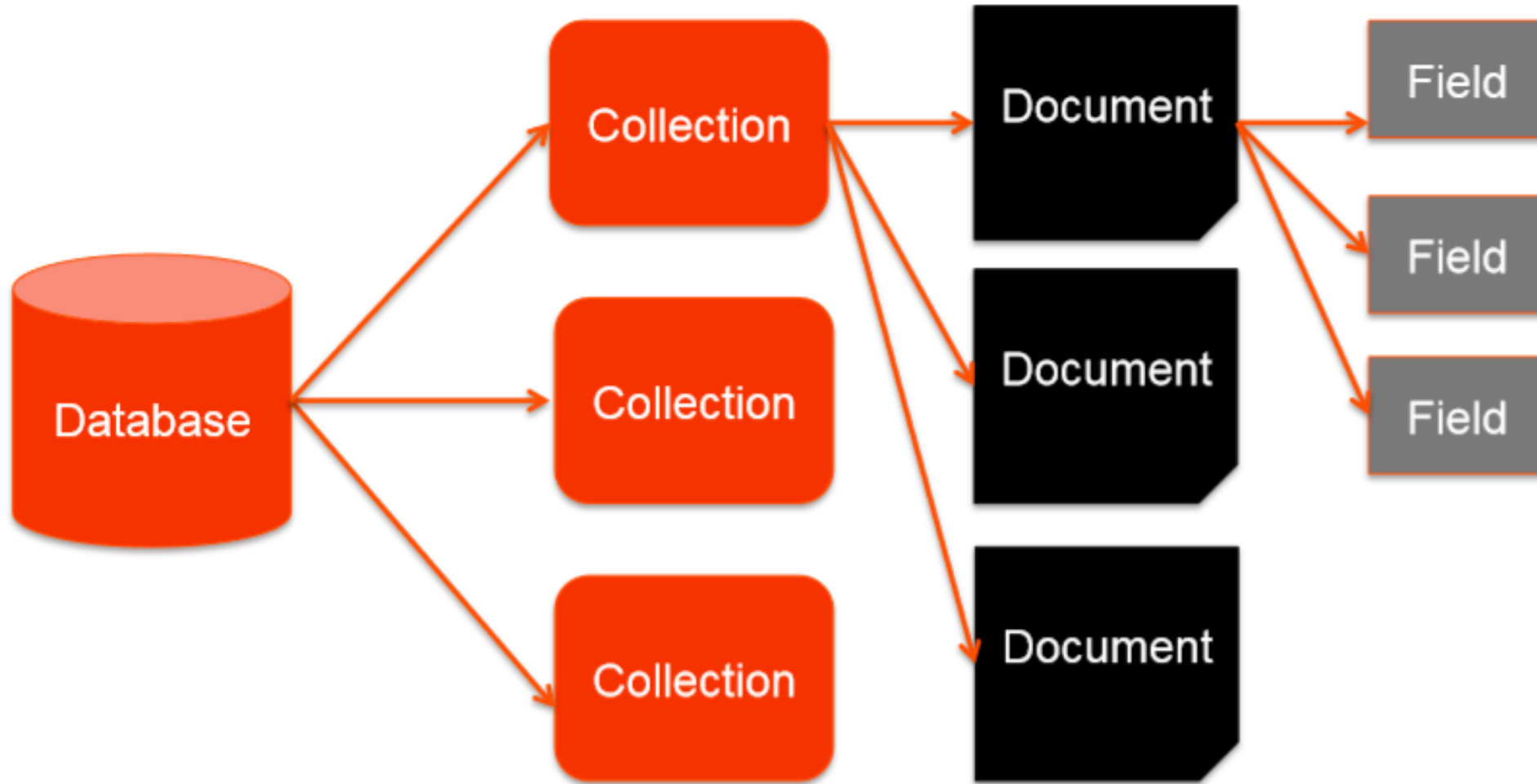
# Modèle Document

- **Principes**
- **MongoDB**
- **CouchBase**
- **ElasticSearch (moteur de recherche NoSQL)**
- **JSON/SQL sous PostgreSQL**

# Base de données documents

- Basé également sur le paradigme [clé, valeur], mais avec une valeur, de type document
- Document : structure arborescente contenant une liste de champs
- Champs est associé à une valeur qui peut elle même être une liste
- Documents principalement de type JSON ou XML
- Représentation d'un document :
  - Forme sérialisée (la plus courante) : contenu marqué par des balises ou par des accolades ouvrante/fermante.
  - Forme arborescente

# Collection de documents

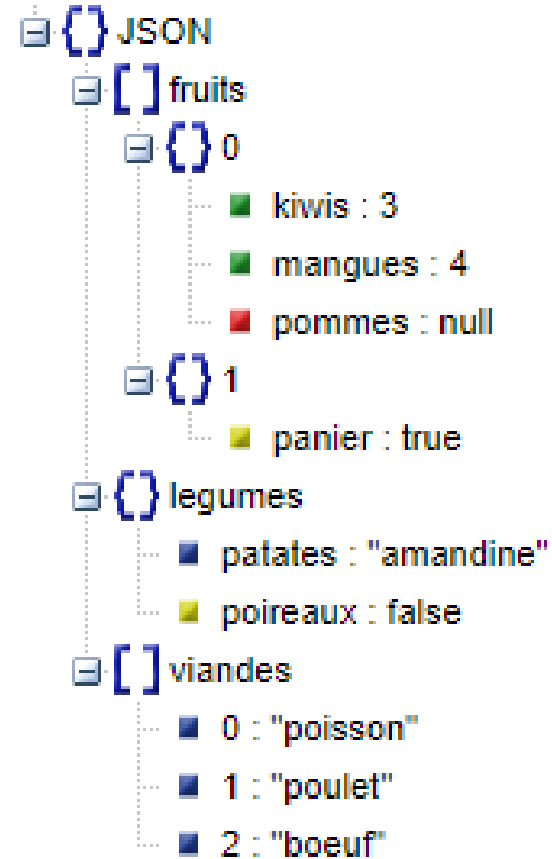


# Document JSON

## Forme sérialisée

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

## Forme arborescente



# Base de données documents : cas d'utilisation

- **Cas d'utilisation :**

- Systèmes de gestion de contenu (profils utilisateurs, avis et commentaires, blogs, etc.)
- Applications de e-commerce (stockage du catalogue, schéma flexible pour les produits et commandes)

- **Non applicable aux:**

- Données interconnectées
- Données non-structurées

# Principaux moteurs orientés document

include secondary database models

56 systems in ranking, February 2023

Rank			DBMS	Database Model	Score		
Feb 2023	Jan 2023	Feb 2022			Feb 2023	Jan 2023	Feb 2022
1.	1.	1.	MongoDB	Document, Multi-model	452.77	-2.42	-35.88
2.	2.	2.	Amazon DynamoDB	Multi-model	79.69	-1.87	-0.67
3.	3.		Databricks	Multi-model	60.33	-0.49	
4.	4.	3.	Microsoft Azure Cosmos DB	Multi-model	36.51	-1.45	-3.45
5.	5.	4.	Couchbase	Document, Multi-model	24.86	-0.44	-5.21
6.	6.	5.	Firebase Realtime Database	Document	18.49	-0.27	-0.66
7.	7.	6.	CouchDB	Document, Multi-model	14.45	-0.44	-3.01
8.	8.	9.	Google Cloud Firestore	Document	11.51	+0.41	+2.45
9.	9.	7.	MarkLogic	Multi-model	8.84	-0.01	-0.61
10.	10.	8.	Realm	Document	8.24	-0.09	-1.17
11.	11.	13.	Google Cloud Datastore	Document	6.89	+0.22	+1.64
12.	12.	10.	Aerospike	Multi-model	6.56	+0.09	+0.95
13.	13.	12.	Virtuoso	Multi-model	6.11	+0.23	+0.72
14.	14.	11.	ArangoDB	Multi-model	5.29	+0.22	-0.11
15.	15.	14.	OrientDB	Multi-model	4.54	+0.06	-0.49

# MongoDB

- Base de données orientée documents (représentation structurée)
- Open source développé par MongoDB Inc depuis 2007
- Nom provenant de l'anglais "humongous" ("énorme")
- Ecrit en C++
- *Mongo Atlas* : mode hébergé dans le cloud sur la base d'une tarification horaire
- Stockage des documents au format BSON (*Binary Serialized dOcument Notation* ou *Binary JSON*) – mais structure visible par les utilisateurs en JSON

# MongoDB : liens utiles

- Site officiel : <https://www.mongodb.com/fr>
- Documentation : : <https://docs.mongodb.com/manual/introduction/>
- Console en ligne dans la documentation :
  - <https://docs.mongodb.com/manual/tutorial/insert-documents/> (il faut sélectionner la version 4.2)
  - <https://www.mycompiler.io/new/mongodb> ou <https://onecompiler.com/mongodb>
- Tutoriels :
  - <https://www.mongodb.com/docs/manual/tutorial/>
  - <http://b3d.bdpedia.fr/docstruct.html#s4-mongodb-une-base-json>
  - <https://stph.scenari-community.org/contribs/nos/Mongo1/co/presentation.html>
  - [https://www.tutorialspoint.com/mongodb/mongodb\\_overview.htm](https://www.tutorialspoint.com/mongodb/mongodb_overview.htm)



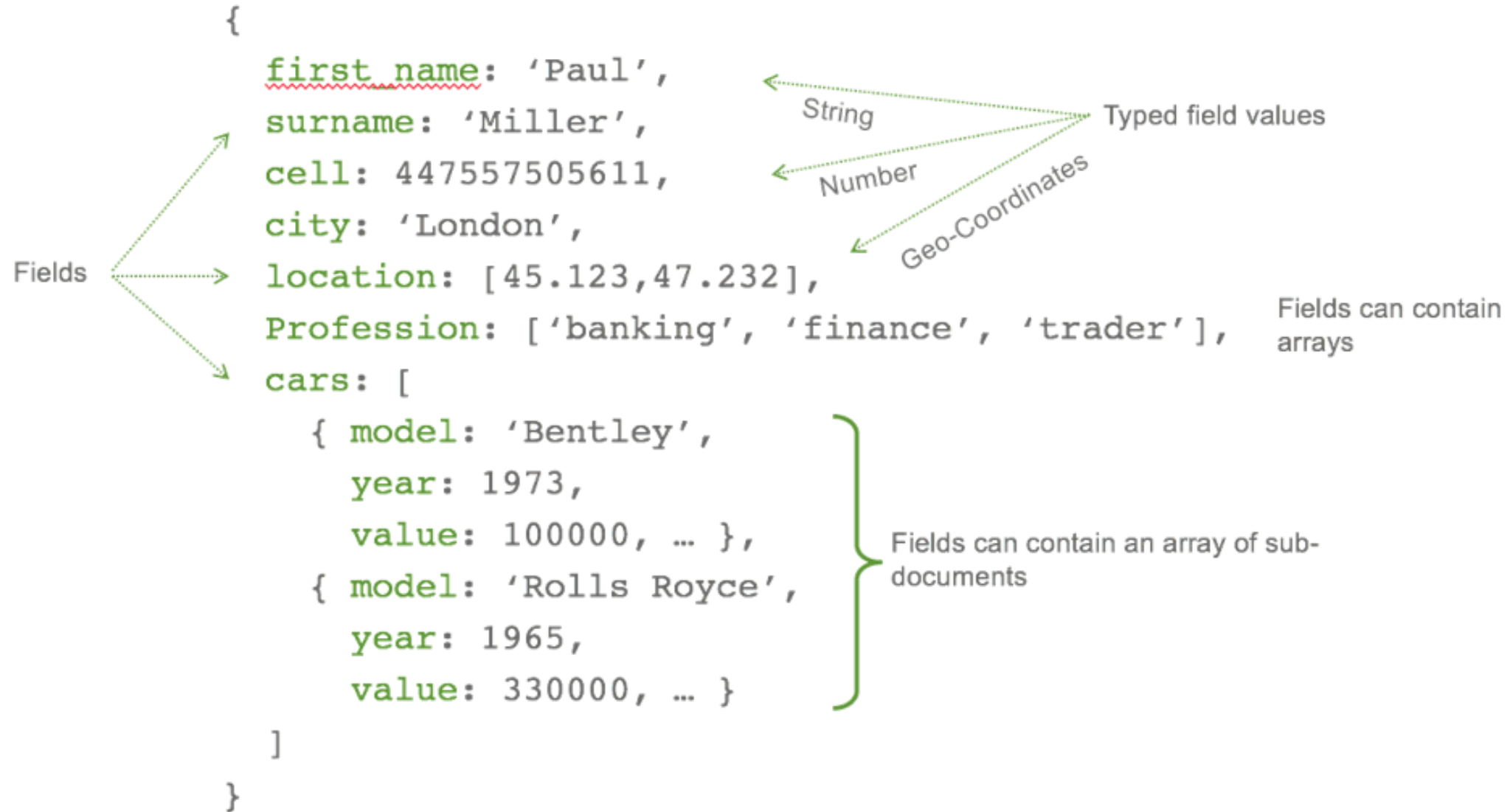
# MongoDB : modèle de données (1/2)

RDBMS	MongoDB
Database	Database
Table	Collection
Tuple/Row	Document
column	Field
Table Join	Embedded Documents
Primary Key	Primary Key (Default key <code>_id</code> provided by mongodb itself)

# MongoDB : modèle de données (2/2)

- Document BSON = unité de stockage (~ une ligne dans une BDR)
- Pour l'utilisateur : structure visible en JSON
- Tout document appartient à une collection et a un champ appelé `_id` qui identifie le document dans la base de données
- Document : hiérarchie de paires clé-valeur, sans contrainte de présence ou de quantité
- Collection : ensemble de documents (~table en relationnel)

# MongoDB : exemple



# MongoDB : attribut identificateur `_id`

```
{
  _id: ObjectId(7df78ad8902c)
  title: 'MongoDB Overview',
  description: 'MongoDB is no sql database',
  by: 'tutorials point',
  url: 'http://www.tutorialspoint.com',
  tags: ['mongodb', 'database', 'NoSQL'],
  likes: 100,
  comments: [
    {
      user: 'user1',
      message: 'My first comment',
      dateCreated: new Date(2011,1,20,2,15),
      like: 0
    },
    {
      user: 'user2',
      message: 'My second comments',
      dateCreated: new Date(2011,1,25,7,45),
      like: 5
    }
  ]
}
```

# MongoDB : relationnel vs document

## Relationnel

Customer ID	First Name	Last Name	City
0	John	Doe	New York
1	Mark	Smith	San Francisco
2	Jay	White	Dallas
3	Meagan	White	London
4	Edward	Daniels	Boston

Phone Number	Type	DNC	Customer ID
1-212-555-1212	home	T	0
1-212-555-1213	home	T	0
1-212-555-1214	cell	F	0
1-212-777-1212	home	T	1
1-212-777-1213	cell	(null)	1
1-212-888-1212	home	F	2



## MongoDB

```
{ customer_id : 1,
  name : {
    "f": "Mark",
    "l": "Smith" },
  city : "San Francisco",
  phones: [ {
    number : "1-212-777-1212",
    dnc : true,
    type : "home"
  },
  {
    number : "1-212-777-1213",
    type : "cell"
  }
]
```

# MongoDB : Relation implémentée par une imbrication de documents

```
{
  "_id": ObjectId("52ffc33cd85242f436000001"),
  "contact": "987654321",
  "dob": "01-01-1991",
  "name": "Tom Benzamin",
  "address": [
    {
      "building": "22 A, Indiana Apt",
      "pincode": 123456,
      "city": "Los Angeles",
      "state": "California"
    },
    {
      "building": "170 A, Acropolis Apt",
      "pincode": 456789,
      "city": "Chicago",
      "state": "Illinois"
    }
  ]
}
```

# MongoDB : Relation implémentée par l'utilisation de référence

```
{
  "_id":ObjectId("52ffc4a5d85242602e000000"),
  "building": "22 A, Indiana Apt",
  "pincode": 123456,
  "city": "Los Angeles",
  "state": "California"
}
```

```
{
  "_id":ObjectId("52ffc33cd85242f436000001"),
  "contact": "987654321",
  "dob": "01-01-1991",
  "name": "Tom Benzamin",
  "address_ids": [
    ObjectId("52ffc4a5d85242602e000000"),
    ObjectId("52ffc4a5d85242602e000001")
  ]
}
```

# MongoDB : principes

- En général, regroupement de toutes les données relatives à un objet dans un même « document »
- ⇒ nombre de jointures réduit et donc impact positif sur les performances (toutes les données sont récupérées en une seule lecture)
- Document proche de la structure des objets dans les langages de programmation (facilite le développement)
  - Modèle des documents pouvant varier de structure dans une même collection
  - Schéma dynamique : possibilité d'ajouter de nouveaux champs sans affecter les autres documents
  - Schéma flexible mais conçu selon le type de requêtes



# MongoDB : opérations CRUD (insertion)

```
db.users.insertOne(
  {
    name: "sue",
    age: 26,
    status: "pending"
  }
)
```

← collection

← field: value

← field: value

← field: value

} document

# MongoDB : Console et un tutoriel interactif

```
db.inventory.insertOne(  
  { item: "canvas", qty: 100, tags: ["cotton"], size: { h: 28, w: 35.5, uom: "cm" } }  
)
```

You can run the operation in the web shell below:

```
Click to connect  
Connecting...  
MongoDB shell version v3.6.0  
connecting to: mongodb://127.0.0.1:27017  
MongoDB server version: 3.6.0  
type "help" for help  
>>> db.inventory.insertOne(  
...   { item: "canvas", qty: 100, tags: ["cotton"], size: { h:  
28, w: 35.5, uom: "cm" } }  
... )  
{  
  "acknowledged" : true,  
  "insertedId" : ObjectId("5c66db7c32ab66414816fcd7")  
}  
>>>
```

# MongoDB : opérations CRUD

## (initialisation ou non de l'attribut `_id`)

- Soit MongoDB crée le champ `_id` et lui affecte une valeur `ObjectId` unique

```
db.products.insert( { item: "card", qty: 15 } )
```

```
{ "_id" : ObjectId("5063114bd386d8fadbd6b004"), "item" : "card", "qty" : 15 }
```

- Soit on crée cet identificateur à l'insertion

```
db.products.insert( { _id: 10, item: "box", qty: 20 } )
```

```
{ "_id" : 10, "item" : "box", "qty" : 20 }
```

# MongoDB : opérations CRUD

## (mixage de la gestion des identificateurs)

```
db.products.insert(  
  [  
    { _id: 11, item: "pencil", qty: 50, type: "no.2" },  
    { item: "pen", qty: 20 },  
    { item: "eraser", qty: 25 }  
  ]  
)
```

```
{ "_id" : 11, "item" : "pencil", "qty" : 50, "type" : "no.2" }  
{ "_id" : ObjectId("51e0373c6f35bd826f47e9a0"), "item" : "pen", "qty" : 20 }  
{ "_id" : ObjectId("51e0373c6f35bd826f47e9a1"), "item" : "eraser", "qty" : 25 }
```

# MongoDB : opérations CRUD (**find**)

```
db.users.find(  
  { age: { $gt: 18 } },  
  { name: 1, address: 1 }  
) .limit(5)
```

← collection  
← query criteria  
← projection  
← cursor modifier

# MongoDB : traduction de `find` en SQL (1/2)

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

```
SELECT * FROM inventory
```

```
db.inventory.find( { status: "D" } )
```

```
SELECT * FROM inventory WHERE status = "D"
```

```
db.inventory.find( { status: { $in: [ "A", "D" ] } } )
```

```
SELECT * FROM inventory WHERE status in ("A", "D")
```

```
db.inventory.find( { status: "A", qty: { $lt: 30 } } )
```

```
SELECT * FROM inventory WHERE status = "A" AND qty < 30
```

# MongoDB : MongoDB : traduction de `find` en SQL (2/2)

```
db.inventory.find( { status: "A" }, { item: 1, status: 1 } )
```

```
SELECT _id, item, status from inventory WHERE status = "A"
```

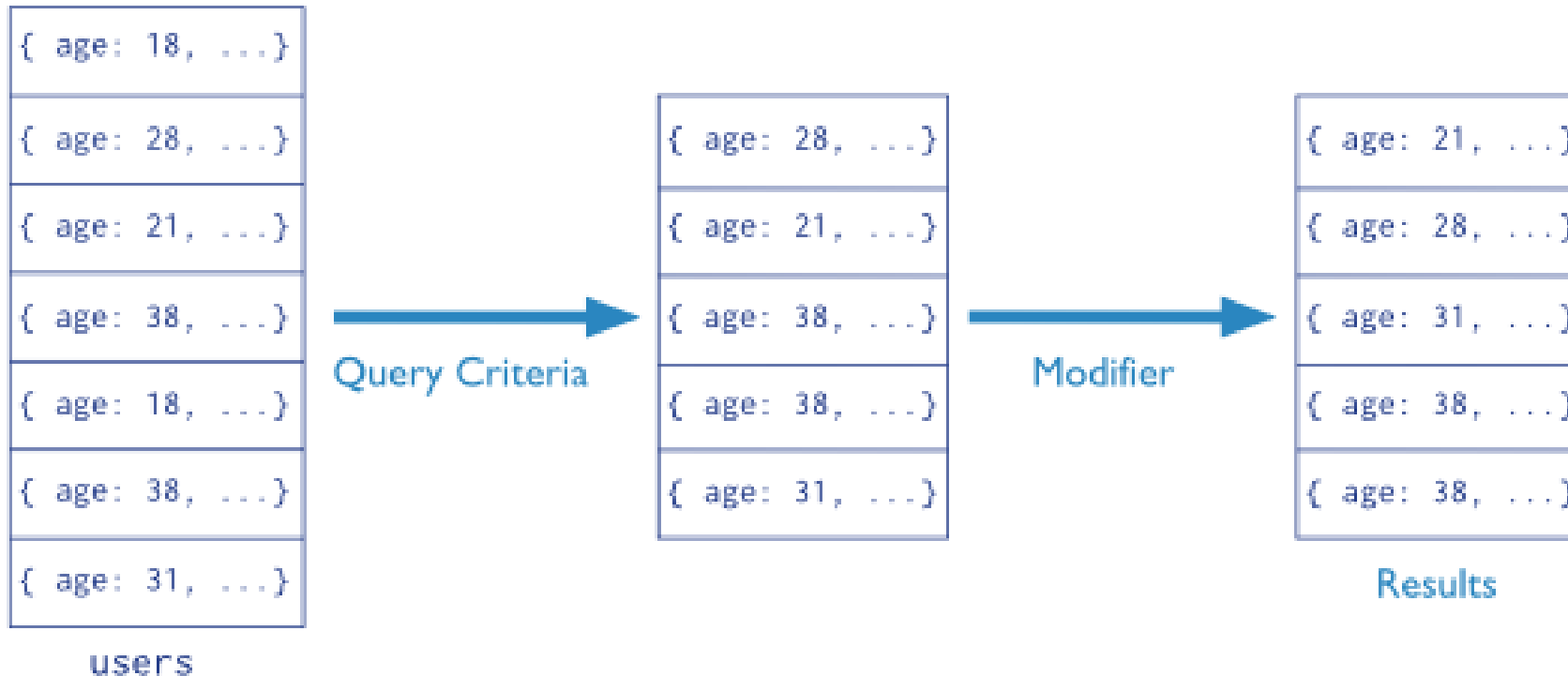
```
db.inventory.find( { status: "A" }, { item: 1, status: 1, _id: 0 } )
```

```
SELECT item, status from inventory WHERE status = "A"
```

**Attention : Pas d'erreur en cas de faute de frappe sur des noms d'attributs**

# MongoDB : exemple de tri

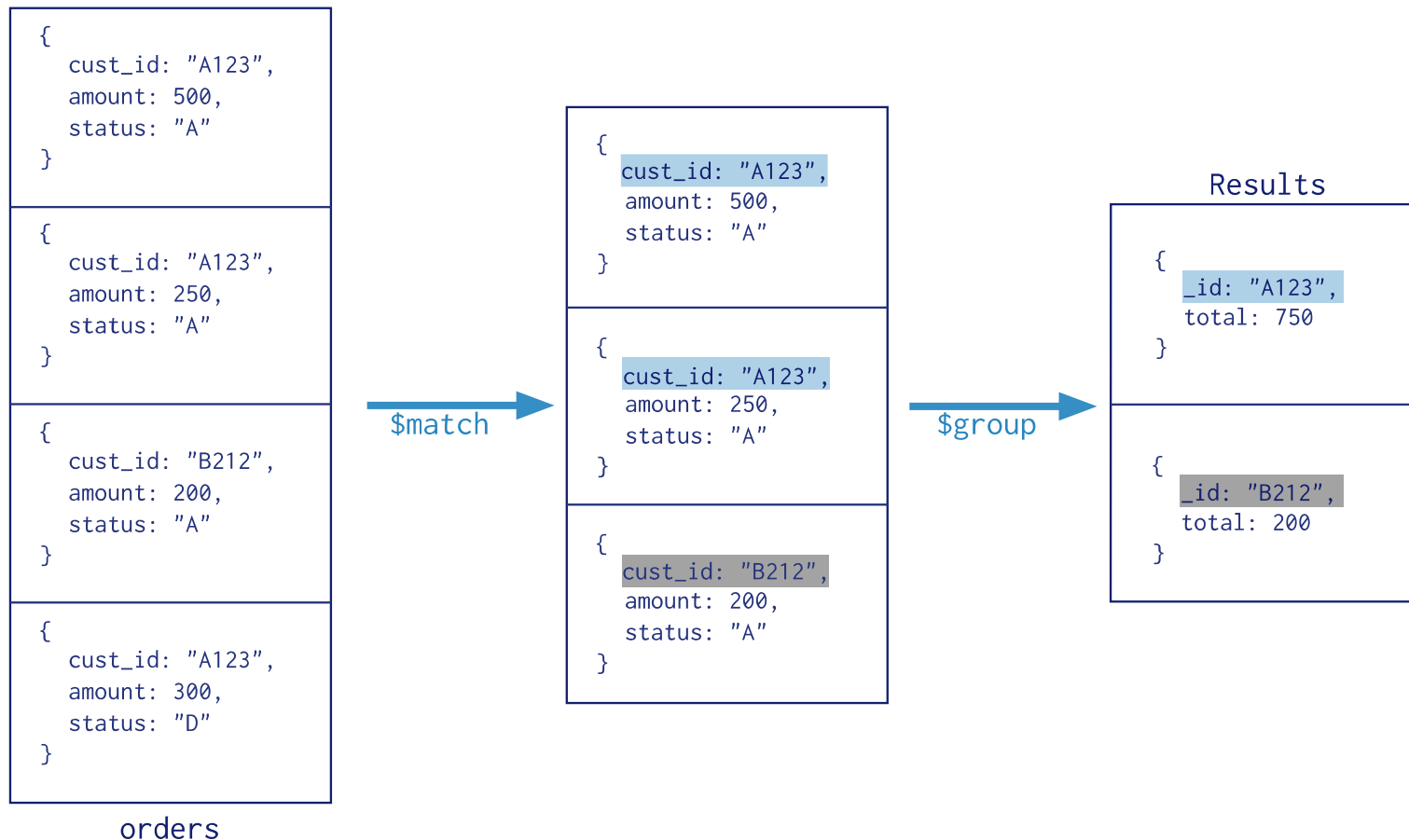
Collection                      Query Criteria                      Modifier  
`db.users.find( { age: { $gt: 18 } } ).sort( {age: 1 } )`





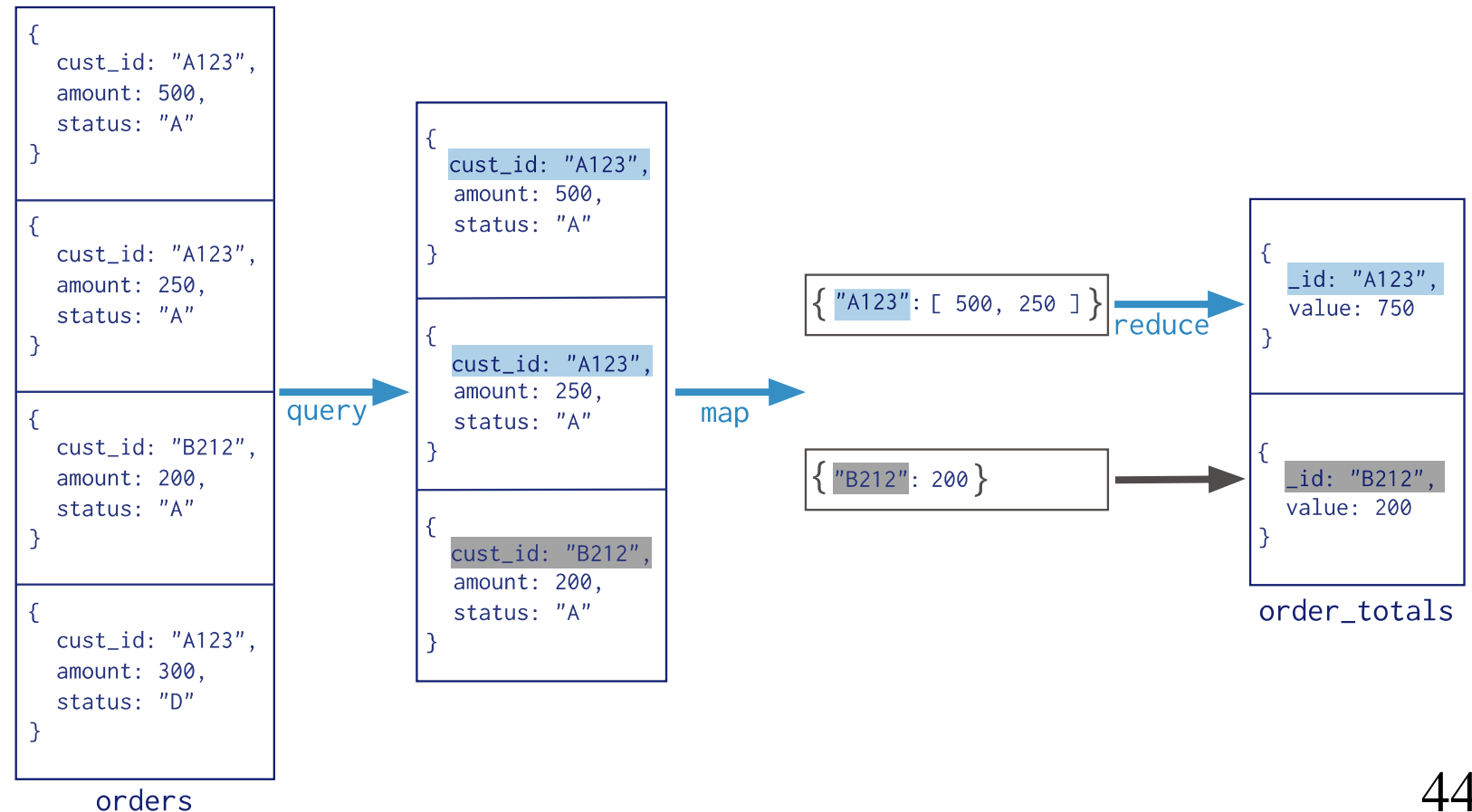
# MongoDB : agrégation

Collection  
↓  
db.orders.aggregate( [  
 \$match stage → { \$match: { status: "A" } },  
 \$group stage → { \$group: { \_id: "\$cust\_id", total: { \$sum: "\$amount" } } }  
] )



# MongoDB : agrégation en utilisant *Map Reduce*

Collection  
↓  
db.orders.mapReduce(  
  map     → function() { emit( this.cust\_id, this.amount ); },  
  reduce  → function(key, values) { return Array.sum( values ) },  
  query   → { query: { status: "A" } },  
  output  → { out: "order\_totals" }  
)



# MongoDB : opérations CRUD (update)

```
db.users.updateMany(  
  { age: { $lt: 18 } },  
  { $set: { status: "reject" } }  
)
```

← collection  
← update filter  
← update action

# MongoDB : opérations CRUD (delete)

```
db.users.deleteMany(  
  { status: "reject" }  
)
```



# MongoDB : possibilités de requêtage

- Possibilité de faire des requêtes d'agrégation et d'utiliser *map-reduce*
- Recherche textuelle
- Gestion de données spatiales
  - Données modélisées en GeoJSON (<http://geojson.org/>)

```
location: {  
  type: "Point",  
  coordinates: [-73.856077, 40.848447]  
}
```

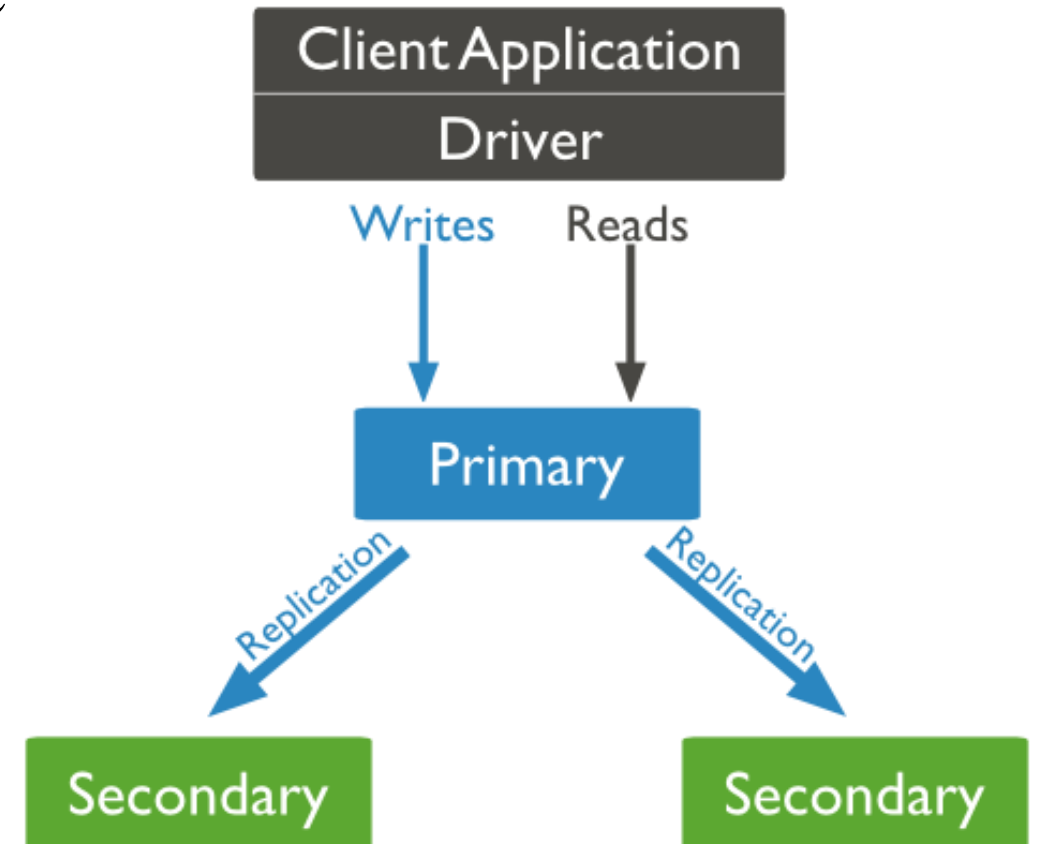
- Requêtes spatiales (cf. <https://docs.mongodb.com/manual/geospatial-queries/>)

# MongoDB : atomicité

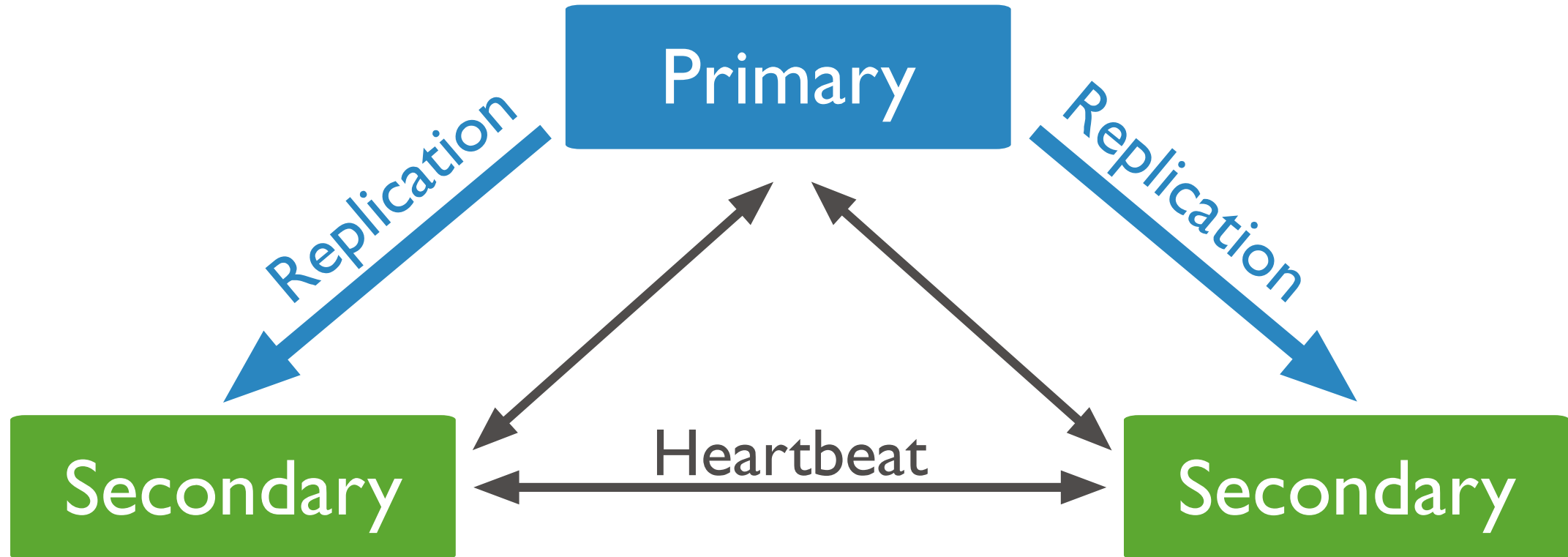
- Ecriture d'un document = opération atomique
- Ecriture de plusieurs documents est atomique pour chacun des documents, mais pas pour l'ensemble
- Ecritures atomiques  $\Rightarrow$  impact sur le design des documents et des collections
- Nécessité de s'assurer de maintenir la cohérence des données

# MongoDB : réplication (1/2)

- *Replicat set* : grappe de serveurs partageant des copies d'un même ensemble de documents
- *Primary* : Un des nœuds avec le rôle de maître
- *Secondaries* : autres nœuds esclaves

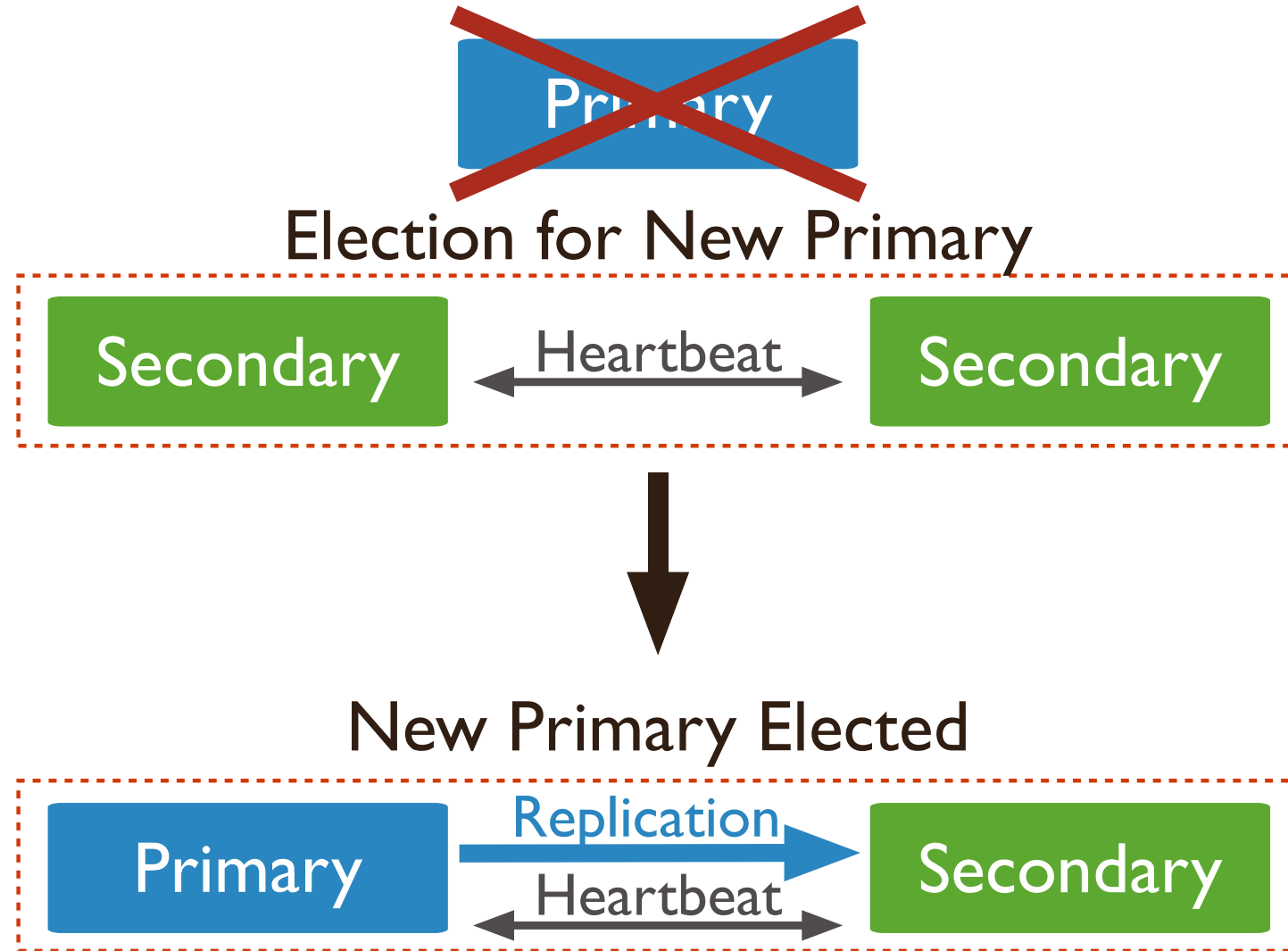


# MongoDB : réplication (2/2)

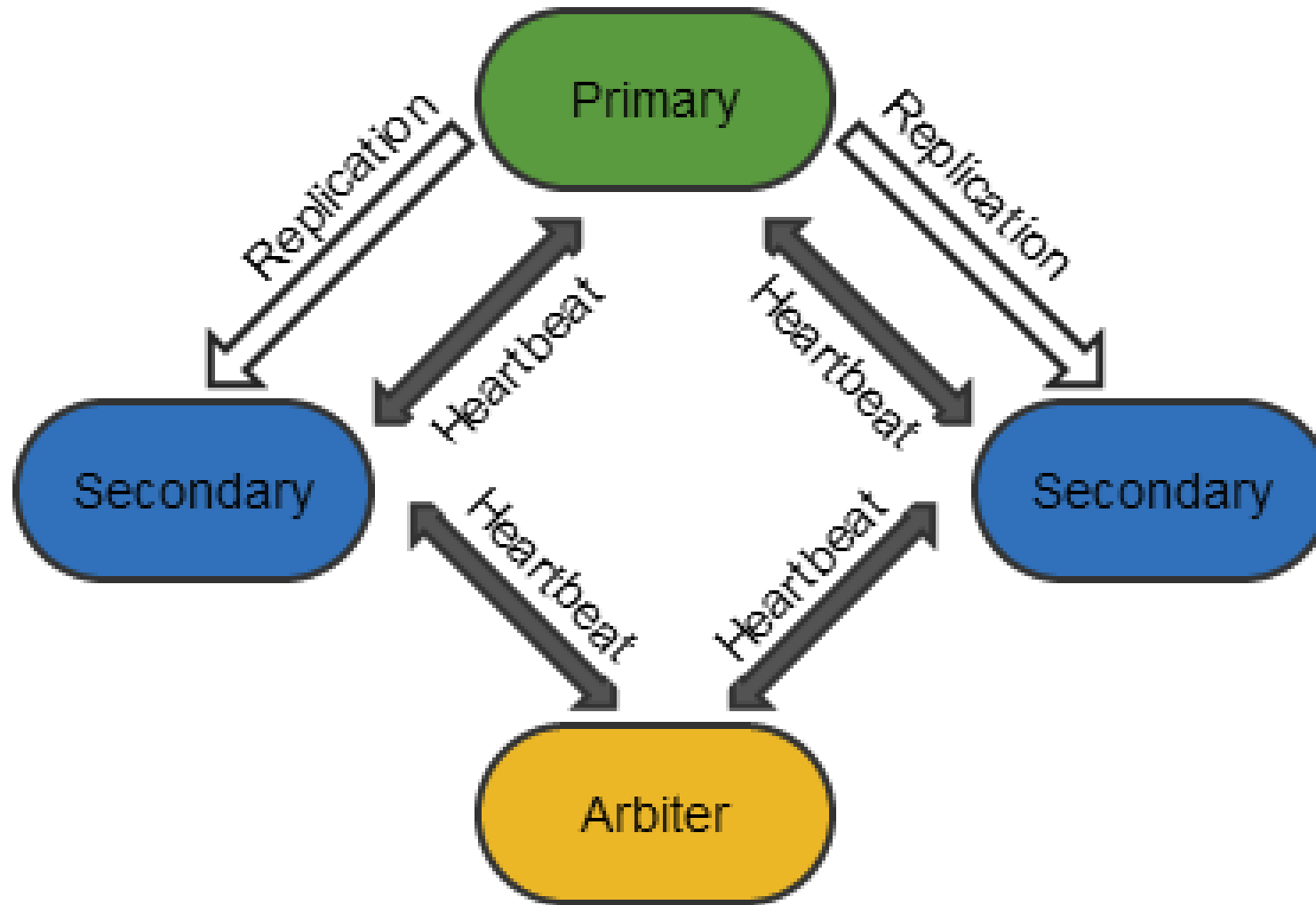




# MongoDB : élection d'un nouveau maitre



# MongoDB : arbitre

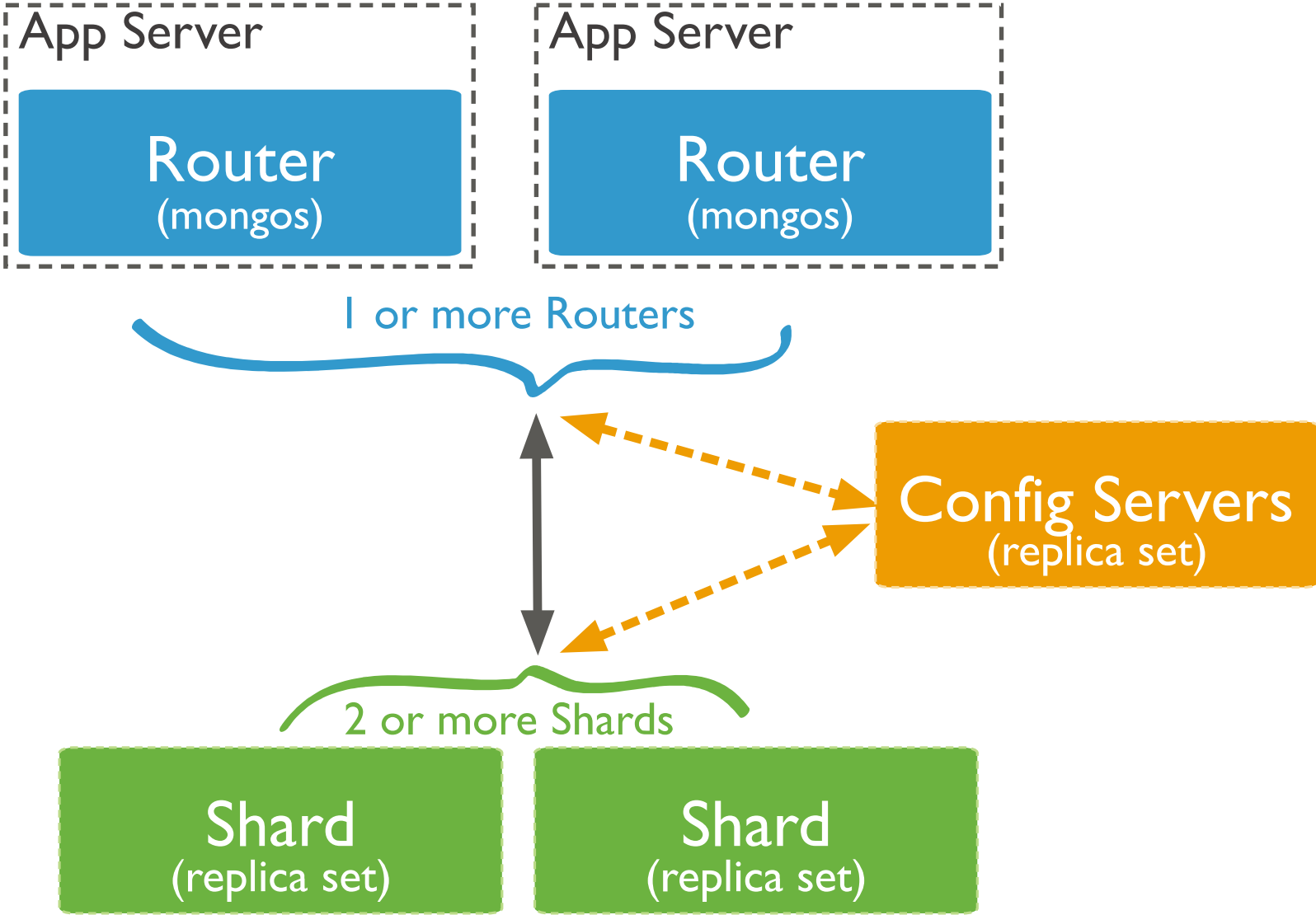


# MongoDB : répartition

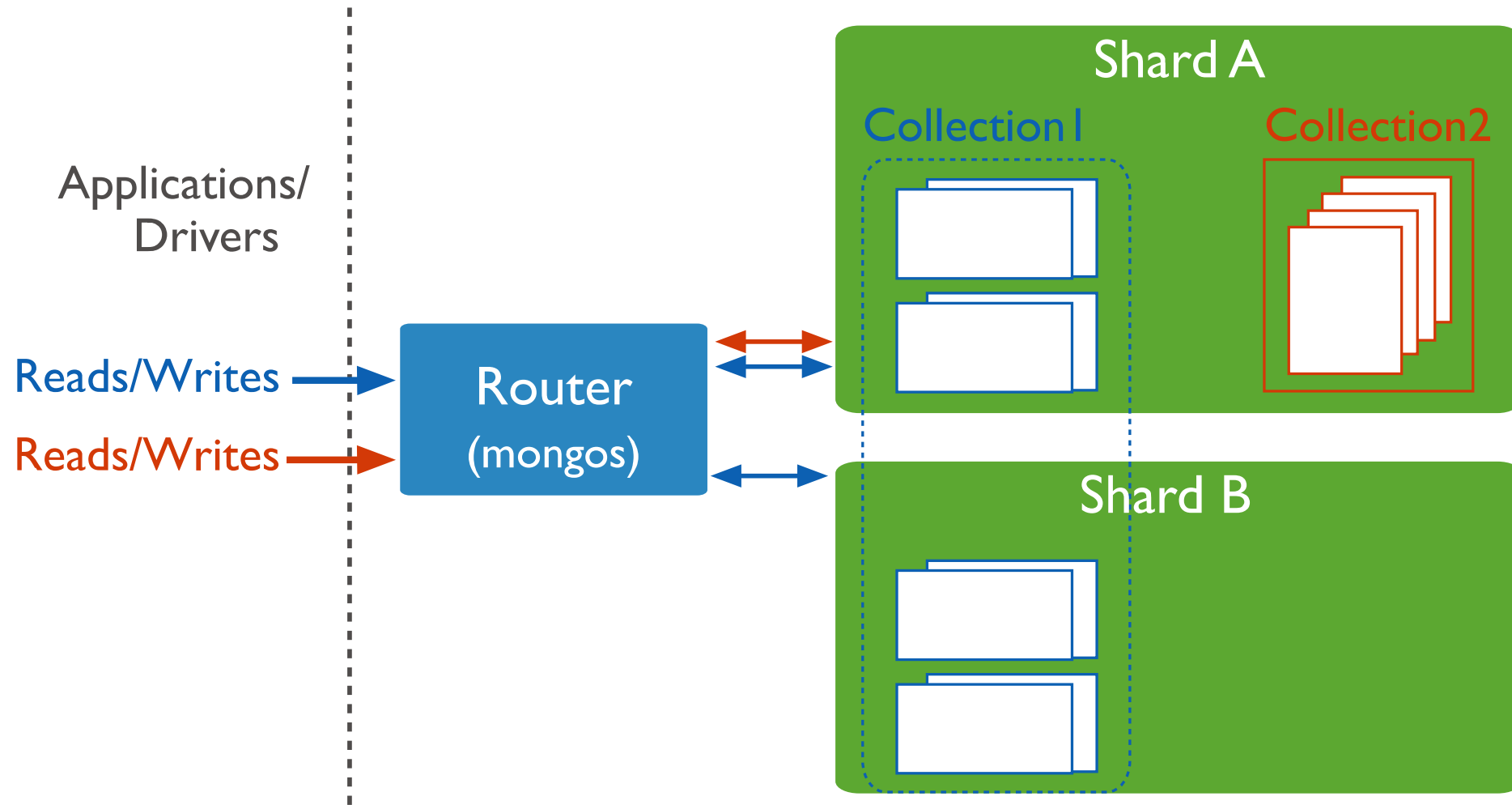
## Principaux composants

- *Sharding* : distribution des blocs de données (*chunk*)
- *Cluster* : géré par 3 types de nœuds (serveurs)
  - Routeurs (*mongos*) : routage des requêtes
  - Serveurs de configuration (*Config Server*) : connaissance du réseau
  - Serveurs de données (*Shard*) : stockage de l'ensemble des données
- Composition minimum d'un cluster : 2 *mongos*, 3 *Config Servers* et 2 *shards*

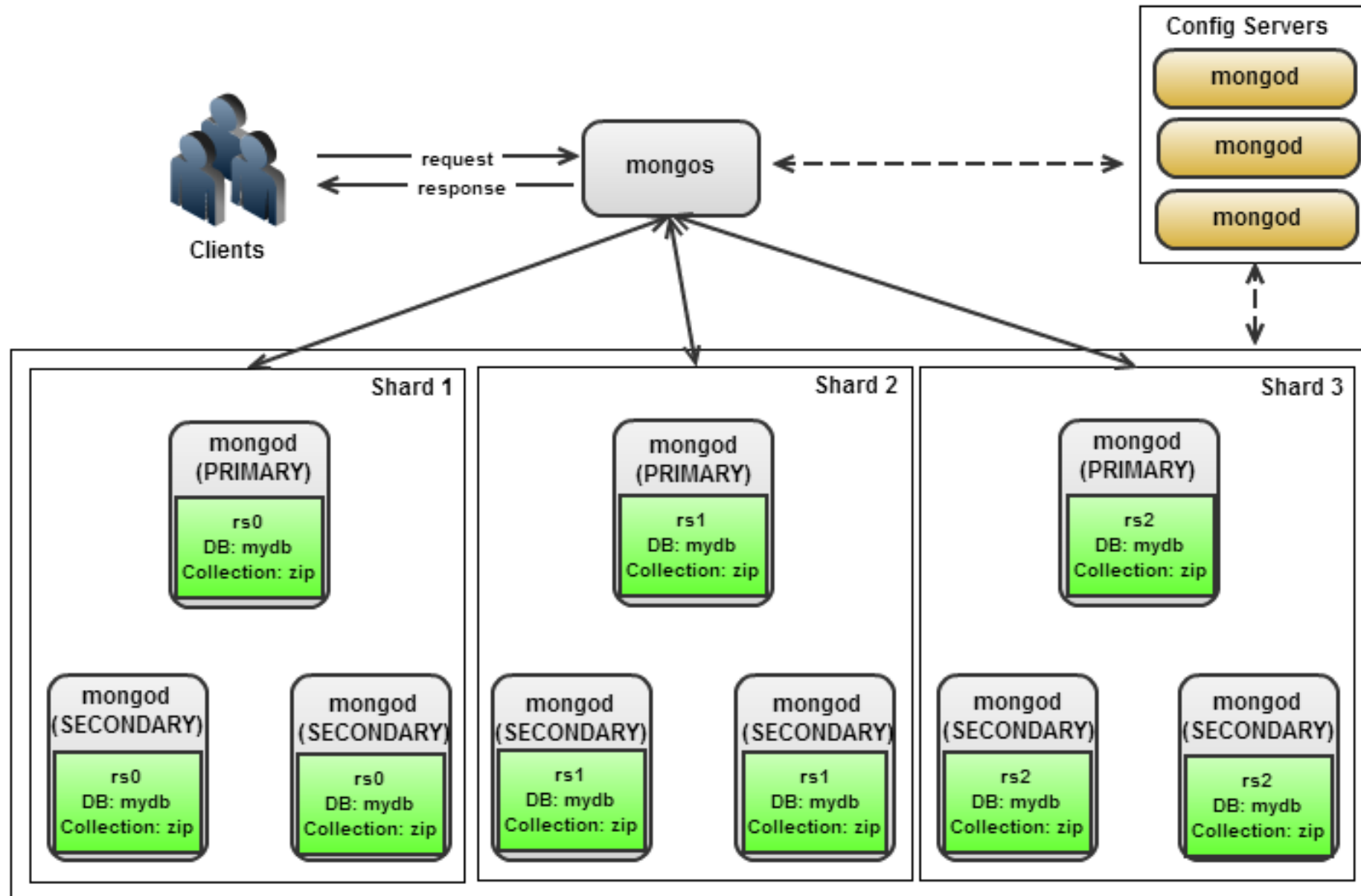
# MongoDB : configuration minimum



# MongoDB : Rôle du *router* et des *shards*

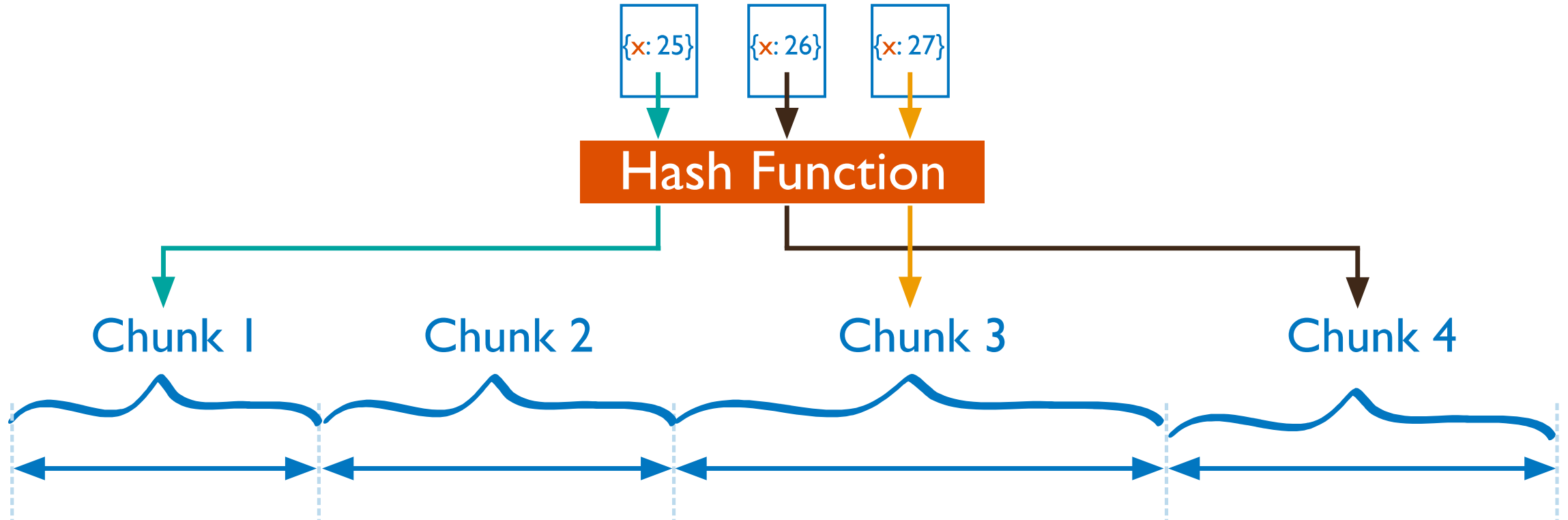


# MongoDB : cluster de *shards*



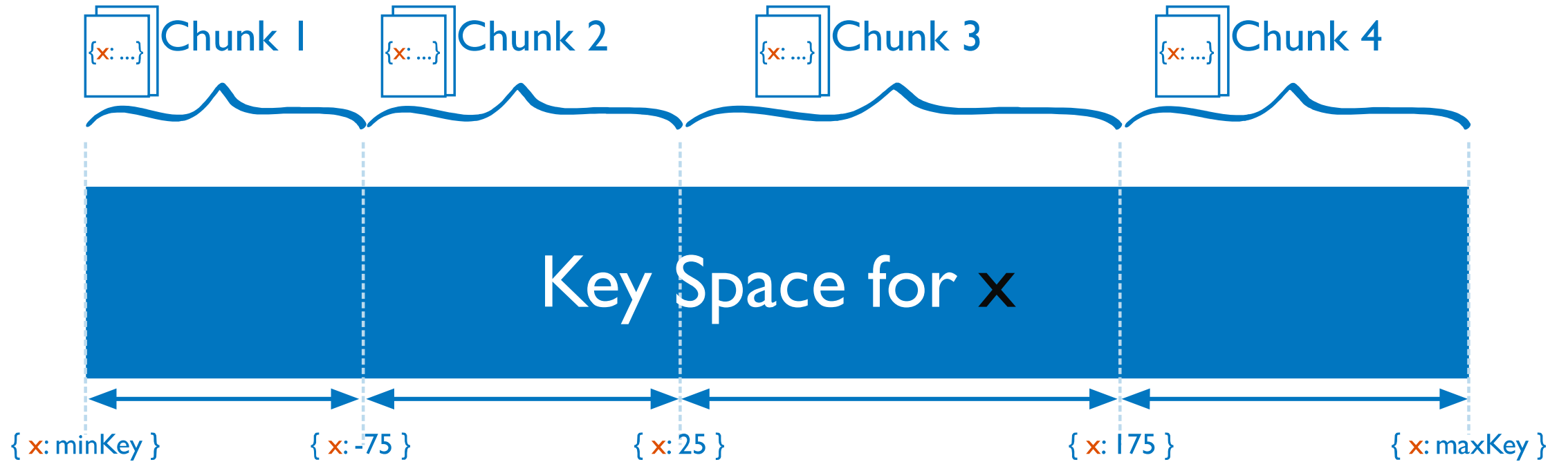
# MongoDB : composition de *shards*

**Shard** : ensemble de blocs de données  
(*chunks*)



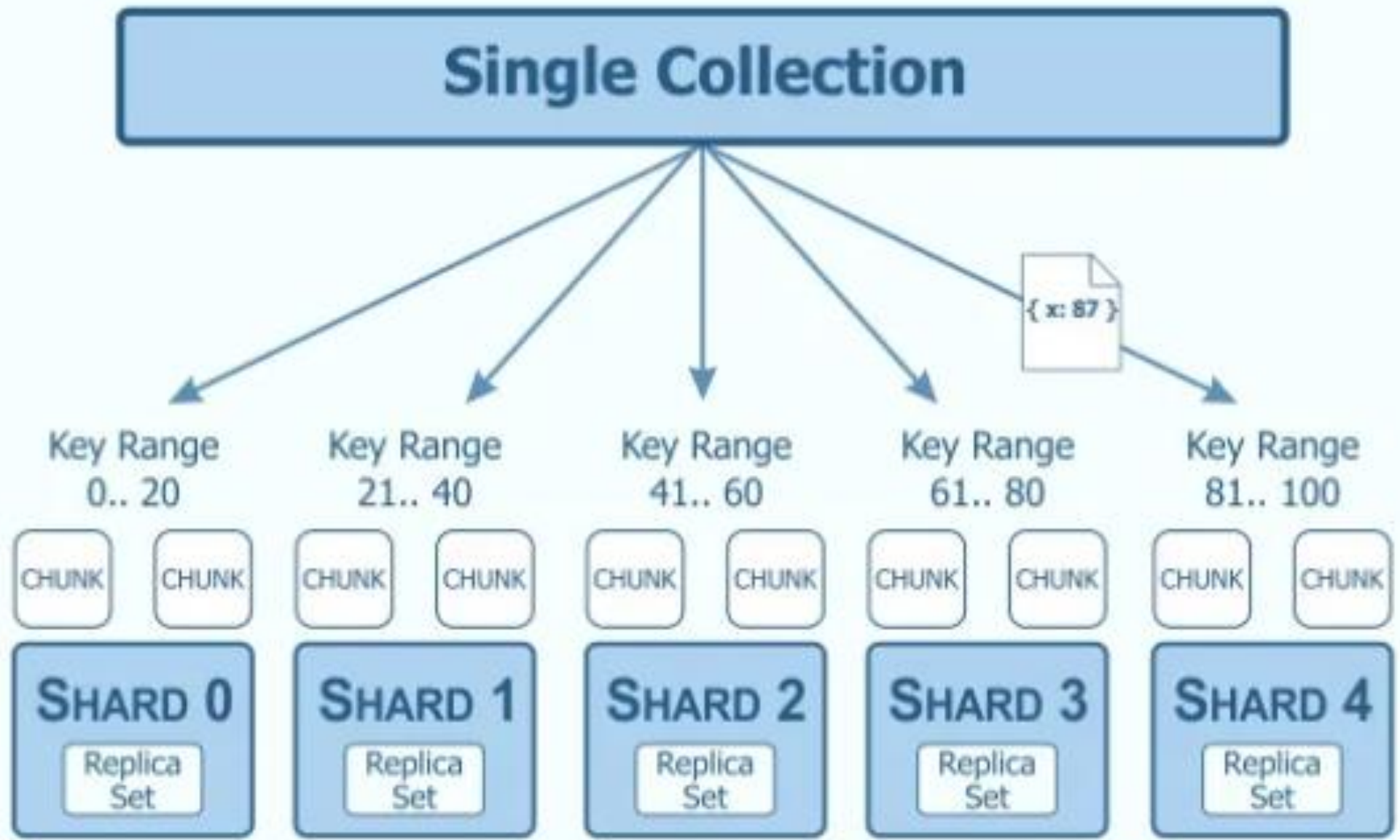
Répartition (*sharding*) des blocs de données (*chunk*) définie par un tri des données

# MongoDB : *Chunk*

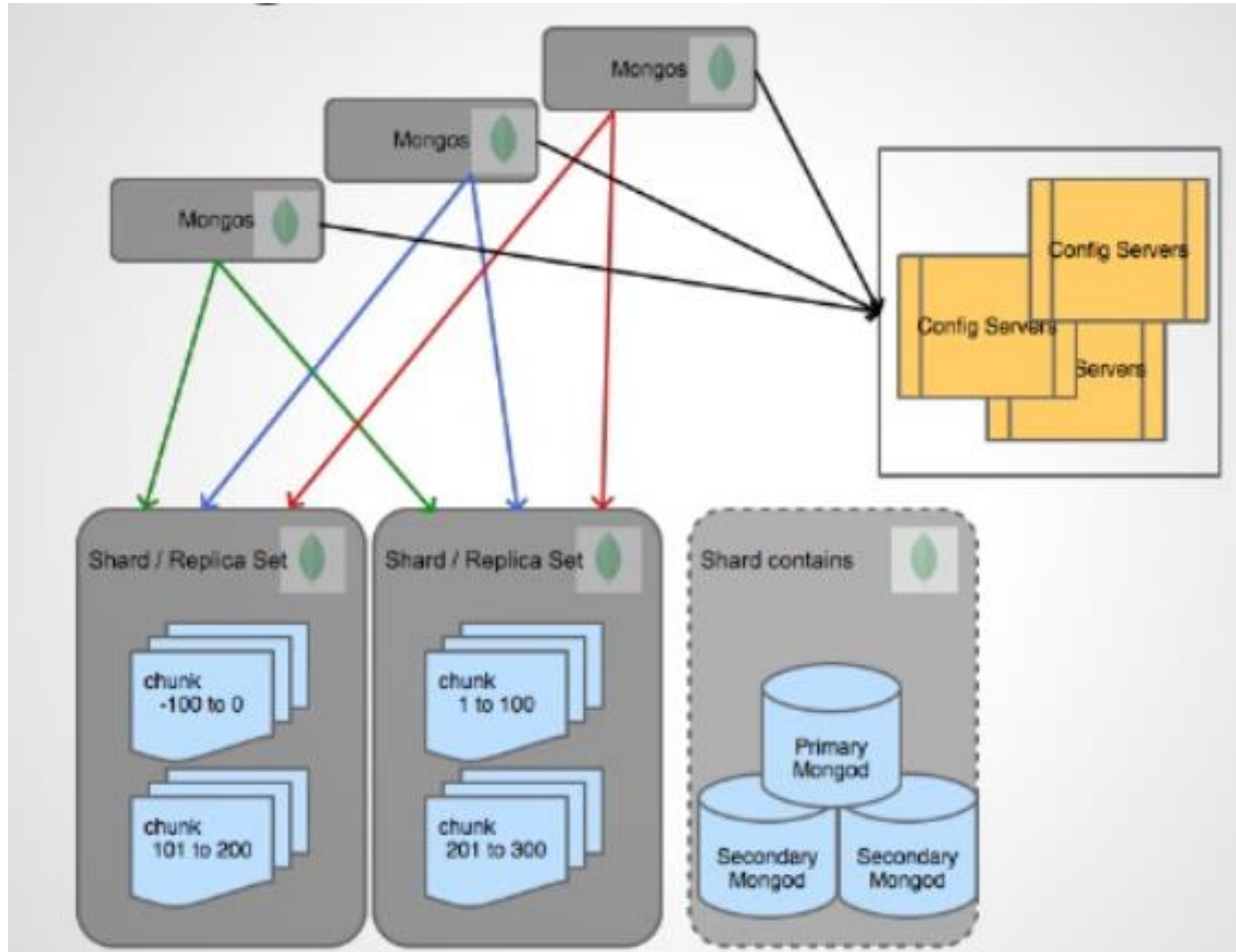




# MongoDB : répartition d'une collection



# MongoDB : répartition (résumé)




# MongoDB : conclusion

 **Flexibilité dans le stockage des données**

 **Montée en charge**

 **Requêtes**

 **Pas de jointure**

 **Transactions (en amélioration)**

 **Gestion de la mémoire**

# CouchBase

- Moteur de recherche distribué, orienté-document et clé-valeur
- Dispose d'un outil de requêtage inspiré de SQL : N1QL
- A vu le jour en juin 2010 sous le nom de *Membase*
- Fusion en 2011 avec celle de *CouchOne* qui supportait *CouchDB*, une base de données de documents basée sur une approche pair-à-pair pour la réplication
- CouchBase : combinaison des parties de *Membase* avec des parties de *CouchDB*












# CouchBase : liens utiles et positionnement

- Site officiel : <https://www.couchbase.com/>
- Cours en ligne : <http://b3d.bdpedia.fr/couchbase.html>
- Tutoriel N1QL en ligne :

<https://query-tutorial.couchbase.com/tutorial/#1>

include secondary database models

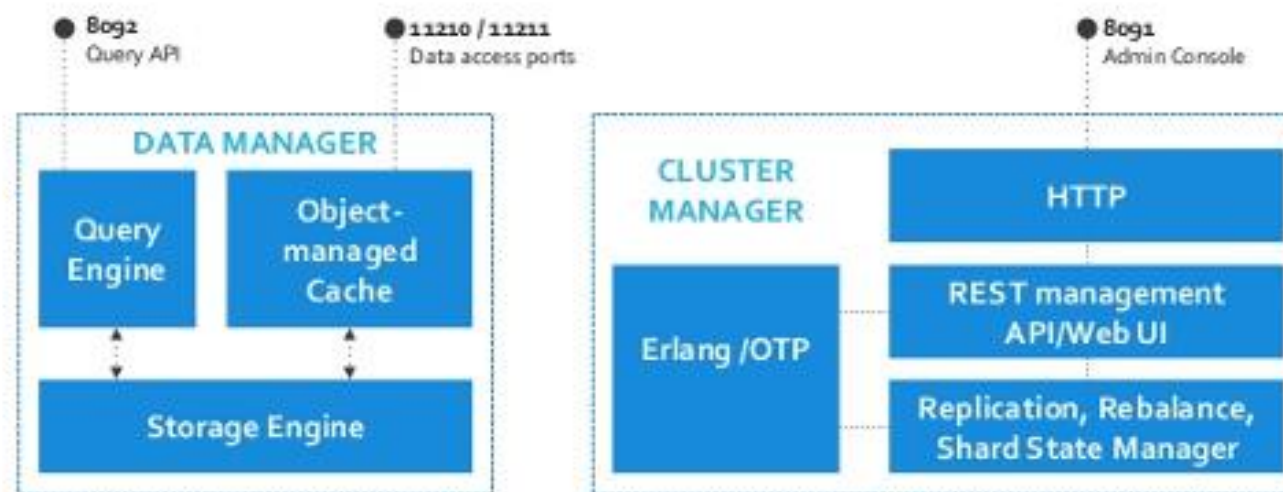
56 systems in ranking, February 2023

Rank			DBMS	Database Model	Score		
Feb 2023	Jan 2023	Feb 2022			Feb 2023	Jan 2023	Feb 2022
1.	1.	1.	MongoDB 	Document, Multi-model 	452.77	-2.42	-35.88
2.	2.	2.	Amazon DynamoDB 	Multi-model 	79.69	-1.87	-0.67
3.	3.		Databricks	Multi-model 	60.33	-0.49	
4.	4.	 3.	Microsoft Azure Cosmos DB 	Multi-model 	36.51	-1.45	-3.45
5.	5.	 4.	Couchbase 	Document, Multi-model 	24.86	-0.44	-5.21

<https://db-engines.com/en/ranking/document+store>

# CouchBase : architecture

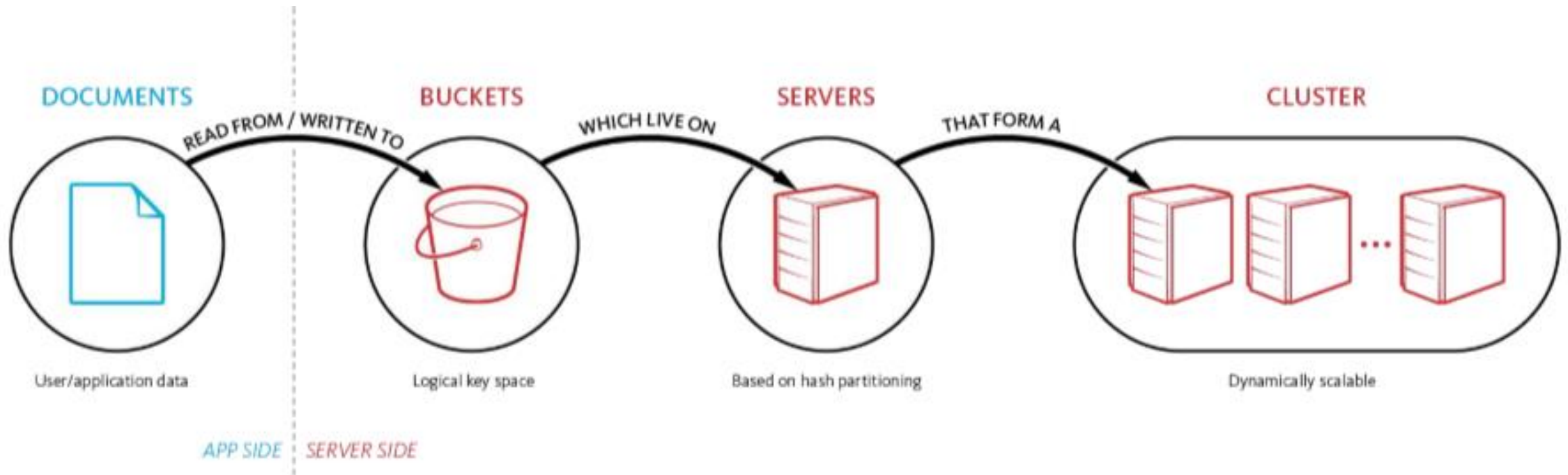
- Fondé sur une architecture SNA pour *Shared-Nothing Architecture*
- Agrégation de serveurs derrière un ou plusieurs serveurs applicatifs frontaux
- Serveur *Couchbase* composé de deux grandes parties : une qui gère l'accès et le stockage des données et l'autre qui gère l'administration du cluster



# CouchBase : principes

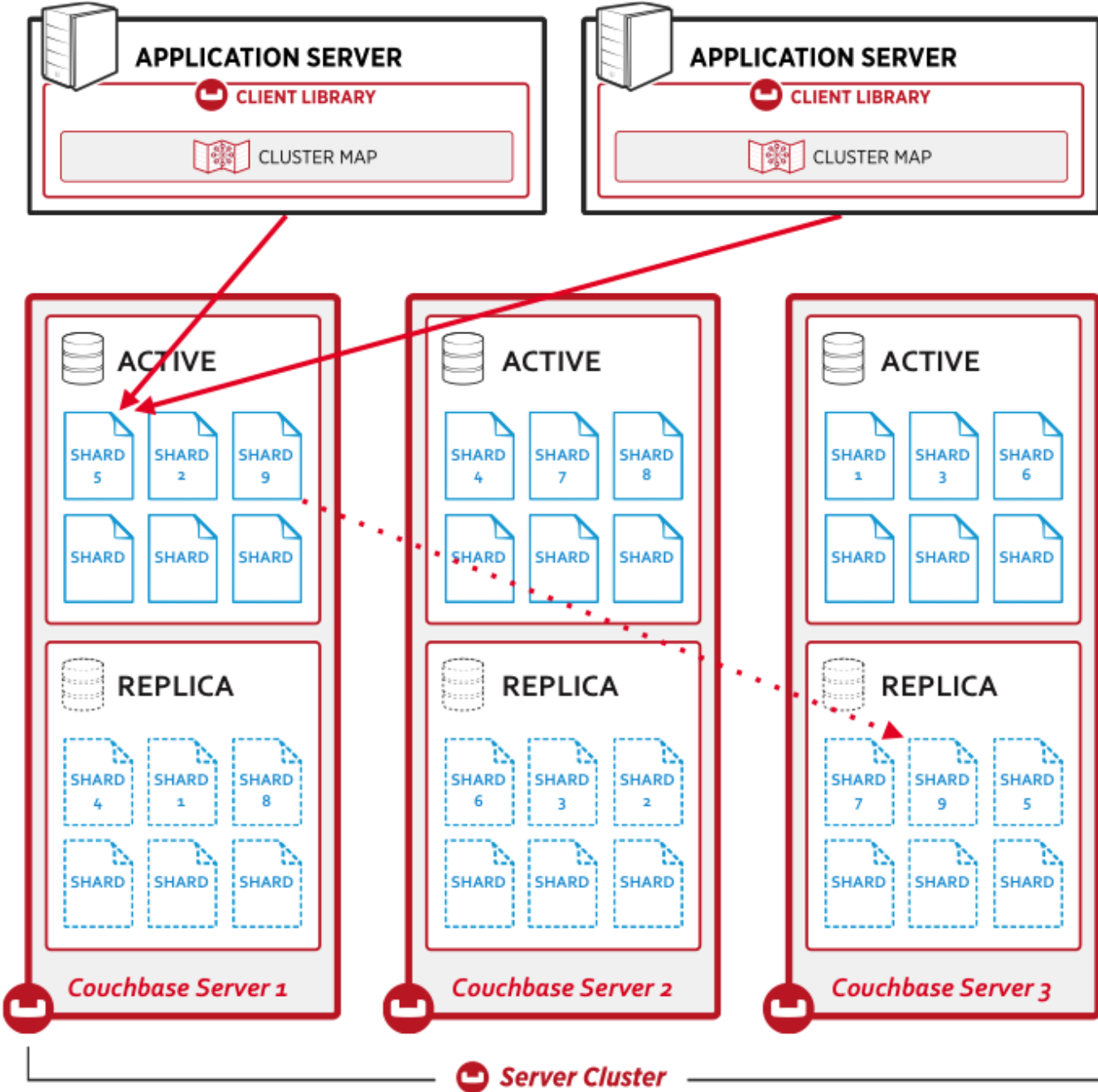
- Sauvegarde des données sous la forme d'*items*
- *Item* : 1 clé et une valeur (binaire ou document JSON)
- Documents distribués uniformément au travers du cluster
- *Items* stockés dans des *buckets* (conteneurs de documents)
- *Bucket* : groupement logique de ressources physiques au sein d'un cluster, associé à de la mémoire RAM pour mettre les données en cache, et à un nombre défini de réplicas
- Découpage de chaque *bucket* en 1024 partitions logiques appelés *vBuckets*
- Partitions *mappées* à des serveurs du cluster
- Données du *mapping* stockées dans le *cluster map*

# CouchBase : organisation des composants





# CouchBase : réplication



# Langage N1QL : exemple de requête renvoyant un document

## Query

```
SELECT *  
FROM tutorial  
WHERE fname = 'Ian'
```

```
{  
  "results": [  
    {  
      "tutorial": {  
        "type": "contact",  
        "title": "Mr.",  
        "fname": "Ian",  
        "lname": "Taylor",  
        "age": 56,  
        "email": "ian@gmail.com",  
        "children": [  
          {  
            "fname": "Abama",  
            "age": 17,  
            "gender": "m"  
          },  
          {  
            "fname": "Bebama",  
            "age": 21,  
            "gender": "m"  
          }  
        ]  
      }  
    ],  
  ]  
}
```

# Langage N1QL : exemple de requête renvoyant un sous-document

```
SELECT children[0].fname AS child_name
FROM tutorial
WHERE fname='Dave'
```

```
"tutorial": {
  "type": "contact",
  "title": "Mr.",
  "fname": "Dave",
  "lname": "Smith",
  "age": 46,
  "email": "dave@gmail.com",
  "children": [
    {
      "fname": "Aiden",
      "age": 17,
      "gender": "m"
    },
    {
      "fname": "Bill",
      "age": 2,
      "gender": "f"
    }
  ],
  "hobbies": [
    "golf",
    "surfing"
  ]
}
```

## Results

```
{
  "results": [
    {
      "child_name": "Aiden"
    }
  ]
}
```

# Langage N1QL : exemple de requête sur les méta-données et les valeurs NULL

```
SELECT META(tutorial) AS meta  
FROM tutorial
```

```
{  
  "results": [  
    {  
      "meta": {  
        "id": "dave"  
      }  
    },  
    {  
      "meta": {  
        "id": "earl"  
      }  
    },  
    {  
      "meta": {  
        "id": "fred"  
      }  
    },  
    {  
      "meta": {  
        "id": "harry"  
      }  
    }  
  ],  
}
```

```
SELECT fname, children  
FROM tutorial  
WHERE children IS NULL
```

```
{  
  "results": [  
    {  
      "children": null,  
      "fname": "Fred"  
    }  
  ]  
}
```

# Langage N1QL : exemple de calculs et d'agrégation

```
|SELECT COUNT(*) AS count  
FROM tutorial
```

```
|SELECT relation, COUNT(*) AS count  
FROM tutorial  
GROUP BY relation
```

```
{  
  "results": [  
    {  
      "count": 6  
    }  
  ]  
}
```

```
{  
  "results": [  
    {  
      "count": 2,  
      "relation": "friend"  
    },  
    {  
      "count": 1,  
      "relation": "coworker"  
    },  
    {  
      "count": 1,  
      "relation": "parent"  
    },  
    {  
      "count": 2,  
      "relation": "cousin"  
    }  
  ]  
}
```

# Langage N1QL : exemple de jointure

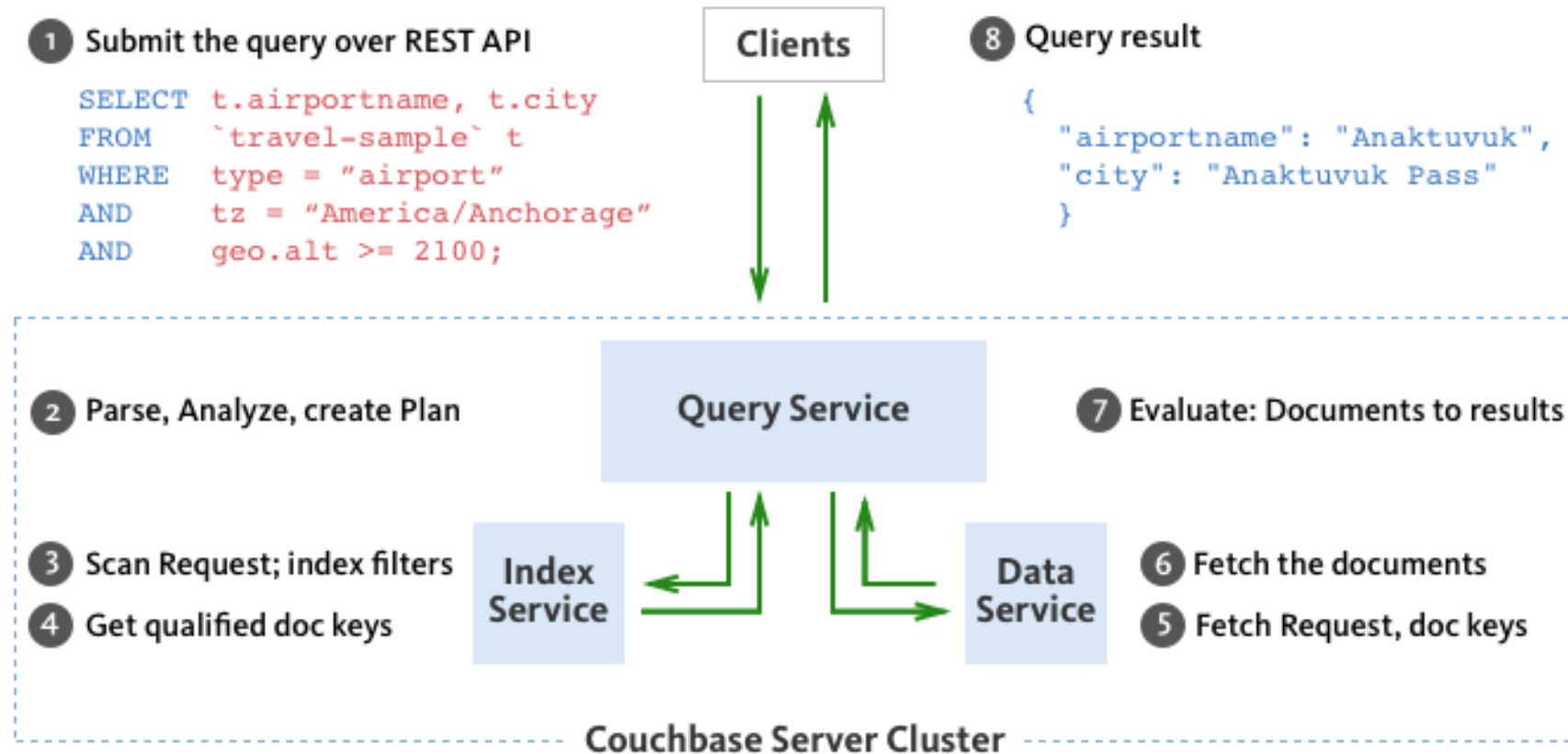
```
{
  "usr": {
    "doc_type": "user_profile",
    "personal_details": {
      "age": 60,
      "display_name": "Elinor Ritchie",
      "email": "Elinor.Ritchie@snailmail.com",
      "first_name": "Elinor",
      "last_name": "Ritchie",
      "state": "Arizona"
    }
  },
  "shipped_order_history": [
    {
      "order_datetime": "Wed May 30 22:00:09 2012",
      "order_id": "T103929516925"
    },
    {
      "order_datetime": "Thu Aug 4 22:00:09 2011",
      "order_id": "T573145204032"
    }
  ]
}
```

```
"orders": {
  "doc_type": "order",
  "order_details": {
    "order_datetime": "Tue Sep 20 18:53:39 2011",
    "order_id": "T499723387040"
  },
  "payment_details": {
    "payment_mode": "Cash On Delivery",
    "total_charges": 33
  },
  "product_details": {
    "currency": "USD",
    "list_price": 31,
    "pct_discount": 10,
    "product_id": "P9928726217",
    "sale_price": 28
  }
}
```

```
SELECT usr.personal_details, orders
FROM users_with_orders usr
USE KEYS "Elinor_33313792"
JOIN orders_with_users orders
ON KEYS ARRAY s.order_id FOR s IN usr
.shipped_order_history END
```

```
"orders": {
  "doc_type": "order",
  "order_details": {
    "order_datetime": "Wed Jun 6 18:53:39 2012",
    "order_id": "T103929516925"
  },
  "payment_details": {
    "payment_mode": "Debit Card",
    "total_charges": 308
  },
  "product_details": {
    "currency": "EUR",
    "list_price": 318,
    "pct_discount": 5,
    "product_id": "P3109994453",
    "sale_price": 303
  },
  "shipping_details": {
    "shipping_charges": 5,
    "shipping_status": "Delivered",
    "shipping type": "Express"
  }
}
```

# CouchBase : exécution d'une requête



# CouchBase : accès au plan d'exécution d'une requête

Query Editor

← history (96/97) →

```
1 SELECT DISTINCT route.destinationairport
2 FROM `travel-sample` airport JOIN `travel-sample` route
3   ON airport.faa = route.sourceairport
4   AND route.type = "route"
5 WHERE airport.type = "airport"
6   AND airport.city = "San Francisco"
7   AND airport.country = "United States";
```

Execute

Explain

✓ explain success | elapsed: 3.76ms | execution: 3.74ms | count: 1 | size: 1914

preferences

Query Results

JSON

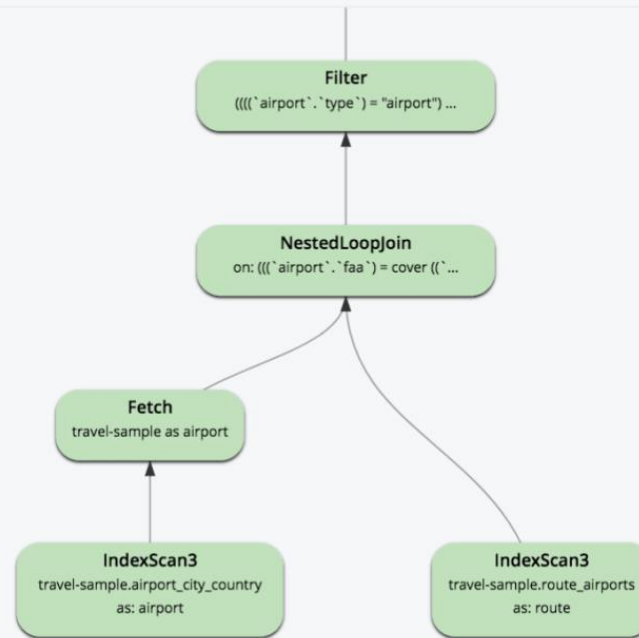
Table

Tree

Plan

Plan Text

Indexes	Buckets	Fields
travel-sample.airport_city_country	travel-sample route	travel-sample.type travel-sample.city travel-sample.country route.destinationairport





# CouchBase vs MongoDB : liens vers de comparaisons

- System Properties Comparison Couchbase vs. MongoDB : <https://db-engines.com/en/system/Couchbase%3BMongoDB>
- Joining JSON: Comparing Couchbase N1QL and MongoDB (2020) : <https://dzone.com/articles/introduction-to-couchbase-for-mongodb-developers-a-1>
- Benchmarking NoSQL Databases (2019) : <https://www.datanami.com/2019/01/16/benchmarking-nosql-databases/>

# CouchBase vs MongoDB : exemple de requête

## MySQL

```
SELECT ac.industry,
SUM(CASE WHEN a.activitytype = 'Task'
THEN 1 ELSE 0 END ) task,
SUM(CASE WHEN a.activitytype
='Appointment'
THEN 1 ELSE 0 END ) appts
FROM crm.activity a
INNER JOIN crm.account ac
ON (a.accid = ac.id)
WHERE a.startdate BETWEEN '2018-10-01'
AND '2018-12-31'
GROUP BY ac.industry
```

## Couchbase - N1QL

```
SELECT ac.industry,
SUM(CASE WHEN a.activityType = 'Task'
THEN 1 ELSE 0 END) task,
SUM(CASE WHEN a.activityType =
'Appointment'
THEN 1 ELSE 0 END ) appts
FROM crm a
INNER JOIN crm ac ON a.accid = ac.id
AND ac.type='account'
WHERE a.type='activity'
AND a.startDate BETWEEN '2018-10-01'
AND '2018-12-31'
GROUP BY ac.industry
```

## MongoDB Query Language

```
db.activity.aggregate(
{ $match: { startDate: { $gt: '2018-01-01',
                        $lt: '2018-12-31' } } },
{
  $lookup: {
    from: "account",
    localField: "accid",
    foreignField: "id",
    as: "account_docs"
  }
},
{ $match: { "account_docs": { $ne: [] } } },
{ $unwind: "$account_docs" },
{
  $project: {
    item: 1,
    task: { $cond: {if:
      { $eq: ["$activityType", "Task"] }, then: 1,
      else: 0 } },
    appt: { $cond: { if:
      { $eq: ["$activityType", "Appointment"] }, then:
1,
      else: 0 } } }
  },
  {
    $group: {
      _id: "$account_docs.industry",
      tasks: { $sum: "$task" },
      appointments: { $sum: "$appt" }
    }
  }
});
```

# ElasticSearch

- Moteur de recherche distribué, orienté-document
- Capable de stocker des documents (format JSON)
- Dispose d'un langage de requêtes
- Basé sur :
  - *Apache Lucene* pour l'indexation et la recherche des données
  - *Logstash* pour la gestion de *logs*
  - *Kibana* pour l'exploration et la visualisation des données

# Positionnement d'ElasticSearch

410 systems in ranking, March 2023

Rank			DBMS	Database Model	Score		
Mar 2023	Feb 2023	Mar 2022			Mar 2023	Feb 2023	Mar 2022
1.	1.	1.	Oracle +	Relational, Multi-model <i>i</i>	1261.29	+13.77	+9.97
2.	2.	2.	MySQL +	Relational, Multi-model <i>i</i>	1182.79	-12.66	-15.45
3.	3.	3.	Microsoft SQL Server +	Relational, Multi-model <i>i</i>	922.01	-7.08	-11.77
4.	4.	4.	PostgreSQL +	Relational, Multi-model <i>i</i>	613.83	-2.67	-3.10
5.	5.	5.	MongoDB +	Document, Multi-model <i>i</i>	458.78	+6.02	-26.88
6.	6.	6.	Redis +	Key-value, Multi-model <i>i</i>	172.45	-1.39	-4.31
7.	7.	7.	IBM Db2	Relational, Multi-model <i>i</i>	142.92	-0.04	-19.22
8.	8.	8.	Elasticsearch	Search engine, Multi-model <i>i</i>	139.07	+0.47	-20.88

include secondary database models

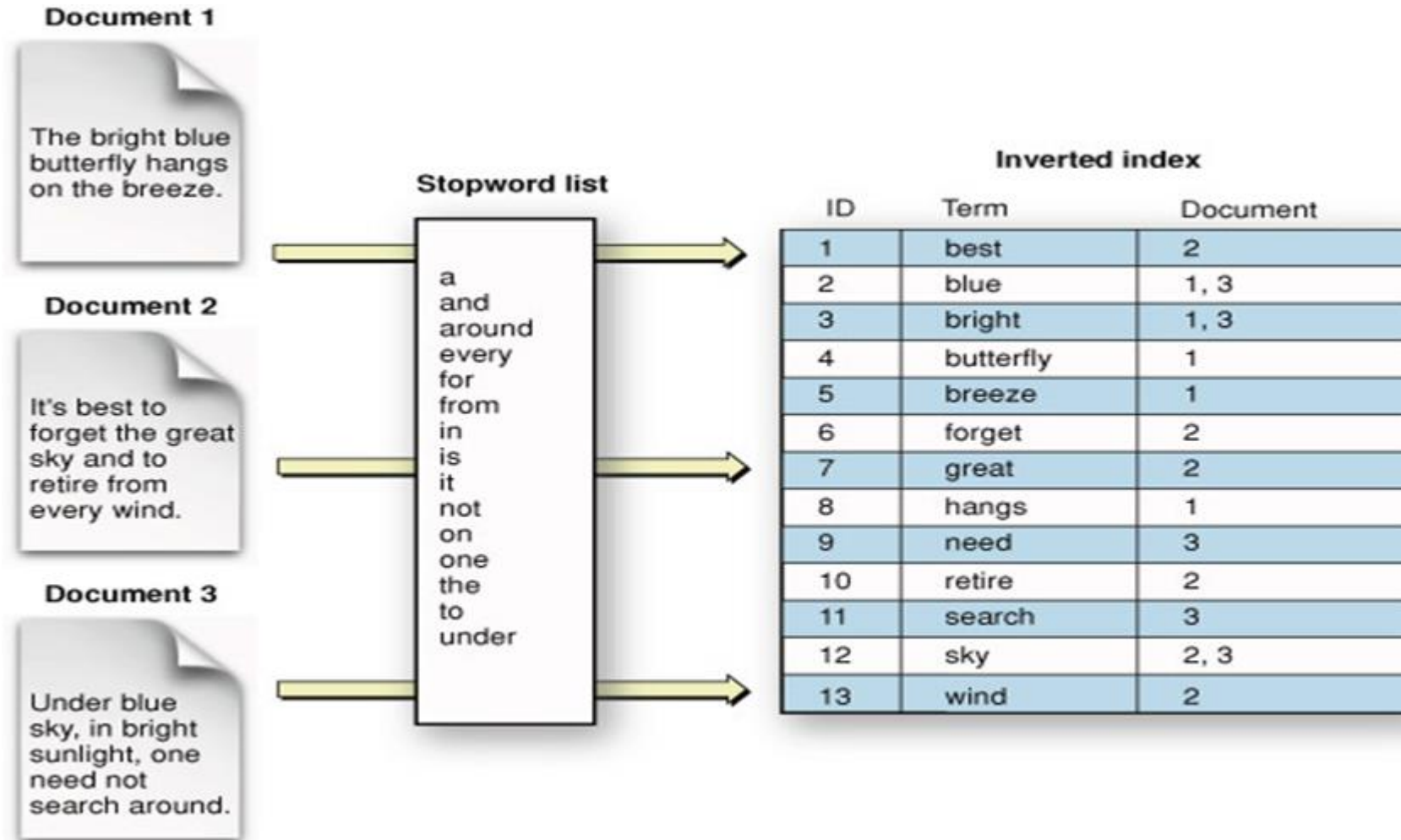
28 systems in ranking, March 2023

Rank			DBMS	Database Model	Score		
Mar 2023	Feb 2023	Mar 2022			Mar 2023	Feb 2023	Mar 2022
1.	1.	1.	Elasticsearch	Search engine, Multi-model <i>i</i>	139.07	+0.47	-20.88
2.	2.	2.	Splunk	Search engine	87.97	+0.89	-7.39
3.	3.	3.	Solr	Search engine, Multi-model <i>i</i>	47.28	+1.40	-11.77
4.	4.	↑ 10.	OpenSearch +	Search engine	11.12	+0.62	+6.95

# ElasticSearch : liens utiles

- **Site officiel** : <https://www.elastic.co/fr/products/elasticsearch>
- **Cours en ligne** :
  - <https://openclassrooms.com/en/courses/4462426-maitrisez-les-bases-de-donnees-nosql/4474691-etudiez-le-fonctionnement-d-elasticsearch>
  - <https://juvenal-chokogoue.developpez.com/tutoriels/elasticsearch-sgbd-nosql/>
  - <https://www.linkedin.com/learning/l-essentiel-de-elasticsearch/chercher-des-documents?u=74606242>
- **Comparaison ElasticSearch/MongoDB** :  
<https://www.slideshare.net/sebprunier/mongodb-et-elasticsearch-meilleurs-ennemis>

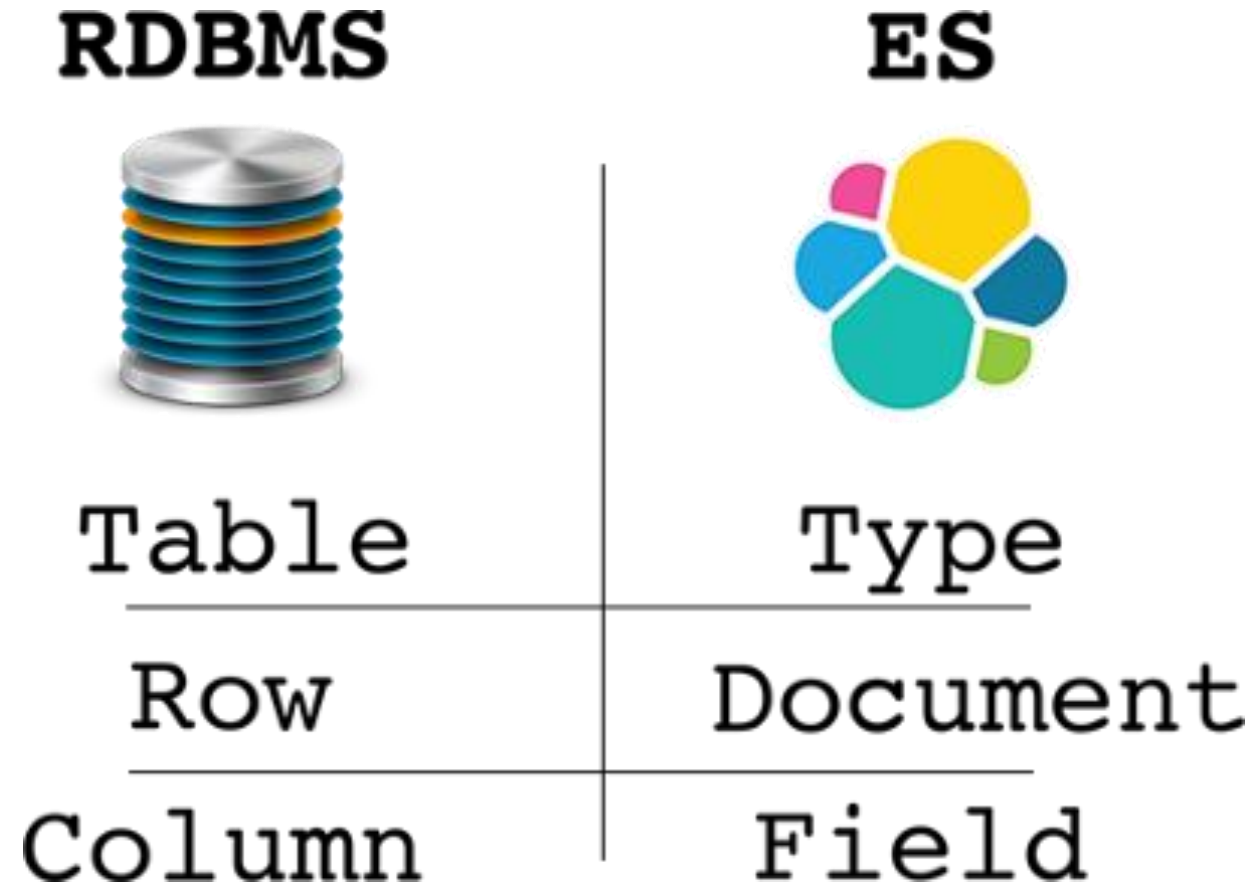
# ElasticSearch ~index inversé



# ElasticSearch : modèle de données (1/2)

- Unité ElasticSearch : document
- Index ElasticSearch : ensemble des documents qui sont stockés
- **Tous** les documents JSON ont le même schéma (même ordre de clés, mêmes types de données, et le même ordre)
- Chaque document est préfixé par un autre document JSON contenant les informations d'import : l'**index** et l'**identifiant** du document

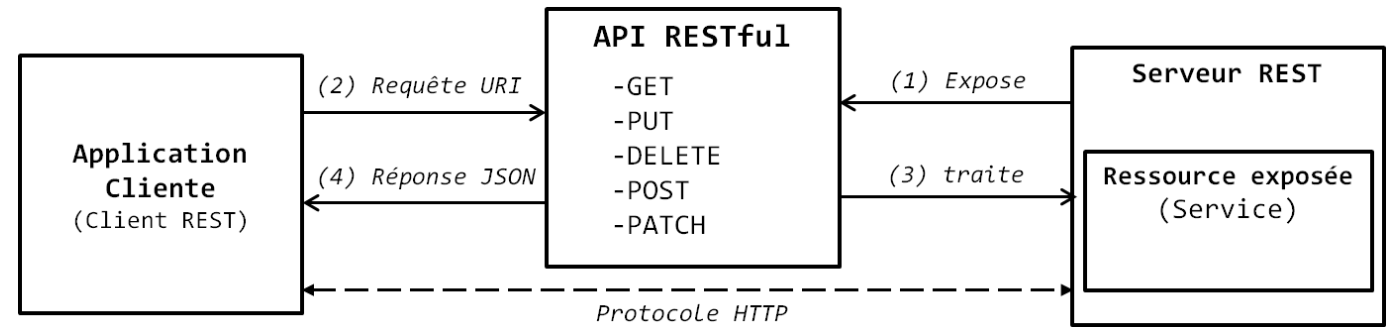
# ElasticSearch : modèle de données (2/2)





# ElasticSearch : utilisation

- A l'aide d'une API REST

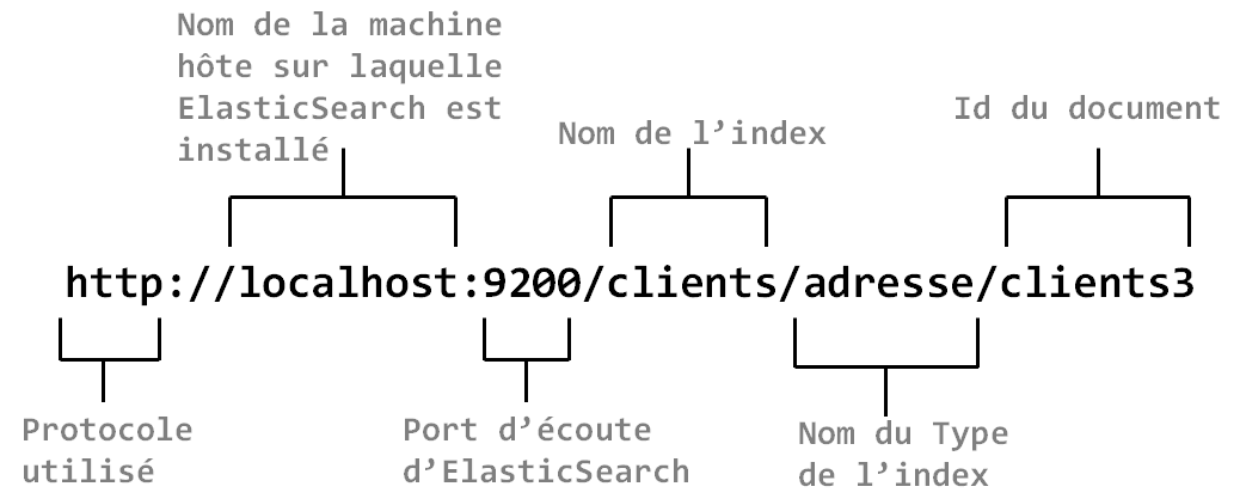


- Index: ressource exposée

- ElasticSearch : rôle du Serveur REST

- Manipulation des documents de l'index à l'aide des verbes HTTP

- JSON : format d'échange de données



# ElasticSearch : exemple de document JSON

## Document JSON

```
POST temp_index/_doc
{
  "name": "Pineapple",
  "botanical_name": "Ananas comosus",
  "produce_type": "Fruit",
  "country_of_origin": "New Zealand",
  "date_purchased": "2020-06-02T12:15:35",
  "quantity": 200,
  "unit_price": 3.11,
  "description": "a large juicy tropical fruit consisting of aromatic ec",
  "vendor_details": {
    "vendor": "Tropical Fruit Growers of New Zealand",
    "main_contact": "Hugh Rose",
    "vendor_location": "Whangarei, New Zealand",
    "preferred_vendor": true
  }
}
```

## Index associé

```
1 {
2   "_index" : "temp_index",
3   "_type" : "_doc",
4   "_id" : "Z5RLyXkBz8GVH0EAFLb6",
5   "_version" : 1,
6   "result" : "created",
7   "_shards" : {
8     "total" : 2,
9     "successful" : 2,
10    "failed" : 0
11  },
12   "_seq_no" : 1,
13   "_primary_term" : 1
14 }
15
```

# ElasticSearch : exemple de mapping

```
GET temp_index/_mapping
```

```
{
  "temp_index" : {
    "mappings" : {
      "properties" : {
        "botanical_name" : {
          "type" : "text",
          "fields" : {
            "keyword" : {
              "type" : "keyword",
              "ignore_above" : 256
            }
          }
        },
        "country_of_origin" : {
          "type" : "text",
          "fields" : {
            "keyword" : {
              "type" : "keyword",
              "ignore_above" : 256
            }
          }
        }
      }
    }
  }
}
```

Field name → "quantity" : {  
                                  "type" : "long"  
                                  },  
                                  "unit\_price" : {  
  "type" : "float"  
  },  
                                  }

Field type ← "long"  
                                  "float"

# ElasticSearch : indexation des chaînes

By default, strings get mapped twice as text and keyword

```
PUT temp_index/_doc/4
{
  "country": "New Zealand"
}

PUT temp_index/_doc/5
{
  "country": "New Zealand"
}

PUT temp_index/_doc/6
{
  "country": "New Caledonia"
}
```

Text  
Text analysis  
Inverted Index

Used for full text search

term	doc id
caledonia	6
new	4,5,6
zealand	4,5

Keyword  
Doc values

Used for aggregations, sorting, exact searches

doc id	doc values
4	New Zealand
5	New Zealand
6	New Caledonia

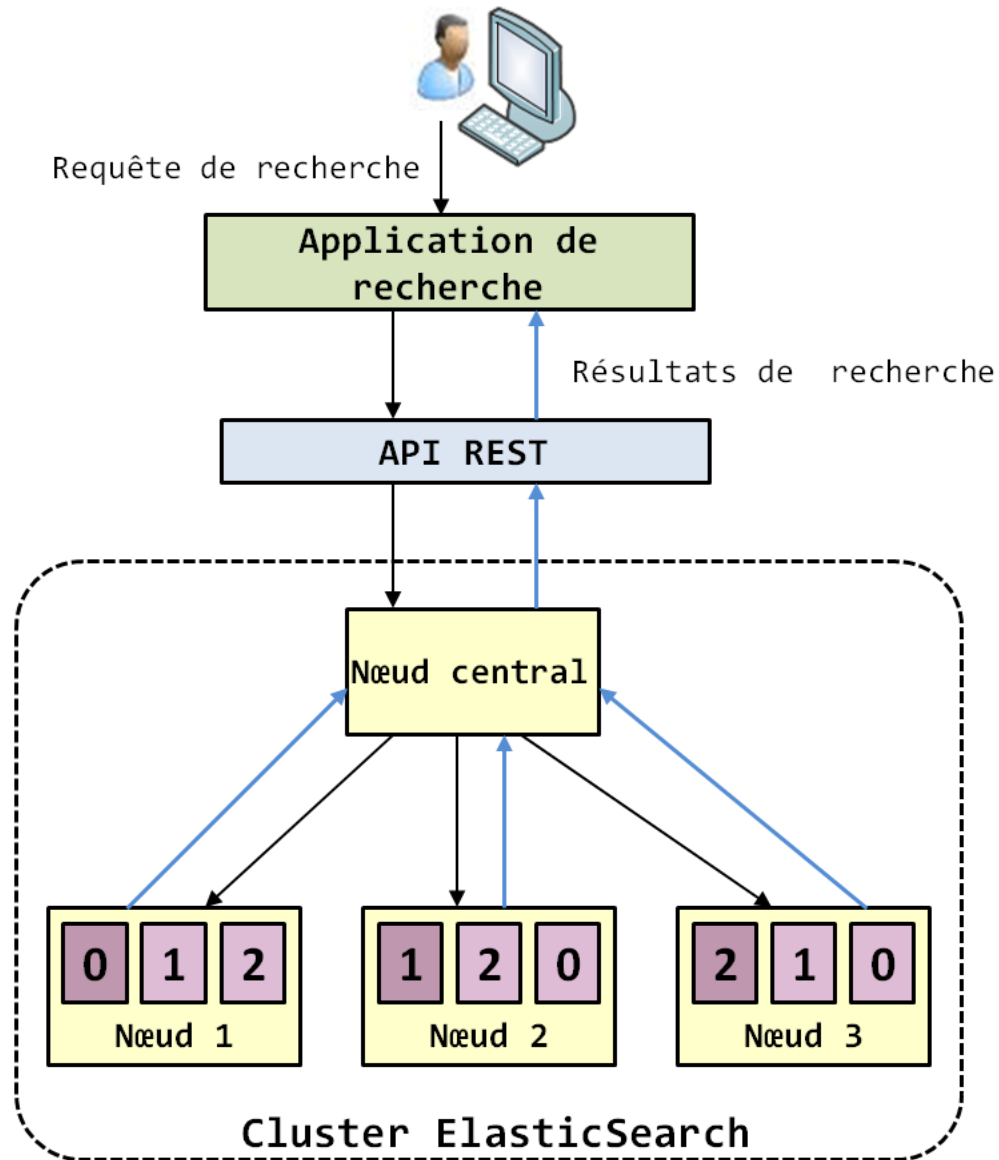
@LisaHJung | Beginner's Crash Course to Elastic Stack

# ElasticSearch : création d'un index

```
PUT my_index ❶
{
  "mappings": {
    "doc": { ❷
      "properties": { ❸
        "title": { "type": "text" }, ❹
        "name": { "type": "text" }, ❺
        "age": { "type": "integer" }, ❻
        "created": {
          "type": "date", ❼
          "format": "strict_date_optional_time||epoch_millis"
        }
      }
    }
  }
}
```

- ❶ Create an index called `my_index`.
- ❷ Add a mapping type called `doc`.
- ❸ Specify fields or *properties*.
- ❹ Specify that the `title` field contains `text` values.
- ❺ Specify that the `name` field contains `text` values.
- ❻ Specify that the `age` field contains `integer` values.
- ❼ Specify that the `created` field contains `date` values in two possible formats.

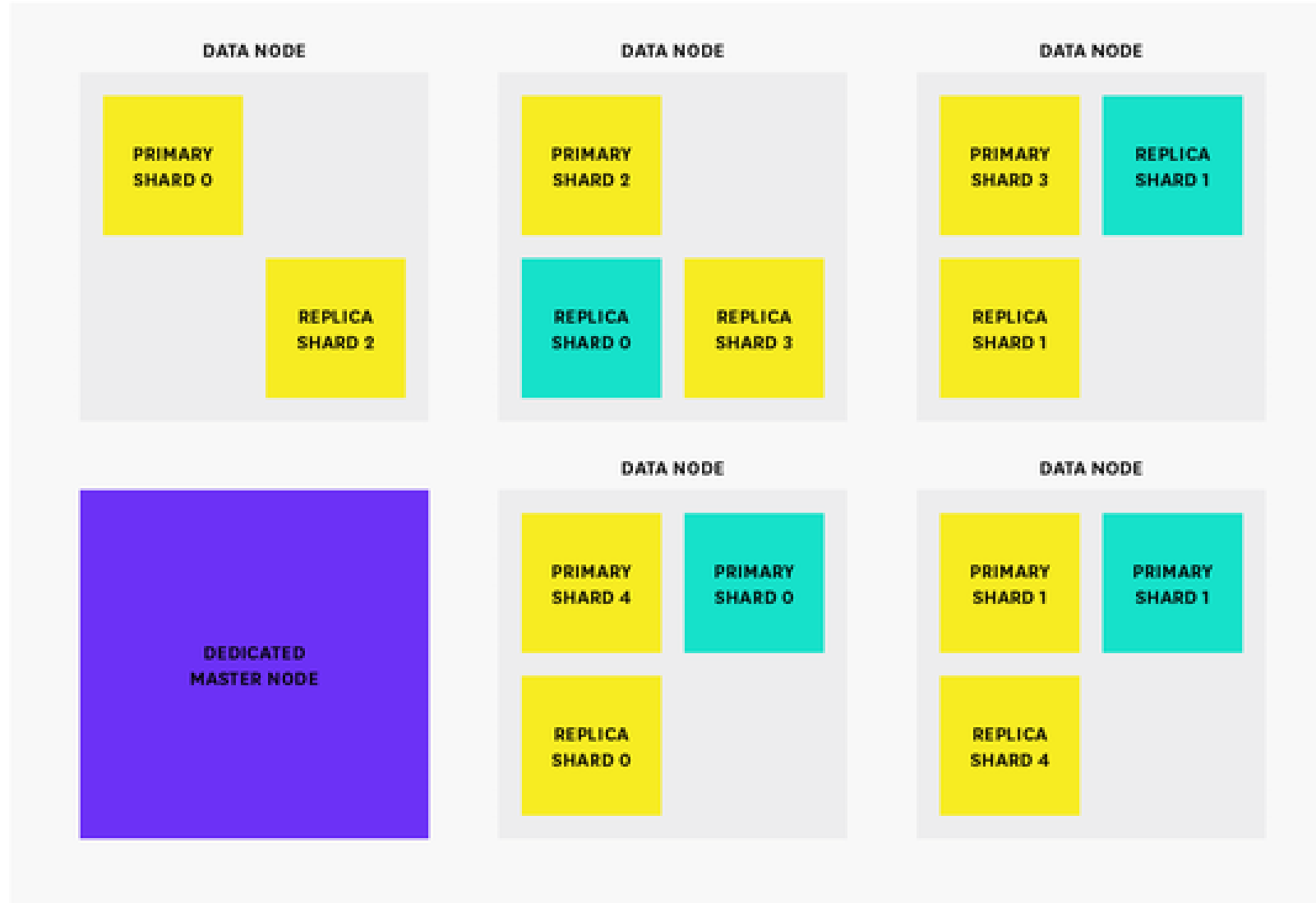
# ElasticSearch : Architecture



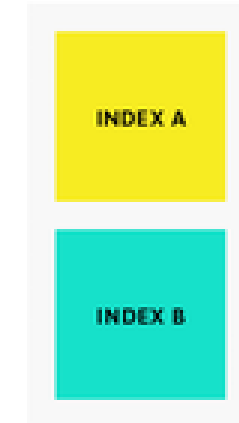
- Partition des documents
- Partitions répliquées entre les nœuds du cluster selon un facteur de réplication
- Réplication des données pour assurer la haute disponibilité du cluster et la garantie de la continuité des traitements en cas de panne dans le cluster
- Désignation d'une réplique primaire pour chaque document

# ElasticSearch : Architecture

Cluster



Legend



# ElasticSearch : Requête en DSL (*Domain Specific Language*)

Verbe HTTP                      Index                      Type                      Action

```
curl -XPOST http://localhost:9200/clients/adresse/_search -d'
{
  "query": {
    "match": {
      "firstname": "juvenal"
    }
  }
}
```

Corps de la requête

« **curl** » pour envoyer des requêtes HTTP




# ElasticSearch : interrogation en SQL

Console Search Profiler Grok Debugger

```
1 POST _sql?format=txt
2 {
3   "query": "select customer_first_name, customer_full_name,
4     customer_gender from kibana_sample_data_ecommerce where
5     customer_gender='MALE'"
6 }
```

	customer_first_name	customer_full_name	customer_gender
3	Eddie	Eddie Underwood	MALE
4	Eddie	Eddie Weber	MALE
5	Oliver	Oliver Rios	MALE
6	Abd	Abd Sutton	MALE
7	Eddie	Eddie Gregory	MALE
8	Sultan Al	Sultan Al Pratt	MALE
9	Eddie	Eddie Wolfe	MALE
10	Sultan Al	Sultan Al Thompson	MALE
11	Sultan Al	Sultan Al Boone	MALE
12	George	George Hubbard	MALE
13	Boris	Boris Maldonado	MALE
14	Yahya	Yahya Rivera	MALE
15	Muniz	Muniz Riley	MALE

Nom de l'index à interroger

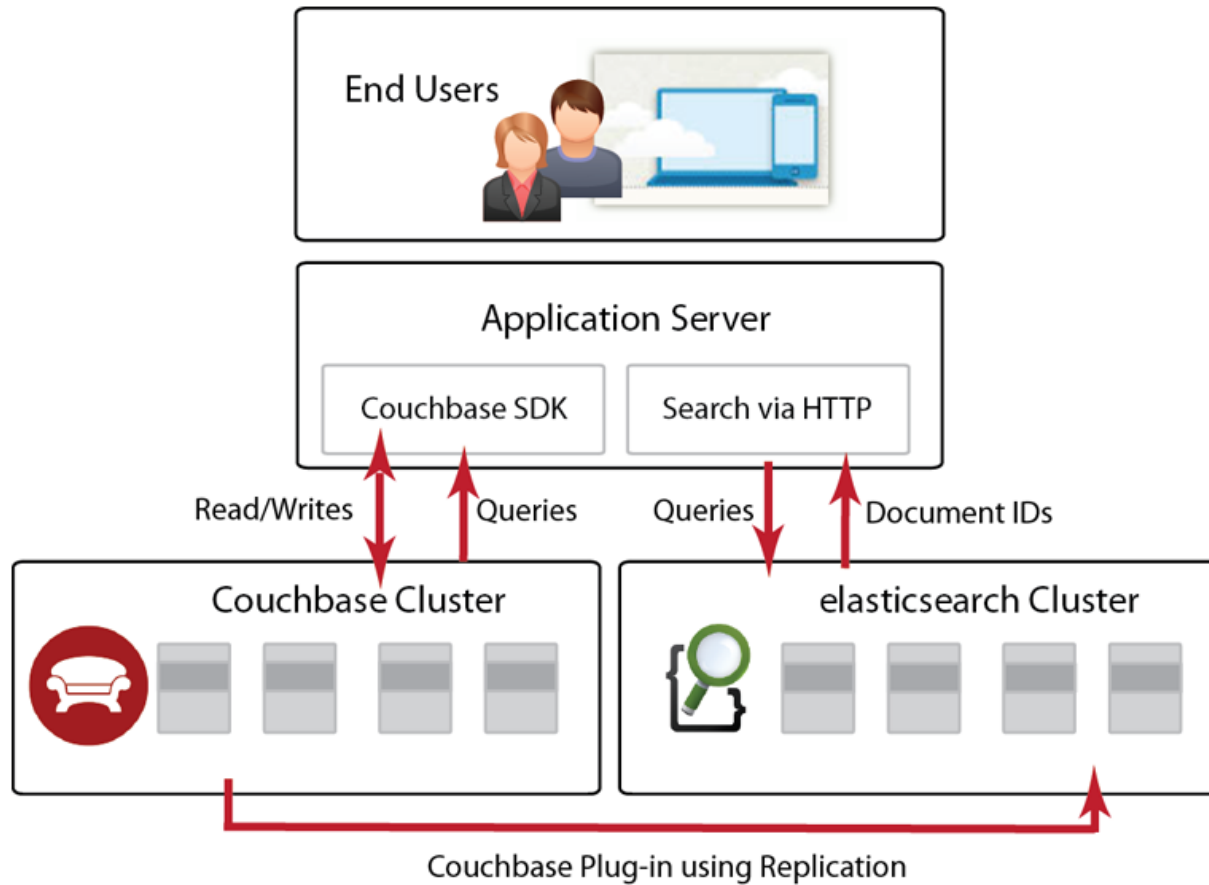


# ElasticSearch : traduction SQL en DSL

```
1 POST /_sql/translate
2 {
3   "query": "select customer_first_name,
4     customer_full_name, customer_gender
5     , products.product_name, sum
6     (taxful_total_price) as CA from
7     kibana_sample_data_ecommerce where
8     customer_gender='MALE' group by
9     customer_first_name,
10    customer_full_name, customer_gender
11    , products.product_name"
12 }
13
14
15
16
17
18
19
20
21
22
23
24
25
26
```

```
1 {
2   "size" : 0,
3   "query" : {
4     "term" : {
5       "customer_gender" : {
6         "value" : "MALE",
7         "boost" : 1.0
8       }
9     }
10  },
11  "_source" : false,
12  "stored_fields" : "_none_",
13  "aggregations" : {
14    "groupby" : {
15      "composite" : {
16        "size" : 1000,
17        "sources" : [
18          {
19            "1033" : {
20              "terms" : {
21                "field" : "customer_first_name.keyword",
22                "missing_bucket" : true,
23                "order" : "asc"
24              }
25            }
26          ]
27        }
28      }
29    }
30  }
31 }
```

# Combinaison Couchbase - Elasticsearch

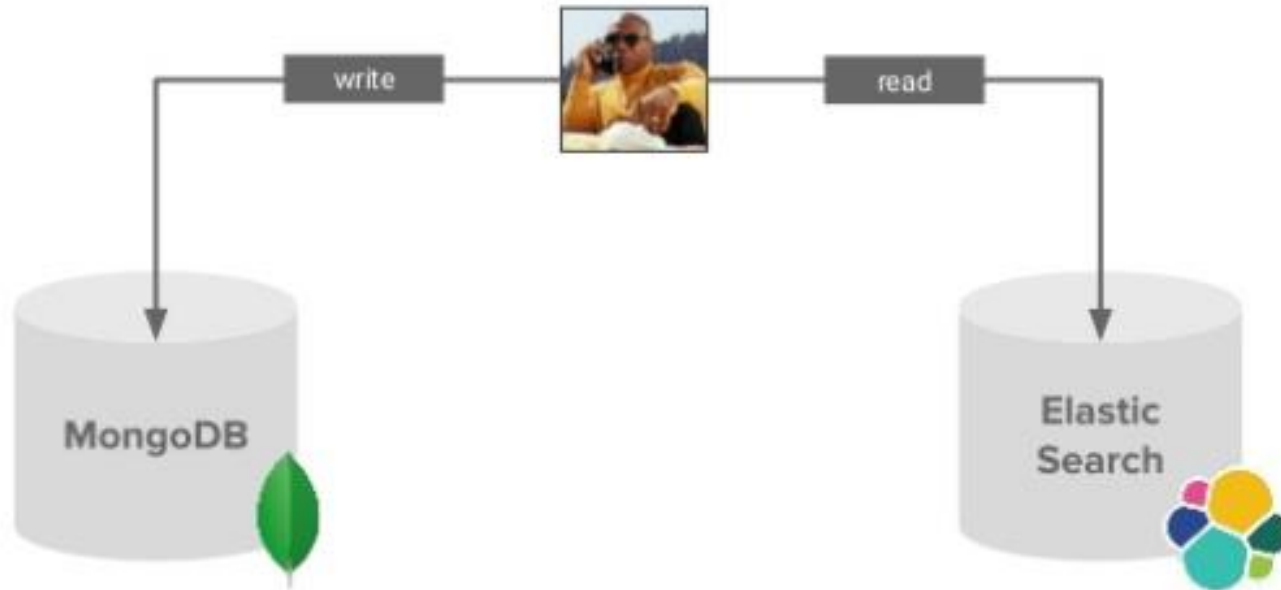


- Après une écriture sur Couchbase : réplication sur Elasticsearch
- Pas de copie intégrale sur Elasticsearch, juste pour créer l'index
- Récupération de l'ID du doc recherché via Elasticsearch et du doc complet via CouchBase

# Combinaison MongoDB - Elasticsearch

**Ecrire** dans MongoDB

**Lire** et **rechercher** dans Elasticsearch



- 5 Different ways to synchronize data from MongoDB to Elasticsearch :

<https://code.likeagirl.io/5-different-ways-to-synchronize-data-from-mongodb-to-elasticsearch-d8456b83d44f>

- MongoDB invite ses clients à se passer d'Elasticsearch (2019) :

<https://www.lemagit.fr/actualites/252476841/MongoDB-invite-ses-clients-a-se-passer-dElasticsearch>

# JSON et bases de données relationnelles

- **SQL/JSON** Intégration de JSON dans le standard SQL (2017 - Part 6) :  
<https://www.iso.org/standard/67367.html> et [https://webstore.iec.ch/preview/info\\_isoiec19075-6%7Bed1.0%7Den.pdf](https://webstore.iec.ch/preview/info_isoiec19075-6%7Bed1.0%7Den.pdf)
- Gestion de données JSON possible dans plusieurs SGBD relationnels :
  - MySQL : <https://dev.mysql.com/doc/refman/8.0/en/json.html>
  - MariaDB : <https://mariadb.com/kb/en/json-data-type/>
  - Oracle : <https://docs.oracle.com/database/121/ADXDB/json.htm#ADXDB6246>
  - SQLSever : <https://docs.microsoft.com/fr-fr/sql/relational-databases/json/json-data-sql-server?view=sql-server-ver15>
  - PostgreSQL : <https://docs.postgresql.fr/14/functions-json.html#FUNCTIONS-JSON-PROCESSING>

# JSON sous PostgreSql

- 2 types de données : JSON et JSONB (binaire avec indexation sur les clés)
- Opérateurs :
  - -> pour renvoyer la clé de la colonne JSON
  - ->> pour renvoyer la valeur de la colonne JSON
  - @> et <@ pour tester si un document est un sous ou un sur-ensemble d'un autre
  - ? Pour tester l'existence d'une clé
  - cf. <https://devhints.io/postgresql-json>,  
<https://www.postgresqltutorial.com/postgresql-tutorial/postgresql-json/> et  
<https://www.postgresql.org/docs/9.5/functions-json.html>

# Exemple de données JSON sous PostgreSQL (1/3)

```
CREATE TABLE json_test (  
  id serial primary key,  
  data jsonb  
);  
  
INSERT INTO json_test (data) VALUES  
  ('{}'),  
  ('{"a": 1}'),  
  ('{"a": 2, "b": ["c", "d"]}'),  
  ('{"a": 1, "b": {"c": "d", "e": true}}'),  
  ('{"b": 2}');
```

id	data
1	{}
2	{"a":1}
3	{"a":2, "b":["c", "d"]}
4	{"a":1, "b":{"c":"d", "e":true}}
5	{"b":2}

Documents ayant un champ 'a' :

```
SELECT * FROM json_test WHERE data ? 'a';
```

id	data
2	{"a":1}
3	{"a":2, "b":["c", "d"]}
4	{"a":1, "b":{"c":"d", "e":true}}

# Exemple de données JSON sous PostgreSQL (2/3)

Id des documents et valeur du champ 'a' :

```
SELECT id, data->'a' as a FROM json_test ;
```

id	a
1	null
2	1
3	2
4	1
5	null

Documents ayant un champ 'a' de valeur 1 :

```
SELECT * FROM json_test WHERE data @> '{"a":1}';
```

id	data
2	{"a":1}
4	{"a":1, "b":{"c":"d", "e":true}}



# Exemple de données JSON sous PostgreSQL (3/3)

Documents ayant un champ 'a' de valeur supérieur à 1 :

```
SELECT * FROM json_test WHERE data ->> 'a' > '1';
```

id	data
3	{"a":2,"b":["c","d"]}

Valeur du champ 'e' du champ 'b' :

```
SELECT data -> 'b' ->> 'e' as e FROM json_test where id = 4;
```

e
true

Documents ayant un champ 'a' ou 'b' :

```
SELECT * FROM json_test WHERE data ?| array['a', 'b'];
```

id	data
2	{"a":1}
3	{"a":2,"b":["c","d"]}
4	{"a":1,"b":{"c":"d","e":true}}
5	{"b":2}

Documents ayant un champ 'a' et un champ 'b' :

```
SELECT * FROM json_test WHERE data ?& array['a', 'b'];
```

id	data
3	{"a":2,"b":["c","d"]}
4	{"a":1,"b":{"c":"d","e":true}}

# Exemple d'agrégation de données JSON sous PostgreSQL (1/2)

```
CREATE TABLE events (  
  name varchar(200),  
  visitor_id varchar(200),  
  properties jsonb,  
  browser jsonb  
);
```

name	visitor_id	properties	browser
pageview	1	{"page":"/"}	{"os":"Mac","name":"Chrome","resolution":{"x":1440,"y":900}}
pageview	2	{"page":"/"}	{"os":"Windows","name":"Firefox","resolution":{"x":1920,"y":1200}}
pageview	1	{"page":"/account"}	{"os":"Mac","name":"Chrome","resolution":{"x":1440,"y":900}}
purchase	5	{"amount":10}	{"os":"Windows","name":"Firefox","resolution":{"x":1024,"y":768}}
purchase	15	{"amount":200}	{"os":"Windows","name":"Firefox","resolution":{"x":1280,"y":800}}
purchase	15	{"amount":500}	{"os":"Windows","name":"Firefox","resolution":{"x":1280,"y":800}}

# Exemple d'agrégation de données JSON sous PostgreSQL (2/2)

```
SELECT browser->>'name' AS browser,  
       count(browser)  
FROM events  
GROUP BY browser->>'name';
```

browser	count
Firefox	4
Chrome	2

```
SELECT visitor_id, SUM(CAST(properties->>'amount' AS integer)) AS total  
FROM events  
WHERE CAST(properties->>'amount' AS integer) > 0  
GROUP BY visitor_id;
```

visitor_id	total
15	700
5	10

```
SELECT AVG(CAST(browser->'resolution'->>'x' AS integer)) AS width,  
       AVG(CAST(browser->'resolution'->>'y' AS integer)) AS height  
FROM events;
```

width	height
1397.33333333	894.66666666

# JSON Path (1/3)

Langage de recherche pour JSON introduit dans le standard SQL:2016 et implémenté sous PostgreSQL

- <https://paquier.xyz/postgres-12/postgres-12-jsonpath/>
- <https://www.postgresql.org/docs/devel/functions-json.html#FUNCTIONS-SQLJSON-FILTER-EX-TABLE>
- <https://jsonpath.com/>

```
CREATE TABLE personnes (datas jsonb );

INSERT INTO personnes (datas) VALUES ( '
{
  "firstName": "Jean",
  "lastName": "Valjean",
  "address": {
    "streetAddress": "43 rue du Faubourg Montmartre",
    "city": "Paris",
    "postalCode": "75002"
  },
  "phoneNumbers": [
    { "number": "06 12 34 56 78" },
    { "type": "bureau",
      "number": "07 89 10 11 12" }
  ],
  "children": [],
  "spouse": null
}
' );

INSERT INTO personnes (datas) VALUES ( '
{
  "firstName": "Georges",
  "lastName": "Durand",
  "address": {
    "streetAddress": "27 rue des Moulins",
    "city": "Châteauneuf",
    "postalCode": "45990"
  },
  "phoneNumbers": [
    { "number": "06 21 34 56 78" },
    { "type": "bureau",
      "number": "07 98 10 11 12" }
  ],
  "children": [],
  "spouse": null
}
' );
```

# JSON Path (2/3)

```
SELECT jsonb_path_query(datas, '$.firstName') FROM personnes;
```

jsonb_path_query
------------------

Jean
------

Georges
---------

```
SELECT jsonb_path_query(datas, '$.address.city') FROM personnes;
```

jsonb_path_query
------------------

Paris
-------

Châteauneuf
-------------

```
SELECT jsonb_path_query (datas, '$.phoneNumbers[*]')  
FROM personnes ;
```

jsonb_path_query
------------------

{"number":"06 12 34 56 78"}
-----------------------------

{"type":"bureau","number":"07 89 10 11 12"}
---

{"number":"06 21 34 56 78"}
-----------------------------

{"type":"bureau","number":"07 98 10 11 12"}
---

# JSON Path (3/3)

```
SELECT jsonb_path_query (datas, '$.phoneNumbers[*] ? (@.type == "bureau") ')  
FROM personnes ;
```

jsonb_path_query
{"type":"bureau","number":"07 89 10 11 12"}
{"type":"bureau","number":"07 98 10 11 12"}

```
SELECT jsonb_path_query (datas, '$.phoneNumbers[*] ? (@.type == "bureau") ') -> 'number' as number  
FROM personnes ;
```

number
07 89 10 11 12
07 98 10 11 12

# JSON sous PostgreSQL : conclusion

 **Combinaison relationnel et documents JSON**

 **Gestion des transactions**

 **Gestion des clés étrangères et des jointures**

 **Pas de réel schéma flexible**

 **Pas un moteur NoSQL**

**Performance par rapport à MongoDB ?**

**cf.** Article scientifique de 2020 comparant les 2 moteurs pour l'utilisation de données géographiques :

<https://link.springer.com/content/pdf/10.1007/s10707-020-00407-w.pdf>

# JSON sous PostgreSql vs MongoDB

- [https://portavita.github.io/2018-10-31-blog A JSON use case comparison between PostgreSQL and MongoDB/](https://portavita.github.io/2018-10-31-blog-A-JSON-use-case-comparison-between-PostgreSQL-and-MongoDB/)
- [« Postgresql – la nouvelle base de données orientée document »  
https://www.youtube.com/watch?v=iOwnUPSY0KI](https://www.youtube.com/watch?v=iOwnUPSY0KI)
- <https://blog.ippon.fr/2017/01/03/rex-bdx-io-postgres-notonlysql/>
- <https://naiveskill.com/mongodb-vs-postgresql/>
- <https://hevodata.com/learn/mongodb-vs-postgresql/>
- <https://www.openlogic.com/blog/postgresql-vs-mongodb#:~:text=MongoDB%20Performance,an%20advantage%20for%20both%20data bases>
- [https://info.enterprisedb.com/rs/069-ALB-339/images/PostgreSQL MongoDB Benchmark-WhitepaperFinal.pdf](https://info.enterprisedb.com/rs/069-ALB-339/images/PostgreSQL-MongoDB-Benchmark-WhitepaperFinal.pdf)