VIETNAM FRANCE UNIVERSITY

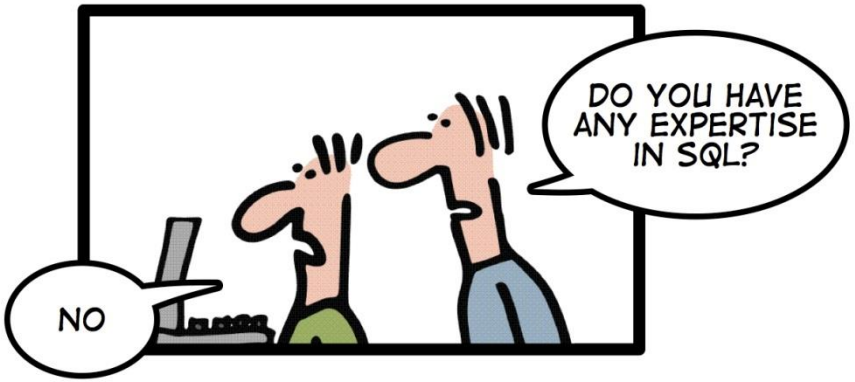
ADVANCED DATABASES

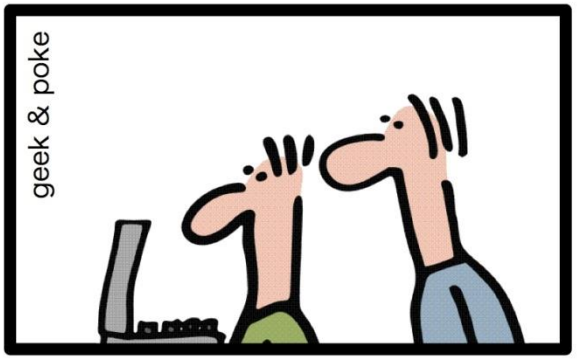
Distributed and high performance NoSQL database systems

Dr. NGUYEN Hoang Ha

Email: nguyen-hoang.ha@usth.edu.vn

HOW TO WRITE A CV





I have no SQL experience
But I am good at NO SQL

Leverage the NoSQL boom

Agenda

- NoSQL Overview
- Introduction to MongoDB
- MongoDb Sharding

Relational database

Customers

ID	lname	fname	address	city	state	zip
1122	Brust	Andrew	123 Main St.	New York	NY	10099
3214	Doe	John	321 Elm St.	Anytown	MI	40001

Orders

ID	custid	amount	tax	shipdate
3001	1122	500	40	2/20/2012
3002	1122	250	17	3/18/2012
3003	3214	1700	150	3/20/2012

A NoSQL example

Customers

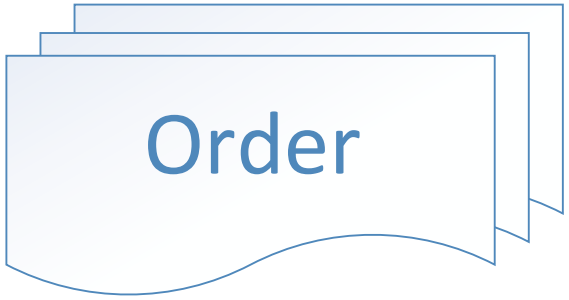
ID: 1122	**lname:** Brust	**fname:** Andrew	**address:** 123 Main St.	**city:** New York	**state:** NY	**zip:** 10099
ID: 3214	**lname:** Doe	**fname:** John	**address:** 321 Waterford Crescent.	**village:** Stodday	**county:** Lancashire	**postal code:** LA2 6ET

Orders

ID: 3001	**customerID:** 1122	**amount:** 500	**tax:** 40	**processdate:** 2/20/2012
ID: 3002	**customerID:** 1122	**amount:** 250	**shipdate:** 3/18/2012	
ID: 3003	**amount:** 1700	**tax:** 150		

NoSQL: Documents and Collections

Customer collections



Order collections

Web Scale

- This term used to justify NoSQL
- Millions concurrent users
  - Amazon
  - Google
- Non-transactional tasks
  - Loading catalog
  - Environment preferences

What is NoSQL?

- NoSQL is a class of database management system identified by its non-adherence to the widely used relational database management system (RDBMS) model with its structured query language (SQL).
- NoSQL has evolved to mean "Not Only" SQL
- NoSQL has become prominent with the advent of web scale data and systems created by Google, Facebook, Amazon, Twitter and others to manage data for which SQL was not the best fit.

NoSQL Definition

From www.nosql-database.org:

> Next Generation Databases mostly addressing some of the points: being **non-relational, distributed, open-source** and **horizontal scalable**. The original intention has been **modern web-scale databases**. The movement began early 2009 and is growing rapidly. Often more characteristics apply as: **schema-free, easy replication support, simple API, eventually consistent** / **BASE** (not ACID), a **huge data amount**, and more.

Beginning NoSQL

- One of first uses of the phrase NoSQL is due to Carlo Strozzi, circa 1998.
- Characteristics:
  - A fast, portable, open-source RDBMS
  - A derivative of the RDB database system
  - Not a full-function DBMS, per se, but a shell-level tool
  - User interface: Unix shell
  - Strozzi's NoSQL RDBMS was based on the relational model
  - Does not have an SQL interface → NoSQL means "no sql" i.e., we are not using the SQL language.

NoSQL Today

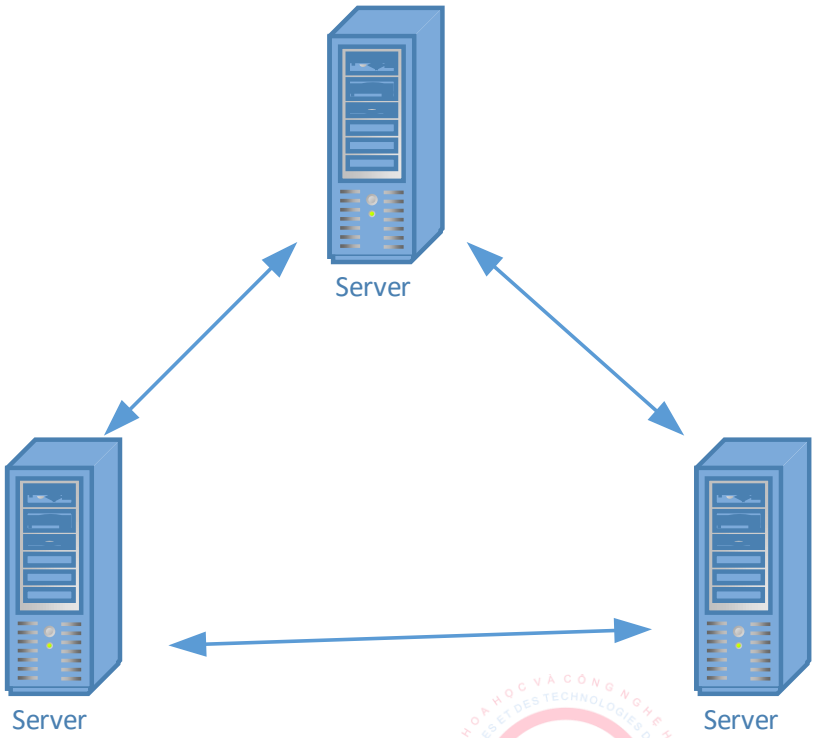
- More recently:
  - The term has taken on different meanings
  - One common interpretation is "not only SQL"
- Most modern NoSQL systems diverge from the relational model or standard RDBMS functionality:
- The data model: not based on relations
- The query model: not relational algebra but graph traversal, not Tuple calculus but text search or map/reduce
- The implementation: rigid schemas vs. flexible schemas (schema-less)
- ACID compliance vs. BASE
- In that sense, NoSQL today is more commonly meant to be something like "non-relational"

NoSQL Distinguishing Characteristics

- Large data volumes
  - Google's "big data"
- Scalable replication and distribution
  - Potentially thousands of machines
  - Potentially distributed around the world
- Queries need to return answers quickly
- Mostly query, few updates
- Asynchronous Inserts & Updates
- Schema-less
- ACID transaction properties are not needed – BASE
- CAP Theorem
- Open source development (most)

Distributed Architecture

- Many NoSQL databases federate a bunch of servers
- Provides
  - Redundant storage
  - Geographic distribution
- Avoid single point of failure

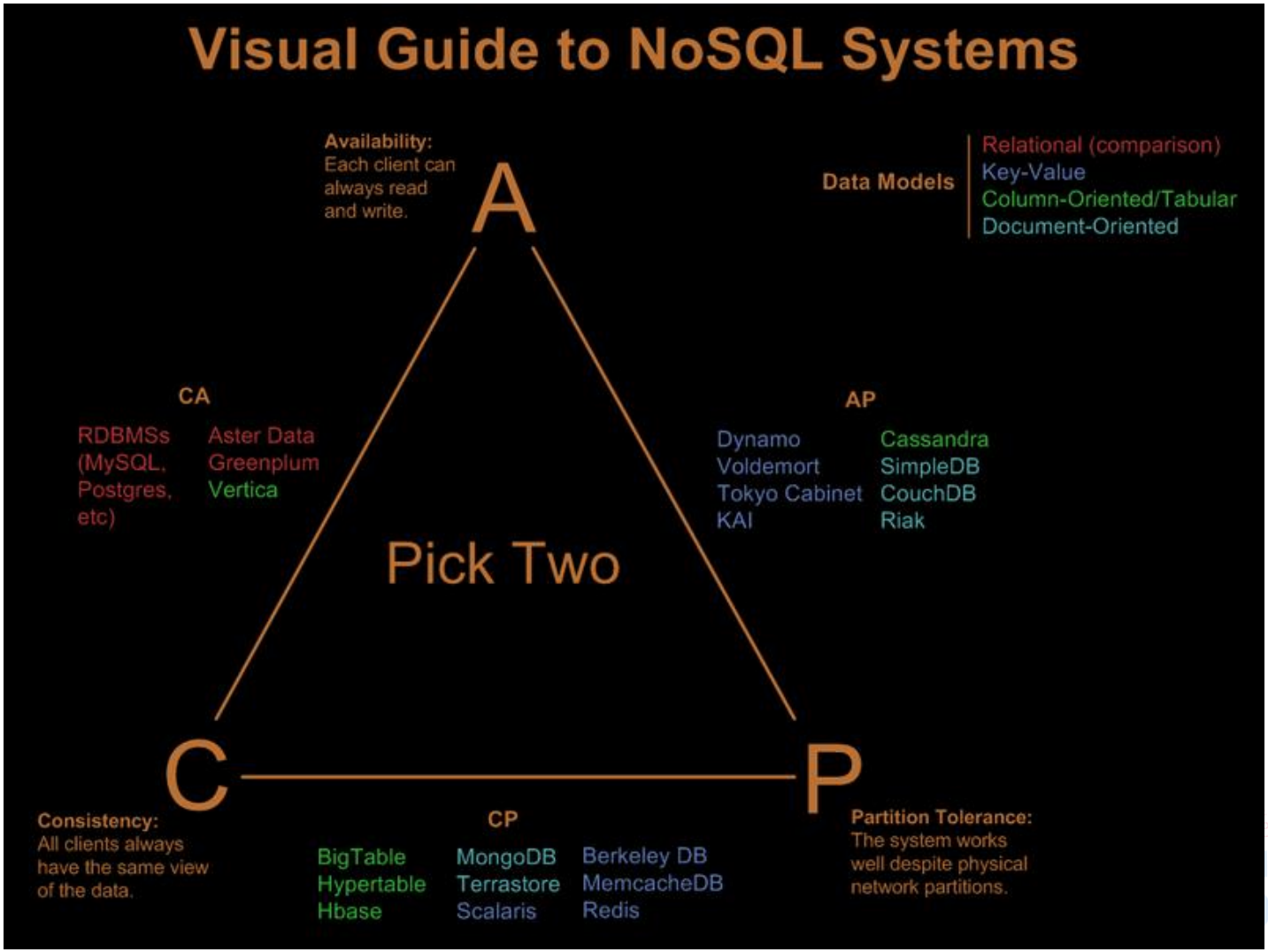


CAP Theorem

A distributed system can support only two of the following characteristics [Eric Brewer 2000]:

- Consistency:
  - All nodes within a cluster see the **same data** at the **same time**
  - System reliably follows established rules
- Availability
  - Node failures do not prevent survivors from **continuing to operate**
  - Every operation must terminate in an **intended response**
- Partition Tolerance
  - The system continues to operate despite arbitrary message loss – Wikipedia
  - Operations will complete, even if individual components are unavailable – Pritchett

Following CAP Theorem



BASE Transactions

- Acronym contrived to be the opposite of ACID
  - **B**asically **A**vailable,
  - **S**oft state,
  - **E**ventually Consistent
- Characteristics
  - Weak consistency – stale data OK
  - Availability first
  - Approximate answers OK
  - Aggressive (optimistic)
  - Simpler and faster

NoSQL db types

- Document Store
  - BaseX, Clusterpoint, Apache Couchbase, eXist, Jackrabbit, Lotus Notes and IBM Lotus Domino LotusScript, MarkLogic Server, MongoDB, OpenLink Virtuoso, OrientDB, RavenDB, SimpleDB, Terrastore
- Graph
  - AllegroGraph, DEX, FlockDB, InfiniteGraph, Neo4j, OpenLink Virtuoso, OrientDB, Pregel, Sones GraphDB, OWLIM
- Key Value
  - BigTable, CDB, Keyspace, LevelDB, membase, MemcacheDB, MongoDB, OpenLink Virtuoso, Tarantool, Tokyo Cabinet, TreapDB, Tuple space
  - Eventually-consistent - Apache Cassandra, Dynamo, Hibari, OpenLink Virtuoso, Project Voldemort, Riak
  - Hierarchical - GT.M, InterSystems Caché
  - Tabular – BigTable, Apache Hadoop, Apache Hbase, Hypertable, Mnesia, OpenLink Virtuoso
  - Object Database - db4o, Eloquera, GemStone/S, InterSystems Caché, JADE, NeoDatis ODB, ObjectDB, Objectivity/DB, ObjectStore, OpenLink Virtuoso, Versant Object Database, Wakanda, ZODB
  - Multivalue databases - Extensible Storage Engine (ESE/NT), jBASE, OpenQM, OpenInsight , Rocket U2, D3 Pick database, InterSystems Caché, InfinityDB
  - Tuple store- Apache River, OpenLink Virtuoso, Tarantool

Large diversity of NoSQL Database Types

- Key-value

- Graph database

- Document-oriented

- Column store

NoSQL Summary

- NoSQL databases reject:
  - Overhead of ACID transactions
  - "Complexity" of SQL
  - Burden of up-front schema design
  - Declarative query expression
  - Yesterday's technology
- Programmer responsible for
  - Step-by-step procedural language
  - Navigating access path

How Does NoSQL compare to SQL?

While there are numerous characteristics that differentiate SQL and NOSQL the two most significant are Scaling and Modeling.

- **Scaling** – Traditionally SQL does not lend itself to massively parallel processing, which lead to larger computers (scale up) vs. distribution to numerous commodity servers, virtual machines or cloud instances (scale out).
- **Modeling** – SQL databases are highly normalized and require pre-defined data models prior to inserting data into the system. In contrast NOSQL databases do not require (although they support) pre-defined data model(s).

Relational vs. NoSQL

Relational

- Relational
- Table based
- Predefined schema
- Vertical scalable
- Emphasized on ACID

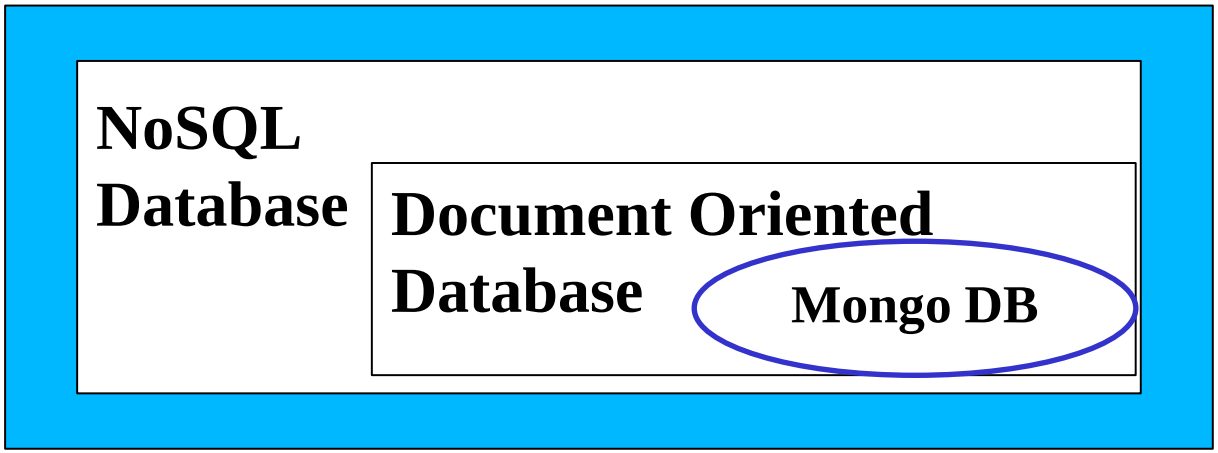
NoSQL

- No-Relational
- Document based, key-valued…
- Dynamic schema
- Horizontally scalable
- Follows BASIC transactions

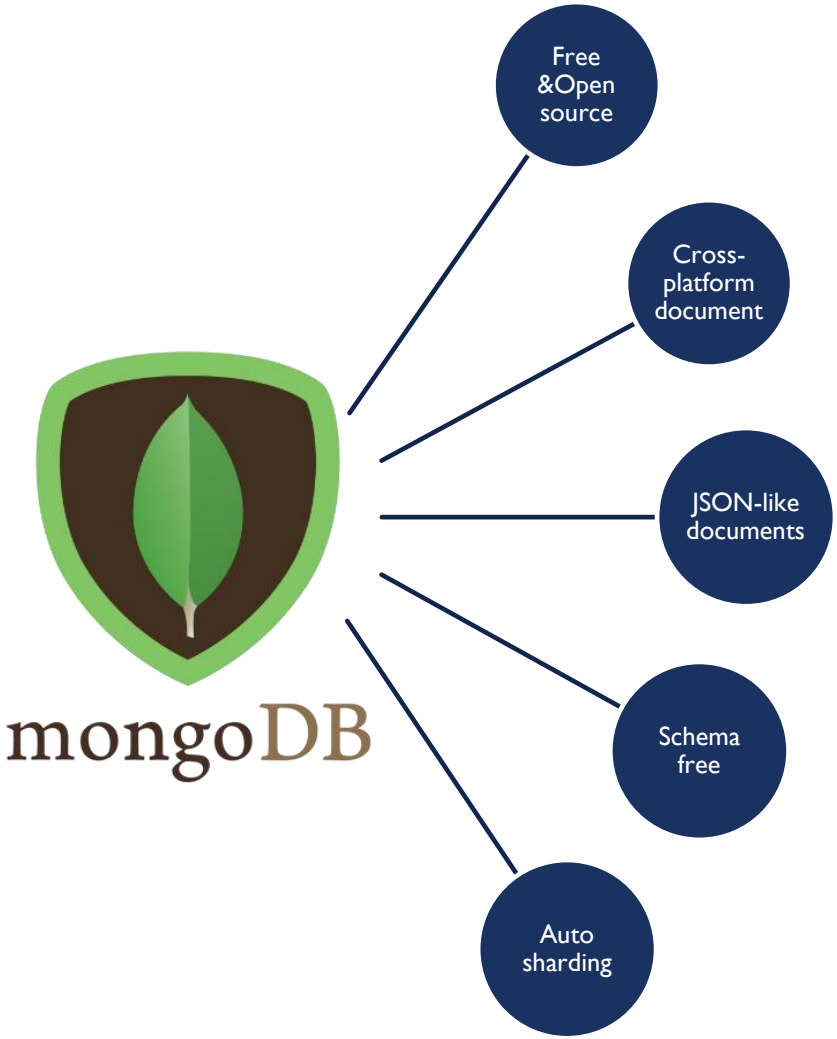
MONGODB

What is MongoDB?

- Definition: MongoDB is an open-source document oriented database that provides high performance, high availability, and automatic scaling.
- Instead of storing data in tables and rows as with a relational database, in MongoDB stores data in JSON-like documents with dynamic schemas(schema-free).



MongoDB main features



Data Storage Mechanism

- MongoDB Server
  - Database A
    - Collection 1
      - Document1
        - Field
          - Key
          - Value: string, int, array of values, documents, array of documents
      - Document1
        - Field
          - Key
          - Value
    - Collection 2
  - Database B
    - Collection 3

Data types

- Documents in MongoDB can be thought of as "JSON-like" in that they are conceptually similar to objects in JavaScript.
  - JSON is a simple representation of data
- Null: Use to represent both a null value and a nonexistent field:
  - {field : null}
- Boolean: which can be used for the values true and false:
  - {field : true}
- String: Any string of characters can be represented using the string type:
  - {field : "foodbar"}

Data types (cont')

- Number: The shell defaults to using 64-bit floating point numbers. Thus, these numbers look "normal" in the shell:
  - {field : 3.14} or: {field : 3}
- For integers, use the NumberInt or NumberLong classes, which represent 4-byte or 8-byte signed integers, respectively.
  - {field : NumberInt(3)}
  - {field : NumberLong(3)}
- Dates: are stored as milliseconds. The time zone is not stored:
  - {field : new Date()}

Data types (cont')

- Array: Sets or lists of values can be represented as arrays:
  - {field : ["a", "b", "c"]}
- Embedded document: Documents can contain entire documents embedded as values in a parent document:
  - {field : {field : "abc"}}
- object id: An object id is a 12-byte ID for documents.
  - {_id : ObjectId()}

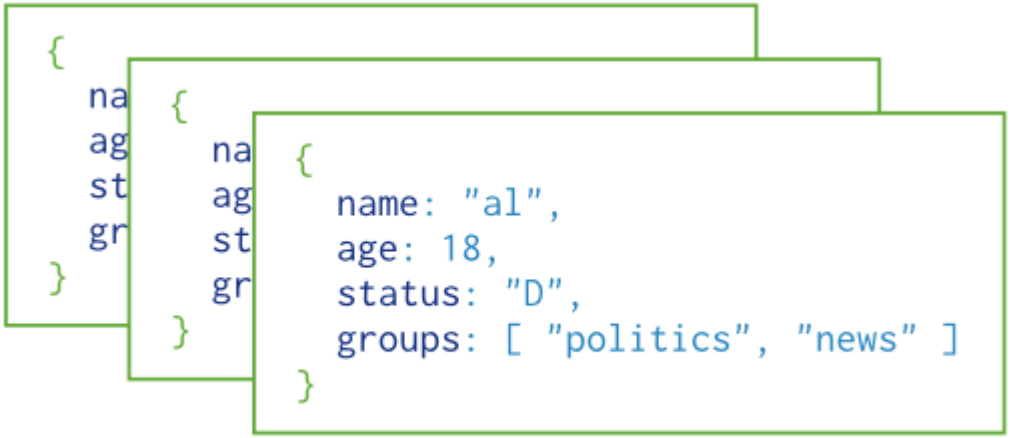
Concept mapping

RDBMS		MongoDB
Database	➡	Database
Table, View	➡	Collection
Row	➡	Document (JSON, BSON)
Column	➡	Field
Index	➡	Index
Join	➡	Embedded Document
Foreign Key	➡	Reference
Partition		Shard

```
> db.user.findOne({age:39})
{
        "_id" :
ObjectId("5114e0bd42…"),
        "first" : "John",
        "last" : "Doe",
        "age" : 39,
        "interests" : [
                "Reading",
                "Mountain Biking ]
        "favorites": {
                "color": "Blue",
                "sport": "Soccer"}
}
```

Documents and Collection

```
{
  name: "sue",                          <-- field: value
  age: 26,                              <-- field: value
  status: "A",                          <-- field: value
  groups: [ "news", "sports" ]          <-- field: value
}
```



CRUD

- Create
  - db.collection.insert(<document>)
  - db.collection.save(<document>)
  - db.collection.update(<query>, <update>, { upsert: true })
- Read
  - db.collection.find(<query>, <projection>)
  - db.collection.findOne(<query>, <projection>)
- Update
  - db.collection.update(<query>, <update>, <options>)
- Delete
  - db.collection.remove(<query>, <justOne>)

31

CRUD operations - CREATE

Insert a new user.

SQL

```
INSERT INTO users                          <-- table
            ( name, age, status )          <-- columns
VALUES      ( "sue", 26, "A" )             <-- values/row
```

MongoDB

```
db.users.insert (                          <-- collection
   {
     name: "sue",                          <-- field: value
     age: 26,                              <-- field: value
     status: "A"                           <-- field: value
   }                                       } document
)
```

CRUD operations – CREATE (cont'd)

Collection

Document

```
db.users.insert(
                  {
                     name: "sue",
                      age: 26,
                   status: "A",
                   groups: [ "news", "sports" ]
                  }
                )
```

Document

```
{
  name: "sue",
  age: 26,
  status: "A",
  groups: [ "news", "sports" ]
}
```

Collection

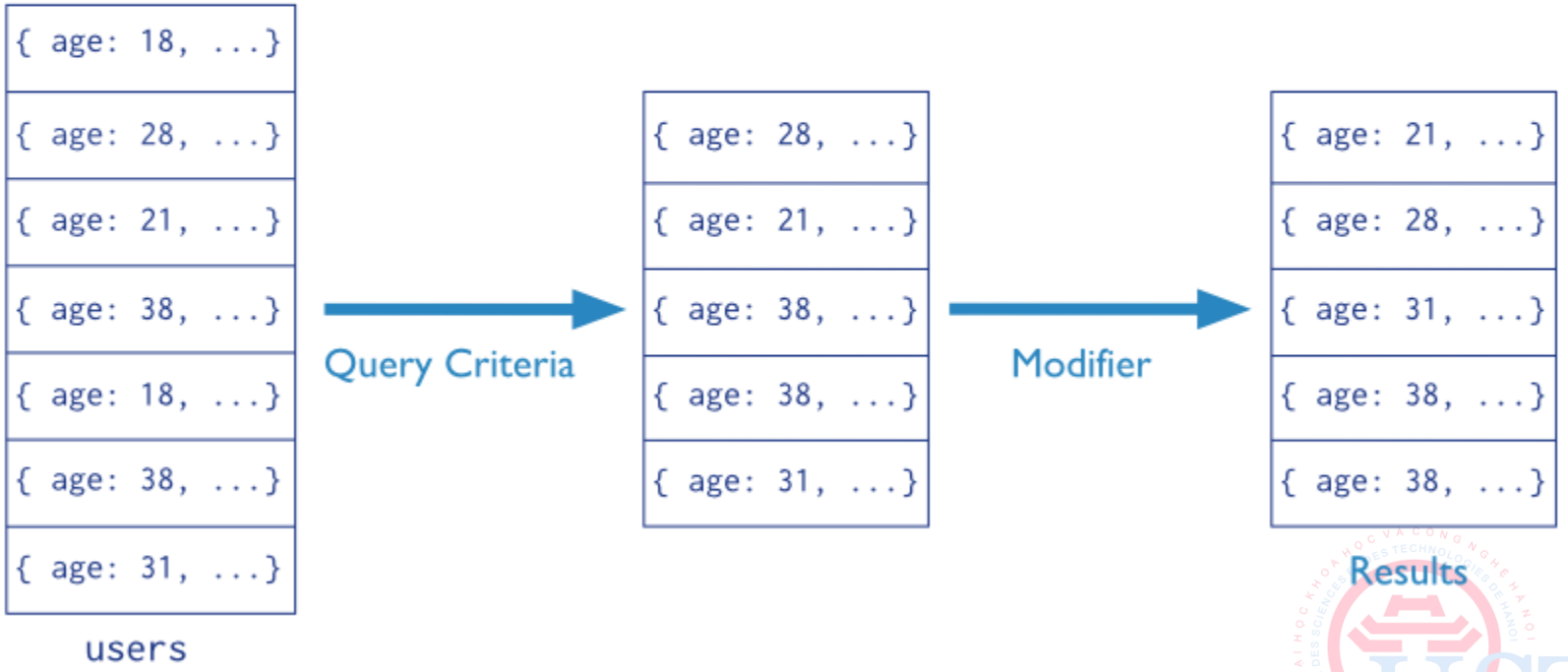
```
{ name: "al", age: 18, ... }
{ name: "lee", age: 28, ... }
{ name: "jan", age: 21, ... }
{ name: "kai", age: 38, ... }
{ name: "sam", age: 18, ... }
{ name: "mel", age: 38, ... }
{ name: "ryan", age: 31, ... }
{ name: "sue", age: 26, ... }
```

CRUD operations - READ

Find the users of age greater than 18 and sort by age.

Collection | Query Criteria | Modifier

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



READ - Aggregation

- SQL
  - SELECT COUNT(*)
  - FROM Restaurants
  - GROUP BY borough
- MongoDB
  - db.restaurants.aggregate(
  - [
  - { $group: { "_id": "$borough", "count": { $sum: 1 } } }
  -]);

```
{ "_id" : "Missing", "count" : 51 }
{ "_id" : "Staten Island", "count" : 969 }
{ "_id" : "Manhattan", "count" : 10259 }
{ "_id" : "Bronx", "count" : 2338 }
{ "_id" : "Queens", "count" : 5656 }
{ "_id" : "Brooklyn", "count" : 6085 }
```

CRUD operations - UPDATE

Update the users of age greater than 18 by setting the status field to A.

SQL

```
UPDATE users          <-- table
SET    status = 'A'   <-- update action
WHERE  age > 18       <-- update criteria
```

MongoDB

```
db.users.update(                    <-- collection
   { age: { $gt: 18 } },            <-- update criteria
   { $set: { status: "A" } },       <-- update action
   { multi: true }                  <-- update option
)
```

CRUD operations - DELETE

Delete the users with status equal to D.

SQL

```
DELETE FROM users      <-- table
WHERE  status = 'D'    <-- delete criteria
```

MongoDB

```
db.users.remove(       <-- collection
   { status: "D" }     <-- remove criteria
)
```

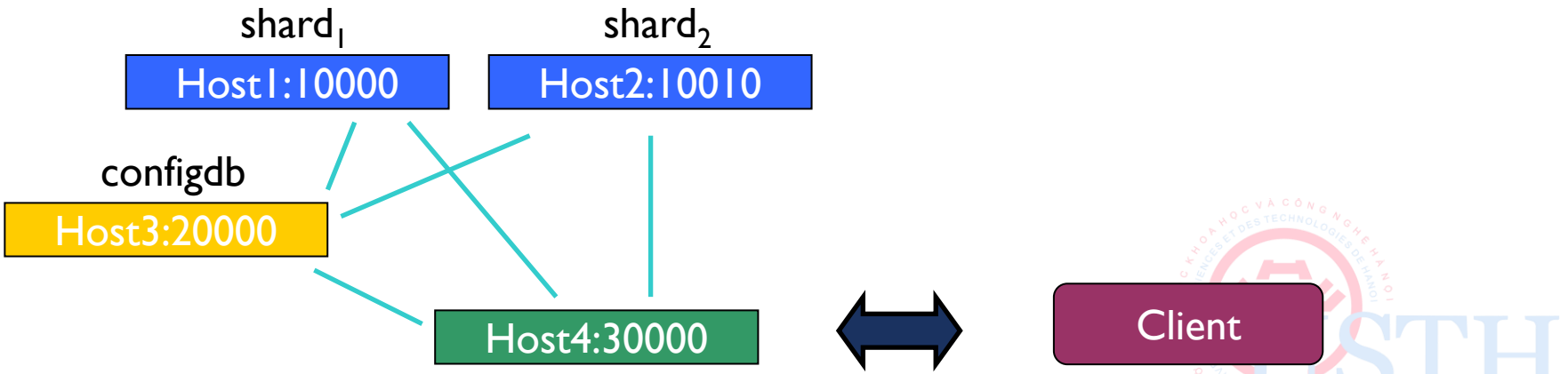
MONGODB SHARDING

Ref: *Kristina Chodorow,* Scaling MongDB Sharding, Cluster Setup, and Administration, 2011

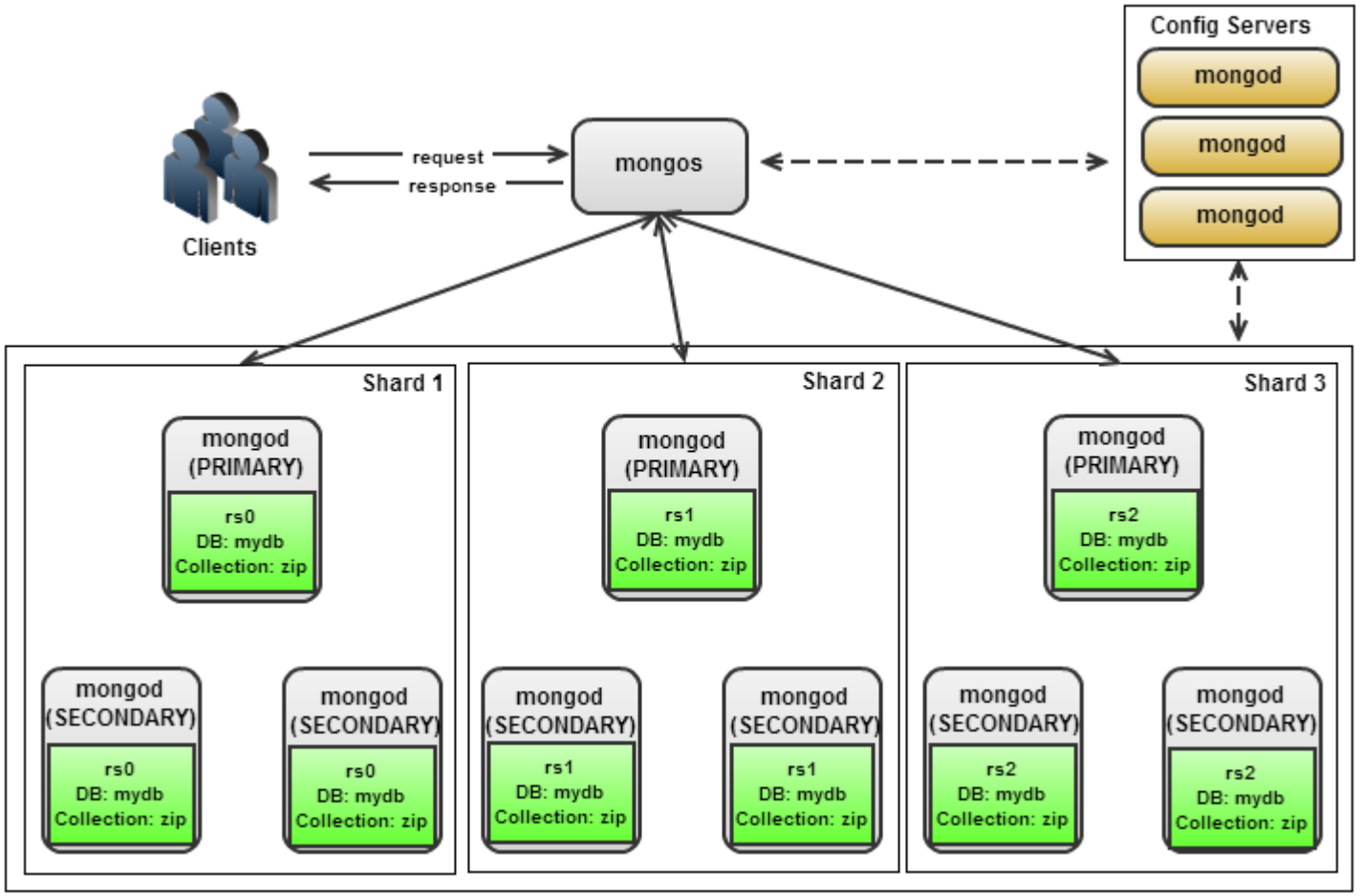
Sharding

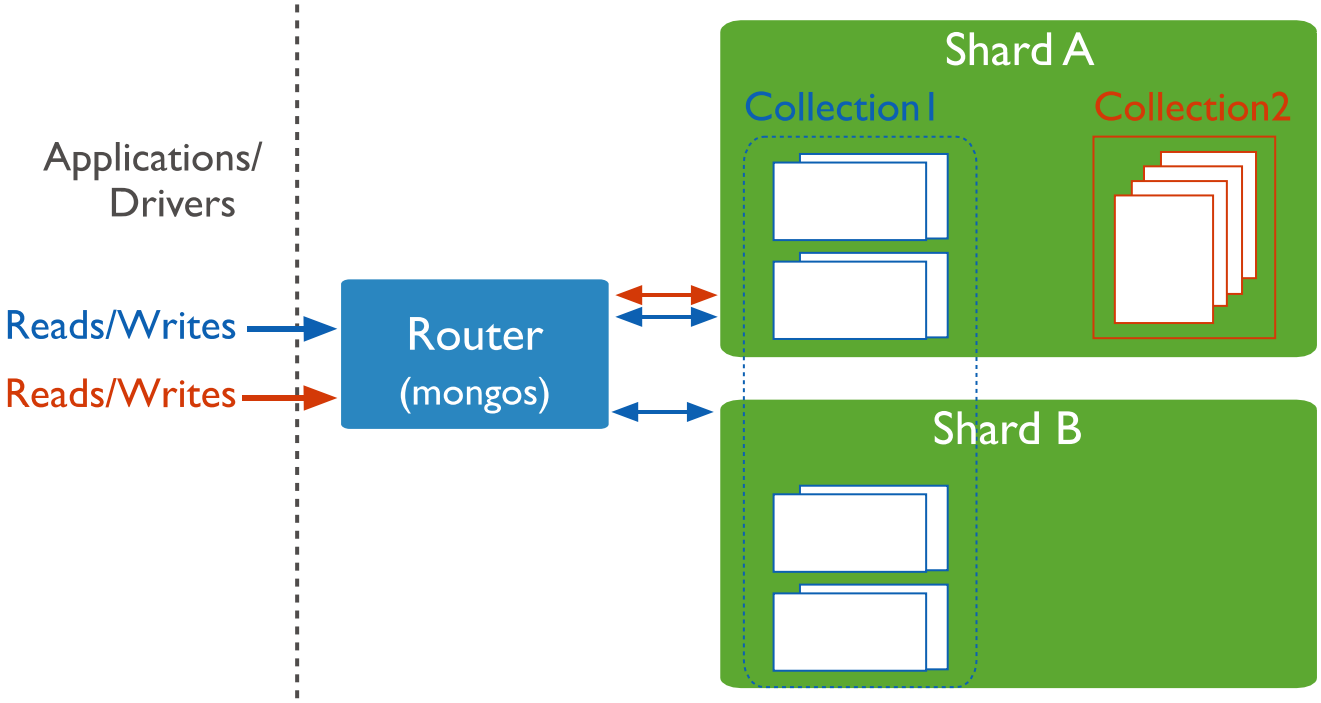
- Partition your data
- Scale write throughput
- Increase capacity
- Auto-balancing

id	company	customer	article	currency	price
4250250	020	073000	5994537812	00	142,50
4250251	020	073000	5994537852	00	141,12
4250252	020	073000	5994537854	00	105,99
4250253	020	073000	5994537856	00	109,52
4250254	020	073000	5994537862	00	131,49
4250255	020	073000	5994567308	00	29,86
4250256	020	073000	5994567422	00	57,13
4250257	020	073000	5994567428	00	68,59
4250258	020	073000	5994605089	00	51,09
4250259	020	073000	5994607975	00	93,93
4250260	020	073000	5994701005	00	74,22



Common architecture

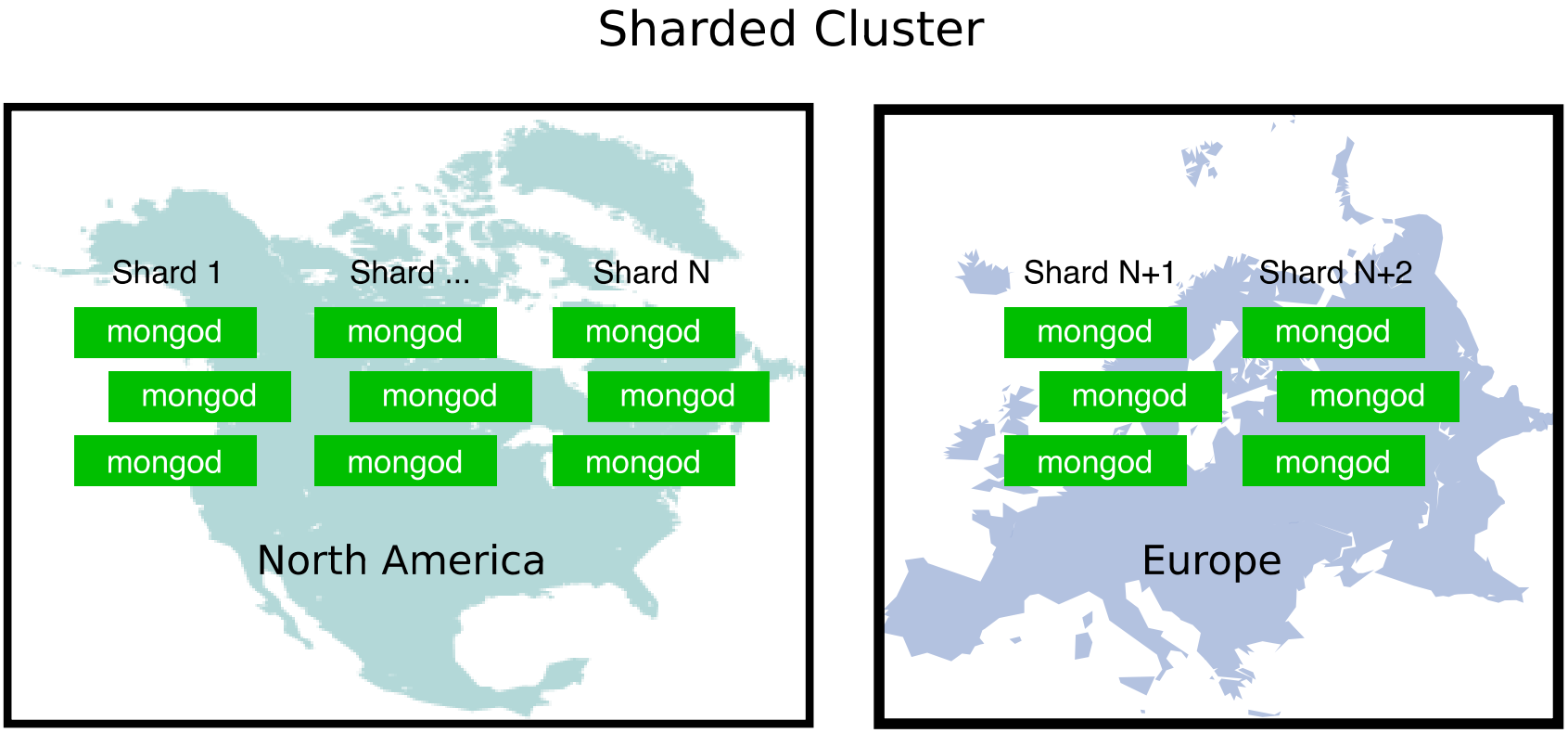




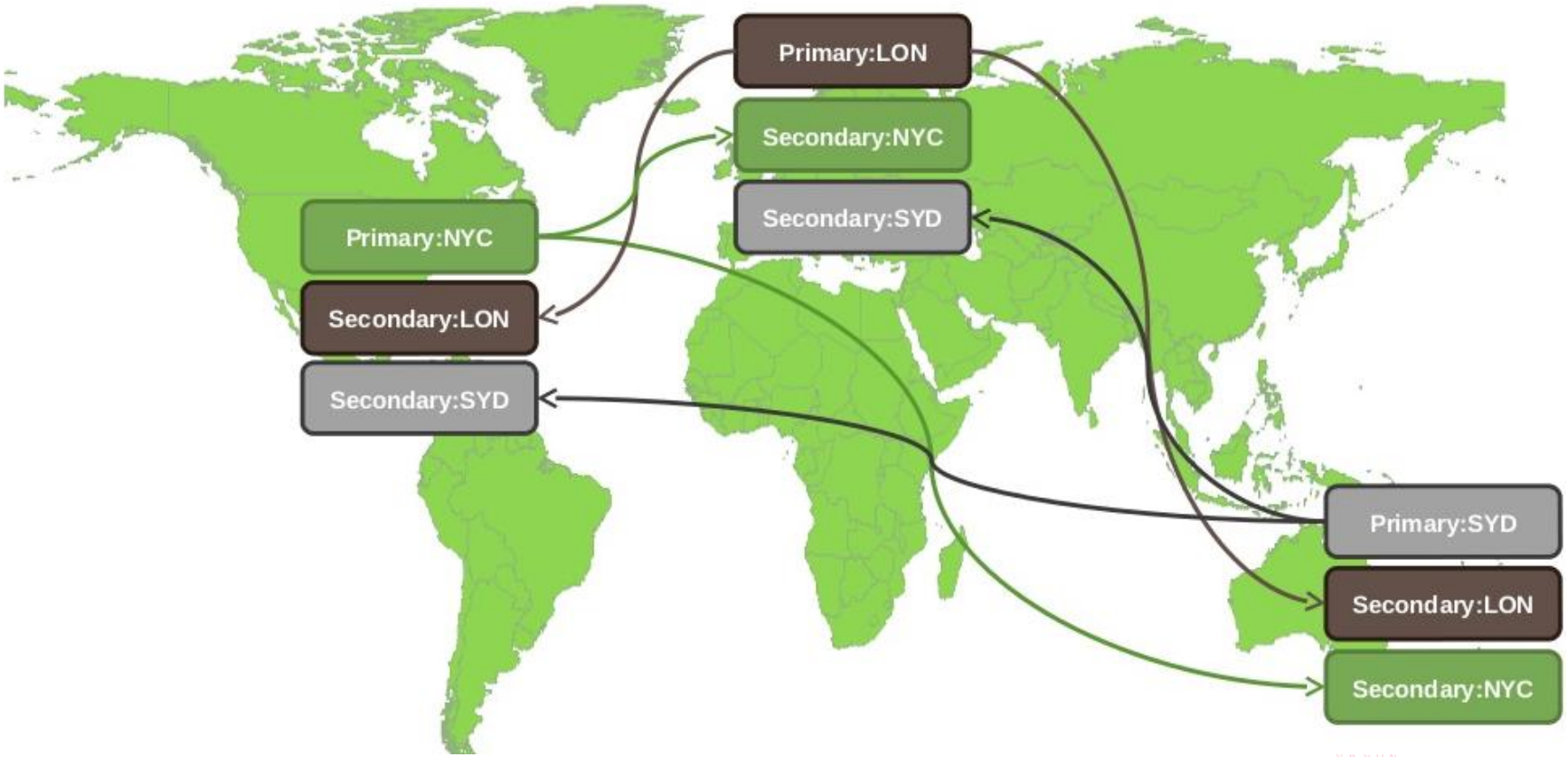
Terminologies

- Replication
  - Replica set: a group of mongod maintaining same data
  - Primary: Endpoint for writes
  - Secondary: Mirror of Primary
  - Election: process to select Primary among mongod
  - Arbiter: mongod responsible for election
- Chunk: data partition of a collection split by shard key
- Shard: a Replica Set holding some chunks

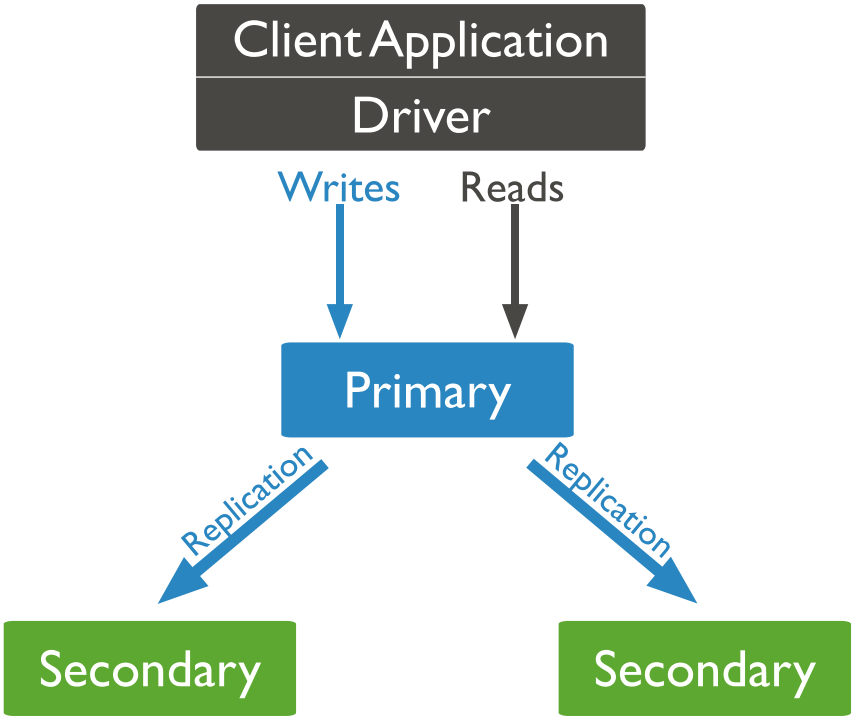
Data segmentation by Location



Read global, write local

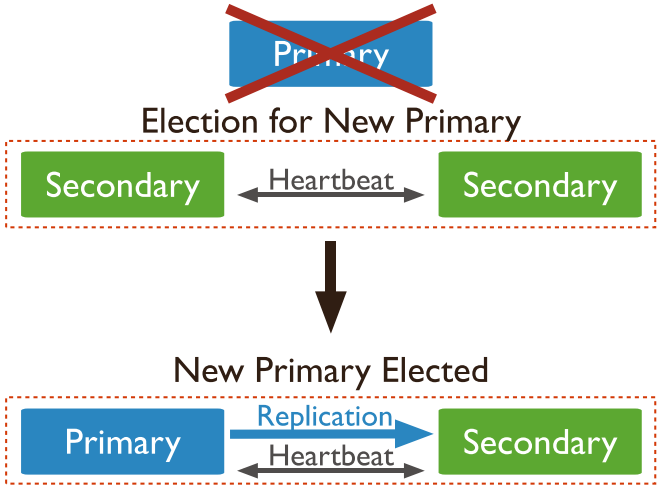


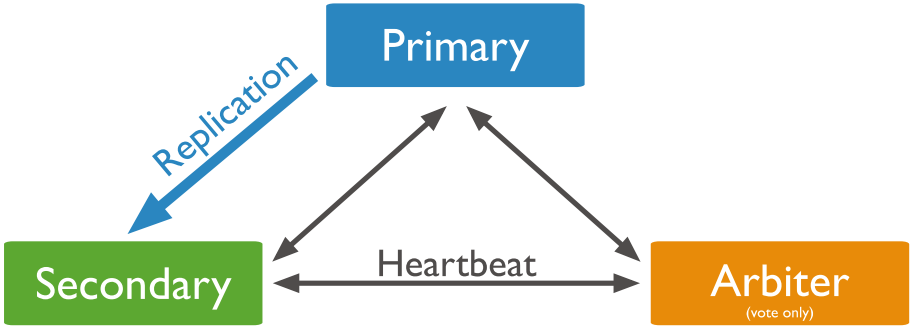
Replication



- Replica set: a group of mongod instances that maintain the same data set
- Goals:
  - High Availability (HA) through data duplication
  - Recovery
  - Workload isolation: distributed access with low latency

Failover





Read Operations to Replica Sets