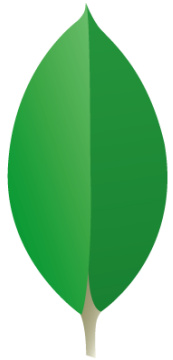
MongoDB



Team# 7

Shubham Vadhera | Sagar Dafle | Jagmohan Singh |Pooja Yelure

MongoDB Design Philosophy

- Combining the critical capabilities of relational databases with the innovations of NoSQL technologies

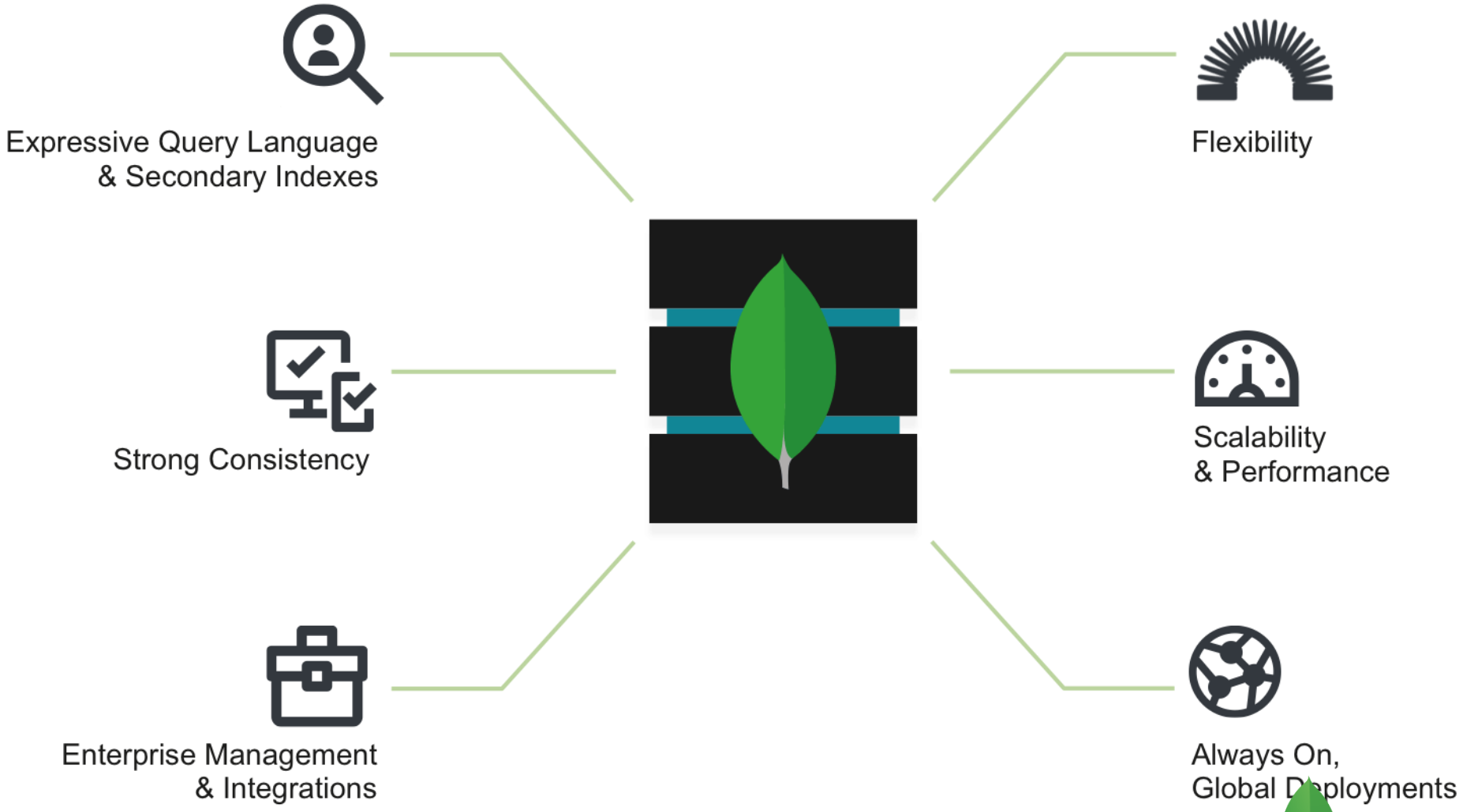
RELATIONAL

NoSQL

mongoDB®

MongoDB Architecture Guide
https://www.google.com/imghp

Mongo DB !



https://www.mongodb.com/mongodb-architecture

mongoDB®

Traditional DB – Relational DB

- **Expressive query language -** Users should be able to access and manipulate their data in sophisticated ways
- **Strong consistency -** Applications should be able to immediately read what has been written to the database
- **Enterprise Management and Integrations -** A database should allow administrators to secure, monitor, automate, and integrate with their existing technology infrastructure

Modern DB – NoSQL DB

- **Flexible Data Model -** Easy to store and combine data of any structure
- **Scalability and Performance -** Enable almost unlimited growth with higher throughput and lower latency than relational databases.
- **Always-On Global Deployments –** Available systems across many nodes

mongoDB®

MongoDB Data Model

- Data As Documents

id	user_name	email	age	city
1	Mark Hanks	mark@abc.com	25	Los Angeles
2	Richard Peter	richard@abc.com	31	Dallas

```
{
   "_id": ObjectId("5146bb52d8524270060001f3"),
   "age": 25,
   "city": "Los Angeles",
   "email": "mark@abc.com",
   "user_name": "Mark Hanks"
}
{
   "_id": ObjectId("5146bb52d8524270060001f2"),
   "age": 31,
   "city": "Dallas",
   "email": "richard@abc.com",
   "user_name": "Richard Peter"
}
```

  - MongoDB stores data as documents in a binary representation called BSON (Binary JSON)
  - BSON extends the popular JSON (JavaScript Object Notation) representation to include additional types such as int, long, date, and floating point
  - Documents that tend to share a similar structure are organized as collections

MongoDB Architecture Guide
https://cdn.tutsplus.com/net/uploads/2013/11/Figure-1-Mapping-Table-to-Collection-1.png

mongoDB®

MongoDB Data Model

- MongoDB documents are composed of field-and-value pairs and have the following structure:

```
{
   field1: value1,
   field2: value2,
   field3: value3,
   ...
   fieldN: valueN
}
```

```
var mydoc = {
               _id: ObjectId("5099803df3f4948bd2f98391"),
               name: { first: "Alan", last: "Turing" },
               birth: new Date('Jun 23, 1912'),
               death: new Date('Jun 07, 1954'),
               contribs: [ "Turing machine", "Turing test", "Turingery" ],
               views : NumberLong(1250000)
             }
```

https://docs.mongodb.org/manual/core/databases-and-collections/#collections

MongoDB Data Model

- Dynamic Schema
  - Fields can vary from document to document
  - There is no need to declare the structure of documents to the system –documents are self describing
- Document Validation
  - Users can enforce checks on document structure, data types, data ranges and the presence of mandatory fields

Key Market Features (Pros) of MongoDB

- **High availability** (by replicating the data)
- **Scalability** (from a standalone server to distributed architectures of huge clusters). This allows us to shard our database transparently across all our shards. This increases the performance of our data processing.
- **Aggregation**: Batch data processing and aggregate calculations using native MongoDB operations.
- **Load Balancing**: Automatic data movement across *different shards* for load balancing. The balancer decides when to migrate the data and the destination Shard, so they are evenly distributed among all servers in the cluster.

http://www.mongodbspain.com/en/2014/08/17/mongodb-characteristics-future/

Why MONGO over other NoSQL DBs

- **MongoDB Management Service** (MMS) is a powerful **web tool** that allows us tracking our databases and our machines and also backing up our data.
- MMS tracks the database and hardware metrics for managing mongodb deployment.
- **Custom alerts**: Discover issues before your MongoDB instance will be affected.
- **Task Automation:** Simple launch and configuration of standalone MongoDB instances, replica sets or sharded clusters.

http://www.mongodbspain.com/en/2014/08/17/mongodb-characteristics-future/

CONS of Mongo DB

- **Less Flexibility** with querying (e.g. no JOINs)
- **Memory Usage**
  MongoDB has the natural tendency to use up more memory because it has to store the key names within each document.
- **Concurrency Issues**
  When you perform a write operation in MongoDB, it creates a lock on the entire database, not just the affected entries, and not just for a particular connection. This lock blocks not only other write operations, but also read operations.

http://halls-of-valhalla.org/beta/articles/the-pros-and-cons-of-mongodb,45/

Optimal use cases of Mongo

- Usually used when we need a **horizontally scalable performance** for high loads.
- Real-time analytics and high-speed logging, caching and high scalability.
- RDBMS replacement for **web applications**.
- With 1.5 million new classified ads posted every day, **Craigslist** must archive billions of records in many different formats, and must be able to query and report on these archives at runtime.

https://www.mongodb.com/customers/craigslist

CRUD Operations

- Read Operations: Queries are the core operations that return data in MongoDB.
- Cursors: Queries return iterable objects, called cursors, that hold the full result set.
- Write operations: Insert, update, or remove documents in MongoDB. Introduces data create and modify operations, their behavior, and performances.
- Atomicity and Transactions: Describes write operation atomicity in MongoDb.

CRUD Operations

- Commands Used to Create Document:
  - db.collection.insert()
  - db.collection.insertOne();
  - db.collection.insert();
- Example to Insert Query: db.bank_data.insertOne({"first_name" : "John", "last_name" : "Cena", "accounts" : [{ "account_type" : "Investment", "account_balance" : 6123524974.110823463, "currency" : "USD" }, { "account_type" : "Savings", "account_balance" : 132933272.569229168, "currency" : "EURO" }]});
- Command Used to Read Documents:
  - db.collection.find();
  - Db.collection.findOne();
- Example for reading : db.bank_data.findOne({"last_name":"SMITH"});

- Projections: In Mongo DB projection meaning is selecting only necessary data rather than selecting whole of the data of a document
  - $
  - $elemMatch
  - $limit

- Example: db.bank_data.findOne({"last_name":"Cena"},{"first_name":1,"_id":0});

Comparison Operators

`$eq`	Matches values that are equal to a specified value.
`$gt`	Matches values that are greater than a specified value.
`$gte`	Matches values that are greater than or equal to a specified value.
`$lt`	Matches values that are less than a specified value.
`$lte`	Matches values that are less than or equal to a specified value.
`$ne`	Matches all values that are not equal to a specified value.
`$in`	Matches any of the values specified in an array.
`$nin`	Matches none of the values specified in an array.

Example: $eq: db.bank_data.find({ "accounts.account_balance": {$eq:132933272.569229168}}).pretty()

$gte,$lte: db.bank_data.find({ "accounts.account_balance": {$gte:8554996,$lte:9000000}}).pretty()[4];

Logical Operators

Name	Description
$or	Joins query clauses with a logical OR returns all documents that match the conditions of either clause.
$and	Joins query clauses with a logical AND returns all documents that match the conditions of both clauses.
$not	Inverts the effect of a query expression and returns documents that do *not* match the query expression.
$nor	Joins query clauses with a logical NOR returns all documents that fail to match both clauses.

- Examples:
  - $and db.bank_data.find({$and:[{"accounts.account_type":"Checking"}, { "accounts.account_balance": {$gte:8554996,$lte:9000000}}]}).pretty()[4];
  - $or db.bank_data.find({$and:[{"accounts.account_type":"Investment"},{"accounts.account_type":"Savings"}]}).pretty()[4];

Array and Regex Operator

Name	Description
$all	Matches arrays that contain all elements specified in the query.
$elemMatch	Selects documents if element in the array field matches all the specified $elemMatchconditions.
$size	Selects documents if the array field is a specified size.

Example $elemMatch: db.bank_data.find({last_name: "SMITH", "accounts.account_type": "Savings" }, { first_name: 1, last_name: 1, accounts: { $elemMatch : { 'account_type' : 'Savings' } } }).pretty();

$regex: Provides regular expression capabilities for pattern matching *strings* in queries. MongoDB uses Perl compatible regular expressions (i.e. “PCRE”) version 8.38 with UTF-8 support.

Example: db.bank_data.find({"first_name":{$regex:/^RI.*/}},{"first_name":1,"last_name":1,"_id":0}).pretty();

mongoDB®

Update and Remove Queries

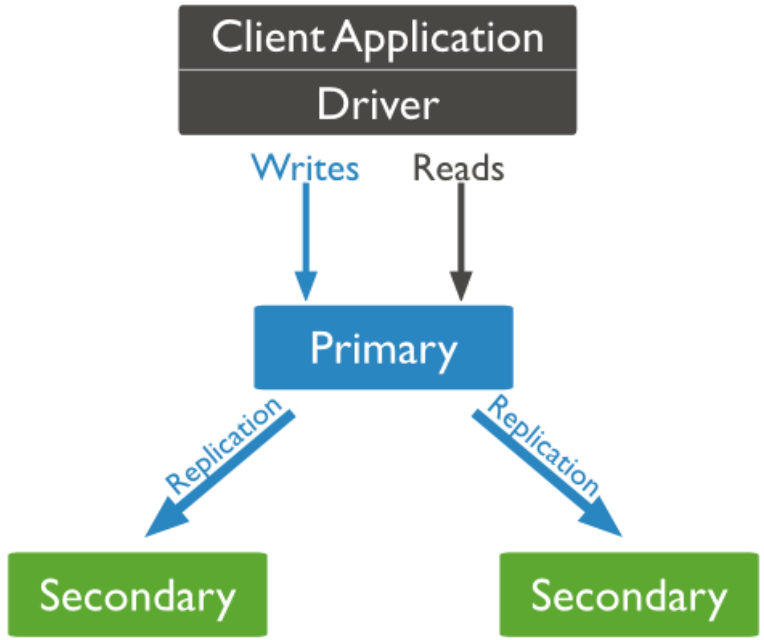
- MongoDB's **update()** and **save()** methods are used to update document into a collection. The update() method update values in the existing document while the save() method replaces the existing document with the document passed in save() method.

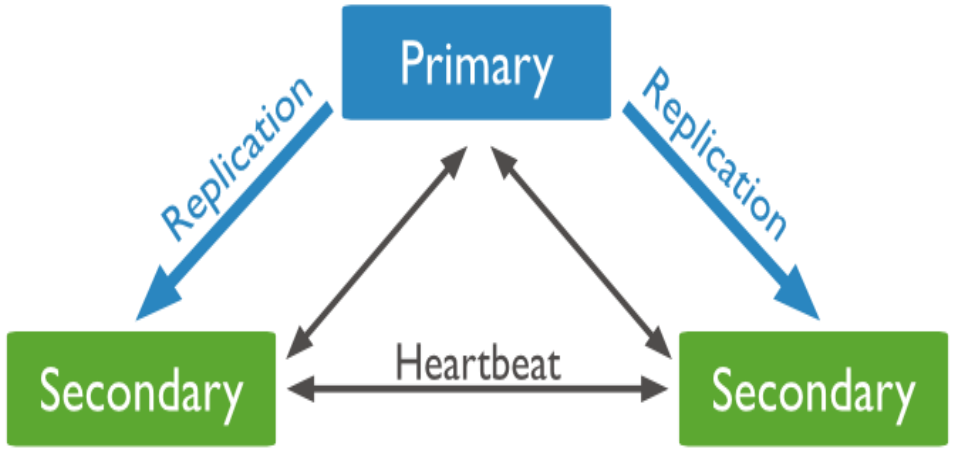
`$rename`	Renames a field.
`$setOnInsert`	Sets the value of a field if an update results in an insert of a document. Has no effect on update operations that modify existing documents.
`$set`	Sets the value of a field in a document.
`$unset`	Removes the specified field from a document.
`$min`	Only updates the field if the specified value is less than the existing field value.
`$max`	Only updates the field if the specified value is greater than the existing field value.

- MongoDB's **remove()** method is used to remove document from the collection. remove() method accepts two parameters. One is deletion criteria and second is justOne flag
- Example: db.collection.remove({”last_name”:”SMITH”});

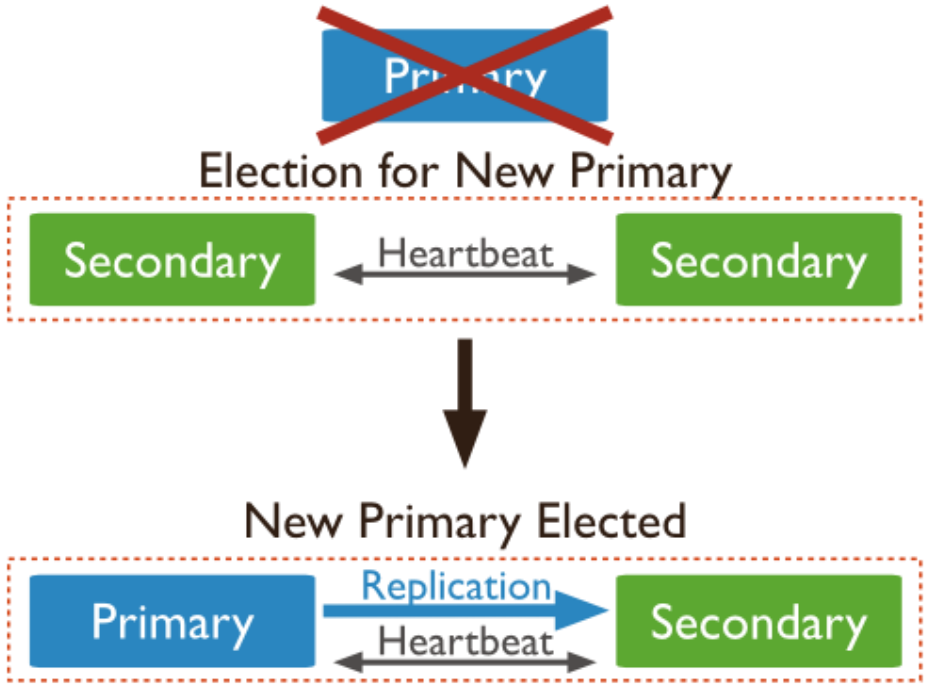
Replication Architecture

- Replication set – multiple mongod instances
- Upto 50 secondary allowed
- Primary authoritative node
  - Read
  - Writes
- Secondary nodes
  - Reads
- At most 12 replica sets are allowed.



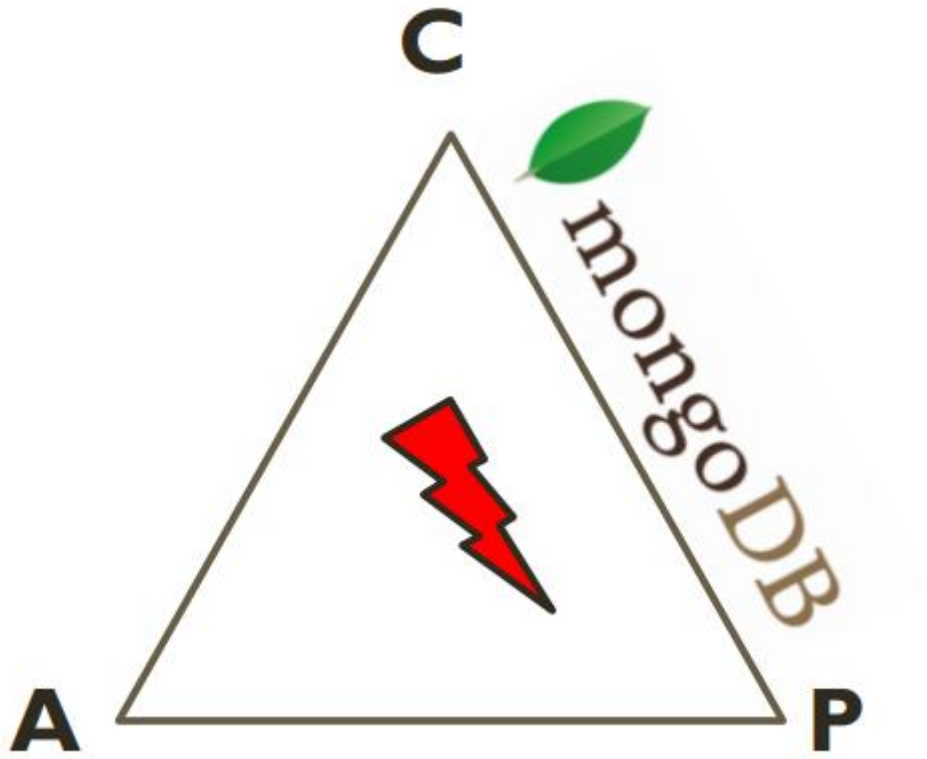


Automatic Failover



https://docs.mongodb.org/manual/core/replica-set-members/

Partition Tolerance



http://www.ccs.neu.edu/home/kathleen/classes/cs3200/20-NoSQLMongoDB.pdf

mongoDB®

How does MongoDB solve the CAP?

- CP System
- Write Concern?
  - Lets you choose when was a write successful. Acknowledgements like
    - error ignored
    - How many nodes must have write acknowledged.
- Read Preferences:
  - Allows you to choose where to read from.
    - Master / Slave

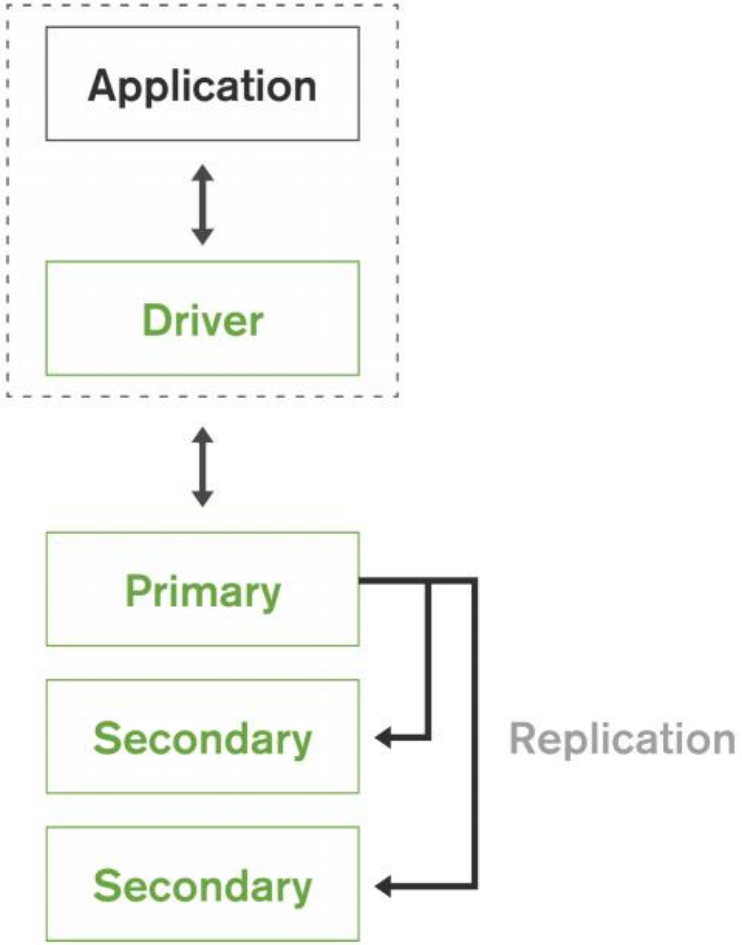


Figure 8: Self-Healing MongoDB Replica Sets for High Availability

http://s3.amazonaws.com/info-mongodb-com/MongoDB_Architecture_Guide.pdf

mongoDB®

- For Write
  - a write quorum can be implemented using write concern.
  - if number of available nodes < specified in write concern, write operation will fail.

- All read operations sent to the primary are consistent to last write operations.

- Reads to a primary have **strict consistency**
  - Reads reflect the latest changes to the data
- Reads to a secondary have **eventual consistency**
  - Updates propagate gradually
- If clients permit reads from secondary sets – then client may read a previous state of the database.

http://www.ccs.neu.edu/home/kathleen/classes/cs3200/20-NoSQLMongoDB.pdf

References

- http://jandiandme.blogspot.de/2013/06/mongodb-and-cap-theorem.html
- http://stackoverflow.com/questions/7339374/nosql-what-does-it-mean-for-mongodb-or-bigtable-to-not-always-be-available
- http://www.ccs.neu.edu/home/kathleen/classes/cs3200/20-NoSQLMongoDB.pdf
- https://docs.mongodb.org/manual/core/replica-set-members/
- https://github.com/sedouard/mongodb-mva