Group13: Siddhant Deshmukh, Sudeep Rege,
Sharmila Prakash, Dhanusha Varik

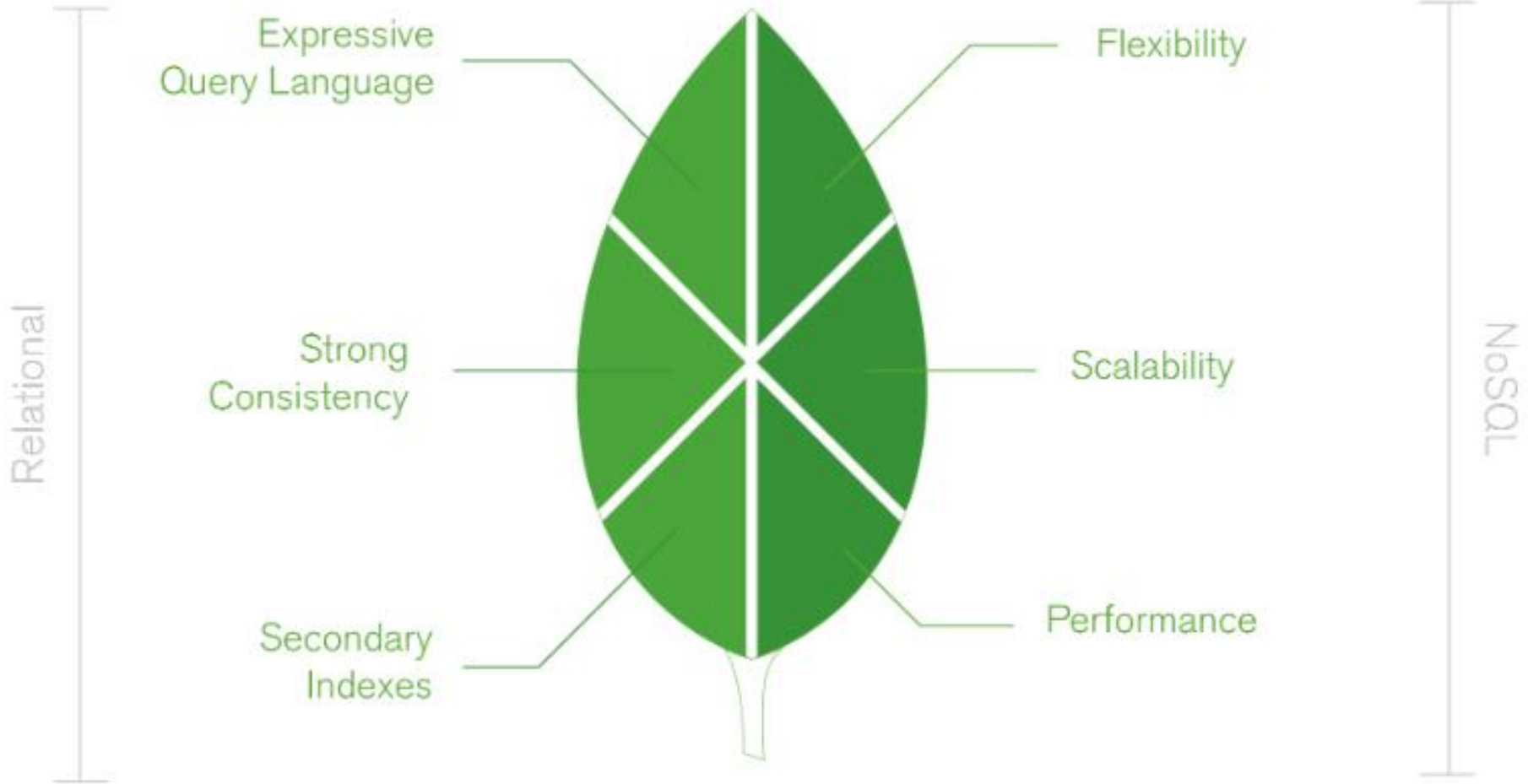
mongoDB (humongous)

Introduction

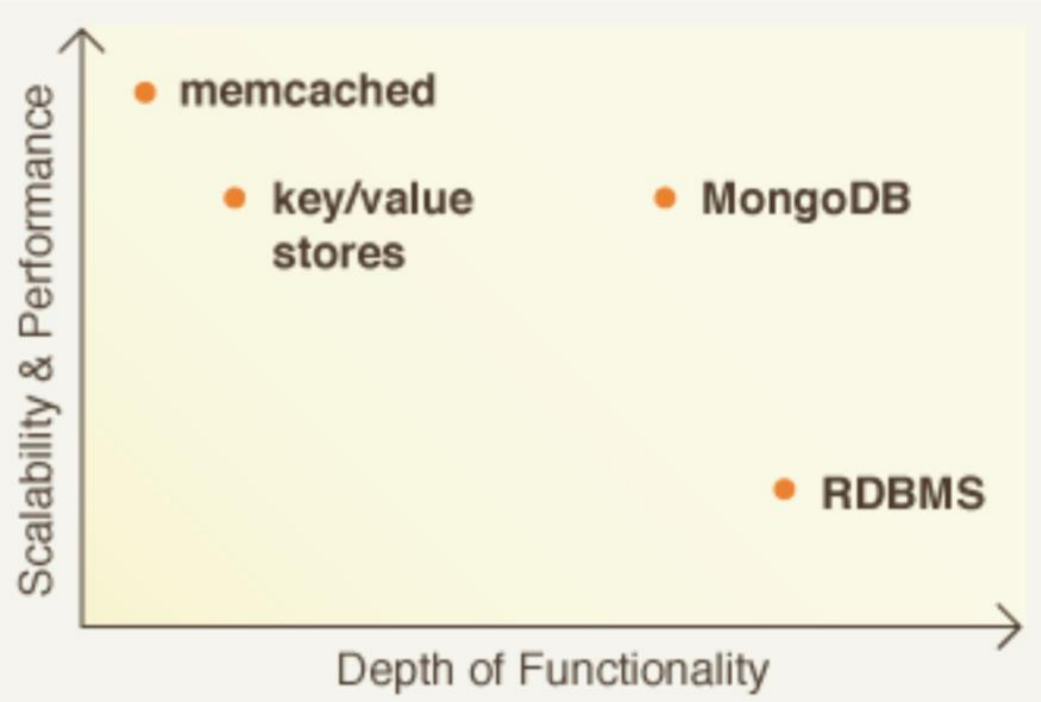
- What is MongoDB?
- Why MongoDB?
- MongoDB Terminology
- Why Not MongoDB?

What is MongoDB?

DOCUMENT STORE

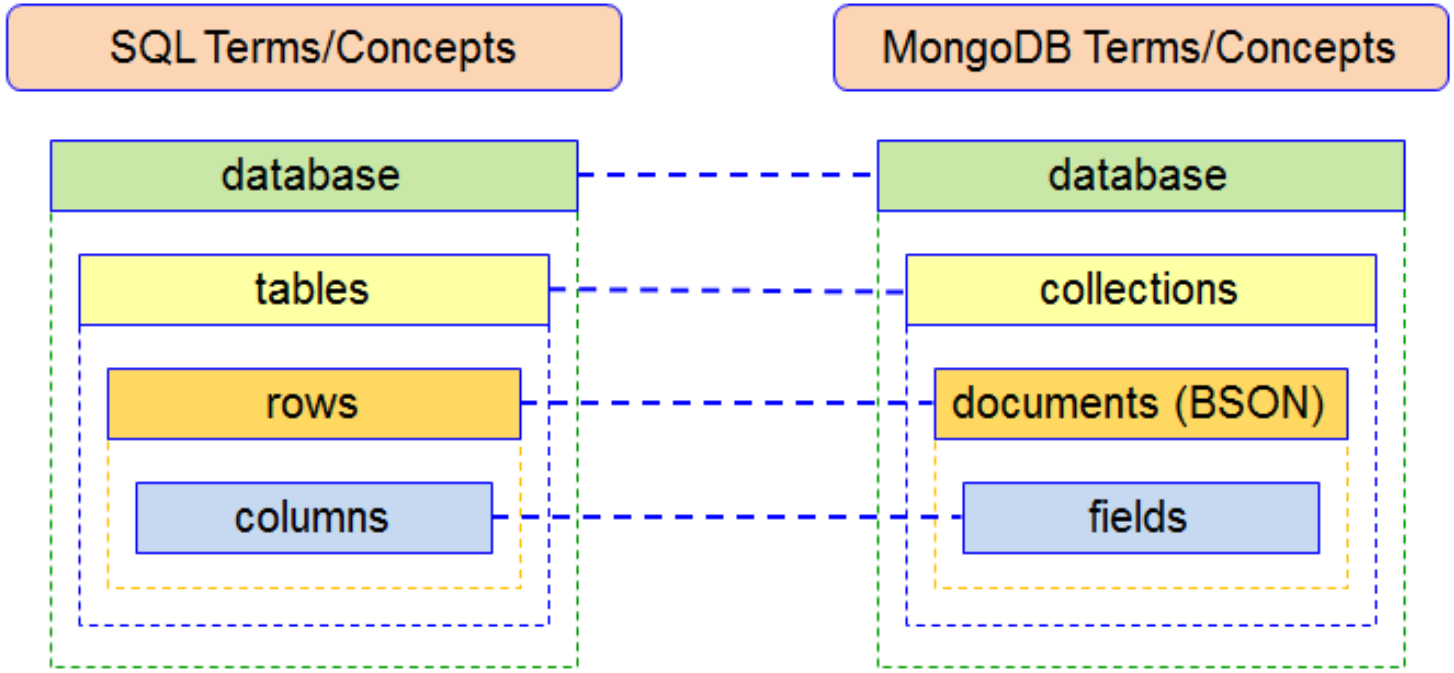


Why MongoDB?


memcached
key/value stores
MongoDB
RDBMS
Scalability & Performance
Depth of Functionality

Popularity of MongoDB

MongoDB Terminology



MongoDB is not all good...

It does not support

- Joins
- Transactions across multiple collections
- Data size in mongoDB is higher document store field names

Atomic transactions supported at single document level

REASON???

Document Stores

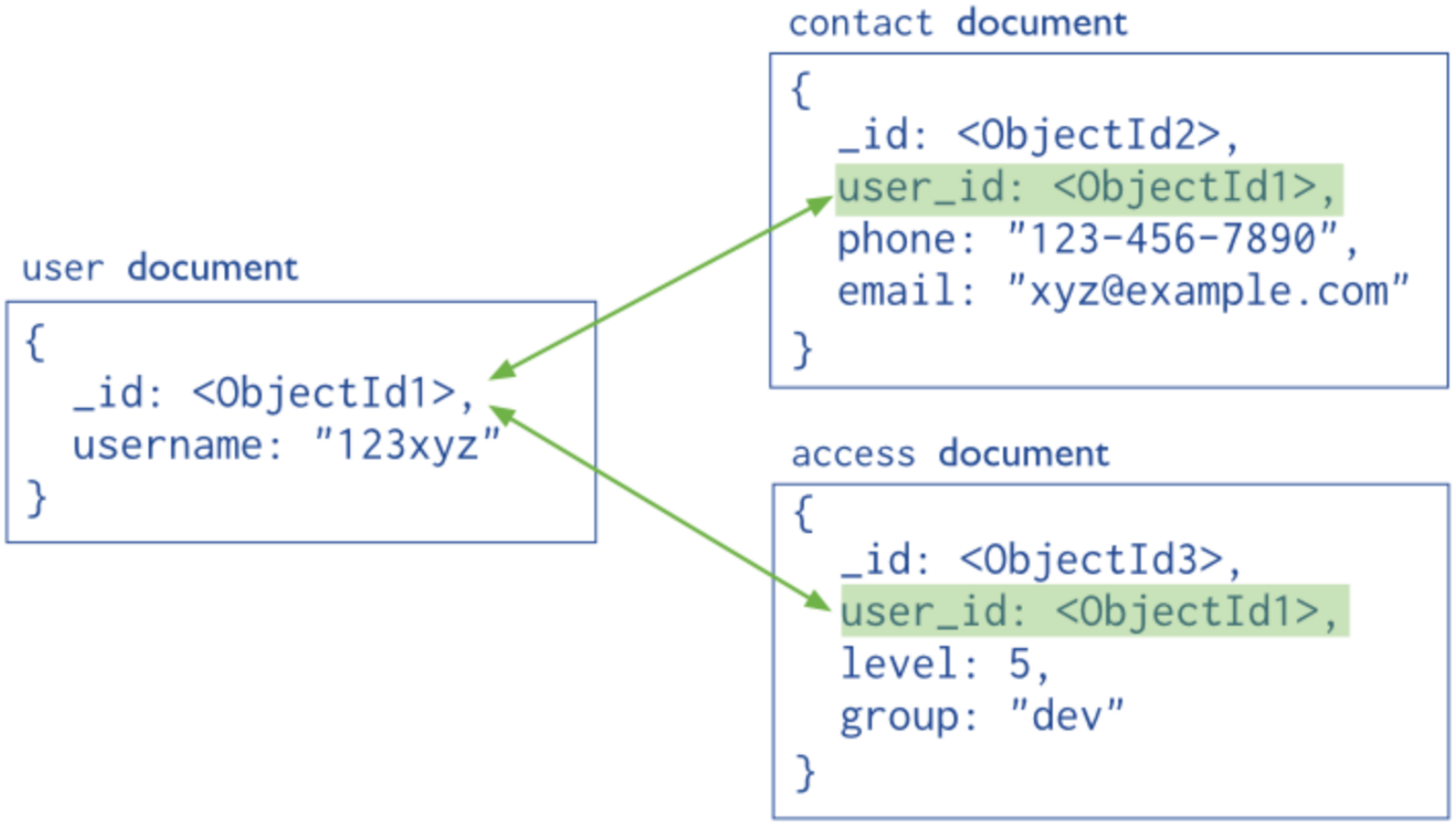
- Data Model
- Storage Model
- Collections
- Capped Collections

Data Model Design

- Embedded data Models
- Normalized data Models

Embedded Data Model

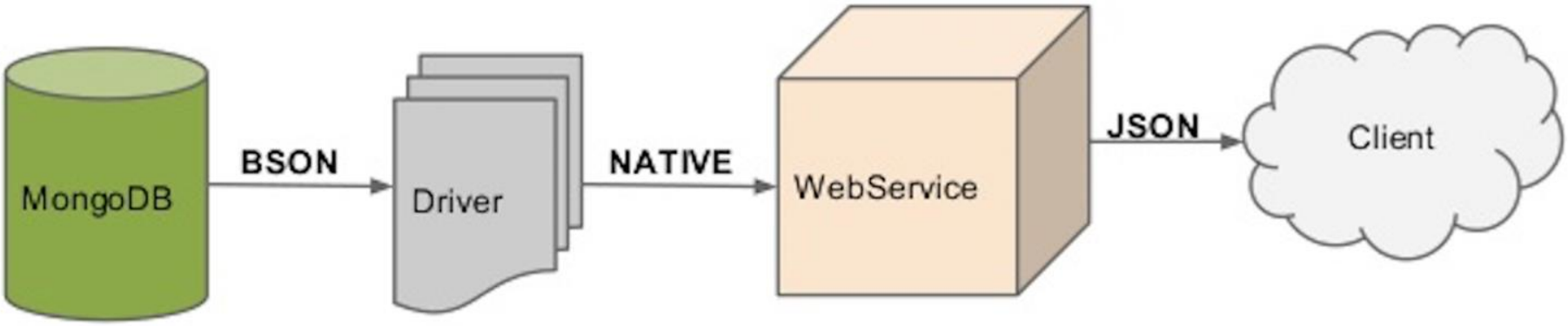
Normalized Data Model



How is the data stored?

BSON

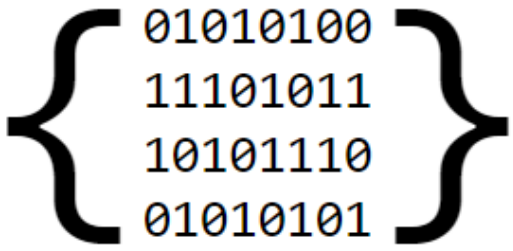
- Lightweight
- Traversable
- Efficient



JSON

```
{
  "array": [
    1,
    2,
    3
  ],
  "boolean": true,
  "null": null,
  "number": 123,
  "object": {
    "a": "b",
    "c": "d",
    "e": "f"
  },
  "string": "Hello World"
}
```

BSON

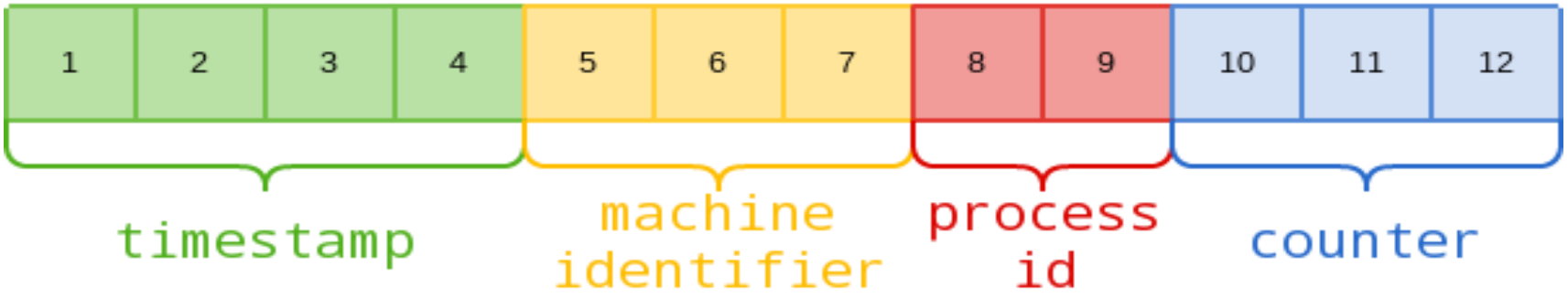


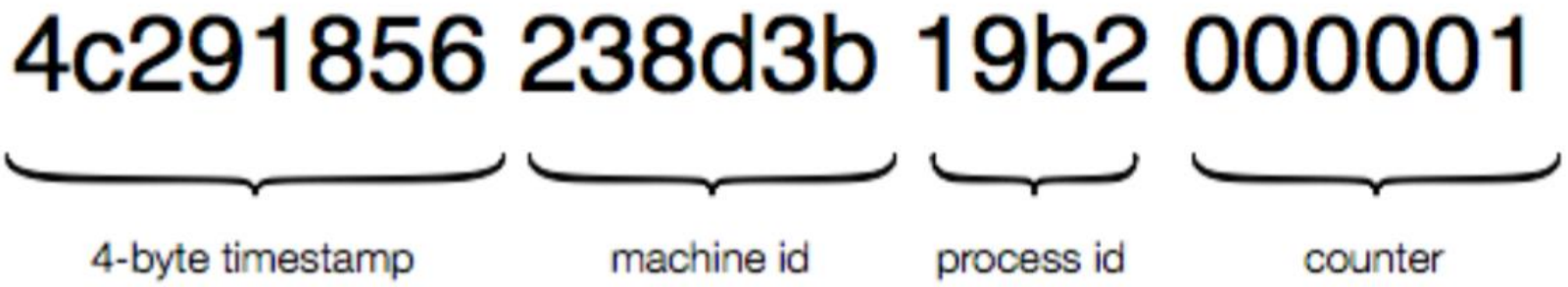

```
import org.bson.BasicBSONEncoder;
import org.bson.BasicBSONDecoder;
```

Sample JSON Data

```
{

    _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)

}
```

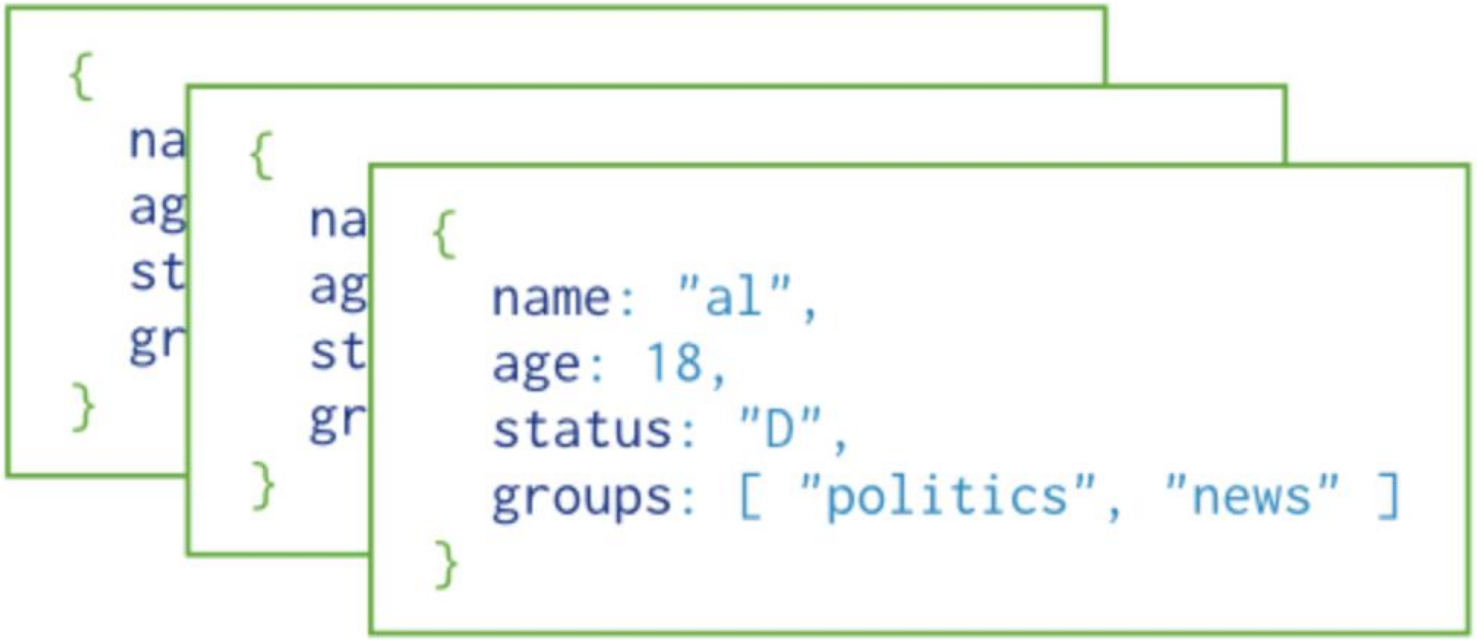
MongoDB Object Id format





MongoDB Collections

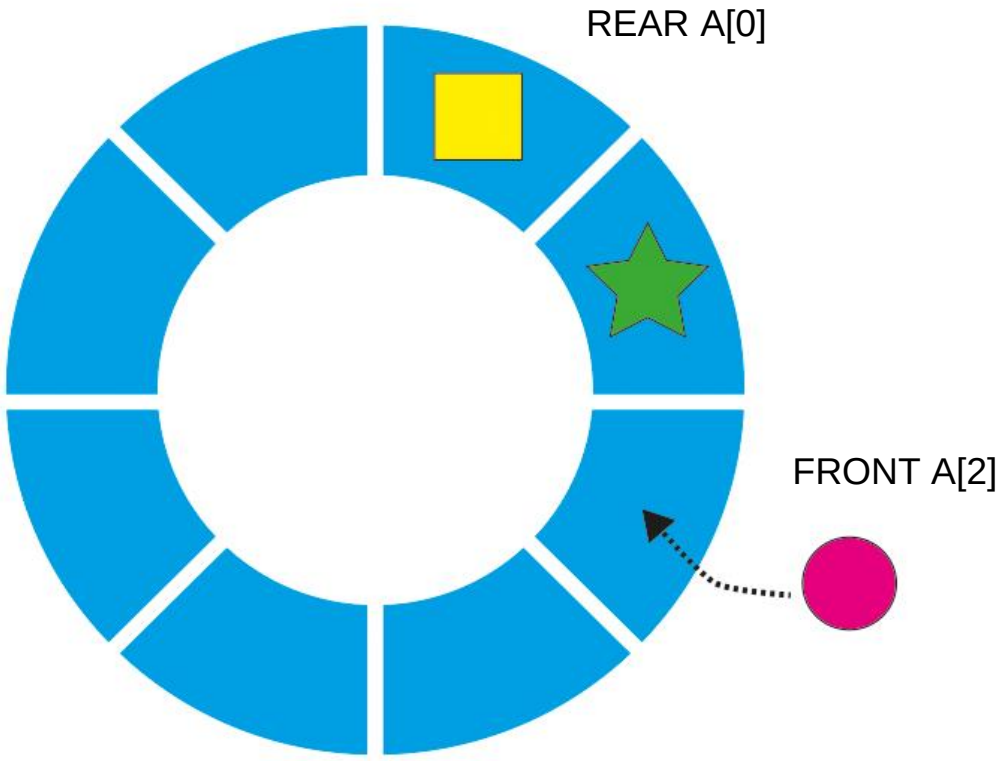
- MongoDB stores documents in collections



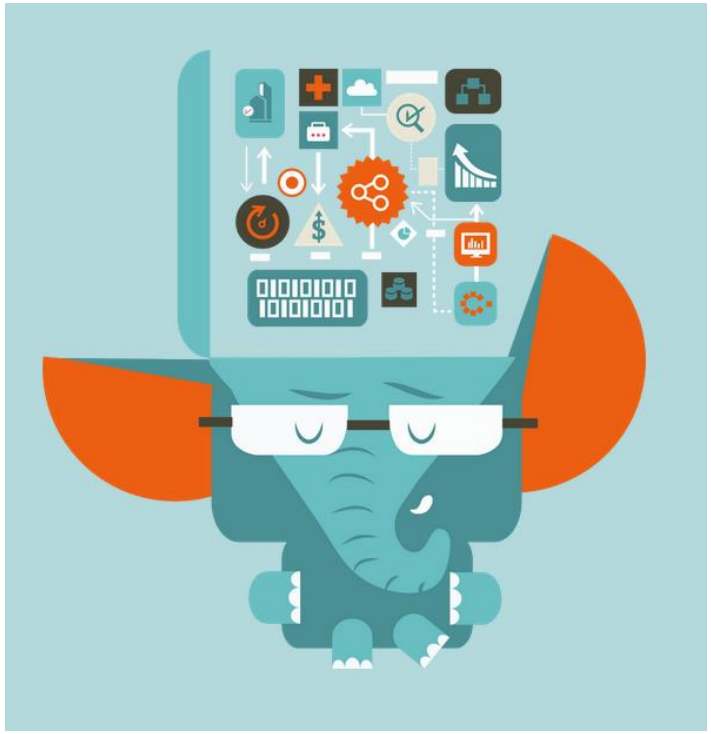
Capped Collections

Capped Collections

- Circular buffers

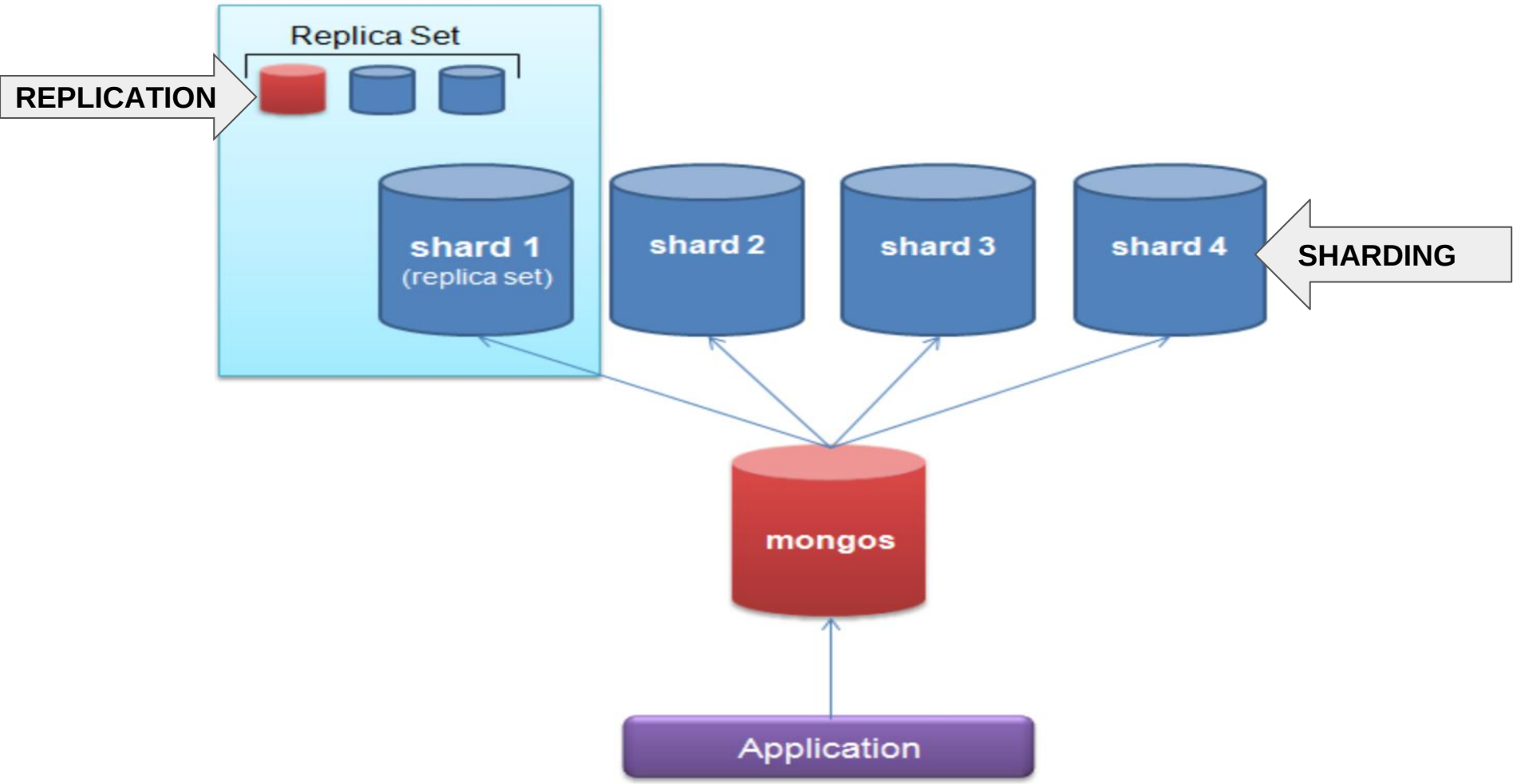


Humongous data: 2 main needs/issues?



1. Data backup
2. Scaling

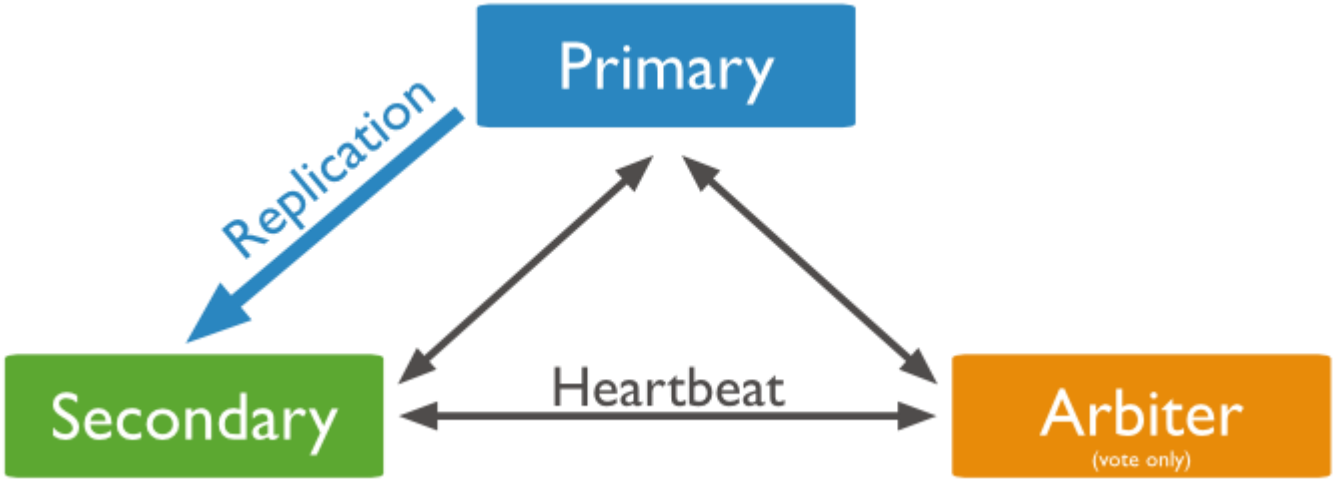
Replication and Sharding in mongoDB



Replication

Replica set

- 1 primary node
- 1+ secondary nodes
- Optional Arbiter node

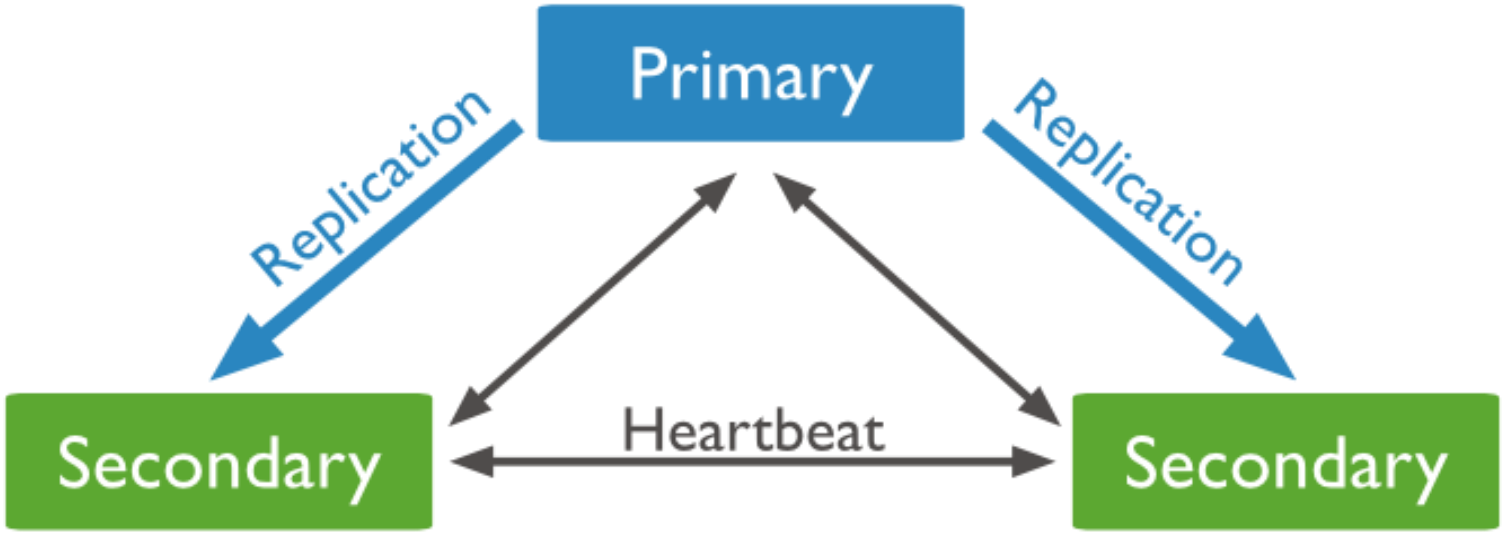


Minimum replica set configuration

Replication

Replica set

- 1 primary node
- 1+ secondary nodes
- Optional Arbiter node



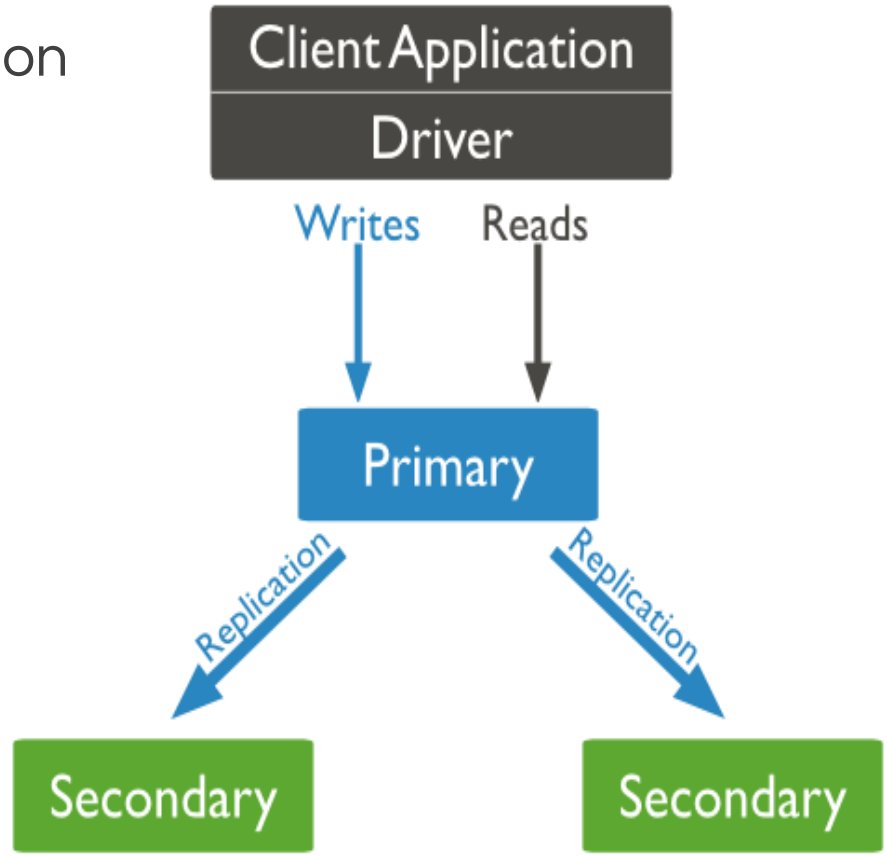
Replica set with 1 primary and 2 secondaries

Replication

Data synchronization
- oplog: capped collection

Write acknowledgement
- primary only (default)
- custom

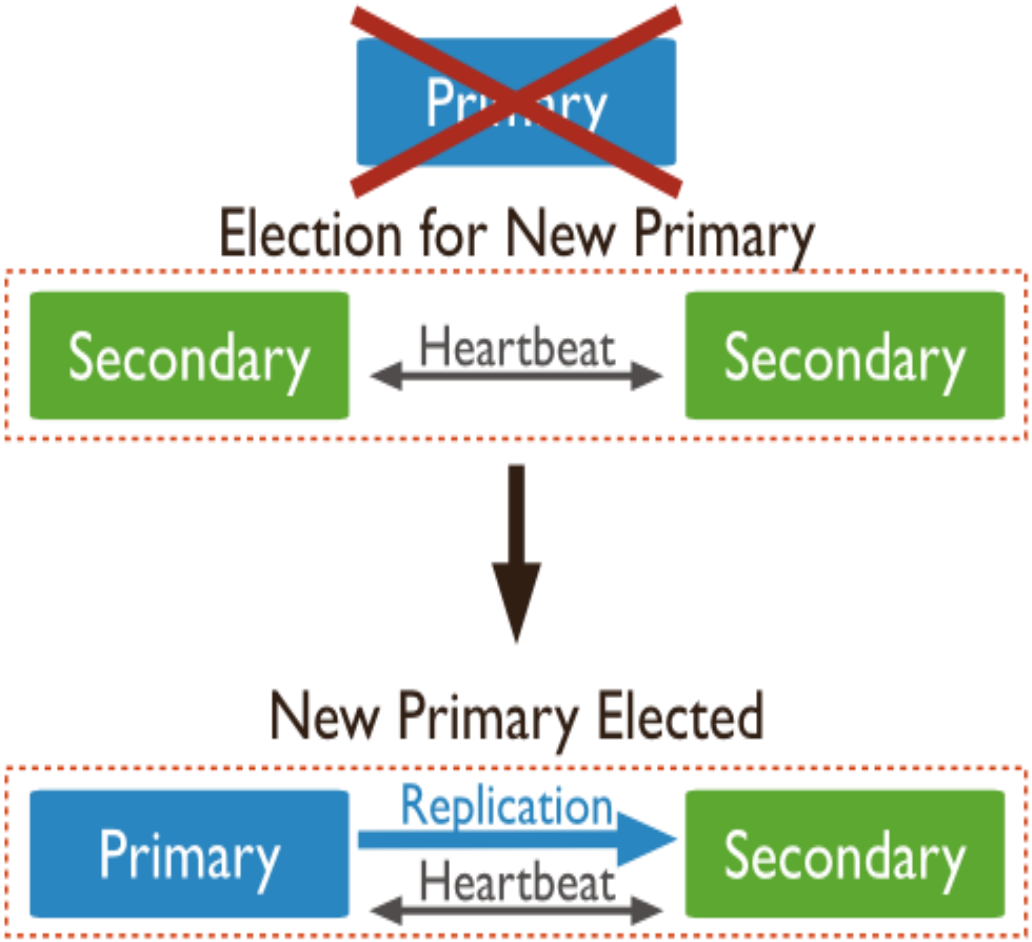
Read concern
- local (default)
- majority



Replication

Automatic failover

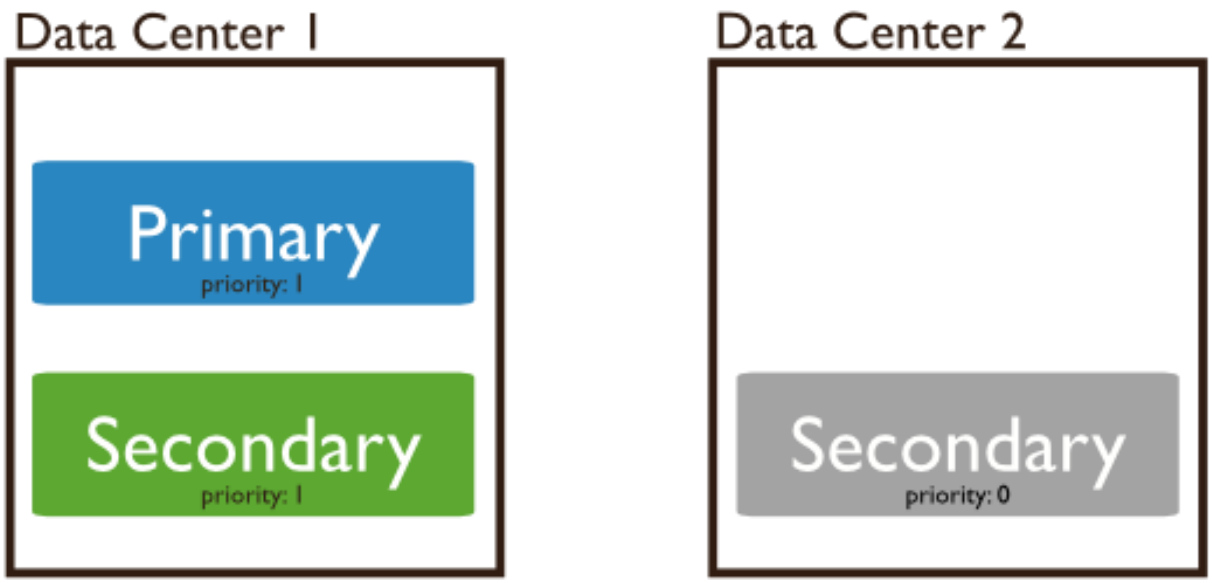
- Heartbeats
- Elections
  - Priorities
- Rollbacks



Replication

Replica set secondary members

- **Priority 0** - can't be primary. Use: standbys
- Hidden
- Delayed



Replication

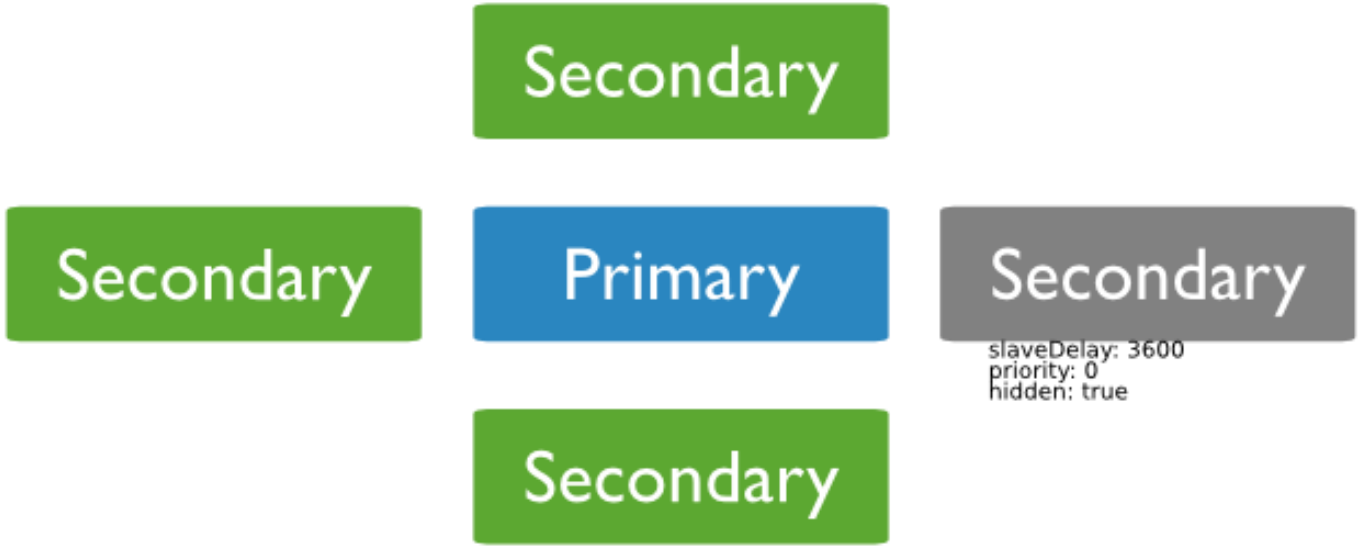
Replica set secondary members

- Priority 0 - can't be primary. Use: standbys
- **Hidden** - priority 0 + invisible to client
- Delayed

Replication

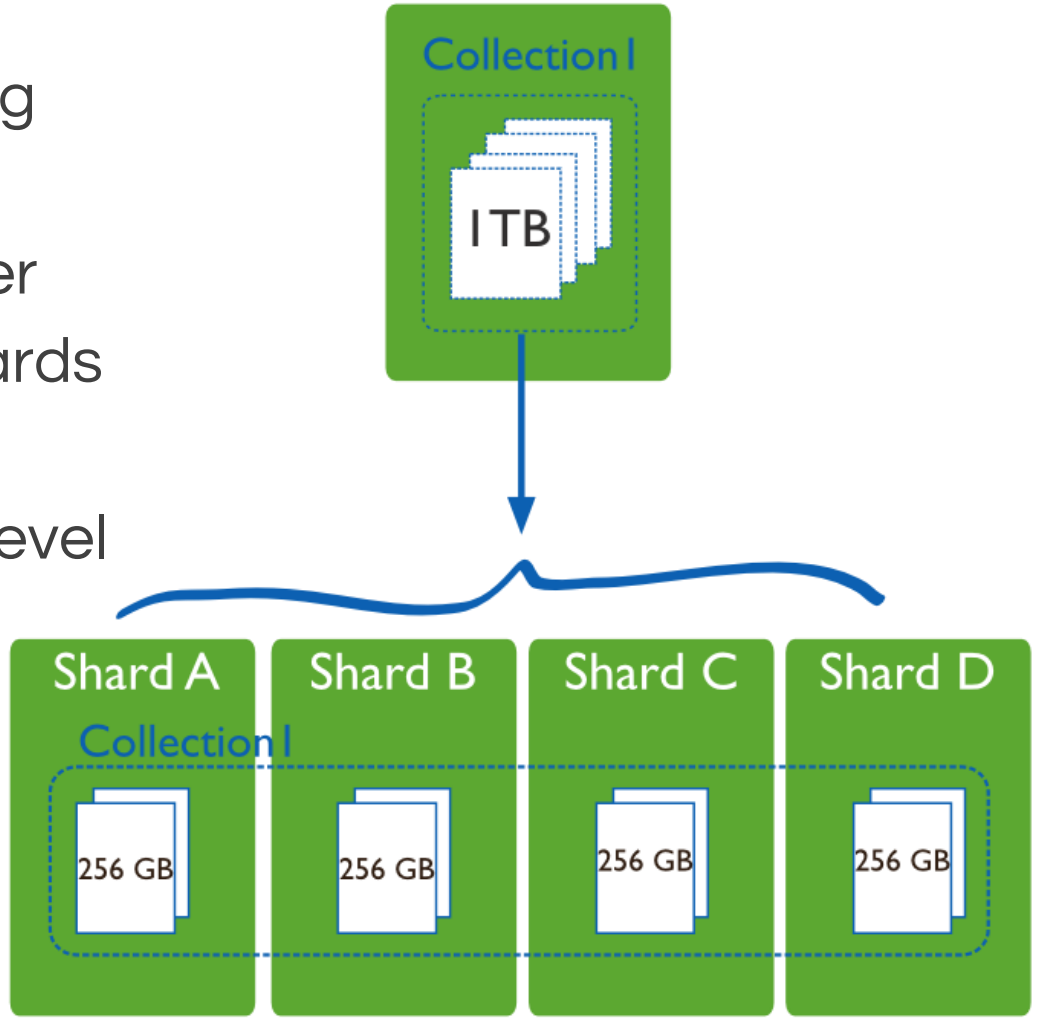
Replica set secondary members

- Priority 0 - can't be primary. Use: standbys
- Hidden - Priority 0 + invisible to client
- **Delayed** - Hidden. Historical snapshot. Use: error recovery



Sharding

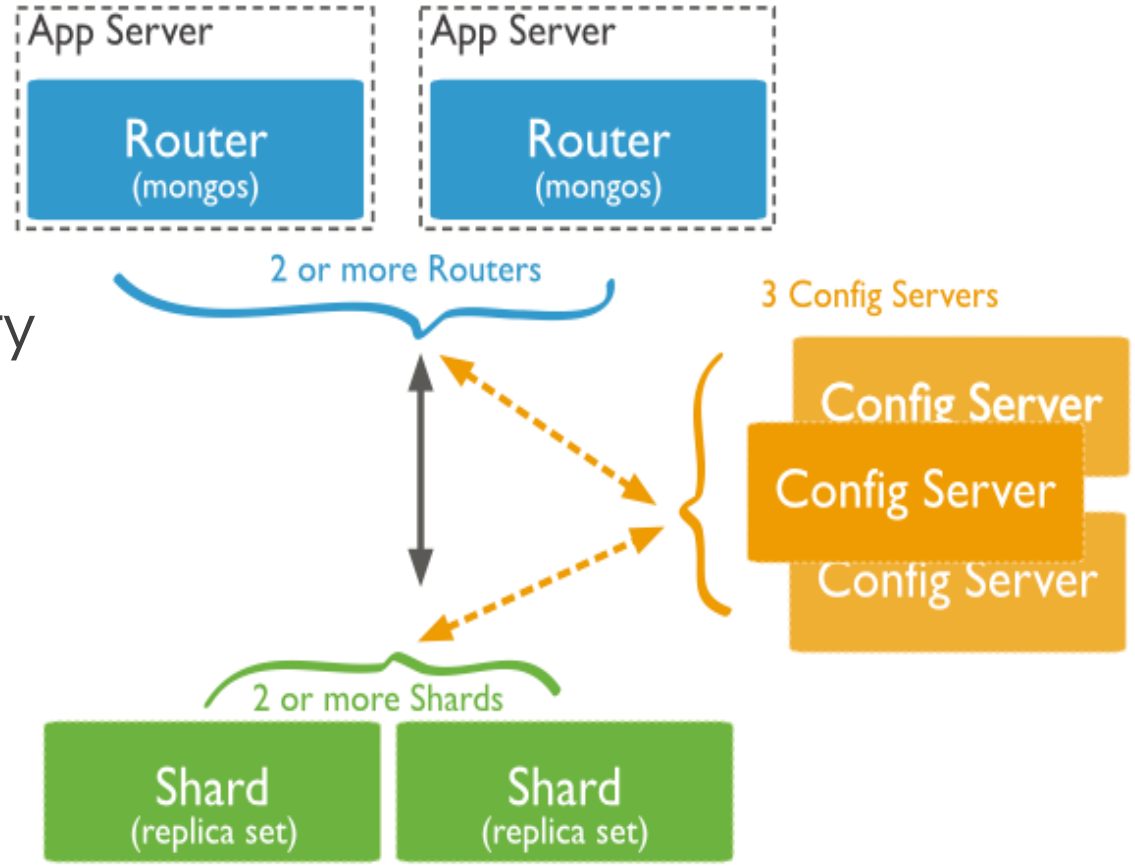
- Horizontal portioning
- Distributes data over multiple servers/shards
- Done at Collection level



Sharding

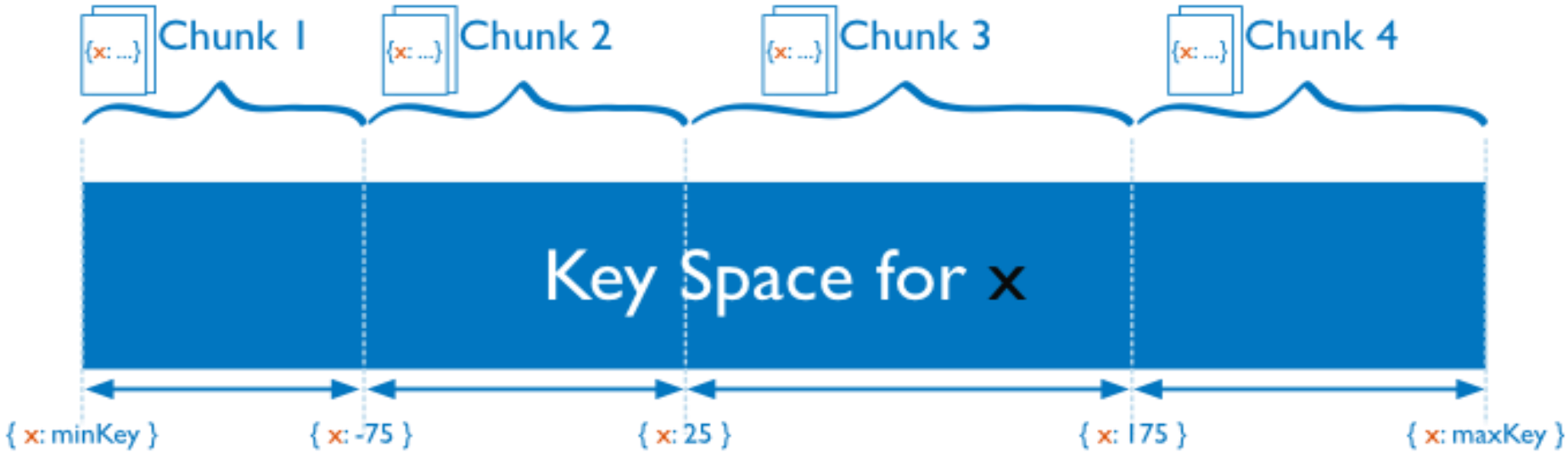
Sharded cluster

- Shard
- “mongos” query router
- Config server



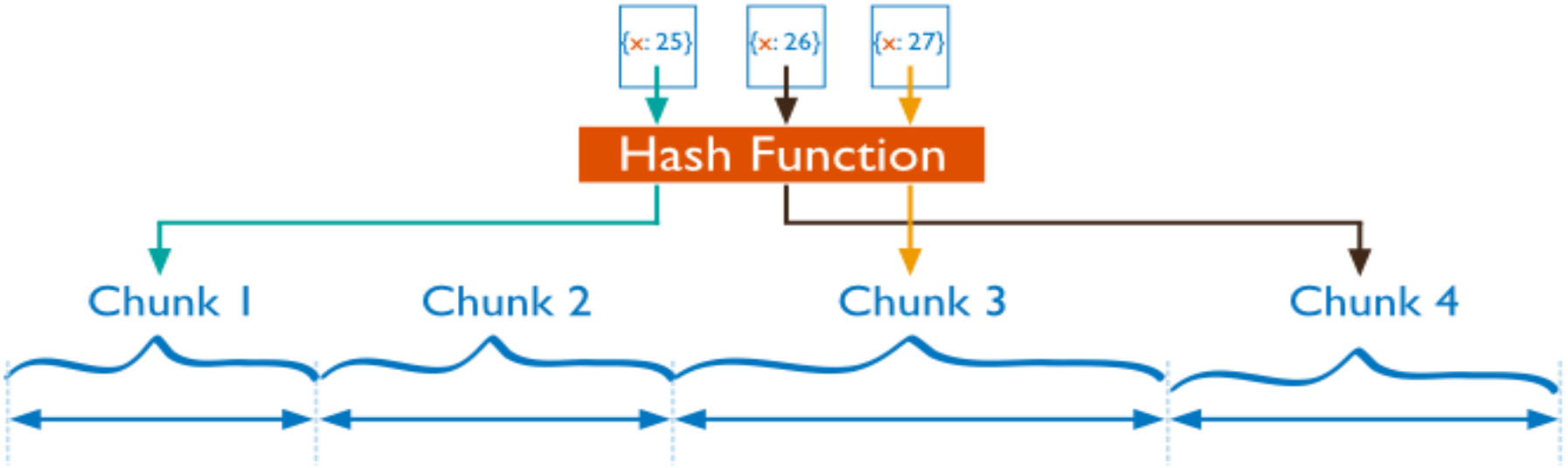
Sharding

- Shard key - used to partition collection
  - **Range based partitioning**
  - Hash based partitioning



Sharding

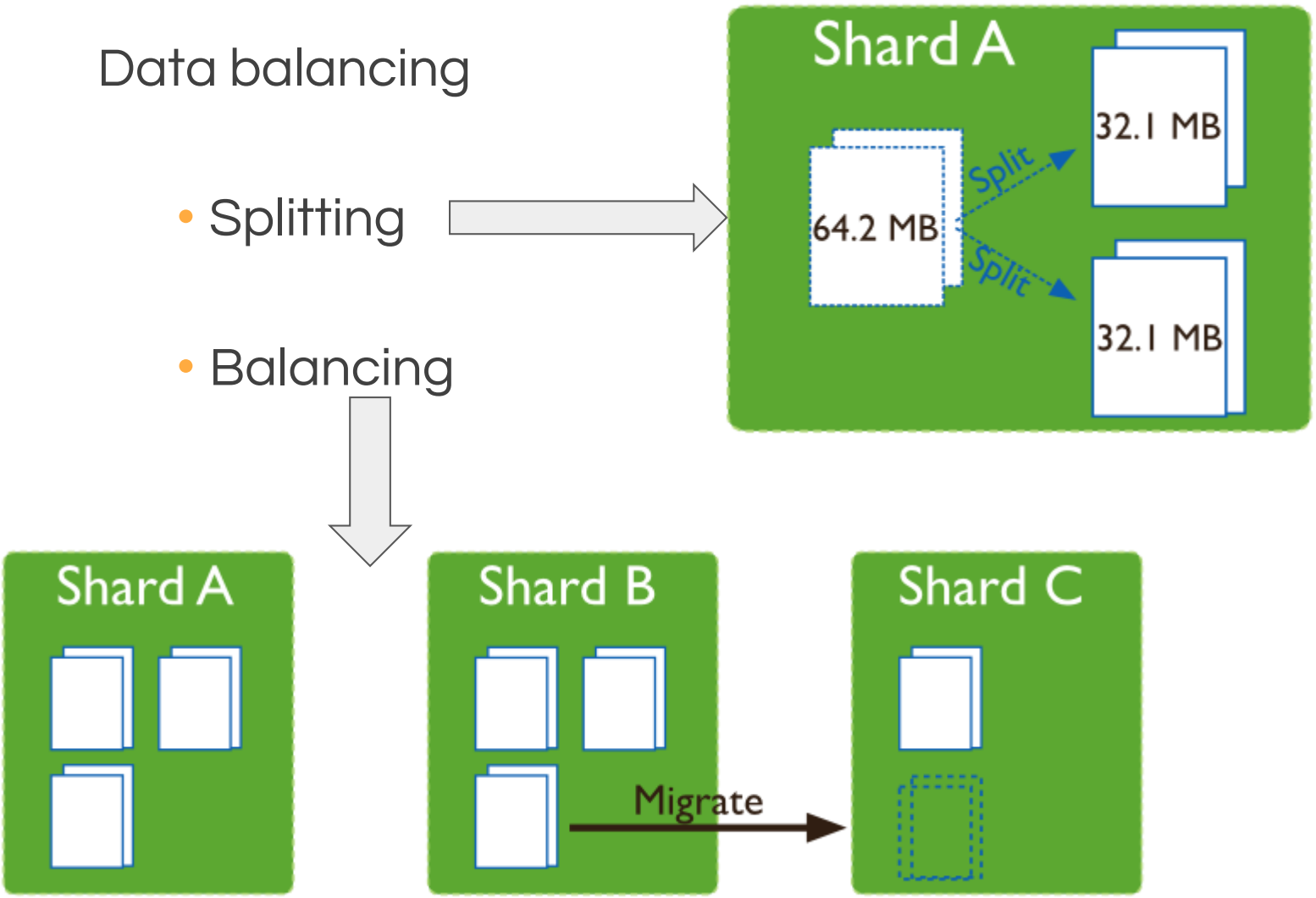
- Shard key - indexed field
  - Range based partitioning
  - **Hash based partitioning**



Sharding

Data balancing

- Splitting
- Balancing



Querying in MongoDB

- Uses mongo shell for querying data
- DB and Collections created automatically when first referenced
- **show** dbs ⟹ list of dbs
- **show** collections ⟹ collections in db

CRUD operations

- To insert document in a database:

db.collection.insert(document)

Collection

e.g. db.gryffindor.insert({
name: "Harry Potter",
age: 11
})

Document

CRUD operations

- To search in collection:

  db.collection.find(<filter>,<projection>)

- Filter => Boolean expression

  e.g {'age' : 14} or {'age' : { $lt : 18}}

- $lt, $gt, $in, $nin, $all etc..
- Projection => Fields to display

e.g. db.gryffindor.find({ name: "Harry Potter" })

CRUD operations

- To update a document :

db.collection.update(<filter>,<upd_oper>)

- Applies update operation to matches
- $set, $unset, $inc, $dec, $rename

e.g. db.gryffindor.update({ name: "Harry Potter" },

{ age: 19 })

filter

Update operation

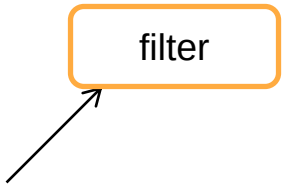
CRUD operations

- To delete a document:

```
db.collection.remove(<filter>)
```

- To delete all documents in collection:

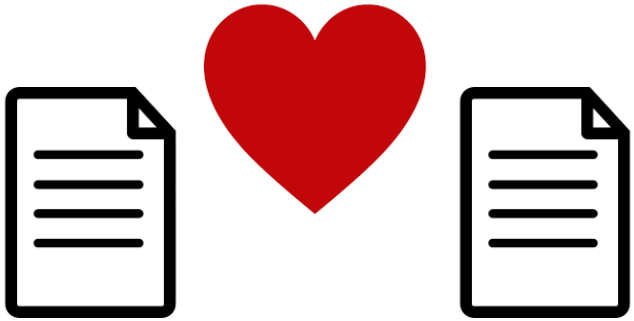
```
db.collection.drop()
```



e.g. db.gryffindor.remove({age: { $gt: 18 }})

Document Relationships

- 1-to-many relationships allowed
- Embedding or Referencing
- Embed one document inside another
- Link two docs by using their ids

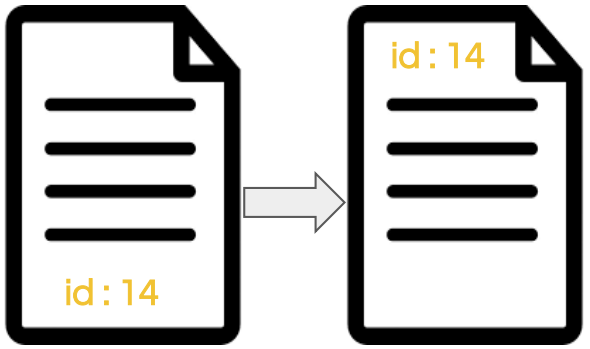


Document Relationships

Embedding

```
{ "_id" : "896",
 "name" : "The Goblet of Fire",
"author" :
                {
             "name" : "J.K. Rowling",
              "age"   : 51
            }
}
```

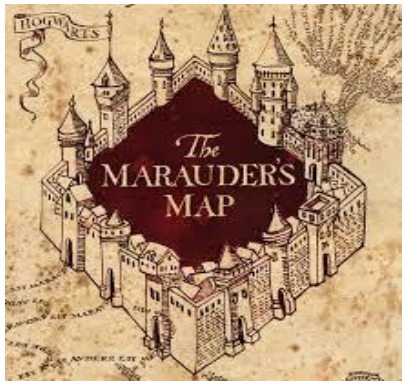
Referencing




```
{ "id" : "886",
  "name" : "The Goblet Of Fire",
  "author_id" : "896" }

{ "id" : "896",
  "name" : "J. K. Rowling",
  "age" : 51 }
```

```
db.collection.find("author.name" : "J.K. Rowling")
```

Advanced features



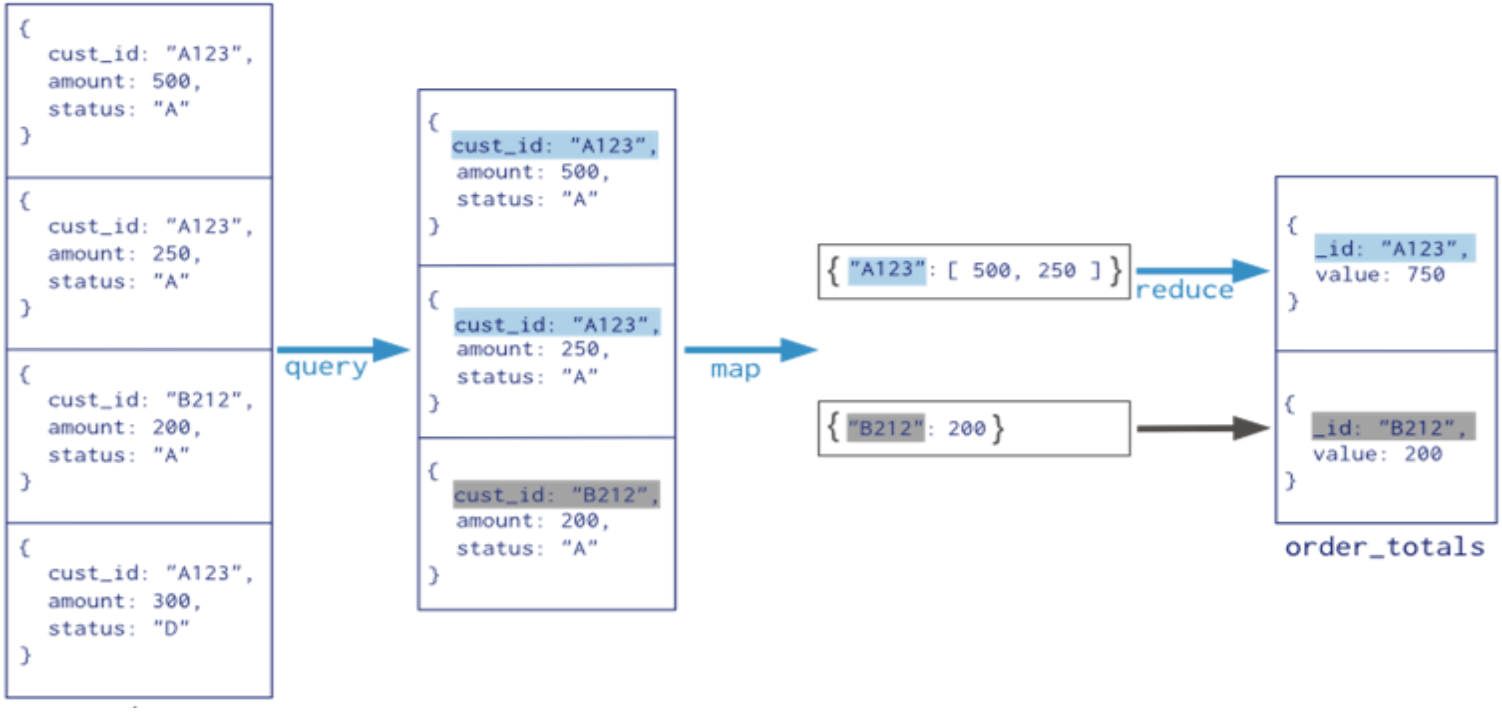
- Geospatial queries :
  - Objects with GeoJSON format
  - $near, $geoWithin, $geoIntersects
- Text search:
  - Using text indexes
  - $text => Full-text search
- Indexing:
  - On single, compound, embedded, arrayed obj

Aggregation in MongoDB

- 3 ways to aggregate !!
  - group(), count(), distinct() etc…
  - Simple grouping of documents
- Map/Reduce framework
  - Runs inside MongoDB
  - Outputs to document or collection

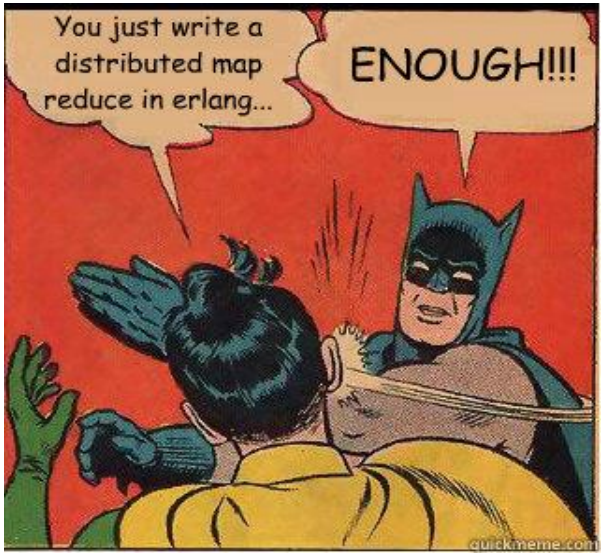
MapReduce in MongoDB

- Two functions: Map and Reduce
- Map => emits a key-value pair from processed document
- Reduce => Reduce all key-value pairs to single object



Aggregation pipeline

- Multi-stage pipeline
- Faster than MapReduce
- More flexible
- Support for sharded clusters



Aggregation framework

- You can
  - reshape document structure
  - filter documents
  - remove embedded documents
  - even (kind of) create joins on docs

All in aggregate!

Where is MongoDB used?


Some Companies using MongoDB in Production
craigslist
IGN
GILT
GROUPE
Eventbrite
Events Made Easy
github
intuit
BUSINESS
INSIDER
Disney
sourceforge
theguardian

Schema design - Real world use case

- Message Inbox
- History
- Multiple Identities

Message Inbox

Design Goals :

- Efficiently send new messages to recipients
- Efficiently read inbox

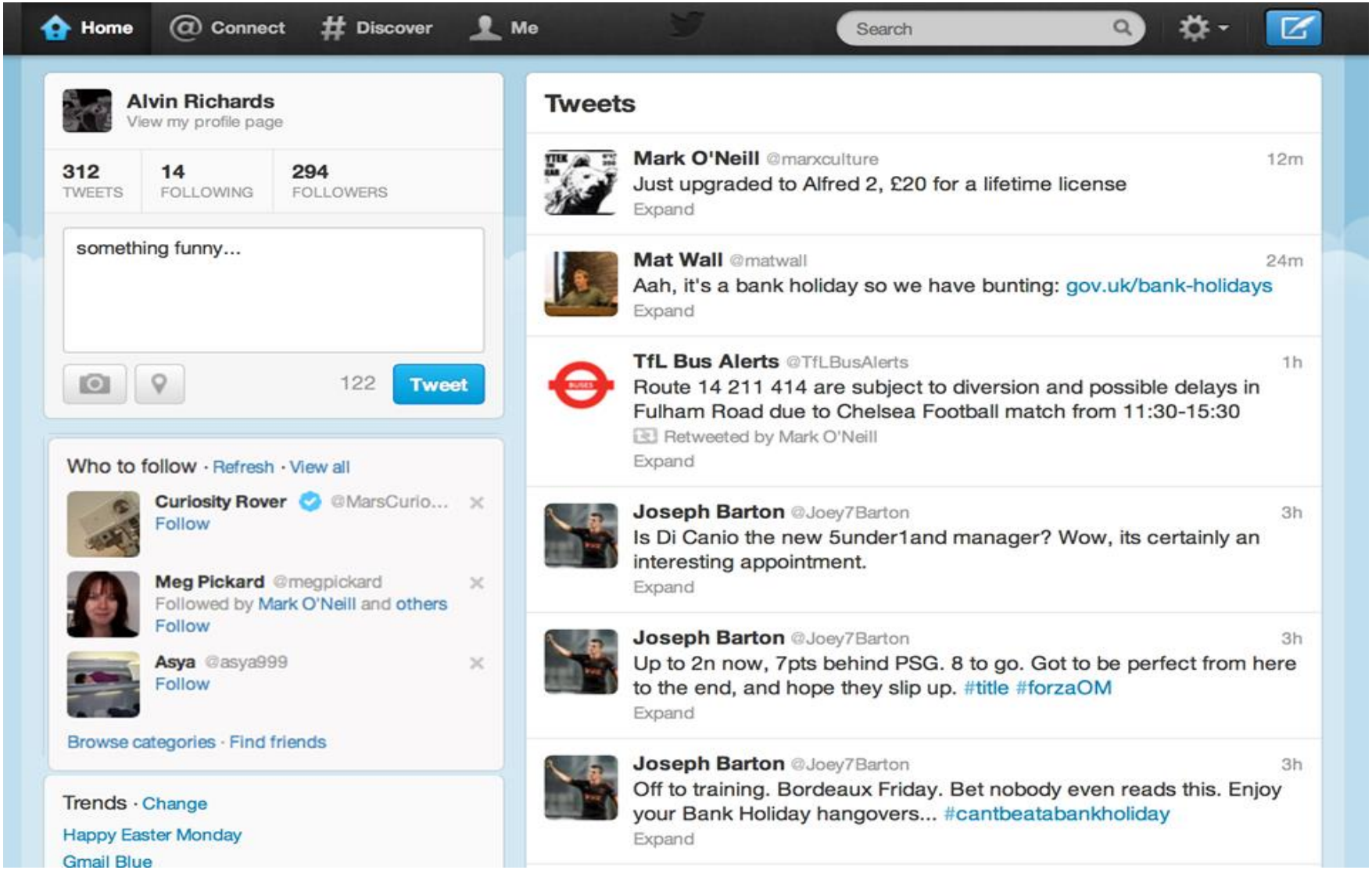
Considerations

- Each “inbox” document is an array of messages
- Append a message onto “inbox” of recipient
- Bucket inboxes
- Can shard on recipient, so inbox reads hit one shard
- 1 or 2 documents to read the whole inbox

```
// Shard on "owner / sequence"
db.shardCollection( "mongodbdays.inbox", { owner: 1, sequence: 1 } )
db.shardCollection( "mongodbdays.users", { user_name: 1 } )

msg = {
      from:    "Joe",
      to:      [ "Bob", "Jane" ],
      sent:    new Date(),
      message: "Hi!",
}
```

```
// Send a message
for( recipient in msg.to) {
      count = db.users.findAndModify({
            query: { user_name: msg.to[recipient] },
      update: { "$inc": { "msg_count": 1 } },
      upsert: true,
      new: true }).msg_count;

      sequence = Math.floor(count / 50);

      db.Inbox.update({
            owner: msg.to[recipient], sequence: sequence },
            { $push: { "messages": msg } },
            { upsert: true } );
}

// Read my inbox
db.inbox.find( { owner: "Joe" } ).sort ( { sequence: -1 } ).limit( 2 )
```

History


Home
Connect
Discover
Me
Search
Alvin Richards
View my profile page
312
TWEETS
14
FOLLOWING
294
FOLLOWERS
something funny...
122
Tweet
Who to follow · Refresh · View all
Curiosity Rover @MarsCurio...
Follow
Meg Pickard @megpickard
Followed by Mark O'Neill and others
Follow
Asya @asya999
Follow
Browse categories · Find friends
Trends · Change
Happy Easter Monday
Gmail Blue
Tweets
Mark O'Neill @marxculture 12m
Just upgraded to Alfred 2, £20 for a lifetime license
Expand
Mat Wall @matwall 24m
Aah, it's a bank holiday so we have bunting: gov.uk/bank-holidays
Expand
TfL Bus Alerts @TfLBusAlerts 1h
Route 14 211 414 are subject to diversion and possible delays in Fulham Road due to Chelsea Football match from 11:30-15:30
Retweeted by Mark O'Neill
Expand
Joseph Barton @Joey7Barton 3h
Is Di Canio the new 5under1and manager? Wow, its certainly an interesting appointment.
Expand
Joseph Barton @Joey7Barton 3h
Up to 2n now, 7pts behind PSG. 8 to go. Got to be perfect from here to the end, and hope they slip up. #title #forzaOM
Expand
Joseph Barton @Joey7Barton 3h
Off to training. Bordeaux Friday. Bet nobody even reads this. Enjoy your Bank Holiday hangovers... #cantbeatabankholiday
Expand

Design Goals

- Need to retain a limited amount of history
  - e.g. Hours, Days, Weeks
- Need to query efficiently by
  - match
  - ranges

Considerations

- TTL

```
// messages: one doc per user per day

db.inbox.findOne()
{

   _id: 1,
   to:  "Joe",
   sequence: ISODate("2013-02-04T00:00:00.392Z"),
   messages: [ ]
}

// Auto expires data after 31536000  seconds = 1 year
db.messages.ensureIndex( { sequence: 1 },
                         { expireAfterSeconds: 31536000 } )
```

Multiple Identities

Design Goals :

- Ability to look up by a number of different identities
  - Username
  - Email address
  - FB Handle
  - LinkedIn URL

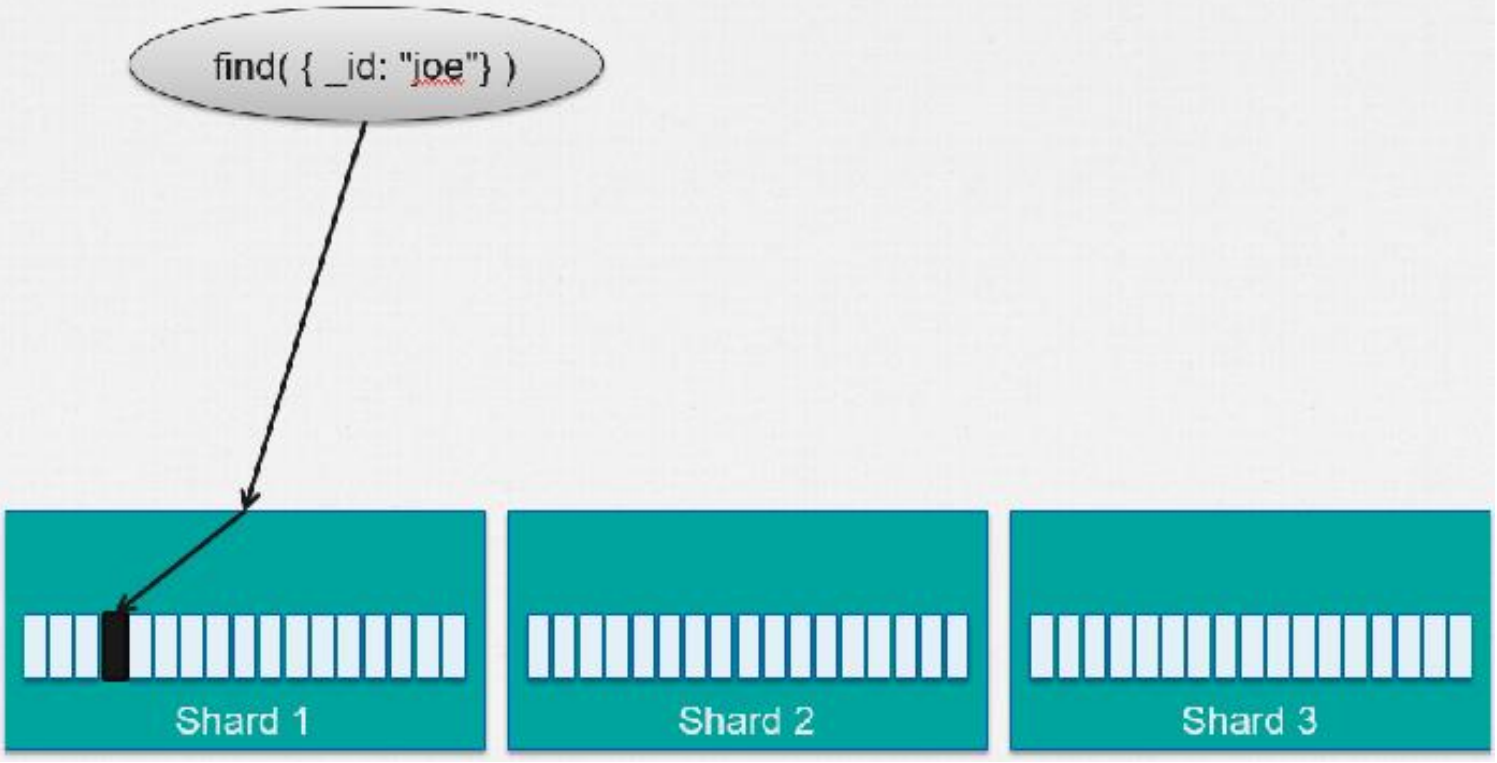
Approaches

- Identifiers in a single document
- Separate Identifiers from Content

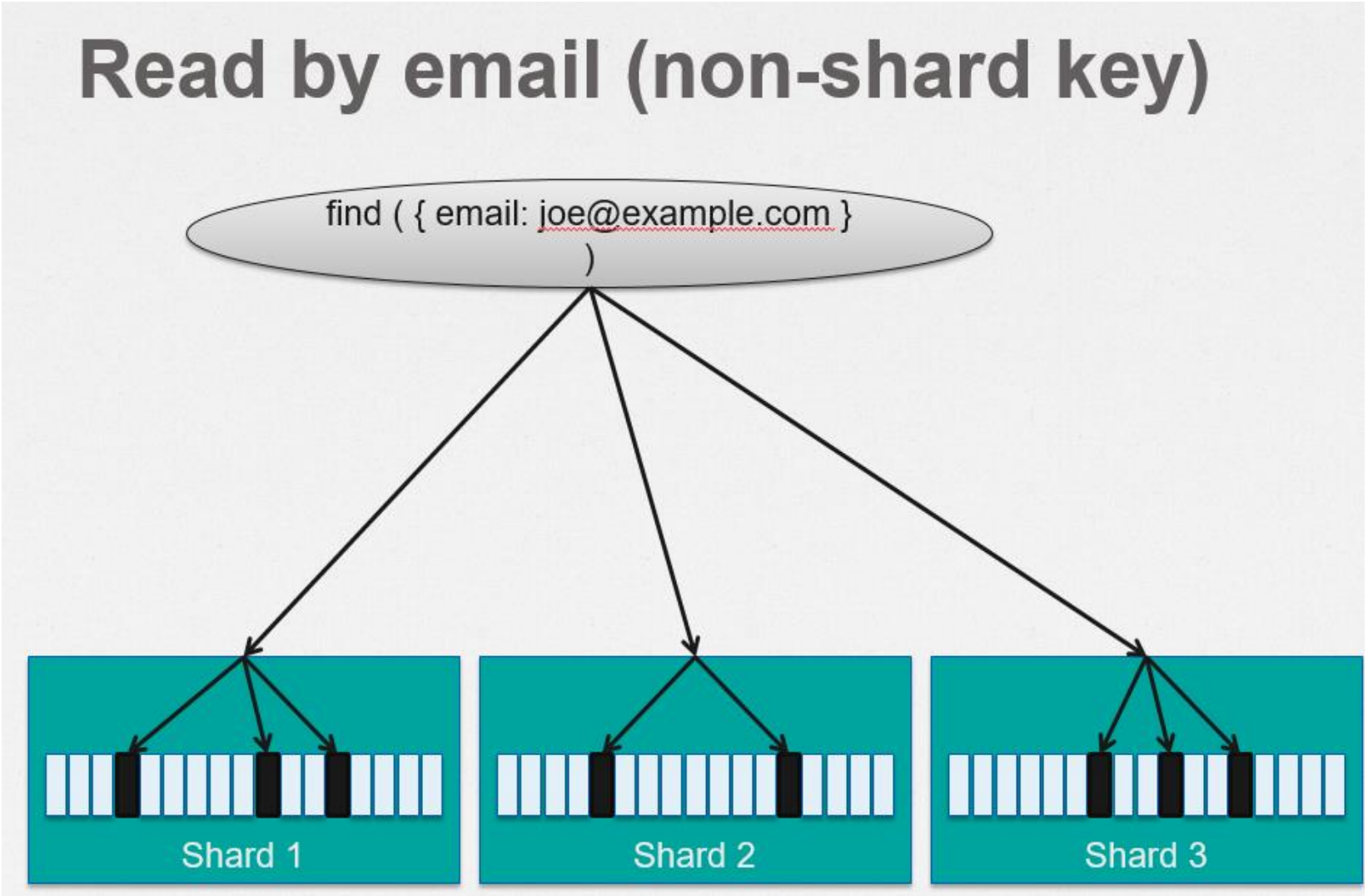
Single Document by User

```
db.users.findOne()
{ _id:    "joe",
  email:  "joe@example.com,
  fb:     "joe.smith",   // facebook
  li:     "joe.e.smith", // linkedin
  other: {...}
}


// Shard collection by _id
db.shardCollection("mongodbdays.users", { _id: 1 } )

// Create indexes on each key
db.users.ensureIndex( { email: 1} )
db.users.ensureIndex( { fb: 1 } )
db.users.ensureIndex( { li: 1 } )
```

Read by _id (shard key)

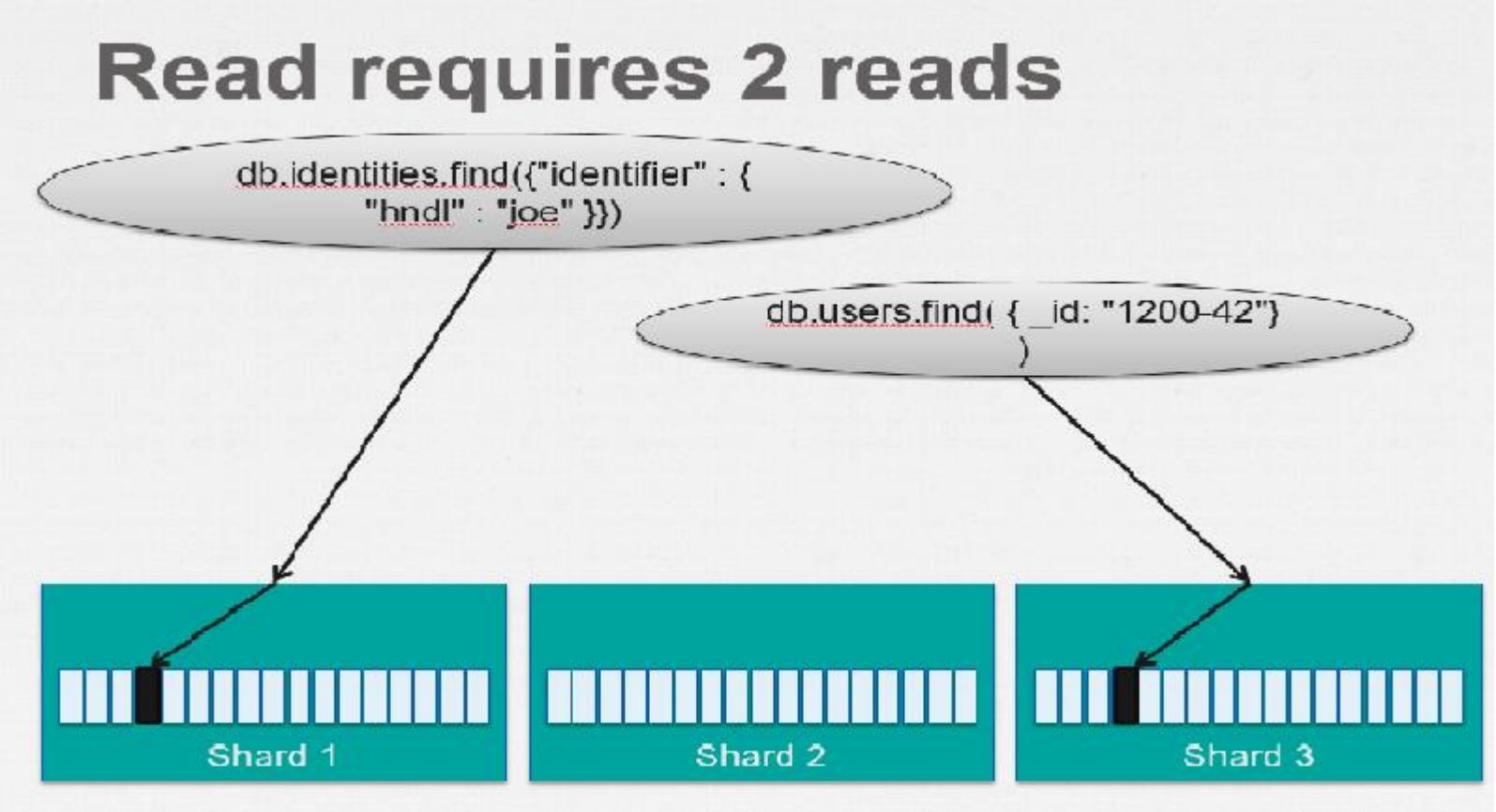


Read by email (non-shard key)



Document per Identity

```
// Create unique index
db.identities.ensureIndex( { identifier : 1} , { unique: true} )

// Create a document for each users document
db.identities.save(
   { identifier : { hndl: "joe" },          user: "1200-42" } )
db.identities.save(
   { identifier : { email: "joe@abc.com" }, user: "1200-42" } )
db.identities.save(
   { identifier : { li:  "joe.e.smith" },   user: "1200-42" } )

// Shard collection by _id
db.shardCollection( "mydb.identities", { identifier : 1 } )

// Create unique index
db.users.ensureIndex( { _id: 1} , { unique: true} )

// Shard collection by _id
db.shardCollection( "mydb.users", { _id: 1 } )
```


Read requires 2 reads
db.identities.find({"identifier" : {
"hndl" : "joe" }})
db.users.find({ _id: "1200-42"}
)
Shard 1
Shard 2
Shard 3

Real World Use Cases


mongoDB


Etsy
Viber
DigiDoc3
URBAN AIRSHIP
Shareaholic

Reasons

Their arguments center around a few core themes:

- Product Maturity
- Design Decisions
- Wrong Trade-Offs

Less Suited Applications

- Complex transactions such as banking systems and accounting.
- Traditional relational data warehouses
- Problem requiring SQL

Best Suited Applications

- Archiving and Event Logging
- Content Management System
- Gaming
- Mobile
- Real time stats/Analytics

To mongoDB or not to mongoDB?