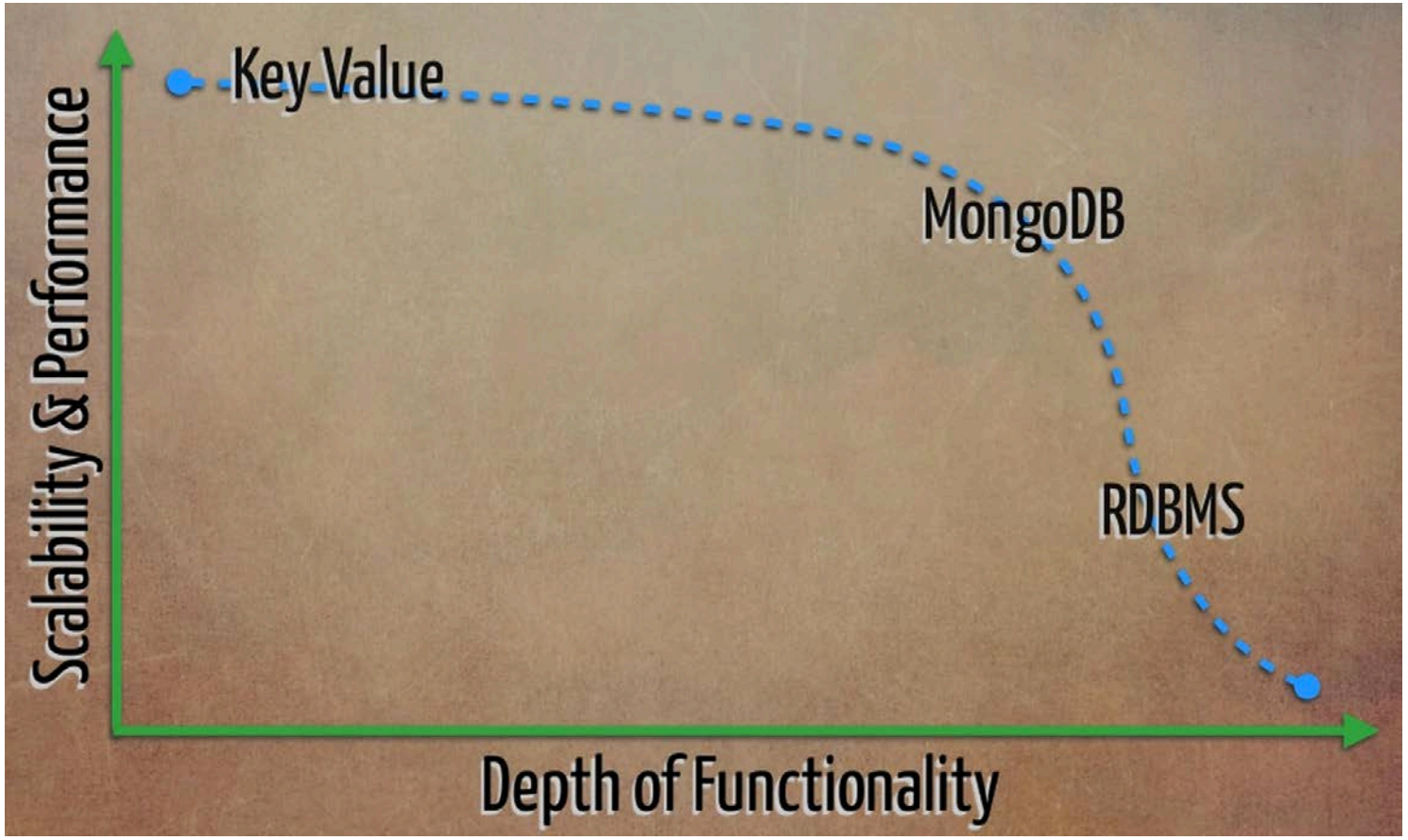
AngularJS

# Introduction to MongoDB

LUXOFT

# Database compared



# What is MongoDB ?

- Scalable High-Performance Open-source, Document-orientated database.
- Built for Speed
- Rich Document based queries for Easy readability.
- Full Index Support for High Performance.
- Replication and Failover for High Availability.
- Auto Sharding for Easy Scalability.
- Map / Reduce for Aggregation.

# Why use MongoDB?

- SQL was invented in the 70's to store data.
- MongoDB stores documents (or) objects.
- Now-a-days, everyone works with objects (Python/Ruby/Java/etc.)
- And we need Databases to persist our objects. Then why not store objects directly ?
- Embedded documents and arrays reduce need for joins. No Joins and No-multi document transactions.

# What is MongoDB great for?

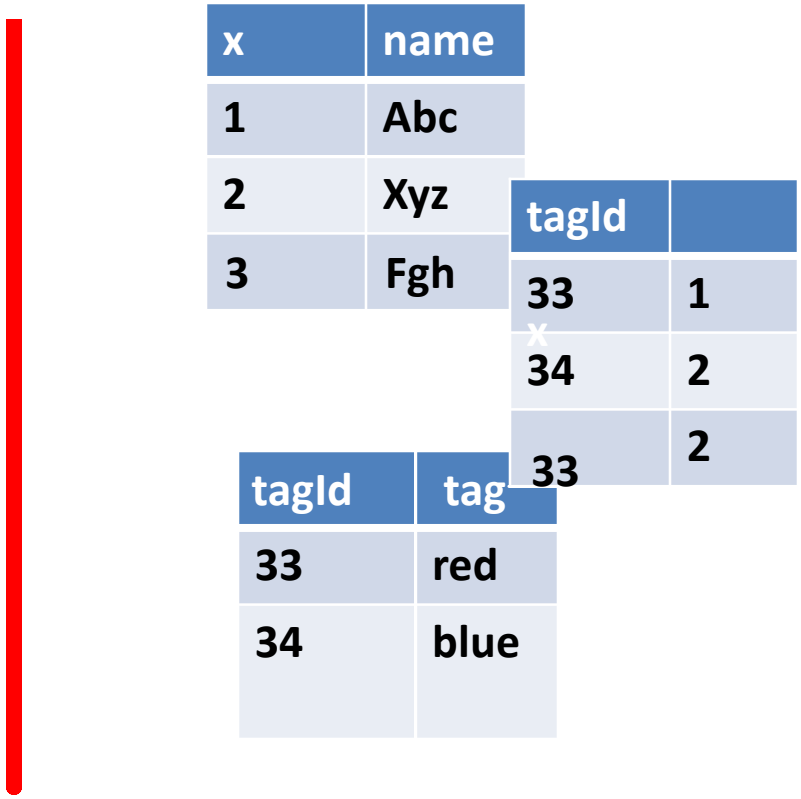
- RDBMS replacement for Web Applications.
- Semi-structured Content Management.
- Real-time Analytics & High-Speed Logging.
- Caching and High Scalability

# Not great for?

- Highly Transactional Applications.
- Problems requiring SQL.

# Impedance Mismatch

```
//  your application code
class Foo { int x; string [] tags;}
```



| x | name |
|---|---|
| 1 | Abc |
| 2 | Xyz |
| 3 | Fgh |

| tagId | |
|---|---|
| 33 | 1 |
| 34 | 2 |
| 33 | 2 |

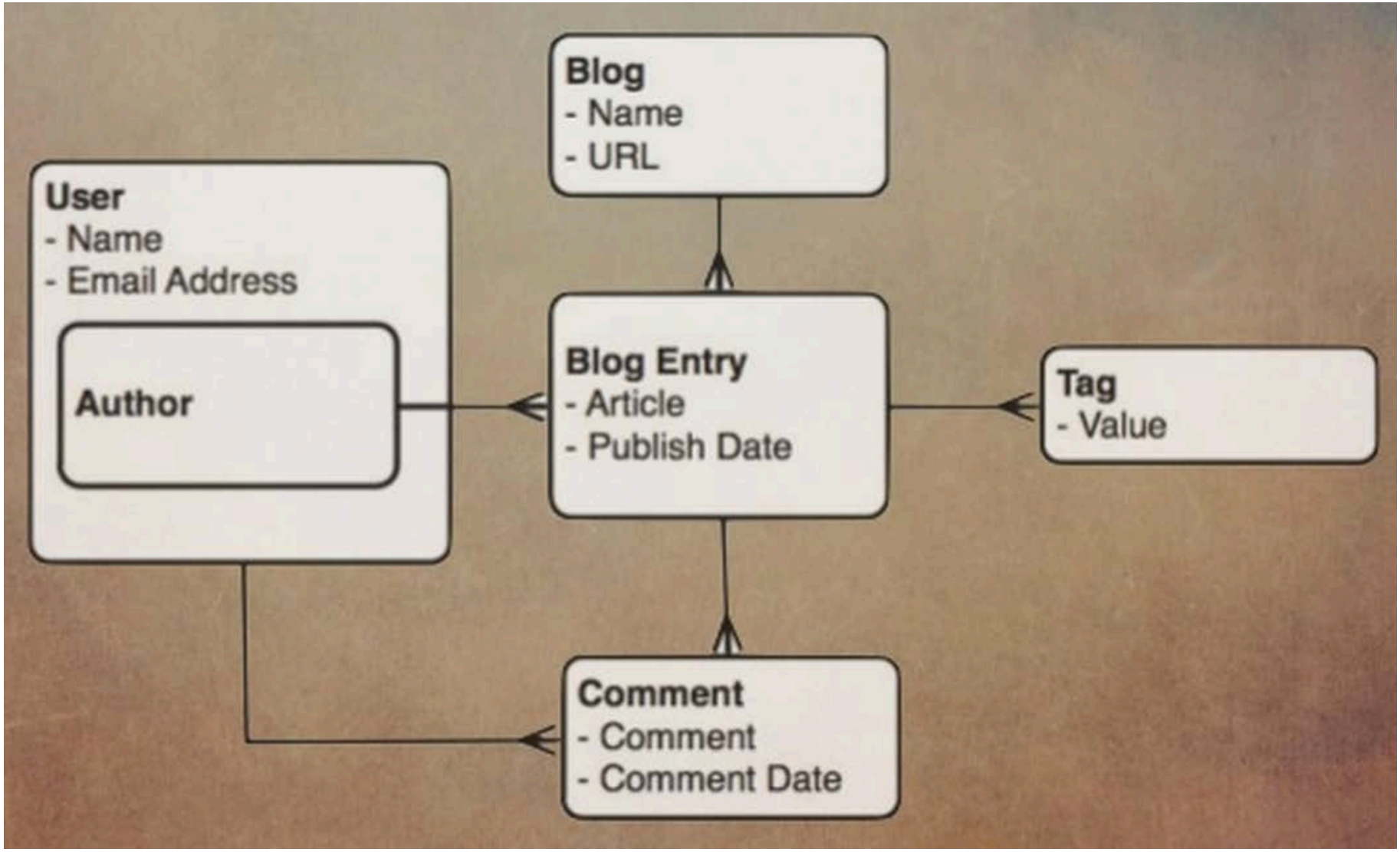
| tagId | tag |
|---|---|
| 33 | red |
| 34 | blue |

# No Impedance Mismatch

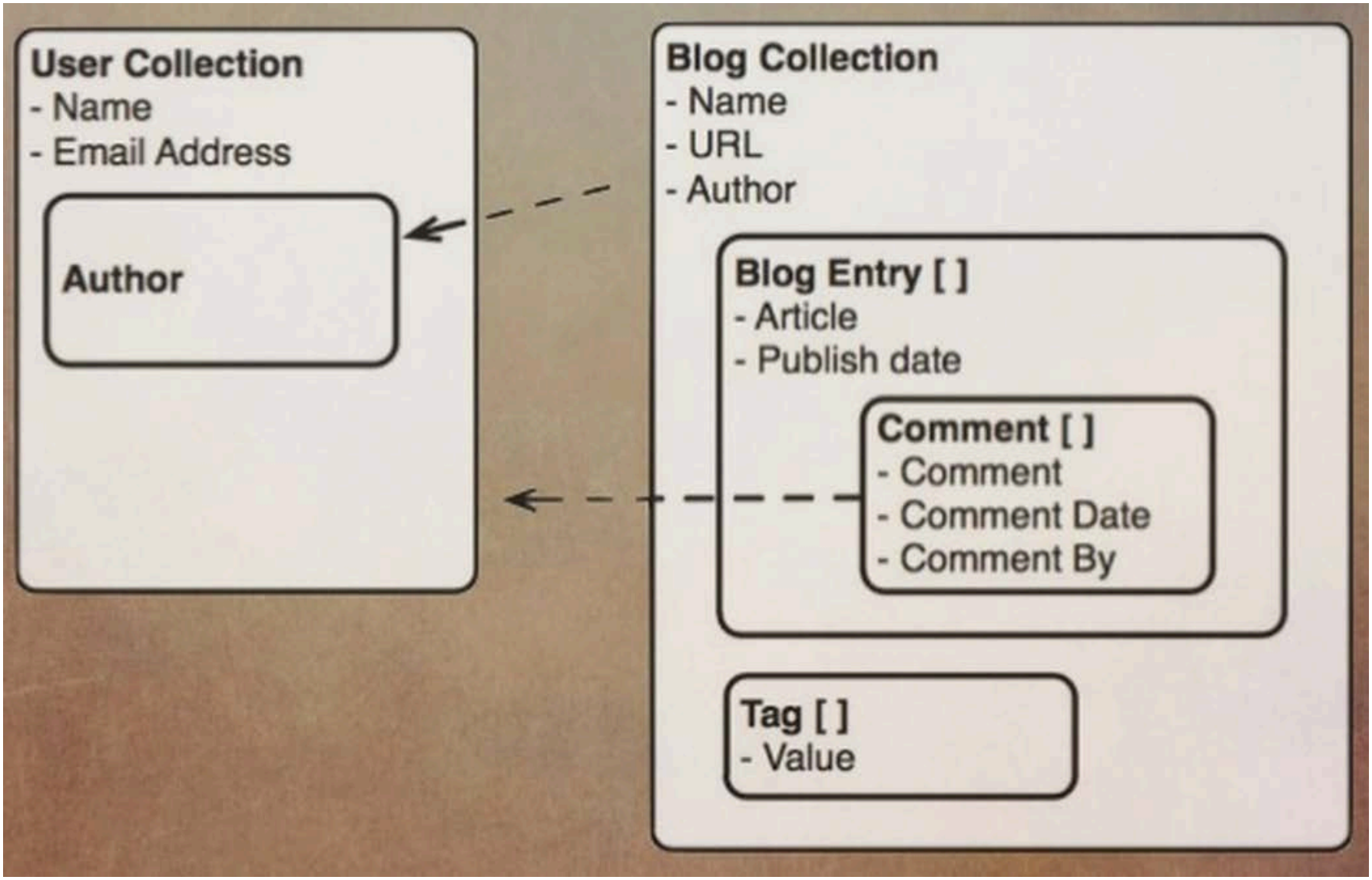
```
// your application code
class Foo { int x; string [] tags;}

// mongo document for Foo
{ x: 1, tags: [‘abc’,’xyz’] }
```

# Blog in relational DB



# Blog post structure in document DB



# Blog post in JSON DB

```
{   _id : ObjectId("4c4ba5c0672c685e5e8aabf3"),
    author : "steve",
    date : "Sat Apr 24 2013 19:47:11",
    text : "About MongoDB...",
    tags : [ "tech", "databases" ],
    comments : [
            {
                author : "Fred",
                date : "Sat Apr 25 2013 20:51:03 GMT-0700",
                text : "Best Post Ever!"
            }
     ]
}
```

When I say

**Database**

Think

**Database**

- Made up of Multiple Collections.
- Created on-the-fly when referenced for the first time.

**When I say**

# Collection

**Think**

# Table

- Schema-less, and contains Documents.
- Indexable by one/more keys.
- Created on-the-fly when referenced for the first time.
- Capped Collections: Fixed size, older records get dropped after reaching the limit.

When I say

# Document

Think

# Record/Row

- Stored in a Collection.
- Have _id key – works like Primary keys in MySQL.
- Supported Relationships – Embedded (or) References.
- Document storage in BSON (Binary form of JSON).

# Understanding the Document Model

```
var post = {
        '_id': ObjectId('3432'),
        'author': ObjectId('2311'),
        'title': 'Introduction to MongoDB',
        'body': 'MongoDB is an open sources.. ',
        'timestamp': Date('01-04-12'),
        'tags': ['MongoDB', 'NoSQL'],
        'comments': [{'author': ObjectId('5331'),
                                'date': Date('02-04-12'),
                                'text': 'Did you see.. ',
                                'upvotes': 7} ]
}

> db.posts.insert(post);
```

# The Problem: slow search

- **You say:**

```
db.foo.find({ x: 10 })
```

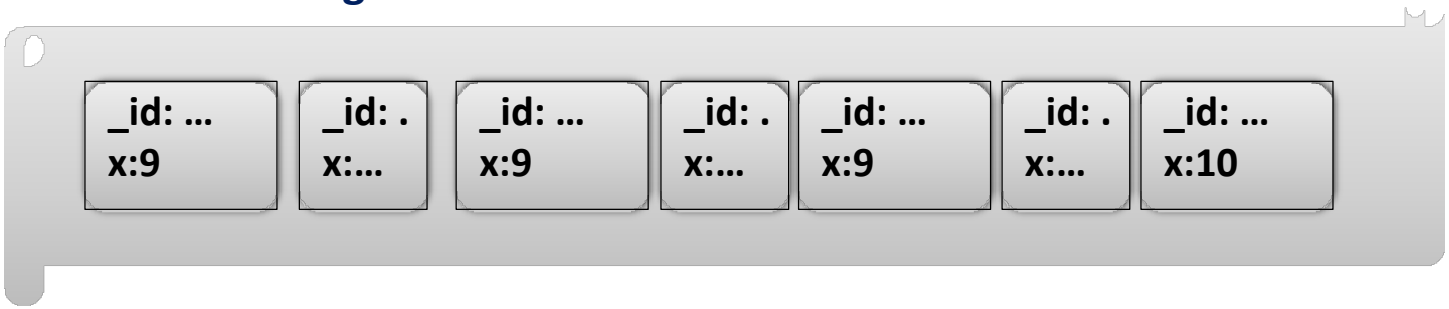
Ouch!

Reads EVERY document!

- **The server does : (pseudo)**

```
for each doc d in 'foo'{
     if ( d.x == 10 ){
                return d
          }
}
```

**Document Storage**



# Solution: indexes

Create Index on any field in the document

```
// 1 means ascending, -1 means descending
> db.posts.ensureIndex({‘author’: 1});

//Index Nested Documents
> db.posts.ensureIndex(‘comments.author’: 1);

// Index on tags
> db.posts.ensureIndex({‘tags’: 1});

// Geo-spatial Index
> db.posts.ensureIndex({‘author.location’: ‘2d’});
```

# Create Index

**Which fields?**
**In what Order?**
**Geo / Text**

```
db.foo.ensureIndex(keys, options)
```

**Collection**

**Name?**
**Build now?**
**Unique**
**Sparse?**
**TTL?**
**Language?**

# Find

```
// find posts which has ‘MongoDB’ tag.
> db.posts.find({tags: ‘MongoDB’});

// find posts by author’s comments.
> db.posts.find({‘comments.author’: ‘Johnson’}).count();

// find posts written after 31st March.
> db.posts.find({‘timestamp’: {‘$gte’: Date(’31-03-12’)}});

// find posts written by authors around [22, 42]
> db.posts.find({‘author.location’: {‘$near’:[22, 42]}});
```

$gt, $lt, $gte, $lte, $ne, $all, $in, $nin…

# Find: projection

**> db.posts.find({}, {title:1})**

```
{ "_id" : ObjectId("5654381f37f63ffc4ebf1964"),
        "title" : "NodeJS server" }
{ "_id" : ObjectId("5654385c37f63ffc4ebf1965"),
        "title" : "Introduction to MongoDB" }
```

Like

select **title** from **posts**

Empty projection like

select ***** from **posts**

# Find

**Which documents?**

```
db.foo.find(query, projection)
```

**Which fields?**

# Find

**Find**

- Query criteria
  - Single value field
  - Array field
  - Sub-document / dot notation

**Projection**

- Filed inclusion and exclusion

**Cursor**

- Sort
- Limit
- Skip

# Paging example

```
var post = {
        ‘_id’: ObjectId(‘3432’),
        ‘author’: ObjectId(‘2311’),
        ‘title’: ‘Introduction to MongoDB’,
        ‘body’: ‘MongoDB is an open sources.. ‘,
        ‘timestamp’: Date(’01-04-12’),
        ‘tags’: [‘MongoDB’, ‘NoSQL’]
}
```

```
> db.posts.insert(post)

var per_page = 10;
var page_num = 3;

db.posts
        .find({ 'tags': 'MongoDB'})
        .sort({‘timestamp': -1})
        .skip((page_num - 1) * per_page)
        .limit(per_page);
```

# Update: replace the document

```
> db.posts.update(
        {"_id" : ObjectId("5654381f37f63ffc4ebf1964")},
        {
                title:"NodeJS server"
        });
```

This will **replace** the document by {title:"NodeJS server"}

# Update: change only the part of document

```
> db.posts.update(
        {"_id" : ObjectId("5654381f37f63ffc4ebf1964")},
        {
                $addToSet: {tags:"JS"},
                $set: {title:"NodeJS server"},
                $unset: { comments: 1}
        });
```

$set, $unset

$push, $pull, $pop, $addToSet

$inc, $decr, many more…

# Update

**Which Document?**

```
db.foo.update(query,update,options);
```

**Collection Name**

**What to Change?**

**One? Many? Upsert?**

**Options**:

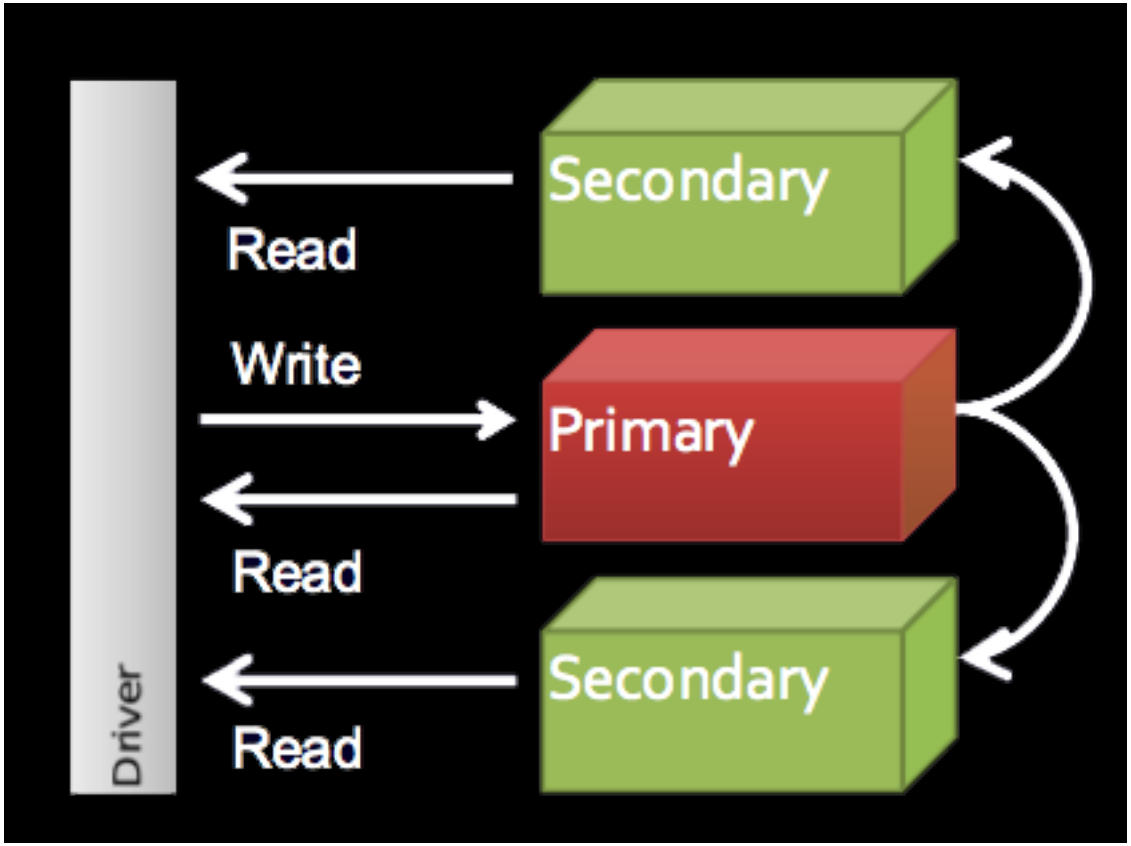
**{multi: true}** – will change all found documents;
by default only first found will be updated

**{upsert: true}** – will insert document if it was not found

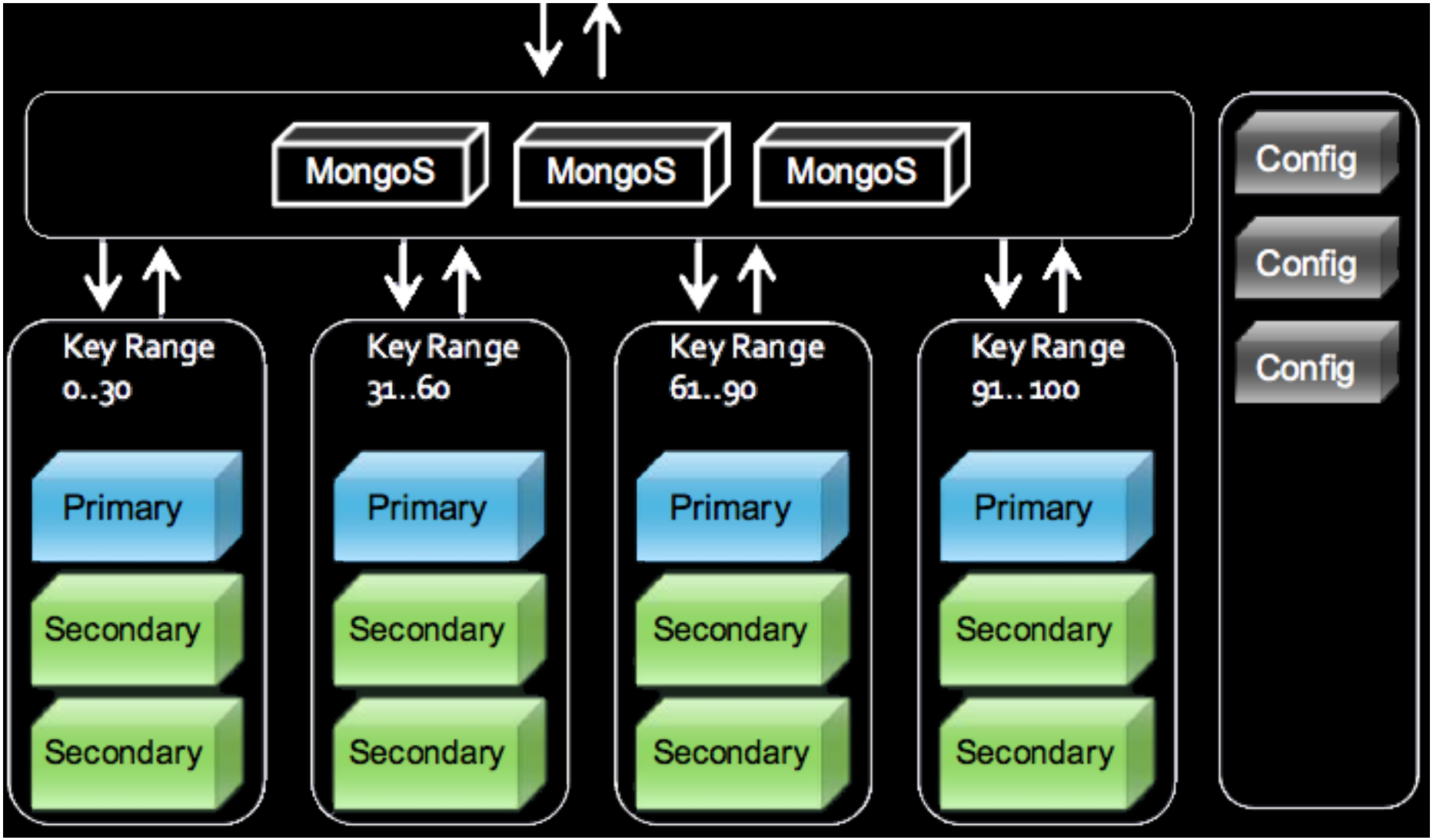
# Some MongoDB specific features

- Geo-spatial Indexes for Geo-spatial queries.
  $near, $within_distance, Bound queries (circle, box)
- GridFS
  Stores Large Binary Files.
- Map/Reduce
  GROUP BY in SQL, map/reduce in MongoDB.

# Replica set



# Sharding


MongoS
MongoS
MongoS
Config
Config
Config
Key Range
0..30
Key Range
31..60
Key Range
61..90
Key Range
91..100
Primary
Secondary
Secondary
Primary
Secondary
Secondary
Primary
Secondary
Secondary
Primary
Secondary
Secondary

# Task 4

## Use mongodb to store notes