mongoDB

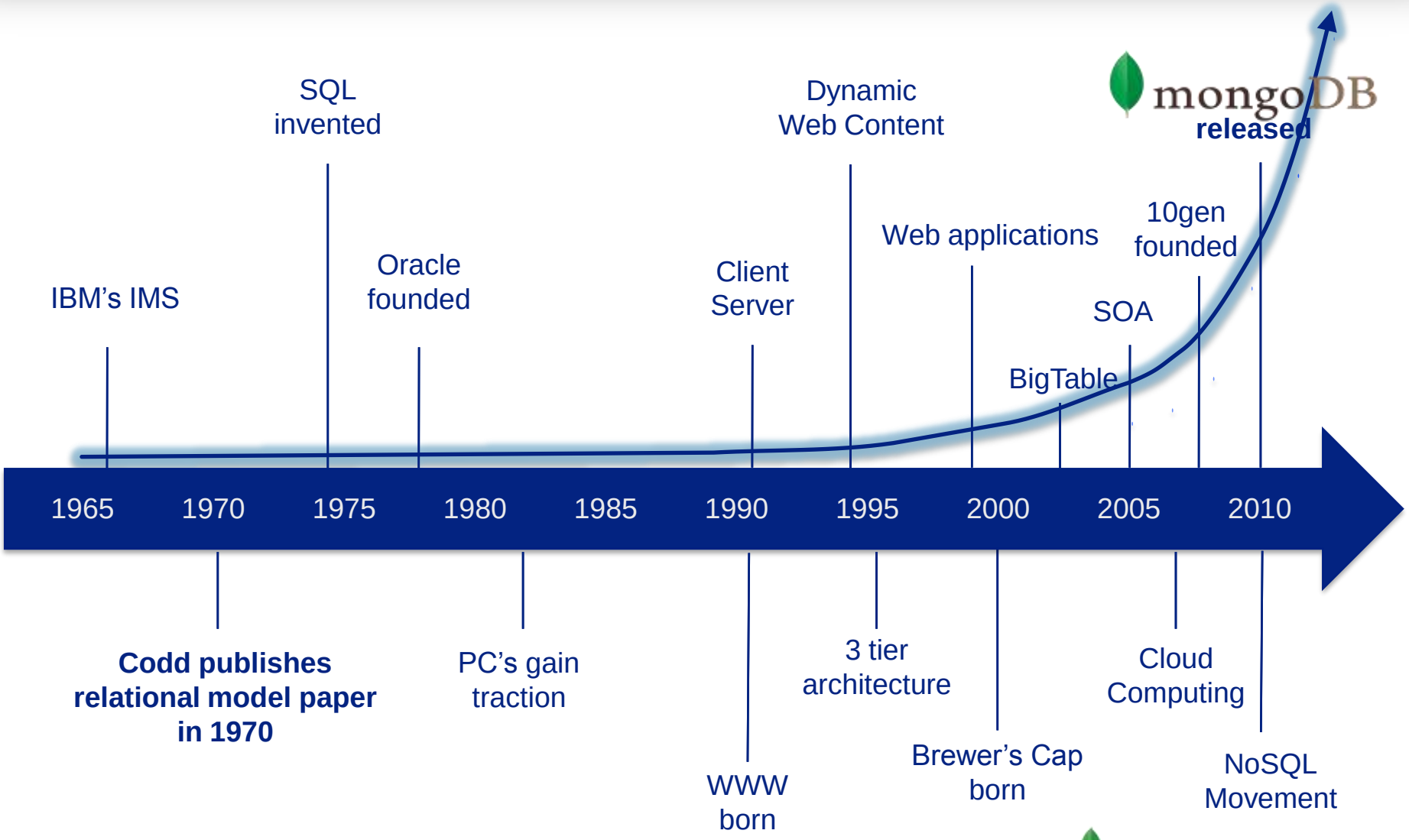
Open source, high performance database

Introduction to NoSQL and MongoDB

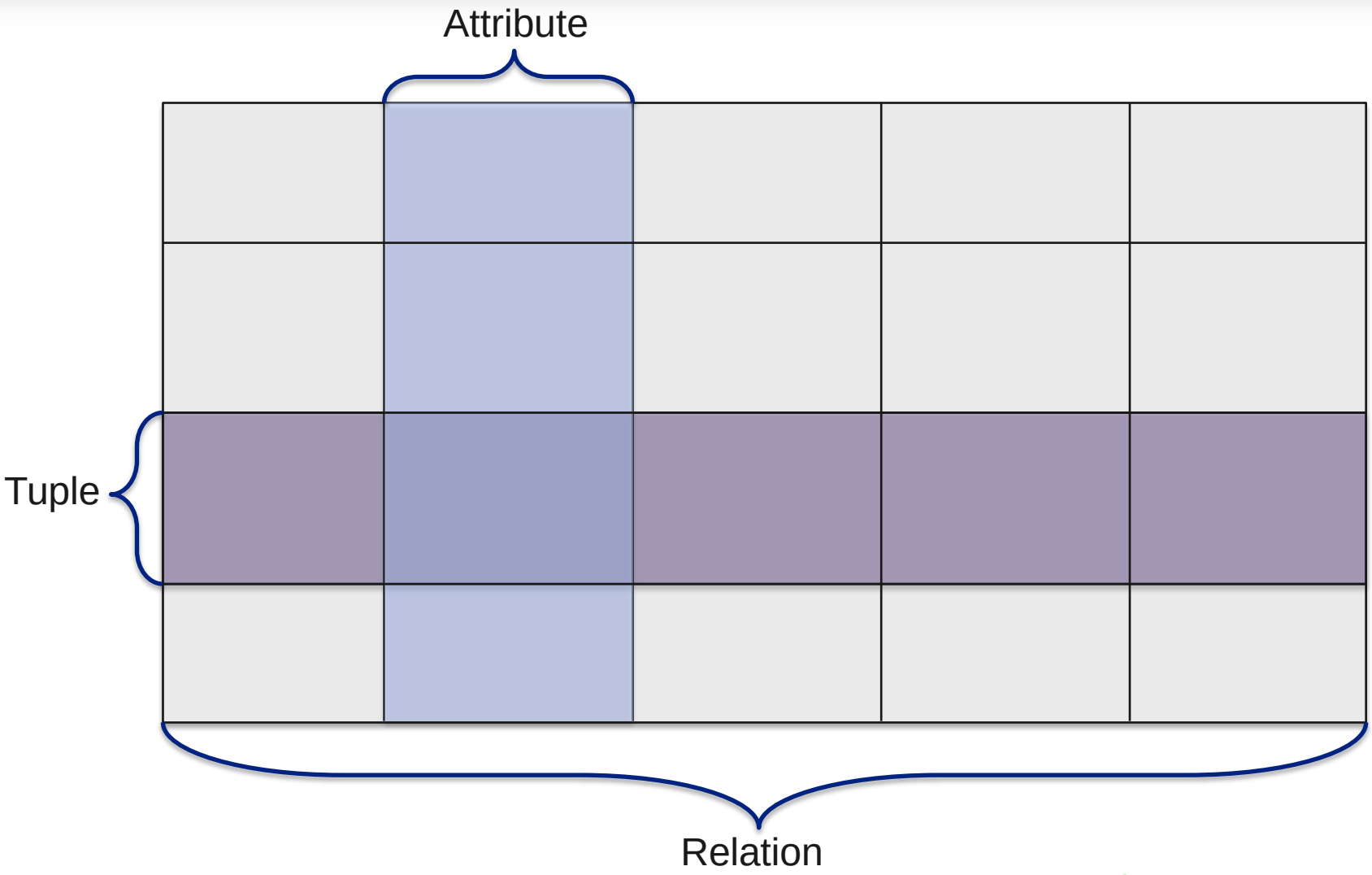
Will LaForest
Senior Director of 10gen Federal
will@10gen.com
@WLaForest

10gen | the MongoDB company

Dawn of Databases to Present

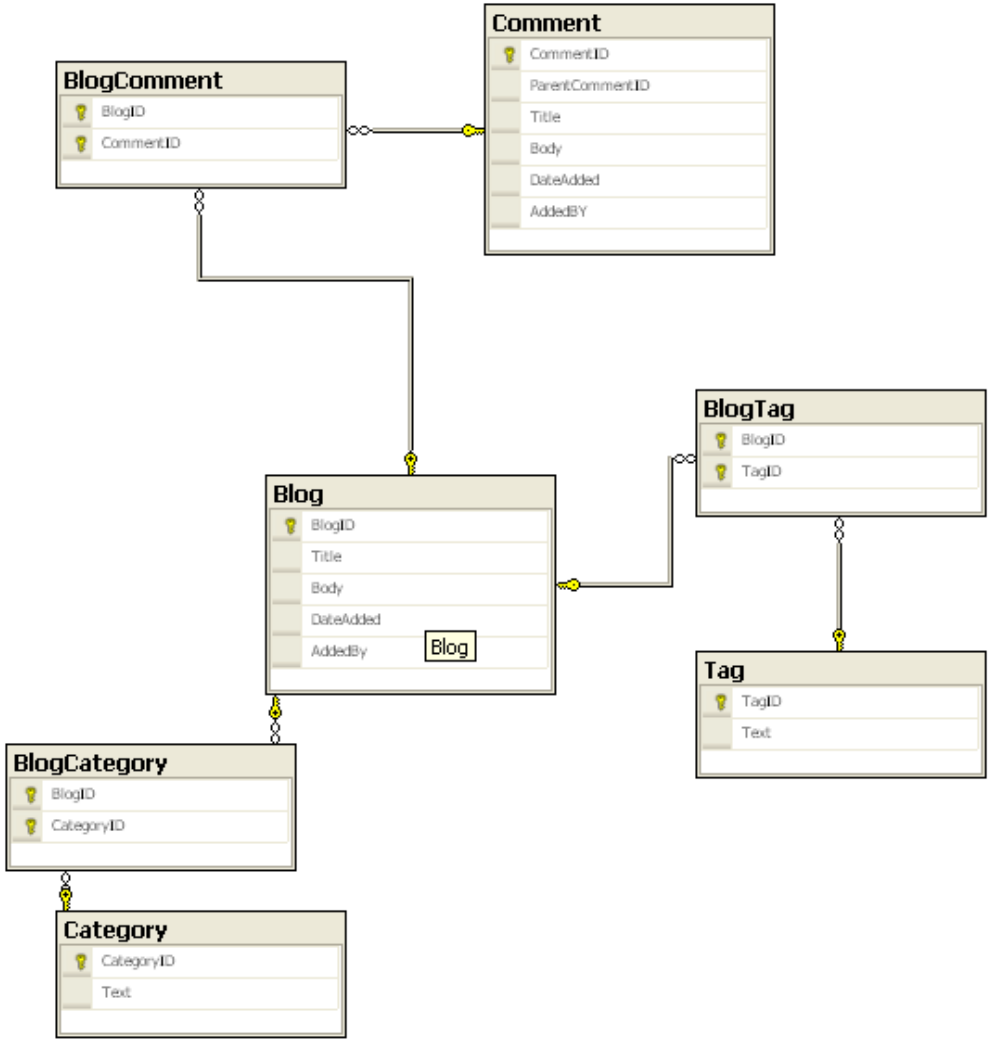


Relational Databases



mongoDB

Relational Databases



Relational Database Strengths

- Data stored in a RDBMS is very compact (disk was more expensive)
- SQL and RDBMS made queries flexible with rigid schemas
- Rigid schemas helps optimize joins and storage
- Massive ecosystem of tools, libraries and integrations
- Been around 40 years!

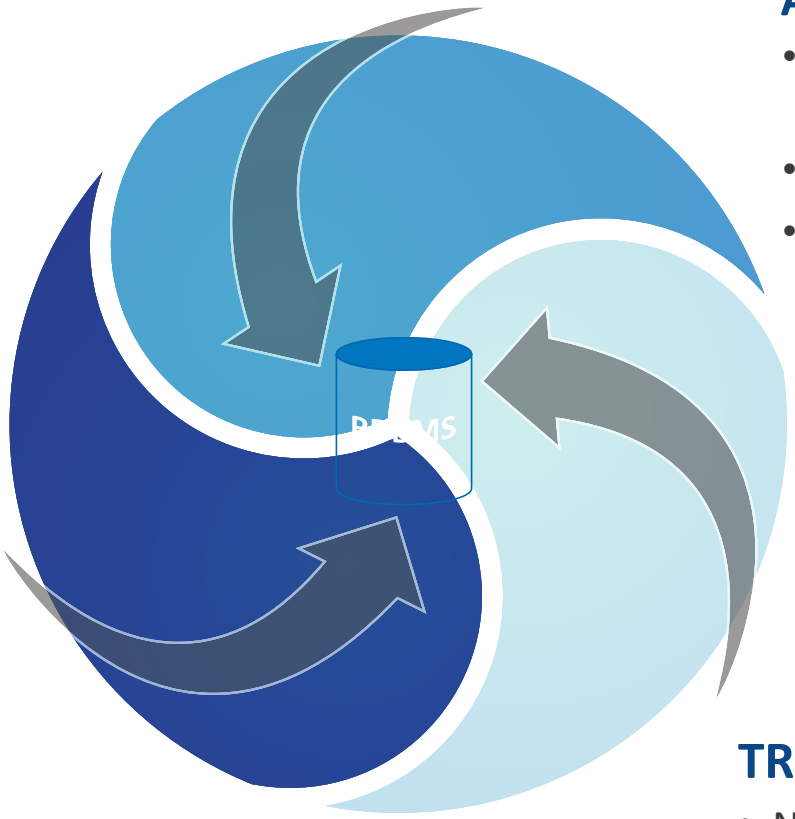
Enter Big Data

- Gartner uses the 3Vs to define
- Volume - Big/difficult/extreme volume is relative
- Variety
  - Changing or evolving data
  - Uncontrolled formats
  - Does not easily adhere to a single schema
  - Unknown at design time
- Velocity
  - High or volatile inbound data
  - High query and read operations
  - Low latency

RDBMS Challenges

VOLUME & NEW ARCHITECTURES

- Systems scaling horizontally, not vertically
- Commodity servers
- Cloud Computing



DATA VARIETY & VOLATILITY

- Extremely difficult to find a single fixed schema
- Don't know data schema a-priori

TRANSACTIONAL MODEL

- N x Inserts or updates
- Distributed transactions

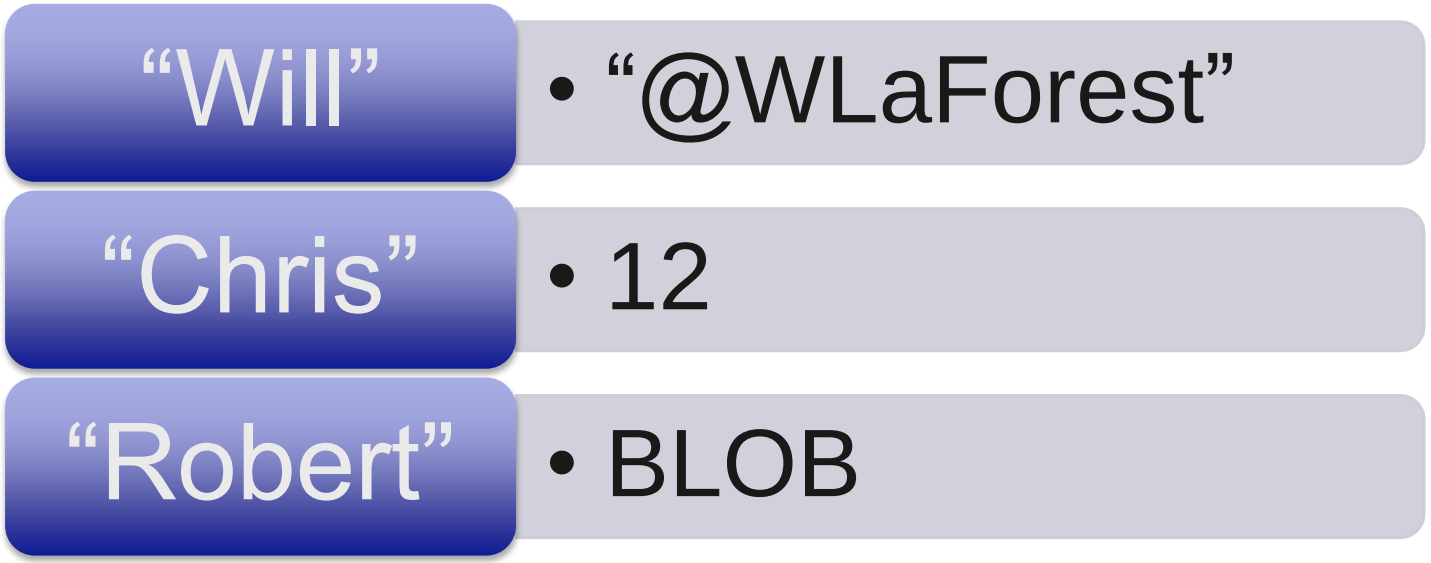
mongoDB

Enter NoSQL

- Non-relational been hanging around (MUMPS?)
- Modern NoSQL theory and offerings started in early 2000s
- Modern usage of term introduced in 2009
- NoSQL = Not Only SQL
- A collection of very different products
- Alternatives to relational databases when they are a bad fit
- Motives
  - Horizontally scalable (commodity server/cloud computing)
  - Flexibility

NoSQL Databases – Key-Value

- Value (Data) mapped to a key (think primary)
- Some typed some just BLOBs
- Redis, MemcachDB, Voldemort



mongoDB

NoSQL Databases – Big Table Descendents

- Data stored on disk in a column oriented fashion
- Predominantly hash based indexing
- Data partitioned by range or consistent hashing
- Google BigTable, HBase, Cassandra, Accumulo

mongoDB

NoSQL Databases – Key-Value

- Key-Value Stores
  - Key-Value Stores
  - Value (Data) mapped to a key (think primary)
- Big Table Descendants
  - Looks like a distributed multi-dimension map
  - Data stored in a column oriented fashion
  - Predominantly hash based indexing
- Document oriented stores
  - Data stored as either JSON or XML documents
- What do they all have in common?

Hadoop

- Not a database
- Map reduce on HDFS or other data source
- When you want to use it
  - Can't use a index
  - Distributing custom algorithms
  - ETL
- Great for grinding through data
- Many NoSQL offerings have native map reduce functionality

MongoDB & 10gen

10gen & MongoDB

- 2007
  - Eliot Horowitz & Dwight Merriman tired of reinventing the wheel
  - 10gen founded
  - MongoDB Development begins
- 2009
  - Initial release of MongoDB
- 73M+ in funding
  - Funded by Sequoia, NEA, Union Square Ventures, Flybridge Capital

10gen Products

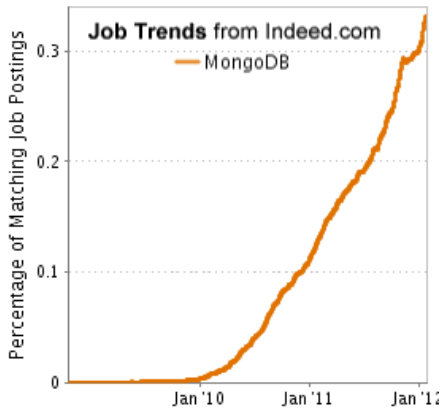
- Pre-production Subscriptions for MongoDB
  - Fixed cost = $30k
  - 6 month term, unlimited servers
- MongoDB Subscriptions
  - 32 month term, $4k per server
  - 24x7 support for development and production, 1 hour response time
  - Onboarding call and quarterly review
- Consulting
  - Cost is $300/hr
- Training
  - MongoDB Developer - $1500/student, 2 day course
  - MongoDB Administrator - $1500/student, 2 day course
  - MongoDB Essential - $2250/student, 3 day course
- MongoDB Monitoring Service

Cost Comparison

- 4 servers with 2 processors each
- Total processors equals 8
- Oracle Enterprise Edition
  - $47,500 per processor plus maintenance
  - 8 x 47,500 = $380,000 + $76,000
  - Total Cost = $456,000
- MongoDB Subscription
  - $4,000 per server (no processor count)
  - No license fee or maintenance charge
  - 4 x 4,000 = $16,000
  - Total Cost = $16,000
- MongoDB is $440,000 less expensive

MongoDB is the leading NoSQL solution

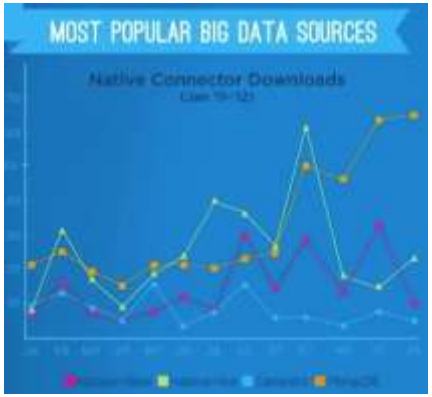
#2 on Indeed's Fastest Growing Jobs



Top Job Trends

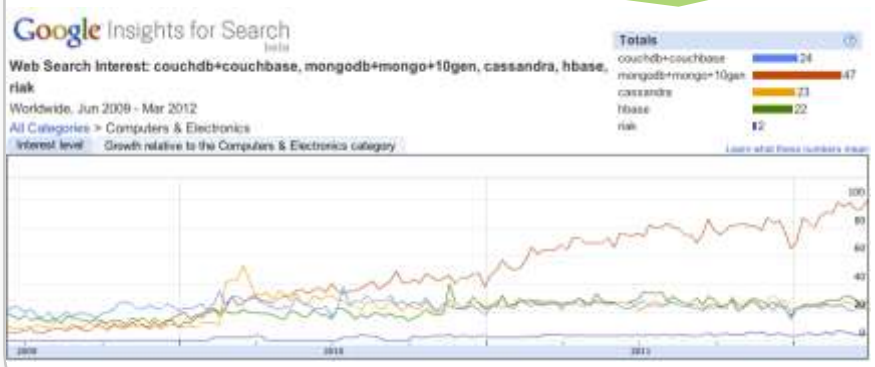
1. HTML5
2. **MongoDB**
3. iOS
4. Android
5. Mobile app
6. Puppet
7. Hadoop
8. jQuery
9. PaaS
10. Social Media

Jaspersoft BigData Index

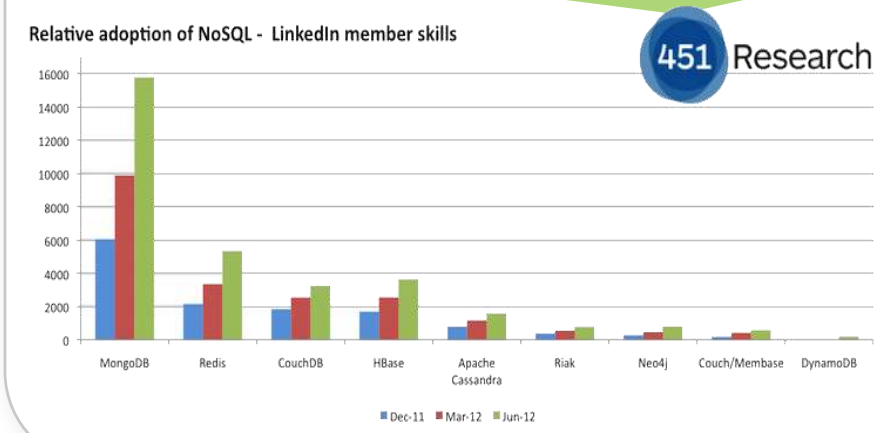


Demand for MongoDB, the document-oriented NoSQL database, saw the biggest spike with over 200% growth in 2011.

Google Searches

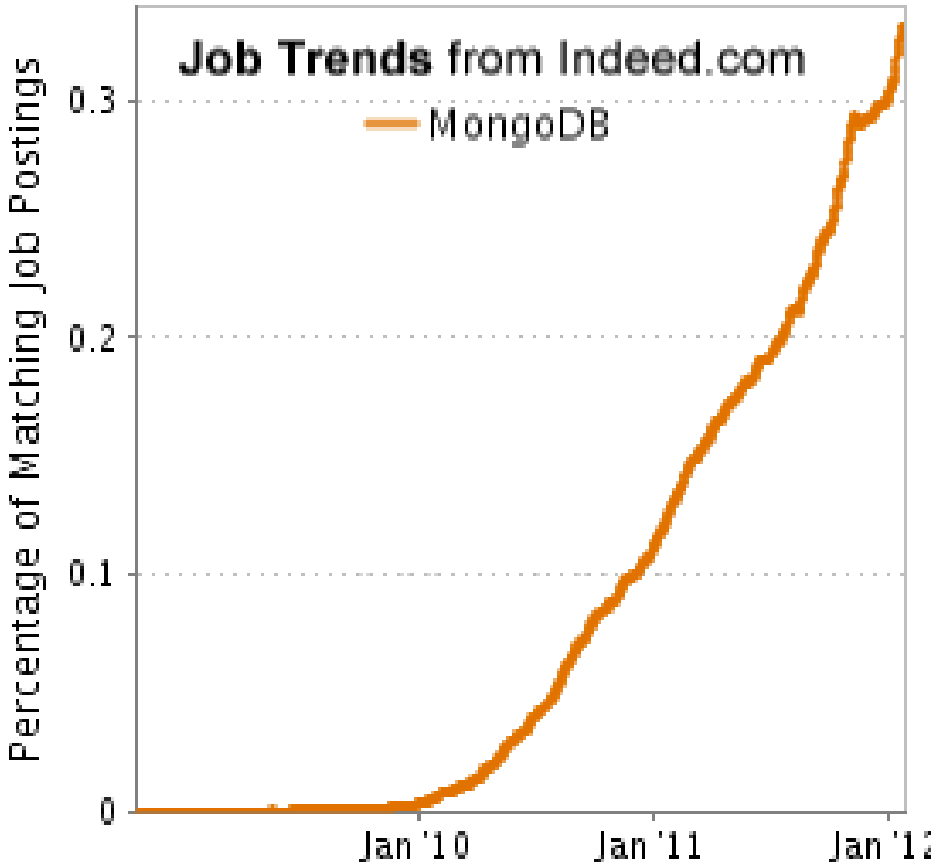


451 Group "MongoDB increasing its dominance"



MongoDB is the leading NoSQL solution

#2 ON INDEED'S FASTEST GROWING JOBS

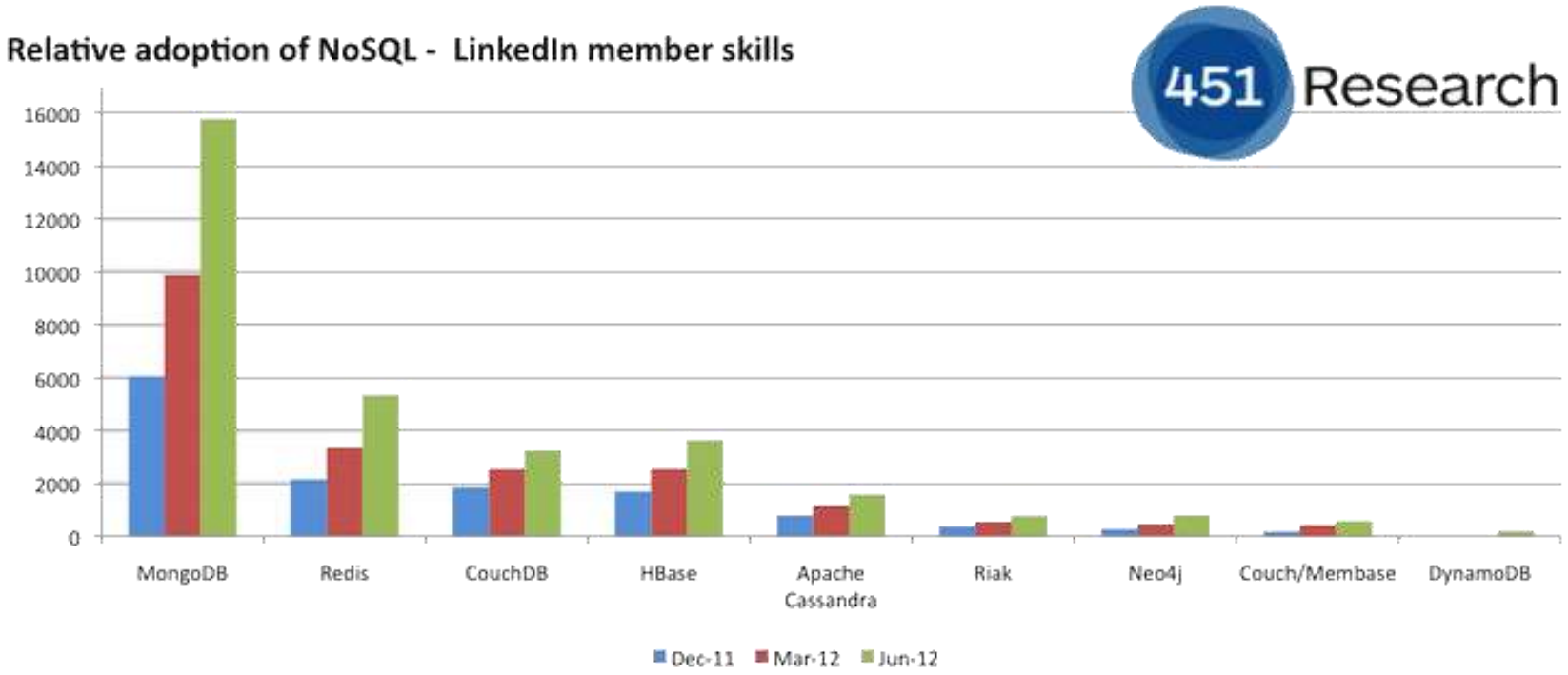


Top Job Trends

1. HTML5
2. **MongoDB**
3. iOS
4. Android
5. Mobile app
6. Puppet
7. Hadoop
8. jQuery
9. PaaS
10. Social Media

MongoDB is the leading NoSQL solution.

“MongoDB INCREASING ITS DOMINANCE”



Different Assumptions

- Scale horizontally over commodity hardware
- RDBMSs great so keep what works
  - Rich data models
  - Adhoc queries
  - Fully featured indexes
- What doesn't distribute well?
  - Long running multi-row transactions
  - Joins
  - Both artifacts of the relational data model
- Do not homogenize programming interfaces
- Local storage first class citizen for DB storage

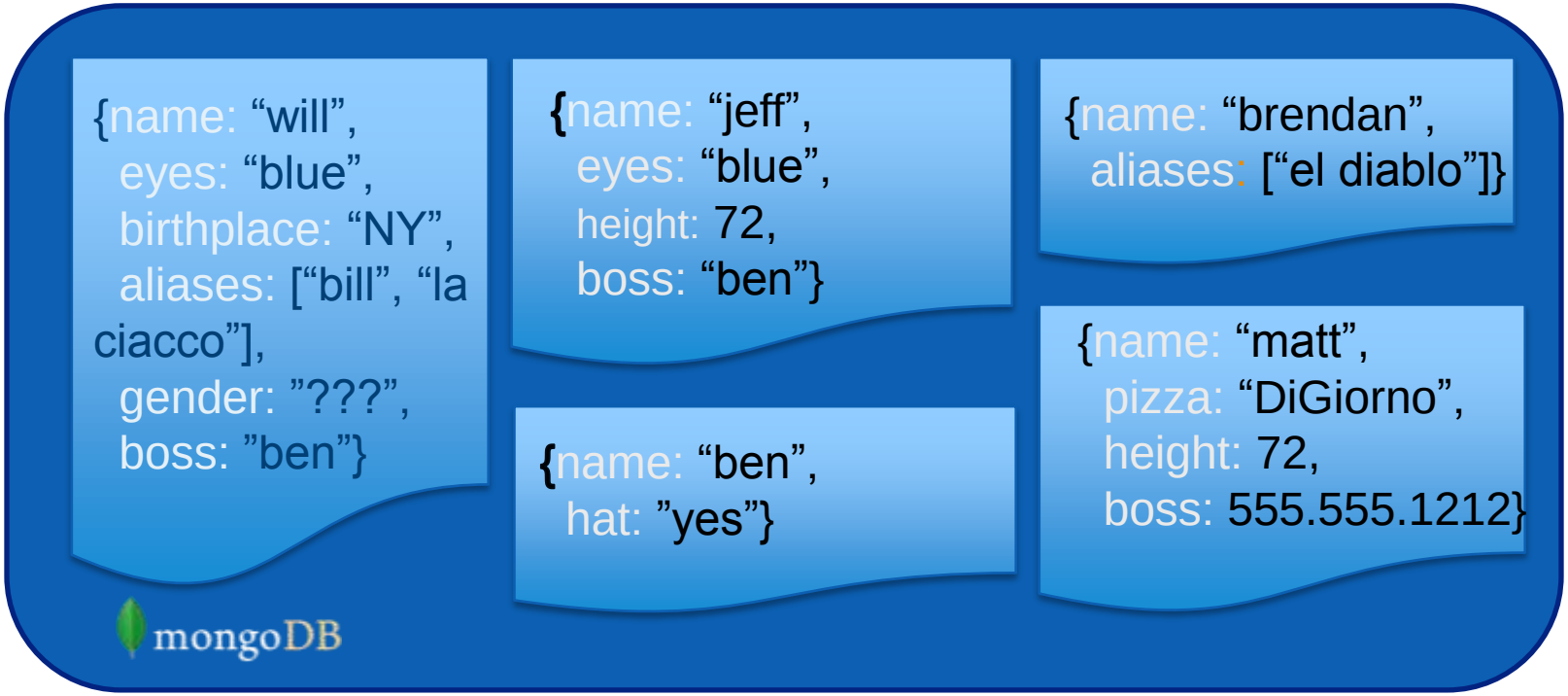
MongoDB Key Features

- Data stored as documents (JSON)
- Schema free
- CRUD operations – (Create Read Update Delete)
- Atomic document operations
- Ad hoc Queries like SQL
  - Equality
  - Regular expression searches
  - Ranges
  - Geospatial
- Secondary indexes
- Sharding (sometimes called partitioning) for scalability
- Replication – HA and read scalability

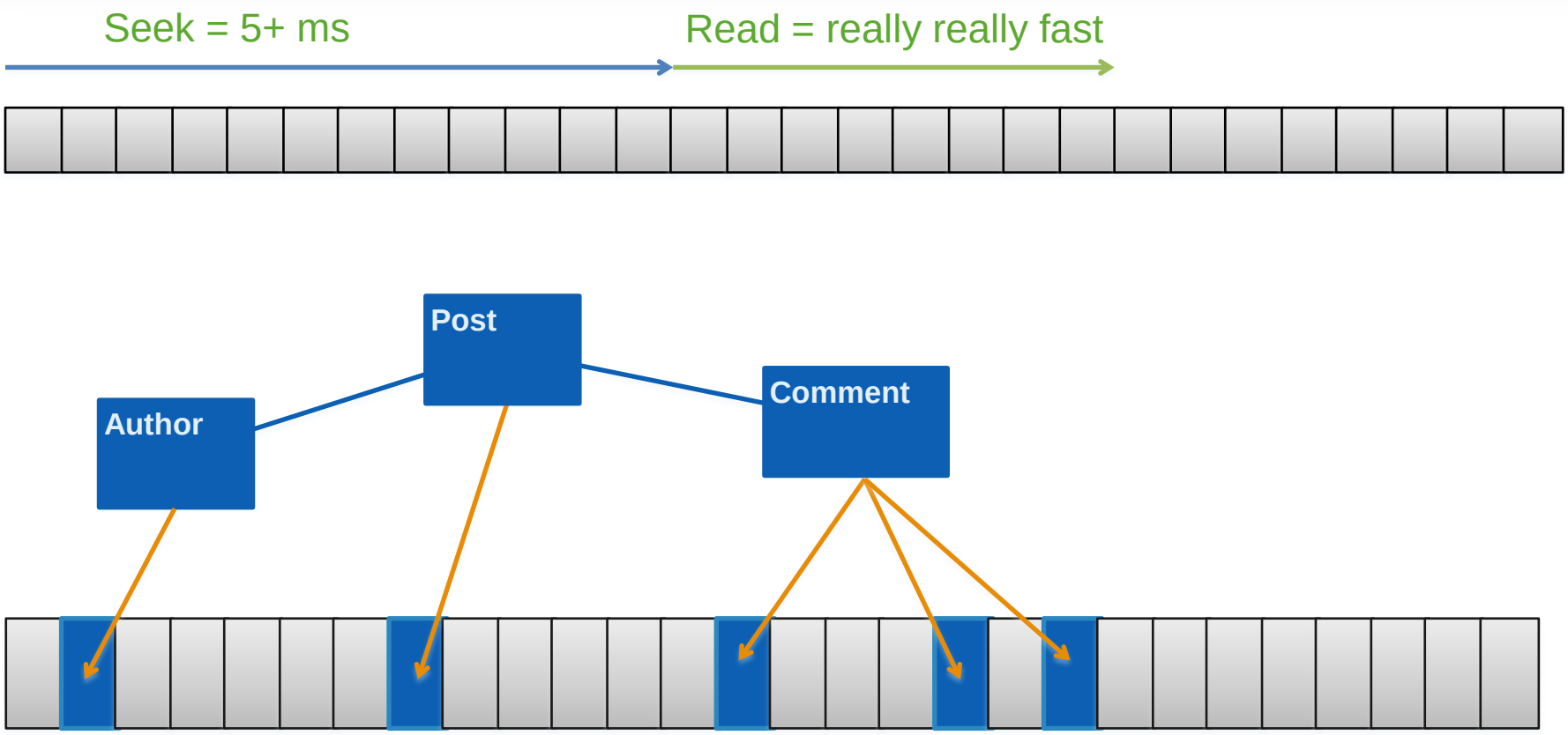
mongoDB

Document Oriented and Schema Free

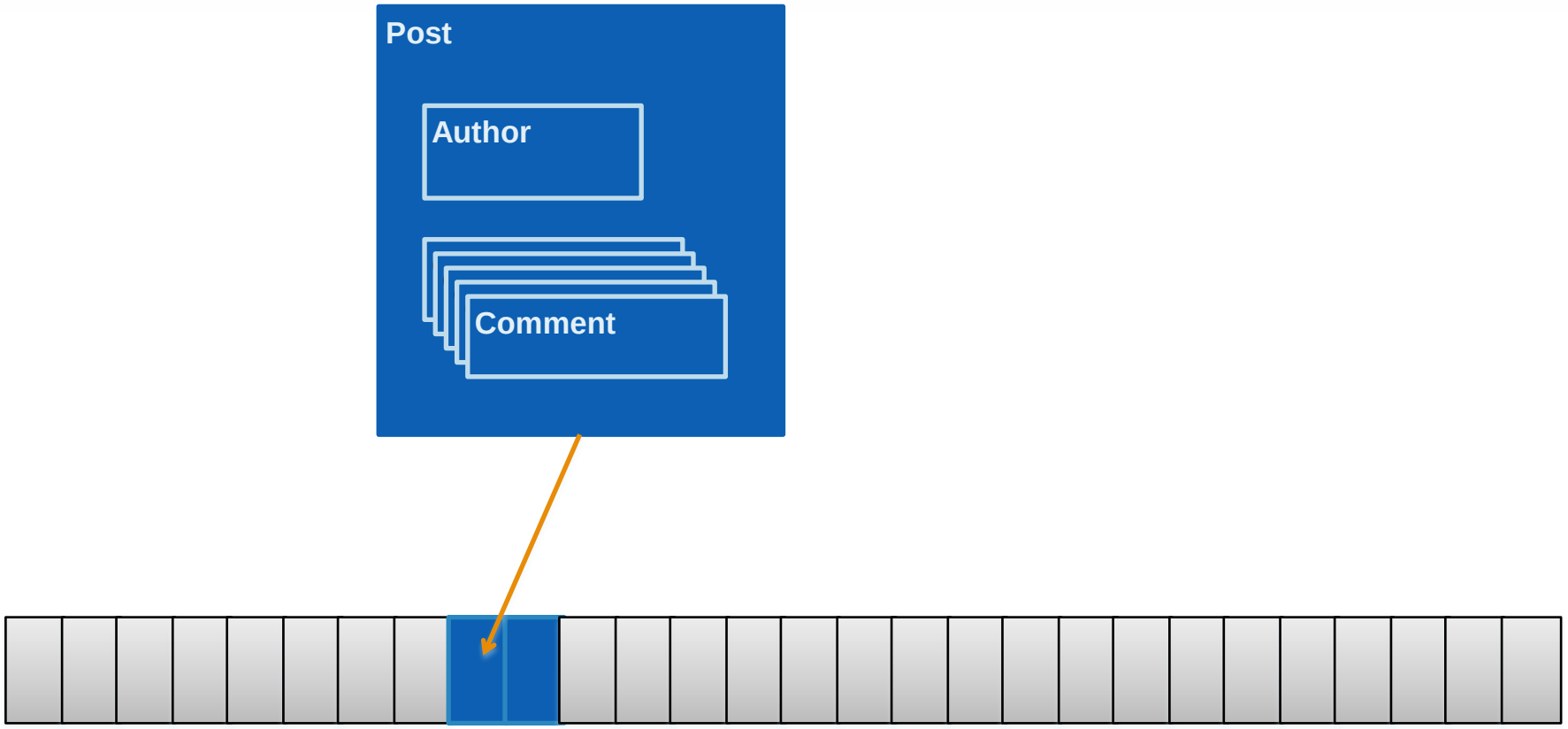
- MongoDB does not need any defined data schema.
- Every document could have different data!



Disk seeks and data locality



Disk seeks and data locality



Terminology

RDBMS	MongoDB
Database	Database
Table	Collection
Row	Document

MongoDB scale

- We are running instances on the order of:
  - 100B objects
  - 50TB storage
  - 50K qps per server
  - ~1200 servers

MongoDB Security

- SSL
  - between client and server
  - Intra-cluster communication
- Authorization at the database level
  - Read Only/Read+Write/Administrator
- Security Roadmap (tentative)
  - Pluggable authentication 2.4
  - Auditing 2.4
  - Cell level security 2.6
  - Common Criteria certification

Usage Examples

Documents

```
var p = { author: “roger”,
        date: new Date(),
        text: “Spirited Away”,
        tags: [“Tezuka”, “Manga”]}

> db.posts.save(p)
```

Querying

```
>db.posts.find()

  { _id : ObjectId("4c4ba5c0672c685e5e8aabf3"),
    author : "roger",
    date : "Sat Jul 24 2010 19:47:11 GMT-0700 (PDT)",
    text : "Spirited Away",
    tags : [ "Tezuka", "Manga" ] }
```

Notes:

- _id is unique, but can be anything you'd like

Secondary Indexes

Create index on any Field in Document

```
//   1 means ascending, -1 means descending

>db.posts.ensureIndex({author: 1})

>db.posts.find({author: 'roger'})

{ _id       : ObjectId("4c4ba5c0672c685e5e8aabf3"),
  author : "roger",
  ... }
```

Query Operations

- Conditional Operators
  - $all, $exists, $mod, $ne, $in, $nin, $nor, $or, $size, $type
  - $lt, $lte, $gt, $gte

```
// find posts with any tags
> db.posts.find( {tags: {$exists: true }} )

// find posts matching a regular expression
> db.posts.find( {author: /^rog*/i } )

// count posts by author
> db.posts.find( {author: 'roger'} ).count()
```

Atomic Operations

- $set, $unset, $inc, $push, $pushAll, $pull, $pullAll, $bit

```
> comment = { author: “fred”,
                date: new Date(),
                text: “Best Movie Ever”}

 > db.posts.update( { _id: “...” },
                    $push: {comments: comment} );
```

mongoDB

Nested Documents

```
{  _id : ObjectId("4c4ba5c0672c685e5e8aabf3"),
   author : "roger",
   date : "Sat Jul 24 2010 19:47:11 GMT-0700 (PDT)",
   text : "Spirited Away",
   tags : [ "Tezuka", "Manga" ],
   comments : [
    {
        author : "Fred",
        date : "Sat Jul 24 2010 20:51:03 GMT-0700 (PDT)",
        text : "Best Movie Ever"
    }
   ]}
```

Secondary Indexes

```
// Index nested documents
> db.posts.ensureIndex( “comments.author”:1 )
  ➢ db.posts.find({‘comments.author’:’Fred’})

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

// geospatial index
> db.posts.ensureIndex( “author.location”: “2d” )
> db.posts.find( “author.location” : { $near : [22,42] } )
```

Aggregates, Statistics, and Analytics

Computations & Aggregates Over MongoDB

- Native Map/Reduce in JS in MongoDB
  - Distributes across the cluster with good data locality
- New aggregation framework
  - Declarative (no JS required)
  - Pipeline approach (like Unix ps -ax | tee processes.txt | more)
- Hadoop
  - Intersect the indexing of MongoDB with the brute force parallelization of hadoop
  - Hadoop MongoDB connector

Native MapReduce

```
// Our key is author's userame;
// our value, the number of votes for the current comment.
var map = function() {
  emit(this.author, {votes: this.votes});
};
```

```
reduce('kbanker', [{votes: 2}, {votes: 1}, {votes: 4}]);
```

```
// Add up all the votes for each key.
var reduce = function(key, values) {
  var sum = 0;
  values.forEach(function(doc) {
    sum += doc.votes;
  });
  return {votes: sum};
};
```

mongoDB

Aggregation Framework

$project	$match	$limit	$skip
$unwind	$group	$sort	

```
{
 title : “this is my title” ,
 author : “bob” ,
 posted : new Date () ,
 pageViews : 5 ,
 tags : [ “fun” , “good” , “fun” ] ,
 comments : [
      { author :“joe” , text : “this is cool” } ,
      { author :“sam” , text : “this is bad” }
 ],
 other : { foo : 5 }
}
```

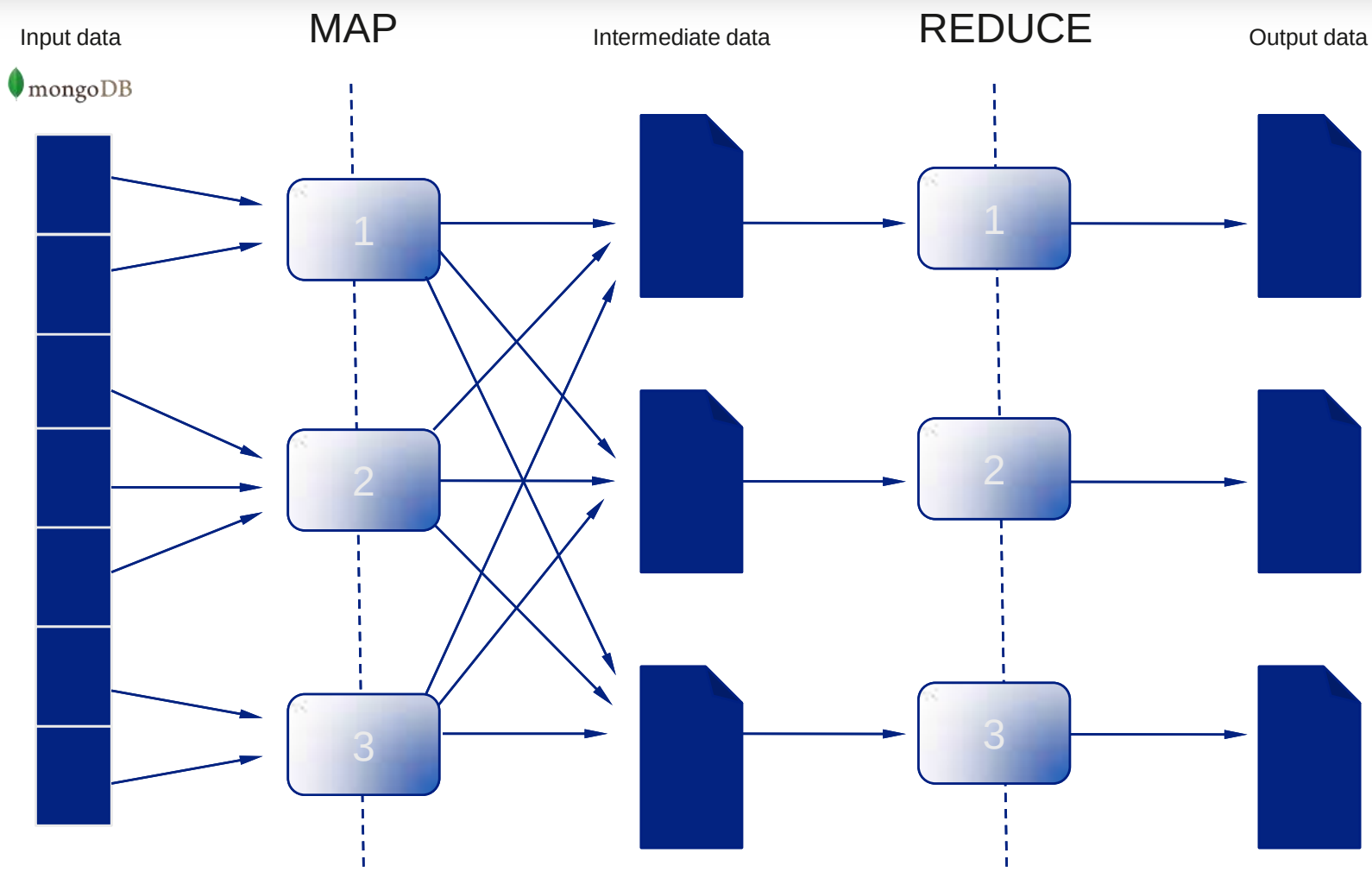
```
db.article.aggregate(
    { $project : {
        author : 1,
        tags : 1,
    }},
    { $unwind : "$tags" },
    { $group : {
        _id :  “$tags”,
        authors : { $addToSet : "$author" }
    }}
);
```

Aggregation Framework

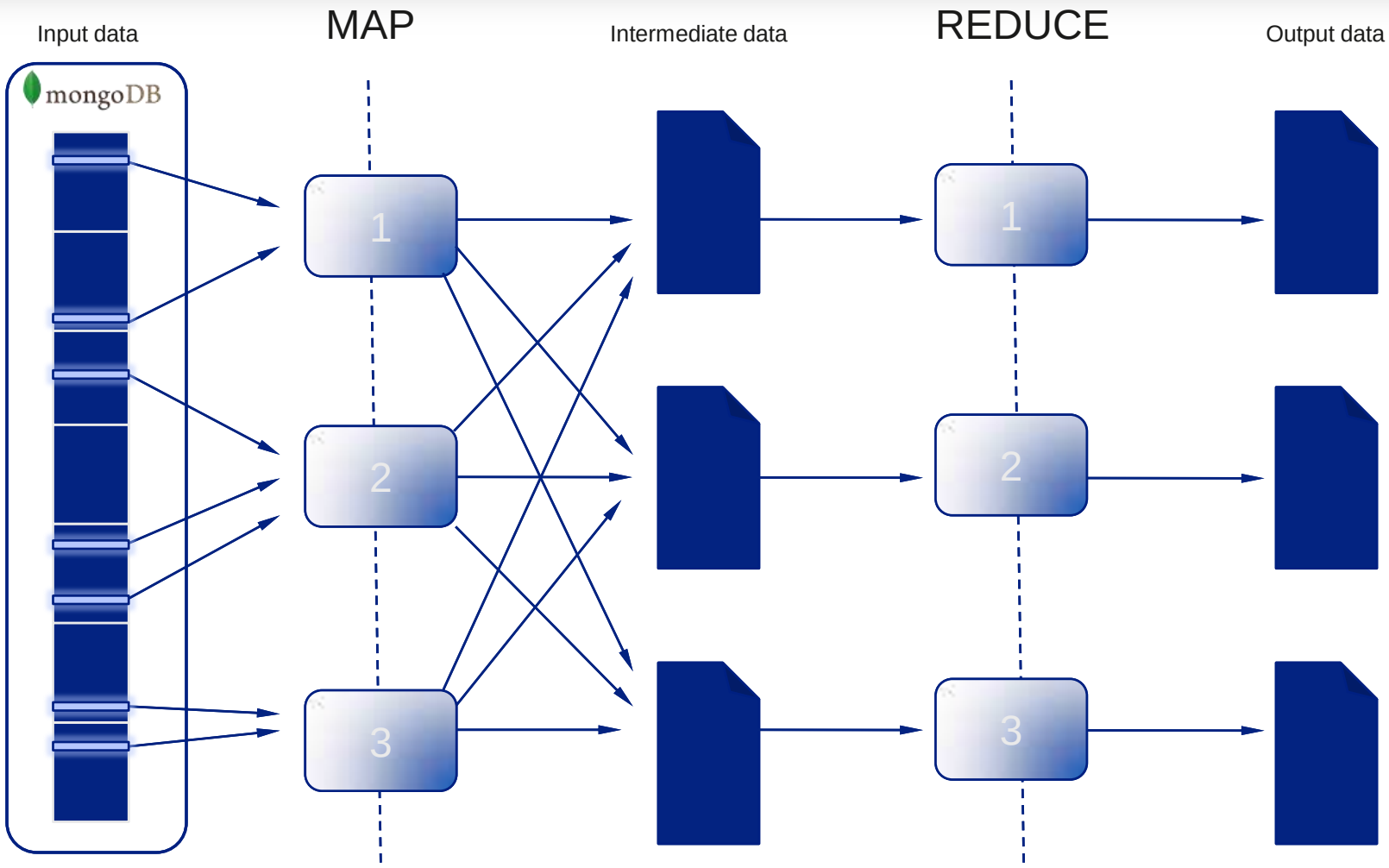
SQL Statement	Mongo Statement
`SELECT COUNT(*) FROM users`	`db.users.aggregate([ { $group: {_id:null, count:{$sum:1}} } ])`
`SELECT SUM(price) FROM orders`	`db.users.aggregate([ { $group: {_id:null, total:{$sum:"$price"} } } ])`
`SELECT cust_id,SUM(price) FROM orders GROUP BY cust_id`	`db.users.aggregate([ { $group: {_id:"$cust_id",total:{$sum:"$price"}} } ])`
`SELECT cust_id,SUM(price) FROM orders WHERE active=true GROUP BY cust_id`	`db.users.aggregate([ { $match:{active:true} }, { $group:{_id:"$cust_id",total:{$sum:"$price"}} } ])`

mongoDB

Hadoop

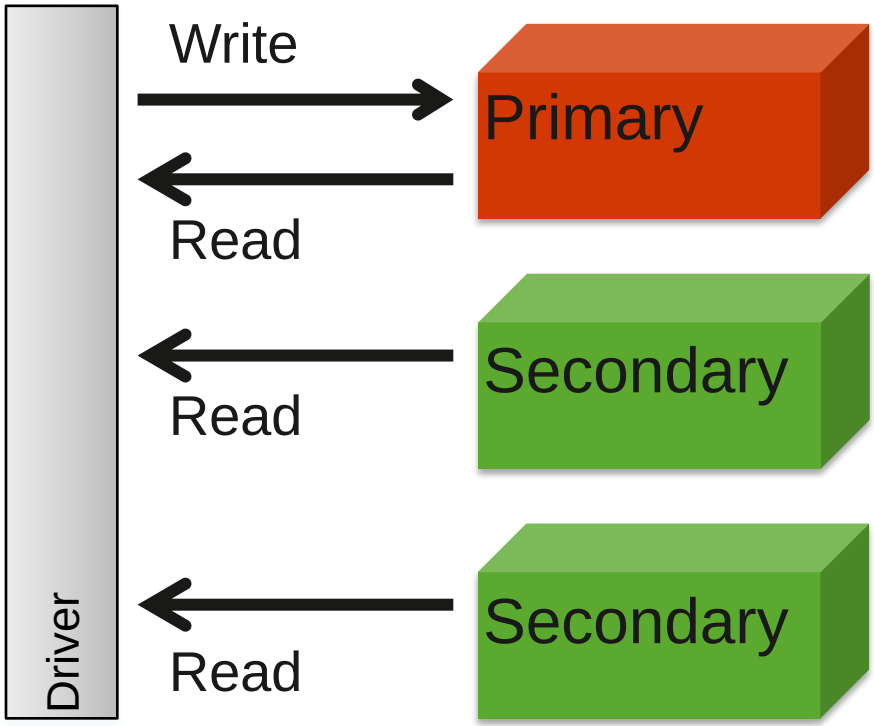

Input data
MAP
Intermediate data
REDUCE
Output data
mongoDB
1
2
3
1
2
3

Hadoop with Database



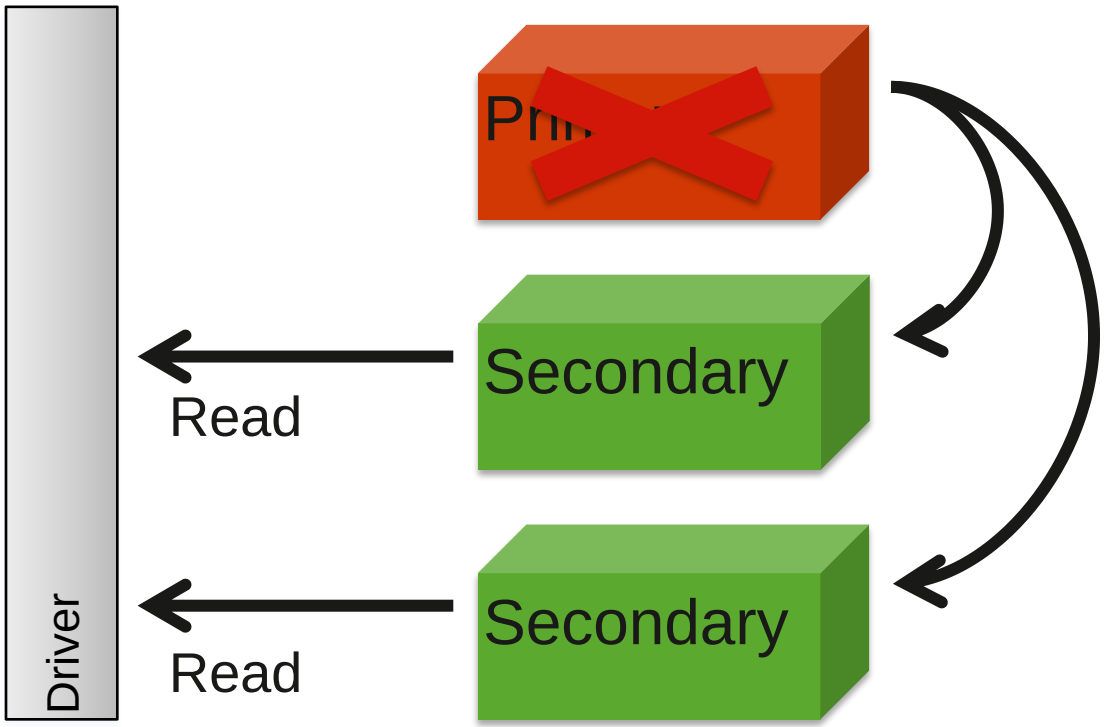
Deployment & Scaling

Replica Sets

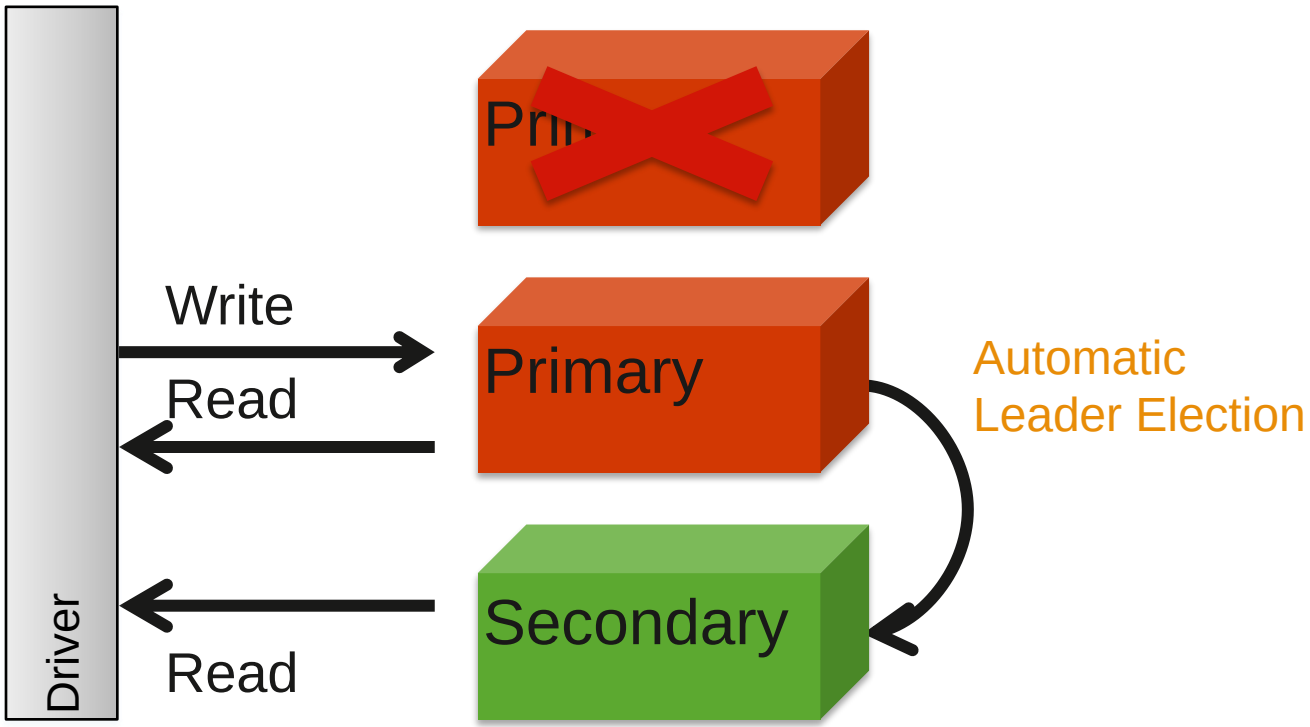


Asynchronous Replication

Replica Sets

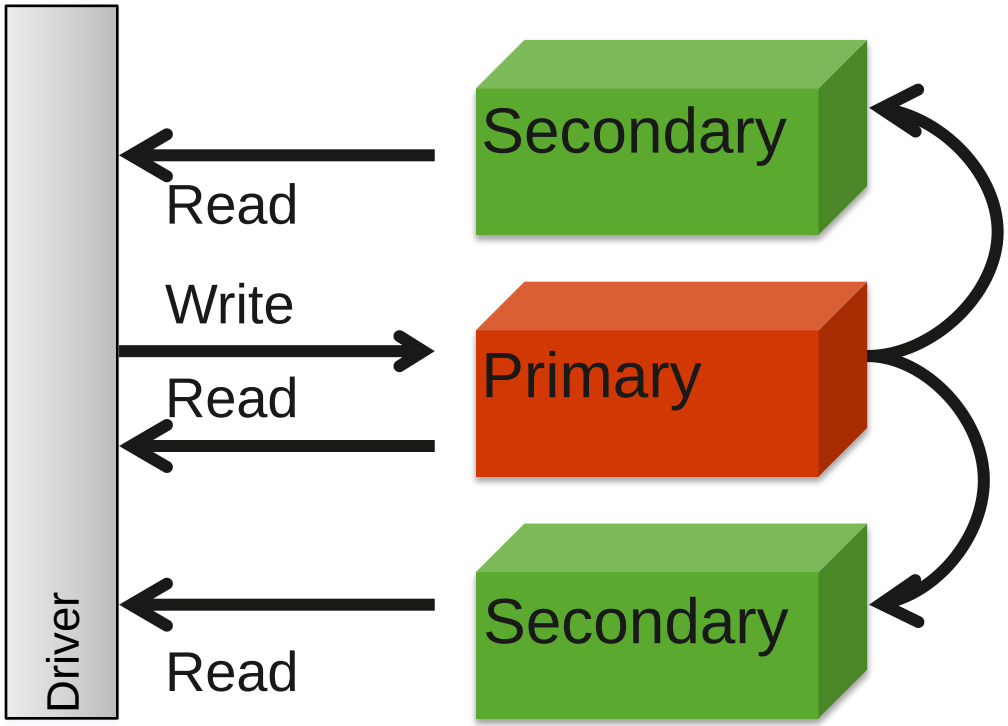


Replica Sets



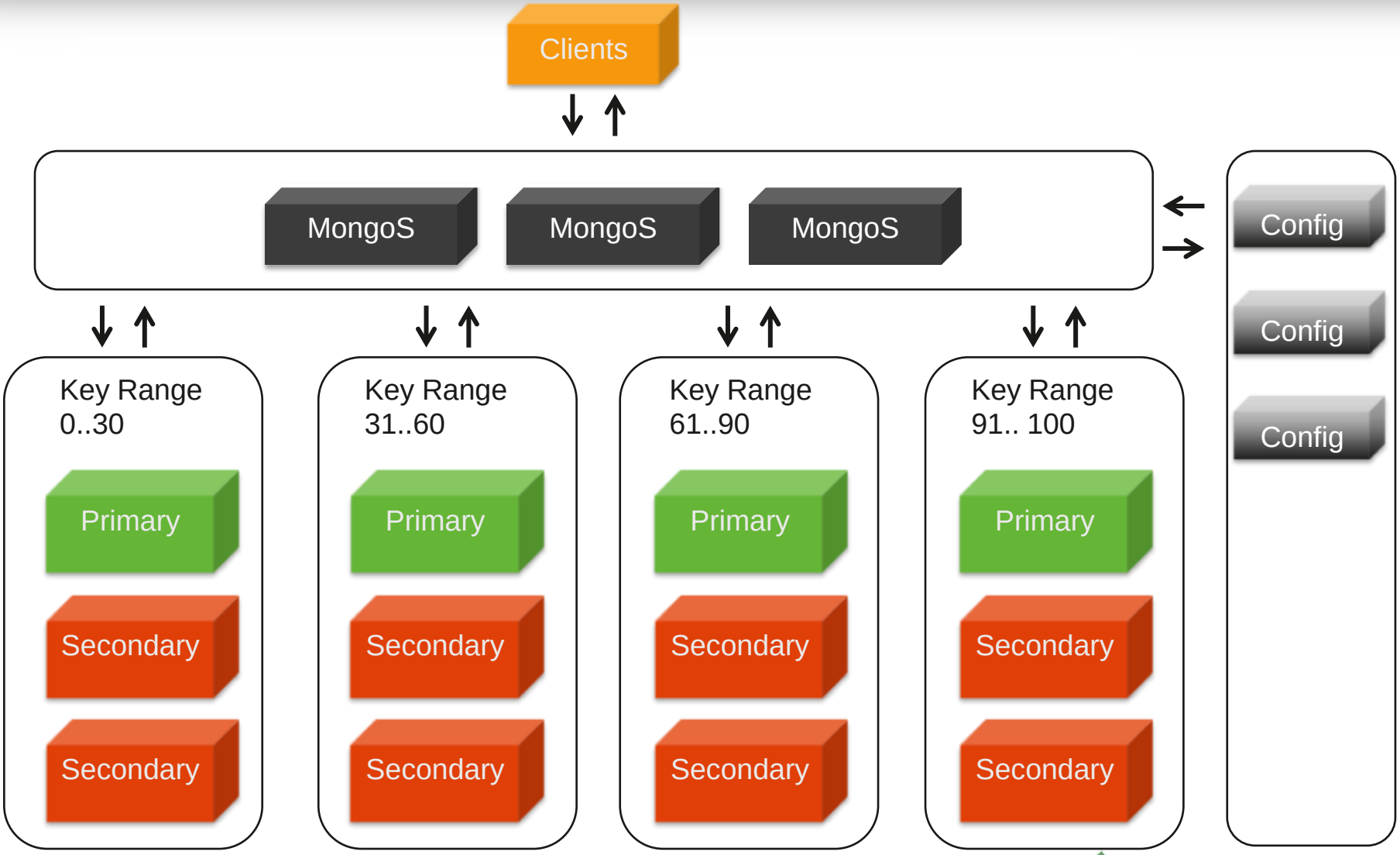
Replica Sets

Addition Details



Sharding

Additional Details



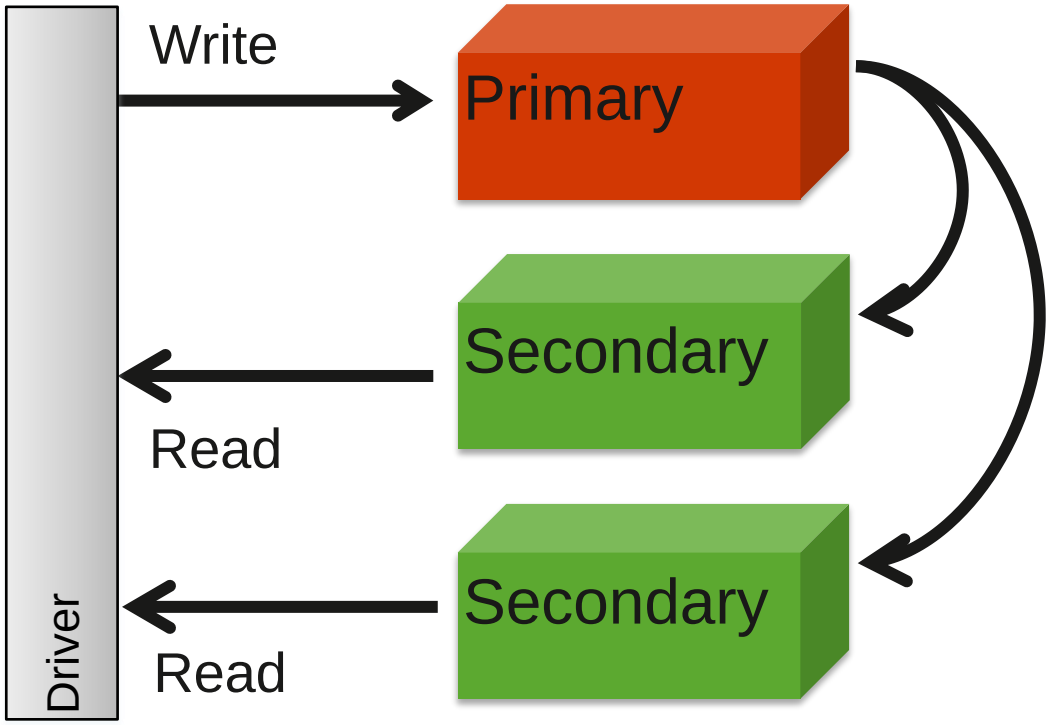
mongoDB

Additional Slides

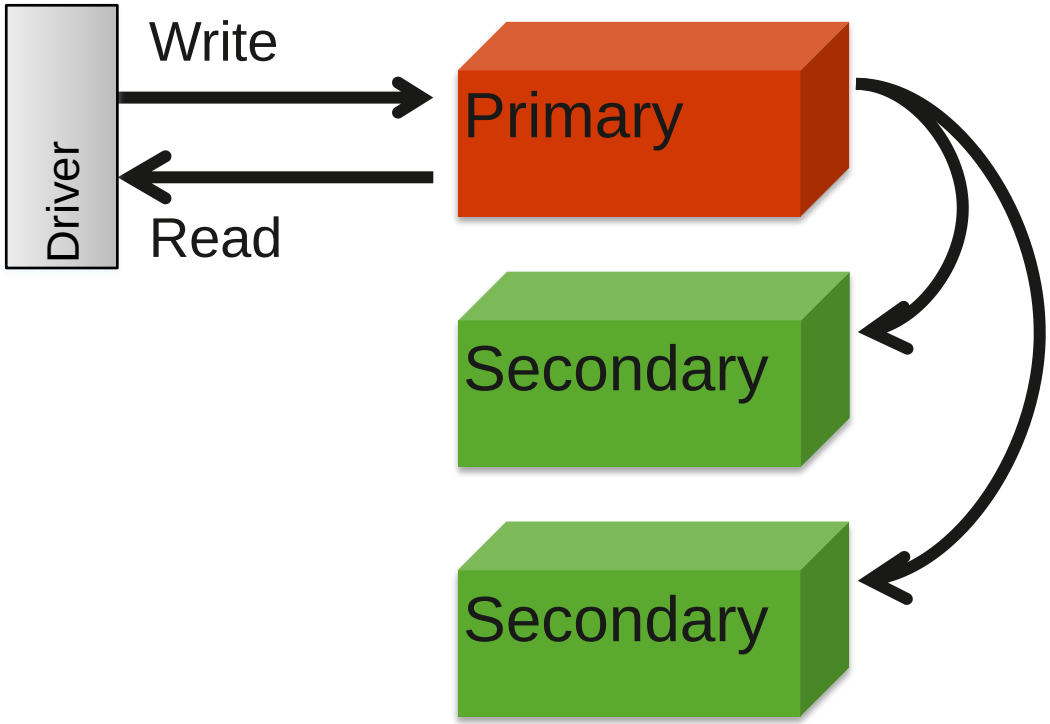
- Sharding Details
- Replica Set Details
- Consistency Details
- Common Deployment Scenarios
- Citations

Replica Set Details

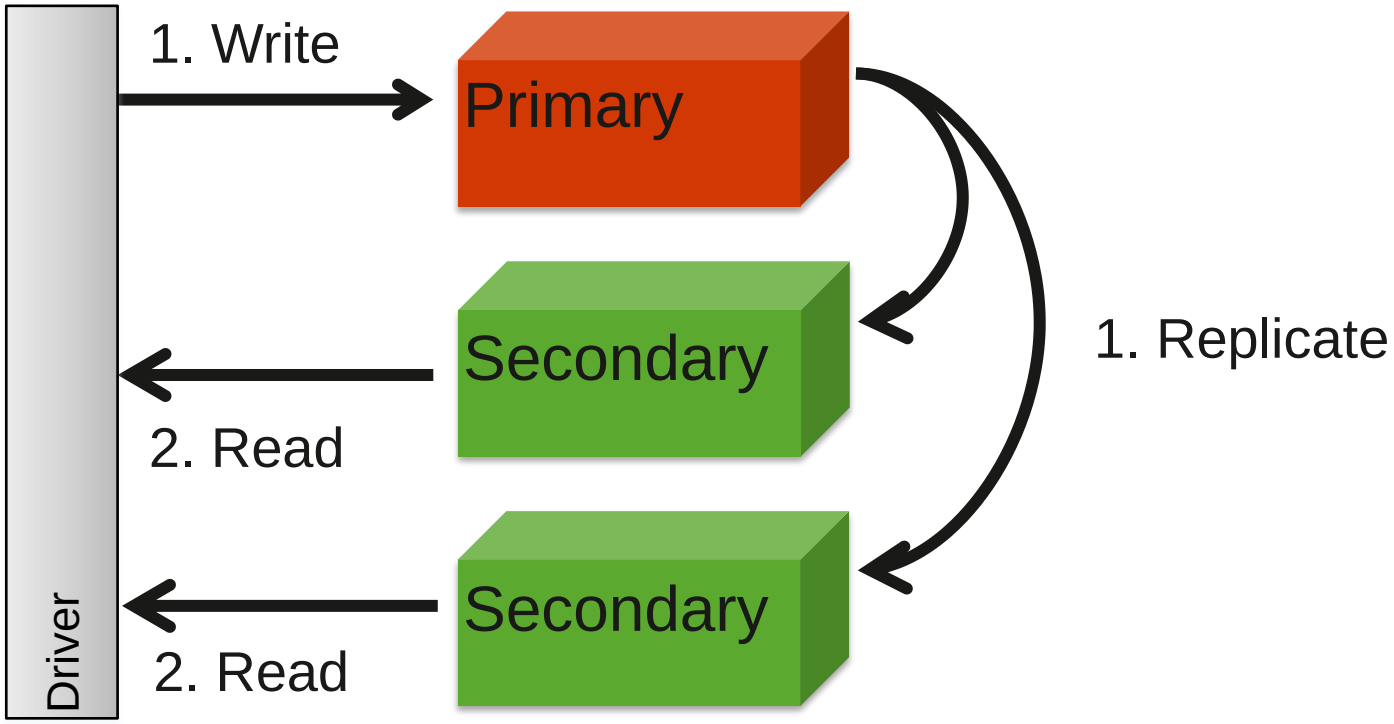
Eventual Consistency



Strong Consistency



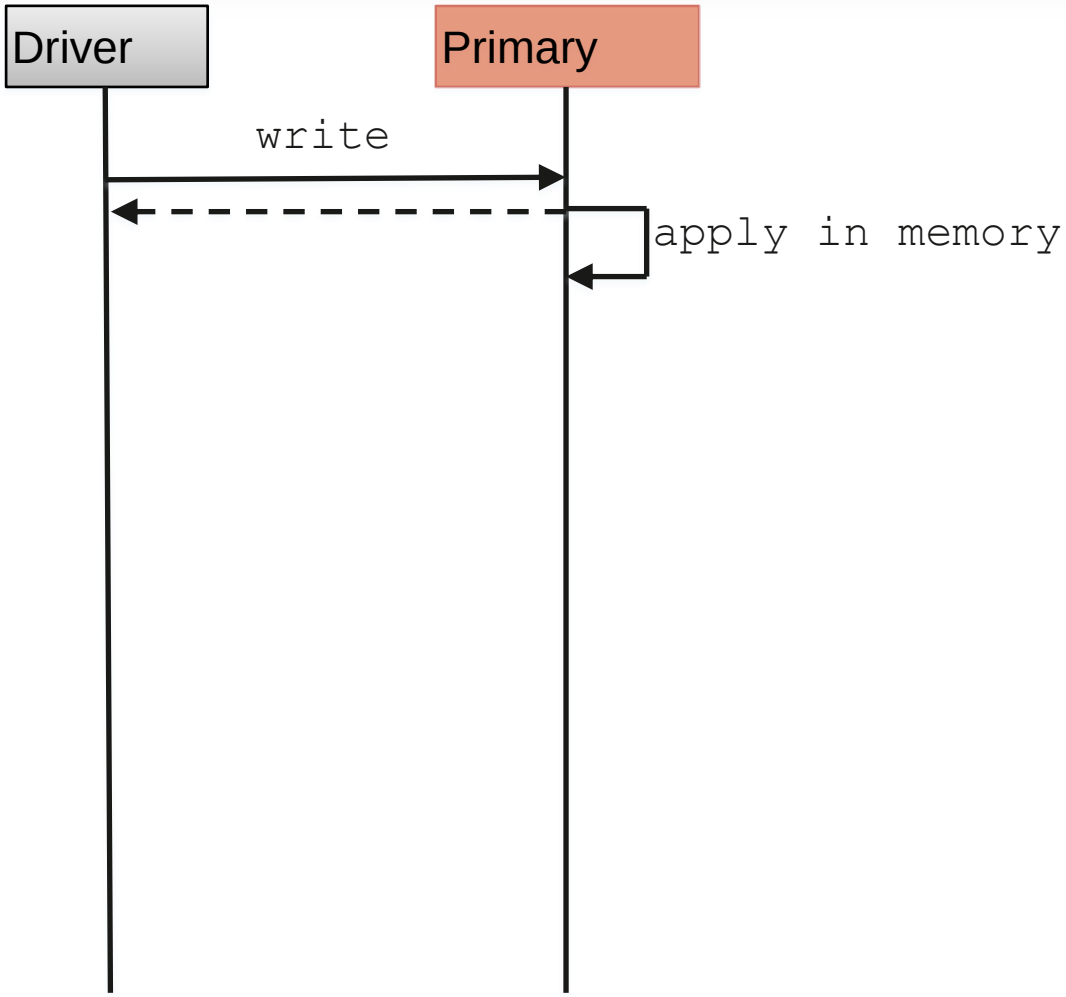
Strong Consistency



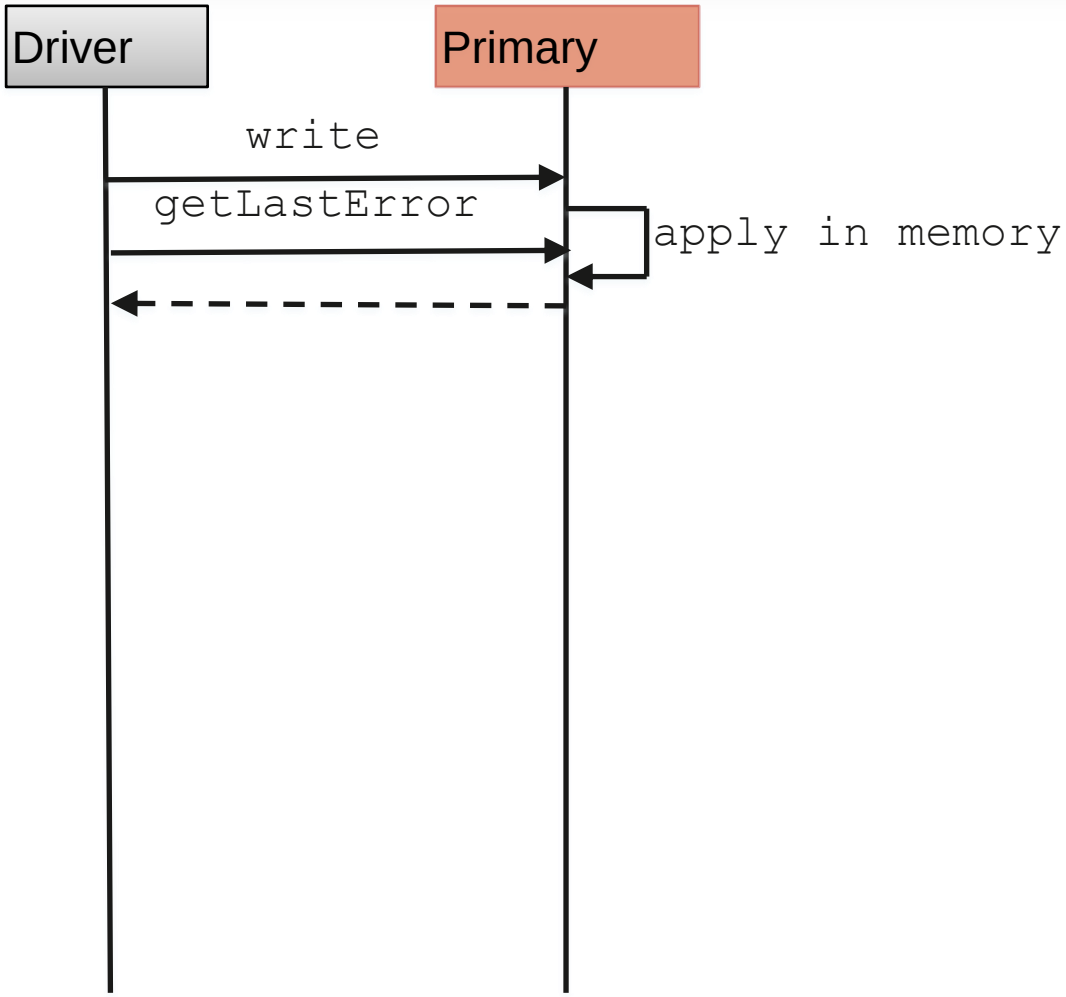
Durability

- Fire and forget
- Wait for error
- Wait for fsync
- Wait for journal sync
- Wait for replication

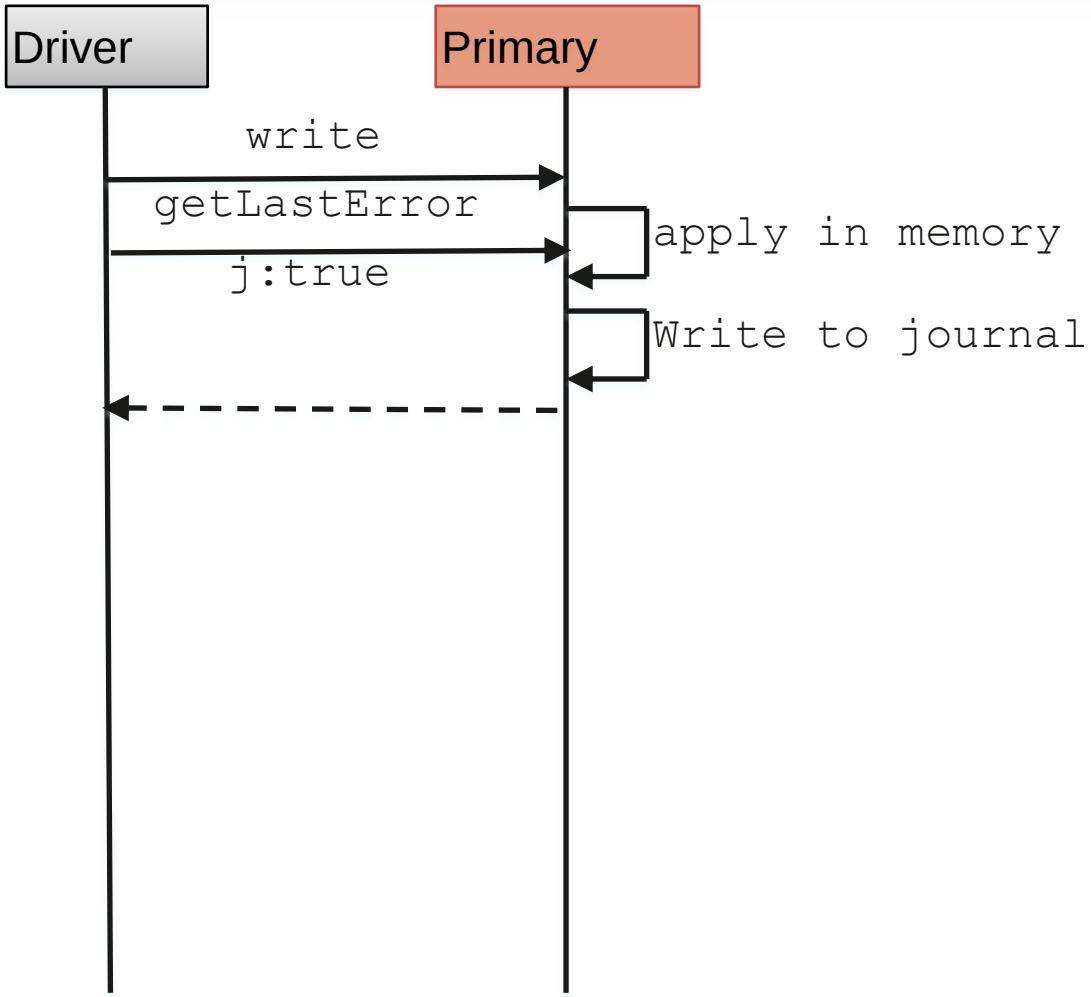
Fire and forget



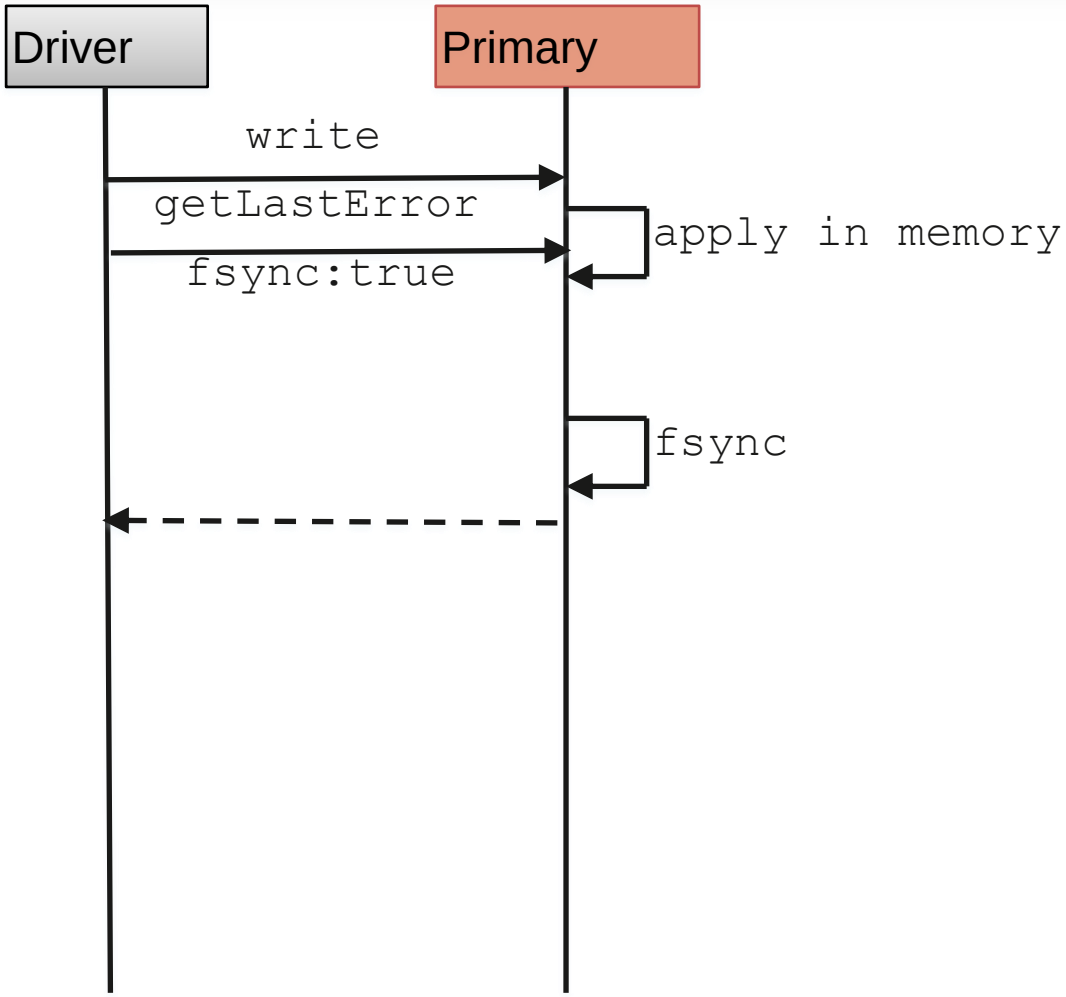
Get last error



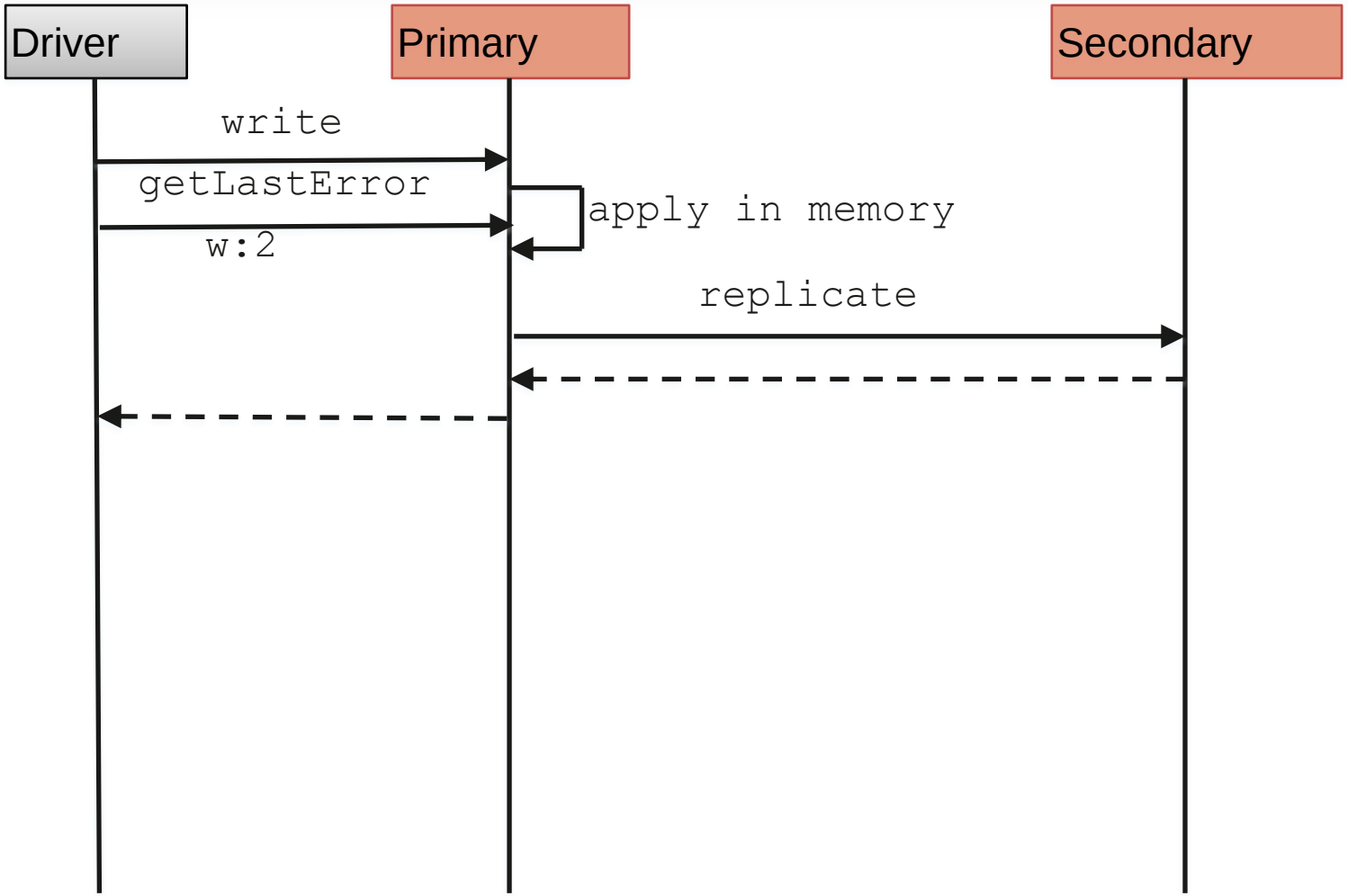
Wait for Journal Sync



Wait for fsync



Wait for replication



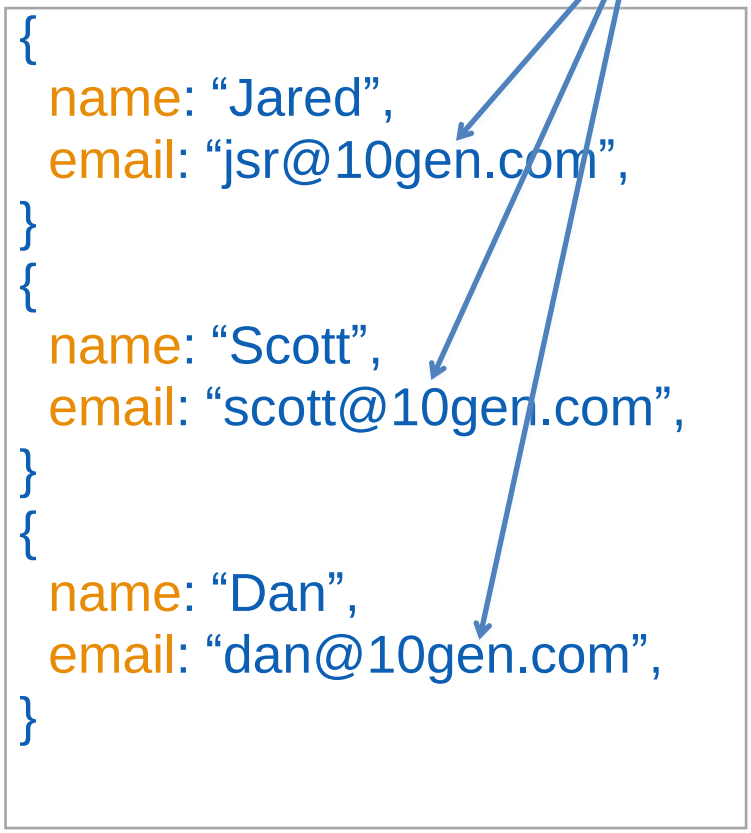
Write Concern Options

Value	Meaning
<n:integer>	Replicate to N members of replica set
“majority”	Replicate to a majority of replica set members
<m:modeName>	Use cutom error mode name

Sharding Details

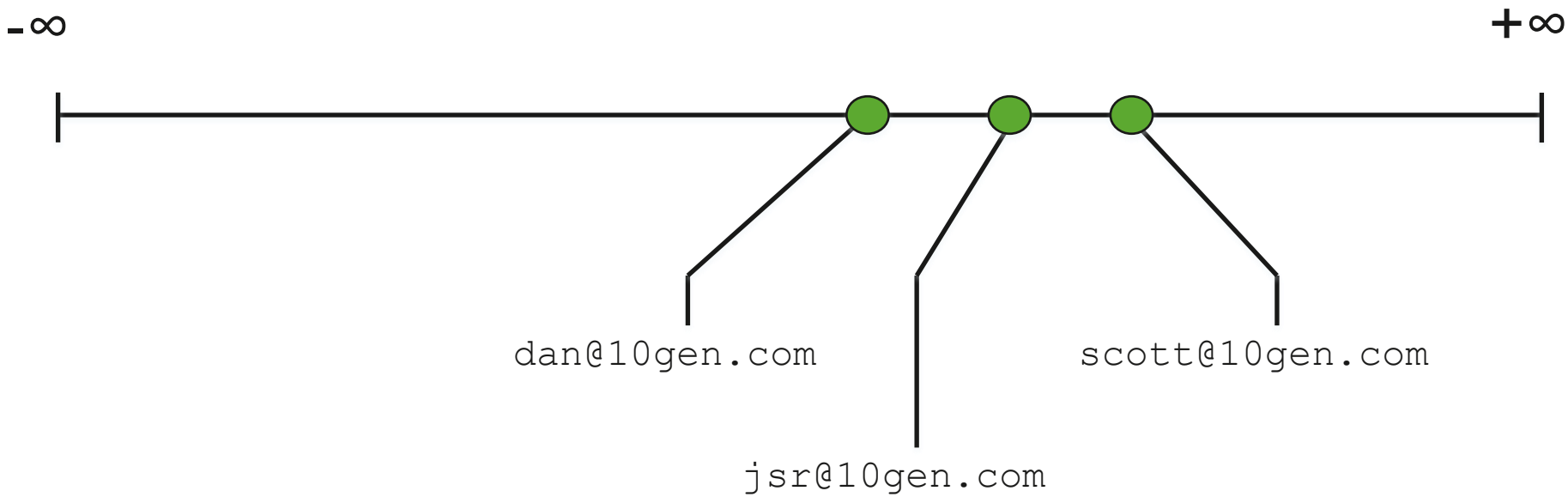
Keys

```
> db.runCommand( { shardcollection: “test.users”,
                   key: { email: 1 } } )
```

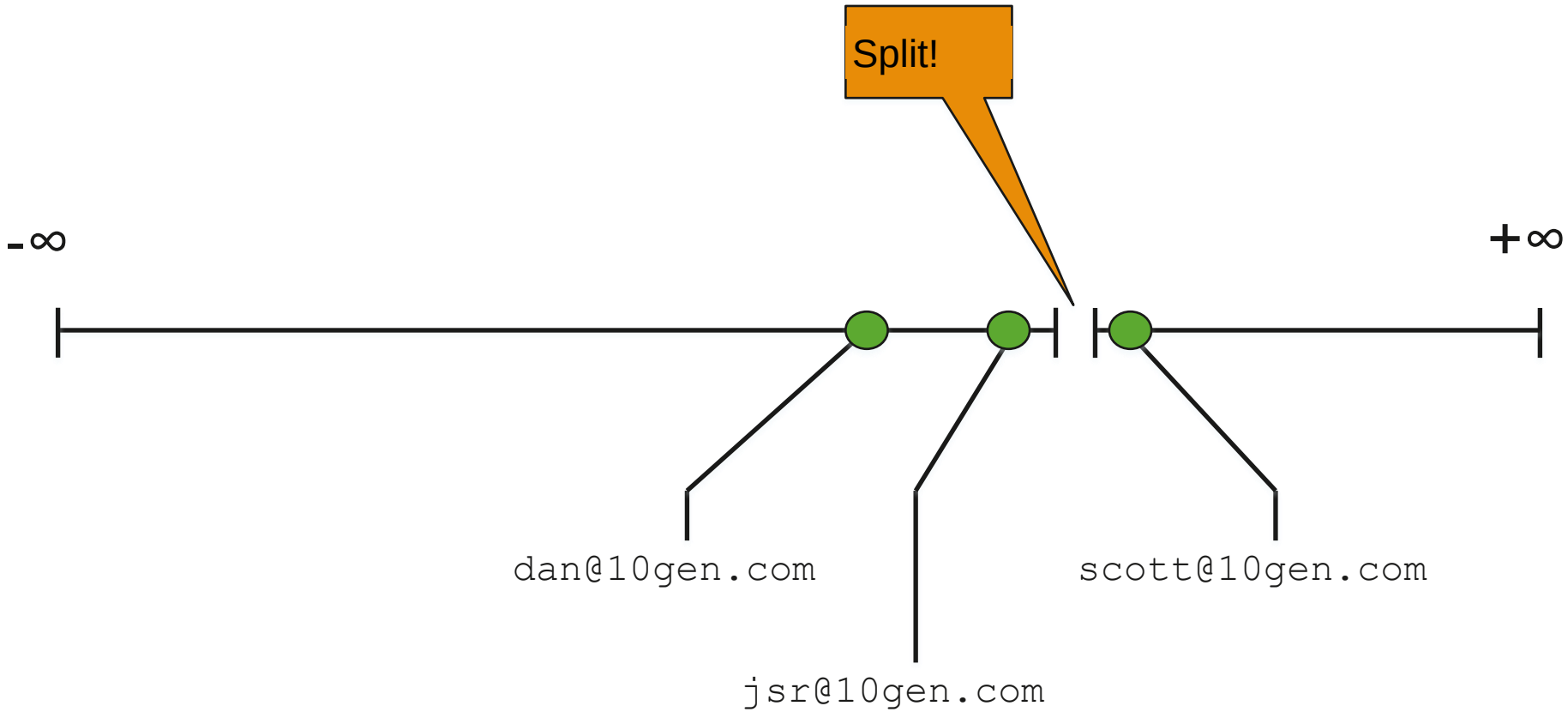


Chunks

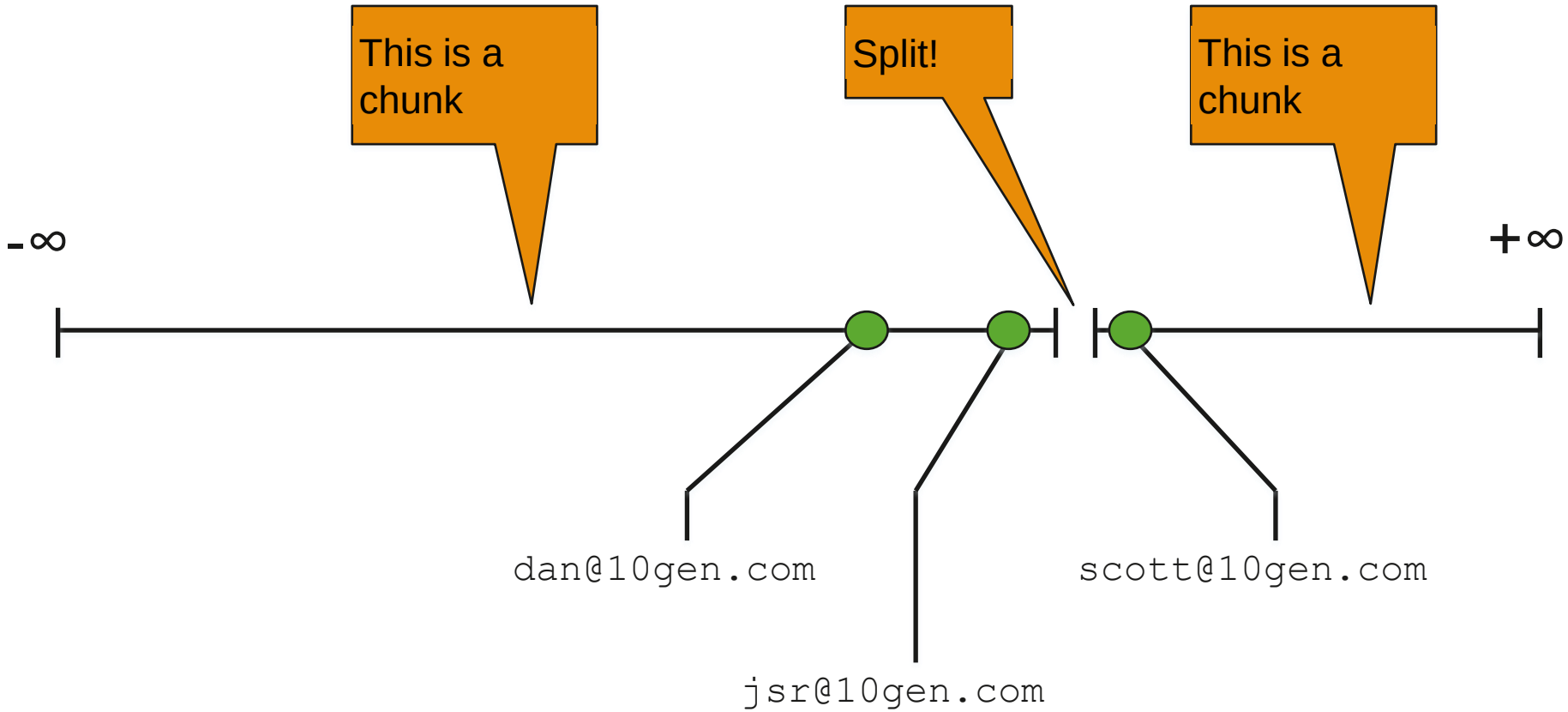
Chunks



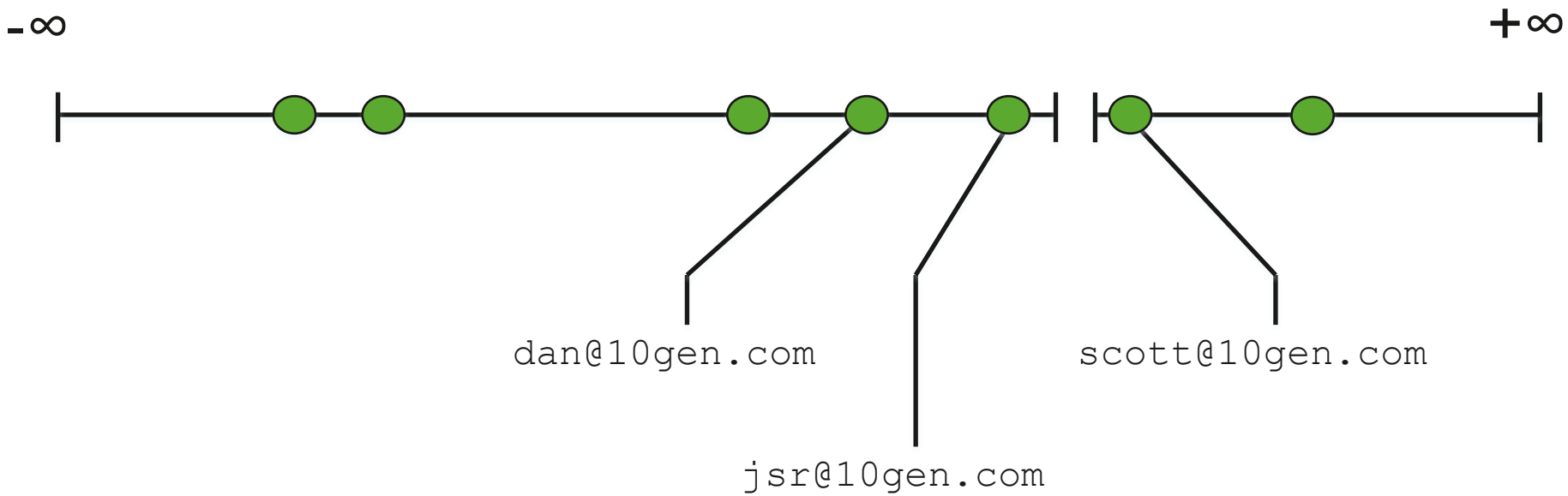
Chunks



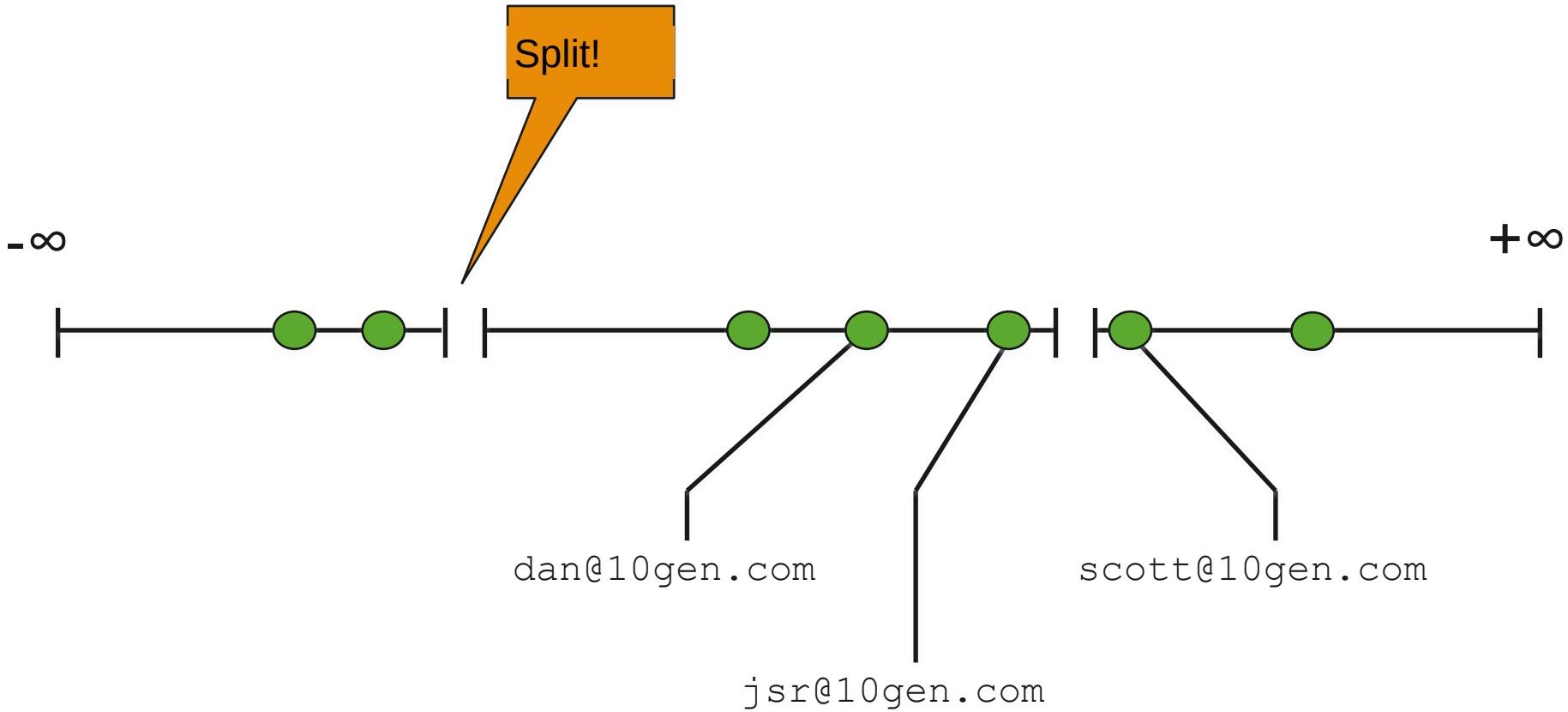
Chunks


This is a chunk
Split!
This is a chunk
-∞
+∞
dan@10gen.com
scott@10gen.com
jsr@10gen.com

Chunks



Chunks

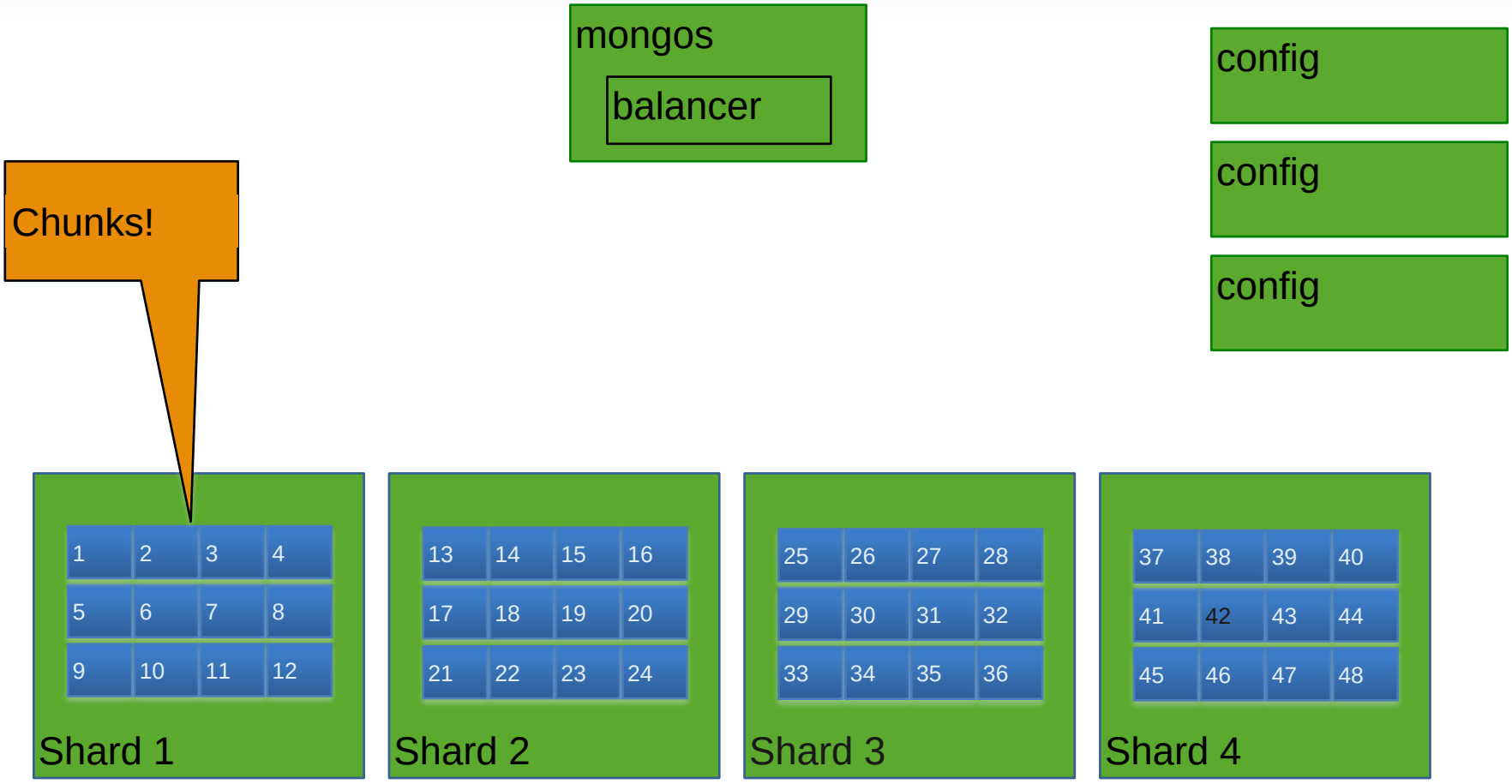


Chunks

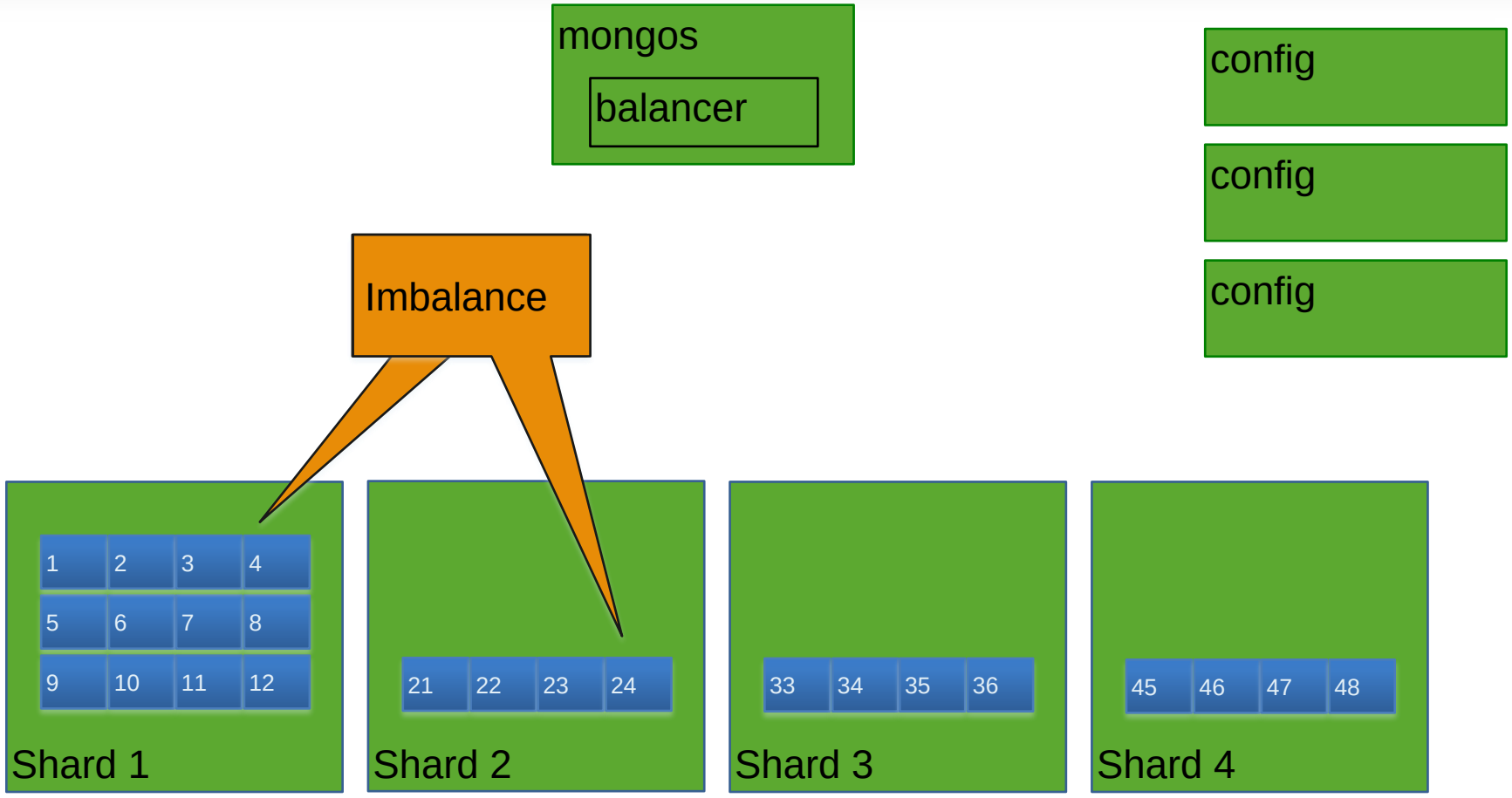
Min Key	Max Key	Shard
-∞	adam@10gen.com	1
adam@10gen.com	jared@10gen.com	1
jared@10gen.com	scott@10gen.com	1
scott@10gen.com	+∞	1

- Stored in the config serers
- Cached in mongos
- Used to route requests and keep cluster balanced

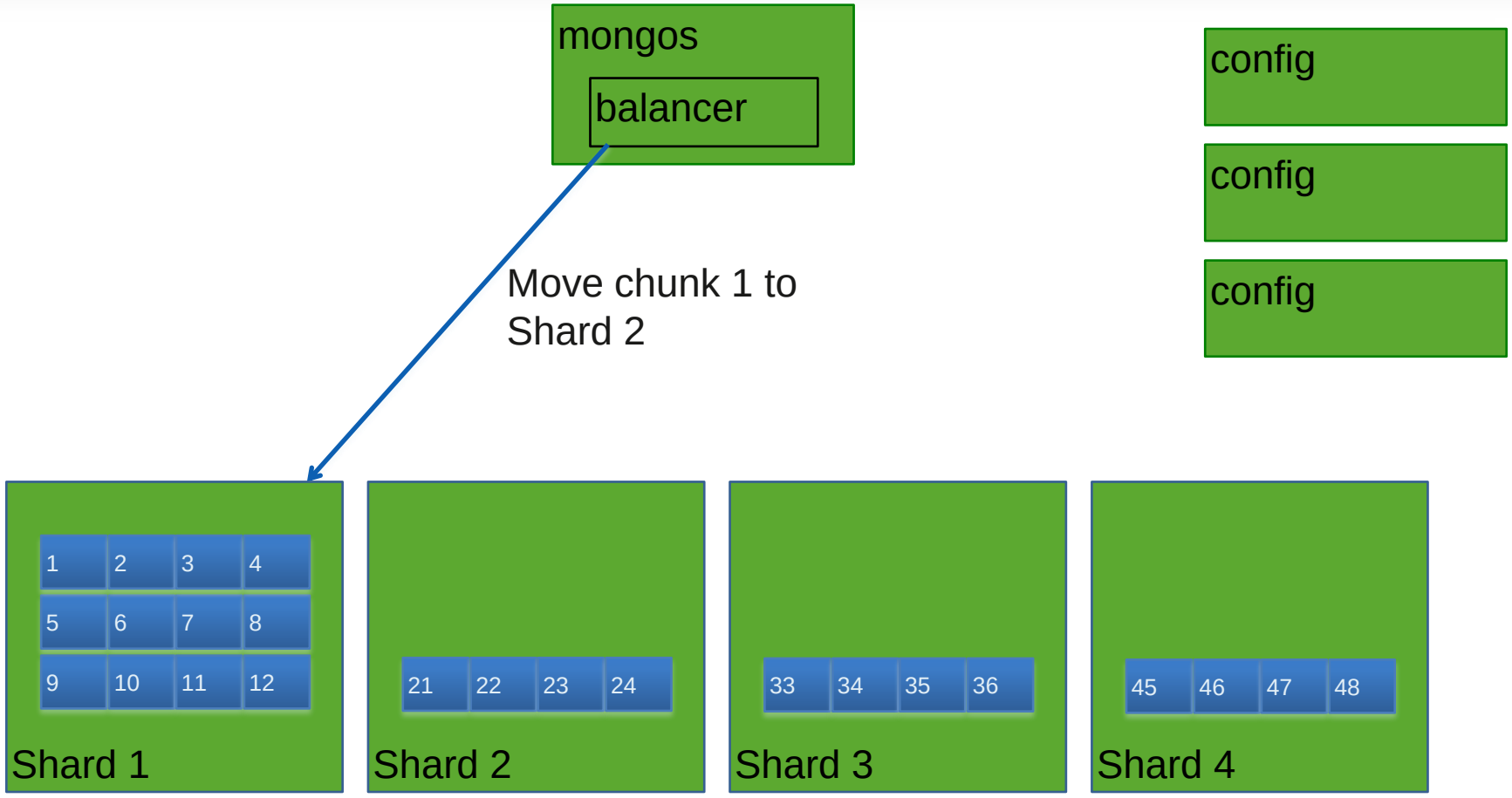
Balancing


mongos
balancer
config
config
config
Chunks!
1 2 3 4
5 6 7 8
9 10 11 12
Shard 1
13 14 15 16
17 18 19 20
21 22 23 24
Shard 2
25 26 27 28
29 30 31 32
33 34 35 36
Shard 3
37 38 39 40
41 42 43 44
45 46 47 48
Shard 4

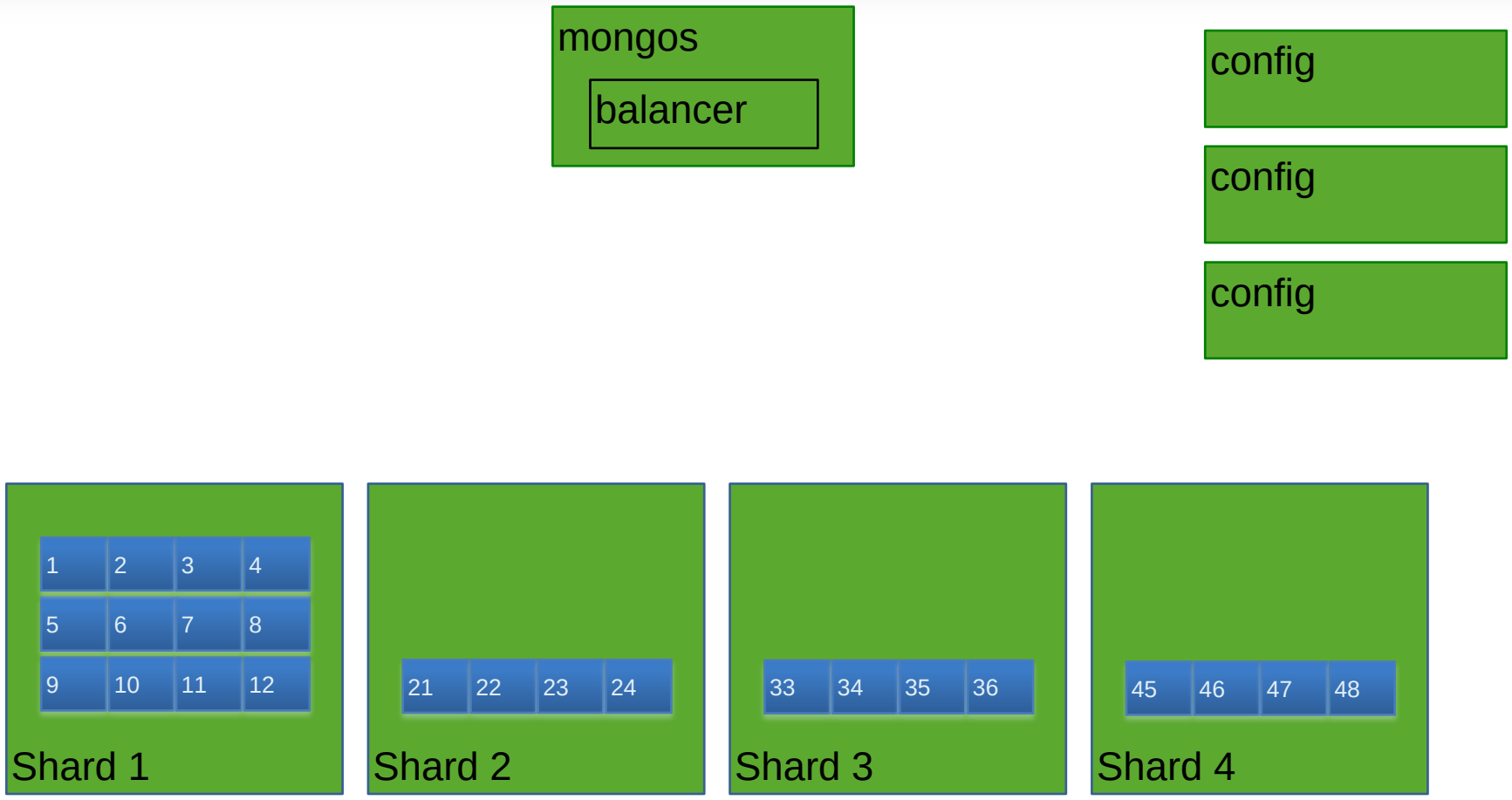
Balancing


mongos
balancer
config
config
config
Imbalance
1
2
3
4
5
6
7
8
9
10
11
12
Shard 1
21
22
23
24
Shard 2
33
34
35
36
Shard 3
45
46
47
48
Shard 4

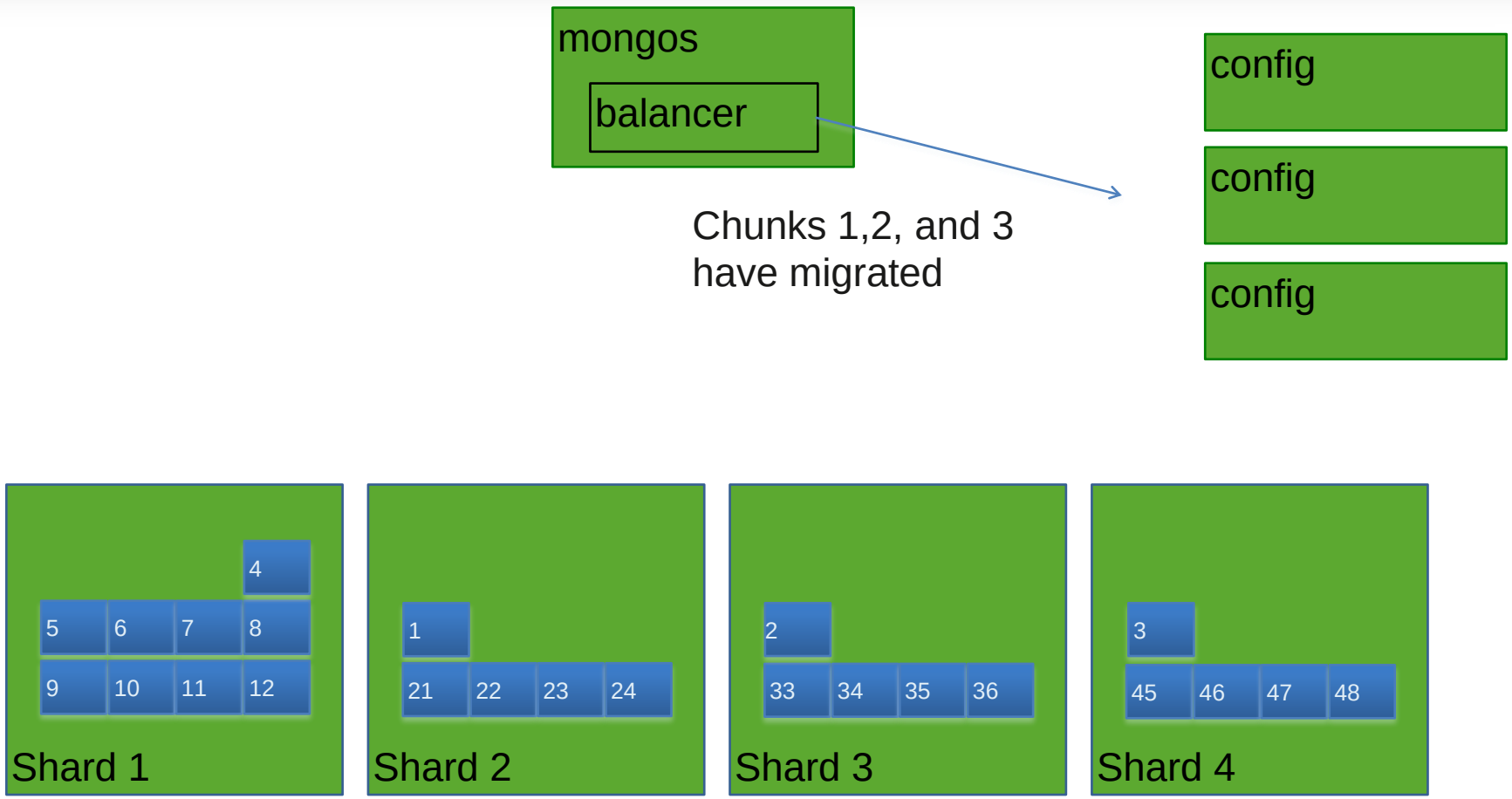
Balancing


mongos
balancer
config
config
config
Move chunk 1 to
Shard 2
1
2
3
4
5
6
7
8
9
10
11
12
Shard 1
21
22
23
24
Shard 2
33
34
35
36
Shard 3
45
46
47
48
Shard 4

Balancing

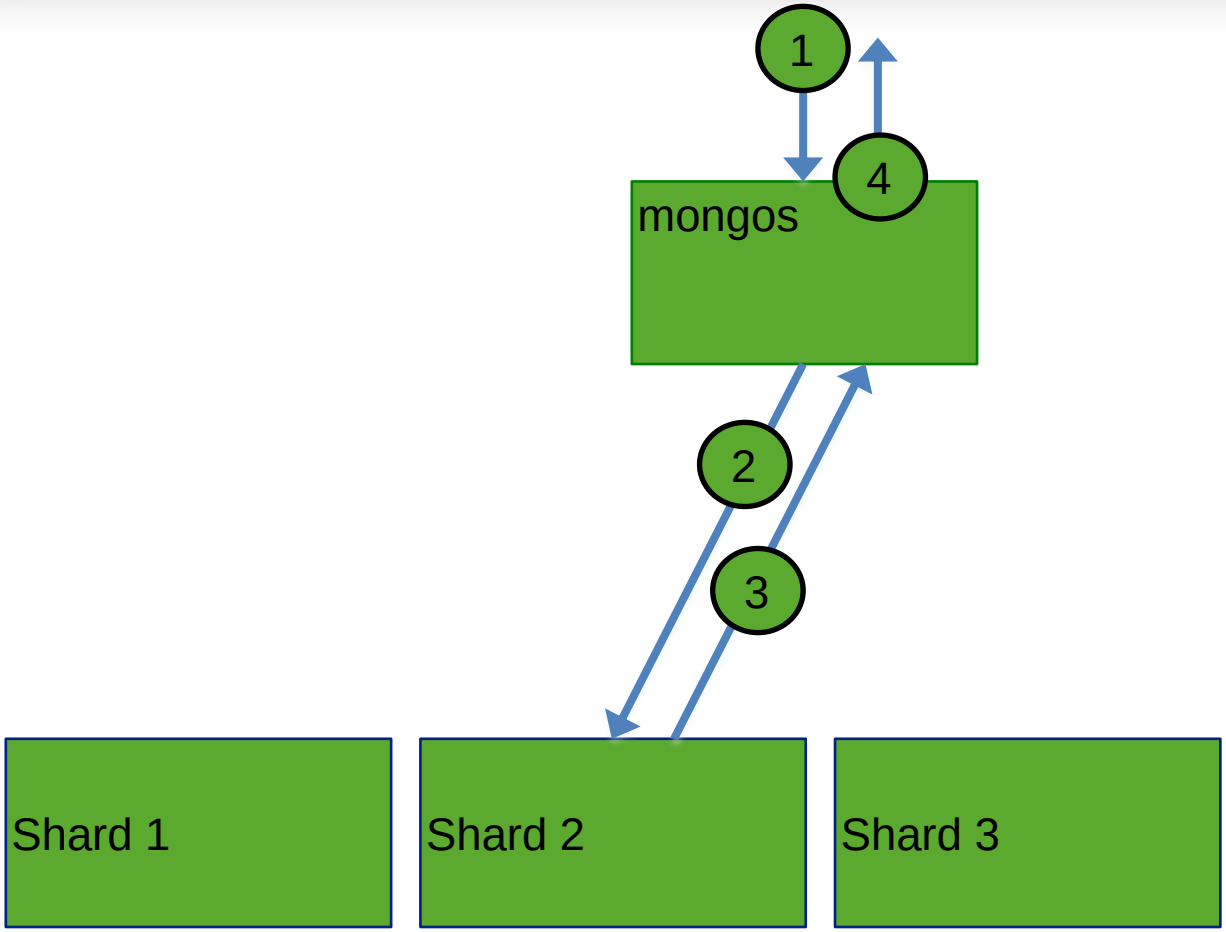

mongos
balancer
config
config
config
1
2
3
4
5
6
7
8
9
10
11
12
Shard 1
21
22
23
24
Shard 2
33
34
35
36
Shard 3
45
46
47
48
Shard 4

Balancing


mongos
balancer
config
config
config
Chunks 1,2, and 3
have migrated
4
5
6
7
8
9
10
11
12
Shard 1
1
21
22
23
24
Shard 2
2
33
34
35
36
Shard 3
3
45
46
47
48
Shard 4

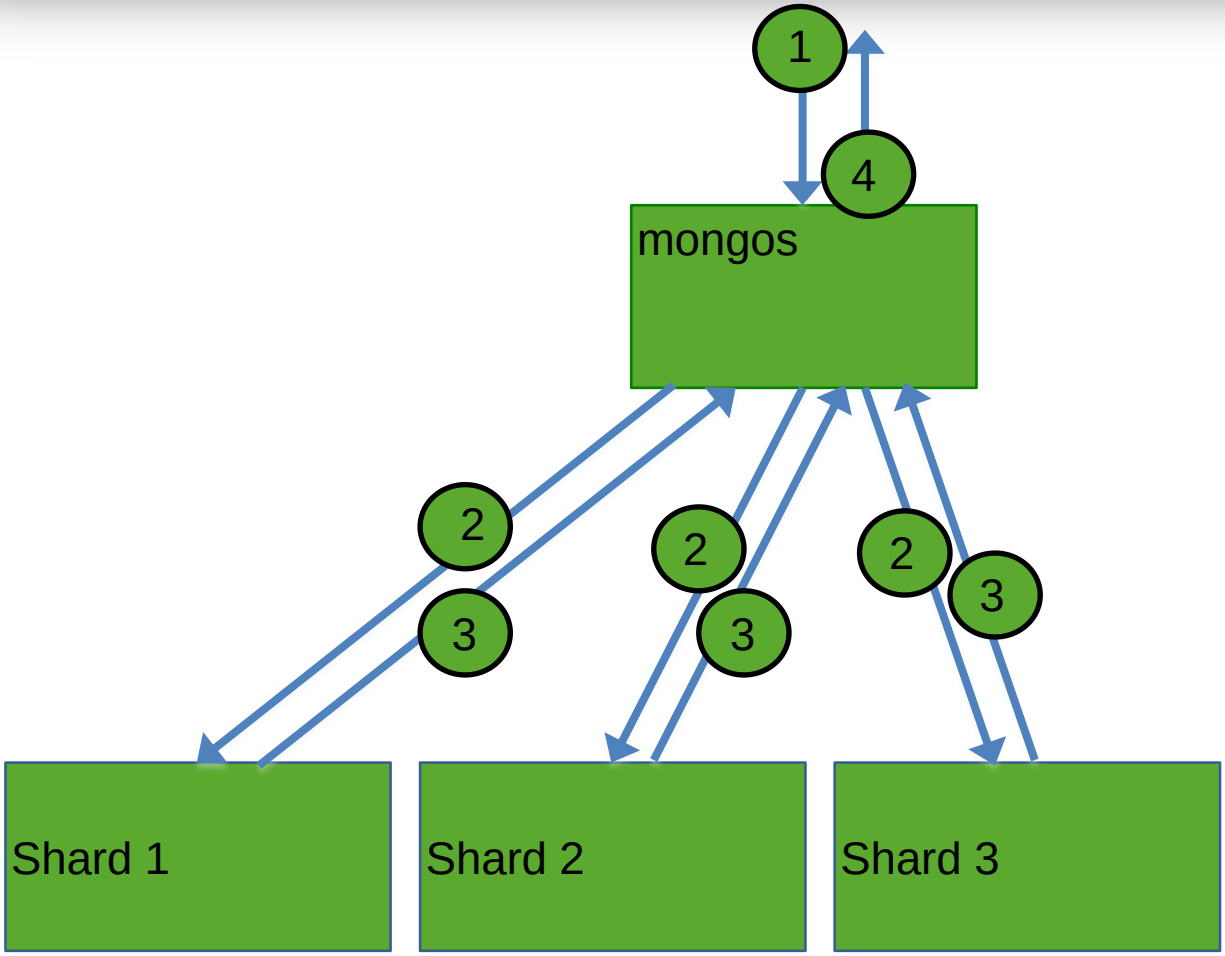
Sharding Details

Routed Request



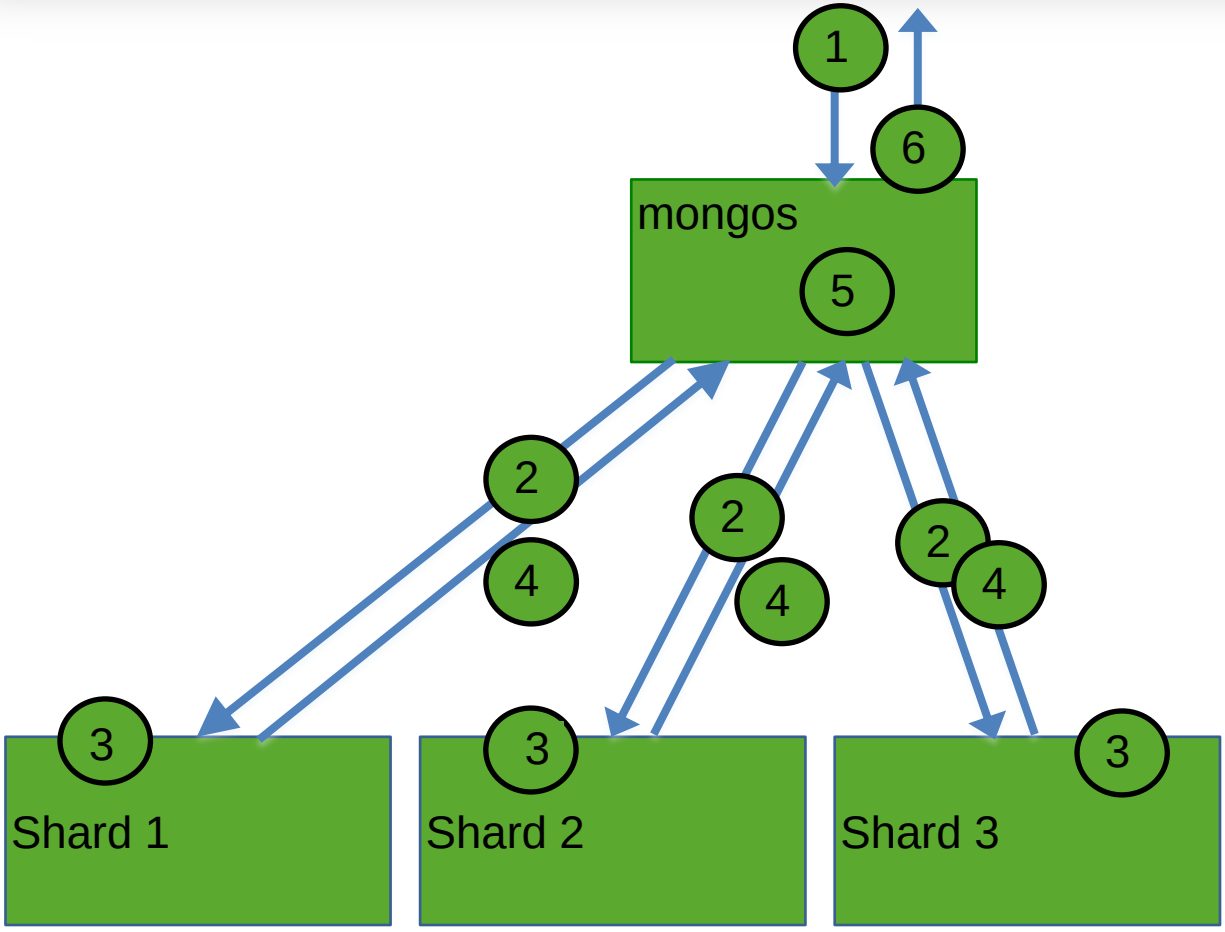
1. Query arrives at mongos
2. mongos routes query to a single shard
3. Shard returns results of query
4. Results returned to client

Scatter Gather



1. Query arrives at mongos
2. mongos broadcasts query to all shards
3. Each shard returns results for query
4. Results combined and returned to client

Distributed Merge Sort



1. Query arrives at mongos
2. mongos broadcasts query to all shards
3. Each shard locally sorts results
4. Results returned to mongos
5. mongos merge sorts individual results
6. Combined sorted result returned to client

Writes

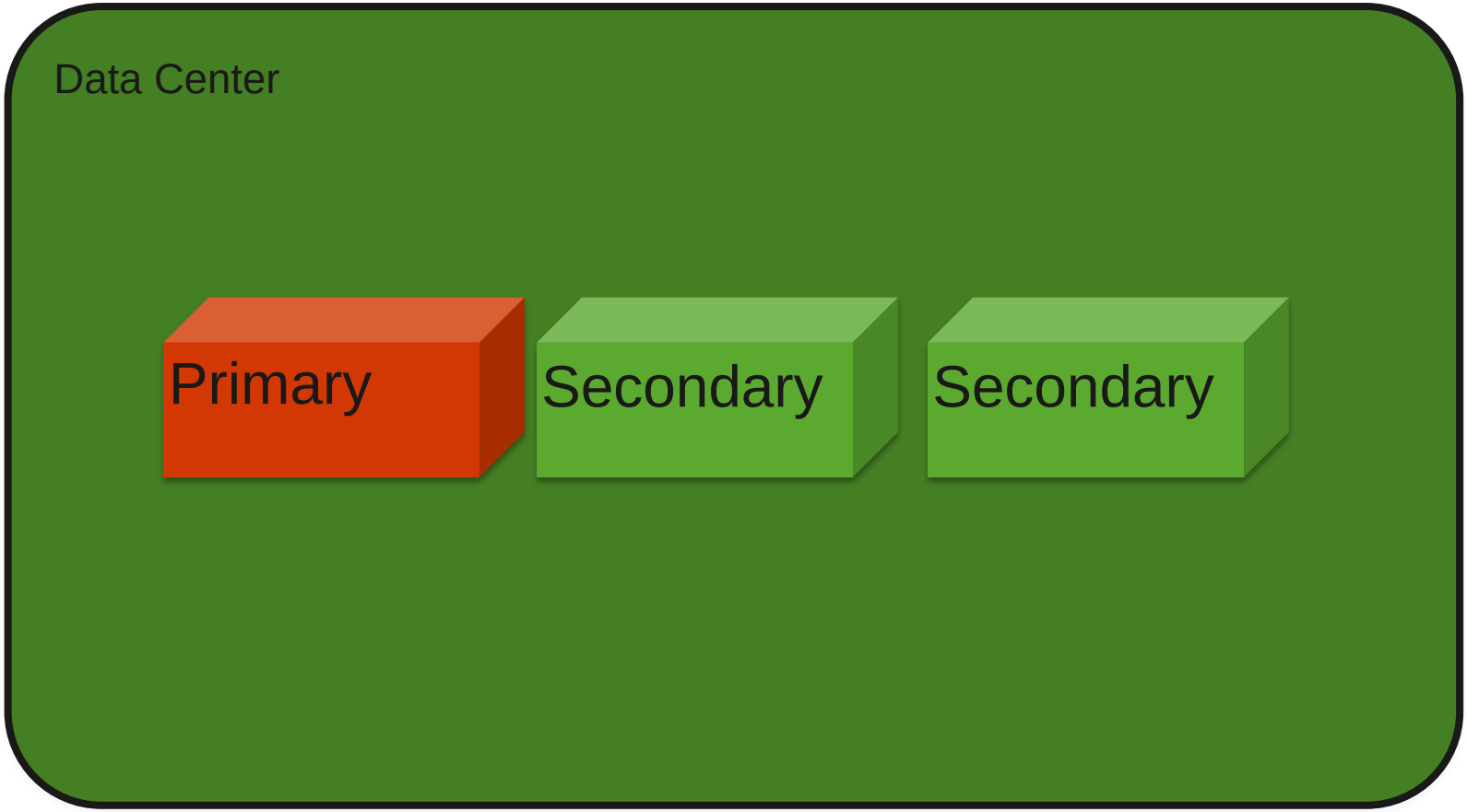
Inserts	Requires shard key	`db.users.insert({ name: "Jared", email: "jsr@10gen.com"})`
Removes	Routed	`db.users.delete({ email: "jsr@10gen.com"})`
	Scattered	`db.users.delete({name: "Jared"})`
Updates	Routed	`db.users.update( {email: "jsr@10gen.com"}, {$set: { state: "CA"}})`
	Scattered	`db.users.update( {state: "FZ"}, {$set:{ state: "CA"}} )`

Queries

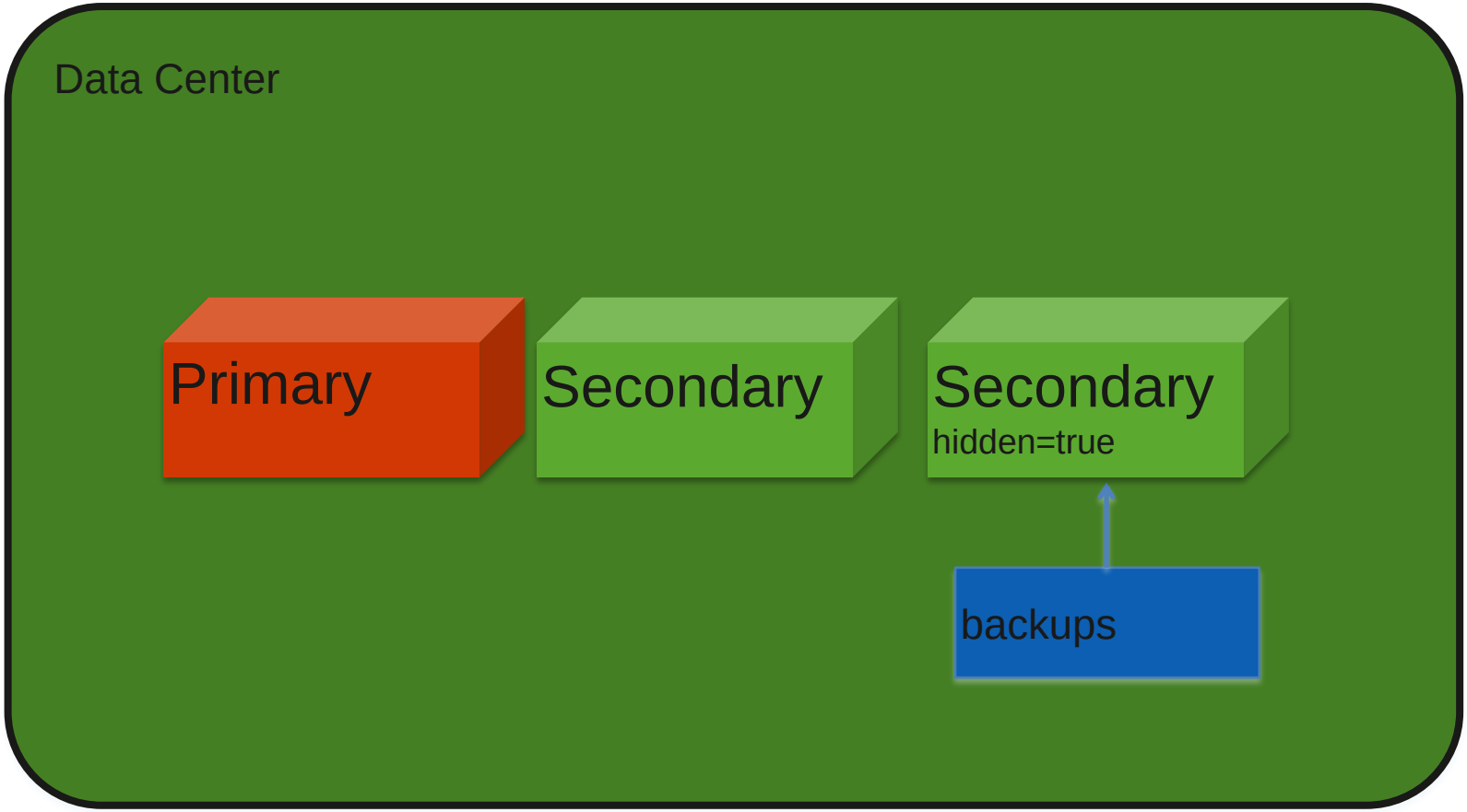
By Shard Key	Routed	`db.users.find( {email: "jsr@10gen.com"})`
Sorted by shard key	Routed in order	`db.users.find().sort({email:-1})`
Find by non shard key	Scatter Gather	`db.users.find({state:"CA"})`
Sorted by non shard key	Distributed merge sort	`db.users.find().sort({state:1})`

Common Deployment Scenarios

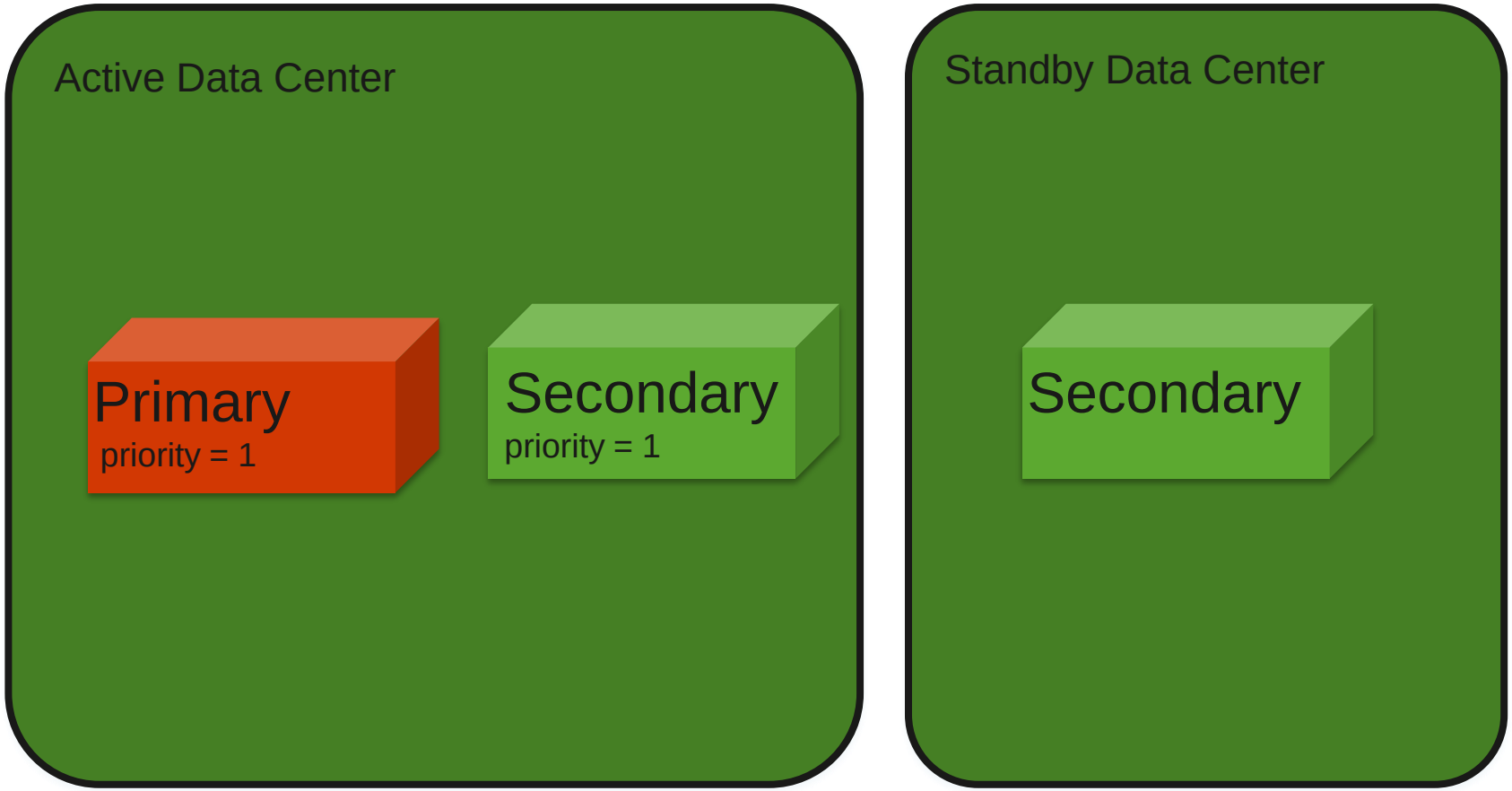
Typical



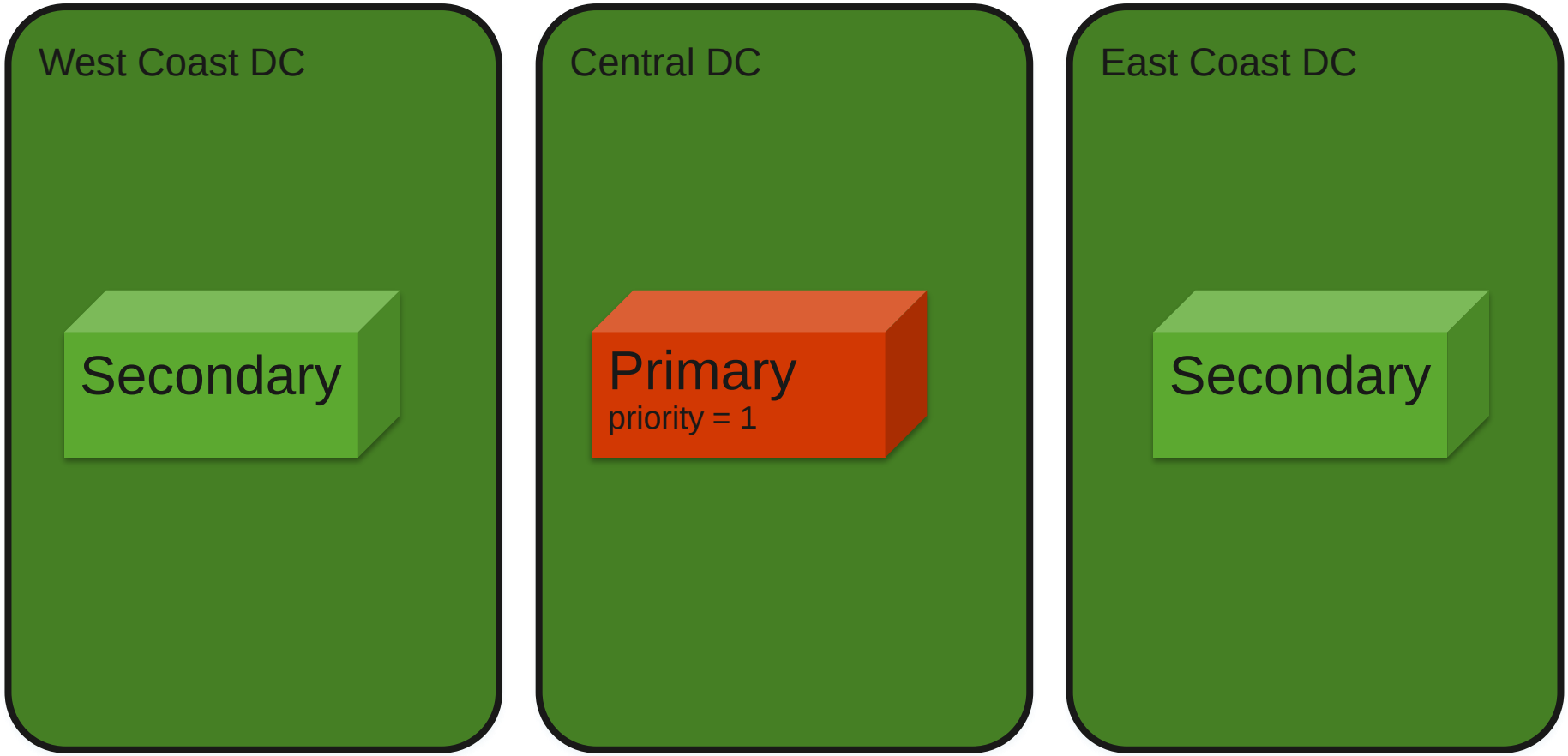
Backup Node

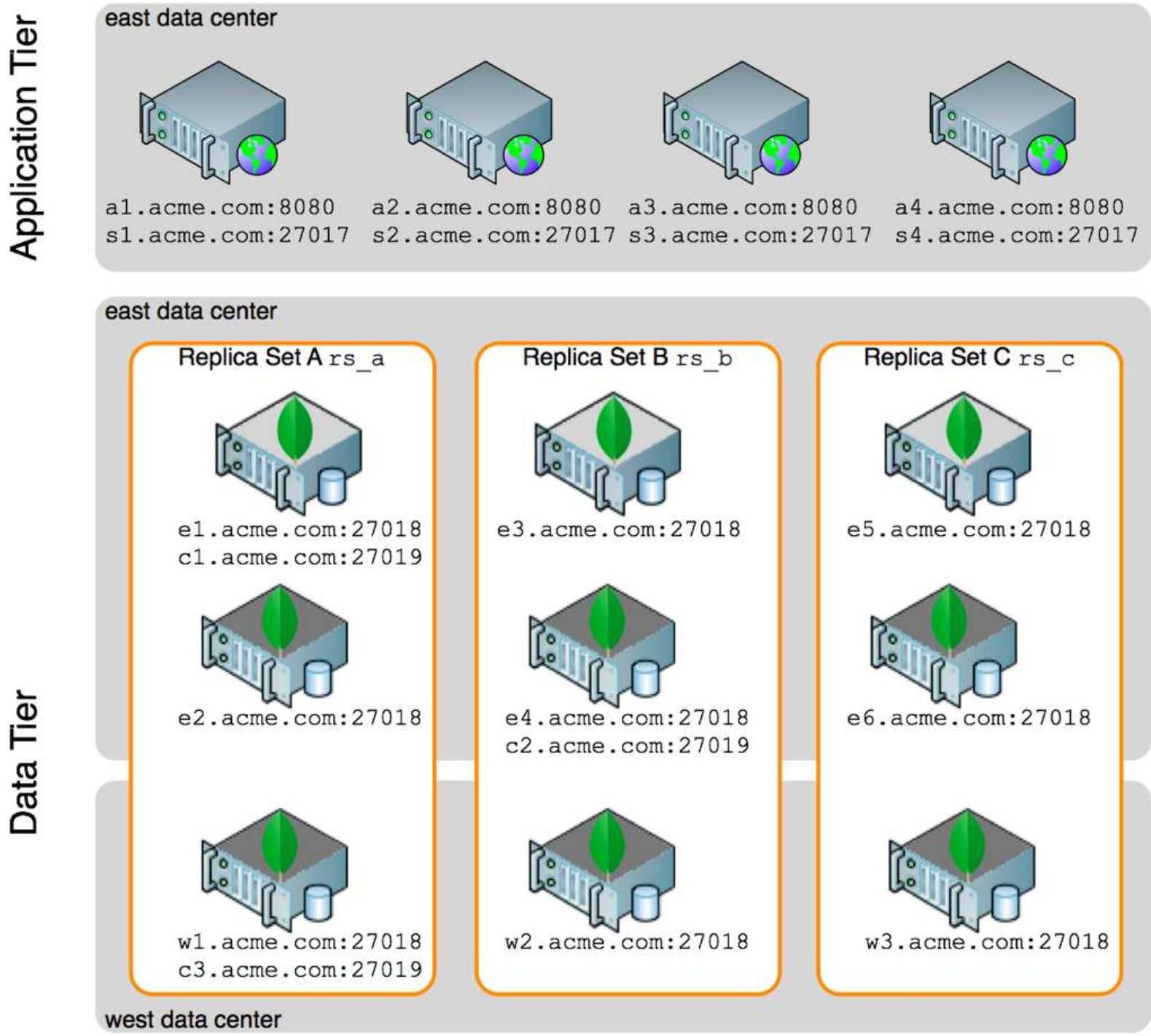

Data Center
Primary
Secondary
Secondary
hidden=true
backups

Disaster Recovery



Multi Data Center




Application Tier
east data center
a1.acme.com:8080
s1.acme.com:27017
a2.acme.com:8080
s2.acme.com:27017
a3.acme.com:8080
s3.acme.com:27017
a4.acme.com:8080
s4.acme.com:27017
Data Tier
east data center
Replica Set A rs_a
e1.acme.com:27018
c1.acme.com:27019
e2.acme.com:27018
w1.acme.com:27018
c3.acme.com:27019
Replica Set B rs_b
e3.acme.com:27018
e4.acme.com:27018
c2.acme.com:27019
w2.acme.com:27018
Replica Set C rs_c
e5.acme.com:27018
e6.acme.com:27018
w3.acme.com:27018
west data center

Citations

- History of Database Management (http://bit.ly/w3r0dv)
- EMC IDC Study (http://bit.ly/y1mJgJ)
- Gartner & Big Data (http://bit.ly/xvRP3a)
- SQL (http://en.wikipedia.org/wiki/SQL)
- Database Management Systems http://en.wikipedia.org/wiki/Dbms)
- Dynamo: Amazon's Highly Available Key-value Store (http://bit.ly/A8F8oy)
- CAP Theorem (http://bit.ly/zvA6O6)
- NoSQL Google File System and BigTable (http://oreil.ly/wOXliP)
- NoSQL Movement whitepaper (http://bit.ly/A8RBuJ)
- Sample ERD diagram (http://bit.ly/xV30v)

Backup Slides

Brewer's CAP Theorem

- Impossible for a distributed computer system to simultaneously provide all three of the following guarantees
  - Consistency - All nodes see the same data at the same time.
  - Availability - A guarantee that every request receives a response about whether it was successful or failed.
  - Partition tolerance - No set of failures less than total network failure is allowed to cause the system to respond incorrectly