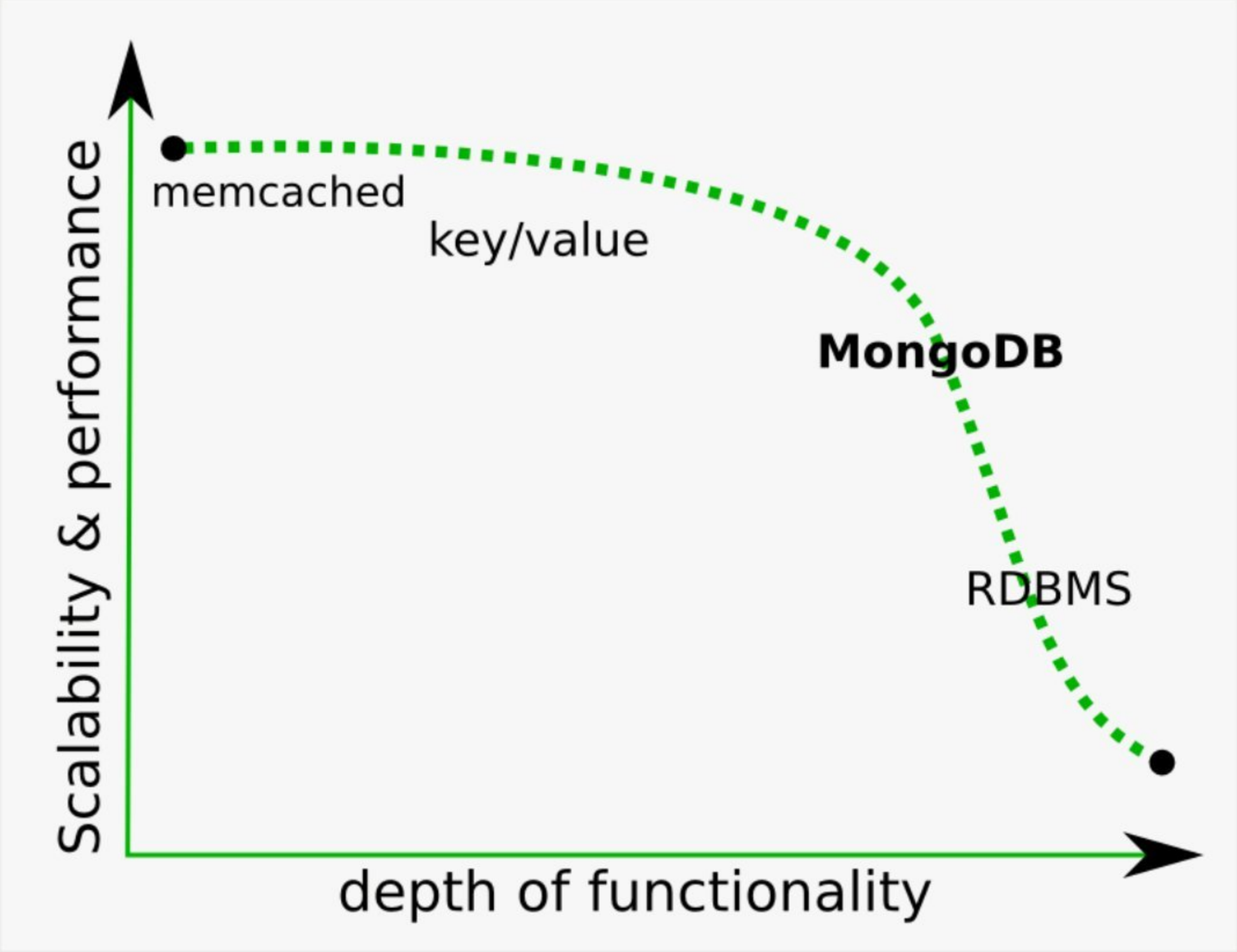
Scaling MongoDB

Derick Rethans

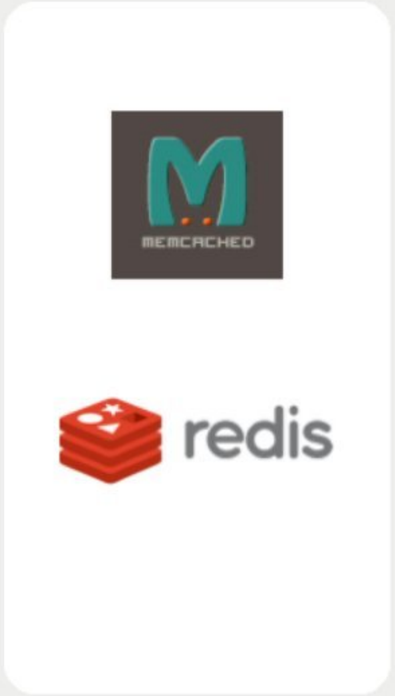
derick@mongodb.com—@derickr

https://joind.in/20481

Database landscape

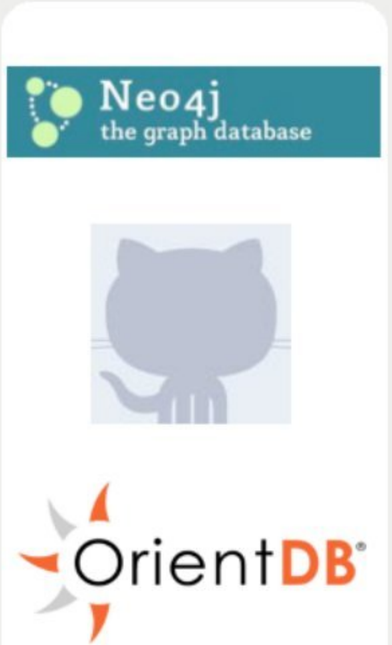


NoSQL



Key/value

Column



Graph

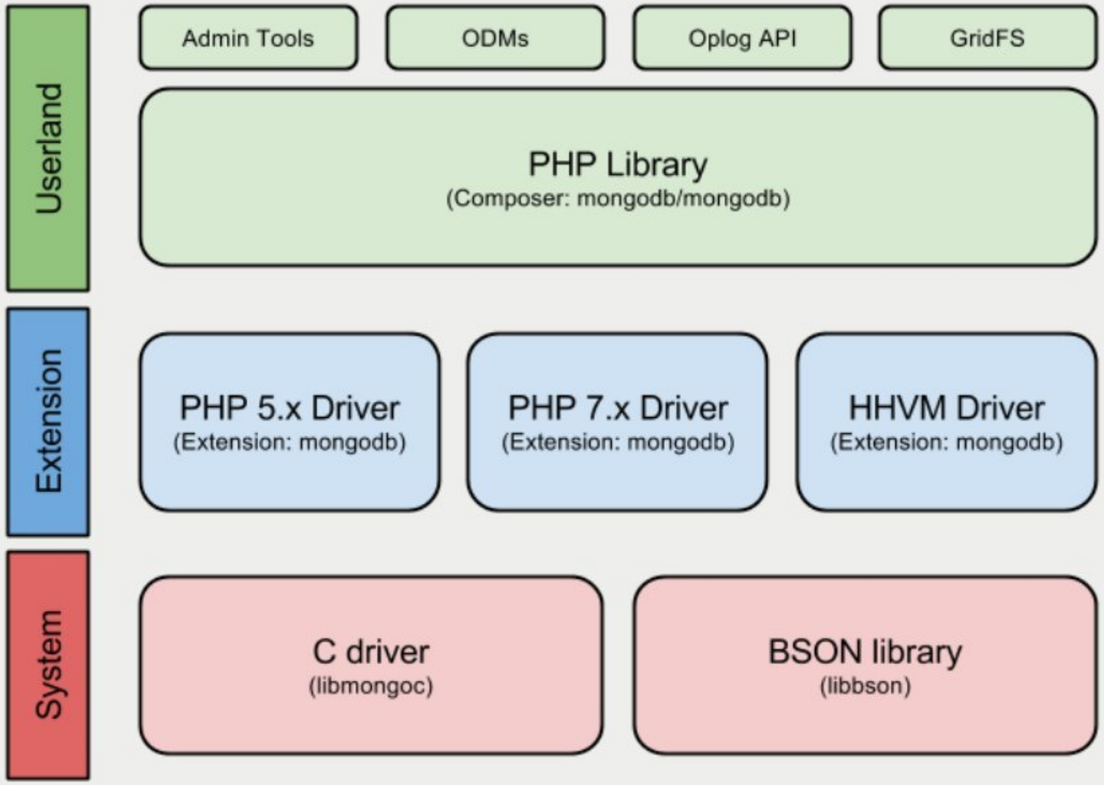
Document

CAP

Eric Brewer in 2002

- **C**onsistency:
  A read sees all previously completed writes
  (Not the same consistency as in ACID)
- **A**vailability:
  Guarantees that every request receives a response about whether it was successful or failed
- **P**artition Tolerance:
  Guaranteed properties are maintained even when network failures prevent some machines from communicating with others.

New architecture



Terminology

- **Document**: the data (row)
- **Collection**: contains documents (table, view)
- **Index**
- **Embedded Document** (~join)
- **Sharding** (partitioning)

Documents: Complex

```
{
  "_id" : "derick@localhost",
  "fullname" : "Derick Rethans",
  "slug" : "derick-rethans",
  "created_at" : 1452546141,
  "timezone" : "Europe/London",
  "confirmed" : true,
  "confirmed_at" : 1452546148,
  "location" : "London, UK",
  "words" : [ "derick", "rethans", "london", "uk" ],
  "count" : 16,
  "count_unique" : 13,
  "badges" : [
    { n: "unique1", l: 1 },
    { n: "age21", l: 3 }
  ],
  "isAdmin" : true
}
```

- `_id`: does not have to be an Object ID
- Values can be arrays, documents, or arrays of documents

Replication

Replication

Use cases

- High Availability (auto-failover)
- Backups
  - Online, Delayed Copy (fat finger)
  - Point in Time (PiT) backups
- Use (hidden) replica for secondary workload
  - Analytics
  - Data-processing
  - Integration with external systems (f.e. ElasticSearch)

Types of outages

Planned

- Hardware upgrade
- Relocation of data to new file-system / storage
- Software upgrade

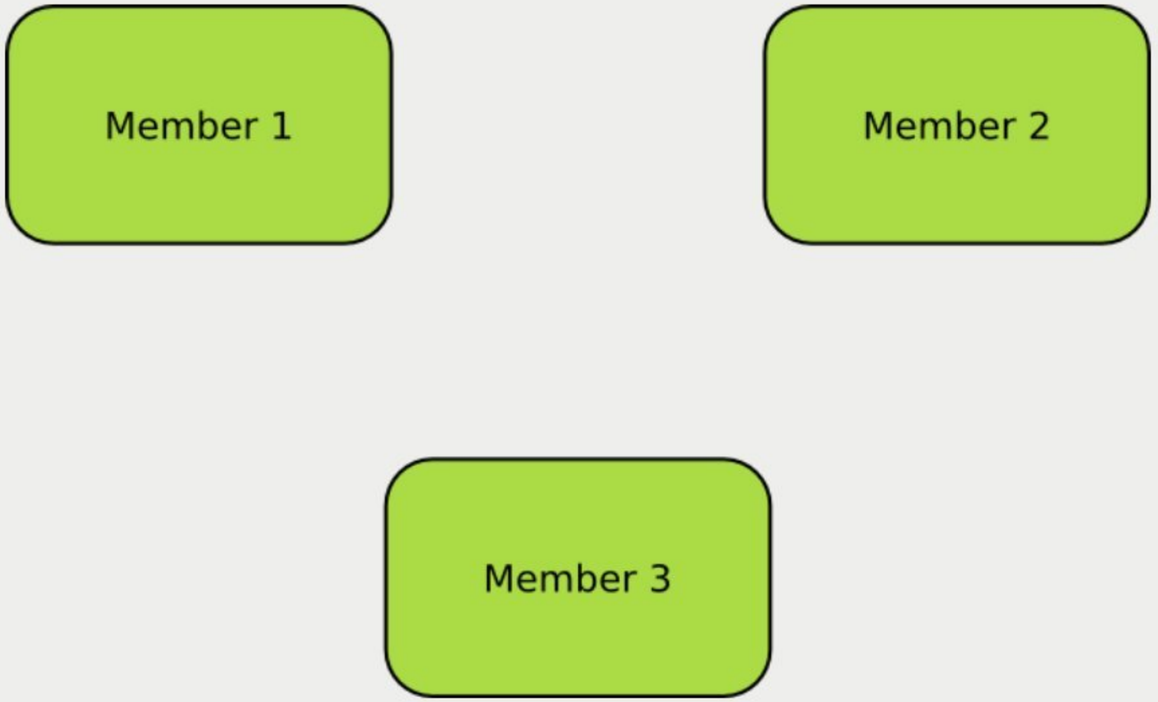
Unplanned

- Hardware failure
- Data center failure
- Region outage
- Human error

MongoDB Replica Set Features

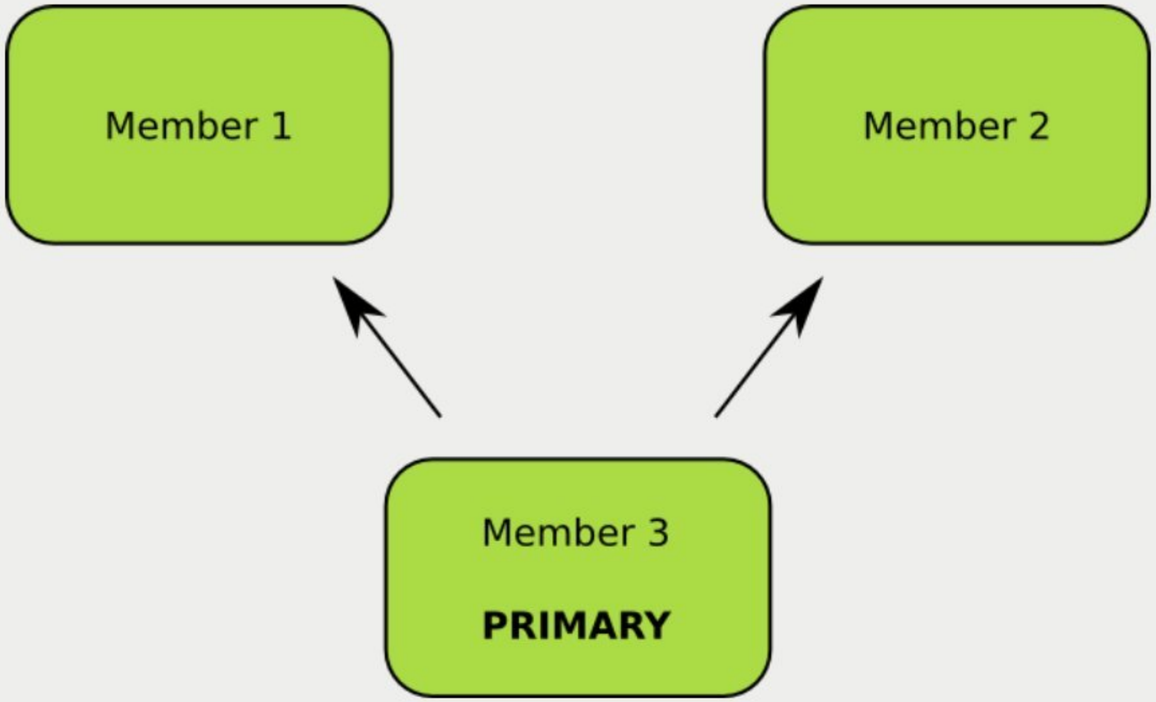
- A cluster of N servers
- All writes to primary
- Reads can be from primary (default) or a secondary
- Any (one) node can be primary
- Consensus election of primary
- Automatic failover and recovery

How MongoDB replication works



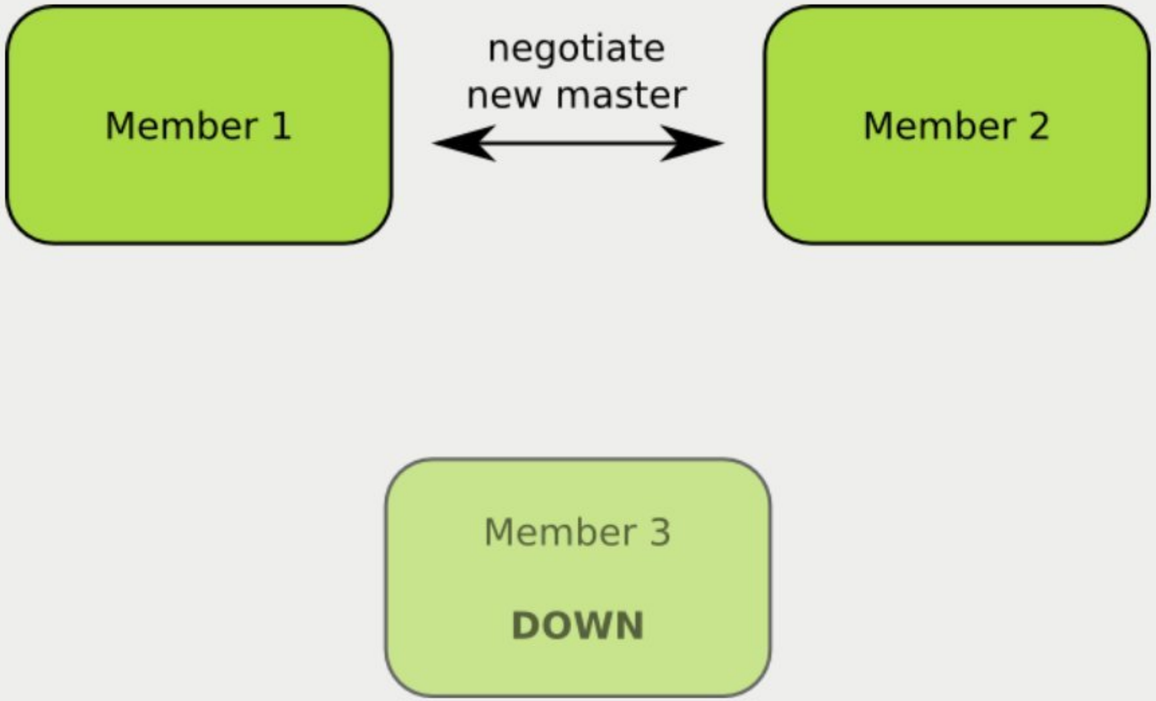
- A set should have 3 or more nodes, and always an **odd** number of voters (for calculating majority)

How MongoDB replication works



- Election establishes the PRIMARY
- Data replication from PRIMARY to SECONDARY

How MongoDB replication works

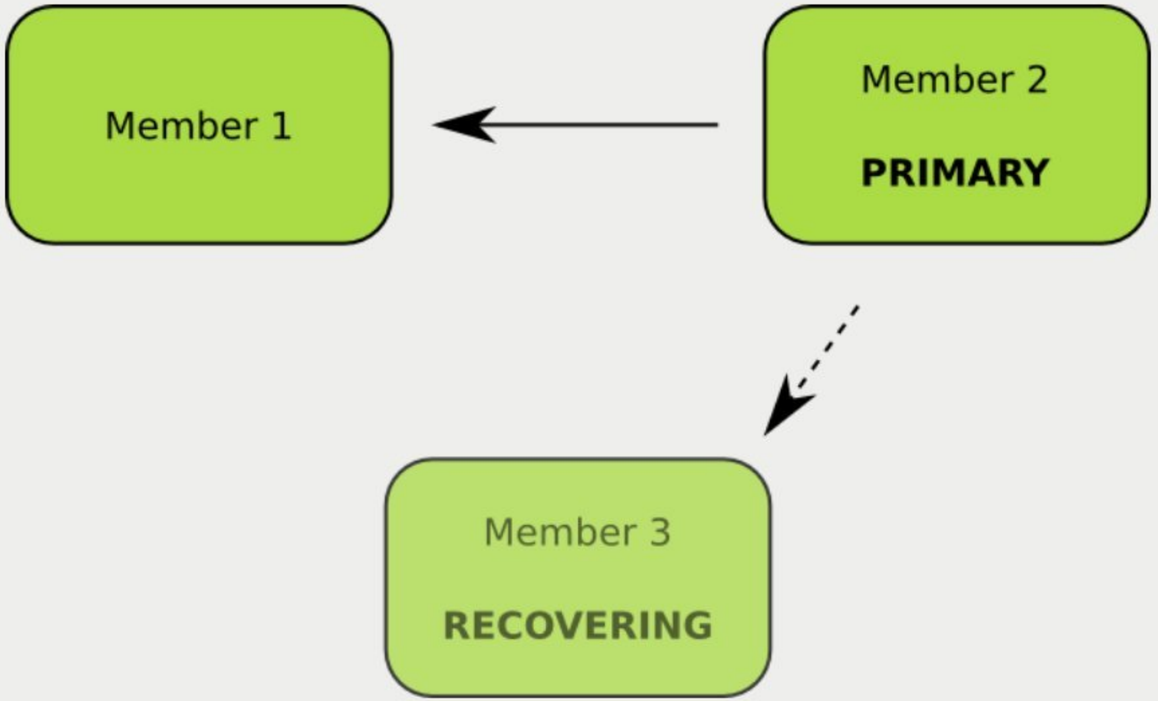


- PRIMARY may fail
- Automatic election of new PRIMARY if majority exists

How MongoDB replication works

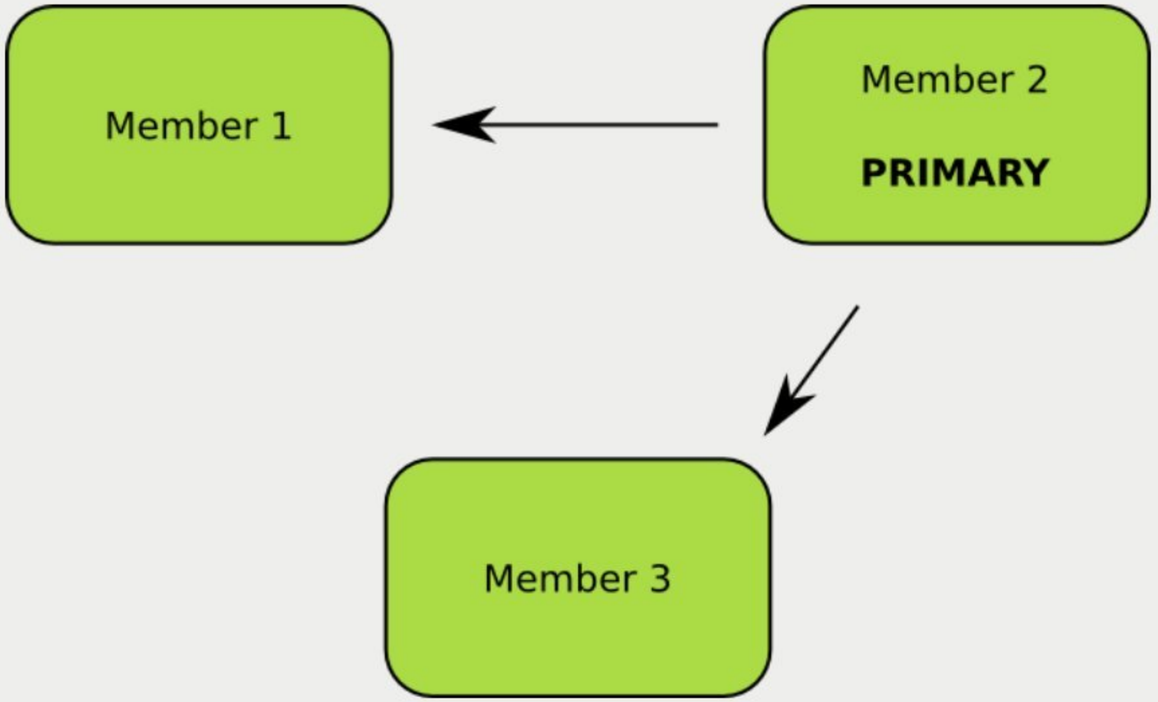
- New PRIMARY elected
- Replica set re-established

How MongoDB replication works



- Automatic recovery

How MongoDB replication works



- Replica set re-established

How does replication work?

- Change operations are written to the oplog
  - A single update/delete affecting multiple documents can result in many oplog entries
  - The oplog is a capped collection (fixed size)
    - Must have enough space to allow new secondaries to catch up after syncing from a primary, and cope with any applicable slaveDelay
- Secondaries query the primary's oplog and apply what they find
- All replica set members contain an oplog

Connection String

```
<?php
$options = [ 'replicaSet' => 'seta' ];

$m = new \MongoDB\Client( 'mongodb://localhost:13000/?replicaSet=seta' );

$m = new \MongoDB\Client( 'mongodb://localhost:13000,localhost:13001', $options );

$m = new \MongoDB\Client( 'mongodb://user:password@localhost:13000/demo', $options );
?>
```

- Add more than one host for seeding (a majority)
- Don't add the arbiters to the connection string

Managing a replica set

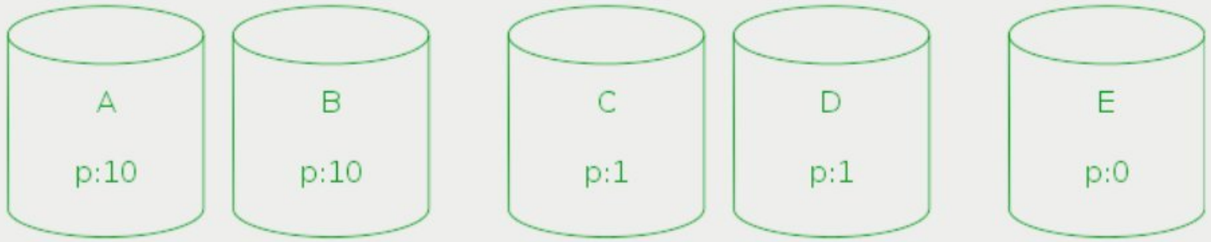
- `rs.conf()`: get current configuration
- `rs.initiate(cfg)`: initiate replica set
- `rs.reconfig(cfg)`: reconfigure a replica set
- `rs.add("hostname:port")`: add a new member
- `rs.addArb("hostname:port")`: add a new arbiter
- `rs.remove("hostname:port")`: remove a member

Priorities

Priorities

- Priority, floating point number between 0 and 100
- Used during an election:
  - Most up to date
  - Highest priority
  - Less than 10s behind failed Primary
- Allows weighting of members during failover

Priorities



- Members A or B will be chosen first
  - Highest priority
- Members C or D will be chosen when:
  - A and B are unavailable, or not up to date
- Member E is never chosen
  - `priority: 0` means it cannot be elected

Write concerns

```
<?php
$manager = new MongoDB\Driver\Manager('mongodb://localhost:27017');

$bulk = new MongoDB\Driver\BulkWrite();
$bulk->insert(['_id' => 1, 'x' => 1]);

$result = $manager->executeBulkWrite( 'db.collection', $bulk, $wc );
```

- Two nodes:

```
$wc = new MongoDB\Driver\WriteConcern( 2 );
```

- Majority of nodes:

```
$wc = new MongoDB\Driver\WriteConcern( MongoDB\Driver\WriteConcern::MAJORITY  );
```

- With timeout:

```
$wc = new MongoDB\Driver\WriteConcern(
    MongoDB\Driver\WriteConcern::MAJORITY, 10000
);
```

Read preferences

Select between candidate servers from a specific set

- primary (default)
- primary_preferred (`RP_PRIMARY_PREFERRED`)
- secondary
- secondary_preferred
- nearest

- Only the candidates within 15ms ping time from the fastest are used
- Random (per-query) among the matched sets

Read preferences

```
<?php
$m = new \Mongo\Driver\Manager(
    'mongodb://localhost:13000/?replicaSet=poiset' .
    '&readPreference=nearest&readPreferenceTags=dc:asia'
);

$m->executeQuery( 'demo.test', $query );
?>
```

or per-query:

```
<?php
$m = new \Mongo\Driver\Manager(
    'mongodb://localhost:13000/?replicaSet=poiset'
);

$rp = new MongoDB\Driver\ReadPreference(
    MongoDB\Driver\ReadPreference::RP_NEAREST, [ [ 'dc' => 'asia' ] ]
);

$m->executeQuery( 'demo.test', $query, $rp );
?>
```

Eventual consistency

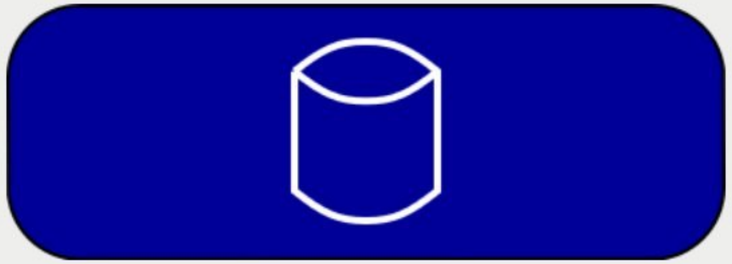
- Read Preferences (except `RP_PRIMARY`)
  - Driver will always send writes to primary
  - Driver will send read requests to secondaries
  - Options to prefer primary, secondary, or nearest
- Warning!
  - Secondaries may be out of date
  - Not appropriate for all applications

Replica Set Wrap-up

- Reads from Primary are always consistent
- Reads from Secondaries are eventually consistent
- Automatic failover if a Primary fails
- Automatic recovery when a node joins the set
- Full control of where writes occur
- Full control of where reads occur

High Availability Scenarios

A single node



- Will have downtime
- Human intervention required if node crashes

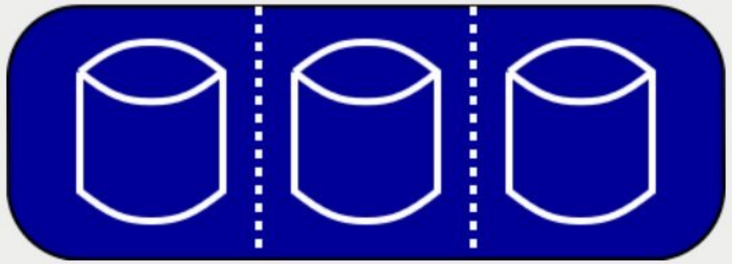
Replica Set (single dc)

- Single datacentre
- Single switch and power
- One node failure
- Automatic recovery of single node crash
- Points of failure:
  - Power
  - Network
  - Datacentre

Replica Set (multiple zones, with arbiter)

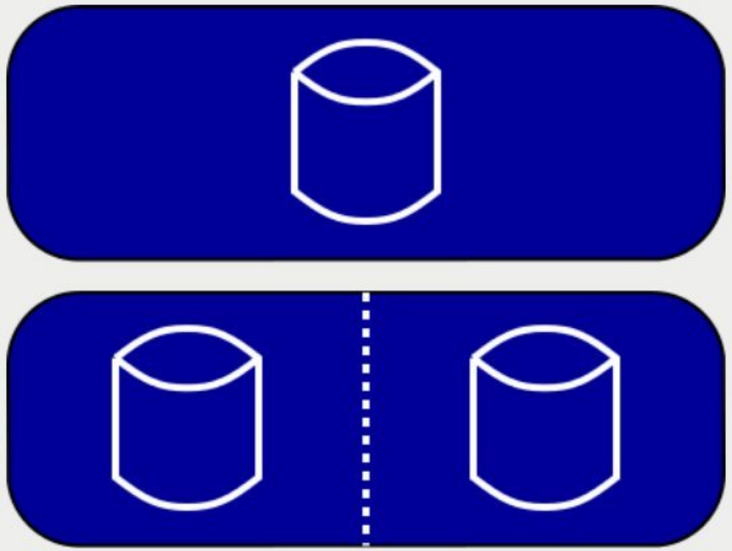
- Single datacentre
- Multiple power/network zones
- Automatic recovery of single node crash
- `w=majority` not viable as no writes with losing a node
- Points of failure:
  - Datacentre
  - Two node failure

Replica Set (multiple zones)



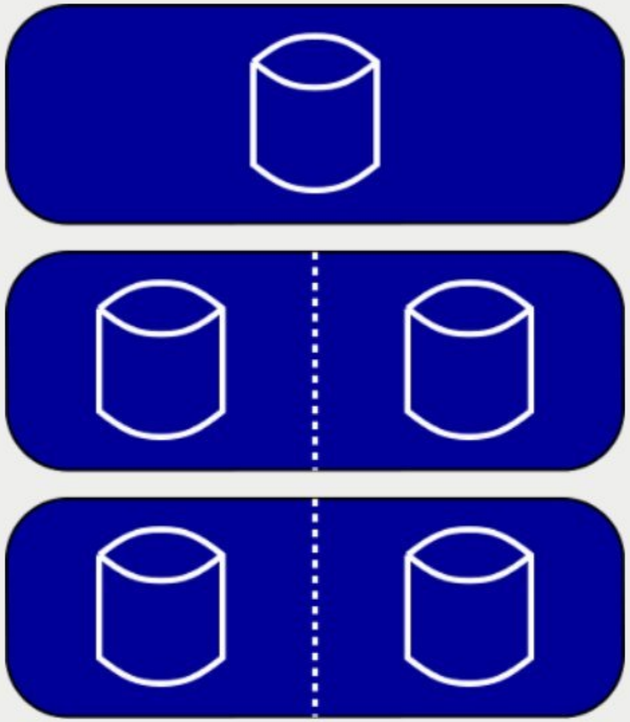
- Single datacentre
- Multiple power/network zones
- Automatic recovery of single node crash
- `w=majority` viable as 2 out of 3 nodes available
- Points of failure:
  - Datacentre
  - Two node failure

Replica Set 4



- Multi datacentre
- DR node for safety
- Can't do multi datacentre durable write safely since only one node in distant datacentre

Replica Set (fully redundant)

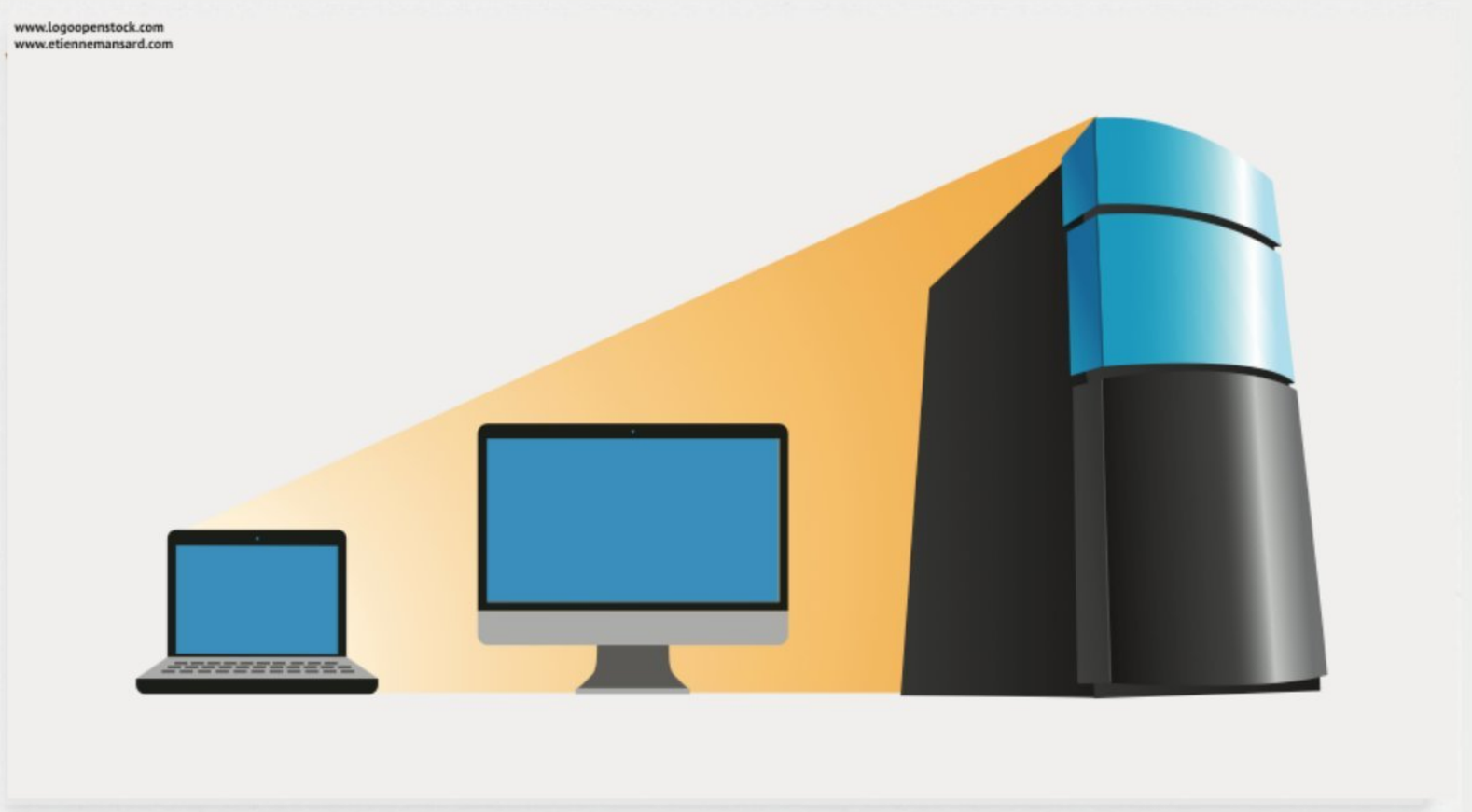


- Three datacentres
- Can survive full datacentre loss
- Can do `w = { dc : 2 }` to guarantee write in two DCs

Typical deployments

Use	Set size	Data protection	High Availability	Notes
	1	No	No	Must use --journal to protect against crashes
✔	2	Yes	No	On loss of one member, set becomes read-only
✔	3	Yes	Yes - 1 failure	On loss of one member, two surviving nodes can elect new primary
	4	Yes	Yes - 1 failure	On loss of two members, two surviving nodes are read-only
✔	5	Yes	Yes - 2 failures	On loss of two members, three surviving nodes can elect new primary

Sharding



1970 - 2000: Vertical Scalability (scale up)

www.etiennemansard.com

Google, ~2000: Horizontal Scalability (scale out)

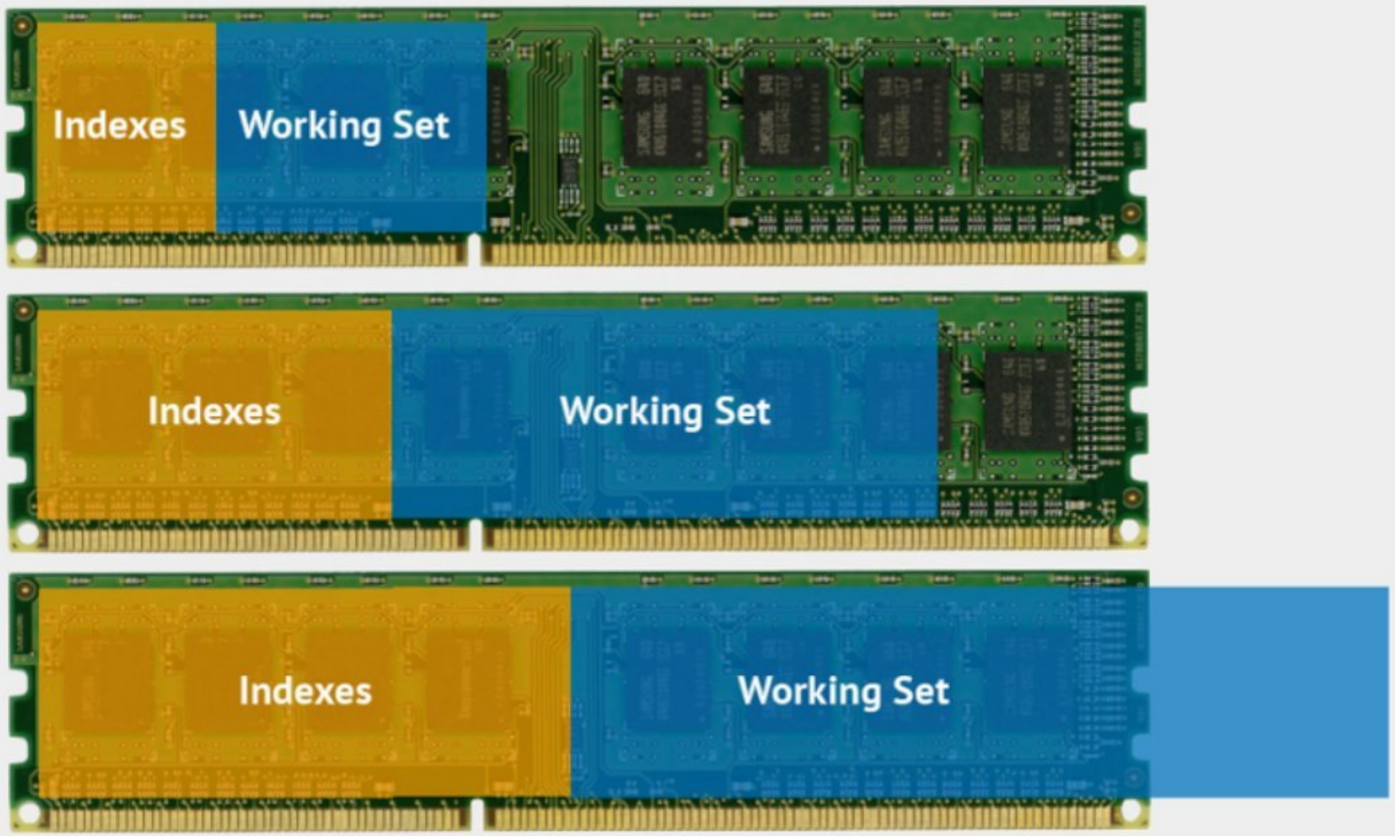
Data Store Scalability in 2005

- Custom Hardware
  - Oracle
- Custom Software
  - Facebook + MySQL

Data Store Scalability Today

- MongoDB auto-sharding available
- A data store that is
  - Free and publicly available
  - Open source (https://github.com/mongodb/mongo)
  - Horizontally scalable
  - Application independent

Why shard?



Working Set Exceeds Physical Memory

Read/Write Throughput Exceeds I/O

MongoDB's approach to sharding

Partition data based on ranges

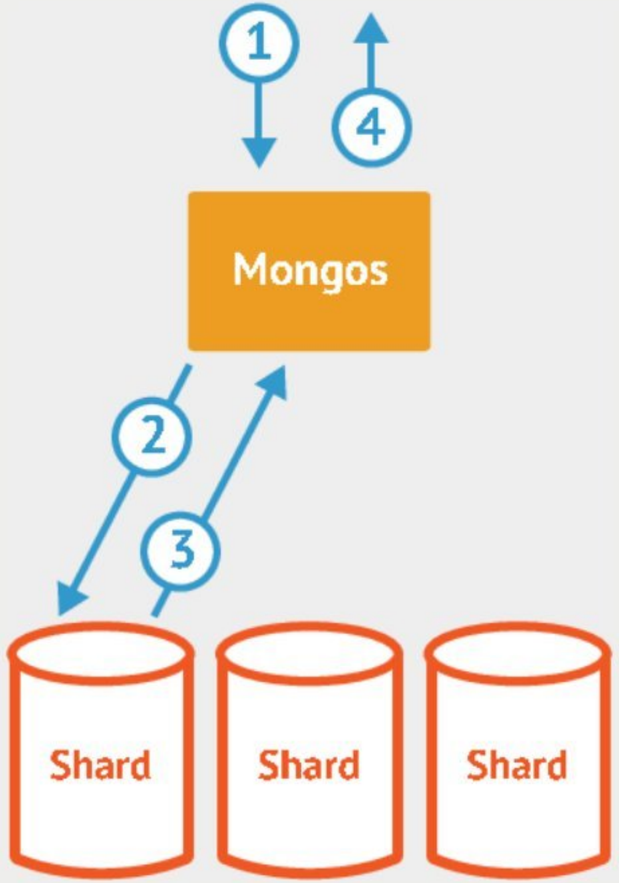
- User defines shard key
- Shard key defines range of data
- Key space is like points on a line
- Range is a segment of that line

Distribute data in chunks across nodes

- Initially one chunk
- Default max chunk size: **64mb**
- MongoDB automatically splits & migrates chunks when max reached

MongoDB manages data

- Queries routed to specific shards
- MongoDB balances cluster
- MongoDB migrates data to new nodes



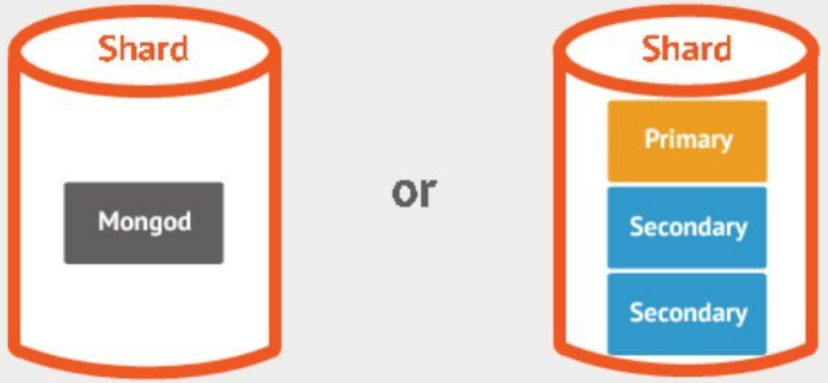
MongoDB Auto-Sharding

- Minimal effort required
  - Same interface as single mongod
- Two steps
  - Enable sharding for a database
  - Shard collection(s) within database

Architecture

Data stored in a shard

- Shard is a node of the cluster
- Shard can be a single mongod or a replica set



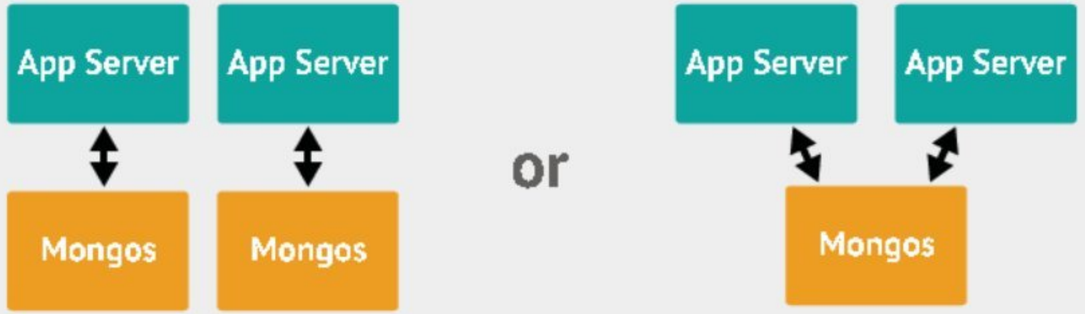
Config server stores meta data

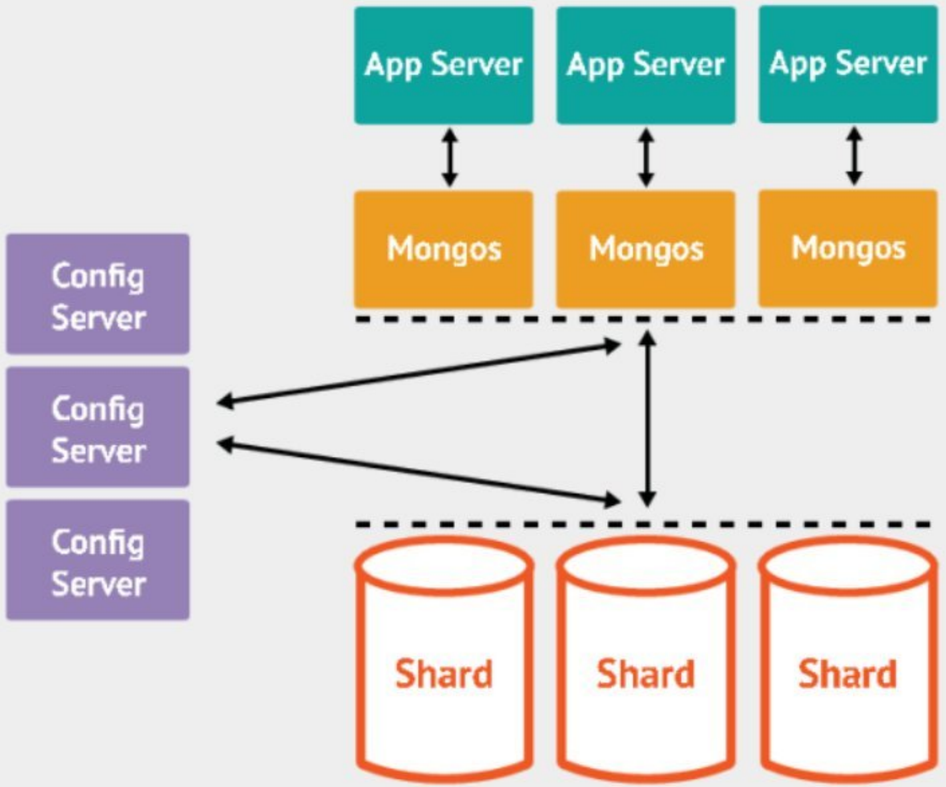
- Config server
  - Stores cluster chunk ranges and locations
  - A Replica Set
  - No arbiters or delayed members

MongoS manages the data

Mongos

- Acts as a router / balancer
- No local data (persists to config database)
- Can have one or many (e.g. each application server)





Sharding Infrastructure

Shard Key

Shard Key

- Shard key is **immutable**
- Shard key values are **immutable**
- Shard key requires index on fields contained in key
- Uniqueness of `_id` field is only guaranteed within individual shard
- Shard key limited to 512 bytes in size

Shard Key Considerations

- Cardinality
- Write distribution
- Query isolation
- Data Distribution

Example: email storage

```
{
        _id: ObjectId("51156a11056d6f966f268f7f"),
        user: 37113,
        time: ISODate( '2013-04-24 10:38:05 '),
        subject: "This is a test",
        recipients: [ "derick@10gen.com", "derick@example.com" ],
        body: "..."
}
```

- Lots of emails per user
- Most common query: get user emails sorted by time
- Index on `{ _id: 1 }`, `{ user: 1, time: -1 }`, and `{ recipients: 1 }`

Example: email storage

	Cardinality	Write scaling	Query isolation	Index Locality
_id	Doc level	1 shard	All shards, merge sort	Great
hash(_id)	Hash level	All shards	All shards, merge sort	Poor
user	Many docs	All shards	One shard, index sort	So-so
user,time	Doc level	All shards	One shard, index sort	Good

```
{
        _id: ObjectId("51156a11056d6f966f268f7f"),
        user: 37113,
        time: ISODate( '2013-04-24 10:38:05 '),
        subject: "This is a test",
        recipients: [ "derick@10gen.com", "derick@example.com" ],
        body: "..."
}
```

Connection String

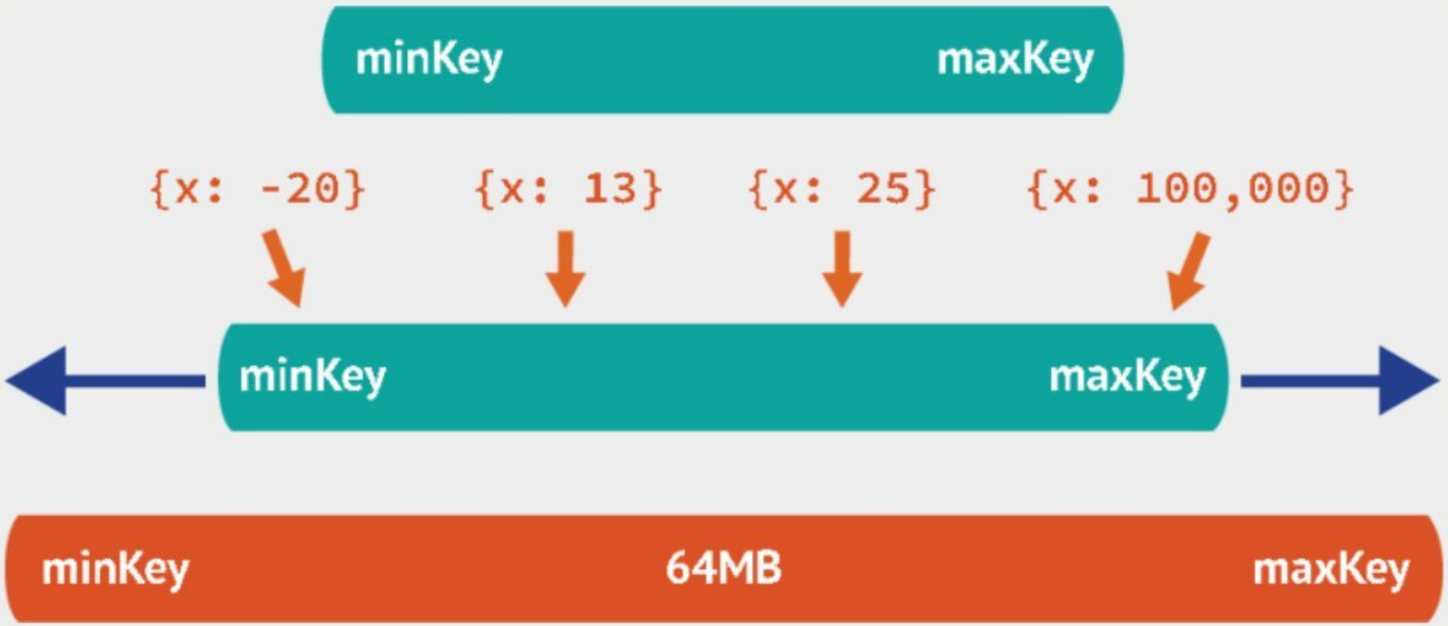
```
<?php
$m = new \MongoDB\Client( 'mongodb://localhost:27019' );

$m = new \MongoDB\Client( 'mongodb://localhost:27019,example.com:27019' );

$m = new \MongoDB\Client( 'mongodb://user:password@localhost:27019/demo', $options );

$m = new \MongoDB\Client( 'mongodb:///tmp/mysocket.sock' );
?>
```

- Remember: mongos may be run on the app server
- Add multiple hosts for redundancy

Mechanics

Partitioning

- Remember: it's based on ranges

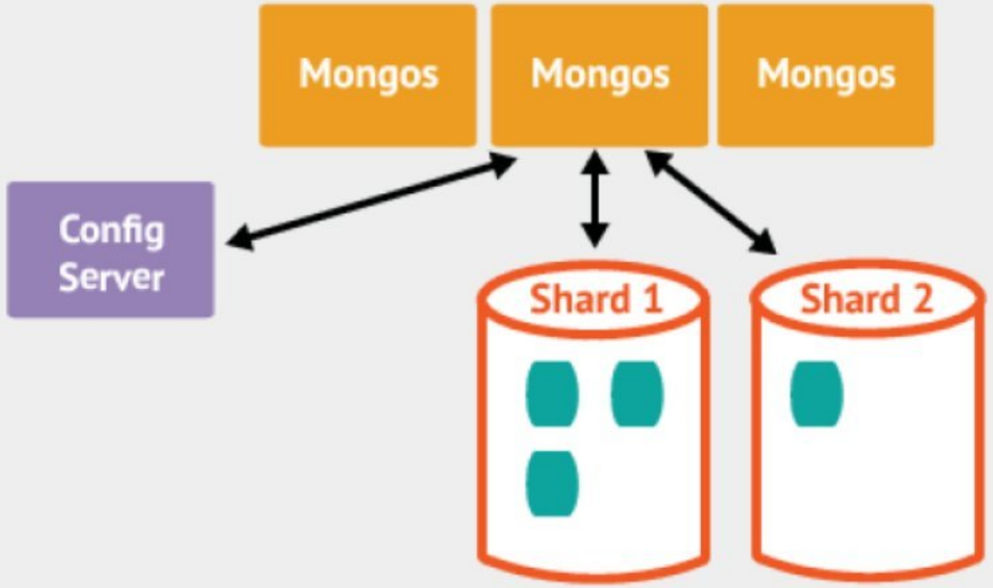


Chunk is a section of an entire range

Chunk splitting

- A chunk is split once it exceeds the maximum size
- There is no split point if all documents have the same shard key
- Chunk split is a logical operation (no data is moved)
- If splitting creates too large a difference in chunks across the cluster, a balancing round starts

Balancing



- Balancer is running on mongos
- Once the difference in chunks between the most dense shard and the least dense shard is above the migration threshold, a balancing round starts

Balancing tips

- Run the balancer during low traffic periods:

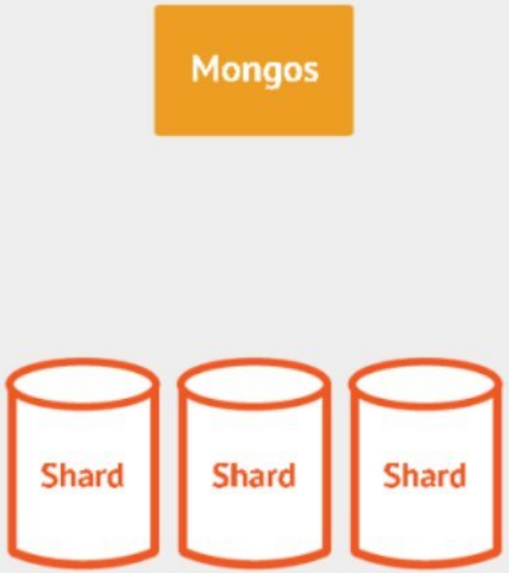
```
use config;

db.settings.update(
        { _id: 'balancer' },
        { $set: { activeWindow: { start: "23:00", stop: "4:00" } } }
);
```

- Can be triggered manually using `moveChunk`
- Pre-creating ranges for heavy insertion loads can avoid over-populating a single shard
- Feel free to write your own smart balancer!

Routing

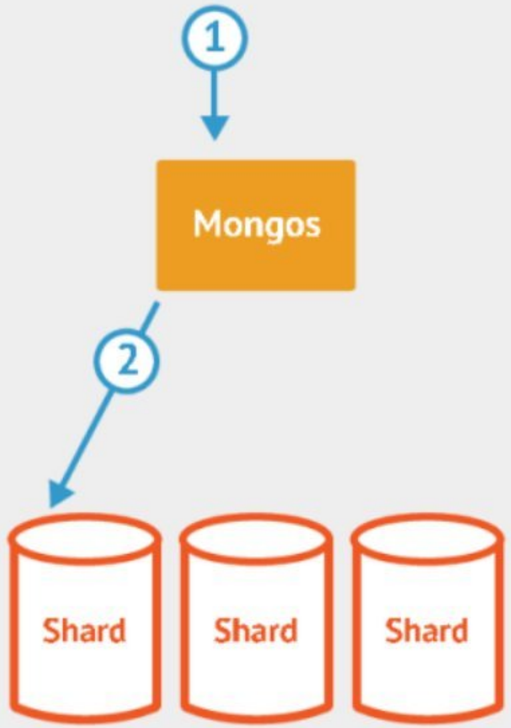
Cluster Request Routing

- Targeted Queries
- Scatter / Gather Queries
- Scatter / Gather Queries with Sort

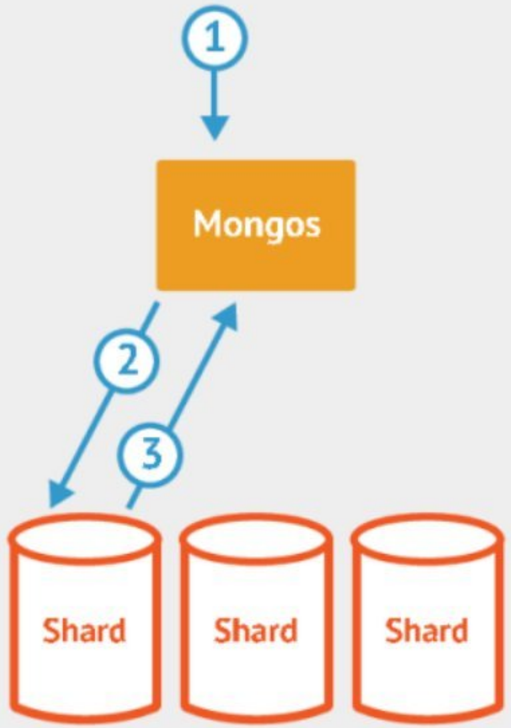


Cluster Request Routing: Targeted Query

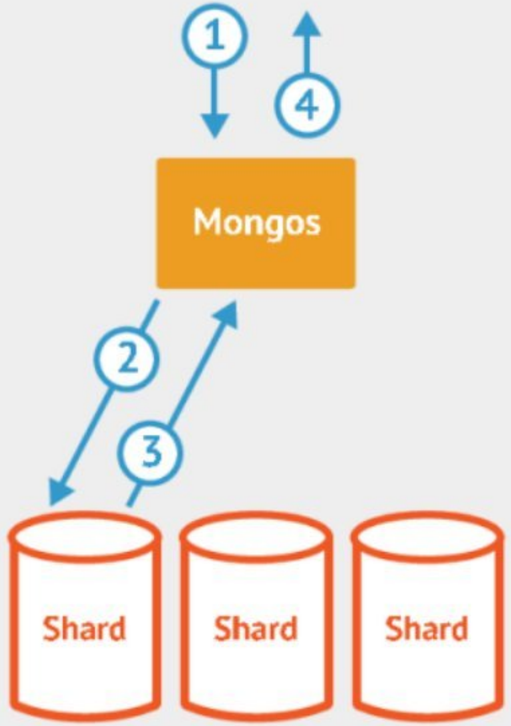
Routable request received



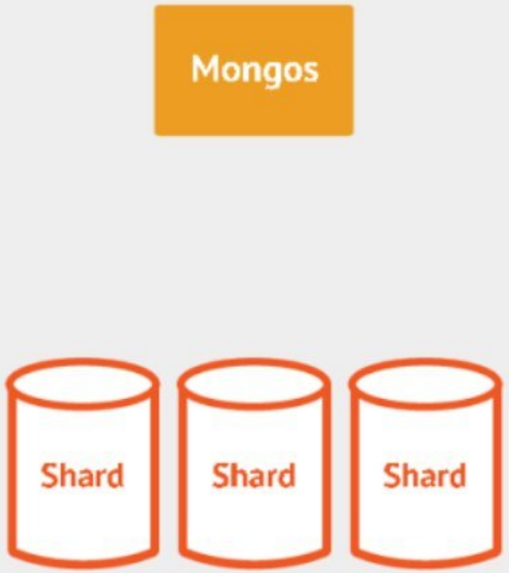
Request routed to appropriate shard



Shard returns results

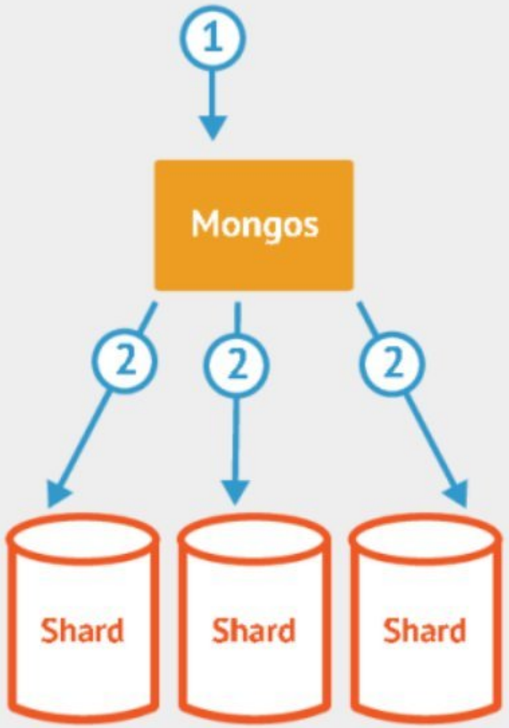


Mongos returns results to client

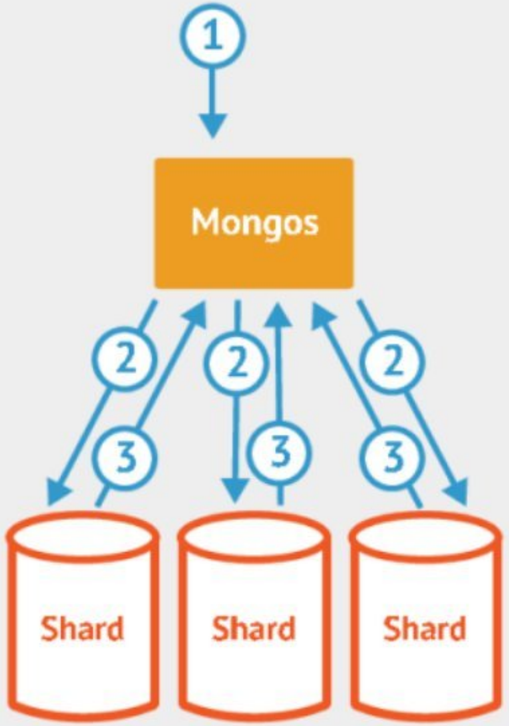


Cluster Request Routing: Scatter / Gather Query

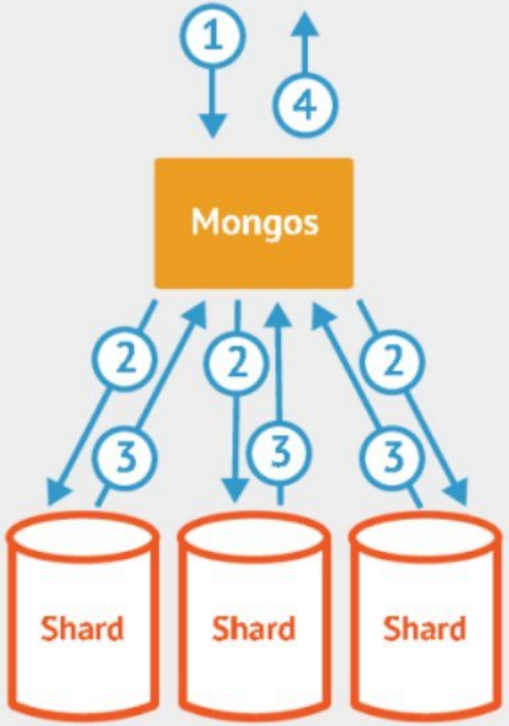
Scatter / Gather Request Received



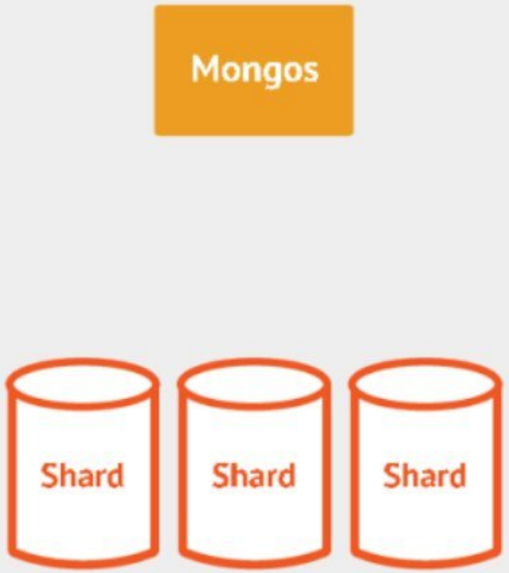
Request sent to all shards



Shards return results to mongos



Mongos returns results to client



Cluster Request Routing: Scatter / Gather Query with Sort

Scatter / Gather Request with Sort Received