Sharding

Contents

- What is Sharding and why it exists
- Concepts of Sharding
- How sharding works
- Common problems
- Internals of sharding
- Misc

What is Sharding?

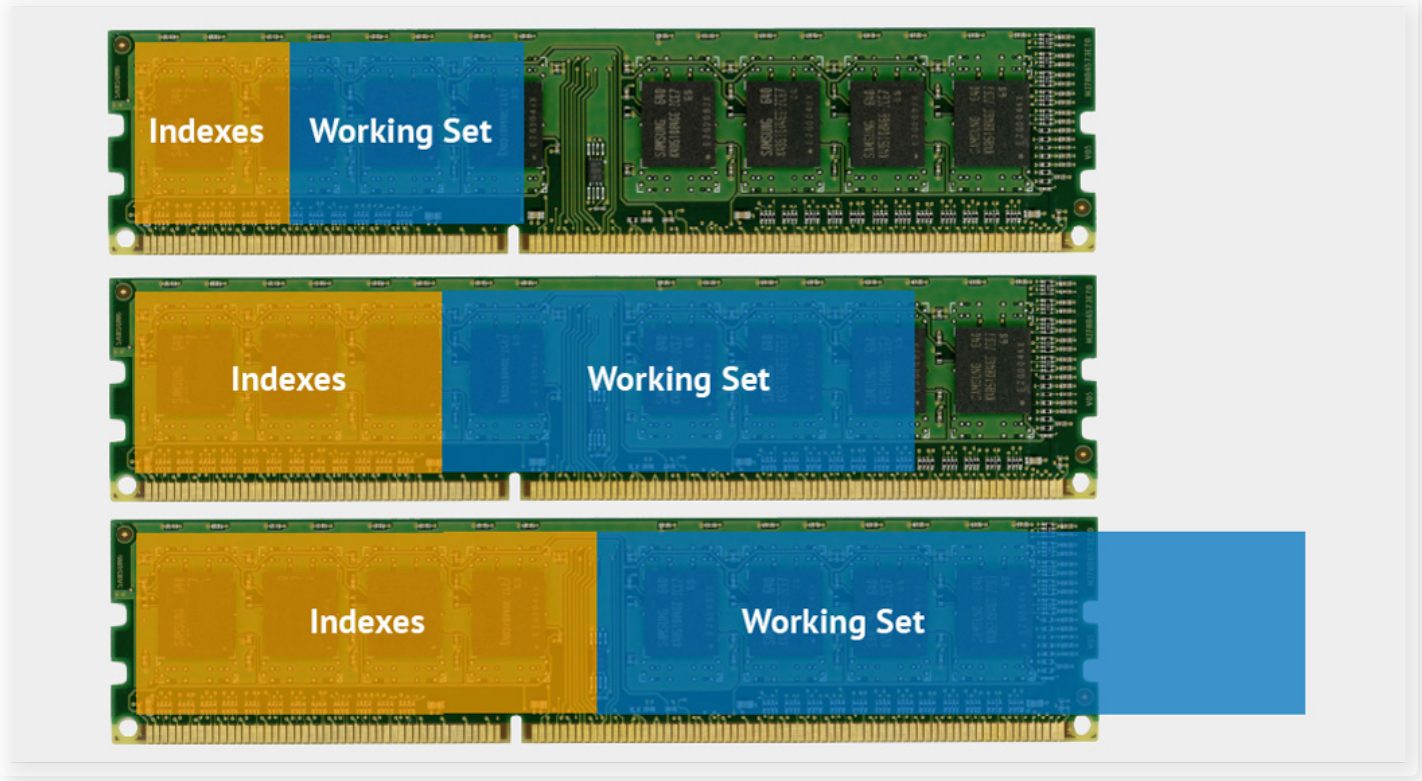
- Scaling out

Why Sharding (cont)?

- Read/Write Throughput exceeds capacity

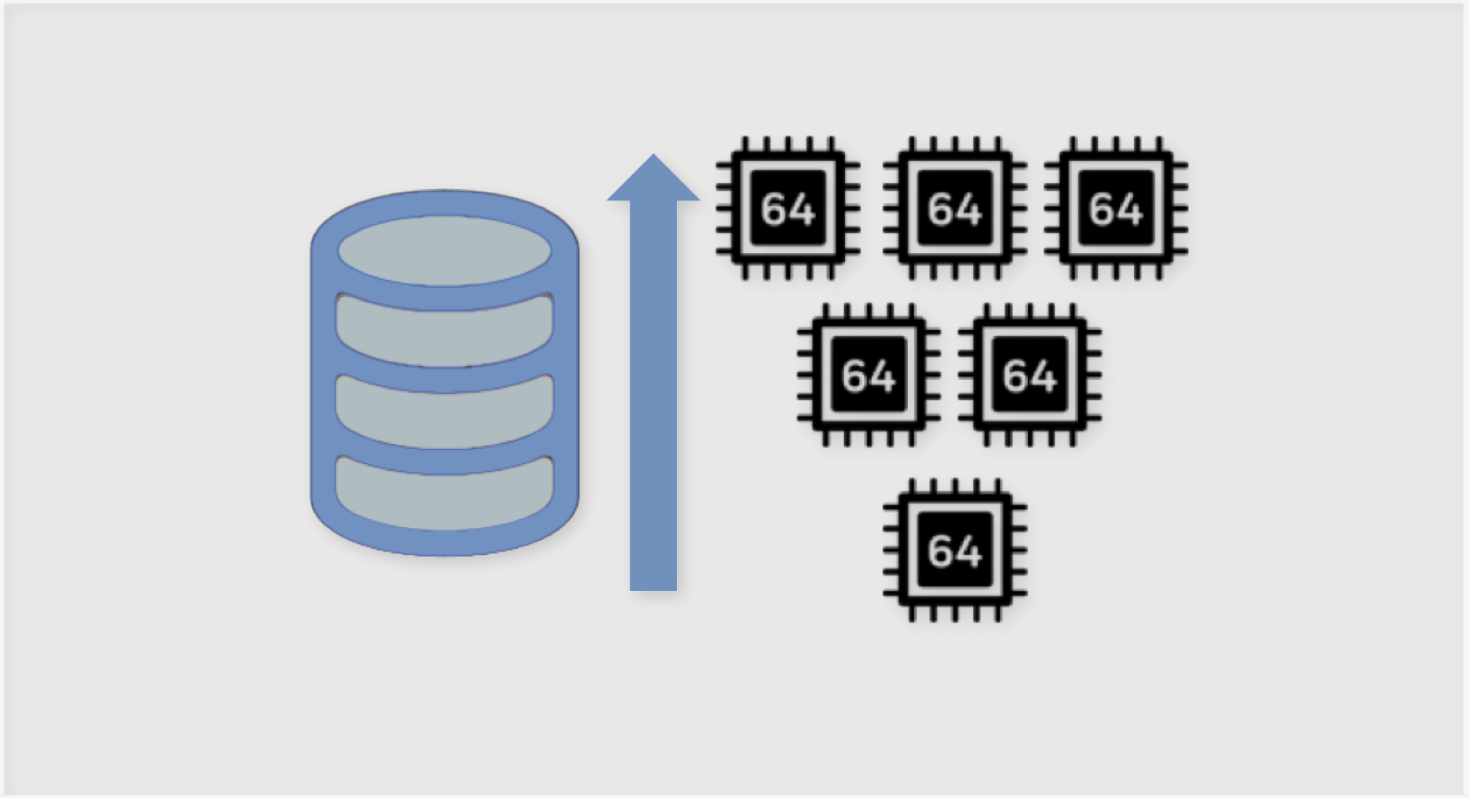
Why Sharding?

- Data set is becoming too big

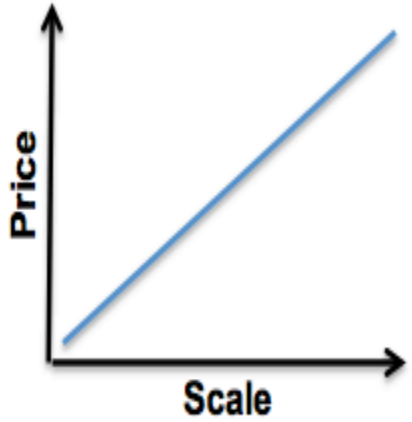


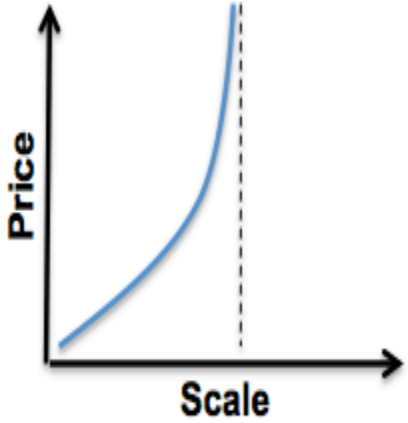
Why Sharding (cont)?

- Can’t scale up anymore



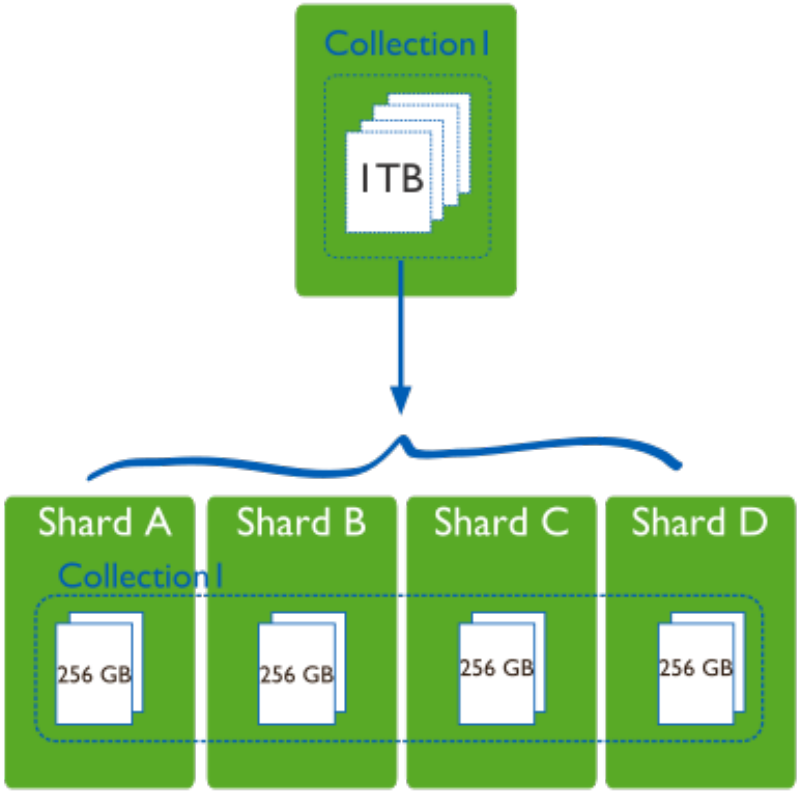
Vs.





Concept: dividing the database

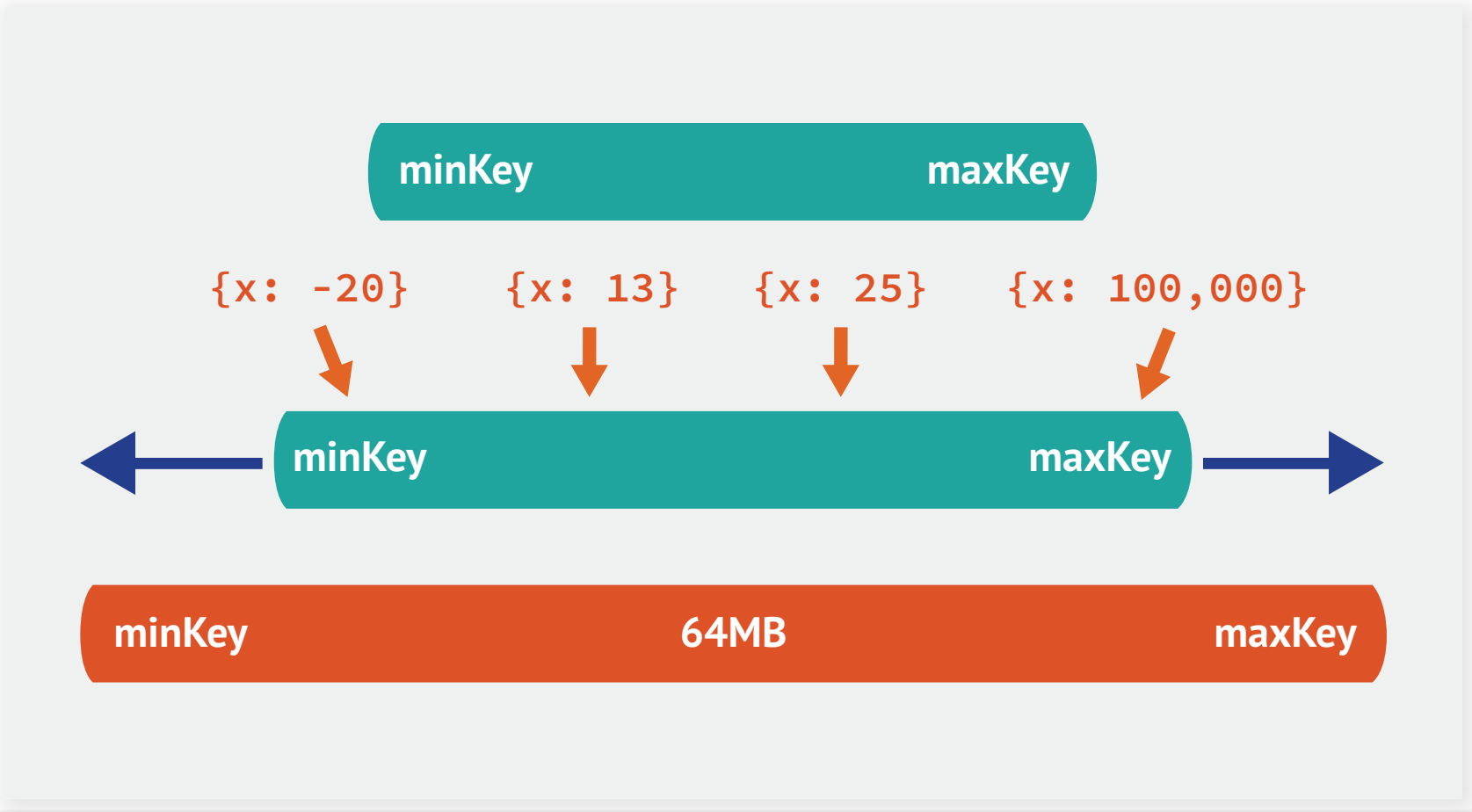
- Each mongod process manage a subset of the data



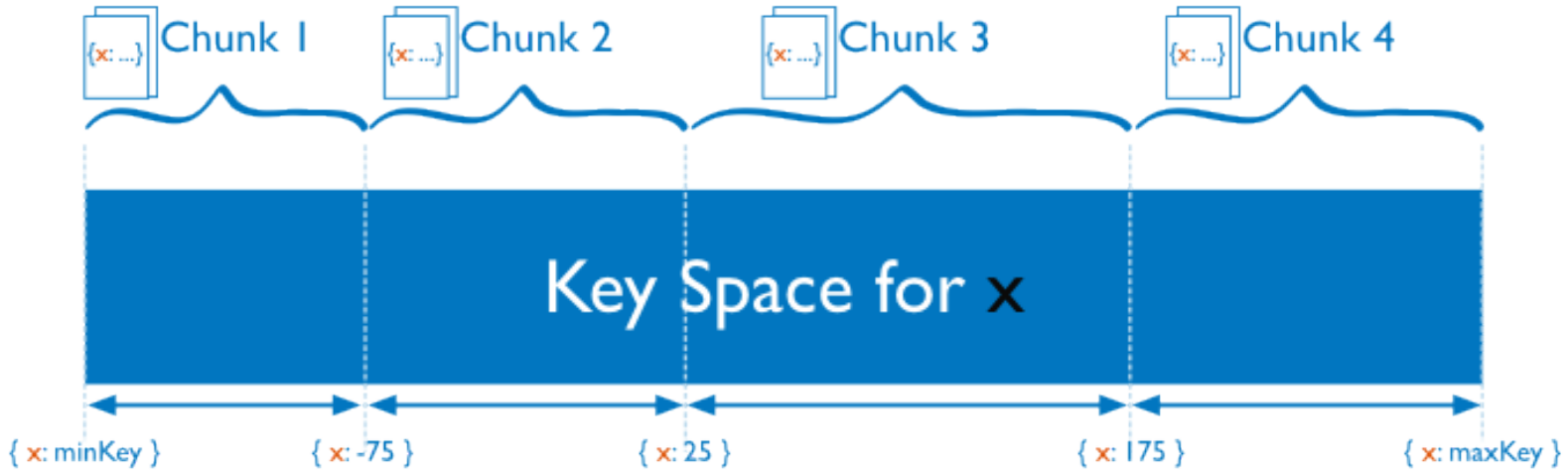
Concept: shard key

- User defines shard key
  - Shard key can be multiple fields, like an index
- Shard key defines range of data
- Key space is like points on a line
- Range is a segment of that line

Concept: chunk

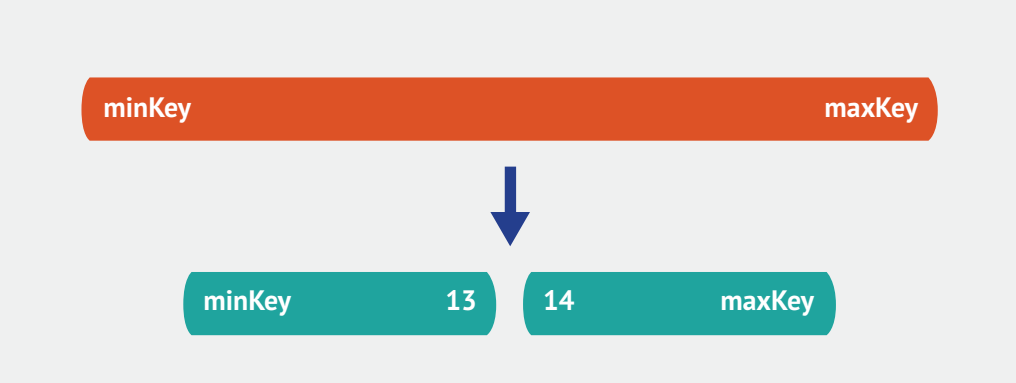


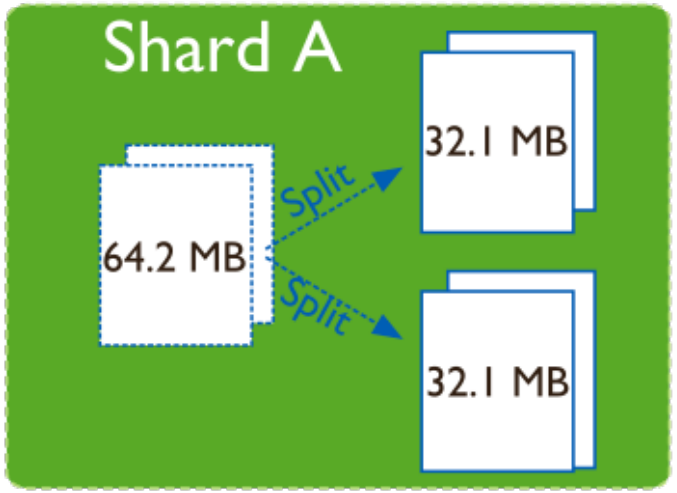
Concept: chunk



Concept: splitting a chunk

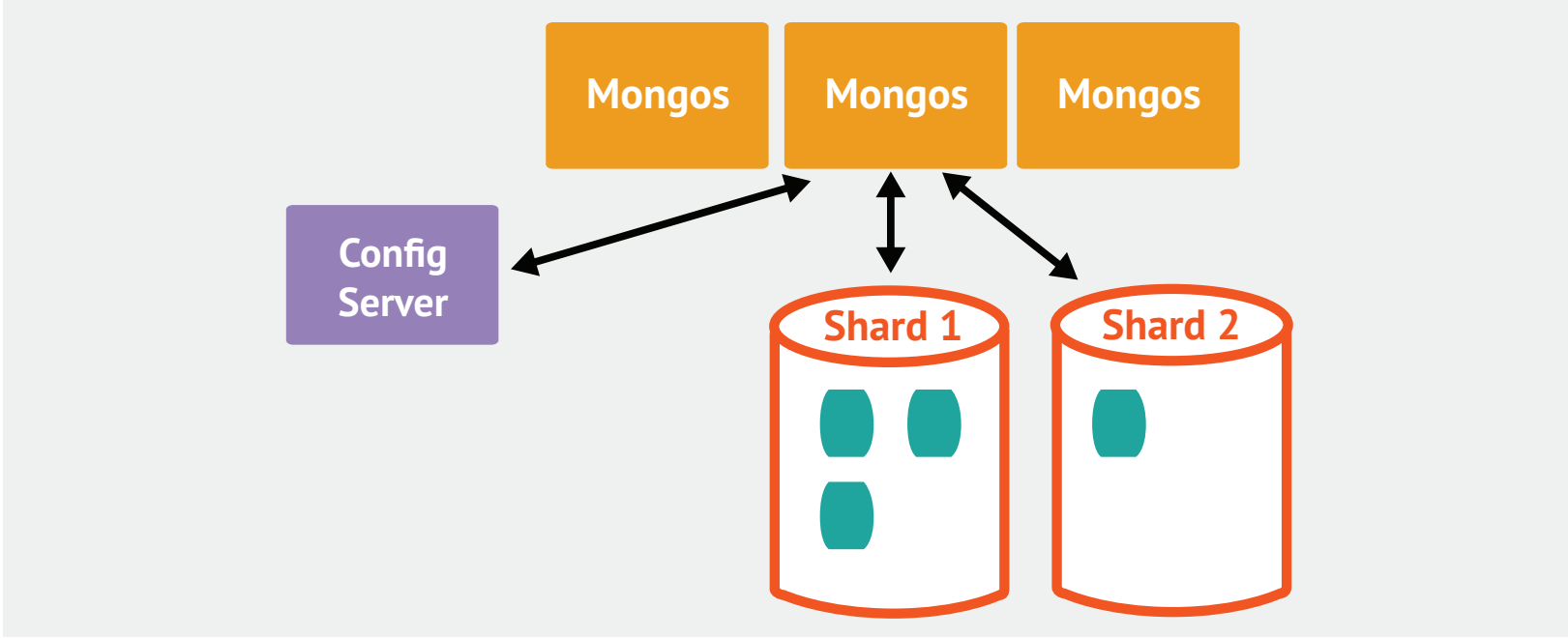
- A chunk is split once it exceeds the maximum size
- There is no split point if all documents have the same shard key
- Chunk split is a logical operation (no data is moved)



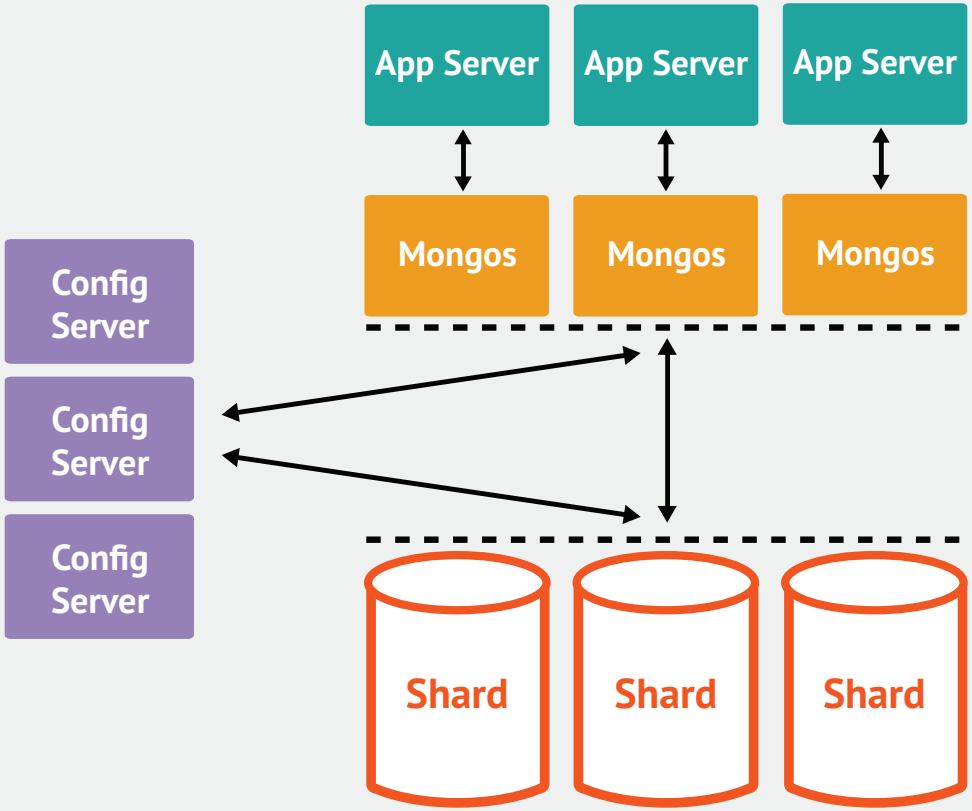


Concept: balancing chunks

- 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



Static Picture of Sharding



The General Goals

- What may be the goals to achieve

The General Goals

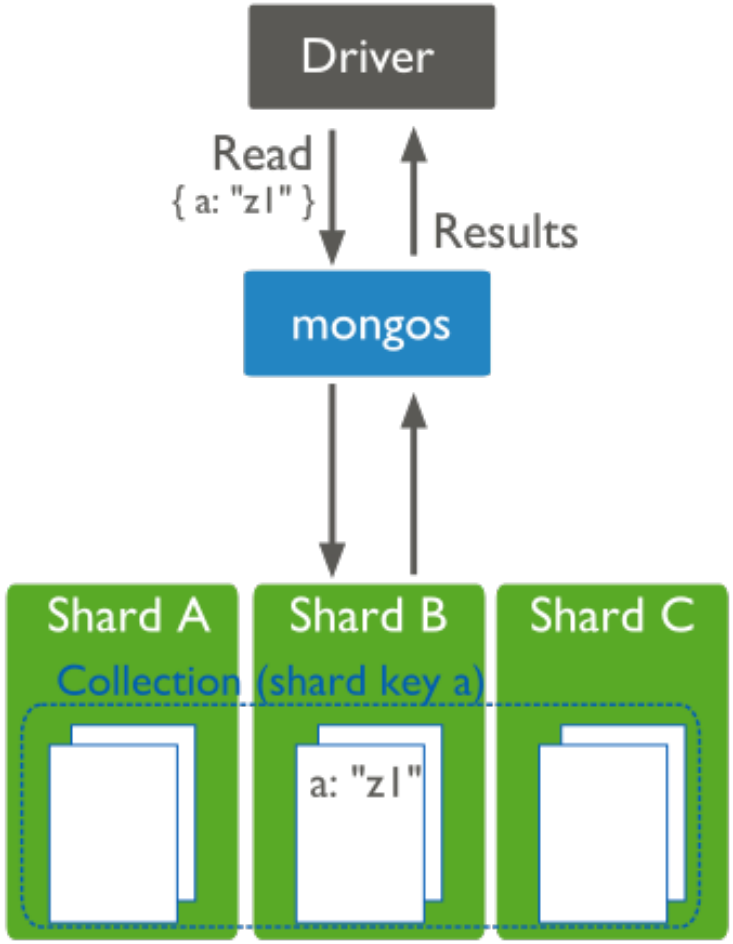
- Even distribution of data
  - We balance chunks, not size, neither documents
- Distribution by geographical region
  - Tagging (tag aware sharding), examples:
    - Location
    - Date/Time
    - Premier Tiers
- Distribution by age, or other criteria

Dynamic Picture...

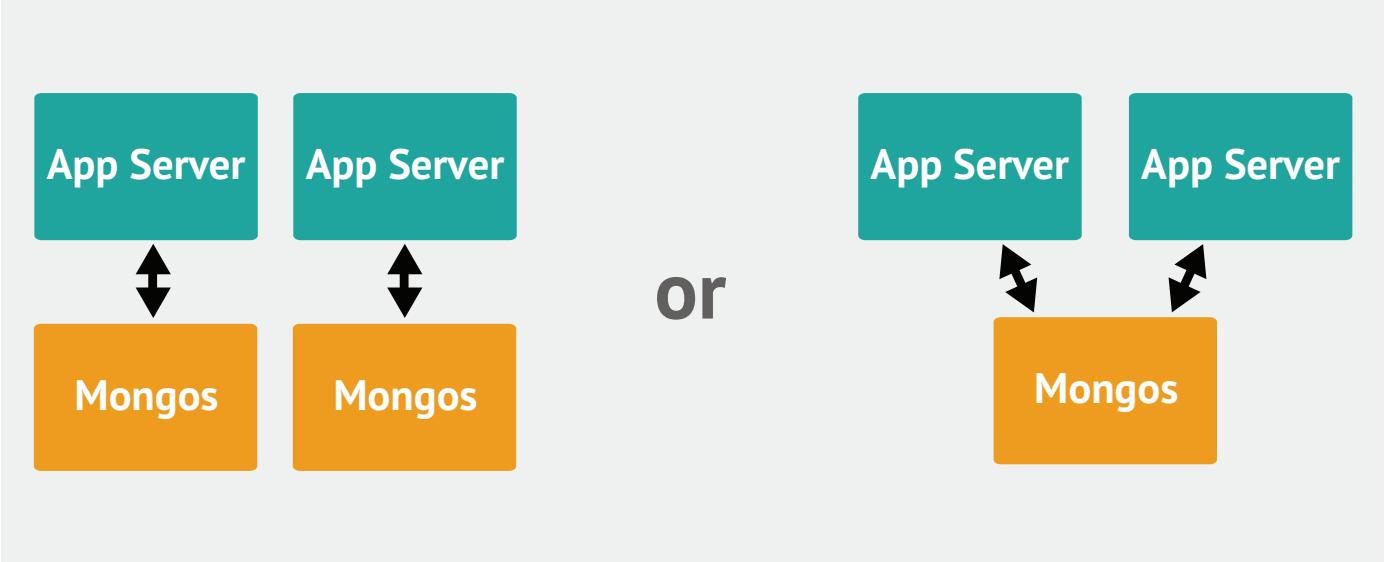
- How is it working?

Dynamic picture: routing

- Targeted queries
- Scatter Gather queries
- … with sort



Dynamic picture: serving application



Common issues

A. Shard Key
   - Bad shard key
     - Monotonically increasing
       - for example, client side generated NumberLong key {_id:1}…

B. Config Server
   - Out of sync

C. Balancing
   - Not happening

A. Shard Key

Selecting a Shard Key

- Cardinality
- Write Distribution
- Read Distribution
- Read Targeting
- Read Locality

- https://www.mongodb.com/blog/post/on-selecting-a-shard-key-for-mongodb

With a …

Good Shard Key

Usually you want…

- Reads hit only 1 or 2 shards per query.
- Writes are distributed across all servers.
- Your disk usage is evenly distributed across shards.
- Things stay this way as you scale.

Bad Shard Key

- Your reads hit every shard.
- Your writes are concentrated on one shard.
- Most of your data is on just a few shards.
- Adding more shards to the cluster will not help.

Hashed Shard Keys

- Monotically increasing keys are distributed evenly
- You can specify the initial number of chunks
- In theory, you can leave the balancer disabled
  - At least turn it off for loading the data
- You can't have range queries

Shard Key characteristics

- Shard key is **immutable**
- Shard key values are **immutable**
- Shard key must be indexed
- Shard key limited to 512 bytes in size
- Shard key used to route queries
  - Choose a field commonly used in queries
- Only shard key can be unique across shards
  - `_id` field is potentially only unique within an individual shard, however duplicate values will create problem for moving chunks.

Shard key example

Email Storage

```
{
  _id: ObjectId(),
  user: 123,
  time: Date(),
  subject: "...",
  recipients: [],
  body: "...",
  attachments: []
}
```

- *Most common scenario, can be applied to 90% cases*
- *Each document can be up to 16MB*
- *Each user may have GBs of storage*
- *Most common query: get user emails sorted by time*
- *Indexes on {_id}, {user, time}, {recipients}*

Shard Key Example (solutions)

	Cardinality	Write Scaling	Query Isolation	Reliability	Index Locality
_id	Doc level	One shard	Scatter/ gather	All users affected	Good
hash(_id)	Hash level	All Shards	Scatter/ gather	All users affected	Poor
user	Many docs	All Shards	Targeted	Some users affected	Good
user, time	Doc level	All Shards	Targeted	Some users affected	Good

B. Config Server

- SCCC – Sync Cluster Connection Config
  - Up to 3.2
  - Likely not available in 3.4
  - Refer to as “mirror style CS”, not “old style”
- CSRS – Config Servers as Replica Set
  - New in 3.2
  - Requires “Protocol Version 1”

Collections in the Config Server

- changelog
- chunks
- collections
- databases
- lockpings
- locks
- mongos
- settings
- shards
- version

Last Ping from a “mongos” in MMS

```
"configLockpings": [
        {   "ping": {"$date": 1424885343297},
            "_id": "osmngbkpswitch02:27017:1424250817:1804289383"},
"configCollections": [
        {
            "noBalance": true,
            "unique": false,
            "dropped": false,
            "_id": "prod_tras_metrics.metric_2015_057",
            "key": {"_id": "hashed"},
            "lastmod": {"$date": 1424852230},
            "lastmodEpoch": {"$oid": "54ed85060be12263545b342a"}
        }…
"configDatabases": [
        {   "partitioned": false,
            "_id": "test_bluewhale_metrics",
            "primary": "osmetcol001"
        }…
"configSettings": [
        { "_id": “chunksize", "value": 64},
        {   "_secondaryThrottle": true,
            "stopped": false,
            "_waitForDelete": true,
            "_id": "balancer",
            "activeWindow": {
                "stop": "4:00",
                "start": "9:00"
            }
        }
    ],
```

Sharding a collection

Pre-splitting

- You may pre-split data before loading data into a sharded cluster.
- Hashed shard key can be initially created with a number of chunks
- Pre-splitting is useful if:
  - You plan to do a large data import early on
  - You expect a heavy initial server load and want to ensure writes are distributed.
- Procedure
  - Create split points
  - Let the balancer distribute the empty chunks
  - Stop the balancer
  - Load the data
  - Start the balancer

Sharding Manually

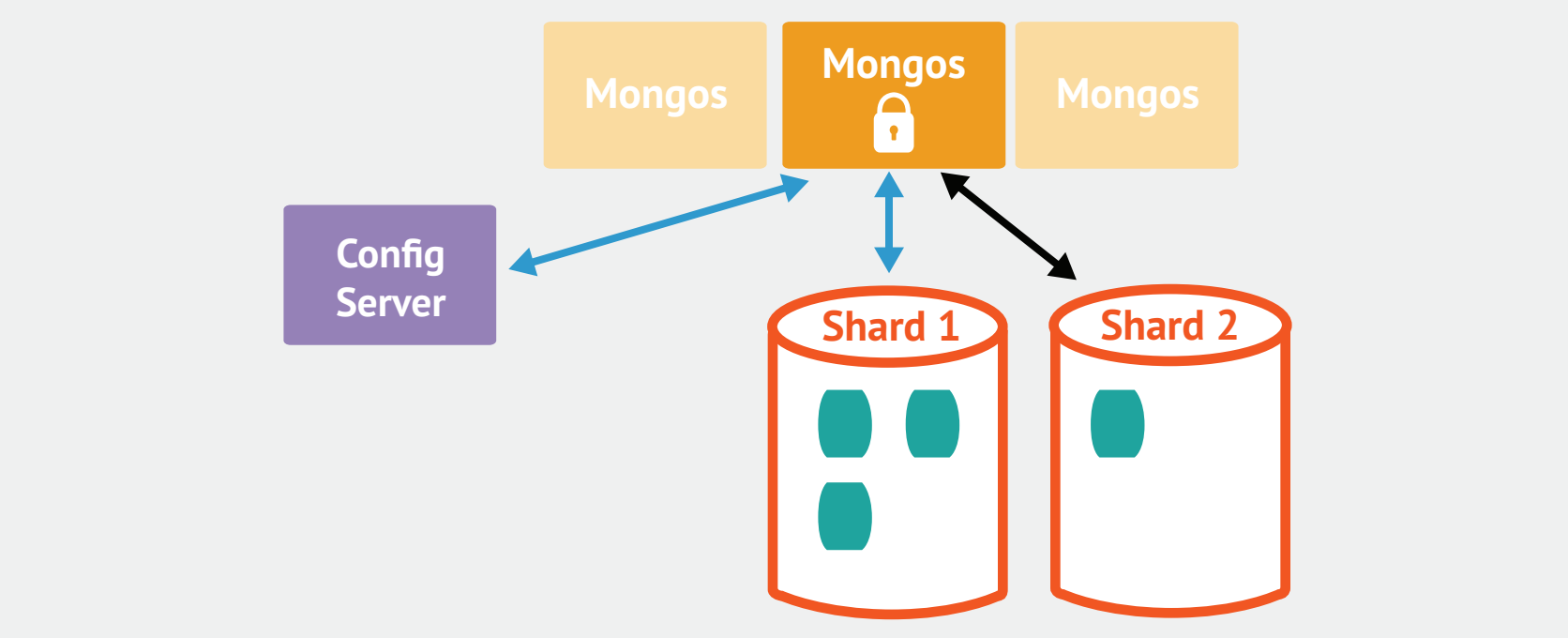
- split*
  - sh.splitAt,() sh.splitFind()
- moveChunk
  - _secondaryThrottle, _waitForDelete

Auto-splitting

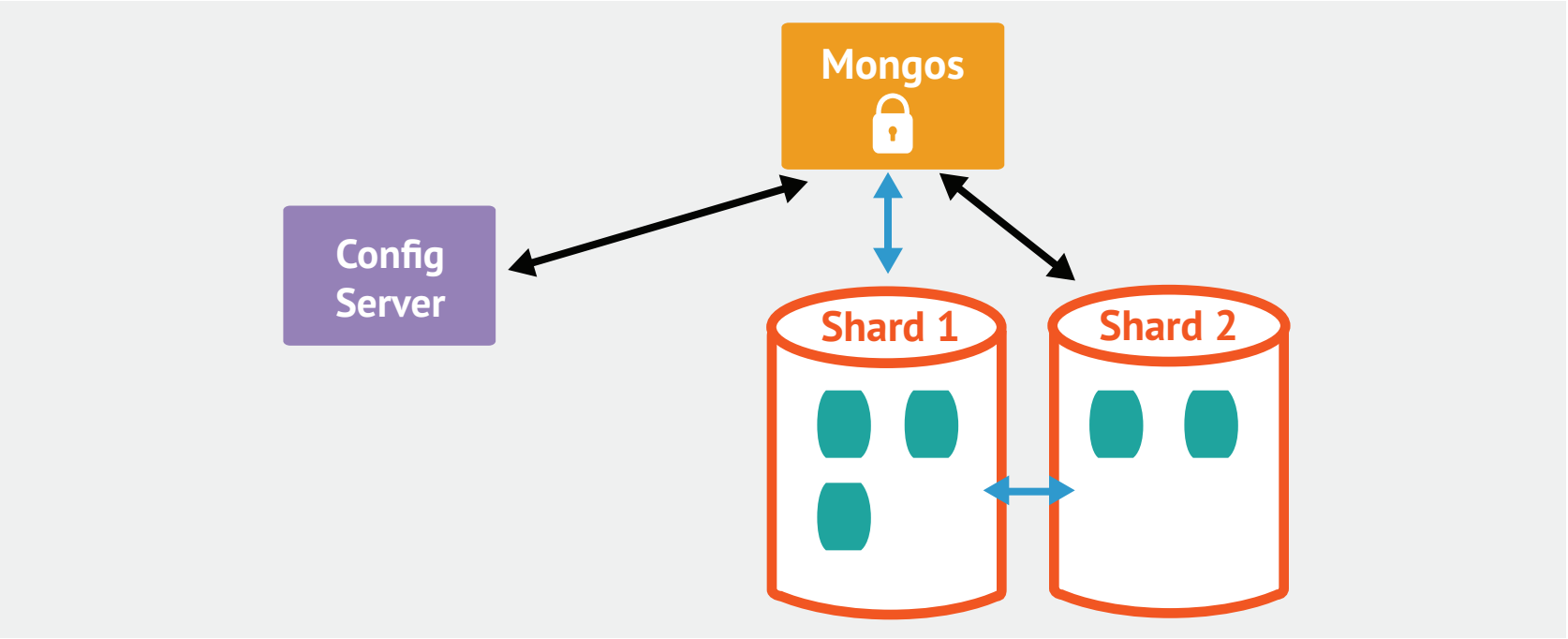
The Balancer

- Wakes up every 30 sec
  - Should I do something? Lock taken?
  - Anything needs attention
    - Drain shard
    - Tag sharding moves
    - Imbalances => balancing round
- Balancing round starts when the imbalance reaches:
  - 2 when the cluster has < 20 chunks
  - 4 when the cluster has 20-79 chunks
  - 8 when the cluster has 80+ chunks

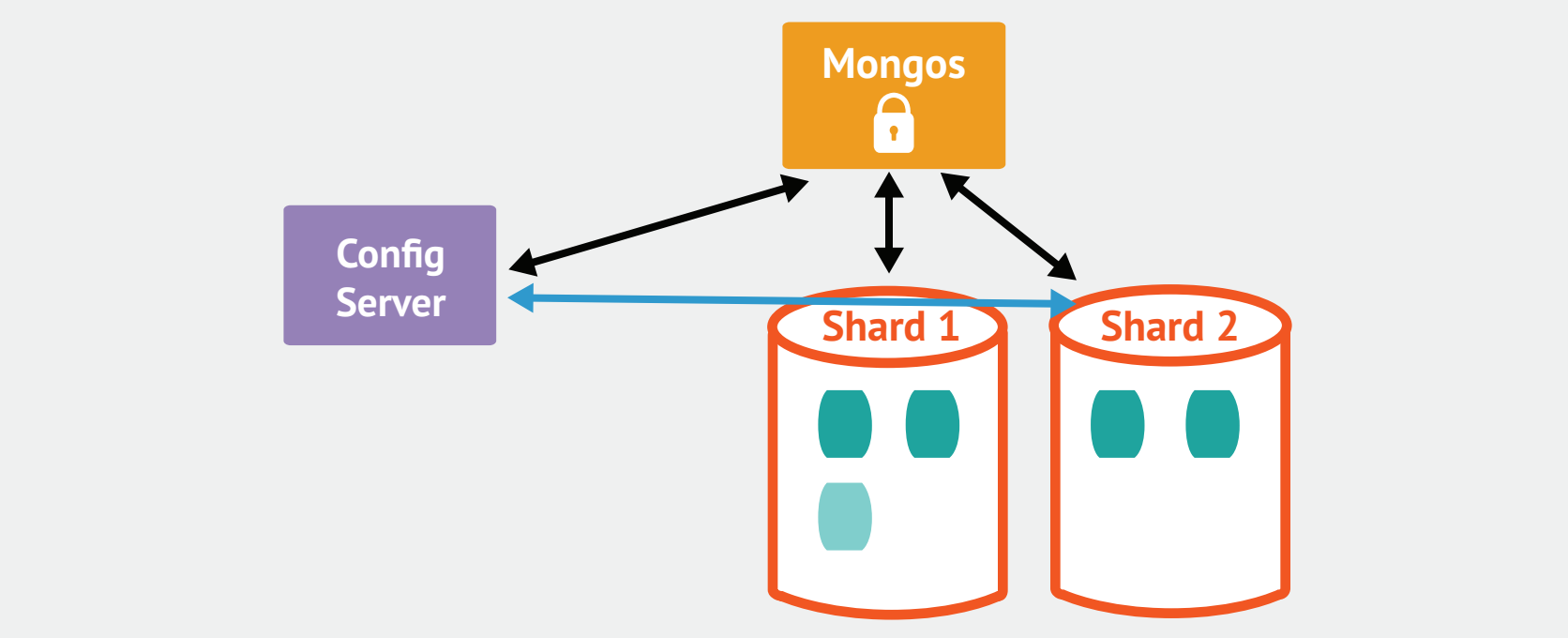
Moving Chunk: acquire the balancer lock



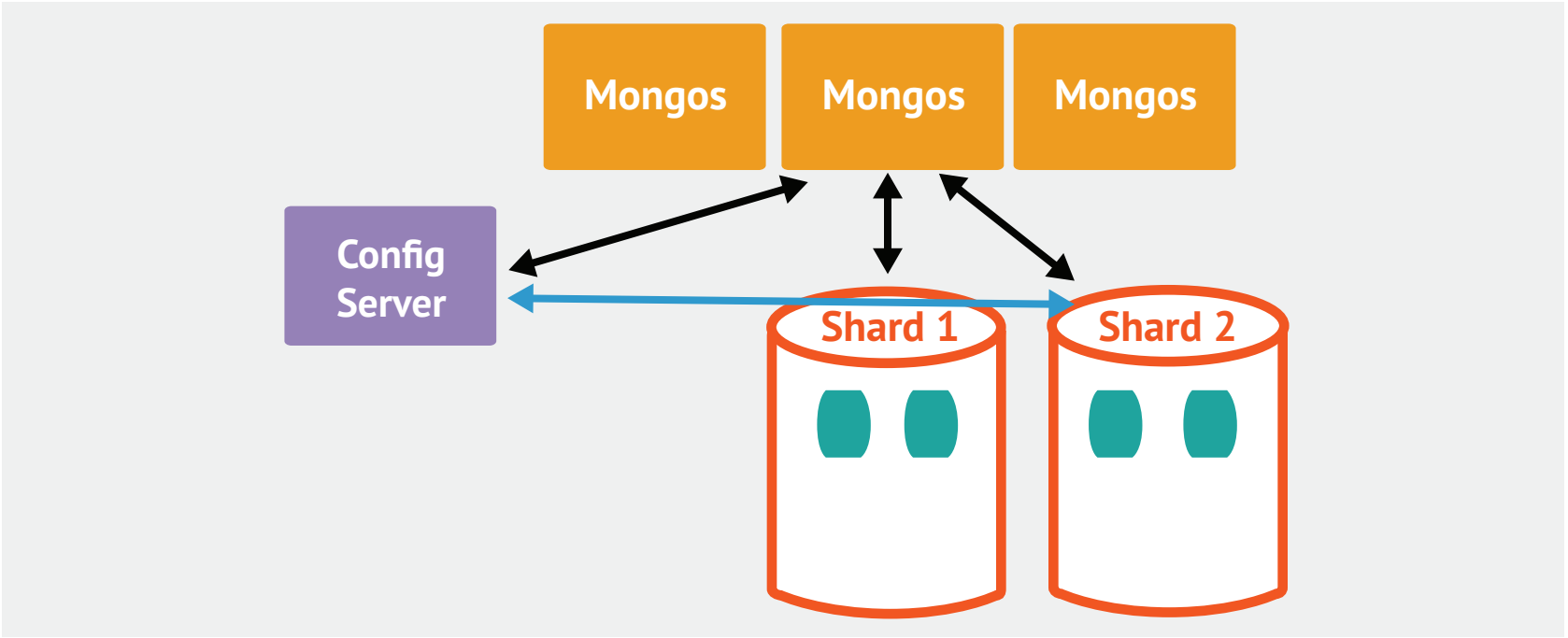
Moving Chunk: moving the chunk



Moving Chunk: committing the migration



Moving Chunk: cleanup



Chunk Migration in plain words

1. The balancer process sends the moveChunk command to the source shard.
2. The source shard continues to process reads/writes for that chunk during the migration.
3. The destination shard requests documents in the chunk and begins receiving copies.
4. After receiving all documents, the destination shard receives any changes to the chunk.
5. Then the destination shard tells the config db that it has the chunk.
6. The destination shard will now handle all reads/writes.
7. The source shard deletes its copy of the chunk.

Chunk Migration in to the code

Mongos, FromShard, ToShard

Get the balancer lock

Identify chunk to move (1-drain, 2-mismatch tag, 3-imbalance)

Send moveChunk command to "FromShard"

- "step 1 of 6" : 0, parse options
- "step 2 of 6" : 8, make sure my view is complete/sanity checks
- "step 3 of 6" : 24, migration/tell ToShard
- "step 4 of 6" : 43241, pause till migrate caught up
  - "step 1 of 5" : 0, copy indexes
  - "step 2 of 5" : 0, delete data already in the range
  - "step 3 of 5" : 42978, initial bulk clone
  - "step 4 of 5" : 0, bulk of modifications
  - "step 5 of 5" : 273, wait for commit
- "step 5 of 6" : 24, critical section
- "step 6 of 6" : 0, wait for cursors to expire and delete data

Reload the updated metadata in the config server

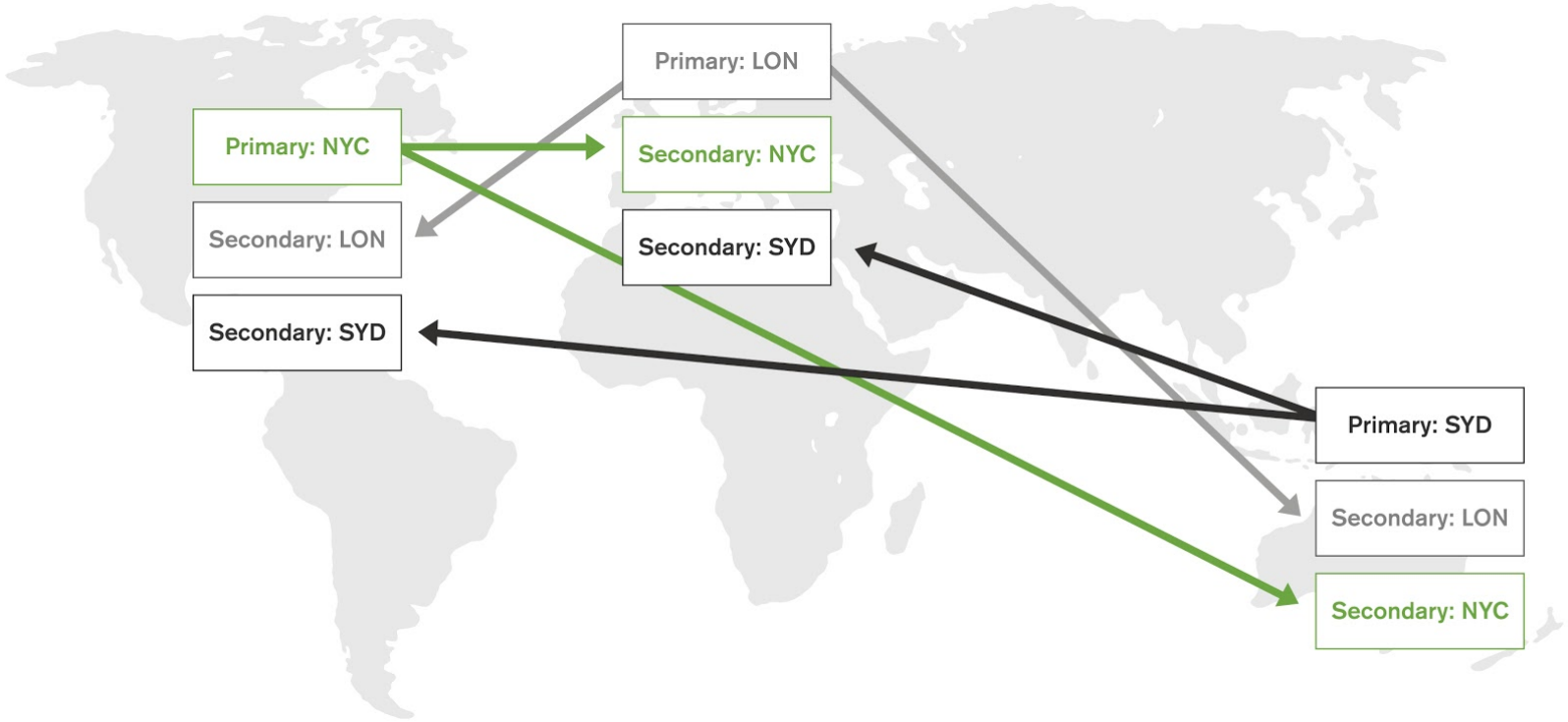
Misc

- Jumbo flag
- Unique keys
- GridFS
- Tag Aware Sharding

Active/Active Data Center

(tolerates server, rack, data center failures, network partitions)

Geo Tag Aware Sharding



Tag aware sharding

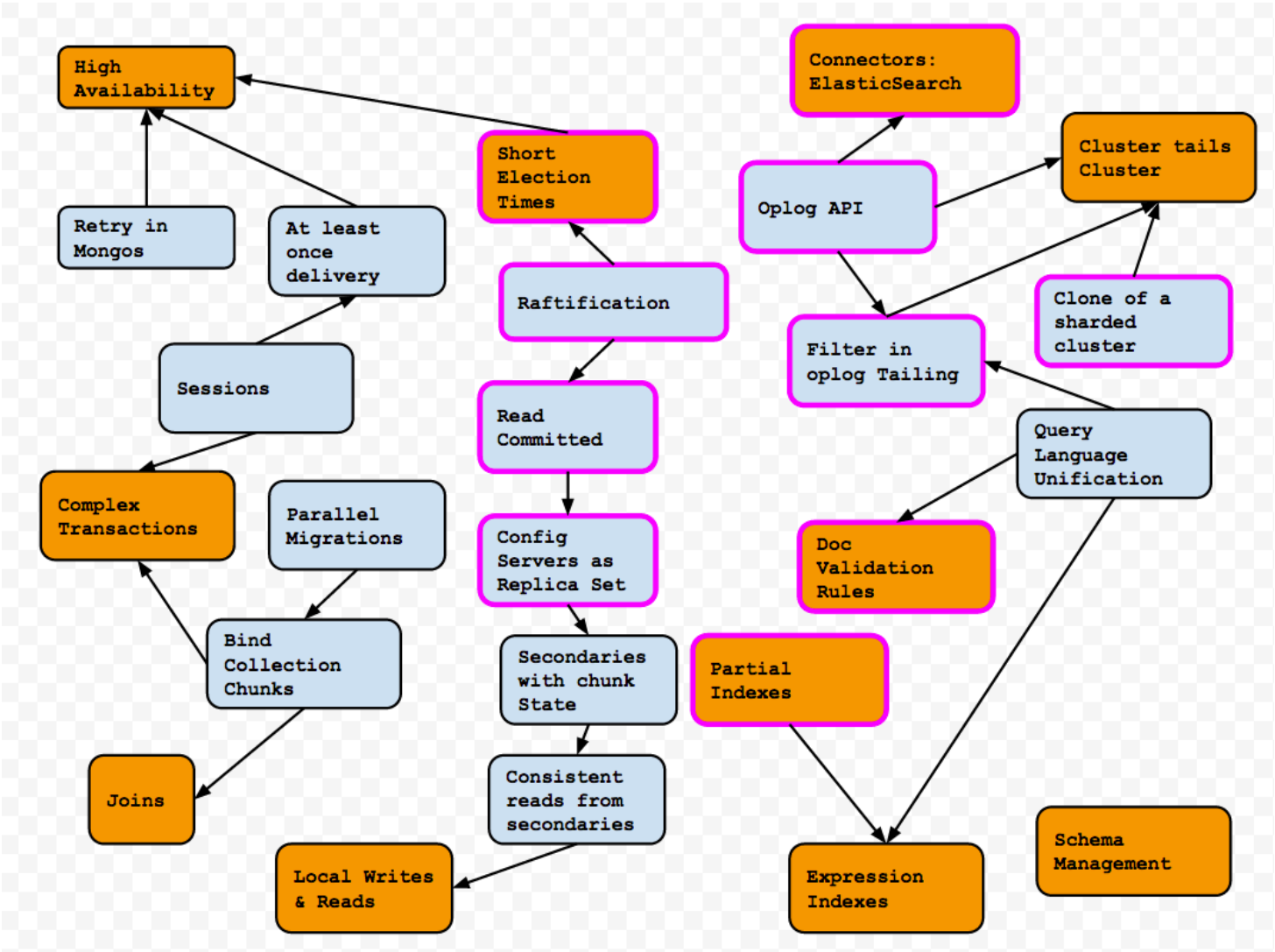
- Tags on Shards and on Key Ranges
  - sh.addShardTag()
  - sh.addTagRange()

- How to Balance Collections Across Your Sharded Cluster:
  - http://askasya.com/post/taggedcollectionbalancing

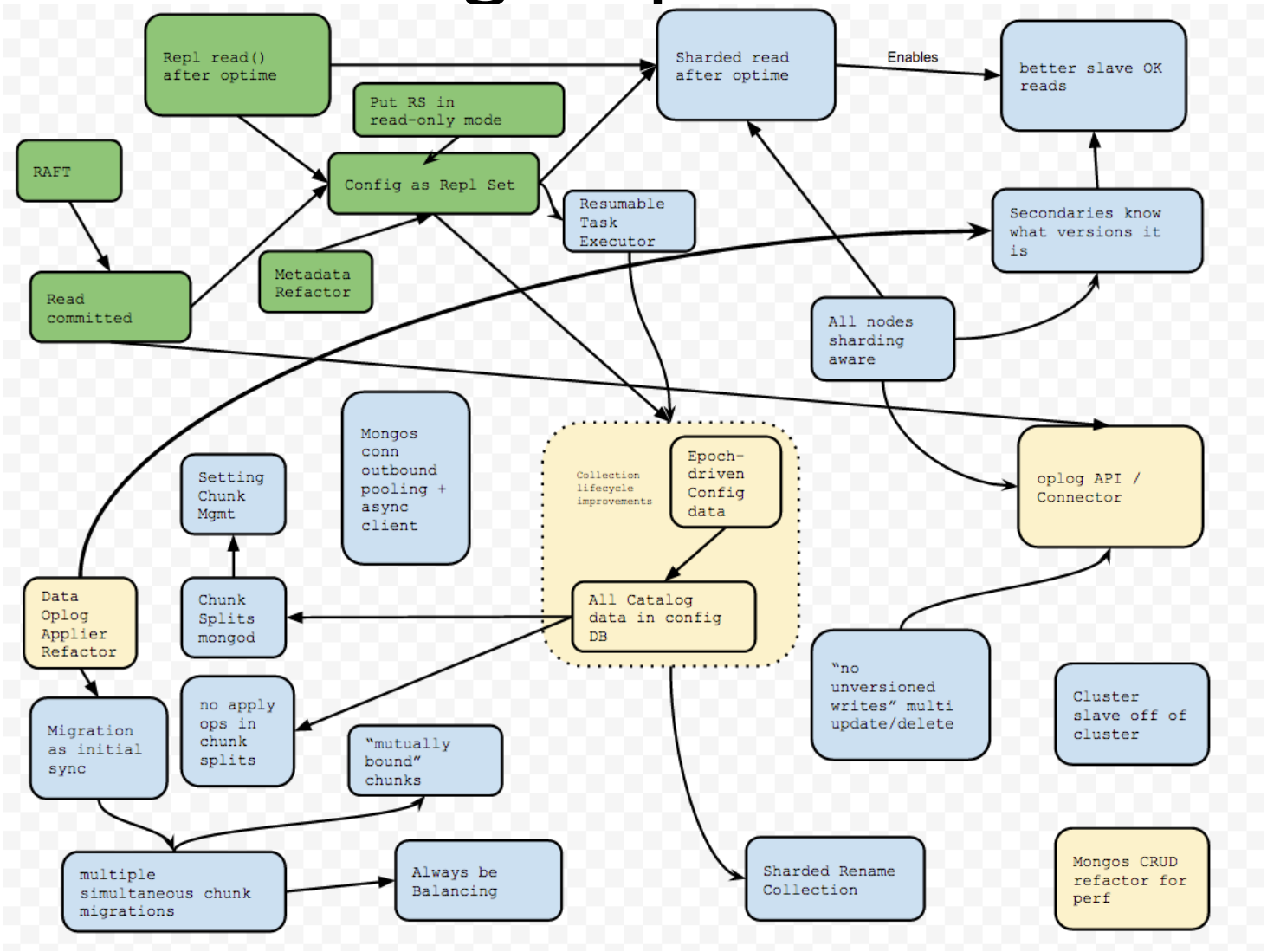
Example: What could possibly go wrong?

- Shard/Replica Set is “working fine”
  - 2 “regular nodes”
  - 2 - 24 hour delayed, hidden, votes=0, priority=0 nodes
  - Arbiter
- Moves failing… at the end… after 10 hours

Sharding Roadmap



Sharding Dependencies



Exercise

- What features work in replica set mode, but not in sharded clusters

References

- Main doc
  - http://docs.mongodb.org/master/MongoDB-sharding-guide.pdf
- Internals
  - https://wiki.mongodb.com/display/KERNEL/Sharding+Internals
- Tutorials
  - http://docs.mongodb.org/manual/administration/sharded-clusters/
- Card game to explain sharding:
  - http://www.kchodorow.com/blog/2011/01/04/how-to-choose-a-shard-key-the-card-game/