VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wananga o te Upoko o te Ika a Maui

MongoDB Architecture

Lecturer* : *Dr. Pavle Mogin

SWEN 432
Advanced Database Design and Implementation

Plan for MongoDB

- Sharding
  - Shard Key,
  - Sharding Architecture,
  - Balancing Data Distribution

- Replication
  - Master-Slave mode
  - Replica Set Operation
  - Failover - Election of a new Master

  - ***Reedings:***
    - *Have a look at Readings on the Home Page*

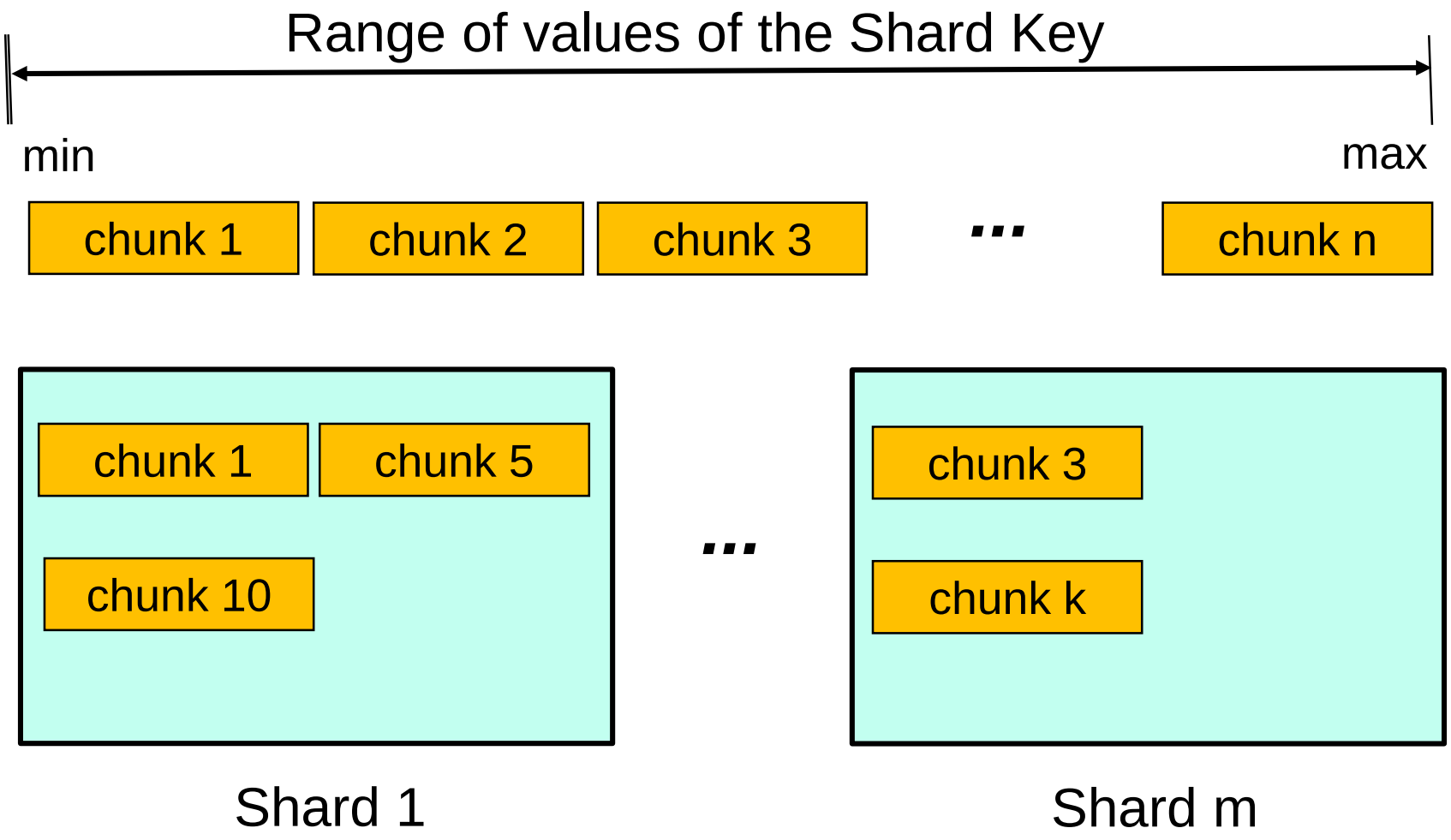
Shard Key

- Sharding partitions a collection's data by the shard key
- A ***shard key*** is:
  - A single or more fields that exist in each document of a collection,
  - It is desirable that a shard key has unique values, or at least very large range of values,
  - The shard key has to be indexed,
  - A field containing an array cannot be a shard key, or a part of a shard key
- A shard key has to be defined within the command enabling sharding for a database collection
- Shard keys are immutable and cannot be changed after insertion
- All sharded collections **must** have an index that starts with the *shard key* (shard key is a prefix)

Chunks of the Shard Key Range

- MongoDB divides the shard key value range into ***chunks*** (subranges of the value range) and distributes the chunks evenly across the shards
- Each document of a collection is stored in a chunk according to its sharding key value
- Dividing shard key values into chunks is done either by:
  - Range based partitioning, or
  - Hash based partitioning
- There also exists a tag aware partitioning, where:
  - Certain key value subranges are tagged,
  - Tags are assigned to particular shards,
  - Shard keys belonging to a tagged subrange go to the corresponding shard

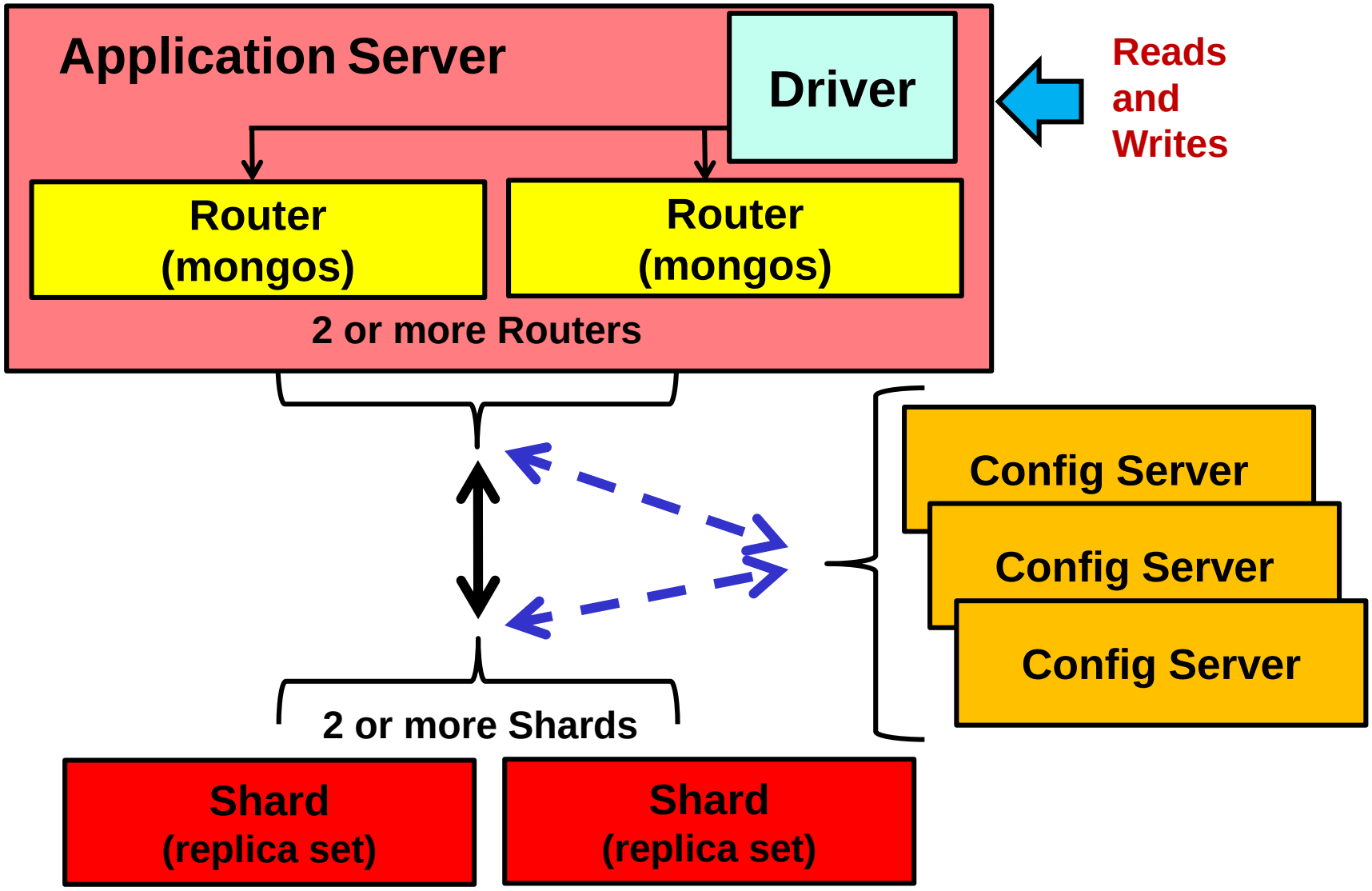
Sharding


Range of values of the Shard Key
min
max
chunk 1
chunk 2
chunk 3
...
chunk n
chunk 1
chunk 5
chunk 10
...
chunk 3
chunk k
Shard 1
Shard m

Range Based and Hash Based Sharding

- ***Range*** based ***sharding***:
  - The space of shard key values is divided into non overlapping ***chuks***
  - Each chunk is stored on a shard cluster
  - Documents having close shard key values are likely to be stored on the same shard
  - Favours range queries by shard key
  - May result in an uneven distribution of data and the work load
- ***Hash*** based ***sharding***:
  - The space of hashes of shard key values is divided into non overlapping ***chuks***
  - Documents having close shard key values are likely to be stored on different shards
  - Favours an even distribution of data and the work load
  - Inefficient range queries

Sharding Architecture



Comments on Sharding Architecture

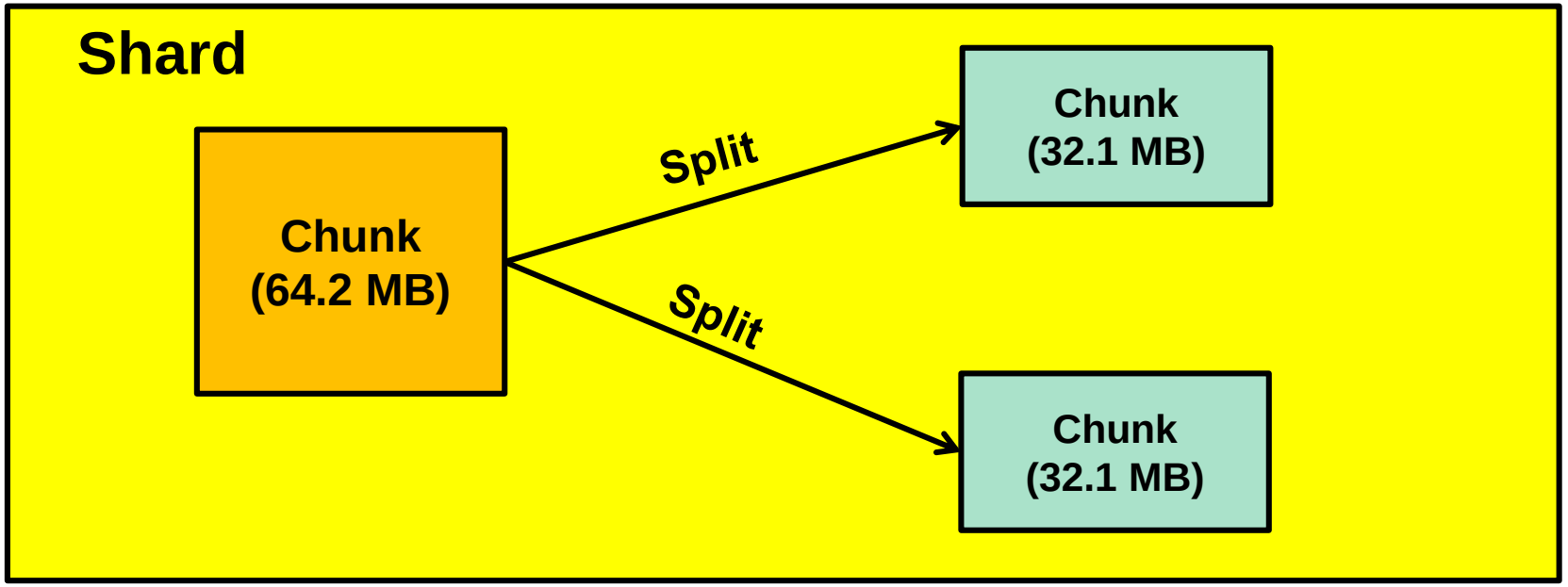
- MongoDB supports sharding through the configuration of sharded clusters
  - A ***sharded cluster*** consists of:
    - Three config processes,
    - One or more shards (replica sets), and
    - One or more mongos routing processes
  - Query ***routers*** or mongos instances interface with client applications via drivers, direct operations to the appropriate shards, and return results to clients
  - ***Config servers*** store the cluster's metadata containing a mapping of the cluster's data set to shards
  - The query routers use the metadata from config servers to target operations to specific shards
  - A driver is a client library for interacting with MongoDB in a particular language

Balanced Data Distribution

- The addition of new data or servers can result in data distribution imbalances
  - A shard can contain significantly more chunks than another, or
  - Size of a chunk may become significantly greater then another
- MongoDB uses two background processes to keep a cluster balanced:
  - Splitting and
  - Balancer

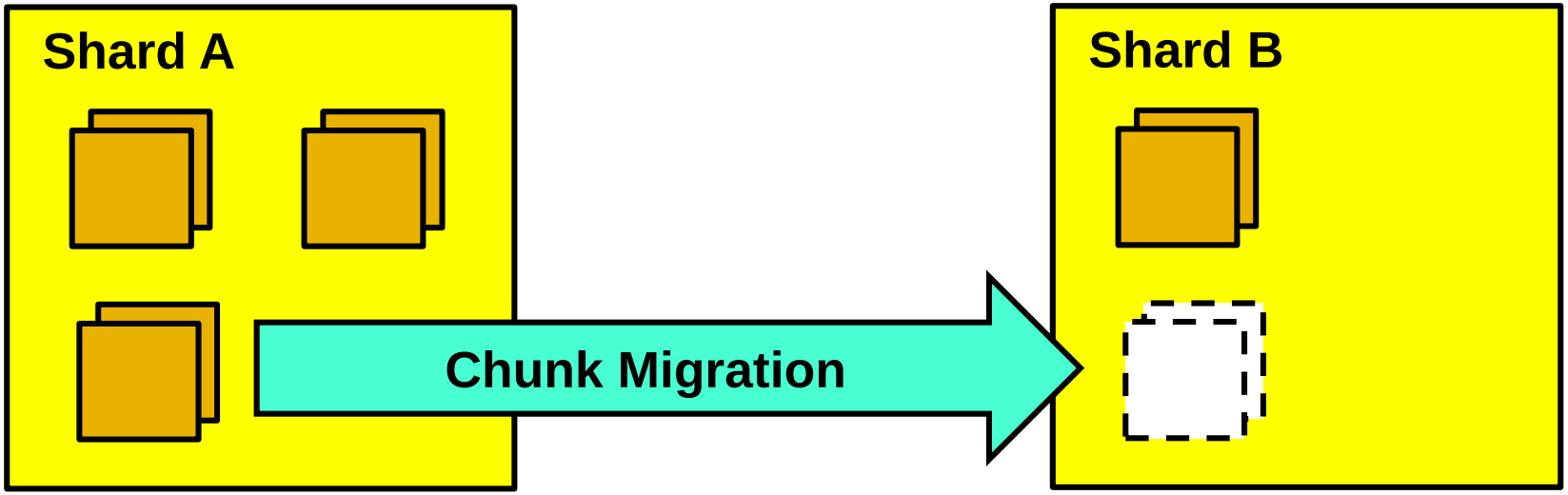
Splitting

- When a chunk grows beyond a specified size (64 MB), MongoDB:
  - Splits the chunk into two halves,
  - Does not migrate any data (the shard contains more chunks now)



Balancing

- When the distribution of chunks in a sharded collection becomes uneven in a cluster, a balancer that runs in all routers:
  - Migrates whole chunks from the shard with the largest number of chunks to the shard with the least number of chunks,
  - The origin shard keeps the chunk and answers queries until the chunk has been successfully transferred to the destination shard
  - Only then, MongoDB:
    - Updates the meta data on config servers appropriately, and
    - Deletes the chunk from the origin shard



Adding and Removing Shards

- When a new shard joins a cluster it creates imbalance, since the new shard has no chunks
  - A balancer starts migrating data immediately
- When a shard needs to be removed from a cluster
  - A balancer migrates all of its chunks to other shards,
  - Updates the meta data on config servers appropriately, and
  - Only then, the shard can be removed

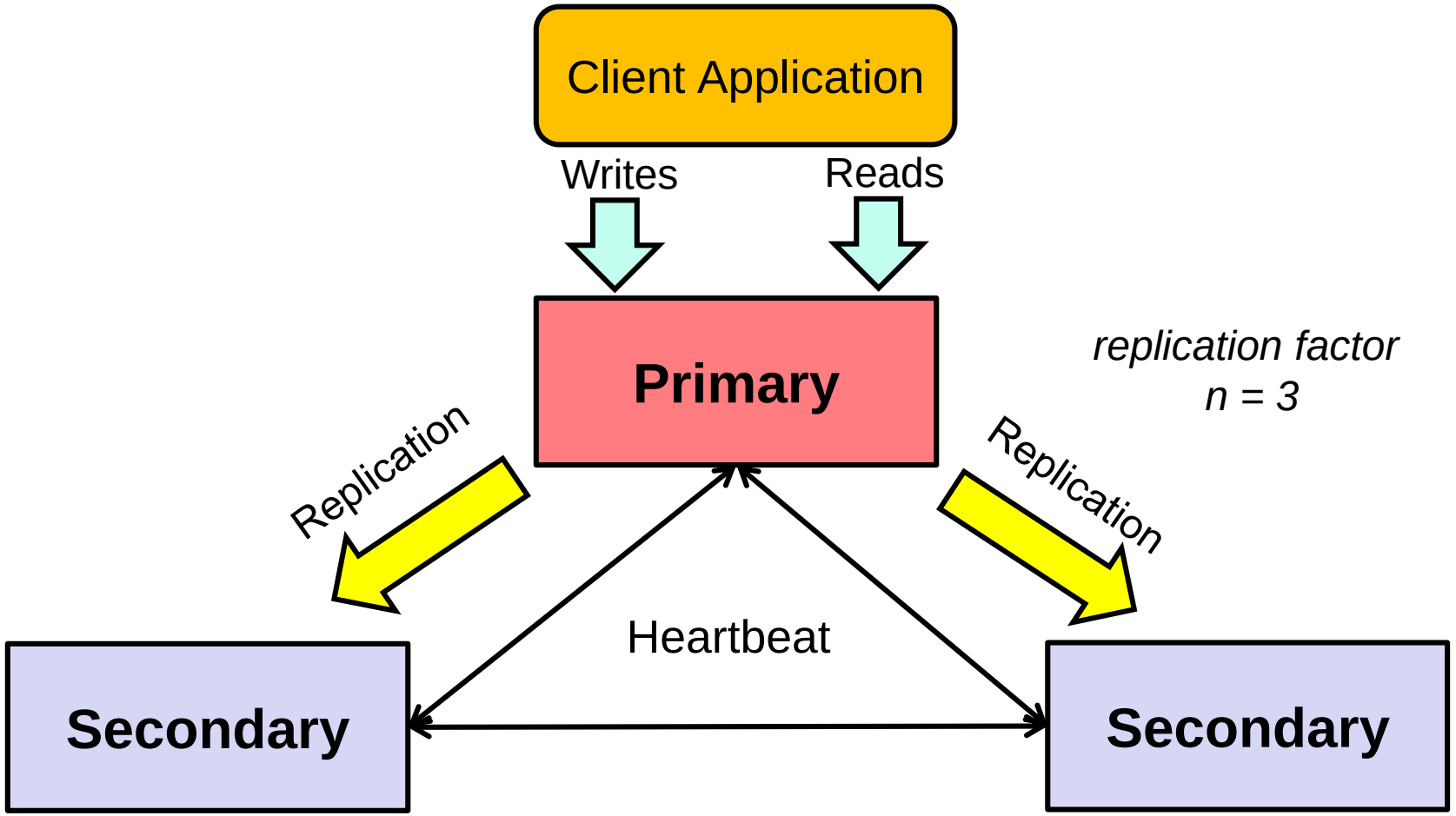
Replication

- In MongoDB, replication is implemented on the level of a shard
- A ***replica set*** contains replicas of a shard
  - A replica set is a cluster of MongoDB servers
  - There is a primary daemon process called `mongod` running on each server,
  - `mongod` handles data requests, manages data format, and performs background management operations
- Comment on the term daemon:
  - In multitasking programming systems, a **daemon**[1] is a program that runs as a background process, rather than being under the direct control of an interactive user
  - Traditionally daemon names end with the letter *d*

Master and Slaves

- One `mongod` – the primary (or master):
  - Accepts all write request from clients,
  - Updates its data set, and
  - Records update operations in its operation log
- All other servers of the replica set – secondaries (or slaves):
  - Receive operations from the primary's oplog, and
  - Apply them on their data sets
- If clients read from the primary, the replica set provides a strict consistency
- If clients are allowed to read from secondaries, the replica set provides an eventual consistency

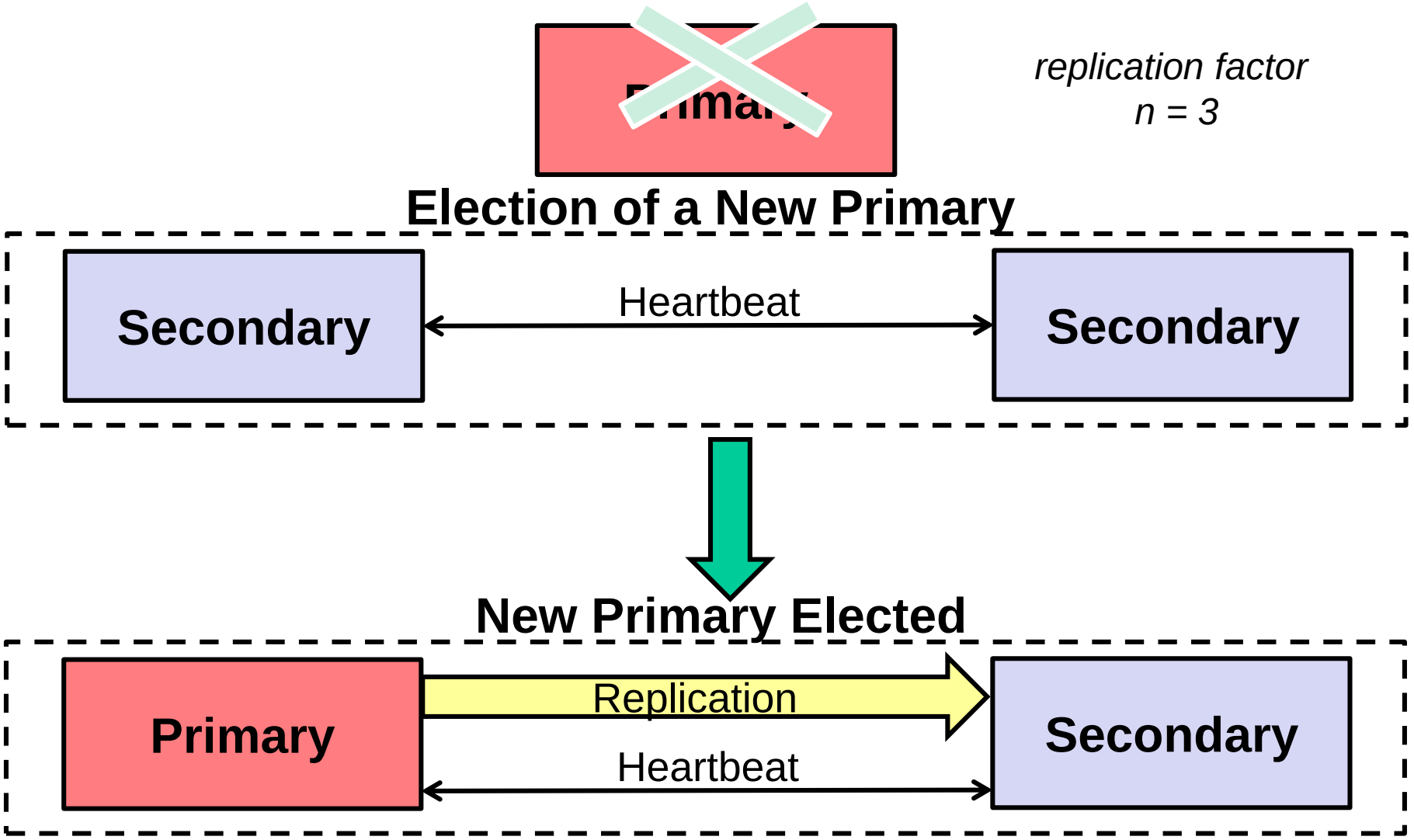
Replica Set Operations



Failover

- Replica set members send ***heartbeats*** (pings) to each other every two seconds
- If a heartbeat does not return within 10 seconds, the other members mark the not responding node inaccessible
  - If a secondary becomes inaccessible, the replica set can continue to function without it (providing that there are enough live servers left),
  - If the primary becomes inaccessible, the remaining members of the replica set have to elect a new primary by voting (if the voting quorum is still available),
  - The replica set can not accept any write request if there is no primary,
  - The first secondary that gets majority of votes becomes a new primary and the replica set resumes the normal mode of operation
    - Secondary having best performance, fastest response, least failure rate is promoted to a primary

Election


replication factor
n = 3
Election of a New Primary
Secondary
Heartbeat
Secondary
New Primary Elected
Primary
Replication
Heartbeat
Secondary

More on Election

- Fault tolerance for a replica set is the number of members that can become unavailable and still leave enough members in the set to elect a primary

No of Members	Election Majority	Fault Tolerance
3	2	1
4	3	1
5	3	2
6	4	2

- In a three members replica set, if two members are unavailable, the remaining one remains, or becomes the ***secondary*** disregarding what role it had before

Summary

- Data partitioning:
  - Sharding (horizontal partitioning) by a shard key, contained in all documents of a collection
- Data replication:
  - Performed on the level of a shard on the master-slave approach
  - Master is an instance of MongoDB that performs all writes and propagates them to slaves
  - If all reads also go to master – strict consistency,
  - Otherwise – eventual consistency
  - Master is a single point of failure
  - If master fails, a slave is elected as a new master
- A ***sharded cluster*** consists of:
    - Three config processes,
    - One or more shards (replica sets), and
    - One or more mongos routing processes
- **No data versioning**