

# MongoDB Robustness and Performance

CS 4417B

The University of Western Ontario

# MongoDB Features and Design Decisions

- Similar concepts to Hadoop
  - Replication
  - Sharding
- Differences from "traditional" DBMS
  - Data consistency is "optional"
  - User control over how much consistency is required
- Very complex in terms of operations and configuration options; this is an overview

# mongod

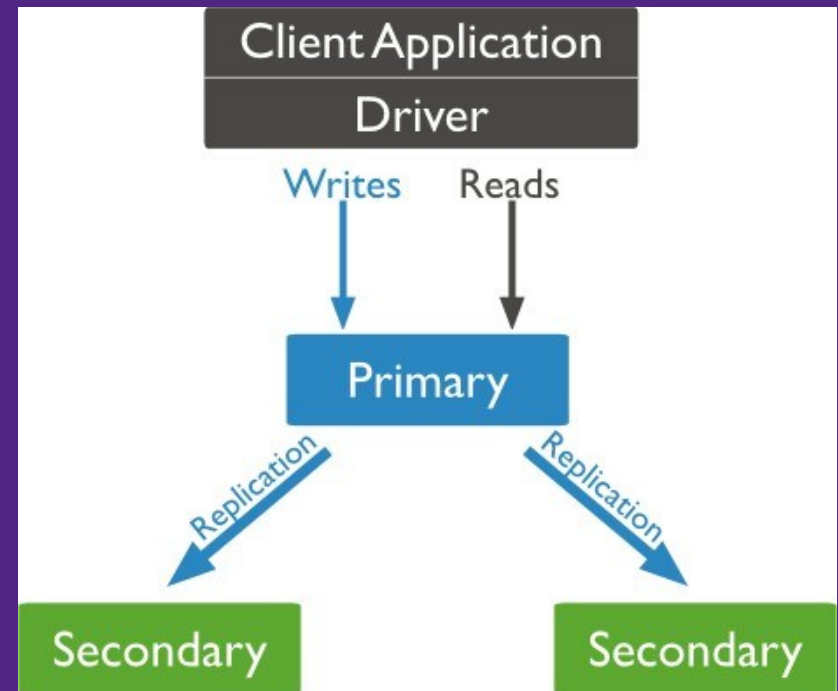
- `mongod` is the main daemon process for the MongoDB system. It handles data requests, manages data access, and performs background management operations.
- `mongod` can be a:
  - "regular" server
  - shard server
  - config server
- The `mongosh` shell connects to a `mongod` process (for non-sharded deployments) or a `mongos` process (for sharded deployments)

# Replica Sets

- A *replica set* is a group of `mongod` processes that maintain the same data:
  - **Primary:** only one - receives all write operations
  - **Secondaries:** normally at least 2 - replicate operations from the primary to maintain an identical data set.
  - Minimum recommended config is 3 in a replica set. (Primary + 2 secondaries.)
- Primary maintains *oplog* – log of all operations on its data.
- Secondaries replicate this and apply to their own data so everything is synced.
- Secondaries can dynamically choose their own *sync from* source based on ping, availability.

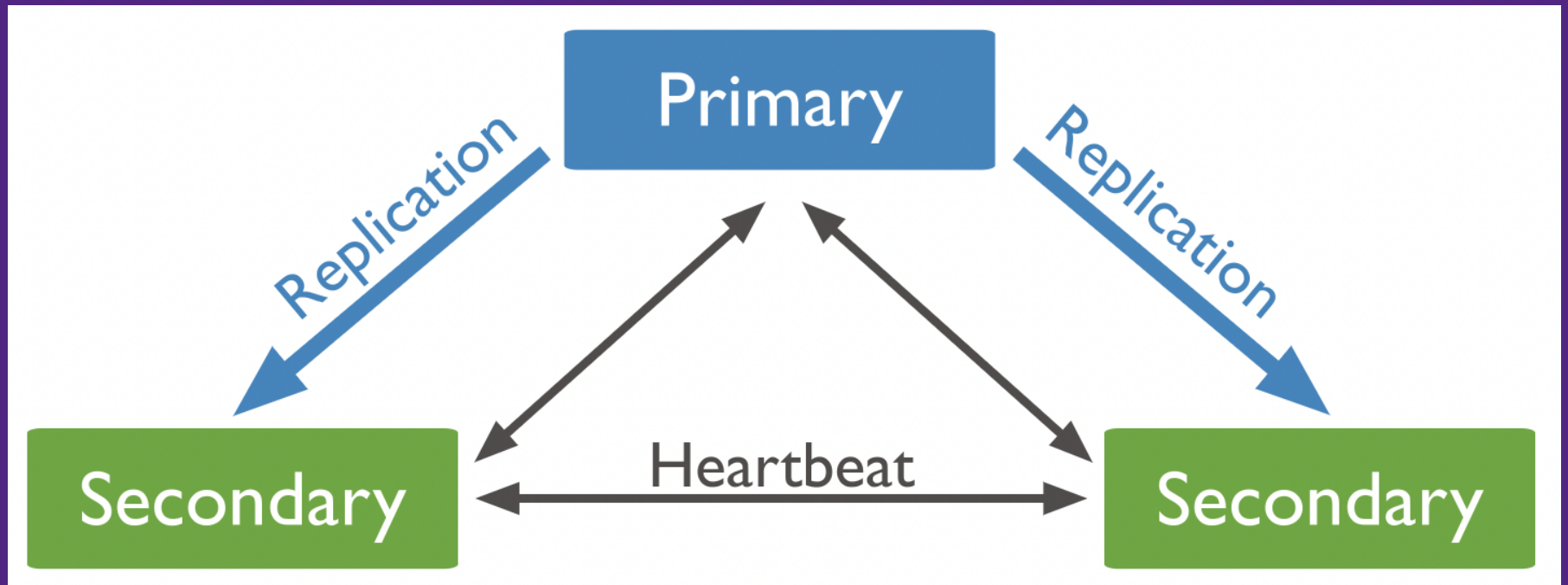
# Replica Sets

- The required number of secondary replicas depends on the read volume
  - Can dynamically be added
  - You don't have to stop operations



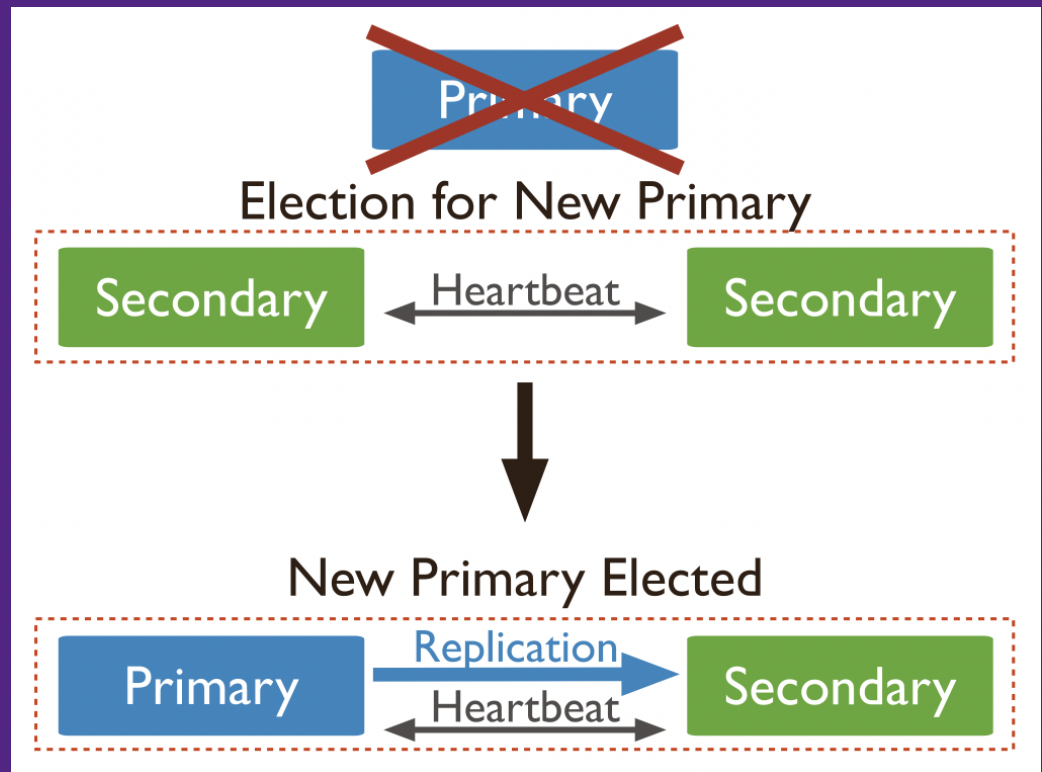
# Replicas

- All replicas have a number, 0, 1, 2, ..., N
- Replicas know about each other
- heartbeats are exchanged



# Automatic Failover

- When a secondary replica does not receive a response from the primary within the specified amount of time, it invokes the **bully algorithm** to “elect” a new primary



# Bully Algorithm

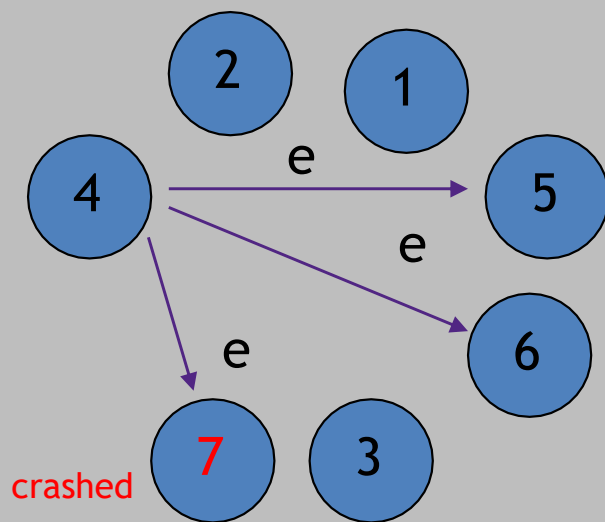
- When a secondary replica, **P**, notices that the primary is no longer responding, it initiates an election.
  - **P** sends an **ELECTION** message to all secondary replicas with **higher numbers**.
  - If no one responds, **P** wins the election and becomes the primary.
  - If one of the higher-ups answers, it takes over. **P**'s job is done.
- This algorithm is used in many distributed systems.



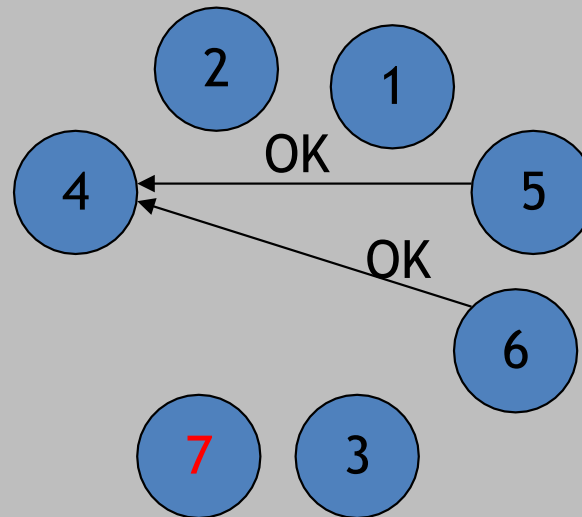
# Bully Algorithm

- When a replica gets an **ELECTION** message from one of its lower-numbered colleagues:
  - Receiver sends an **OK** message back to the sender to indicate that it is alive and will take over.
  - Receiver holds an election, unless it is already holding one.
  - Eventually, all replicas give up but one, and that one is the new primary.
  - The new primary announces its victory by sending all processes a message telling them that starting immediately it is the new primary.

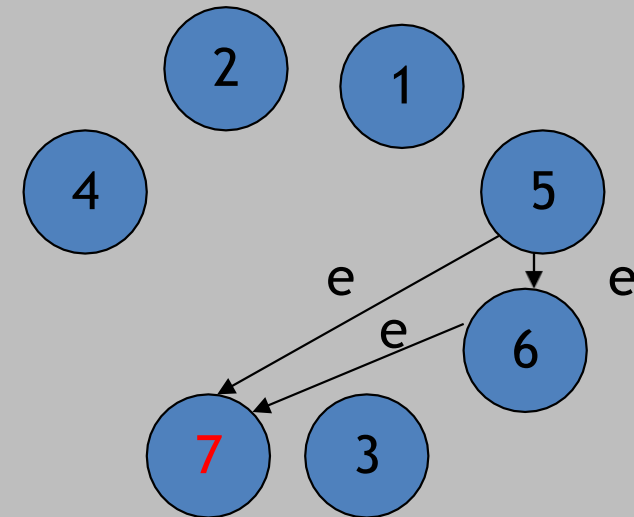
# The Bully Algorithm (Example)



- The primary (or coordinator) which is 7 crashes
- Secondary replica 4 holds an election

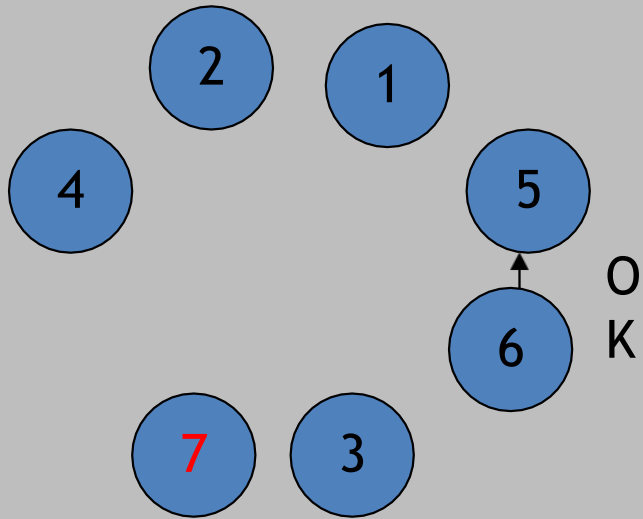


- 5 and 6 Respond
- Replica 4 stops participating

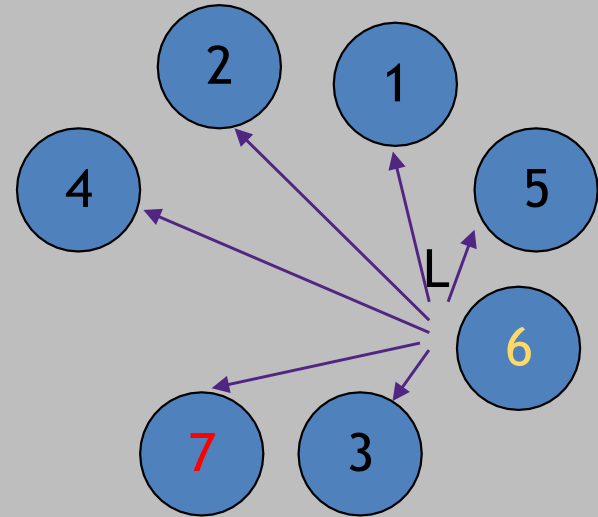


- 5 and 6 each hold an election

# The Bully Algorithm



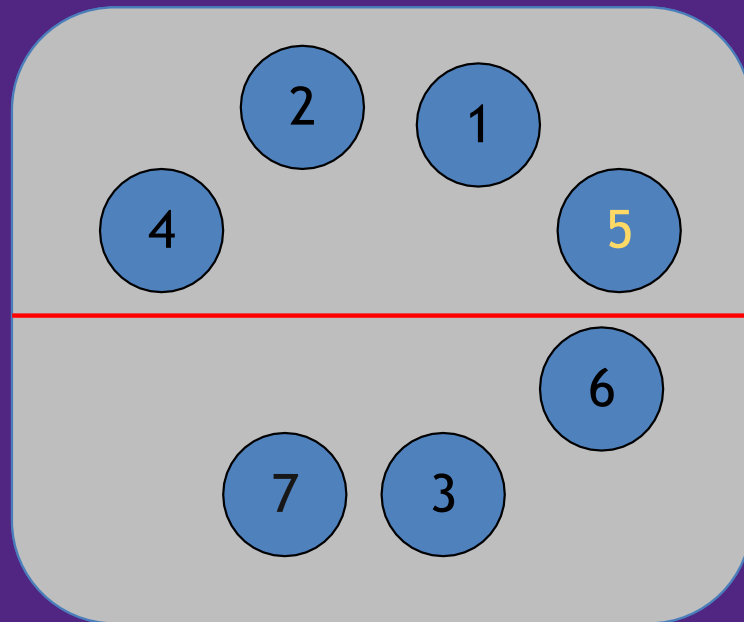
- 6 Responds
- Replica 5 stops participating



- 6 announces that it is the new primary (L)

# Network Partition

- If the Primary cannot see a majority of the nodes in its replica set, it steps down and becomes a Secondary.
- Secondary nodes that can't see a majority won't start an election.



# Recovery

- If a Primary that was previously down comes back:
  - It becomes a Secondary replica
  - Things might be inconsistent!

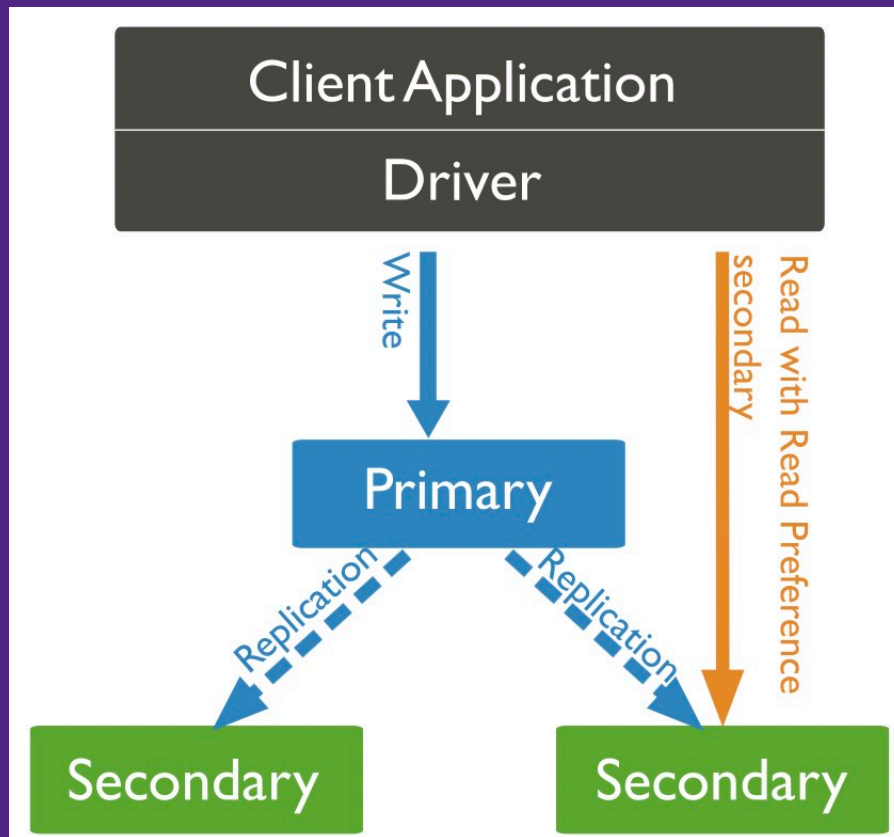
# Rollbacks

- Suppose primary accepts a write, but crashes (or network fails) before replication.
- Primary is removed from the Replica Set
- Later on, it comes back as a Secondary
  - But it's inconsistent because of the write that never got replicated
- The node "rolls back" (undoes) the write operation.
- "Shouldn't happen often," says MongoDB docs.

# Availability During Elections

- The replica set cannot process write operations until the election completes successfully.
- The replica set can continue to serve read queries if such queries are configured to run on secondaries while the primary is offline.

# Read Preference



- primary
- primaryPreferred
- secondary
- secondaryPreferred
- nearest



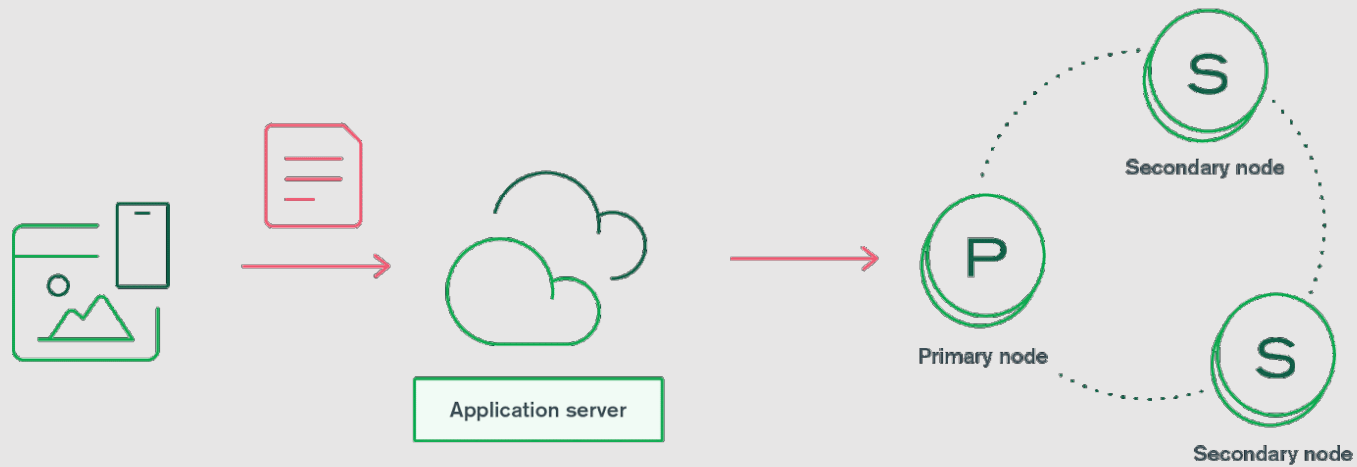
# Consistency of Data

- All read operations issued to the primary are consistent with the last write
- Reads to a primary have strict consistency
  - Since it reflects the latest changes to the data
- Reads to a secondary have eventual consistency
  - Updates propagate gradually
- It is possible that a client reads stale data

# Write "Durability"

- Data "durability" is just how resistant a system is to data loss.
- "Write concern":
  - How many nodes in the MongoDB database cluster need to acknowledge the write before success is passed back to the application?
  - w:1 – "success" after writing to at least one node
  - w:majority – "success" after writing to majority of replicas
  - w:majority avoids rollbacks (if confirmed)

# w:majority



# Read Concern



fast

- "local"
  - No guarantees; **might be rolled back**
- "available"
  - No guarantees; **might be rolled back**, might return "orphaned documents" (sharding issue – discuss later)

slow



- "majority"
  - Only returns **majority-acknowledged** writes
- "linearizable"
  - Returns **\*all\*** **majority-acknowledged** writes
- "snapshot"
  - Returns **\*all\*** **majority-acknowledged** writes at specified timestamp

Why didn't we have  
these "Concerns" with  
Hadoop?

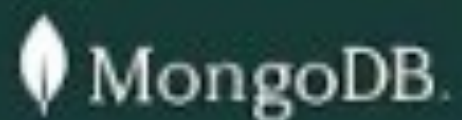
# Hadoop has *one-copy-update* semantics

- Roughly, an HDFS file must behave like a regular POSIX file (Linux, etc.)
- Once a create/update/delete completes, **everybody sees the same thing.**
- A file doesn't become visible to anybody until it is completely written.
- This is ensured by the **NameNode**

# Sharding

**REQUIRED WATCHING:**

<https://www.youtube.com/watch?v=8sk75-6W0ik>



# Demystifying Sharding in MongoDB

Albert T. Wong, Solutions Architect, MongoDB