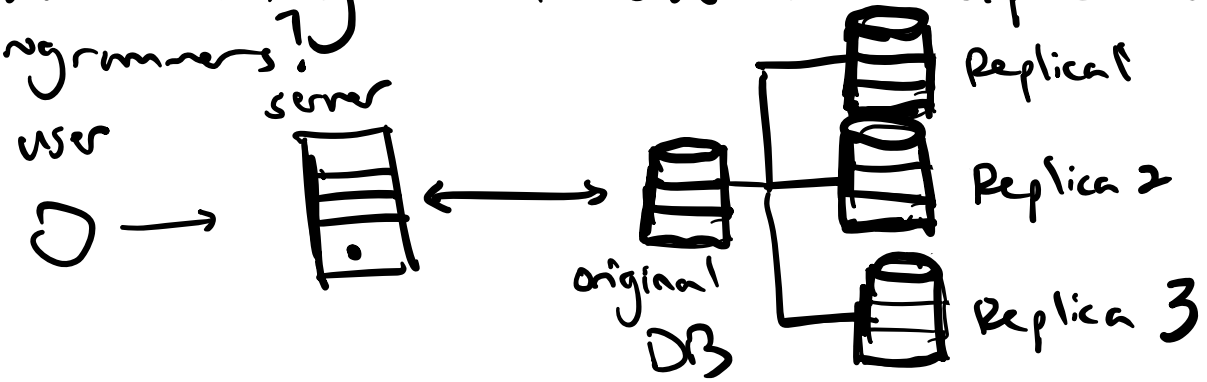
Data Replication

- We need the following characteristics from our data store:
  - ↳ Availability under faults (failure of some disk, nodes, network, & power outages)
  - ↳ scalability (w/ increasing reads, writes, & other ops)
  - ↳ performance (low latency & high throughput for the client)

- Replication
  - Keeping multiple copies of the data at various nodes achieves availability, scalability, performance
  - Complexities:
    - ↳ How do we keep multiple copies of data consistent w/ each other
    - ↳ How do we deal w/ failed replica nodes
    - ↳ Should we replicate synchronously or async?
    - ↳ How do we handle concurrent writes
    - ↳ What consistency model needs to be exposed to end programmers?



- Synchronous vs. Asynchronous Replication
  - Synchronous Replication
    - ↳ Leader node waits for acknowledgements from follower nodes that data has been updated.
    - ↳ After follower nodes return successfully, leader node returns successfully
    - ↳ Pro: All follower nodes are up to date w/ the leader

↳ con: If one of the follower node fails, primary node won't return → High latency

– Asynchronous Replication
↳ Leader node doesn't wait for follow nodes
↳ Pro: Leader node can continue its work even if follower nodes are down
↳ con: If primary node fails, the writes that won't copied to the secondary nodes will be lost

- Data Replication Models
  - Single leader replication
  - Multi-leader replication
  - Peer to Peer or leaderless replication

- Single Leader (Leader - Follower)
  - Data is replicated across multiple nodes
  - One node is leader
    ↳ responsible for processing writes to data stored on cluster
    ↳ sends writes to follower nodes, keeps them in sync
    ↳ Works well for read-heavy systems
      - can add more follower nodes easily for increased reads
    ↳ Replicating to many followers makes a bottleneck
  - Read resilient: follower nodes can handle reads even if leader fails
  - con: Single leader Replication is inconsistent if we use asynchronous replication – The secondary nodes may have incorrect data since async may return before it gets copied to secondary node handling that read

- If leader node fails, a follower is selected as a leader:
  - ↳ manually: operator decides new leader
  - ↳ auto: follower nodes decide w/ a leader election (choosing processor w/ highest identifier)
- Statement Based Replication
  - ↳ Leader node saves all executed statements: insert, delete, update, etc. Then sends them to secondary nodes to perform
  - ↳ Disadvantage: nondeterministic functions such as (Now()) might result in distinct writes on the follower & leader
    - If a write statement depends on a prior write, & both of them reach the follower in the wrong order, the follower node output is uncertain
- Write Ahead Log shipping (PostgreSQL, Oracle)
  - ↳ Leader saves the query before executing it in a log file known as write ahead log file
    - Then uses these logs to copy to follower nodes
  - ↳ Problem: WAL defines data at a very low level, tightly coupled w/ inner structure of the DB engine, which makes updating software on the leader & followers complex
- Logical (row-based) log replication
  - ↳ All secondary nodes replicate the actual data changes
  - ↳ If a row is inserted or deleted in a table, follower nodes will replicate that change in that specific table
  - ↳ Binary log records change to DB tables on the leader node @ the record level
  - ↳ To create a primary/leader replica, followers read the data & change their records accordingly
  - ↳ less complex than WAL: doesn't need info about data layout inside DB engine

- Multi-Leader replication
  - Multiple primary nodes that process the writes & send them to all other primary & secondary nodes to replicate
    - ↳ used in databases along w/ external tools like Tungsten Replicator for MySQL
    - ↳ useful in apps in which we can continue work even if we're offline
  - Conflict
    - ↳ Multi-leader replication gives better performance & scalability than single leader replication
      - Disadvantage: All primary nodes deal w/ write requests which creates a conflict
  - Handle conflicts
    - ↳ Conflict Avoidance
      - All writes for a record go thru same leader
      - Conflict might still happen if user moves & is now near a diff data center
    - ↳ Last write-in
      - using local clock: all nodes assign a timestamp to each update. When a conflict happens, update w/ latest timestamp is chosen
      - challenge: clock sync is tough across nodes - clock skew = data loss
    - ↳ Custom Logic
    - ↳ Multi-Leader Replication: circular topology, star topology, all-to-all topology

- Peer-to-peer / leaderless replication (DynamoDB)
  - Leader is a bottleneck & single point of failure
  - Fails to provide write scalability
  - In P2P: no single leader node, all nodes have weights

& can accept reads & write requests
- Inconsistency: when several nodes accept write requests, may lead to concurrent writes. Solution: Quorums