CSC 369

Introduction to Distributed Computing

Scaling Up: Shared Memory

- Simple approach: more powerful computer
- Many CPUs, RAM chips, disks merged together under one O/S
  - Treated as one single machine
- Cost/performance not linear
  - A 2x more powerful machine costs more than 2x more
  - 2x components can't necessarily handle 2x the load

Method 1: create a more powerful computer. but cost/performance is not linear.

Scaling Storage: Shared Disk

- Several machines with independent CPU/RAM
- Data stored on a shared array of disks
  - Network Attached Storage (NAS)
  - Storage Area Network (SAN)
- Downsides: contention, locking required for writes

Method 2: several independent CPU/RAM but sharing the same storage. we then need locking for write

Shared Nothing Architecture

- Each server or virtual machine (**node**) has independent CPU, RAM, and disk
- Coordination occurs over conventional network
- Individual machines can be configured for best price/performance, located in multiple geographic locations

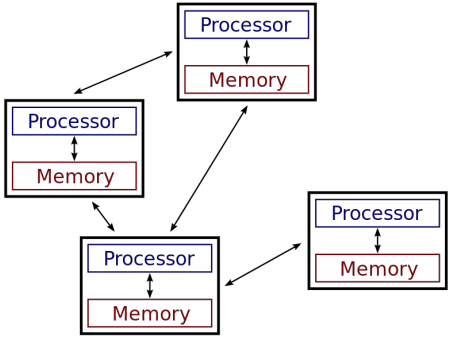


Image from Wikipedia

Method 3

Distributing Data

2 approaches if we use share nothing architecture: Replication & Partitioning

Two (complementary) approaches to distributing data across nodes:

- Replication

  Replication is discuss in this lecture

  - Keep an exact copy of data on multiple nodes

- Partitioning
  - Split a large dataset into smaller subsets, store each subset on a different node

Separate mechanisms, but often used together

Replication

- Geographic proximity to users (reduce latency)

  shorter distance to get data

- Allow node failure or downtime for upgrades (increase availability)

  if one node is down we can use other node from other region to get the data. then we can have downtime for the failure node

- More nodes to handle *read* requests (increase read throughput)

  reading data is very efficient. however write is not.

Straightforward if each node can store entire dataset and/or data does not change frequently

Method 1 for replication

Leader-Based Replication

- Each node that stores a copy of the database is called a **replica**
- One of the replicas is designated as **leader**.
  Every client write request is sent to the leader.
- Other replicas are called **followers** (sometimes referred to as: read replicas, secondaries, hot standbys)

Method 2 for replication is can handle write request better

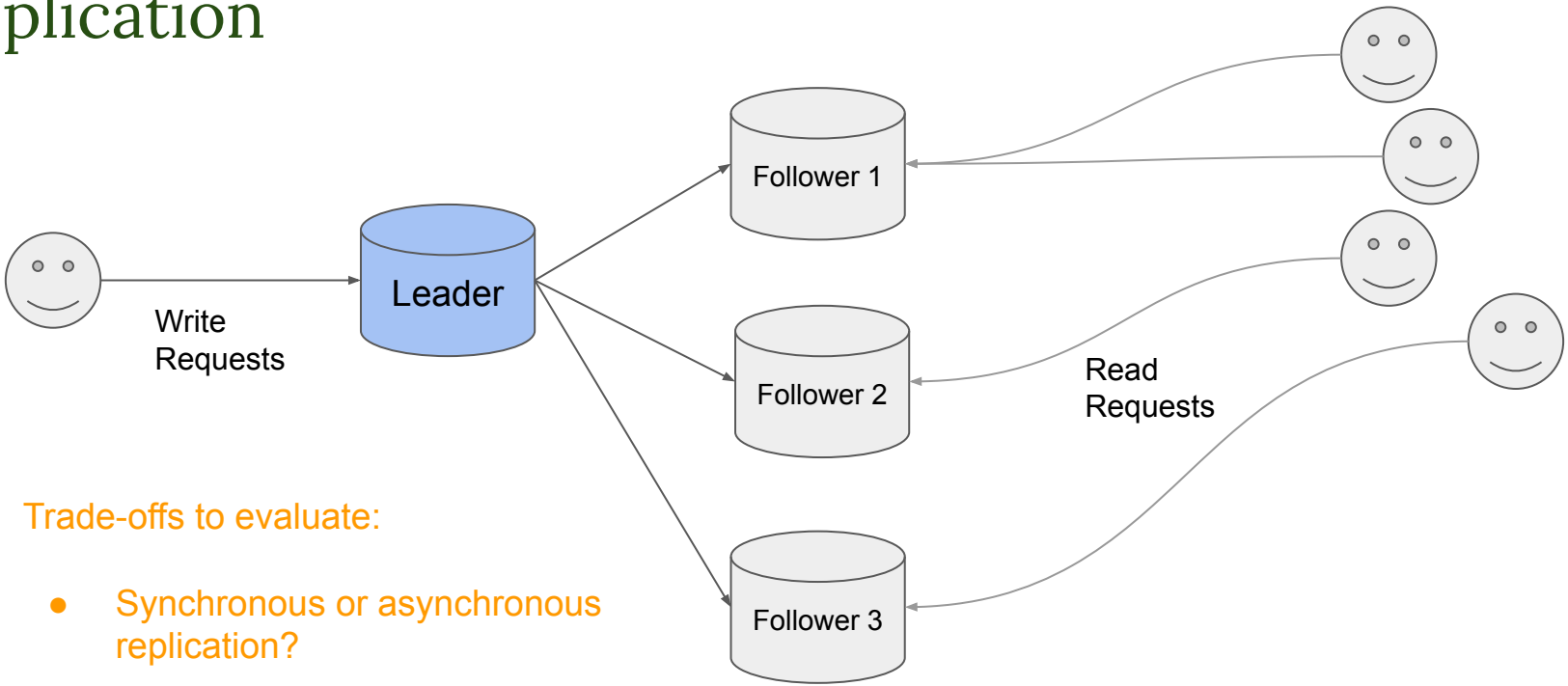
Leader-Based Replication

- Particularly suitable for workloads that consist of many more read requests than write requests.

  why? should it be the opposite ?

- Leader is write-only, read requests may be served by any replica

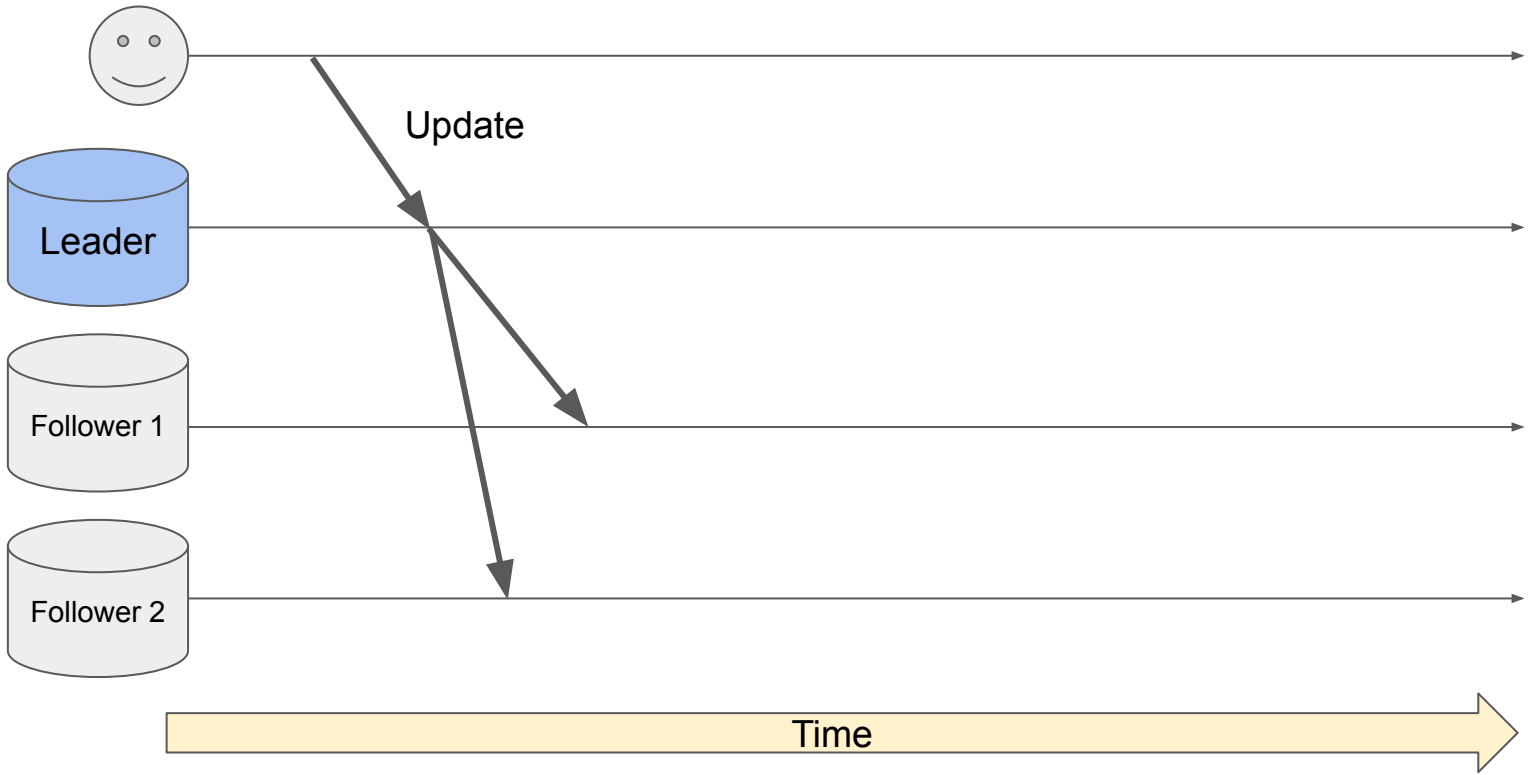
Replication



Trade-offs to evaluate:

- Synchronous or asynchronous replication?
- Single-leader, multi-leader, leaderless?

Replication



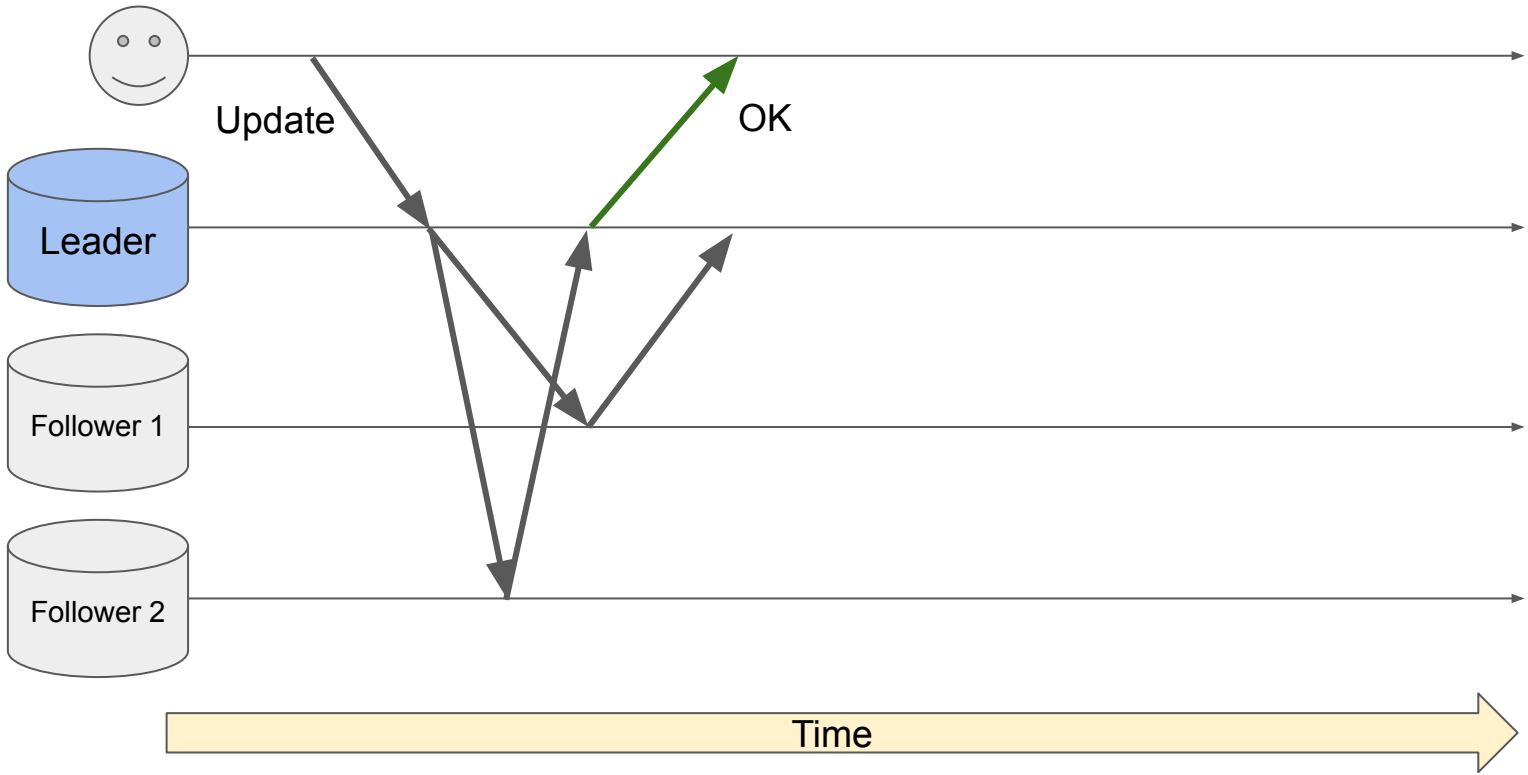
Synchronous Replication

- In a **synchronous** replication configuration, the leader waits until one or more followers has confirmed each write before reporting success to the user

- Impractical for *all* followers to be synchronous.

- **Semi-synchronous**: just *one* follower is synchronous, others are asynchronous

Semi-Synchronous Replication



Synchronous Replication

- Advantage: At least one follower is guaranteed to have an up-to-date copy of data that is consistent with the leader

- Disadvantage: If asynchronous follower does not respond, write cannot be processed

Asynchronous Replication

- Writes are sent to leader, stored there, then forwarded to all followers.
- Leader *does not wait* for follower response before reporting success to client.
- Problem: Durability
  - Solutions: ? (active area of research)
  - Microsoft Azure Storage uses an approach called *chain replication*.

Adding New Followers

New follower must sometimes be added to an existing system

- Handle increased load
- Replace failed nodes

How to ensure that the new follower has an accurate copy of leader's data?

- Direct file copy is problementic (new writes arriving constantly)
- Locks are undesirable (leader would block writers for a period of time)

Not fucking good!!!!! the correct way is in the next slide

Adding a New Follower

- Take snapshot of leader's data (record timestamp or some other sequence number or transaction id, to identify "point in time")
- Copy snapshot to new follower
- Follower requests all changes since snapshot timestamp
- Follower applies changes; can begin to process requests when caught up

Node Outages

- Temporary outage of a follower can be handled using catch-up process, similar to adding a new follower

- Leader outage is more difficult, requires **failover** process
  - Existing follower must be chosen as new leader
  - Clients must be reconfigured to send requests to new leader
  - Other followers need to be notified of new leader

Node Outages / Failover

Automatic failover process

1. Determine that the leader has failed
    a. Typically using a timeout
2. Choose a new leader
    a. Consensus problem
3. Reconfigure system to use new leader

Implementation of Replication

- Statement-based replication deo hieu
  - INSERT, UPDATE, DELETE
  - Issues with non-deterministic functions, triggers, side-effects
- Write-ahead-log (WAL) shipping
  - Log is an append-only sequence of bytes containing all writes
- Logical replication
  - Middle-ground between statement and physical/WAL approach
  - Writes are described using a replication-specific format: new data, id of removed records, etc.

Limitations of Replication

- Fully-synchronous replication avoids inconsistent views of data (all readers see the same data)
  - Single node failure is a problem

- Fully-asynchronous replication allows (temporarily) inconsistent views of data - some clients may see outdated information
  - Eventual consistency

Fundamental Problems in Distributed Systems

- Node failures
- Unreliable networks
- Tradeoffs surrounding
  - Replica consistency
  - Durability
  - Avilability
  - Latency

Multi-Leader Replication

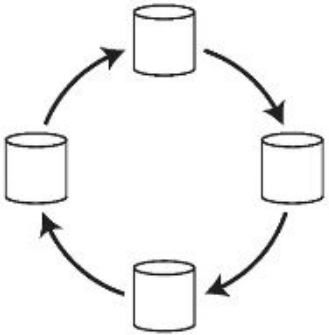
Single-leader configuration introduces single point of failure

Why not allow more than one node to accept writes?

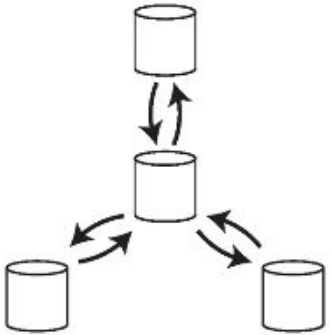
- Multiple datacenters
- Offline mode

Multi-Leader Replication - each node that processes a write must forward the write to all other followers

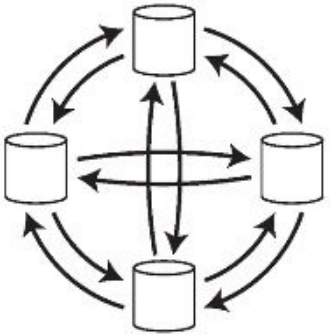
Multi-Leader Replication Topologies



(a) Circular topology



(b) Star topology



(c) All-to-all topology

Multi-Leader Replication: Circular / Star

Widely implemented (MySQL supports circular replication by default)

Issues:

- Writes need to pass through multiple nodes
- Node failures
- Replication loops

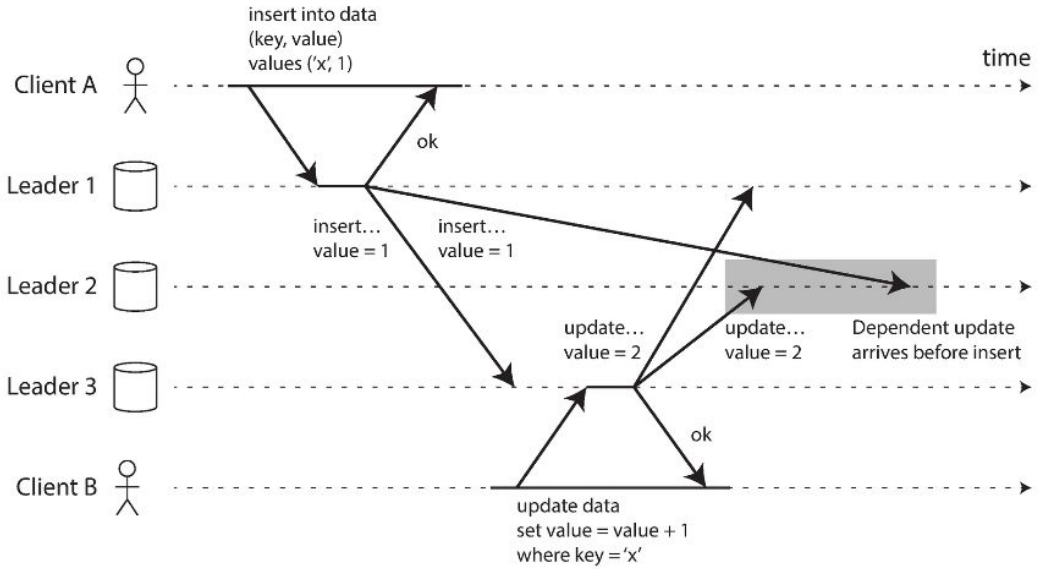
NOT UNDERSTAND YET

Multi-Leader Replication: All-to-All

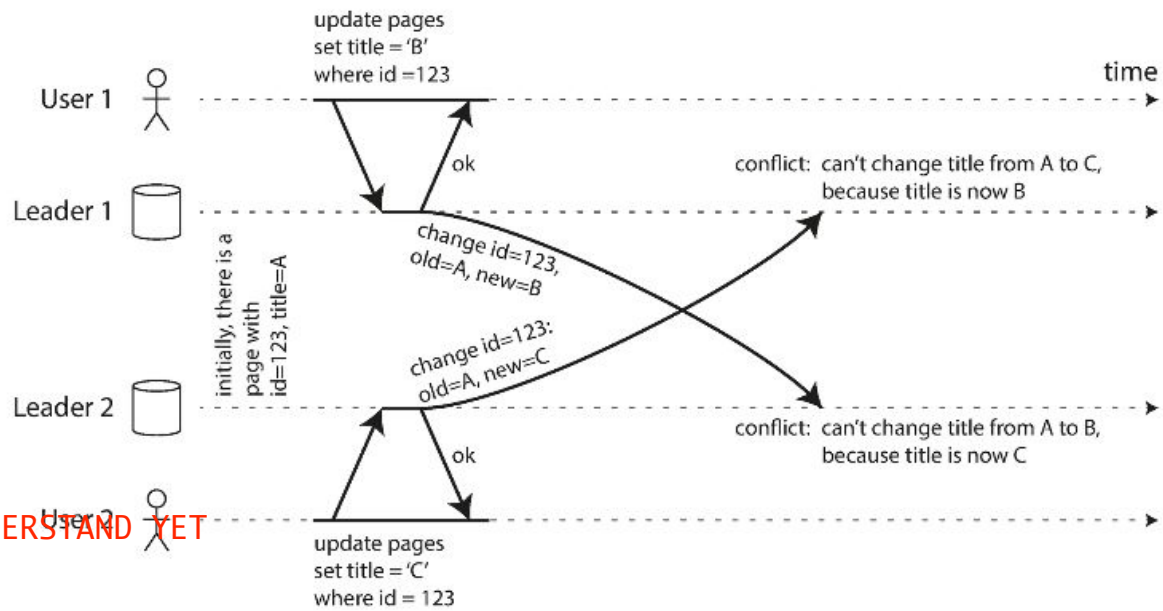
Can improve fault tolerance

Issues: varying network link speeds -- out-of-order writes

NOT UNDERSTAND YET



Write Conflicts



NOT UNDERSTAND YET

Possible Approaches to Handling Write Conflicts

- Last writer wins (LWW)
- Assign precedence to each replica
- Marge values (title "B/C" in previous example)
- Record conflict, write application code to resolve (possibly with user input)

NOT UNDERSTAND YET

Leaderless Replication

No leader; any replica can accept client write requests ("Dynamo-style")

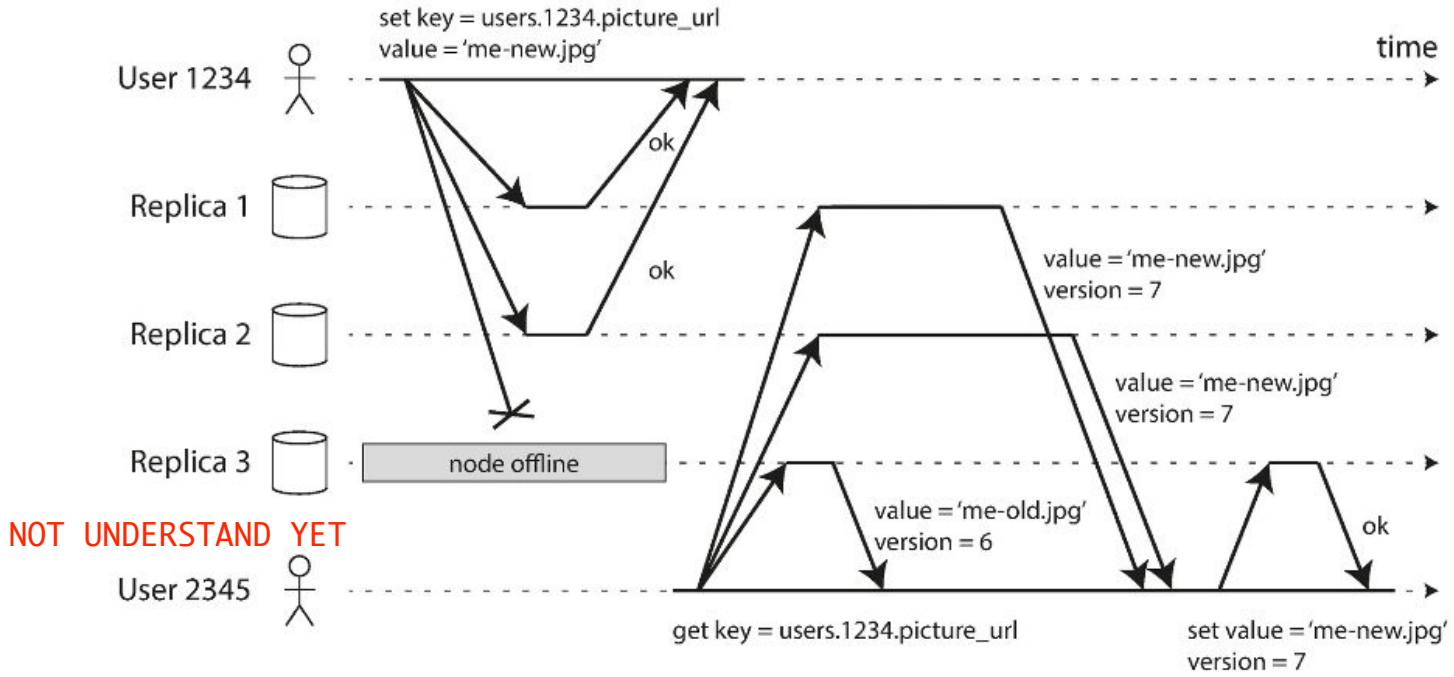
In some implementations, clients directly send writes to several replicas.
Sometimes, a "coordinator" node sends writes on behalf of clients

Leaderless Replcation - Quorum

- With three replicas, a write is considered successful if processed by two or more replicas.
- If we later read from two replicas, we can be be sure that at least one is up to date

- To generalize for *n* replicas:
  - Write must be confirmed by *w* nodes
  - Read from *r* nodes
  - If $w + r > n$ we expect to get an up-to-date value when reading

Leaderless Replication Example



Quorums for Read and Write

- *n* replicas:
  - Write must be confirmed by *w* nodes
  - Read from *r* nodes
  - If *w* + *r* > *n* we expect to get an up-to-date value when reading
- Parameters *n*, *w*, *r* are typically configurable
  - Commonly: choose odd number for *n*, set *w* = *r* = *(n+1) / 2* (rounded up)
- *w* < *n* and *r* < *n* allows reads/writes to succeed even when nodes are unavailable
  - For example: *n = 3, w = 2, r = 2* tolerates one unavailable node
  - If n = 5 what is the tolerance for unavailability?

NOT UNDERSTAND YET

Replication