Chapter 5: Replication - Part I

Chapter 5 of this book covers the important topic of replication in the context of database systems. Replication is the process of creating and maintaining multiple copies of the same database on different servers. This is done to ensure data availability, improve performance, and provide fault tolerance.

The chapter starts by introducing the concept of replication and its benefits. It then goes on to explain the different types of replication, including master-slave, master-master, and multi-master replication. Each type is discussed in detail, with its advantages and disadvantages.

The chapter also covers the different replication topologies, such as star, tree, and mesh. It explains how each topology works and when it is appropriate to use them.

Next, the chapter discusses the issues related to replication, such as consistency, conflicts, and latency. It explains how these issues can be addressed using locking, timestamping, and conflict resolution techniques.

Finally, the chapter concludes with a discussion on the challenges of replication and how they can be overcome. It also provides some best practices for designing and implementing replication systems.

Overall, Chapter 5 provides a comprehensive overview of the replication process and its importance in database systems. It is a must-read for anyone interested in building scalable and fault-tolerant database systems.

Date: @March 13, 2023 9:20 PM

Topic: PART II - Distributed Data

Recall	Notes
Why do you need to embrace	• Scalability: workload grows bigger than a single machine • Fault tolerance/high availability

distributed databases?

- Latency

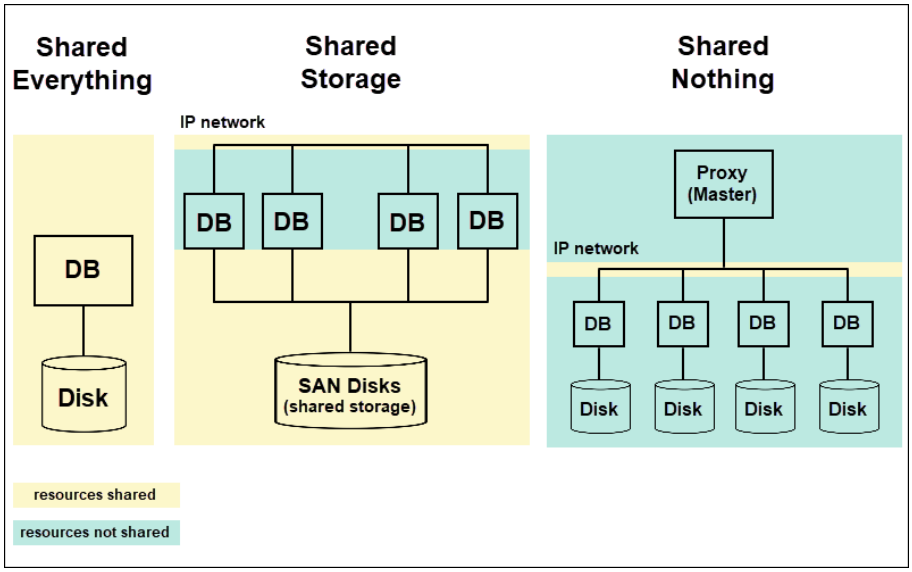
Scaling to a Higher load approaches

1. Vertical scaling (shared-memory architecture)
2. shared-disk architecture
3. Horizontal scaling (shared-nothing architecture)

The problems with shared-memory architecture?

- the cost grows faster than linearly
- limited fault tolerance
- higher latency

What's shared-nothing architecture? Cons & Pros



#1 Replication

#2 Partitioning

The two common ways to distribute data across multiple nodes

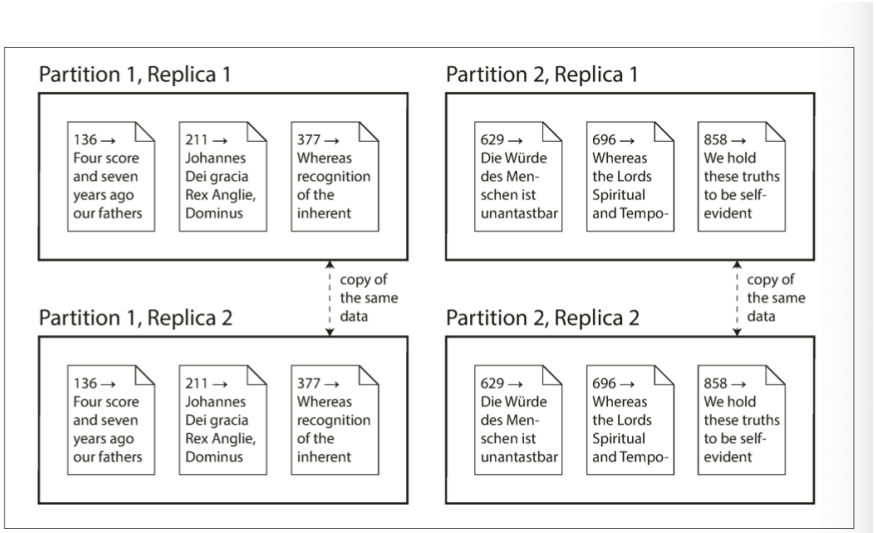


Figure II-1. A database split into two partitions, with two replicas per partition.

📌 **SUMMARY: To achieve scalability, fault tolerance, and low latency, it is necessary to use distributed databases.**

Date: @March 13, 2023 9:57 PM

Topic: Chapter 5: Replication

Recall

Why replication?

Notes

- Scalability: workload grows bigger than a single machine
- Fault tolerance/high availability
- Latency

📌 **SUMMARY:**
Replication means keeping a copy of the same data on multiple machines that are connected via a network. All of the difficulty in replication lies in handling *changes* to replicated data. This chapter discusses three commonly used approaches: *single-leader*, *multi-leader*, and *leaderless* replication

---

Date: @March 13, 2023 10:05 PM

Topic: Leaders & Followers

Recall

How do we ensure that all the data ends up on all the replicas with multiple replicas?

Synchronous and Asynchronous replication

Notes

- Pick up a leader node. Write have to happen on this node. Other replicas are named followers.
- Whenever the leader writes new data, it generates a replication log or change stream and then sends them to followers.
- When a client wants to read from the database, it can query either from the leader or from any of the followers.

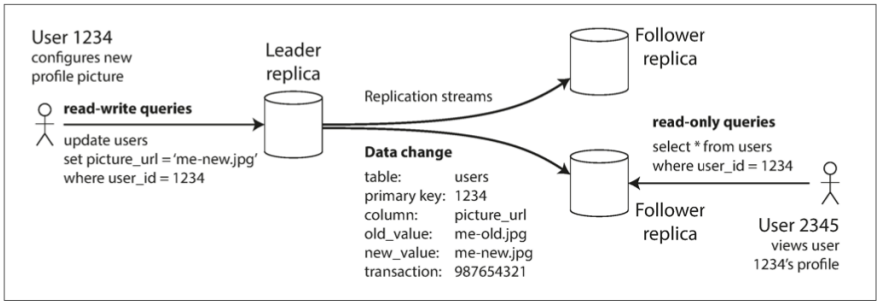


Figure 5-1. Leader-based (master–slave) replication.

Cons & Pros of Synchronous replication

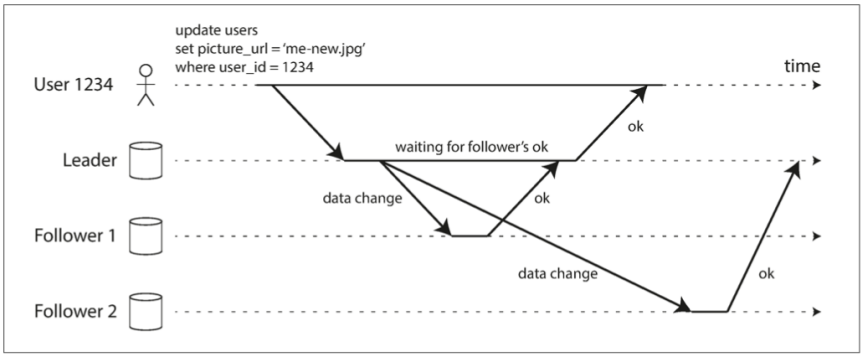


Figure 5-2. Leader-based replication with one synchronous and one asynchronous follower.

Cons & Pros of Asynchronous

cons: up-to-date copy

pros: write depends on followers

Why completely asynchronous is widely used nevertheless it has weakening durability guarantees.

pros: write is not guaranteed to be durable. incomplete replication possibly lost in the case of the leader outage.

cons: leader con continue processing writes even if all of its followers have fallen behind.

📌 **SUMMARY:**

Each node of the distributed database is called a *replica.* It is impractical for all followers to be synchronous: any one node outage would cause the whole system to grind to a halt. In practice, if you enable synchronous replication on a database, it usually means that *one* of the followers is synchronous, and the others are asynchronous(*semi-synchronous)*.

Date: @March 14, 2023 8:55 PM

Topic: Setting Up New Followers & Handling Node Outages

Recall	Notes
How do we ensure the newly added follower has an intact copy of the leader's data without downtime?	1. take a consistent snapshot 2. copy the snapshot to the new follower 3. `binlog` coordinates: get replication log of delta changes 4. caught up.
How do we handle follower nodes outage?	**Catch-up recovery** • keeps durable change logs • when crashed or network issue recovery from its log
How do we handle leader node outages?	**Failover** 1. *Determining that the leader has failed (timeout)* 2. *Choosing a new leader (run election process,* consensus problem*)* 3. *Reconfiguring the system to use the new leader. Clients send writes to the new leader.*
	1. inconsistent data, the new leader may not have received all the writes from the old leader with an asynchronous replication approach. → semi-synchronous replication

Probably problems with Failover?	2. discarding writes is bad. Caused primary key conflicts in the incident at GitHub. 3. split brain: two nodes both believe they are the leader. consensus issue 4. `timeout` to declare the leader dead is not deterministic. Too long or too short.

📌 **SUMMARY:**
Our goal is to keep the system as a whole running despite individual node failures and to keep the impact of a node outage as small as possible.
When the leader is failed, one of the followers needs to be promoted to be the new leader, clients need to be reconfigured to send their writes to the new leader, and the other followers need to start consuming data changes from the new leader. This process is called ***failover***.
These issues—node failures; unreliable networks; and trade-offs around replica consistency, durability, availability, and latency—are in fact fundamental problems in distributed systems. There are no easy solutions to there problems. We have to discuss them in greater depth later.

Date: @March 14, 2023 9:40 PM

Topic: Implementation of Replication logs

Recall	Notes
How does leader-based replication work under the hood?	#1 Statement-base replication. #2 Write-ahead log (WAL) shipping: storage versions must be identical #3 Logical(row-based) log replication: transaction log, CDC(change data capter) MyQL version greater than 5.1 binlog

	#4 Trigger-based replication: greater overheads.
What's statement-based replication? And what are the problems?	Definition: the leader sends write statements to its followers. The followers execute the statement as if they had been received from a client. Problems: NOW(), RAND() functions; execution order;

📌 **SUMMARY:**
TODO

Date: @March 15, 2023 9:01 PM

Topic: Problems with Replication lag

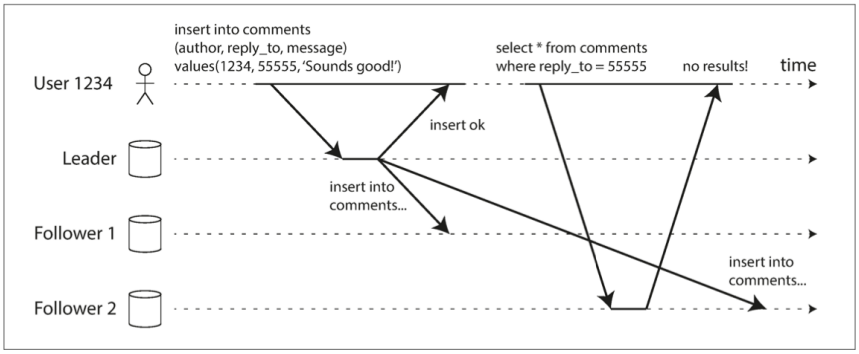
Recall	Notes
Eventual consistency	The leader and followers may be inconsistent at a moment, however, as long as stop writing and wait a while, the followers will eventually catch up and become consistent with the leader. This effect is known as eventual consistency.
Replication lag	the delay between a write happening on the leader and being reflected on a follower
The approaches to solving	1. Reading your own writes 2. Monotonic Reads

inconsistencies? 3. Consistent Prefix Reads

📌 **SUMMARY:**
In the read-scaling architecture, by improving the read throughput, we just create many followers and distribute the read requests across those followers. However, this approach only works with asynchronous replication as a single node failure would make the entire system unavailable for writing. Unfortunately, asynchronous replication leads to apparent inconsistencies in the database.

Date: @March 15, 2023 9:12 PM

Topic: Reading Your Own Writes

Recall	Notes
Asynchronous replication inconsistency issue	Read-after-write consistency is not guaranteed!
How can we implement read-after-write consistency?	 *Figure 5-3. A user makes a write, followed by a read from a stale replica. To prevent this anomaly, we need read-after-write consistency.*
	Relevant technicals

1. read it from the leader if the data possibly have been modified
2. Read from leader or followers based on the last write time.
3. Clients remember the most recent write timestamp

📌 **SUMMARY:**
read-after-write consistency, also known as read-your-own-writes consistency is a guarantee that the user will always see any updates they submitted if they reload the page.
cross-device read-after-write consistency: the user should see identical data.

Date: @March 15, 2023 9:36 PM

Topic: Monotonic Reads

Recall

moving backward in time anomaly

How can implement Monotonic Reads consistency?

Notes

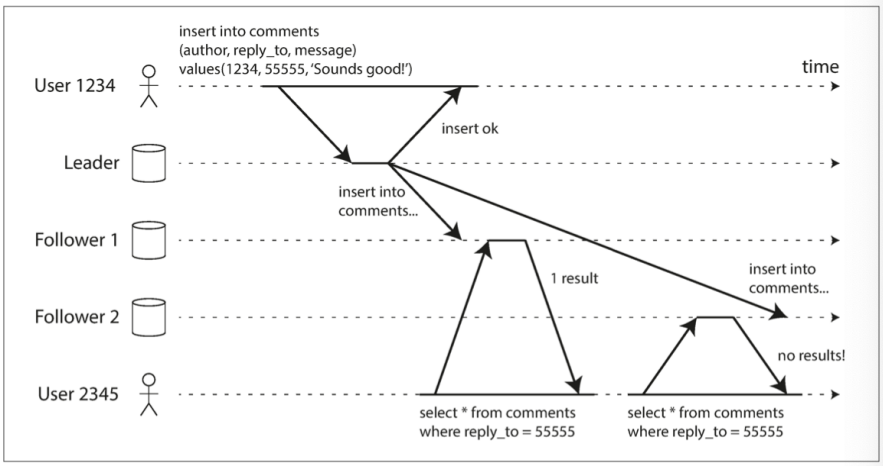


Figure 5-4. A user first reads from a fresh replica, then from a stale replica. Time appears to go backward. To prevent this anomaly, we need monotonic reads.

#1. always make reads from the same replica

#2 … (not mentioned in the book)

📌 **SUMMARY:**
If the user makes several reads from different replicas, the user can see the response change randomly. This is known as *moving backward in time* **anomaly.**

By solving the moving backward in time anomaly, we have to guarantee Monotonic reads consistency. When you read data, you may see an old value; monotonic reads only means that if one user makes several reads in sequence, they will not see time go backward— i.e., they will not read older data after having previously read newer data

---

Date: @March 15, 2023 9:44 PM

Topic: Consistent Prefix Reads

Recall

violation of causality

Notes

Expected:

Mr. Poons
How far into the future can you see, Mrs. Cake?

Mrs. Cake
About ten seconds usually, Mr. Poons.

Got: (Answer comes first violation to the causality)

The root cause of causality violation.

Mrs. Cake
About ten seconds usually, Mr. Poons.

Mr. Poons
How far into the future can you see, Mrs. Cake?

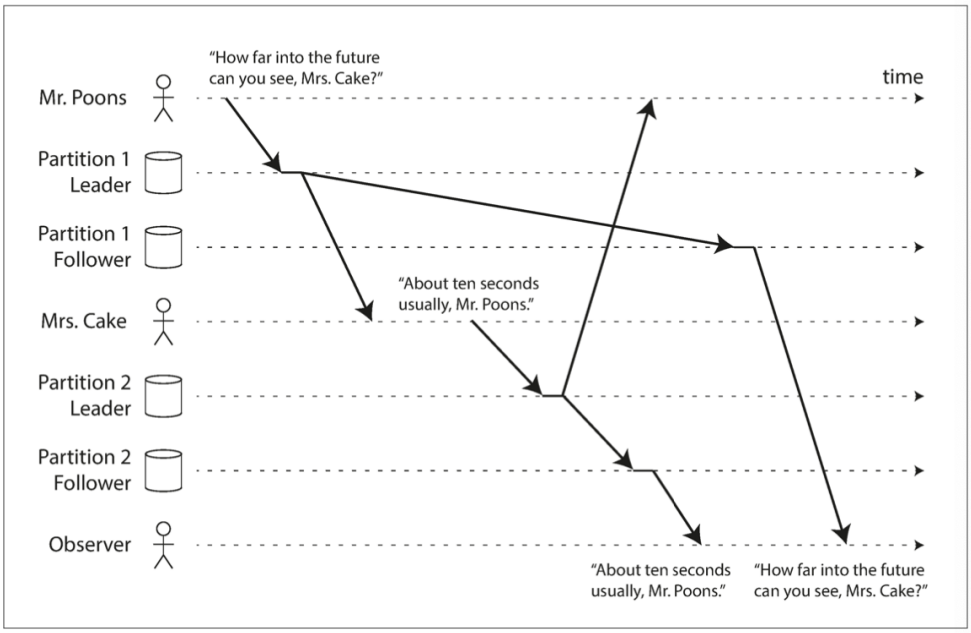


Figure 5-5. If some partitions are replicated slower than others, an observer may see the answer before they see the question.

📌 **SUMMARY:**
With replication lag, a causality violation could occur. E.g. In a dialog, the answer comes first before the question.
To avoid such violations, consistent prefix reads guarantee should be implemented.
This guarantee says that if a sequence of writes happens in a certain order, then anyone reading those writes will see them appear in the same order. (Reads in the same order with writes happened order)

Date: @November 5, 2019

Topic: Solutions for Replication Lag

📌 **SUMMARY:**
When working with an eventually consistent system, it is worth thinking about how the application behaves if the replication lag increases to several minutes or even hours. If the answer is “no problem,” that’s great. However, if the result is a bad experience for users, it’s important to design the system to provide a stronger guarantee, such as read-after-write.

Date: @March 15, 2023 10:11 PM

Topic: Multi-Leader Replication

Recall	Notes
Downsides of sinder leader	• write is unavailable if the network between clients and leads is broken • low efficiency with multiple data centers
Comparison between single-leader and multi-	1. performance: M > S 2. tolerance of data center outages

leader configurations

3. tolerance of network problems: M > S

📌 **SUMMARY:**
Multi-leader replication across multiple data centers.

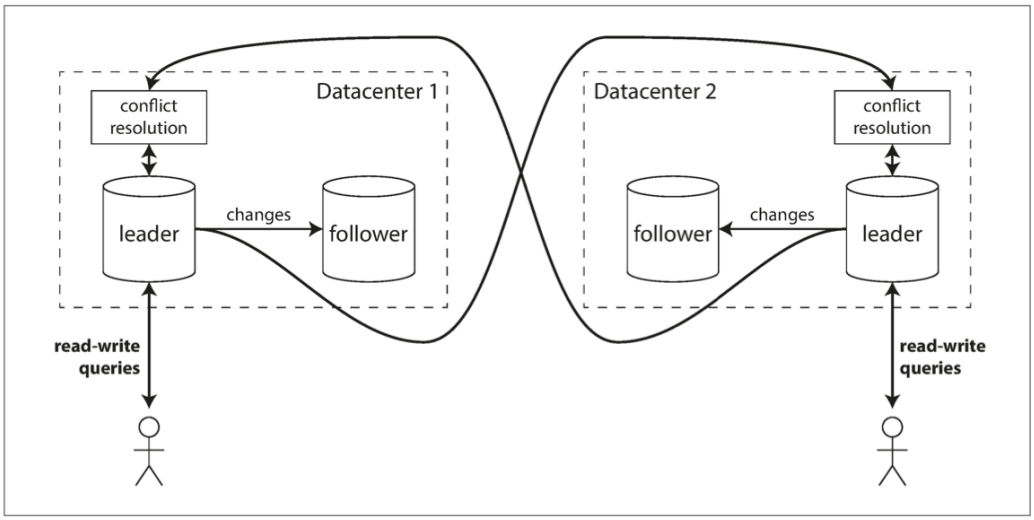


Figure 5-6. Multi-leader replication across multiple datacenters.

the big downside of multi-leader replication is: write conflicts. The same data may be concurrently modified in two different data centers, and those write conflicts must be resolved.

Date: @March 16, 2023 9:12 PM

Topic: Clients with offline operation

📌 **SUMMARY:**
The situation in which multi-leader replication is appropriate is if you have an application that needs to continue to work while it is disconnected from the internet. CouchDB is designed for this mode of operation.
Google Docs, allow multiple people to edit a text document or spreadsheet simultaneously. This kind of application is required real-time collaborative editing.

Date: @March 16, 2023 9:17 PM

Topic: Handling Write Conflicts

Recall	Notes
How do we detect write conflicts?	In a single-leader database, write conflicts are prevented by transactions. In the multi-leader database, write conflicts are detected at some later point in time when it may be too late to ask the user to resolve the conflicts.
Conflict avoidance	make sure all writes for a particular record go through the same leader.
Converging toward a consistent state	#1 Last write wins(LWW): assign each write a unique ID, and pick the write with the highest ID as the winner when conflicting. #2 Assign each leader replica a priority unique ID. Take precedence according to the priority ID.

	#3 Somehow merge the conflicts #4 Store the conflict in a compatible structure. Postpone conflict resolution at some later time by the user.
Custom conflict resolution logic	Application resolves the conflicts when reading or writing. Case on read: The application may prompt the user to resolve the conflicts and write the result back to the database. (CouchDB) Case on write: Calling conflict handler as soon as db detects a conflict. (Bucardo)

📌 **SUMMARY:**
Two writes concurrently modify the same object known as *conflict*. The biggest problem with multi-leader replication is that write conflicts. We have to resolve conflicts properly when they occur.
An example, consider a wiki page is simultaneously being edited by two users:

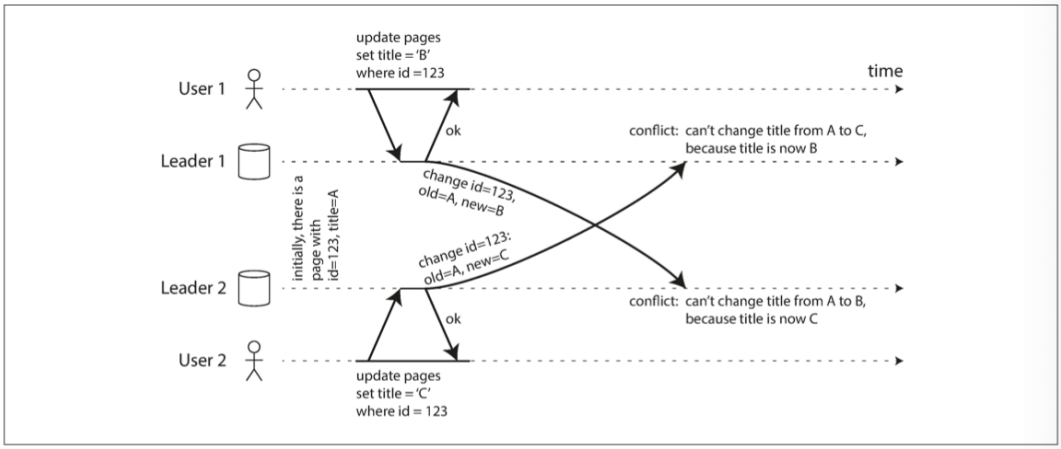


Figure 5-7. A write conflict caused by two leaders concurrently updating the same record.

Date: @March 16, 2023 9:40 PM

Topic: Multi-Leader Replication Topologies

Recall	Notes
Leaders replication topologies	#1 Circular: pay attention to infinite replication loop & node fails #2 Star: node fails #3 All-to-all: overtake or causality → version vectors

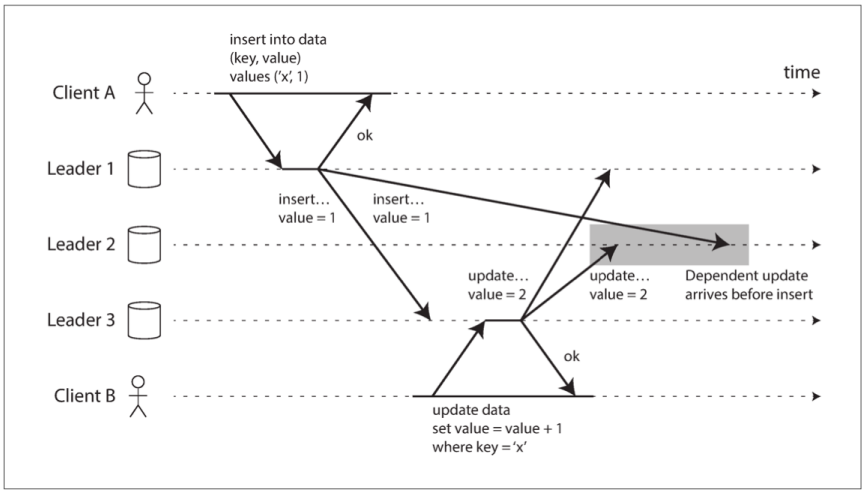


Figure 5-9. With multi-leader replication, writes may arrive in the wrong order at some replicas.

📌 **SUMMARY:**

If you have multiple leaders, you have to propagate all of its writes from one leader to another. There are many communication paths along with writes are propagated from one node to another. The most general topology is *all-to-all*.

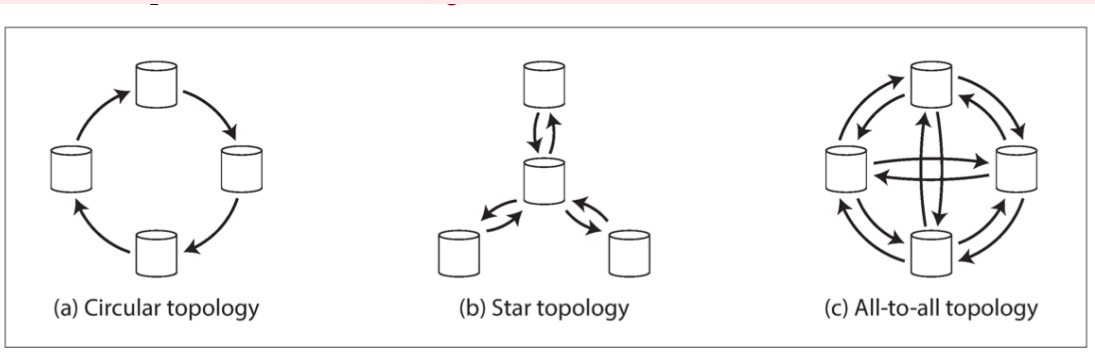


Figure 5-8. Three example topologies in which multi-leader replication can be set up.