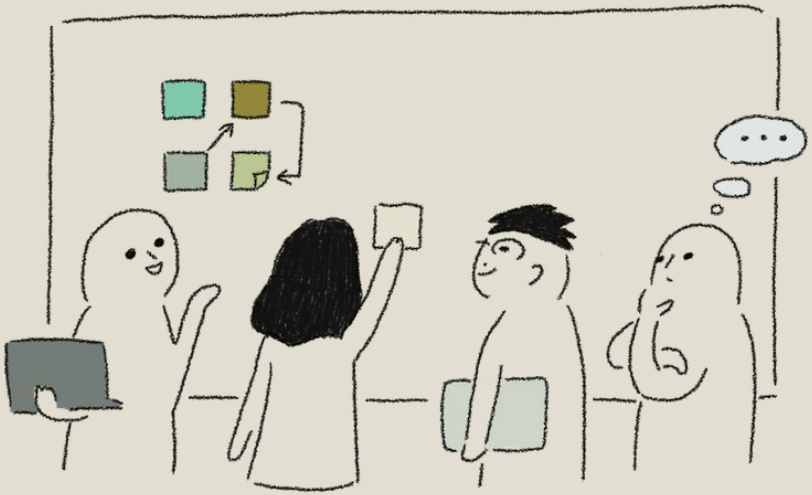
DATE 20250310

Session 2



backend

cohort #0 by open camp

#2's agenda

1. admin matters (if any)
2. databases
3. horizontal scaling
4. consensus and gossip protocols
5. w2 assignment

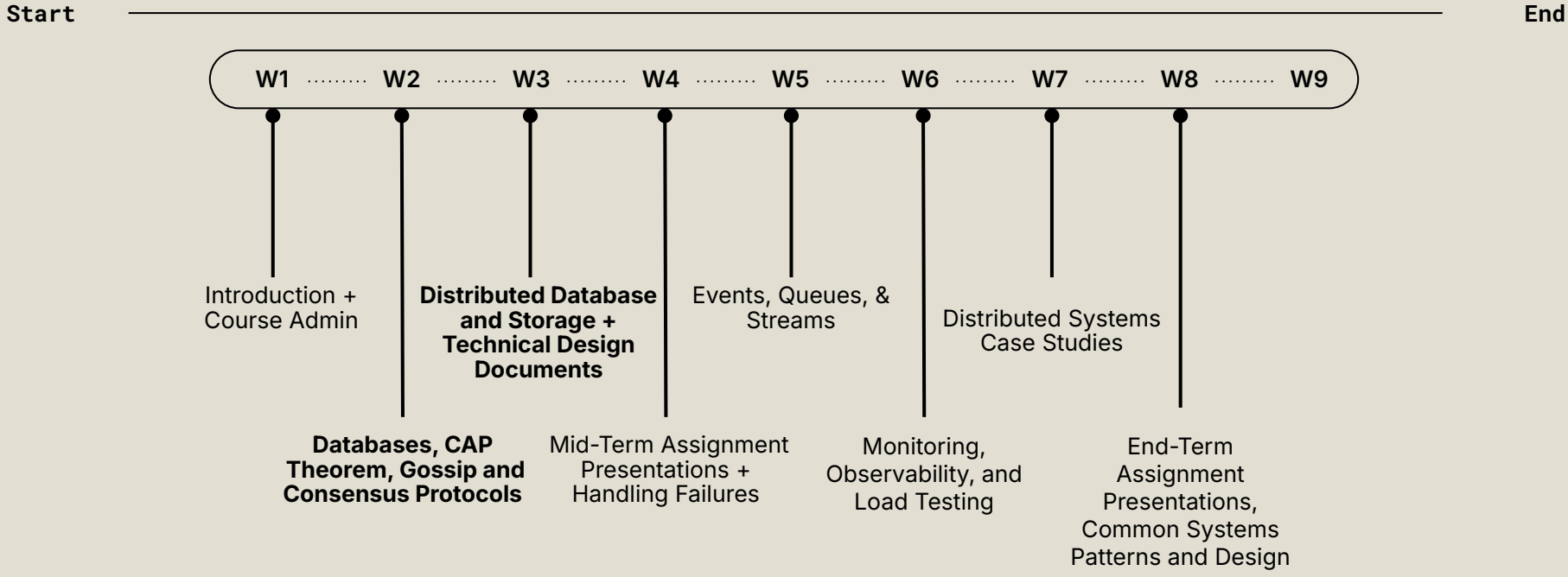
2.1 admin matters

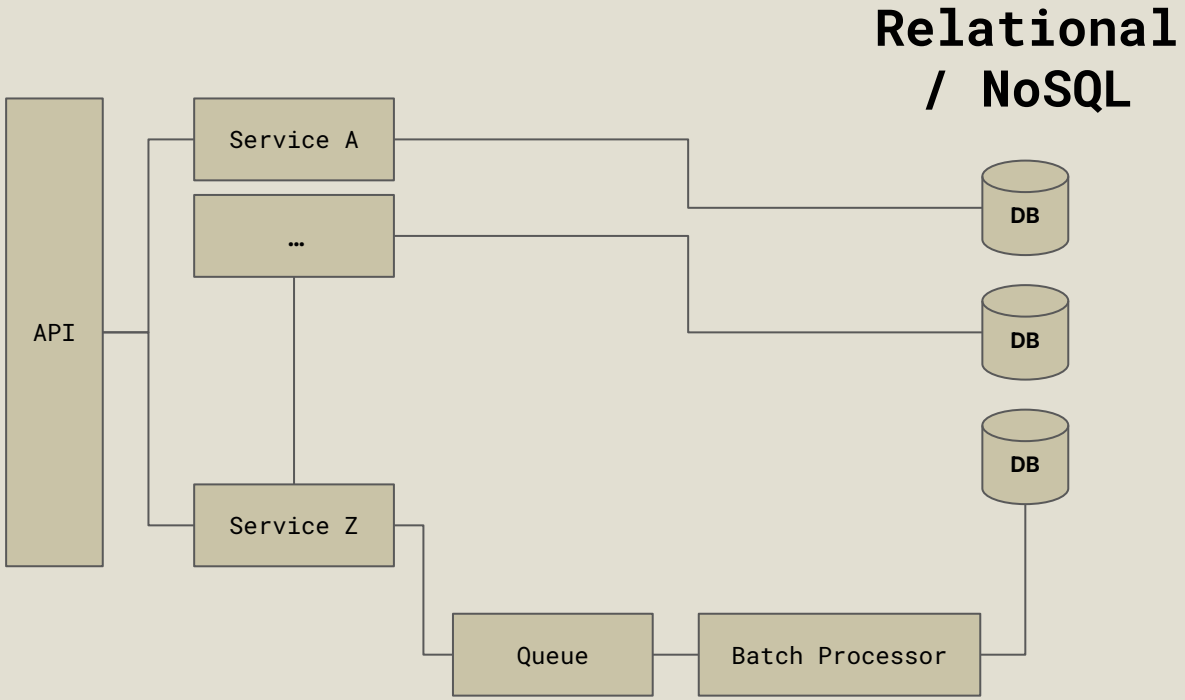
W1 Assignment

- 2 chose to review AT Protocol
  2 chose to review ActivityPub
  2 chose to review and compare both

2.2 databases

Curriculum: https://opencamp-cc.github.io/backend-curriculum/




Relational
/ NoSQL
API
Service A
...
Service Z
DB
DB
DB
Queue
Batch Processor

Flow of Data

Databases

A traditional single-instance database, by default, runs in a single location on a single machine.

Assumes 1 machine by default

Distributed Databases

“A distributed database is a database that runs and stores data across multiple computers, as opposed to doing everything on a single machine.”

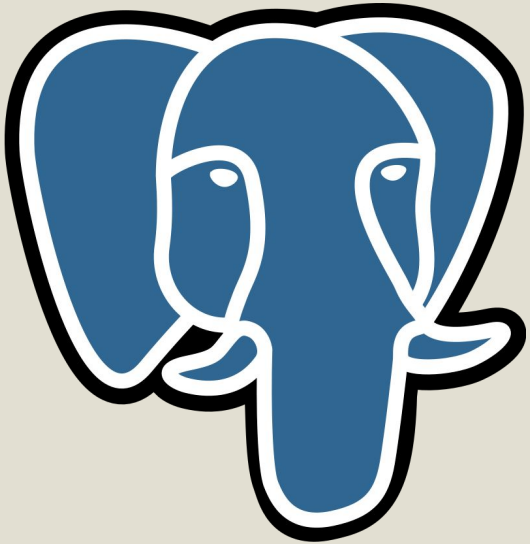
Assumes 3* or more machines by default

What is a distributed database and how do they work?
https://www.cockroachlabs.com/blog/what-is-a-distributed-database/

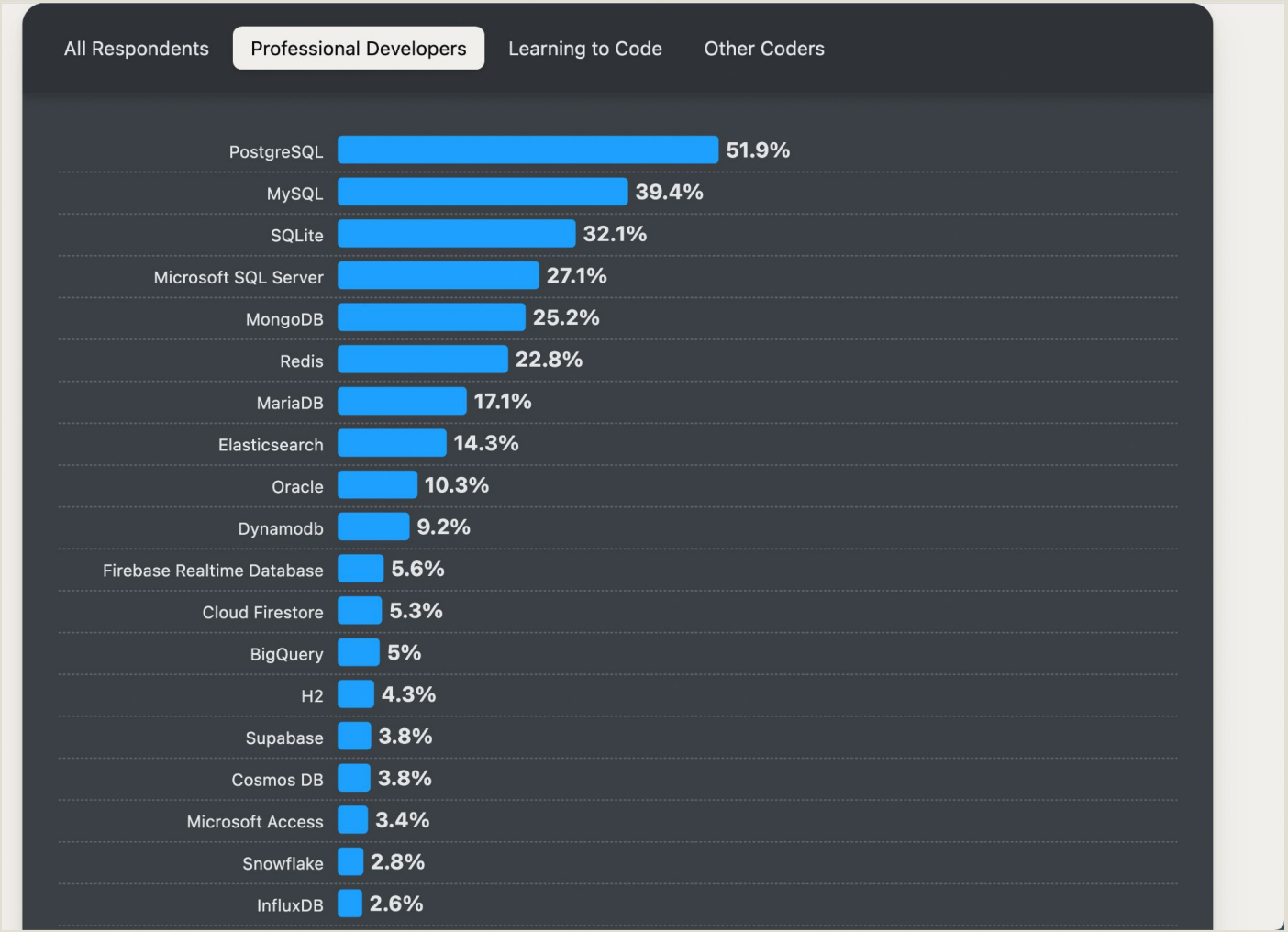
“Which database environments have you done extensive development work in over the past year, and which do you want to work in over the next year?

> 50%

That's the % of Professional Developers who have used PostgreSQL extensively in the past year and/or want to use it the following year



2024 Stack Overflow Developer Survey: https://survey.stackoverflow.co/2024/technology#most-popular-technologies-database-prof



2024 Stack Overflow Developer Survey: https://survey.stackoverflow.co/2024/technology#most-popular-technologies-database-prof

“Good Database?”

Insulated

Stainless Steel

Plastic

Warm Up

- What are some of the factors you would consider when deciding which database to use or change to?

People

Existing Skills

Training / Learning time

Resistance to Change

Talent Pool

Technology

Speed of Development

Maturity

Reliability & Consistency

Documentation

Security and Updates

Type of Data
(Relational or no)

Business

Cost (Buy / Maintenance)

Legal & Compliance

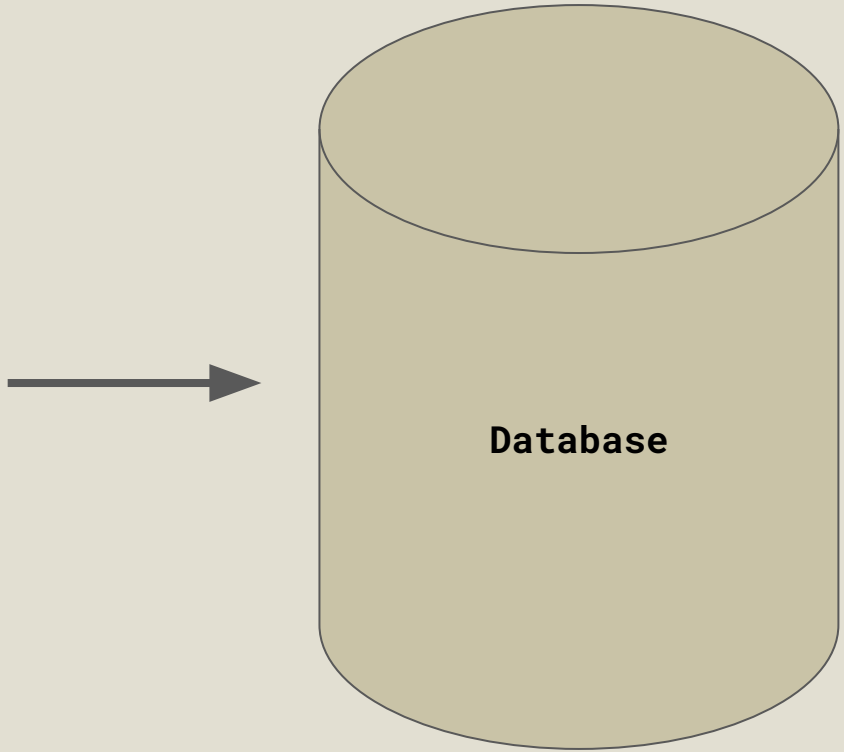
Agility

Considerations

(List is not exhaustive)

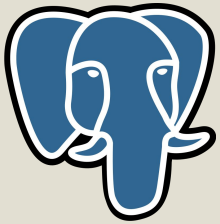
2.2 database basics

```
INSERT INTO users
VALUES (‘victor_neo’,
        ‘...’,
        …);
```

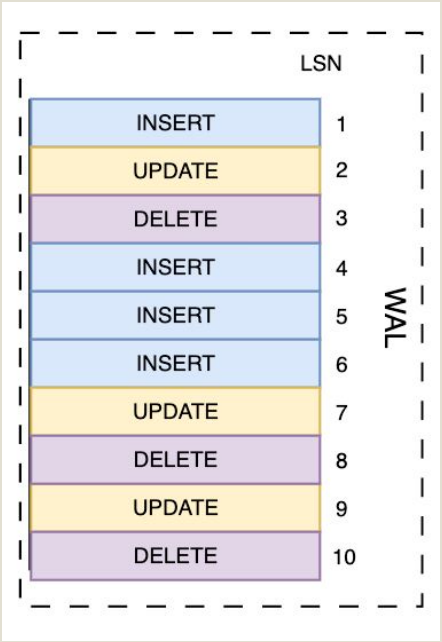


```
INSERT INTO users
VALUES (‘victor_neo’, ‘...’, …);
```

ID	Username	…
1	…	…
2	**victor_neo**	**…**



```
INSERT INTO users
VALUES ('victor_neo', '...', …);
```

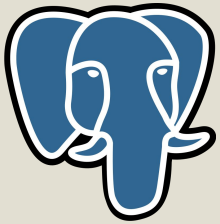


New data does not get written to the table right away!

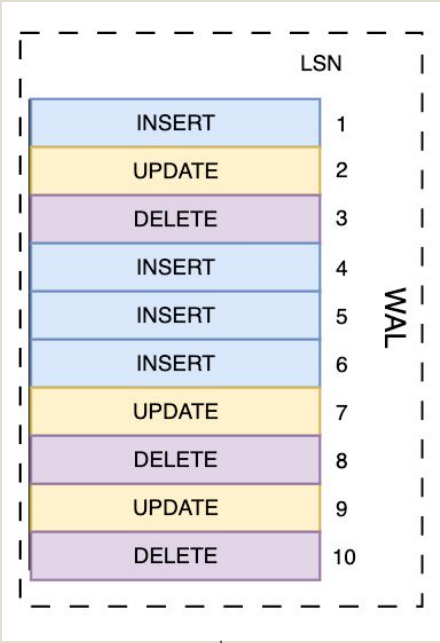
It's faster to append it to a WAL log, and say that the SQL transaction is complete.

WAL log is append-only.

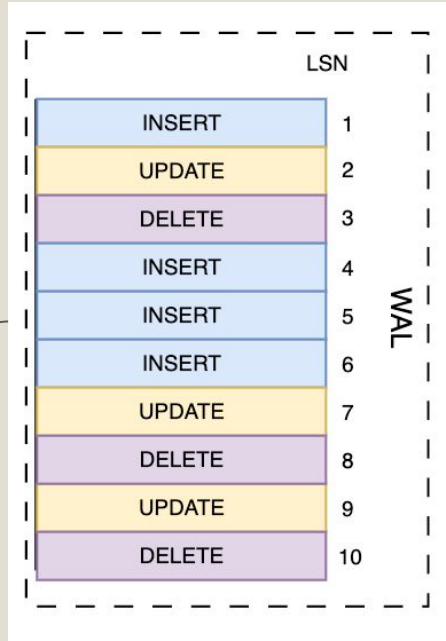
Postgres Write-Ahead Logs: https://www.artie.com/blogs/postgres-write-ahead-logs



WAL Segment 0001

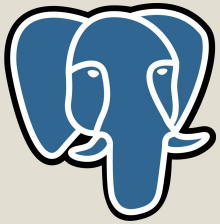


WAL Segment 0002

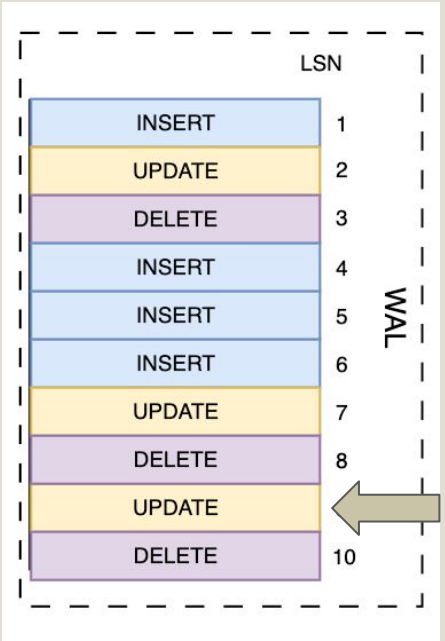


Update all changes to actual tables (commit)

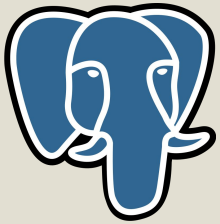
Usual WAL segment size is **16MB**
Default max_wal_size is **1GB**



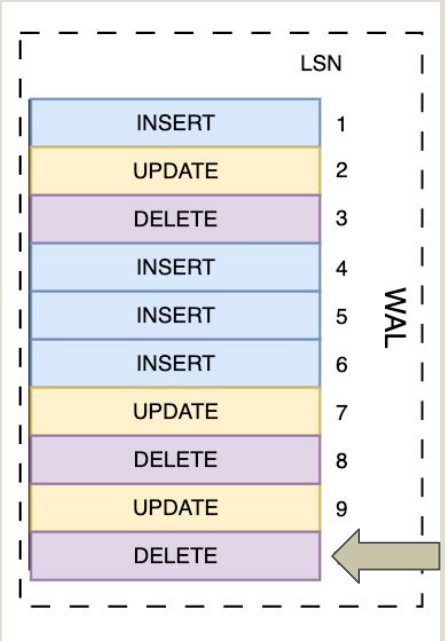
WAL Segment 0001



DELETE FROM users…

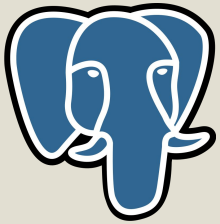


WAL Segment 0001

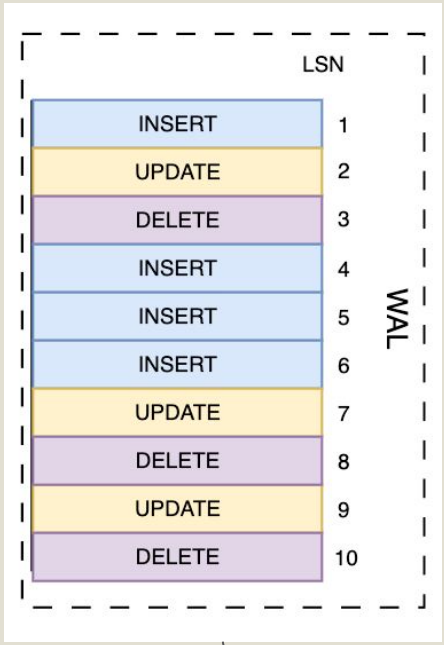


DELETE FROM users…

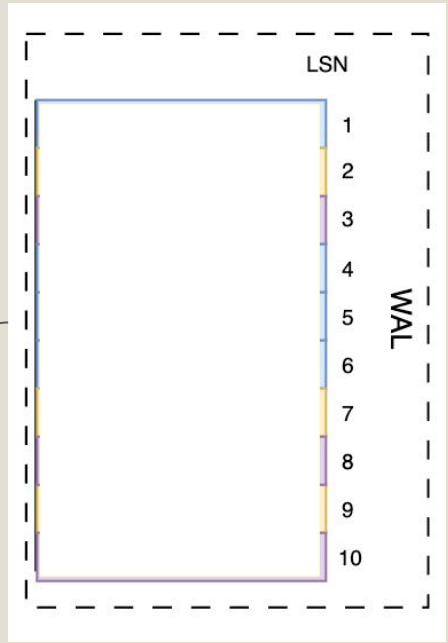
Append DELETE changes to the WAL log



WAL Segment 0001

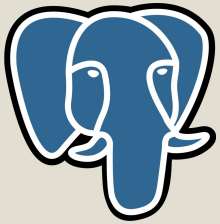


WAL Segment 0002

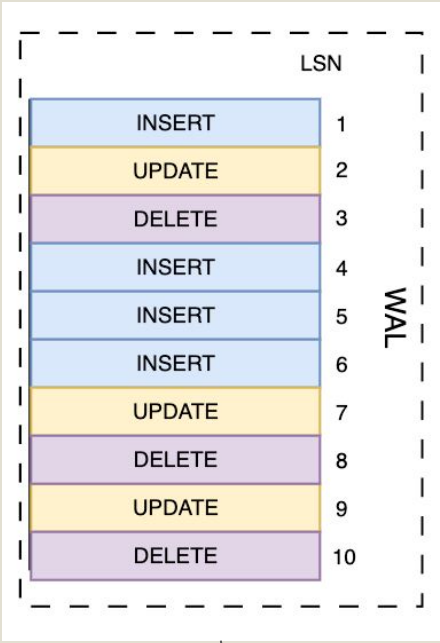


1. Write changes to tables and disk

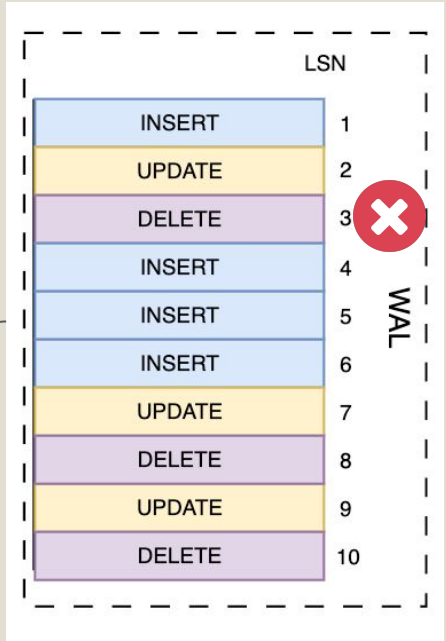
2. New WAL Segment is created after segment 1 is full



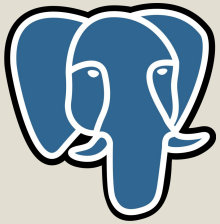
WAL Segment 0001



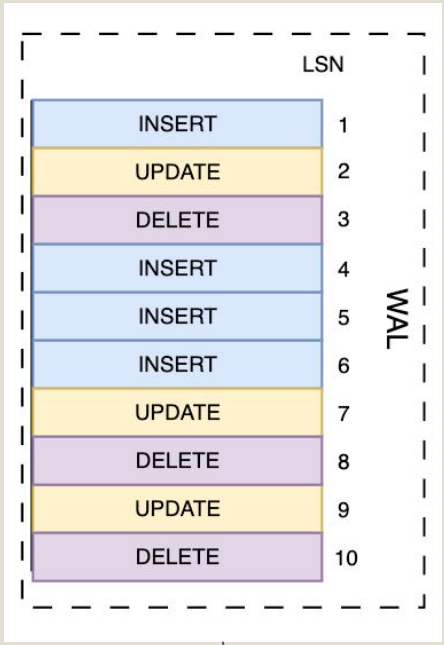
WAL Segment 0002



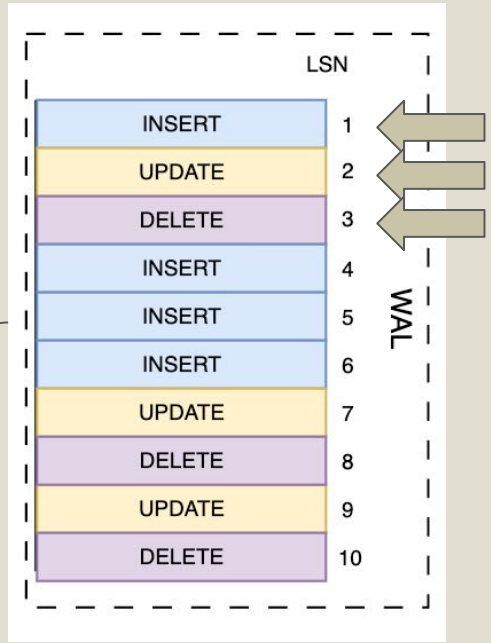
What happens when DB crashes after LSN 3 is committed?



WAL Segment 0001



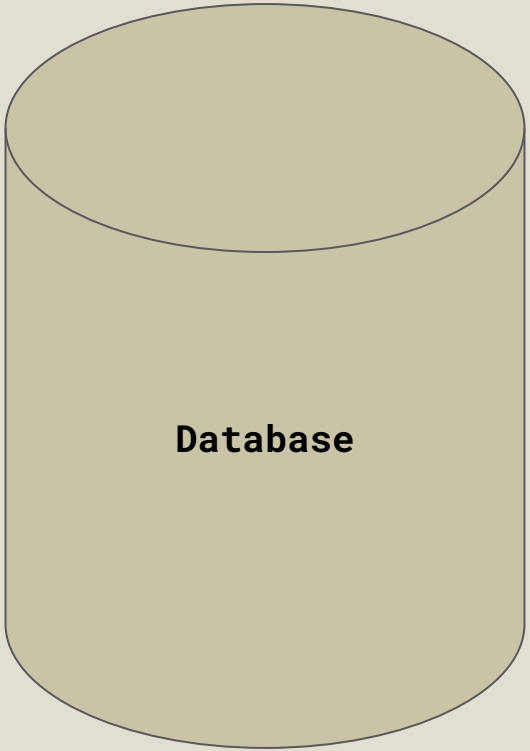
WAL Segment 0002



All changes in 0001 are committed to disk nothing to do here.

“Replay” from LSN 1, 2, and so on to recover data.

scaling databases

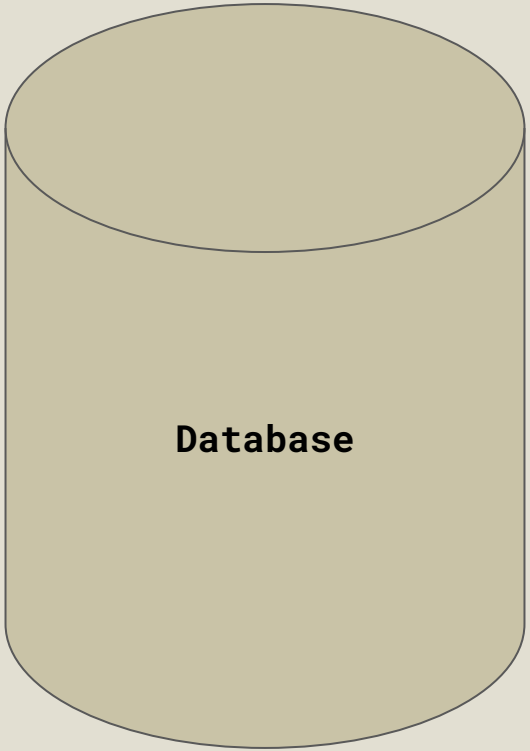

Database

Data Size

Connections

CPU Load

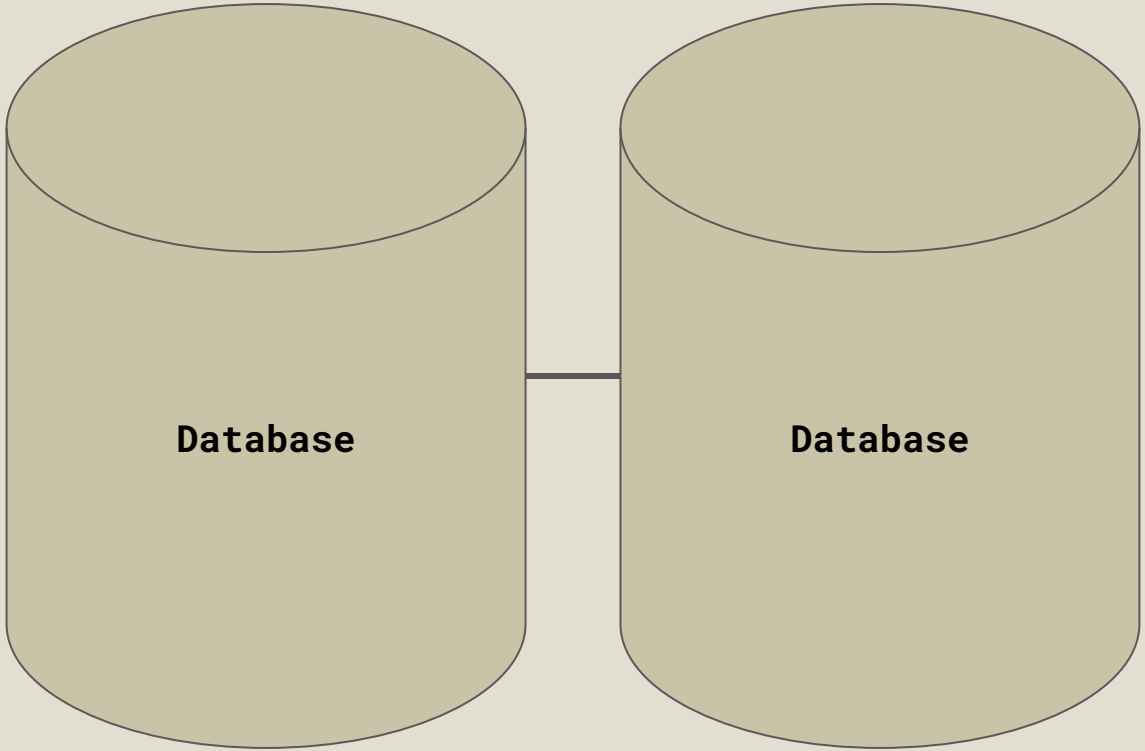
Memory Load



Writes Heavy

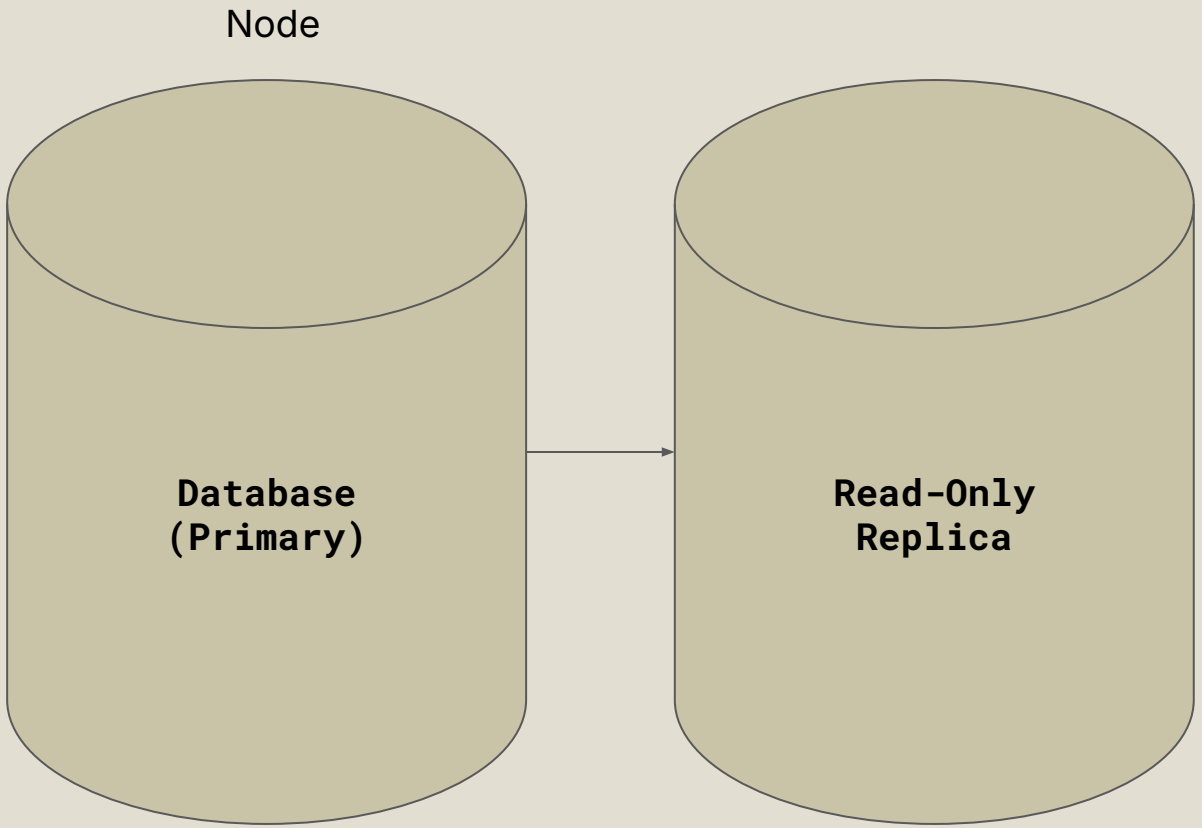
Reads Heavy

Transactional vs Warehousing

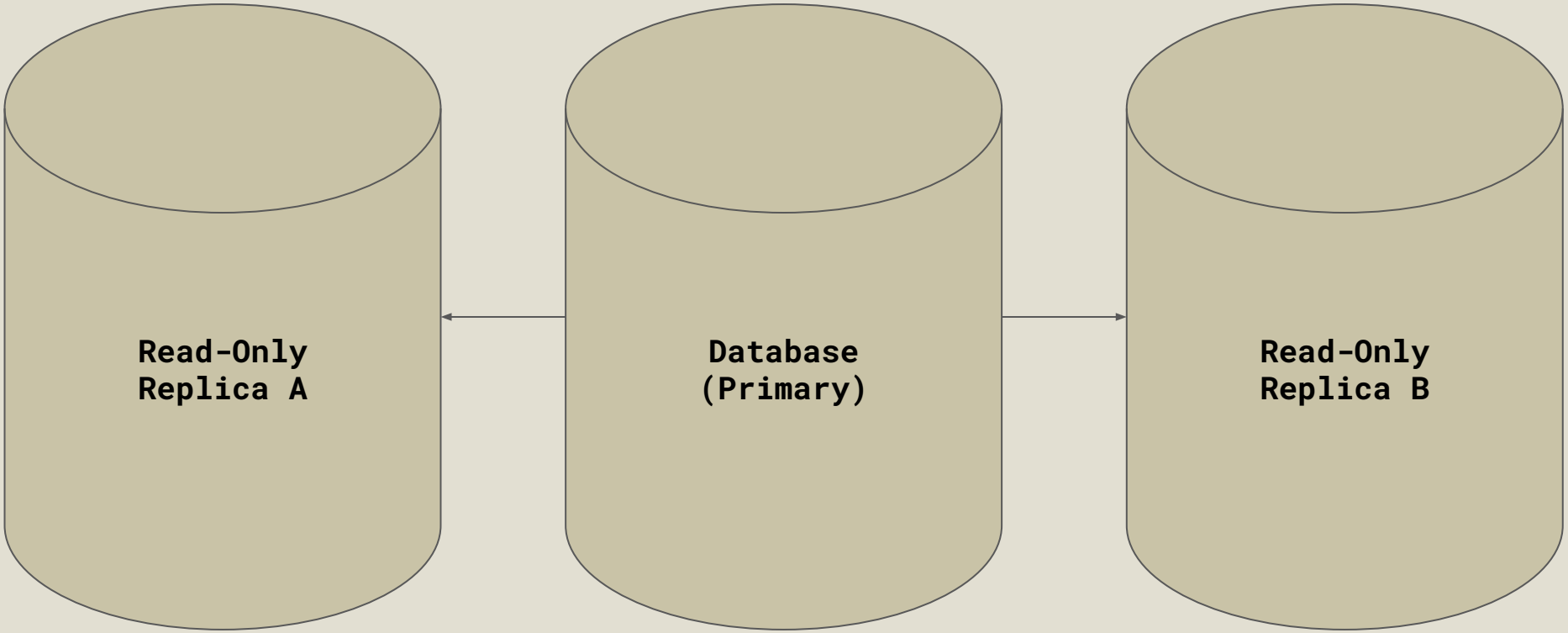


Horizontal Scaling

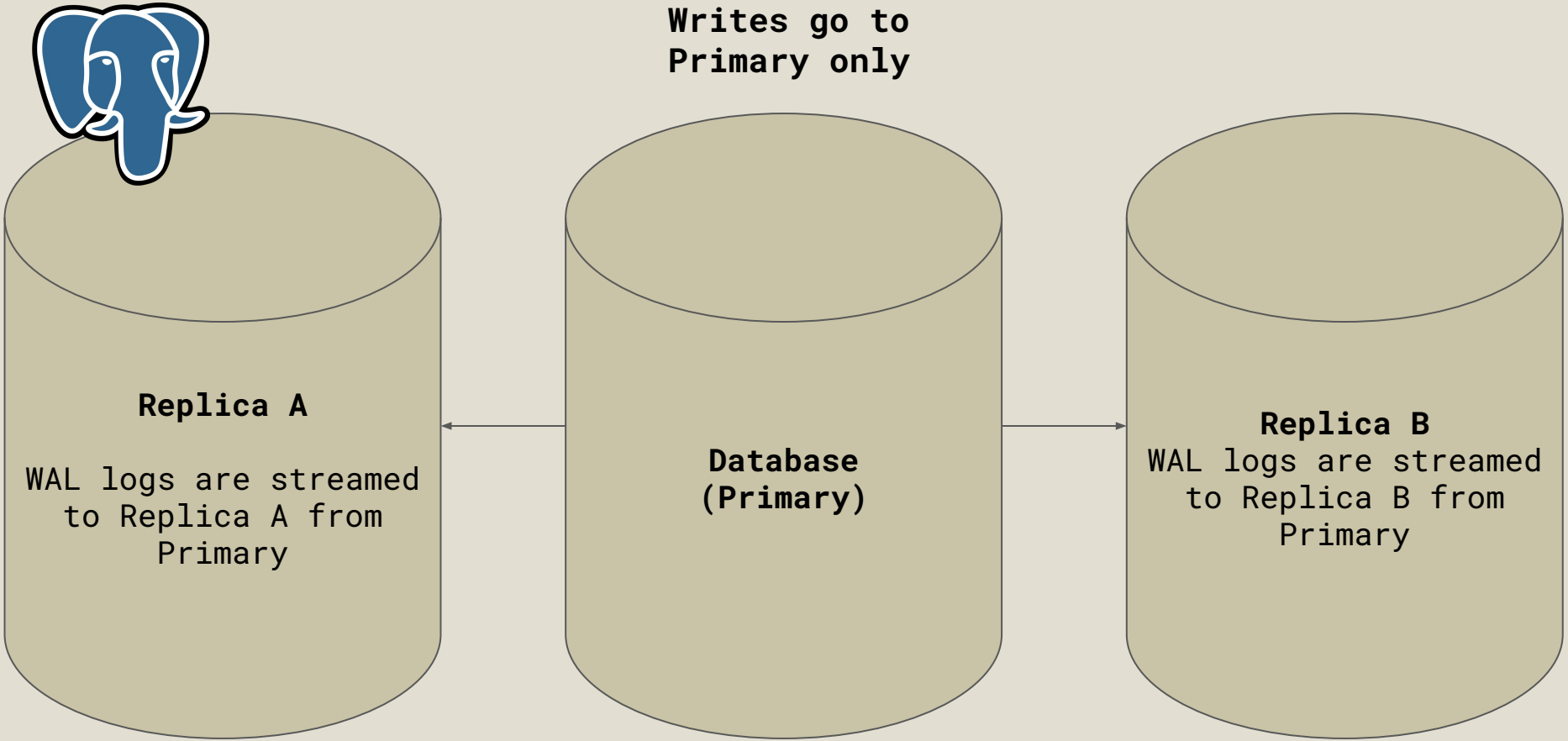
2.3 horizontal scaling



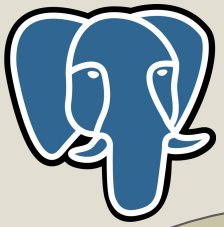
Replication



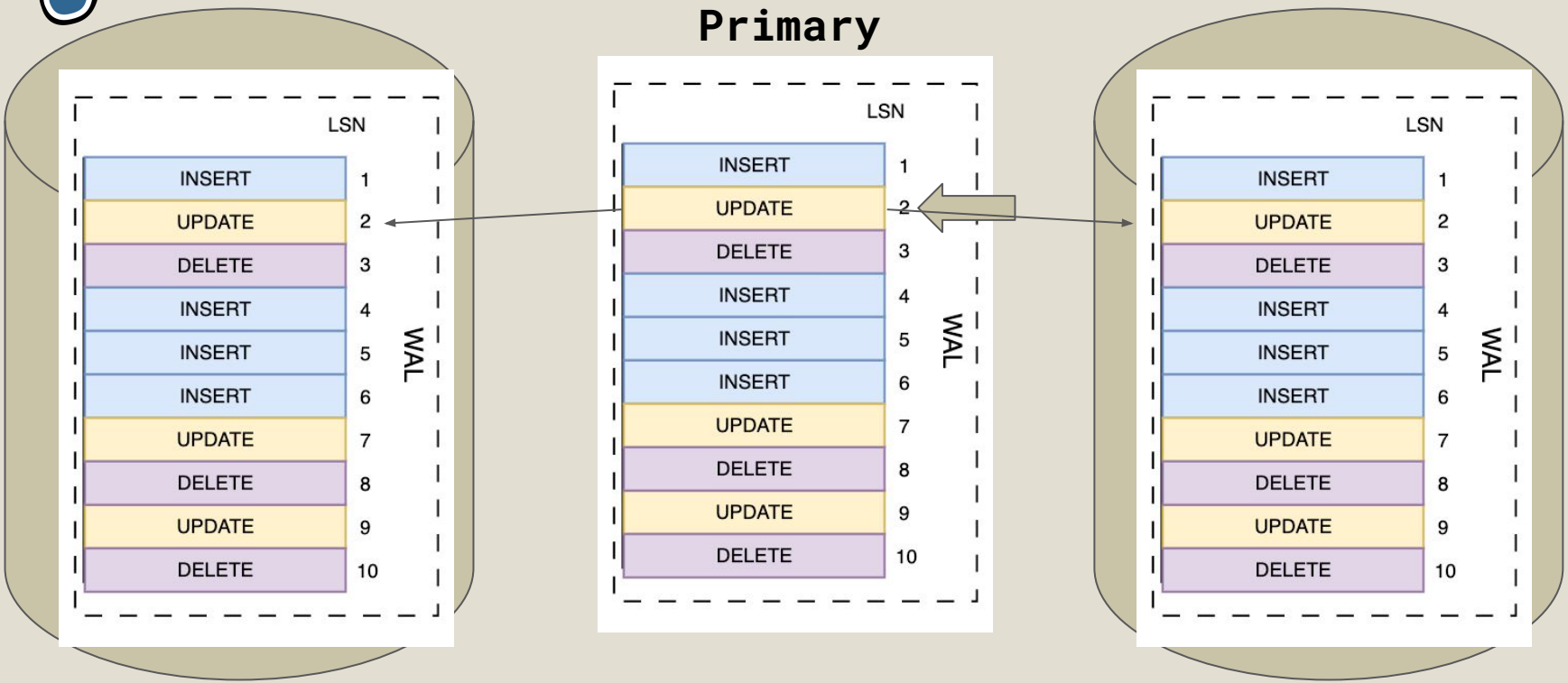
Replication



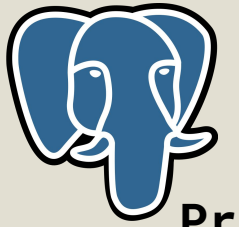
Streaming Replication



Burden is on the Primary node to keep in sync

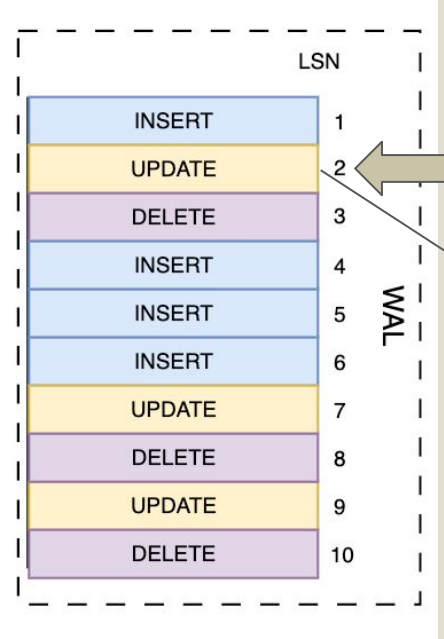


Streaming Replication



Lesser burden on Primary node, at the cost of more latency

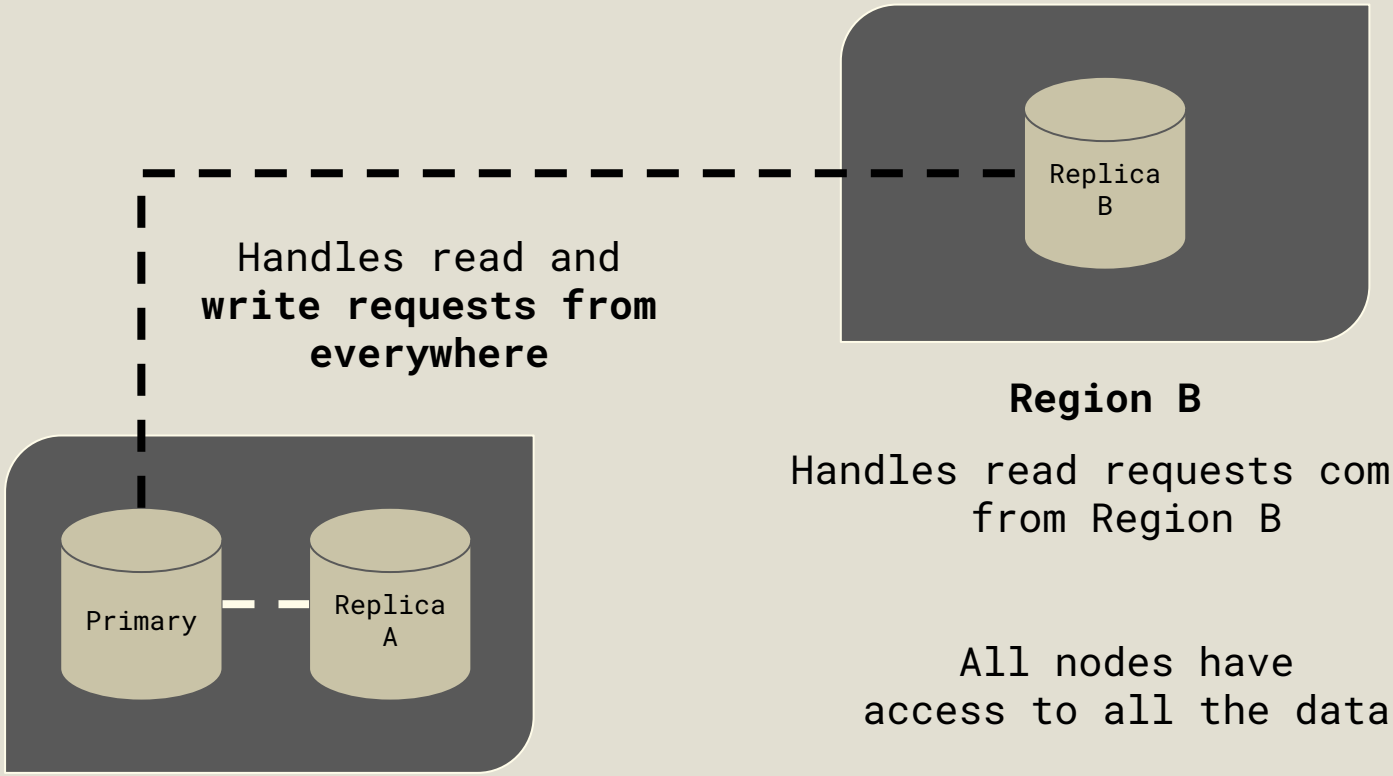
Primary



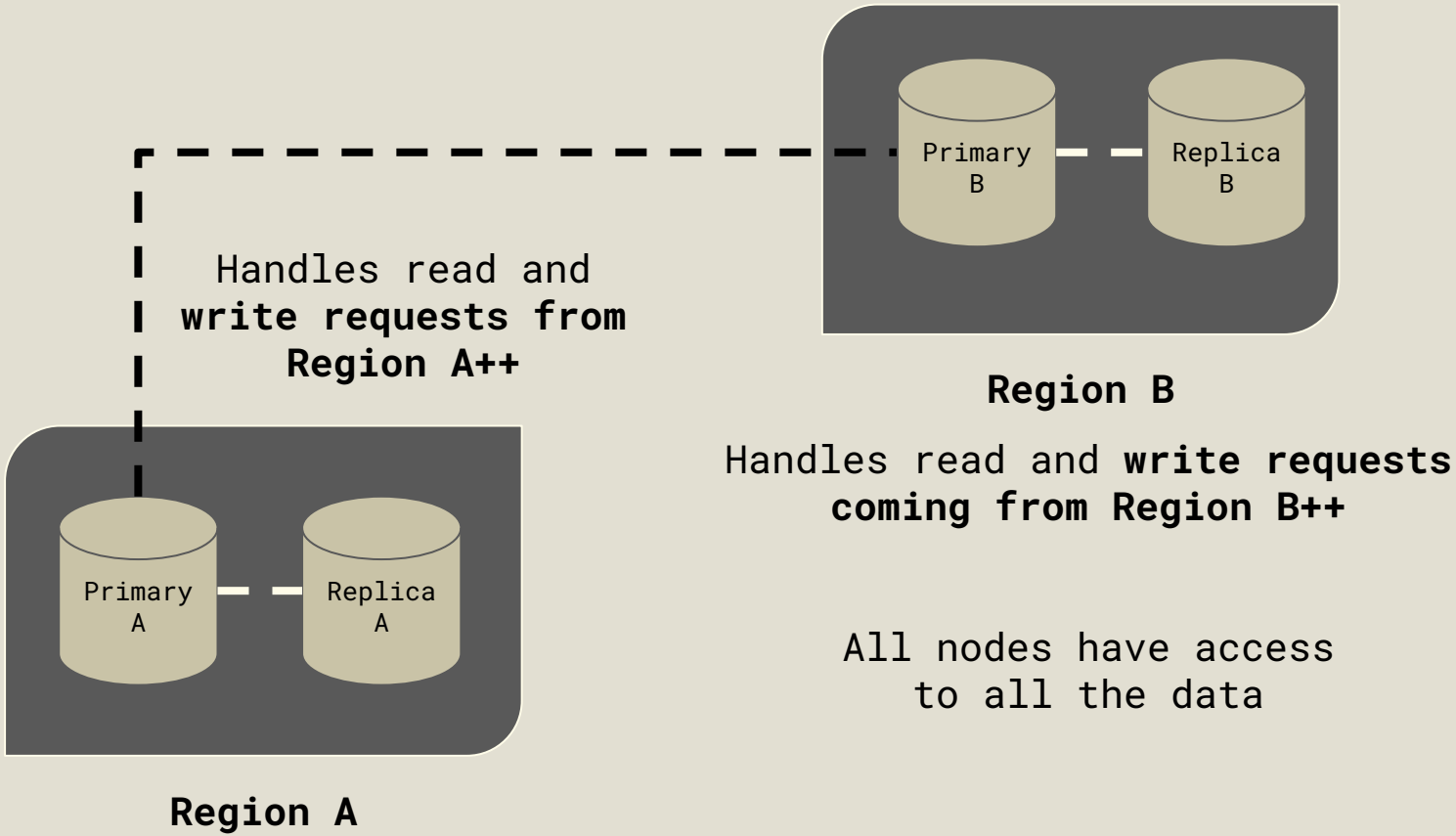
Replica A

Replica B

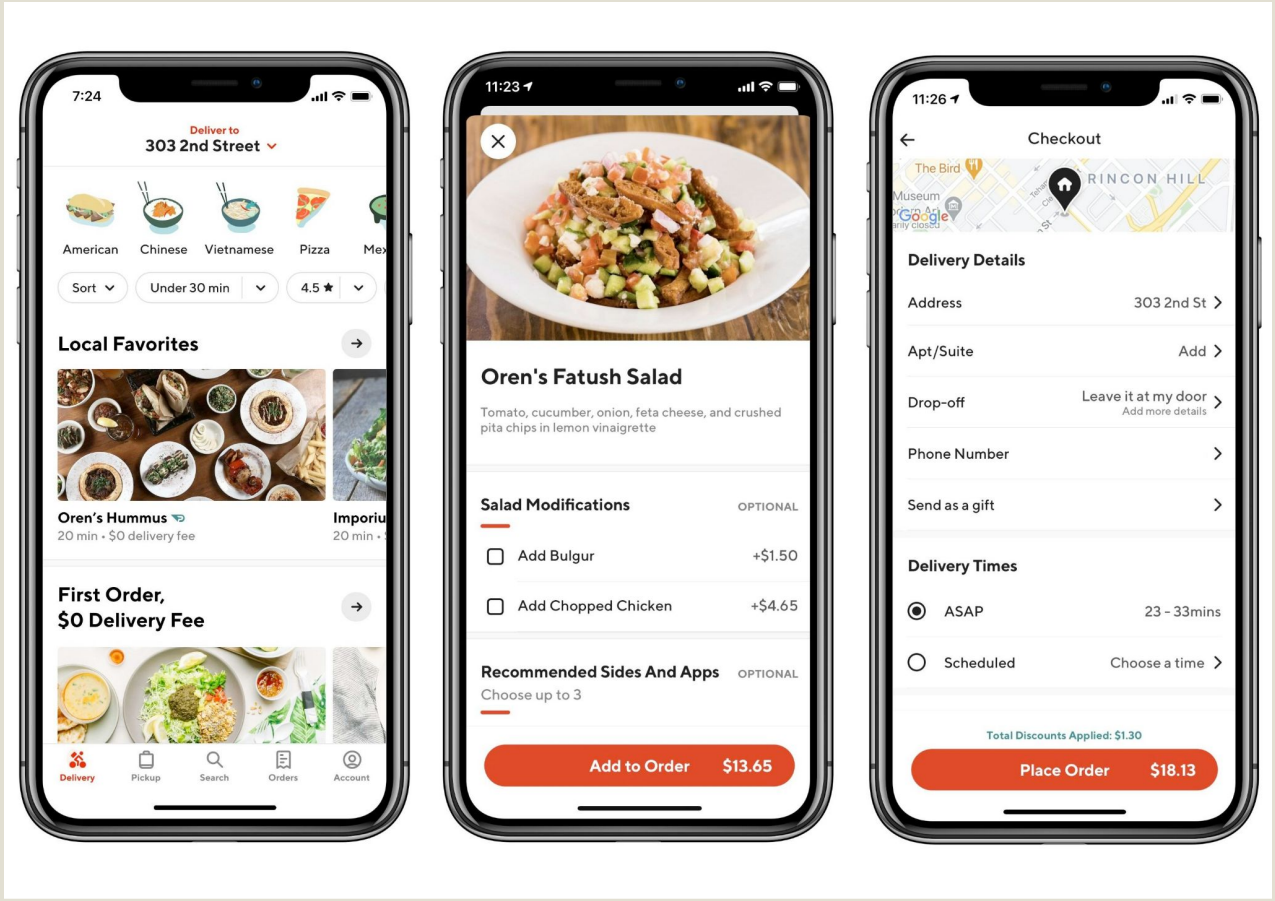
Chained Replication



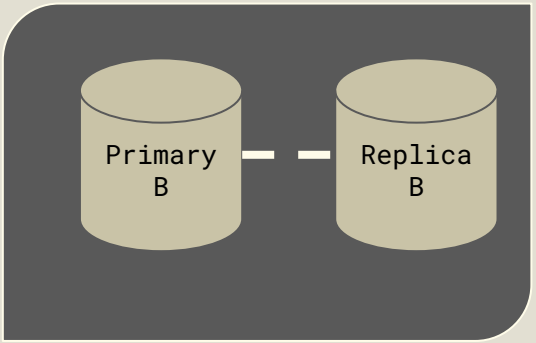
Multi-Region Reads (R)



Multi-Region Reads + Writes (RW)



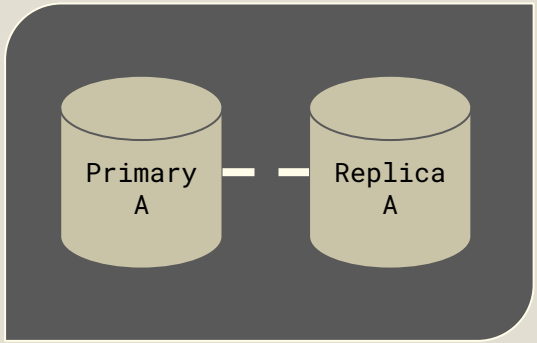
Food Delivery: Multi-Region R or RW?



Region B

Handles read and **write requests** coming from Region B++

Handles read and
write requests from
Region A++



Region A

All nodes only have data in their own region

Partitioning

Nodes only keep data they are responsible for

2.3.1 partitioning

SELECT * FROM users WHERE ID = '...'

Handles read and **write requests**
based on some **partition or routing key**

Eg. Primary = Primary ID % 2

Partitioning

Pros

- Scale each primary and replicas independently, eg. storage

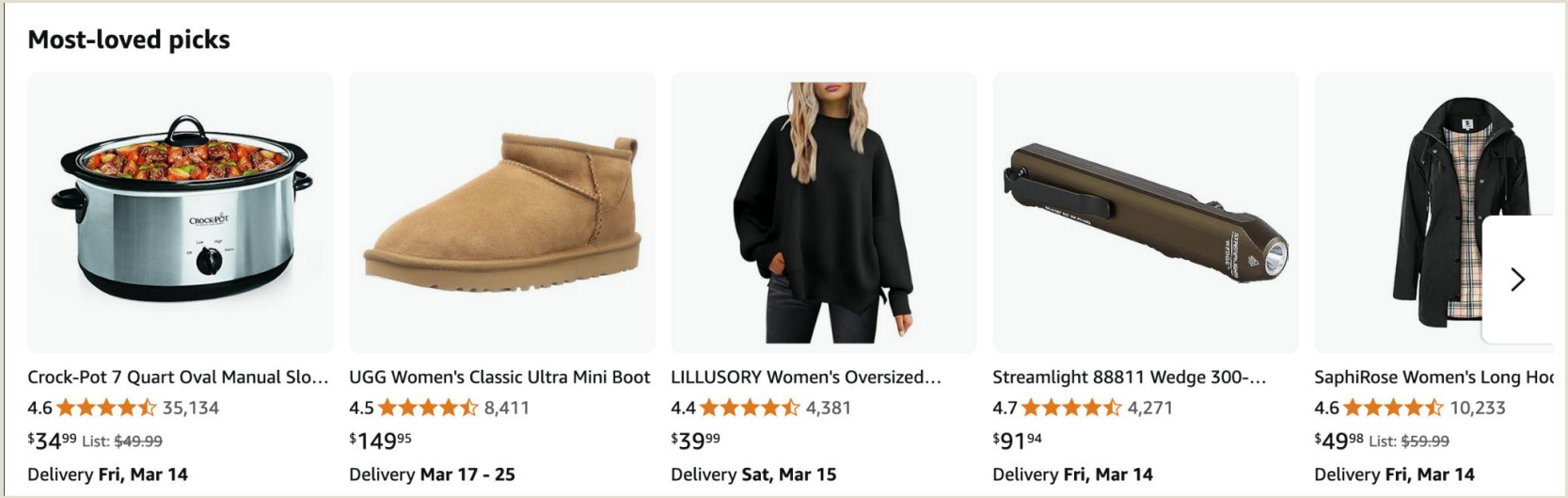
- Data can be local and situated nearer to users

Cons

- Partition key selection is crucial to avoid overloading of a set of nodes

- Disaster Recovery needs to be tested for each set of nodes

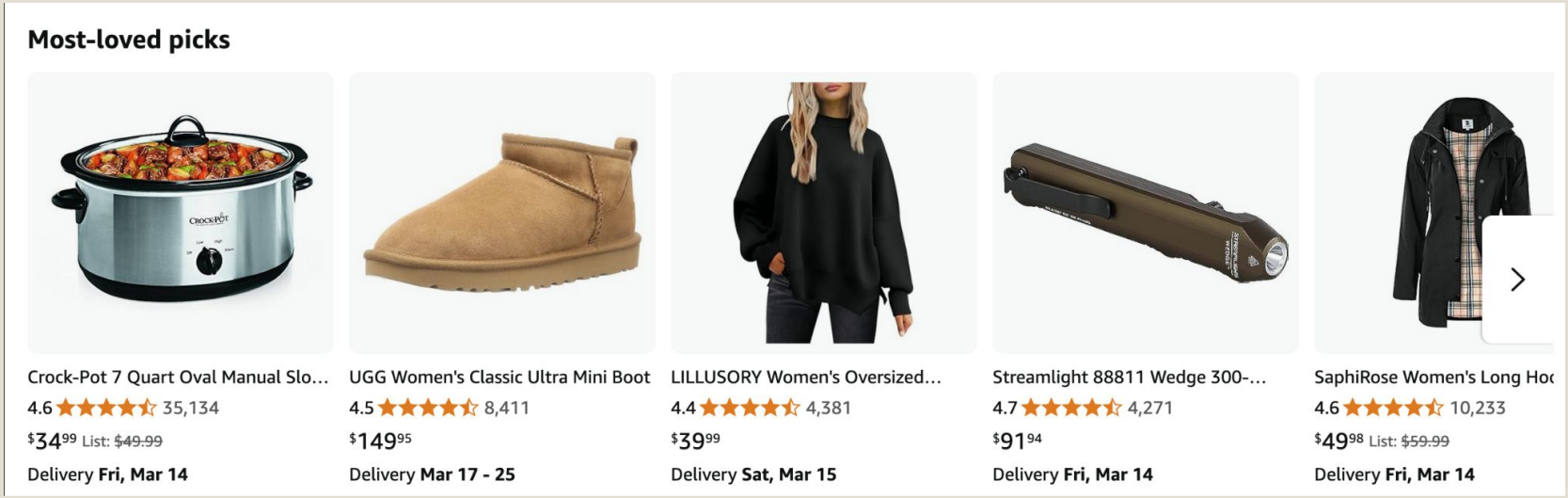
- Queries that need to span across different set of nodes are expensive



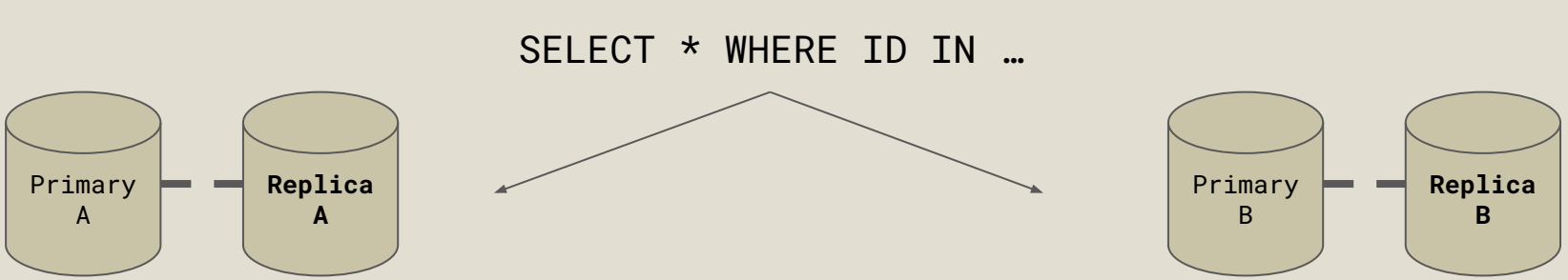
Partition Key for an E-Commerce Website?

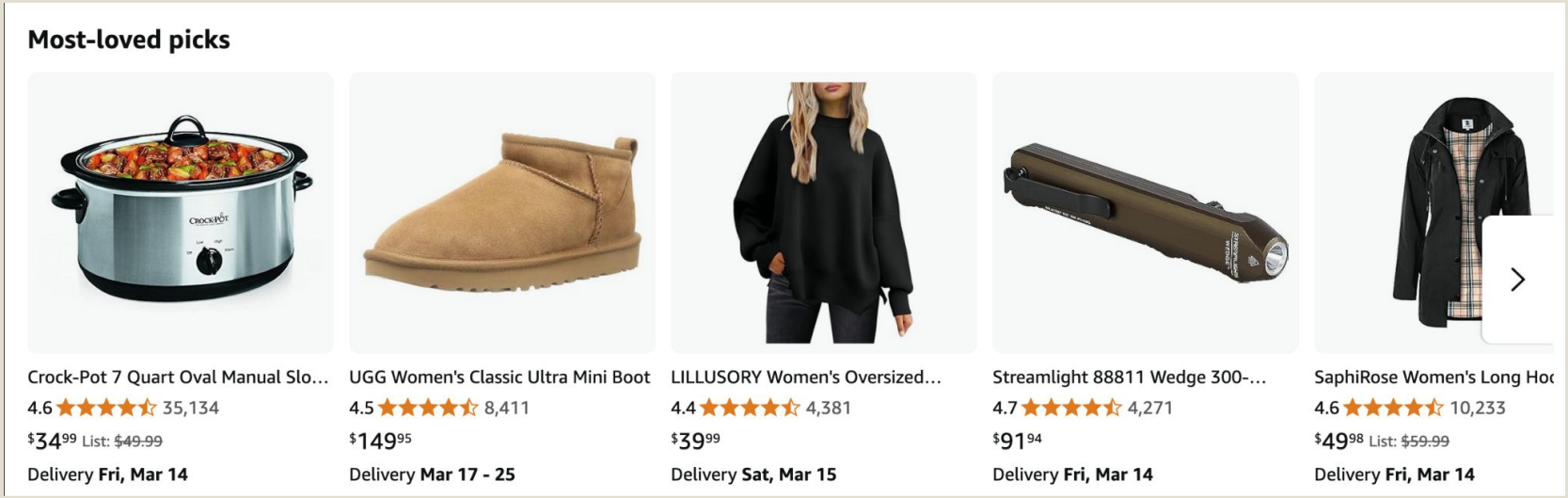
Partition Key = Primary ID % 2

Show list of most-loved products



Poor key choice for an E-Commerce Website





Data split between partitions

Peer Discussion

1. What other kinds of products / services can benefit from partitioning? (Name at least 2)

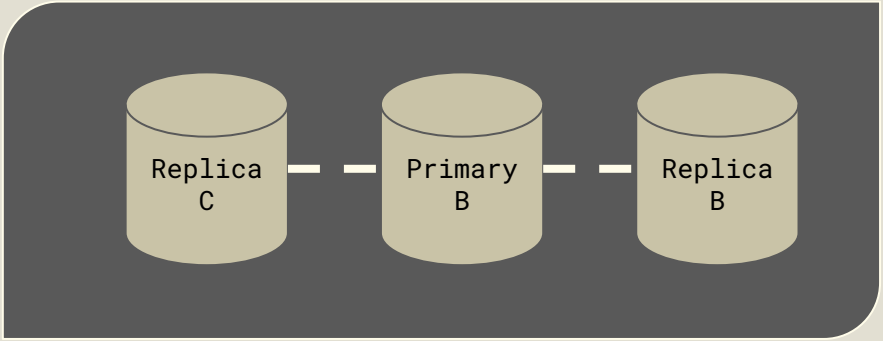
2. What sort of keys would make sense to be partition keys for these products / services?

~10 mins, discuss in pairs and pick a person to share.

General Guideline: Select a key that best allow 50 - 80% of Reads to be within a single partition

What is data partitioning, and how to do it right

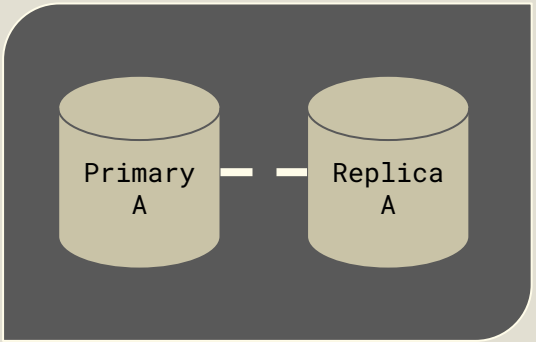
https://www.cockroachlabs.com/blog/what-is-data-partitioning-and-how-to-do-it-right/



Region B

Handles read and
write requests coming
from Region B++

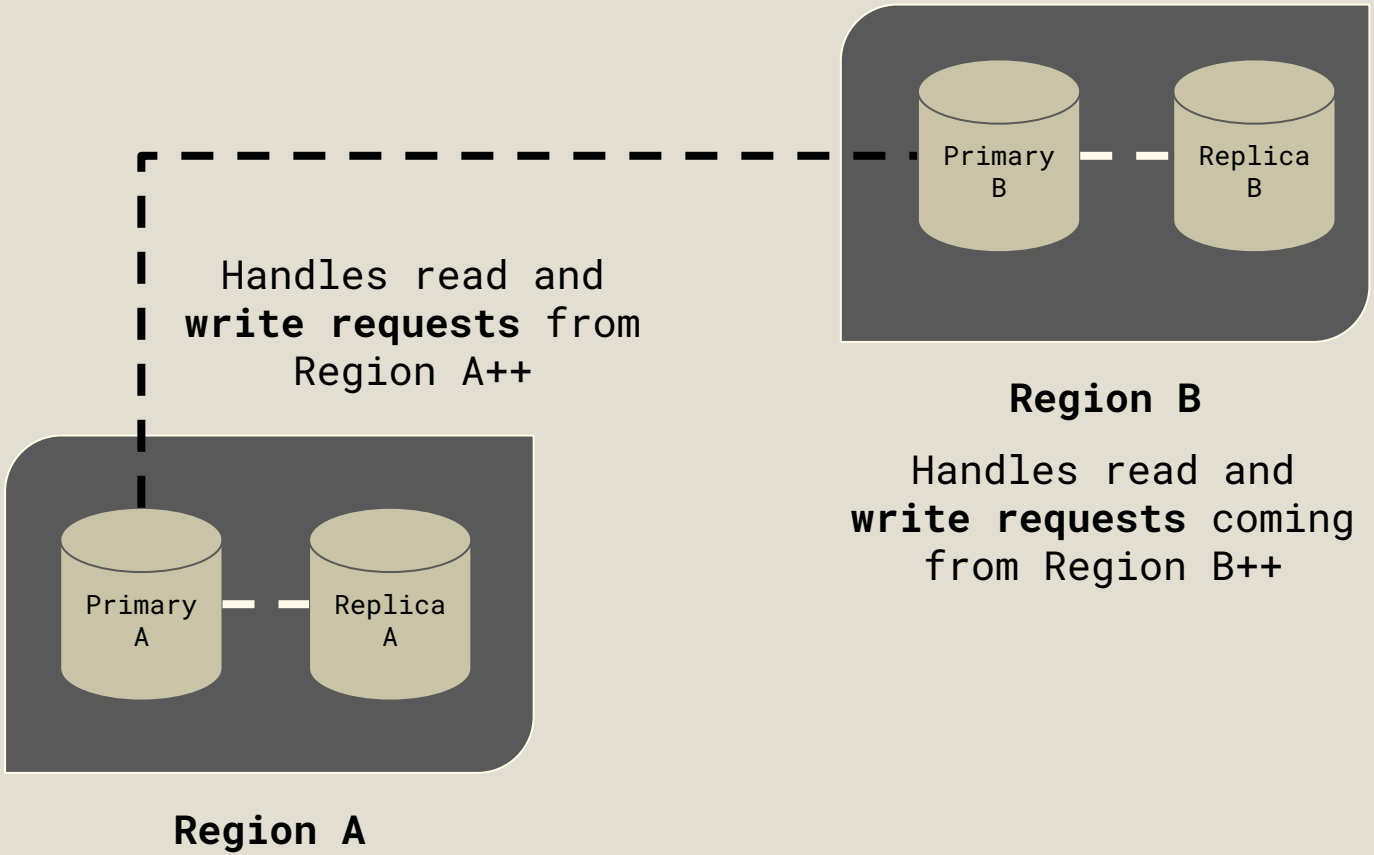
Handles read and
write requests from
Region A++



Region A

Regional Scaling

2.3.2 multi-primary



Multi-Primary

All nodes keep all the data

Pros

- All nodes have full copies of all data* (easier time with disaster recovery)

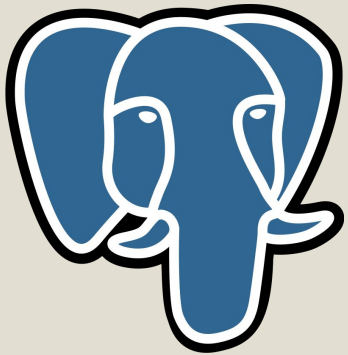
- Each DB has a complete view of the entire Database

Cons

- Storage space multiplied by N nodes

- Usually need at least 3 or more nodes

- Primaries need to achieve **consensus** for a write to be confirmed

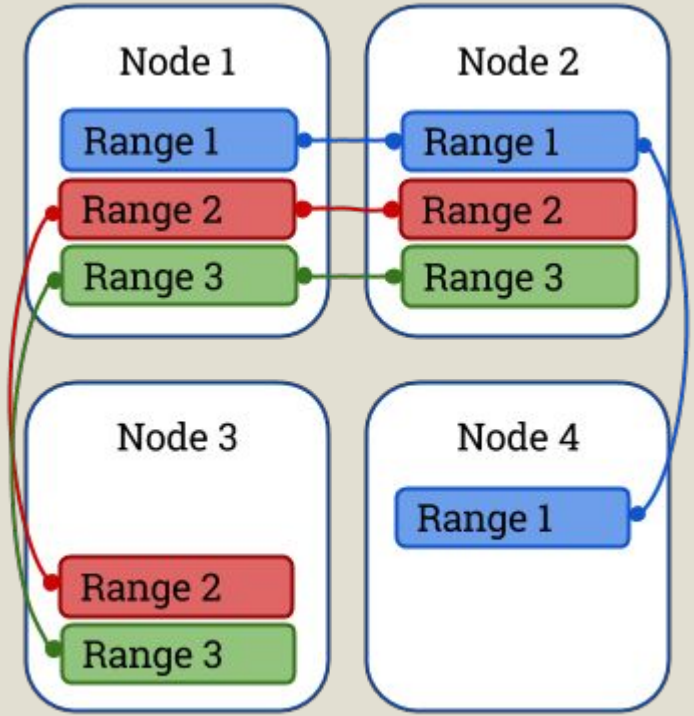


Does not natively Multi-Primary replication!

(3rd party tools exist to do so)

Supports Multi-Primary replication

MySQL Multi-Primary Mode: https://dev.mysql.com/doc/refman/8.4/en/group-replication-multi-primary-mode.html



Range Replication in CockroachDB

Number of Nodes and **Number of Replicas (Replication Factor)** available determines the number of node failures we can tolerate.

Number of replicas = 3 by default in CockroachDB

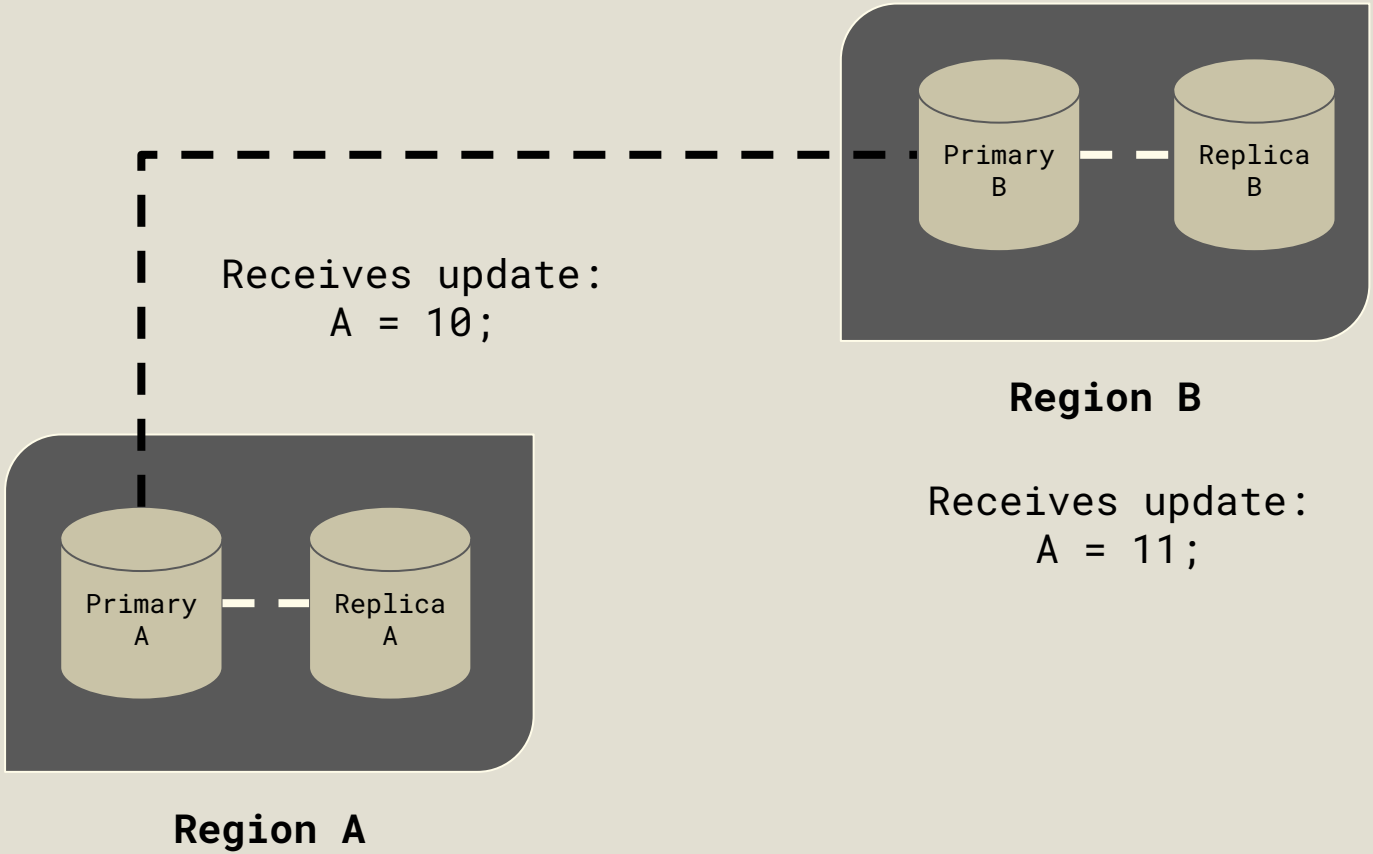
More on Multi-Primary in W3

https://www.cockroachlabs.com/blog/the-new-stack-meet-cockroachdb-the-resilient-sql-database/

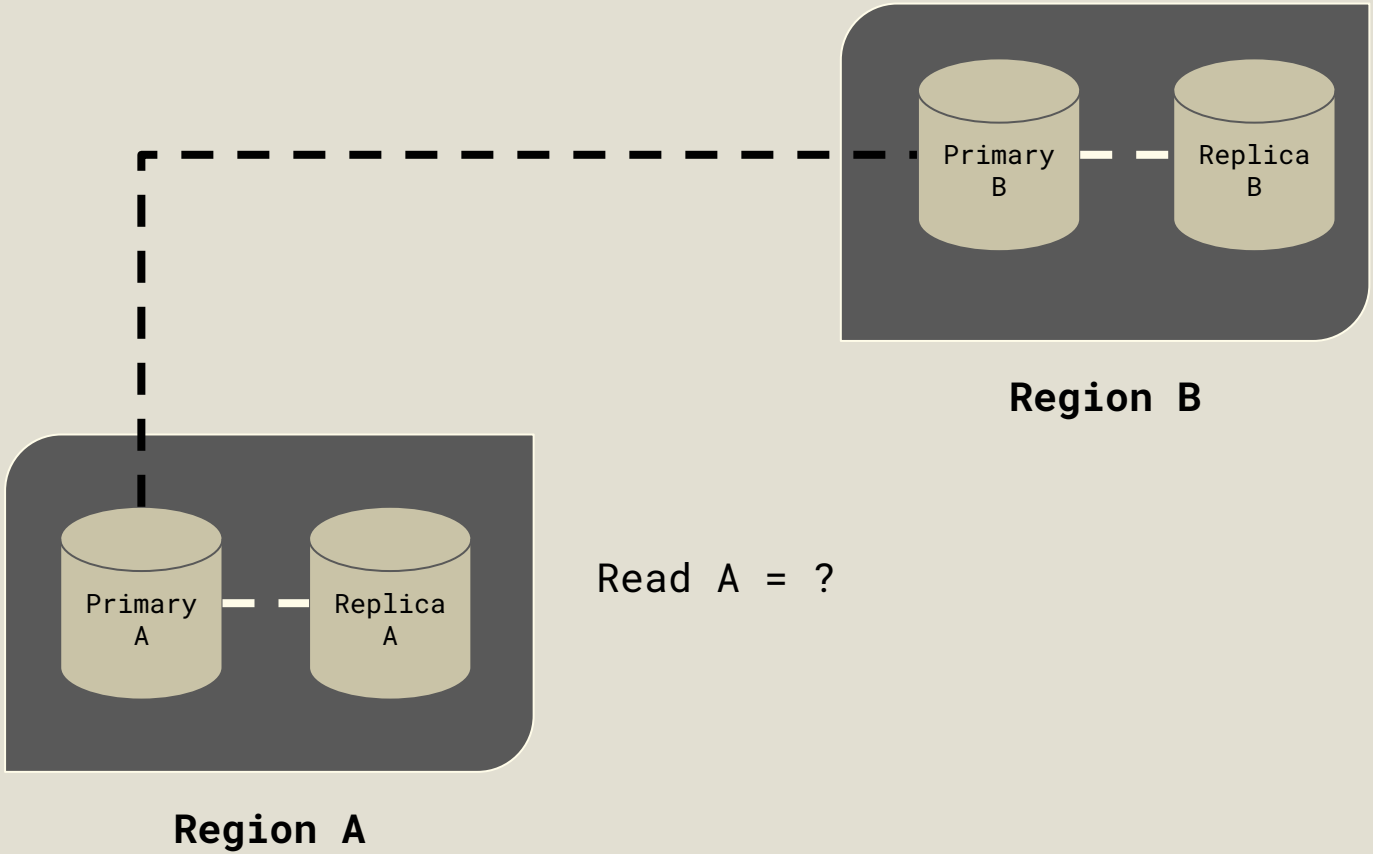
	Dataset on each node	Risks
Partitioning	Partial	Routing key needs to be well selected
Multi-Primary	Partial	Consensus* needs to be achieved, latency
 **Streaming Replication**	Full	Primary node might be overwhelmed
 **Chained Replication**	Full	Latency for data to reach nodes that are far in the chain

Horizontal Scaling Methods

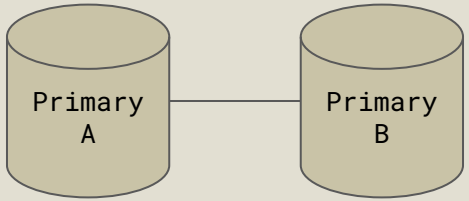
2.4: consensus and gossip



Consensus Problem

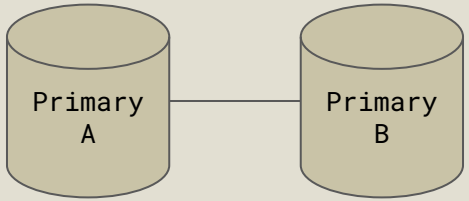

Primary
B
Replica
B
Region B
Primary
A
Replica
A
Region A
Read A = ?

Receives update: A = 10;



Receives update: A = 11;

Receives update: A = 10;

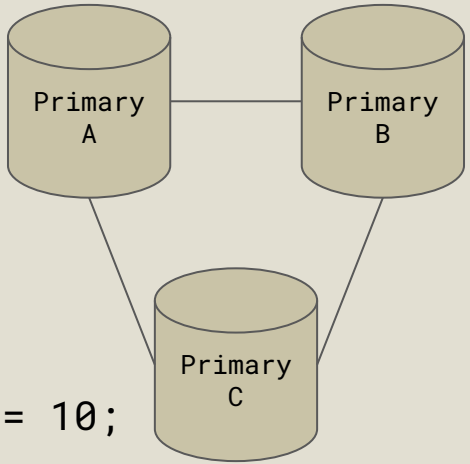


Receives update: A = 11;

Unlikely to achieve consensus without intervention

1.1 Receives update: A = 10;

1.2 Receives update: A = 11;

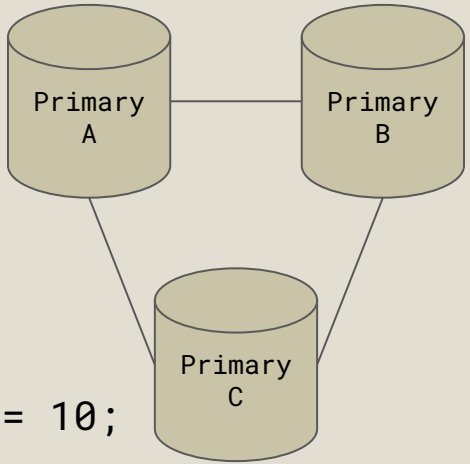


2.1 A = 10;

Consensus: A and C agrees

1.1 Receives update: A = 10;

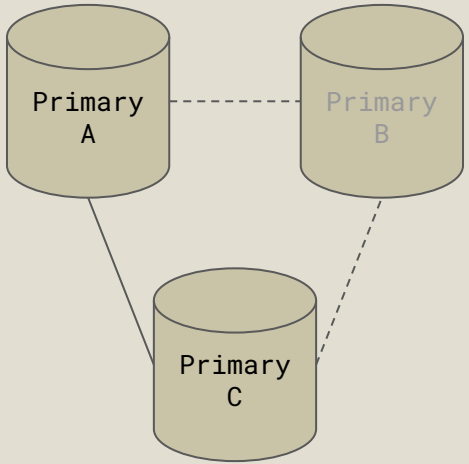
3.1 Receives update: A = 10;



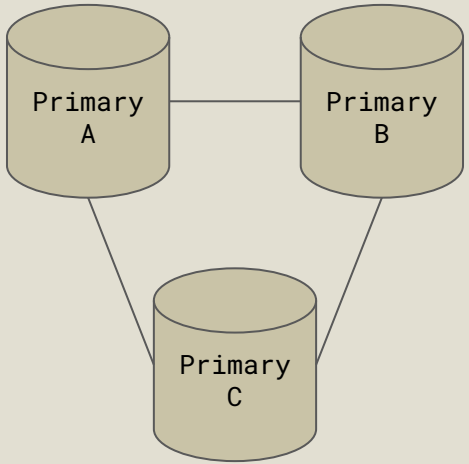
2.1 Receives update: A = 10;

Consensus achieved: ⅔ agrees

Even More Questions



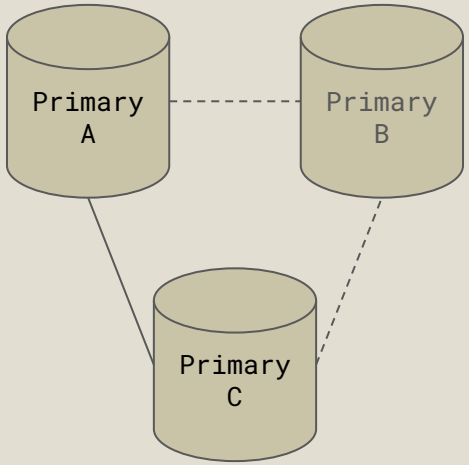
Q. What happens when a node disappears?



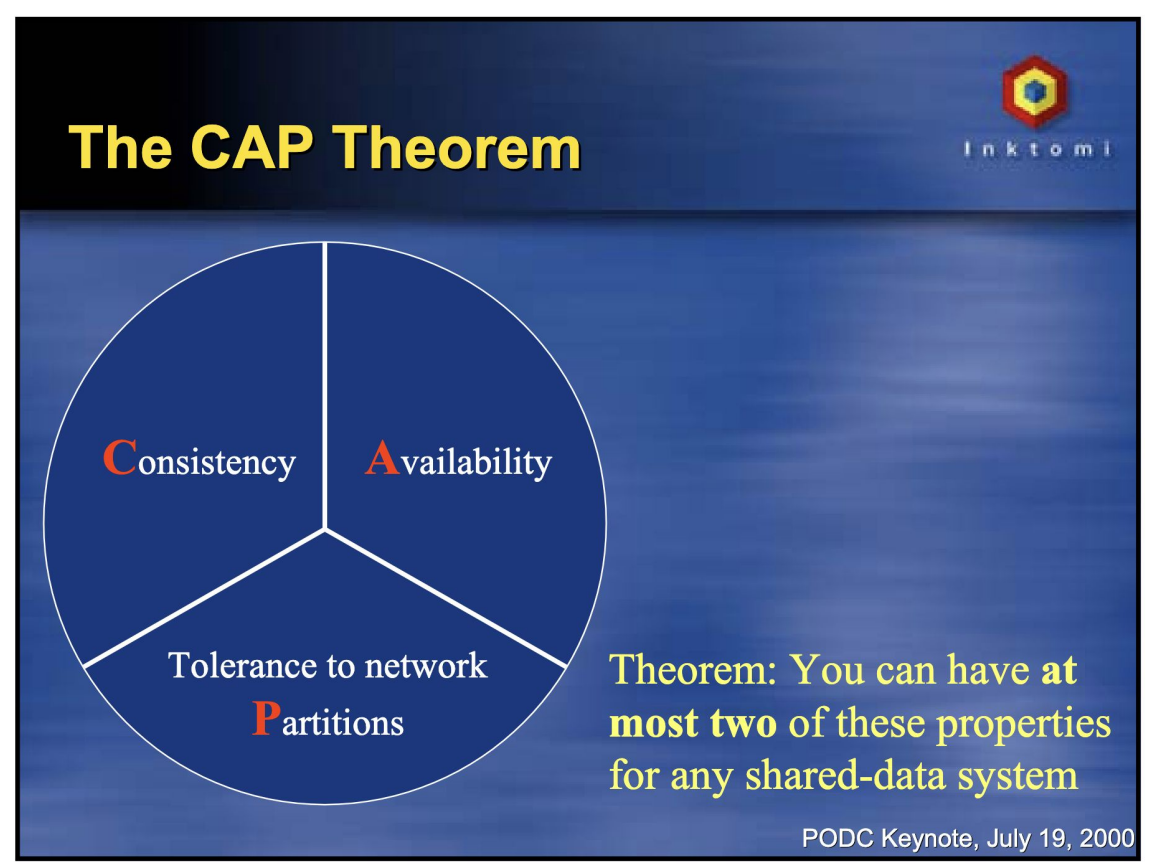
Q. How do nodes talk amongst each other?

Break Time

CAP Theorem



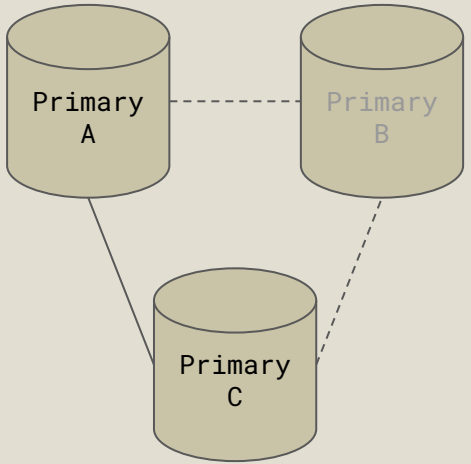
Q. What happens when a node disappears in a multi-primary setup?



CAP Theorem

https://sites.cs.ucsb.edu/~rich/class/cs293b-cloud/papers/Brewer_podc_keynote_2000.pdf

When network partitions happen, you need to decide if you want to prioritize **Availability** or **Consistency.**



CAP Theorem

ATM network encounters a partition, and the ATM may be unable to get the current actual account balance.

Will you:
a. Deny all transactions
b. Allow all transactions
c. Allow some types of transactions? (what types?)

ATM: Consistency or Availability?

Types of transactions:

- Deposit
- **Withdrawal**
- Check balance

Can withdrawals happen without knowing the perfect account balance?

ATM: Consistency or Availability?

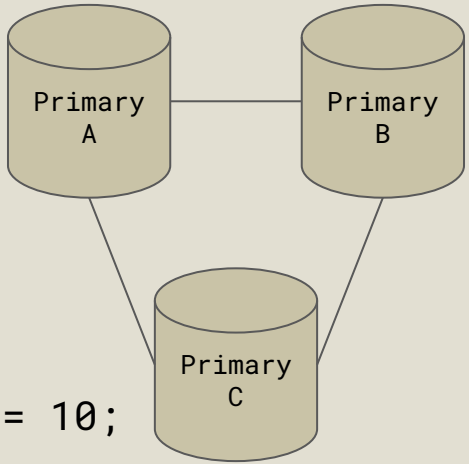
Limit withdrawals to some amount, eg. $100 or less → **prioritize Availability**

Deny all withdrawals and deposits → **prioritize Consistency**

ATM: Consistency or Availability?

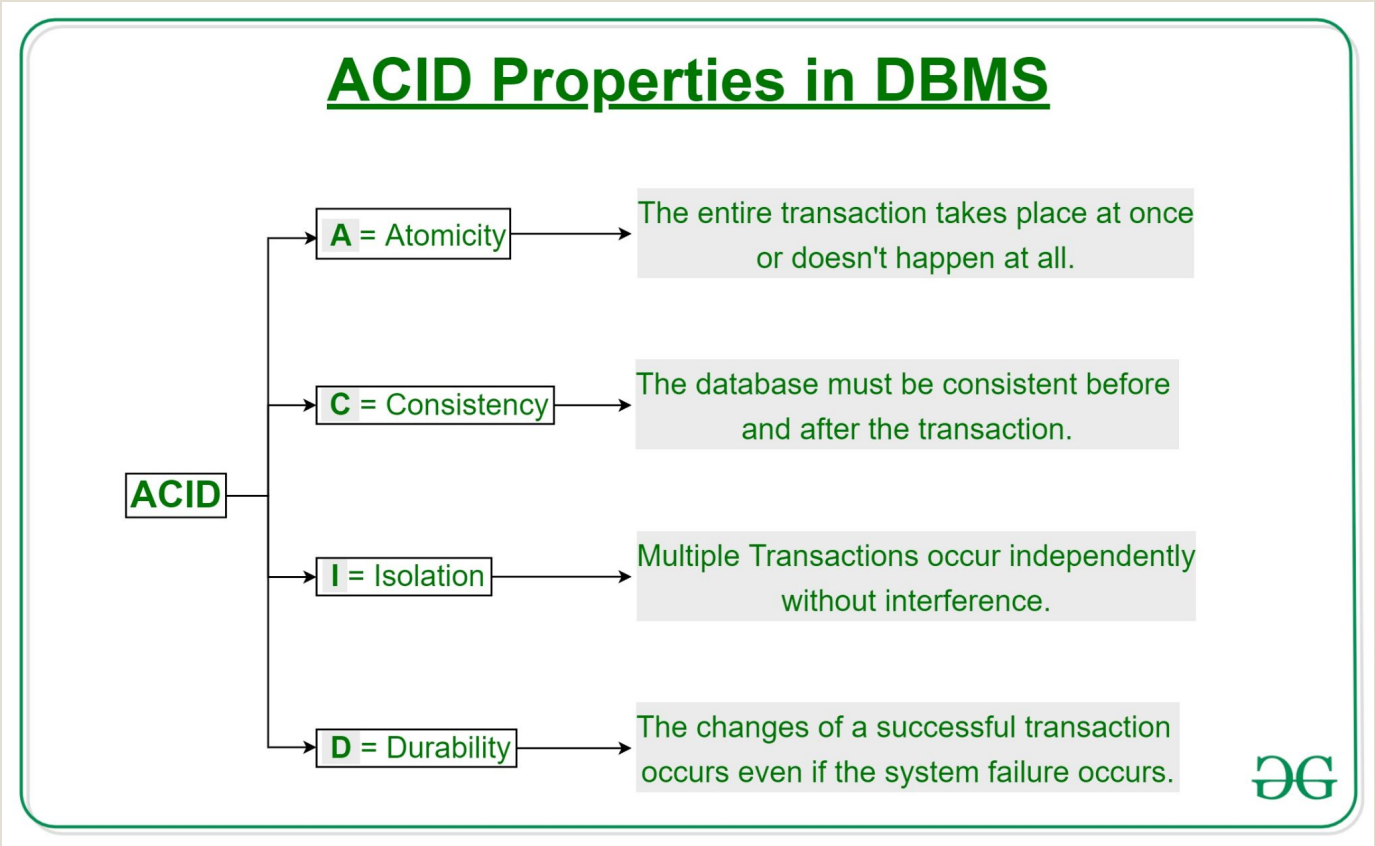
1.1 Receives update: A = 10;

3.1 Receives update: A = 10;



2.1 A = 10;

Note about Consistency in CAP

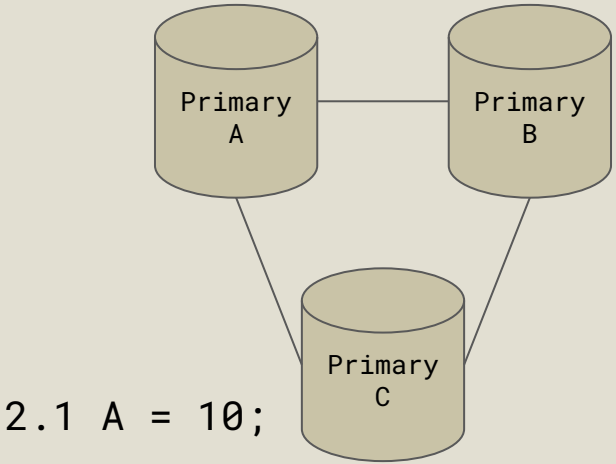


CAP Consistency != ACID Consistency

https://www.geeksforgeeks.org/acid-properties-in-dbms/

1.1 Receives update: A = 10;

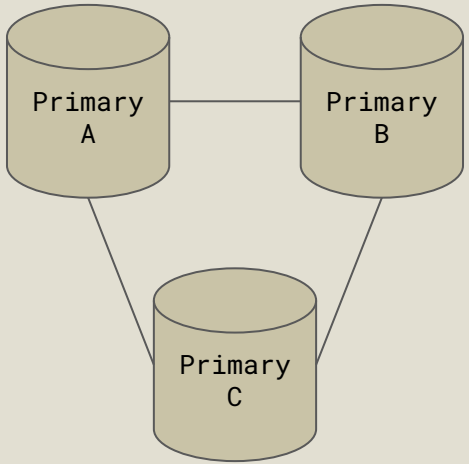
3.1 Receives update: A = 10;



2.1 A = 10;

Eventual Consistency:
All nodes will be consistent, eventually

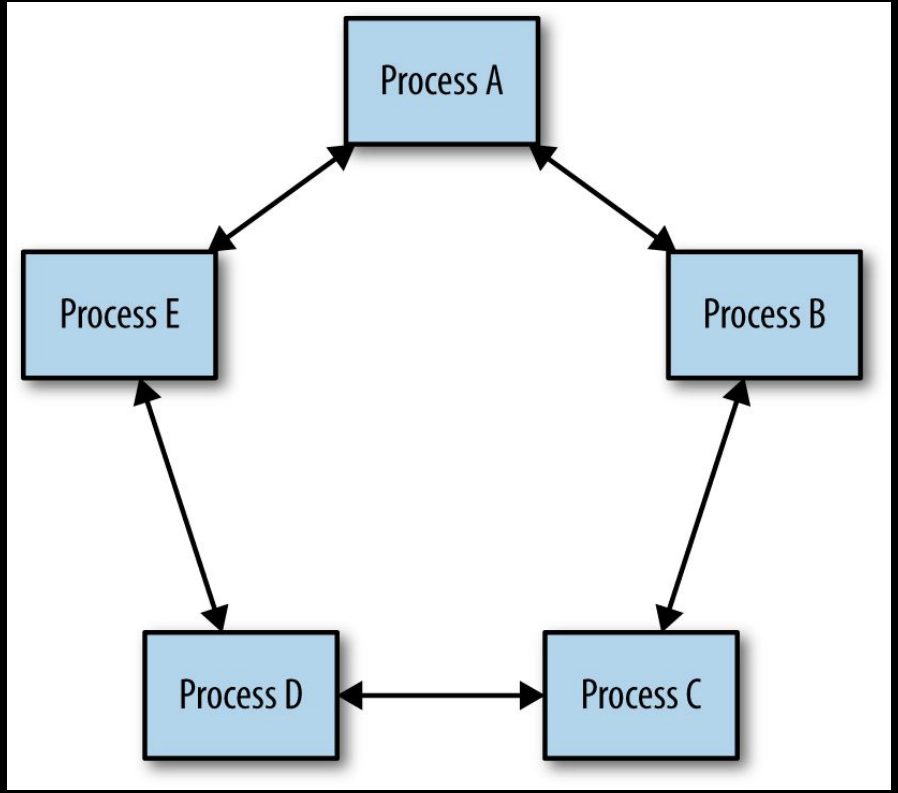
Consensus and Gossip Protocols



Q. How do nodes talk amongst each other?

“We’ve found distributed consensus to be effective in building reliable and highly available systems that require a consistent view of some system state.

The distributed consensus problem deals with reaching agreement among a group of processes connected by an unreliable communications network.”



Managing Critical State: Distributed Consensus for Reliability
https://sre.google/sre-book/managing-critical-state/

“We’ve found distributed consensus to be effective in building reliable and highly available systems that require a consistent view of some system state.

The distributed consensus problem deals with reaching agreement among a group of processes connected by an unreliable communications network.”

Managing Critical State: Distributed Consensus for Reliability
https://sre.google/sre-book/managing-critical-state/

System State = Data in Database

Group of Processes = Database Nodes

Raft

Paxos
(no mascot)

Popular Consensus Algorithms

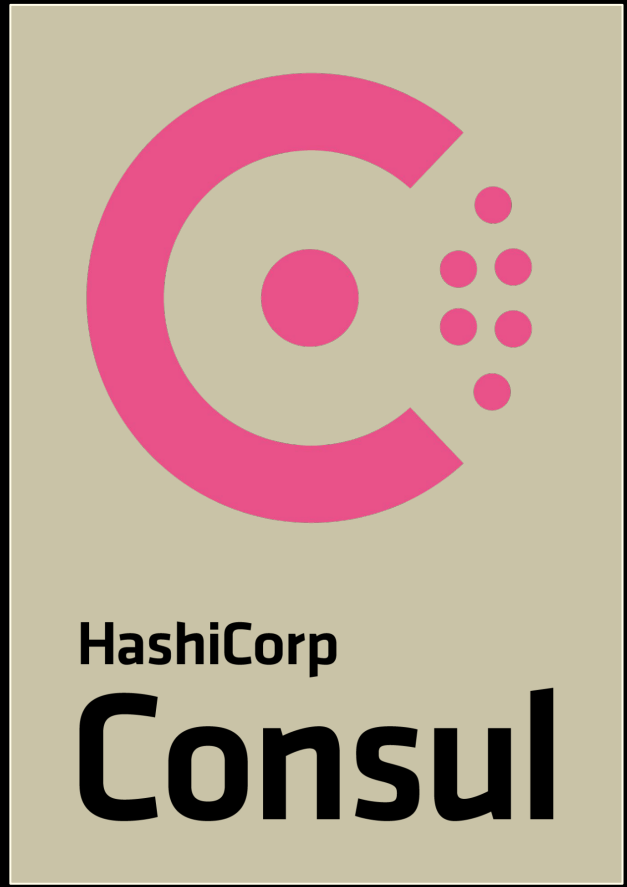
“A Raft cluster of 3 nodes can tolerate a single node failure while a cluster of 5 can tolerate 2 node failures.

The recommended configuration is to either run 3 or 5 Consul servers per datacenter.

This maximizes availability without greatly sacrificing performance. The deployment table below summarizes the potential cluster size options and the fault tolerance of each.”

Consensus Protocol

https://developer.hashicorp.com/consul/docs/architecture/consensus

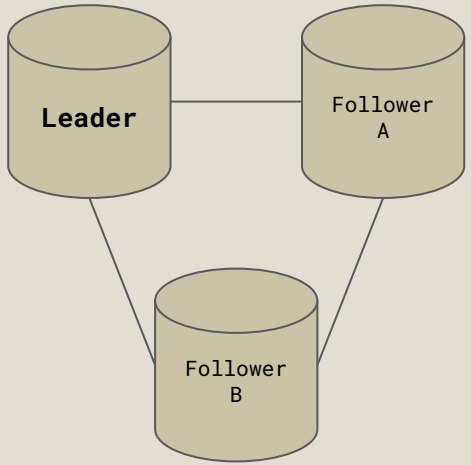


- There is always a **Leader**

- **Leader decides what data will be committed**

- When Leader goes missing, Followers will put themselves up for voting to choose next Leader **(Leader Election)**

- A quorum is a majority of members: for a cluster of size N, quorum requires at least (N/2)+1 members, eg. cluster of 3 nodes can tolerate 1 failure only



Raft Algorithm: Quick Summary

https://raft.github.io/raft.pdf

Append-only log in Raft

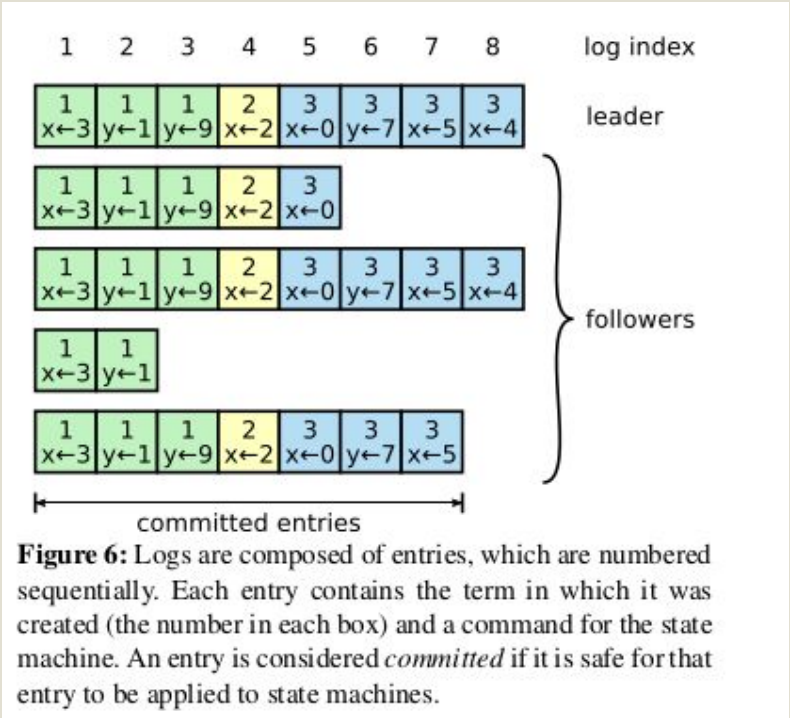
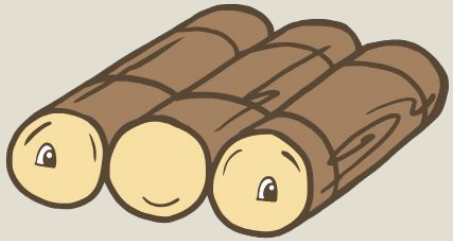


Figure 6: Logs are composed of entries, which are numbered sequentially. Each entry contains the term in which it was created (the number in each box) and a command for the state machine. An entry is considered *committed* if it is safe for that entry to be applied to state machines.



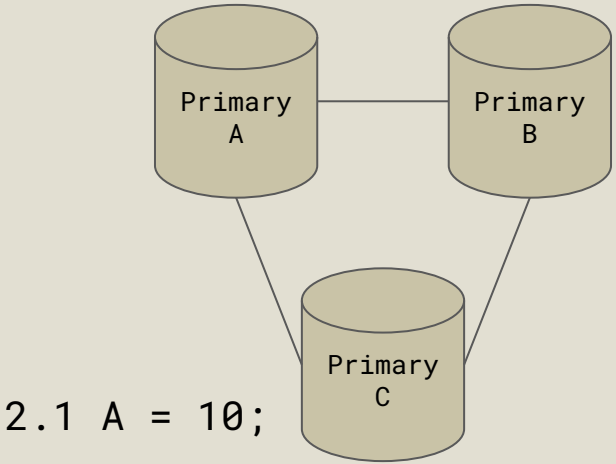
Raft

Raft Algorithm: Quick Summary

https://raft.github.io/raft.pdf

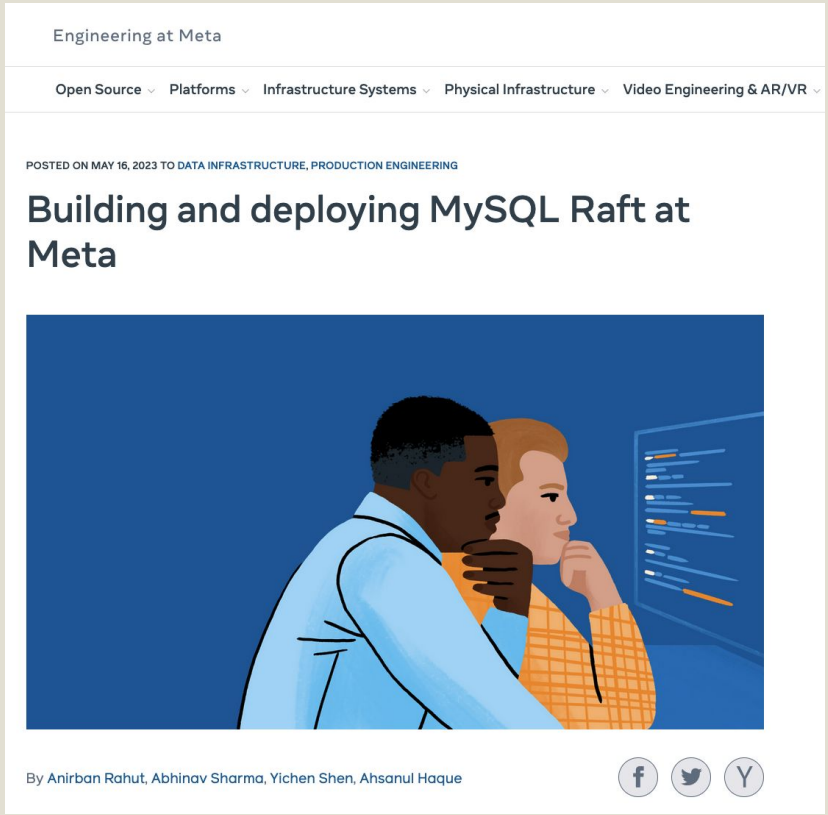
1.1 Receives update: A = 10;

3.1 Receives update: A = 10;



2.1 A = 10;

Raft used for ensuring eventual consistency



Before: MySQL semisynchronous databases (which uses Paxos internally)

After: MySQL Raft

Building and deploying MySQL Raft at Meta

https://engineering.fb.com/2023/05/16/data-infrastructure/mysql-raft-meta/

Summary

Summary: DB

- Understand how the underlying mechanics work for different databases, eg. WAL logs for PostgreSQL

- Prefer simple horizontal scaling options (eg. replication) over complicated ones such as Multi-Primary

- Partitioning is another option if data can be split cleanly between regions

Summary: CAP

- CAP Theorem: Managing risks of network failures (partitions) by prioritizing either Consistency or Availability

- Consensus: How distributed nodes talk to each other and agree, eg. Raft algorithm

- We will explore a little more on CAP when we talk about Distributed Databases next week

p.s.
Current Trends

searchcode.com itself is now officially a ship of Theseus project, as every part I started with has now been replaced. A brief history,

- First version released using PHP, CodeIgniter, MySQL, Memcached, Apache2 and Sphinx search
- Rewritten using Python, Django, MySQL, Memcached, Sphinx search, Nginx and RabbitMQ
- Never publicly released version using Java, MySQL, Memcached, Nginx and Sphinx search
- Start of Covid 19 rewritten using Go, MySQL, Redis, Caddy and Manticore search
- Replaced Manticore search with custom index now the stack consists Go, MySQL, Redis and Caddy
- As of a few days ago Go, SQLite, Caddy

The constant between everything till now was the use of MySQL as the storage layer. The reasons for using it initially was it was there, I knew how to work it and it would scale along with my needs fairly well. So what changed? If you look at my previous choices you will see there is in general a move to reducing the number of dependencies. The older and more crusty I get the more I appreciate having a single binary I can just deploy. Single binary deploys are very simple to reason about.

searchcode.com's SQLite database is probably 6 terabytes bigger than yours

https://boyter.org/posts/searchcode-bigger-sqlite-than-you/

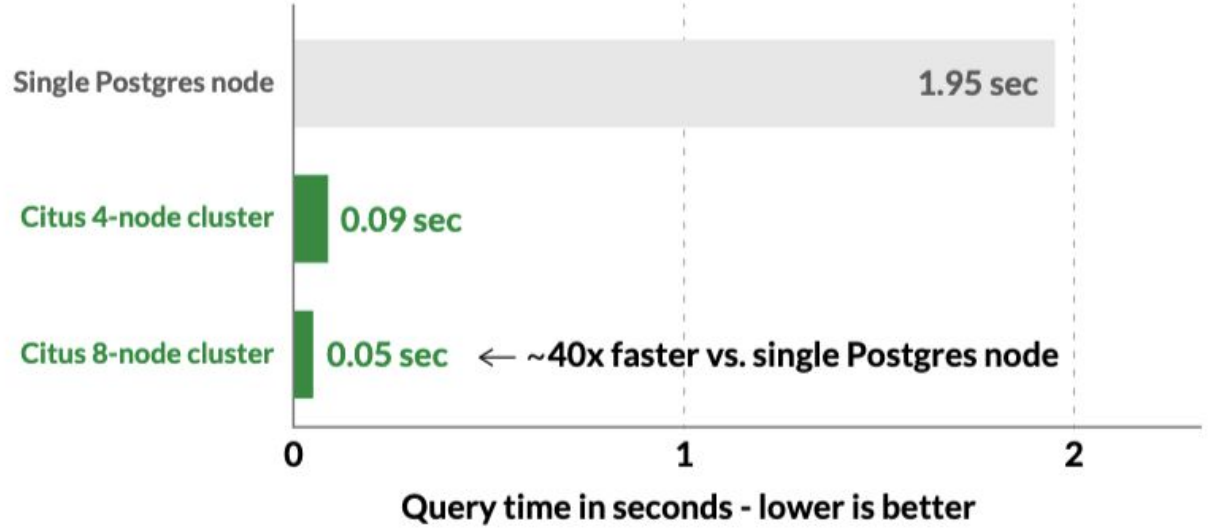
CockroachDB

Database Category	Database Name	Wire-Protocol Compatibility	Syntax Compatibility	Feature Compatibility	Runtime Compatibility
Sharded Postgres	CitusData	Full	Full	High	High
	Azure CosmosDB for PostgreSQL	Full	Full	High	High

Database Category	Database Name	Wire-Protocol Compatibility	Syntax Compatibility	Feature Compatibility	Runtime Compatibility
Distributed SQL Databases	Google Spanner	Full	High	Low	Low
	CockroachDB	Full	High	Moderate	Low
	YugabyteDB	Full	Full	High	High

Distributed Databases offering PostgreSQL Compatibility

https://www.yugabyte.com/postgresql/compare-postgresql-compatibility/

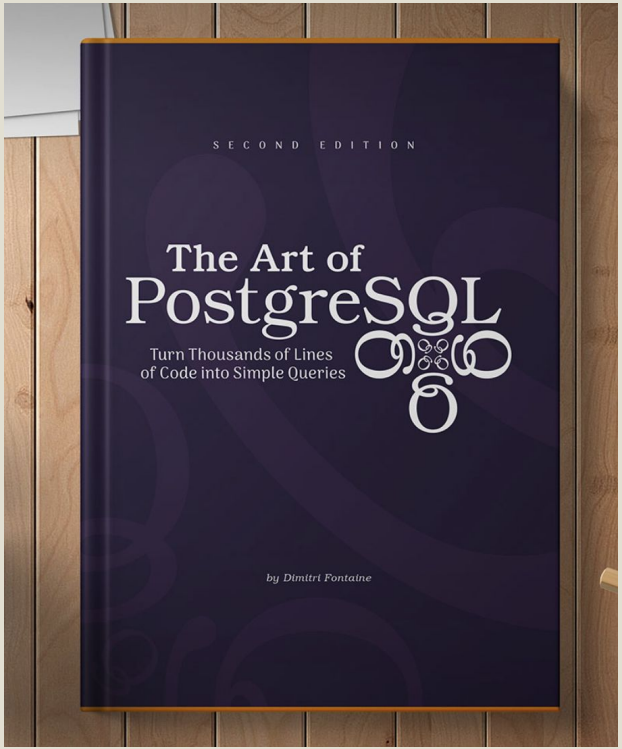


Time to complete a SQL query, using a benchmark that measures analytical query performance. Run on ~100 GB of GitHub archive data in JSON format. All servers are Azure VMs with 16 vCPUs, 64 GB of memory, and network-attached disks with 7500 IOPS —running Postgres 13 and Citus 9.5, with default settings.

Citus's Claims for Performance for Distributed PostgreSQL

https://www.citusdata.com/getting-started/

optional readings



Art of PostgreSQL
https://theartofpostgresql.com/

The Internals of PostgreSQL
https://www.interdb.jp/pg/index.html

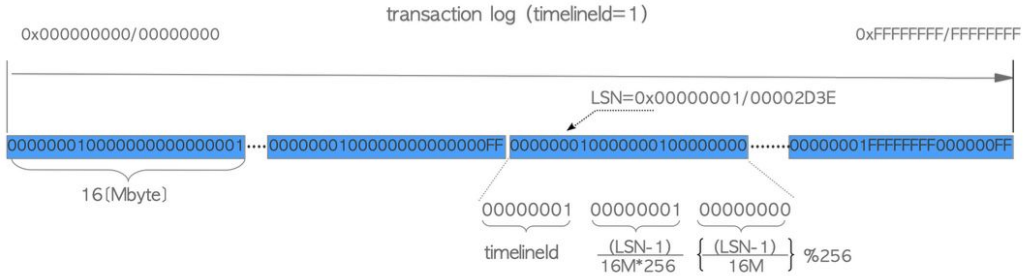


Figure 9.6. Transaction log and WAL segment files

The WAL segment filename is in hexadecimal 24-digit number and the naming rule is as follows:

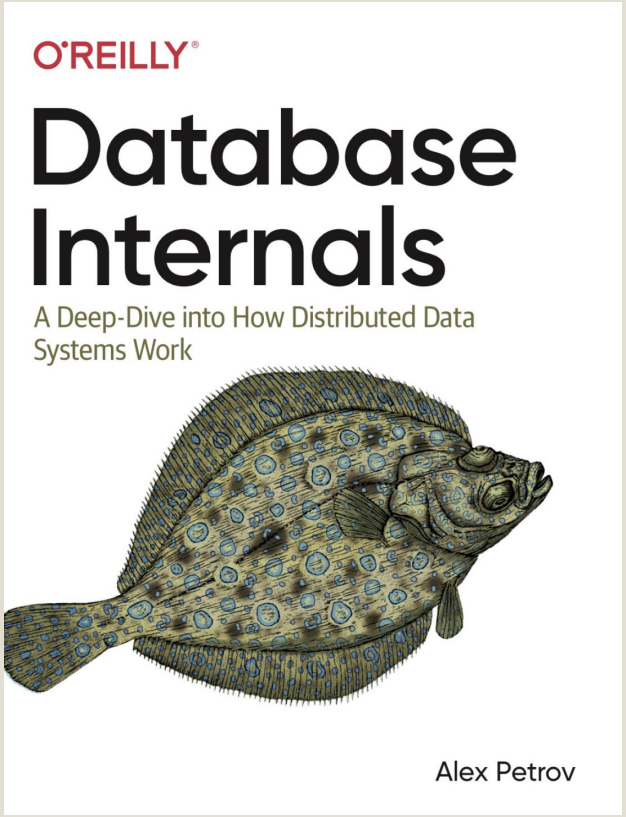
$$\text{WAL segment file name} = \text{timelineId} + (\text{uint32})\frac{\text{LSN} - 1}{16\text{M} * 256} + (\text{uint32})\left(\frac{\text{LSN} - 1}{16\text{M}}\right)$$

timelineId

PostgreSQL's WAL contains the concept of **timelineId** (4-byte unsigned integer), which is for Point-in-Time Recovery (PITR) described in Chapter 10. However, the timelineId is fixed to 0x00000001 in this chapter because this concept is not required in the following descriptions.

The Internals of PostgreSQL

https://www.interdb.jp/pg/index.html

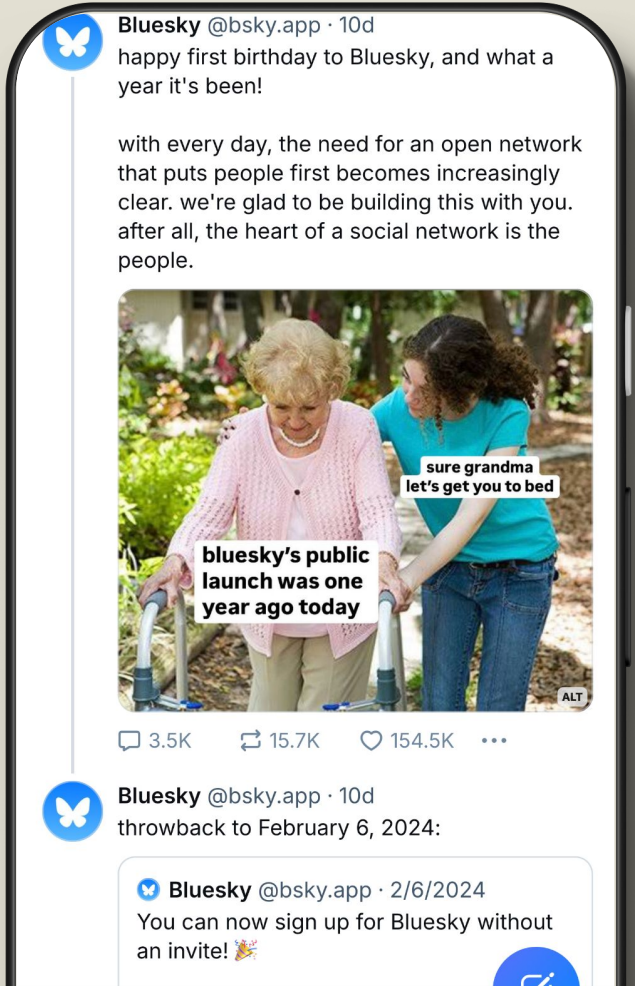


Database Internals
https://www.databass.dev/

assignment

Assignment

Extend a Distributed Social Media Platform



Assignment #1

- You are a Senior Engineer in a team responsible for building a new product that will extend upon one of these platforms / protocols.

- Product: Let user post and upload their location check-ins, similar to Foursquare / Swarm) with GPS coordinates and Photos, with a global newsfeed.

- **Choose one of the two protocols, analyze it, and share in a 3-page technical document about why it would be a good or bad choice for this project.**

- Audience: Tech Leads, Software Engineers, Product Managers, Senior Business Leaders

- **Due date: Sunday, 23:59 (UTC+8)**

W1 Assignment

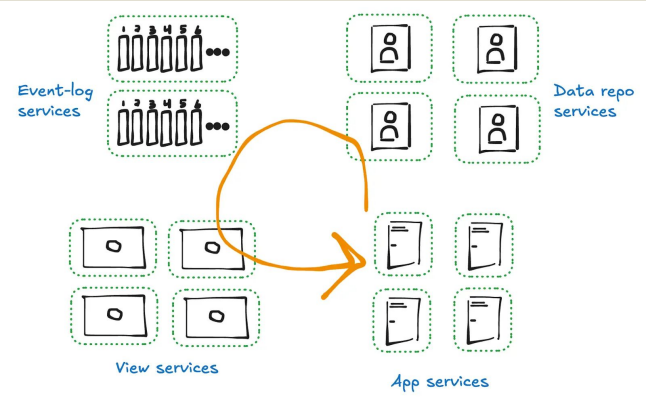
- 2 chose to review AT Protocol
  2 chose to review ActivityPub
  2 chose to review and compare both

Assignment

W1 → Distributed Social Media Research
W2 → Review W1 + Build PoC
W3 → Design 1st Draft of System
W4 → Implementation + Sharing
W5 → Implementation (Queues)
W6 → Implementation (Load Testing)
W7 → Technical Retrospective
W8 → Complete System Implementation

Bluesky: AT Protocol

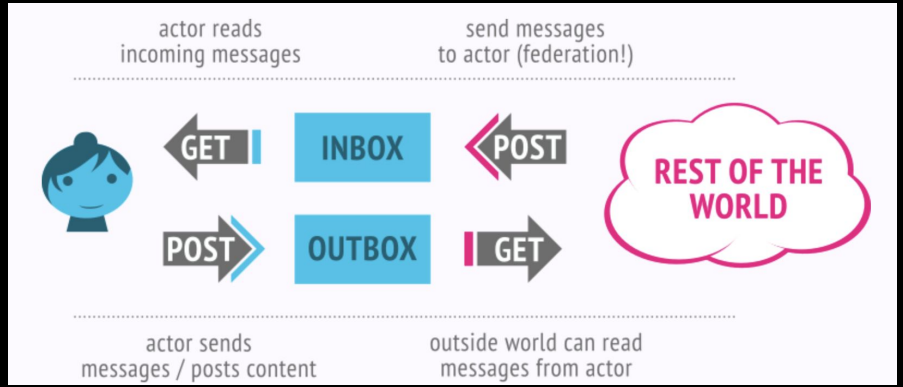
"The AT Protocol is an open, decentralized network for building social applications."



AT Protocol: https://atproto.com/

Mastodon: ActivityPub

"ActivityPub is a decentralized social networking protocol based on the ActivityStreams 2.0 data format."



ActivityPub: https://activitypub.rocks/

Assignment #2

1. Review 1 other technical design document written by others
   - Choose one that has decided on a different protocol from yours
   - Focus on understanding their whys, and insights from them
   - Where and how did their analysis differ from yours?
   - Are there potential Pros and Cons that they have missed out?

2. Build a quick and hacky PoC to create a Post on the platform you have chosen (share link to code repository)

Due date: Sunday, 23:59 (Anywhere on Earth, UTC-12)

- Which protocol will you use? Decide after reviewing the documents.

- For the PoC, just focus on creating a Post first; avoid doing any form of systems design until you have created a Post