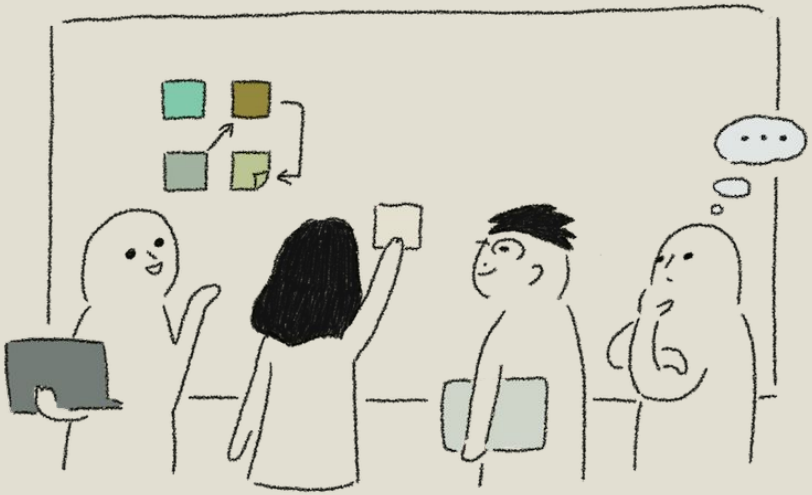
DATE 20250317

Session 3



backend

cohort #0 by open camp

#3's agenda

1 ..... admin matters (if any)
2 ..... reCAP
3 ..... distributed databases
4 ..... w3 assignment
5 .....

admin matters

W2 Assignment

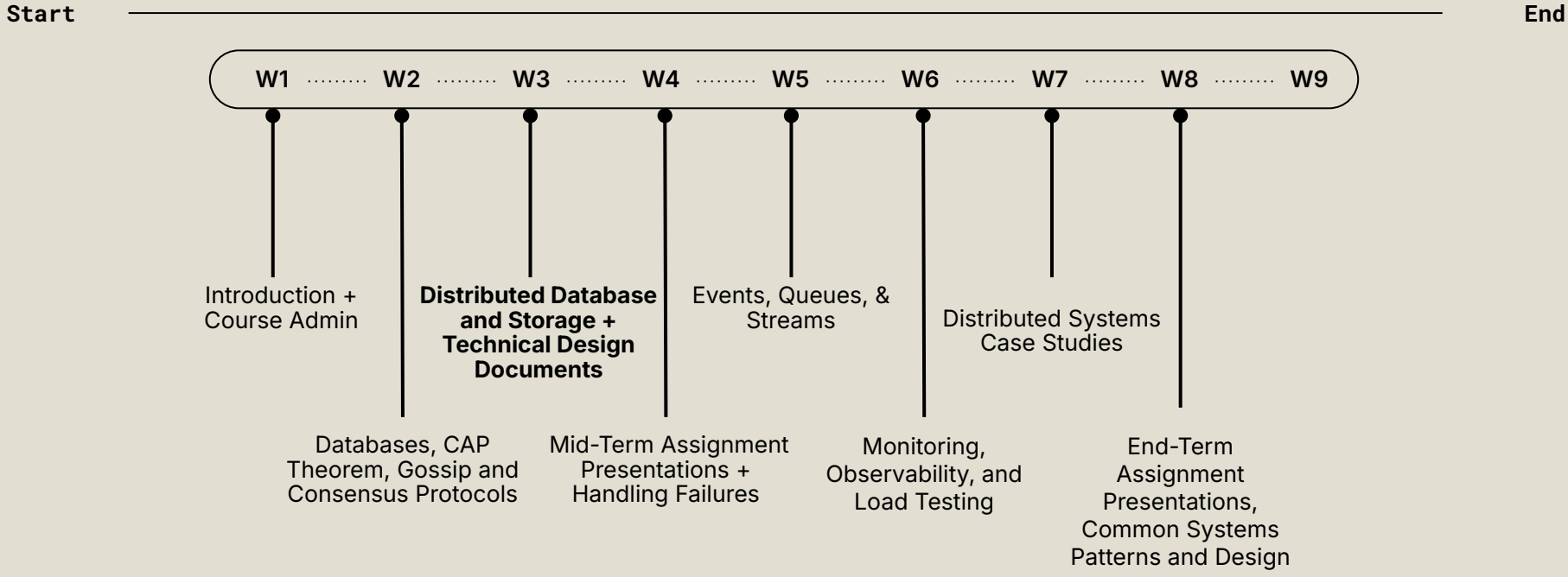
- All reviews have been submitted, but if you have not, please let me know which document you have reviewed

- I will consolidate others' feedback + mine and send them over this week

POCs:

- 3 AT Protocol
- 3 ActivityPub

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



3.1 reCAP

Databases

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

Assumes 1 machine by default

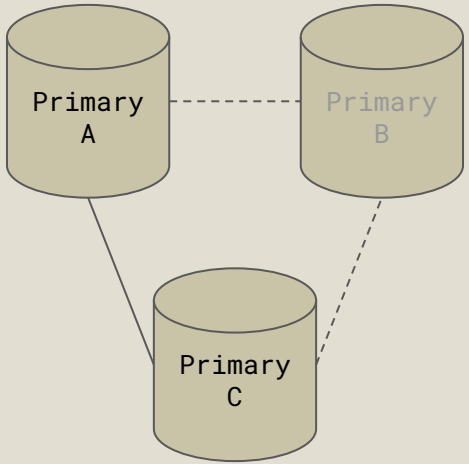
Distributed Databases

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

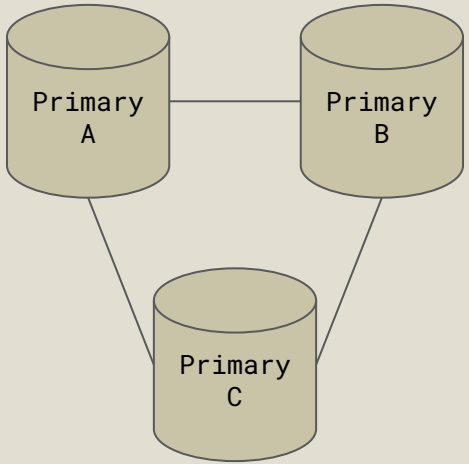
Assumes 3* or more machines by default

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

	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

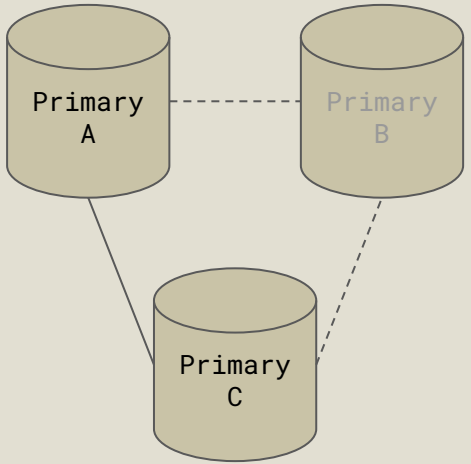


Q. What happens when a node disappears?



Q. How do nodes talk amongst each other?

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



CAP Theorem

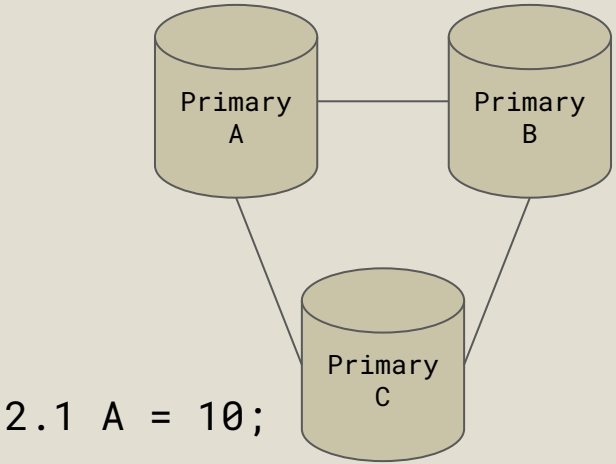
Limit withdrawals to some amount, eg. $100 or less → **prioritize Availability**

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;

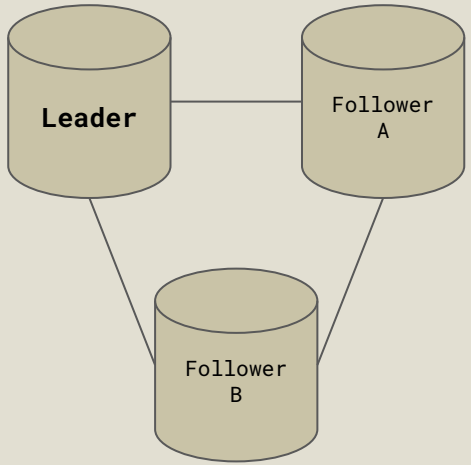
Eventual Consistency:
All nodes will be consistent, eventually

- 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

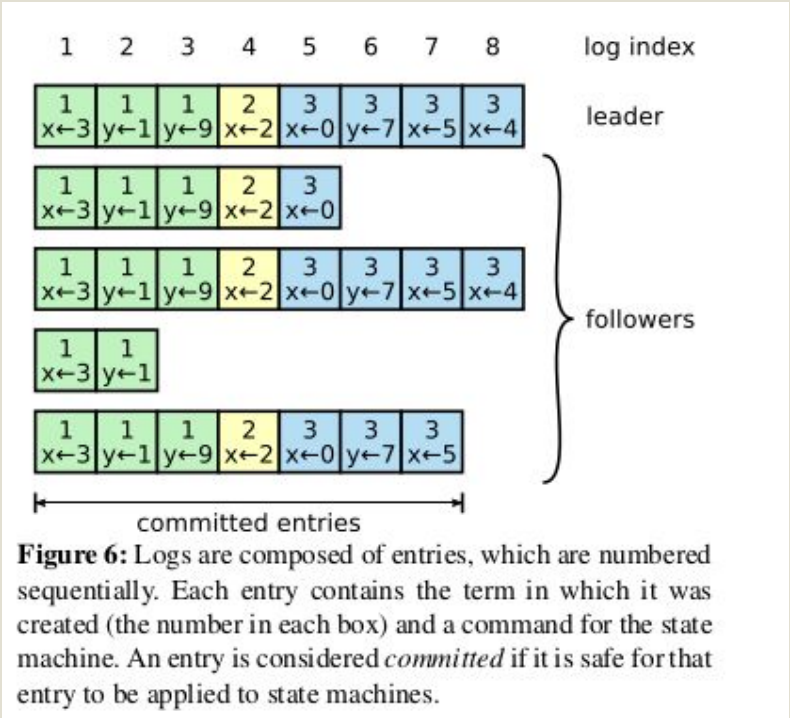
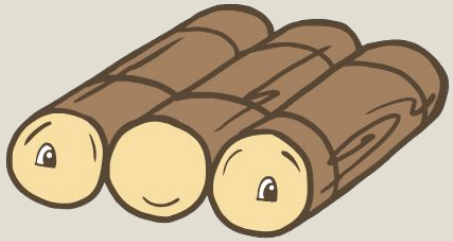


Figure 6: Logs are composed of entries, which are numbered sequentially. Each entry contains the term in which it was created (the number in each box) and a command for the state machine. An entry is considered *committed* if it is safe for that entry to be applied to state machines.



Raft

Raft Algorithm: Quick Summary

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

3.2 distributed databases

Distributed Database
PostgreSQL Compatible

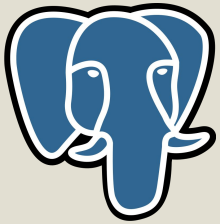
Distributed Data Store
Securely store and access secrets for services

3.2.1 cockroachdb

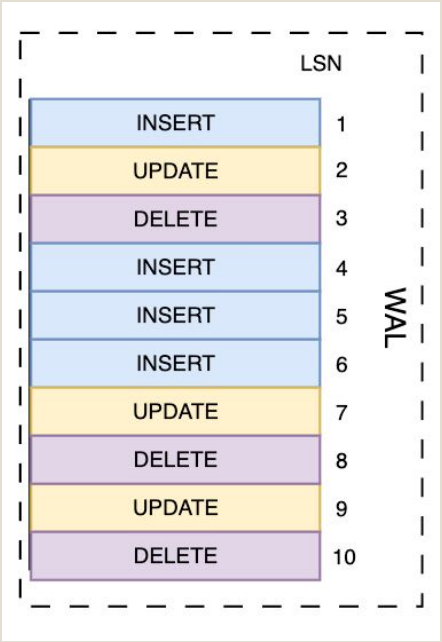
Distributed Database: Horizontal Scalability

PostgreSQL Compatible: can use PostgreSQL's client as they are compatible

p.s. The name CockroachDB means resilience and scale.



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

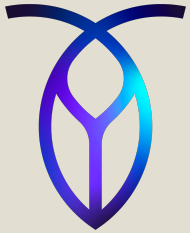


New data does not get written to the table right away!

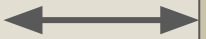
It’s faster to append it to a WAL log, and say that the SQL transaction is complete.

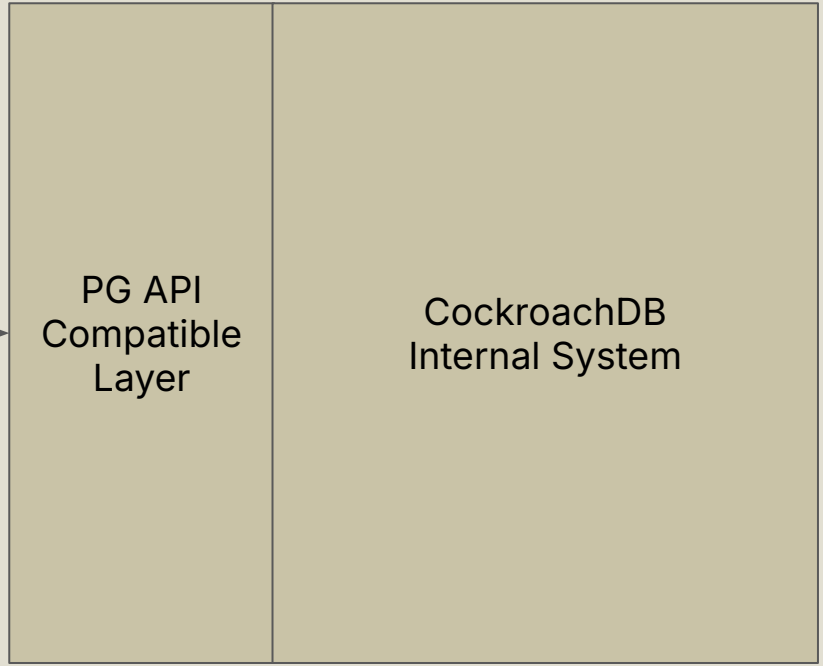
WAL log is append-only.

PostgreSQL: Uses WAL Logs Internally



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





CockroachDB was designed to meet the following goals:

- Make life easier for humans. This means being low-touch and highly automated for operators and simple to reason about for developers.
- Offer industry-leading consistency, even on massively scaled deployments. This means enabling distributed transactions, as well as removing the pain of eventual consistency issues and stale reads.
- **Create an always-on database that accepts reads and writes on all nodes without generating conflicts.**
- Allow flexible deployment in any environment, without tying you to any platform or vendor.
- **Support familiar tools for working with relational data (i.e., SQL).**

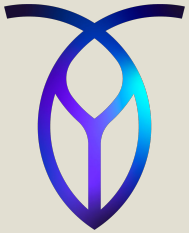
CockroachDB Architecture Overview

https://www.cockroachlabs.com/docs/stable/architecture/overview

Layer	Order	Purpose
SQL	**1**	**Translate client SQL queries to KV operations.**
Transactional	2	Allow atomic changes to multiple KV entries.
Distribution	**3**	**Present replicated KV ranges as a single entity.**
Replication	4	Consistently and synchronously replicate KV ranges across many nodes. This layer also enables consistent reads using a consensus algorithm.
Storage	5	Read and write KV data on disk.

CockroachDB High-Level Architecture

https://www.cockroachlabs.com/docs/stable/architecture/overview



id	name	weight
34	carl	10.1
7A	dagne	13.4
94	figment	65.8
BC	jack	49.7

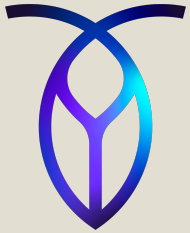
key	value
dog/34/name	carl
dog/34/weight	10.1
dog/7A/name	dagne
dog/7A/weight	13.4
dog/94/name	figment
dog/94/weight	65.8
dog/BC/name	jack
dog/BC/weight	49.7

Key: <table>/<index>/<key>/<columnName>
Value: <columnValue>

Cockroach Labs

The architecture of a distributed SQL database, part 1

https://www.cockroachlabs.com/blog/distributed-sql-key-value-store/



id	name	weight
34	**carl**	**10.1**
7A	dagne	13.4

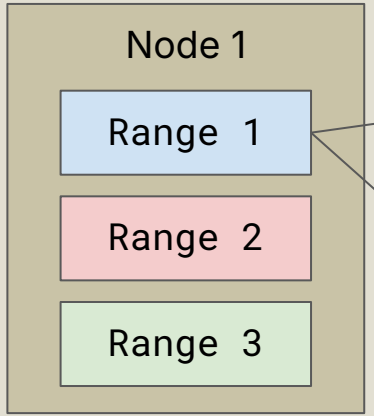
dog table (relational DB)

key	value
dog/34/name	**carl**
dog/34/weight	**10.1**
dog/7A/name	dagne
dog/7A/weight	13.4

Equivalent in Key-Value Store

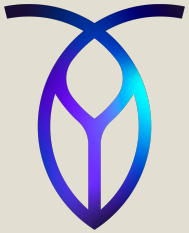
The architecture of a distributed SQL database, part 1

https://www.cockroachlabs.com/blog/distributed-sql-key-value-store/

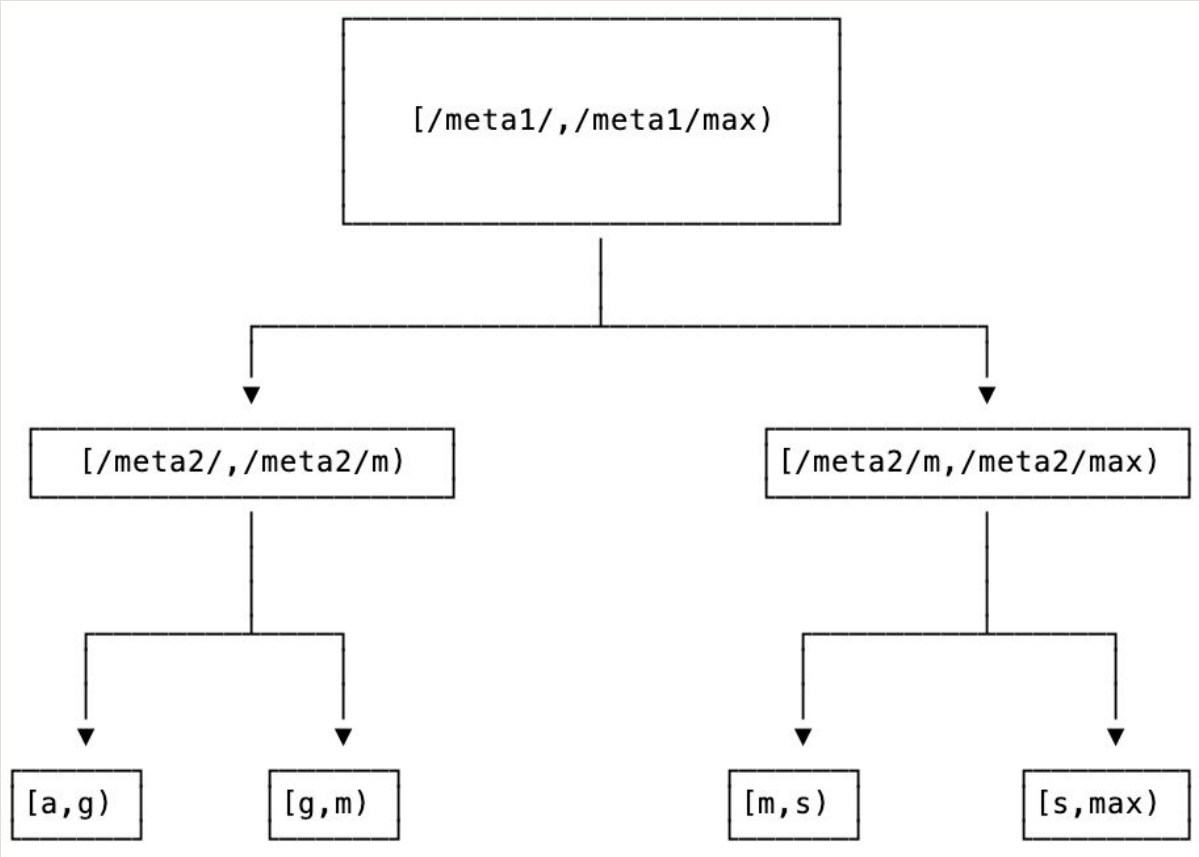


key	value
dog/34/name	**carl**
dog/34/weight	**10.1**
…	…
dog/7A/name	dagne
dog/7A/weight	13.4

Literally a range of Data (Key Value Pairs)



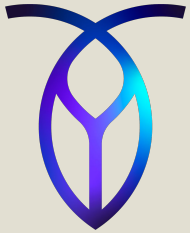
Keys up to M can be found here



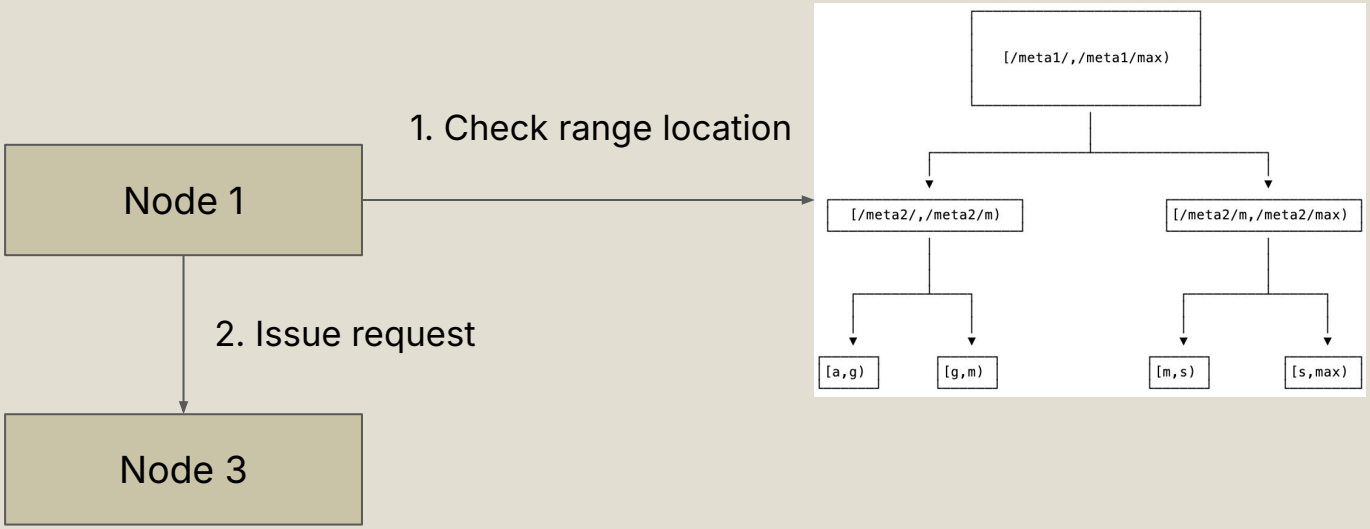
Keys from M to Max can be found here

Actual data

Meta Ranges: 2 level index
Used by Distribution Layer



```
INSERT INTO users
VALUES (1, 'victor_neo', '...', …);
```



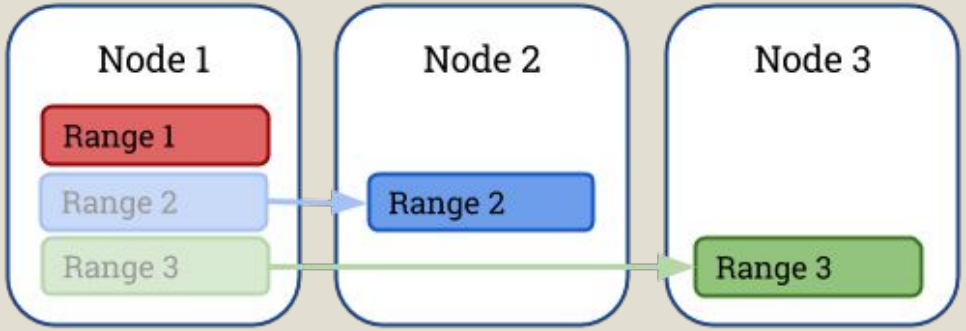
Addresses of each replica is stored in the meta2 range

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

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

Eg. Primary = Primary ID % 2

Sounds similar to Partitioning?



Range Distribution Across Nodes in CockroachDB

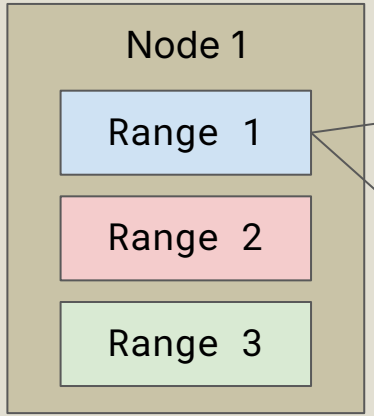
`range_max_bytes`: The maximum size, in bytes, for a range of data in the zone. When a range reaches this size, CockroachDB will split it into two ranges.

Default: 536870912 (512 MiB)

A range in CockroachDB (called a “shard” in other databases) is a chunk of data, stored as key-value pairs. The Distribution Layer breaks tables apart into these chunks so the data can be distributed across different nodes.

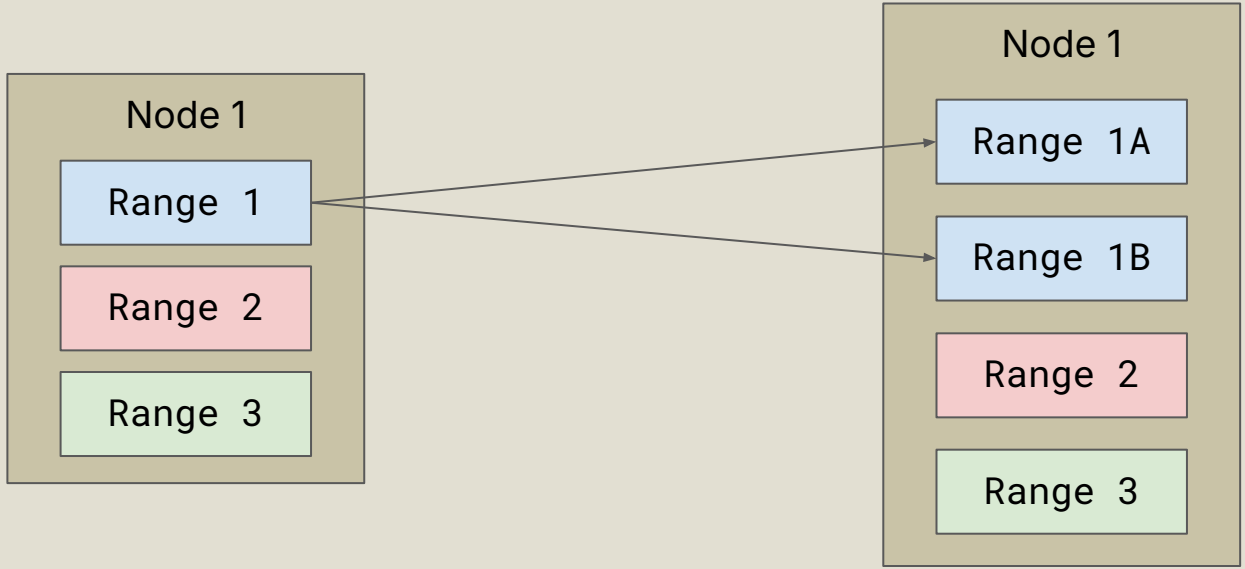
CockroachDB High-Level Architecture

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

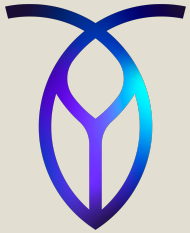


key	value
dog/34/name	carl
dog/34/weight	10.1
…	…
dog/7A/name	dagne
dog/7A/weight	13.4

What if the range gets too big or hot?



Range is split if its too big or too hot



Load-Based Splitting

To optimize your cluster's performance, CockroachDB can split frequently accessed keys into smaller ranges. In conjunction with load-based rebalancing, load-based splitting distributes load evenly across your cluster.

Enable and disable load-based splitting

Load-based splitting is enabled by default. To enable and disable load-based splitting, set the `kv.range_split.by_load_enabled` cluster setting. For example, to disable load-based splitting, execute:

```
> SET CLUSTER SETTING kv.range_split.by_load_enabled = false;
```

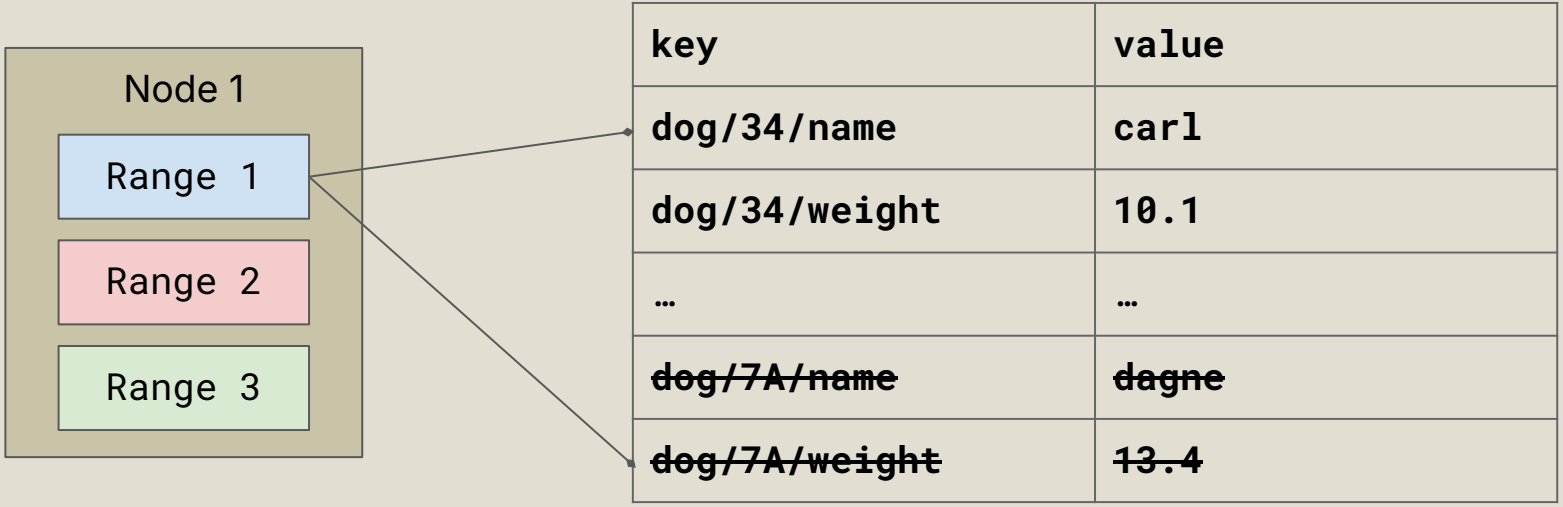
When to enable load-based splitting

Load-based splitting is on by default and beneficial in almost all situations.

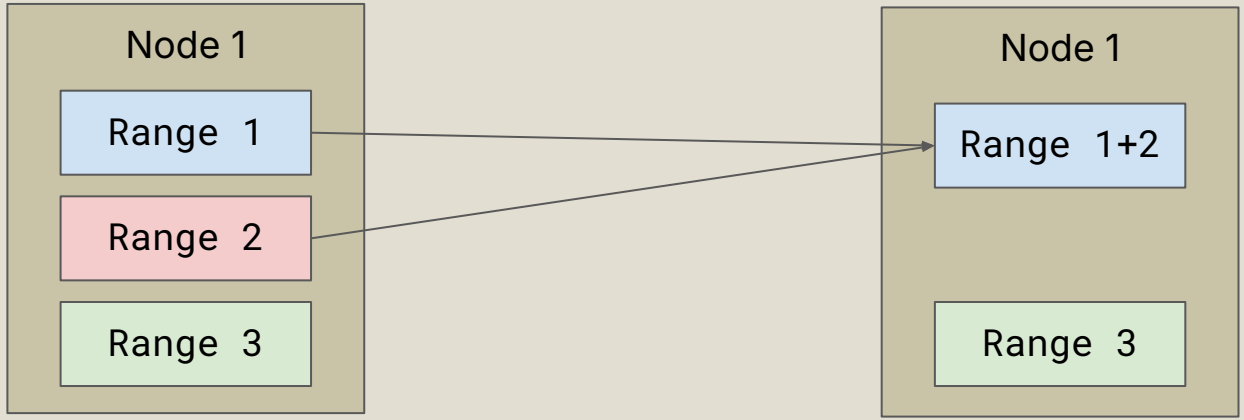
When to disable load-based splitting

You might want to disable load-based splitting when troubleshooting range-related issues under the guidance of Cockroach Labs support.

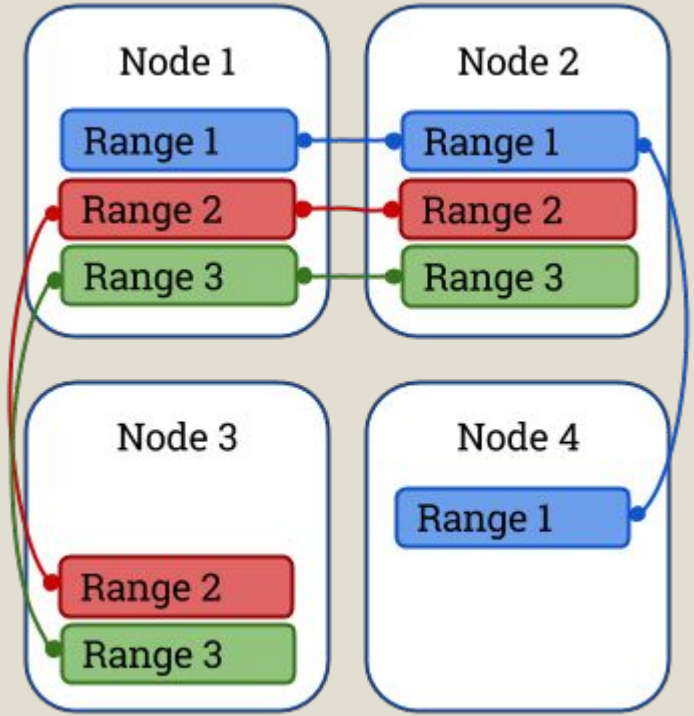
Load-Based Splitting Automates the Process



What if the range gets too small or cold?



Ranges can be merged



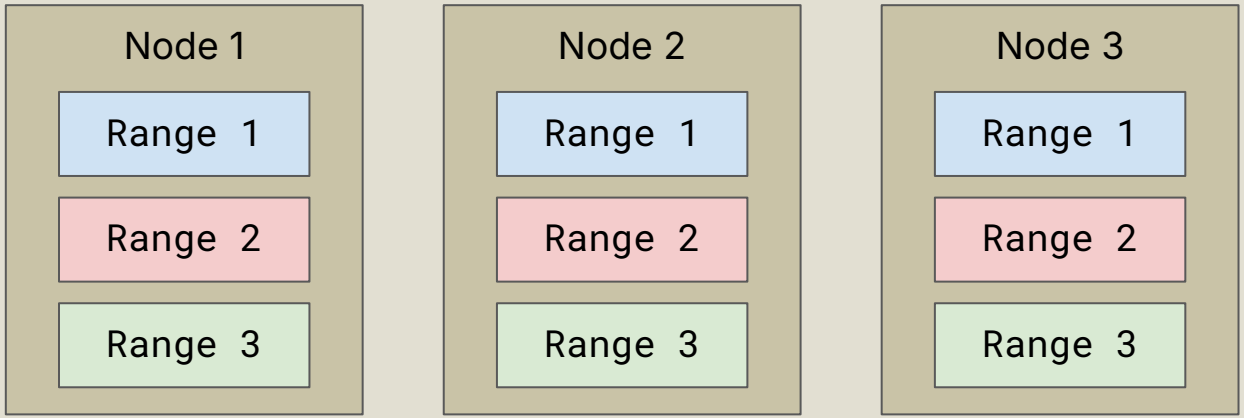
Range Replication in CockroachDB

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

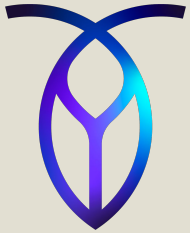
Number of replicas = 3 by default in CockroachDB

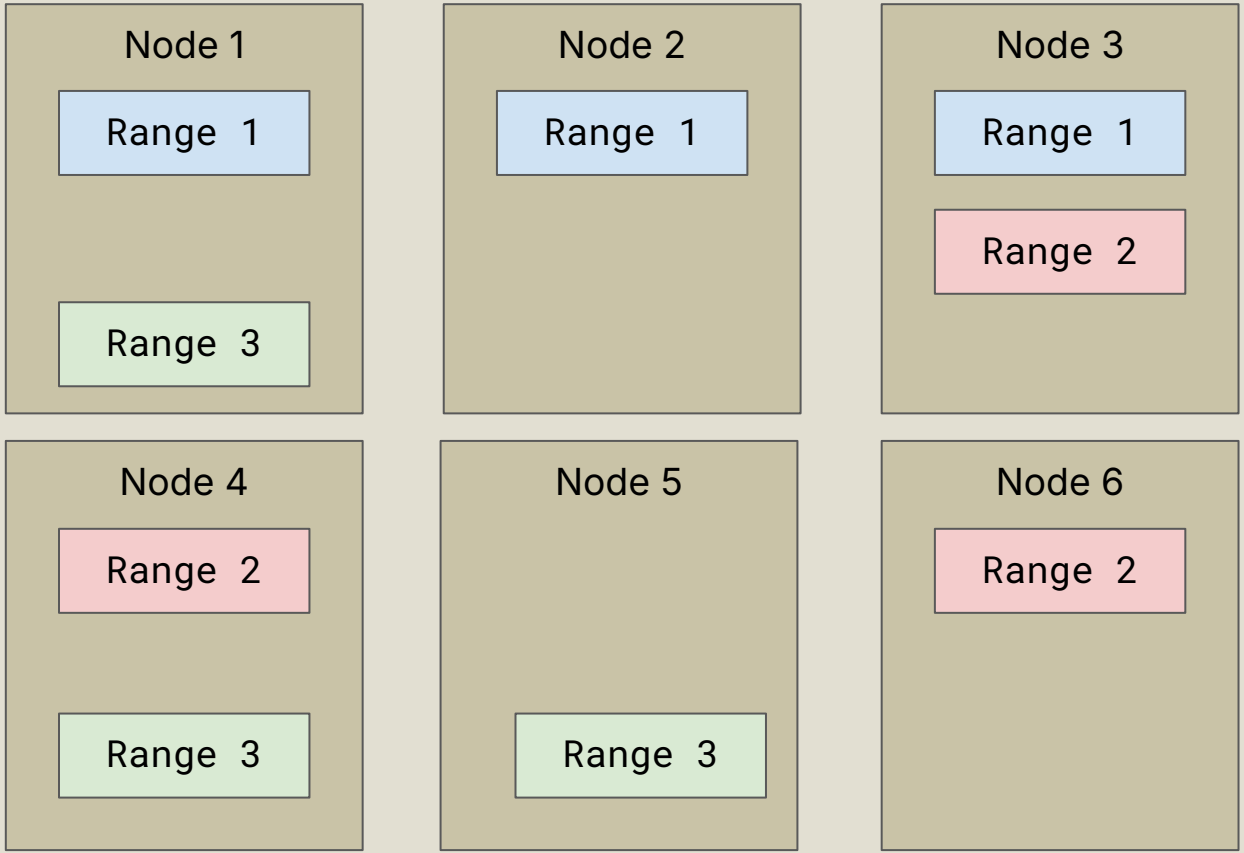
Range Replication on Nodes

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



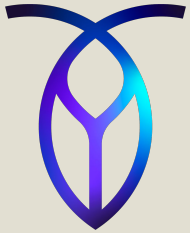
Replication Factor of 3 on 3 Nodes

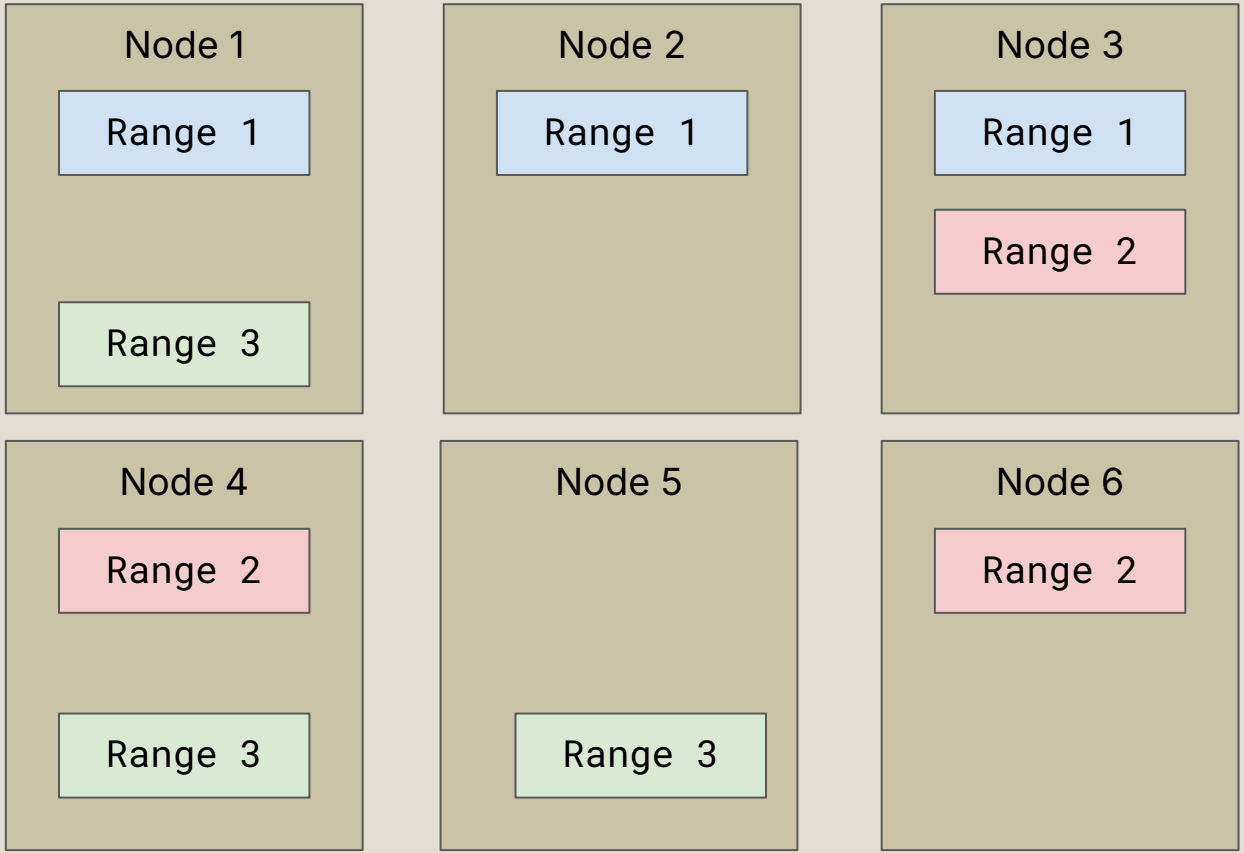




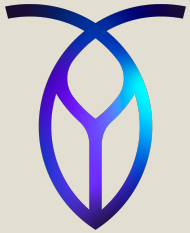
Replication Factor of 3 on 6 Nodes

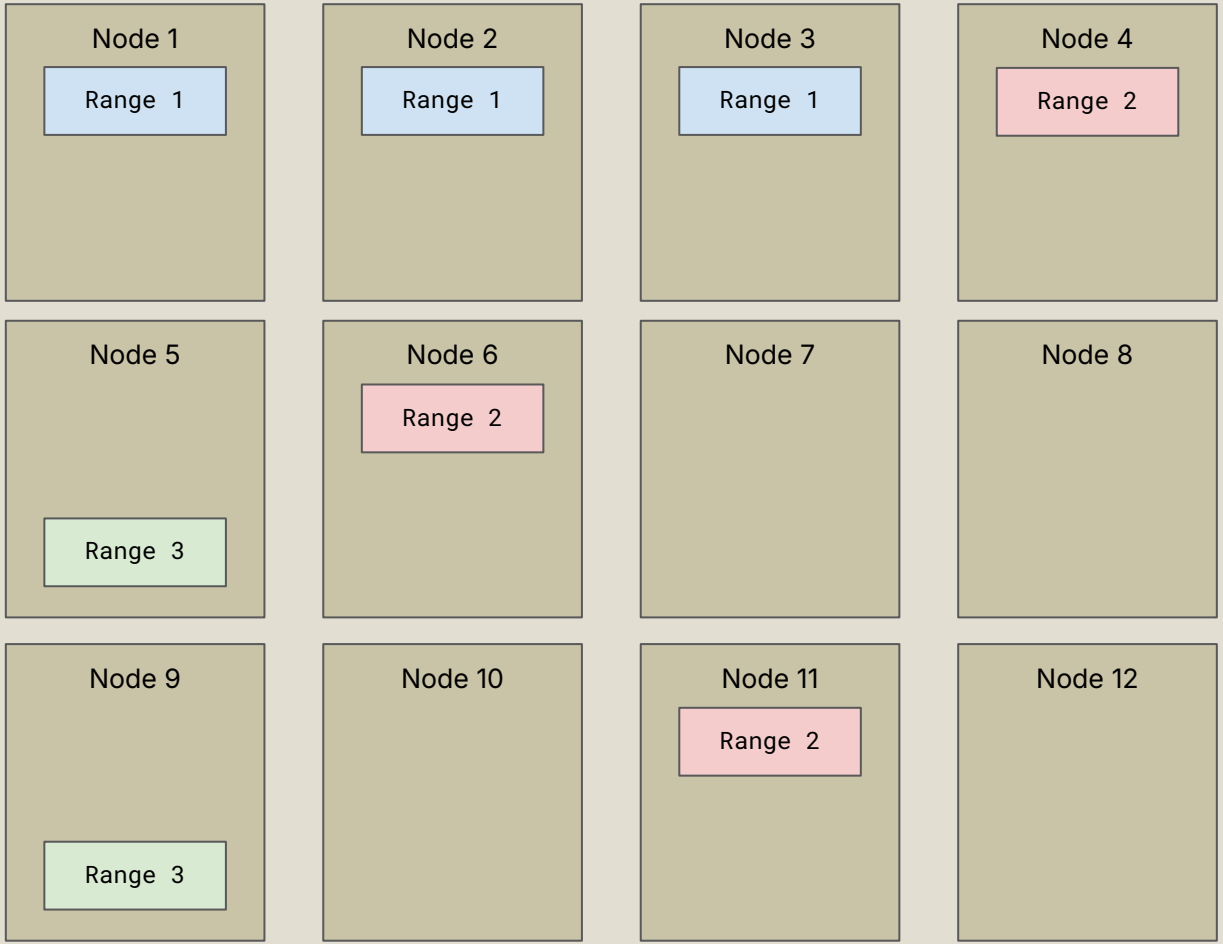
Can be randomly distributed as long as each range is on 3 nodes

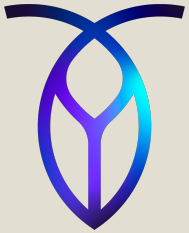


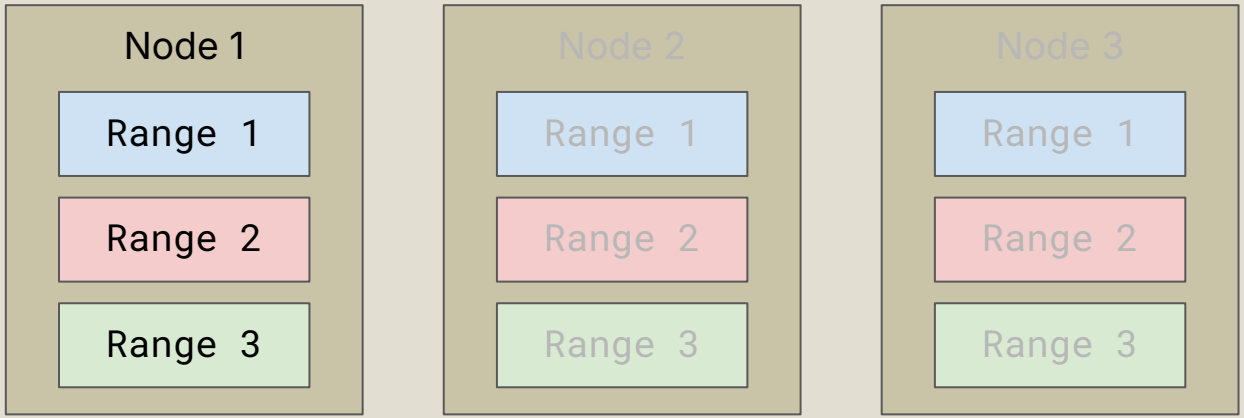


Replication Factor of 3 on 12 Nodes?



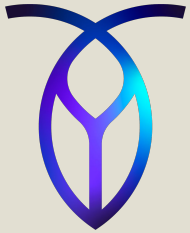

Node 1
Range 1
Node 2
Range 1
Node 3
Range 1
Node 4
Range 2
Node 5
Range 3
Node 6
Range 2
Node 7
Node 8
Node 9
Range 3
Node 10
Node 11
Range 2
Node 12





Seems ok?

Why can we only tolerate 1 node failure?



“**Raft organizes all nodes that contain a replica of a range into a group--unsurprisingly called a Raft group**. Each replica in a Raft group is either a "leader" or a "follower".

The leader, which is elected by Raft and long-lived, coordinates all writes to the Raft group. It heartbeats followers periodically and keeps their logs replicated. In the absence of heartbeats, followers become candidates after randomized election timeouts and proceed to hold new leader elections.”

Replication Layer uses Raft

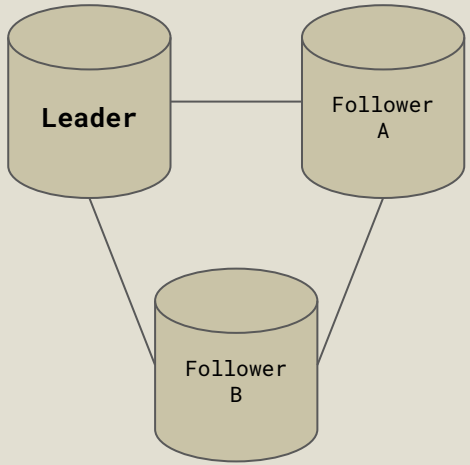
https://www.cockroachlabs.com/docs/stable/frequently-asked-questions

- There is always a **Leader**

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

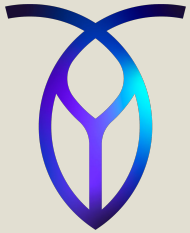
- When Leader goes missing, Followers will put themselves up for voting to choose next Leader **(Leader Election)**

- 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



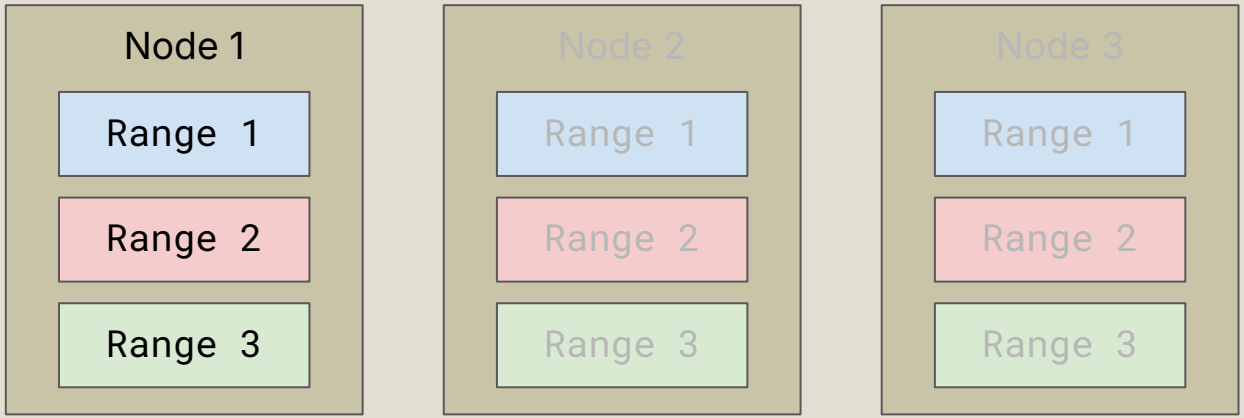
Raft Algorithm: Quick Summary

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



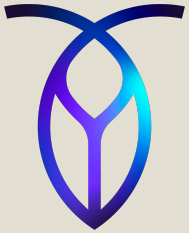
Max node failures = min(floor((n-1)/2), r-1)

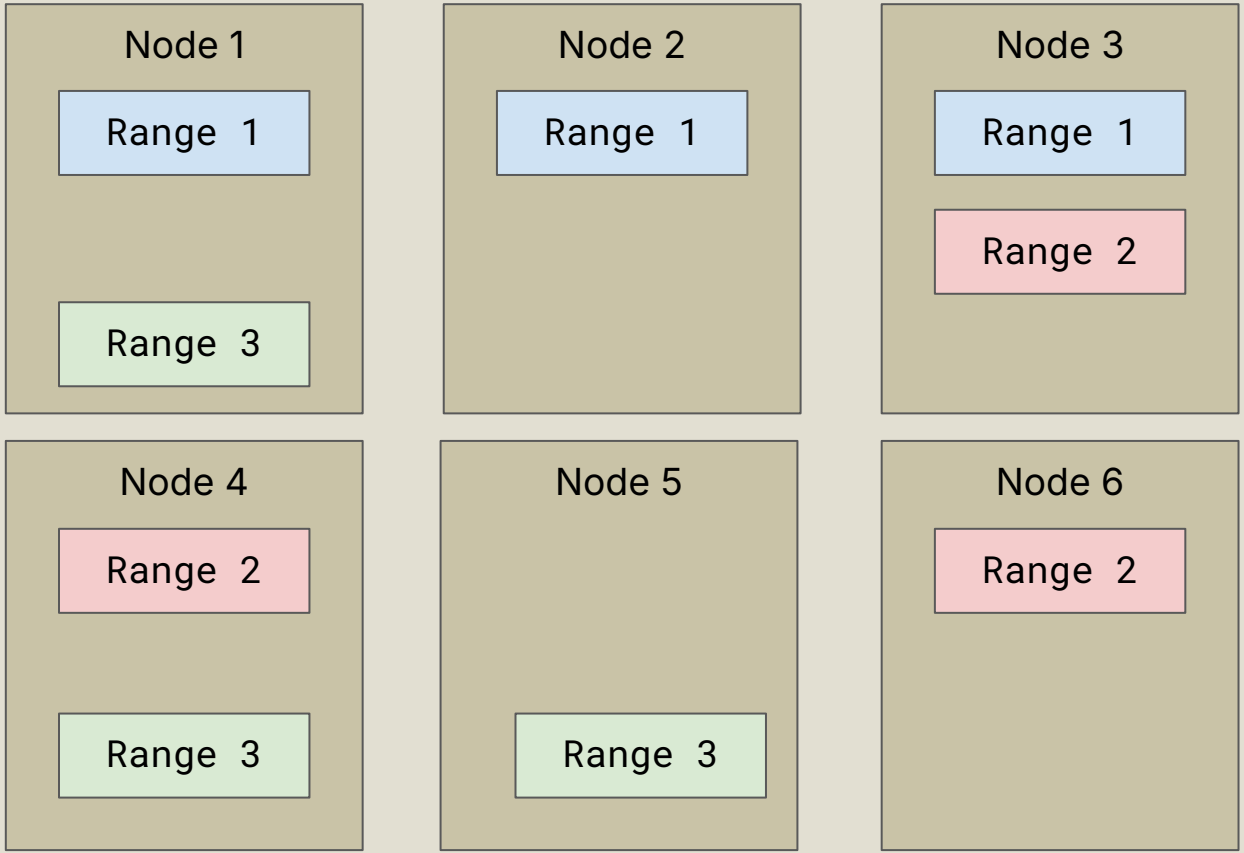
n = no. of nodes, r = replication factor



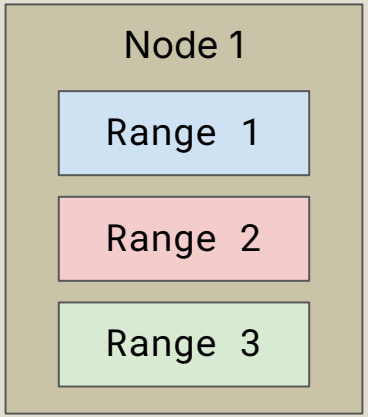
Max node failures = min(floor((3-1)/2), 3-1)
= min(1, 2) = 1

Consensus (Quorum) vs Data Available

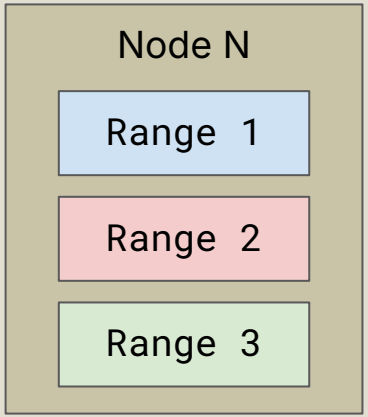




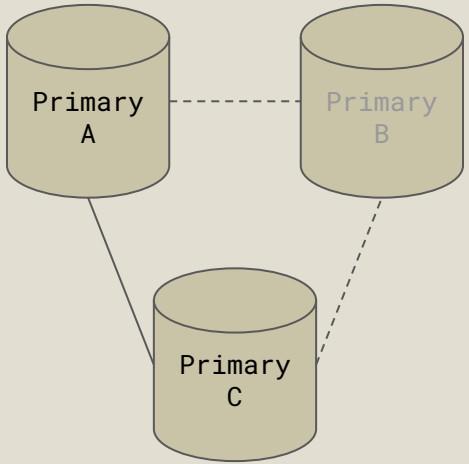
Max node failures = min(floor((6-1)/2), 3-1)
= min(2, 2) = 2



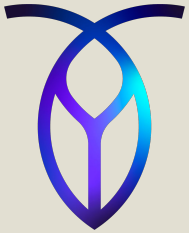
Common Cluster Sizes:
3, 5, 7, 9, ..., N

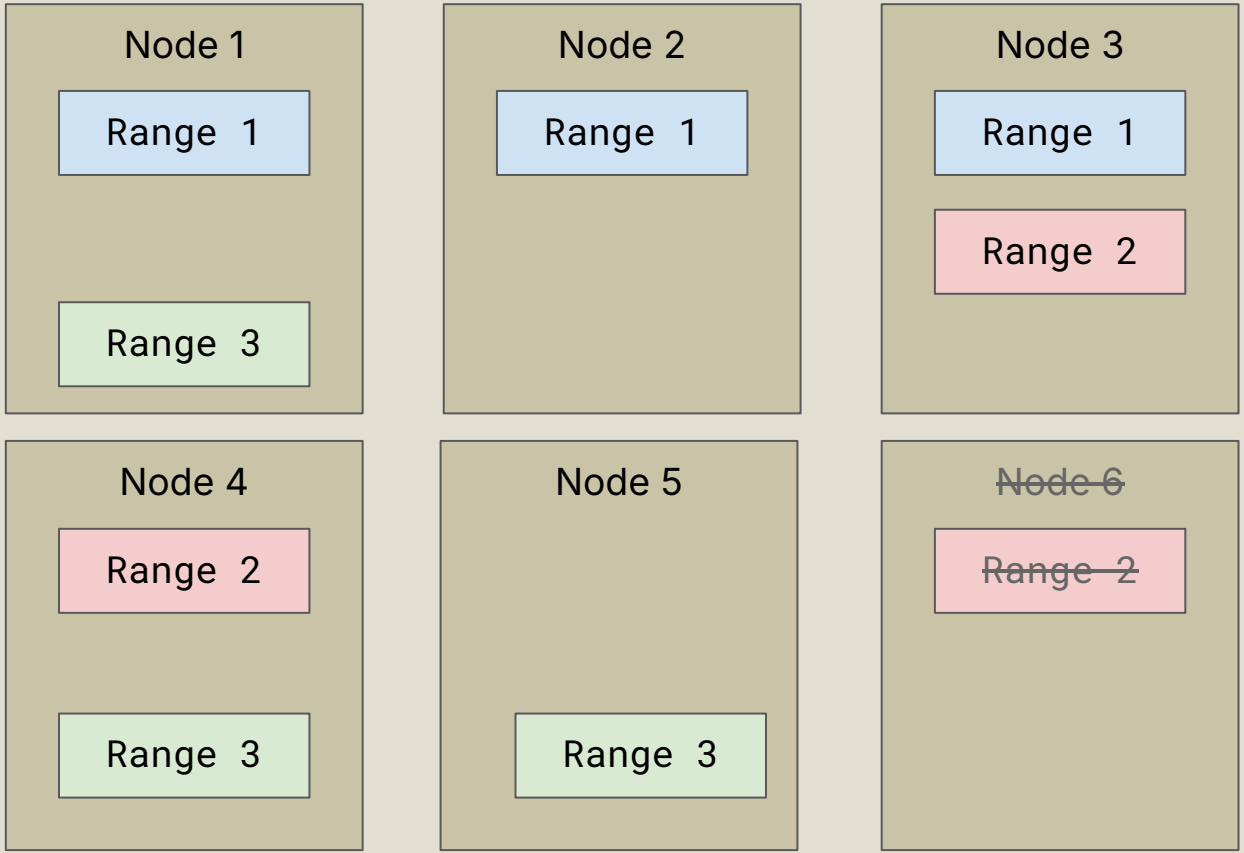


Cluster Size	Quorum	Max No. Failures
3	2	1
4	3	1
5	3	2
6	4	2



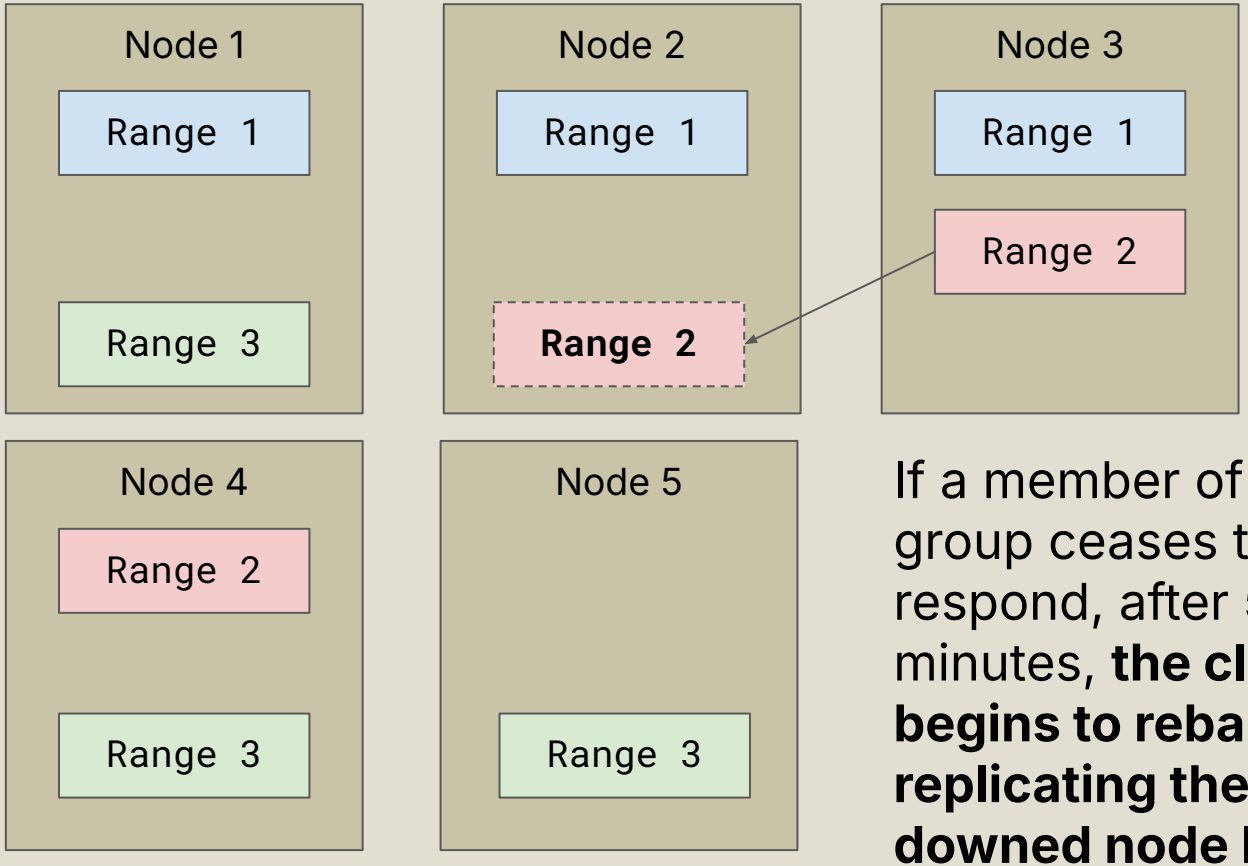
Q. What happens when a node disappears?



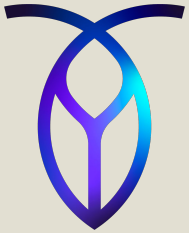


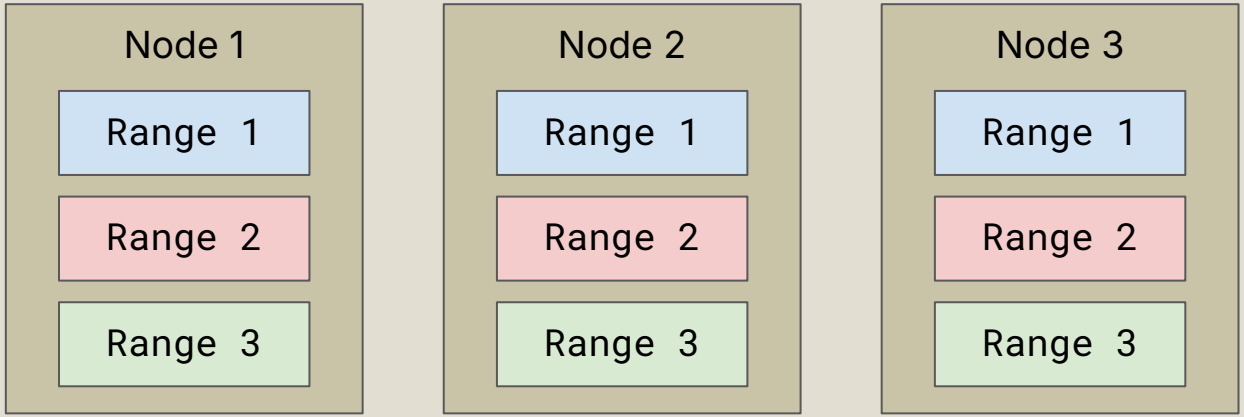
Membership Change: Node Offline

https://www.cockroachlabs.com/docs/stable/architecture/replication-layer#membership-changes-rebalance-repair



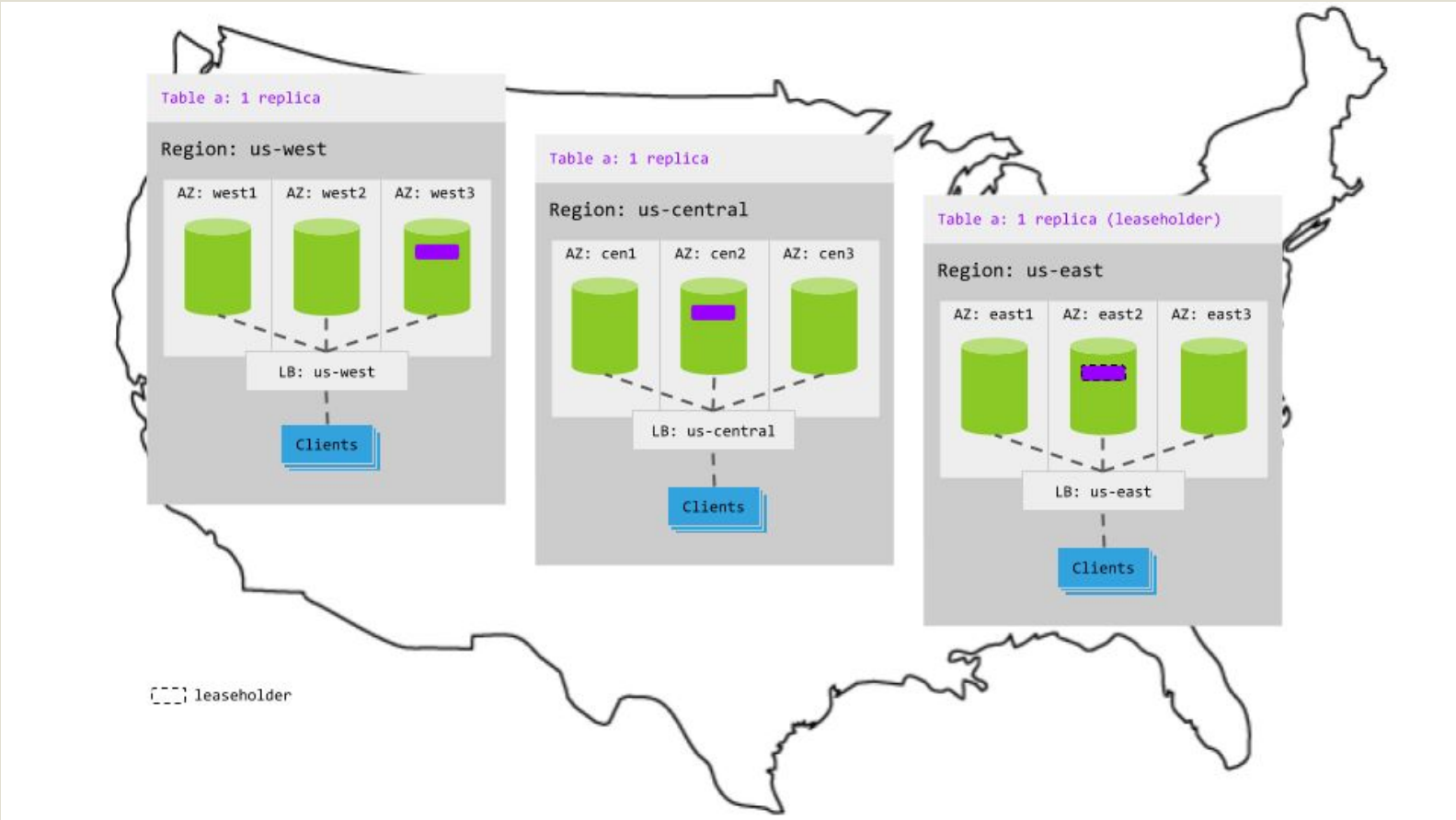
If a member of a Raft group ceases to respond, after 5 minutes, **the cluster begins to rebalance by replicating the data the downed node held onto other nodes**.





Cluster size: Bigger clusters are more expensive to run as more nodes are required, **consensus takes a longer time to achieve**, and maintenance (eg. software updates) can be painful

Replication factor: Higher means more data (ranges) is backed up but more network bandwidth + storage space needed



Topology Patterns Overview

https://www.cockroachlabs.com/docs/stable/topology-patterns

Summary of the AWS Service Event in the US East Region

July 2, 2012

We'd like to share more about the service disruption which occurred last Friday night, June 29th, in one of our Availability Zones in the US East-1 Region. **The event was triggered during a large scale electrical storm which swept through the Northern Virginia area**. We regret the problems experienced by customers affected by the disruption and, in addition to giving more detail, also wanted to provide information on actions we'll be taking to mitigate these issues in the future.

...

Approximately 7% of the EC2 instances in the US-EAST-1 Region were in the impacted Availability Zone and impacted by the power loss. These instances were offline until power was restored and systems restarted.

https://aws.amazon.com/message/67457/

Zone Survival

You can accept a single node failure up to an entire zone failure. If multiple zones fail in the same region, the database may become unavailable.

Requires at least 3 zones in 1 region

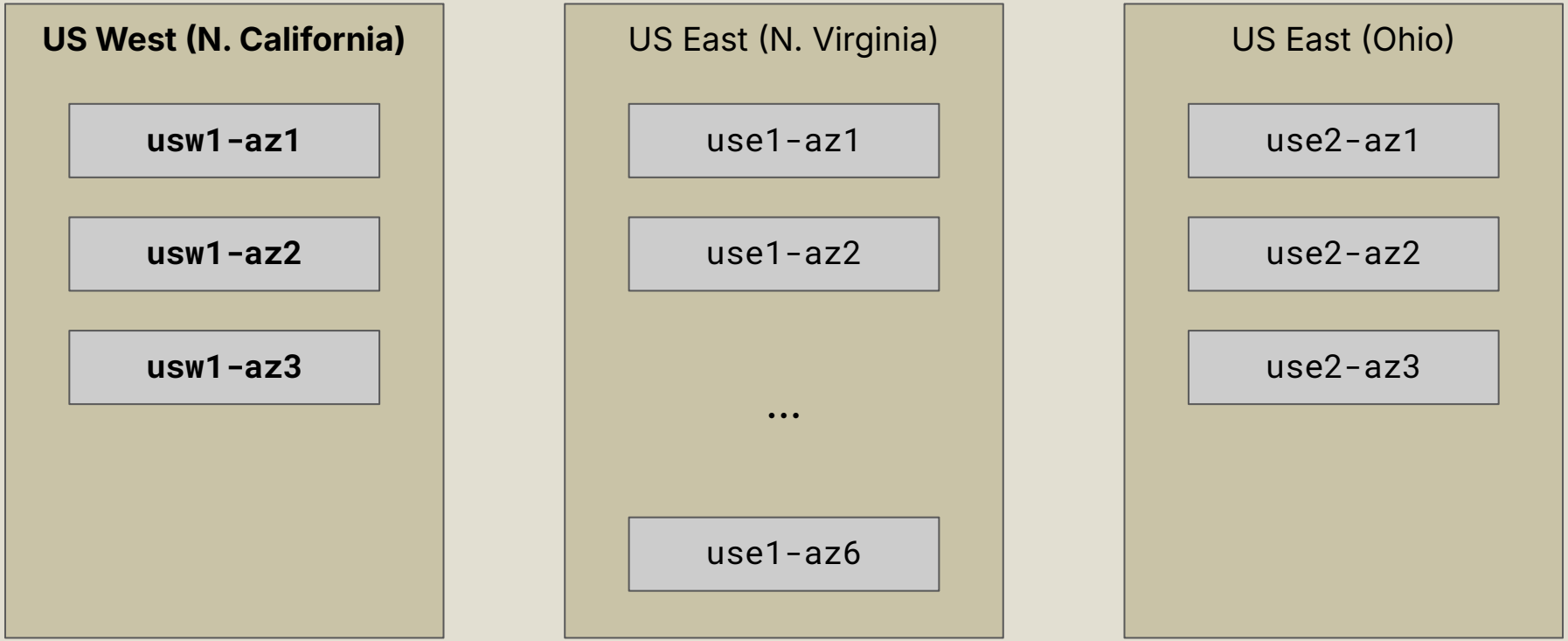
Region Survival

The database must remain available, even if a region goes down.

You can accept the performance cost: write latency will be increased by at least as much as the round-trip time to the nearest region.

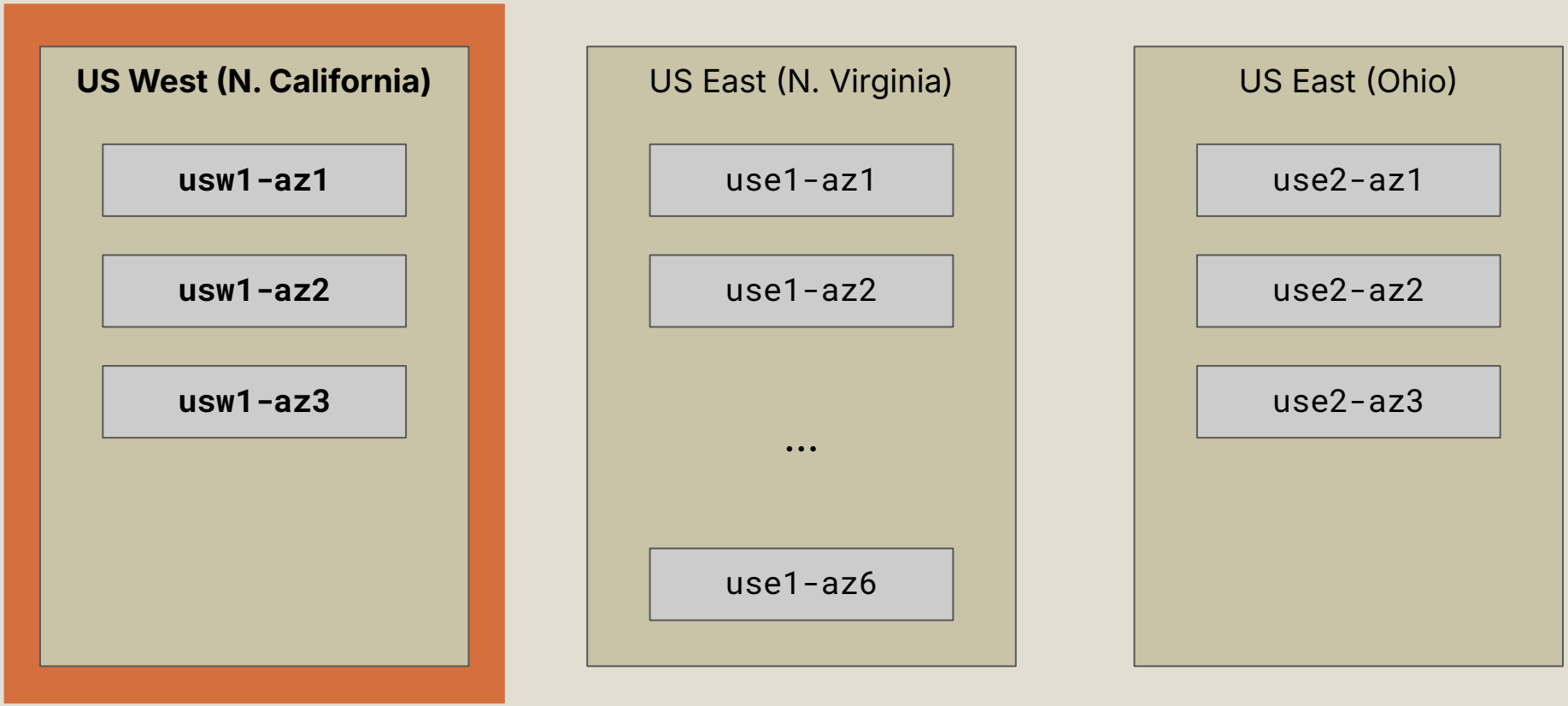
Assumes 3* or more regions by default

Multi-Region Survival Goals
https://www.cockroachlabs.com/docs/v25.1/multiregion-survival-goals



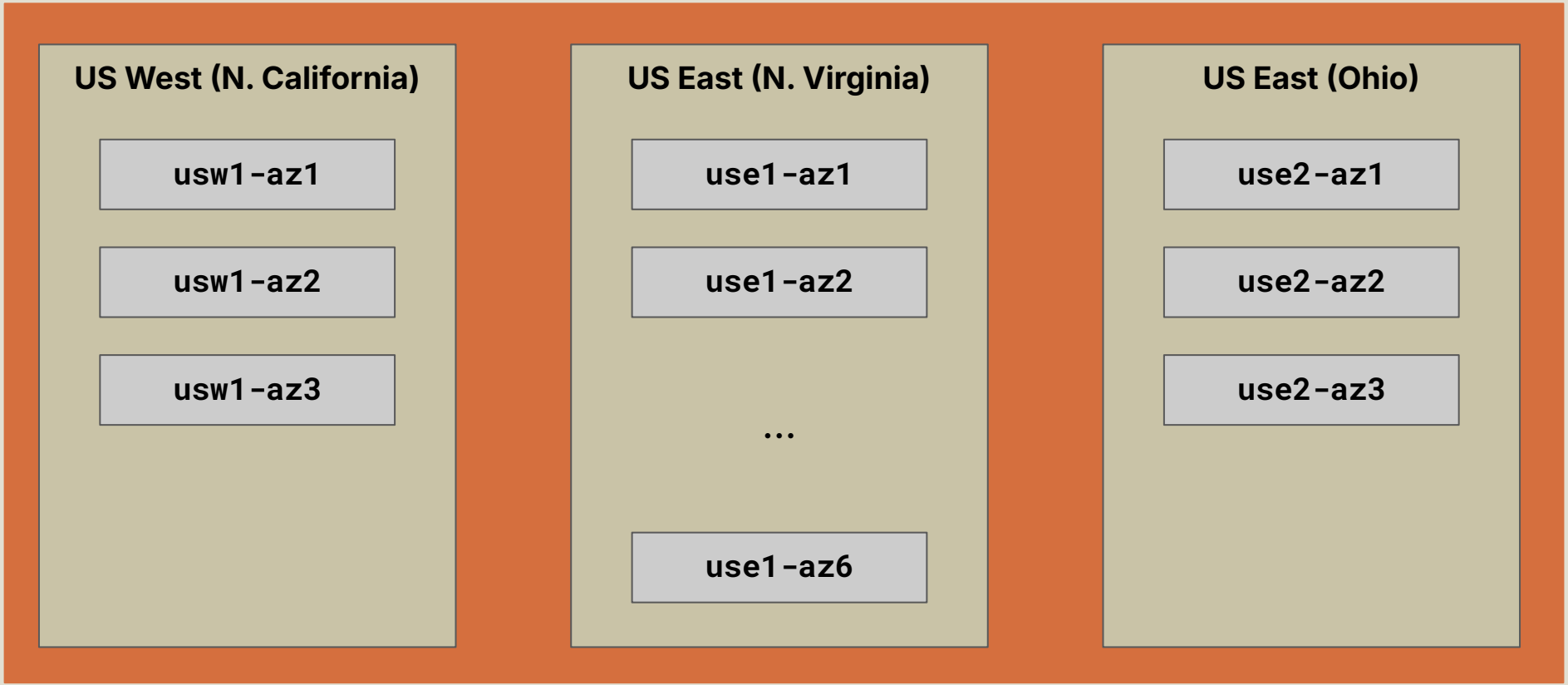
AWS Availability Zones

https://docs.aws.amazon.com/global-infrastructure/latest/regions/aws-availability-zones.html



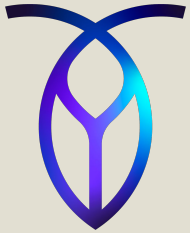
Zone Survival Minimum Requirements

At least 3 zones in 1 region



Region Survival Minimum Requirements

At least 3 regions

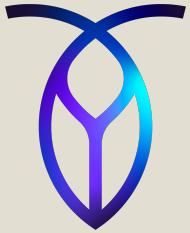


“**CockroachDB is a CP (consistent and partition tolerant) system.** This means that, in the presence of partitions, the system will become unavailable rather than do anything which might cause inconsistent results.

For example, writes require acknowledgments from a majority of replicas, and reads require a lease, which can only be transferred to a different node when writes are possible.”

CockroachDB is a CP Database

https://www.cockroachlabs.com/docs/stable/frequently-asked-questions



- **PostgreSQL Compatibility** 
- Presents itself as a relational database but is internally a Key-Value Store
- Data is stored in ranges which can be split or merged as required
- Range splitting and replica rebalancing based on Load allows for “auto”-scaling to some degree
- Uses Raft internally for consensus + offers CP
- Replication strategies allow you to scale horizontally within a single region or across multiple regions

CockroachDB Summary

Databases

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

Assumes 1 machine by default

Distributed Databases

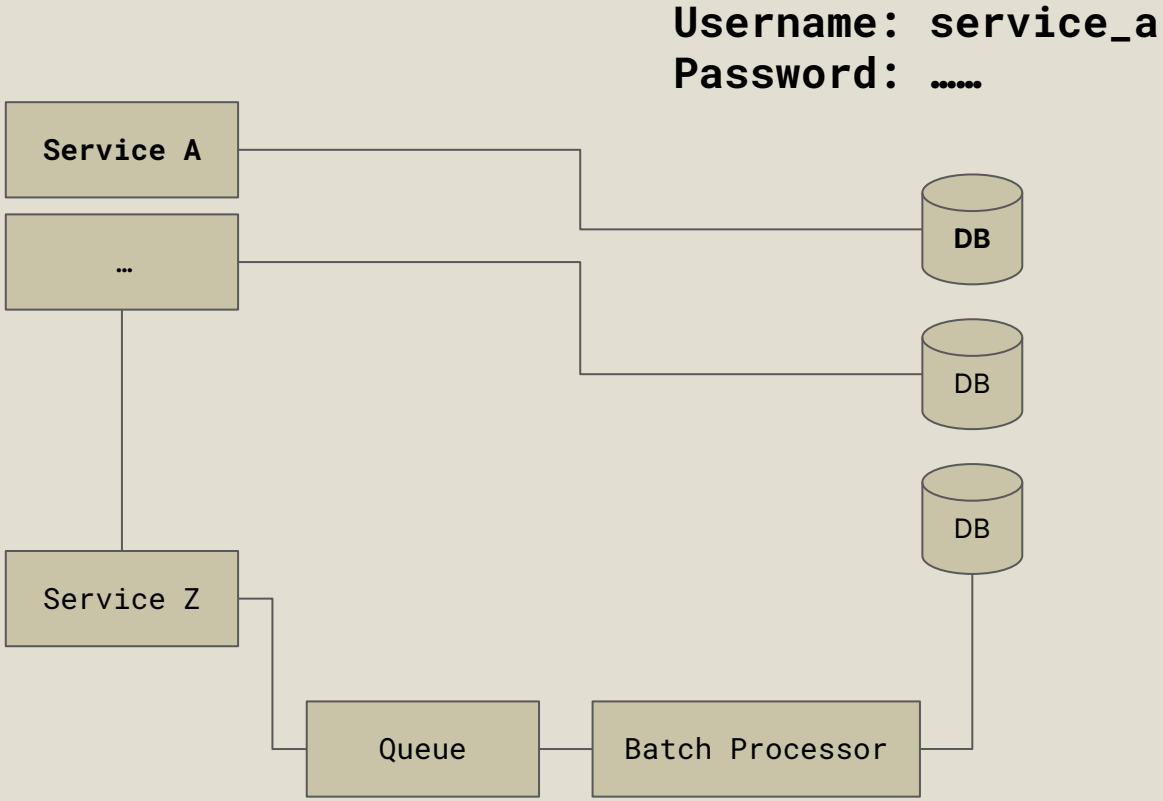
“A distributed database is a database that runs and stores data across multiple computers, as opposed to doing everything on a single machine.”

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/

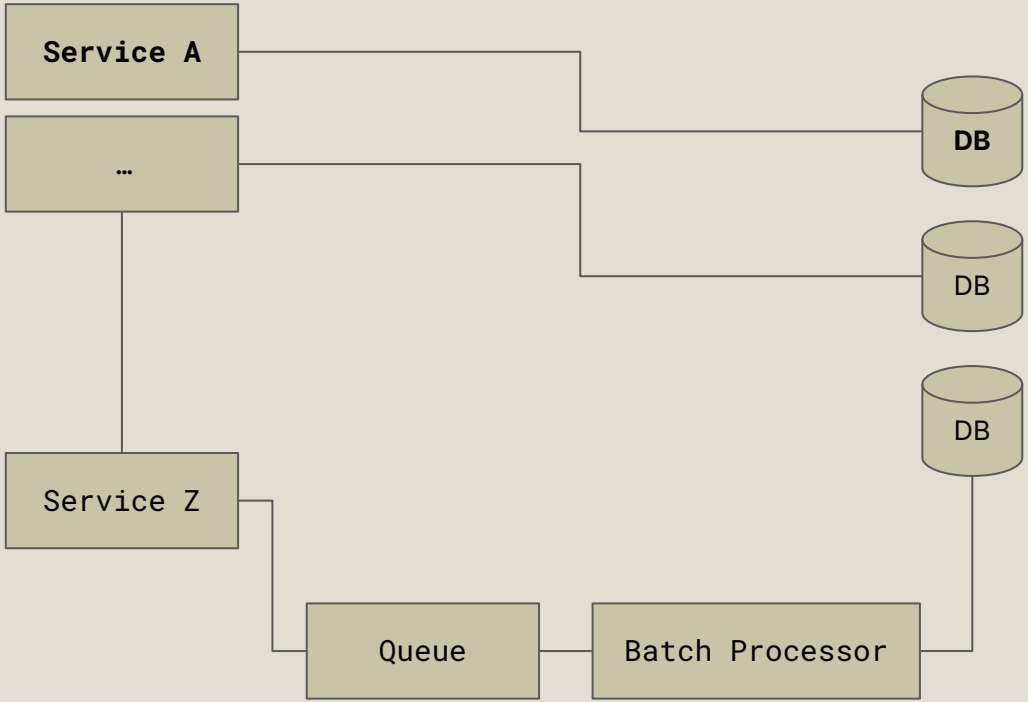
3.2.1 hashicorp vault

Distributed Data Store
Securely store and access secrets for services

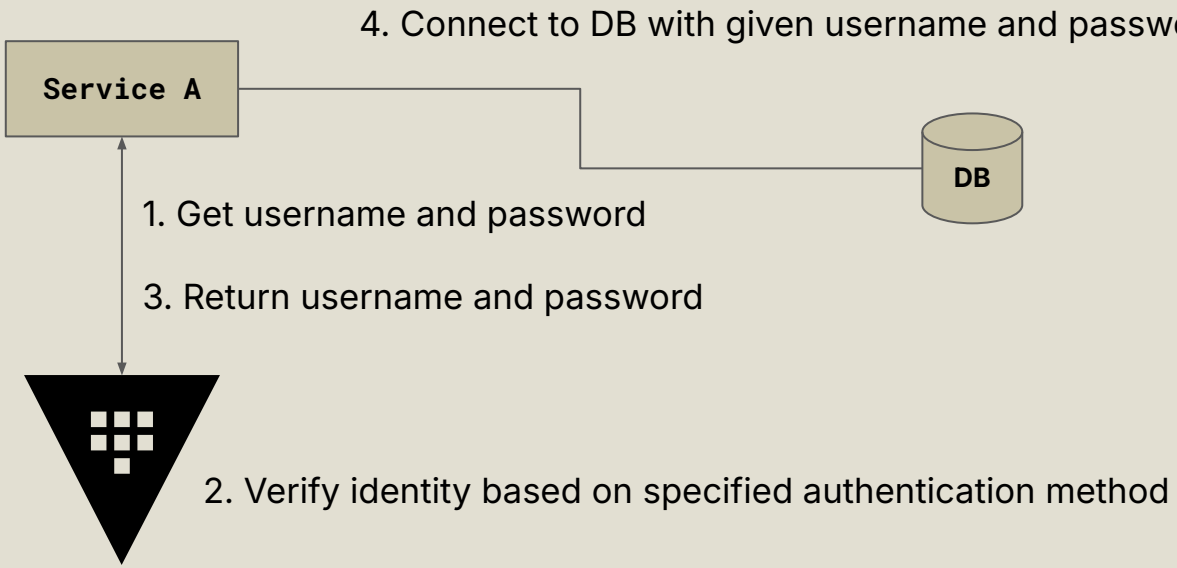


Hardcoded Passwords

```
.env
PG_USERNAME=service_a
PG_PASSWORD=...
```



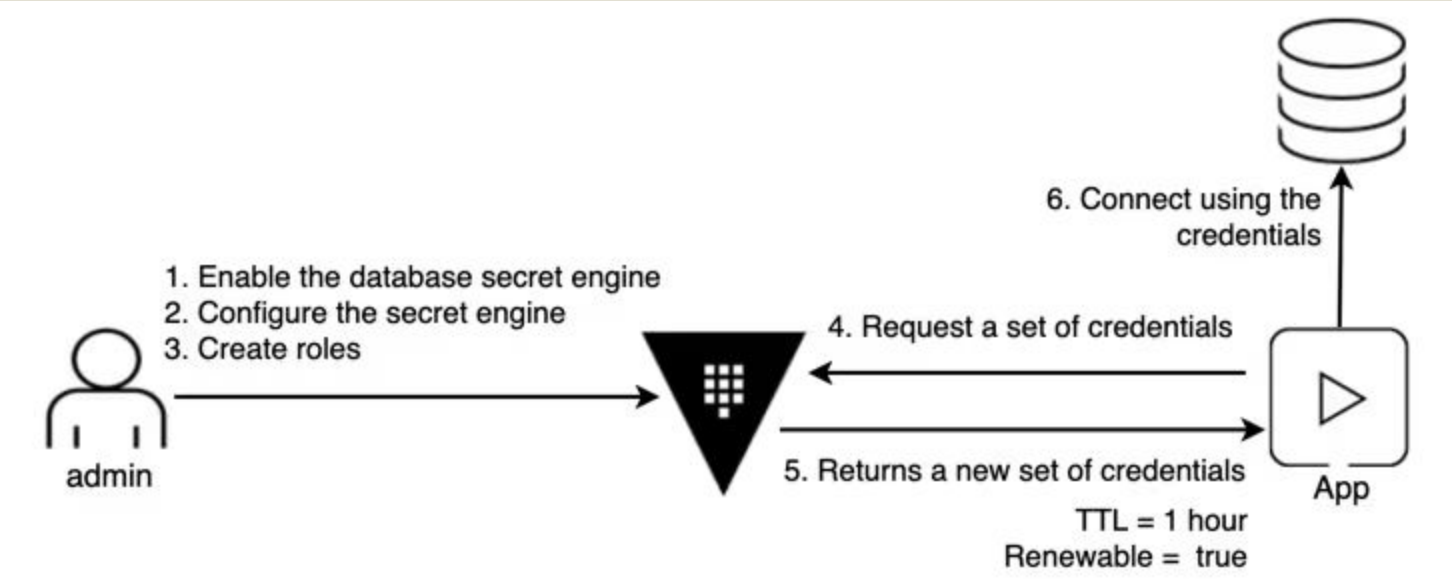
Passwords littered all around



Static secrets for credential management

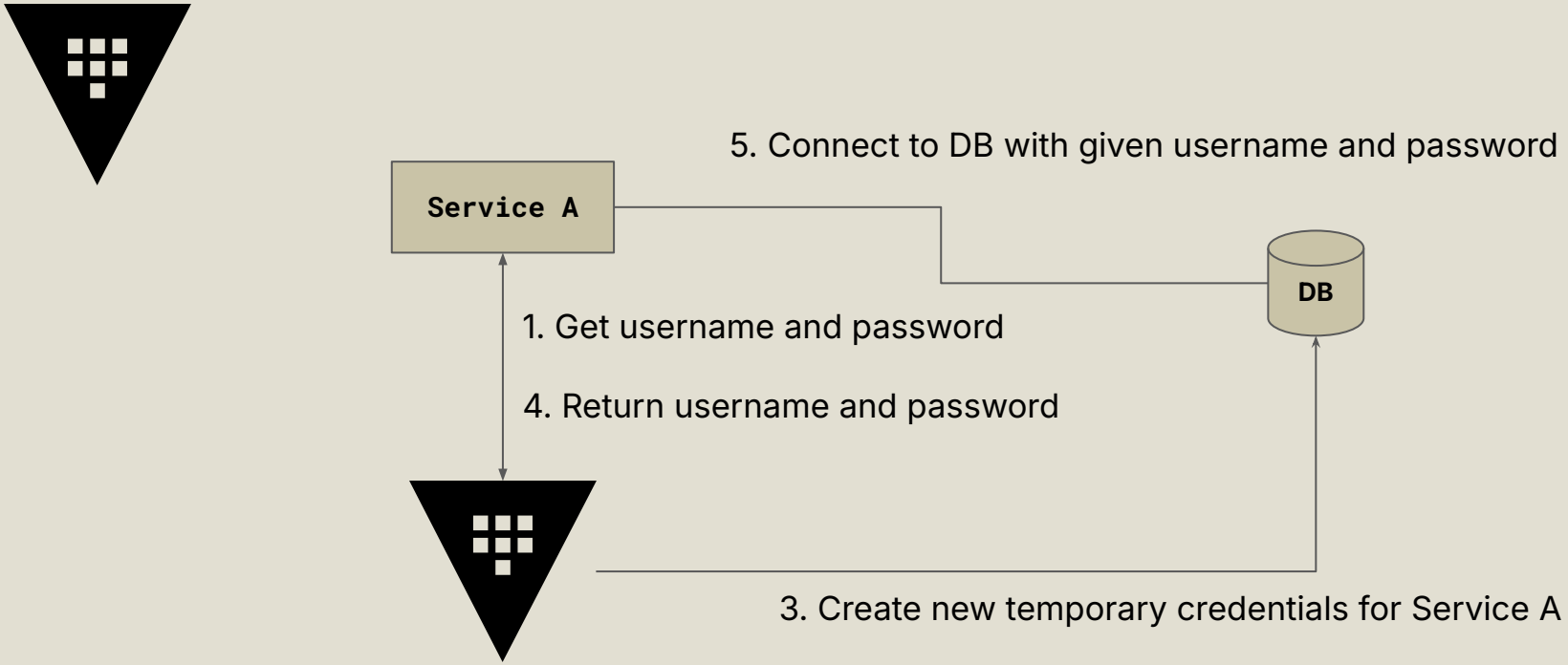
https://developer.hashicorp.com/vault/tutorials/db-credentials/database-secrets

Vault generates a new username and password in the DB for every new access request



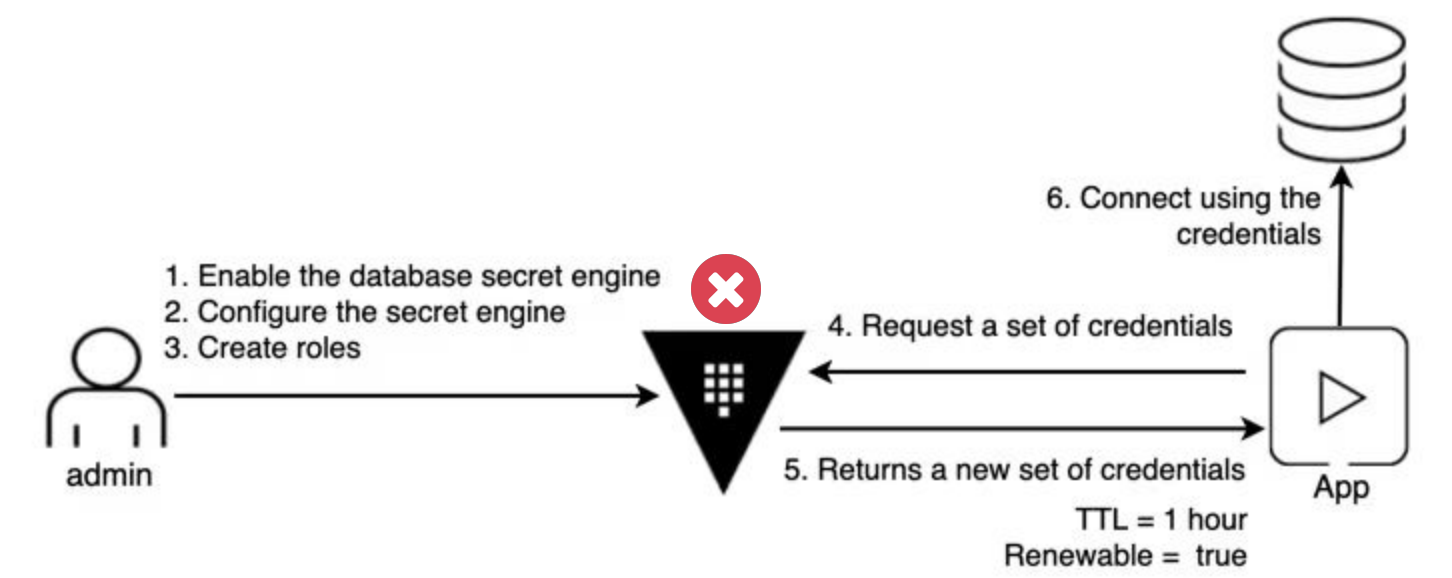
Dynamic secrets for credential management

https://developer.hashicorp.com/vault/tutorials/db-credentials/database-secrets



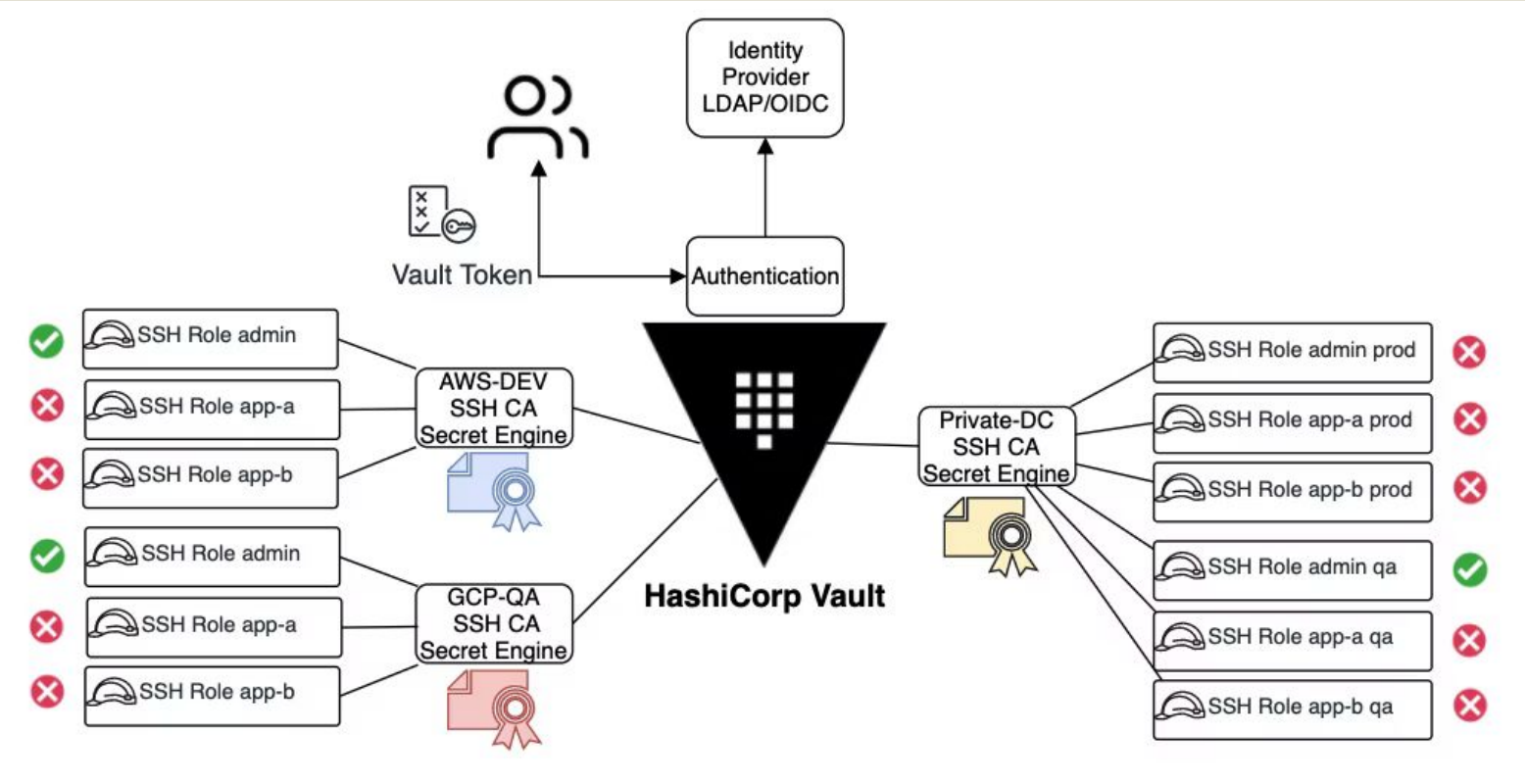
Dynamic secrets for credential management

https://developer.hashicorp.com/vault/tutorials/db-credentials/database-secrets



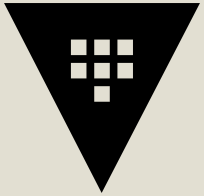
Really Important Not To Fail

https://developer.hashicorp.com/vault/tutorials/db-credentials/database-secrets



Also used for server access

https://www.hashicorp.com/en/blog/managing-ssh-access-at-scale-with-hashicorp-vault



High availability

v1.19.0 (latest)

Vault can run in a high availability (HA) mode to protect against outages by running multiple Vault servers.

Design overview

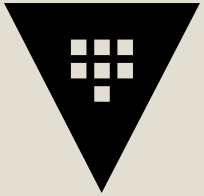
The primary design goal for making Vault Highly Available (HA) is to minimize downtime without affecting horizontal scalability. Vault is bound by the IO limits of the storage backend rather than the compute requirements. Being bound by the IO limits simplifies the HA approach and avoids complex coordination.

Storage backends, such as Integrated Storage, provide additional coordinative functions enabling Vault to run in an HA configuration. Supported by the backend, Vault will automatically run in HA mode without further configuration.

When running in HA mode, Vault servers have two states they can be: **standby** and **active**. For multiple Vault servers sharing a storage backend, only a single instance is active at any time. All standby instances are placed in hot standbys.

High Availability Mode

https://developer.hashicorp.com/vault/docs/internals/high-availability



Leadership elections

Nodes become the Raft leader through Raft leadership elections.

All nodes in a Raft cluster start as **followers**. Followers monitor leader health through a **leader heartbeat**. If a follower does not receive a heartbeat within the configured **heartbeat timeout**, the node becomes a **candidate**. Candidates watch for election notices from other nodes in the cluster. If the **election timeout** period expires, the candidate starts an election for leader. If the candidate gets responses from a quorum of other nodes in the cluster, the candidate becomes the new leader node.

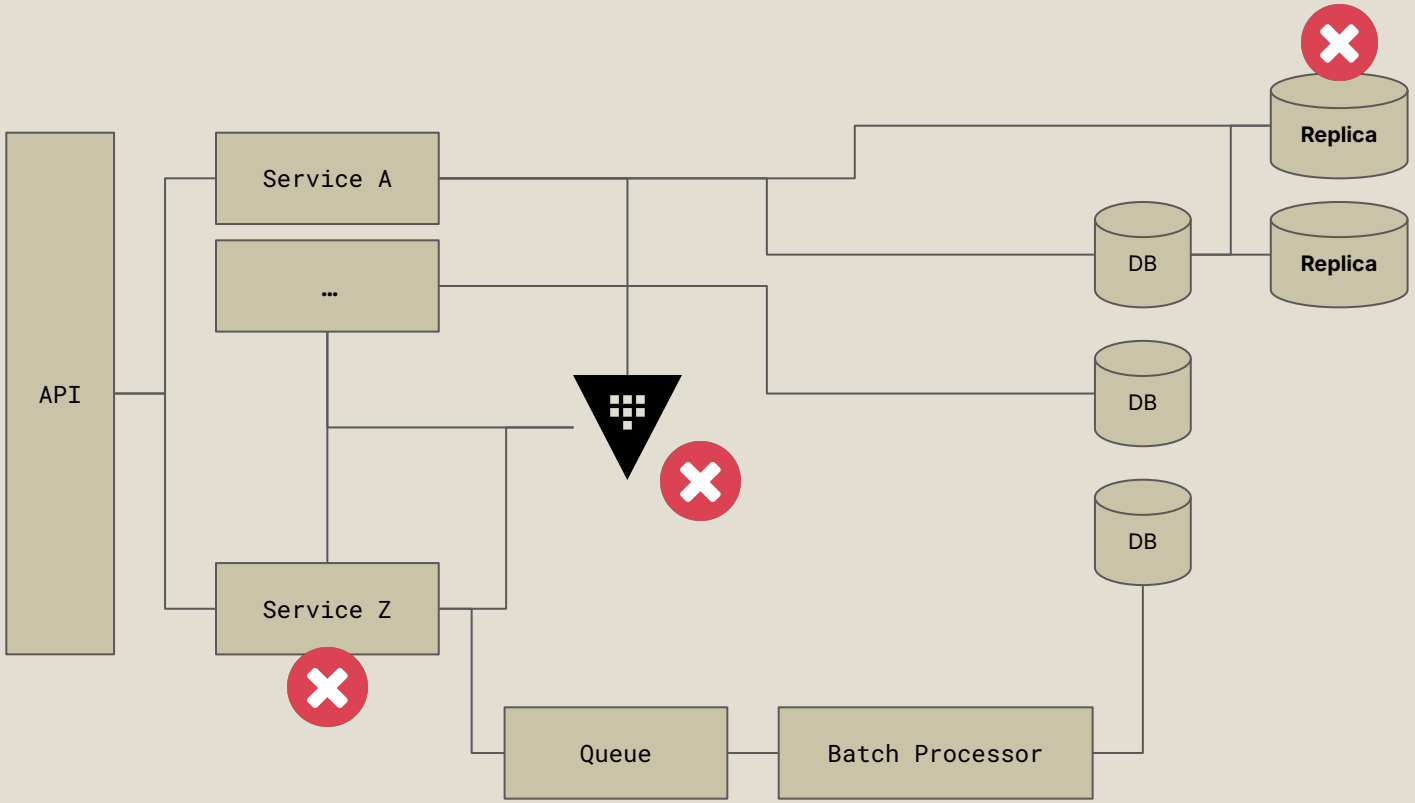
Raft leaders may step down voluntarily if the node cannot connect to a quorum of nodes with the **leader lease timeout** period.

The relevant timeout periods (heartbeat timeout, election timeout, leader lease timeout) scale according to the `performance_multiplier` setting in your Vault configuration. By default, the `performance_multiplier` is 5, which translates to the following timeout values:

Timeout	Default duration
Heartbeat timeout	5 seconds
Election timeout	5 seconds
Leader lease timeout	2.5 seconds

Uses Raft Internally

https://developer.hashicorp.com/vault/docs/internals/integrated-storage#leadership-elections



New Tools = Higher Overhead, Increased Risks

- Covered how different databases offer horizontal scalability: replication, partitioning, multi-primary, etc. to manage the flow of data

- Database or Distributed Database? Make sure you understand how they perform and their operational requirements

- CAP theorem offers us a way to reason about how we want the distributed systems to perform in the presence of network partitions

- Raft is a popular algorithm used by many tools to offer CP

Databases: Summary

Break time

3.3 technical design docs

Peer Discussion

Review this RFC for the GitLab project in pairs.

Do you think the RFC is:

- Well-written? If so, why?
- Confusing or missing? If so, why?
- How would you improve it?

~10 mins, pick 1 person to share after discussing.

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)**

1. Objective of the document:
    - Make decisions? Inform? Record?

2. Audience:
    - Technical? Non-technical?
    - Read by people who have background on the topic, or no?

3. Length of document:
    - Too short or too long?

Assignment #1

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.

- Is the objective here to inform or to help make a decision?

- “You are a Senior Engineer in a team responsible for building a new product that will extend upon one of these platforms / protocols.”
  - You are very likely to be the or one of the key decision makers
  - What’s your personal stance? Yes or No to the chosen protocol
  - What’s your recommendation

Assignment #1

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

- Tech Leads and Software Engineers:
  - Technical Feasibility
  - Pitfalls and Advantages
  - Complexity, easy / difficult to implement

- Product Managers:
  - Time to ship
  - Product Feasibility
  - Pitfalls and Advantages

- Business Leaders:
  - Time to ship
  - Legal and compliance risk
  - Competitive advantage or disadvantage, etc.

RFCs / Request for Comments

Used to propose, or suggest new changes that will require feedback from others / stakeholders.

Example: New Database selection

Technical Plan

Typically used to document how changes are being planned and will be executed.

Example: Database Migration process

Retrospective / Post-Mortems

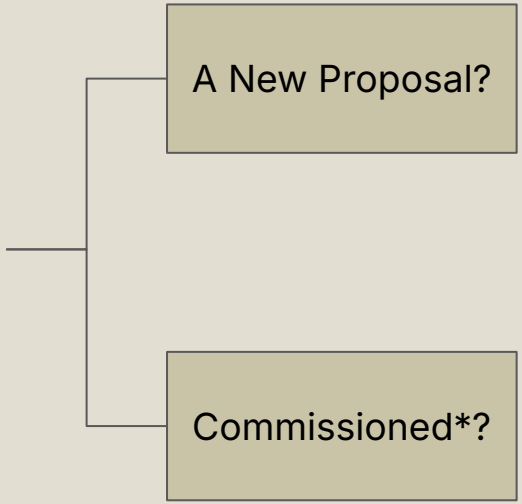
When an event occurs, retrospectives and post-mortems are run to capture learnings.

Does not necessarily need to be a negative event.

Example: DB downtime retrospective

Most common types of technical docs

Is this document...



Assume people have little to no context, and may be resistant* to the proposal.

*People are generally resistant to change

Assume people have some to little context, and may have interest in the proposal.

Is my objective to...

Drive or make a decision?

It’s important to take a stand and have an opinion on what decision to take, and highlight other options considered.

Inform?

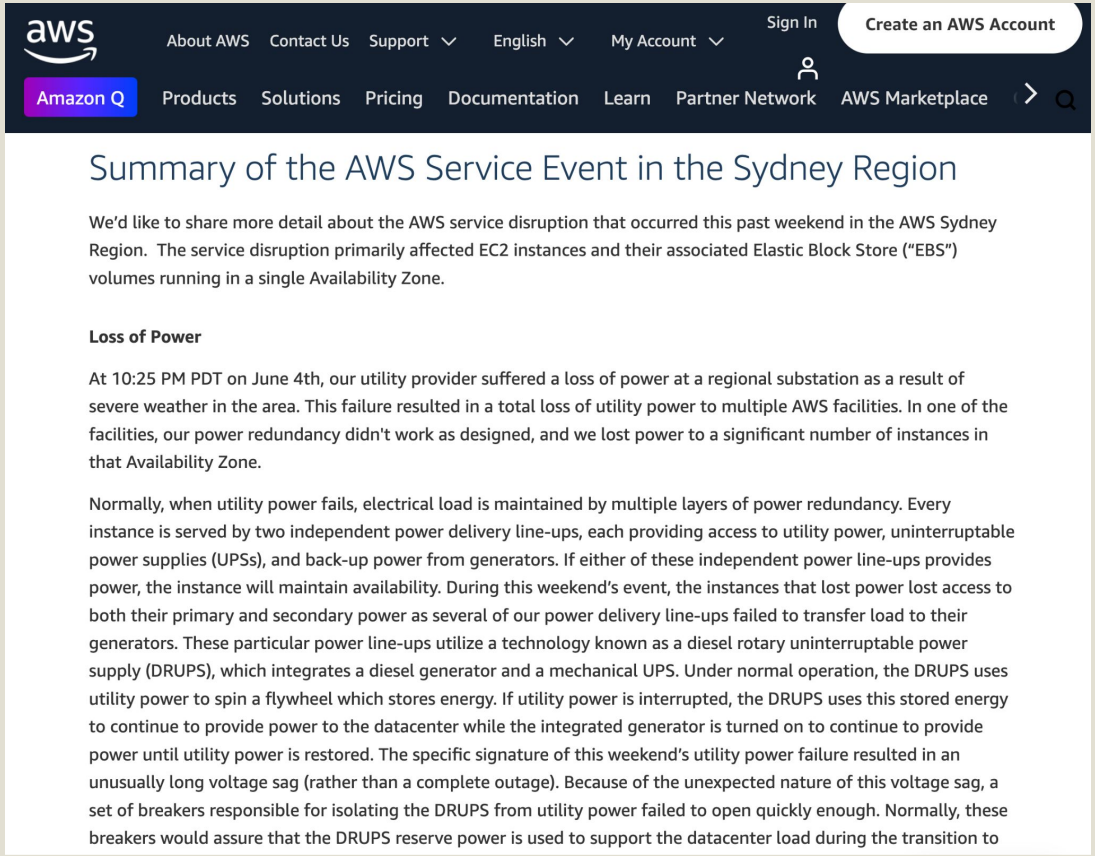
Take a neutral stance and highlight options that are possible without arriving at a conclusion.

Is my audience...

- Technical? — Focus more on technical aspects of the proposals. 70% technical, 30% non-technical.
- Mixed of tech and non-tech? — Share equally between aspects of the proposals. 50% technical, 50% non-technical.

Is my audience...

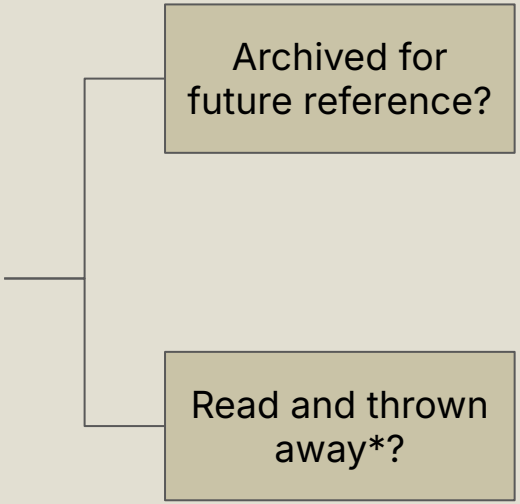
- Part of my organization?
  - Be direct, and include as much details as what your audience might want to know internally
- Public / Users/ Customers?
  - Include details that your users or customers may need to instill confidence without overly sharing too much internal details



AWS Service Event in Sydney

https://aws.amazon.com/message/4372T8/

Will this document be...



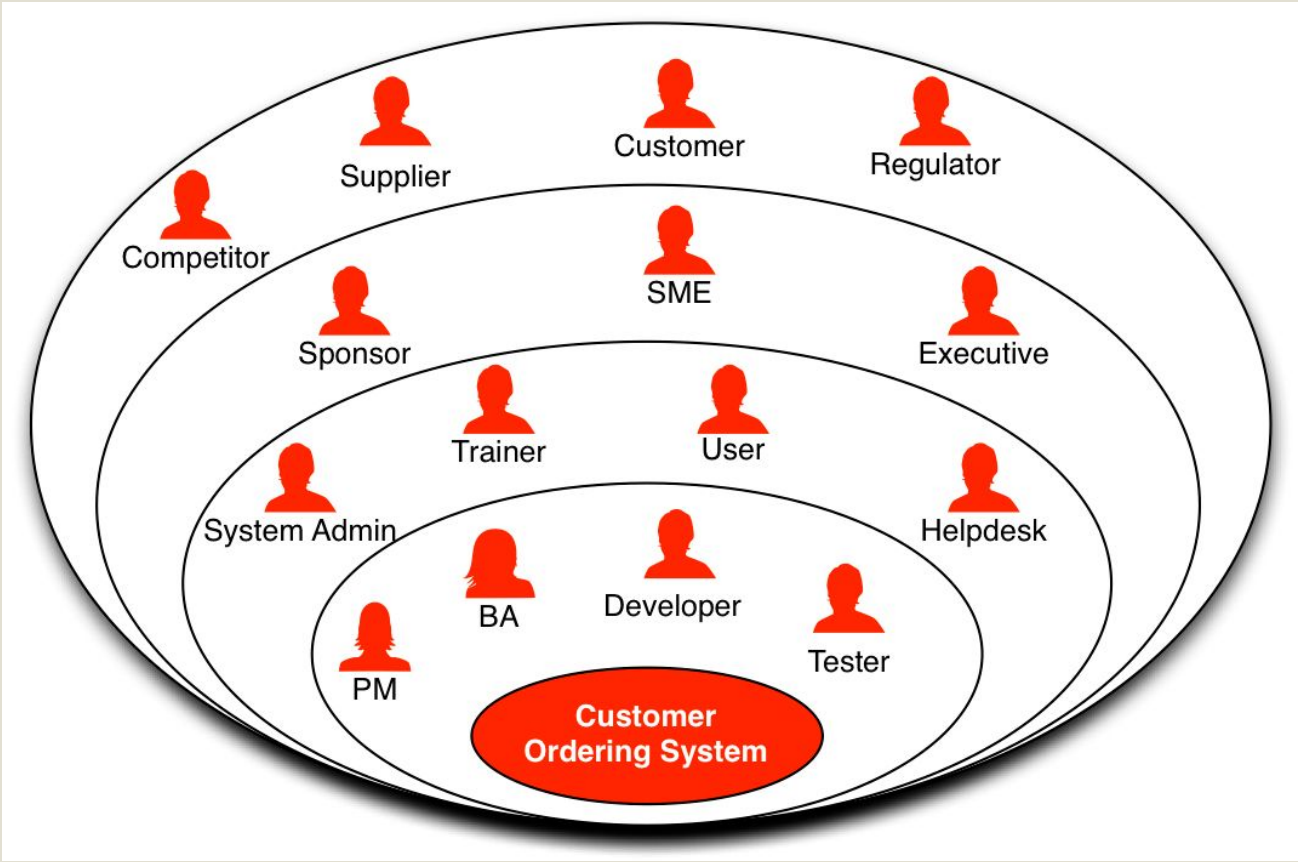
Some companies keep a record of all technical documents, and so ample context should be provided for future readers

Many documents are “throw away” once they have been discussed. In this case, don’t over invest into its context

Exec

Senior

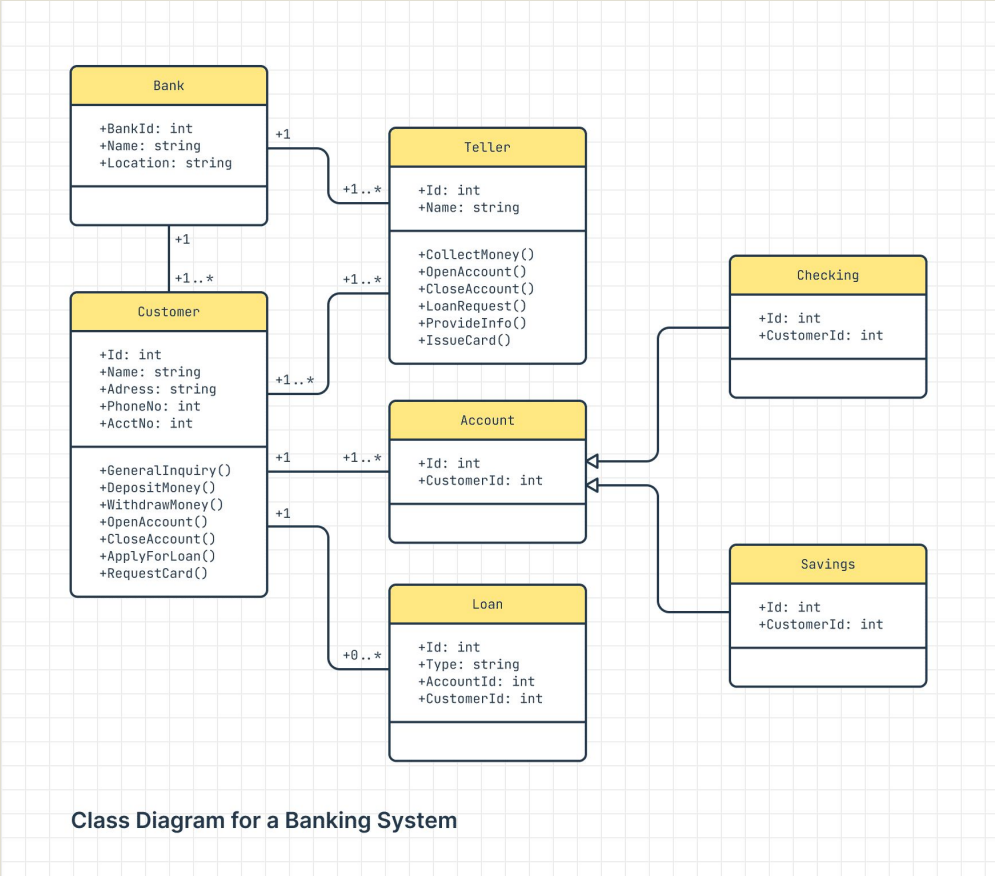
Junior



Stakeholder Onion Diagram

https://www.businessanalystlearnings.com/ba-techniques/2013/1/22/how-to-draw-a-stakeholder-onion-diagram

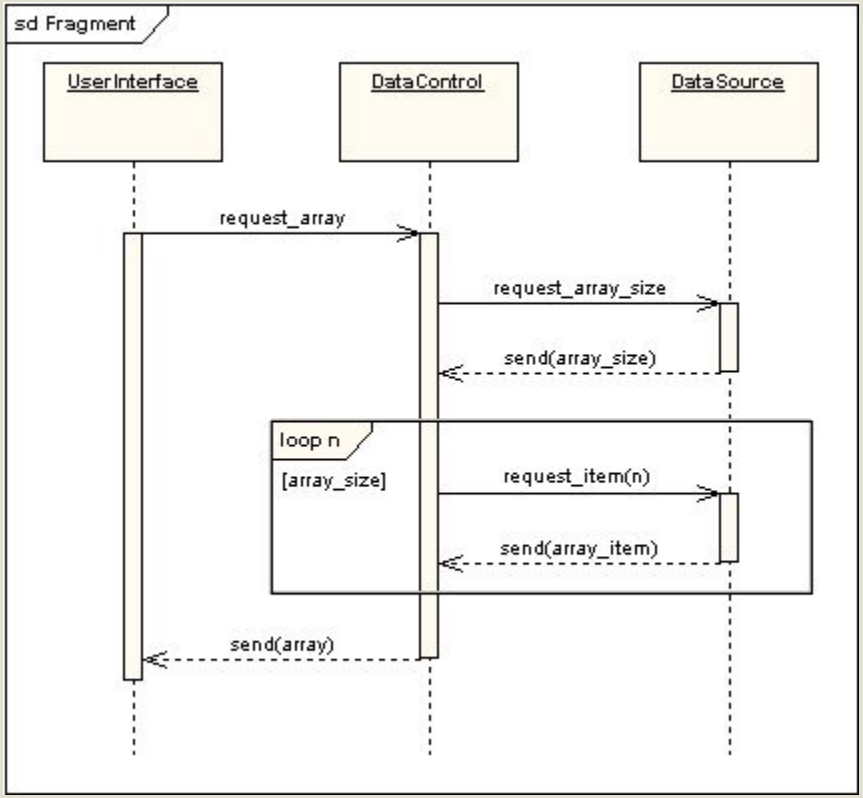

UNIFIED
MODELING
LANGUAGE™
UML®



Class Diagram for a Banking System

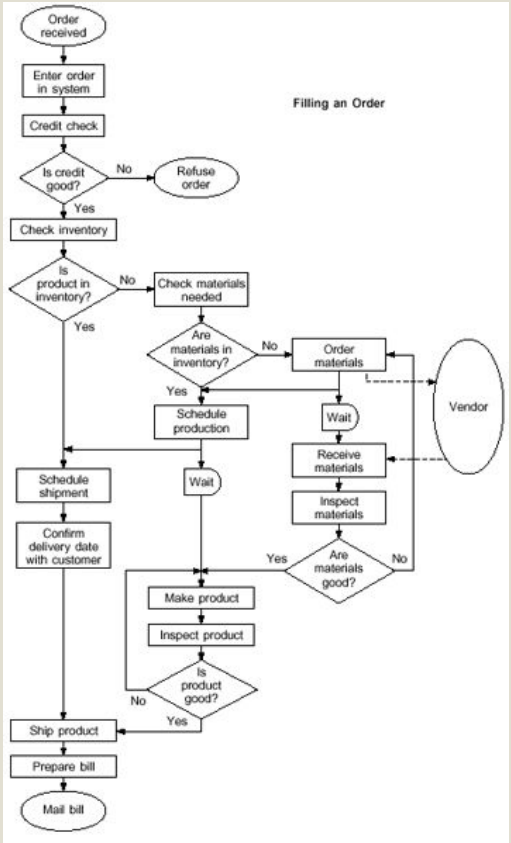
UML Class Diagram

https://slickplan.com/blog/how-to-make-a-uml-diagram



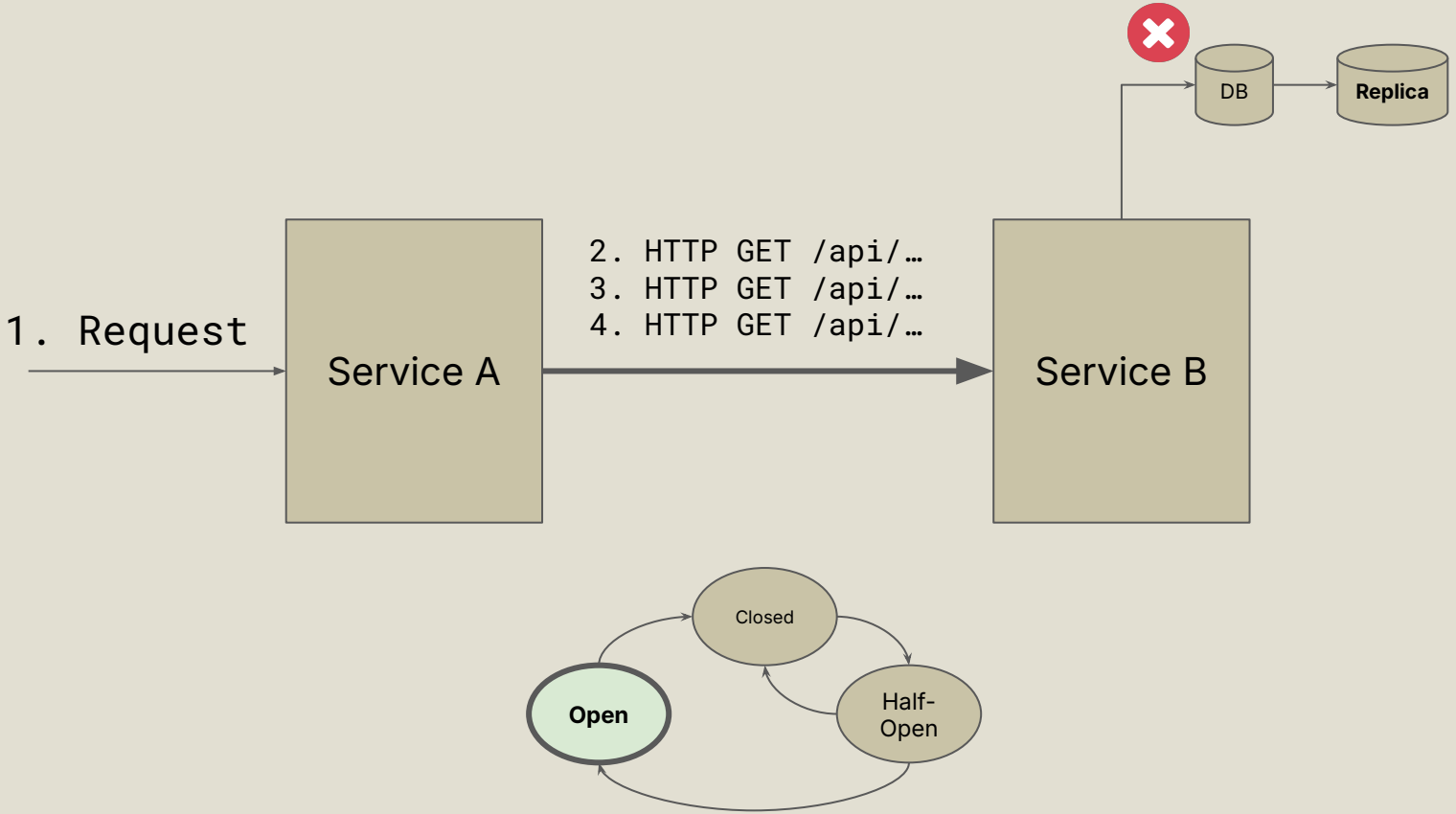
UML Sequence Diagram

https://sparxsystems.com/resources/tutorials/uml2/sequence-diagram.html



Flow Charts

https://asq.org/quality-resources/flowchart



State Diagrams

1. **Writing a technical document in a company for a first time?**

   Ask for feedback from manager / team lead while writing, not after you are done. They are likely to share tips on what the company or stakeholders prioritize.

2. **Proposals are not homework or graded assignments!** They should spark conversations, so solicit feedback, and generate conversations from there.

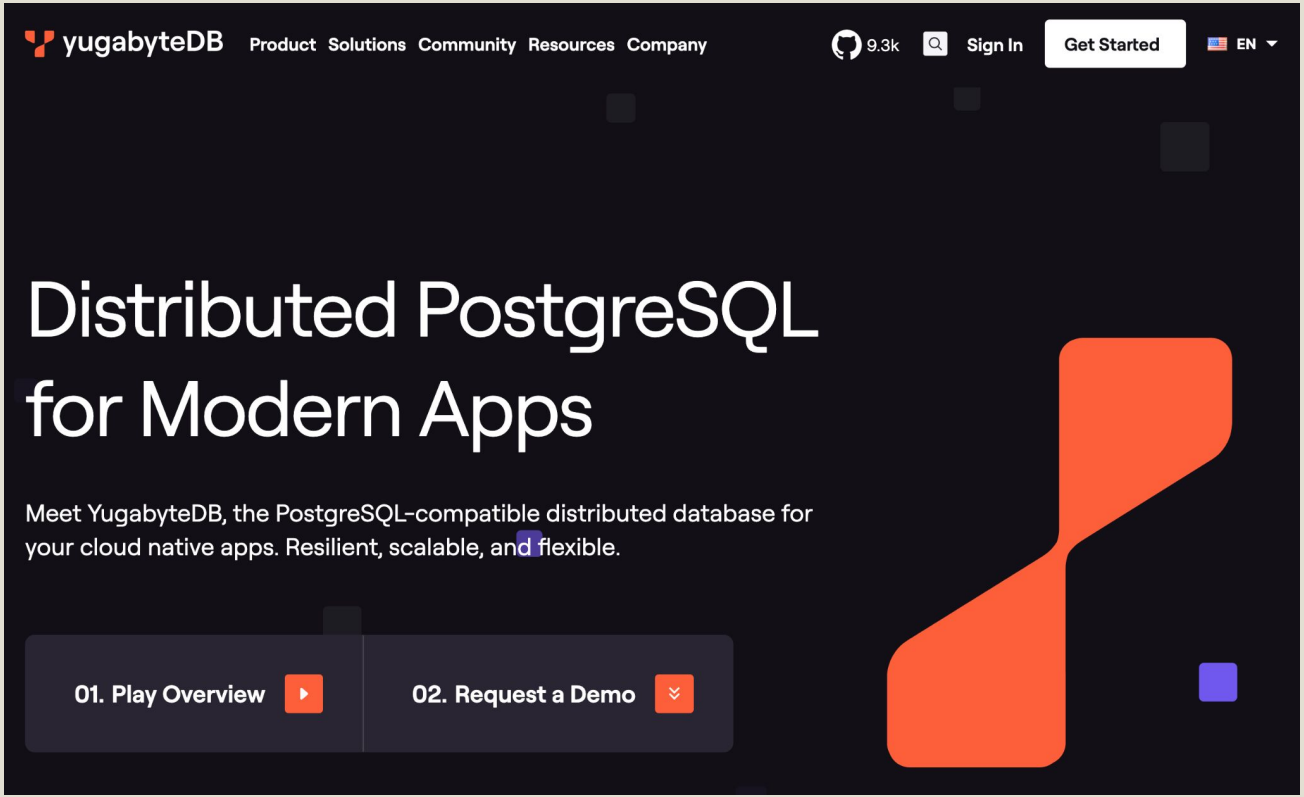
3. Ask for links and references to good existing documents in the company. It’s easier to start from an existing format than to start one from zero.

Other Tips and Tricks

p.s.
Current Trends

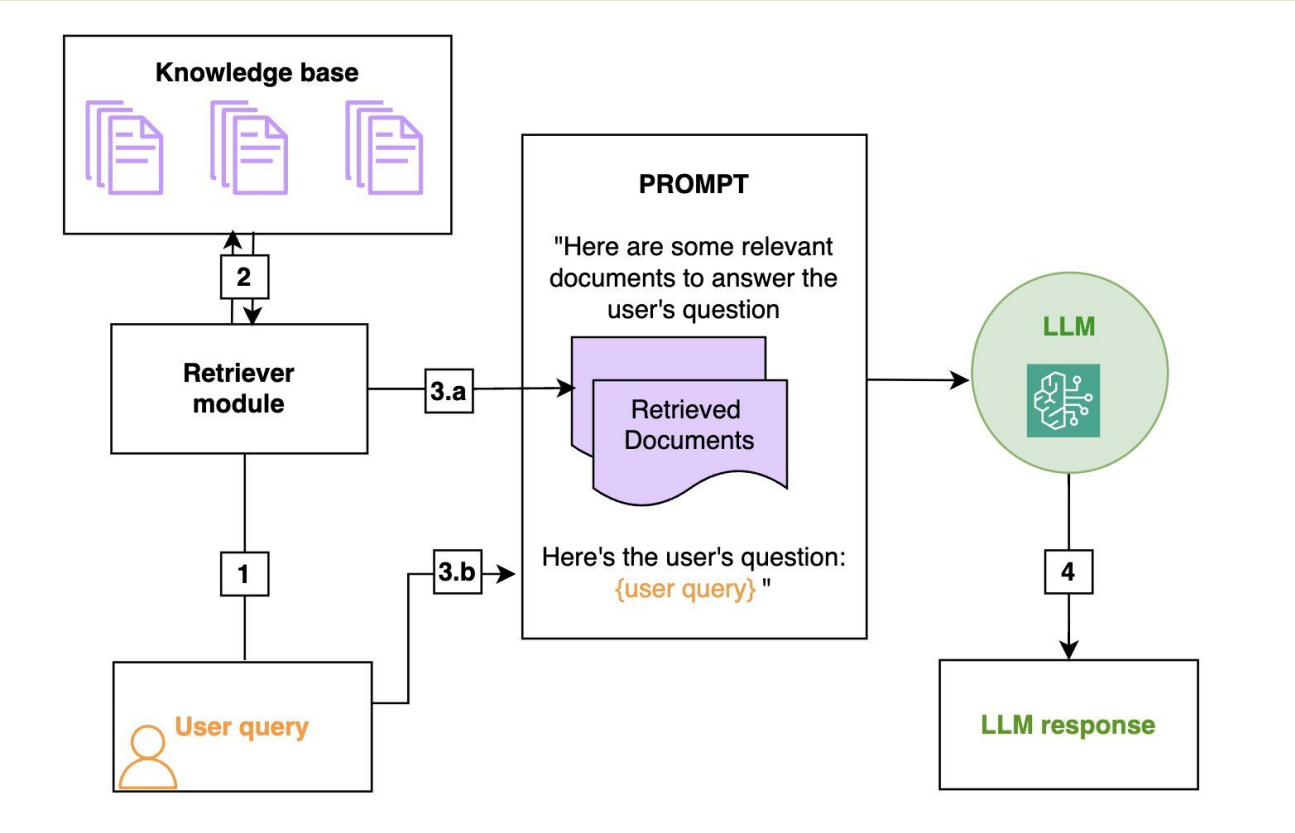
Writing RFCs: Our best practices and their benefits

https://developers.mews.com/writing-rfcs-our-best-practices-and-their-benefits/



YugabyteDB: PG Compatibility

https://www.yugabyte.com/



From RAG to fabric: Lessons learned from building real-world RAGs at GenAIIC

https://aws.amazon.com/blogs/machine-learning/from-rag-to-fabric-lessons-learned-from-building-real-world-rags-at-genaiic-part-1/

optional readings

JEPSEN

Blog | Analyses | Talks | Consistency | Services | Ethics

Analyses

Since 2013, Jepsen has analyzed over two dozen databases, coordination services, and queues—and we've found replica divergence, data loss, stale reads, read skew, lock conflicts, and much more. Here's every analysis we've published.

Aerospike	2015-05-04	3.5.4
	2018-03-07	3.99.0.3
Bufstream	2024-11-12	0.1.0
Cassandra	2013-09-24	2.0.0
Chronos	2015-08-10	2.4.0
CockroachDB	2017-02-16	beta-20160829
Crate	2016-06-28	0.54.9

Jepsen Analyses

https://jepsen.io/analyses

by **Gergely Orosz**

Home
Newsletter
Popular Articles
The Software Engineer's Guidebook
My Books
Early trends

Companies Using RFCs or Design Docs and Examples of These

RFCs – requests for comment – or Design Docs are a common tool that engineering teams use to build software faster, by clarifying assumptions and circulating plans earlier. There are some similarities between writing automated tests for your code, and writing RFCs before you start working on a non-trivial project:

Companies Using RFCs or Design Docs and Examples of These
https://blog.pragmaticengineer.com/rfcs-and-design-docs/

RFC Examples and Templates

Google

Design Docs overview. Typical document structure:

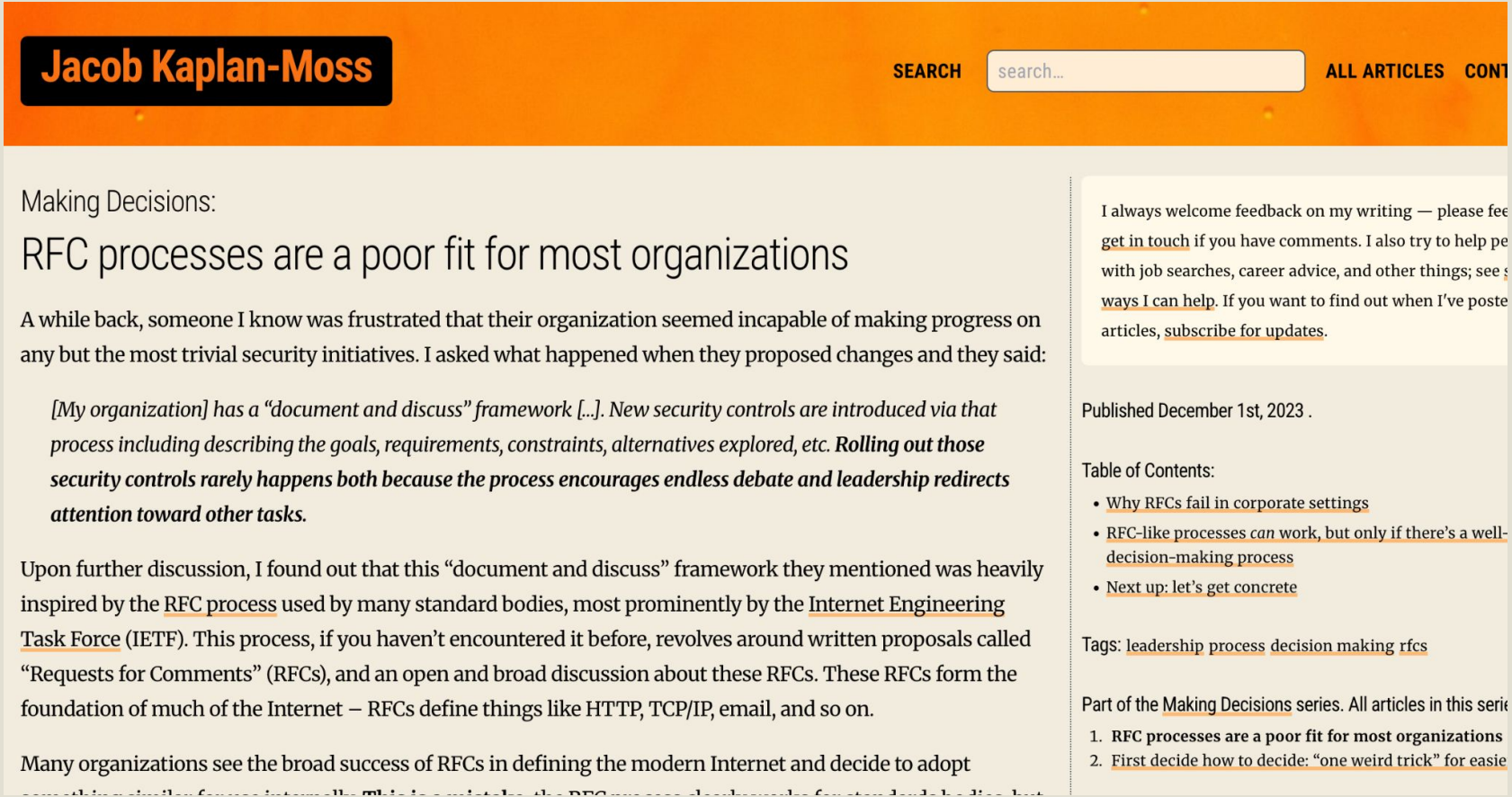
- Context and scope
- Goals and non-goals
- The actual design
- System-context diagram
- APIs
- Data storage
- Code and pseudo-code
- Degree of constraint
- Alternatives considered
- Cross-cutting concerns

Uber

RFC process overview (used up to around 2019).

Typical structure for services:

- List of approvers
- Abstract (what is the project about?)
- Architecture changes
- Service SLAs
- Service dependencies
- Load & performance testing
- Multi-data-center concerns

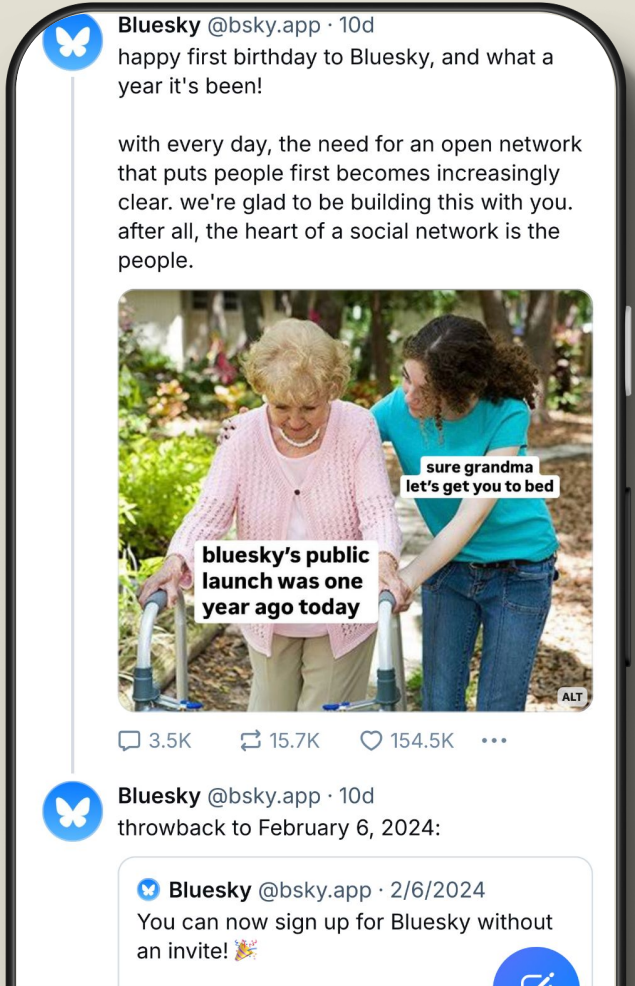


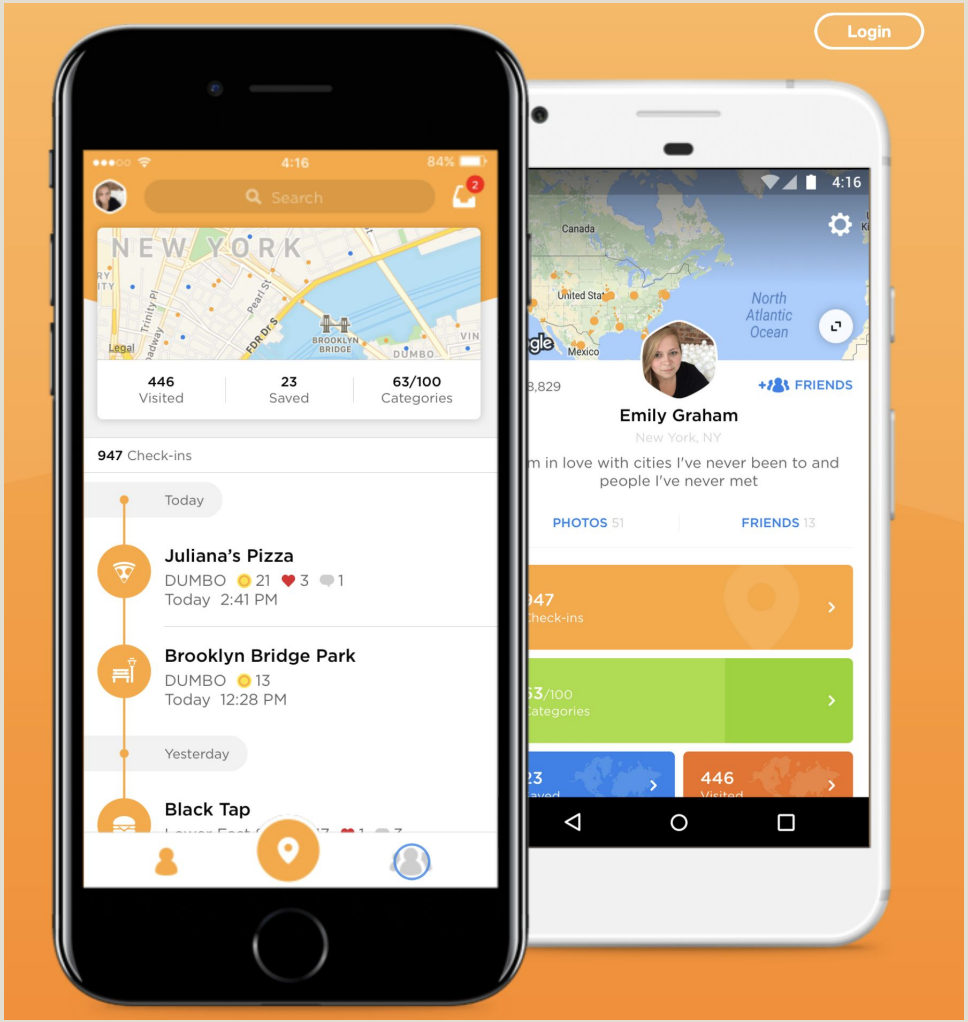
RFC processes are a poor fit for most organizations

https://jacobian.org/2023/dec/1/against-rfcs/

Assignment

Extend a Distributed Social Media Platform




Login
4:16
84%
Search
NEW YORK
446
Visited
23
Saved
63/100
Categories
947 Check-ins
Today
Juliana's Pizza
DUMBO 21 3 1
Today 2:41 PM
Brooklyn Bridge Park
DUMBO 13
Today 12:28 PM
Yesterday
Black Tap
4:16
Canada
North Atlantic Ocean
Mexico
FRIENDS
Emily Graham
New York, NY
PHOTOS 51
FRIENDS 13
446

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.

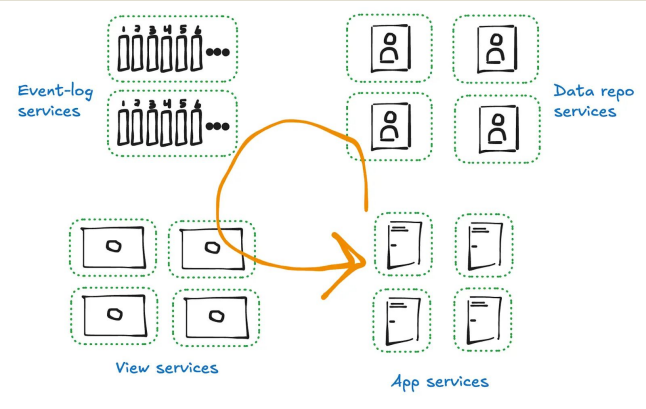
- For the PoC, just focus on creating a Post first; avoid doing any form of systems design until you have created a Post

Assignment

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

Bluesky: AT Protocol

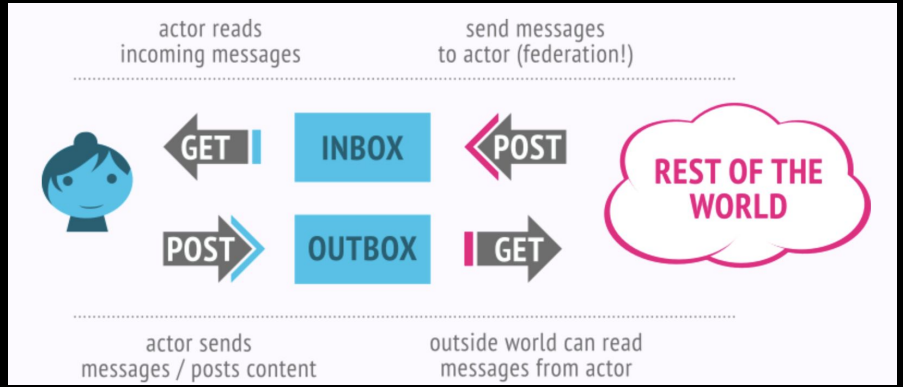
"The AT Protocol is an open, decentralized network for building social applications."



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 #3

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

Start designing your system:

1. How would you handle creating new Check-ins and a timeline?

2. Will you store GPS coordinates as meta-data, or do your chosen protocol support other ways to do so?

3. What database will you use? *My suggestion: Try something new!*

Assignment #3

Sharing in W4:

- You sharing should be around 6-10 minutes with simple slides
- Share about your choice of protocol (why it was chosen), and your learnings when building a PoC (eg. surprises, challenges)
- What’s the tech stack you are planning to use to build the system?
- Do you need help with anything? Feel free to call out.