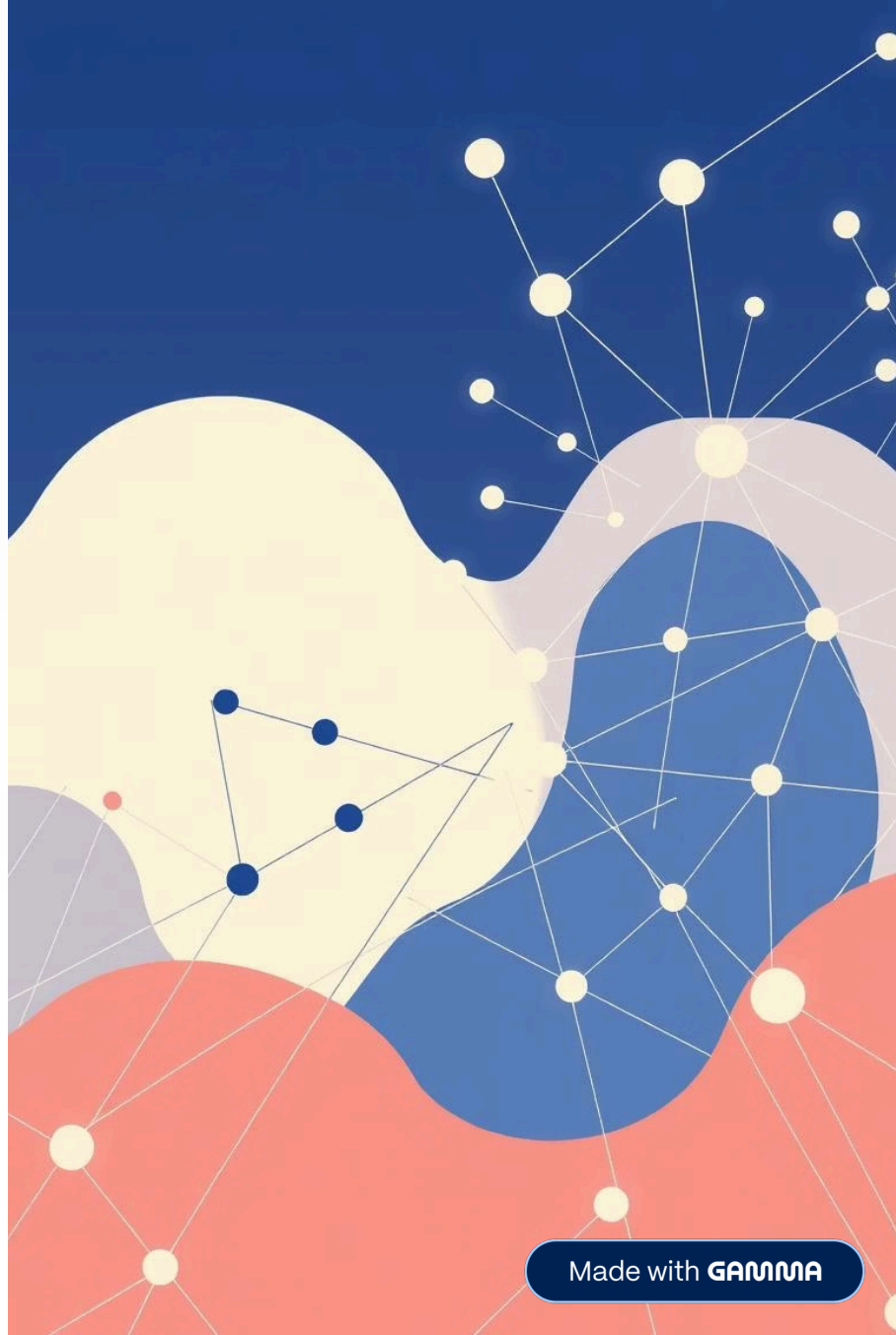
Distributed Locking Explained

A Practical Guide for System Architects and Backend Engineers



Once Upon a Time... The Ancient Ways

We have a simple traditional voting system:

One Room

A cozy, private space.

One Voter

Peaceful, undisturbed.

One Lock (the door!)

Perfect correctness,
terrible speed.

Version 1: The Local App

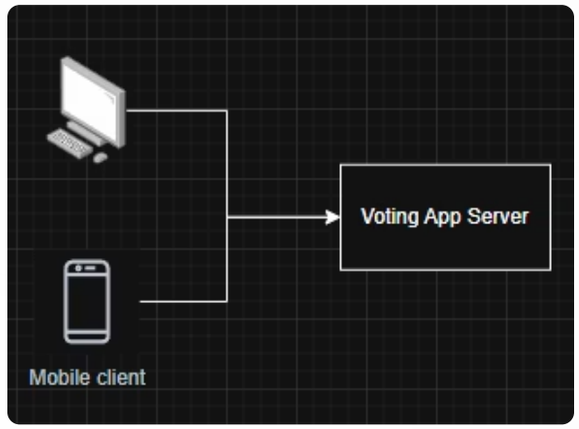
Our first foray into digital democracy:

A **single mobile app** to pass around.

Voters queued for hours...

Version 2: The Shiny Web App + One Big Server

Everyone votes from their phone! So modern, so convenient! ✨

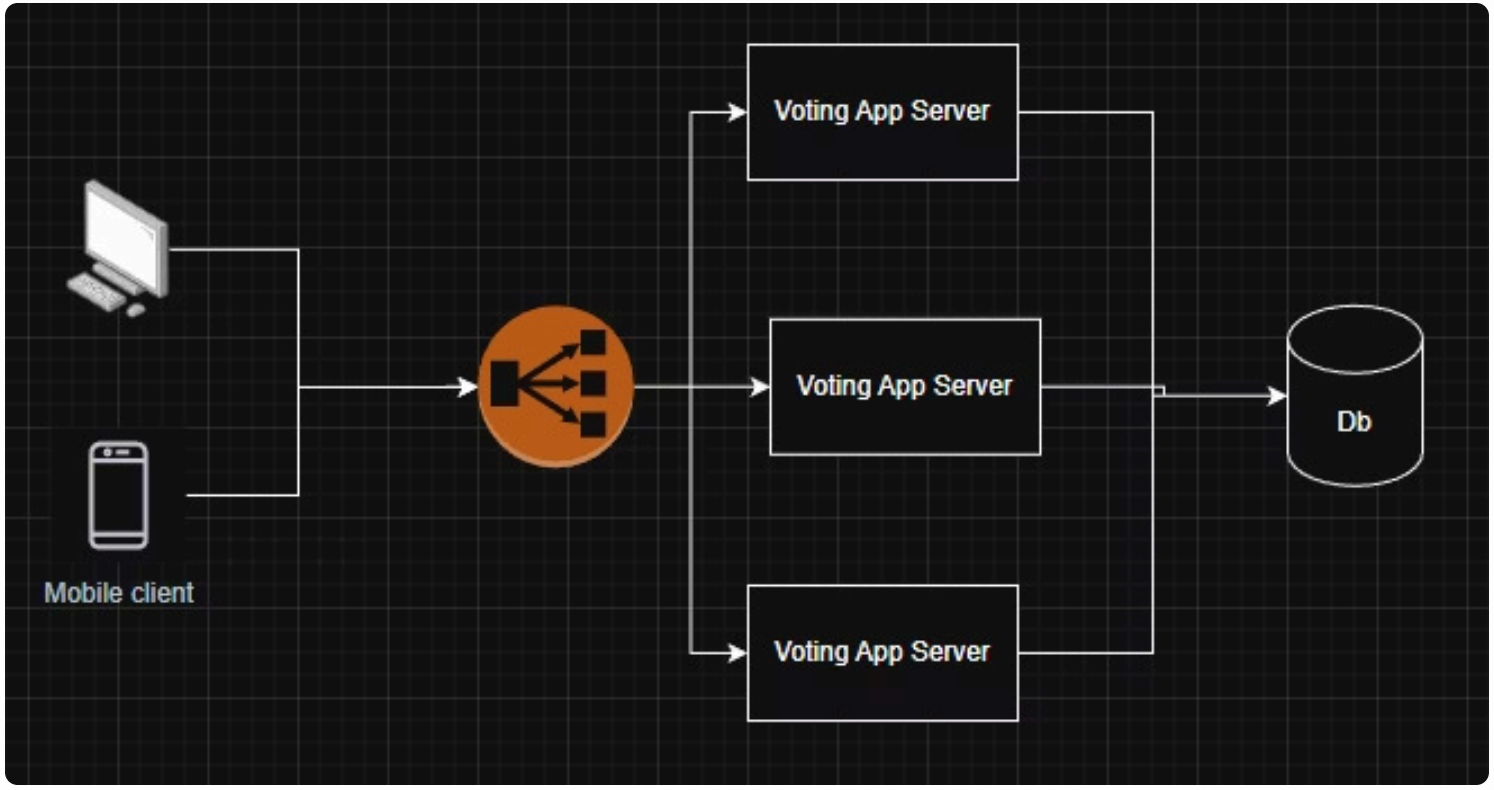


But then... the race condition reared its ugly head.

> **Read vote count (5)**
>
> **Read vote count (5)**
>
> **Write new count (6)**
>
> **Write new count (6)**
>
> One vote vanished! 👻

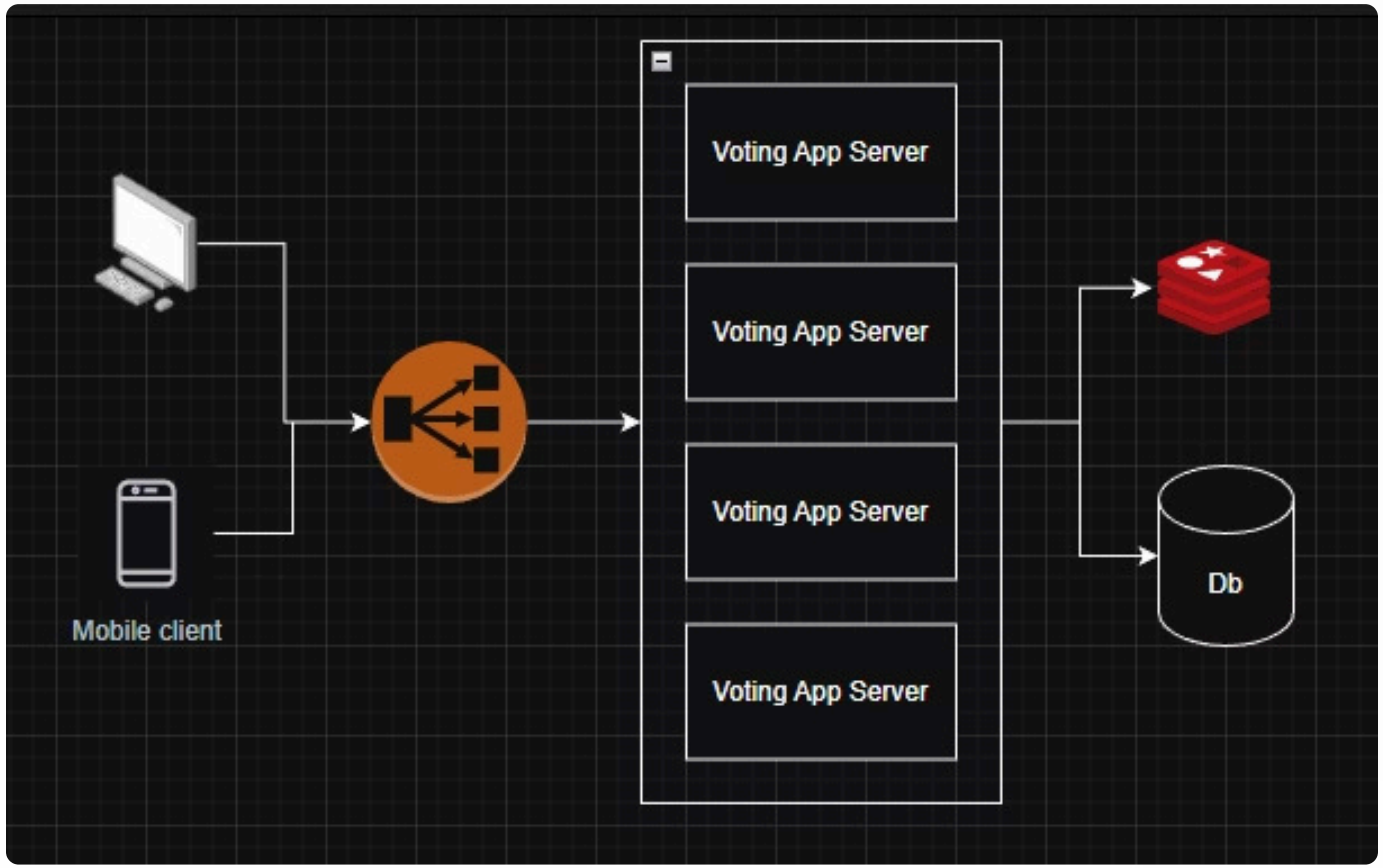
Version 3: Horizontal Scaling + Database

More servers, more power! We scaled up to:



But a new nightmare emerged: **two different servers updating the same row at once!**

Redis Distributed Lock



When vote, create a lock key => Remove when done/fail

Problem 1: Server down

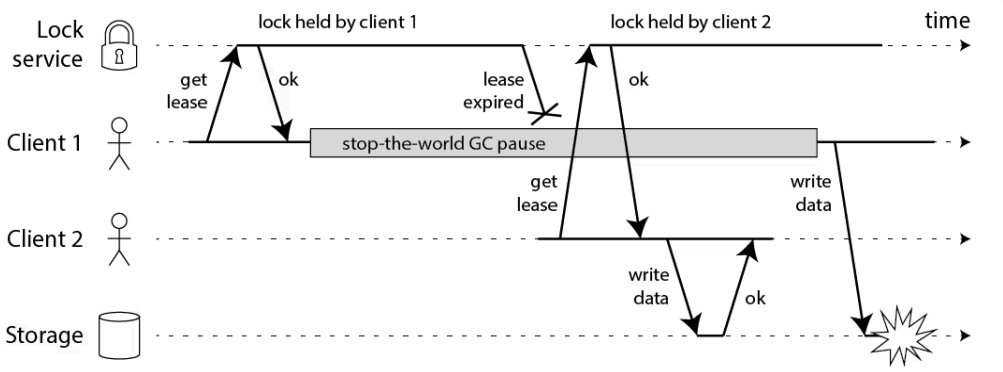
Server down => Lock forever

=> Solution: set TTL … but how long?

Problem 2: Process pause

- GC run => Process pause
- Delete another client's lock

=> Solution: Set random UUID as value to identify.



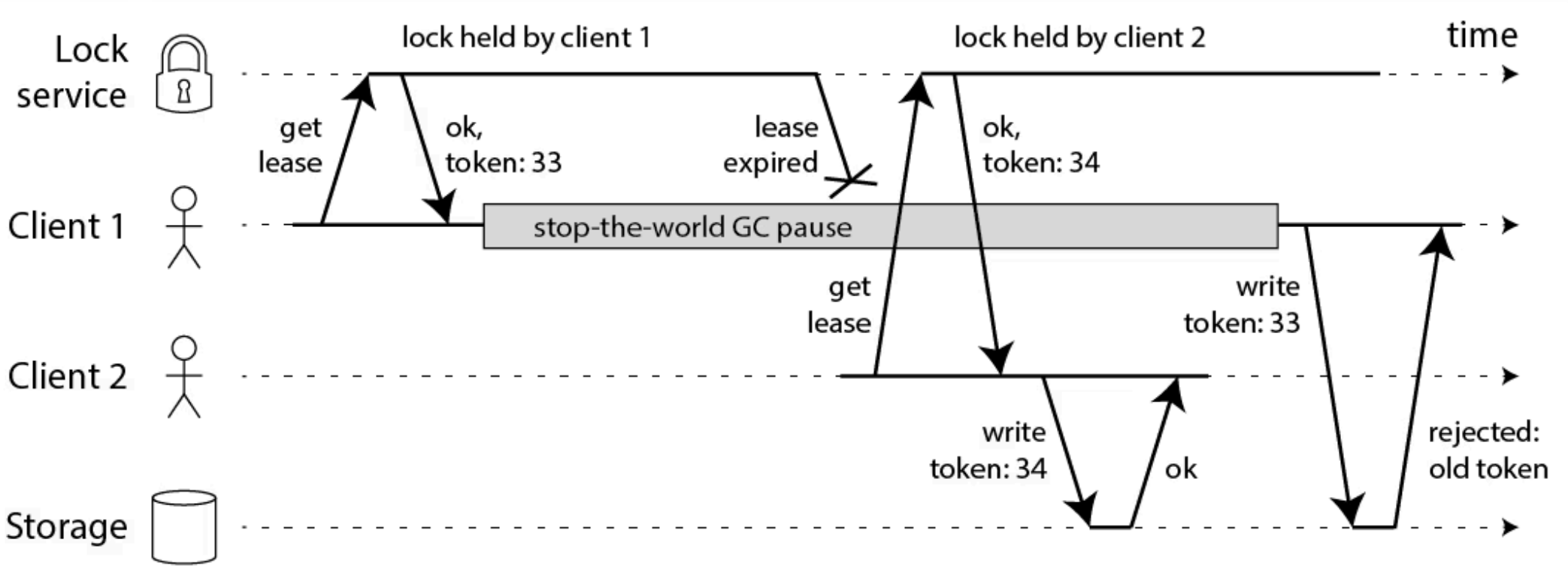
Problem 2: Process pause

- GC run => Process pause

=> Solution:

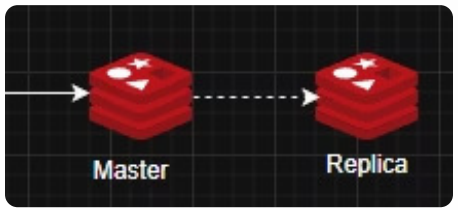
- Timestamp
- Fencing token

Fencing token

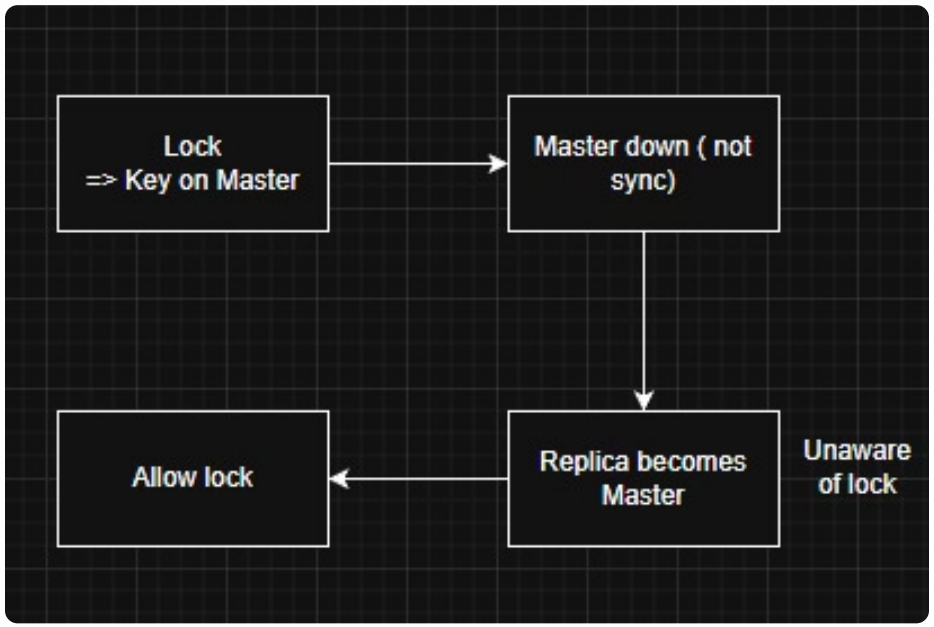


Problem 3: Redis fail

Async Replication ?



Two instance could obtain the same lock:

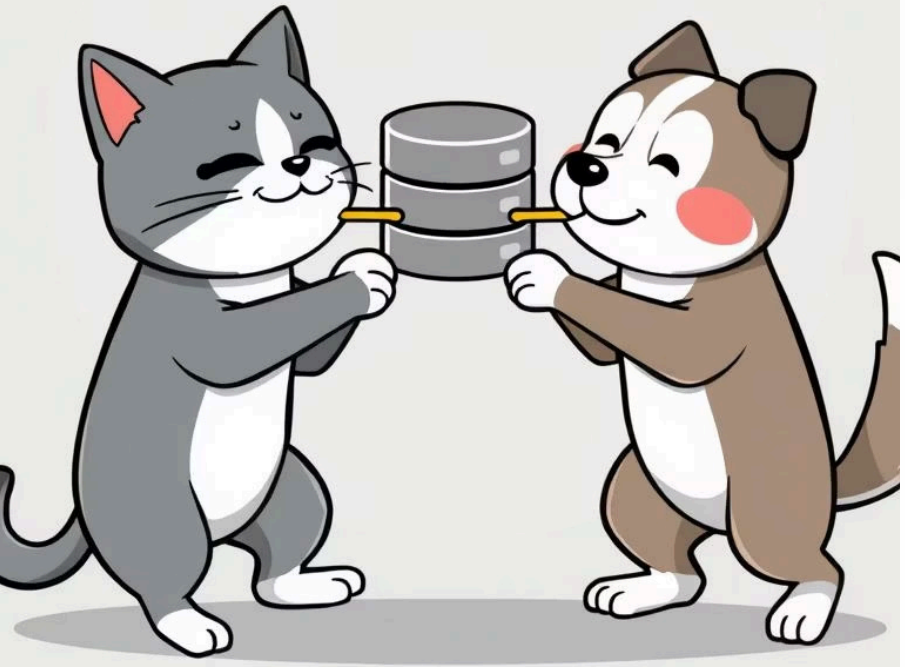


Database Pessimistic Locking

Our brilliant solution: `SELECT ... FOR UPDATE`

Correctness? ✅ Absolutely!

But then, the performance hit and **deadlock dance** began.



Database Lock Table

A dedicated lock table + `INSERT ON CONFLICT`. Works perfectly when you only have 2 parties contending for 2 hot rows forever!

```
CREATE TABLE distributed_locks (
    resource VARCHAR(255) NOT NULL PRIMARY KEY,
    token    VARCHAR(36)  NOT NULL,
    expires_at TIMESTAMPTZ NOT NULL
);
```

```
WITH attempt AS (
    INSERT INTO distributed_locks (resource, token, expires_at)
    VALUES ('vote:cat', 'server-123-uuid', now() + interval '30 seconds')
    ON CONFLICT (resource) DO NOTHING
    RETURNING token
)
SELECT token FROM attempt;
```

Apache ZooKeeper 🐘

Our mighty elephant, ZooKeeper (or its cousin etcd), to the rescue!

It uses **ephemeral sequential nodes**:

- Each client creates a unique node.
- Smallest sequence ID gets the lock.
- Node is deleted automatically if client crashes.

Crash = Auto-Release! No more lingering locks!

Choose Your Lock Wisely!

Just like choosing the right party outfit, picking the right lock is crucial!

Single Thread

Works great! ... but doesn't scale for a party this big.

DB Locks

Correct, but too slow for peak vote season.

Redis Locks

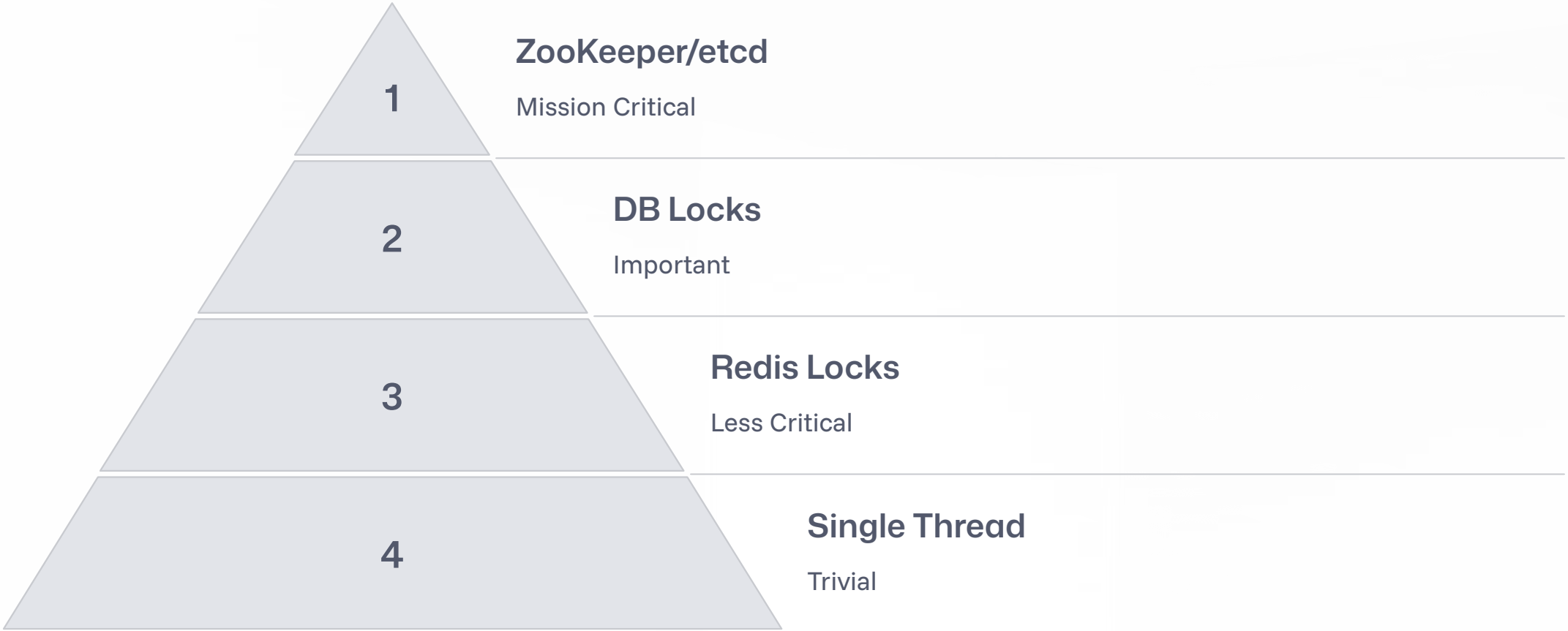
Fast! ... but can lose correctness.

ZooKeeper/etcd

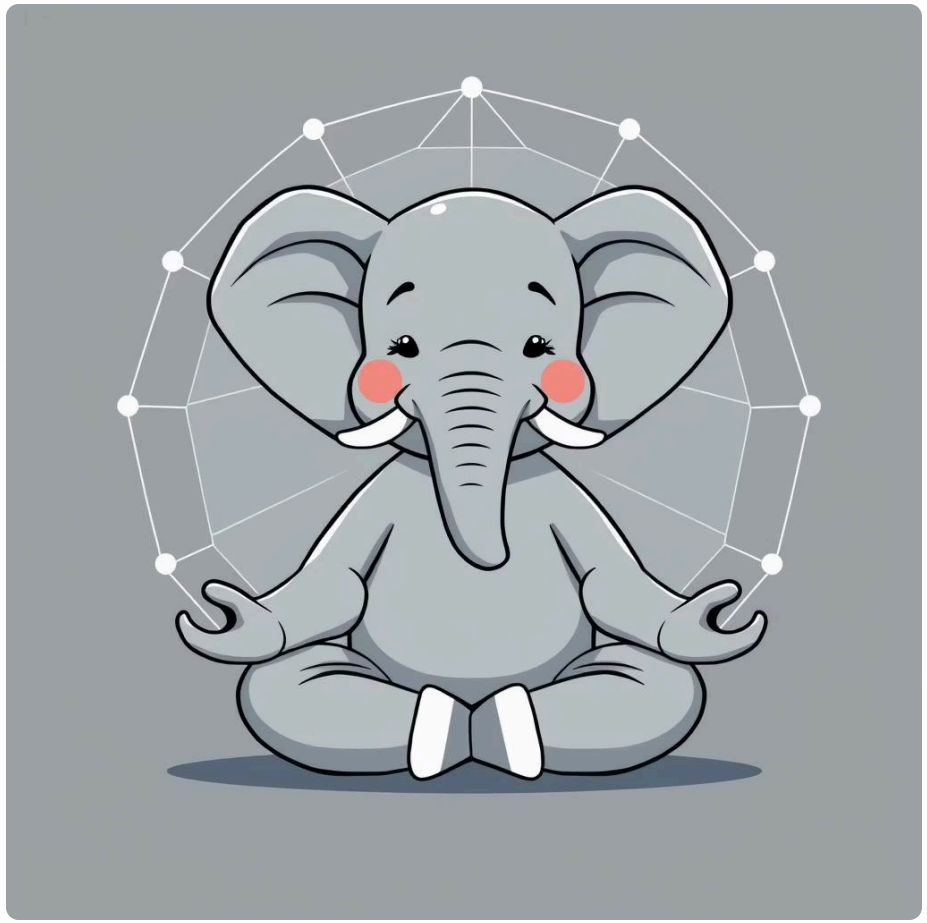
Bullet-proof (but heavier). The ultimate party bouncer.

The Right Lock for Your Election's Importance 🗳️

How important is your distributed election? Choose a lock that matches the gravity of your feline or canine conflict.



Thank You! Questions?



Stay calm and collected, just like a ZooKeeper user!