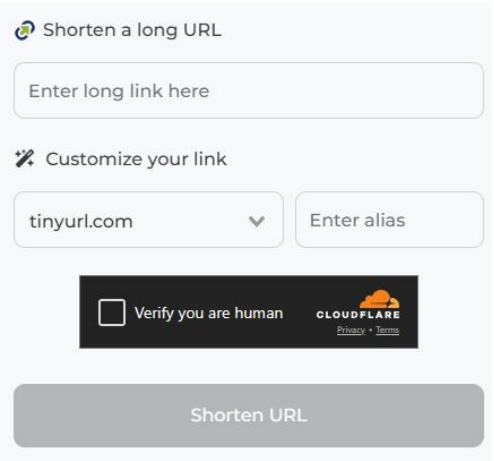
TinyURL/Bitly

Functional Requirements

CreateShortURL(String longURL)

- Returns String shortURL*



FetchShortURL(String shortURL)

- Redirects user to corresponding longURL

*Short URL is in the format tinyurl.com/xxxxxxx

**We'll be leaving "PasteBin" and click counts for subsequent videos

Supported Scale

URL Creations: 600 million per month = 228 per second

URL Retrievals: 10 billion per month = 3805 per second

Short URL Format

How many characters should we use for our URL suffix?

600 million creations/mo = 7.2 billion creations/y = 720 billion creations/century

- If each URL can contain a-z and 0-9, we have 36 choices per character
- If using 8 characters, we have 36^8 = 3 trillion possibilities

URL Mapping Table

Short URL Suffix	Long URL	CreatorID	Expiry Time
hlyu76tt	JordansOnlyFans.com	23	Jan 1 2026, 00:00:00
09iomn43	JordanIsSmelly.com	42	Jan 1 2025 00:00:00
45647hvc			Jan 1 1970, 00:00:00

- 8 bytes for suffix
- 50 bytes (on average) for long URL
- 8 bytes for CreatorID
- 8 bytes for expiry time

74 bytes per row!

Number Of Database Nodes

3805 database reads per second

- Simple reads, can probably be handled on a single node

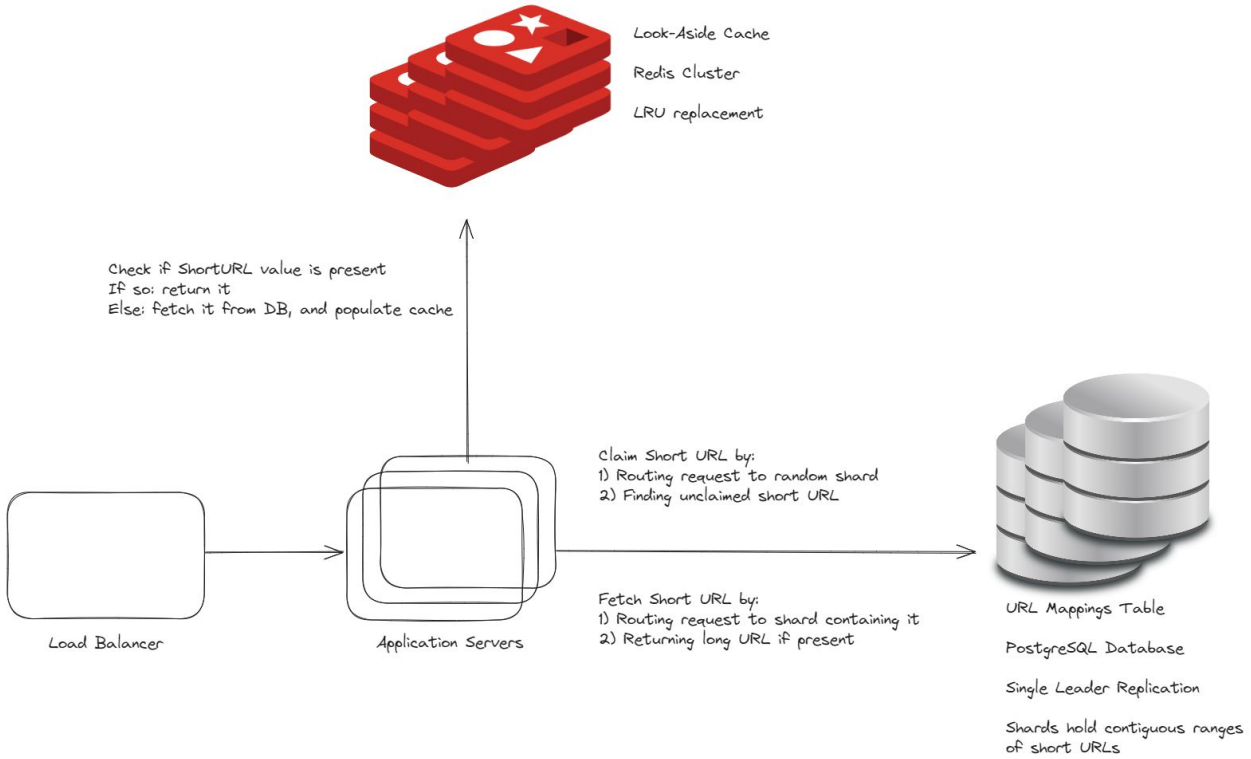
228 writes per second

- Can most likely be handled on a single node, depending on write complexity

74 bytes * 720 billion = 48TB of data

- Can be handled on a beefy node, but likely worth horizontally scaling

High Level Design



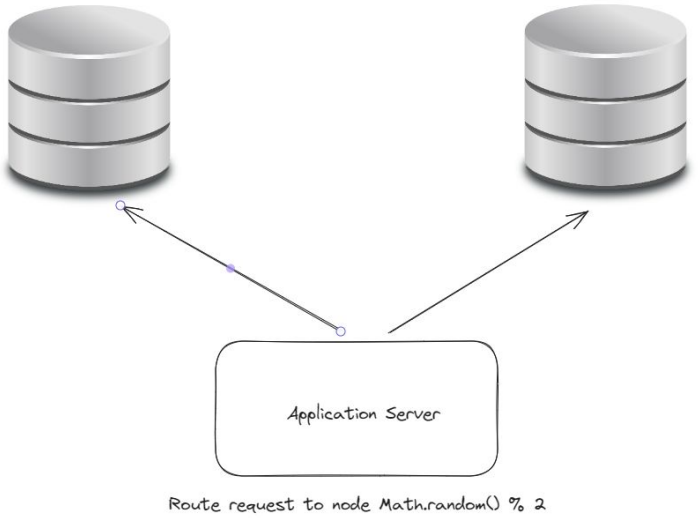
Key Generation Service Overview

shortURL	long URL	clientId	expiration
0000000			0
0000001			0
0000002			0
.			.
.			.
.			.
hzzzzzzz			0

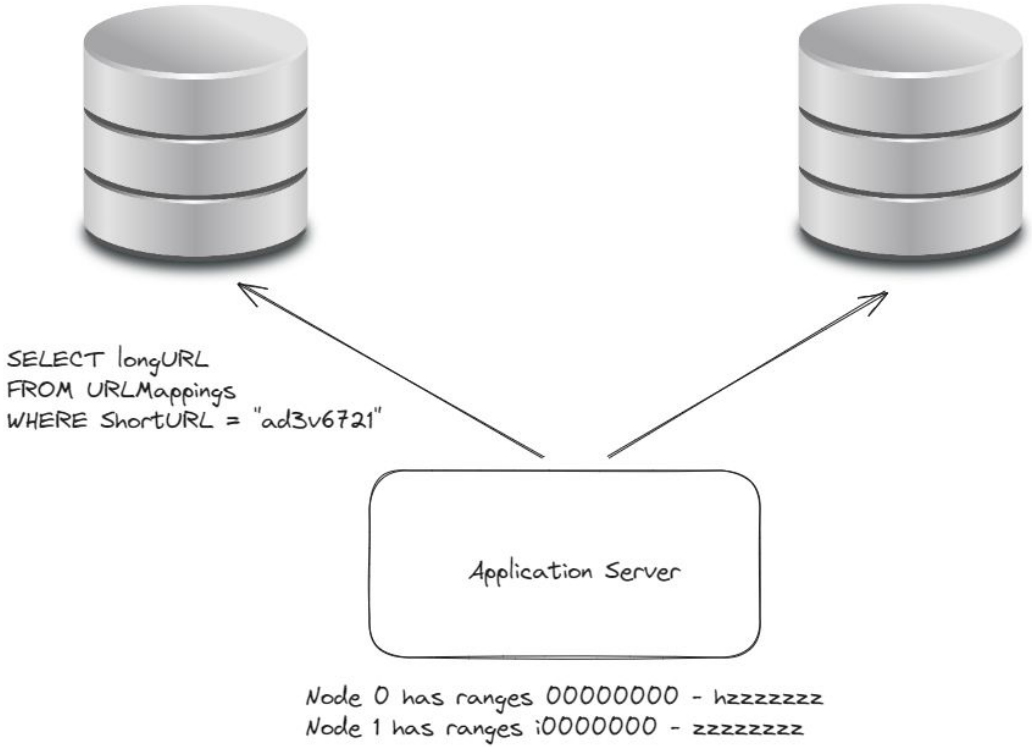
shortURL	long URL	clientId	expiration
i000000			0
i000001			0
i000002			0
.			.
.			.
.			.
zzzzzzzz			0

Creating A Short URL

```
WITH candidate AS (
  SELECT *
  FROM URLMappings
  WHERE expiryTime < NOW()
  LIMIT 1
  FOR UPDATE SKIP LOCKED
)
UPDATE URLMappings
SET expiryTime = NOW() + INTERVAL '1 year',
    userID = 4
    longURL = “toes-are-my-passion.com”
FROM candidate
WHERE URLMappings.id = candidate.id
RETURNING URLMappings.*;
```



Retrieving A Short URL

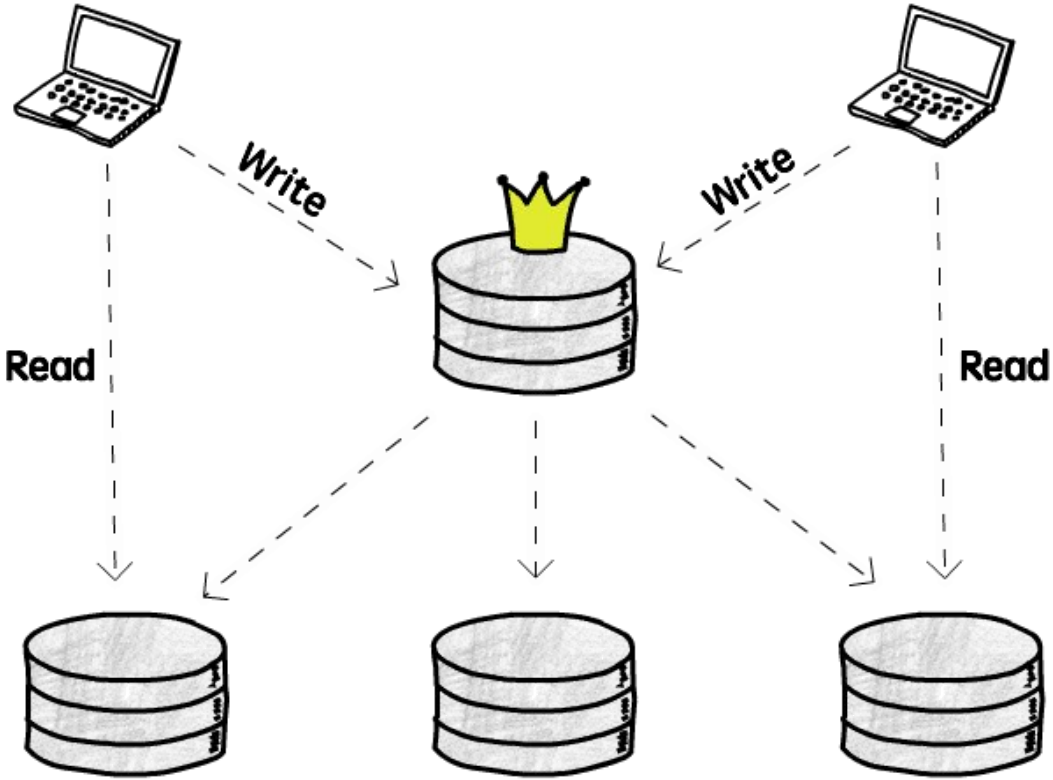


Database Indexing

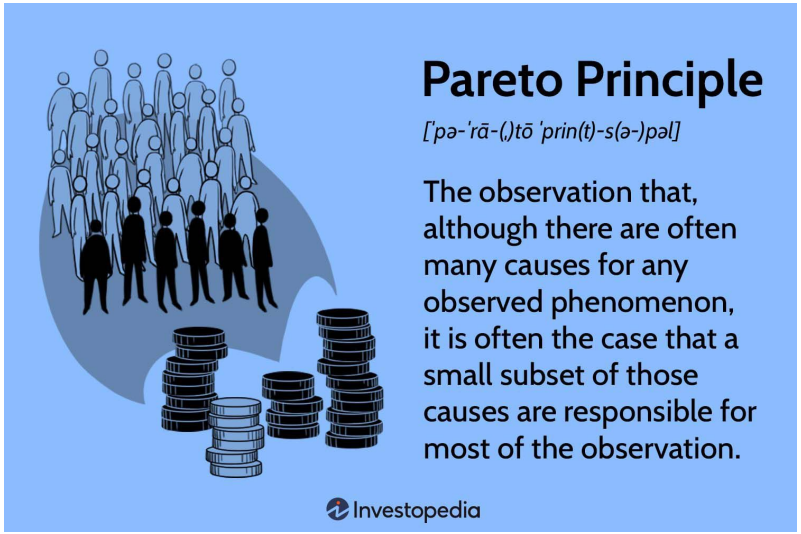
Example Query:

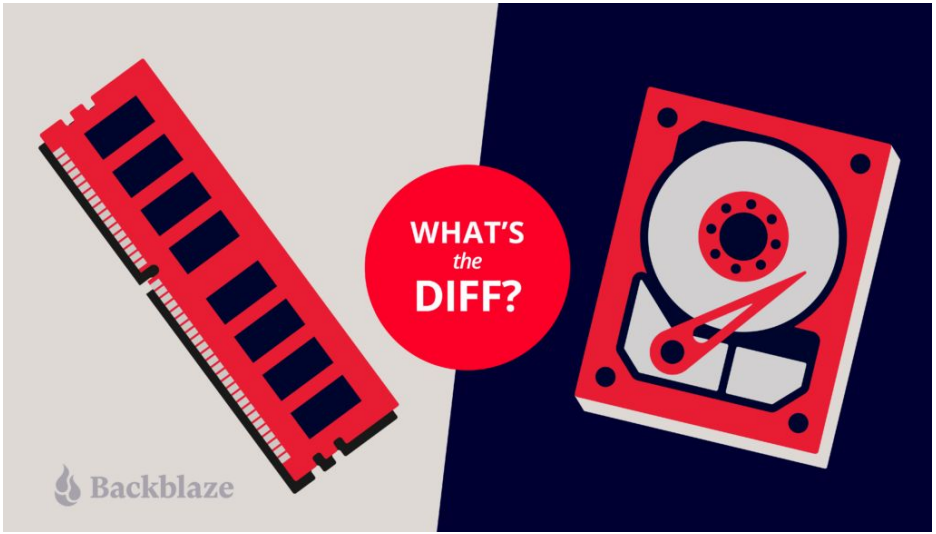
SELECT longURL FROM URLMappings WHERE ShortURL = "ad3v6721"

Database Replication



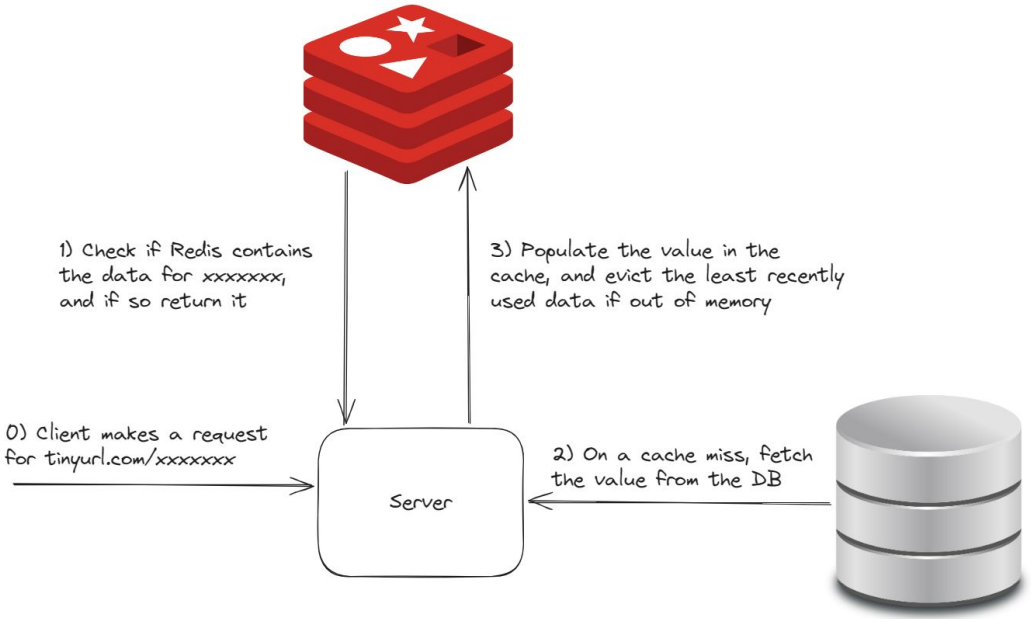
Caching Overview





Redis - Look Aside Caching

- 1% of keys account for 50% of load
- We need 480gb of memory to serve 2000 reads per second



Deep Dives

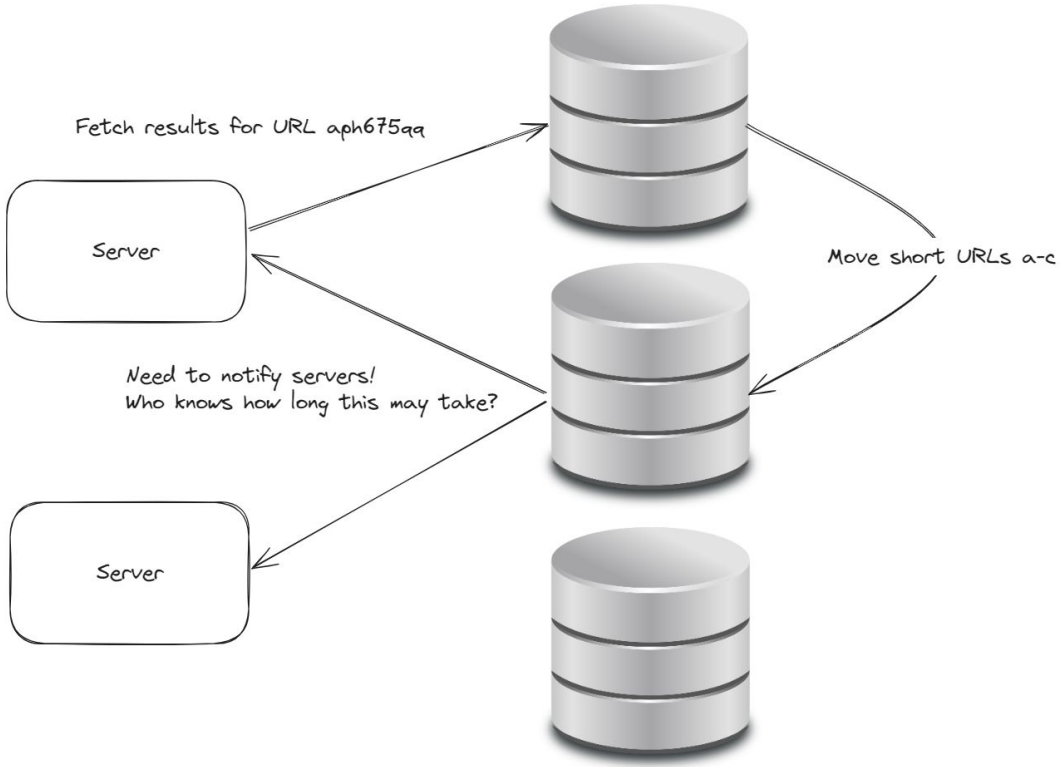
- Database Contention
- Database Sharding
- Citus
- Hashing vs. Key Generation Service
- Choosing A Database
  - Index
  - Replication
- Caching

Deep Dives - Database Contention

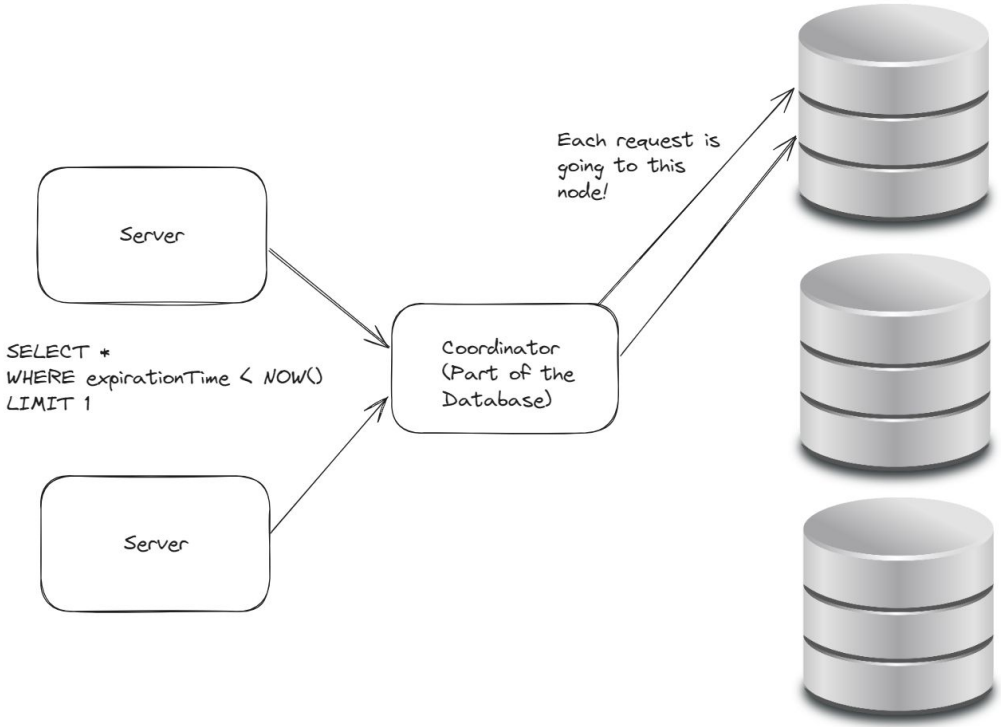
```
WITH candidate AS (
  SELECT *
  FROM URLMappings
  WHERE expiryTime < NOW()
  LIMIT 1
  FOR UPDATE SKIP LOCKED
)
UPDATE URLMappings
SET expiryTime = NOW() + INTERVAL '1 year',
    userID = 4
    longURL = “toes-are-my-passion.com”
FROM candidate
WHERE URLMappings.id = candidate.id
RETURNING URLMappings.*;
```

Key1 - claimed
Key2 - claimed
Key3 - claimed
Key4 - locked
Key5 - unclaimed (don’t block, skip to here!!)
Key6 - unclaimed
Key7 - unclaimed
Key8 - unclaimed
Key9 - unclaimed
Key10 - unclaimed

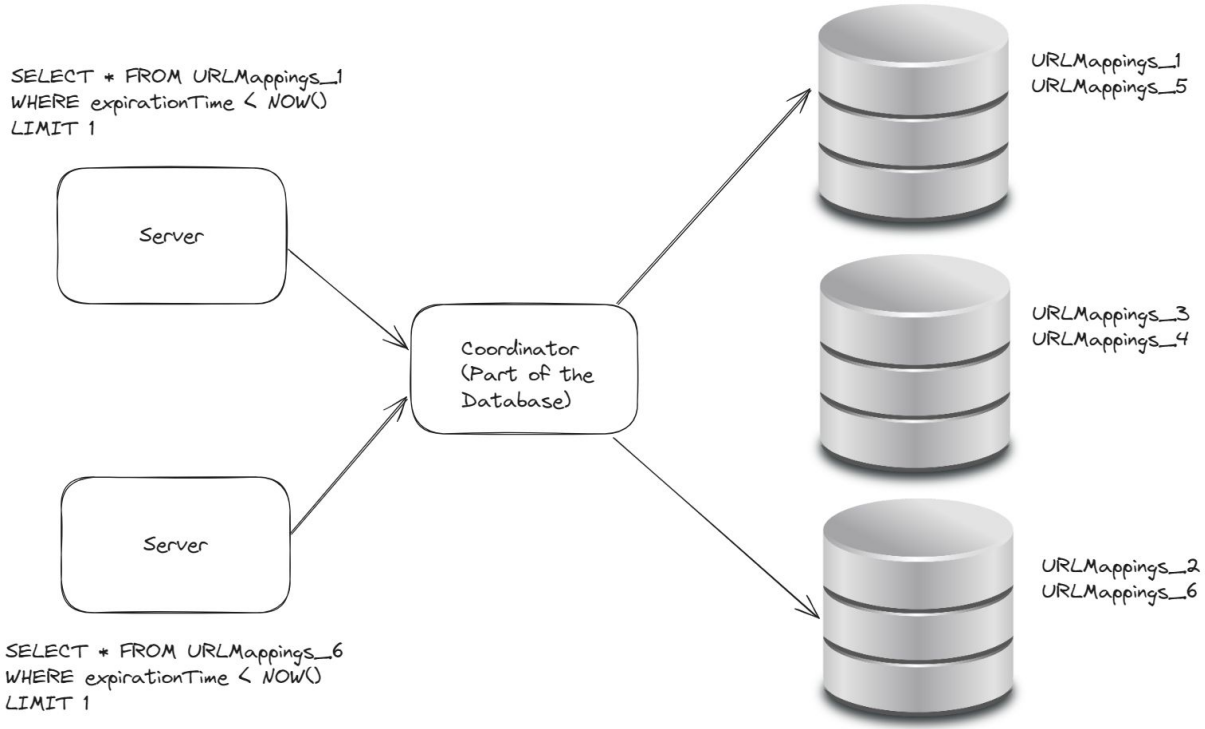
Deep Dives - Key Rebalancing



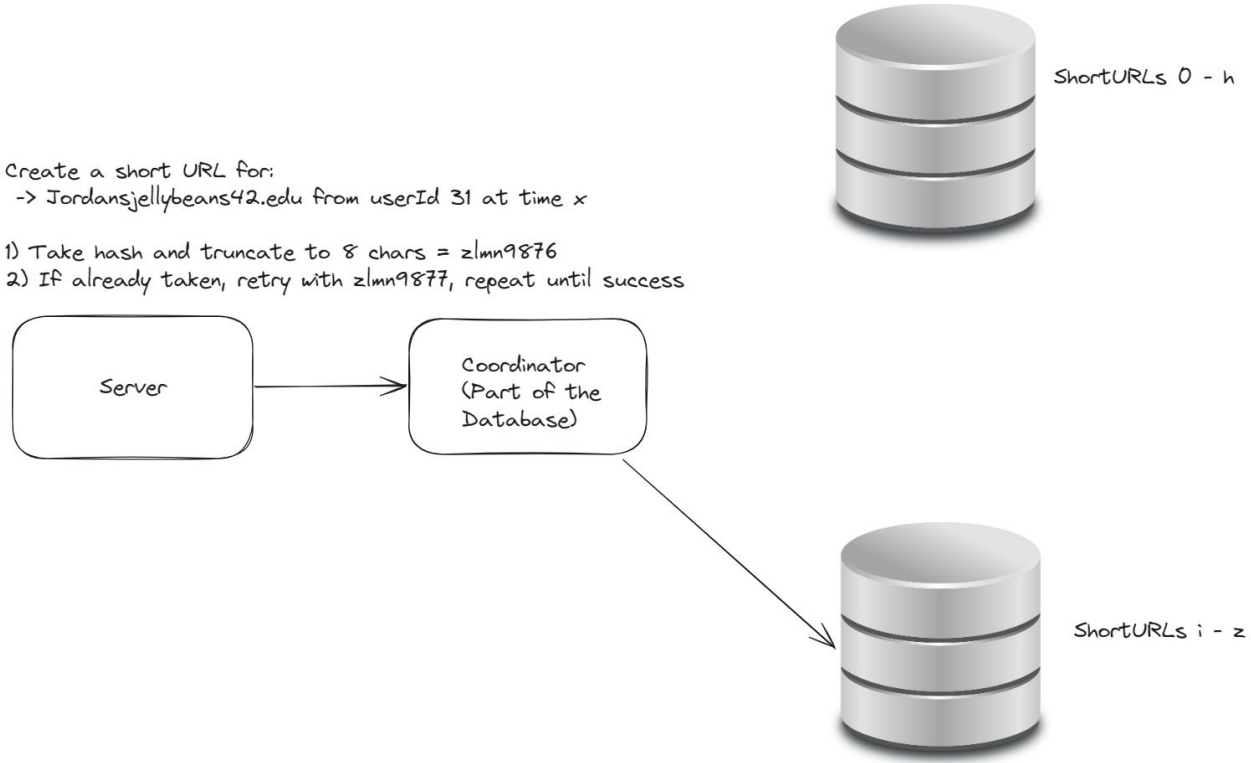
Deep Dives - Database Managed Request Routing



Deep Dives - Citus



Deep Dives - Hashing Vs. Key Generation Service

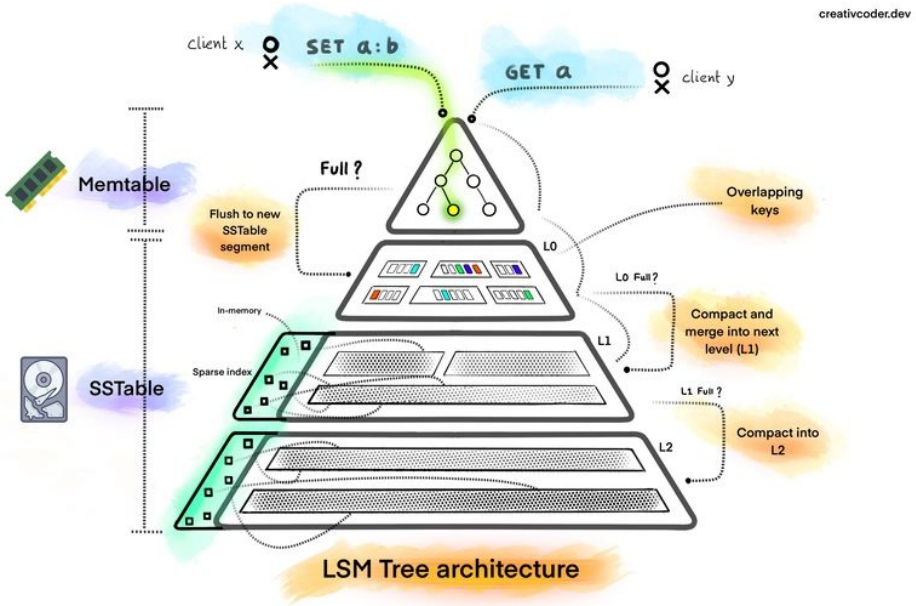


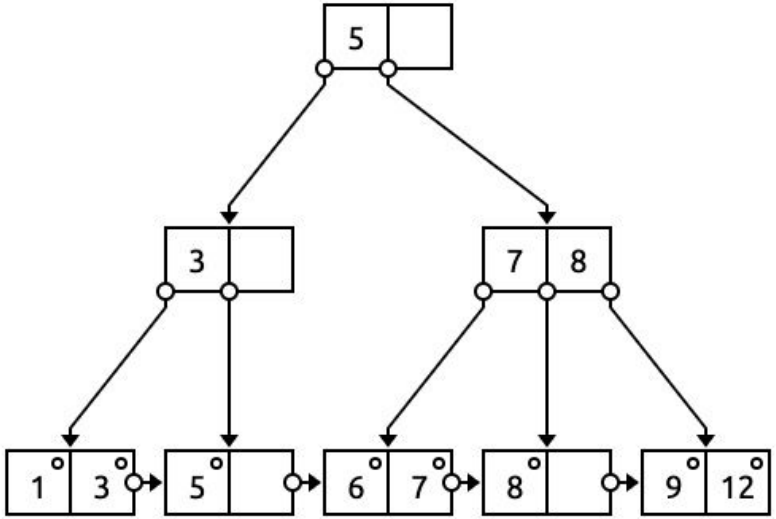
Deep Dives - Choosing A Database

- Index - PostgreSQL uses a B-Tree
- Replication - PostgreSQL uses single leader replication

Maybe sharding techniques as well? Already covered before with Citus

Deep Dives - Database Index

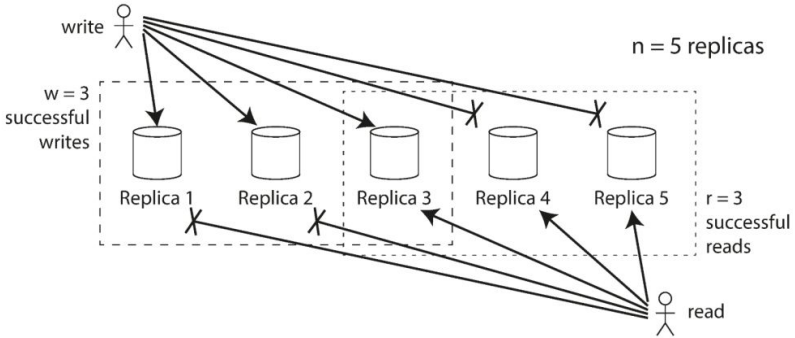




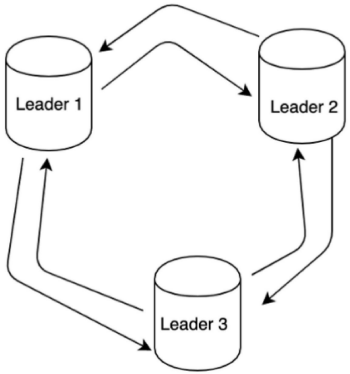
- Writes go to in-memory sorted memtable
- When memtable gets full, flush to SSTable file on disk
- SSTables can be compacted to reclaim space taken up by old values of keys
- Reads first check memtable, then SSTables in order from newest to oldest

- Reads/writes traverse B-tree on disk and operate on data in place

Deep Dives - Replication

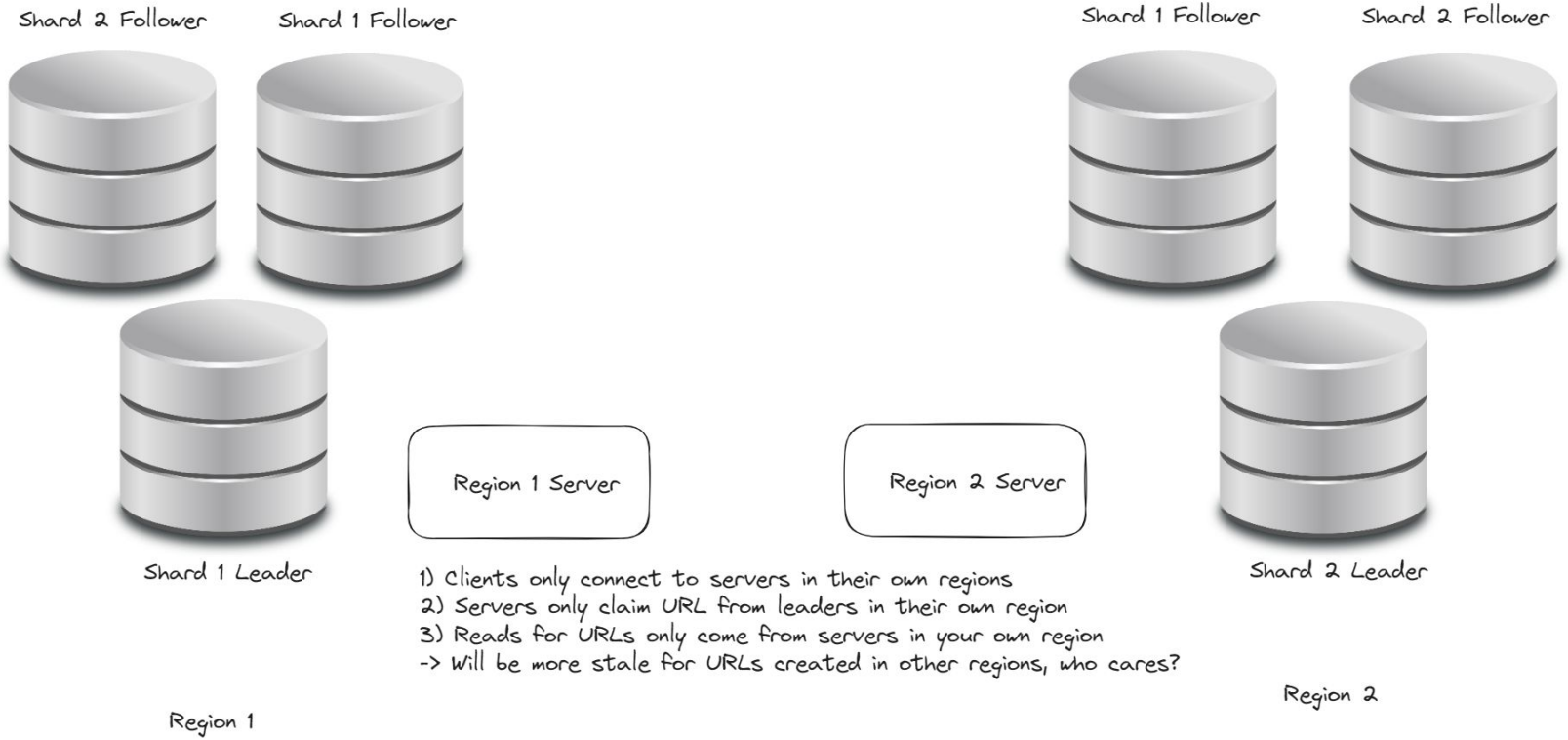


Leaderless Replication



Multi-Leader Replication

Deep Dives - Multiple Regions



Deep Dives - Read Through Caching