Et elle va scaler, cette base de données ?

SnowCamp 2026 – Grenoble

About Me

Vincent Primault, Senior Software Engineer @ Datadog

- Currently working at Datadog on a large-scale workflows platform using the open source Temporal product.
- Previously worked at Salesforce for 6+ years on scaling CRM search.
- Passionate about comics (« *bandes dessinées* ») and strategy board games.

Databases Come in All Shapes and Sizes

Relational Databases

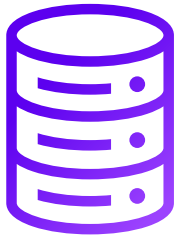
Non-relational Databases

Search Engines

https://dbdb.io

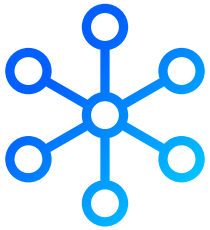

BRACE YOURSELF
LIVE DEMO IS COMING

Dimensions of Database Scaling



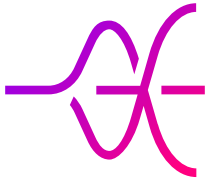
Volume of Data

Limited by disk size, IOPS



Volume of Requests

Limited by CPU, memory, threads, bandwidth



Size of Requests

Limited by CPU, memory

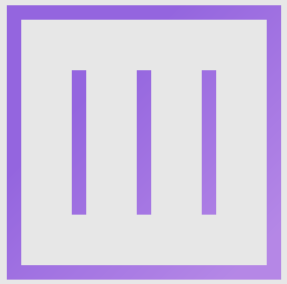
https://en.wikipedia.org/wiki/Database_scalability

Vertical vs Horizontal Scaling

u7inh-32tb.480xlarge

1,920 vCPUs

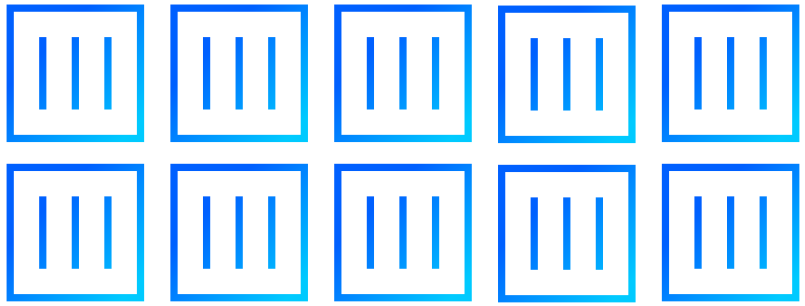
32,768 GiB memory



10 x r8a.48xlarge

1,920 vCPUs

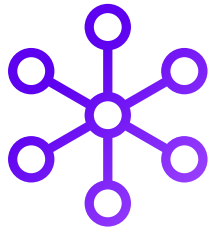
15,360 GiB memory




IF YOU ONLY HAVE "THE SERVER"
YOU PROBABLY DON'T
HAVE HIGH AVAILABILITY

Scaling Technique #1: Data Replication

Challenges to Solve



Volume of Requests

High Availability

Physical vs Logical Replication

Physical Replication

Replicate opaque bytes from the disk

Database-agnostic (similar to backups)

Limited selectivity of what is replicated

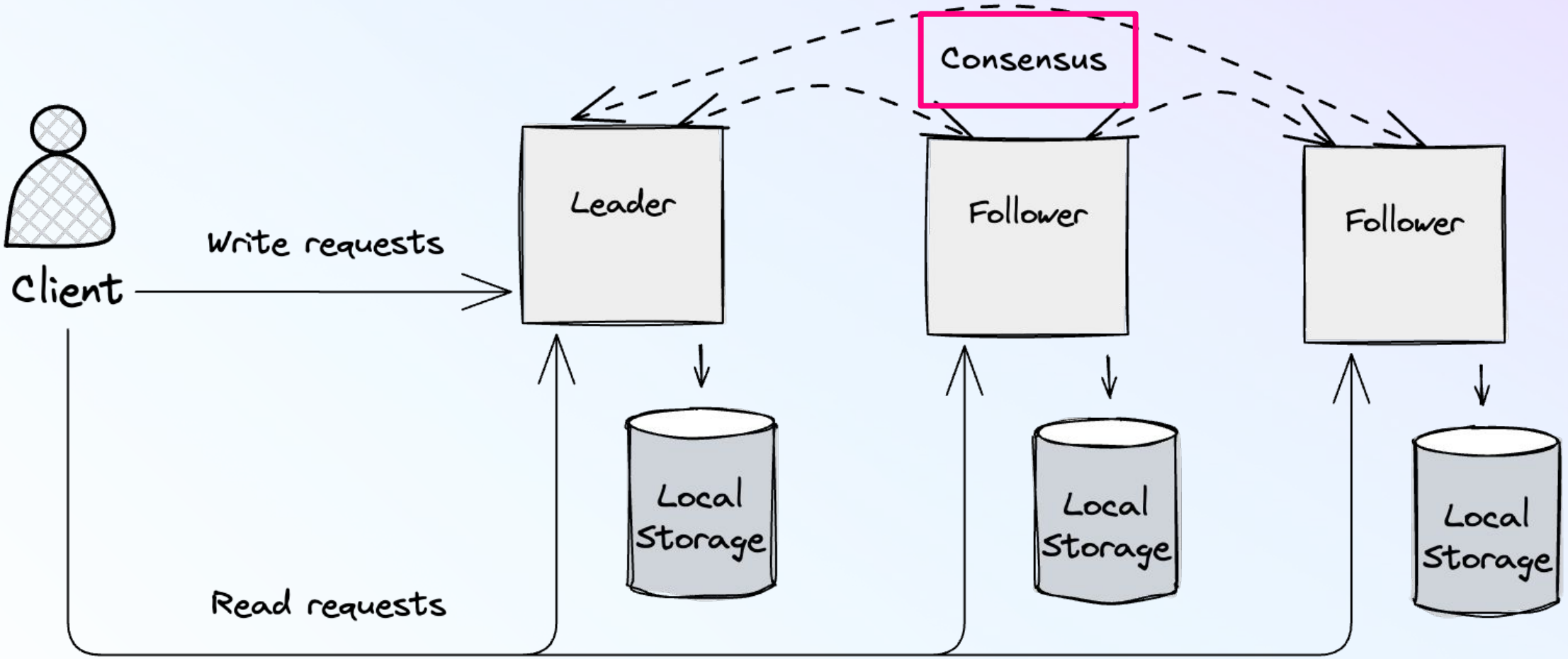
Logical Replication

Replicate application-level operations

Specific to the database

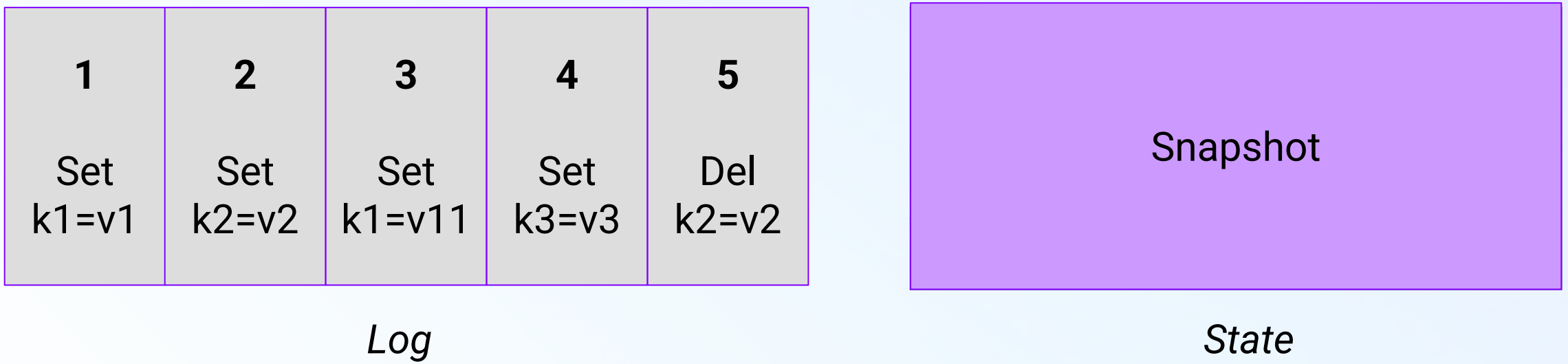
Fine-grain selectivity of what is replicated

Single-Leader Replication



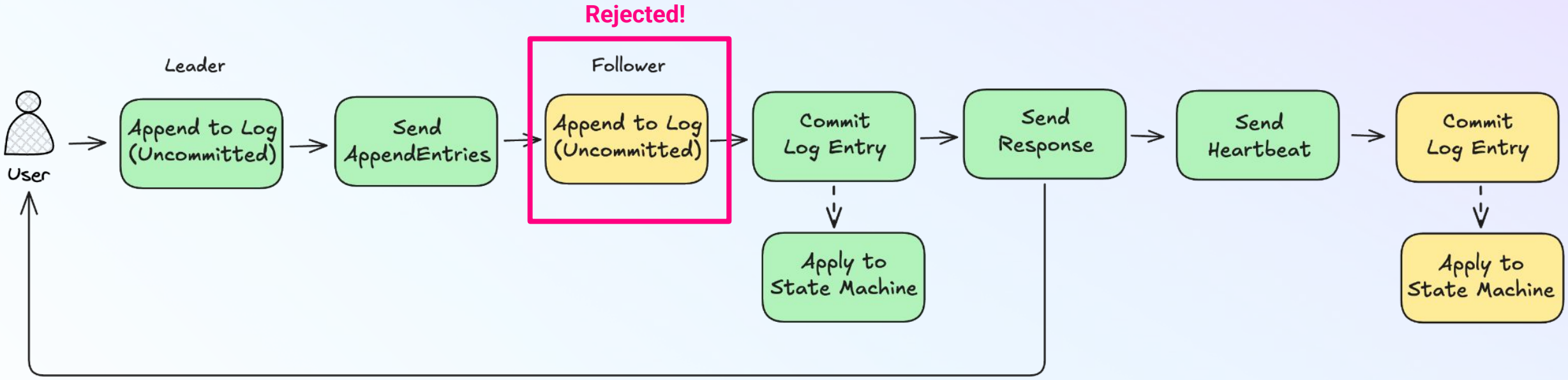
The Log

> A log is perhaps the simplest possible storage abstraction. It is an *append-only* and *totally-ordered* sequence of records *ordered by time*.



https://engineering.linkedin.com/distributed-systems/log-what-every-software-engineer-should-know-about-real-time-datas-unifying

Log Replication with Raft



[In Search of an Understandable Consensus Algorithm]

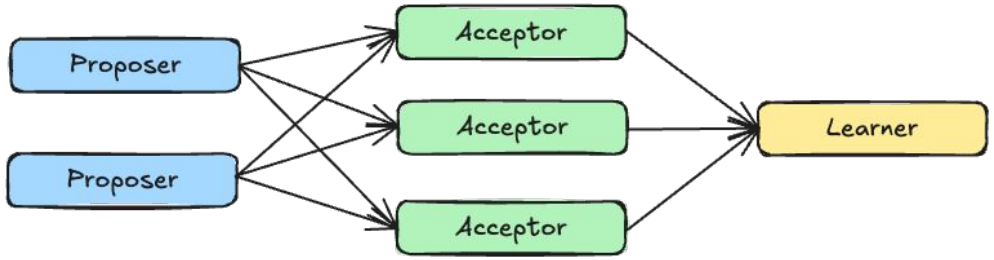

TALK IS CHEAP
SHOW ME THE CODE

Alternatives to Raft

Paxos

Reputed to be more complex to understand

Quorum-based, no dedicated leader



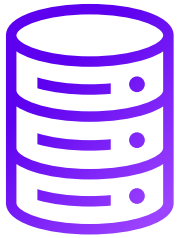
Coordination Services: “Raft-as-a-service”

Offer primitives to build distributed systems

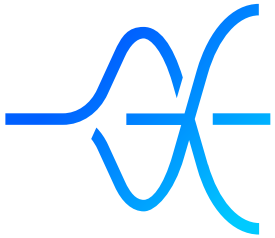
Useful if there is no good quality Raft implementation in your language

Scaling Technique #2: Data Sharding

Challenges to Solve



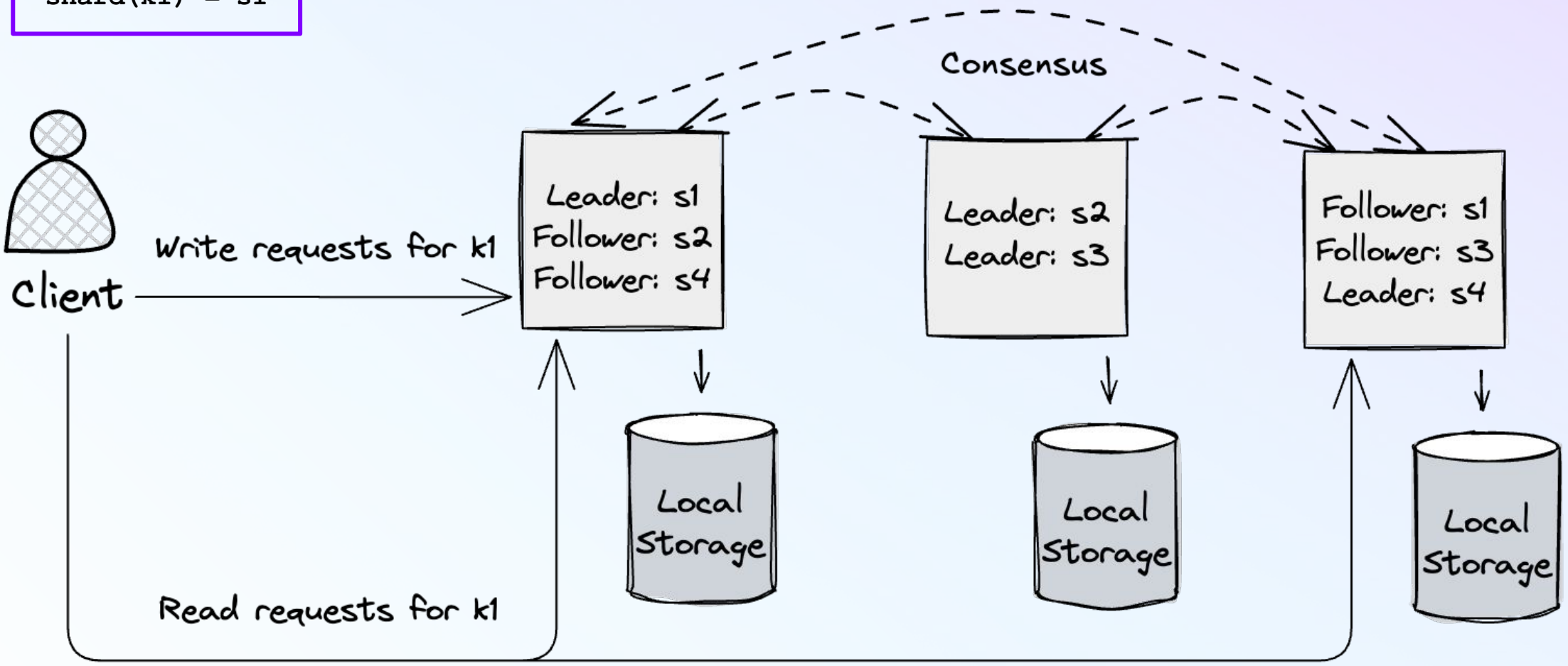
Volume of Data



Size of Requests

Data Sharding

```
shard(k1) = s1
```



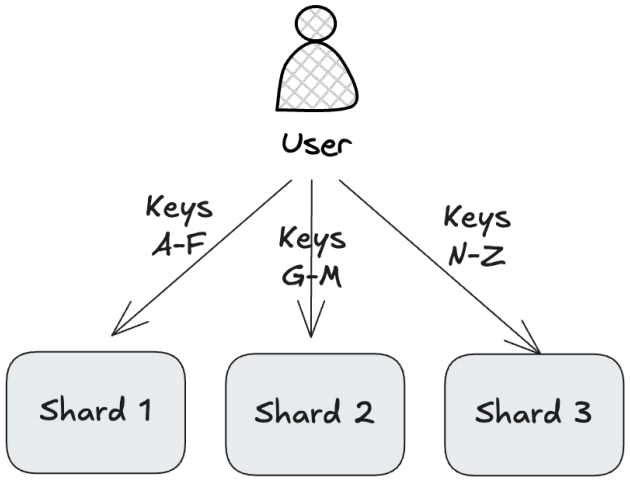
Replication with Multiple Shards

Sharding Functions

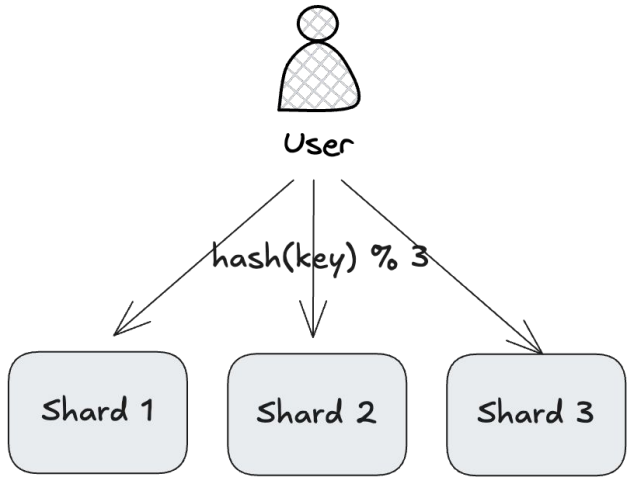
```
f(key) = shard
```

A sharding function *deterministically* assigns a key to a shard. The assignment of replicas to nodes can be done separately.

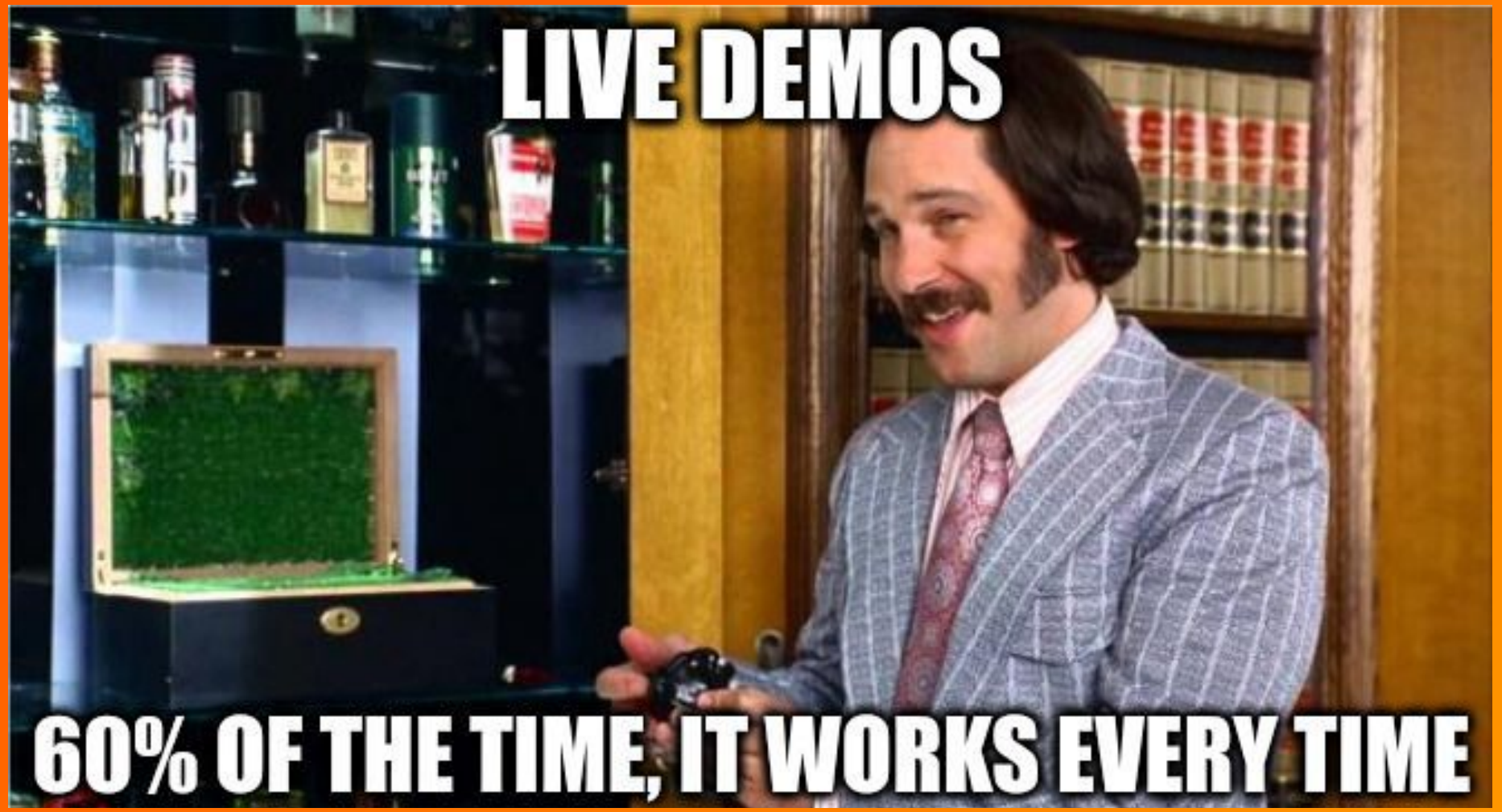
Range-based



Hash-based



DATADOG


LIVE DEMOS
60% OF THE TIME, IT WORKS EVERY TIME

Cluster Management

Adding a Node

The new node needs to be assigned some replicas, which will be moved from other nodes. Data associated with those replicas needs to be transferred before the new node can accept requests.

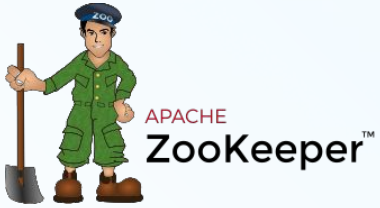
Removing a Node

Replicas assigned to the node to remove need to be moved to other nodes. Once data associated with those replicas has been transferred, the node receives no traffic anymore and can be safely removed.

Rebalancing Replicas

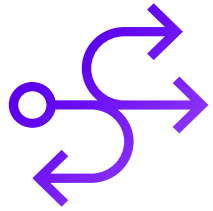
Replicas may need to be moved between nodes to ensure that one or several scaling dimensions are balanced across nodes. For example, we may want all nodes to hold the same volume of data.

Dynamic Cluster Metadata



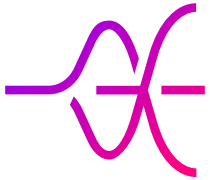
Scaling Technique #3: Compute-Storage Separation

Challenges to Solve



Elasticity

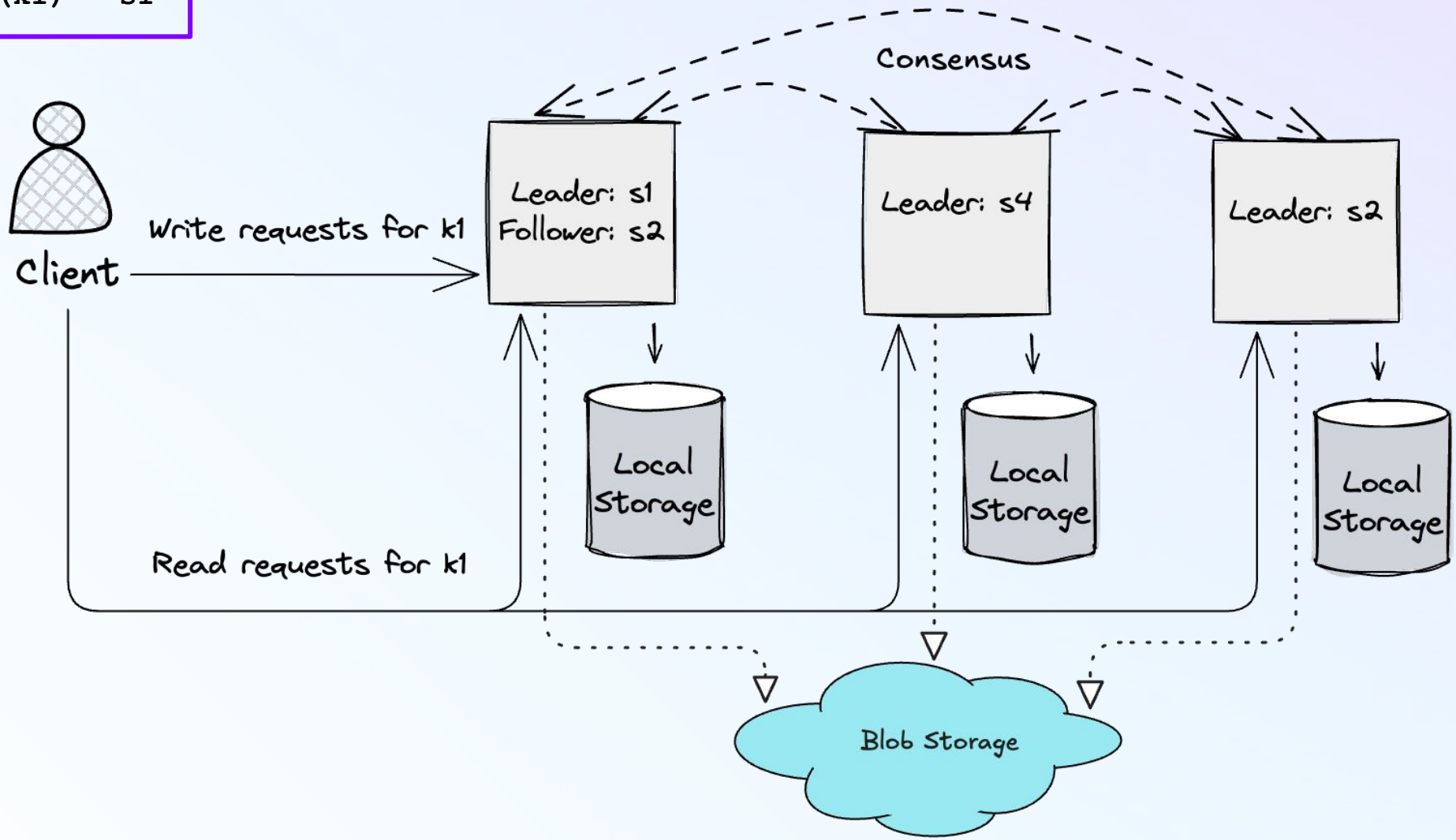
Cost Optimization



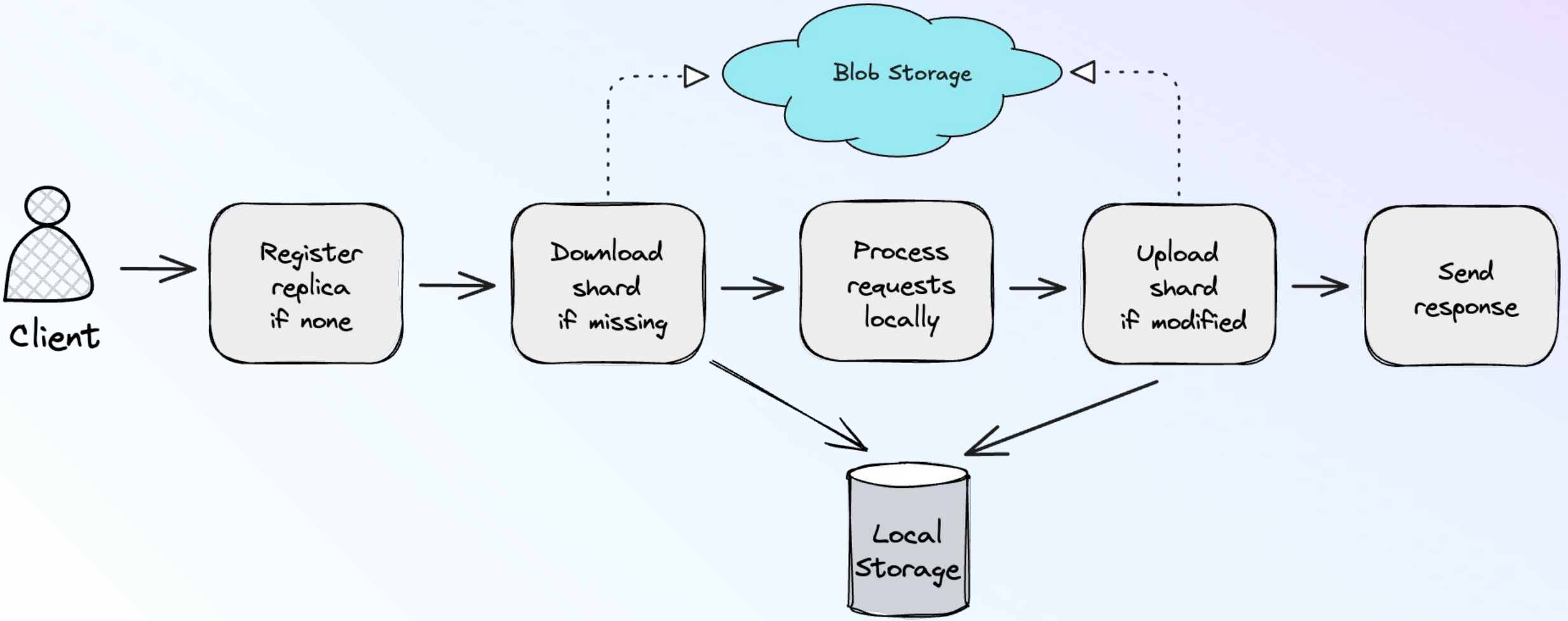
Size of Requests

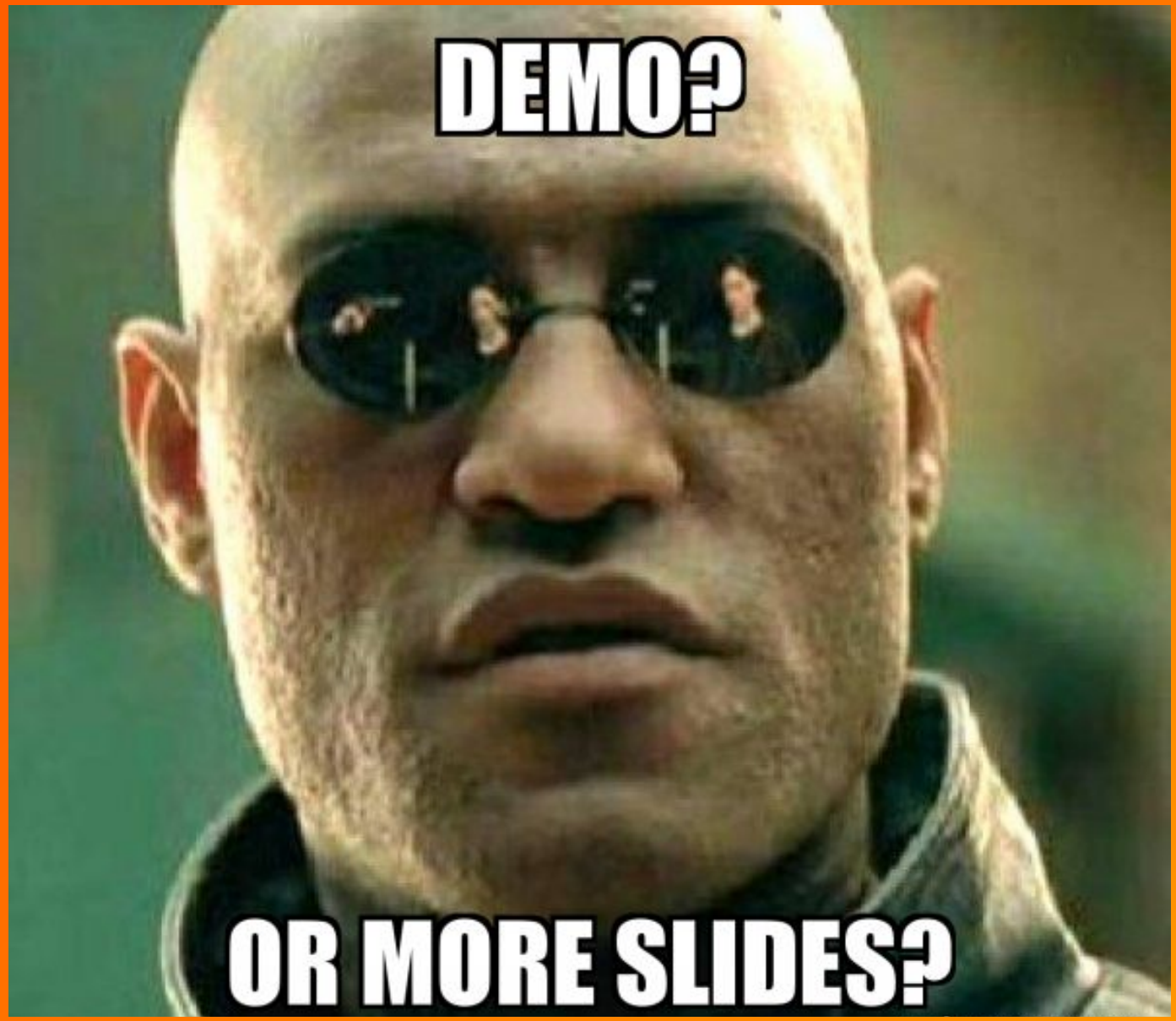
Blob Storage

```
shard(k1) = s1
```



Interactions with Blob Storage

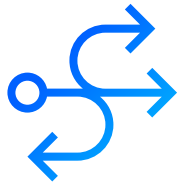



DEMO?
OR MORE SLIDES?

Scaling Superpowers

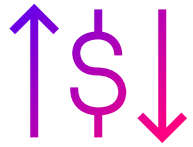
Faster Autoscaling

Starting a new node with an empty disk drastically reduces the time to scale a cluster up. This allows to react to changes of request patterns faster.



Vary Number of Replicas in Shards

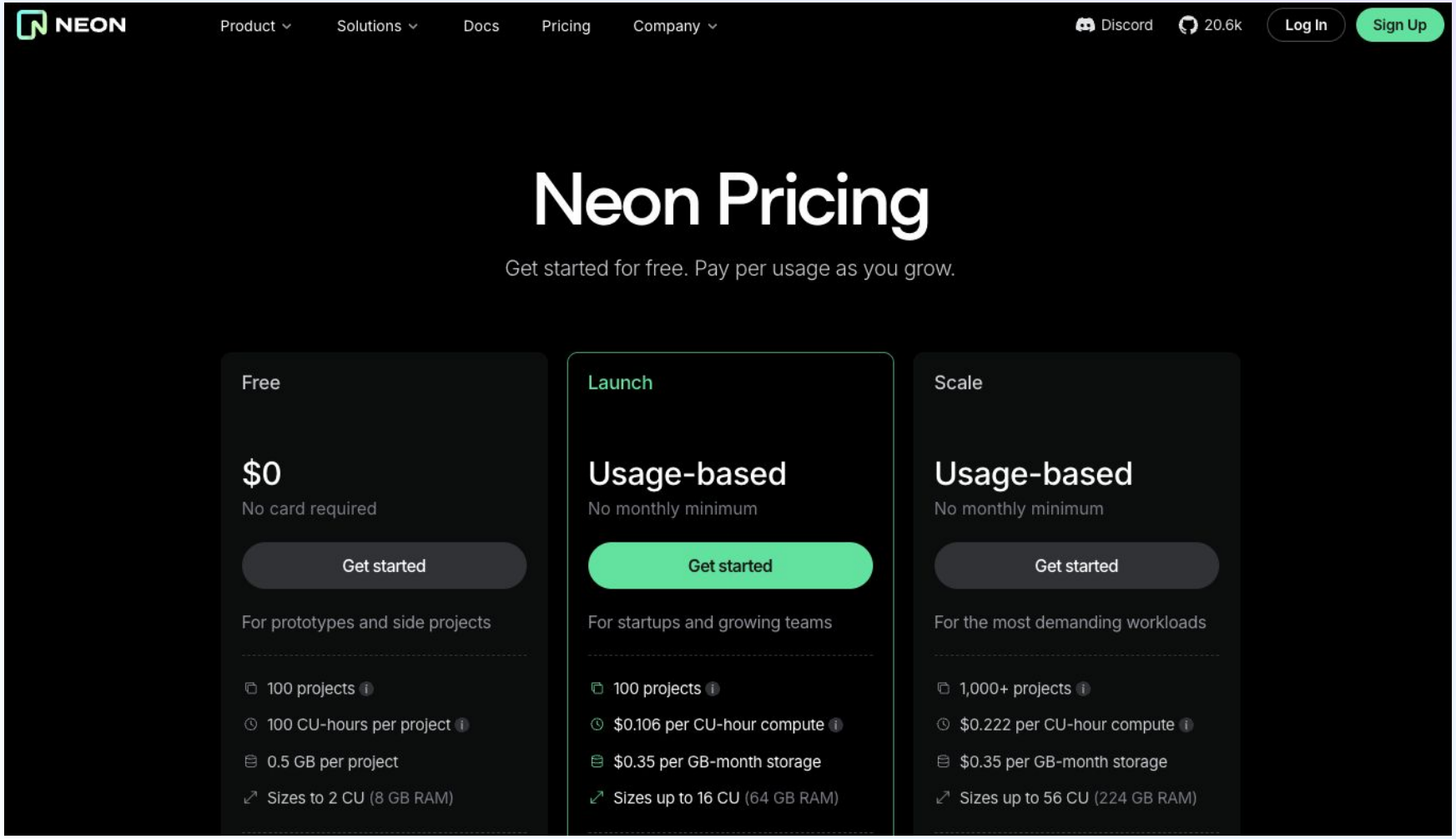
Number of replicas in each shard can be fixed independently. This allows to adapt to the volume of requests and size of requests.



Scale to Zero

Shards can have zero replicas because blob storage handles durability. This allows to save money when data is never accessed.

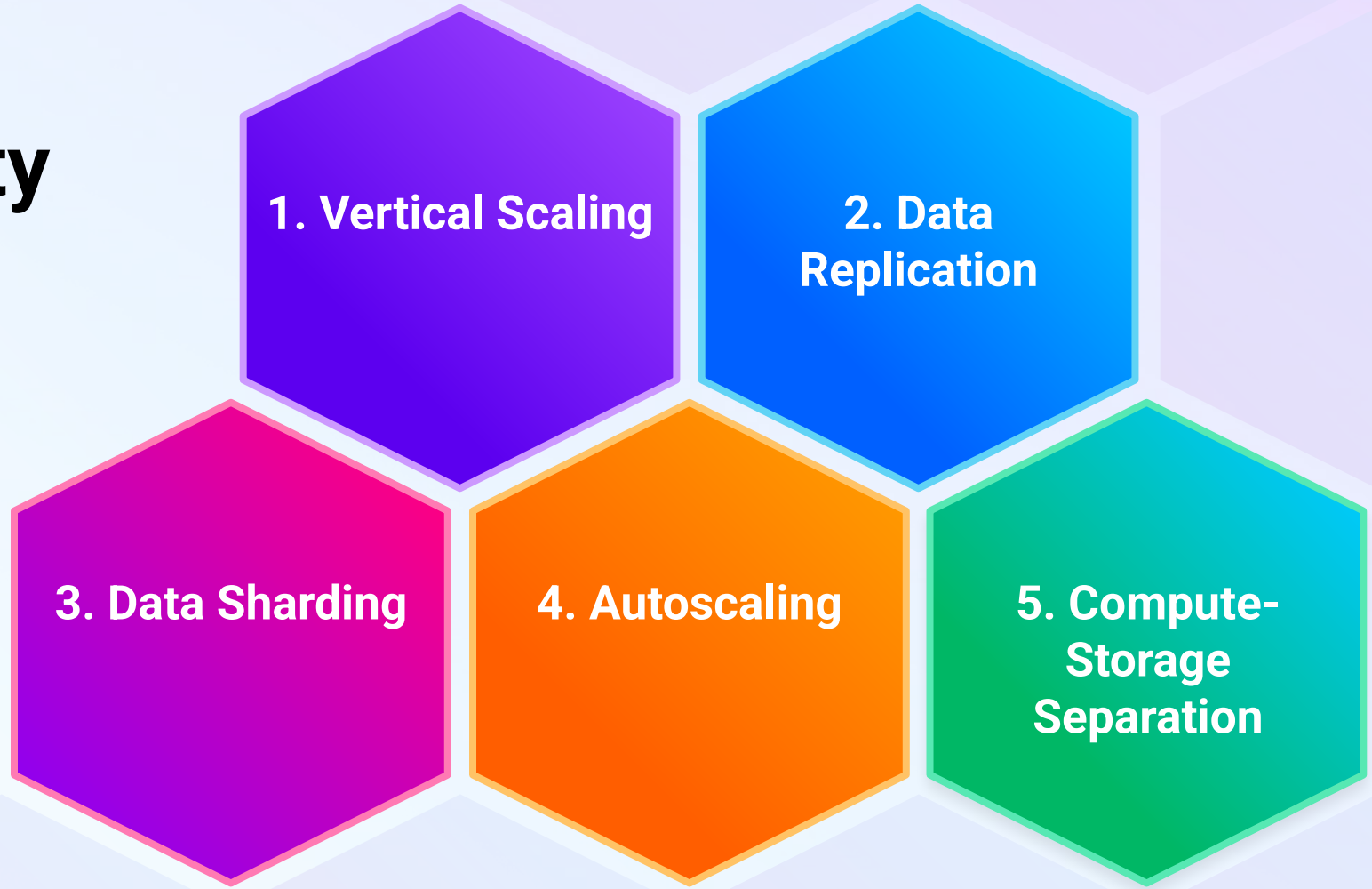
Serverless Databases?



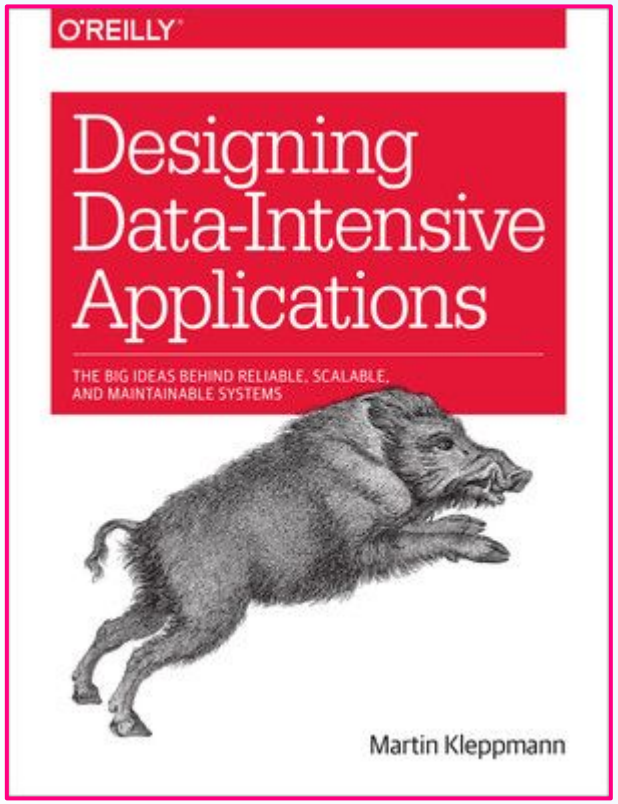
Conclusion

The Journey Towards Scalability

The example of Husky:
https://www.datadoghq.com/blog/engineering/introducing-husky/



To Continue Exploring…



Amazon DynamoDB: A Scalable, Predictably Performant, and Fully Managed NoSQL Database Service

Mostafa Elhemali, Niall Gallagher, Nicholas Gordon, Joseph Idziorek, Richard Krog
Colin Lazier, Erben Mo, Akhilesh Mritunjai, Somu Perianayagam ,Tim Rath
Swami Sivasubramanian, James Christopher Sorenson III, Sroaj Sosothikul, Doug Terry, Akshat Vig
dynamodb-paper@amazon.com
Amazon Web Services

Abstract

Amazon DynamoDB is a NoSQL cloud database service that provides consistent performance at any scale. Hundreds of thousands of customers rely on DynamoDB for its fundamental properties: consistent performance, availability, durability, and a fully managed serverless experience. In 2021, during the 66-hour Amazon Prime Day shopping event, Amazon systems - including Alexa, the Amazon.com sites, and Amazon fulfillment centers, made trillions of API calls to DynamoDB, peaking at 89.2 million requests per second, while experiencing high availability with single-digit millisecond performance. Since the launch of DynamoDB in 2012, its design and implementation have evolved in response to our experiences operating it. The system has successfully dealt with issues related to fairness, traffic imbalance across partitions, monitoring, and automated system operations without impacting availability or performance. Reliability is essential, as even the slightest disruption can significantly impact customers. This paper presents our experience operating DynamoDB at a massive scale and how the architecture continues to evolve to meet the ever-increasing demands of customer workloads.

1 Introduction

Amazon DynamoDB is a NoSQL cloud database service that supports fast and predictable performance at any scale. DynamoDB is a foundational AWS service that serves hundreds of thousands of customers using a massive number of servers located in data centers around the world. DynamoDB powers multiple high-traffic Amazon properties and systems including Alexa, the Amazon.com sites, and all Amazon fulfillment centers. Moreover, many AWS services such as AWS Lambda, AWS Lake Formation, and Amazon SageMaker are built on DynamoDB, as well as hundreds of thousands of customer applications.

These applications and services have demanding operational requirements with respect to performance, reliability, durability, efficiency, and scale. The users of DynamoDB rely on its ability to serve requests with consistent low latency. For DynamoDB customers, consistent performance at any scale is often more important than median request service times because unexpectedly high latency requests can amplify through higher layers of applications that depend on DynamoDB and lead to a bad customer experience. The goal of the design of DynamoDB is to complete *all* requests with low single-digit millisecond latencies. In addition, the large and diverse set of customers who use DynamoDB rely on an ever-expanding feature set as shown in Figure 1. As DynamoDB has evolved over the last ten years, a key challenge has been adding features without impacting operational requirements. To benefit customers and application developers, DynamoDB uniquely integrates the following six fundamental system properties:

DynamoDB is a fully managed cloud service. Using the DynamoDB API, applications create tables and read and write data without regard for where those tables are stored or how they're managed. DynamoDB frees developers from the burden of patching software, managing hardware, configuring a distributed database cluster, and managing ongoing cluster operations. DynamoDB handles resource provisioning, automatically recovers from failures, encrypts data, manages software upgrades, performs backups, and accomplishes other tasks required of a fully-managed service.

DynamoDB employs a multi-tenant architecture. DynamoDB stores data from different customers on the same physical machines to ensure high utilization of resources, enabling us to pass the cost savings to our customers. Resource reservations, tight provisioning, and monitored usage provide isolation between the workloads of co-resident tables.

DynamoDB achieves boundless scale for tables. There are no predefined limits for the amount of data each table can store. Tables grow elastically to meet the demand of the customers' applications. DynamoDB is designed to scale the resources dedicated to a table from several servers to many thousands as needed. DynamoDB spreads an application's data across more servers as the amount of data storage and the demand for throughput requirements grow.

DynamoDB provides predictable performance. The simple

Thank you!

https://www.linkedin.com/in/vincent-primault/

https://github.com/pvcnt/kvdog

DATADOG