
KubeCon
CloudNativeCon
Europe 2025
LONDON BUS

KubeCon | CloudNativeCon
Europe 2025

# Taming 50 Billion Time Series

## Operating Global-Scale Prometheus Deployments on Kubernetes



# Alan Protasio

Sr. Software Engineer, AWS
/alanprot

# Orcun Berkem

Principal Engineer, AWS
/oberkem


KubeCon | CloudNativeCon
Europe 2025



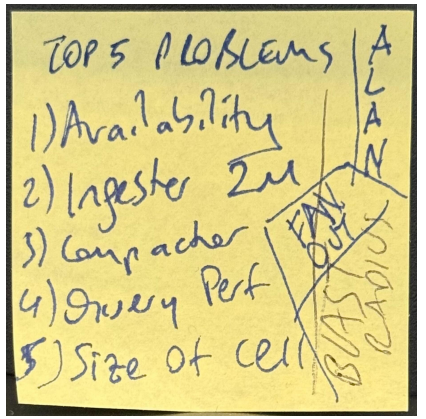
# 99 Problems



Growth in time series

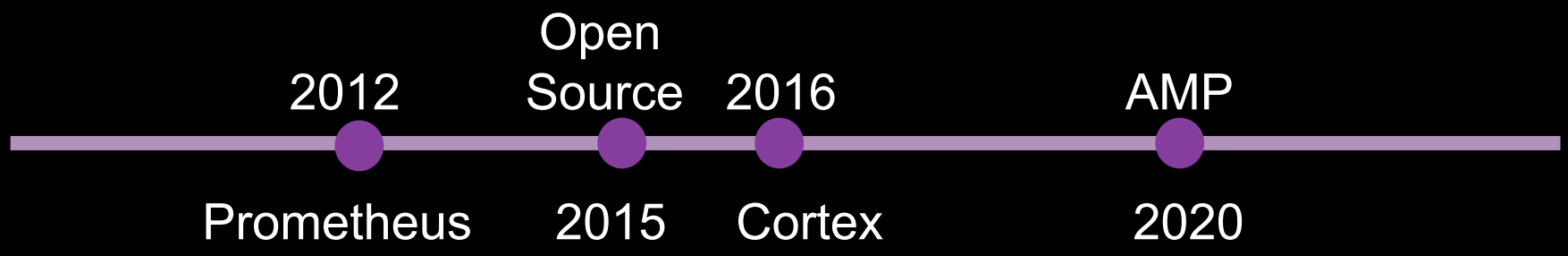
```
cpu_utilization(pod="pod_A1", cluster="cluster_A", service="search")           -> 1 timeseries
cpu_utilization(pod="pod_B1", cluster="cluster_A", service="product_details")  -> 2 timeseries
cpu_utilization(pod="pod_C1", cluster="cluster_A", service="cart")             -> 3 timeseries
cpu_utilization(pod="pod_D1", cluster="cluster_A", service="checkout")         -> 4 timeseries
cpu_utilization(pod="pod_A2", cluster="cluster_A", service="search")           -> 5 timeseries
... let's say we scale out to 16 pods                                          -> 16 timeseries
... let's say we scale out to 16 pods for each service                         -> 16 x 4 = 64 timeseries
... let's say add "node" label and have 10 nodes                               -> 64 x 10 = 640 timeseries
... let's say we have the same dimensions on labels on 20 metrics              -> 620 x 20 = 12,400 timeseries
... let's say we add ip addresses w/ 1000 values on each metric                -> 12M timeseries
```

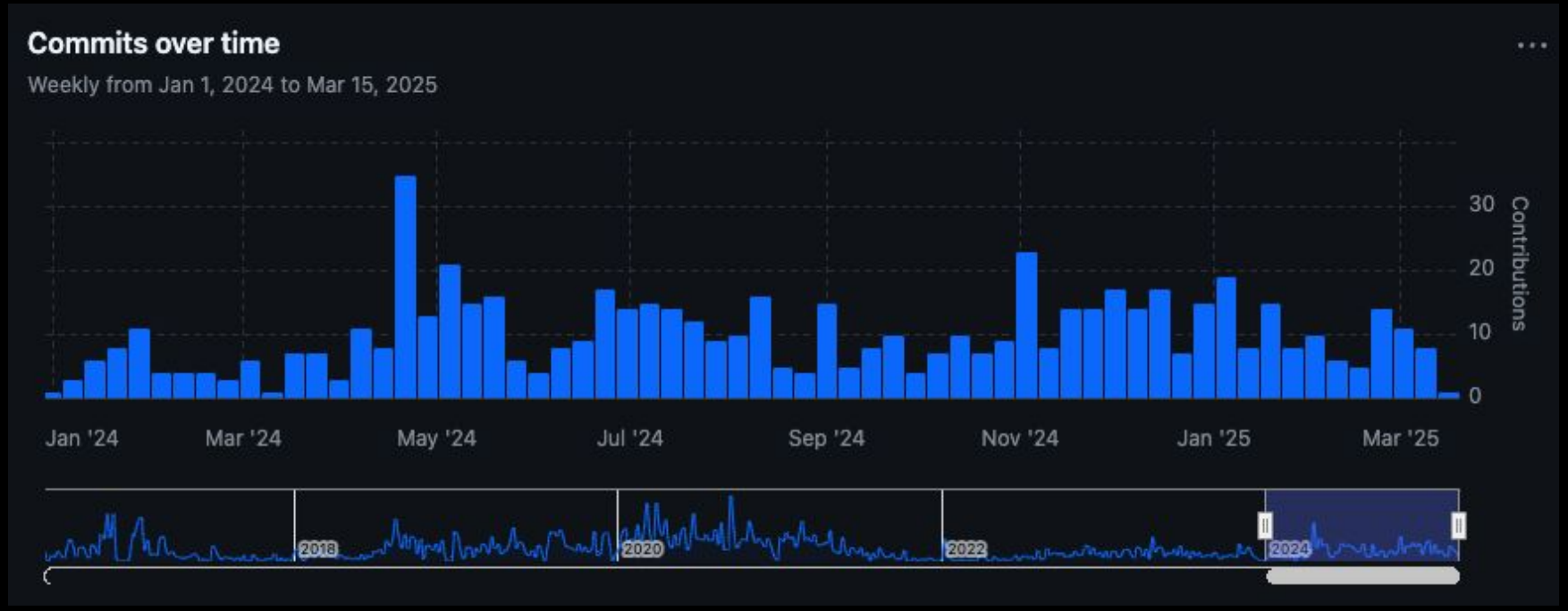
Availability
Multi-Tenancy
Blast Radius Reduction
Operating at global scale
Kubernetes Complexity



# Brief History





# Our 2020 Vision



**AMP (Amazon Managed Service for Prometheus) -** A fully managed, Prometheus-compatible monitoring service by AWS.

**Scalable**
Automatically handles large workloads

**High Available**
Replicate data across multiple AWS Availability Zones

**Integrated**
Works seamlessly with ECS, EKS, and other AWS services

**Secure**
Supports IAM-based authentication & data encryption

**Data Retention**
Retaining data for years

**Managed Collector**
Offers a fully managed, agentless scraper

**So, what to use…**

# Why not Prometheus?



## Scalability

**Manual setup** is needed to shard data across multiples instances
**Lack of aggregate view** across instances

## Availability

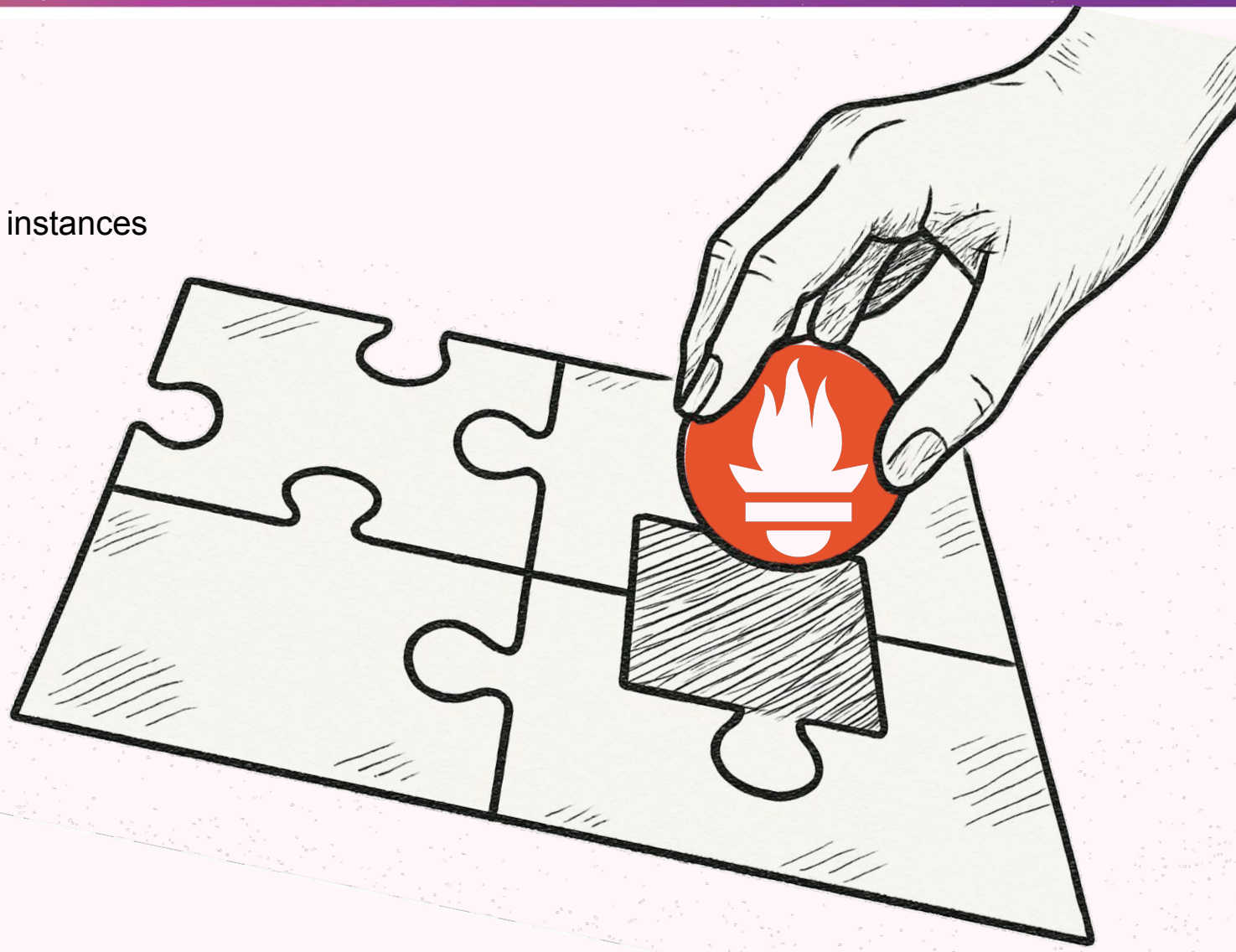
**No built-in support** for high availability

## Storage

All data is kept in a local storage.
As your data grows, **local storage can quickly become a limitation**

## Multi Tenancy

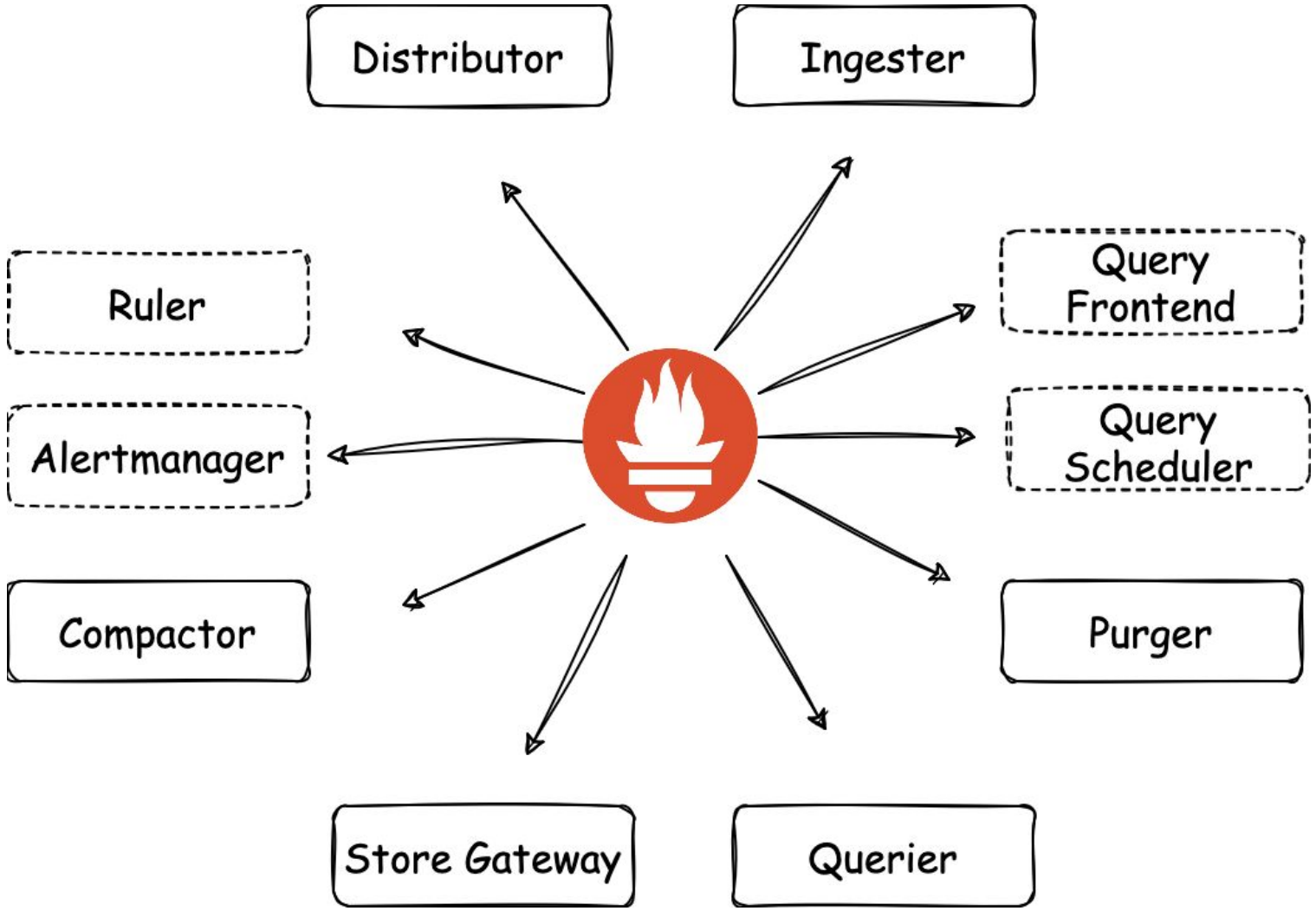
**No native support for a multi-tenancy**



# …What about Cortex?

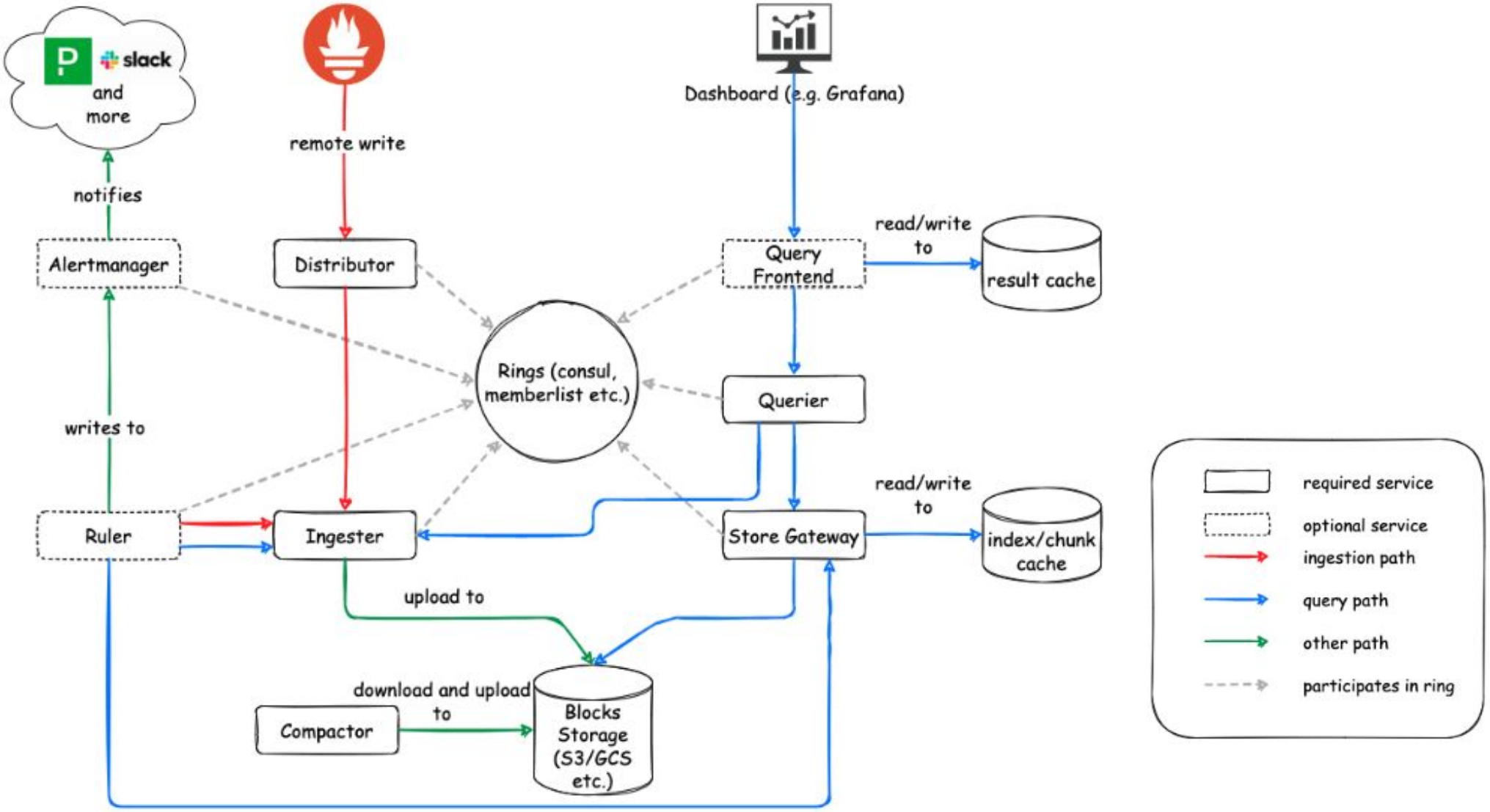


cortex



# Cortex Architecture

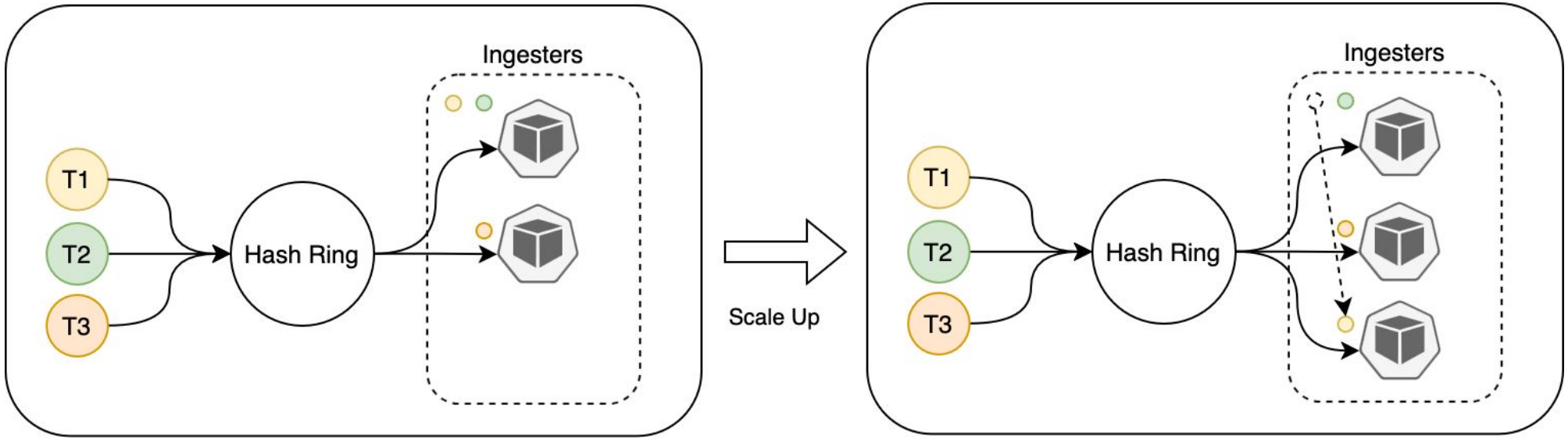




# Horizontally Scalable

**Each cortex component can scale individually**

**Uses consistent hash to rebalance the load automatically (auto balance)**



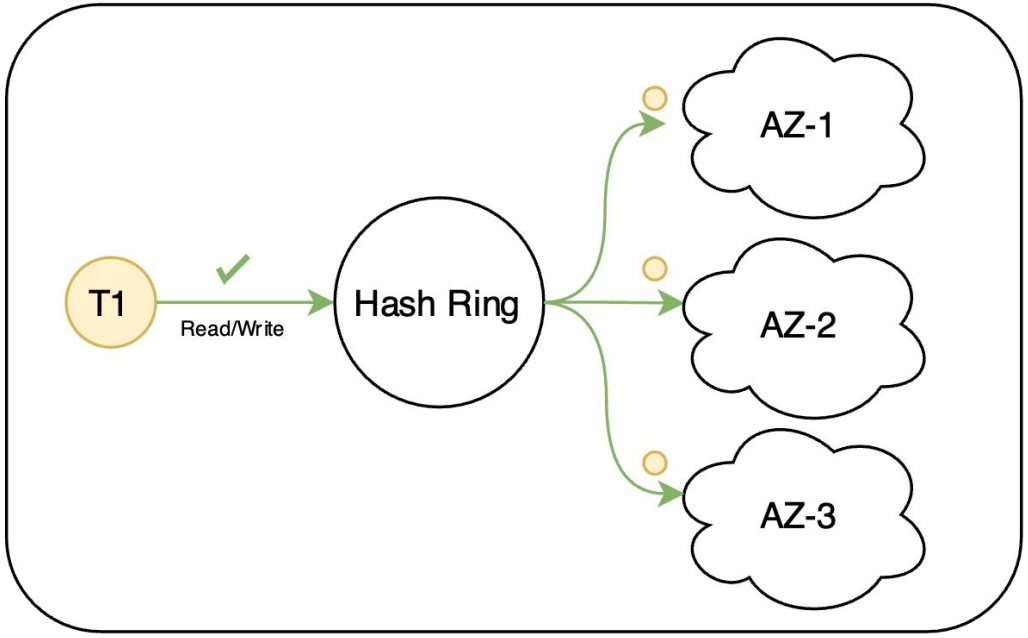
# Why Cortex?

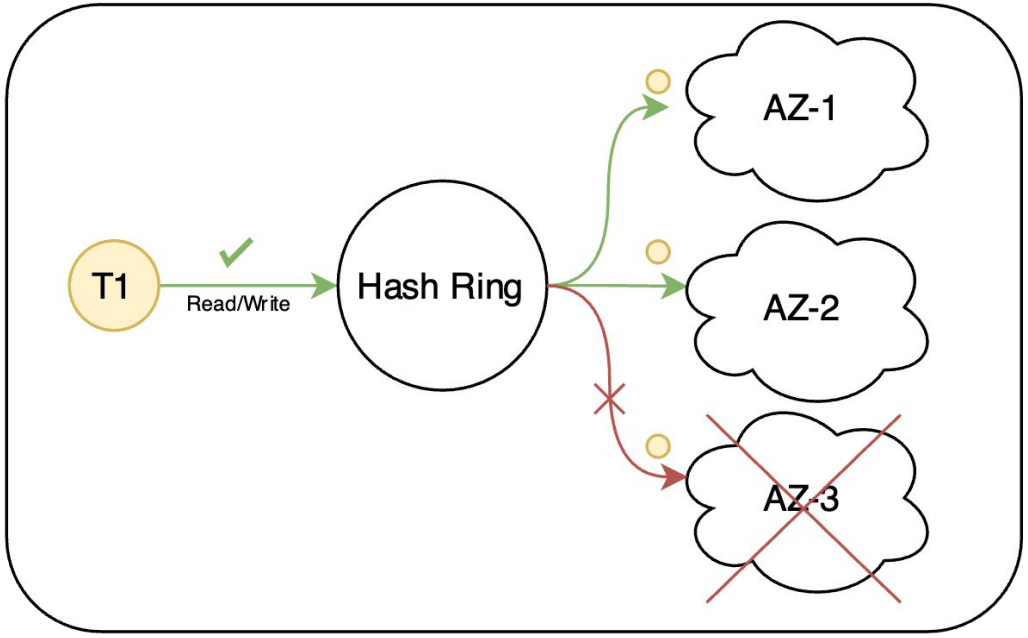


## Highly Available

**Replicates the data across multiples nodes and availability zones**

**Uses Quorum to read and write data**



# Why Cortex?

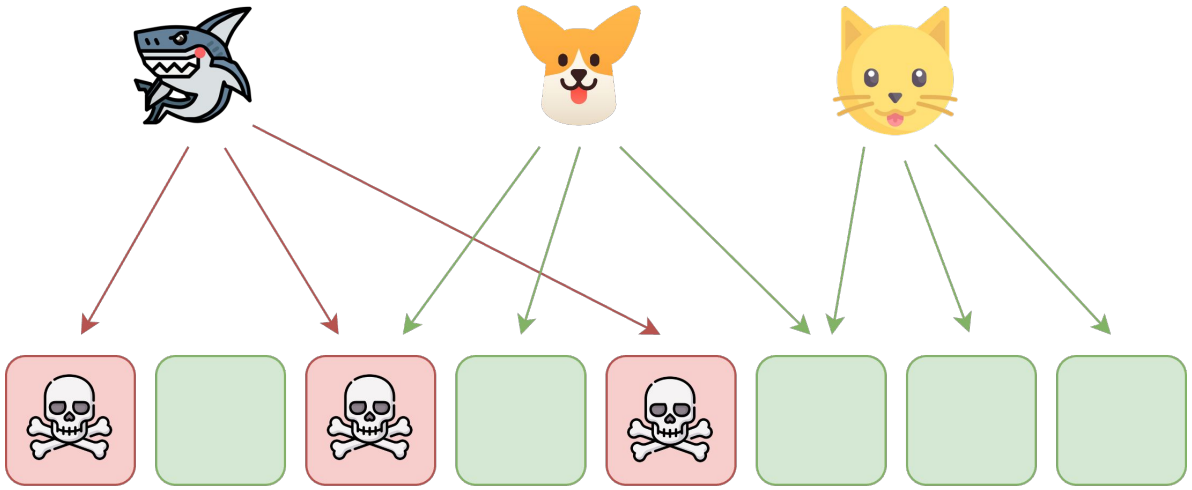


## Native support for Multi Tenancy

**Data Isolation**

**Rate limits & quotas**

**Implements multiple noisy neighbor mitigations such as shuffle sharding**



*Link: https://aws.amazon.com/blogs/architecture/shuffle-sharding-massive-and-magical-fault-isolation/*

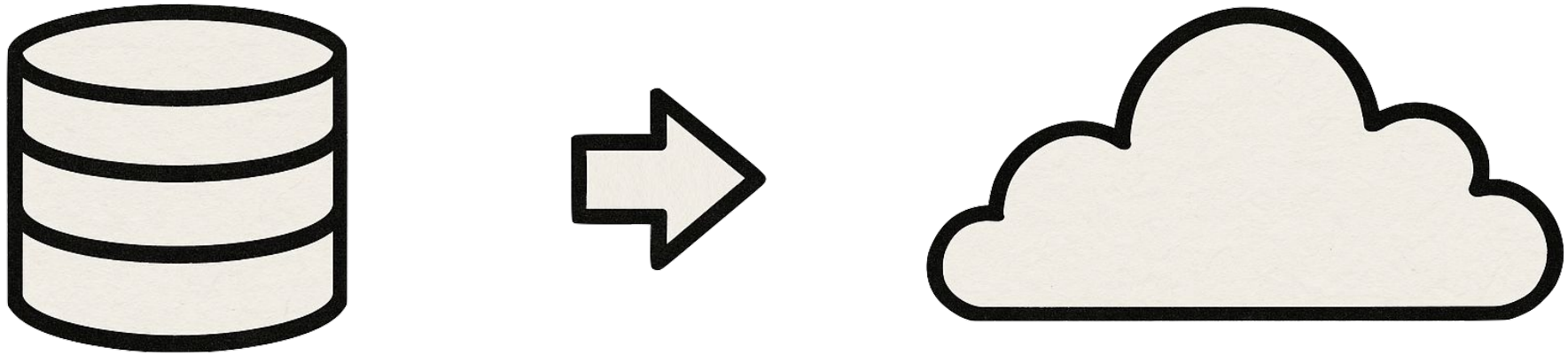
# Long-Term Storage

Uses object storage (S3, GCS, Azure) for data persistence

Enables querying data directly from object storage

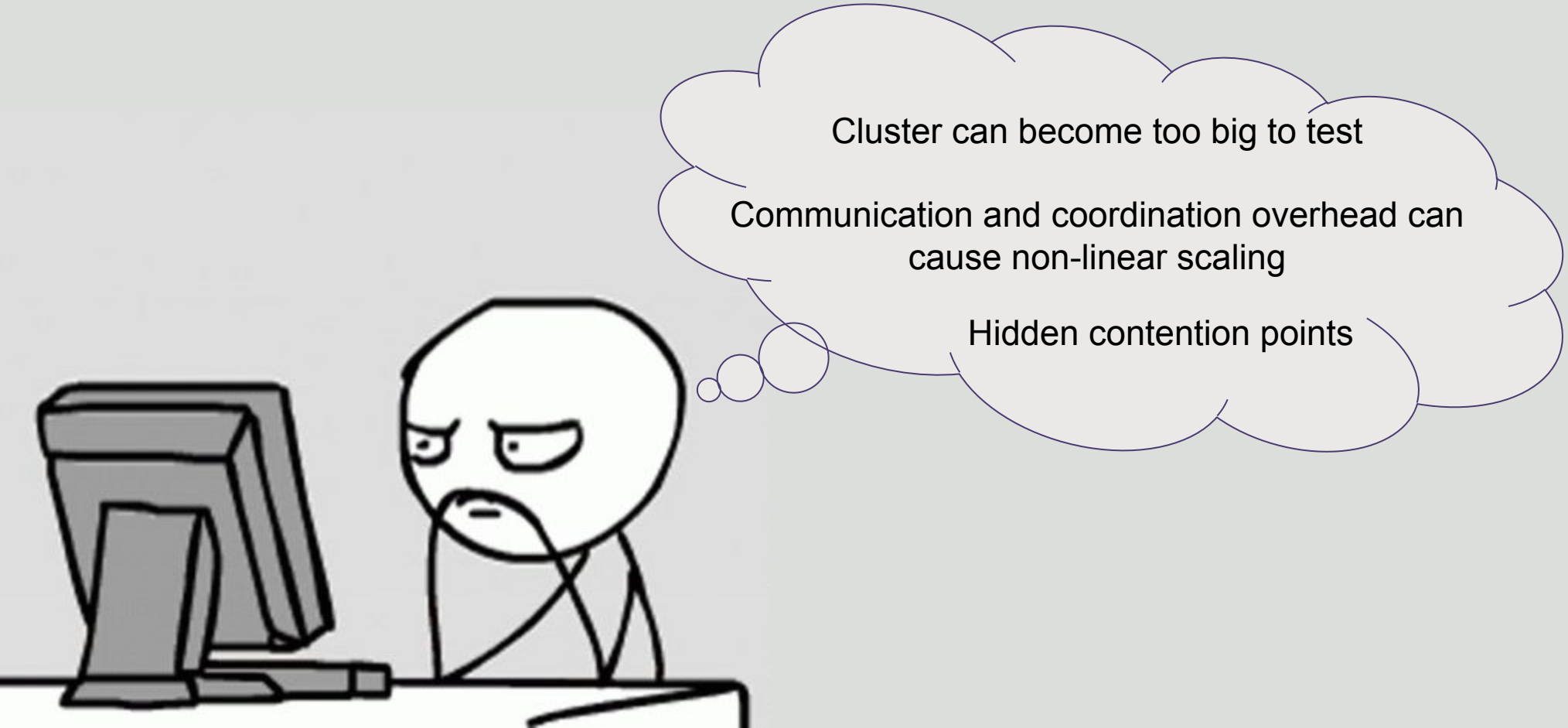
Improves data durability and reliability

Enables extended retention policies - Storing data for years



# Ok, Cortex seems a good fit,

# but we still have problems…



# Cellular Architecture



## What is is?

A **scalable, resilient** architecture pattern

System is divided into **independent "cells"**

Each cell is **self-contained** and serves a subset of users/traffic

### More Scalability

Cells can be added as demand grows

### Blast Radius Reduction

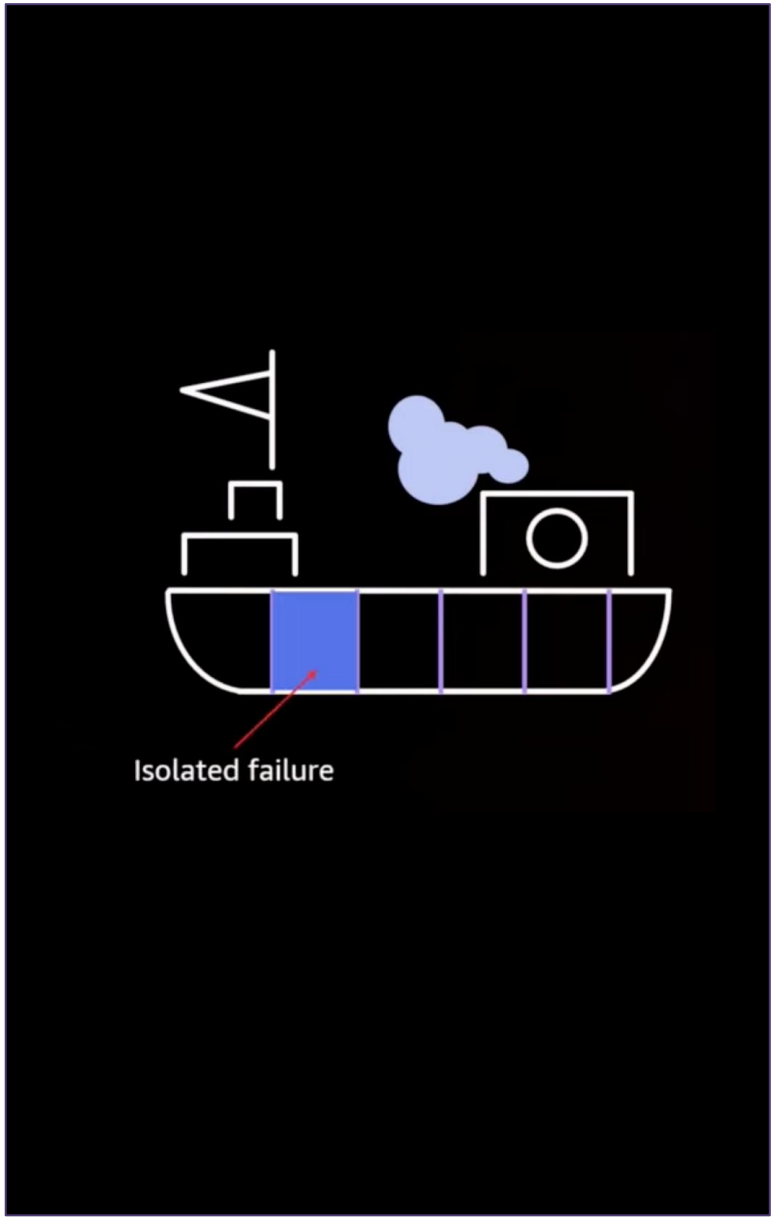
Failure in one cell **does not** impact others

### Safer changes

Changes are deployed per cell

### Lower mean time to recovery (MTTR)

Reduced complexity with smaller, self-contained cells



# Cellular Architecture

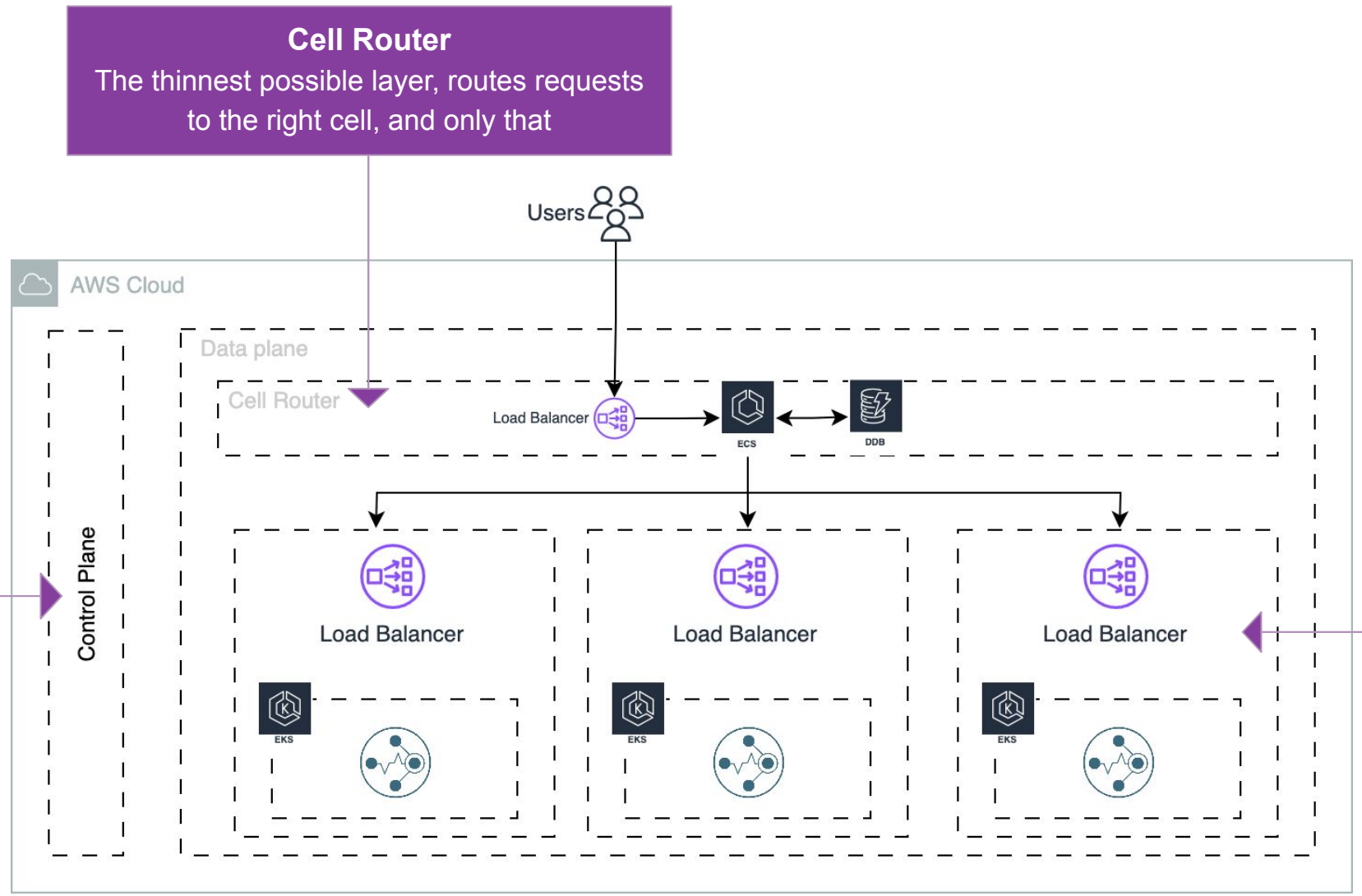
KubeCon CloudNativeCon Europe 2025

## Components

**Cell Router**
The thinnest possible layer, routes requests to the right cell, and only that

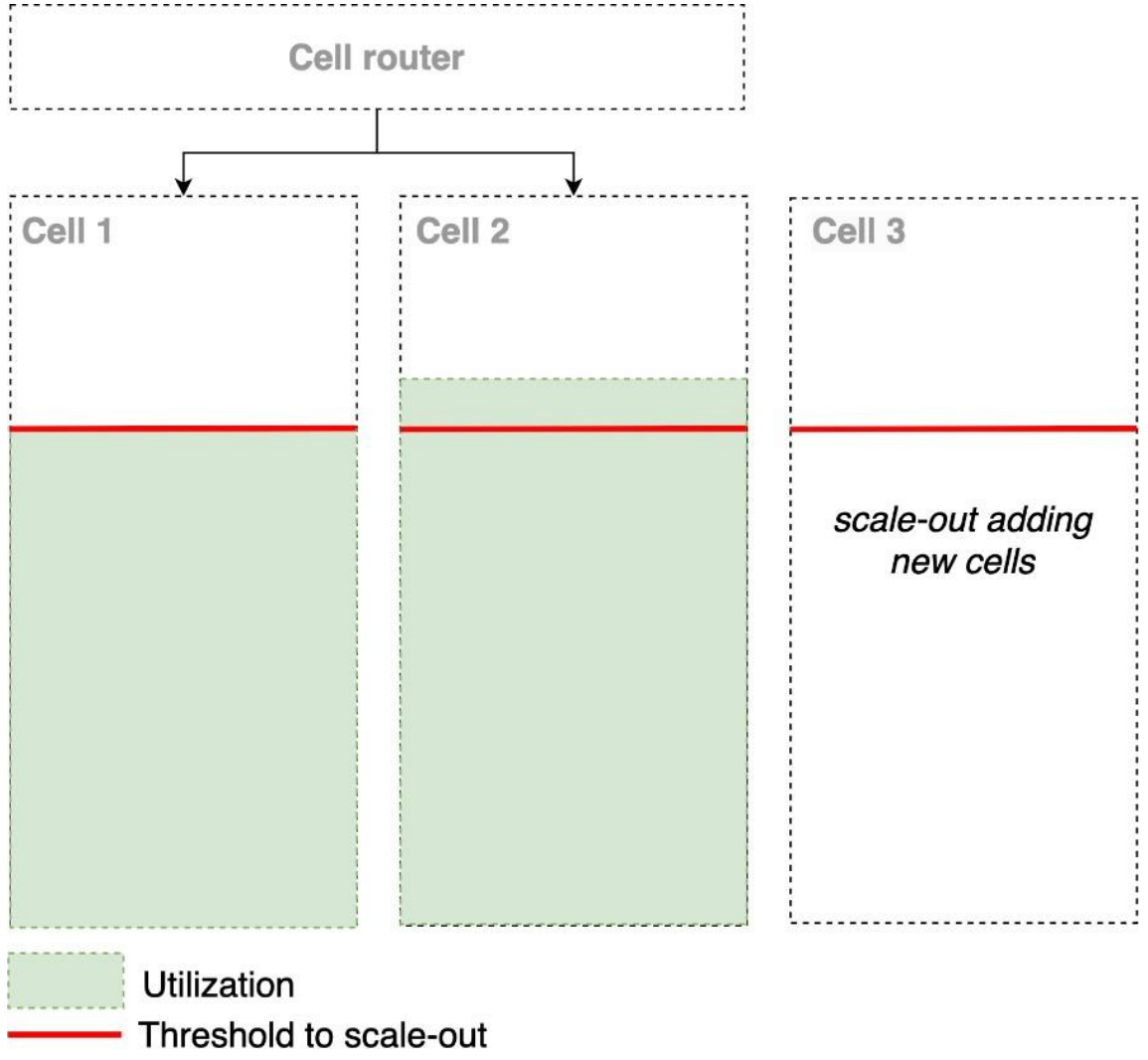
**Control Plane**
Responsible for administration tasks

**Cell**
A complete workload, with everything needed to operate independently



# Capacity Management

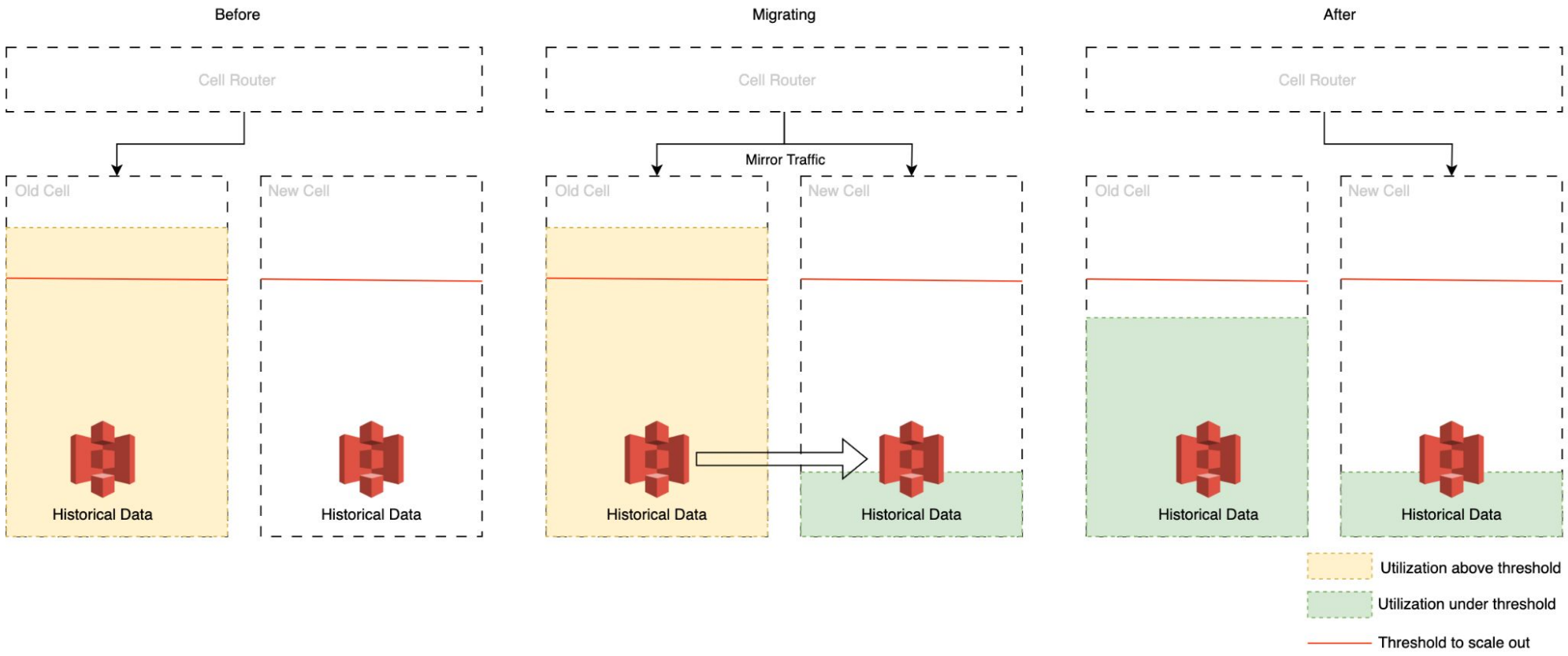
- Scales out instead of scaling up
- Keep cell size within a safe limit
- Dynamic Limits and Quotas
- Dynamic Resource Allocation



# Cell Migration

## Existing customer growth inside a cell

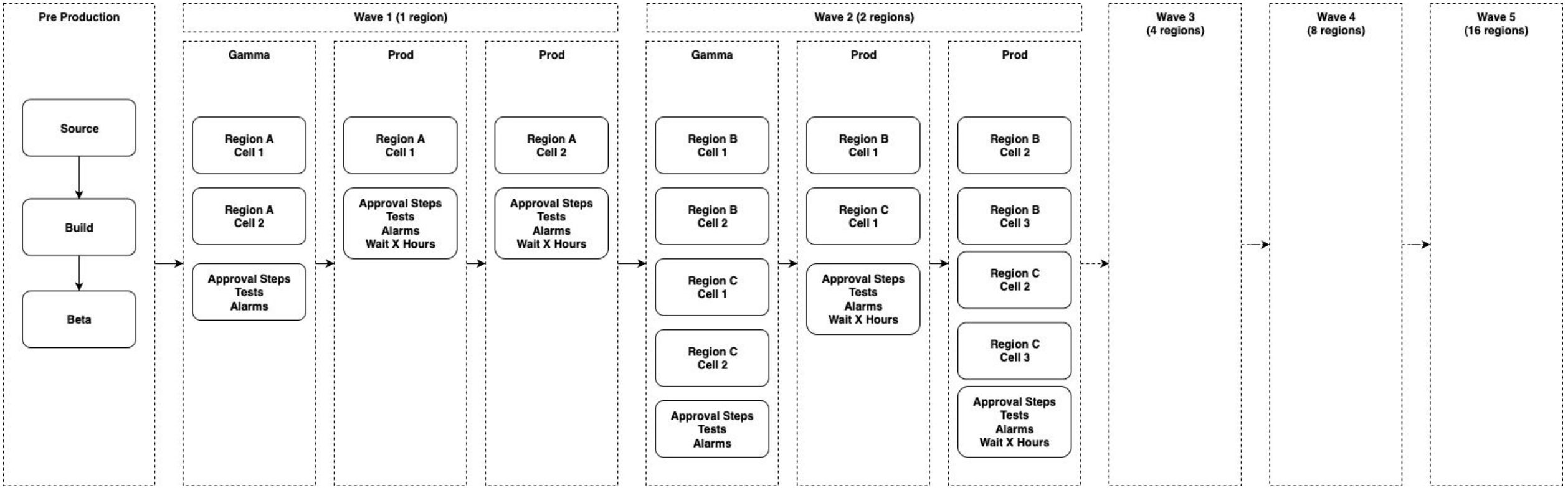
A cell migration may be needed when a customer or group grows too large, exceeding the cell's limit.



# Deployment Waves



- [x] Safety
- [x] Cellular Boundaries
- [x] Regional Boundaries



# Quick word about Testing



### 🧱 Unit Testing

- Tests individual components or functions in isolation
- Fast, reliable, and foundational for test-driven development
- Helps catch bugs early and ensure code correctness

### ✅ Acceptance/Integration Tests

- End-to-end validation of system behavior
- Ensure the application is ready for production
- Integral part of CI/CD pipelines

### 🐤 Canary Tests

- Continuously run in all environments including production
- Useful to detect non-code-related issues (e.g., dependency failures)
- Also uncover issues on less frequently exercised code paths

### 🎲 Fuzz and correctness Tests

- Sends unexpected and random data to the system and compare the results against a baseline (e.g., vanilla Prometheus)
- Identify crashes, edge case bugs, correctness issues, and vulnerabilities

### ⚡ Fault Injection

- Deliberately introduces failures (e.g. network latency, disk errors)
- Validate the system’s resilience and graceful degradation under stress

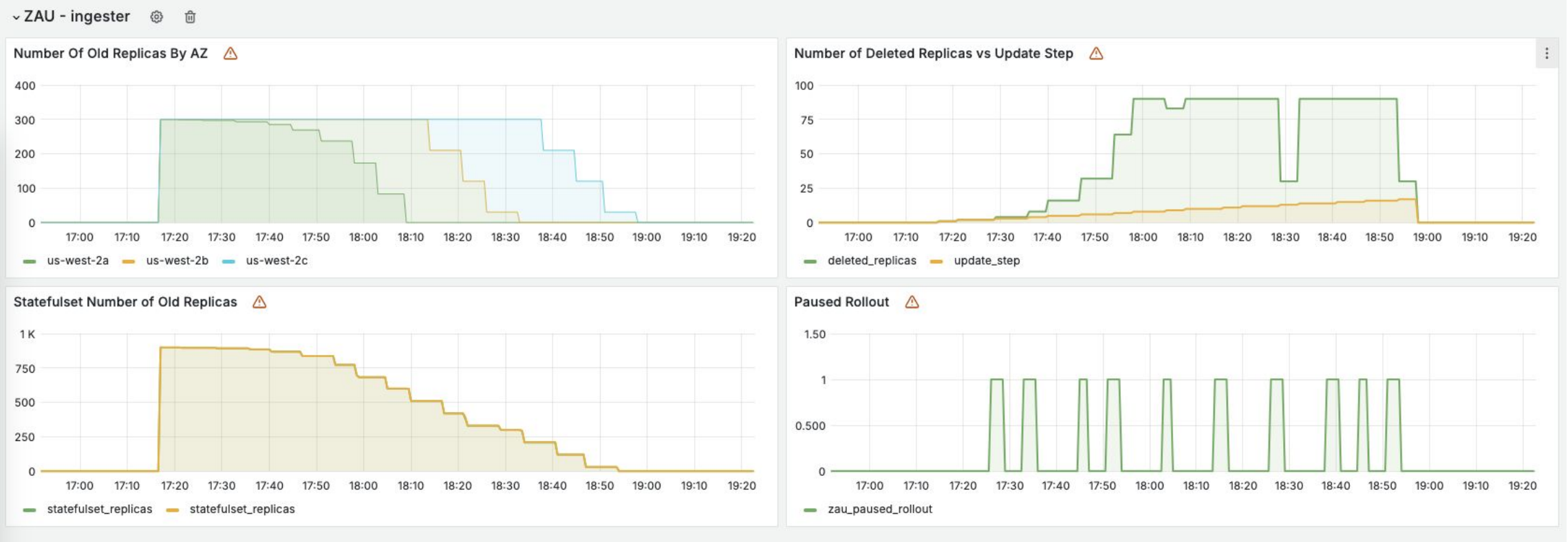
# Zone Aware Controllers

**Allow PDBs to scope disruptions based on topology (e.g., Availability Zone)**

Multiple pods can be disrupted as long as they're in the same zone

**Enables faster deployments for a StatefulSet with pods spread across multiple AZs**

Applying zone-aware logic to safely update pods in parallel within zones




*Link: https://github.com/aws/zone-aware-controllers-for-k8s*

# Host Health Monitoring

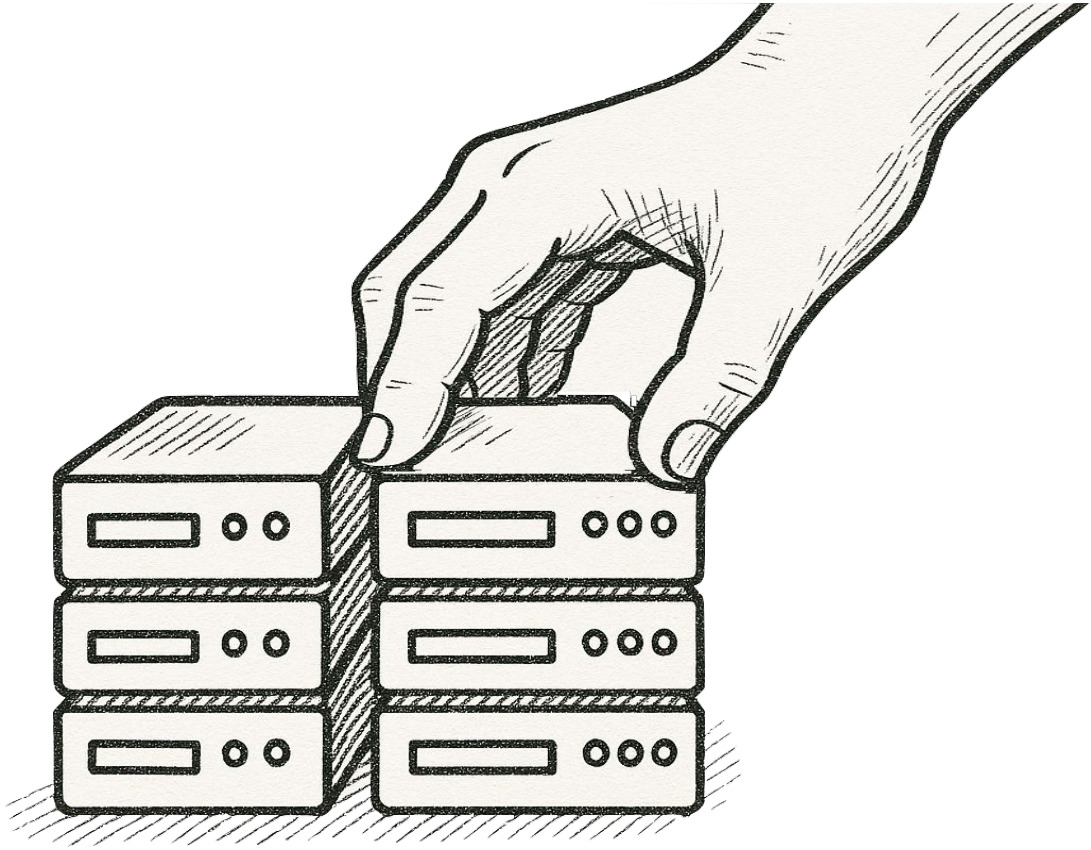
## Automatic resource replacement

Automatically detect and replace individual hosts/volumes having "gray failures"

- Gray Failures: failures that is not perceived by normal health checks.
  Ex: Network impairment, bad credential rotation, disk failure

## Automatic AZ Traffic Shifting

Detects Availability Zone failures and shifts traffic automatically



# Moving to Karpenter

Karpenter

## Dynamic & Fast Scaling

Rapidly adjusts to changing workloads

## Smart Instance Selection

Chooses the best available instance types

## Simplified & Flexible

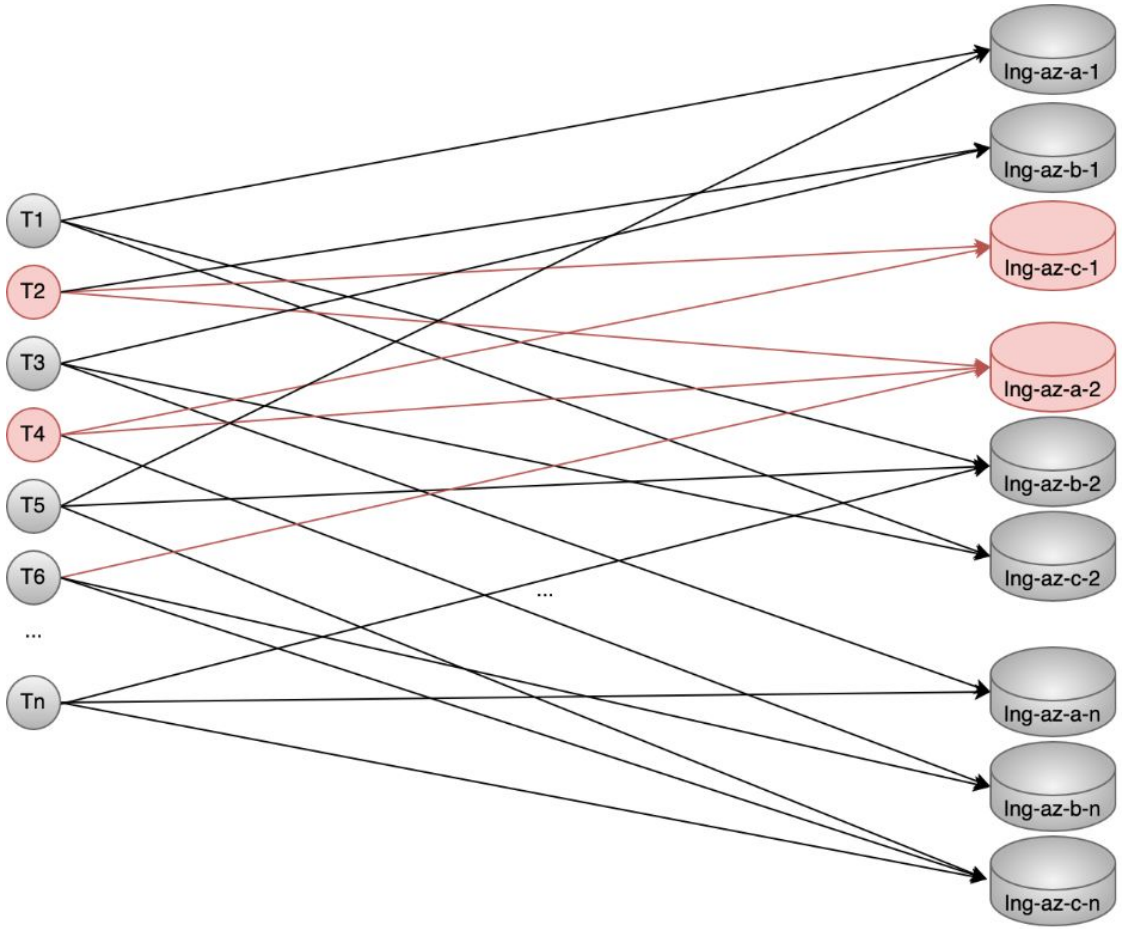
Easy-to-configure, extensible autoscaler

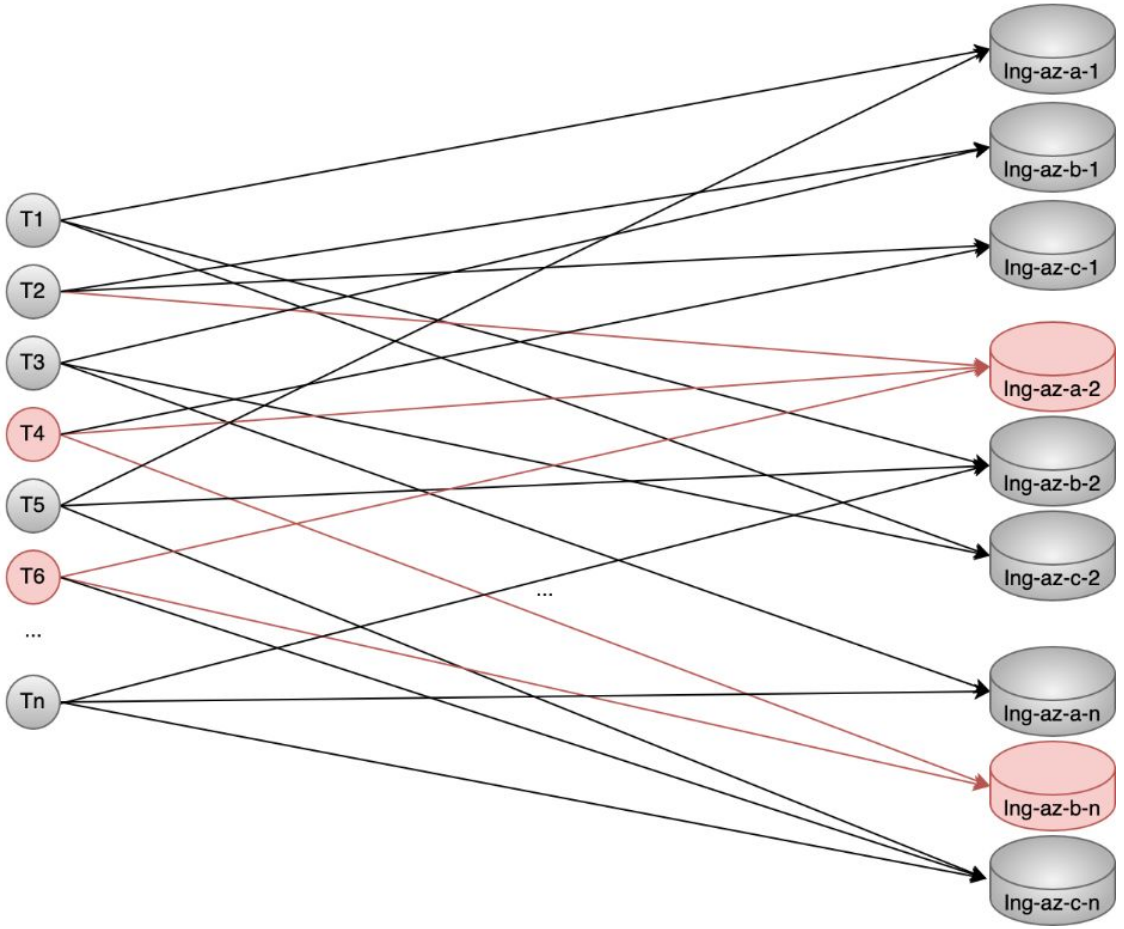
## Drift Detection

Maintains optimal cluster state with drift mode

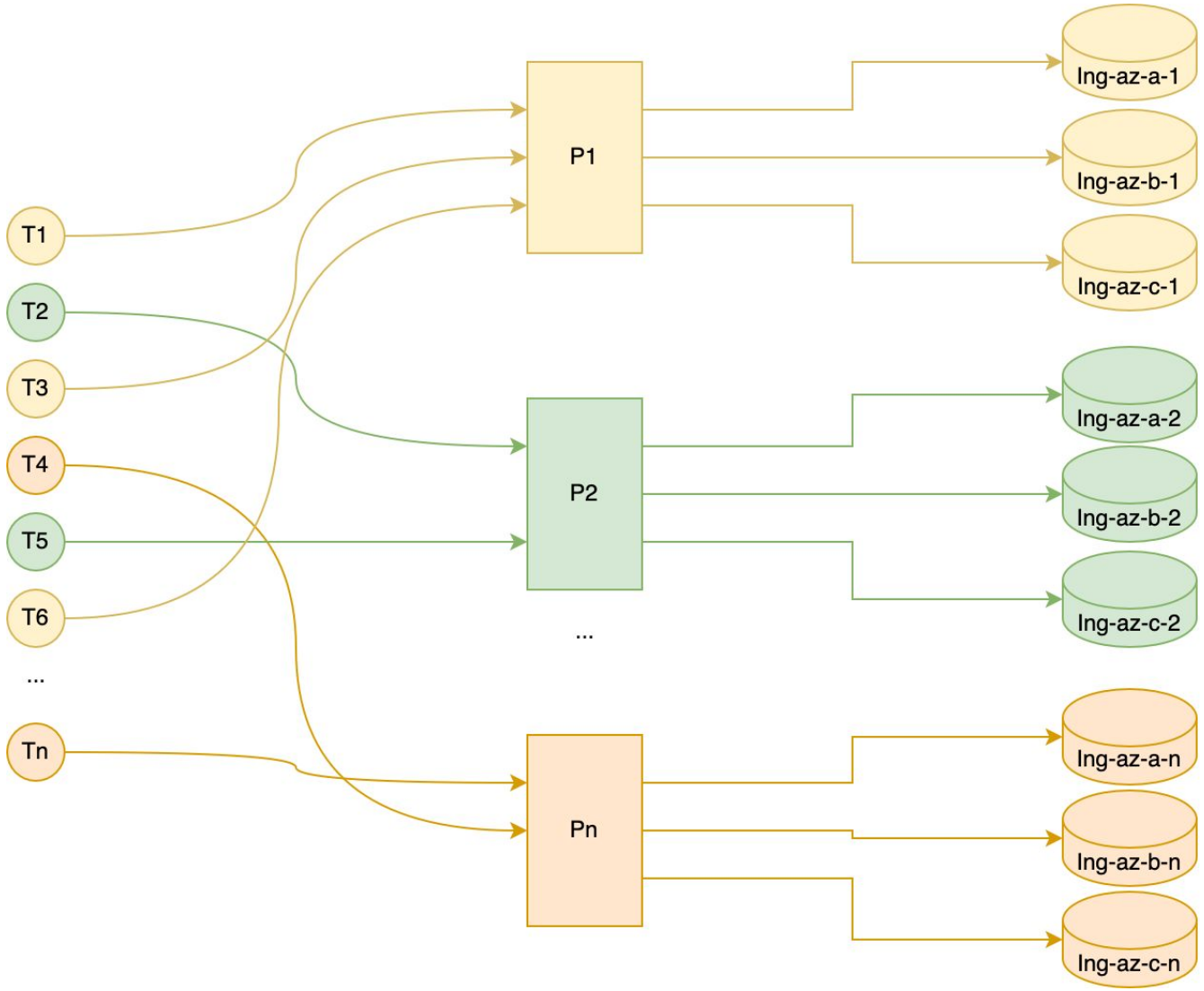
*Link: https://karpenter.sh/*

🚧 Improve availability even when multiple nodes fail across different AZs

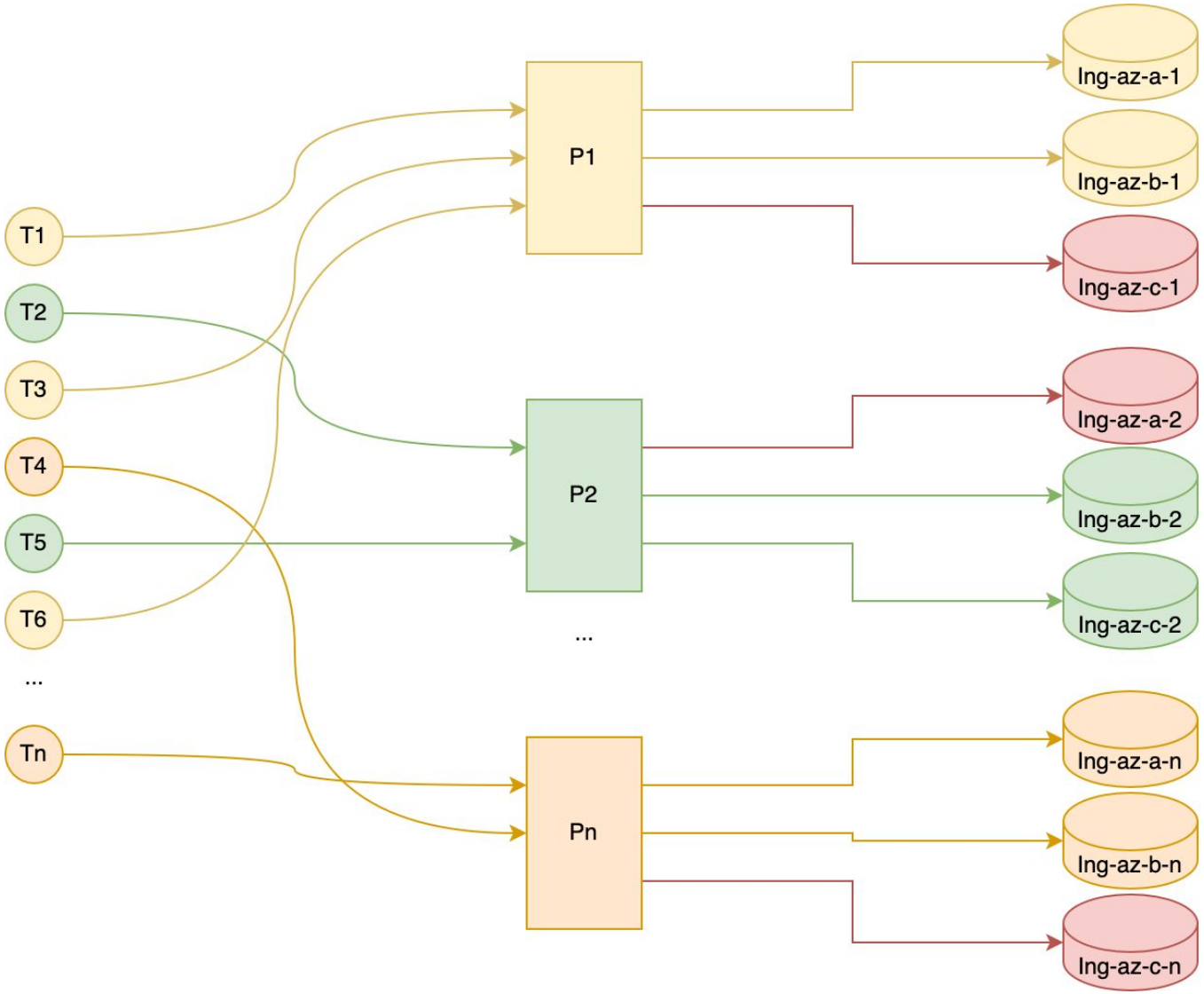


🚧 Improve availability even when multiple nodes fail across different AZs

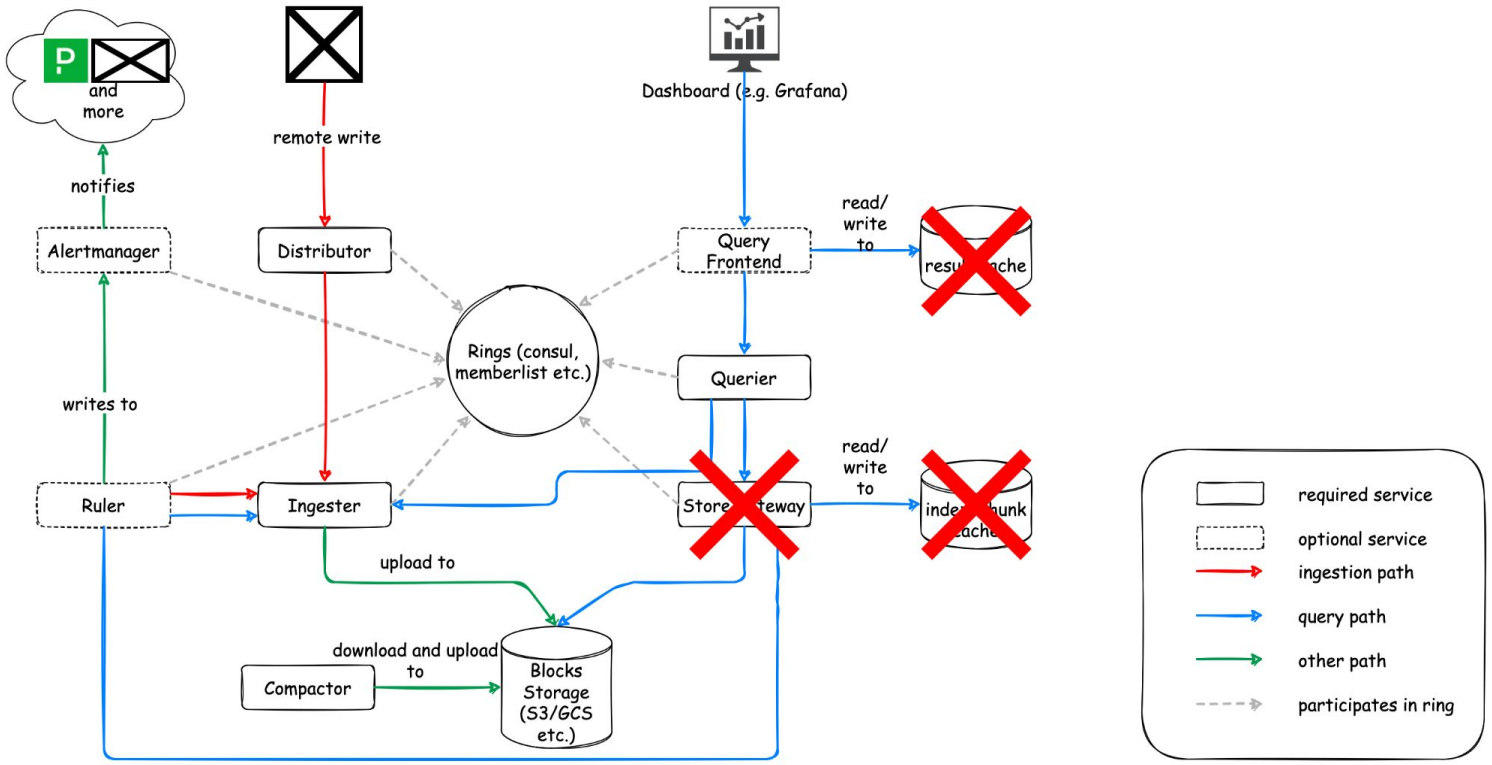


🚧 Improve availability even when multiple nodes fail across different AZs



## 🚧 Use parquet format as long term storage

- Eliminate the need of a stateful service (store gateway) when querying historical data
- Simplify scaling operations
- Reduce infrastructure
- Inspired by Thanos Maintainer talk: https://www.youtube.com/watch?v=V8Y4VuUwg8I






# Thank you