ROTATIONAL

How to Build a Distributed System

(And Should You?)

Patrick Deziel
@PatrickDeziel

Patrick is a distributed systems engineer & machine learning specialist. He was employee #1 at Rotational Labs.

Dr. Rebecca Bilbro
@rebeccabilbro

Rebecca is a machine learning engineer & distributed systems researcher. She's Founder/CTO at Rotational Labs.

Want to see what we're up to? rotational.app

ROTATIONAL

Table of contents

01
Case Study: Signal for Crypto

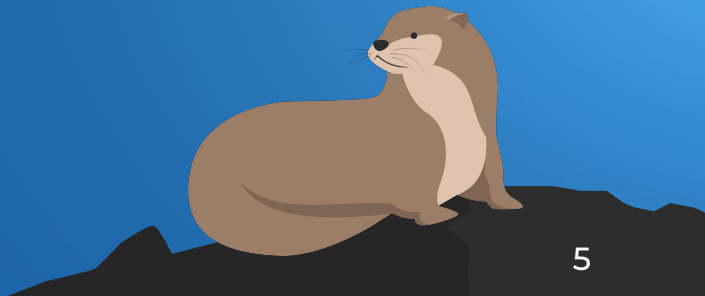
02
What is a distributed system?

03
Lessons learned

04
What's on the horizon?

ROTATIONAL

Case Study: Building the Global Directory Service



The Travel Rule:
Cracking down on crypto

Imagine a communications protocol that facilitates secure, low-latency transfer of PII to prevent financial crime in international transactions.

In addition to being fast, it must be secure, private, and decentralized.

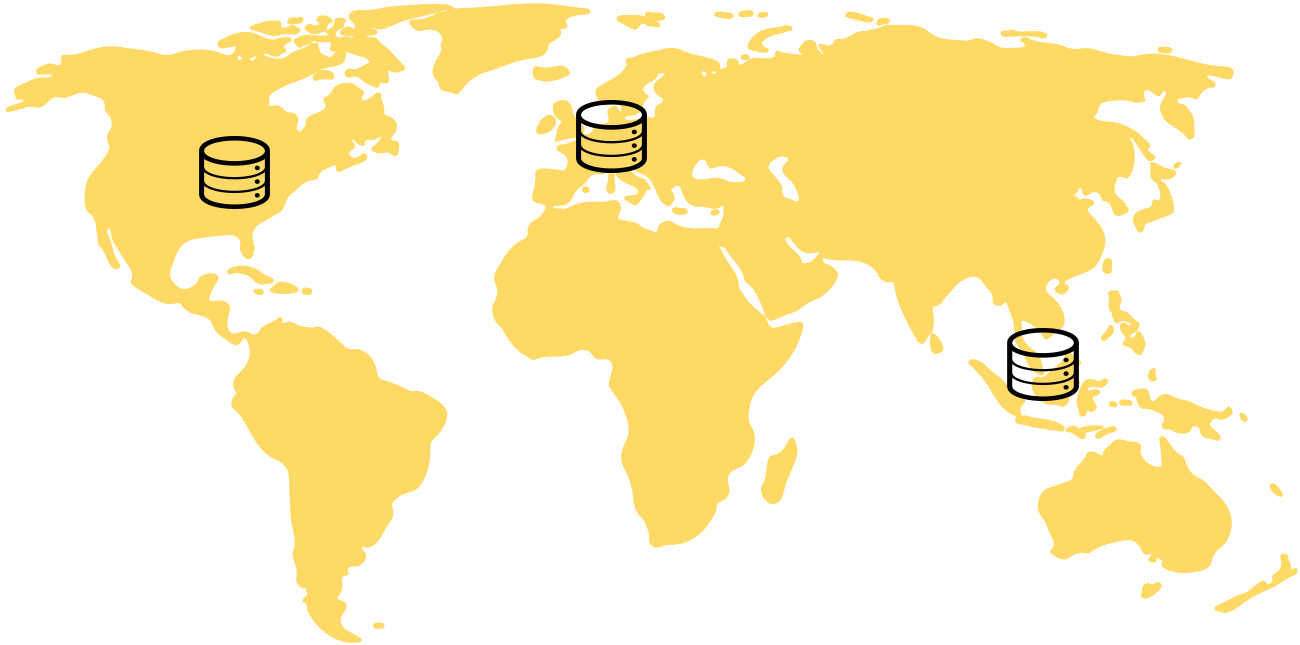
Kind of like **Signal** for crypto.

Well, we built it

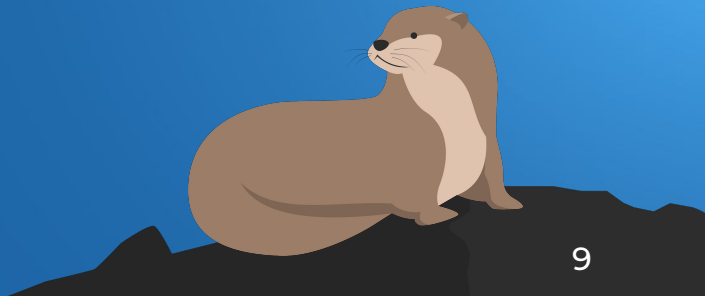
- Commissioned by an international nonprofit working group (TRISA).
- Organizations around the world opt in, enrolling via a web app.
- Leverages public key cryptography; messages transmitted via mutual authentication (mTLS).
- Vast majority of throughput is lookups during transactions. Achieve speed with gRPC.

The Global Directory Service

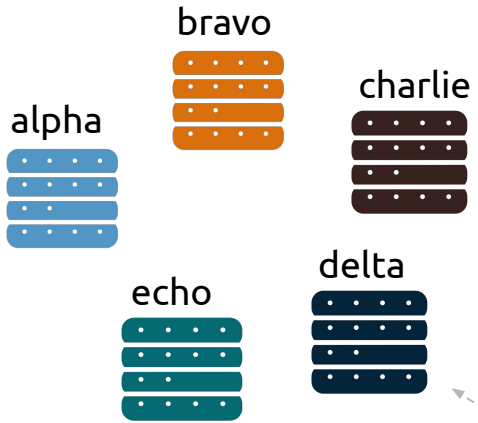
Problem:
Centralized storage is an unfair advantage



What is a distributed system?



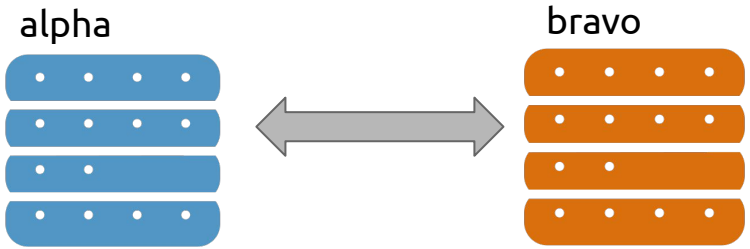
A distributed system is a network of computers working together as to appear to the end-user as a single computer.



The computers are *peers*. Working together, they can make the system more tolerant to failures like earthquakes and outages.

They also make it more *available*, that is, responsive even to many simultaneous client requests.

Each peer has a copy of the database and can respond independently.

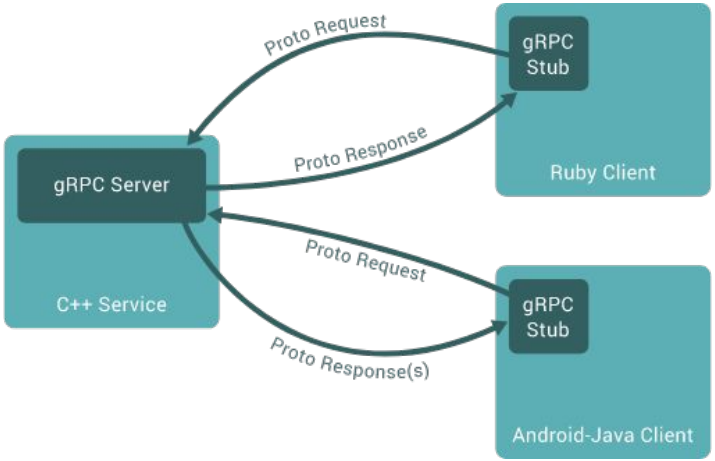


Why do they need to work together?

How do they work together?

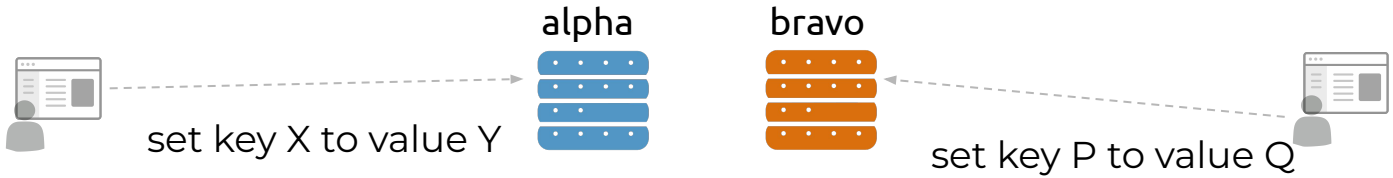
ROTATIONAL

What is gRPC?

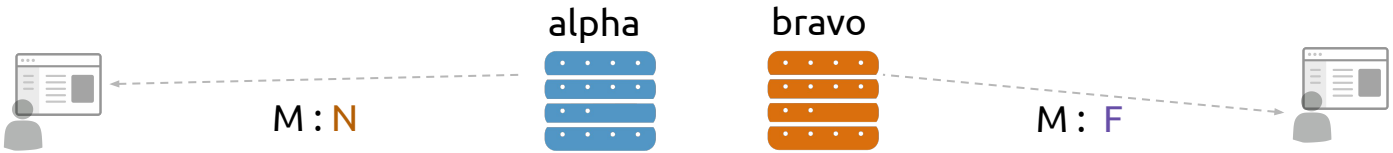


- Google-flavored remote procedure calls
- Define your API in a .proto file
- Compile to your language of choice
- Enjoy!

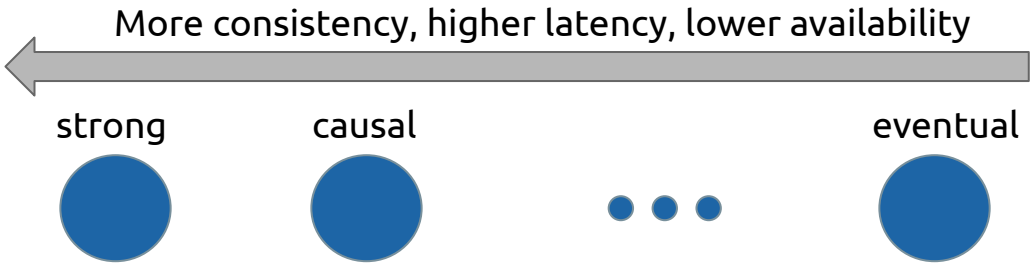
They have to synchronize independent, potentially **concurrent** requests to update data...



... and they may become **inconsistent.**



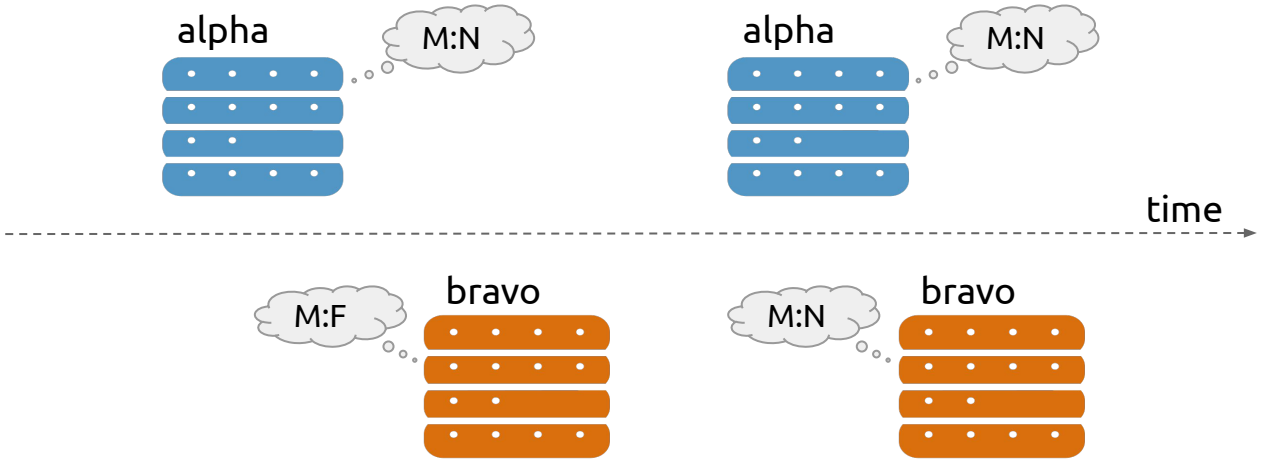
There are several ways to synchronize peers, depending on how ***consistent*** the system needs to be.



The tradeoffs include things like:

- Availability
- Staleness
- Cost

Eventual consistency is a good budget-friendly choice.



Anti-entropy (aka “gossip”) is a mechanism for achieving eventual consistency.

Honu

A library for embedded database replication that adds metadata versioning to traditional in-memory database ops to support replication.

Job: Compute and assign version vectors that encode a commit's provenance and change history.

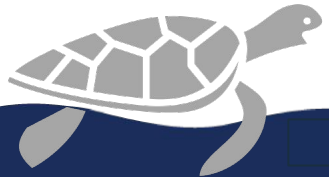
Dependencies: LevelDB and Protobuf.

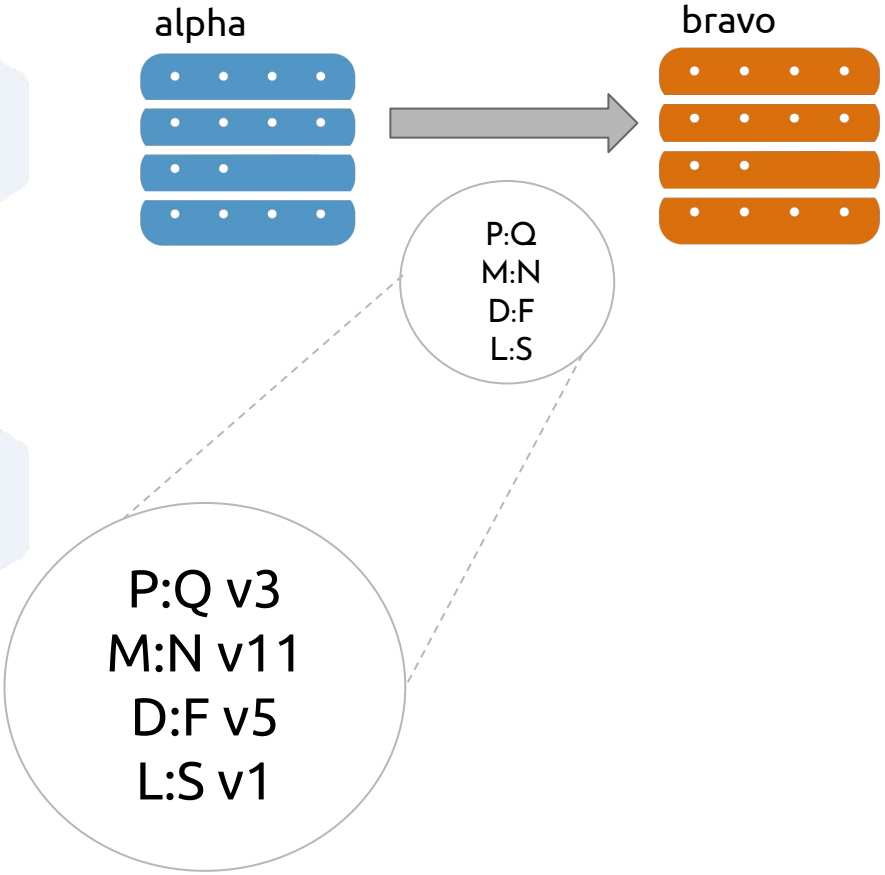
Trtl

A package for geo-distributed bilateral anti-entropy replication to enable our system's data storage and management layer.

Job: Enable peers to perform synchronizations at some configurable interval.

Dependencies: Zerolog and Prometheus.

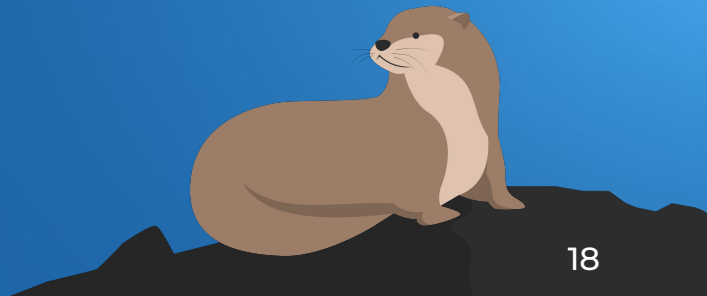




- Whenever a peer updates a value, it increments the object's *version vector*. During sync, alpha shares its version vectors for each object with bravo.
- If bravo sees an object it doesn't have, it will add it locally. If it has a smaller version number for an object, it will perform a local repair.
- What if peers have different values for the same key, but the same version vector? The peer with the lowest Process ID wins! PIDs are unique, so each peer has a different number.

ROTATIONAL

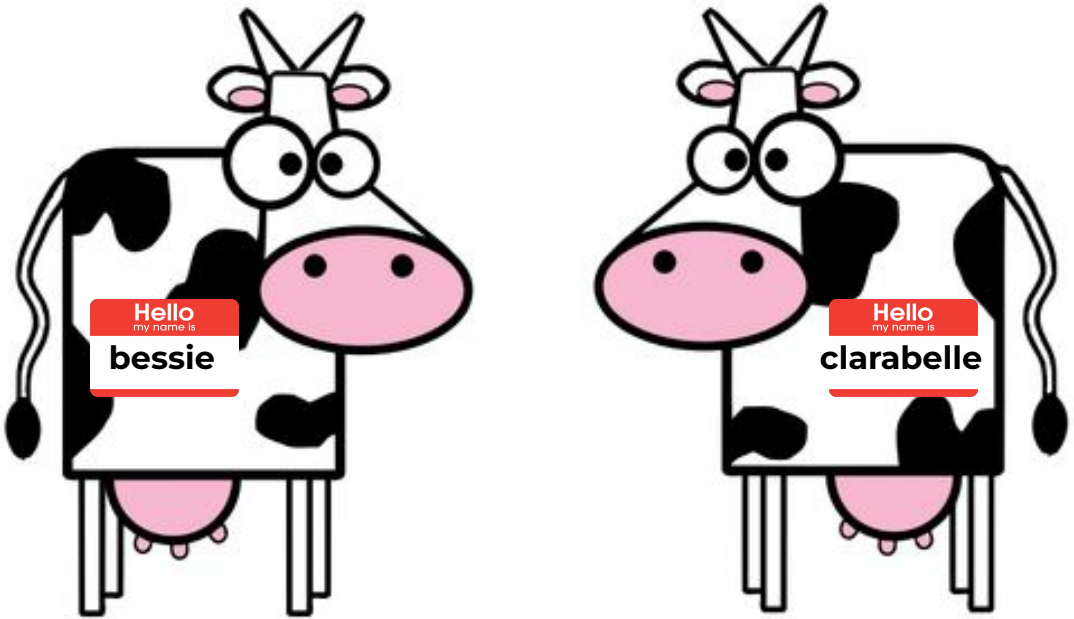
Lessons learned



ROTATIONAL

Naming the Heifers

In order to perform replication, Pods need to be individually addressable



This doesn't work well with a "cattle, not pets" point of view

```yaml
apiVersion: v1
kind: ConfigMap
data:
  replicas.json: |
    {
        "20": {
            "enabled": true,
            "pid": 20,
            "region": "gcp-us-central1",
            "name": "bessie.cattle.us.rotational.dev",
            "gossip_interval": 60000000000,
            "gossip_sigma": 3000000000
        },
        "50": {
            "enabled": true,
            "pid": 50,
            "region": "gcp-europe-west3",
            "name": "ida.cattle.de.rotational.dev",
            "gossip_interval": 60000000000,
            "gossip_sigma": 3000000000
        },
        "80": {
            "enabled": true,
            "pid": 80,
            "region": "gcp-asia-southeast1",
            "name": "clarabelle.cattle.sg.rotational.dev",
            "gossip_interval": 60000000000,
            "gossip_sigma": 3000000000
        },
    }
```

ROTATIONAL

```yaml
apiVersion: apps/v1
kind: StatefulSet
spec:
  template:
    ...
    spec:
      volumes:
      - name: replicas-configmap
        configMap:
          name: replicas-configmap
```

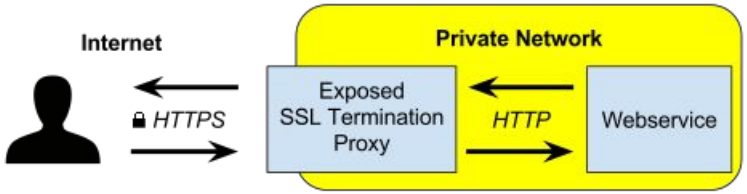
ConfigMap

Google Load Balancer

Google cloud load balancer supports multi-region which fits with our globally-oriented users, but:

- We want clients to be able to connect to endpoints over gRPC which CLB is not built for
- Protocols are restricted (SSH, VPN, h2c, gRPC, native TCP; e.g. everything but HTTPS)
- They require TLS termination at the load balancer (so no mTLS)

What is TLS Termination?

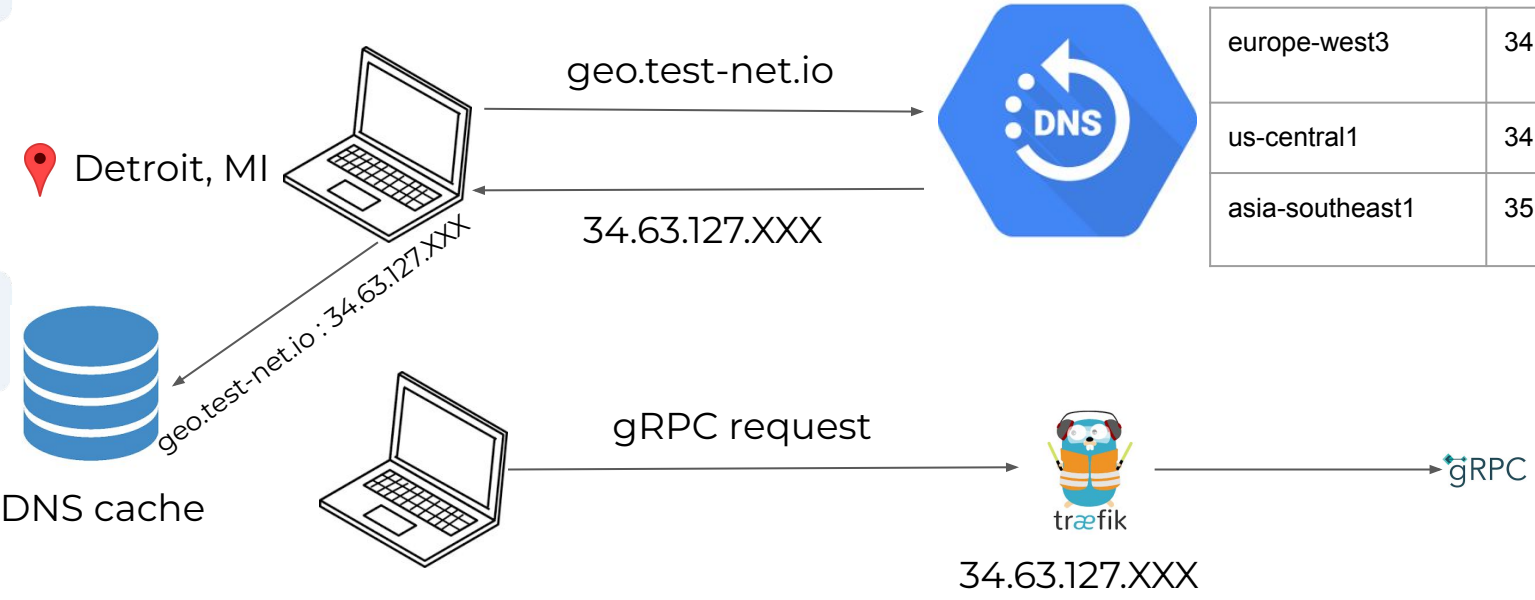


Some proxy servers (e.g. Google CLB) interrupt secure communication (SSL/TLS) by decrypting traffic in transit

This is sometimes desirable!

We still need mTLS to securely interact with our gRPC endpoint services (Trtl)

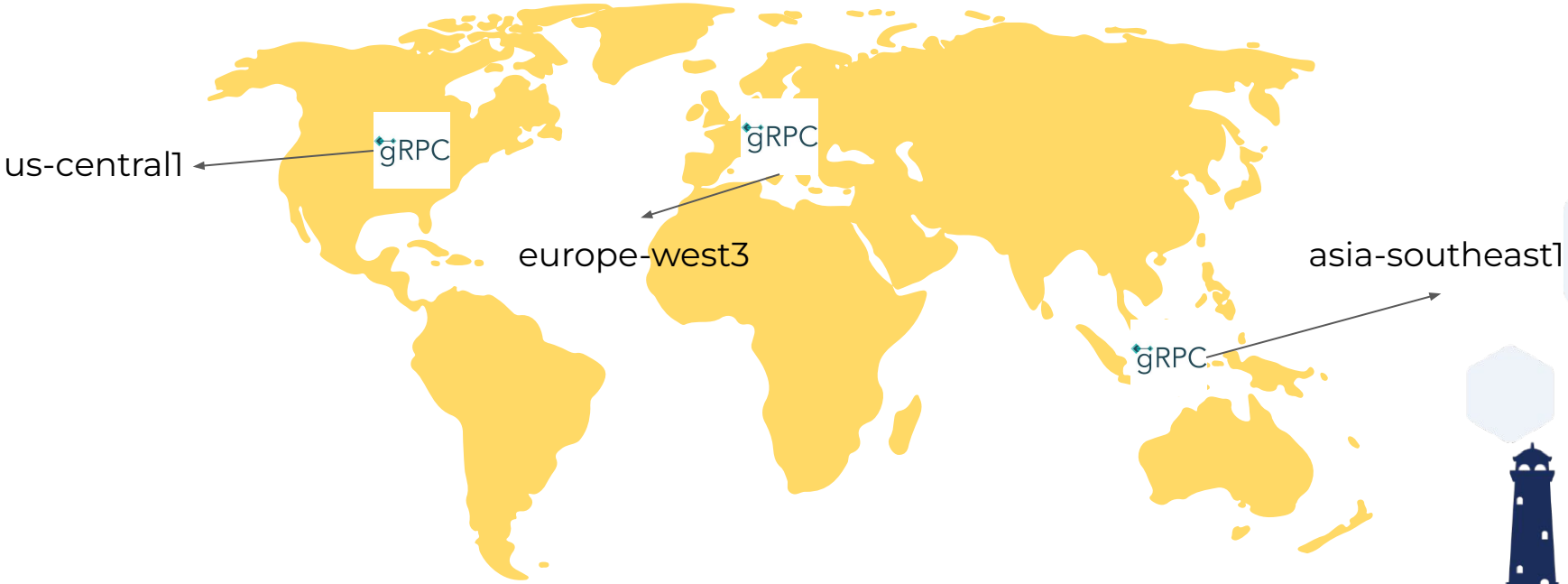
Routing at the DNS Layer



europe-west3	34.167.23.XXX
us-central1	34.63.127.XXX
asia-southeast1	35.236.103.XXX

GeoPing

A gRPC service that knows where it is

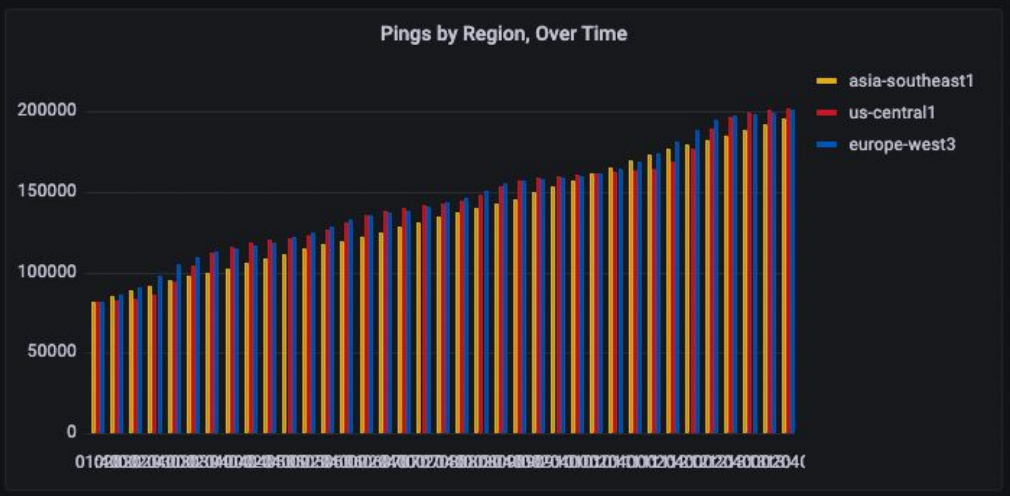


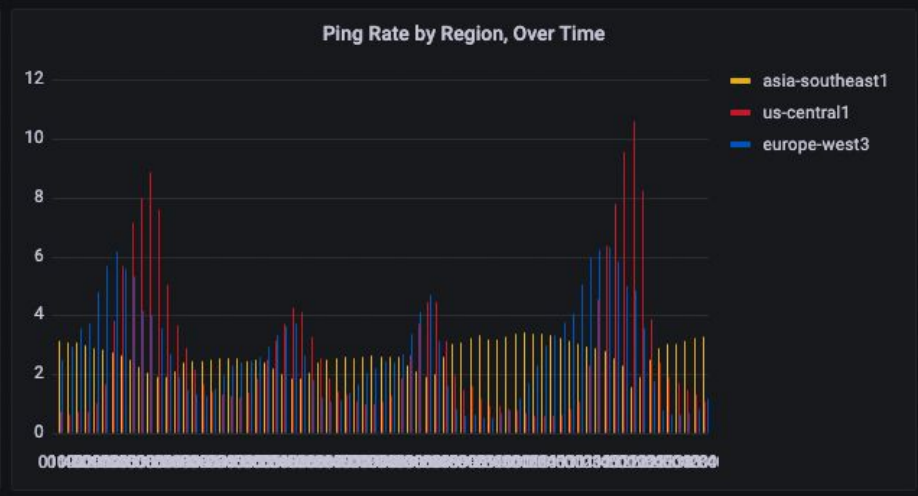
GeoPing

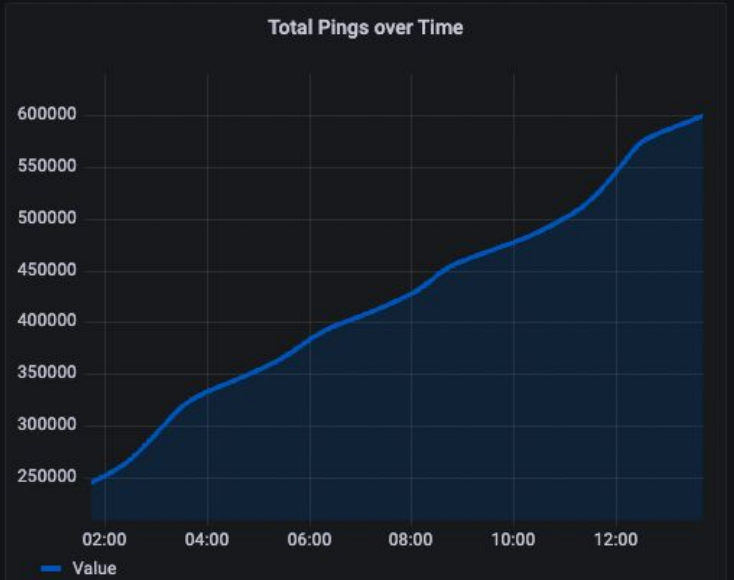
```
service GeoPing {
    rpc Status(RegionRequest) returns (ServerStatusReply) {}
}

message RegionRequest {
    // The client's region - optional
    string region = 1;
}

message ServerStatusReply {
    // Region of the server
    string region = 1;

    // Version of the server
    string version = 2;

    // Uptime of the server
    string uptime = 3;

    // Status message
    string status = 4;
 }
```


-zsh
"status": "ok"
}
patrick@Patricks-MacBook-Pro geoping % clear
patrick@Patricks-MacBook-Pro geoping % geoip ping -u geo.test-net.io:443
{
"region": "us-central1",
"version": "0.1",
"uptime": "616h22m17.012357708s",
"status": "ok"
}
patrick@Patricks-MacBook-Pro geoping % geoip ping -u geo.test-net.io:443
{
"region": "europe-west3",
"version": "0.1",
"uptime": "580h35m17.9309685s",
"status": "ok"
}
patrick@Patricks-MacBook-Pro geoping % geoip ping -u geo.test-net.io:443
{
"region": "europe-west3",
"version": "0.1",
"uptime": "580h36m2.485867081s",
"status": "ok"
}
patrick@Patricks-MacBook-Pro geoping % sudo dscacheutil -flushcache; sudo killall -HUP mDNSResponder
Password:
patrick@Patricks-MacBook-Pro geoping % geoip ping -u geo.test-net.io:443
{
"region": "asia-southeast1",
"version": "0.1",
"uptime": "623h53m48.837175531s",
"status": "ok"
}
patrick@Patricks-MacBook-Pro geoping %


CONNECTED
Singapore #462
Help
Search
Iceland
Indonesia
Ireland
Israel
Italy
Japan
Latvia
Lithuania
Luxembourg
Malaysia
Mexico
Moldova
Netherlands
New Zealand
North Macedonia
Norway
Poland
Portugal
Romania
Serbia
Singapore
Slovakia
Slovenia
South Africa
South Korea
Spain
Sweden
Switzerland
Taiwan
Thailand
Turkey
Ukraine
United Kingdom
United States
Vietnam

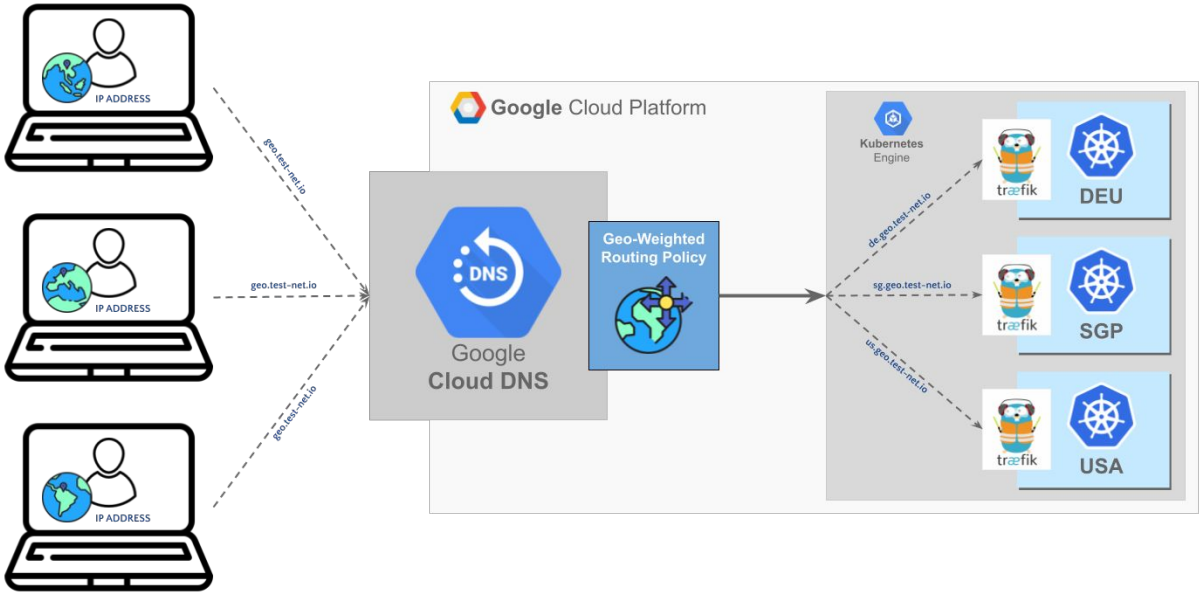

Pings by Region, Over Time
asia-southeast1
us-central1
europe-west3
200000
150000
100000
50000
0


Ping Rate by Region, Over Time
asia-southeast1
us-central1
europe-west3
12
10
8
6
4
2
0


Total Pings over Time
600000
550000
500000
450000
400000
350000
300000
250000
02:00
04:00
06:00
08:00
10:00
12:00
Value


GeoPings
NORTH AMERICA
ASIA
AFRICA
SOUTH AMERICA
OCEANIA
Pacific Ocean
Indian Ocean

Geo-Weighted DNS Routing

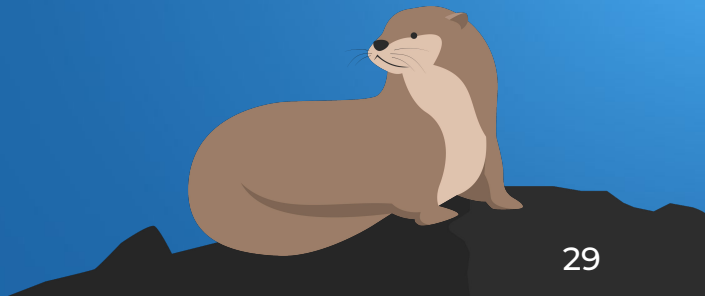


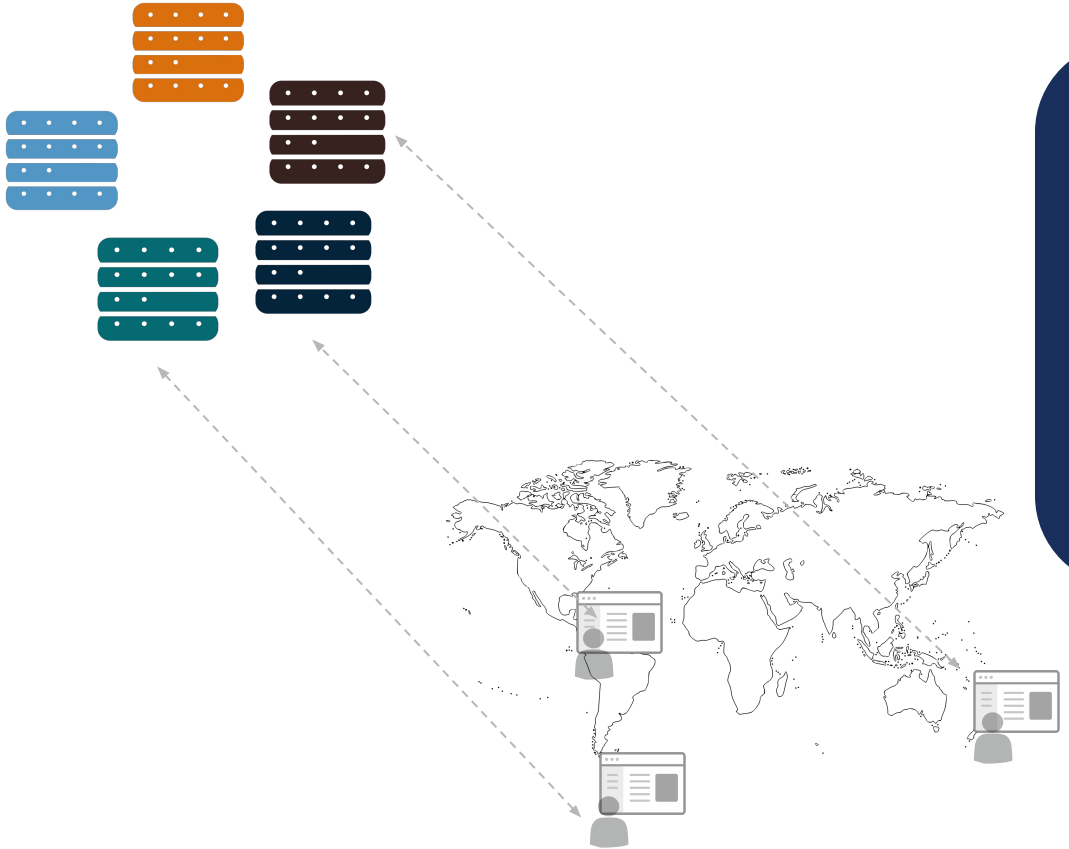
We set up our own ingress, but then we needed geo-weighted DNS routing (using Cloud DNS and Route53). Some silver linings:

- Simplified our GDPR compliance, i18n, and other app-specific details
- Enabled multi-cluster deployments
- We are cloud native, but still in the driver's seat

ROTATIONAL

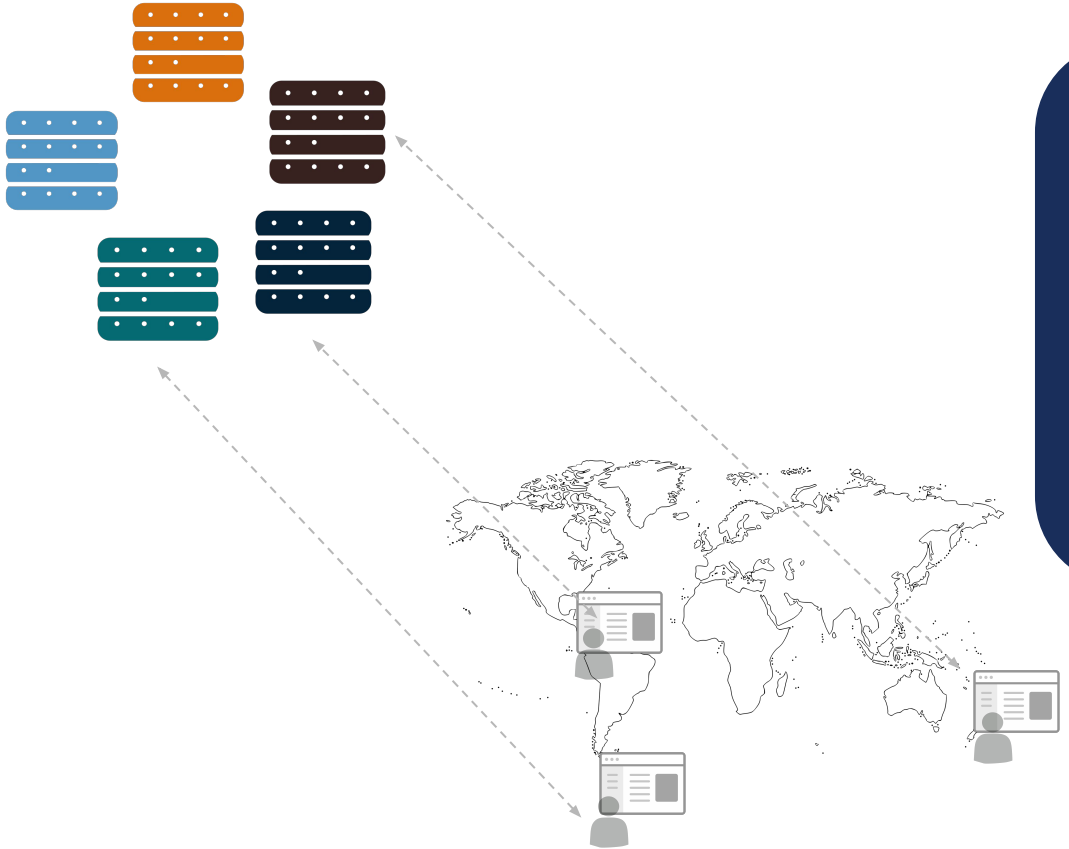
The future of distributed systems





Distributed System:

A network of computers working together as to appear to the end-user as a single computer.



This is how the “hyperscalers” achieved their scale (add hardware/devOps). YMMV

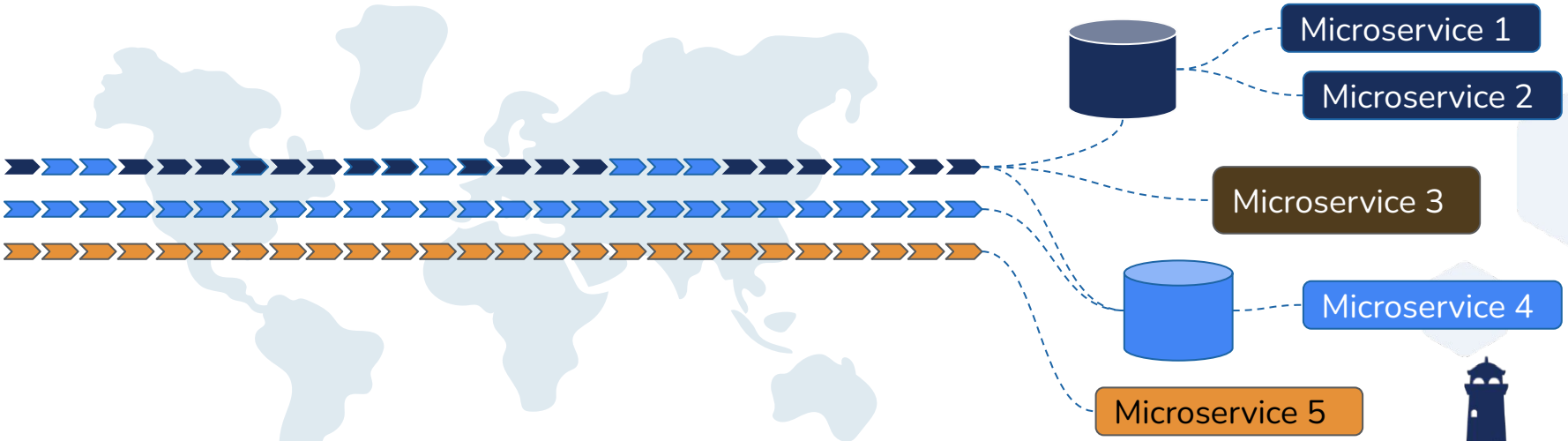
Technically still true, but rooted in older ideas about

- who is part of the Internet
- what constitutes a server

It’s a very different world now!

Distributed System (New Definition):

A flow of events across time and space.

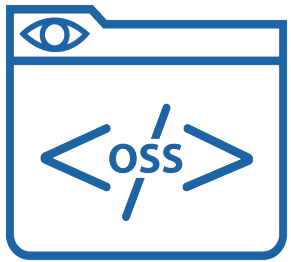


The tools* for the next generation of DistSys may look familiar...

- Message Queues
- Pub/Sub & Event Queues
- Steam Processing

*These won't replace databases 😉They're the wedges we need to help us connect our clusters together.

New Features on the Horizon



Persistence by Default

- Most tools require configuration to turn persistence on.
- Persisting the log slows down throughput and increases latency.
- However, having it turned on by default means features like automatic disaster recovery and time travel.

Geographic Event Encoding

- For the most part, cloud service providers assume you want geography abstracted away from you.
- What if all events encoded geographic metadata?
- This could enable geographic awareness in the app (e.g. better UX), simplify compliance with GDPR, etc.

Total Ordering by Default

- Most tools can guarantee total ordering only for brokers on the same node.
- Total ordering requires consensus, which can impede performance.
- However, with total ordering we can provide new features that enable more equitable global engagement in our applications.

Automatic Reconfiguration

- Success of first-gen DistSys relied on $$$ hardware and huge devOps teams.
- We need a new way to multi-cluster that's easier to maintain and makes more sense for smaller teams.
- New eventing tools can leverage more dynamic consensus algorithms that respond to the system and do automatic reconfiguration either to scale up or to manage costs.

ROTATIONAL

Thank you!