
Practical Microservices

Thijs Schreijer

Kong Inc, Solutions architect

@thijsschreijer, @thekonginc

https://konghq.com

We are on an architectural journey

We are on a journey

Monolith

Services

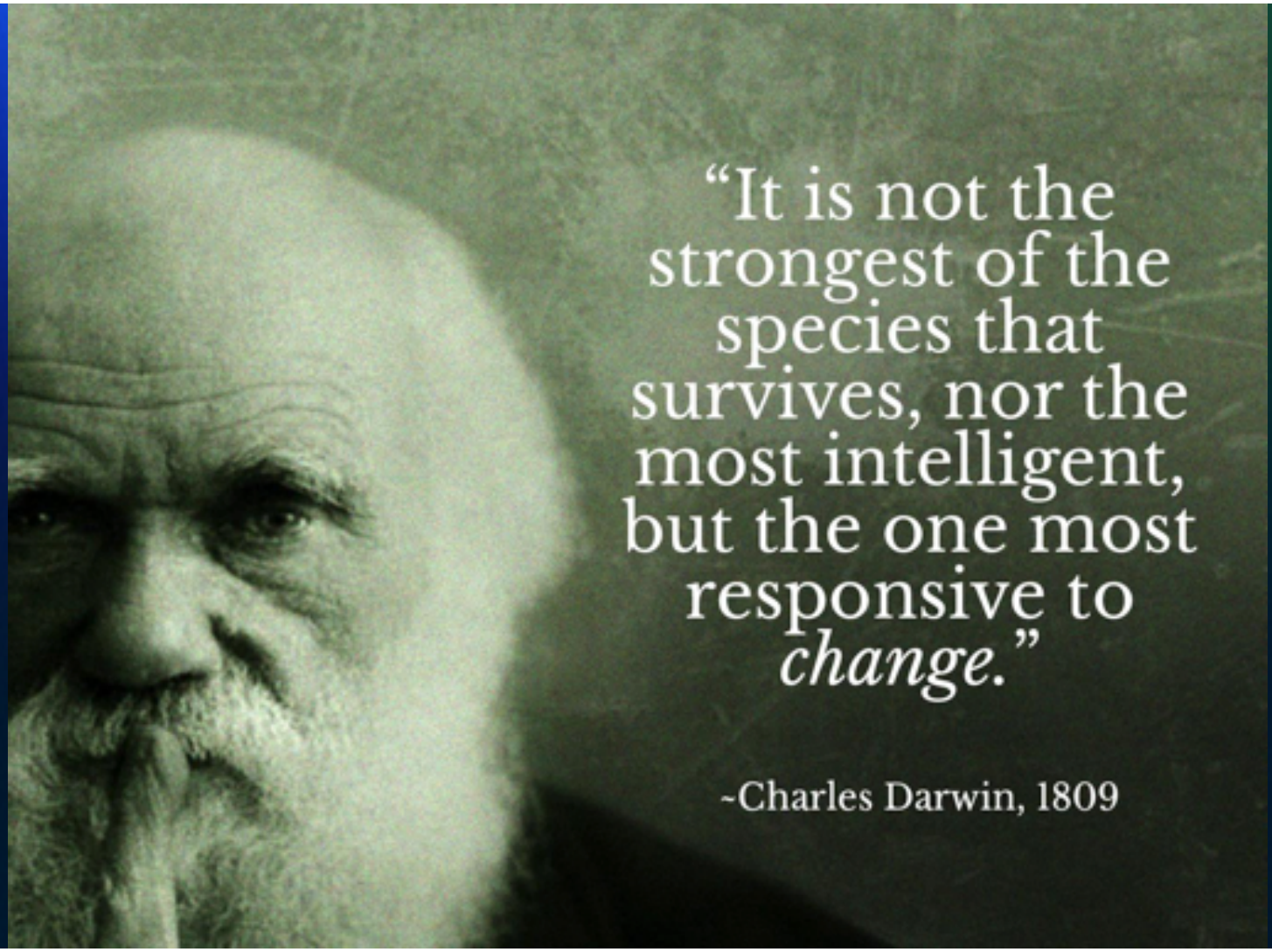
Microservices/
Service Mesh

Serverless/FaaS

Emerging Patterns

Why are architectures changing?

To scale our business


"It is not the strongest of the species that survives, nor the most intelligent, but the one most responsive to *change*."
~Charles Darwin, 1809

3 Trends

1

Information in **Flight**

180 ZB

Data created annually by 2025*

25%

Data will be real-time*

Source: https://www.seagate.com/www-content/our-story/trends/files/Seagate-WP-DataAge2025-March-2017.pdf

2

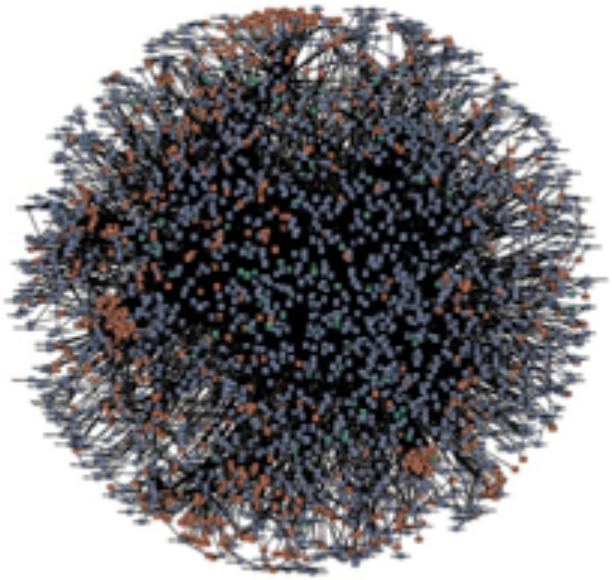
Cloud-native first, **Hybrid always**

3

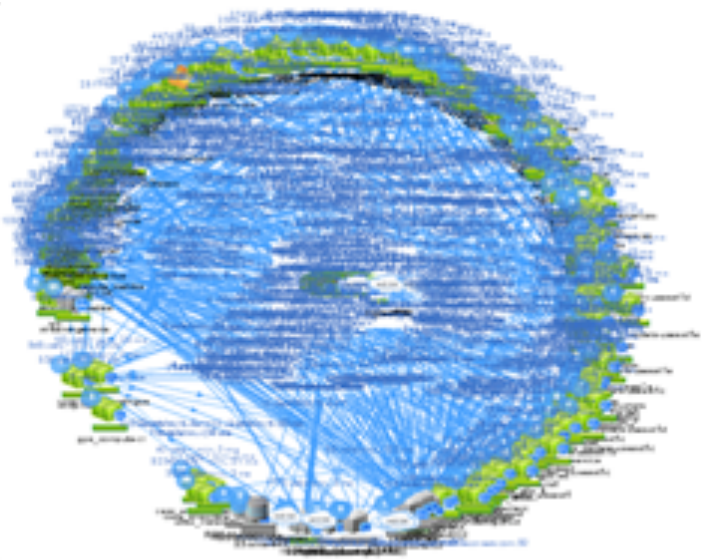
Service **Explosion**
From one to thousands

Should we transition to Microservices?

Microservices Premium



amazon.com



NETFLIX

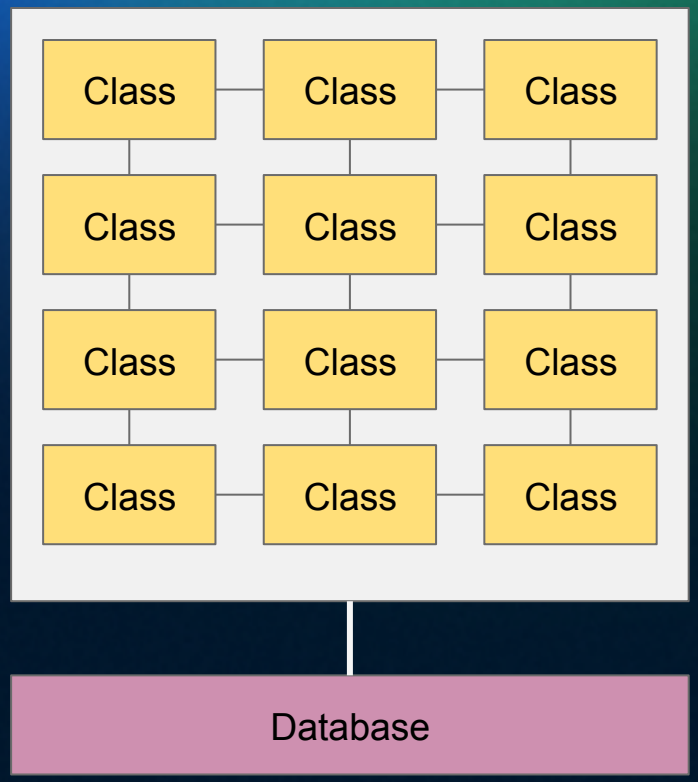
Let’s have a look at what it takes

3 Tracks

Technical / Operational / Organizational

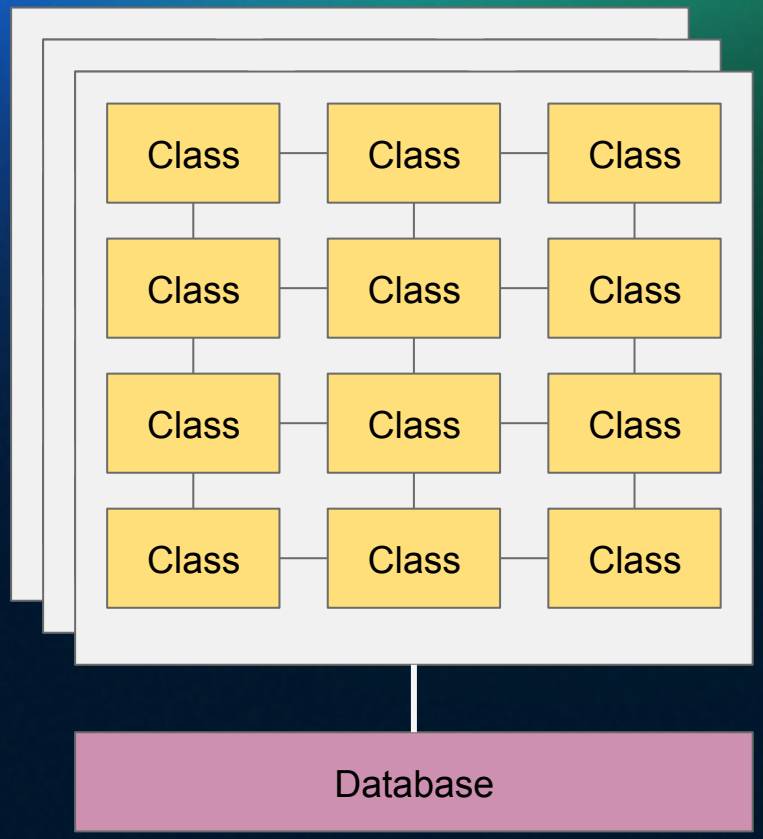
Monolith

Lots of **function calls**
across different objects



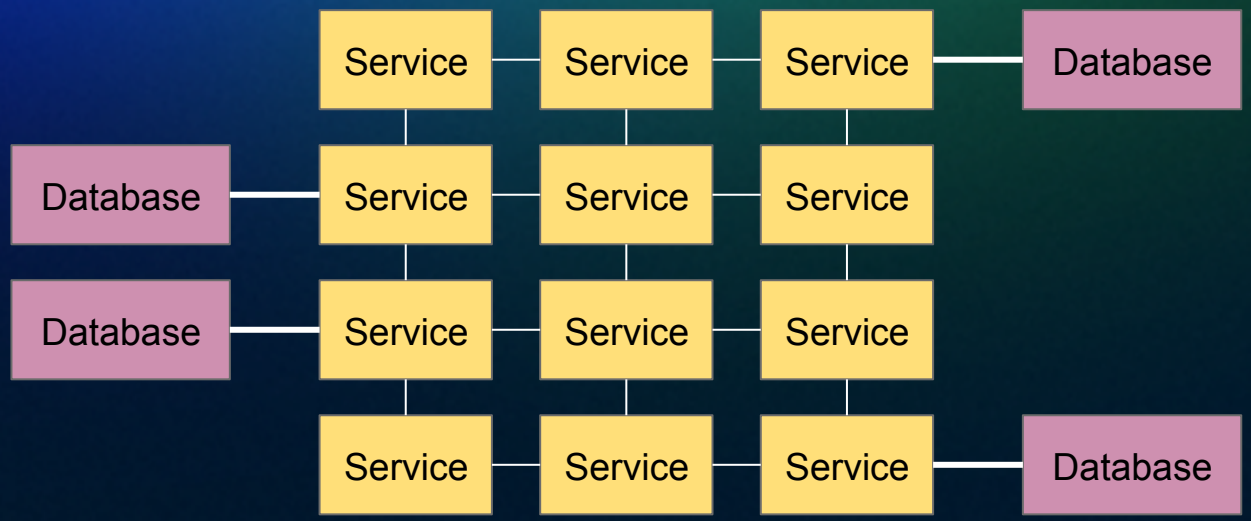
Monolith

Lots of **function calls**
across different objects



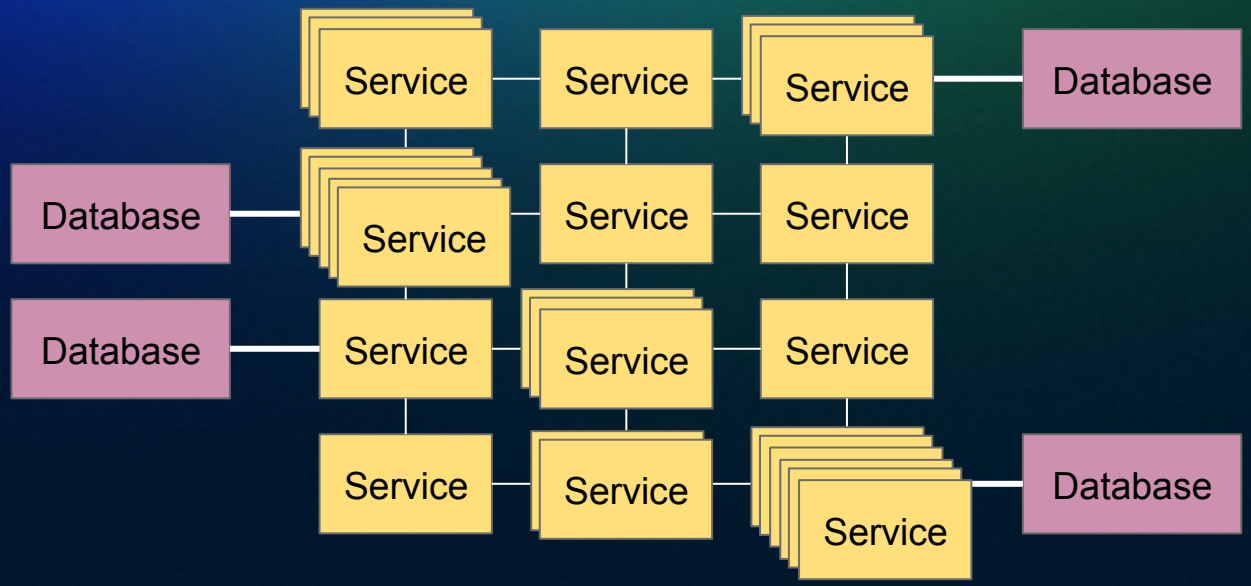
Microservices

Lots of **network calls** across different services



Microservices

Lots of **network calls** across different services



In monoliths we have objects , interfaces, and function calls.

In microservices we have services , interfaces, and network calls.

Little Hints:

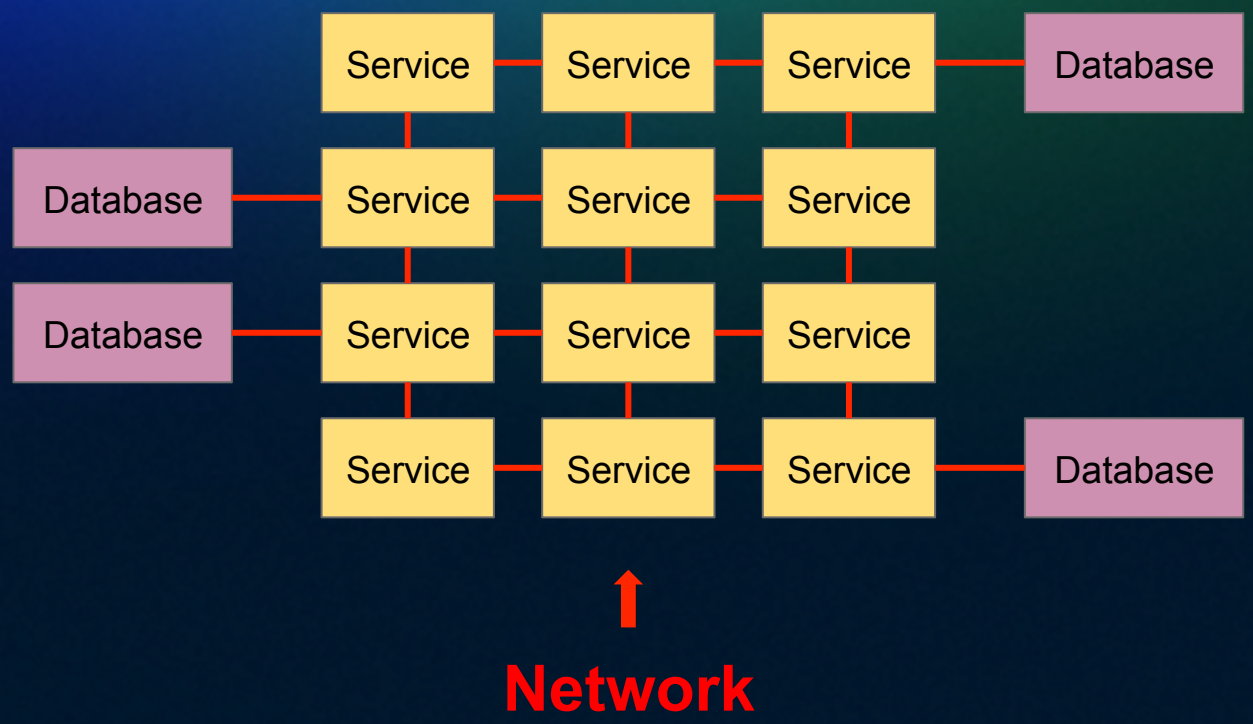
Why do our mobile phone calls drop?

What are the most frequent causes for bad home internet?

Why are our downloads usually failing?

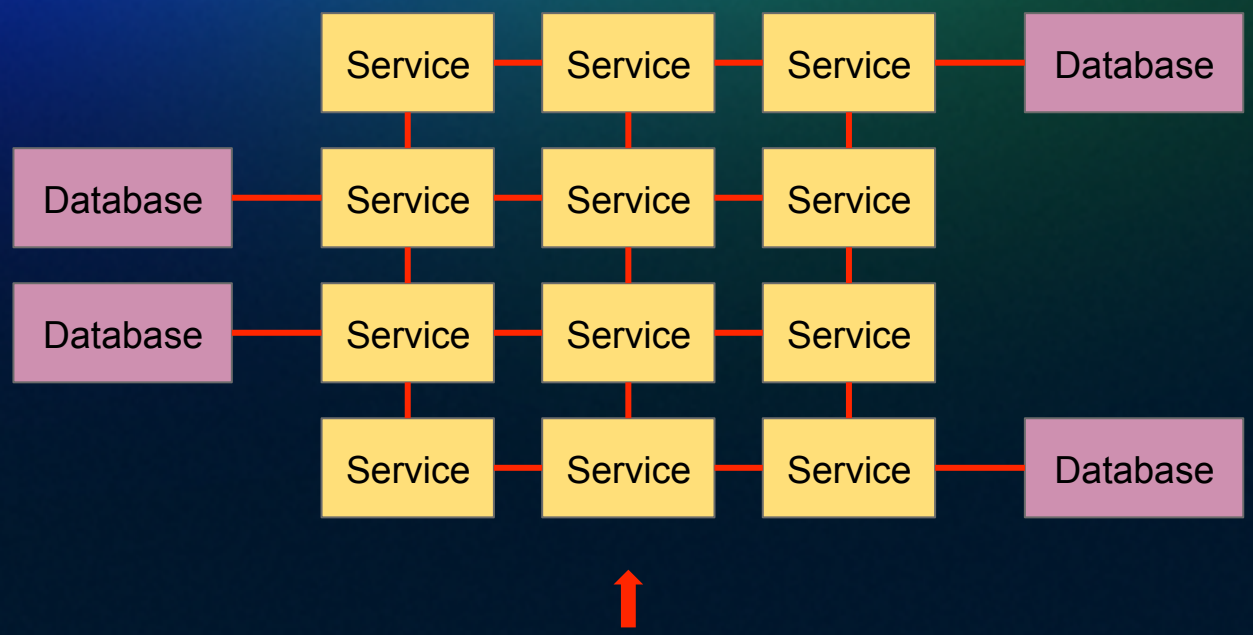
Microservices

Lots of **network calls** across different services



Microservices

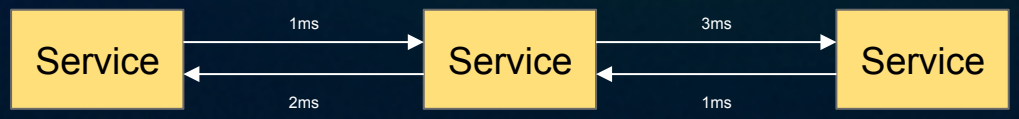
Lots of **network calls** across different services



Typical network problems: latency, security, routing, error handling, observability

Latency

Cannot be ignored anymore. It compounds over calls.



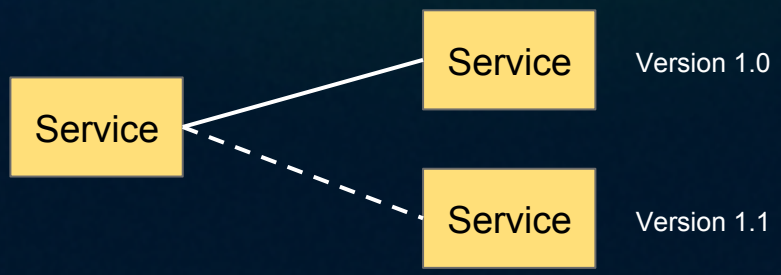
Total network latency: **7ms**

Security

Mutual TLS to protect service-to-service communication and used an authentication scheme for the service.

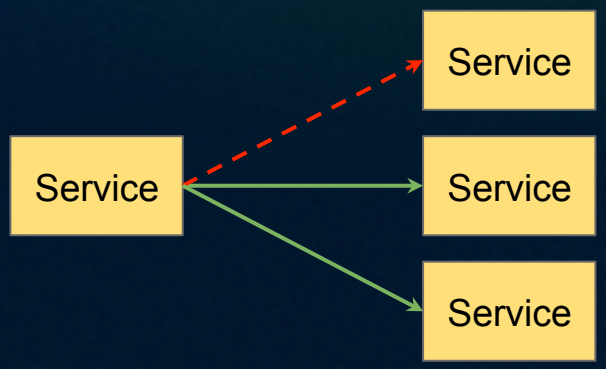
Routing

Intelligently route traffic across different services, different versions, different regions/DCs, and so on.



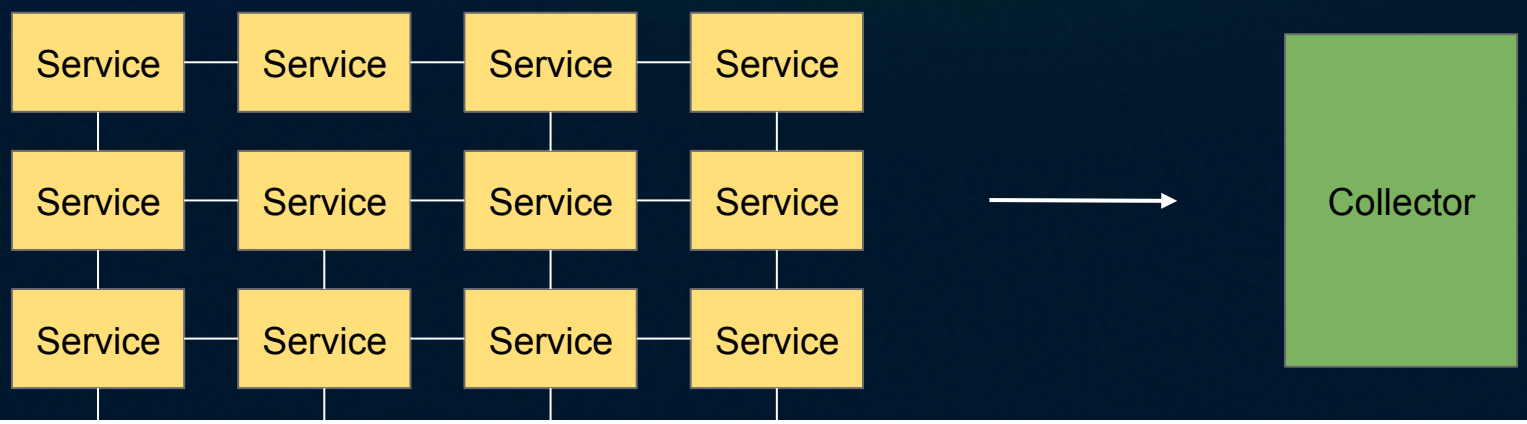
Error Handling / Resiliency

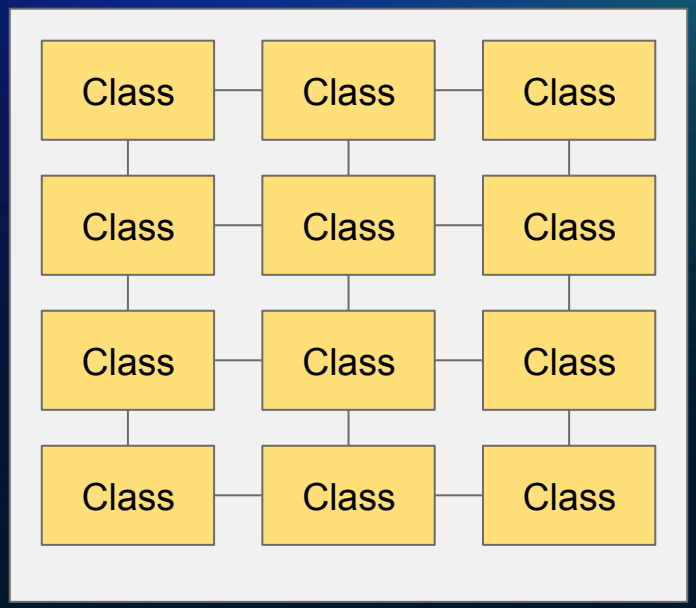
Retries, Timeout Handling, Health-Checks, Circuit Breakers



Observability

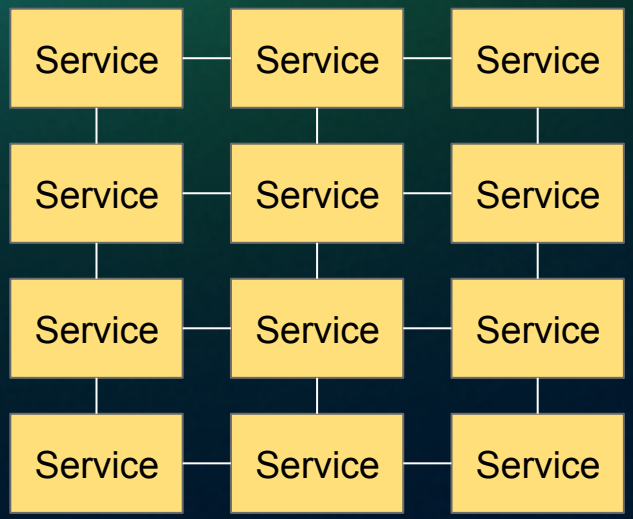
Retrieving traces, latency and analytics data across the entire system.




Class
Class
Class
Class
Class
Class
Class
Class
Class
Class
Class
Class

O(1)

vs


Service
Service
Service
Service
Service
Service
Service
Service
Service
Service
Service
Service

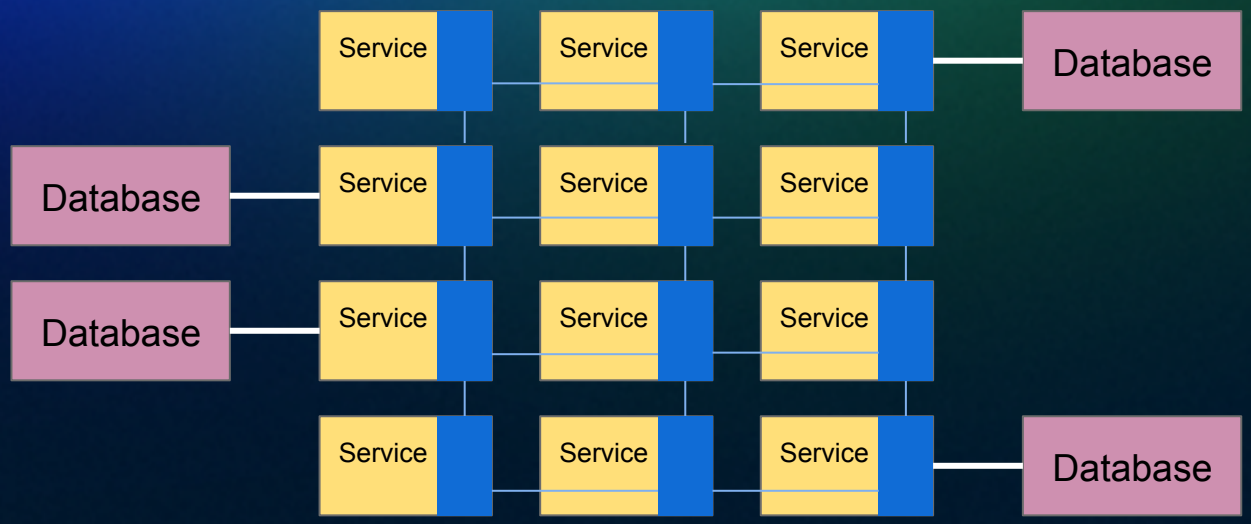
O(n)

Many ways to make the network reliable.

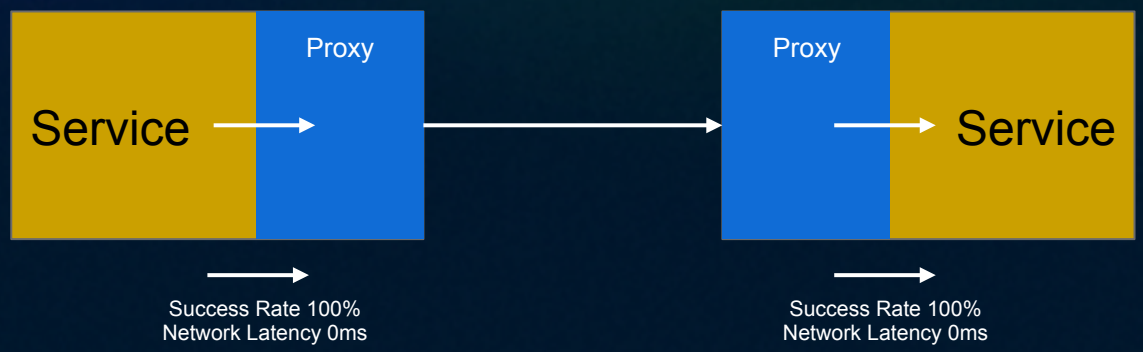
One of them is the Service Mesh pattern.

Service Mesh

Lots of **network calls** across different services through a **decentralized proxy**



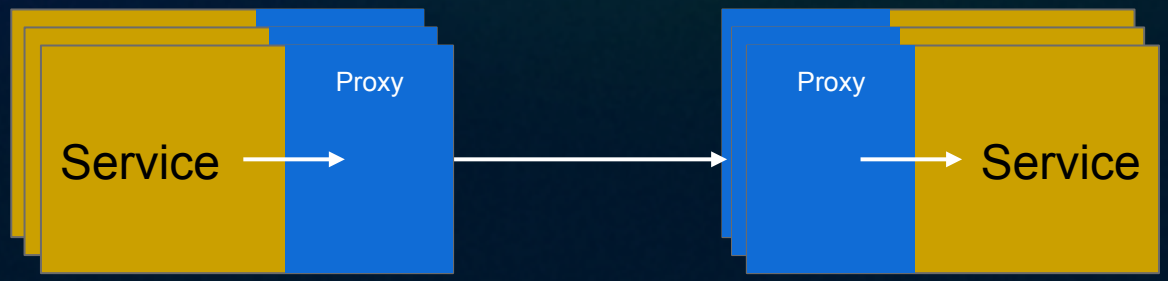
The biggest assumption is that the network latency between the service and the **local** proxy is negligible

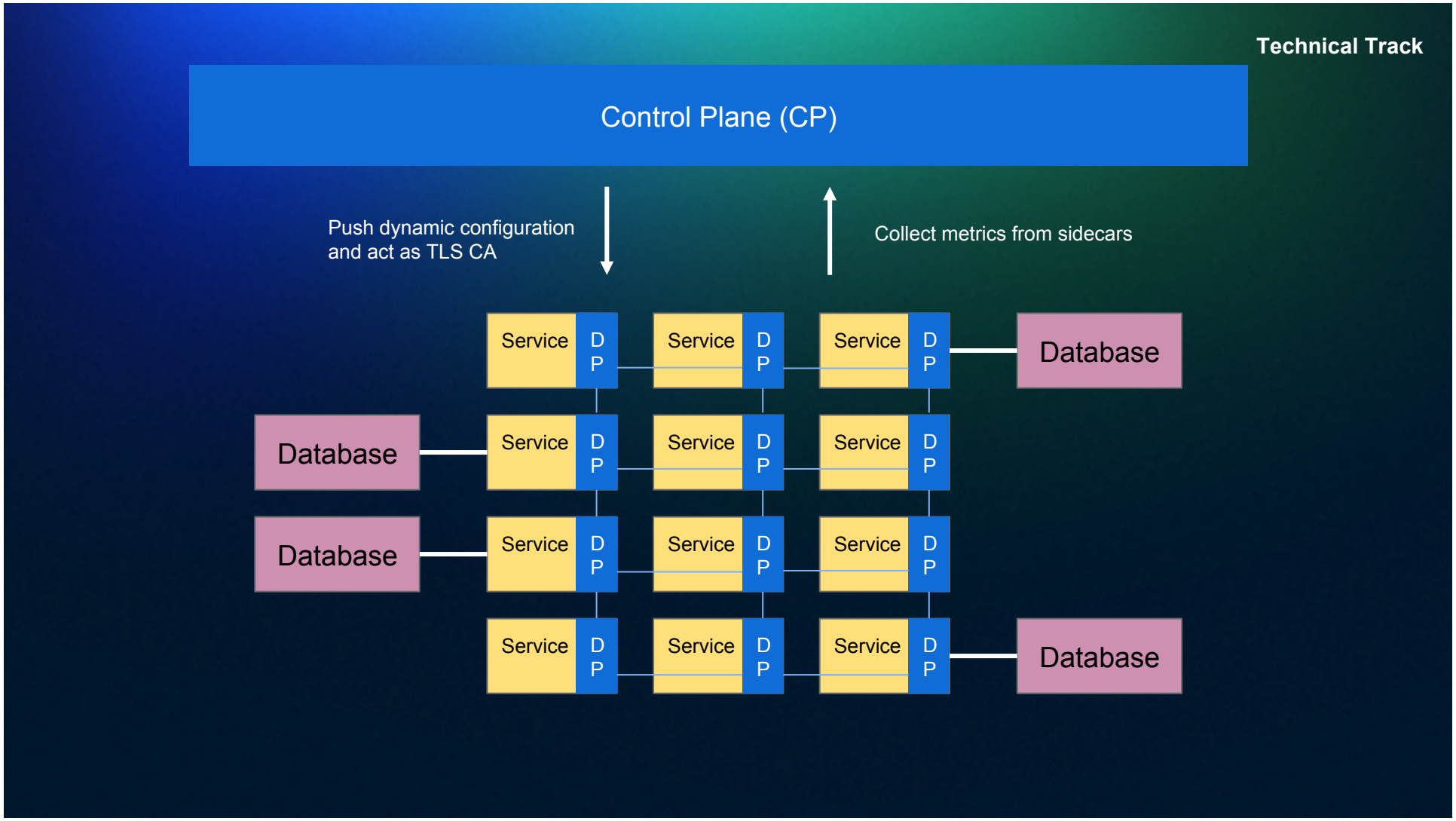


Hence the proxy must be a **sidecar** proxy

aka available on **127.0.0.1**

One instance of proxy for each instance of the service



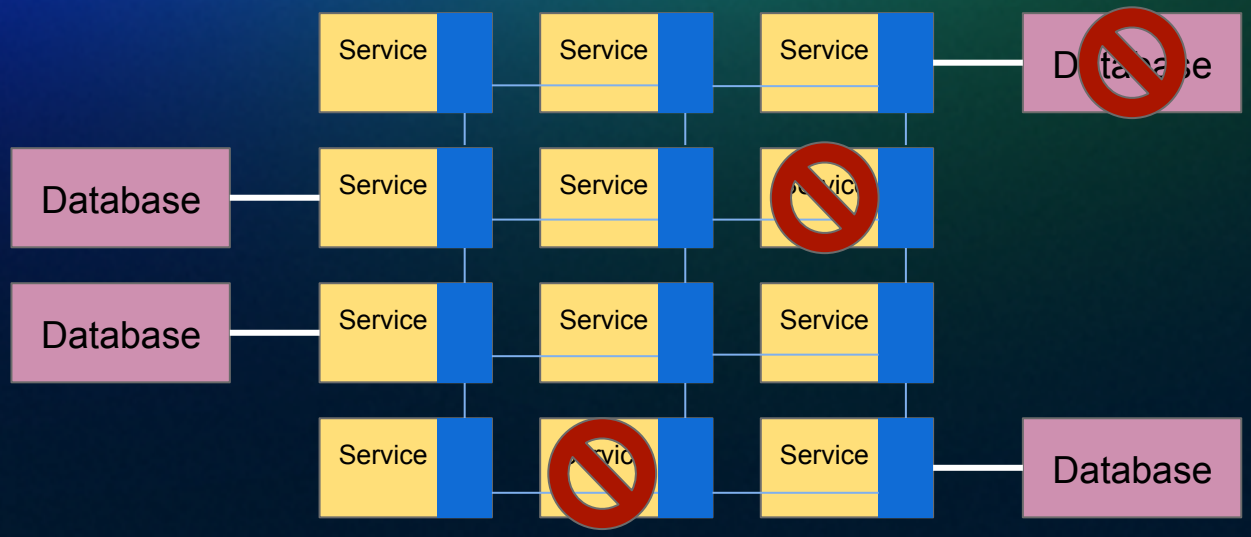

Control Plane (CP)
Push dynamic configuration
and act as TLS CA
Collect metrics from sidecars
Service
D
P
Database

Chaos Engineering

Simulating turbulence in the system to improve how the system responds and performs.

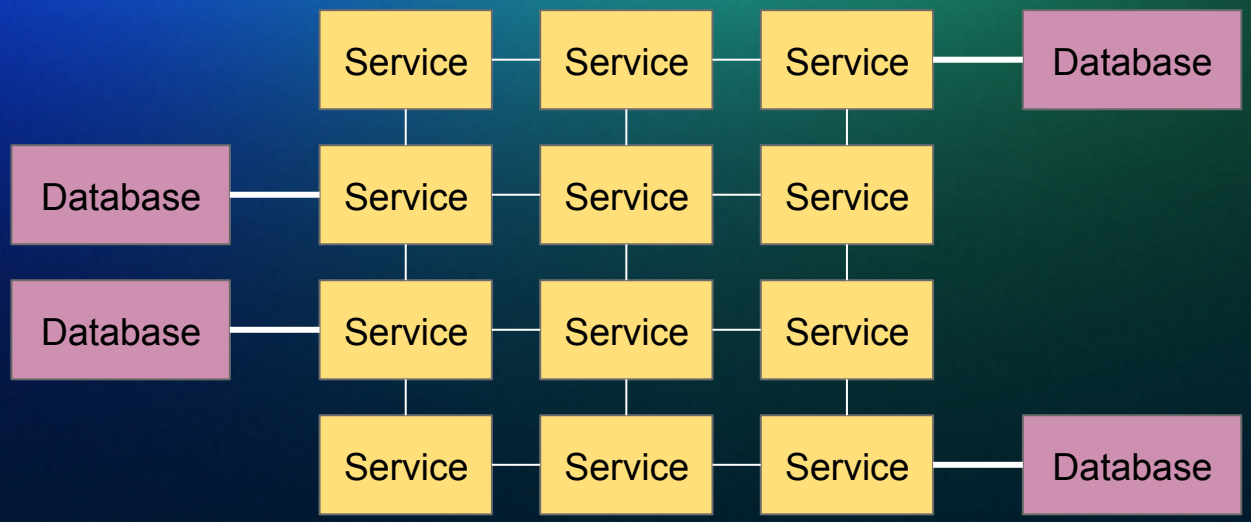
Chaos Engineering

Simulating stress in the system with the goal of improving the system

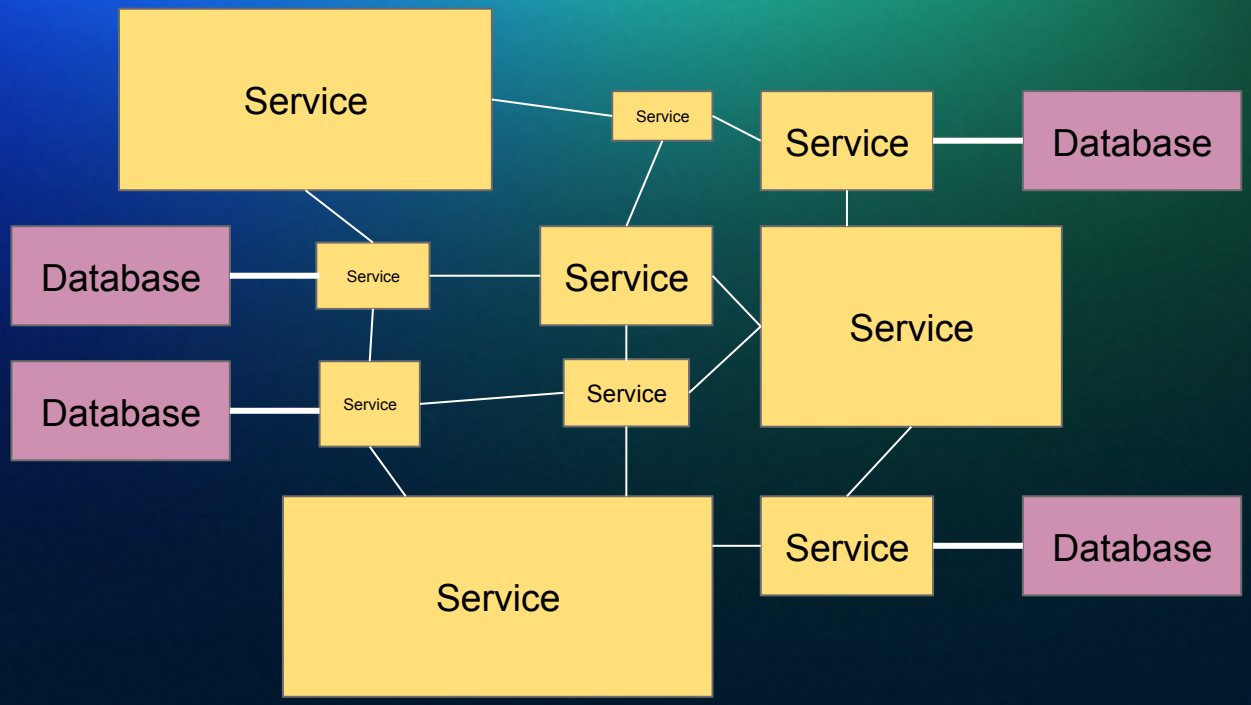


Set up scenarios, execute them, identify weak areas and improve them

Let’s talk about size


Service
Service
Service
Database
Database
Service
Service
Service
Database
Service
Service
Service
Service
Service
Service
Database

Ideal World


Service
Service
Service
Database
Database
Service
Service
Service
Database
Service
Service
Service
Database
Service

Reality

Big enough to hold the domain logic we are decoupling

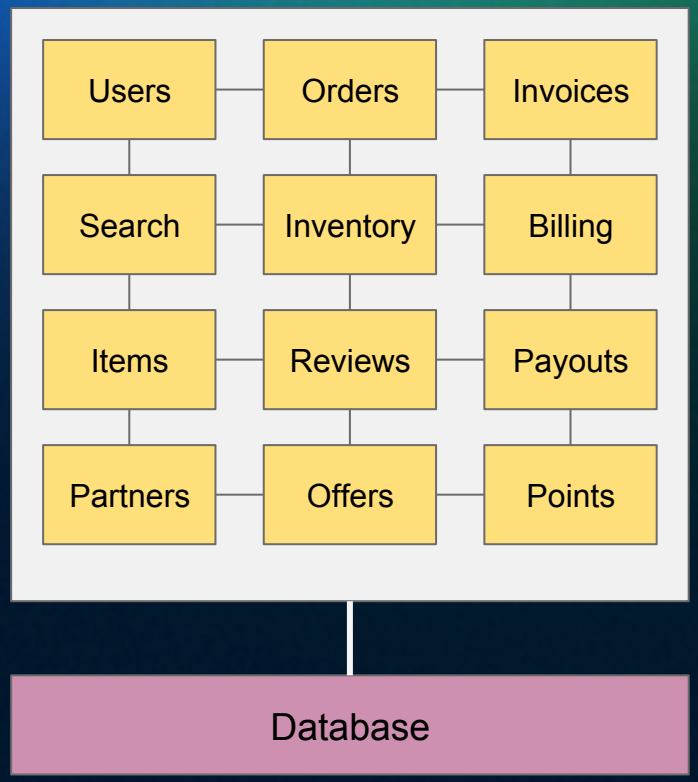
3 strategies

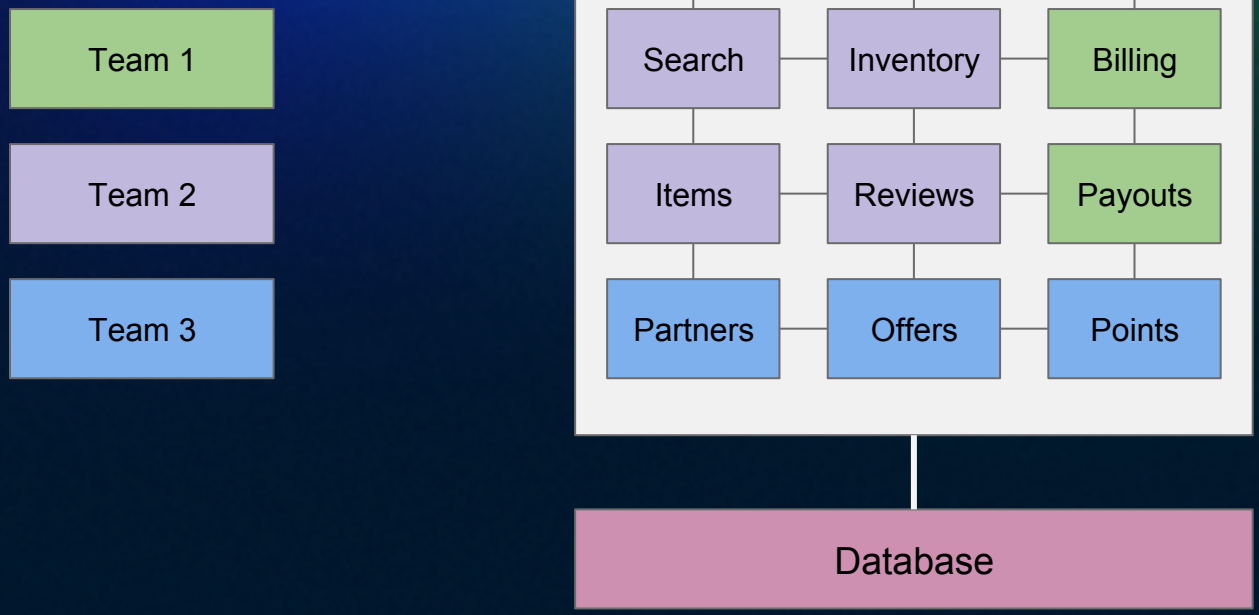
Ice Cream Scoop

Lego Strategy

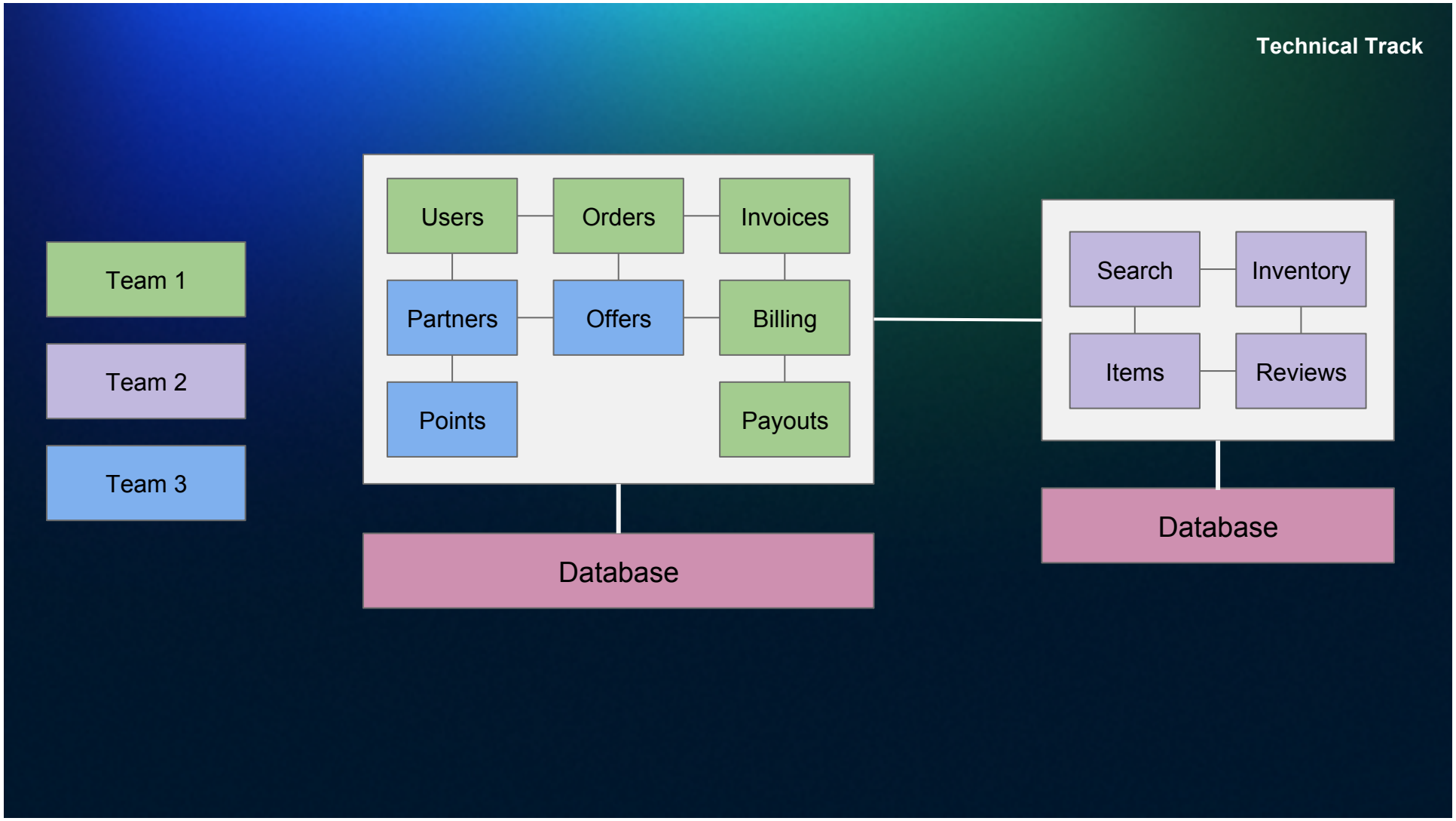
Nuclear Strategy

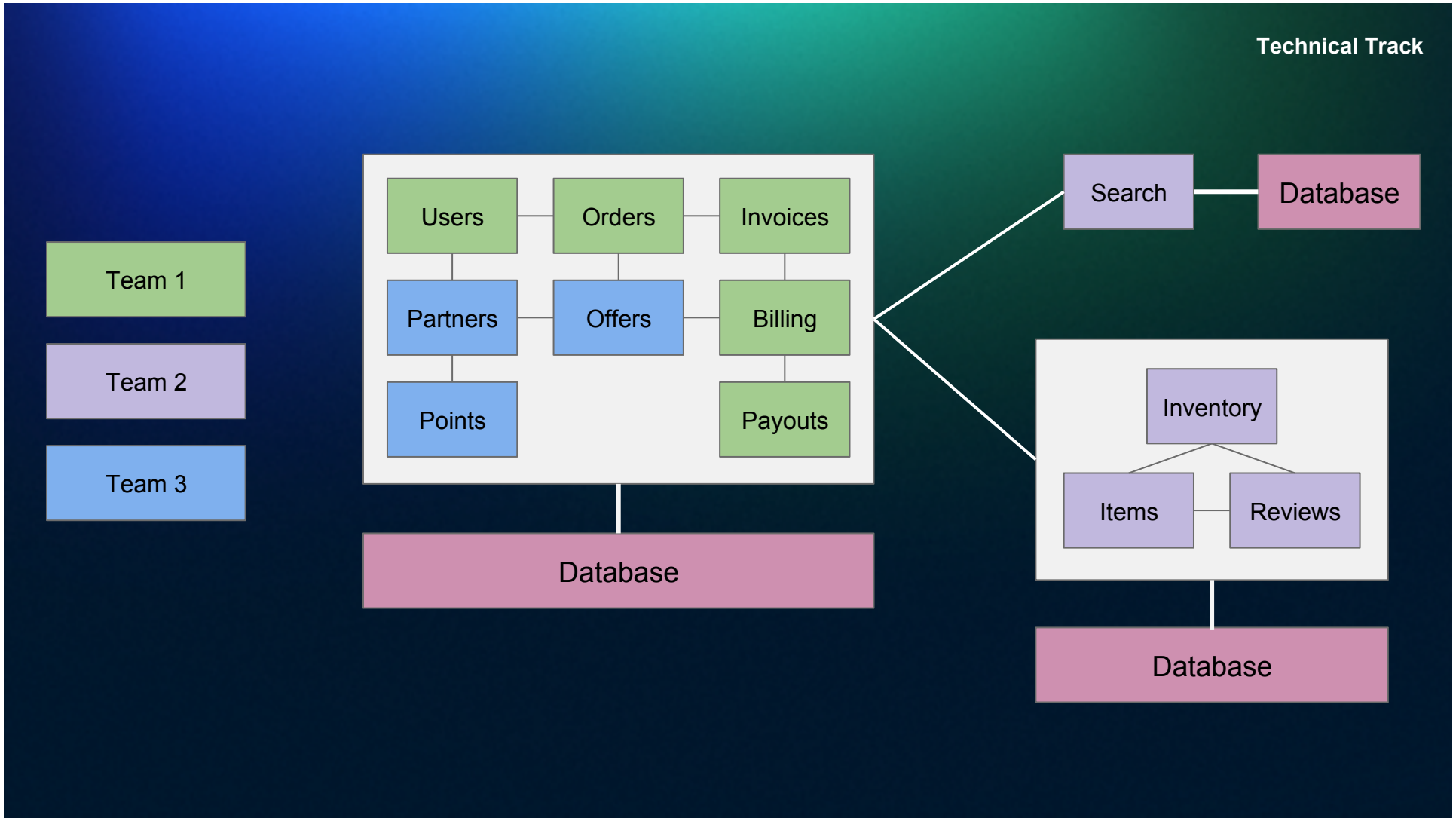
Never go nuclear


Users
Orders
Invoices
Search
Inventory
Billing
Items
Reviews
Payouts
Partners
Offers
Points
Database


Team 1
Team 2
Team 3
Users
Orders
Invoices
Search
Inventory
Billing
Items
Reviews
Payouts
Partners
Offers
Points
Database

Team 2 keeps making code optimizations
that cause frequent deployments


Technical Track
Team 1
Team 2
Team 3
Users
Orders
Invoices
Partners
Offers
Billing
Points
Payouts
Database
Search
Inventory
Items
Reviews
Database


Technical Track
Team 1
Team 2
Team 3
Users
Orders
Invoices
Partners
Offers
Billing
Points
Payouts
Database
Search
Database
Inventory
Items
Reviews
Database

Transitioning to microservices is refactoring

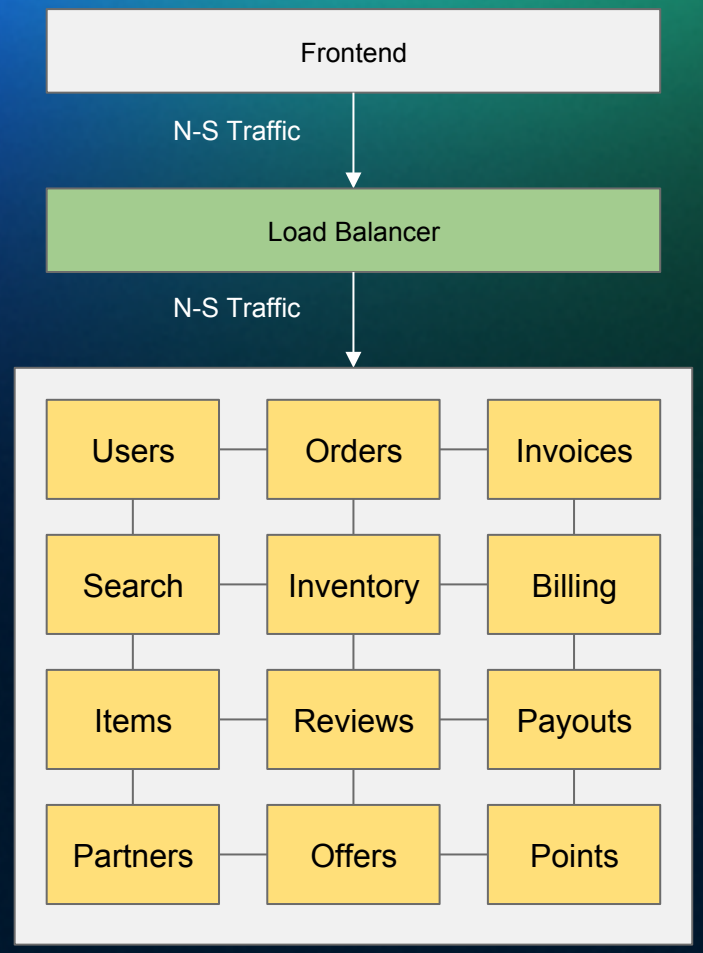
Approaching the transition

Clear understanding of what the Monolith does and does not

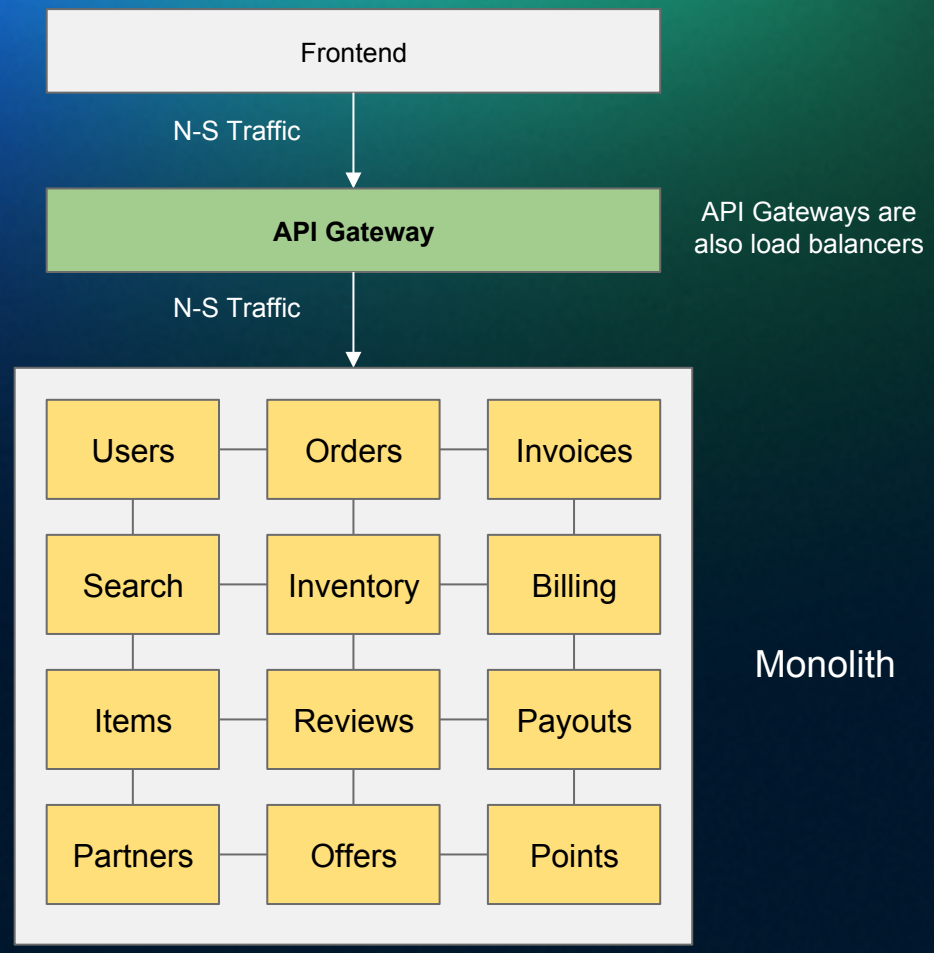
Understanding of clients that are consuming the monolith

Tests, Tests, Tests.

Decouple clients from the monolithic server

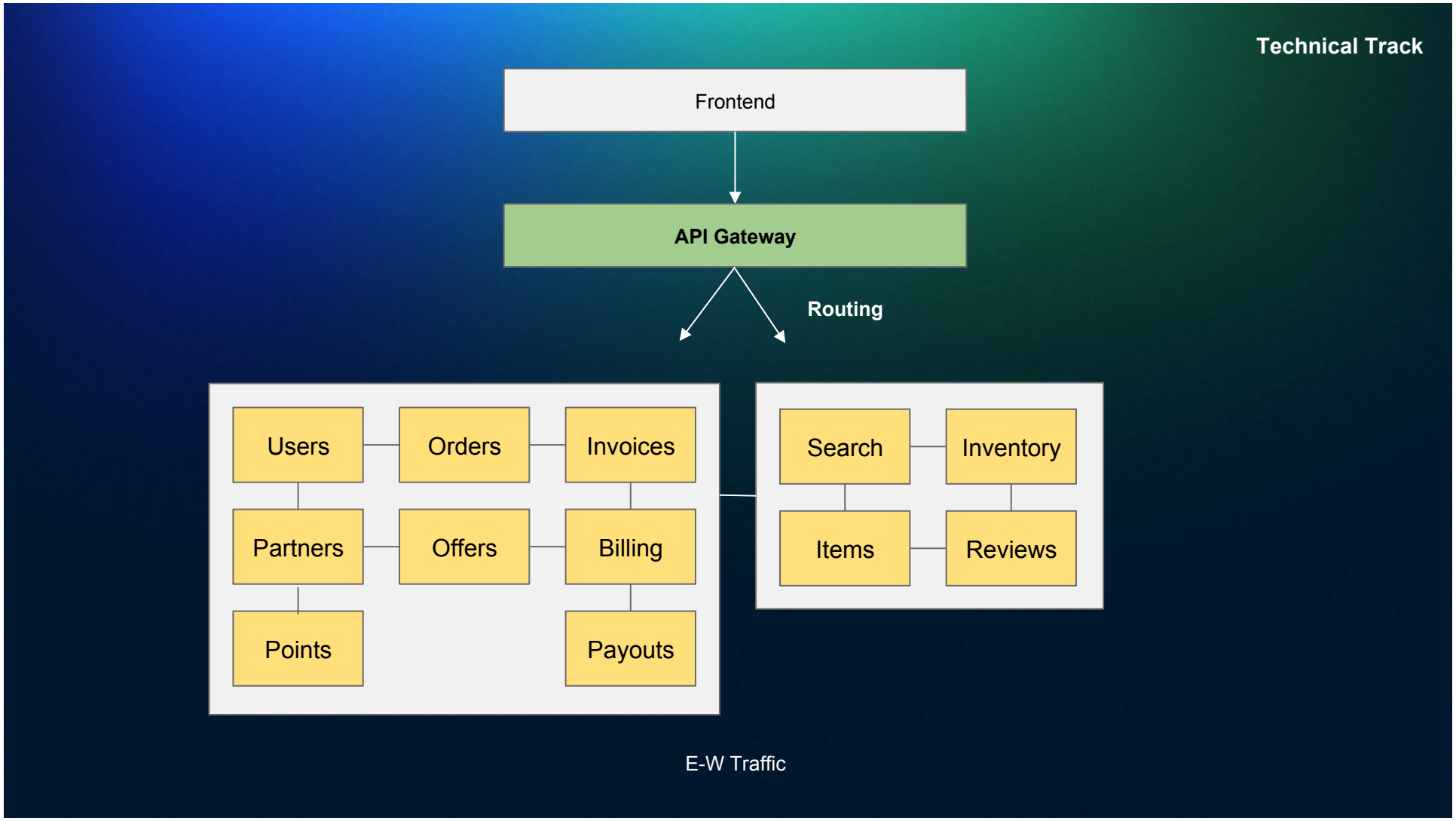

Frontend
N-S Traffic
Load Balancer
N-S Traffic
Users
Orders
Invoices
Search
Inventory
Billing
Items
Reviews
Payouts
Partners
Offers
Points

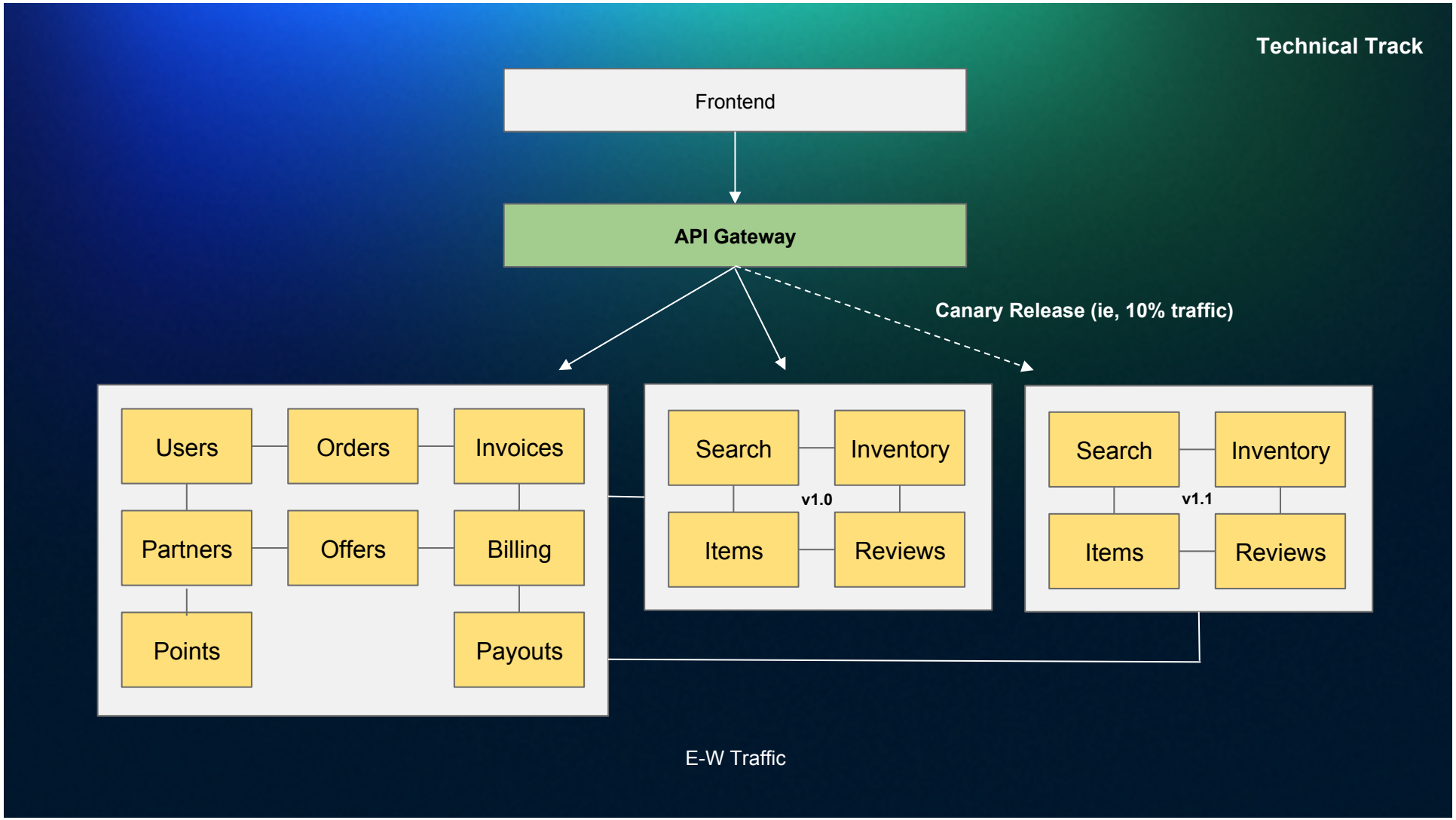
Monolith



API Gateways are also load balancers

Monolith


Technical Track
Frontend
API Gateway
Routing
Users
Orders
Invoices
Partners
Offers
Billing
Points
Payouts
Search
Inventory
Items
Reviews
E-W Traffic


Technical Track
Frontend
API Gateway
Canary Release (ie, 10% traffic)
Users
Orders
Invoices
Partners
Offers
Billing
Points
Payouts
Search
Inventory
v1.0
Items
Reviews
Search
Inventory
v1.1
Items
Reviews
E-W Traffic

Convergence of API Gateway feature-set in both N-S and E-W

N-S Features:

AuthN / AuthZ
Routing
Logging
Transformation
Analytics
Developer Portal
Integration Layer
Healthchecks
Circuit Breakers
Request Collapsing
CORS
Rate-Limiting
Throttling
Mutual TLS
…

E-W Features:

AuthN / AuthZ
Routing
Logging
Transformation
Analytics
Developer Portal
Healthchecks
Circuit Breakers
Mutual TLS
…

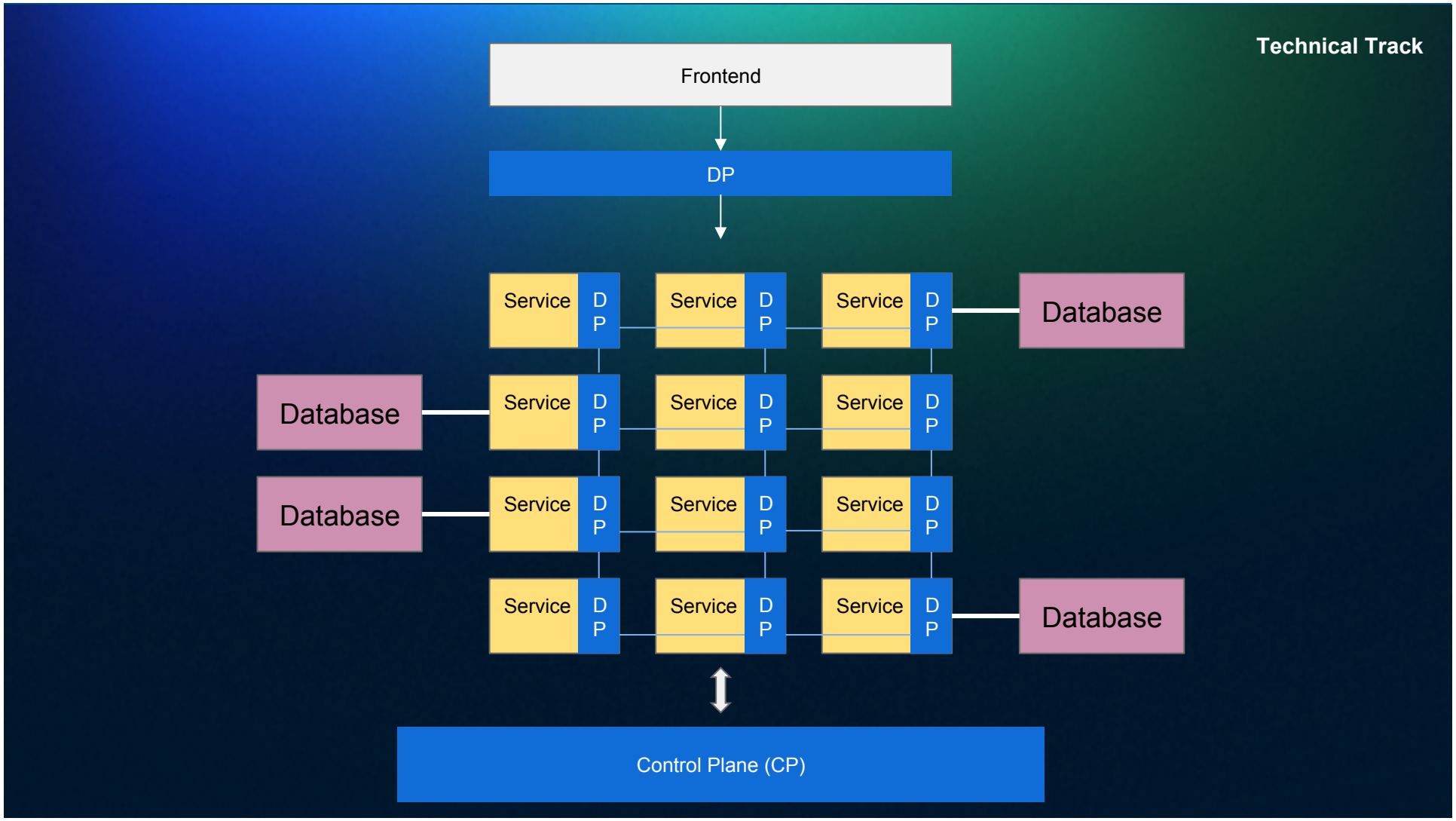
N-S Features:

AuthN / AuthZ
Routing
Logging
Transformation
Analytics
Developer Portal
Integration Layer
Healthchecks
Circuit Breakers
Request Collapsing
CORS
Rate-Limiting
Throttling
Mutual TLS
…

E-W Features:

AuthN / AuthZ
Routing
Logging
Transformation
Analytics
Developer Portal
Healthchecks
Circuit Breakers
Mutual TLS
…

E-W uses a **subset** of the feature-set that N-S provides

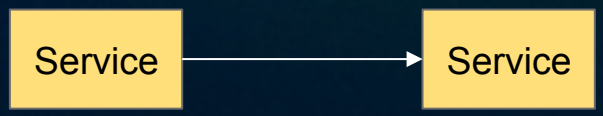

Technical Track
Frontend
DP
Service
D
P
Database
Control Plane (CP)

Service Mesh is a pattern - not a technology

State Propagation

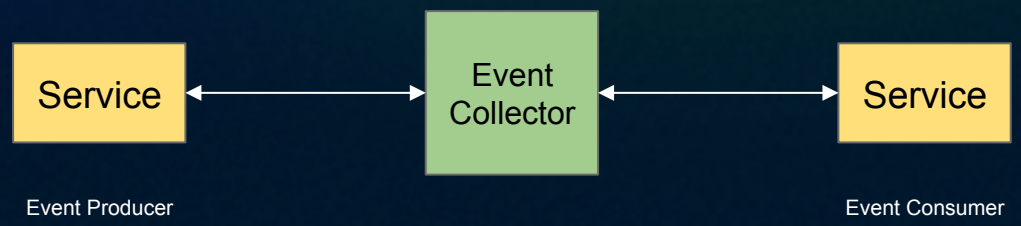
Service-to-Service

Until now we focused on Service-to-Service communication



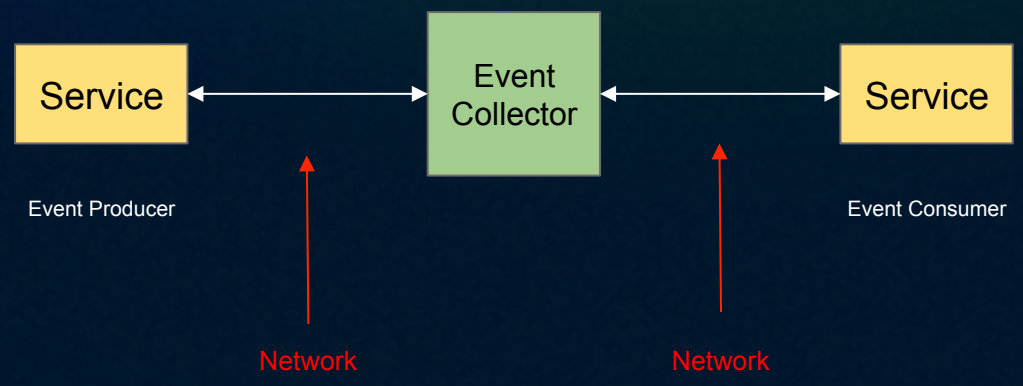
Event Based Microservices

Ideal for eventual consistent state propagation



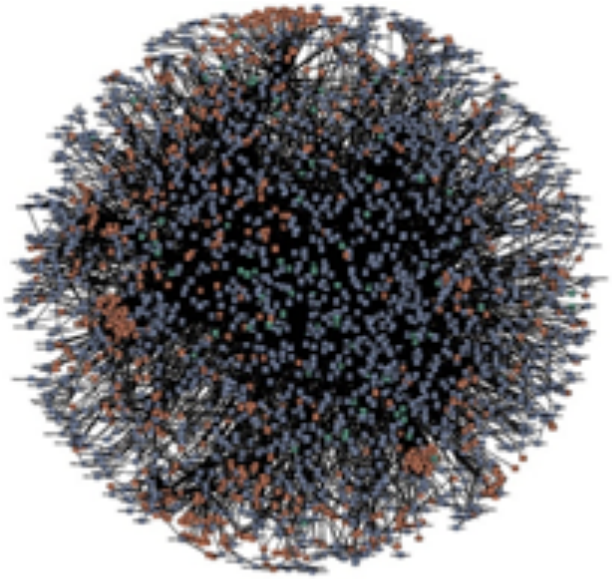
No need for Healthchecks and Circuit Breakers

As long as the Event Collector is up and running



Service-To-Service (Synchronous)	Event Based (Asynchronous)
Microservices and clients directly consume and invoke other microservices.	Microservices and clients push event into an event collector that's being consumed by other microservices.
Ideal for clients that require an immediate response or need to aggregate multiple services together.	Ideal for microservice-to-microservice communication for changing state without requiring an immediate response.
Done via HTTP, TCP/UDP, gRPC,etc.	Done via Kafka, RabbitMQ, AWS SQS, etc.
Example: Making a request to retrieve an immediate response of some sort (ie, retrieve list of users).	Example: Making a request that doesn't require an immediate response (ie, "orderCreated" event that triggers an invoice creation by another microservice).

Operational considerations for microservices



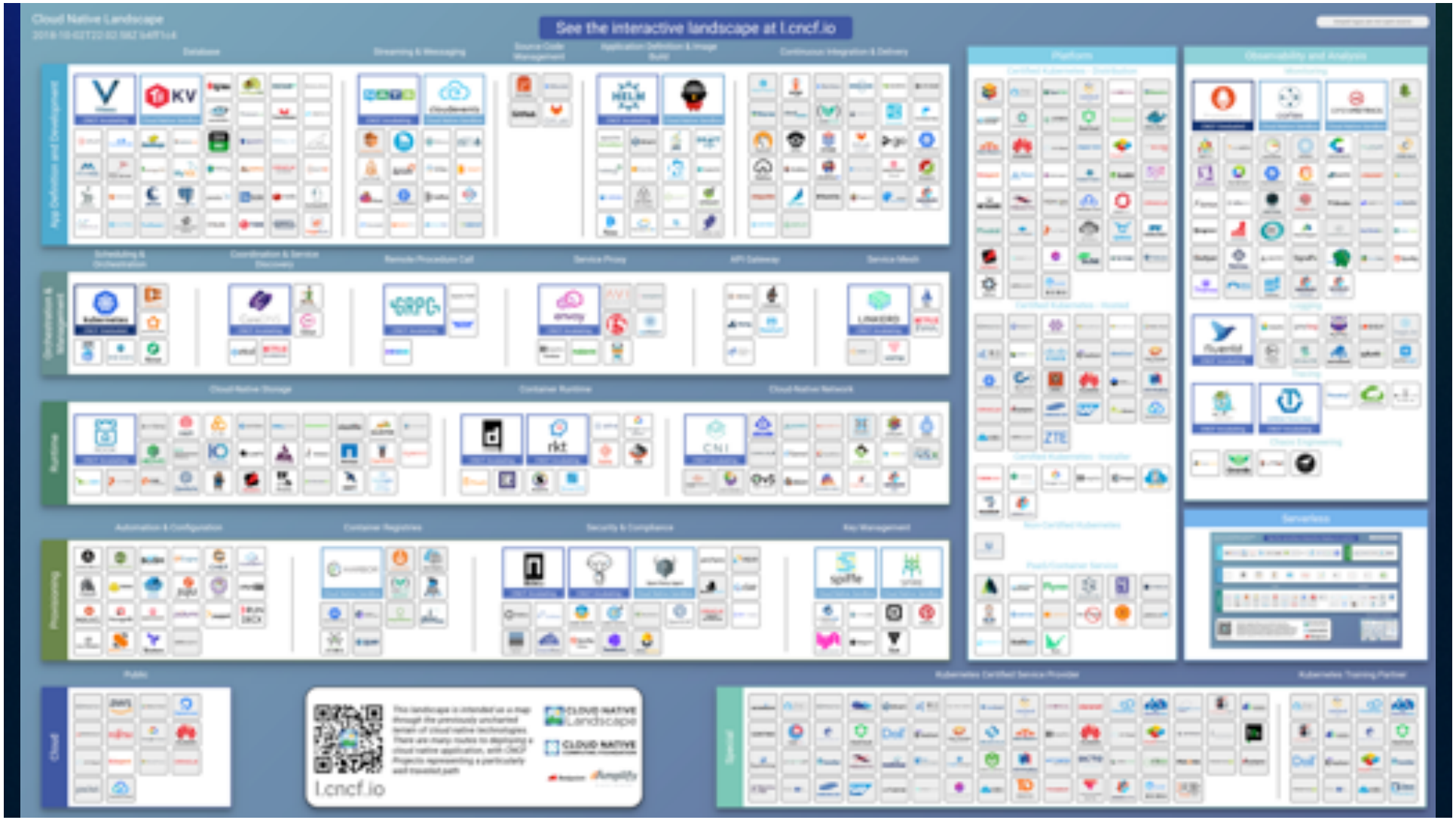
amazon.com

NETFLIX

With microservices we expect an ever increasing number of moving parts moving forward

Deployments, Logging, Testing, CI/CD, and so on

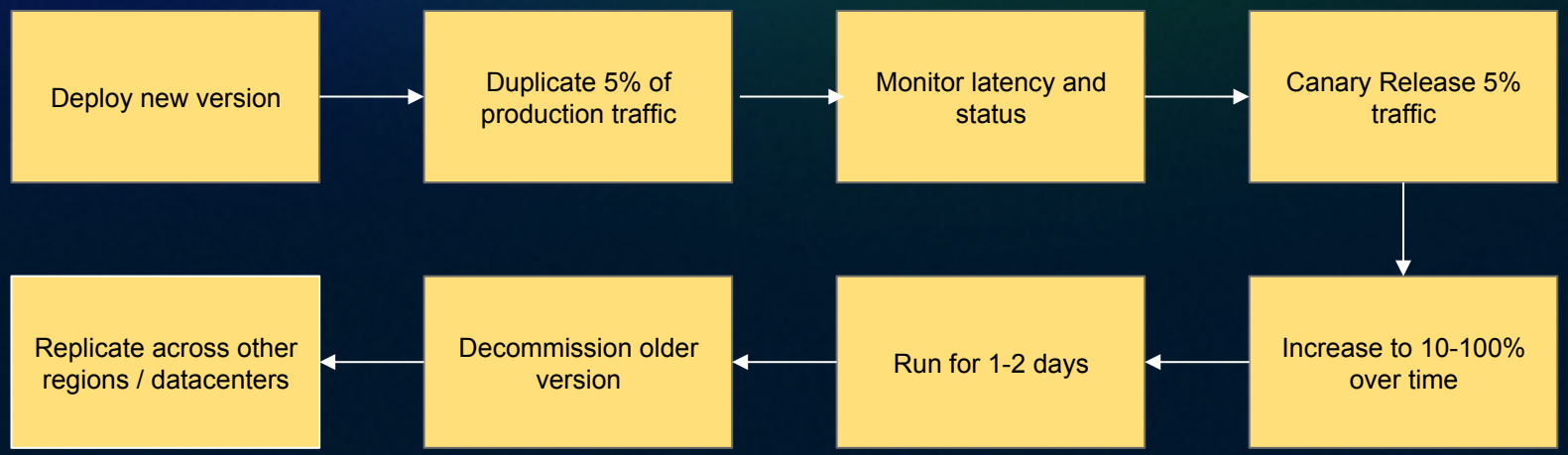
must be rethought


Cloud Native Landscape
See the interactive landscape at l.cncf.io
Database
Streaming & Messaging
Source Code Management
Application Definition & Image Build
Continuous Integration & Delivery
App Definition and Development
Scheduling & Orchestration
Coordination & Service Discovery
Remote Procedure Call
Service Proxy
API Gateway
Service Mesh
Orchestration & Management
Cloud Native Storage
Container Runtime
Cloud Native Network
Runtime
Automation & Configuration
Container Registries
Security & Compliance
Key Management
Provisioning
Public
Cloud
Platform
Certified Kubernetes - Distribution
Certified Kubernetes - Hosted
Certified Kubernetes - Installer
Non-Certified Kubernetes
PaaS/Container Service
Observability and Analysis
Monitoring
Logging
Tracing
Chaos Engineering
Serverless
Kubernetes Certified Service Provider
Kubernetes Training Partner
Special
l.cncf.io
CLOUD NATIVE Landscape
CLOUD NATIVE

If we can’t operate a monolith today

do not move to microservices

Microservices Lifecycle

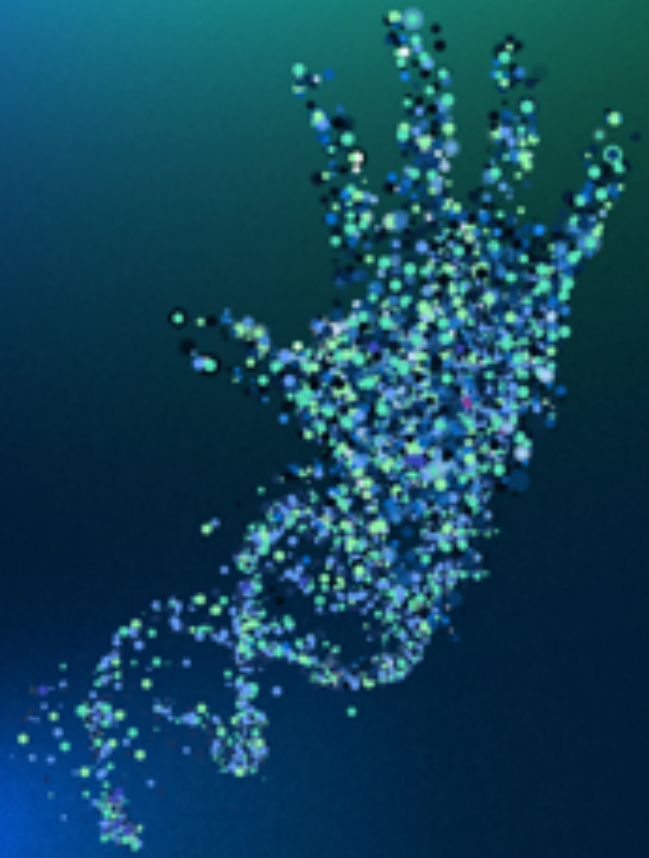


If we can’t operate a monolith today

do not move to microservices

Organizational considerations for microservices

Service Catalog, Dev Portals, Governance, AuthN/AuthZ, and so on become exponentially more important



Documentation and on-boarding becomes critical

Organizational Track

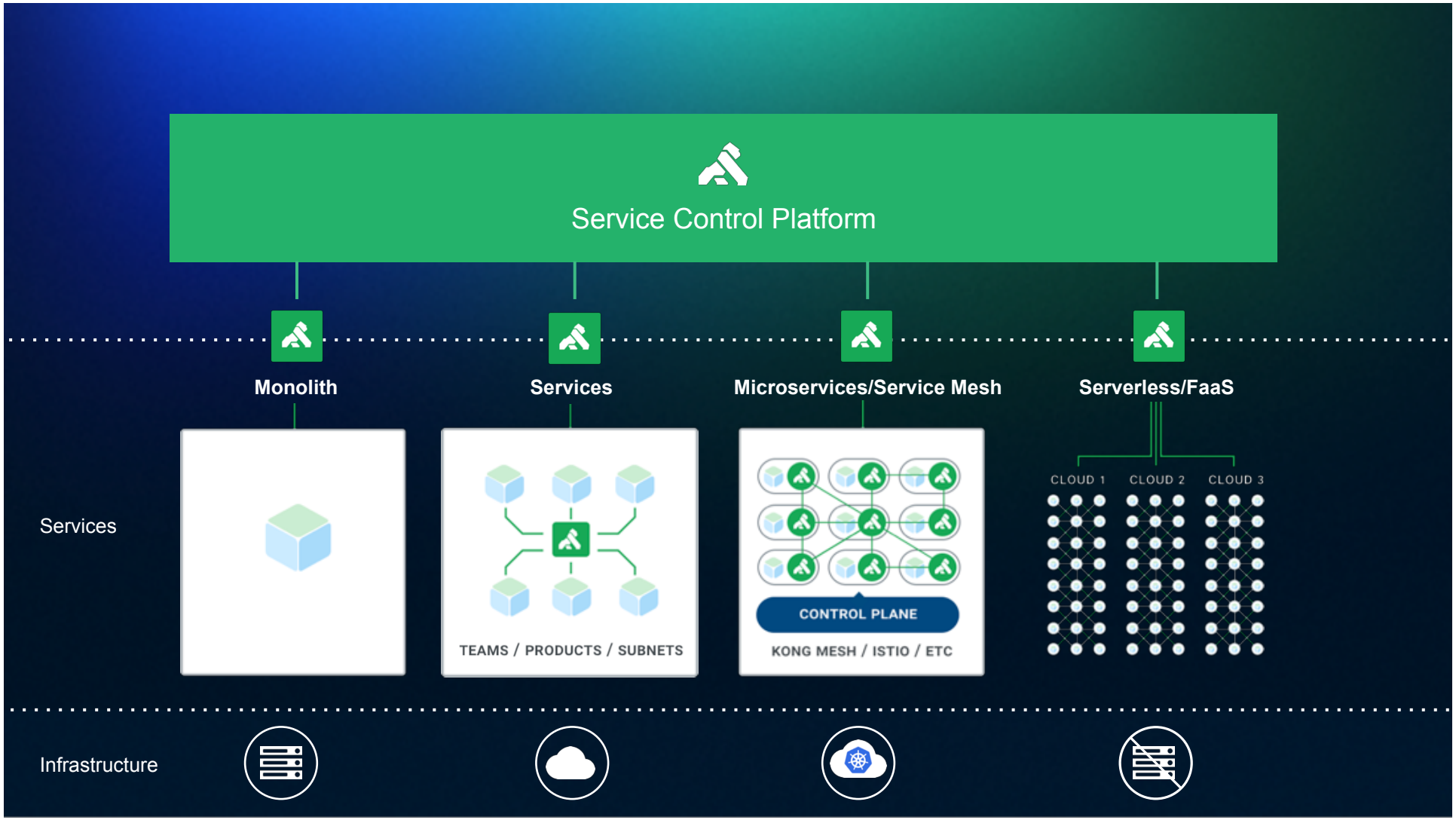
Technology	Monolith	Services	Microservices
Organization	Single large team	Pizza teams	+ individual contributors
Org + Tech Optimization	Inefficient	Efficient, decoupled	Hyperoptimized

Isn’t this SOA all over again?

SOA was driven by vendors.

Microservices are driven by developers.

Traditional API Management is Outdated


Service Control Platform
Monolith
Services
Microservices/Service Mesh
Serverless/FaaS
Services
TEAMS / PRODUCTS / SUBNETS
CONTROL PLANE
KONG MESH / ISTIO / ETC
CLOUD 1
CLOUD 2
CLOUD 3
Infrastructure

Platforms

Languages

Containers

Protocols

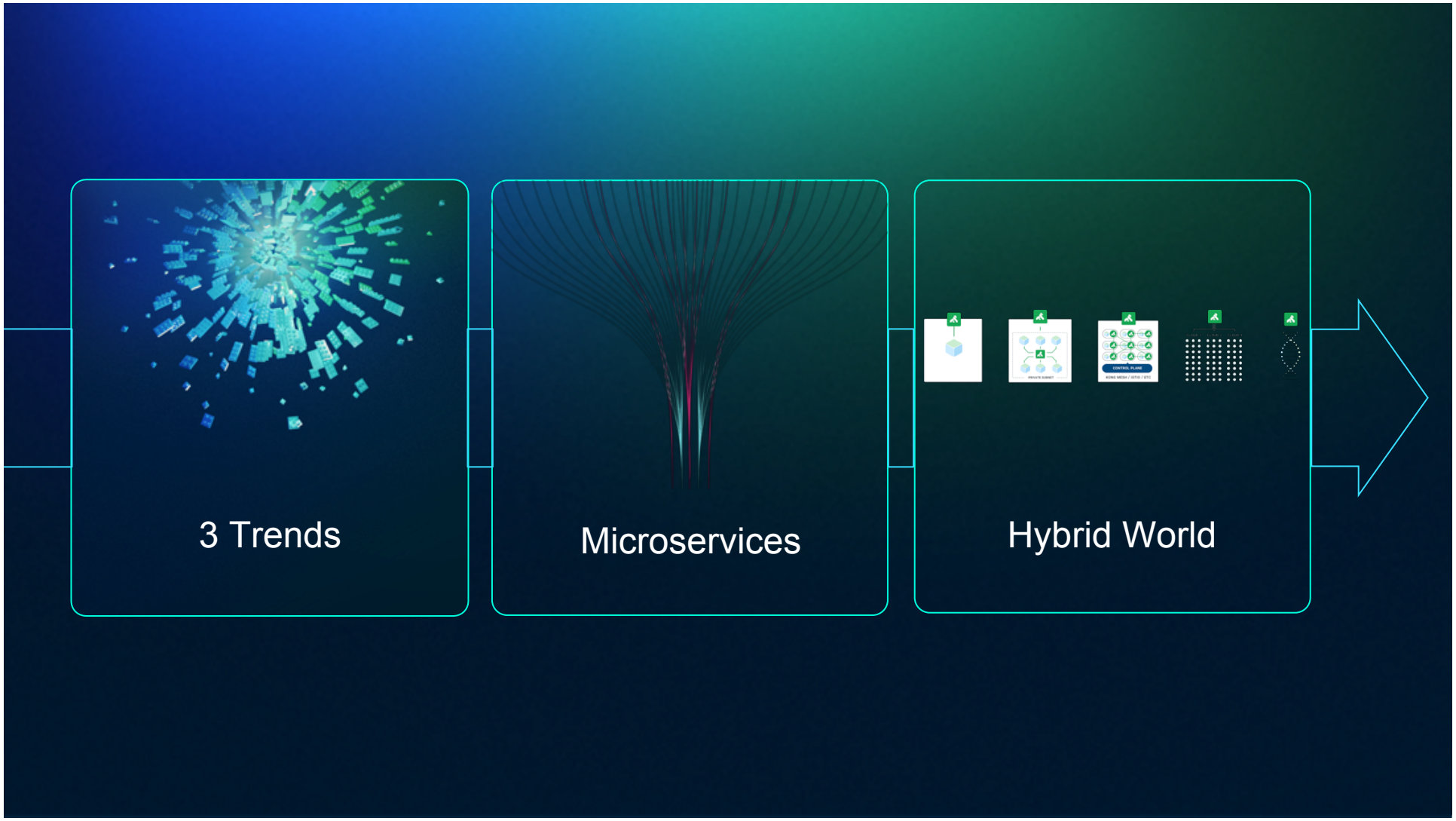
Hybrid Services

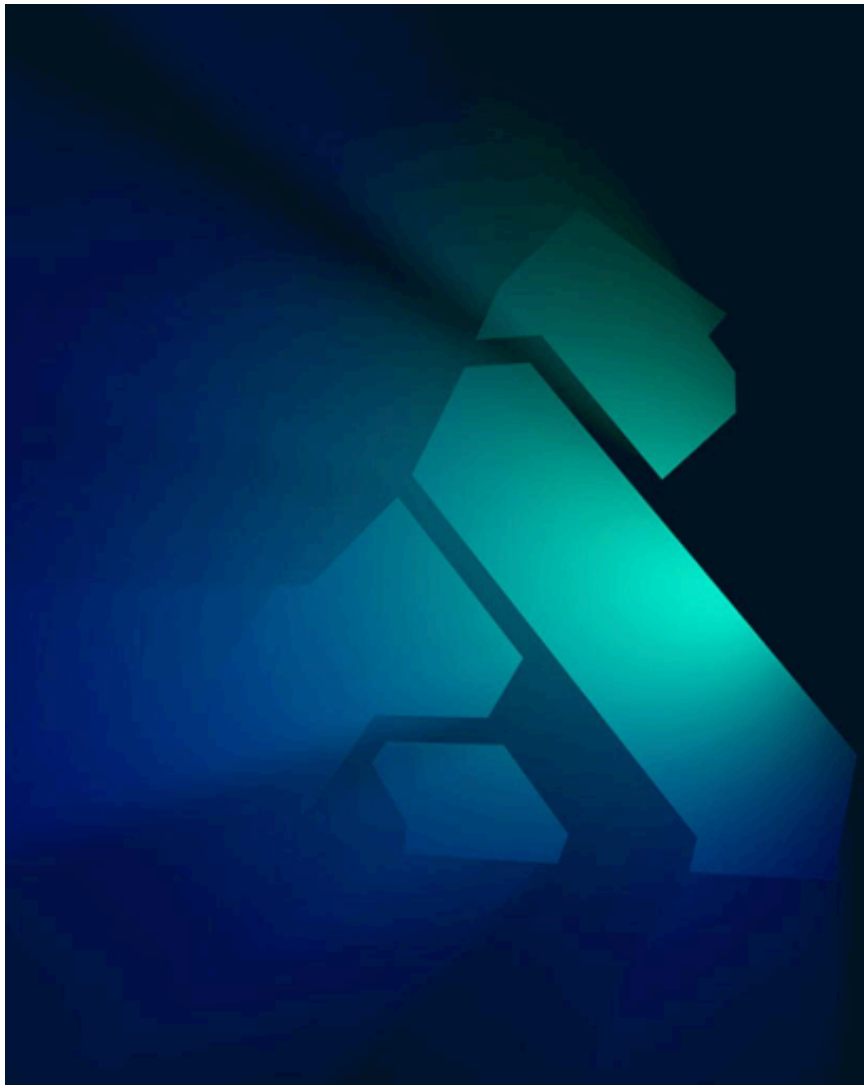
Clouds

Architectures

Teams

Organizations


3 Trends
Microservices
Hybrid World



Thank you!

@thijsschreijer @thekonginc

https://konghq.com


Questions?
https://konghq.com
@thijsschreijer @thekonginc