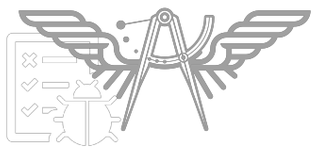
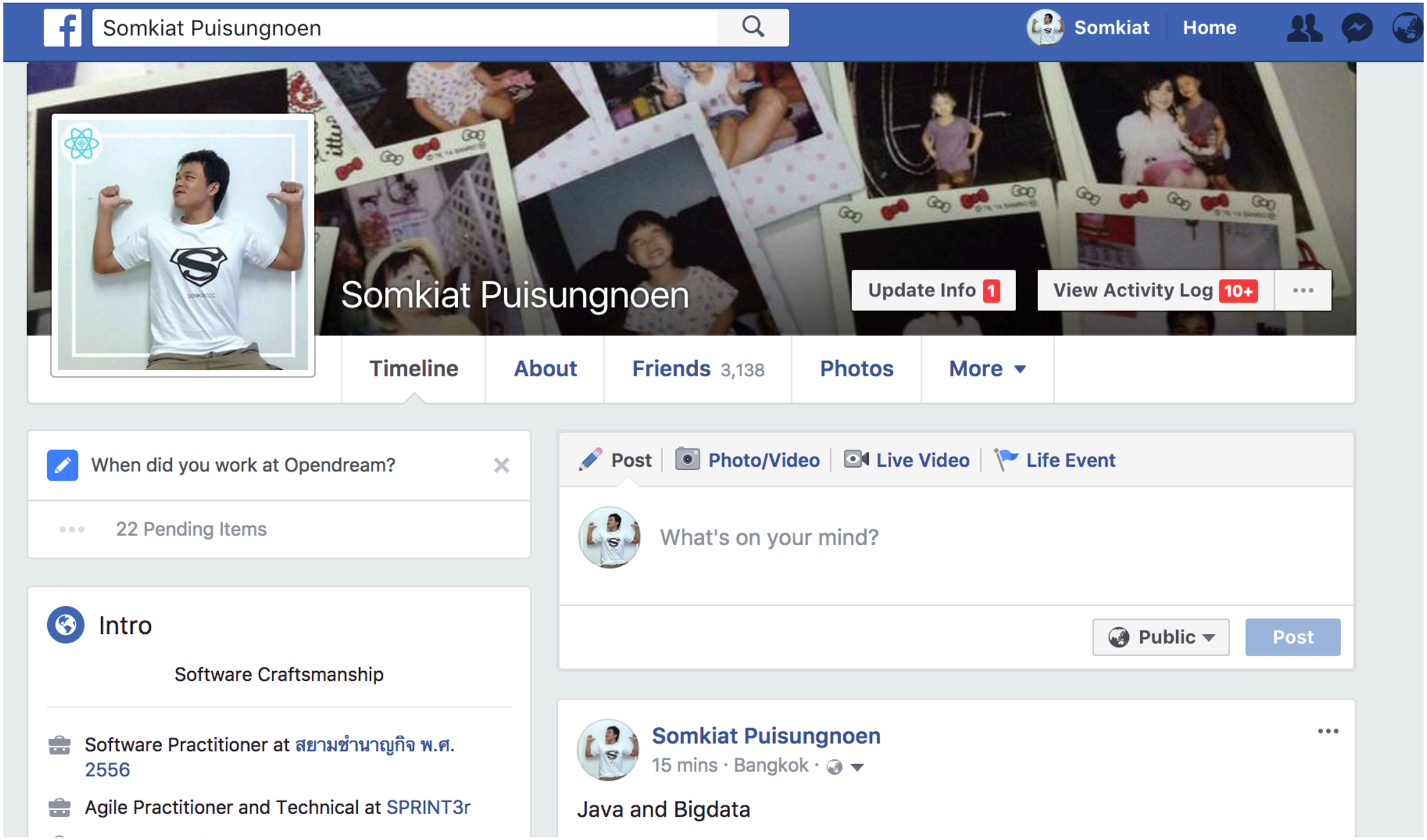
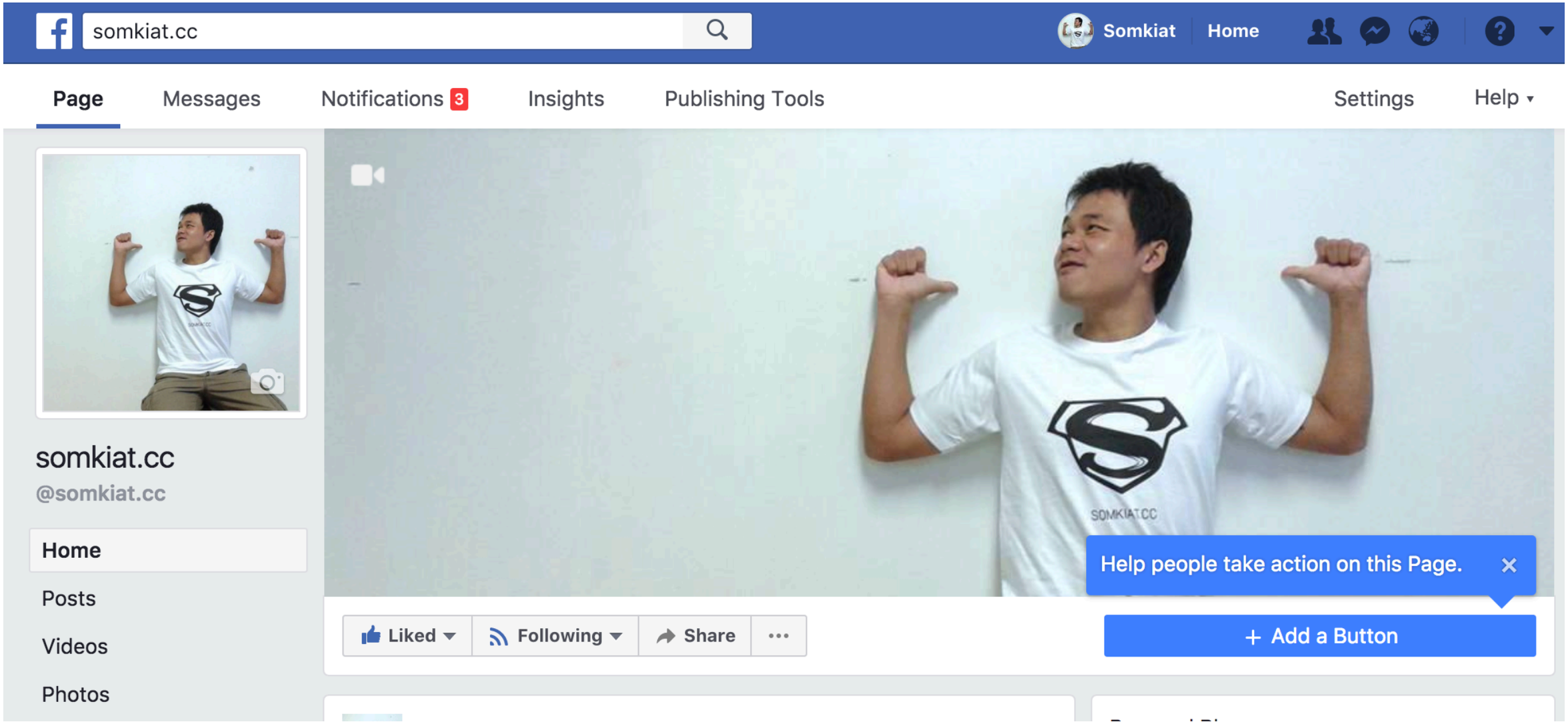
สยามชำนาญกิจ
SIAM CHAMNAN KIT

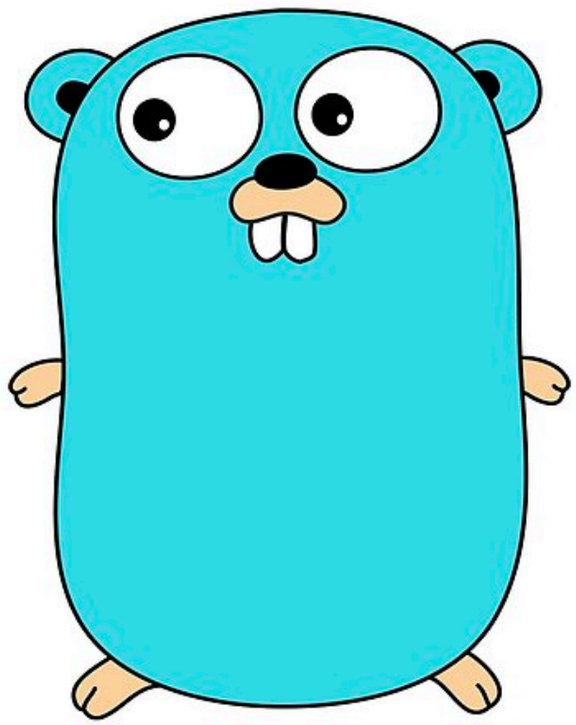
Microservice with Go





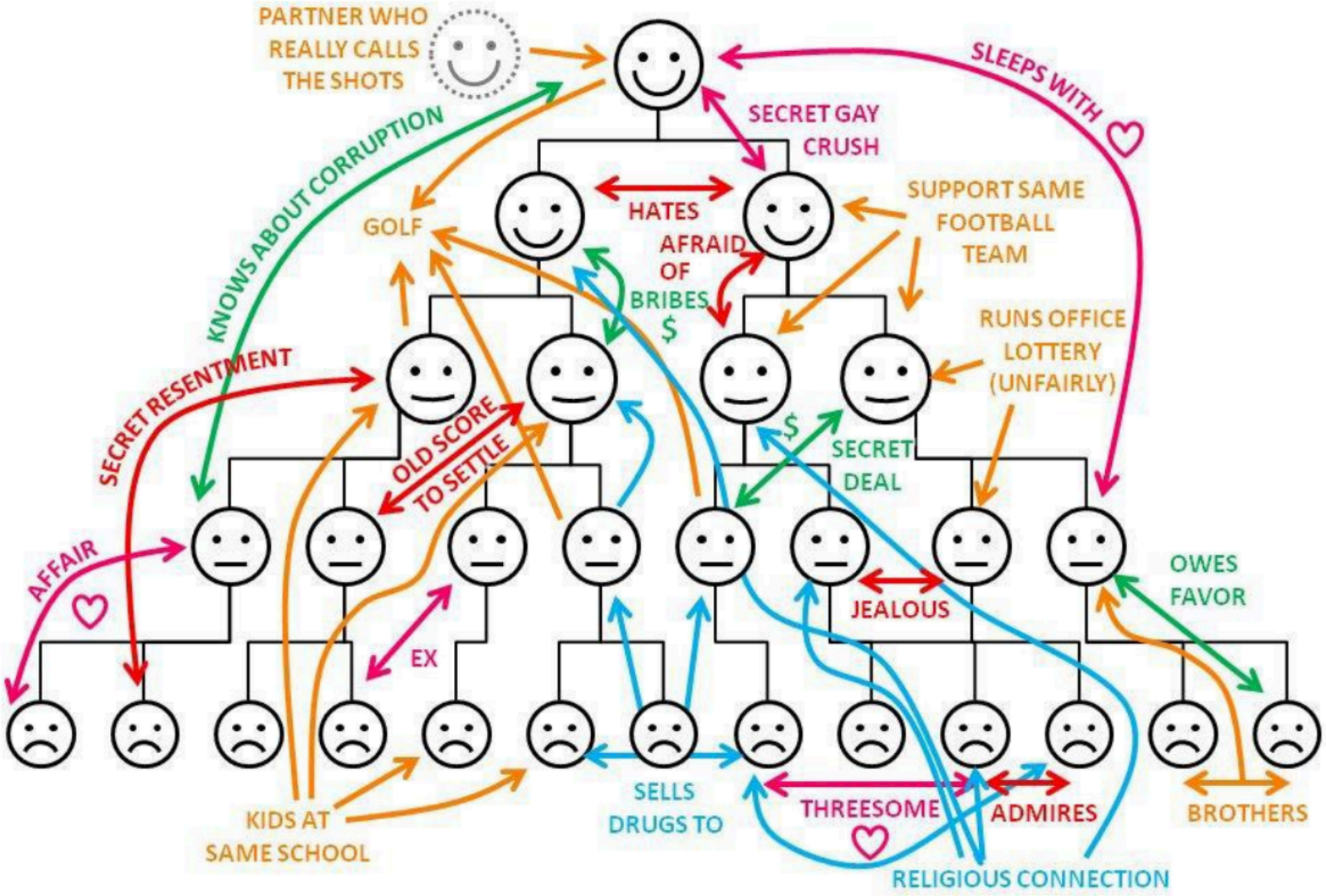


Microservice



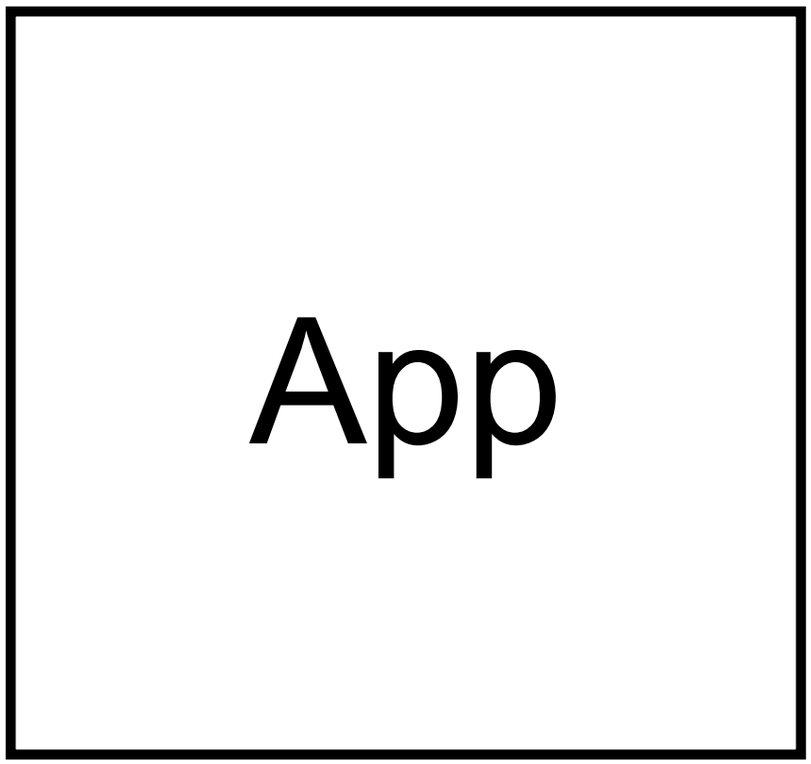
”organizations which design systems ... are constrained to produce designs which are copies of the communication structures of these organizations”

- Melvin Conwey, 1968 -

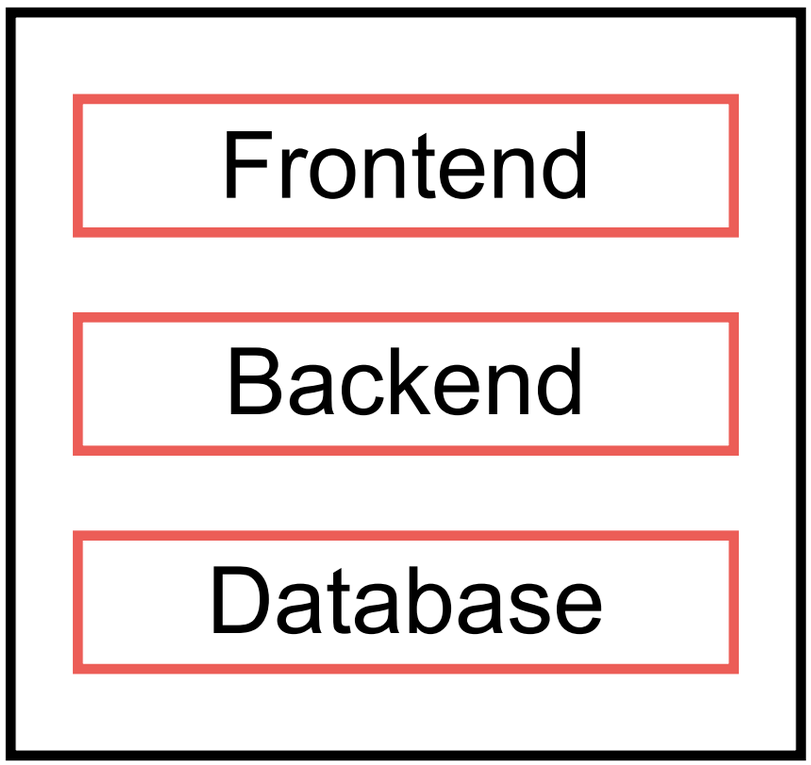


Evolution of Architecture

Monolith



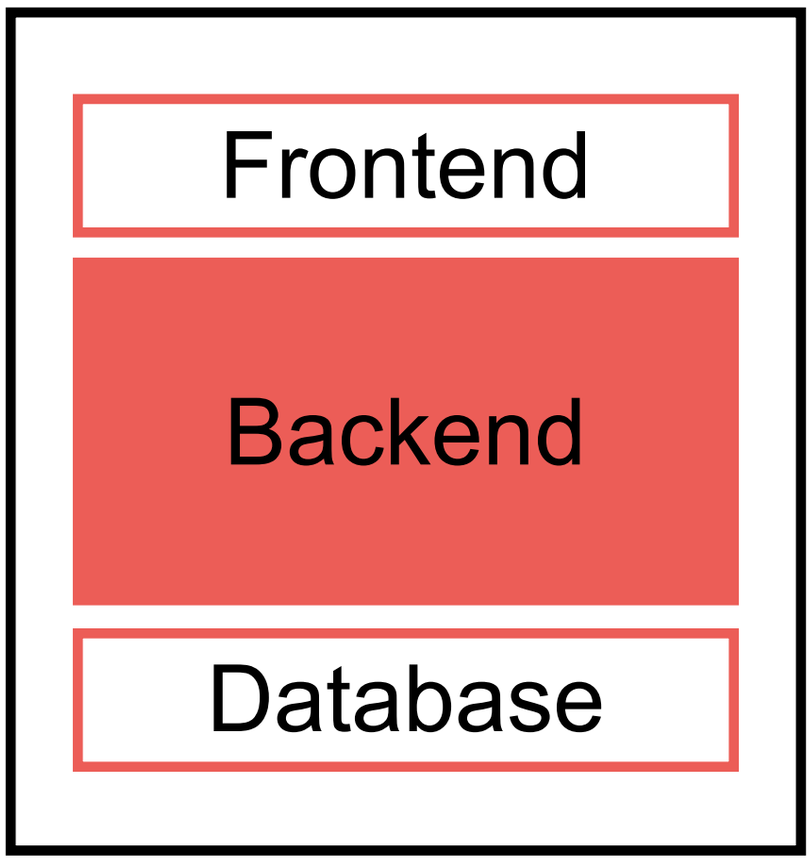
Layer



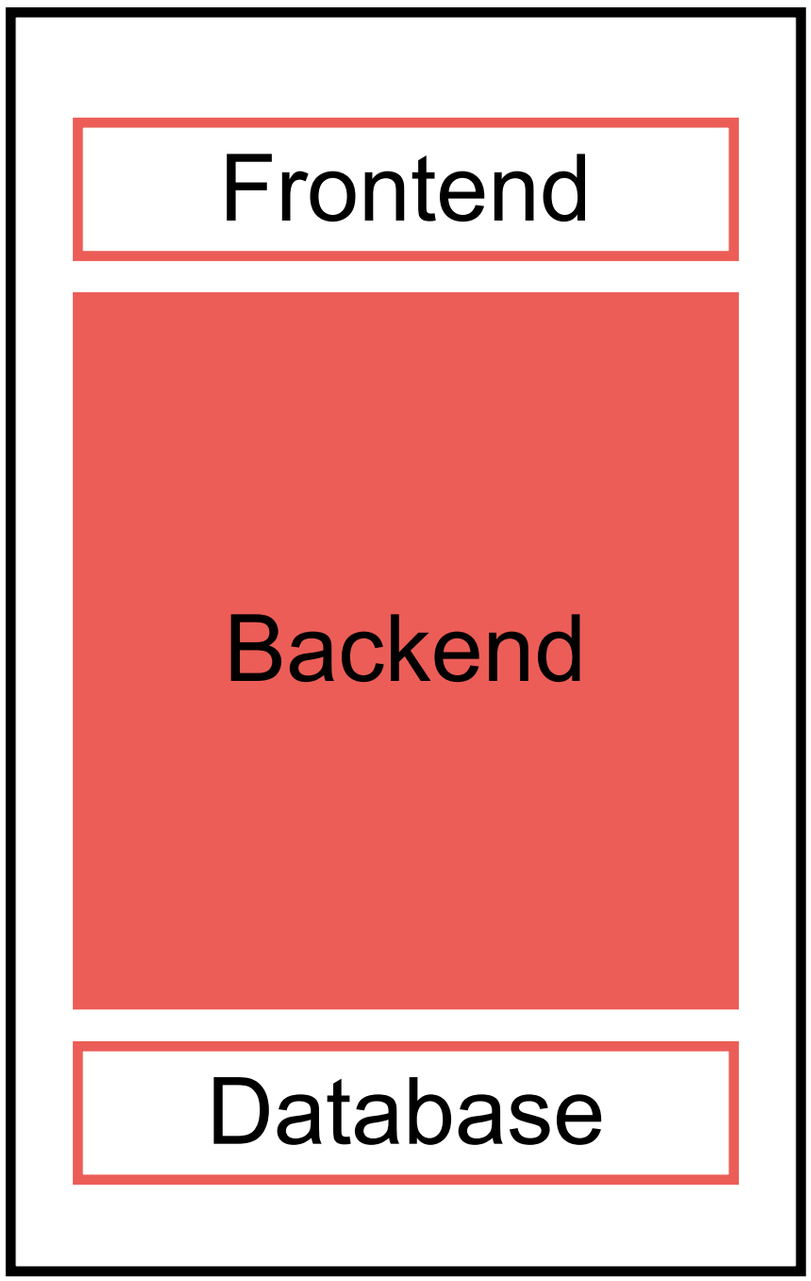
Monolith

Easy to develop
Easy to change
Easy to testing
Easy to deploy
Easy to scale

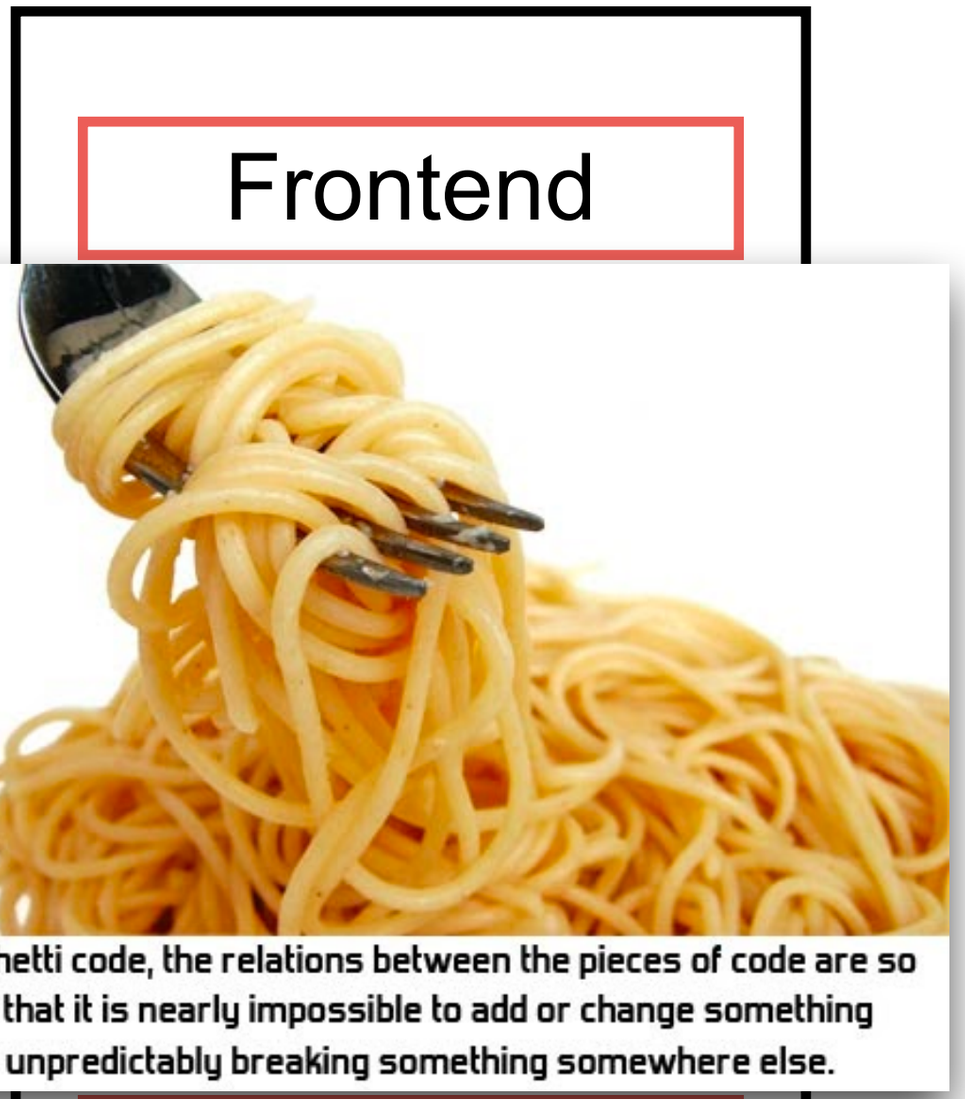
More features …



More features ...



More features …



More features …

In spaghetti code, the relations between the pieces of code are so tangled that it is nearly impossible to add or change something without unpredictably breaking something somewhere else.

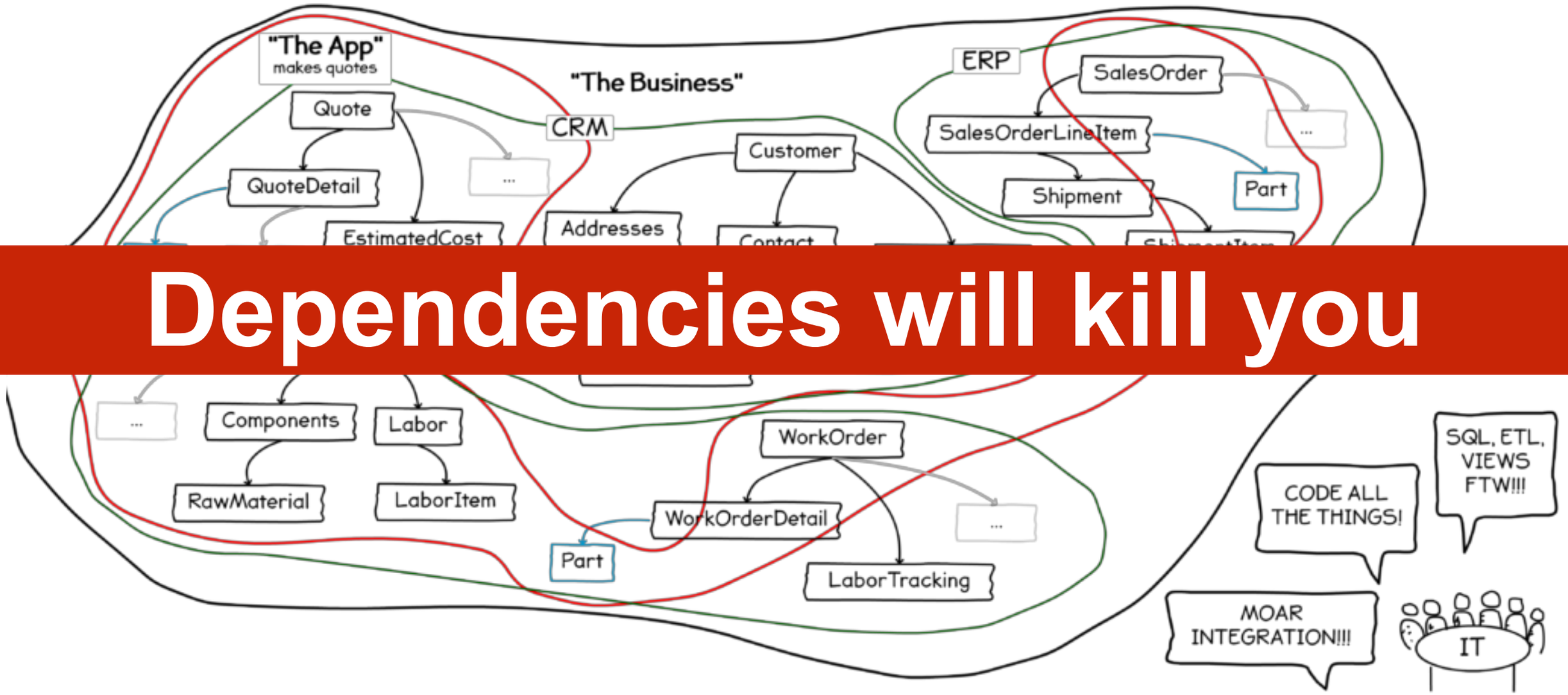
More features …

In spaghetti code, the relations between the pieces of code are so tangled that it is nearly impossible to add or change something without unpredictably breaking something somewhere else.

In spaghetti code, the relations between the pieces of code are so tangled that it is nearly impossible to add or change something without unpredictably breaking something somewhere else.

More features …

Monolith Hell

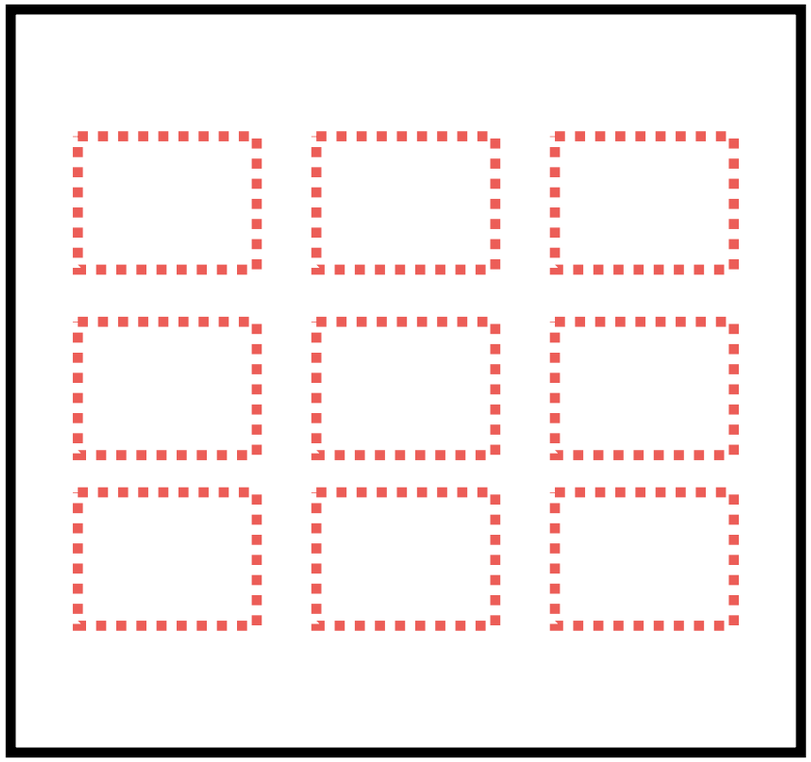


Dependencies will kill you

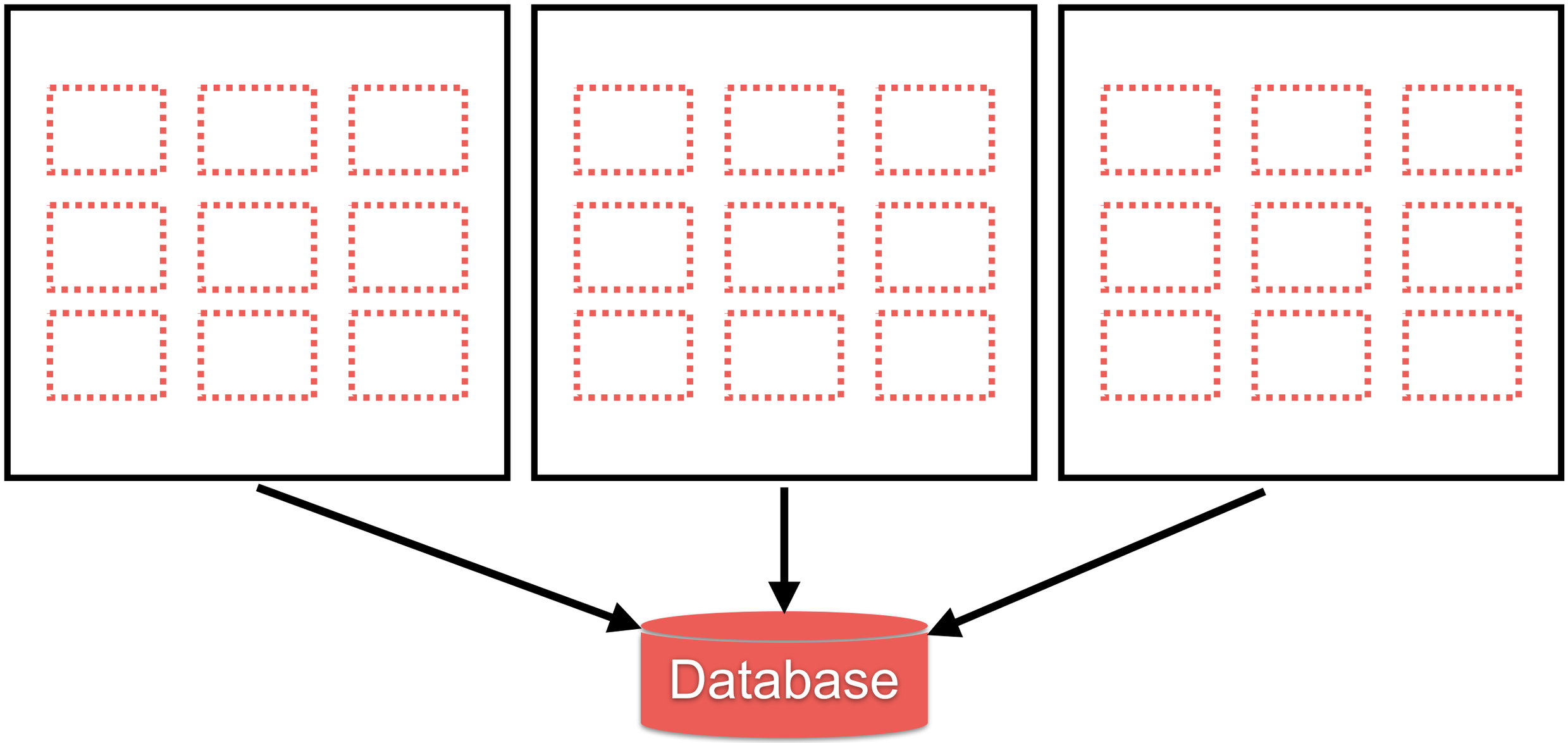
We need to improve

We need to get better

Modules



How to scale ?



What happens when we need more servers ?

What if we don’t use all modules equally ?

How can we update individual models/database ?

Do all modules need to use the same Database, Language and Runtime … ?

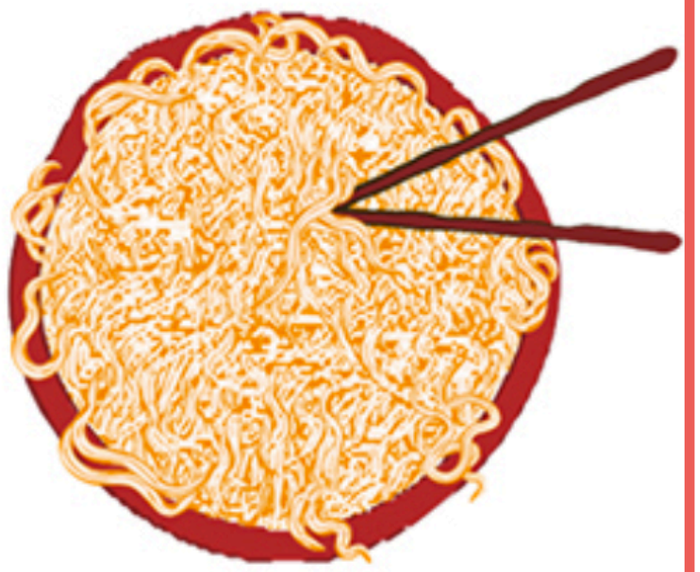
We need to improve

We need to get better

SOA

1990s and earlier

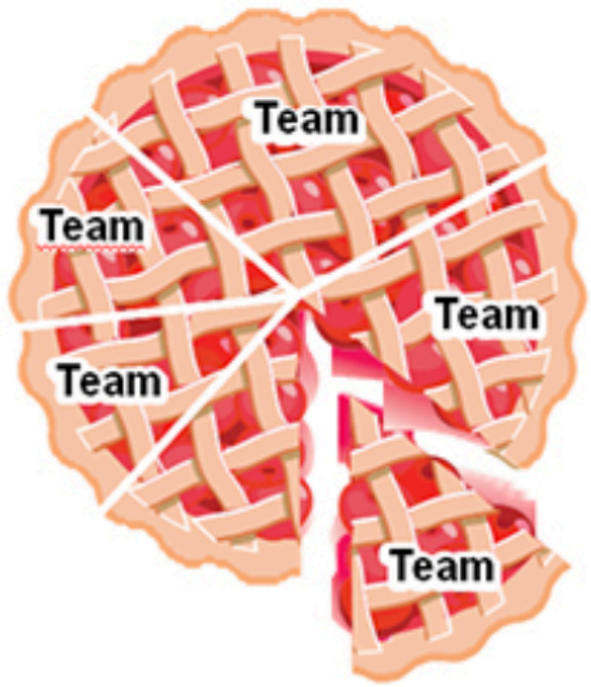
Pre-SOA (monolithic)
Tight coupling



For a monolith to change, all must agree on each change. Each change has unanticipated effects requiring careful testing beforehand.

2000s

Traditional SOA
Looser coupling



Elements in SOA are developed more autonomously but must be coordinated with others to fit into the overall design.

2010s

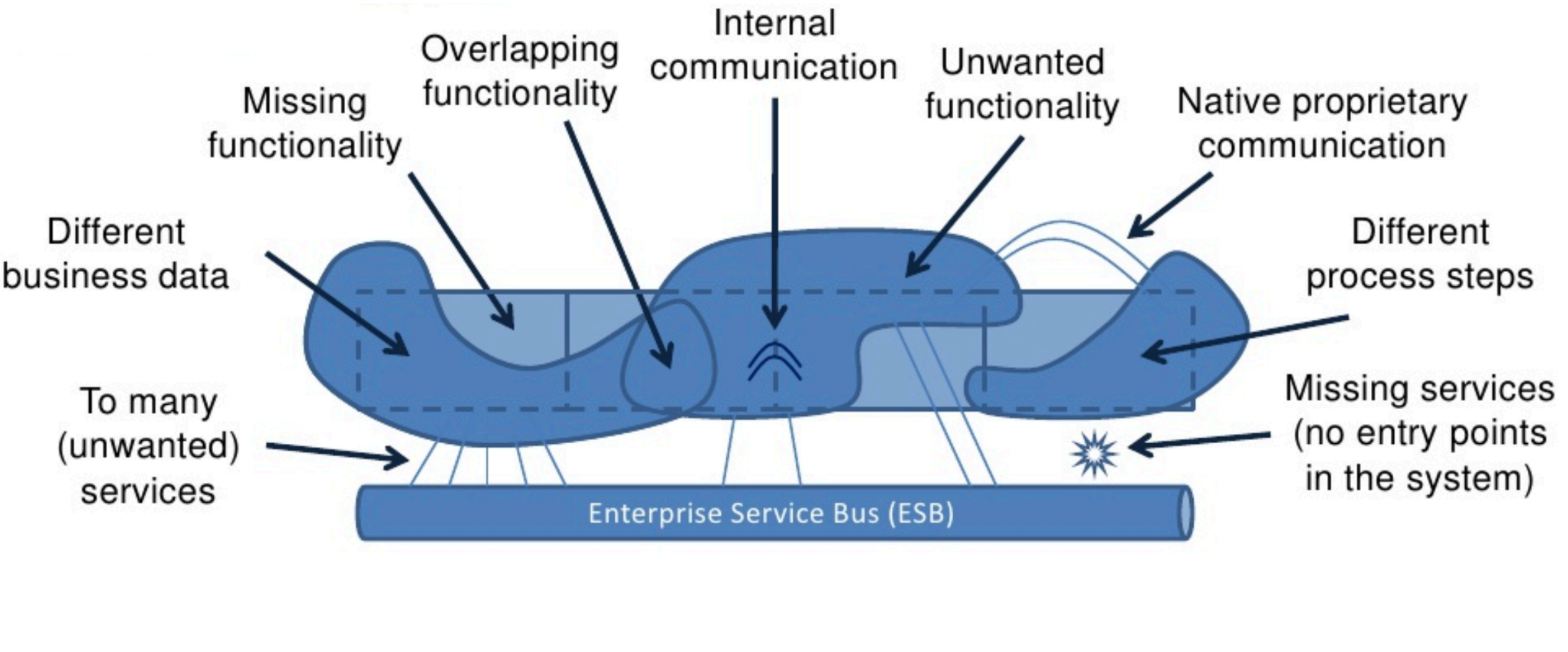
Microservices
Decoupled

Developers can create and activate new microservices without prior coordination with others. Their adherence to MSA principles makes continuous delivery of new or modified services possible.

SOA promise

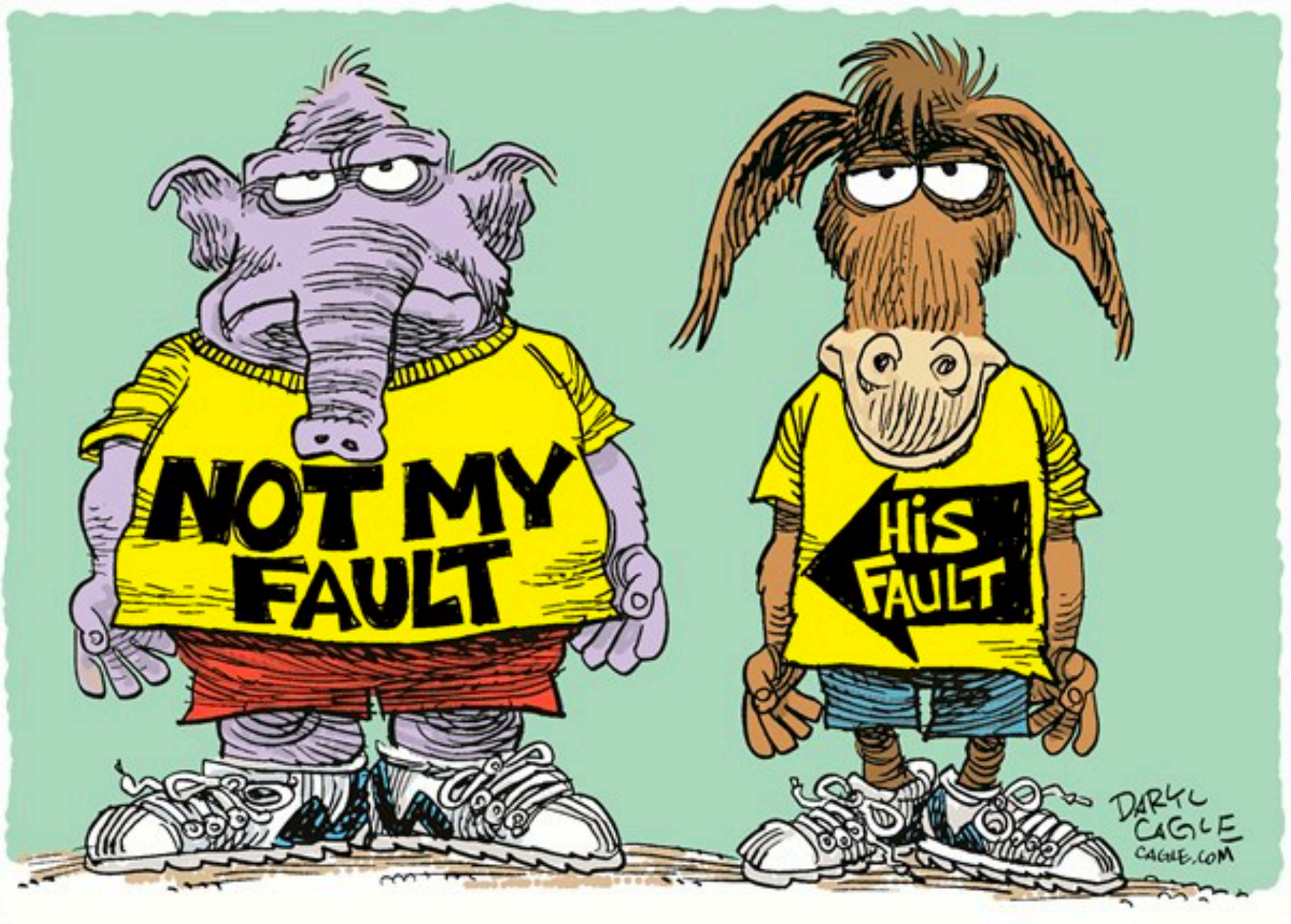
Simpler system
Lowering integration cost
Lowering maintenance cost
Enhancing architectural flexibility
Becoming more agile

Real World



SOA promise

Simpler system
Lowering integration cost
Lowering maintenance cost
Enhancing architectural flexibility
Becoming more agile



We need to improve

We need to get better

Microservices

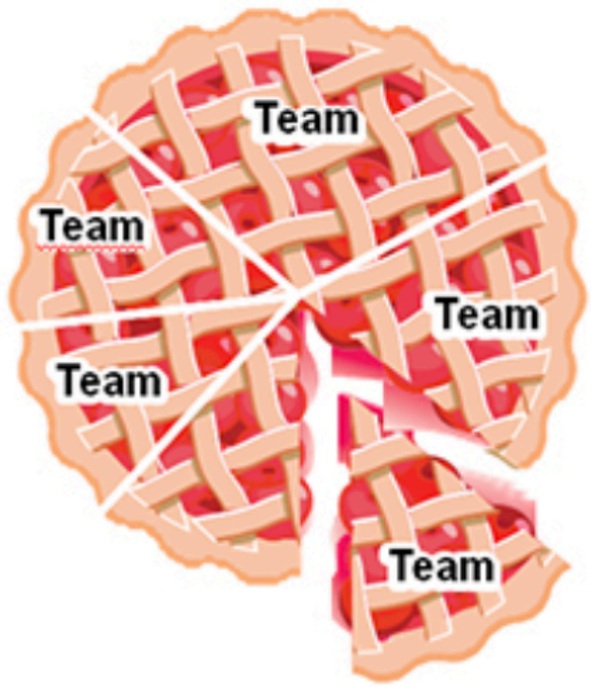
1990s and earlier

Pre-SOA (monolithic)
Tight coupling

For a monolith to change, all must agree on each change. Each change has unanticipated effects requiring careful testing beforehand.

2000s

Traditional SOA
Looser coupling



Elements in SOA are developed more autonomously but must be coordinated with others to fit into the overall design.

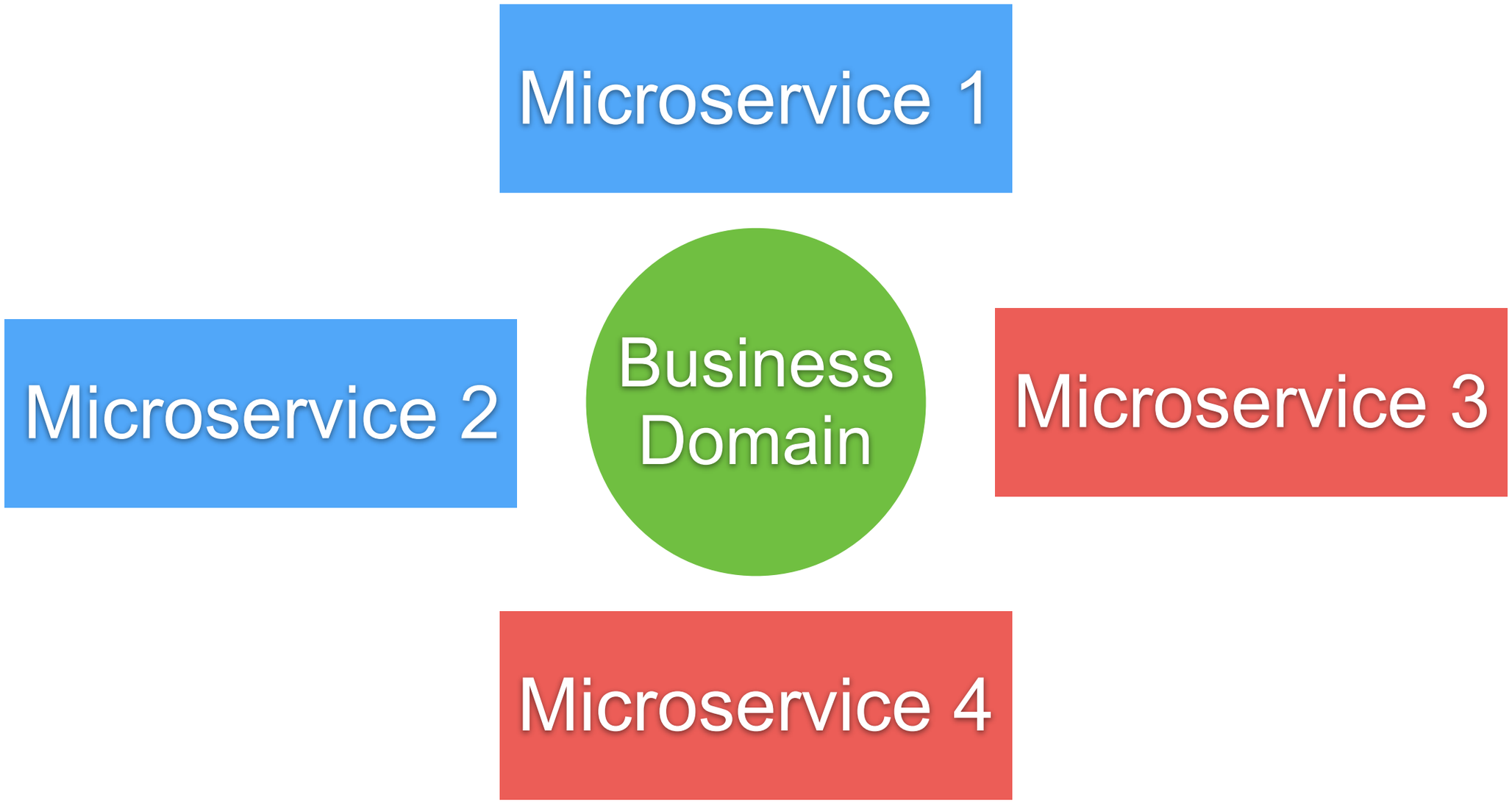
2010s

Microservices
Decoupled

Developers can create and activate new microservices without prior coordination with others. Their adherence to MSA principles makes continuous delivery of new or modified services possible.

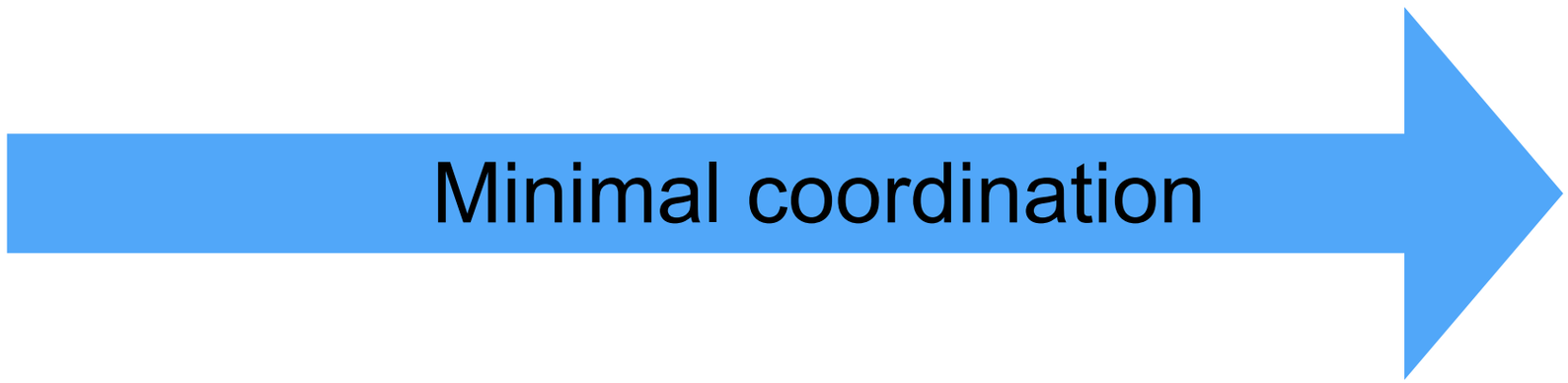
Microservice

Microservice

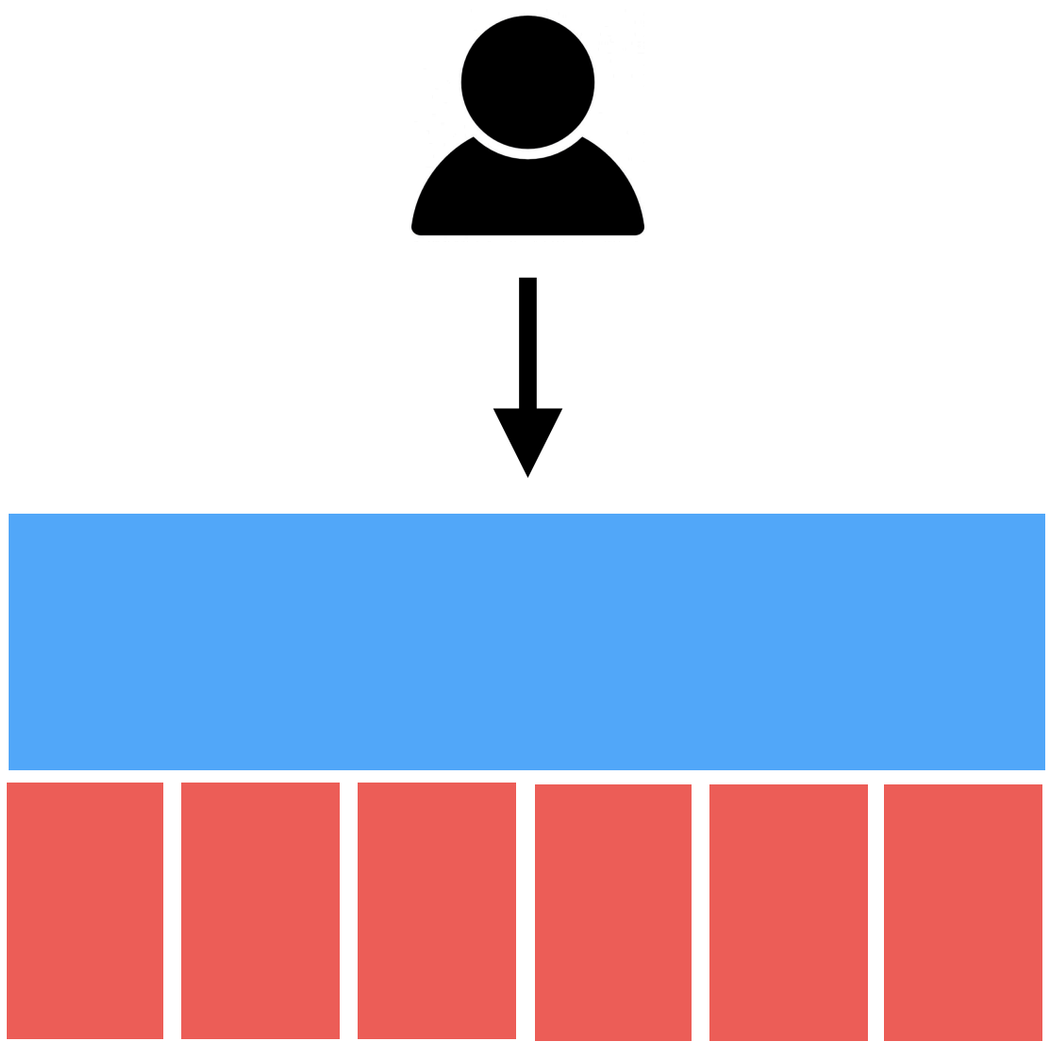


Microservice

Application Development Team



Microservice



Microservice history

1990

Continuous **I**ntegration

2001

Agile Manifesto

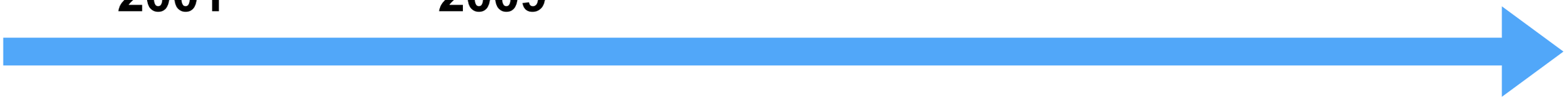
2006

Amazon EC2

2009

DevOps

Java EE 6



Microservice history

1990

Continuous **I**ntegration

2001

Agile Manifesto

2006

Amazon EC2

2009

Java EE 6

DevOps

2010

Netflix + AWS

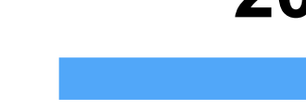
2012

Ribbon

Hystrix

Eureka

Microservices



Microservice history

1990
Continuous **I**ntegration

2001
Agile Manifesto

2006
Amazon EC2

2009
Java EE 6
DevOps

2010
Netflix + AWS

2012
Ribbon
Hystrix
Eureka
Microservices

2013
Container
Docker

2013
Spring Boot

Microservice history

1990
Continuous **I**ntegration

2001
Agile Manifesto

2006
Amazon EC2

2009
Java EE 6
DevOps

2010
Netflix + AWS

2012
Ribbon
Hystrix
Eureka
Microservices

2013
Container
Docker

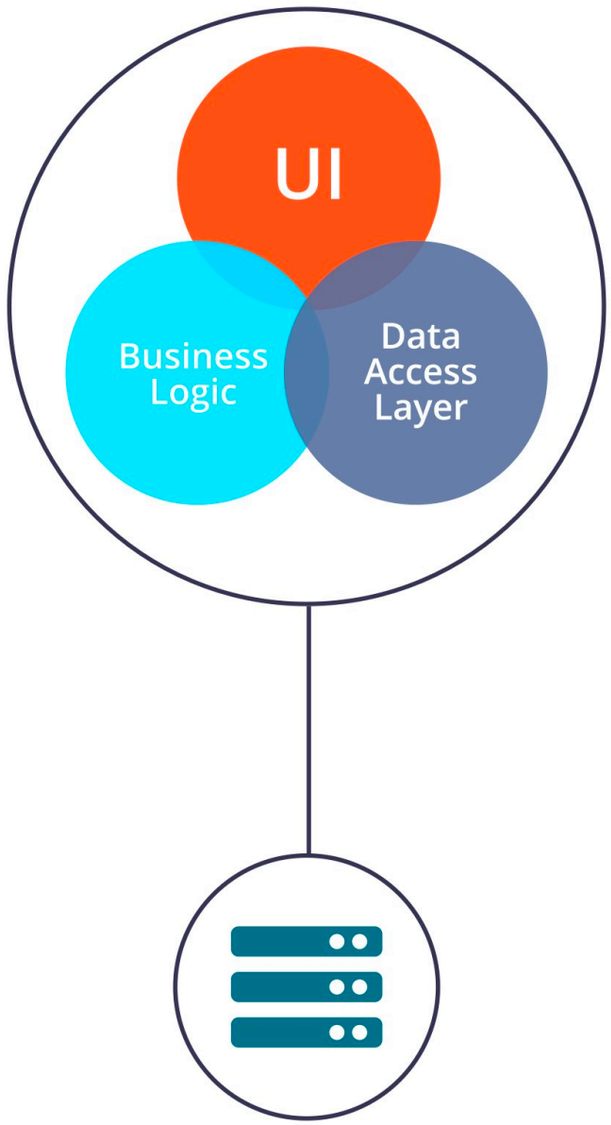
2013
Spring Boot

2014
Adopt Microservices

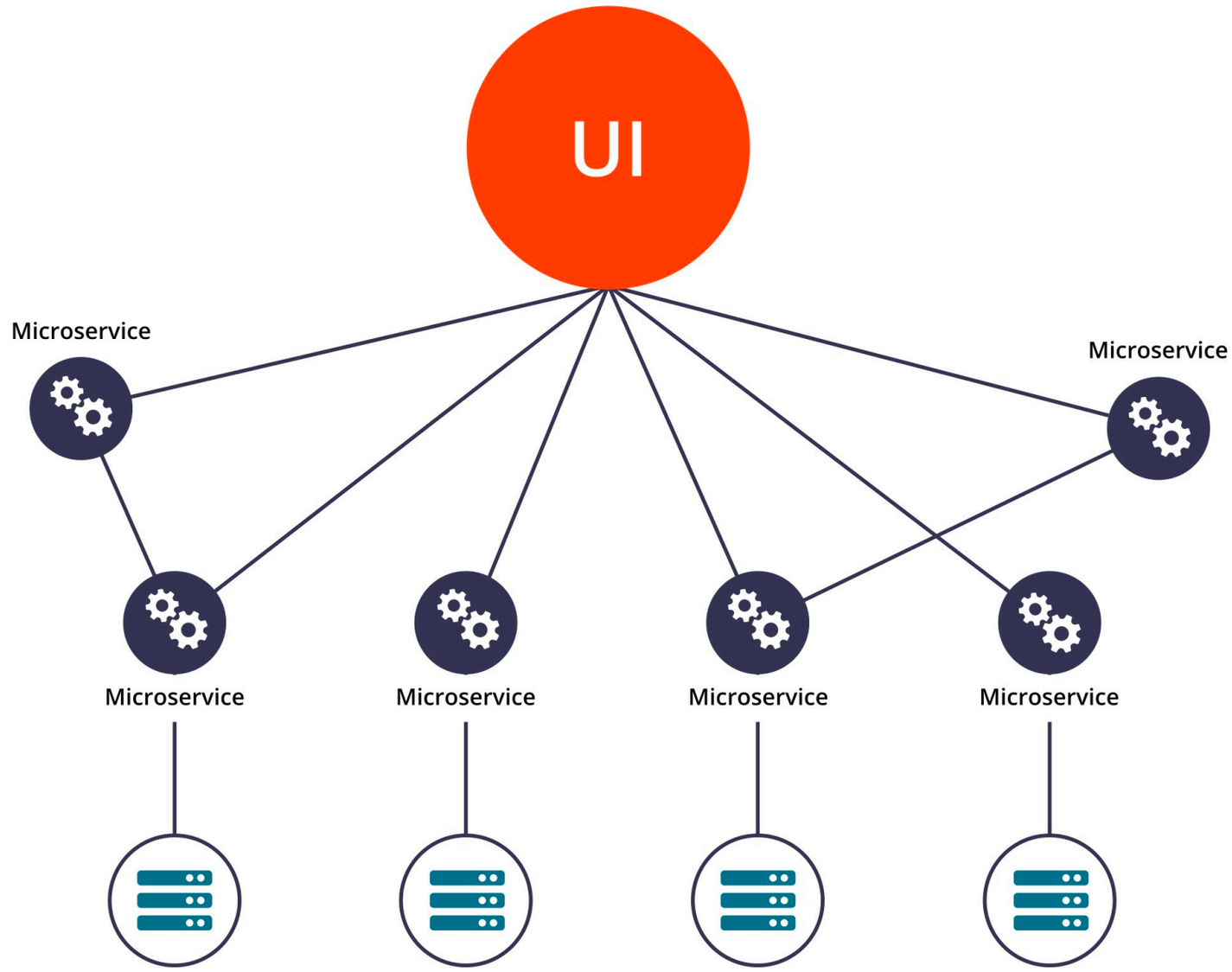
Microservice history

Microservice

Small, Do one thing (Single Responsibility)
Modular
Easy to **understand**
Easy to **develop**
Easy to **deploy**
Easy to **maintain**
Scale independently

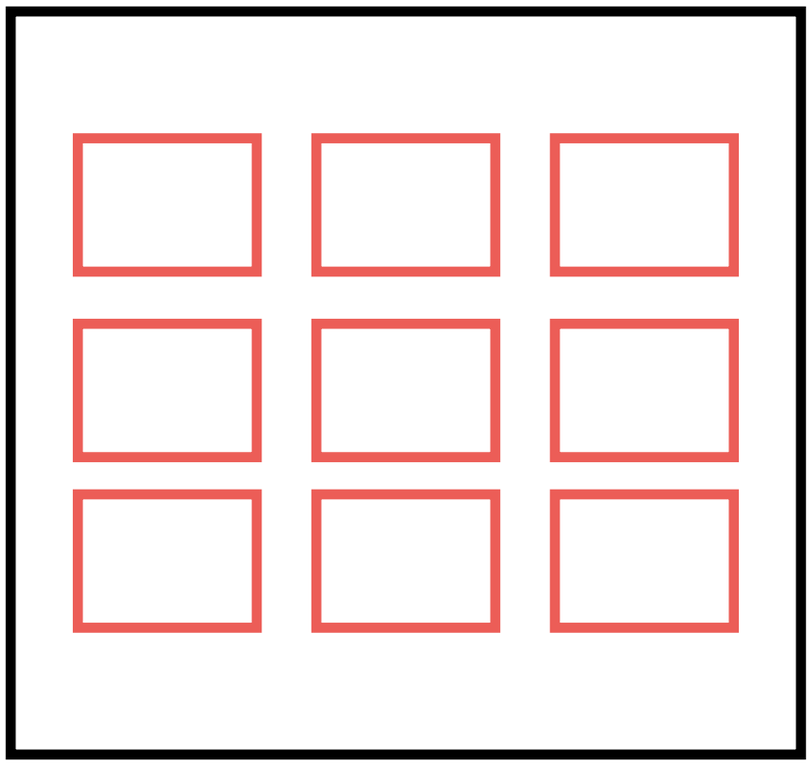


Monolithic Architecture

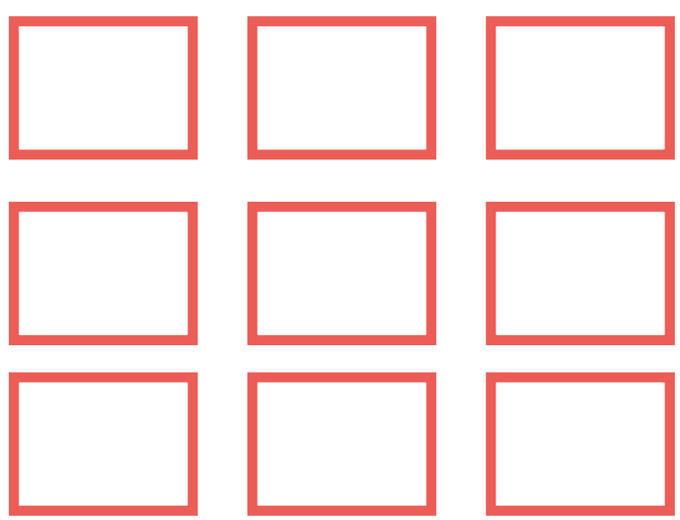


Microservice Architecture

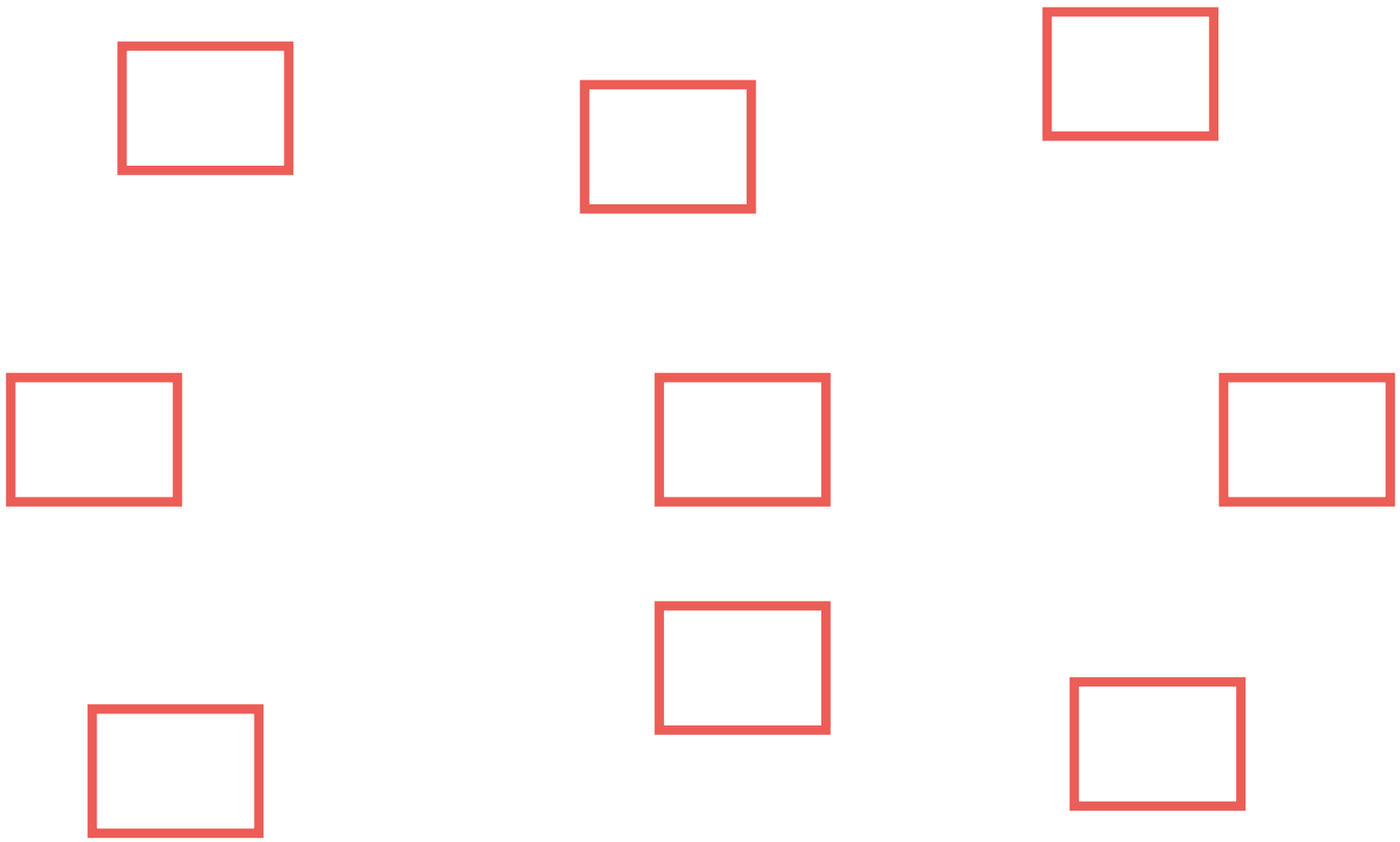
Microservice



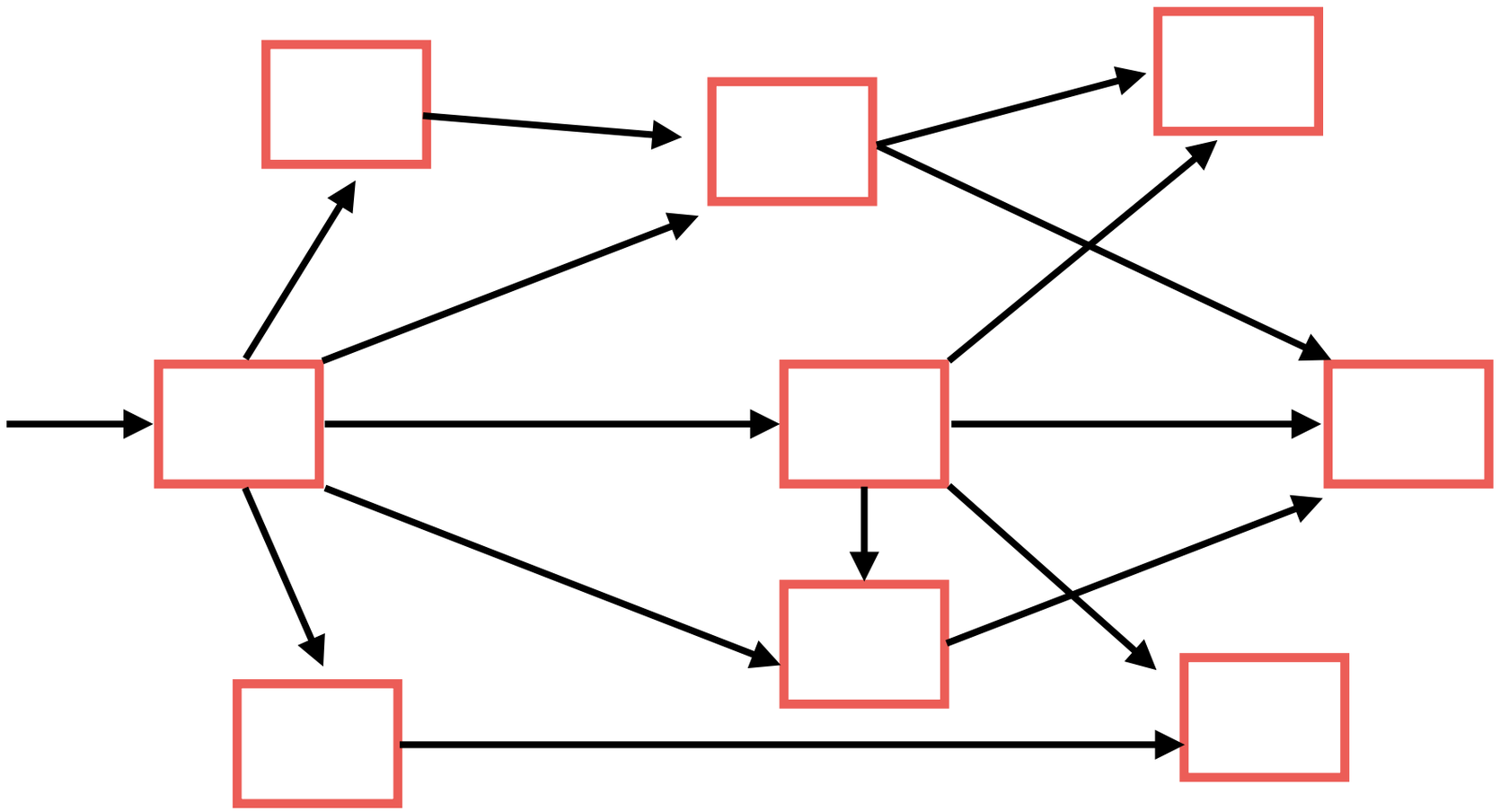
Microservice



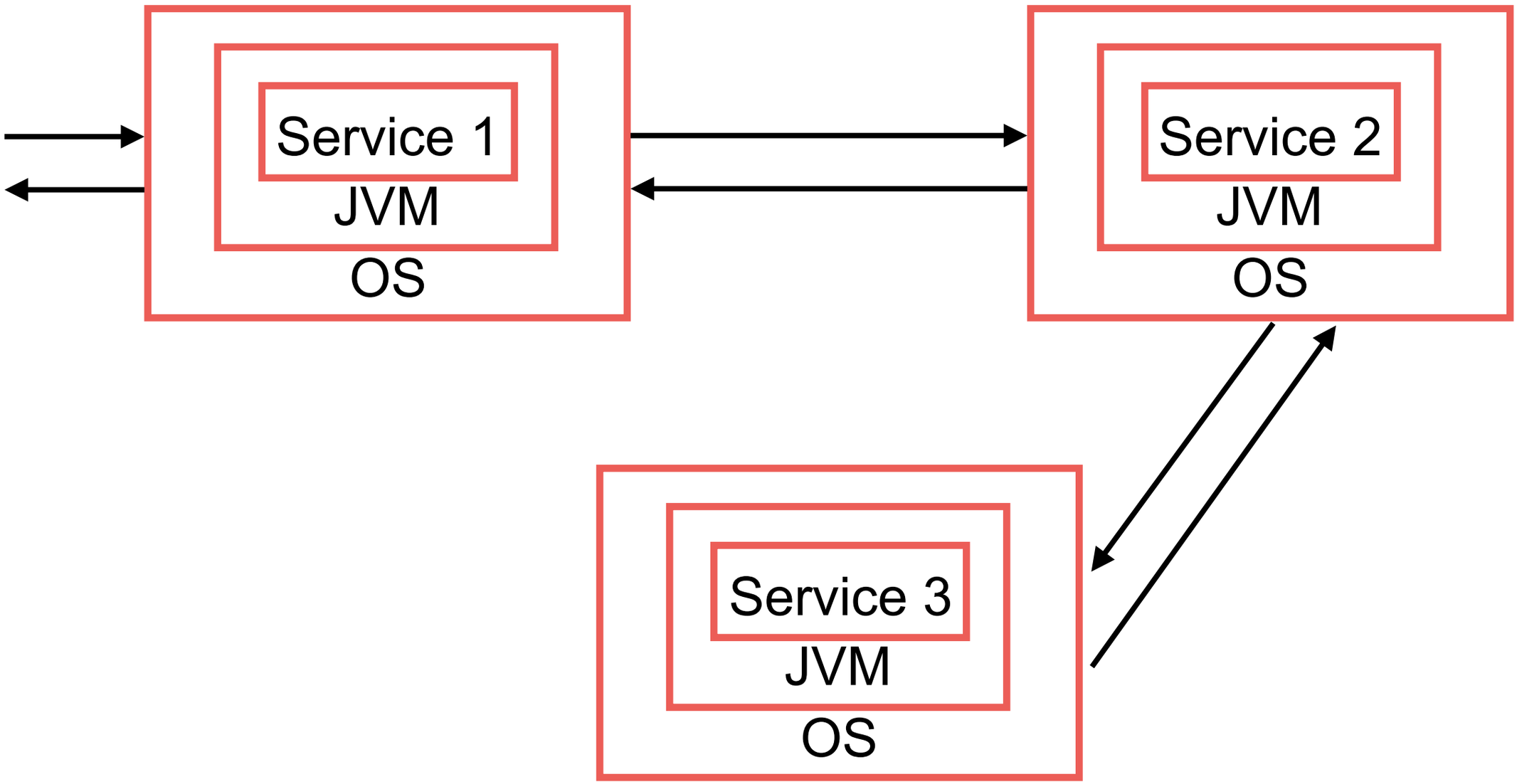
Microservice



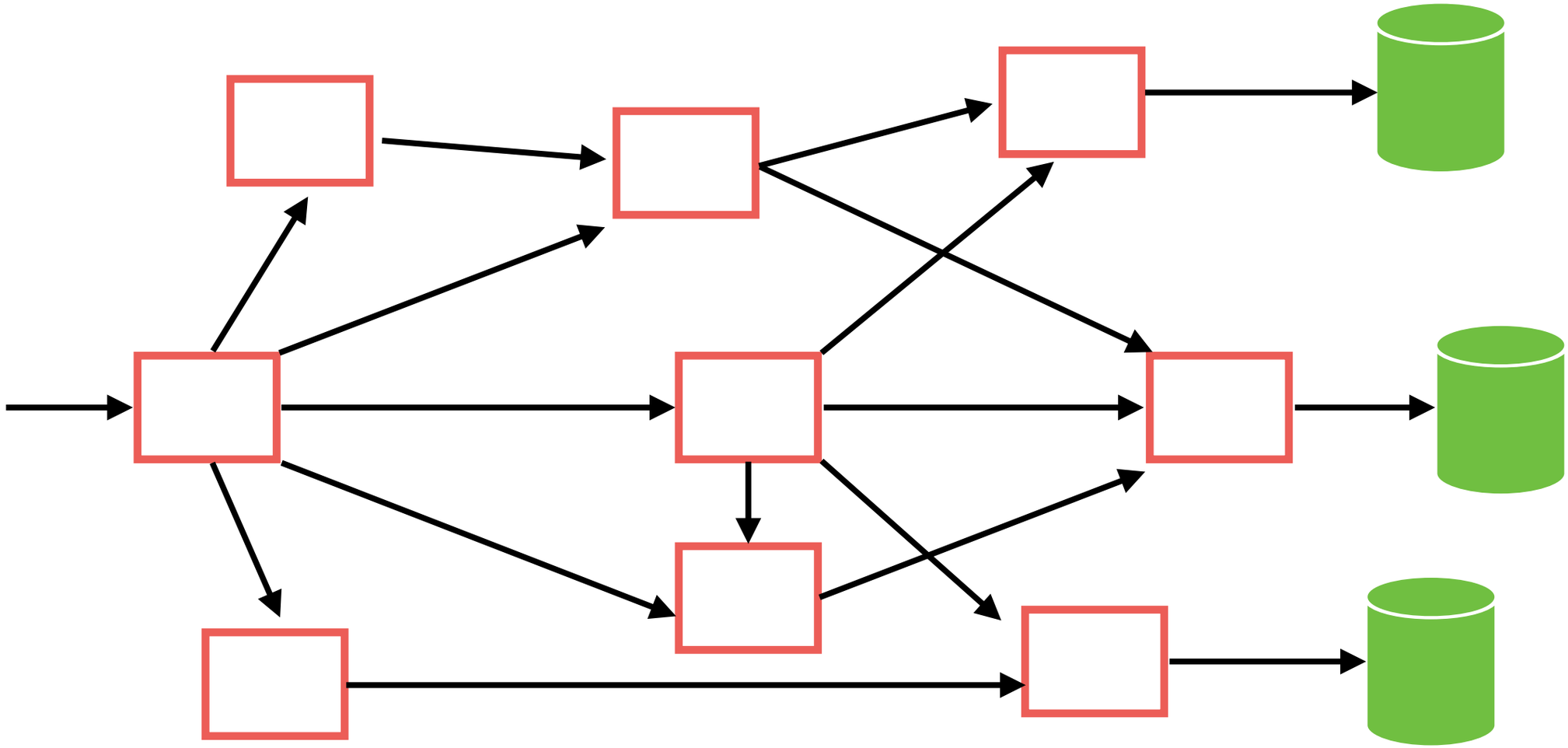
Microservice == Distributed



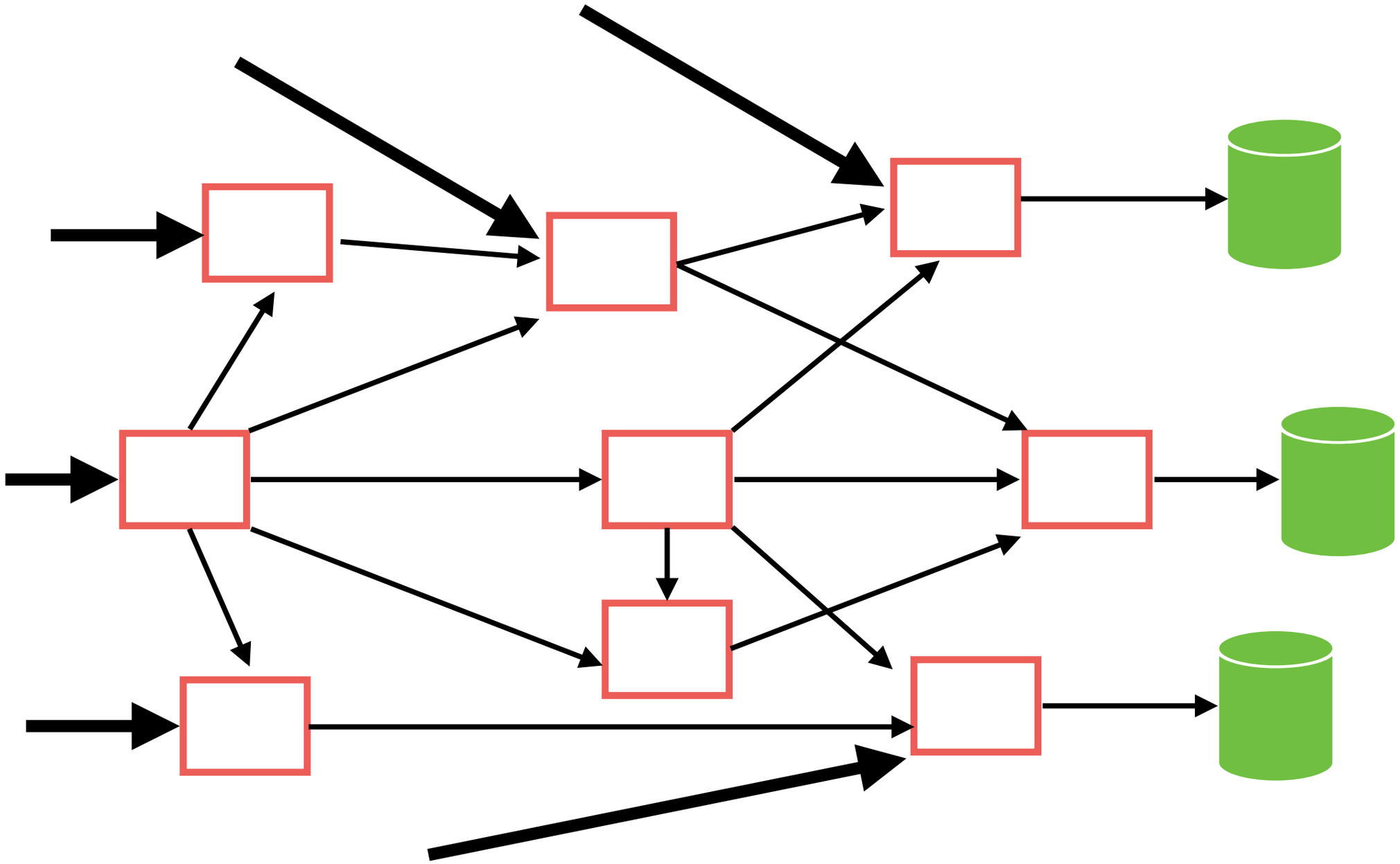
Network of services



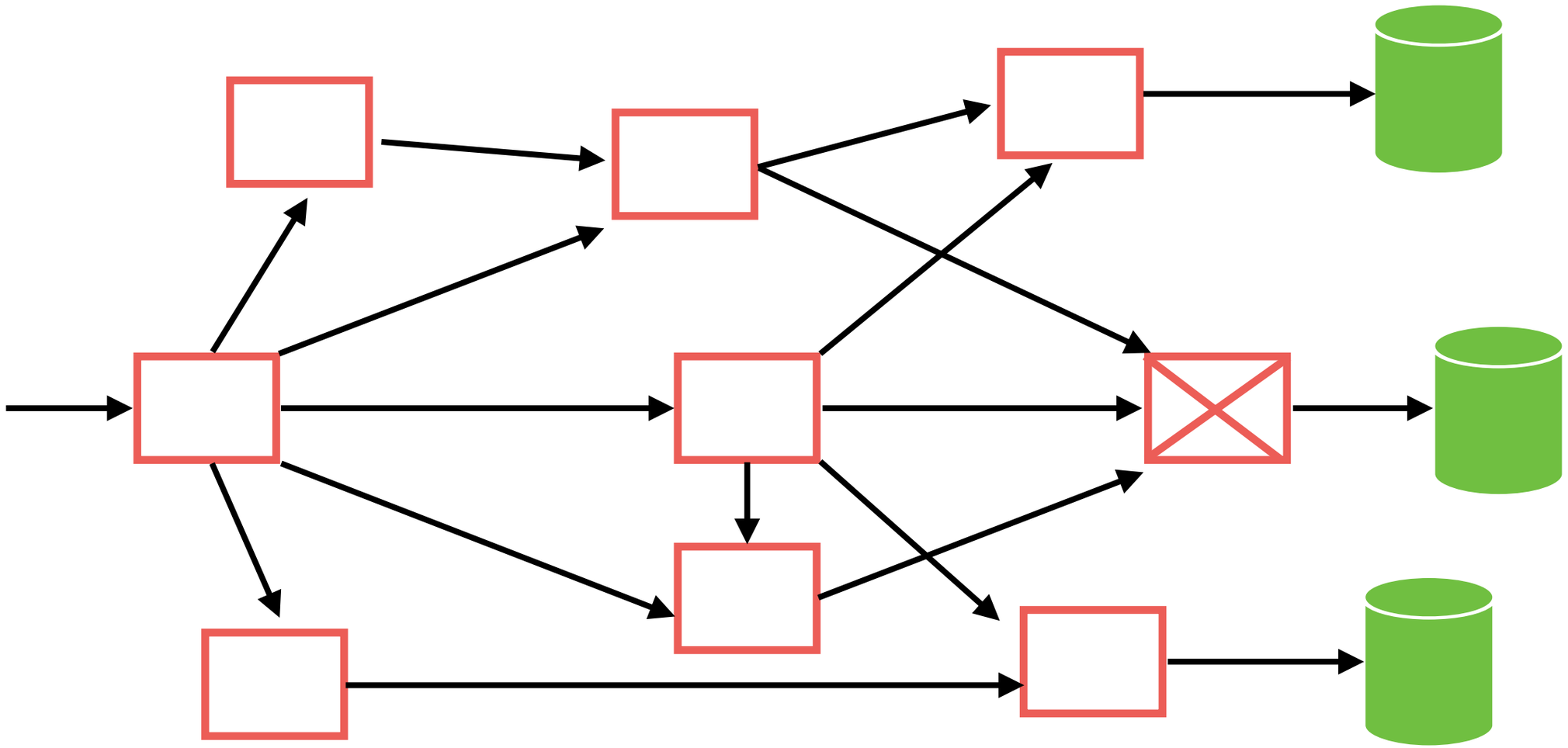
Own data



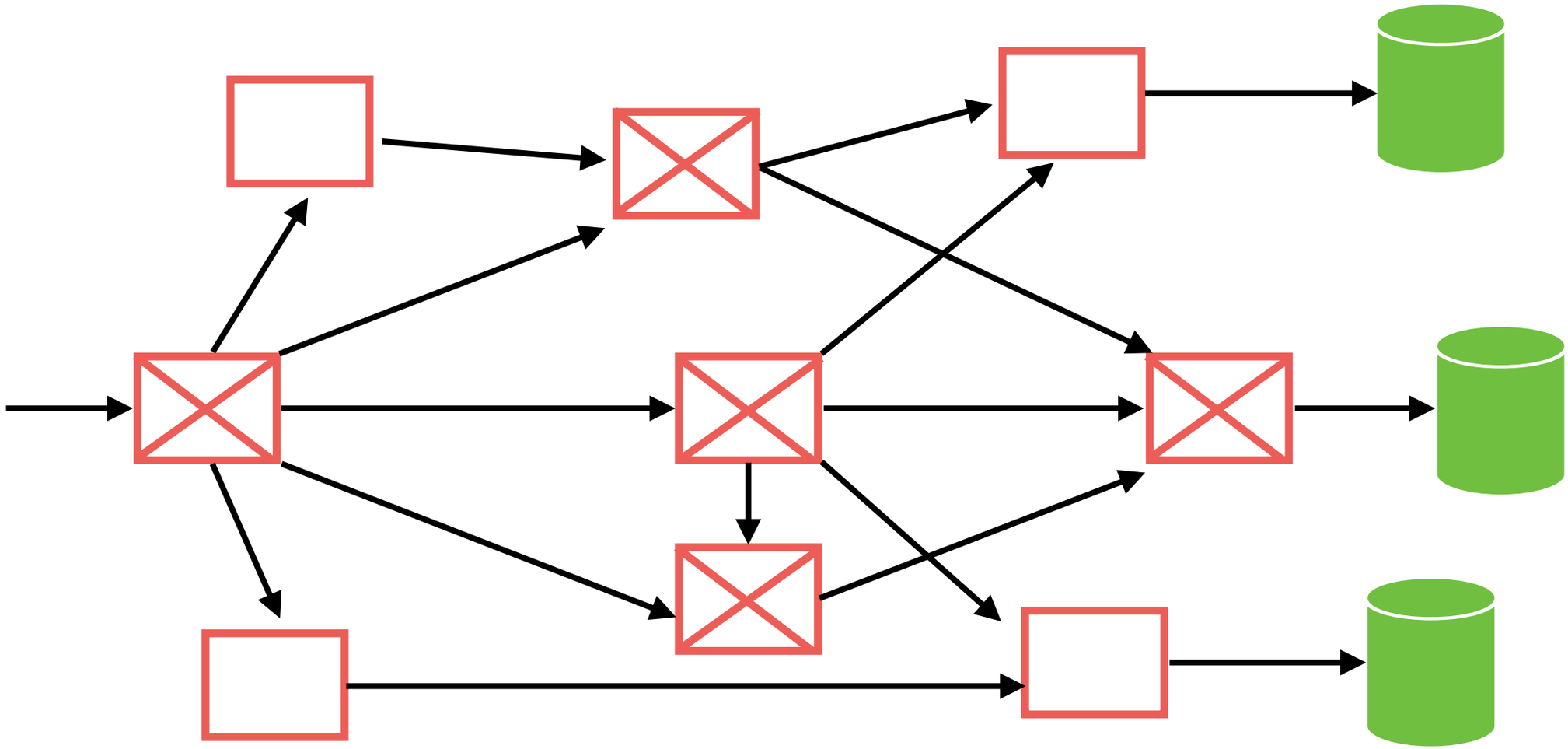
Multiple entry points



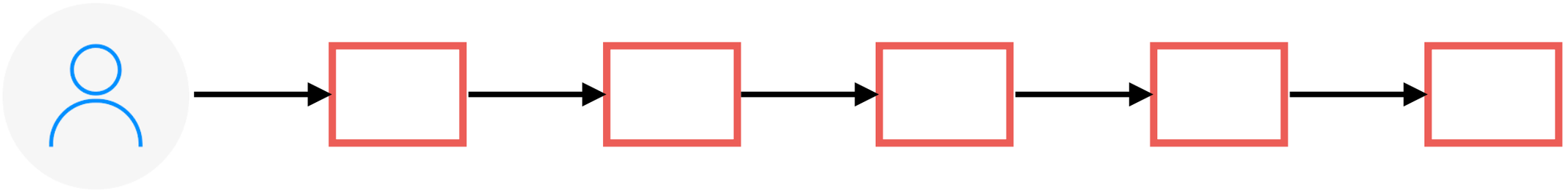
Failure !!



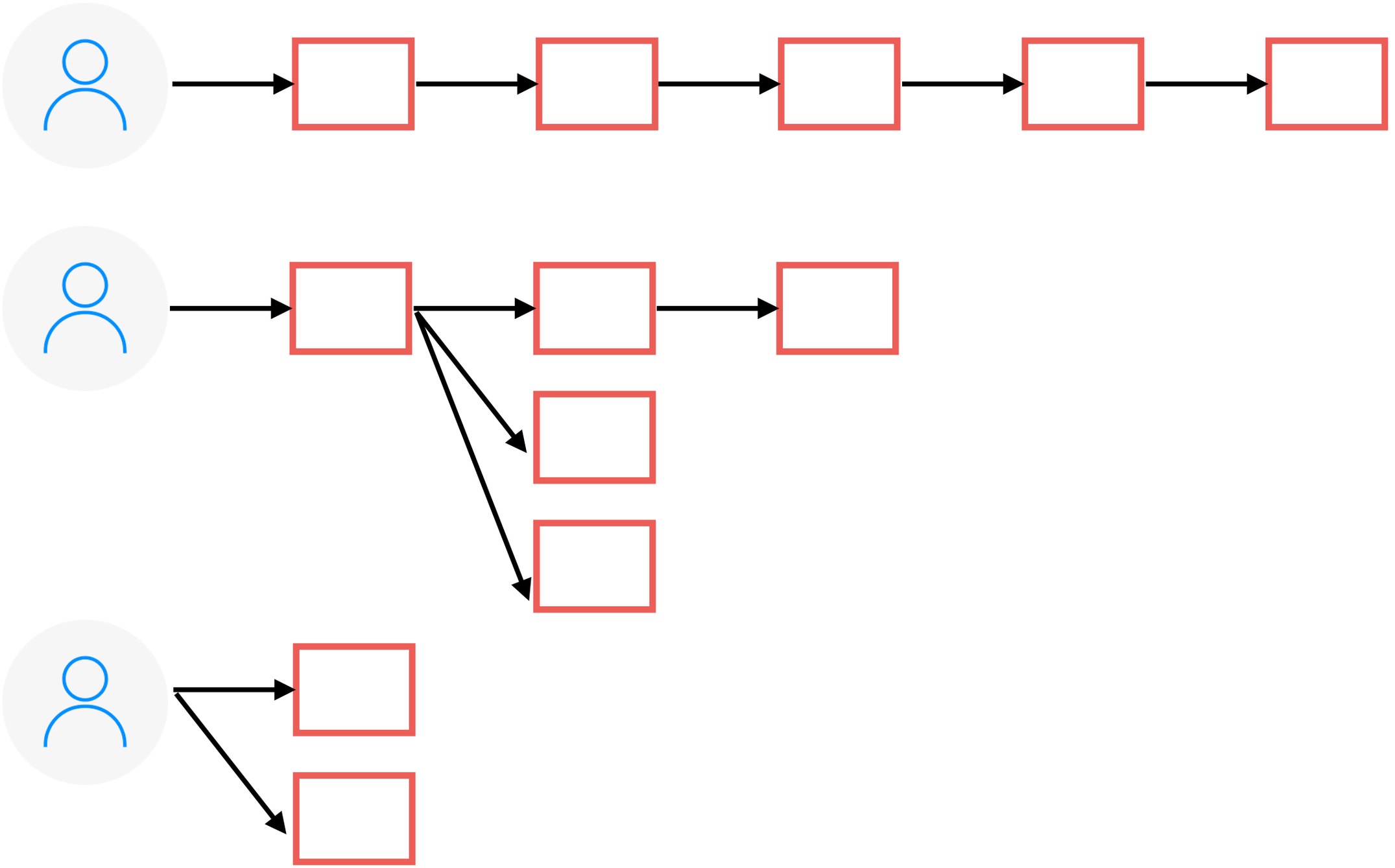
Cascading Failure !!



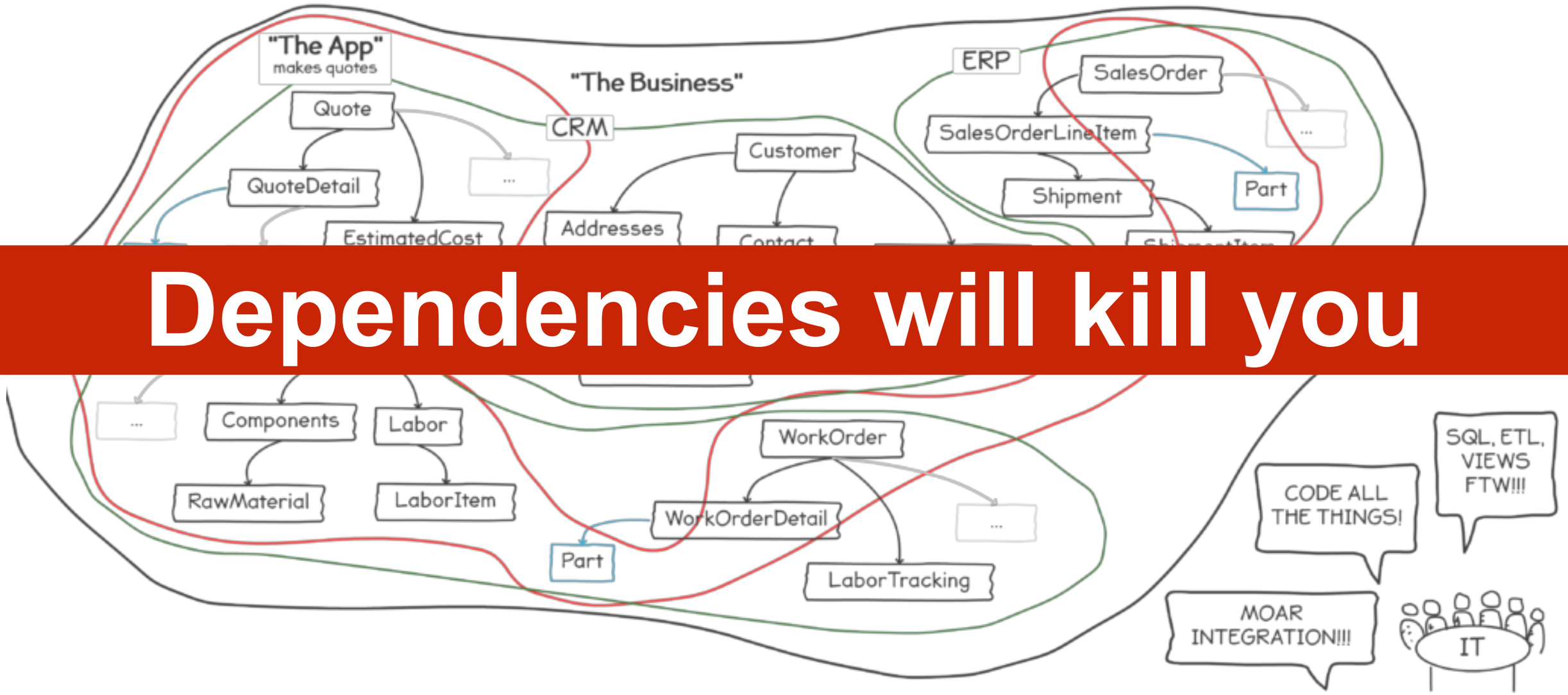
Service Dependencies



Service Dependencies



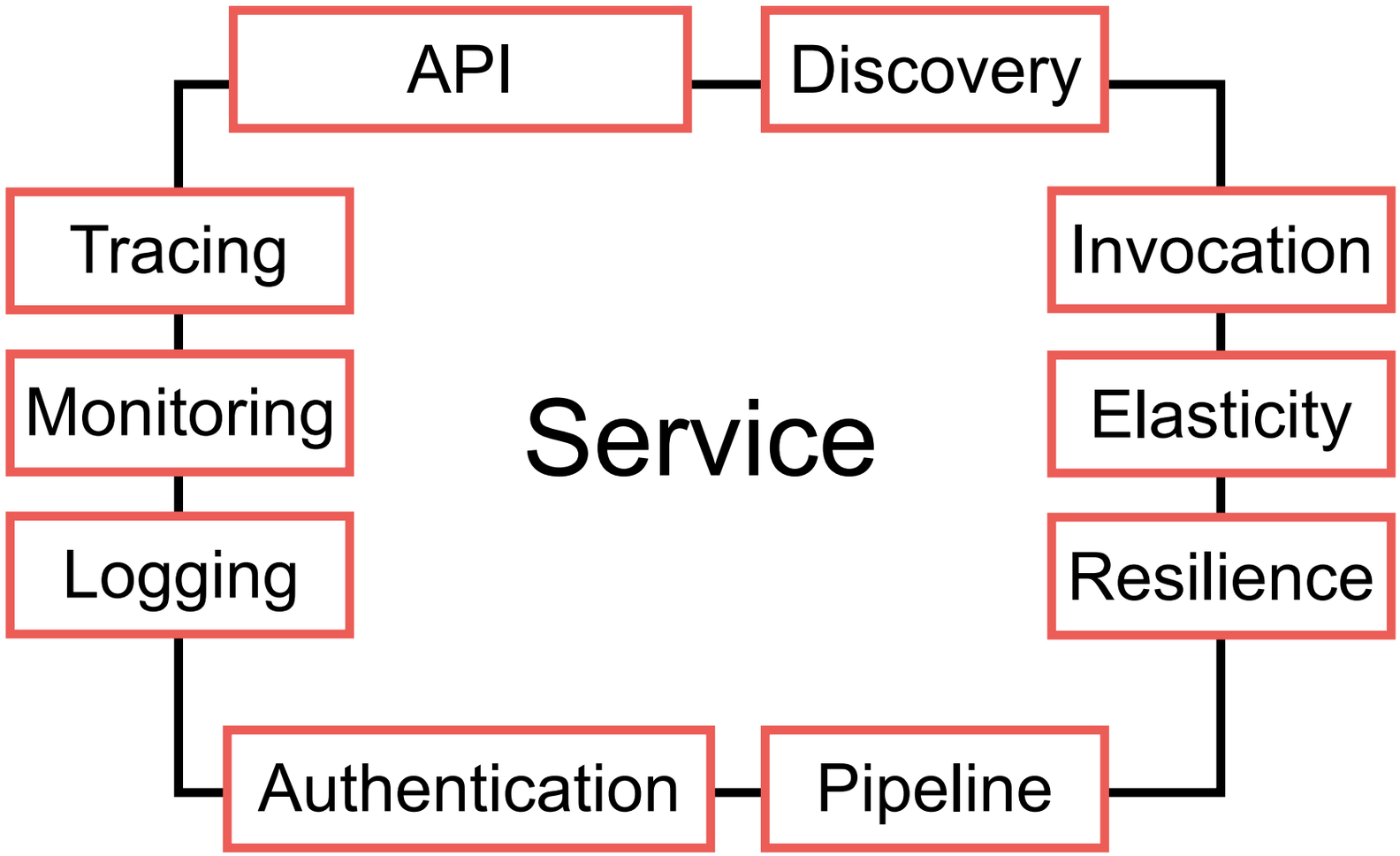
Microservice Hell



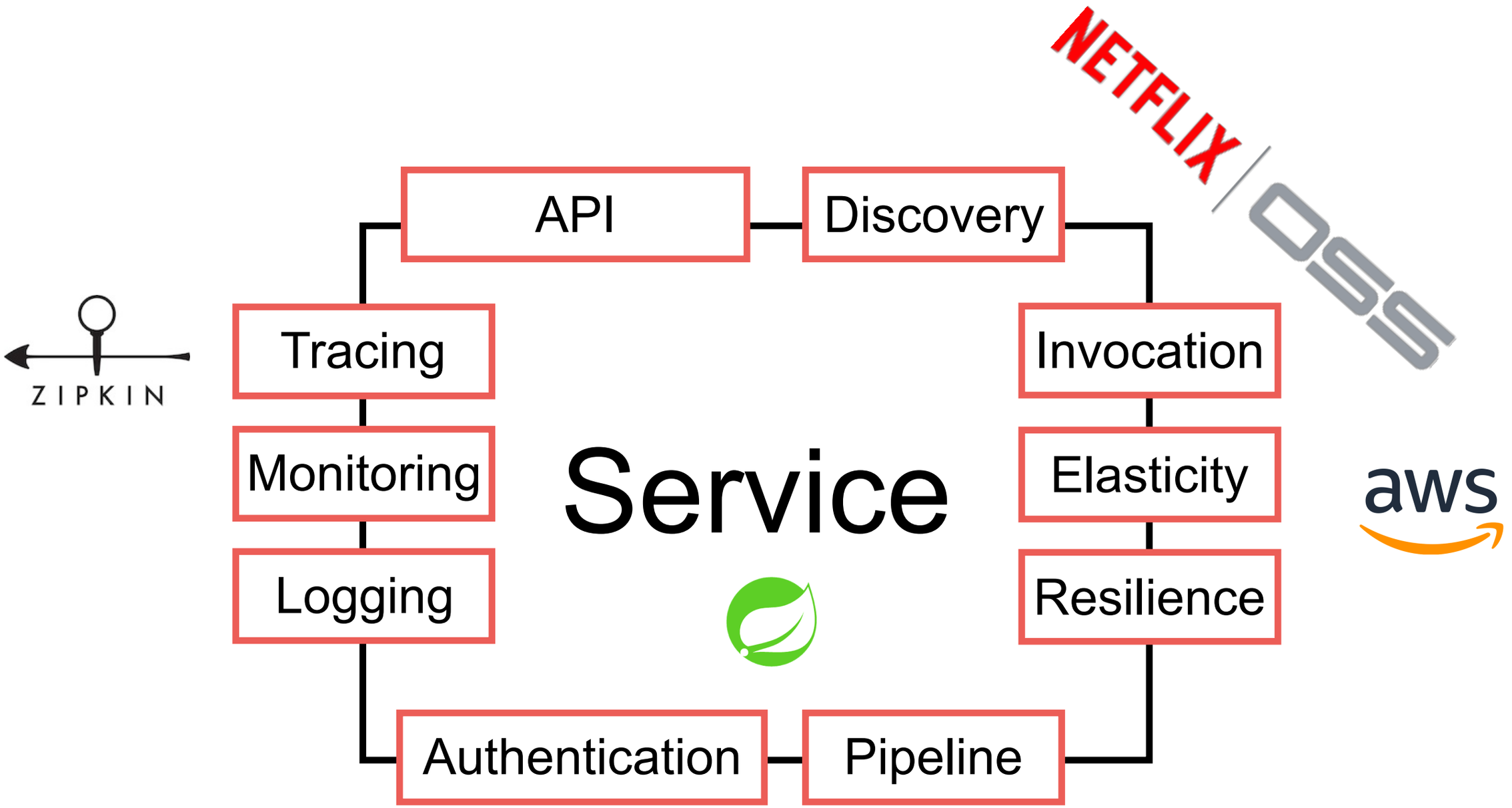
Dependencies will kill you

Properties of Microservice

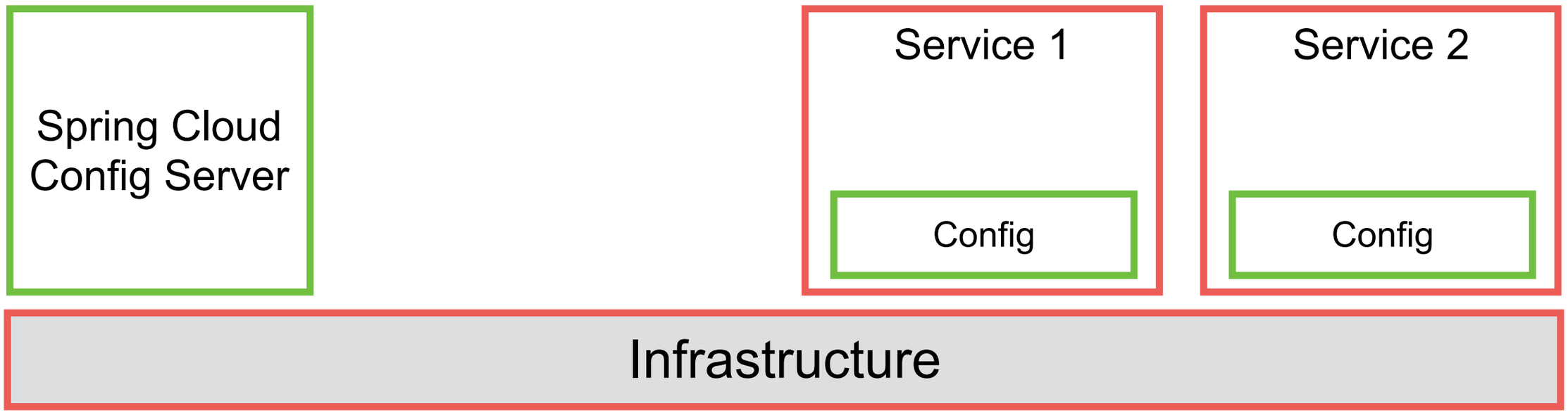
Properties of Microservice



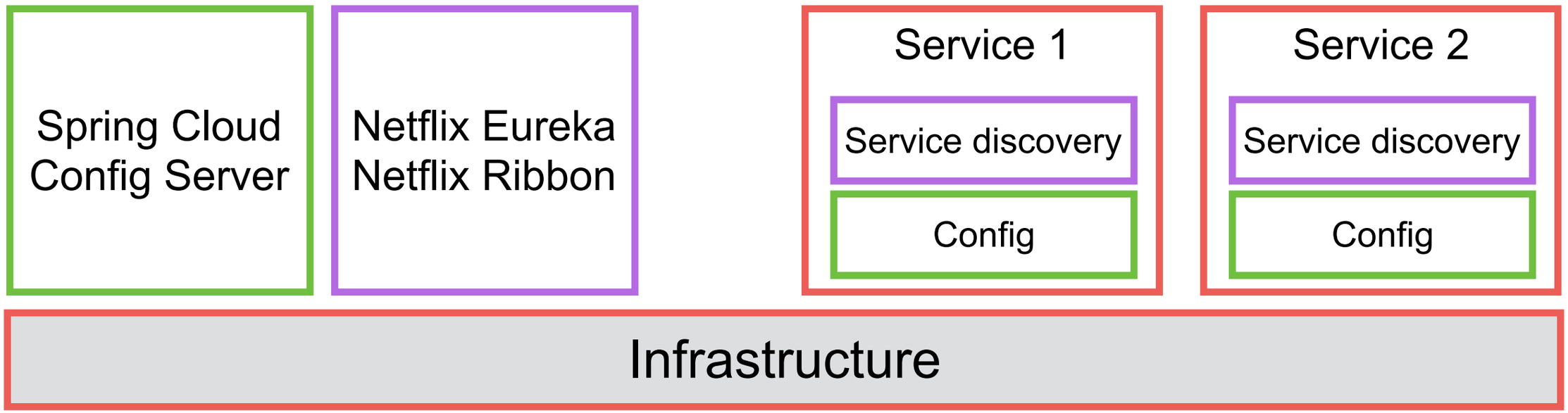
Microservice 1.0



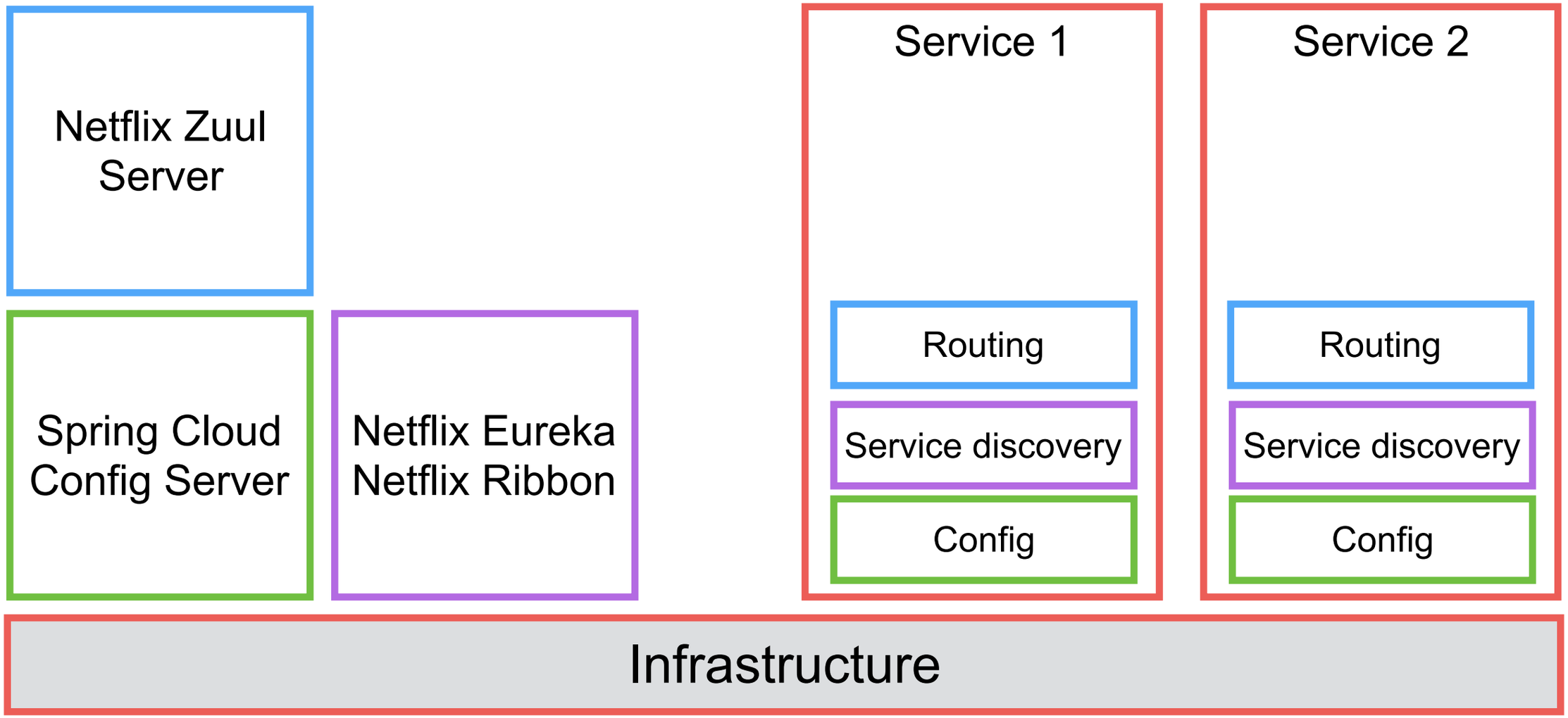
Configuration



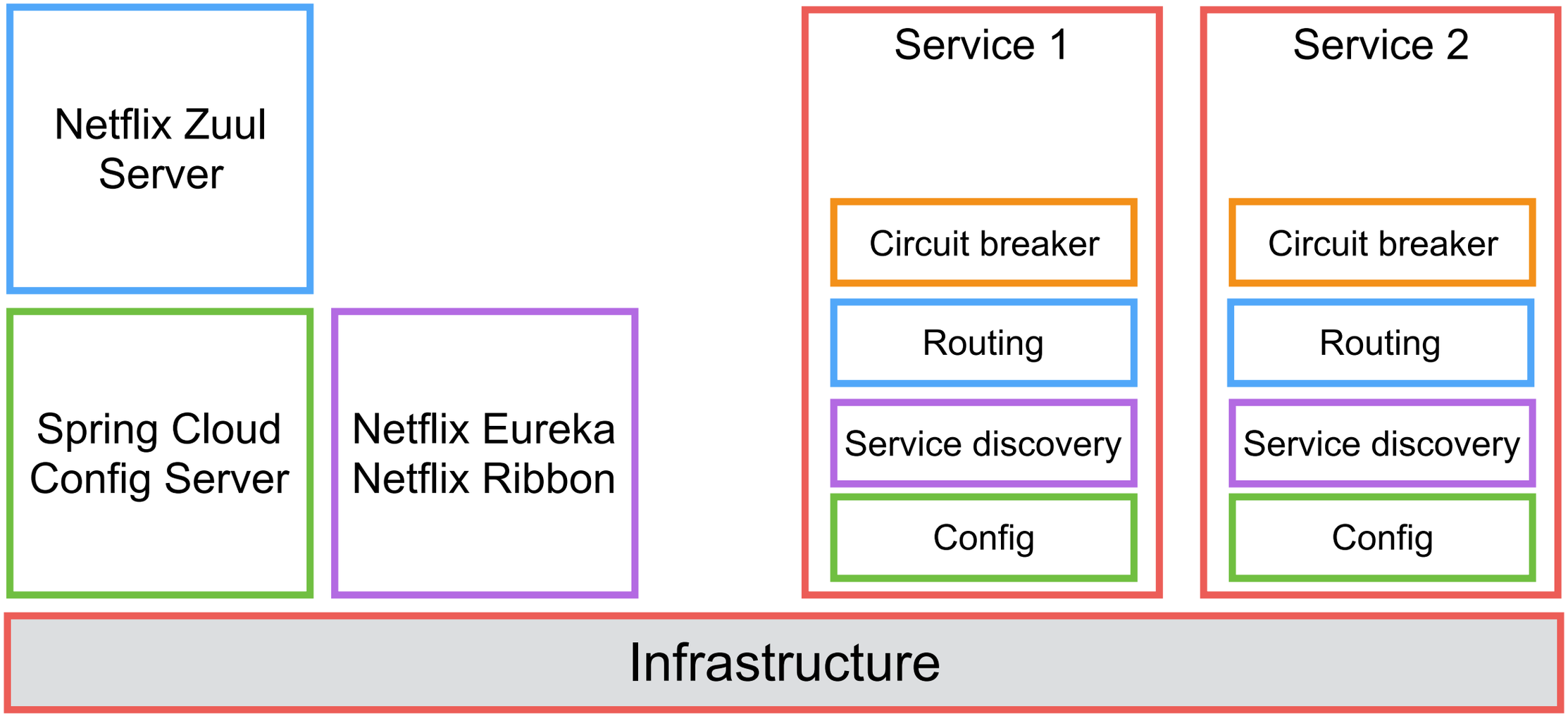
Service Discovery



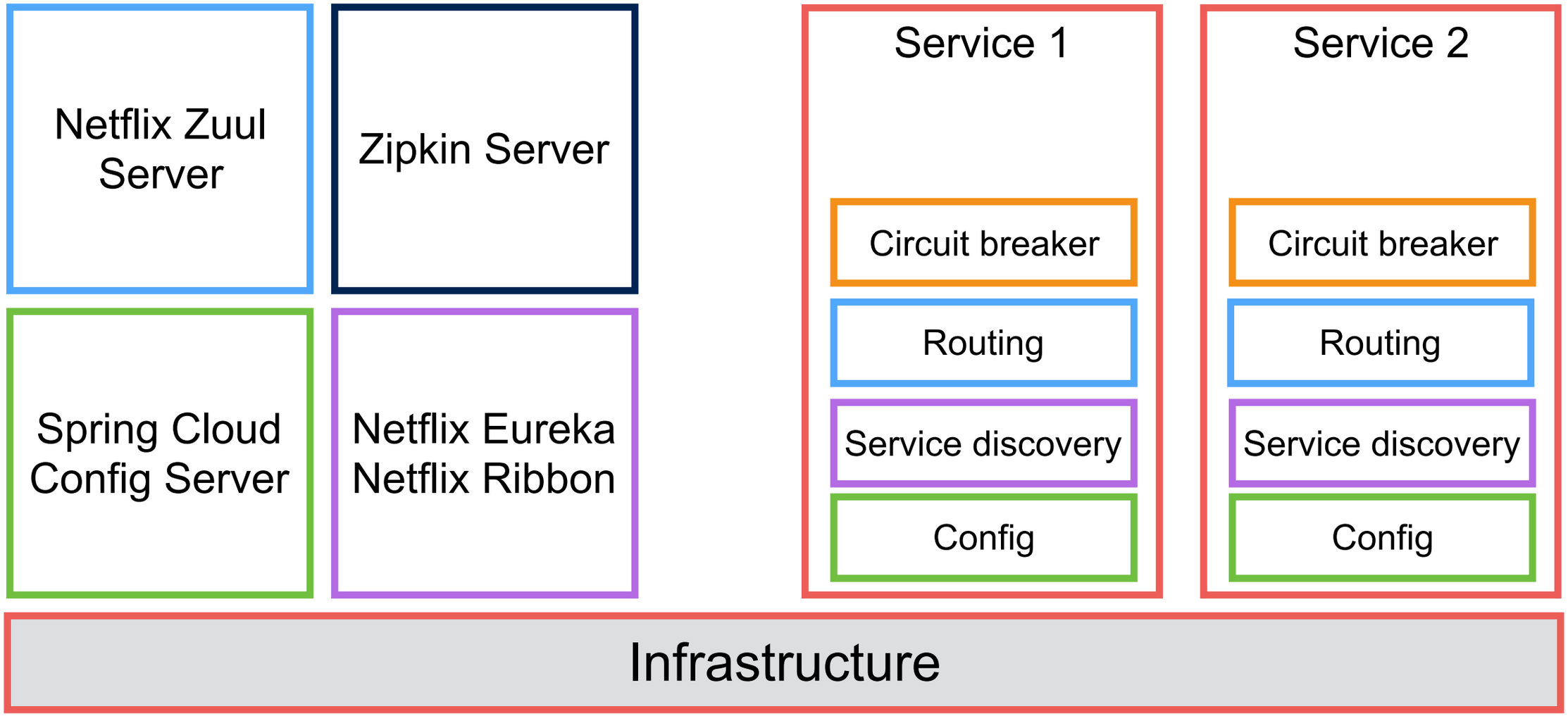
Dynamic Routing



Fault Tolerance



Tracing and Visibility

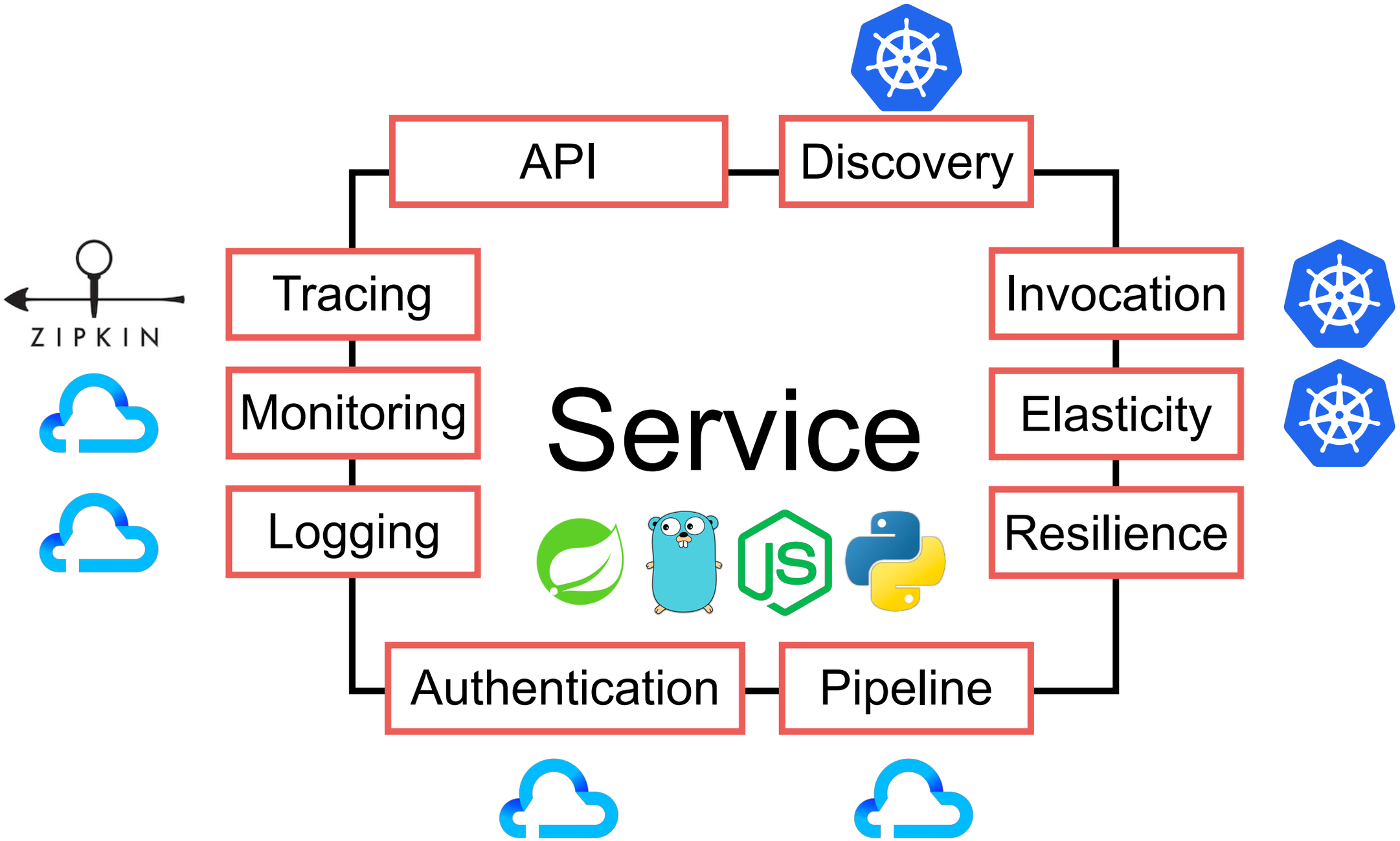


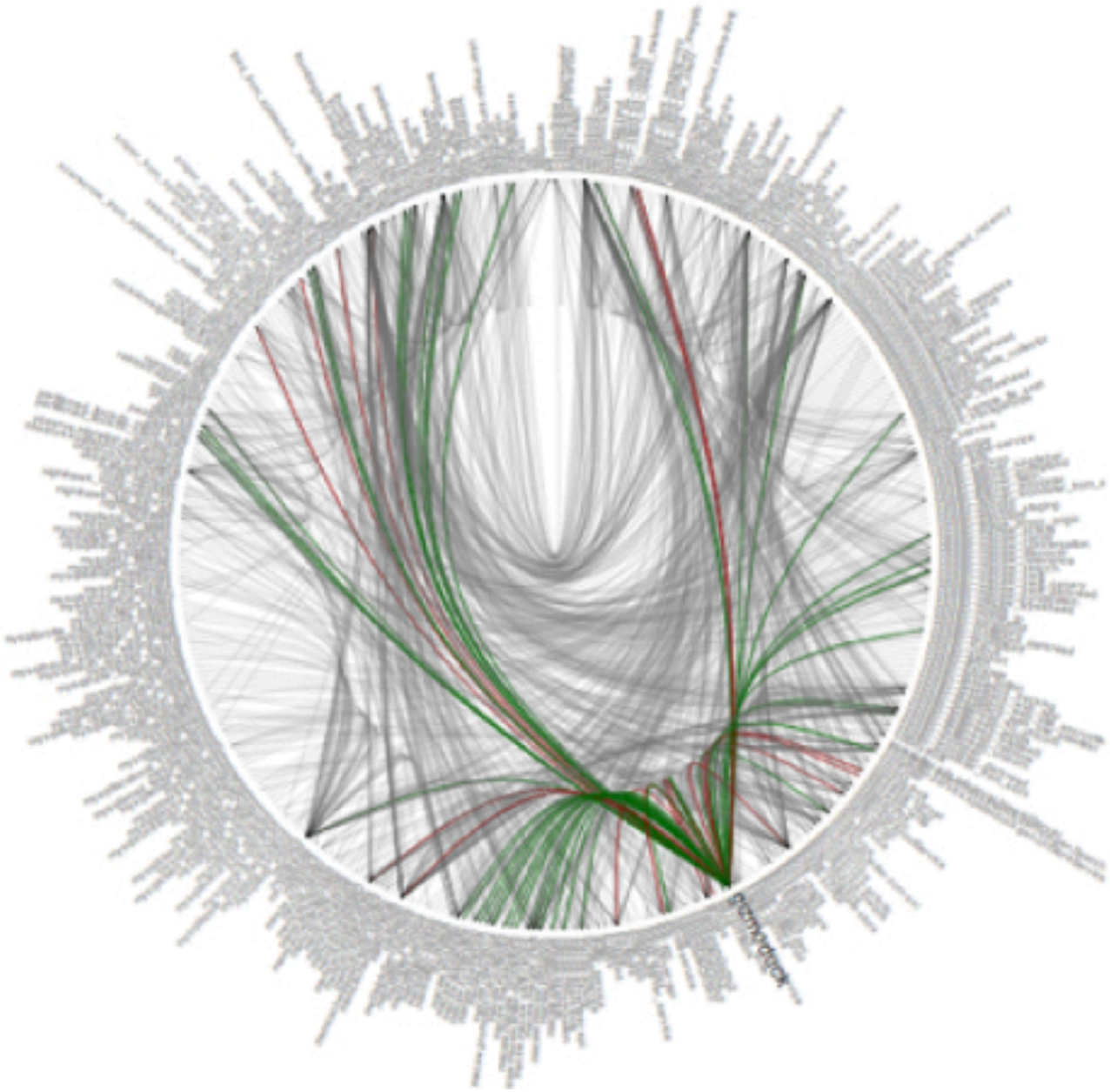
Microservice 1.0

JVM only
Add libraries to your code/service

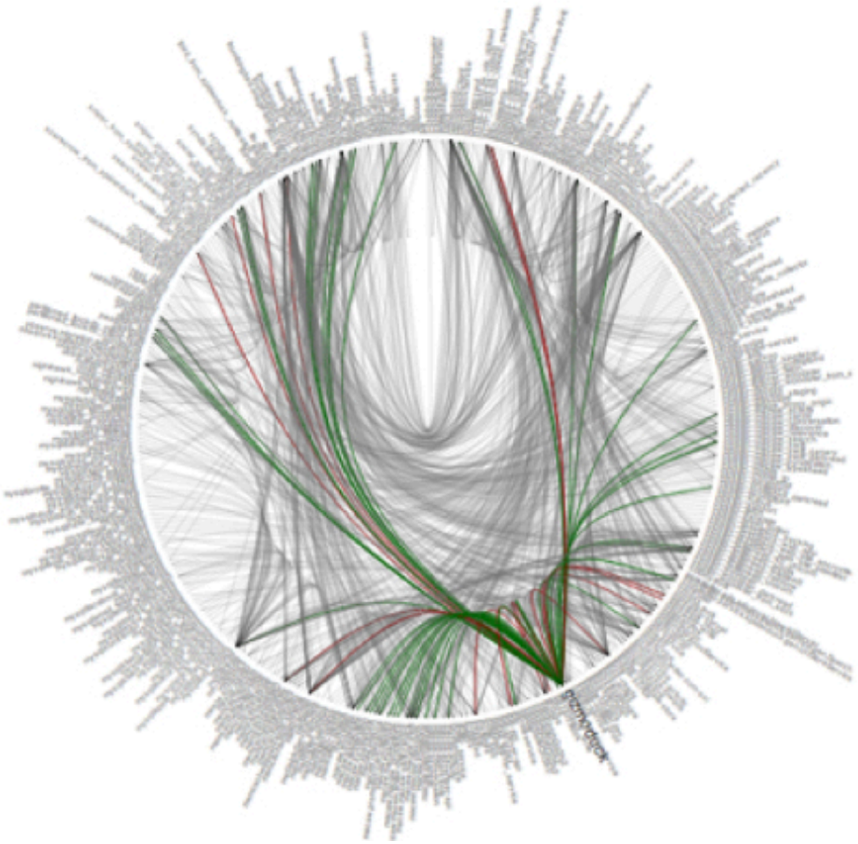
kubernetes

Microservice 2.0



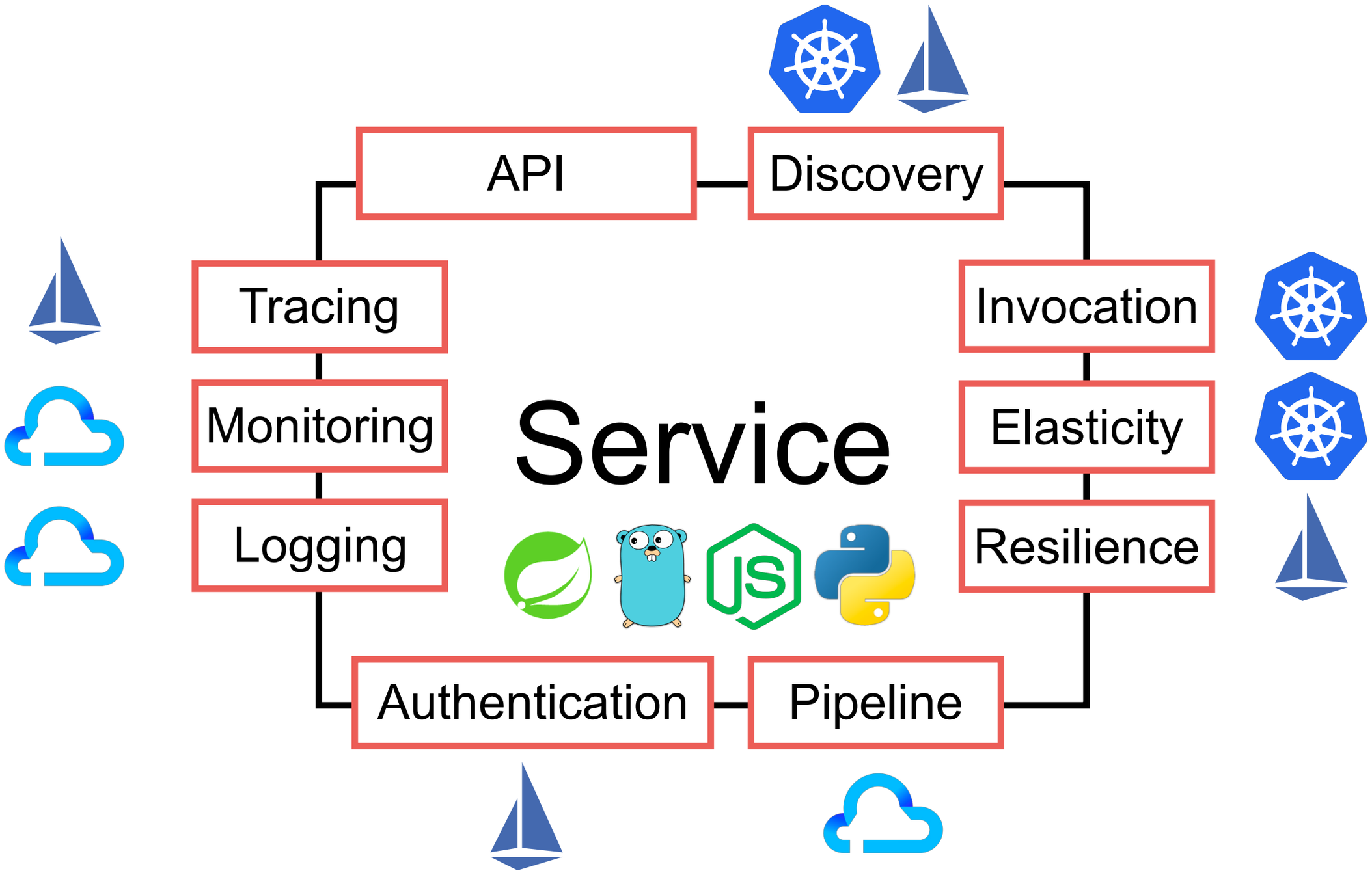


Service Mesh

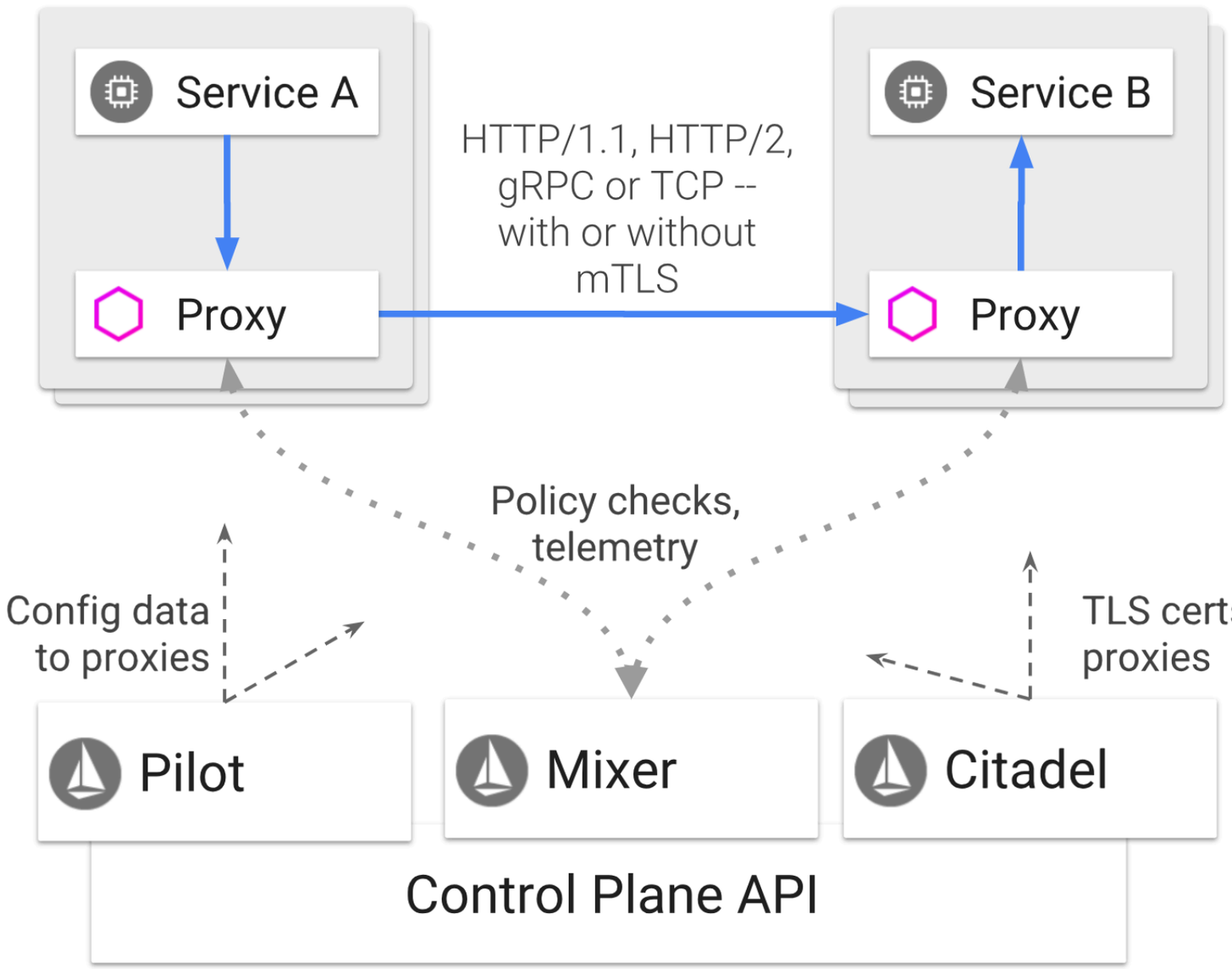


Service Mesh with Istio

Microservice 3.0

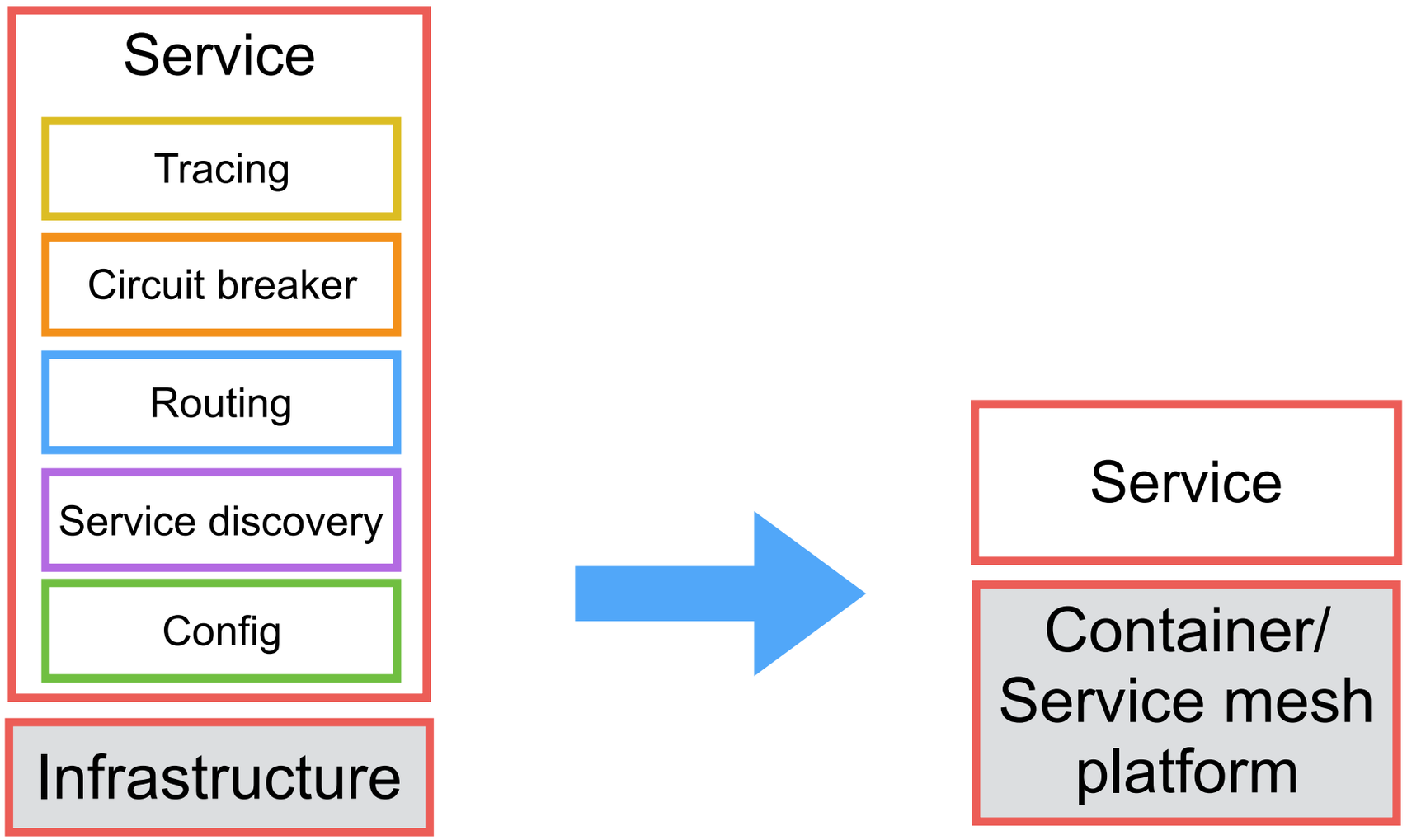


Istio



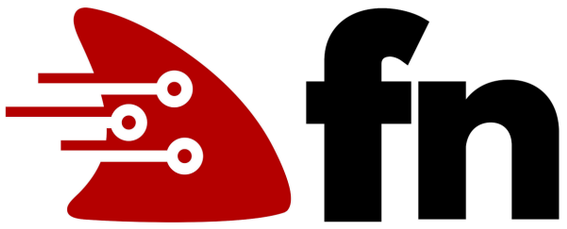
https://istio.io/docs/concepts/what-is-istio/

Microservice Evolution



Function-as-a-Service (FaaS)

Microservice 4.0 == FaaS



OPENFAAS

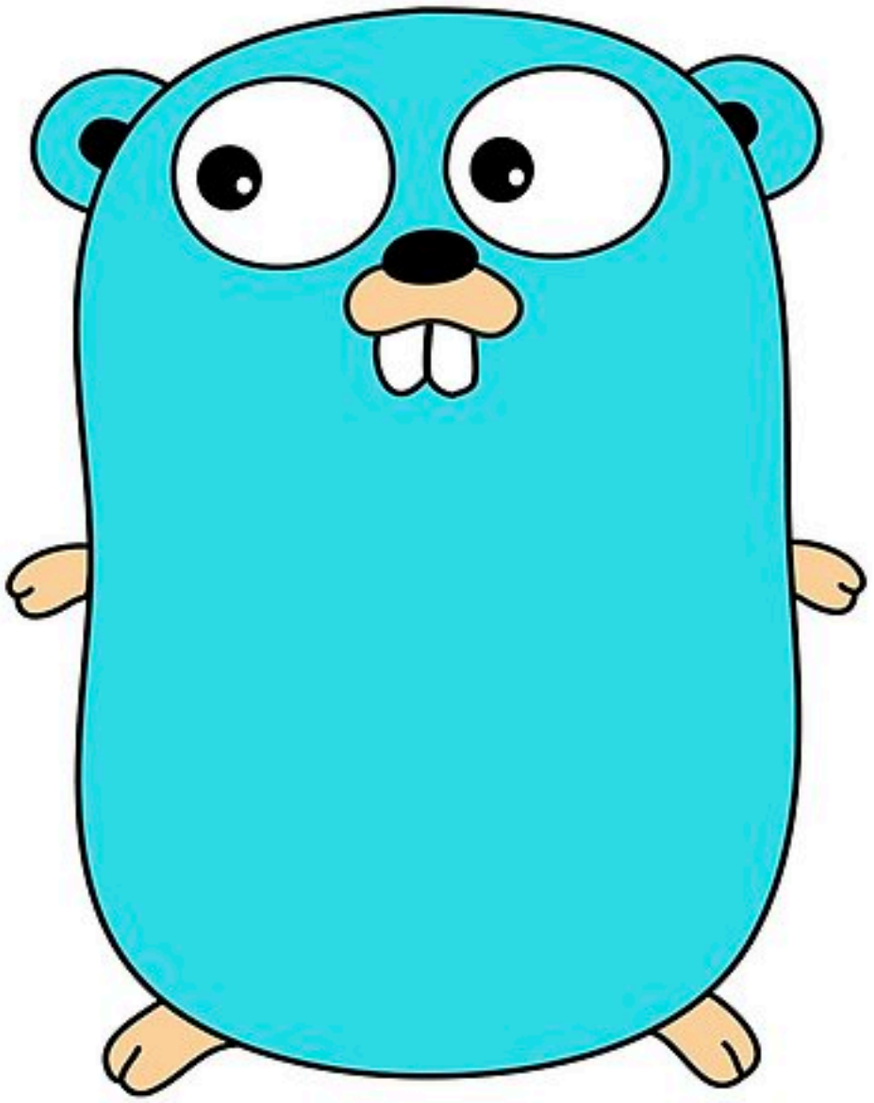
APACHE
OpenWhisk™

fission

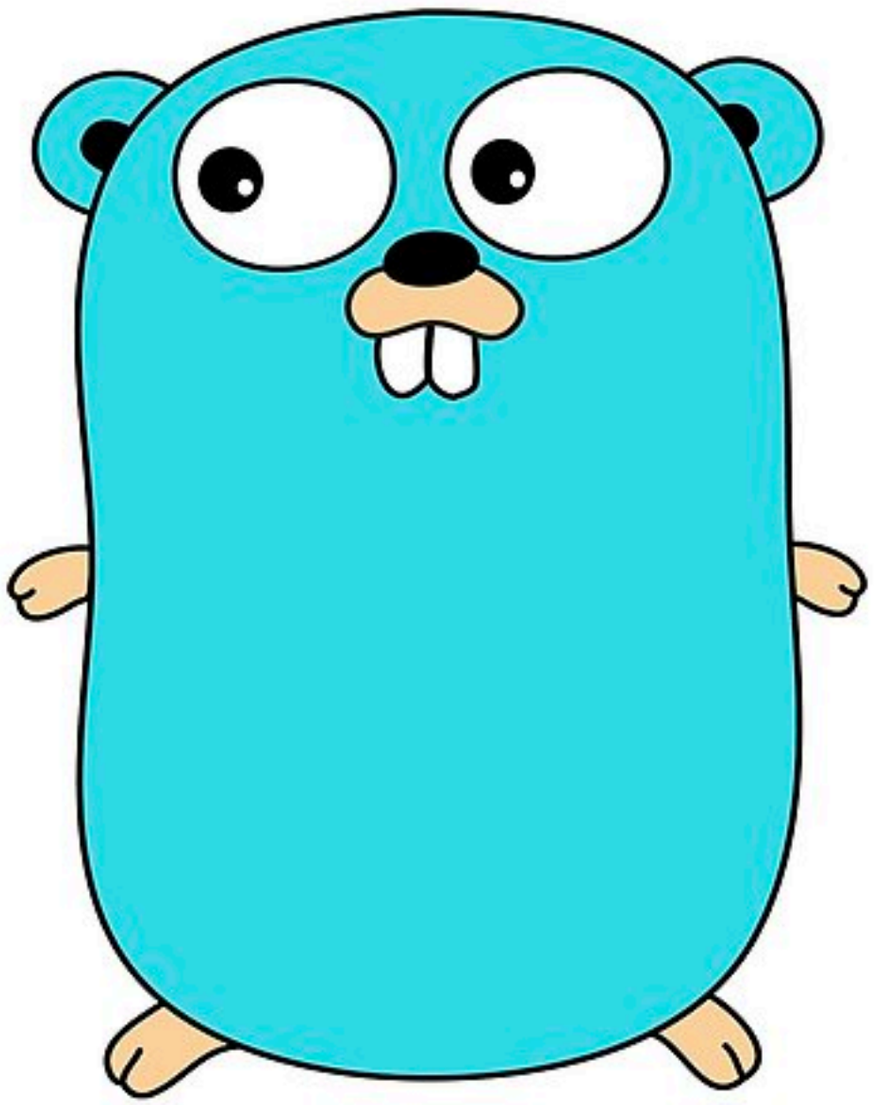
Kubeless

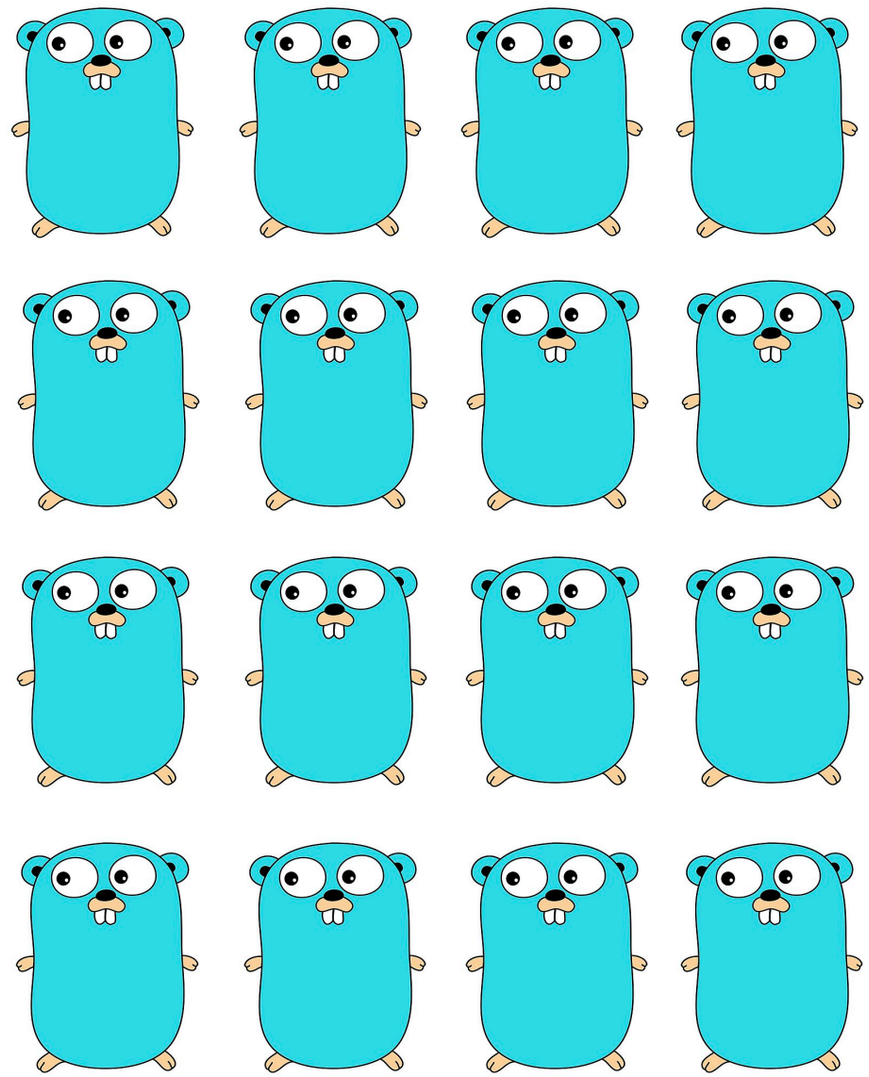
Microservice with Go

Microservice



Microservice





Go missing !!

Transport library to solve RPC problem

Standard logging

Generic interface for service discovery, event stream …

Hard to work with other team

Go-kit

Go-micro

Goa

Need

dgota

Twirp

Gizmo

https://awesome-go.com/

Focus on outcome

Not on tools/technologies

Composition over Inheritance

Library over Framework

Go-kit

Go kit

A toolkit for microservices

Home Examples FAQ Blog · **GitHub** GoDoc Slack Mailing list

🚀 Adopt Go in your organization.

Go is a lovely little language that's perfectly suited to writing microservices. Go kit fills in the gaps left by the otherwise excellent standard library, giving your team the **confidence** to adopt Go throughout your stack.

🔍 Focus on your business logic.

Adopting microservices means building a distributed system, and that comes with a lot of challenges. Go kit provides **guidance and solutions** for most of the common operational and infrastructural concerns. Allowing you to focus your mental energy on your business.

😌 Few opinions, lightly held.

You know your domain and context better than anyone. Go kit is lightly opinionated, and was designed for **interoperability** from day one. Use the databases, components, platform, and architecture that works best for you.

👥 Similar to...

Finagle • Dropwizard • Spring Boot • Netflix Eureka · Ribbon · Hystrix • Nameko

⚽ Plays nice with...

Kubernetes • Mesos • Marathon • DC/OS • Docker • Docker Swarm • Heroku

Why Go?

Go is designed from first principles to advance the practice of software engineering. It's easy to learn, easy to master, and — most importantly — easy to maintain, by large and dynamic teams of engineers. And with highly-efficient concurrency, an expansive standard library, and a steadily-improving runtime, it's practically the perfect language for writing microservices.

Why microservices?

Almost all of contemporary software engineering is focused on the singular goal of improving time-to-market. Microservices are an evolution of the service-oriented architecture pattern that elegantly eliminate organizational friction, giving your engineers and teams the autonomy they need to continuously ship, iterate, and improve.

Why Go kit?

Go is a great general-purpose language, but microservices require a certain amount of specialized support. RPC safety, system observability, infrastructure integration, even program design — Go kit fills in the gaps left by the standard library, and makes Go a first-class language for writing microservices in any organization.

Go-kit

Collections of Go packages

Robust

Reliable

Maintainable

Go-kit

Observability and resiliency pattern

Go-kit

Observability and resiliency pattern

Logging

Tracing

Metric

Circuit-breaker

Rate-limit

Goals of Go-kit

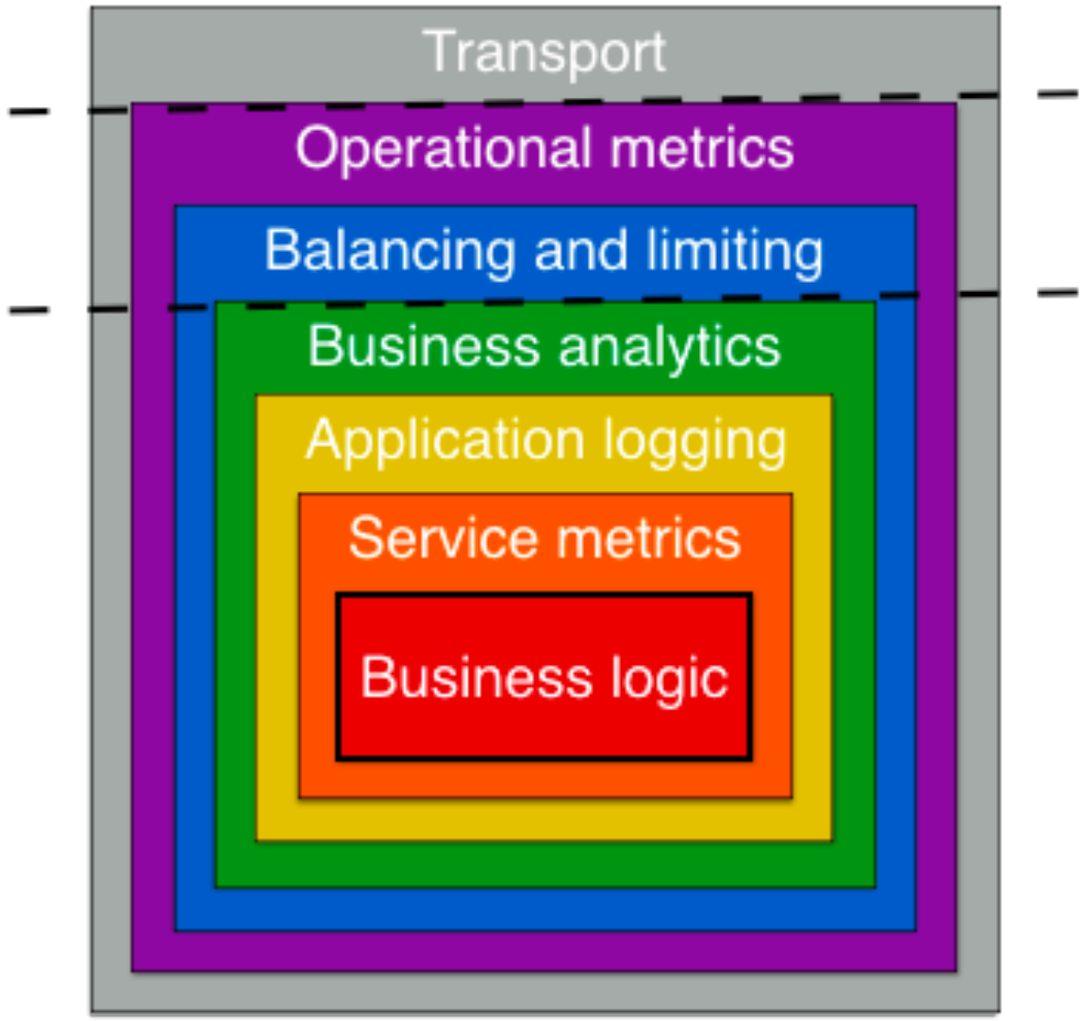
No global state

Composition

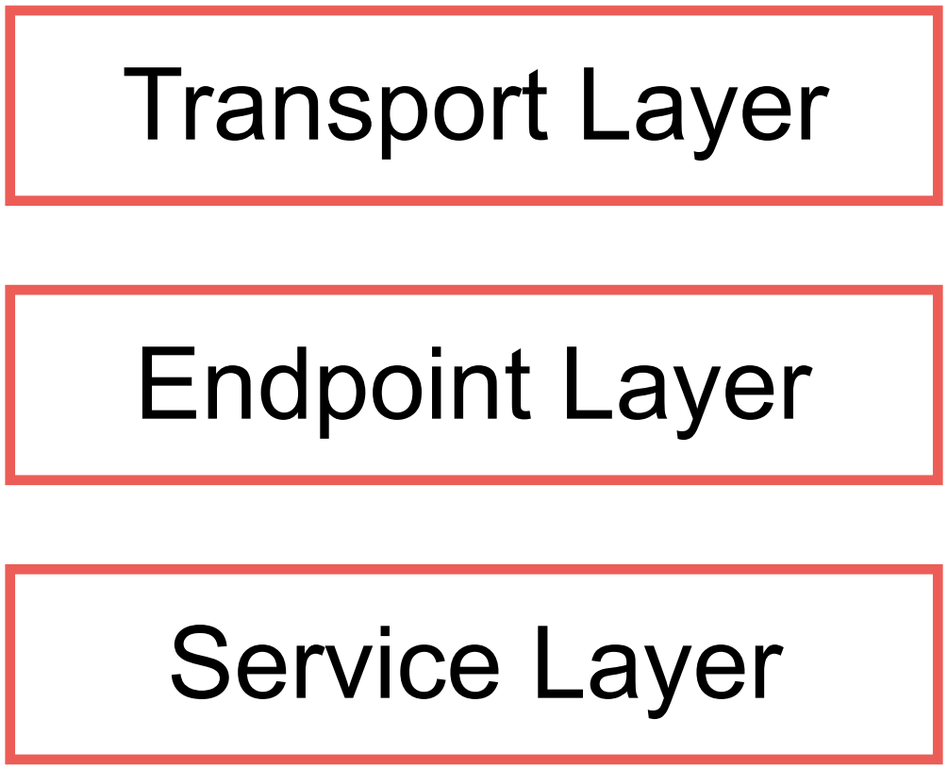
Explicit dependencies

Interface as contract

Go-kit :: Library



Go-kit :: Library



Transport Layer

Interaction between services

Synchronous

Asynchronous

Transport Layer

HTTP/REST, gRPC
Pub/Sub, AMQP, NATS, Thrift

Endpoint Layer

Expose service via endpoint

Single endpoint can be using multiple transports

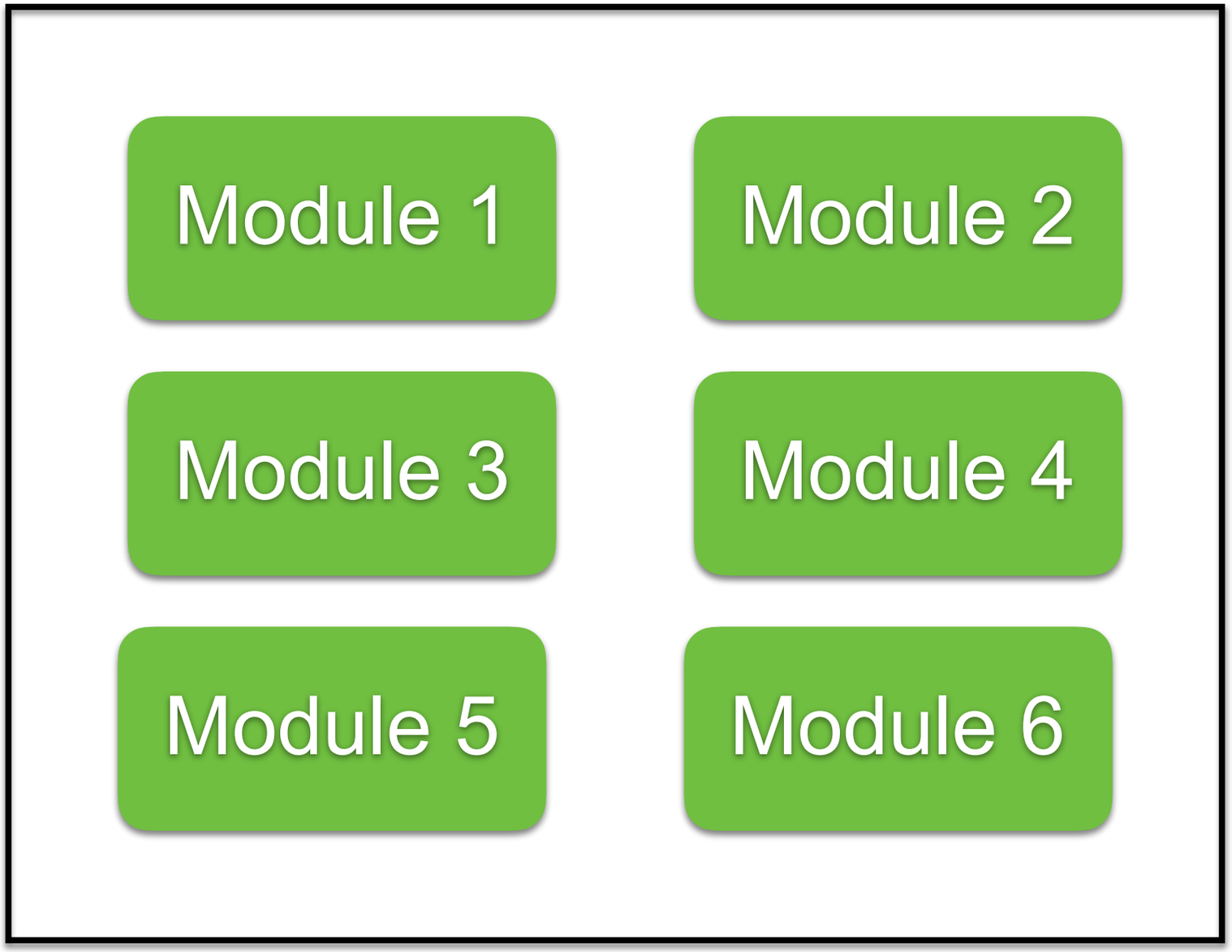
Service Layer

Business logic

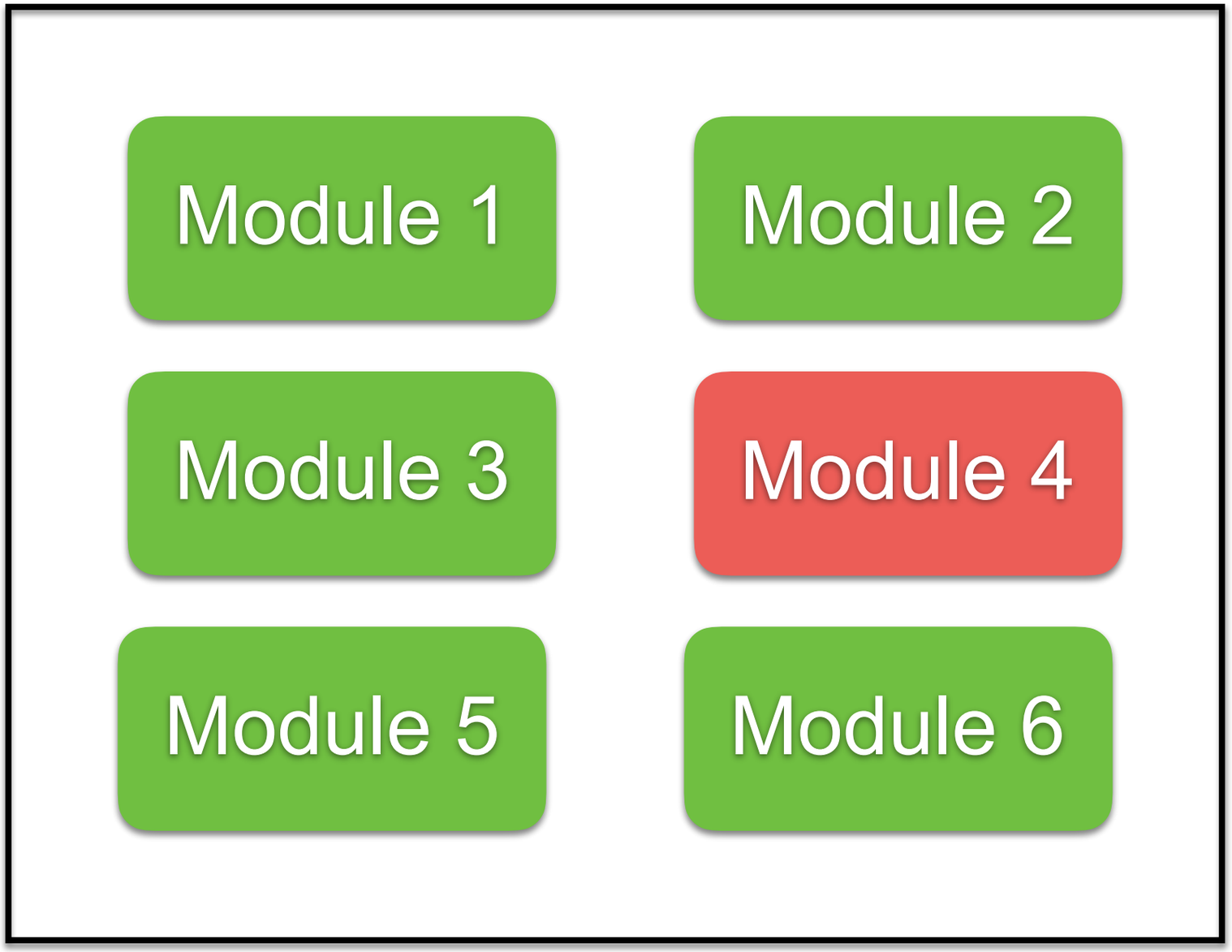
Demo with Go-kit

Summary

Let’s start with good monolith



Find your problem



Module 1

Module 2

Module 3

Module 5

Module 6

Module 4