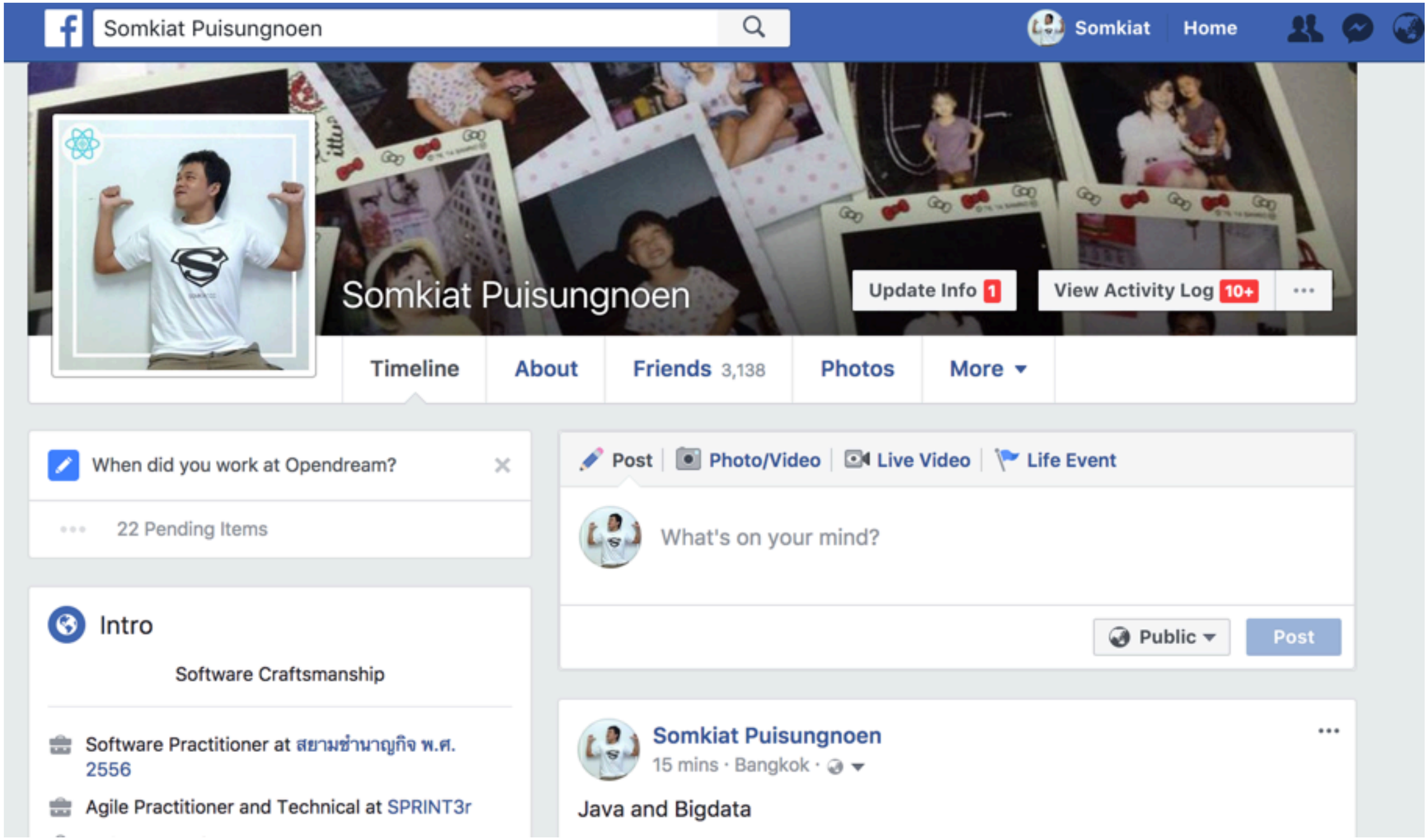
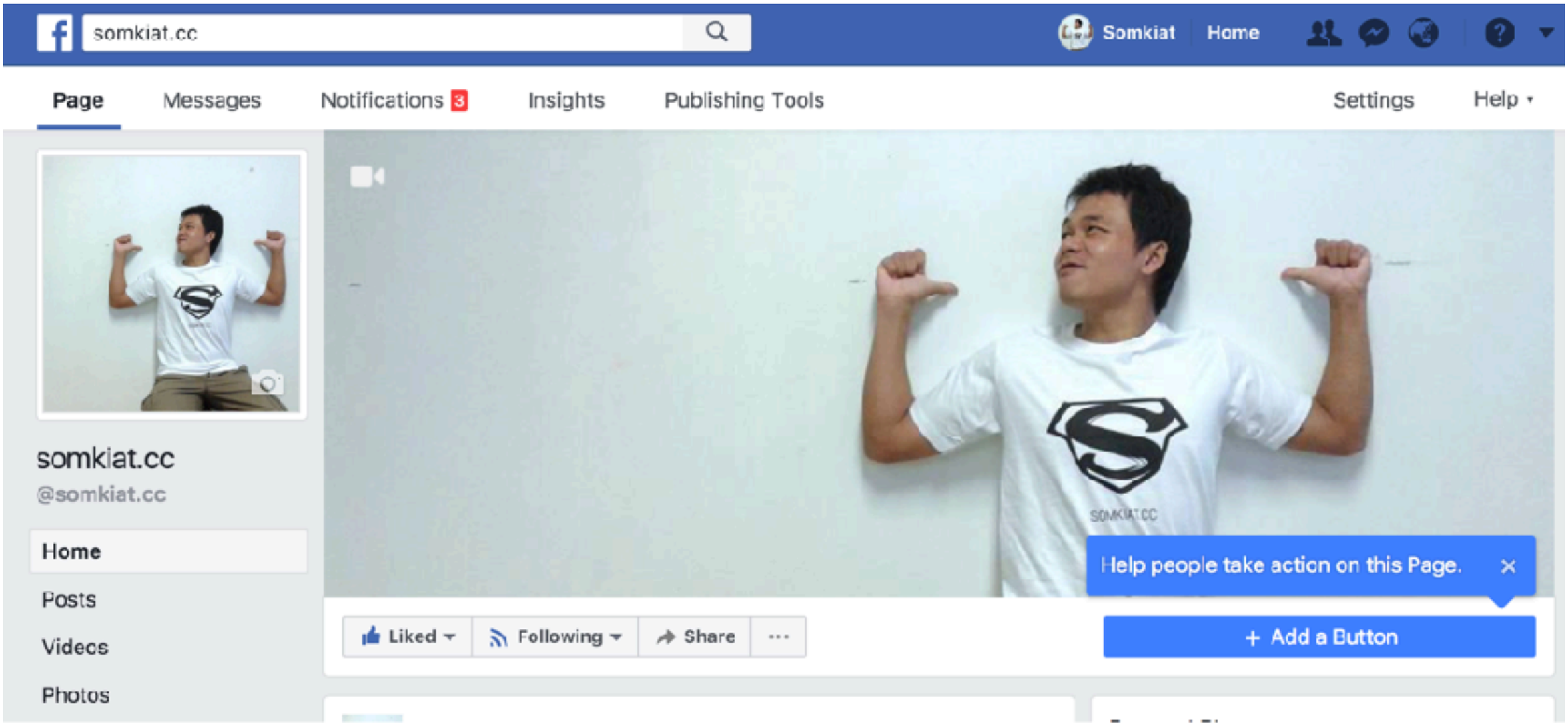
สยามชำนาญกิจ
SIAM CHAMNAN KIT

# Microservices

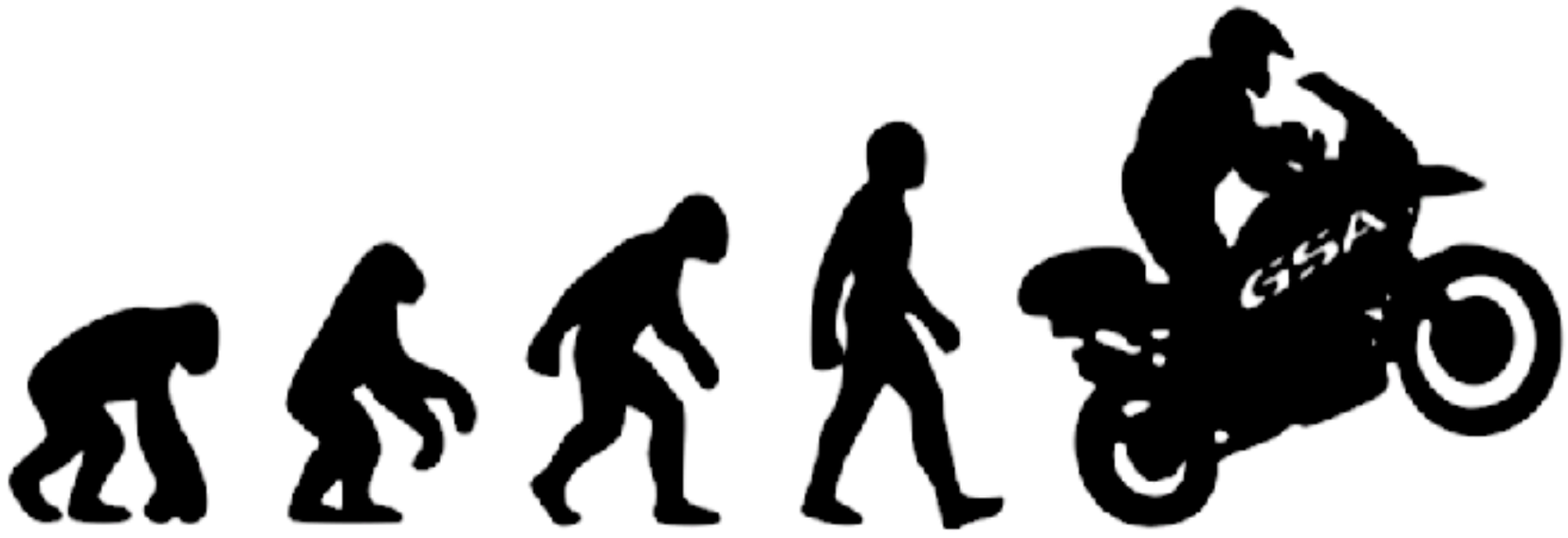


# Agenda

- Cloud Native Application
- Microservices and DevOps
- The architecture of Microservices
- How to model Microservices
- Integrating multiple Microservices

# Agenda

- Testing and Developing Microservices
- Monitoring Microservices
- Deploy and Scale Microservices




monolith n-tier service oriented microservices serverless

# Evolution of Architecture

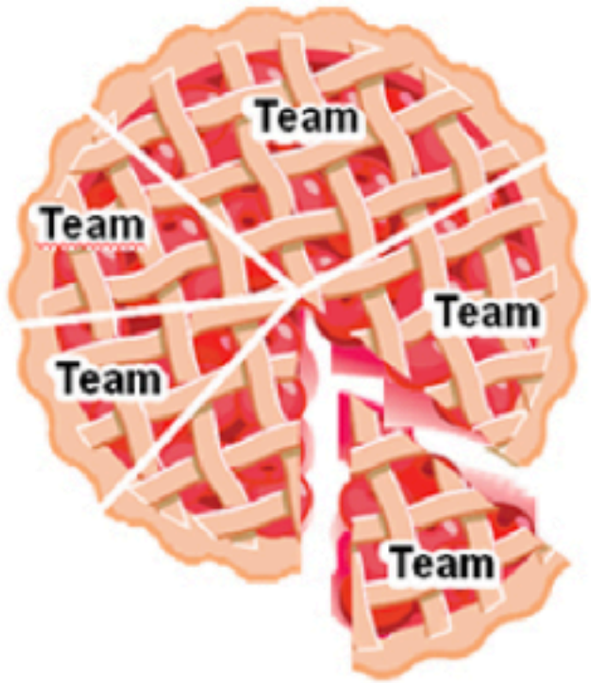
**1990s and earlier**

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.

## ABSTRACTION IN THE ENTERPRISE


### Era of Mainframe

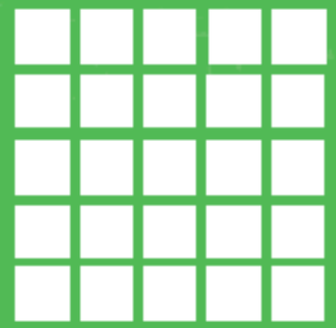
### Era of Client Server

### Era of the Web


### Era of the Cloud

# Monolith

ABSTRACTION IN THE ENTERPRISE



1960 1970 1980 1990 2000 010 2020

Era of Mainframe

Era of Client Server

Era of the Web


Era of the Cloud

# Monolith issues

## Grow Fat

Code base grows. All the things slow down.

## Age

Your code base will become a jurassic park introducing new tech becomes hard

## Ownership

Who is responsible for which part and more important: who has the pager

## Economies of Scale

The bigger the team the more they interrupt each other

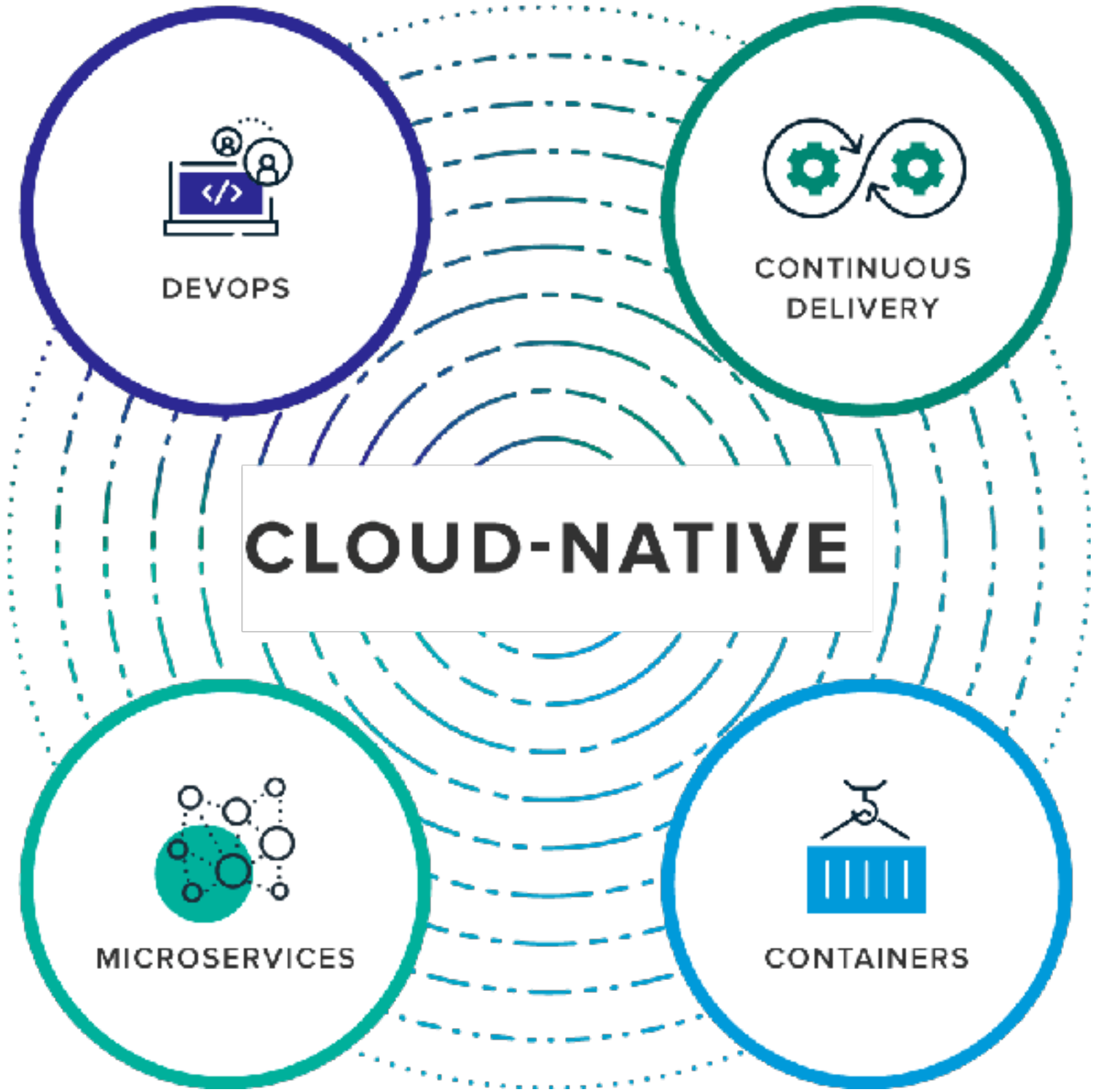
https://trello.com/

# We need Change

# “Remember, we are not all webscale”

# “Optimize for rapid and sustainable flow of value”

*Dan Nord*




https://pivotal.io/cloud-native

# Microservices

# Microservices

## Small

The size will be reasonable and manageable

## Independent lifecycle

Nothing will hold the team back. Go as fast as you can

## Optimise for the problem

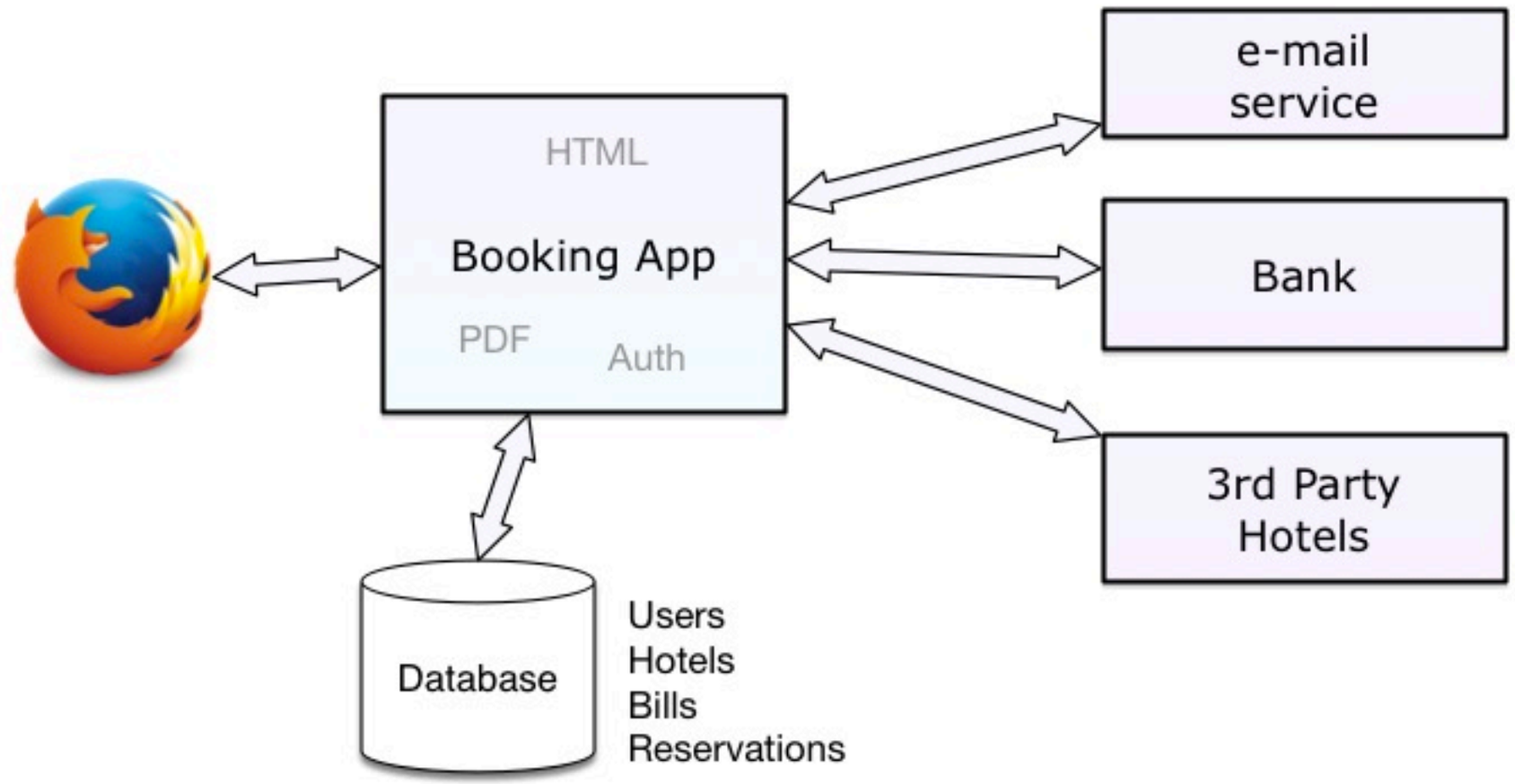
Pick solution and tech based on the problem at hand

## Replaceable

It is easier to replace if there is a need for it

https://trello.com/

# Monolithic

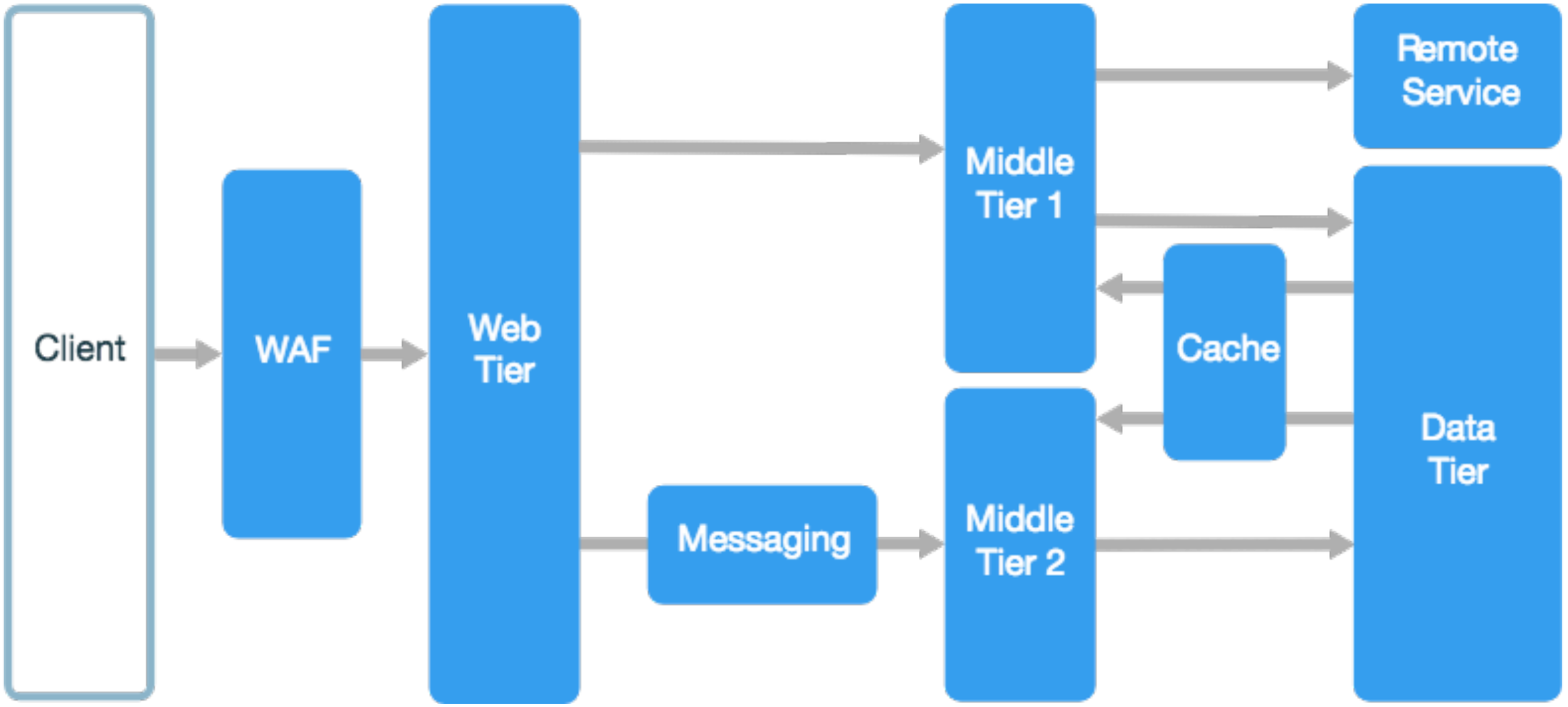


# Monolithic

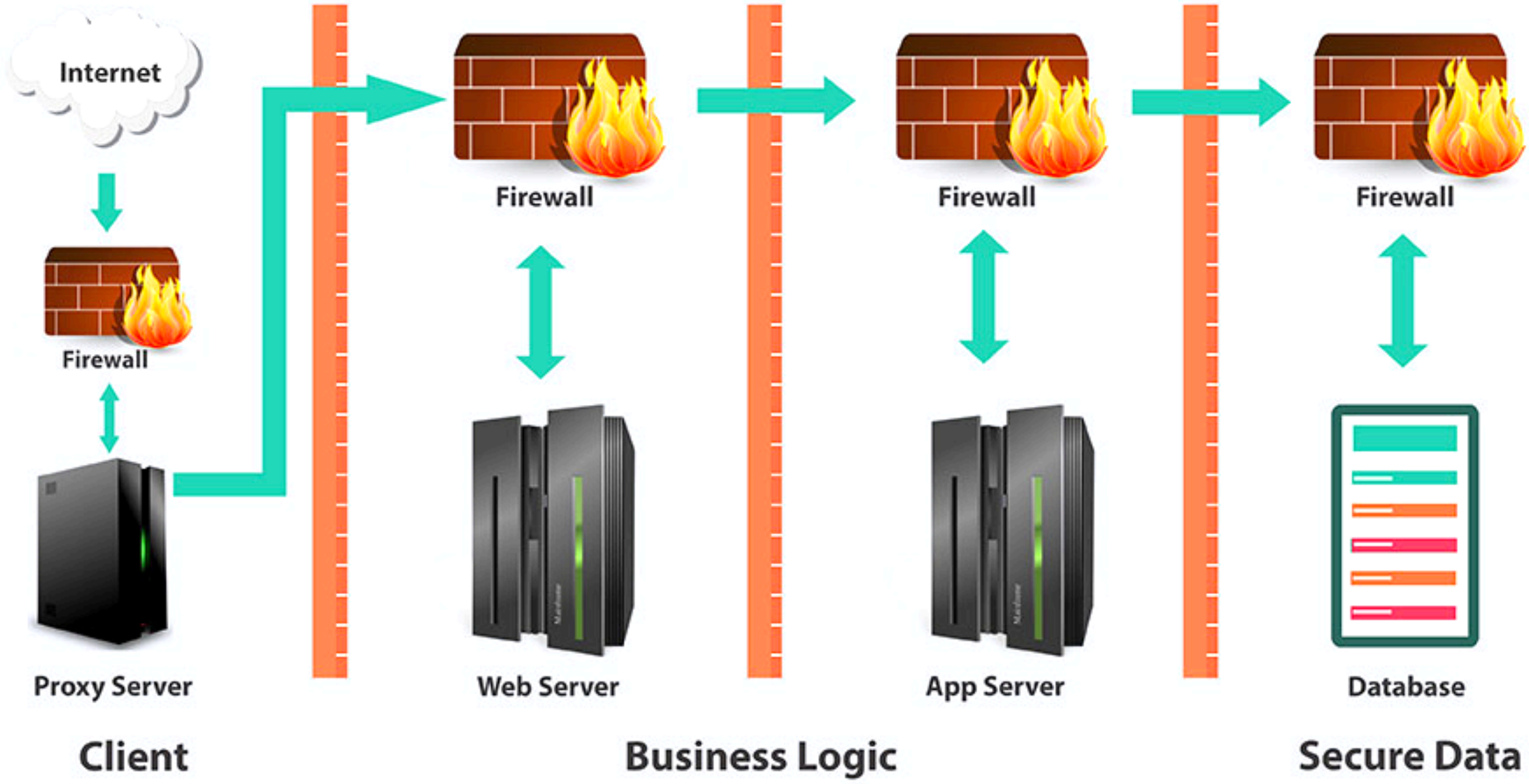
# Monolithic

# Monolithic

# N-Tier



# N-Tier



# Problem ?

# SOA

# SOA

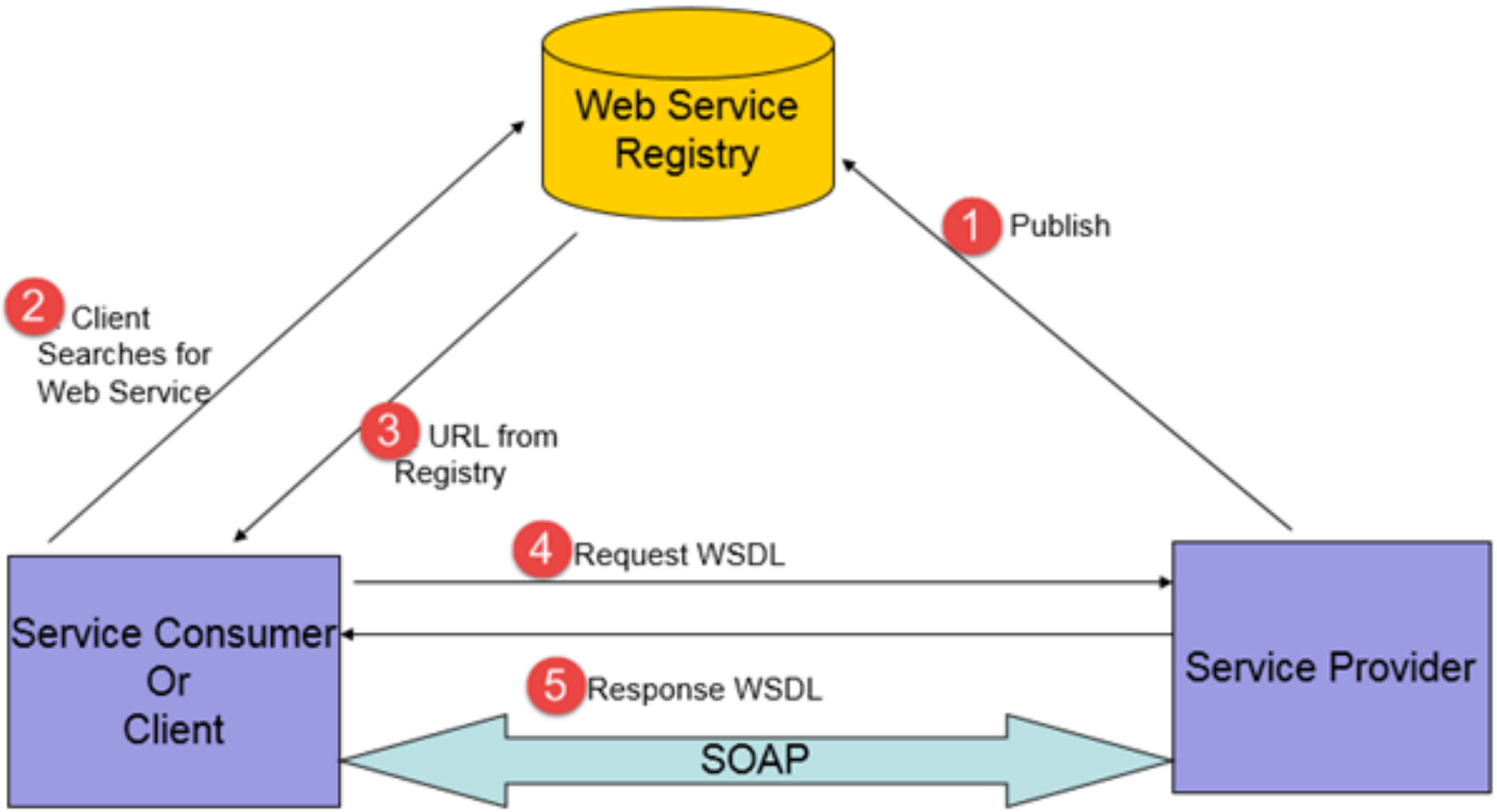
Focus on business/service reusability

# SOA

For communication it use ESB

# WebService

WebService is a implementation of SOA



# SOA

CIO FROM IDG

Home >

OPINION

## Top 10 Reasons Why People are Making SOA Fail

SOA is not something you buy, it is something you do. Research shows us that very few companies are
well. But the reasons for so many failures are usually people issues, not technology issues.

By Mike Kavis

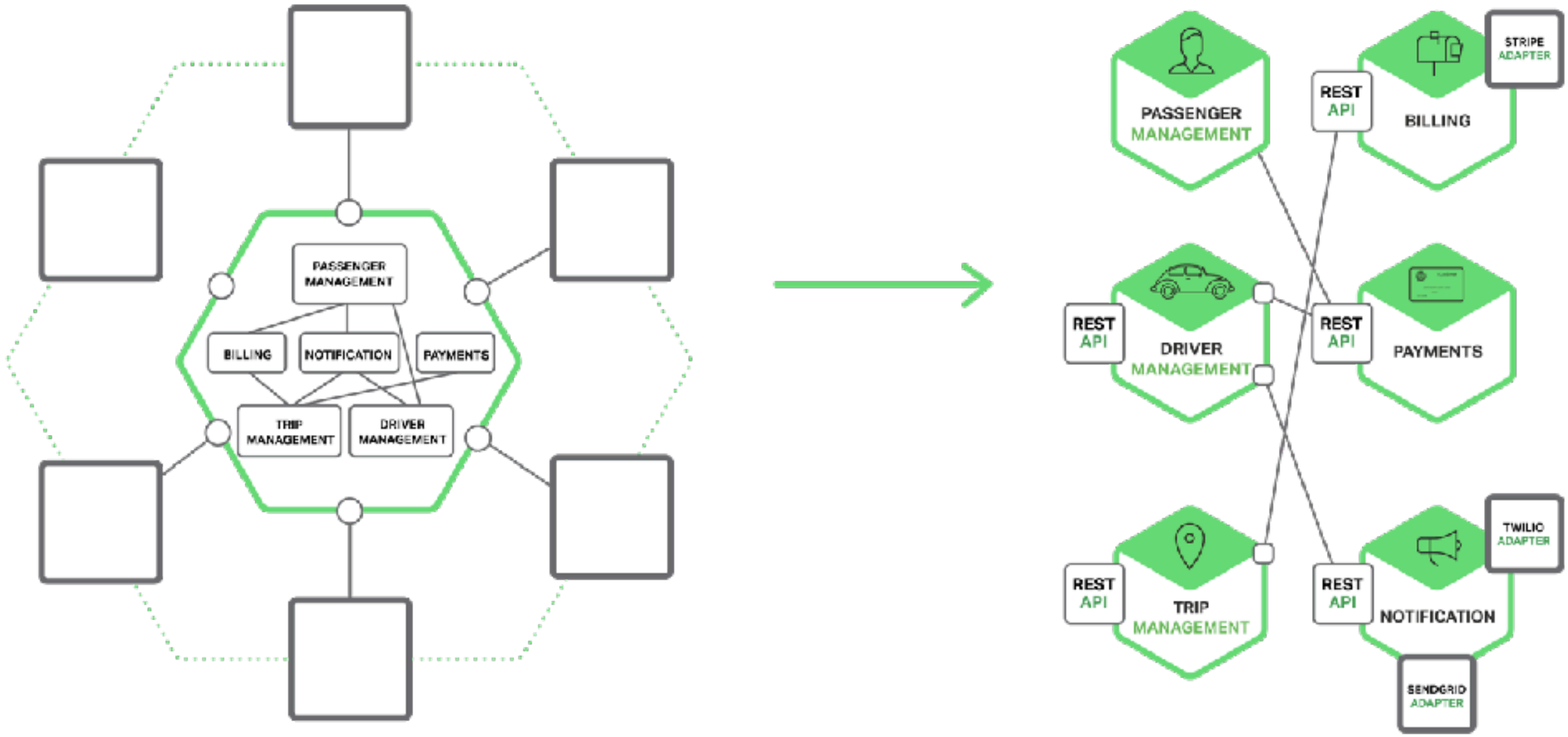
CIO | JUL 18, 2008 8:00 AM PT

**2008**

https://www.cio.com/article/2434865/service-oriented-architecture/top-10-reasons-why-people-are-making-soa-fail.html

# Microservices

# Microservices




https://www.nginx.com/

# What are Microservices ?

# “Microservices is about people”

# Microservices

## Engineering approach

# Microservices

Engineering approach

**Decompose application into single-function modules**

# Microservices

Engineering approach

Decompose application into single-function modules

**Well-defined interfaces**

# Microservices

Engineering approach

Decompose application into single-function modules

Well-defined interfaces

**Independently deploy**

# Microservices

Engineering approach

Decompose application into single-function modules

Well-defined interfaces

Independently deploy

**Operate by small team who own the lifecycle of service**

# Microservices

Accelerate delivery by minimizing communication and coordination between people

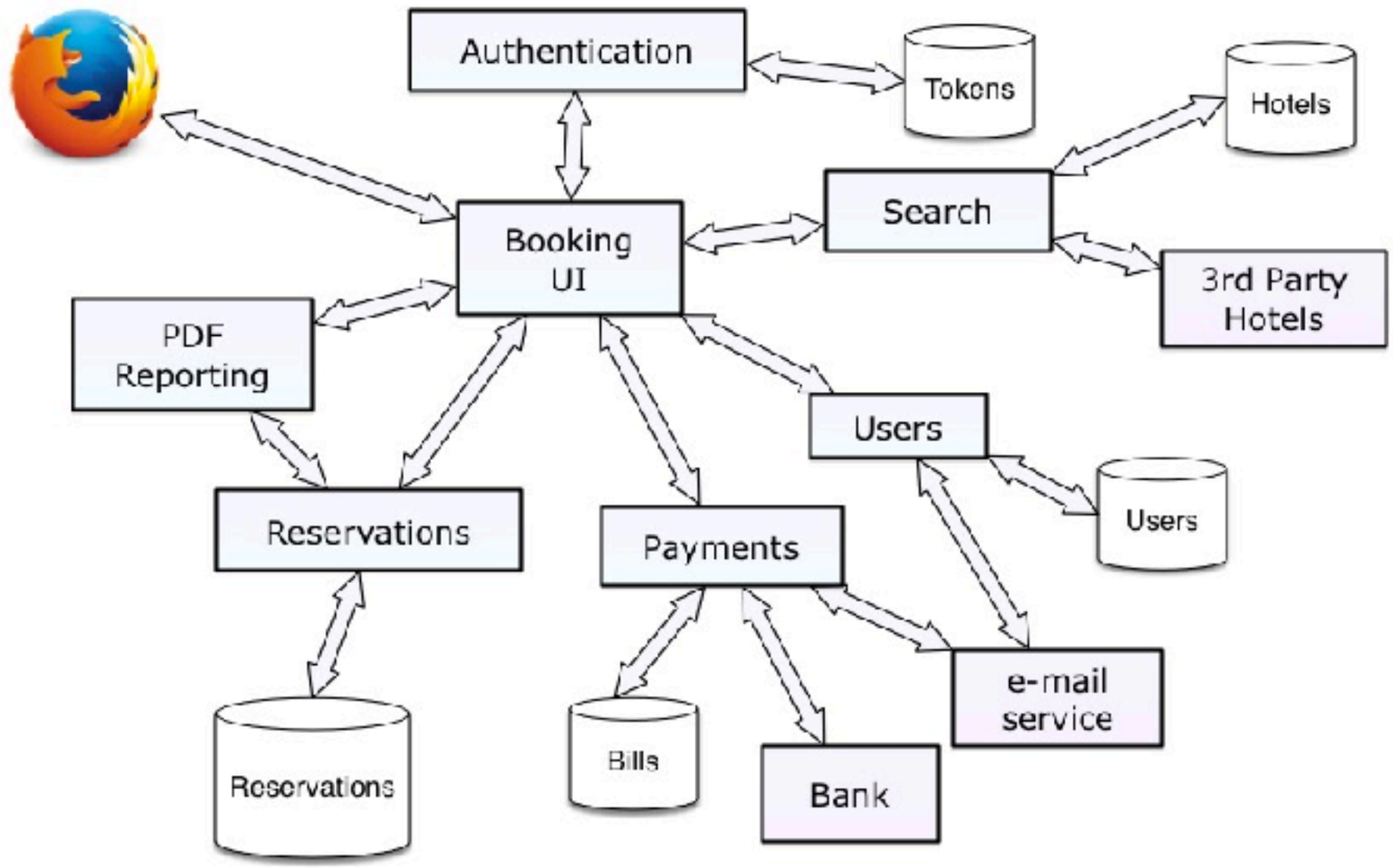
Reduce scope and risk of change

# Microservices

Decompose
Single-function
Well-defined interfaces
Independent
Small team
Lifecycle
Minimize communication
Reduce the scope of change

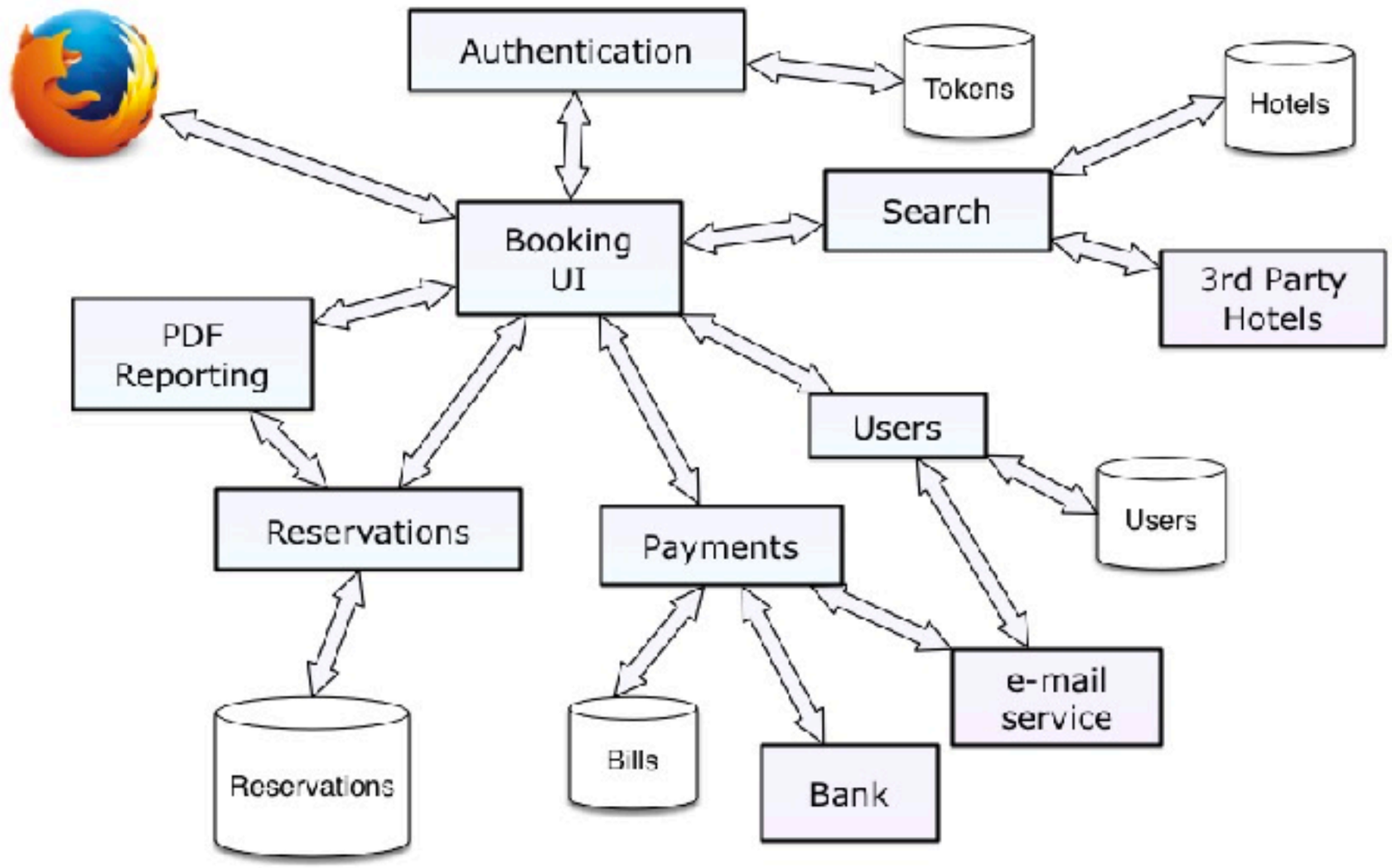
# Microservices

## Focus on decoupling of services



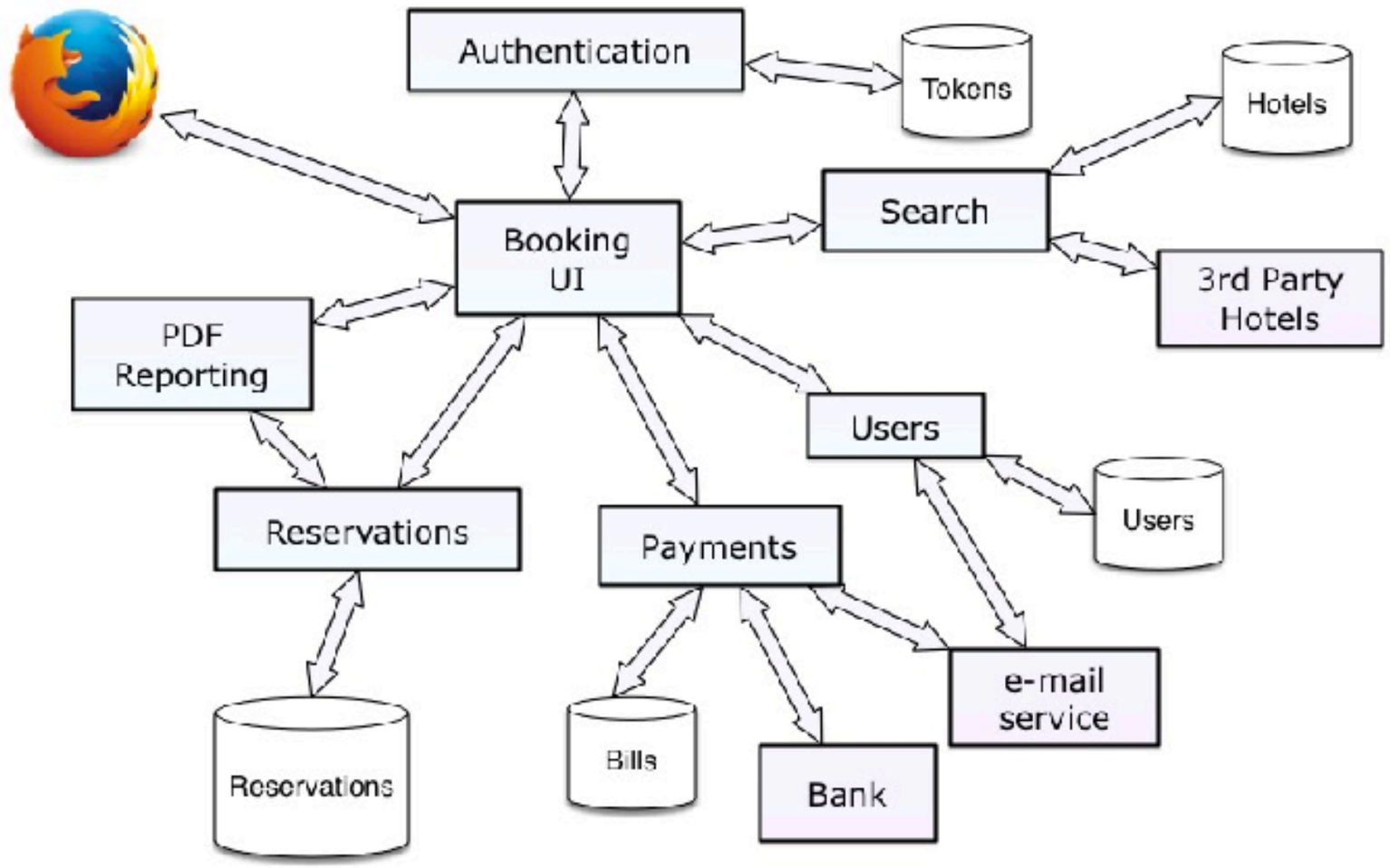
# Microservices

## Application/Web server is not required



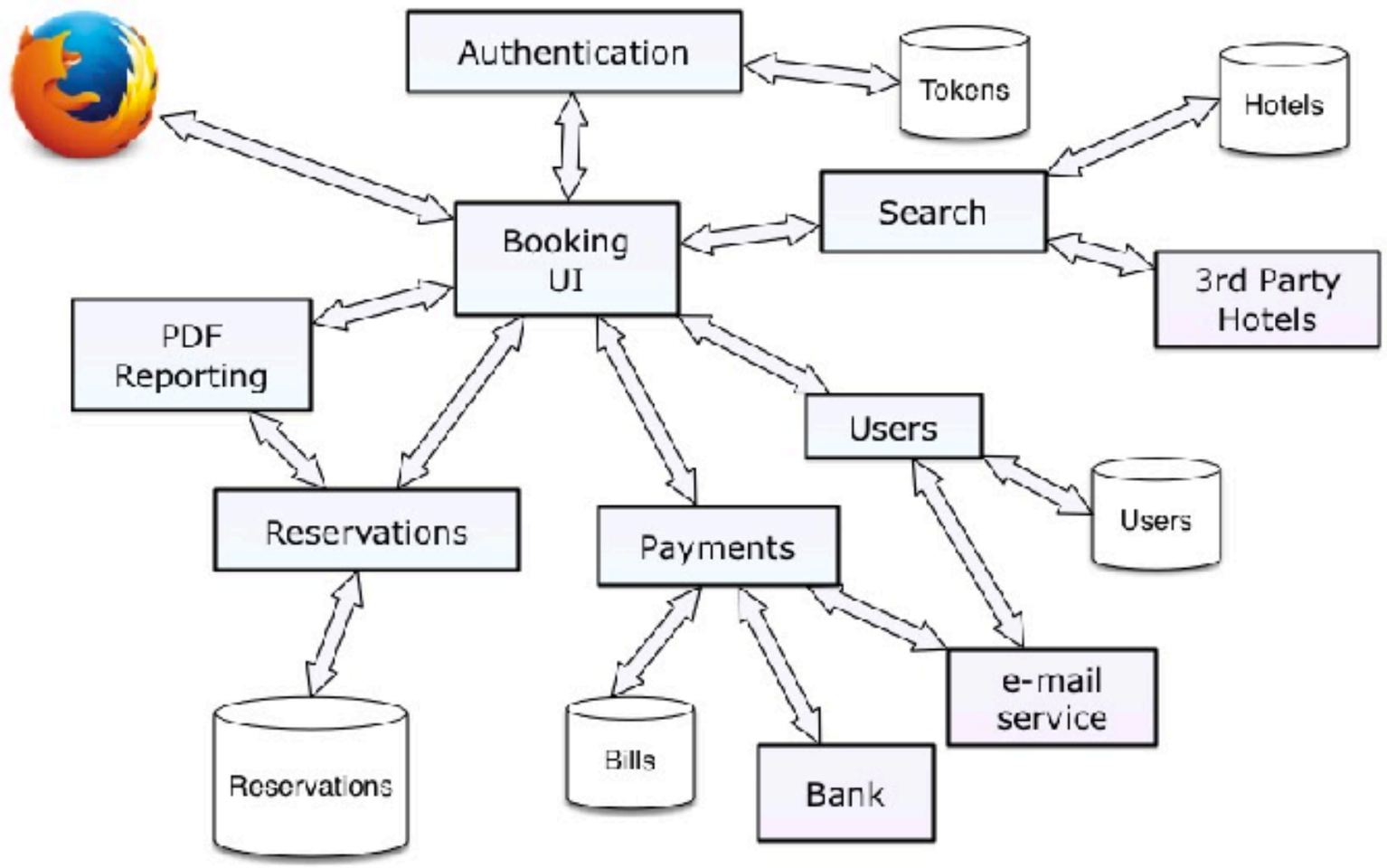
# Microservices

## Communication use a simple messaging system



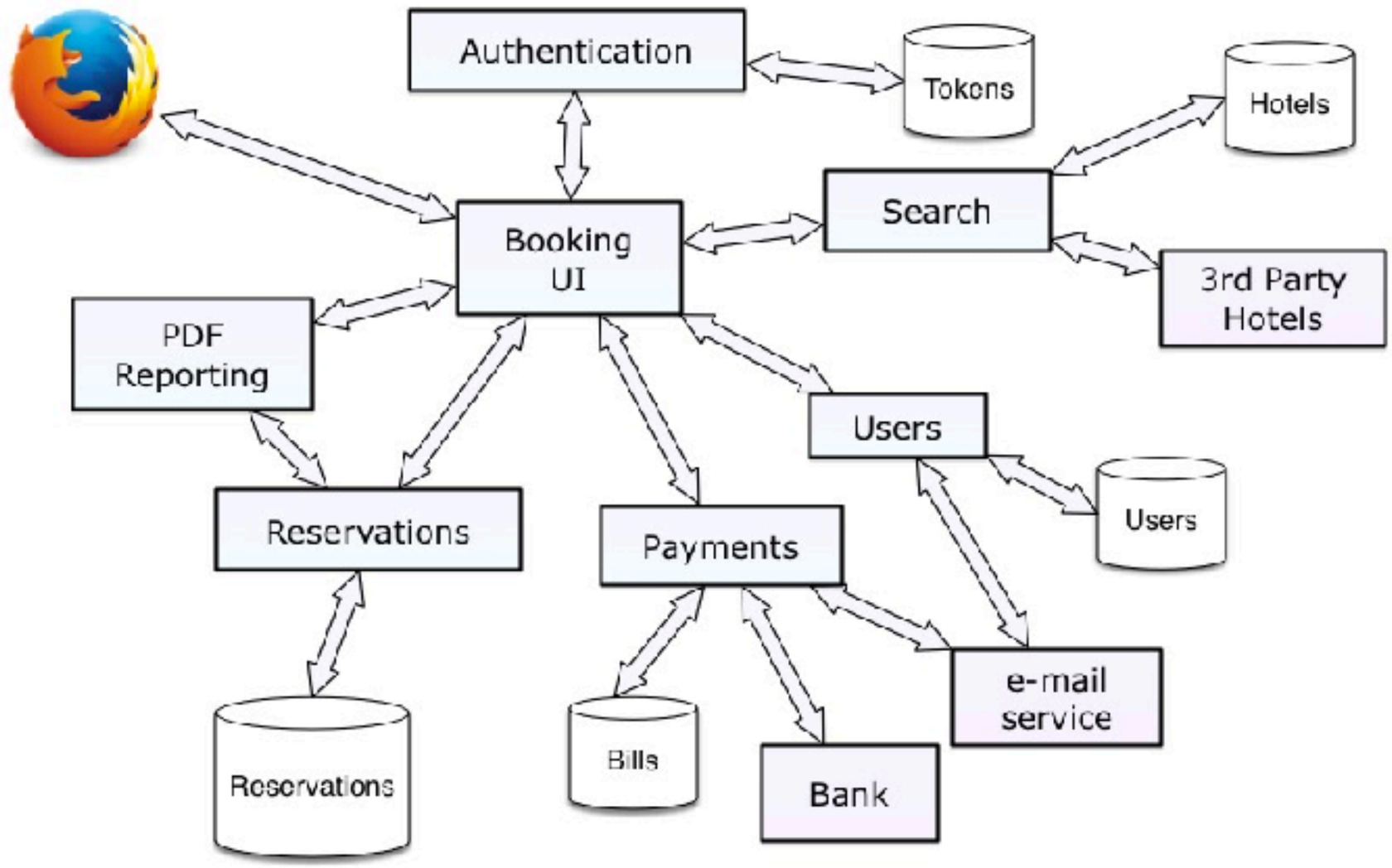
# Microservices

Focus on team collaboration and freedom of choice



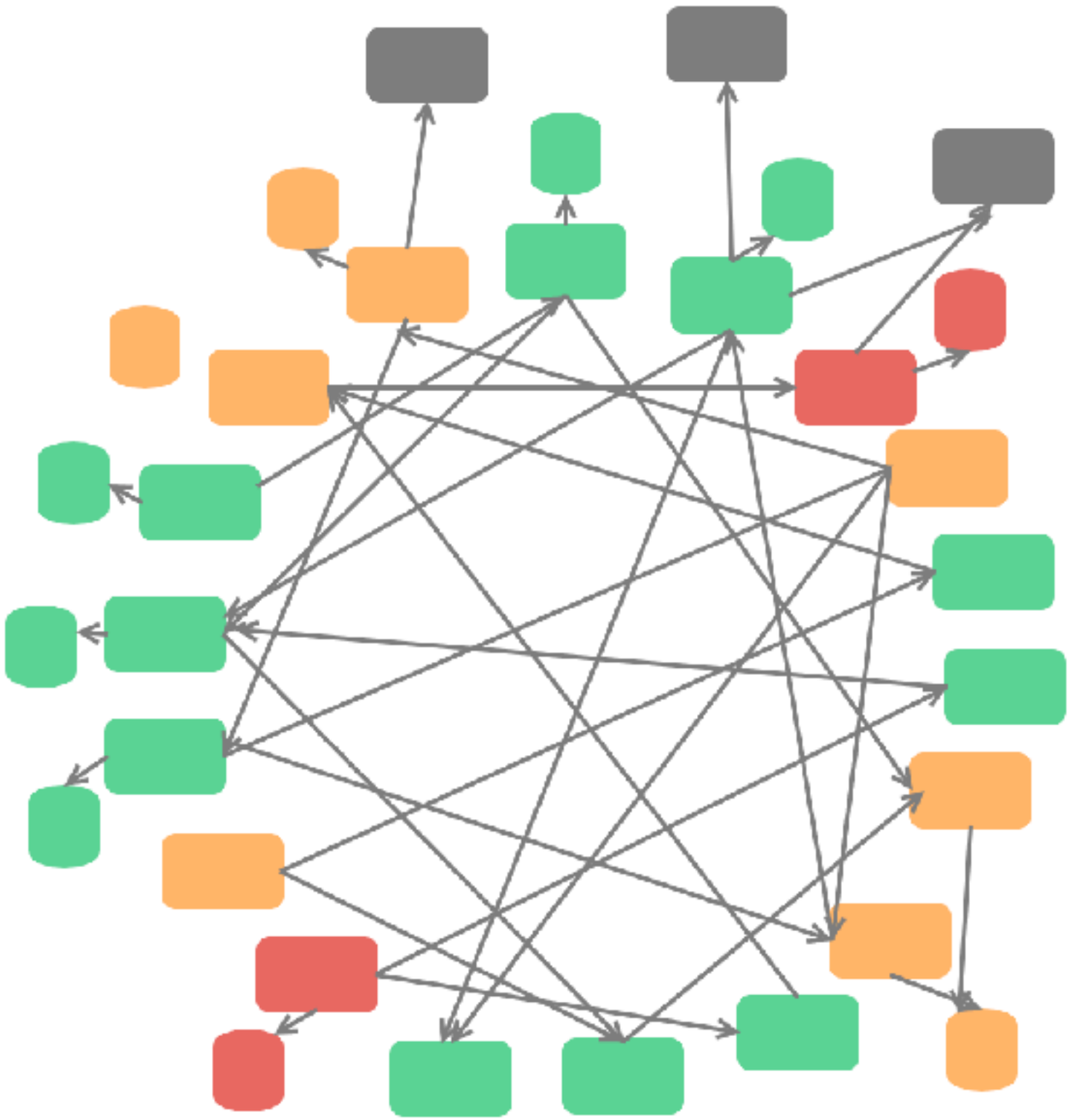
# Microservices

## Minimize on sharing

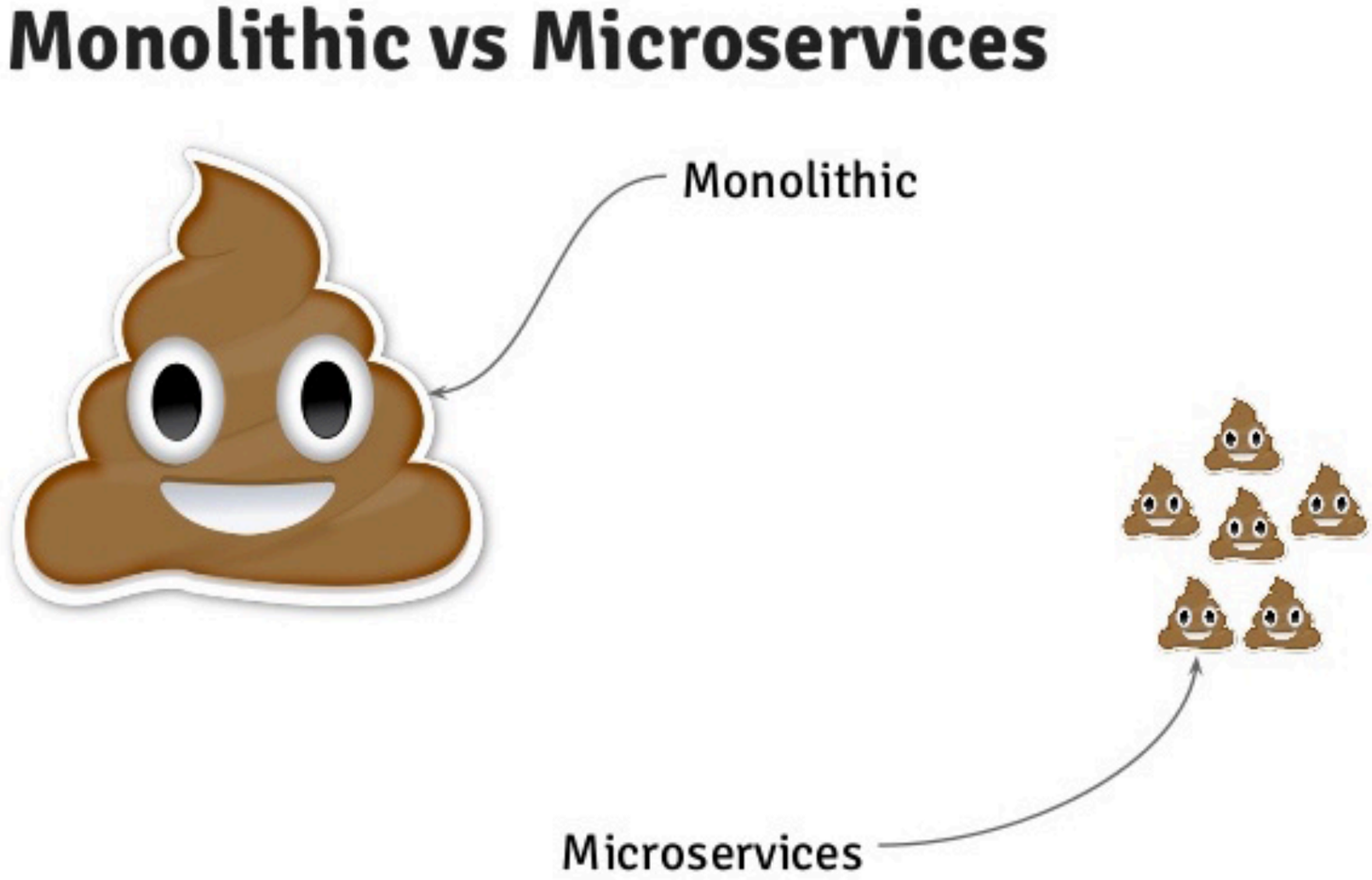


# Microservices spaghetti

# Microservices spaghetti



# Microservices spaghetti



# SOA vs Microservices

# SOA vs Microservices

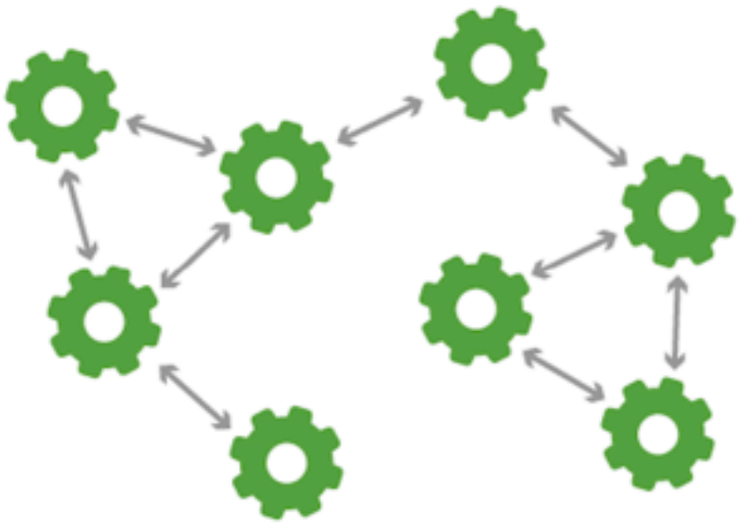
2000's

## SERVICE ORIENTED ARCHITECTURE


**SOA** based applications are compromised of more loosely coupled components that use an Enterprise Services Bus messaging protocol to communicate between themselves.

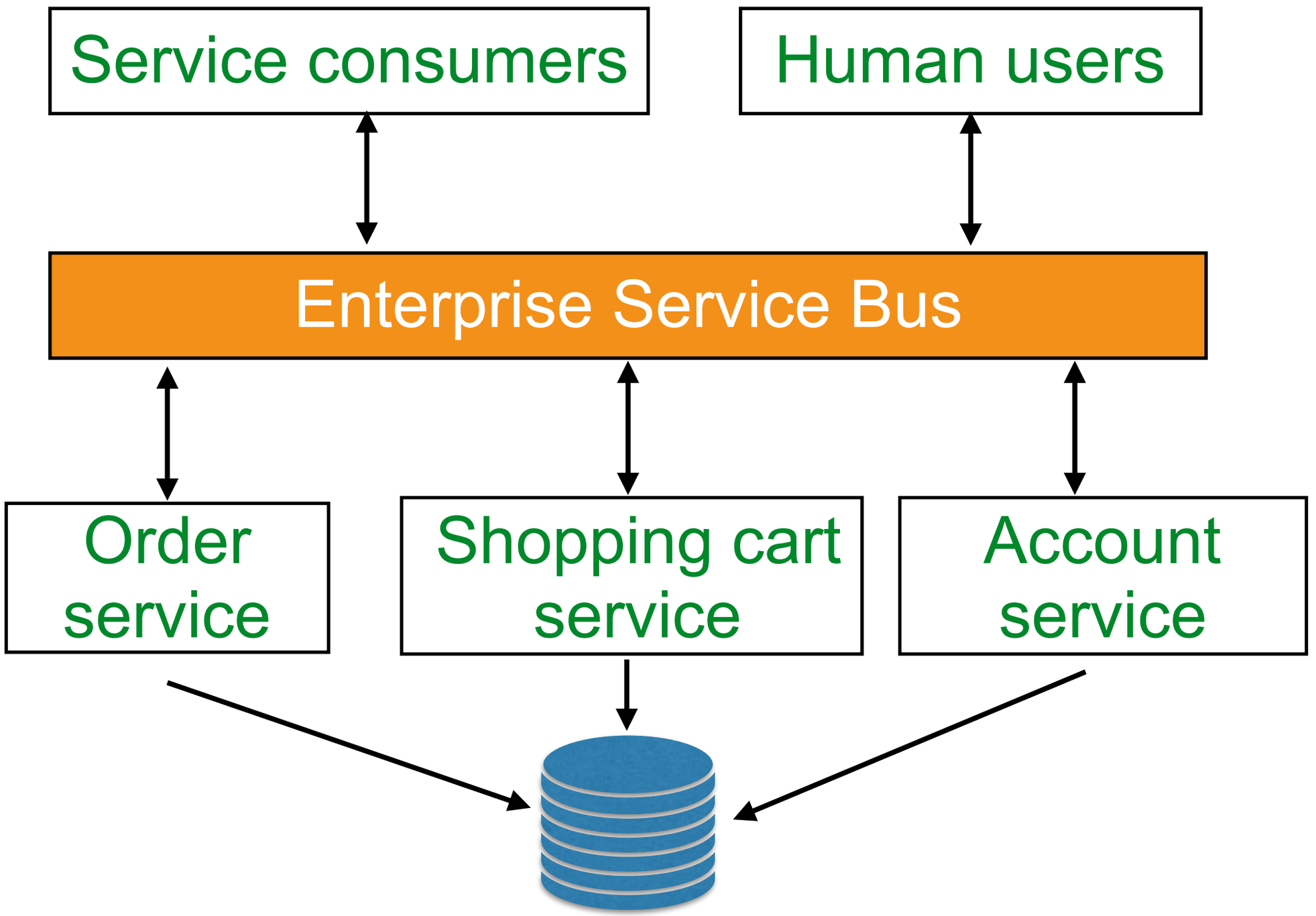
2010's

## MICROSERVICES ARCHITECTURE

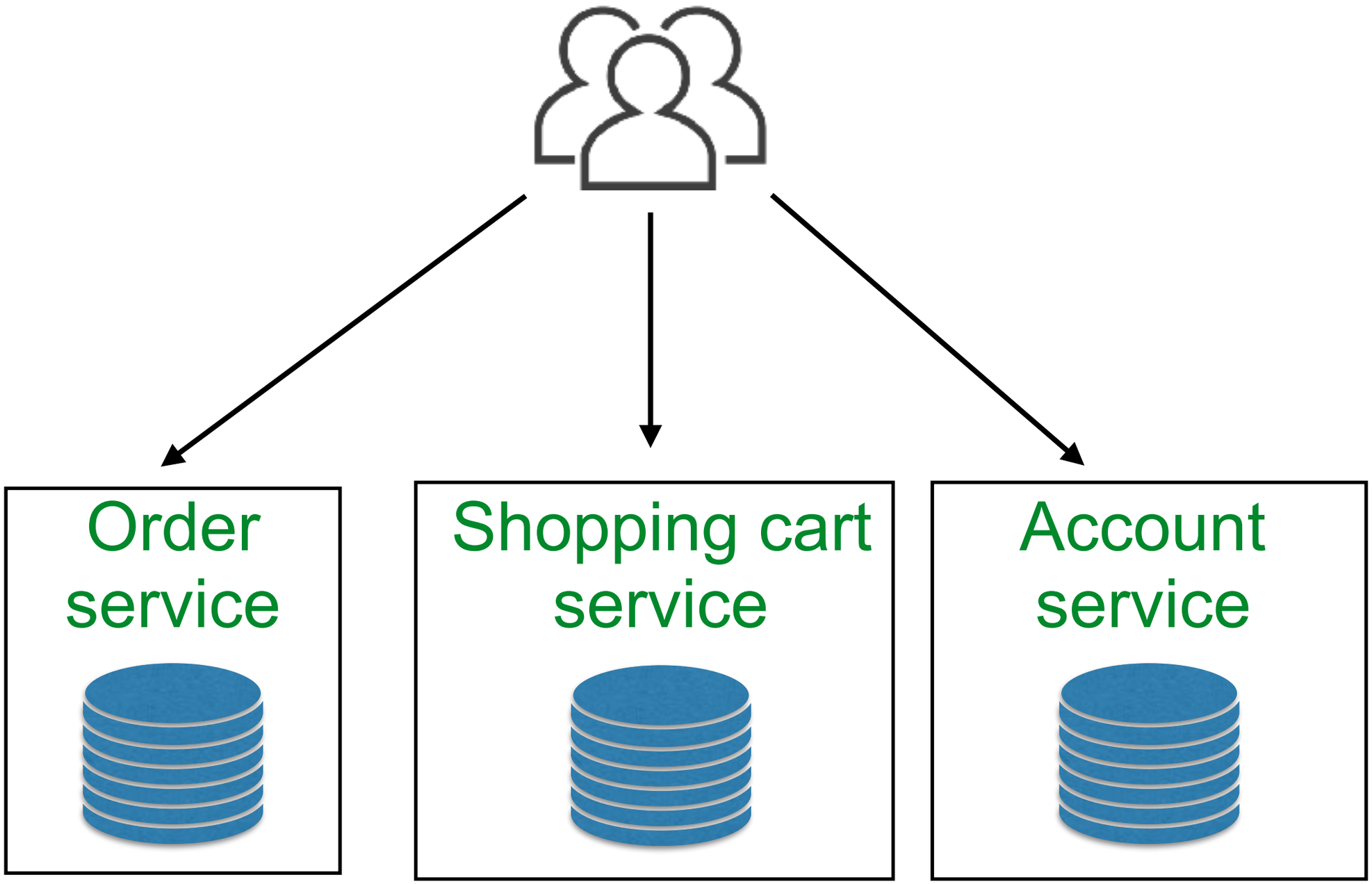


**Microservices** are a number of independent application services delivering one single functionality in a loosely connected and self-contained fashion, communicating through light-weight messaging protocols such as HTTP, REST or Thrift API.

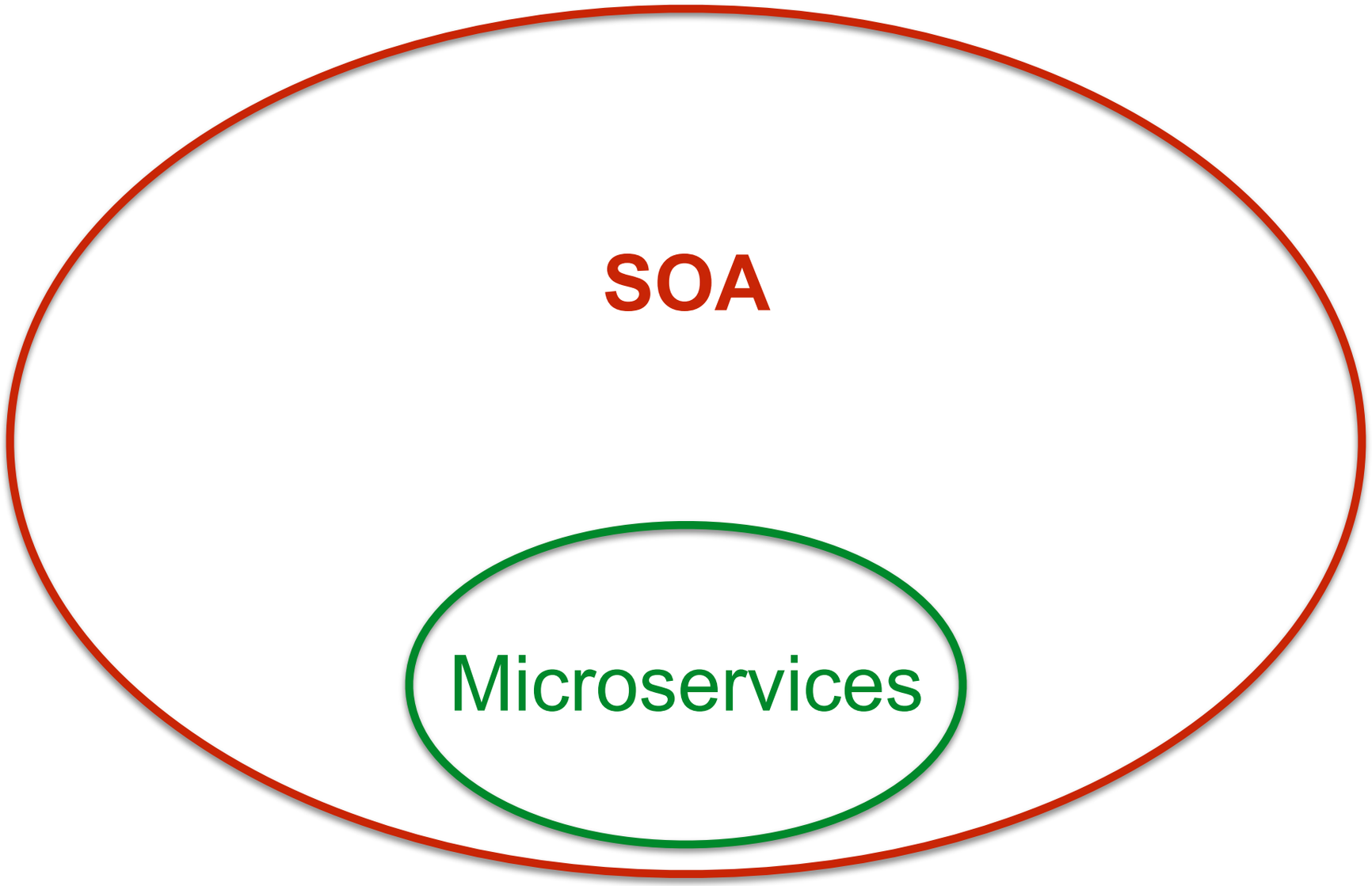
# SOA



# Microservices



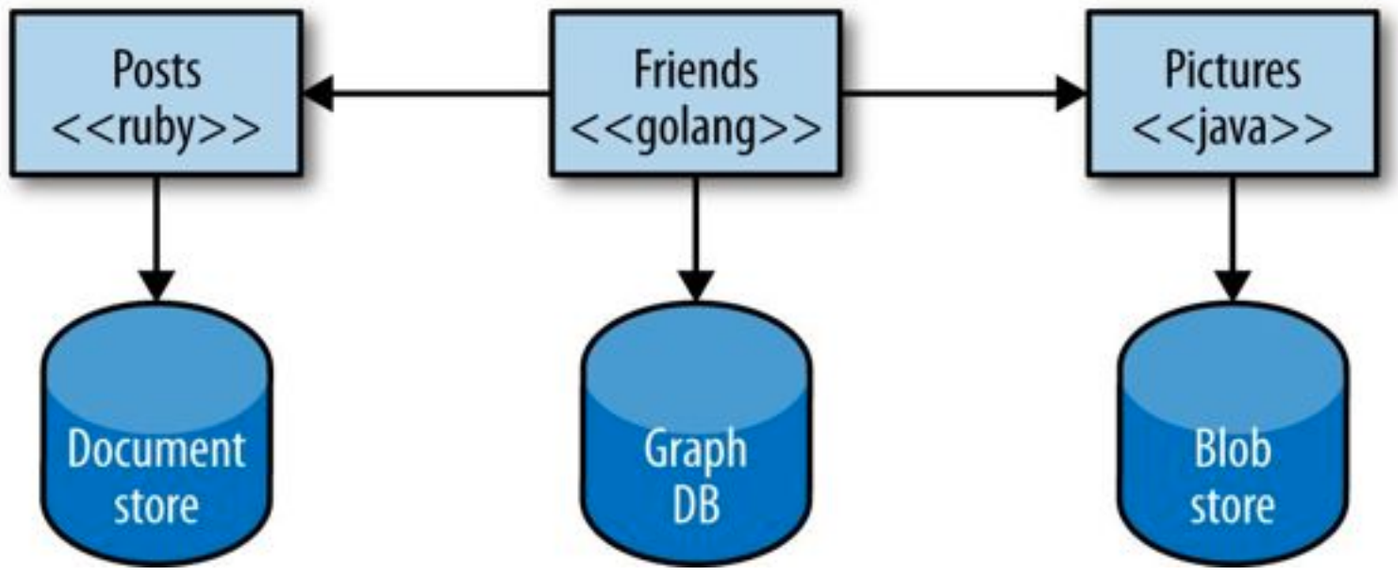
# SOA vs Microservices



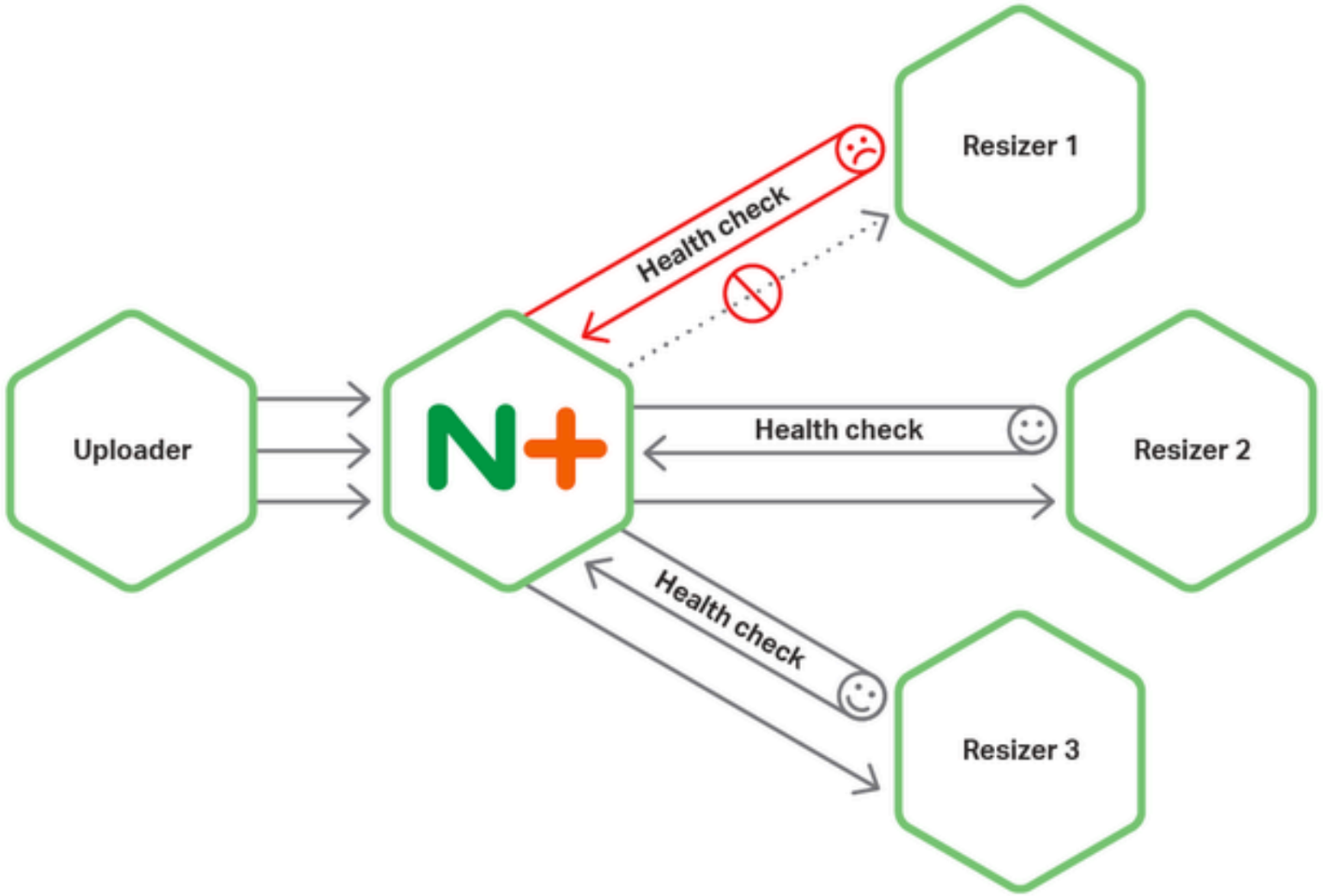
# Key Benefits

# 1. Technology heterogeneity

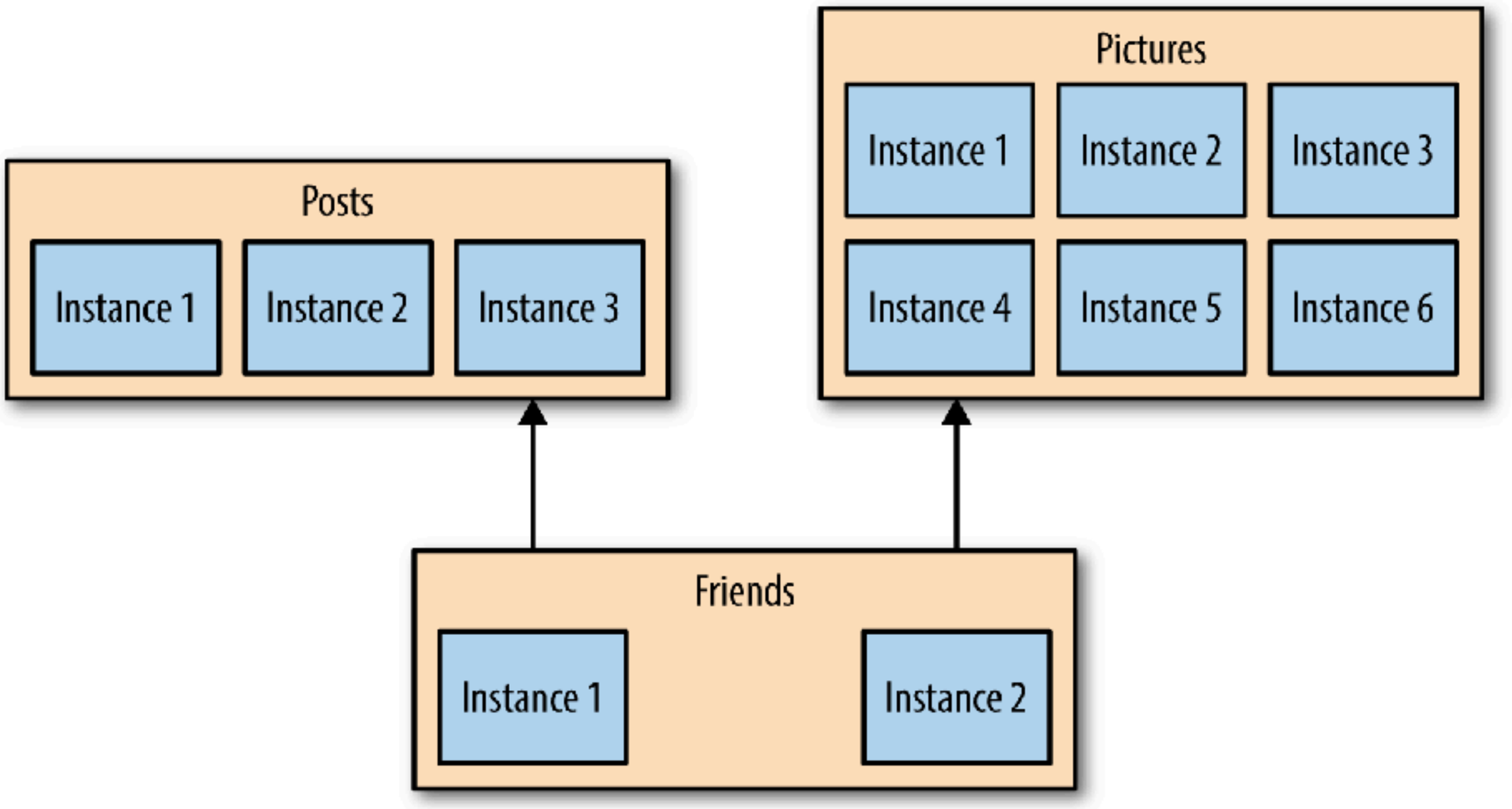
The right tool for each job



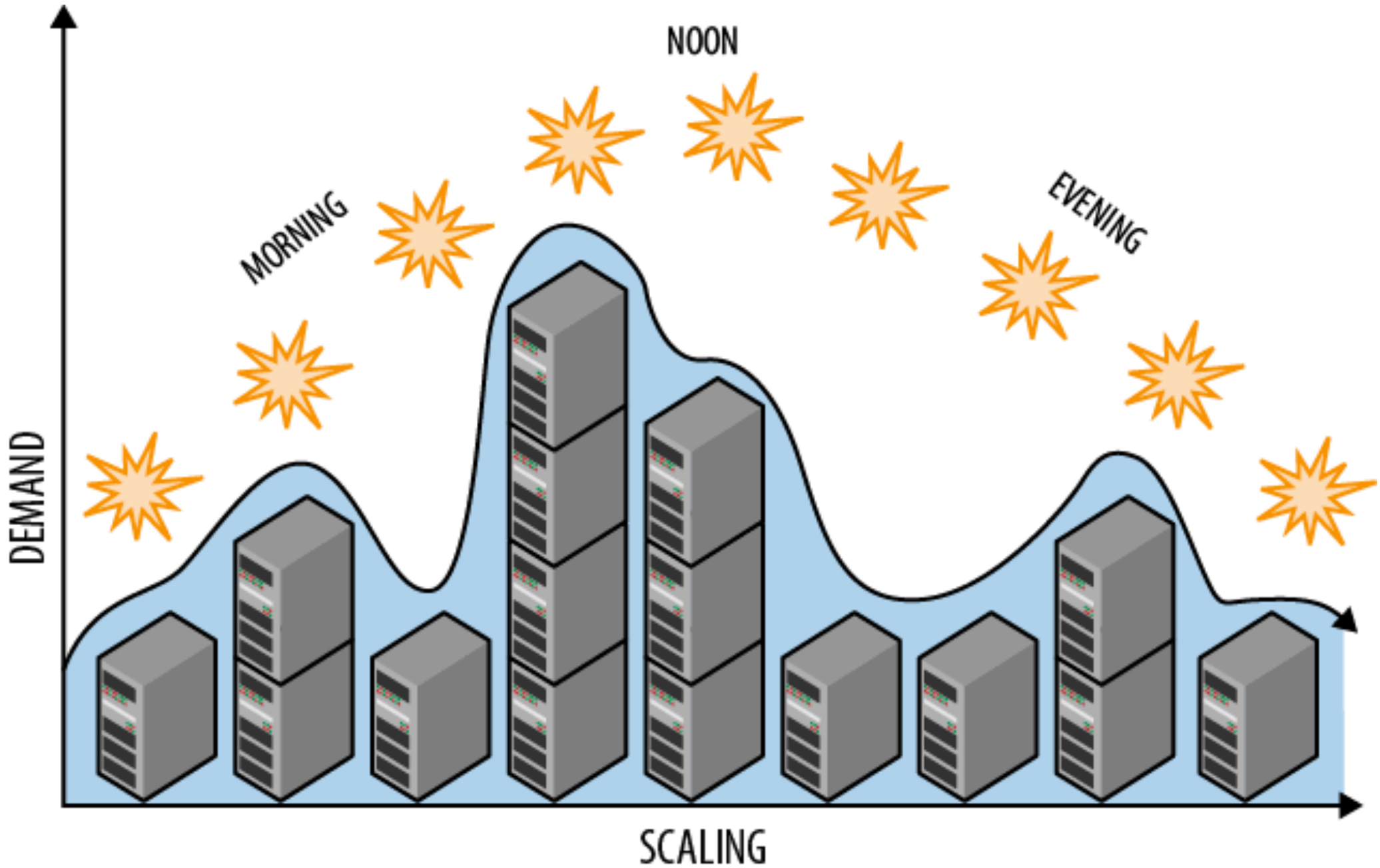
# 2. Resilience



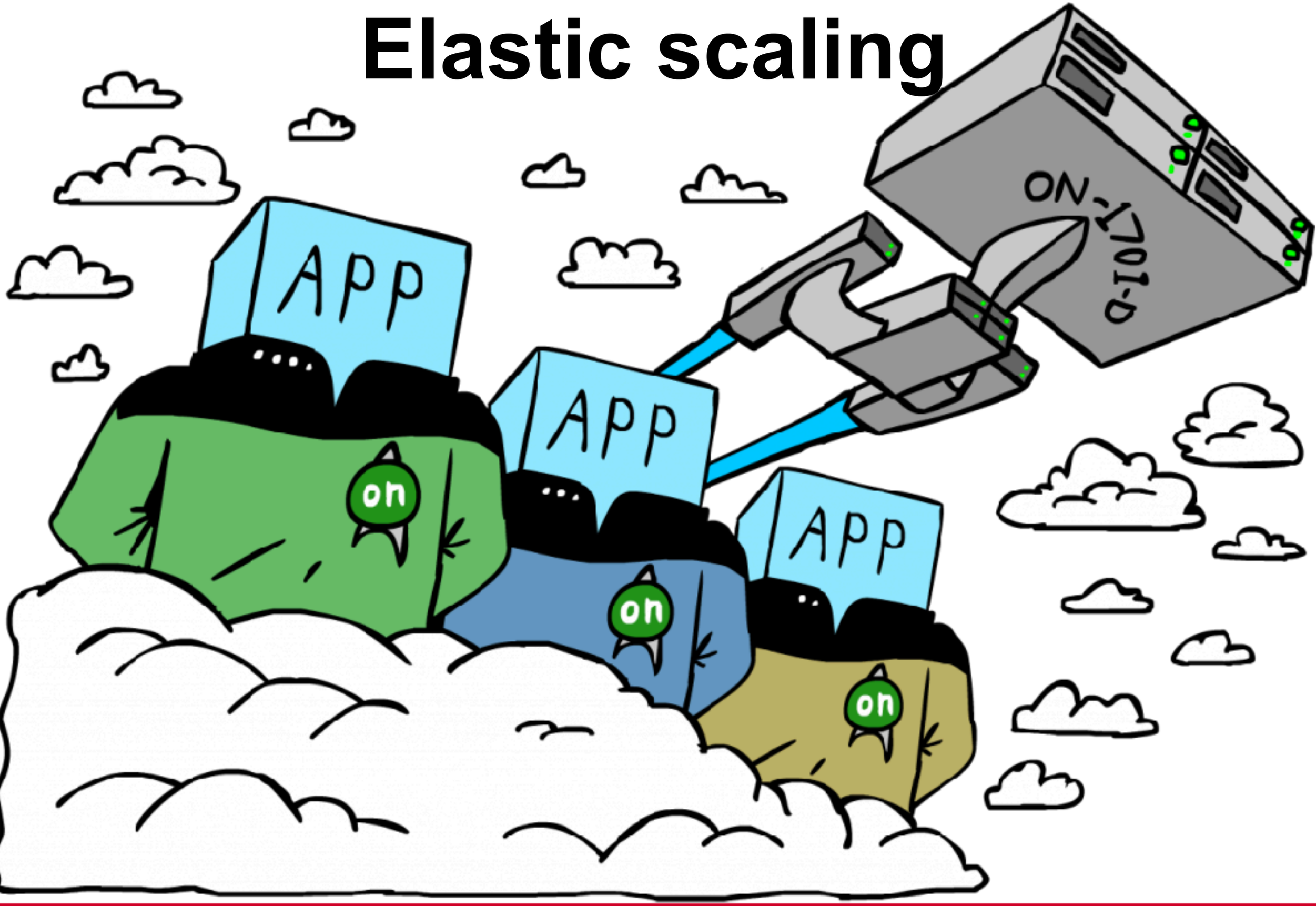
# 3. Scaling



# Elastic scaling

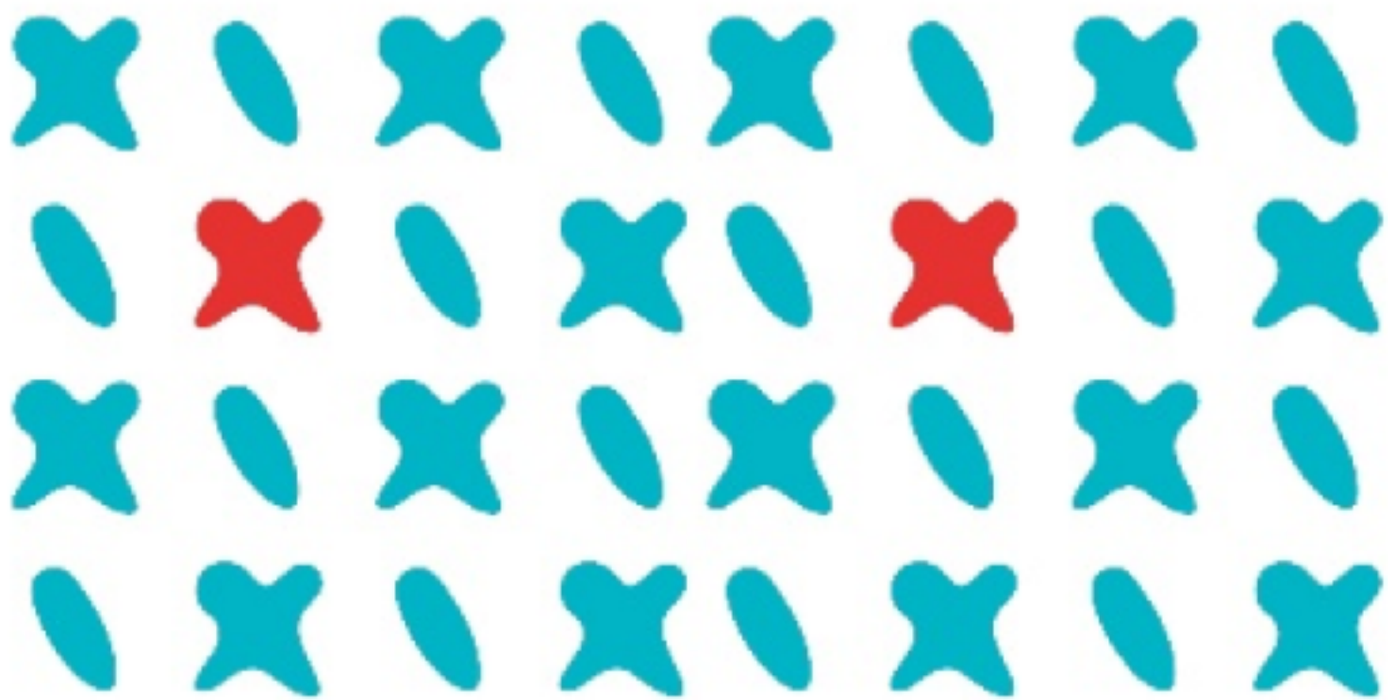


# Elastic scaling



# 4. Ease of deployment

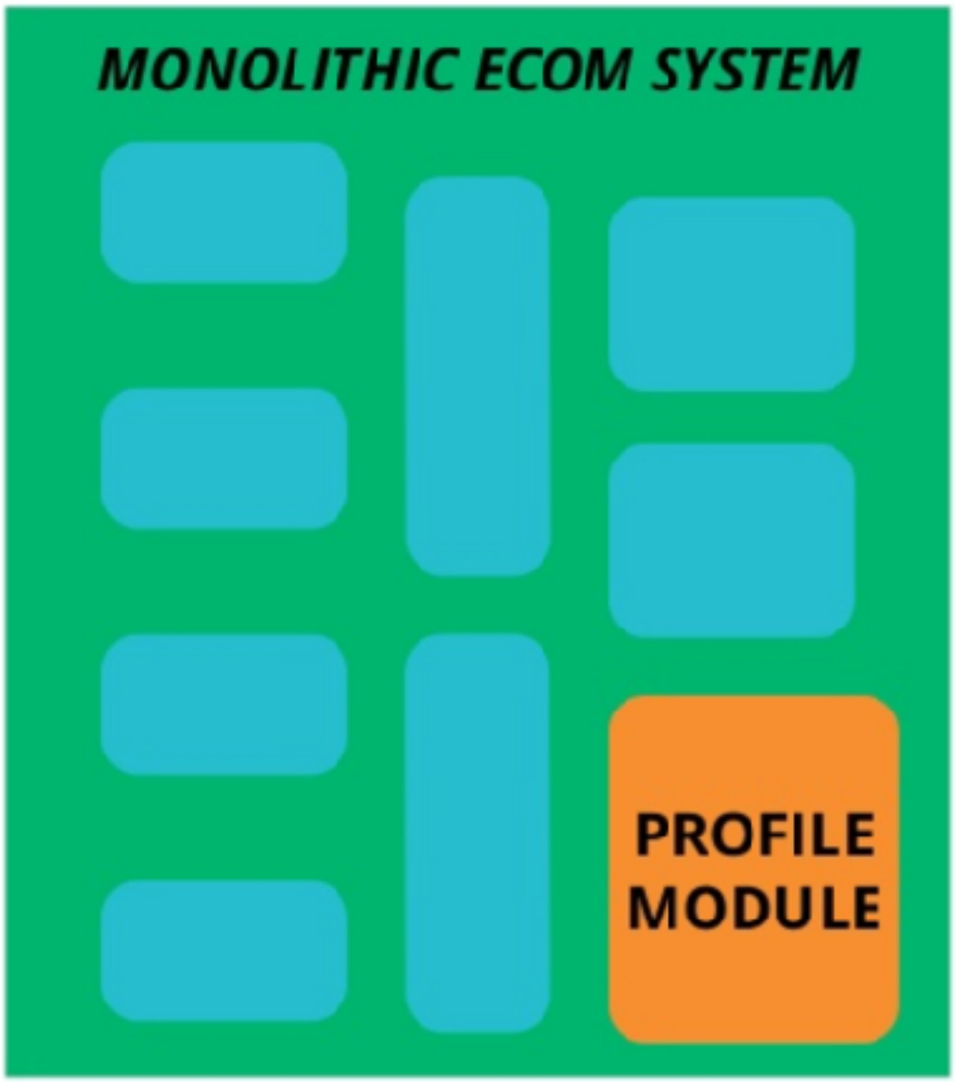
Deploys are faster, independent and problems can be isolated more easily



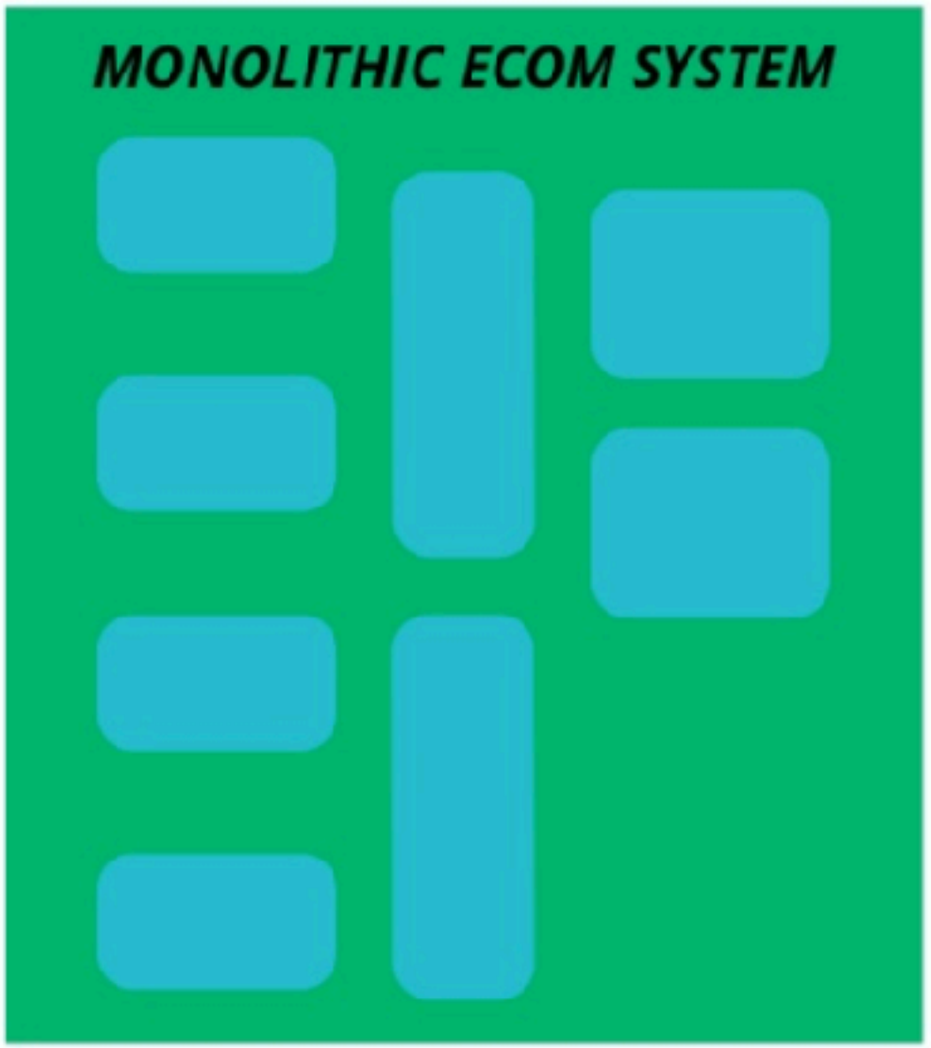
# 5. Organization alignment

Small teams and smaller codebases

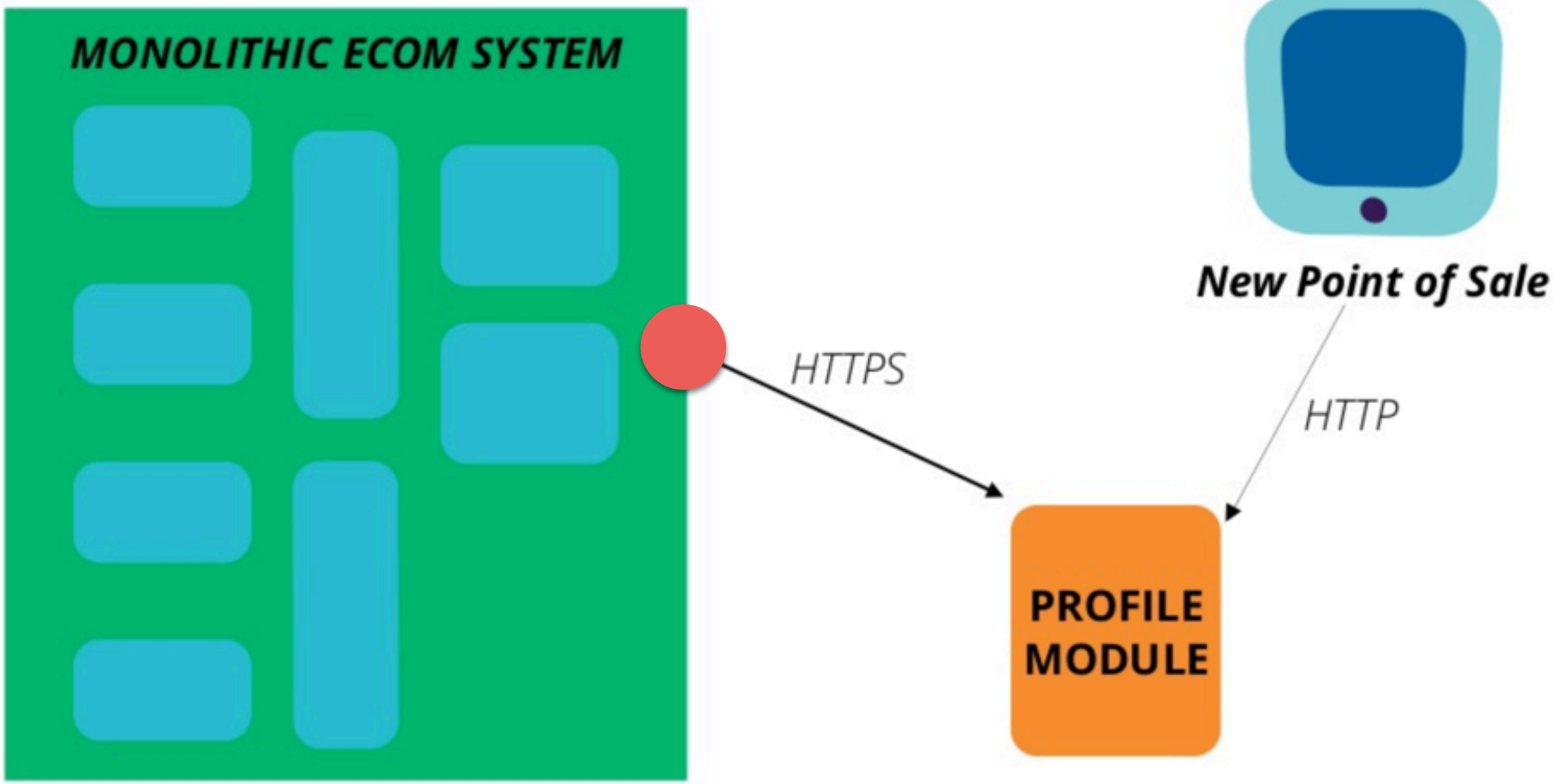
# 6. Composability and replaceability



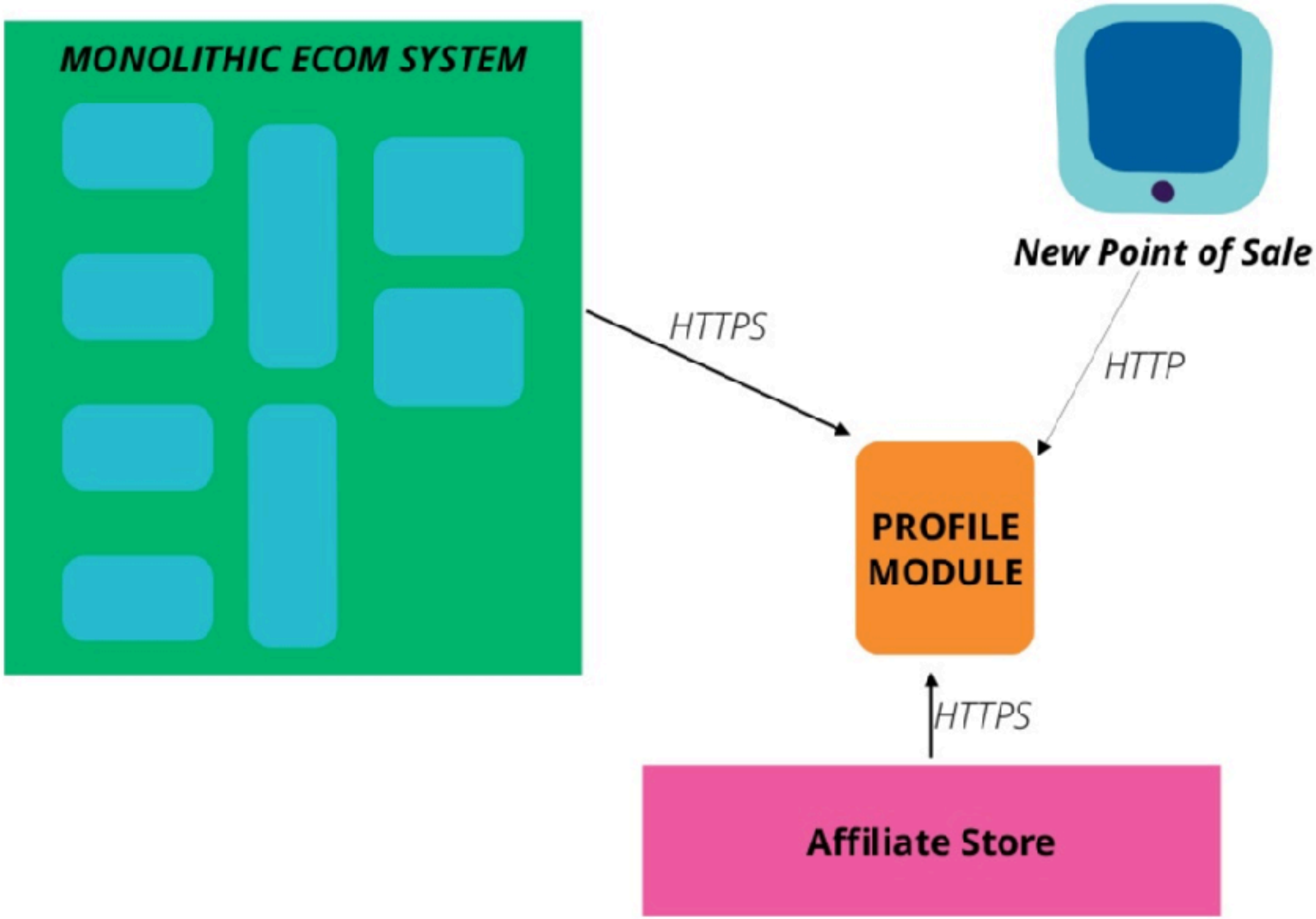
# 6. Composability and replaceability




# 6. Composability and replaceability



# 6. Composability and replaceability



# Characteristics

# 1. Responsible for a single capability

# Types of capabilities

Business capability
Technical capability

Booking System

# Booking System

Web

Mobile

Payment

Listing

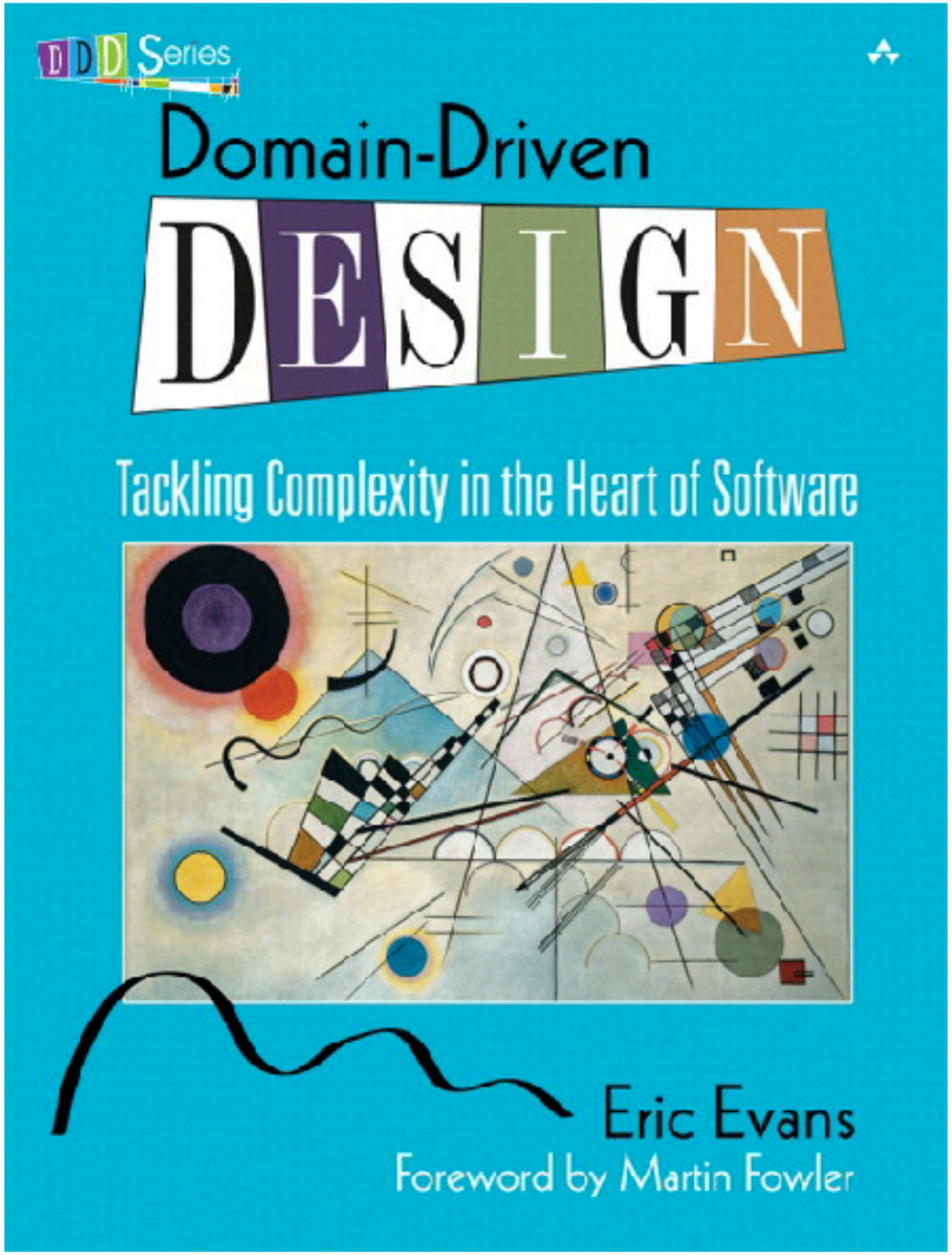
Search

Booking

Marketing

Backoffice

Reporing



# 2. Individually deployable

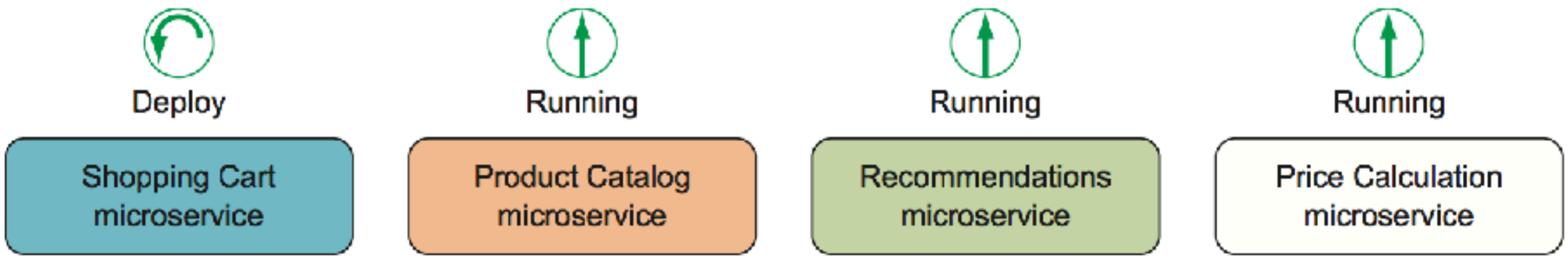



**Figure 1.2 Other microservices continue to run while the Shopping Cart microservice is being deployed.**

# 3. Consists of one or more processes

**Problematic process boundary. Microservices should run in separate processes to avoid coupling.**

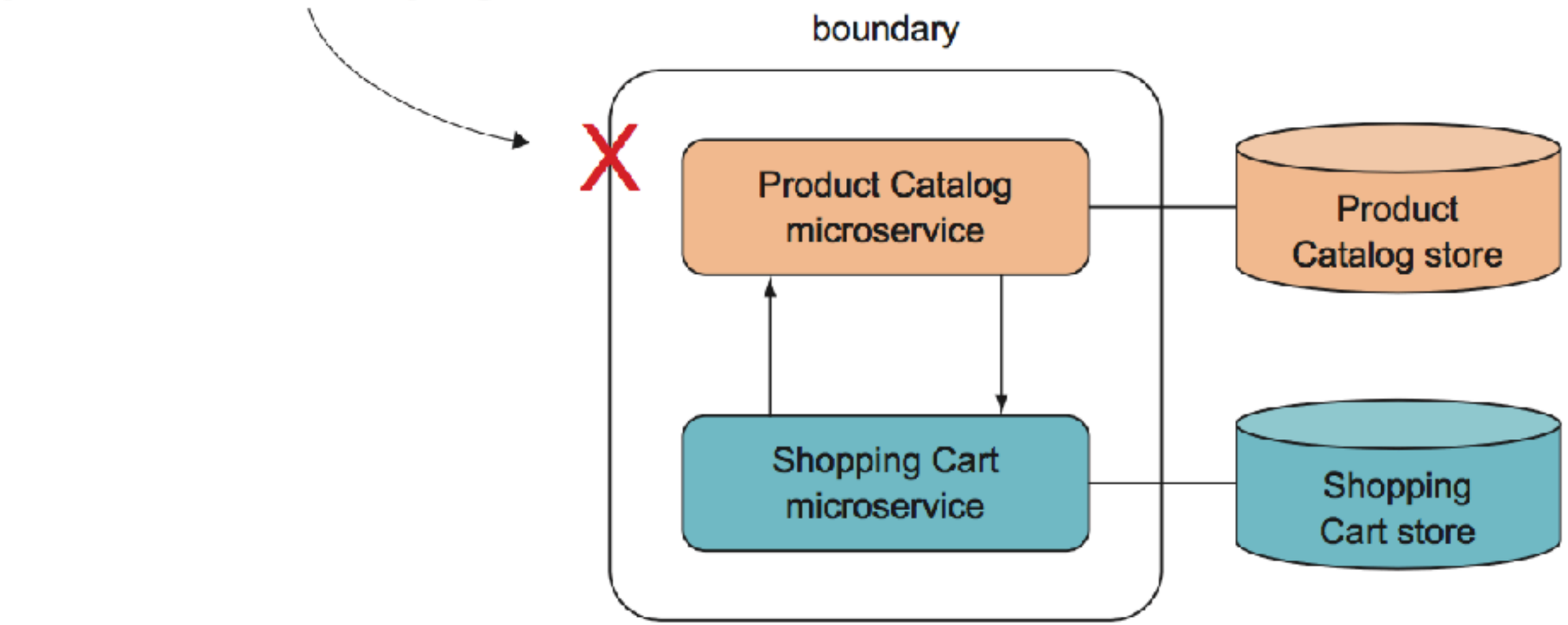



**Figure 1.3 Running more than one microservice within a process leads to high coupling.**

**Problematic process boundary. Microservices should run in separate processes to avoid coupling.**

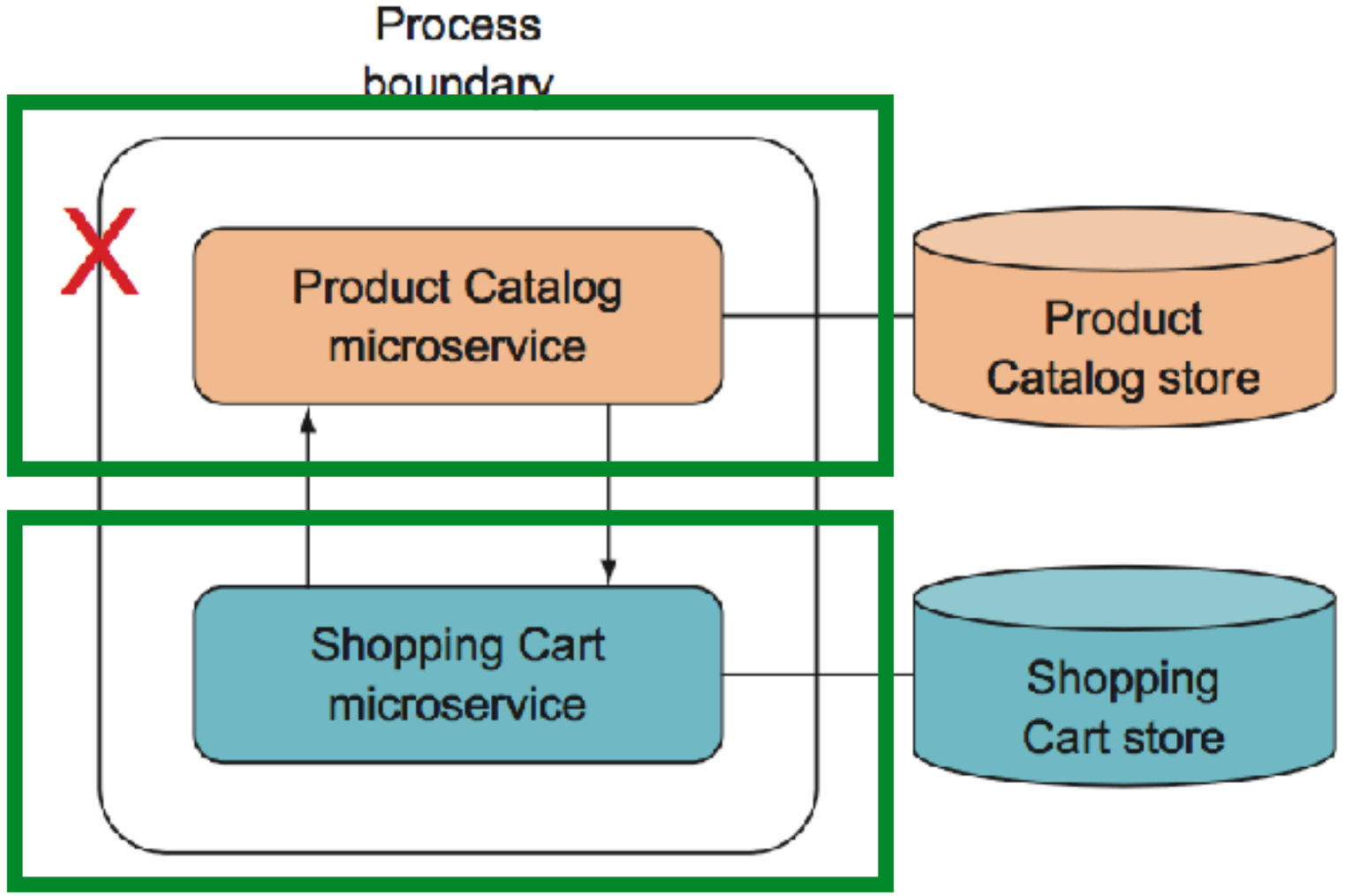



**Figure 1.3 Running more than one microservice within a process leads to high coupling.**

# 4. Own data store

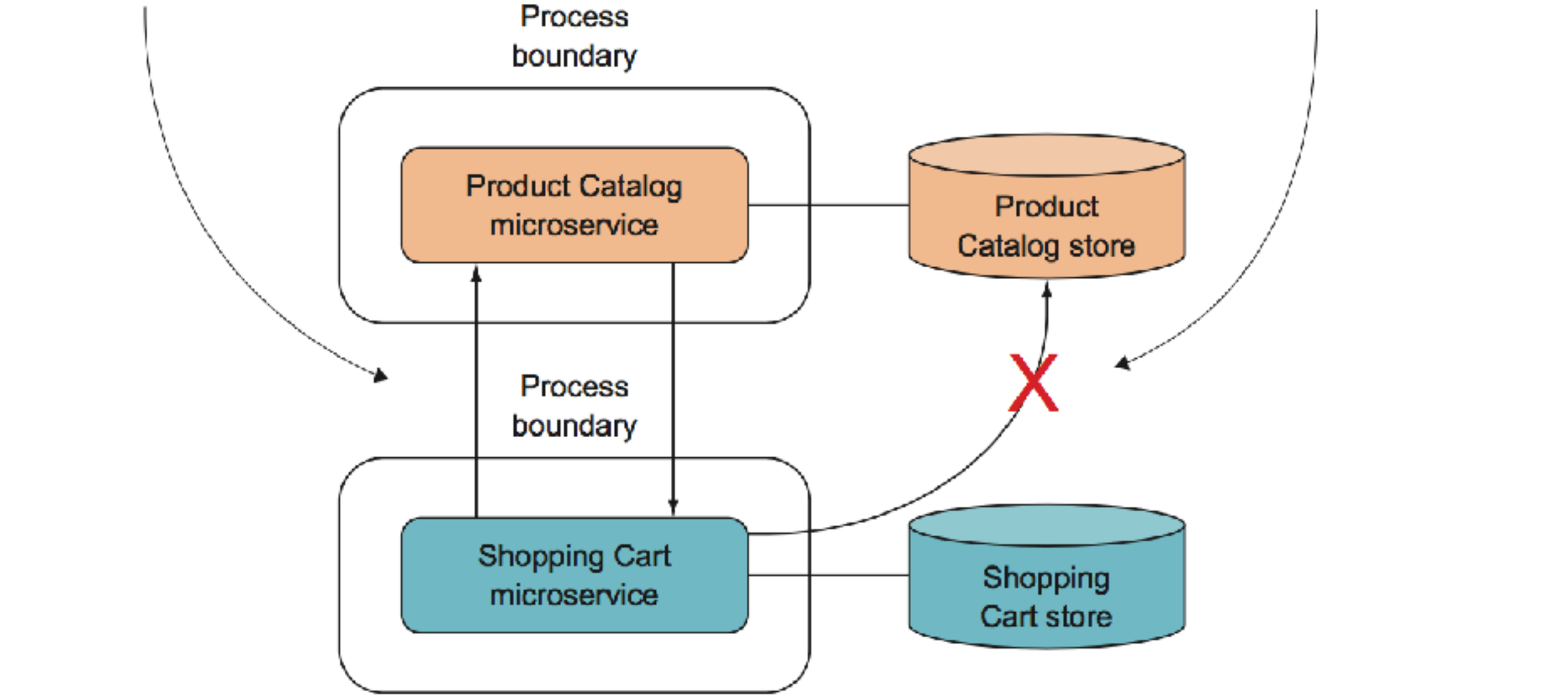



**Figure 1.4 One microservice can't access another's data store.**

All communication with the Product Catalog microservice must go through the public API.

Direct access to the Product Catalog store is not allowed. The Product Catalog microservice owns the Product Catalog store.

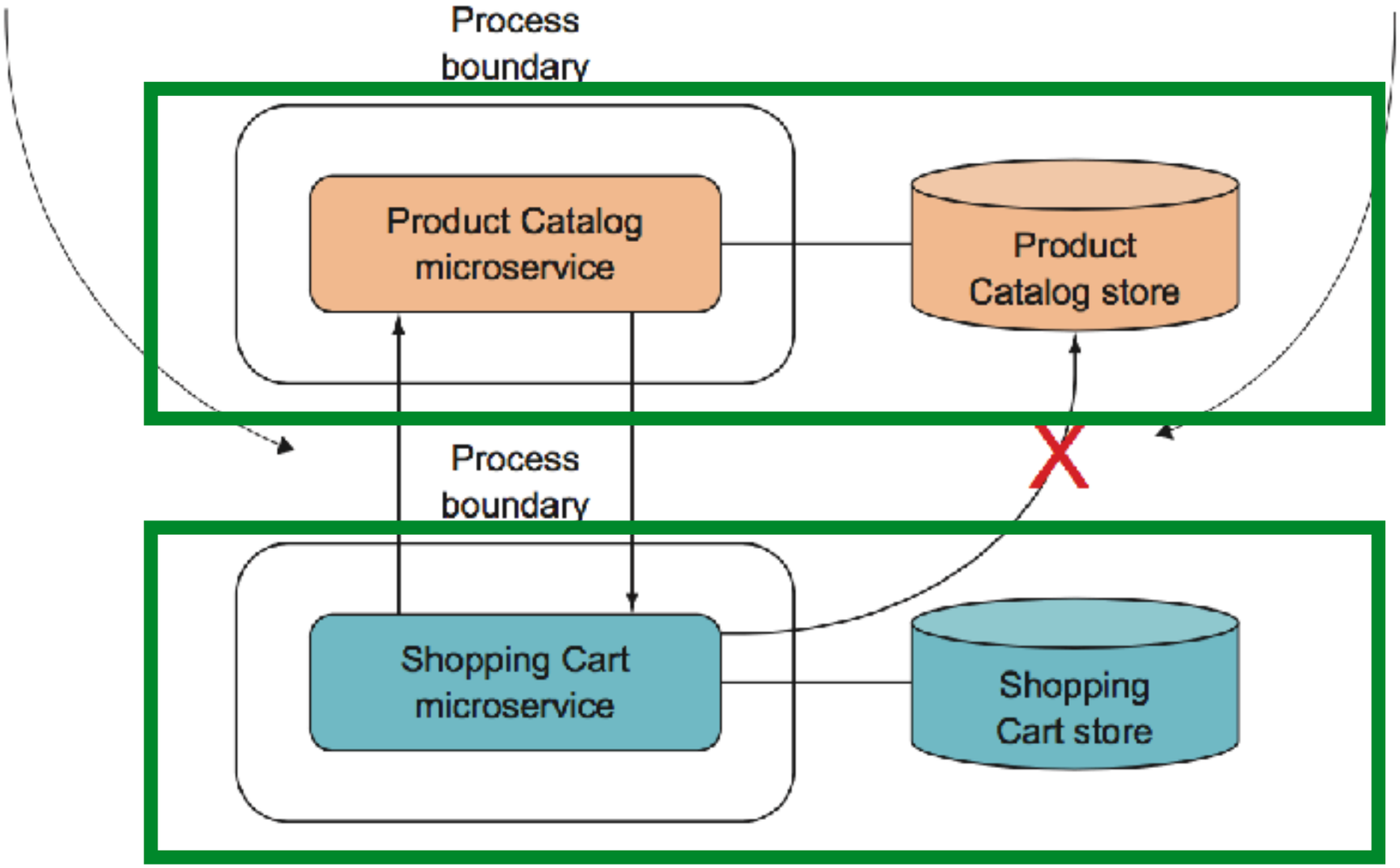
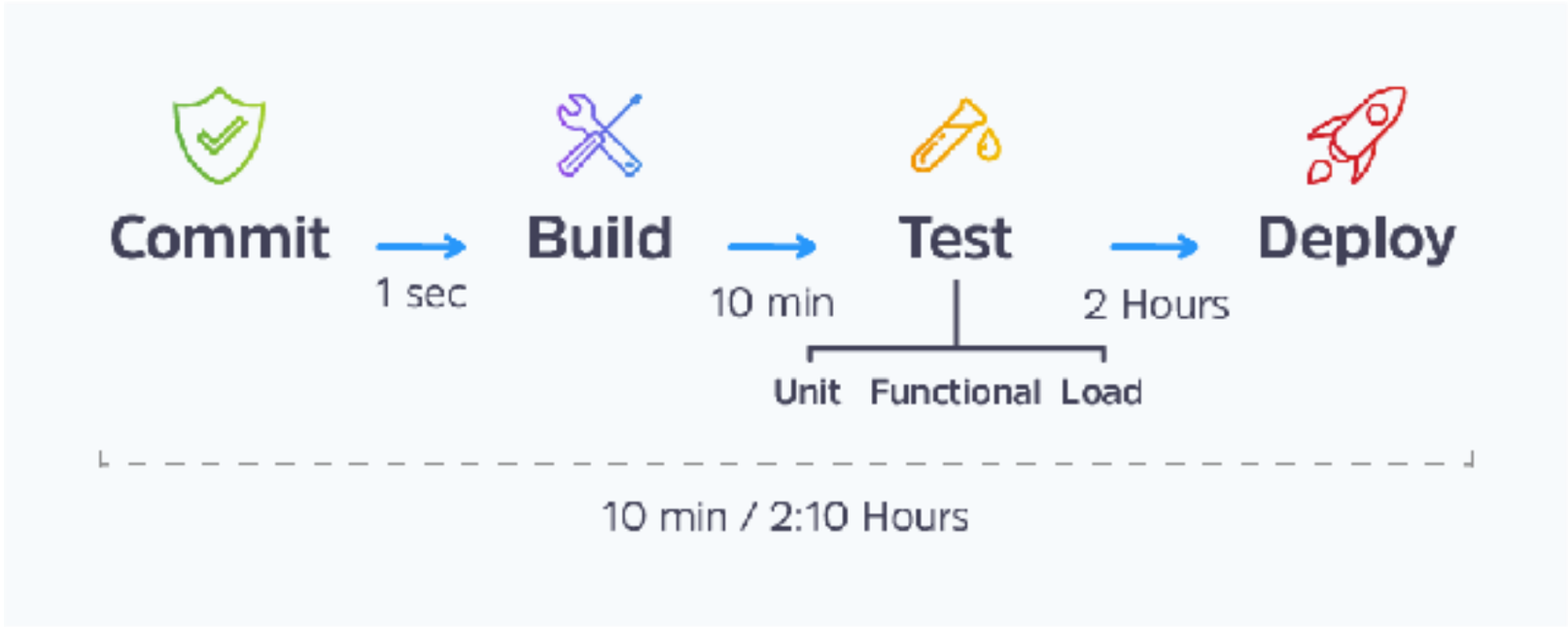



**Figure 1.4 One microservice can't access another's data store.**

# 5. Small team can maintain



# 6. Replaceable

# Problems
# with Microservices ?

Challenges

~~Problems~~

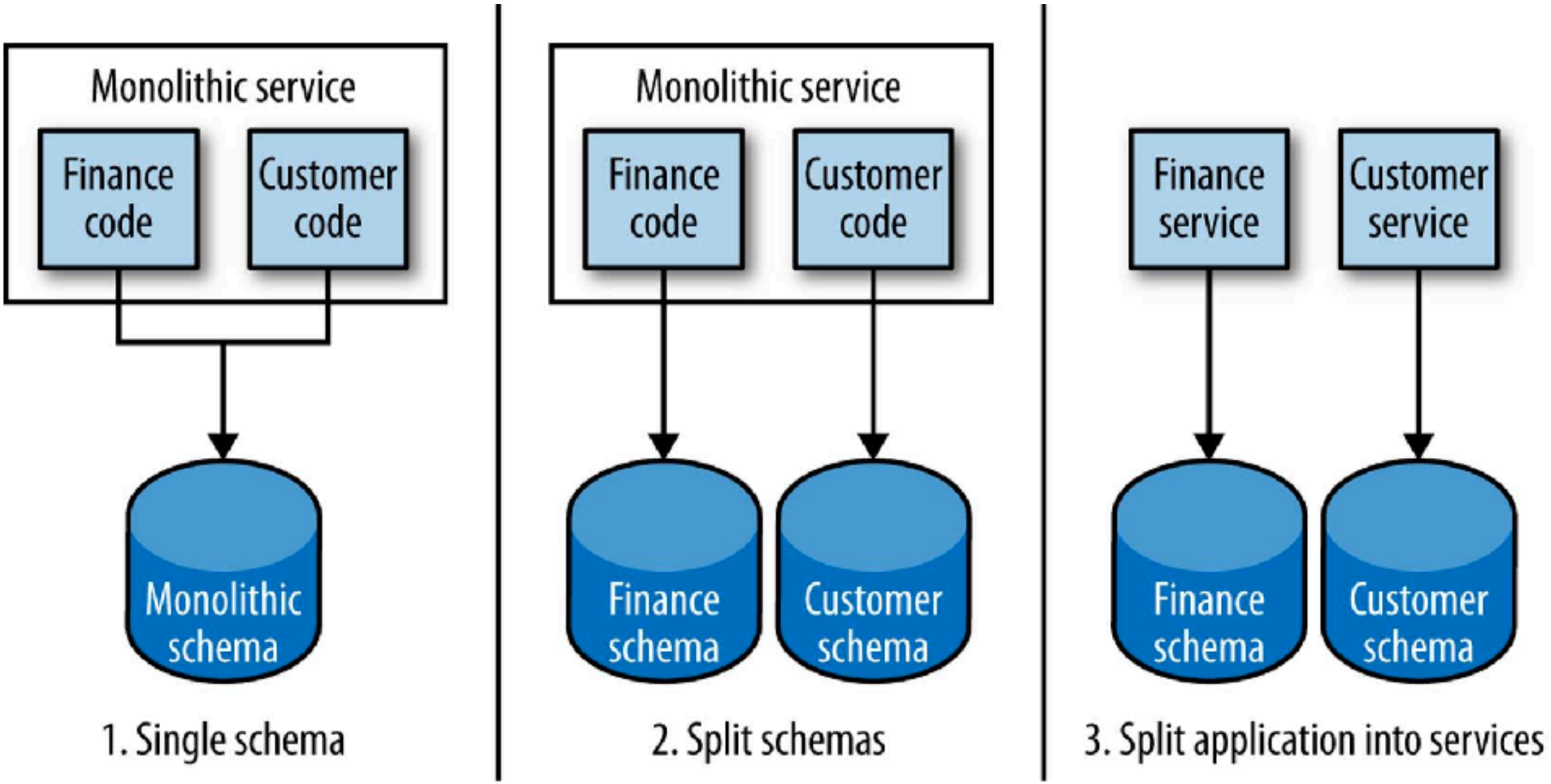
# with Microservices ?

# 1. How to define the boundaries of each microservices ?

Premature splitting is the root of all **evil**.

Every time you make the decision to split out a new microservices, there's a **risk** of ending up with a bloated app.

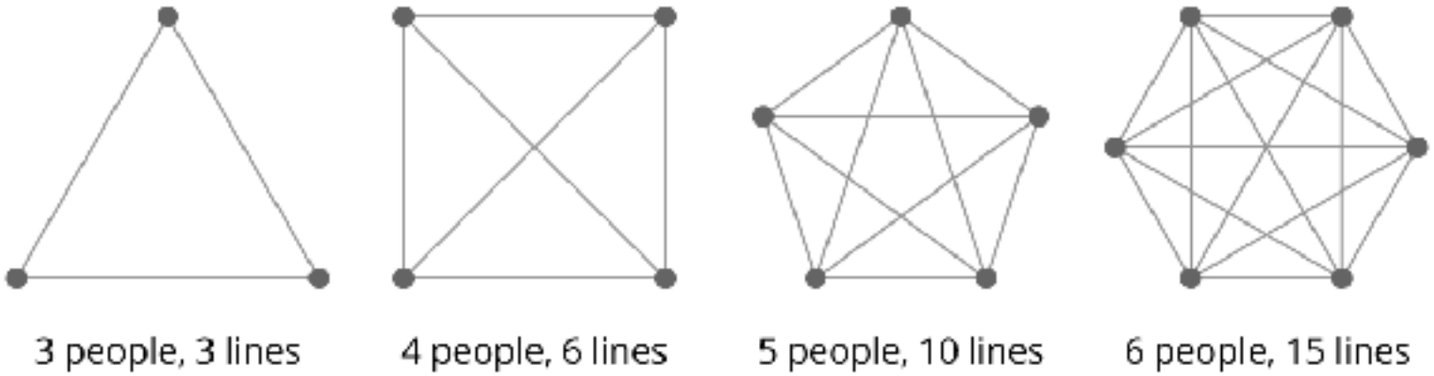
# Example



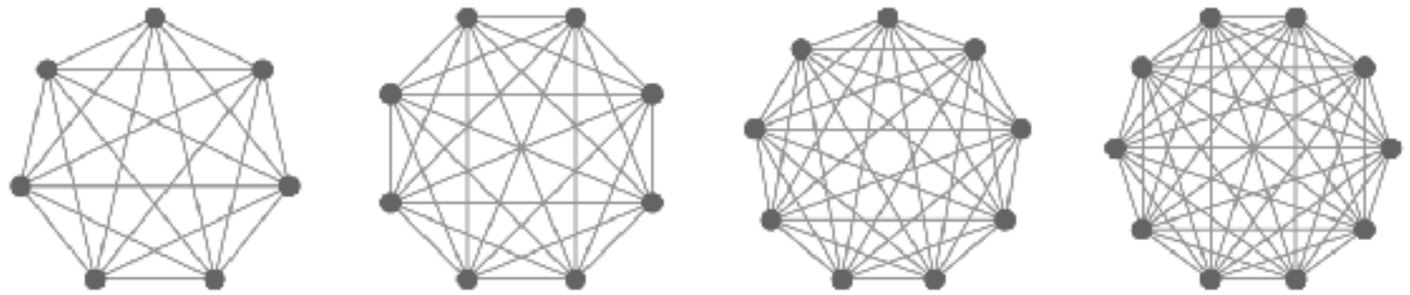
# How Small should be Microservices ?

# Two pizzas team

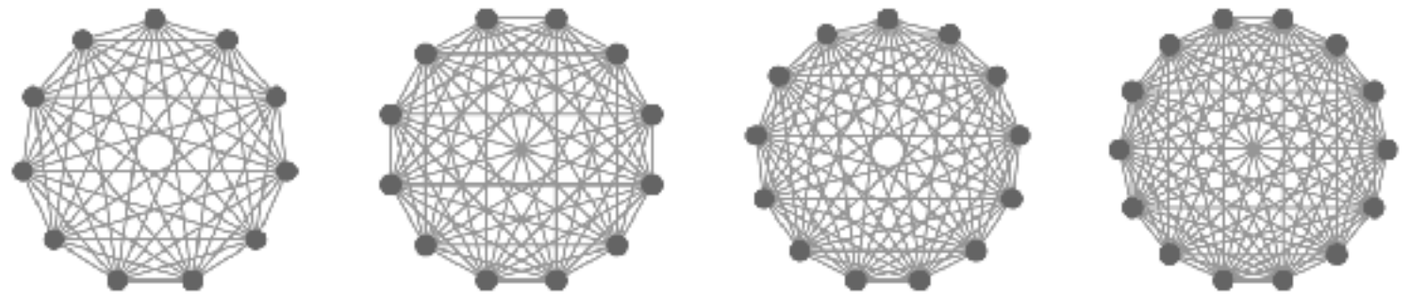
# Effective communication



3 people, 3 lines | 4 people, 6 lines | 5 people, 10 lines | 6 people, 15 lines



7 people, 21 lines | 8 people, 28 lines | 9 people, 36 lines | 10 people, 45 lines



11 people, 55 lines | 12 people, 66 lines | 13 people, 78 lines | 14 people, 91 lines

# Single Responsibility Principle

# Isolation from changes perspective

1

2

3

# Isolation from business critical services

1

2

3

# Team setup perspective

1

2

3

# Technology perspective

1

2

3

# Data consistency and transaction boundary perspective

1

2

3

# Non-functional requirement perspective

1

2

3

# Try by yourself

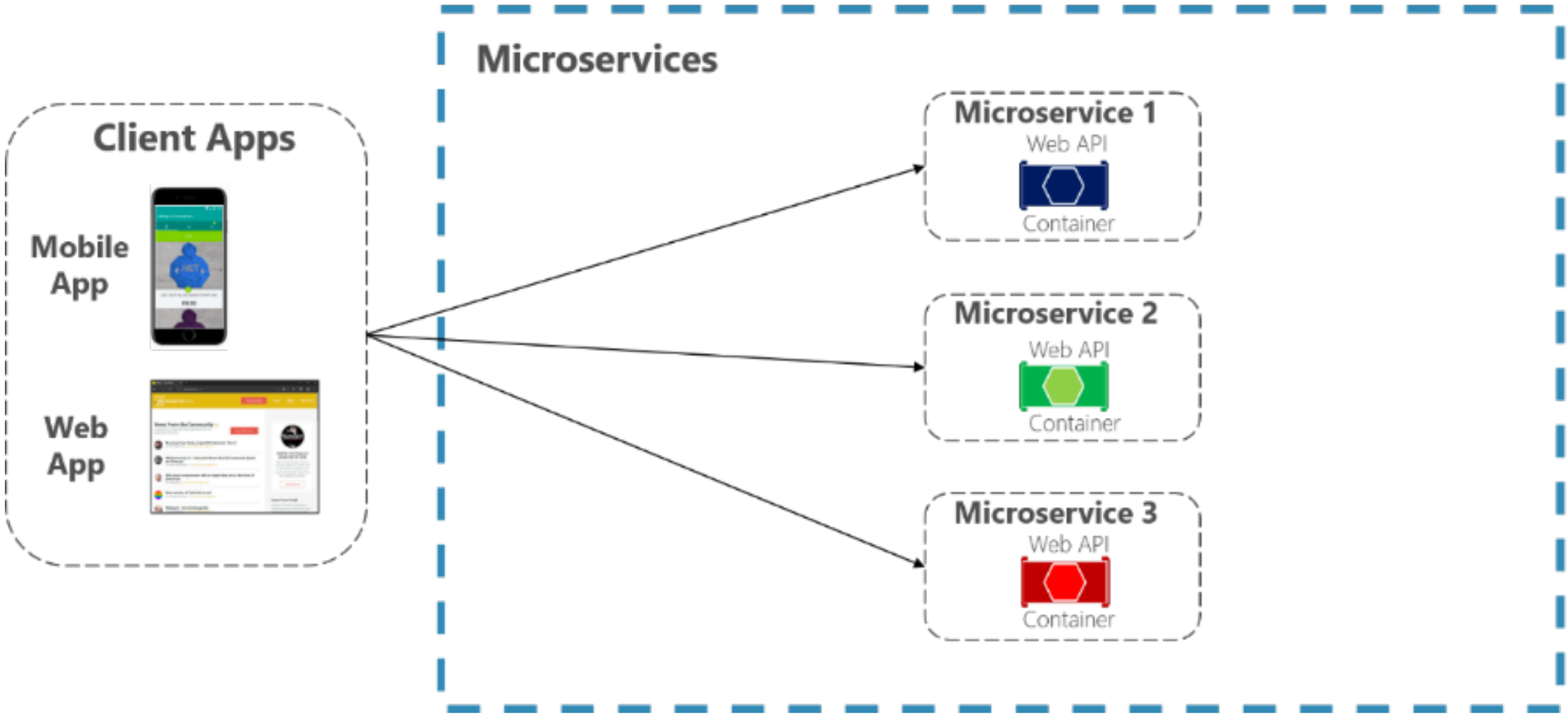
# 2. How to create queries that retrieve data from several Microservices ?

# Popular solutions

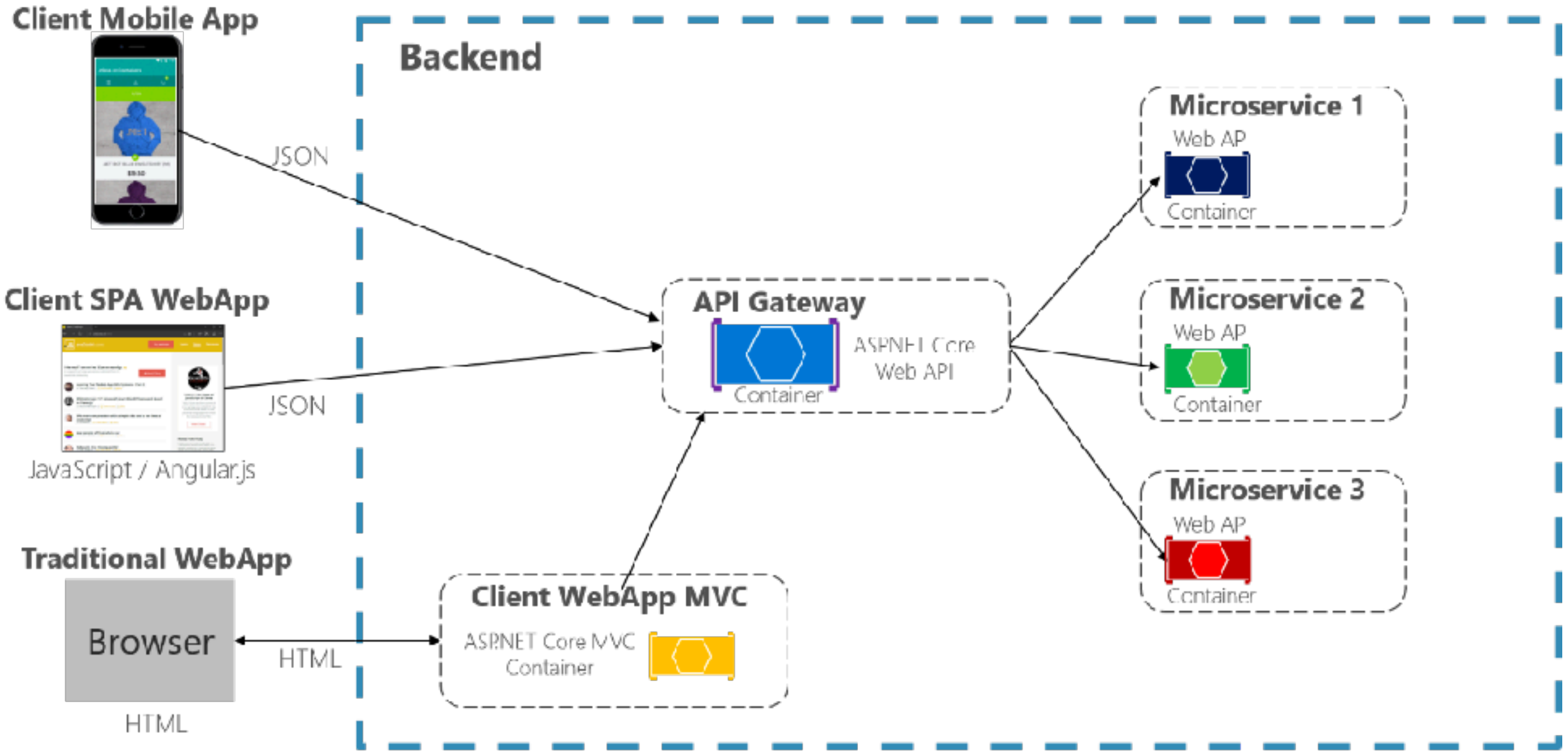
API Gateway
CQRS with query/read tables
Cold data in centralize database

# Direct Client-To-Microservice communication

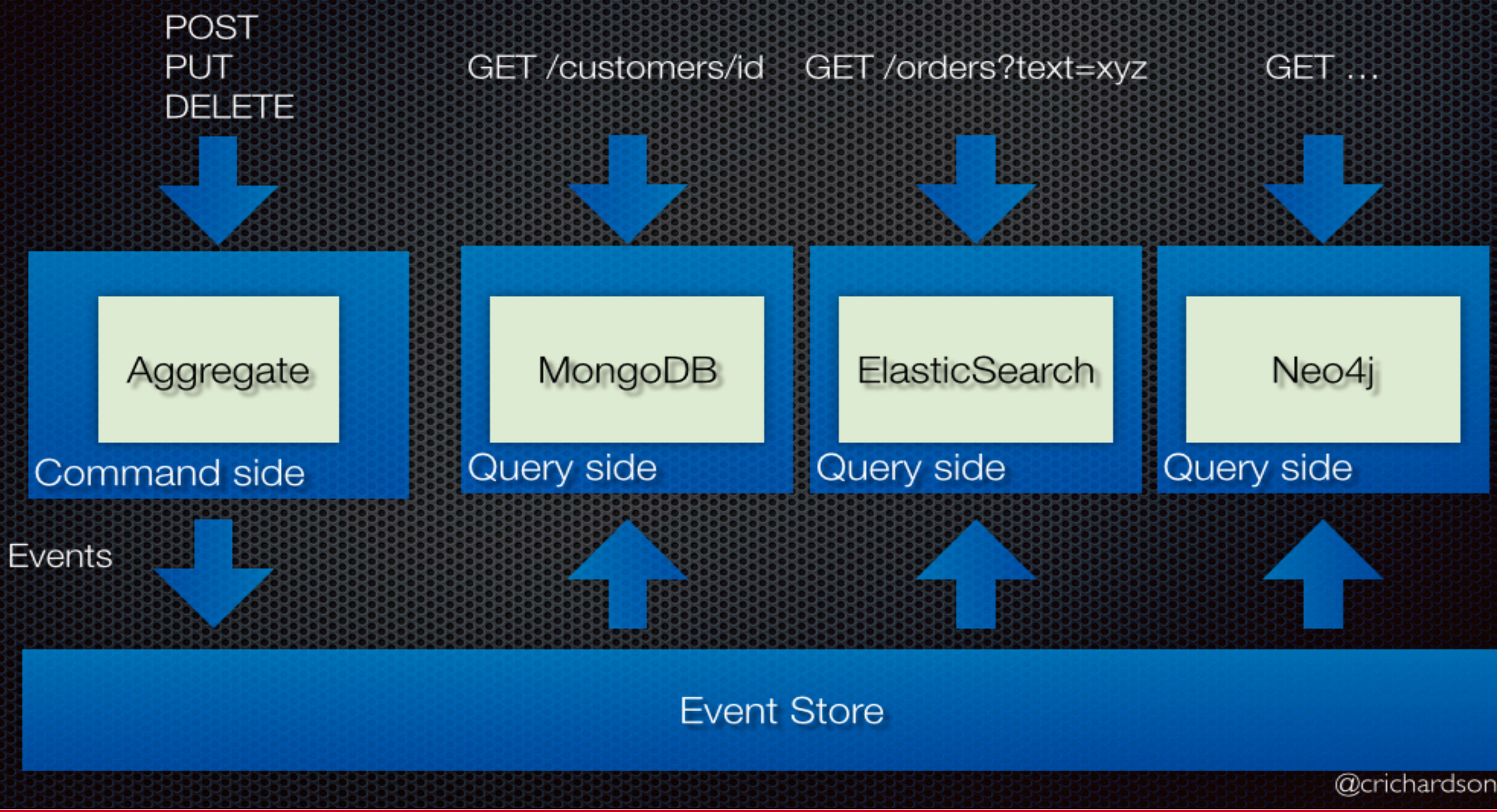
## Architecture



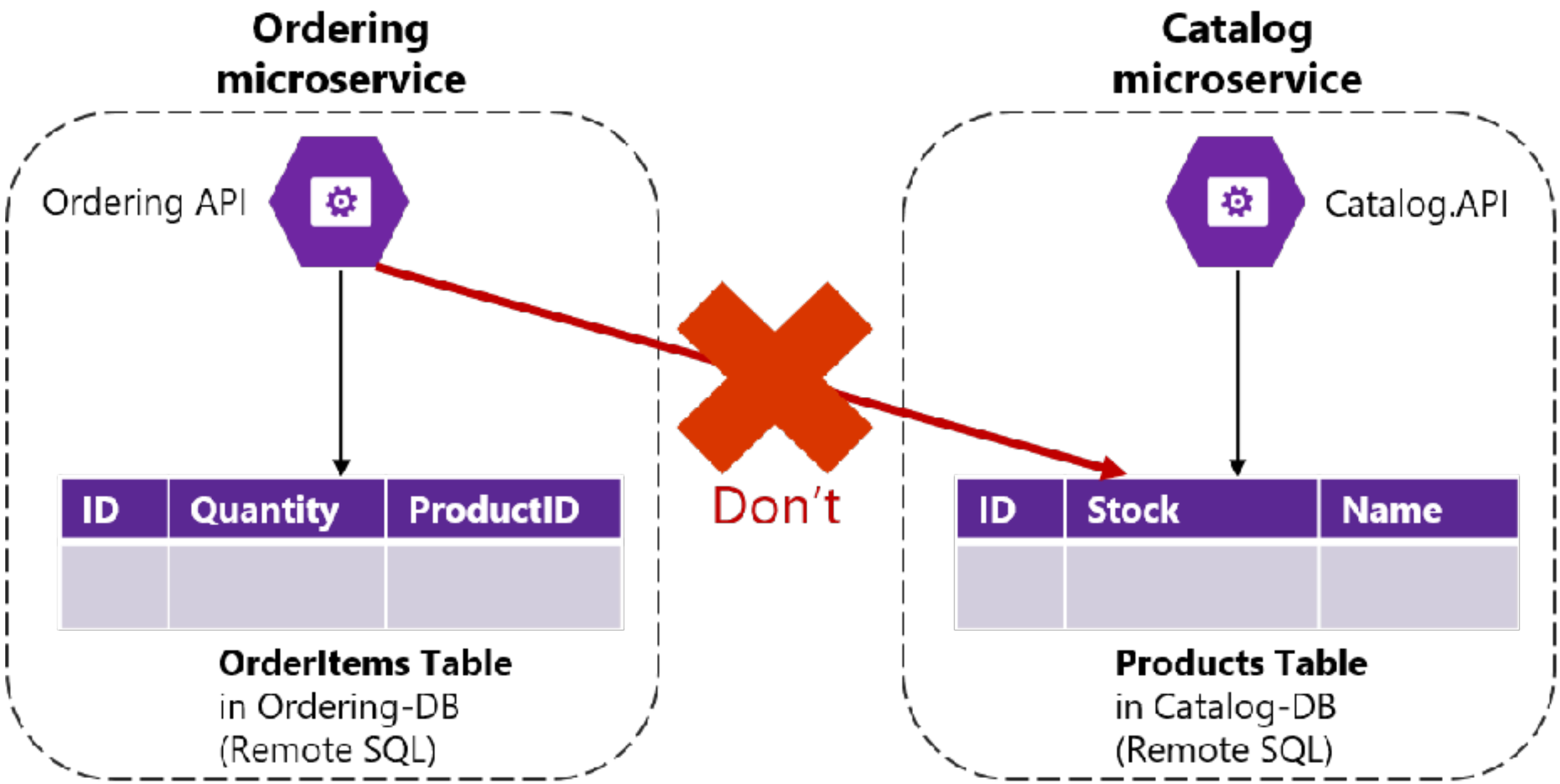
# Using the **API Gateway Service**



# Queries ⇒ database (type)



# 3. How to achieve consistency across multiple Microservices ?




Databases are private per microservice

# CAP Theorem

**Consistency**


**Availability**


**Partition Tolerance**


http://robertgreiner.com/2014/08/cap-theorem-revisited/

# 4. How to design communication across microservices boundaries ?

# Protocols

HTTP and REST
Messaging

# Communication

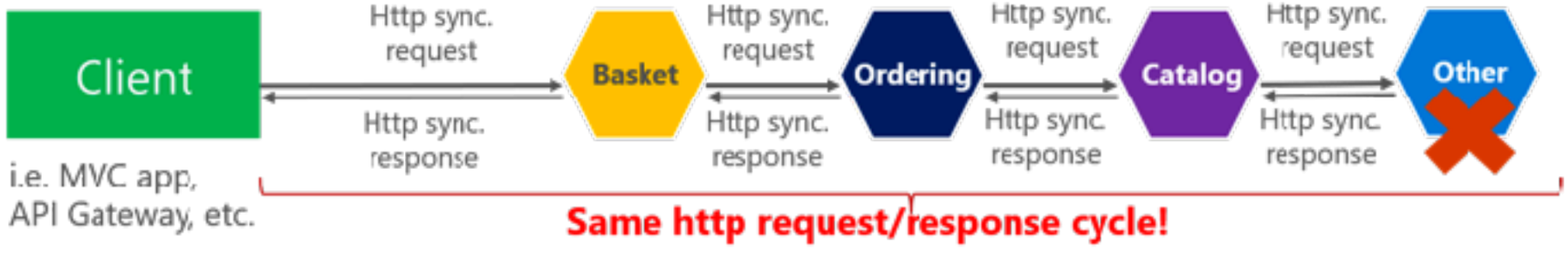
Request-Response model
Observer model

# Communication

## Synchronous vs. async communication across microservices

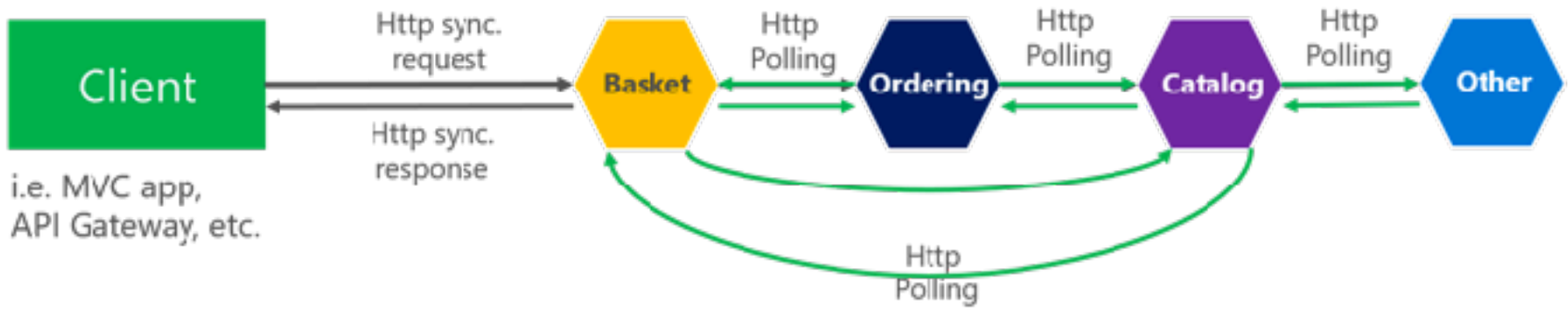
**Anti-pattern**

**Synchronous**
all req./resp. cycle

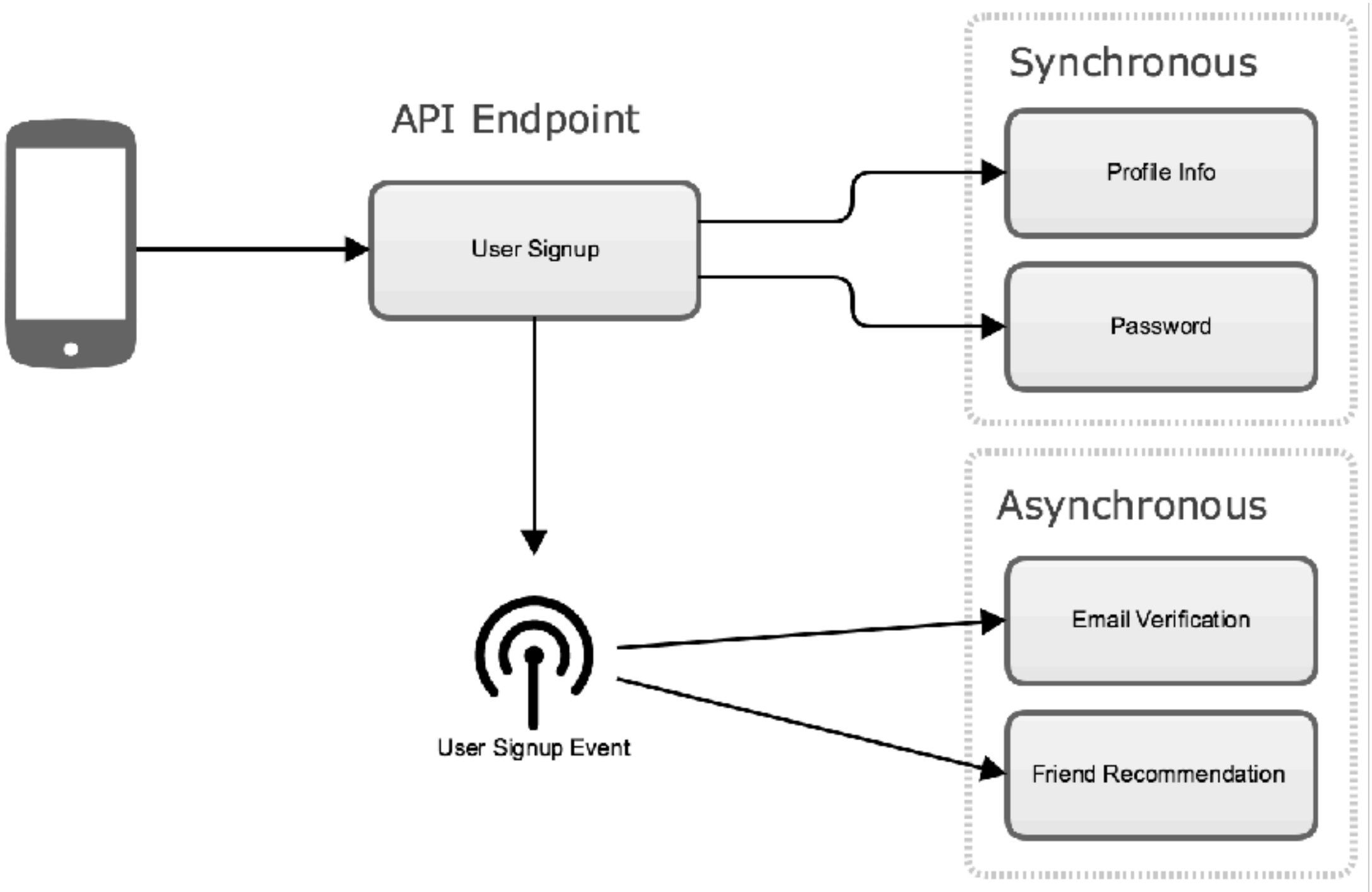



**Asynchronous**
Comm. across
internal microservices
(EventBus: i.e. **AMPQ**)

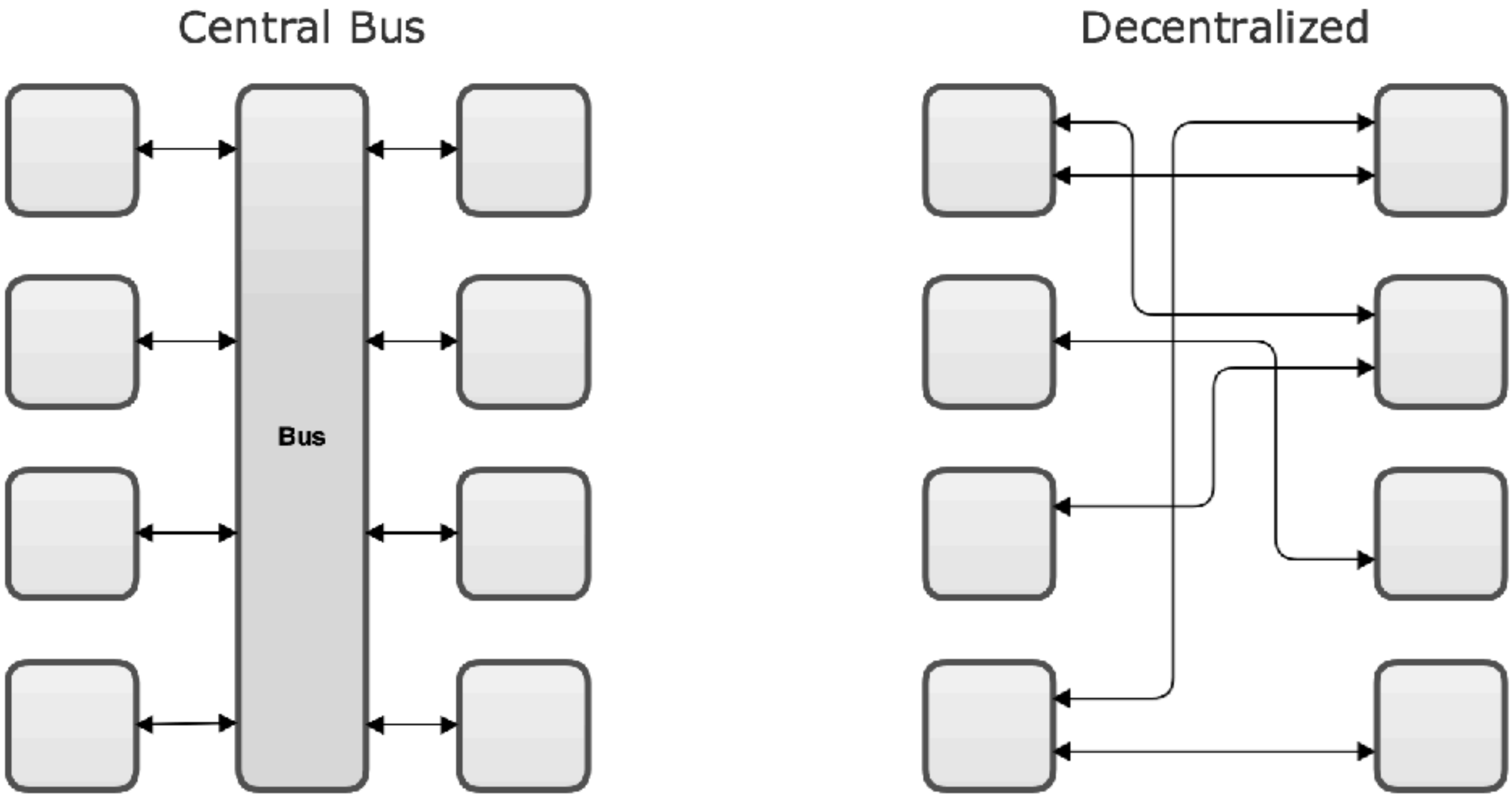

**"Asynchronous"**
Comm. across
internal microservices
(Polling: **Http**)



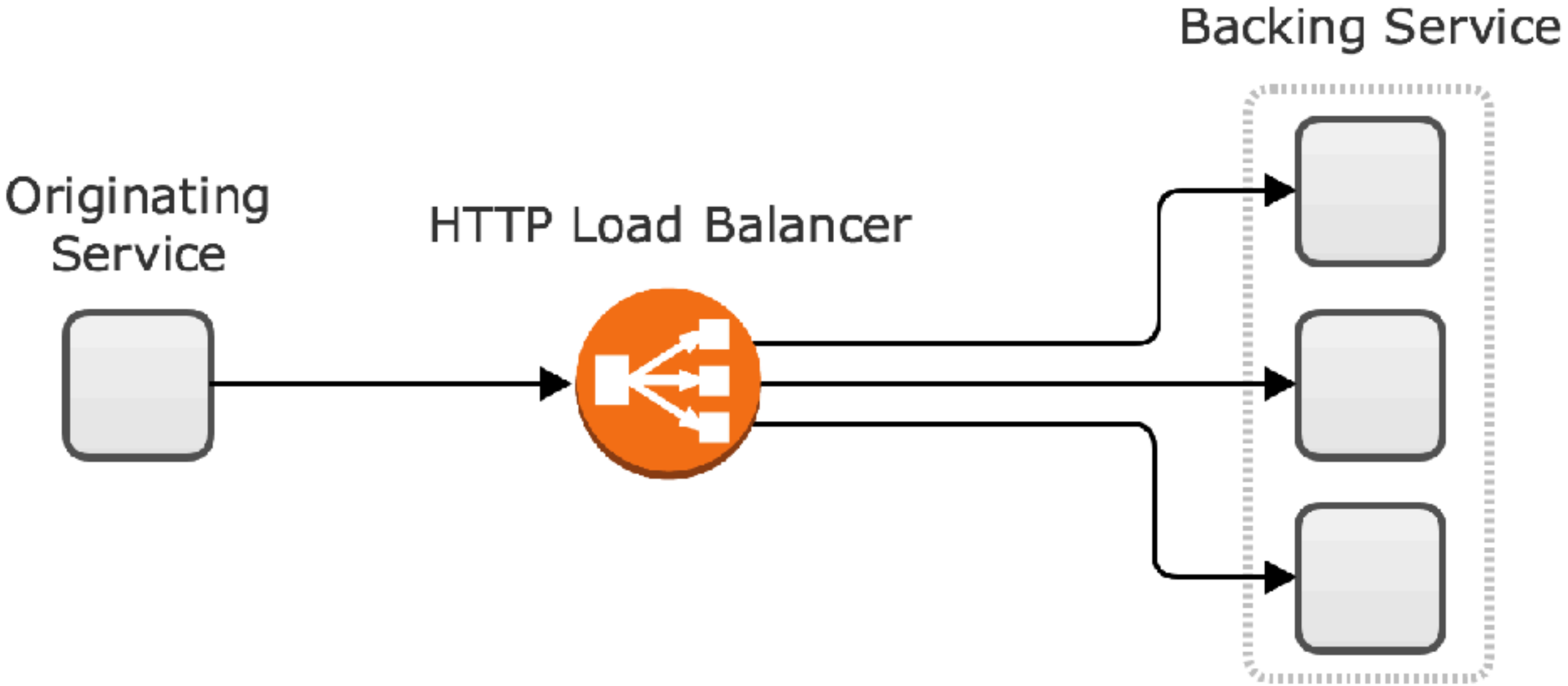
# Communication



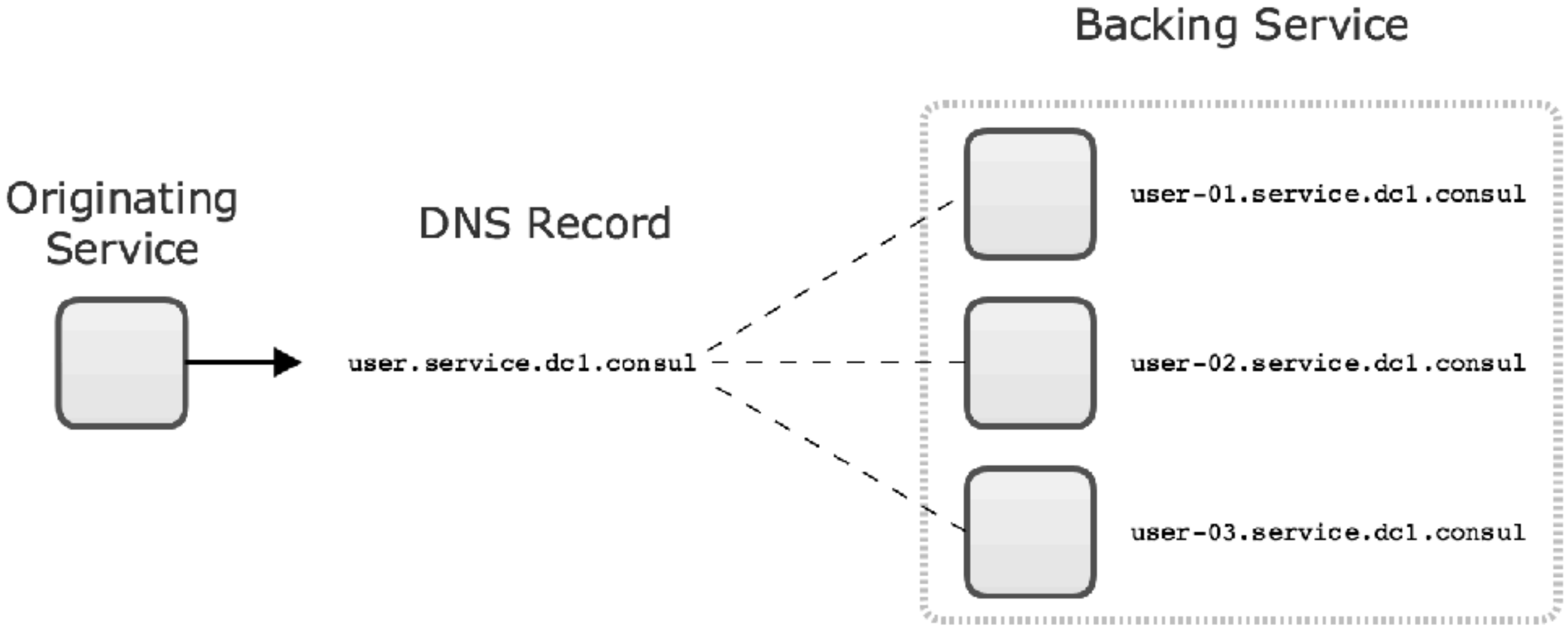
# Anti-pattern :: centralize bus service



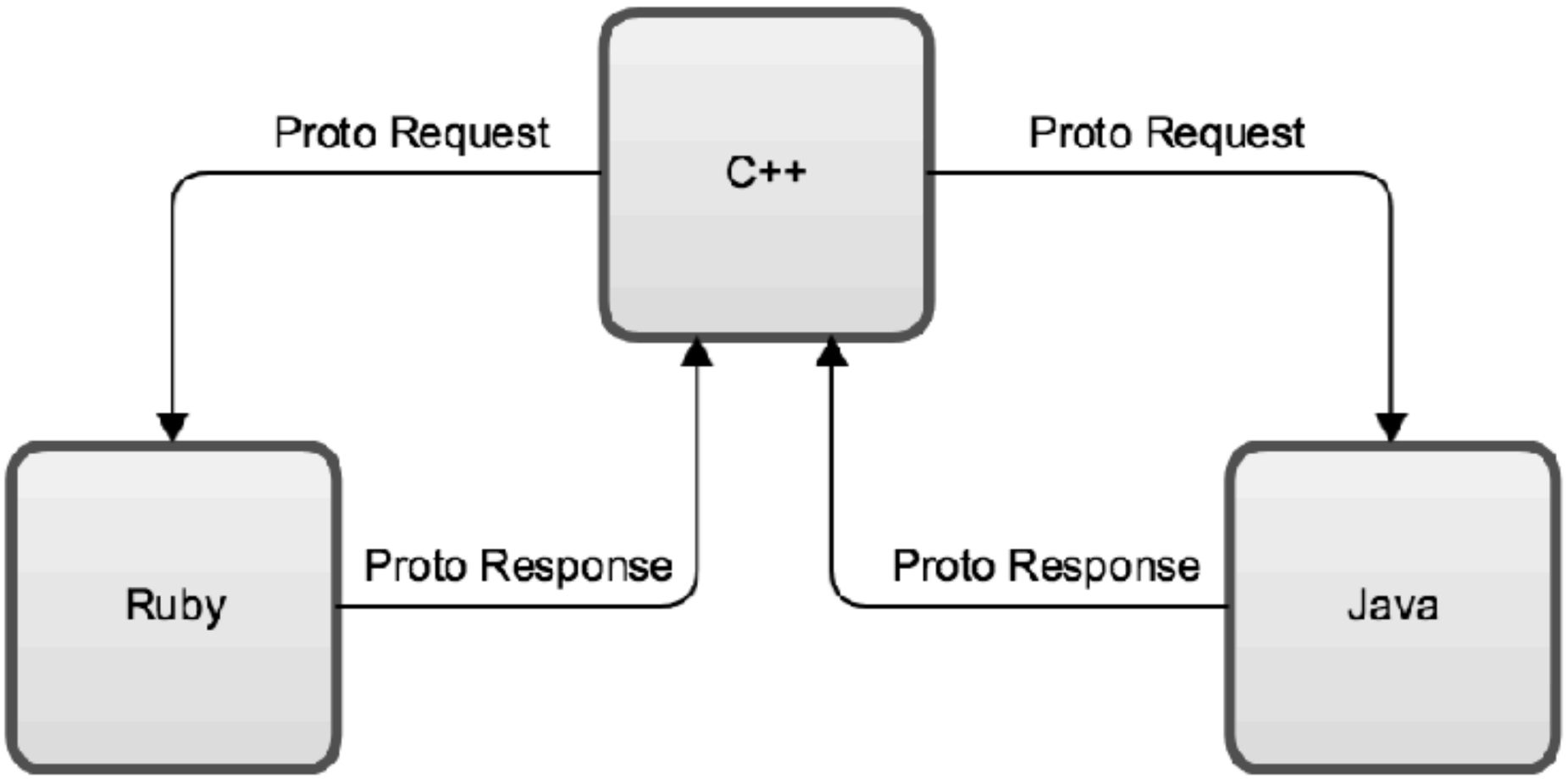
# Request-response model



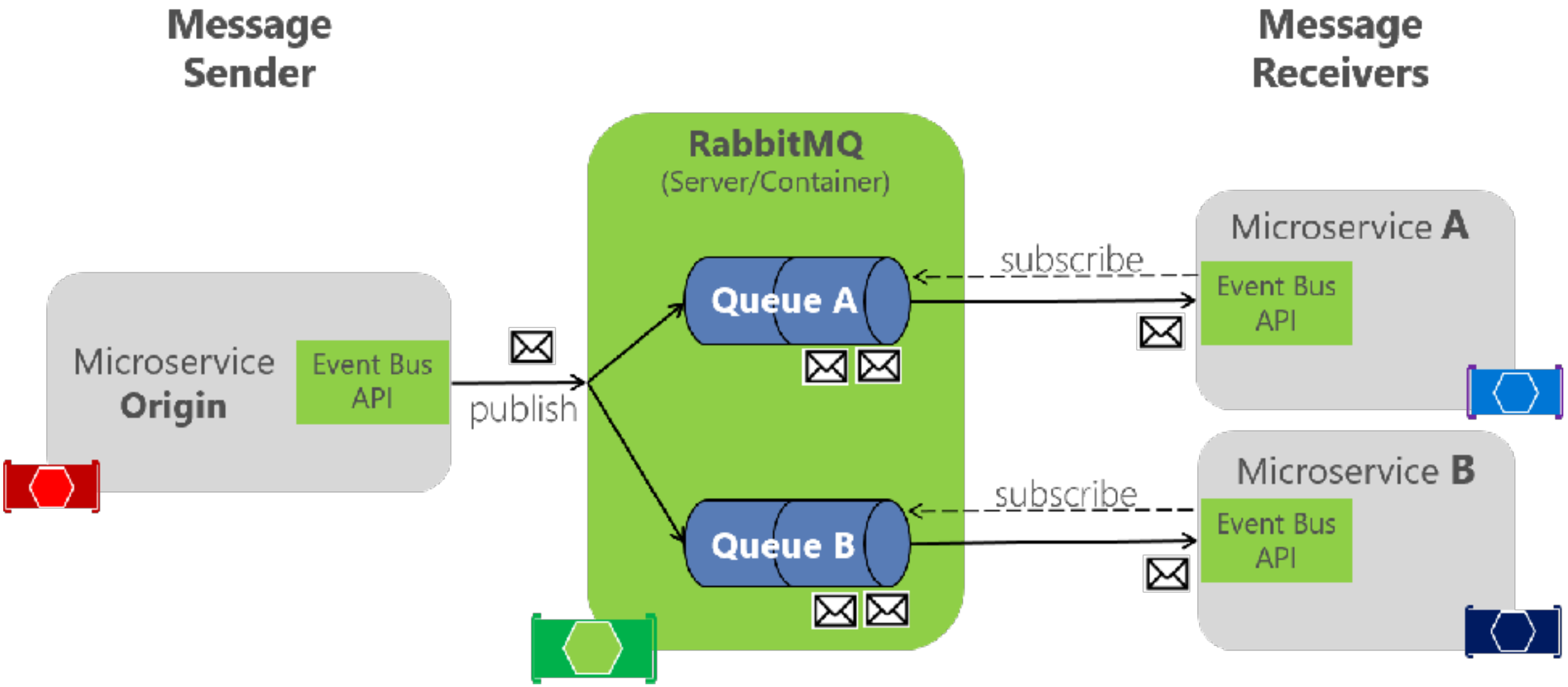
# Request-response model



# Request-response model

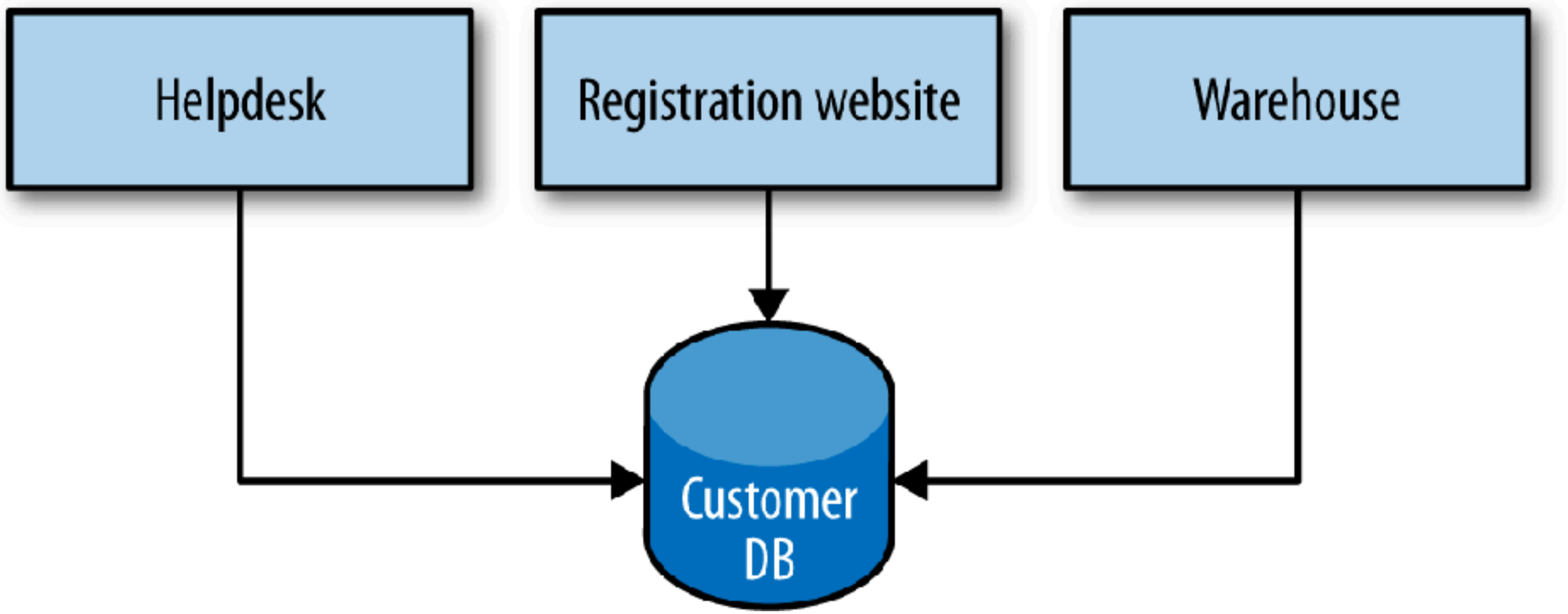


# Observer model

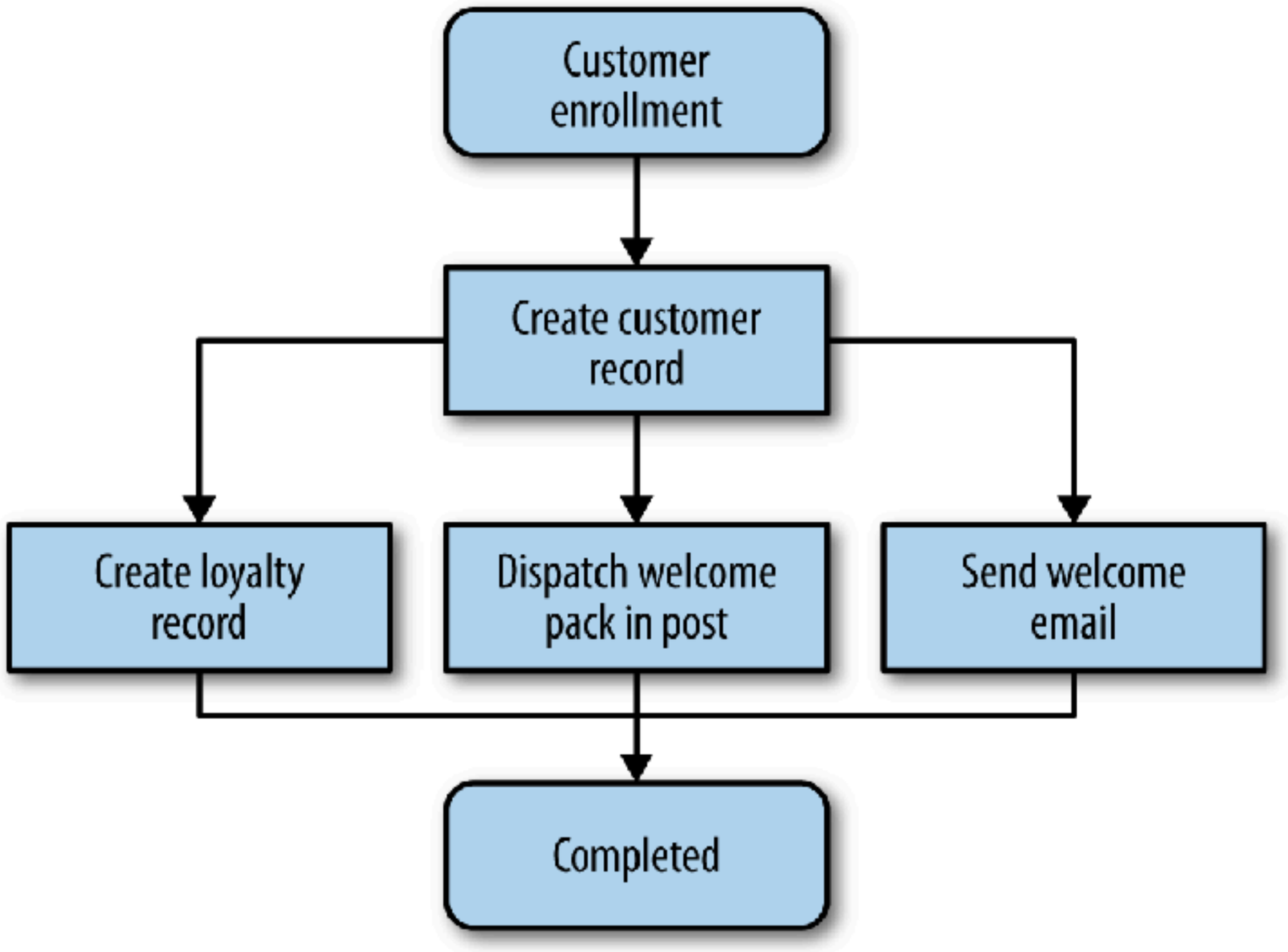


# Services Integration

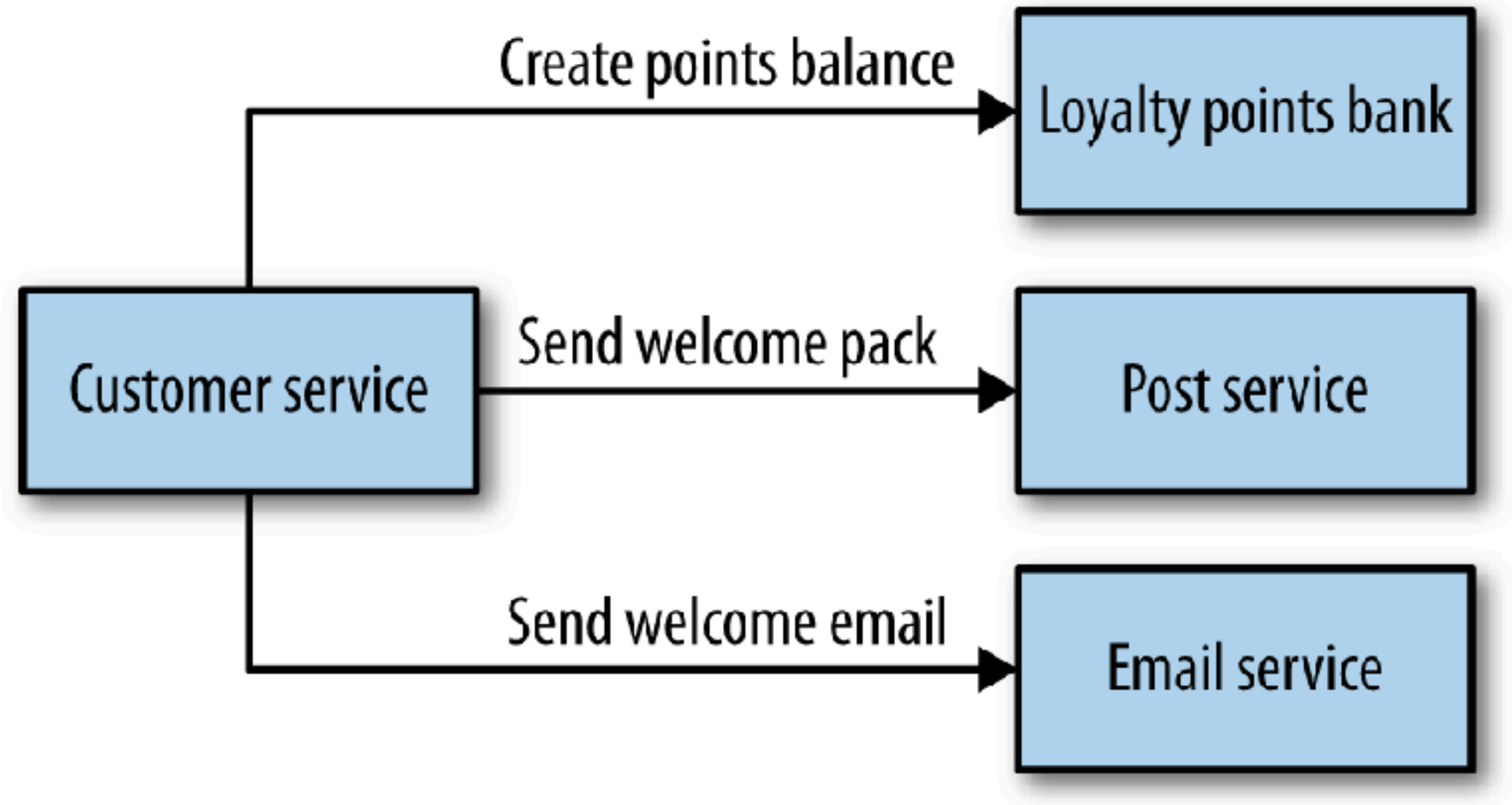
# Shared database



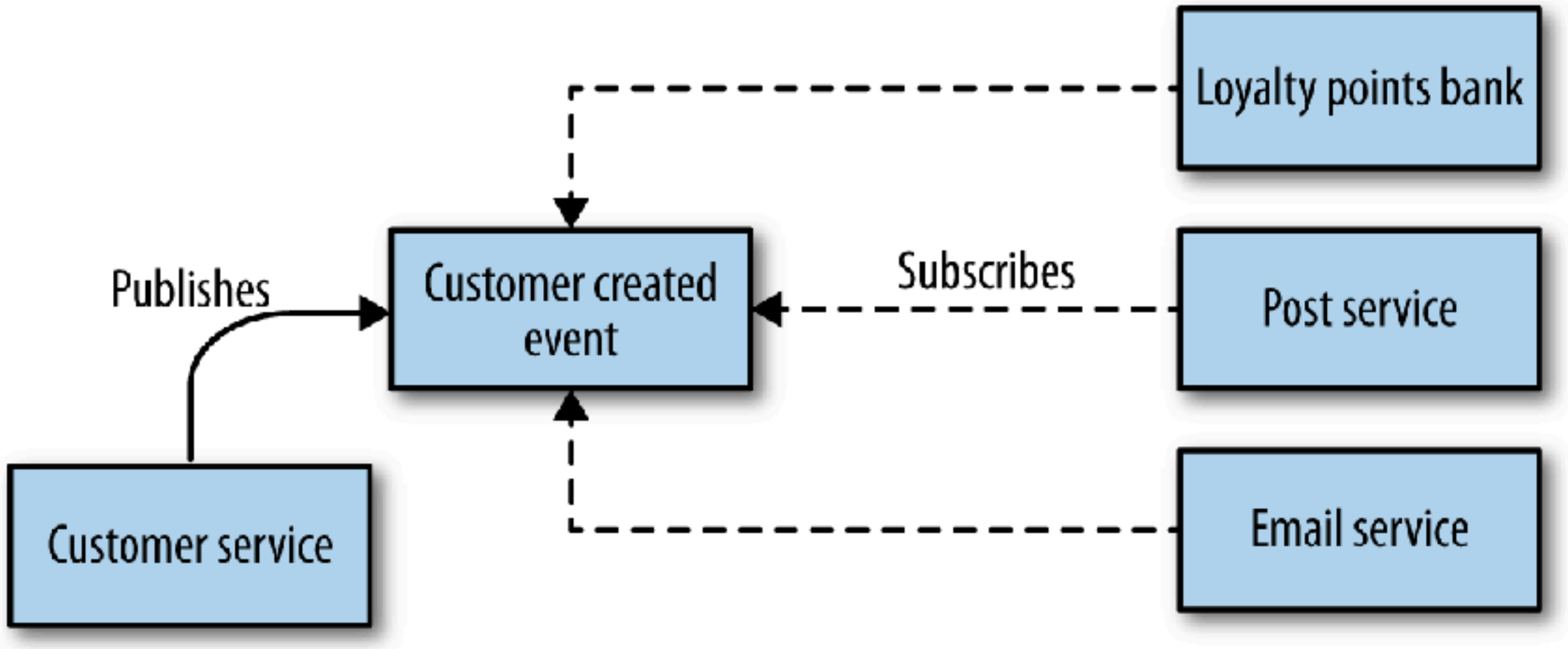
# Orchestration vs Choreography



# Orchestration

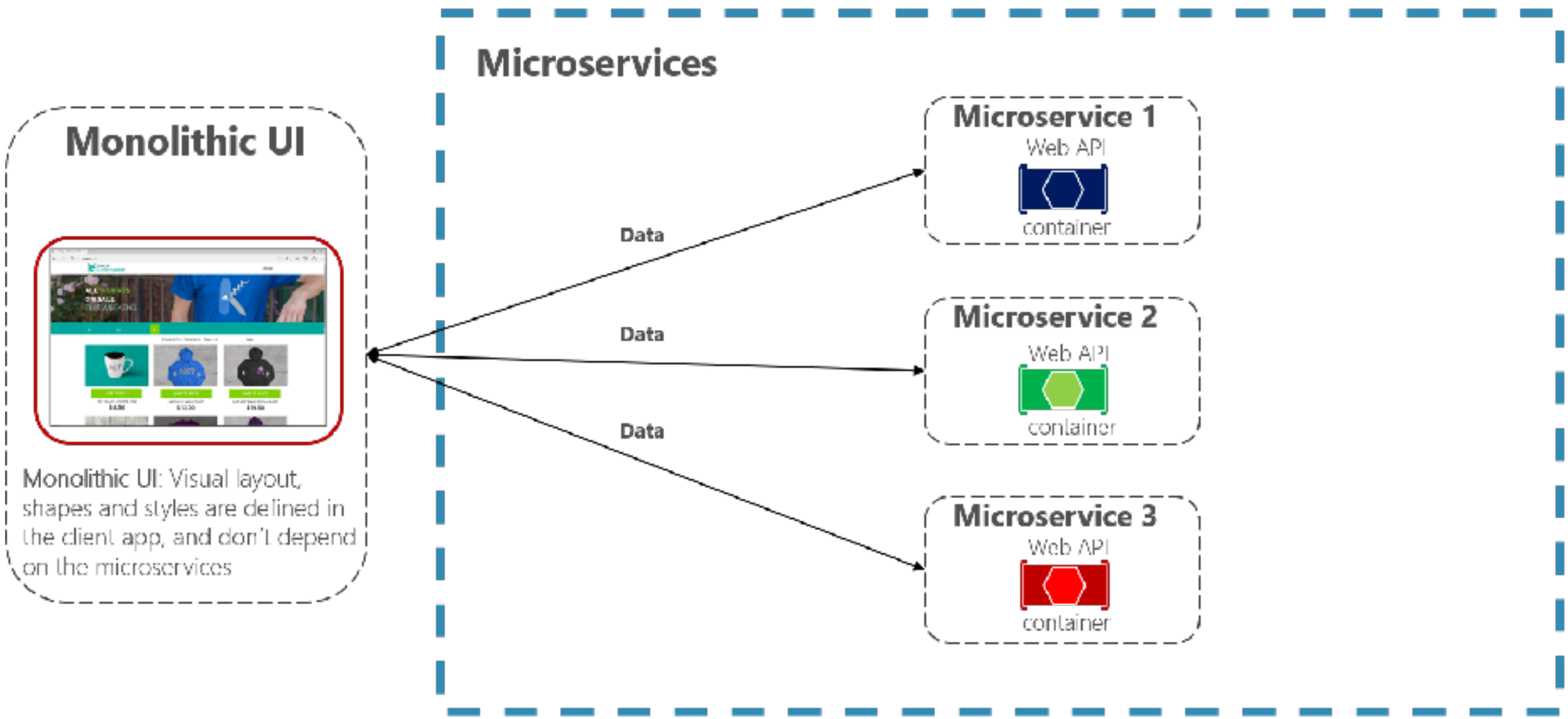


# Choreography

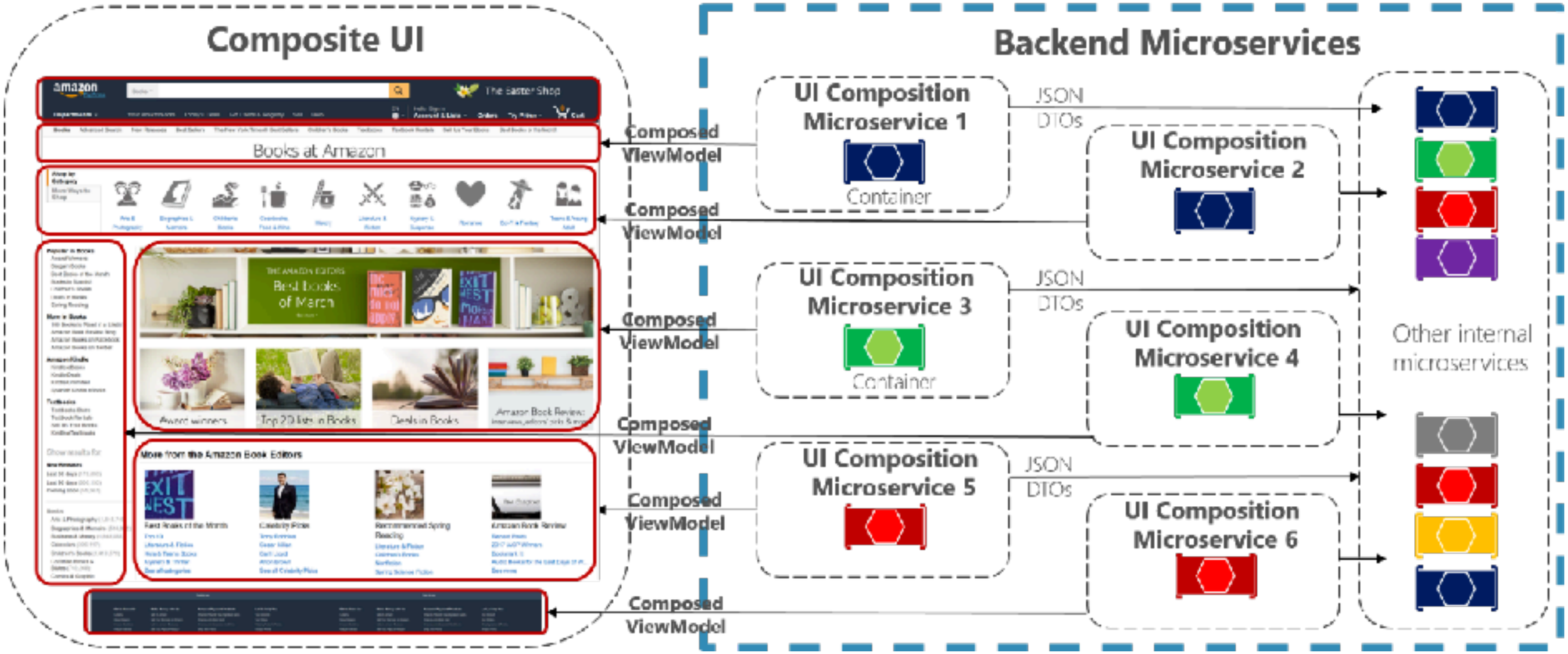


# Integrate with User Interface

# **Monolithic UI** consuming microservices



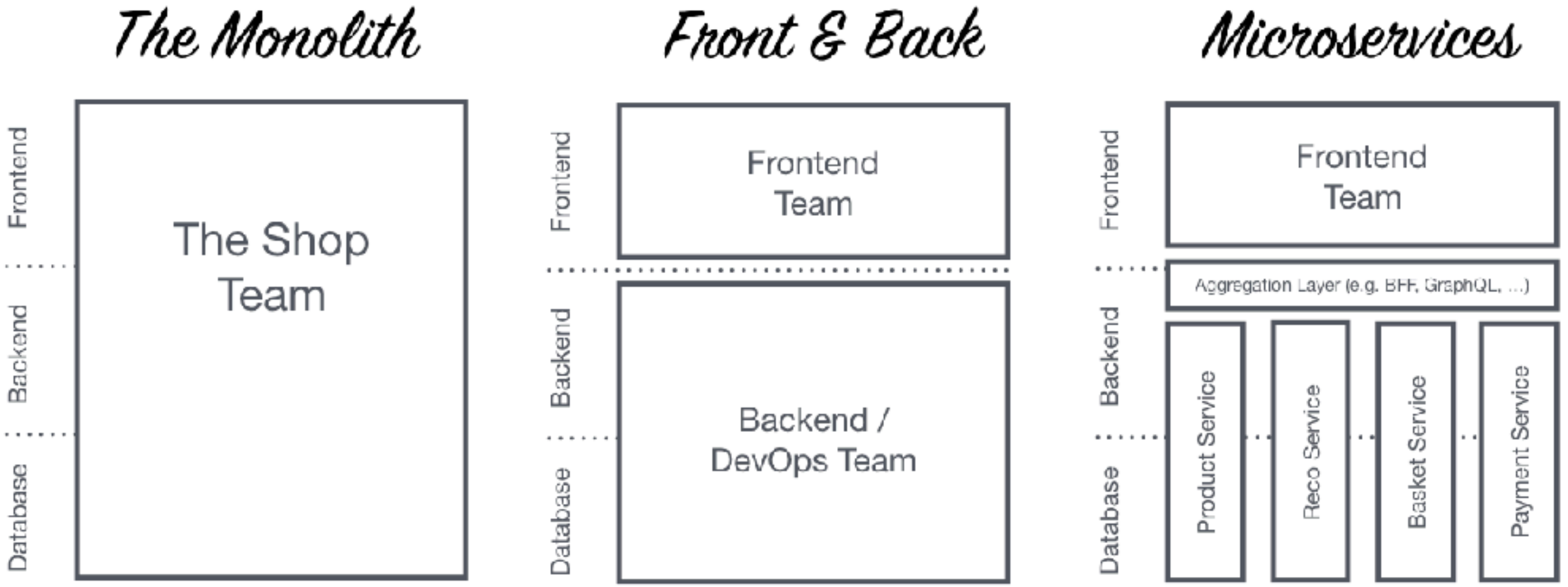
# **Composite UI** *generated* by microservices



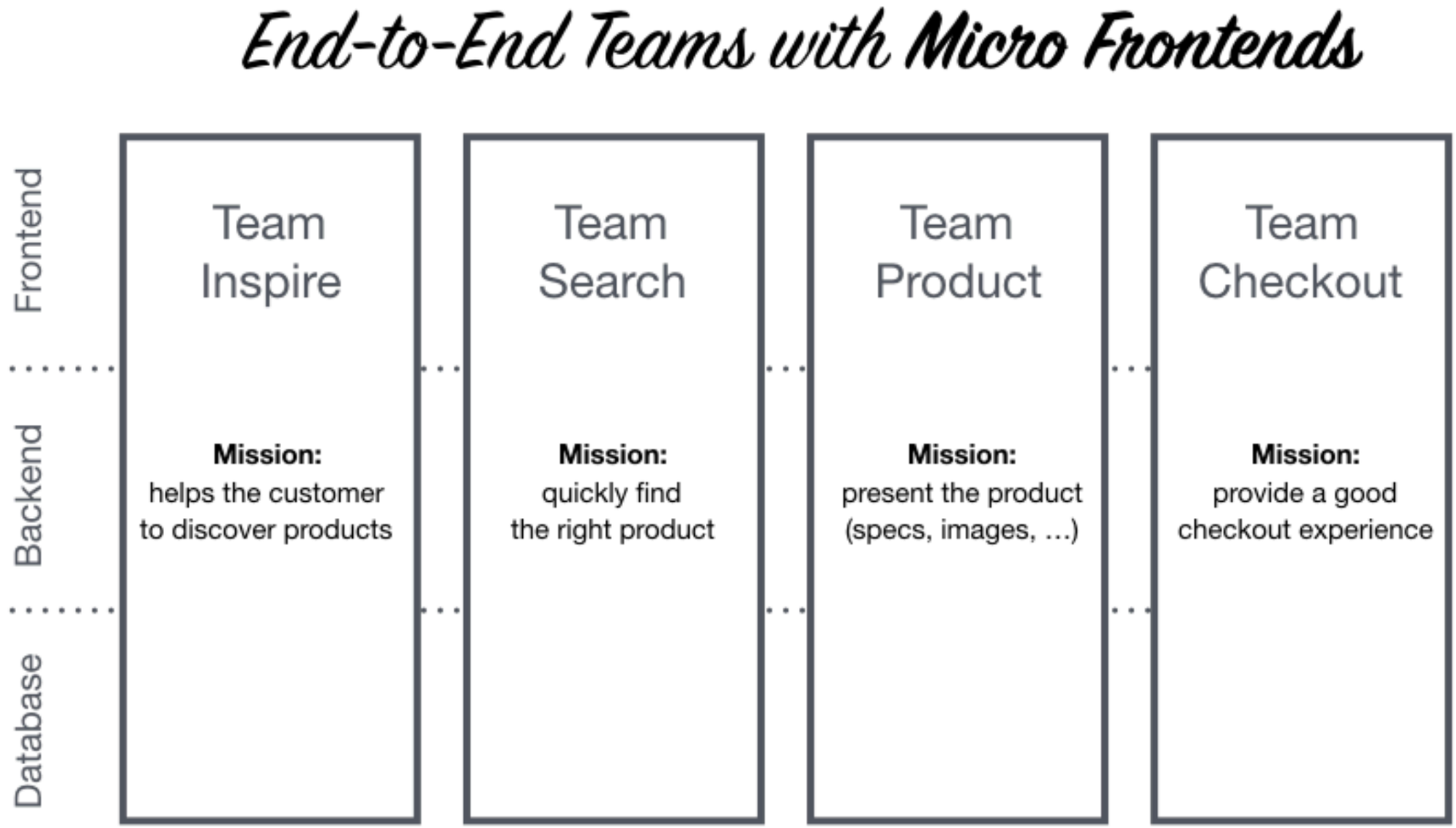
# Micro Frontend

https://micro-frontends.org/

# Monolith Frontend



# Micro Frontend



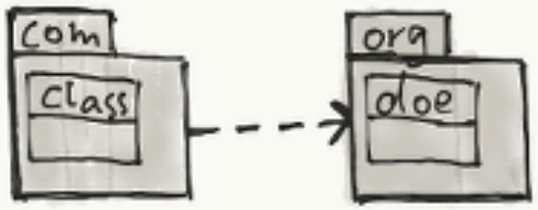
# Microservices pitfalls

# Microservices pitfalls

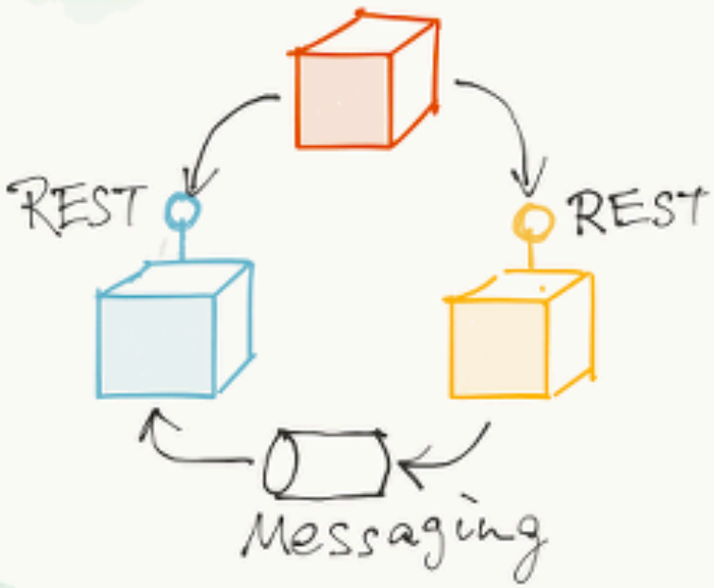
More/Low splitting
More network interaction
Data storing and sharing
Compatibility issues
Testing
Operation & Monitoring
Security
Performance

# Microservices in one page

# Architecture



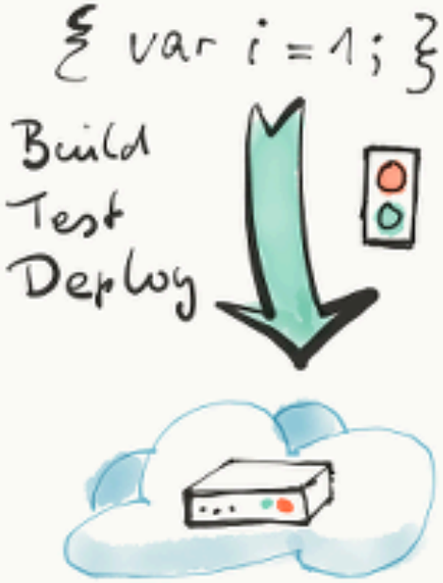

# Microservices




# People & Teams

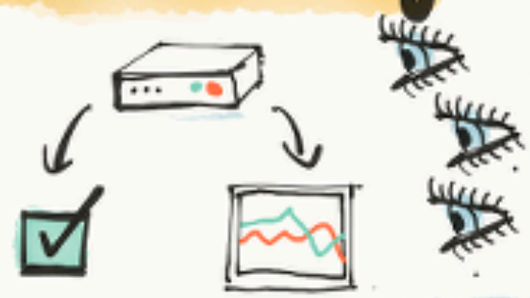

# Deployment

## Continous Delivery




# Infrastructure


# Monitoring



SEQUOIA

# MICROSERVICES ECOSYSTEM

DEVELOPER TOOLS

DATA CENTER

## SOURCE
Atlassian
GitHub
Gitlab

## CONTINUOUS INTEGRATION
Atlassian CloudBees
JFrog Codeship
CircleCI Werker
Shippable

## CONTAINER REGISTRY
Docker
Amazon
Google

## SERVICE DISCOVERY & PLANNING
Docker Kubernetes
Hashicorp

## SERVICE OPTIMIZATION
Force12.io

## SECURITY & COMPLIANCE
Illumio Apcera CloudPassage Banyan
Twistlock Redlock Palo Alto Networks StackRox
Conjur Scalock

## MONITORING | LOG ANALYSIS
Wavefront DataDog Sysdig
Nagios Gencore App Dynamics
Runscope New Relic SignalFX

Elastic Logentries
SumoLogic
Splunk

## INTER-SERVICE COMMUNICATIONS
Confluent Tensyr Rabbit (Pivotal)
Hystrix Thrift Finagle
NATS gRPC

## ORCHESTRATION
Docker Mesosphere Apcera
Kubernetes HashiCorp

## PLATFORM MANAGEMENT
Docker Mesosphere AppFormix
Nirmata Rancher Stack Engine (Oracle)
Apcera Flexiant Containership
ManageIQ Kubernetes

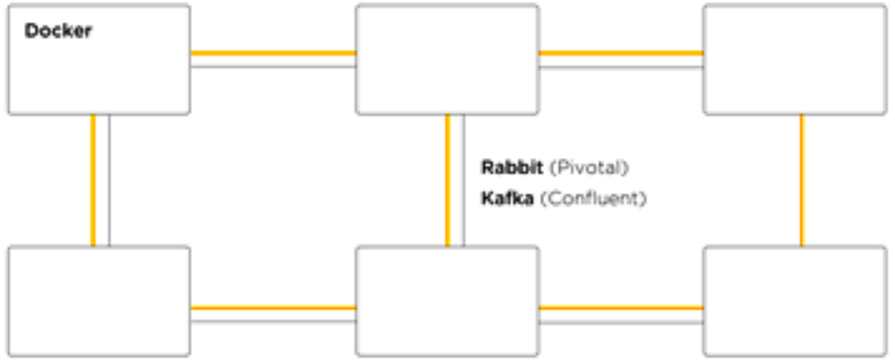
## API MANAGEMENT
Mulesoft Akana Apigee Runscope
Kong WSO2 3Scale Mashery

## REGISTRATION
Zookeeper CoreOS

## LOAD BALANCING
NGINX Datawire Buoyant HAProxy
Traefik

## MICROSERVICES

Get small to get big. Microservices is an approach to building software that shifts away from large monolithic applications towards small, loosely coupled and composable autonomous pieces.




## OPERATING SYSTEM
Linux UNIX Windows CoreOS OpenStack Mesosphere

## INFRASTRUCTURE AUTOMATION
HashiCorp Ansible (Red Hat)
Puppet SaltStack
Chef

## DATABASE & DATA MANAGEMENT
ClusterHQ Minio
MongoDB Crate.io
Cockroach

## NETWORK
Cumulus Docker
Big Switch Weaveworks
FBOSS Calico
OpenSwitch

## CONVERGED INFRASTRUCTURE
Ceph (Red Hat) Datawise
Springpath Portworx

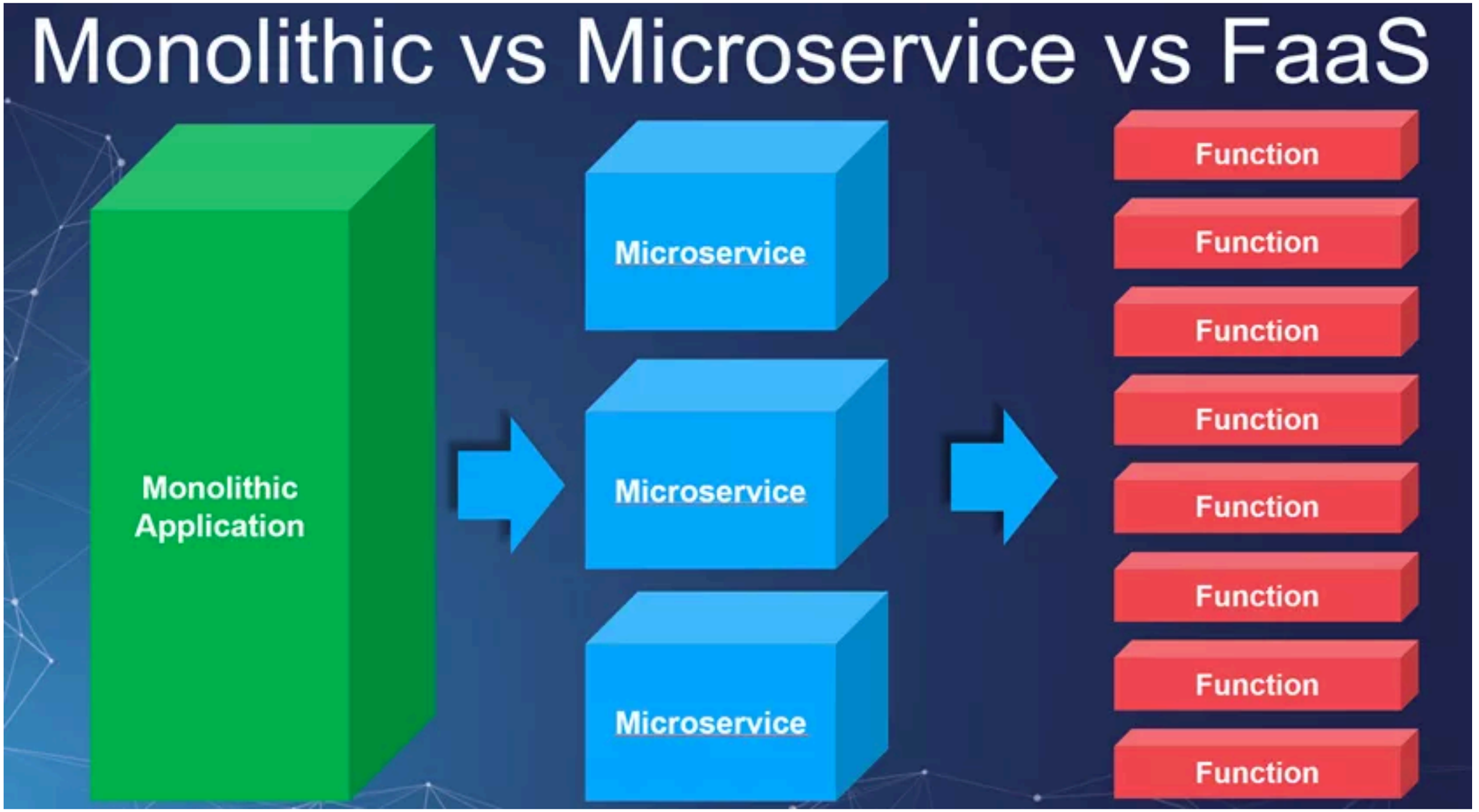
## PLATFORMS
OpenShift Joyent
Cloud Foundry Deis
Docker

## PUBLIC CLOUD
AWS DigitalOcean
Azure VMware
IBM Google

Includes both companies and open source projects

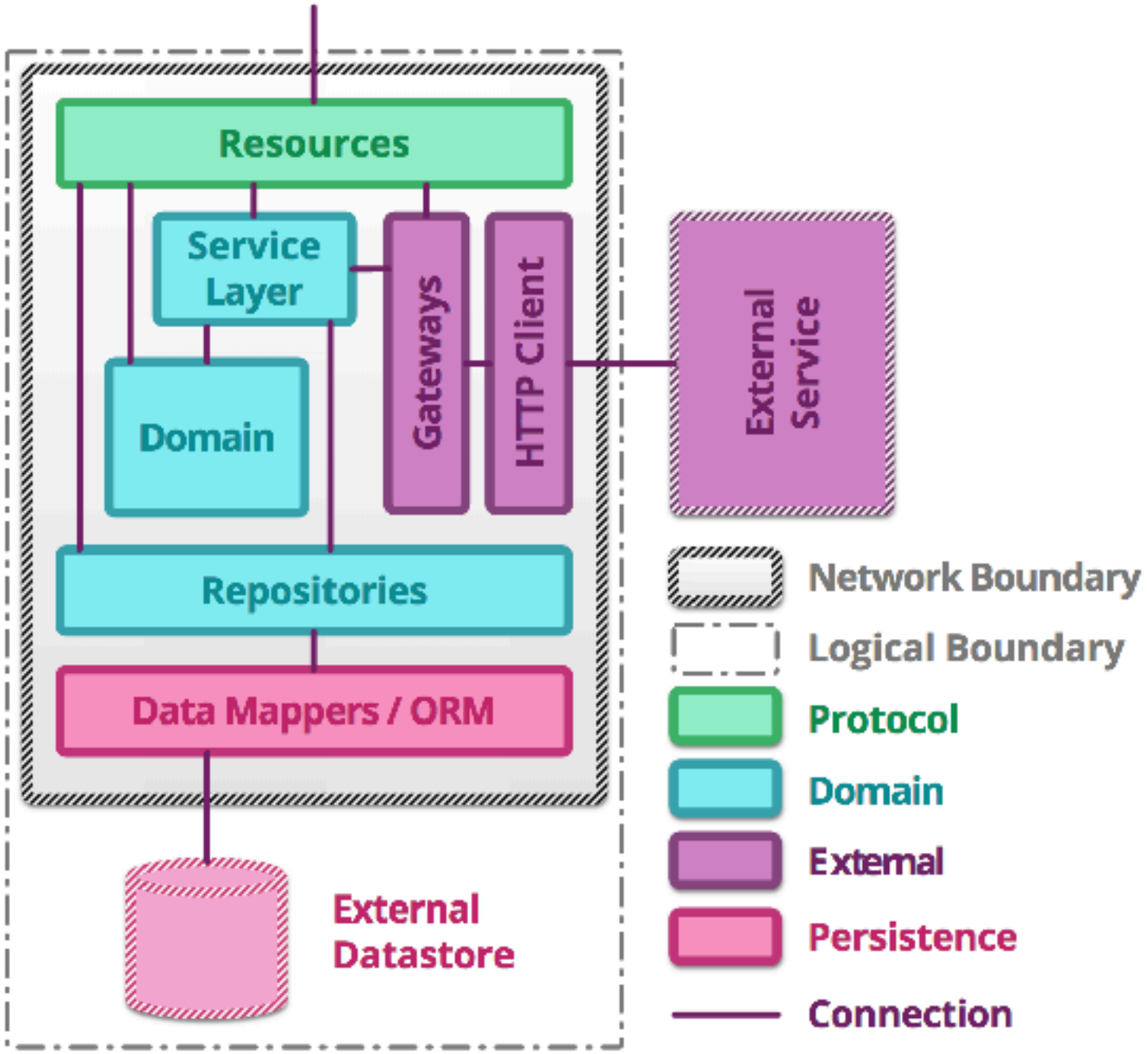
*Version 1.2 | 01.18.16*

# more ...



# How to Design ?

# Service structure

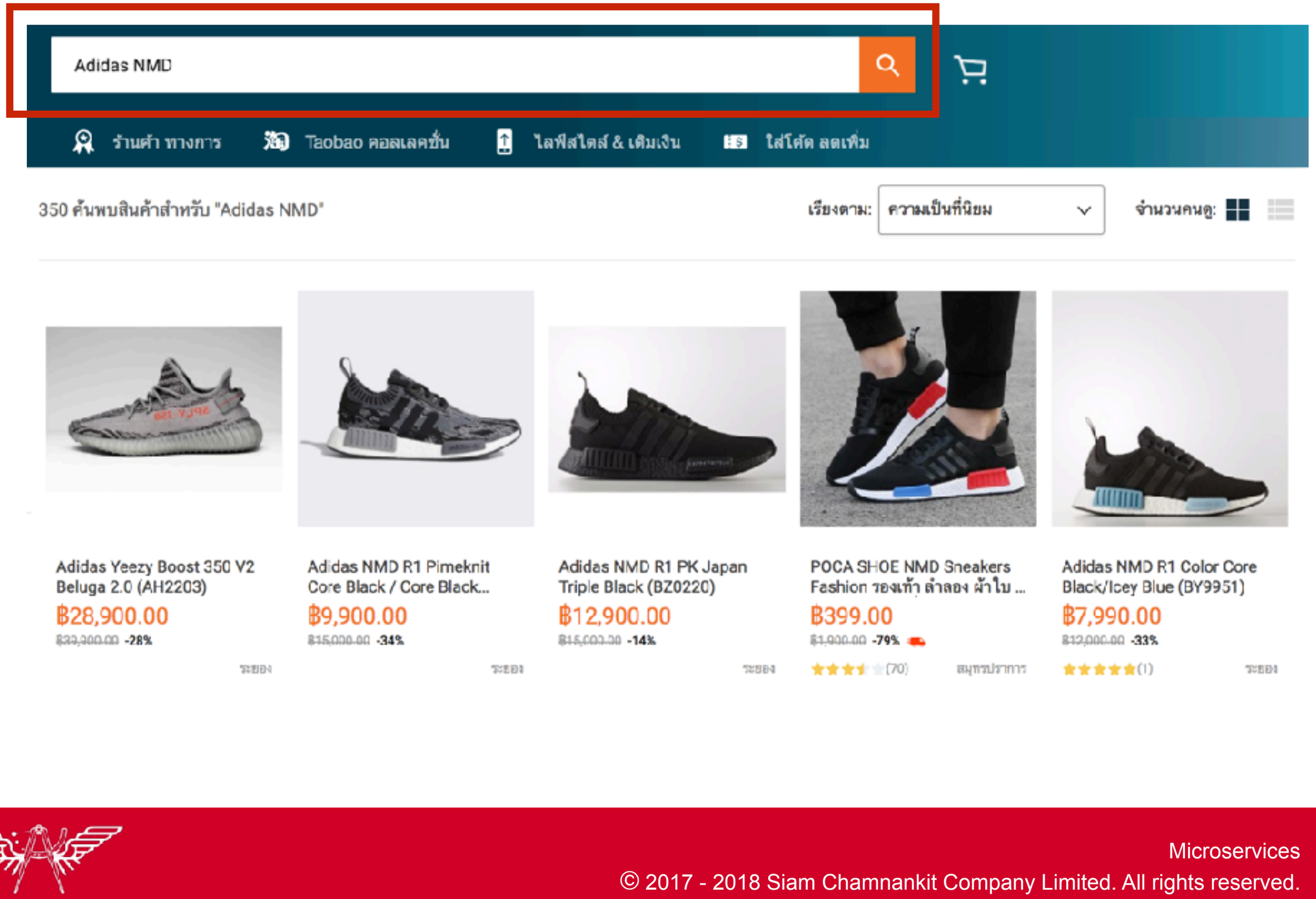



https://martinfowler.com/articles/microservice-testing

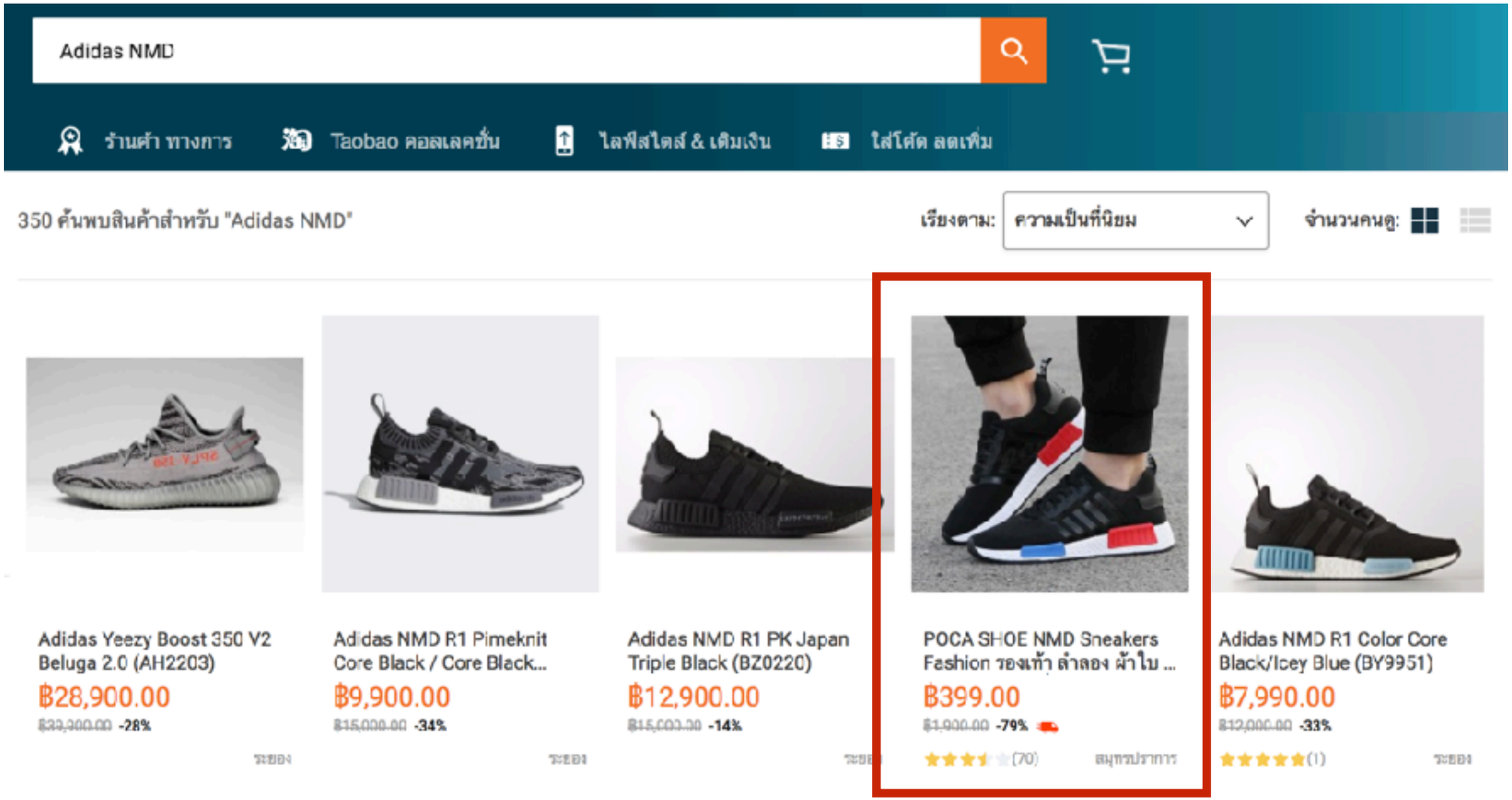
# Let’s workshop with Design

# E-commerce system

# 1. Search product by name



# 2. Choose a product

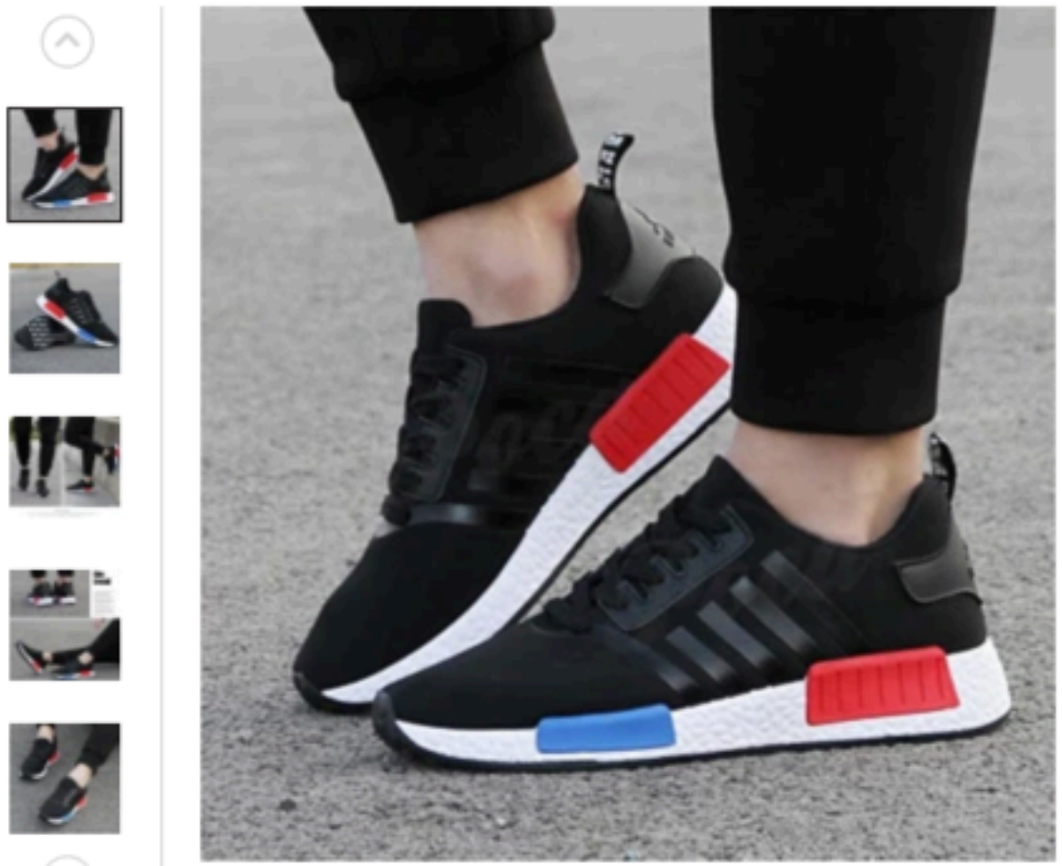


# 3. Show product detail

POCA SHOE NMD Sneakers Fashion รองเท้า ลำลอง ผ้าใบ ผู้หญิง-ผู้ชาย แฟชั่น ราคาถูกสวยๆ Sport Unisex รุ่น PSN-Black/White

(70) แสดงความคิดเห็น

ยี่ห้อ Poca Shoes | เพิ่มเติม สุภาพบุรุษ จาก Poca Shoes



**2 Weeks** Warranty by Seller เพิ่มเติม

- สวมใส่สบาย

เพิ่มเติม

เลือก ขนาด

ขนาด

ขนาด ไม่มี

399 บาท

ราคาปกติ ~~1,900 บาท,~~
ประหยัดทันที **79%**
ราคาโปรโมชั่นสามารถใช้ได้กับ 25/2/2018

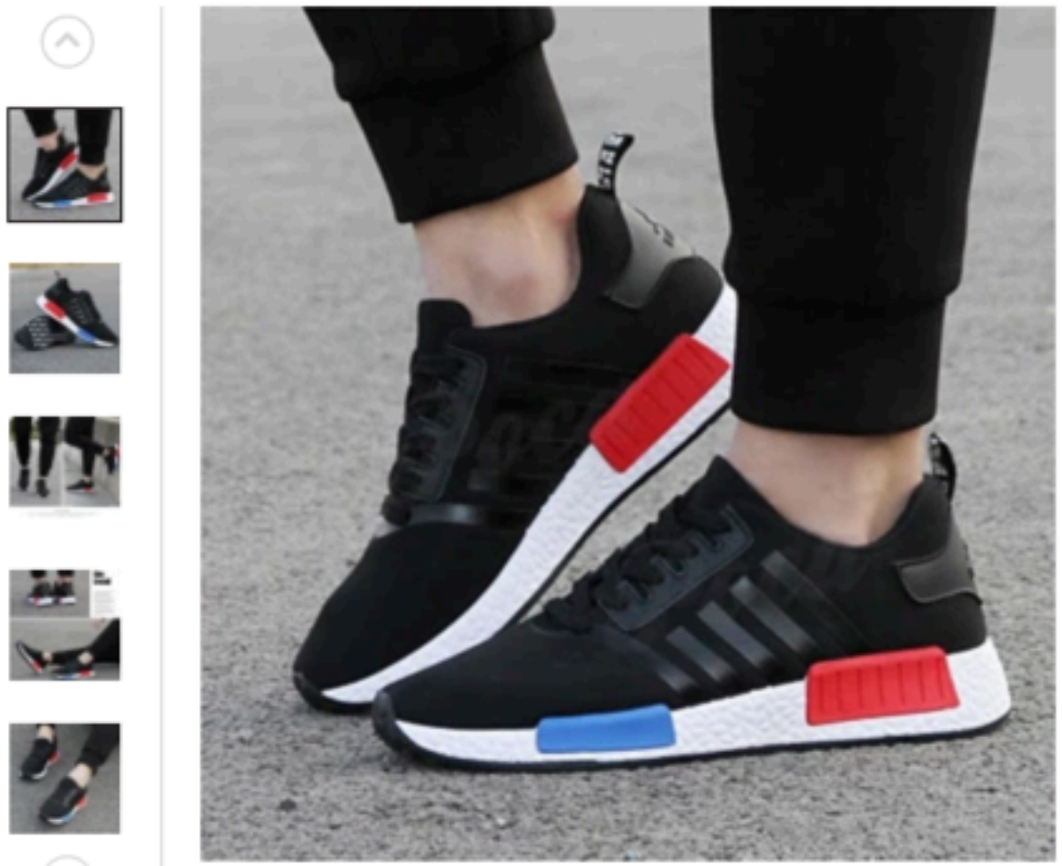
ใส่ตะกร้า วิธีการสั่งซื้อ

# 4. Add product to basket

POCA SHOE NMD Sneakers Fashion รองเท้า ลำลอง ผ้าใบ ผู้หญิง-ผู้ชาย แฟชั่น ราคาถูกสวยๆ Sport Unisex รุ่น PSN-Black/White

(70) แสดงความคิดเห็น

ยี่ห้อ Poca Shoes | เพิ่มเติม สุภาพบุรุษ จาก Poca Shoes



**2 Weeks** Warranty by Seller เพิ่มเติม

- สวมใส่สบาย

เพิ่มเติม

เลือก ขนาด

ขนาด

ขนาด ไม่มี

399 บาท

ราคาปกติ ~~1,900 บาท,~~
ประหยัดทันที **79%**
ราคาโปรโมชั่นสามารถใช้ได้กับ 25/2/2018

ใส่ตะกร้า

วิธีการสั่งซื้อ

# 5. Show data in basket

✓ สินค้า 1 ชิ้น ได้ถูกเพิ่มเข้าไปยังตะกร้าสินค้าของคุณ

POCA SHOE NMD Sneakers Fashion รองเท้า ลำลอง ผ้าใบ ผู้หญิง-ผู้ชาย แฟชั่น ราคาถูกสวยๆ Sport Unisex รุ่น PSN-Black/White

ไซส์: EU:40

Poca Shoes

**399 บาท**

~~1,900 บาท~~ 79% ปิด

**ตะกร้าสินค้าของคุณ** (1 สินค้า)

มูลค่าสินค้า: 399 บาท

**ยอดสุทธิ** ราคารวมภาษีมูลค่าเพิ่ม (ถ้ามี): **399 บาท**

เลือกซื้อสินค้าต่อ | ชำระค่าสินค้า

**People Who Bought This Item Also Bought**

กางเกงสแลคขายาว Hopper Progress ผ้ายืด ทรงเข้ารูป

~~900 บาท~~

67% ปิด

299 บาท

# 6. Checkout

✓ สินค้า 1 ชิ้น ได้ถูกเพิ่มเข้าไปยังตะกร้าสินค้าของคุณ

POCA SHOE NMD Sneakers Fashion รองเท้า ลำลอง ผ้าใบ ผู้หญิง-ผู้ชาย แฟชั่น ราคาถูกสวยๆ Sport Unisex รุ่น PSN-Black/White

ไซส์: EU:40

Poca Shoes

**399 บาท**

~~1,900 บาท~~ 79% ปิด

**ตะกร้าสินค้าของคุณ** (1 สินค้า)

มูลค่าสินค้า: 399 บาท

**ยอดสุทธิ** ราคารวมภาษีมูลค่าเพิ่ม (ถ้ามี): **399 บาท**

เลือกซื้อสินค้าต่อ

ชำระค่าสินค้า

**People Who Bought This Item Also Bought**

กางเกงสแลคขายาว Hopper Progress ผ้ายืด ทรงเข้ารูป

~~900 บาท~~

67% ปิด

299 บาท

# 7. Shipping

LAZADA .CO.TH

1. ค่าซื้อขั้นต่ำ | 2. ชำระเงิน

**ที่อยู่ที่จะจัดส่ง** Login for speedy checkout

อีเมล: กรุณาใส่ อีเมล์ ของท่าน

ชื่อ และ นามสกุล: ชื่อและนามสกุล

ที่อยู่: ที่อยู่

รหัสไปรษณีย์: รหัสไปรษณีย์ ทางเราจะทำการตรวจสอบเมืองและจังหวัดของคุณ

เขต: เขต

จังหวัด: กรุงเทพมหานคร/ Bangkok

โทรศัพท์มือถือ: +66 เบอร์โทรติดต่อ เพื่อให้มั่นใจว่าทำการจัดส่งได้

☐ ที่อยู่สำหรับใบเสร็จ/ใบกำกับภาษี - กรุณาเลือกและกรอกข้อมูลเพื่อทำการขอใบกำกับภาษี

**ข้อมูลการส่งสินค้า**

ส่งแบบธรรมดา: ฟรี

Get it วันอังคาร, 27 ก.พ. - วันจันทร์, 5 มี.ค. 2018

ดำเนินการต่อ

**สรุปการสั่งซื้อ** (1 items)

| สินค้า | จำนวน | ราคา |
|---|---|---|
| POCA SHOE NMD Sneakers Fashion รองเท้า ลำลอง ผ้าใบ ผู้หญิง-ผู้ชาย แฟชั่น ราคาถูกสวยๆ Sport Unisex รุ่น PSN-Black/White ขนาด: EU:40 | 1 | 399 |

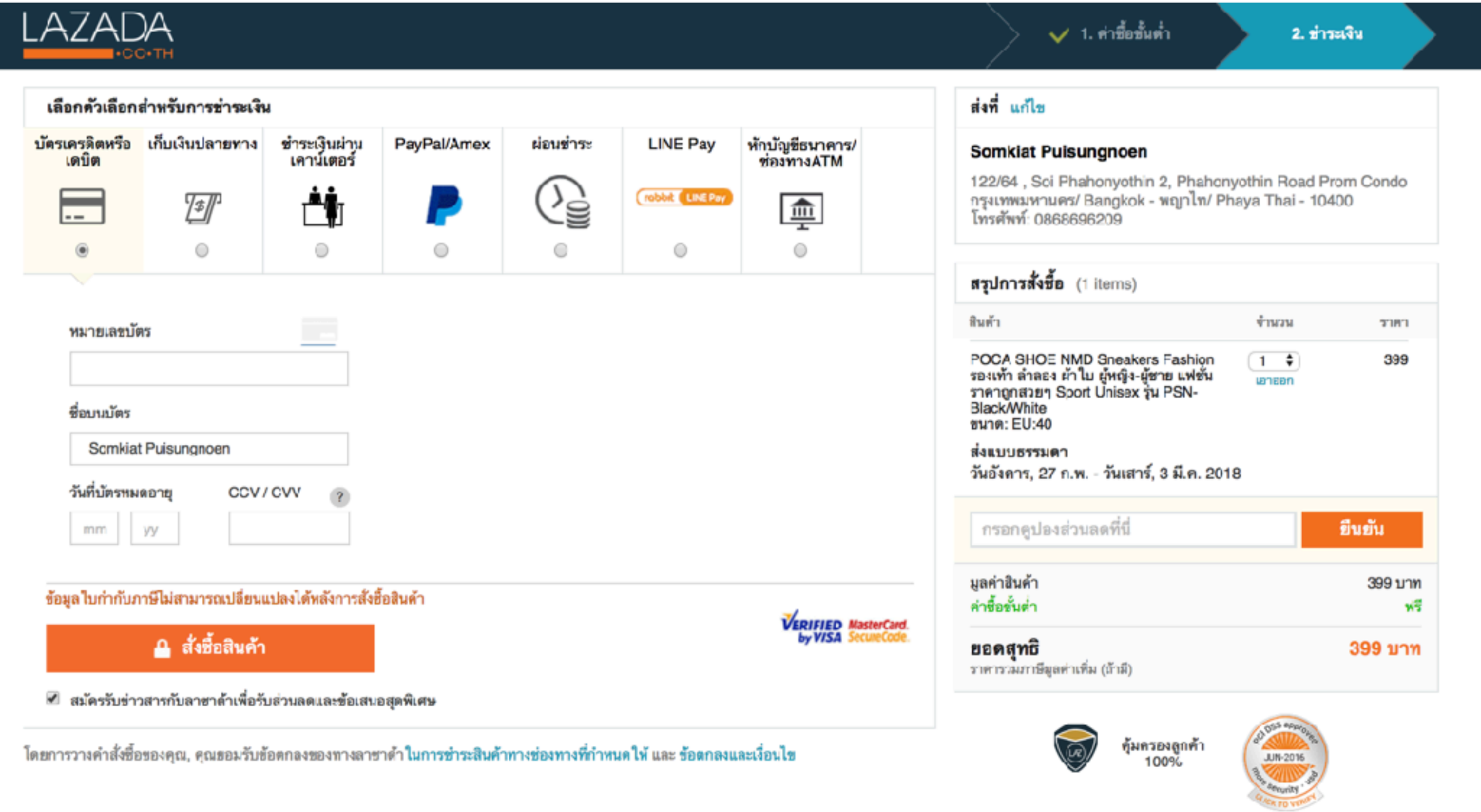
มูลค่าสินค้า 399 บาท

**ยอดสุทธิ** 399 บาท

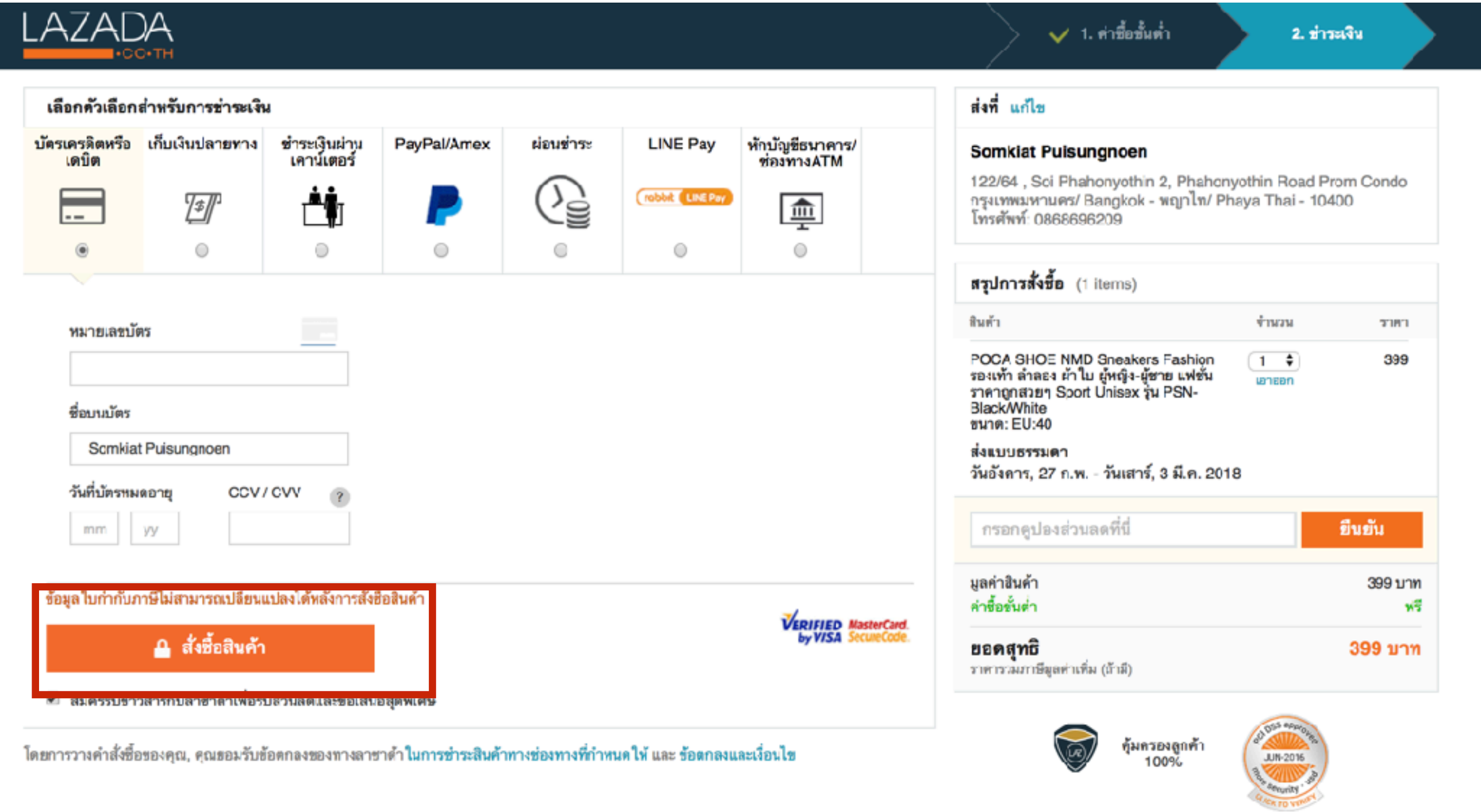
ราคารวมภาษีมูลค่าเพิ่ม (ถ้ามี)

คุ้มครองลูกค้า 100%

# 8. Payment



# 9. Confirm to order



# 10. Summary

LAZADA .CO.TH

Pay@All by counterservice

**ใบแจ้งการชำระเงิน(PaySlip)**

Counter Service Co., Ltd.

| | |
|---|---|
| **เลขที่ใบแจ้งสินค้า/Invoice No:** | **3779254692** |
| **ผู้ชำระเงิน/Payer:** | Somkiat Puisungnoen |
| **วันที่ทำรายการ / Transaction Date:** | 25/02/2018 23:33 |
| **กำหนดชำระเงิน / Expired Date:** | **27/02/2018 23:33** |
| **เพื่อเข้าบัญชี / Payee:** | www.lazada.co.th<br>Tel: **020180000** |
| **รายละเอียด / Detail:** | Lazada |

806010855864737

**จำนวนเงินที่ชำระ / Amount:**

**399**.00 บาท /THB

* ไม่รวมค่าธรรมเนียมของเคาน์เตอร์เซอร์วิส
(Excluding service fees at Counter Service)

คลิกปุ่ม **"Print"** พิมพ์ใบแจ้งการชำระเงิน
**หรือ**
จด **"รหัส 15 หลักใต้บาร์โค้ด"** เพื่อนำไปชำระเงินที่
Press "Print" button or write down paycode 15 digits for pay in cash at counter service(7-11)

Back to merchant | Print

# Try to design system

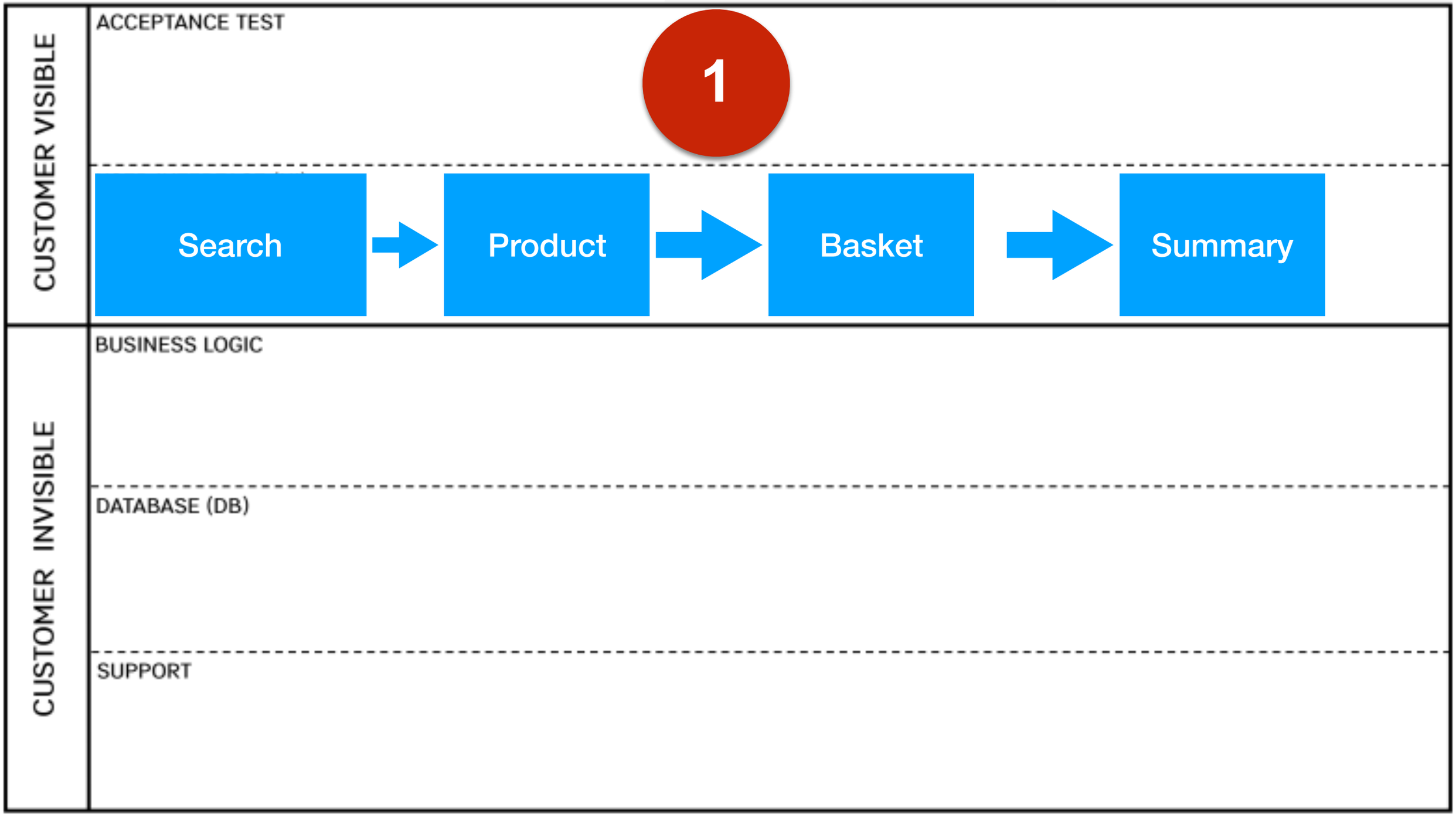
# A-DAPT Blueprint

| THEME: | EPIC: | FEATURE: | STORY: |
|---|---|---|---|
| DESIGNED BY: | DATE: | NOTE: | |

| | |
|---|---|
| CUSTOMER VISIBLE | ACCEPTANCE TEST |
| | USER INTERFACE (UI) |
| CUSTOMER INVISIBLE | BUSINESS LOGIC |
| | DATABASE (DB) |
| | SUPPORT |

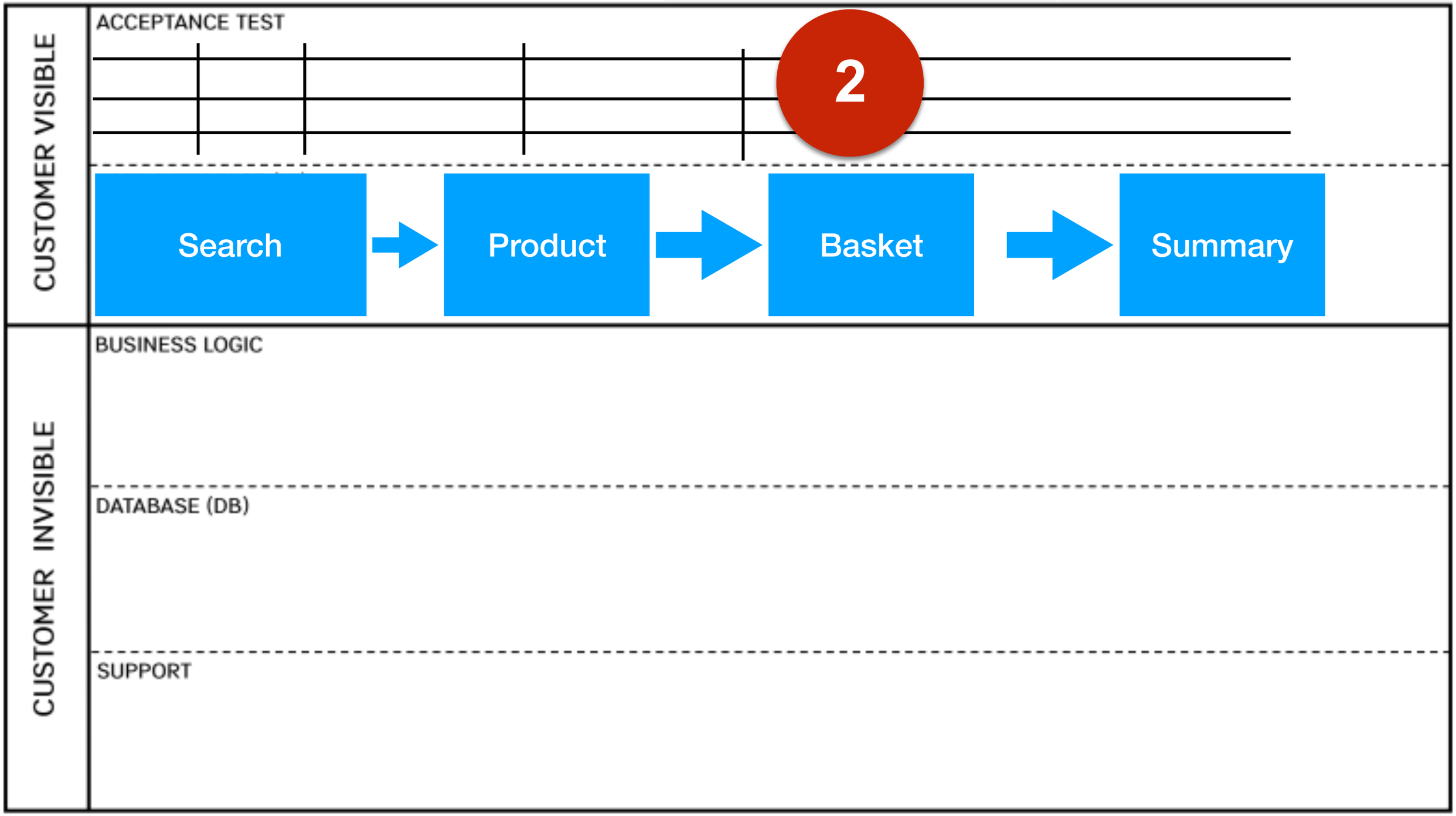
# A-DAPT Blueprint

| THEME: | EPIC: | FEATURE: | STORY: |
|---|---|---|---|
| DESIGNED BY: | DATE: | NOTE: | |



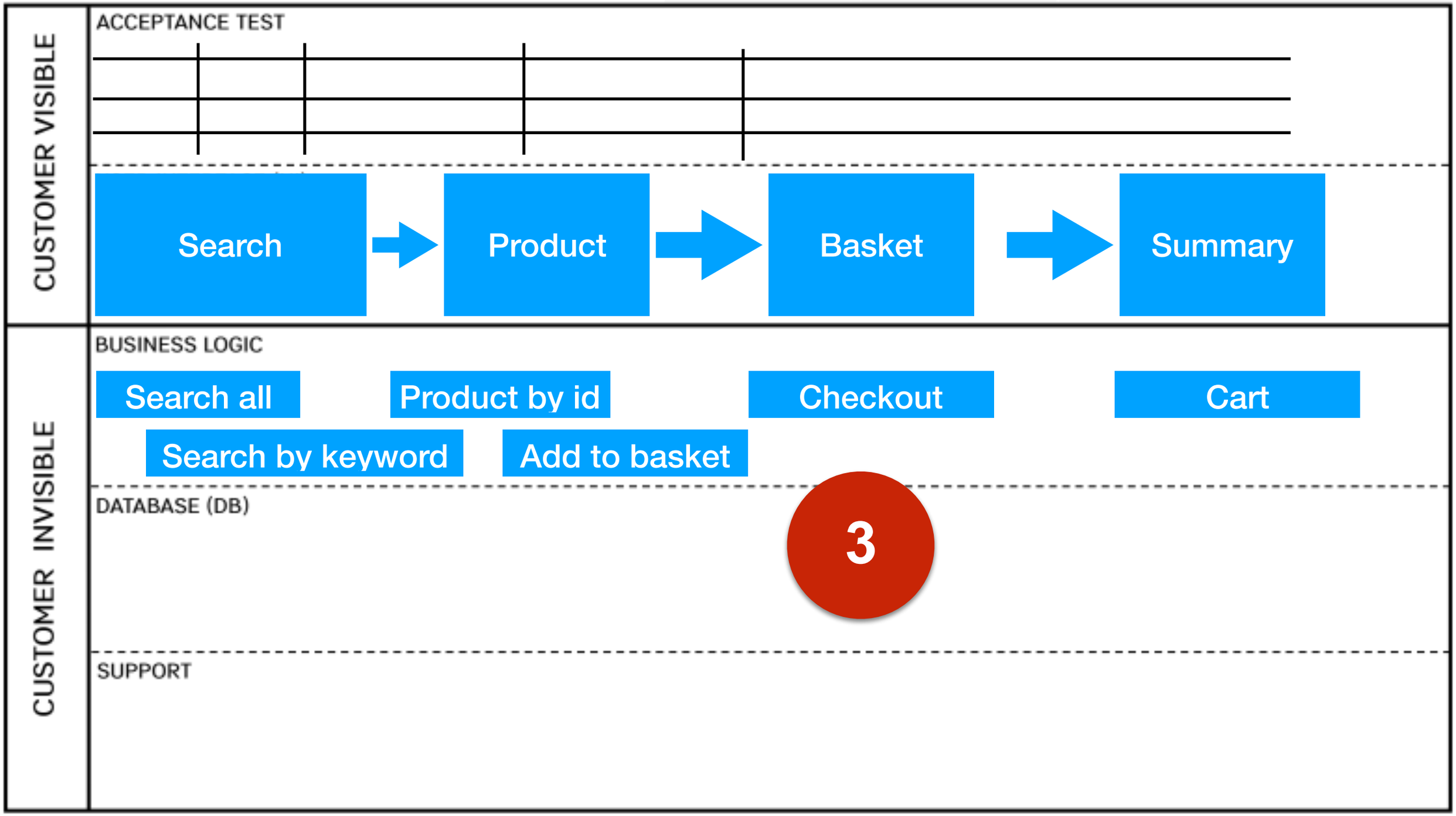
# A-DAPT Blueprint

| THEME: | EPIC: | FEATURE: | STORY: |
|---|---|---|---|
| DESIGNED BY: | DATE: | NOTE: | |



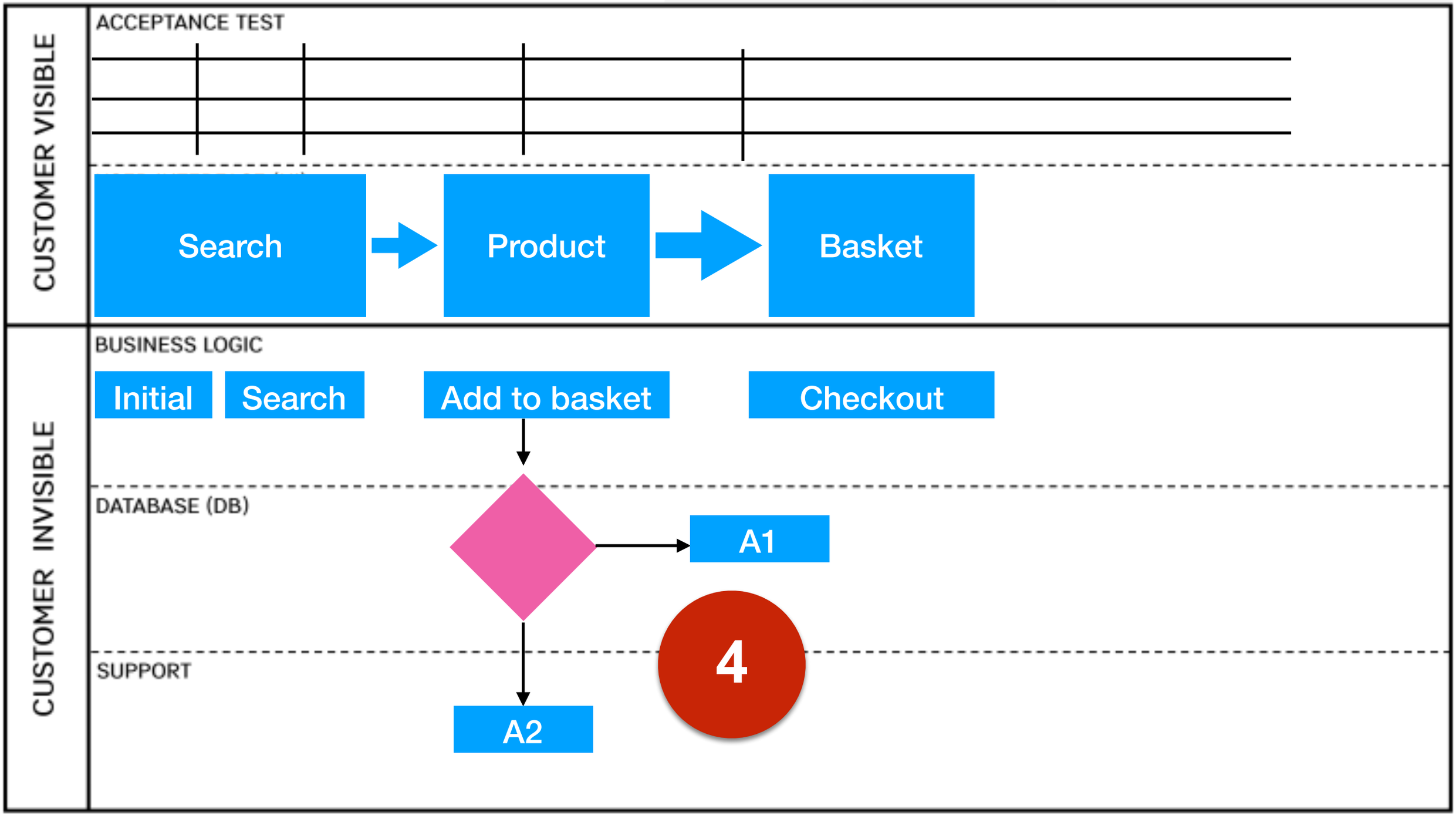
# A-DAPT Blueprint

| THEME: | EPIC: | FEATURE: | STORY: |
|---|---|---|---|
| DESIGNED BY: | DATE: | NOTE: | |



# A-DAPT Blueprint

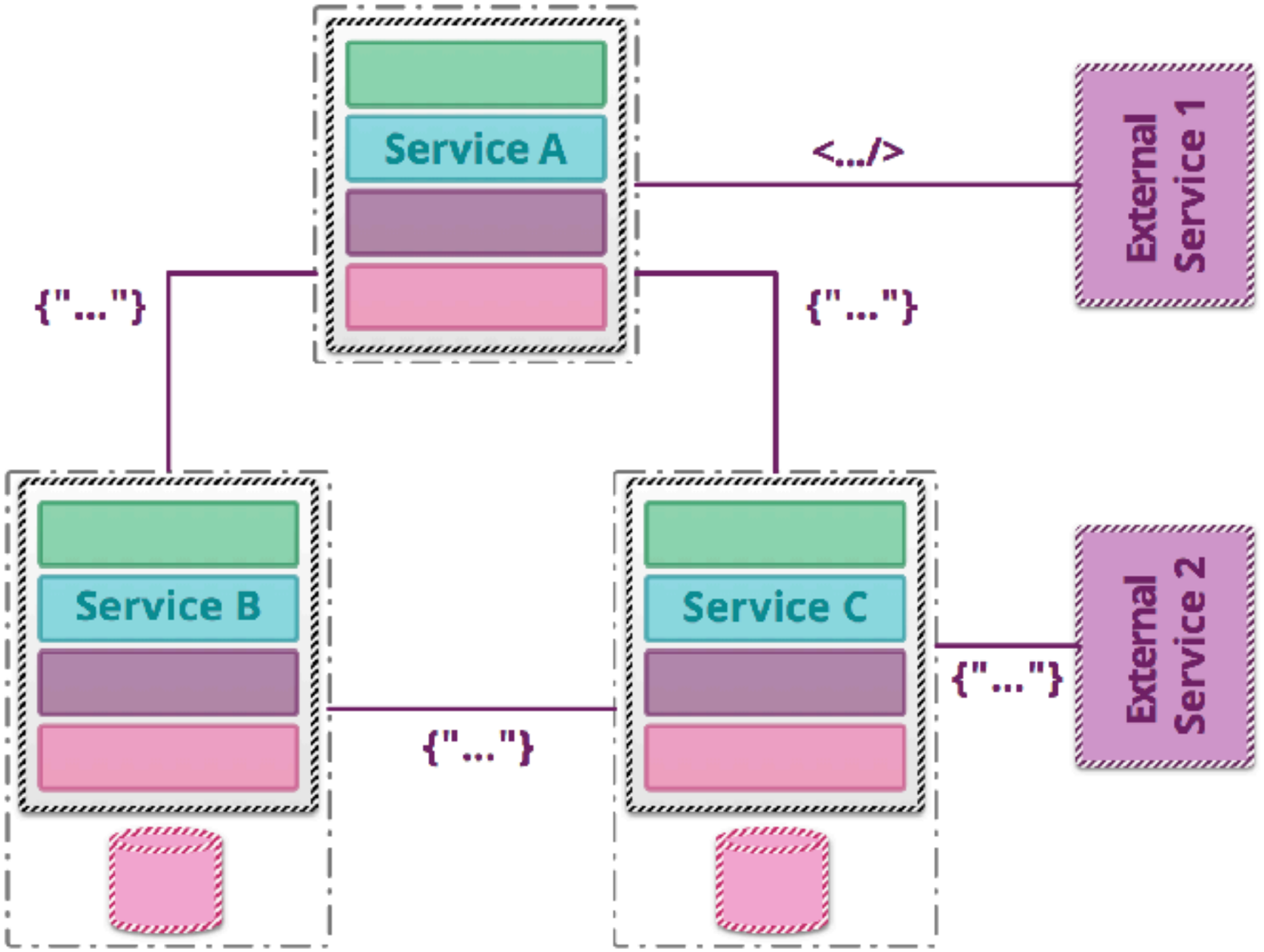
| THEME: | EPIC: | FEATURE: | STORY: |
|---|---|---|---|
| DESIGNED BY: | DATE: | NOTE: | |



# Retrospective

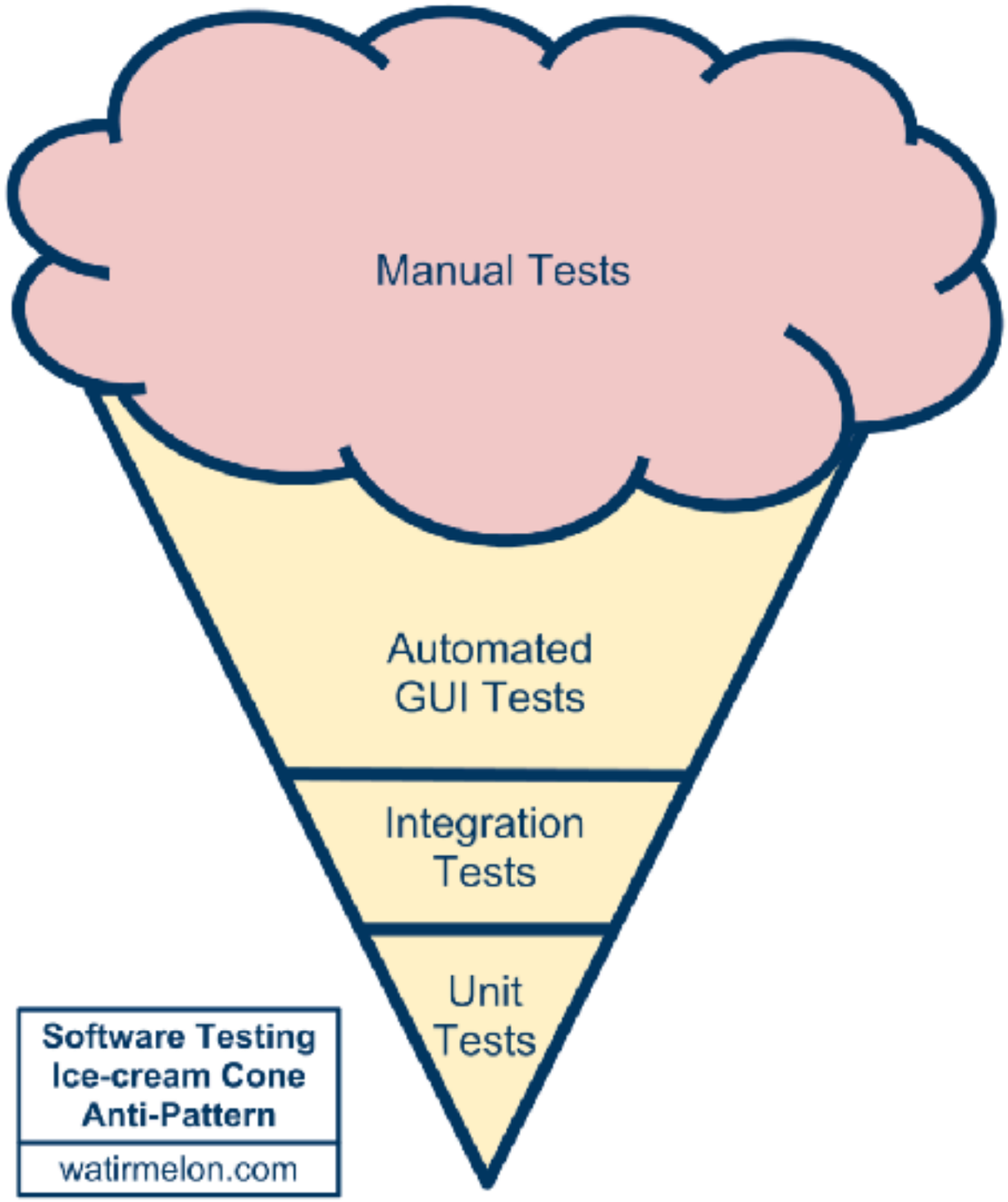
# How to Test ?

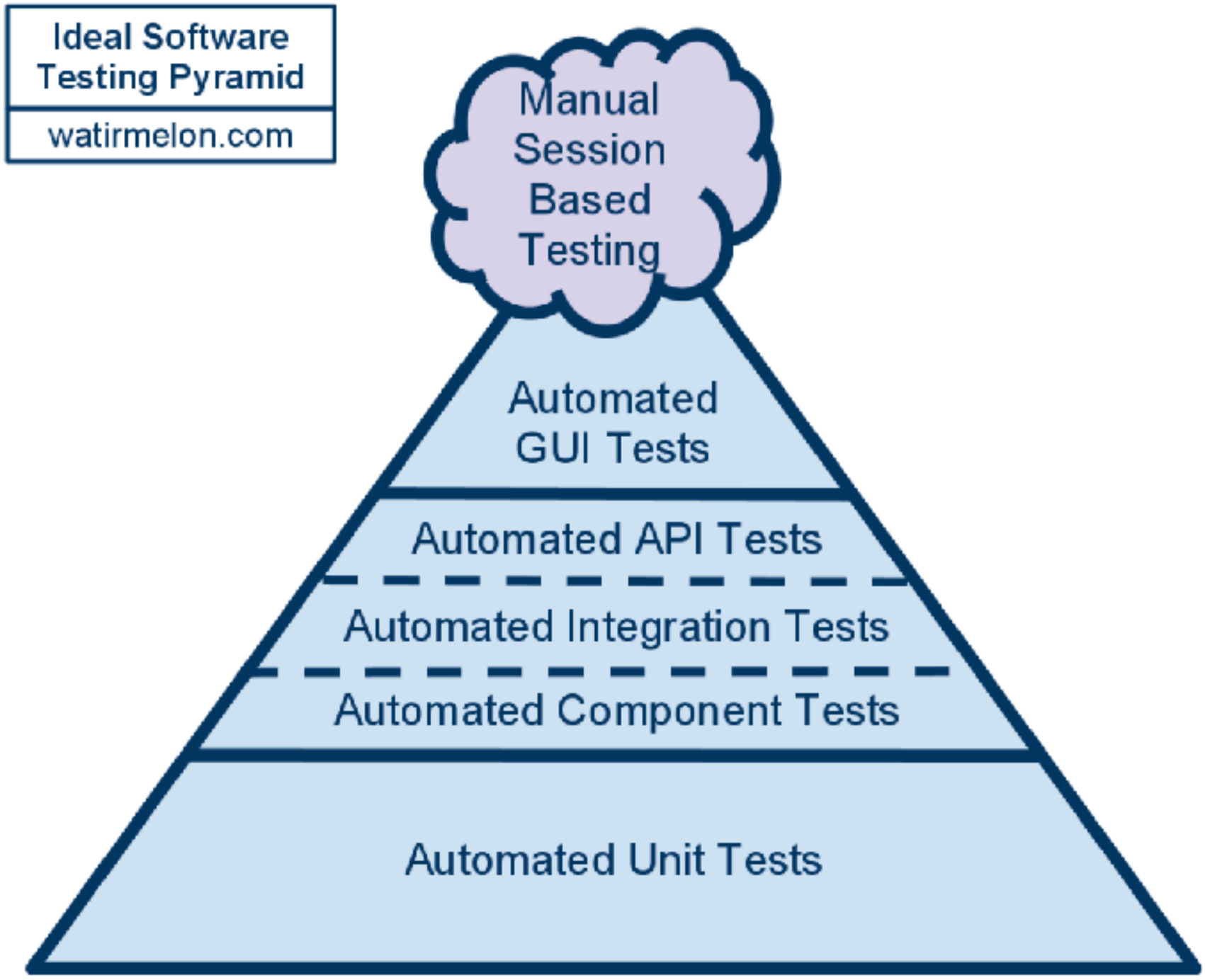
# Multiple services

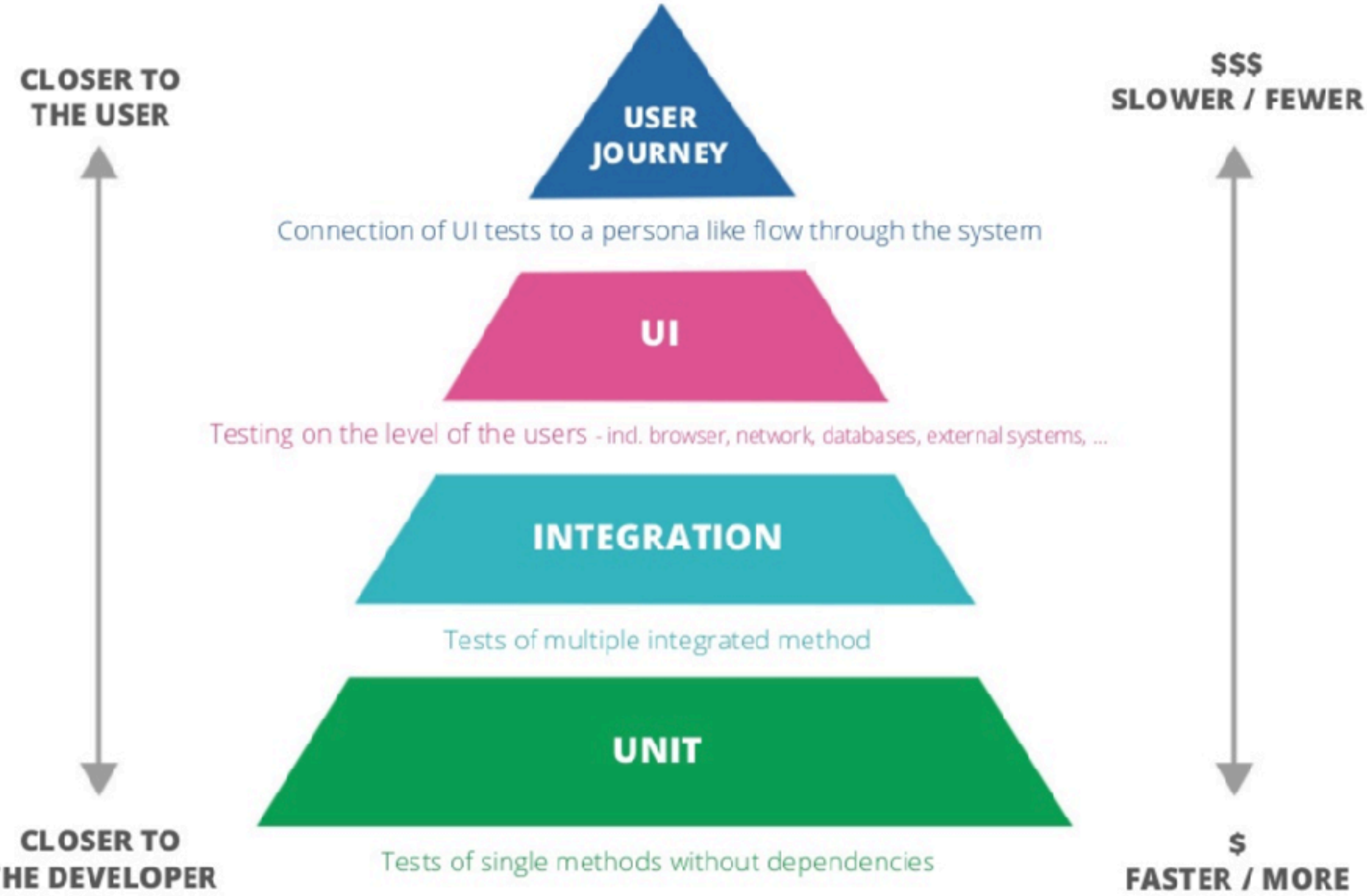


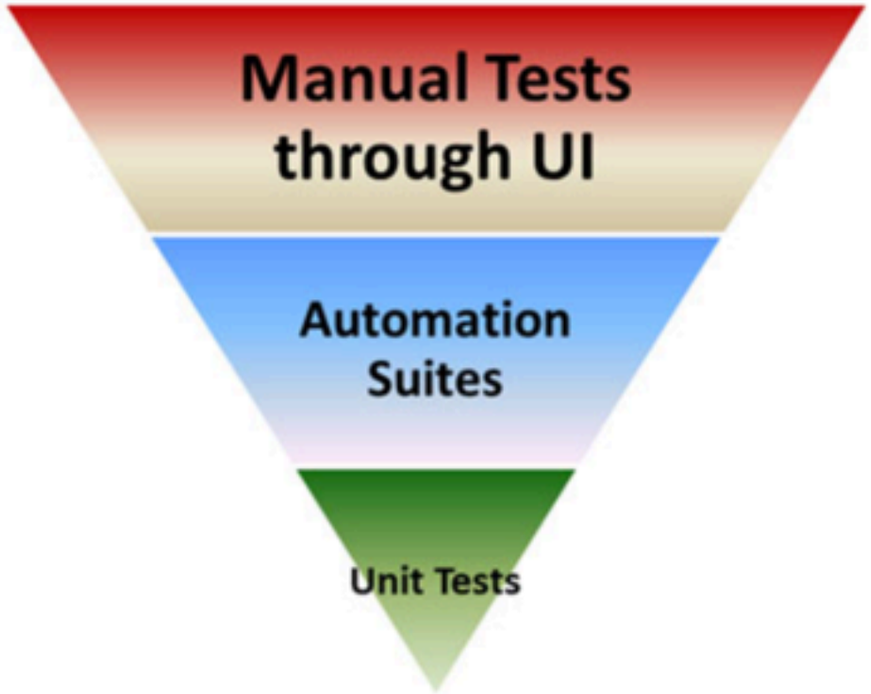

https://martinfowler.com/articles/microservice-testing

# Microservice Testing

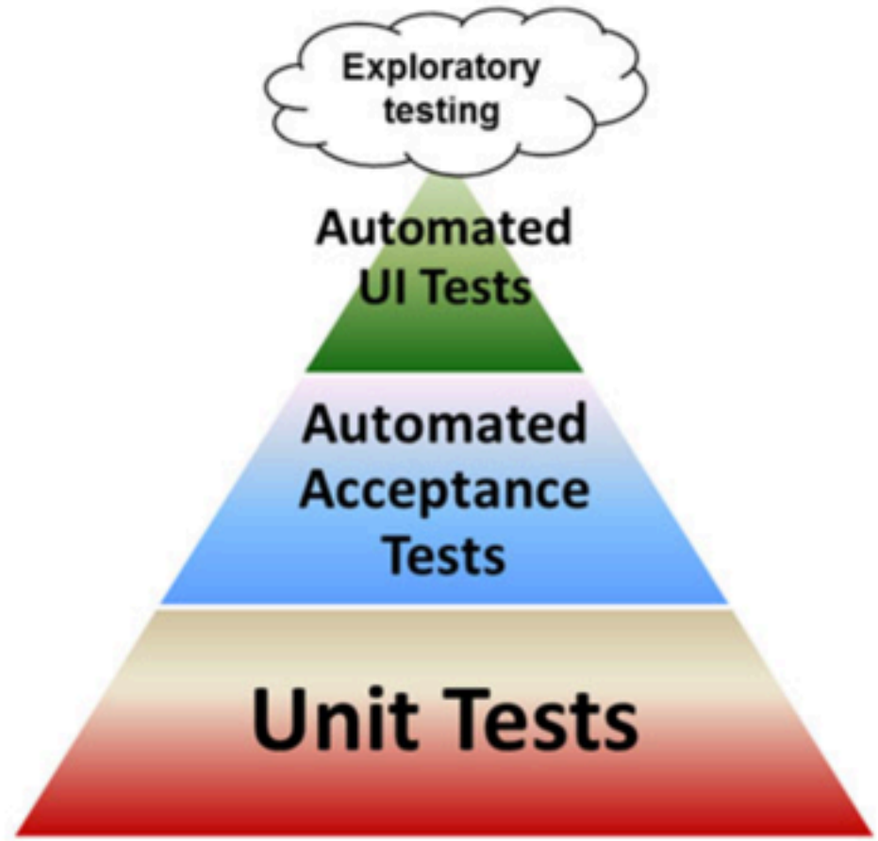






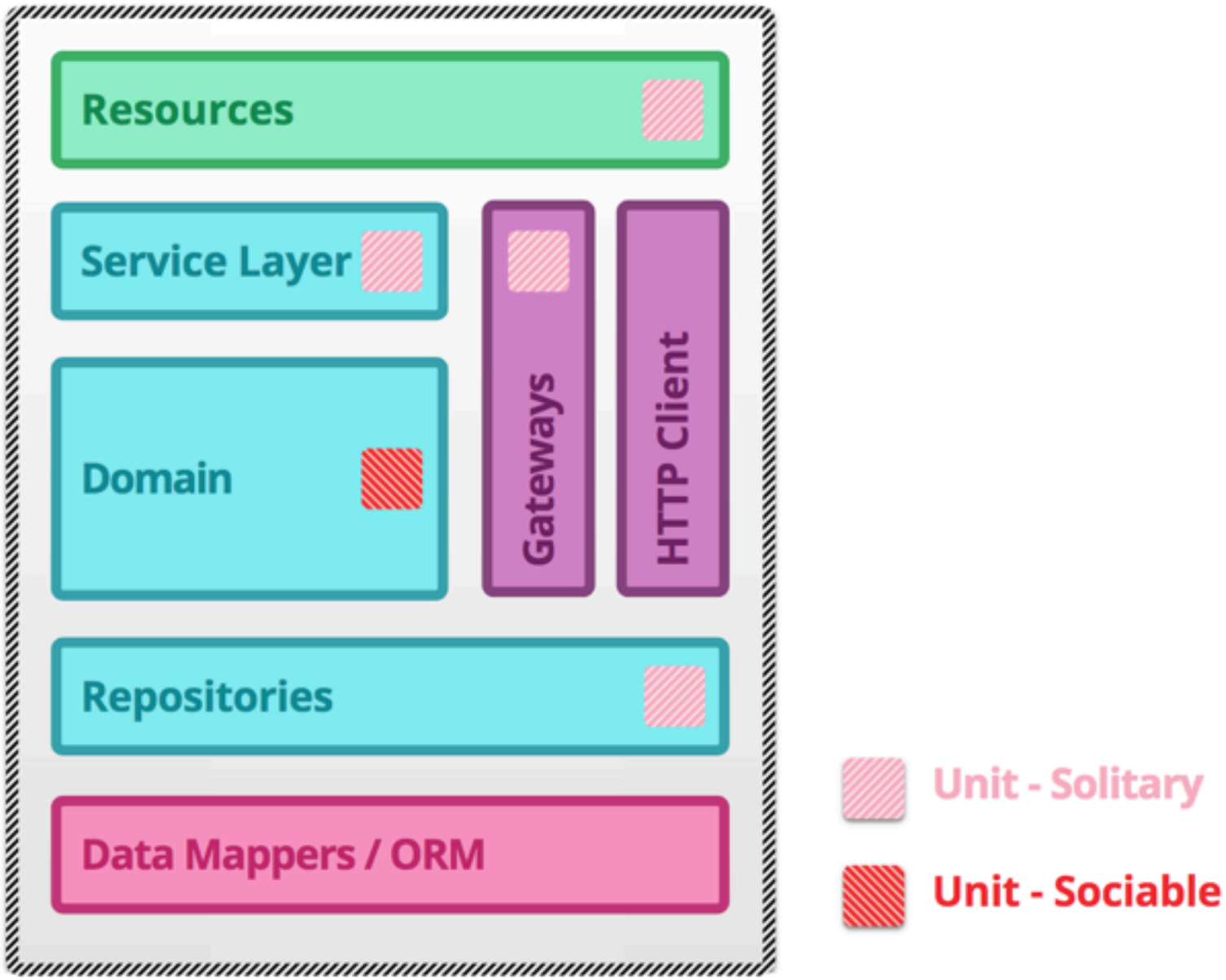



Traditional
(find bugs)

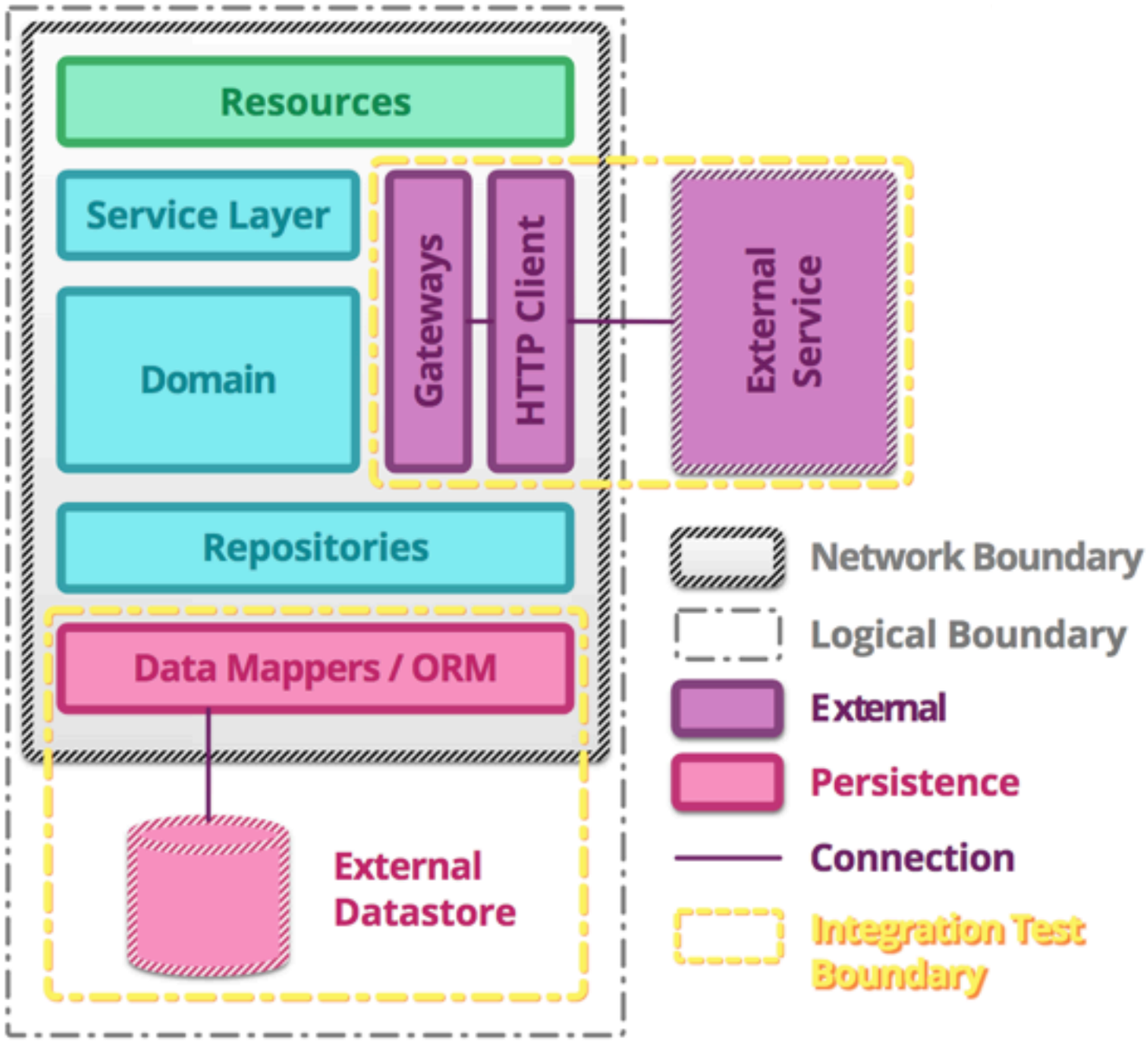



Agile
(prevent bugs)

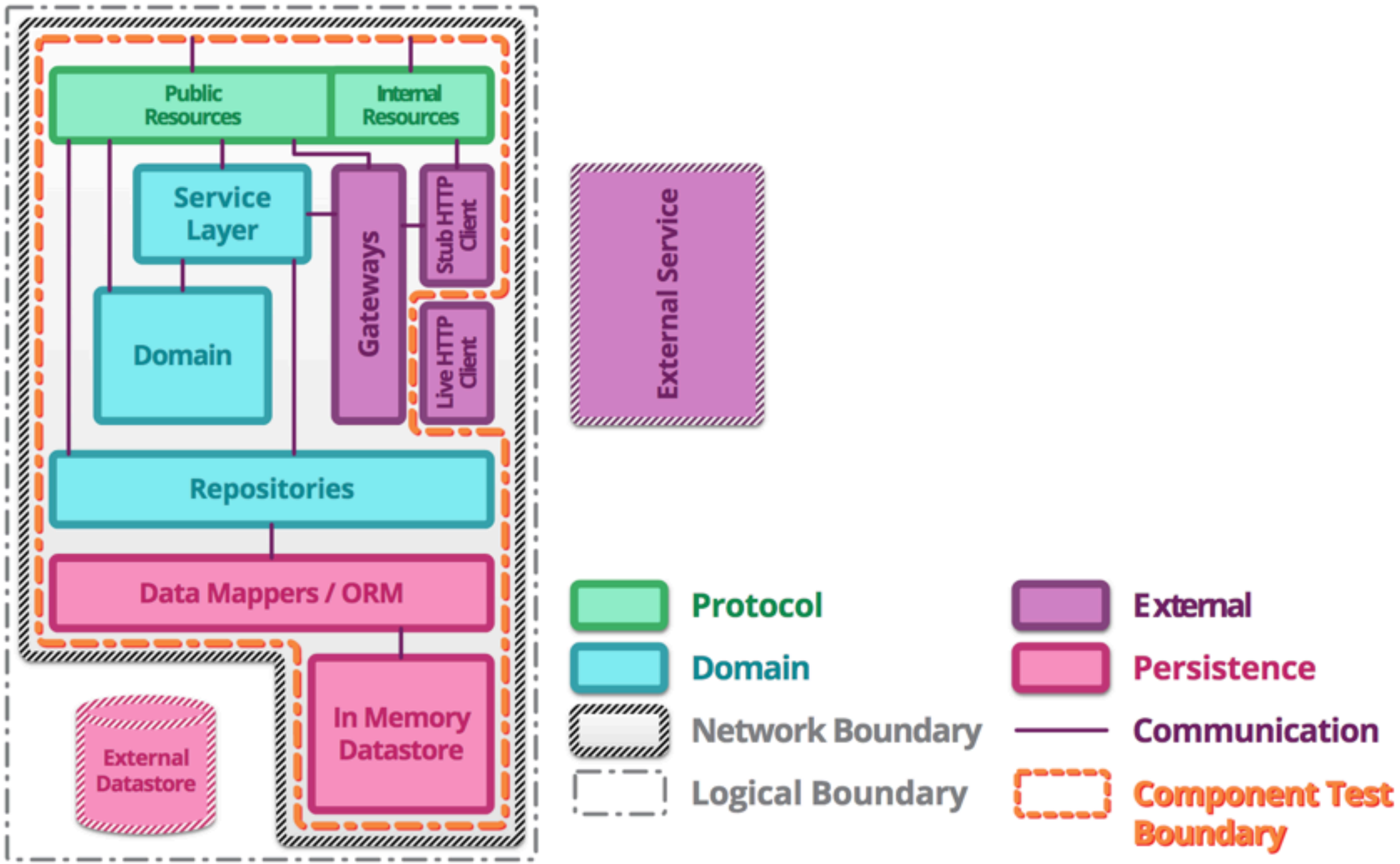
# Unit testing



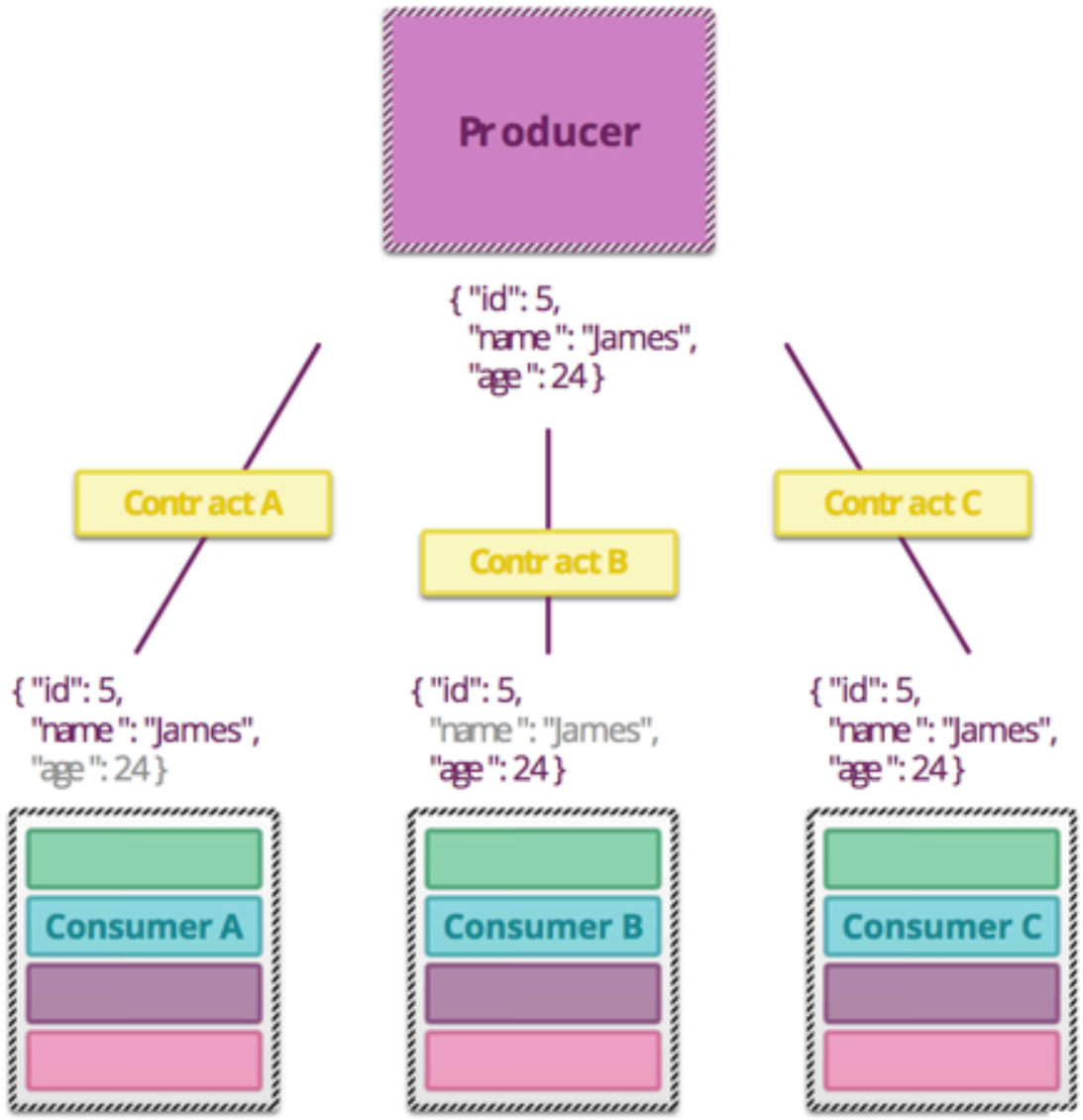
# Integration testing



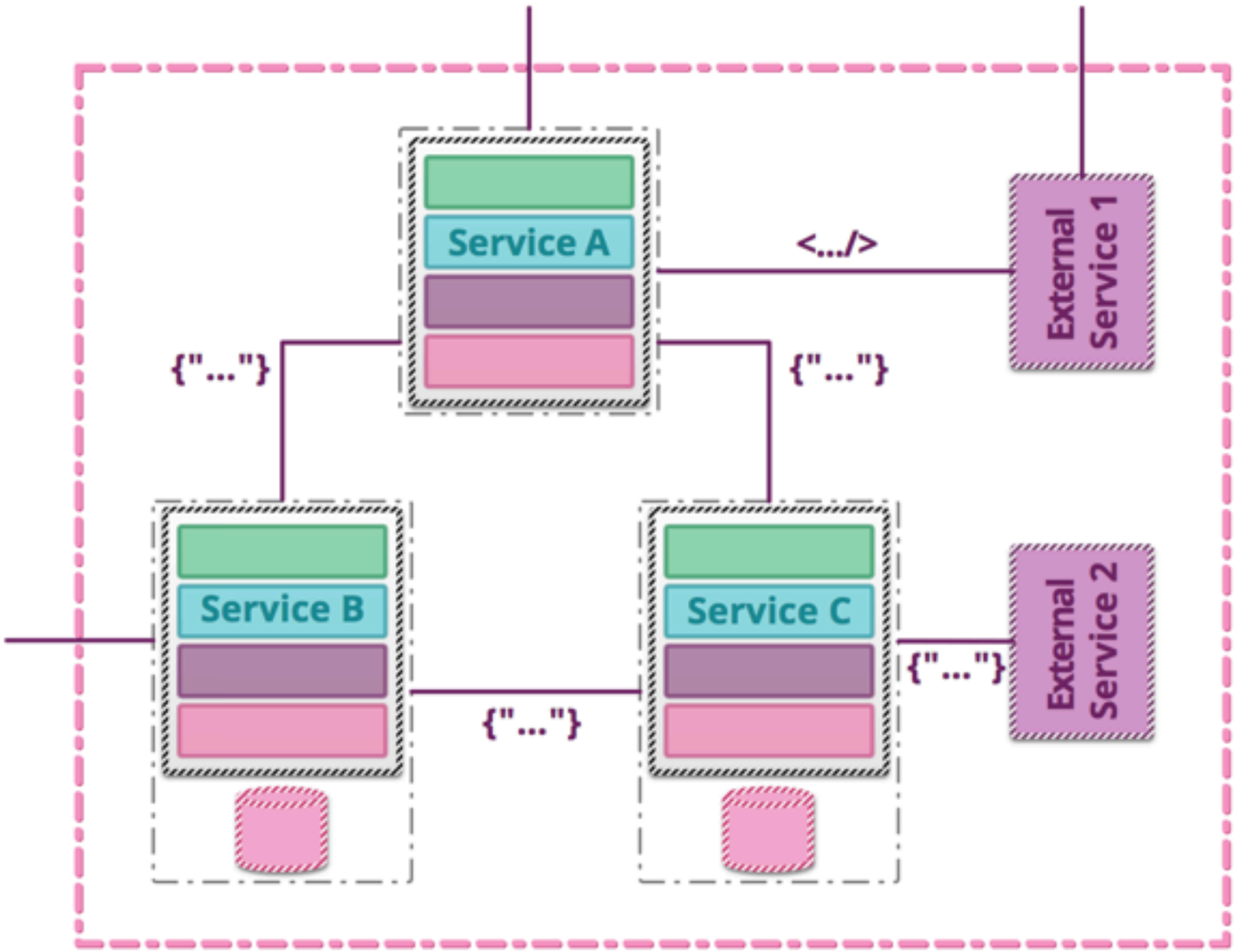
# Component testing



# Contract testing



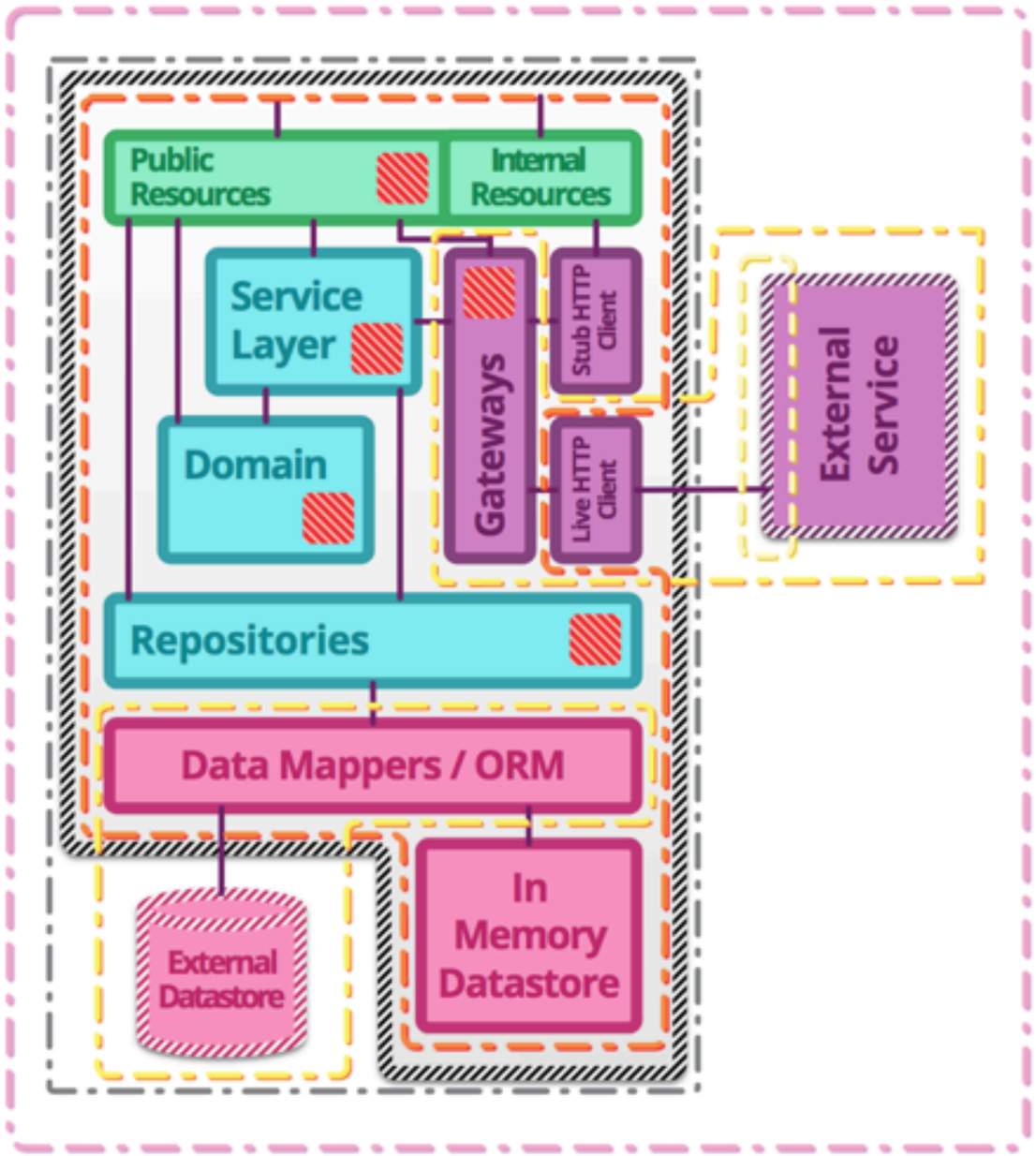
# End-to-End testing



# Summary

**Unit tests** : exercise the smallest pieces of testable software in the application to determine whether they behave as expected.

**Integration tests** : verify the communication paths and interactions between components to detect interface defects.




**Component tests** : limit the scope of the exercised software to a portion of the system under test, manipulating the system through internal code interfaces and using test doubles to isolate the code under test from other components.

**Contract tests** : verify interactions at the boundary of an external service asserting that it meets the contract expected by a consuming service.

**End-to-end tests** : verify that a system meets external requirements and achieves its goals, testing the entire system, from end to end.

# What is your testing strategy ?

# more ...

# Performance testing ?

# Security testing ?

# Common standards

## OpenID

Allows identity and some metadata only


## OAuth 2.0

Grant a client access to resources based on a newly created set of credentials

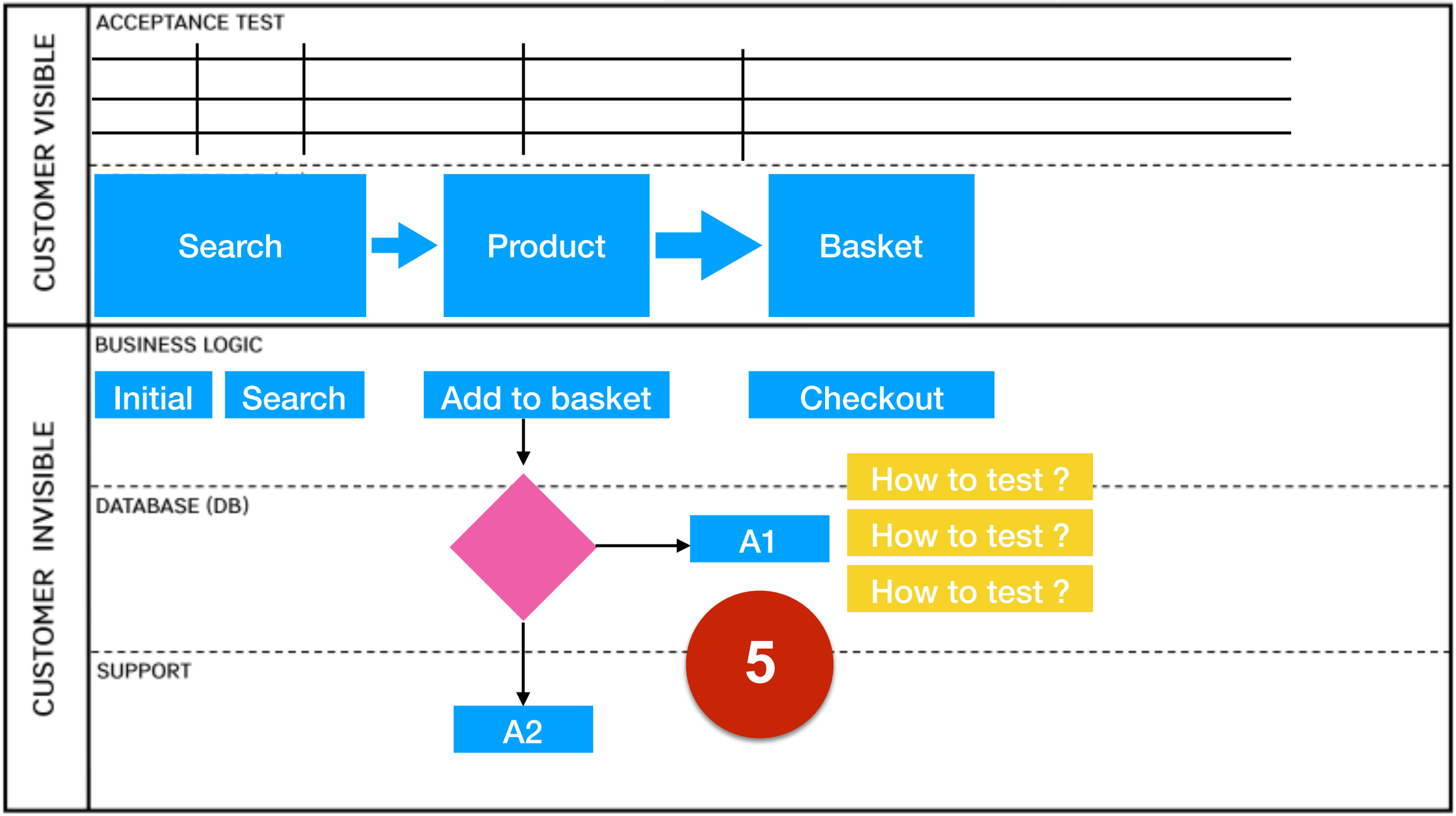
## OpenID Connect

Identity on top of OAuth 2

# Try to test your services

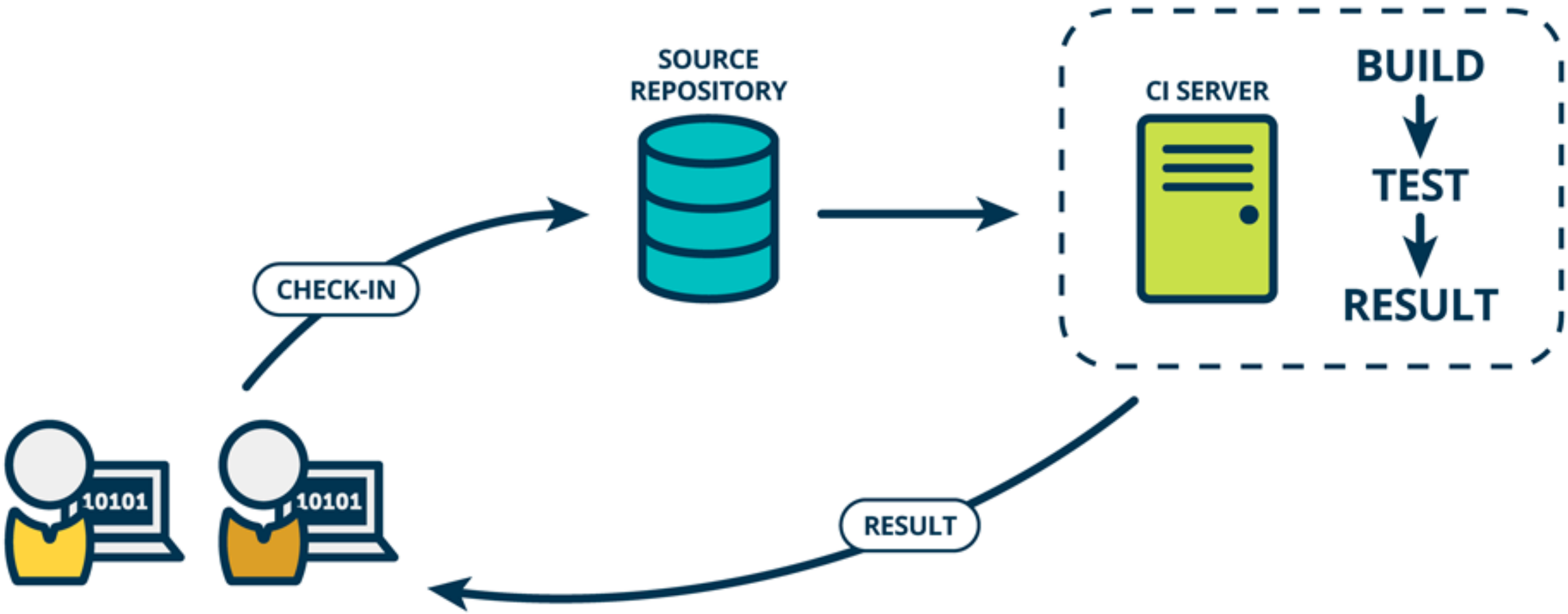
# A-DAPT Blueprint

| THEME: | EPIC: | FEATURE: | STORY: |
|---|---|---|---|
| DESIGNED BY: | DATE: | NOTE: | |

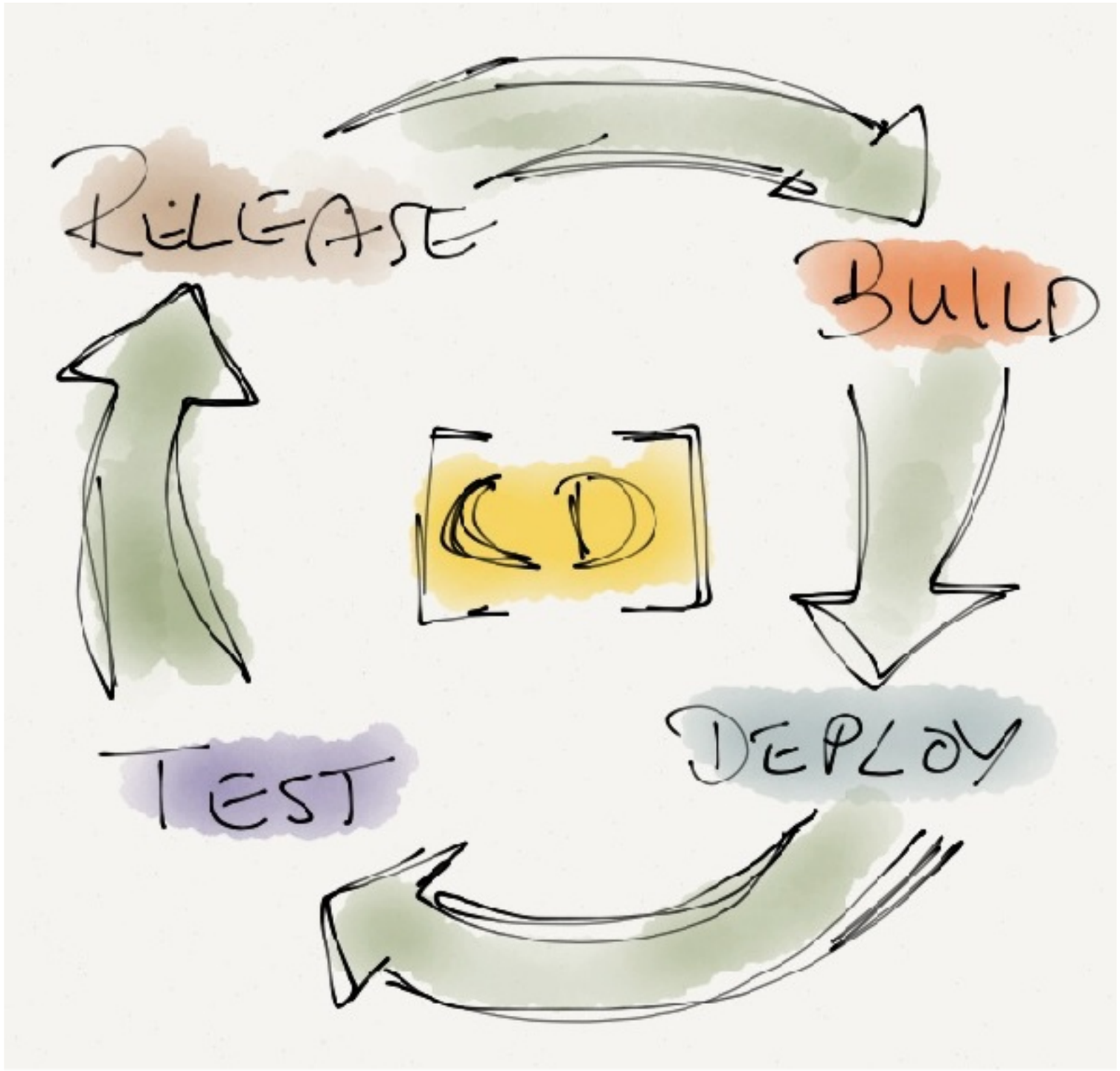


# How to Deploy ?

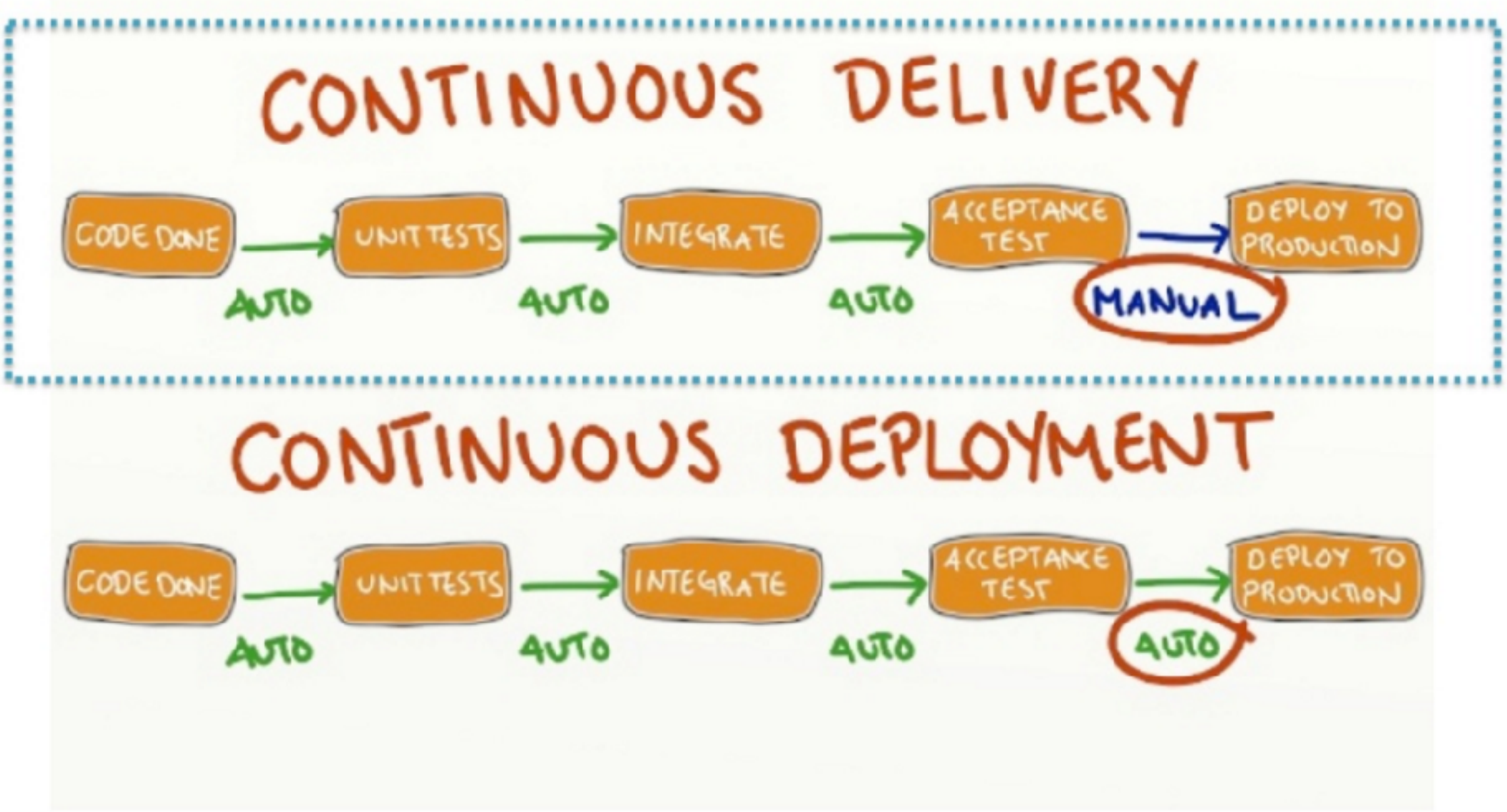
# Continuous Integration



# CD ?

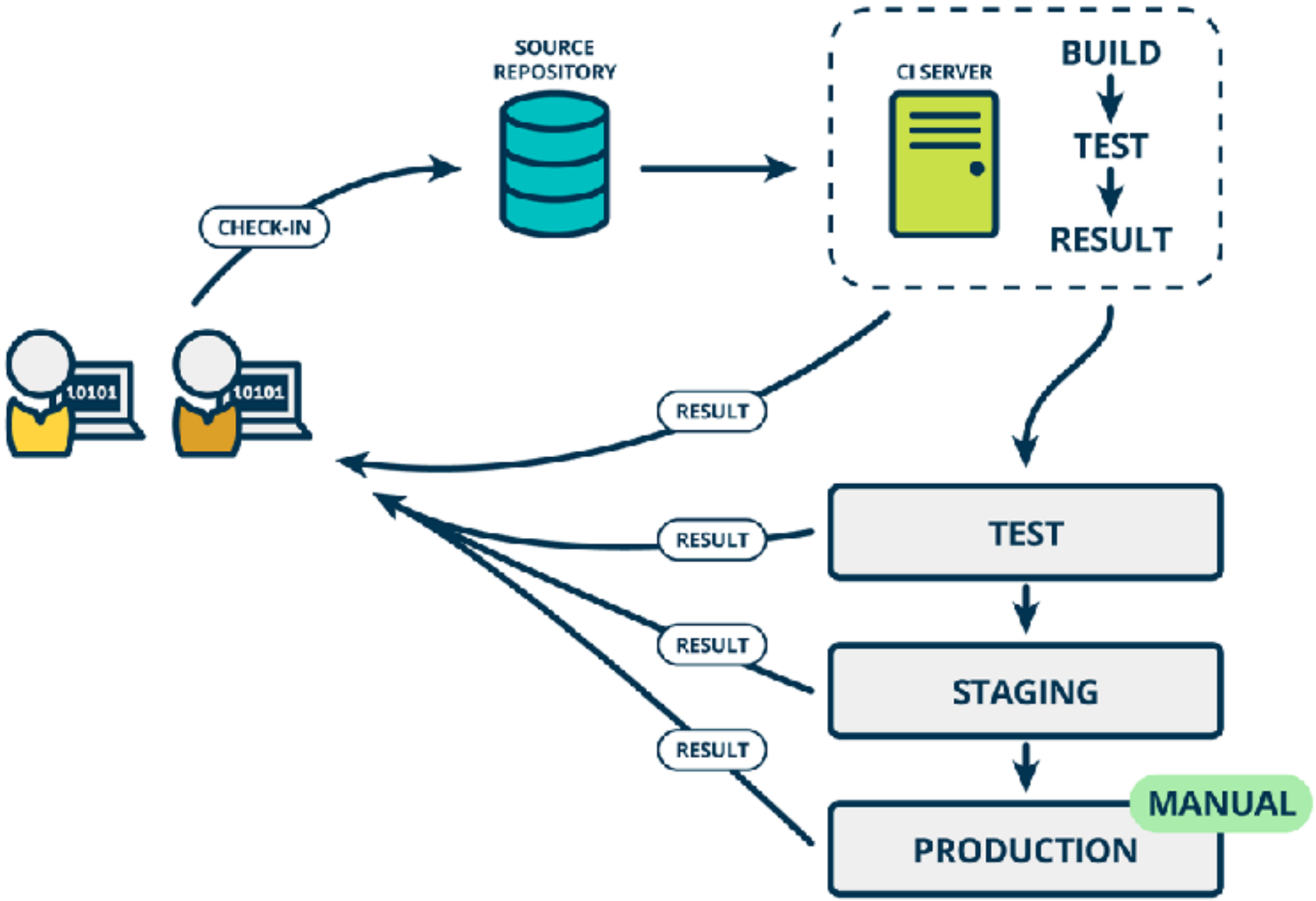


# CD ?

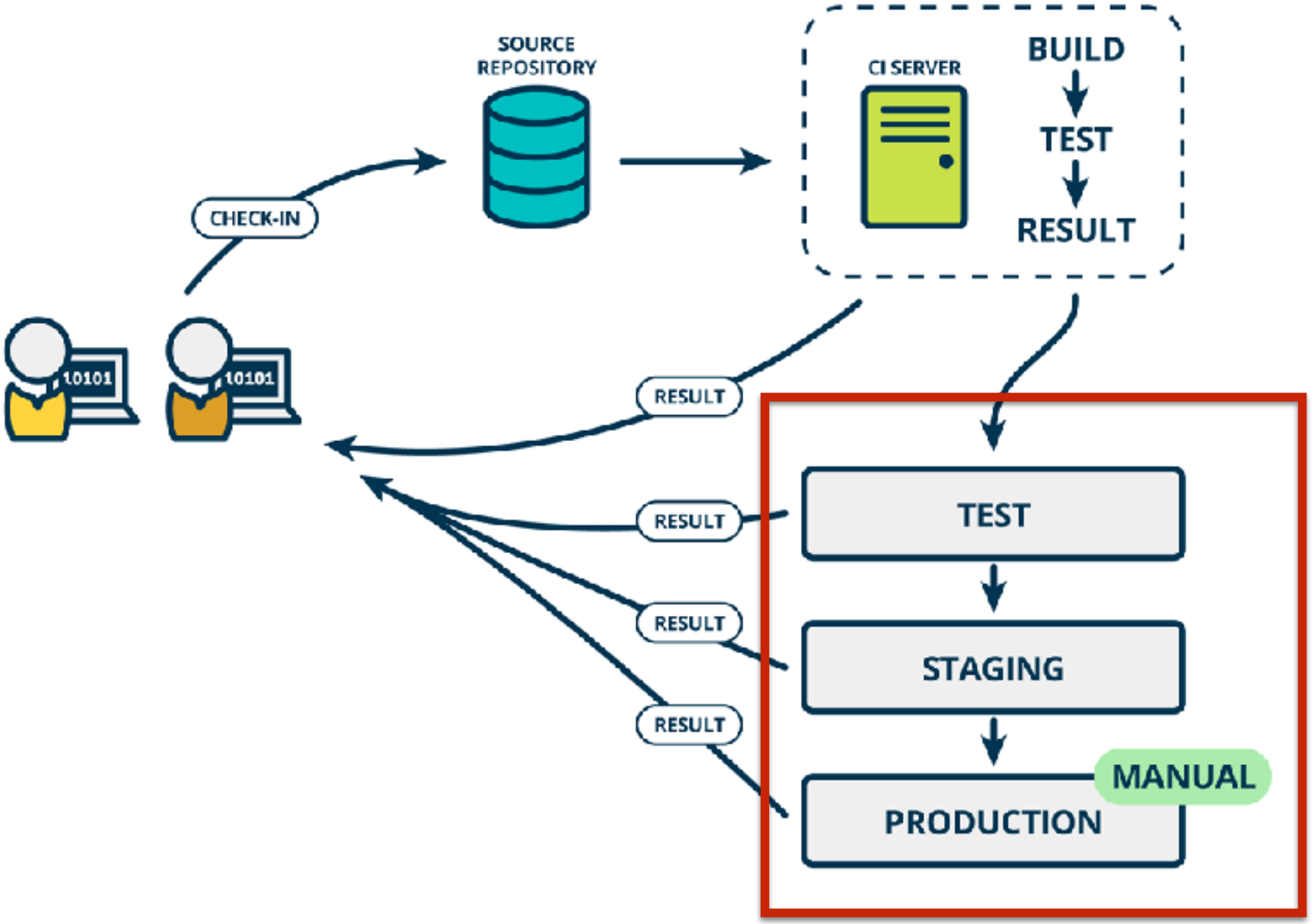



http://blog.crisp.se/2013/02/05/yassalsundman/continuous-delivery-vs-continuous-deployment

# Continuous Delivery



# Rise of DevOps



# More …

# How to Monitoring ?

# How to Configuration ?

# How to Scaling ?

# Summary

# Take to your home

Testing a monolith is easy !!,
think about your **service testing strategy**

# Take to your home

**Security must bundle in development**
workflow but not make slow down

# Take to your home

You build it you run it

# Take to your home

Focus on **value**, not **quantity**

# Suggestion

Don’t start with a Microservices

# Suggestion

Start with a monolith, keep it modular

# Suggestion

Split it into Microservices when become a problem

# Suggestion

Start with one manageable step at a time

# Suggestion

Reduce overhead and increase performance

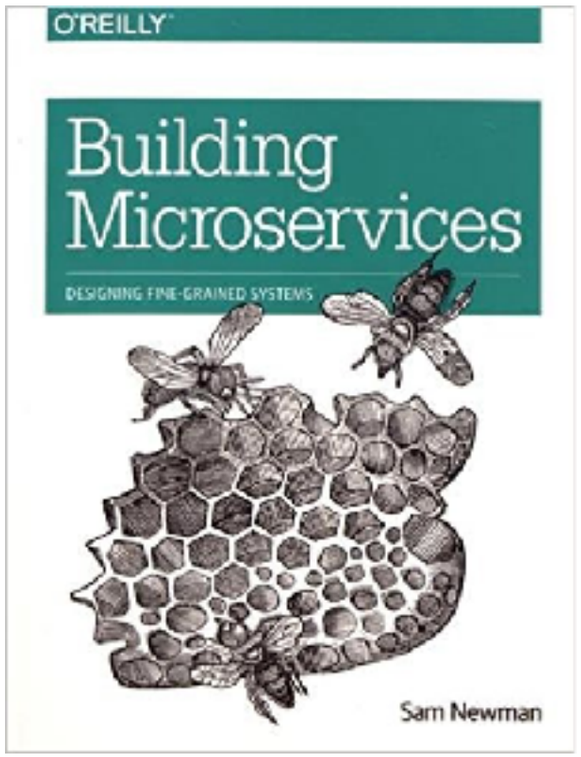
# Suggestion

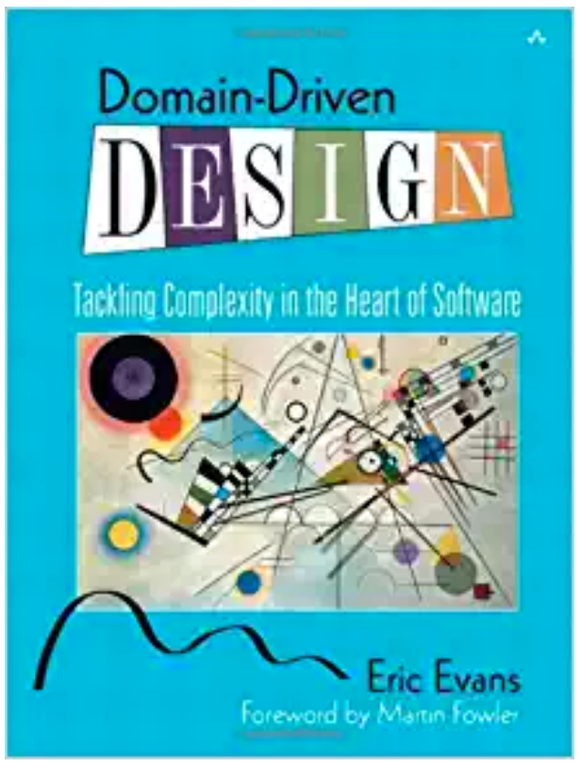
Take care of Authorization handling early

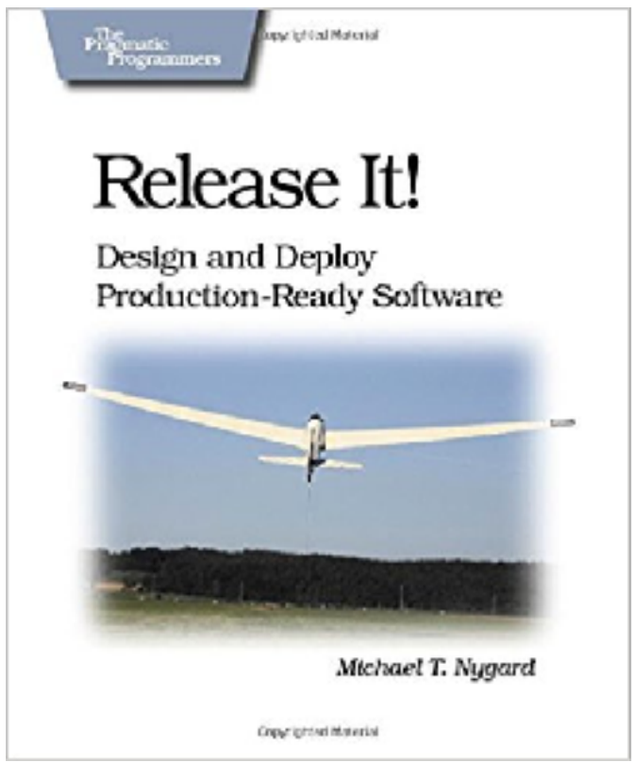
# Suggestion

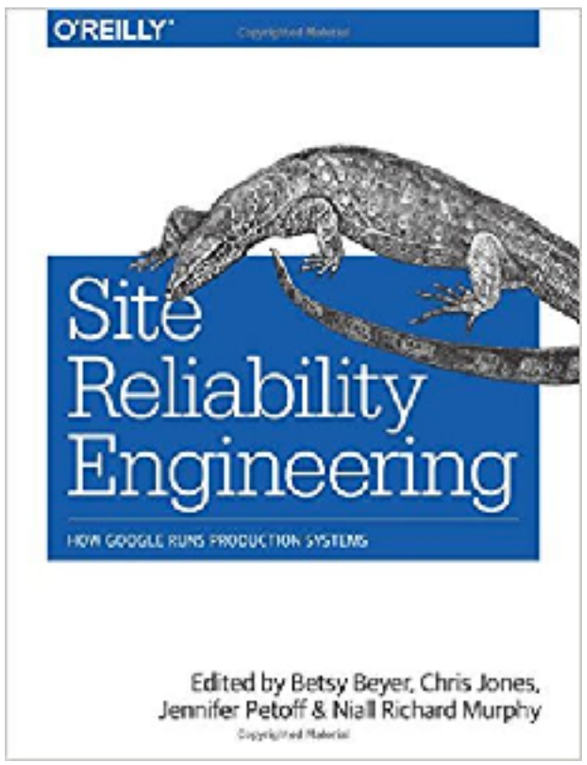
Align strategy and Microservices goals

# Resources




# Thank you
# Q/A