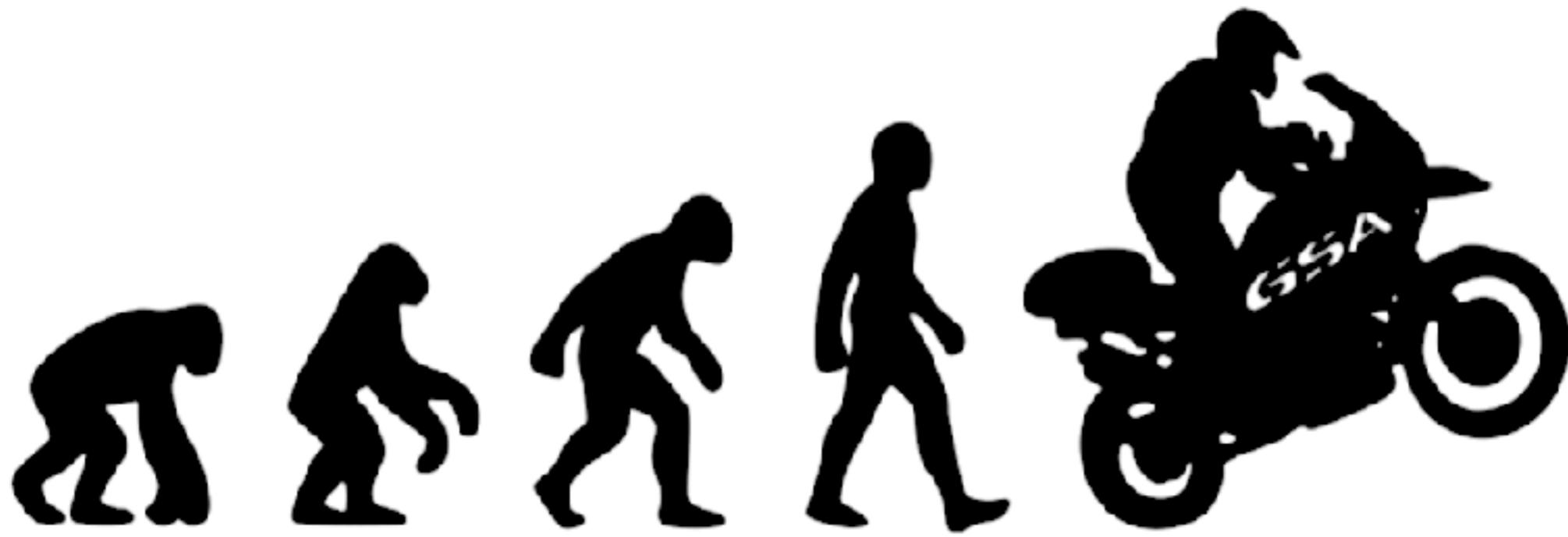


# Evolution of Architecture



monolith

n-tier

service  
oriented

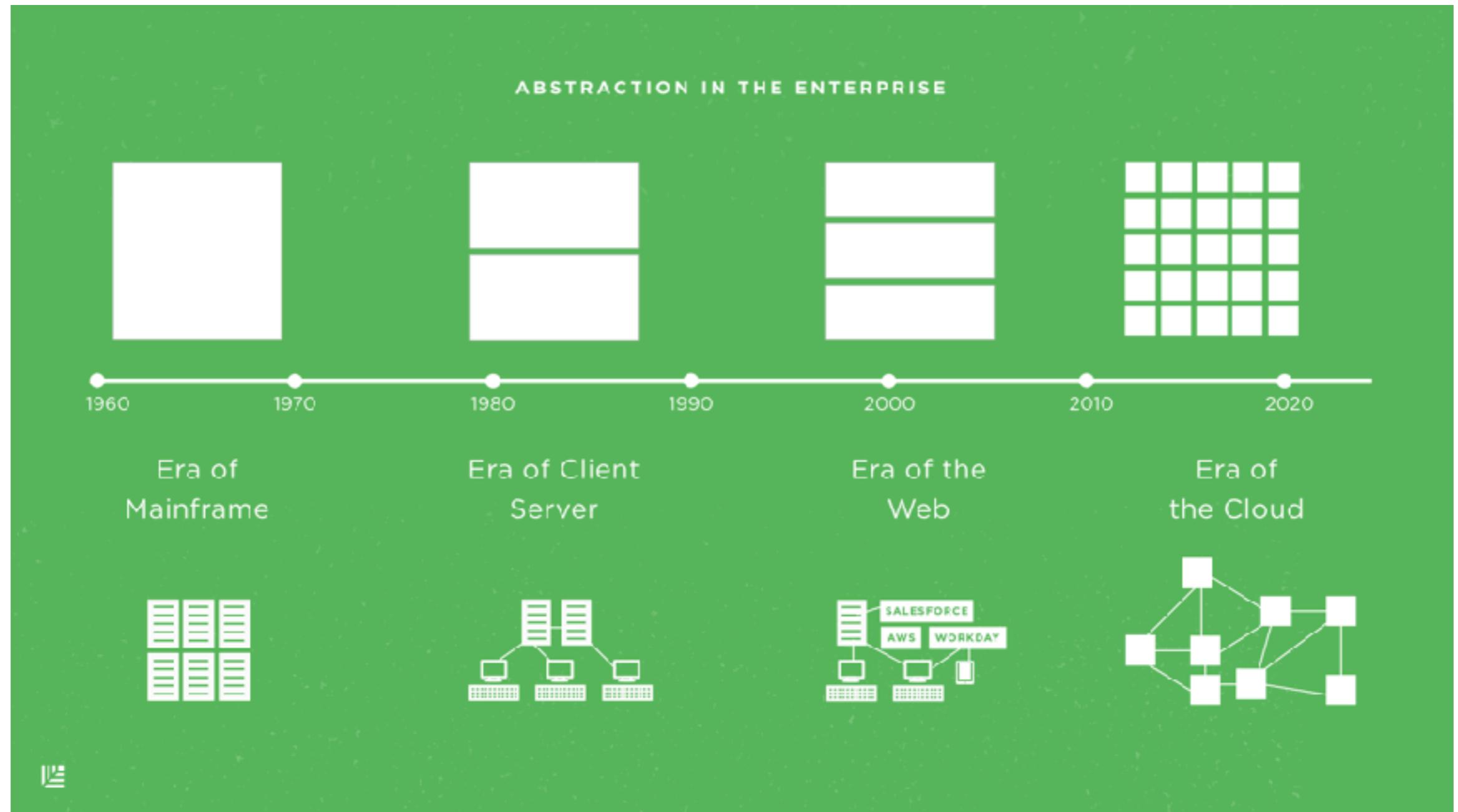
microservices

serverless

@somkiat



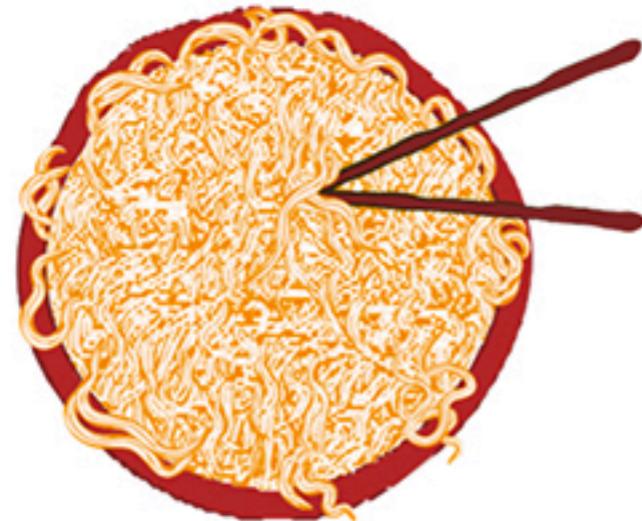
# Evolution of Architecture



# Developer's perspective

1990s and earlier

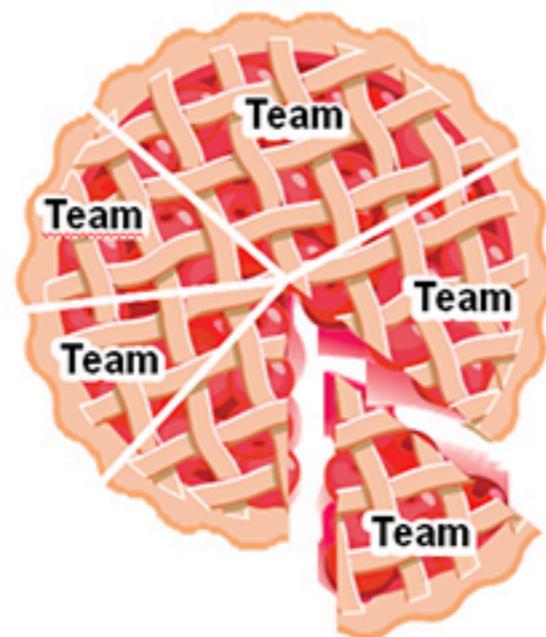
Pre-SOA (monolithic)  
Tight coupling



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.

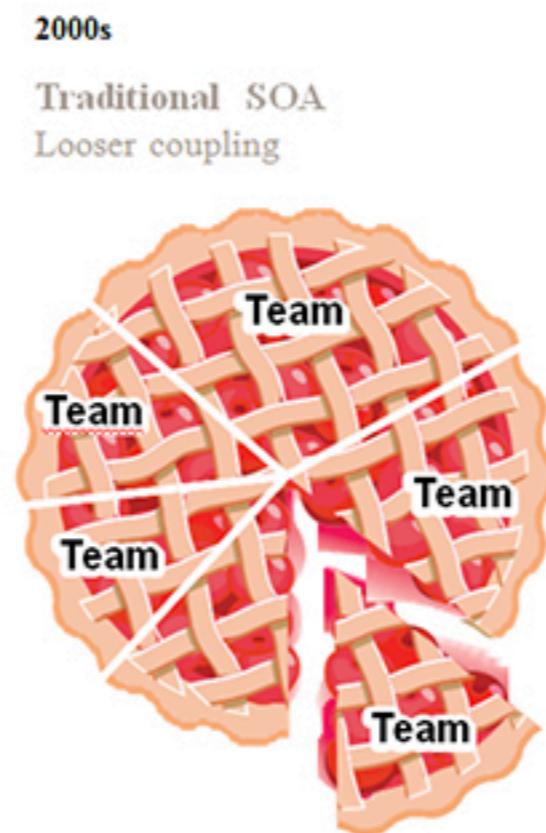
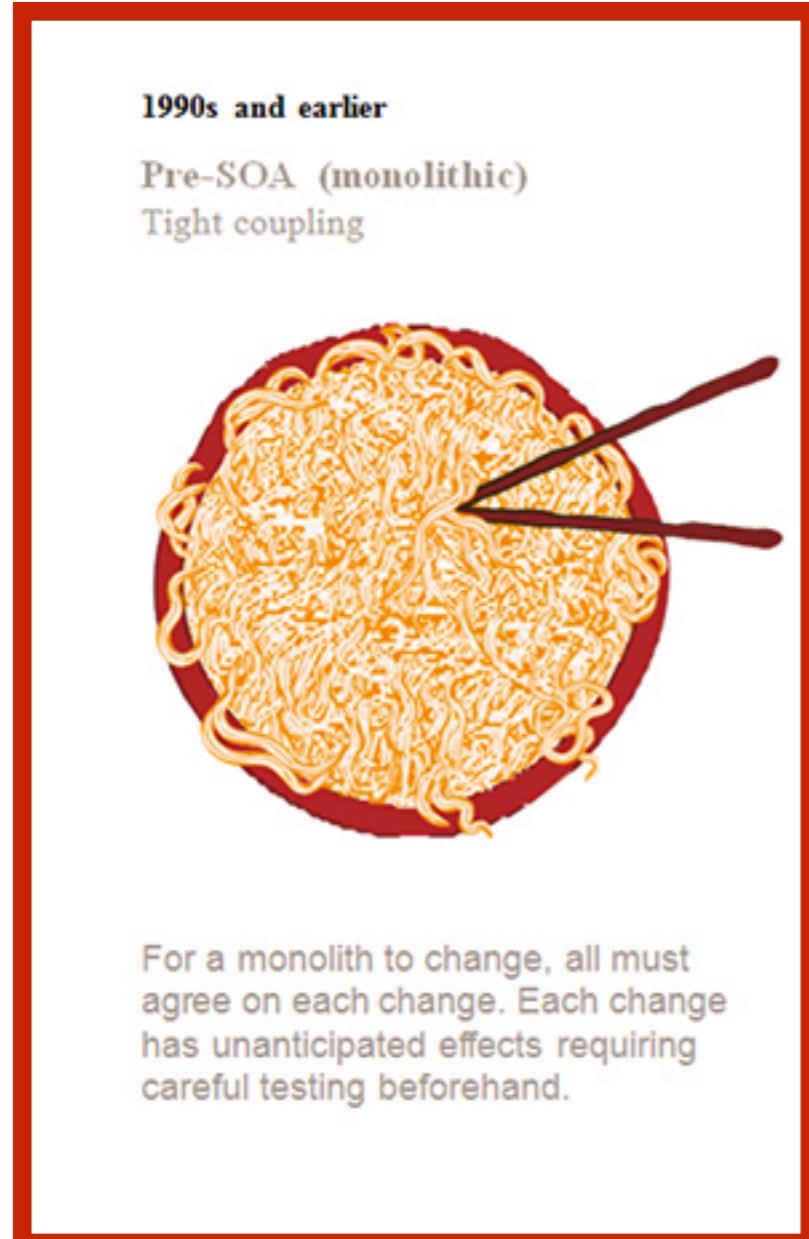
2010s

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.

# Developer's perspective



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

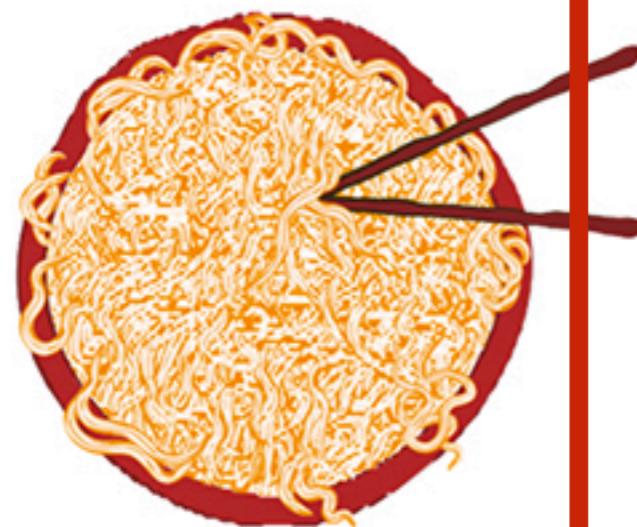


Developers can create and activate new microservices without prior coordination with others. Their adherence to MSA principles makes continuous delivery of new or modified services possible.

# Developer's perspective

1990s and earlier

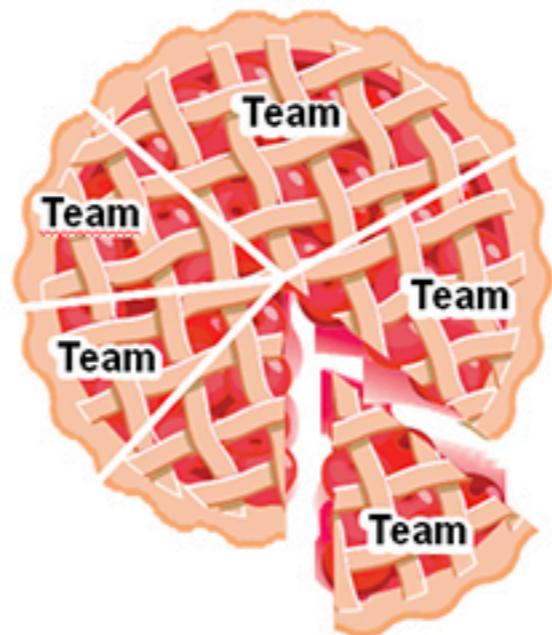
Pre-SOA (monolithic)  
Tight coupling



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



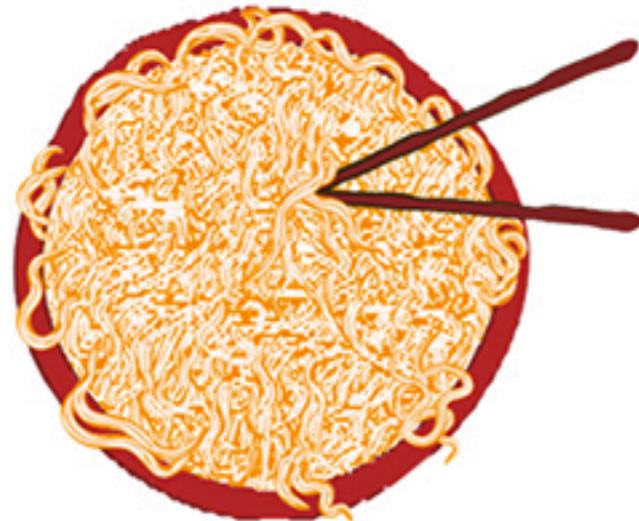
Developers can create and activate new microservices without prior coordination with others. Their adherence to MSA principles makes continuous delivery of new or modified services possible.



# Developer's perspective

1990s and earlier

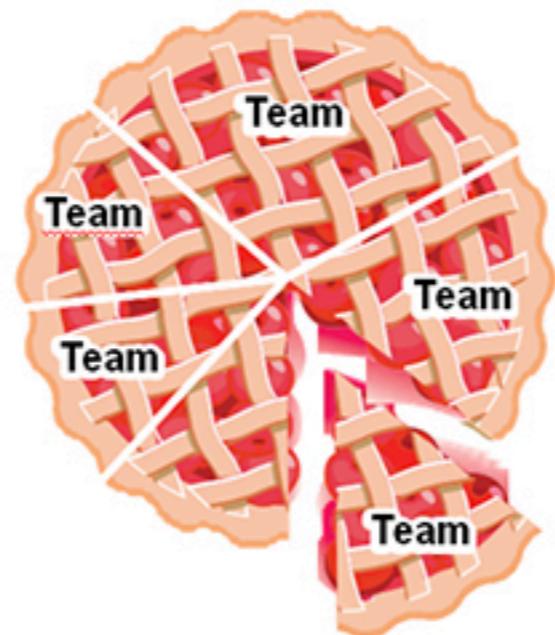
Pre-SOA (monolithic)  
Tight coupling



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.

# Evolution of services

1990s and earlier

## Coupling

Pre-SOA (monolithic)

Tight coupling



2000s

Traditional SOA

Looser coupling



2010s

Microservices

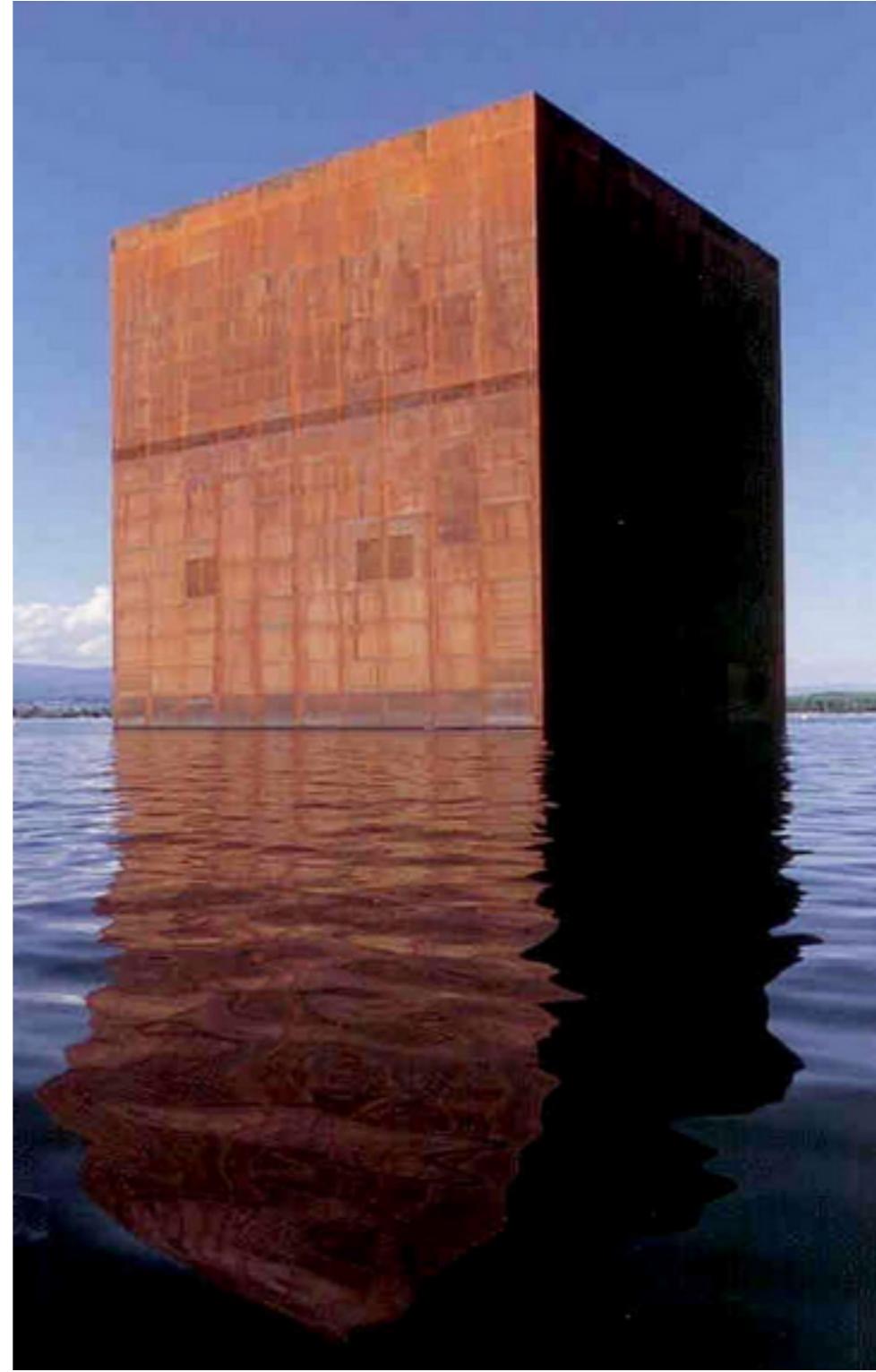
Decoupled



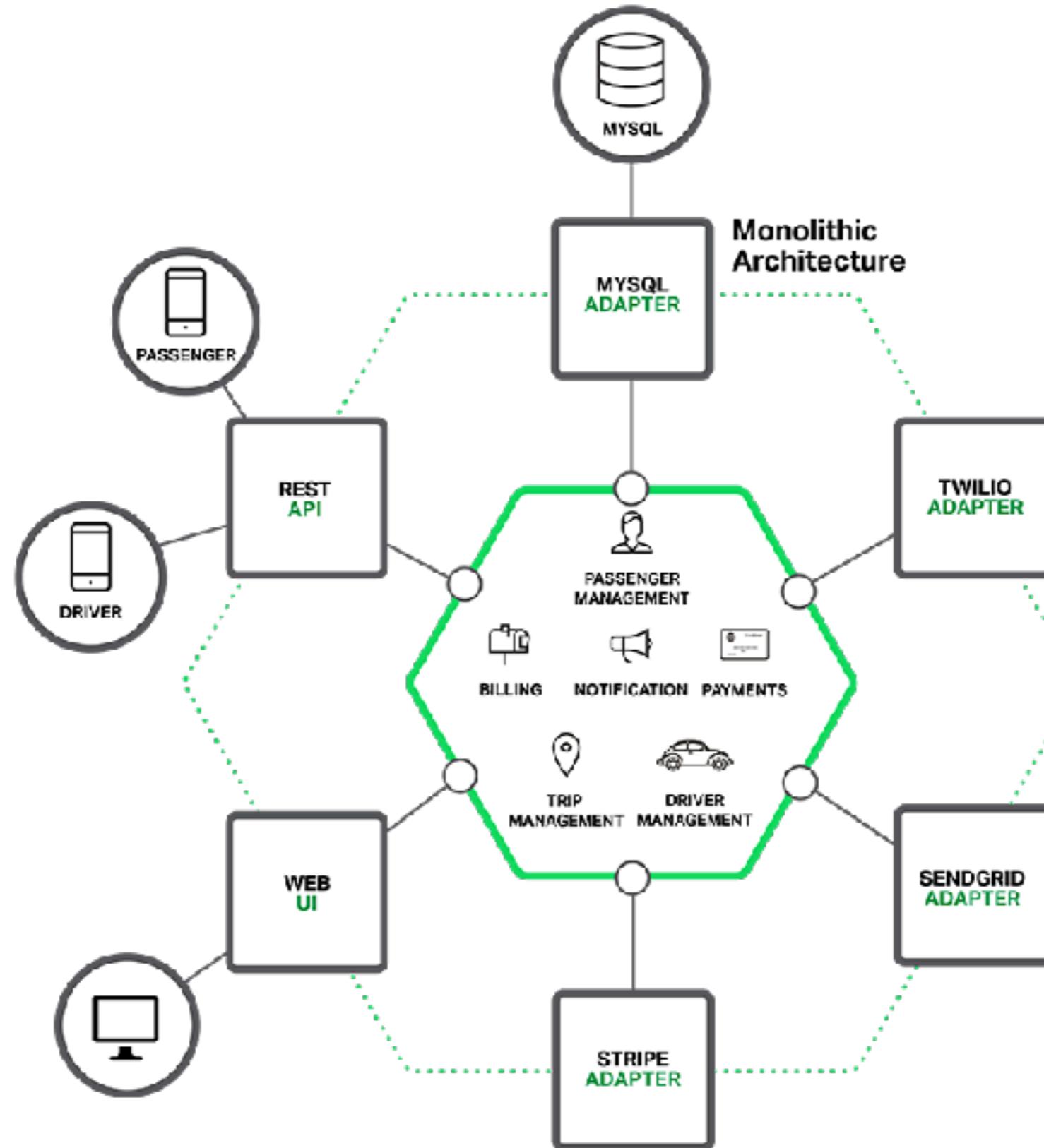
# Monolith



# Monolith



# Monolith



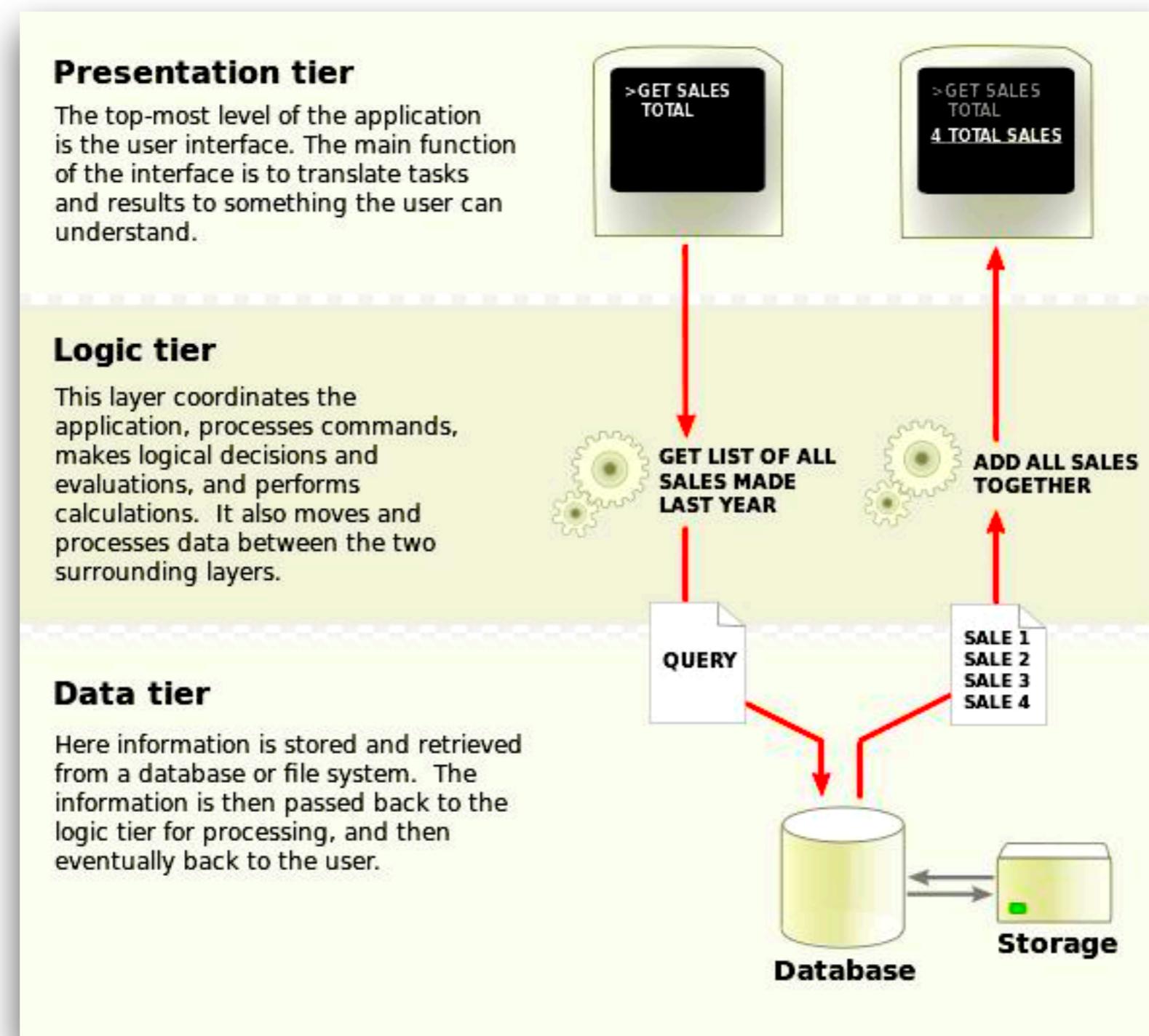
# N-tiers



# N-tiers



# N-tiers

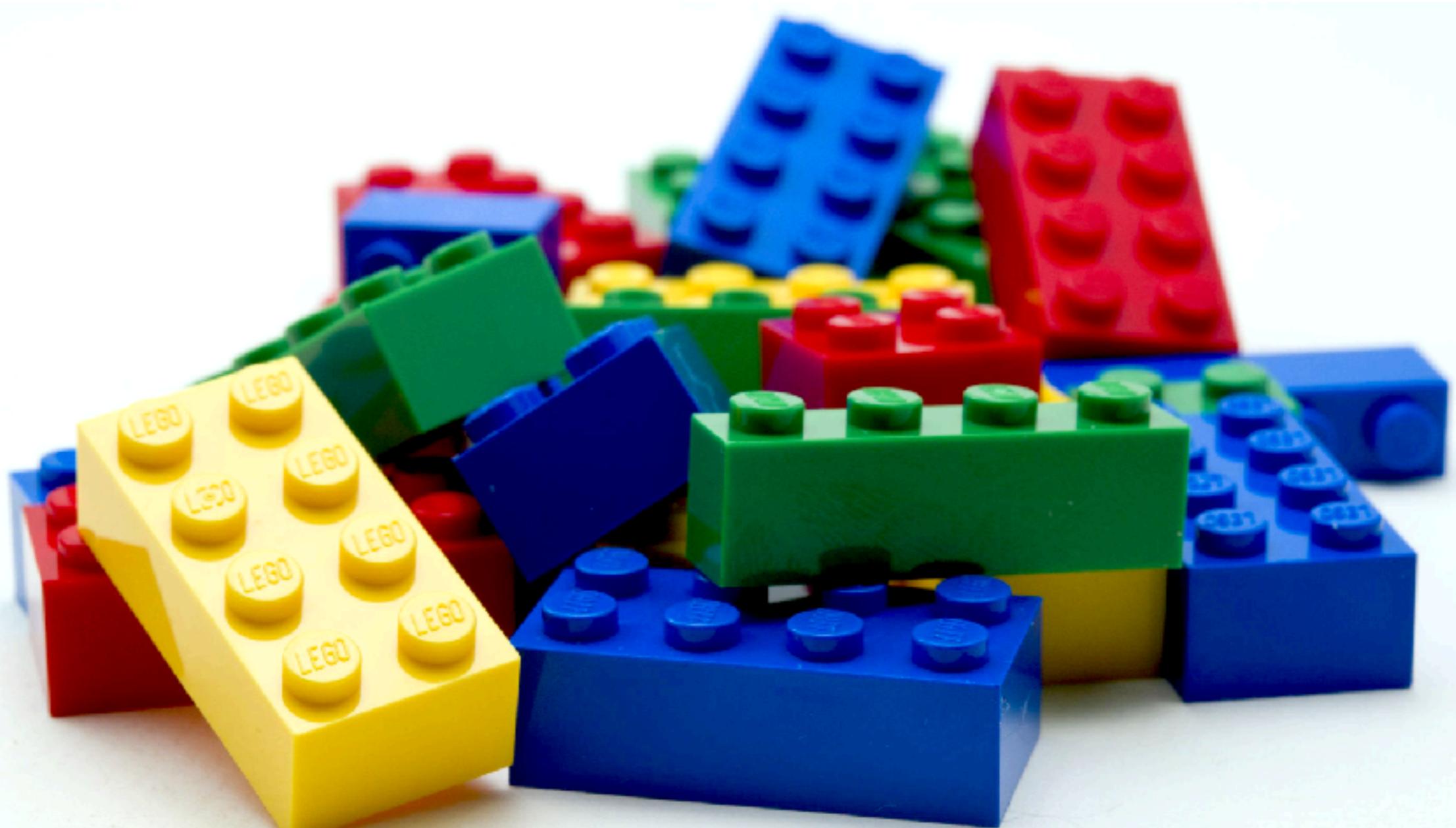


# **SOA**

# **Service Oriented Architecture**



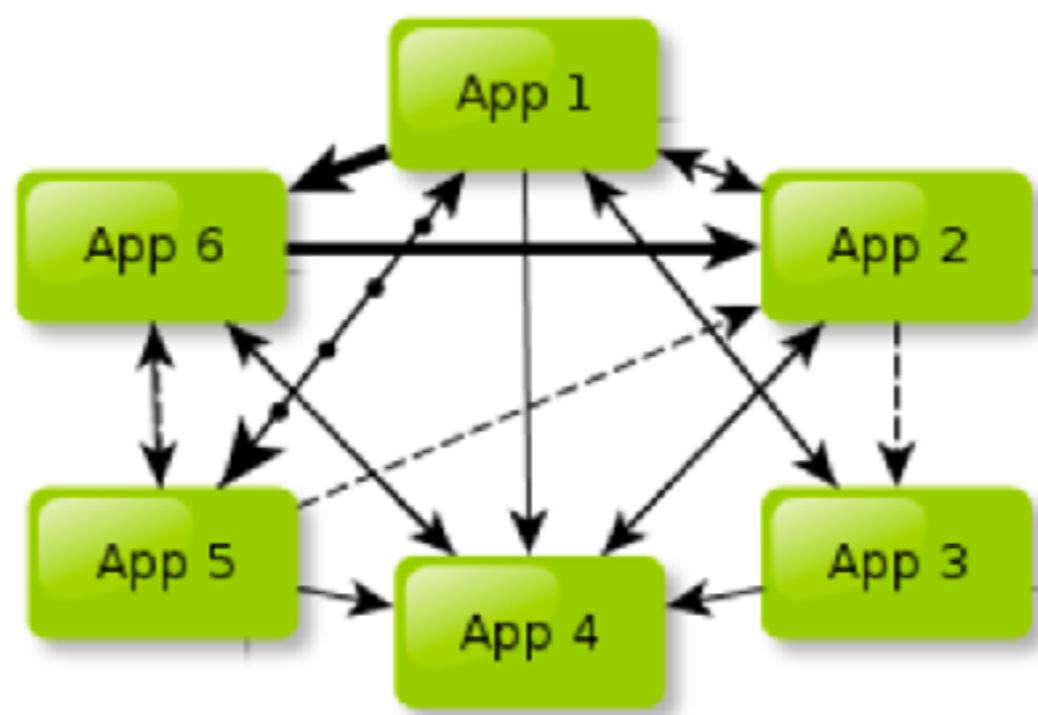
# SOA



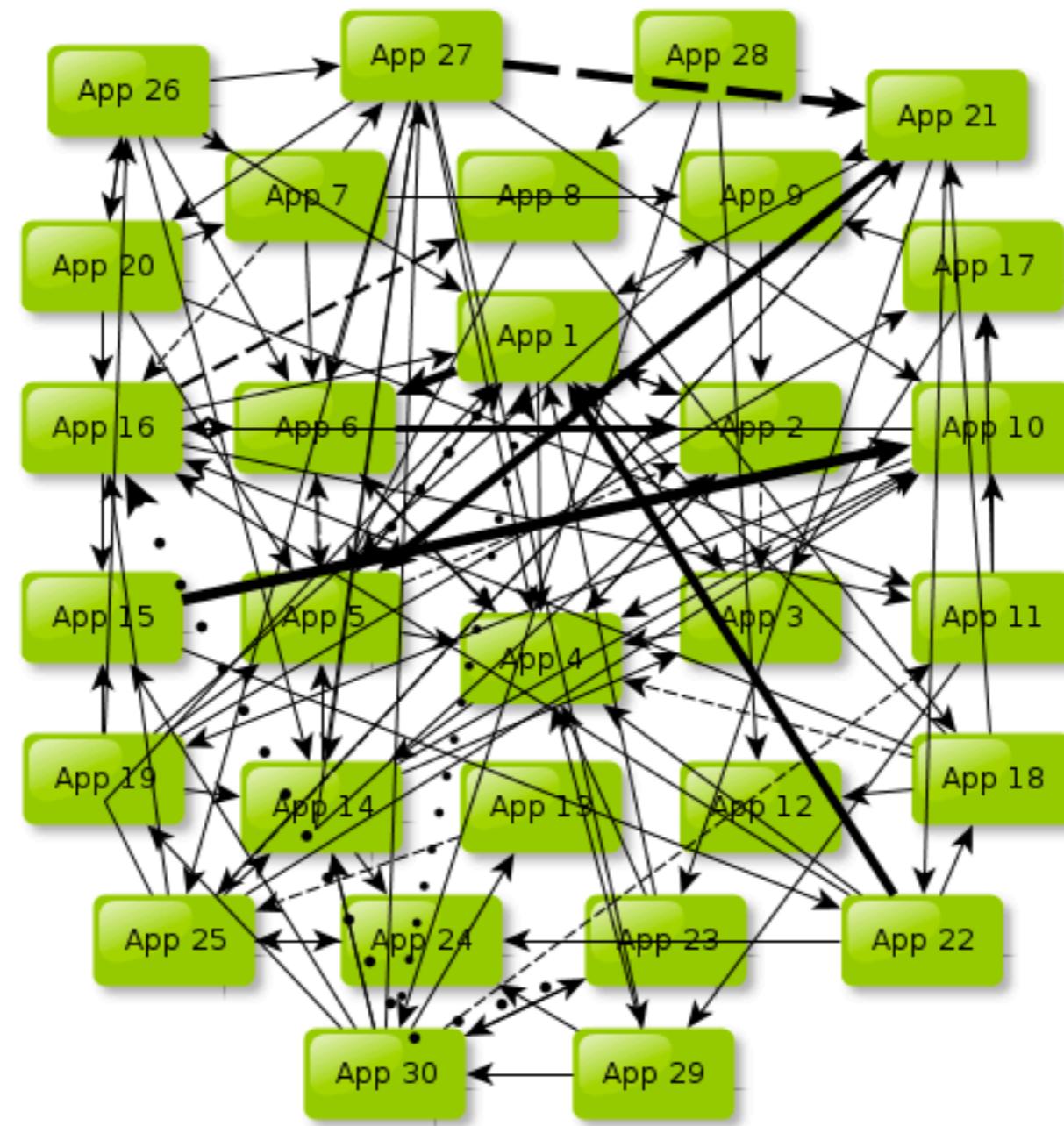
# SOA



# SOA ?



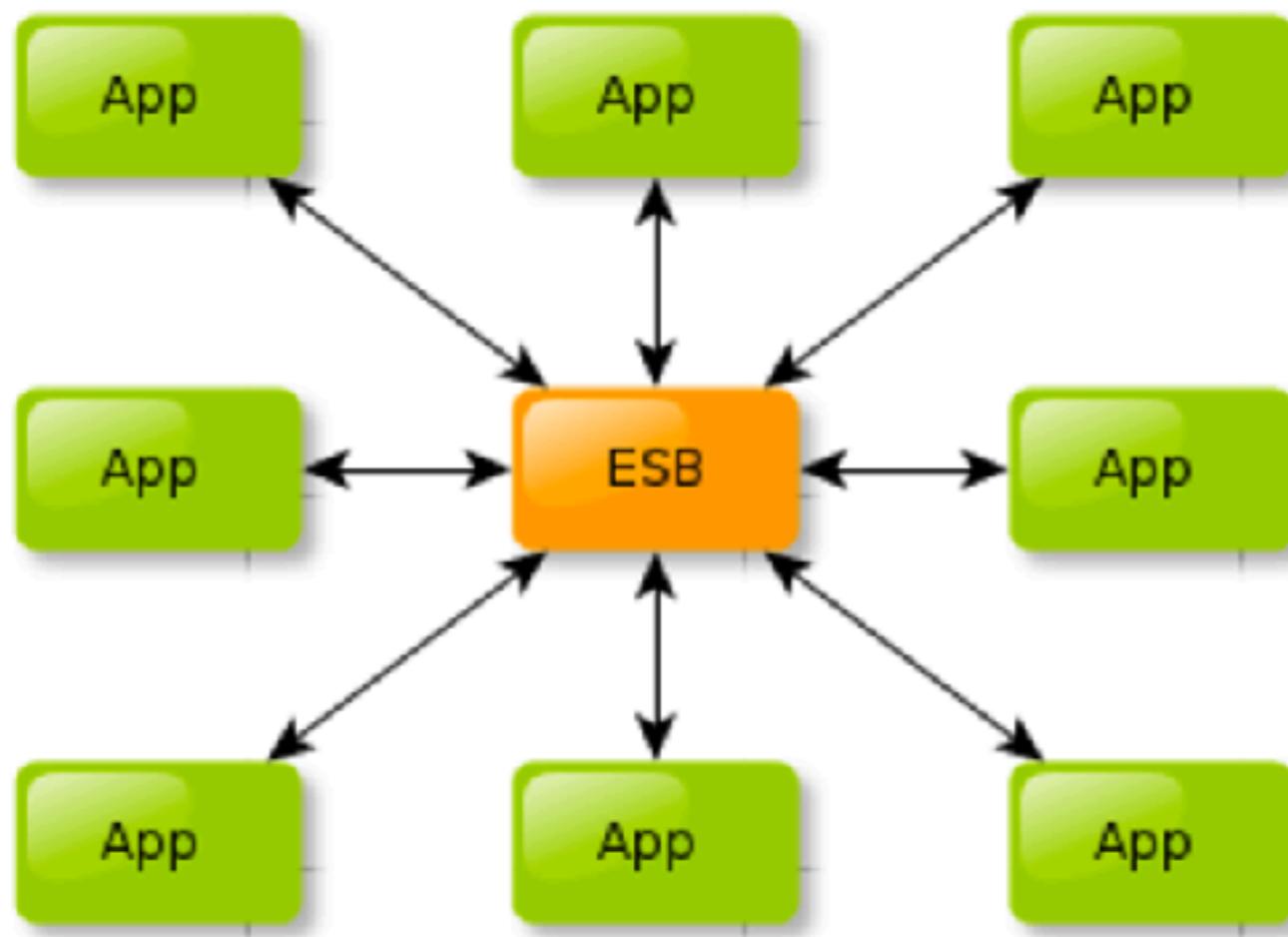
# SOA ?



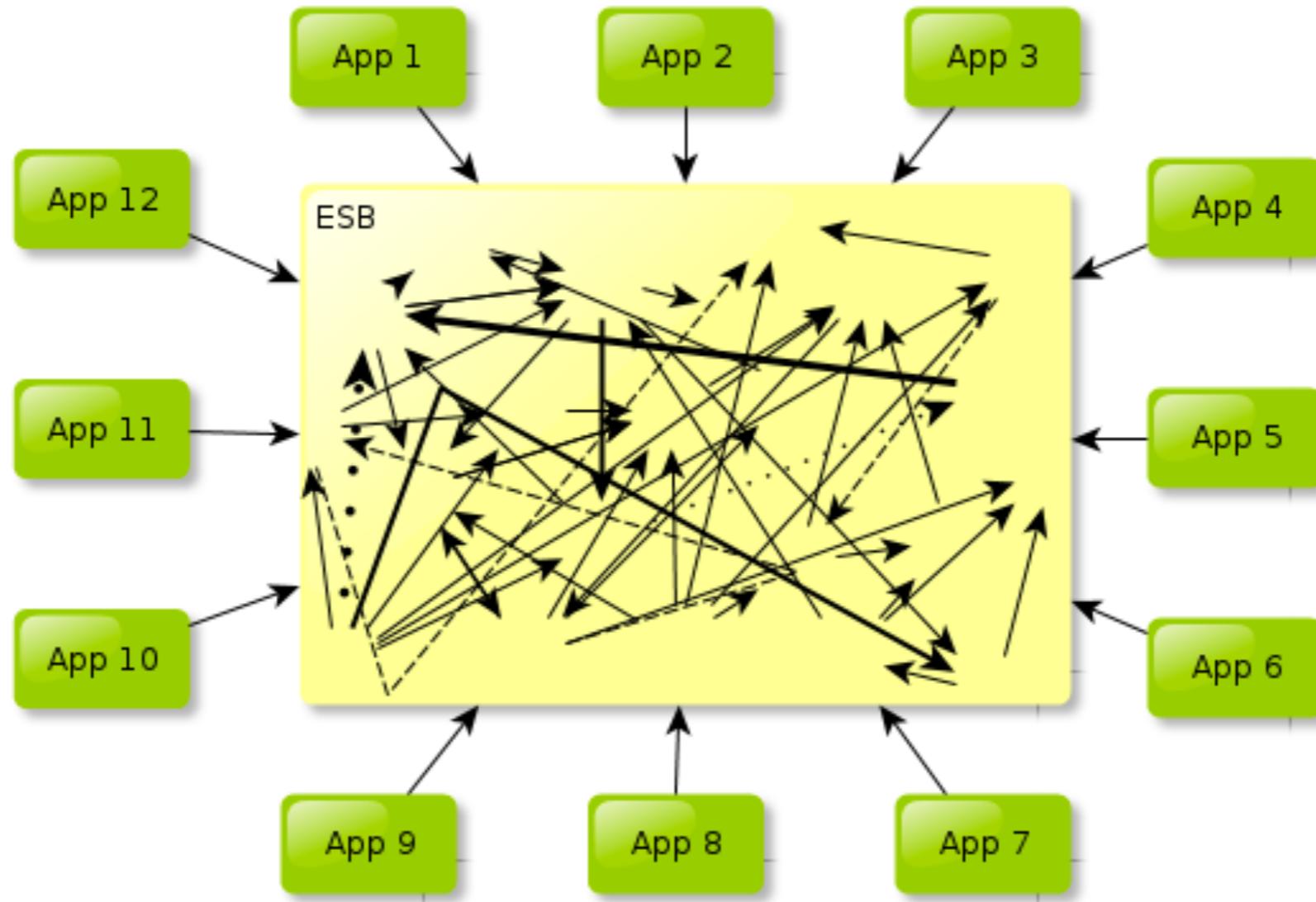
# Solution ?



# SOA with ESB



# SOA with ESB



# ESB

## Enterprise Service Bus

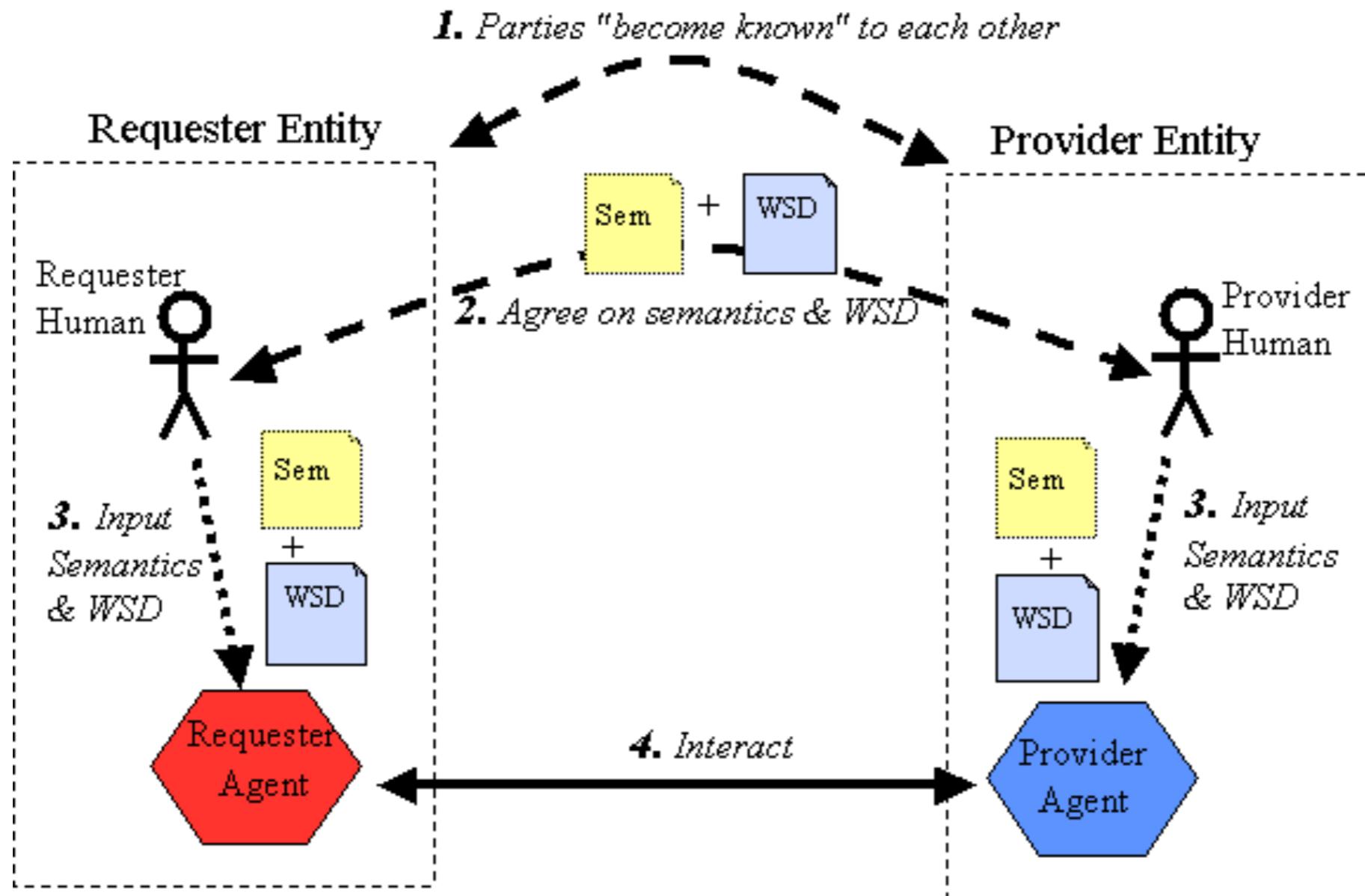


# WebService



บริษัท สยามชนาญกิจ จำกัด และเพื่อนพ้องน้องพี่

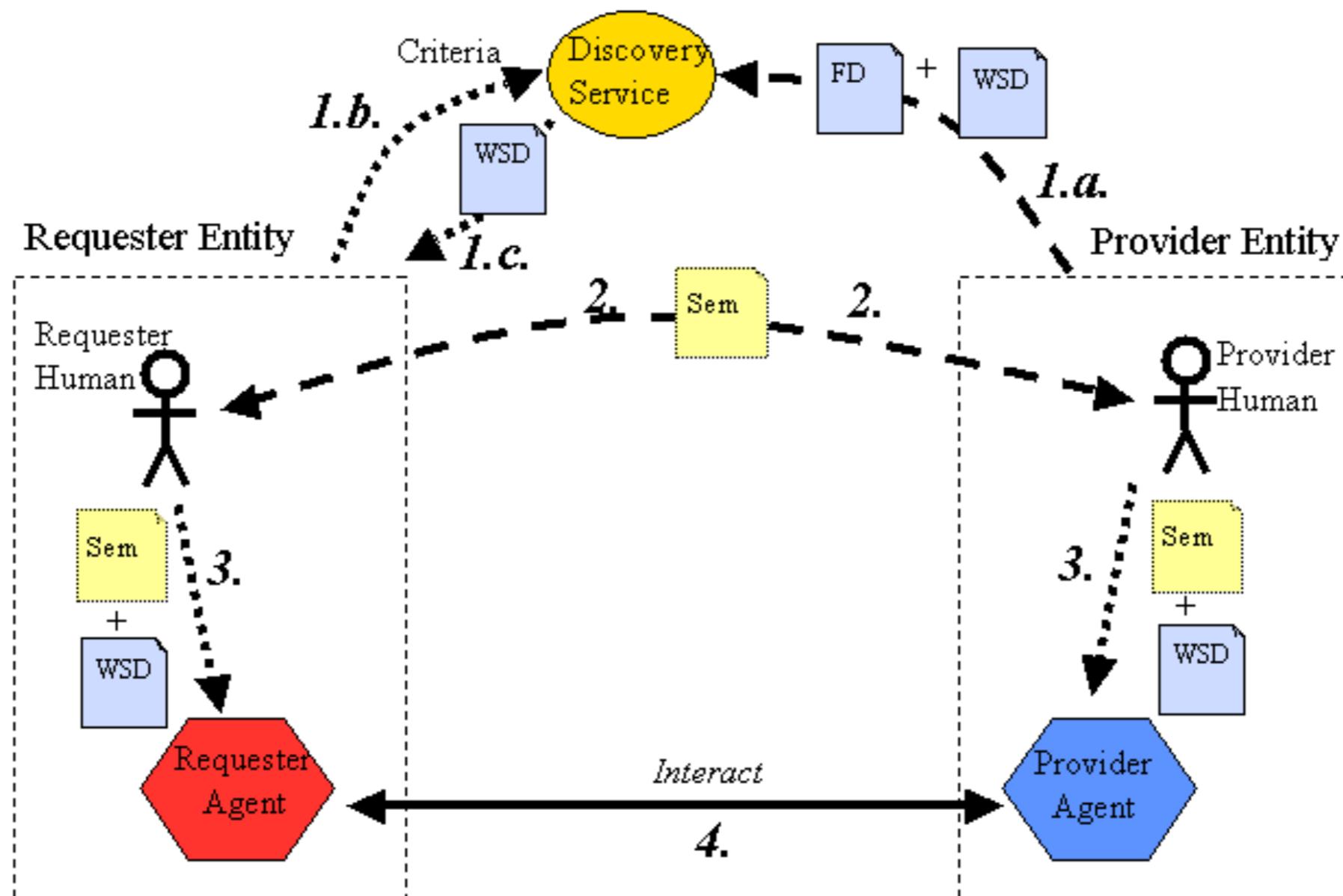
# WebService



<https://www.w3.org/TR/ws-arch>



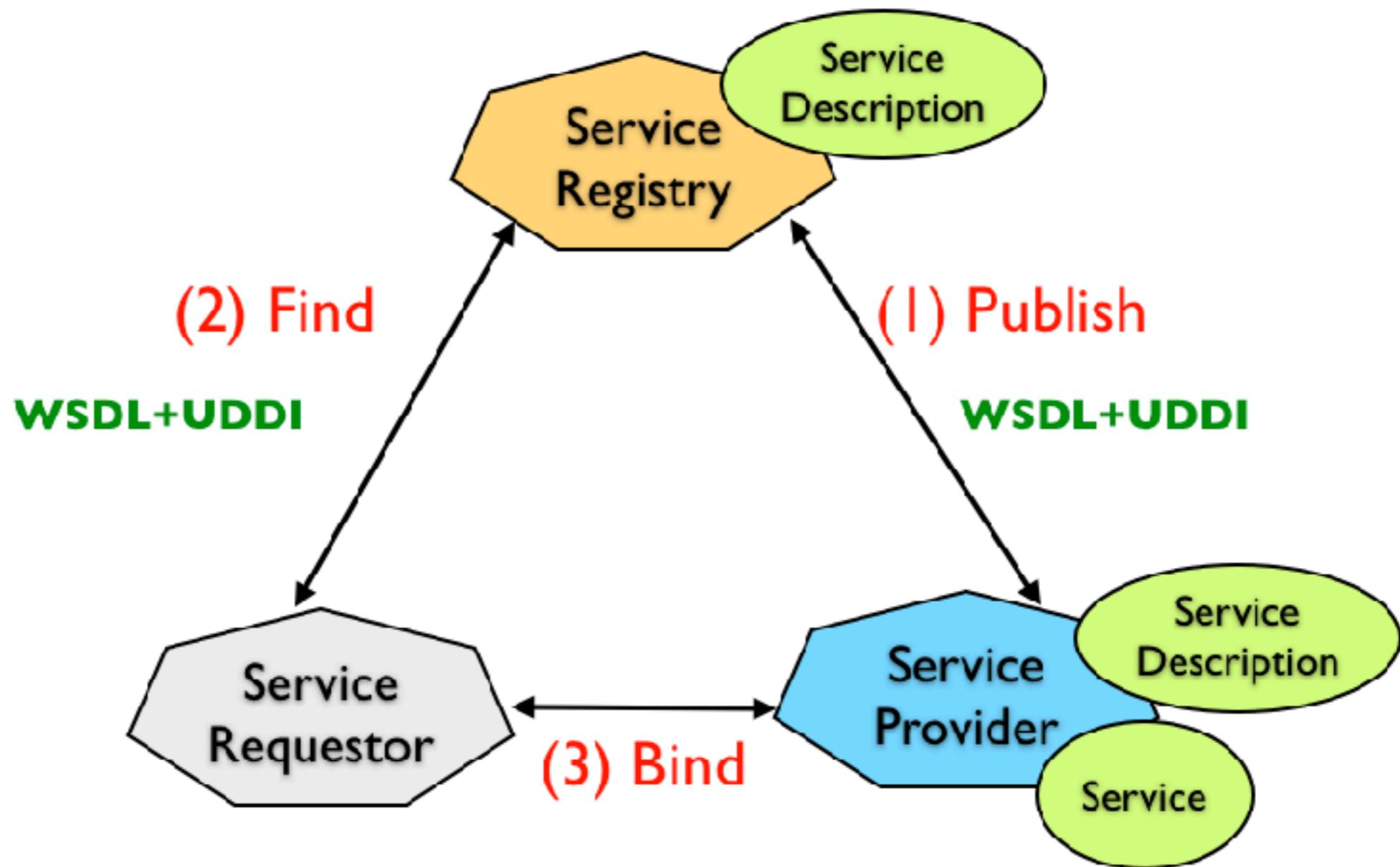
# Service Discovery



<https://www.w3.org/TR/ws-arch>



# WebService



# WebService

WSDL  
UDDI  
SOAP



# **WebService**

**WebService Description Language**

**Universal Descriptor Discovery Integration**

**Simple Object Access Protocol**



# **REST API**

# **REpresentational State Transfer**



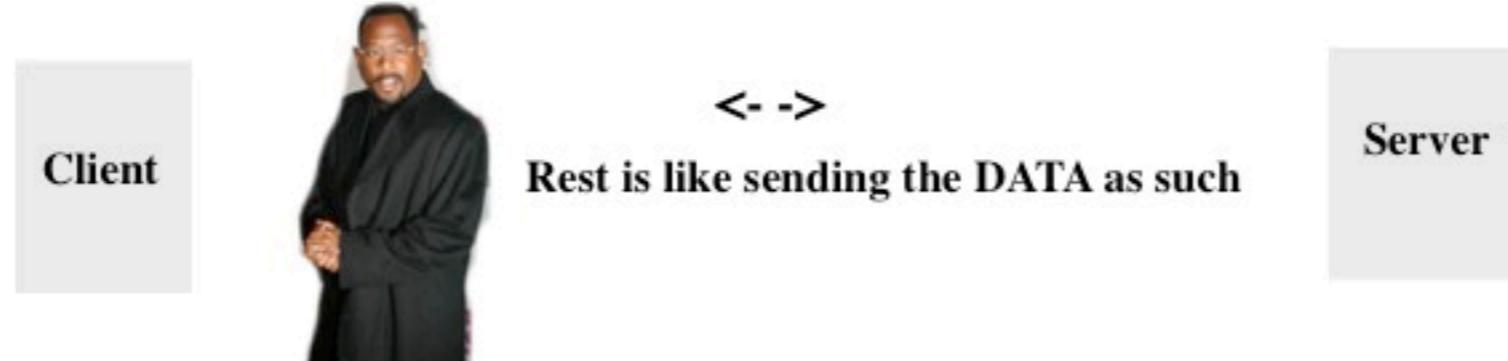
# REST

Consider "Martin Lawrence" as your data

## SOAP



## REST



# REST



Rides directly on HTTP. Plain and simple. In reality, this is all you need to send data from point A to point B and get the required response. Catch: Until something that represents a service contract is put in to place, it's kinda "anything goes".

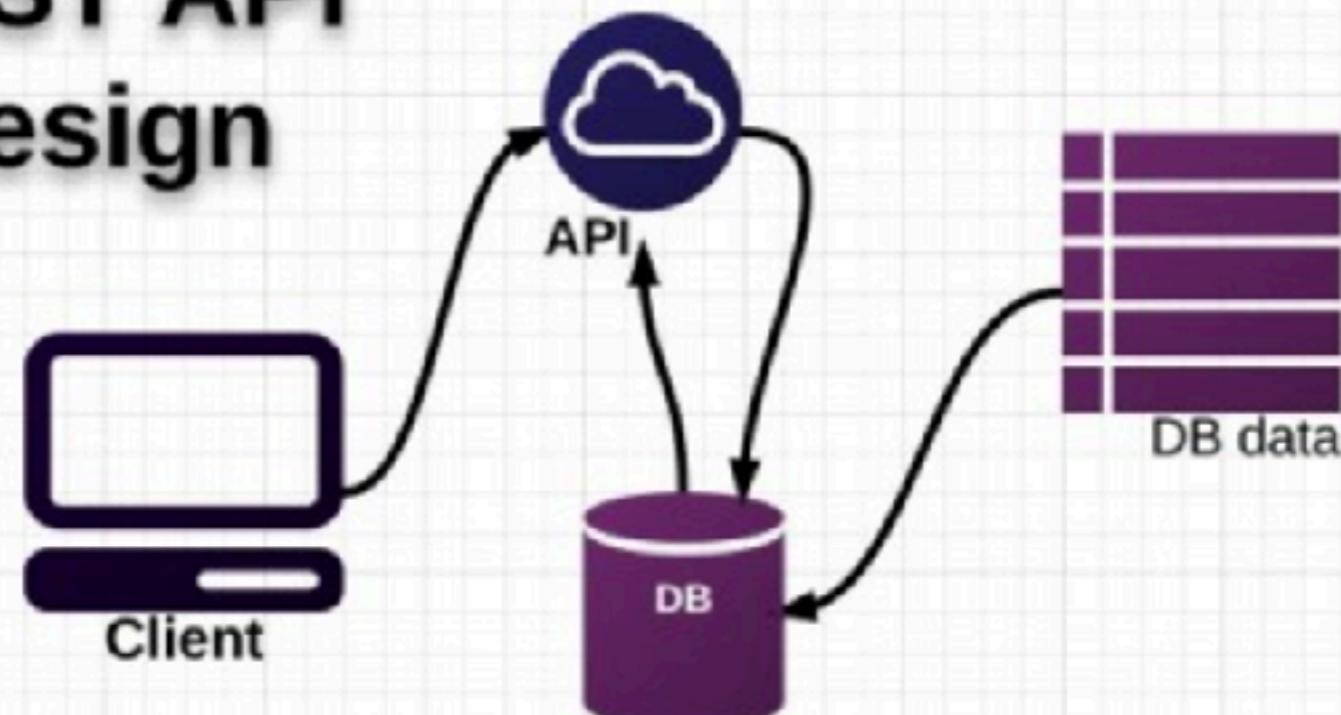


The coach is your SOAP envelope: it wraps your data. Main strength is the presence of a contract: the WSDL. Gives you the "comfort" of easily generating artifacts. Catch: look at the complexity and added weight.

# REST

## REST API Design

**GET** /tasks - display all tasks  
**POST** /tasks - create a new task  
**GET** /tasks/{id} - display a task by ID  
**PUT** /tasks/{id} - update a task by ID  
**DELETE** /tasks/{id} - delete a task by ID



**Monolith**

**N-tier**

**SOA/  
WebService**

**REST API**



# Microservice

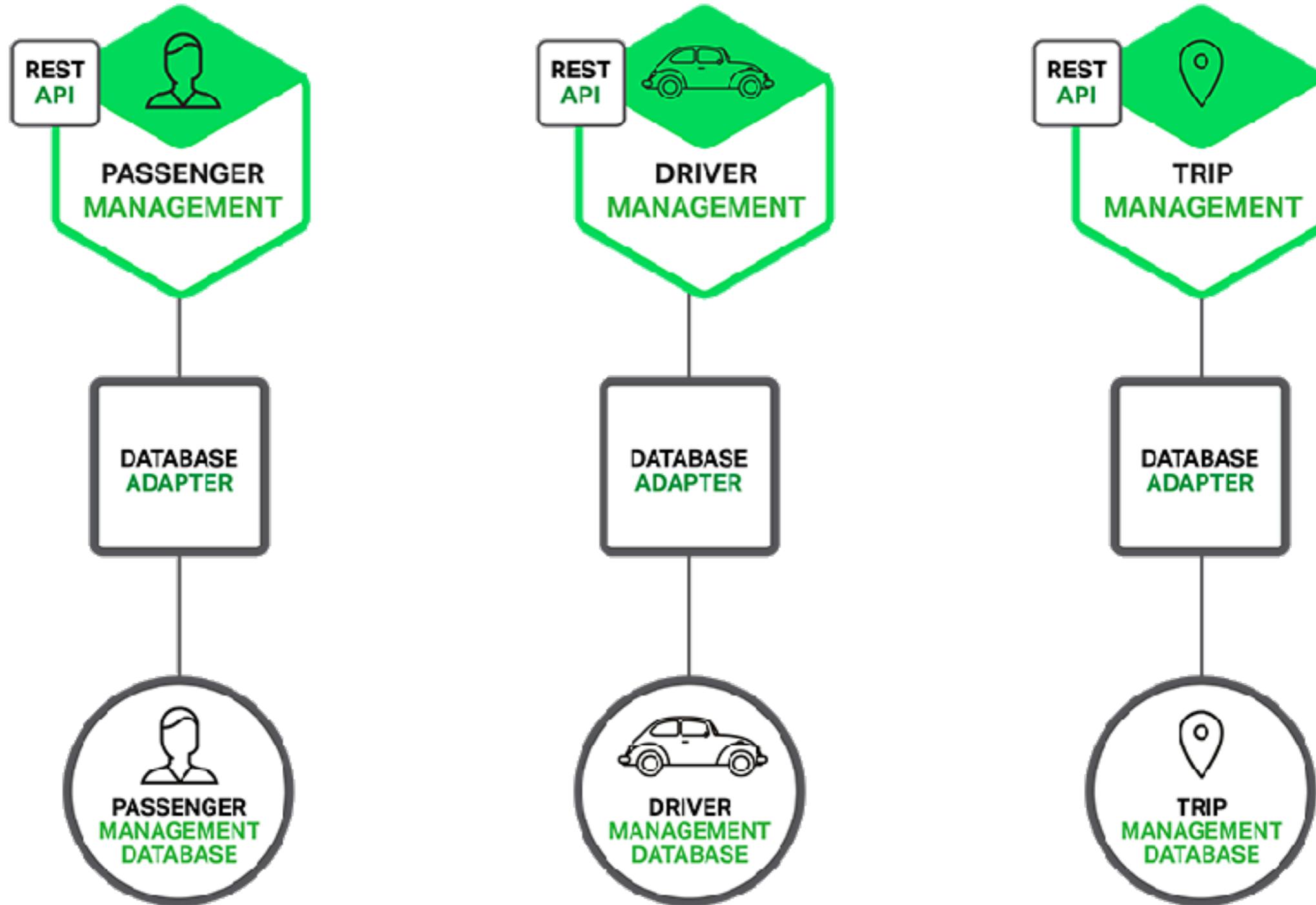


# **Microservice**

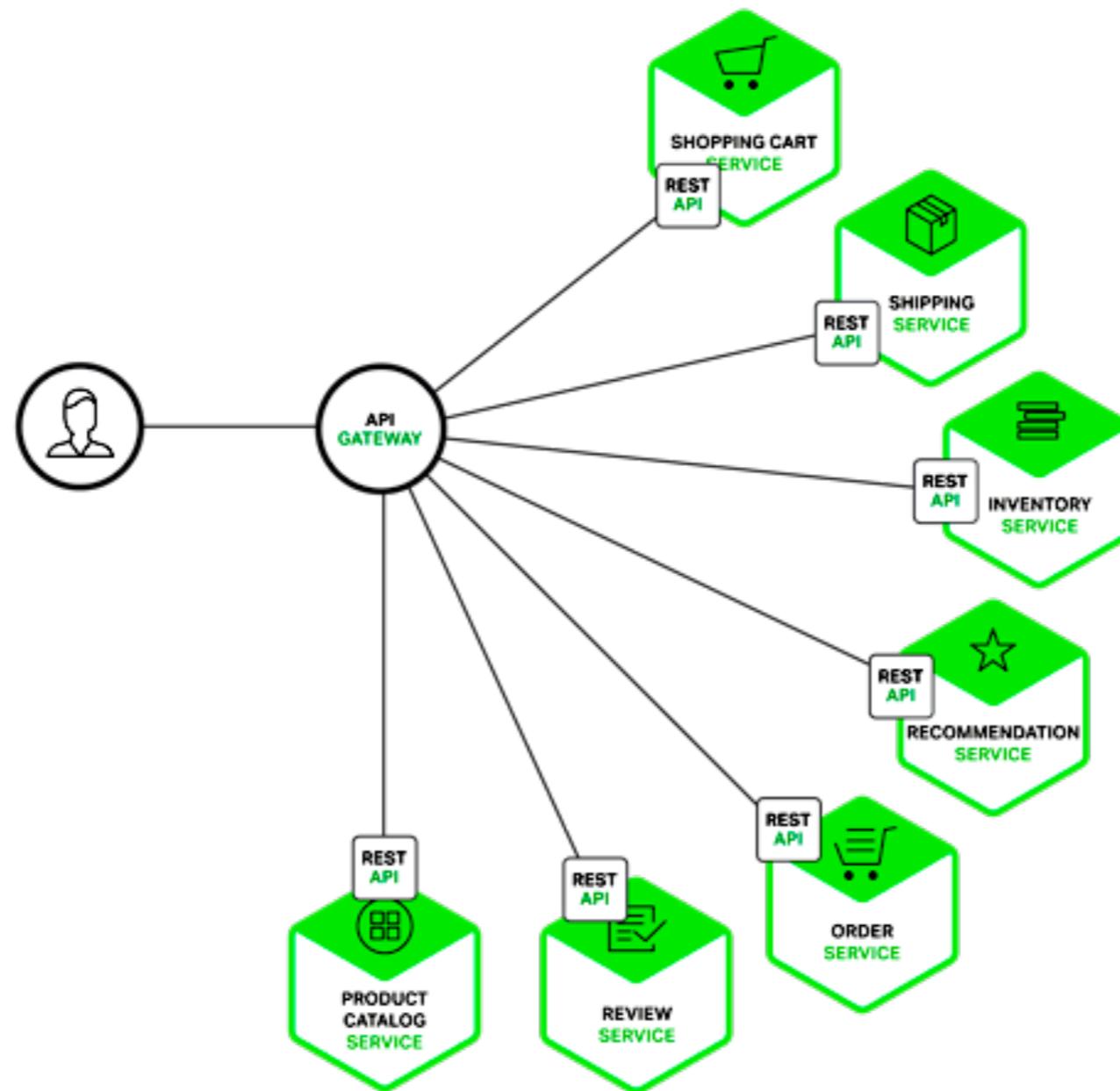
## **not a silver bullet**



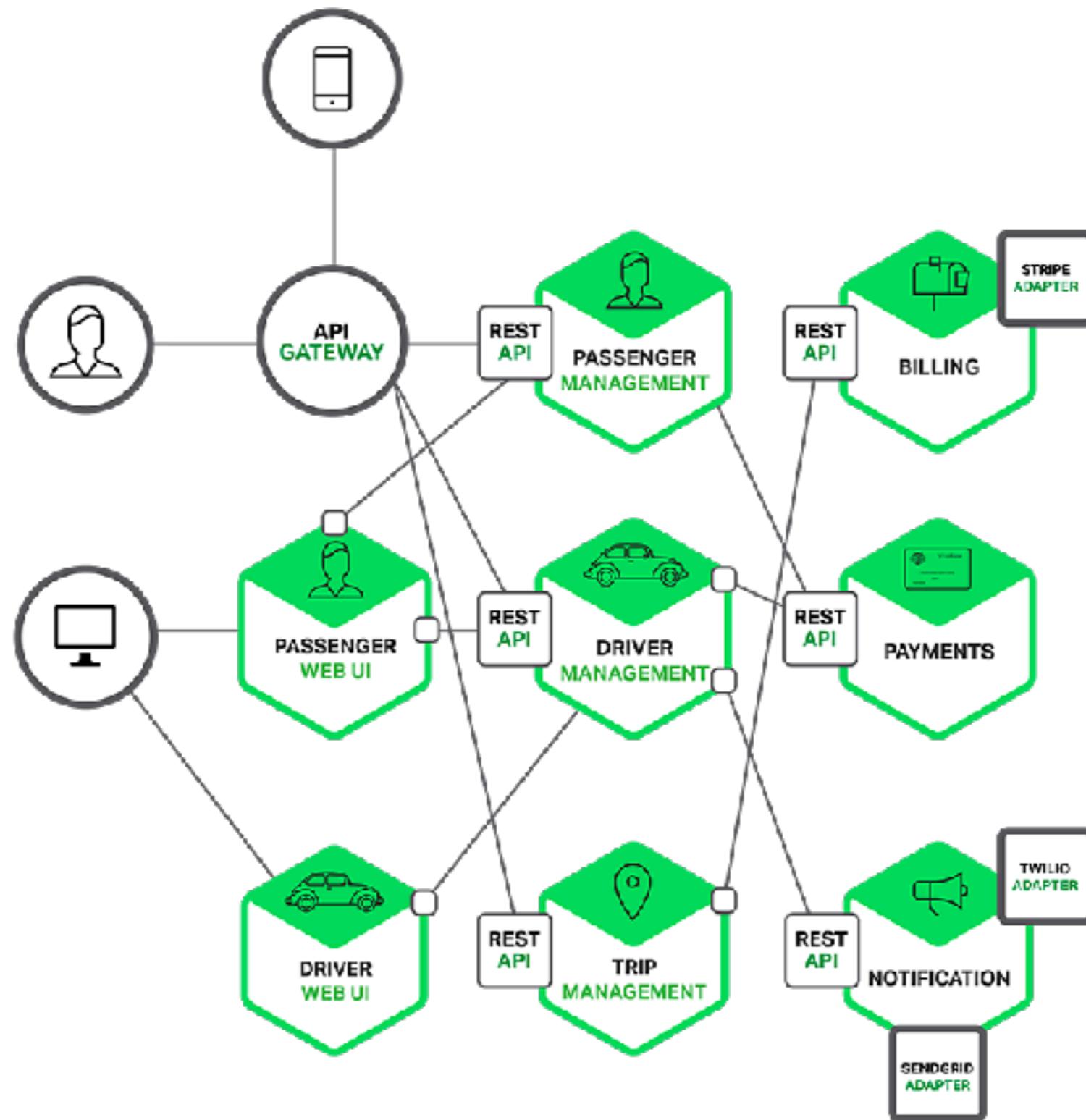
# Microservice



# Microservice



# Microservice



# Pros & Cons

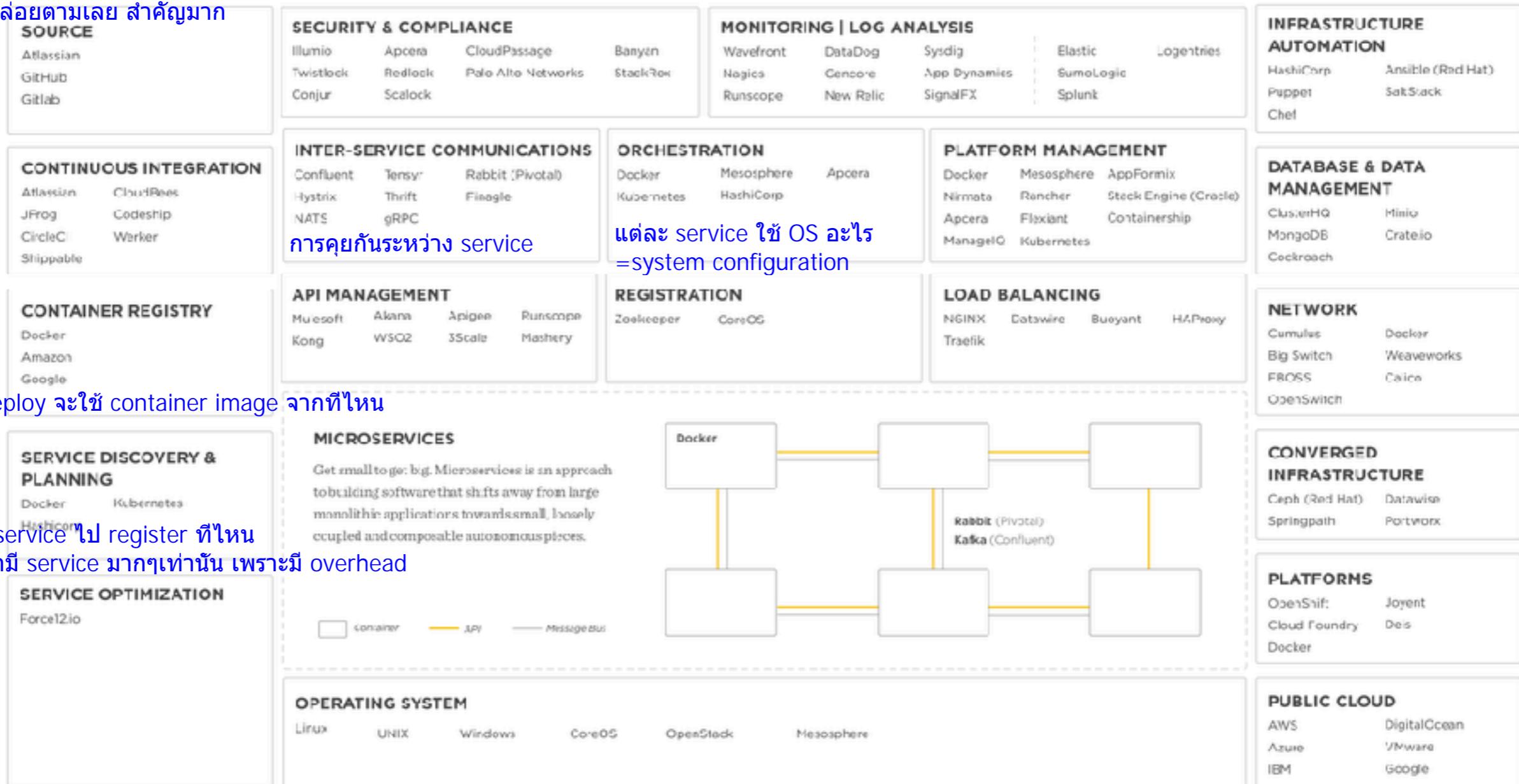
?



# Microservice Ecosystem

การทำ Microservices ต้องจัดการเรื่องอะไรบ้าง มีหลายเรื่องมาก

ทดลองว่าจะสร้าง branch ได้เมื่อไร  
แล้วจะ merge กันเมื่อไร  
อย่างล้อยตามเลย สำคัญมาก



Includes both companies and open source projects

Version 12 | GL19.16

<https://twitter.com/mcmiller00/status/690894999370633216>



บริษัท สยามชำนาญกิจ จำกัด และเพื่อนพ้องน้องพี่

# Serverless

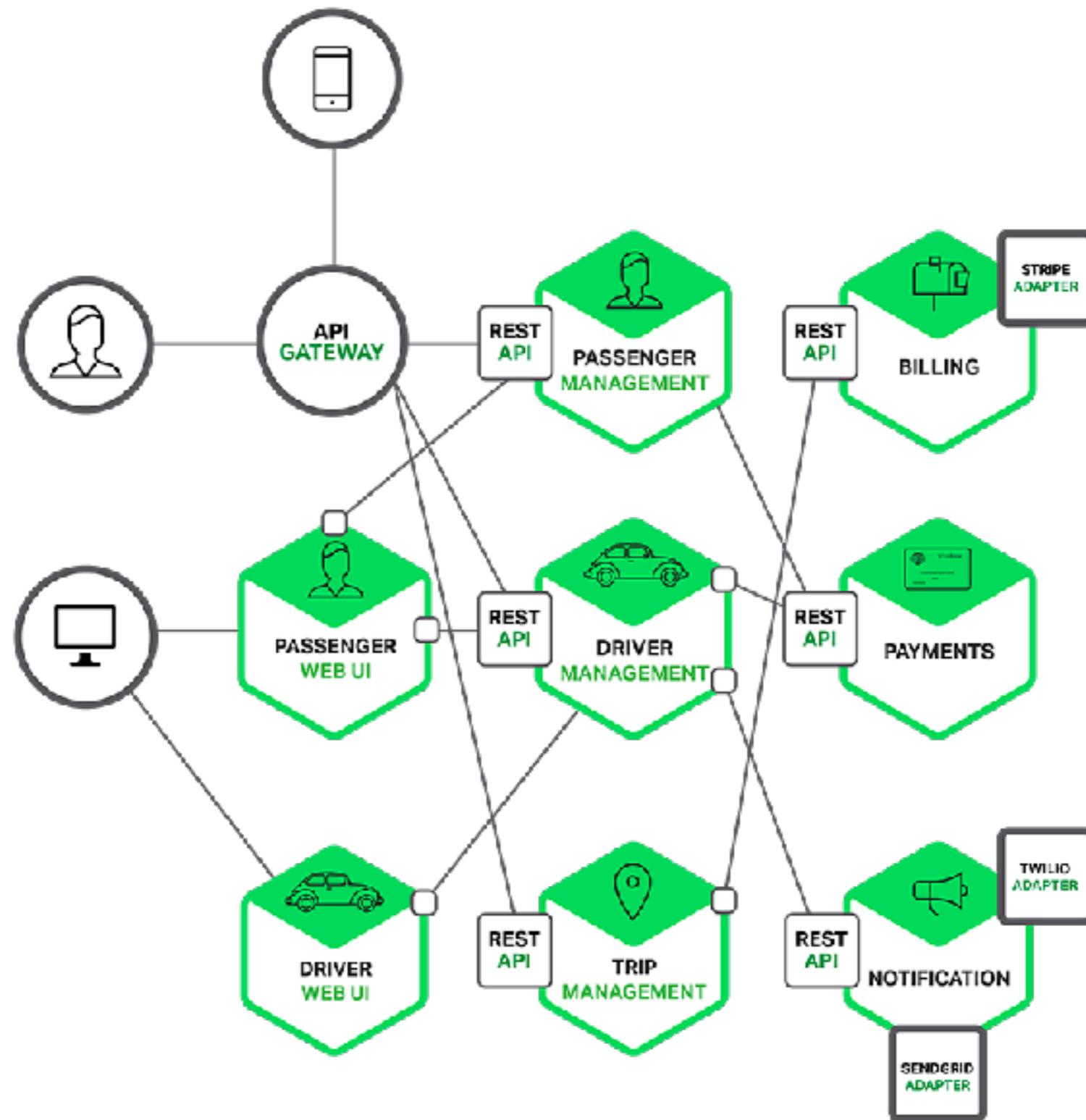


# FaaS

# Function as a Service



# Microservice



# Design Microservice ?



# Design for failure

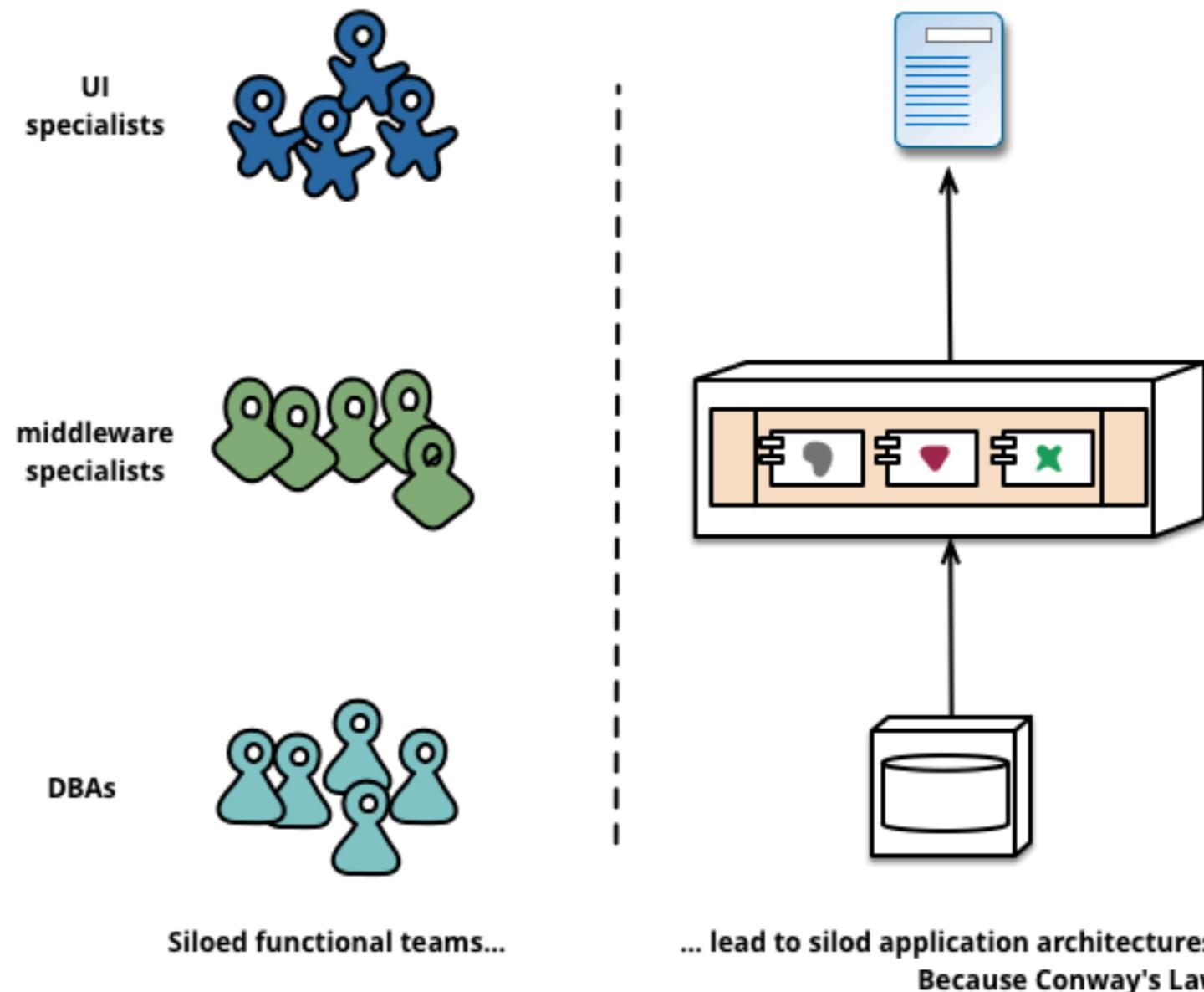


# **Evolutionary design**



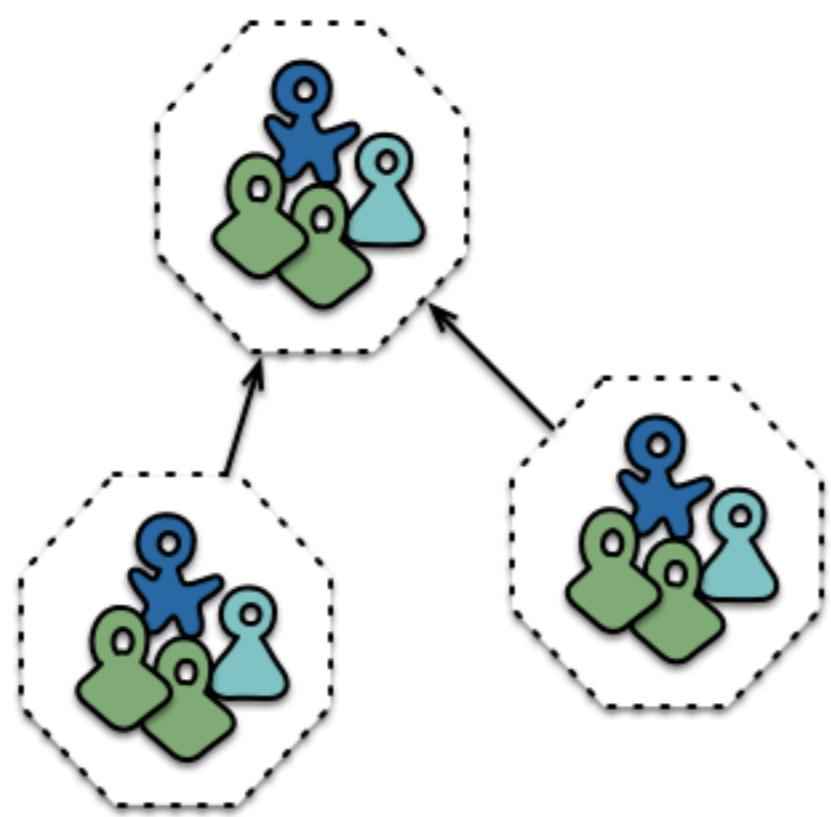
# Implementation ?



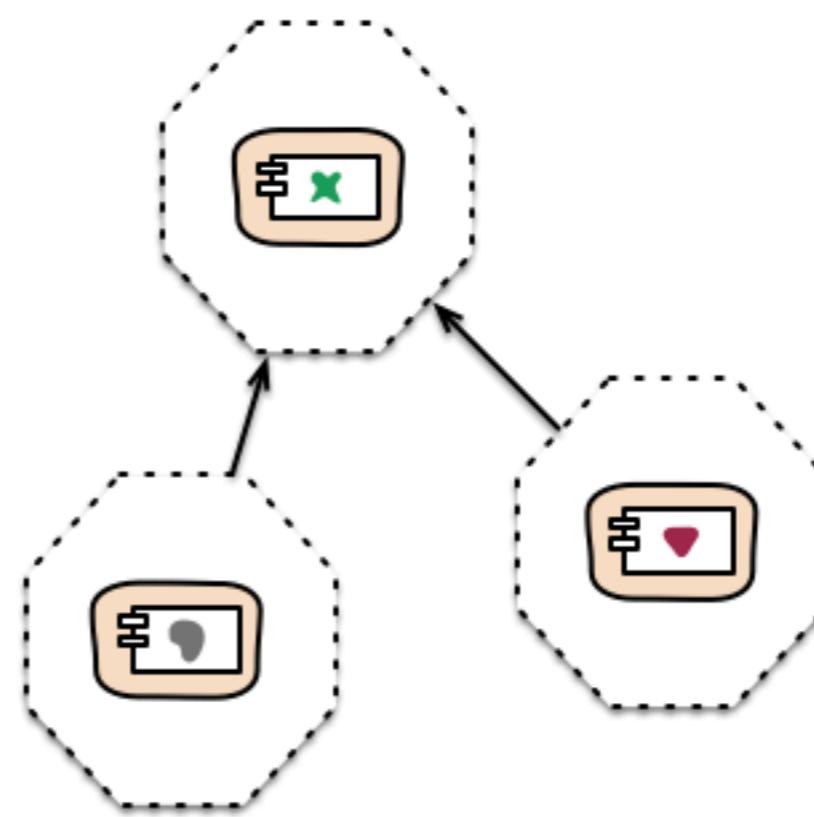


<https://martinfowler.com/articles/microservices.html>





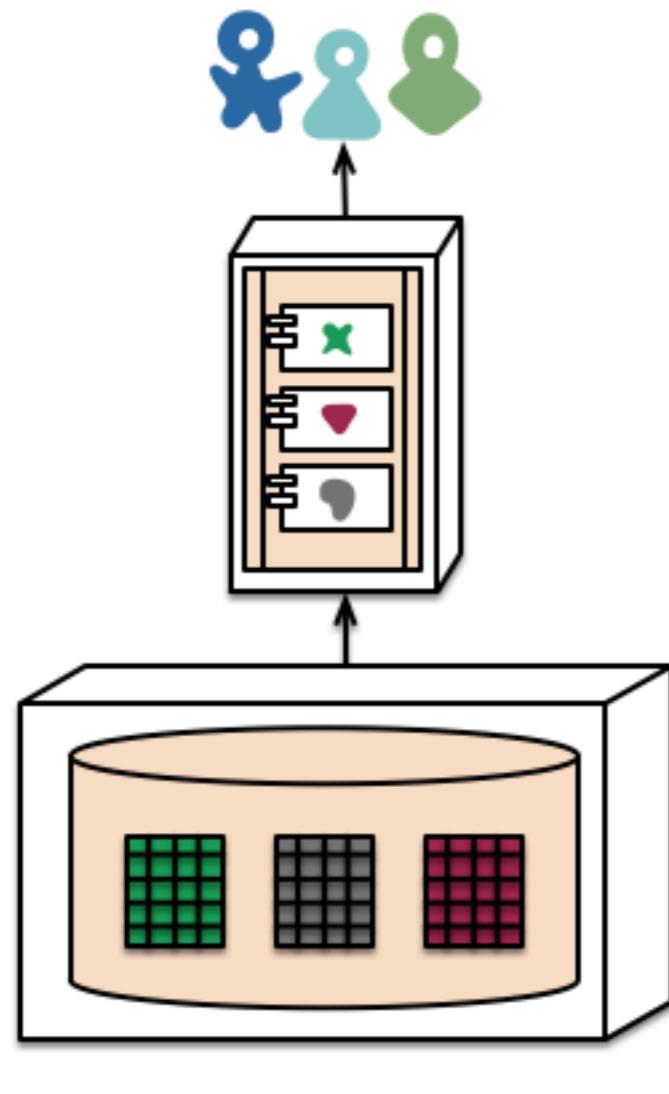
Cross-functional teams...



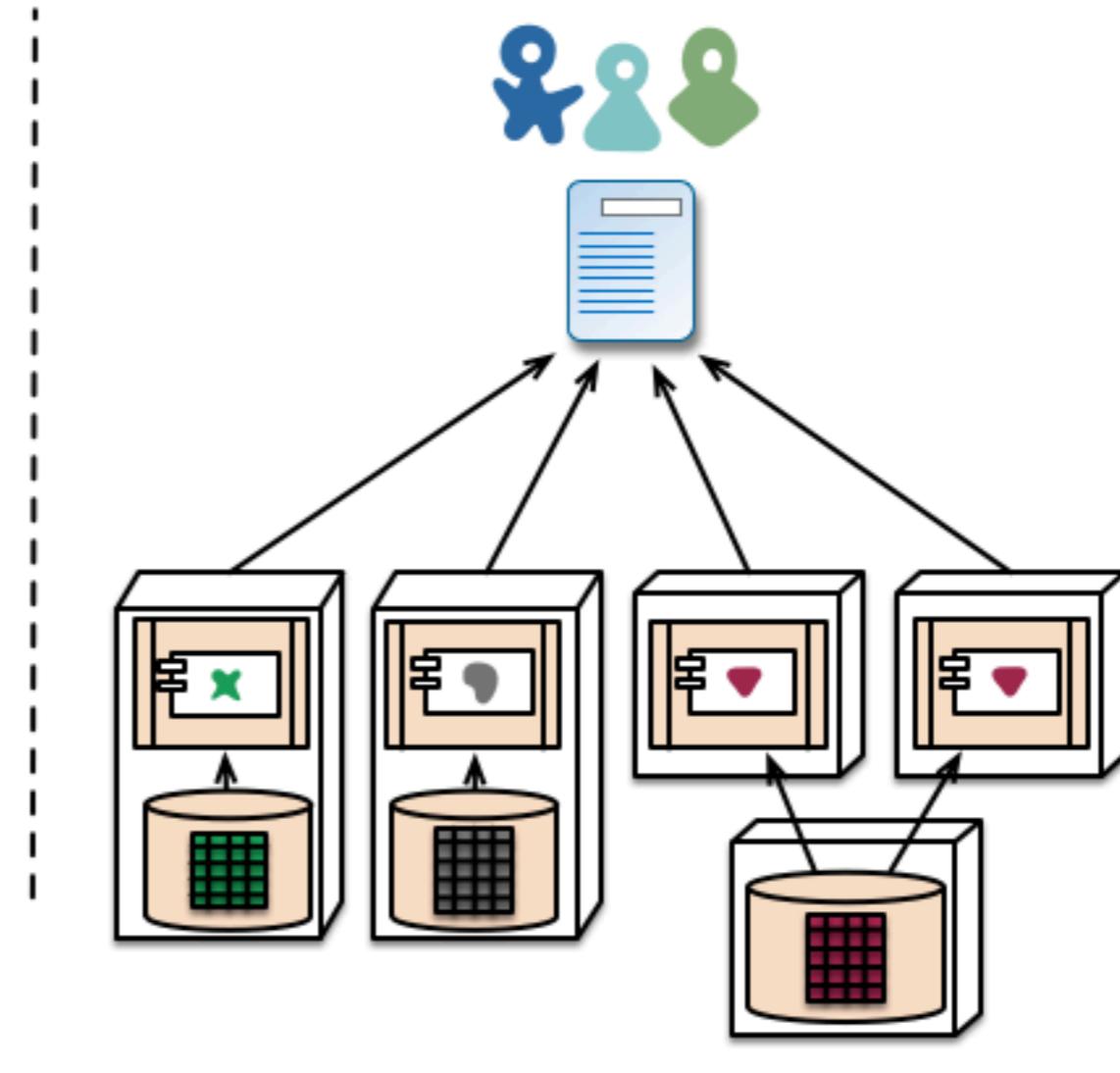
... organised around capabilities  
Because Conway's Law

<https://martinfowler.com/articles/microservices.html>





monolith - single database



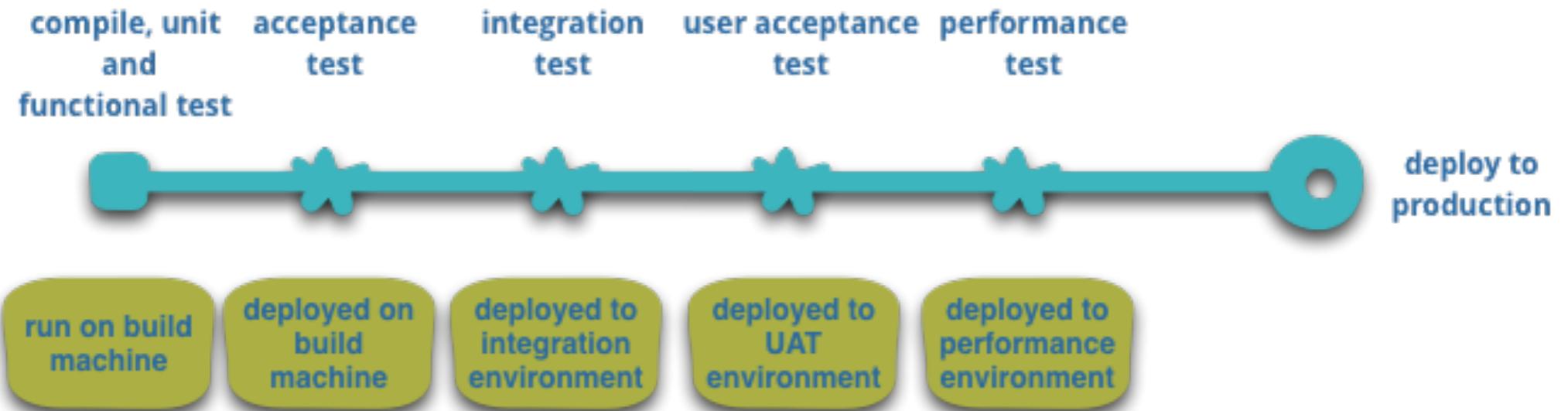
microservices - application databases

<https://martinfowler.com/articles/microservices.html>



# Deployment ?

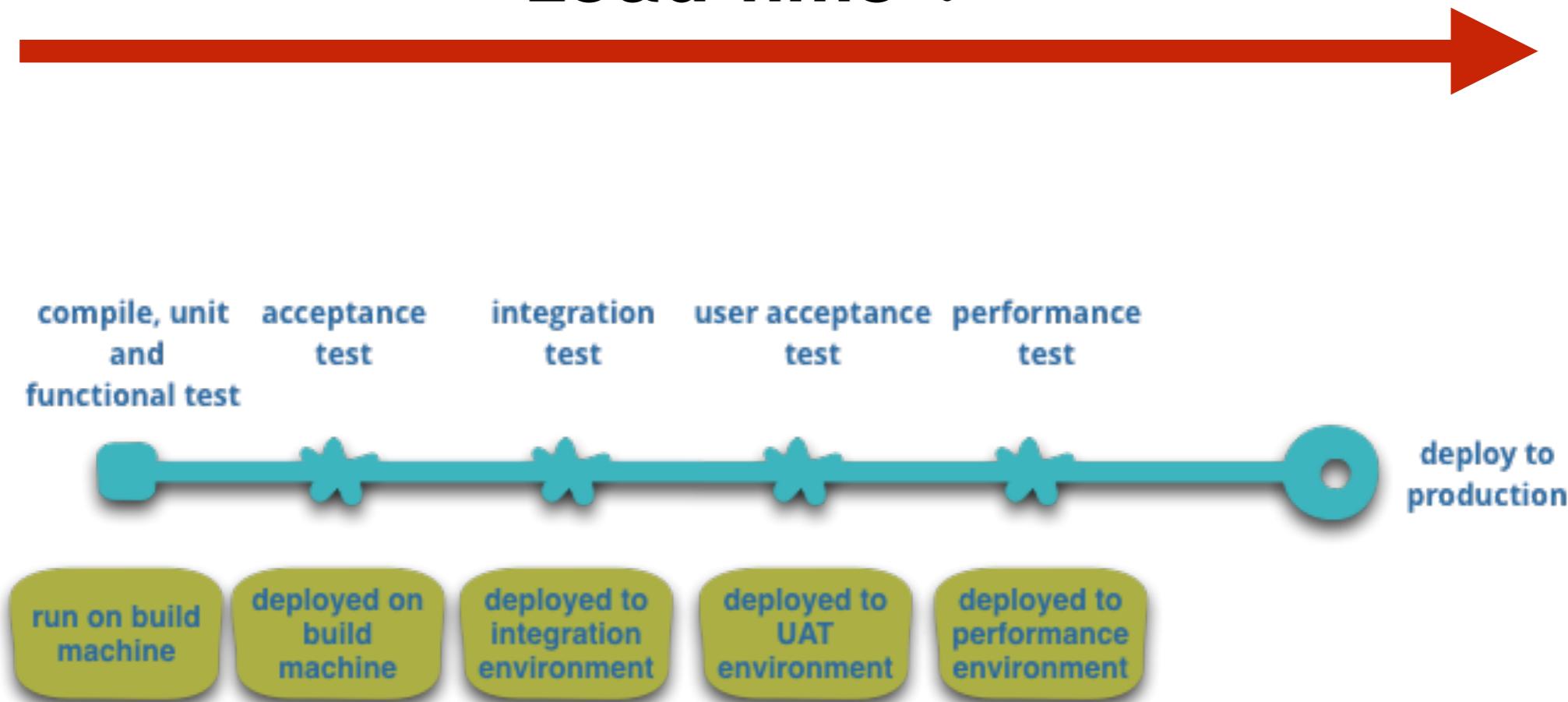


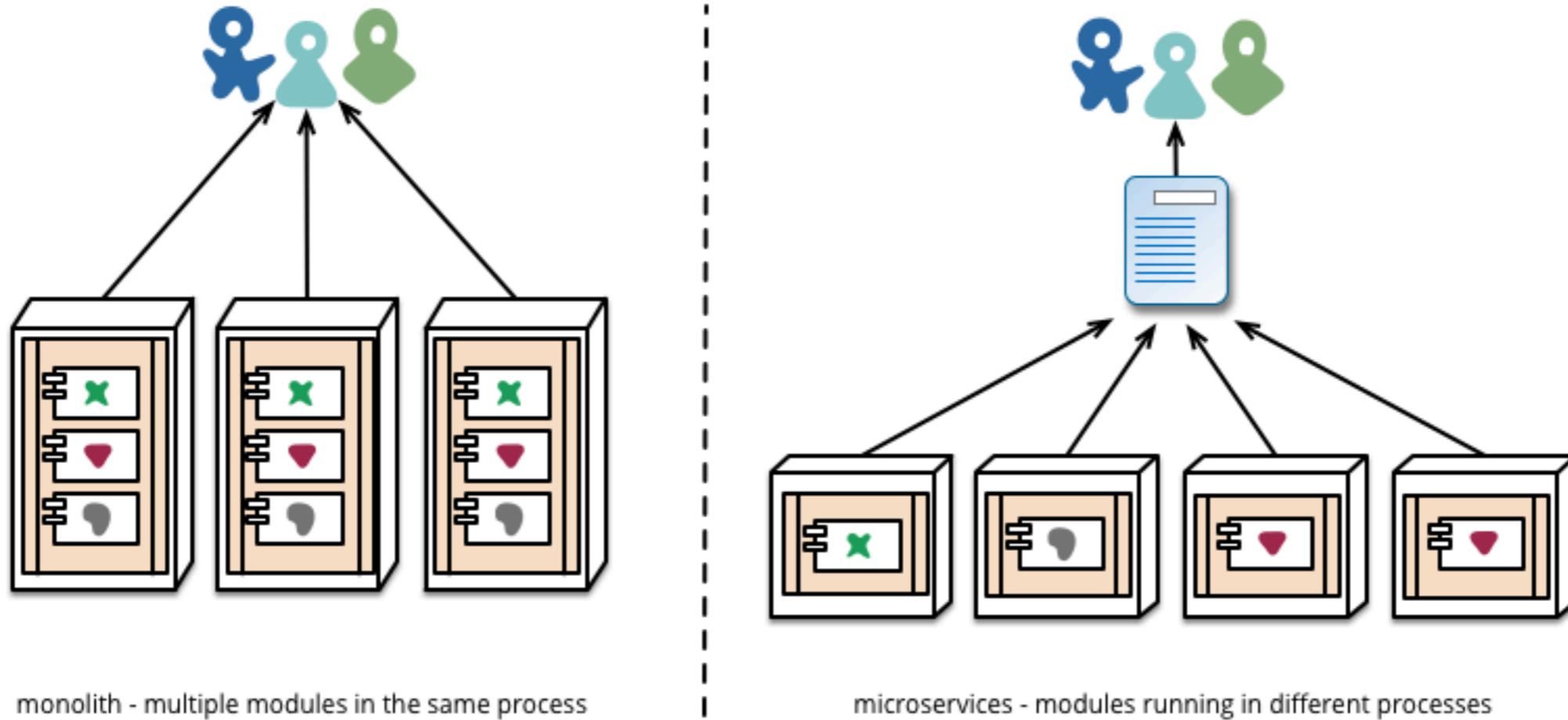


<https://martinfowler.com/articles/microservices.html>



# Lead time ?

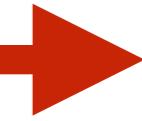




<https://martinfowler.com/articles/microservices.html>

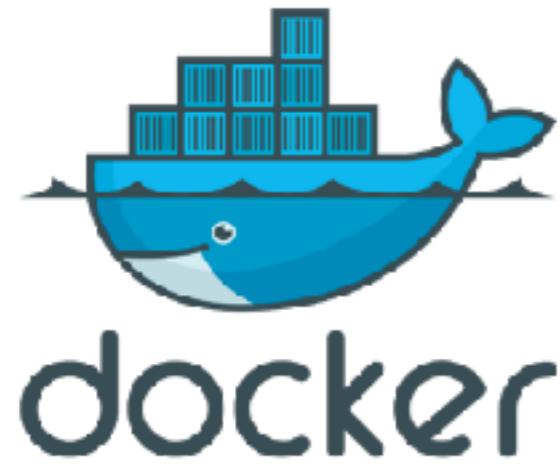
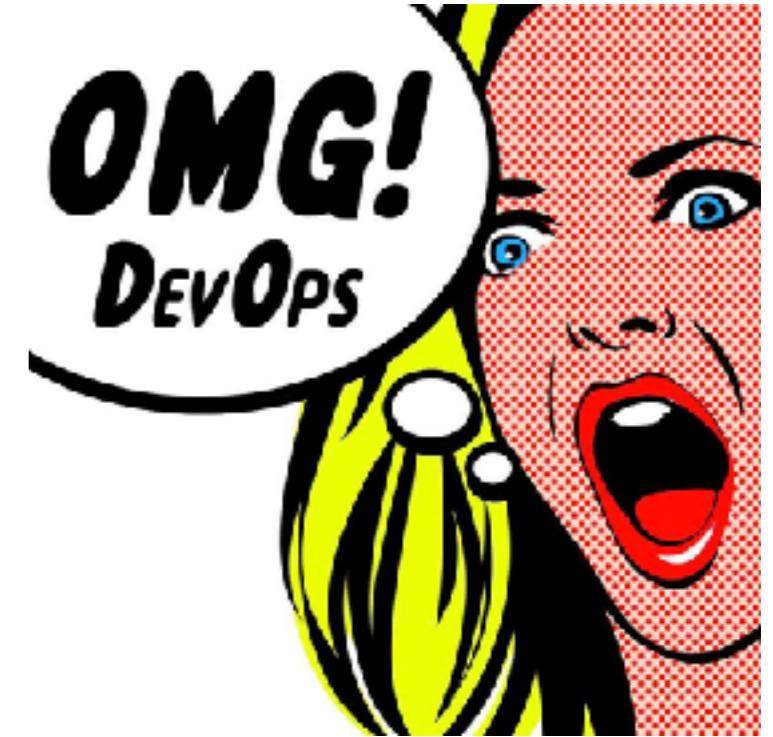


# Reduce Lead time ?



# Containerization ?





kubernetes



MESOS

# Monitoring ?



# Visualization ?



# Tracing ?



# **Alert and Notification ?**



# Performance ?



# Security ?



# **What's next ?**



บริษัท สยามชนาญกิจ จำกัด และเพื่อนพ้องน้องพี่



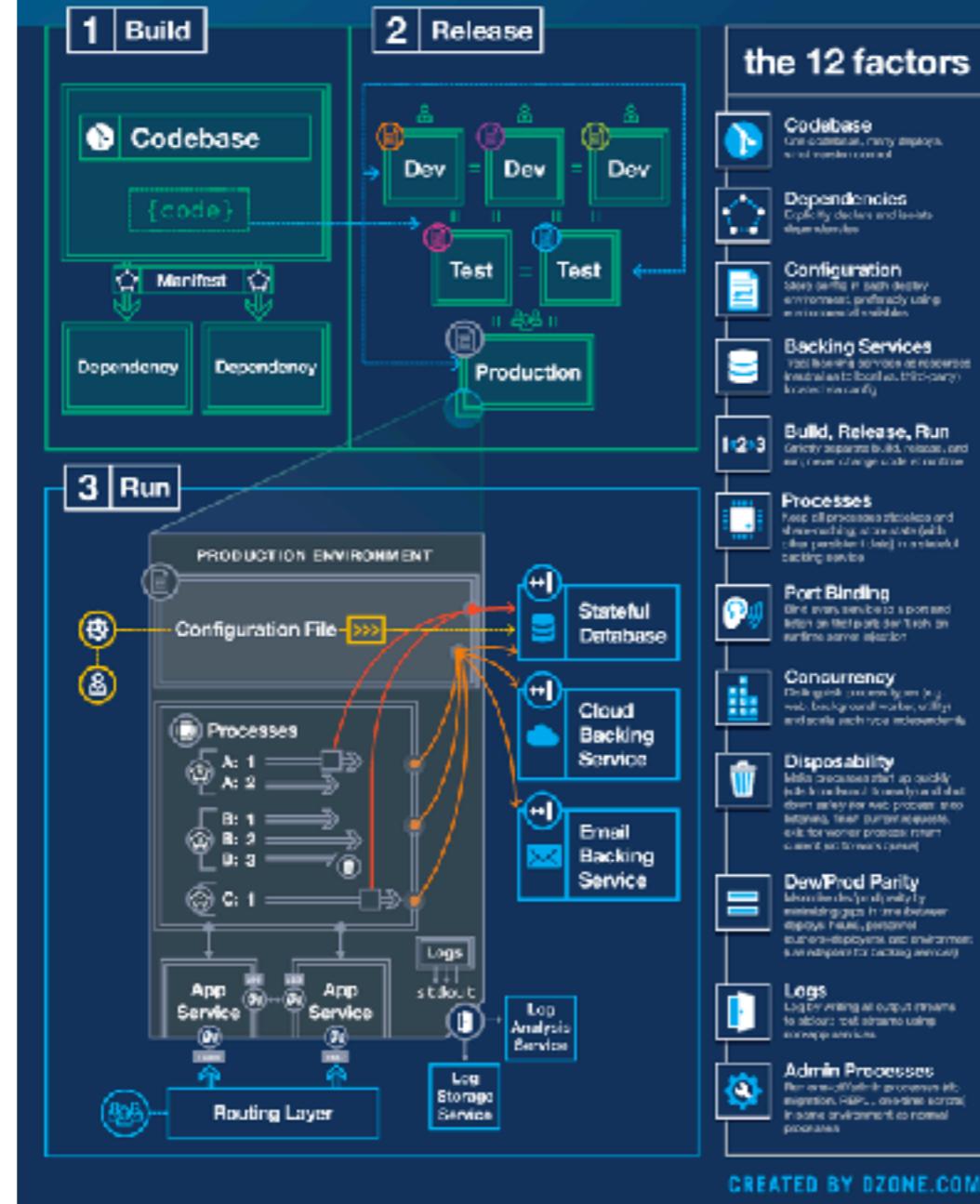
# THE TWELVE-FACTOR APP

<https://12factor.net/>



# The 12-Factor App

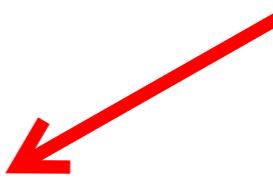
Modern web applications run in heterogeneous environments, scale elastically, update frequently, and depend on independently deployed backing services. Modern application architectures and development practices must be designed accordingly. The PaaS-masters at Heroku summarized lessons learned from building hundreds of cloud-native applications into the twelve factors visualized below.



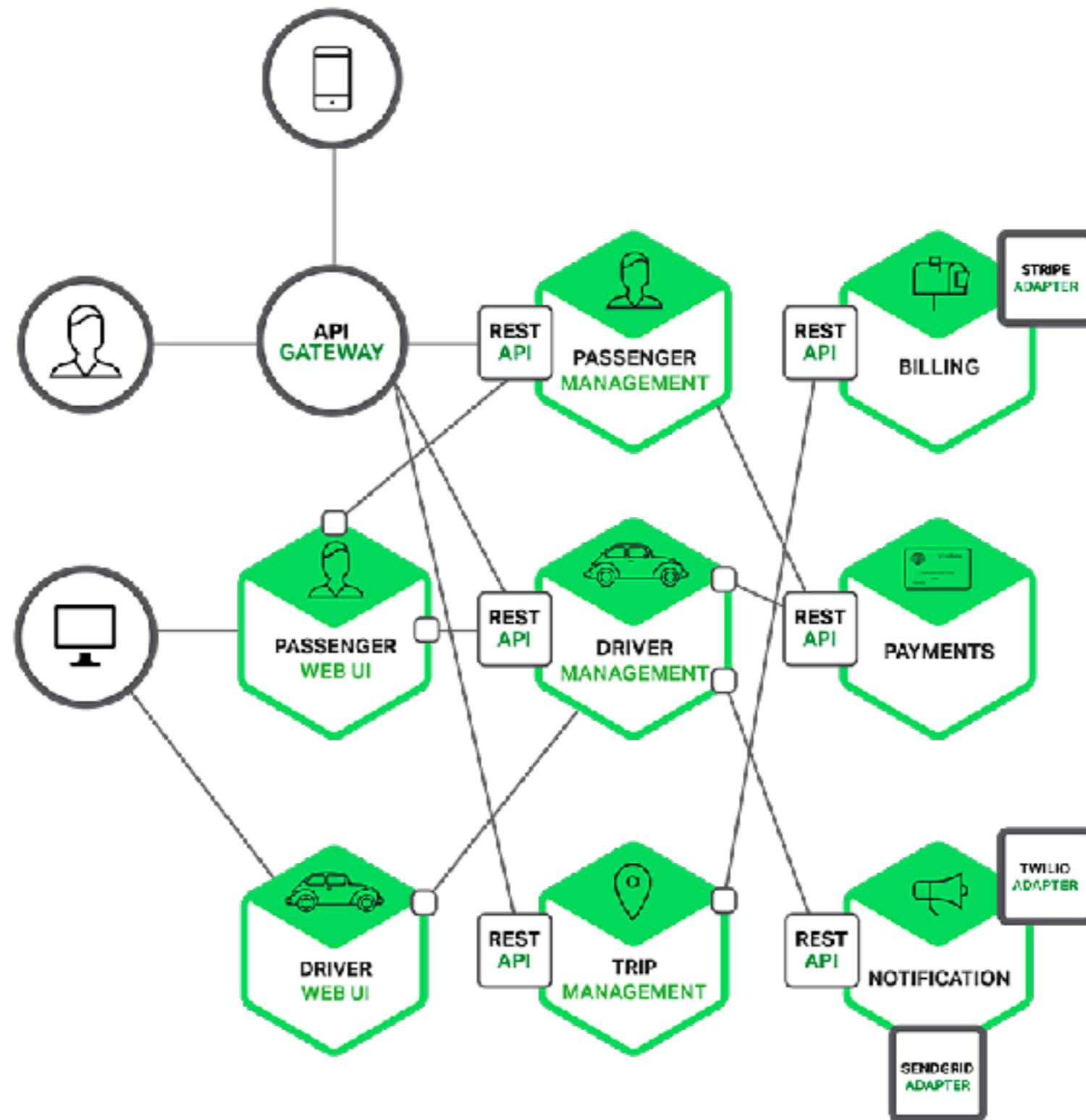
# Testing ?

จะทดสอบ microservices จะทำอย่างไร  
ให้ด้วยมาก

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

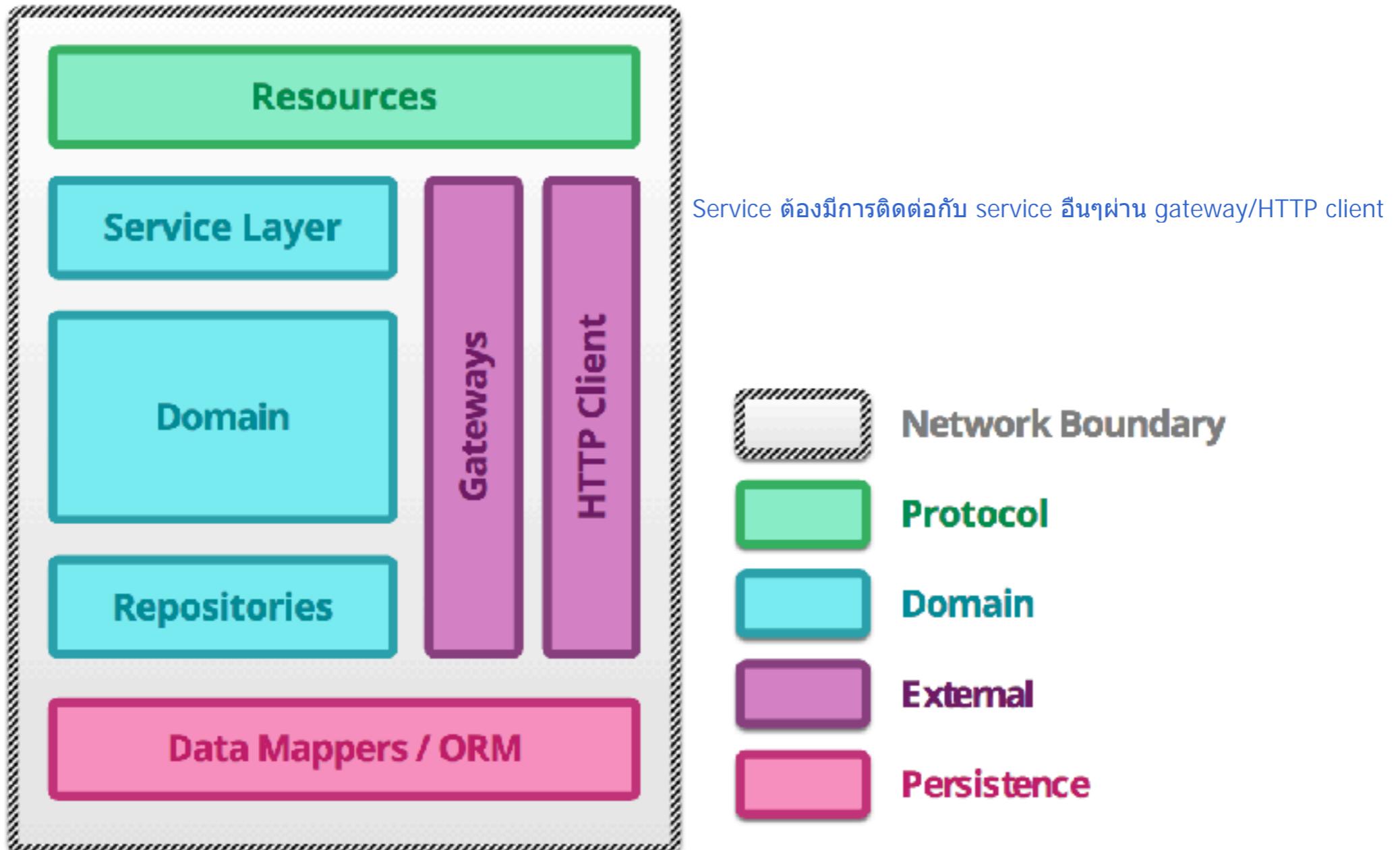


# Microservice

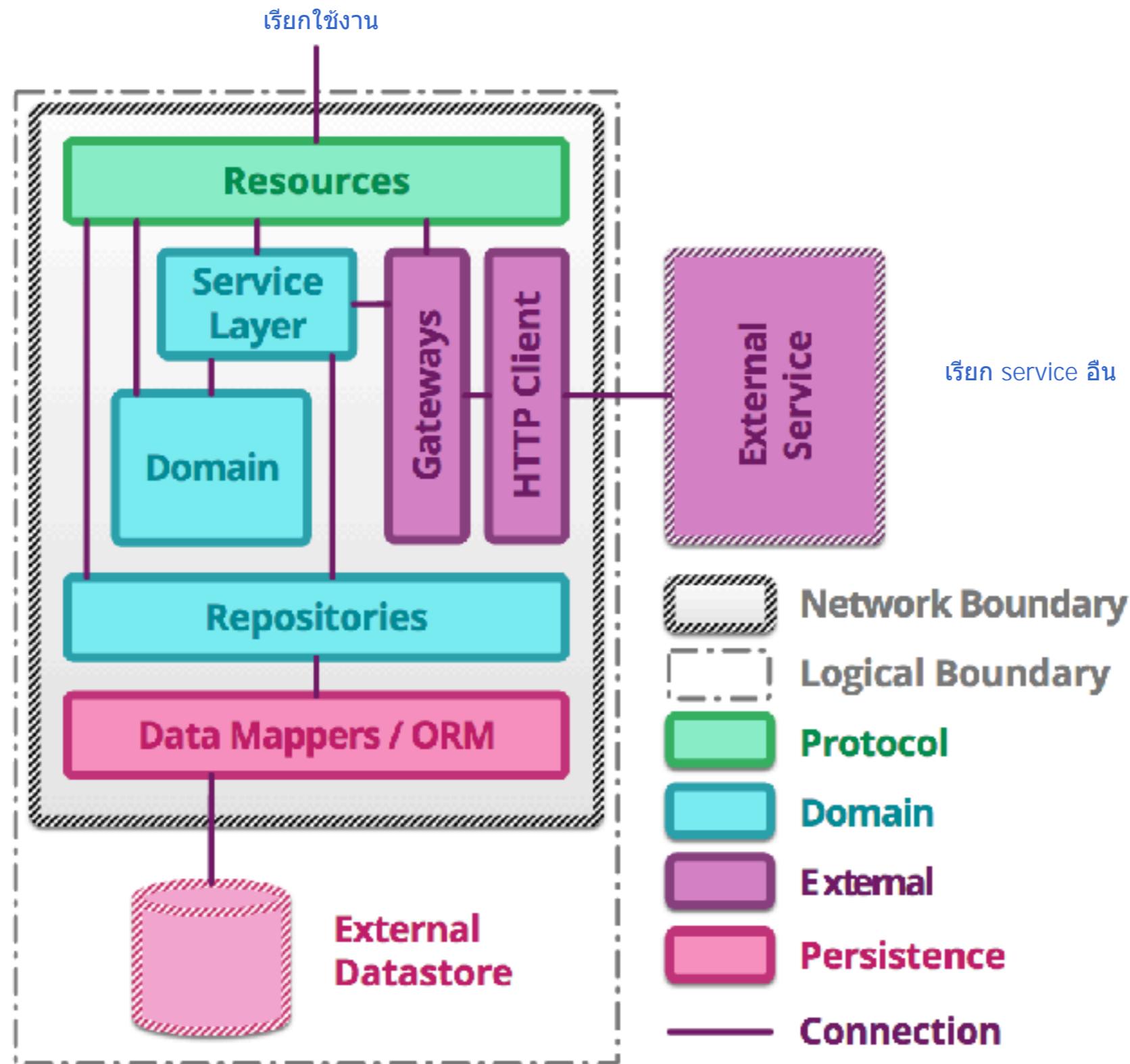


# Microservice

1. มี resource อะไรบ้าง
2. ใช้ domain อะไร
3. ใช้ data จากที่ไหน (Repositories)



# Microservice

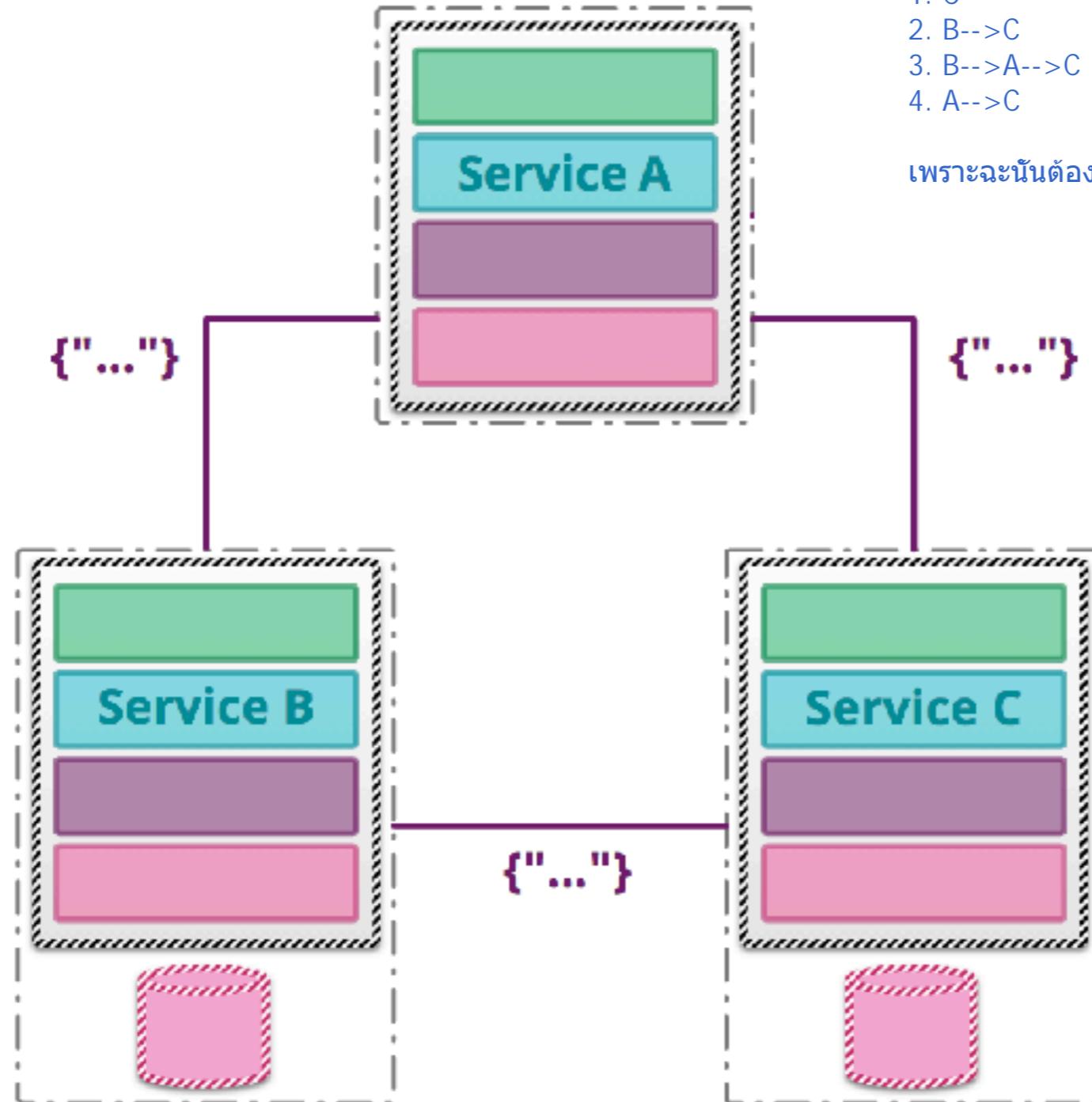


# Microservice(s)

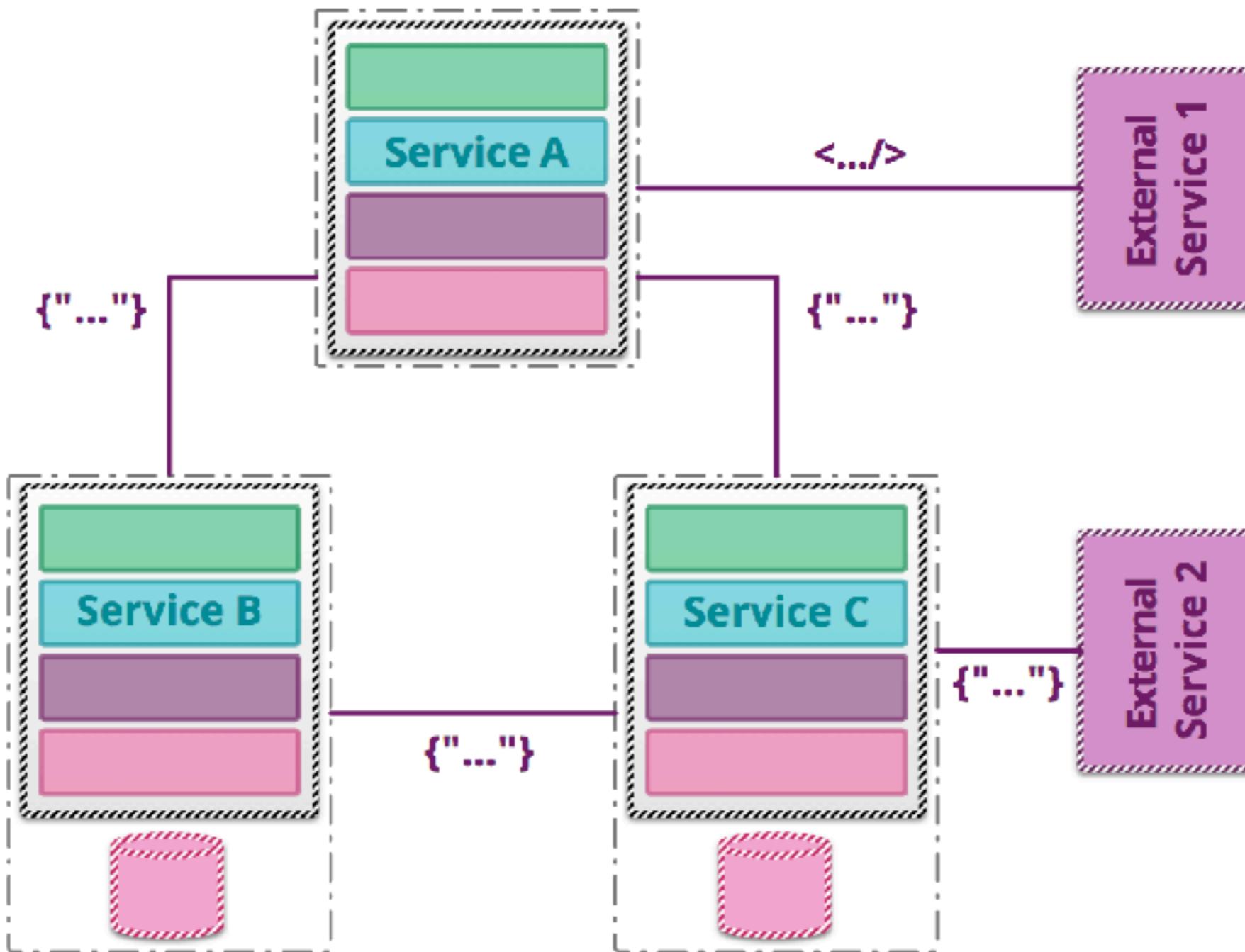
ถ้าแก้ C แล้วต้องทดสอบอะไรบ้าง?

1. C
2. B-->C
3. B-->A-->C
4. A-->C

เพราะฉะนั้นต้องรู้ว่า service ไหนใช้ service ใดบ้าง



# Microservice(s)



# Testing strategies ?



ไม่ต้องทดสอบหมด แต่ยังทดสอบมาก ก็มั่นใจมาก

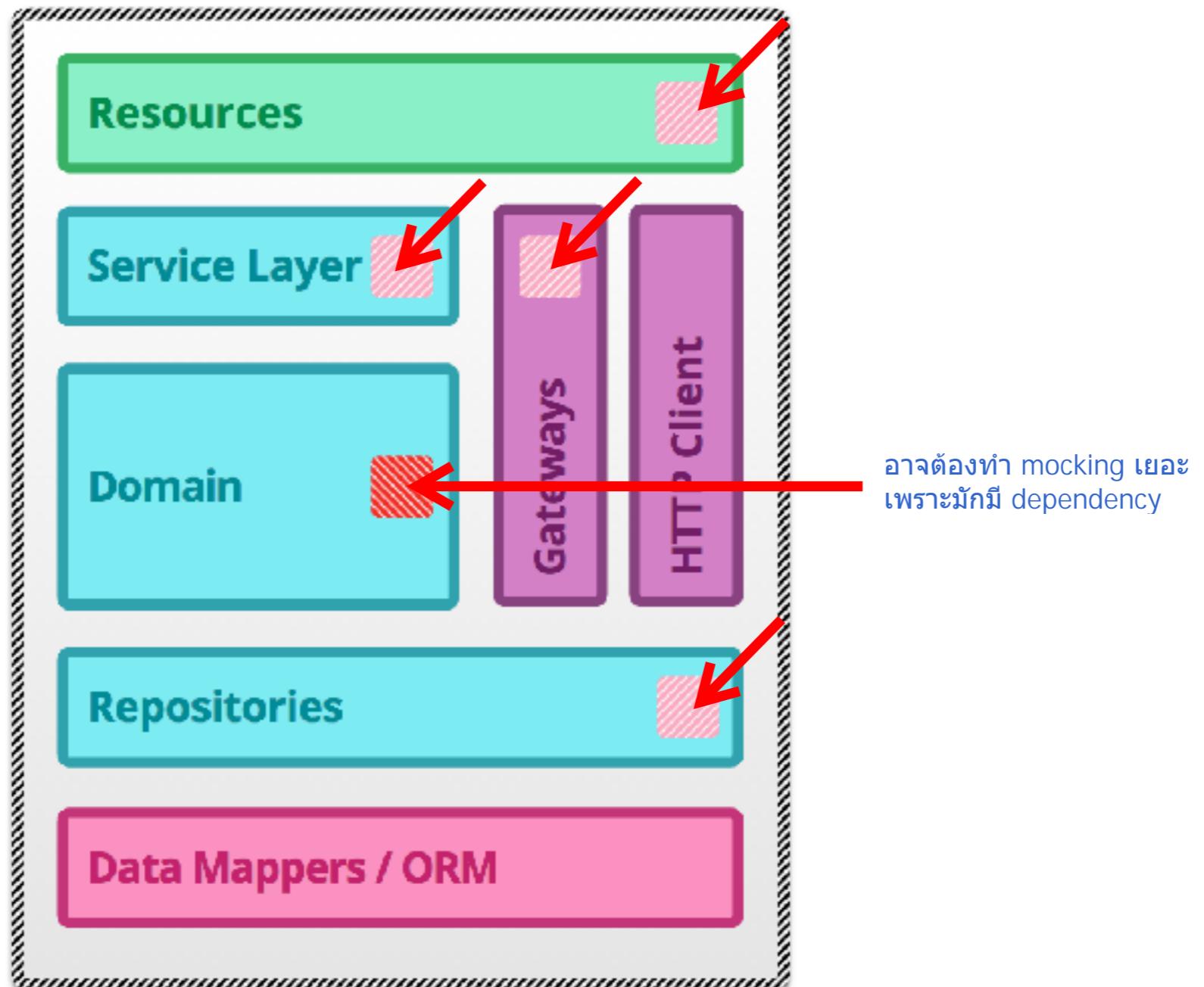
ควรทำ automation ทั้งหมด

# Unit Integration Component Contract End-to-end



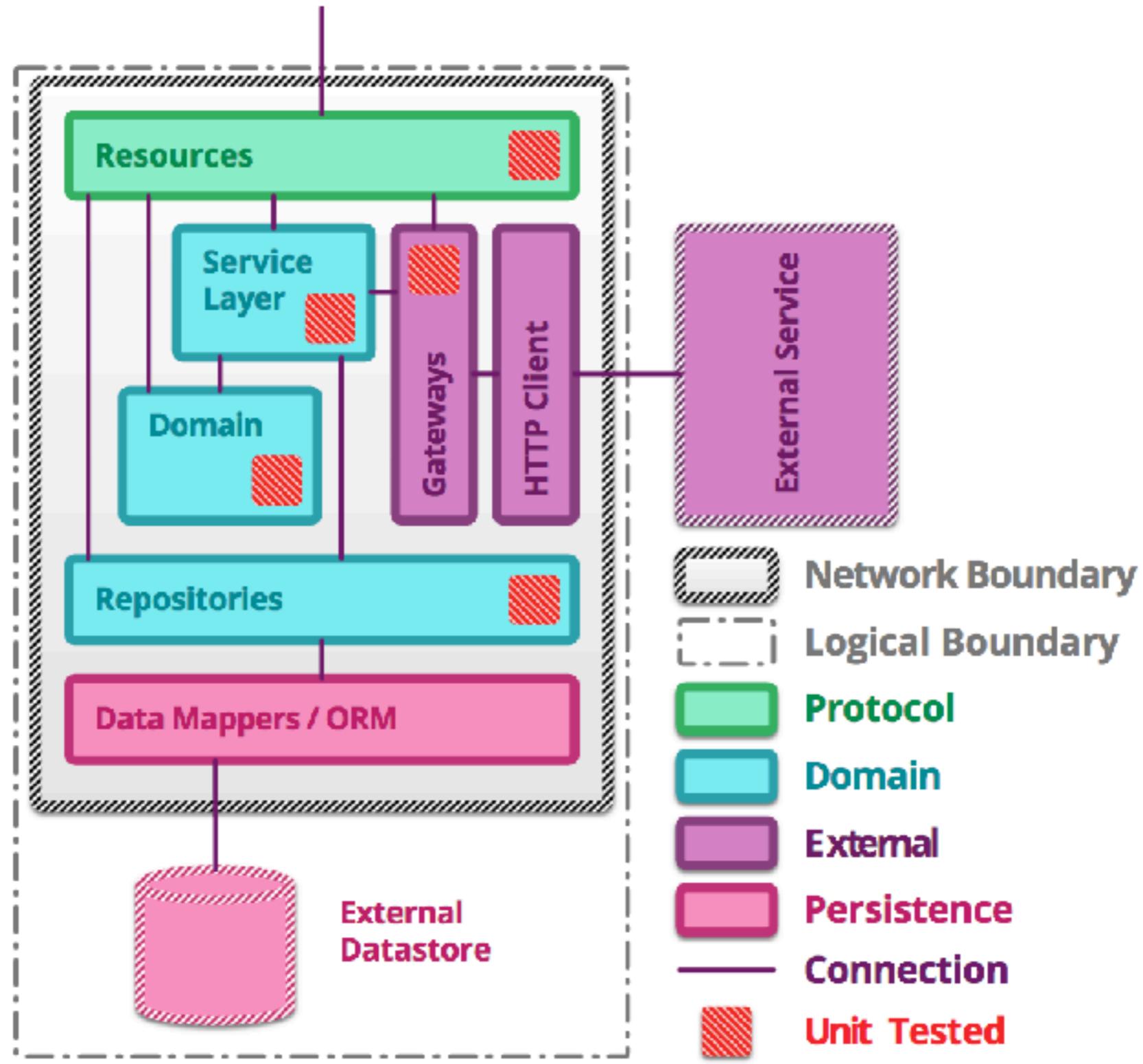
# Unit testing

Unit test แค่บอกว่า ส่วนๆหนึ่งใช้ได้ แต่ไม่ได้หมายความว่าจะทำงานร่วมกันได้



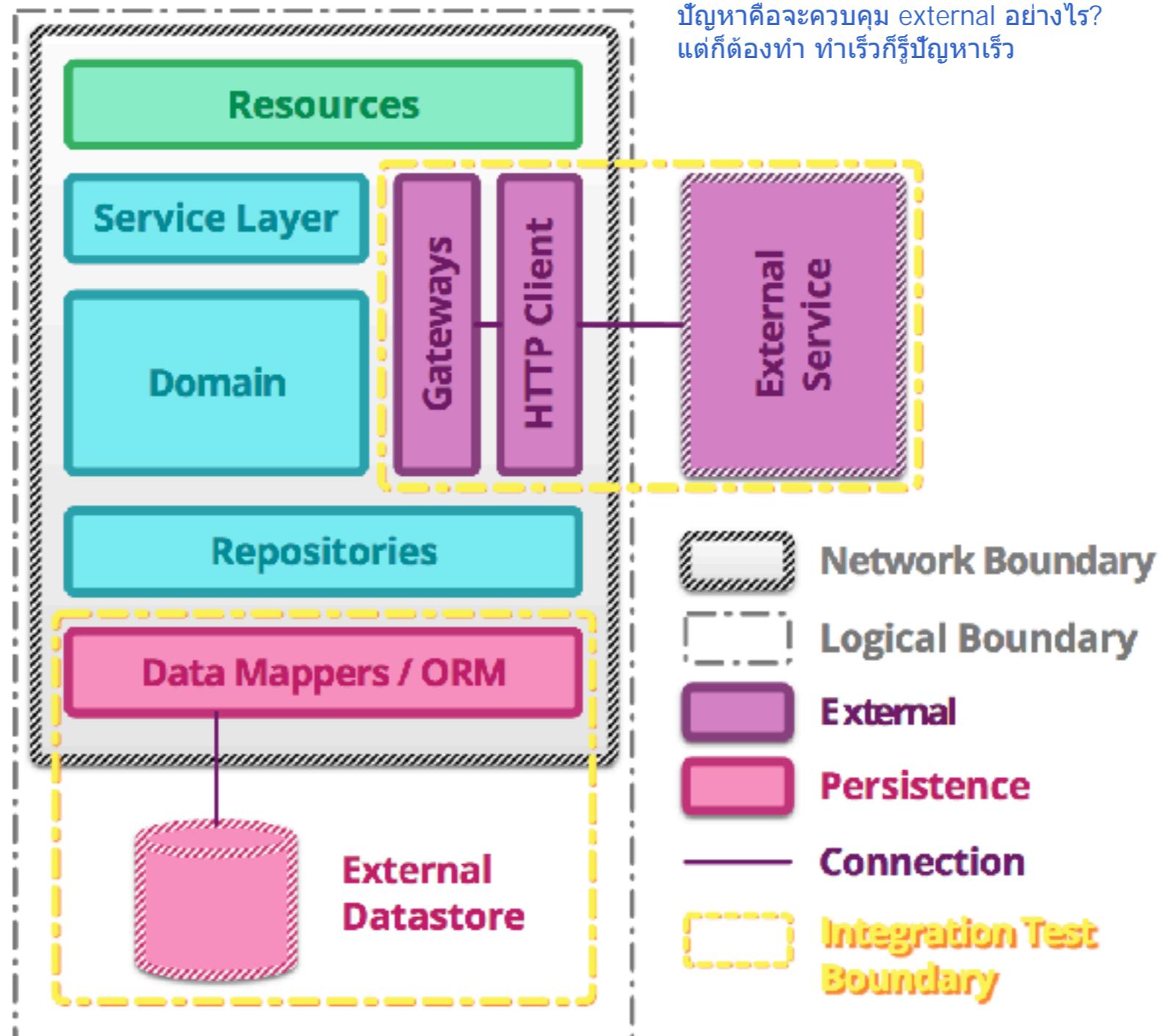
Unit testing not guarantee  
about the **behavior** of system





# Integration testing

เข่น ต่อ service ข้างนอกจริง หรือ ต่อ database จริง



No confidence about  
**biz requirement** are satisfied

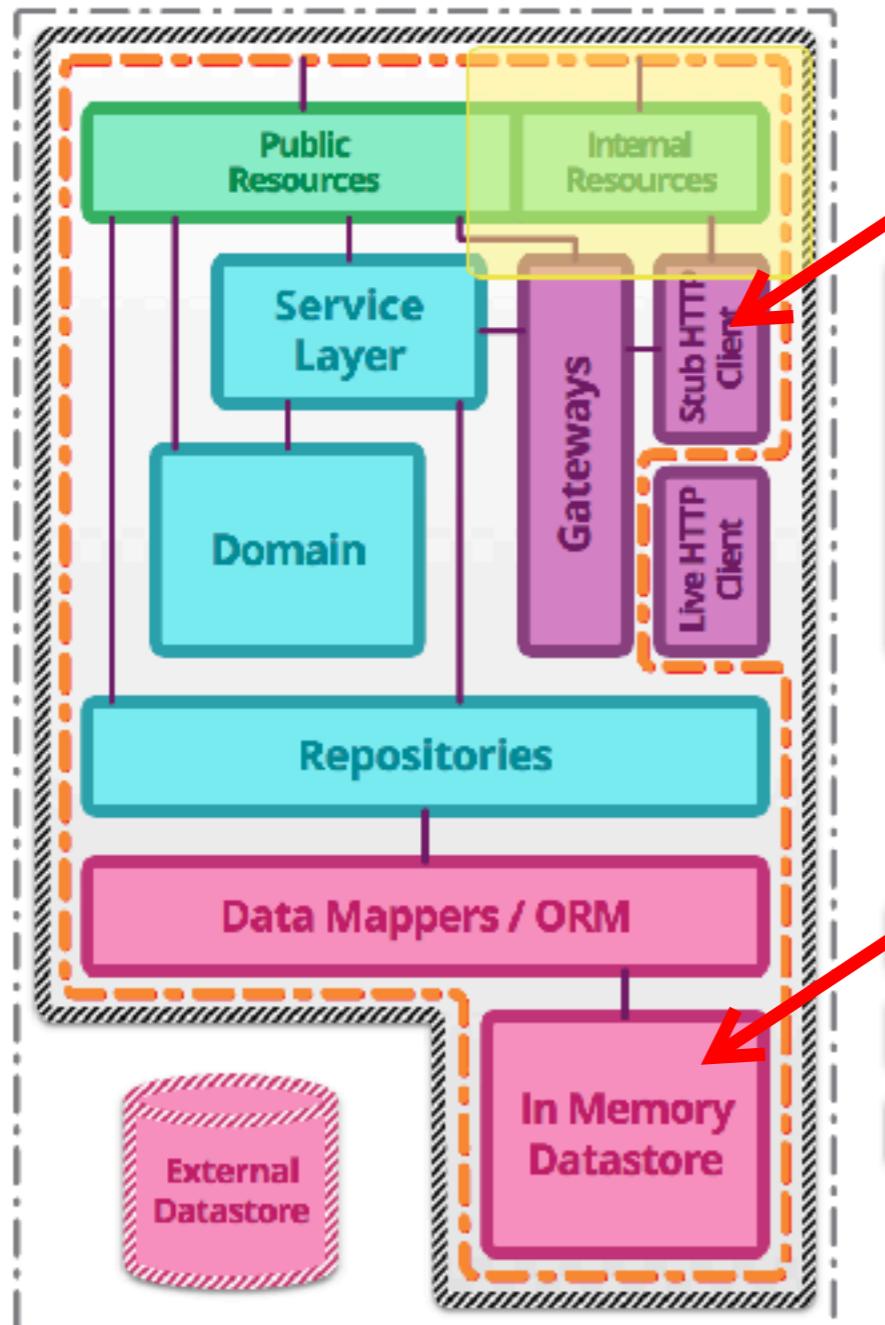


# Component testing

ทดสอบทั้งยัง โดยทำจำลอง external dependency ขึ้นมา เช่น Stub HTTP โดยไม่เปลี่ยน code แค่เปลี่ยน configuration

เป็นการทดสอบในบ้าน ทดสอบในบ้านให้รอบก่อนไปต่อนอกบ้าน จะได้รู้ว่า error เกิดที่ไหน

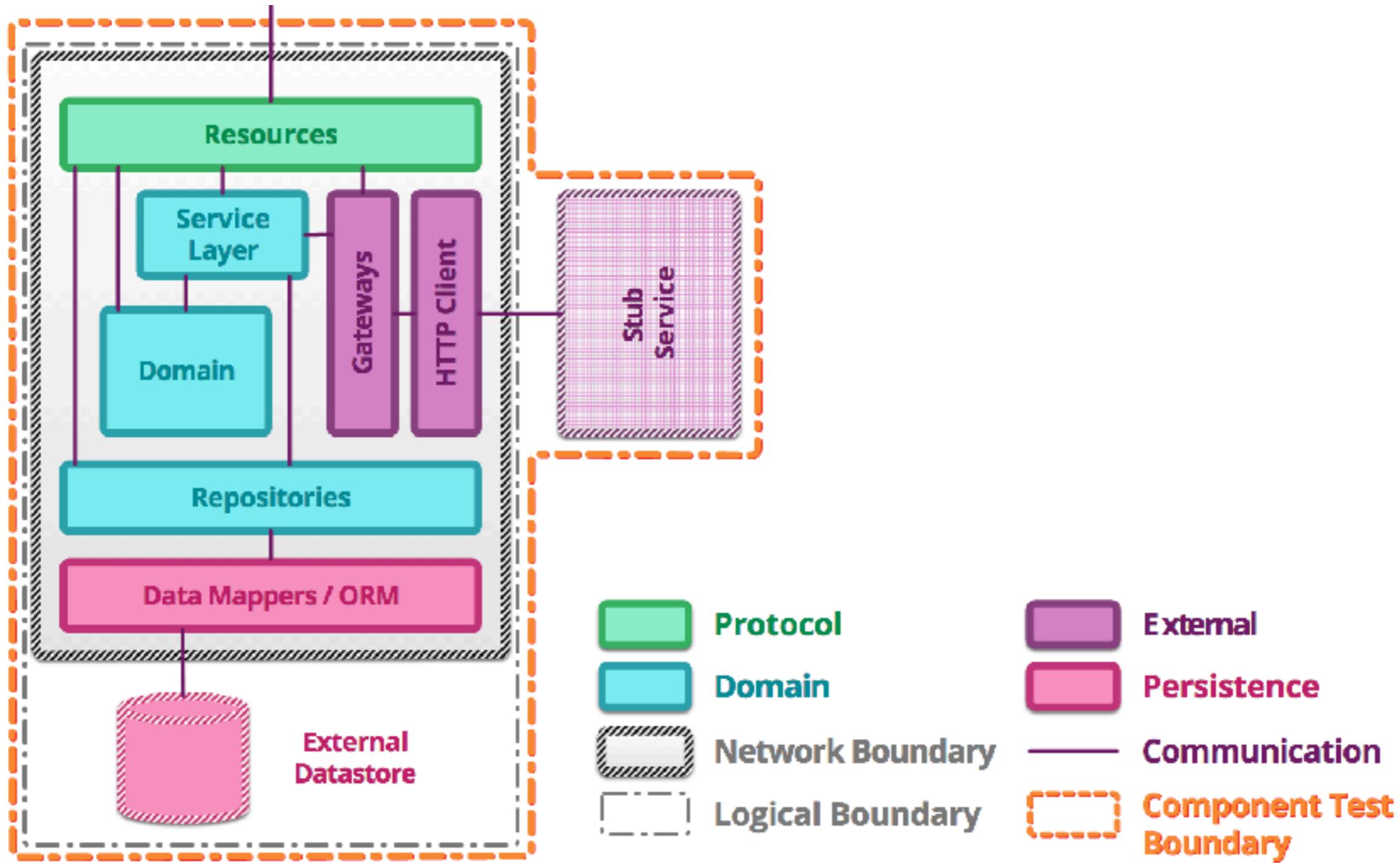
มักทำ Integration Test ก่อน เพื่อคุ้มครอง return อะไรมา  
จะได้อ้าไปทำ mock / stub ได้ เพื่อทำ component



- Protocol**: External
- Domain**: Persistence
- Network Boundary**: Communication
- Logical Boundary**: Component Test Boundary

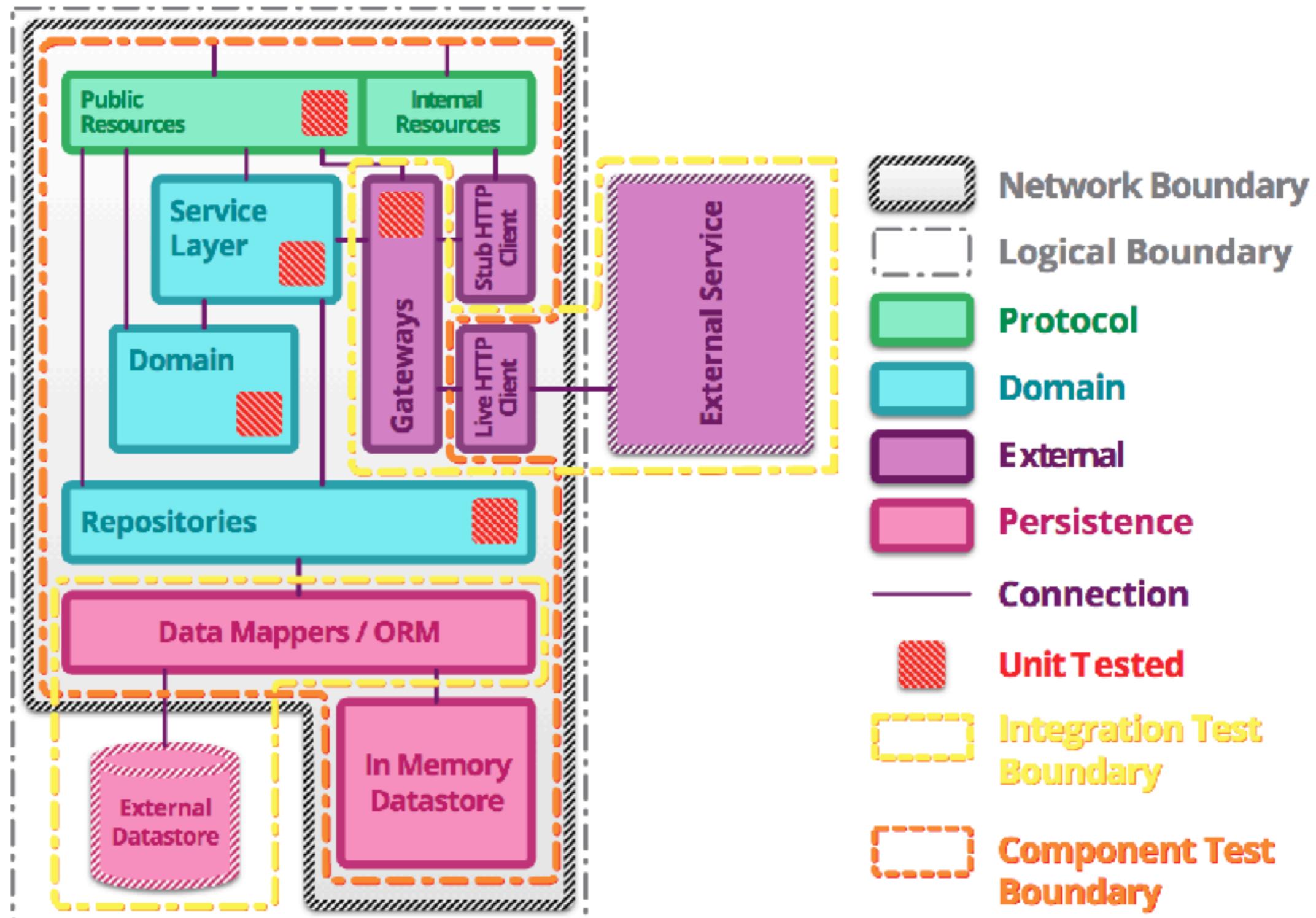


# Component testing



More high coverage  
**More confidence**

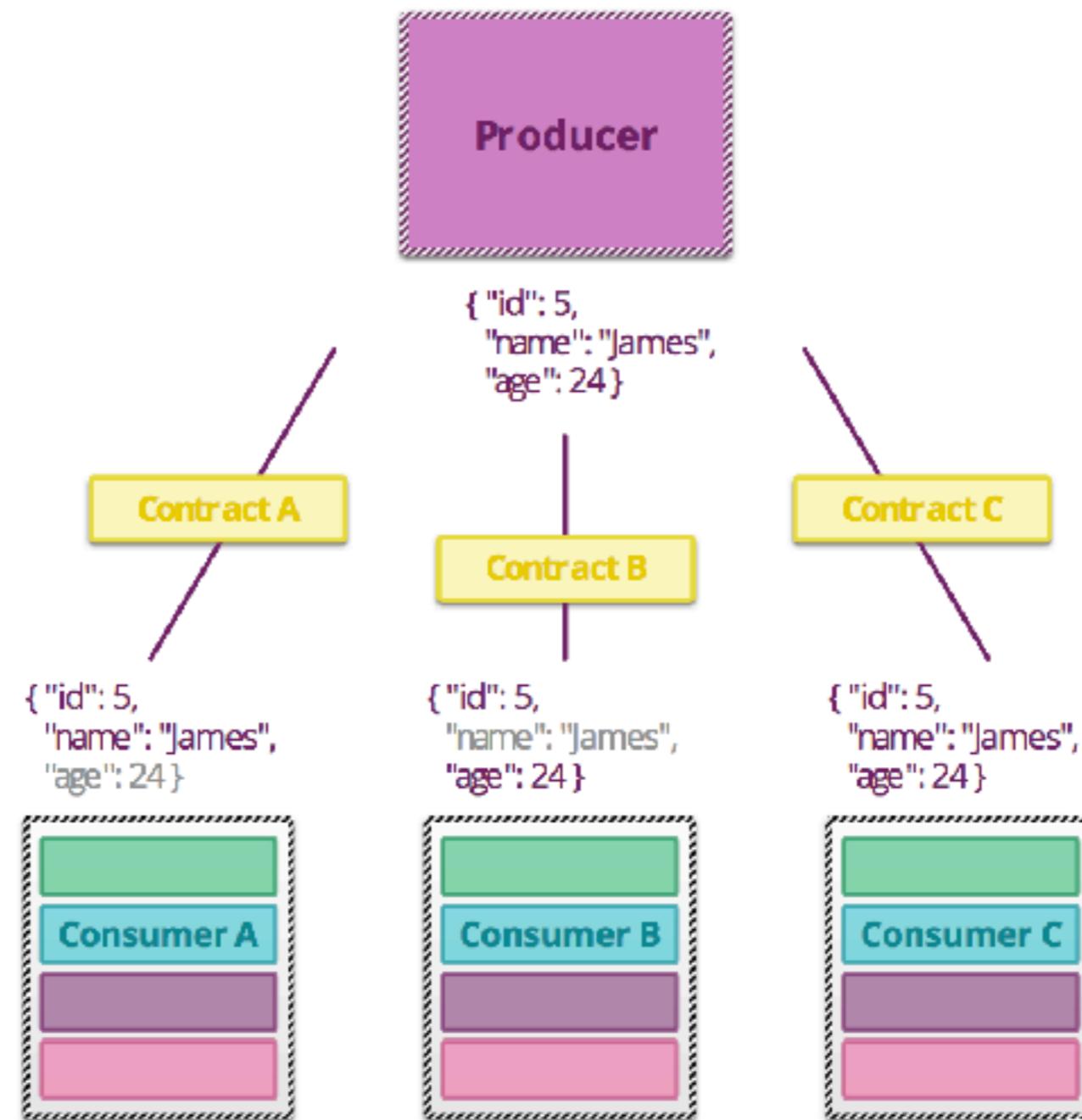




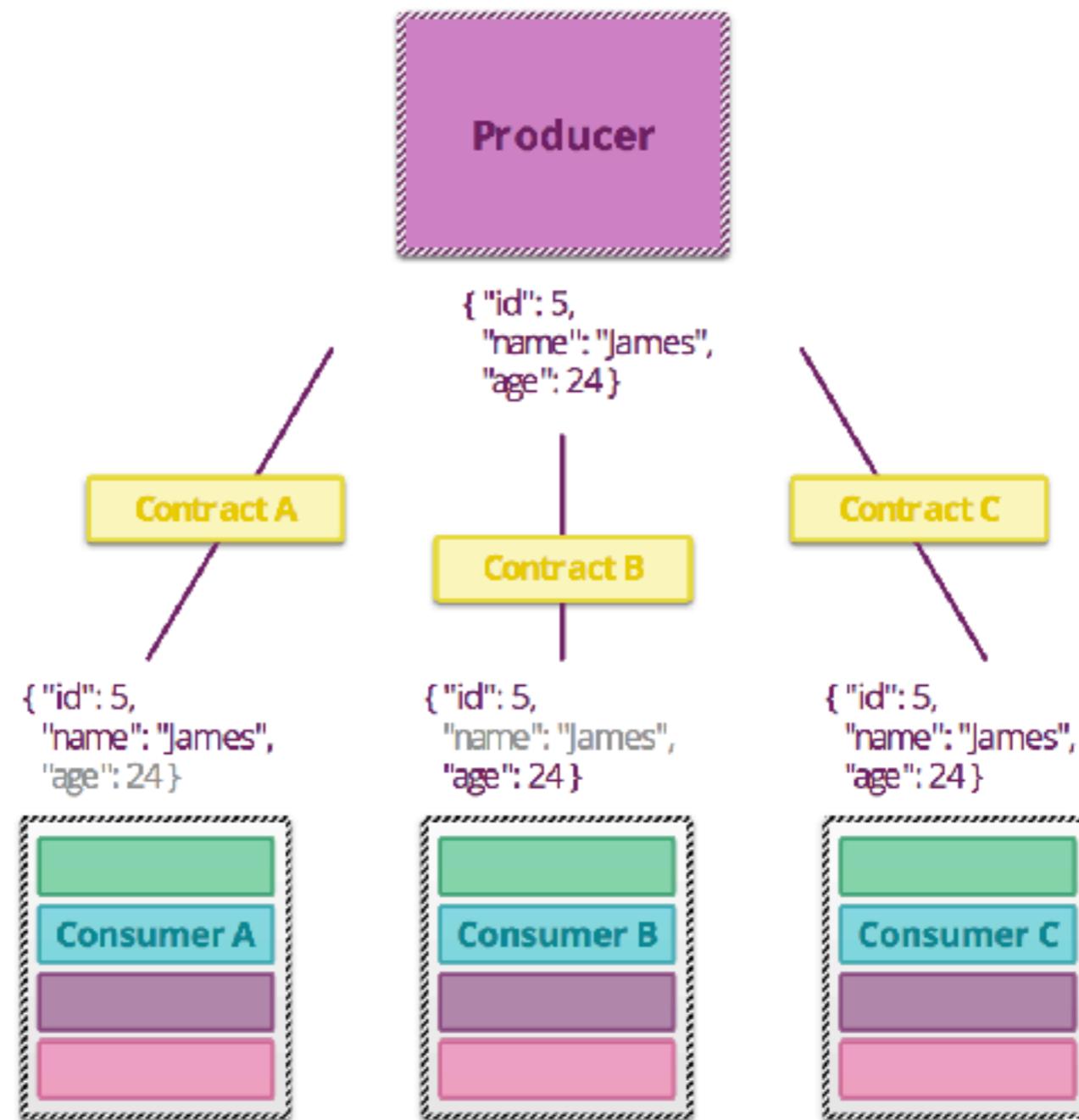
# More tests ?



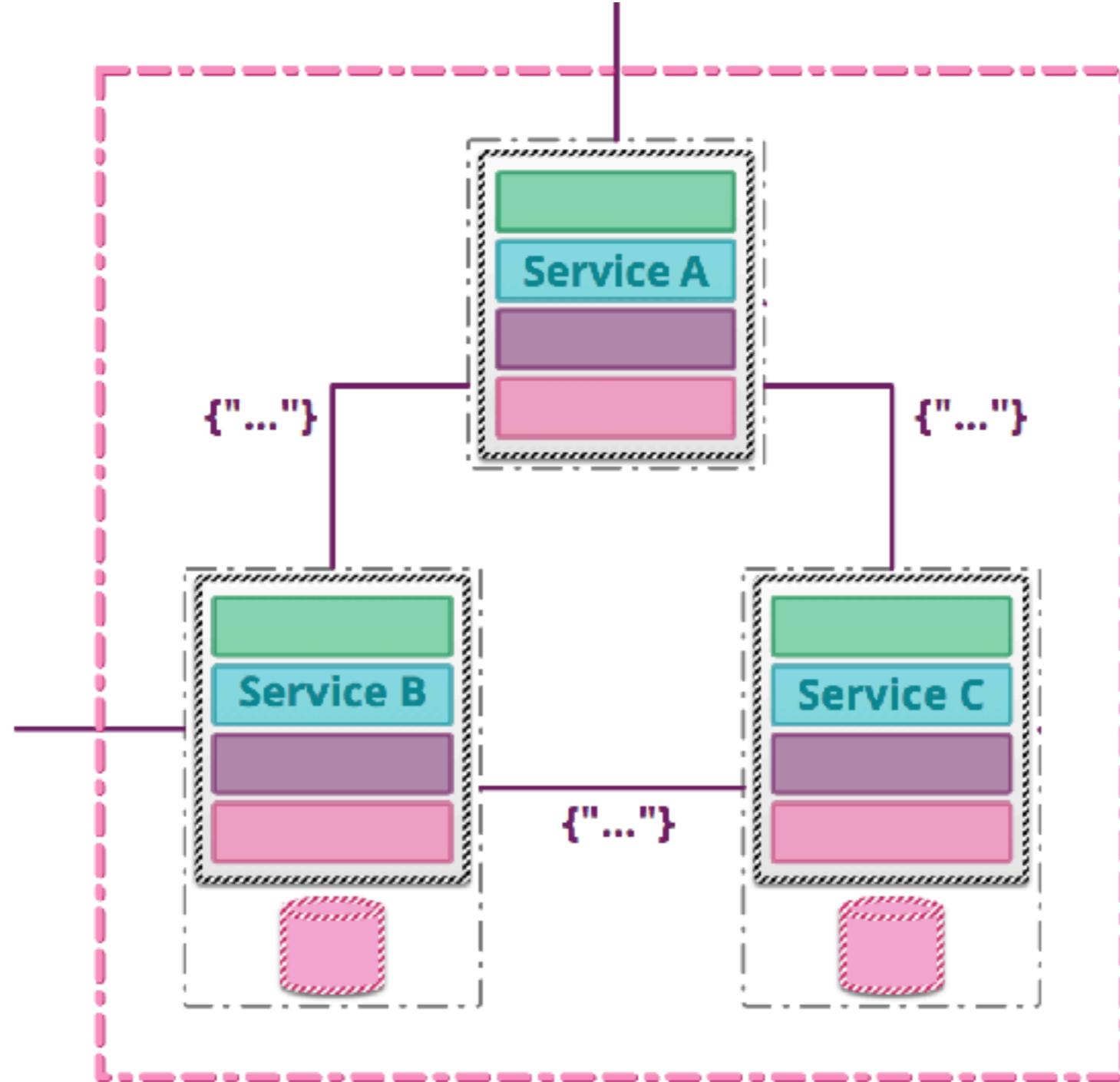
# Usage ?



# Contract testing



# End-to-end testing



# Resources ?



