

Distributed Systems

Chun-Feng Liao

廖峻鋒

Department of Computer Science

National Chengchi University

Distributed Systems

SOA and Microservices

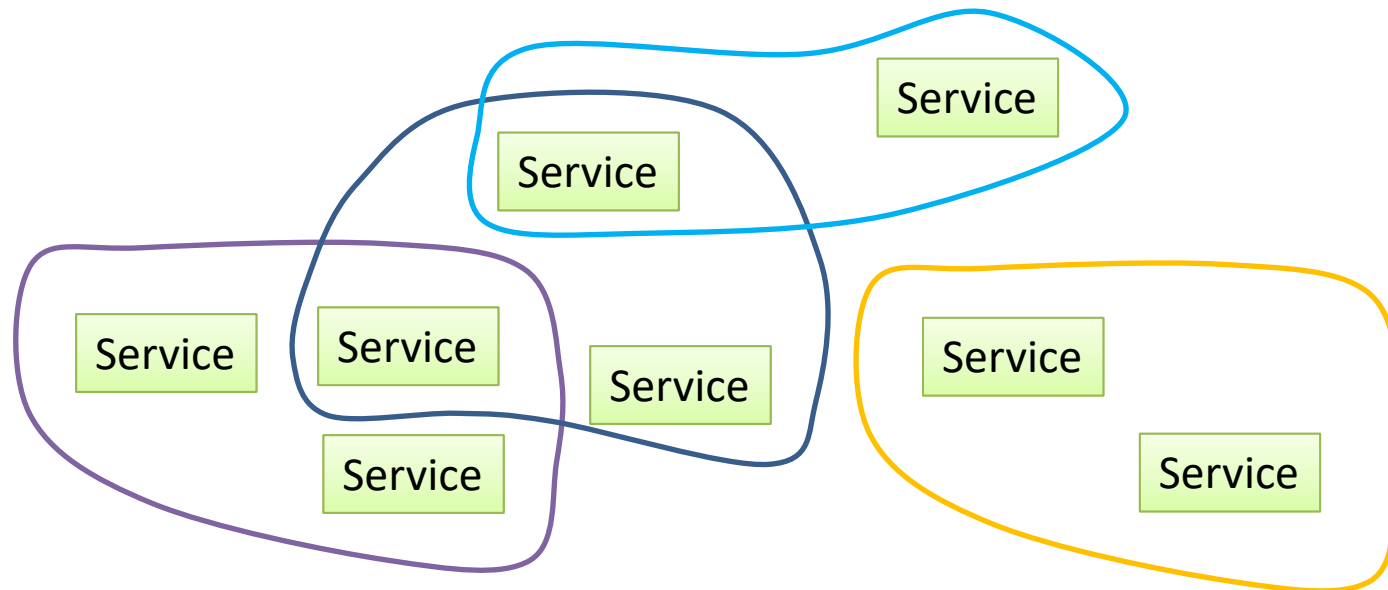
Chun-Feng Liao

廖峻鋒

Dept. of Computer Science
National Chengchi University

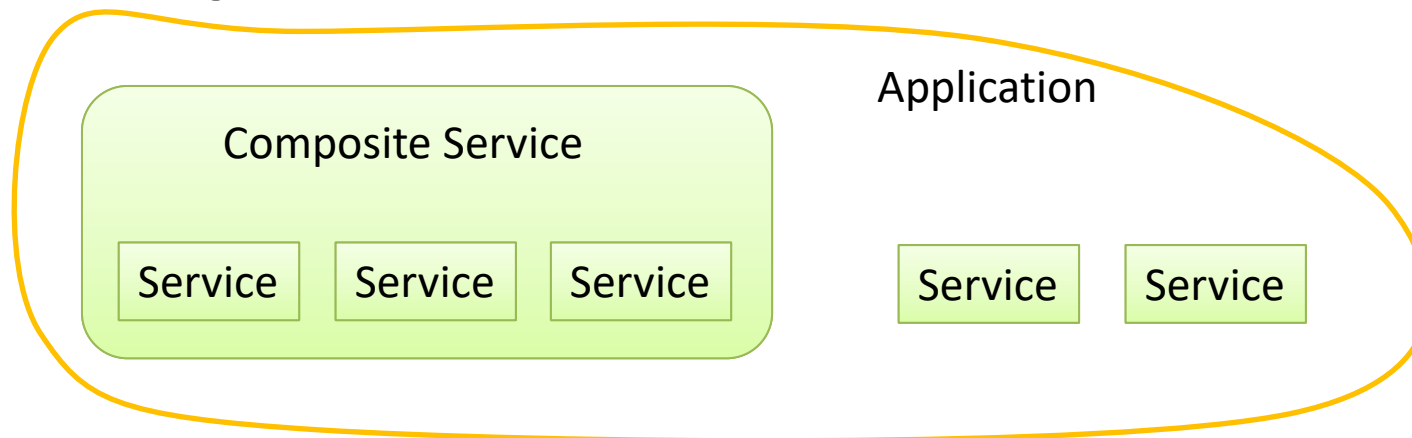
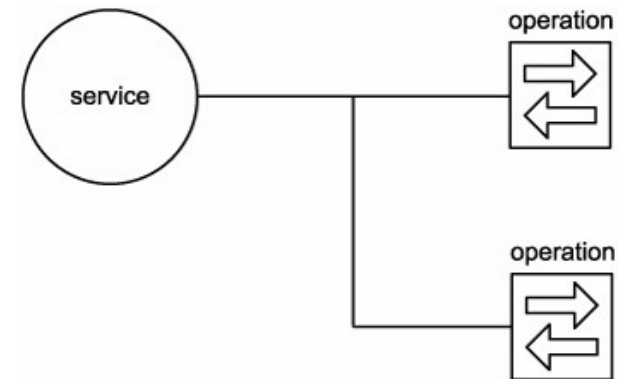
Service-Oriented Architecture

- An architectural style, in which:
 - Services are building blocks of applications
 - Services can be reused to build new applications
 - Services integration is based on open standards



Service

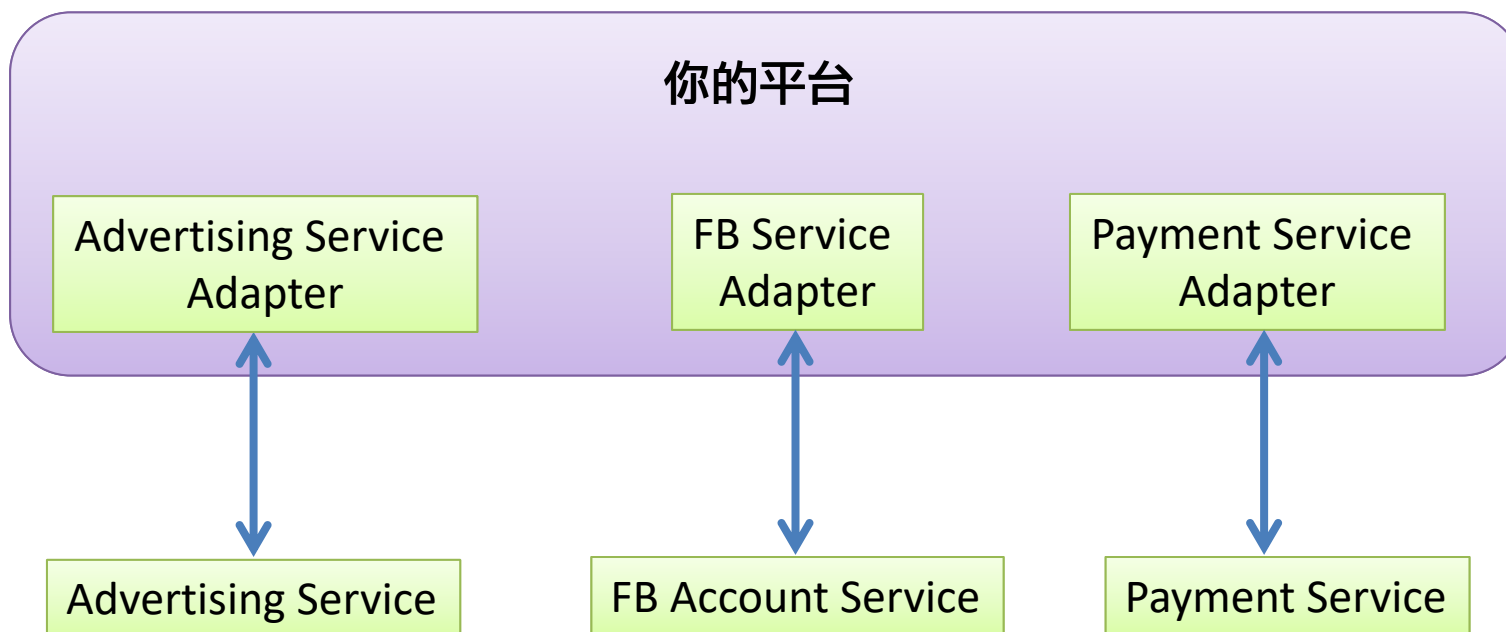
- A reusable component
 - Units of processing logic
 - Consists of a set of operations
- Can be building blocks of
 - Larger, more complex applications
 - Larger, more complex another services



範例

- 新創公司希望能透過廣告及電子商務營利
- 必須整合多個廠商提供的服務
 - 廣告平台 (API) Advertising Service
 - XX廣告公司提供專屬的Java RMI API
 - 銀行信用卡服務 Payment Service
 - XX銀行透過專屬的c-based library由遠端進行加密呼叫
 - 帳號認證機制 FB Account Service
 - FB 提供RESTful API

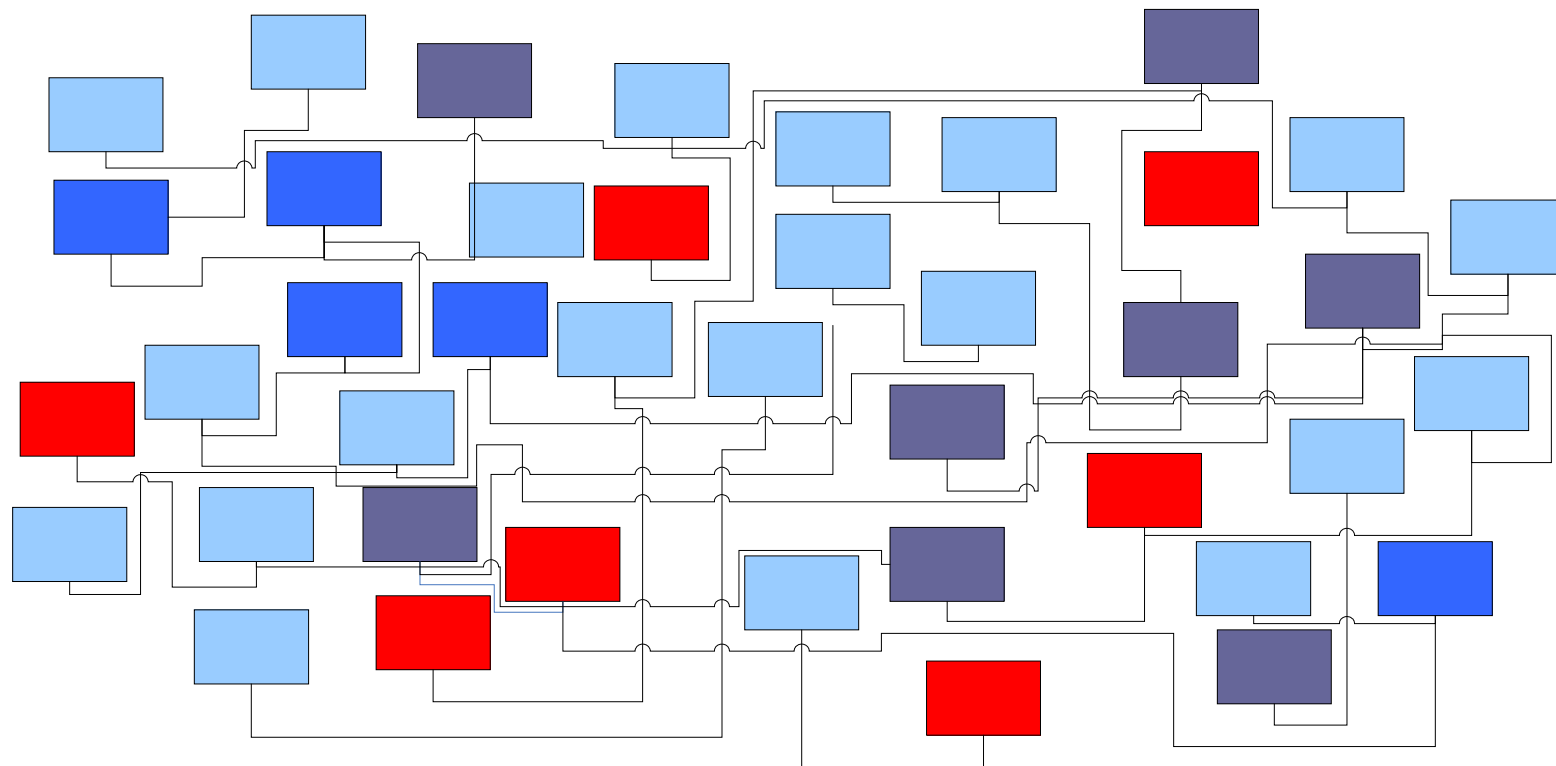
一開始



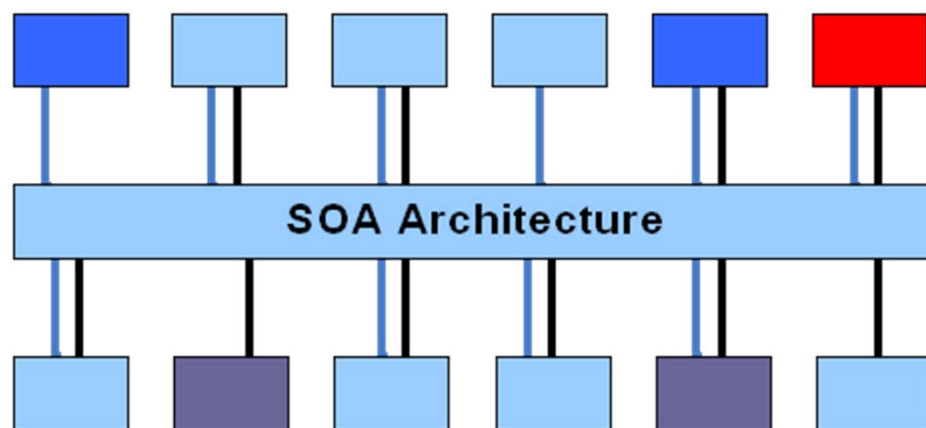
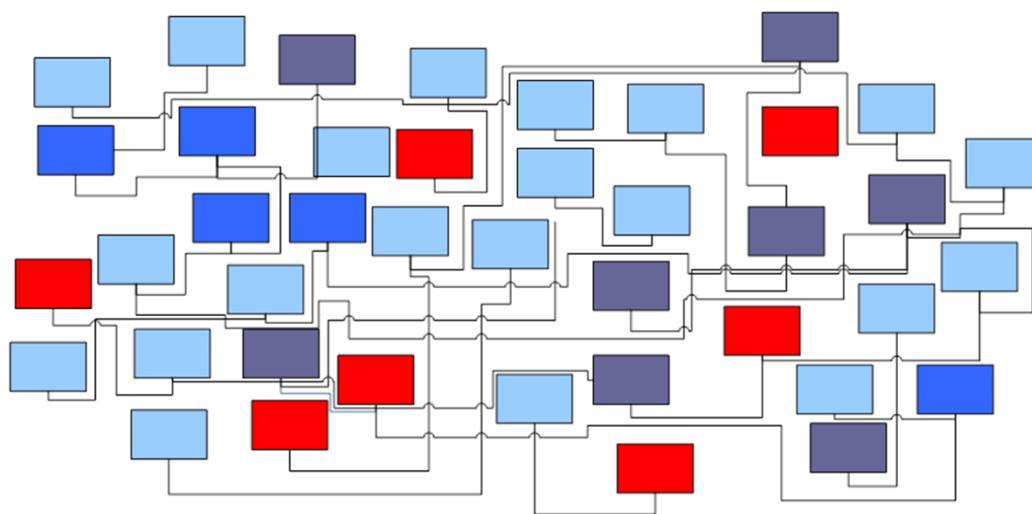
傳統服務整合

- 提供服務的廠商
 - 不太能確保其服務的品質
- 開發Adapter
 - 為每個廠商的API開發一份整合用的code接到自己的平台
 - 異質語言有可能無法整合
 - 依廠商而需學習多種平台的code (人力、時間成本!)
- 要更換廠商或更新時
 - 重新再找廠商
 - 要重新修改自己平台的原始碼
 - 而要新語言or技術要再學一次→人力、時間成本!

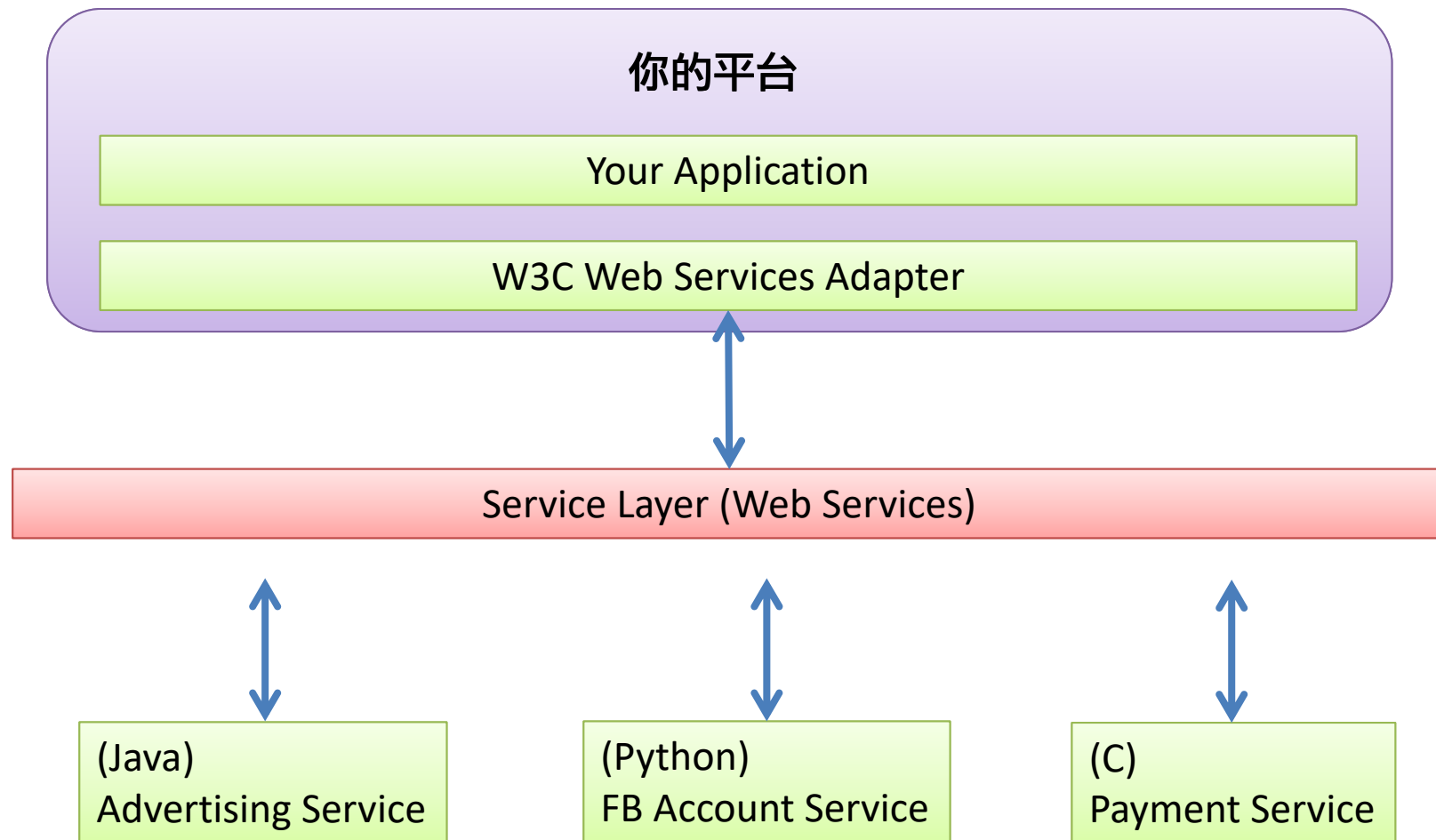
一年後



SOA的目標



Service-Oriented Integration

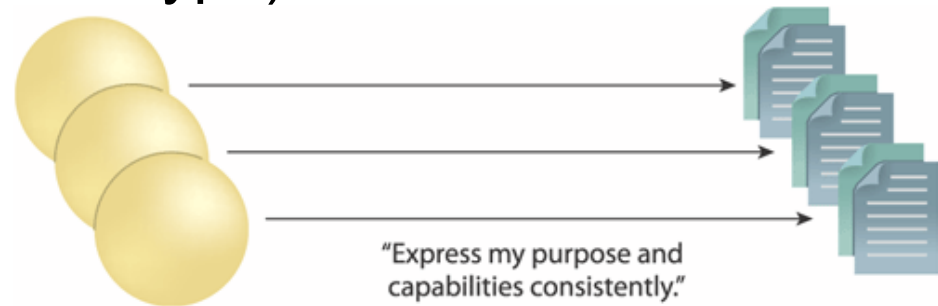


服務導向整合特色

- Standardization
- Loosely coupling
- Statelessness
- Discoverability
- Composability

Standardized Service Contracts

- Services use standardized service contract to
 - Express their capabilities
 - Express their quality
- Focus on the areas of
 - Interface description (signature)
 - Data representation (data type)
 - Policy (rule)

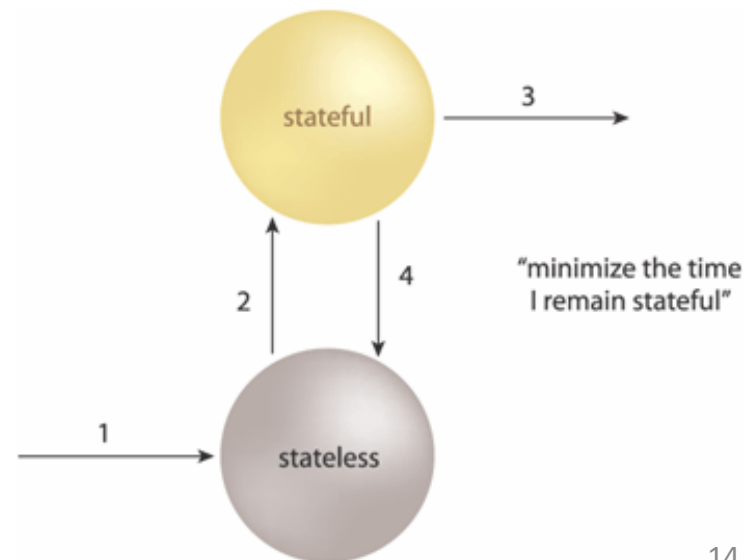


Loose Coupling

- The consumer of the service provides only the required interface definition
- Expect only the specified results on the interface definition
- The service is capable of handling all processing (including exception processing)

Minimal Statefulness

- Minimize resource consumption by deferring the management of state information when necessary
- Most services do not maintain state between invocations
- Benefits
 - Increase scalability (why?)
 - Minimize dependency



Discoverability



電話服務
機動性:高，體積:小，位置:客廳



電話服務
機動性:無，體積:大，位置:書房



電話服務
機動性:無，體積:中，位置:客廳



視訊服務
面板:大，位置:客廳



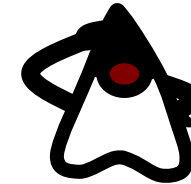
視訊服務
面板:小，位置:客廳

服務目錄

視訊電話應用程式

Hi, 請給我一個電話服務和
視訊服務，電話服務最好是
機動性高的，視訊服務
最好是在廚房...

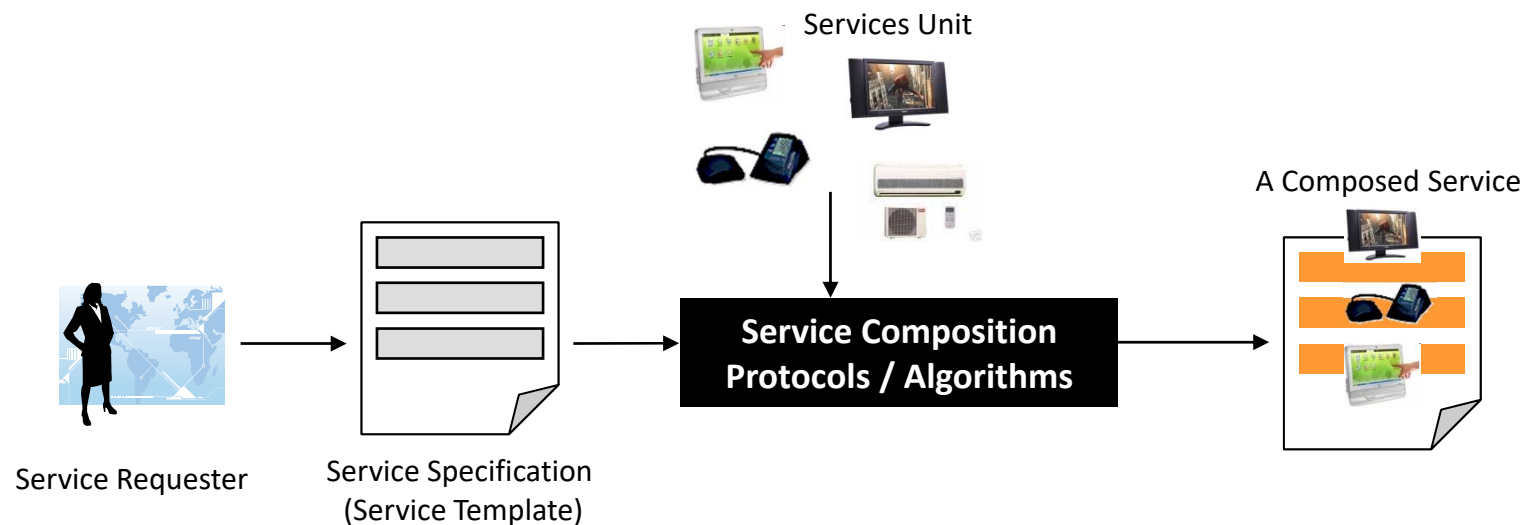
最符合您的要求的只
有這些，你要不要？



服務目錄管理員

Composability

- Services are able be aggregated to solve larger problems
- If multiple qualified candidates are discovered the runtime picks the one with highest “quality”



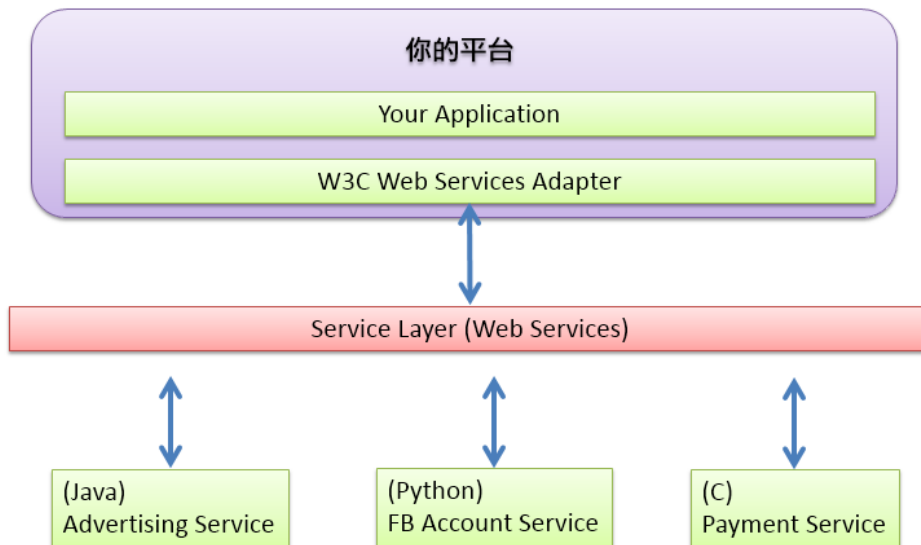
問題

服務使用者

如何呼叫別人的服務?

如何知道服務的功能、特色與品質?

如何找到有那些服務可以用?



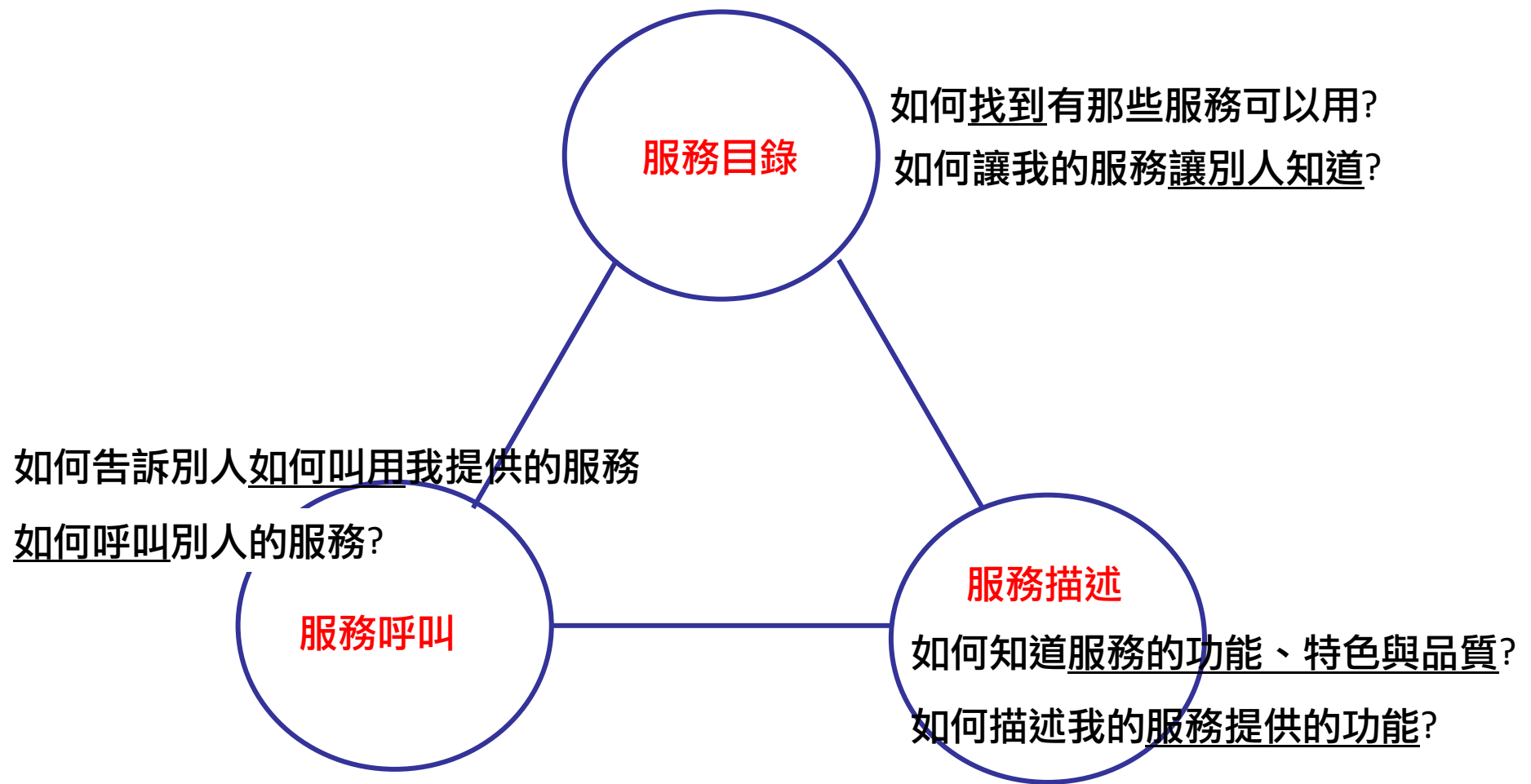
服務提供者

如何描述我的服務提供的功能?

如何告訴別人如何叫用我提供的服務

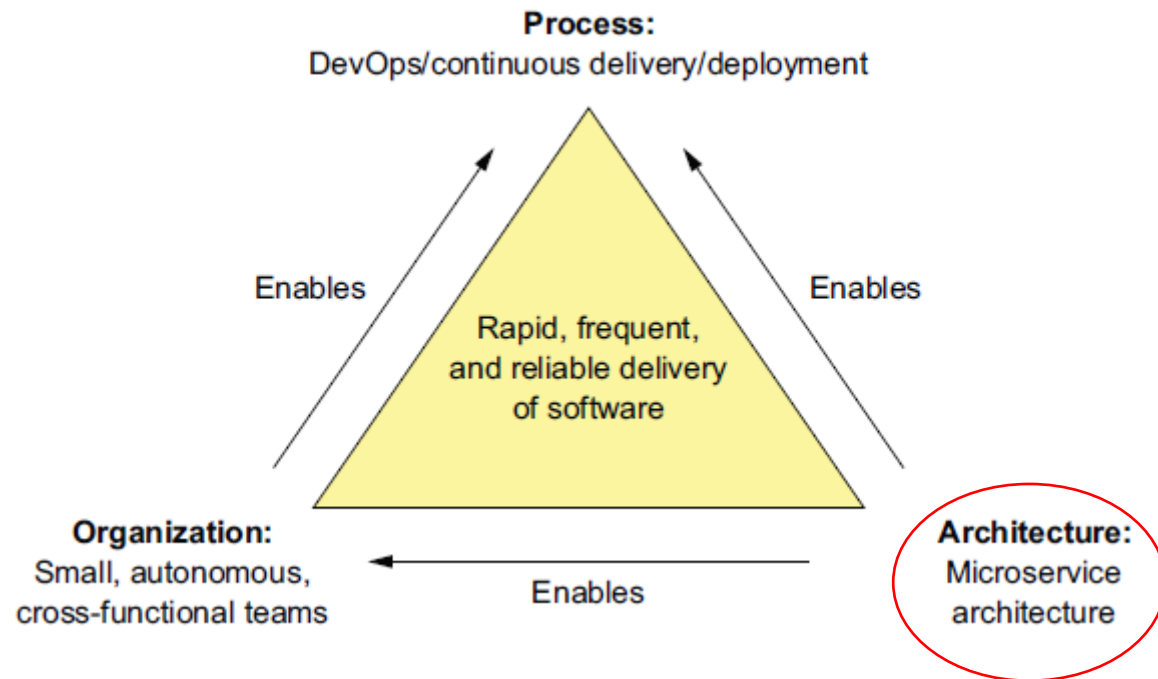
如何讓我的服務讓別人知道?

Essential SOA Requirements



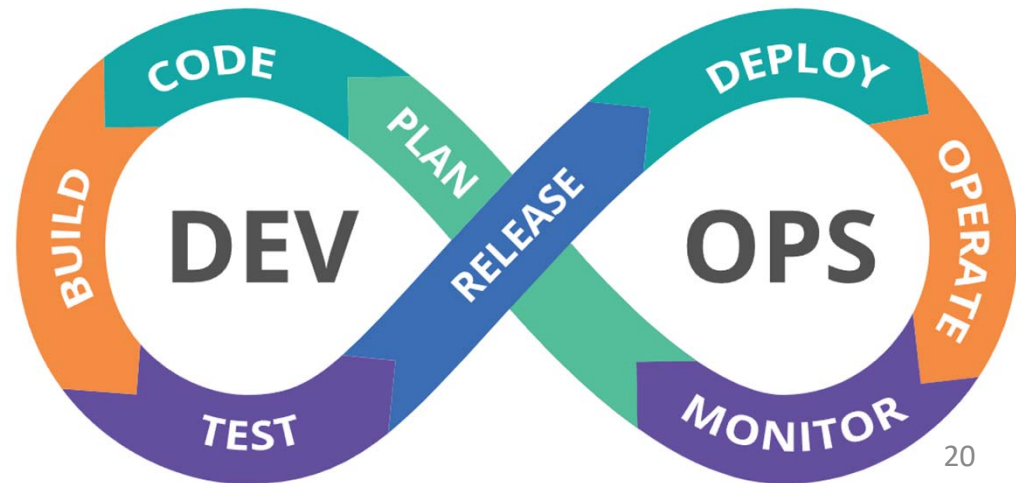
當代企業應用程式特色

- 3大需求
 - Rapid delivery
 - Frequent delivery
 - Reliable delivery
- 3大策略
 - 製程
 - DevOps/CI/CD
 - 組織
 - 小型全端團隊
 - 架構
 - 微服務



什麼是DevOps?

- 定義 (CMU 軟體工程學院)
 - A set of practices, which (實踐方法)
 - reduces the time between change to production (快速回應)
 - while ensuring high quality (保持穩定、可靠、安全)
- 關注開發者Dev與維運者Ops順暢的協作模式
 - 開發者: 程式開發、品保、測試人員
 - 維運者: SP、OP



組織：逆Conway定律

The reverse Conway maneuver

In order to effectively deliver software when using the microservice architecture, you need to take into account Conway's law (https://en.wikipedia.org/wiki/Conway%27s_law), which states the following:

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

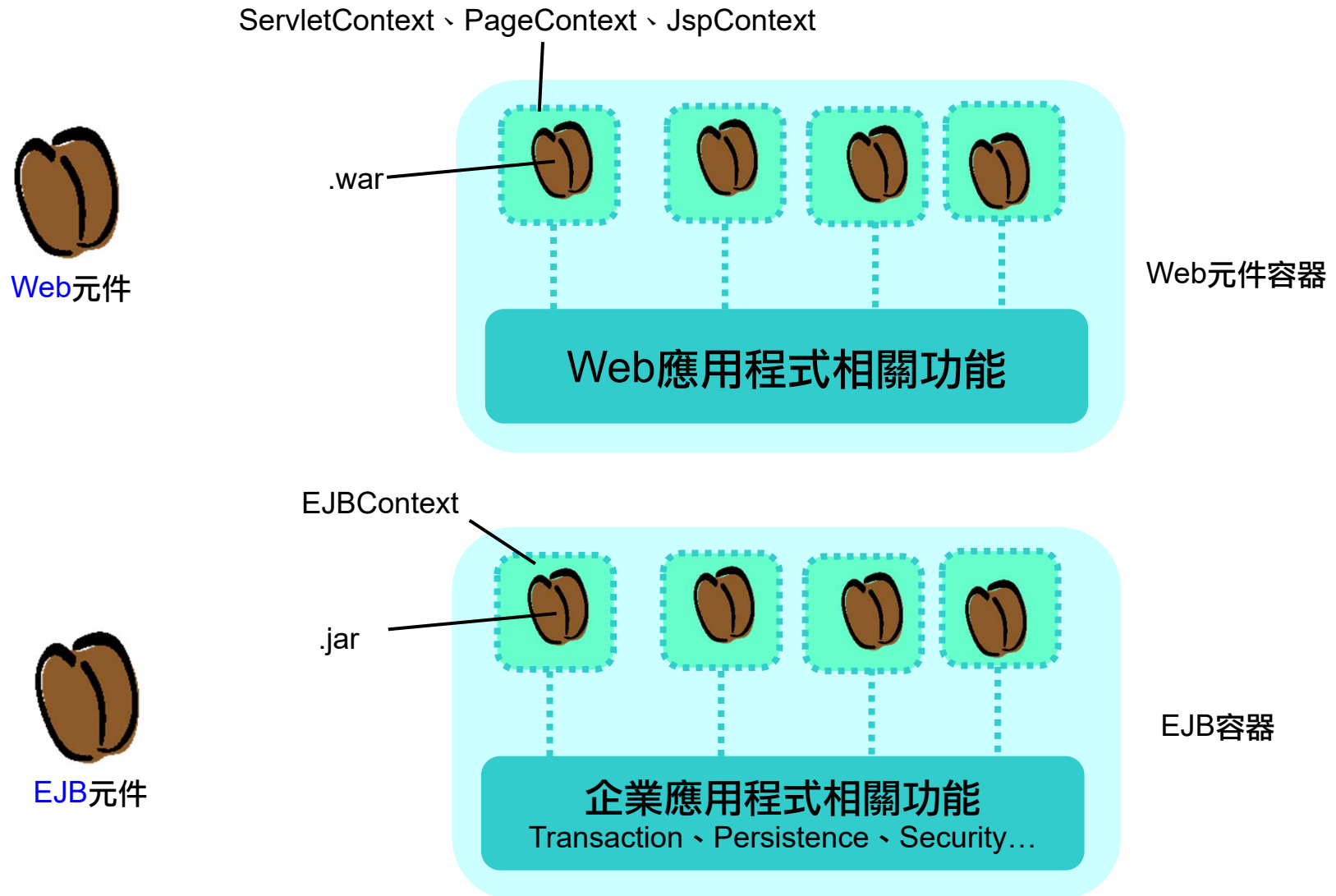
組織慢慢會變成和系統結構相似

Melvin Conway

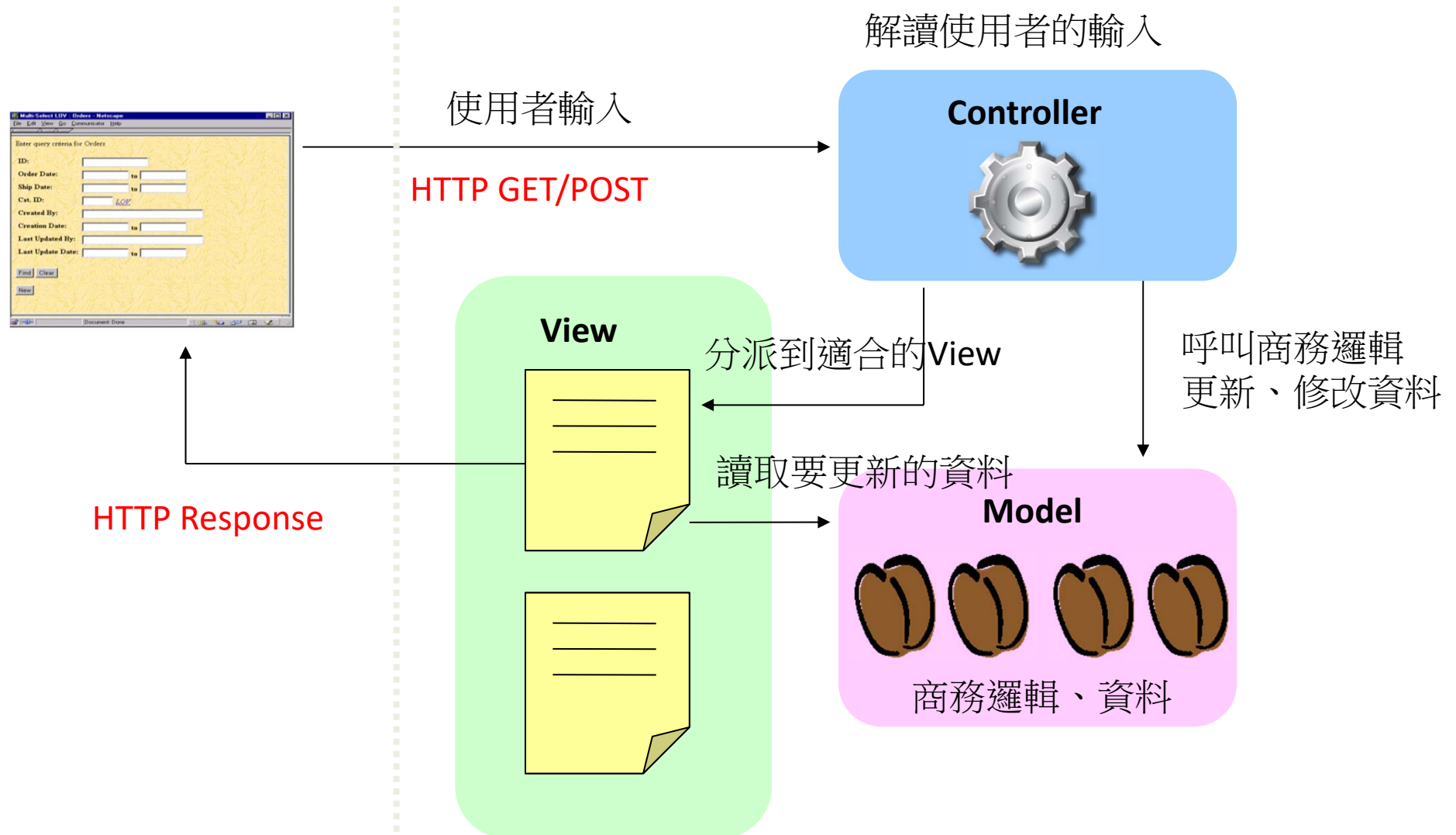
In other words, your application's architecture mirrors the structure of the organization that developed it. It's important, therefore, to apply Conway's law in reverse (www.thoughtworks.com/radar/techniques/inverse-conway-maneuver) and design your organization so that its structure mirrors your microservice architecture. By doing so, you ensure that your development teams are as loosely coupled as the services.

一開始就根據系統結構來設計組織

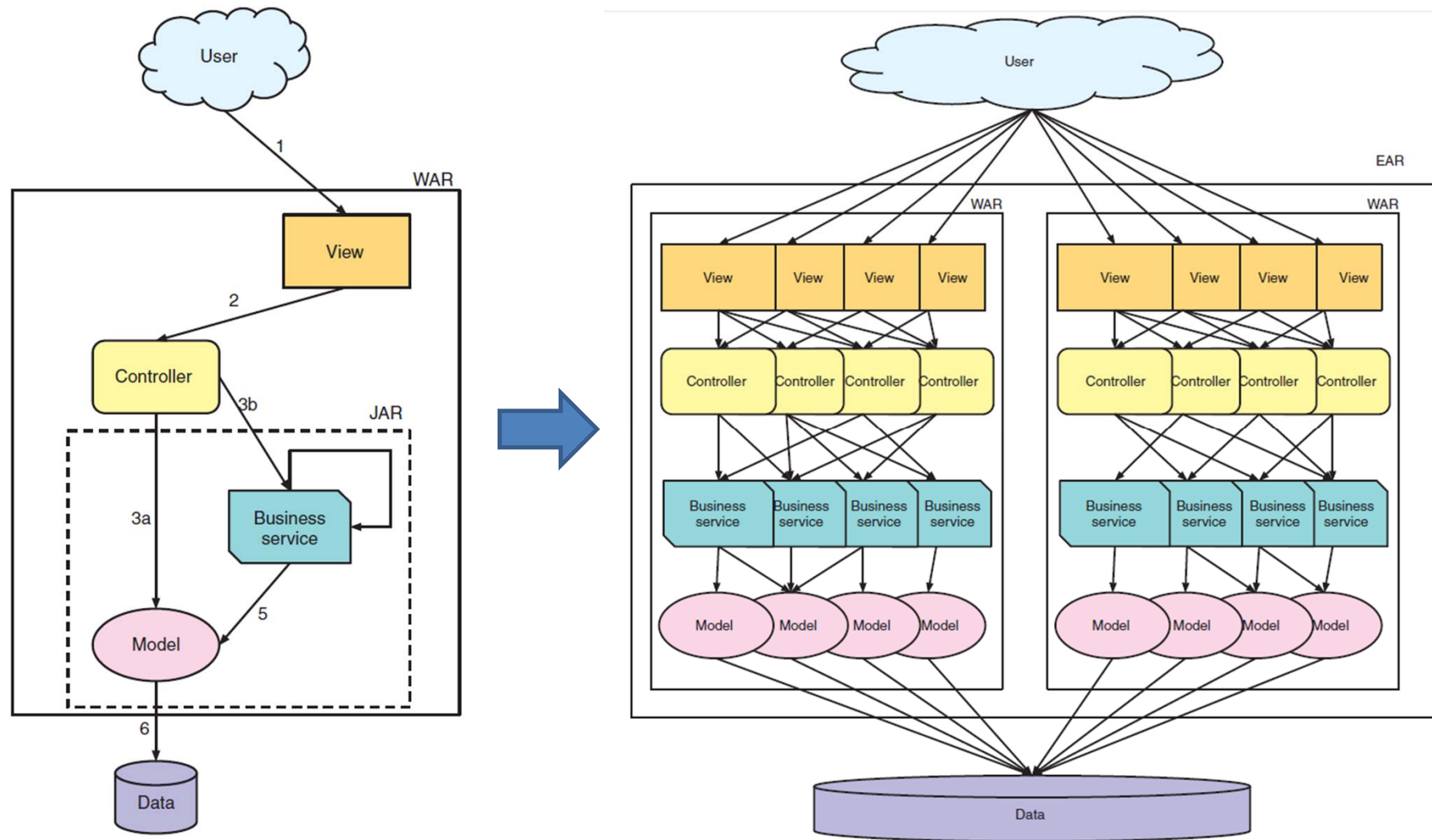
Review: Java EE Containers



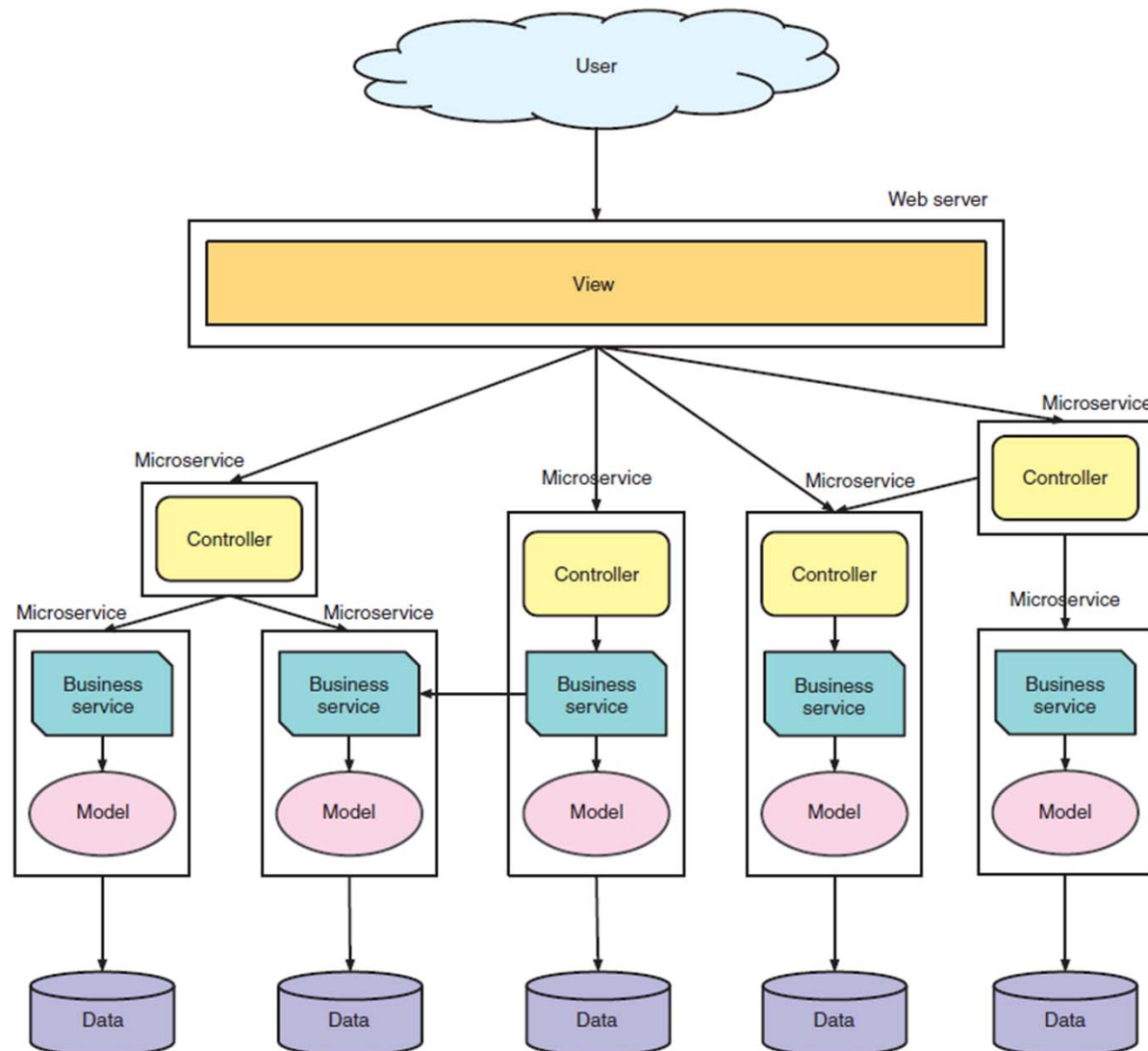
Review: Web MVC Architecture



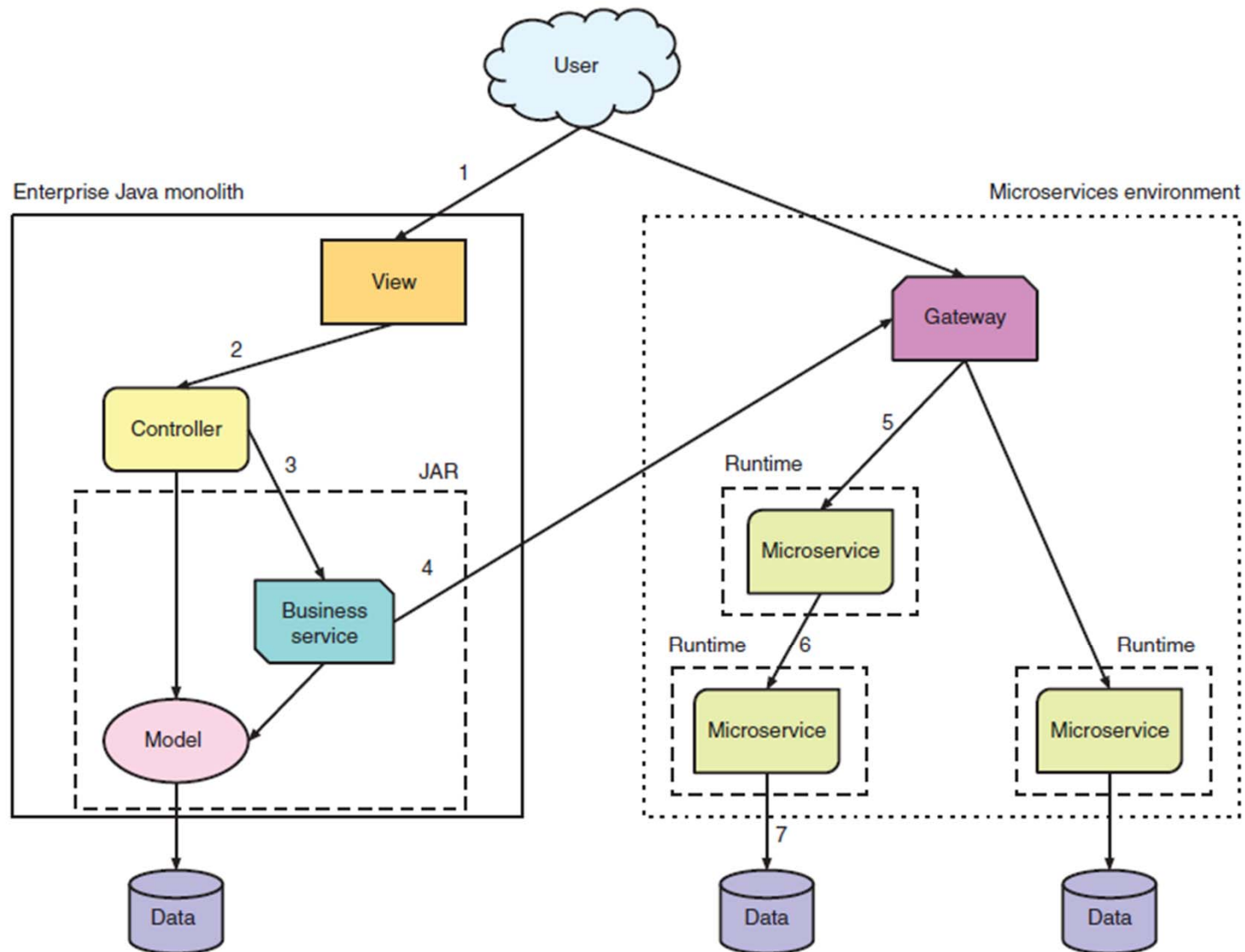
Typical Java EE “monolith” Architecture



Microservice Architecture Style



共存架構

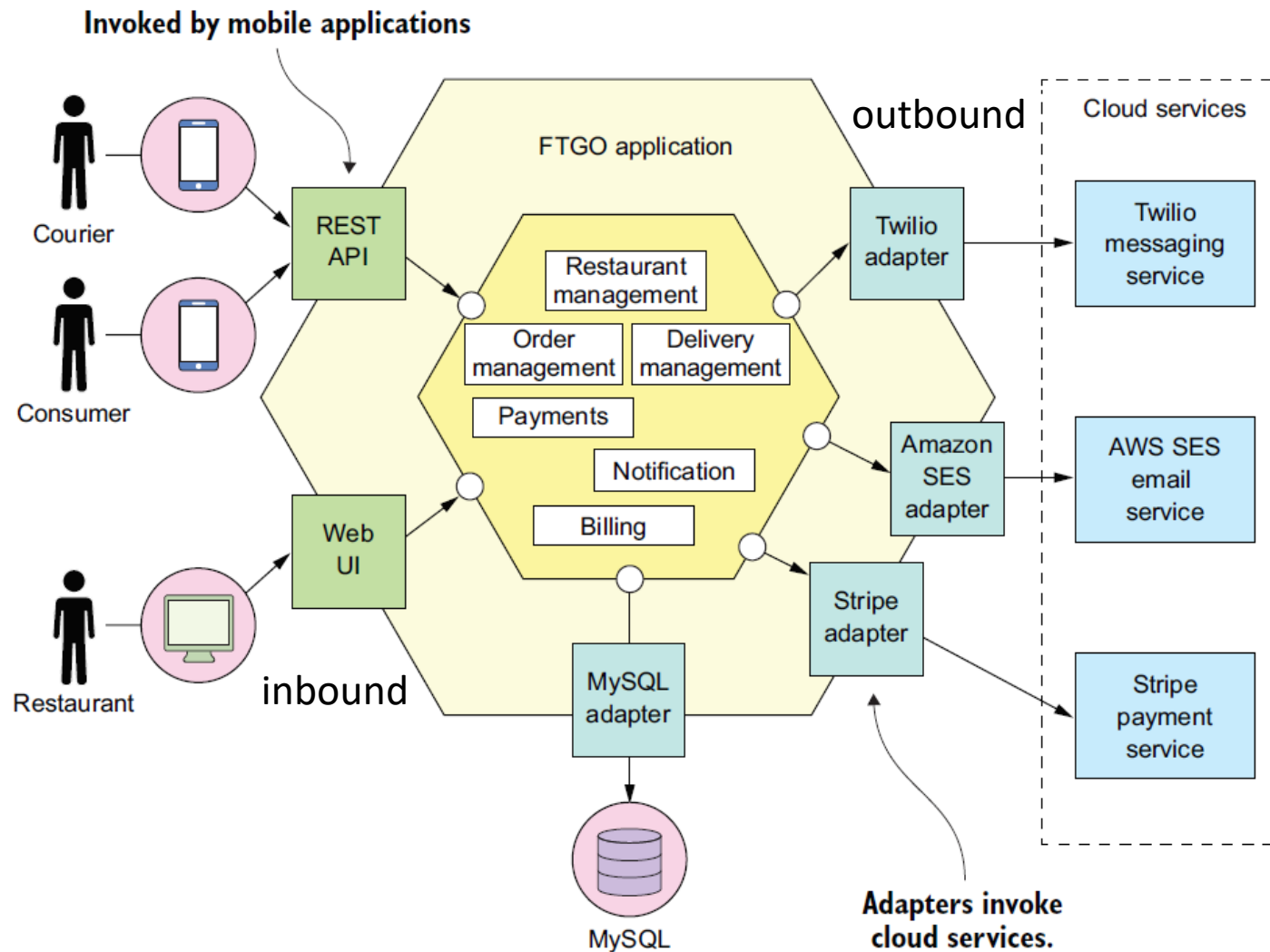


Microservice

- Definition
 - Structure the application as a collection of loosely coupled, independently deployable services
- Service
 - A standalone component that implements some useful functionality
 - A service's API encapsulates its internal implementation
 - The API can not be bypassed: enforcing the modularity
 - Each service has its own architecture and technology stack

The FTGO Application

傳統SOA架構



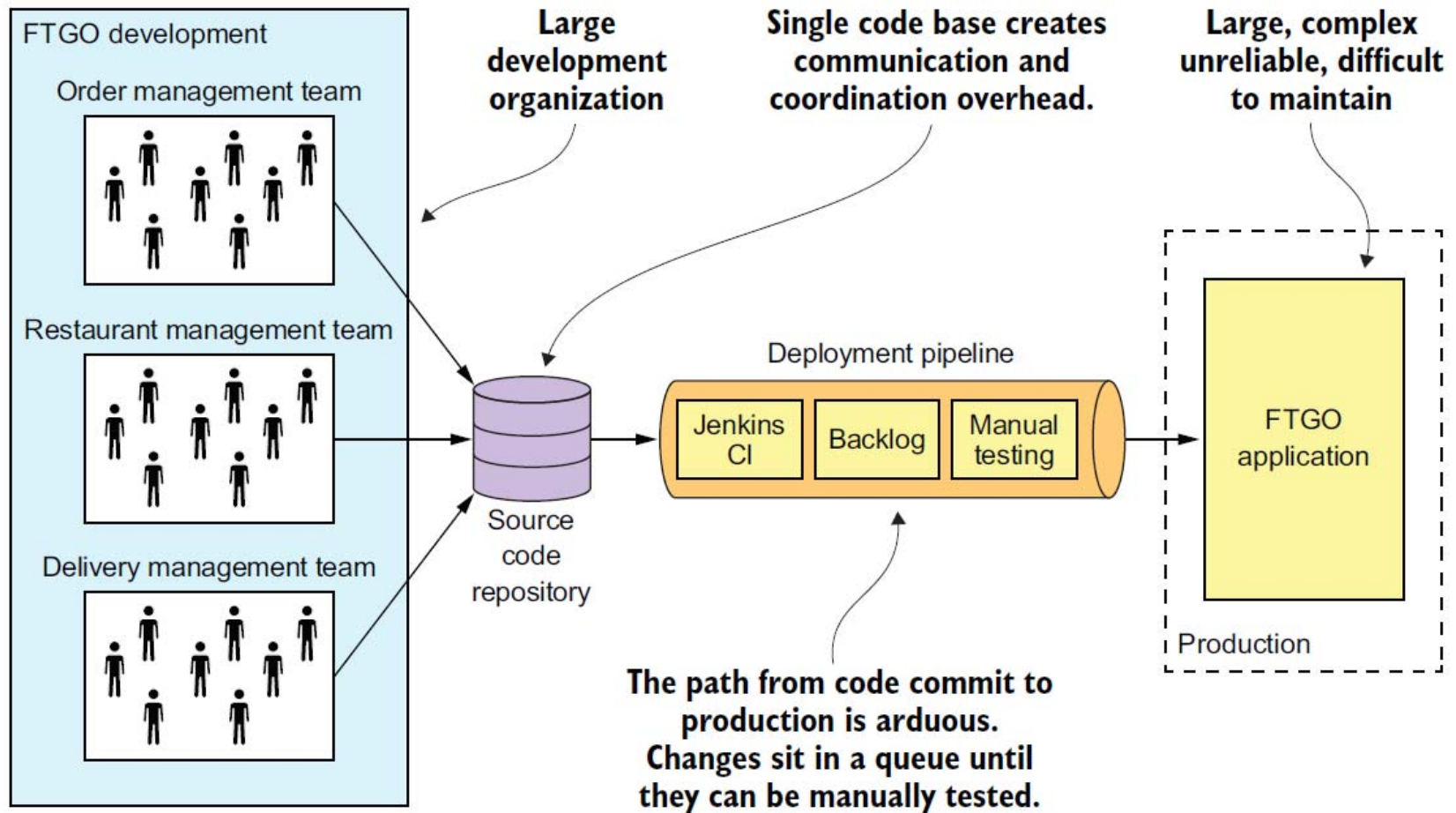
The FTGO Application

- Hexagonal architecture
 - Core: business logic; surrounding: external services
- A typical Java EE application
 - Packaged as a single WAR file

Benefits of Monolithic Architecture

- (前提) When the application was relatively small
 - **Simple to develop:** less IPC
 - **Easy to make changes:** change the code and the database schema, build, and deploy
 - **Straightforward to test:** the developers wrote end-to-end tests that launched the application
 - **Straightforward to deploy:** copy the WAR file to a server
 - **Easy to scale:** run multiple instances of the application behind a load balancer

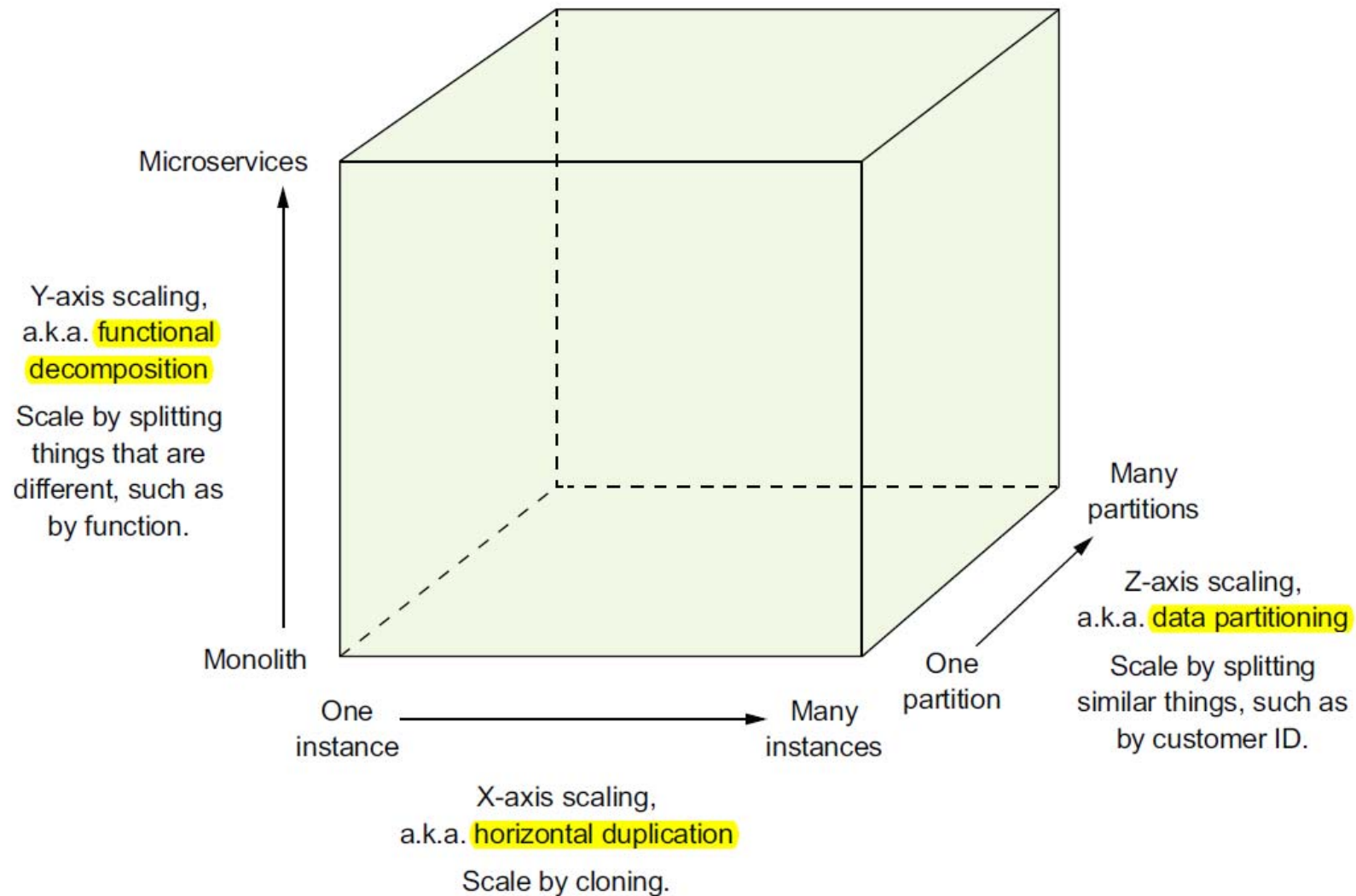
Step into the Monolithic Hell

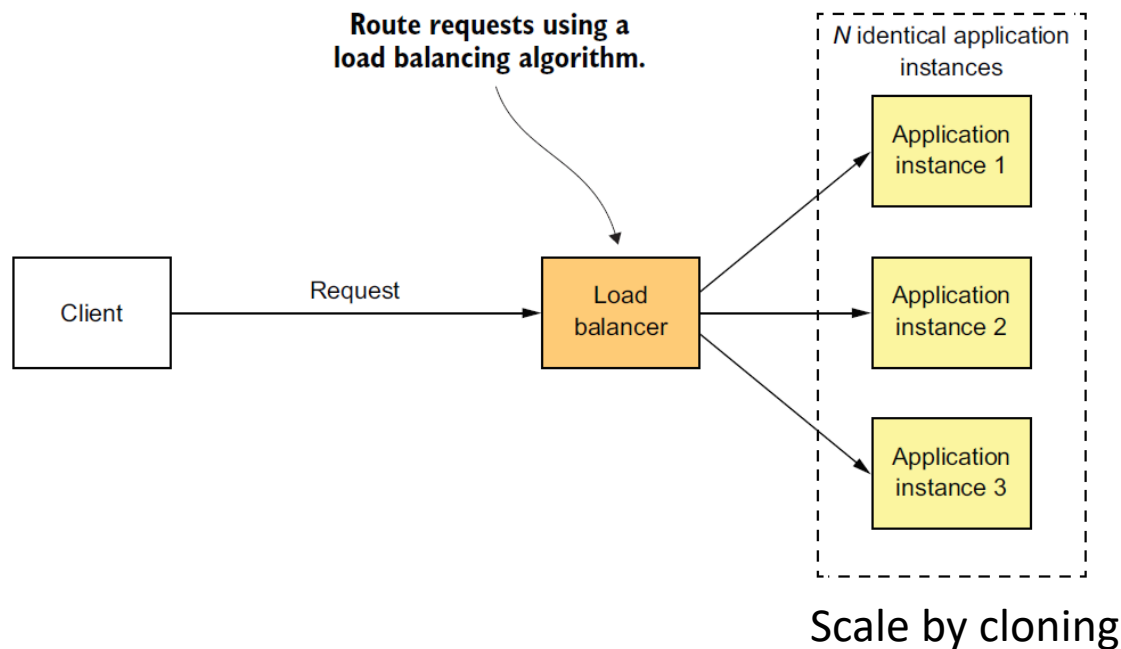
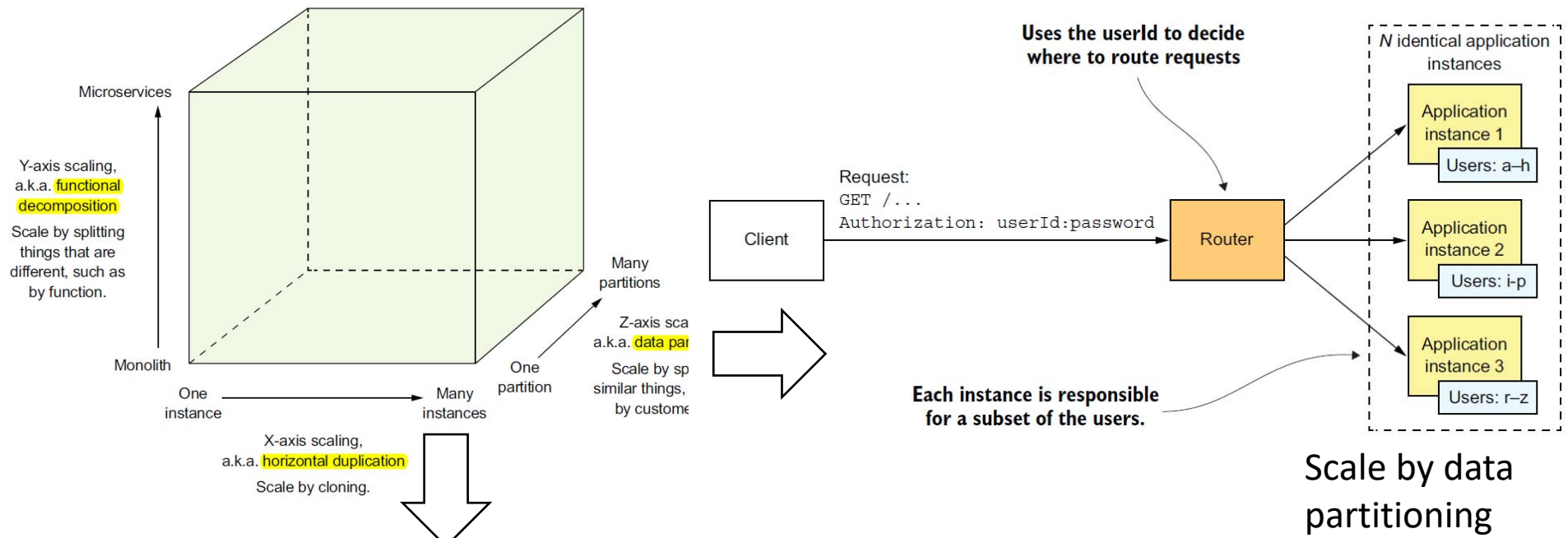


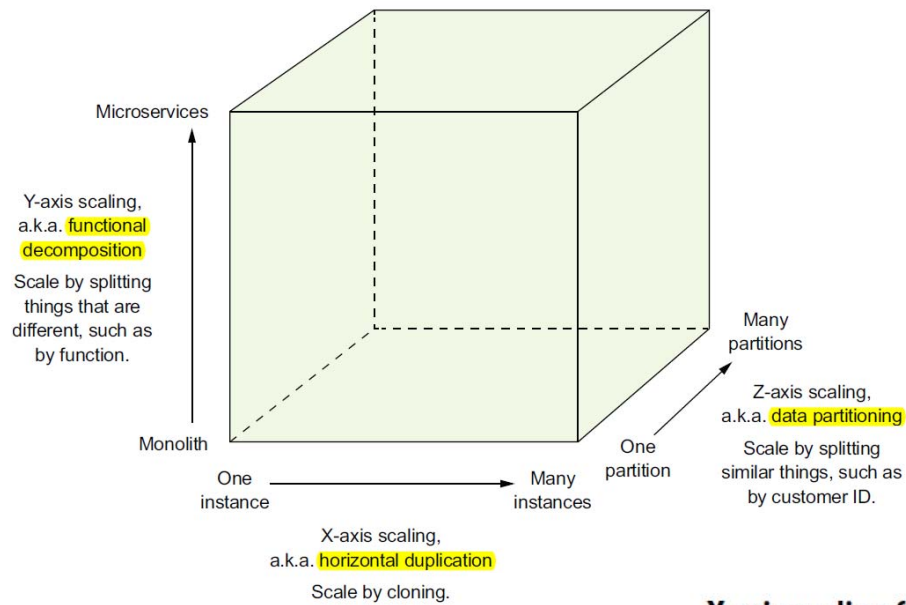
Monolithic Hell

- 前提: After the application grows
 - Development becomes slow
 - Path from commit to deployment is long and 費力的；困難的 arduous
 - Scaling becomes difficult
 - Locked into obsolete technology stack

The Scale Cube

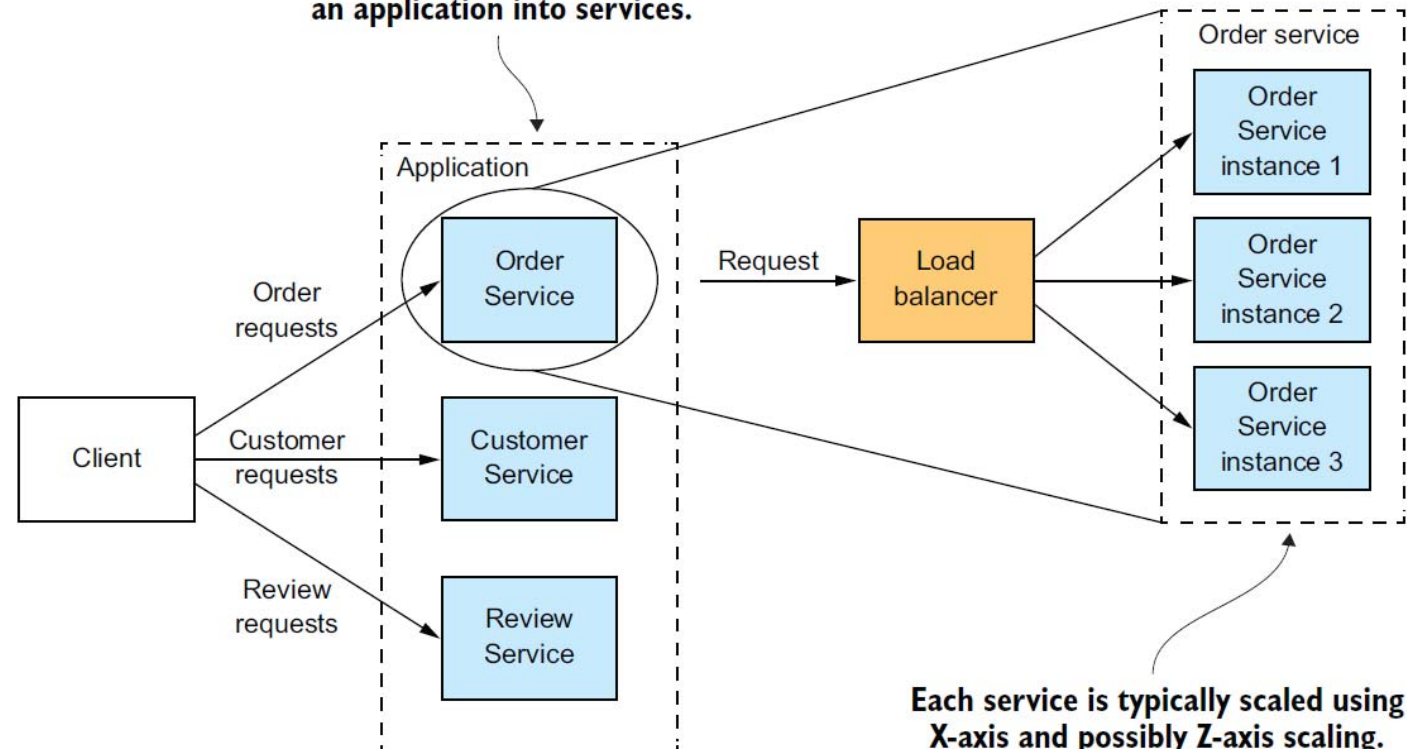




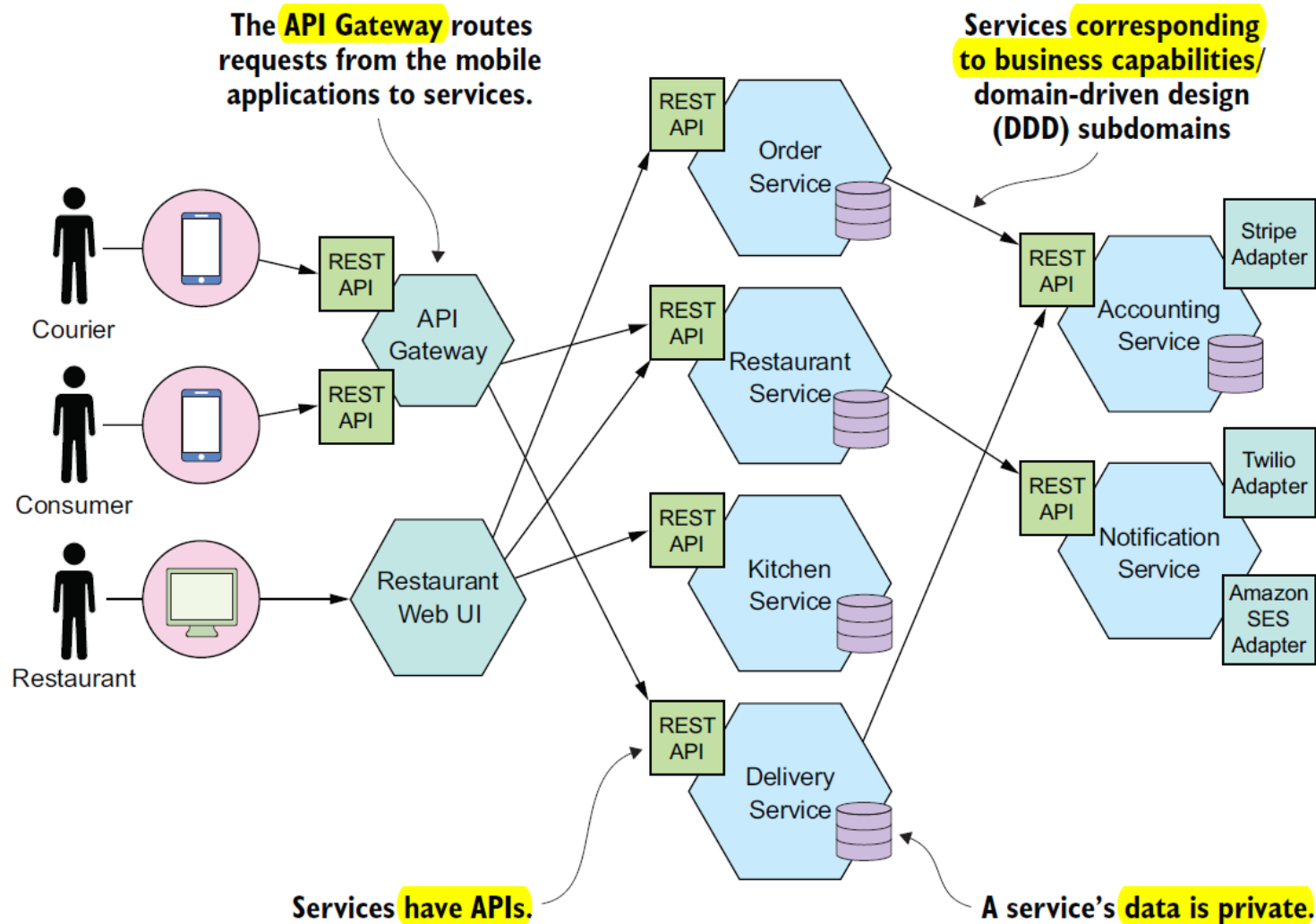


Microservice scaling
 Layer 1: Functional Decomposition (Y)
 Layer 2: Horizontal (X) + data (Z)

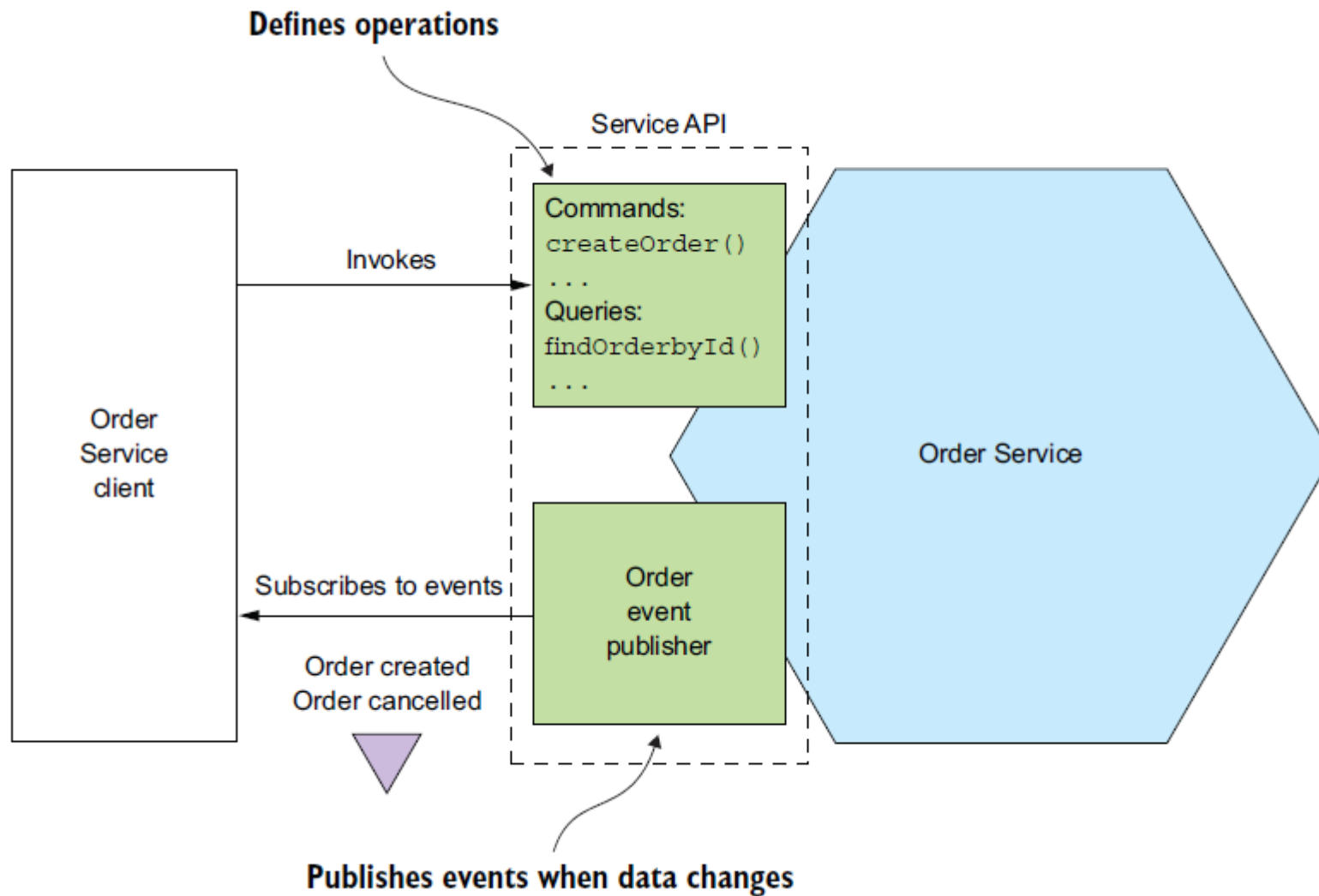
Y-axis scaling functionality decomposes an application into services.



FTGO Microservice Architecture



Service



Advantages of Microservices

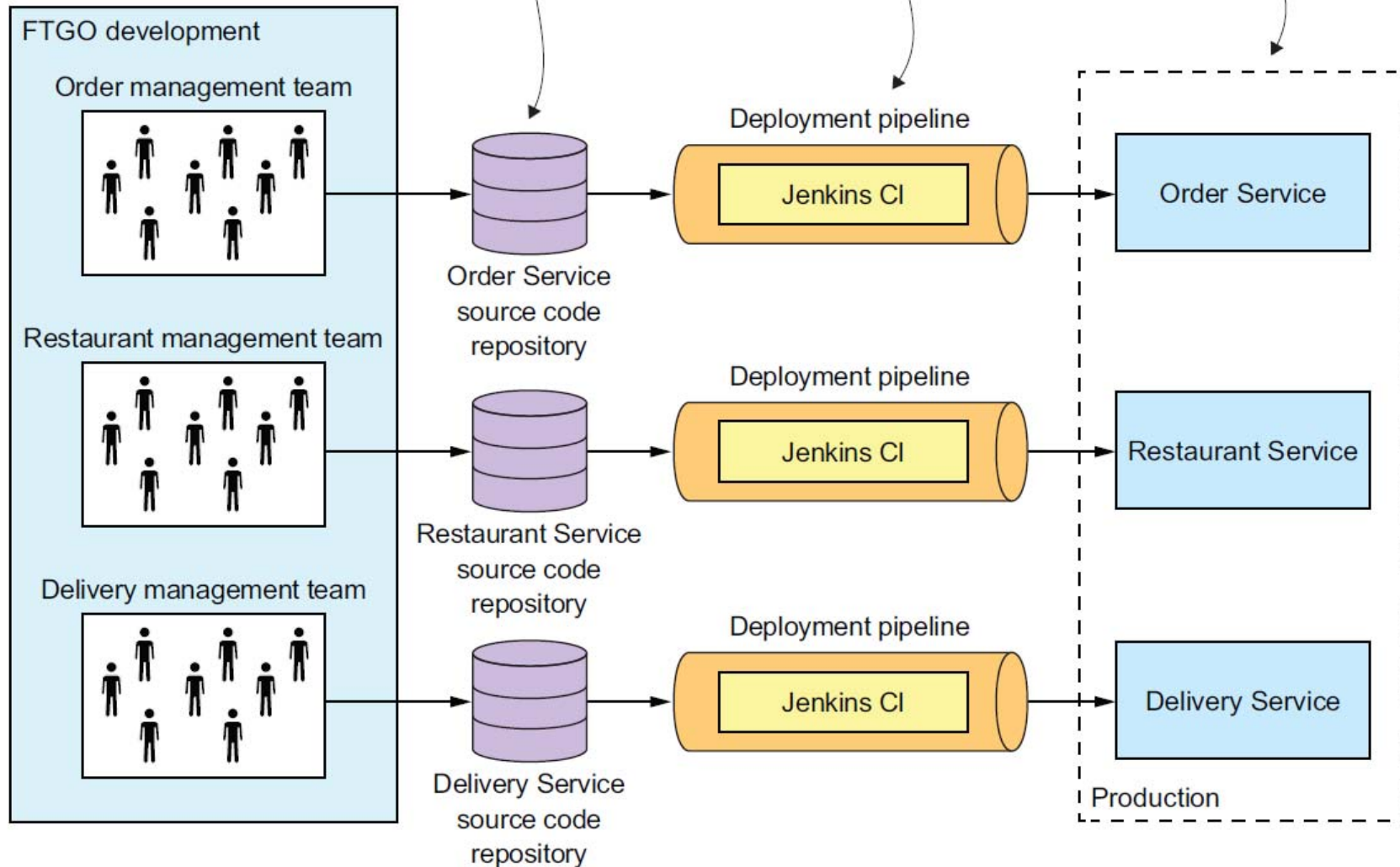
- Enables the continuous delivery and deployment of large, complex applications
- Services are small and easily maintained
- Services are independently deployable
- Services are independently scalable
- Enables teams to be autonomous
- Allows easy experimenting and adoption of new technologies
- Better fault isolation

**Small, autonomous,
loosely coupled teams**

**Each service has
its own source
code repository.**

**Each service has
its own automated
deployment pipeline.**

**Small, simple,
reliable, easy to
maintain services**

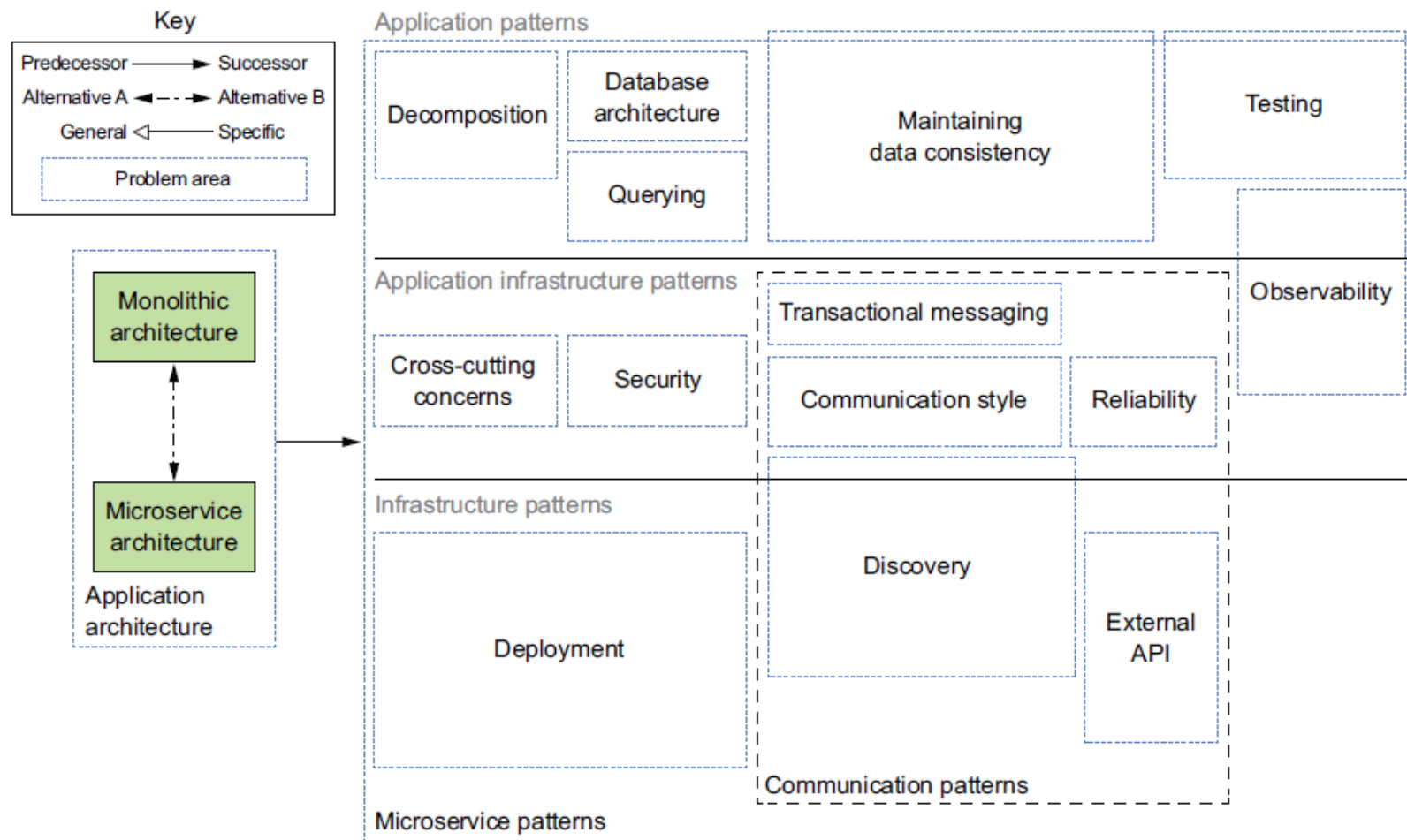


Drawbacks of Microservices

- Finding the right set of services is challenging
 - Bounded context
- Distributed systems are complex
 - CQRS and Sagas
- Deploying features that span multiple services requires careful coordination
- Deciding when to adopt the microservice architecture is difficult
 - For complex applications, it is always the right choice
 - Ex: consumer-facing web applications, SaaS applications

Microservices相關架構議題

<https://microservices.io/patterns/index.html>



Q & A