CSC12108

Distributed Application

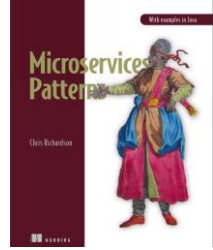
INTRODUCTION

Microservive Architecture

Instructor – Msc. PHAM MINH TU

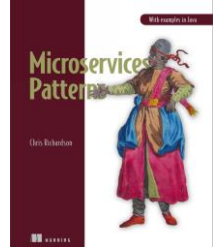
KHOA CÔNG NGHỆ THÔNG TIN
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

Outline



- Introduction
  - Case study
  - Microservice architecture
    - Benefits and drawbacks
- Decomposition strategies
  - Architectural styles
  - Defining services by applying the Decompose by business capability pattern
  - Defining services by applying the Decompose by sub-domain pattern
  - Decomposition guidelines

Outline

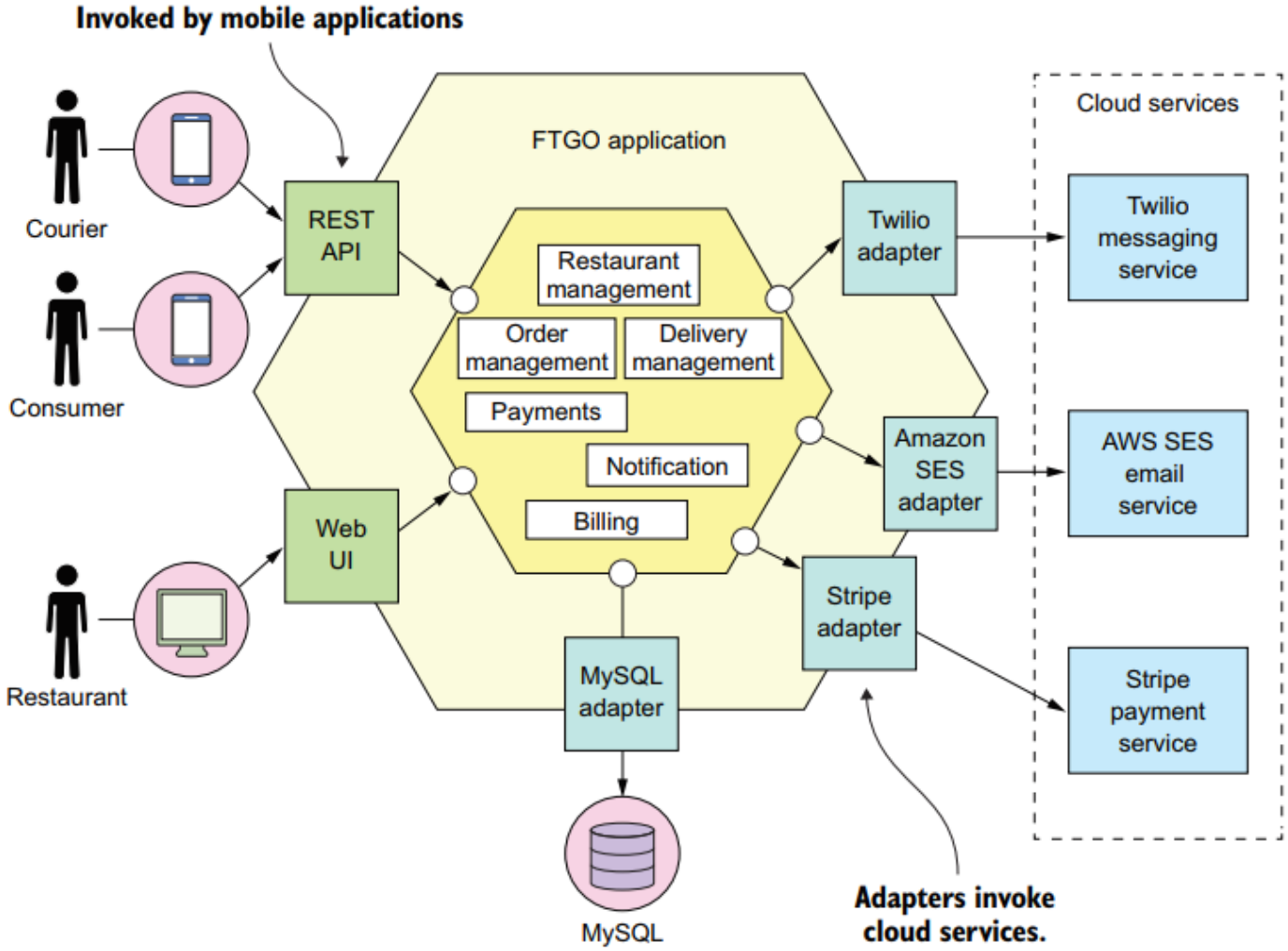


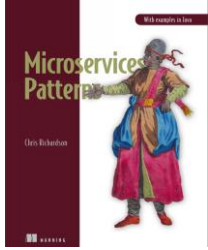
- Introduction
  - Case study
  - Microservice architecture
    - Benefits and drawbacks
- Decomposition strategies
  - Architectural styles
  - Defining services by applying the Decompose by business capability pattern
  - Defining services by applying the Decompose by sub-domain pattern
  - Decomposition guidelines

Introduction

Case study

FTGO



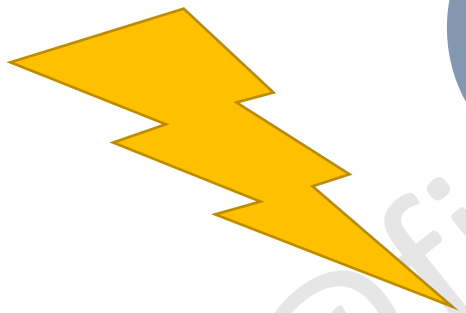


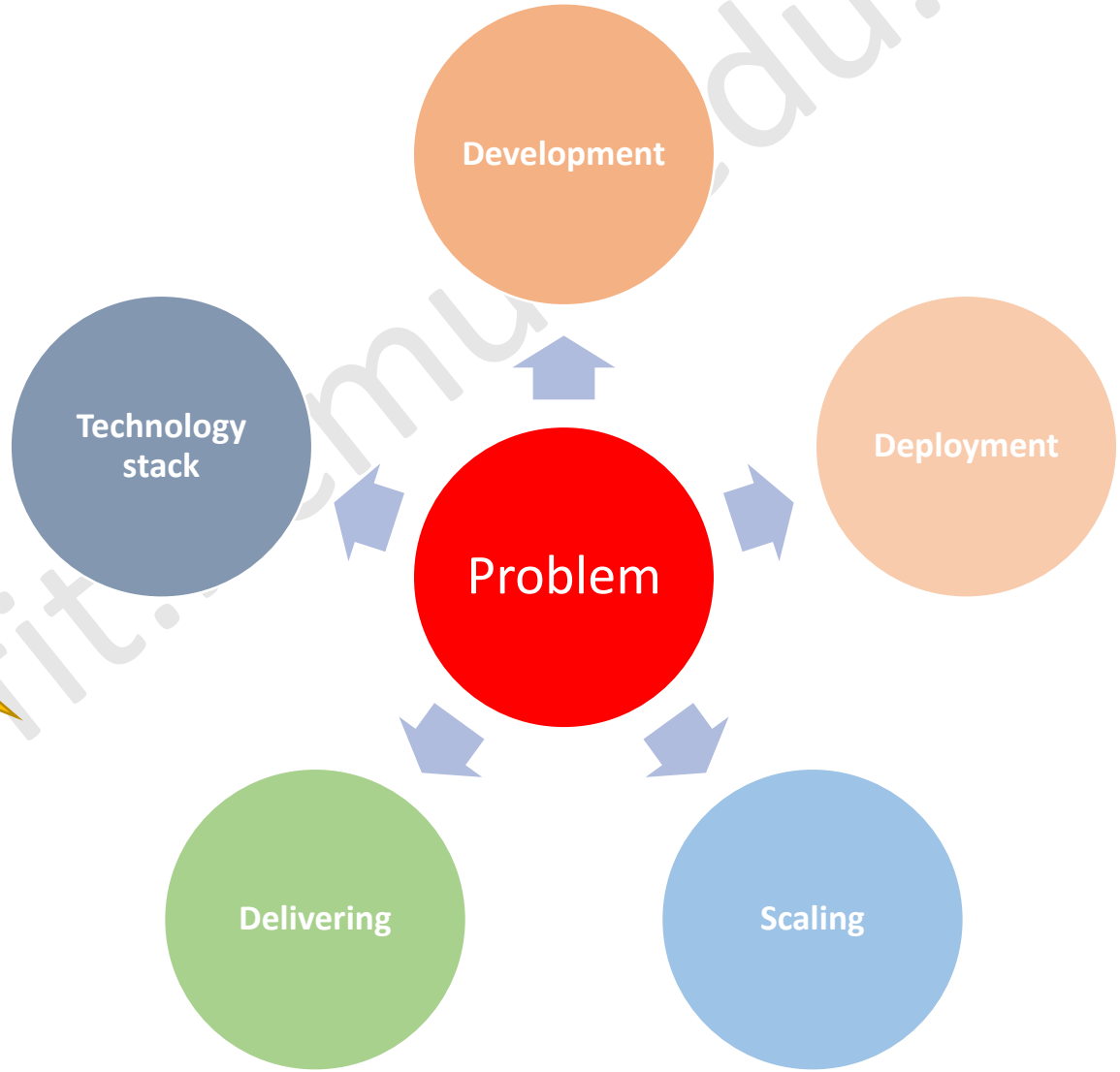
Introduction

- Case study
  - What is the main function in the architecture?
  - What are the benefits in architecture?

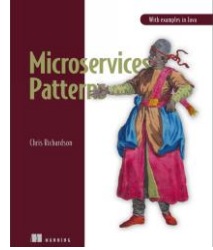
Introduction

Case study





Introduction

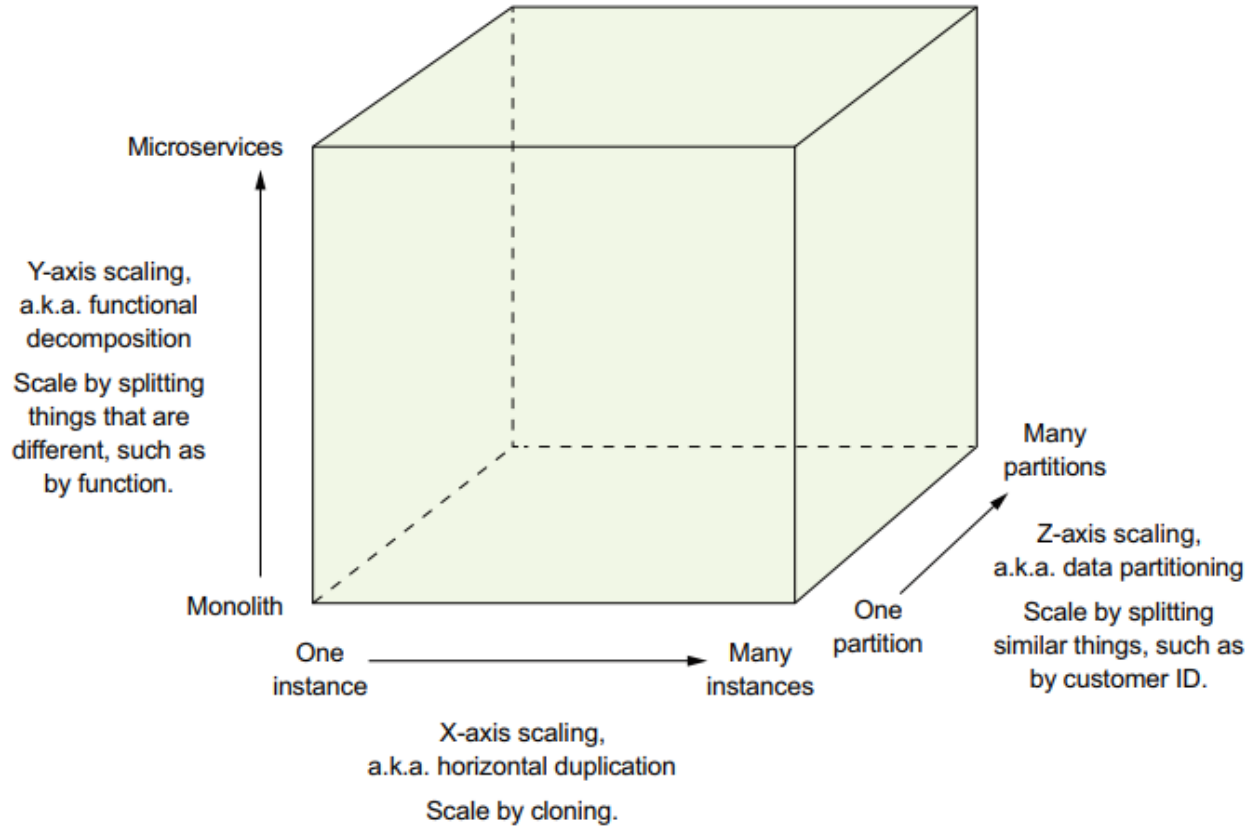


Microservice architecture

- Microservices is a specialization of an implementation approach for service-oriented architectures (SOA) used to build flexible, independently deployable software systems

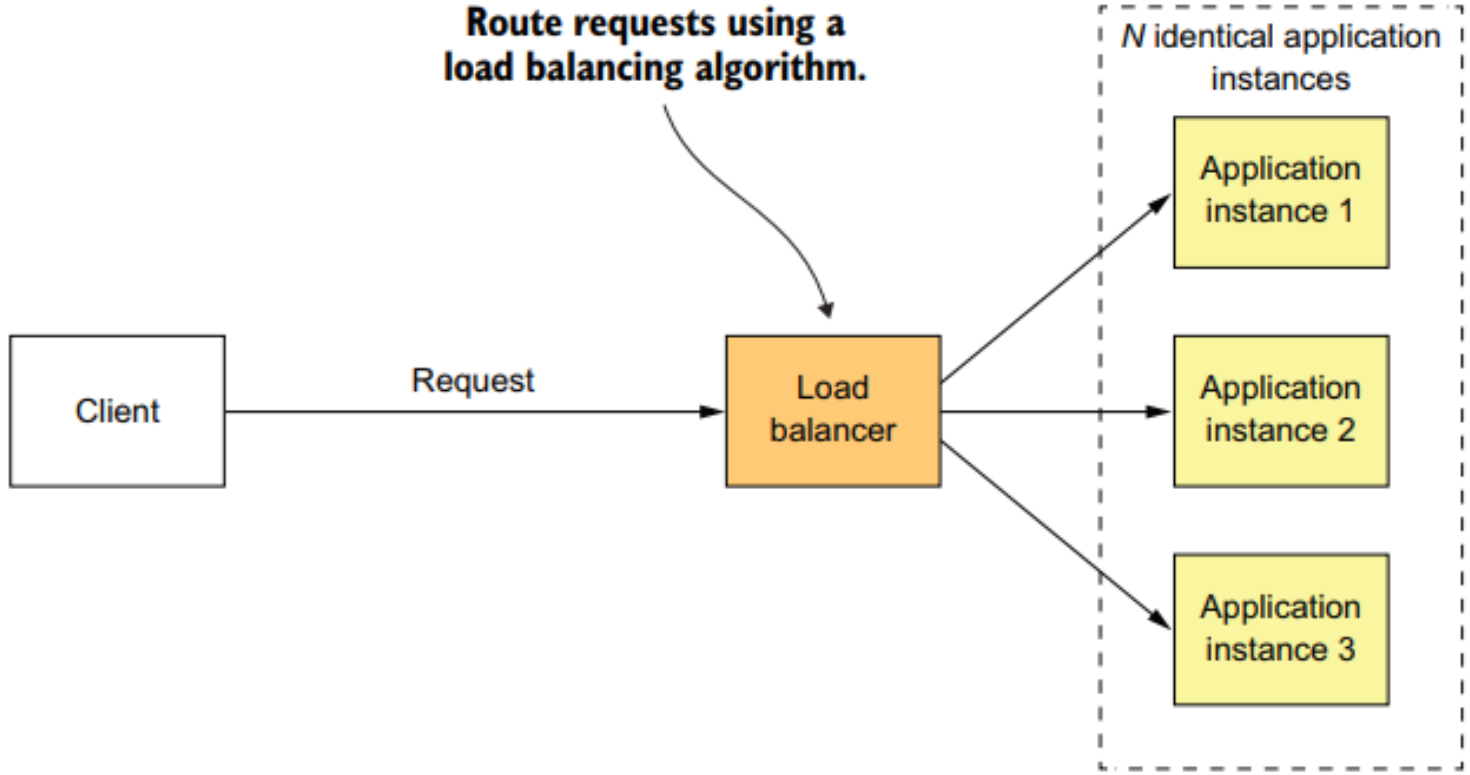
Introduction

- Microservice architecture
  - Scale cube - The Art of Scalability (Addison-Wesley, 2015)



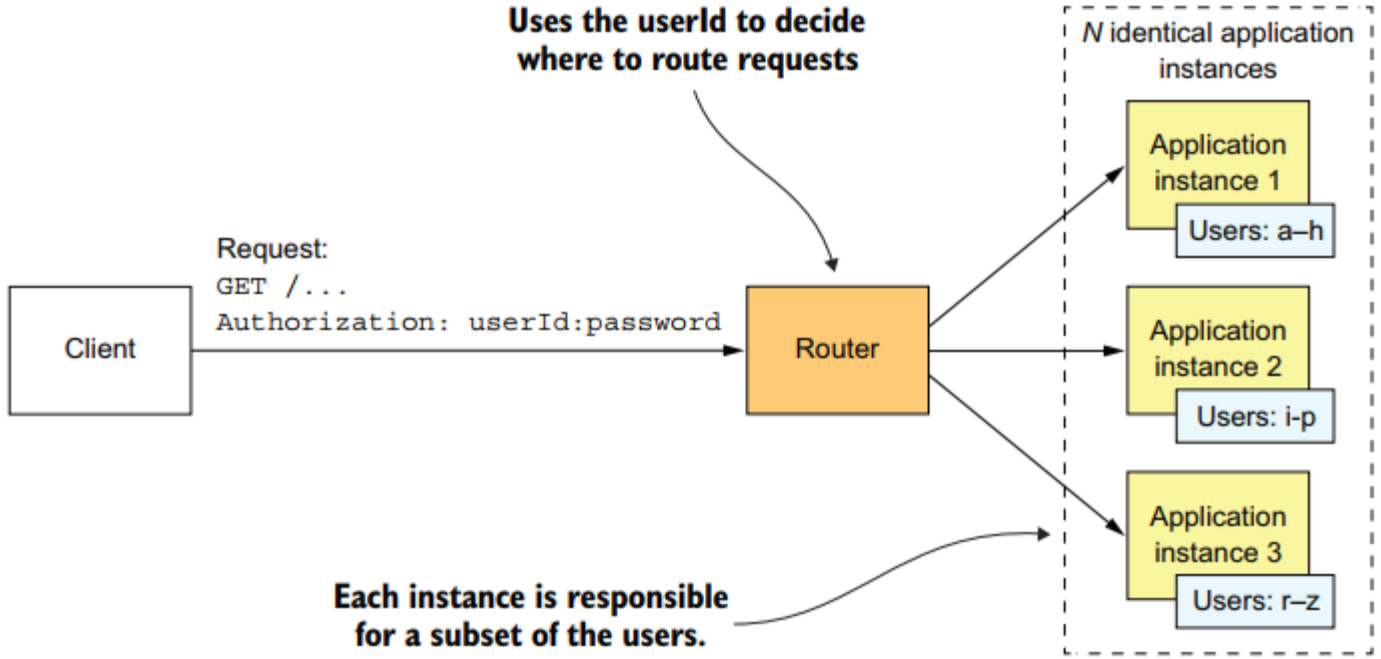
Introduction

- Microservice architecture
  - X-axis scaling load balances requests across multiple instances
    - Multiple instances of the application behind a load balancer



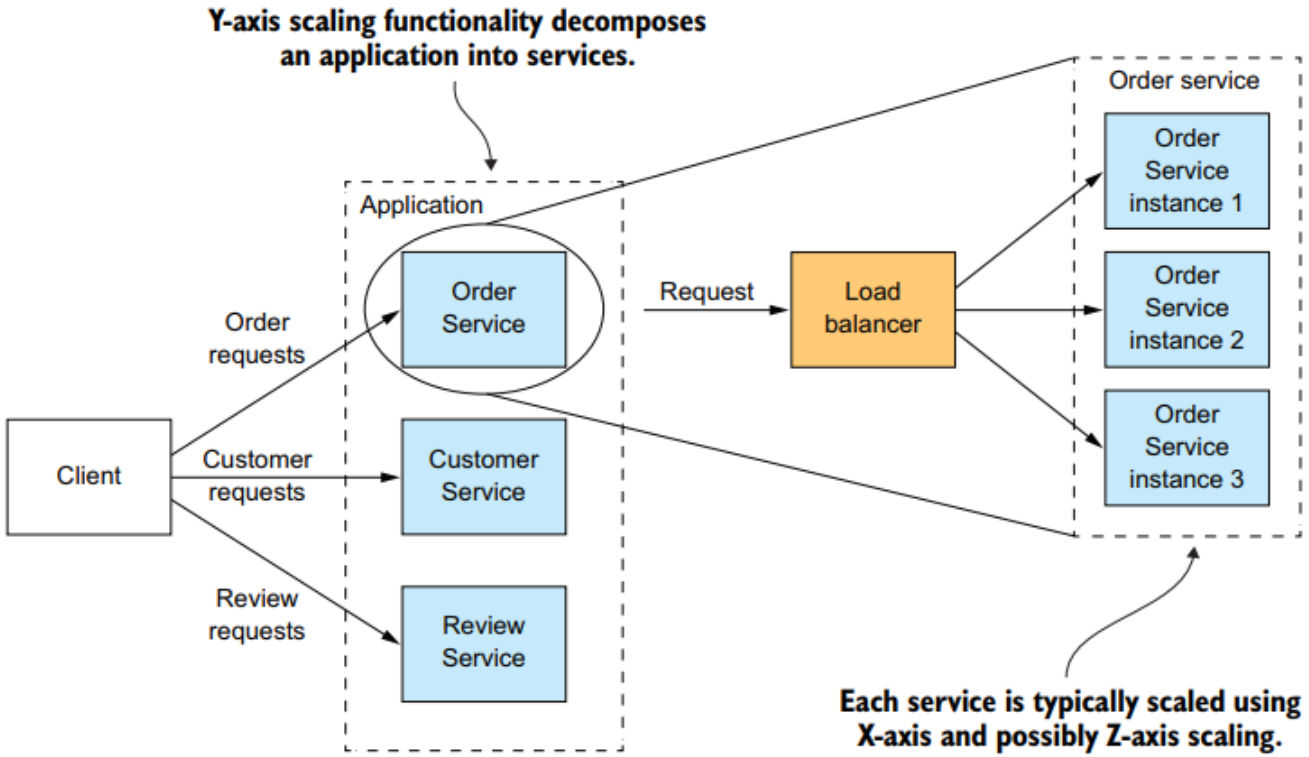
Introduction

- Microservice architecture
  - Z-axis scaling routes requests based on an attribute of the request
    - Each instance is responsible for only a subset of the data



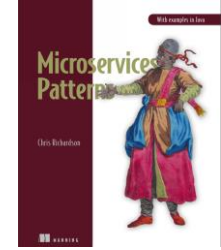
Introduction

- Microservice architecture
  - Y-axis scaling functionally decomposes an application into services
    - A service is a mini application that implements narrowly focused functionality
    - Functionally decomposes an application into a set of services



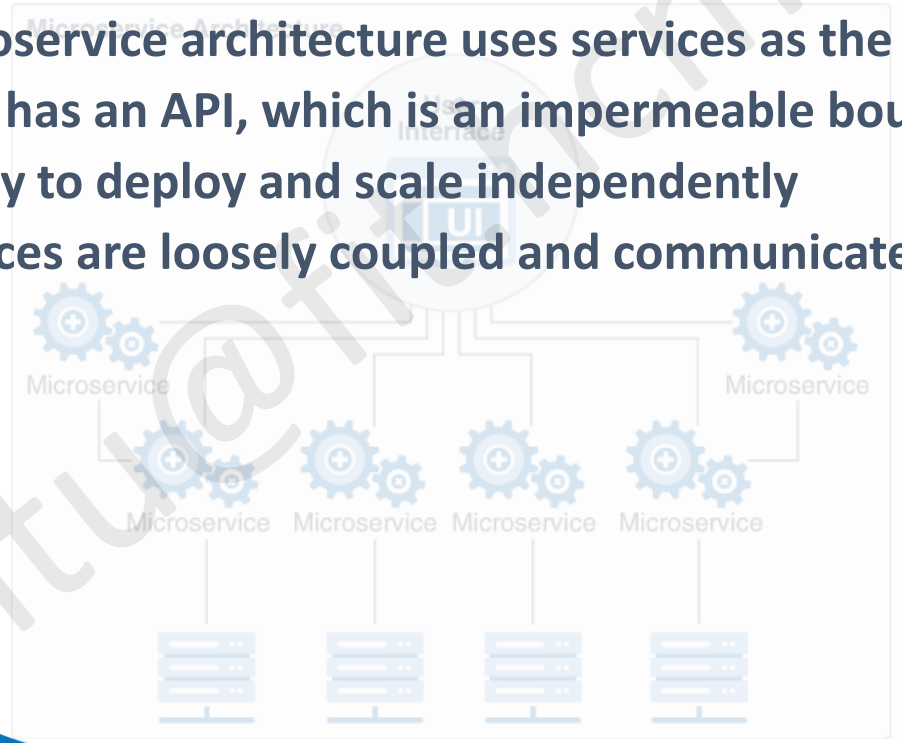
Introduction

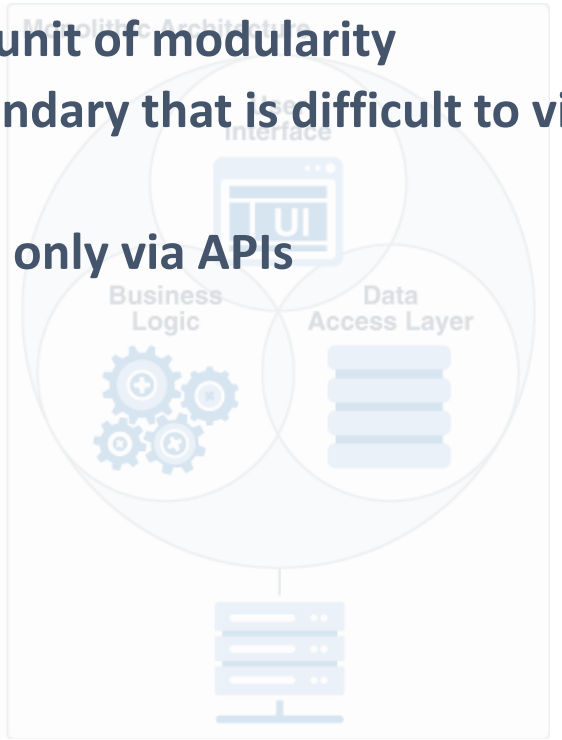
- Microservice architecture
  - What is benefit of X-Y-Z axis scaling?



Introduction

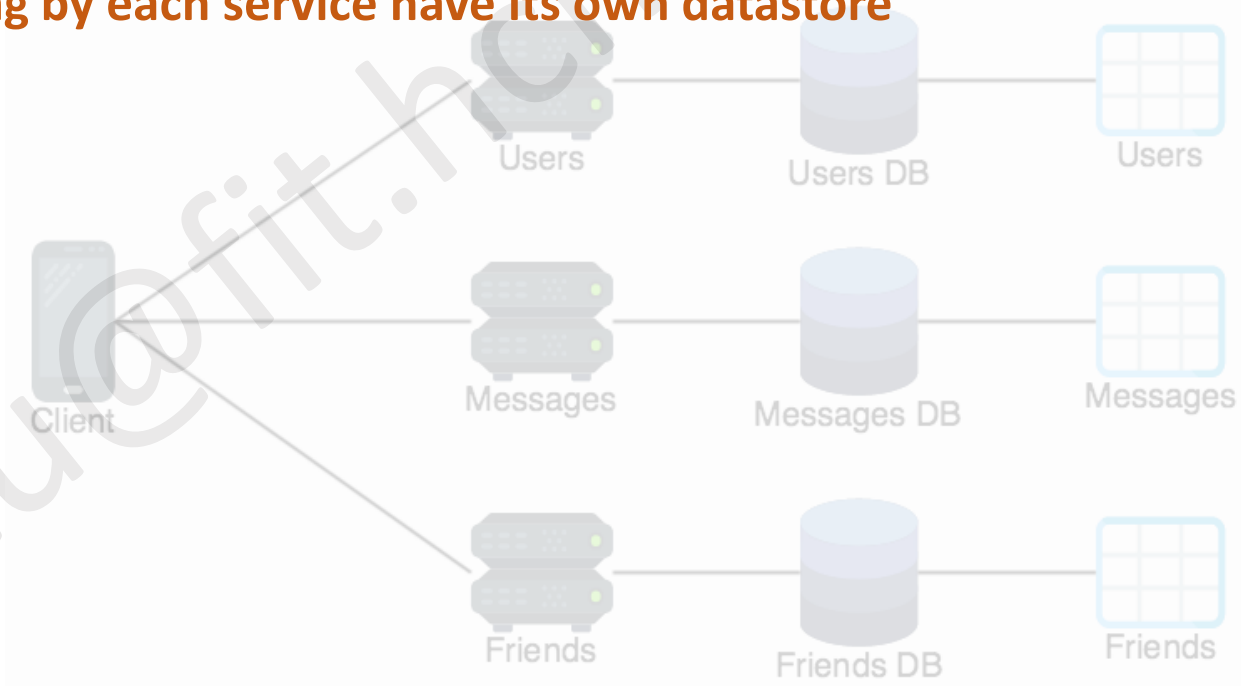
- Microservice architecture
  - Microservices as a form of modularity
    - The microservice architecture uses services as the unit of modularity
    - A service has an API, which is an impermeable boundary that is difficult to violate
    - The ability to deploy and scale independently
    - The services are loosely coupled and communicate only via APIs





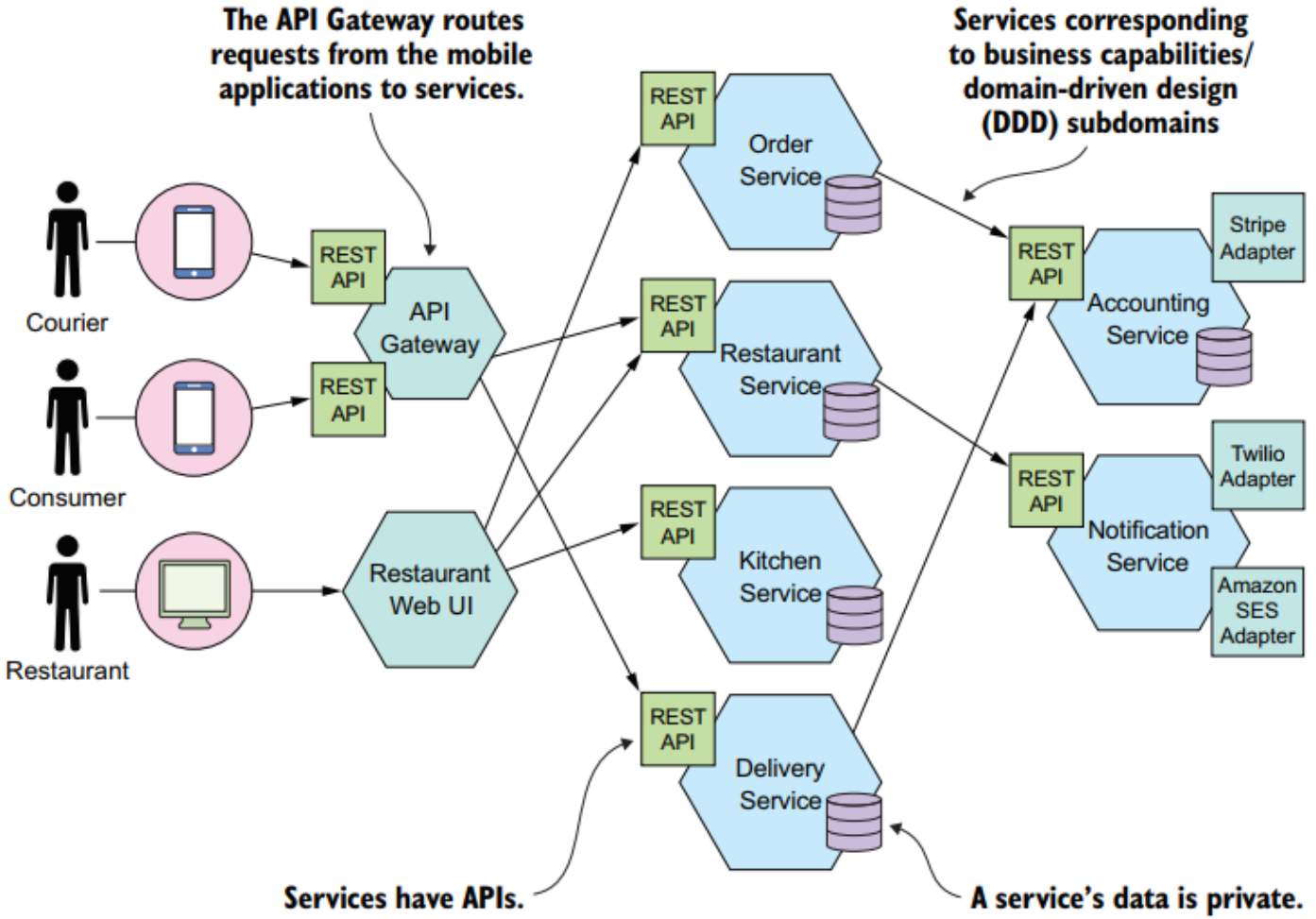
Introduction

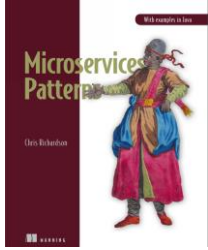
- Microservice architecture
  - Each service has its own database
    - Loose coupling by each service have its own datastore



Introduction

- Microservice architecture
  - Microservice for FTGO





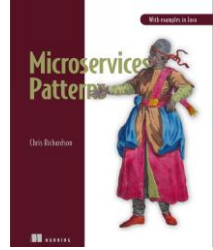
Introduction

Microservice architecture

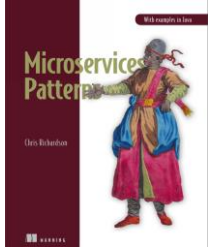
Microservice architecture and SOA

	SOA	Microservices
Inter-service communication	Smart pipes, such as Enterprise Service Bus, using heavyweight protocols, such as SOAP and the other WS* standards.	Dumb pipes, such as a message broker, or direct service-to-service communication, using lightweight protocols such as REST or gRPC
Data	Global data model and shared databases	Data model and database per service
Typical service	Larger monolithic application	Smaller service

Introduction



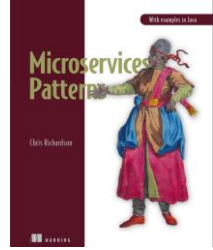
- Microservice architecture
  - Benefits
    - continuous delivery and deployment of large, complex applications.
    - Services are small and easily maintained.
    - Services are independently deployable.
    - Services are independently scalable.
    - The microservice architecture enables teams to be autonomous.
    - It allows easy experimenting and adoption of new technologies.
    - It has better fault isolation.



Introduction

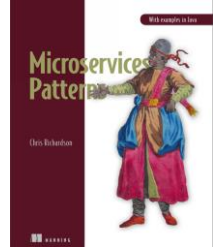
- Microservice architecture
  - Drawbacks

Introduction



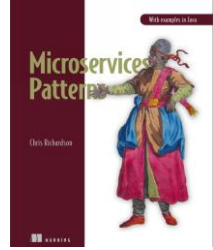
- Microservice architecture
  - Drawbacks
    - Finding the right set of services is challenging.
    - Distributed systems are complex, which makes development, testing, and deployment difficult.
    - Deploying features that span multiple services requires careful coordination.
    - Deciding when to adopt the microservice architecture is difficult.

Introduction



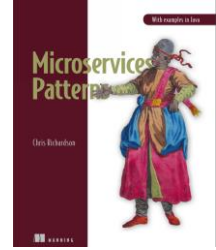
- Microservice architecture
  - The Microservice architecture pattern language
    - A pattern is a reusable solution to a problem that occurs in a particular context
    - A commonly used pattern structure includes three especially valuable sections:
      - Forces
      - Resulting context
      - Related patterns

Introduction



- Microservice architecture
  - The Microservice architecture pattern language
    - Forces
      - The forces section of a pattern describes the forces (issues) that you must address when solving a problem in a given context
    - Resulting context
      - The resulting context section of a pattern describes the consequences of applying the pattern
        - Benefits
        - Drawbacks
        - Issues
    - Related patterns
      - The related patterns section of a pattern describes the relationship between the pattern and other patterns

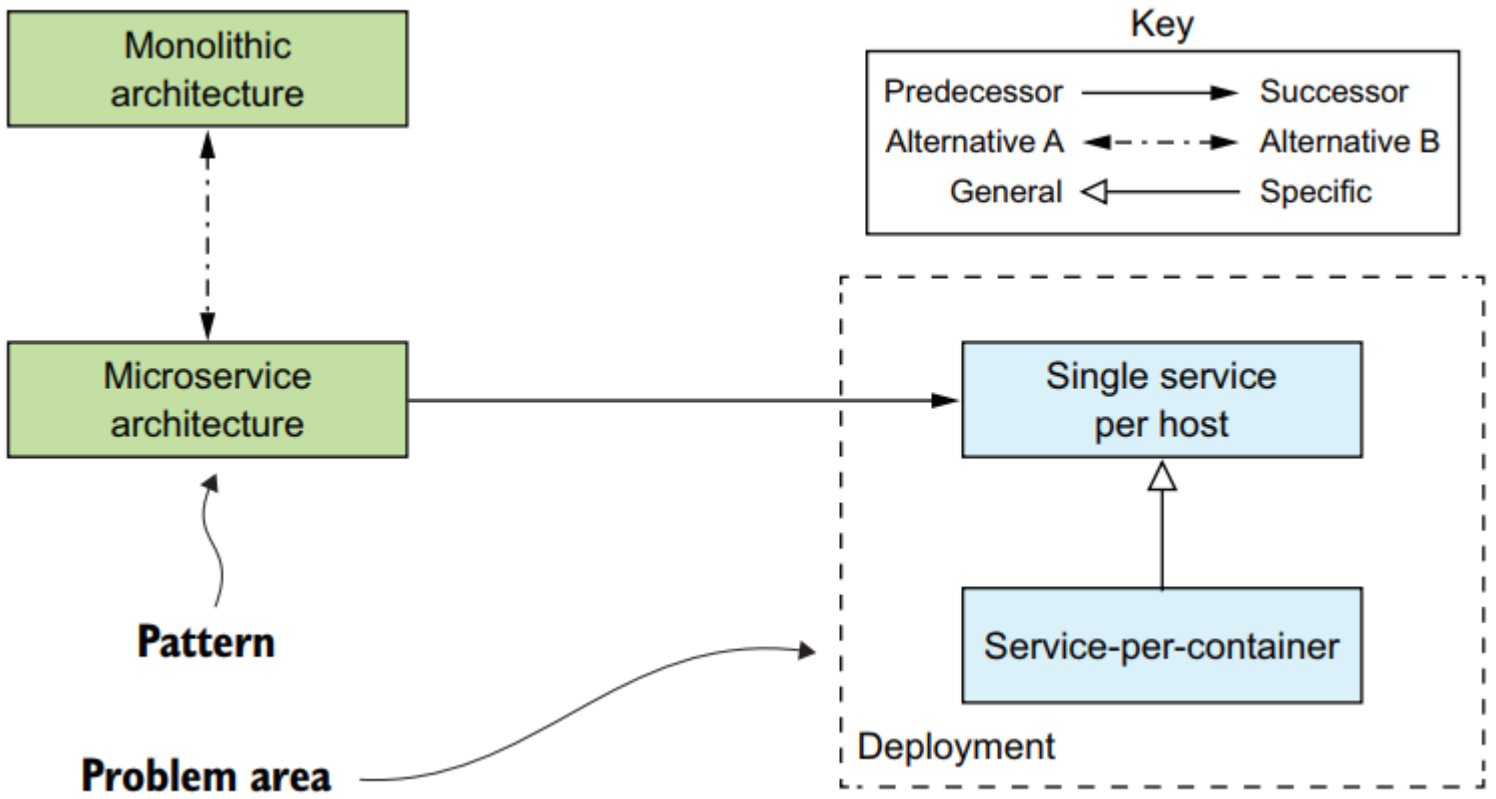
Introduction



- Microservice architecture
  - The Microservice architecture pattern language
    - Related patterns
      - Predecessor: A predecessor pattern is a pattern that motivates the need for this pattern
      - Successor—A pattern that solves an issue that has been introduced by this pattern.
      - Alternative—A pattern that provides an alternative solution to this pattern.
      - Generalization—A pattern that is a general solution to a problem.
      - Specialization—A specialized form of a particular pattern

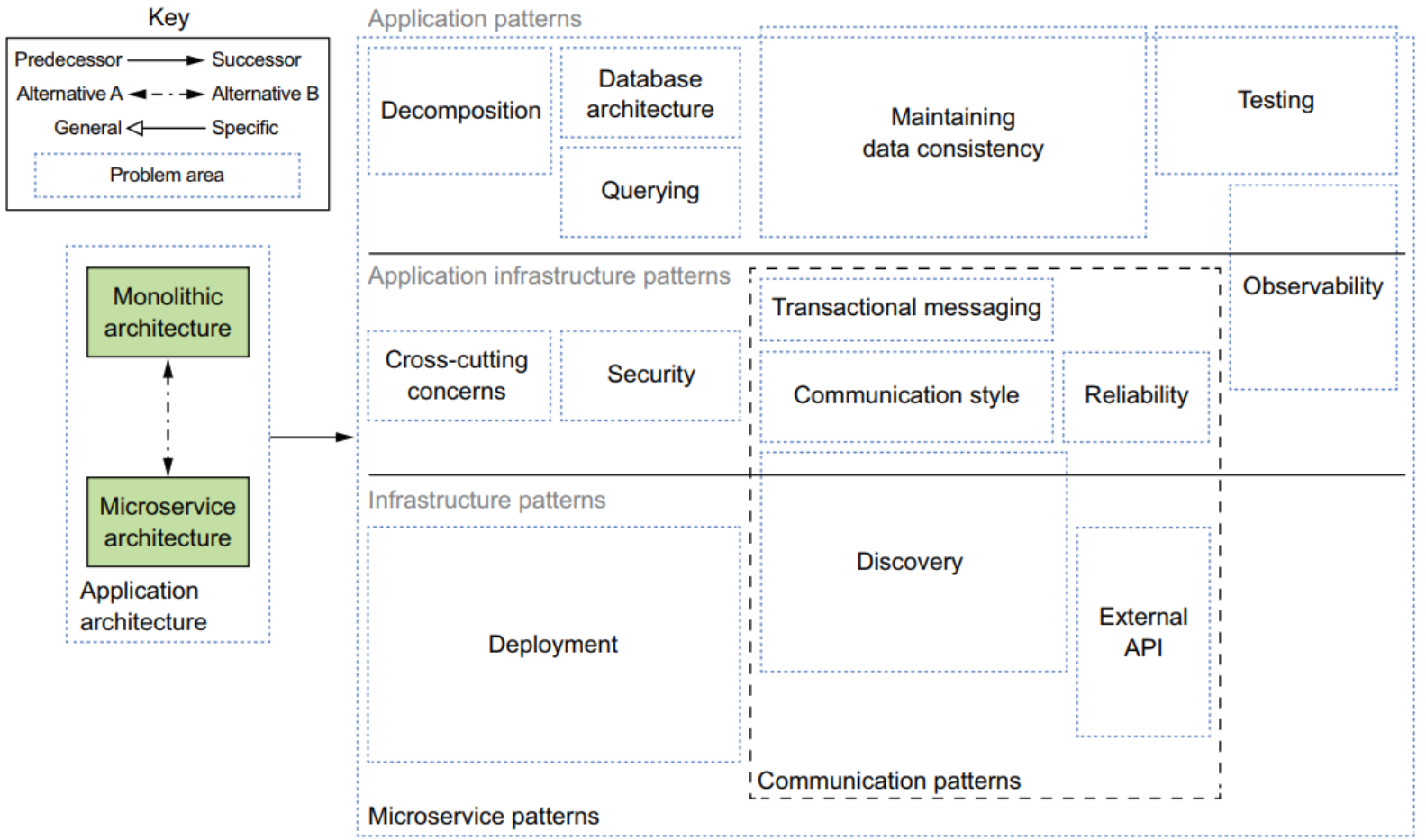
Introduction

- Microservice architecture
  - The Microservice architecture pattern language
    - Example

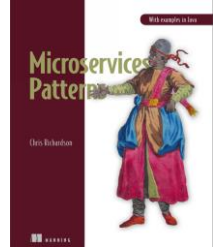


Introduction

- Microservice architecture
  - The Microservice architecture pattern language



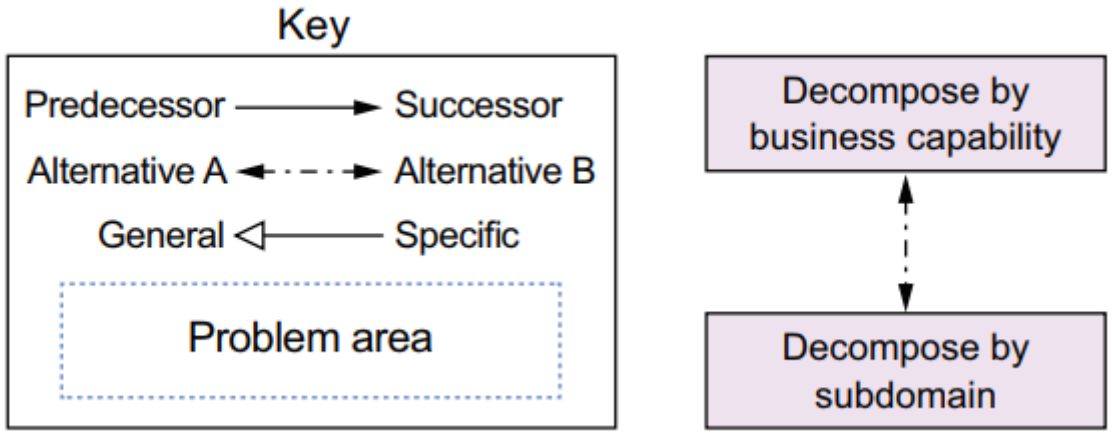
Introduction



- Microservice architecture
  - The Microservice architecture pattern language
    - Infrastructure patterns —These solve problems that are mostly infrastructure issues outside of development.
    - Application infrastructure —These are for infrastructure issues that also impact development.
    - Application patterns—These solve problems faced by developers.

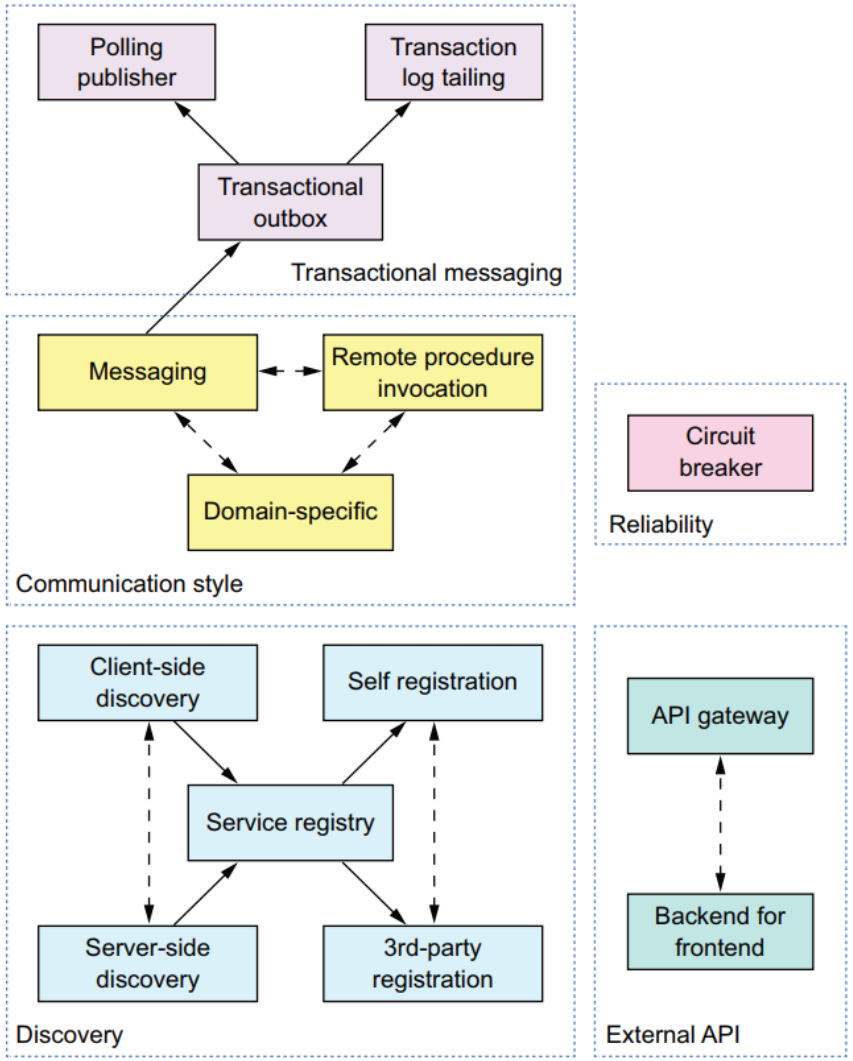
Introduction

- Microservice architecture
  - Patterns for decomposing an application into services



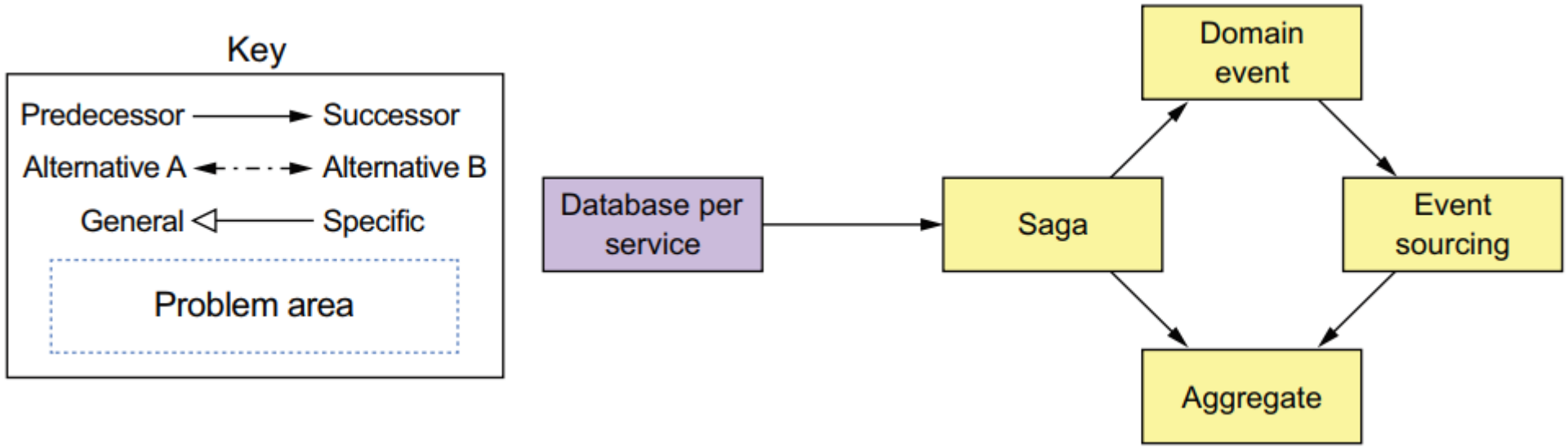
Introduction

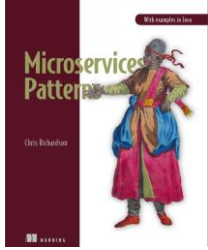
- Microservice architecture
  - Communication patterns
    - Communication style—What kind of IPC mechanism should you use?
    - Discovery—How does a client of a service determine the IP address of a service instance so that, for example, it makes an HTTP request?
    - Reliability—How can you ensure that communication between services is reliable even though services can be unavailable?
    - Transactional messaging—How should you integrate the sending of messages and publishing of events with database transactions that update business data?
    - External API—How do clients of your application communicate with the services?



Introduction

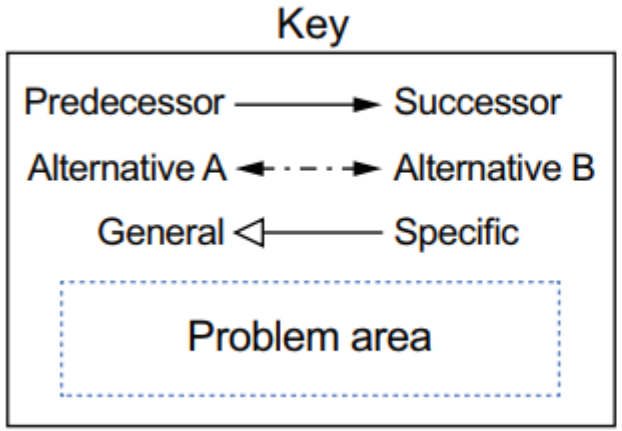
- Microservice architecture
  - Data consistency patterns for implementing transaction management

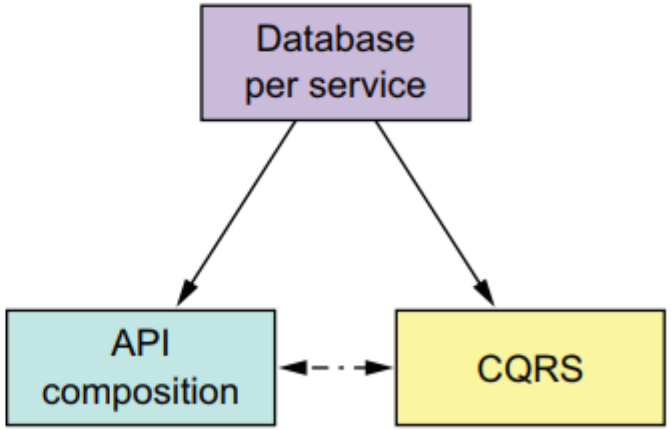




Introduction

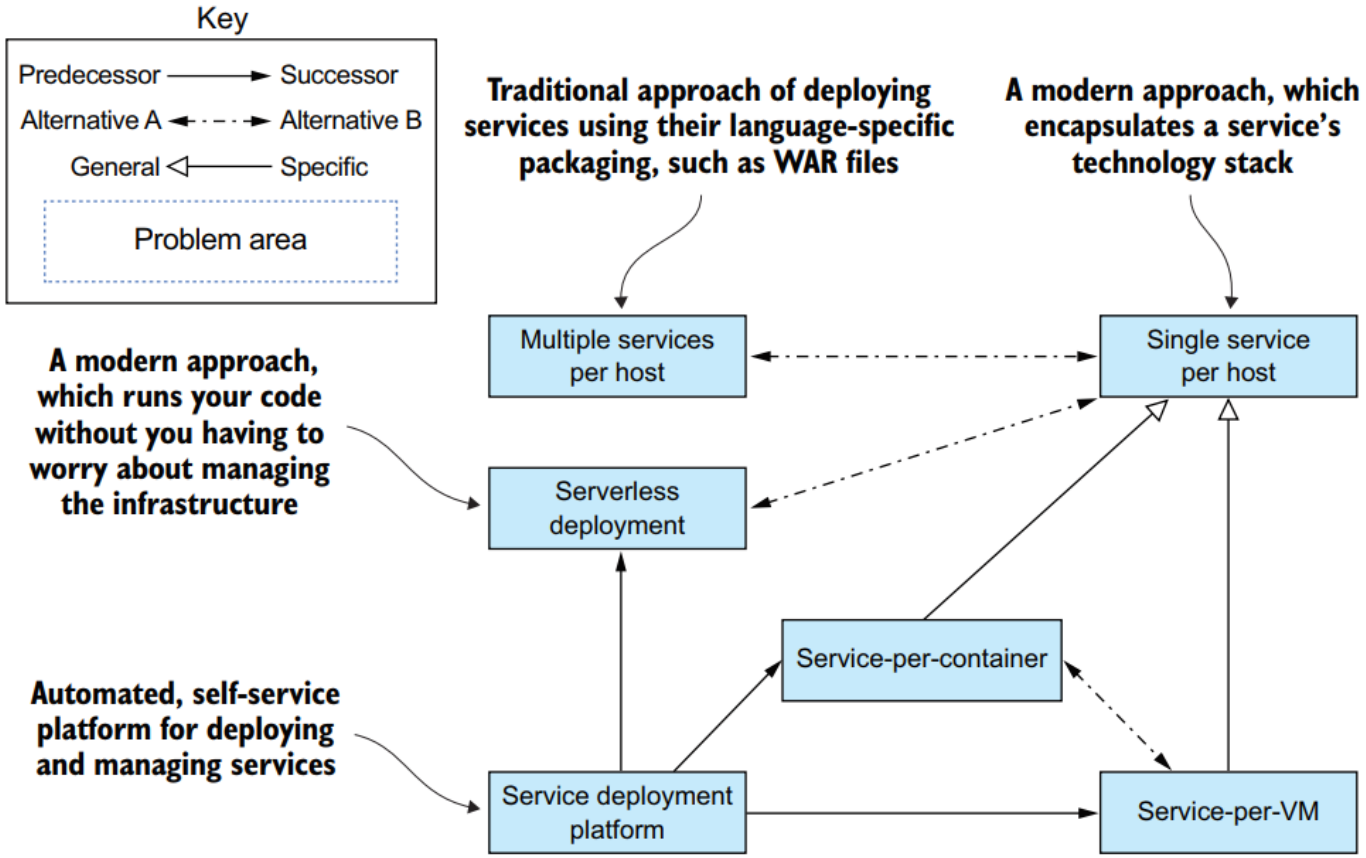
- Microservice architecture
  - Patterns for querying data in a microservice architecture



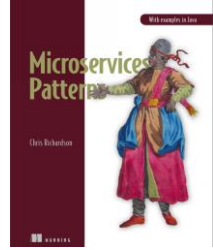


Introduction

- Microservice architecture
  - Service deployment patterns



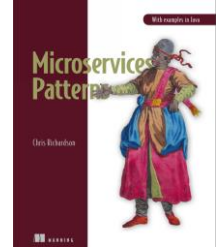
Introduction

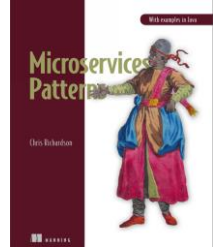


- Microservice architecture
  - Observability patterns provide insight into application behavior
  - Patterns for the automated testing of services
  - Patterns for handling cross-cutting concerns
  - Security patterns

Outline

- Introduction
  - Case study
  - Microservice architecture
    - Benefits and drawbacks
- **Define Application's Microservice Architecture**
  - **Architectural styles**
  - **Defining services by applying the Decompose by business capability pattern**
  - **Defining services by applying the Decompose by sub-domain pattern**
  - **Decomposition guidelines**

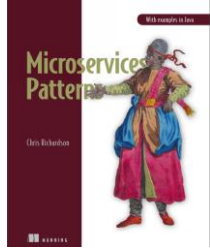




Define Application's Microservice Architecture

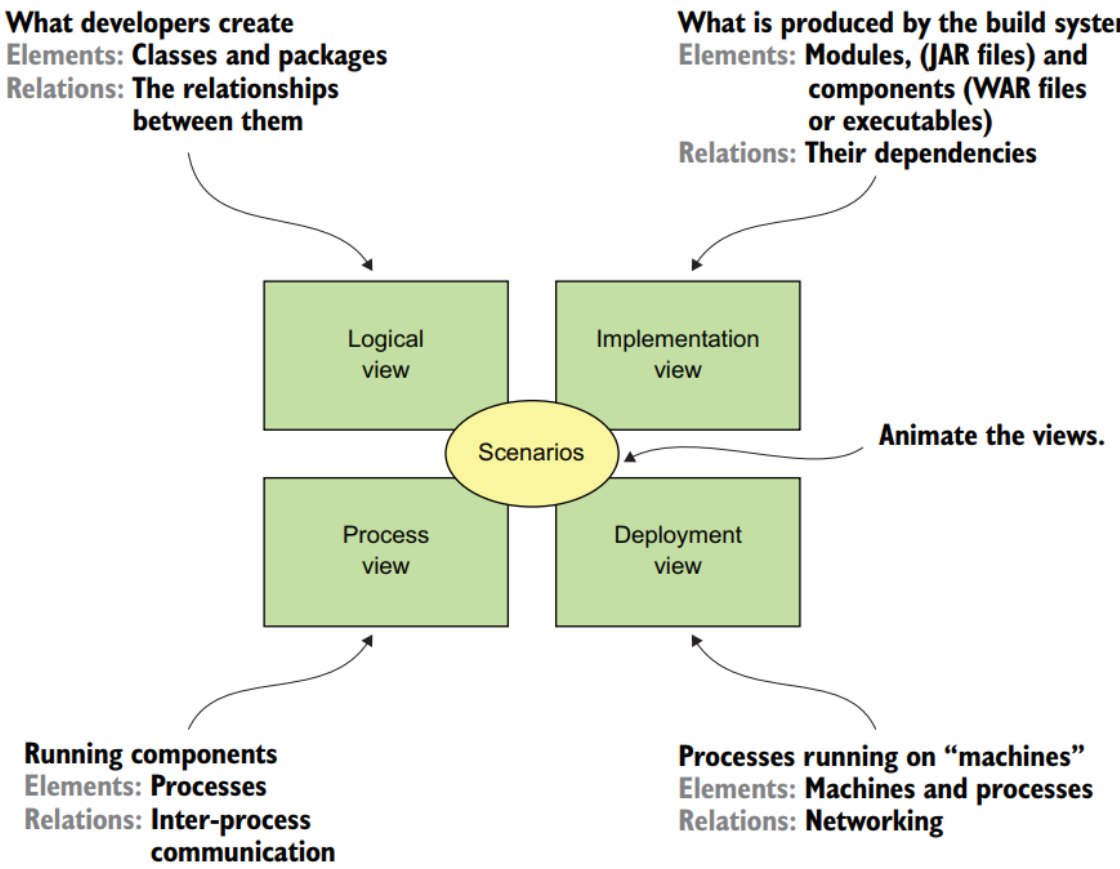
- Architectural styles
  - A definition of software architecture

> The software architecture of a computing system is the set of structures needed to reason about the system, which comprise **software elements**, **relations** among them, and properties of both



Define Application's Microservice Architecture

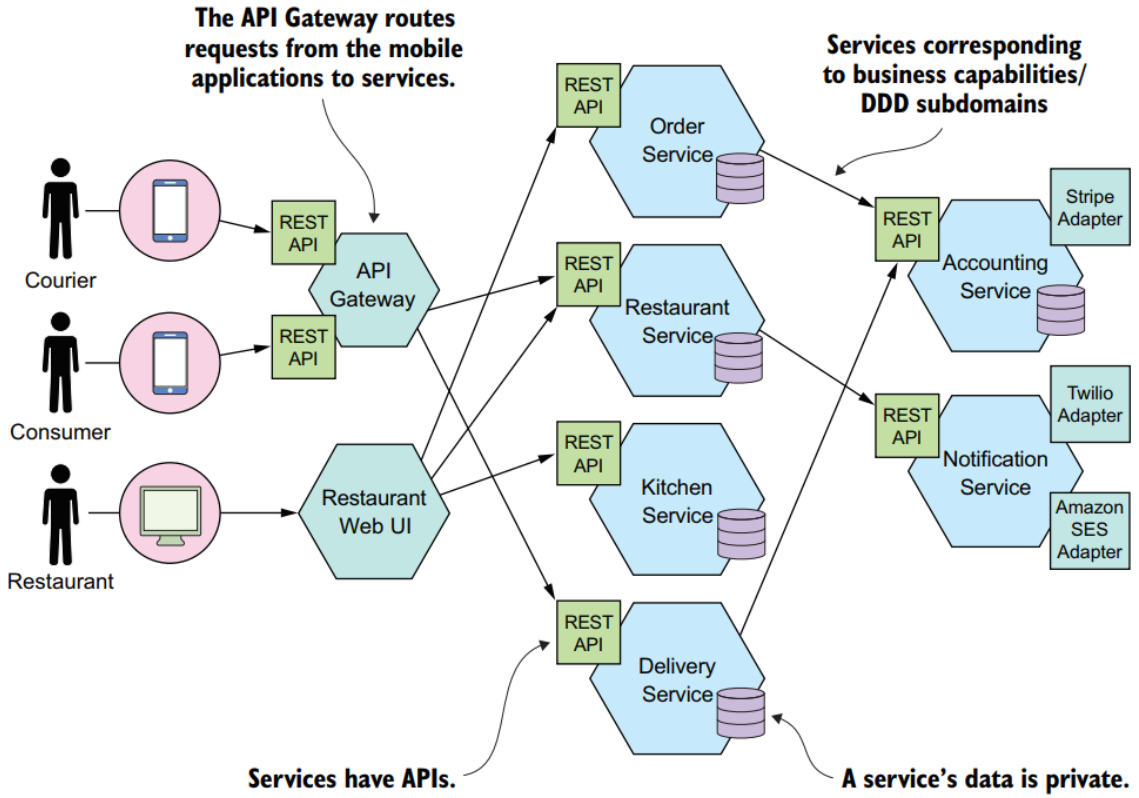
- Architectural styles
  - The 4+1 view model of software architecture

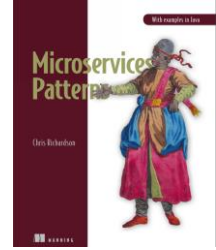


Define Application's Microservice Architecture

- Architectural styles
  - Microservice architecture
    - What is a service?

A service is a standalone, independently deployable software component that implements some useful functionality

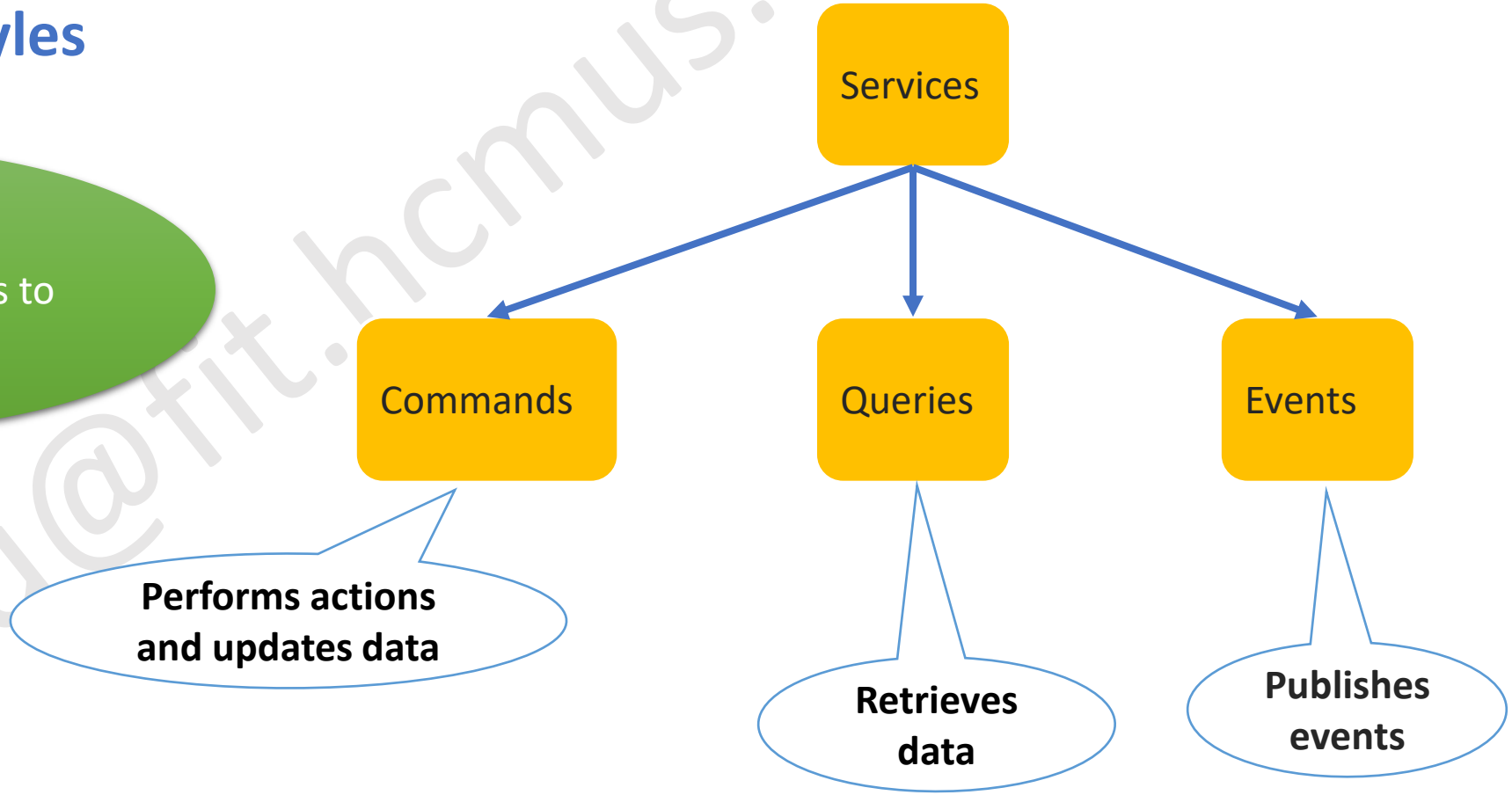




Define Application's Microservice Architecture

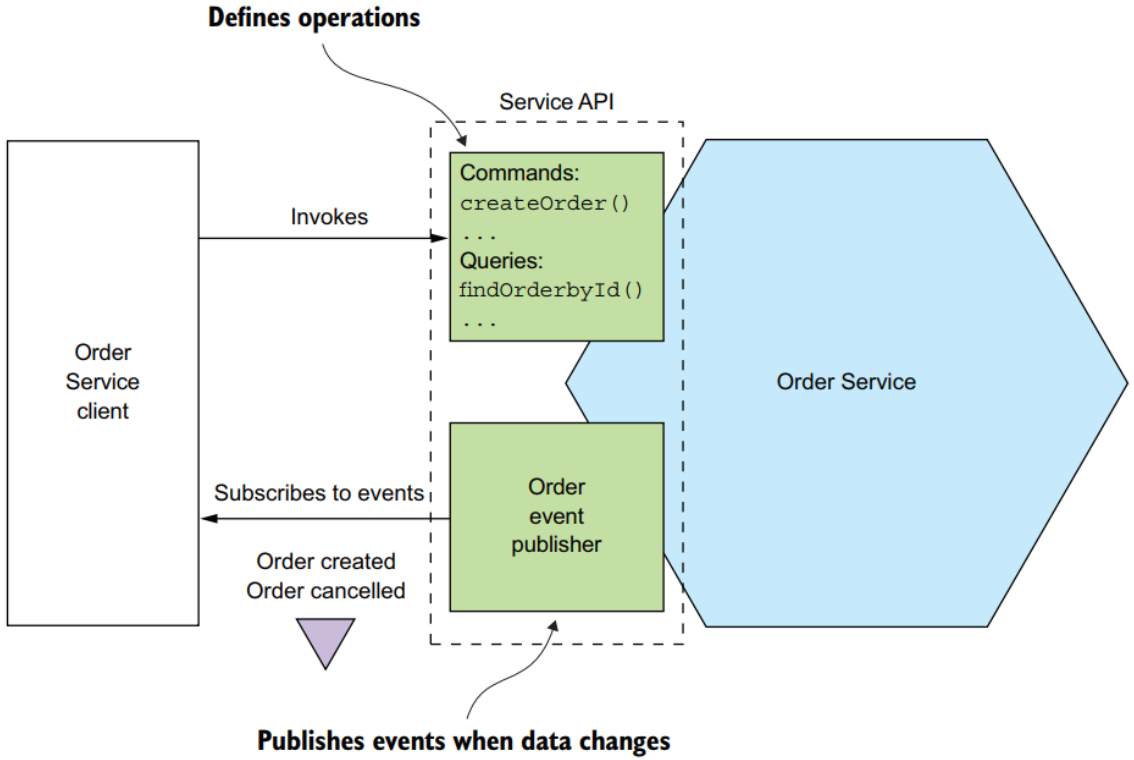
❑Architectural styles

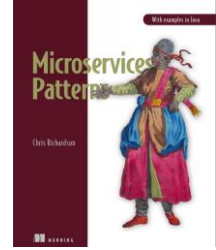
A service has an API that provides its clients access to its functionality



Define Application's Microservice Architecture

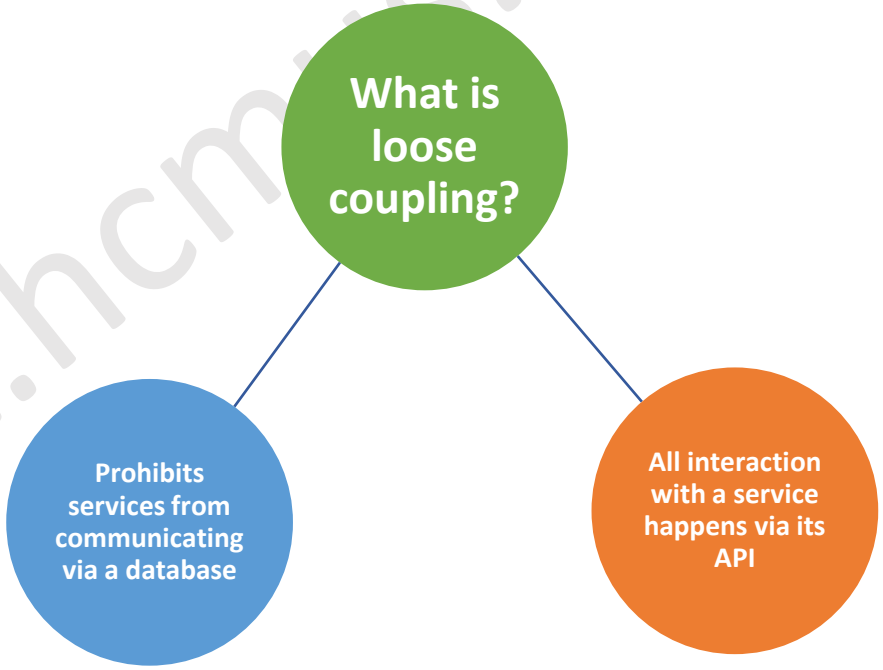
❑Architectural styles

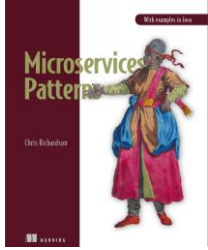




Define Application's Microservice Architecture

- Architectural styles
  - Loosely coupled services
    - Maintainability
    - Testability





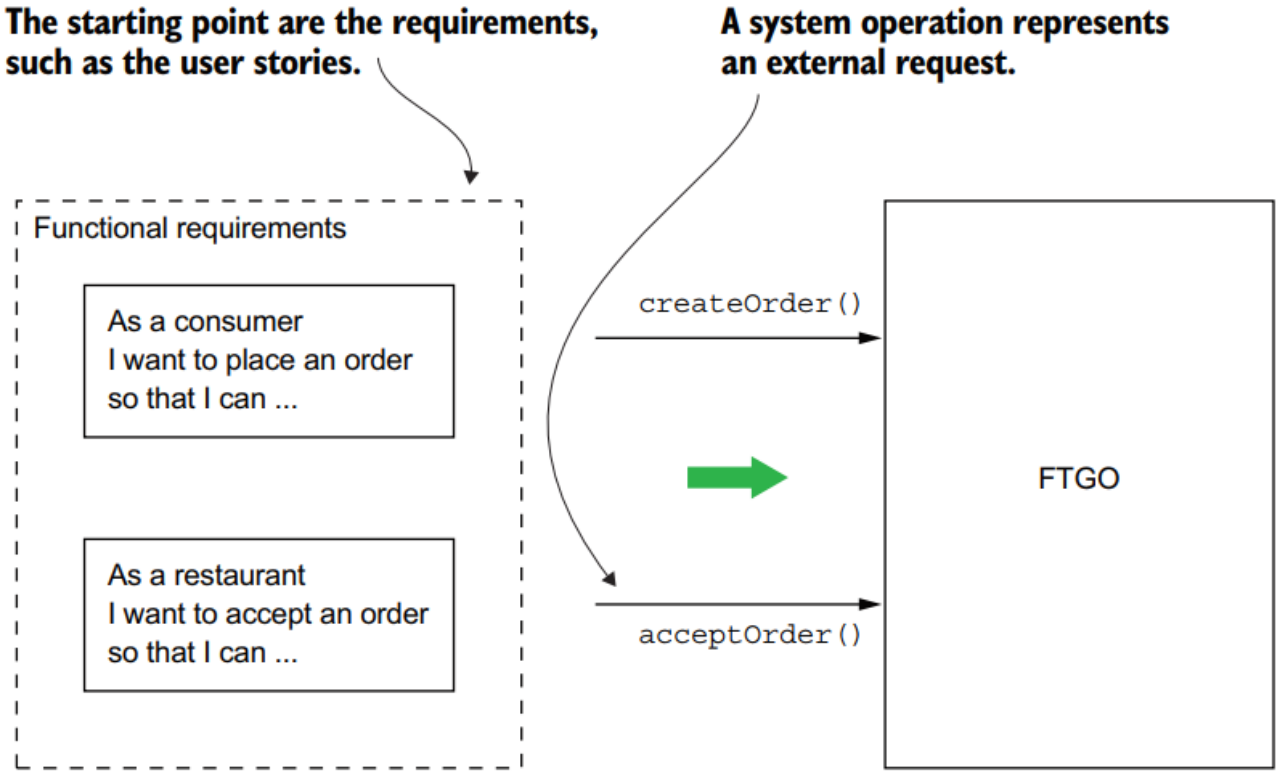
Define Application's Microservice Architecture

Architectural styles

- The size of a service is mostly unimportant

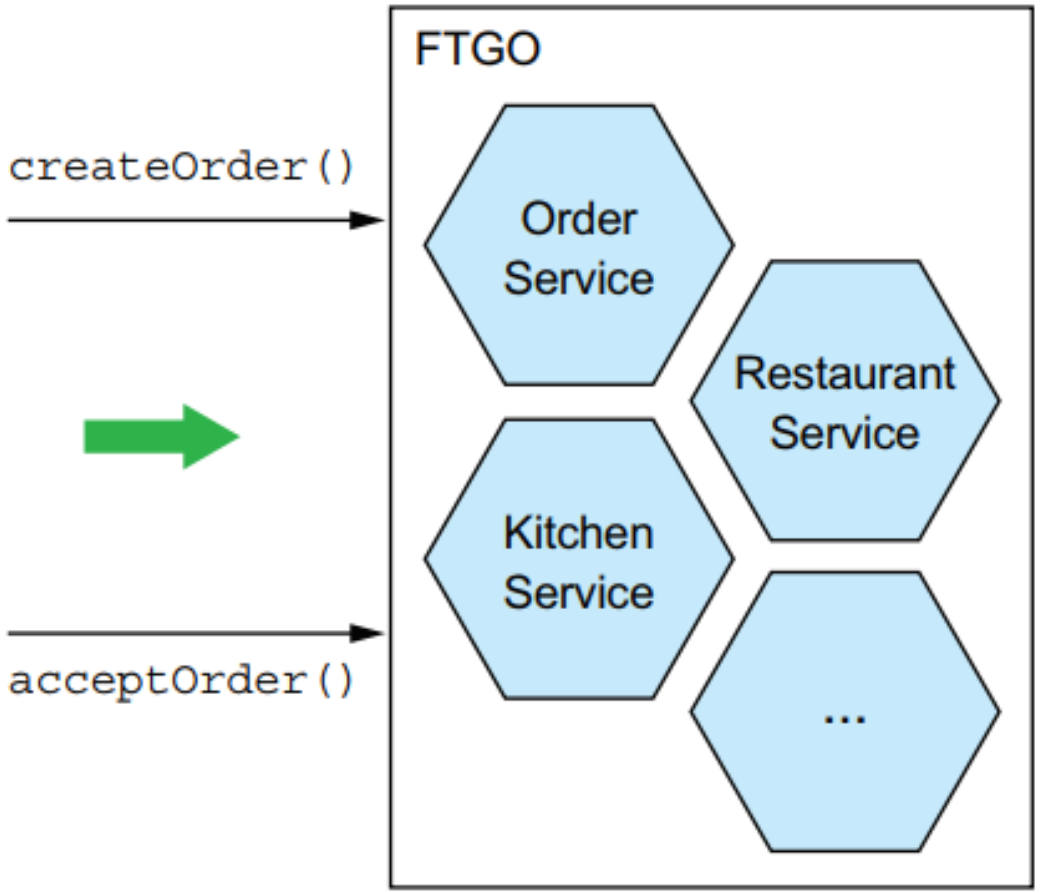
Define Application's Microservice Architecture

- Three-step process for defining an application's architecture
  - **Step 1: Identify system operations**



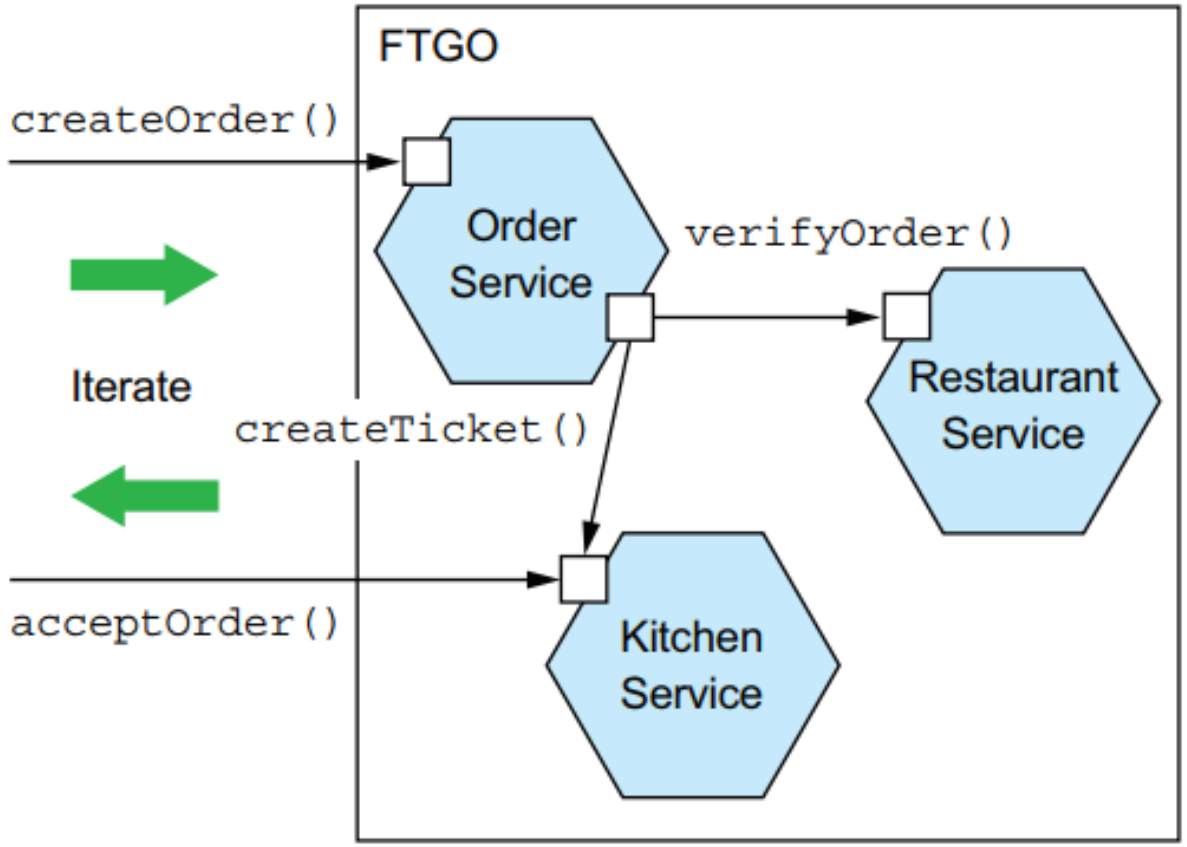
Define Application's Microservice Architecture

- Three-step process for defining an application's architecture
  - **Step 2: Identify services**



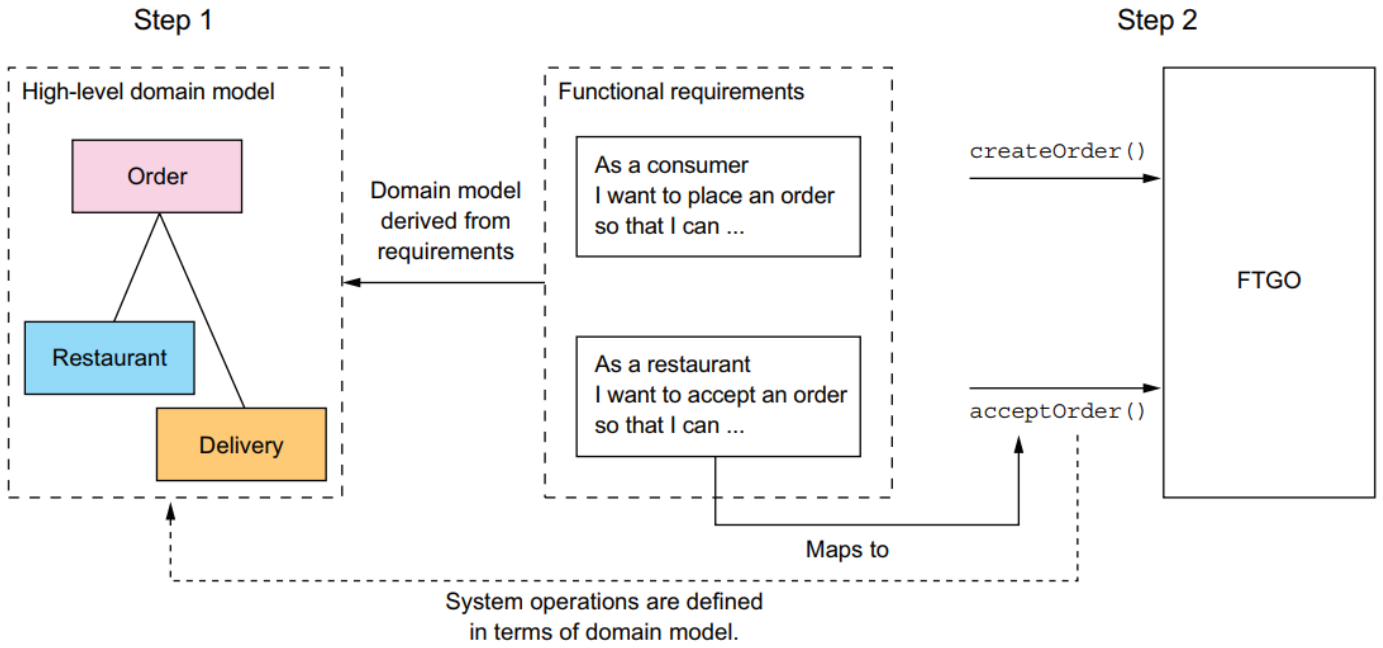
Define Application's Microservice Architecture

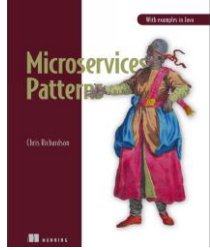
- Three-step process for defining an application's architecture
  - **Step 3: Define service APIs and collaborations**



Define Application's Microservice Architecture

- **Step 1: Identify system operations**



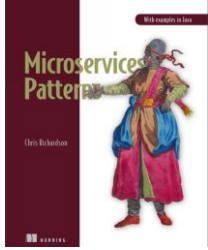


Define Application's Microservice Architecture

- **Step 1: Identify system operations**
  - **Creating a high-level dom model**
    - A domain model is created using standard techniques such as analyzing the nouns in the stories and scenarios

Given a consumer
And a restaurant
And a delivery address/time that can be served by that restaurant
And an order total that meets the restaurant's order minimum
When the consumer places an order for the restaurant
Then consumer's credit card is authorized
And an order is created in the PENDING_ACCEPTANCE state
And the order is associated with the consumer
And the order is associated with the restaurant

Consumer, Order, Restaurant, and CreditCard



Define Application's Microservice Architecture

- **Step 1: Identify system operations**
  - Creating a high-level domain model
    - A domain model is created using standard techniques such as analyzing the nouns in the stories and scenarios

Given an order that is in the PENDING_ACCEPTANCE state and a courier that is available to deliver the order

When a restaurant accepts an order with a promise to prepare by a particular time

Then the state of the order is changed to ACCEPTED

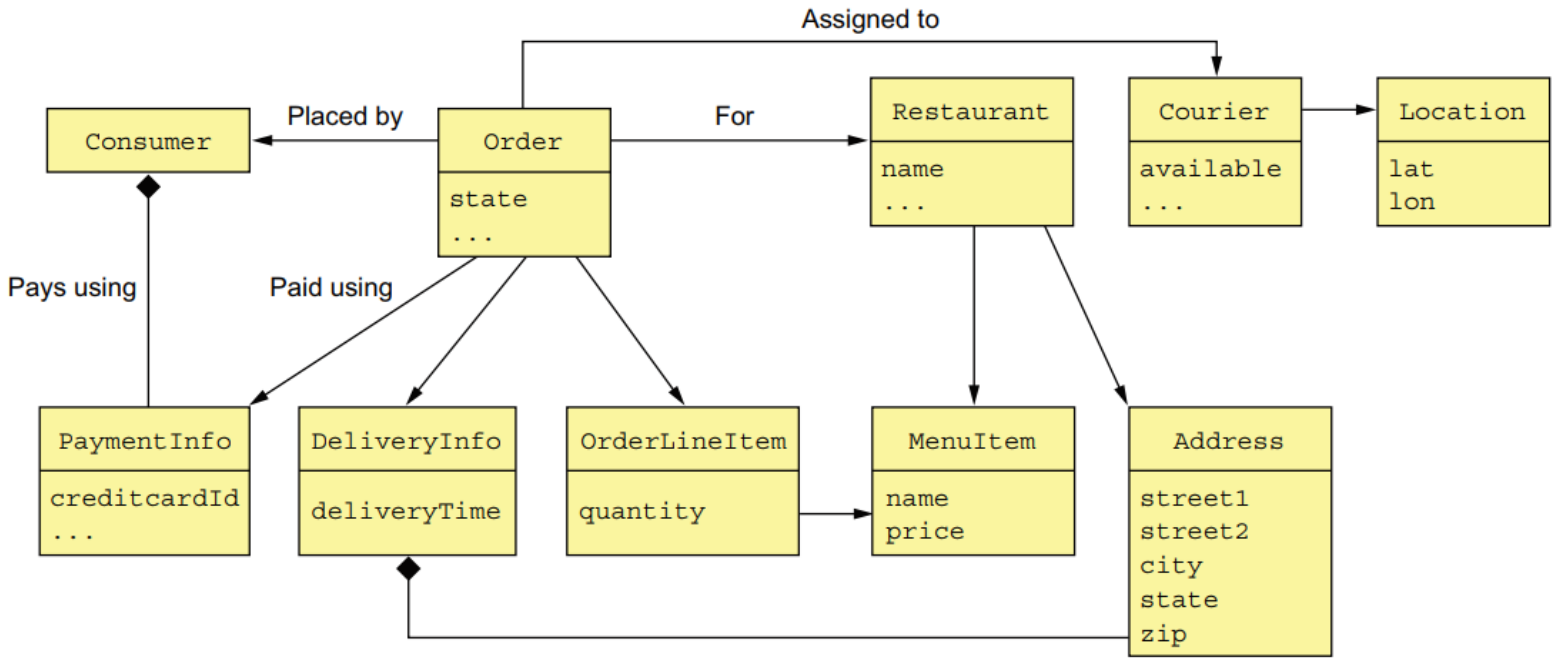
And the order's promiseByTime is updated to the promised time

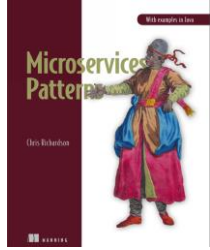
And the courier is assigned to deliver the order

???

Define Application's Microservice Architecture

- **Step 1: Identify system operations**
  - **Creating a high-level domain model**

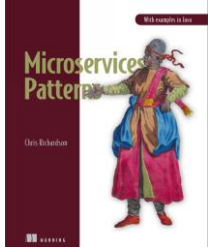




Define Application's Microservice Architecture

- **Step 1: Identify system operations**
  - **Defining system operations**
    - **Commands—System operations that create, update, and delete data**
    - **Queries—System operations that read (query) data**

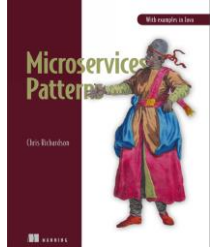
ANALYZE THE VERBS IN THE USER STORIES AND SCENARIOS



Define Application's Microservice Architecture

- **Step 1: Identify system operations**
  - **Defining system operations**

Actor	Story	Command	Description
Consumer	Create Order	`createOrder()`	Creates an order
Restaurant	Accept Order	`acceptOrder()`	Indicates that the restaurant has accepted the order and is committed to preparing it by the indicated time

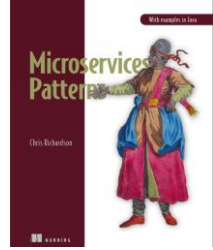


Define Application's Microservice Architecture

❑Step 1: Identify system operations

❑Defining system operations

Actor	Story	Command	Description
Restaurant	Order Ready for Pickup	`noteOrderReadyForPickup()`	Indicates that the order is ready for pickup
Courier	Update Location	`noteUpdatedLocation()`	Updates the current location of the courier
Courier	Delivery picked up	`noteDeliveryPickedUp()`	Indicates that the courier has picked up the order
Courier	Delivery delivered	`noteDeliveryDelivered()`	Indicates that the courier has delivered the order

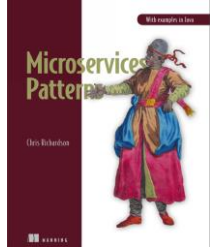


Define Application's Microservice Architecture

❑Step 1: Identify system operations

❑Defining system operations

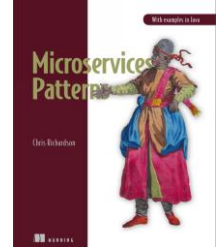
Operation	`createOrder` (consumer id, payment method, delivery address, delivery time, restaurant id, order line items)
Returns	`orderId`, ...
Preconditions	▪ The consumer exists and can place orders. ▪ The line items correspond to the restaurant's menu items. ▪ The delivery address and time can be serviced by the restaurant.
Post-conditions	▪ The consumer's credit card was authorized for the order total. ▪ An order was created in the `PENDING_ACCEPTANCE` state.



Define Application's Microservice Architecture

- **Step 1: Identify system operations**
  - **Defining system operations**

Operation	`acceptOrder(restaurantId, orderId, readyByTime)`
Returns	—
Preconditions	▪ The `order.status` is `PENDING_ACCEPTANCE`. ▪ A courier is available to deliver the order.
Post-conditions	▪ The `order.status` was changed to `ACCEPTED`. ▪ The `order.readyByTime` was changed to the `readyByTime`. ▪ The courier was assigned to deliver the order.

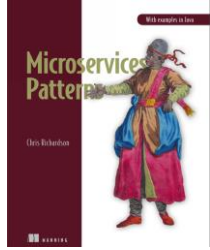


Define Application's Microservice Architecture

❑Step 1: Identify system operations

❑Defining system operations

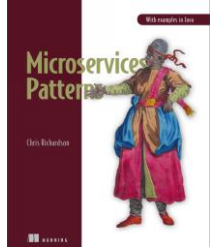
1. User enters delivery address and time.
2. System displays available restaurants.
3. User selects restaurant.
4. System displays menu.
5. User selects item and checks out.
6. System creates order.



Define Application's Microservice Architecture

- **Step 2: Identify services**
  - **Defining services by applying the Decompose by business capability pattern**

Business capability	→	A business does in order to generate value
Insurance company	→	Underwriting, Claims management, Billing, Compliance
Online store	→	Order management, Inventory management, Shipping



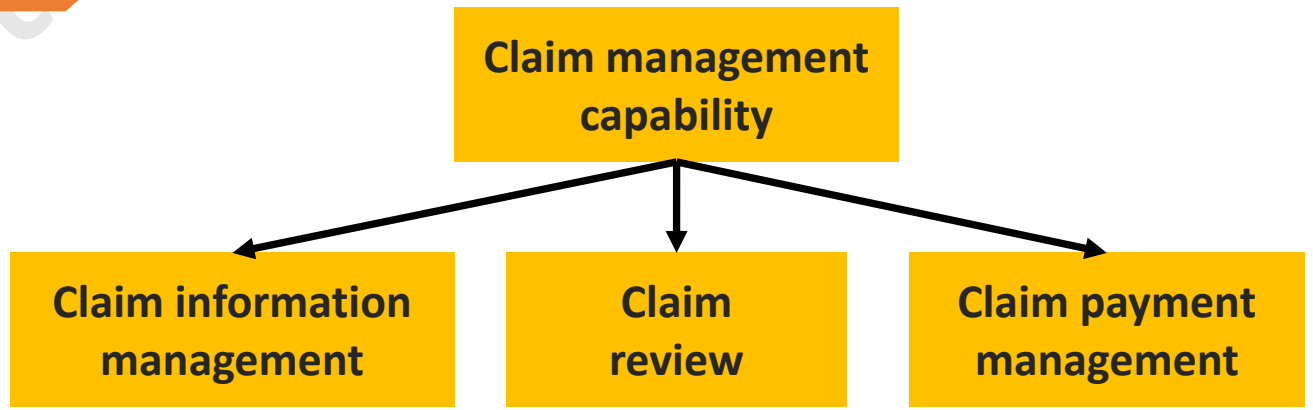
Define Application's Microservice Architecture

❑Step 2: Identify services

❑Defining services by applying the Decompose by business capability pattern

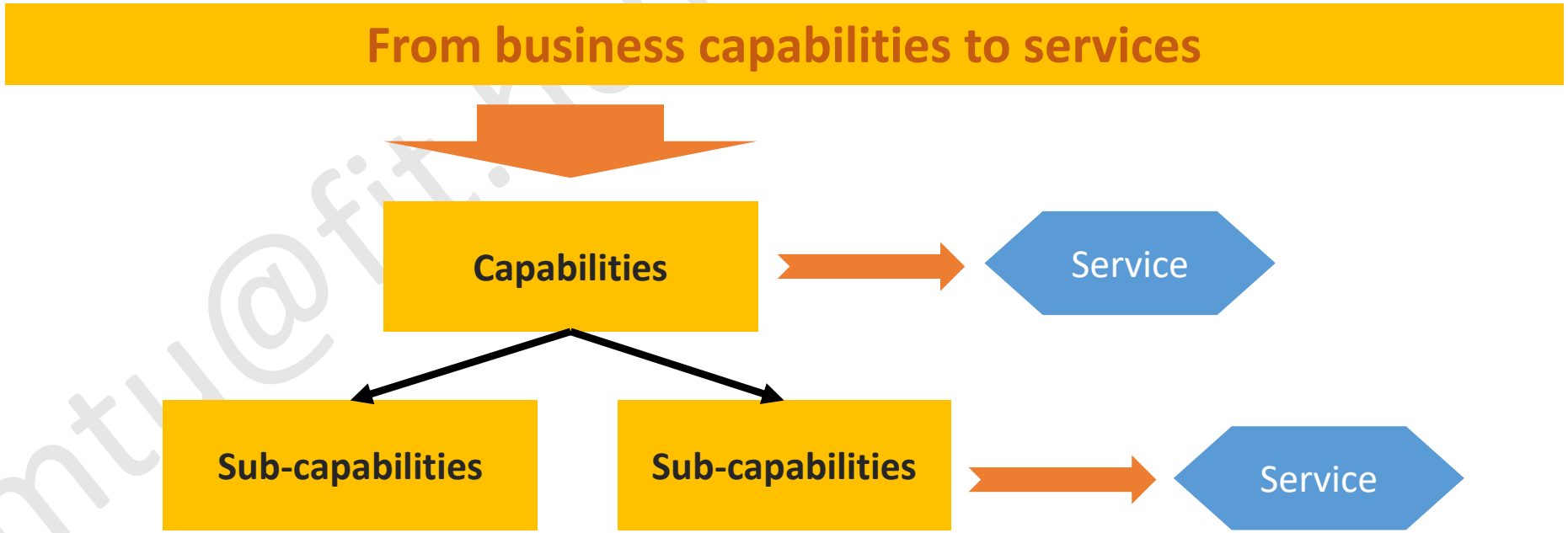
Business capability → Service

1. Business-oriented rather than technical
2. A capability can often be decomposed into sub-capabilities



Define Application's Microservice Architecture

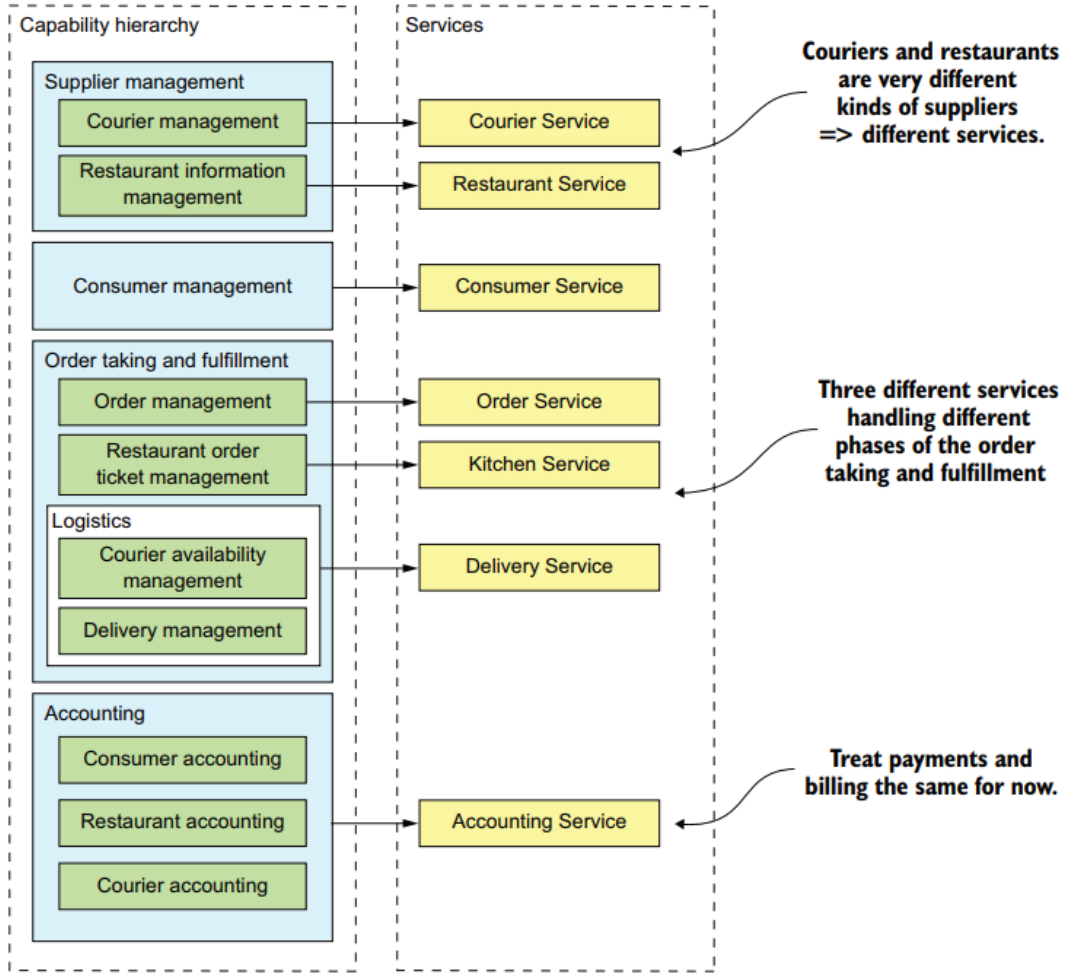
- Step 2: Identify services
  - Defining services by applying the Decompose by business capability pattern

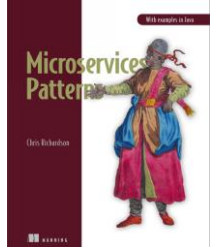


Define Application's Microservice Architecture

- **Step 2: Identify services**
  - **Defining services by applying the Decompose by business capability pattern**

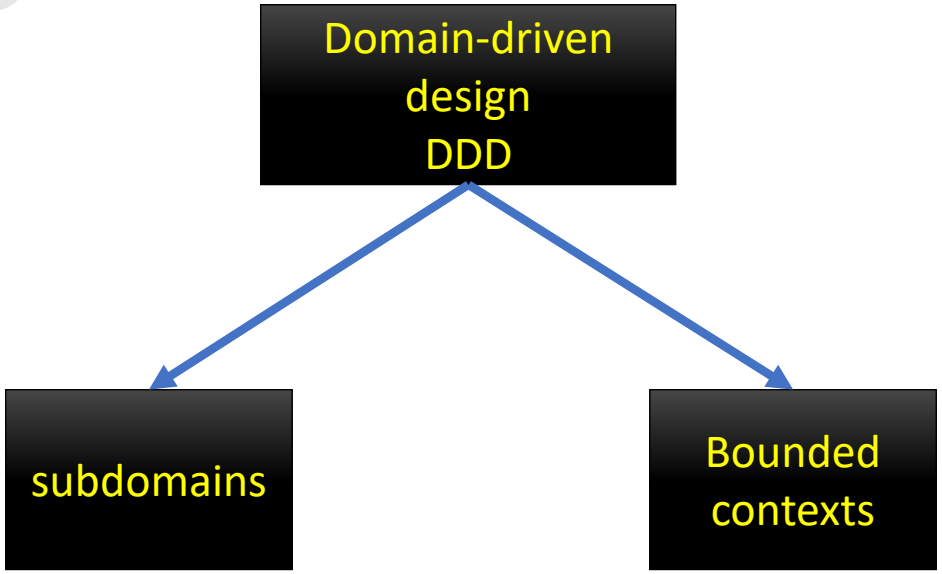
Example





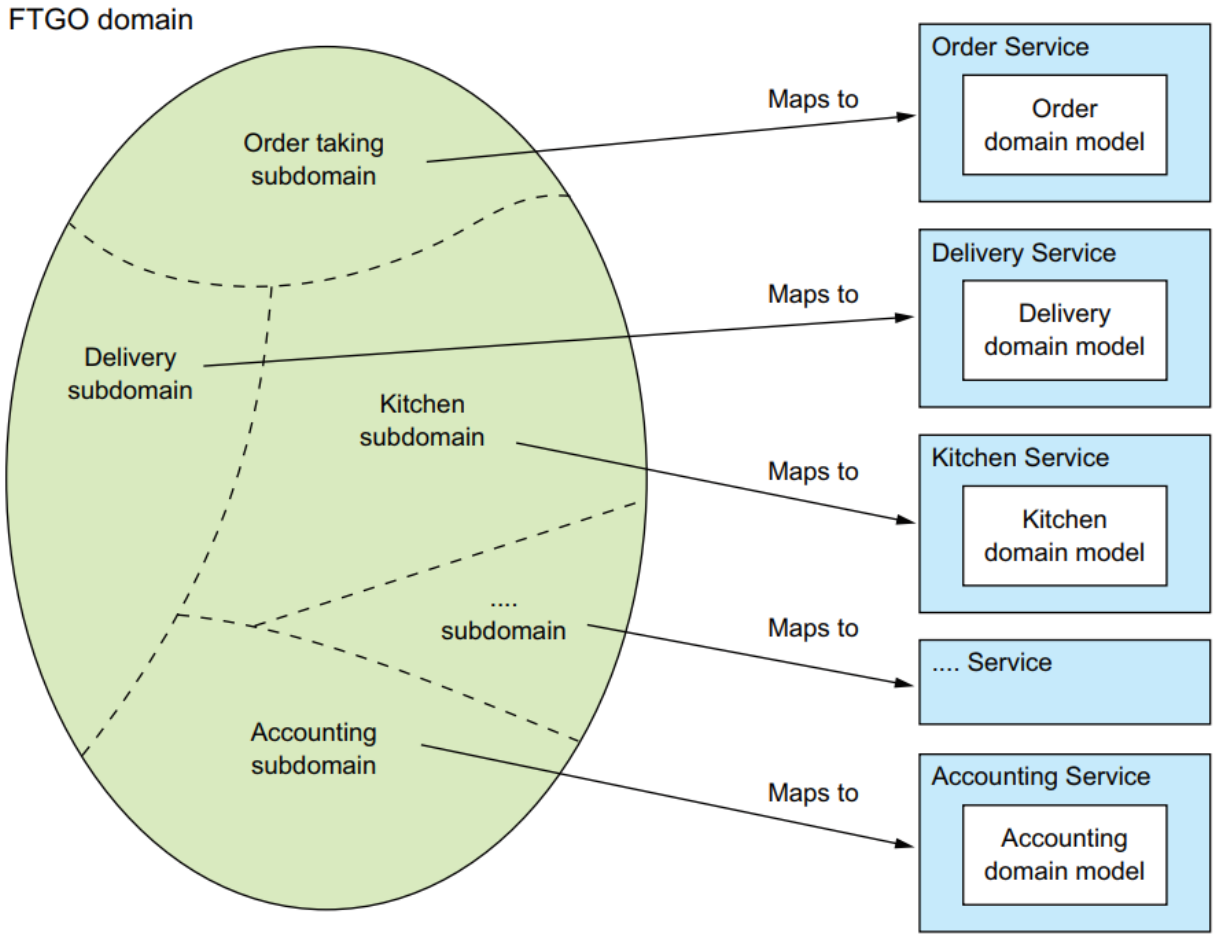
Define Application's Microservice Architecture

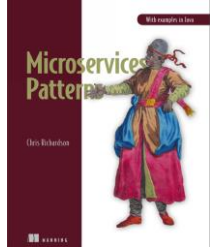
- **Step 2: Identify services**
  - **Defining services by applying the Decompose by sub-domain pattern**
    - DDD defines a separate domain model for each subdomain
    - Identified Subdomain: Analyze the business and identify the different areas of expertise
    - A bounded context includes the code artifacts that implement the model that is a service or possibly a set of services



Define Application's Microservice Architecture

- **Step 2: Identify services**
  - **Defining services by applying the Decompose by sub-domain pattern**

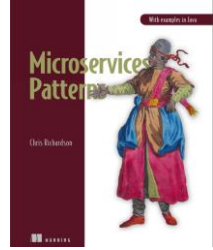




Define Application's Microservice Architecture

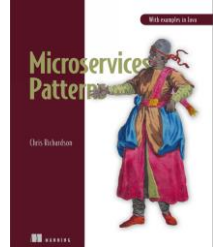
- **Step 2: Identify services**
  - **Decomposition guidelines**

SINGLE RESPONSIBILITY PRINCIPLE	COMMON CLOSURE PRINCIPLE



Define Application's Microservice Architecture

- Obstacles to decomposing an application into services
  - Network latency
  - Synchronous interprocess communication reduces availability
  - Maintaining data consistency across services
  - Obtaining a consistent view of the data
  - God classes prevent decomposition

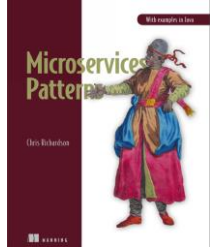


Define Application's Microservice Architecture

- **Step 3: Define service APIs and collaborations**
  - **Assigning system operations to services**

Table 2.2 Mapping system operations to services in the FTGO application

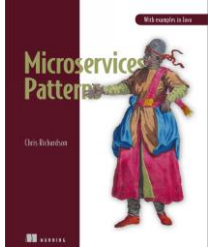
Service	Operations
Consumer Service	`createConsumer()`
Order Service	`createOrder()`
Restaurant Service	`findAvailableRestaurants()`
Kitchen Service	▪ `acceptOrder()` ▪ `noteOrderReadyForPickup()`
Delivery Service	▪ `noteUpdatedLocation()` ▪ `noteDeliveryPickedUp()` ▪ `noteDeliveryDelivered()`



Define Application's Microservice Architecture

- **Step 3: Define service APIs and collaborations**
  - **Determining the APIs required to support collaboration between services**

Service	Operations	Collaborators
Consumer Service	verifyConsumerDetails()	—
Order Service	createOrder()	▪ Consumer Service verifyConsumerDetails() ▪ Restaurant Service verifyOrderDetails() ▪ Kitchen Service createTicket() ▪ Accounting Service authorizeCard()
Restaurant Service	▪ findAvailableRestaurants() ▪ verifyOrderDetails()	—
Kitchen Service	▪ createTicket() ▪ acceptOrder() ▪ noteOrderReadyForPickup()	▪ Delivery Service scheduleDelivery()
Delivery Service	▪ scheduleDelivery() ▪ noteUpdatedLocation() ▪ noteDeliveryPickedUp() ▪ noteDeliveryDelivered()	—
Accounting Service	▪ authorizeCard()	—



Summary

- Architecture determines application's -ilities, including maintainability, testability, and deployability, which directly impact development velocity.
- The microservice architecture is an architecture style that gives an application high maintainability, testability, and deployability.
- Services in a microservice architecture are organized around business concerns-business capabilities or subdomains—rather than technical concerns.
- There are two patterns for decomposition:
  - Decompose by business capability, which has its origins in business architecture
  - Decompose by subdomain, based on concepts from domain-driven design
- Can eliminate god classes, which cause tangled dependencies that prevent decomposition, by applying DDD and defining a separate domain model for each service.