# Agenda

## Decomposition & Integration Patterns

- Decomposition Patterns
  - Decompose by Business Capability
  - Decompose by Subdomain
  - Decompose by Strangler
- Integration Patterns
  - API Gateway Pattern
  - Aggregator Pattern
  - Client-Side UI Composition Pattern

ScholarHat

## Decomposition Patterns

**Problem:** Microservices make loosely coupled services that apply the single responsibility principle. However, breaking an application into smaller pieces has to be done logically. How do we decompose an application into small services?

**Solution:** There are the following patterns to rescue the above problem:

- Decompose by Business Capability
- Decompose by Subdomain
- Decompose by Strangler - Legacy App

## 1. Decompose by Business Capability

- Microservices should be decomposed by business capability. A business capability often corresponds to a business object.
- Business capabilities of an online store include:
  - Authentication Management
  - Catalog Management
  - Order Management etc.

## 2. Decompose by Subdomain

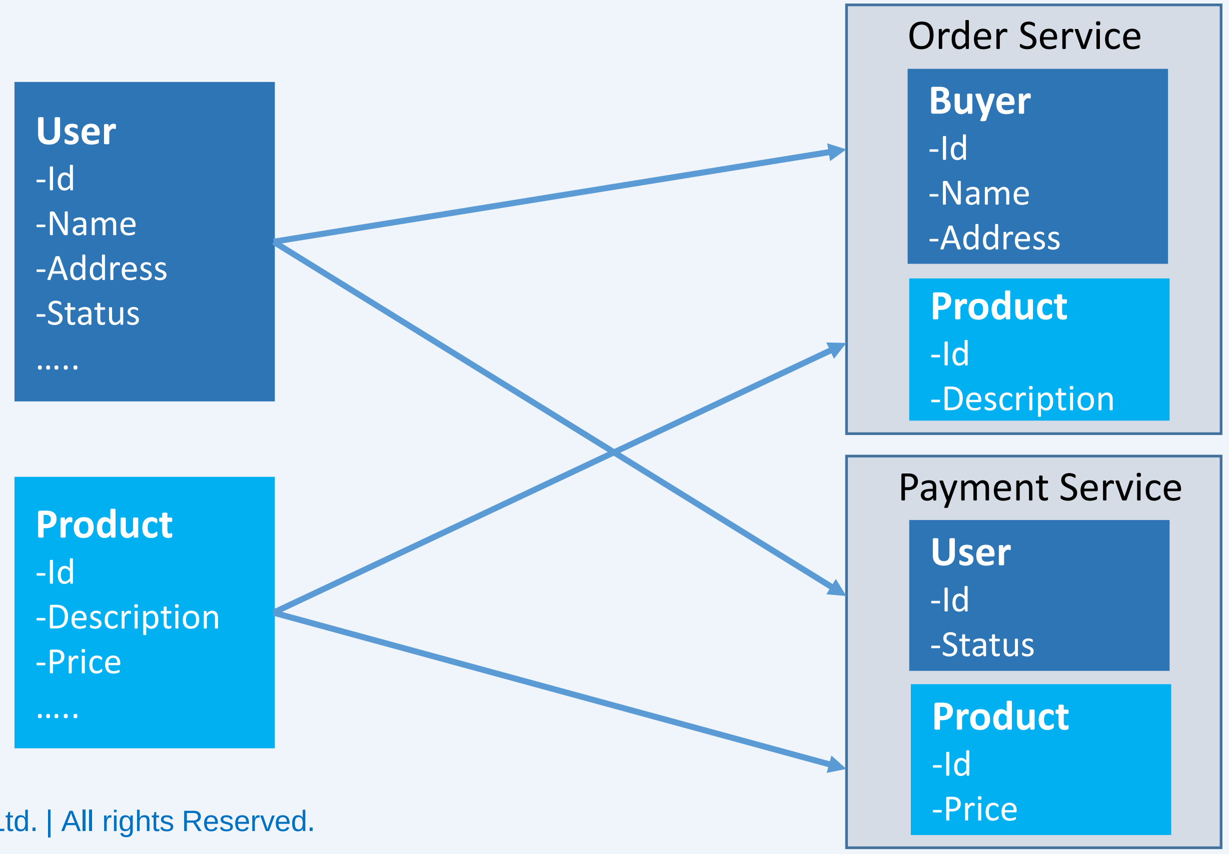
- Decomposing an application using Domain-Driven Design (DDD) which refers to the application’s problem space as the domains and subdomains.
- A domain consists of multiple subdomains. Each subdomain corresponds to a different part of the business. Subdomains can be classified using the following criteria:
  - **Core**: Most important and key differentiator of an application.
  - **Supporting**: Business-related and are used to support the business activities.
  - **Generic**: Not specific to business but is used to enhance the business operations.

## 2. Decompose by Subdomain Contd..

- For example, the subdomains of an E-Commerce Domain application include:
  - Catalog Management (Catalog Service)
  - Inventory Management (Inventory Service)
  - Order Management (Order Service)
  - Delivery Management (Delivery Service)

## 2.1 Decompose Traditional Data Model

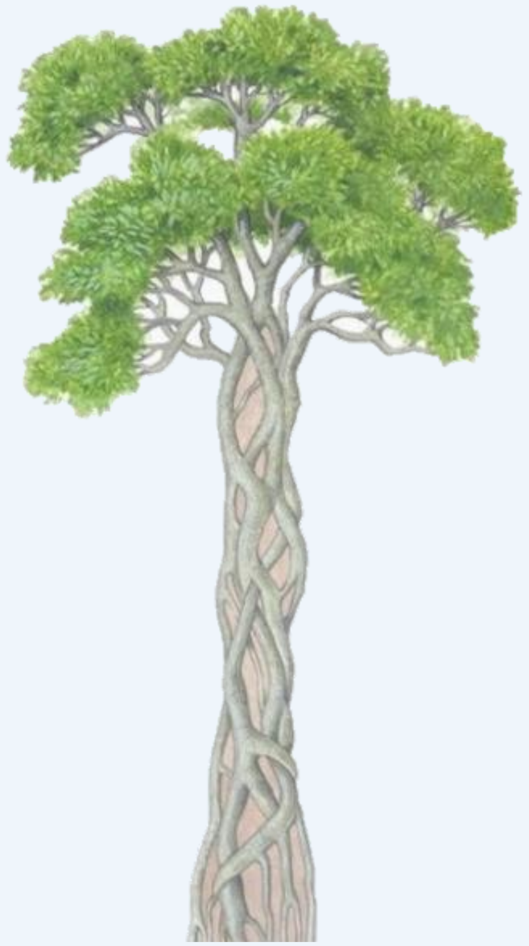
- Decompose traditional model into multiple domain models i.e. One domain model per microservice or bounded context.



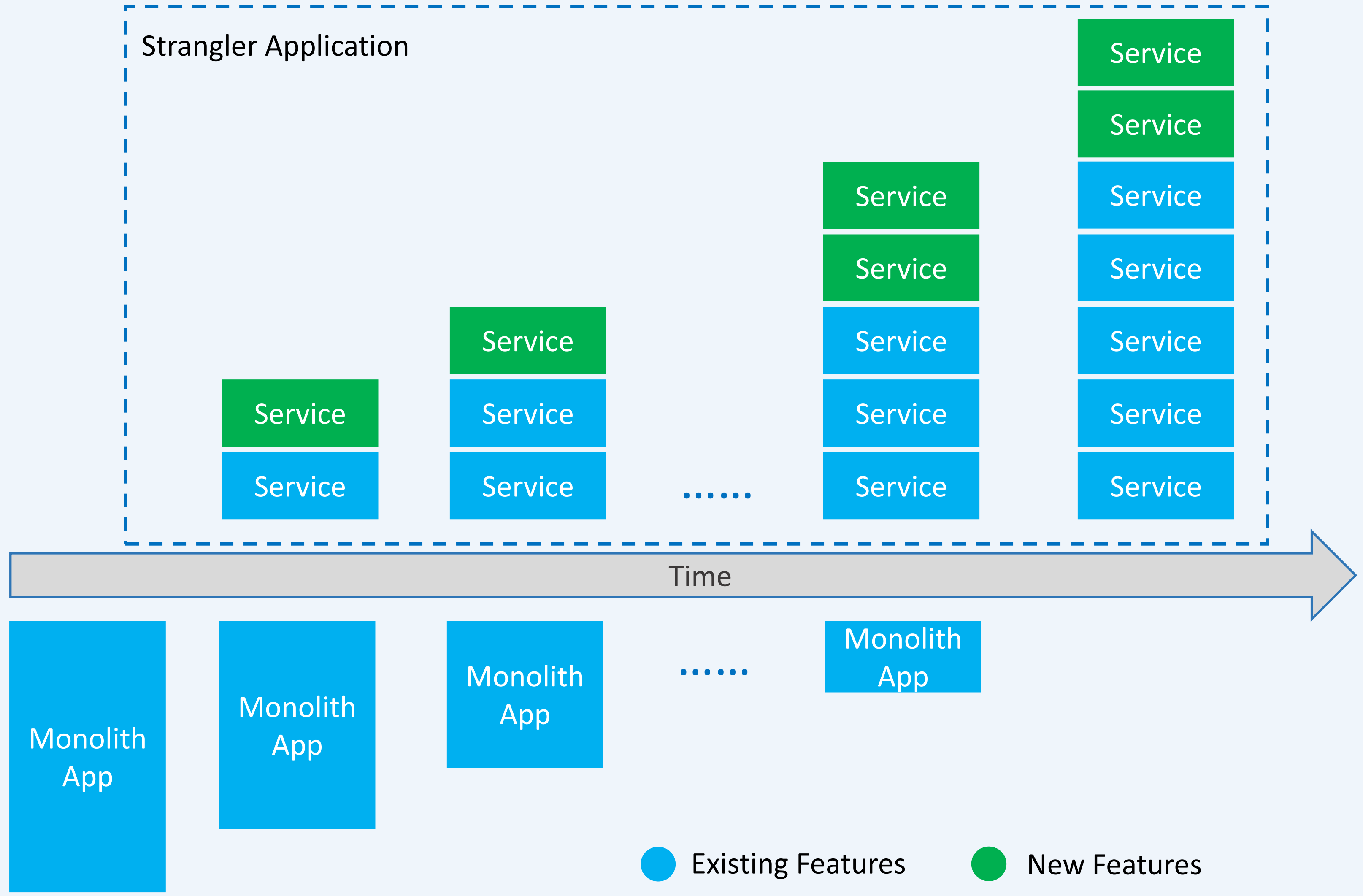
ScholarHat

## 3. Strangler Pattern

- This pattern is used to decompose an existing, monolithic application. Since applying all the above design patterns to them will be difficult because breaking them into smaller pieces at the same time it's being used live is a big task.
- The Strangler pattern creates two separate applications that live side by side in the same URI space. Over time, the newly refactored application “strangles” or replaces the existing monolithic application.



## 3.1 Strangler Pattern



● Existing Features ● New Features

## Integration Patterns

**Problem:** When an application is broken down into smaller microservices, there are a few concerns that need to be addressed:

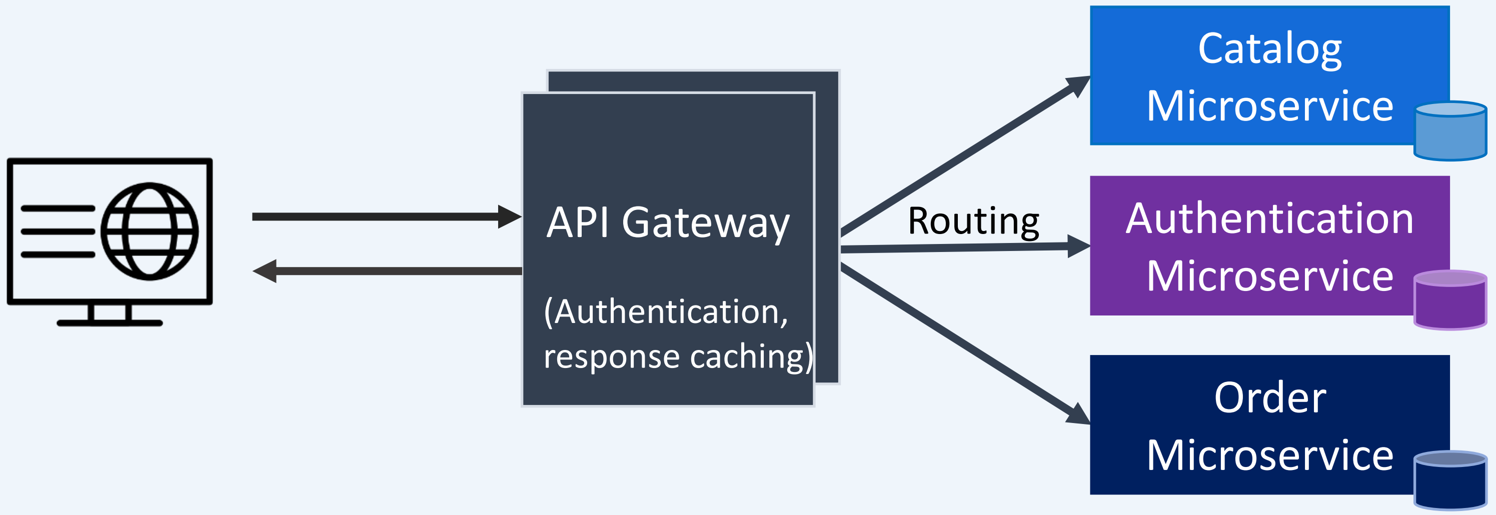
- There are multiple calls for multiple microservices by different channels
- There is a need for handling different types of Protocols
- Different consumers might need a different format of the responses

**Solution:** There are the following patterns to rescue the above concerns:

- API Gateway Pattern
- Aggregator Pattern
- Client-Side UI Composition Pattern

## 1. API Gateway Pattern

- An API Gateway is a single point of entry for any microservice calls.
- Work as a proxy service to route a request to the backend microservices.
- Aggregate the results to send back to the consumer.
- Offload the authentication/authorization responsibility of the microservice.


API Gateway
(Authentication, response caching)
Routing
Catalog Microservice
Authentication Microservice
Order Microservice

# Microservices API Gateways

Ocelot API Gateway

NGINX Ingress Controller

Azure API Management

AWS API Management

Free: Open Source

Cloud Managed

## 2. Aggregator Pattern

- Aggregator patterns help your how to collaborate the data returned by each service.
- Aggregation can be done in two ways:
  - A composite microservice will make calls to all the required microservices, consolidate the data, and transform the data before sending it back.
  - An **API Gateway** can also partition the request to multiple microservices and aggregate the data before sending it to the consumer.

## 3. Client-Side UI Composition Pattern

- With microservices, the UI has to be designed as a skeleton with multiple sections/regions of the screen/page.
- Each section will make a call to an individual backend microservice to pull the data. That is called composing UI components specific to the service.
- Frameworks like Angular and React help to do that easily. These screens are known as Single Page Applications (SPA).
- This enables the app to refresh a particular region of the screen instead of the whole page.