Microservices

Introduction

Topics Covered

1. Motivation for Microservices Architecture

2. Motivation for Event-Driven Architecture

3. Problems with Monolithic Architecture

Topics Covered

1. Motivation for Microservices Architecture

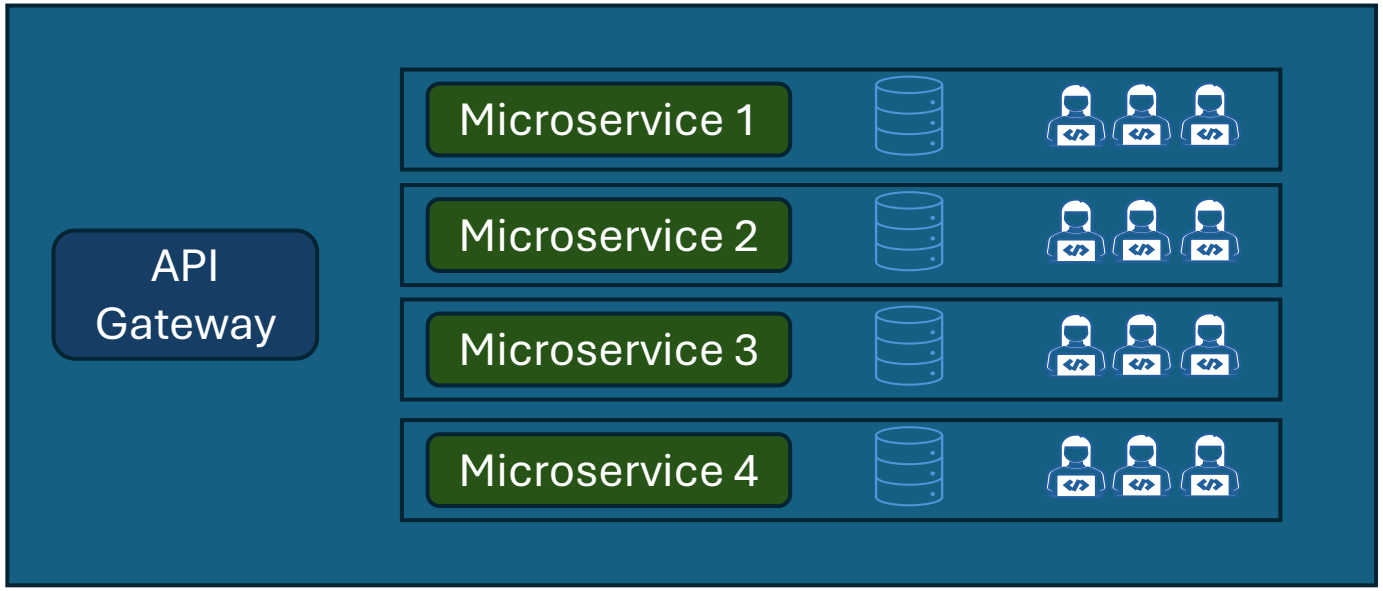
2. Motivation for Event-Driven Architecture

3. Problems with Monolithic Architecture

Motivation for Microservices Architecture

- Microservices Architecture is the most:
  - Modern
  - Popular

  Architectural style in the industry



Motivation for Microservices Architecture

- Microservices are the main topic in tech conferences
- Leading tech companies attribute a significant part of their success to this architecture.

- When done **correctly**
  - Allows organizations to scale
  - Reach billions of users
  - Keep operational costs low
  - Stay efficient and innovative

Challenges of Microservices

Microservices are exciting. BUT...

- Many organizations struggle/ rethinking their decision

Microservices is NOT a silver bullet

When applied correctly:

- It can be very beneficial

When applied incorrectly:

- It can introduce unnecessary overhead

It's no issue!

- Intuition and Skills to benefit from Microservice
- Knowledge on how to avoid costly
  - Mistakes
  - Pitfalls
  - Anti-patterns
- Side Benefit:
  - Preparation for System Design Interview @ Sherwin

Topics Covered

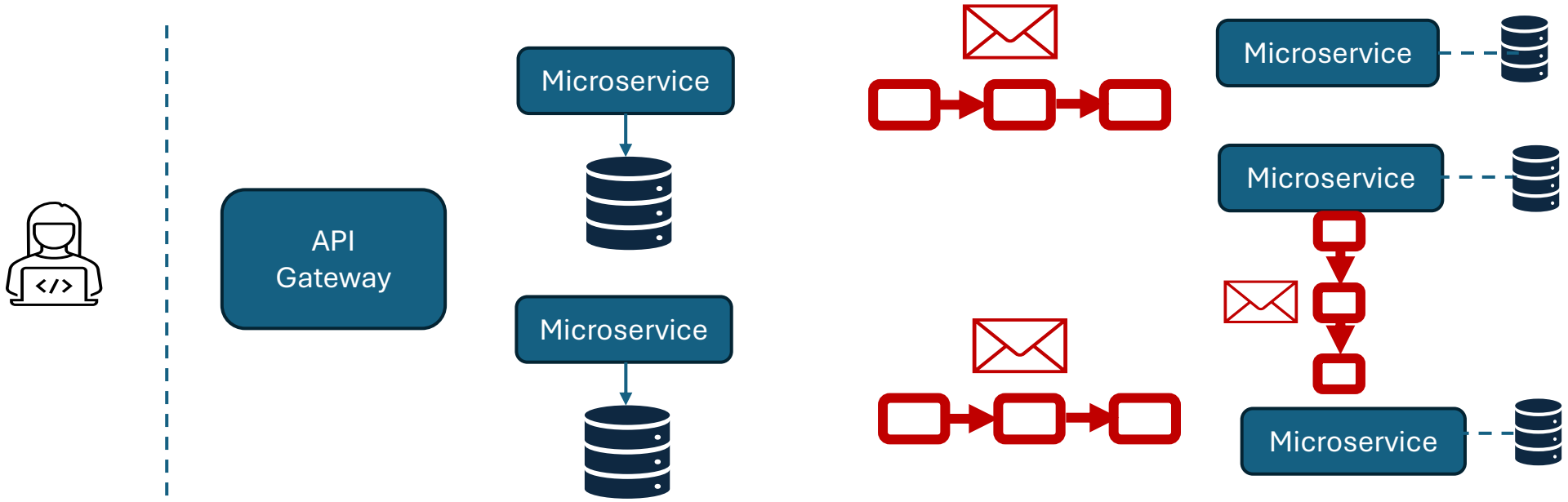
1. Motivation for Microservices Architecture

2. Motivation for Event-Driven Architecture

3. Problems with Monolithic Architecture

Event-Driven Architecture

- Commonly used with Microservices Architecture.



- Allows implementing powerful patterns for microservices

Topics Covered

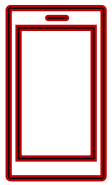
1. Motivation for Microservices Architecture

2. Motivation for Event-Driven Architecture

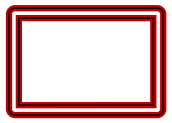
3. Problems with Monolithic Architecture

Typical Web-Based Application Architecture

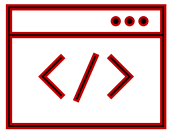
Monolithic Architecture



Mobile App

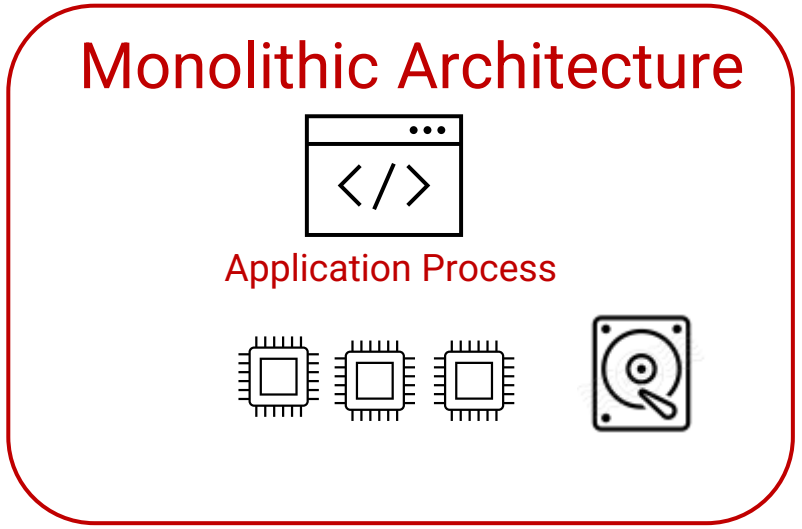


Tablet App



Webpage

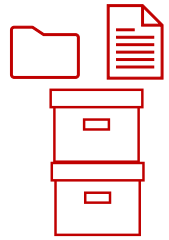
Presentation Tier



Server

Logic/ Business/Application Tier

Database

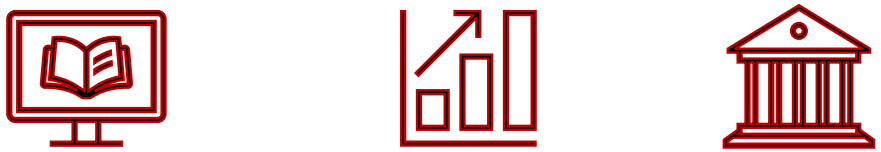


File System

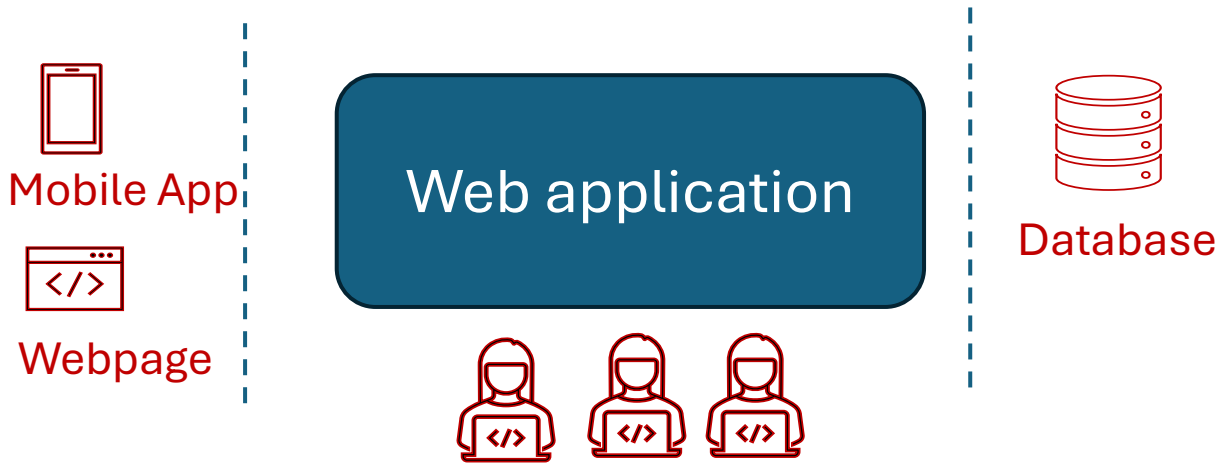
Data Tier

Monolithic Architecture- Benefits

- Easy to design

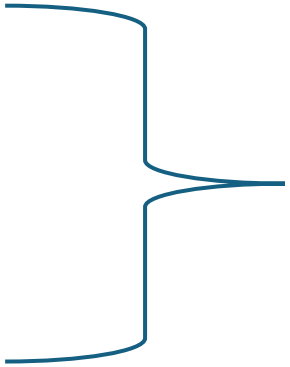


- Easy to implement



Monolithic Architecture- Benefits

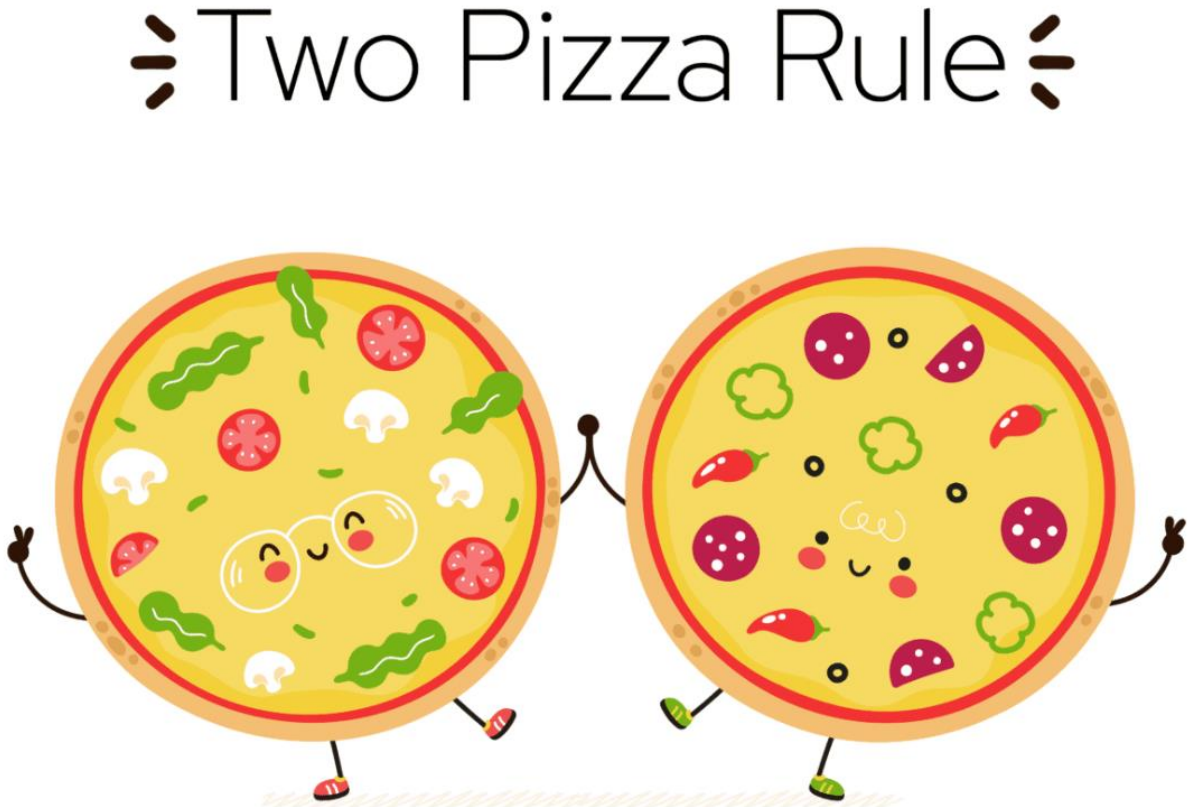
- Easy to design
- Easy to implement



Perfect for Startup companies

Perfect for small teams

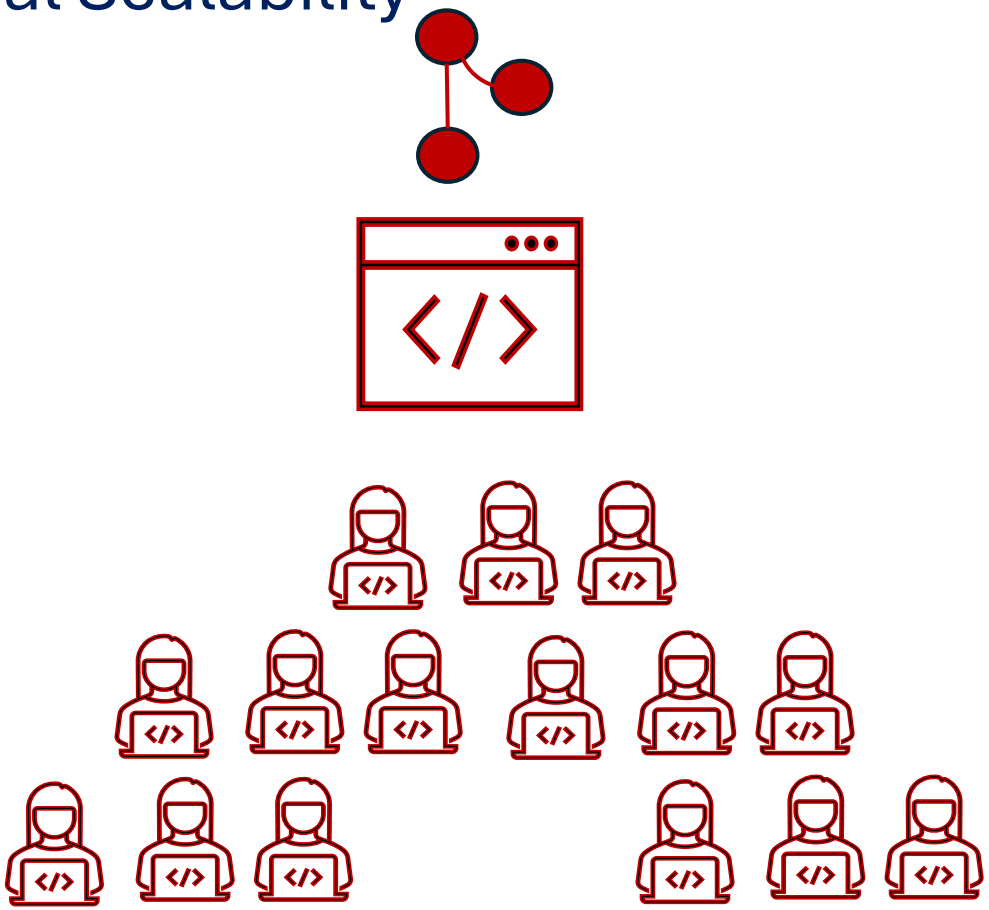
The 2 Pizza Rule



https://calcey.com/blog/why-the-two-pizza-team-rule-is-the-name-game-at-calcey/

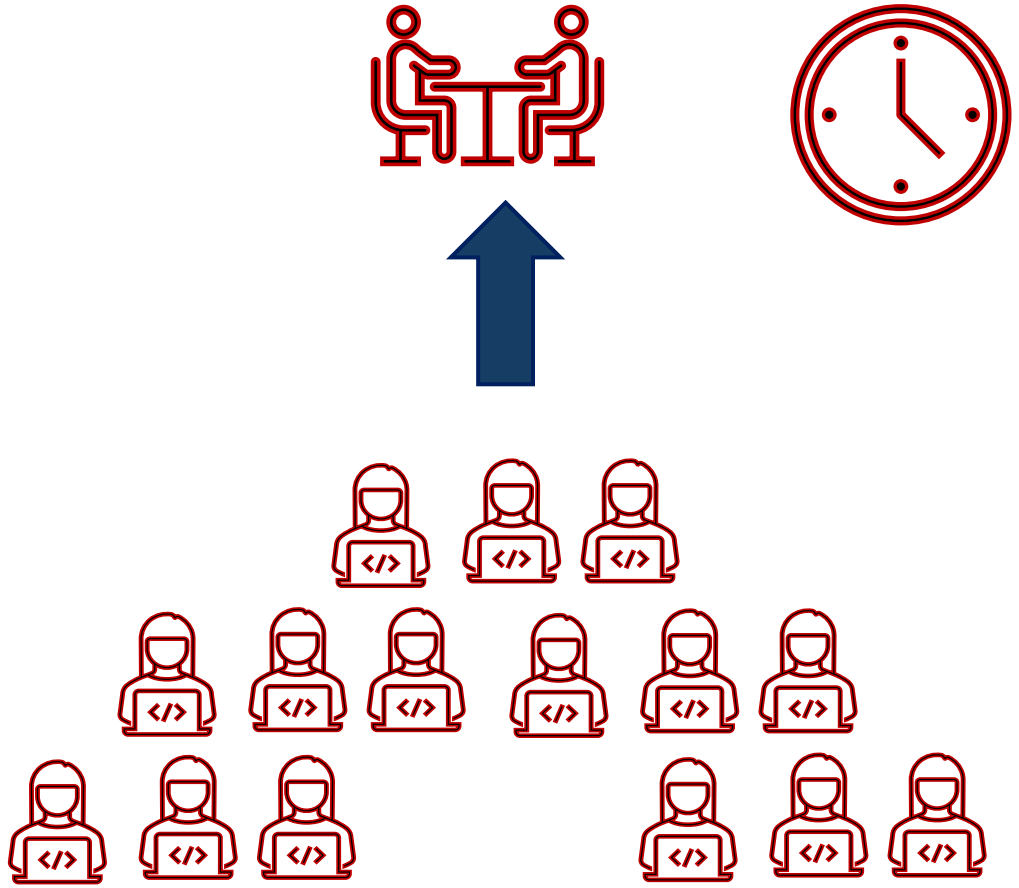
Issues with Monolithic Architecture

- Low Organizational Scalability



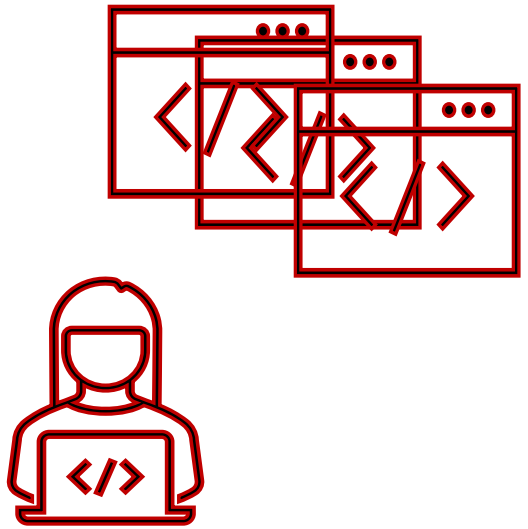
Issues with Monolithic Architecture

- Low Organizational Scalability



Issues with Monolithic Architecture

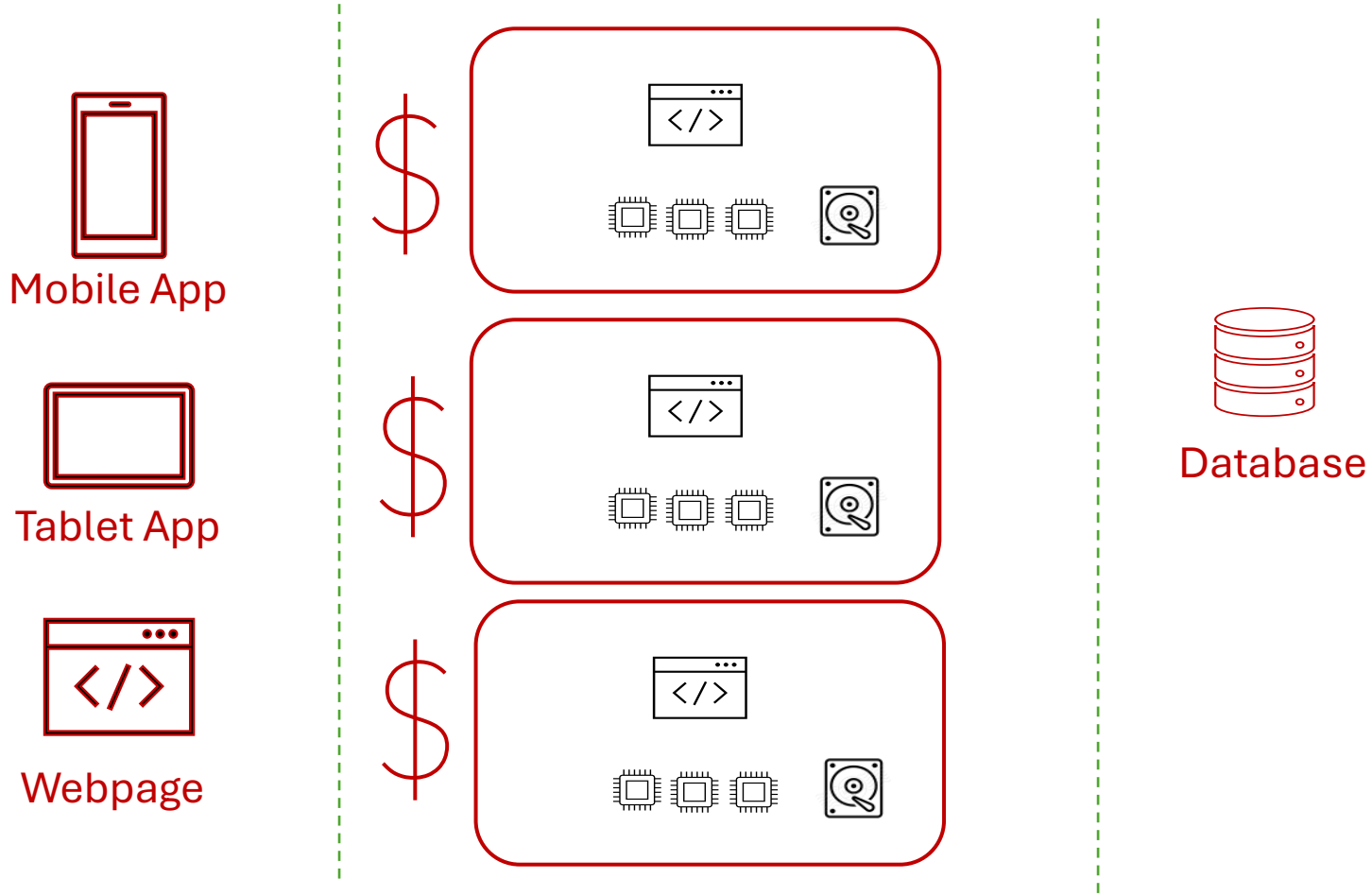
- Low Organizational Scalability
  - Complex codebase
    - Hard to reason about
    - Takes longer to load in IDE
    - Slower to build/ test
    - Risky to deploy
    - Larger and less frequent releases



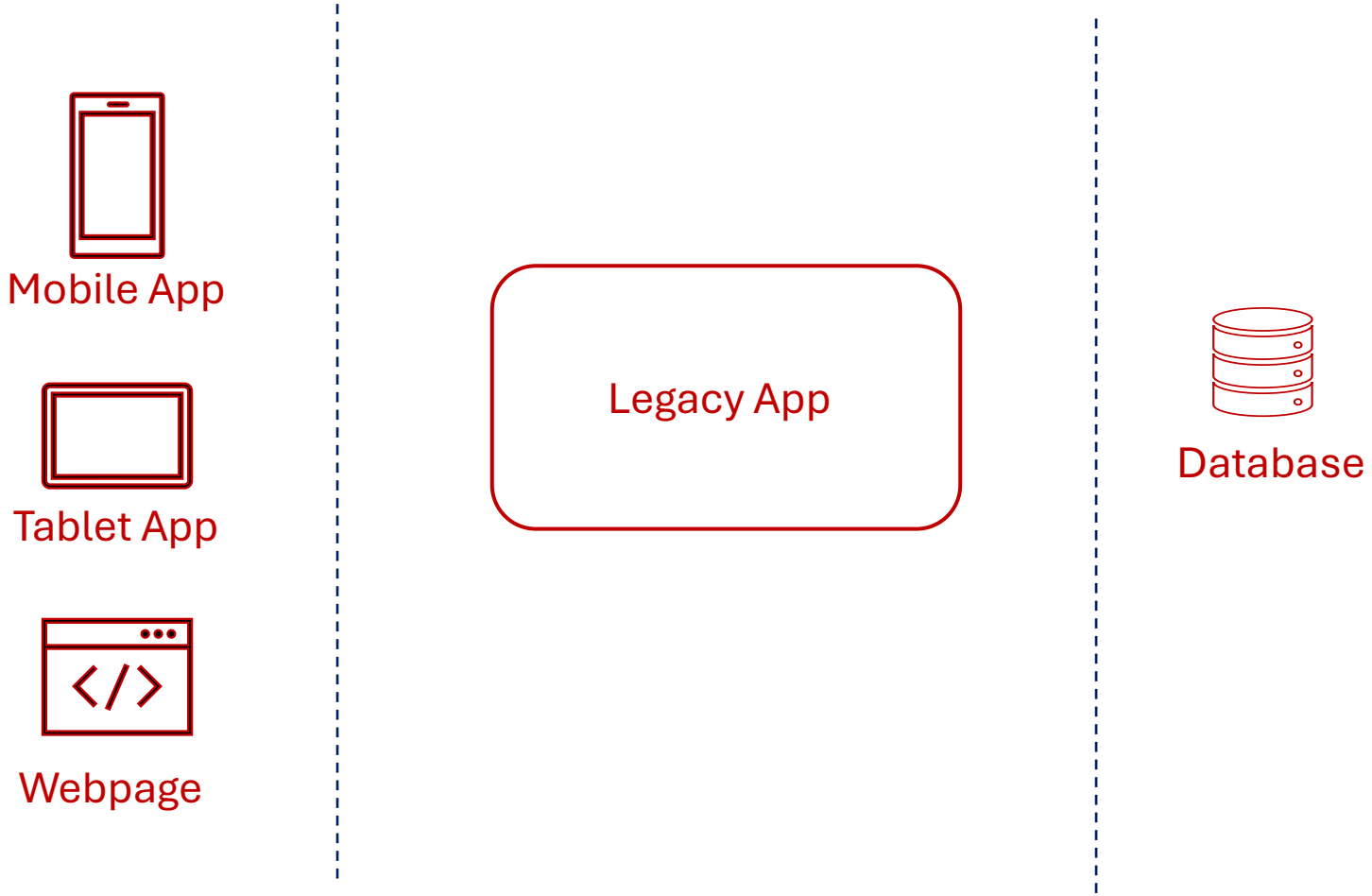
Issues with Monolithic Architecture

1. Low organizational scalability
2. Low system scalability

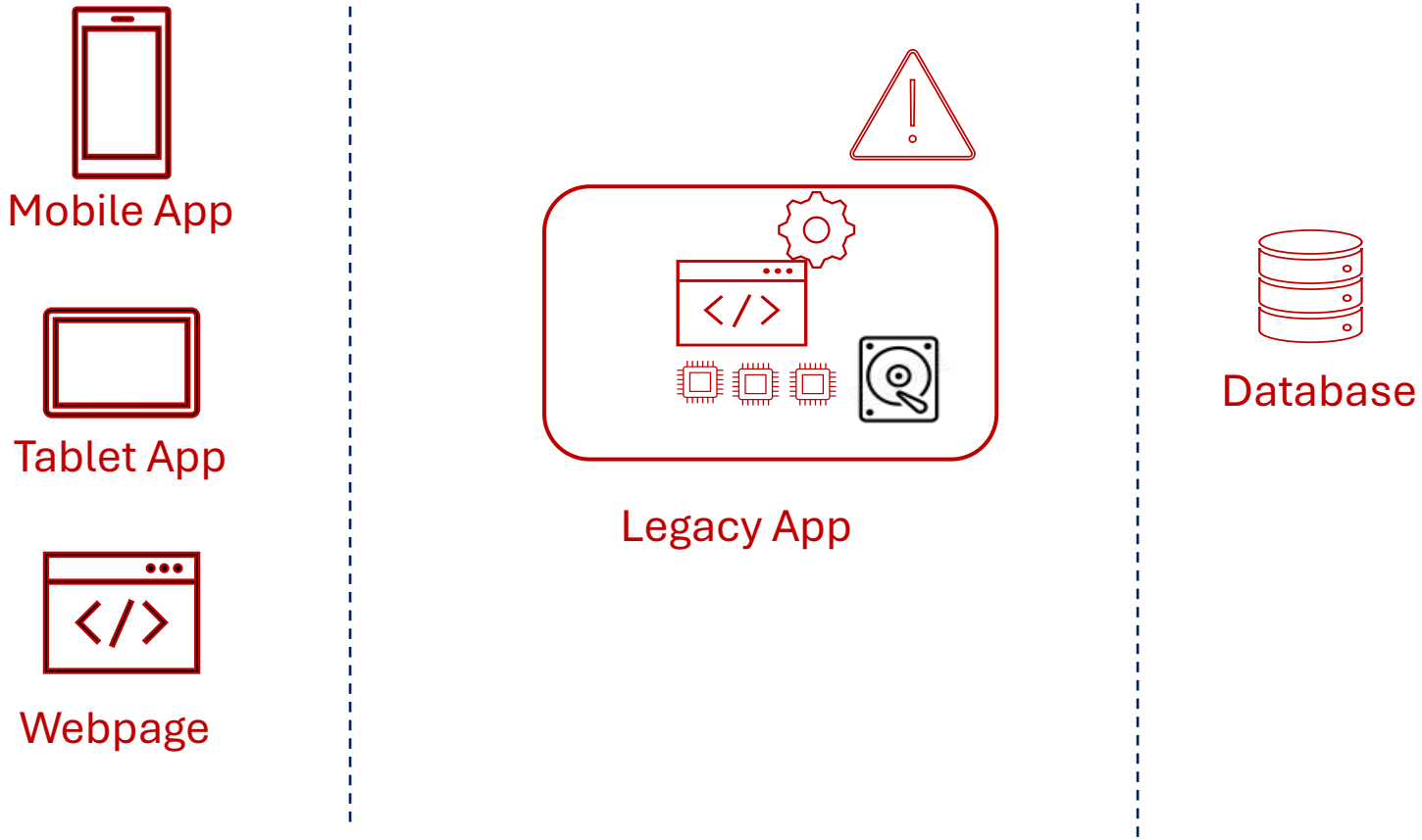
Issues with Monolithic Architecture



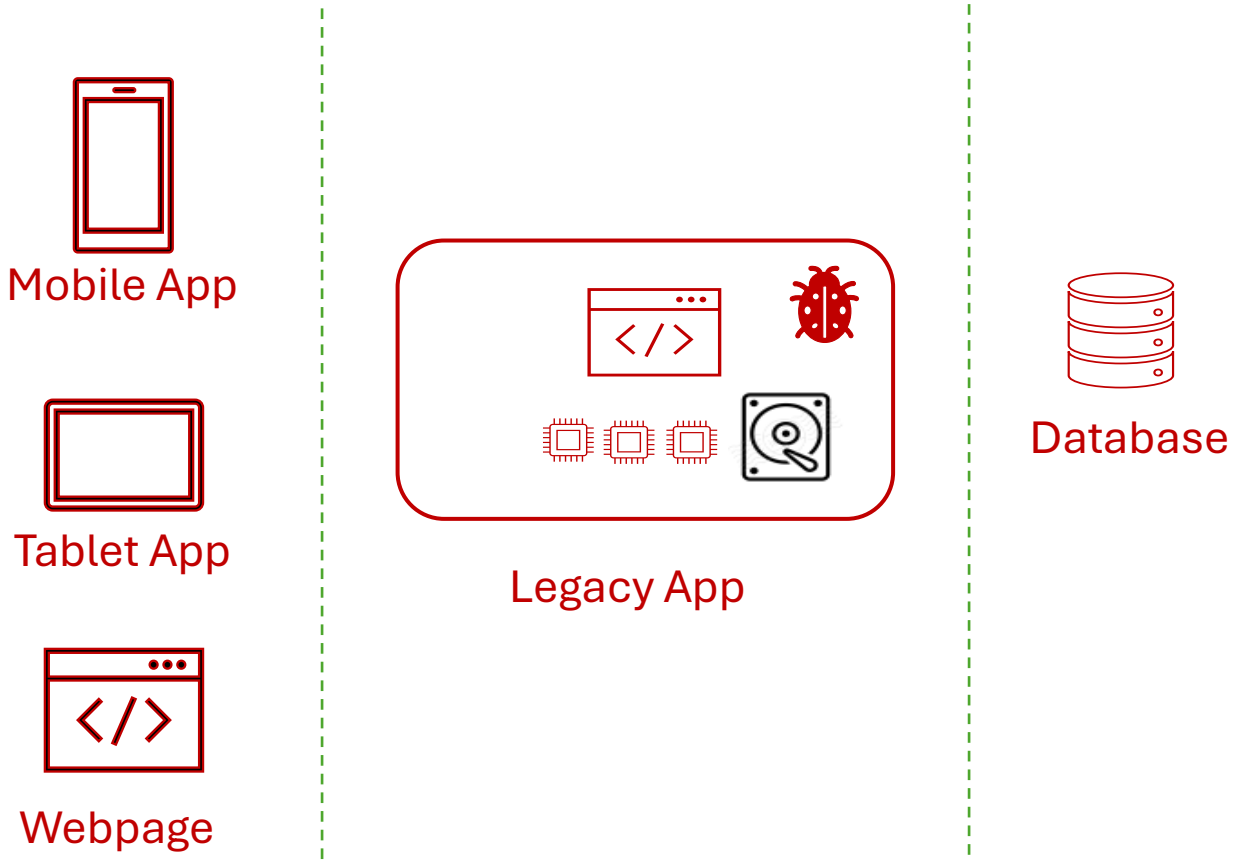
Issues with Monolithic Architecture



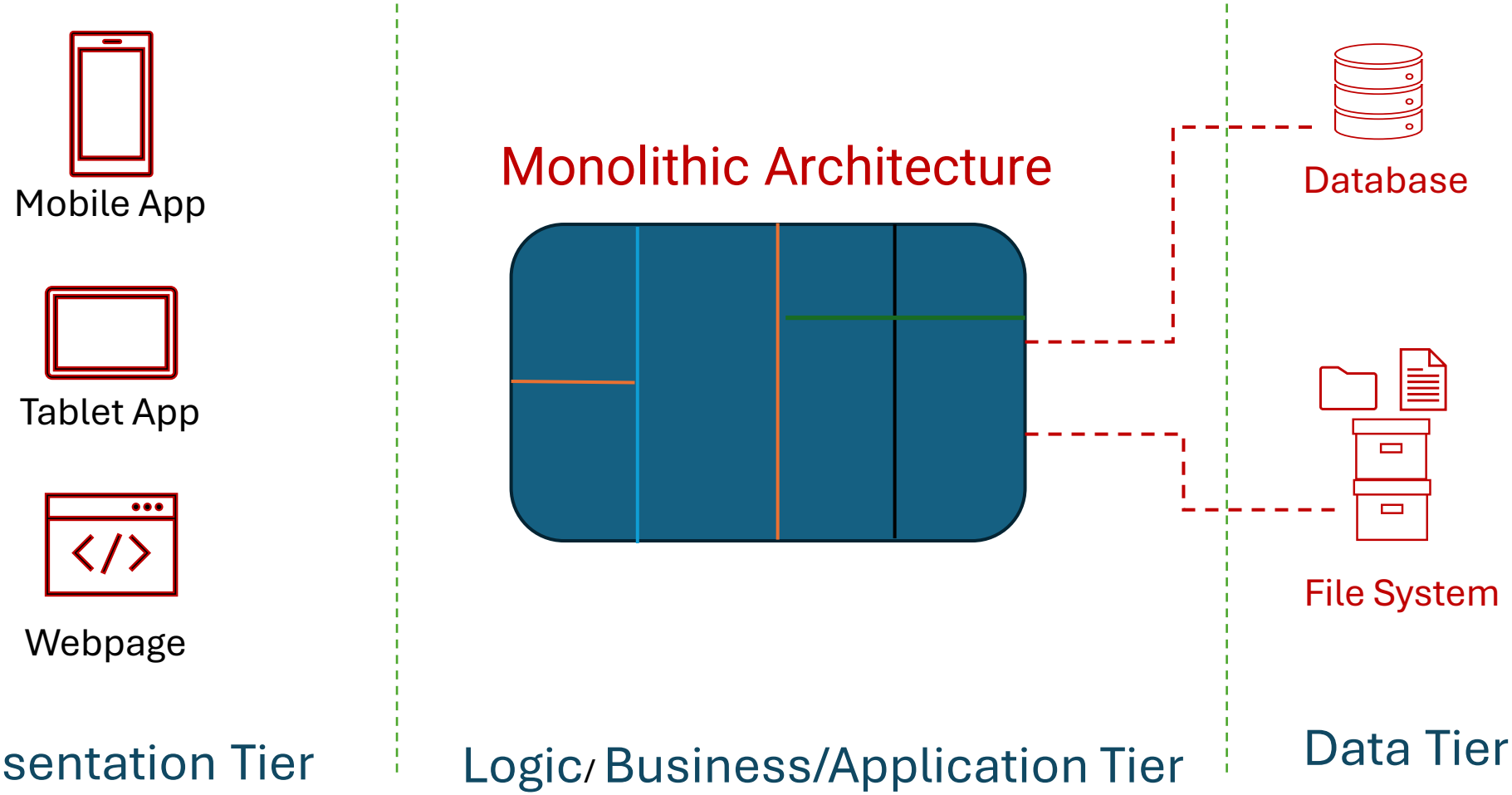
Issues with Monolithic Architecture



Issues with Monolithic Architecture



Logical Separation Alternative?



Summary

- Motivation for :
  - Microservices Architecture
  - Event-driven architecture
- Microservices:
  - The most significant contributors to many companies' success
  - Not a "silver bullet"
- 3-Tier/Monolithic Architecture
  - Perfect for:
    - Startup companies
    - Companies with a small dev team (2 Pizza Rule)
- Monolithic Architecture has issues with :
  - Organizational Scalability
  - System Scalability

Summary

- Solution:
  - Microservices Architecture with Event-Driven Architecture

Microservices Architecture

Benefits and Challenges

Topics

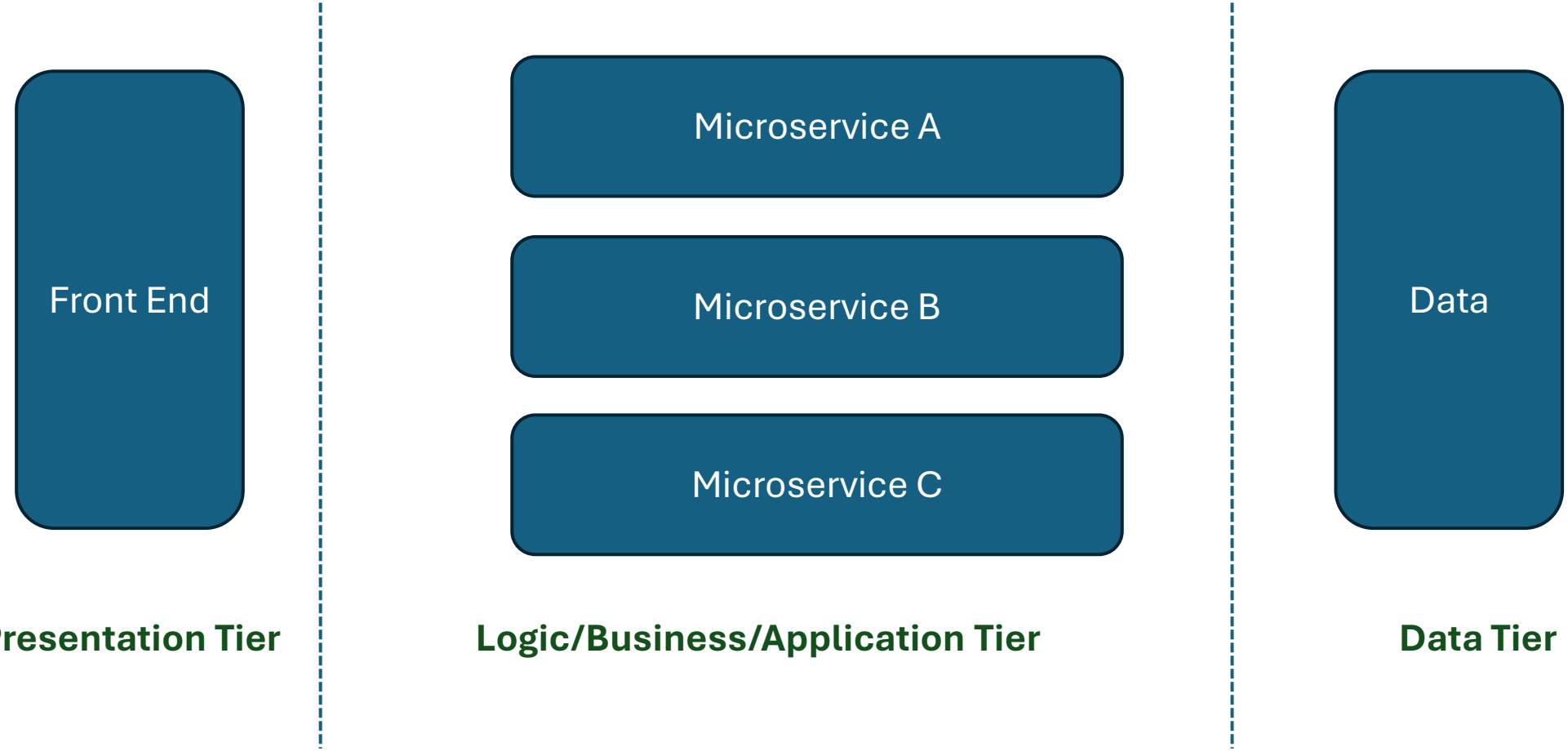
- Introduction to Microservices
- Benefits of Microservice Architecture
- Challenges of Microservices Architecture

Topics

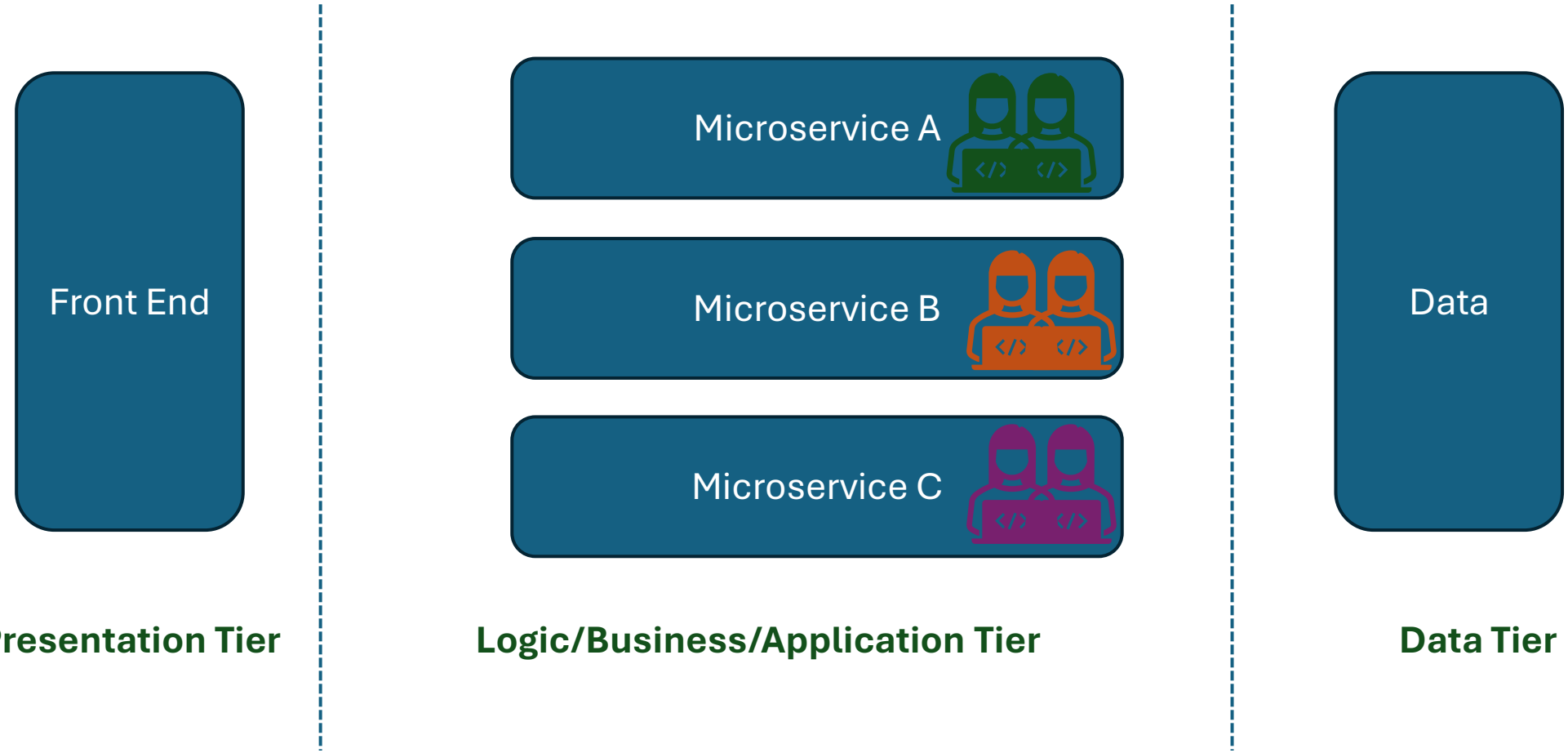
- Introduction to Microservices
- Benefits of Microservice Architecture
- Challenges of Microservices Architecture

What is Microservices Architecture

What is Microservices Architecture



What is Microservices Architecture



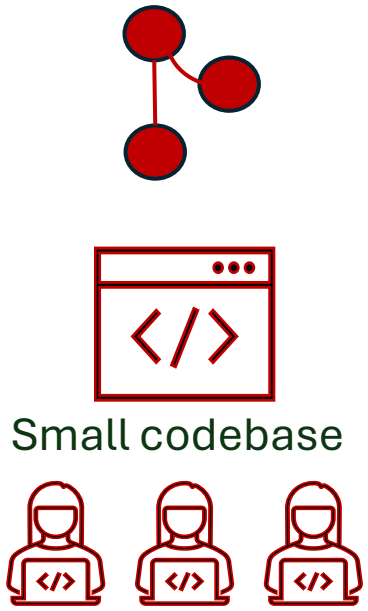
Topics

- Introduction to Microservices
- Benefits of Microservice Architecture
- Challenges of Microservices Architecture

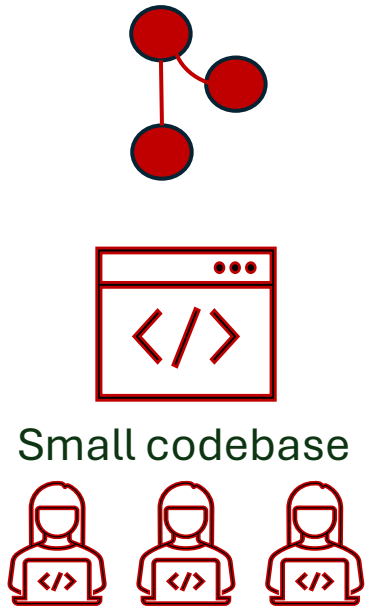
Microservices- Benefits

1. High Organizational Scalability

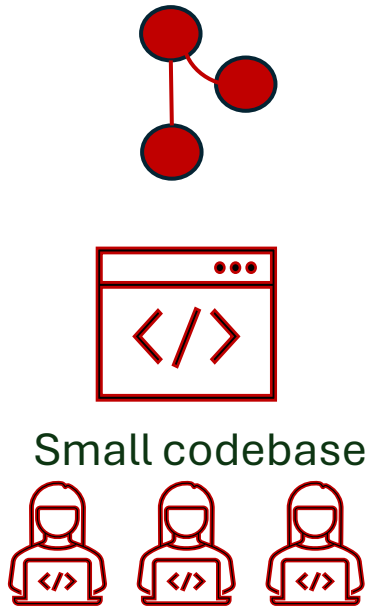
Smaller Codebase



Microservices 1 Team

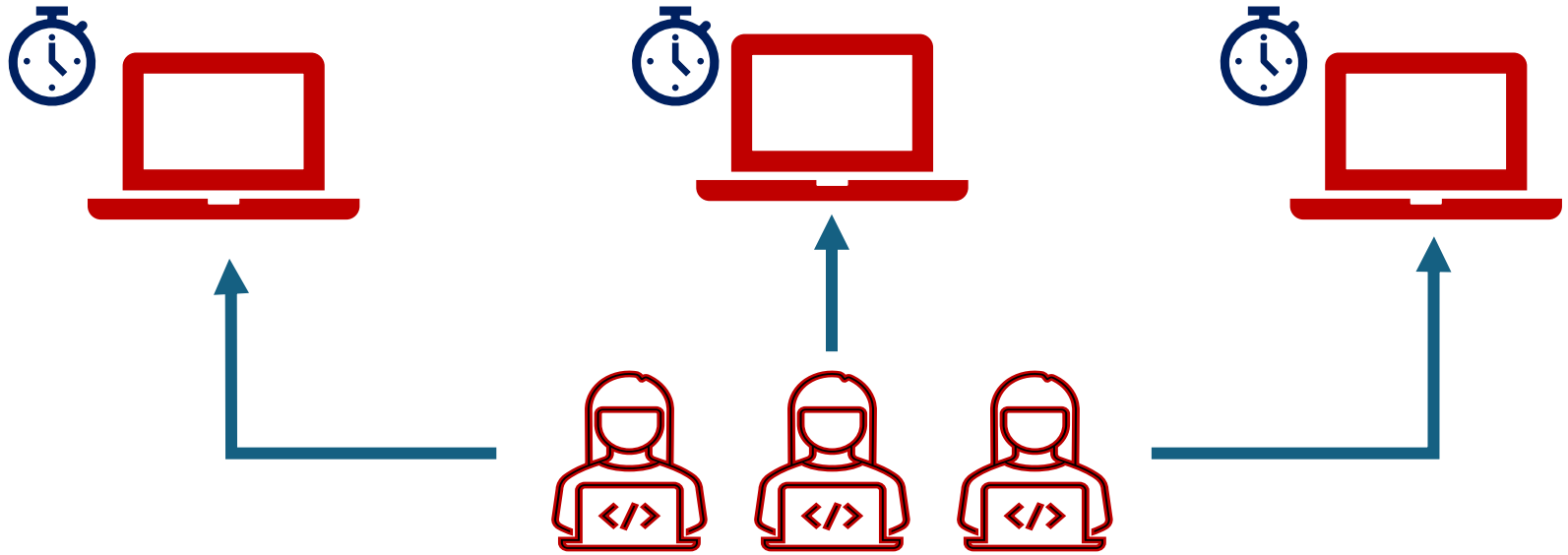


Microservices 2 Team

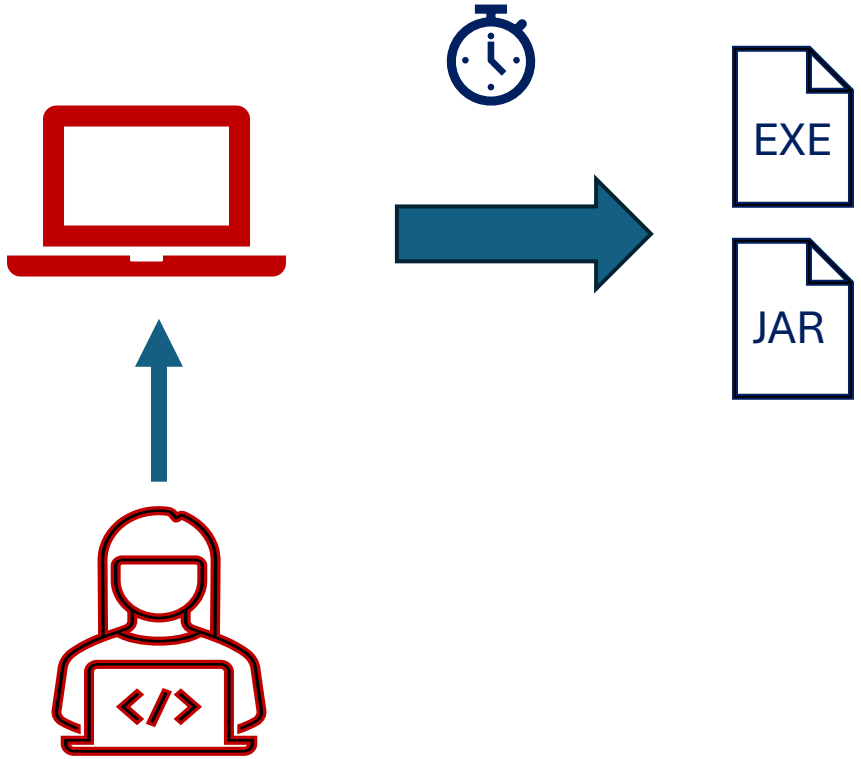


Microservices 3 Team

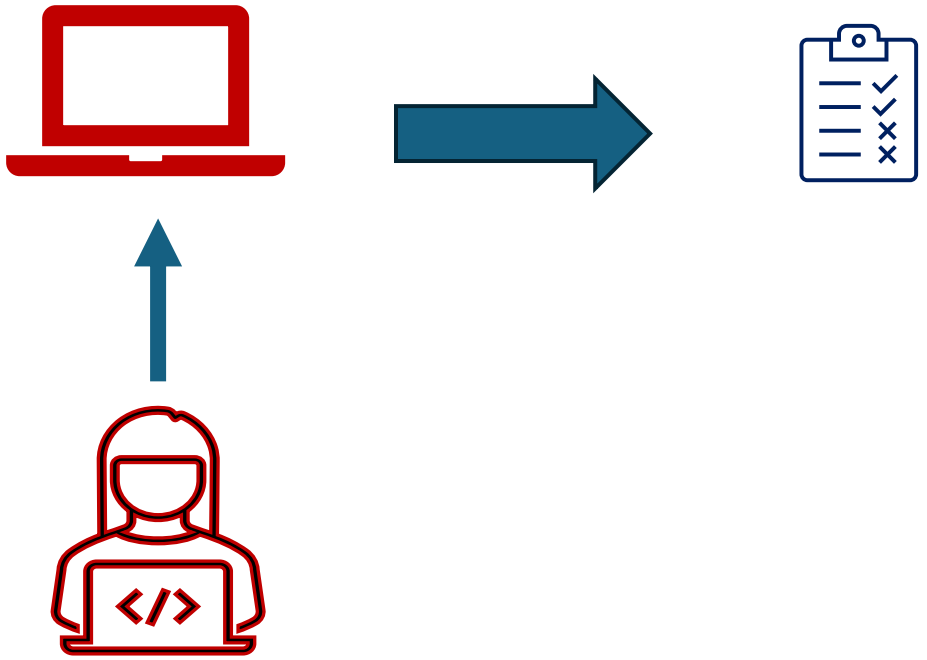
Fast Load Time



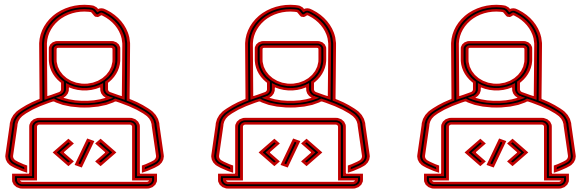
Shorter Build Time



Easier to Test and Understand



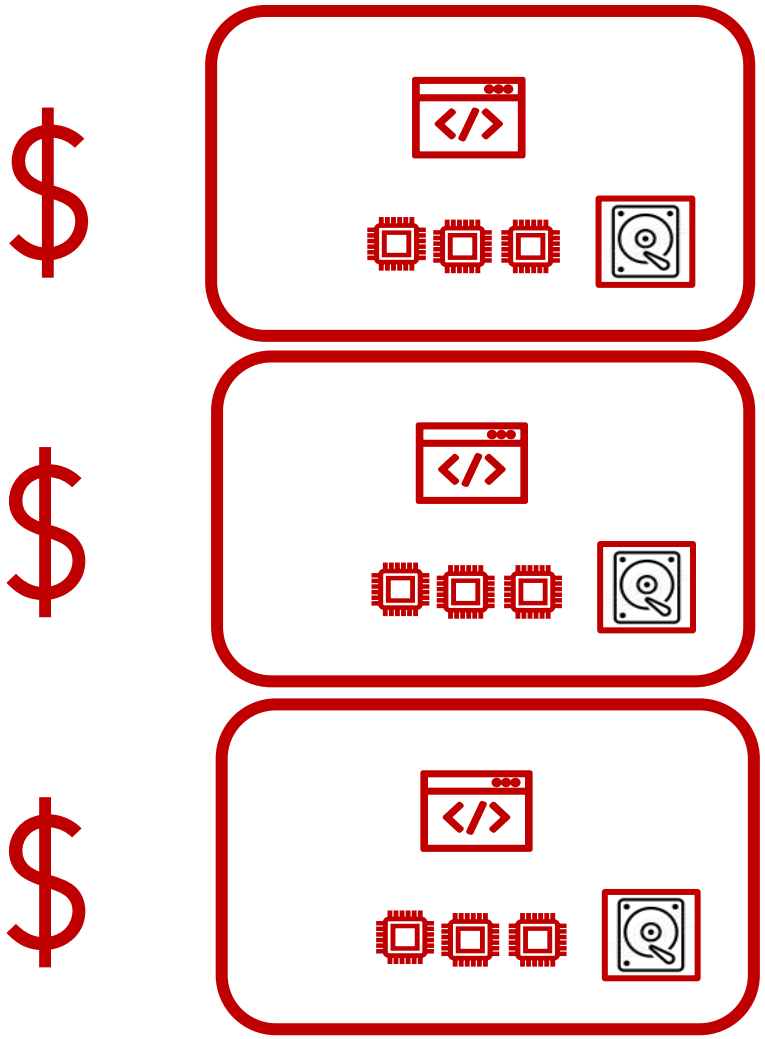
Higher Team Velocity



Microservice - Benefits

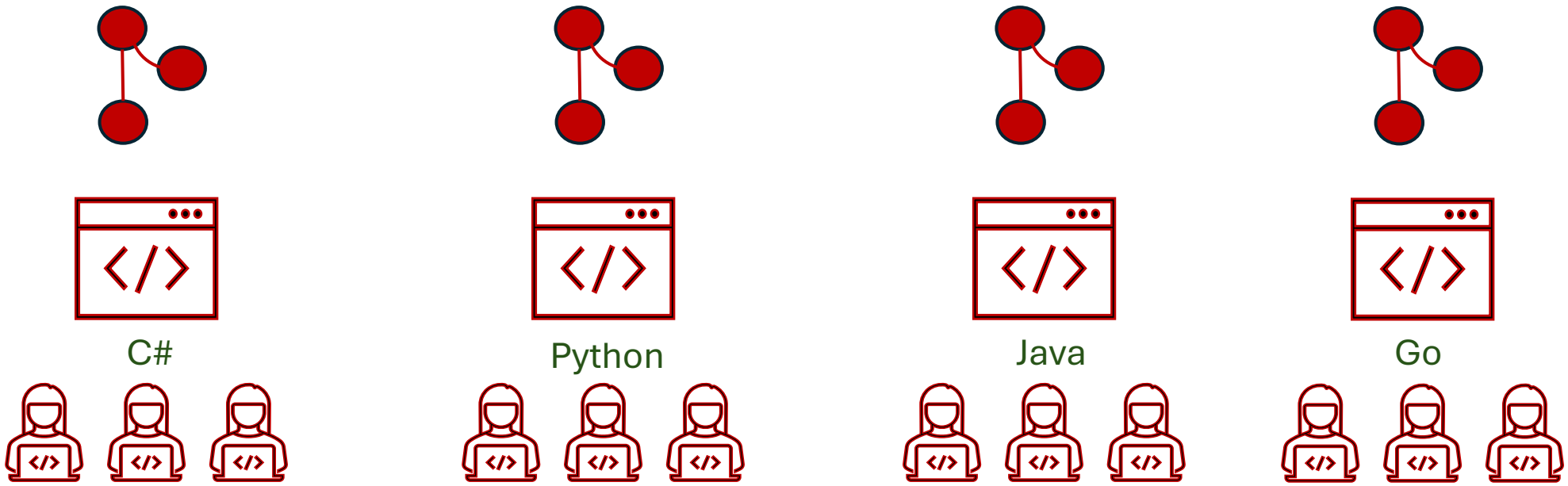
- High Organizational Scalability
- High System Scalability

Monolithic Application Instances

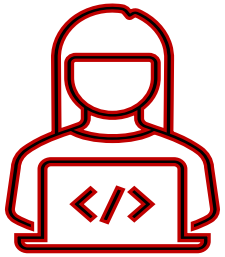


Microservices Instances

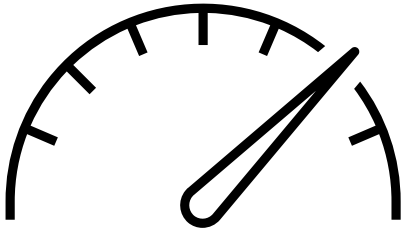
Different Technologies for Each Microservice

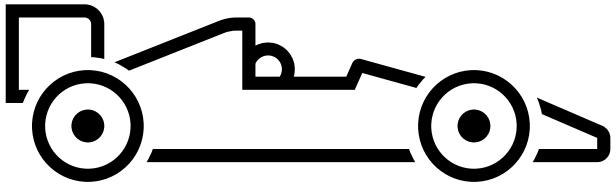


Higher Stability

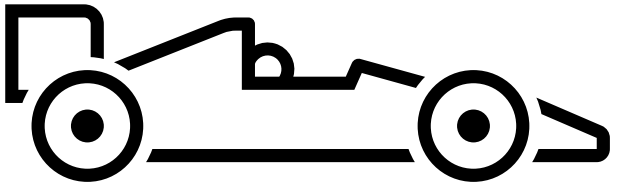


Monolithic to Microservices





Monolithic Architecture
Speed range (Lower gear)



Microservices Architecture
Speed range (High gear)

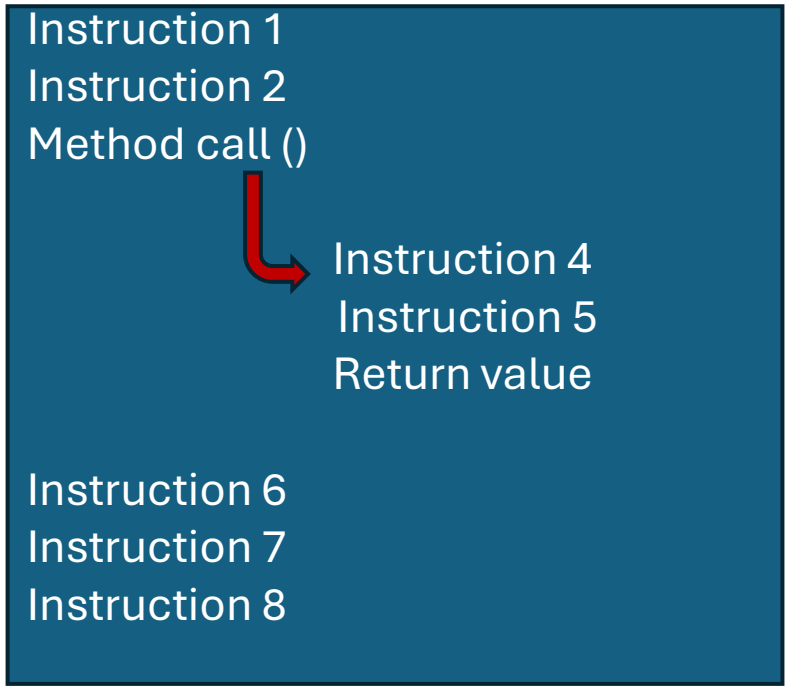
Topics

- Introduction to Microservices
- Benefits of Microservice Architecture
- Challenges of Microservices Architecture

Microservice - Challenges

1. Microservices is a highly distributed system

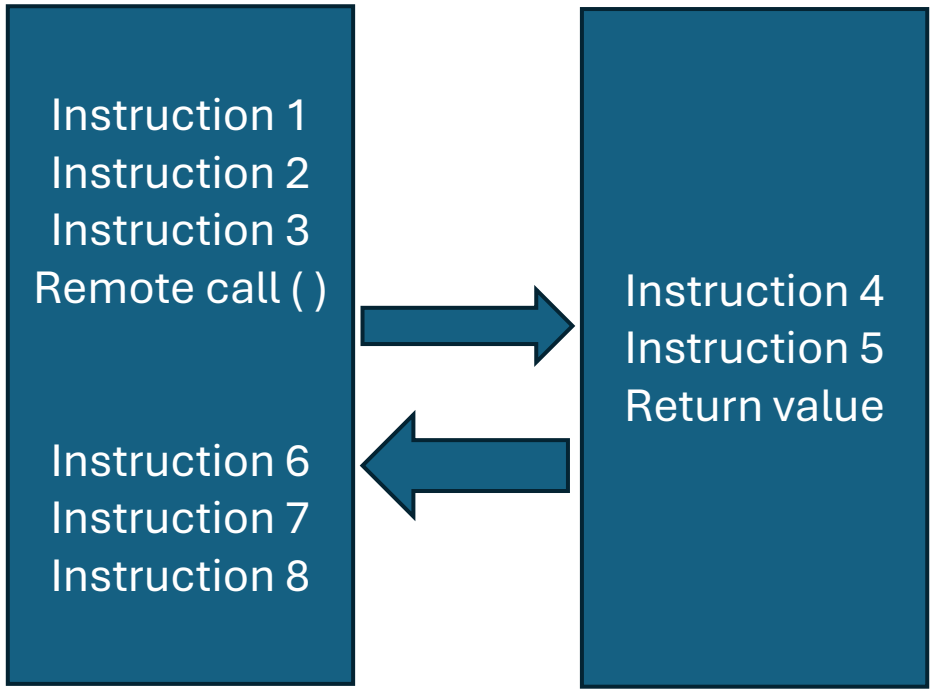
Centralized System



Monolithic application code

- ❑ Predictable behavior
- ❑ Predictable success
- ❑ Predictable performance

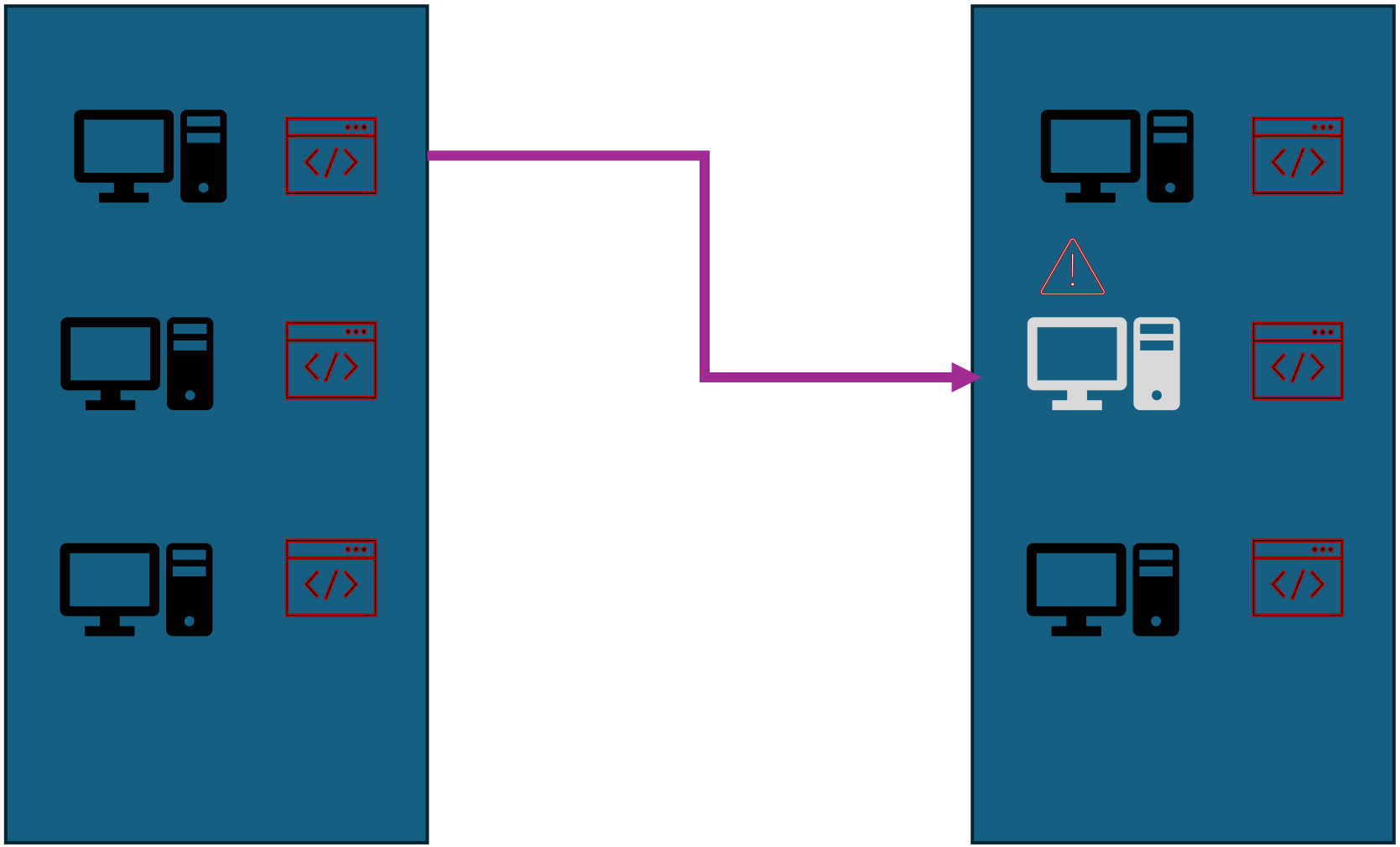
Distributed System



Microservice A

Microservice B

X Predictable behavior
X Predictable success
X Predictable performance

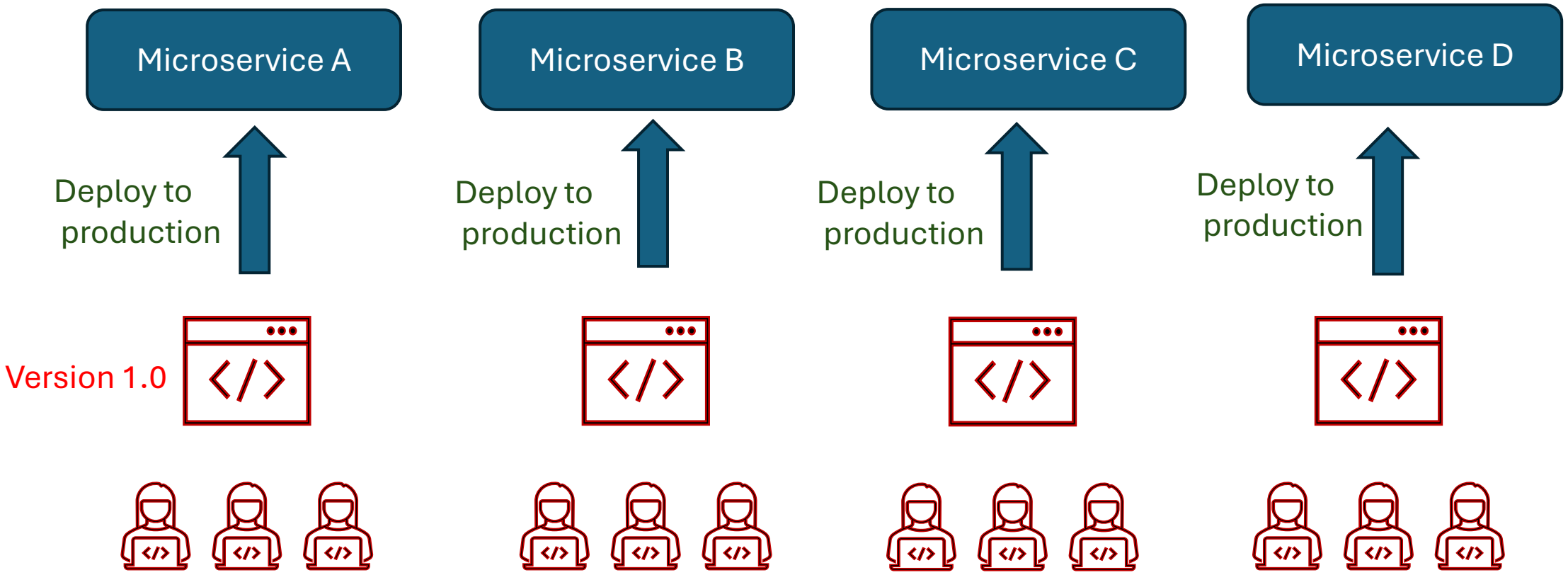


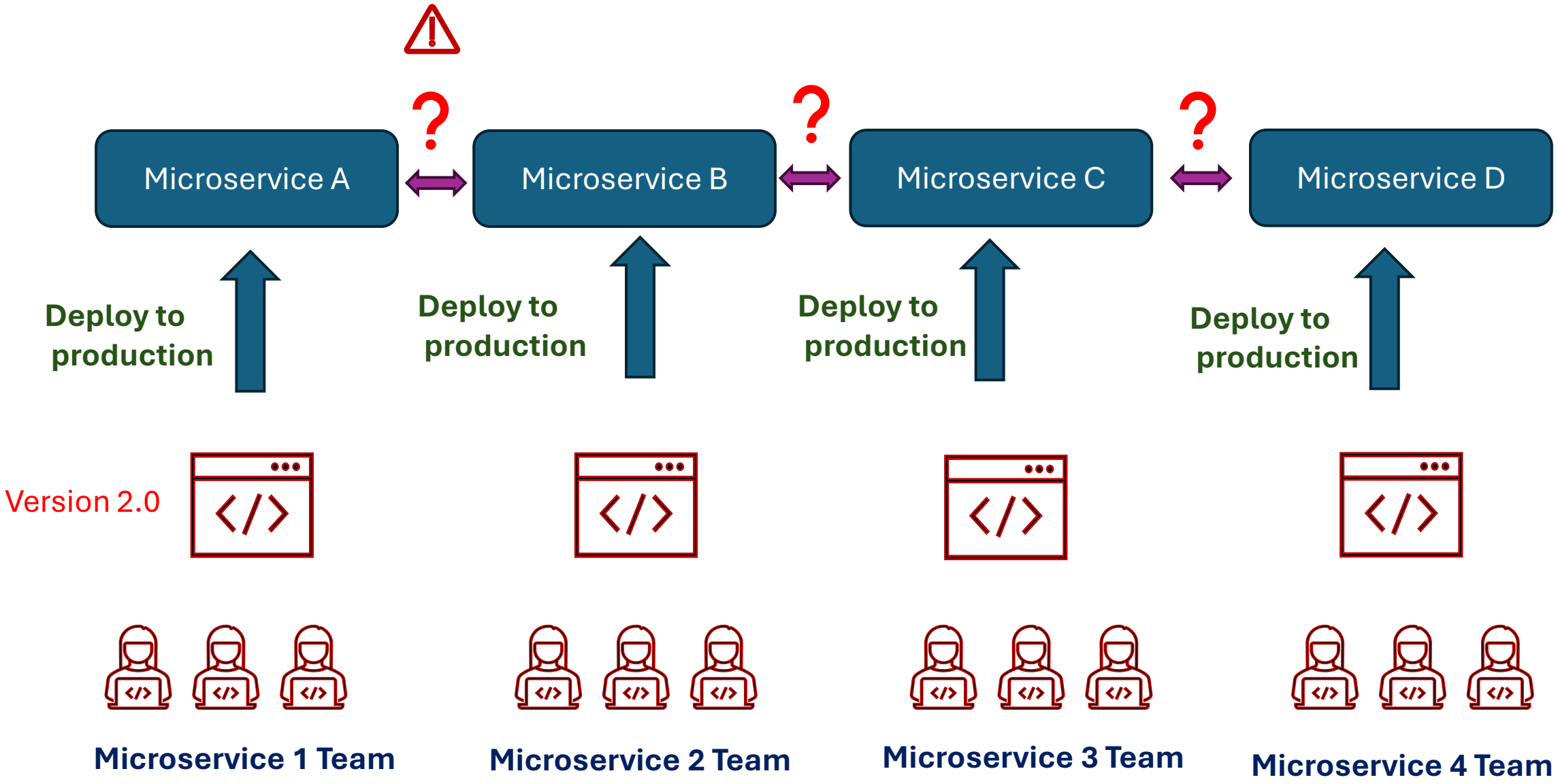
Microservice A

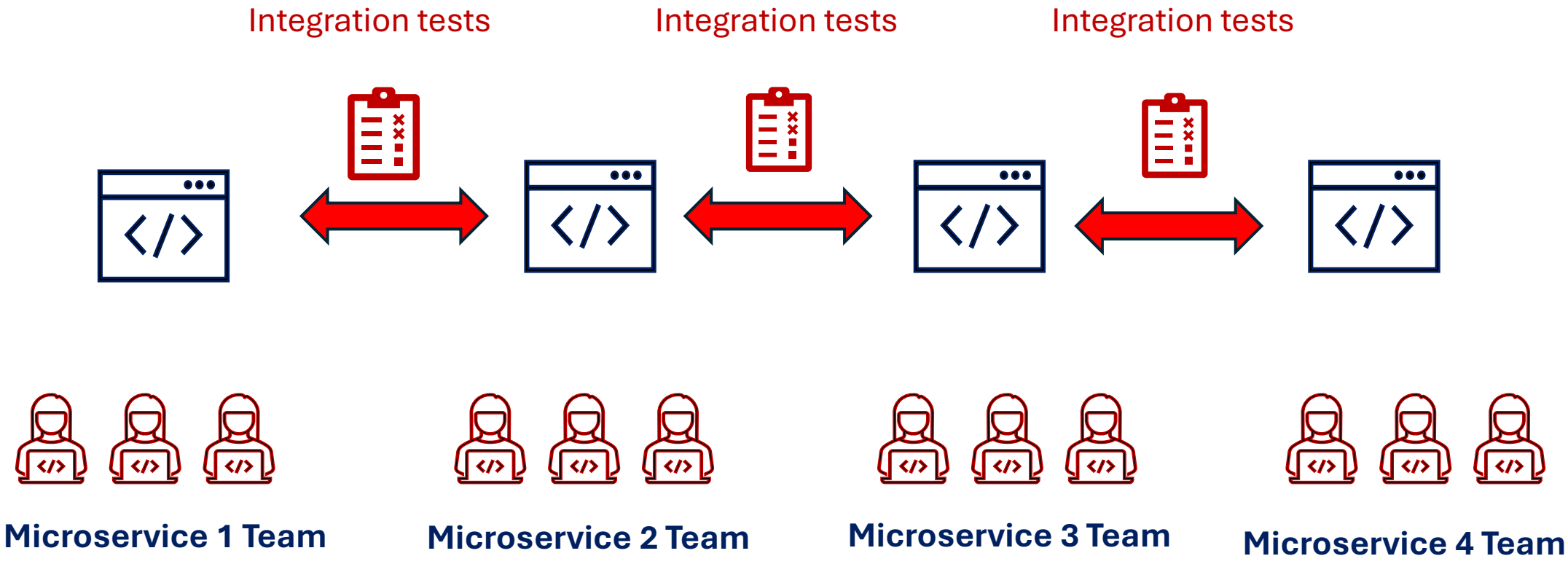
Microservice B

Microservice - Challenges

1. Microservices is a highly distributed system
2. Testing Microservices

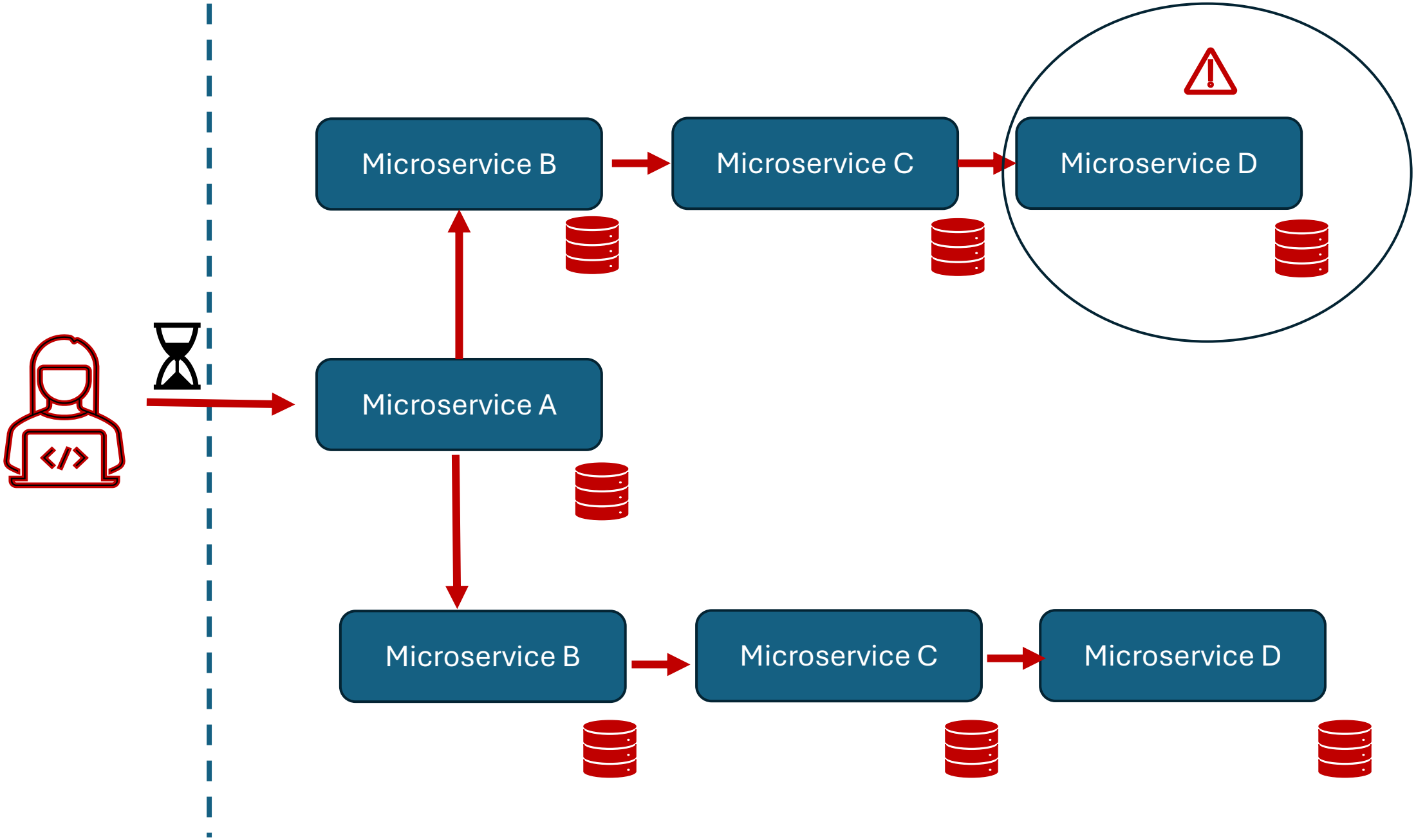

Microservice A
Microservice B
Microservice C
Microservice D
Deploy to production
Deploy to production
Deploy to production
Deploy to production
Version 1.0


Microservice A
Microservice B
Microservice C
Microservice D
Deploy to production
Deploy to production
Deploy to production
Deploy to production
Version 2.0
Microservice 1 Team
Microservice 2 Team
Microservice 3 Team
Microservice 4 Team


Integration tests
Integration tests
Integration tests
Microservice 1 Team
Microservice 2 Team
Microservice 3 Team
Microservice 4 Team

Microservice - Challenges

1. Microservices is a highly distributed system
2. Testing Microservices
3. Troubleshooting and debugging


Microservice B
Microservice C
Microservice D
Microservice A
Microservice B
Microservice C
Microservice D

Microservice - Challenges

1. Microservices is a highly distributed system
2. Testing Microservices
3. Troubleshooting and debugging
4. Organizational scalability

Responsibilities

Payment

Product images

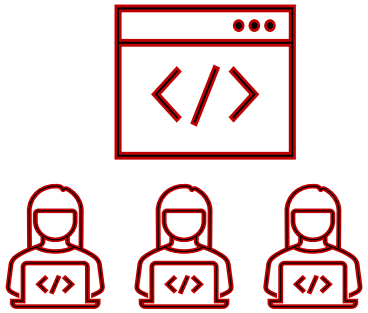
Authentication

orders

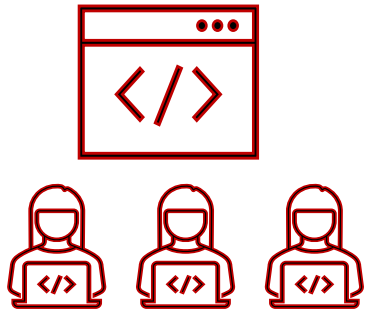
Authorization

Inventory

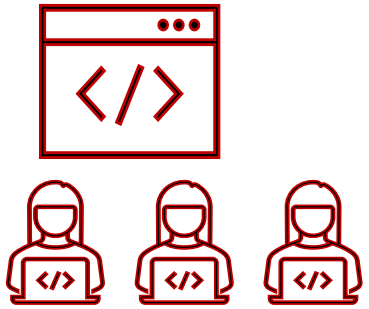
Returns



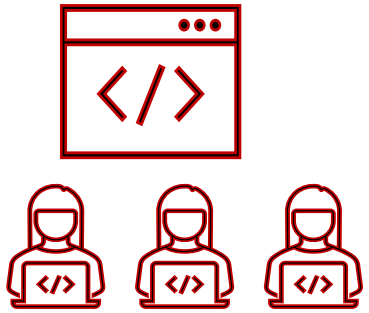
Microservice 1 Team



Microservice 2 Team

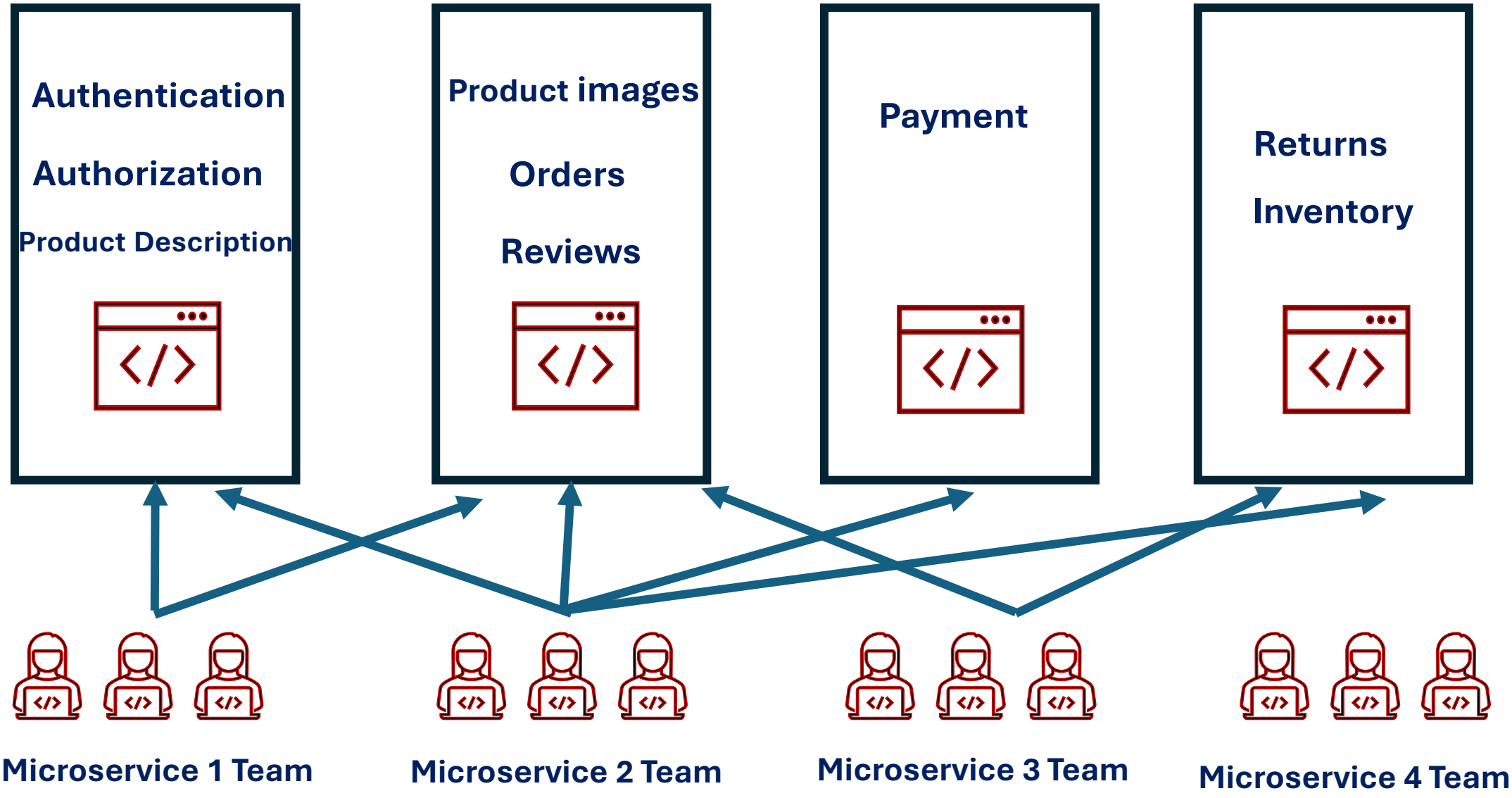


Microservice 3 Team

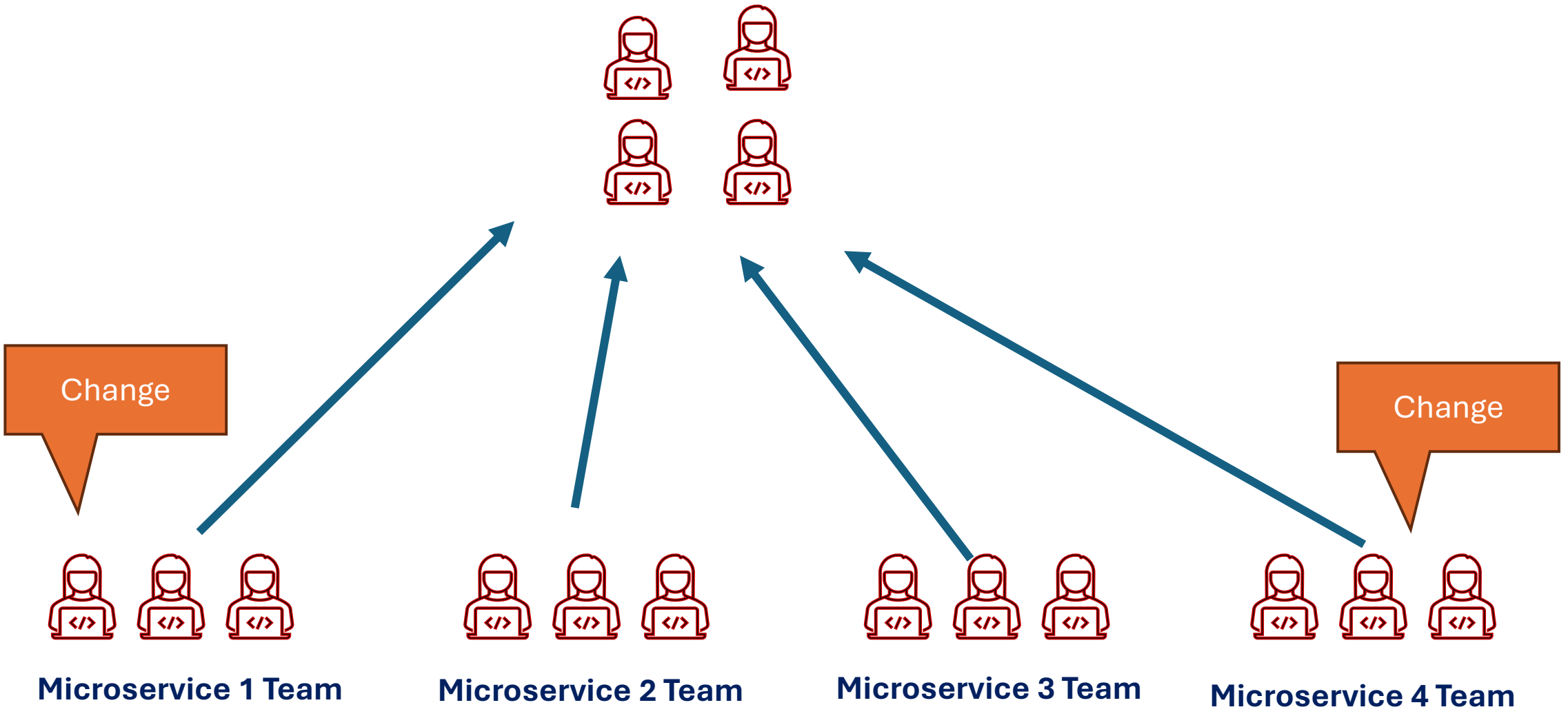


Microservice 4 Team

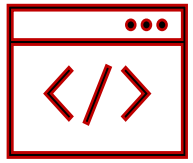
Responsibilities



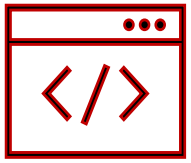
Coordination



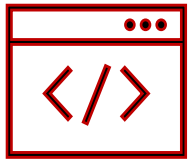
Duplicated Effort



Java
Spring boot
Maven
gRPC



Python
Django
Sprinnaker
SOAP



C#
ASP.NET
NuGet
TeamCity
WCF

Microservice 1 Team

Microservice 2 Team

Microservice 3 Team

Microservice - Challenges

1. Microservices is a highly distributed system
2. Testing Microservices
3. Troubleshooting and debugging
4. Organizational scalability

“Distributed Monolith”

“Big Ball of Mud”

Good News!

- Many companies have already gone through all those growing pains
- Software architects have been sharing :
  - Knowledge
  - Success stories
  - Mistakes
- We now have a set of:
  - **Principles**
  - **Best practices**

Summary

- Learn about Microservices Architecture

- **Benefits**
  - **Organizational scalability**
  - **System scalability**

- **Challenges:**
  - **The complexity of running a distributed system**
  - **Risk of decreased organizational scalability**

Time for

a break

Microservices Boundaries

Core principles

Topics

- System Introduction
- Attempt 1: Splitting by Application Layers
- Attempt 2: Splitting by Technology Boundaries
- Attempt 3: Splitting for Minimum Size

Topics

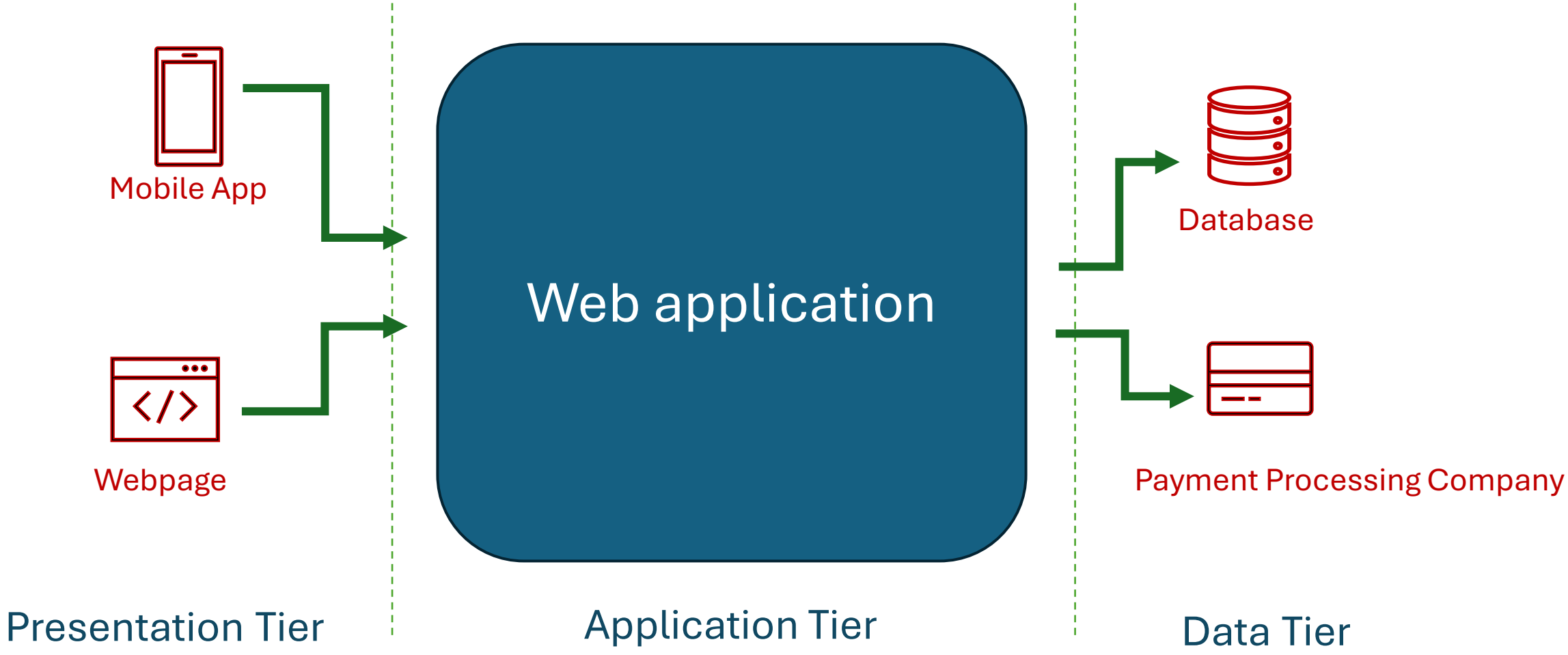
- System Introduction
- Attempt 1: Splitting by application layers
- Attempt 2: Splitting by Technology Boundaries
- Attempt 3: splitting for Minimum size

Statement

Just breaking a large code base into an arbitrary set of microservices

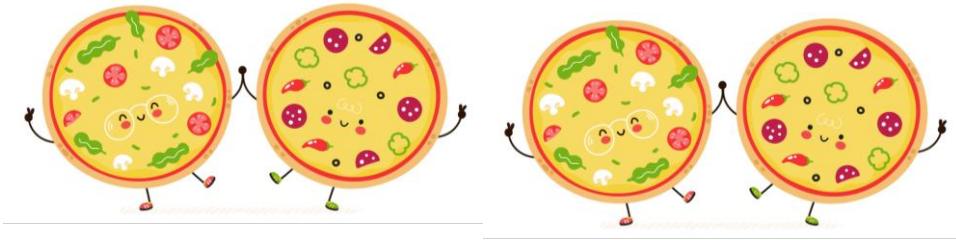
Monolithic E-Commerce Application

Monolithic E-Commerce Application



Monolithic E-Commerce Application

- The codebase is too large and complex
- Binary size is big – requires expensive hardware
- The development team is too big

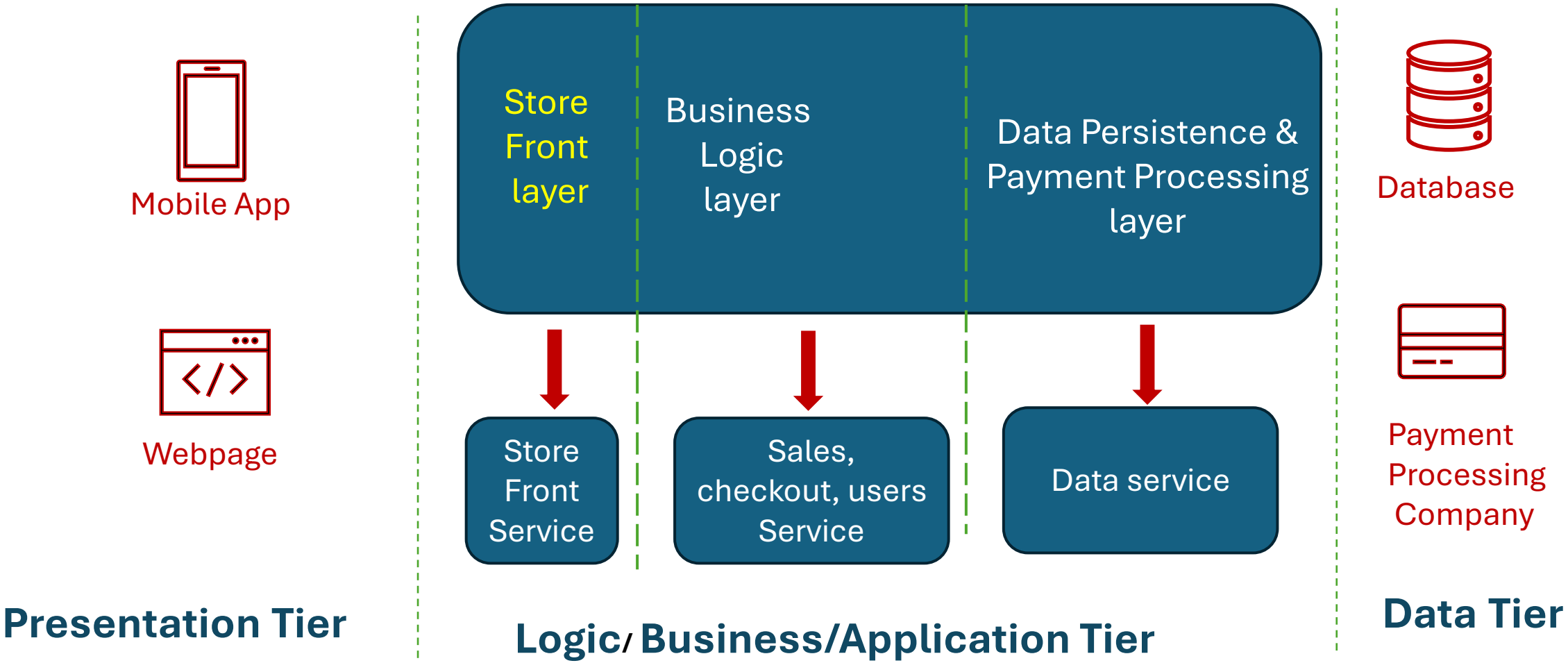


Microservices!?

Topics

- System Introduction
- Attempt 1: Splitting by application layers
- Attempt 2: Splitting by Technology Boundaries
- Attempt 3: splitting for Minimum size

Splitting by Application Layers

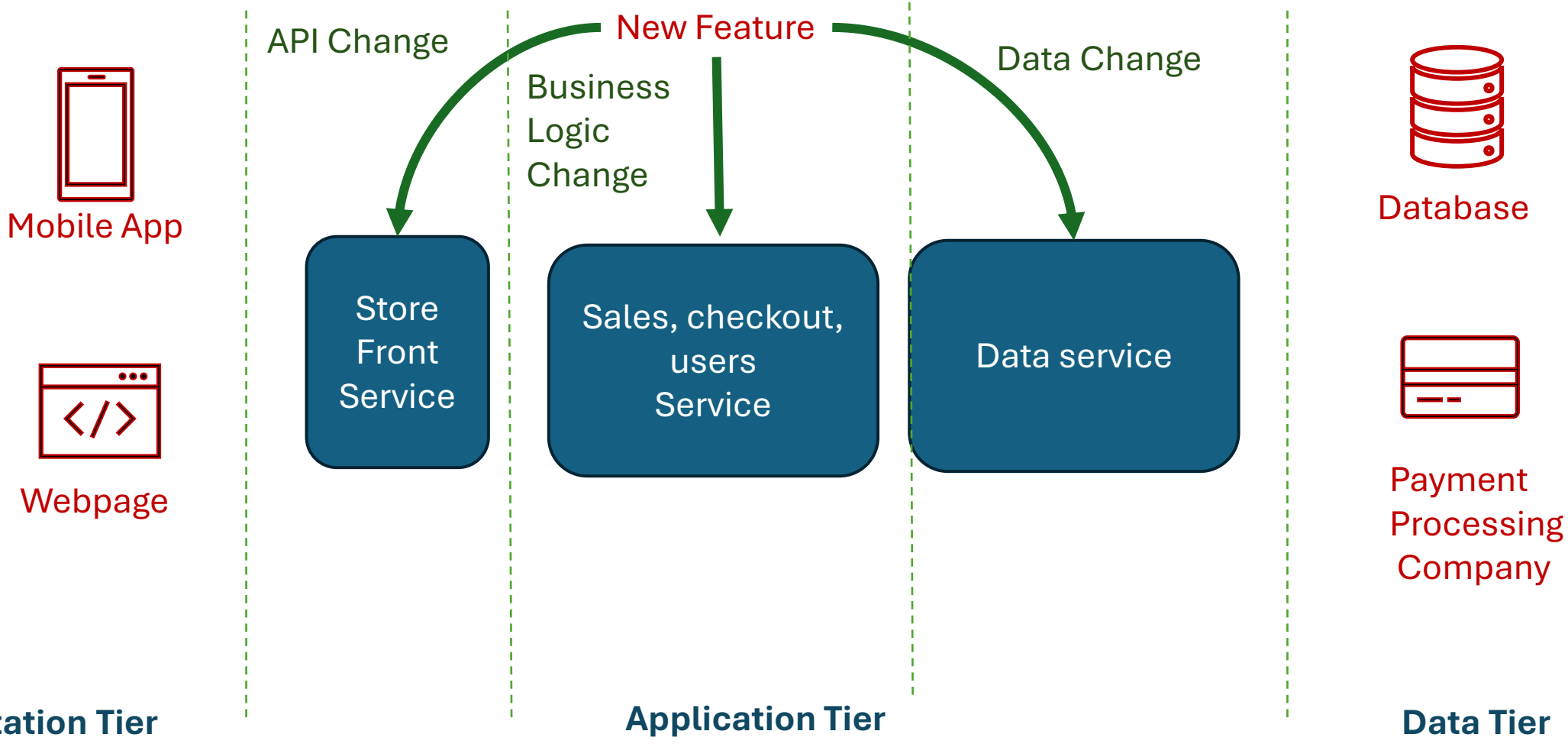


Attempt 1: Results

- Seems like a good idea!
  - Takes advantage of existing logical layers
  - No major refactoring is required

- This approach does not work

Splitting by Application Layers



Microservices Boundaries – Core Principles

- **Cohesion:**

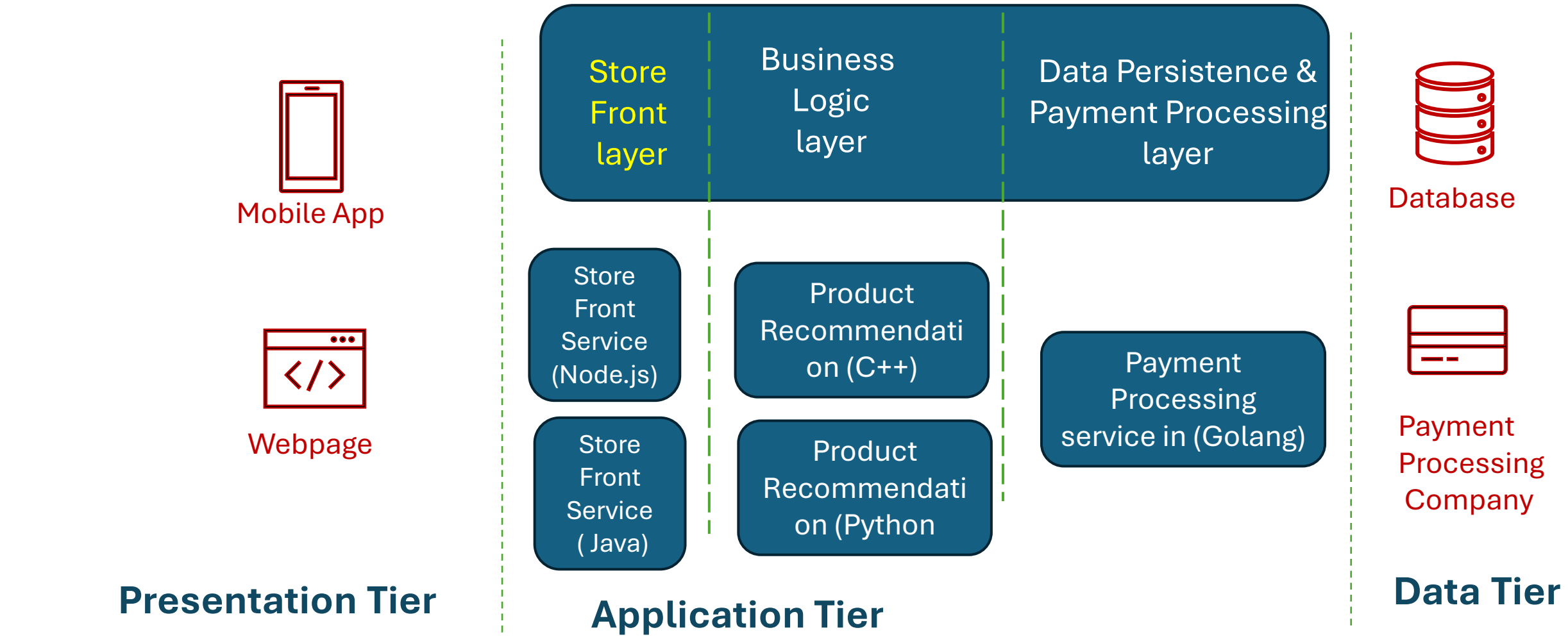
Elements that are tightly related to each other and change together should stay together.

  - The logic that changes together stays within the boundaries of the same microservices
  - Each team can operate independently
  - Our migration failed b/c microservices were not cohesive

Topics

- System Introduction
- Attempt 1: Splitting by application layers
- Attempt 2: Splitting by Technology Boundaries
- Attempt 3: splitting for Minimum size

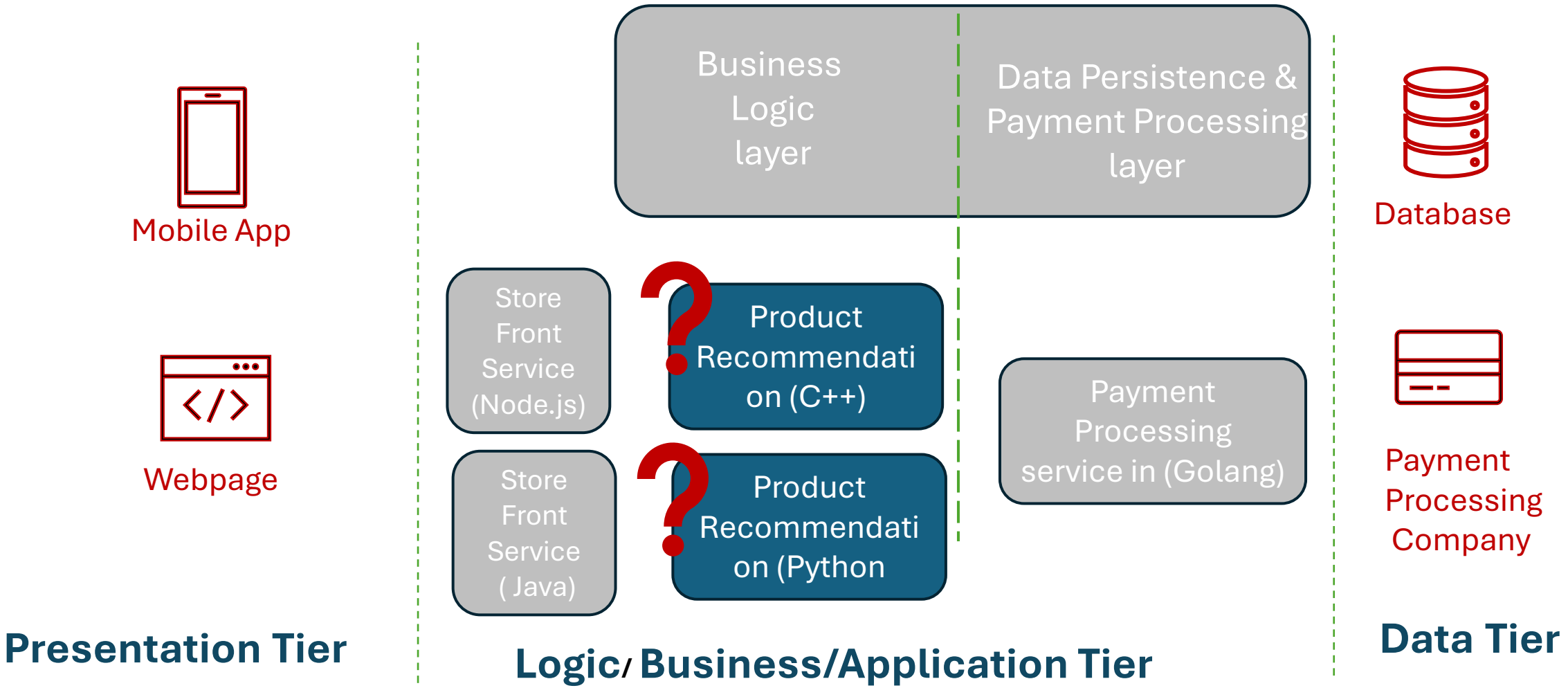
Splitting by Application Layers



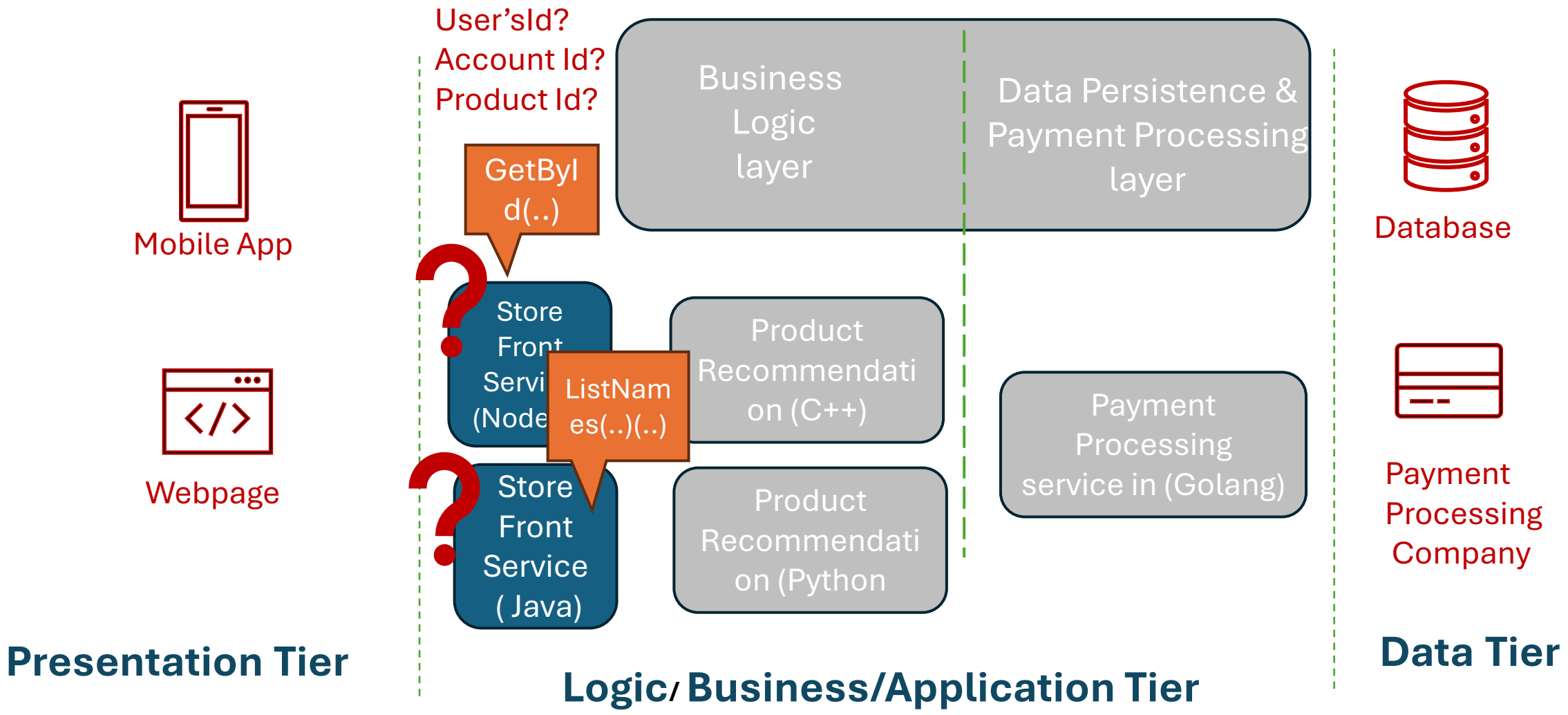
Attempt 2: Results

- Boundaries are purely technological
- Outside stakeholders don’t know which subteam/service gets the task

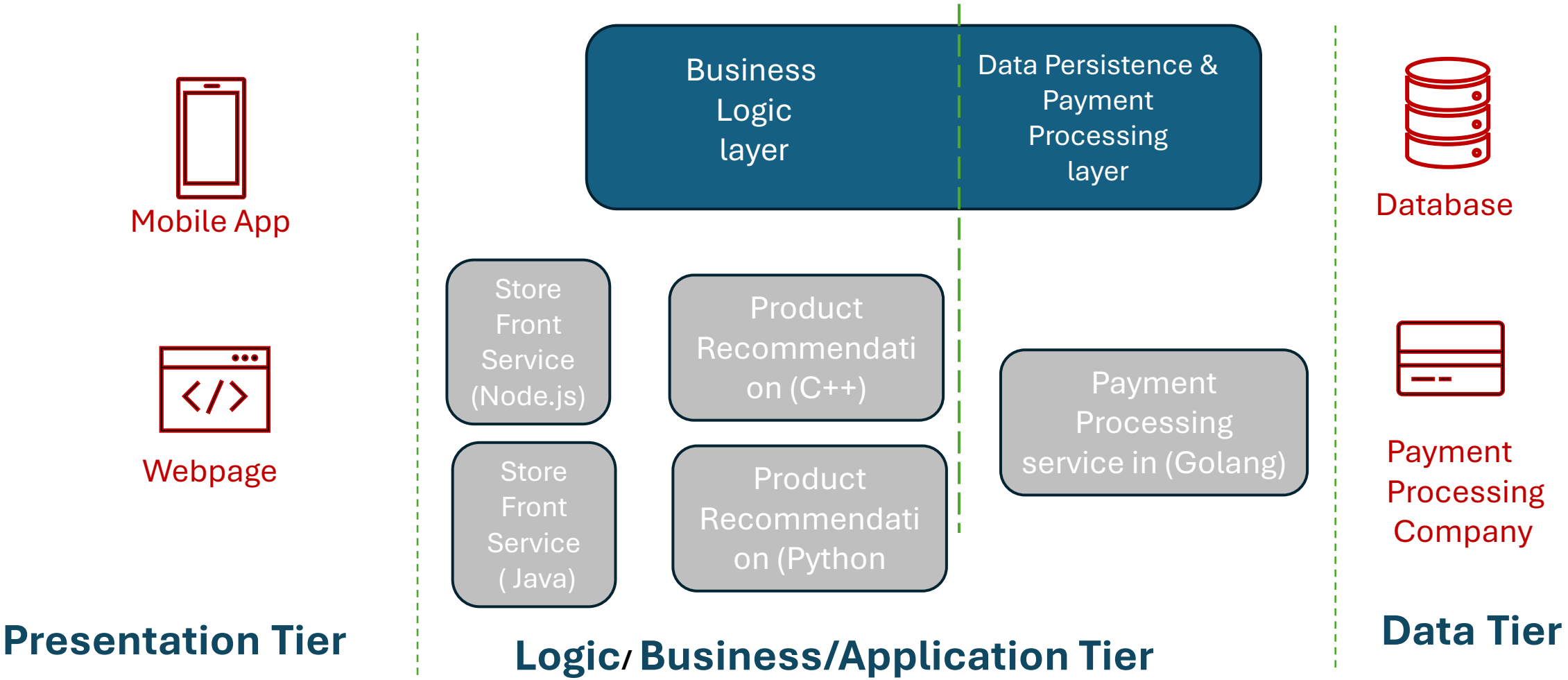
Splitting by Application Layers



Splitting by Application Layers



Splitting by Application Layers



Microservices Boundaries – Core Principles

1. Cohesion
2. Single-Responsibility Principle (SRP)

Microservices Boundaries – Core Principles

1. Cohesion
2. Single-Responsibility Principle (SRP)

Every microservice should do only one thing

Single Responsibility Principle

- There’s no ambiguity about:
  - Where new functionality needs to go
  - Which team owns what
- Create an easy-to-follow API
  - Terminology is bound to a given context
  - Examples:
    - Users Service- Users context
    - Product service- Product context

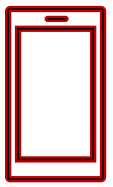
Topics

- System Introduction
- Attempt 1: Splitting by application layers
- Attempt 2: Splitting by Technology Boundaries
- Attempt 3: Splitting for Minimum Size

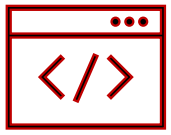
Splitting for Minimum Size

- **Micro**services
- Assumption – Splitting into tiny services would give us the best benefits

Splitting by Minimum Size



Mobile App



Webpage

Presentation Tier

Store Front layer

Business Logic layer

Data Persistence & Payment Processing layer

SF-Package 1

SF-Package 2

SF-Package 3

SF-Package 4

User-Pkg 1

User-pk2

Prod.pkg1

review.pkg1

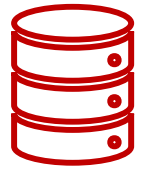
user table 1

User table 2

Product table1

Prod table 2

Application Tier

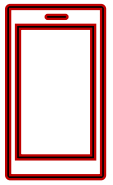


Database

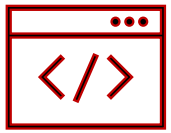
Payment Processing Company

Data Tier

Splitting by Technology Boundaries



Mobile App



Webpage

Presentation Tier

Microservice

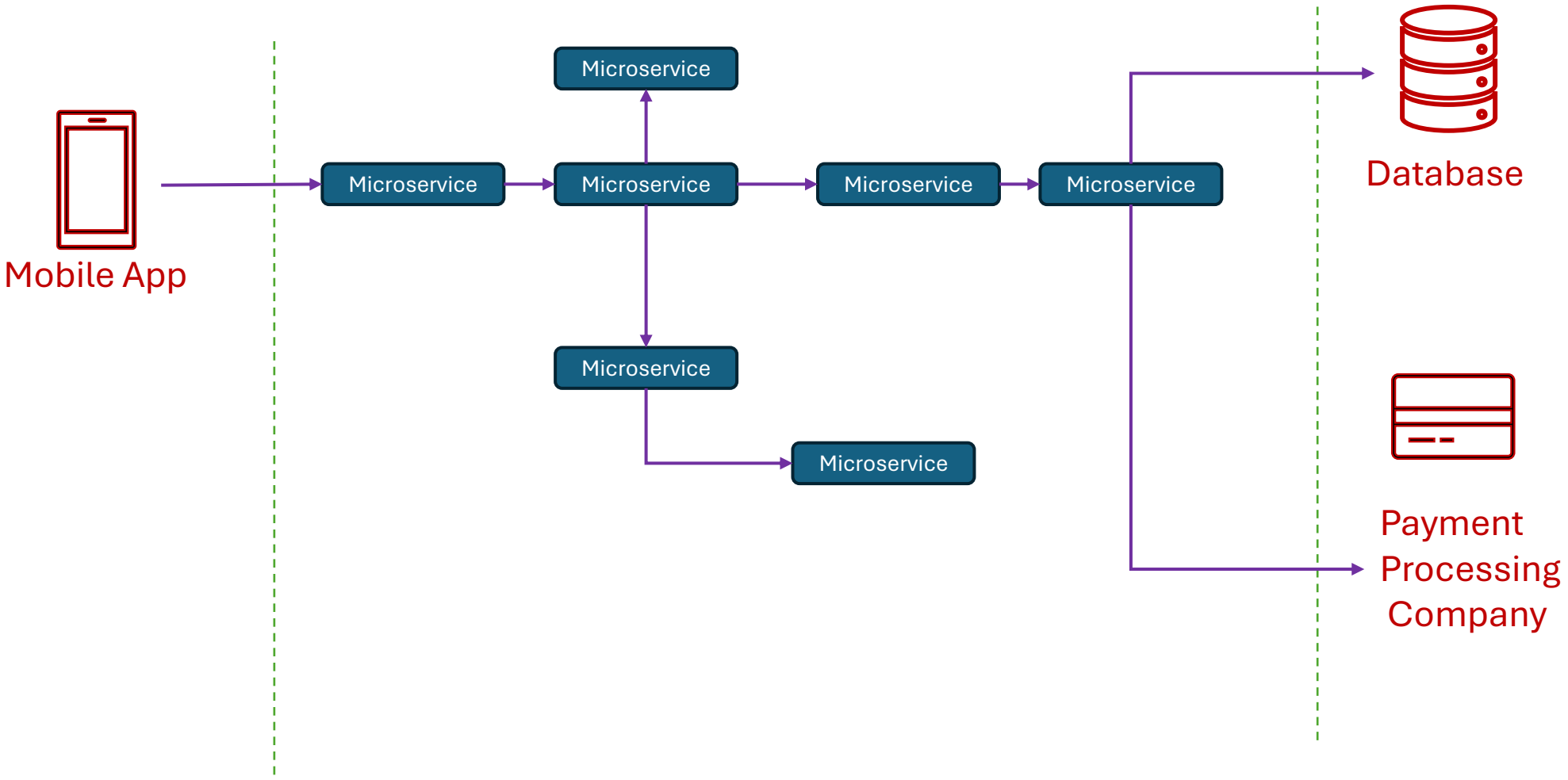
Application Tier

Database

Payment Processing Company

Data Tier

Attempt 3: Results



Microservices Boundaries – Core Principles

1. Cohesion
2. Single-Responsibility Principle (SRP)
3. Loose Coupling

Loose Couple

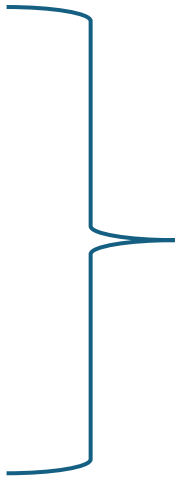
- Little or no interdependencies
- Minimum communication with other microservices

Important Note

- The size of a microservice is not important
- As long as the microservices are:
  - Cohesive
  - Follow the single responsibility principle
  - Loosely coupled
- Different microservices may have different sizes

Microservices Boundaries – Core Principles

1. Cohesion
2. Single-Responsibility Principle (SRP)
3. Loose Coupling



Prerequisites for Successful Microservice Architecture

How do you split a Monolithic application into Microservices?

Summary

- Core principles for Microservices Boundaries:
  - Cohesion
  - SRP
  - Loose Coupling
- The size of the **Micro**services doesn’t matter

Microservices

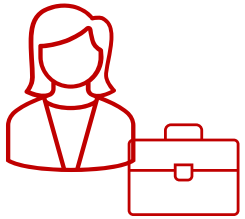
Decomposition of a Monolithic Application to Microservices

Topics

- Decomposition by Business Capabilities
- Decomposition by domain/Subdomain
- Decomposition by business Capabilities vs Subdomains

Topics

- Decomposition by Business Capabilities
- Decomposition by domain/Subdomain
- Decomposition by business Capabilities vs Subdomains



System

1. Decomposition by Business Capabilities

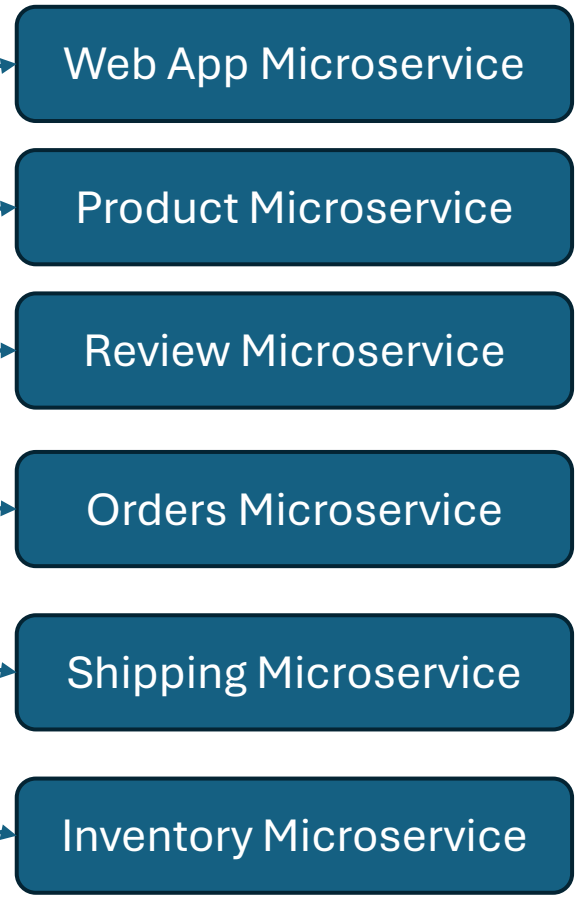
- Core capability that provides value to:
  - Business
  - Customers
- Examples:
  - Revenue
  - Marketing
  - Customer experience
- Each capability → **Microservice**

Identify Business Capabilities

- Run a thought experiment:
  - “Describe the system to a non-technical person.”
  - Explain what the system does / what value each capability provides

Example: Online Store

- **Browse** through products
- Search/view for **products**
- Read the **reviews**
- place an **order**
- **Shipping** of the order
- Update/maintain **inventory**



How do we know that those services follow our three core principles?

Single Responsibility Principle

Orders Microservice

Shipping Microservice

Inventory Microservice

Web App Microservice

Product Microservice

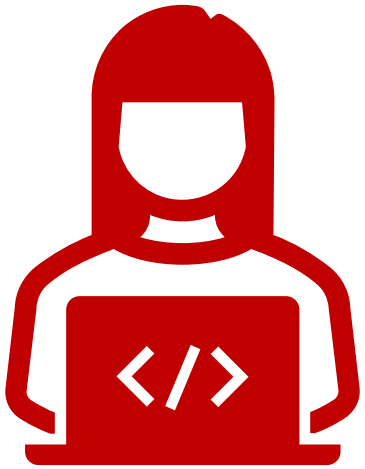
Review Microservice

1. **SRP**
2. **Cohesive**
3. **Loosely Coupled**

Topics

- Decomposition by Business Capabilities
- Decomposition by Domain/Subdomain
- Decomposition by business Capabilities vs Subdomains

2. Decomposition by Domain/Subdomain



System

Decomposition by Domain/Subdomain

- Core:
  - Cannot be bought off the shelf or outsourced
  - Provide value to the business

- Supporting:
  - Integral in delivering the core capabilities
  - It is not different from other competitors
- Generic:
  - Not specific to a particular business
  - Can be bought or used off the shelve

Decomposition by Domain/Subdomain

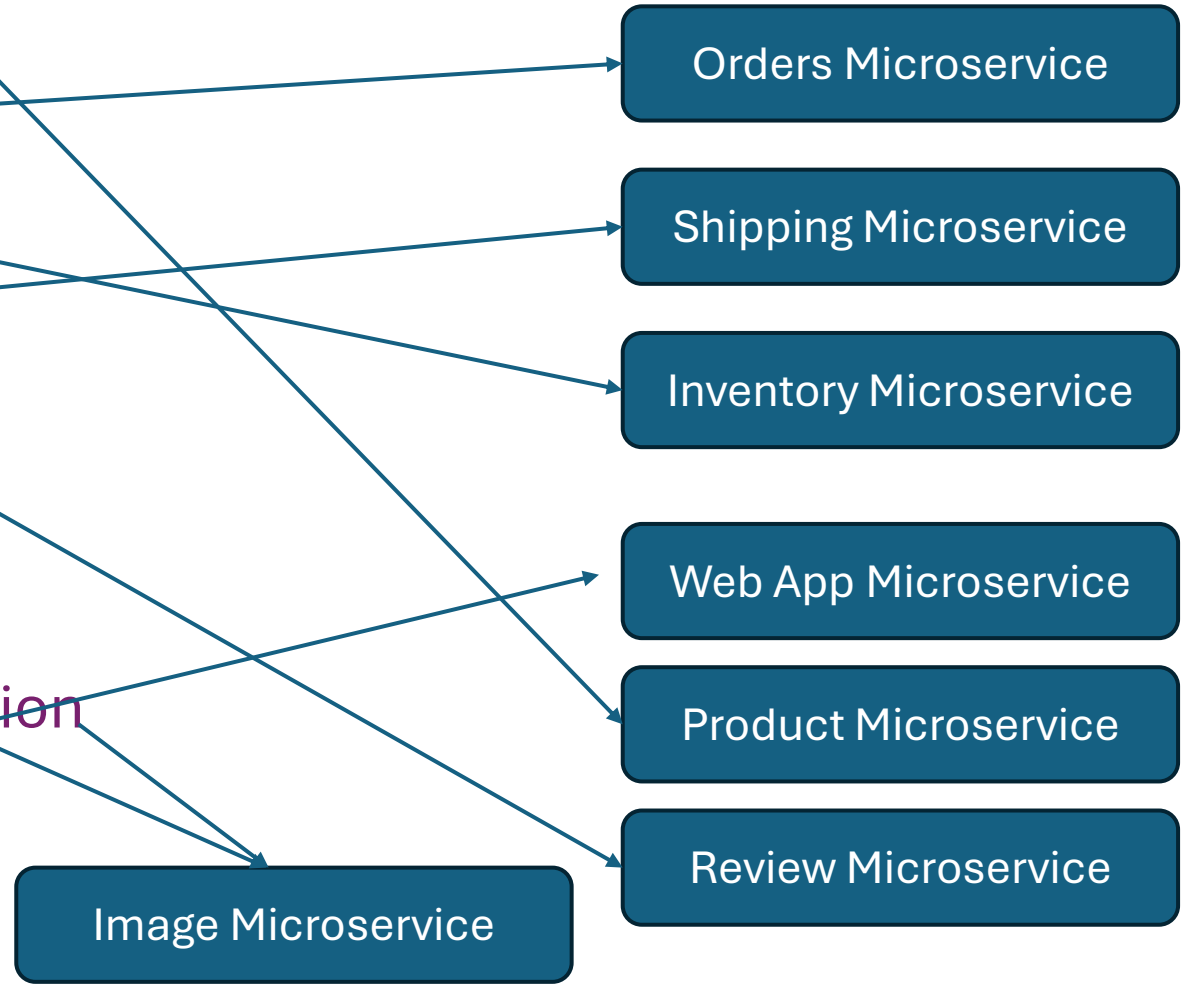
- Subdomain categorization helps
  - Prioritize the investment in each subdomain
  - Allocate engineering by experience
  - Save costs and time

Example: Online Store

- Core subdomain
  - Product catalog
- Supporting subdomain
  - Orders
  - Inventory
  - shipping
- Generic subdomain
  - Reviews
  - Payment
  - Search
  - Images
  - Image Compression
  - Security
  - Web UI

Subdomain → Microservices

- Product catalog
- Orders
- Inventory
- shipping
- Reviews
- Payment
- Search
- Images
- Image Compression
- Security
- Web UI



Other Methods

- Decomposition by Action
- Decomposition by Entities

Topics

- Decomposition by Business Capabilities
- Decomposition by Domain/Subdomain
- Decomposition by business Capabilities vs Subdomains

Comparison

	Business Capabilities	Subdomain
Cohesion and loose coupling	X	
Size of microservices		X
Stability of the design	X	
Intuitive for engineers		X

Final Note

- No one right way for the decomposition of microservices
- No perfect decomposition
- No techniques are bulletproof

Summary

- Techniques for Decomposition to Microservices:
  - By Business Capabilities
  - By Subdomain
    - Core
    - Supporting
    - Generic

Migration to Microservices

Steps, Tips, and Patterns

Topics

- Where To Start The Migration To Microservices
- Preparing for the migration
- Executing the migration using the Strangler Fig pattern
- Tip to Ensure Smooth Migration

Topics

- Where to Start the Migration to Microservices
- Preparing for the migration
- Executing the migration using the Strangler Fig pattern
- Tip to Ensure Smooth Migration

Big Bang Approach

- The plan
  - Map out the microservices boundaries
  - Stop any development of new features

Big Bang Approach-Problems

- “Too Many Cooks In The Kitchen”
- Hard To Estimate The Effort For Large And Ambiguous Projects
- High Risk Of Abandonment
- Stopping Development Is Detrimental To The Business

Incremental and Continuous Approach

- Identify the components that can benefit the most from the migration
- Best candidates:
  - Areas with the most development/frequent changes.
  - Components with high scalability requirements.
  - Components with the least technical debt.

Incremental and Continuous Approach- Benefits

- No hard deadlines necessary
- Consistent, visible, and measurable progress
- Business is not disrupted
- Exceeding the time estimates is not a problem

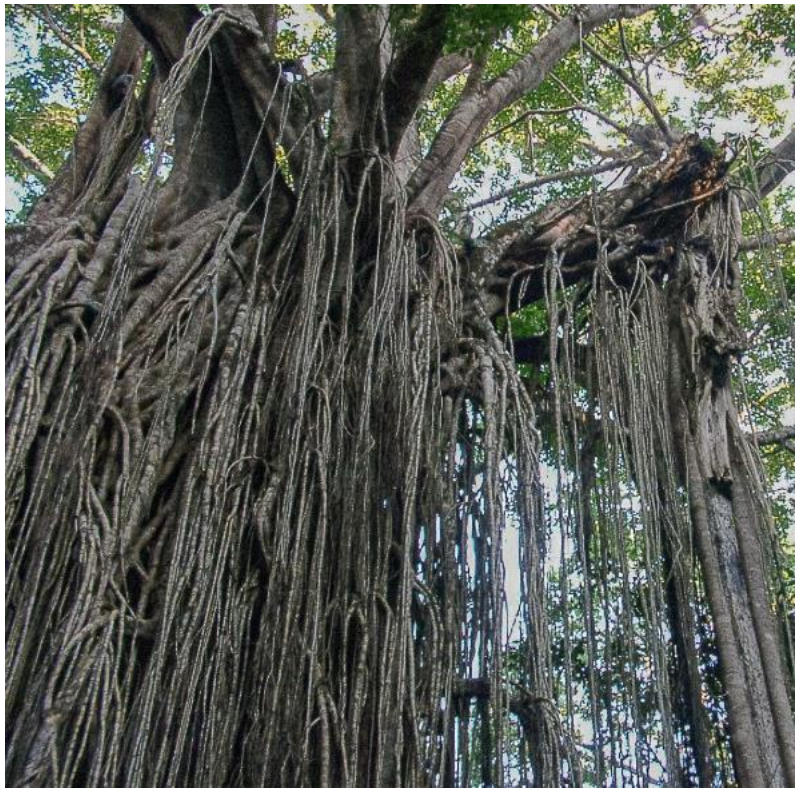
Steps to Prepare for Migration

- Add/ensure code test coverage
- Define component API
- Isolate the component by removing interdependencies to rest of the application

Topics

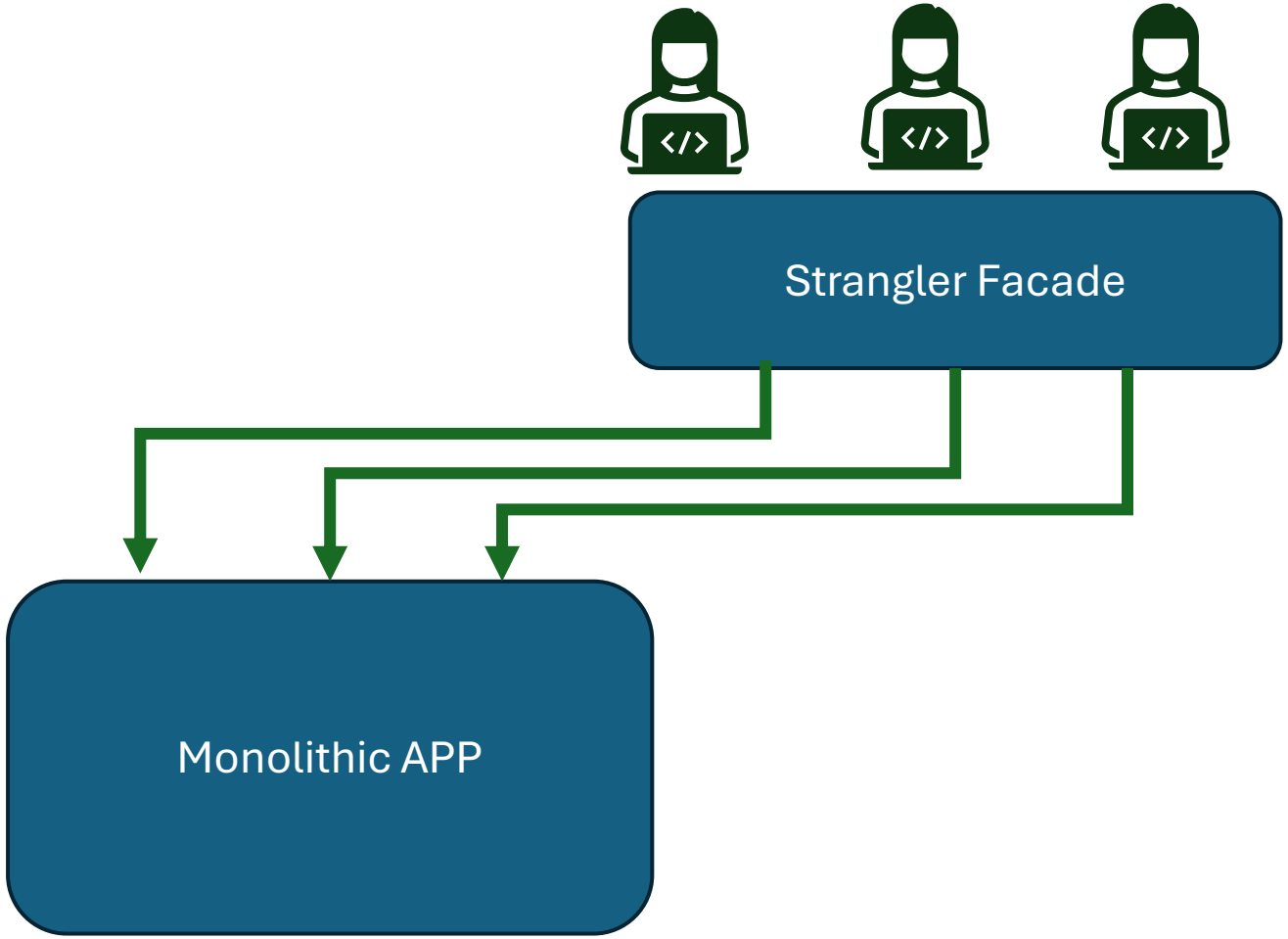
- Where To Start The Migration To Microservices
- Preparing for the migration
- Executing the migration using the Strangler Fig pattern
- Tip to Ensure Smooth Migration

Strangler Fig Pattern

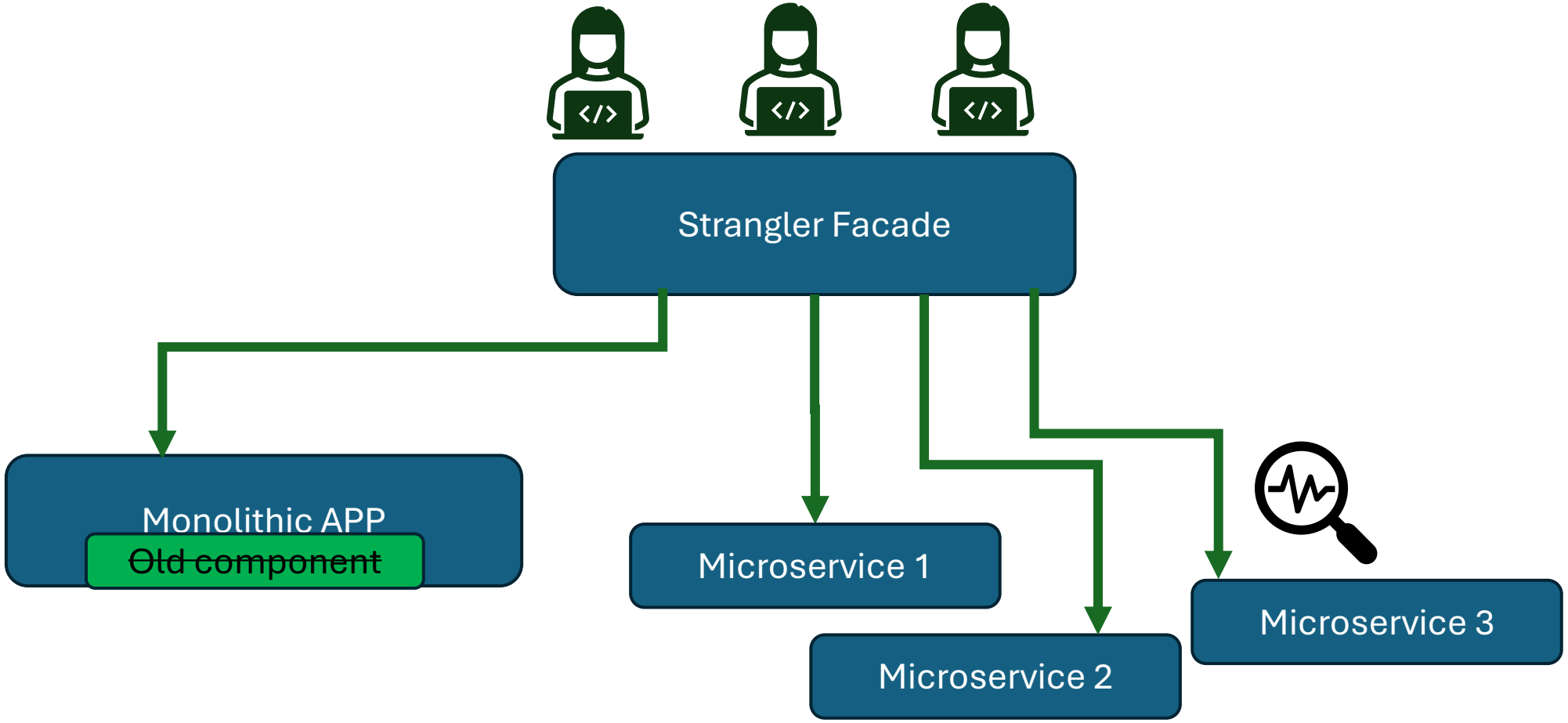


https://martinfowler.com/bliki/StranglerFigApplication.html

Strangler Fig Pattern



Strangler Fig Pattern



Topics

- Where To Start The Migration To Microservices
- Preparing for the migration
- Executing the migration using the Strangler Fig pattern
- Tip to Ensure Smooth Migration

Tip to Ensure Smooth Migration

- Keep the code and technology stack unchanged
- Risk Mitigation

Summary

- Prepare and execute the migration to Microservice Architecture
- Best candidates:
  - Components that change frequently
  - Components that require higher scalability
  - Components that have the least technical debt
- Migration process:
  - add/ ensure “test coverage”
  - Define the API
  - Isolate the component
  - Use the strangler Fig Pattern