Databases in Microservices Architecture

Database Per Microservice

Topics

- Motivation for Database Per Microservice
- Benefits of Database Per Microservice Principle
- Downside of Database Per Microservice

Topics

- Motivation for Database Per Microservice
- Benefits of Database Per Microservice Principle
- Downside of Database Per Microservice

User

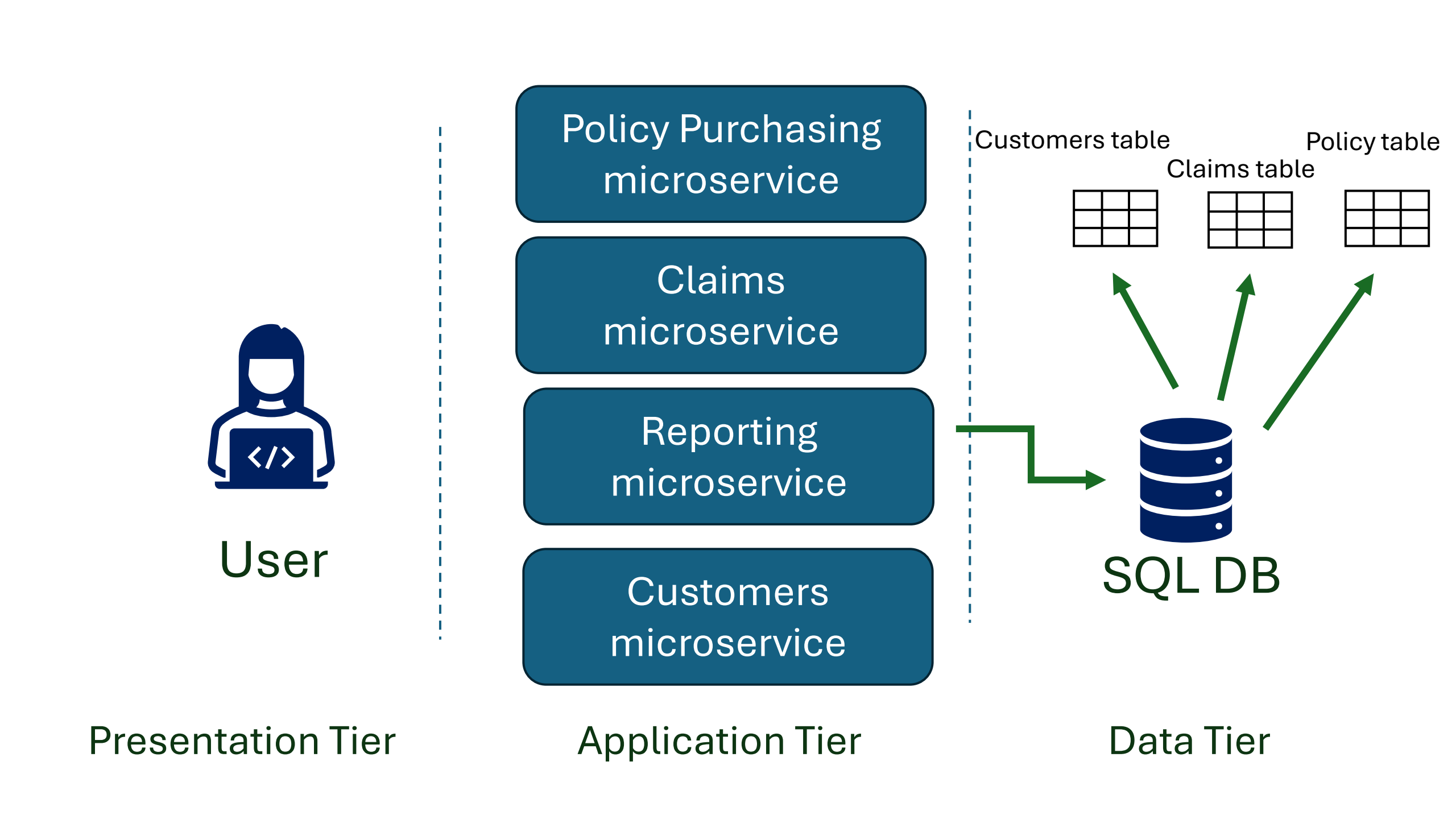
Presentation Tier

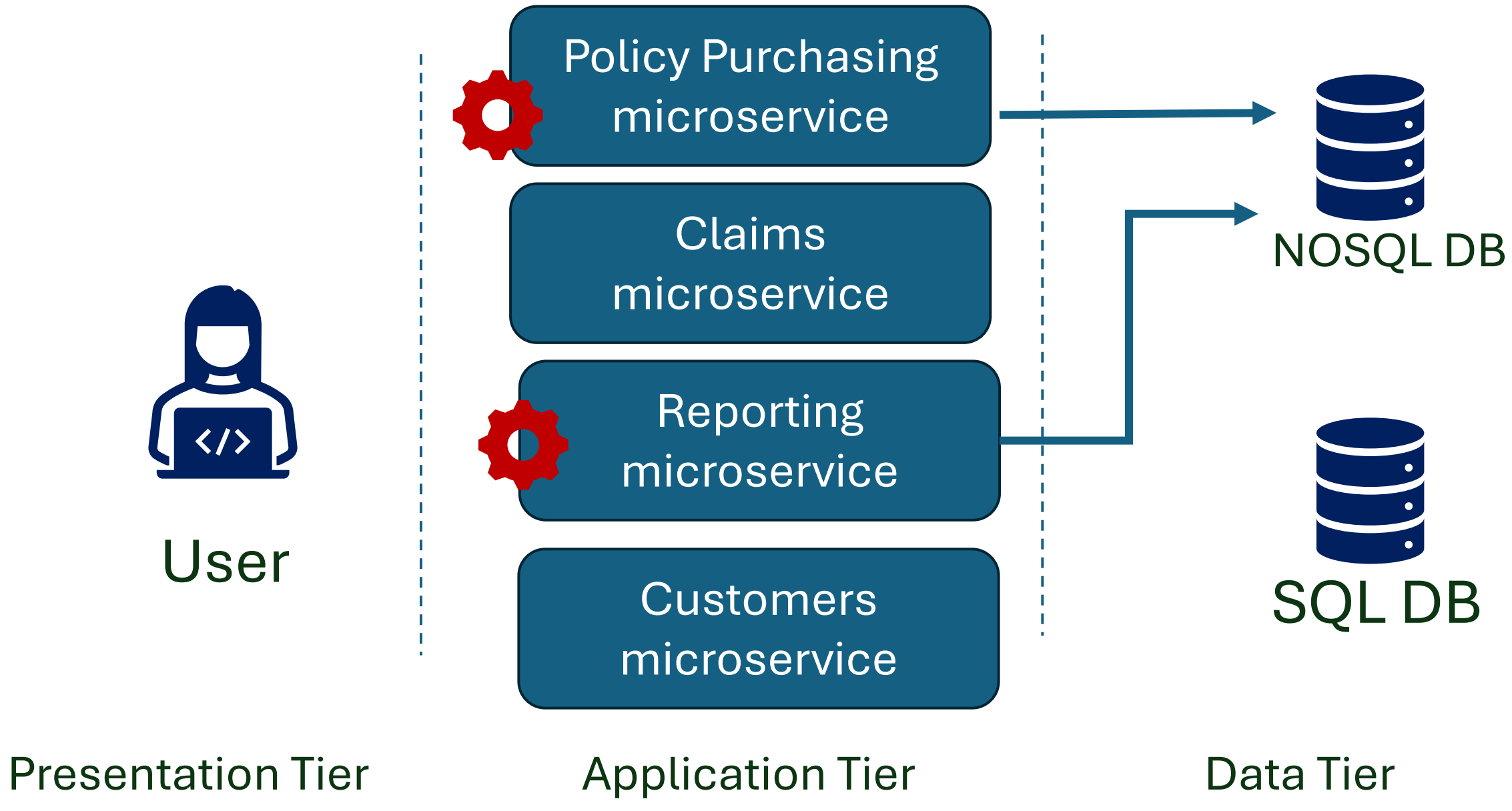
Insurance Web Application

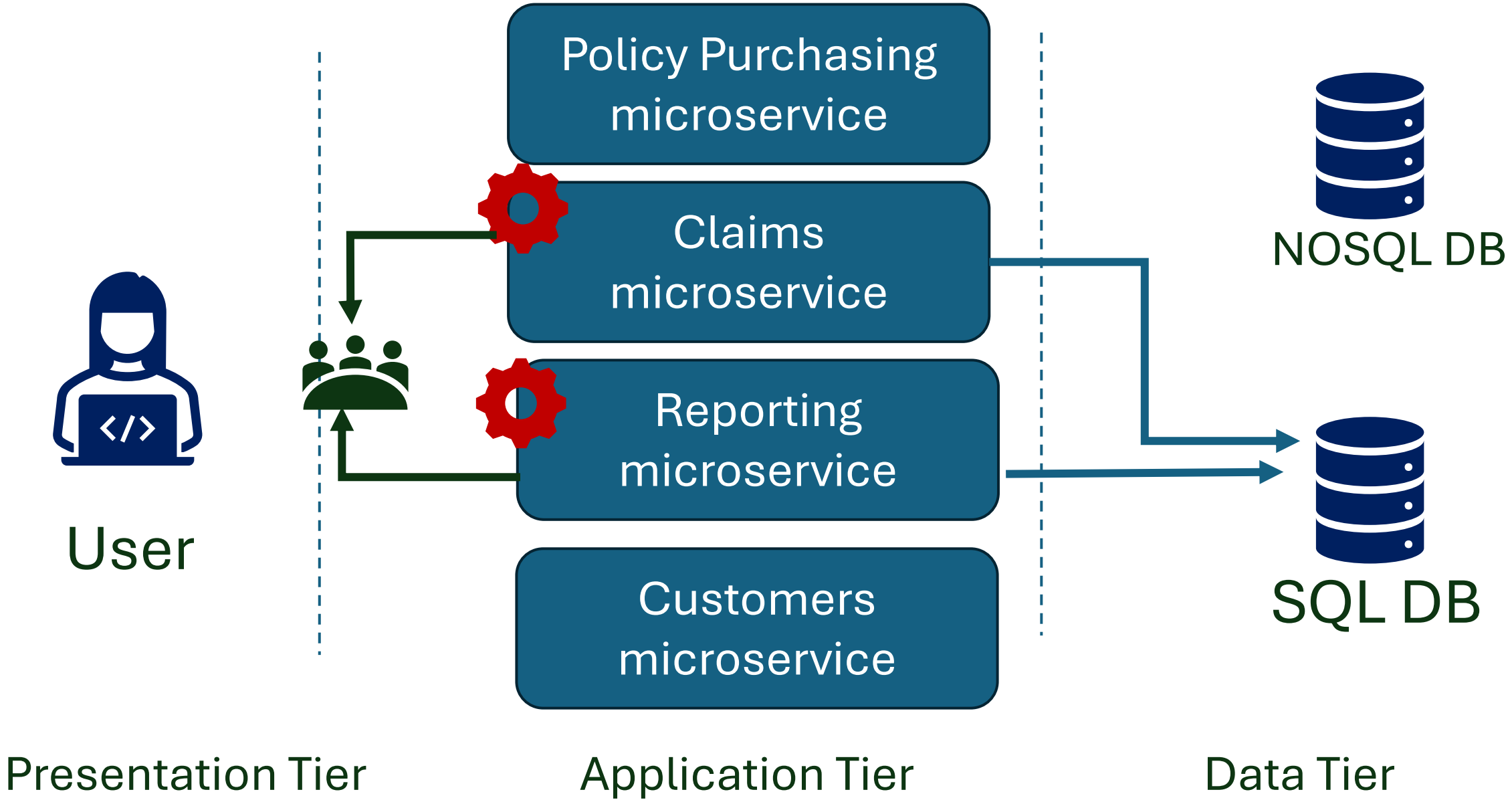
Application Tier

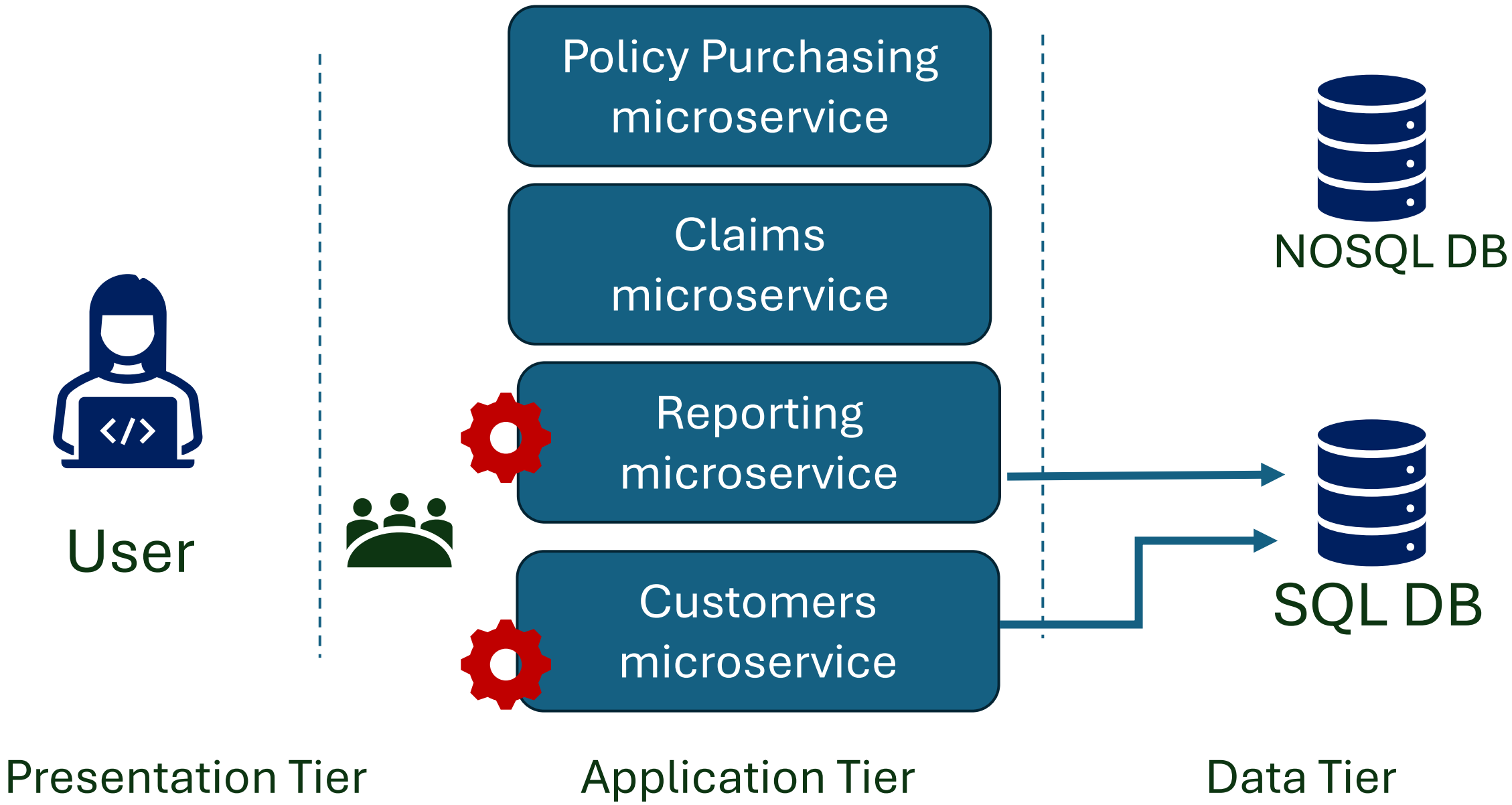
SQL DB

Data Tier


Policy Purchasing microservice
Claims microservice
Reporting microservice
Customers microservice
Customers table
Claims table
Policy table
SQL DB
User
Presentation Tier
Application Tier
Data Tier

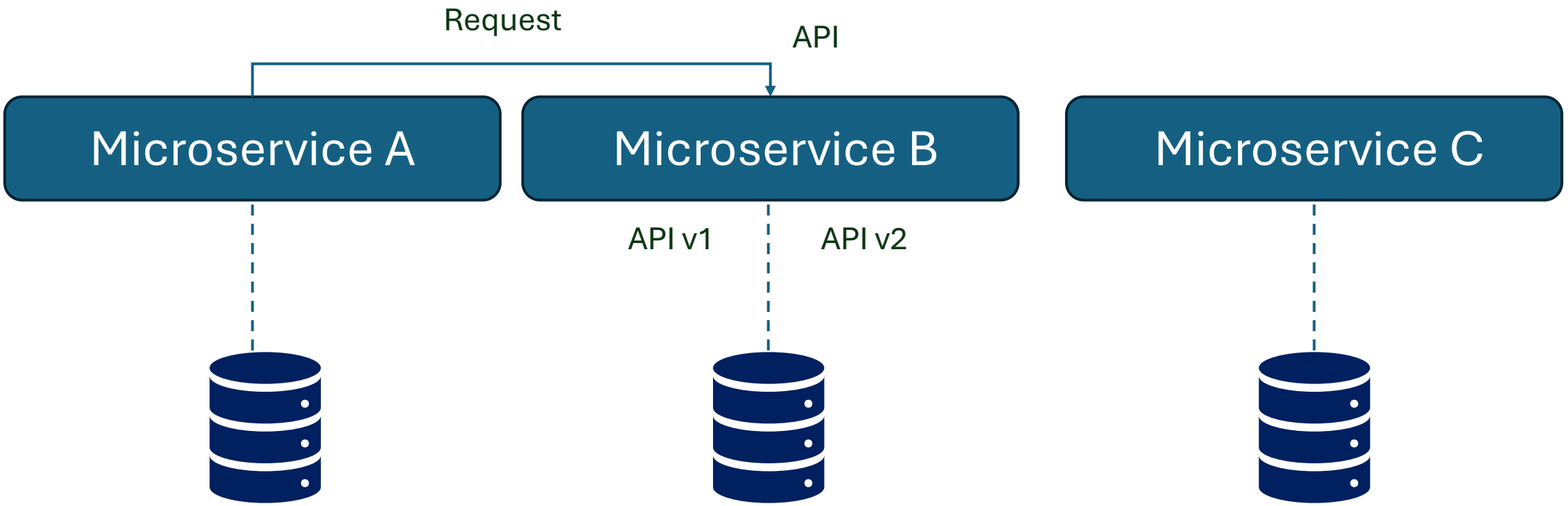

Policy Purchasing microservice
Claims microservice
Reporting microservice
Customers microservice
NOSQL DB
SQL DB
User
Presentation Tier
Application Tier
Data Tier


Policy Purchasing microservice
Claims microservice
Reporting microservice
Customers microservice
NOSQL DB
SQL DB
User
Presentation Tier
Application Tier
Data Tier


Policy Purchasing microservice
Claims microservice
Reporting microservice
Customers microservice
NOSQL DB
SQL DB
User
Presentation Tier
Application Tier
Data Tier

Topics

- Motivation for Database Per Microservice
- Benefits of Database Per Microservice Principle
- Downside of Database Per Microservice

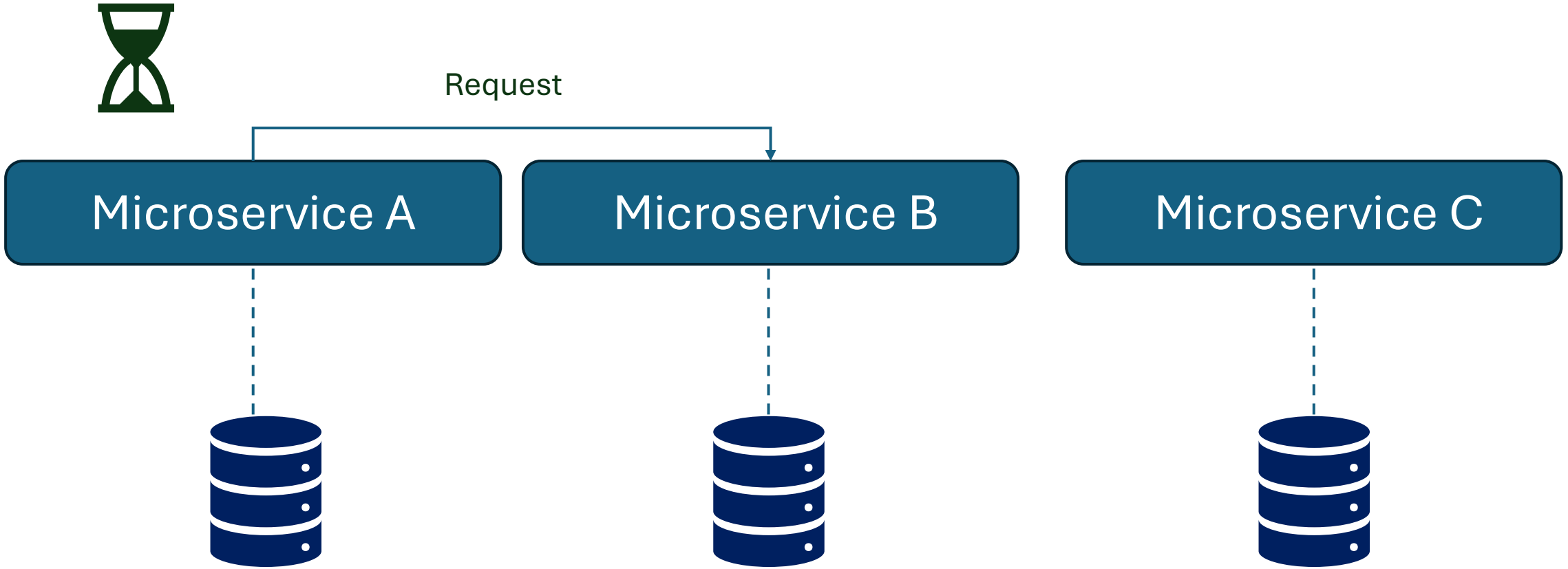

Request
API
Microservice A
Microservice B
Microservice C
API v1
API v2

Topics

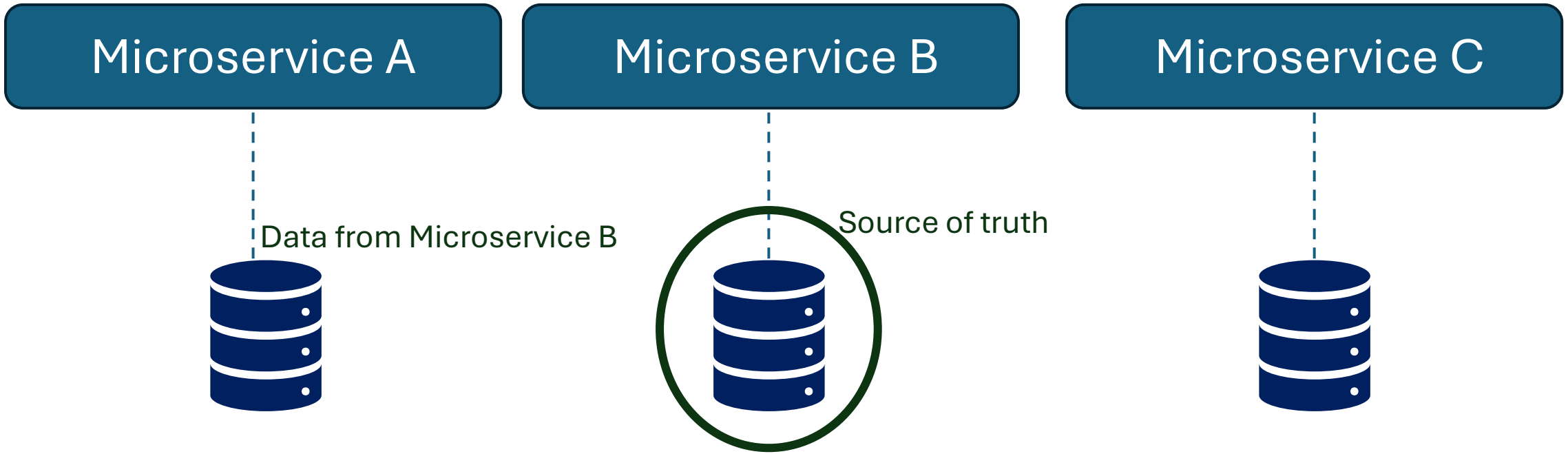
- Motivation for Database Per Microservice
- Benefits of Database Per Microservice Principle
- Downside of Database Per Microservice

Downsides of Database Per Microservice

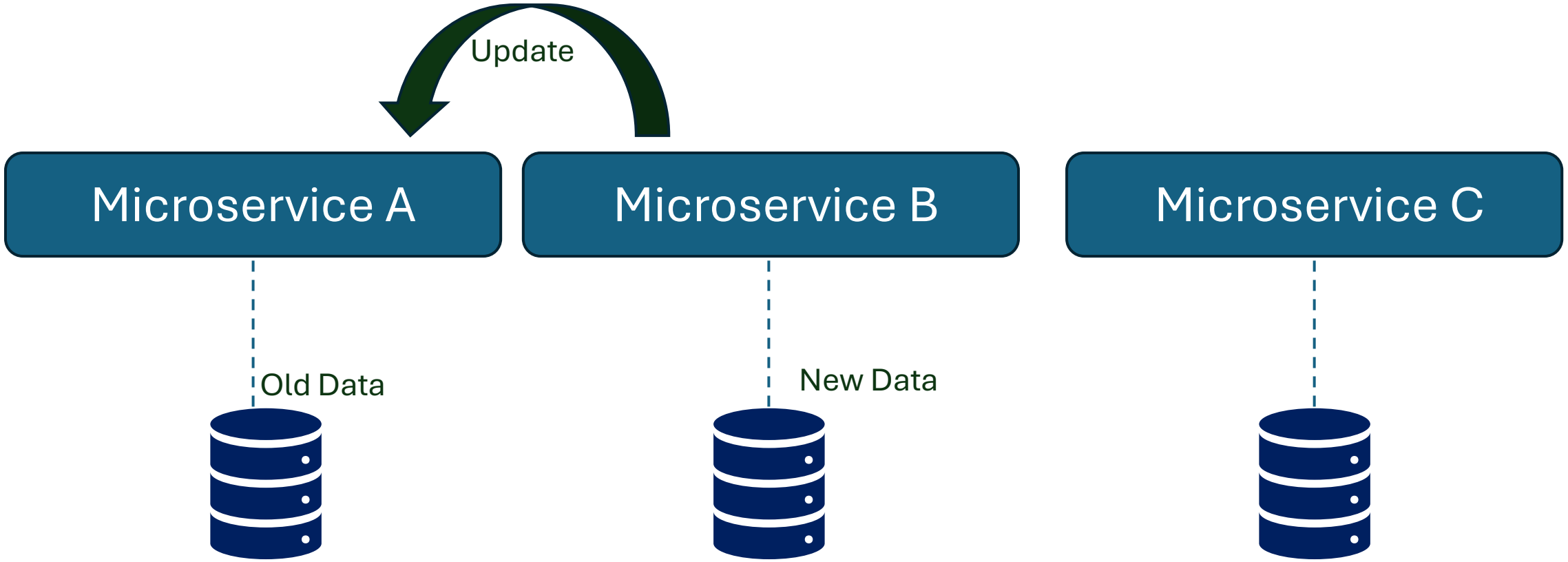
1. Added latency



Data Caching

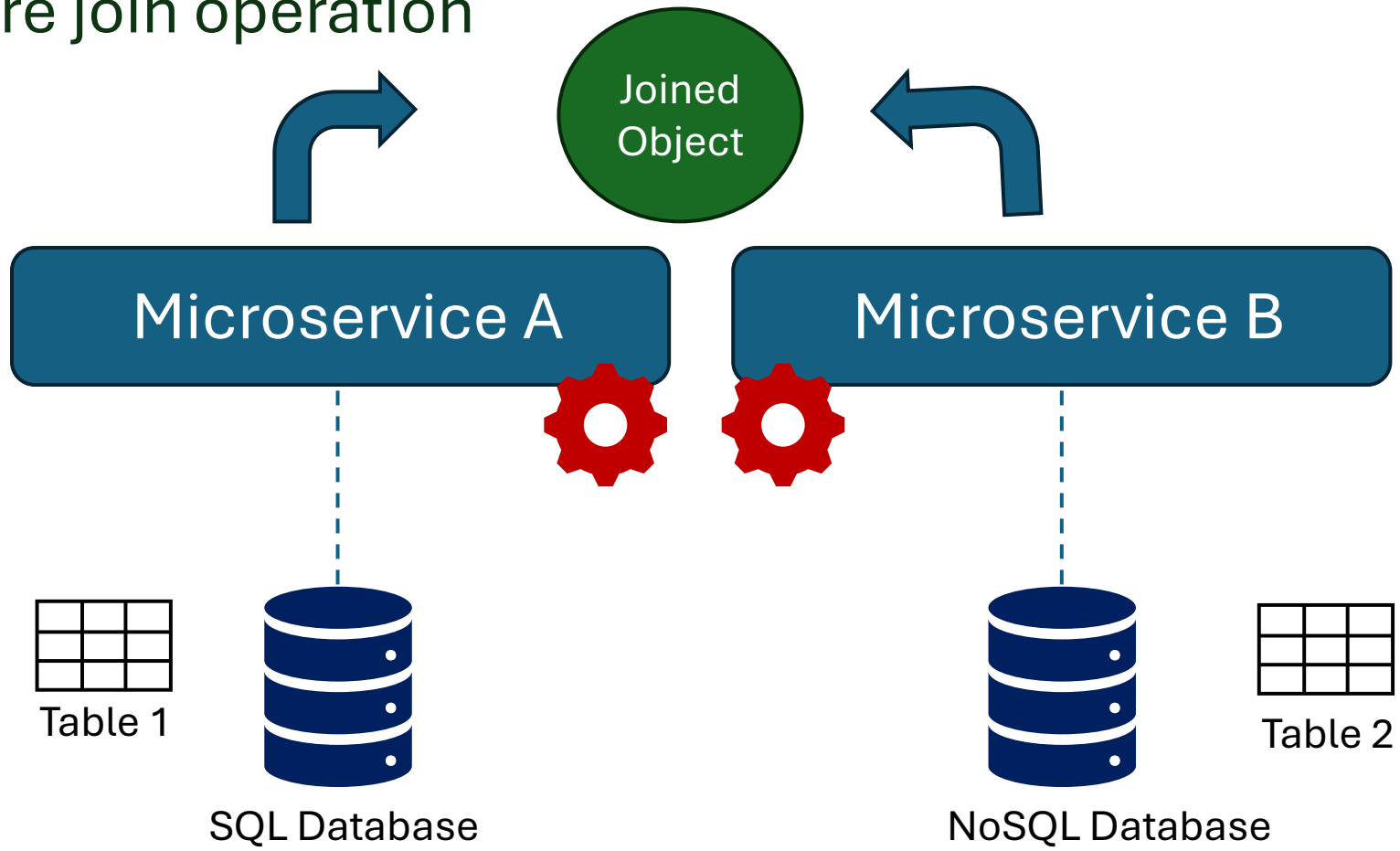


Eventual Consistency

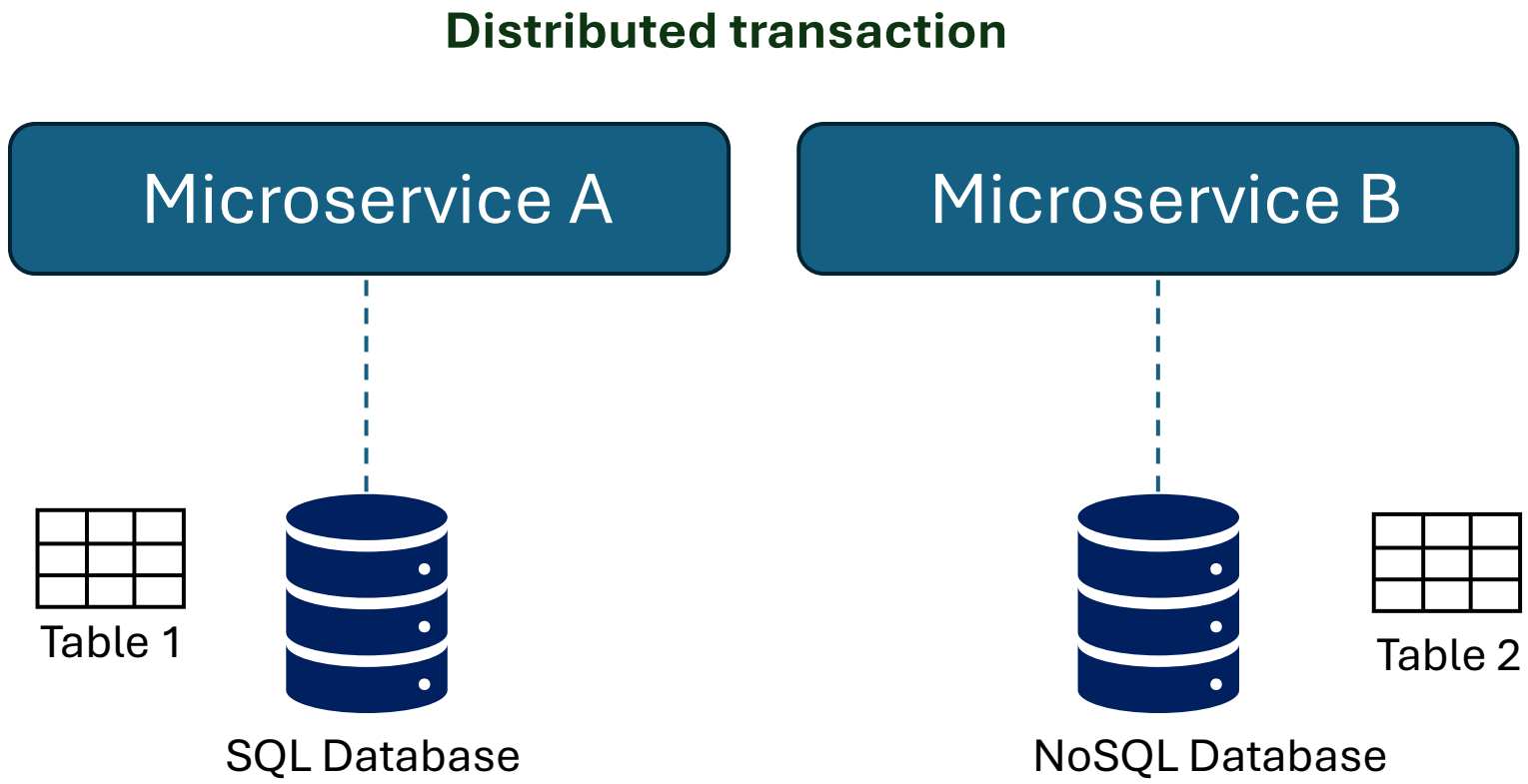


Downsides of Database Per Microservice

2. No more join operation



Downsides of Database Per Microservice



Summary

- Database Per Microservice Principle
  - Motivation
    - Sharing a database tightly couples
      - Codebases
      - Teams
- Drawbacks of Database Per Microservice:
  - Worse performance
  - Complex join operations
  - No transactions

Microservices

DRY – Don’t Repeat Yourself

Topics

- Importance of DRY
- Challenges of Following The DRY And Shared Libraries
- Alternative to Shared Libraries in Microservice Architecture
- Sharing / Duplicating Data Across Microservices

Topics

- Importance of DRY
- Challenges of Following the DRY and Shared Libraries
- Alternative to Shared Libraries in Microservice Architecture
- Sharing / Duplicating Data Across Microservices

DRY – Don’t Repeat Yourself

- If you find yourself repeating the same logic or the same data, you should always consolidate it in a single place as a shared method
  - Class or
  - Variable
  - Shared library

Benefits of following DRY

- We can make a change in only one place
- Reduce duplicate effort
- Work of a single engineer can be reused

DRY doesn’t always apply to Microservices

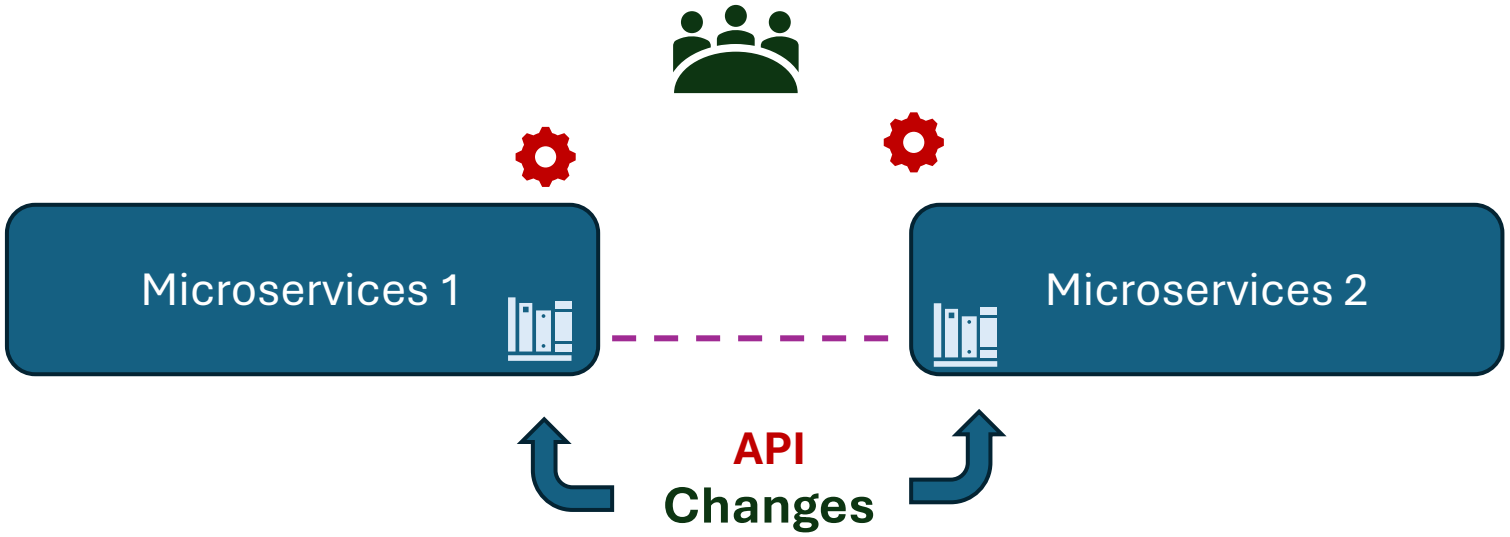
Topics

- Importance of DRY
- Challenges of Following The DRY and Shared Libraries
- Alternative to Shared Libraries in Microservice Architecture
- Sharing / Duplicating Data Across Microservices

Challenges of Shared Libraries

- Tight coupling

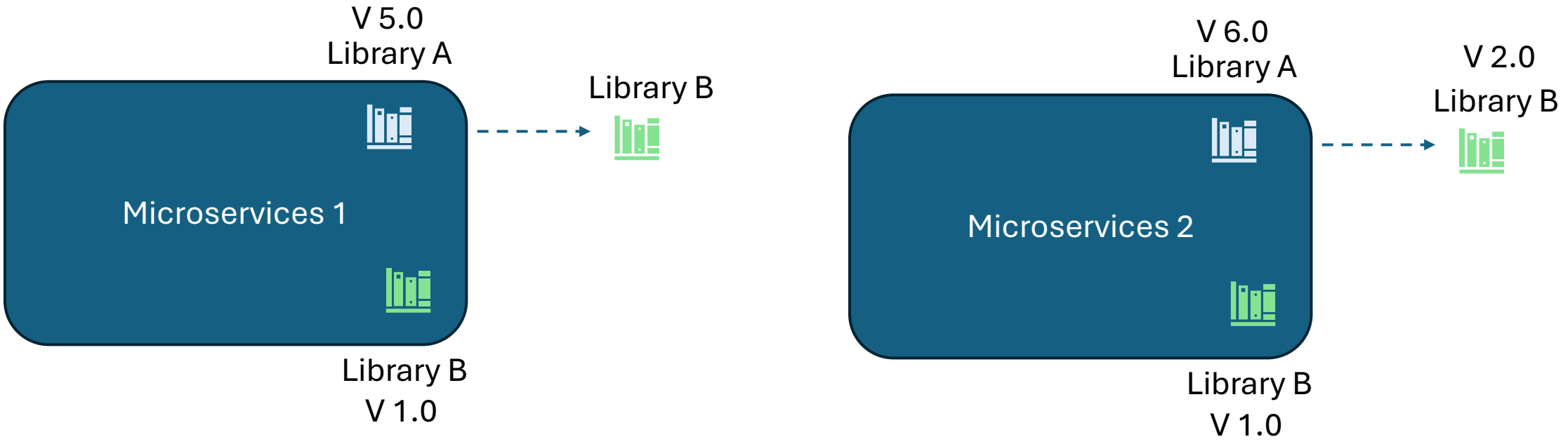
Shared Library - Challenges



Challenges of Shared Libraries

- Tight coupling
- Every change requires:
  - Rebuild
  - Retest
  - Redeploy
- Bug/Vulnerability in a shared library impacts all microservices
- Dependency Hell

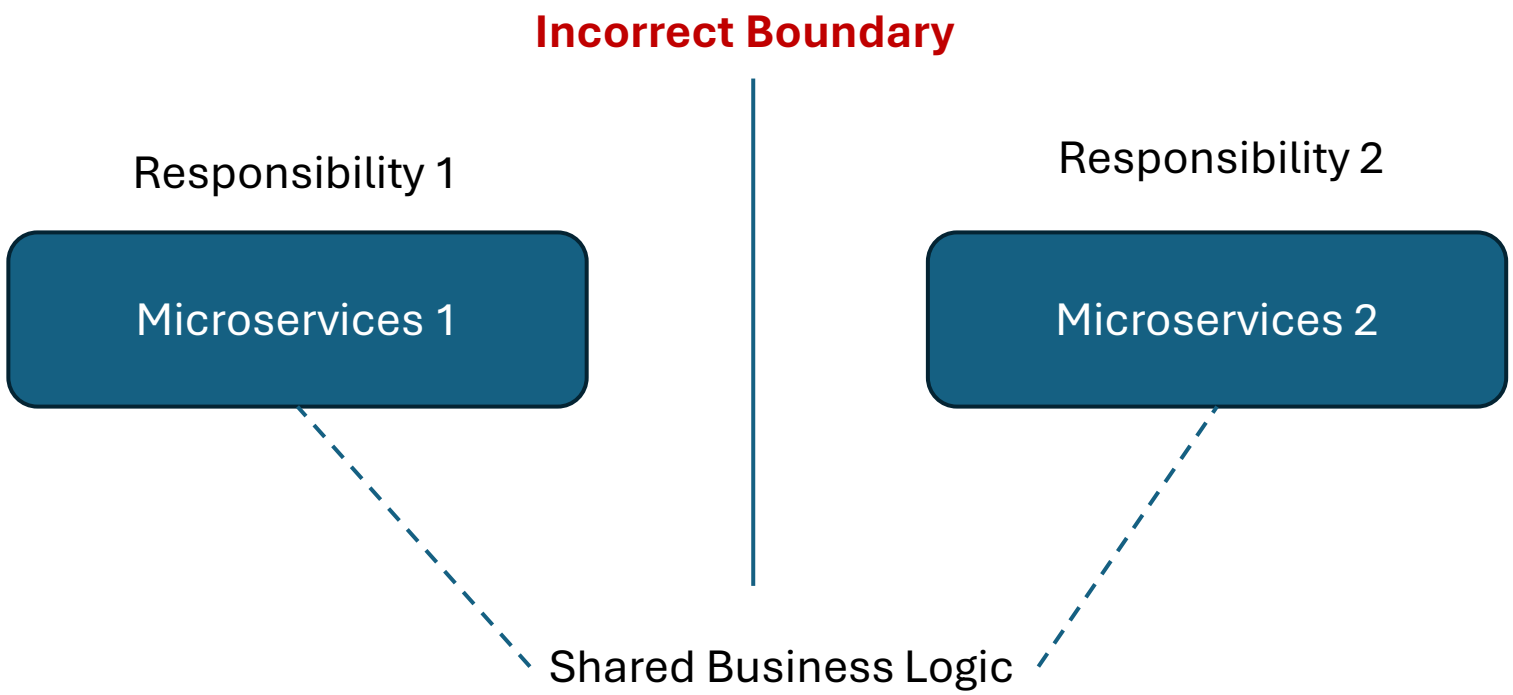
Shared Library – Dependency Hell



Topics

- Importance of DRY
- Challenges of Following The DRY And Shared Libraries
- Alternative to Shared Libraries in Microservice Architecture
- Sharing / Duplicating Data Across Microservices

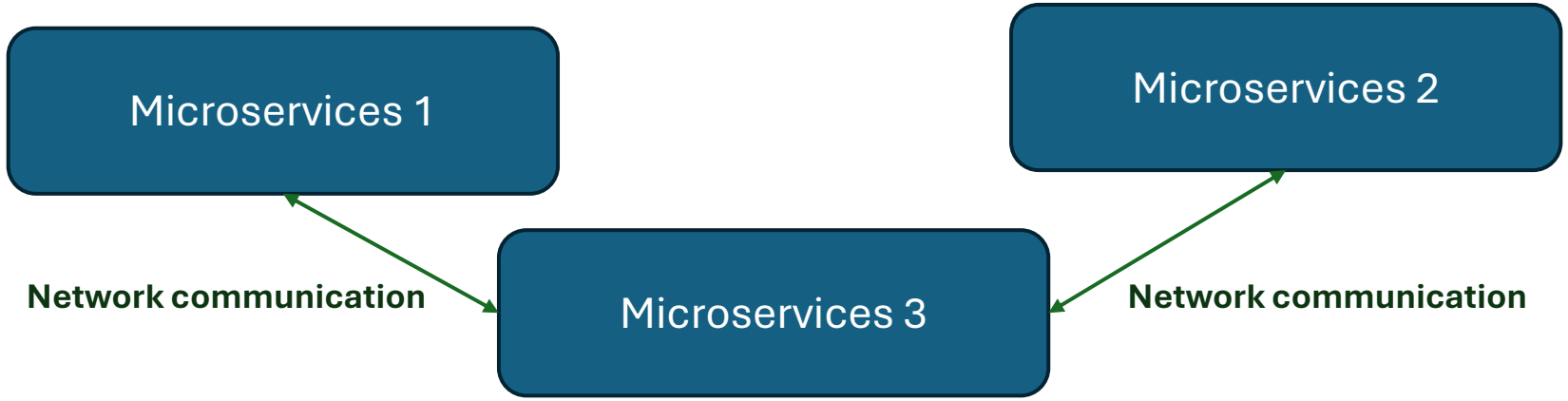
Situation 1 : Shared Business Logic



Solutions for Shared Business Logic

Option 1: Re-evaluate the Boundaries

Option 2: Re-evaluate the Boundaries

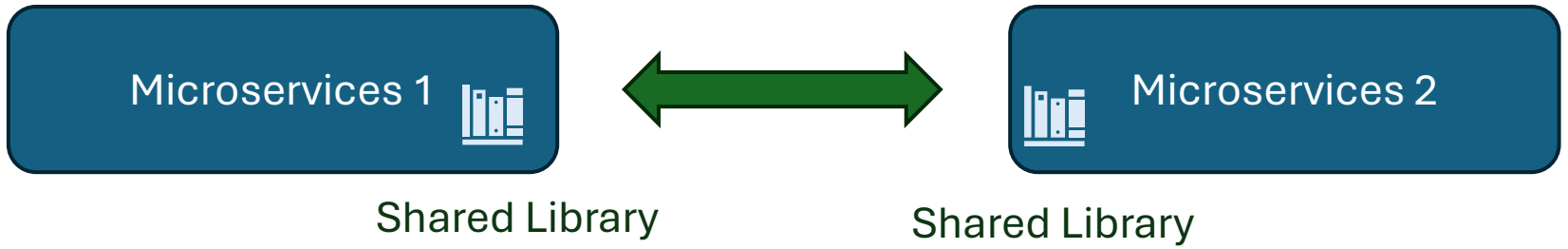


Situation 2: Common Data Model for Communication

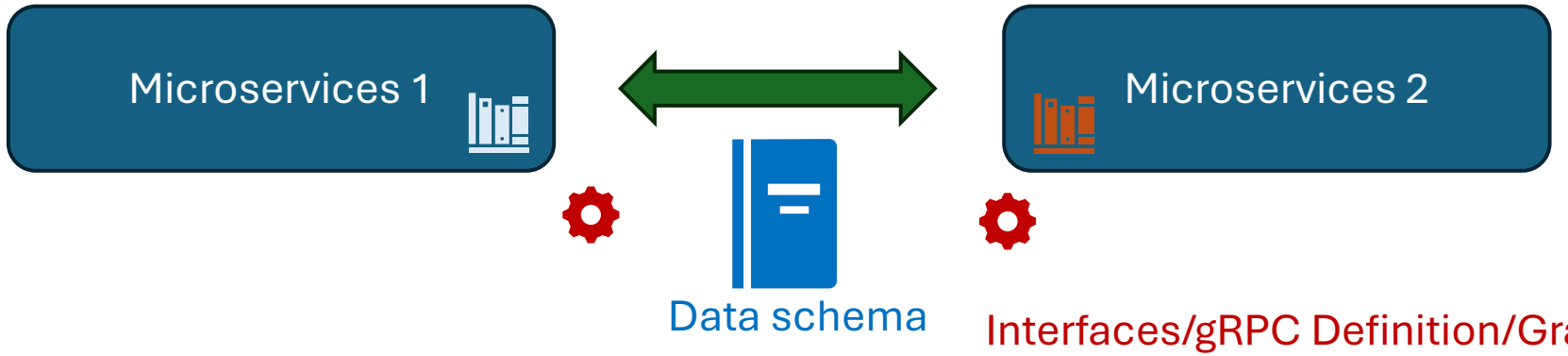
User
Account
Payment Result

Solution for the Common Data Model for Communication

Option 1: Shared Library



Option 2:Code Generation

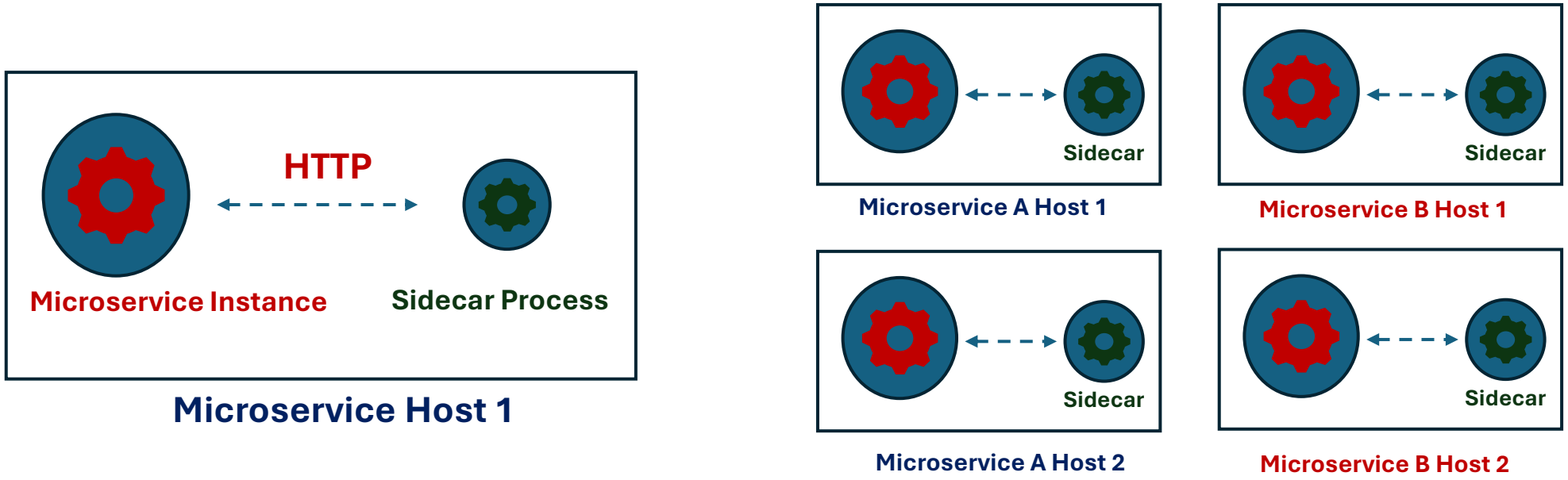


Code that is OK to Duplicate

- Utility methods that change frequently
- The reason:
  - Each microservice can have its optimized implementation
  - Make it easier to migrate microservices to other programming languages

No Code Duplication Options

Option 1: Side Pattern

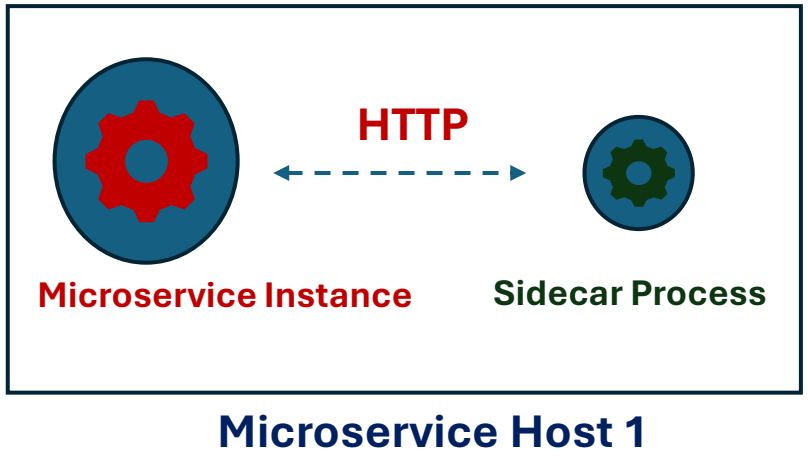


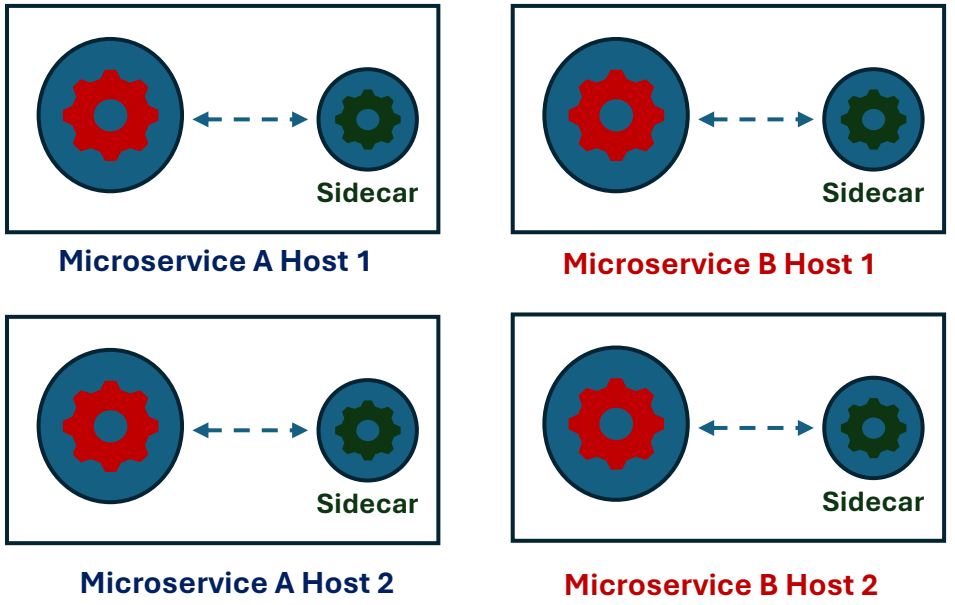
Microservice to sidecar Communication Overhead

- Smaller than with other services/hosts
- Higher than with code in a shared library

No Code Duplication Options

Option 1: Side Pattern





Option 2: Use a Shared Library

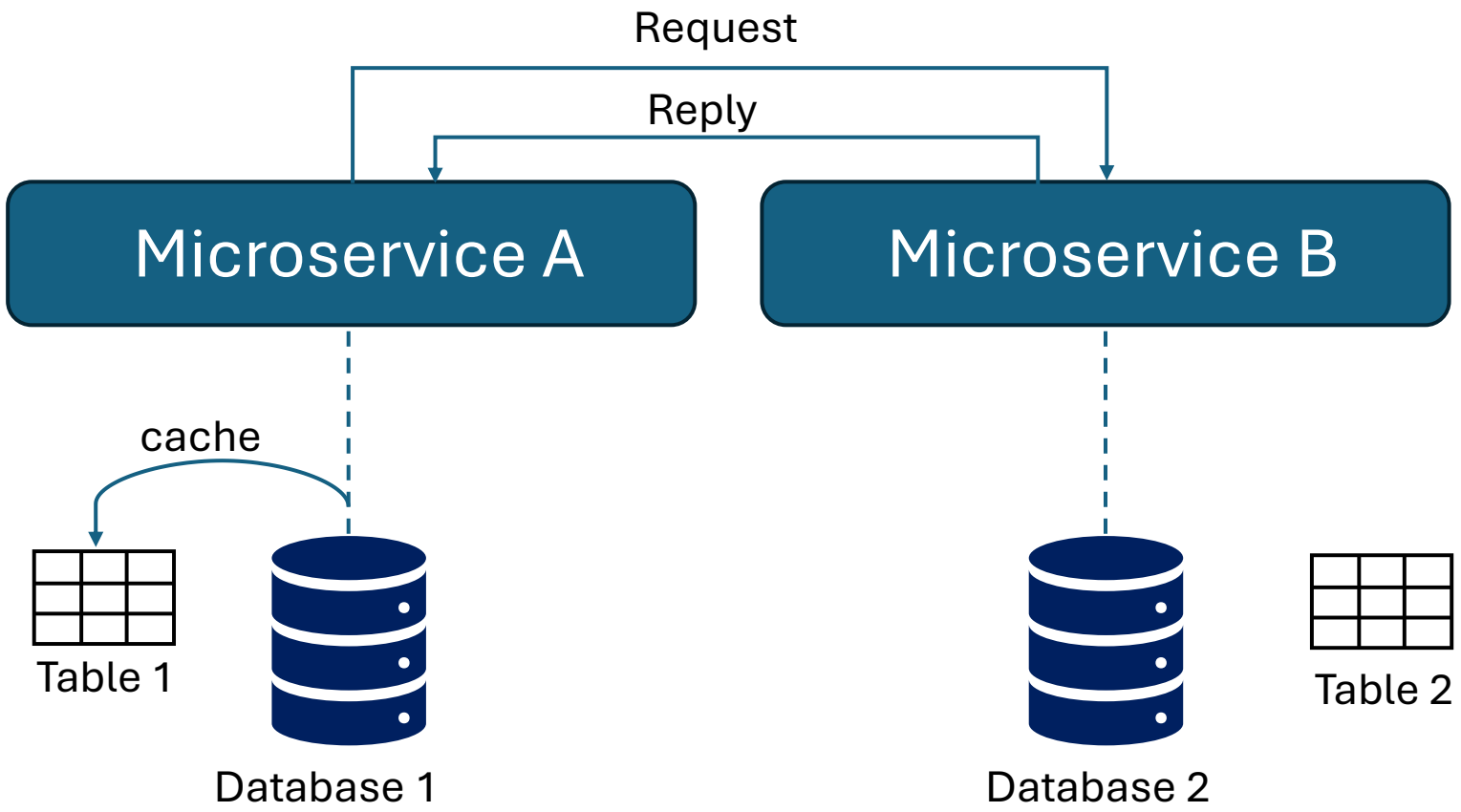
Final note on DRY in Microservices

- Inside each microservice, we still follow DRY
- Code duplication is unacceptable

Topics

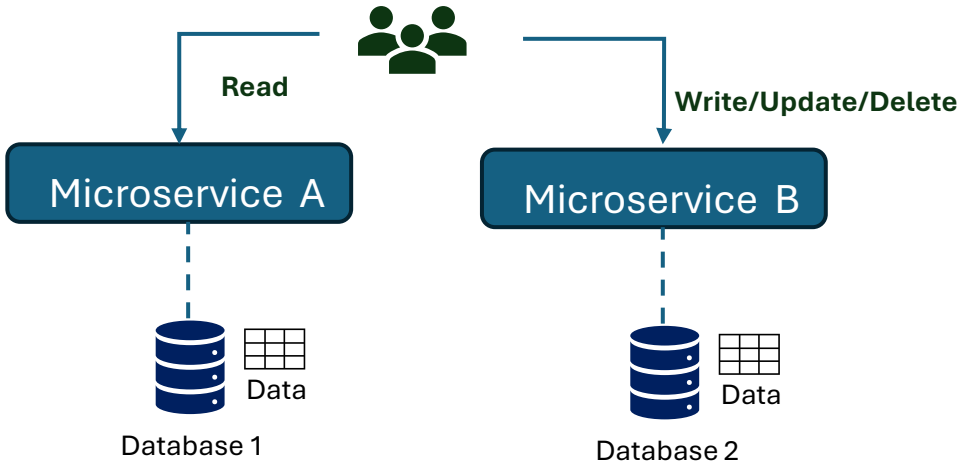
- Importance of DRY
- Challenges of Following The DRY And Shared Libraries
- Alternative to Shared Libraries in Microservice Architecture
- Sharing / Duplicating Data Across Microservices

Data Duplication in Microservices



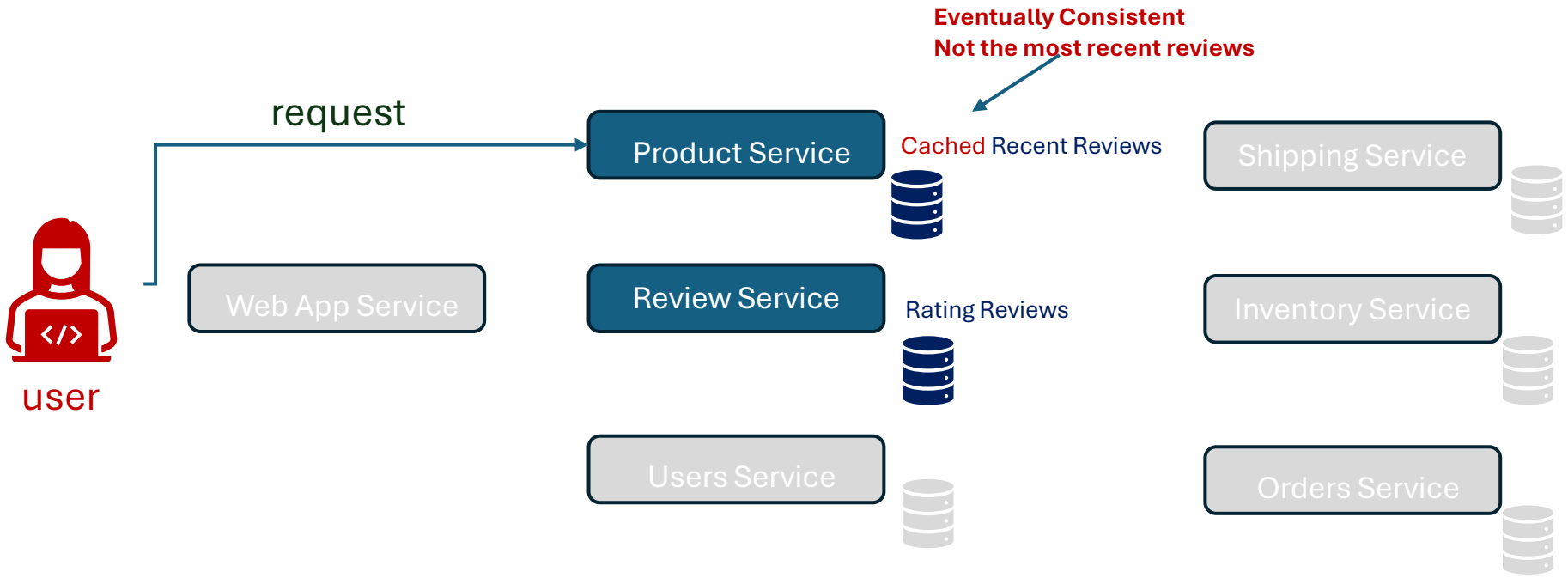
Note on Duplicating Data

1. Only one owner/ source of truth

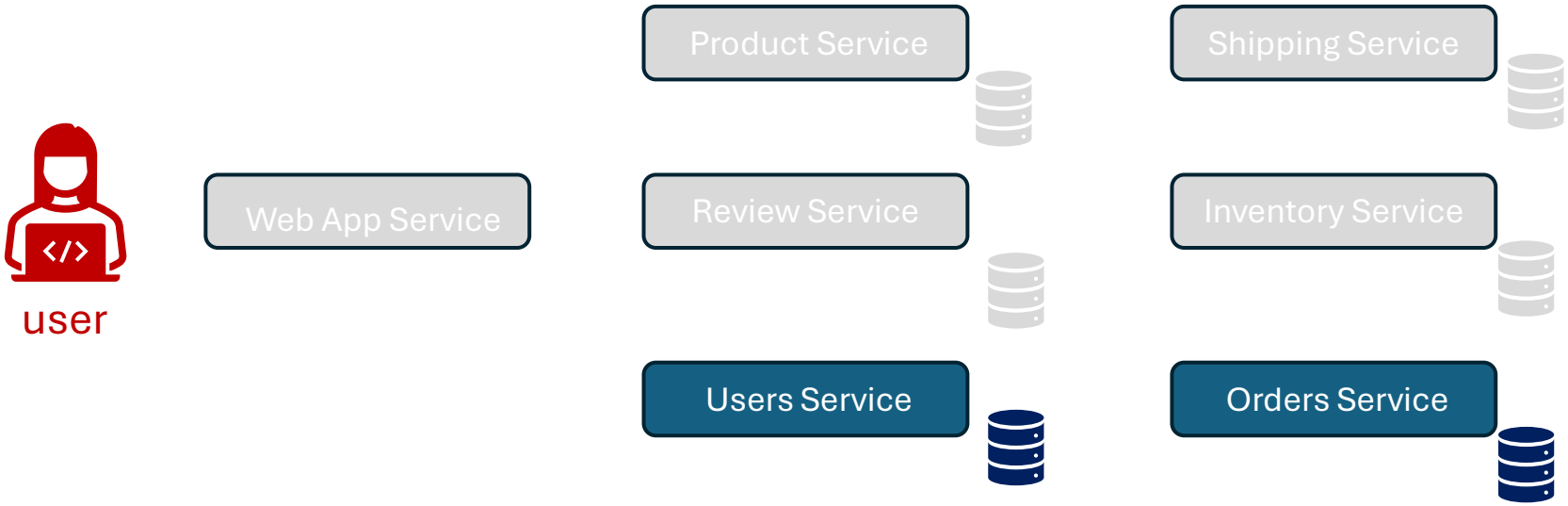


2. We can only guarantee eventual consistency

Data Consistency



Data Consistency



Summary

- Revisited the dry principle for sharing libraries and data in microservices architecture.
- **Benefits** of DRY in general, but not always in Microservices
- **Challenges** of a shared library in Microservices
  - Tight coupling
  - Rebuild, Retest, Redeploy
- Alternative to shared library:
  - New Microservices
  - Sidecar Pattern
  - Code generation
  - Code duplication
- Data duplication in Microservice
  - Important for performance reasons
  - Makes data eventually consistent
  - Only one microservice needs to remain the owner of each data

Structured Autonomy for Development Teams

Microservices Architectures Best Practices

Topics

- Problem with full team autonomy
- 3 tiers of developers’ team autonomy
- Factors o team autonomy boundaries

Topics

- Problem with full team autonomy
- 3 tiers of developers' team autonomy
- Factors of team autonomy boundaries

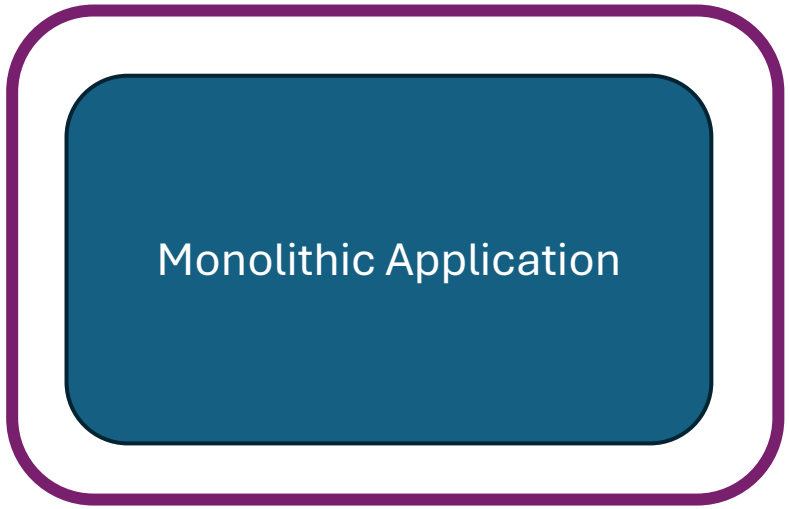
Myth

- Each team can choose its own:
  - Technology stack
  - Tools
  - Databases
  - API
  - Framework

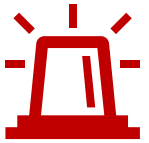
Full Team Autonomy Problem

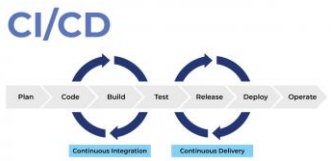
- Upfront cost of infrastructure

- **Tests:**
  - **Unit Tests**
    - Test1
    - Test2
  - **Functional Tests**
    - Test 1
    - Test 2
  - **Integration Tests**
    - Test 1
    - Test2

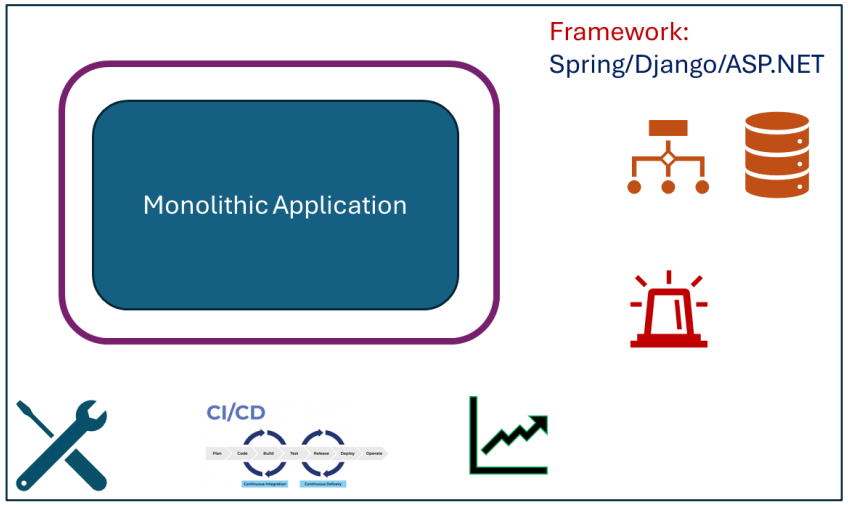


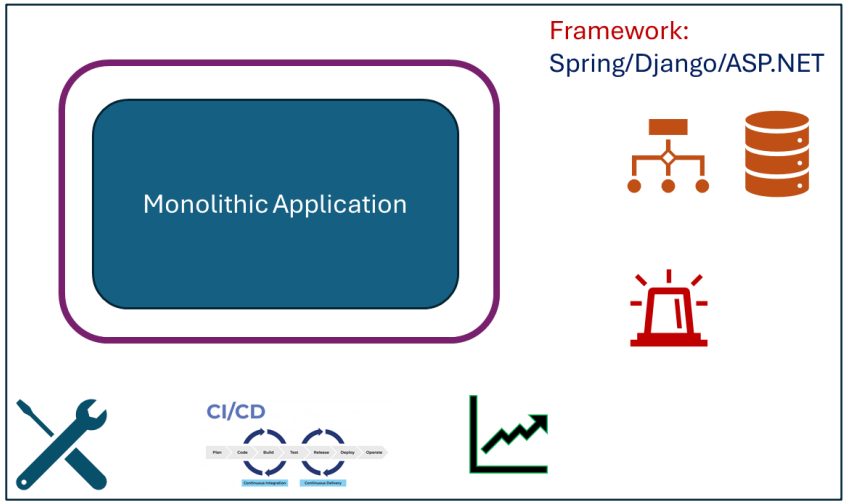
Framework:
Spring/Django/ASP.NET

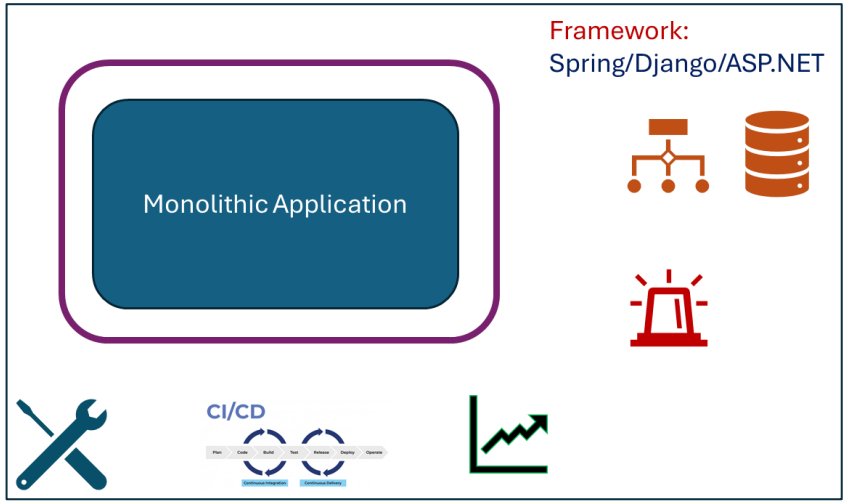


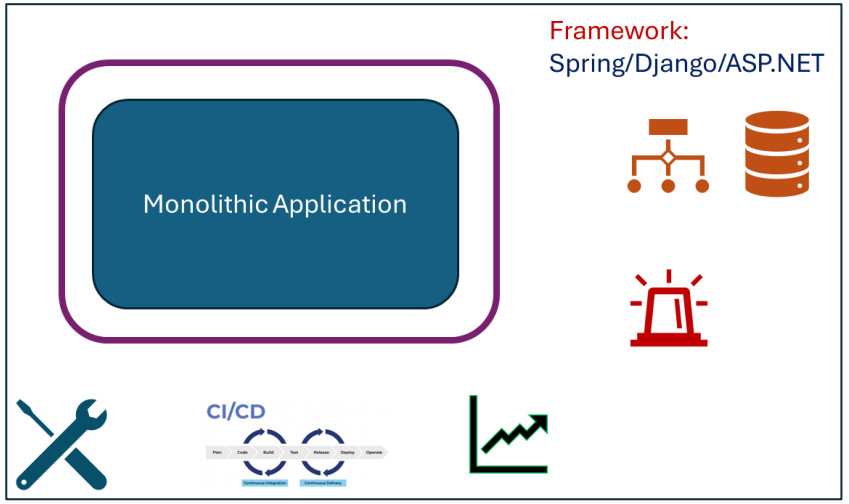


1. Upfront Cost of Infrastructure







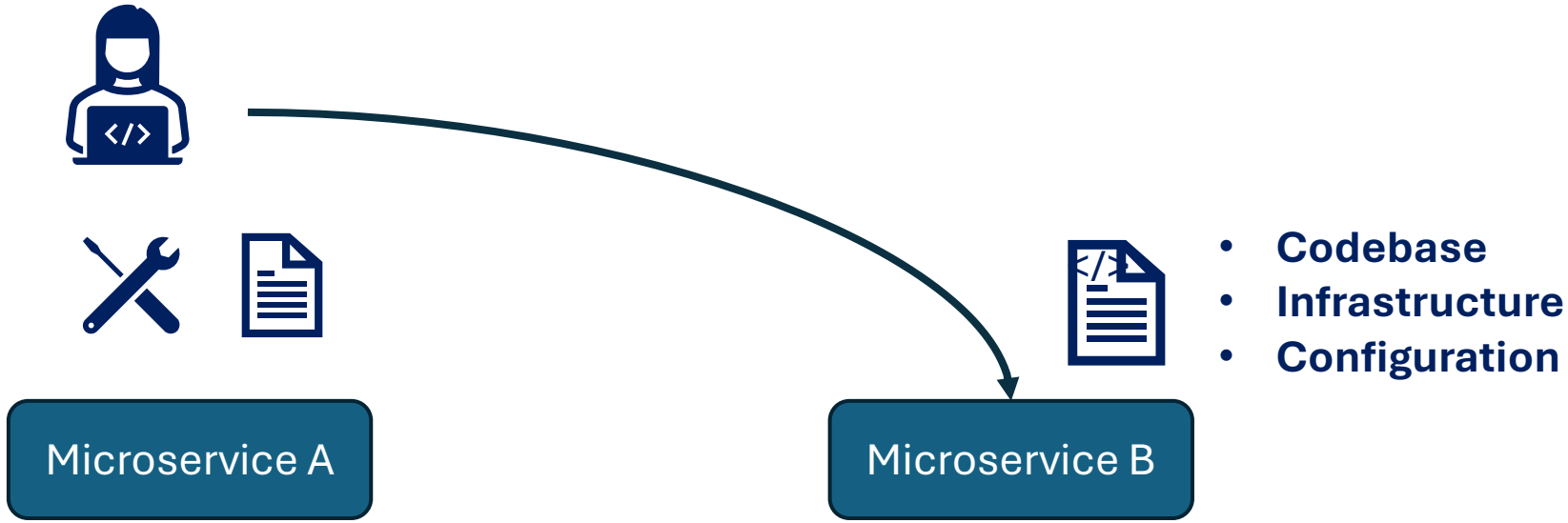


Full Team Autonomy Problem

1. Upfront Cost of Infrastructure
2. Infrastructure maintenance cost

Full Team Autonomy Problem

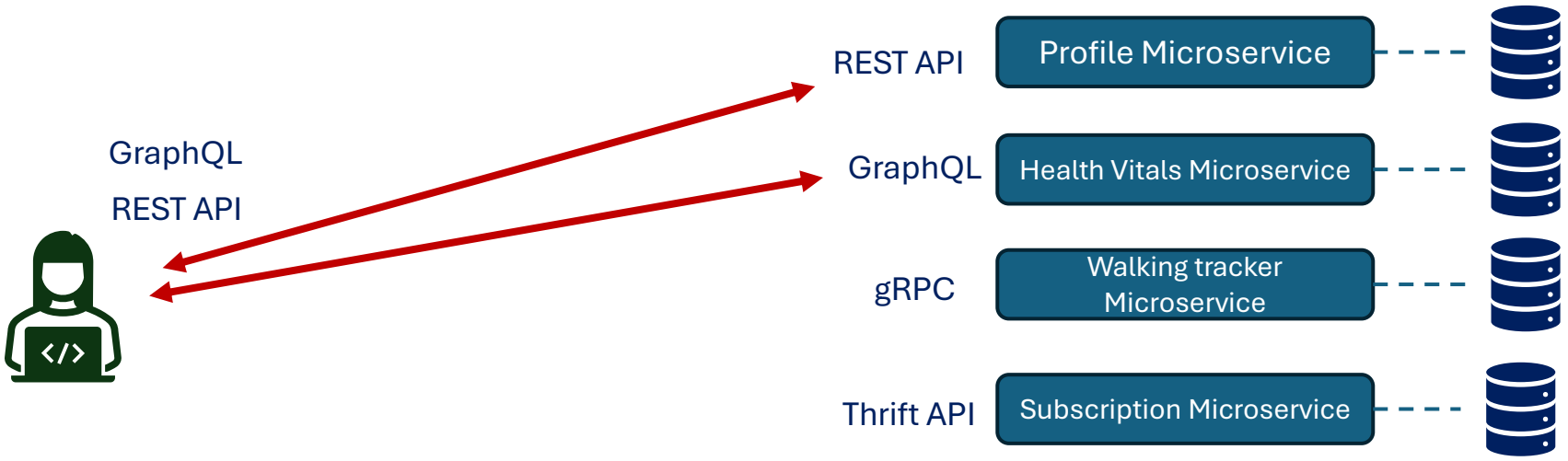
1. Upfront Cost of Infrastructure
2. Infrastructure maintenance cost
3. Steep Learning Curve

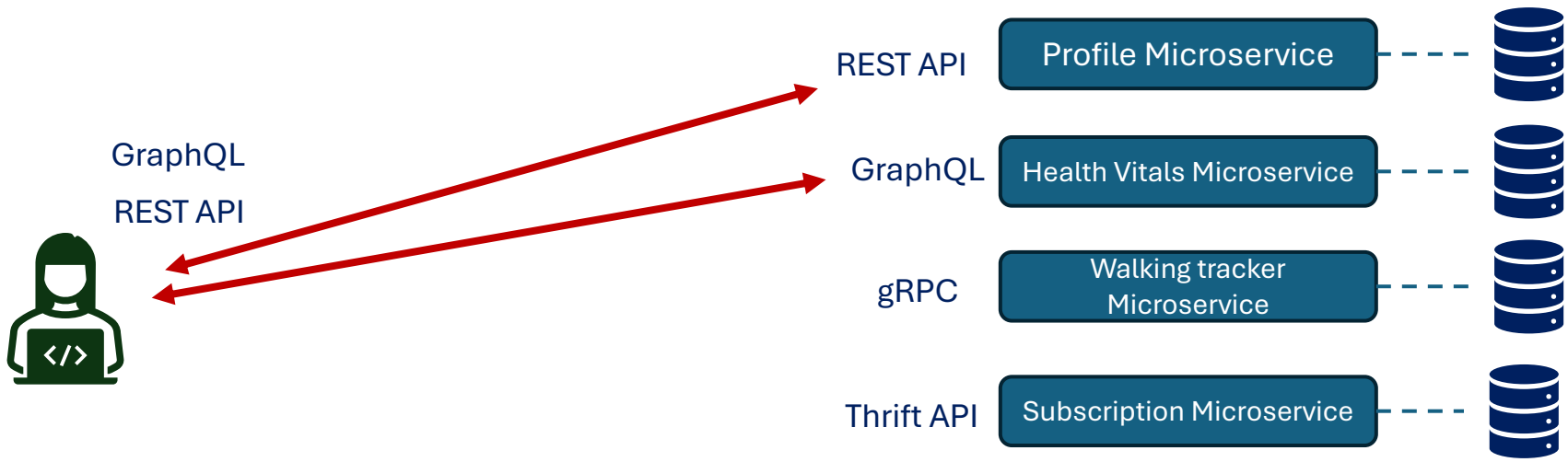


Full Team Autonomy Problem

1. Upfront Cost of Infrastructure
2. Infrastructure maintenance cost
3. Steep Learning Curve
4. Non-Uniform API

4. Non-Uniform API




GraphQL
REST API
REST API
Profile Microservice
GraphQL
Health Vitals Microservice
gRPC
Walking tracker Microservice
Thrift API
Subscription Microservice

Isn’t the point of Microservice to allow team independence?

Balance between autonomy and structure

Structures Autonomy

Topics

- Problem with full team autonomy
- 3 tiers of developers' team autonomy
- Factors of team autonomy boundaries

Tier 1 – Fully Restrictive

- Infrastructure
  - Monitoring and alerting
  - CI/CD
- API guidelines and best practices
- Security and data compliance

Tier 2 – Freedom with Boundaries

- Programming Languages
- Database technologies

Tier 3 – Complete Autonomy

- Release process
- Release schedule and frequency
- Custom scripts for local development and testing
- Documentation
- Onboarding process for new developers

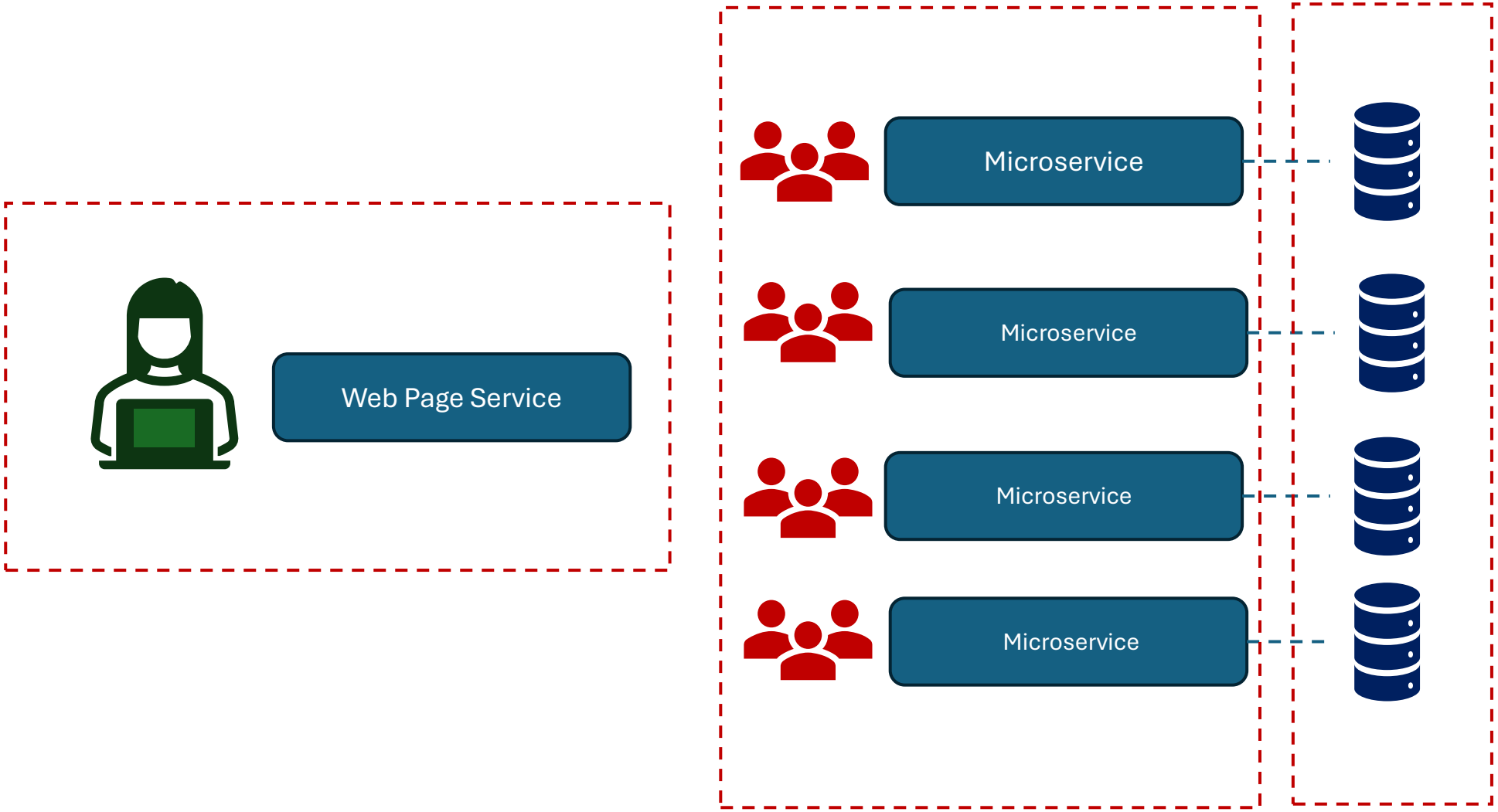
Topics

- Problem with full team autonomy
- 3 tiers of developers' team autonomy
- Factors of team autonomy boundaries

Factors of team autonomy boundaries

- Size/influence of DevOps / SRE team
- Seniority of developers
- Company's culture

Micro-Frontends Architecture Pattern


Web Page Service
Microservice
Microservice
Microservice
Microservice

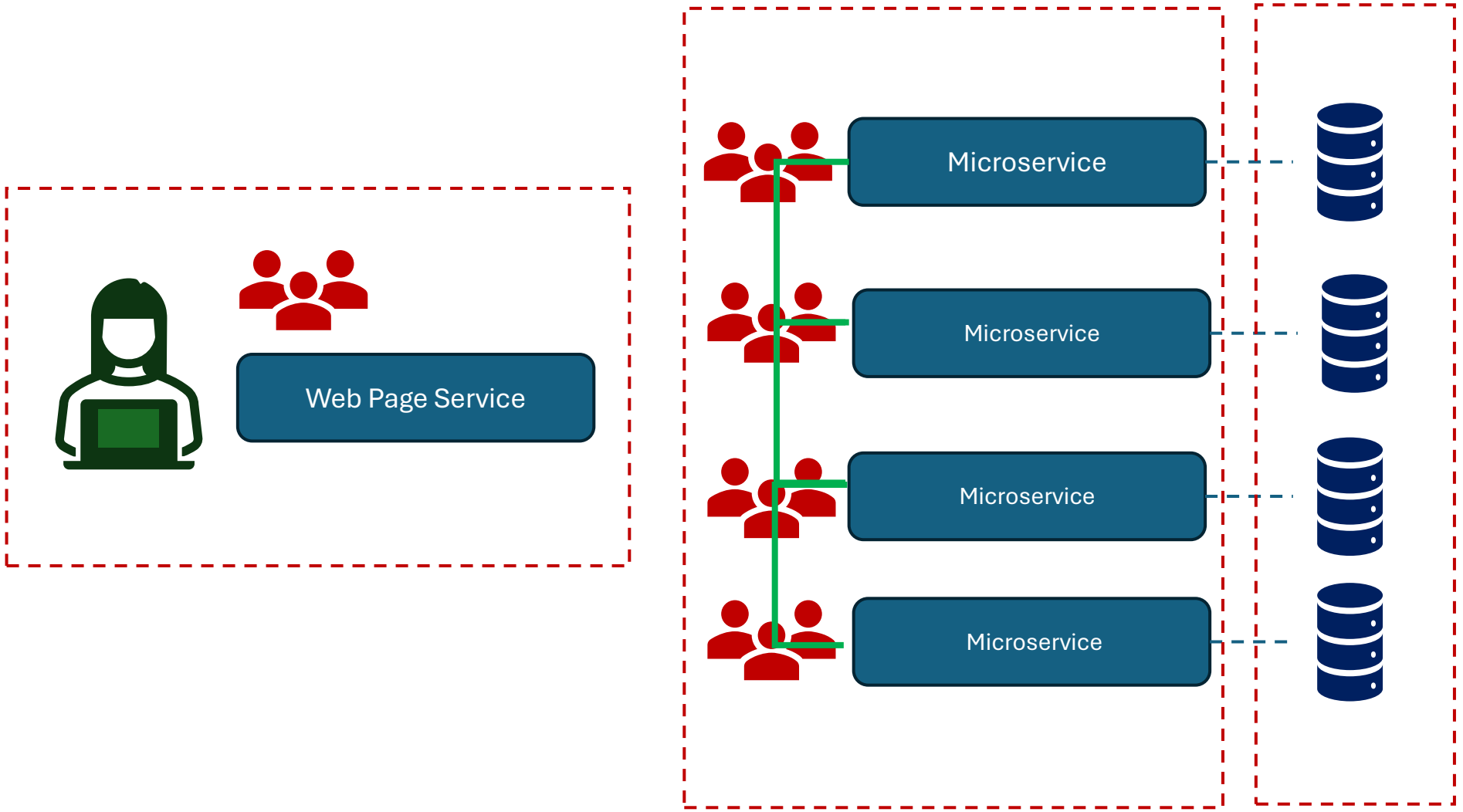
Topics

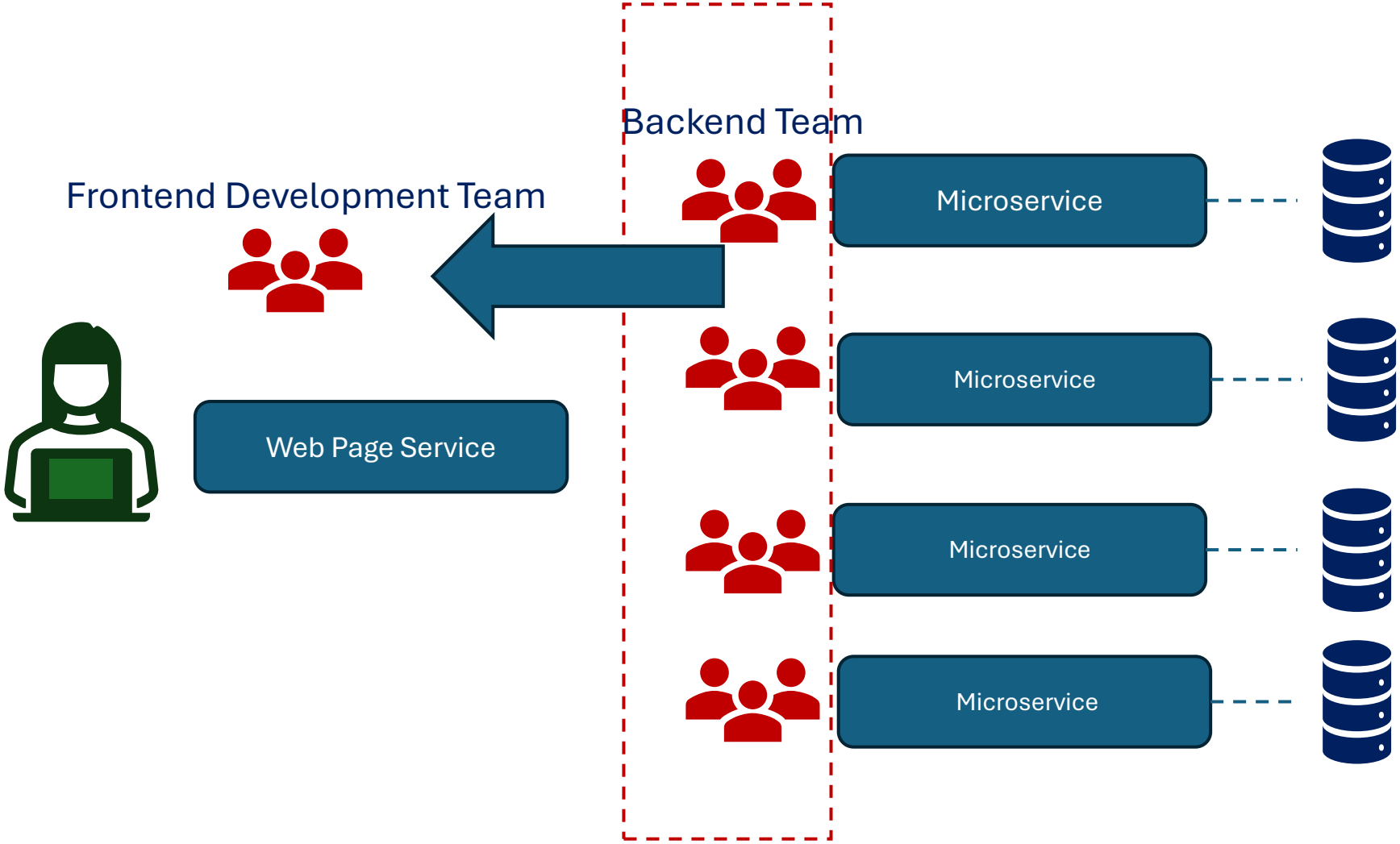
- Problems of a Monolithic frontend
- Micro-frontends architecture pattern
- Real-life example of micro-frontends + microservice
- Benefits and best practices

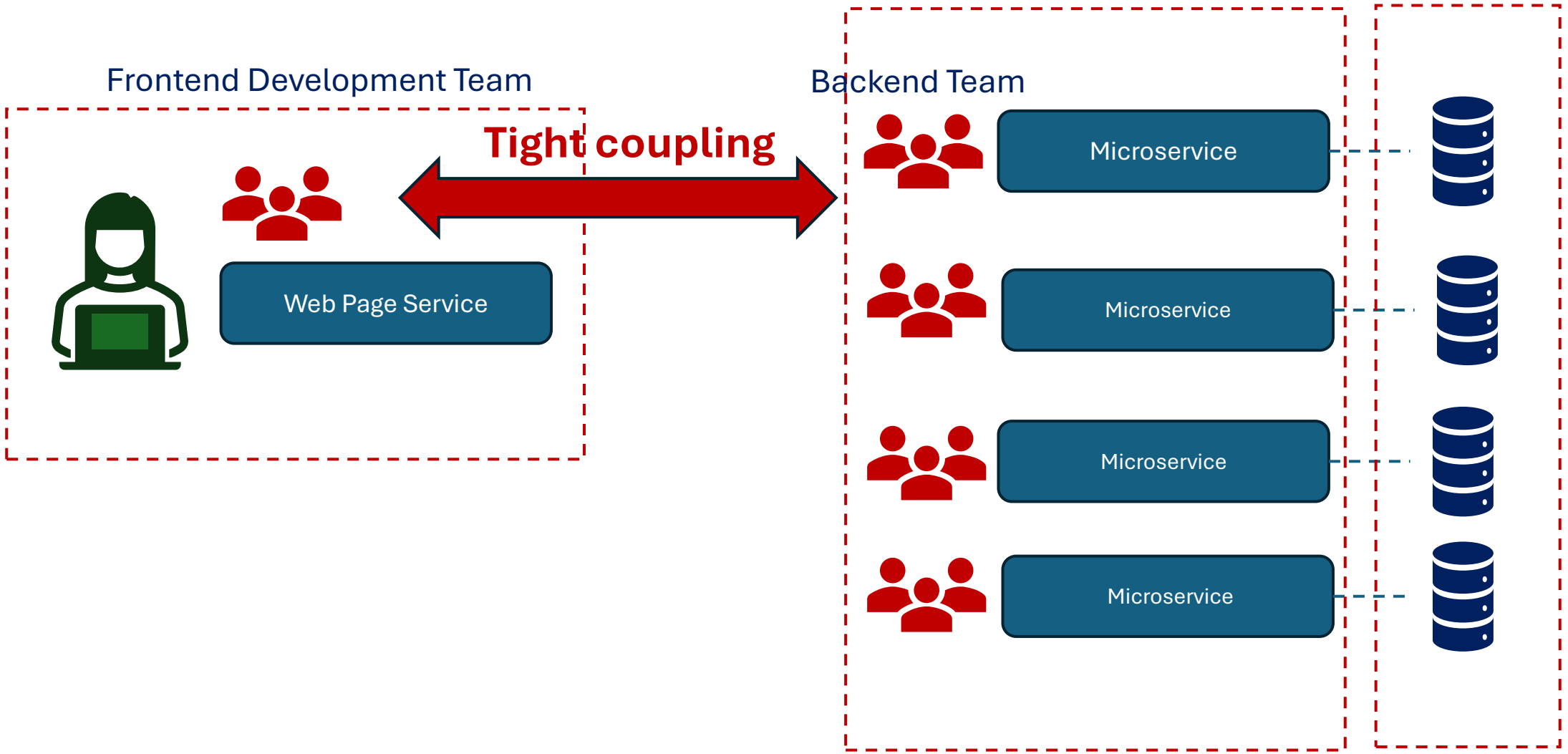
Topics

- Problems of a Monolithic frontend
- Micro-frontends architecture pattern
- Real-life example of micro-frontends + microservice
- Benefits and best practices


E-LEARNING


Web Page Service
Microservice
Microservice
Microservice
Microservice

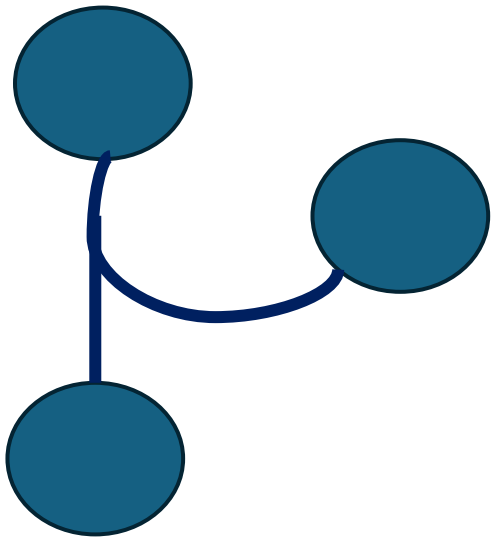

Backend Team
Frontend Development Team
Microservice
Microservice
Microservice
Microservice
Web Page Service


Frontend Development Team
Tight coupling
Backend Team
Web Page Service
Microservice
Microservice
Microservice
Microservice

Topics

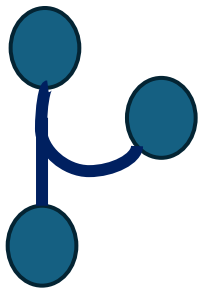
- Problems of a Monolithic frontend
- Micro-frontends architecture pattern
- Real-life example of micro-frontends + microservice
- Benefits and best practices

Splitting a Monolithic Frontend to Micro-frontends

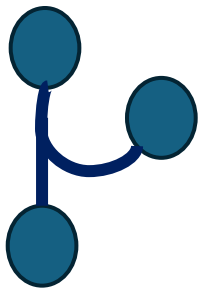


Monolithic Frontend Codebase (CSS, JS, HTML)

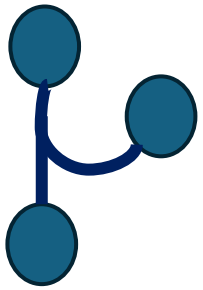
Splitting a Monolithic Frontend to Micro-frontends



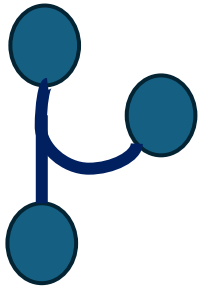
Monolithic Frontend Codebase (CSS, JS, HTML)



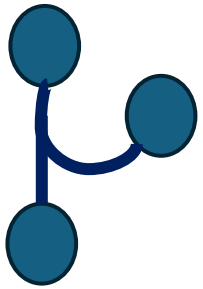
Monolithic Frontend Codebase (CSS, JS, HTML)



Monolithic Frontend Codebase (CSS, JS, HTML)

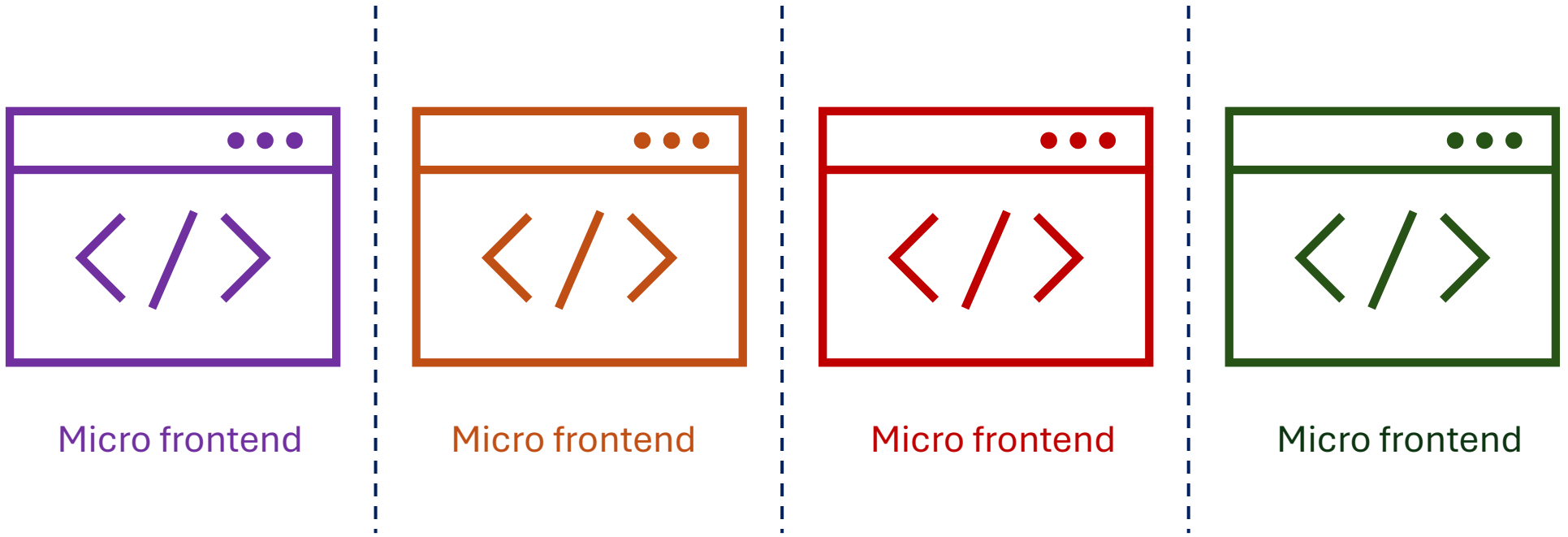


Monolithic Frontend Codebase (CSS, JS, HTML)

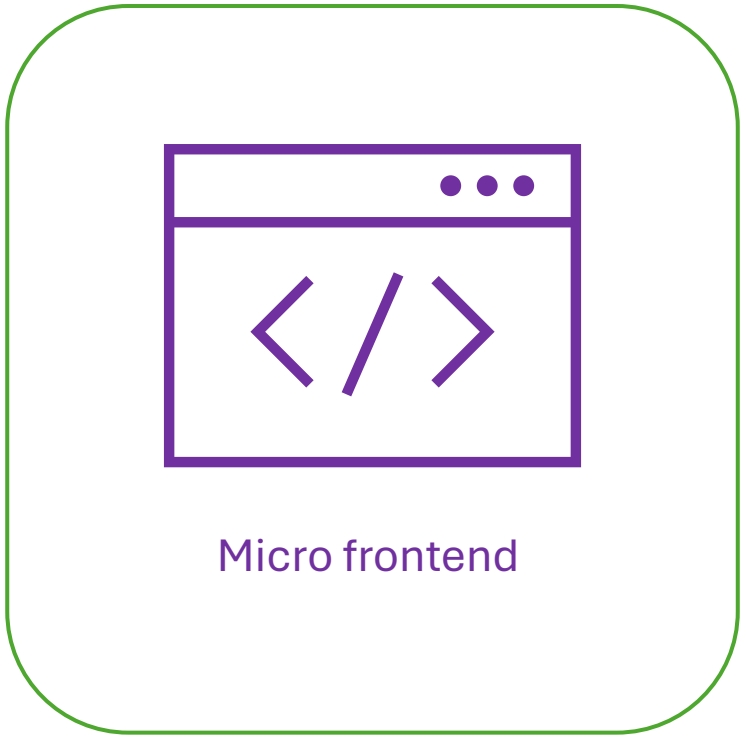


Monolithic Frontend Codebase (CSS, JS, HTML)

Splitting a Monolithic Frontend to Micro-frontends

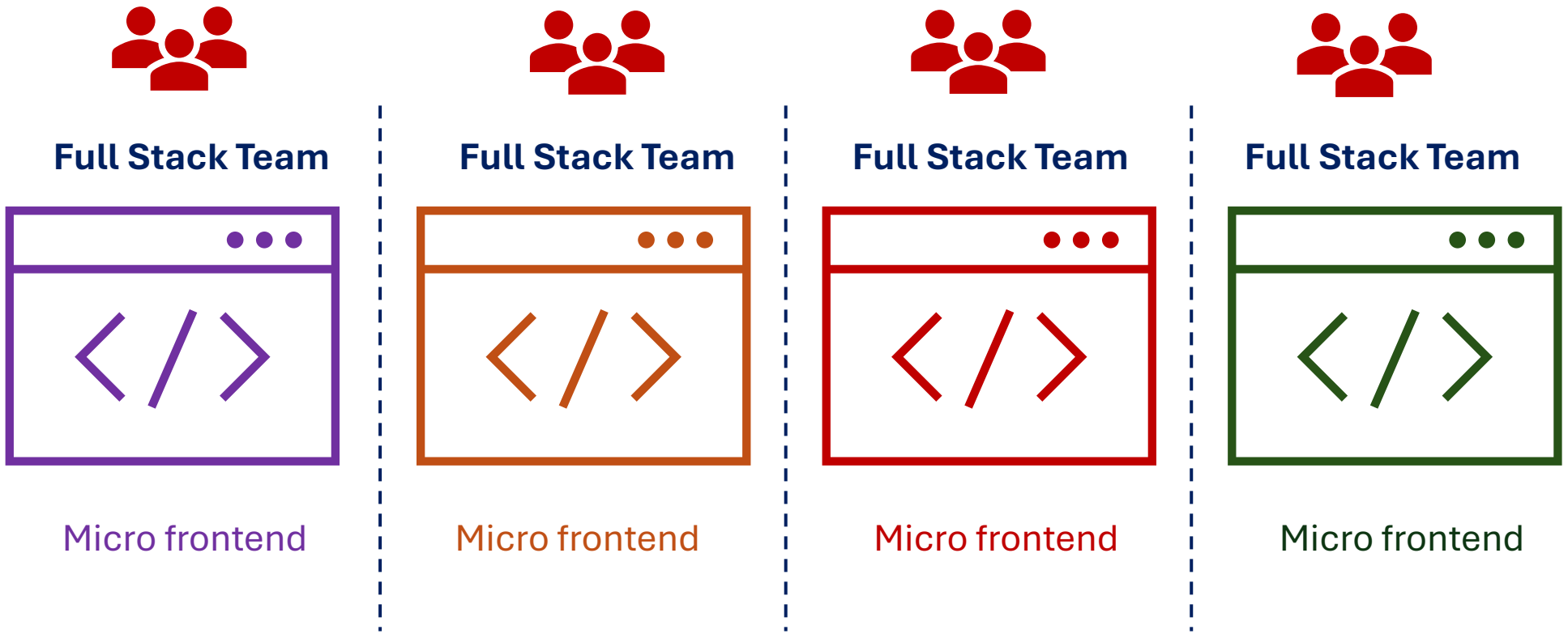


Splitting a Monolithic Frontend to Micro-frontends

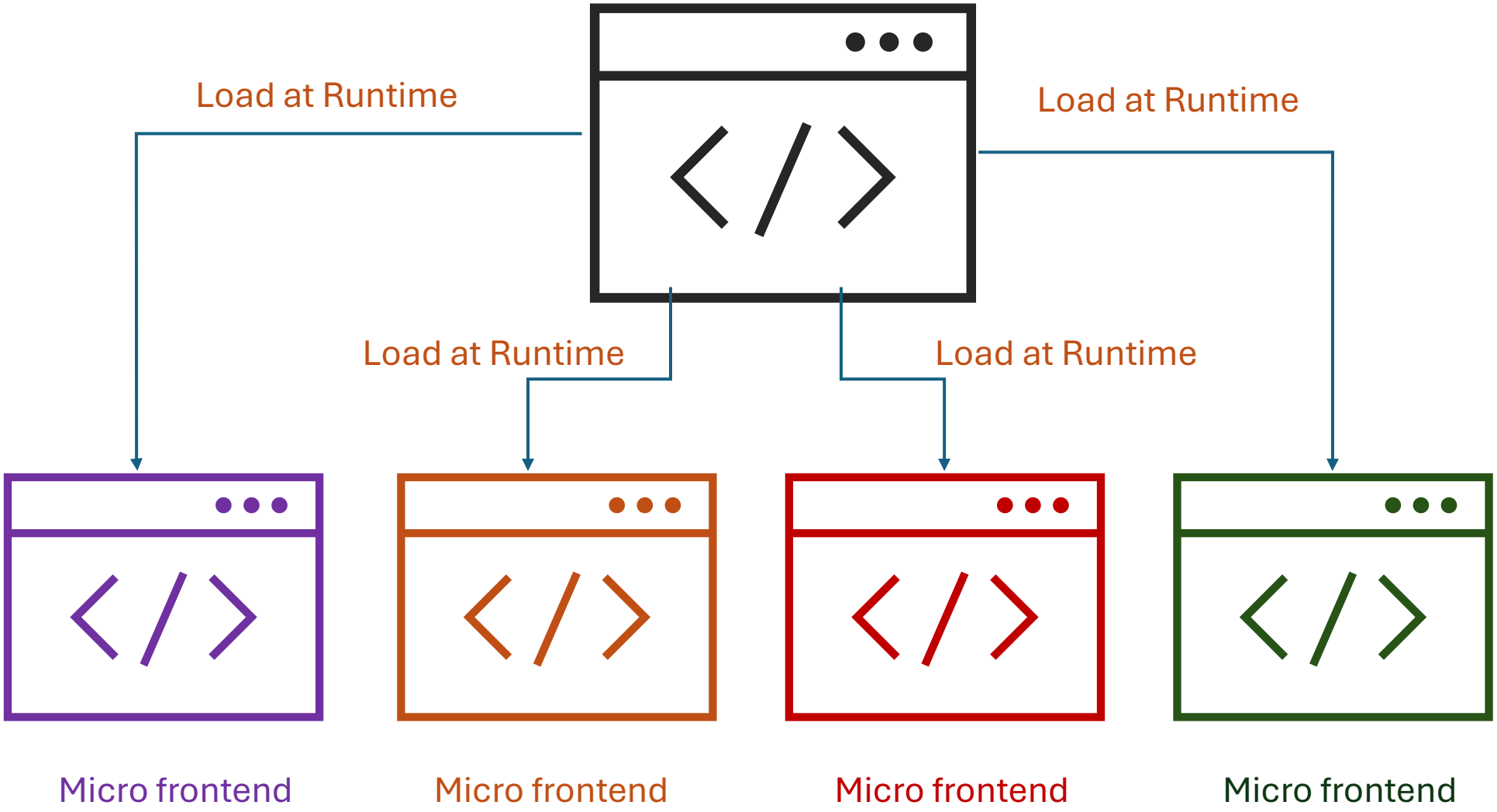


Standalone

Splitting a Monolithic Frontend to Micro-frontends



Container Application



Role of the Container Application

1. Render common element
2. Take care of common functionality
3. Tell each micro-frontend where/when to be rendered

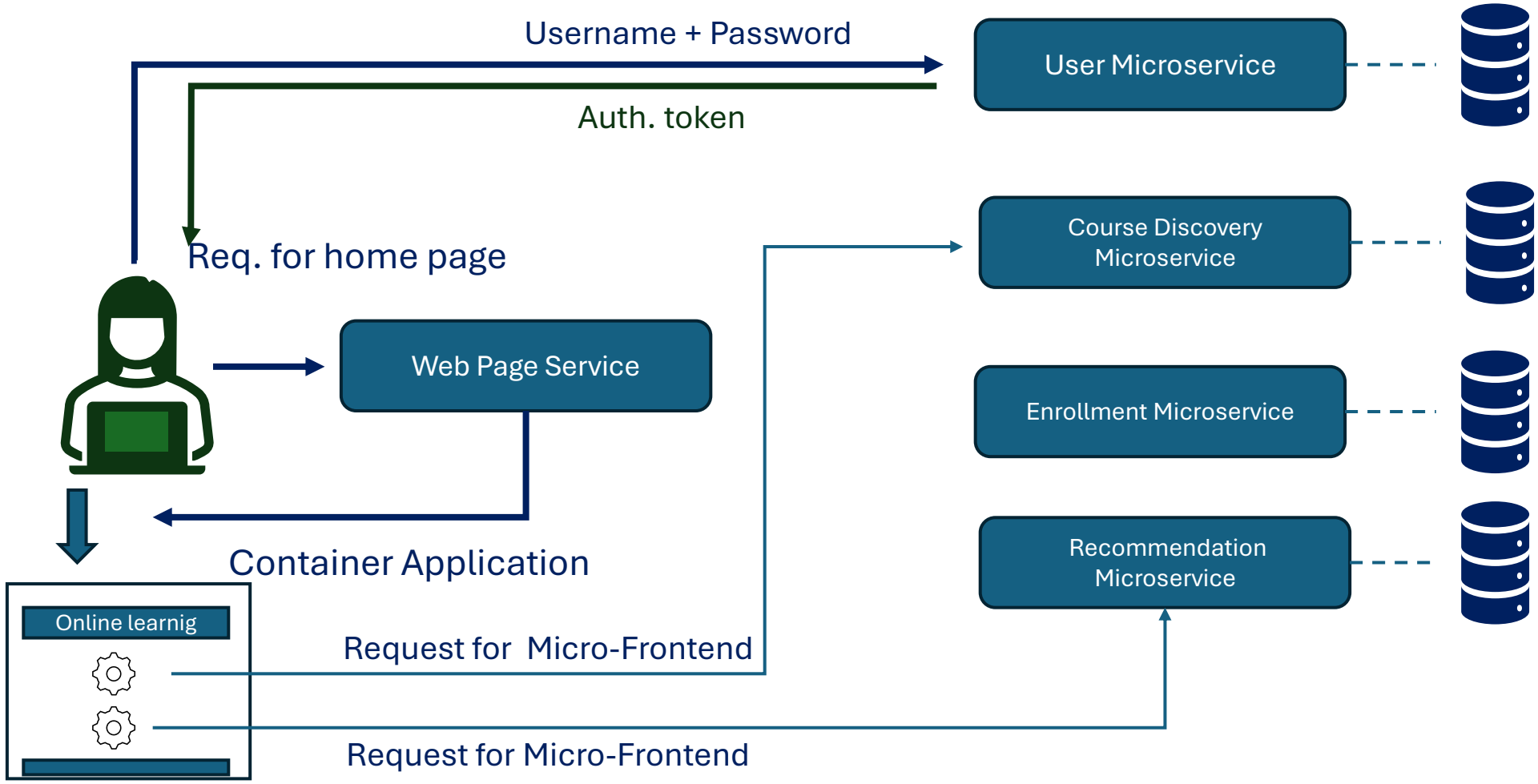
Source of confusion

- Micro-frontends are an architecture pattern, not a framework
- Micro-frontends are not reusable UI

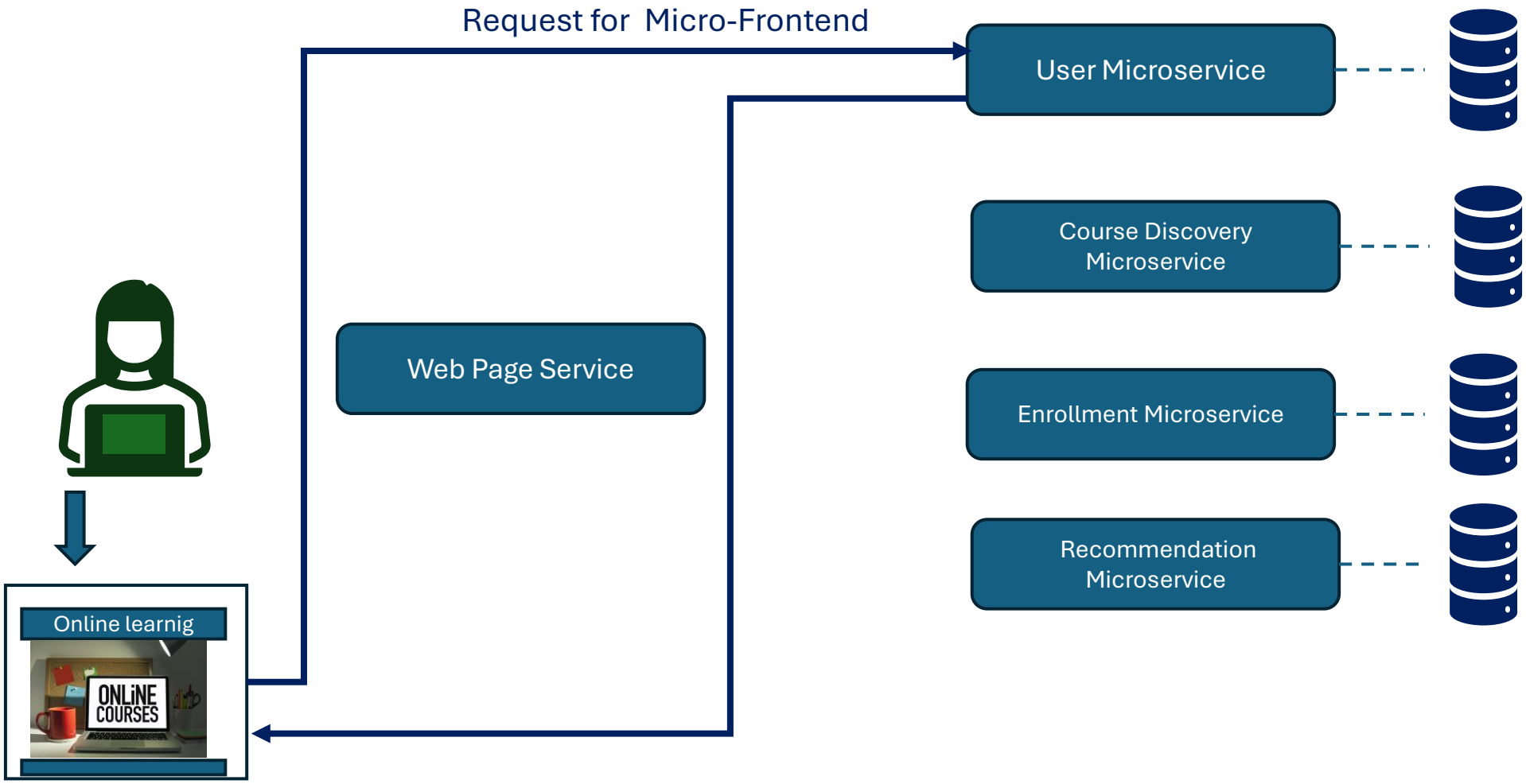
Topics

- Problems of a Monolithic frontend
- Micro-frontends architecture pattern
- Real-life example of micro-frontends + microservice
- Benefits and best practices

Online Learning Platform – with Micro-Frontends



Online Learning Platform – with Micro-Frontends



Topic

- Problems of a Monolithic frontend
- Micro-frontends architecture pattern
- Real-life example of micro-frontends + microservice
- Benefits and best practices

Benefits and best practices

- Replaced the complex monolithic codebase with small and manageable micro-frontends
- Full-stack ownership of each micro-frontend
- Easier/Faster to test in isolation
- Separate CI/CD pipeline
- Separate release schedule

Best Practices

- Micro-frontends are loaded at runtime
- No shared state in the browser
- Intercommunication through:
  - Custom Events
  - Callbacks
  - Address bar

Summary

- Identified our system’s bottleneck: Monolith frontend
- Learned about micro-frontends Architecture pattern
- Splits the web application into
  - Independent single-page application
  - Maintained by separate teams
  - Assembled by a container application at runtime
- Perfect for microservices architecture

API Management for Microservice Architecture

API Management

Topics

- The problem of managing APIs in Microservices Architecture
- The API Gateway path
- Load Balancer vs API Gateway

Topics

- The problem of managing APIs in Microservices Architecture
- The API Gateway path
- Load Balancer vs API Gateway

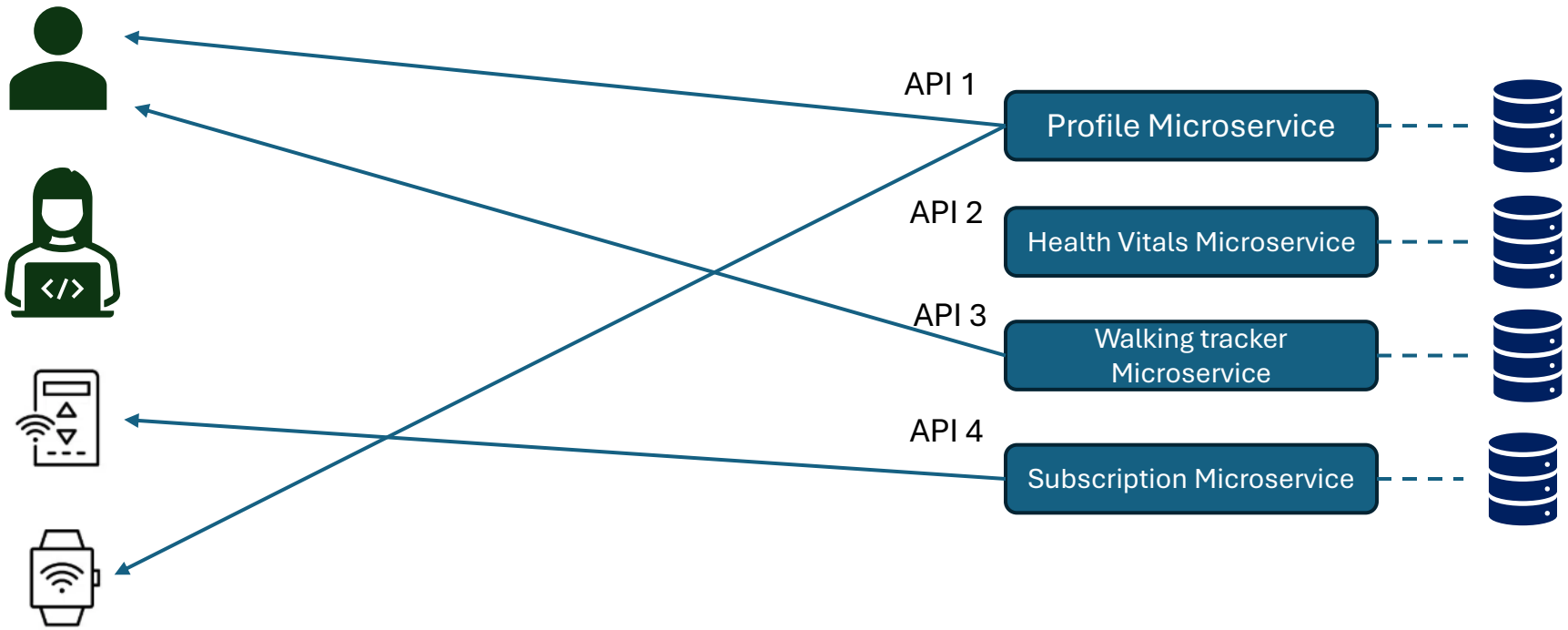
API

- **API** stands for **Application Programming Interface**. It is a set of rules, protocols, and tools that allow different software applications to communicate with each other.
- Role of API in Software Development
  - Facilitates Integration
  - Enables Reusability
  - **Improves Efficiency**
  - Supports Modularity

Technology Choices

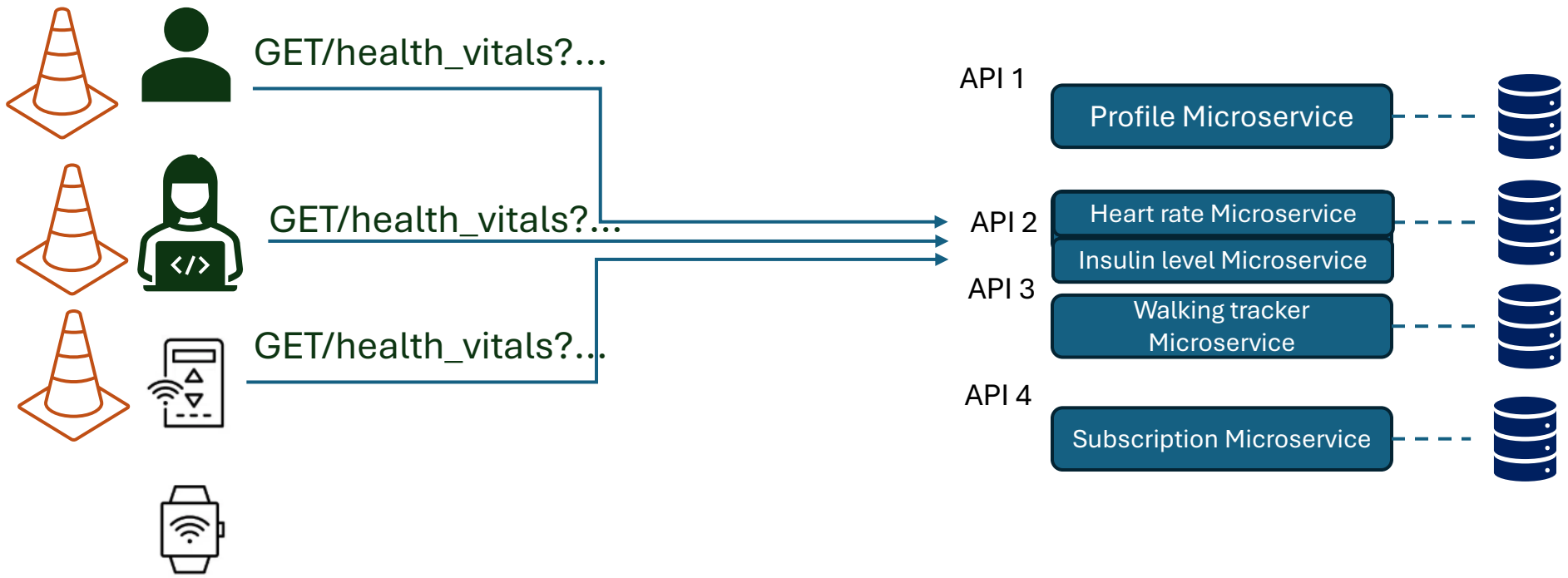
- RPC
  - SOAP
  - gRPC
- REST
- GraphQL
- Message brokers

API Management Problem



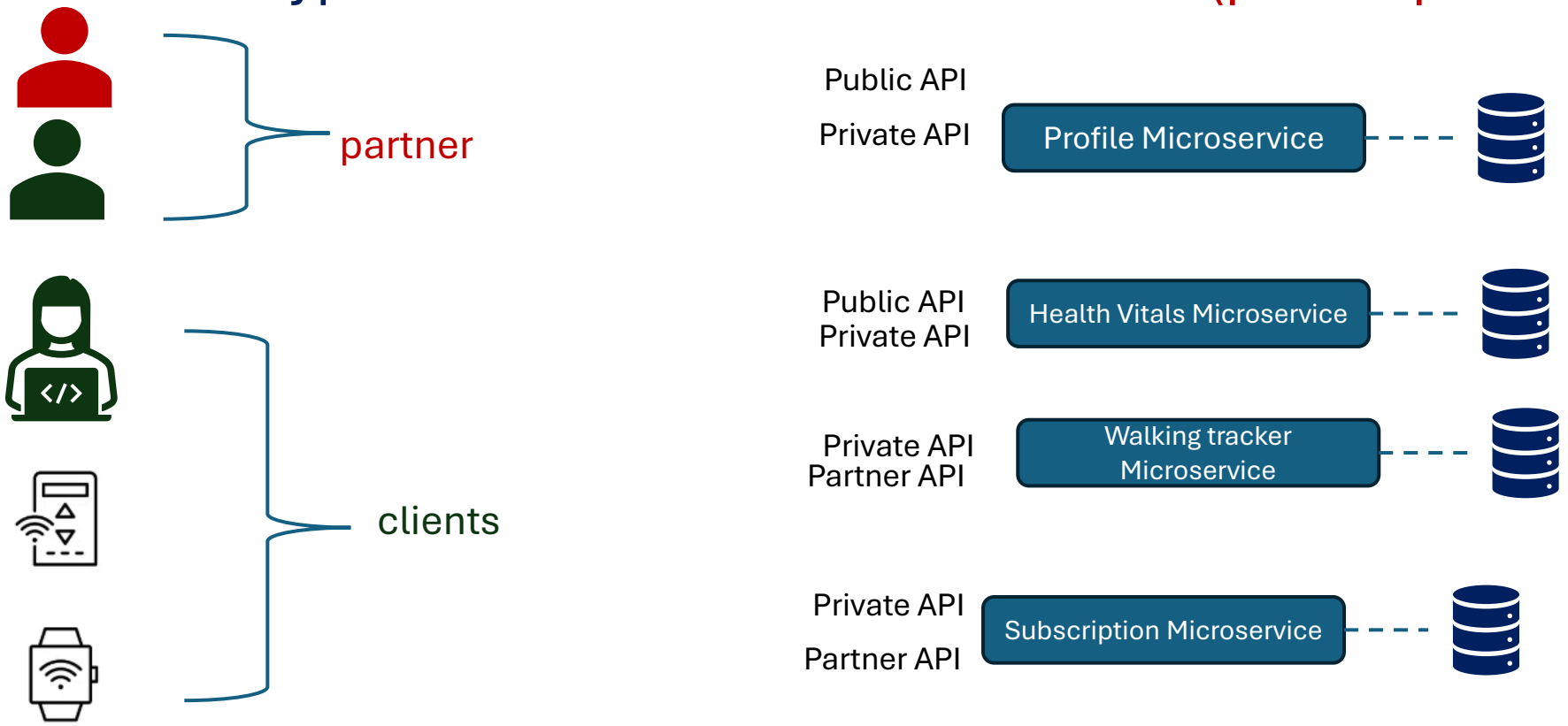
API Management Problems

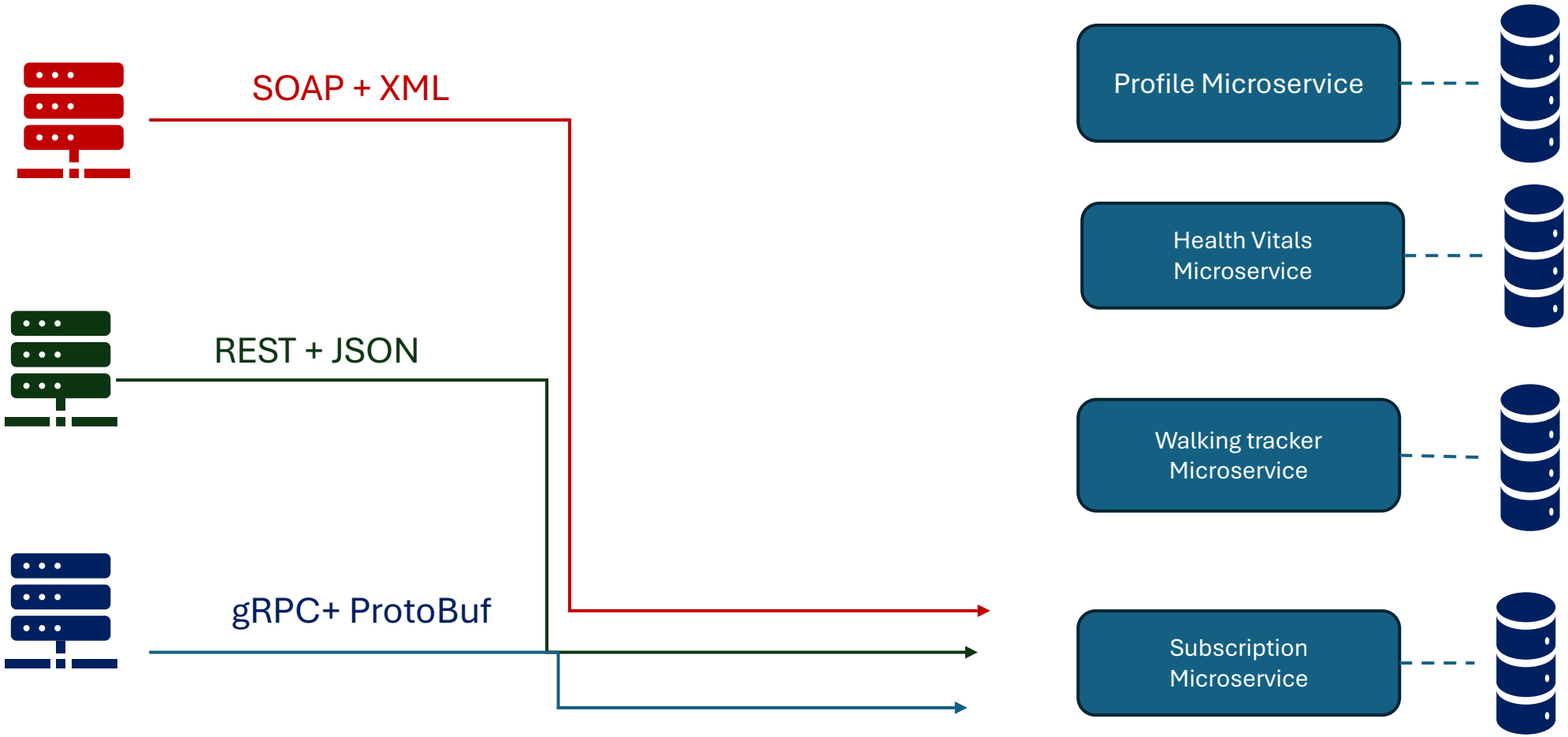
1. Tight coupling of API endpoints to client-side code



API Management Problem

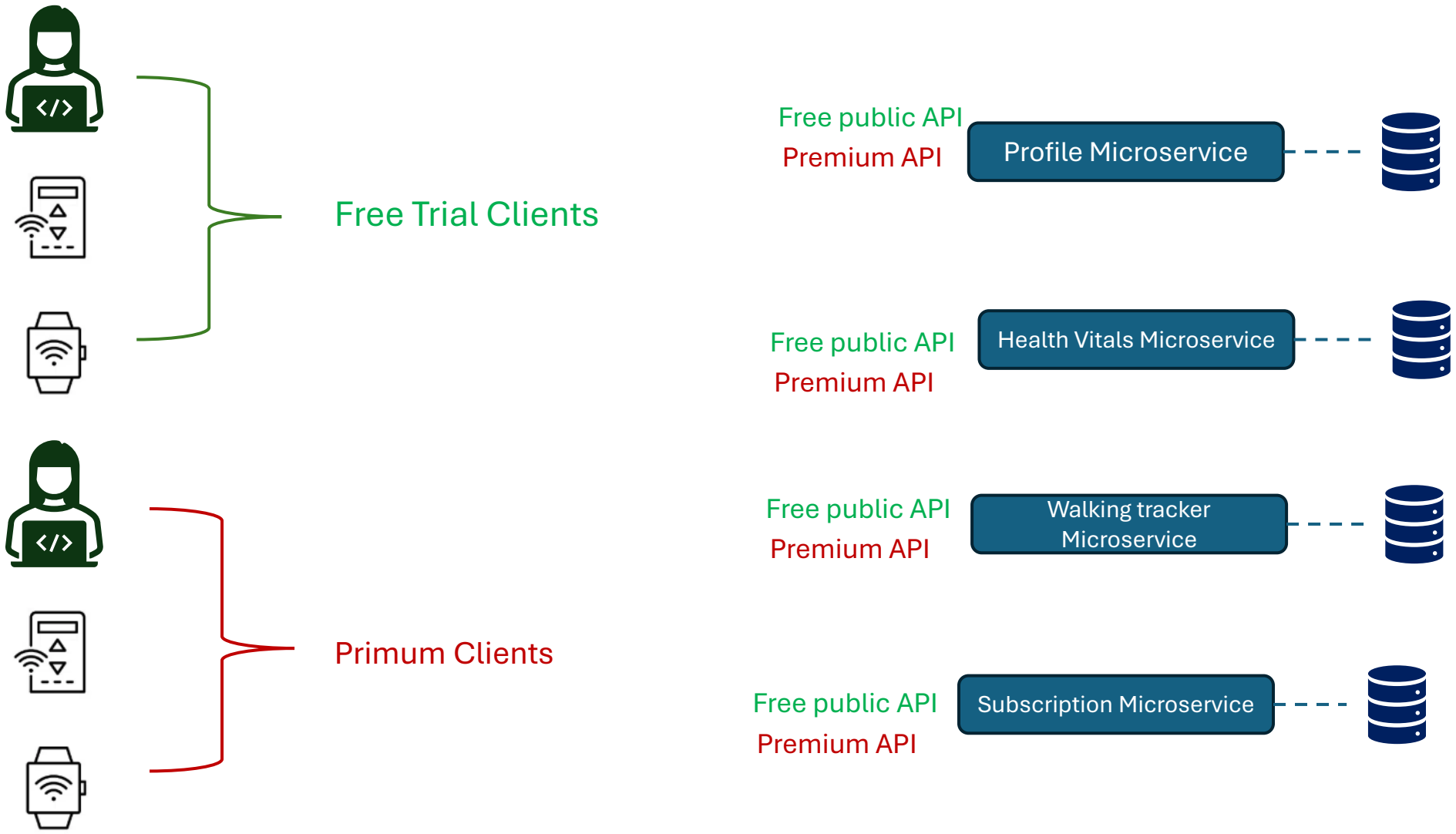
1. Tight coupling of API endpoints to client-side code
2. Different types of API for different customers (public/private/partner)




SOAP + XML
REST + JSON
gRPC+ ProtoBuf
Profile Microservice
Health Vitals Microservice
Walking tracker Microservice
Subscription Microservice

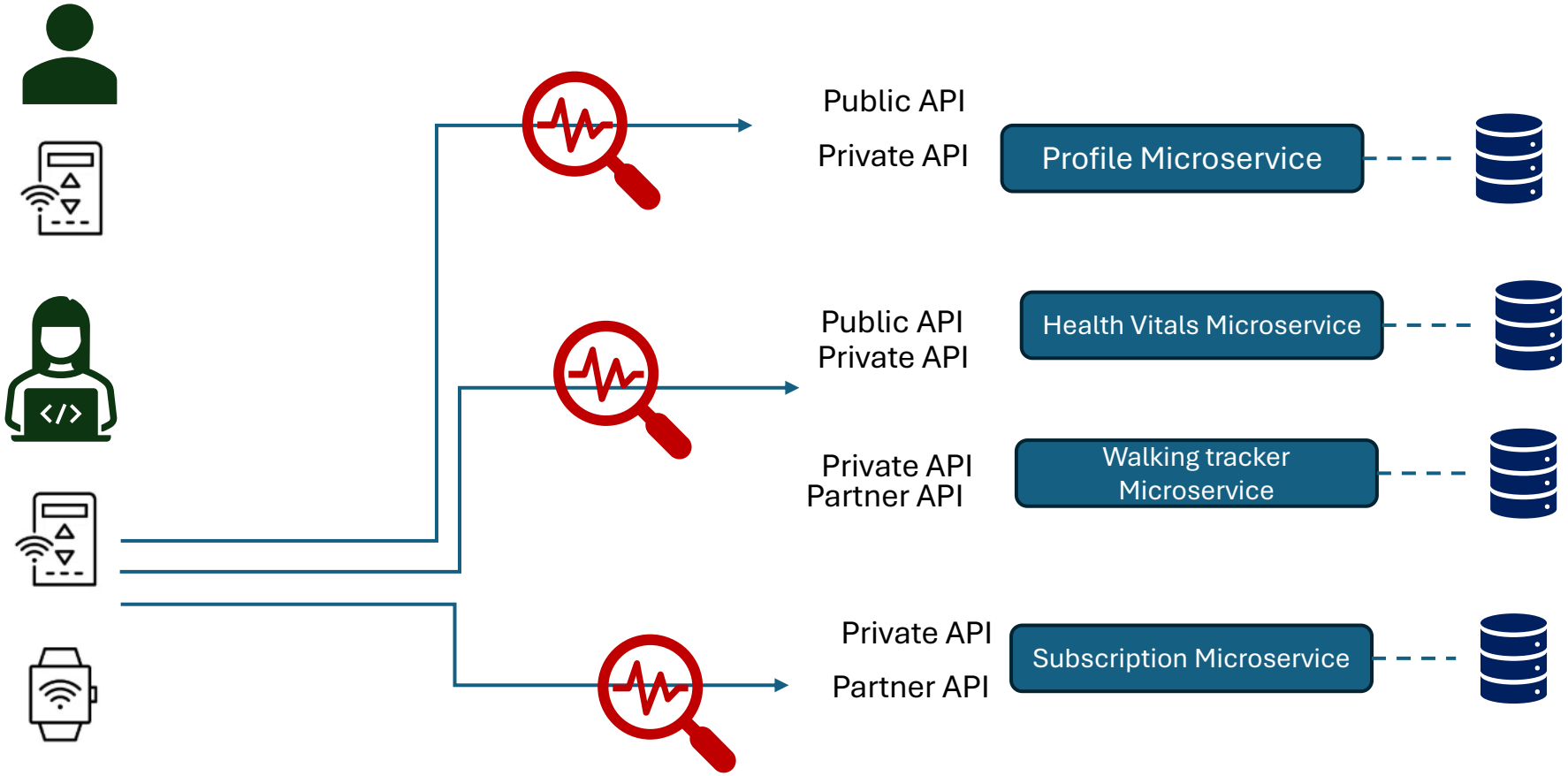
API Management Problems

1. Tight coupling of API endpoints to client-side code
2. Different types of API for different customers (public/private/partner)
3. Different API tiers based on subscription


Free Trial Clients
Primum Clients
Free public API
Premium API
Profile Microservice
Free public API
Premium API
Health Vitals Microservice
Free public API
Premium API
Walking tracker Microservice
Free public API
Premium API
Subscription Microservice

API Management Problems

1. Tight coupling of API endpoints to client-side code
2. Different types of API for different customers (public/private/partner)
3. Different API tiers based on subscription
4. Traffic Control and monitoring


Public API
Private API
Profile Microservice
Public API
Private API
Health Vitals Microservice
Private API
Partner API
Walking tracker Microservice
Private API
Partner API
Subscription Microservice

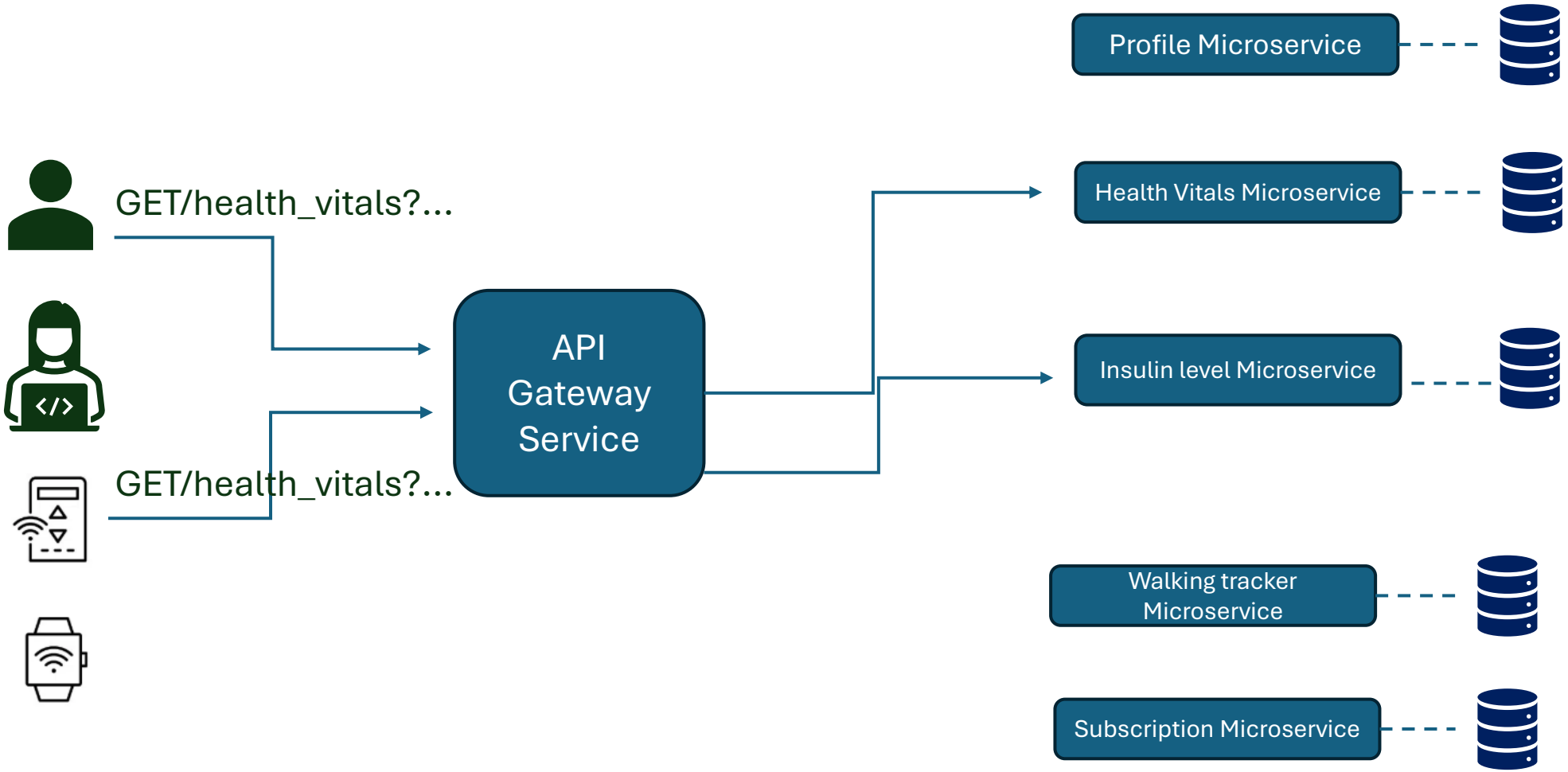

Profile Microservice
Health Vitals Microservice
Walking tracker Microservice
Subscription Microservice

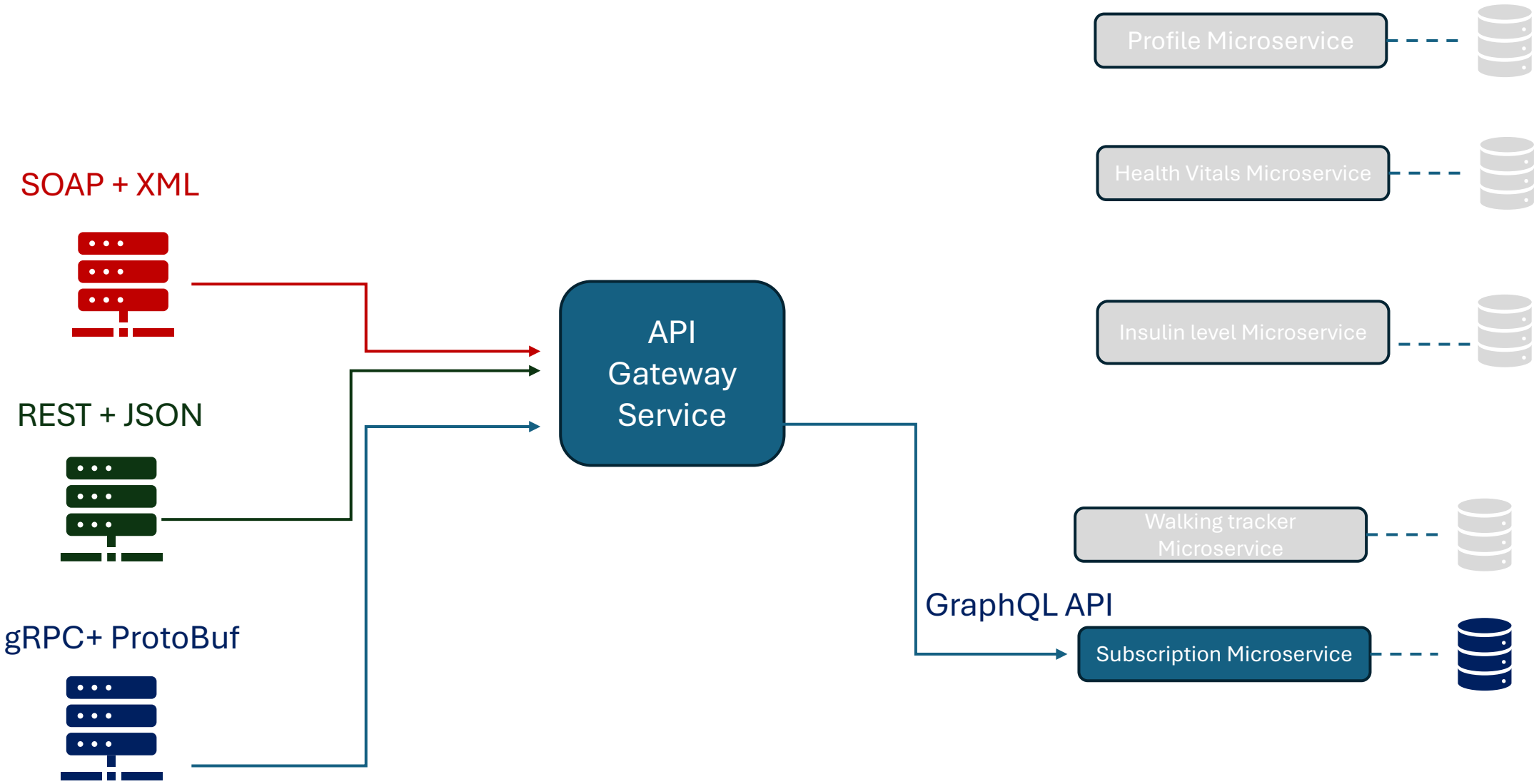
API Management Problems

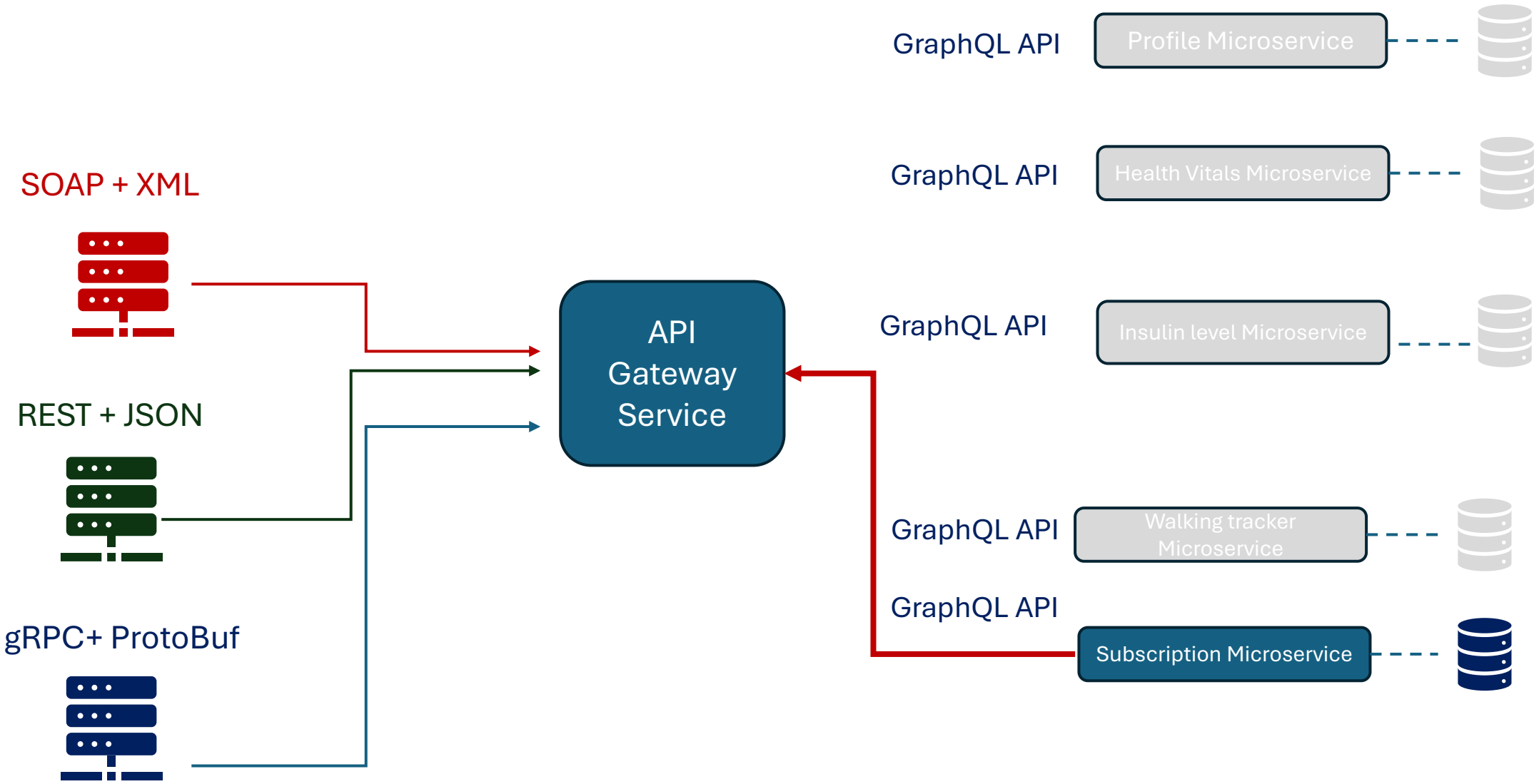
1. Tight coupling of API endpoints to client-side code
2. Different types of API for different customers (public/private/partner)
3. Different API tiers based on subscription
4. Traffic Control and monitoring
5. Duplicate effort across microservices

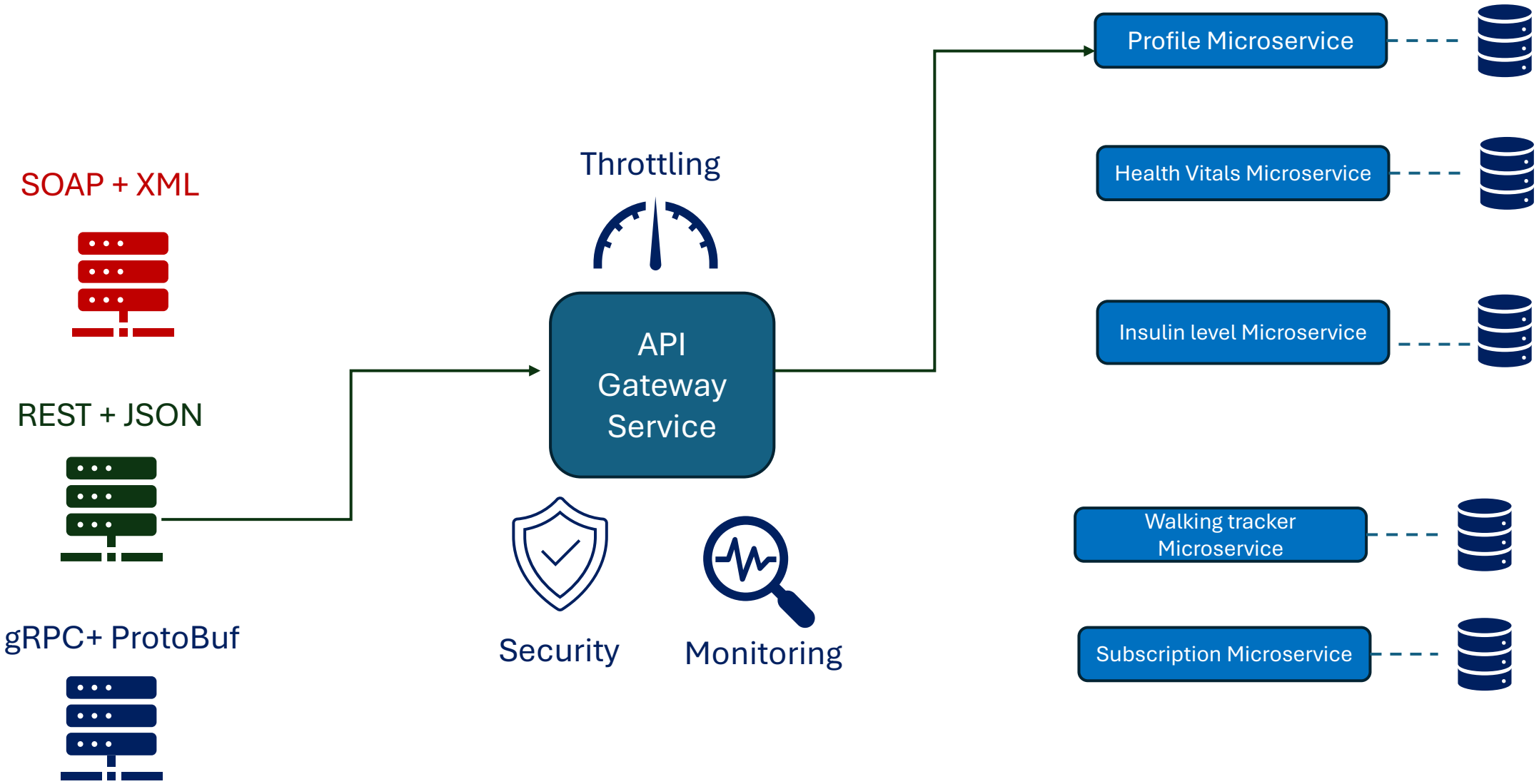
Topics

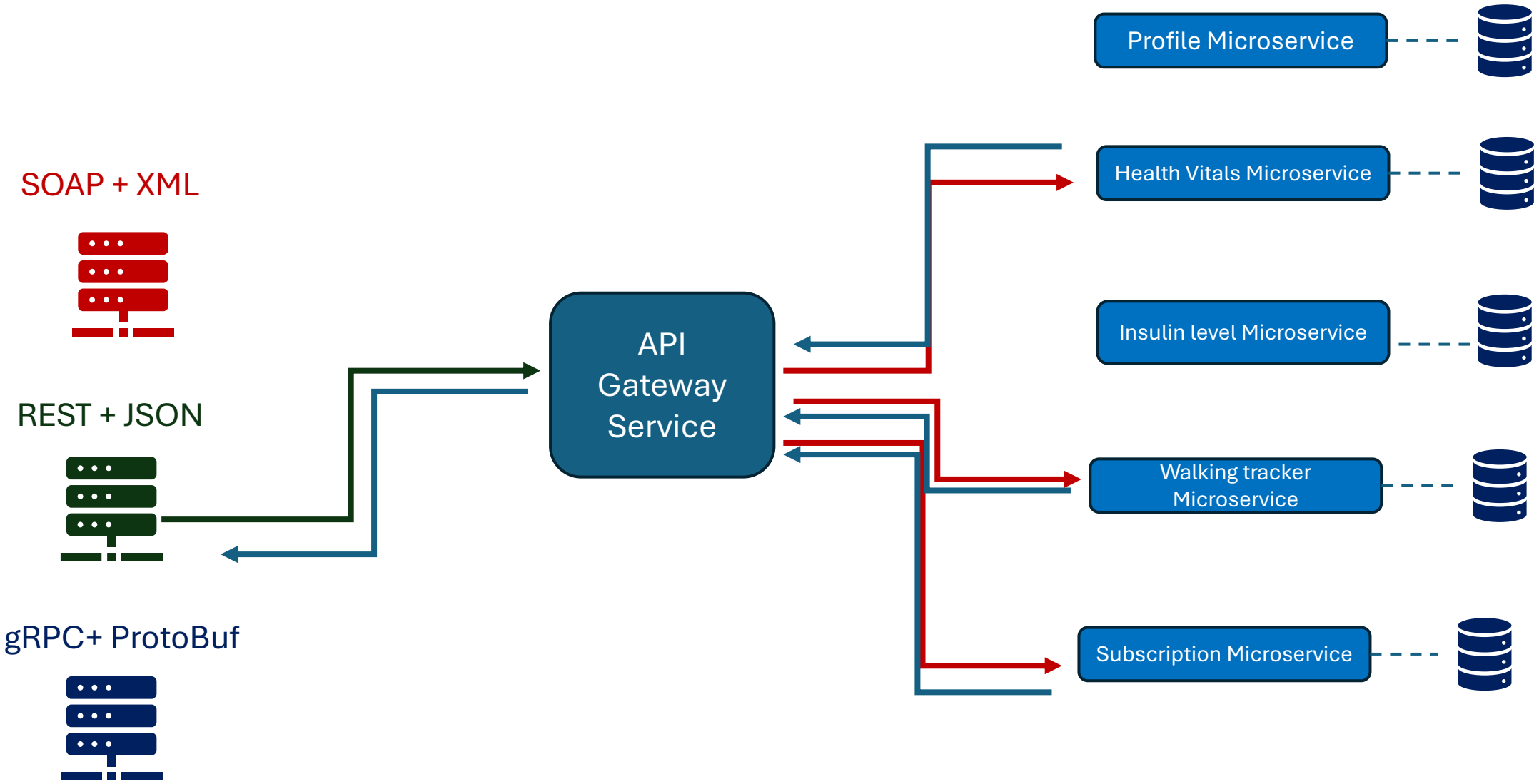
- The problem of managing APIs in Microservices Architecture
- The API Gateway path
- Load Balancer vs API Gateway


GET/health_vitals?...
GET/health_vitals?...
API Gateway Service
Profile Microservice
Health Vitals Microservice
Insulin level Microservice
Walking tracker Microservice
Subscription Microservice


SOAP + XML
REST + JSON
gRPC+ ProtoBuf
API Gateway Service
GraphQL API
Profile Microservice
Health Vitals Microservice
Insulin level Microservice
Walking tracker Microservice
Subscription Microservice


SOAP + XML
REST + JSON
gRPC+ ProtoBuf
API Gateway Service
GraphQL API
Profile Microservice
GraphQL API
Health Vitals Microservice
GraphQL API
Insulin level Microservice
GraphQL API
Walking tracker Microservice
GraphQL API
Subscription Microservice


SOAP + XML
REST + JSON
gRPC+ ProtoBuf
Throttling
API Gateway Service
Security
Monitoring
Profile Microservice
Health Vitals Microservice
Insulin level Microservice
Walking tracker Microservice
Subscription Microservice


SOAP + XML
REST + JSON
gRPC+ ProtoBuf
API Gateway Service
Profile Microservice
Health Vitals Microservice
Insulin level Microservice
Walking tracker Microservice
Subscription Microservice

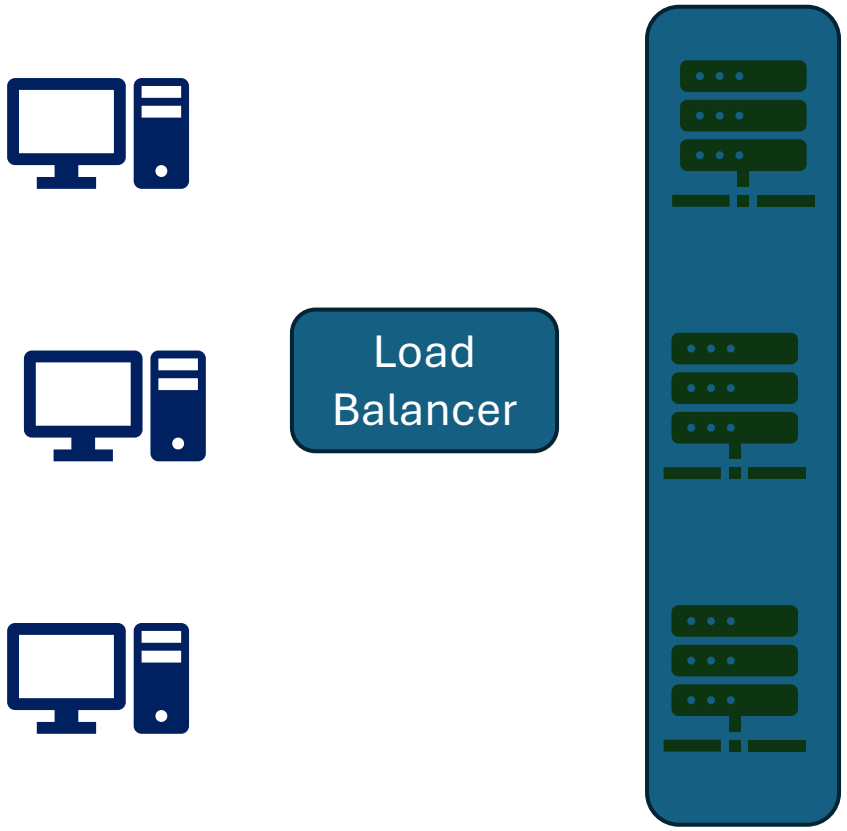
Topics

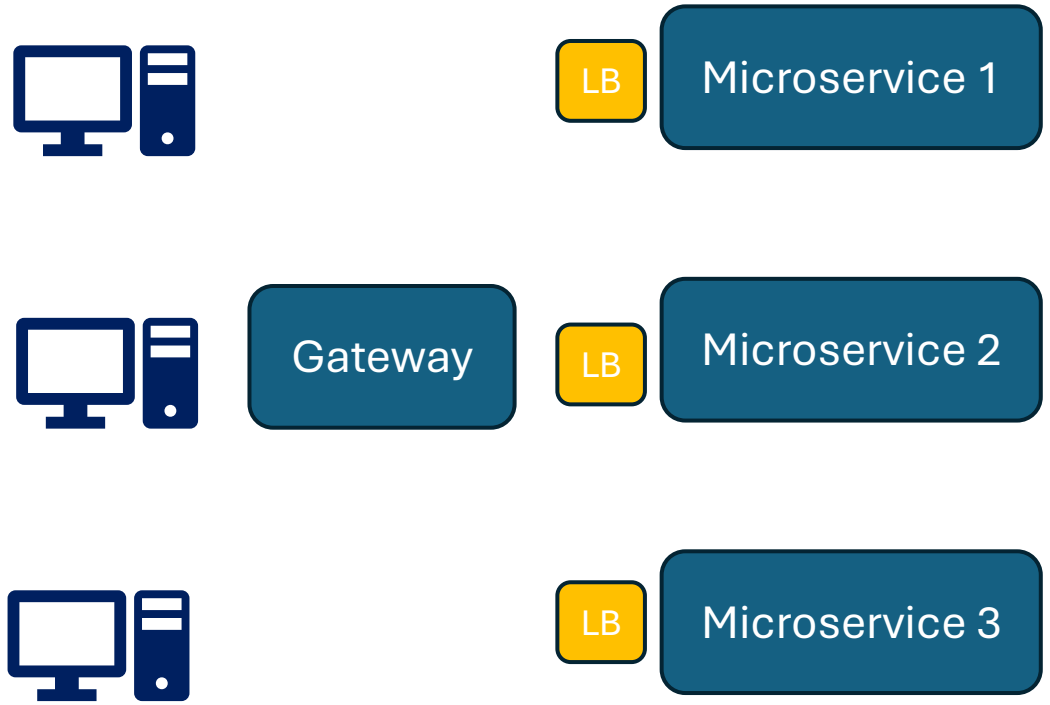
- The problem of managing APIs in Microservices Architecture
- The API Gateway path
- Load Balancer vs API Gateway

Load Balancer and API Gateway

- Route a request to a single destination
- Have two different purposes

Load Balancer





Load Balancer vs API Gateway

- Load Balancer features:
  - Little performance overhead
  - Health checks
  - Different routing algorithms

- Gateway:
  - Throttling
  - Monitoring
  - API Versioning & management
  - Protocols / Data translation

Summary

- Problems of managing APIs in Microservices Architecture
- API Gateway Pattern:
  - Routes requests to microservices
  - Throttling
  - Authorization and TLS termination
  - Protocol and data translation
  - Monitoring
- Load Balancer vs API Gateway