# Patterns

Mojtaba Shahin

## Week 10: Lectorial – Part 2

# Content

- Part 1
  - DevOps and Deployment Patterns
- Part 2
  - Decomposition Patterns
  - Legacy Modernization Patterns
  - C4 Model Dynamic and Deployment Diagrams

# Acknowledgements


- Most of the **texts** and **images** in the slides come from the following sources:
  - https://c4model.com/
  - https://github.com/structurizr/examples
  - https://structurizr.com/share/36141
  - Tremel, E. (2017): Six strategies for application deployment (https://thenewstack.io/deployment-strategies/)
  - Pautasso, C., Software Architecture- Visual Lecture Notes, LeanPub, 2023 (https://leanpub.com/software-architecture/)
  - Introduction to DevOps by Len Bass – URL: goo.gl/iMwdfg
  - Len Bass, Ingo Weber, Liming Zhu, DevOps: A Software Architect's Perspective, 2015
  - Martin Fowler (2024), "Branch By Abstraction" https://martinfowler.com/bliki/BranchByAbstraction.html
  - Ian Cartwright, Rob Horn, and James Lewis (2024), Patterns of Legacy Displacement (https://martinfowler.com/articles/patterns-legacy-displacement/)
  - The University of Queensland's Software Architecture Course Materials, @ Richard Thomas, CC BY-SA 4.0 (https://github.com/CSSE6400)
  - Neal Ford, Mark Richards, Pramod Sadalage, Zhamak Dehghani, Software Architecture: the Hard Parts, O'Reilly Media, 2021

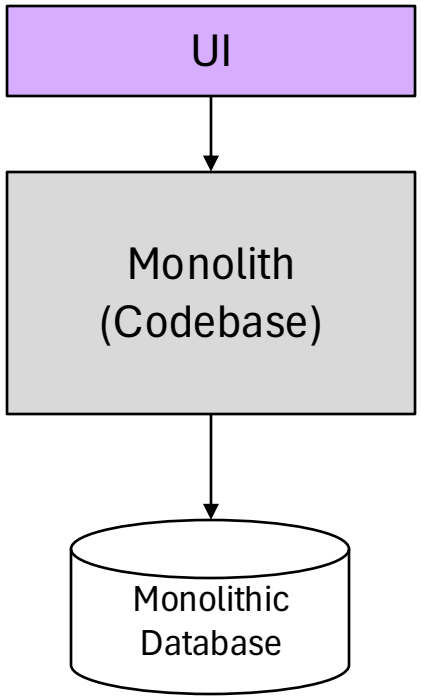
# Acknowledgements


- Most of the **texts** and **images** in the slides come from the following sources:
  - Newman, Sam. Building Microservices, O'Reilly Media, Second Edition, 2021
  - Mojtaba Shahin, Mansooreh Zahedi, Muhammad Ali Babar and Liming Zhu, *An Empirical Study of Architecting for Continuous Delivery and Deployment*, In: Empirical Software Engineering, 24(3), 2019, Springer
  - Mojtaba Shahin, Muhammad Ali Babar and Liming Zhu, *Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices*, In: IEEE Access, 5 (99), 2017, IEEE.
  - Matthew Skelton, Chris O'Dell , Continuous Delivery with Windows and .NET, 2016, O'Reilly Media, Inc.

# Decomposition Patterns

# Monolithic Architectures

- **Benefits**
  - Simplicity and Overall Costs
    - Easy Deployment
    - Simple communication between modules
    - Simple system testing & debugging
- **Problems**
  - Scalability
    - Components and database scale poorly
  - Elasticity
  - Fault tolerance
  - …



# Monolithic Architectures

## Question

What can be done if a monolith architecture is no longer suitable?

## Answer

(1) Greenfields replacement

(2) Migrate to another architecture

# Monolithic Architectures

## Question

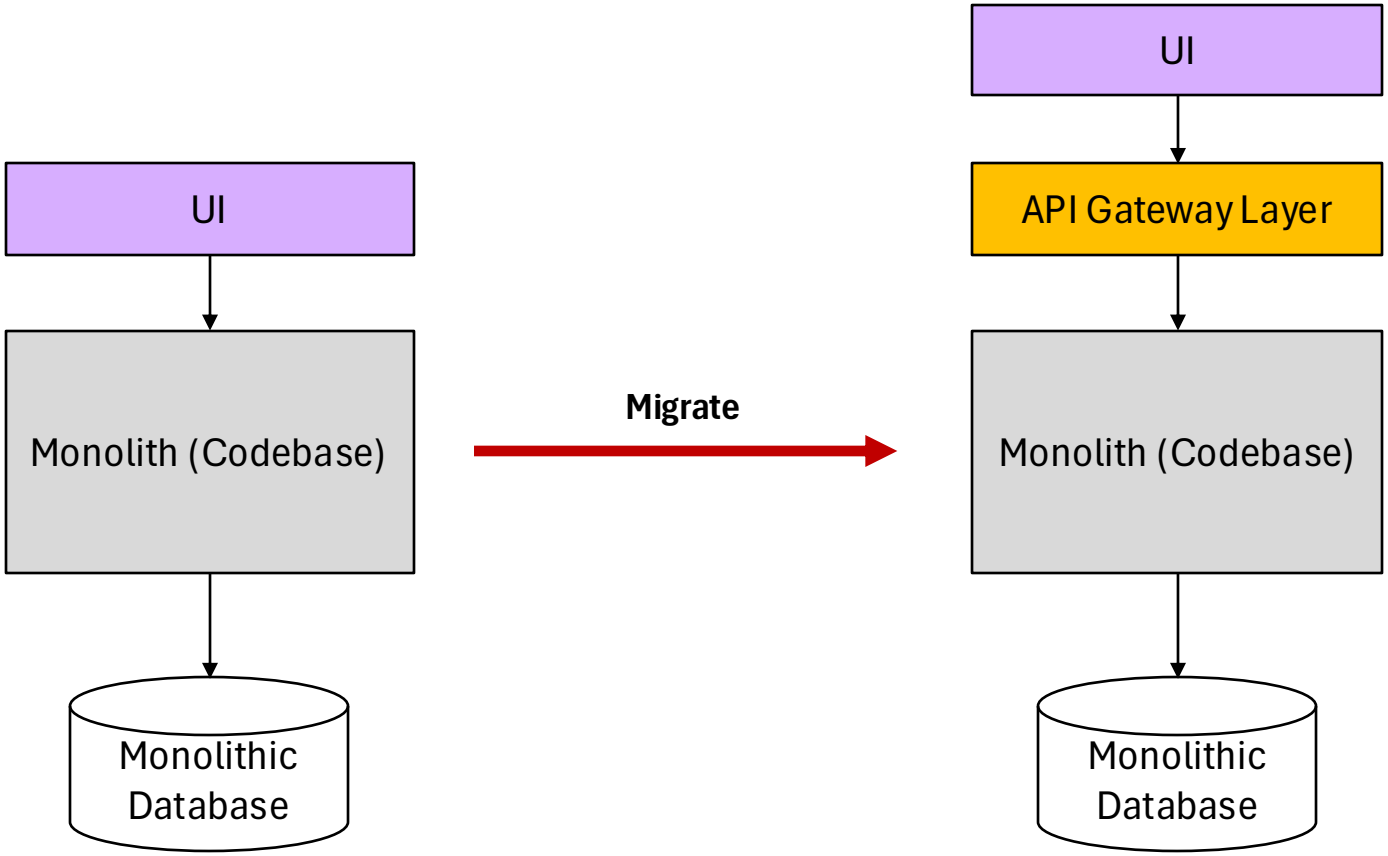
How to migrate the monolith to another architecture?

## Answer

Split (decompose) the monolith into (micro) services

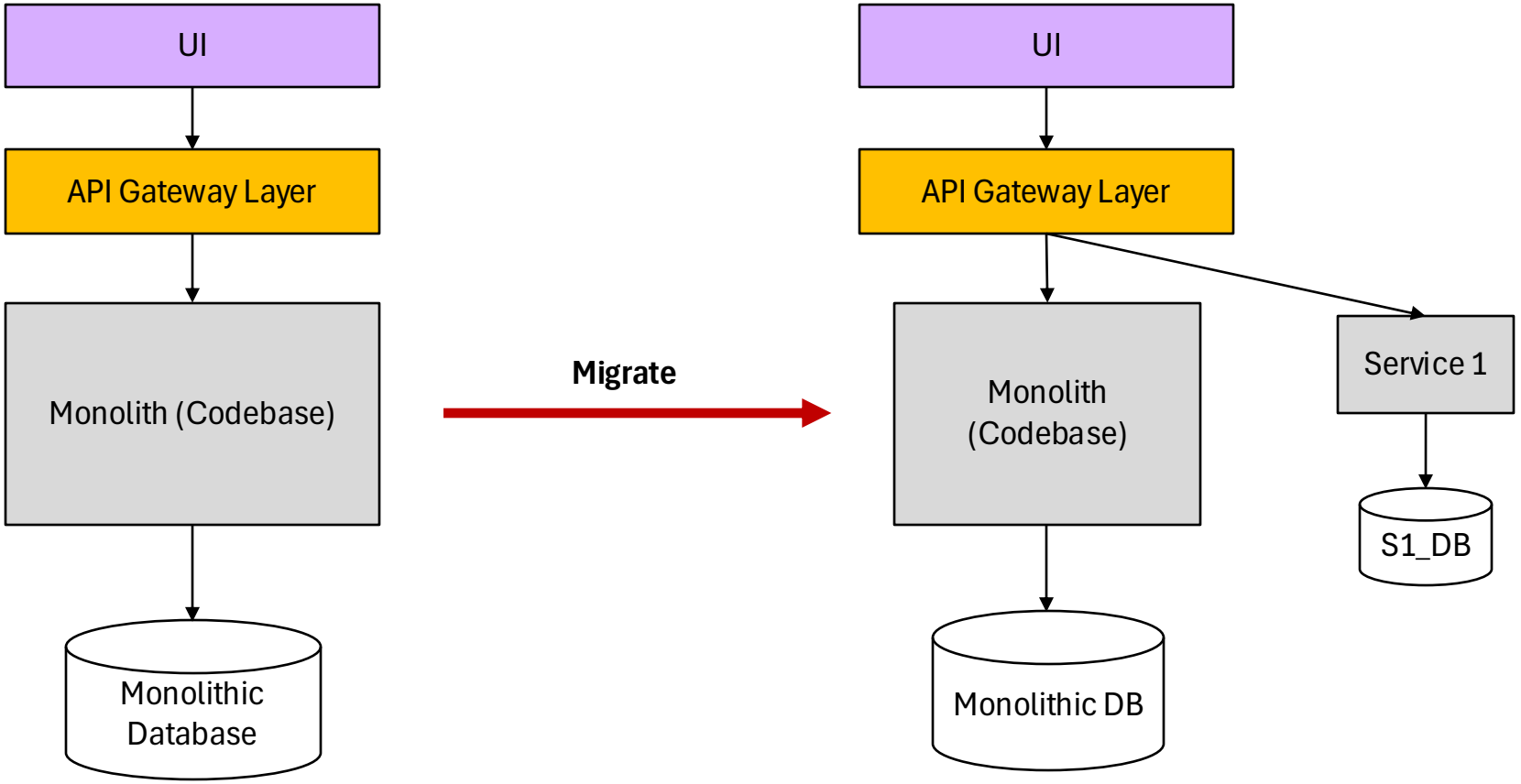
# Strangler Fig Pattern

**Step 1**. Add an intermediate layer (e.g., API layer) to intercept and redirect traffic



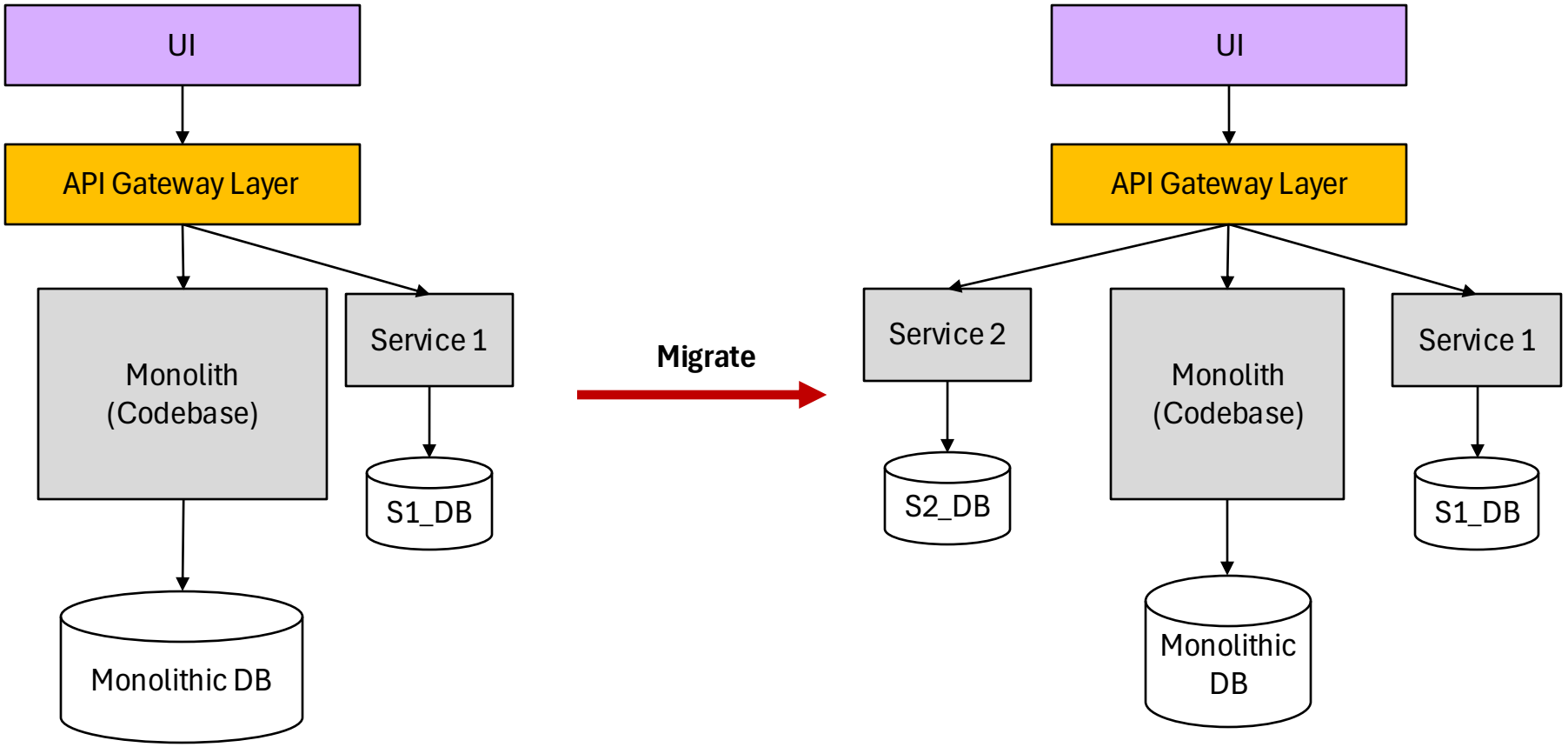
# Strangler Fig Pattern

**Step 2**. Extract the first microservice and redirect requests towards it



# Strangler Fig Pattern

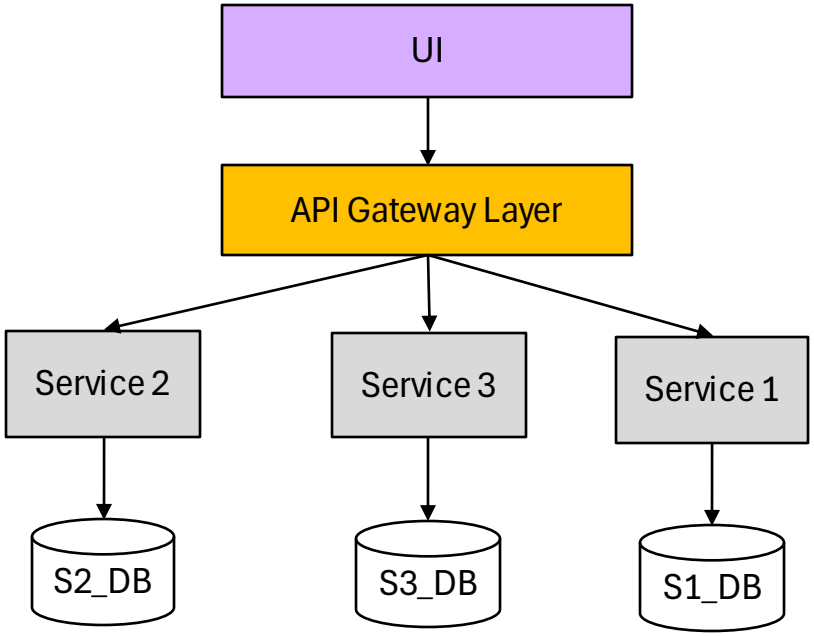
**Step 3**. Keep extracting microservices (most valuable, least risky first)



# Strangler Fig Pattern

**Notes**.

- This migration can be done **gradually** without affecting clients
- Much easier to split stateless components than stateful ones
- Only at the very end, the monolith can be retired

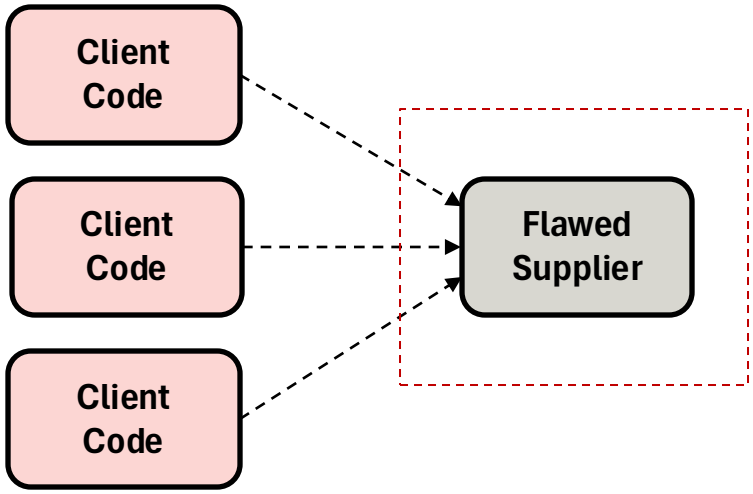


# Using Legacy Modernization Patterns to Replace a Legacy System*

*Source: Ian Cartwright, Rob Horn, and James Lewis (2024), Patterns of Legacy Displacement (https://martinfowler.com/articles/patterns-legacy-displacement/)

# Branch by Abstraction Pattern*

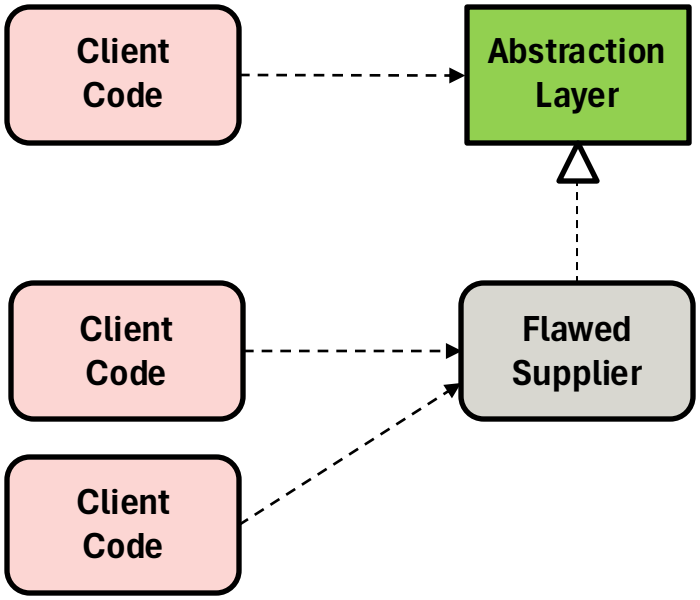
- **Branch by Abstraction**
  - It is a pattern for making a **large-scale change** to a software system in a **gradual way** that allows you to **release** the system **regularly** while the **change** is still **in progress**.




**Note**: Various parts of the software system are dependent on a module, library, or framework (e.g., **Flawed Supplier** in this example) that we wish to replace.

Source: Martin Fowler (2024), "Branch By Abstraction" https://martinfowler.com/bliki/BranchByAbstraction.html

# Branch by Abstraction*

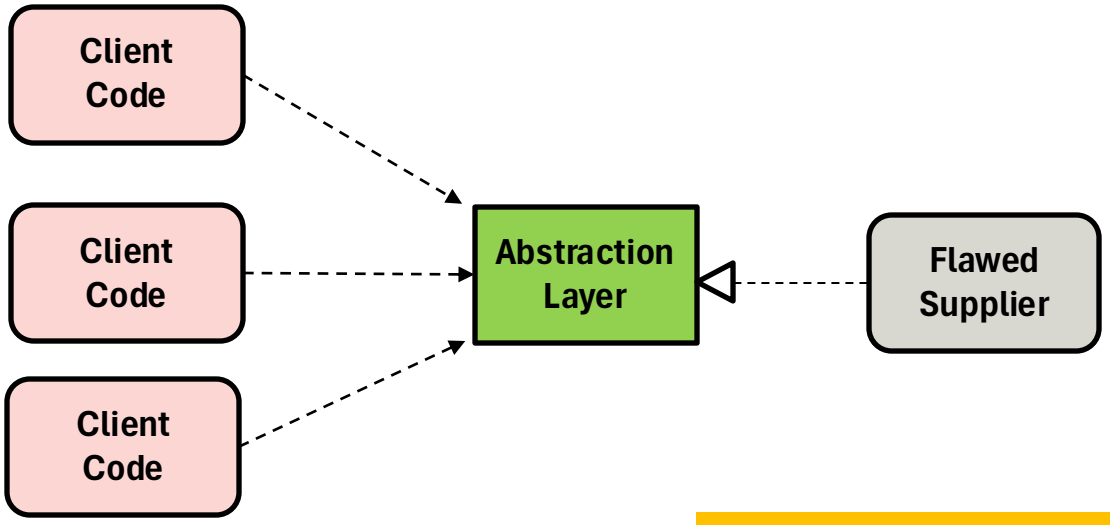



**(1)** Create an abstraction layer that captures the interaction between one section of the client code and the current supplier.

**(2)** Change that section of the client code to call the supplier entirely through this abstraction layer.

Source: Martin Fowler (2024), "Branch By Abstraction" https://martinfowler.com/bliki/BranchByAbstraction.html

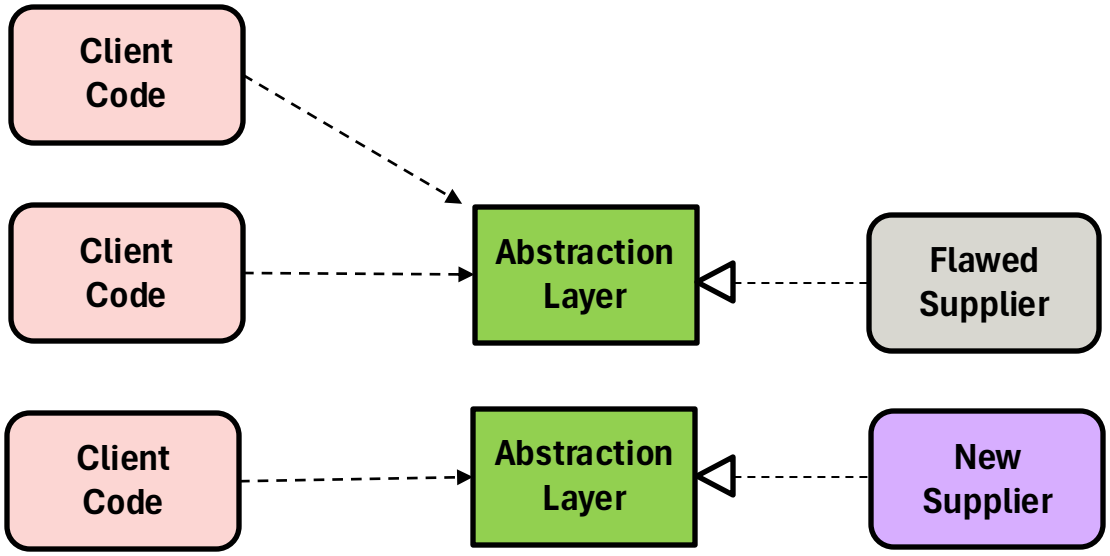
# Branch by Abstraction*




- Gradually move all client code over to use the abstraction layer until all interaction with the supplier is done by the abstraction layer.

Source: Martin Fowler (2024), "Branch By Abstraction" https://martinfowler.com/bliki/BranchByAbstraction.html

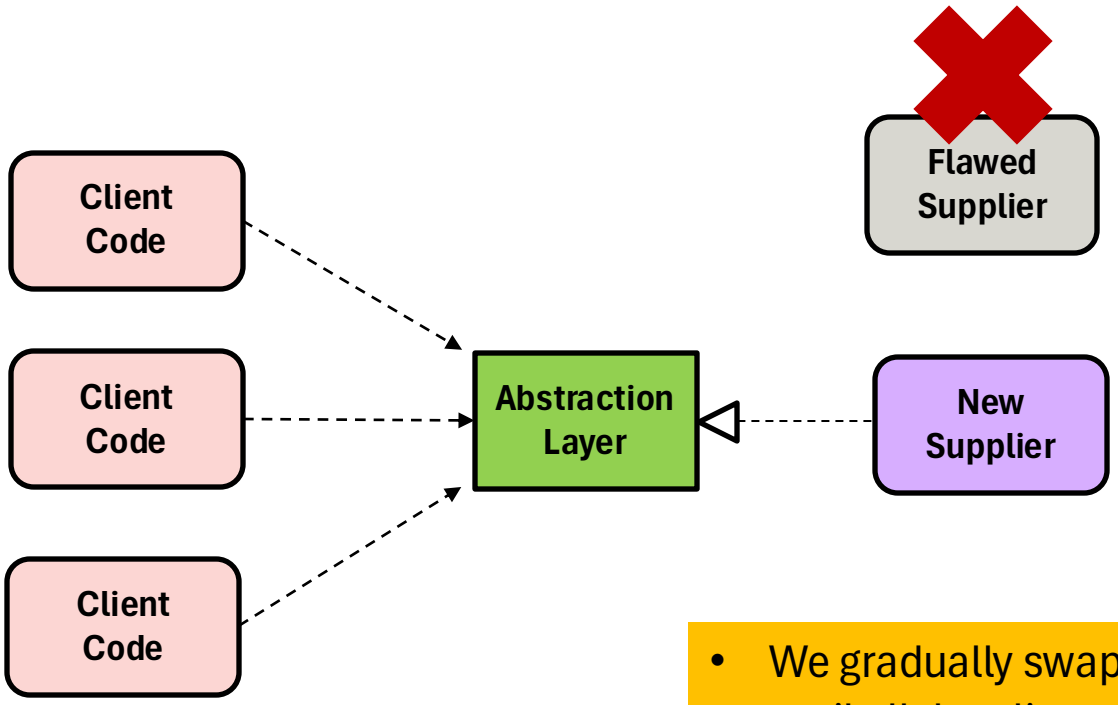
# Branch by Abstraction*




- We build a new supplier that implements the features required by one part of the client code using the same abstraction layer.
- Once we are ready, we switch that section of the client code to use the new supplier.

Source: Martin Fowler (2024), "Branch By Abstraction" https://martinfowler.com/bliki/BranchByAbstraction.html

# Branch by Abstraction*




- We gradually swap out the flawed supplier until all the client code uses the new supplier. Once the flawed supplier isn't needed, we can delete it.
- We may also choose to delete the abstraction layer once we no longer need it for migration.

Source: Martin Fowler (2024), “Branch By Abstraction” https://martinfowler.com/bliki/BranchByAbstraction.html

# A Case Study*

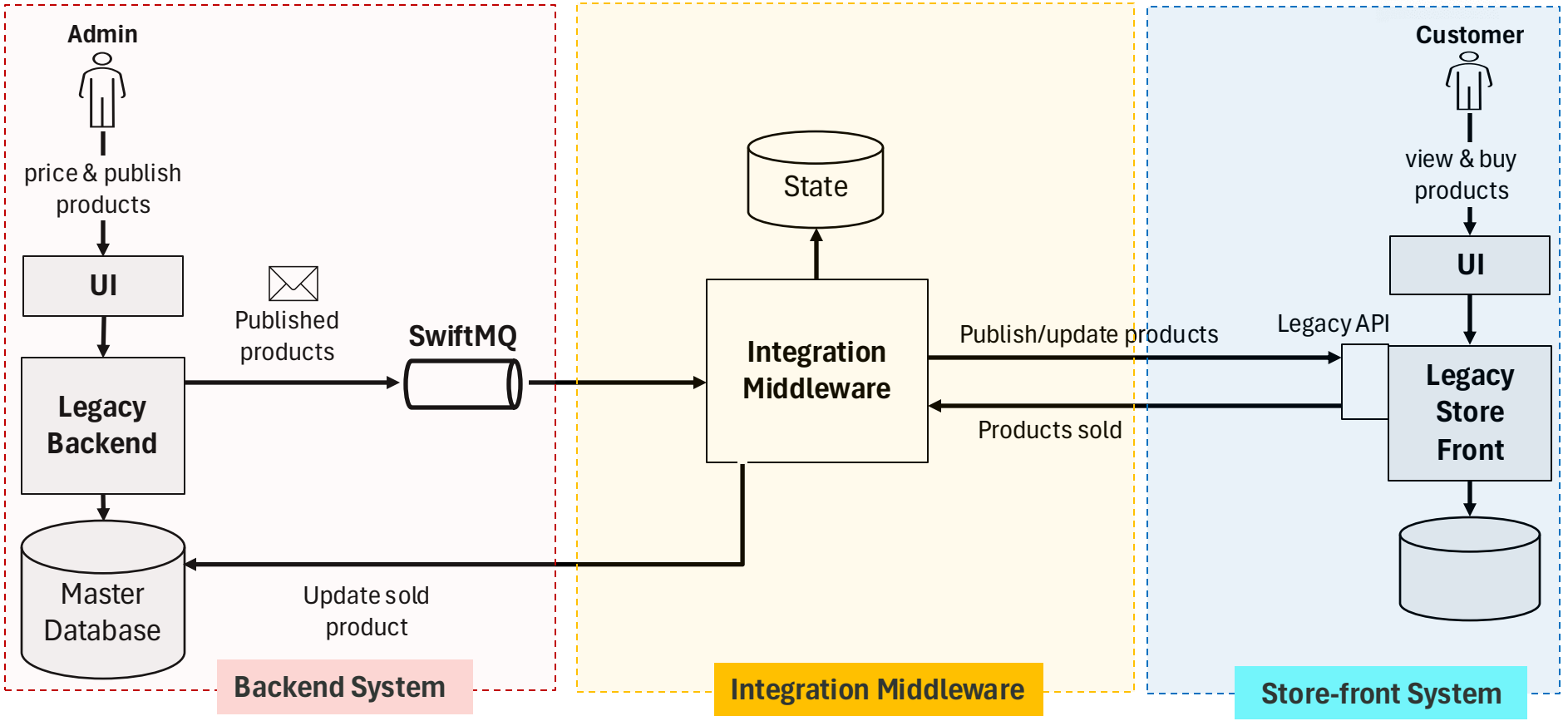
*Source: Ian Cartwright, Rob Horn, and James Lewis (2024), Patterns of Legacy Displacement (https://martinfowler.com/articles/patterns-legacy-displacement/)

# A Legacy "Sale & Purchase System"*

- A system for selling high-value unique products worth tens of millions of pounds every day.
- It includes three systems
  - A Backend System
  - A Store-front System
  - Integration Middleware
    - It is used to integrate the Backend System and Store-front System

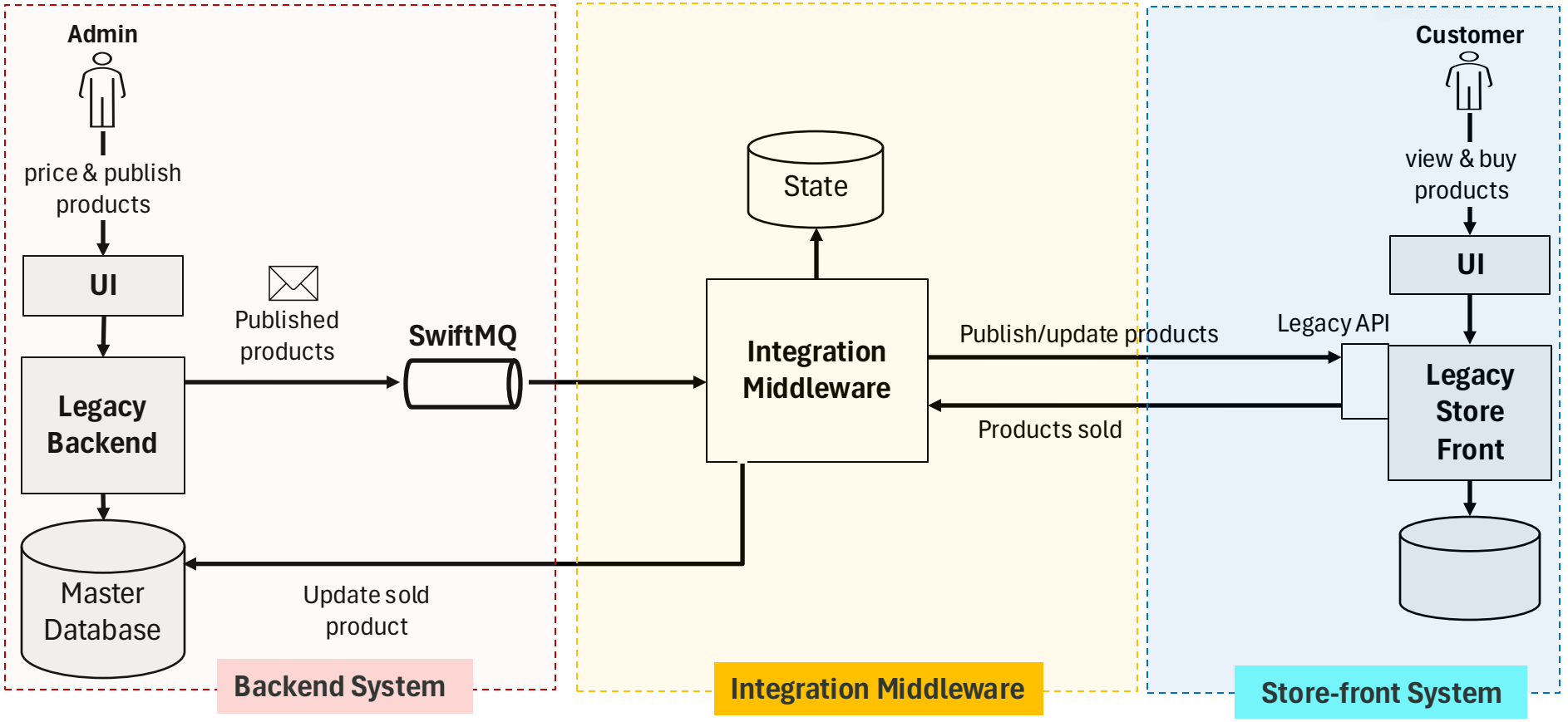
*Source: Ian Cartwright, Rob Horn, and James Lewis (2024), Patterns of Legacy Displacement (https://martinfowler.com/articles/patterns-legacy-displacement/)

# Architecture of the "Sale & Purchase System"*




*Source: Ian Cartwright, Rob Horn, and James Lewis (2024), Patterns of Legacy Displacement (https://martinfowler.com/articles/patterns-legacy-displacement/)

# Architecture of the "Sale & Purchase System" *

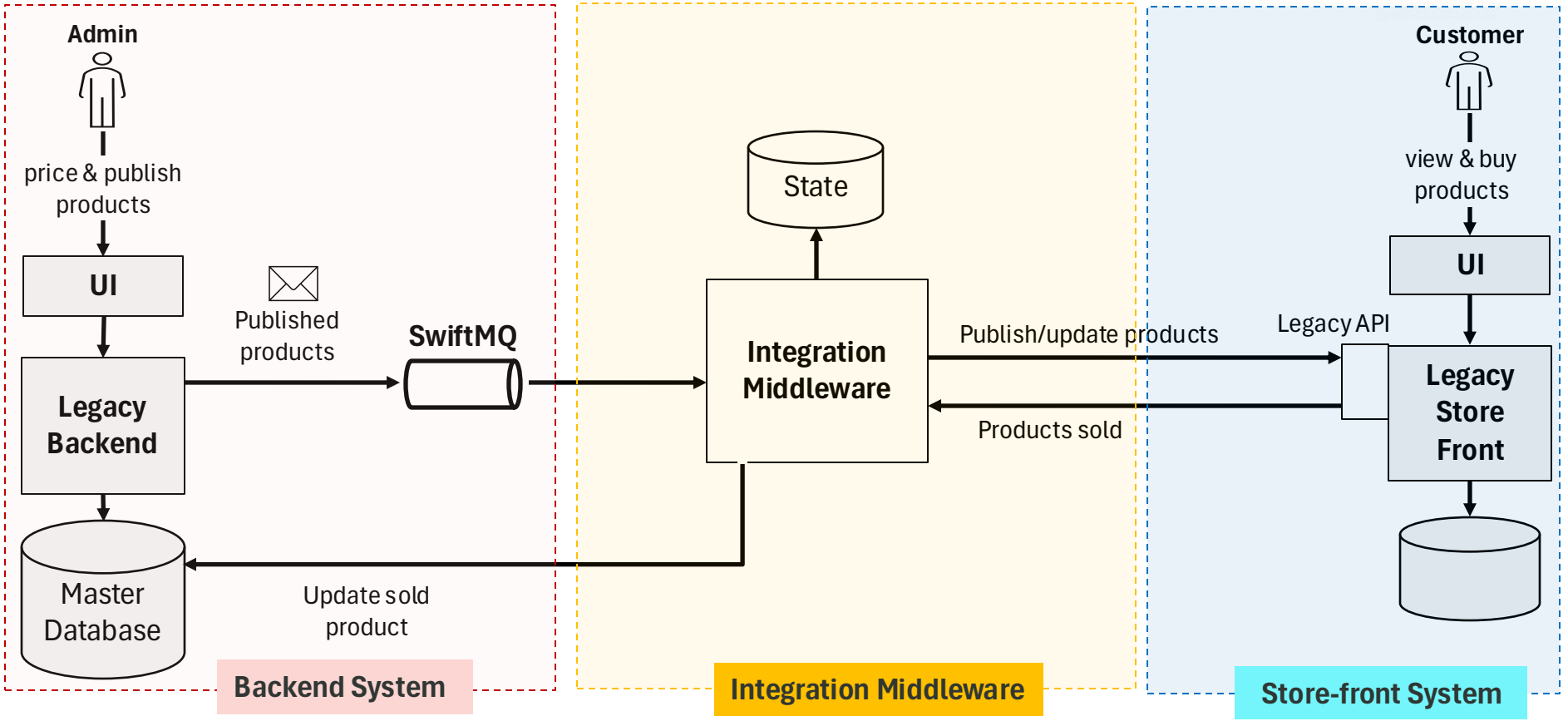



**Backend System (Legacy backend, an aging J2EE application)**

- Admins individually manage the pricing and publishing of products using screens within the legacy backend system.
- The system place "publish product" messages onto a **queue** provided by a very **old version of SwiftMQ**.

*Source: Ian Cartwright, Rob Horn, and James Lewis (2024), Patterns of Legacy Displacement (https://martinfowler.com/articles/patterns-legacy-displacement/)

# Architecture of the "Sale & Purchase System"*

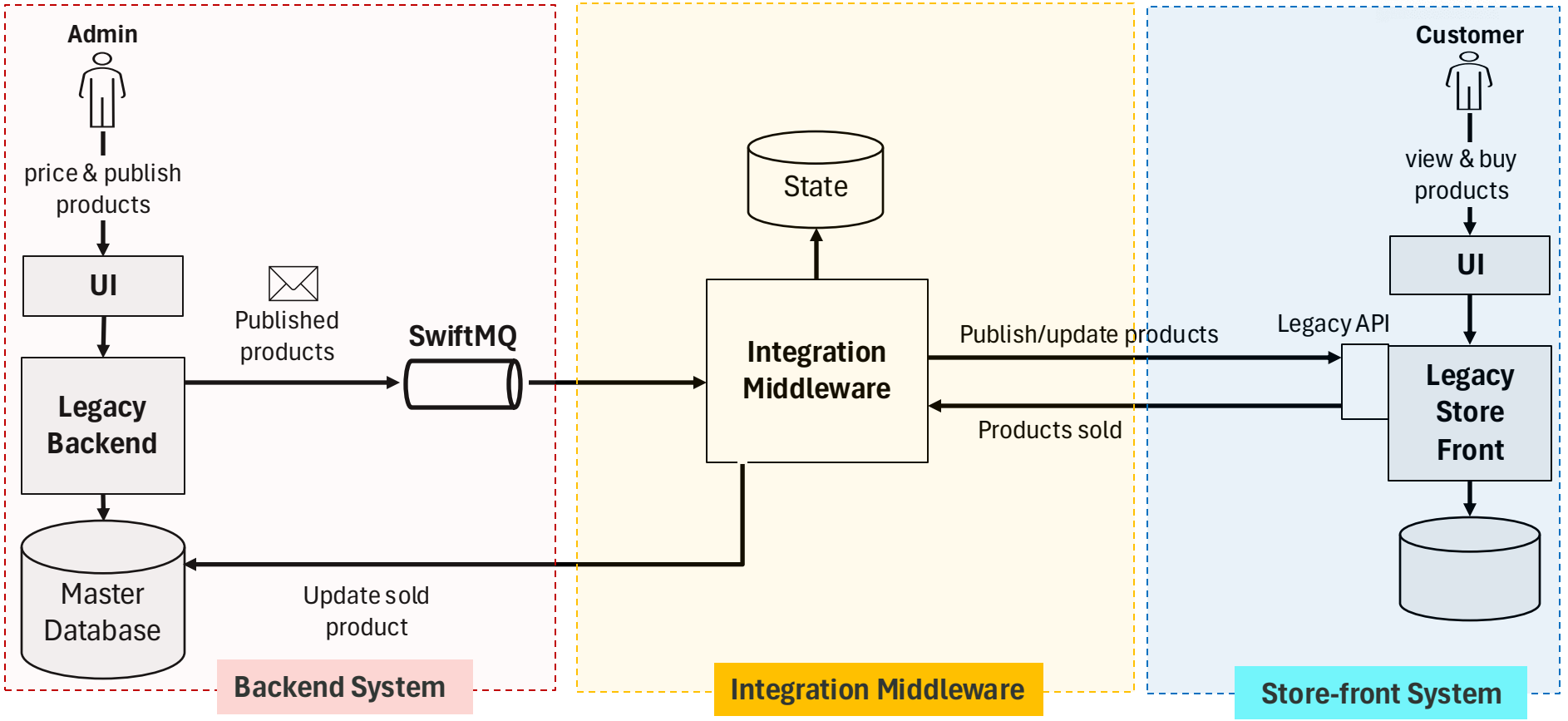



**Integration Middleware & Store-front System**

- The middleware would consume that message, create its own view of the state of the product and call a legacy **SOAP API** on the store-front system to publish it. Over time, the middleware would update the state of the product using the API to change how the product was made available to customers (e.g. change the product from "preview only" to "newly available" etc).

*Source: Ian Cartwright, Rob Horn, and James Lewis (2024), Patterns of Legacy Displacement (https://martinfowler.com/articles/patterns-legacy-displacement/)

# Architecture of the "Sale & Purchase System"*




**Integration Middleware & Store-front System**

- When a customer purchased a product, the legacy storefront would call an **API provided** by the integration middleware.
- The middleware would update its own state of the product and update the legacy system's master database with the sale information using **JDBC**.

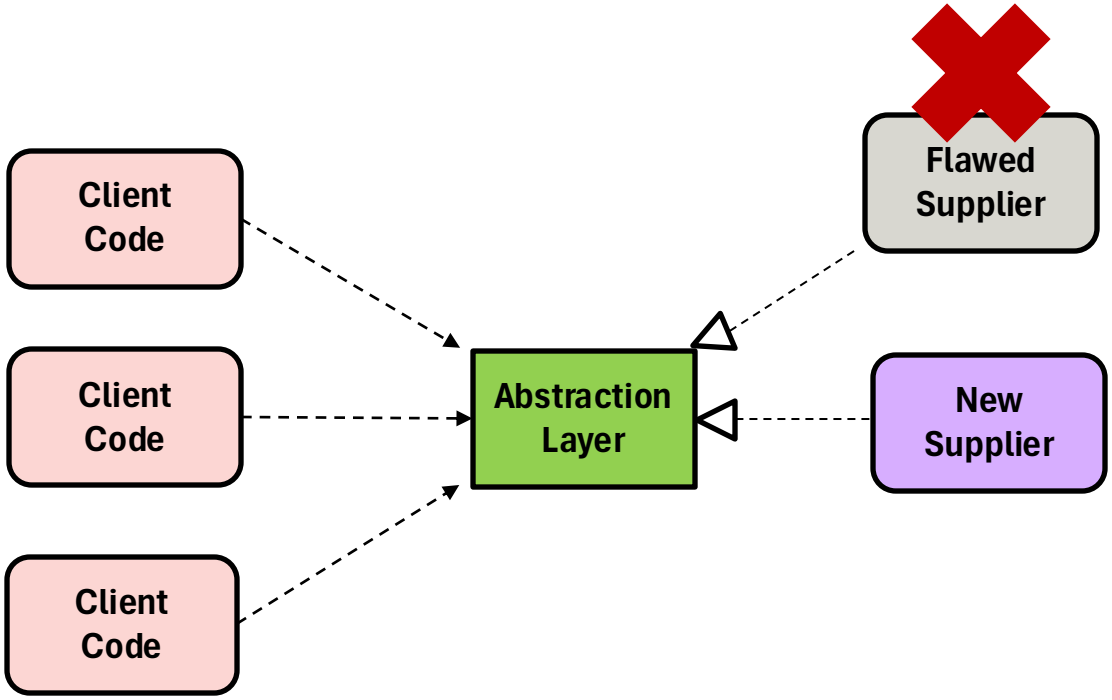
*Source: Ian Cartwright, Rob Horn, and James Lewis (2024), Patterns of Legacy Displacement (https://martinfowler.com/articles/patterns-legacy-displacement/)

# Challenge

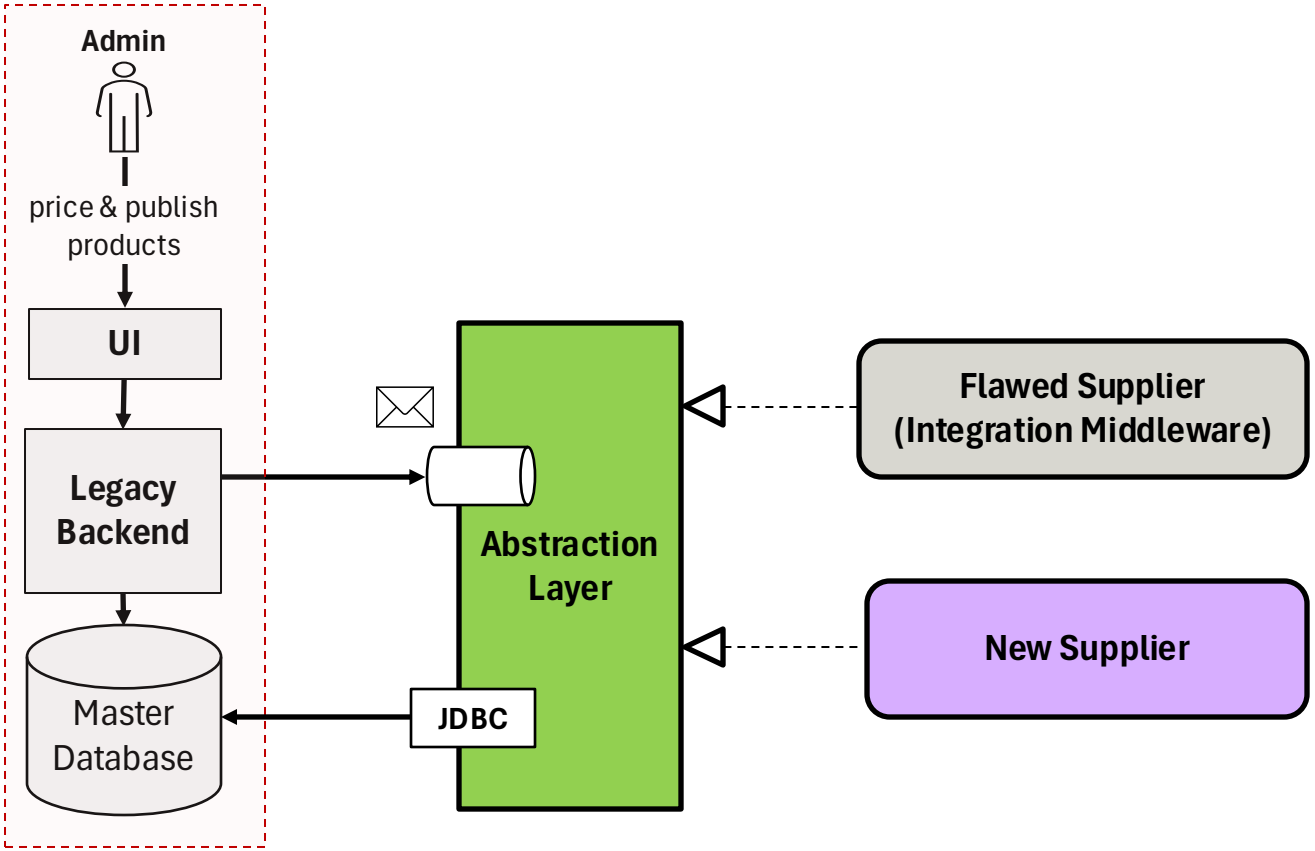
- How to replace **integration middleware** that is out of support, hard to change and very costly with a new supportable, flexible solution for the business without disrupting or putting at risk existing business operations.

*Source: Ian Cartwright, Rob Horn, and James Lewis (2024), Patterns of Legacy Displacement (https://martinfowler.com/articles/patterns-legacy-displacement/)

# Branch by Abstraction Pattern (Remember)



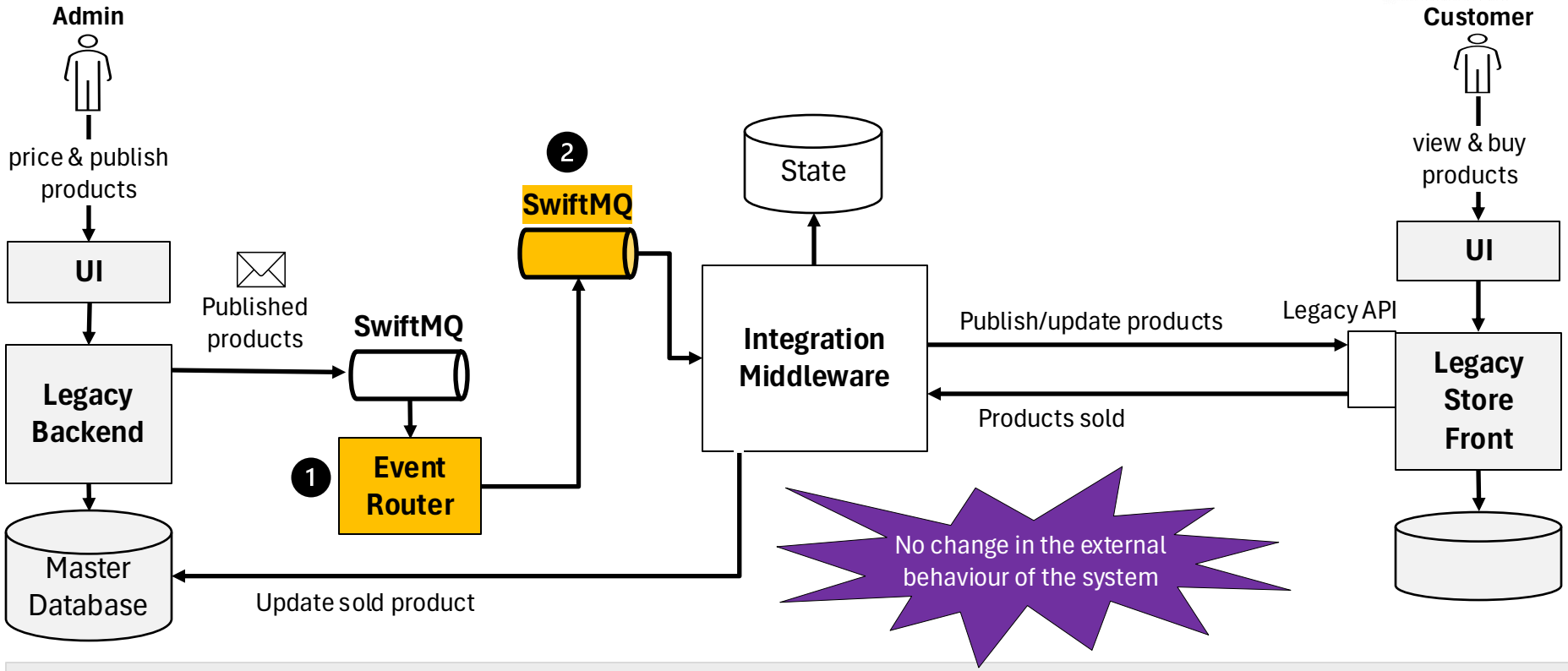
# Applying Branch by Abstraction Pattern to the Case System




The abstraction layer is the **queues** and the **JDBC**. By ensuring that the new implementation **adheres** to that abstraction layer, it could be swapped for the "flawed supplier" without impacting the business operations.

*Source: Ian Cartwright, Rob Horn, and James Lewis (2024), Patterns of Legacy Displacement (https://martinfowler.com/articles/patterns-legacy-displacement/)

# New Architecture of the "Sale & Purchase System" *

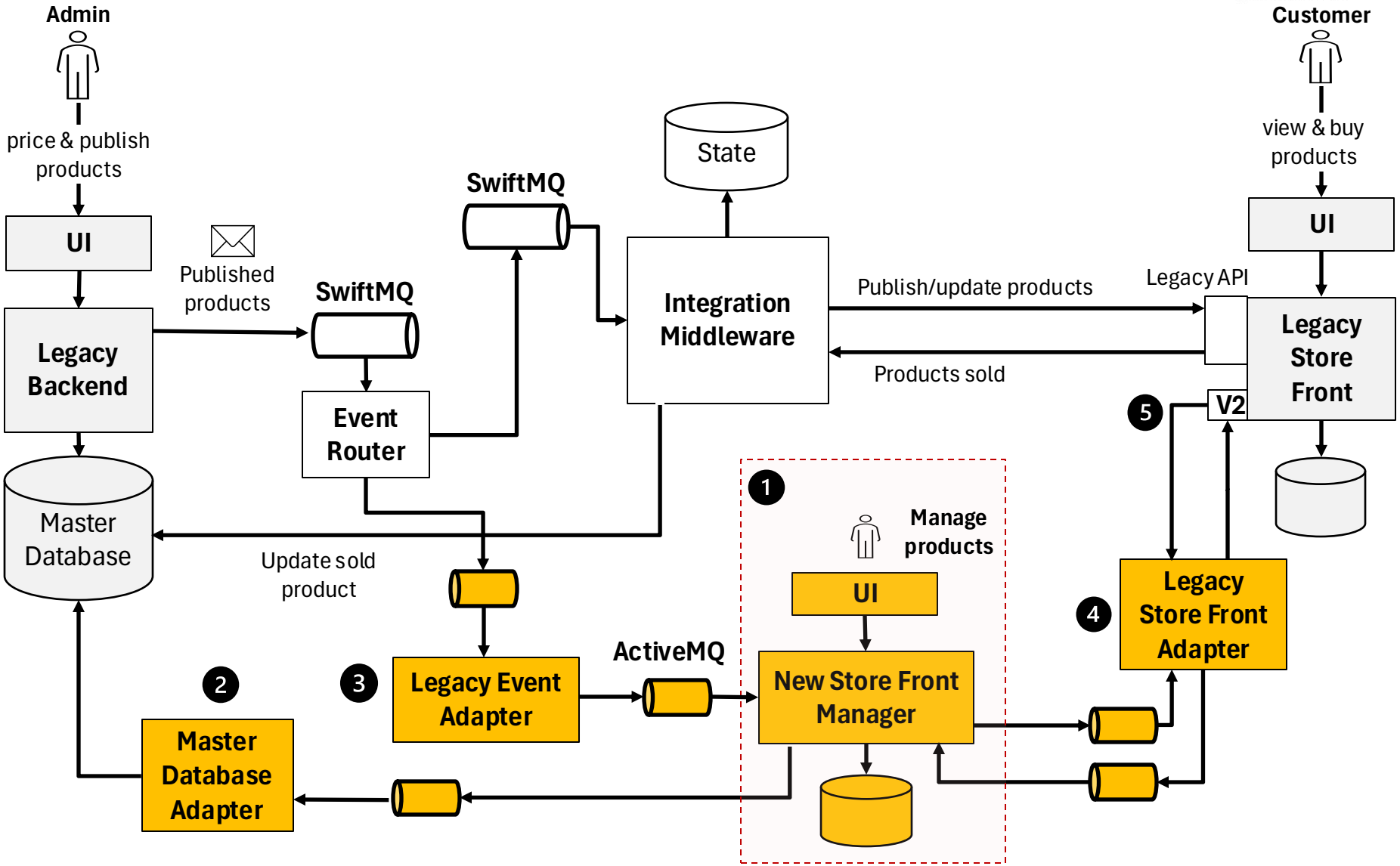



**Step 1:** Adding an **Event Router** via refactoring and alternative **SwiftMQ queue**

- When a product is published by a user the legacy backend system still places a publish message onto a **(1) SwiftMQ queue**. Instead of the integration middleware consuming it, the Event Router now consumes the message from that queue and enqueues it, unchanged, onto **(2)** an **alternative SwiftMQ** queue.
- The integration middleware consumes the message from this alternative queue, a change that was possible via a trivial **configuration setting**.

*Source: Ian Cartwright, Rob Horn, and James Lewis (2024), Patterns of Legacy Displacement (https://martinfowler.com/articles/patterns-legacy-displacement/)

# New Architecture of the "Sale & Purchase System"*




*Source: Ian Cartwright, Rob Horn, and James Lewis (2024), Patterns of Legacy Displacement (https://martinfowler.com/articles/patterns-legacy-displacement/)

# New Architecture of the "Sale & Purchase System"*

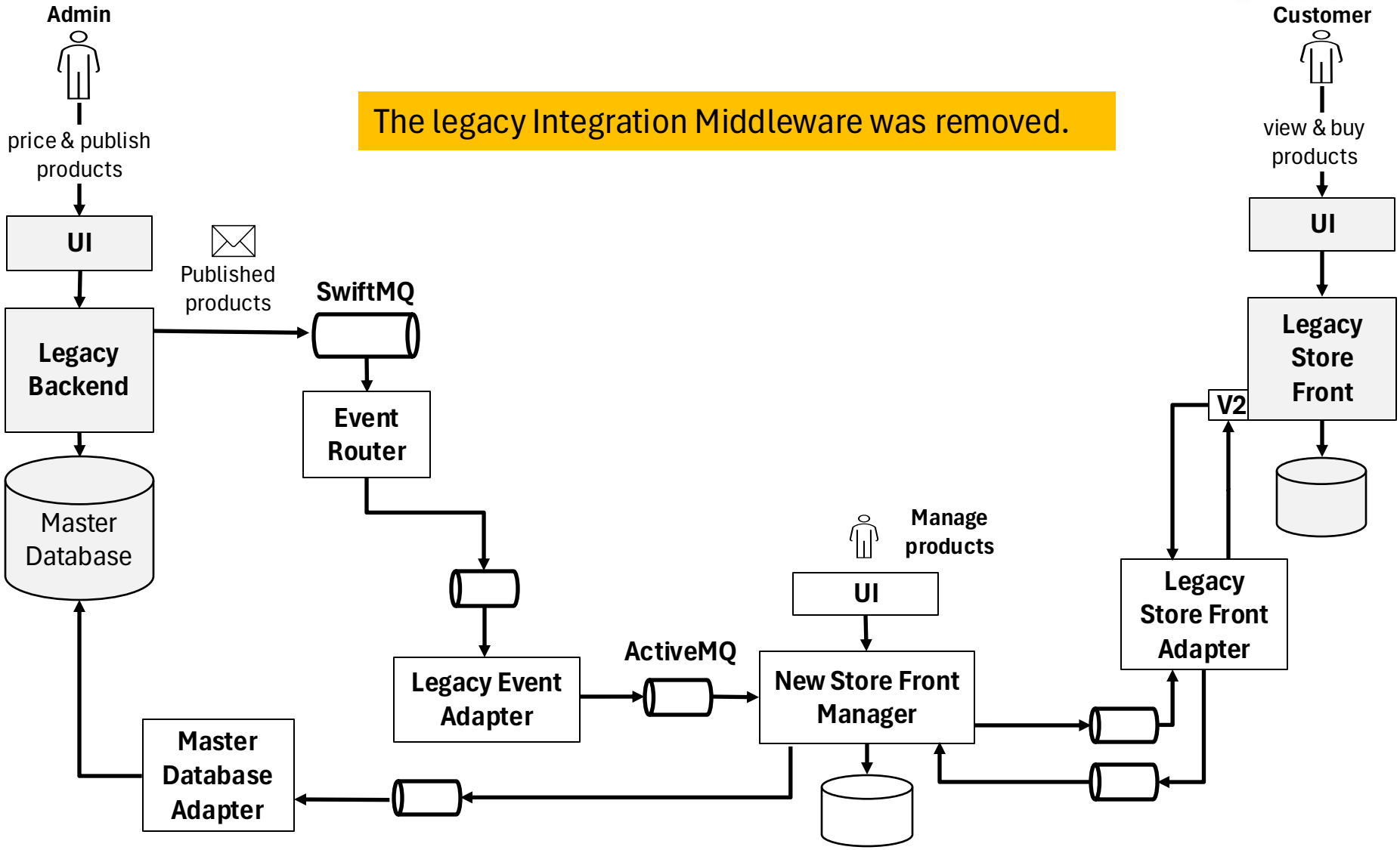
**Step 2:** Successfully deliver the parts: building out the functionality, maintaining the contract

- Add the **new Store Front Manager (1)**
- Add the **Master Database Adapter (2)** implementing the **Legacy Mimic pattern** to update the Master Database with sales information received from the Store Front.
- As the Event Router did not transform messages, a **Legacy Event Adapter (3)** (Message Translator) is created to transform messages into a new format (ActiveMQ), not exposing the legacy world to the new, and aligning to the principles of the new architecture.
- The **Legacy Storefront Adapter (4)** is added between the new **Store Front Manager (1)** and the Legacy Store Front to isolate the new implementation from future changes that would be coming when the Legacy Store Front is replaced.
- A **new API (5)** is introduced on the Legacy Store Front that the new Store Front Manager is to use.

**Legacy Mimic pattern:** New system interacts with the legacy system in such a way that the old system is not aware of any changes.

*Source: Ian Cartwright, Rob Horn, and James Lewis (2024), Patterns of Legacy Displacement (https://martinfowler.com/articles/patterns-legacy-displacement/)

# Final Architecture of the "Sale & Purchase System" *

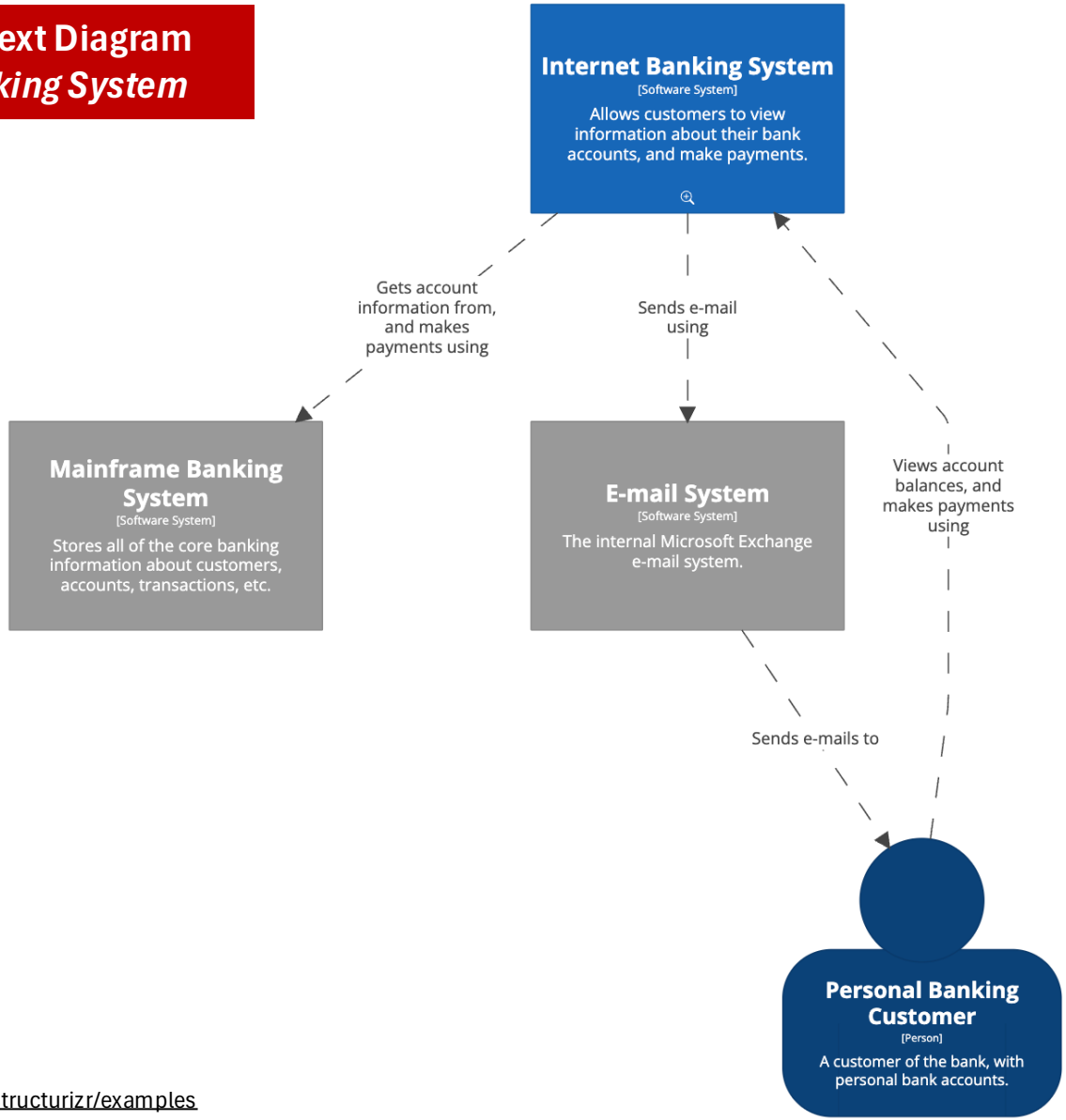



The legacy Integration Middleware was removed.

*Source: Ian Cartwright, Rob Horn, and James Lewis (2024), Patterns of Legacy Displacement (https://martinfowler.com/articles/patterns-legacy-displacement/)

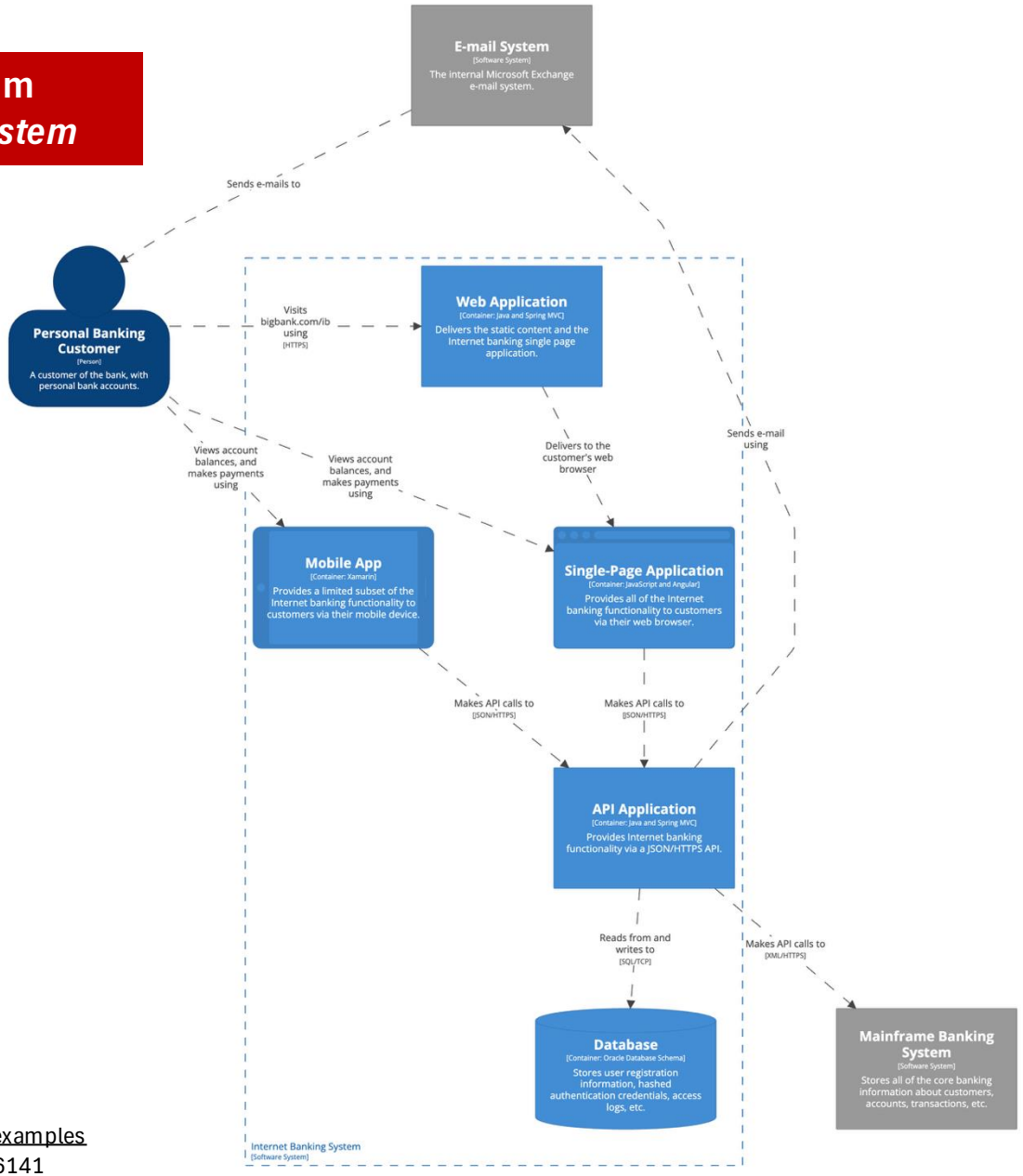
# Dynamic and Deployment Diagrams with C4 Model

# System Context Diagram
## *Internet Banking System*



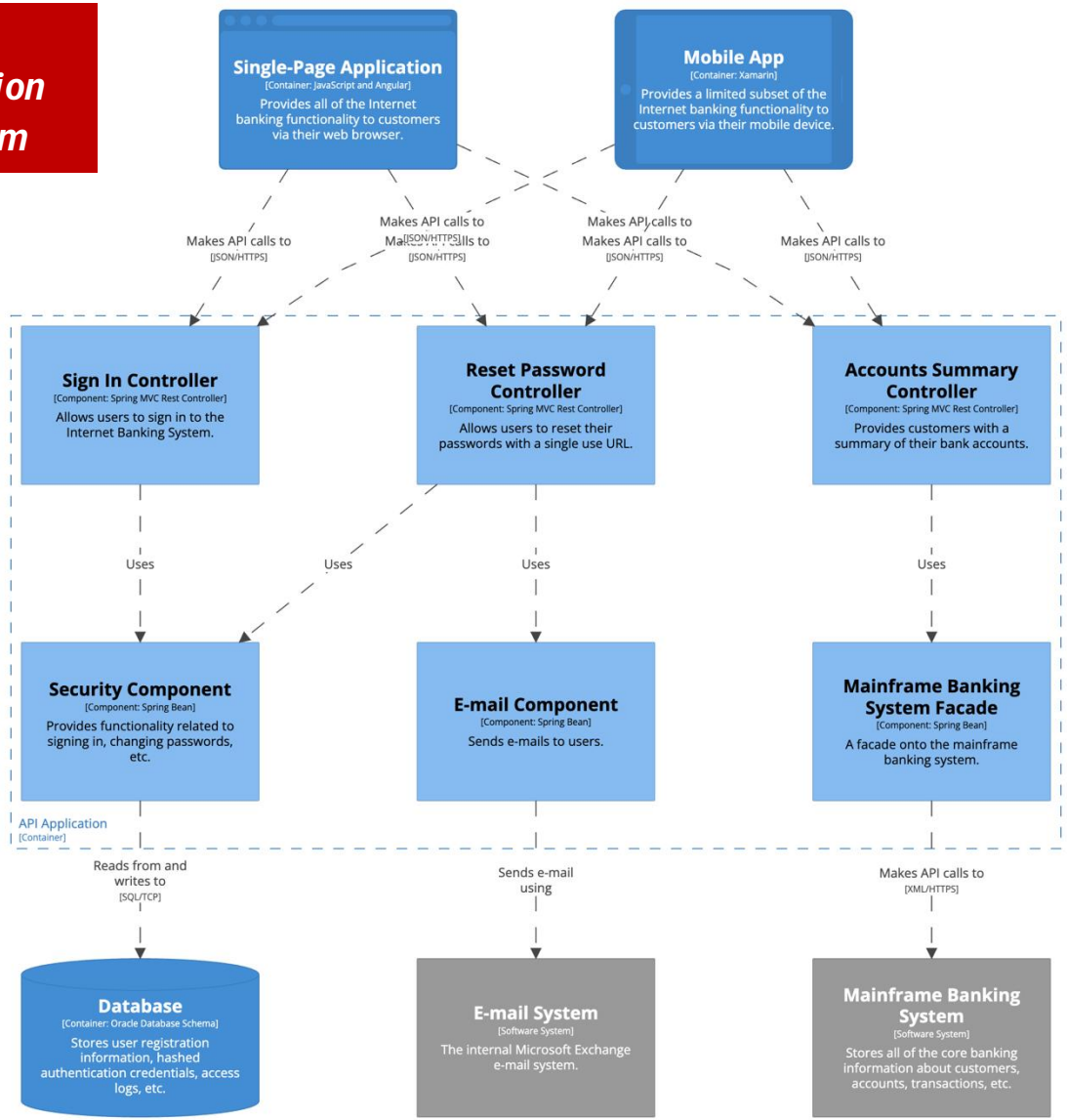

Source: https://github.com/structurizr/examples
Source:https://structurizr.com/share/36141

# Container Diagram
## *Internet Banking System*




Source: https://github.com/structurizr/examples
Source:https://structurizr.com/share/36141

# Component Diagram Container: *API Application Internet Banking System*

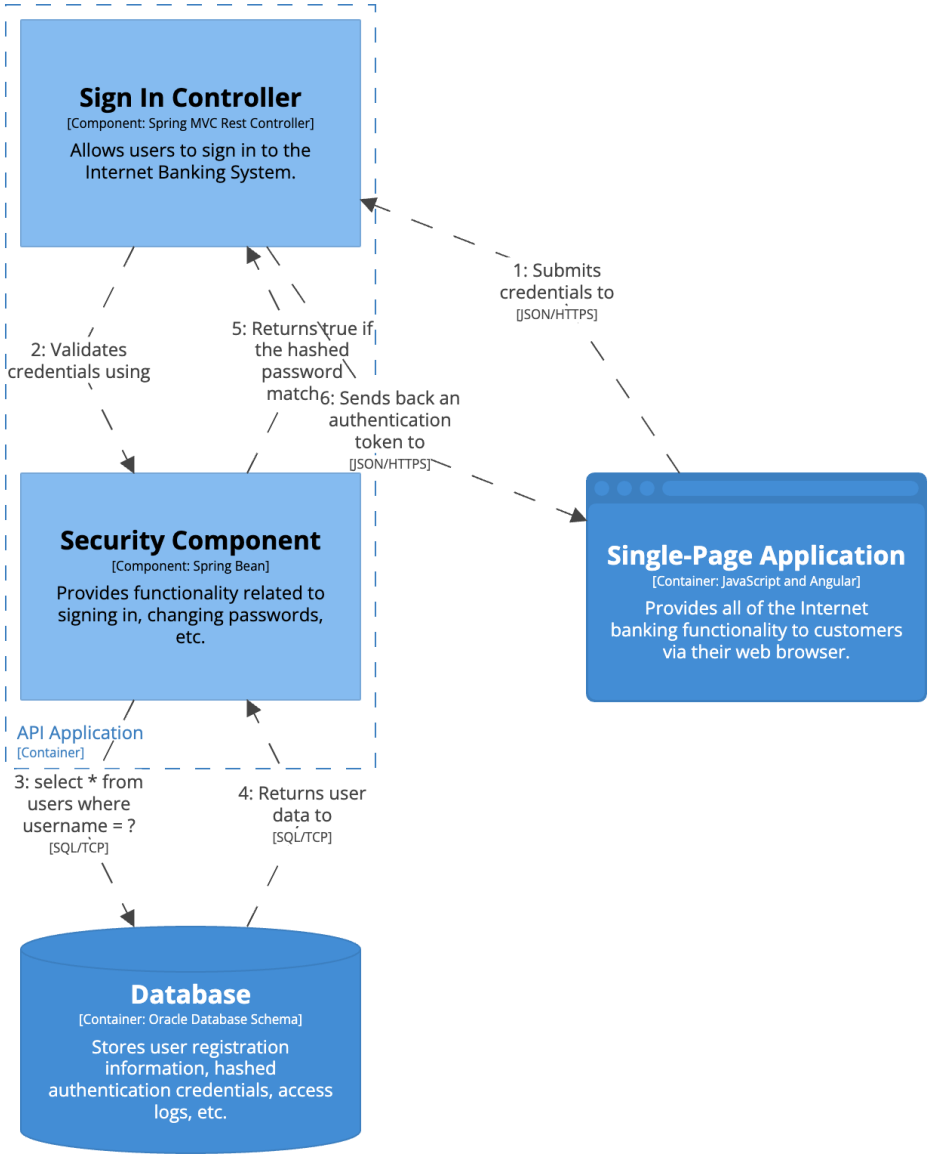



Source: https://github.com/structurizr/examples
Source:https://structurizr.com/share/36141

# Dynamic Diagram

- A **dynamic diagram** can be useful when you want to show how elements in the static model collaborate at runtime to implement a user story, use case, feature, etc.
- It is similar to a **UML sequence diagram** although it allows a free-form arrangement of diagram elements with numbered interactions to indicate ordering.

# Dynamic Diagram SignIn feature




Source: https://github.com/structurizr/examples
Source:https://structurizr.com/share/36141

RMIT UNIVERSITY

# Dynamic Diagram SignIn feature

The scope is a container (i.e., **apiApplication**). So, we can have **external systems**, **users**, **containers** and **components** in this dynamic diagram

```
dynamic apiApplication "SignIn" {
    singlePageApplication -> signinController "Submits credentials to"
    signinController -> securityComponent "Validates credentials using"
    securityComponent -> database "select * from users where username = ?"
    database -> securityComponent "Returns user data to"
    securityComponent -> signinController "Returns true if the hashed password matches"
    signinController -> singlePageApplication "Sends back an authentication token to"
    autoLayout
    description "Summarises how the sign in feature works in the single-page application."
}
```

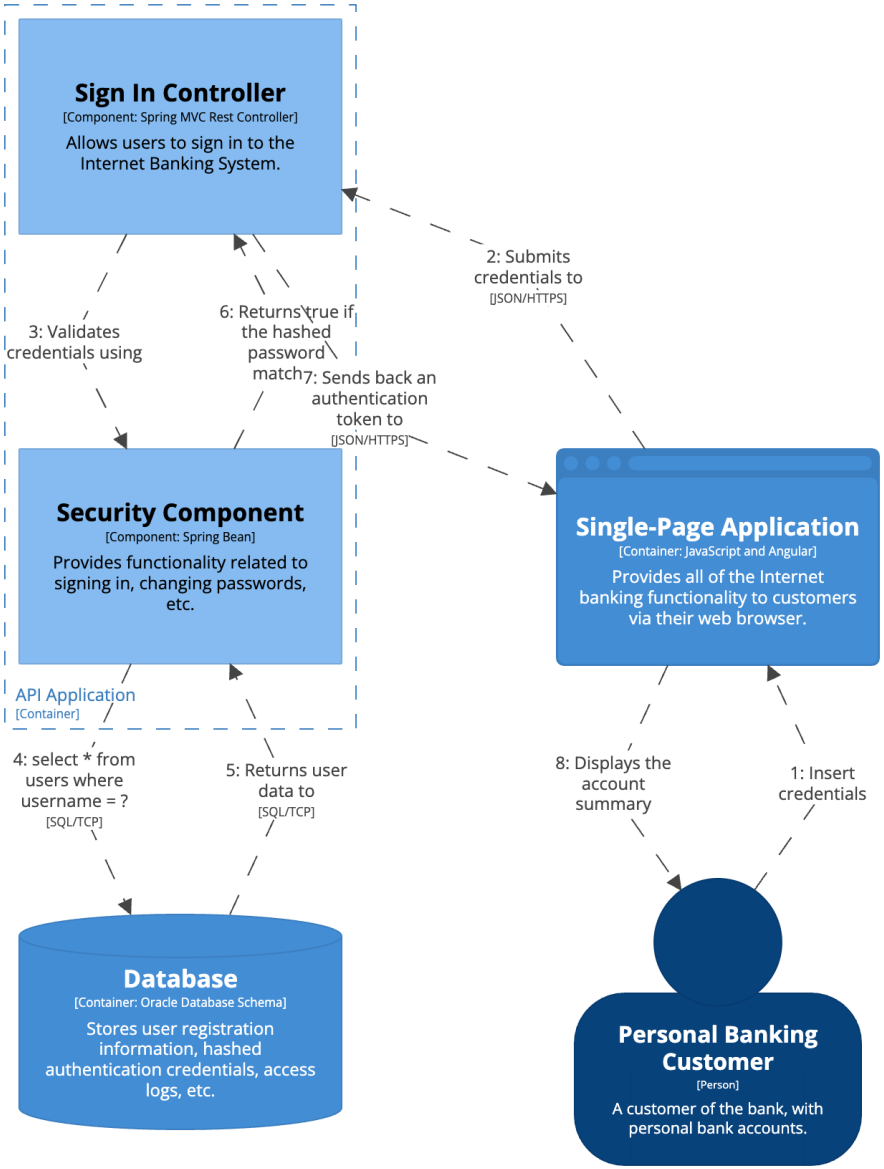
This code should be placed in the views part

```
workspace {

    model {

    }

    views {

    }
}
```

Source: https://github.com/structurizr/examples
Source:https://structurizr.com/share/36141

# Dynamic Diagram SignIn feature

Users (customers) are also added to the dynamic diagram.




Source: https://github.com/structurizr/examples
Source:https://structurizr.com/share/36141

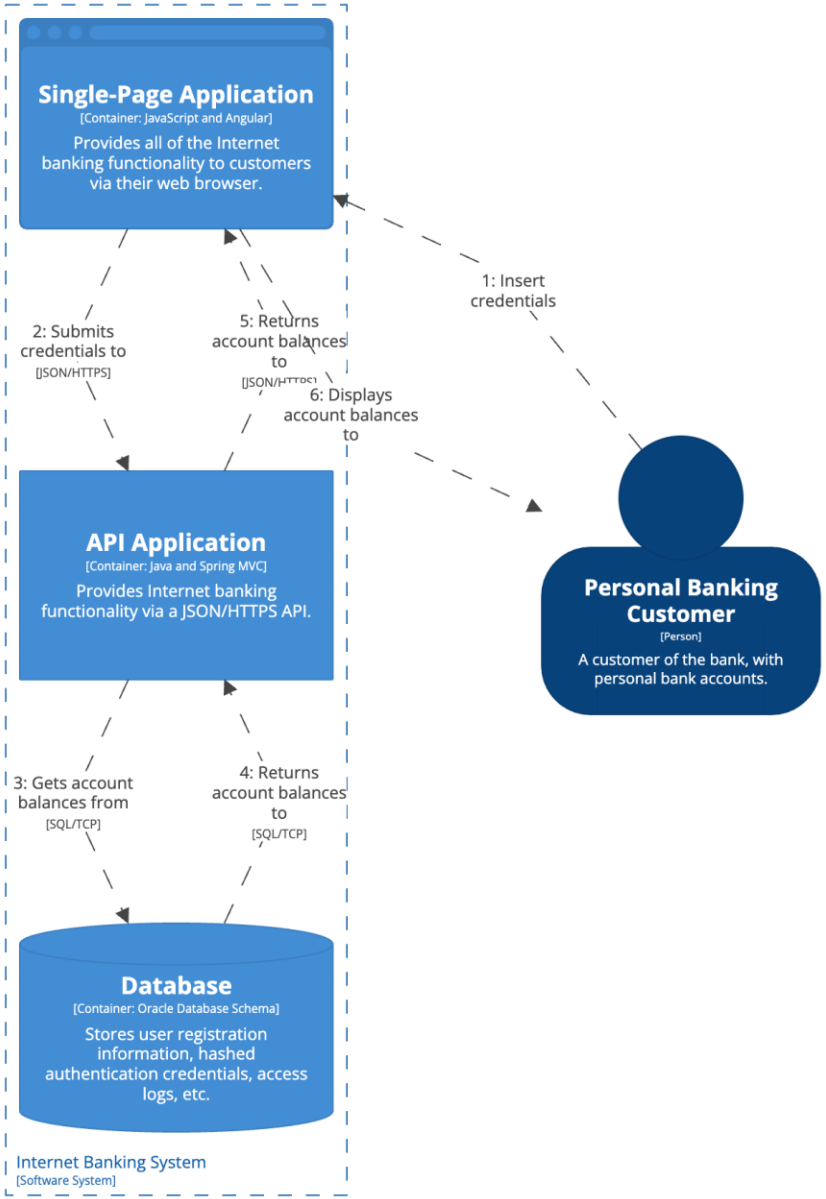
# Dynamic Diagram SignIn feature

Users (customers) are also added to the dynamic diagram.

```
dynamic apiApplication "SignIn" {
    customer -> singlePageApplication "Insert credentials"
    singlePageApplication -> signinController "Submits credentials to"
    signinController -> securityComponent "Validates credentials using"
    securityComponent -> database "select * from users where username = ?"
    database -> securityComponent "Returns user data to"
    securityComponent -> signinController "Returns true if the hashed password matches"
    signinController -> singlePageApplication "Sends back an authentication token to"
    singlePageApplication -> customer "Displays the account summary"
    autoLayout
    description "Summarises how the sign in feature works in the single-page application."
}
```

Source: https://github.com/structurizr/examples
Source:https://structurizr.com/share/36141

# Dynamic Diagram
# SignIn feature (high-level)




Source: https://github.com/structurizr/examples
Source:https://structurizr.com/share/36141

RMIT UNIVERSITY

## Dynamic Diagram
## SignIn feature (high-level)

The scope is a software system (i.e., **internetBankingSystem**). So, we can have **ONLY external systems**, **users**, and **containers** in this dynamic diagram

```
dynamic internetBankingSystem "HighLevelSignIn" {
    customer -> singlePageApplication "Insert credentials"
    singlePageApplication -> apiApplication "Submits credentials to"
    apiApplication -> database "Gets account balances from"
    database -> apiApplication "Returns account balances to"
    apiApplication -> singlePageApplication "Returns account balances to"
    singlePageApplication -> customer "Displays account balances to"
    autoLayout
    description "At a high level, Summarises how the sign in feature works in the Internet Banking System."
}
```

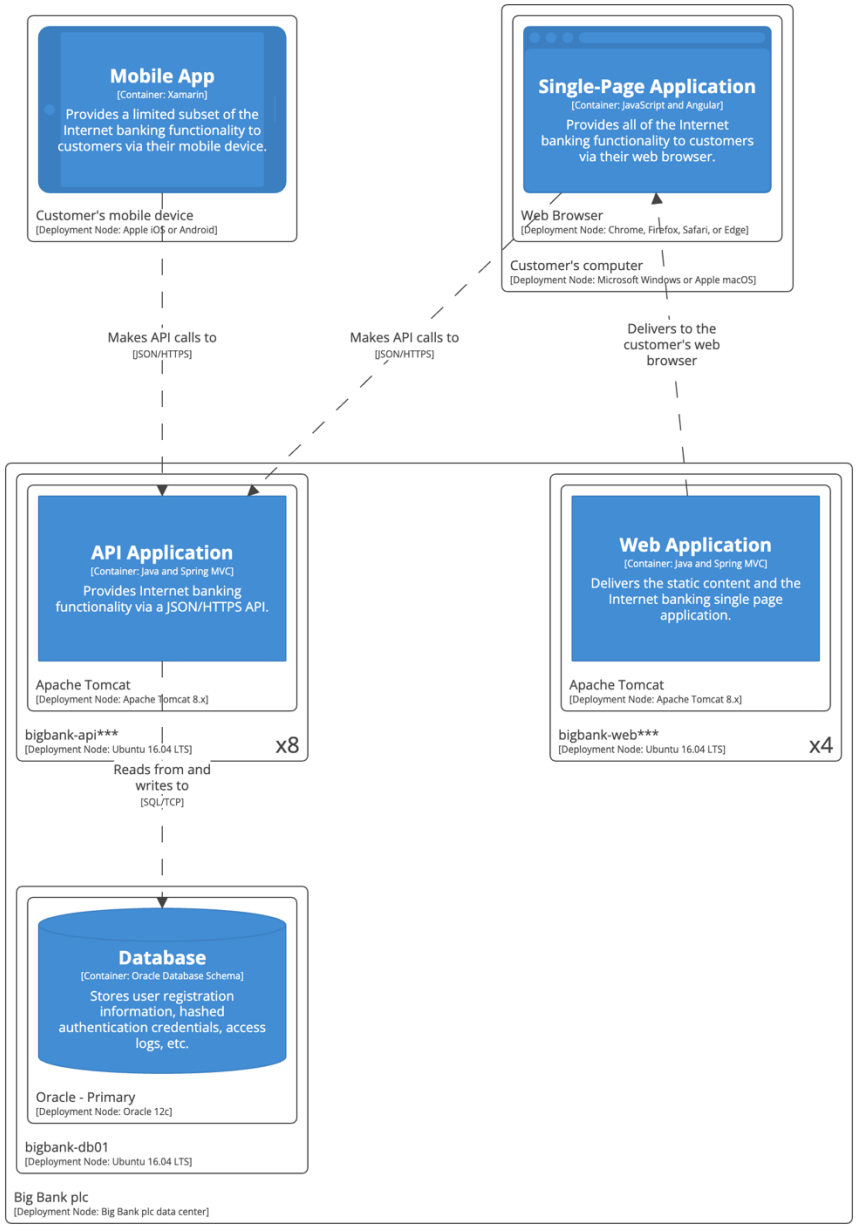
**Components can't be added to a dynamic view when the scope is a software system**

Source: https://github.com/structurizr/examples
Source:https://structurizr.com/share/36141

# Deployment Diagram

- A deployment diagram allows you to illustrate how instances of software systems and/or containers in the static model are deployed on to the infrastructure within a given deployment environment (e.g. production, staging, development, etc).
- A **deployment node** represents where an instance of a **software system/container** is running;
  - Physical infrastructure (e.g. a physical server or device)
  - Virtualised infrastructure (e.g. IaaS, PaaS, a virtual machine)
  - Containerised infrastructure (e.g. a Docker container),
  - An execution environment (e.g. a database server, Java EE web/application server, Microsoft IIS), etc.
- **Deployment nodes can be nested.**

# Deployment Diagram
## *Internet Banking System*




Source: https://github.com/structurizr/examples
Source:https://structurizr.com/share/36141

# Deployment Diagram
## *Internet Banking System*


Source: https://github.com/structurizr/examples
Source:https://structurizr.com/share/36141

# Deployment Diagram
## *Internet Banking System*

```
deploymentEnvironment "Live" {
    deploymentNode name: "Customer's mobile device" description: "" technology: "Apple iOS or Android" {
        liveMobileAppInstance = containerInstance identifier: mobileApp
    }
    deploymentNode name: "Customer's computer" description: "" technology: "Microsoft Windows or Apple macOS" {
        deploymentNode name: "Web Browser" description: "" technology: "Chrome, Firefox, Safari, or Edge" {
            liveSinglePageApplicationInstance = containerInstance identifier: singlePageApplication
        }
    }
    deploymentNode name: "Big Bank plc" description: "" technology: "Big Bank plc Data Center" {
        deploymentNode name: "bigbank-web***" description: "" technology: "Ubuntu 16.04 LTS" tags: "" instances: 4 {
            deploymentNode name: "Apache Tomcat" description: "" technology: "Apache Tomcat 8.x" {
                liveWebApplicationInstance = containerInstance identifier: webApplication
            }
        }
        deploymentNode name: "bigbank-api***" description: "" technology: "Ubuntu 16.04 LTS" tags: "" instances: 8 {
            deploymentNode name: "Apache Tomcat" description: "" technology: "Apache Tomcat 8.x" {
                liveApiApplicationInstance = containerInstance identifier: apiApplication
            }
        }
        deploymentNode name: "bigbank-db01" description: "" technology: "Ubuntu 16.04 LTS" {
            primaryDatabaseServer = deploymentNode name: "Oracle - Primary" description: "" technology: "Oracle 12c" {
                livePrimaryDatabaseInstance = containerInstance identifier: database
            }
        }
    }
}
```

This code should be placed in the model part

```
workspace {

    model {
    }

    views {
    }
}
```

Source: https://github.com/structurizr/examples
Source:https://structurizr.com/share/36141

# Deployment Diagram
## *Internet Banking System*

Deployment nodes

```
deploymentEnvironment "Live" {
    deploymentNode name: "Customer's mobile device" description: "" technology: "Apple iOS or Android" {
        liveMobileAppInstance = containerInstance identifier: mobileApp
    }
    deploymentNode name: "Customer's computer" description: "" technology: "Microsoft Windows or Apple macOS" {
        deploymentNode name: "Web Browser" description: "" technology: "Chrome, Firefox, Safari, or Edge" {
            liveSinglePageApplicationInstance = containerInstance identifier: singlePageApplication
        }
    }
    deploymentNode name: "Big Bank plc" description: "" technology: "Big Bank plc Data Center" {
        deploymentNode name: "bigbank-web***" description: "" technology: "Ubuntu 16.04 LTS" tags: "" instances: 4 {
            deploymentNode name: "Apache Tomcat" description: "" technology: "Apache Tomcat 8.x" {
                liveWebApplicationInstance = containerInstance identifier: webApplication
            }
        }
        deploymentNode name: "bigbank-api***" description: "" technology: "Ubuntu 16.04 LTS" tags: "" instances: 8 {
            deploymentNode name: "Apache Tomcat" description: "" technology: "Apache Tomcat 8.x" {
                liveApiApplicationInstance = containerInstance identifier: apiApplication
            }
        }
        deploymentNode name: "bigbank-db01" description: "" technology: "Ubuntu 16.04 LTS" {
            primaryDatabaseServer = deploymentNode name: "Oracle - Primary" description: "" technology: "Oracle 12c" {
                livePrimaryDatabaseInstance = containerInstance identifier: database
            }
        }
    }
}
```

Nested deployment nodes

This code should be placed in the model part

```
workspace {

    model {

    }

    views {

    }
}
```

Source: https://github.com/structurizr/examples
Source:https://structurizr.com/share/36141

# Deployment Diagram
## *Internet Banking System*

```
deployment internetBankingSystem "Live" "LiveDeployment" {
    include *
    autoLayout
    description "An example live deployment scenario for the Internet Banking System."
}
```

This code should be placed in the views part

```
workspace {

    model {

    }

    views {

    }
}
```

Source: https://github.com/structurizr/examples
Source:https://structurizr.com/share/36141

# References


- https://c4model.com/
- https://github.com/structurizr/examples
- https://structurizr.com/share/36141
- Tremel, E. (2017): Six strategies for application deployment (https://thenewstack.io/deployment-strategies/)
- Pautasso, C., Software Architecture- Visual Lecture Notes, LeanPub, 2023 (https://leanpub.com/software-architecture/)
- Introduction to DevOps by Len Bass – URL: goo.gl/iMwdfg
- Len Bass, Ingo Weber, Liming Zhu, DevOps: A Software Architect's Perspective, 2015
- Martin Fowler (2024), "Branch By Abstraction" https://martinfowler.com/bliki/BranchByAbstraction.html
- Ian Cartwright, Rob Horn, and James Lewis (2024), Patterns of Legacy Displacement (https://martinfowler.com/articles/patterns-legacy-displacement/)
- The University of Queensland's Software Architecture Course Materials, @ Richard Thomas, CC BY-SA 4.0 (https://github.com/CSSE6400)
- Neal Ford, Mark Richards, Pramod Sadalage, Zhamak Dehghani, Software Architecture: the Hard Parts, O'Reilly Media, 2021

# References


- Newman, Sam. Building Microservices, O'Reilly Media, Second Edition, 2021
- Mojtaba Shahin, Mansooreh Zahedi, Muhammad Ali Babar and Liming Zhu, An Empirical Study of Architecting for Continuous Delivery and Deployment, In: Empirical Software Engineering, 24(3), 2019, Springer
- Mojtaba Shahin, Muhammad Ali Babar and Liming Zhu, Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices, In: IEEE Access, 5 (99), 2017, IEEE.
- Matthew Skelton, Chris O'Dell , Continuous Delivery with Windows and .NET, 2016, O'Reilly Media, Inc.