Introduction to Enterprise Architecture

Home of All the Laws of Nature

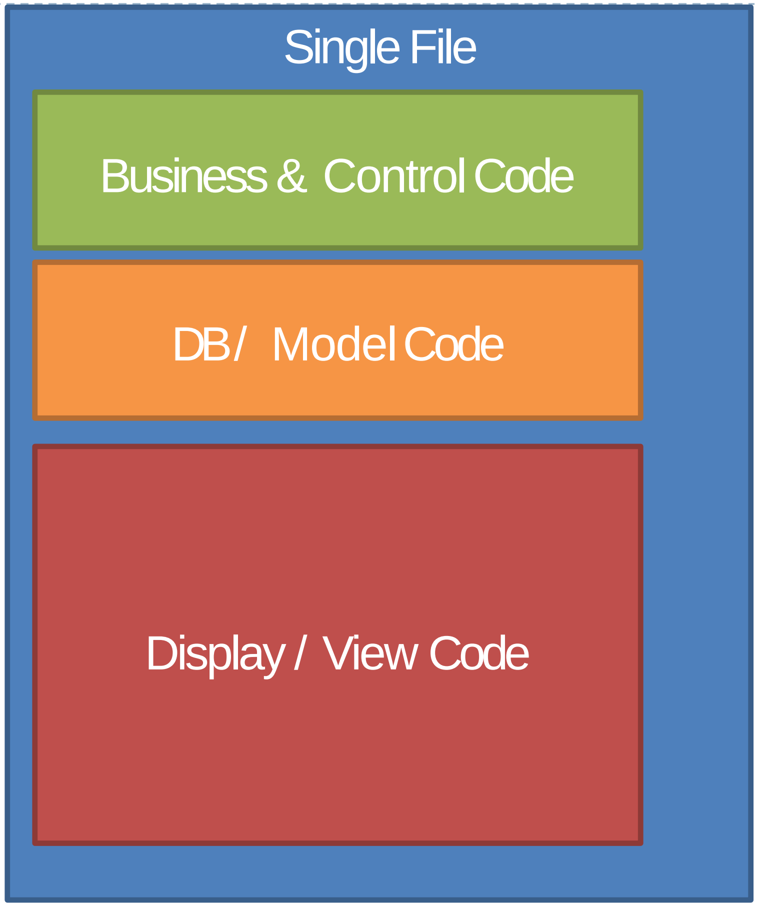
Enterprise

- Enterprise == **Big Business**
  - Businesses typically need to (at least) keep track of their what they sell and who they sell it to
  - Generally business applications are a front-end to a database.

- The **complexity** comes in:
  - implementing the business rules, how things interact.
  - Scaling the application to (big business) size

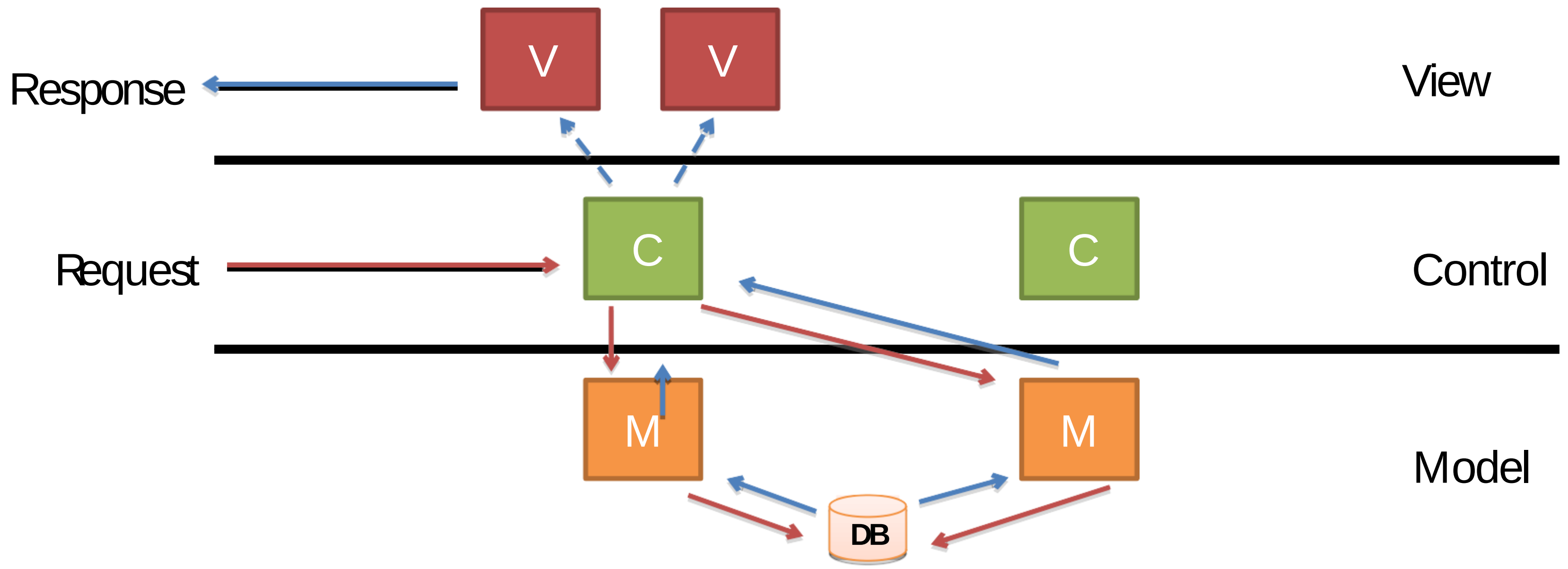
Applications

- As a business grows, a small application becomes a big application.
  - **Small applications** are okay with **Model 1**
  - Not so maintainable

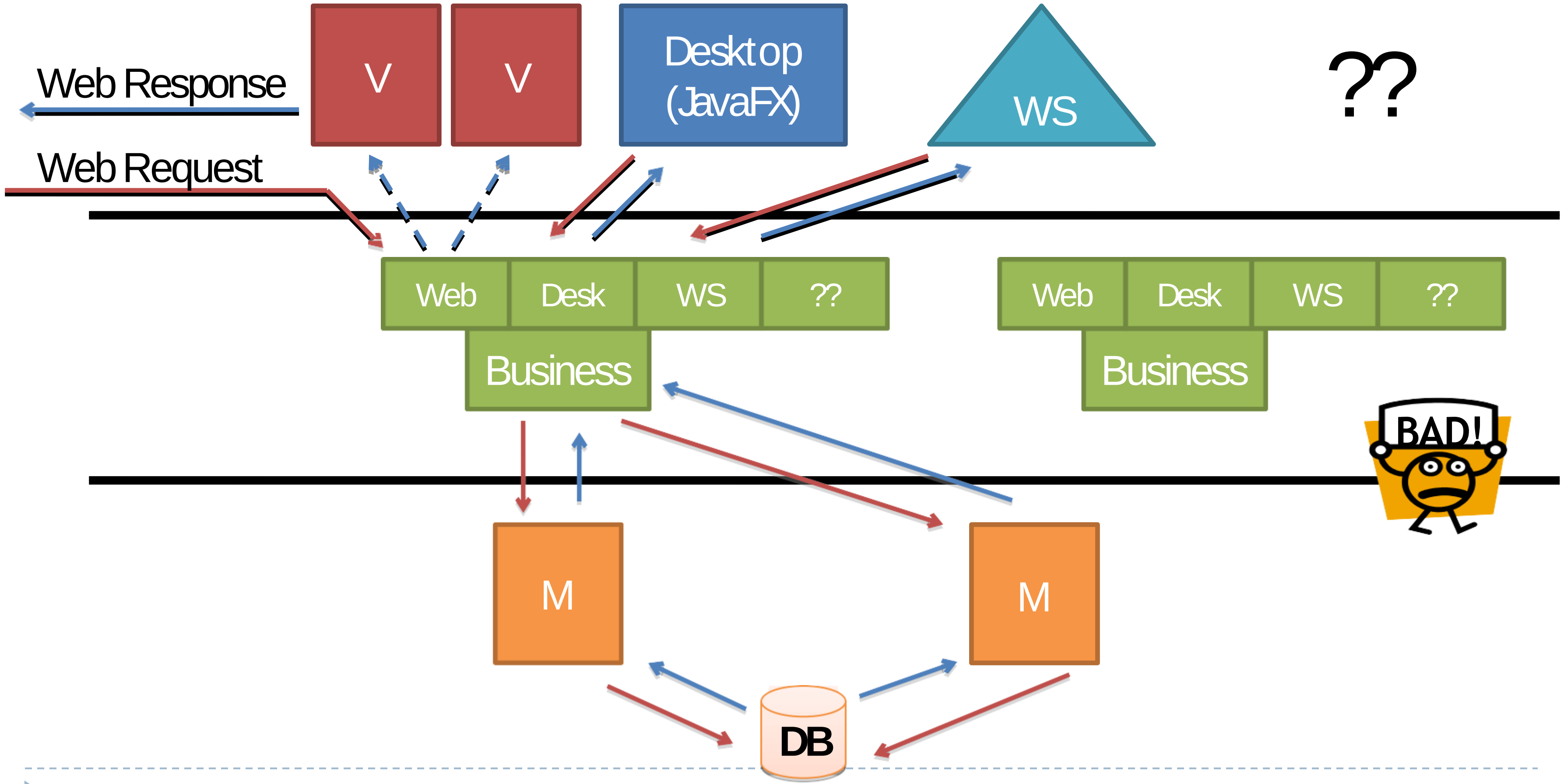


Growing Application

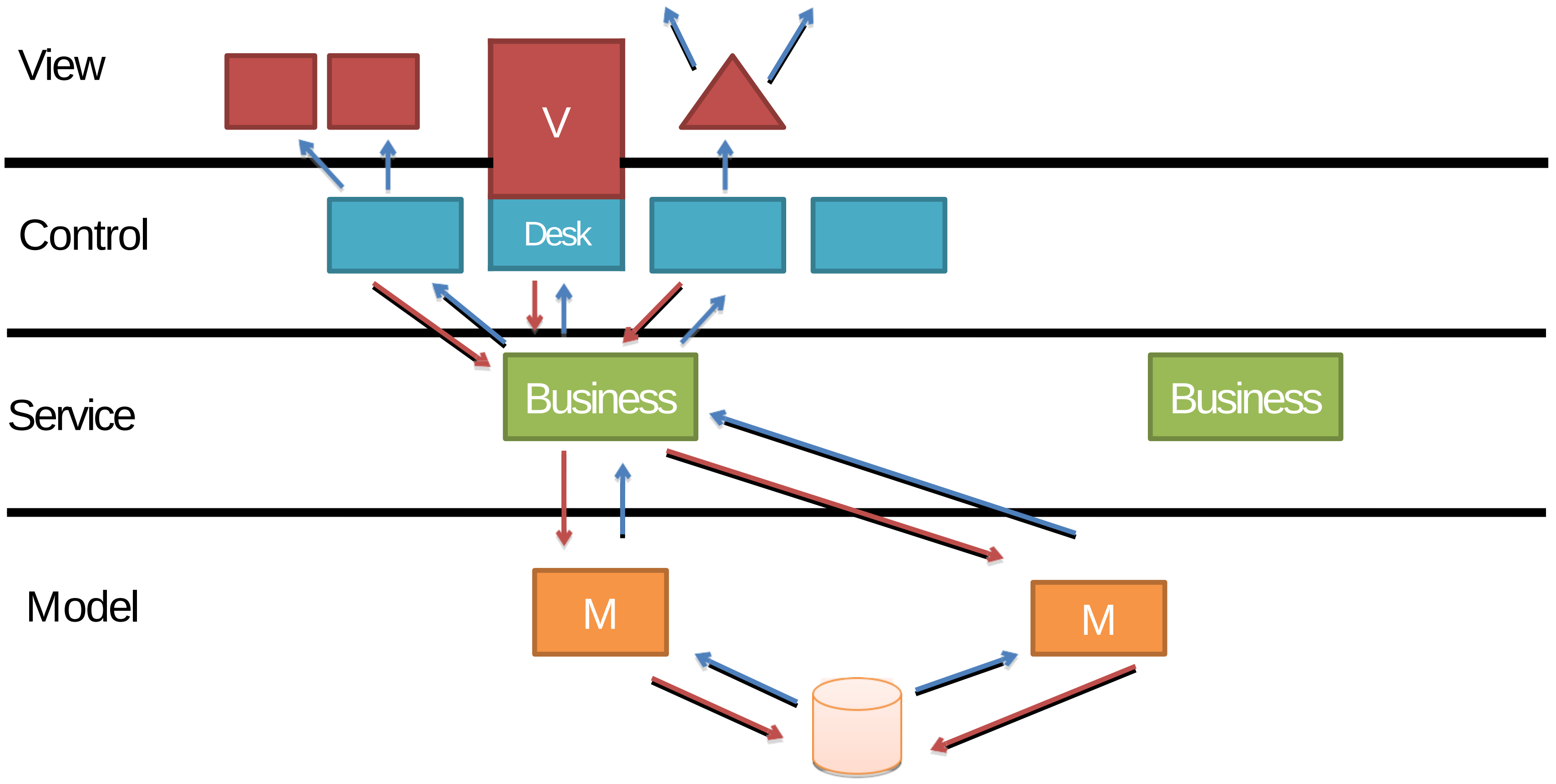
- For better **maintainability** split it into tiers
  - First 3 Tiers ~ Model / View / Control



Problem with 3 Tiers

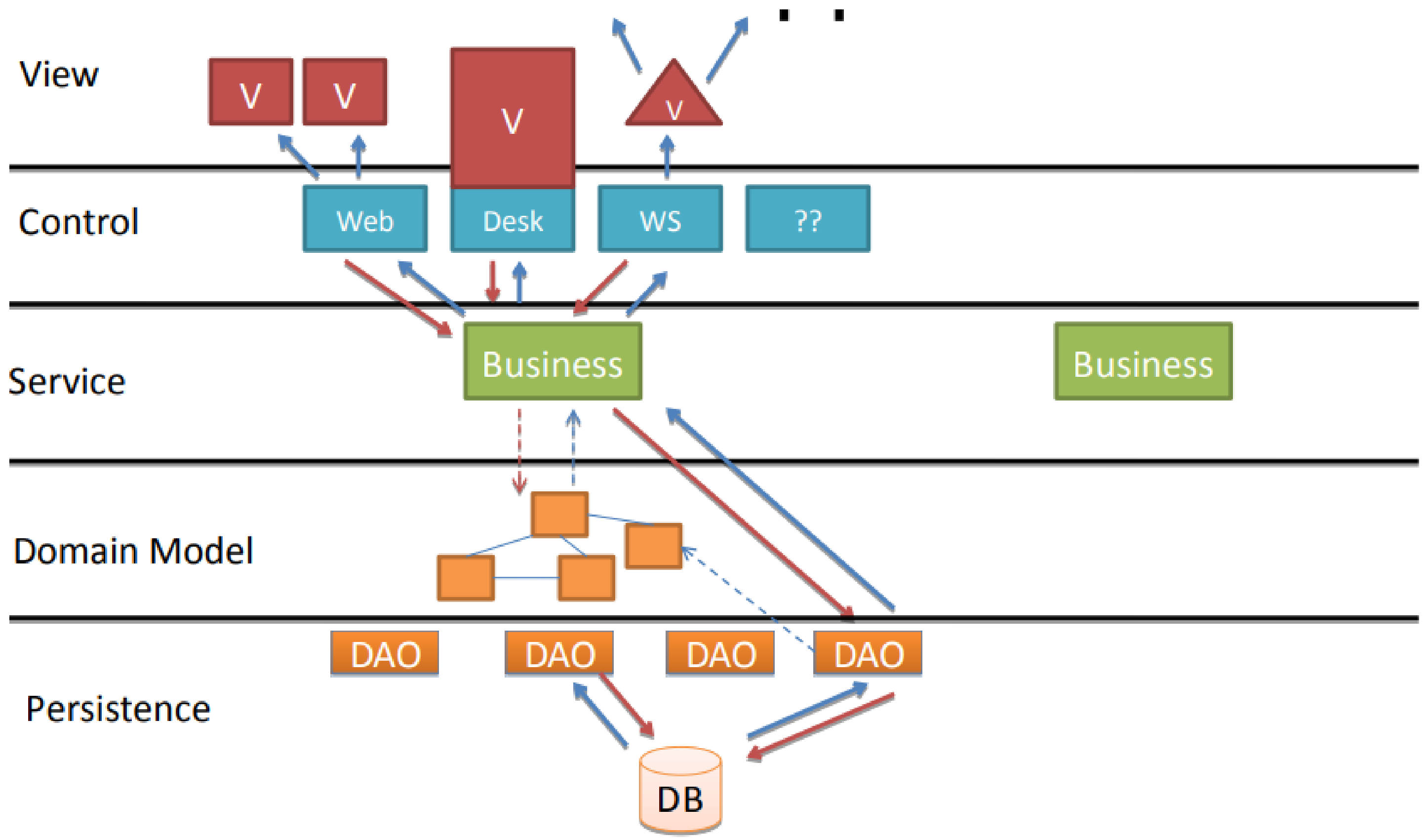


Service Layer

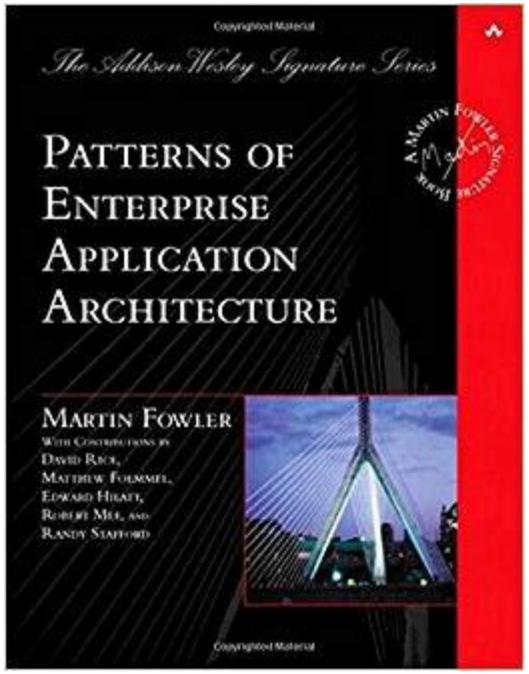


- The **Model layer** presents another problem
- It often has 2 different concerns:
  - An Object Oriented data object (a domain model)
  - Code to persist it into a relational DB

N-Tier Application



Enterprise Patterns



- Martin Fowler, 2002
  - Domain Model,
  - Service Layer,
  - Repository (aka DAO)

– These are some of the many **patterns from this** book that we will use in this course

Big Application

- A single big application is called a **monolith**.
- These often have **maintainability issues**:
  - A change in one place means recompiling the app
  - A change in one place could affect many other things (parts of the same big application)

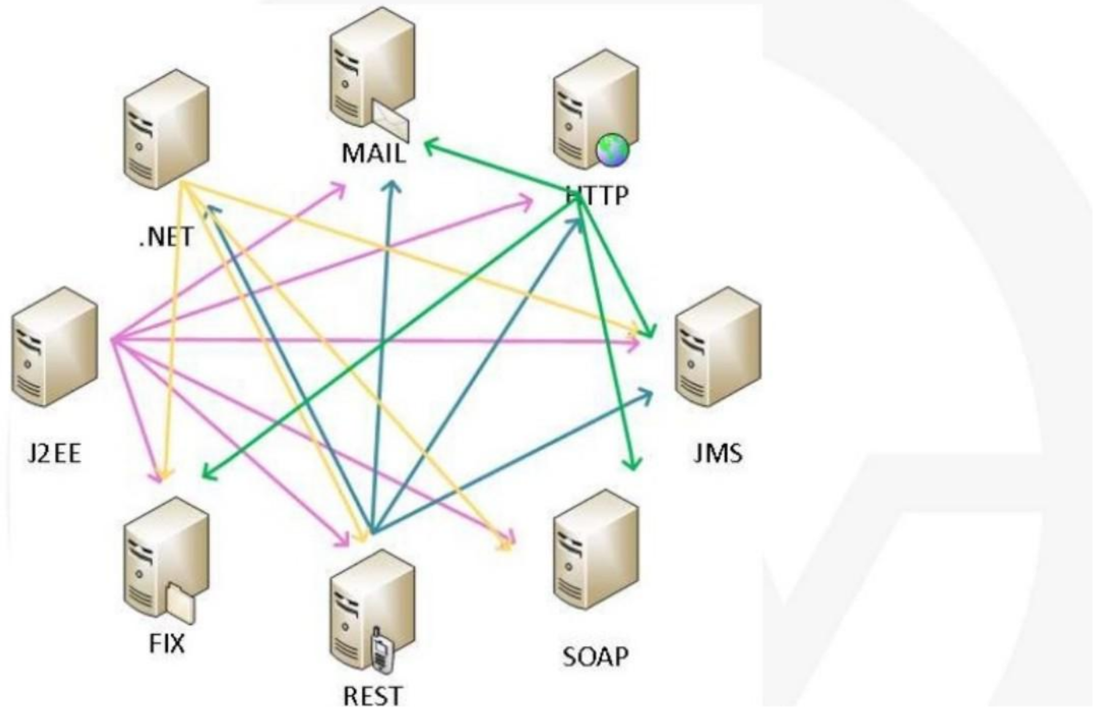
Multiple Monoliths

- At some point another application is made, or bought, or two businesses combine
- Whatever the reason, **integration is needed**
  - This has been true since the earliest days of Enterprise Applications

Initial Integration

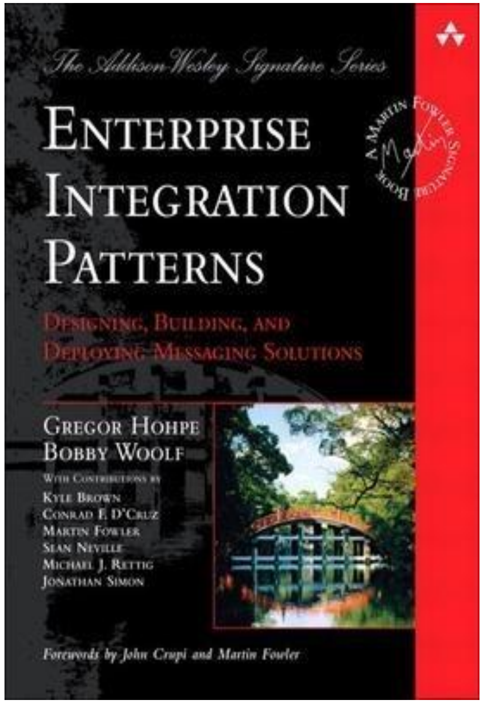
Spaghetti Integration

What about maintainability, scalability, troubleshooting and governance?



From: https://www.slideshare.net/wso2.org/resilient-enterprise-messaging-with-wso2-esb

Enterprise Integration Patterns


The Addison-Wesley Signature Series
Enterprise Integration Patterns
Designing, Building, and Deploying Messaging Solutions
Gregor Hohpe
Bobby Woolf
With Contributions by
Kyle Brown
Conrad F. D'Cruz
Martin Fowler
Sean Neville
Michael J. Rettig
Jonathan Simon
Forewords by John Crupi and Martin Fowler
A Martin Fowler Signature Book

By Source (WP:NFCC#4), Fair use, https://en.wikipedia.org/w/index.php?curid=42964088

- Gregor Hohpe and Bobby Woolf, 2003
  - We will introduce (mention) some of these **patterns** at the end of this course

Book specifies 4 Ways to Integrate

- Used in the past
  - File Transfer (leaves a lot to the developer)
  - Shared Database (does not scale as well)

- Modern approaches:
  - **Remote Procedure Invocation** (synchronous)
  - **Messaging** (Asynchronous)

Integration and Architecture

- **Integration** is not just something that can be used to connect different applications
  - It can also be an **architectural solution**

- Why have one big, hard to maintain, hard to scale, monolith when you can break it into parts?
  - Each part provides a “service”

Service Oriented Architecture

- To Solve Monolith problems
- **Separate applications** each provide their own **service**
  - The ‘service layer’ is the point of integration

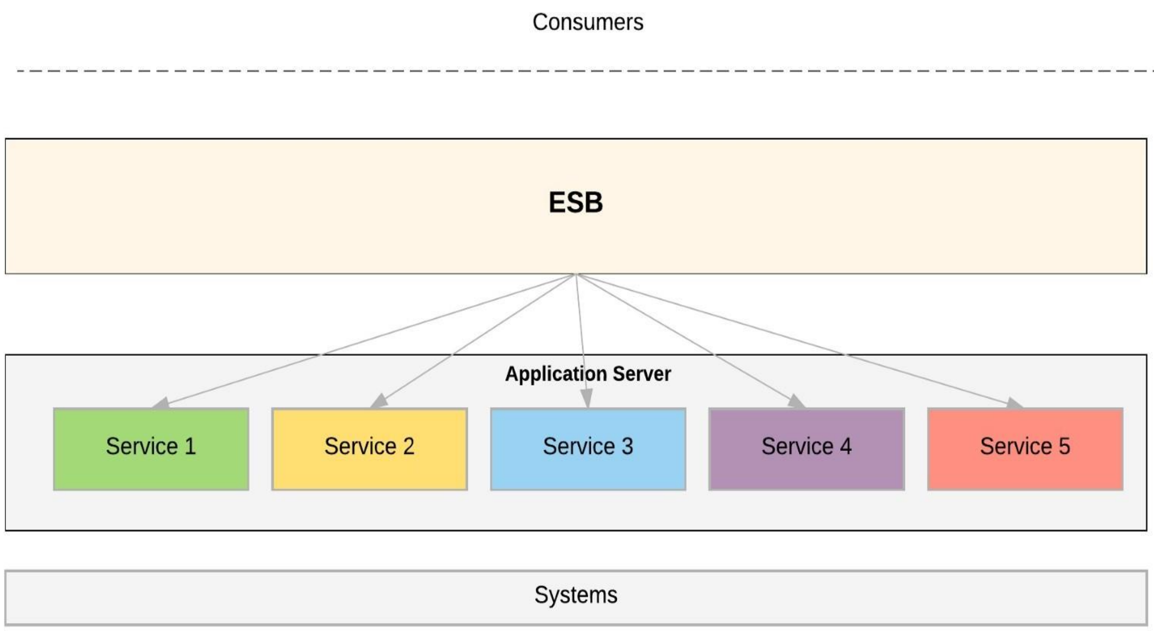
- [Wikipedia] Each service / application:
  - Represents a business activity with a specified outcome Is self-contained
  - Is a black box for its consumers.
  - May consist of other underlying services

Service Oriented Architecture

- A coherent integration plan is needed, how are the different services going to communicate?
  - How are they going to be combined?

- A common solution:
- An **Enterprise Service Bus** (ESB)
  - Coordinates activities between the services
  - Can contain logic and combine services

Enterprise Service Bus



Similar to hardware, a channel through which all communication flows

From: https://medium.com/@kasunindrasiri/microservices-apis-and-integration-7661448e8a86

ESB

- ESBs generally **contain logic** to provide:
- Routing, choreography (combine), transformation, business rules

- Useful, but can also be **a real problem**:
- Business logic spread between services and ESB (what is where?)
- ESB is a single, monolithic, center of the application
    - Single point of failure

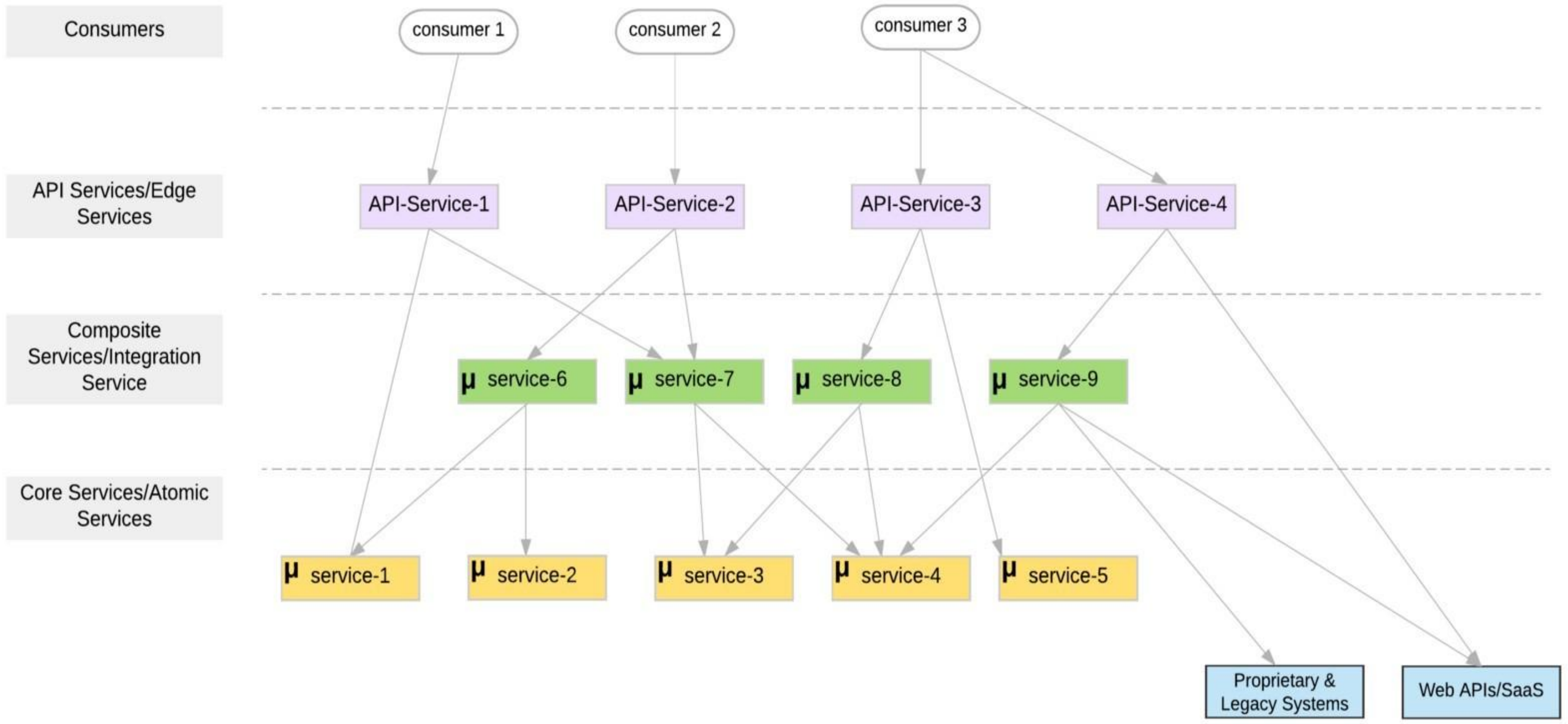
MicroServices

- The Micro Services architectural style is often seen as a response to the use of an ESB

- **Micro Services** emphasizes:
  - Smart Endpoints (services) and **Dumb Pipes**

- Also generally smaller (more fine grained) services
  - What is or is not small is undefined

Combining Micro Services



From: https://medium.com/@kasunindrasiri/microservices-apis-and-integration-7661448e8a86

Main Point

A software framework encapsulates the knowledge of experts, allowing the developers to take advantage of sound solutions and focus on the project qualities.

Science of Consciousness: *Through the practice of Transcendental Meditation, a person taps the value of Pure Consciousness which encapsulates knowledge of all the laws of nature..*