

# What is **Software Architecture**... and **an overview**.

---

PV260 Software Quality



Ondřej “Ondra” Krajíček  
[ondrej.krajicek@ysoft.com](mailto:ondrej.krajicek@ysoft.com)  
@OndrejKrajicek



“Perfection is not achieved when there is nothing to add, but when there is nothing to remove.”

**Antoine de Saint Exupéry**

“Insanity is doing the same thing over and over and expecting different results.”

**(attributed to) Albert Einstein**

“It has to work.”

**IETF RFC 1925**

Software Architecture is seldom about functional requirements.

**THE SOFTWARE**



**IN REALITY**

**THE SOFTWARE IN DEMO**

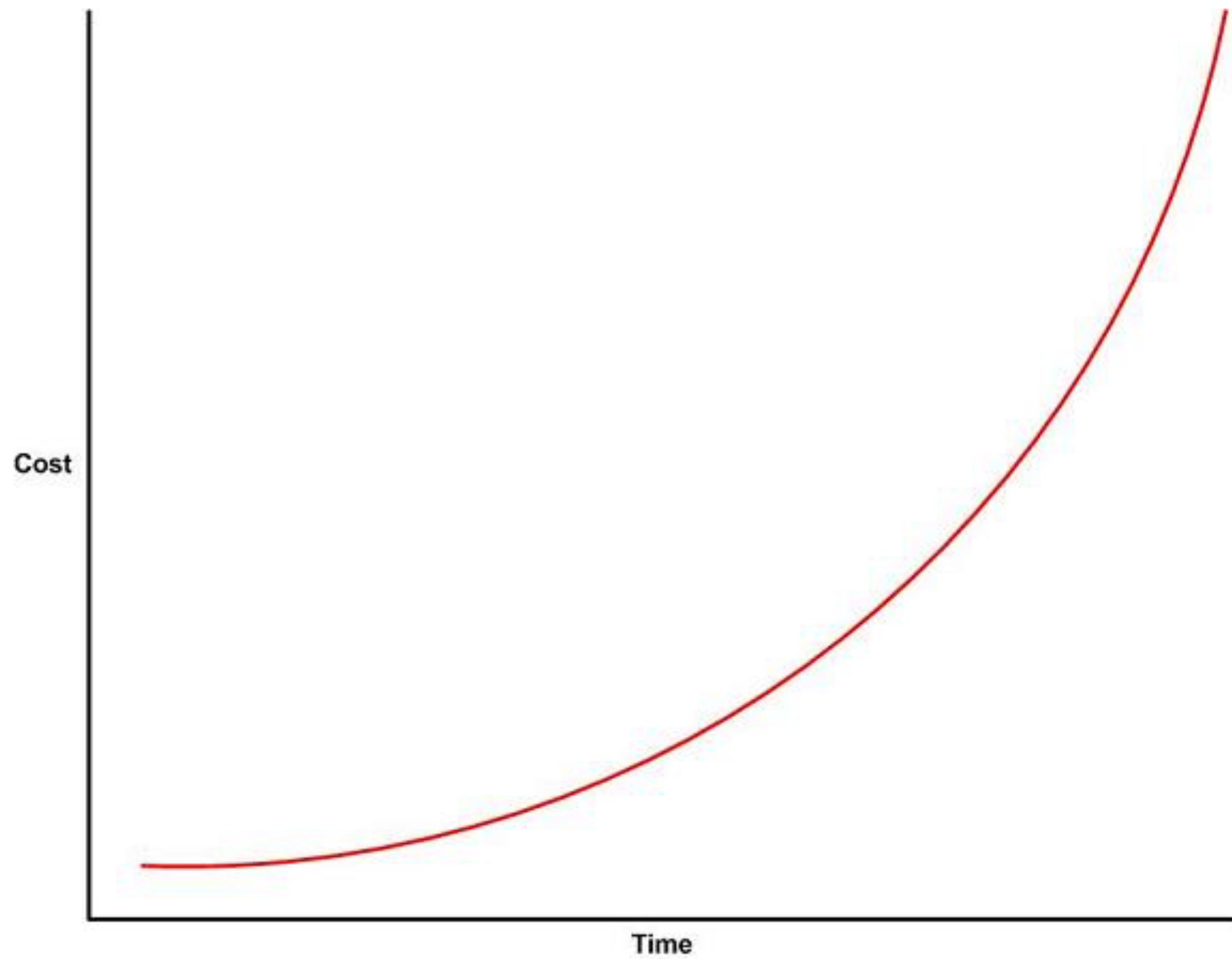


**TO THE CUSTOMERS**

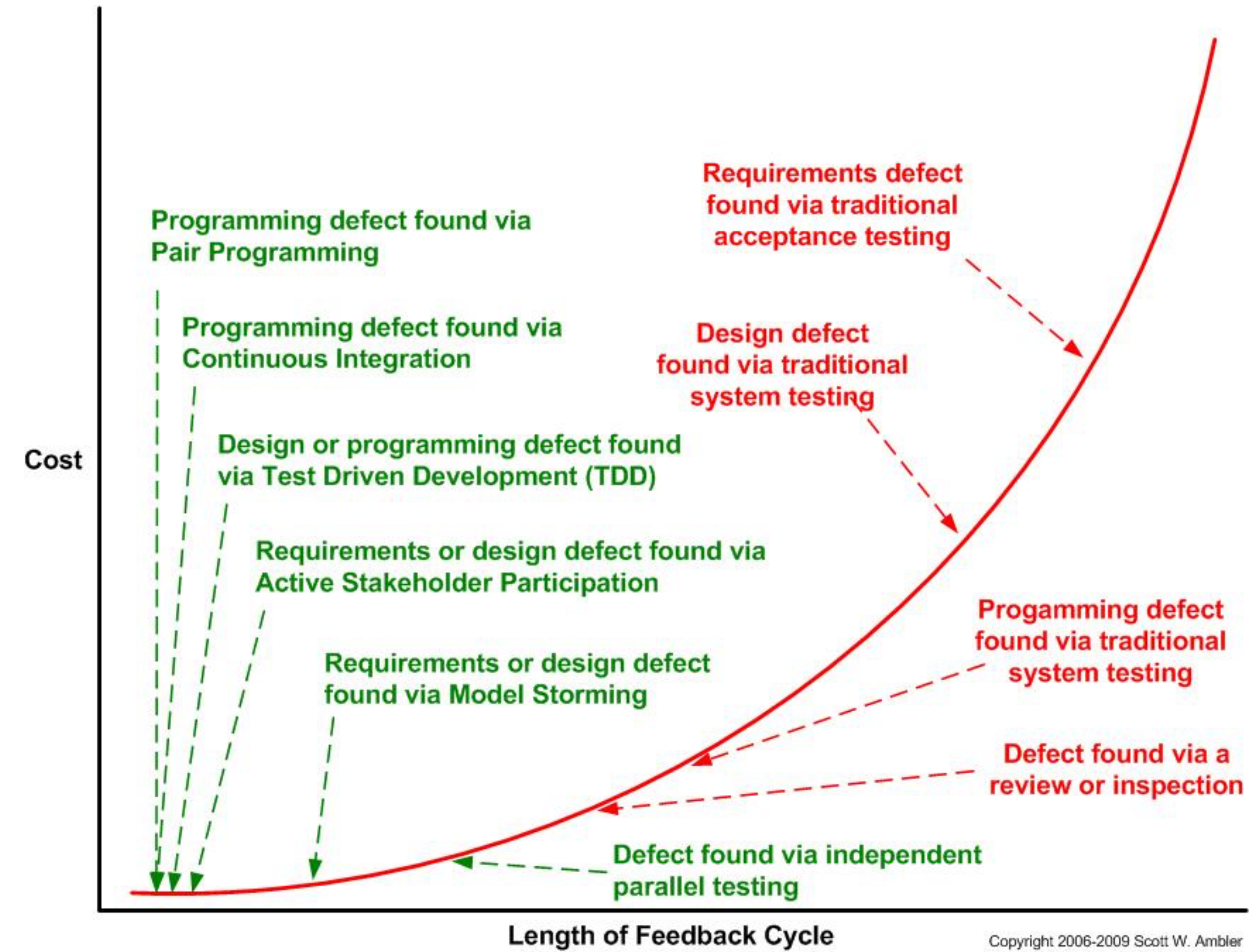
Software Architecture is the important stuff  
(whatever that is).

---

**Ralph Johnson**



Cost of Change Curve



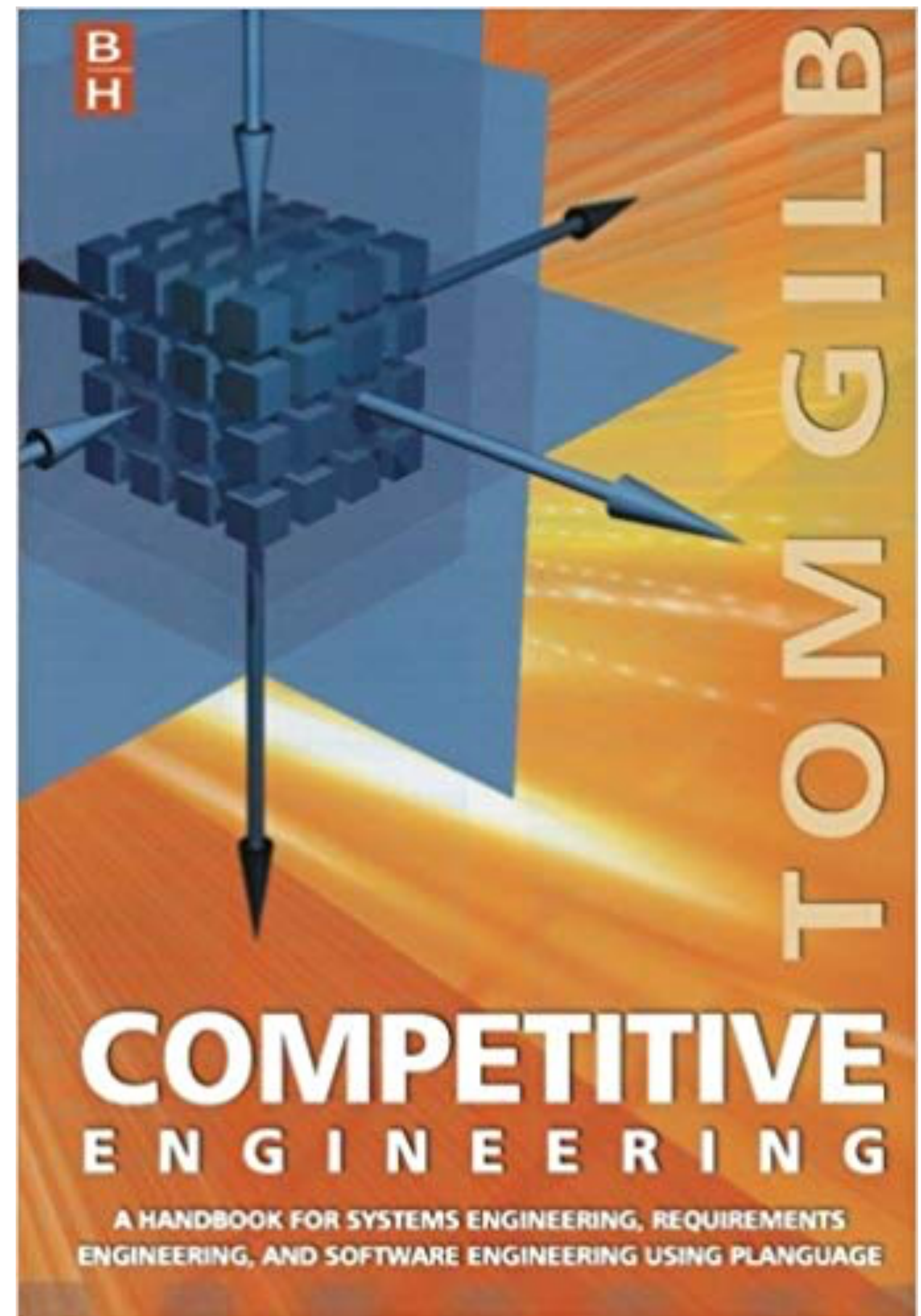
Copyright (c) 2006-2009 Scott W Ambler



**SOFTWARE ARCHITECTURE IS THE  
SERVANT OF HIGH-PRIORITY  
STAKEHOLDER VALUES.**

IS AS SIMPLE AS POSSIBLE, BUT NOT  
SIMPLER AND IS DESIGNED TO BE  
REPLACEABLE.

**Tom Gilb (Architecture Manifesto)**



SERVANT

HIGH-PRIORITY

STAKEHOLDER

VALUES

AS SIMPLE AS POSSIBLE

NOT SIMPLER

REPLACEABLE

# Software Architecture is **Strategy**

---

Strategy is a plan how to deliver on your goals. Your goals are defined by high-priority stakeholders.

# SOFTWARE ARCHITECTURE **IS THE STRATEGY HOW TO DELIVER HIGH-PRIORITY STAKEHOLDER VALUES.**

---

It is still important to **accept change** because no battle plan survives the first contact with the enemy.

# Software Architecture is **Risk Mitigation**

---

What happens when things go wrong? Is it important?

# Software Architecture is **Communication**

---

What does it mean? How to deliver and protect stakeholder values?

# Software Architect is a **Teacher**



# Who **does** software architecture then?

---

Software Architecture is done by **everyone**.



# 60 Minutes Software Architecture Crash Course

---

# Architectural Styles

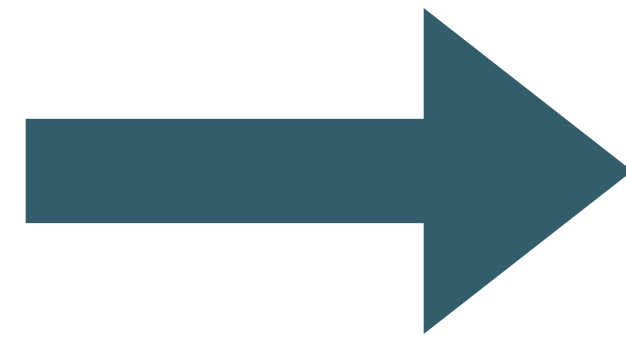
---

- Tiered Architecture
- Hexagonal Architecture
- Onion Architecture
- Object Oriented Architecture
- Service Oriented Architecture
- Microservices

Which one is the best one?



- **Consistency**
- **Cohesion**
- **Coupling**
- **Clarity**



**Cost of  
Change**

# Principles over Patterns.

---

Consistency

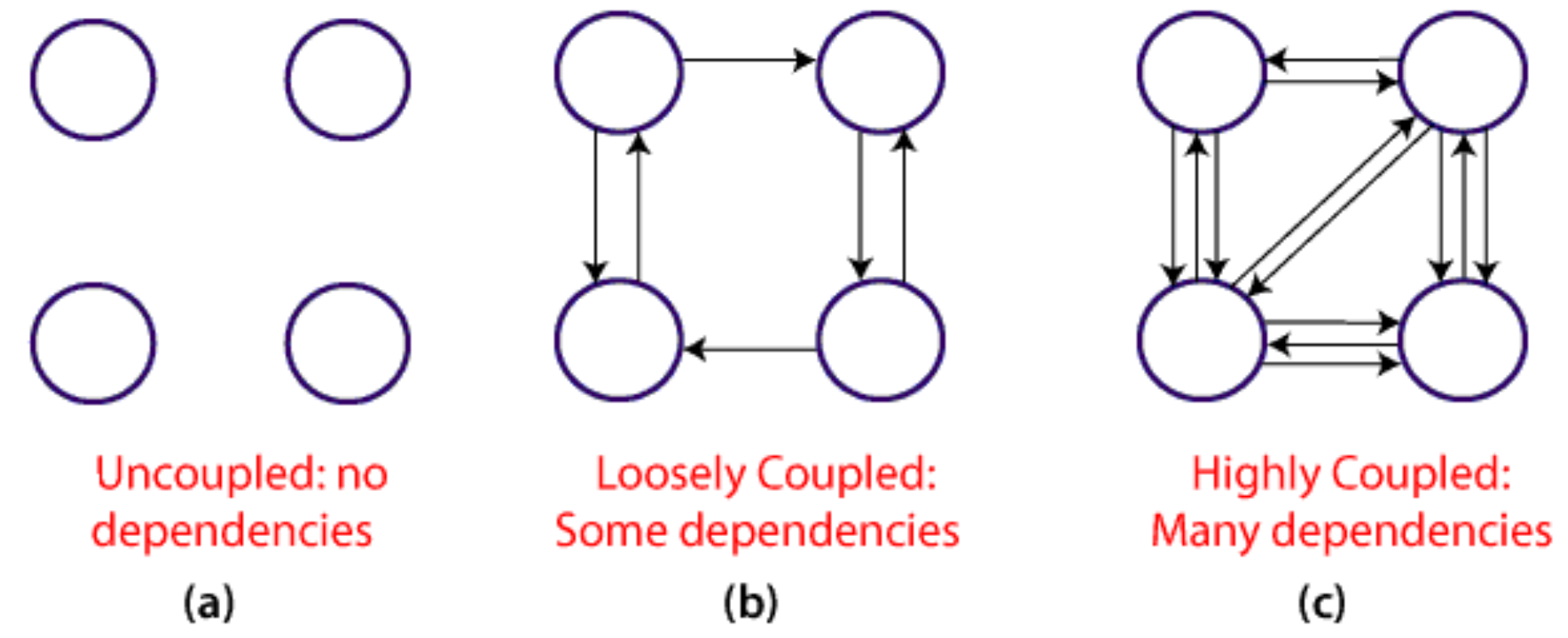


# Module, Data, Service, Interaction Dependencies.

---

## Coupling

### Module Coupling

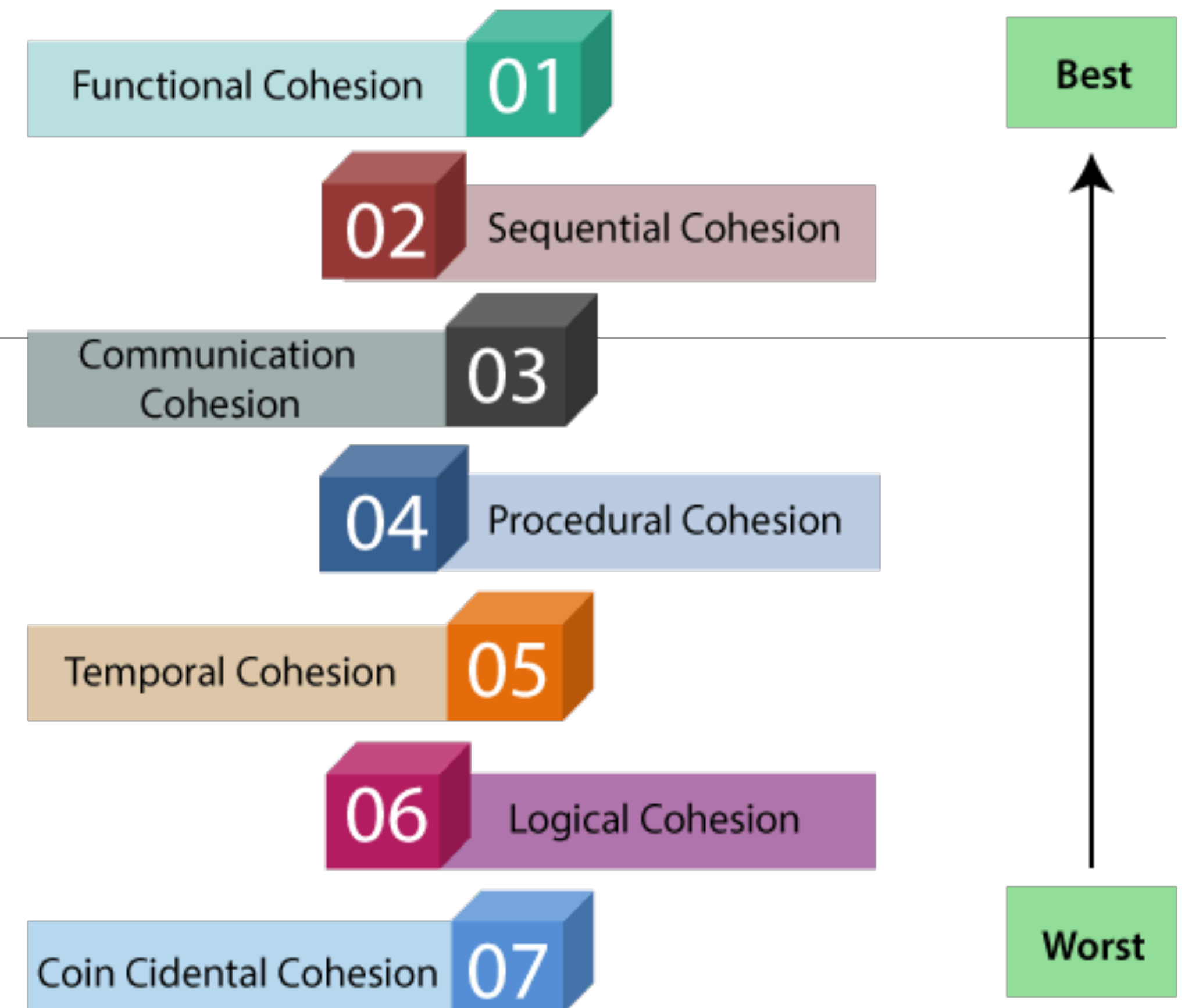


<https://www.javatpoint.com/software-engineering-coupling-and-cohesion>

# Well-Designed or Leaking Abstractions.

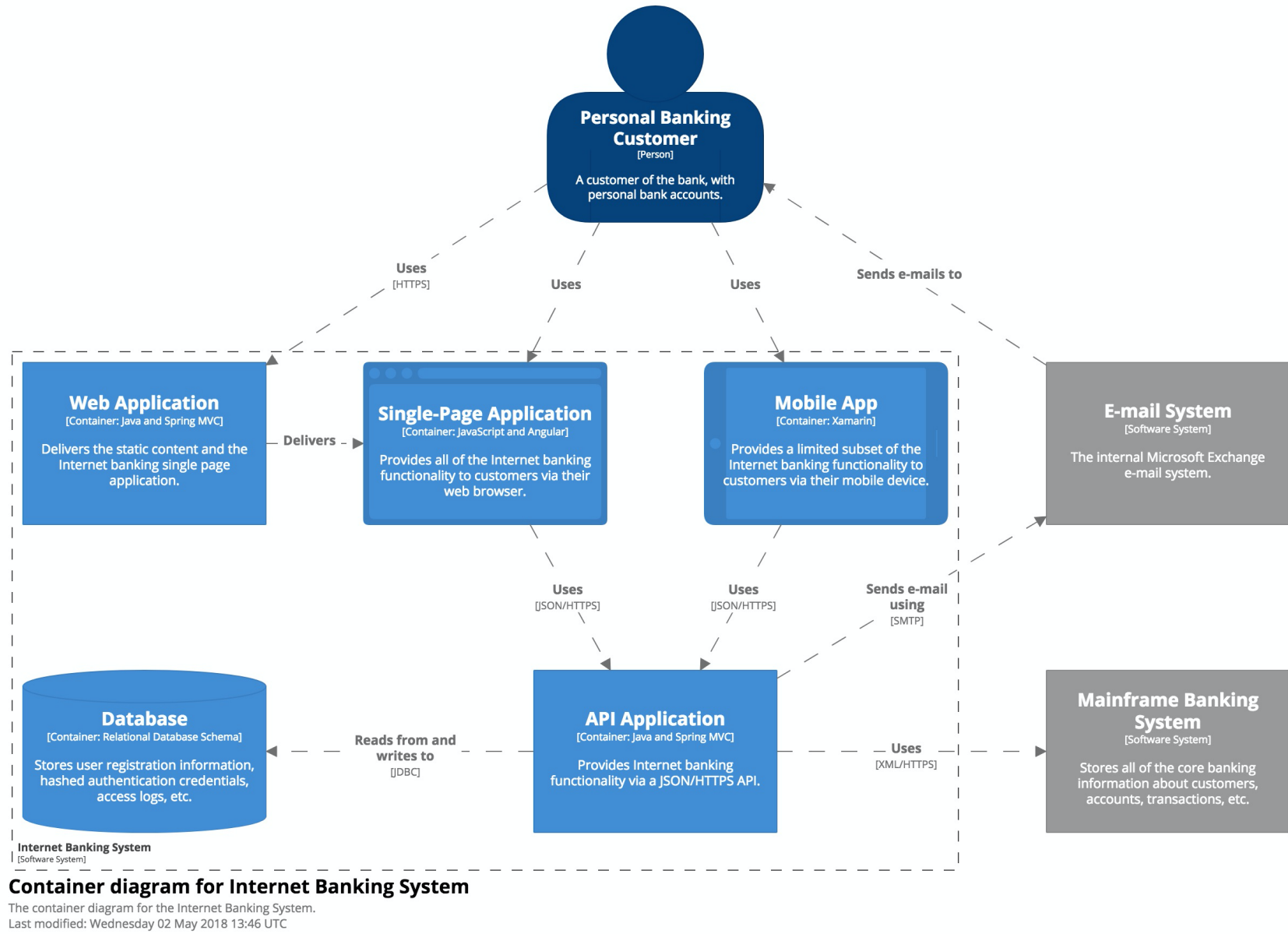
Cohesion

## Types of Modules Cohesion



Everyone understands why, how and what to do. System deteriorates slower and technical debt does not grow quickly.

Clarity





# Replaceable Architecture

---

Wait, what?

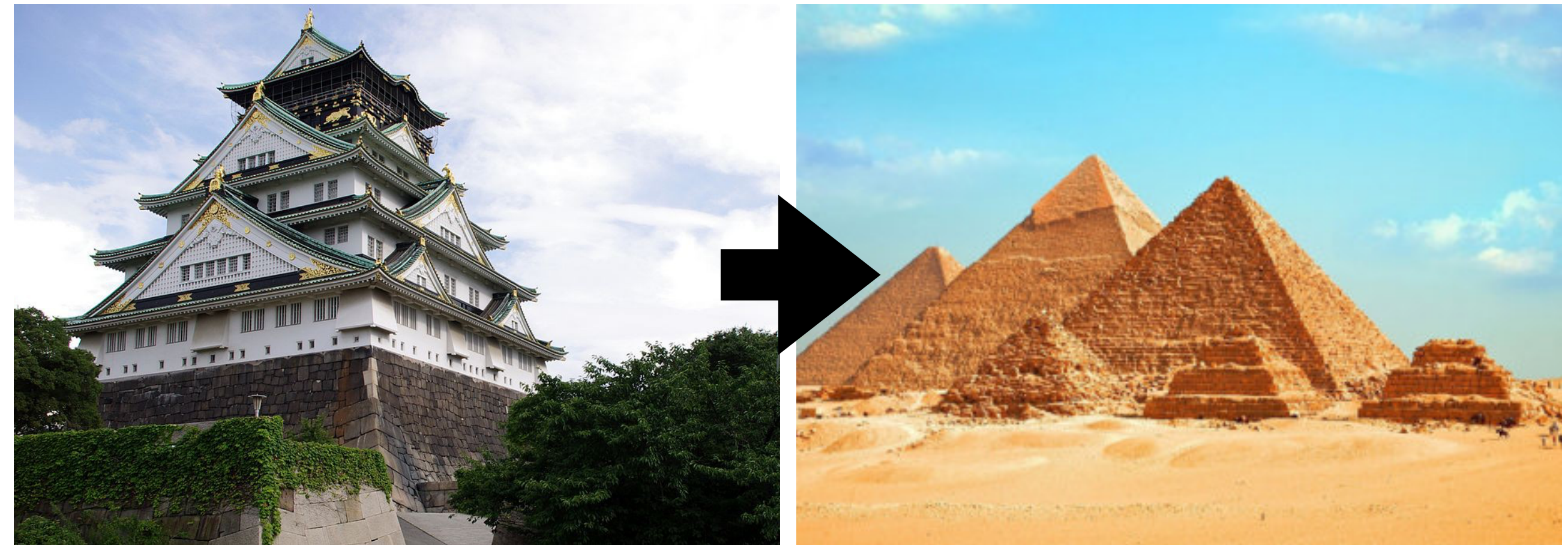


Osaka Castle



Giza Pyramids

- Built to last: hundreds and thousands of years.
- Built to survive natural disasters, especially earthquakes (shinbashira).
- Both have very different architecture.
- You cannot replace one with the other.
- Why would you?



Replaceable as in ***Having rather low Cost of Change***

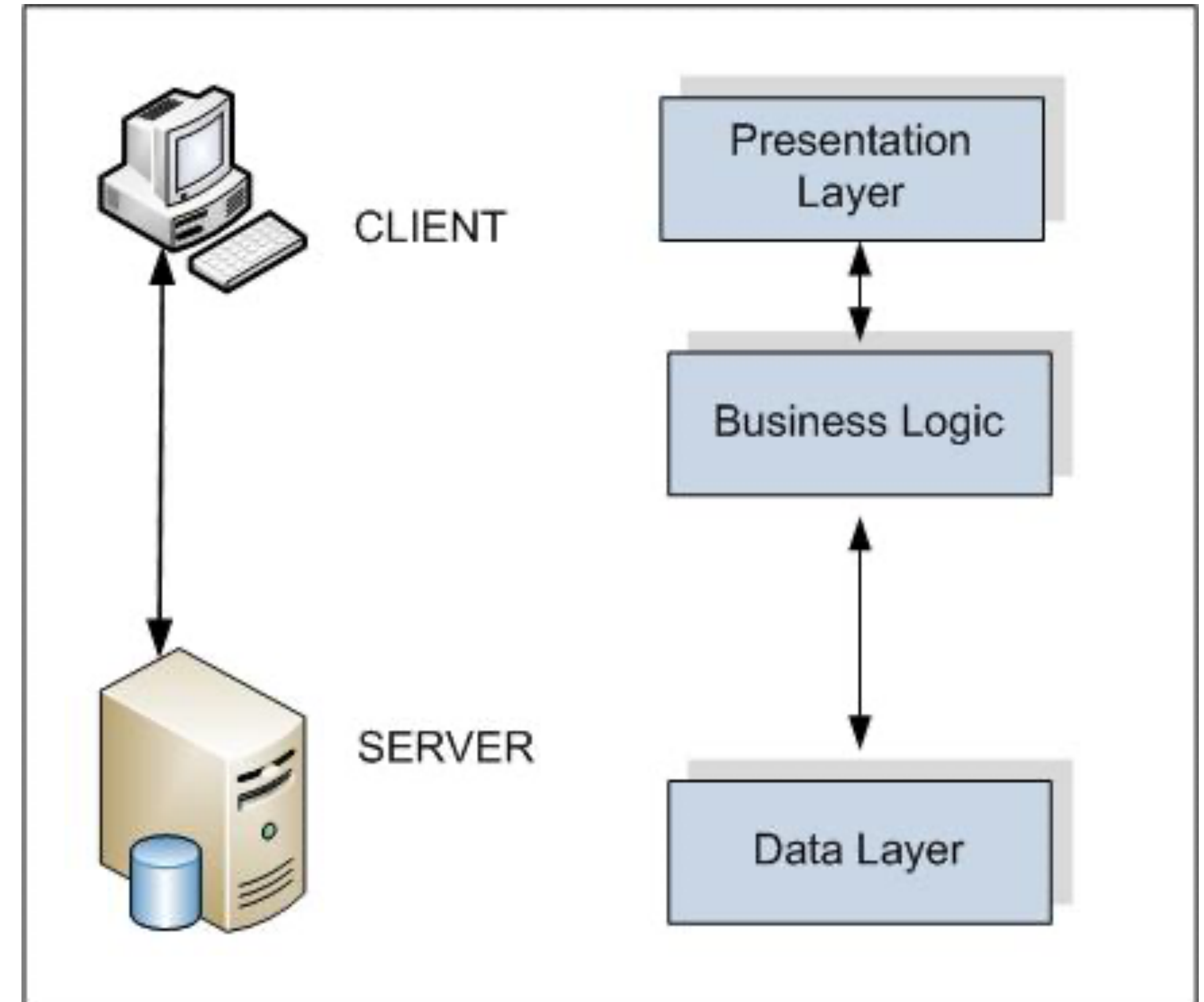
---

How to decrease Cost of Change?

# 2-Tier Architecture

---

- Original Client / Server
- Business Logic is implemented on the client, server or both.
- What are the issues?



# 3-Tier Architecture

- Decouple presentation from business logic. Business logic is isolated from client and server.
- Business layer often historically hosted in *application* servers with obscure technologies (j2ee, Microsoft ASP, PHP, ColdFusion, etc.).
- How is it different from 2-Tier?

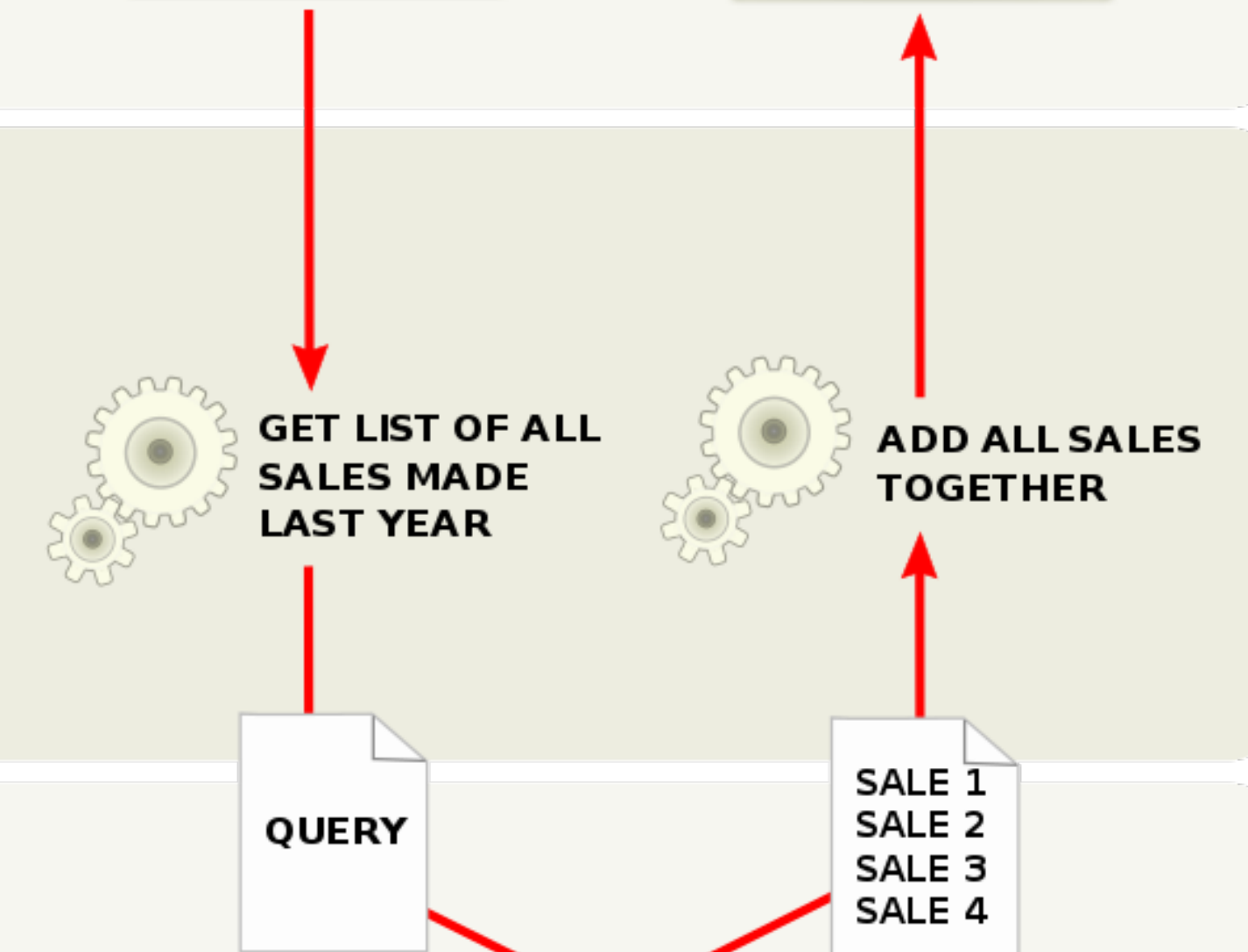
## Presentation tier

The top-most level of the application is the user interface. The main function of the interface is to translate tasks and results to something the user can understand.



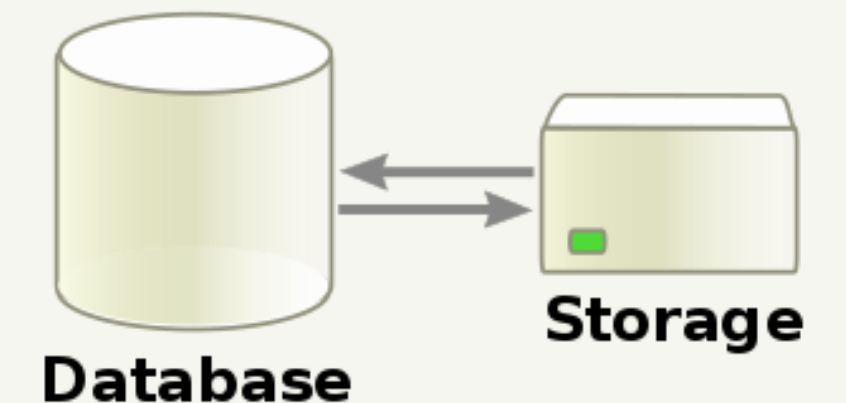
## Logic tier

This layer coordinates the application, processes commands, makes logical decisions and evaluations, and performs calculations. It also moves and processes data between the two surrounding layers.



## Data tier

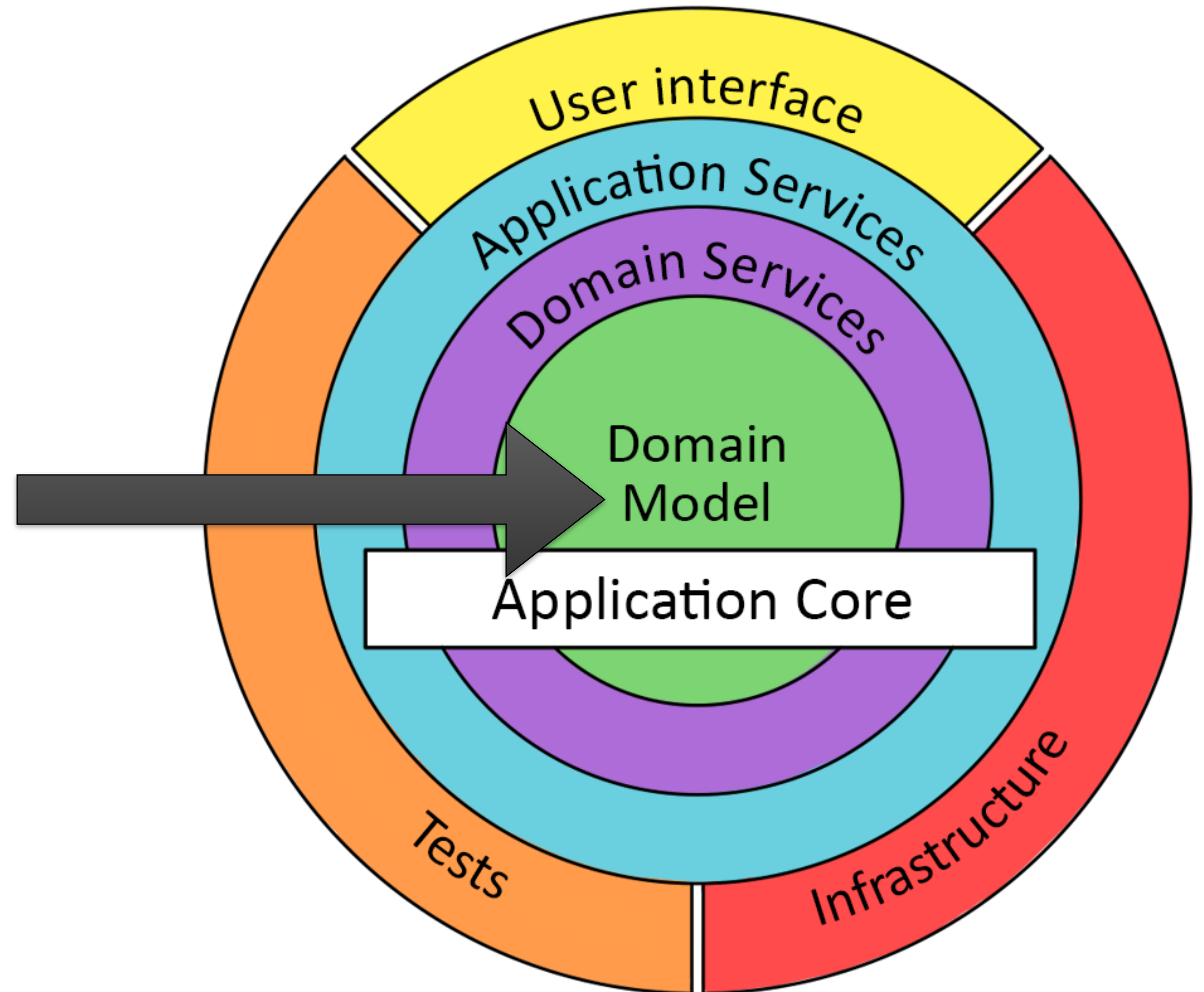
Here information is stored and retrieved from a database or file system. The information is then passed back to the logic tier for processing, and then eventually back to the user.



# Onion Architecture

---

- Built on the observation that most / all interfaces are alike.
- Outer layers depend on inner layers.
- Inner layers must not depend on outer layers.
- Enforces Inversion of Control.
- How is it different from N-Tier?

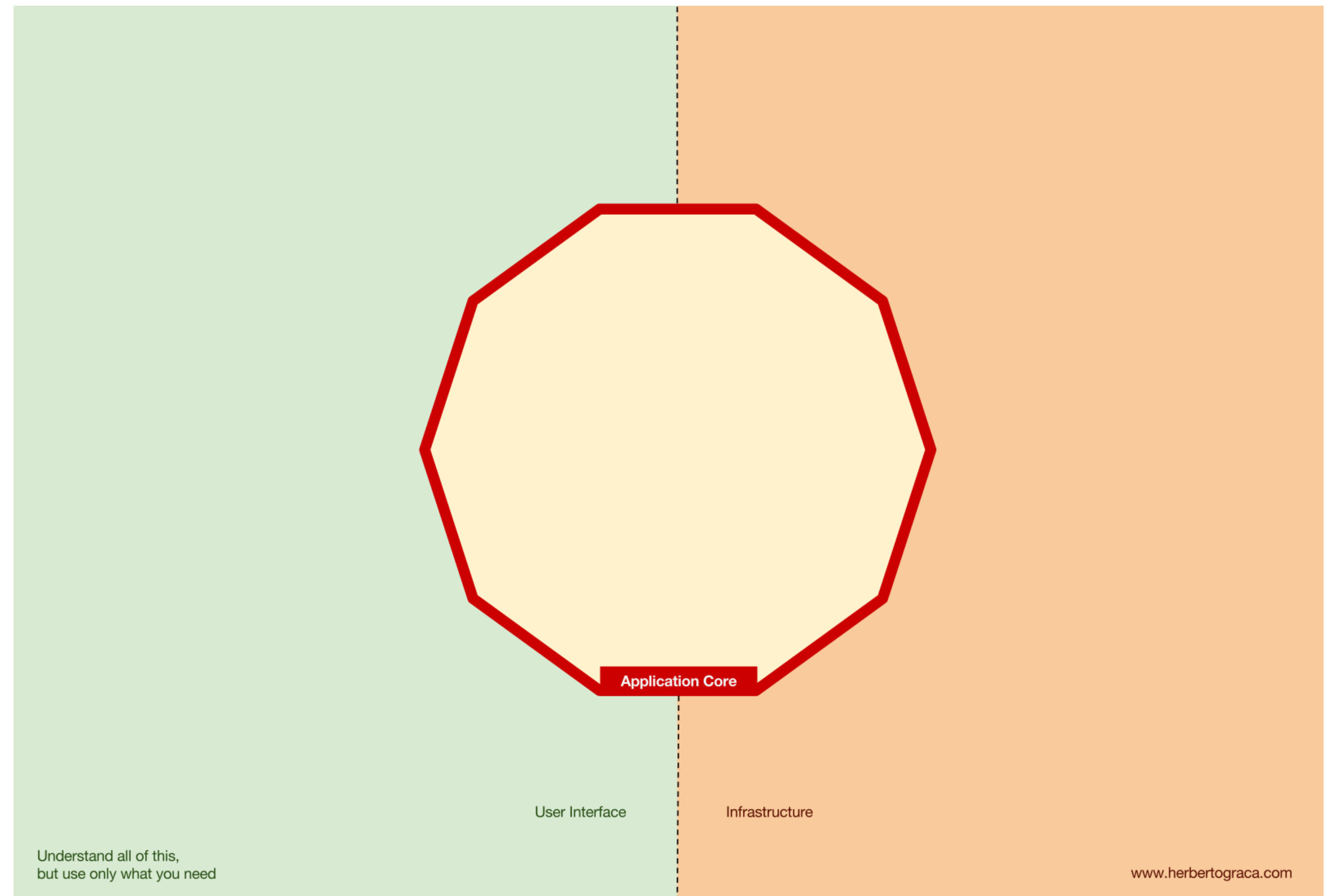




# Hexagonal Architecture

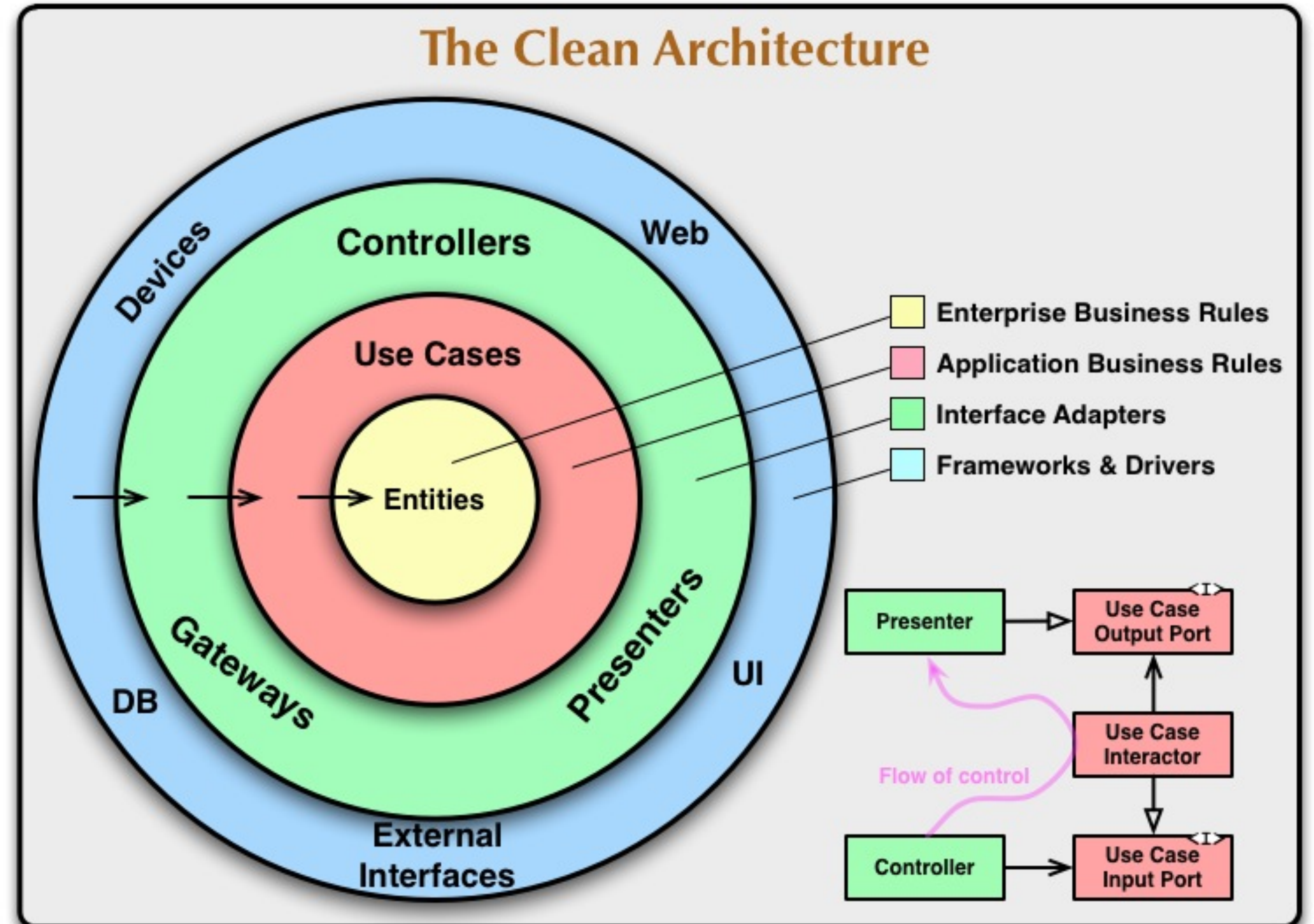
---

- Ports and Adapters Architecture
- Sometimes Onion and Hexagonal are viewed as the same.
- Hexagonal Architecture is more explicit and structured.
- Recommended reading:  
<https://herbertograca.com/2017/11/16/explicit-architecture-01-ddd-hexagonal-onion-clean-cqrs-how-i-put-it-all-together/>



# The Clean Architecture

- Onion + Screaming Architecture.
- Independent of Frameworks.
- Testable: all parts and as a whole.
- Independent of Interfaces.
- Independent of the data store / database / object persistence.
- Independent of any external impact.



So what does the architecture of your application *scream*?

When you look at the top level directory structure, and the source files in the highest level package; do they scream: **Health Care System**, or **Accounting System**, or **Inventory Management System**?

Do they scream **global**, **distributed**, **consistent** or **available**?

Or do they scream: **Rails**, or **Spring/Hibernate**, or **ASP**?

Recommended viewing: <https://www.youtube.com/watch?v=ZsHMHukIJY>

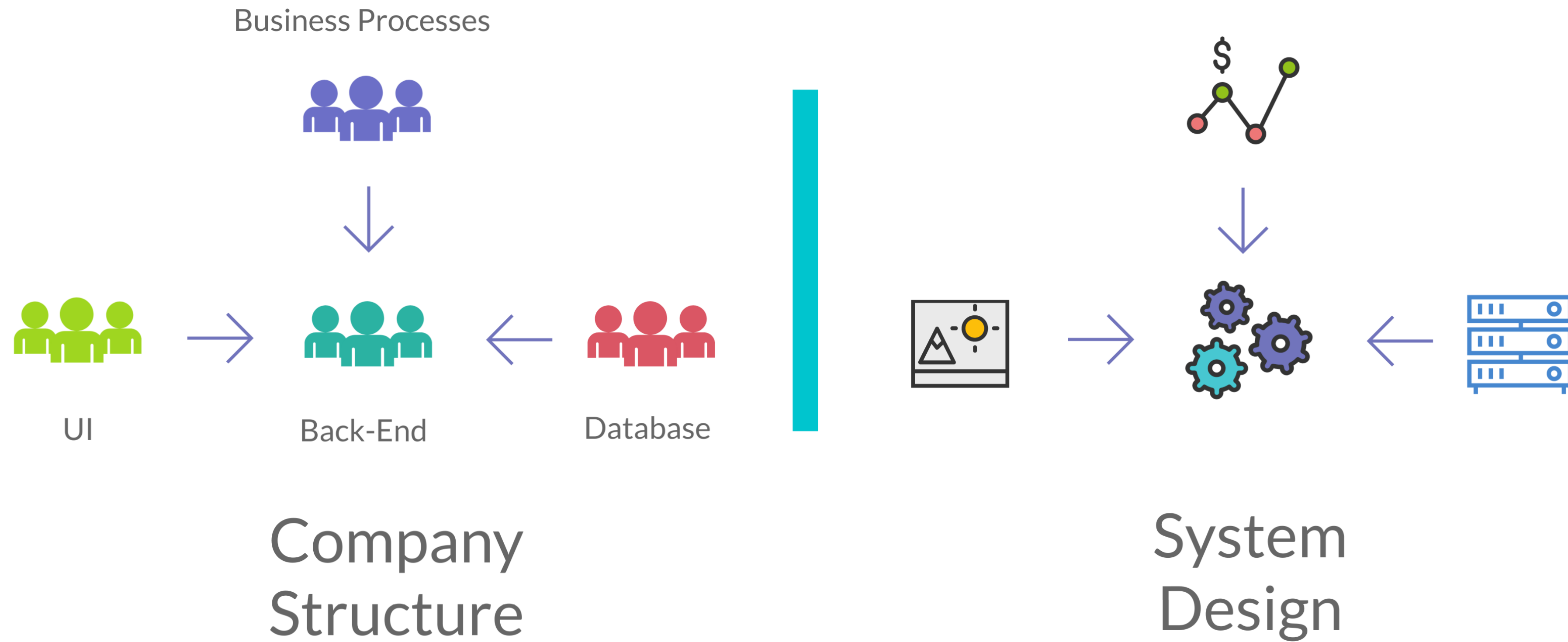


Microservices

also known as the *current silver bullet*.

<https://www.cgl.ucsf.edu/Outreach/pc204/NoSilverBullet.html>

# Conway's Law



Organization structure determines system architecture / design.

<https://medium.com/@learnstuff.io/conways-law-in-software-dev-3aa6324ead52>

As the systems get larger, complexity grows quickly and systems become unmanageable.

---

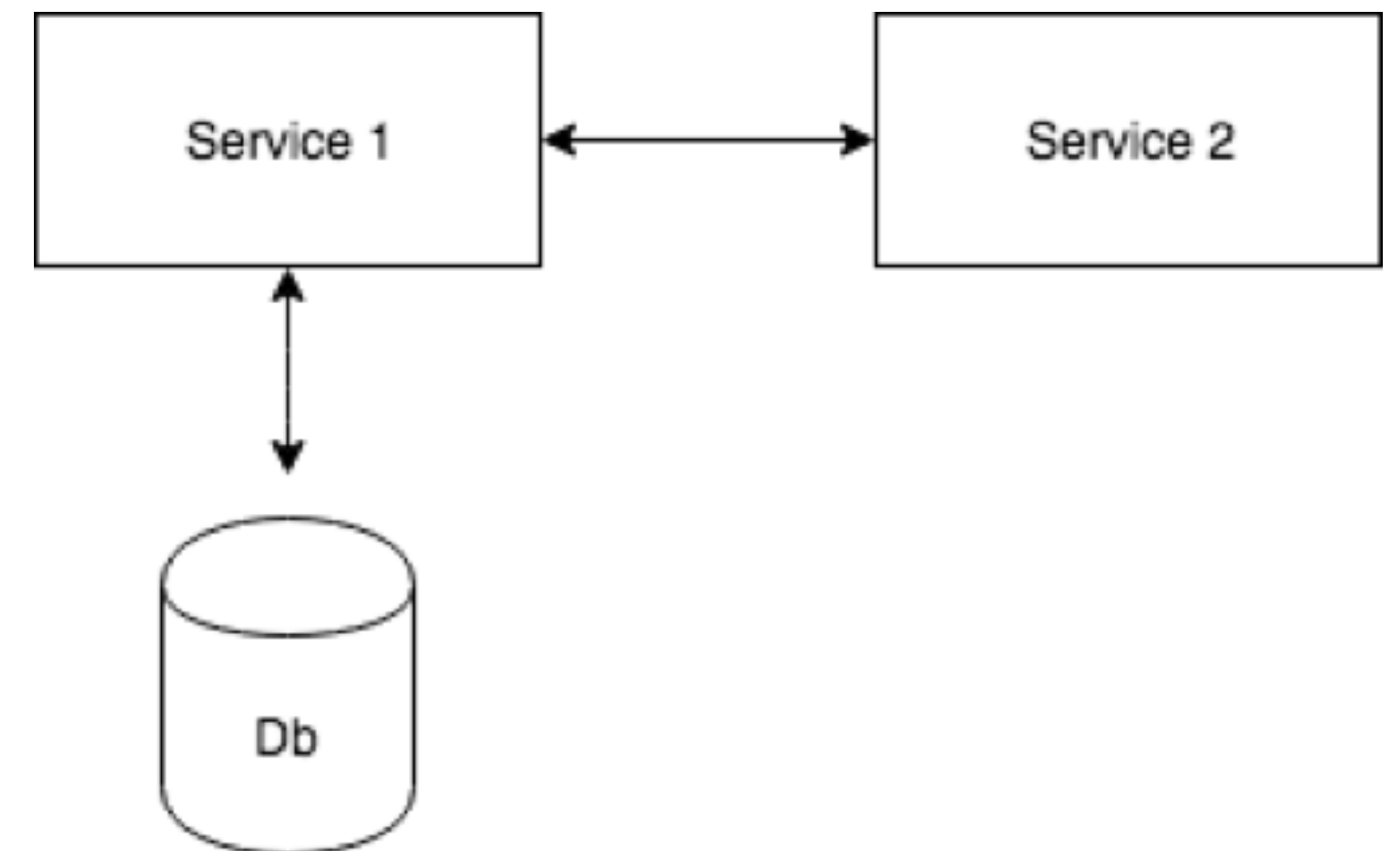
<http://www.laputan.org/mud/mud.html#BigBallOfMud>



- Split system in a set of loosely coupled, cohesive services.
- Each service does only one thing and does it well.
- Each service is represented only by its API.
- Each service has its own data.

---

## Microservices



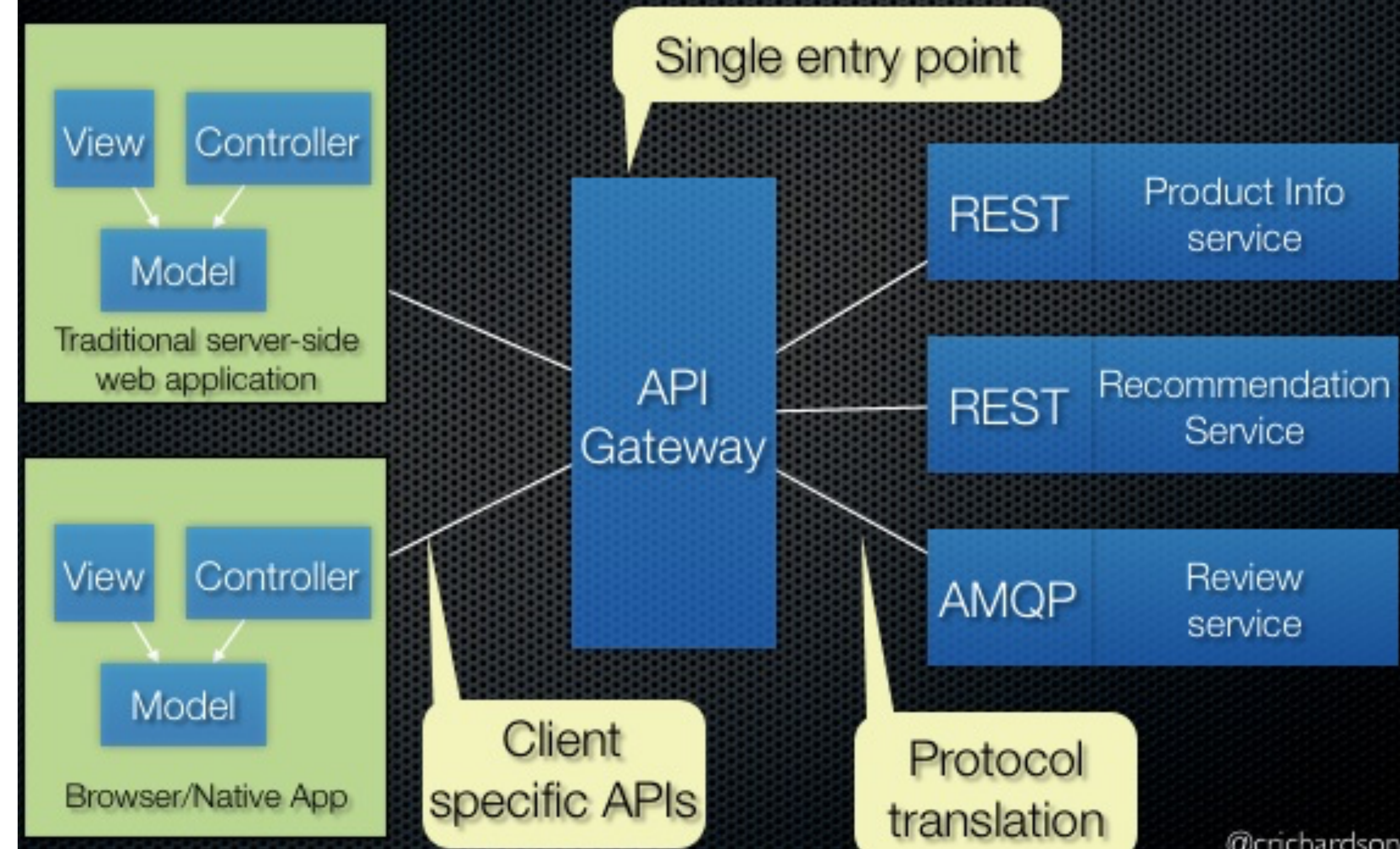
# What are the challenges of Microservices?

---

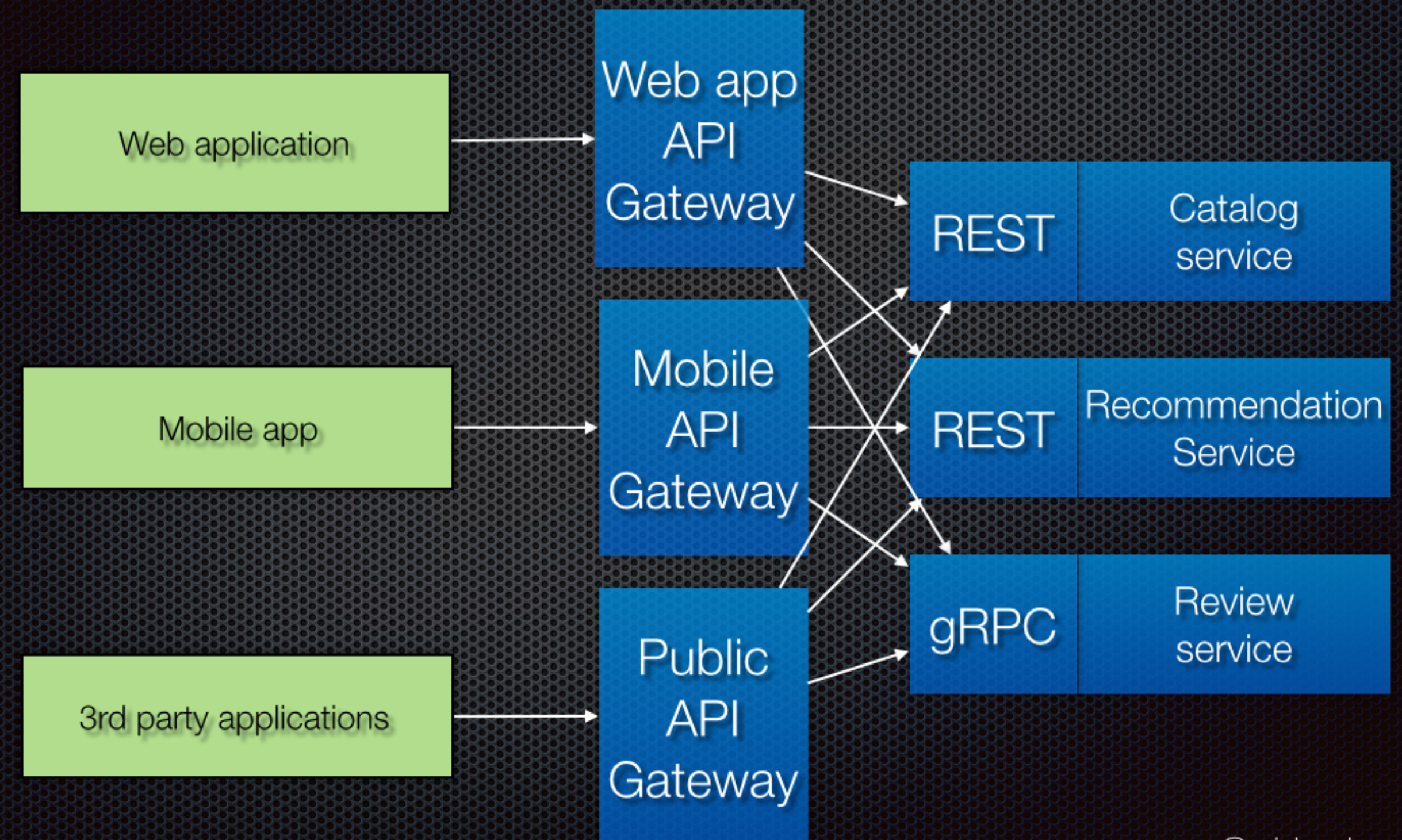
Compensating for high decentralisation. Patterns are emerging.



# Use an API gateway



# Variation: Backends for frontends



<https://microservices.io/patterns/apigateway.html>

API Gateway Pattern

Thank you NETFLIX!

The absolute minimum to remember.

Architecture is a servant of high priority stakeholder values.

Is as simple as possible, but not simpler.

Is designed to be replaceable.

Software Architecture is a Strategy.

Software Architecture is Communication.

Software Architect is a Teacher.

Feel free to connect.

