Microservices und SCS zur Architekturmodernisierung

Michael Vitz

Alexander Heusingfeld

Alexander Heusingfeld
Senior Consultant @ innoQ

alexander.heusingfeld@innoq.com

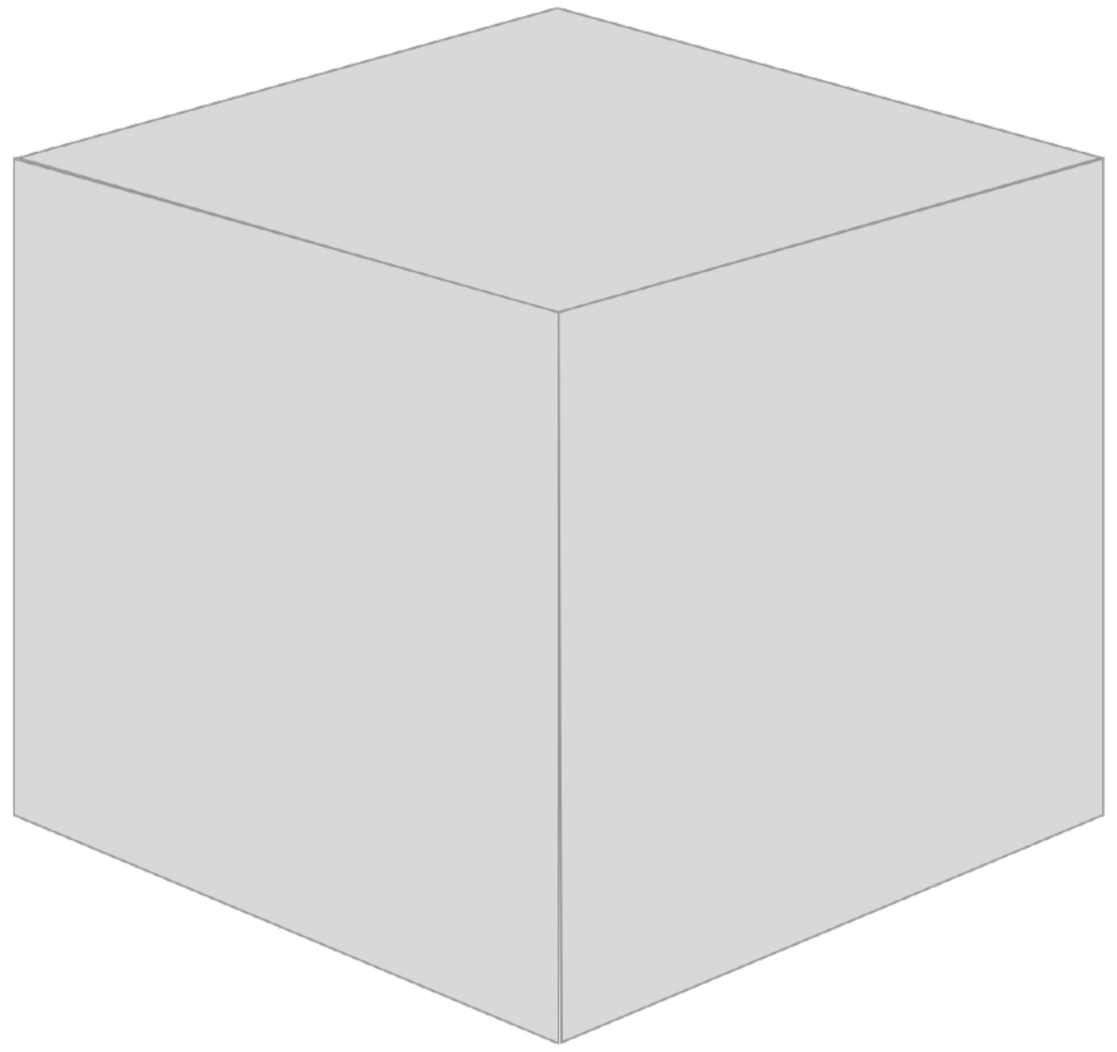
@goldstift

Michael Vitz
Senior Consultant @ innoQ

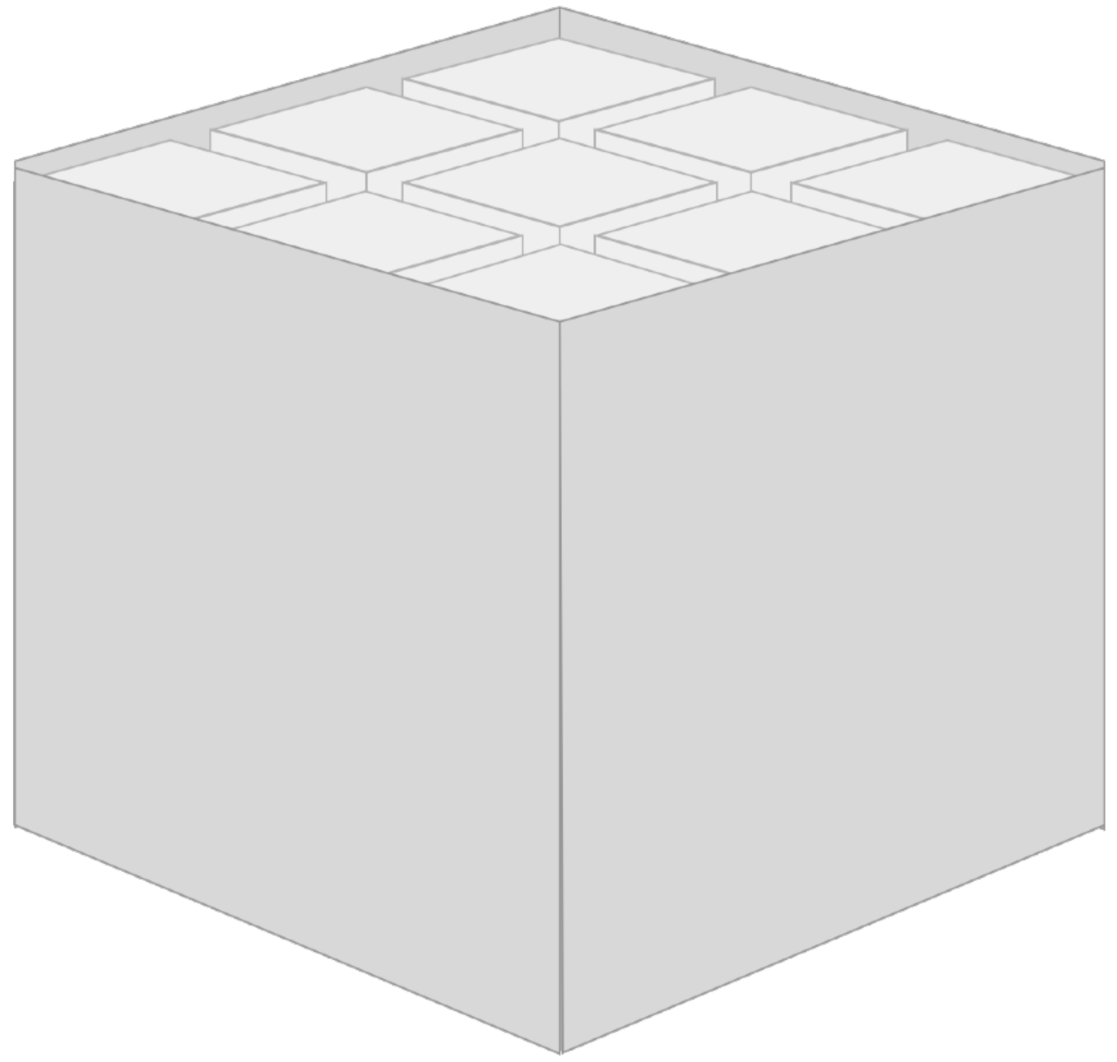
michael.vitz@innoq.com

@michaelvitz

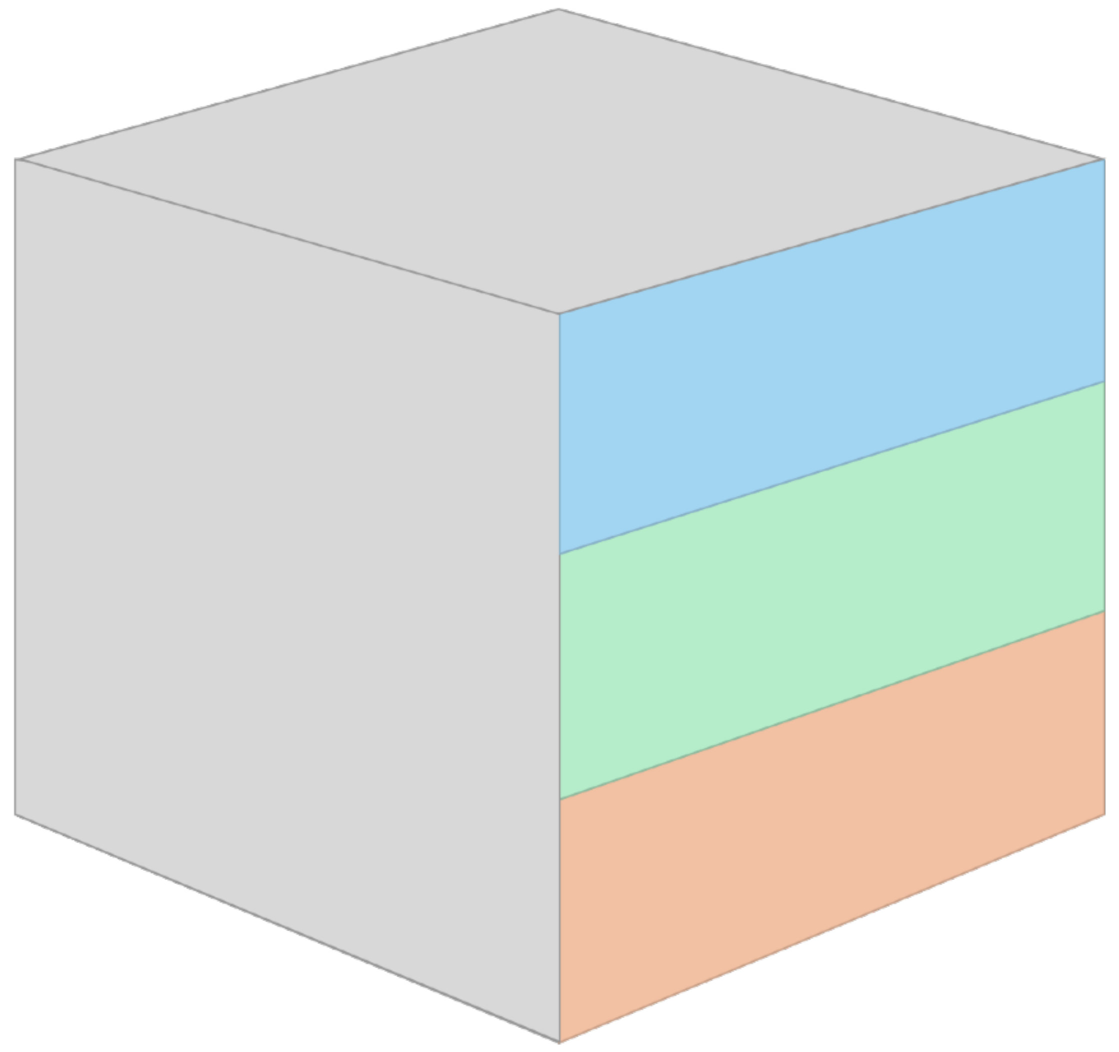
Typical Scenario?!



A monolith contains **numerous** things inside of a single system …



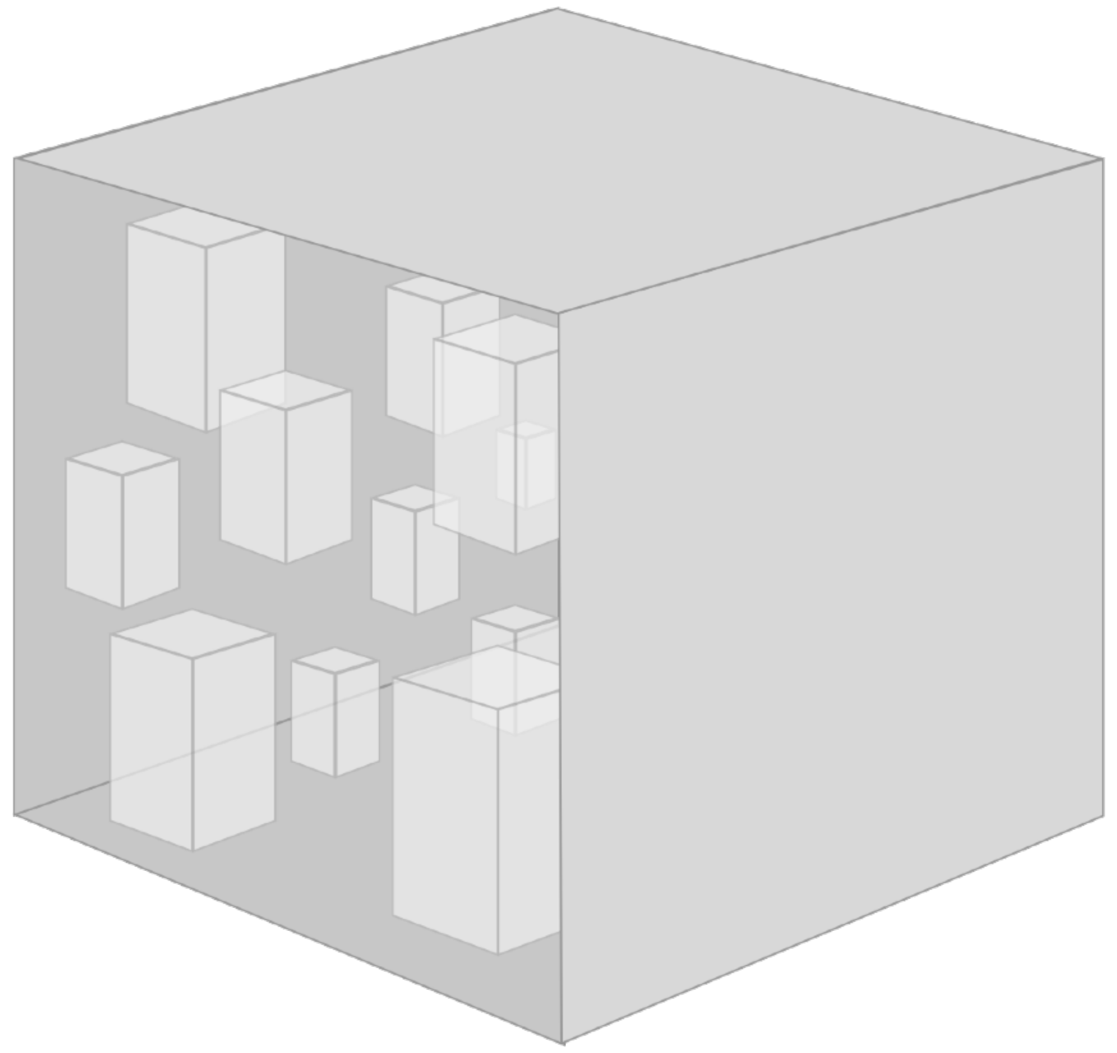
Various Domains



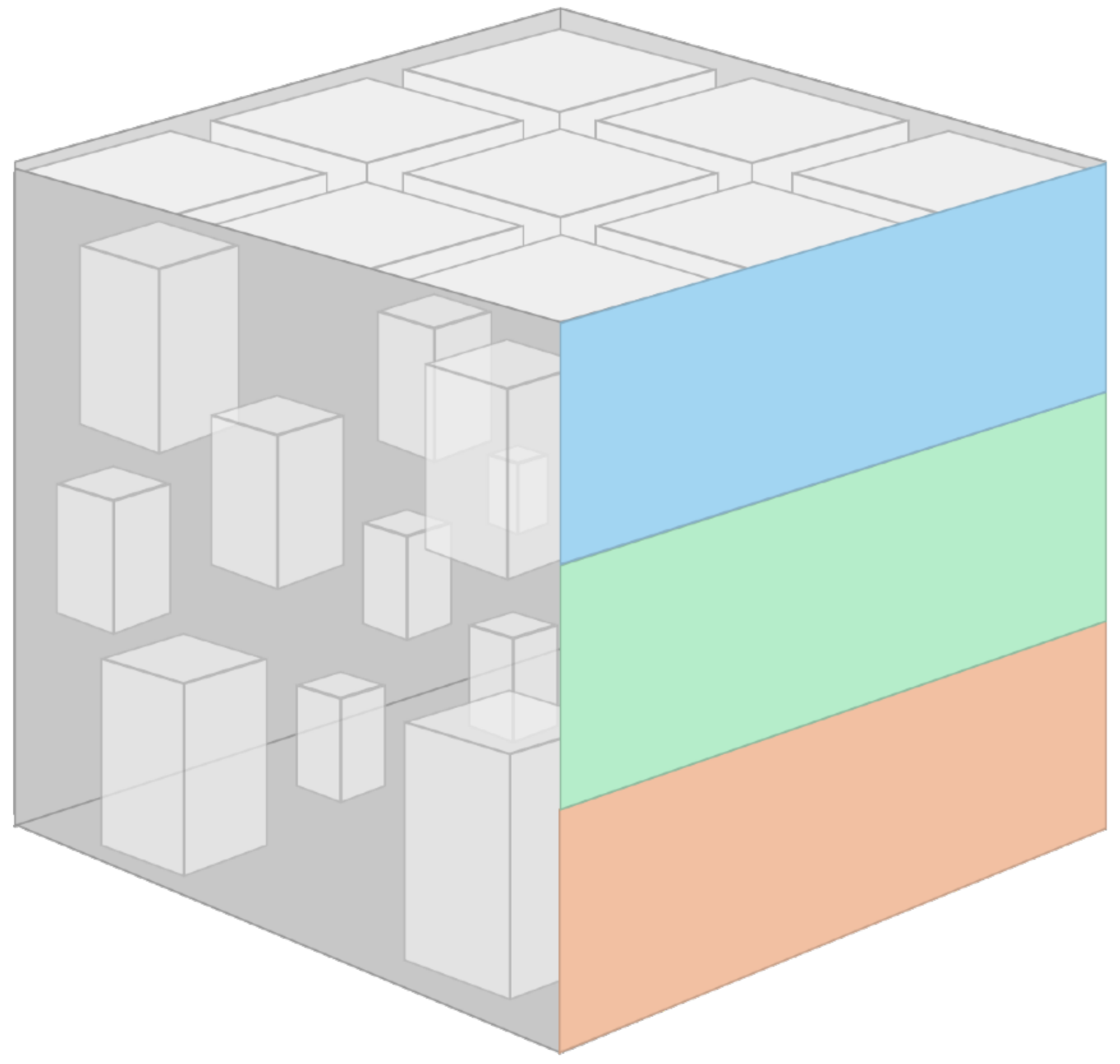
User interface

Business logic

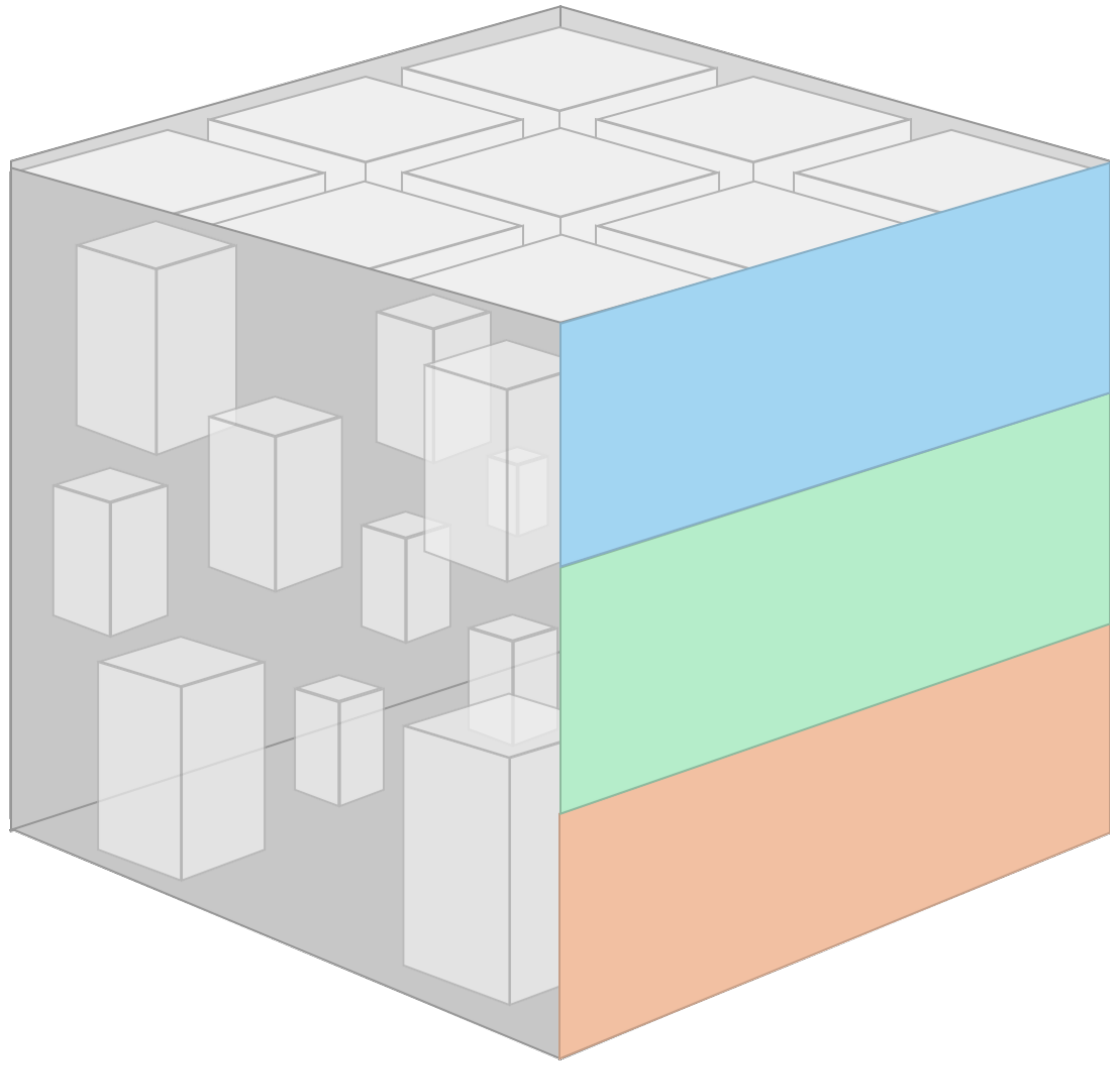
Persistence



… as well as **a lot** of modules, components, frameworks and libraries.

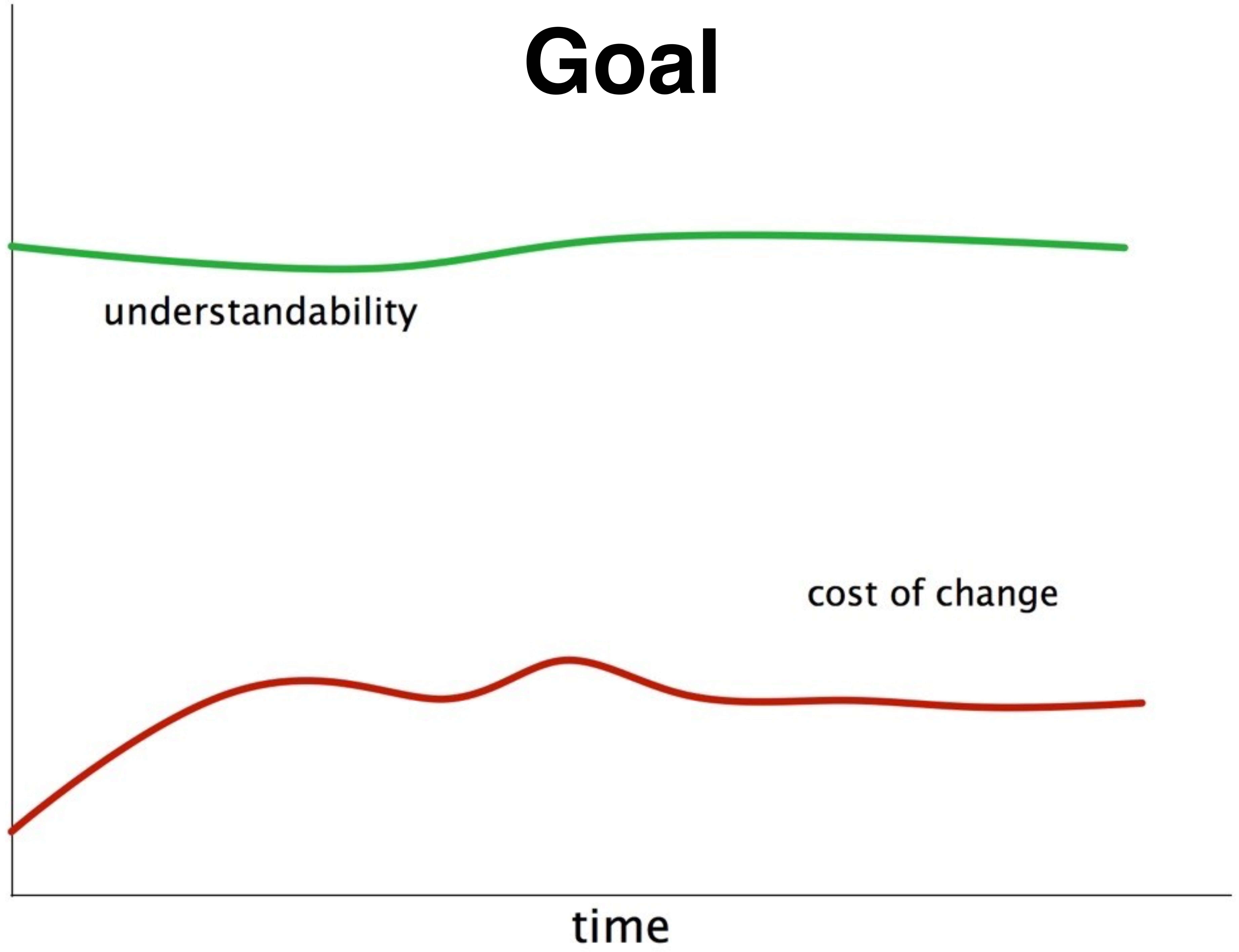


With all these layers in one place, a monolith tends to **grow**.

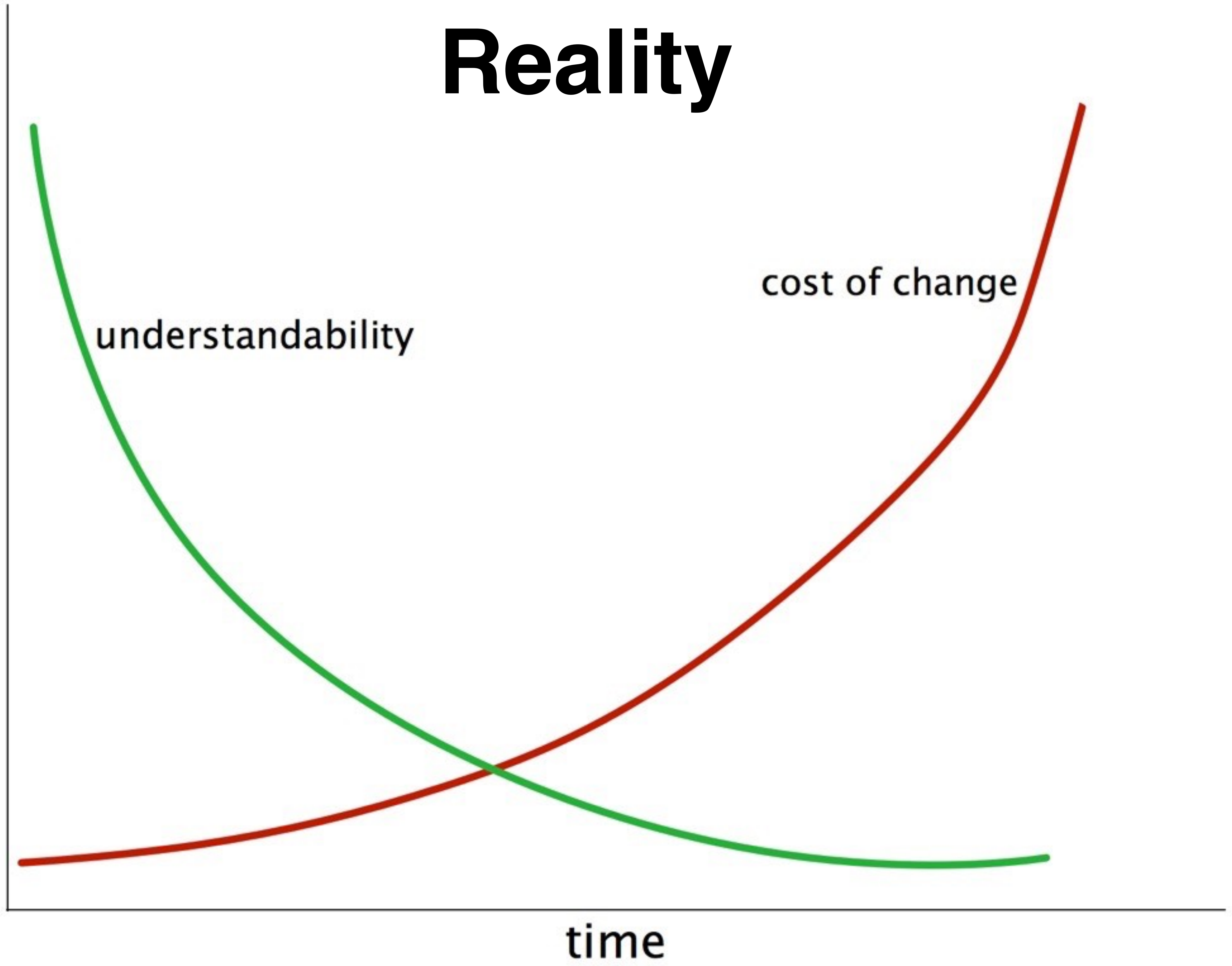


With all these layers in one place, a monolith tends to **grow**.

Goal



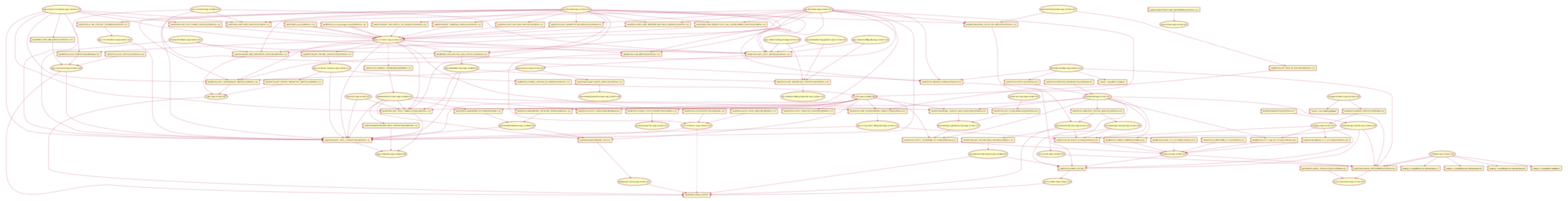
Reality



Why?

Company X App - Module dependencies

The following is a graph visualizing the dependencies between the OSGi modules in Company X Application, defined via Spring Dynamic Modules XML files.

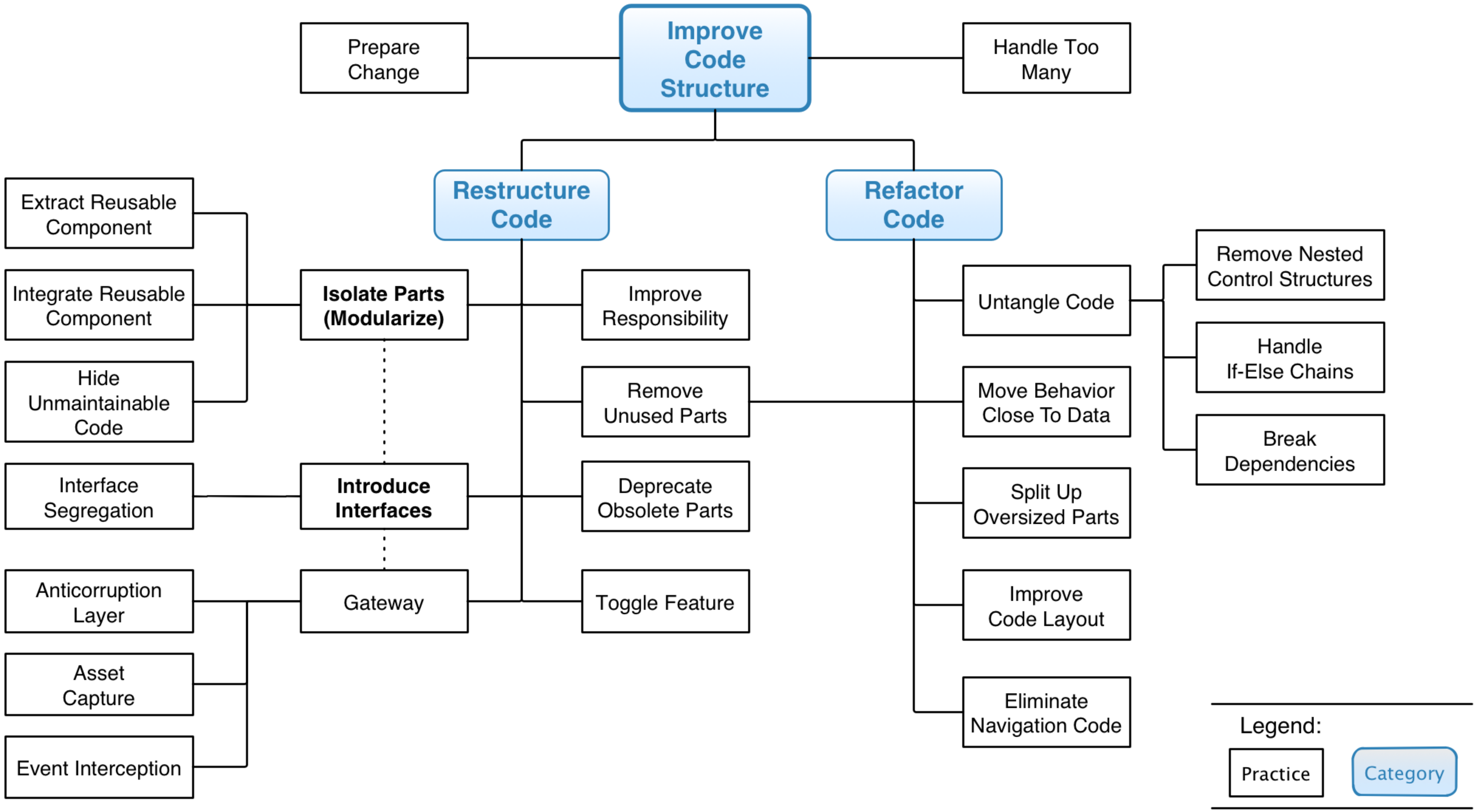


Typical Reaction?

Code Improvements

Code Improvements

improve



Alternatives?

Focus on Technology

Focus on ~~Technology~~

Focus on ~~Technology~~ Business Value

Thesis:

of Systems

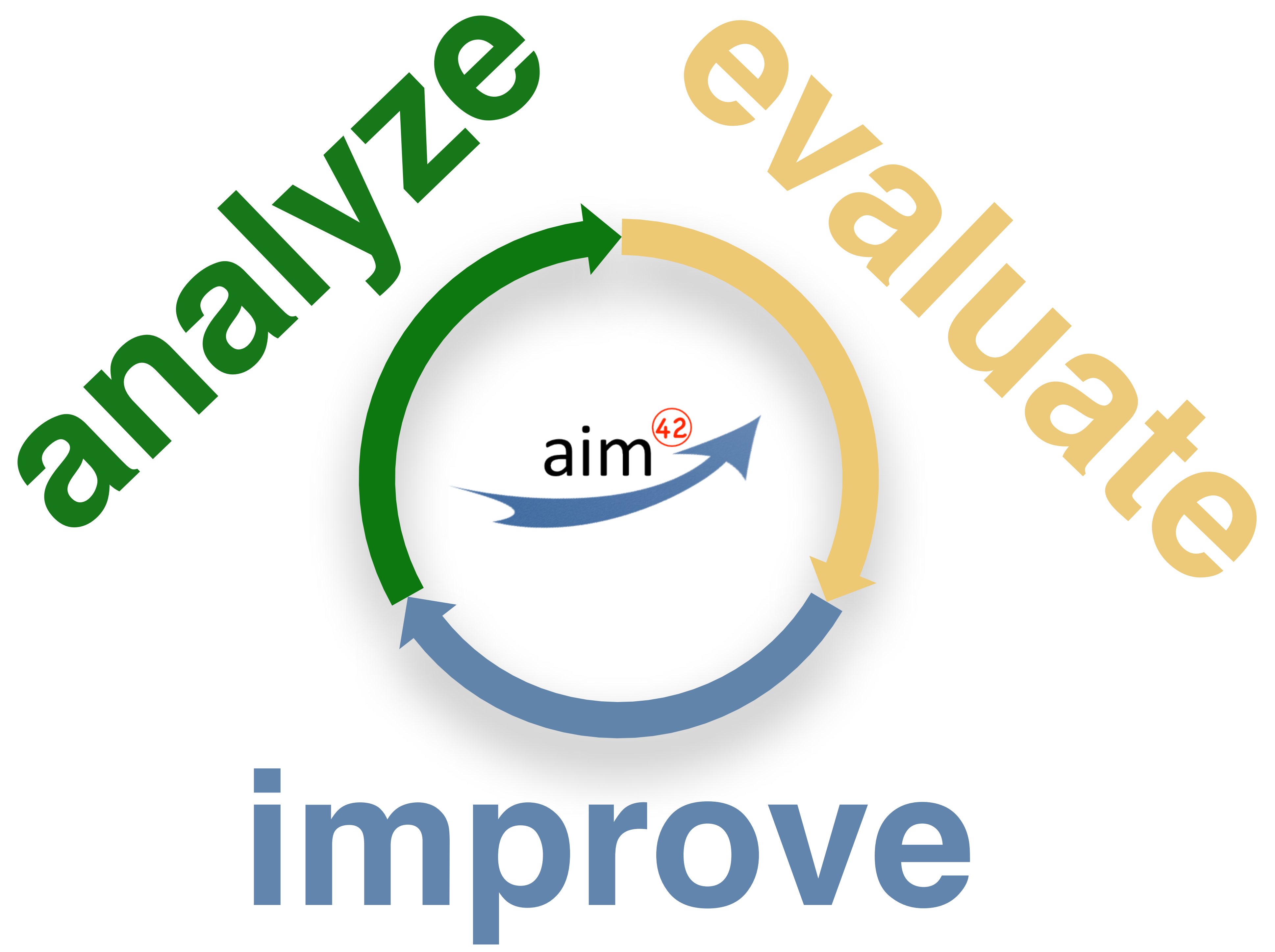
Improvement

of single classes

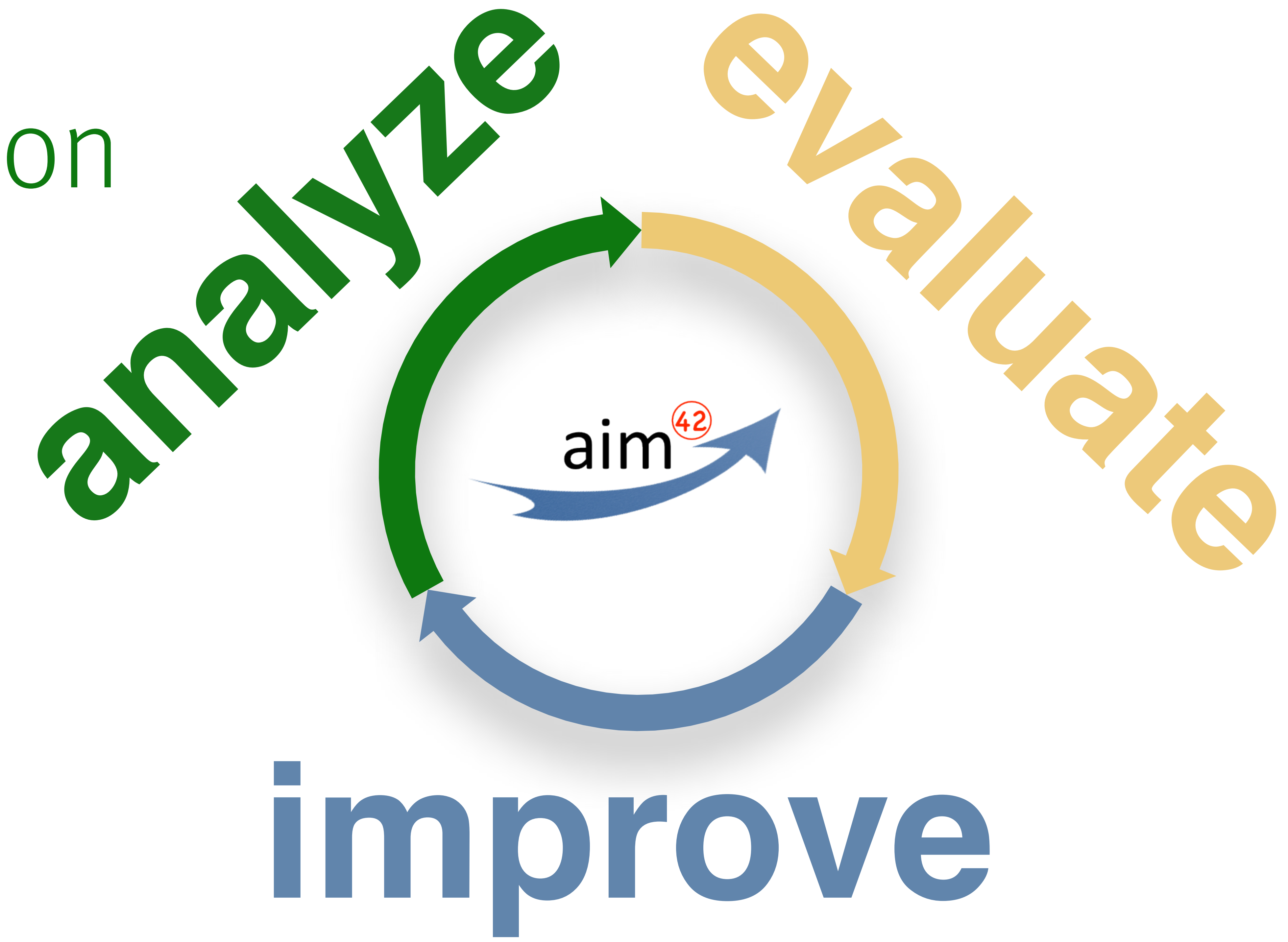
is more than Refactoring

aim42

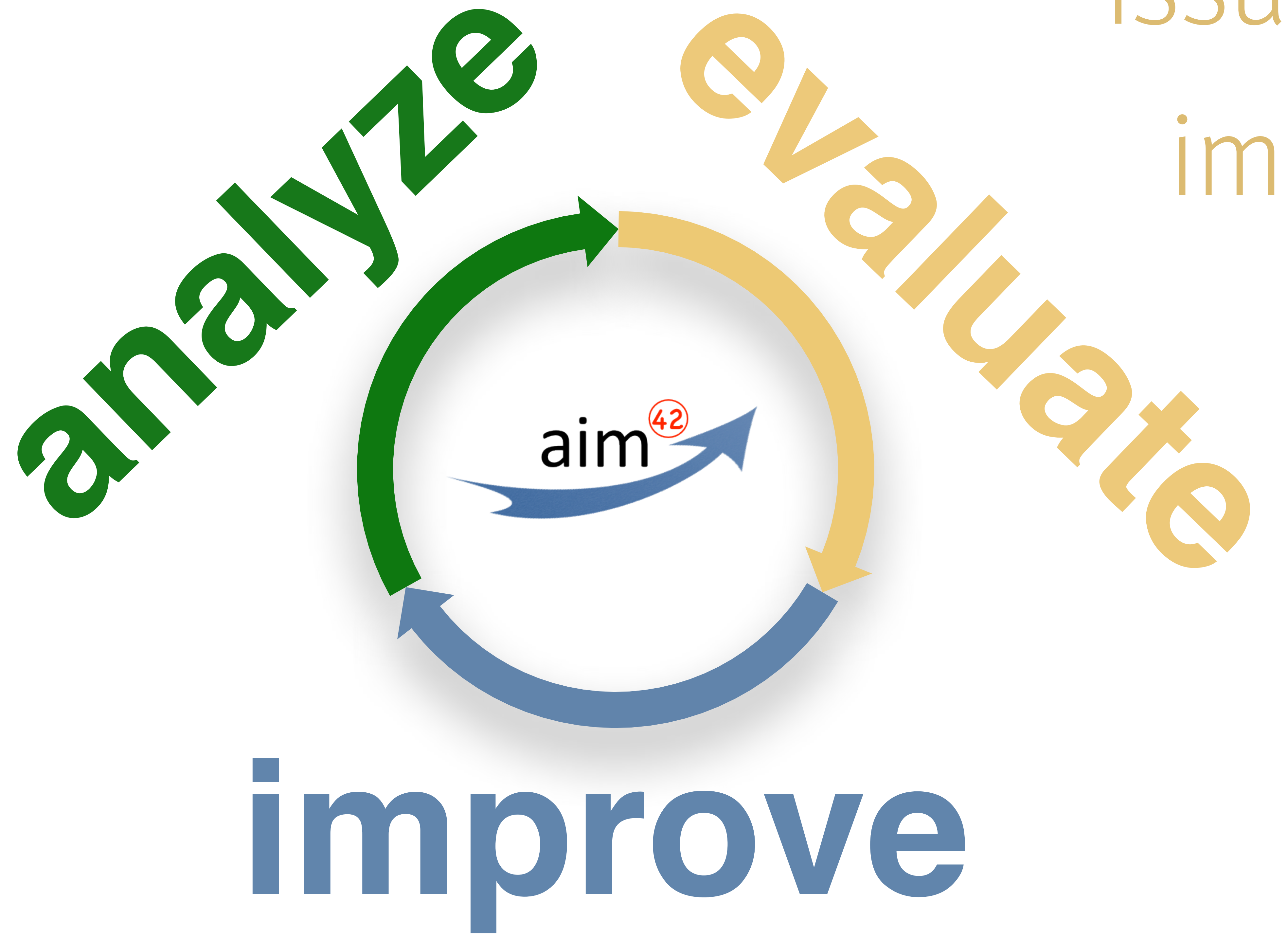
Architecture Improvement Method

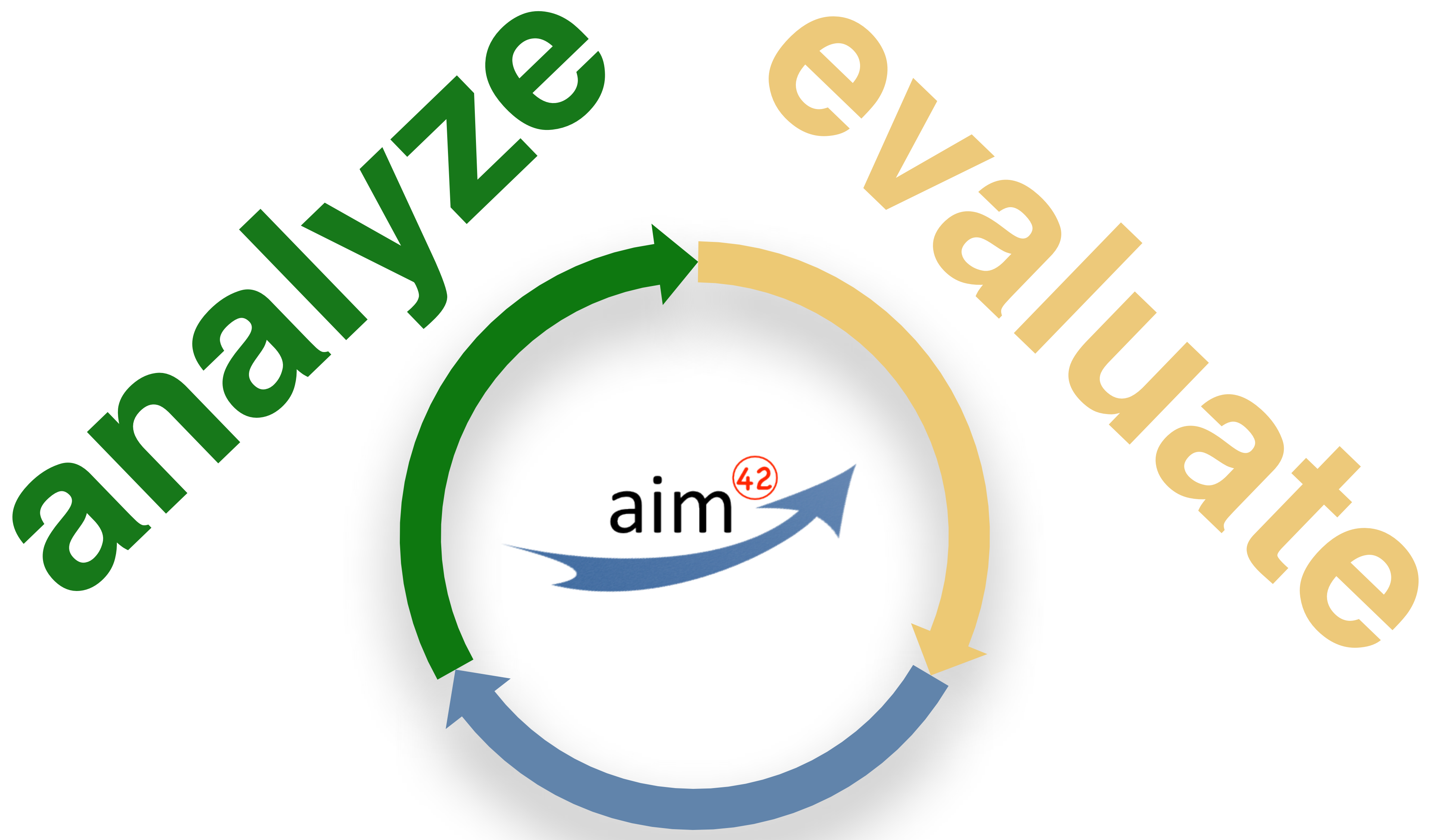

analyze
evaluate
improve
aim42

- architecture
- code
- runtime
- organization



determine „value“ of problems / risks / issues <u>and</u> their improvements



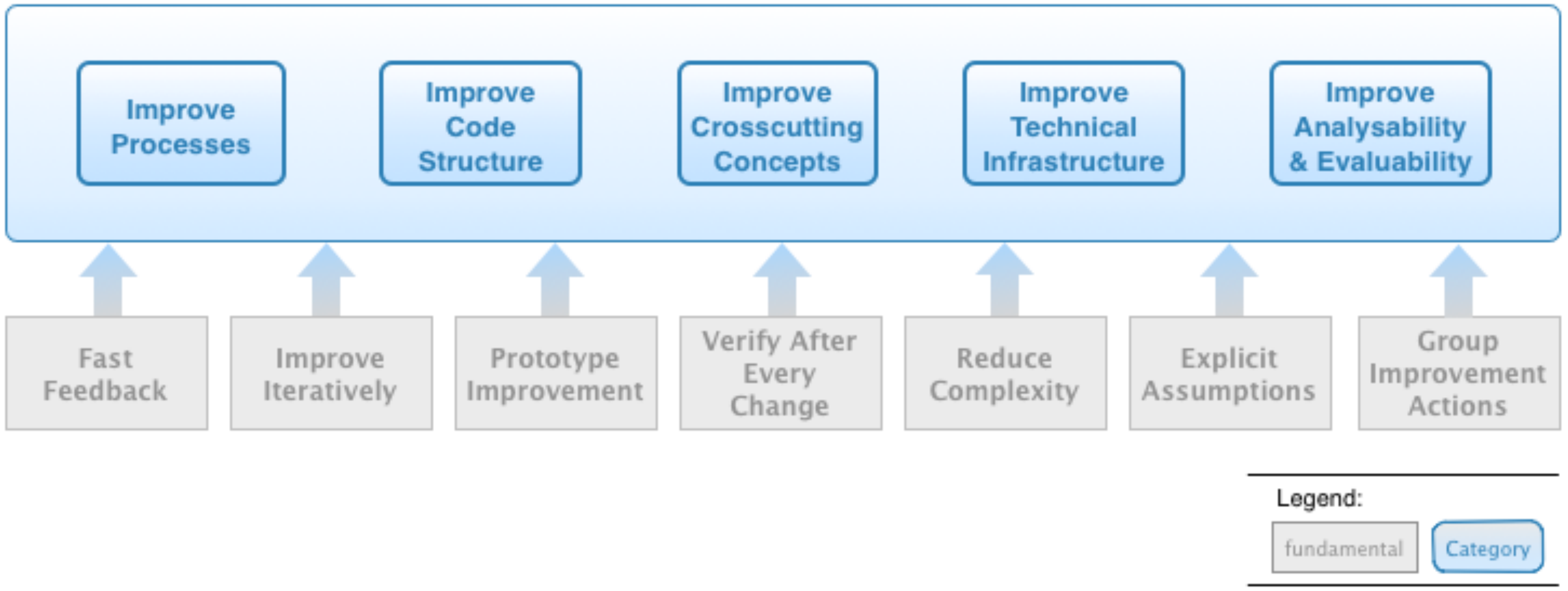


improve

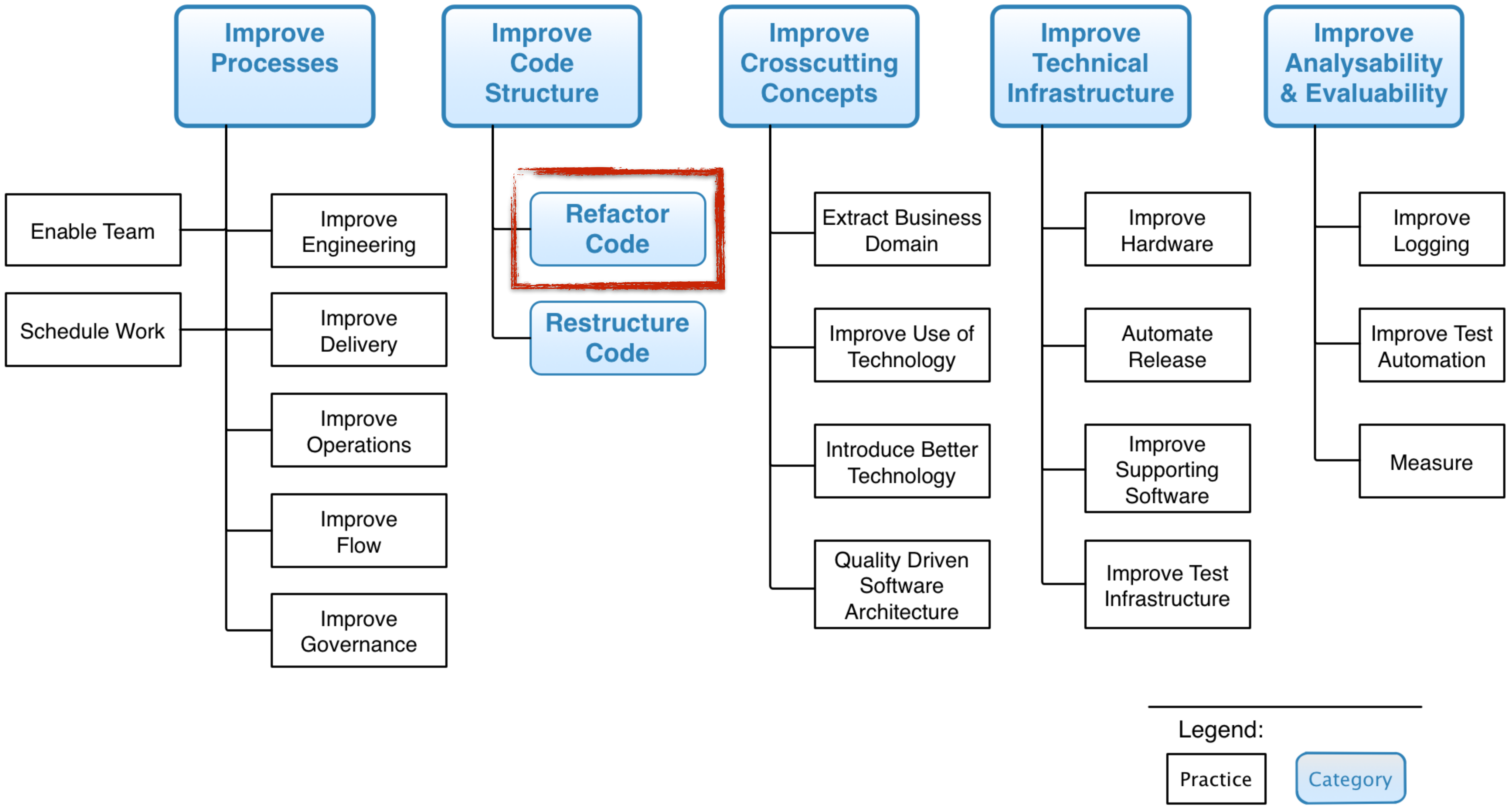
- define improvement strategy
- refactor
- re-architect
- re-organize
- remove debt

improve

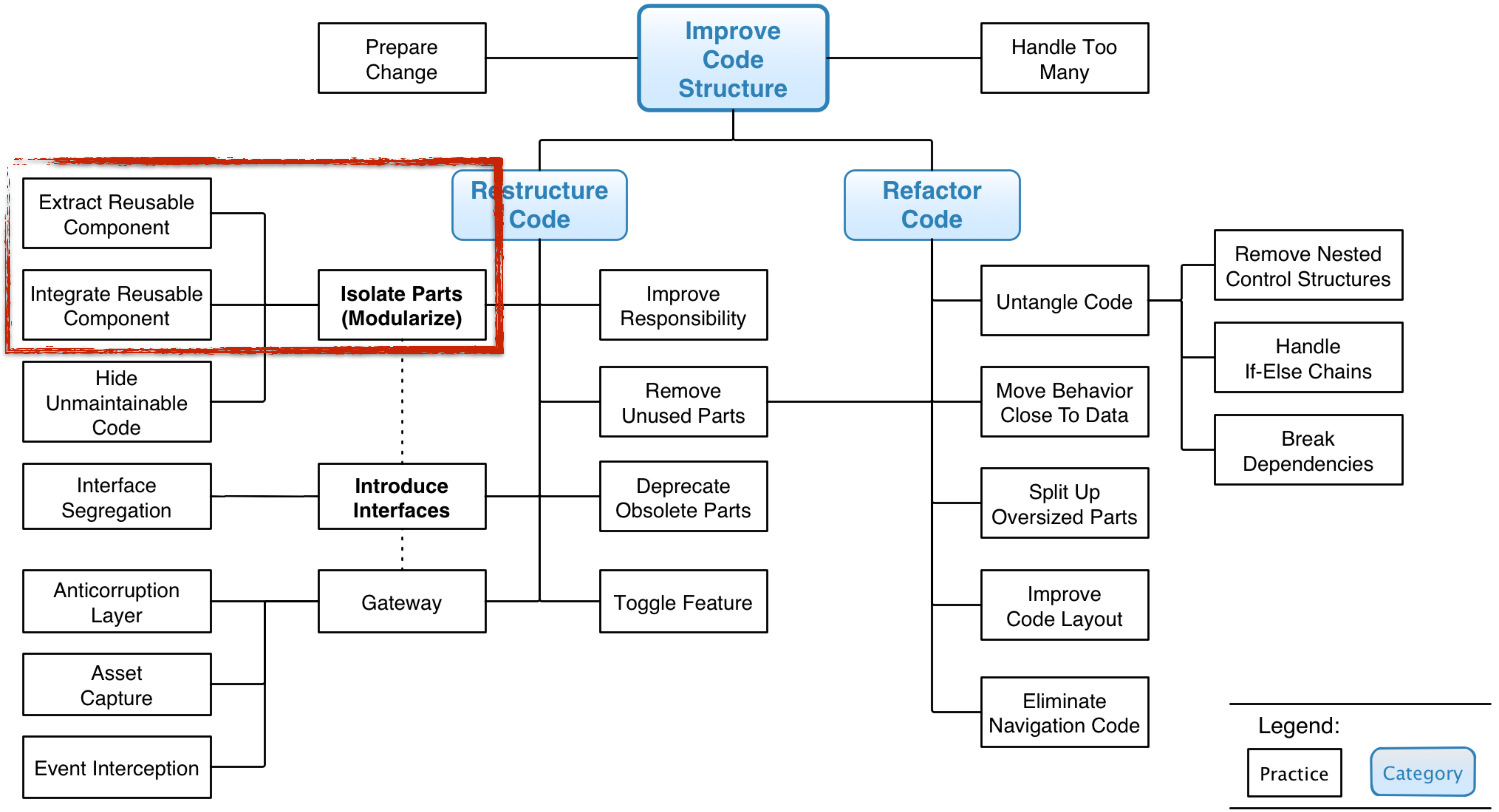
Fundamentals

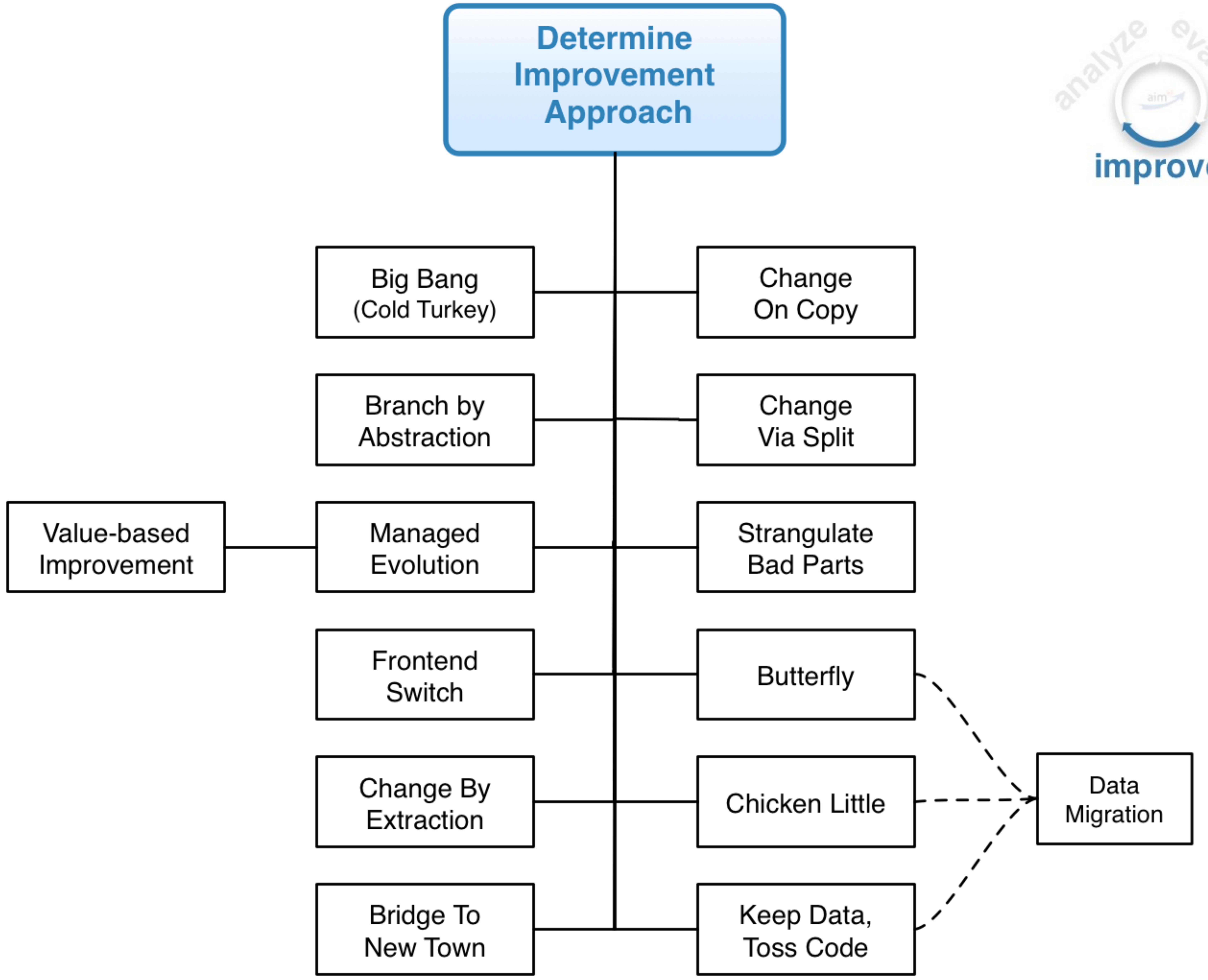


Practices


Improve Processes
Enable Team
Schedule Work
Improve Engineering
Improve Delivery
Improve Operations
Improve Flow
Improve Governance
Improve Code Structure
Refactor Code
Restructure Code
Improve Crosscutting Concepts
Extract Business Domain
Improve Use of Technology
Introduce Better Technology
Quality Driven Software Architecture
Improve Technical Infrastructure
Improve Hardware
Automate Release
Improve Supporting Software
Improve Test Infrastructure
Improve Analysability & Evaluability
Improve Logging
Improve Test Automation
Measure
Legend:
Practice
Category

Practices


Improve Code Structure
Prepare Change
Handle Too Many
Restructure Code
Refactor Code
Extract Reusable Component
Integrate Reusable Component
Hide Unmaintainable Code
Isolate Parts (Modularize)
Interface Segregation
Introduce Interfaces
Anticorruption Layer
Asset Capture
Event Interception
Gateway
Improve Responsibility
Remove Unused Parts
Deprecate Obsolete Parts
Toggle Feature
Untangle Code
Move Behavior Close To Data
Split Up Oversized Parts
Improve Code Layout
Eliminate Navigation Code
Remove Nested Control Structures
Handle If-Else Chains
Break Dependencies
Legend:
Practice
Category



analyze evaluate improve

A smaller Codebase makes things easier

introduce explicit boundaries

Just use Microservices

- Everyone’s doing Microservices, so you should, too
- Everything will be faster with Microservices
- There are lots of interesting tools to play with, much more interesting than the boring business domain
- With Microservices we’ll be more agile

Just use Microservices

- Everyone's doing Microservices, so you should, too
- Everything will be faster with Microservices
- There are lots of interesting tools to play with, much more interesting than the boring business domain
- With Microservices we'll be more agile

Business Value?

Microservice Characteristics

small

each running in its own process

lightweight communicating mechanisms (often HTTP)

built around business capabilities

independently deployable

mininum of centralized management

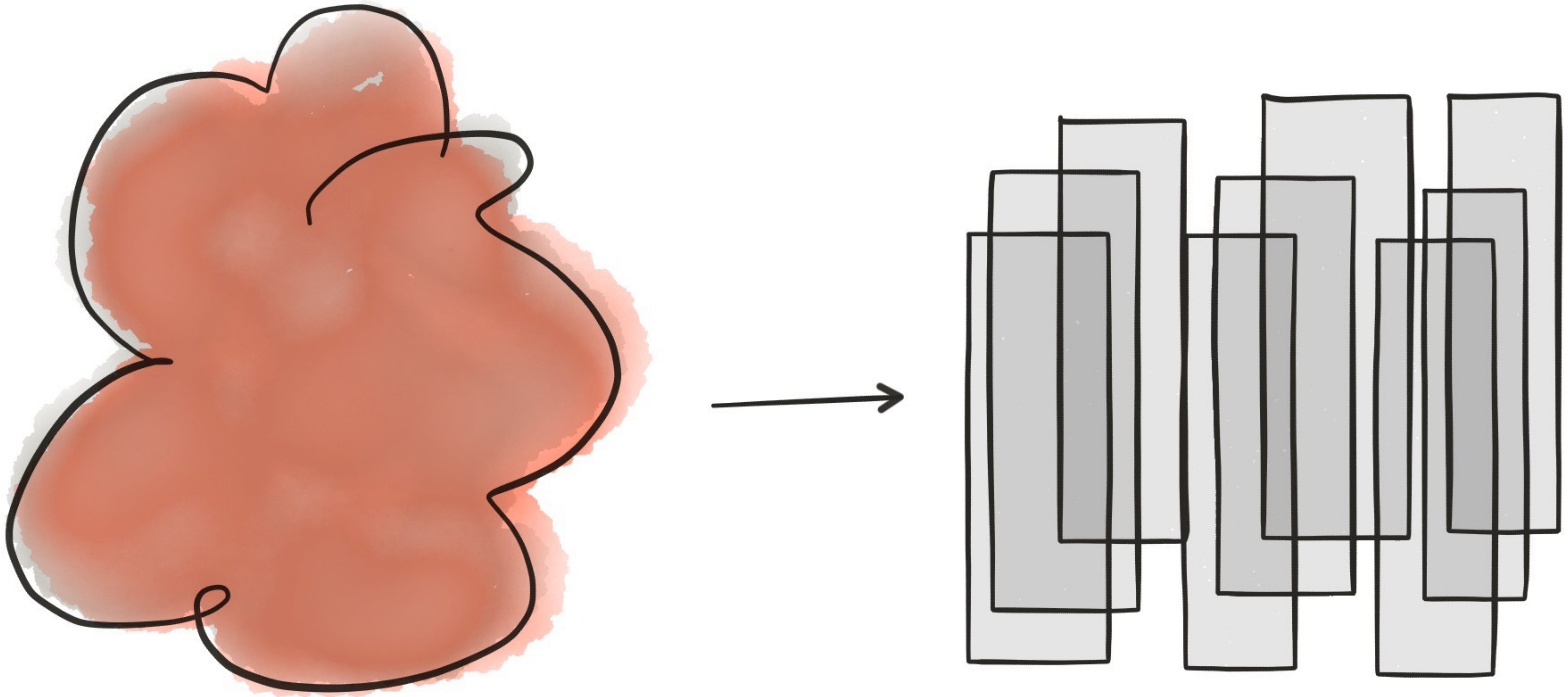
may be written in different programming languages

may use different data storage technologies

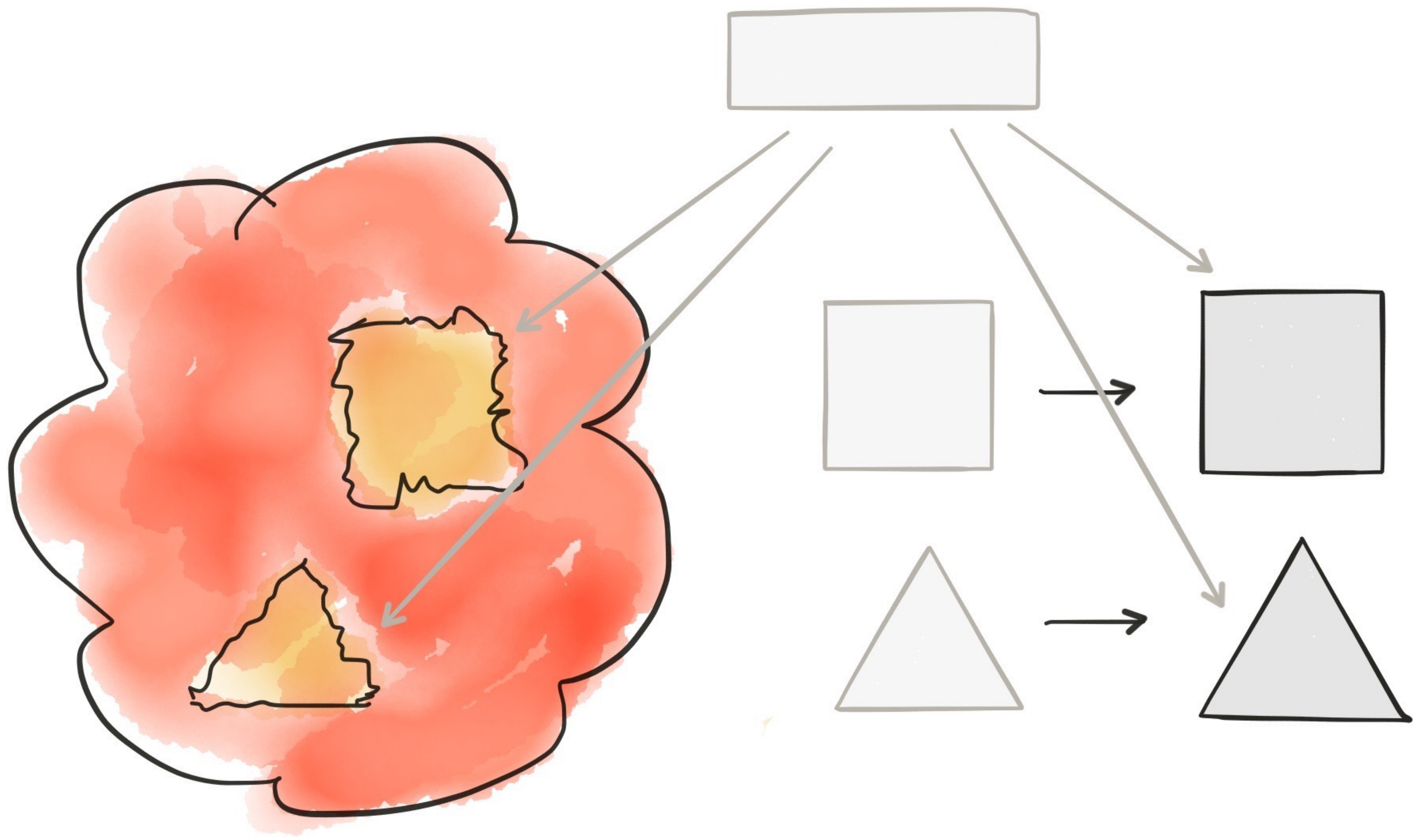
http://martinfowler.com/articles/microservices.html

Improvement Approaches applied

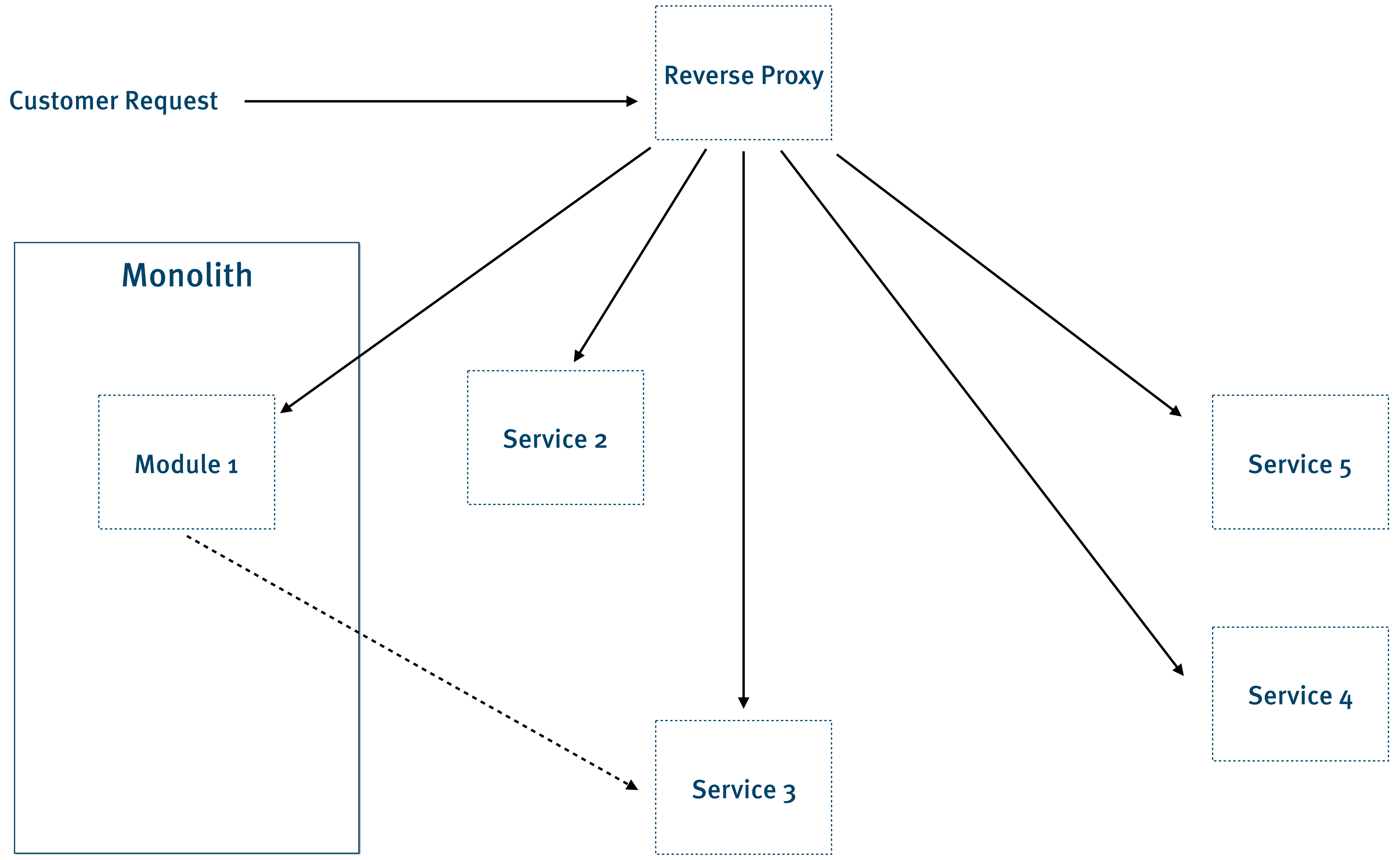
Big Bang



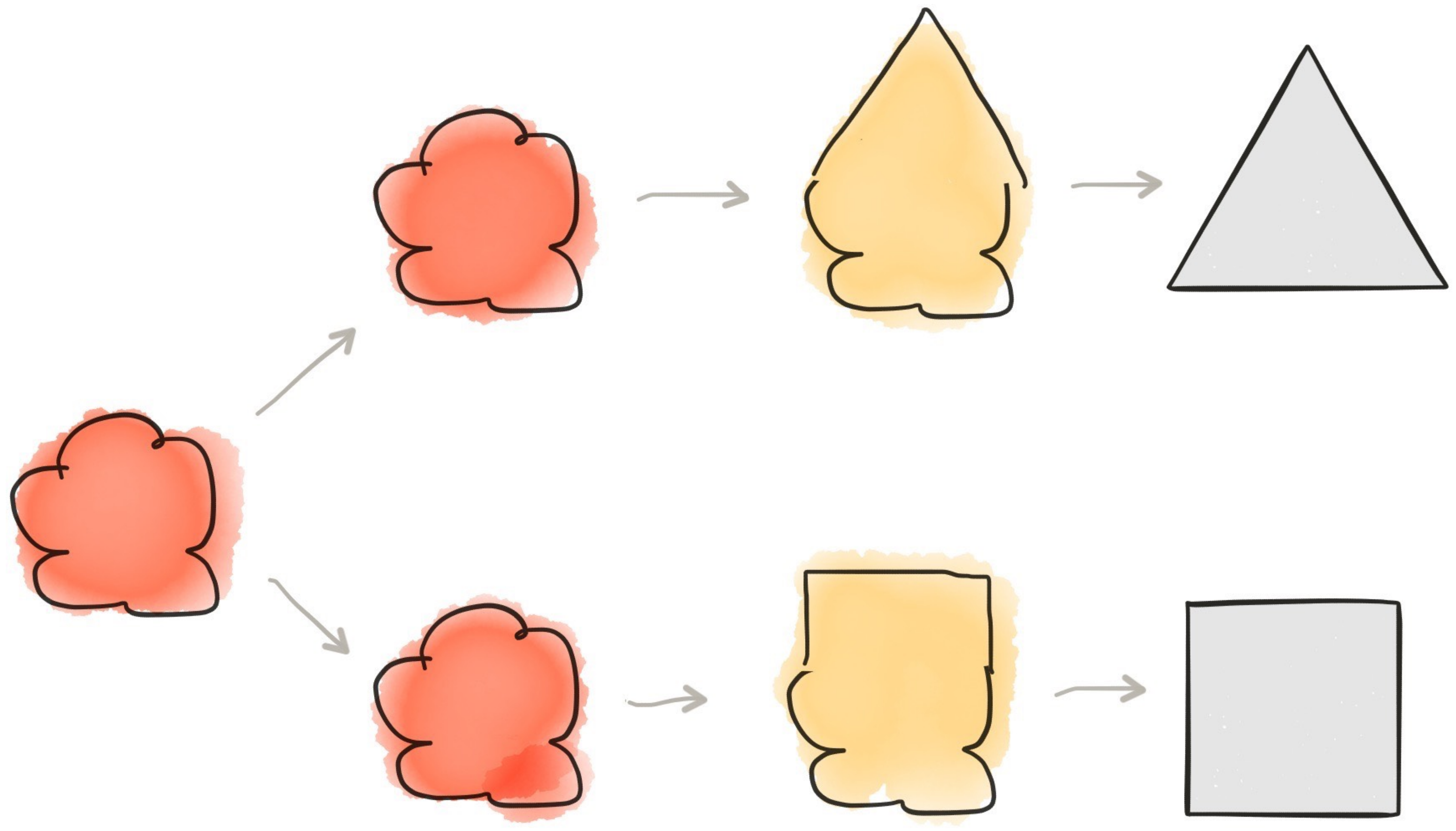
Frontend Switch



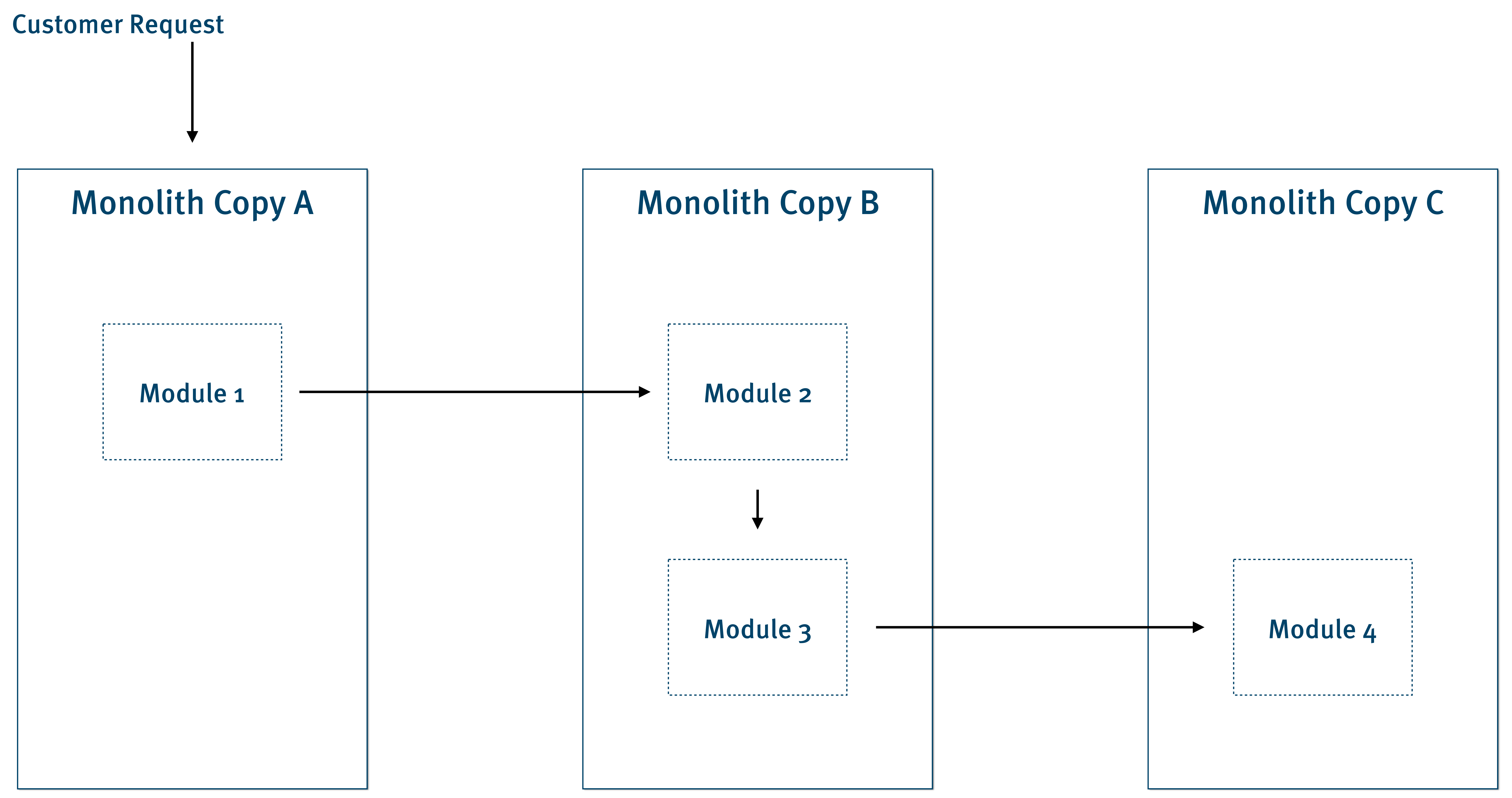
Frontend Switch



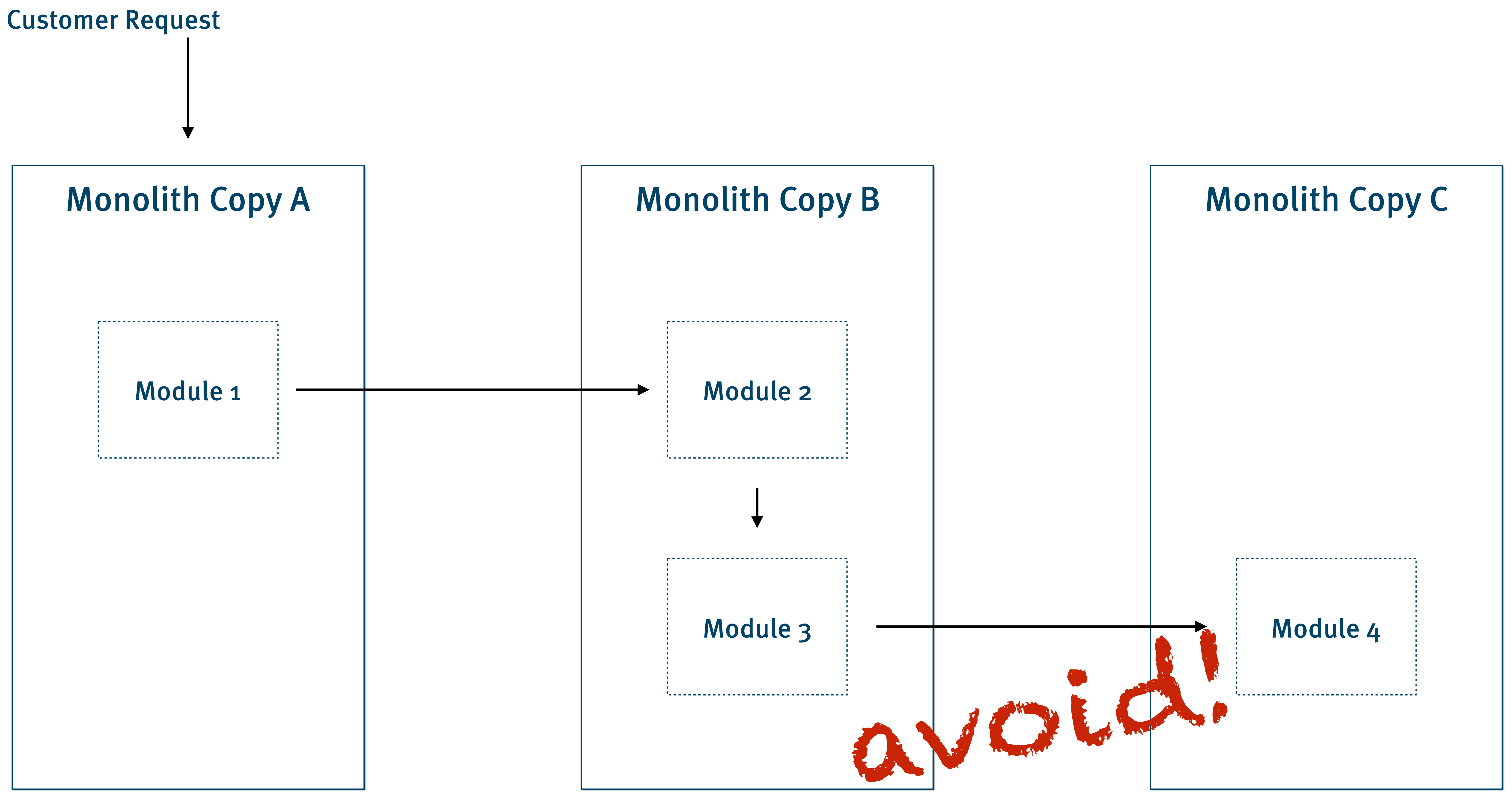
Change on Copy



Request Cascades

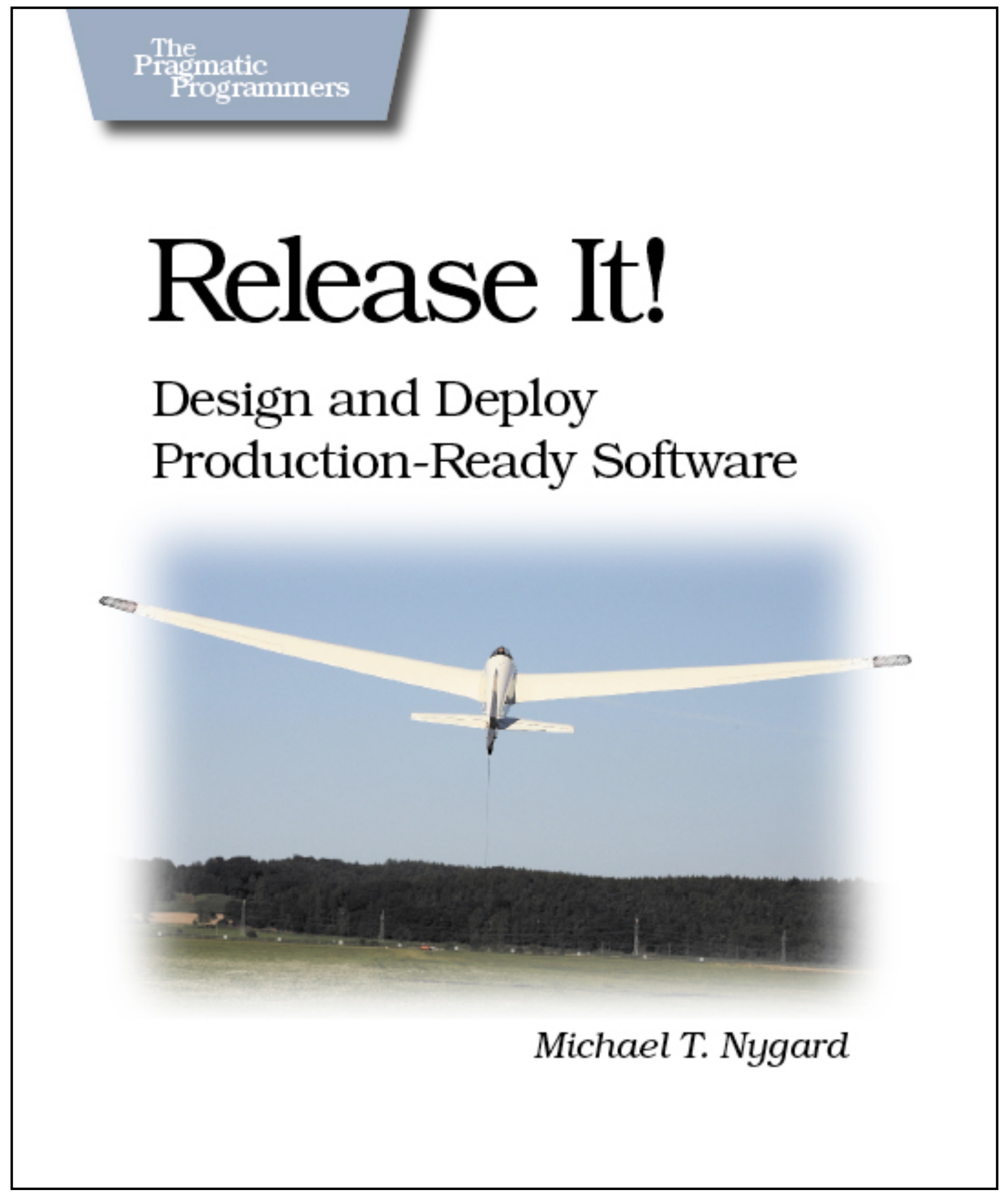


Request Cascades

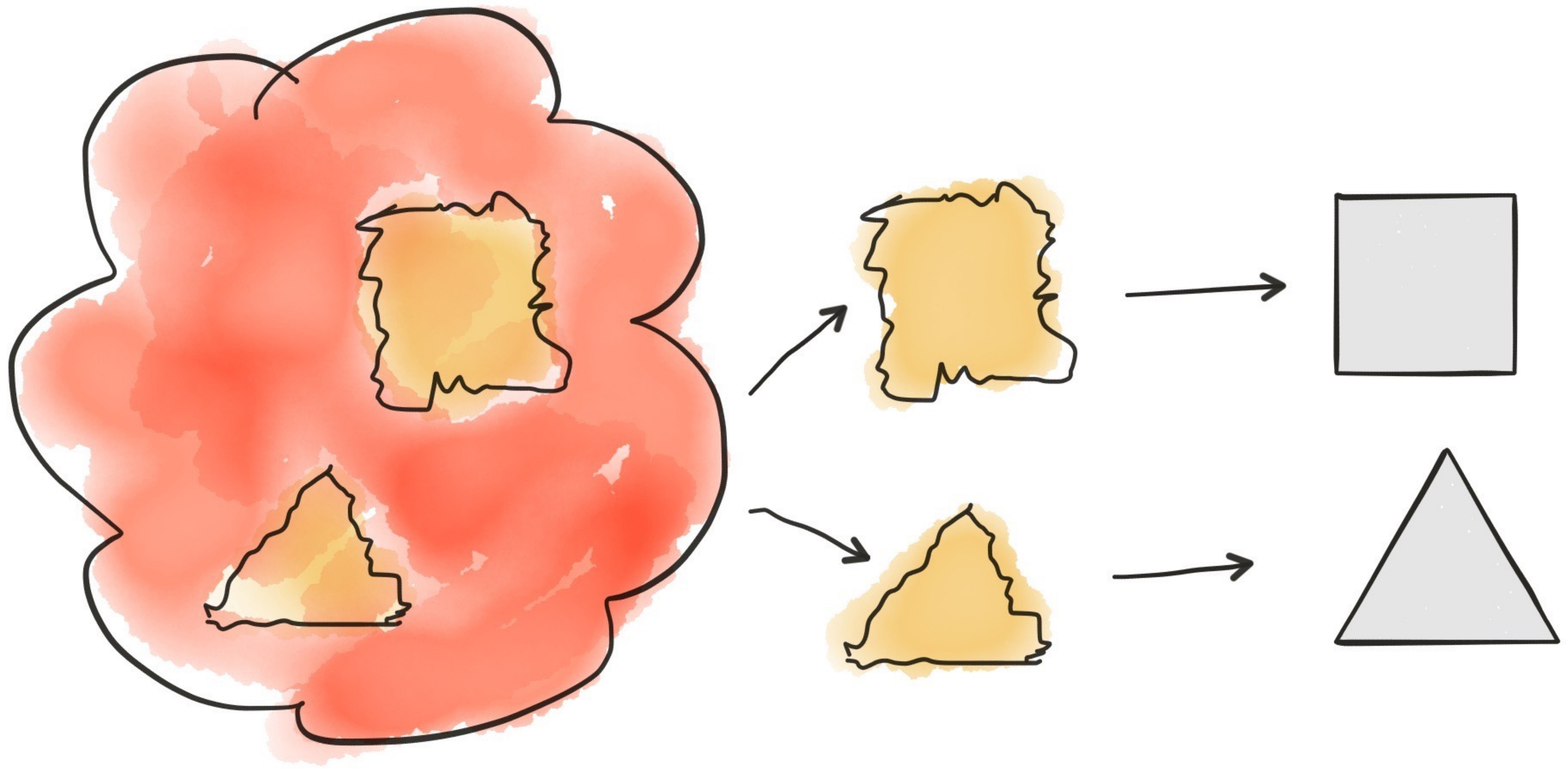


Resilience

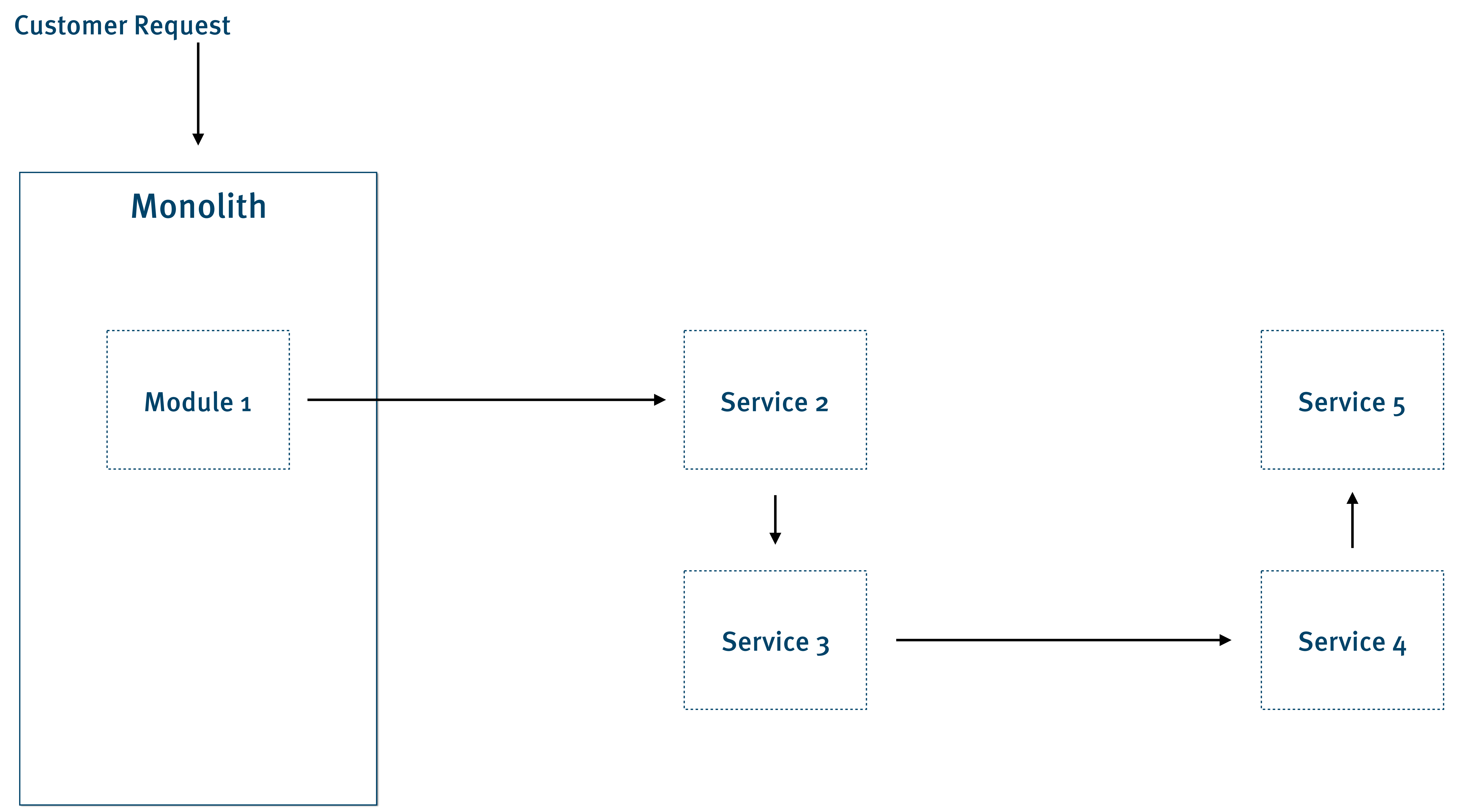
- isolate Failure
- apply graceful degradation
- be responsive in case of failure


The Pragmatic Programmers
Release It!
Design and Deploy Production-Ready Software
Michael T. Nygard

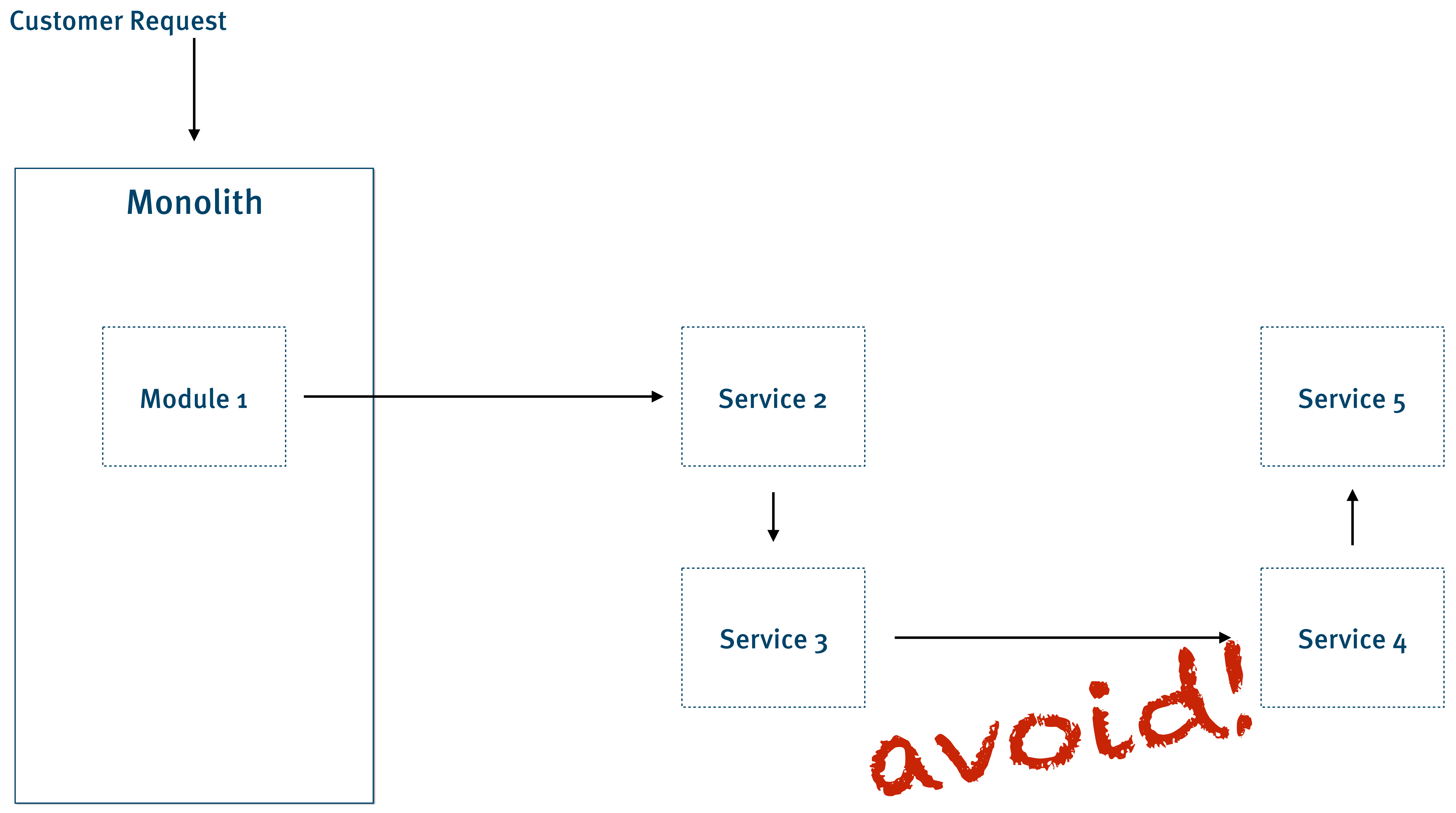
Change via Extraction



Request Cascades

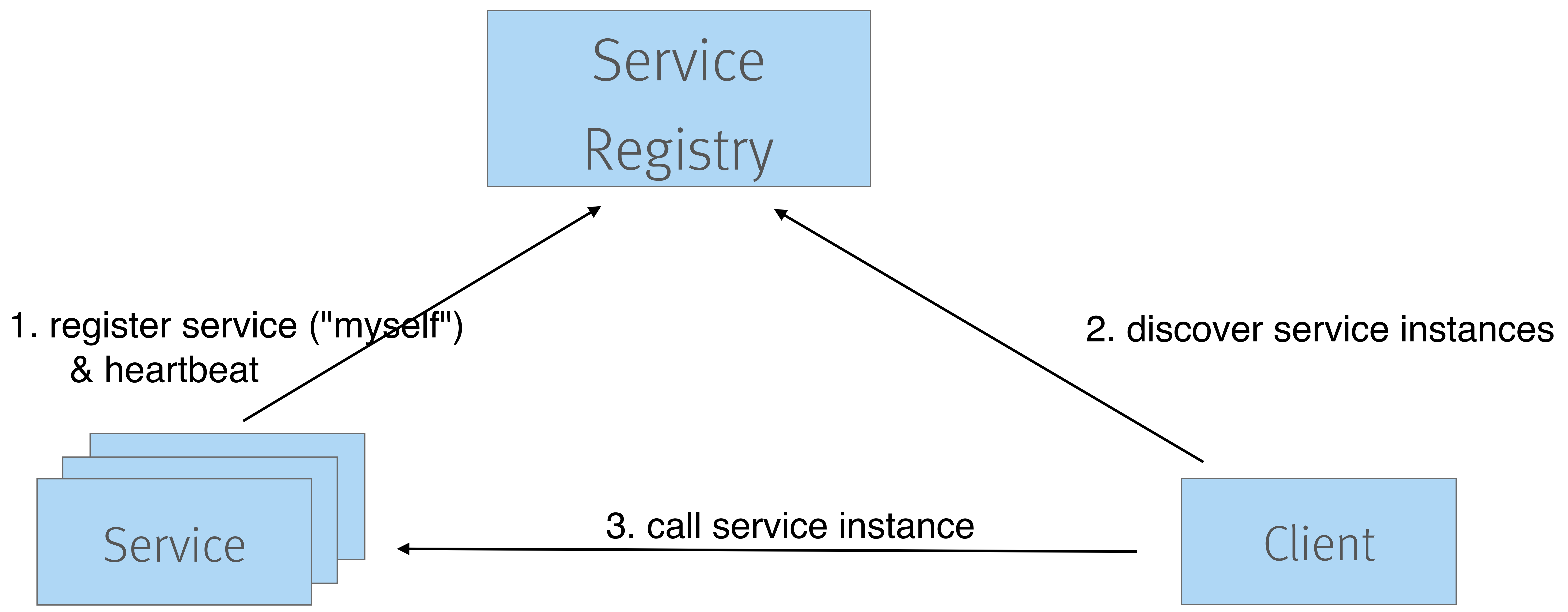


Request Cascades

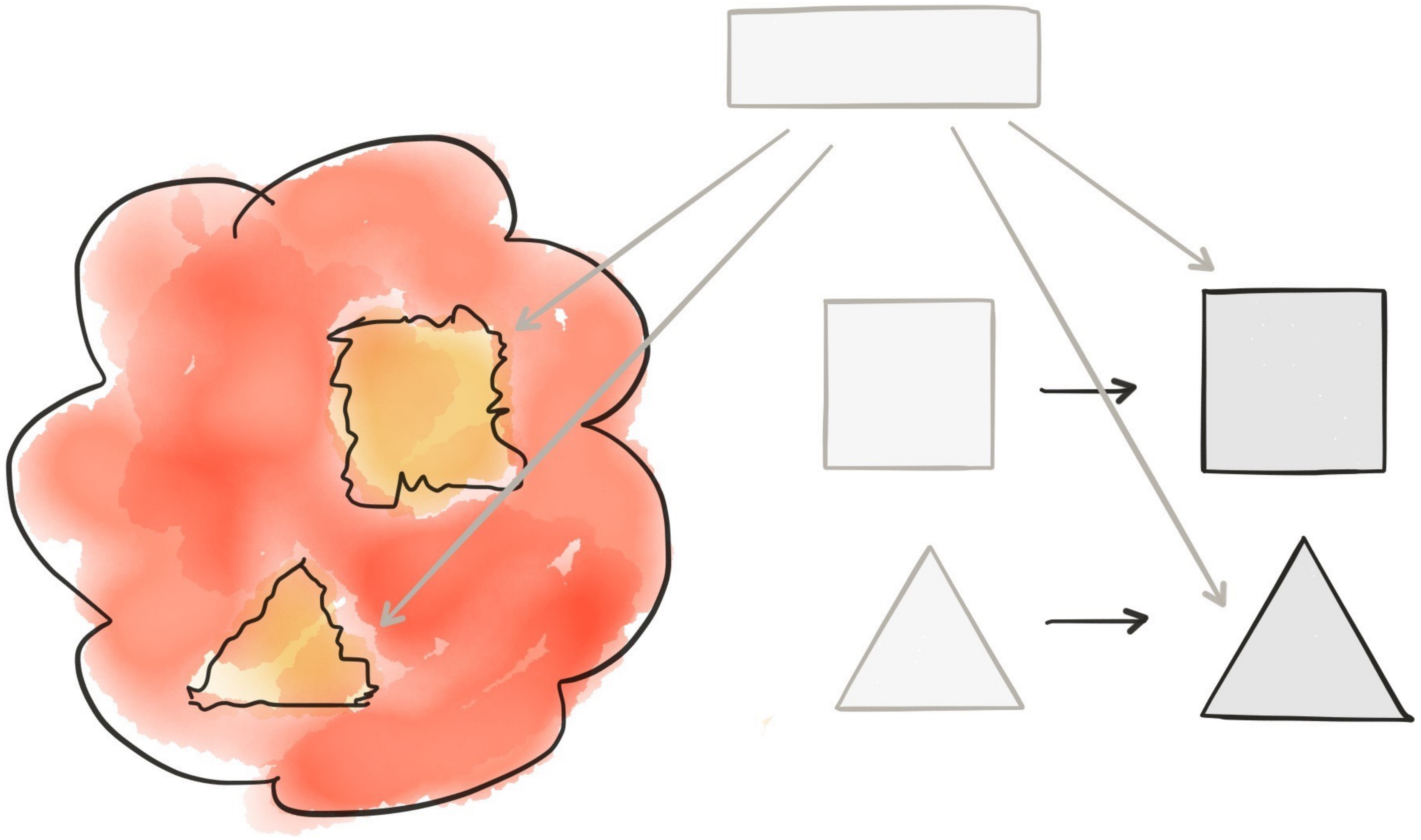


Request Cascades Lower Availability

Service Discovery

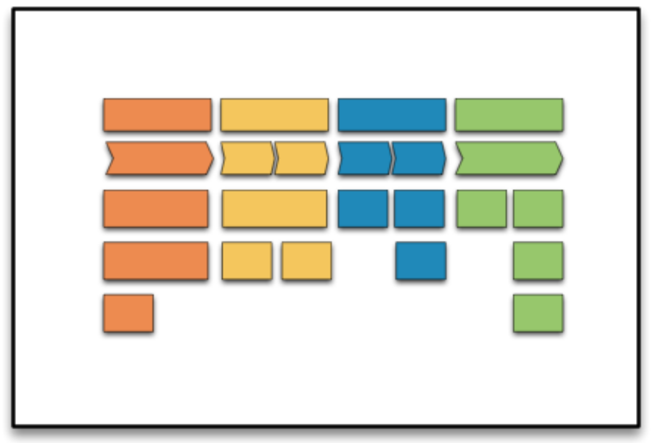

Service Registry
1. register service ("myself") & heartbeat
2. discover service instances
3. call service instance
Service
Client

Strangulate Bad Parts

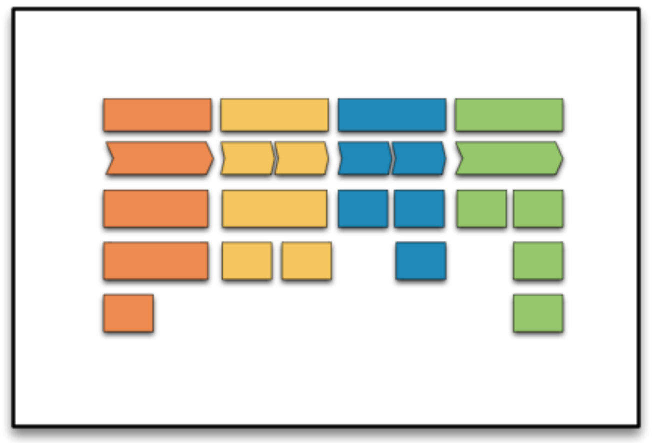


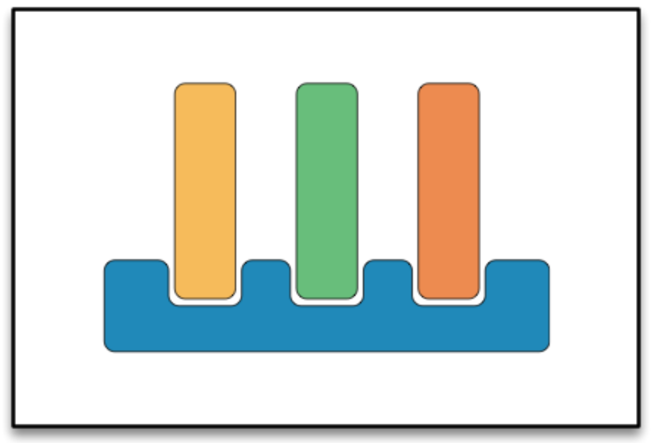
Architectural Decisions

Architectural Decisions



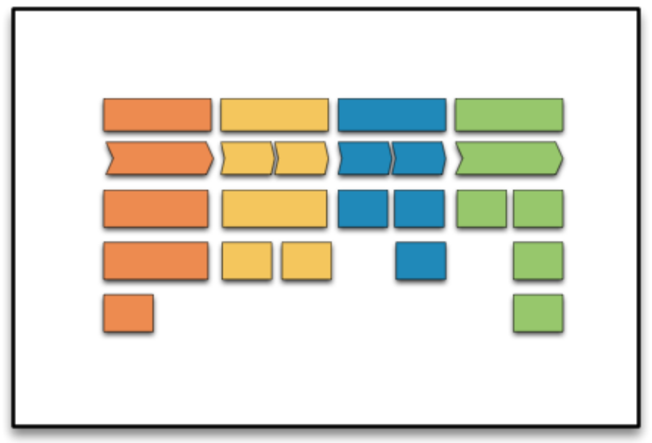
Architectural Decisions



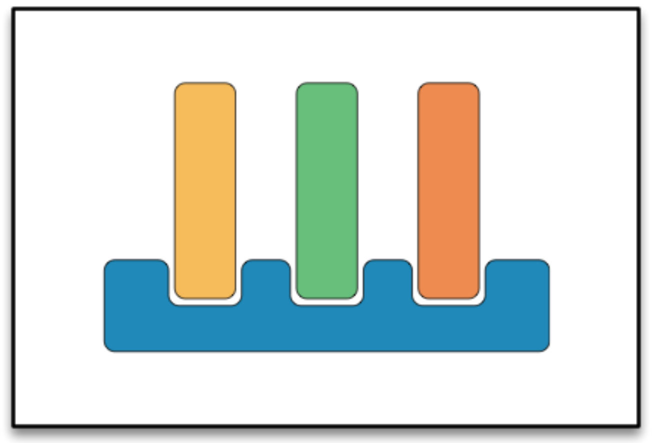


- Macro Architecture

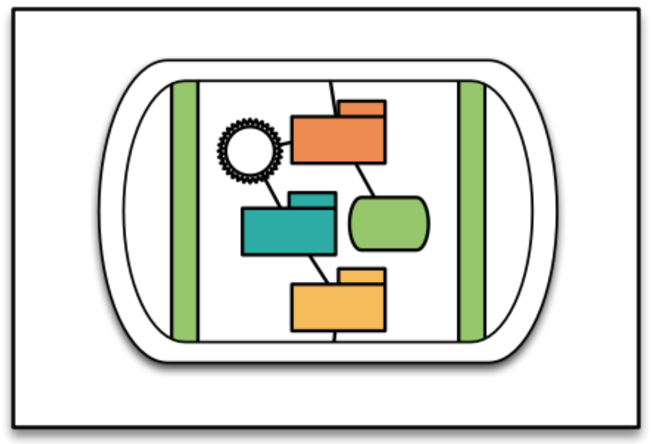
Architectural Decisions



- Domain Architecture

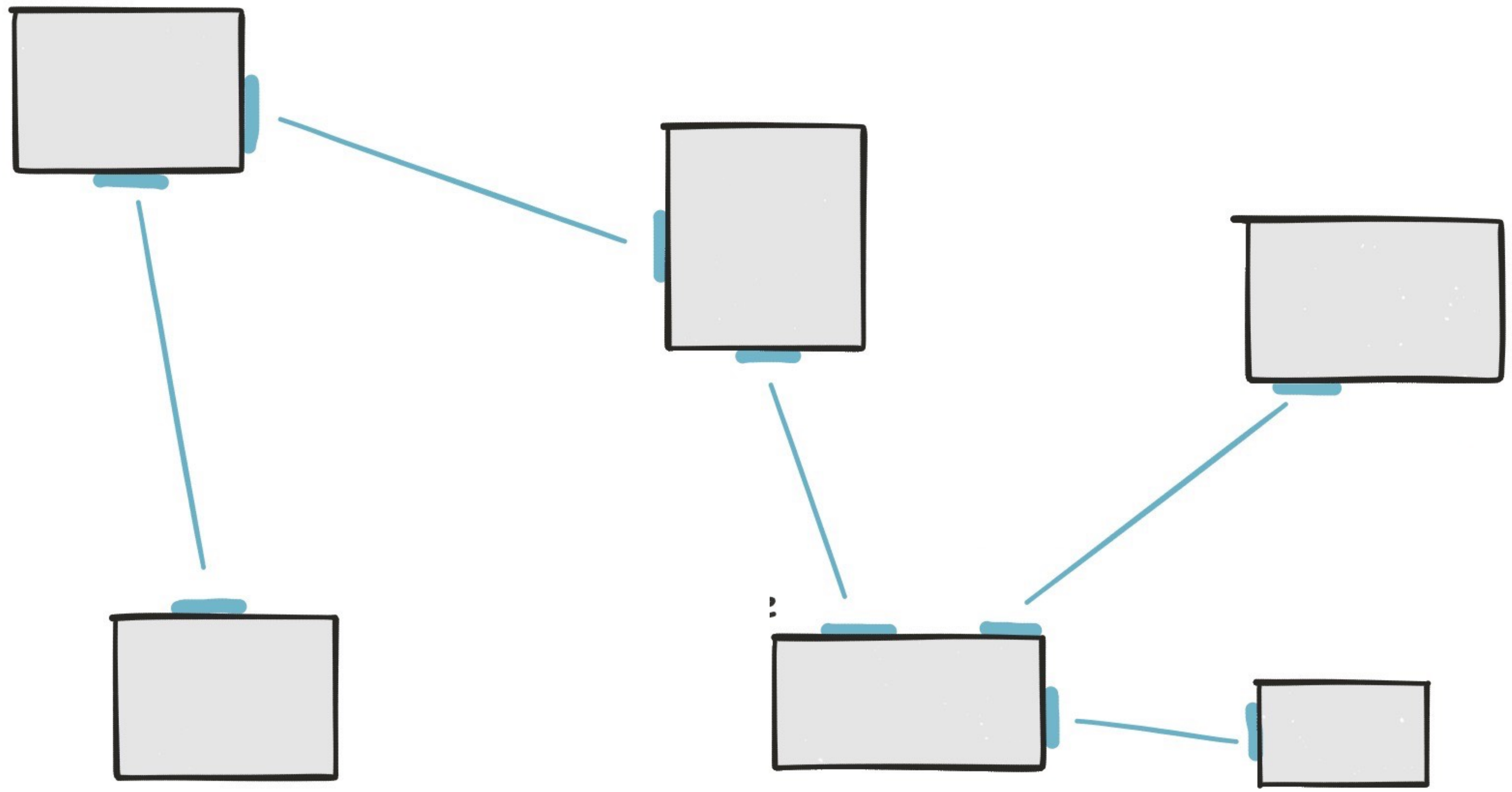


- Macro Architecture



- Micro Architecture

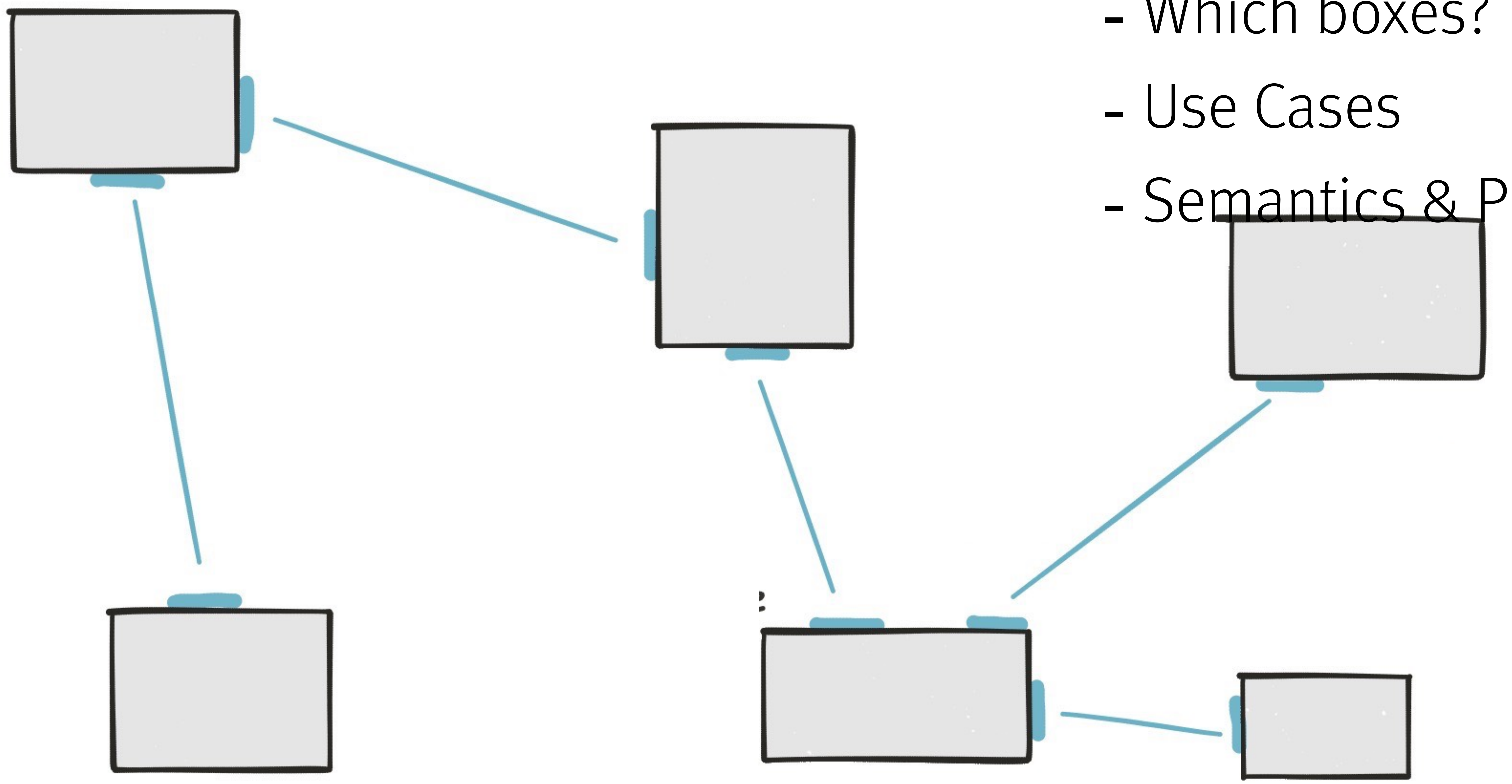
...so we show the different levels of decisions...



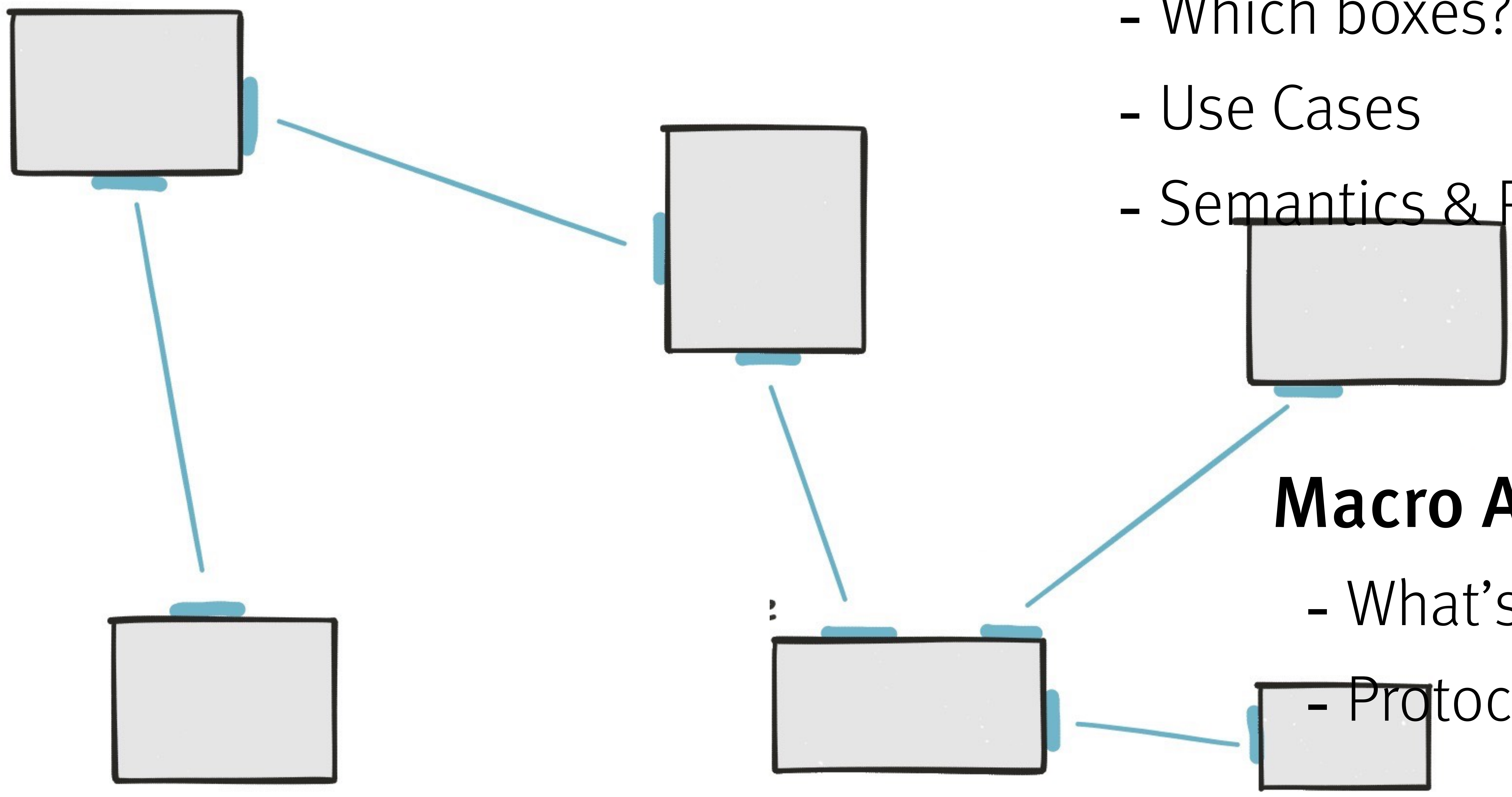
...so we show the different levels of decisions...

Domain Architecture

- Which boxes?
- Use Cases
- Semantics & Purpose



...so we show the different levels of decisions...



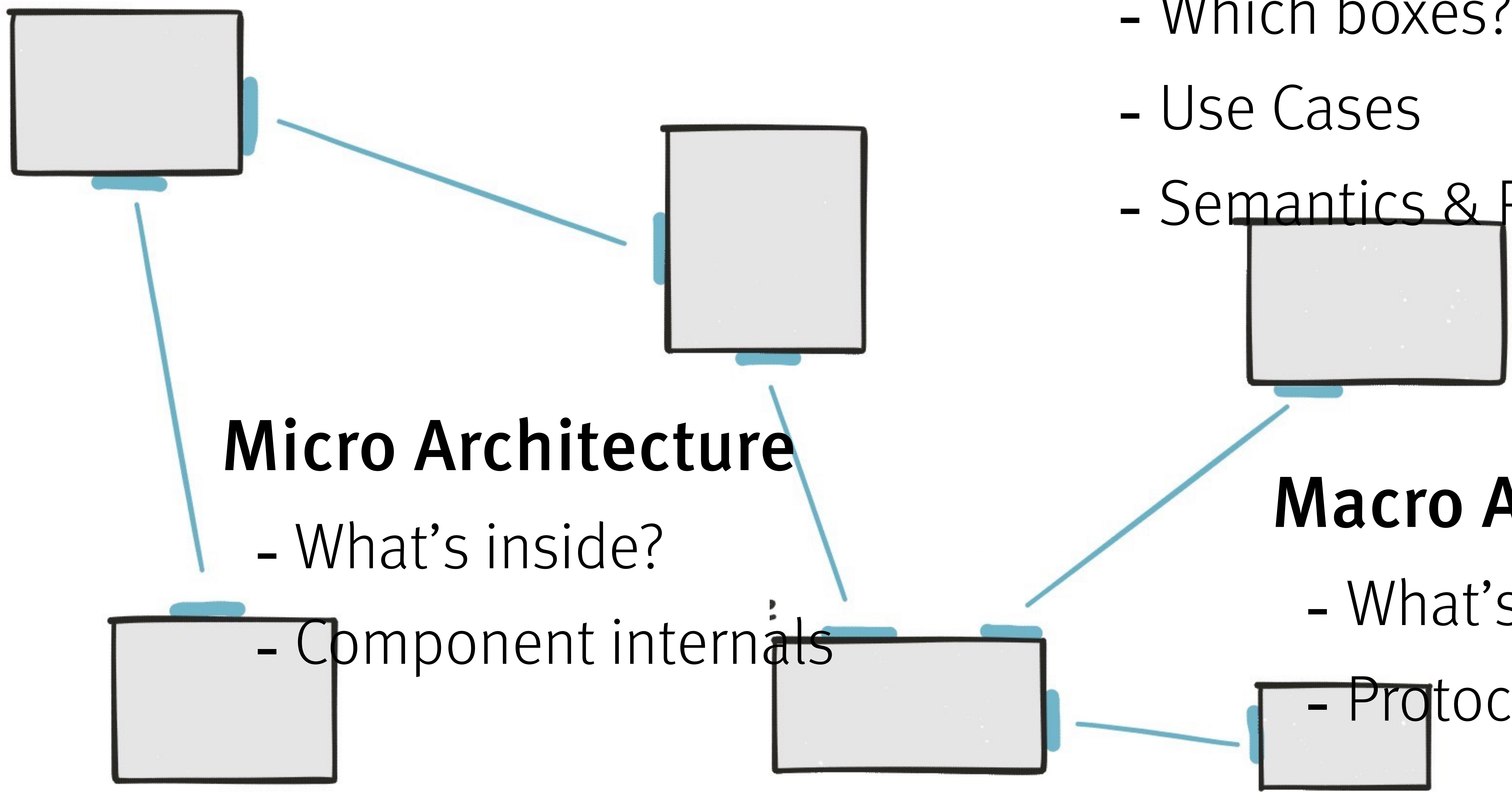
Domain Architecture

- Which boxes?
- Use Cases
- Semantics & Purpose

Macro Architecture

- What's in between?
- Protocols, Deployment

...so we show the different levels of decisions...



Domain Architecture

- Which boxes?
- Use Cases
- Semantics & Purpose

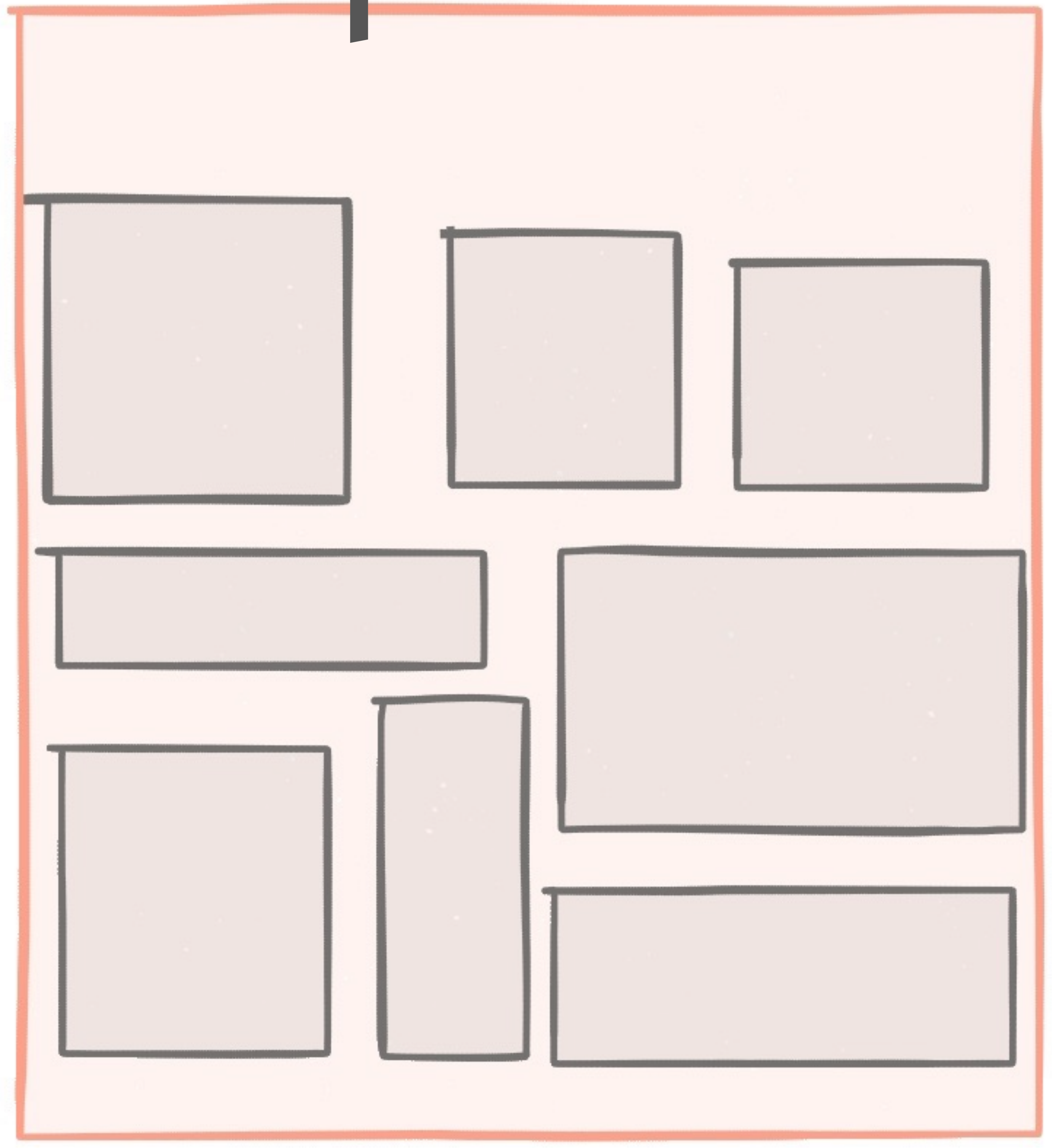
Micro Architecture

- What's inside?
- Component internals

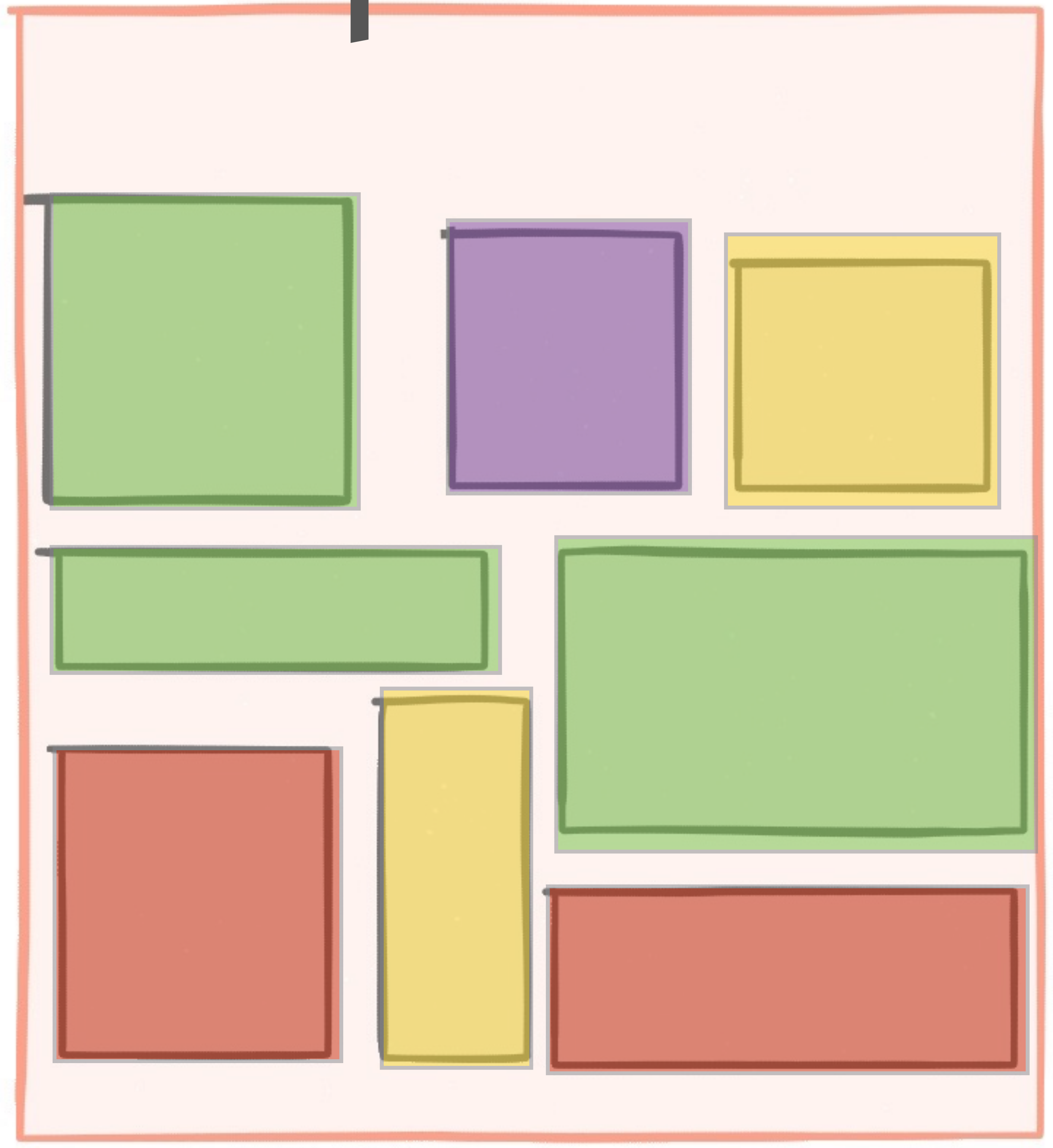
Macro Architecture

- What's in between?
- Protocols, Deployment

Steps for modularisation



Steps for modularisation



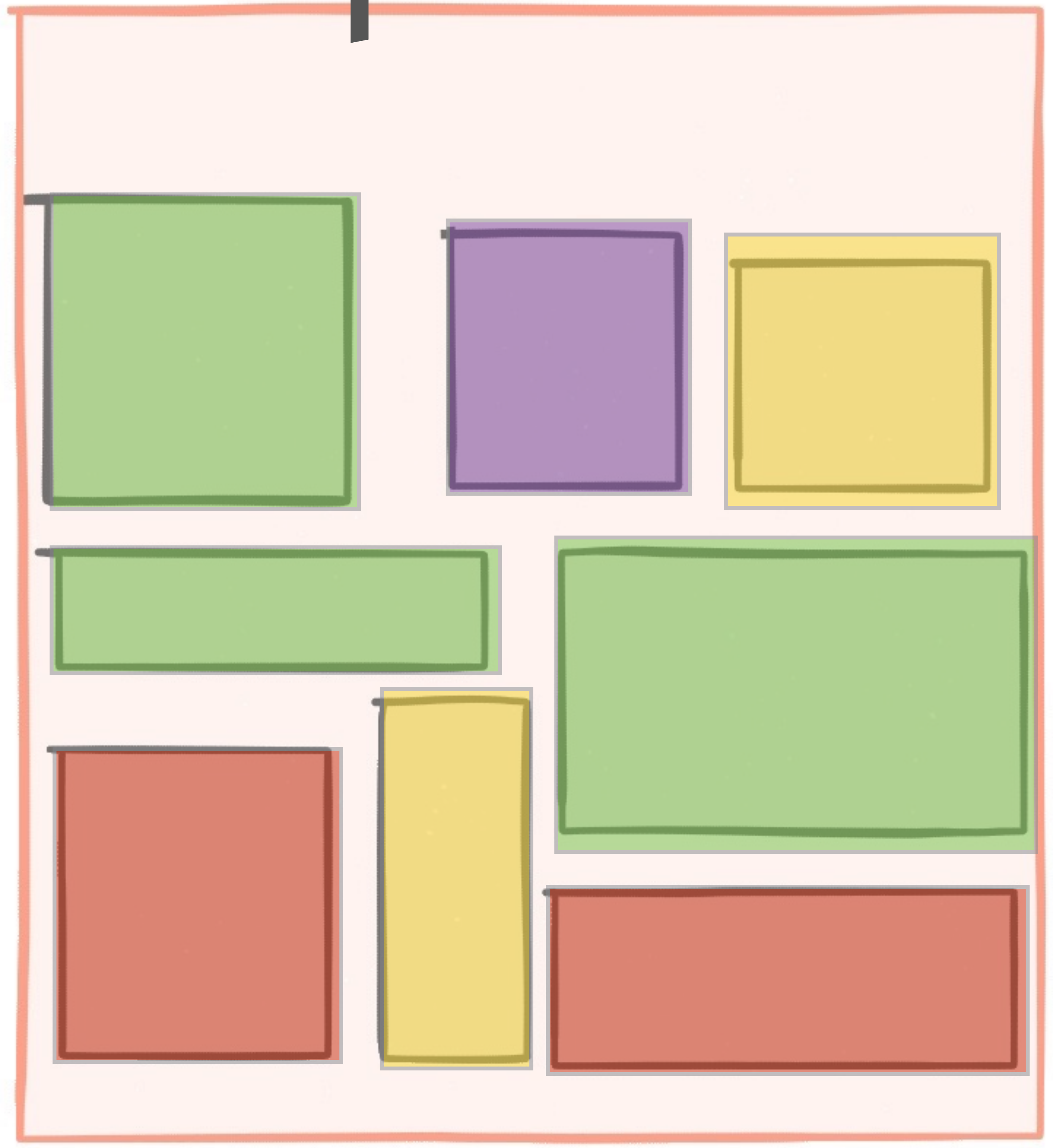
- identify domains

User Management

Product Management

Payment

Steps for modularisation



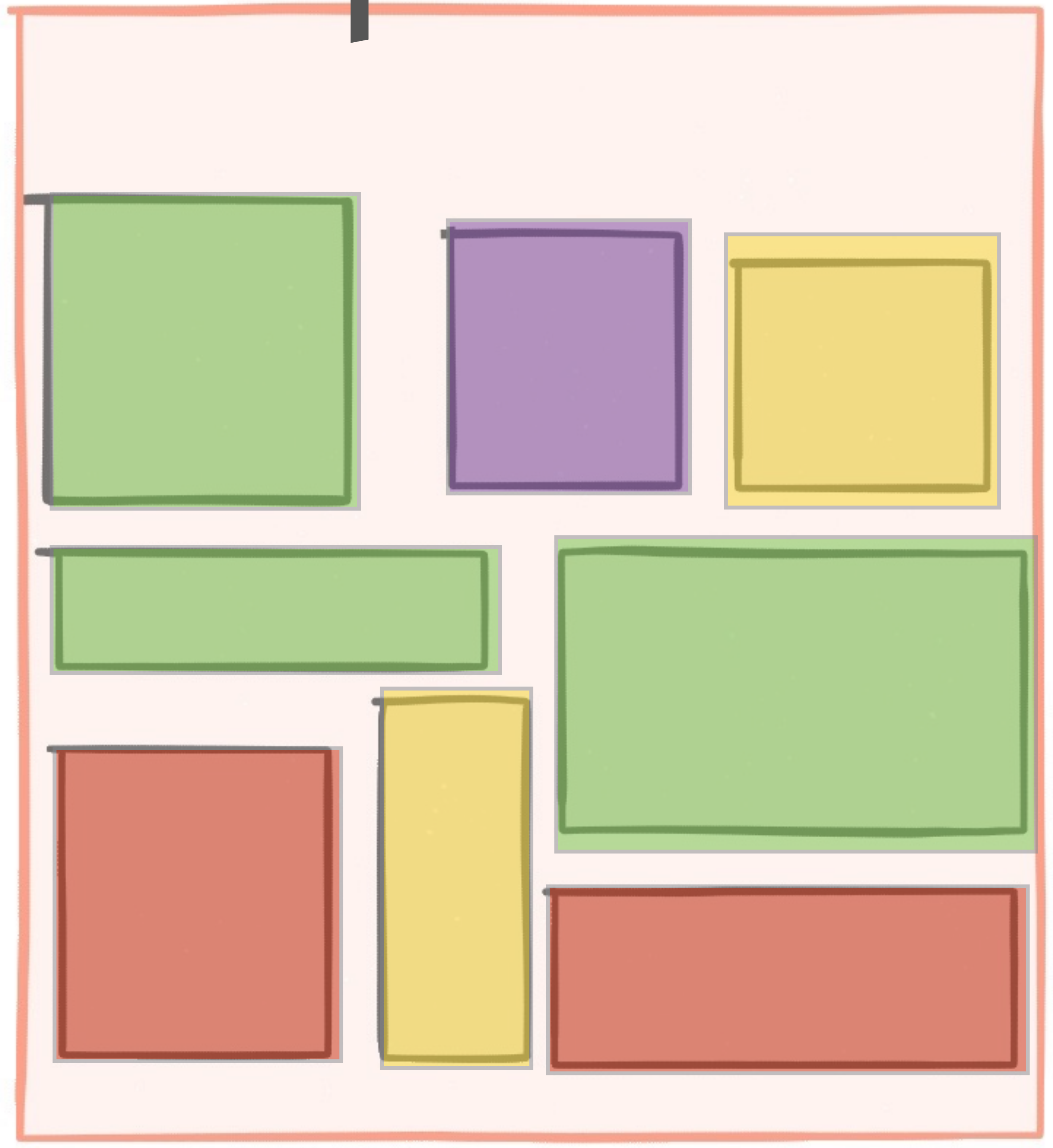
- identify domains
- group teams by domain

User Management

Product Management

Payment

Steps for modularisation



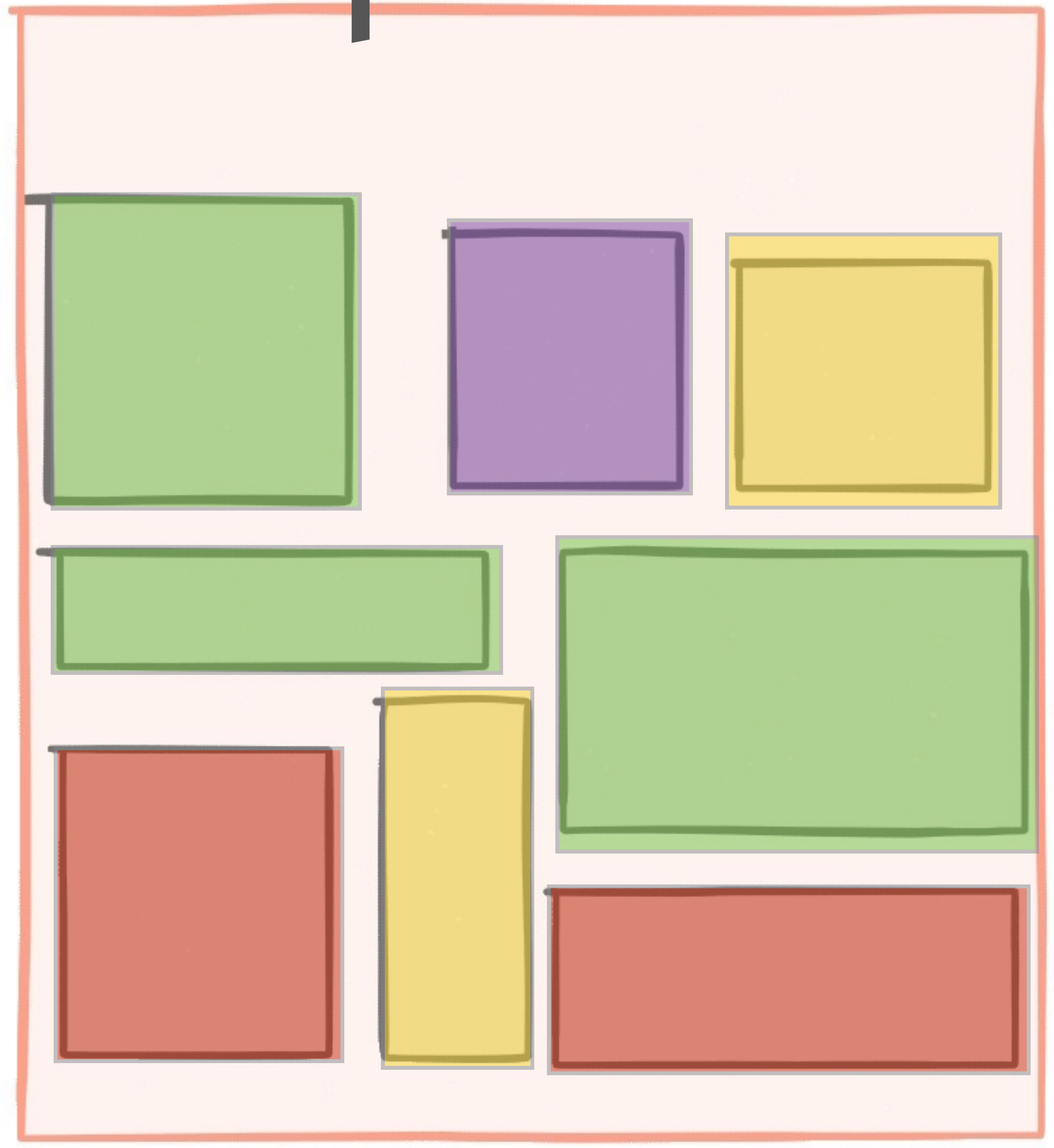
User Management

Product Management

Payment

- identify domains
- group teams by domain
- agree on macro architecture

Steps for modularisation



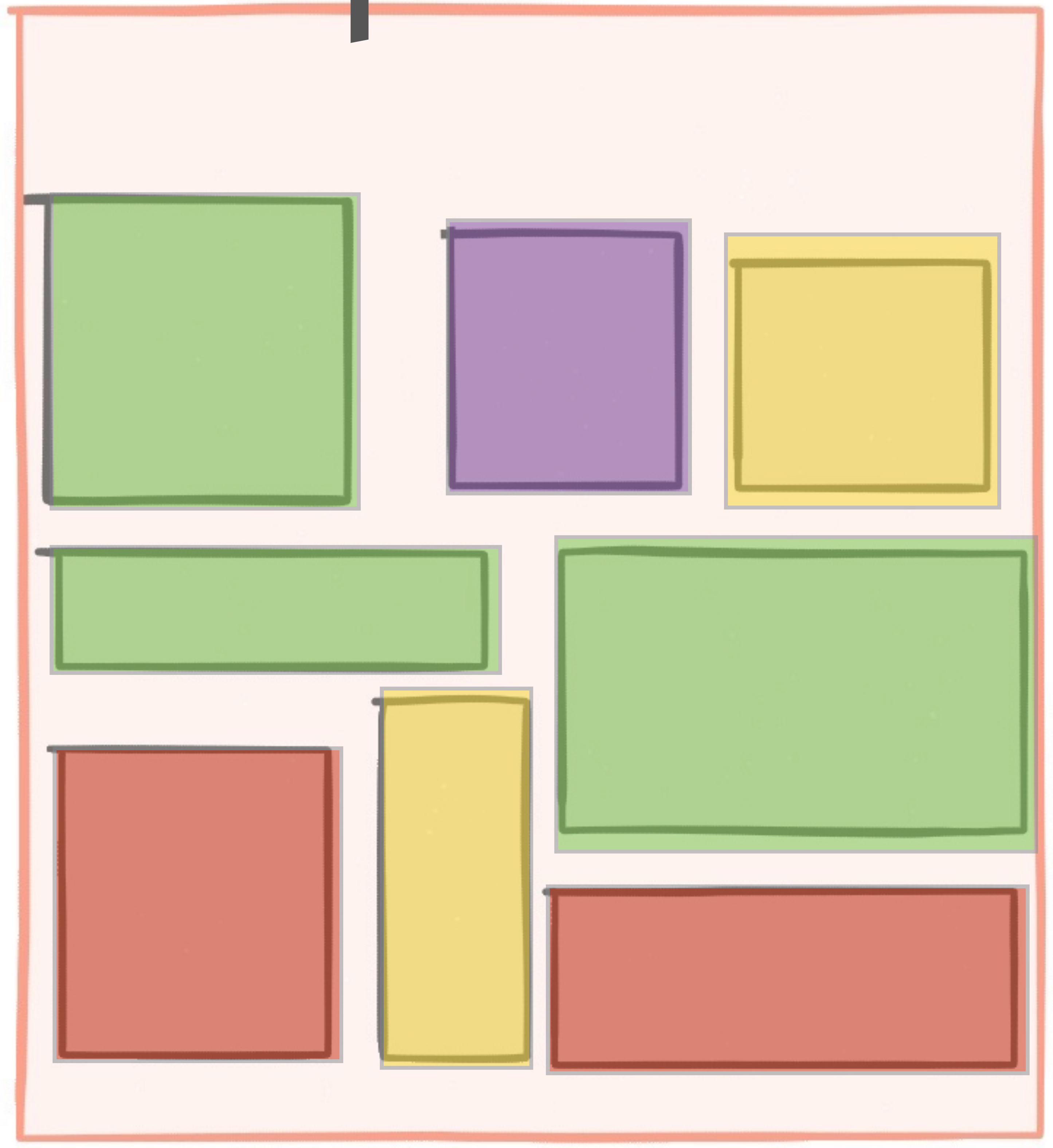
User Management

Product Management

Payment

- identify domains
- group teams by domain
- agree on macro architecture
- focus delivery pipeline on end-to-end features

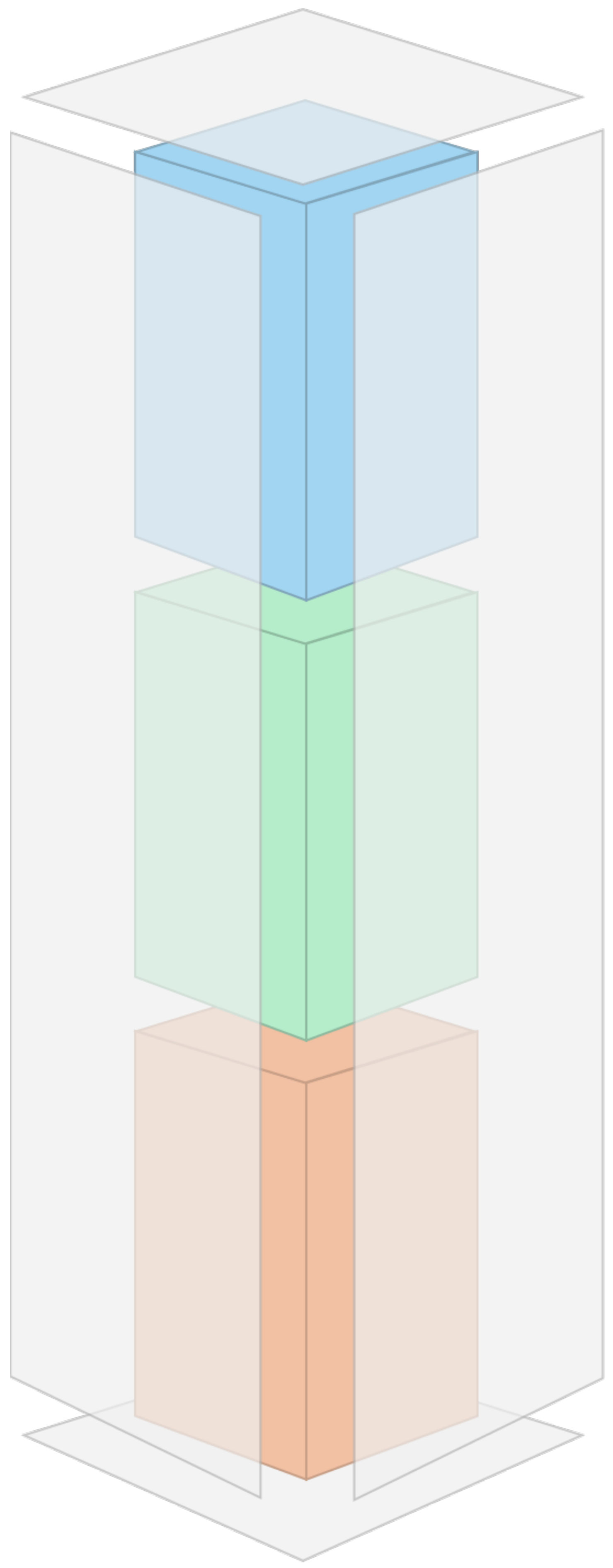
Steps for modularisation



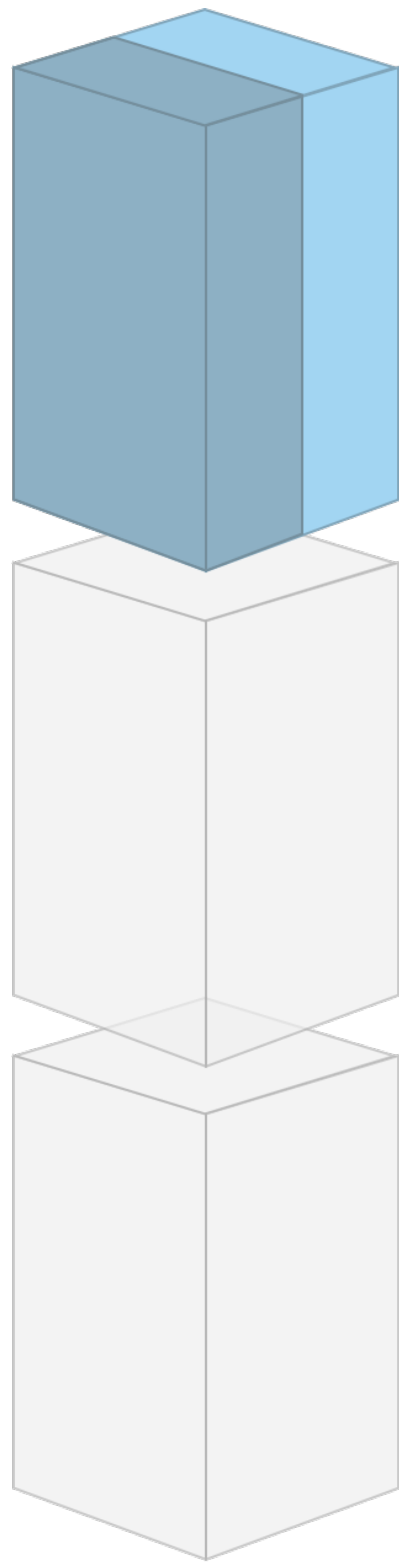
User Management
Product Management
Payment

- identify domains
- group teams by domain
- agree on macro architecture
- focus delivery pipeline on end-to-end features
- team decides migration approach case-by-case

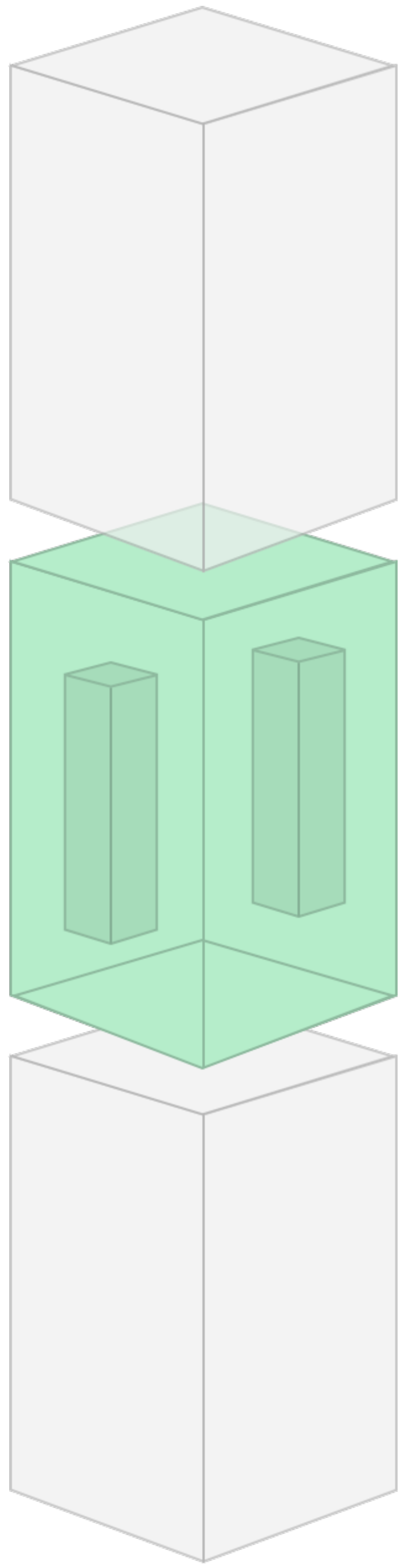
Self-Contained System (SCS)



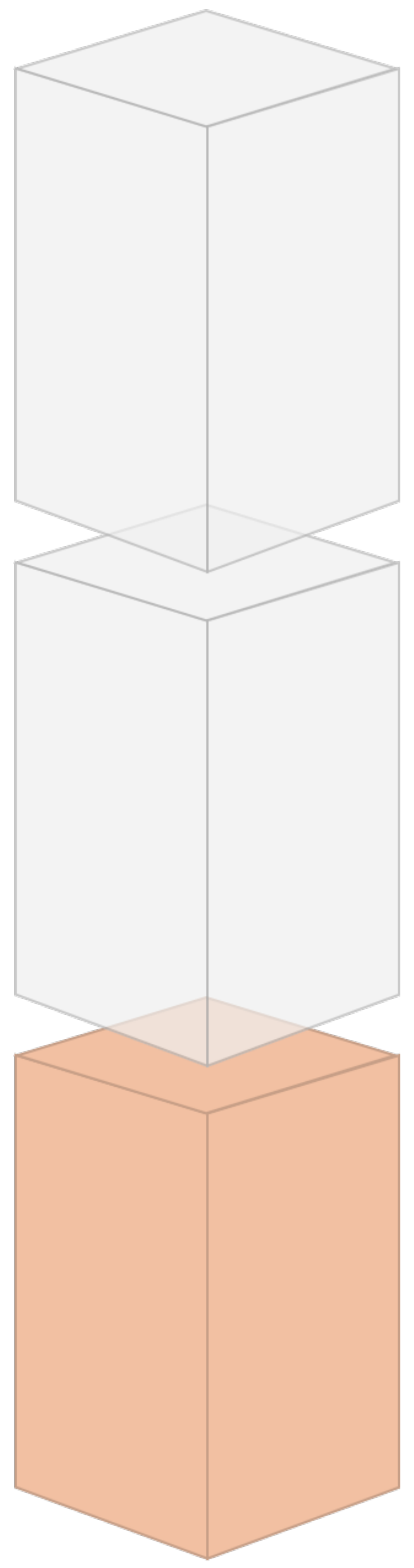
An SCS contains its own user interface, specific business logic and separate data storage



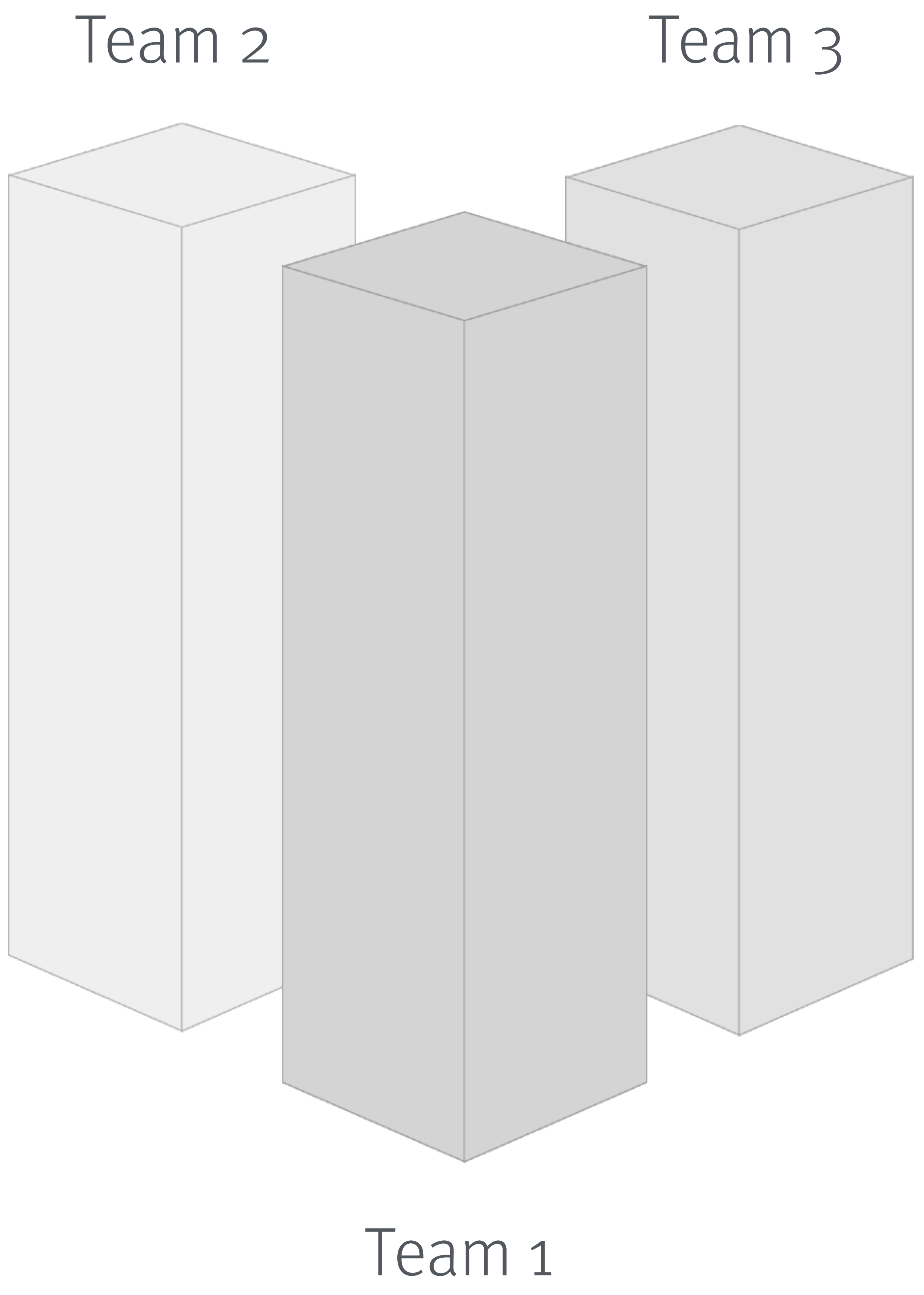
Besides a web interface a self-contained system can provide an **optional API**.



The business logic can consist of **microservices** to solve domain specific problems.

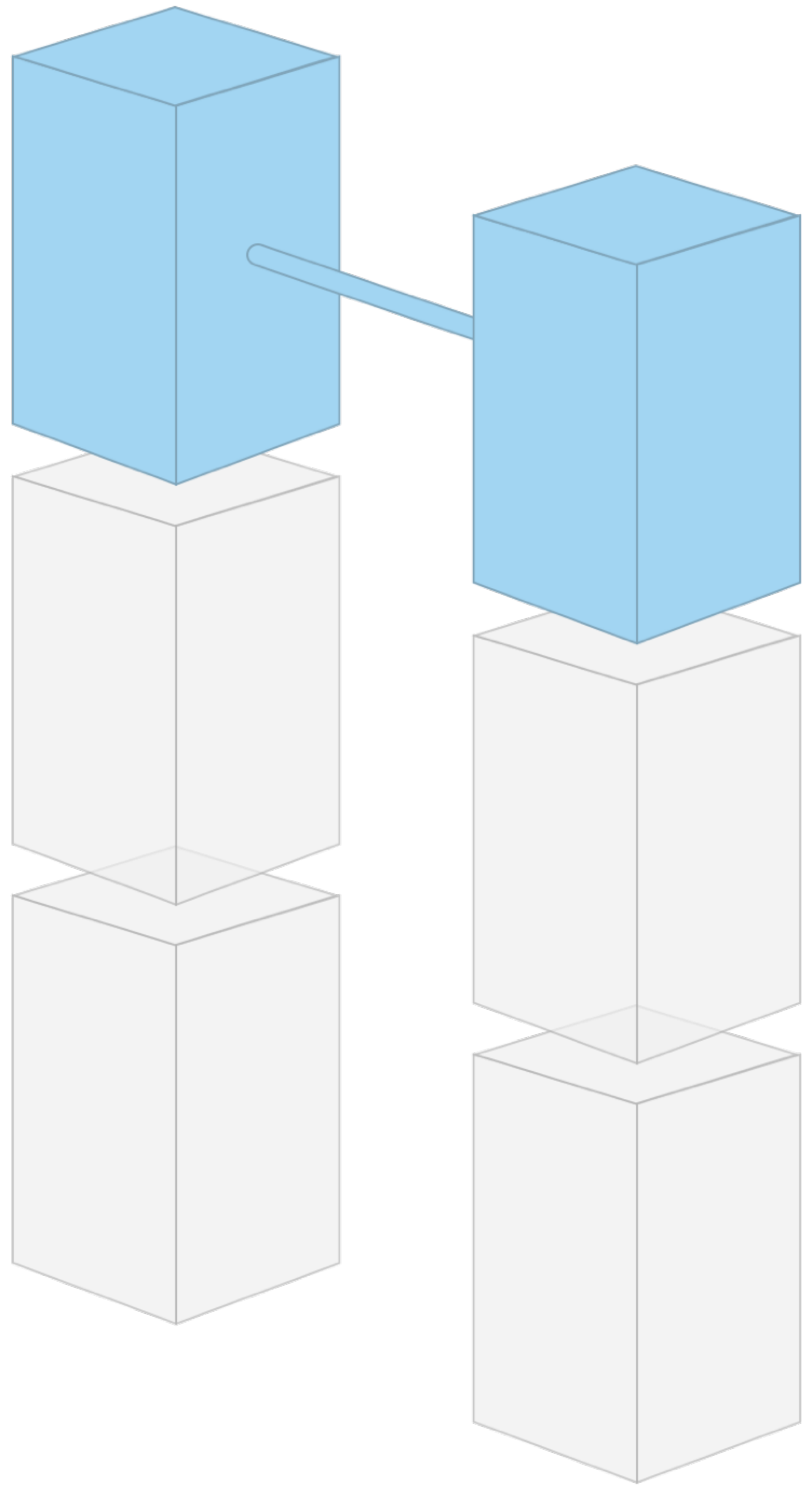


Every SCS brings its **own data storage** and with its redundant data depending on the context and domain.

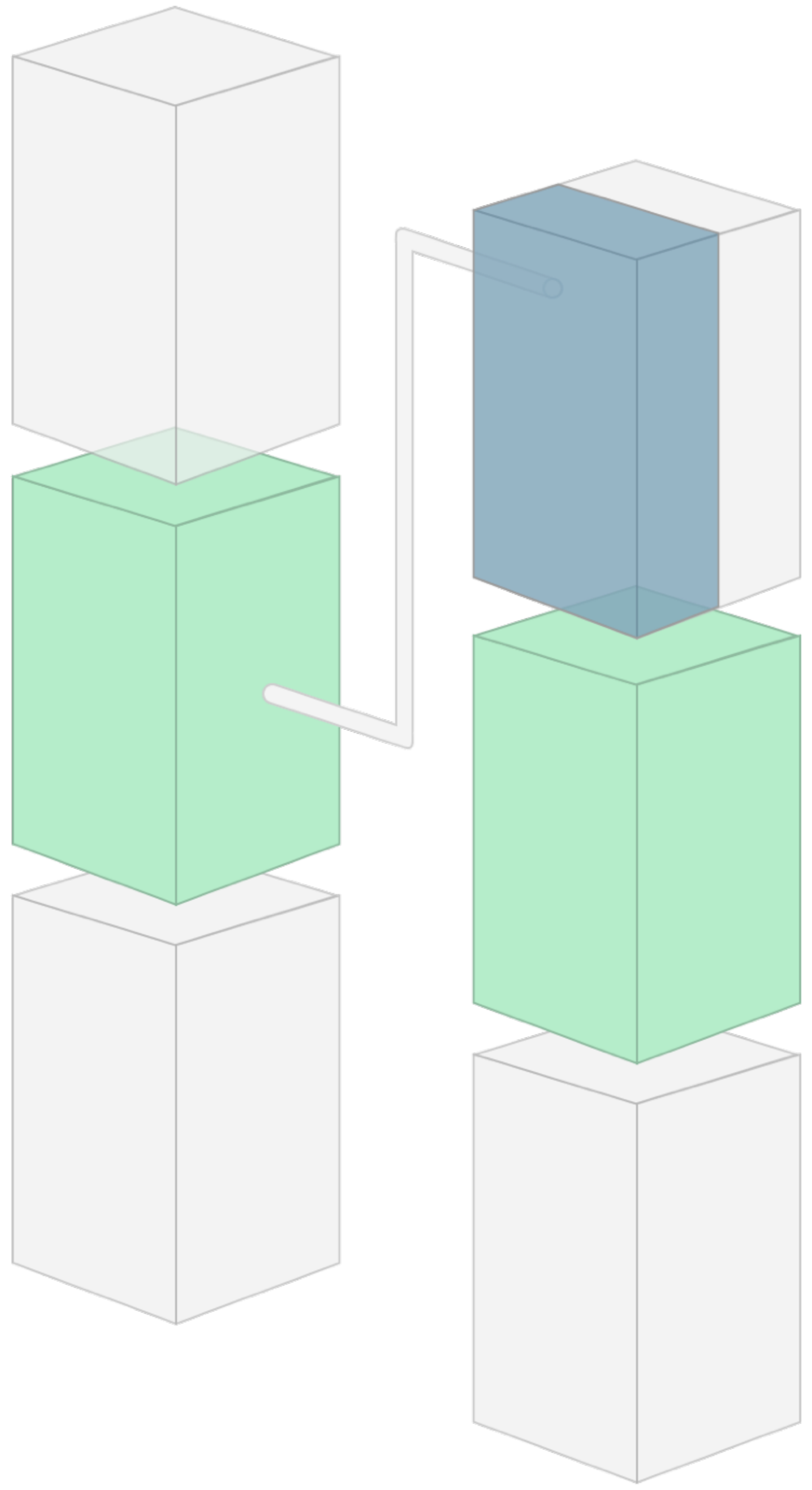


The manageable domain specific scope enables the development, operation and maintenance of an SCS by a **single team**.

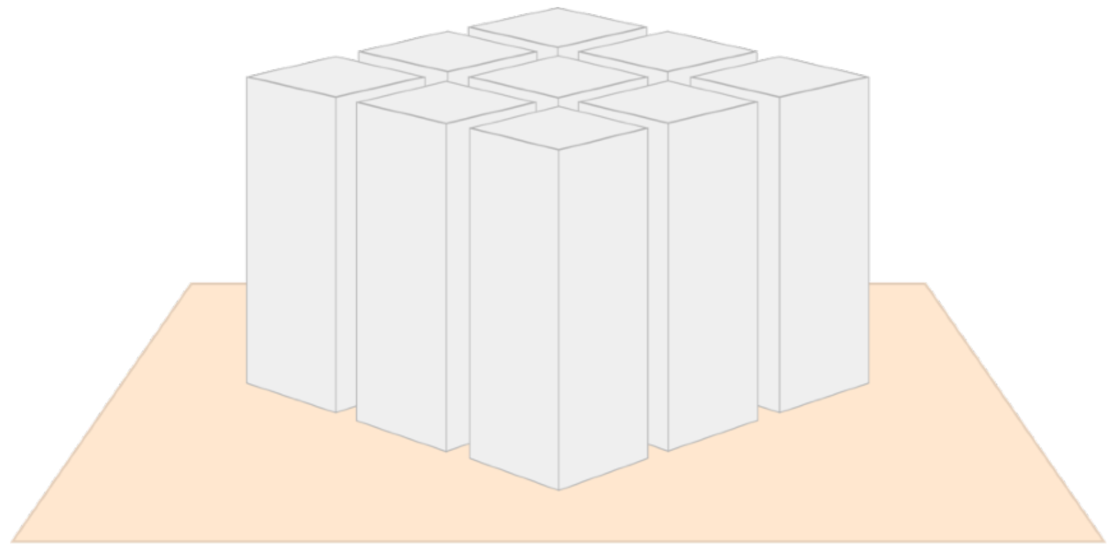
Integration?



Self-contained Systems
should be integrated over their
web interfaces to minimize
coupling to other systems.



Instead remote API calls should be handled **asynchronously** to reduce dependencies and prevent error cascades.



more information on **self-contained systems (SCS)** can be found at

http://scs-architecture.org/

conclusion

Summary

Summary

- aim42 provides structure for software modernization

Summary

- aim42 provides structure for software modernization
- SCSs are a reasonable approach to Microservices

Summary

- aim42 provides structure for software modernization
- SCSs are a reasonable approach to Microservices
- Not everyone who **wants** microservices is immediately **capable** to establish them

Summary

- aim42 provides structure for software modernization
- SCSs are a reasonable approach to Microservices
- Not everyone who **wants** microservices is immediately **capable** to establish them
- **Don’t overwhelm people**, change one thing at a time

Thank you!

Questions?

Comments?

Alexander Heusingfeld, @goldstift
alexander.heusingfeld@innoq.com

Michael Vitz, @michaelvitz
michael.vitz@innoq.com

https://www.innoq.com/en/talks/