TRAILHEAD
TECHNOLOGY PARTNERS

Avoidifying Over-Complexification

Rooting Out Over-Engineering in Your Projects

Jonathan “J.” Tower


TRAILHEAD
TECHNOLOGY PARTNERS

TRAILHEAD
TECHNOLOGY PARTNERS

credit: scienceworld.scholastic.com

Identify **10 Types** of Over-Engineering

10 Rules to Help Avoid It

TRAILHEAD
TECHNOLOGY PARTNERS

Jonathan "J." Tower

Principal Consultant & Partner

TRAILHEAD
TECHNOLOGY PARTNERS

Microsoft MVP in .NET

jtower@trailheadtechnology.com

trailheadtechnology.com/blog

jtowermi

jtower

FREE

CONSULTATION

bit.ly/th-offer

github.com/trailheadtechnology/over-engineering

10 Common Types of Over-Engineering

TRAILHEAD
TECHNOLOGY PARTNERS

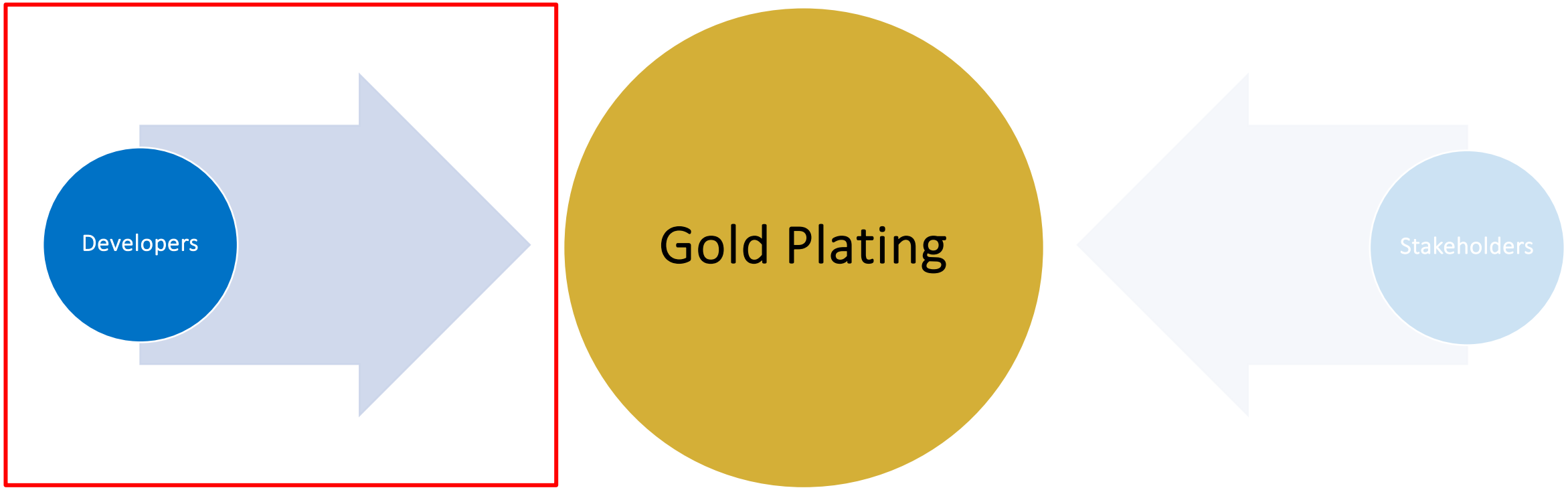
Gold-Plating

Over-Engineering Type 1

Sources of Gold-Plating

TRAILHEAD
TECHNOLOGY PARTNERS

Sources of Gold-Plating



TRAILHEAD
TECHNOLOGY PARTNERS

Reasons Developers Gold-Plate

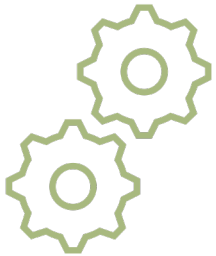
Perfectionism

Burned Previously

Personal Interest

TRAILHEAD
TECHNOLOGY PARTNERS

Issues With Gold-Plating



Complexity

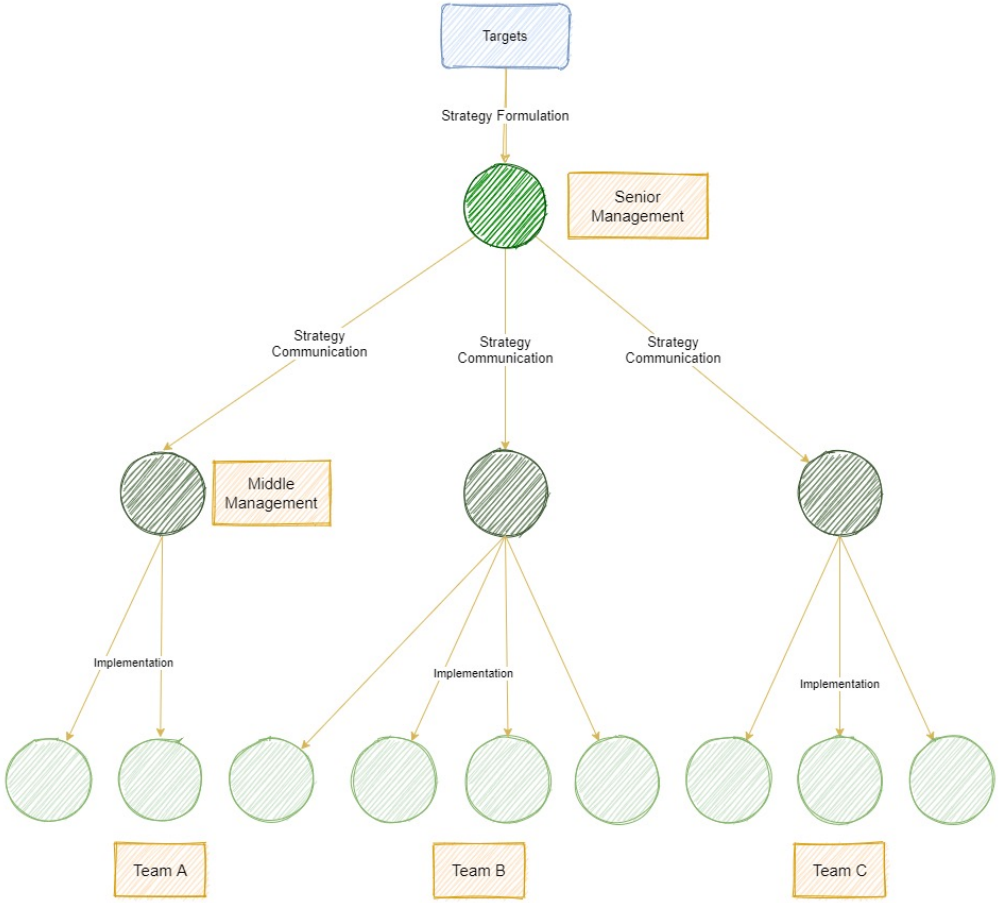
Time & Cost

Loss of Focus

TRAILHEAD
TECHNOLOGY PARTNERS

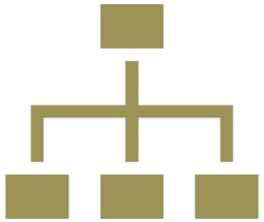
OO Gymnastics

Over-Engineering Type 2



Types of OO Gymnastics

Unnecessary Generics



Complex Inheritance Hierarchies

Properties for everything

Reasons Developers Do OO Gymnastics

Showing Off

Future Proofing

Personal Interest

Over-Abstraction

Over-Engineering Type 3

Over-Abstraction

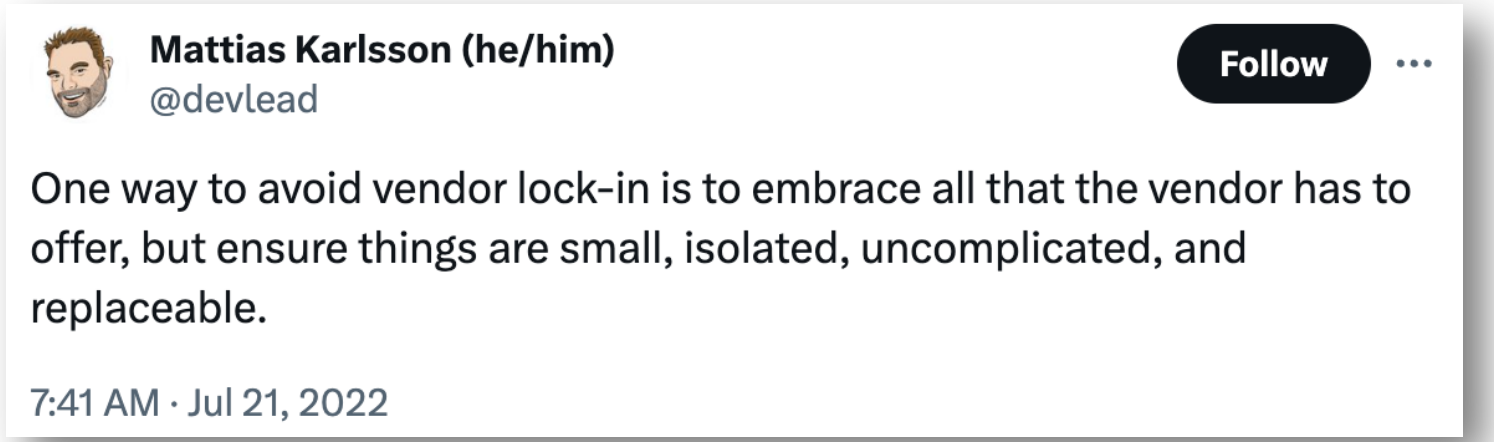
TRAILHEAD
TECHNOLOGY PARTNERS

Reasons Developers Over-Abstract

Avoid Vendor Lock-In

Why You Don't Need Over-Abstraction

- Embrace all 3rd-party library has to offer
- Ensure to select libraries that:
  1. Are small
  2. Isolated
  3. Uncomplicated
  4. Replaceable

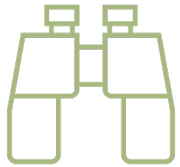


TRAILHEAD
TECHNOLOGY PARTNERS

Over-Built Scalability

Over-Engineering Type 4

Reasons Developers Over-Build for Scalability



Anticipation of Future Needs

Fear of Rework

Influence of Trends

Pressure from Stakeholders

TRAILHEAD
TECHNOLOGY PARTNERS

Rule of Thumb

"Build your software for, at most, 1-2 orders of magnitude more than you currently need."

- J.

Issues with Over-Building for Scalability

Increased Time & Costs

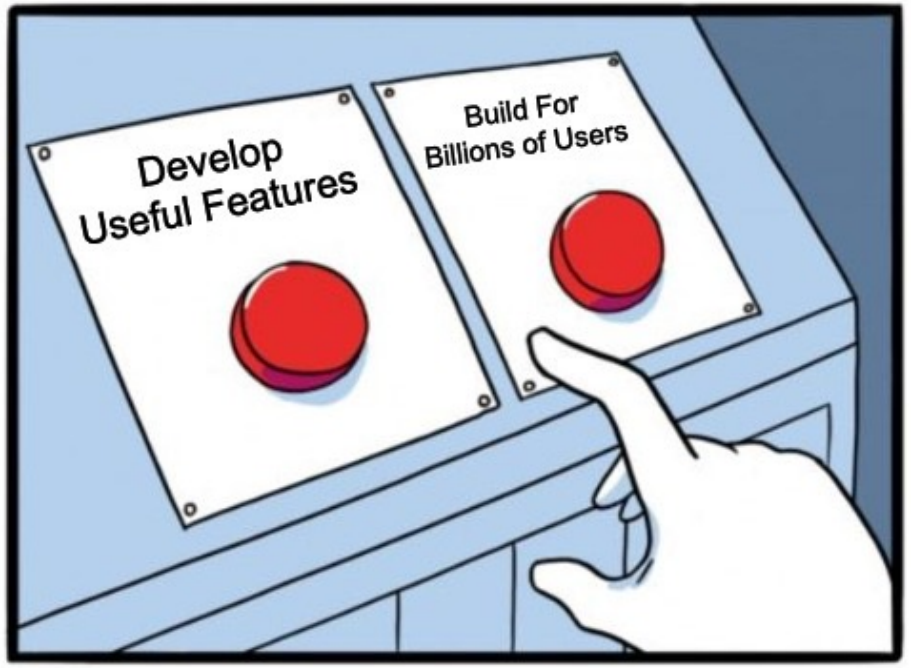
Not Agile

Solving the Wrong Problems

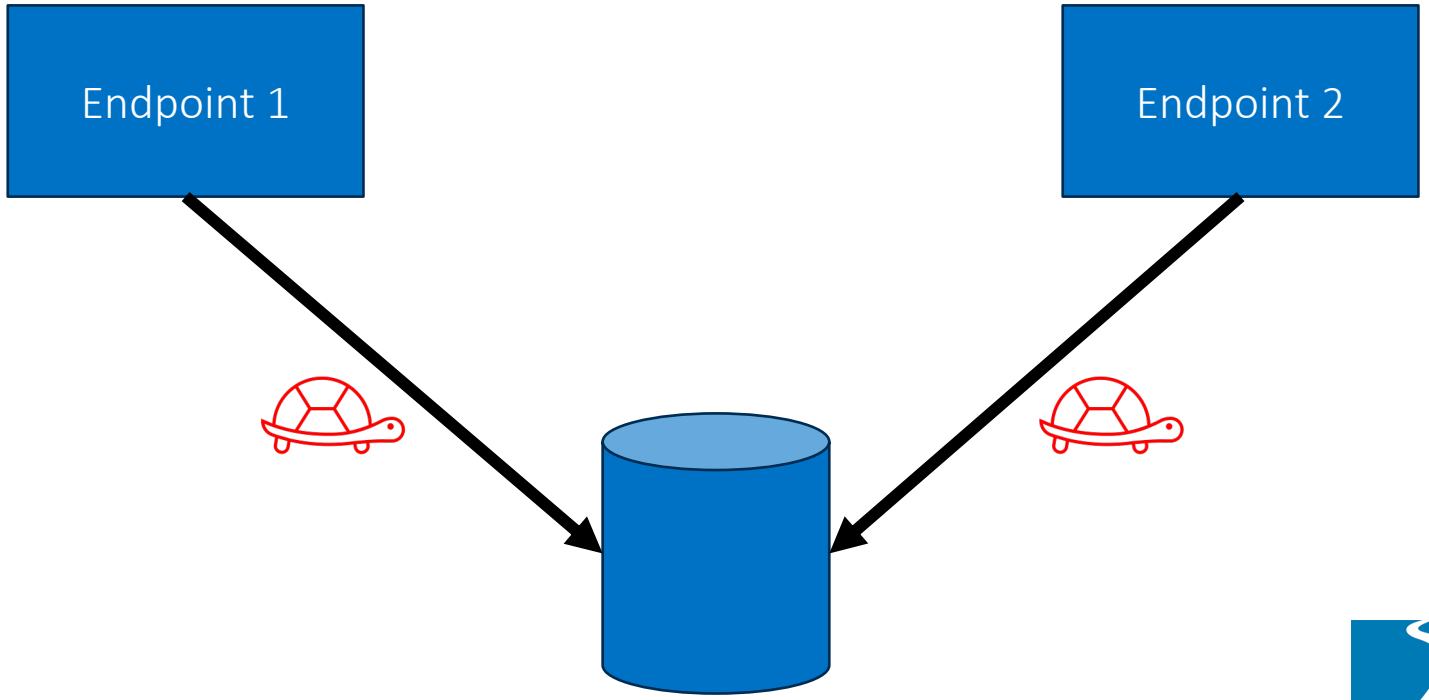
TRAILHEAD
TECHNOLOGY PARTNERS

Premature Optimization

Over-Engineering Type 5

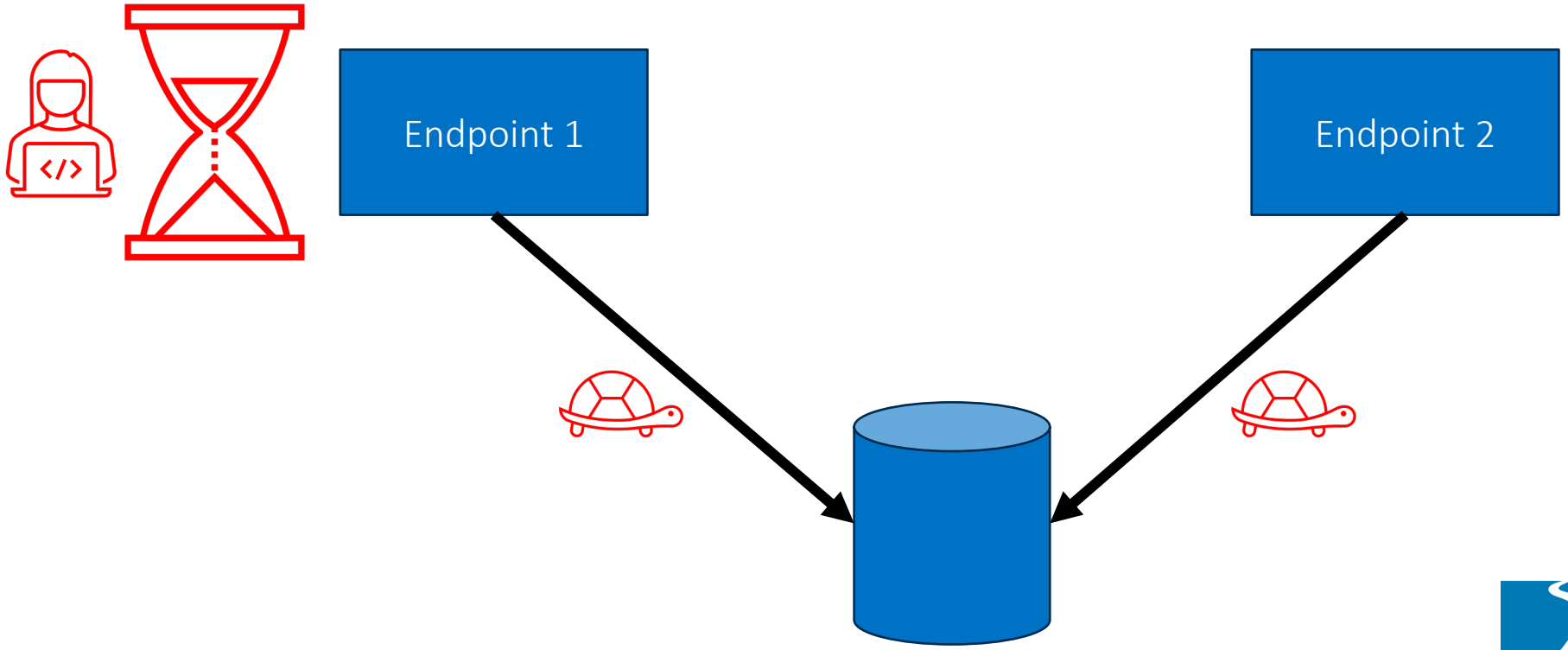


Premature Optimization



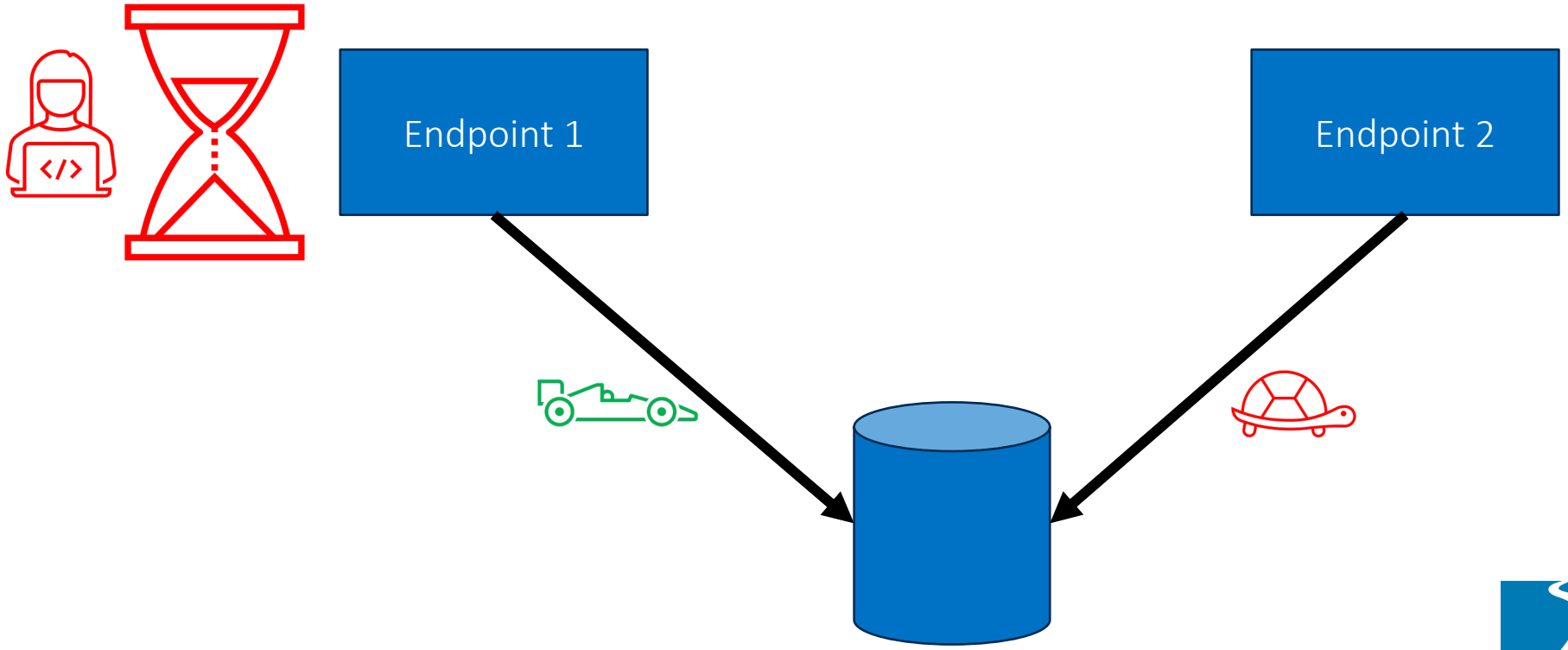
TRAILHEAD
TECHNOLOGY PARTNERS

Premature Optimization



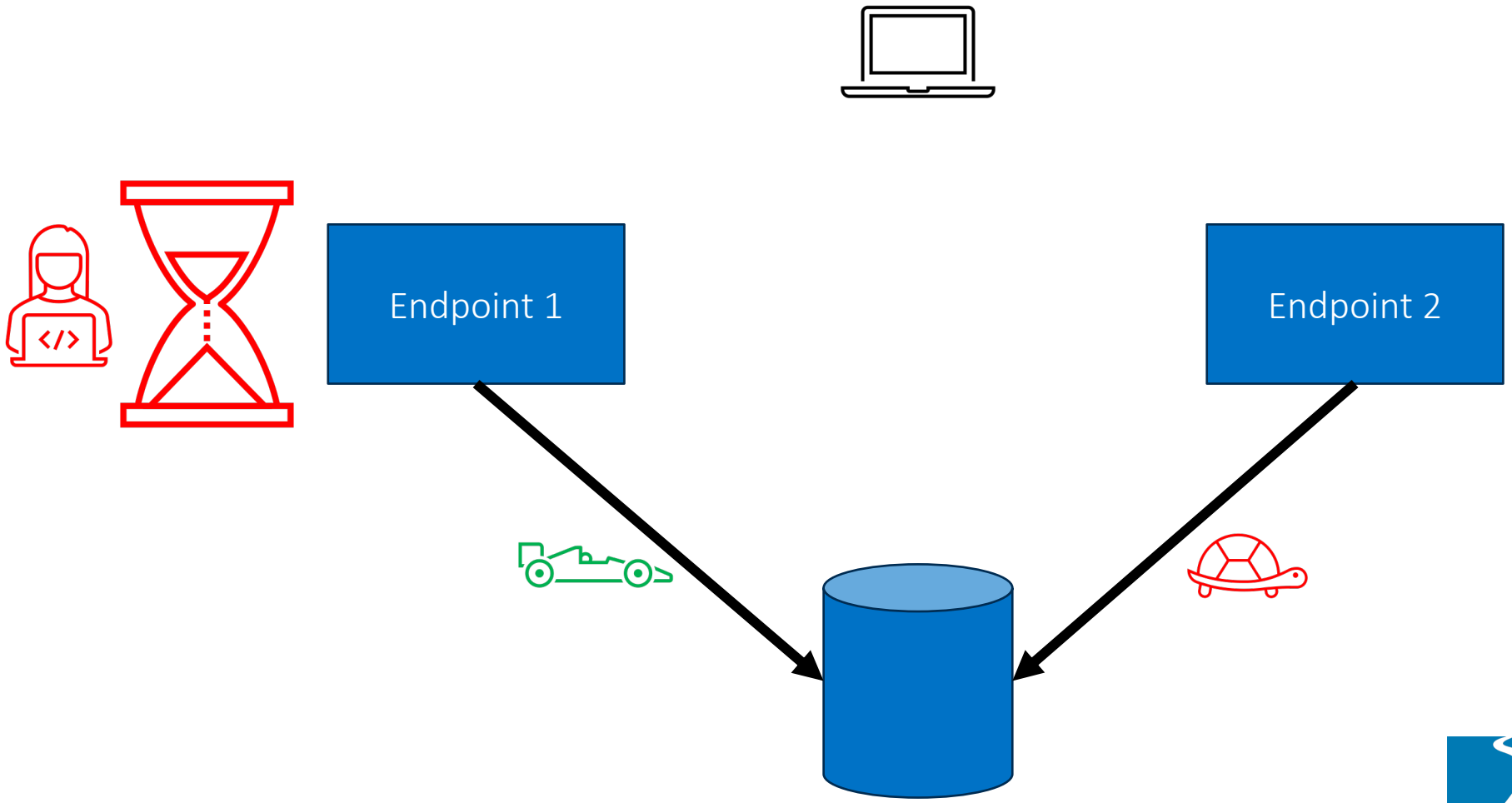
TRAILHEAD
TECHNOLOGY PARTNERS

Premature Optimization



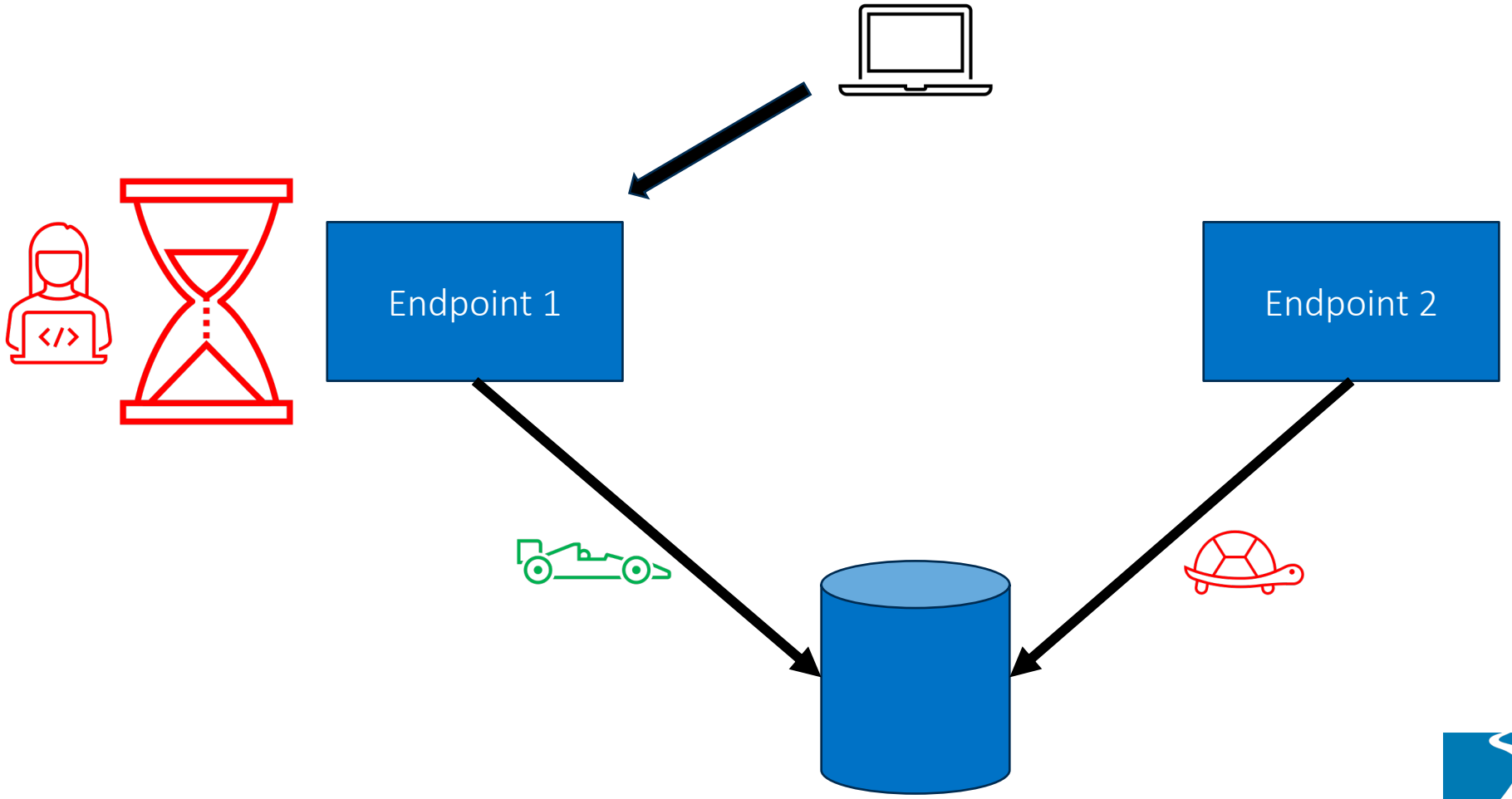
TRAILHEAD
TECHNOLOGY PARTNERS

Premature Optimization



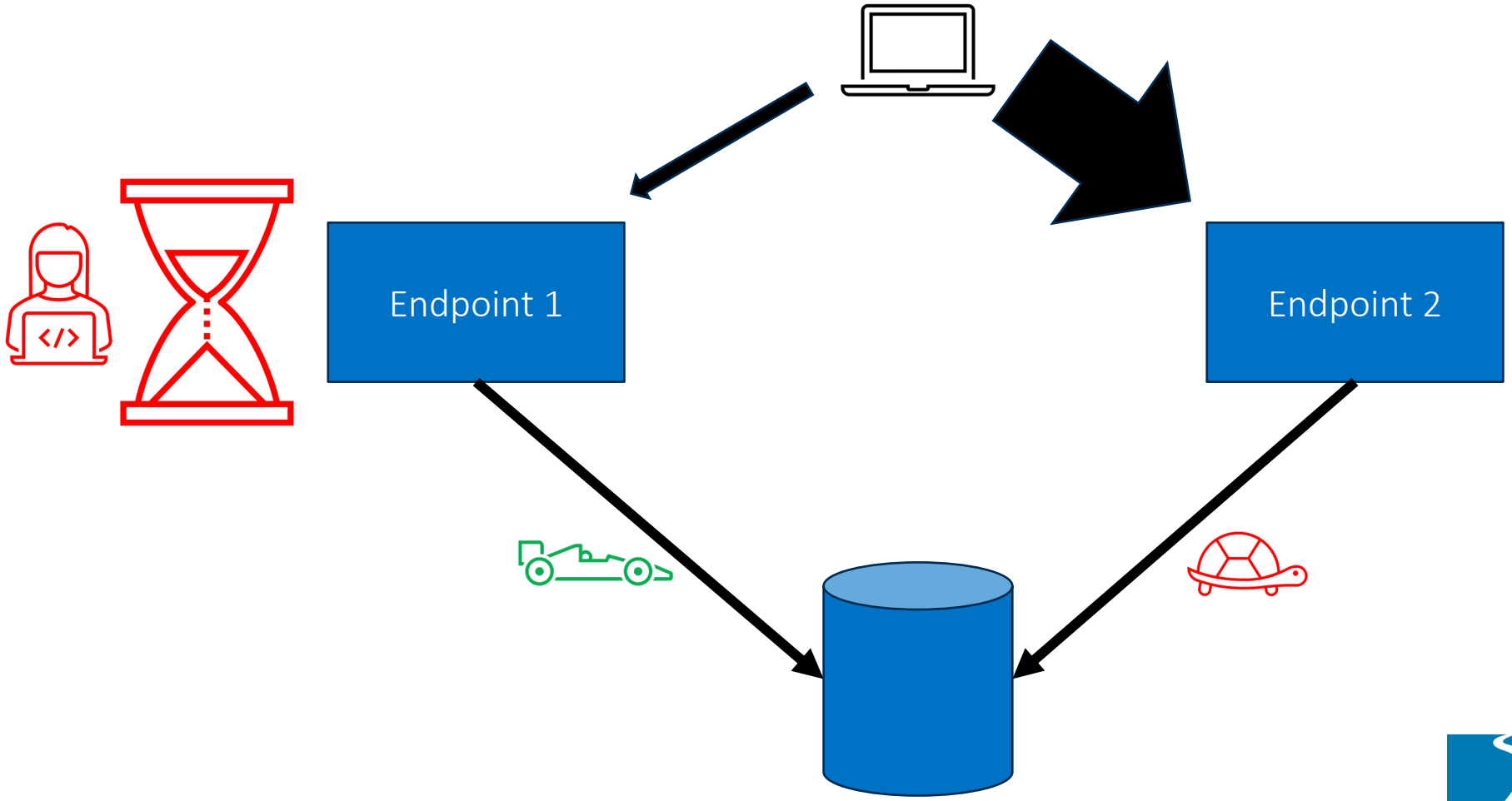
TRAILHEAD
TECHNOLOGY PARTNERS

Premature Optimization



TRAILHEAD
TECHNOLOGY PARTNERS

Premature Optimization



TRAILHEAD
TECHNOLOGY PARTNERS

Reasons Developers Optimize Prematurely

Perfectionism

Assumptions About Performance Needs

Fear of Future Performance Issues

Influence of High-Performance Domains

Preemptive Solution to Rare Issues

TRAILHEAD
TECHNOLOGY PARTNERS

Issues with Premature Optimization

Increased Costs

Extended
Development Time

Solving the Wrong
Problems

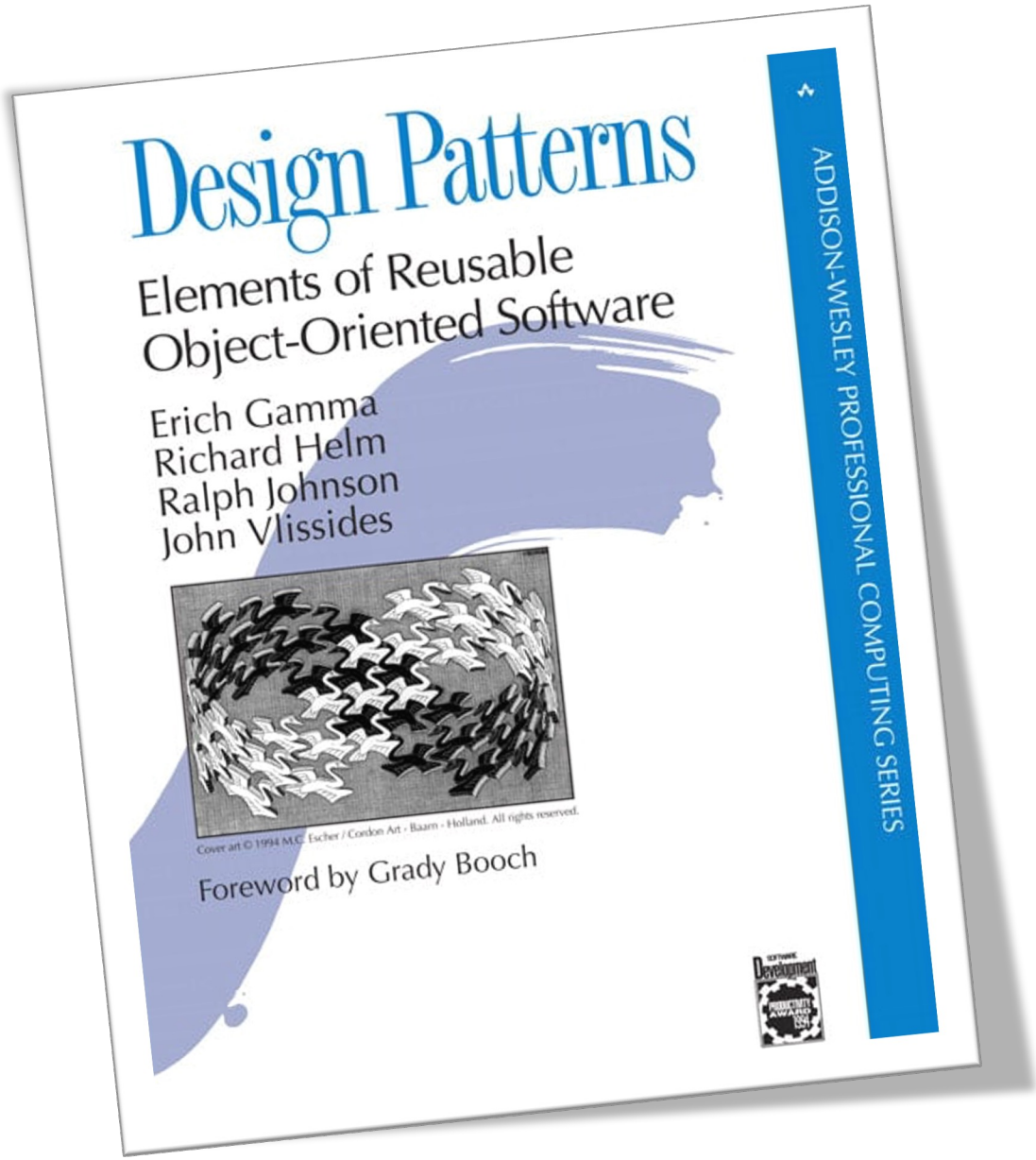
TRAILHEAD
TECHNOLOGY PARTNERS

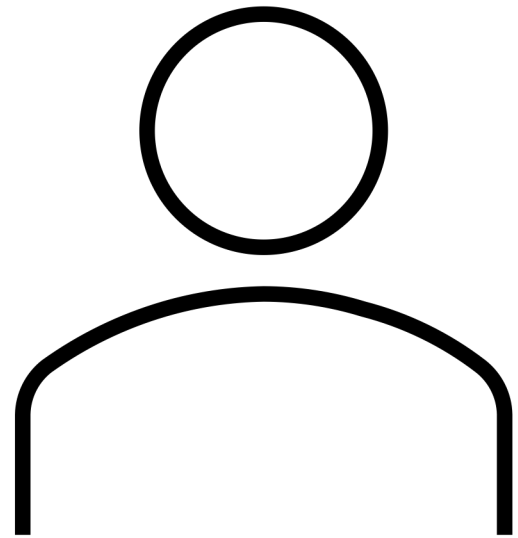
Overuse of Design Patterns

Over-Engineering Type 6

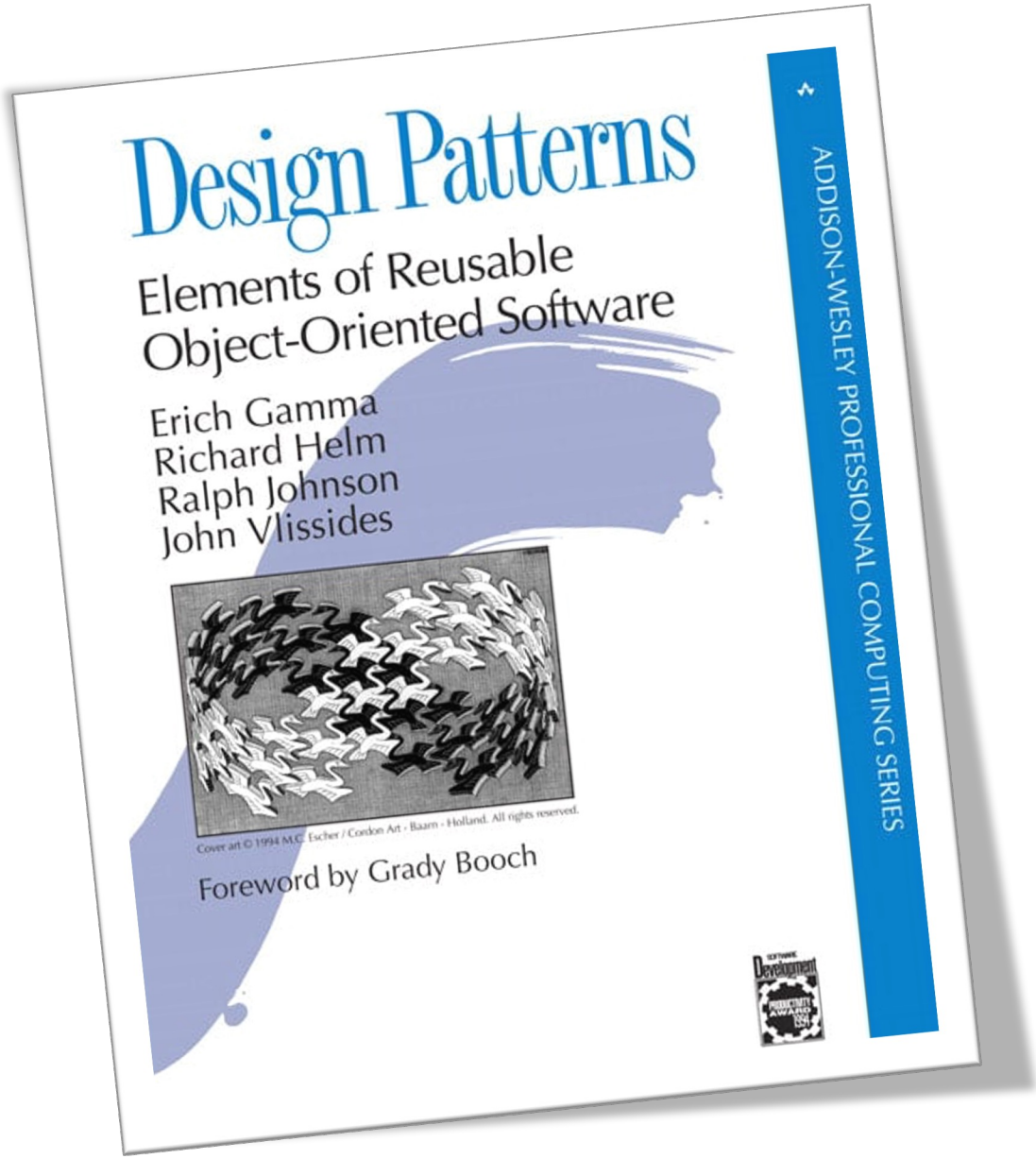
Design Patterns

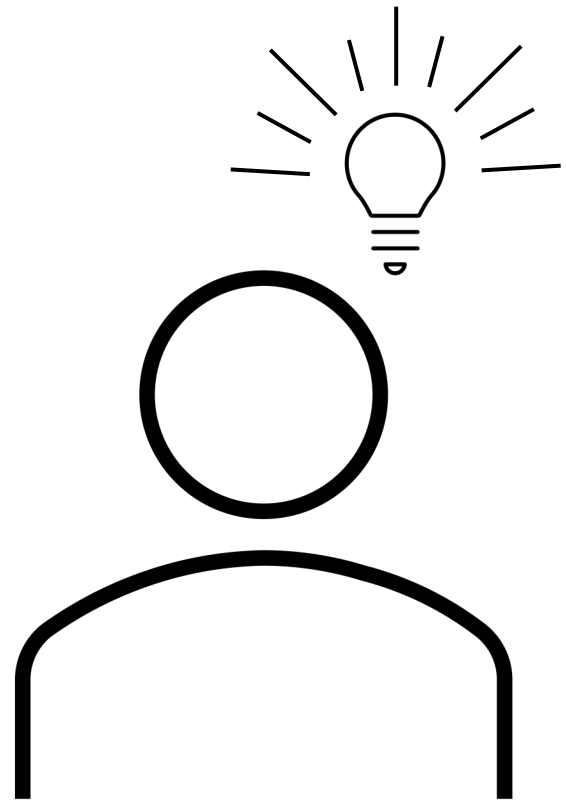
Abstract Factory
Builder
Factory Method
Object Pool
Prototype
Singleton
Adapter
Bridge
Composite
Decorator
Facade
Flyweight
Private Class Data
Proxy
Chain of responsibility
Command
Interpreter
Iterator
Mediator
Memento
Null Object
Observer
State
Strategy
Template method
Visitor





TRAILHEAD
TECHNOLOGY PARTNERS





TRAILHEAD
TECHNOLOGY PARTNERS

“If the only tool **you have is a hammer**, it is tempting to treat **everything** as if it were a nail.”

- Abraham Maslow

Reasons Developers Overuse Design Patterns

Educational Enthusiasm

Misunderstanding of Design Patterns

Desire to Follow Best Practices

Peer Pressure and Industry Trends

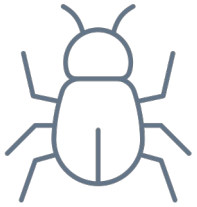
Fear of Code Smells

TRAILHEAD
TECHNOLOGY PARTNERS

Issues with Overuse of Design Patterns

Increased Costs

Extended Development Time



Higher Risk Of Bugs

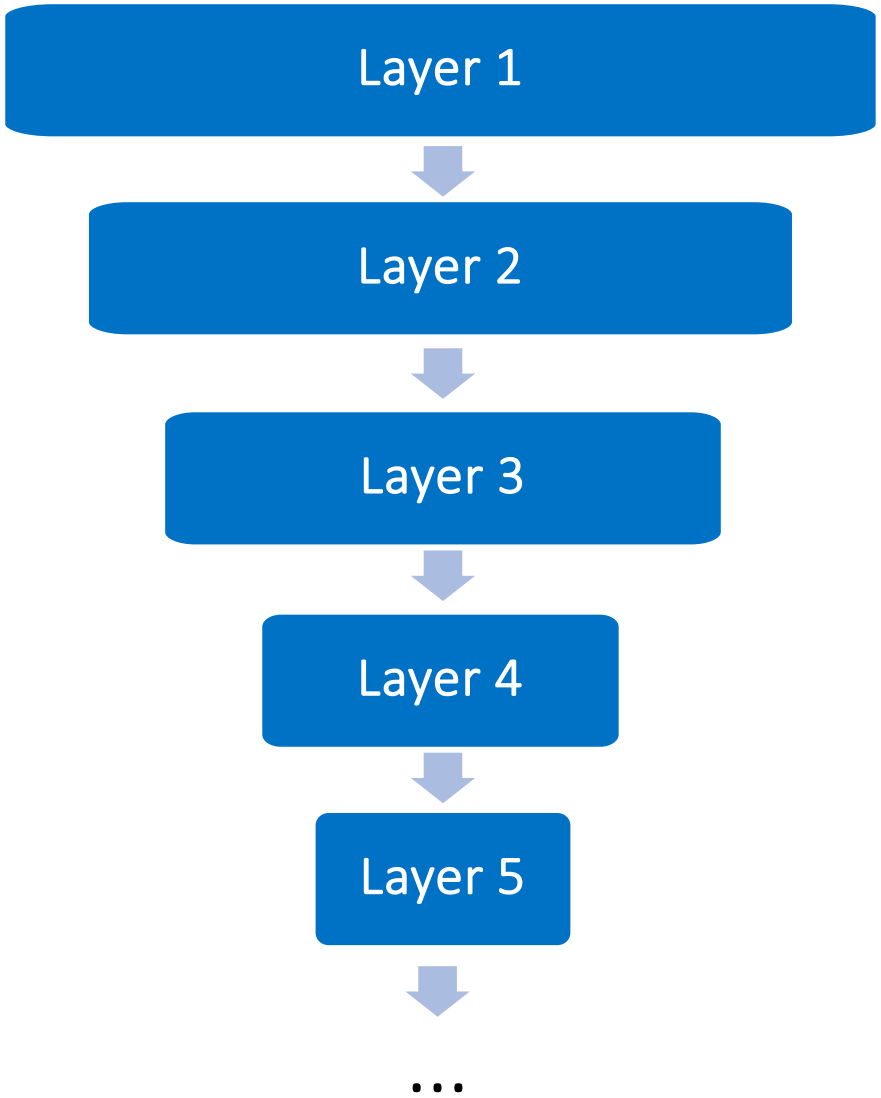
Slowing New Developers

TRAILHEAD
TECHNOLOGY PARTNERS

Lasagna Architecture

Over-Engineering Type 7

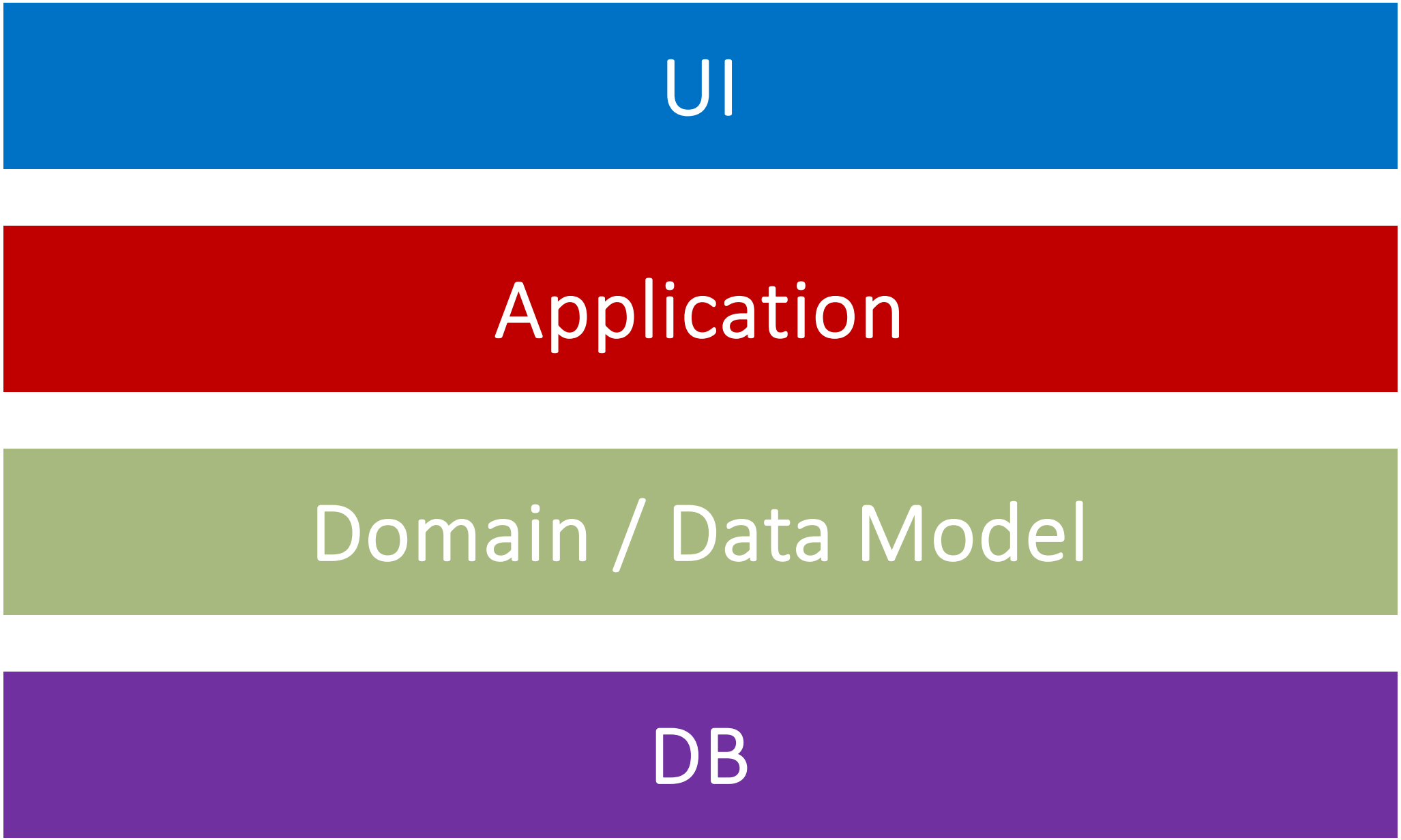
Too Many Layers

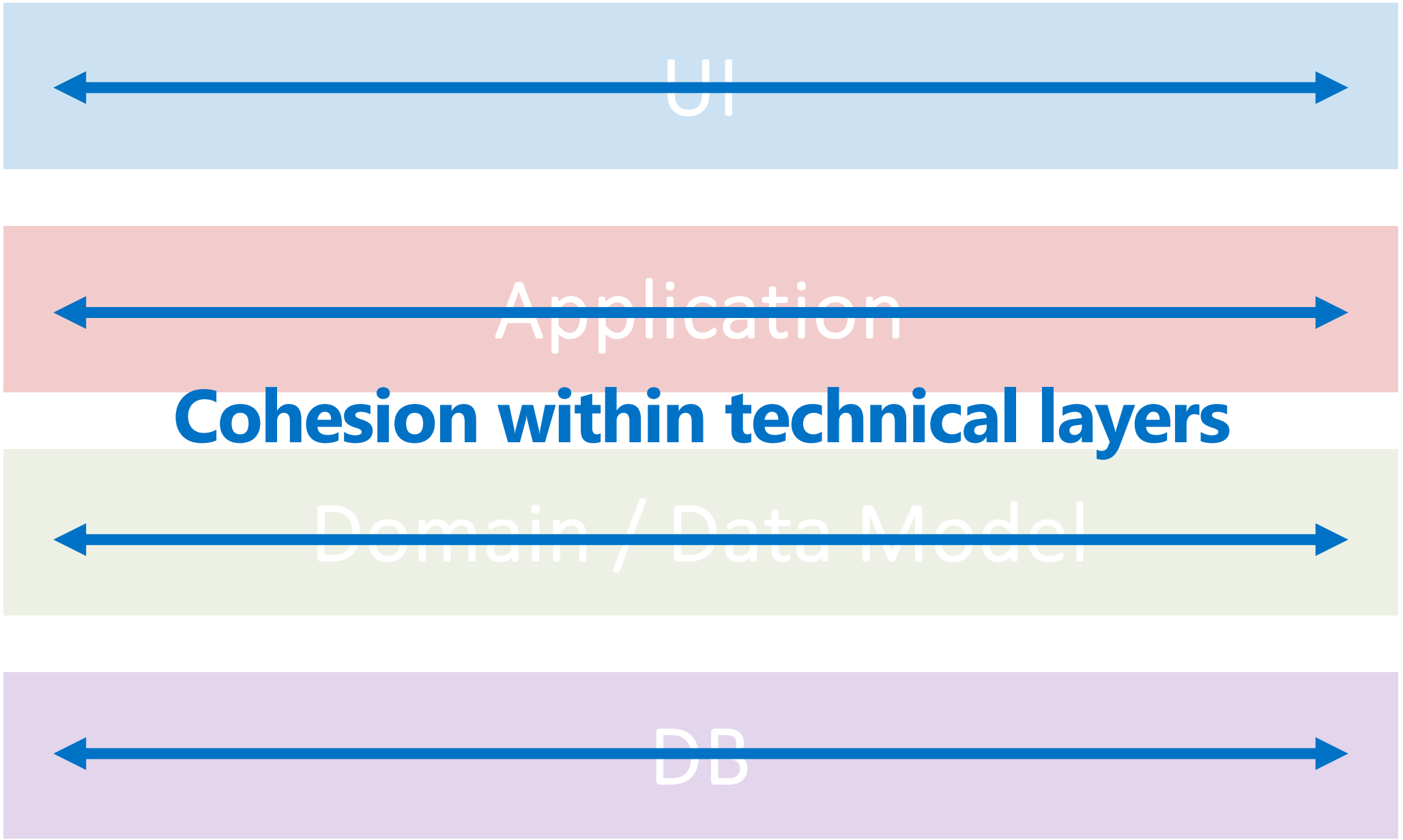


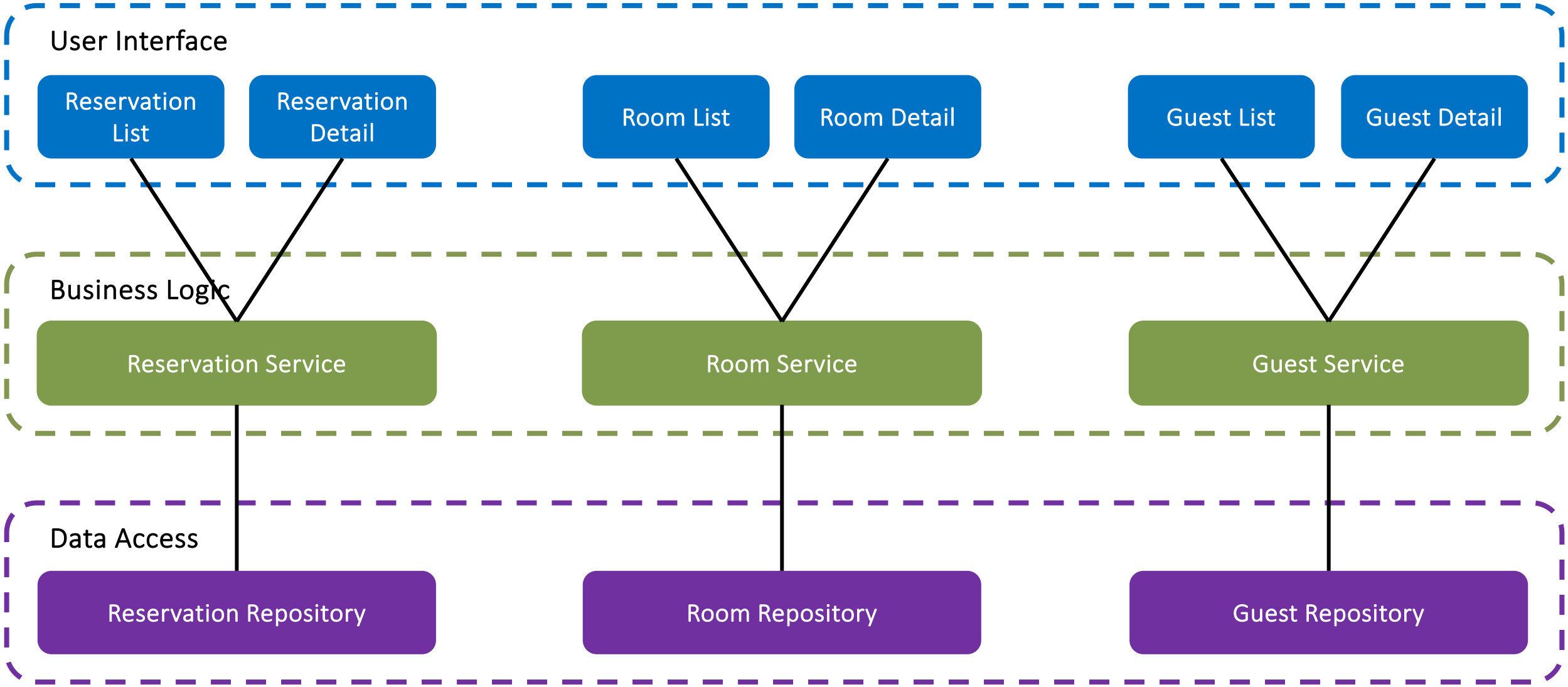
Vertical Slice Architecture

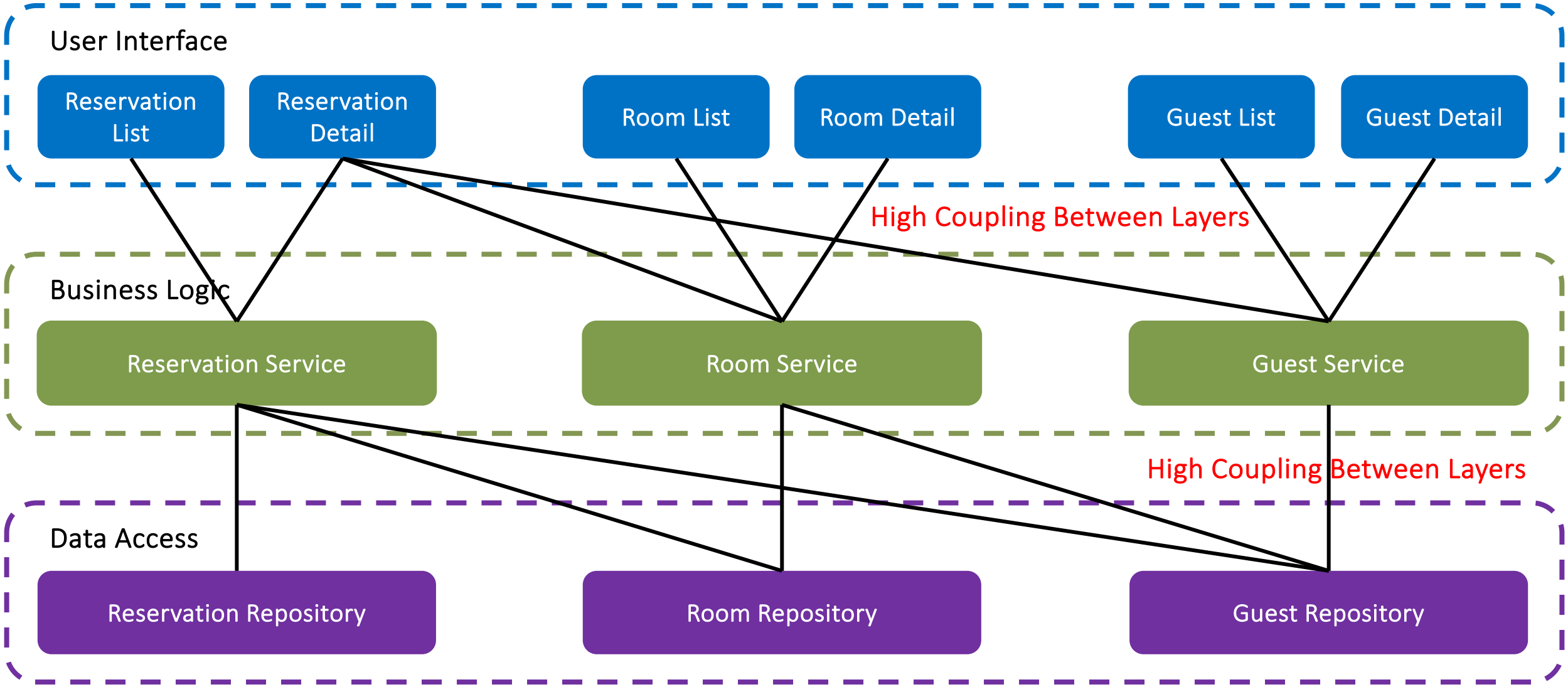
Vertical slice architecture is a **software architecture** pattern that **organizes** code **by features** or use cases **instead of technical** concerns.

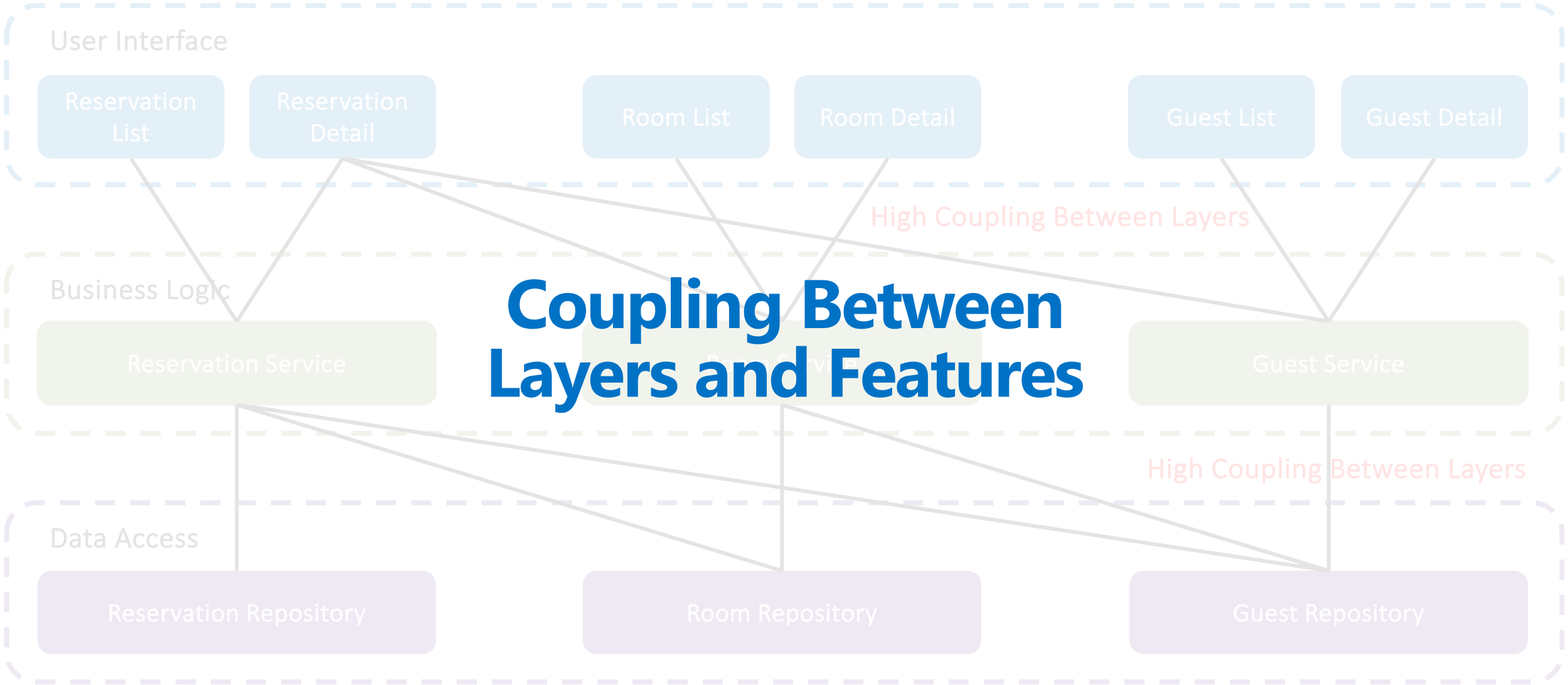
No more layers for their own sake!


UI
Application
Domain / Data Model
DB

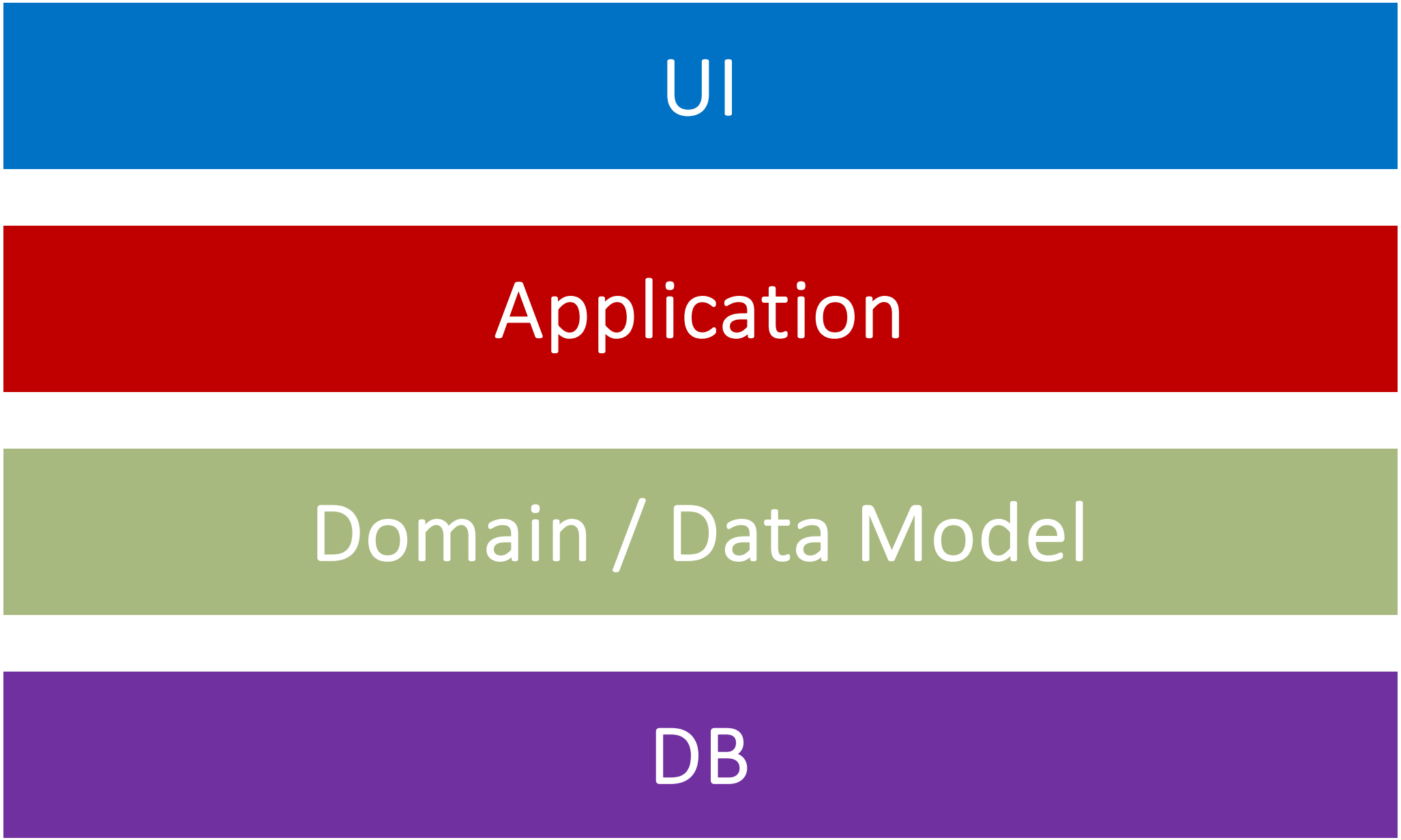

UI
Application
Cohesion within technical layers
Domain / Data Model
DB

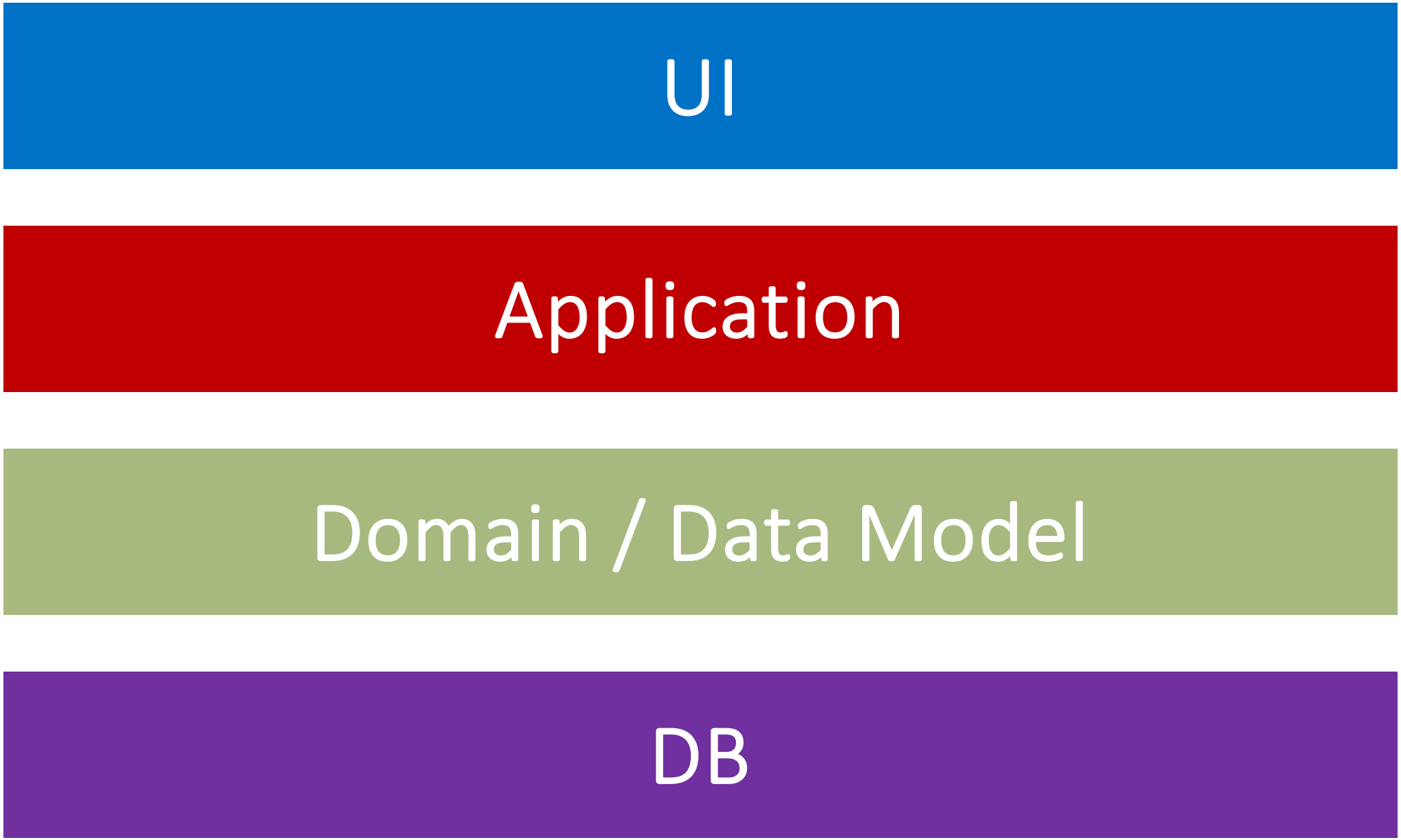

User Interface
Reservation List
Reservation Detail
Room List
Room Detail
Guest List
Guest Detail
Business Logic
Reservation Service
Room Service
Guest Service
Data Access
Reservation Repository
Room Repository
Guest Repository

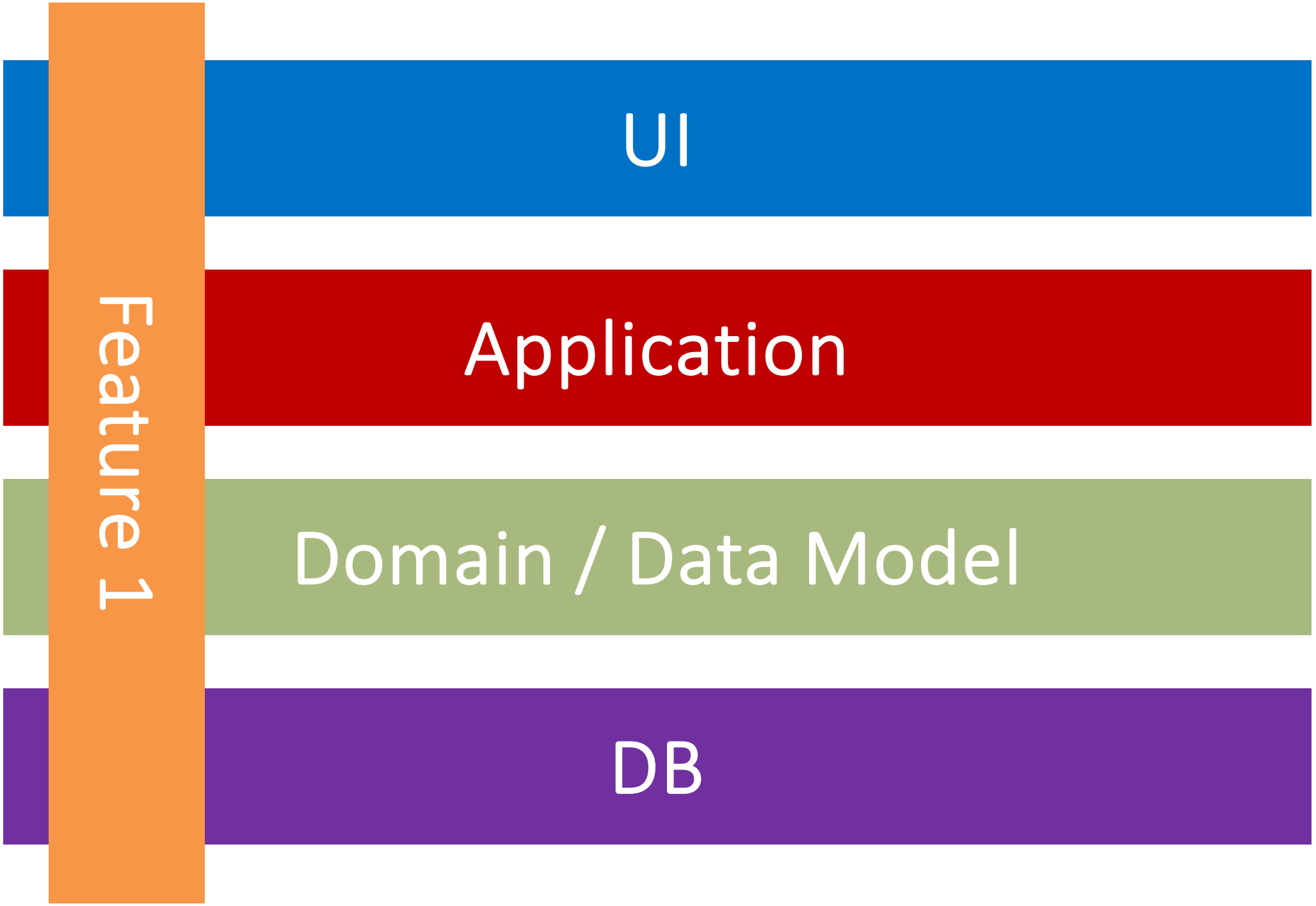

User Interface
Reservation List
Reservation Detail
Room List
Room Detail
Guest List
Guest Detail
High Coupling Between Layers
Business Logic
Reservation Service
Room Service
Guest Service
High Coupling Between Layers
Data Access
Reservation Repository
Room Repository
Guest Repository

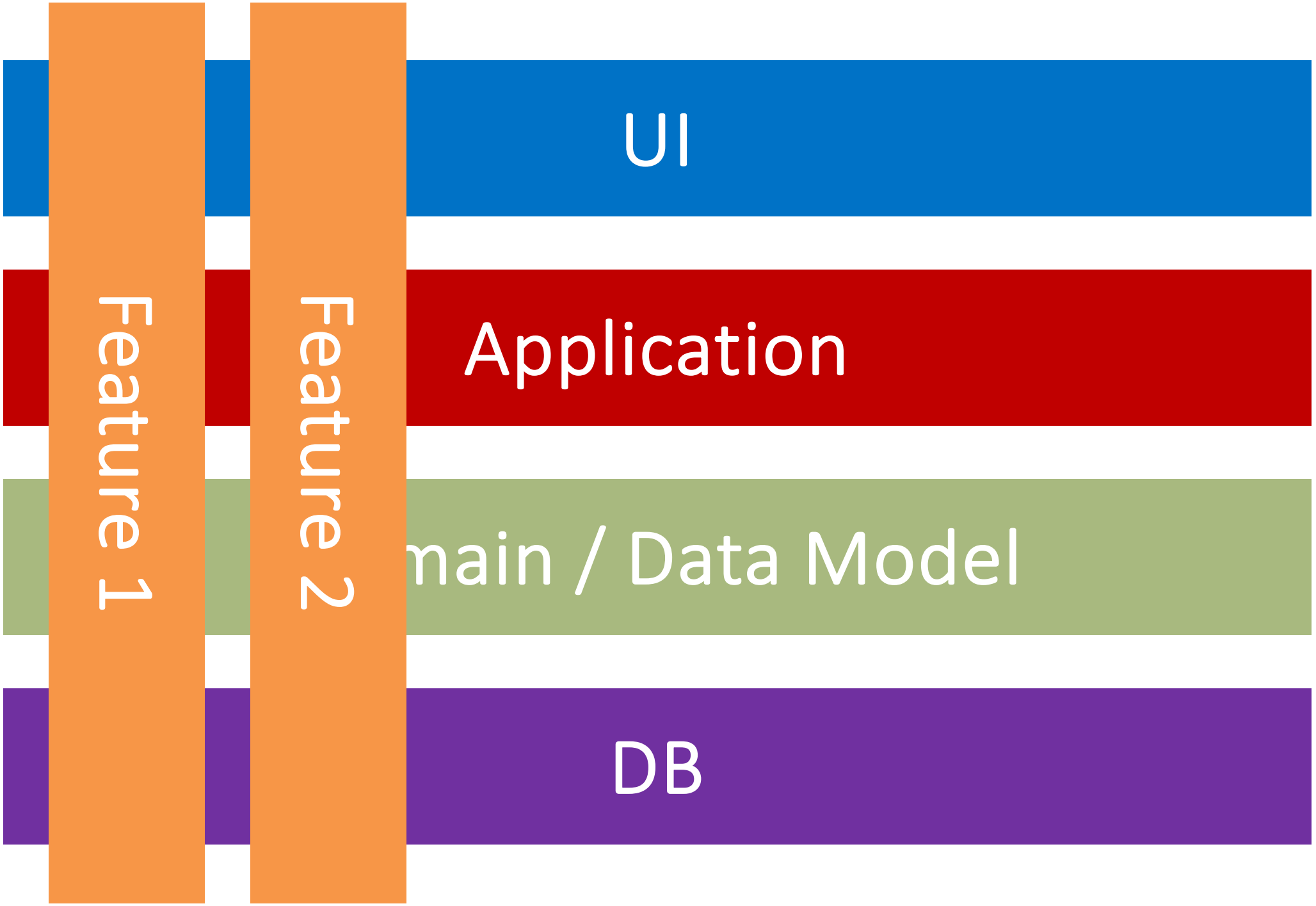


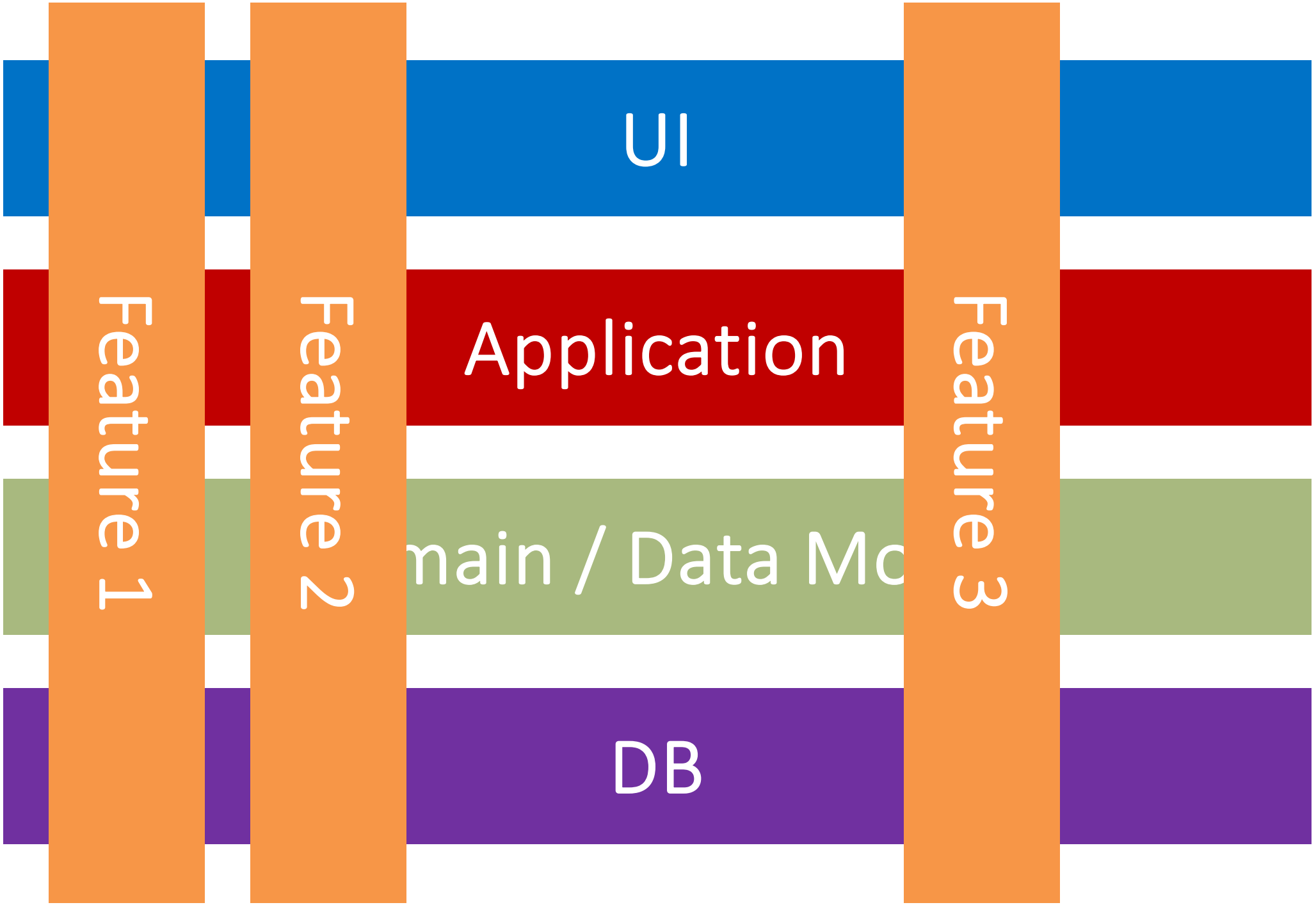
Coupling Between Layers and Features

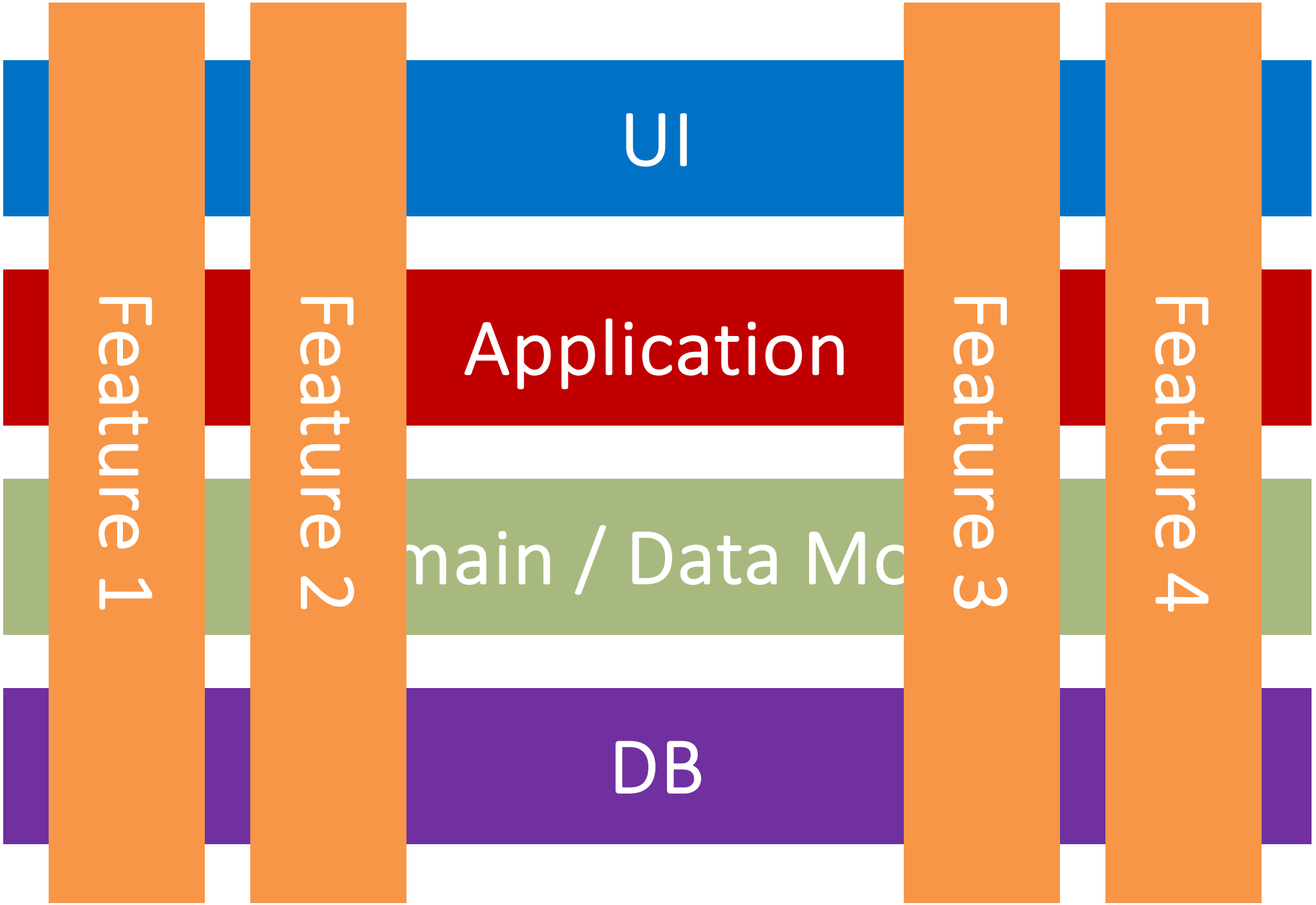

UI
Application
Domain / Data Model
DB

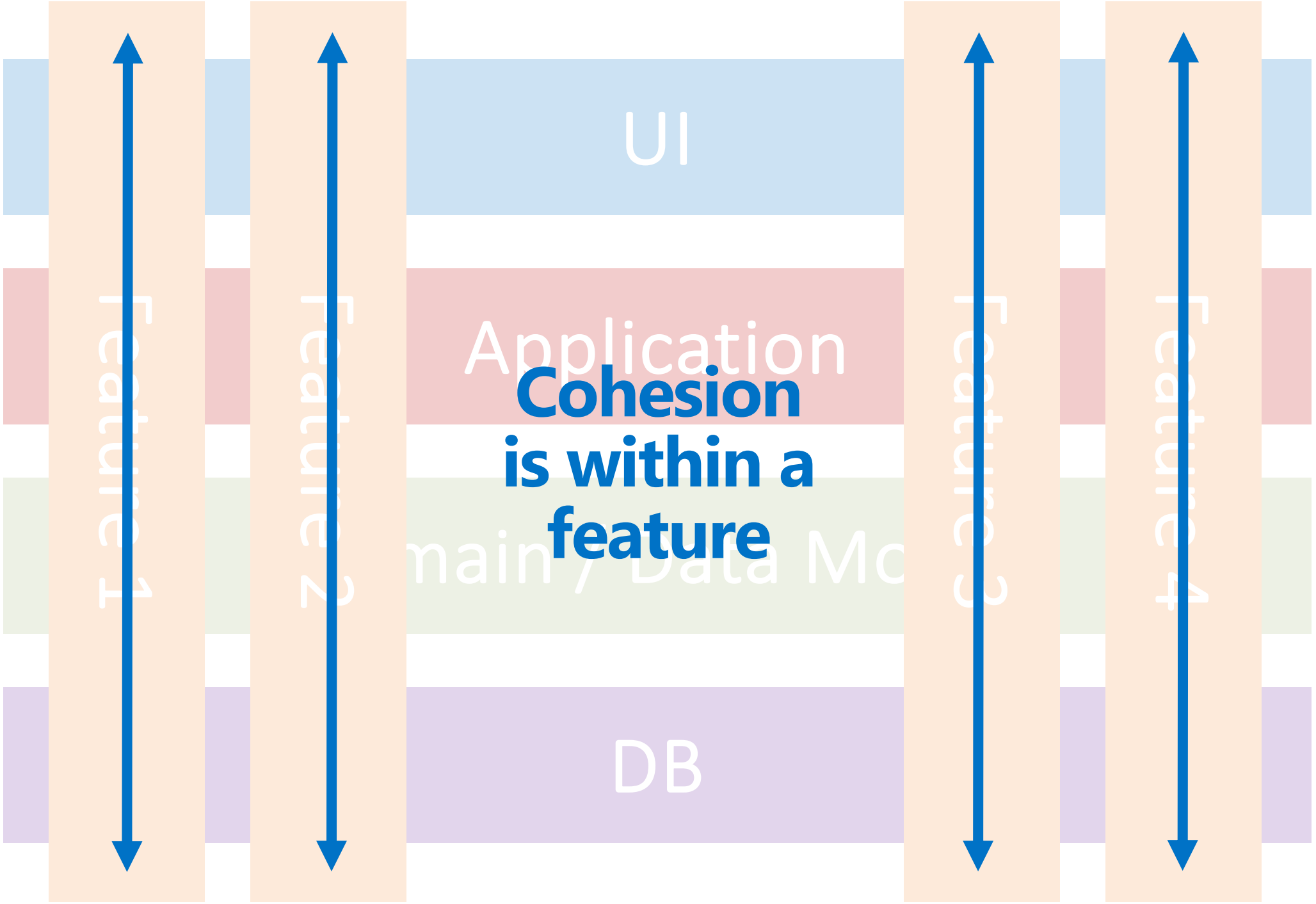

UI
Application
Domain / Data Model
DB

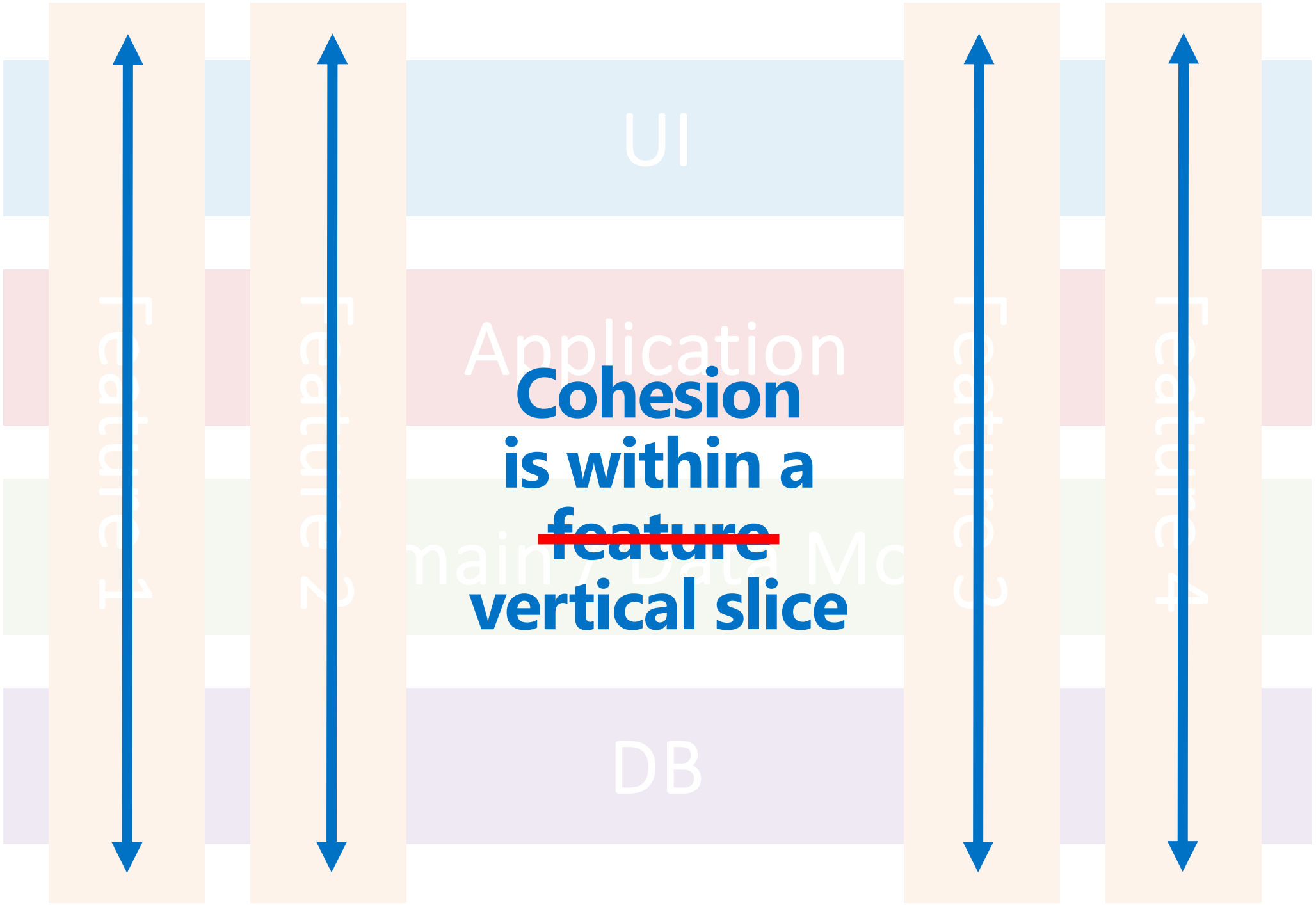

UI
Application
Domain / Data Model
DB
Feature 1


UI
Application
main / Data Model
DB
Feature 1
Feature 2


UI
Application
main / Data Mc
DB
Feature 1
Feature 2
Feature 3


UI
Application
main / Data Mo
DB
Feature 1
Feature 2
Feature 3
Feature 4


UI
Application
Domain / Data Mo
DB
Feature 1
Feature 2
Feature 3
Feature 4
Cohesion
is within a
feature



Cohesion is within a ~~feature~~ vertical slice

Shiny Object Syndrome

Over-Engineering Type 8

Which Stack Is More Fun?

Stuff You Already Know

All New Stuff

Reasons Developers Go For Shiny Objects

Curiosity and Enthusiasm

Perceived Prestige

Peer Pressure

Fear of Missing Out

Marketing and Hype

TRAILHEAD
TECHNOLOGY PARTNERS

Issues with Shiny Object Syndrome

Wrong Tool

Rewrite

Hard to Maintain

TRAILHEAD
TECHNOLOGY PARTNERS

Rolling Your Own

Over-Engineering Type 9

Reasons Developers Roll Their Own

Perceived Unique Requirements

Desire for Control

Lack of Awareness

Integration Issues

Performance Optimization

TRAILHEAD
TECHNOLOGY PARTNERS

Issues with Rolling Your Own

Increased Costs

Extended Development Time

Lack of Expertise

Reduced Testing Exposure

TRAILHEAD
TECHNOLOGY PARTNERS

Misapplied Libraries

Over-Engineering Type 10

Reasons Developers Misapply Libraries

Lack of Understanding

Overconfidence

Overcompensation

Fear of Simplicity

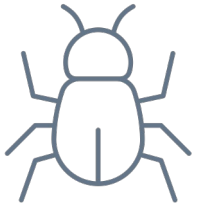
Educational Gap

TRAILHEAD
TECHNOLOGY PARTNERS

Issues with Misapplied Libraries

Increased Costs

Extended Development Time



Higher Risk Of Bugs

Slowing New Developers

TRAILHEAD
TECHNOLOGY PARTNERS

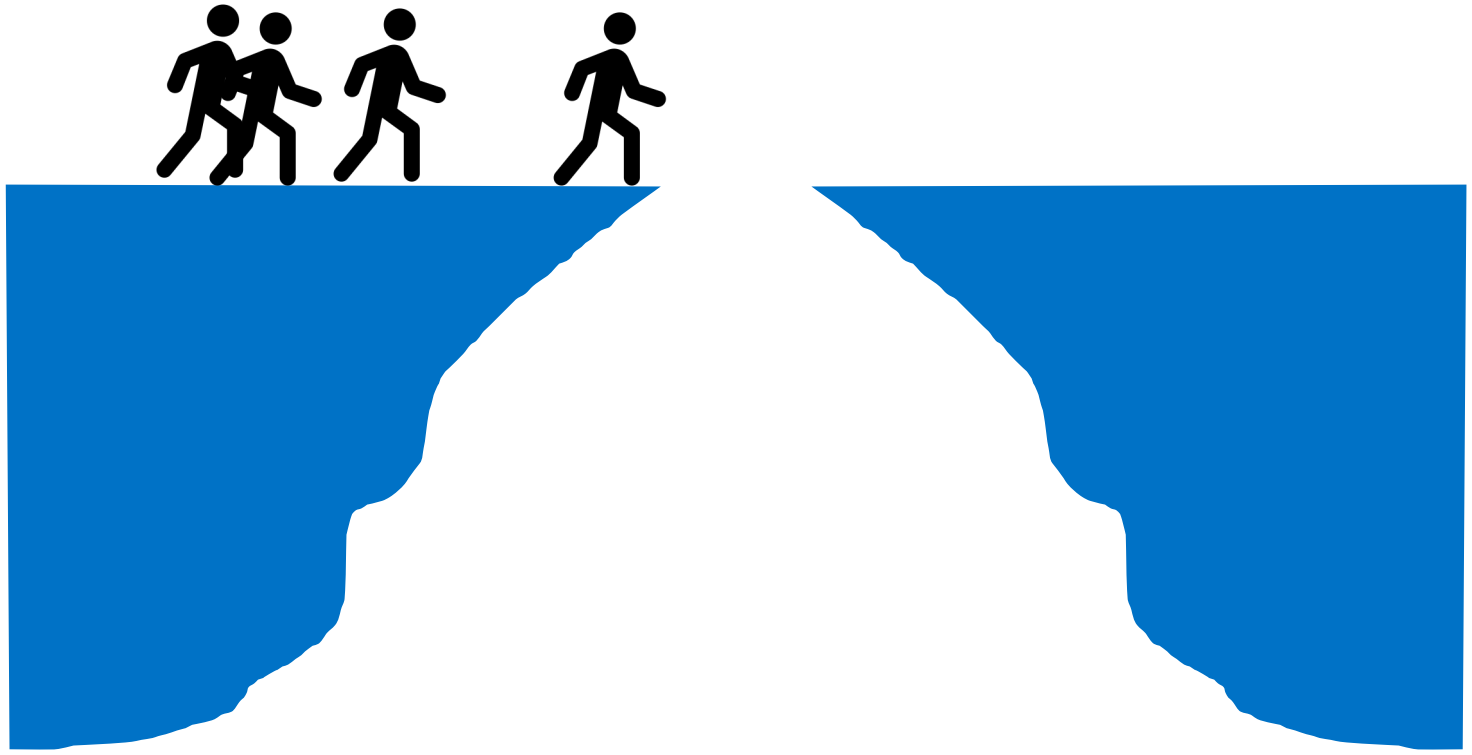
10 Rules To Avoid
Over-Engineering

TRAILHEAD
TECHNOLOGY PARTNERS

Document Engineering Decisions

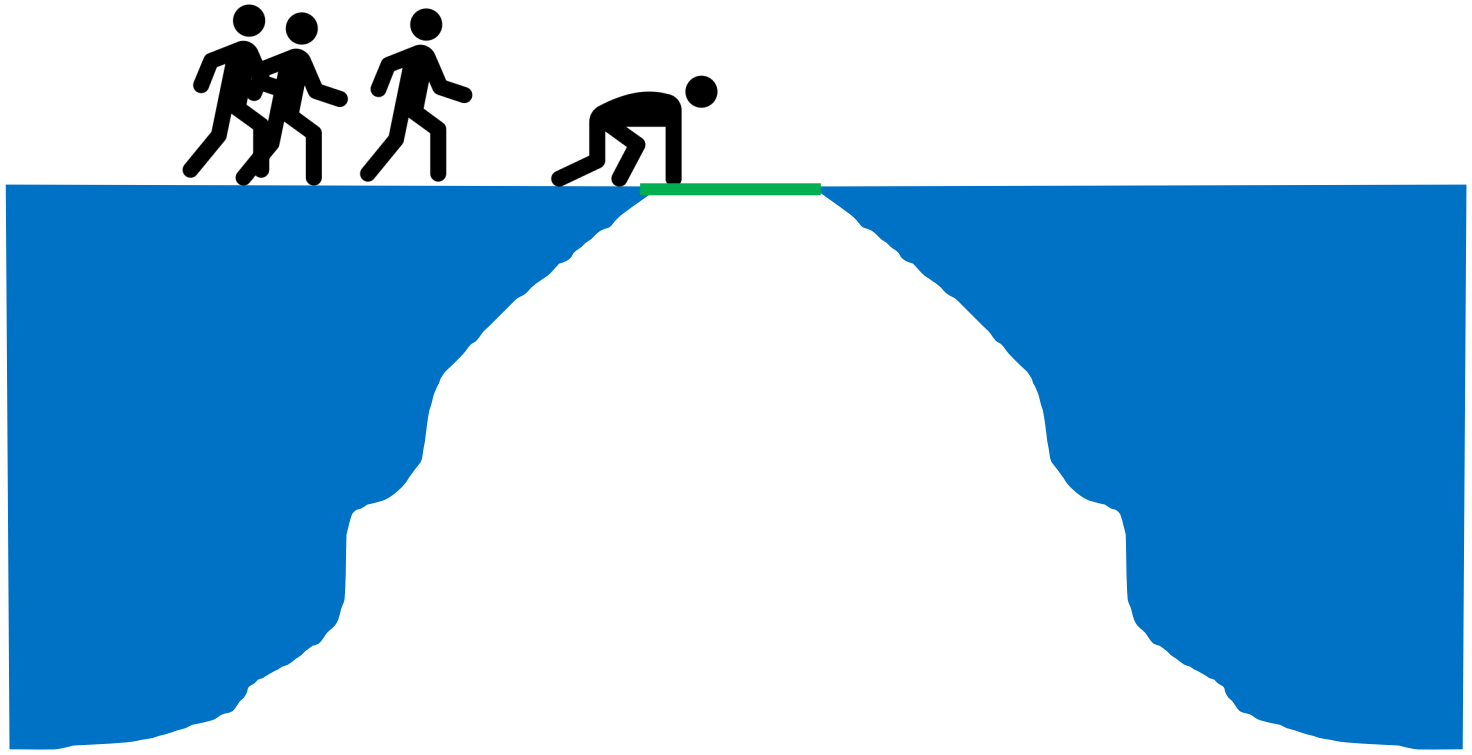
Rule 1

Avoid Mistakes In The Future



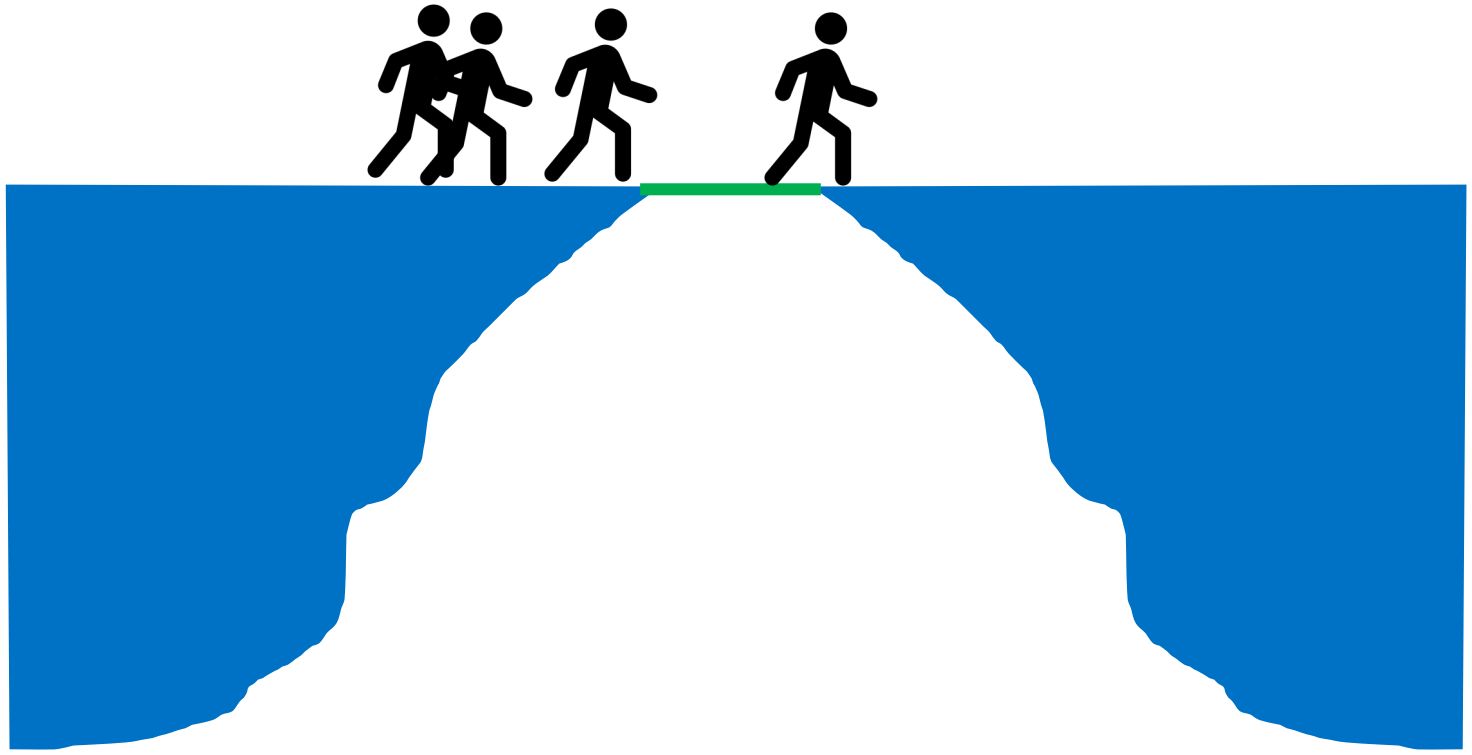
TRAILHEAD
TECHNOLOGY PARTNERS

Avoid Mistakes In The Future



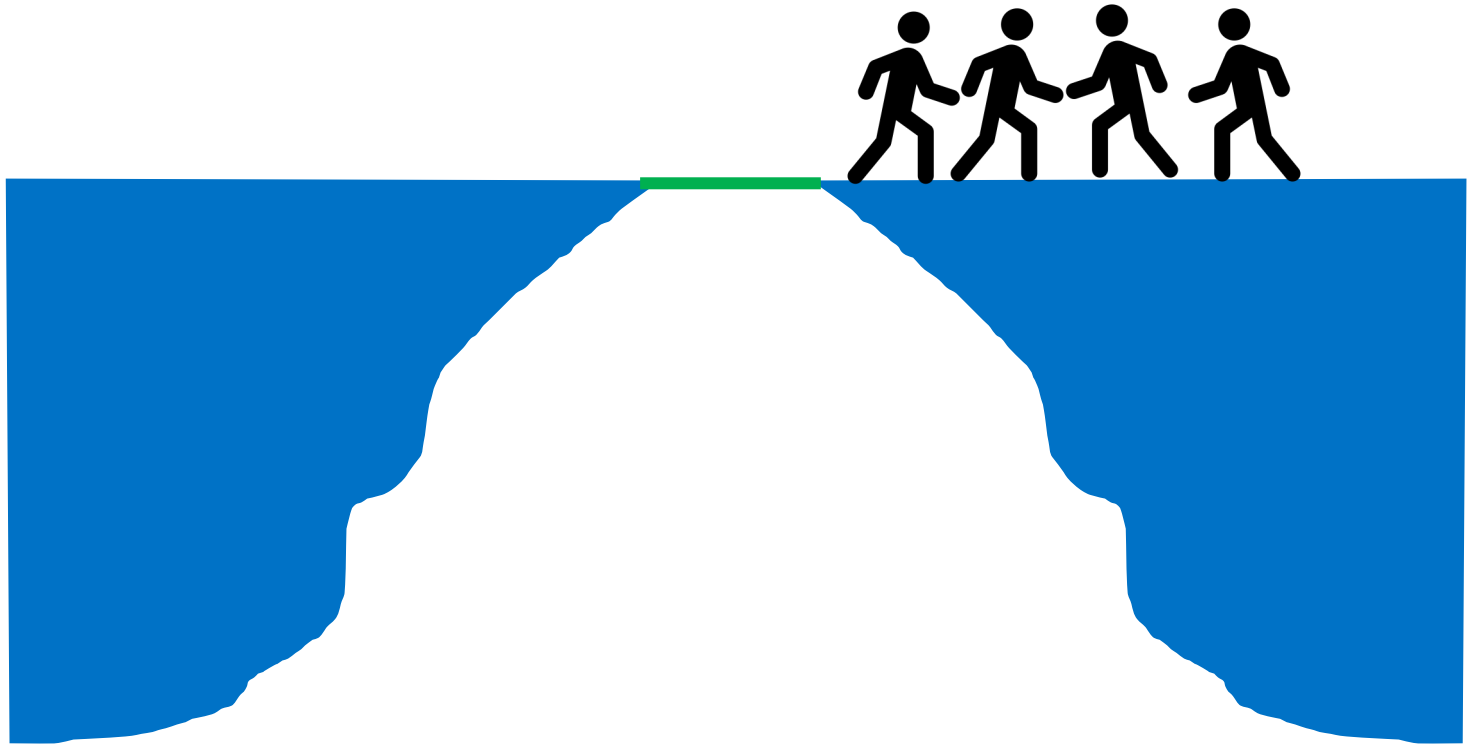
TRAILHEAD
TECHNOLOGY PARTNERS

Avoid Mistakes In The Future



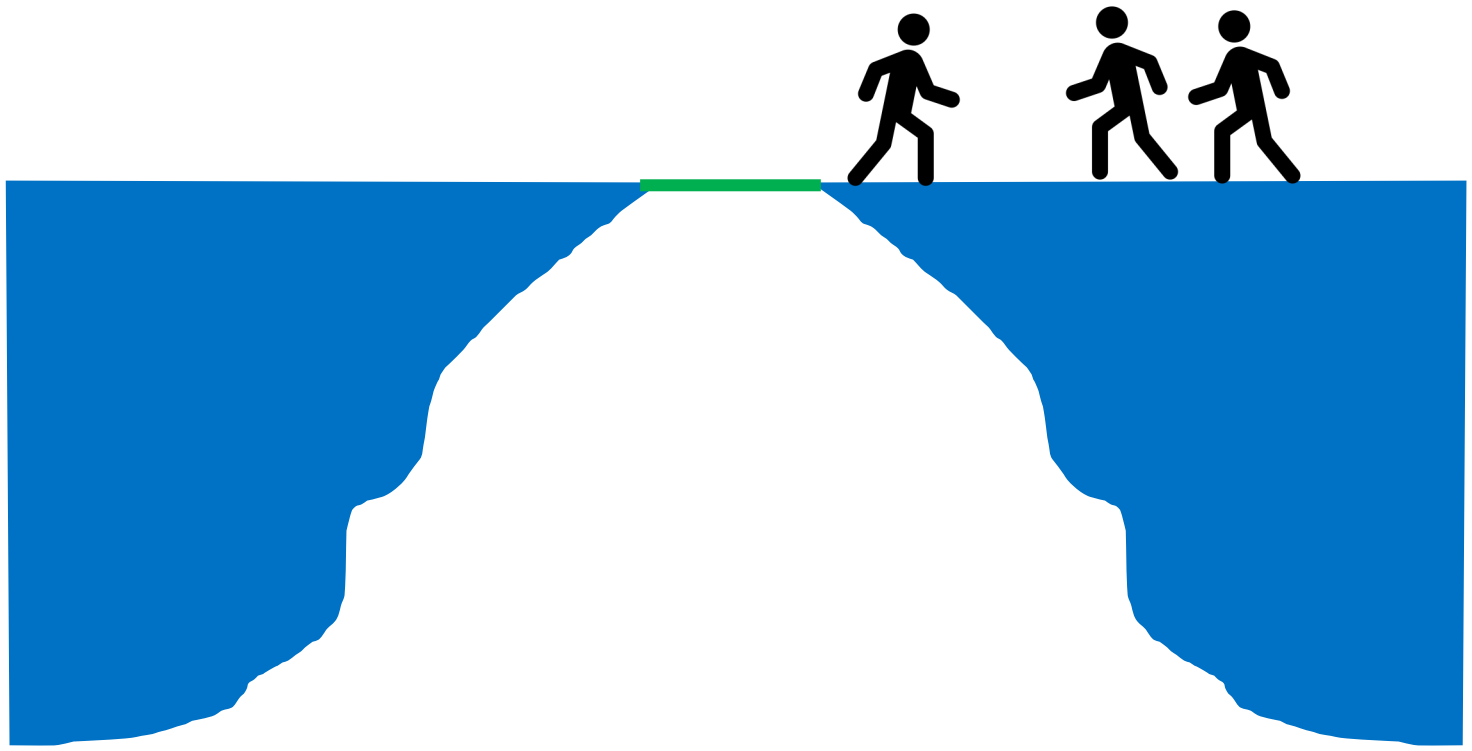
TRAILHEAD
TECHNOLOGY PARTNERS

Avoid Mistakes In The Future



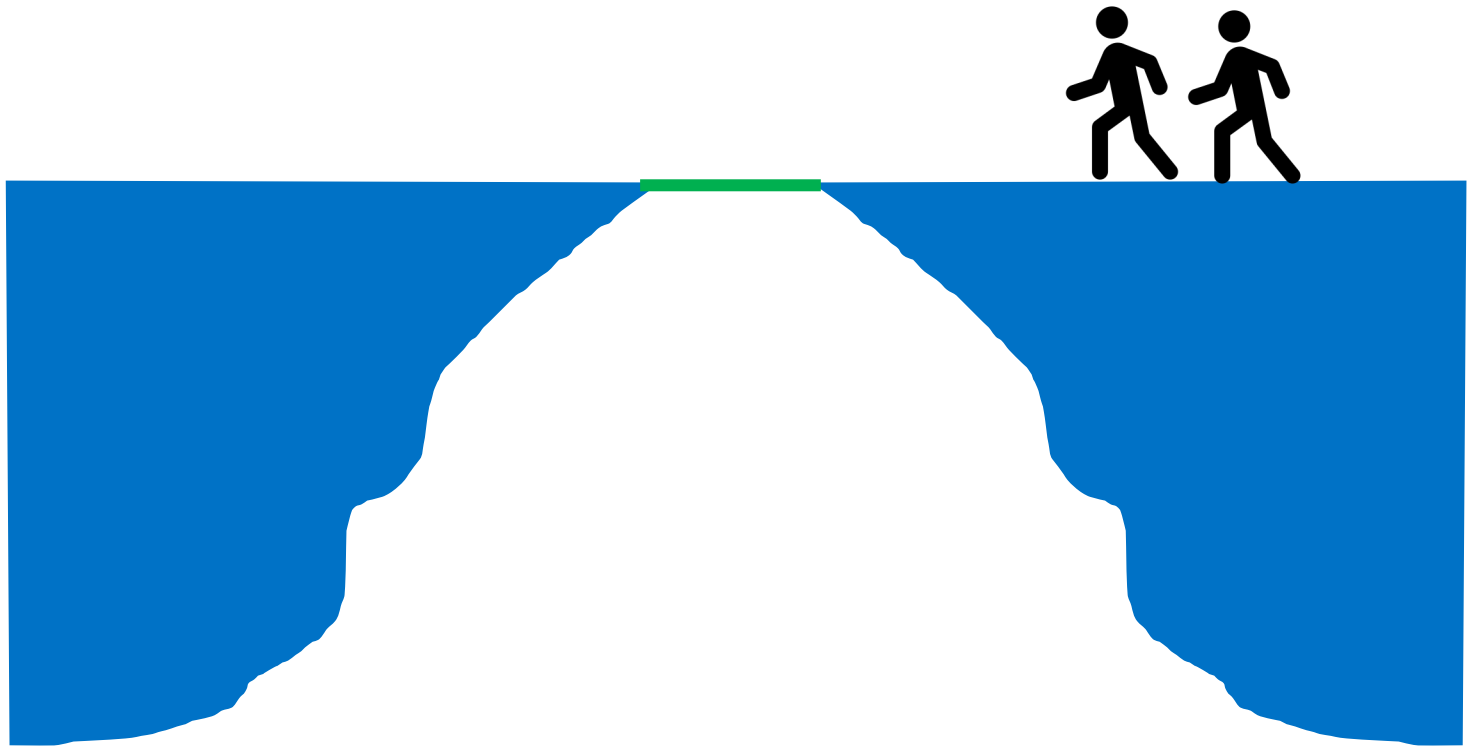
TRAILHEAD
TECHNOLOGY PARTNERS

Avoid Mistakes In The Future



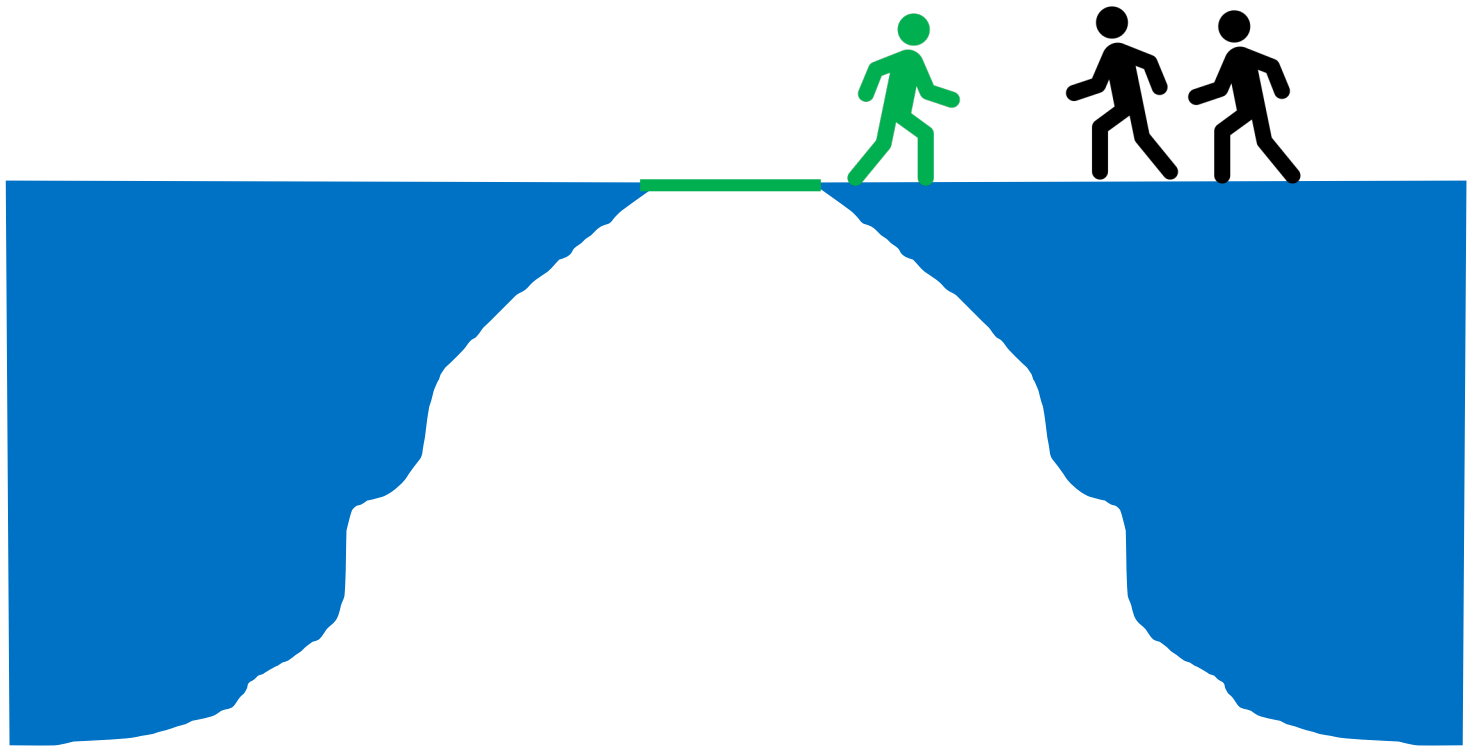
TRAILHEAD
TECHNOLOGY PARTNERS

Avoid Mistakes In The Future



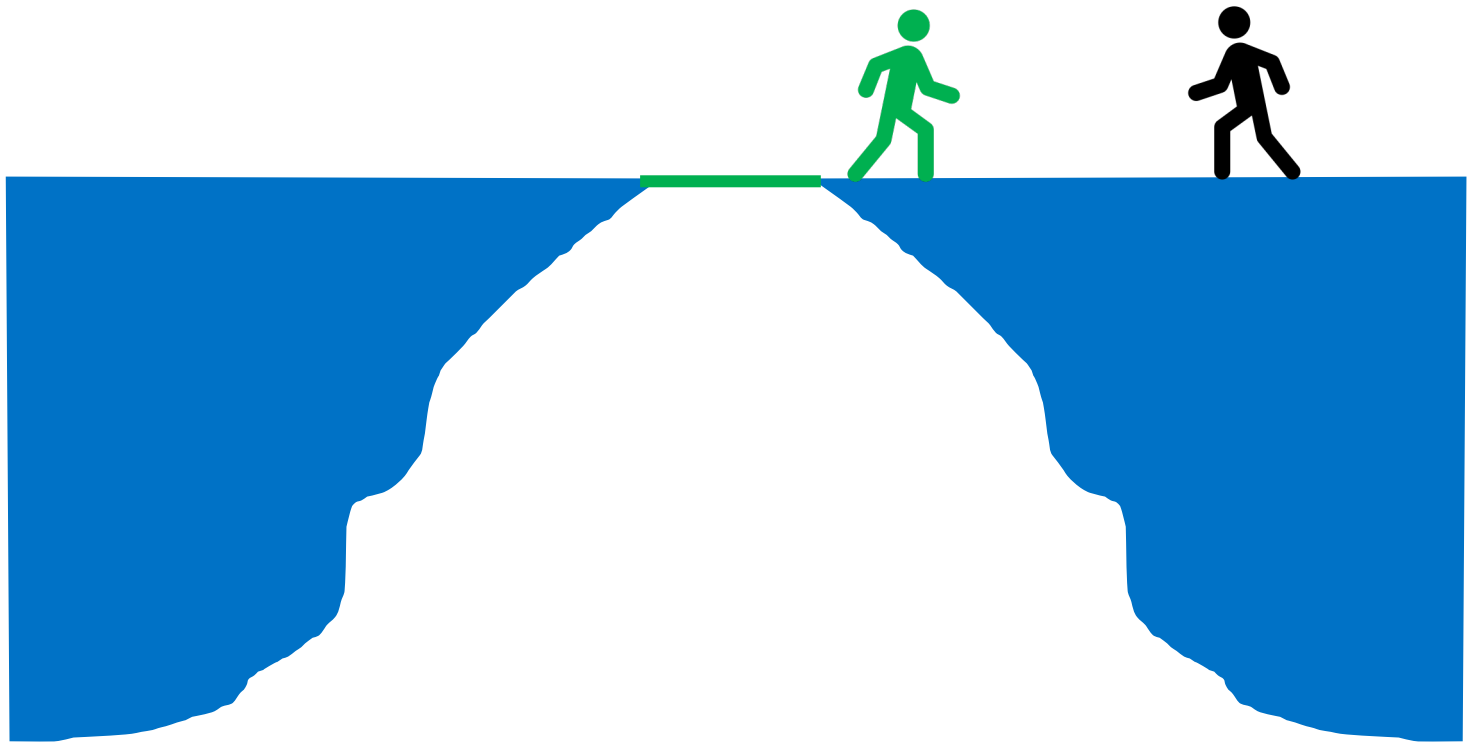
TRAILHEAD
TECHNOLOGY PARTNERS

Avoid Mistakes In The Future



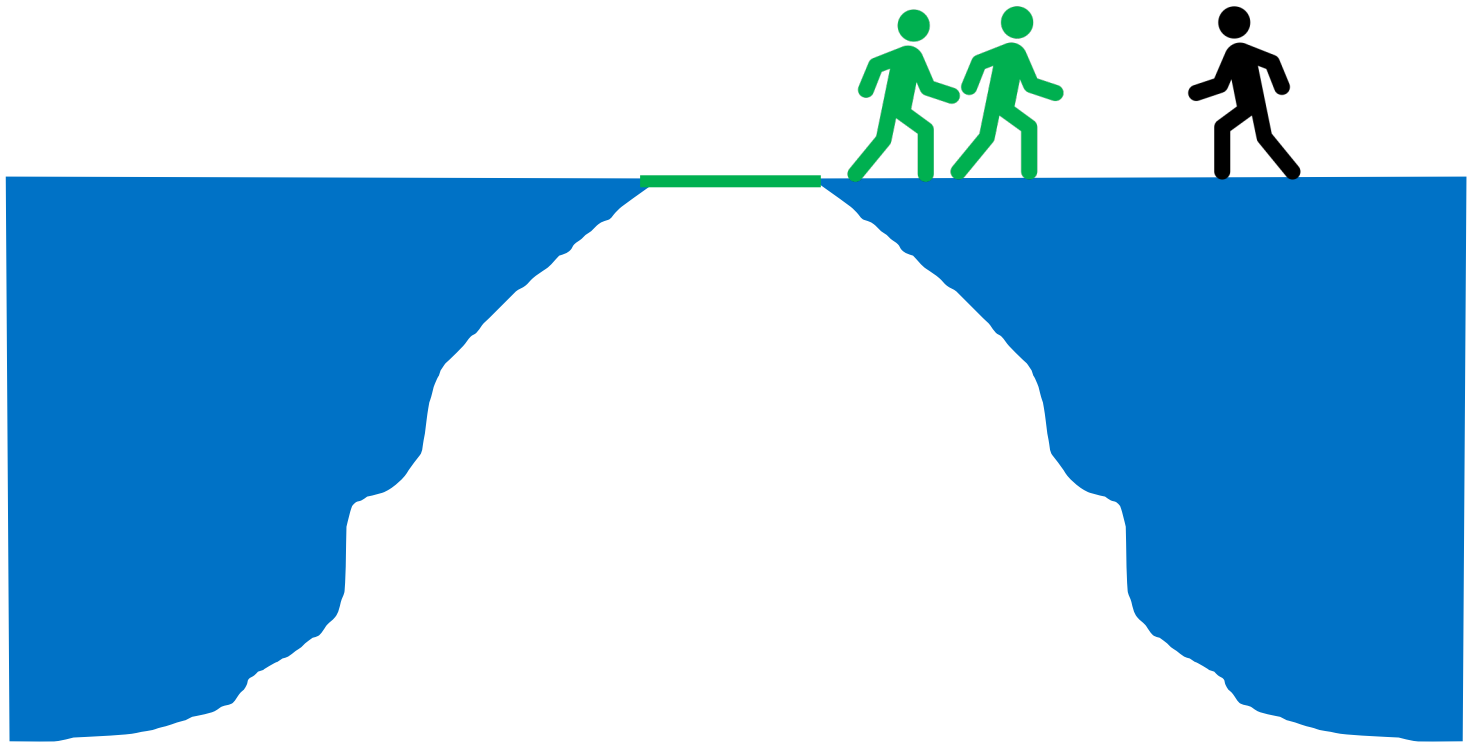
TRAILHEAD
TECHNOLOGY PARTNERS

Avoid Mistakes In The Future



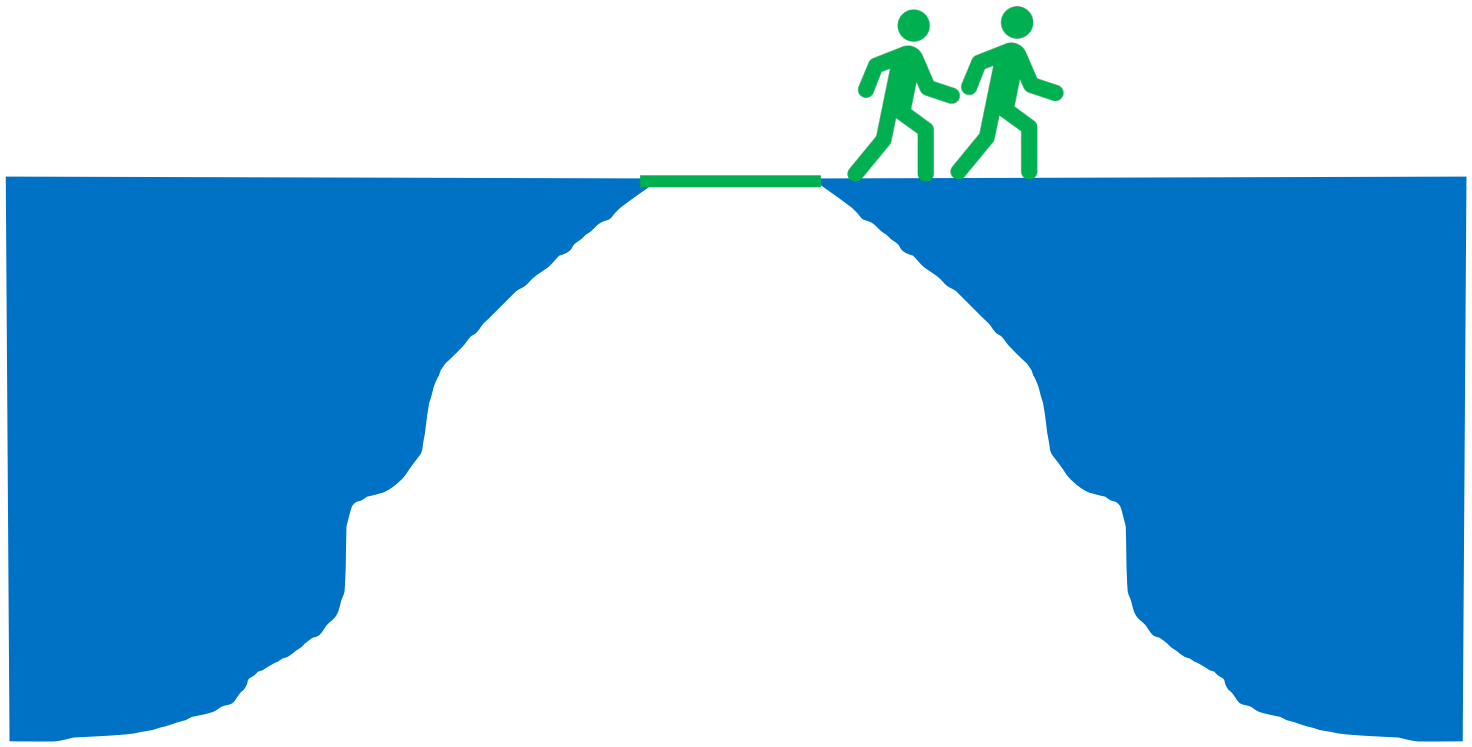
TRAILHEAD
TECHNOLOGY PARTNERS

Avoid Mistakes In The Future



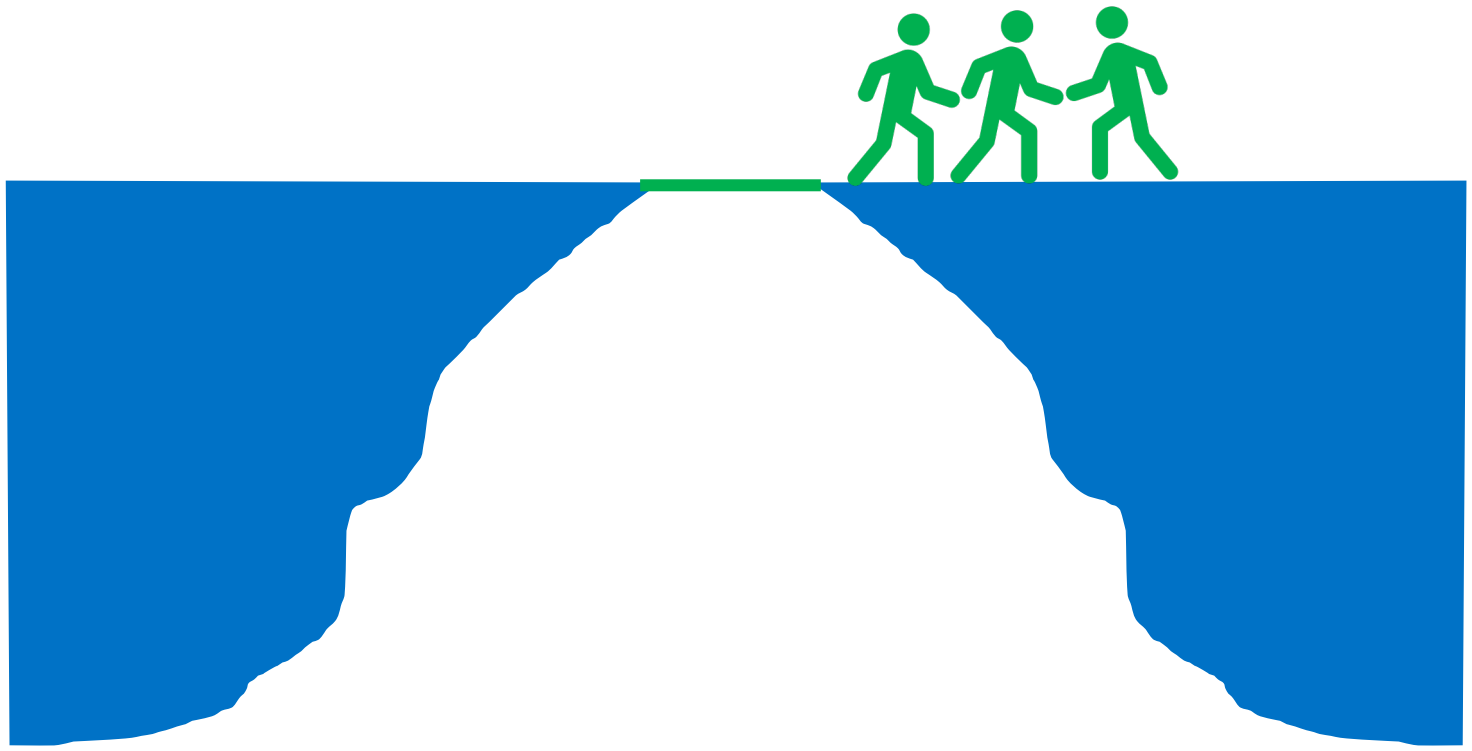
TRAILHEAD
TECHNOLOGY PARTNERS

Avoid Mistakes In The Future



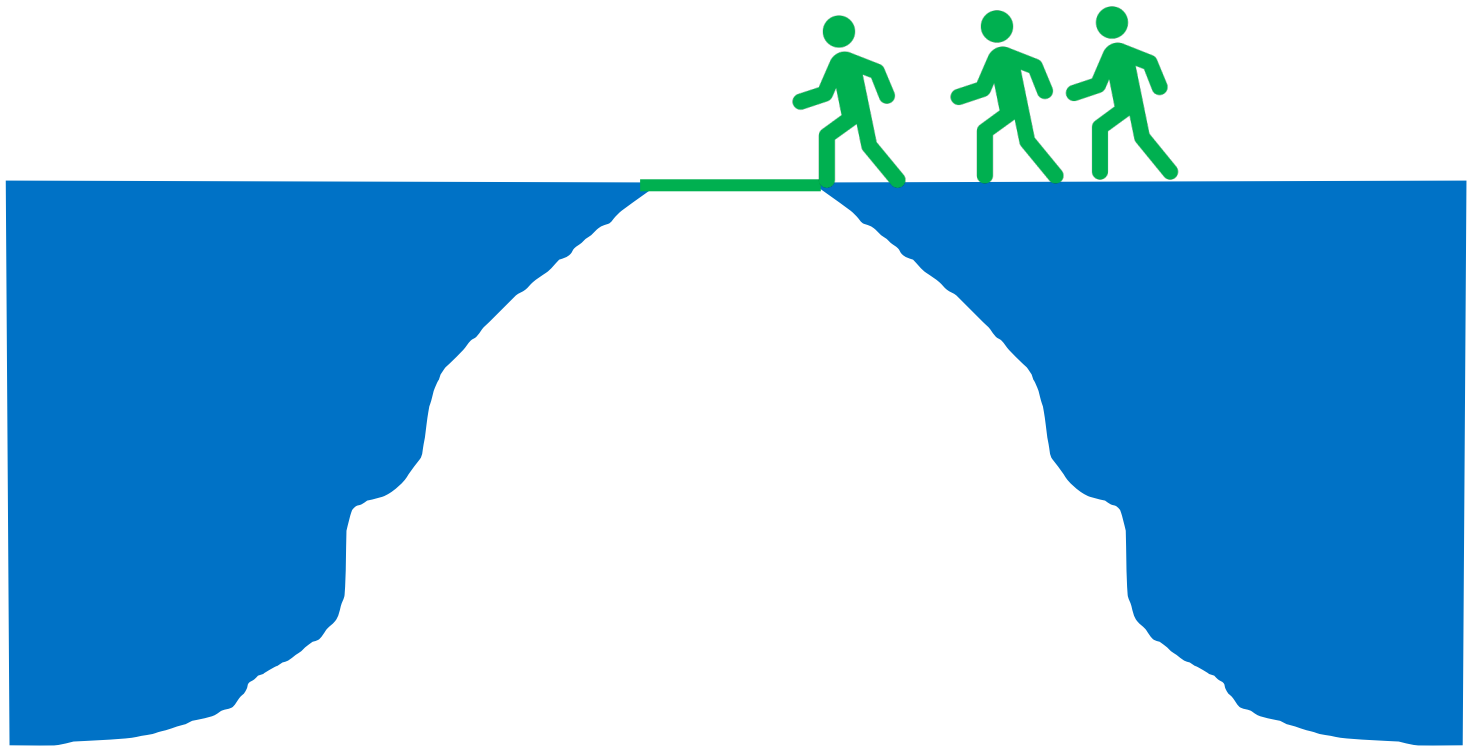
TRAILHEAD
TECHNOLOGY PARTNERS

Avoid Mistakes In The Future



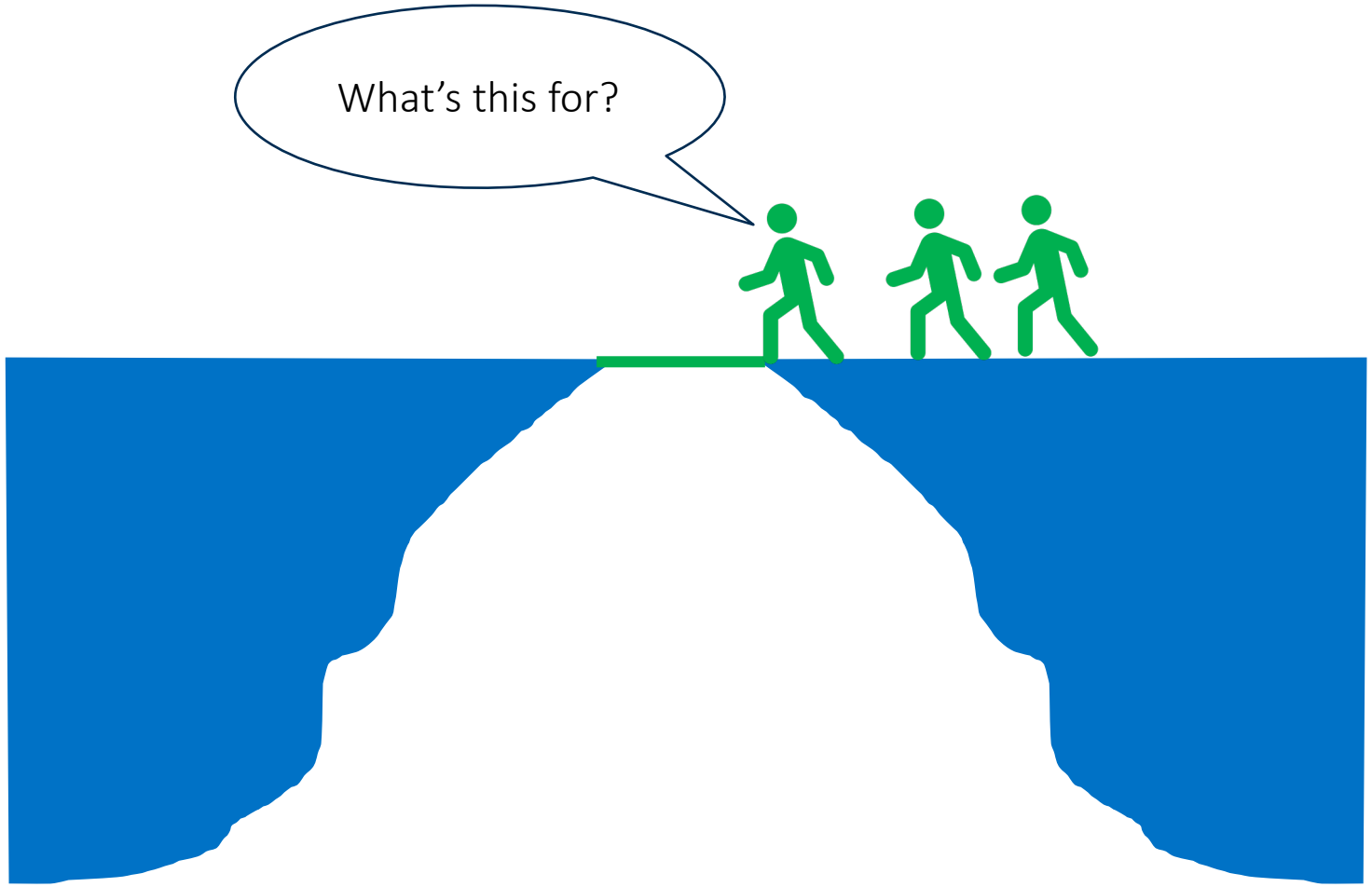
TRAILHEAD
TECHNOLOGY PARTNERS

Avoid Mistakes In The Future



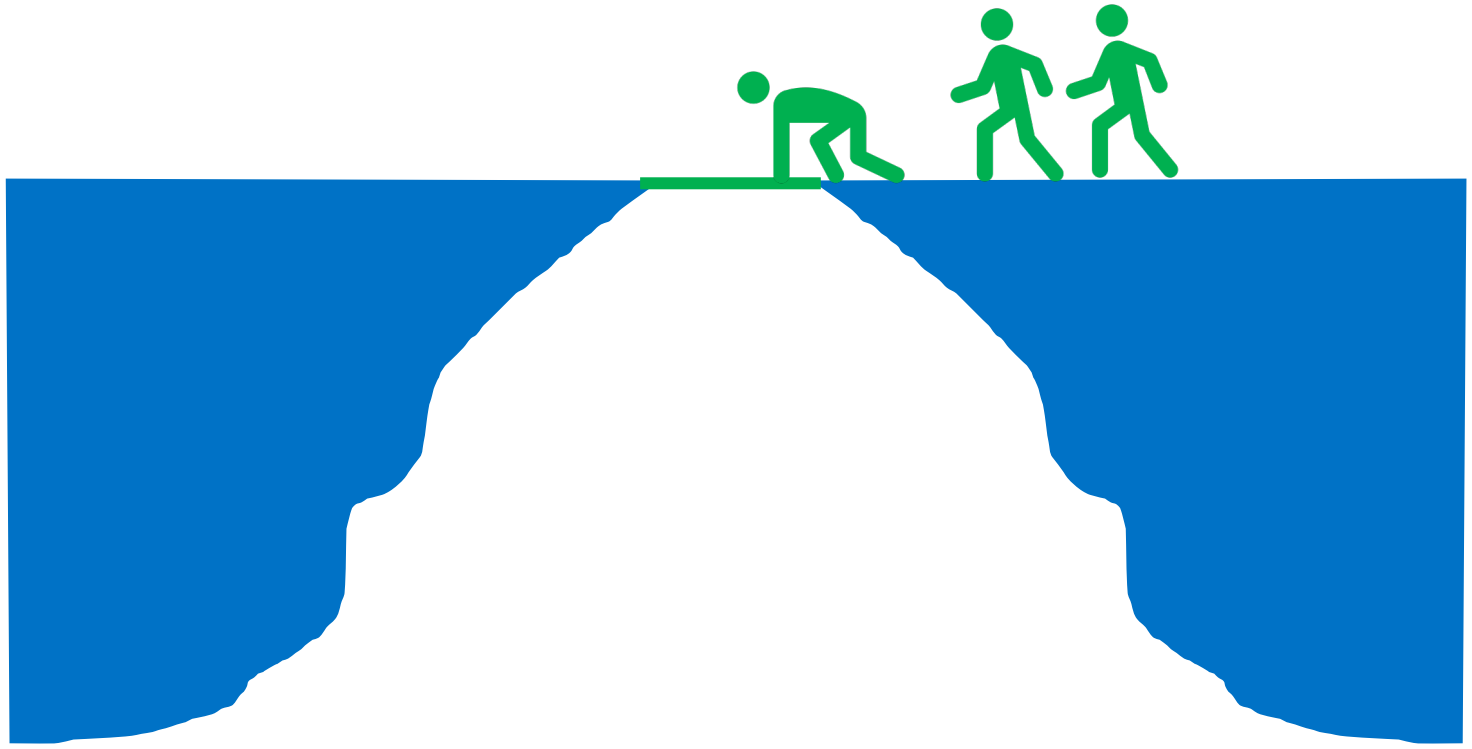
TRAILHEAD
TECHNOLOGY PARTNERS

Avoid Mistakes In The Future



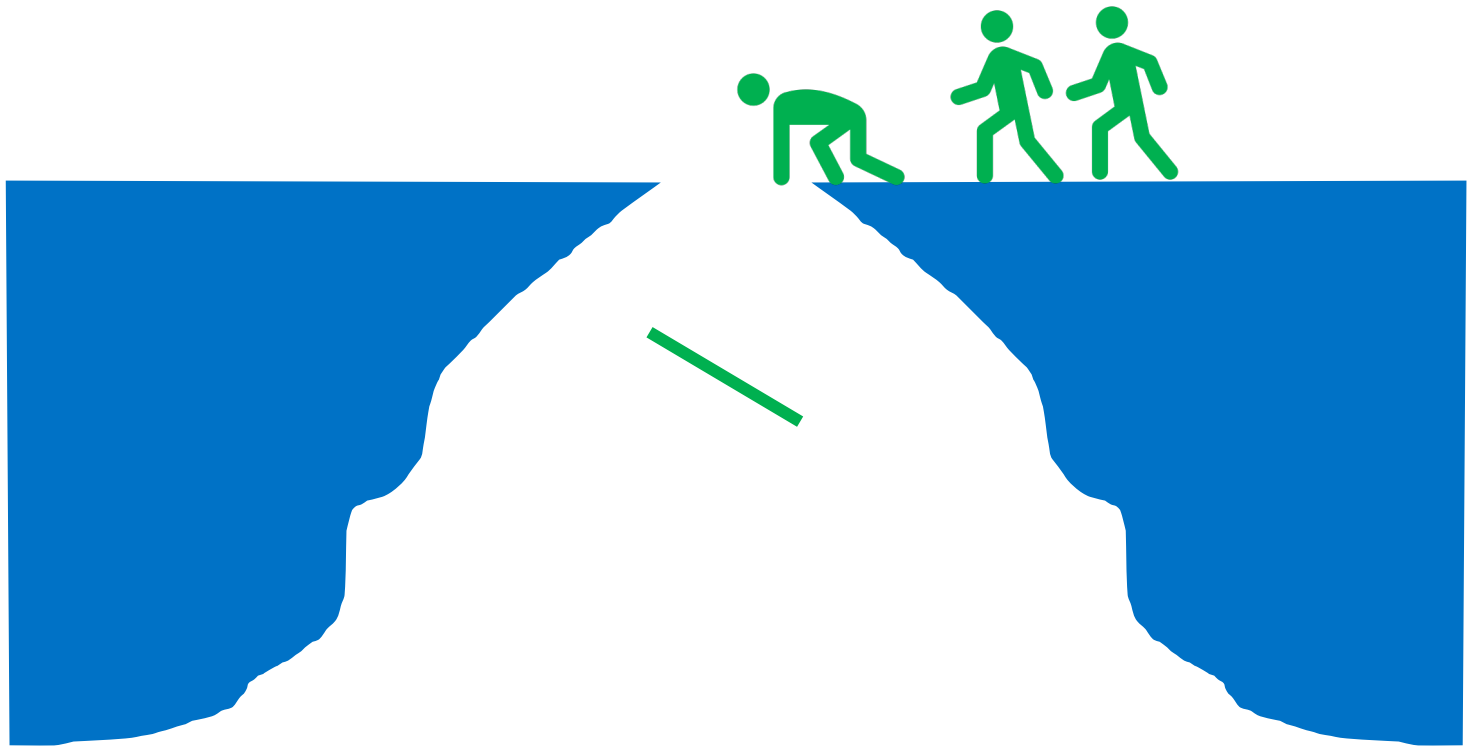
TRAILHEAD
TECHNOLOGY PARTNERS

Avoid Mistakes In The Future



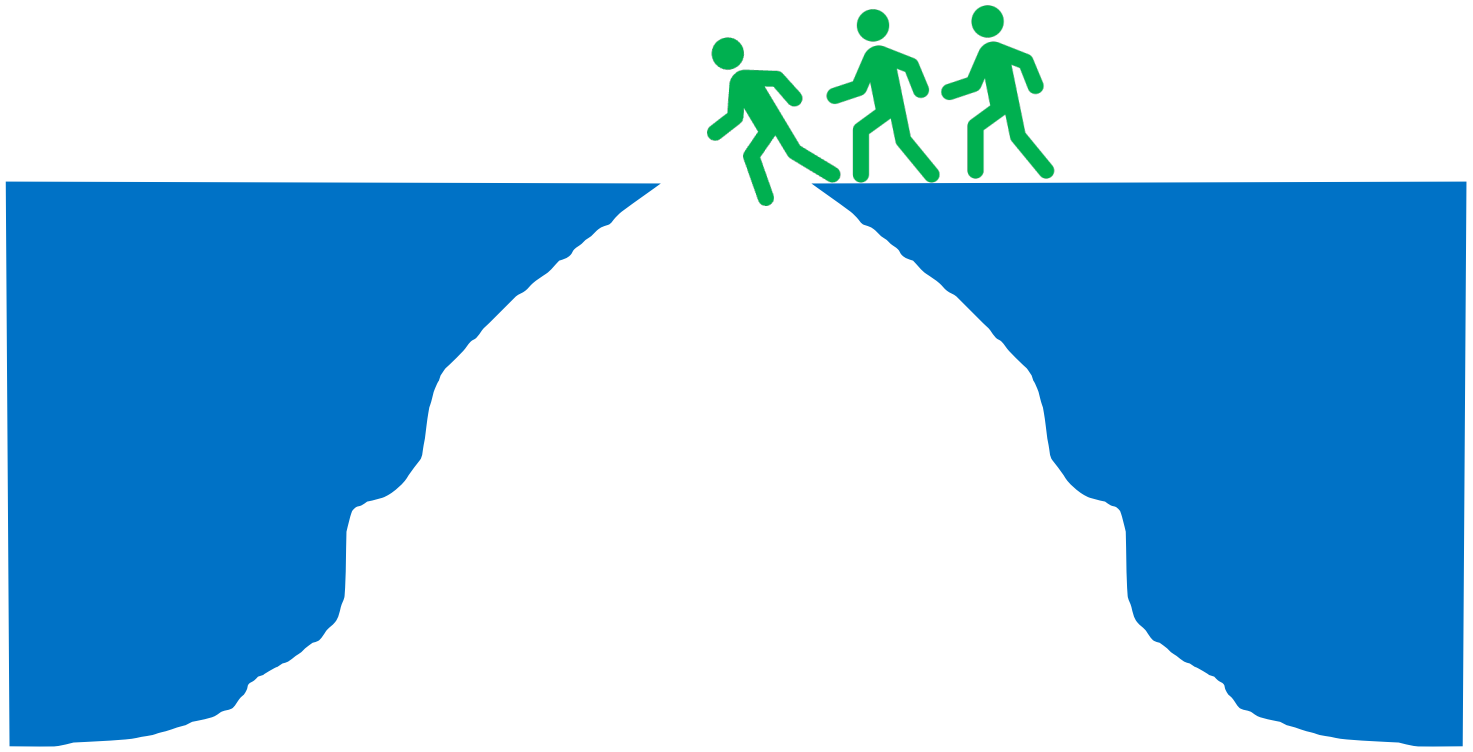
TRAILHEAD
TECHNOLOGY PARTNERS

Avoid Mistakes In The Future



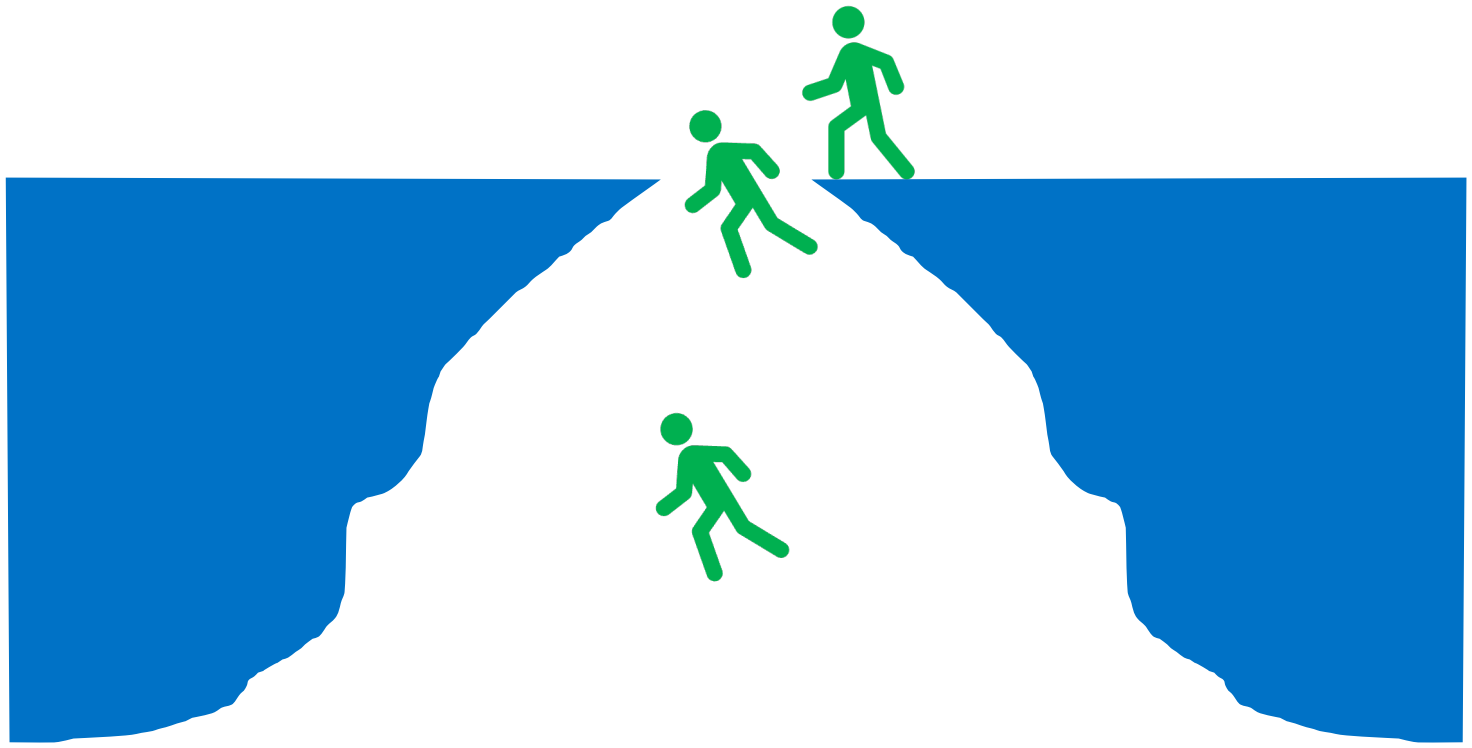
TRAILHEAD
TECHNOLOGY PARTNERS

Avoid Mistakes In The Future



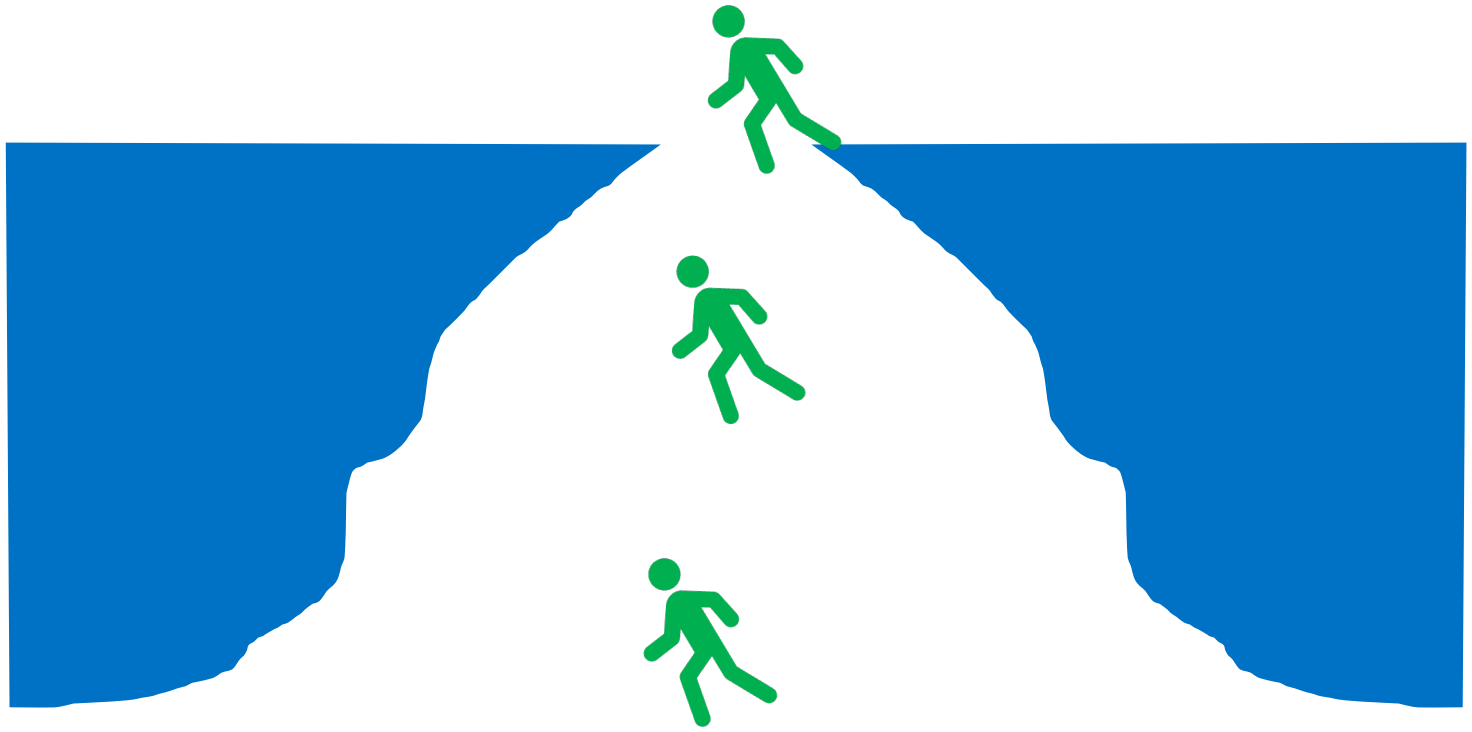
TRAILHEAD
TECHNOLOGY PARTNERS

Avoid Mistakes In The Future



TRAILHEAD
TECHNOLOGY PARTNERS

Avoid Mistakes In The Future

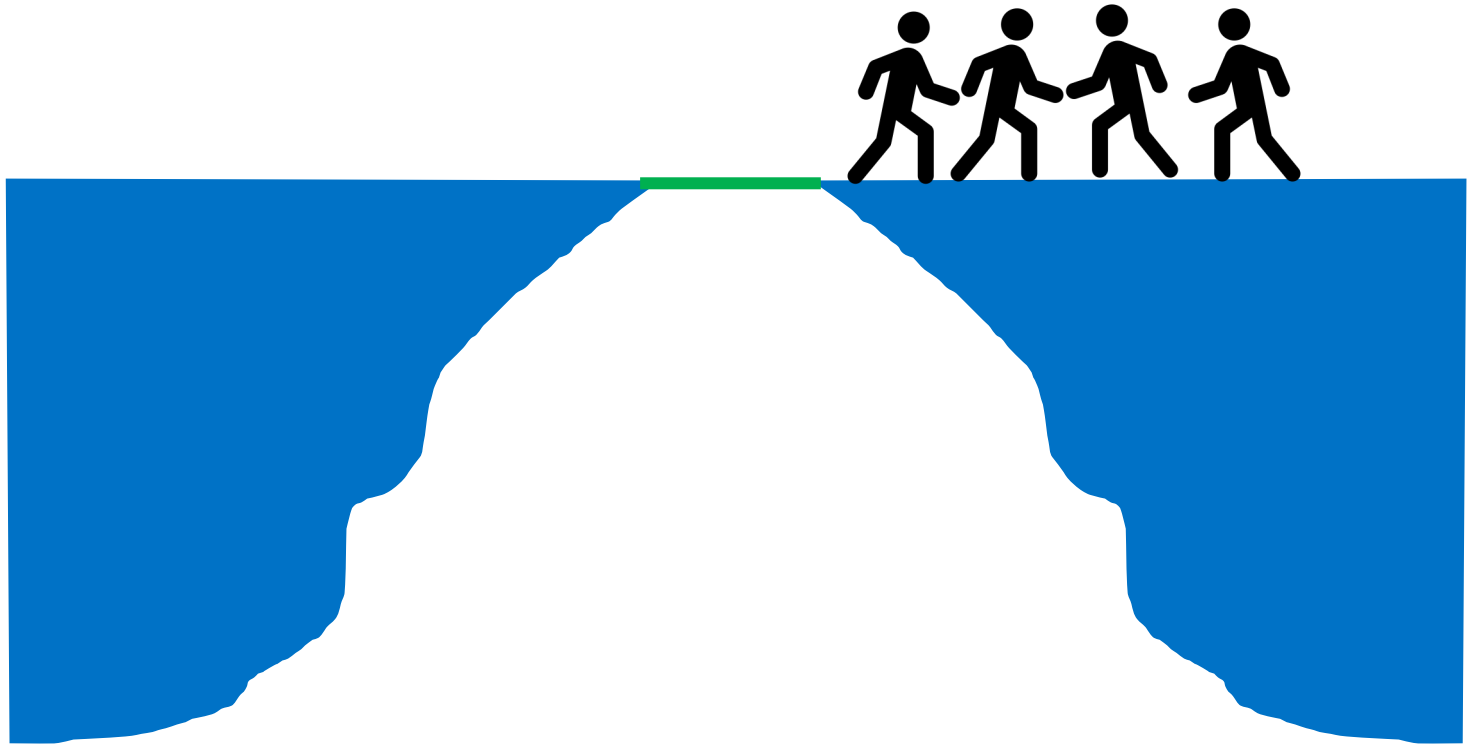


TRAILHEAD
TECHNOLOGY PARTNERS

Avoid Mistakes In The Future

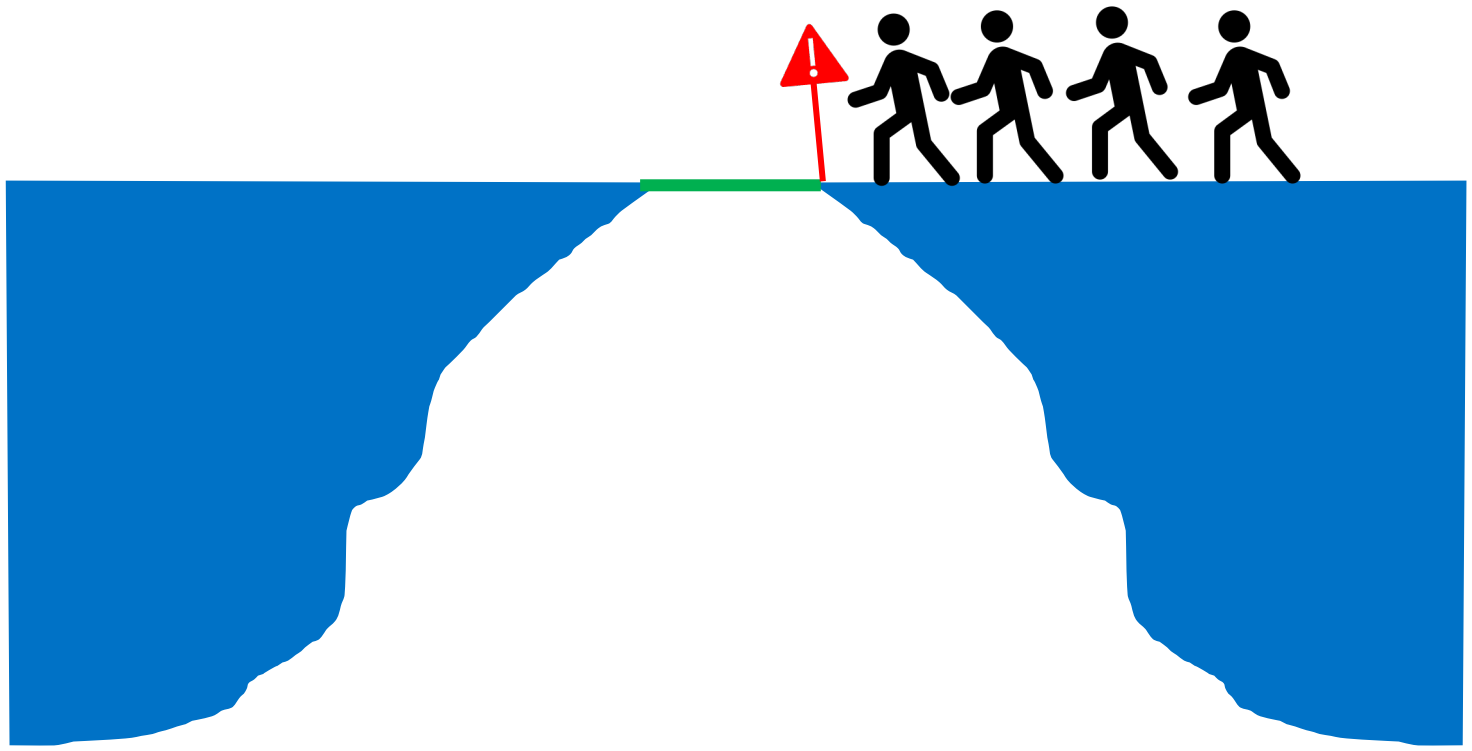
TRAILHEAD
TECHNOLOGY PARTNERS

Avoid Mistakes In The Future



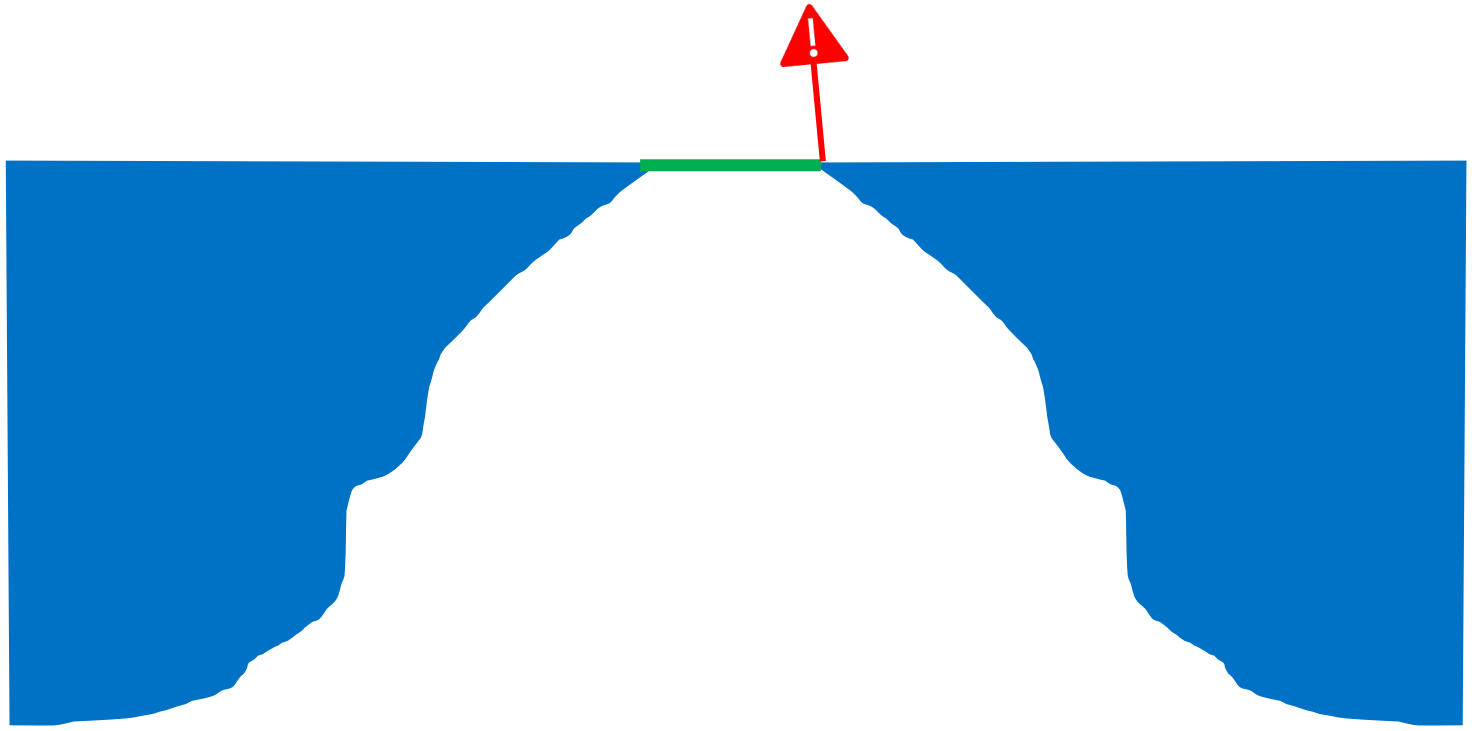
TRAILHEAD
TECHNOLOGY PARTNERS

Avoid Mistakes In The Future



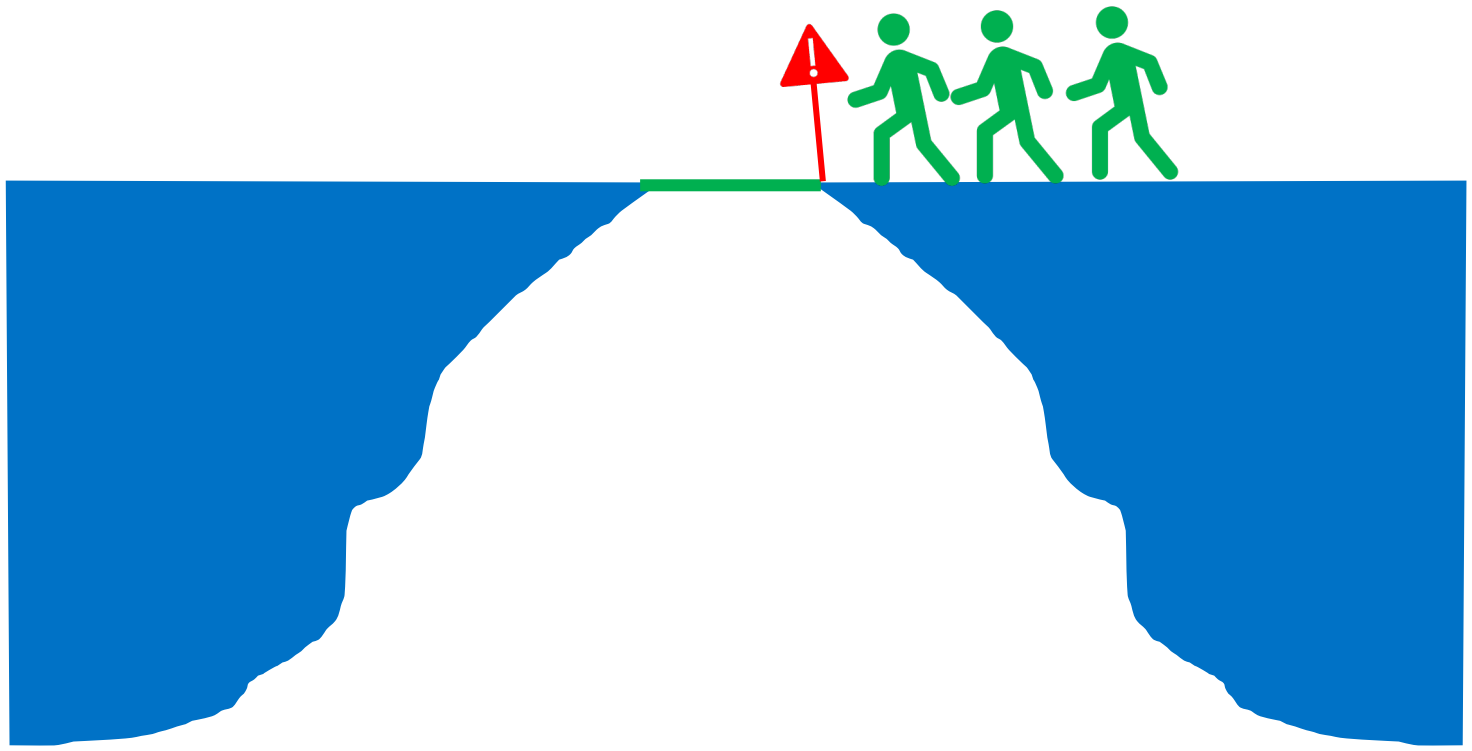
TRAILHEAD
TECHNOLOGY PARTNERS

Avoid Mistakes In The Future



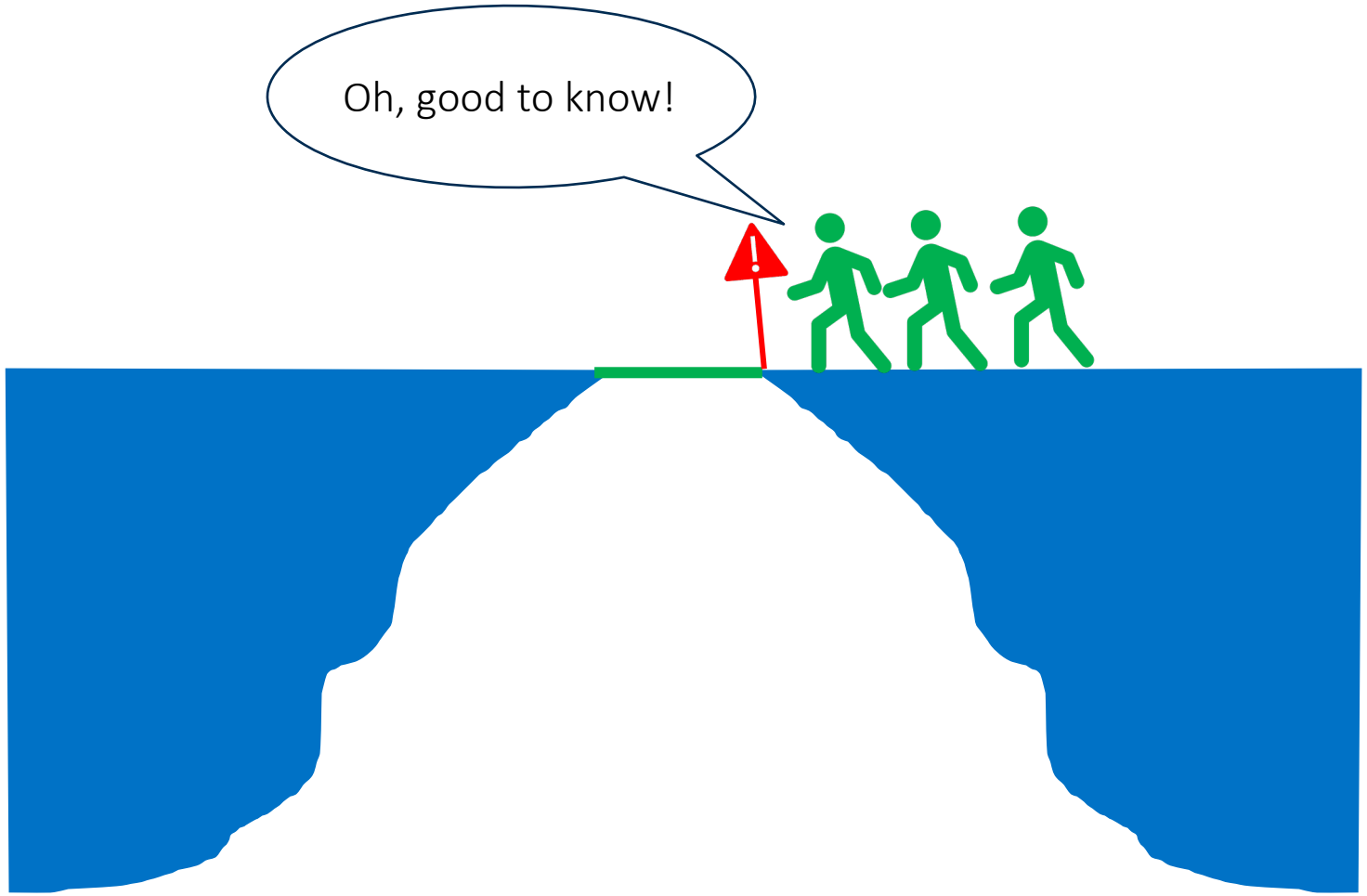
TRAILHEAD
TECHNOLOGY PARTNERS

Avoid Mistakes In The Future



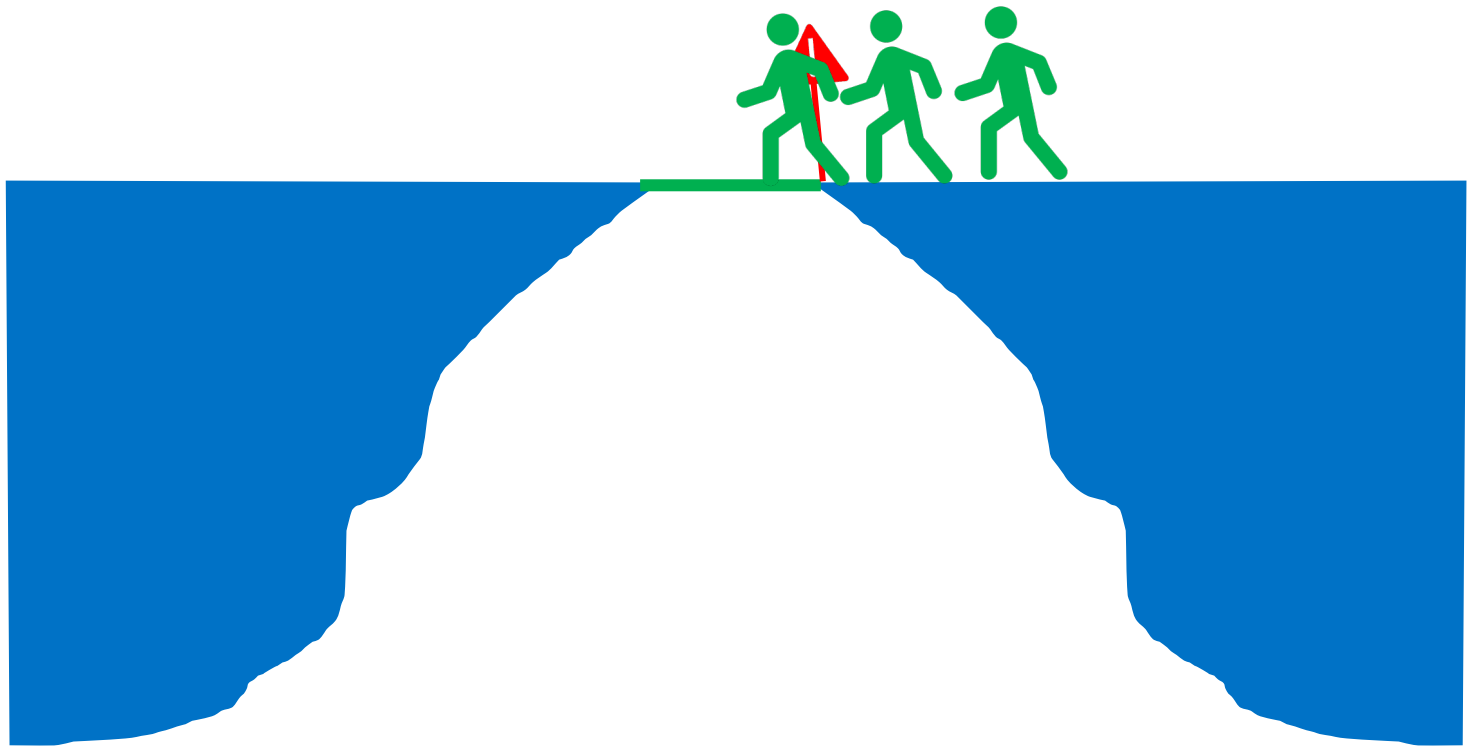
TRAILHEAD
TECHNOLOGY PARTNERS

Avoid Mistakes In The Future



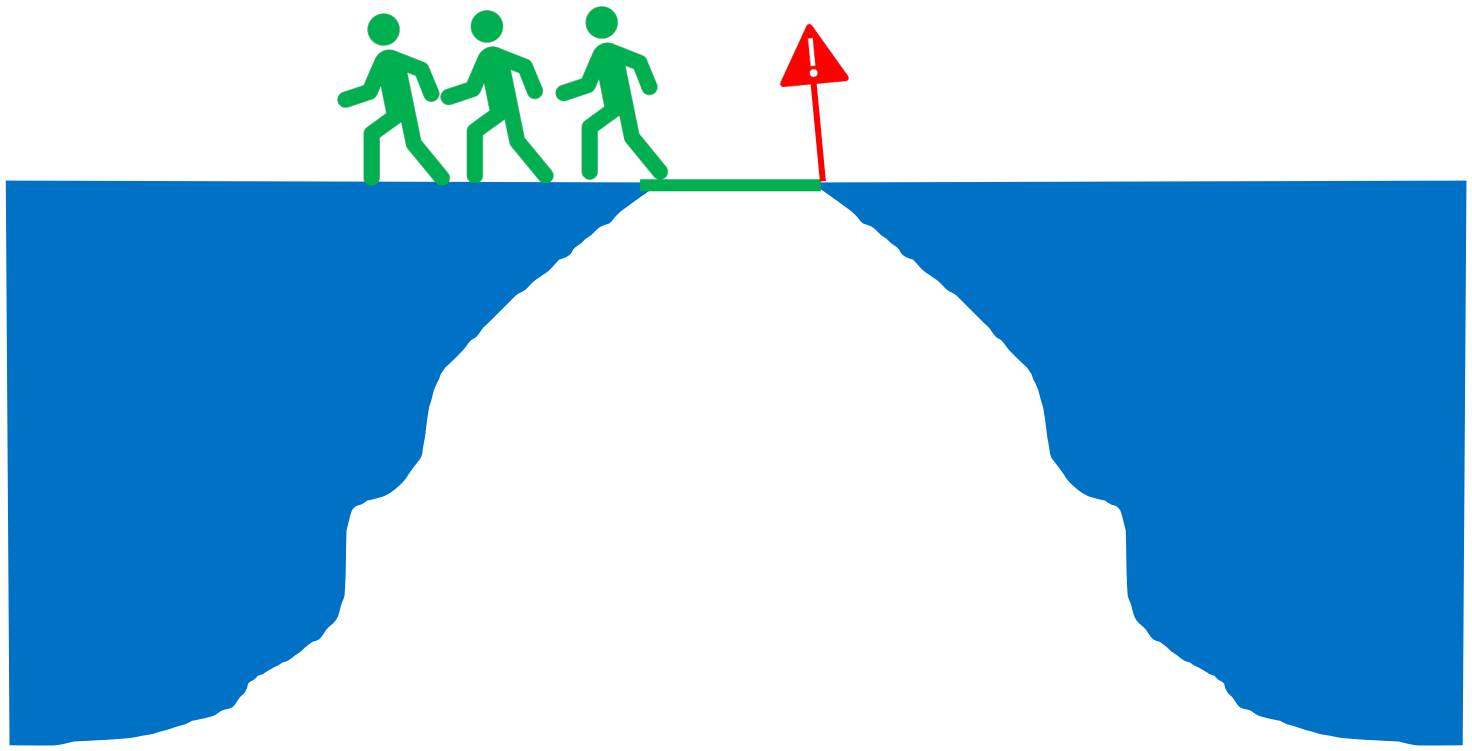
TRAILHEAD
TECHNOLOGY PARTNERS

Avoid Mistakes In The Future



TRAILHEAD
TECHNOLOGY PARTNERS

Avoid Mistakes In The Future



TRAILHEAD
TECHNOLOGY PARTNERS

Architecture Decision Records (ADR)

ADR.md

Title: Avoid Implementing Feature ‘Awaken Balrog’

Status: Accepted

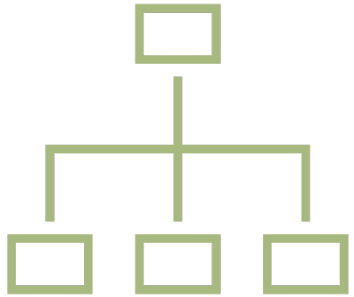
Context:
In deciding whether to implement the Balrog Awakening feature, we draw inspiration from Gandalf's advice in Moria.

Decision:
We will not implement the Balrog Awakening feature to avoid potential catastrophic issues, such as losing our only wizard.

Consequences:
Every time we’ve awaken a Balrog in the past, we’ve lost a perfectly good wizard.

TRAILHEAD
TECHNOLOGY PARTNERS

Architecture Decision Records (ADR)



Git

Wiki

TRAILHEAD
TECHNOLOGY PARTNERS

Agile / Iterative Development

Rule 2

Agile / Iterative Development

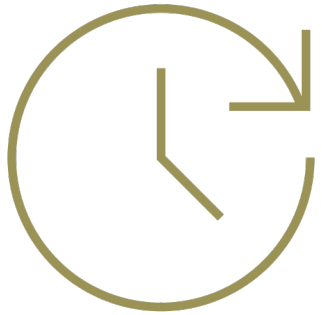
- Implement **as little as possible** to solve the problem (**MVP**)
- **Iterate** quickly with **feedback**

YAGNI
(You Aren't Gonna Need It)

Rule 3

YAGNI

Implement Only
Current Requirements



Don’t Build For
Future Needs

TRAILHEAD
TECHNOLOGY PARTNERS

KISS (Keep It Simple, Software-developer)

Rule 4

KISS

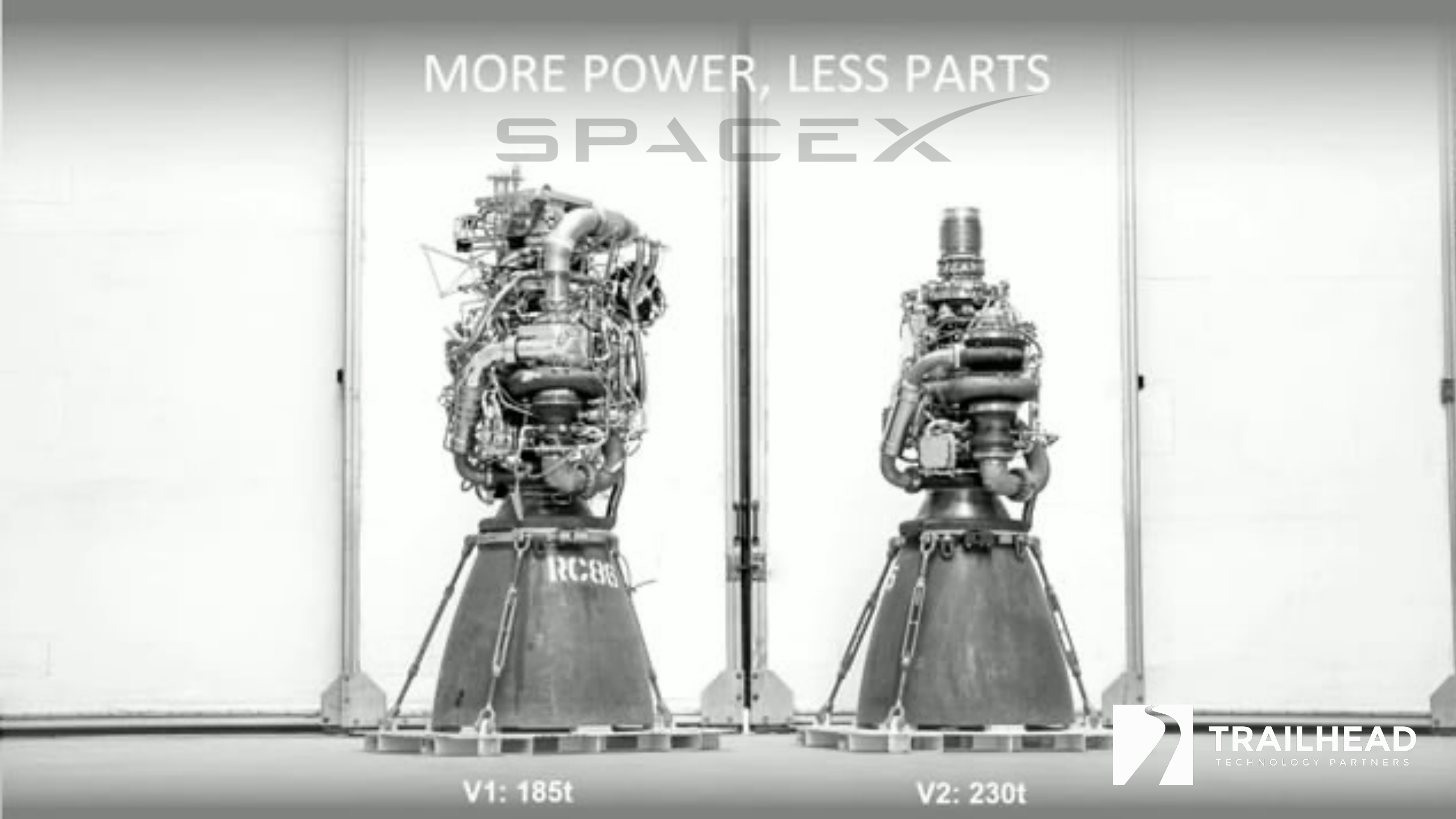
Less Surface Area
For Failures

Easier to
Maintain

TRAILHEAD
TECHNOLOGY PARTNERS

MORE POWER, LESS PARTS

SPACEX



V1: 185t

V2: 230t

TRAILHEAD
TECHNOLOGY PARTNERS

DRY (Don't Repeat Yourself)

Rule 5

DRY

- Reduce the amount of duplicate code written.
- This can save time and energy,
- Helps developers focus on more important problems.
- Can also make software more maintainable, readable, and consistent

Principle of Least Astonishment (POLA)

Rule 6

Principle of Least Astonishment

"A **component** of a system **should behave** in a way that **most users will expect** it to behave, and therefore **not astonish** or surprise users."

Principle of Least Astonishment (POLA)

Reduces
Cognitive Load

Reduces
Mistakes

Easier to
Learn

TRAILHEAD
TECHNOLOGY PARTNERS

Pull Reqs / Code Reviews

Rule 7

TRAILHEAD
TECHNOLOGY PARTNERS

Pull Reqs / Code Reviews

Detecting Patterns of Over-Engineering

Catching Unnecessary Features Early

Ensuring Adherence to Best Practices

Encouraging Incremental Development

TRAILHEAD
TECHNOLOGY PARTNERS

“Shift Left”

The **earlier** in the development lifecycle you **catch a defect**, the **less expensive** it is to fix.

Typical Code Review Checklist

- Bug Check
- Acceptance Criteria Check
- Code Standards Check
- Clarity Check
- Performance Check
- Documentation Check

Typical Code Review Checklist

- Bug Check
- Acceptance Criteria Check
- Code Standards Check
- Clarity Check
- Performance Check
- Documentation Check

- ✓ Simplicity Check
- ✓ YAGNI Check
- ✓ KISS Check
- ✓ Premature Optimization Check

Feedback Loops

Rule 8

Feedback Loops

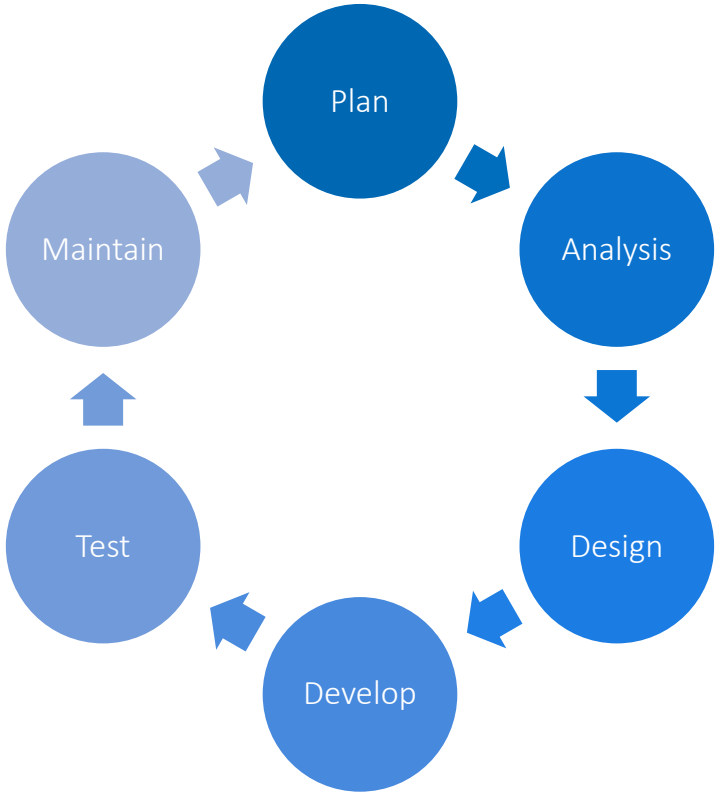
Aligning Development with User Needs

Encouraging Iterative Development

Detecting and correcting over-engineering early

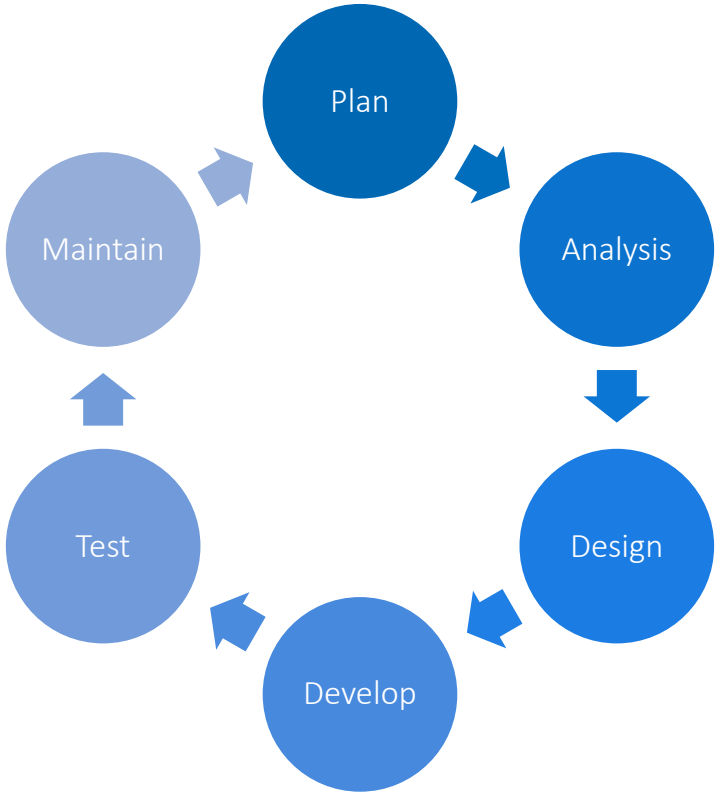
TRAILHEAD
TECHNOLOGY PARTNERS

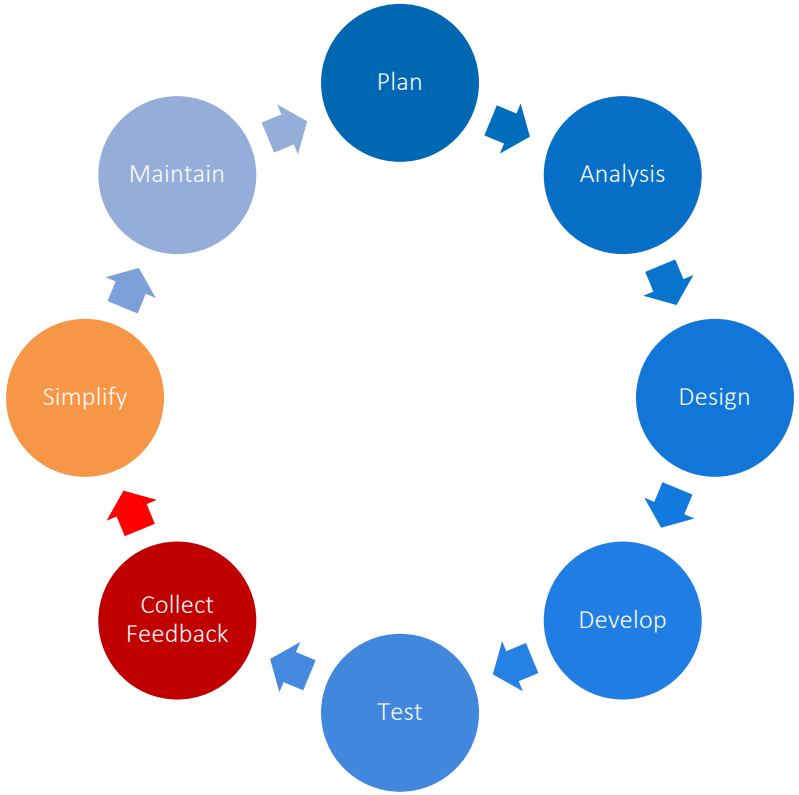
Feedback Loops



TRAILHEAD
TECHNOLOGY PARTNERS

Feedback Loops





TRAILHEAD
TECHNOLOGY PARTNERS

Adopt Proven Solutions

Rule 9

Adopt Proven Solutions

Adapting Proven Solutions

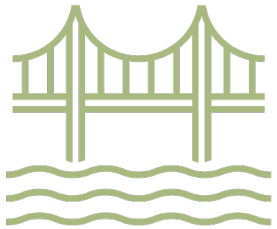
More predictable outcomes

Reduces unknowns

Manages risk

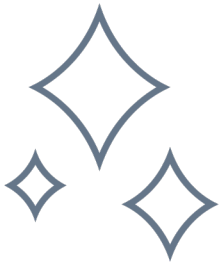
TRAILHEAD
TECHNOLOGY PARTNERS

Adapting Proven Solutions



Use Frameworks
You Know

New Architectures
Only as POC



Adopt New Tech Behind
Buzzword Curve

TRAILHEAD
TECHNOLOGY PARTNERS

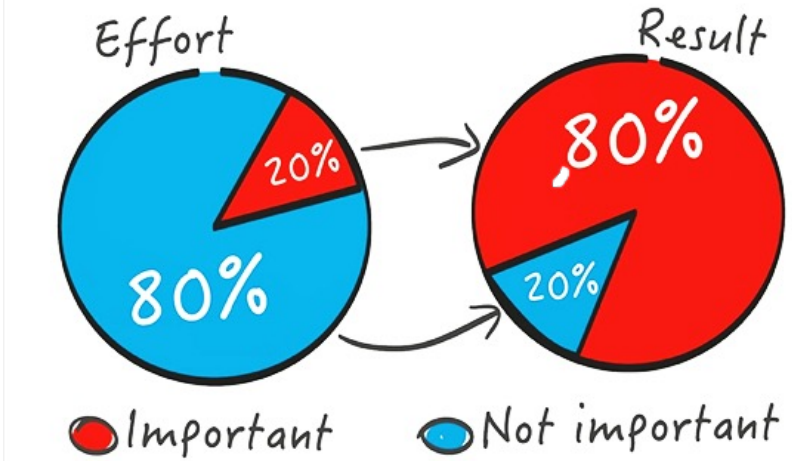
80/20 Rule (Pareto Principle)

Rule 10

TRAILHEAD
TECHNOLOGY PARTNERS

80/20 Rule

"Roughly 80% of consequences come from 20% of causes."

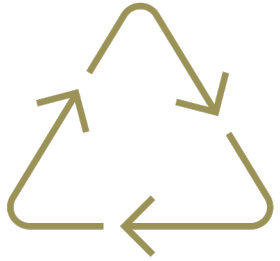


SOURCE https://www.sreedeep.com/the-pareto-principle-explained/

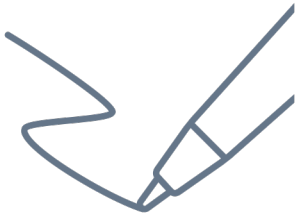
TRAILHEAD
TECHNOLOGY PARTNERS

80/20 Rule Allows

Focusing on
high-impact features



Efficient
use of resources



Avoid
perfectionism

TRAILHEAD
TECHNOLOGY PARTNERS

Summing Up

1. Over-engineering is **common** and **costly**
2. **Know the signs** of over-engineering
3. Implement the **rules to avoid** over-engineering

TRAILHEAD
TECHNOLOGY PARTNERS

Thanks! Questions?

Jonathan "J." Tower

- Microsoft MVP in .NET
- jtower@trailheadtechnology.com
- trailheadtechnology.com/blog
- jtowermi
- jtower

FREE

CONSULTATION

bit.ly/th-offer

github.com/trailheadtechnology/over-engineering