

Hello UCD!

Nice to meet you





Roy Berris

Software Engineer at iO

Working with

- Umbraco
- MACH
- .NET



Can you *work* agile
When your technology *isn't*?

Technology should meet demand



umbraco



Clients want to act now.

We claim we're agile, but is your technology agile enough?



*Building a Strong Foundation for Your
Modern .NET Project*



Breaking it down

1. Architecture
2. Design
3. Code

A Strong Foundation

Defining a foundation



What makes a foundation strong?

- Stability?
- Scalability?
- Ease of maintenance?

You *define* the foundation



The foundation needs to suit your solution.

- Functional requirements
- Non-functional requirements

Functional requirements



Can be defined in user stories

As a logged-in user I want to place an item in my cart so I can order it later

As an employee I want to update orders so I avoid orders being sent to the wrong address

Non-functional requirements



Define the operation rather than specific behaviors.

- Number of requests per second
- Up-time % of a module
- Downtime after deployment

Metrics



A non-functional requirements is within a system of measurement.

- Performance
- Security
- Availability
- Usability
- Scalability
- Reliability
- Maintainability

Future proof Architecture



- Freedom of choice
- Flexibility
- Adoptability

Future proof Architecture



Choose best of class

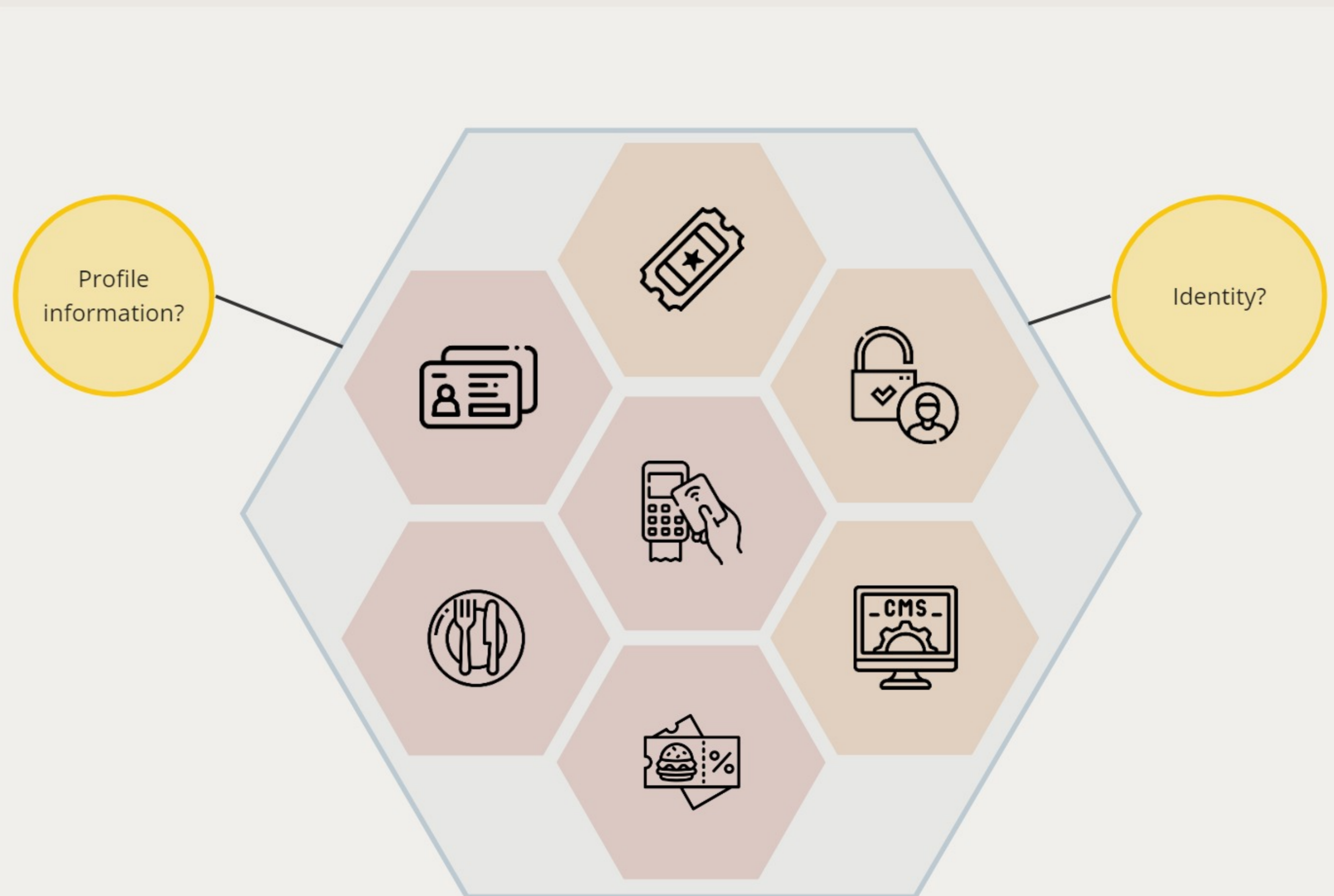


Choose best of class over best of breed.

Best of breed is the very best product in the eco-system.

Best of class is choosing the right fit for your class, instead of buying what is the very best overall.

Why not best of breed



Best of breed refers to 'enterprise' software. Typically closed development and poor integrability.

Best of class are focusing on a single category and on the core of the needs.

Best of class focusses on reusability.

Compose to the business needs

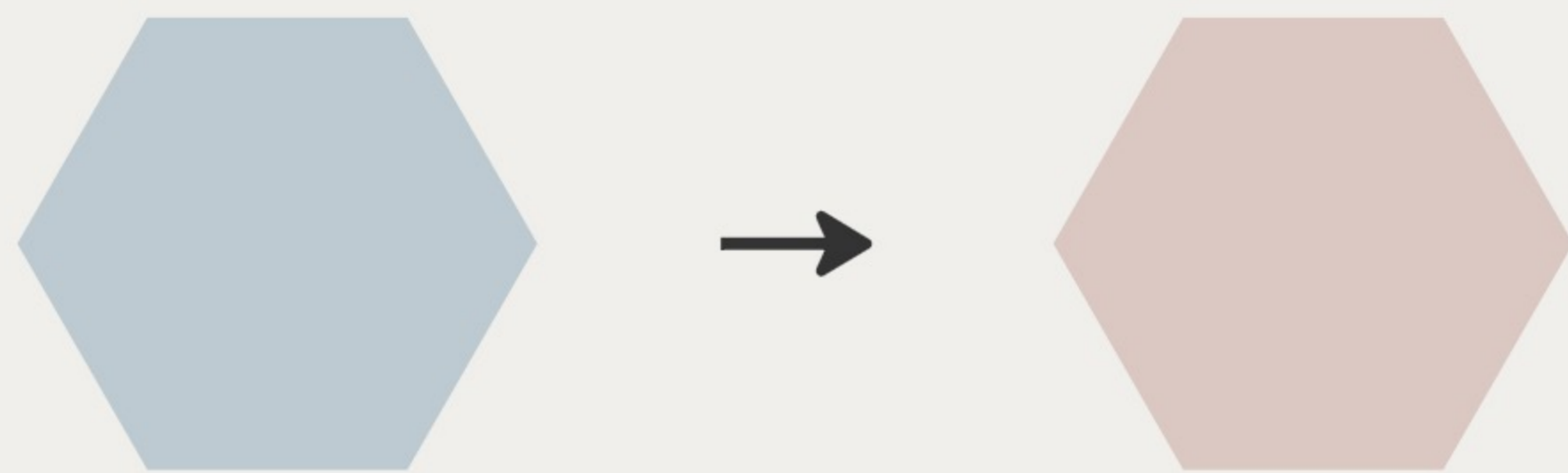


Start small and *'try before you buy'*.

Evaluate based on your business needs.

Don't be afraid to replace components.

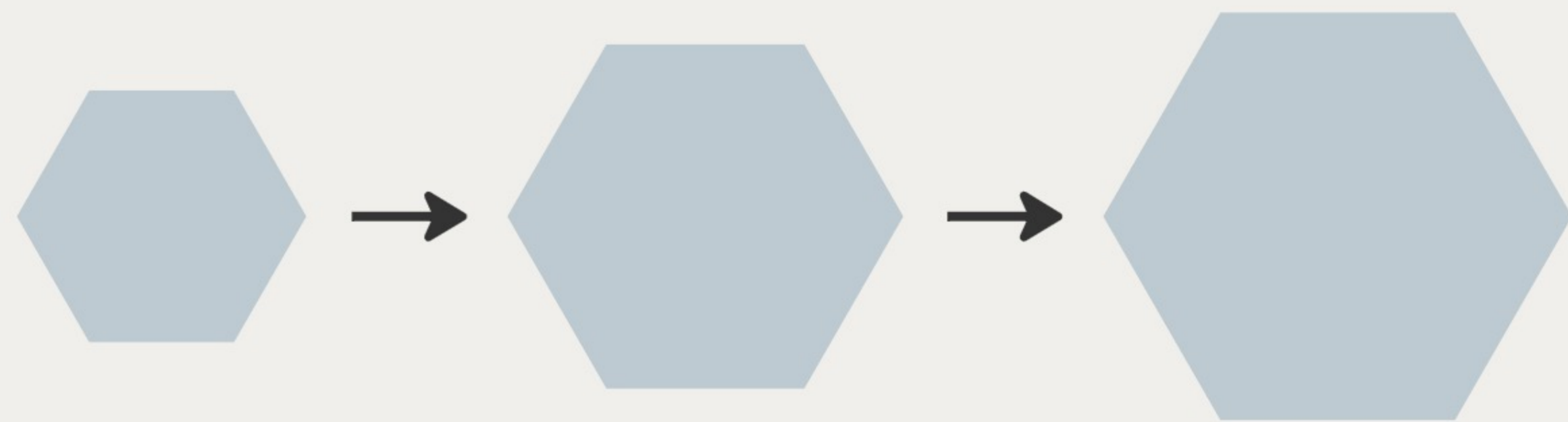
Flexibility



Flexibility stimulates innovation.

Replace components when the business demands it.

Adoptability



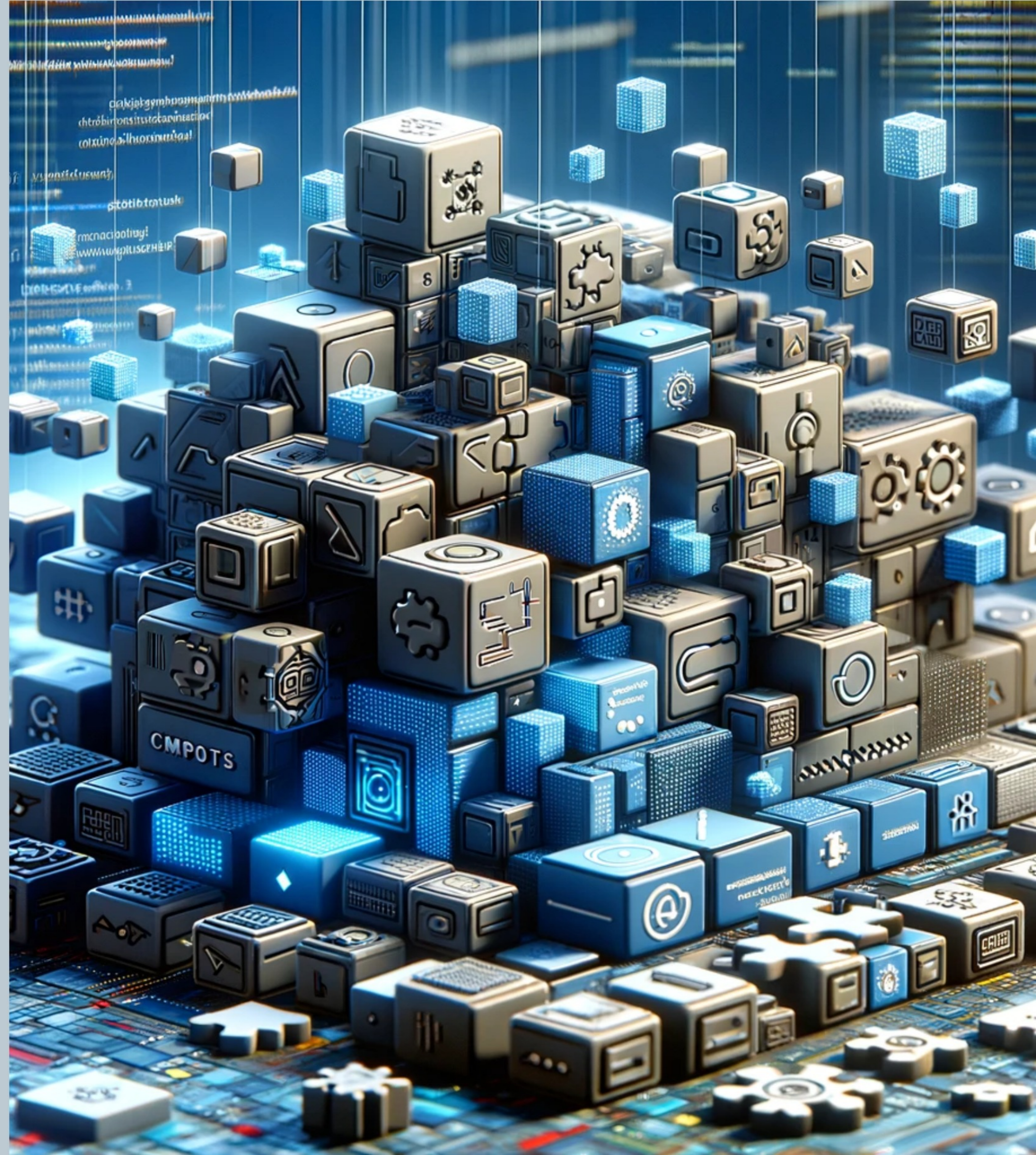
Adopt new technologies.

Upgrading components creates innovation.

Future proof architecture is

- composing to your needs.
- keeping options open
- a strong foundation

*A composable
solution*



Composing

Composable



Composable is decoupling the application.

Separate modules in to their own.

Keeping dependencies loosely coupled.

Composing our own

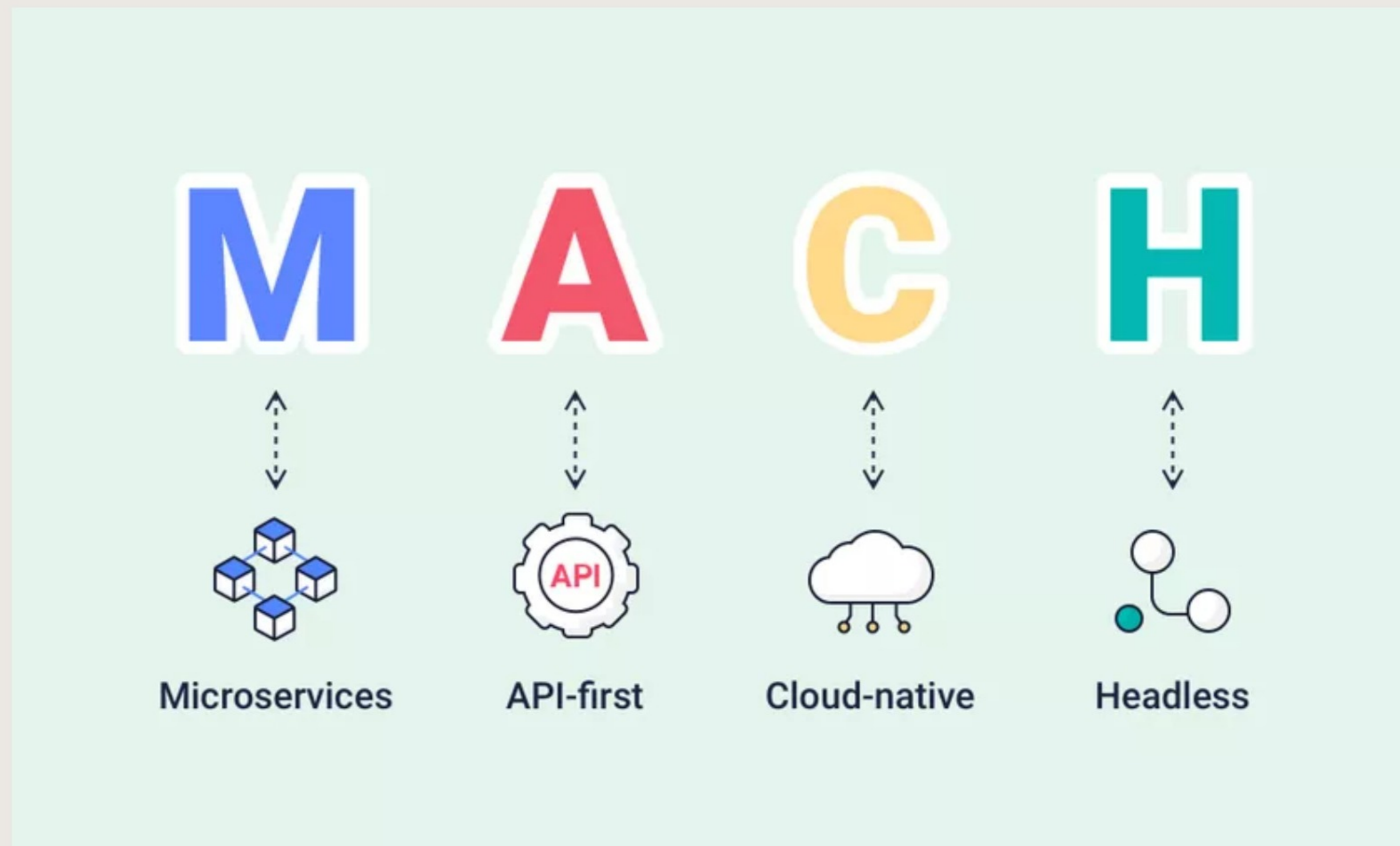


Composable doesn't have to be all SaaS.

We can create our own components.

Write middleware to communicate between components.

The MACH way



It's an acronym for

- Microservices
- API-first
- Cloud-native
- Headless

Microservices

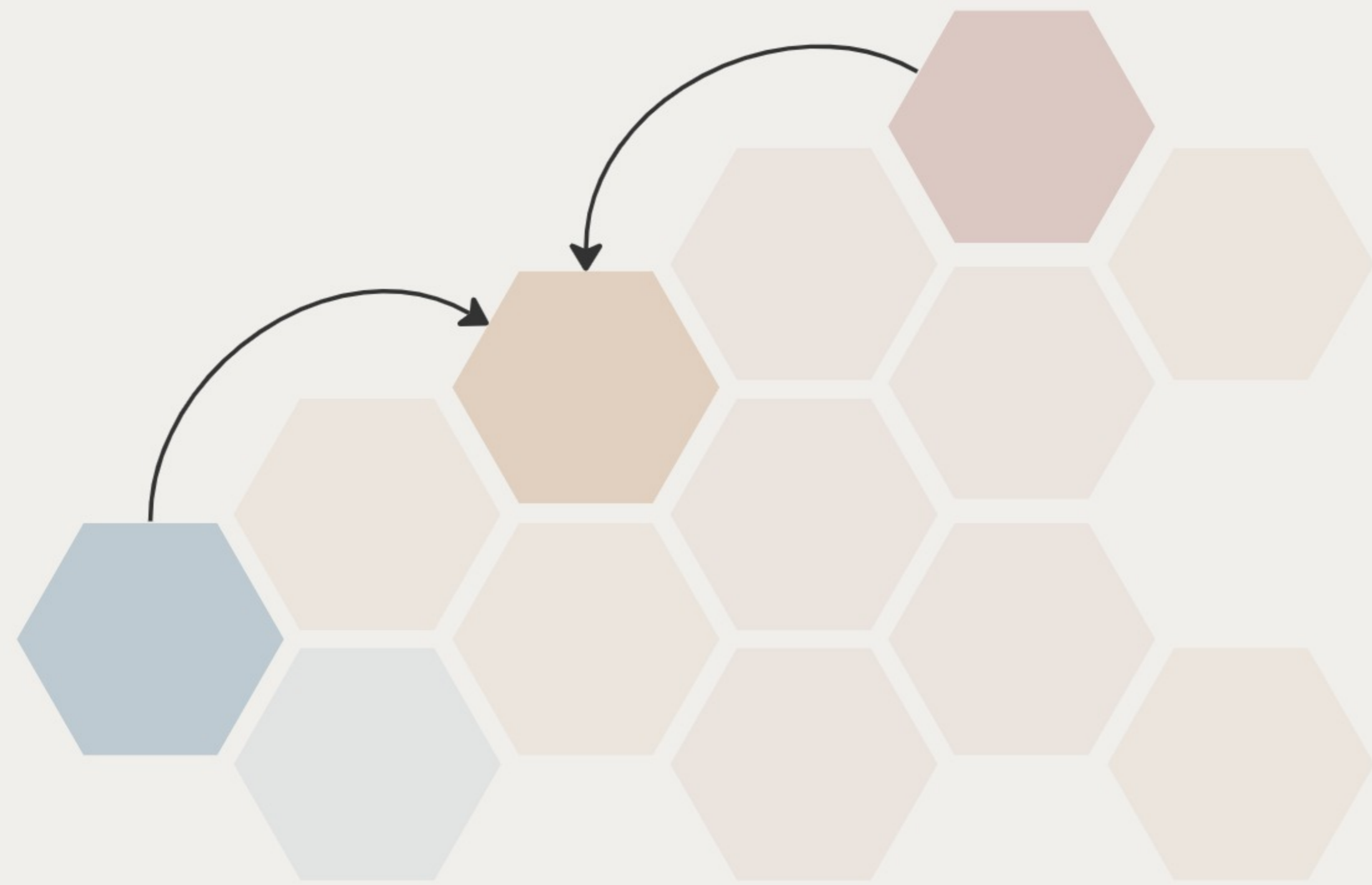


Component should have a single responsibility.

Flexibility in development.

Innovate on microservices without exposing the whole system.

API-first

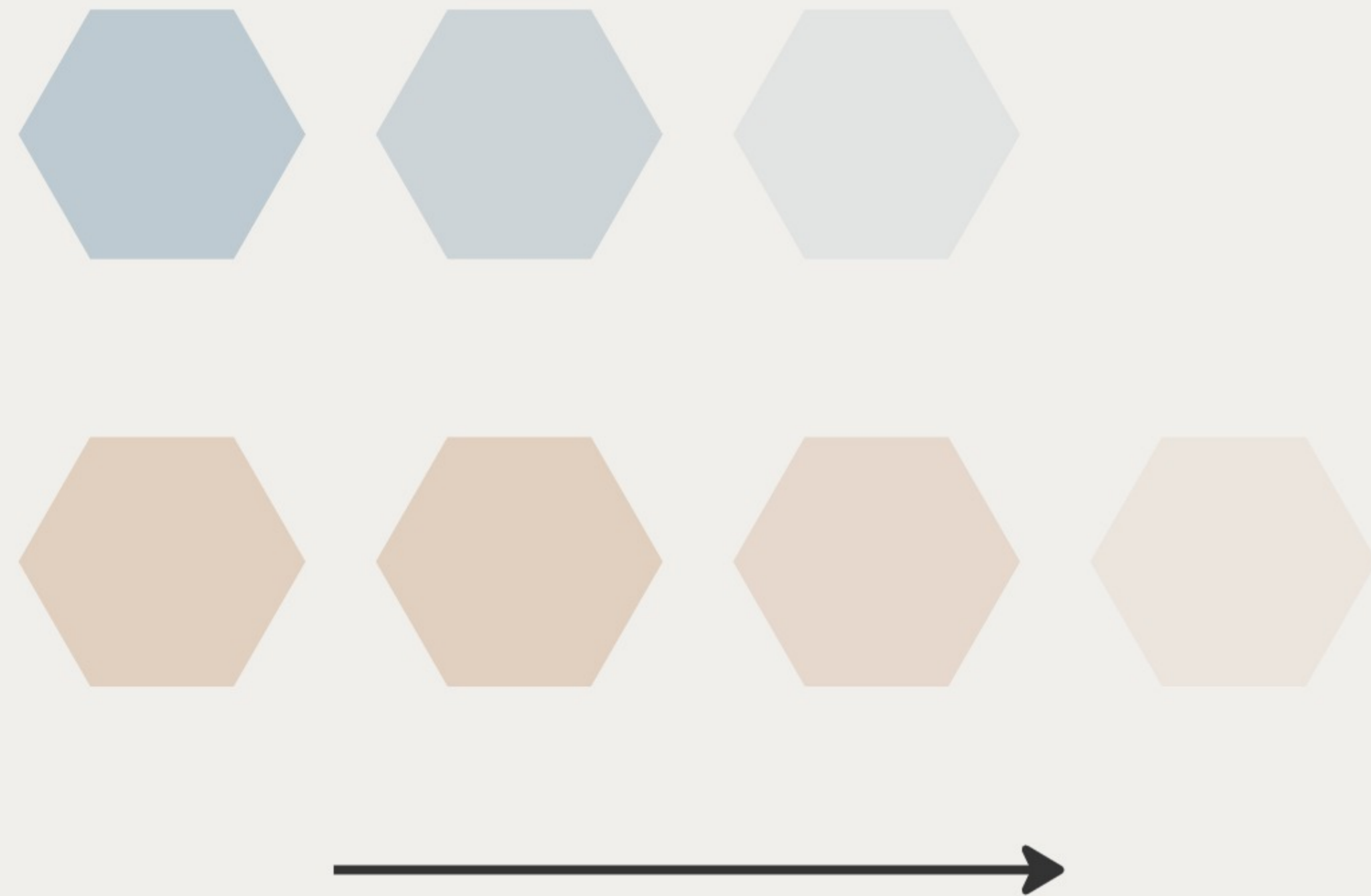


An API specification is the contract between modules.

Change the modules without changing the contract.

A contract allows for adoptability of new technology.

Cloud native

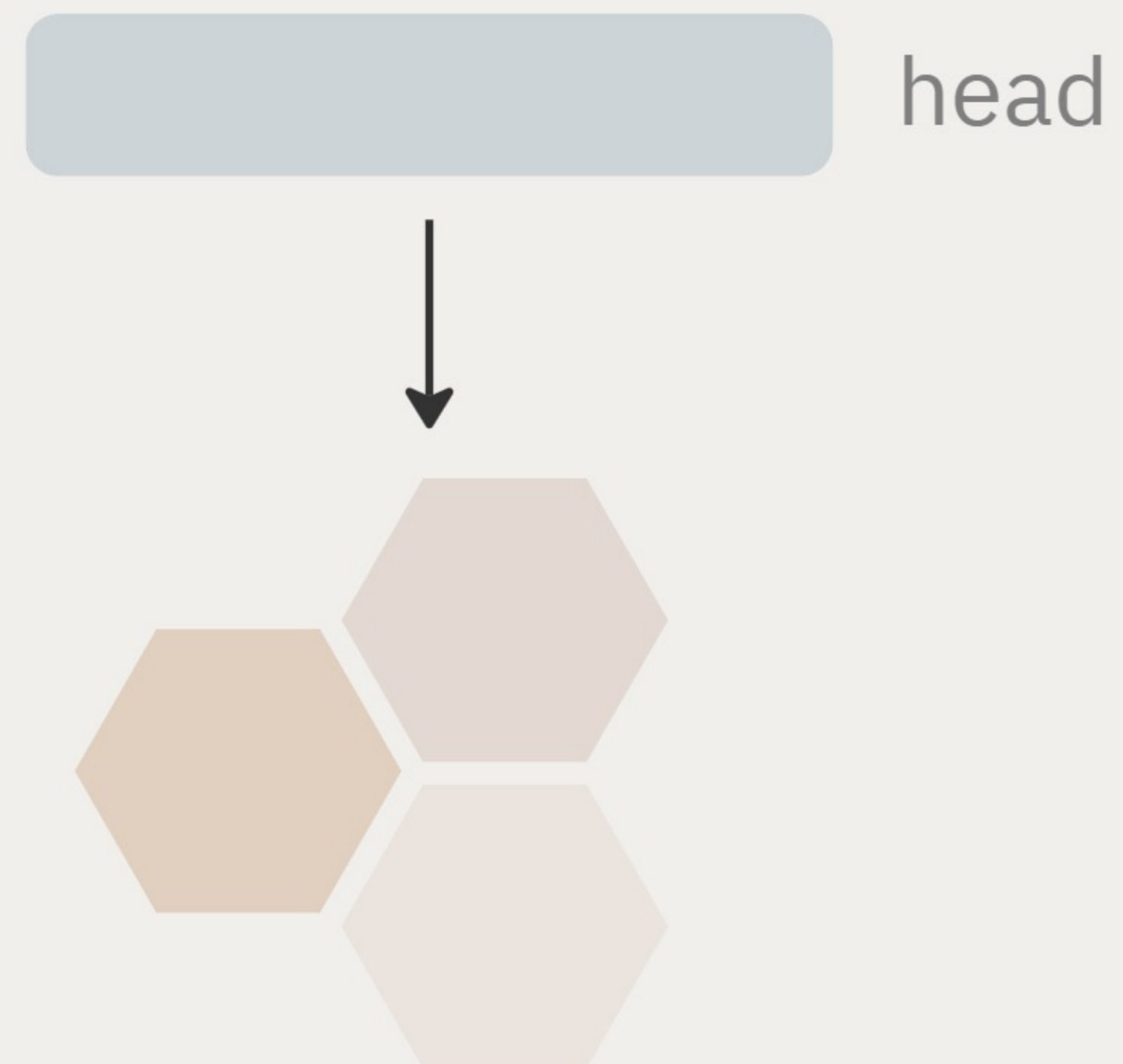


Building and designing for the cloud

Scale your application when needed.

Rapidly recover from system failures.

Headless



Decoupling presentation from the logic allows us to be

- Omni-channel
- Technology agnostic
- Optimized

Working MACH is

- flexible
- scalable
- resilient

*Responsive to
changing markets*



Software Design



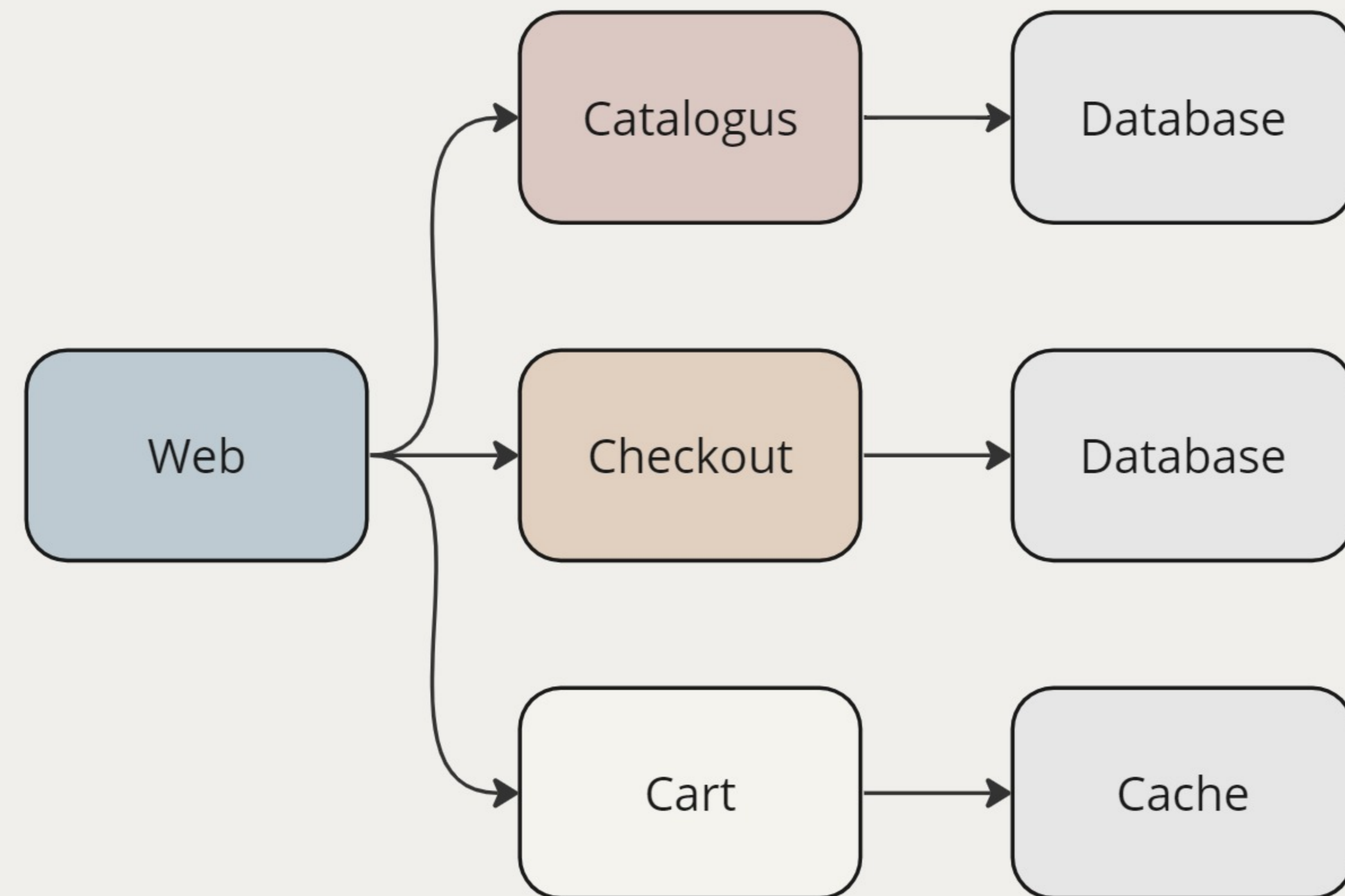
Software architecture vs design



Architecture is the concern of the whole.

Design is the concern of the individual component.

Architecture

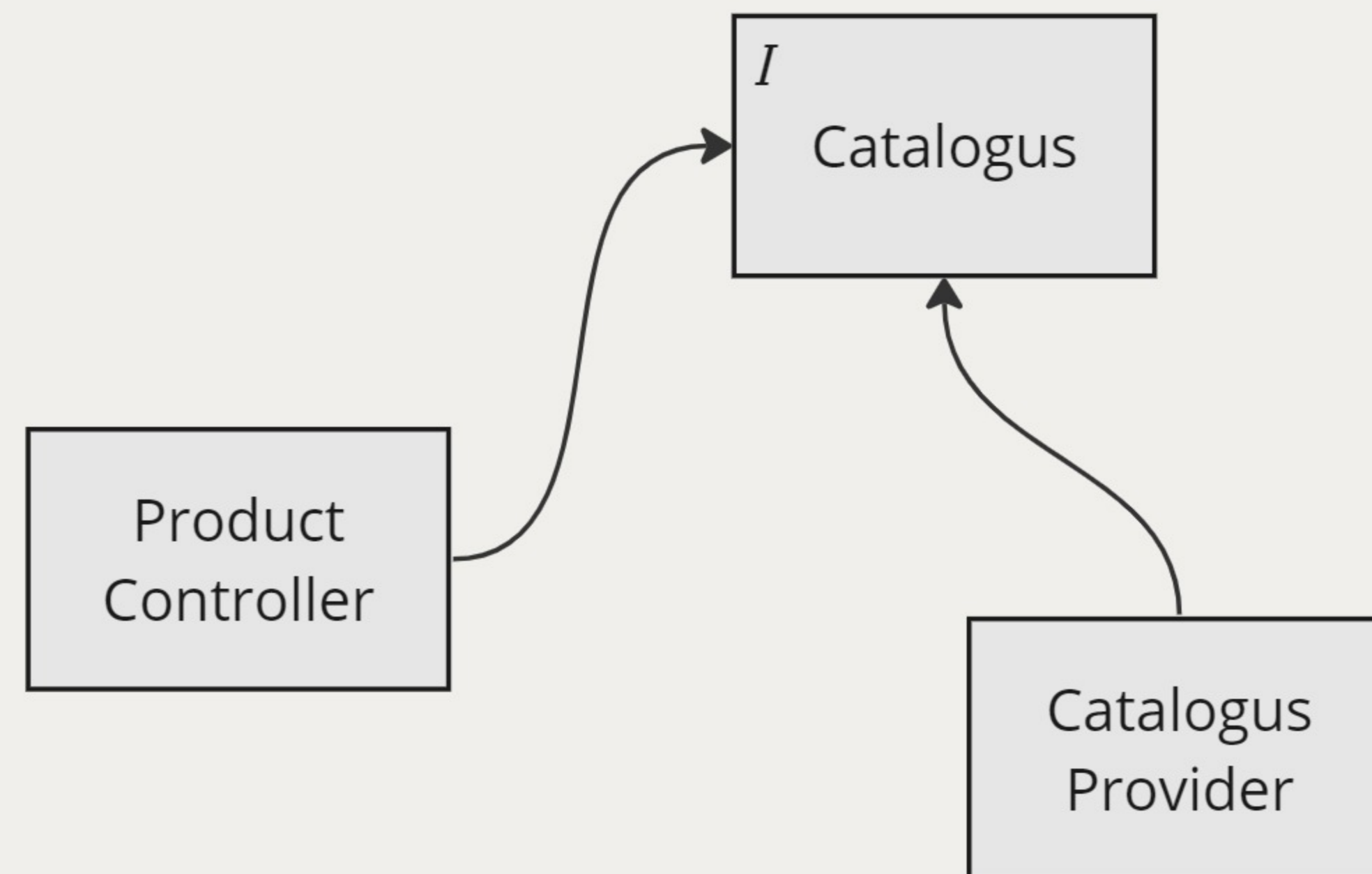


A high level abstract overview of the system.

Contains the externally visible properties of the components.

Design

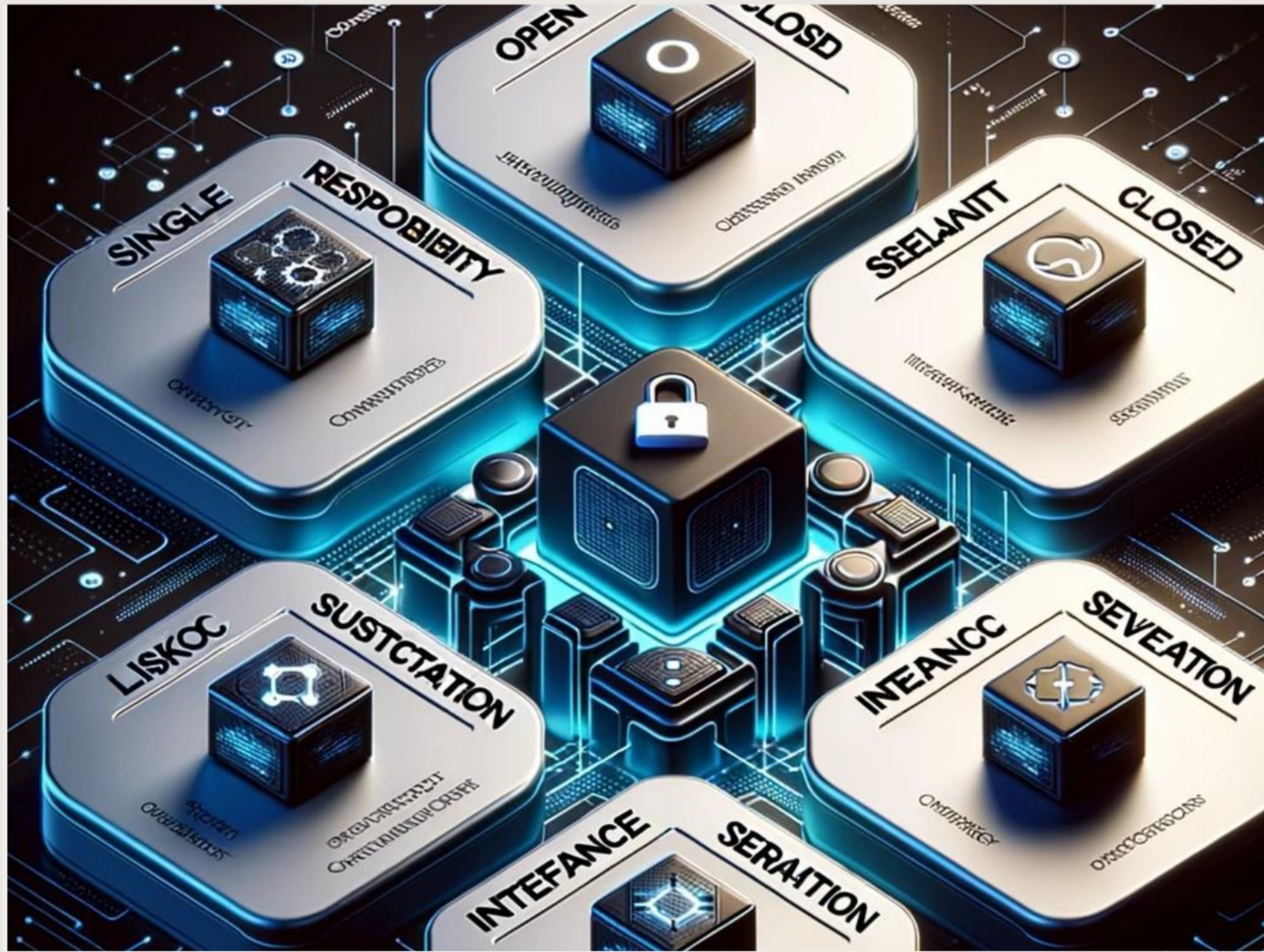
Catalogus



A detailed specification of a software component.

It's a design plan to implement the component.

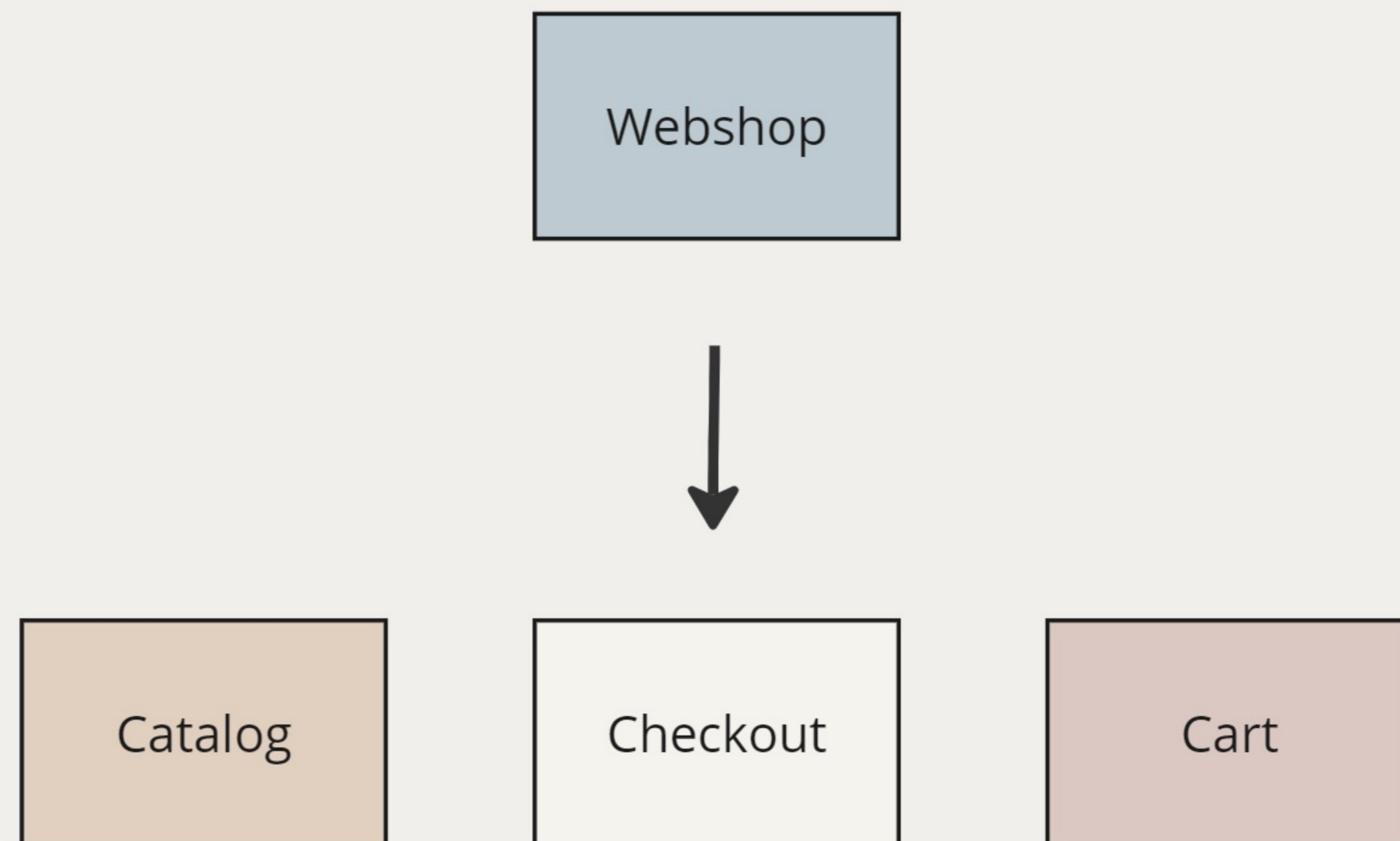
Design principles



Designing software requires principles.

Let's look at **SOLID**.

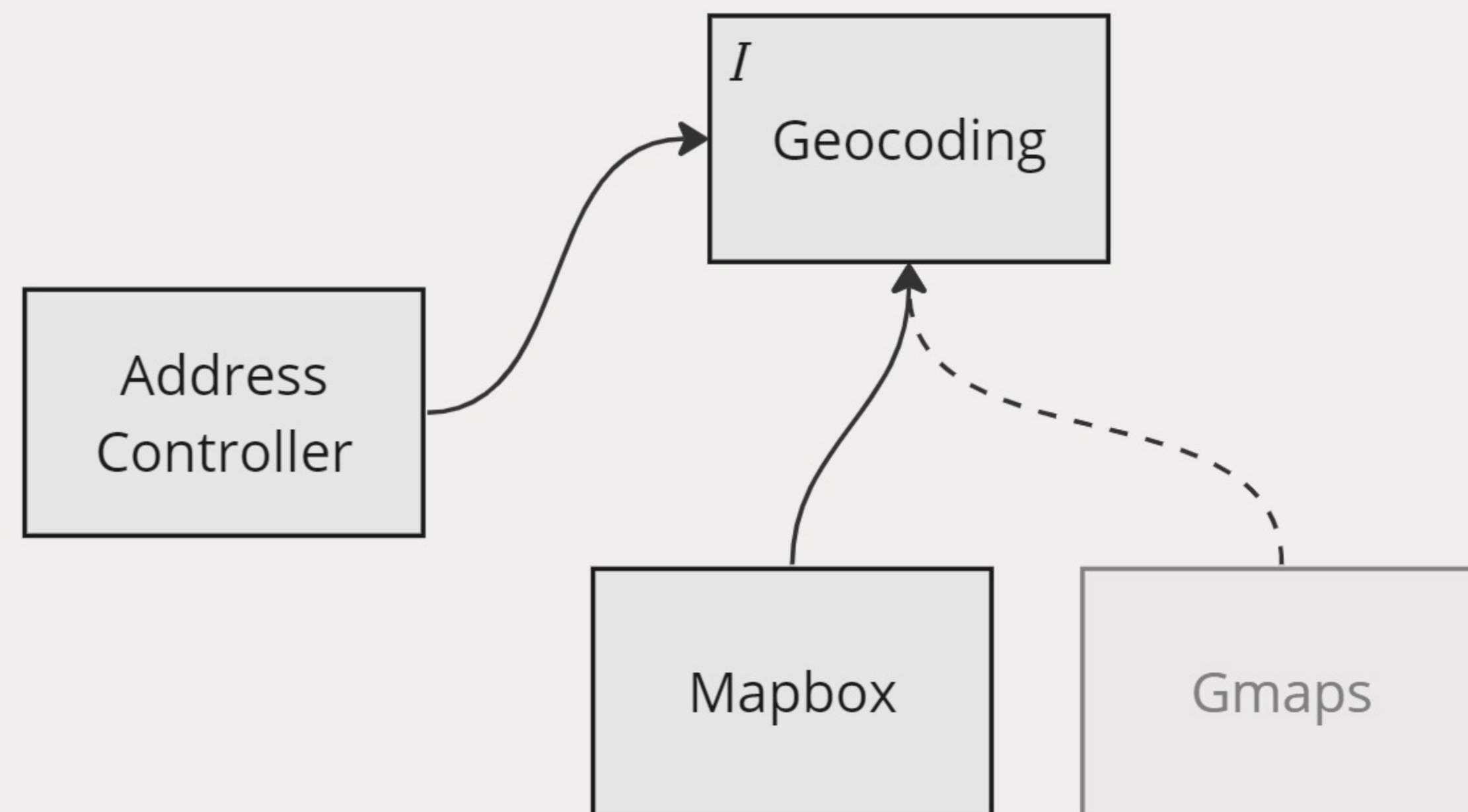
Single Responsibility Principle



A module should have only one reason to change.

It should do things in the scope of a single subject.

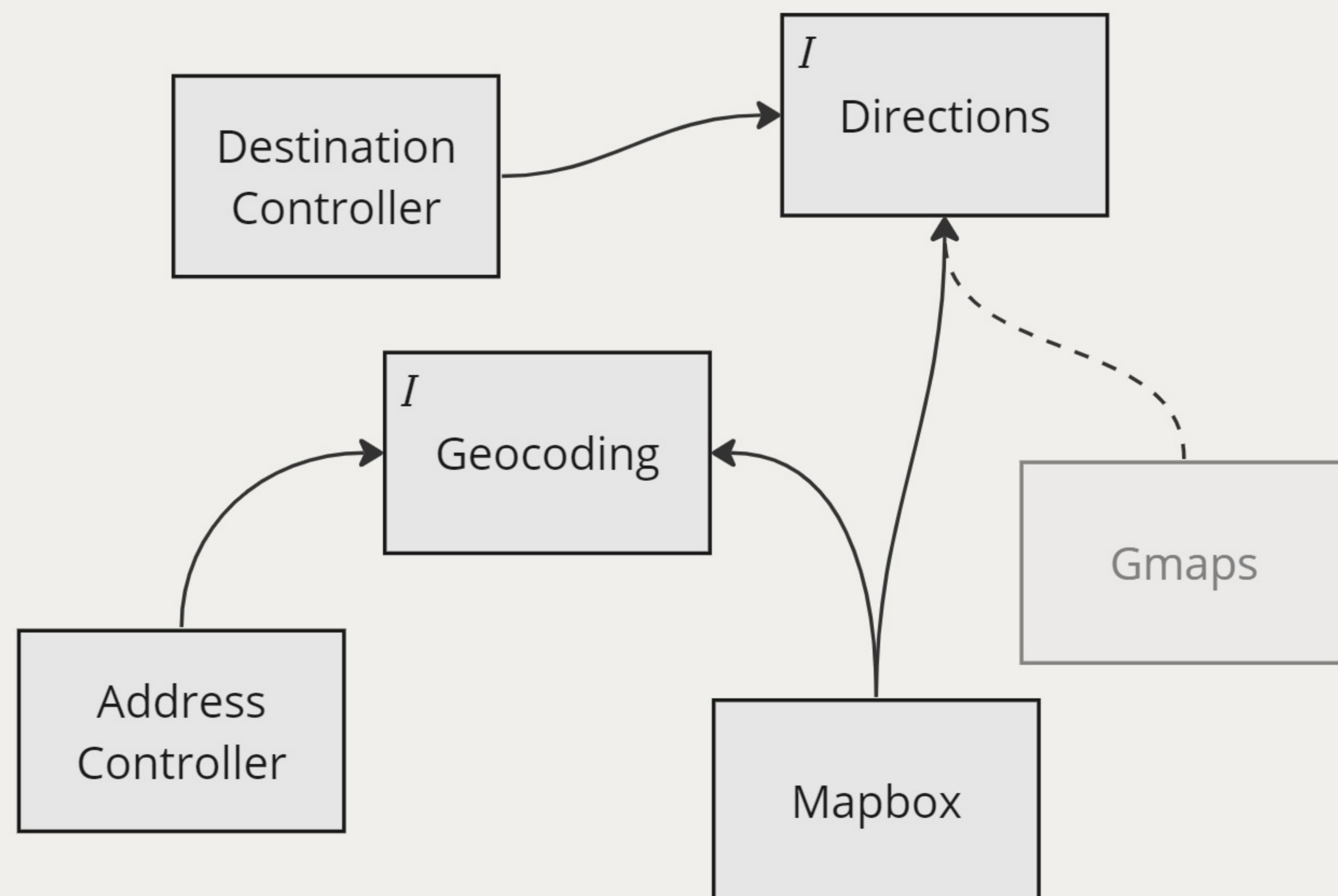
Dependency Inversion Principle



High-level modules should not depend on low-level modules.

It will decouple the low-level implementation from the high-level components.

Interface Segregation Principle



Clients should not be forced to depend on interfaces they do not use.

If the SRP and DIP had a child.

Keeping options open



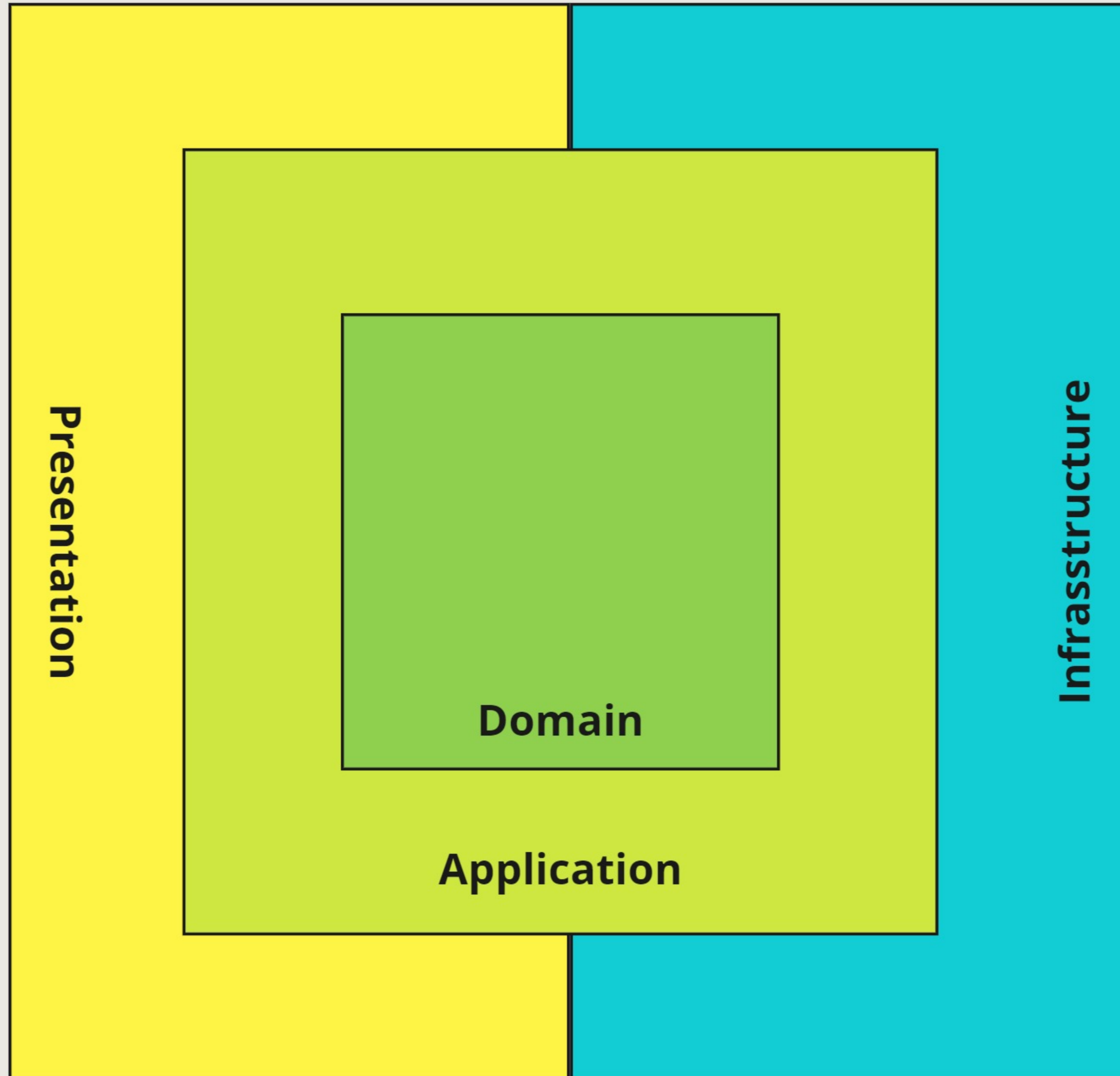
Separating the system into modules with a single responsibility.

Isolating modules through interfaces.

We open the pathways for the future.

Make it
CLEAN

CLEAN architecture

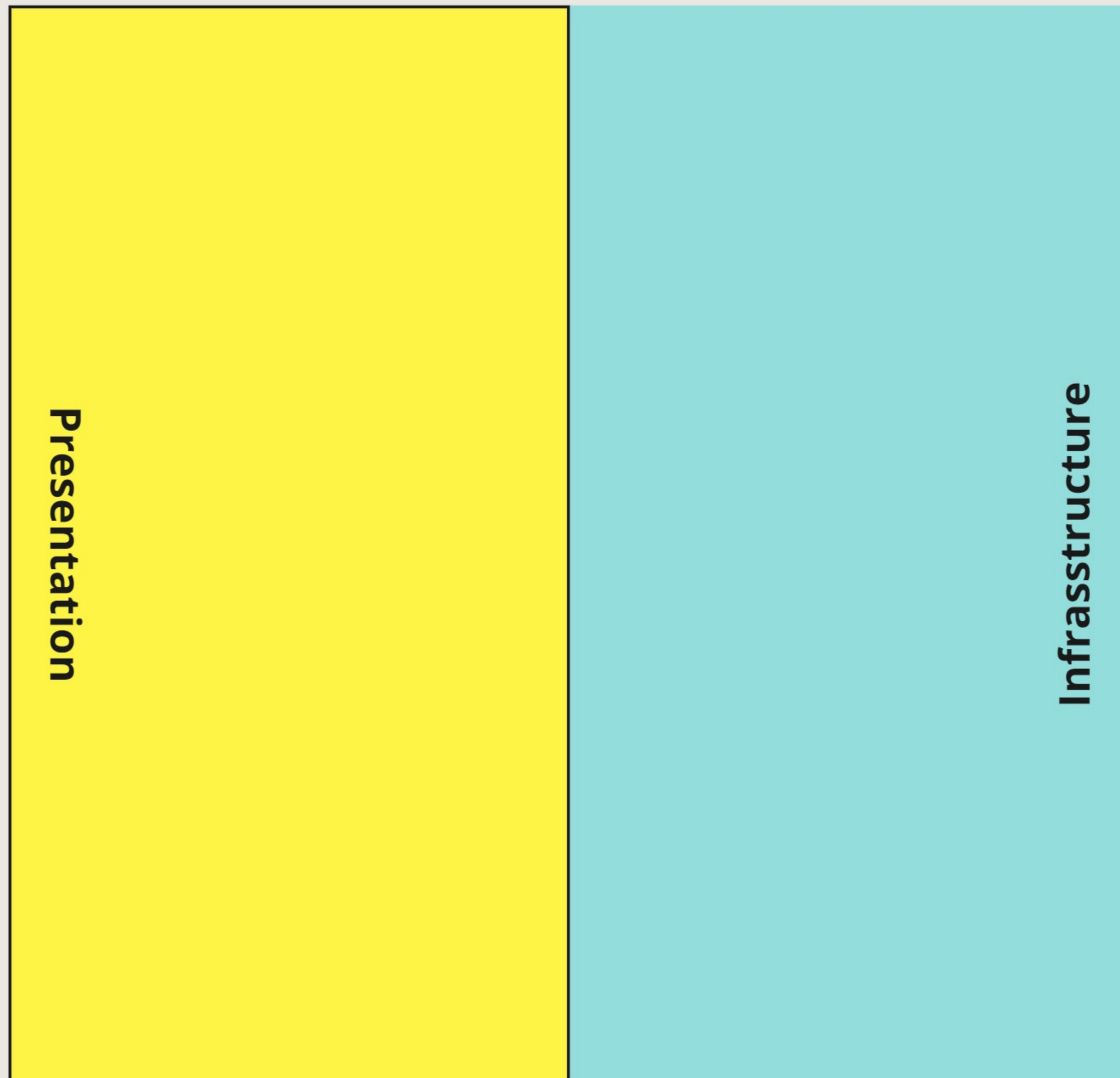


Clean architecture is a design philosophy.

Splitting the system in to different layers.

Each layer having it's own responsibilities.

The presentation

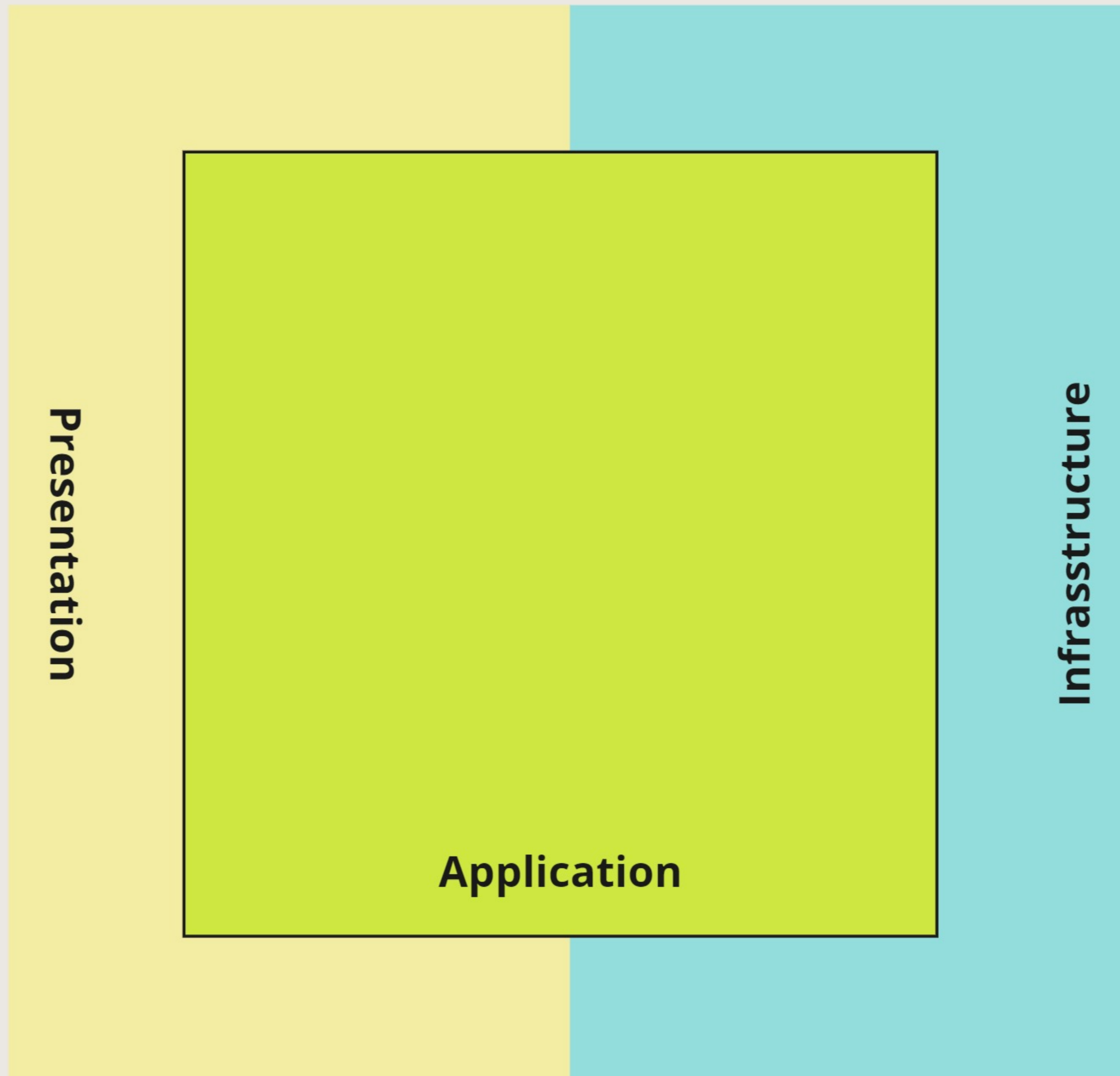


The presentation layer is on the outer ring.

It defines the specifications with the outside world.

Supports an API-first approach.

The application

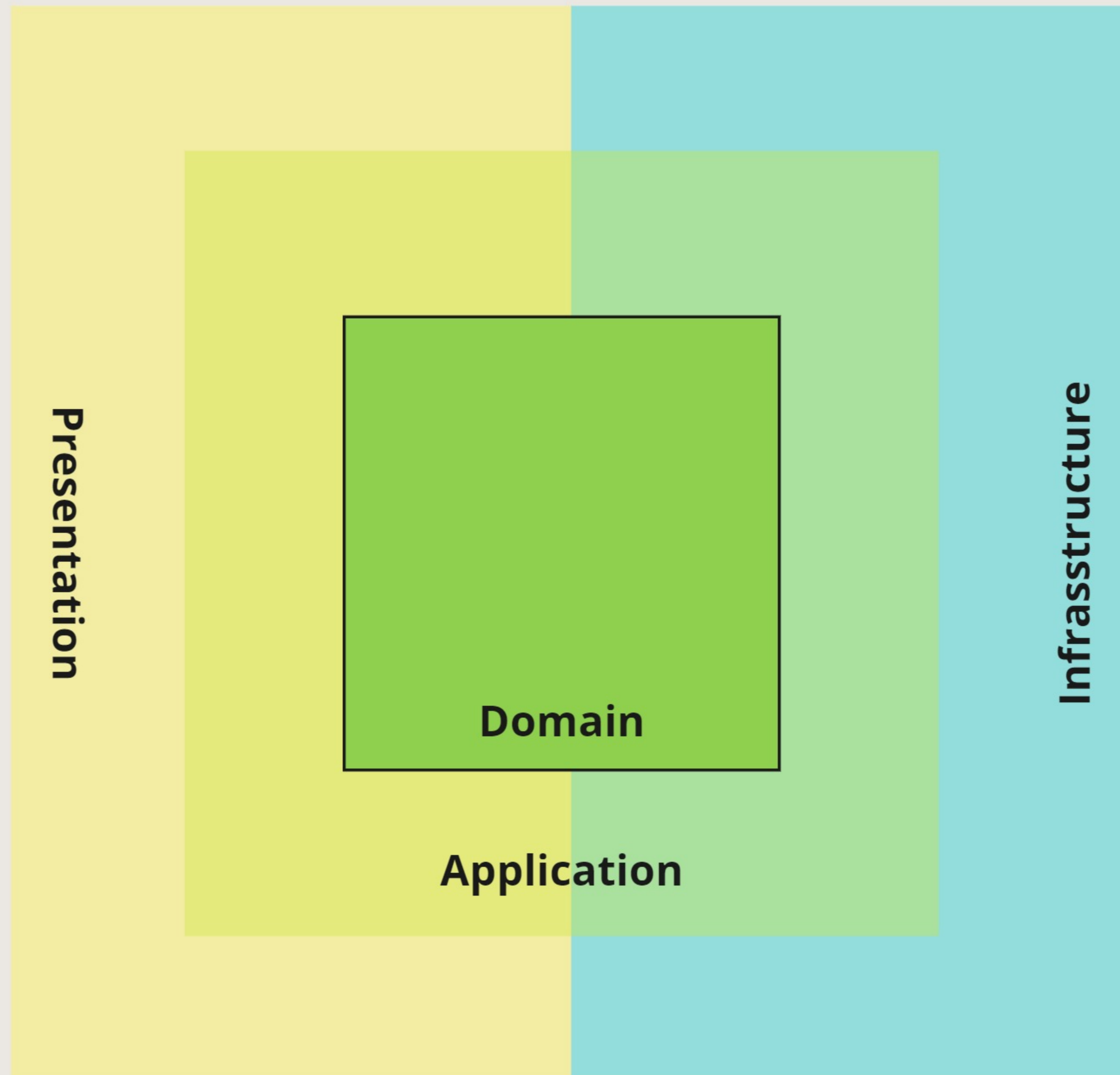


The application layer is the middle ring.

It couples use case with the domain.

Allowing flexibility within the component.

The domain

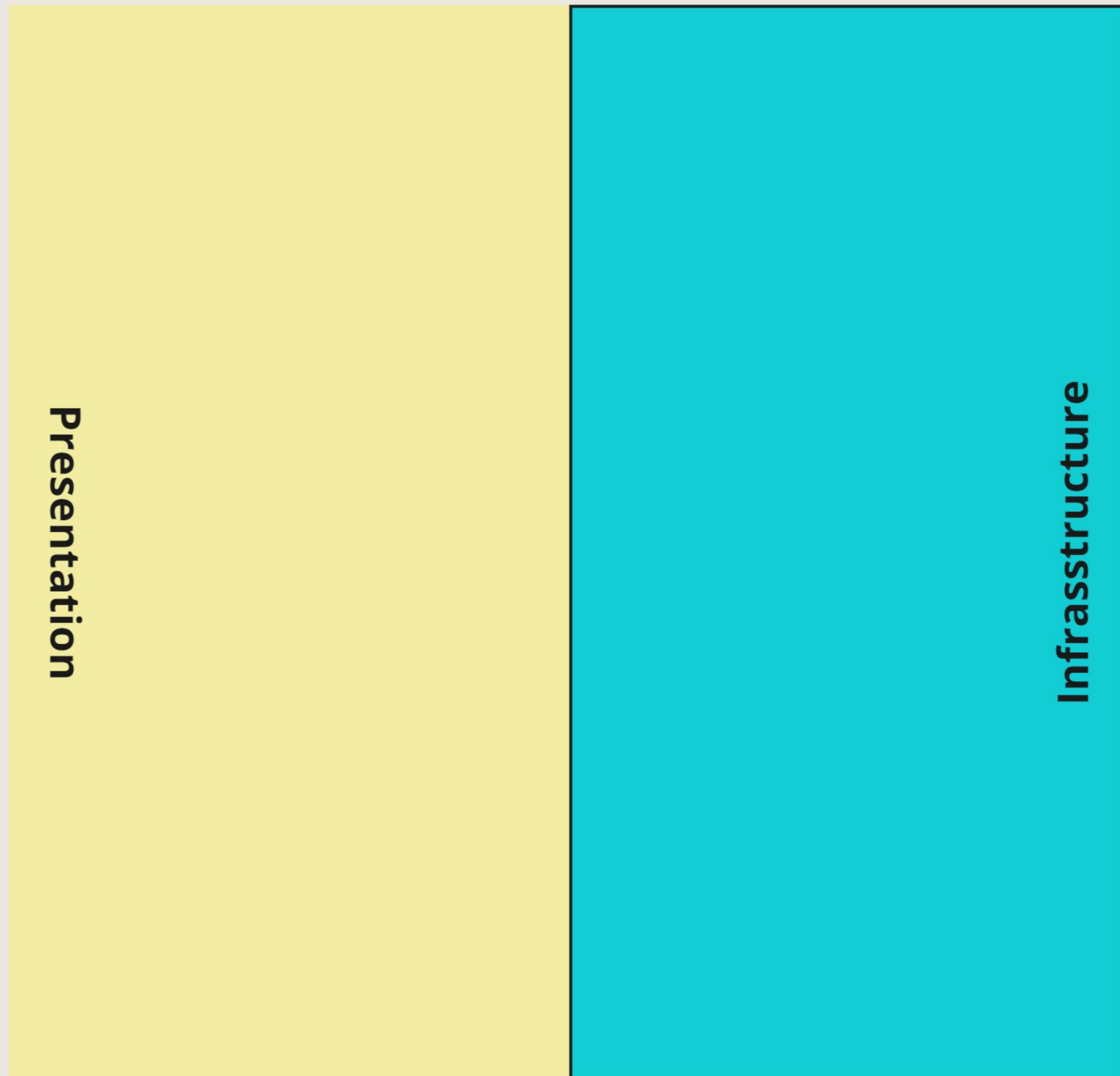


The domain layer is the inner ring.

It couples use case with the domain.

Allowing flexibility within the component.

The infrastructure



The infrastructure layer is on the outer ring.

It couples external systems with the use cases of the system.

Key principles



Decoupling

- Independence of frameworks
- Independence from API specifications
- Independence of any external service

Key benefits



Key benefits of separating our concerns

- Maintainability
- Testability
- Flexibility

Proof it!

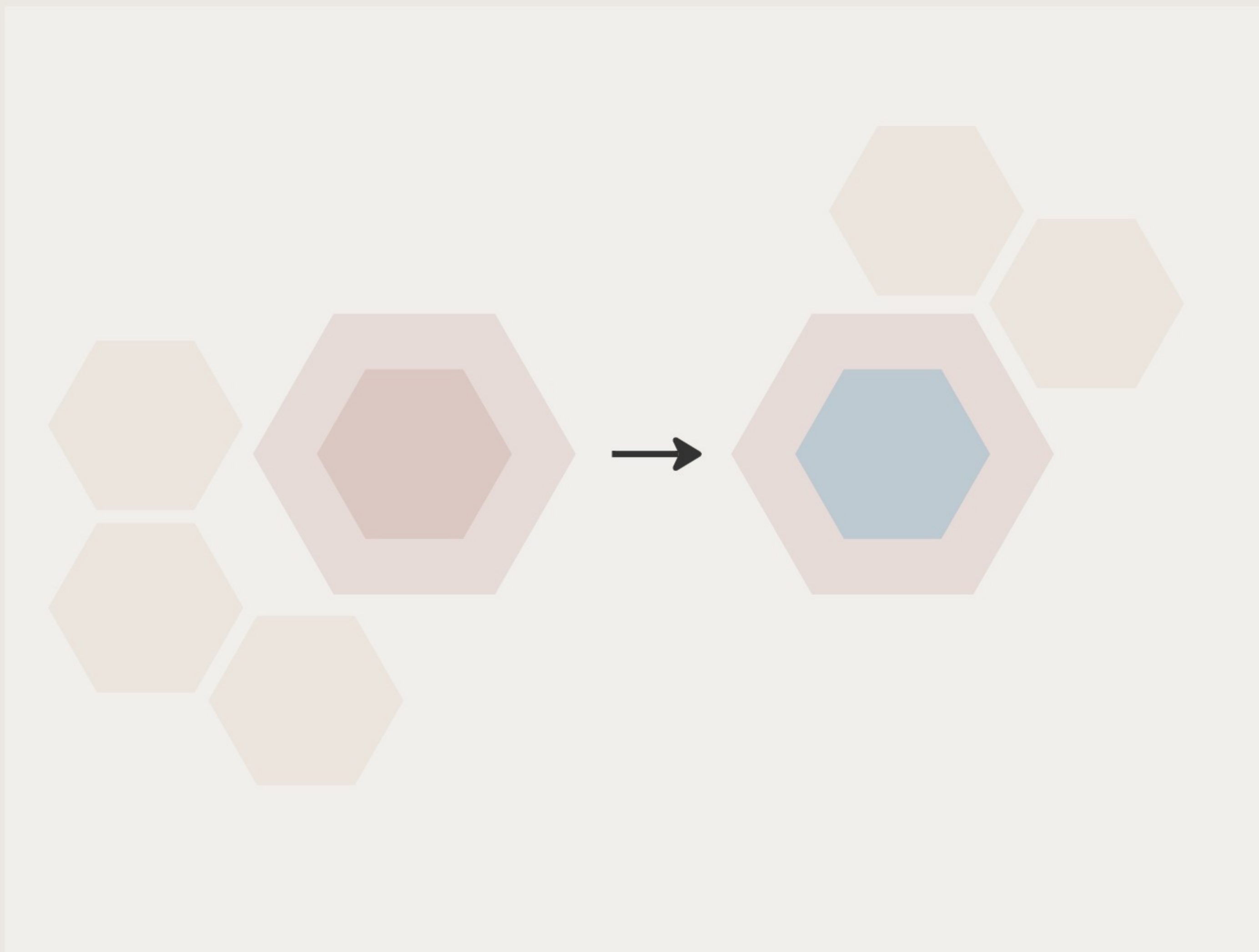


Outer ring and inner ring have different responsibilities.

Outer ring can change without affecting the inner ring.

The inner ring can change without affecting the outer ring.

Proof it!



Outer ring is the contract with the solution.

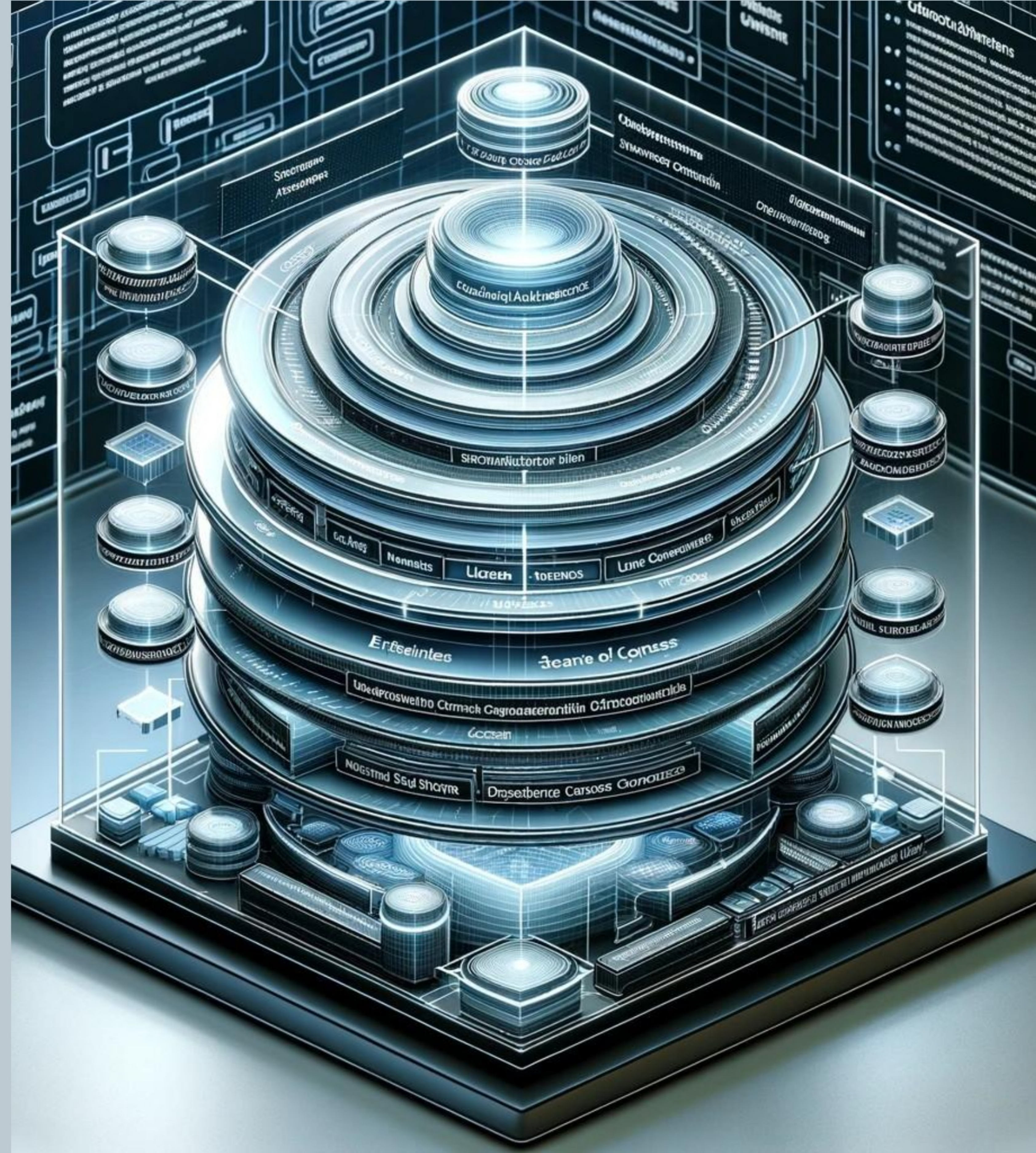
When keeping the contract the same we can change the inner ring.

Adopting new technologies.

CLEAN architecture is

- Freedom
- Flexible
- Adoptable

*Future proof
architecture*



Code

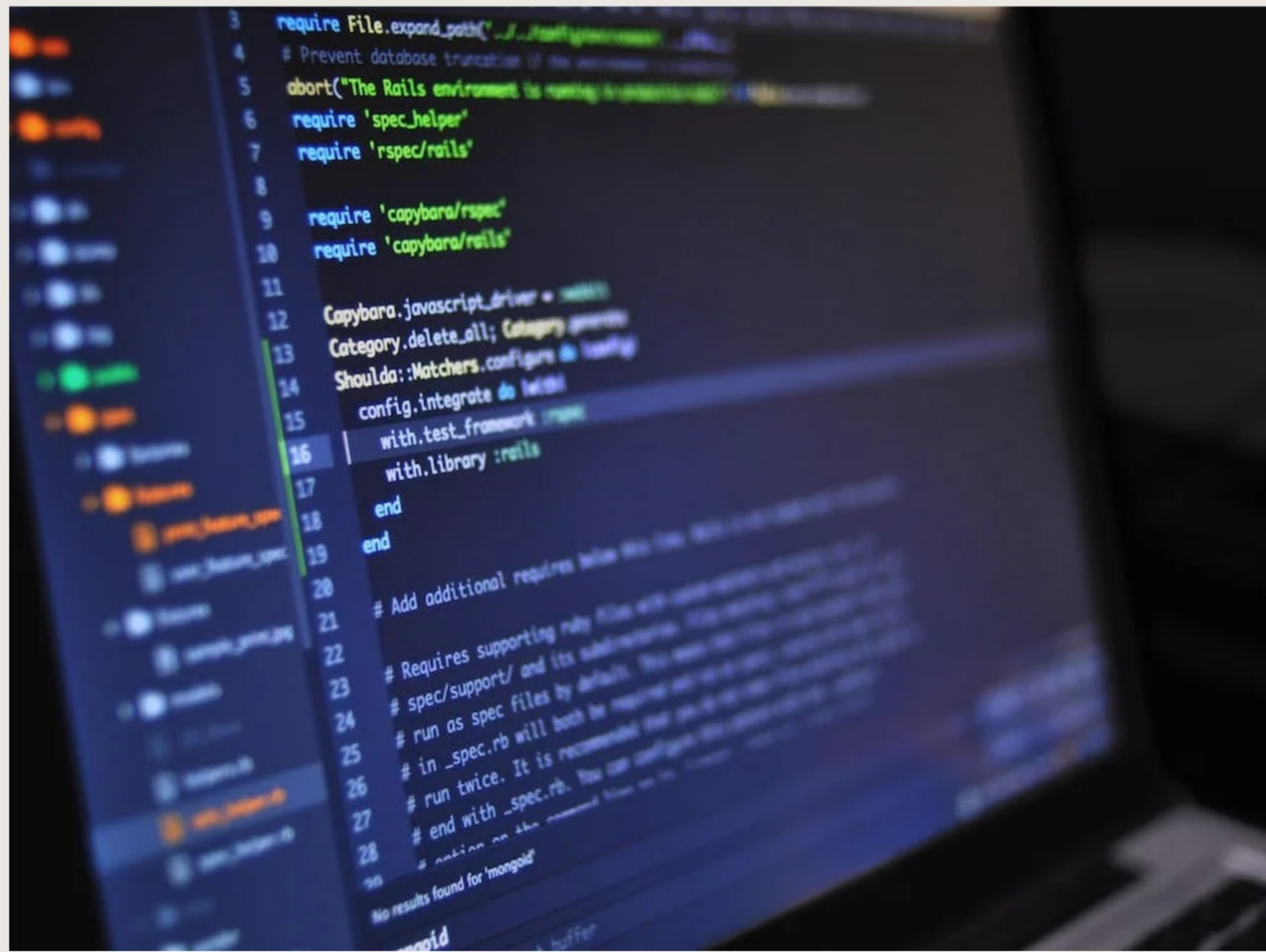
Code is a tool

The foundation is more than code.

Code is the tool to realize it.



Code



When coding we should make it.

- Readable
- Maintainable
- Scalable
- Efficient
- Testable

Codestyle

```
function (g) { var q = { vertical: !1, rtl: !1, sta
nitCallback: null, setupCallback: null, reloadCallb
temLastOutCallback: null, itemVisibleInCallback: nu
<div>", buttonNextEvent: "click", buttonPrevEvent: "c
"load.jcarousel", function () { m = !0 }); g.jcarou
this.buttonPrevState = this.buttonNextState = this.b
this.options.rtl = (g(a).attr("dir") || g("html").at
this.options.vertical ? this.options.rtl ? "right" :
"left") != -1) { g(a).removeClass(d[f]); b = d[f]; br
this.list.parents(".jcarousel-clip"), this.container
this.clip = this.container.find(".jcarousel-clip"));
this.container = this.clip.wrap("<div></div>").parer
class=" ' + b + '></div>'); this.buttonPrev = g(".j
this.buttonPrev = g(this.options.buttonPrevHTML).app
next", this.container); if (this.buttonNext.size() =
this.buttonNext.addClass(this.className("jcarousel-
this.className("jcarousel-list"))); app(f, overflow: '

```

Software design requires a solid code style.

Having a consistent codestyle is key.

The team should code as one.

Gate keeping












Applying the codestyle is key.

Use code analyzers to warn you about codestyle offenses.

Use analyzers such as StyleCop.

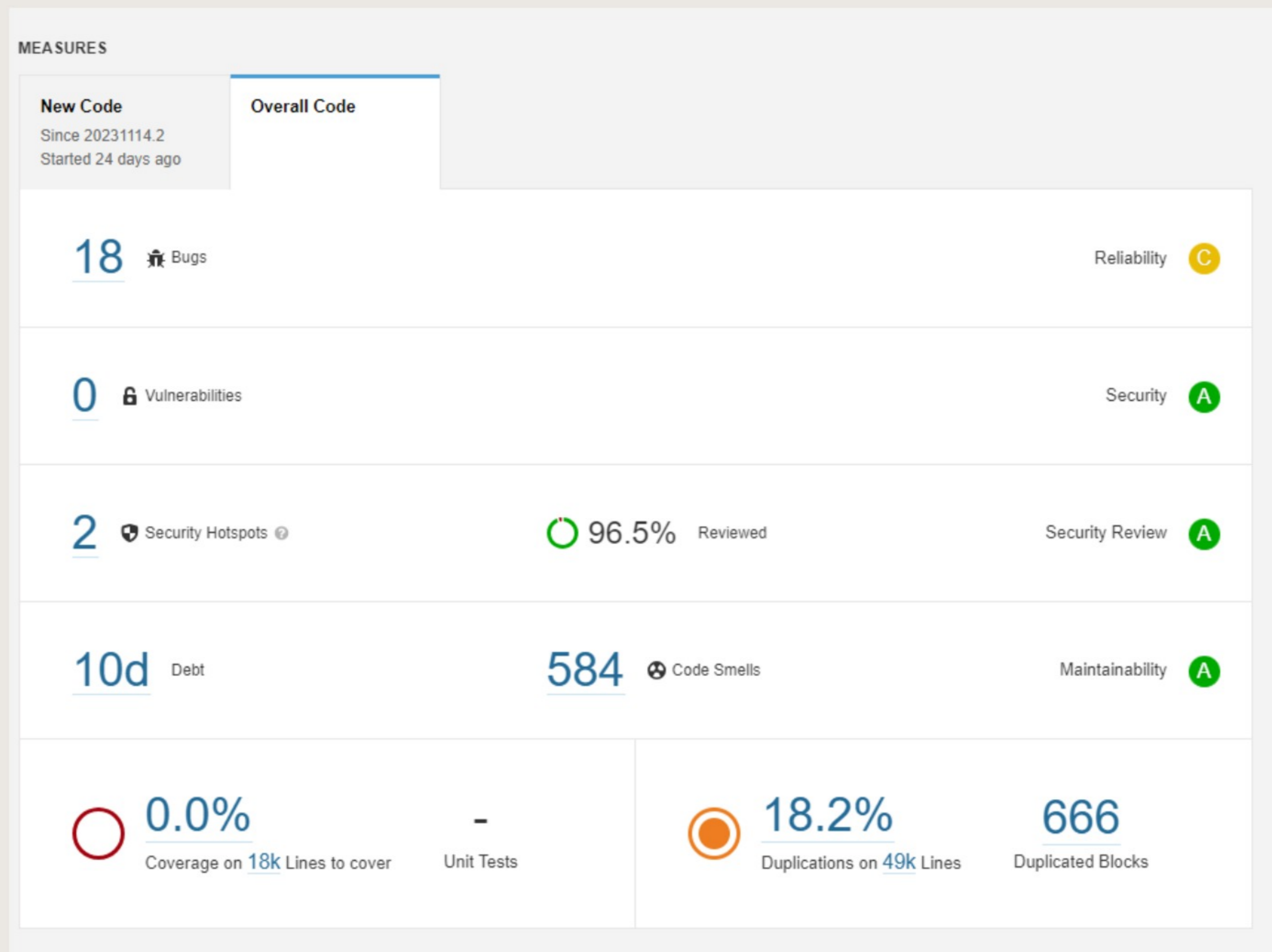
Warnings == Errors

	Code	Description
▷ 	SA1508	A closing brace should not be pre
	MA0053	Make class sealed
▷ 	SA1005	Single line comment should begin
	MA0053	Make class sealed
▷ 	SA1413	Use trailing comma in multi-line ir
▷ 	SA1122	Use string.Empty for empty string
▷ 	SA1122	Use string.Empty for empty string
▷ 	SA1505	An opening brace should not be f
▷ 	SA1028	Code should not contain trailing v

A warning is something that is not breaking to the function of the application.

It is breaking to the maintainability of the code.

Static Code Analysis

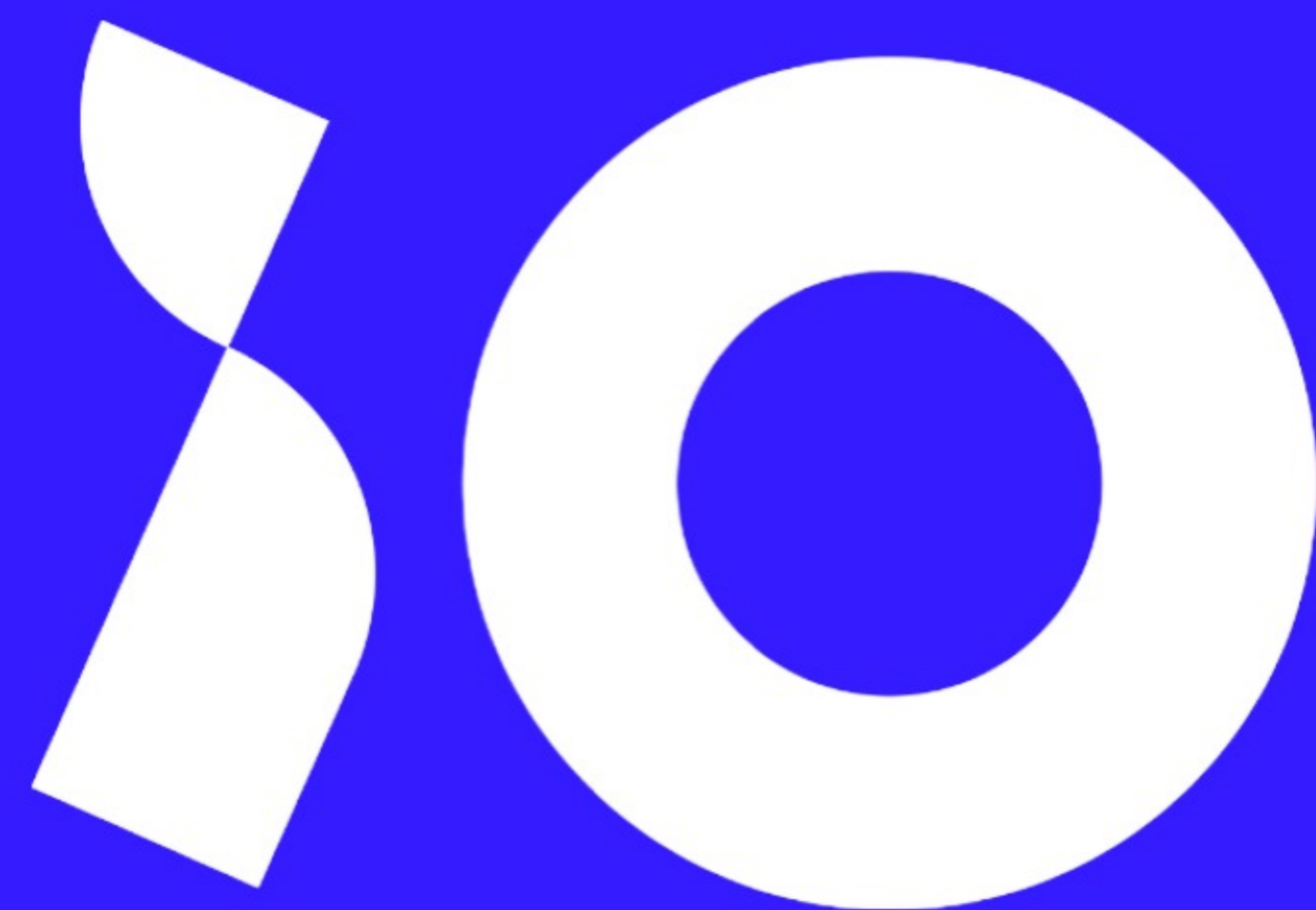


Use Static Code Analysis for further insights.

Avoid technical dept.

Use tools such as SonarQube.

That's all



Contact me

Roy Berris

@royberris 

www.berris.dev