

Understanding Foundational Architectural Principles



Gill Cleeren

CTO Xpirit Belgium

@gillcleeren | xspirit.com/gill

Overview

- Foundational design principles
- Different application architecture styles
- Understanding Clean Architecture





Foundational Architectural Principles

Important Design Principles

**Dependency
inversion**

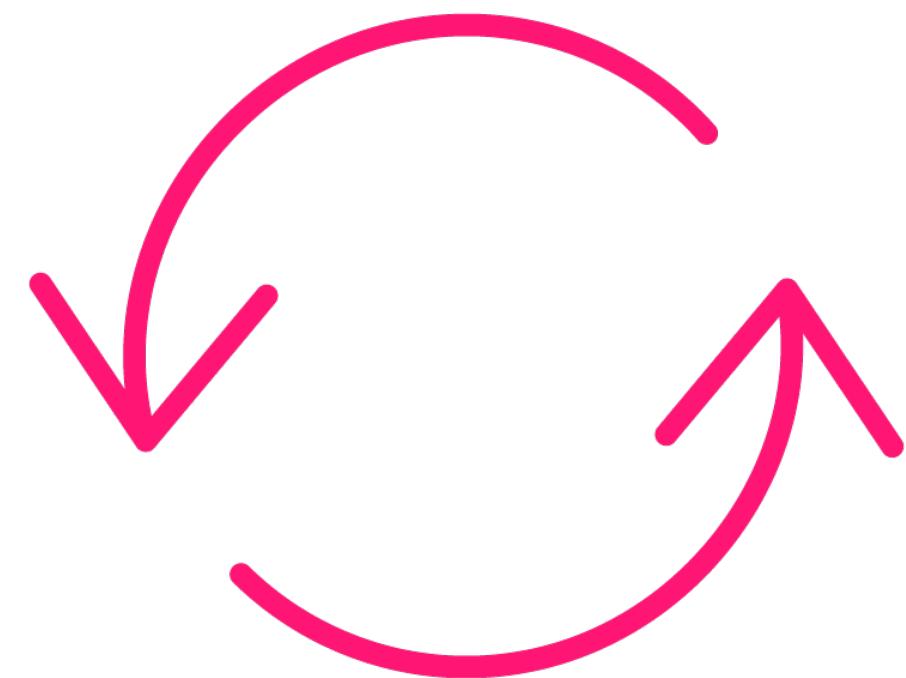
**Separation of
concerns**

Single responsibility

DRY

**Persistence
ignorance**





Dependency Inversion

- Decoupling modules

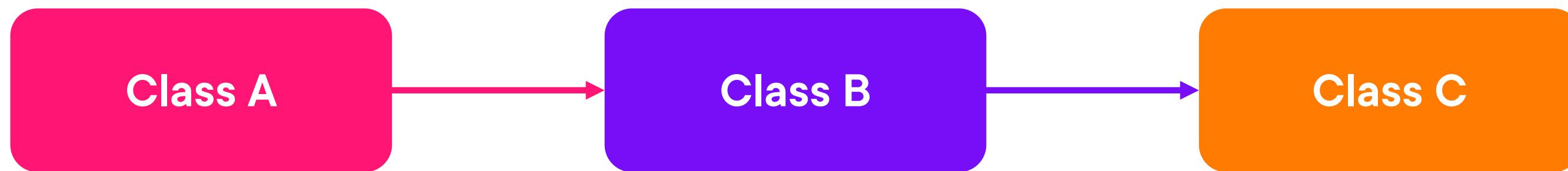
Dependencies should be pointing to abstractions

- Typically top to bottom

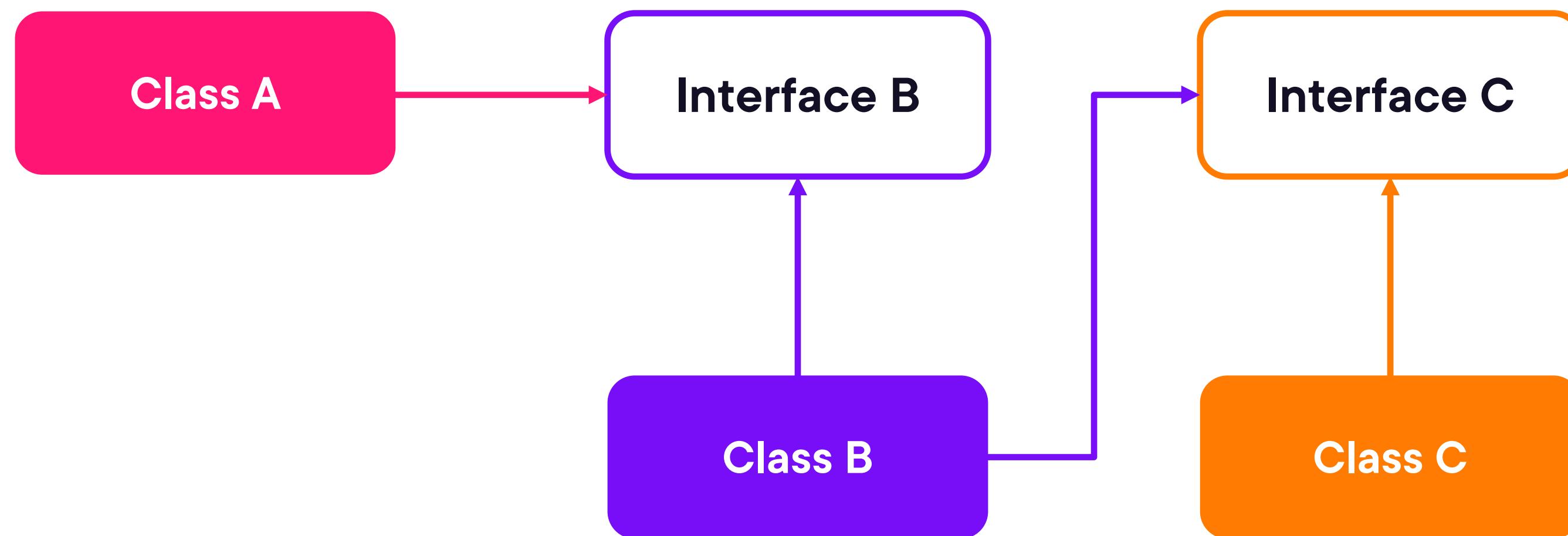
Helps with building more loosely-coupled applications



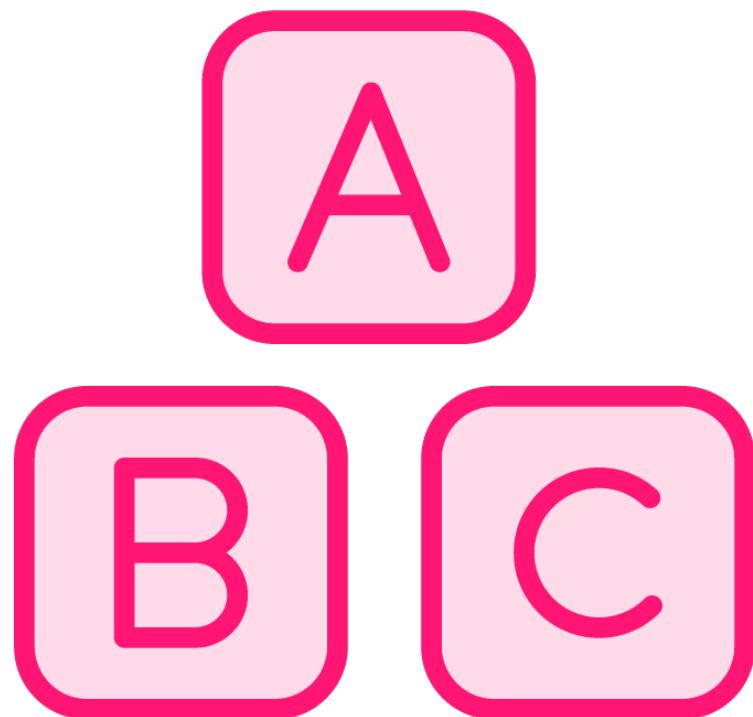
Typical Approach



Adding Dependency Inversion



Separation of Concerns



Split into blocks of functionality

- Each covering a concern

More modular code

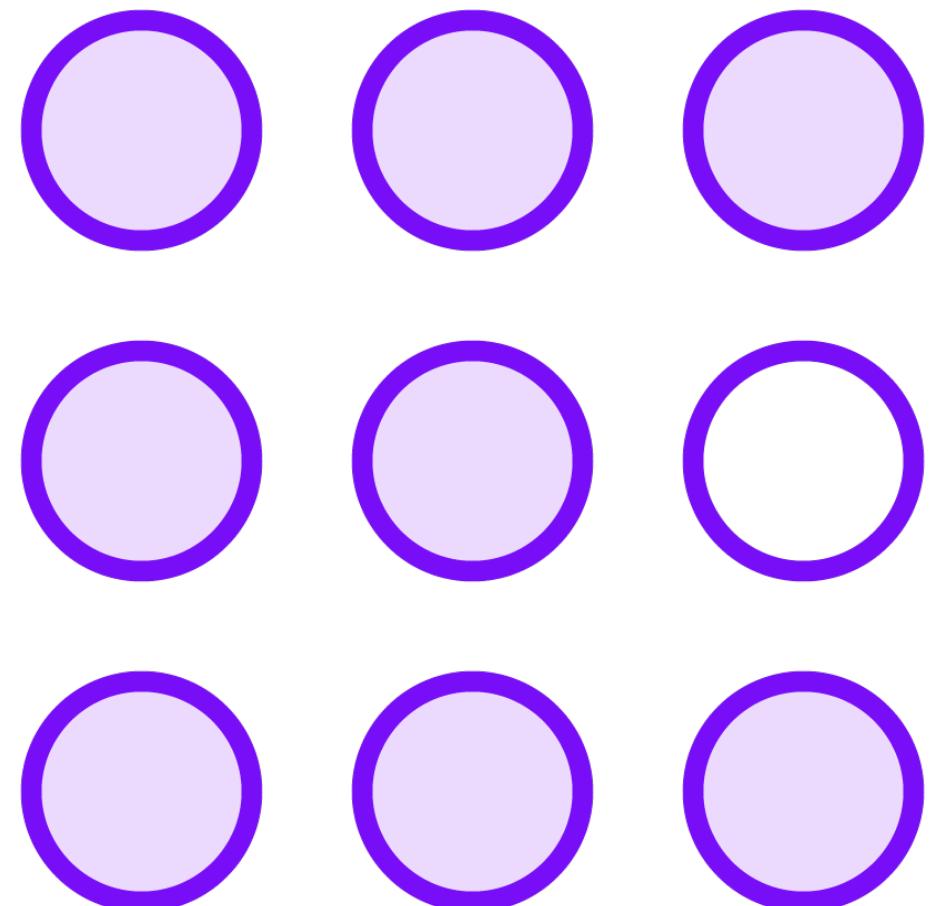
- Encapsulation within a module

Typical layered application

Easier to maintain



Single Responsibility



OO terminology

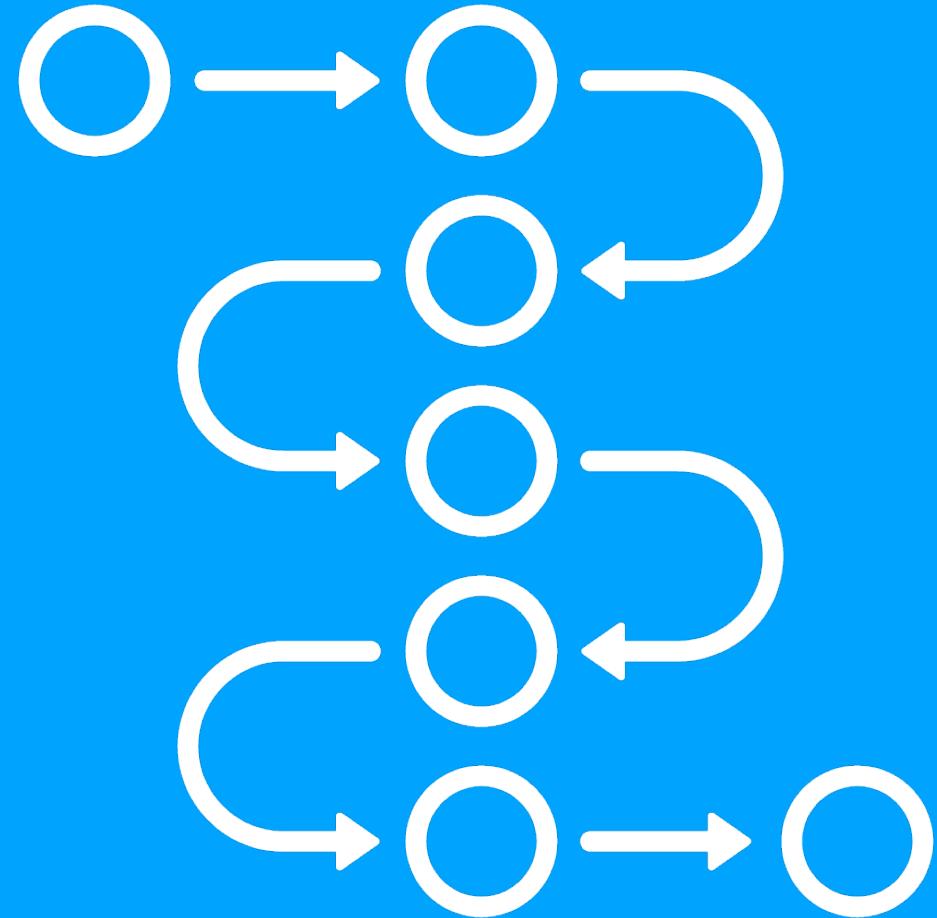
Each block should just have one single responsibility which it encapsulates

More, new classes are created

Can be extended to application-level

- Different layers have their own responsibility**





DRY (aka Don't Repeat Yourself)

Less code repetition

Easier to make changes



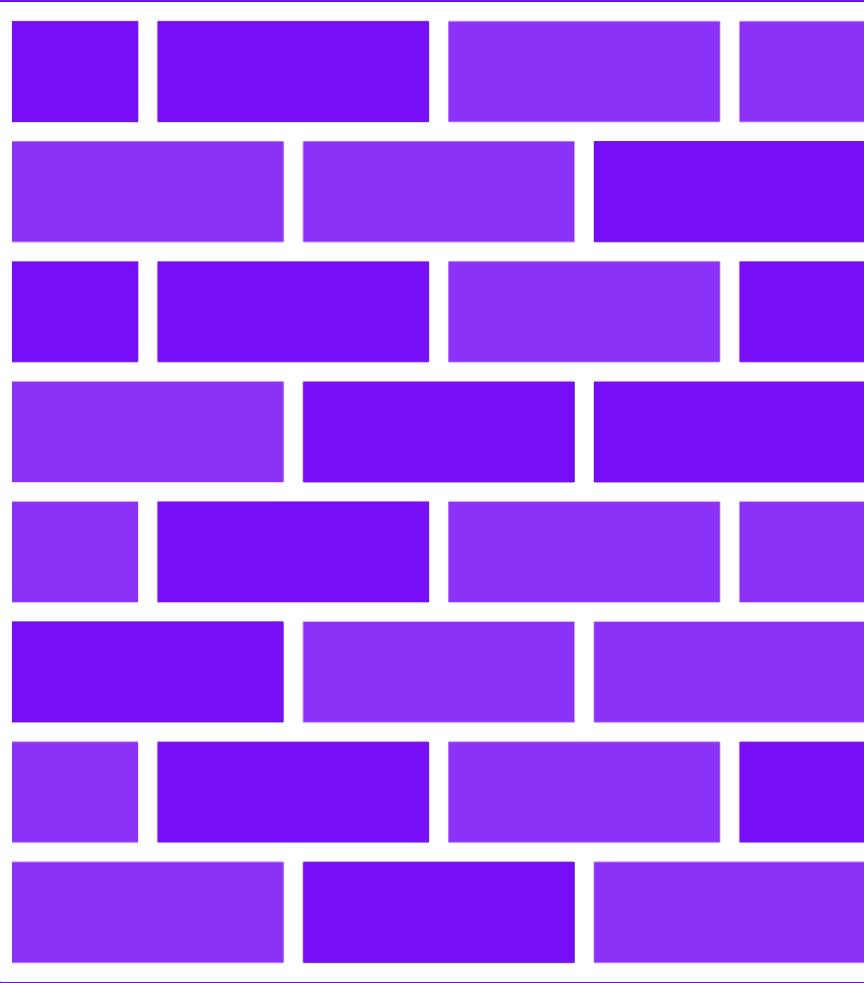
Persistence Ignorance

POCO

Domain classes
shouldn't be
impacted by how
they are persisted

Typically requires base
class or attributes





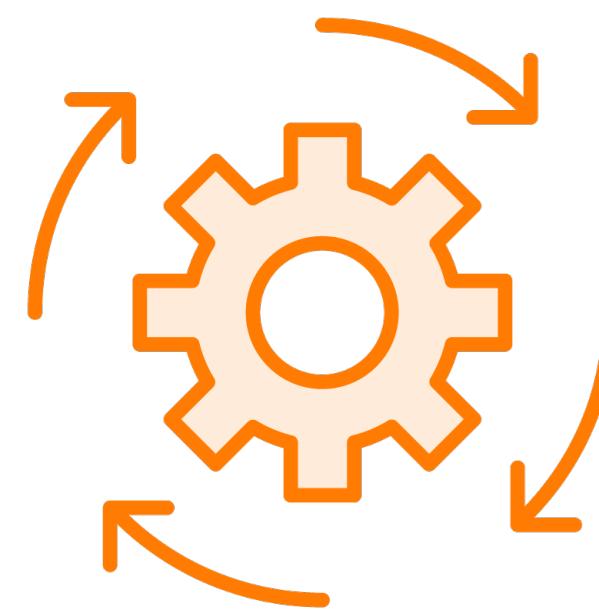
Foundational Patterns

These will come in useful
throughout this entire course!



Different Application Architecture Styles

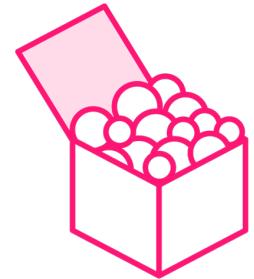
Different Types of Application Architecture



All-in-one
architecture



All-in-one Architecture



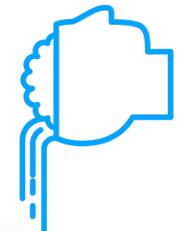
Contained typically in one (large) Visual Studio project



File → New Project



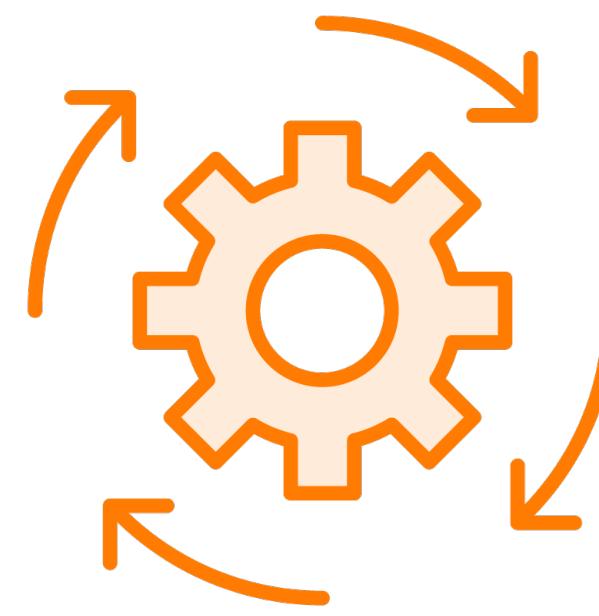
“Layers” are folders



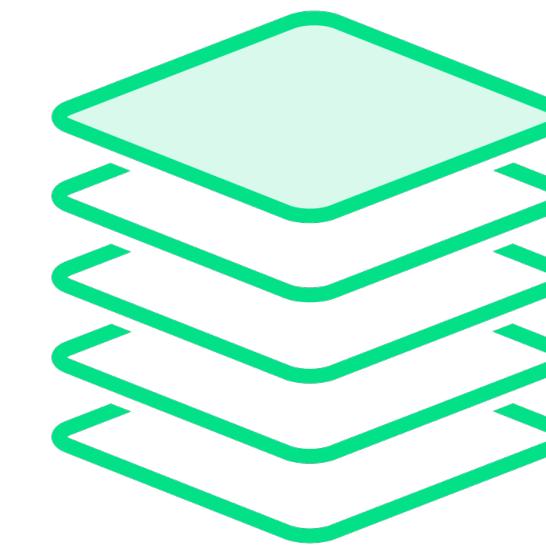
Can be difficult to maintain



Different Types of Application Architecture



**All-in-one
architecture**



Layered architecture

Layered Architecture



Split according to concern



Promote reuse



Easier to maintain



Pluggable



Typical Layered Architecture

Presentation layer

Business logic layer

Data access layer





Disadvantages of Layered Architecture

Still “coupling” between layers

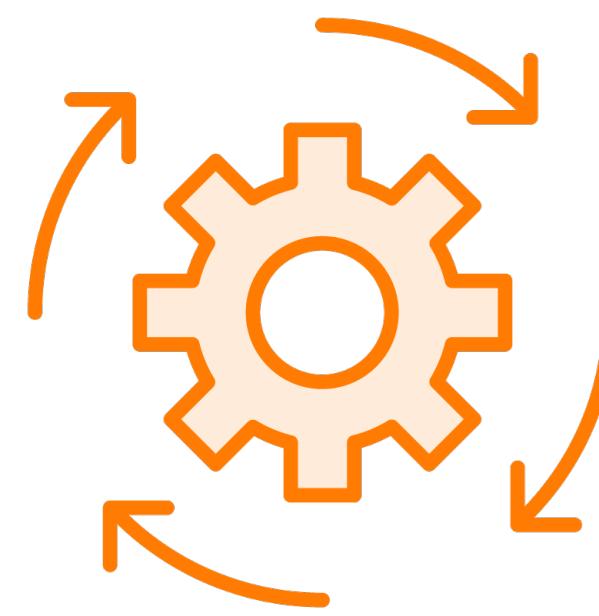
Behaves as single application



Understanding Clean Architecture



Different Types of Application Architecture



All-in-one
architecture



Layered architecture



Clean architecture



Introducing Clean Architecture

Based on design principles

Separate concerns

Create maintainable and testable
application code





Variation on hexagonal and onion architecture

- Introduced in 2012

Separation of concerns

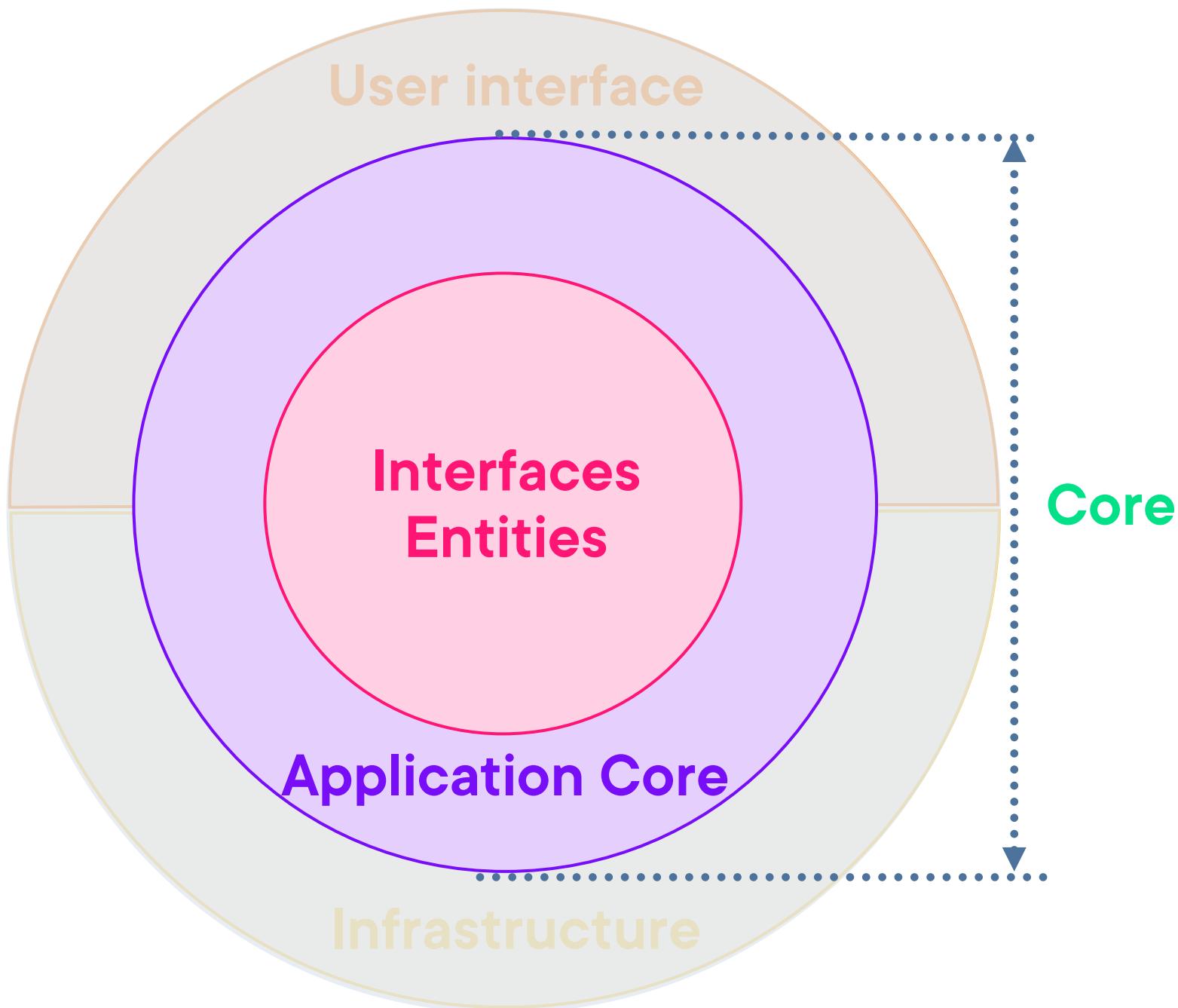
Loose coupling

Independent of “external” influences

- UI
- Database



Circular Design



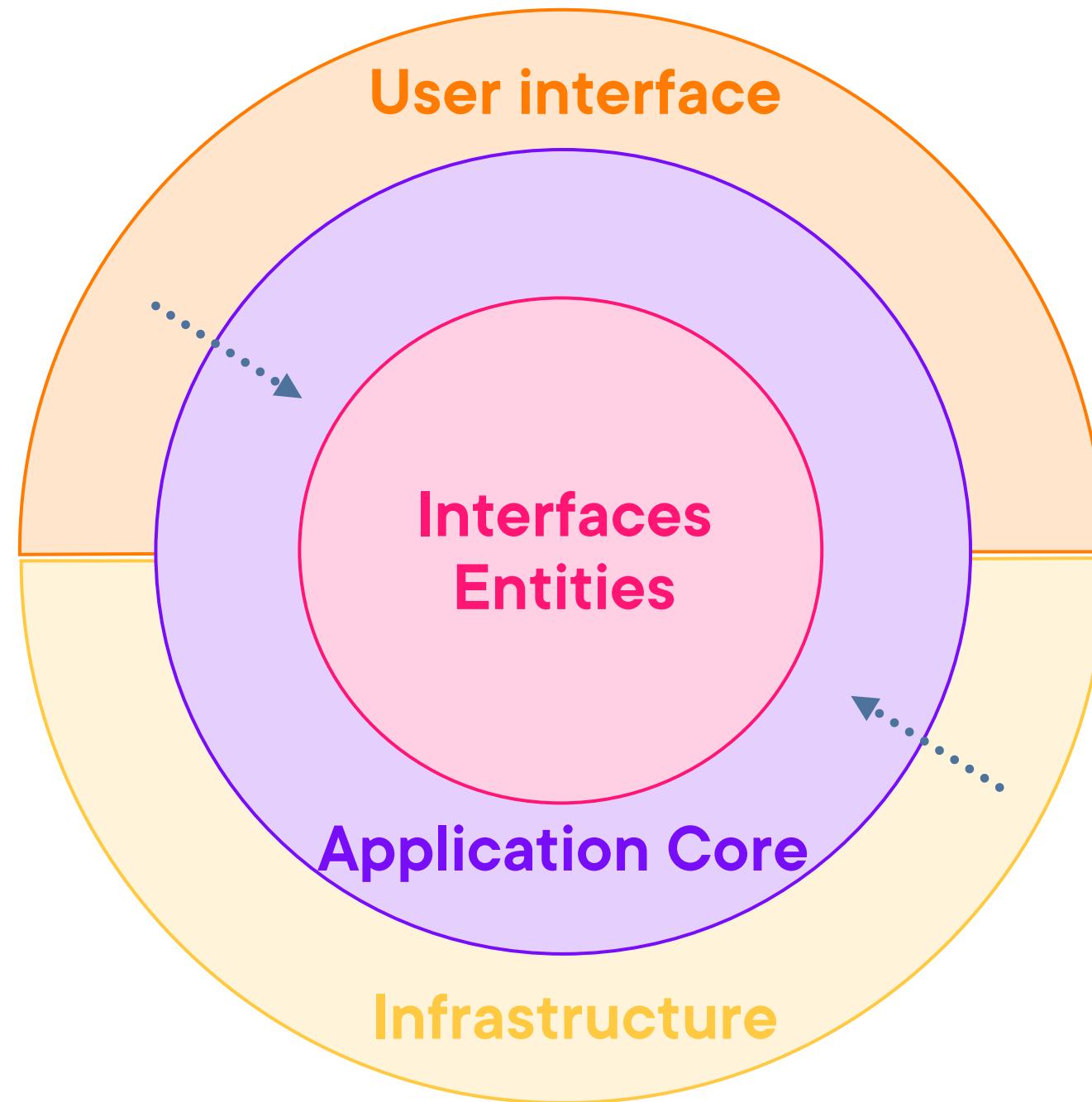
Different circles for different “layers”

Application Core

- Abstractions (high-level)
- Interfaces and entities
- Business logic at the center of the application (use-cases)
- Agnostic to outer circles
- Has no dependencies on external influences



Understanding Clean Architecture



Outer circles are infrastructure (mechanisms)

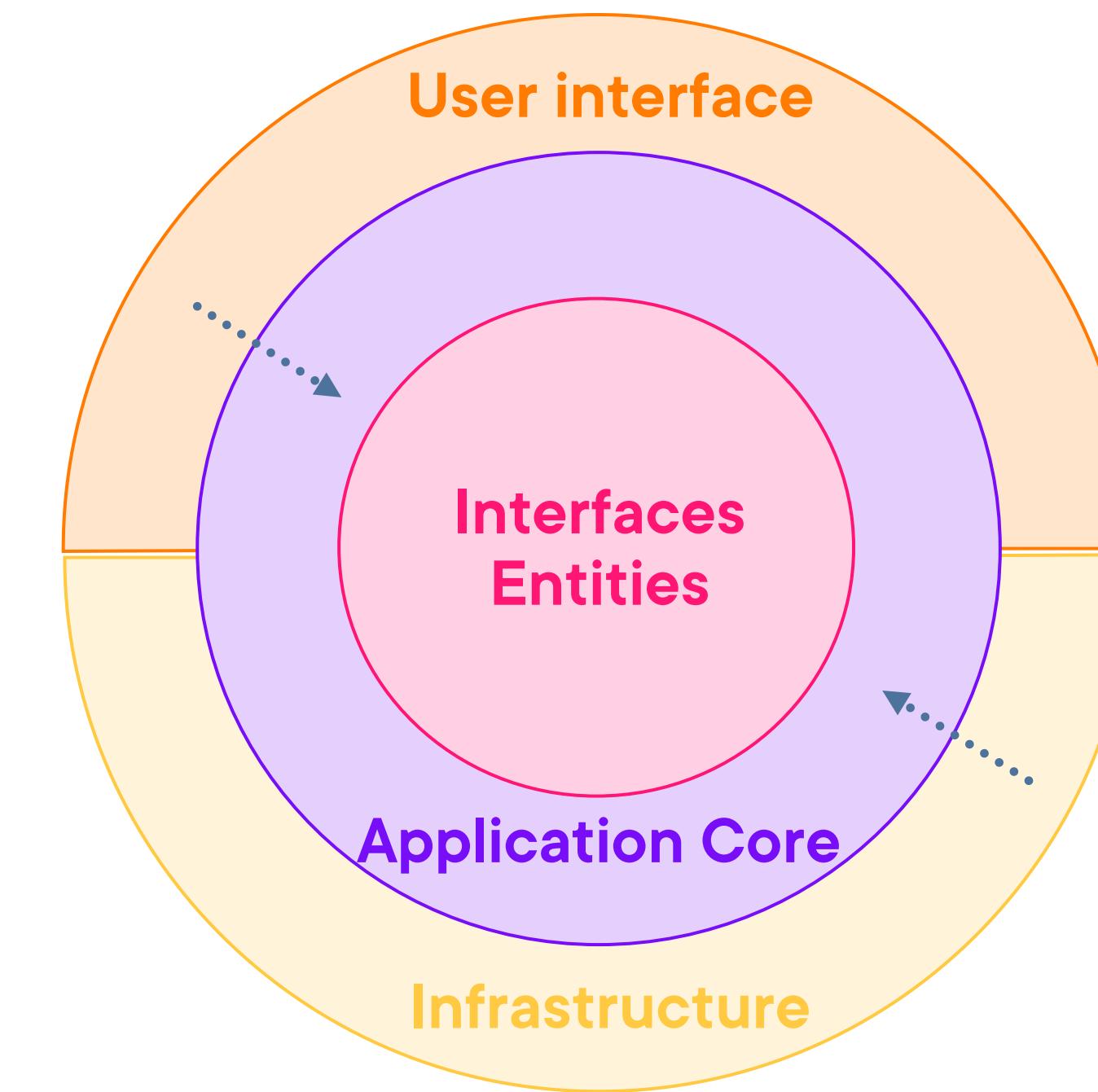
- Depends on Core
- Implements interfaces from Core

Dependencies are inverted

- Pointing inwards
- UI
– Dependencies on Core



Understanding Clean Architecture



Two Important Principles



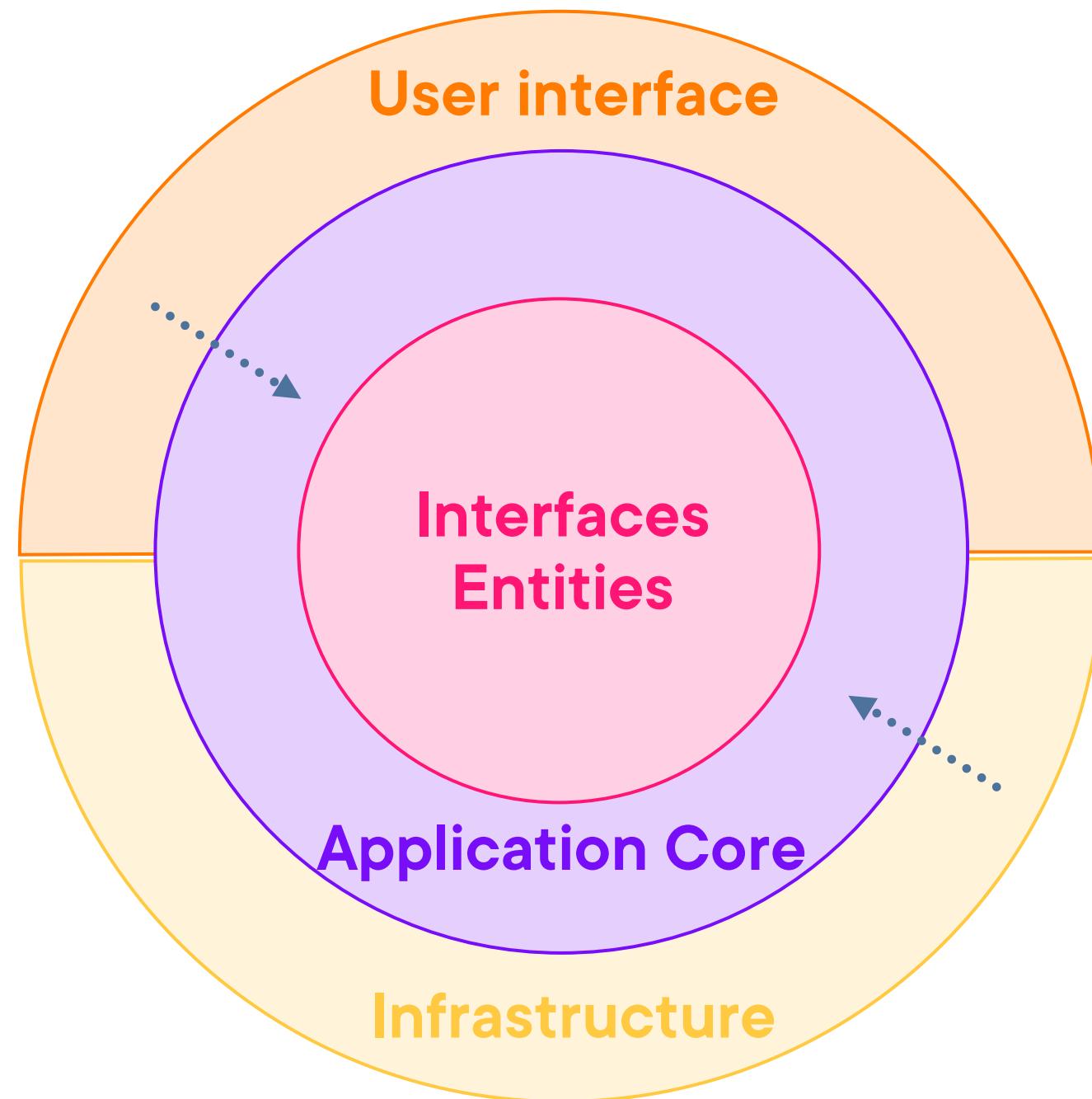
Dependency Inversion



Mediator



Who Goes Where?



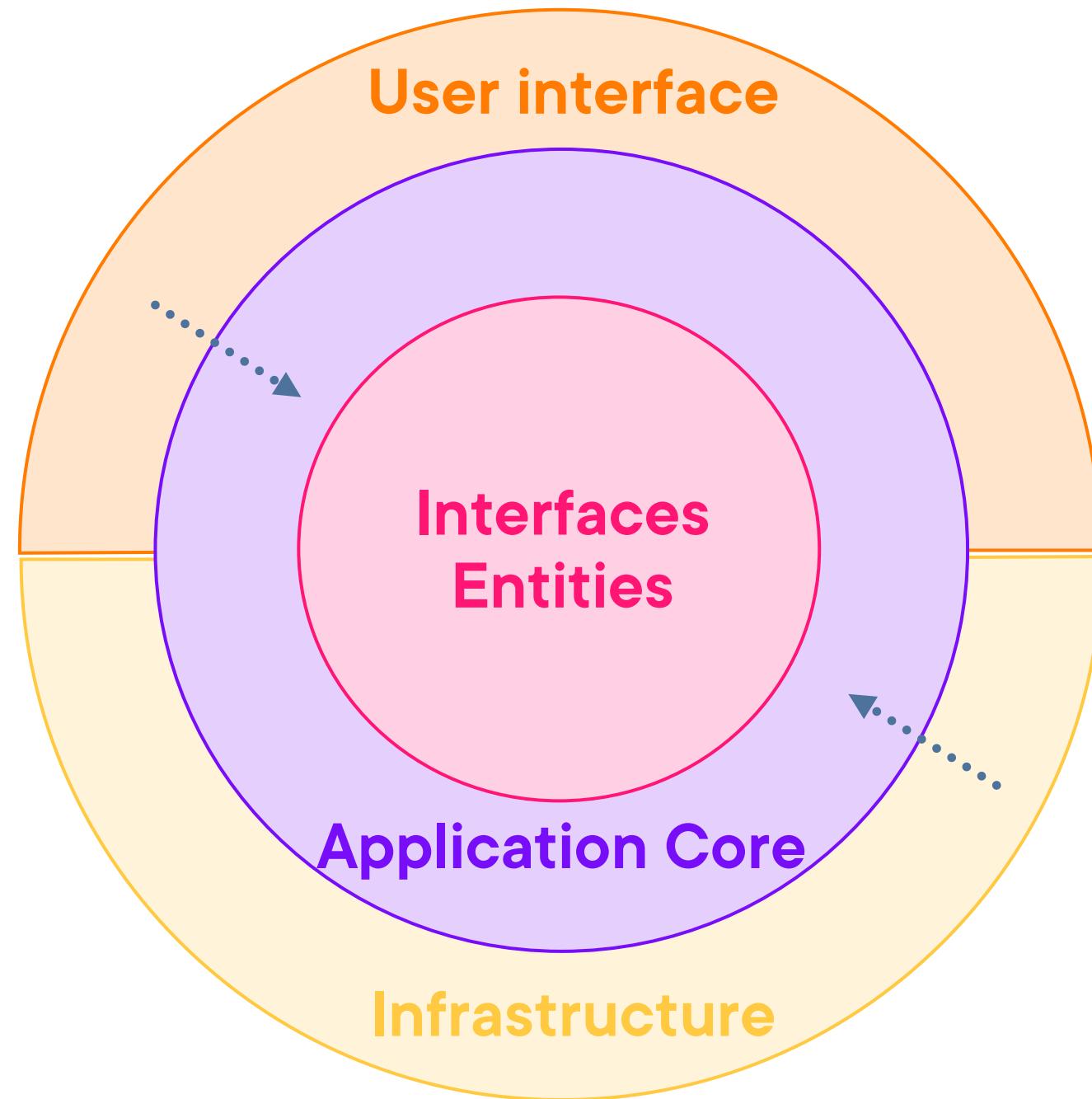
Core

- Entities
- Interfaces
 - Core
 - Infrastructure
- Services
- Exceptions

No dependency to any Infrastructure-related code or package



Understanding Clean Architecture

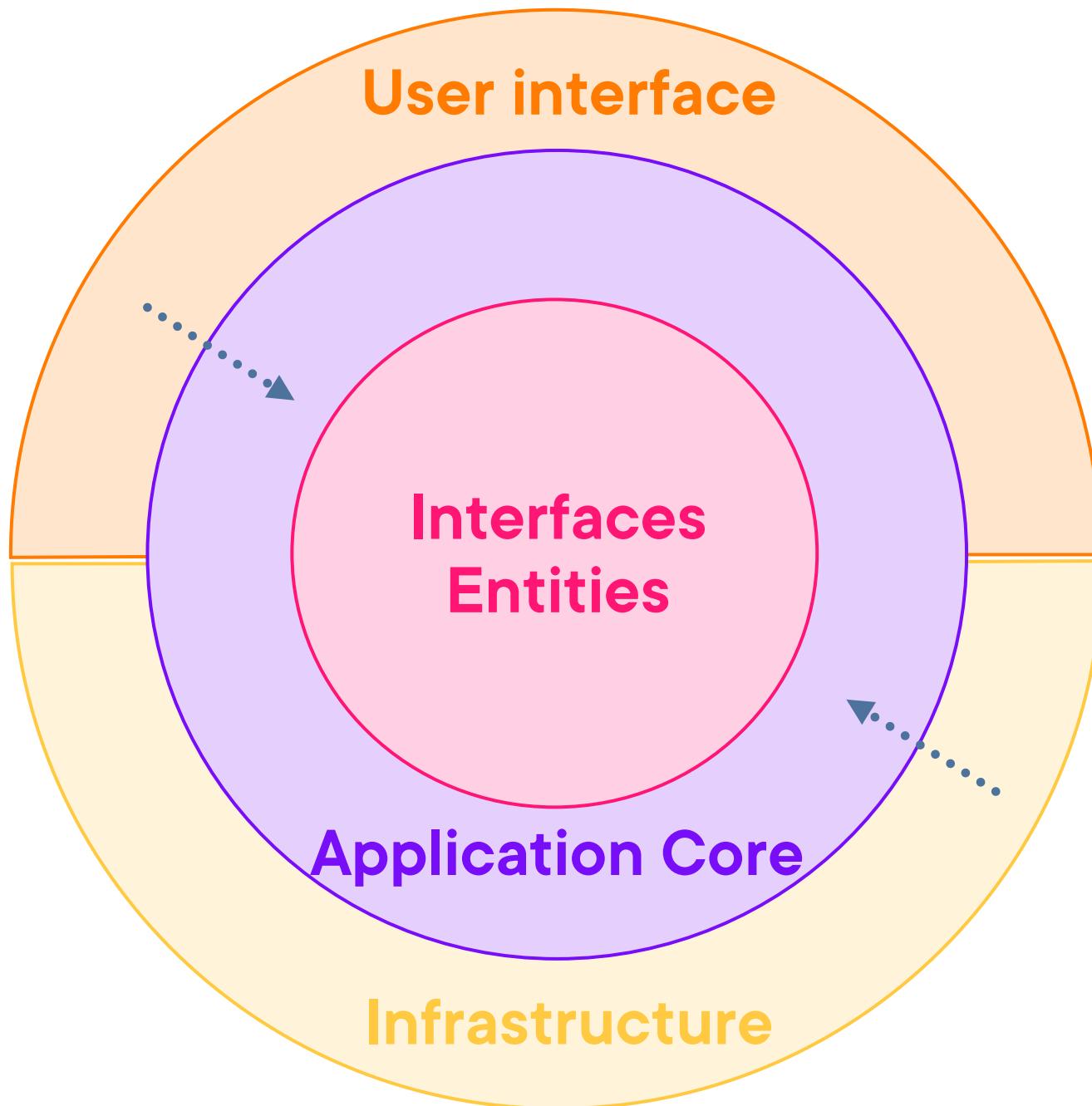


Infrastructure

- Data access (EF Core)
- Logging
- API Clients
- Identity
- File access



Understanding Clean Architecture



UI

- API/MVC/Razor
- Specific ASP.NET Core items
 - Middleware
 - Filters
- Interact with services through MediatR
 - Loose coupling
 - Lightweight controllers



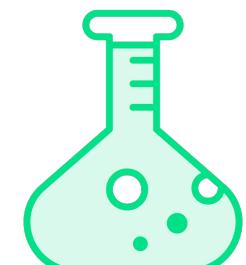
Clean Architecture Benefits



Independent of UI or used frameworks



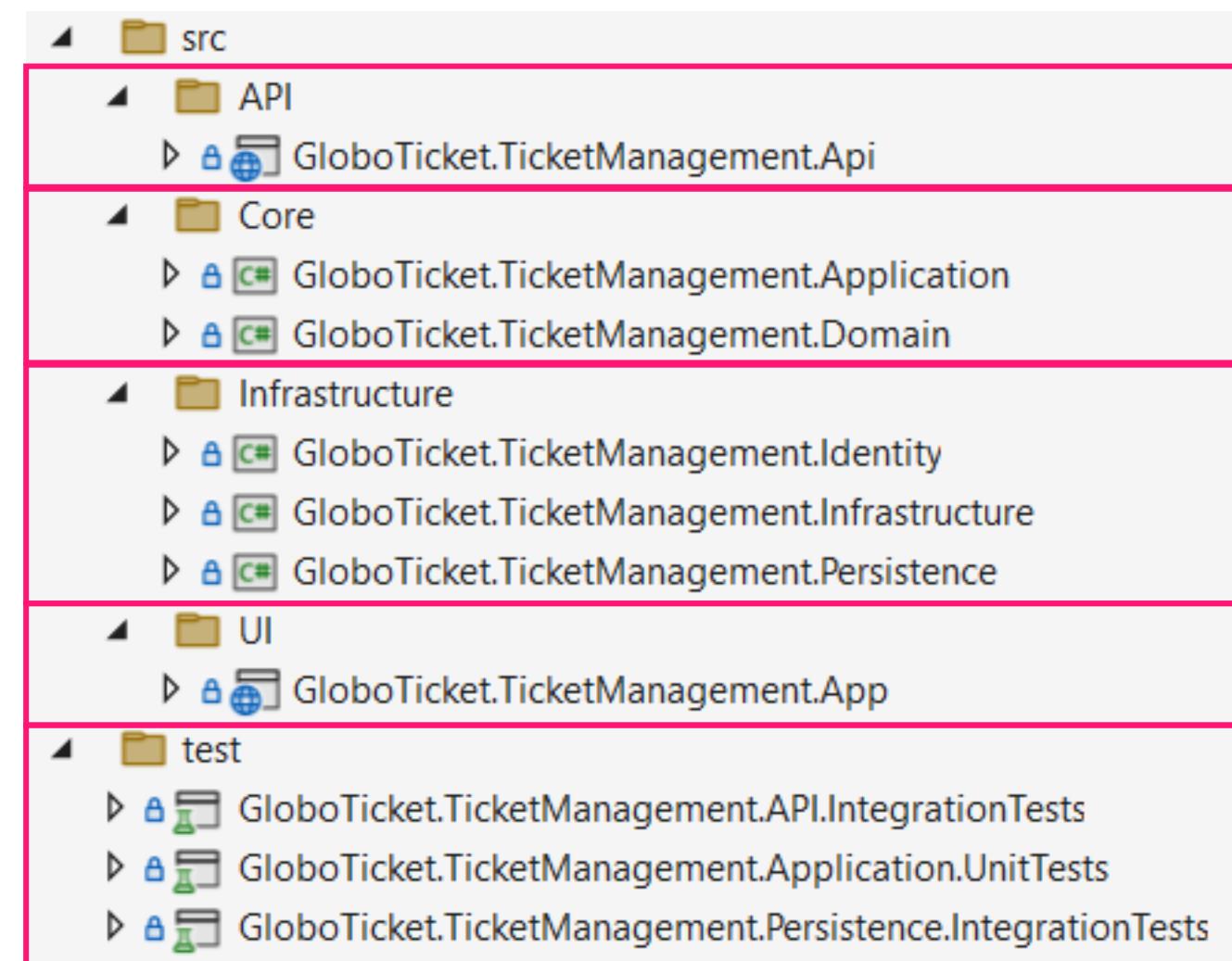
No knowledge of used database

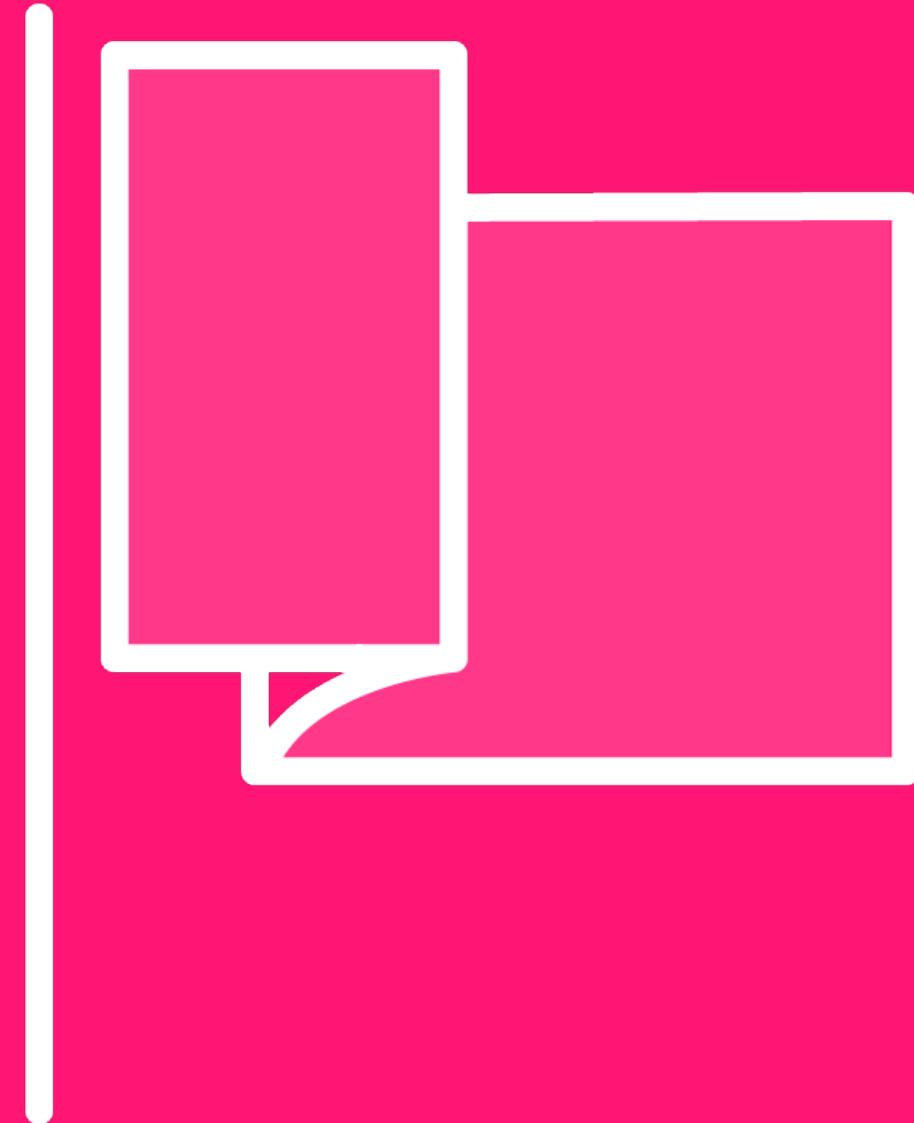


Testable and maintainable



High-level Code Organization





The end result...

**Testable and maintainable
application code**

Not for every application though!



Summary

Reviewed foundational design principles

Clean architecture relies on these

- Will result in more testable and maintainable applications



Up Next:

Creating the application core

