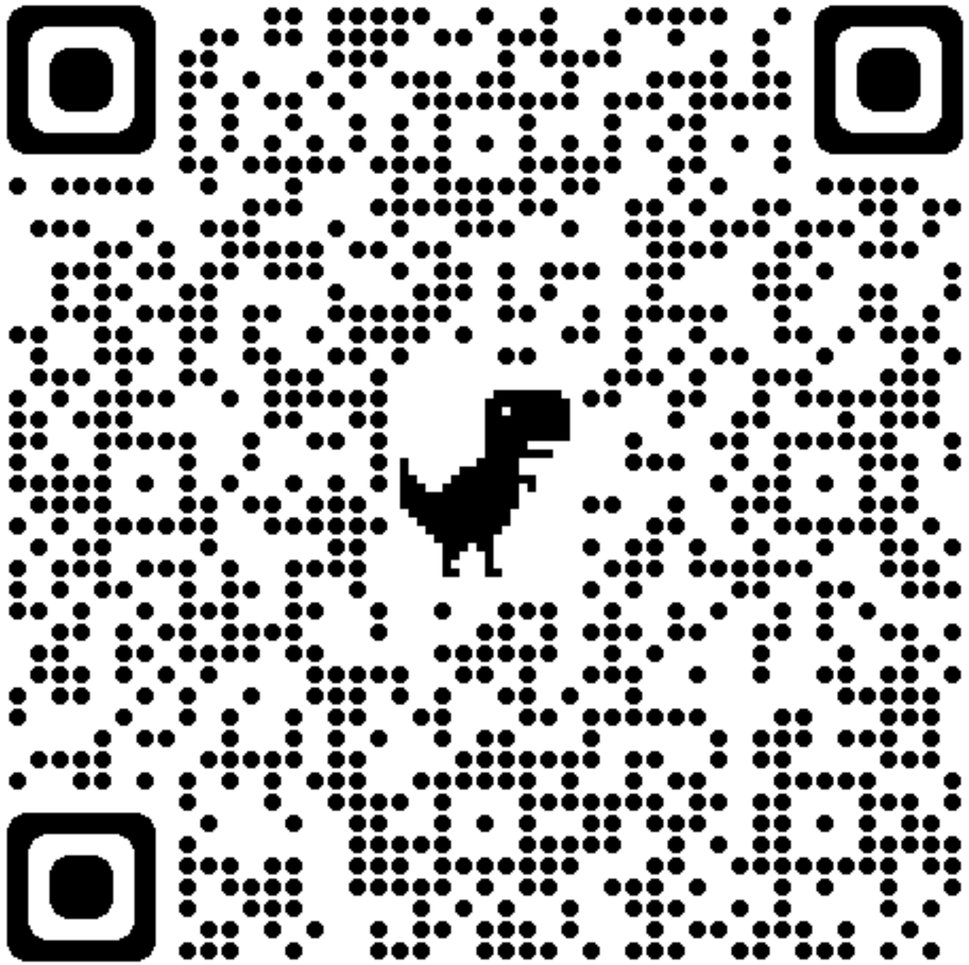


첫 단추부터 시작하는 SW아키텍처

Layered 아키텍처부터 알아보고 개선책을 찾아보자

원티드

주제

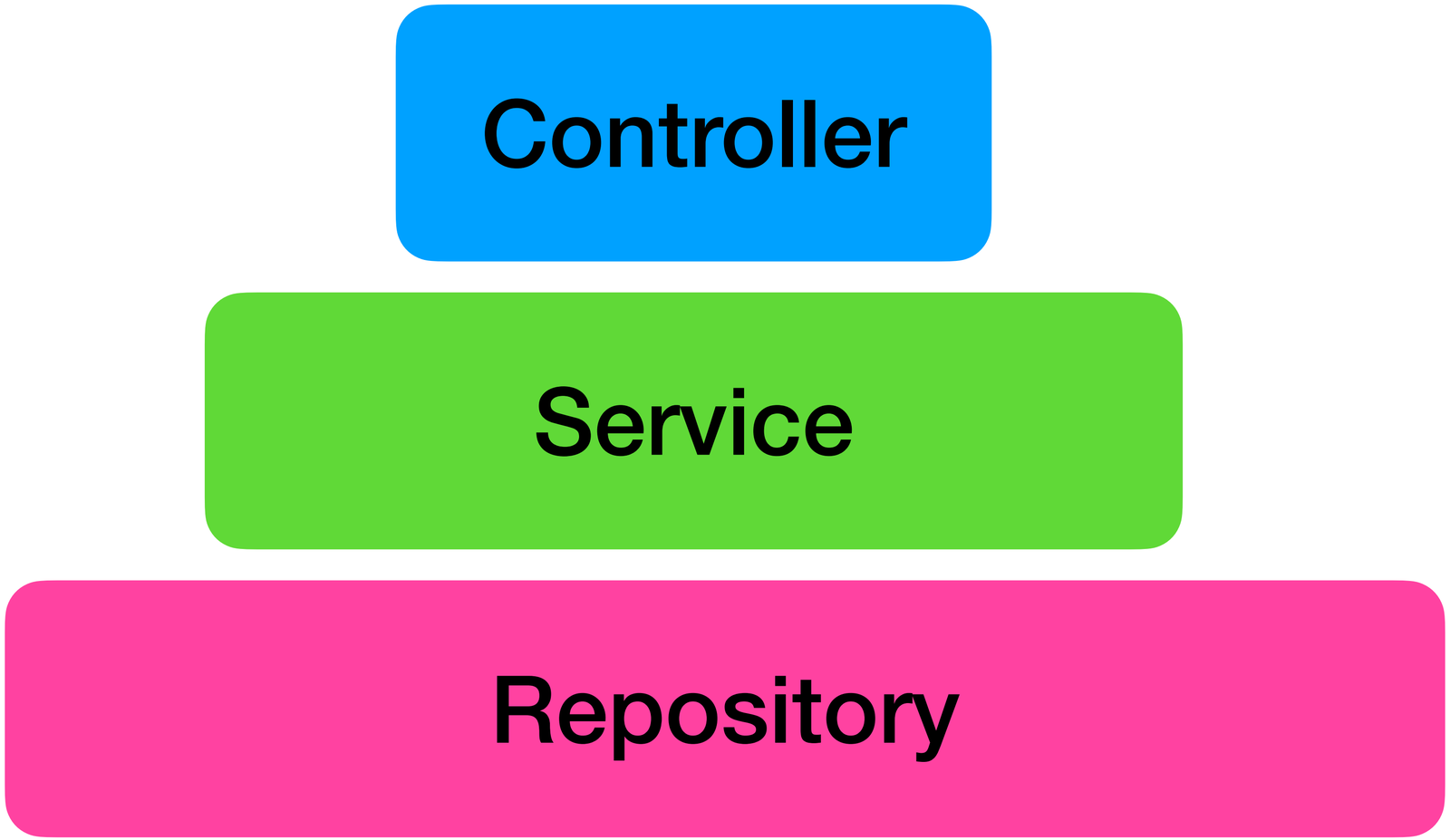
- Layered 아키텍처를 보완한 아키텍처 알아보기
- 소프트 웨어 아키텍처에서 도메인이 필요한 이유 알아보기
- 개선한 아키텍처에서 생기는 문제점 알아보기

저번 시간에는?

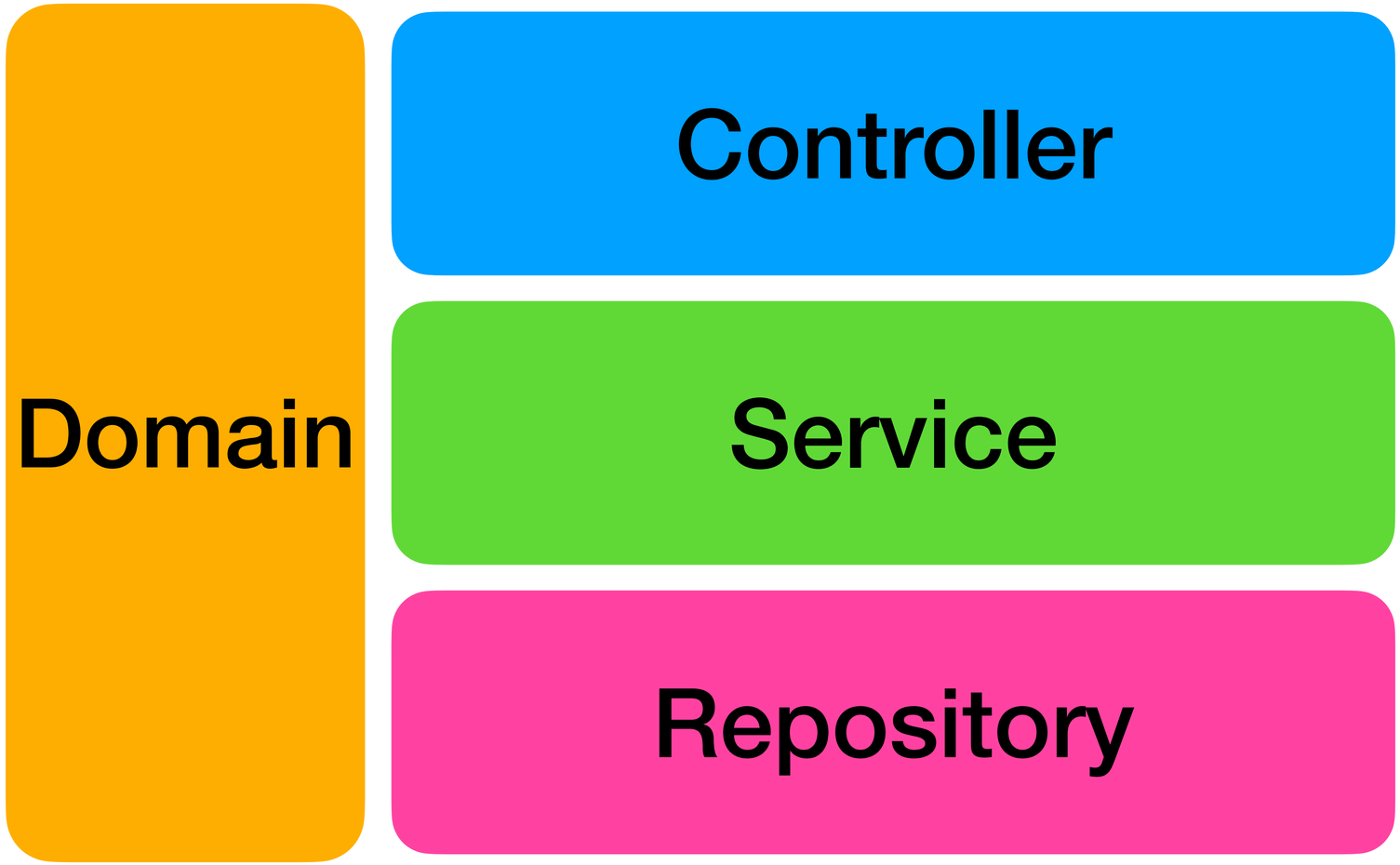
- layered Architecture에 대해 학습
- layered Architecture를 개선하는 방법에 대해 학습
    - SOLID
    - TEST

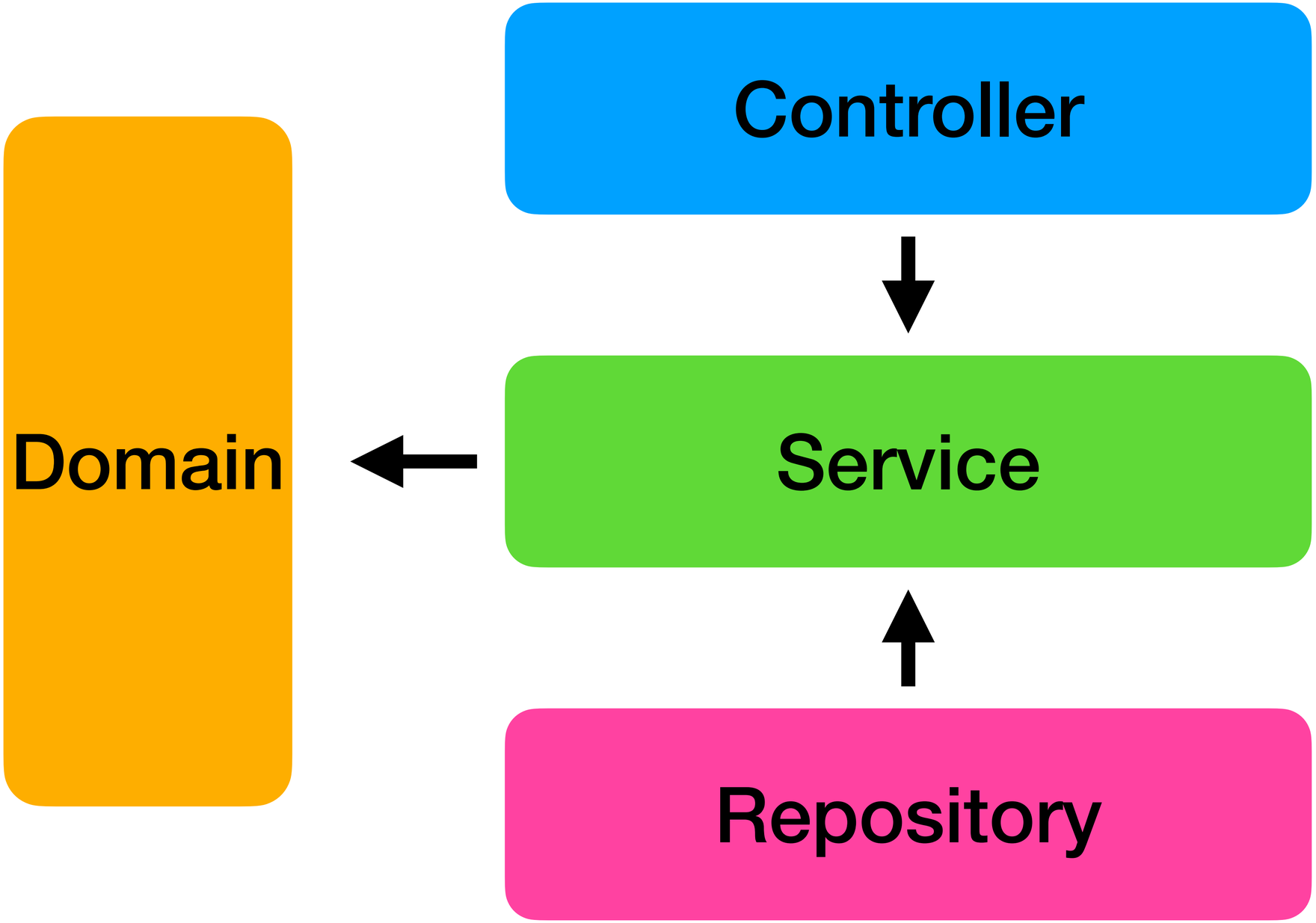
한번 더 개선해보자

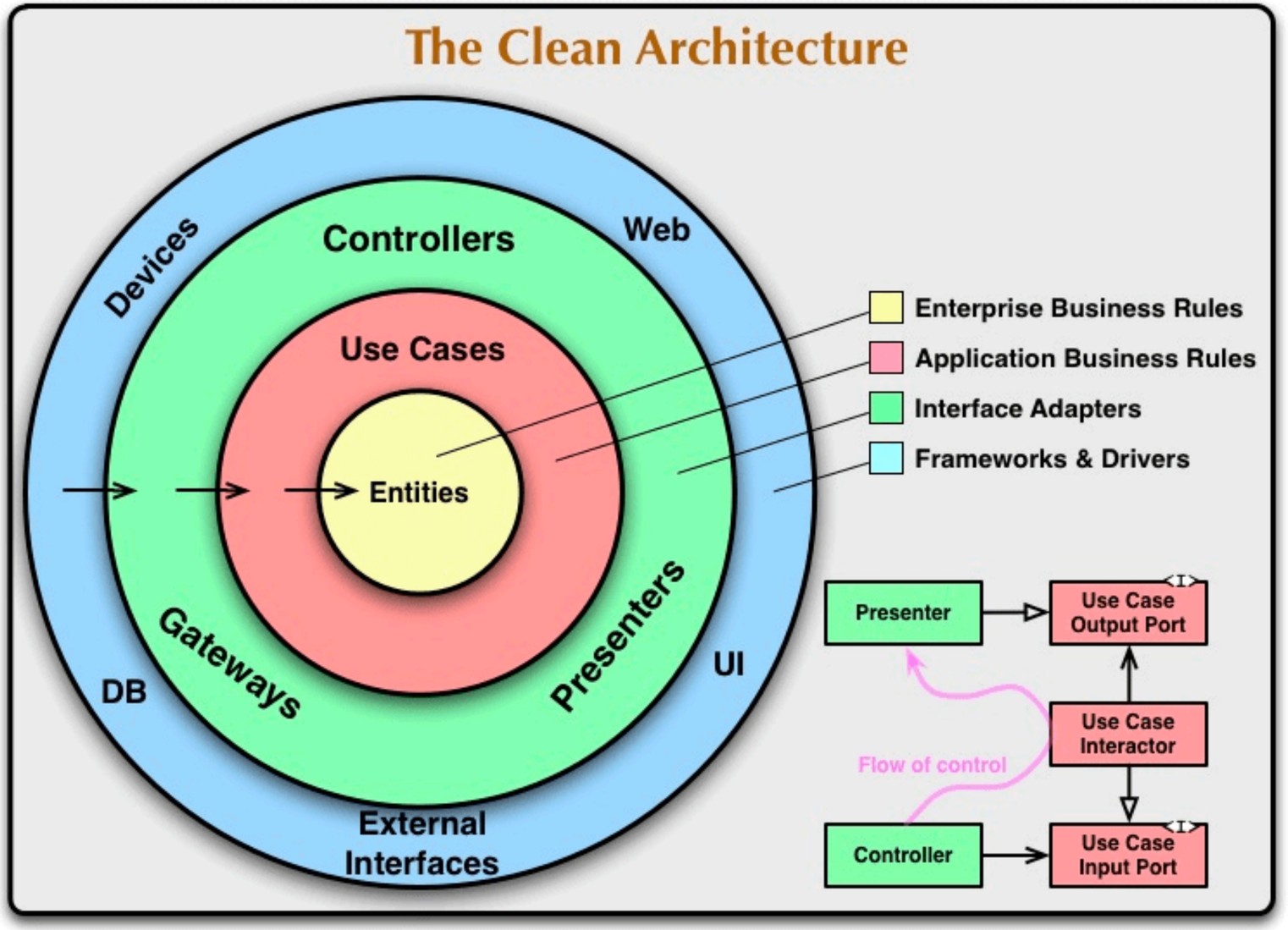
- service 계층이 repository의 entity를 중심으로 돌아가는게 맞을까?


Controller
Service
Repository

과연 우리의 도메인 로직은 어디로 가야할까?


Domain
Controller
Service
Repository

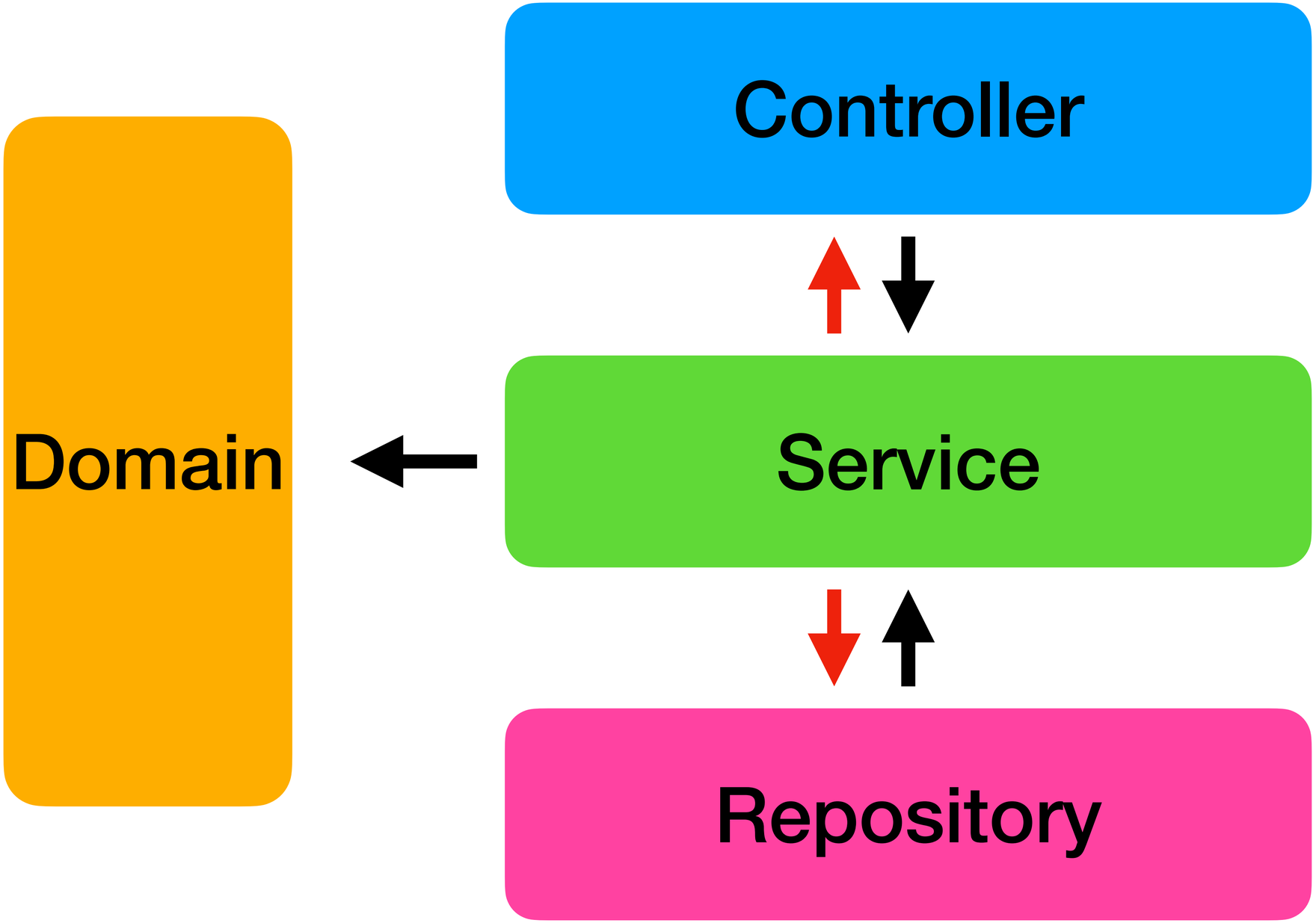

Controller
Domain
Service
Repository

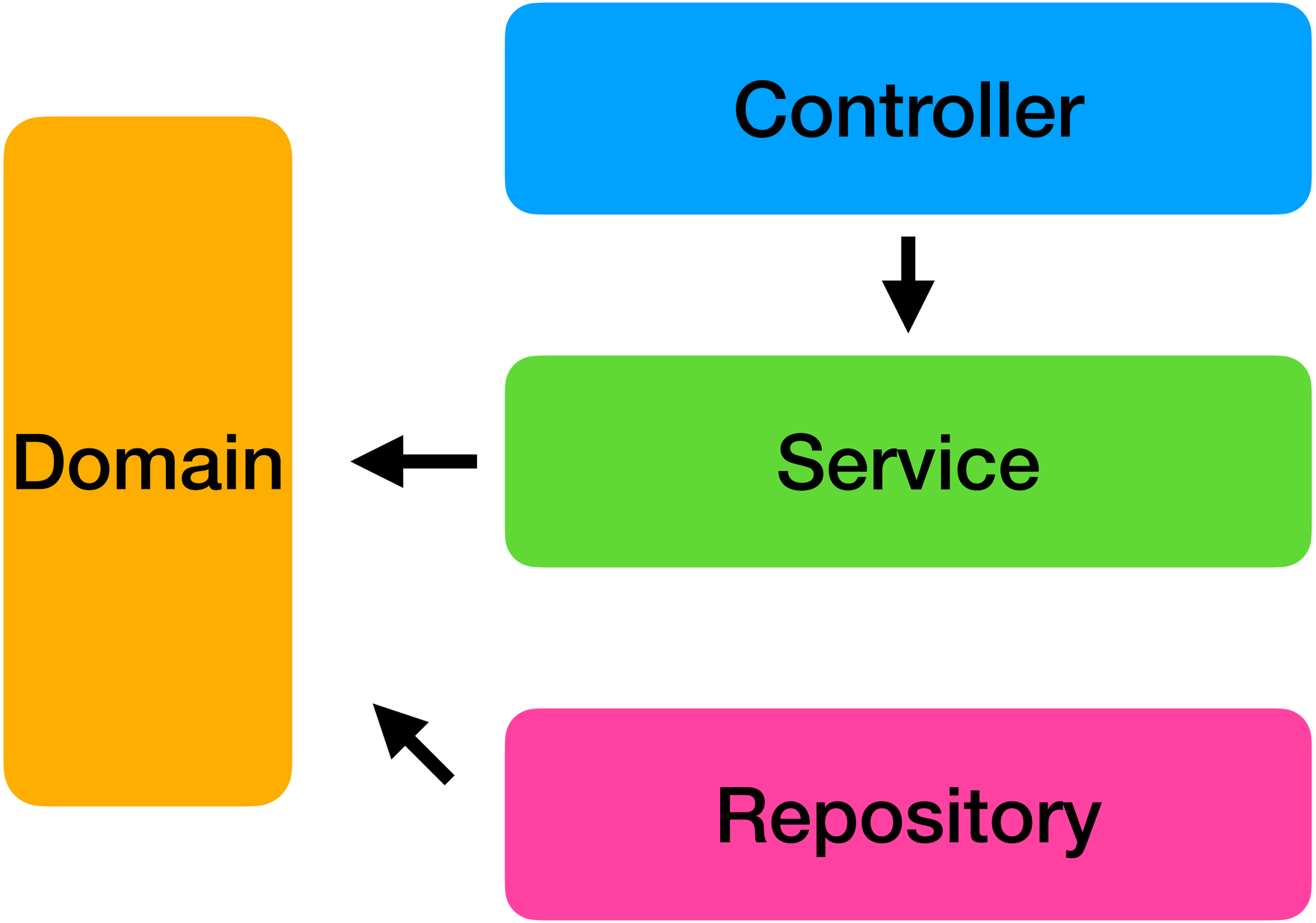


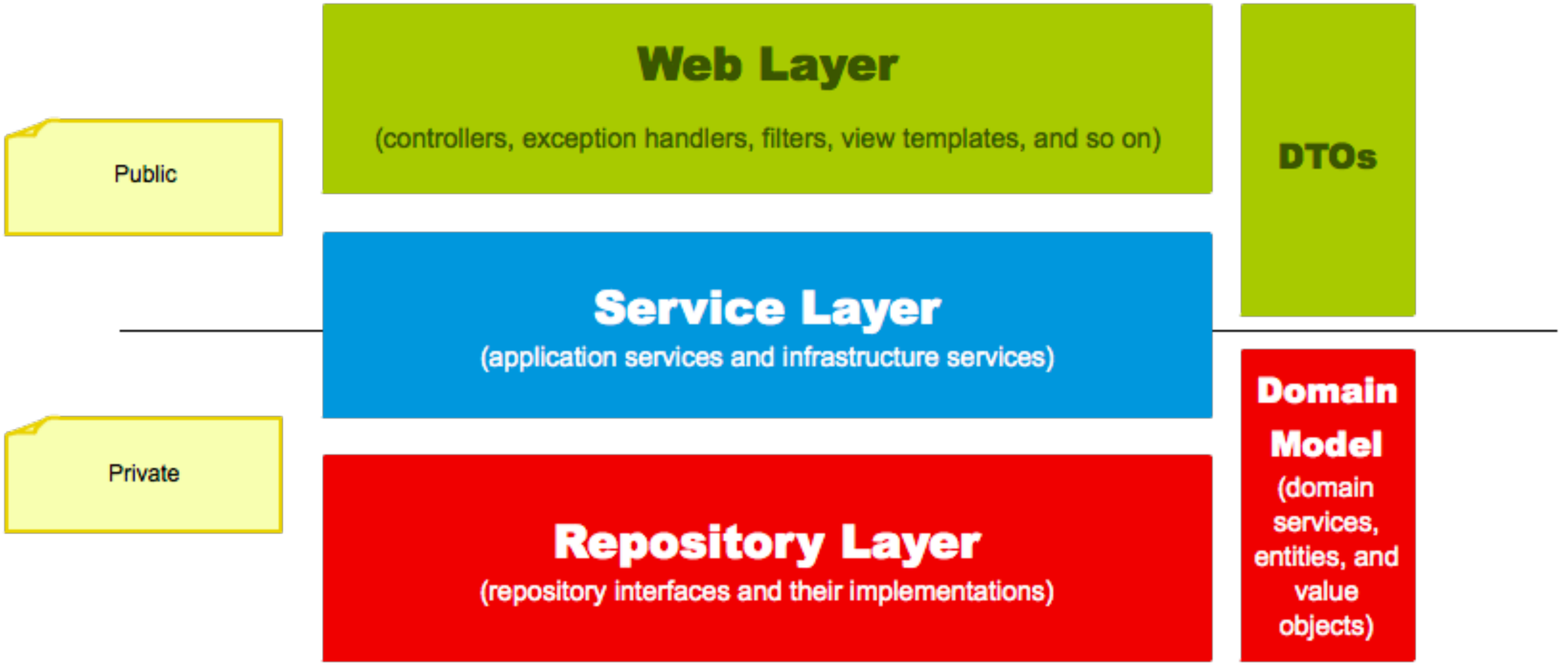
https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html

도메인 패키지가 생기면?

- 도메인 주도 개발을 할 수 있다.
- OOP를 바탕으로 개발을 할 수 있다.
- 단위 테스트에 대해서 경계가 명확해진다.


Controller
Domain
Service
Repository


Controller
Domain
Service
Repository

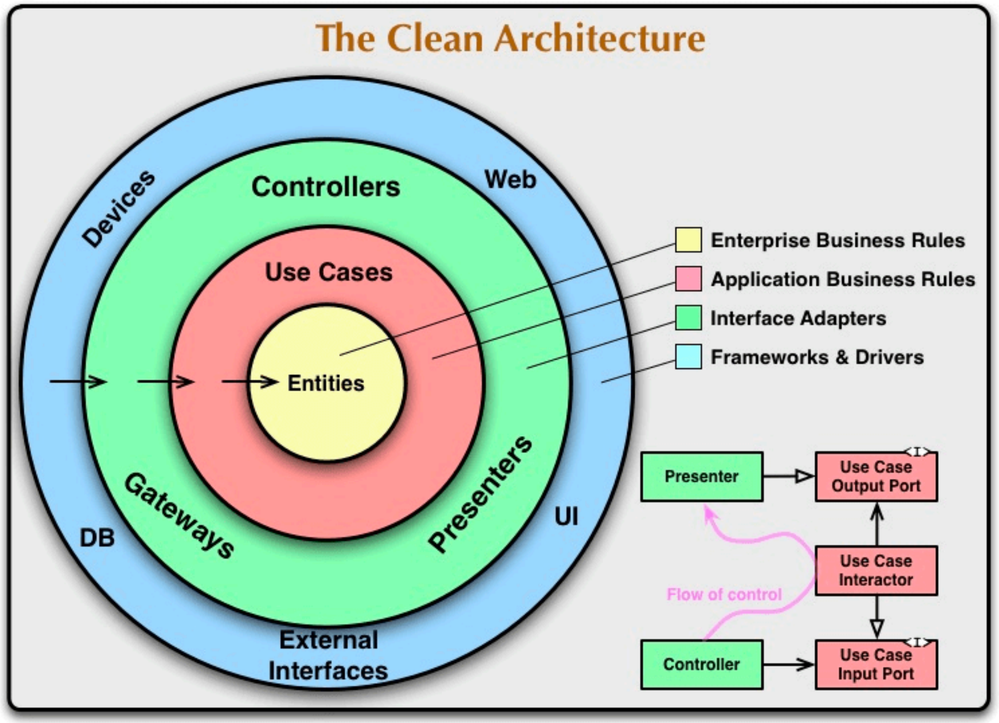


https://www.petrikainulainen.net/software-development/design/understanding-spring-web-application-architecture-the-classic-way/

다른 아키텍처를 학습해보자.

- 레이어드 아키텍처의 장점과 단점을 알고 개선까지 해보았다.
- 레이어드 아키텍처를 보완하기 위하여 나온 아키텍처를 학습해보자.

Clean Architecture



https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html

유스케이스란?

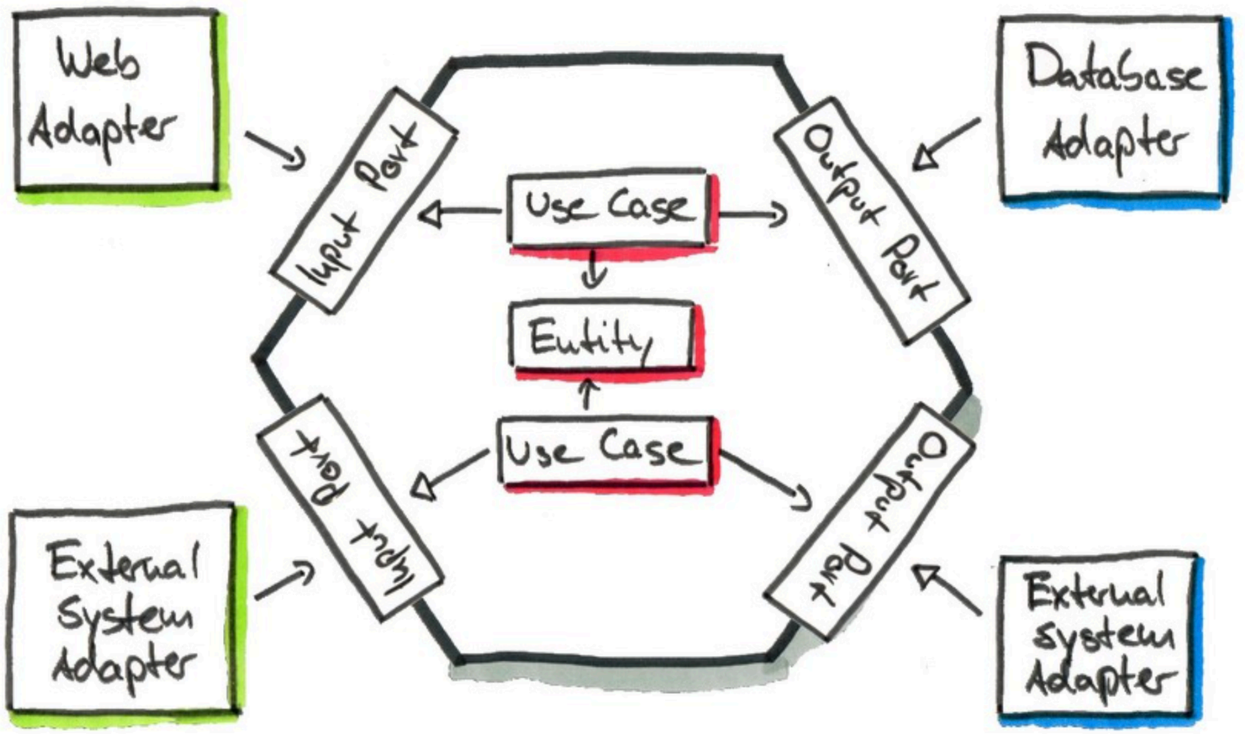
- 비즈니스를 달성하기 위한 시나리오

  ex) 자판기에서 콜라를 주문한다.

- 유스케이스의 목적은 도메인의 활동을 통하여 달성시킨다.

- 유스케이스가 변경이 되어도 컨트롤러와 레포지토리가 변경되면 안된다.

Hexagonal Architecture

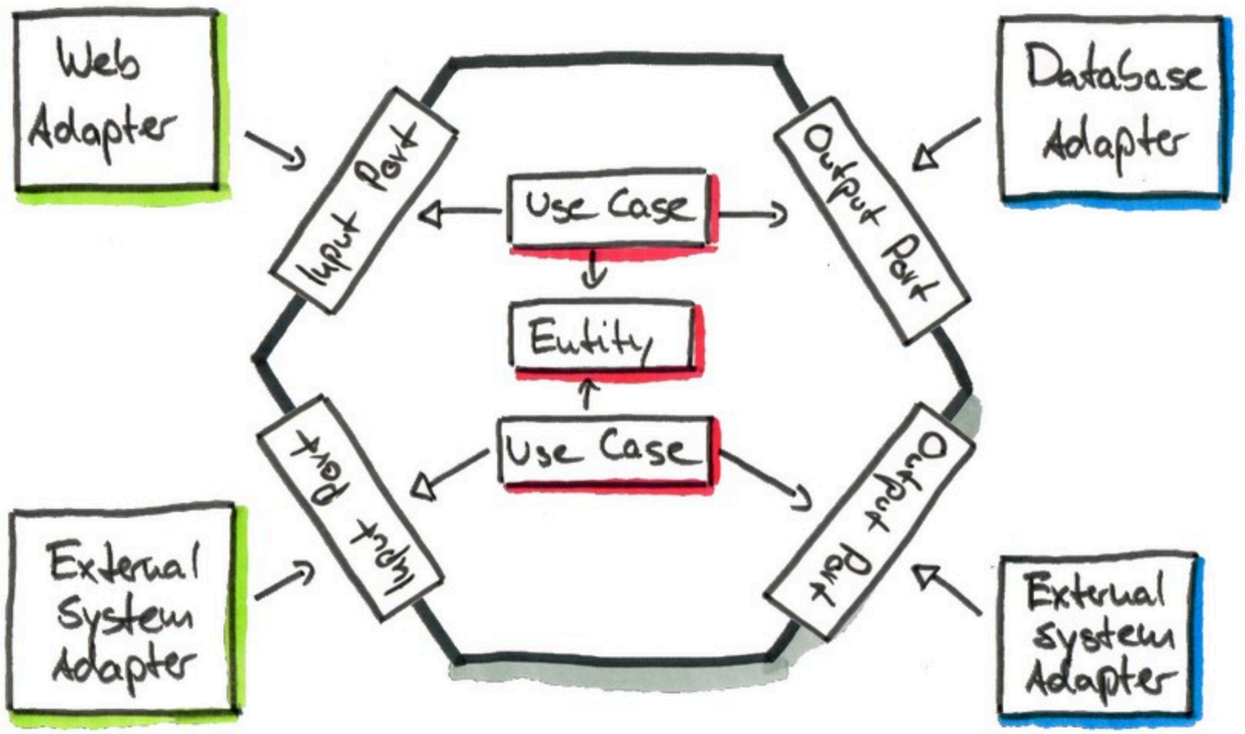


https://speakerdeck.com/thombergs/o-2019?slide=29

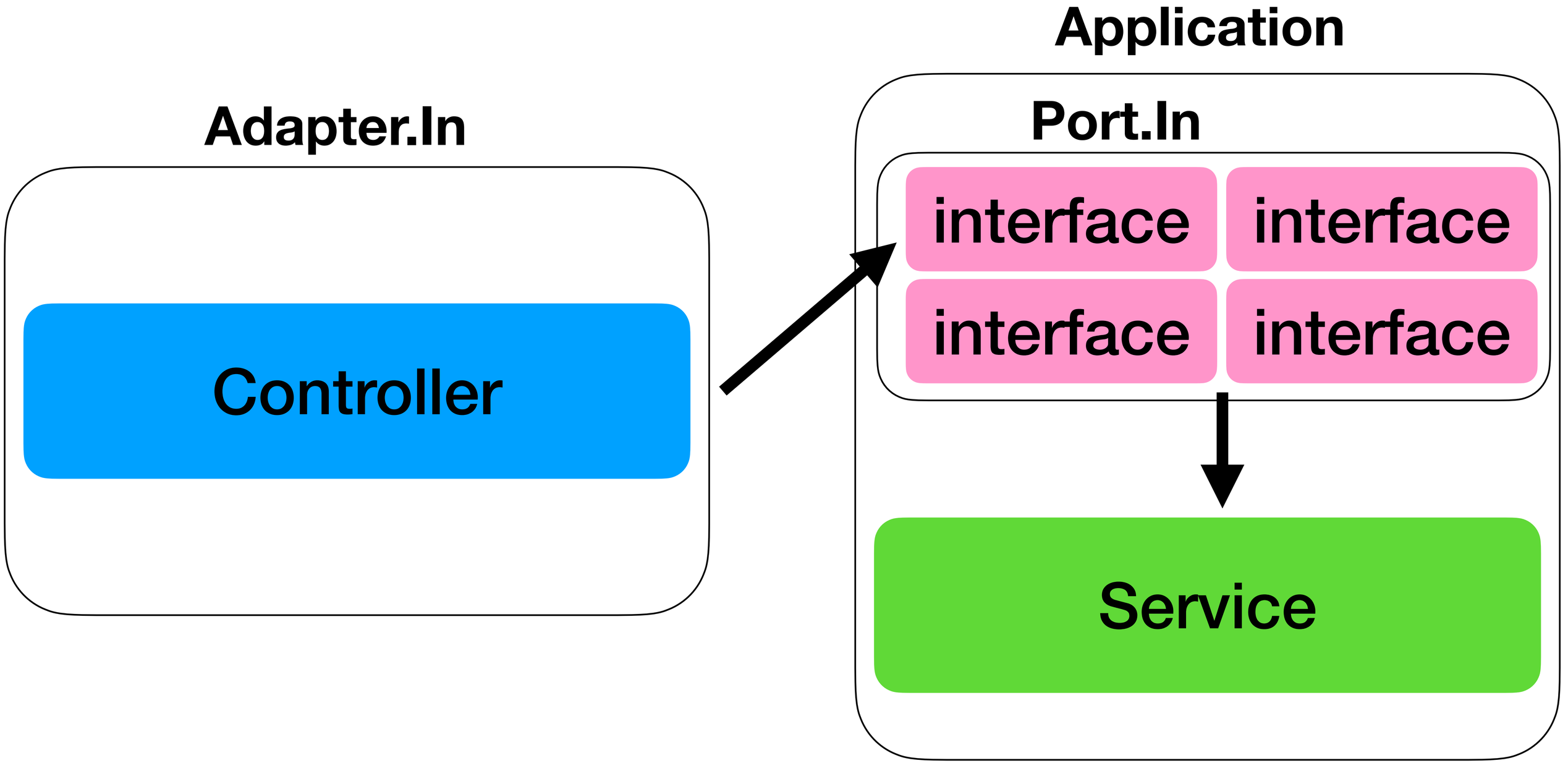
포트와 어댑터란?

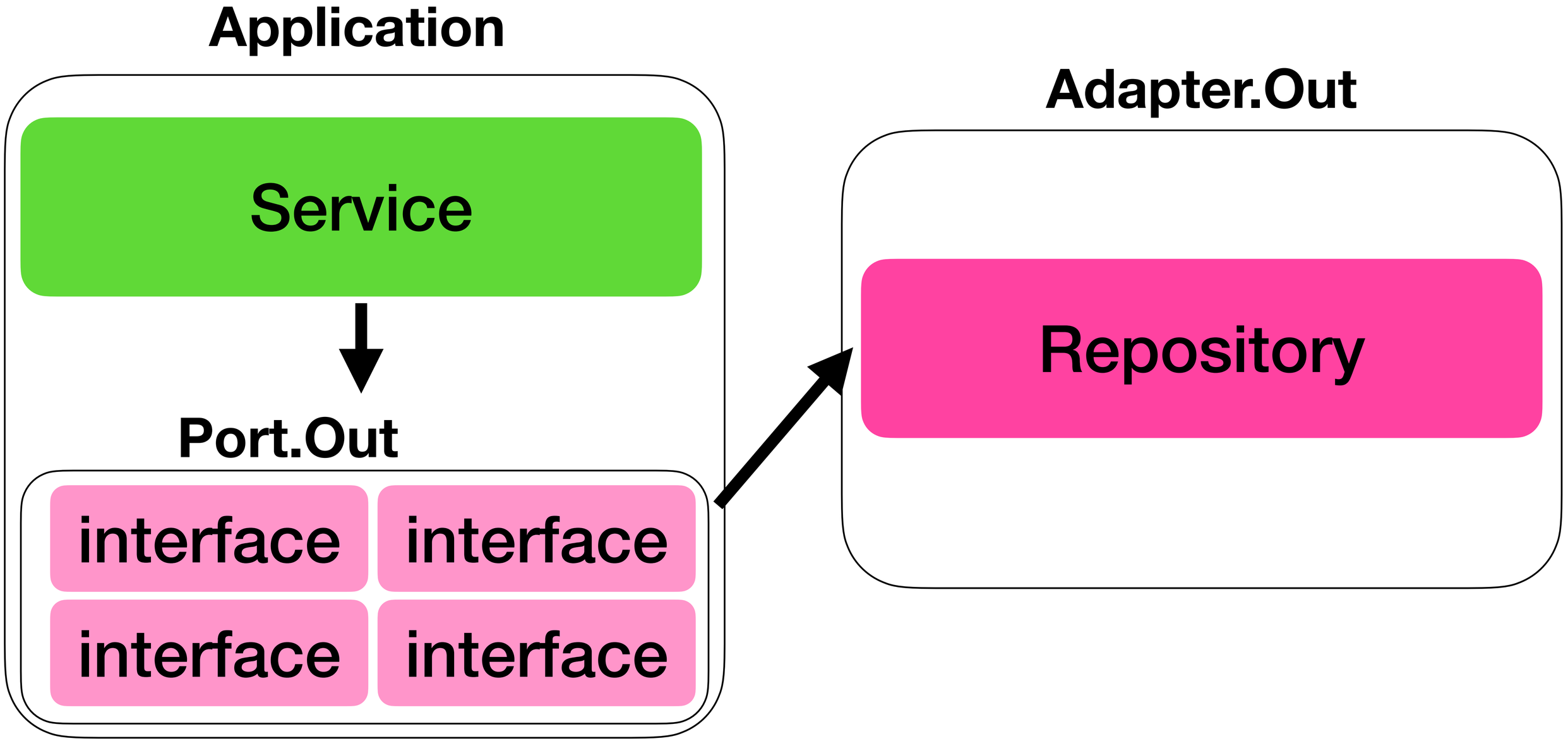
- 포트 : 서비스 영역에 대해서 외부 계층에 신경쓰지 않고 서비스 영역을 구현하기 위한 계층
- 어댑터 : 포트에 대한 직접적인 구현체 역할을 한다.
- 포트와 어댑터에 대해 서비스 영역(use case)은 DIP를 지켜줘야한다.
- 대표적으로 웹 어댑터, 영속성 어댑터가 있다.
- 목적 : 모든 기술적 관심사로부터 비즈니스 로직을 분리

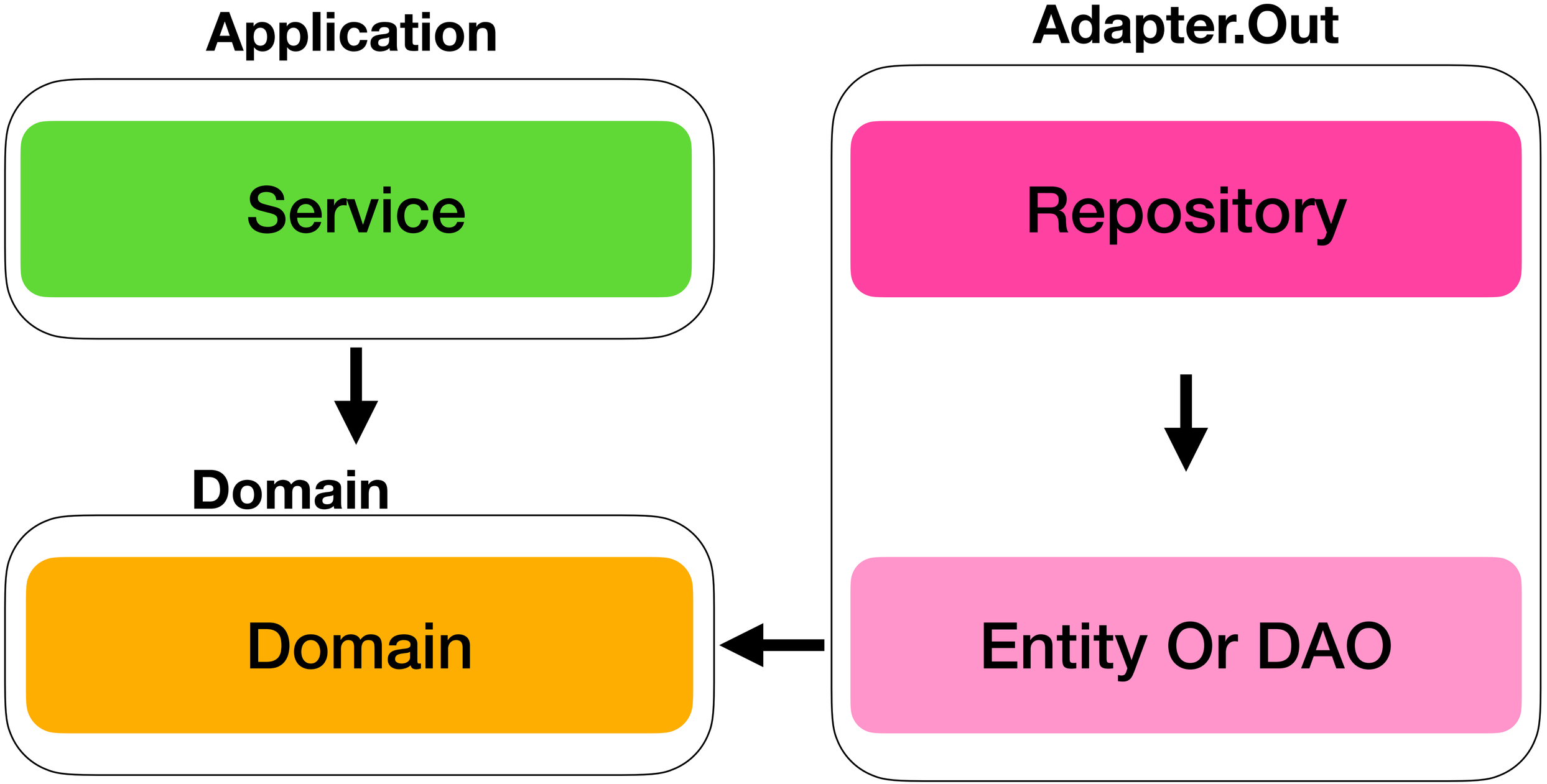
Hexagonal Architecture



https://speakerdeck.com/thombergs/o-2019?slide=29


Application
Adapter.In
Port.In
interface
interface
interface
interface
Controller
Service


Application
Service
Port.Out
interface
interface
interface
interface
Adapter.Out
Repository


Application
Service
Domain
Domain
Adapter.Out
Repository
Entity Or DAO

다 좋지만...

많은 인터페이스가 생긴다.
컨트롤러 -> 서비스 -> 레포지토리를 항상 횡단해야한다.
테스트 시에 mock 객체가 많아진다.
도메인과 엔터티를 분리해야한다.
도메인이 외부의 것들과 의존되어서는 안된다.

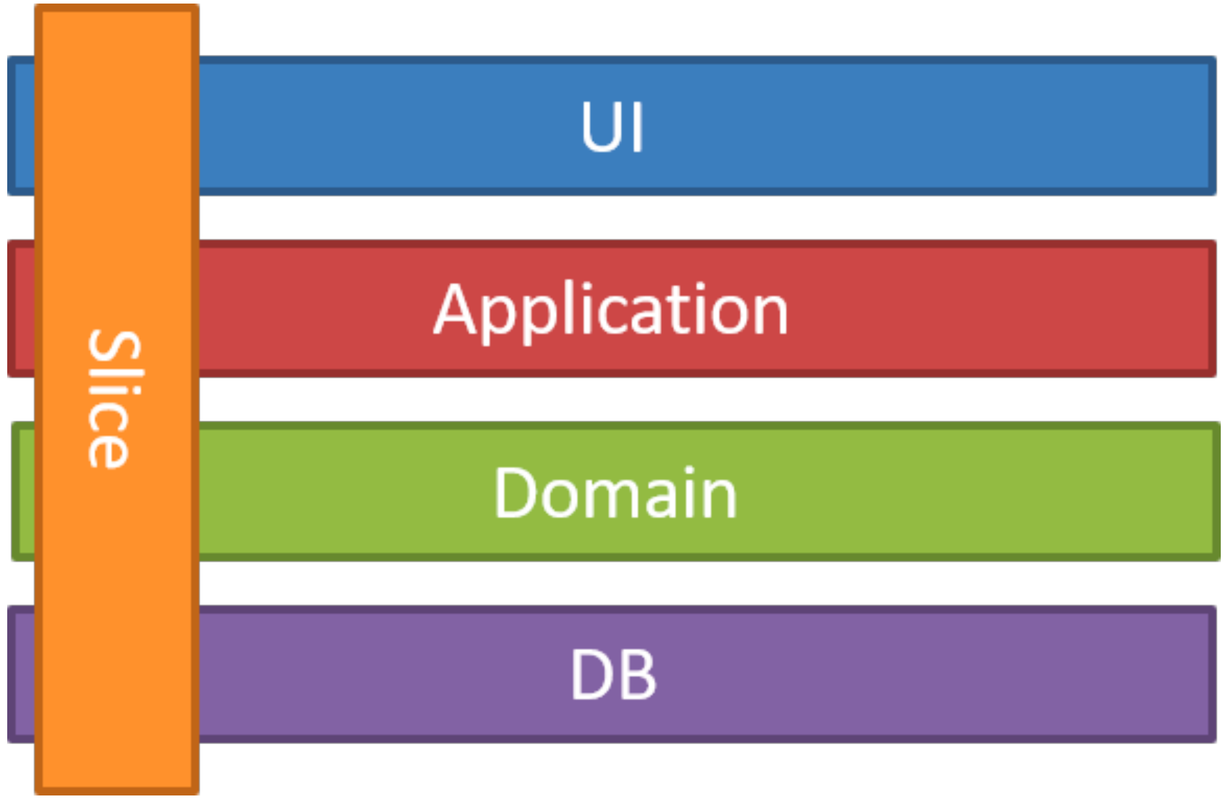
아키텍처 구현 방법을 모두 따라야할까?

도메인과 엔터티를 분리해야한다.

~~도메인과 엔터티를 분리해야한다.~~

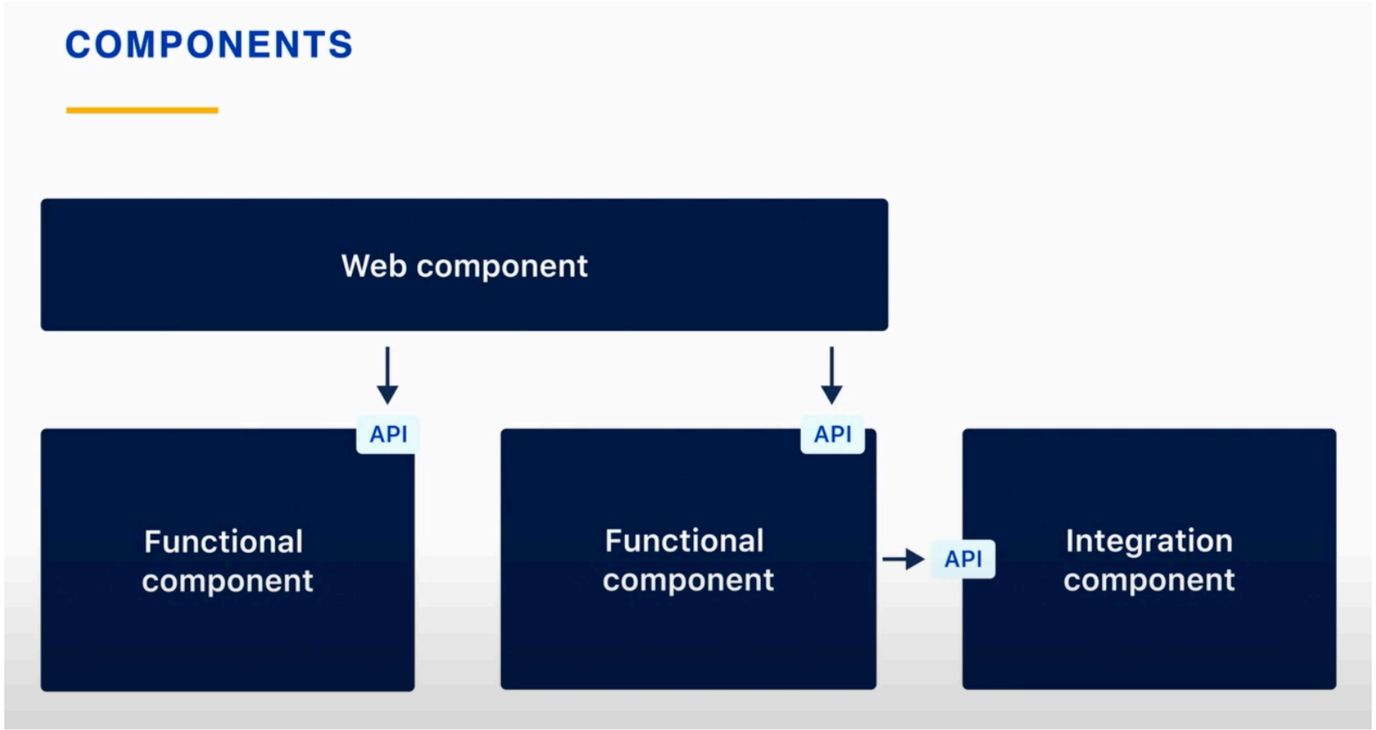
아키텍처의 룰에 대해서 모두 반영할 것인지
반영하지 않을 것인지에 대한 부분은 여러분의 **선택**입니다.

Vertical Slice Architecture



https://jimmybogard.com/vertical-slice-architecture/

Component-based Architecture



https://speakerdeck.com/thombergs/lets-build-components-not-layers

많은 아키텍처가 있지만,

- 서비스를 성장 시킬 수 있는 아키텍처
- 유지보수를 잘 할 수 있는 아키텍처
- 생산성을 위한 아키텍처
- 대화를 할 수 있는 아키텍처

는 과연 무엇일까요?

아하! 모먼트

SI, 솔루션, 서비스 어떤 곳에서 일해볼래?