

KHOA CÔNG NGHỆ THÔNG TIN
TRƯỜNG ĐH KHOA HỌC TỰ NHIÊN - TPHCM



CLEAN ARCHITECTURE

NGUYỄN PHÚC THÀNH
NGUYỄN HỒNG QUÂN

TẠI SAO CHÚNG TA CẦN “ARCHITECTURE” ?

❓ TẠI SAO PHẦN MỀM CÀNG VỀ SAU CÀNG KHÓ MỞ RỘNG?

- ✅ BAN ĐẦU PHẦN MỀM CHẠY RẤT ỔN
- ⚡ NHƯNG CÀNG VỀ SAU:
 - KHÓ BẢO TRÌ
 - KHÓ THÊM TÍNH NĂNG MỚI
 - SỬA MỘT CHỖ DỄ “VỠ” CHỖ KHÁC
 - KHÓ KIỂM THỬ

• •



TẠI SAO CHÚNG TA CẦN “ARCHITECTURE” ?

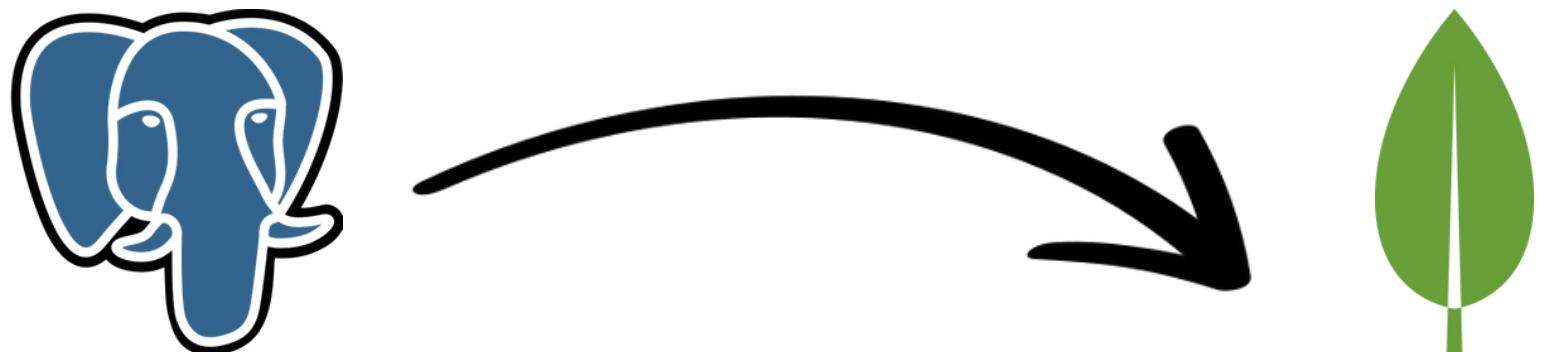
★ VÍ DỤ THỰC TẾ:

👤 KHÁCH HÀNG (HOẶC THẦY) YÊU CẦU:

+ THÊM TÍNH NĂNG MỚI

⌚ THAY ĐỔI DATABASE (MONGODB → POSTGRES)

♻️ THAY ĐỔI HÀNG LOẠT BUSINESS LOGIC



TAI SAO CHÚNG TA CẦN “ARCHITECTURE” ?

PAGE 04

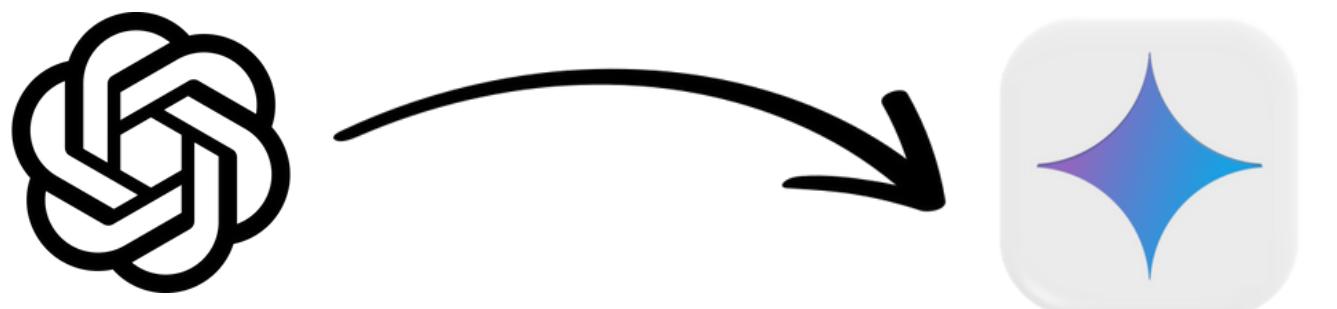
🌟 VÍ DỤ THỰC TẾ:

👤 KHÁCH HÀNG (HOẶC THẦY) YÊU CẦU:

+ THÊM TÍNH NĂNG MỚI

⌚ THAY ĐỔI DATABASE (MONGODB → POSTGRES)

♻️ THAY ĐỔI HÀNG LOẠT BUSINESS LOGIC



TẠI SAO CHÚNG TA CẦN “ARCHITECTURE” ?

PAGE 05

★ HẬU QUẢ:

- 🔧 SỬA NHIỀU CHỖ TRONG ỨNG DỤNG
- ✖ LOGIC NẰM RẢI RÁC, DÍNH CHẶT VỚI UI HOẶC DB
- 🔴 PHẢI “ĐẬP ĐÌ XÂY LẠI” TỪNG PHẦN VÌ KIẾN TRÚC BAN ĐẦU KHÔNG LINH HOẠT
- 🐌 MẤT THỜI GIAN, DỄ PHÁT SINH BUG

• •



**“VẤN ĐỀ KHÔNG NĂM Ở TÍNH NĂNG.
VẤN ĐỀ LÀ Ở KIẾN TRÚC!”**

..



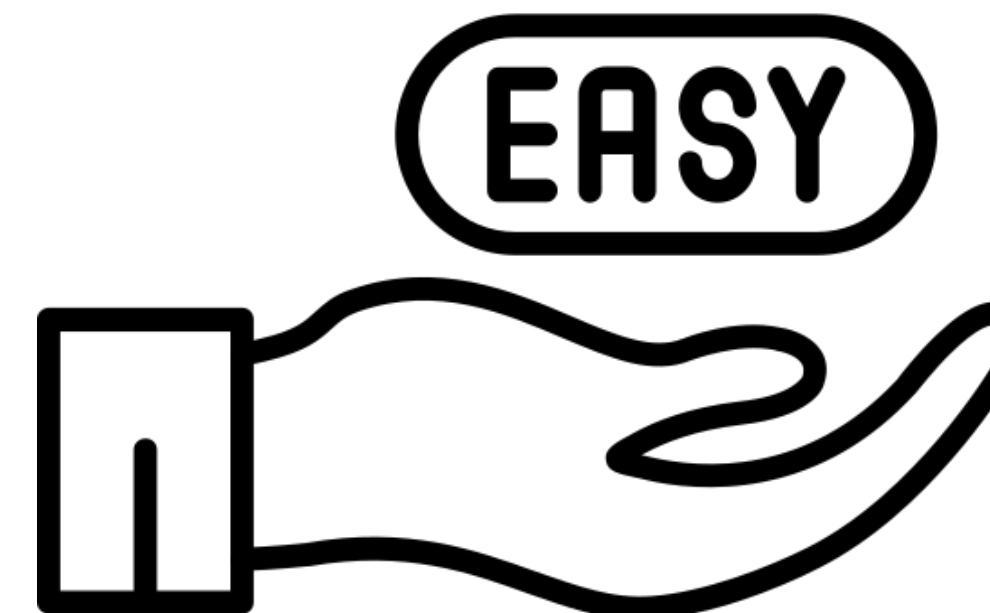
NHƯ NÀO LÀ KIẾN TRÚC “SẠCH”?

• •



NHƯ NÀO LÀ KIẾN TRÚC “SẠCH” ?

LÀ MỘT KIẾN TRÚC GIÚP CHO PHẦN MỀM CÓ 4 “DỄ”:



🔧 DỄ BẢO TRÌ

🔄 DỄ THAY ĐỔI

♻️ DỄ TÁI SỬ DỤNG

✓ DỄ KIỂM THỦ



CLEAN ARCHITECTURE

ROBERT C. MARTIN
(UNCLE BOB)
(2012)



WHAT IS CLEAN ARCHITECTURE ?

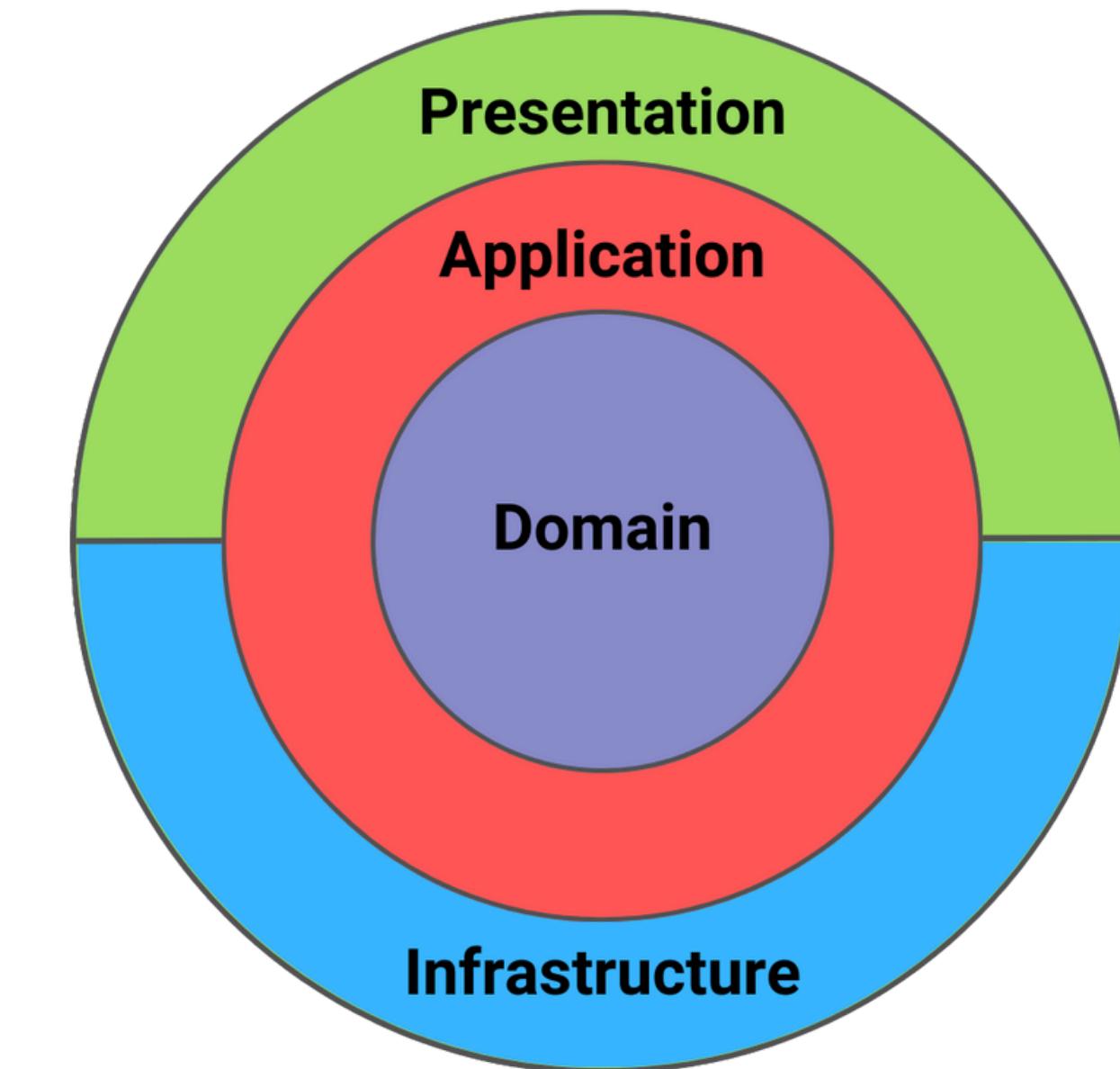
01 LOẠI KIẾN TRÚC

KIẾN TRÚC ĐA TẦNG/ ĐA LỚP(LAYERED ARCHITECTURE)

- PRESENTATION (UI/API)
- APPLICATION (USE CASE)
- DOMAIN (BUSINESS RULES)
- INFRASTRUCTURE (DB, EXTERNAL SERVICES)

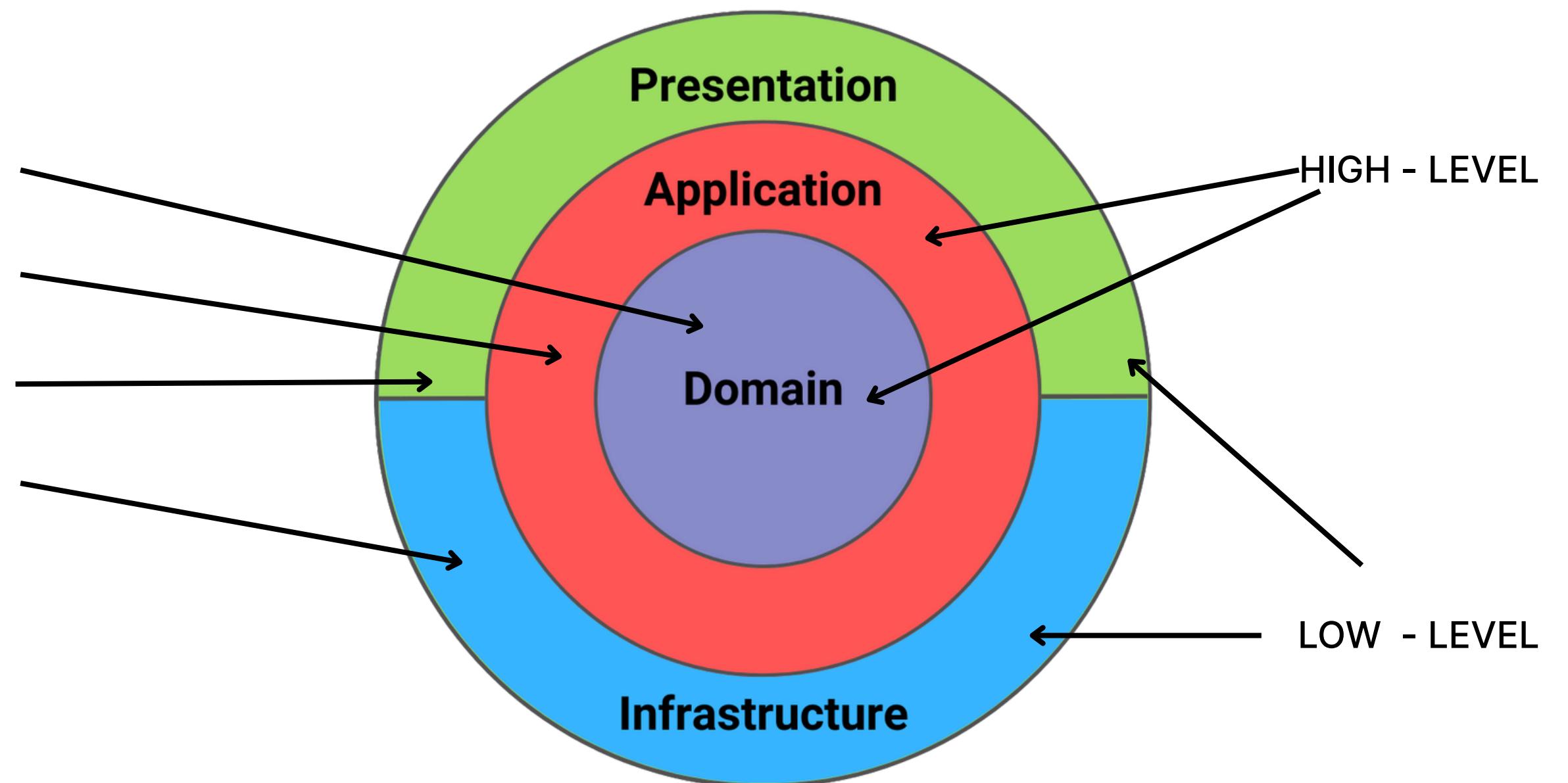
02 CÔ LẬP BUSINESS RULES:

- LOGIC NGHIỆP VỤ NẰM TRONG DOMAIN VÀ USE CASE
- KHÔNG PHỤ THUỘC VÀO UI, DB HAY FRAMEWORK
- DỄ TEST, DỄ TÁI SỬ DỤNG



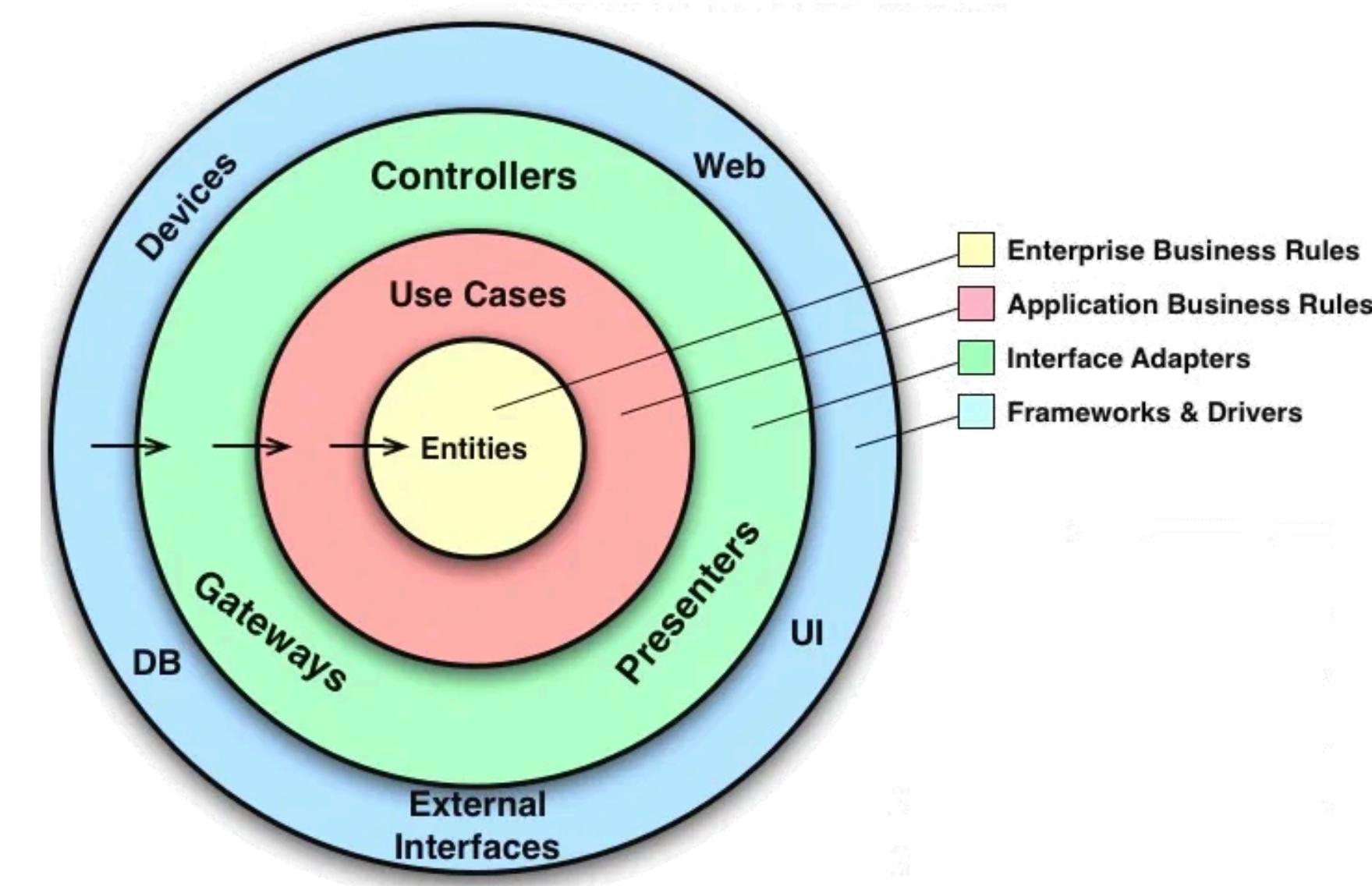
STRUCTURE

1. BUSINESS LOGIC
2. USE CASES
3. PUBLIC API
4. EXTERNAL SERVICES



DESIGN PRINCIPLES

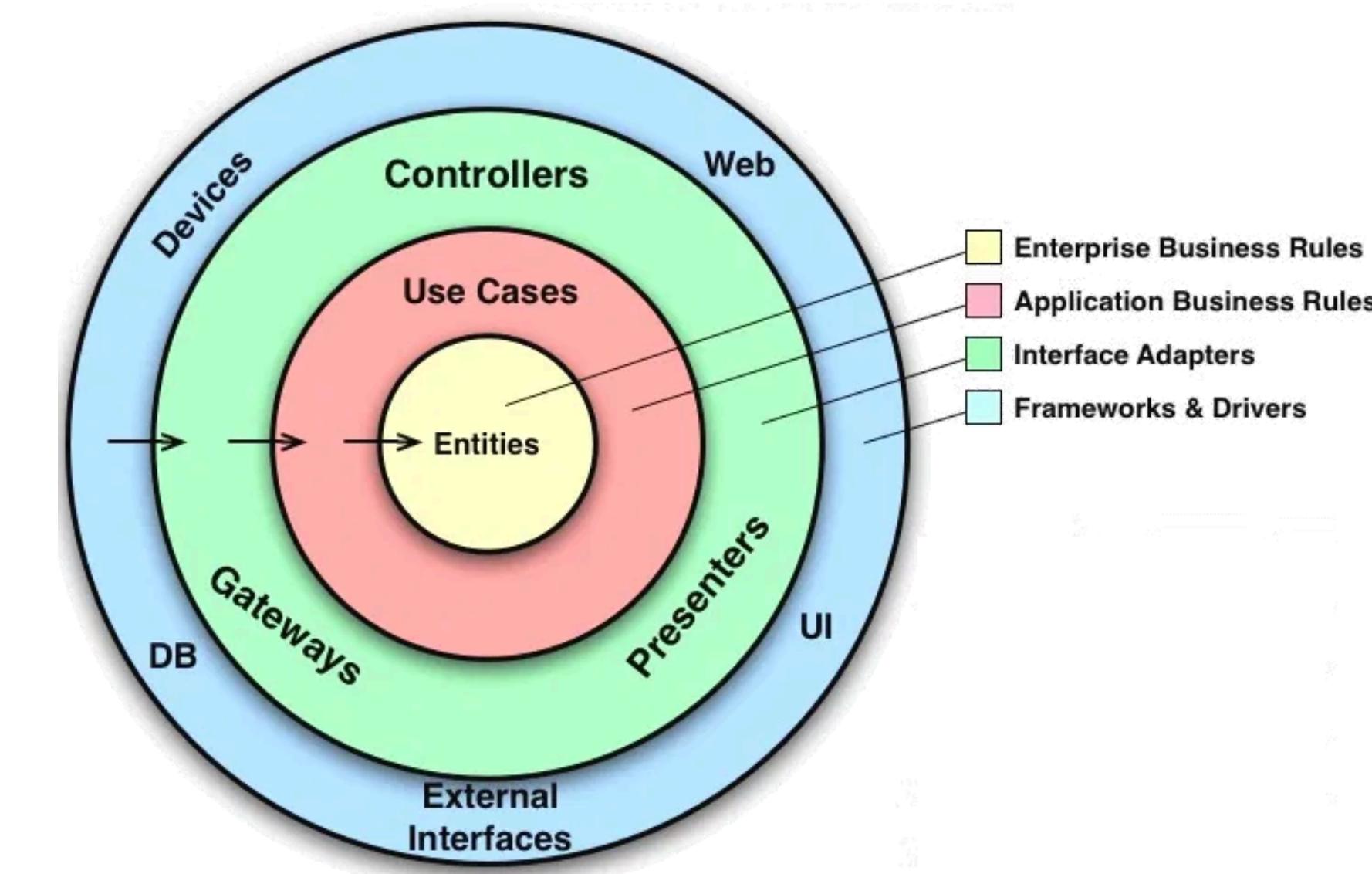
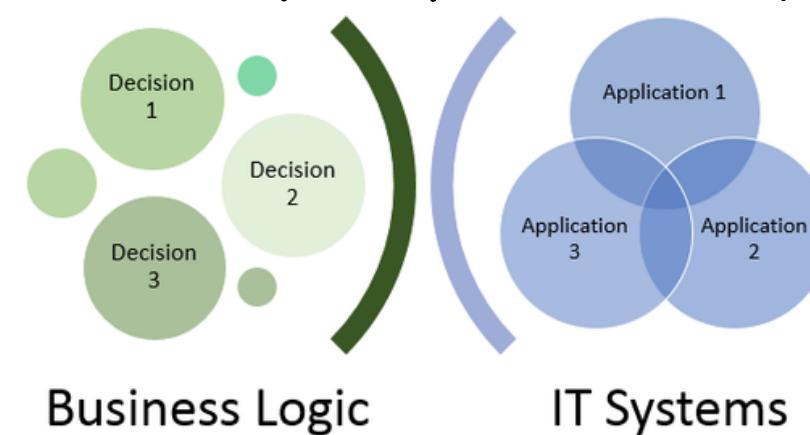
1. SEPARATION OF CONCERNS



DESIGN PRINCIPLES

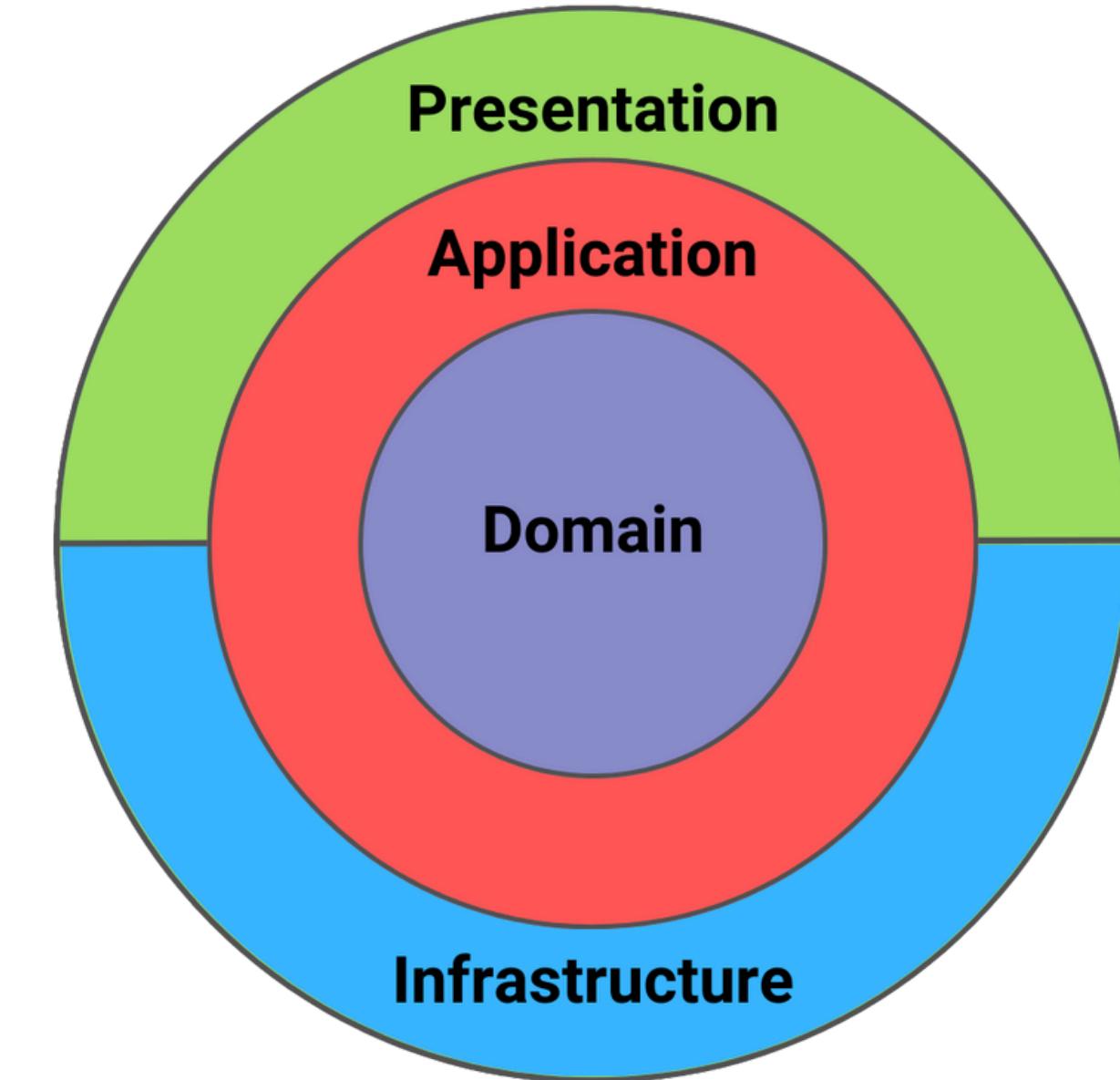
1. SEPARATION OF CONCERNS

- STRUCTURE HỆ THỐNG THEO MỘT CÁCH NÀO ĐÓ MÀ TỪNG THÀNH PHẦN (COMPONENT) ĐỀU CÓ MỘT TRÁCH NHIỆM CỤ THỂ NÀO ĐÓ.
- VÍ DỤ:
 - DOMAIN CHỈ QUAN TÂM TỚI BUSINESS LOGIC
 - APPLICATION CHỈ QUAN TÂM TỚI USECASES (CỐ GẮNG IMPLEMENT CÁC USECASE SAO CHO ĐÔC LẬP VỚI EXTERNAL SERVICES NHẤT)
 - VÍ DỤ: CHỈ QUAN TÂM TỚI VIỆC GỬI MAIL MÀ KHÔNG CẦN QUAN TÂM TỚI GỬI MAIL BẰNG DỊCH VỤ GÌ OUTLOOK, GMAIL,...



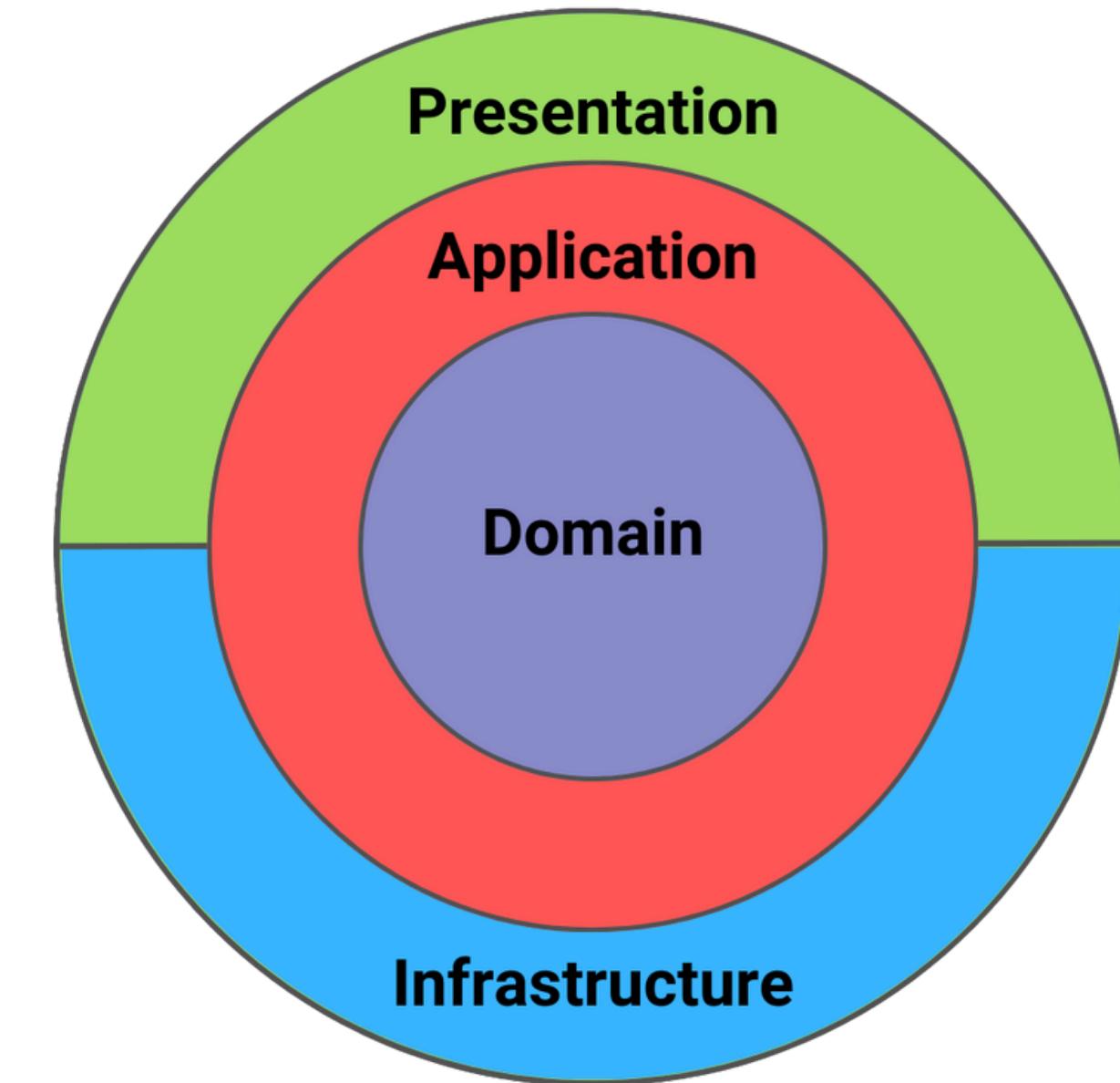
DESIGN PRINCIPLES

- 1. SEPARATION OF CONCERNS
- 2. ENCAPSULATION



DESIGN PRINCIPLES

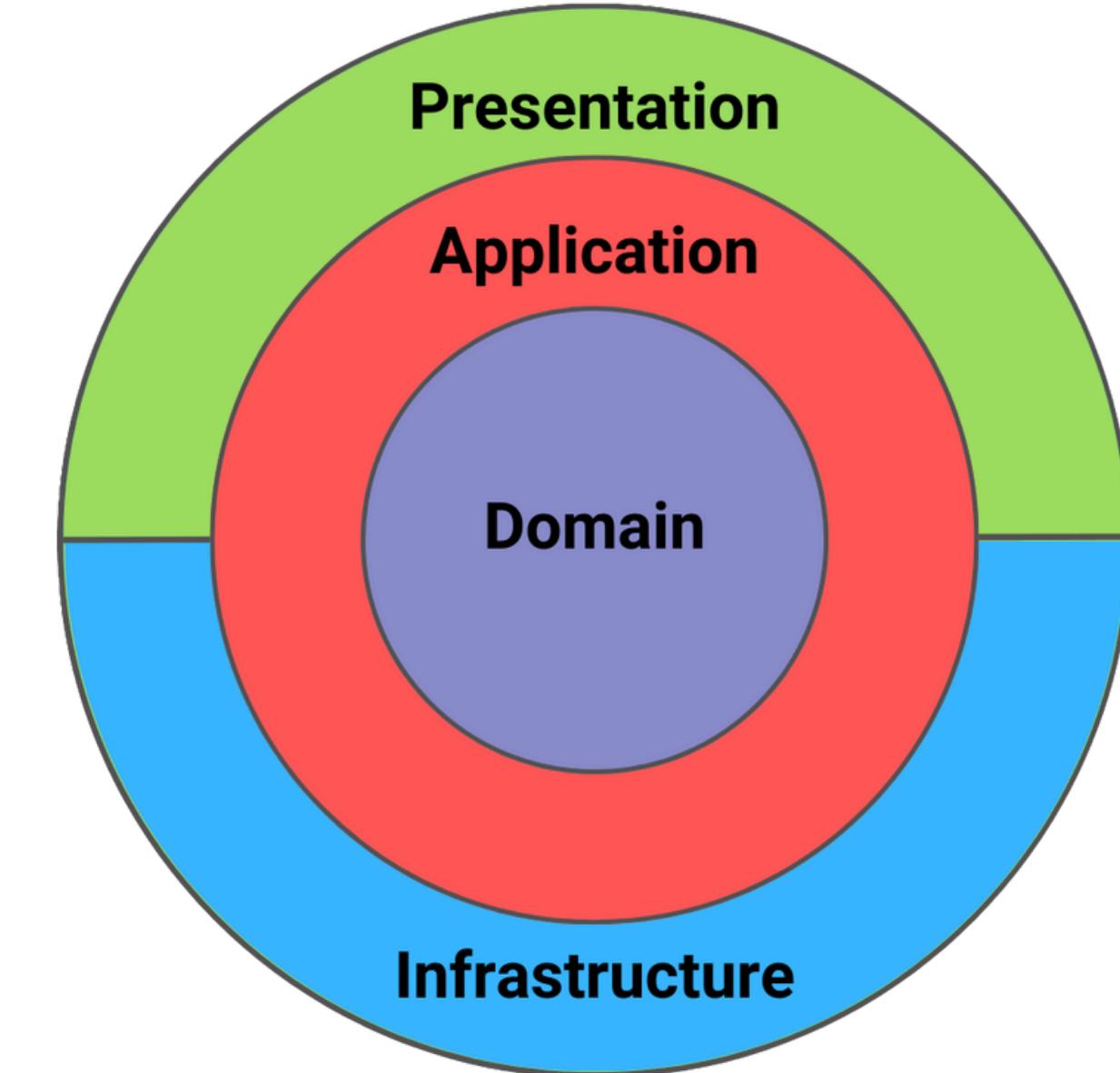
2. ENCAPSULATION



DESIGN PRINCIPLES

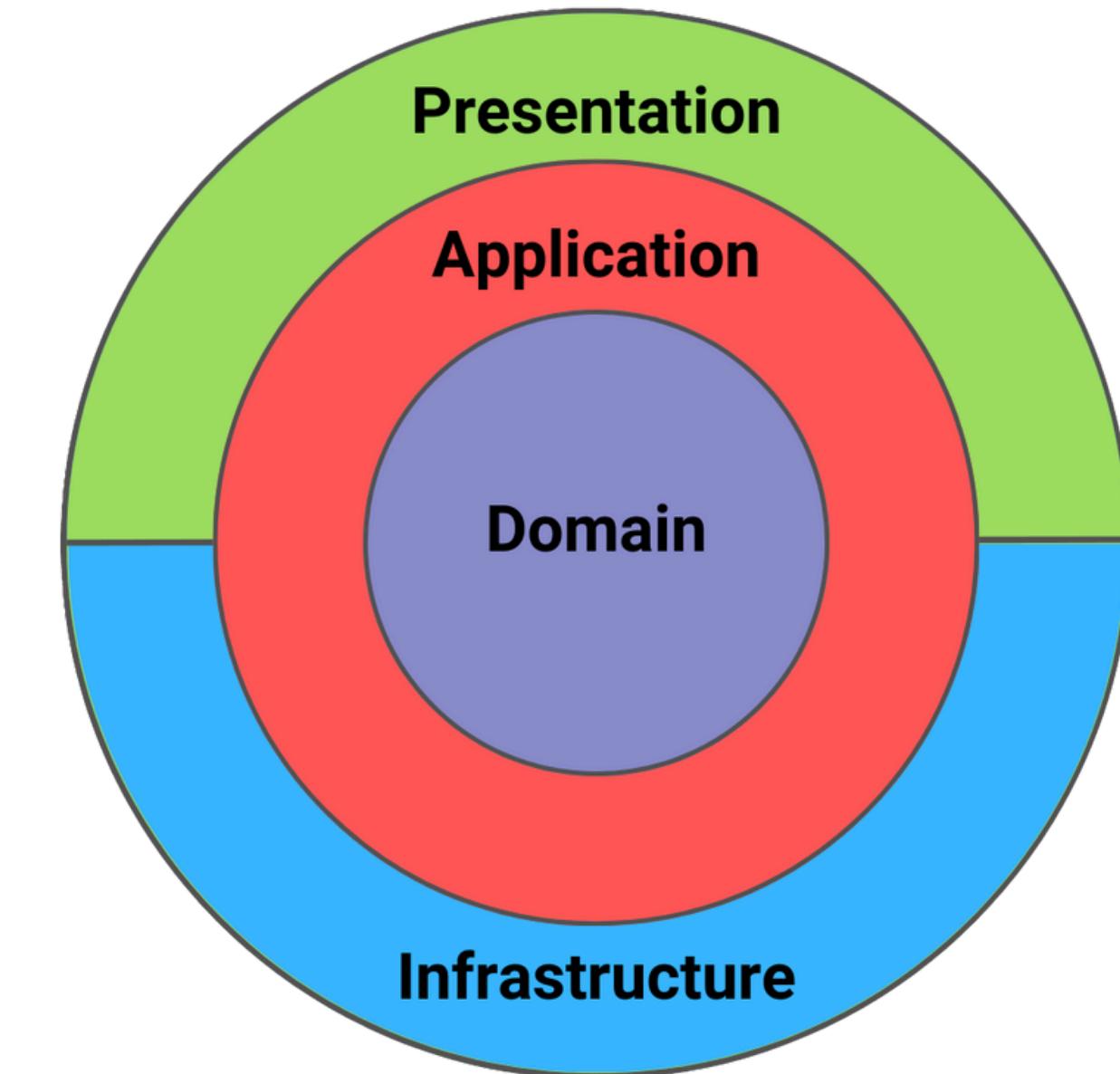
2. ENCAPSULATION

- ẨN GIẤU CHI TIẾT BÊN TRONG, CHỈ EXPOSE NHỮNG GÌ CẦN THIẾT
- TẦNG NGOÀI KHÔNG CẦN BIẾT DOMAIN (APPLICATION) XỬ LÝ CHI TIẾT RA SAO, CHỈ CẦN GỌI INTERFACE/USECASE



DESIGN PRINCIPLES

- ▣ 1. SEPARATION OF CONCERNS
- ▣ 2. ENCAPSULATION
- ▣ 3. EXPLICIT DEPENDENCY
- ▣ 4. SINGLE RESPONSIBILITY
- ▣ 5. PERSISTENCE IGNORANCE
- 🔥 6. DEPENDENCY INVERSION



DEPENDENCY INVERSION LÀ GÌ?

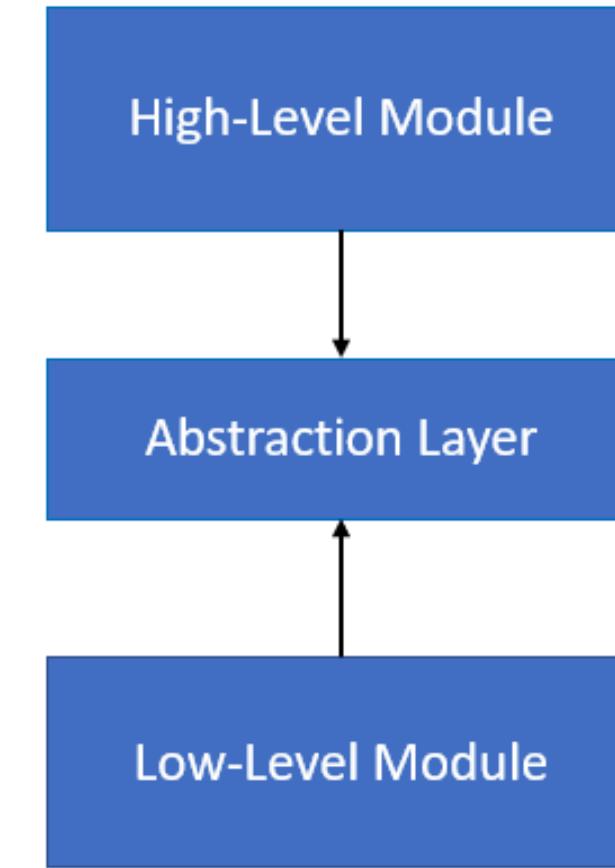
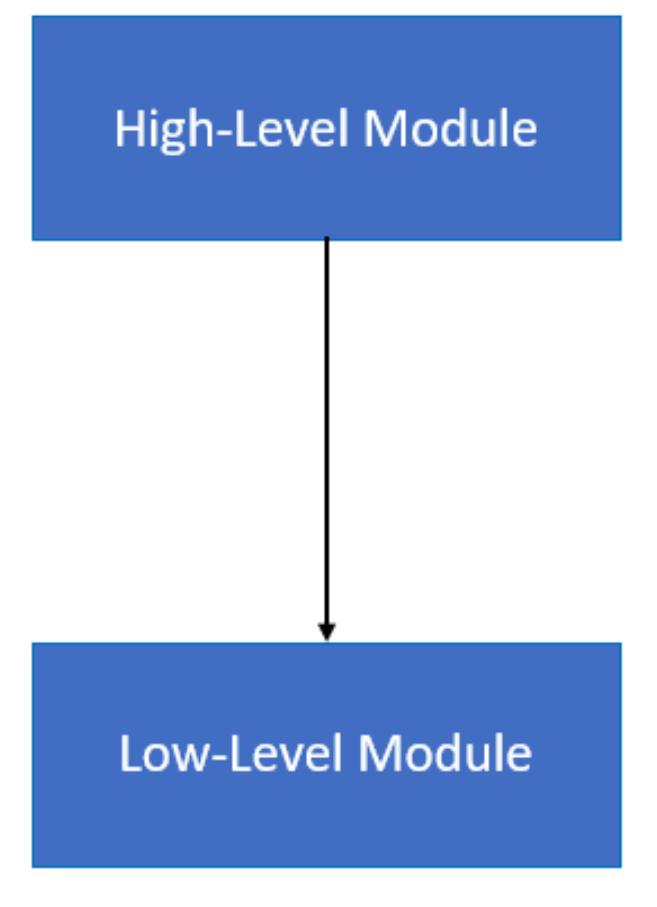
NGUYÊN LÝ CỐT LÕI CỦA CLEAN ARCHITECTURE

“HIGH-LEVEL MODULES SHOULD NOT DEPEND ON LOW-LEVEL MODULES.
BOTH SHOULD DEPEND ON ABSTRACTIONS.”



DEPENDENCY INVERSION

NGUYÊN LÝ CỐT LÕI CỦA CLEAN ARCHITECTURE



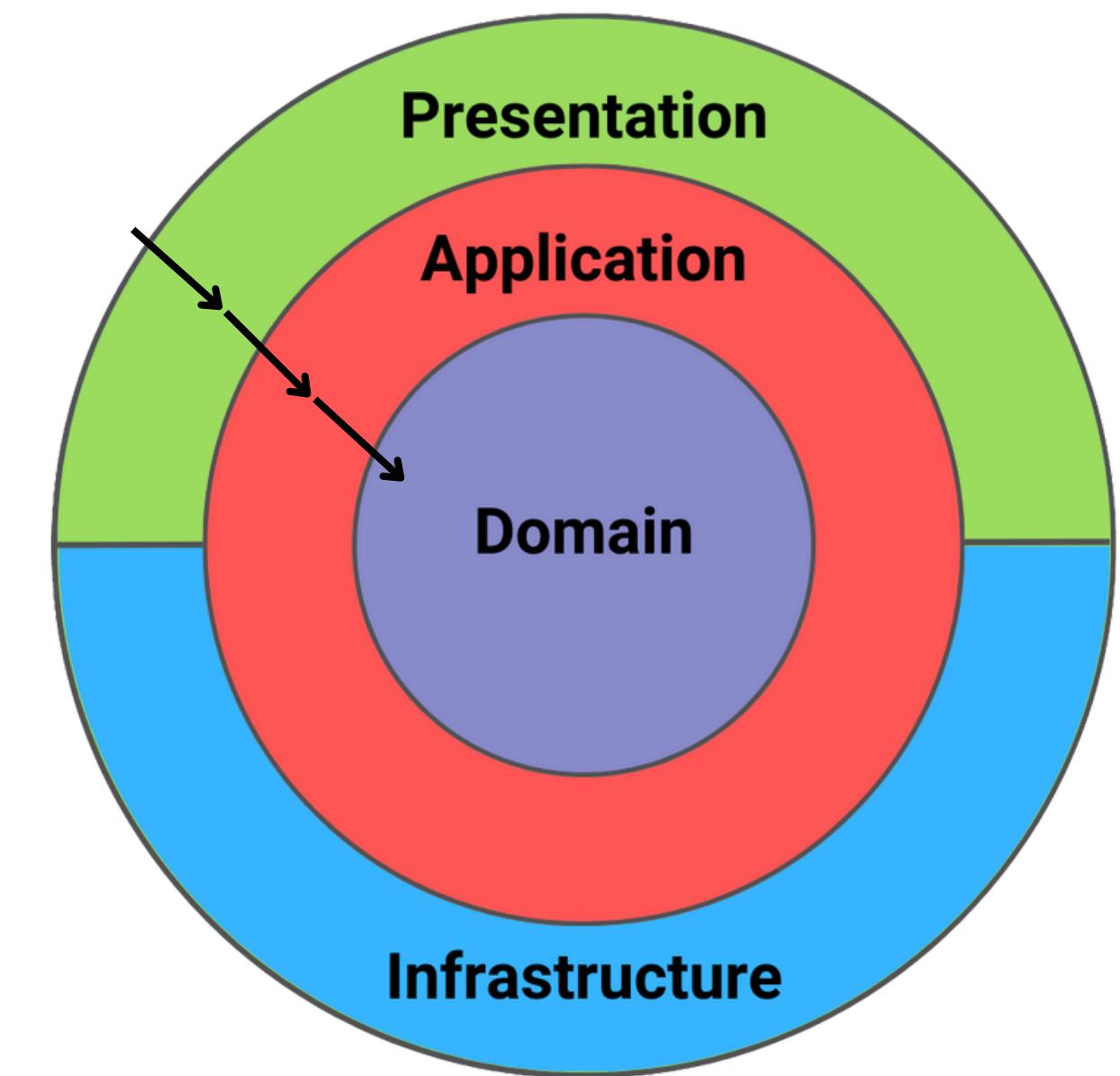
DEPENDENCY INVERSION

1. DEPENDENCIES FLOW INWARD

- CÁC LAYER Ở NGOÀI THÌ SẼ PHỤ THUỘC VÀO LAYER PHÍA TRONG
- HIGH LEVEL MODULE SẼ KHÔNG PHẢI PHỤ THUỘC (KHÔNG QUAN TÂM) TỚI CÁC LOW LEVEL MODULE

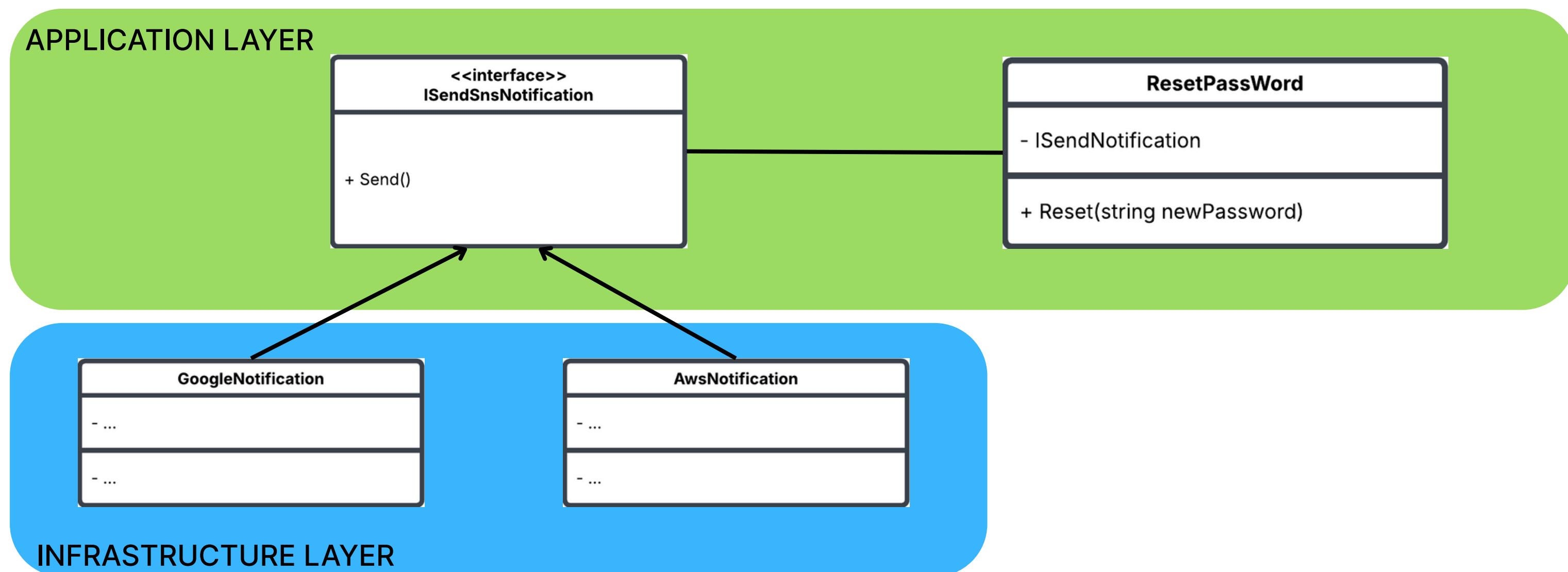
2. INNER LAYERS DEFINE INTERFACES

- APPLICATION/DOMAIN ĐỊNH NGHĨA INTERFACE
- INFRASTRUCTURE IMPLEMENT CÁC INTERFACE ĐÓ



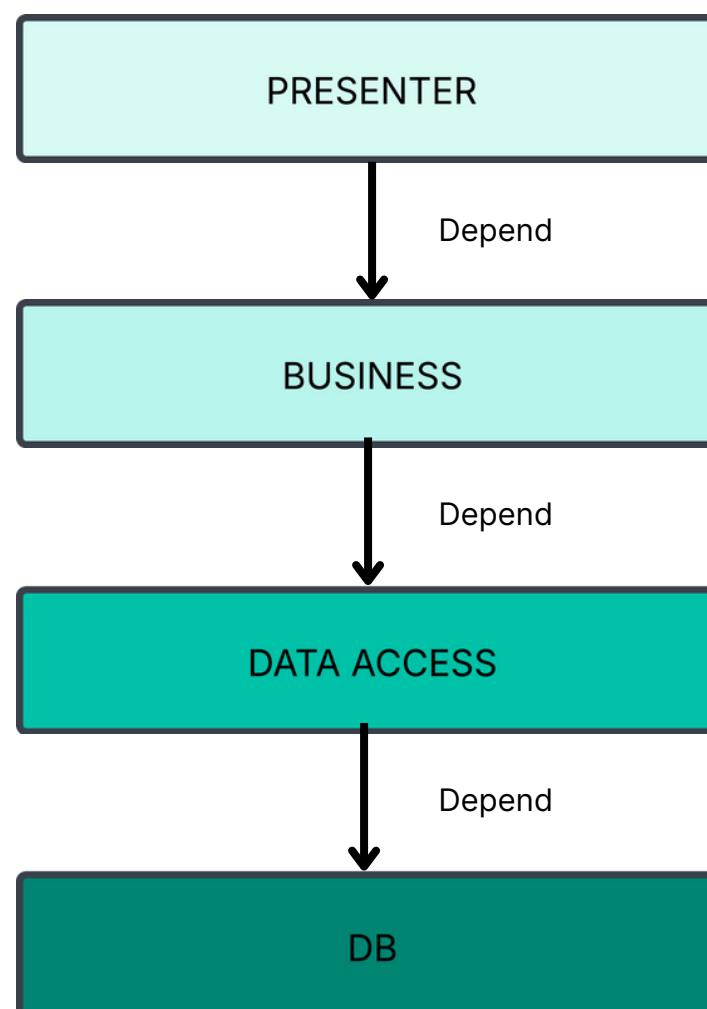
DEPENDENCY INVERSION

📌 VÍ DỤ THỰC TẾ: RESET PASSWORD CẦN GỬI SNS



CLEAN VS N-LAYER ARCHITECTURE

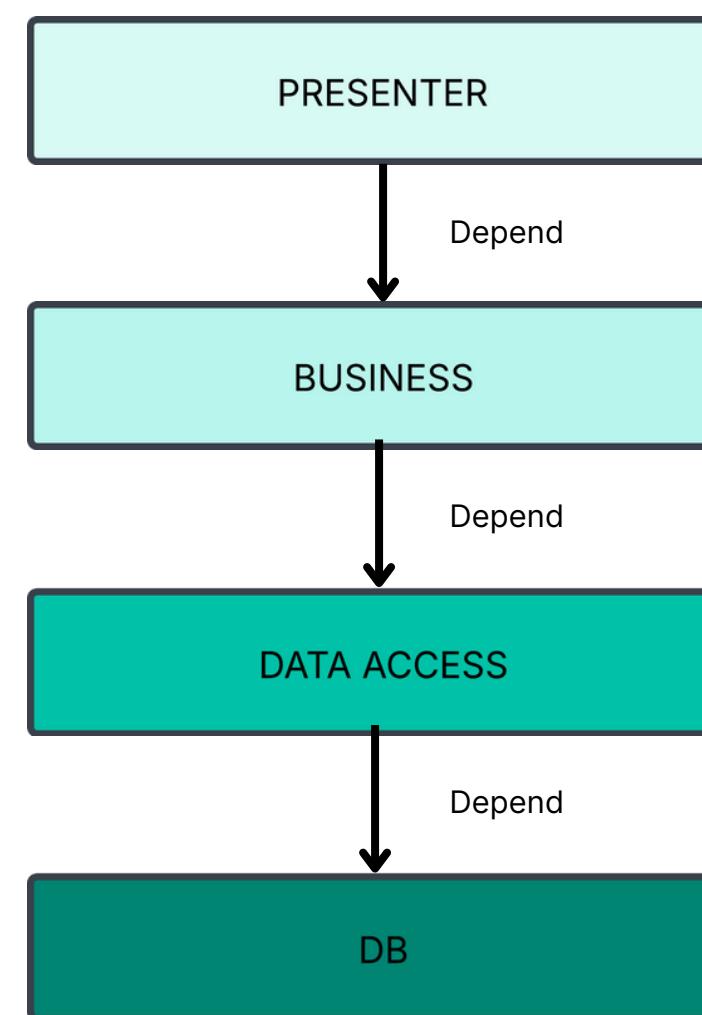
N-LAYER ARCHITECTURE



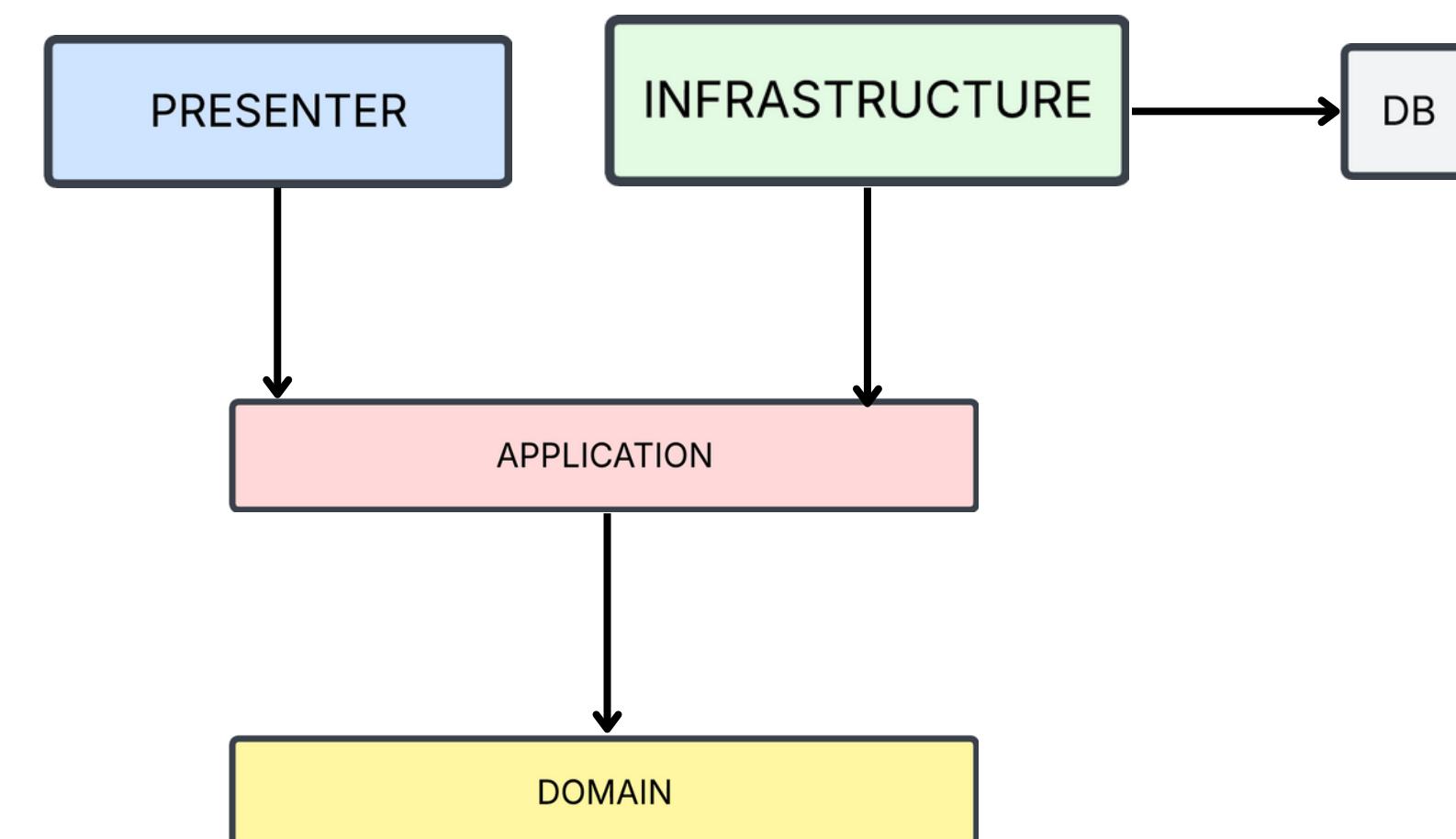
EVERYTHING DEPEND ON THE DATABASE

1. KHÓ VIẾT UNIT TEST CHO CÁC BUSINESS LOGIC
2. LOGIC BỊ “RẢI RÁC” VÀ GẮN CHẶT VÀO TẦNG DƯỚI (DATA ACCESS HAY PRESENTER)
2. KHÓ THAY ĐỔI DB VÌ CÁC LOGIC KHÁC BỊ PHỤ THUỘC VÀO DB.

CLEAN VS N-LAYER ARCHITECTURE

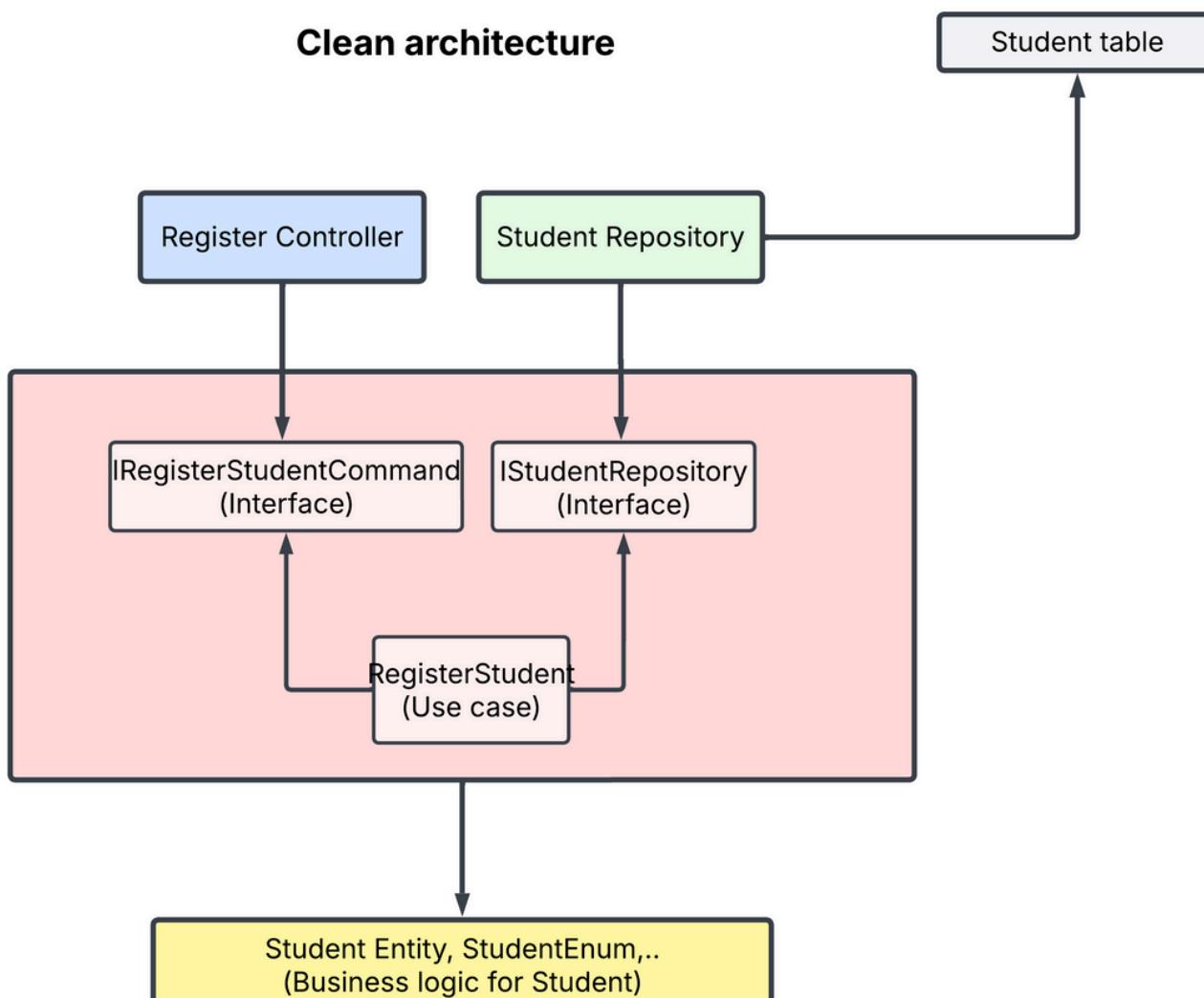


EVERYTHING DEPEND ON THE DATABASE



EVERYTHING DEPEND ON THE DOMAIN LAYER
(BUSINESS LOGIC)

CLEAN VS N-LAYER ARCHITECTURE



EVERYTHING DEPEND ON THE DOMAIN LAYER
(BUSINESS LOGIC)

WHERE SHOULD WE USE CLEAN ARCHITECTURE?

- 1. KHI BẠN THEO ĐUỔI DOMAIN-DRIVEN DESIGN (DDD)
- 2. KHI HỆ THỐNG CÓ BUSINESS LOGIC PHỨC TẠP



WHERE SHOULD WE USE CLEAN ARCHITECTURE?

- ✓ 1. KHI BẠN THEO ĐUỔI DOMAIN-DRIVEN DESIGN (DDD)
- ✓ 2. KHI HỆ THỐNG CÓ BUSINESS LOGIC PHỨC TẠP
- ✓ 3. KHI BẠN CẦN HỆ THỐNG DỄ TEST
- ✓ 4. KHI BẠN MUỐN KIẾN TRÚC ĐÓNG VAI TRÒ KIỂM SOÁT (ENFORCE POLICIES)

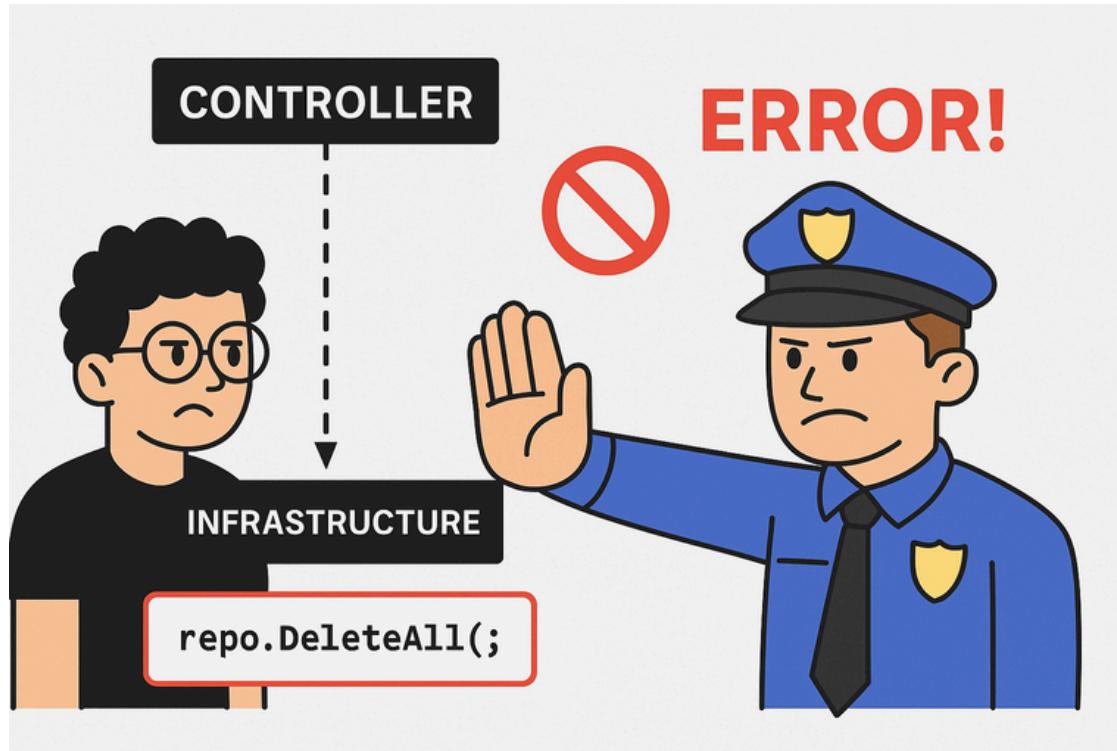


ENFORCE POLICIES BY ARCHITECTURE

🔑 1. KIẾN TRÚC ÉP DEV ĐI ĐÚNG – NHỜ COMPILER KIỂM SOÁT

TRONG CLEAN ARCHITECTURE:

- ✗ DEV KHÔNG THỂ "LỠ TAY" GỌI THẲNG TỪ CONTROLLER XUỐNG DATABASE
- ✓ VÌ NẾU LÀM VẬY... SẼ LỖI COMPILE!



```
var userProfile = await _userRepository.GetUserProfileById(userDto.Id);  
userProfile.Id = new Guid();
```

✗DEV KHÔNG BIẾT VÀ TỰ Ý THAY ĐỔI ID,...





**"KIẾN TRÚC LÀ LUẬT LỆ, COMPILER
LÀ CẢNH SÁT – CÙNG NHAU BẢO VỆ
SỰ 'TRONG SẠCH' CHO HỆ THỐNG."**



ENFORCE POLICIES BY ARCHITECTURE

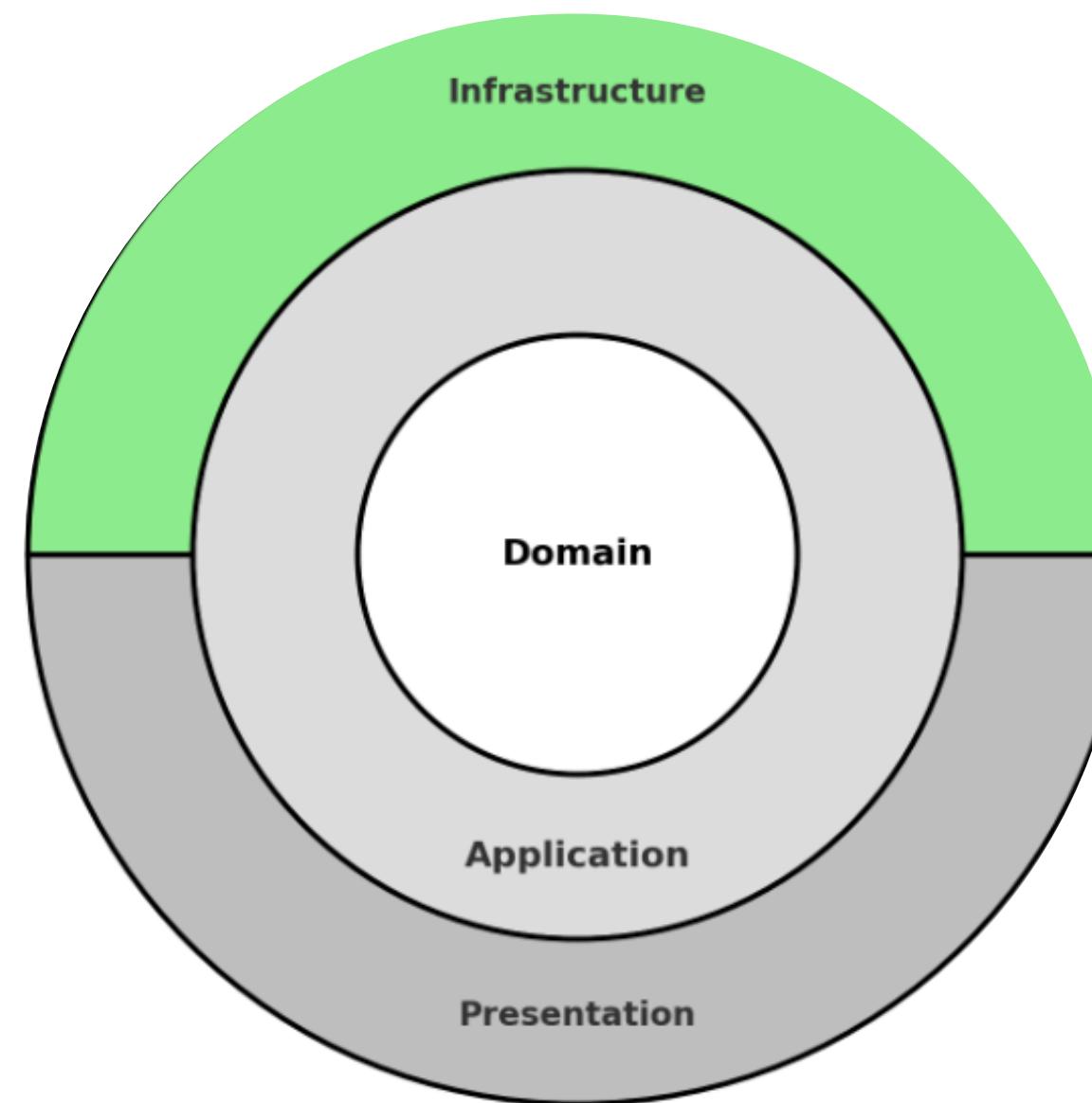
Quy định	Thực tế
 Tài liệu nội bộ	Có thể bị quên, bỏ qua
 Quy ước miệng	Ai viết cũng khác nhau
 Clean Architecture	Không tuân đúng → compile không chạy



CLEAN ARCHITECTURE - PHÂN TẦNG & TRÁCH NHIỆM



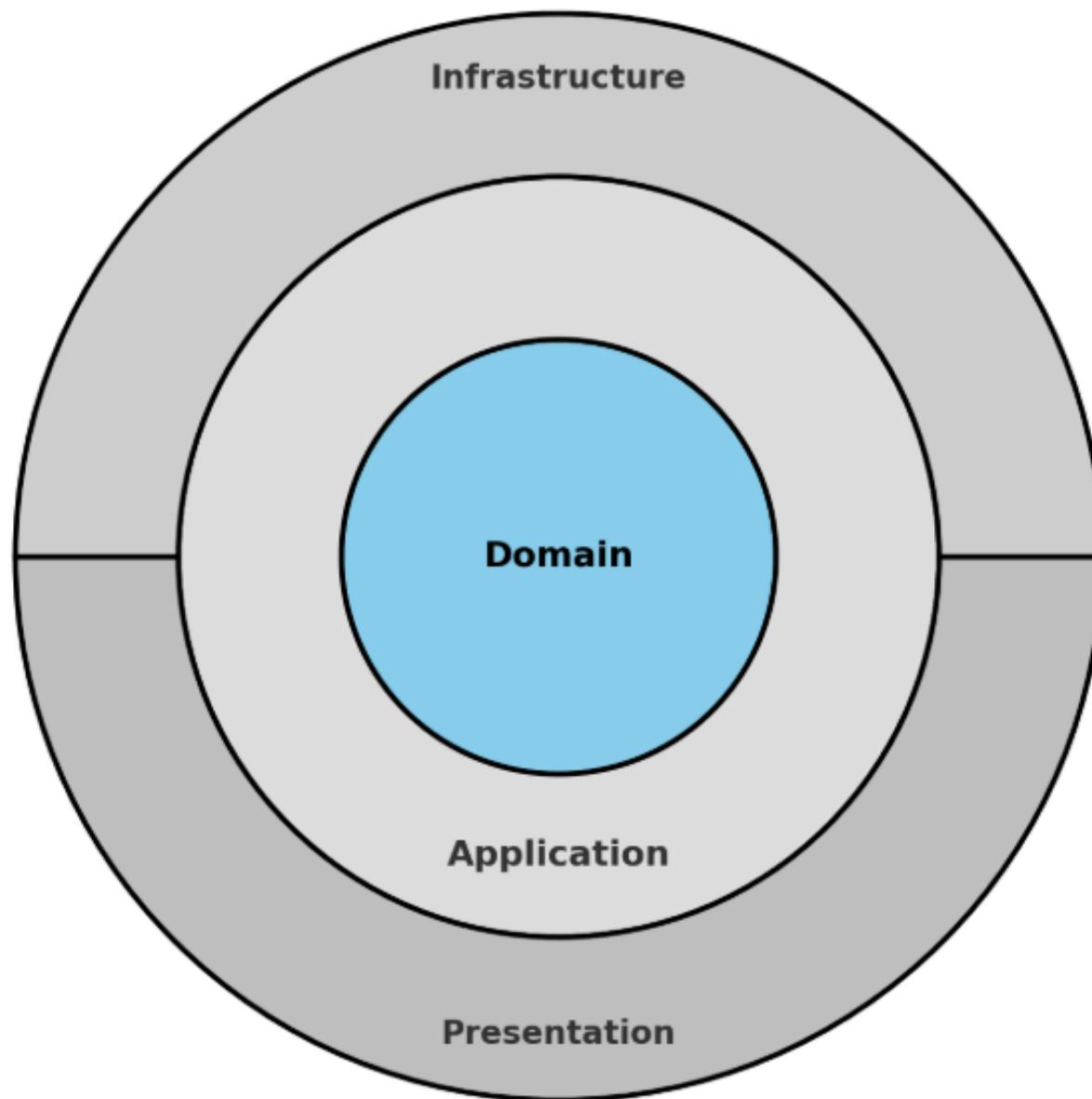
CLEAN ARCHITECTURE – PHÂN TẦNG & TRÁCH NHIỆM





DOMAIN LAYER - LOGIC NGHIỆP VỤ

PAGE 32



📌 1. VAI TRÒ CHÍNH:

- LÀ TẦNG CỐT LÕI CỦA HỆ THỐNG
- CHỨA TOÀN BỘ LUẬT NGHIỆP VỤ (BUSINESS RULES)
- KHÔNG PHỤ THUỘC VÀO BẤT KỲ TẦNG NÀO KHÁC (UI, DB, API...)



DOMAIN LAYER - LOGIC NGHIỆP VỤ

💡 2. THÀNH PHẦN CHÍNH:

◆ ENTITY

- CÁC ĐỐI TƯỢNG CÓ THẬT TRONG NGHIỆP VỤ, MANG HÀNH VI CỤ THỂ

VÍ DỤ: STUDENT, STUDYPROGRAM, FACULTY

```
public class User
{
    public string NickName { get; private set; } = string.Empty;
    public Email Email { get; private set; }
    public string FirstName { get; private set; } = string.Empty;
    public string LastName { get; private set; } = string.Empty;
    public DateTime JoinDate { get; private set; }

}
```



DOMAIN LAYER - LOGIC NGHIỆP VỤ

2. THÀNH PHẦN CHÍNH:

◆ ENTITY

- CÁC ĐỐI TƯỢNG CÓ THẬT TRONG NGHIỆP VỤ, MANG HÀNH VI CỤ THỂ

VÍ DỤ: STUDENT, STUDYPROGRAM, FACULTY

◆ VALUE OBJECTS

- ĐẠI DIỆN CHO CÁC GIÁ TRỊ NGHIỆP VỤ CÓ CẤU TRÚC

VÍ DỤ: EMAIL, ADDRESS, MONEY

→ KHÔNG CÓ DANH TÍNH RIÊNG, THƯỜNG LÀ THUỘC
TÍNH CỦA ENTITY

```
public class Email
{
    public string Value { get; }
    public bool isValidEmail();
    static public Email Create();
}
```



DOMAIN LAYER - LOGIC NGHIỆP VỤ

PAGE 35

2. THÀNH PHẦN CHÍNH:

◆ ENTITY

- CÁC ĐỐI TƯỢNG CÓ THẬT TRONG NGHIỆP VỤ, MANG HÀNH VI CỤ THỂ

VÍ DỤ: STUDENT, STUDYPROGRAM, FACULTY

◆ VALUE OBJECTS

- ĐẠI DIỆN CHO CÁC GIÁ TRỊ NGHIỆP VỤ CÓ CẤU TRÚC

VÍ DỤ: EMAIL, ADDRESS, MONEY

→ KHÔNG CÓ DANH TÍNH RIÊNG, THƯỜNG LÀ THUỘC
TÍNH CỦA ENTITY

◆ EXCEPTIONS

◆ DOMAIN EVENTS

APPLICATION LAYER – USE CASE

1. NHIỆM VỤ CHÍNH:

ORCHESTRATE THE DOMAIN

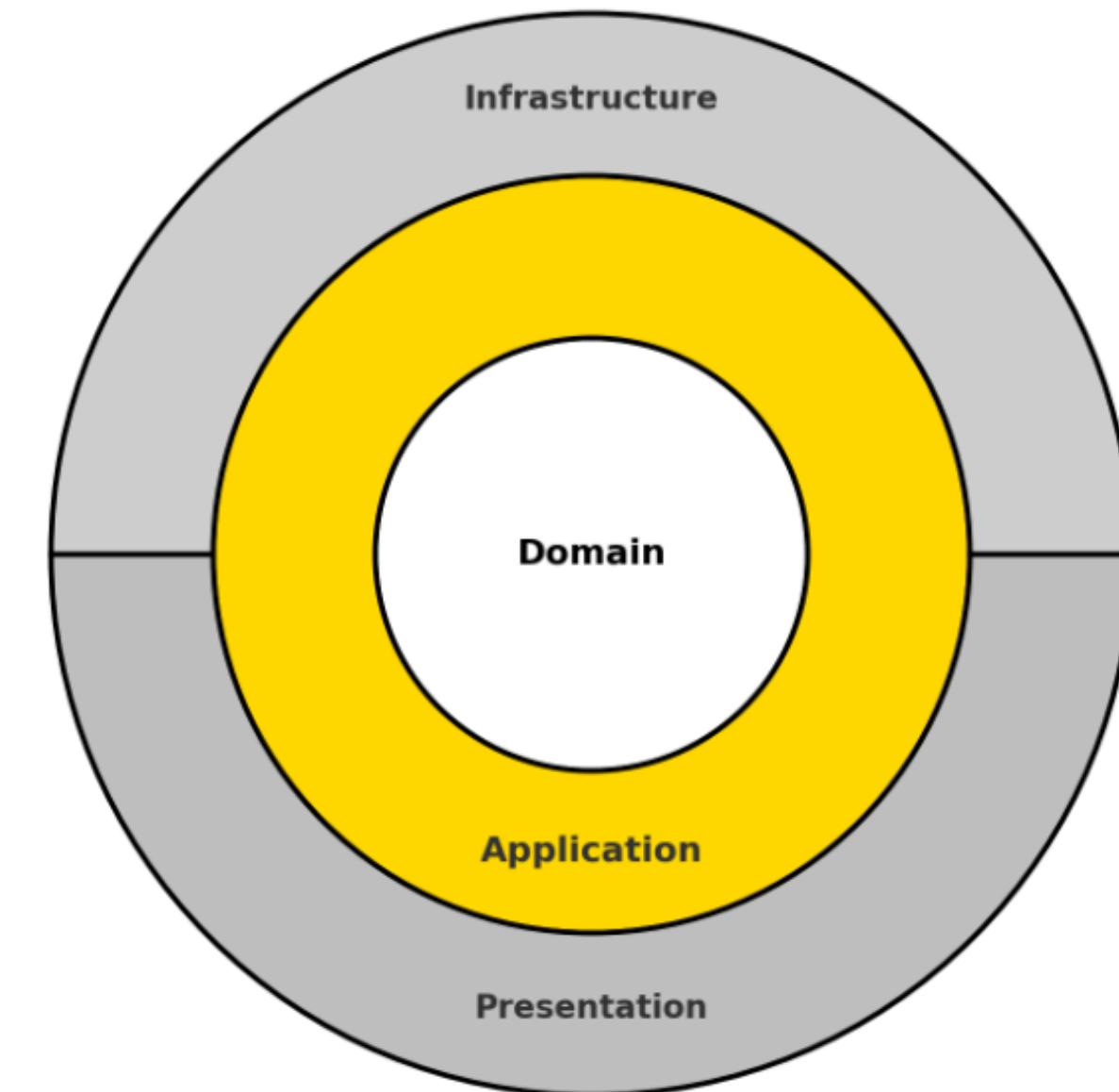
→ ĐIỀU PHỐI CÁC ENTITY, VALUE OBJECT TỪ DOMAIN
ĐỂ XỬ LÝ NGHIỆP VỤ

CONTAIN BUSINESS LOGIC (USE CASE LOGIC)

→ CHỨA LOGIC CỦA TỪNG TRƯỜNG HỢP CỤ THỂ (ADD,
DELETE, UPDATE...)

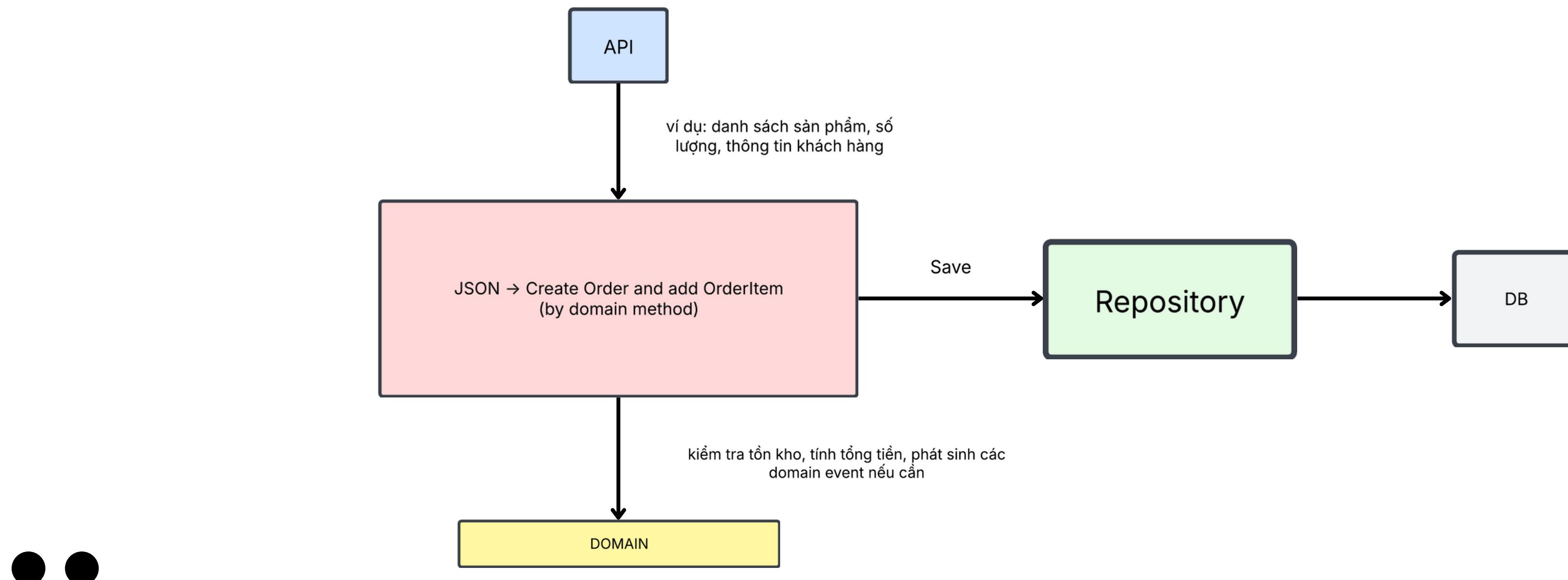
DEFINE USE CASES

→ MỖI CLASS TRONG TẦNG NÀY ĐẠI DIỆN CHO MỘT USE
CASE CỤ THỂ

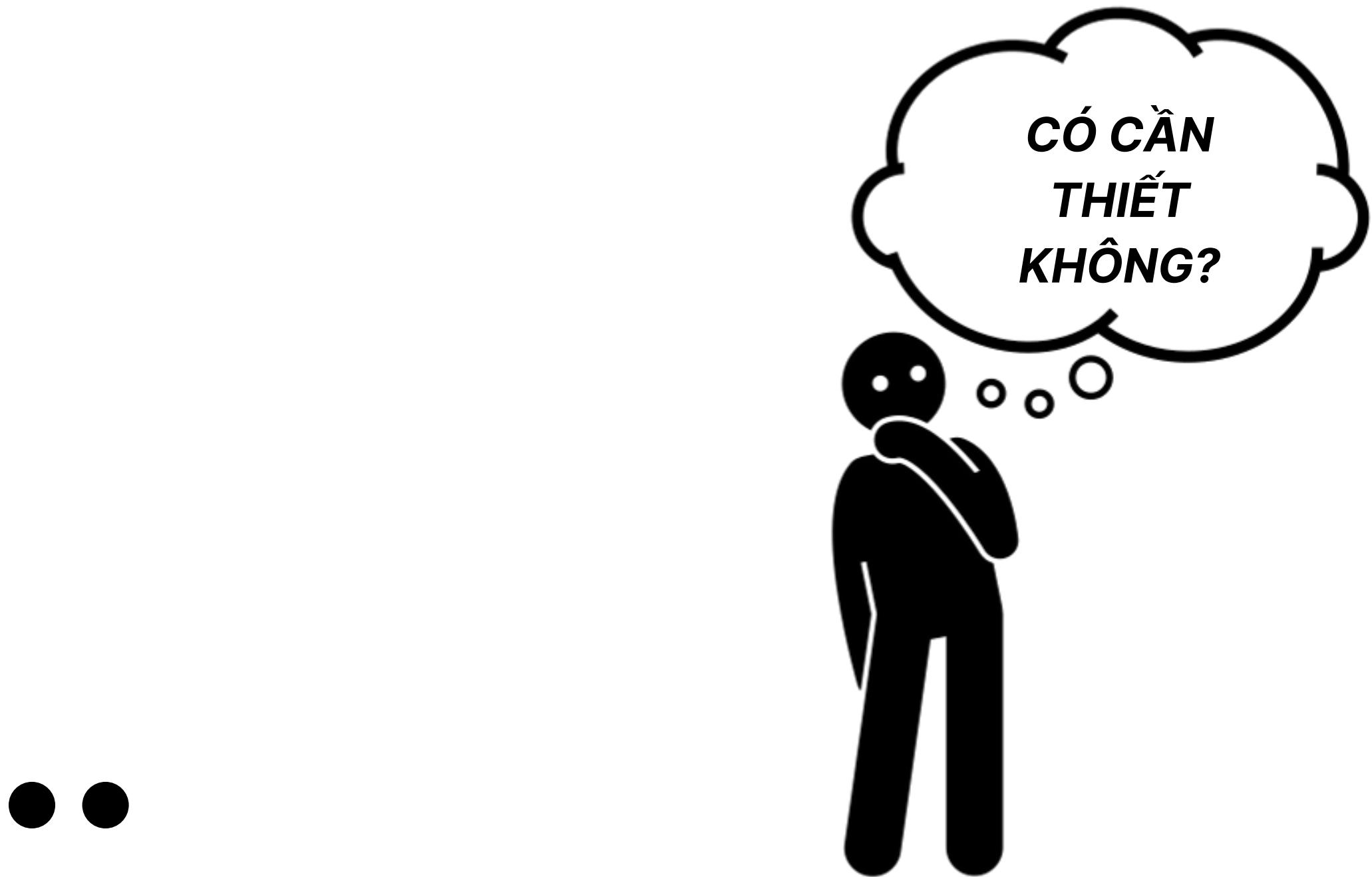


APPLICATION LAYER – USE CASE

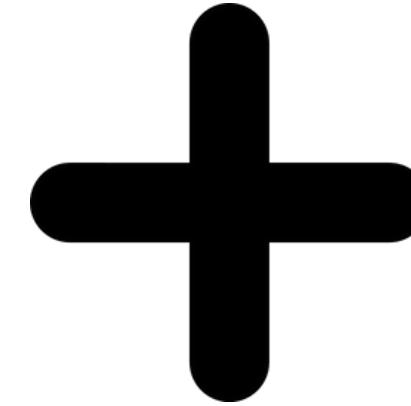
VÍ DỤ: TẠO ĐƠN HÀNG



CQRS – COMMAND QUERY RESPONSIBILITY SEGREGATION



CQRS - GIA VỊ HOÀN HẢO CHO CLEAN ARCHITECTURE



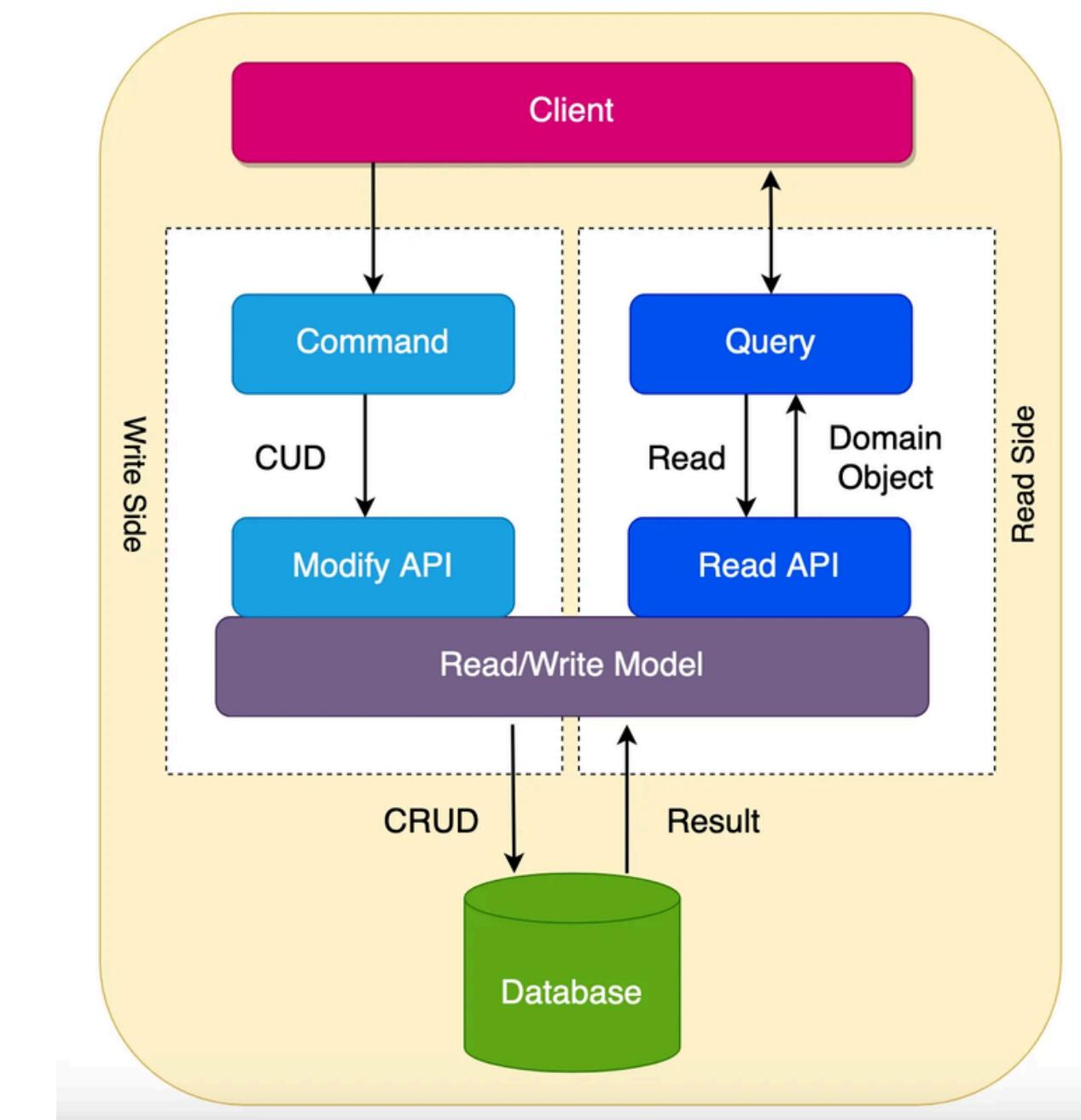
CQRS - GIA VỊ HOÀN HẢO CHO CLEAN ARCHITECTURE

PROS:

- SINGLE RESPONSIBILITY
- INTERFACE SEGREGATION
- LOOSE COUPLING

CONS:

- INDIRECTION



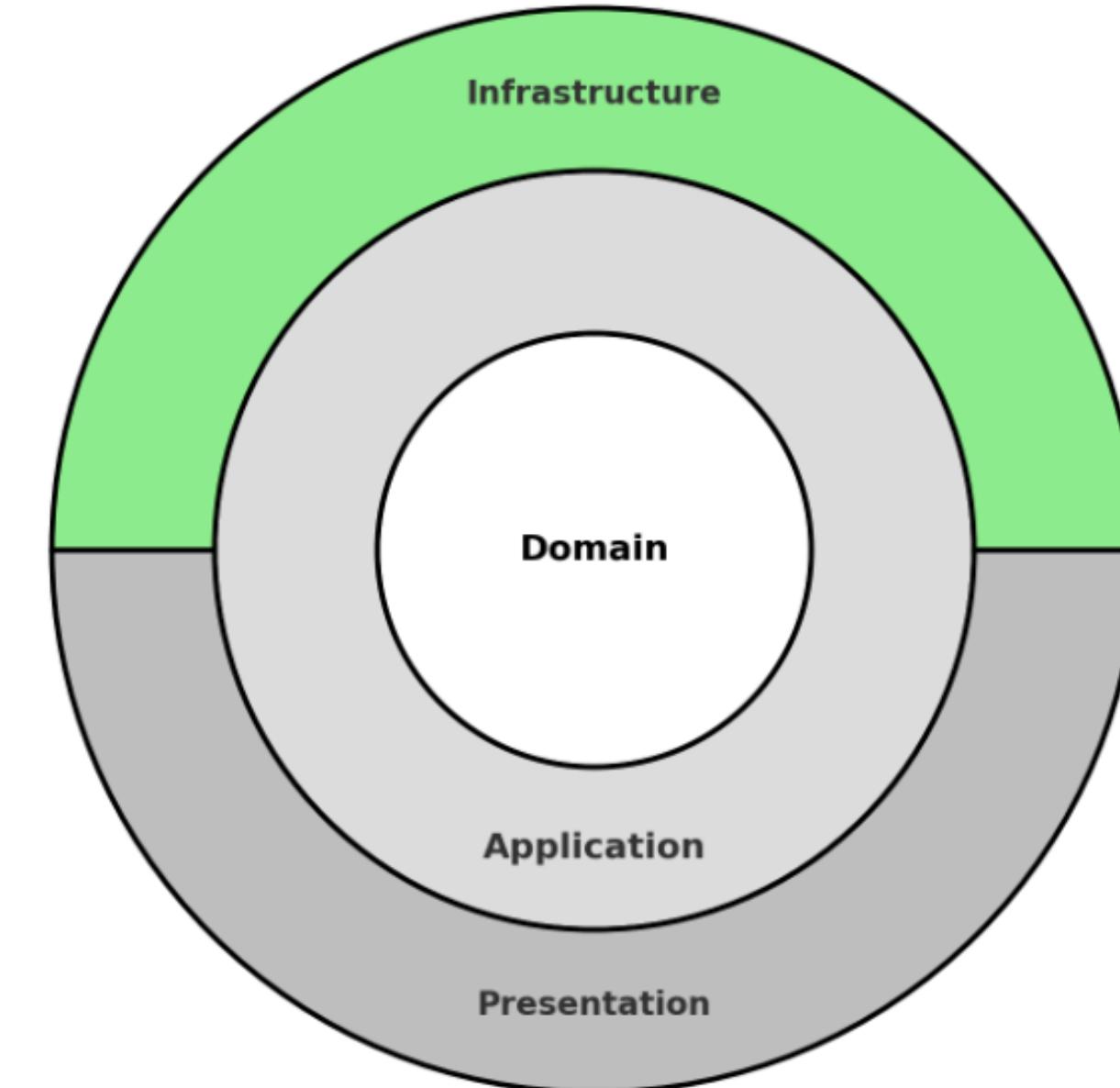


INFRASTRUCTURE LAYER – CUNG CẤP NỀN TẢNG CHO HỆ THỐNG

PAGE 41

1. TẦNG INFRASTRUCTURE LÀ GÌ?

- INFRASTRUCTURE (CƠ SỞ HẠ TẦNG) TRONG CLEAN ARCHITECTURE TƯƠNG TỰ NHƯ CƠ SỞ HẠ TẦNG TRONG ĐỜI SỐNG THỰC:
 - GIỐNG NHƯ HỆ THỐNG ĐƯỜNG SÁ, CẦU CỐNG, ĐIỆN, NƯỚC...
- CUNG CẤP NỀN TẢNG CHO CÁC TẦNG TRÊN HOẠT ĐỘNG.

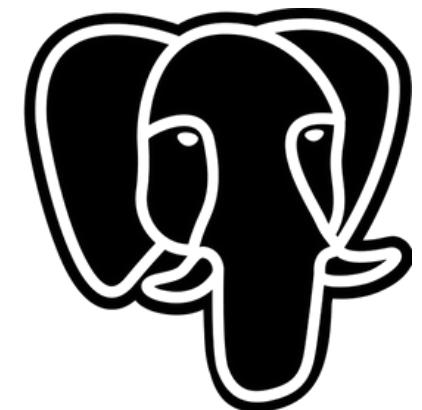




INFRASTRUCTURE LAYER - CUNG CẤP NỀN TẢNG CHO HỆ THỐNG

2. THÀNH PHẦN TRONG INFRASTRUCTURE LAYER:

- ◆ DATABASE
- ◆ EMAIL PROVIDER
- ◆ LLM (LARGE LANGUAGE MODELS)
- ◆ IDENTITY & AUTHENTICATION
- ◆ ETC,

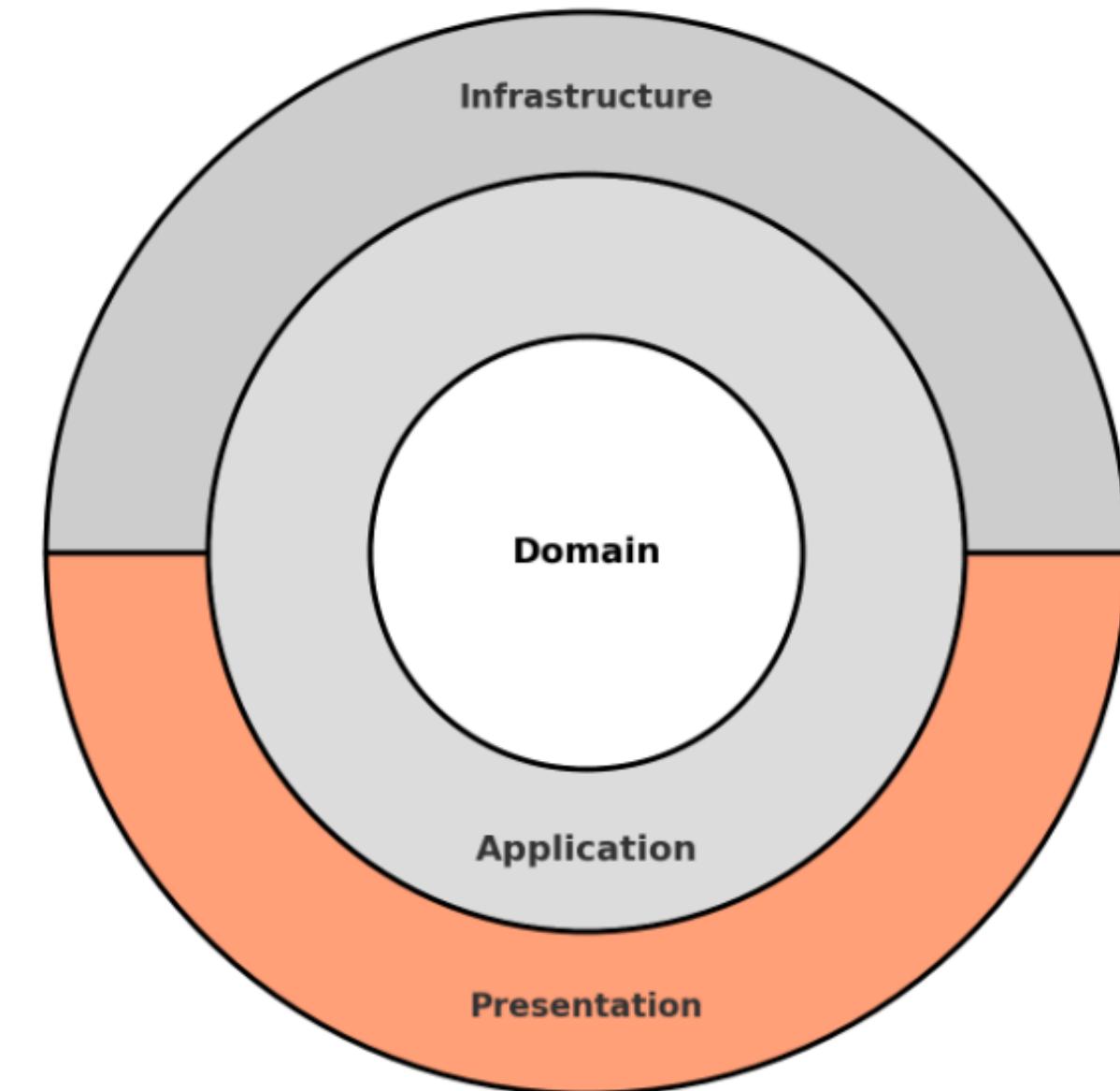




PRESENTATION LAYER – ENTRY POINT CỦA ỨNG DỤNG

1. PRESENTATION LÀ GÌ?

- PRESENTATION = TẦNG TRÌNH BÀY / GIAO DIỆN
- LÀ ĐIỂM KHỞI ĐẦU (ENTRY POINT) CHO MỌI TƯƠNG TÁC TỪ THẾ GIỚI BÊN NGOÀI VỚI HỆ THỐNG
 - NHẬN YÊU CẦU TỪ CLIENT → GỬI VÀO USE CASE
→ TRẢ VỀ KẾT QUẢ





PRESENTATION LAYER – ENTRY POINT CỦA ỨNG DỤNG

* 3. MỘT SỐ HÌNH THỨC CỤ THỂ CỦA PRESENTATION LAYER:



WEB API



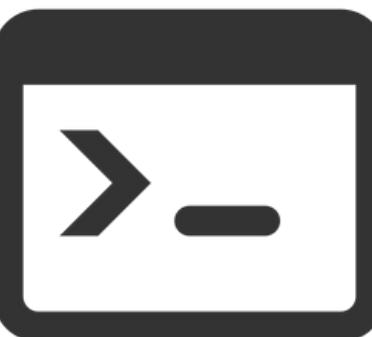
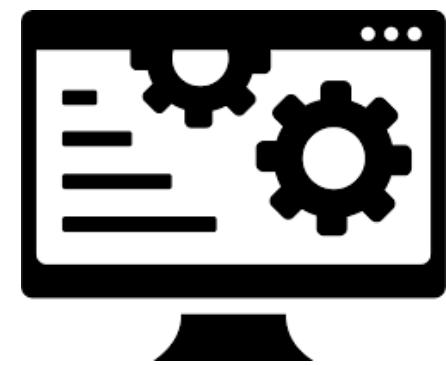
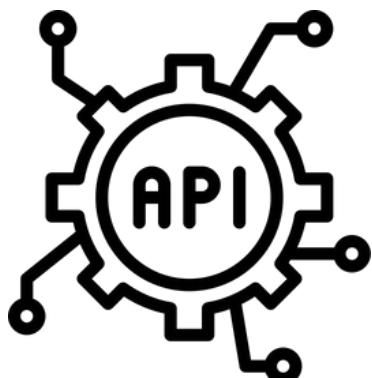
WPF / WINUI (DESKTOP APP)



CONSOLE APP / CLI



MOBILE APP / BLAZOR / GRPC CLIENT...



DEMO SOURCE CODE



Q & A





END

CẢM ƠN CÁC BẠN ĐÃ LẮNG NGHE

..

