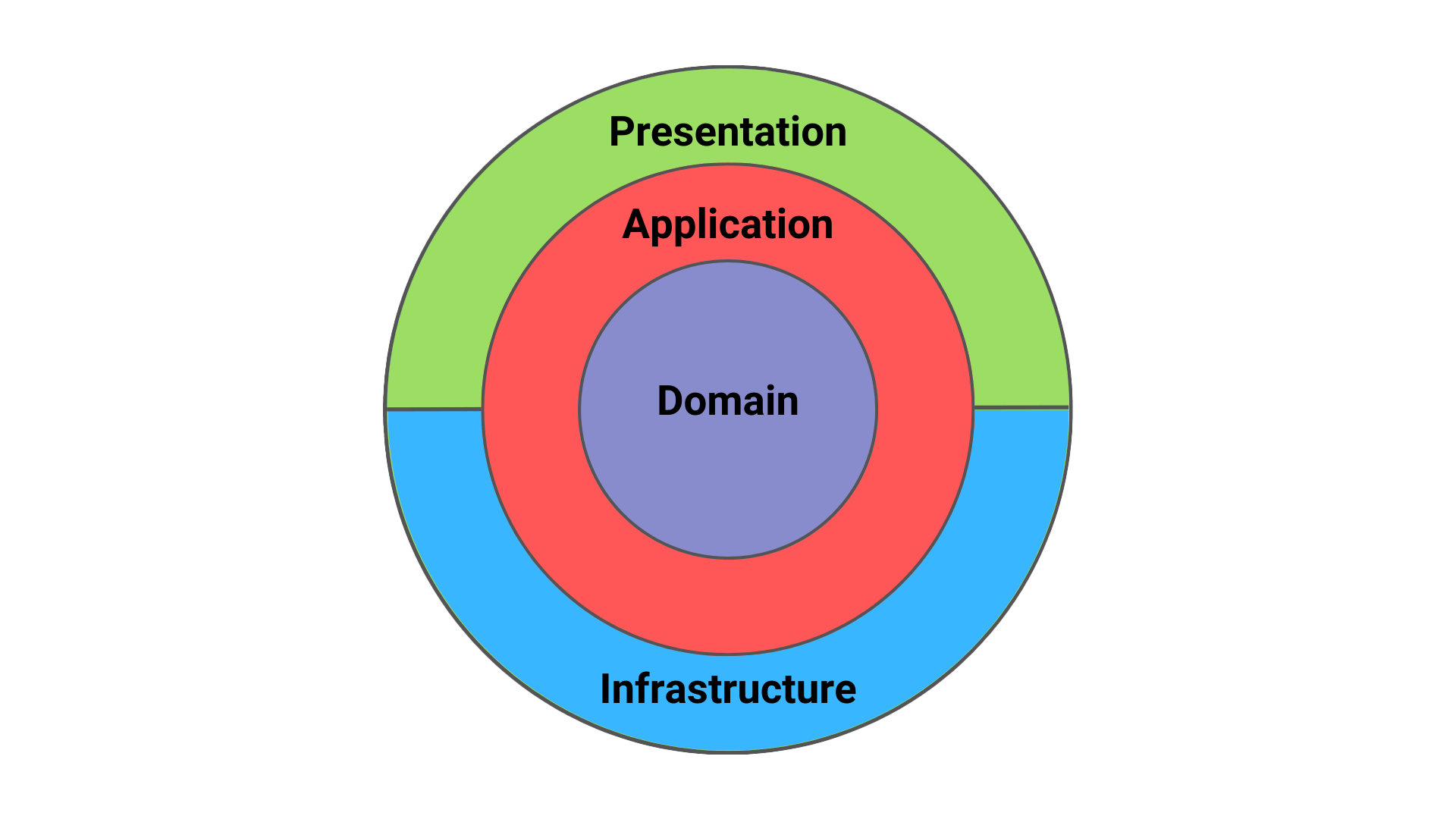
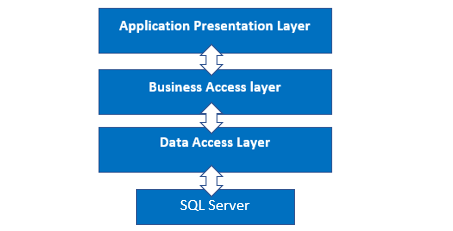
**1. Clean architecture**



**2. Layered Architecture**



## **Clean Architecture Layers**

**4 Main Layers** (from outer to inner):

### **1. Presentation Layer (API/UI)**

* Controllers, endpoints, user interfaces
* Handles requests/responses
* **Depends on:** Application layer

### **2. Application Layer (Business Logic)**

* Use cases, commands, queries (CQRS)
* Business rules and workflows
* **Depends on:** Domain layer

### **3. Infrastructure Layer (Data & External)**

* Database, repositories, external APIs
* File systems, email services
* **Depends on:** Application + Domain layers

### **4. Domain Layer (Core Business)**

* Entities, value objects, domain rules
* **Pure business logic - NO dependencies**

## **Key Rule: Dependency Inversion**

API → Application → Domain ← Infrastructure

* **Inner layers** don't know about outer layers
* **Domain** is the center with zero dependencies
* **Infrastructure** implements interfaces from Application
* All dependencies point **inward** toward Domain

## **CQRS (Command Query Responsibility Segregation)**

**Core Concept:** Separate **Write operations** from **Read operations**

## **Key Components**

### **Commands (Write Operations)**

* Create, Update, Delete
* Change system state
* Return success/failure (not data)

CreateUserCommand → CreateUserHandler → Database

### **Queries (Read Operations)**

* Get, List, Search
* Don't change system state
* Return data

GetUserQuery → GetUserHandler → Read-optimized data

## **Benefits**

✅ **Optimized Models**: Different models for reads vs writes

✅ **Scalability**: Scale read/write operations independently

✅ **Clarity**: Clear separation of concerns

✅ **Performance**: Optimized queries without write constraints

## **Simple Example**

// Command (Write)  
CreateProductCommand → Normalized database tables  
  
// Query (Read)   
GetProductsQuery → Denormalized view/DTO optimized for display