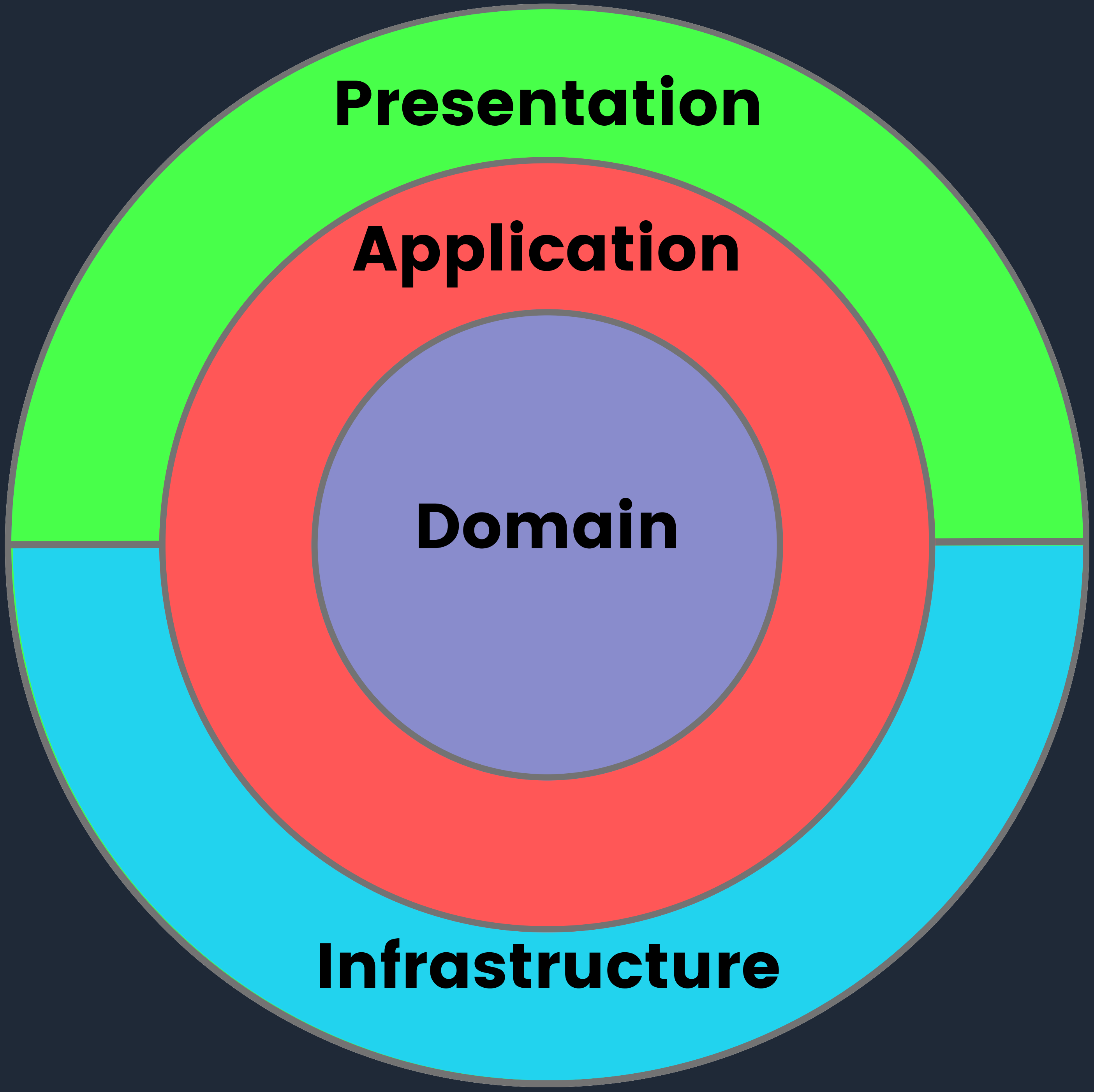
Pragmatic
Clean Architecture

Building production-ready applications

Architectural Principles

- Maintainability
- Testability
- Loose coupling

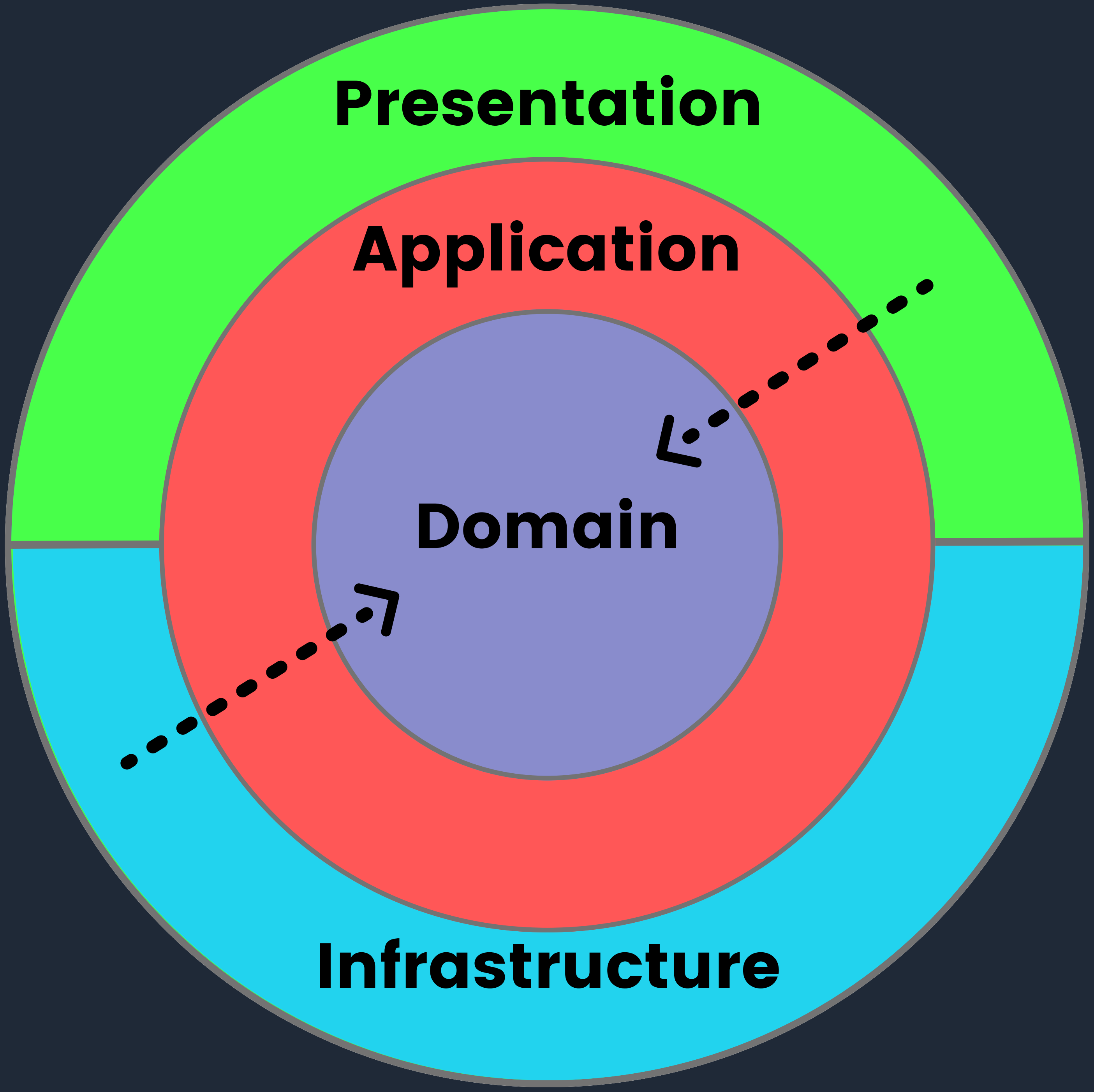


Guiding Design Principles

- Separation of concerns
- Encapsulation
- Dependency inversion
- Explicit dependencies
- Single responsibility
- DRY
- Persistence ignorance
- Bounded contexts

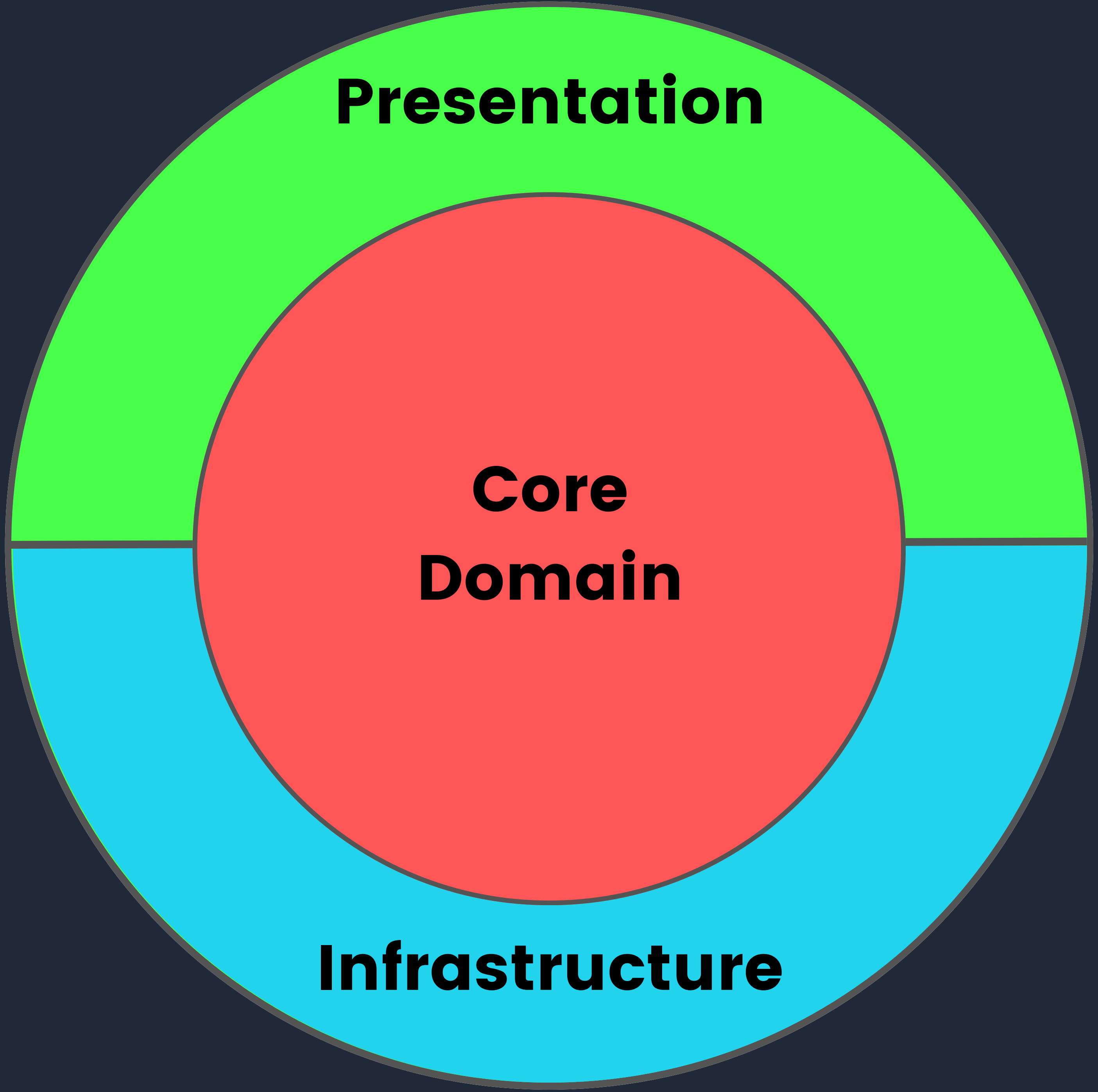
Dependency Inversion

- Dependencies flow inwards
- Inner layers define interfaces
- Outer layers implement them



Domain-Centric Architecture

- Domain-centric approach to organizing dependencies
- Similar architectures
  - Onion architecture
  - Hexagonal architecture

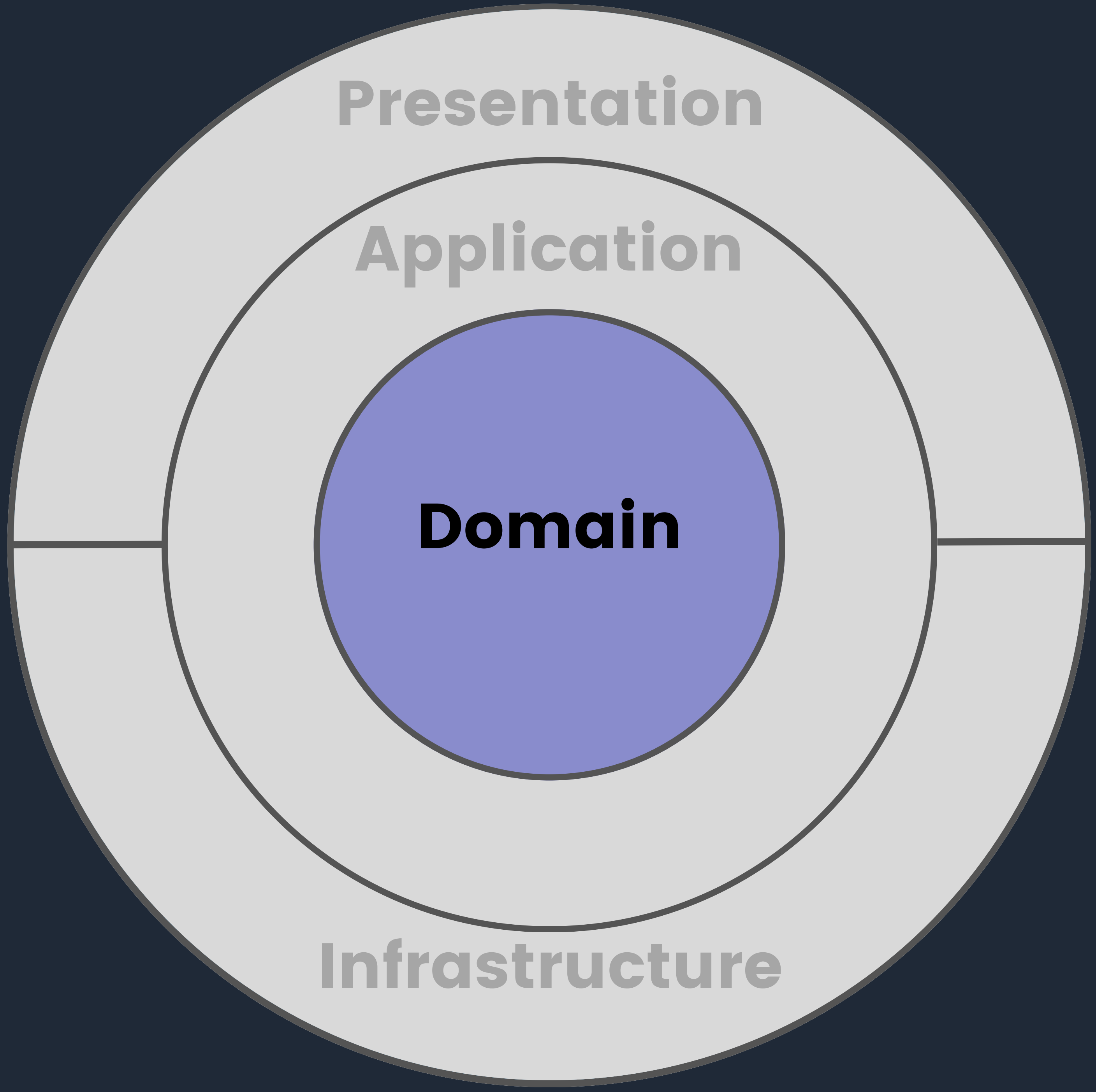


Where Should You Use It?

- Domain-Driven Design
- Complex business logic
- Highly testable projects
- Want architecture to enforce design policies

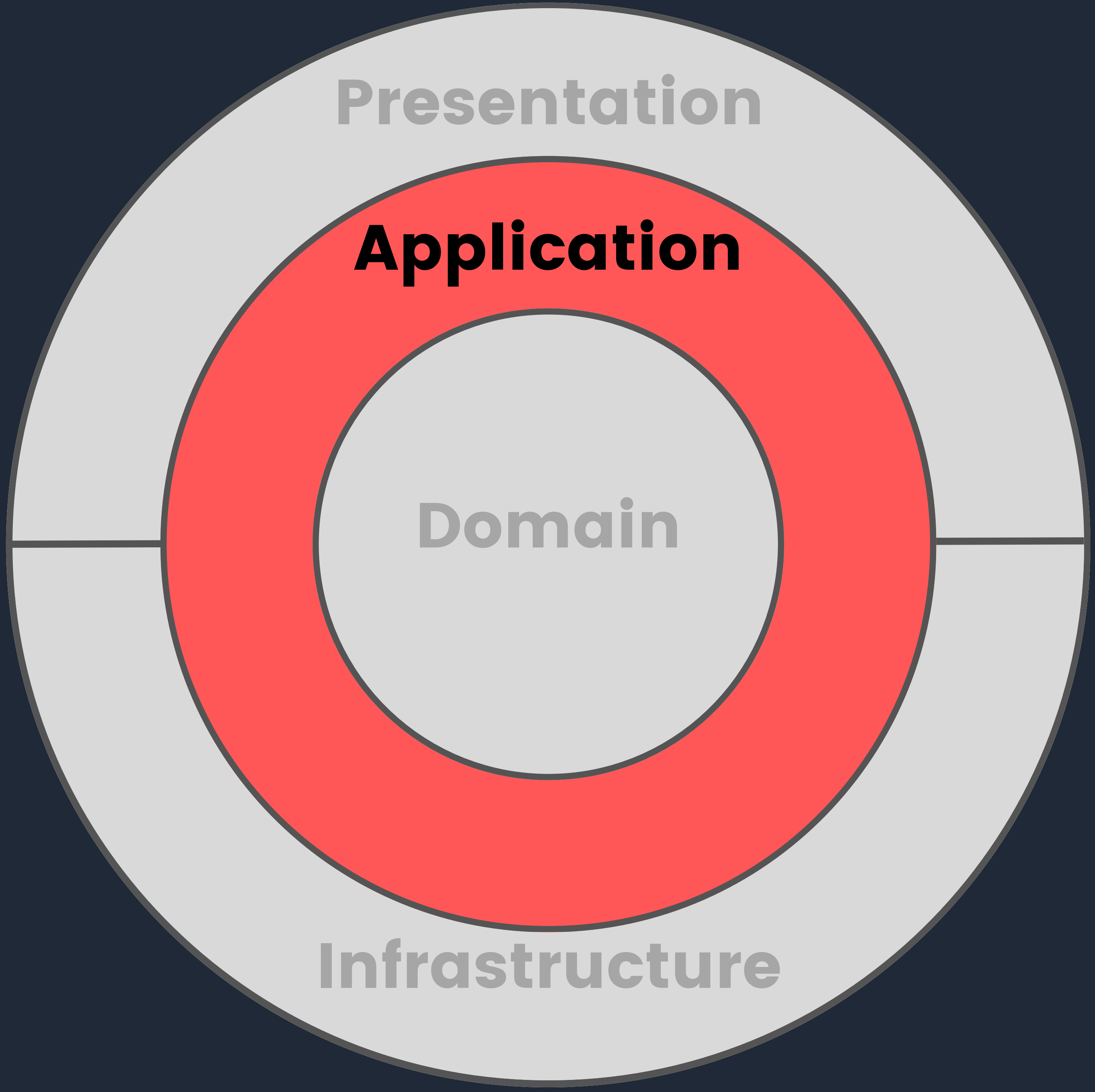
Domain Layer

- Entities
- Value objects
- Domain events
- Domain services
- Interfaces
- Exceptions
- Enums



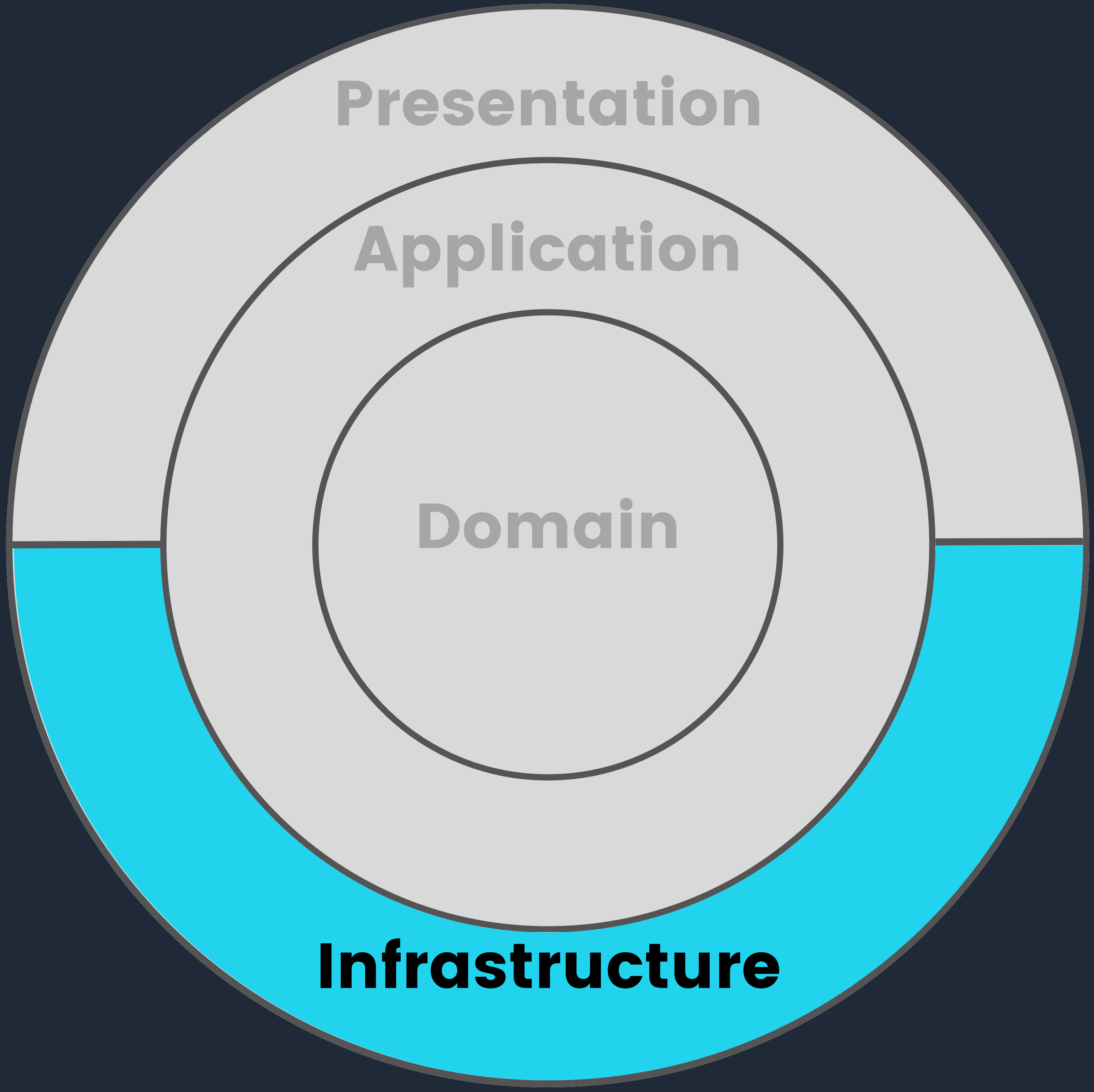
Application Layer

- Orchestrates the Domain
- Contains business logic
- Defines the Use Cases
    - Application services
    - CQRS with MediatR



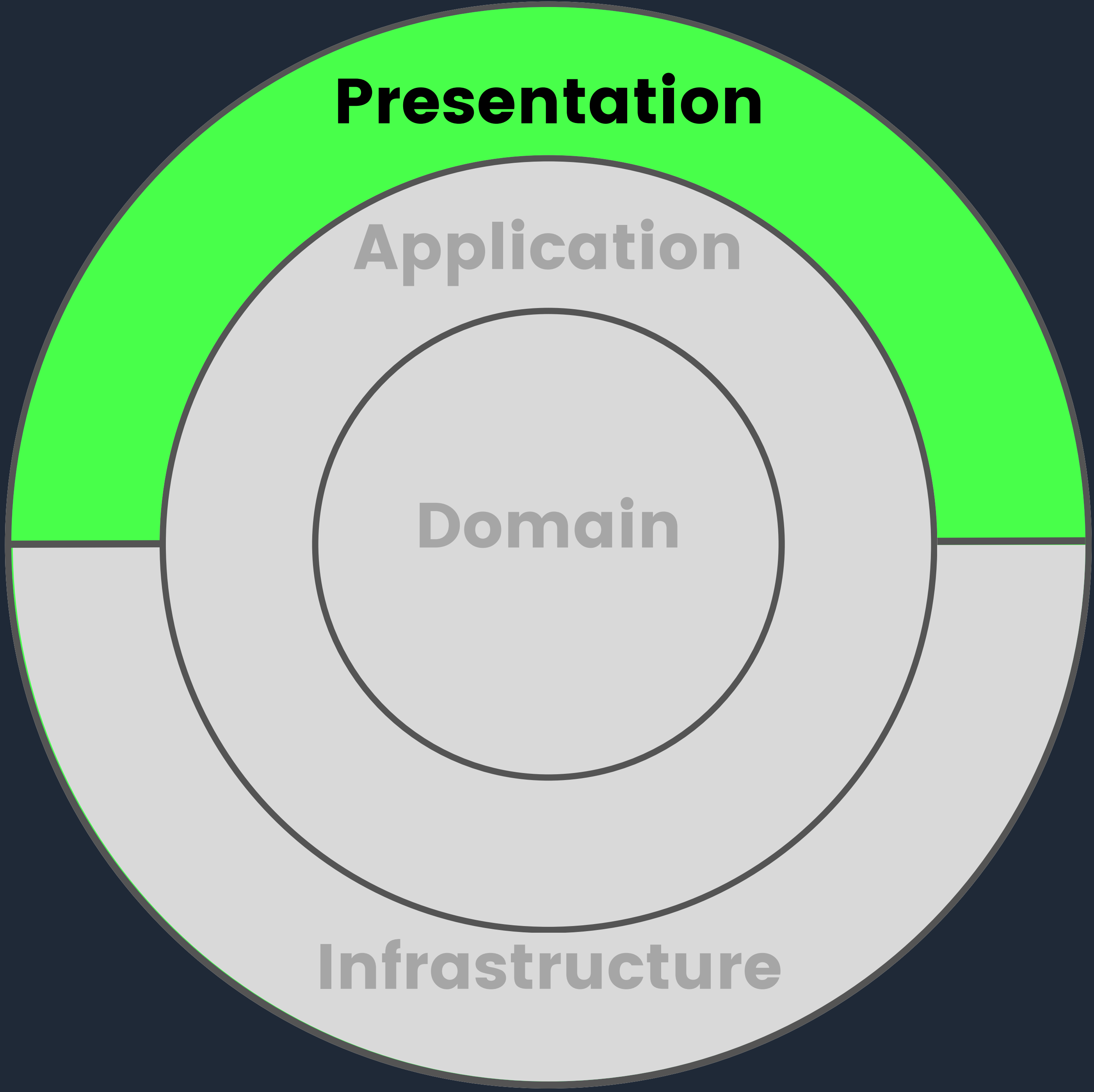
Infrastructure Layer

- External systems
  - Databases
  - Messaging
  - Email providers
  - Storage services
  - Identity
  - System clock



Presentation Layer

- Defines the entry point to the system
- REST API built with .NET 7
  - API endpoints
  - Middleware
  - DI setup

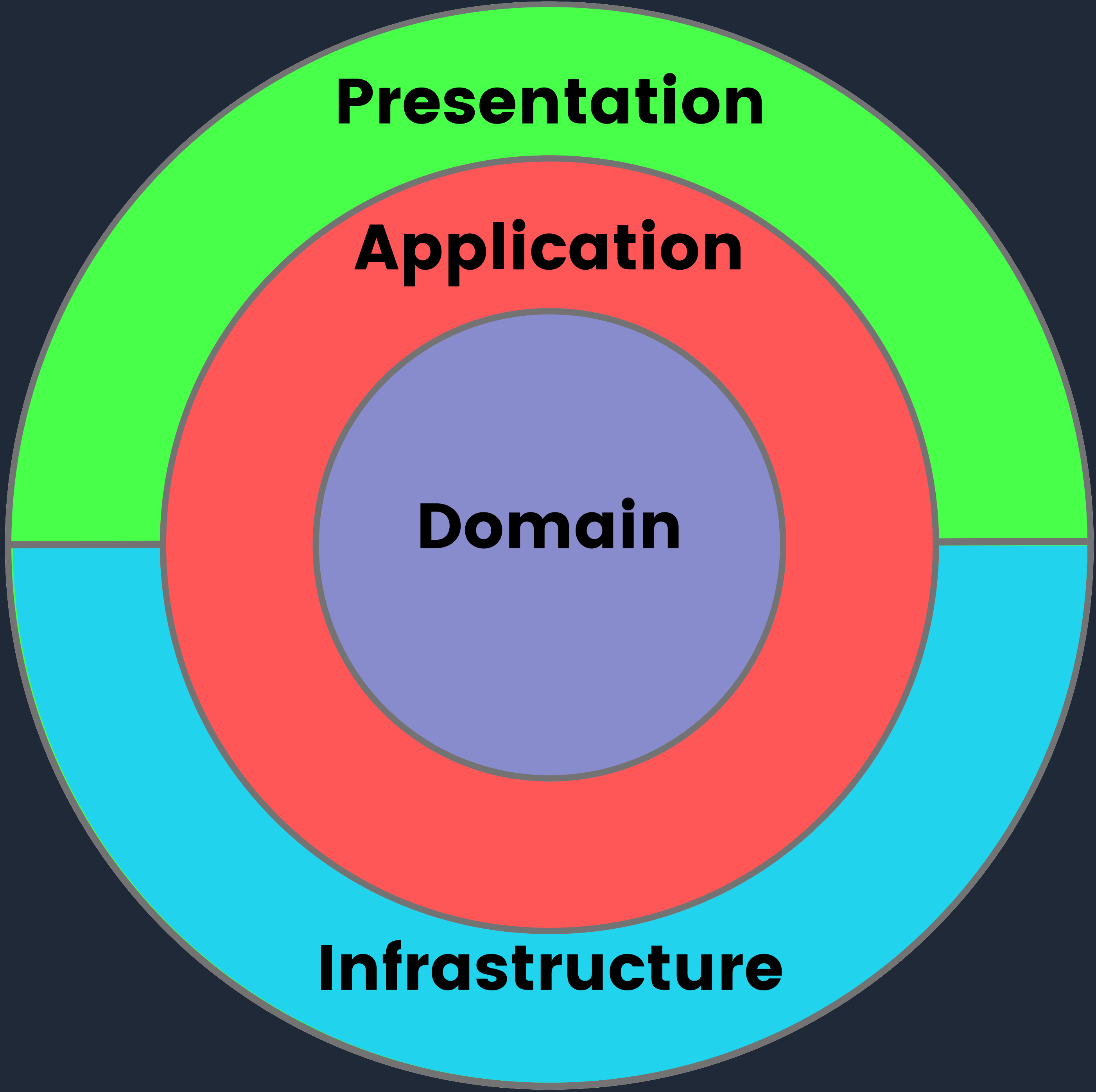


What are we building & why?

- What:
  - Apartment booking system
- Why:
  - Familiar, easy to understand
  - Not trivial, interesting business rules and logic
  - Doesn't require extensive domain knowledge

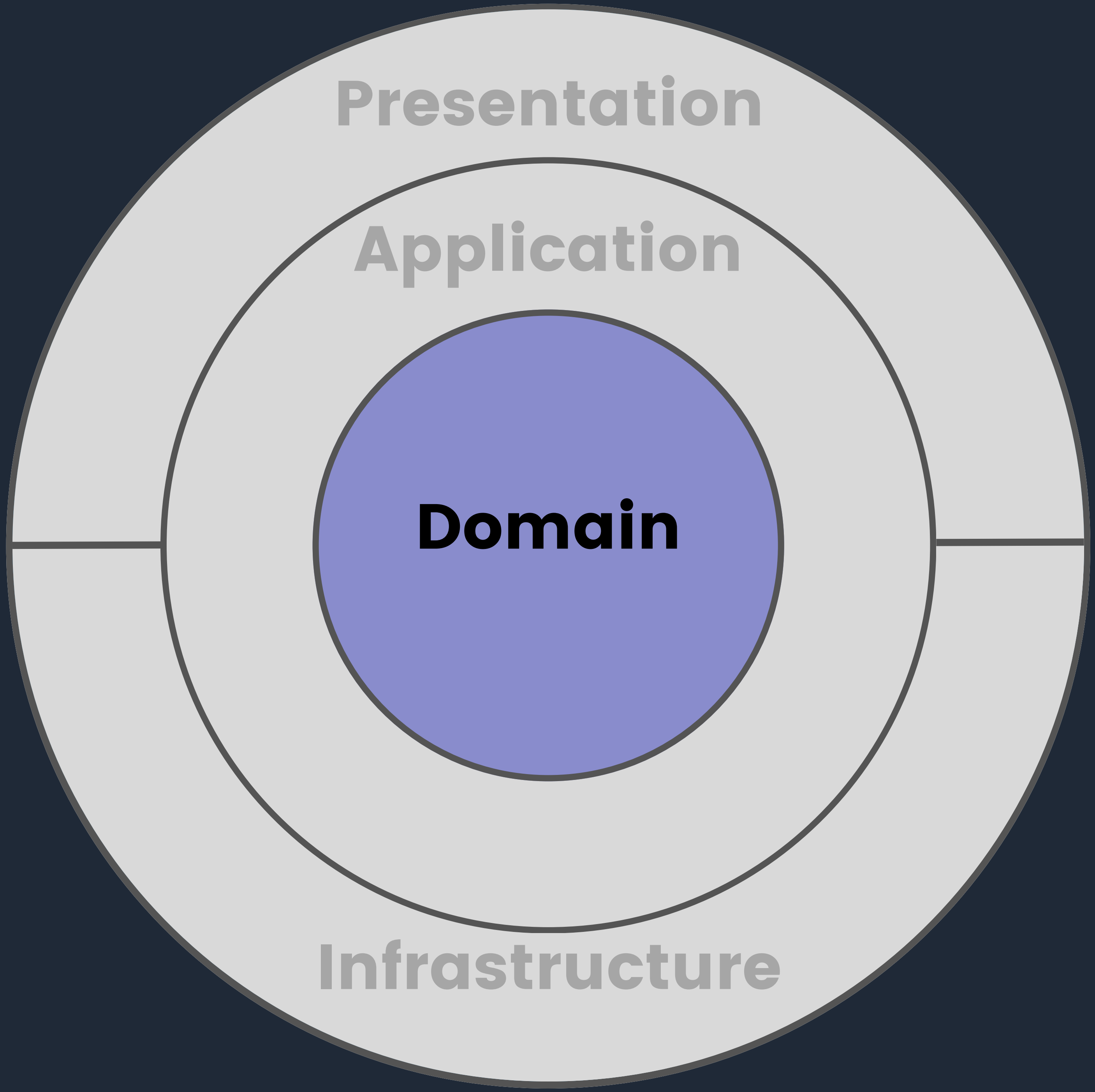
Applying Clean Architecture

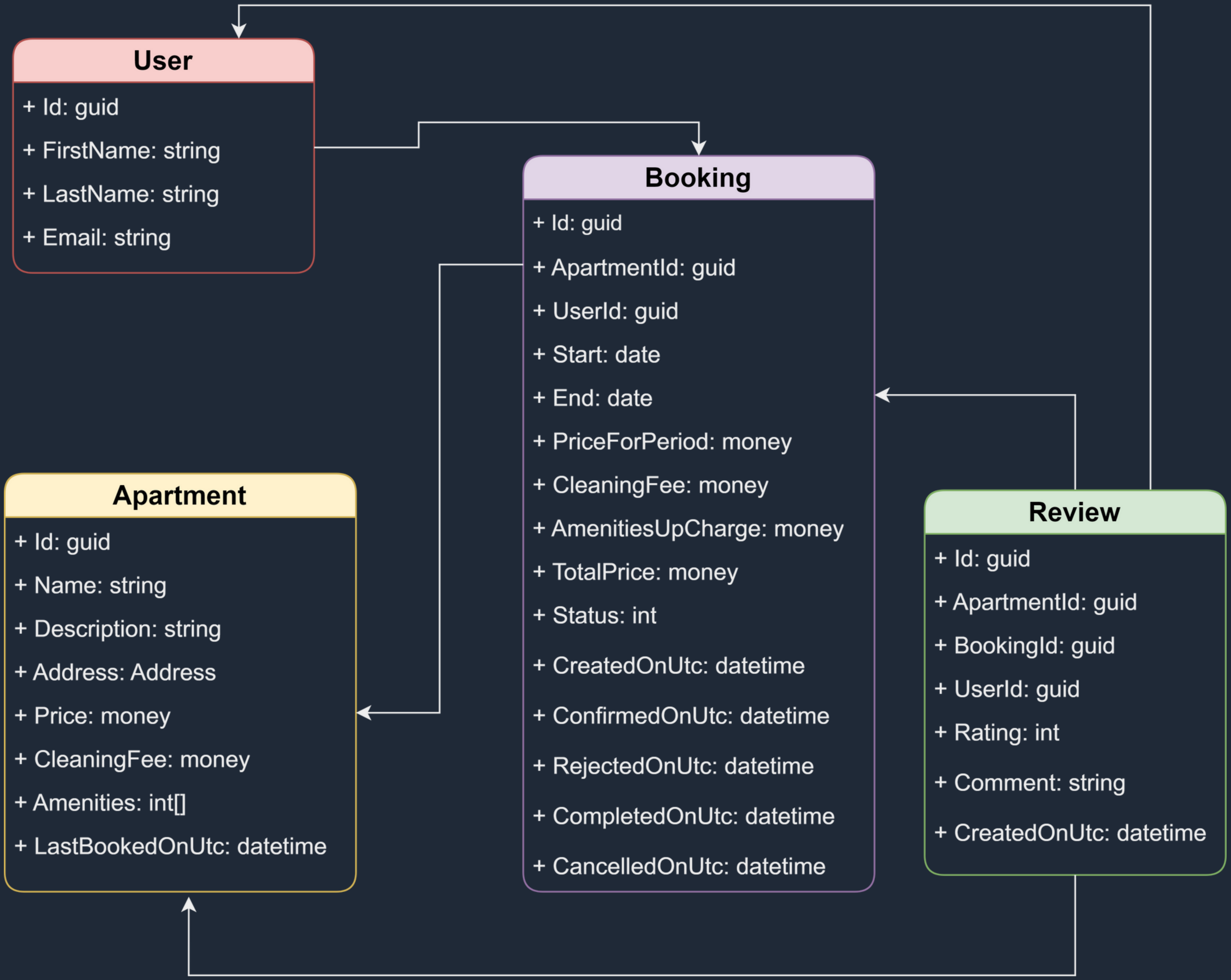
- Start from the Domain layer & work our way up
- Show best practices
- Discuss architectural considerations
- Domain-Driven Design



Domain Layer

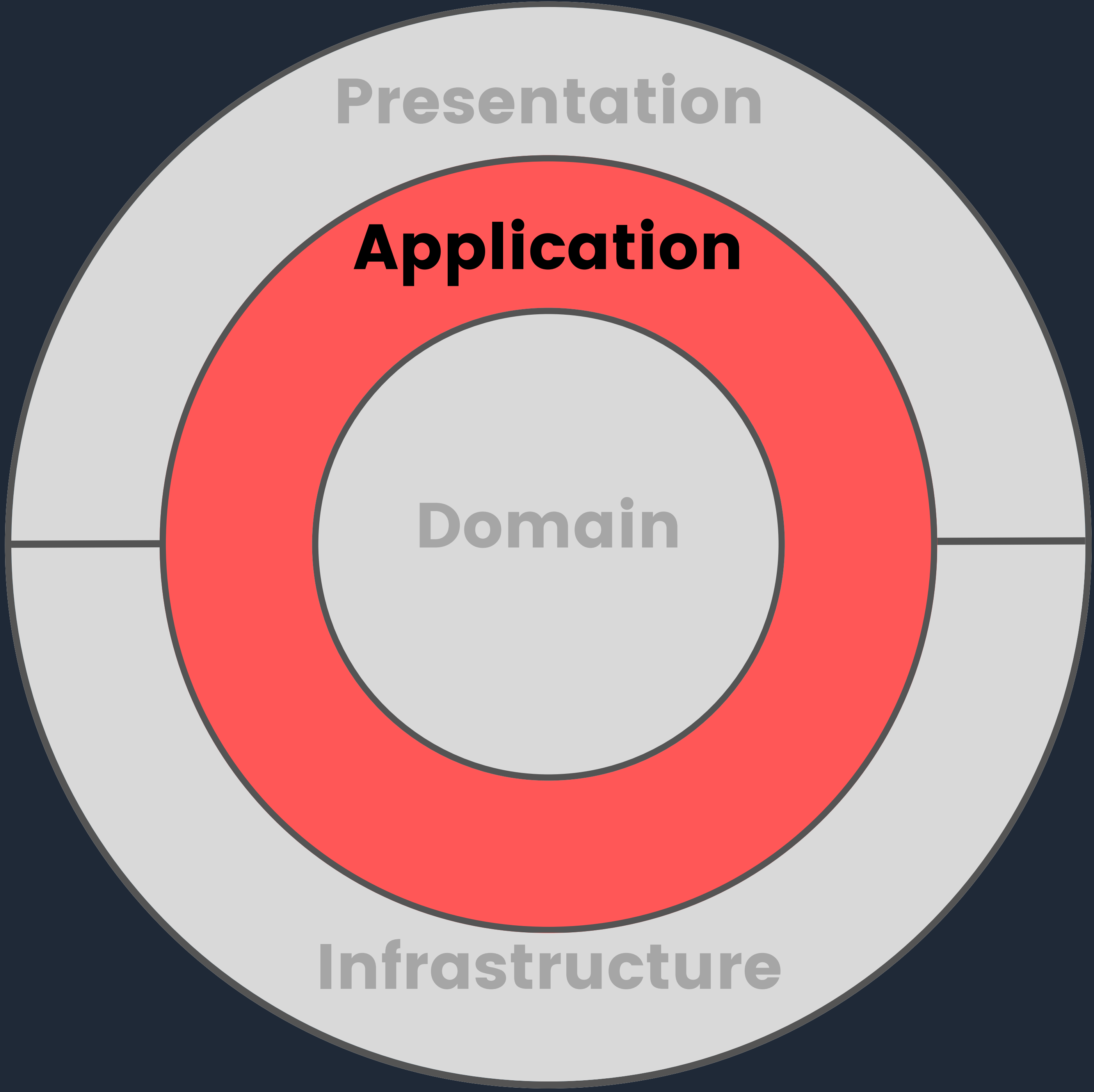
- Entities
- Value objects
- Domain events
- Domain services
- Interfaces
- Exceptions
- Enums




User
+ Id: guid
+ FirstName: string
+ LastName: string
+ Email: string
Booking
+ Id: guid
+ ApartmentId: guid
+ UserId: guid
+ Start: date
+ End: date
+ PriceForPeriod: money
+ CleaningFee: money
+ AmenitiesUpCharge: money
+ TotalPrice: money
+ Status: int
+ CreatedOnUtc: datetime
+ ConfirmedOnUtc: datetime
+ RejectedOnUtc: datetime
+ CompletedOnUtc: datetime
+ CancelledOnUtc: datetime
Apartment
+ Id: guid
+ Name: string
+ Description: string
+ Address: Address
+ Price: money
+ CleaningFee: money
+ Amenities: int[]
+ LastBookedOnUtc: datetime
Review
+ Id: guid
+ ApartmentId: guid
+ BookingId: guid
+ UserId: guid
+ Rating: int
+ Comment: string
+ CreatedOnUtc: datetime

Application Layer

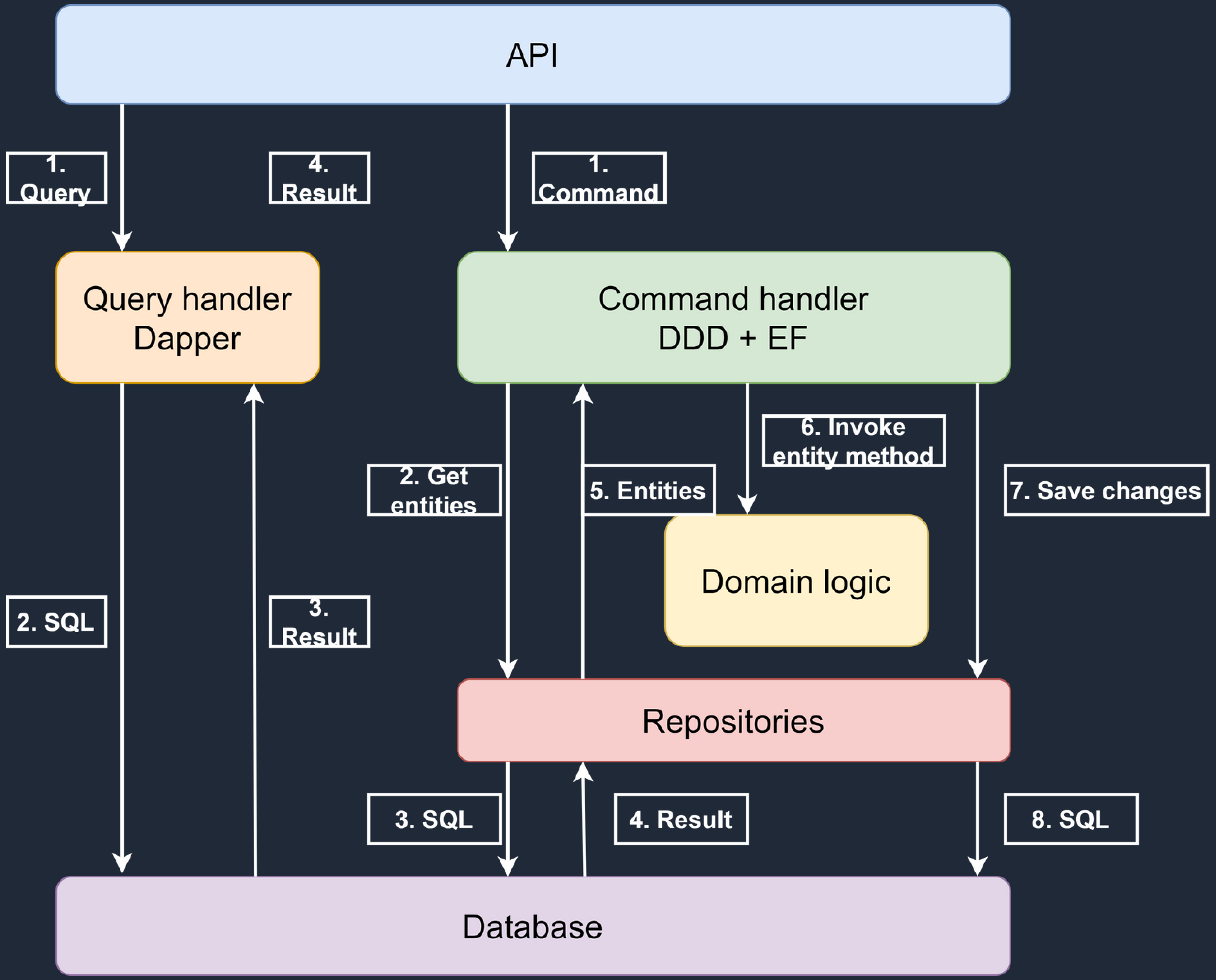
- Use cases = CQRS + MediatR
- Cross-cutting concerns
  - Logging
  - Validation
- Exceptions
- DI configuration



Benefits of CQRS

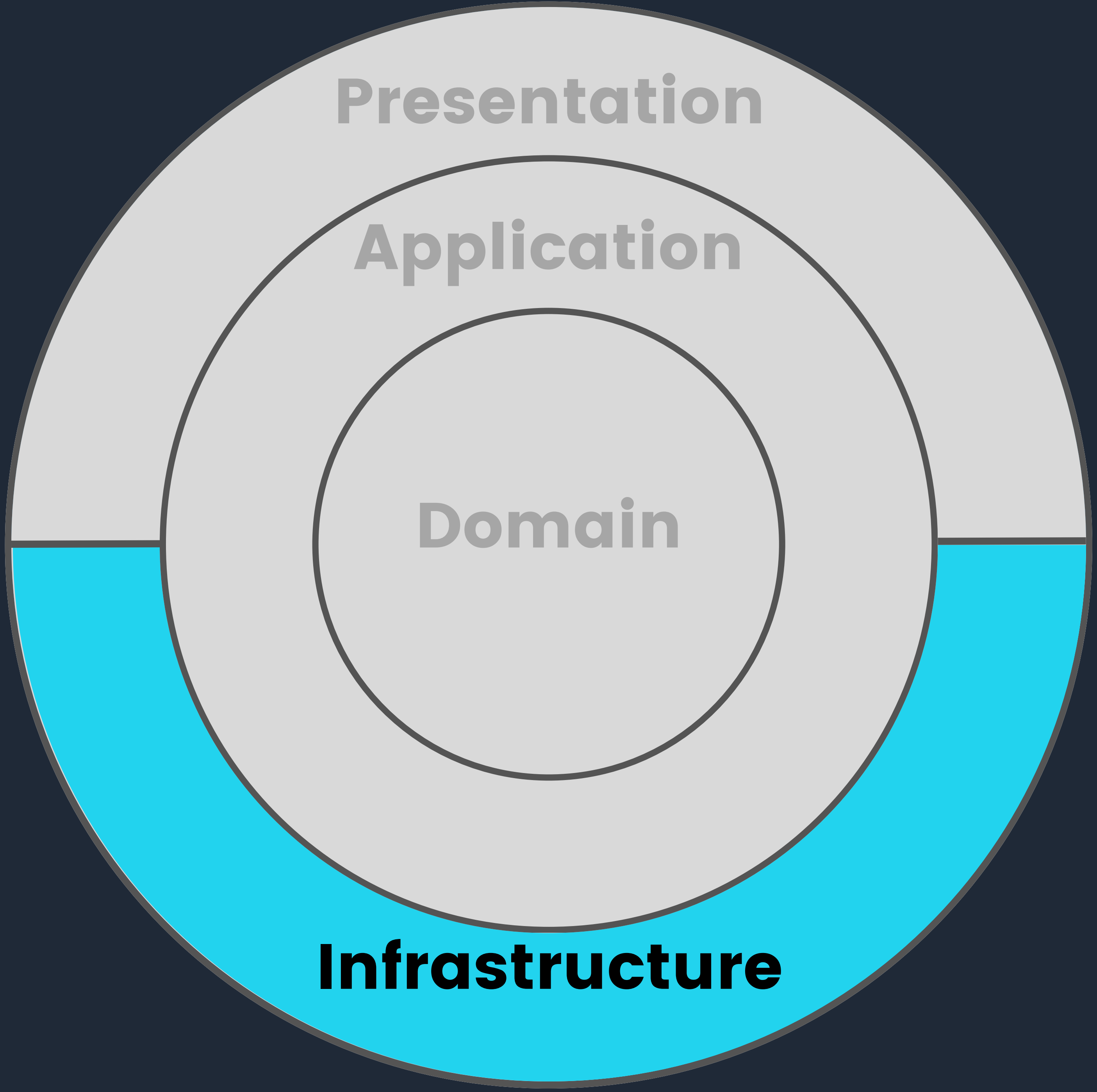
- Pros:
  - Single responsibility principle
  - Interface segregation principle
  - Decorator pattern
  - Loose coupling
- Cons:
  - Indirection

CQRS


API
1. Query
4. Result
1. Command
Query handler Dapper
Command handler DDD + EF
2. Get entities
5. Entities
6. Invoke entity method
7. Save changes
Domain logic
2. SQL
3. Result
Repositories
3. SQL
4. Result
8. SQL
Database

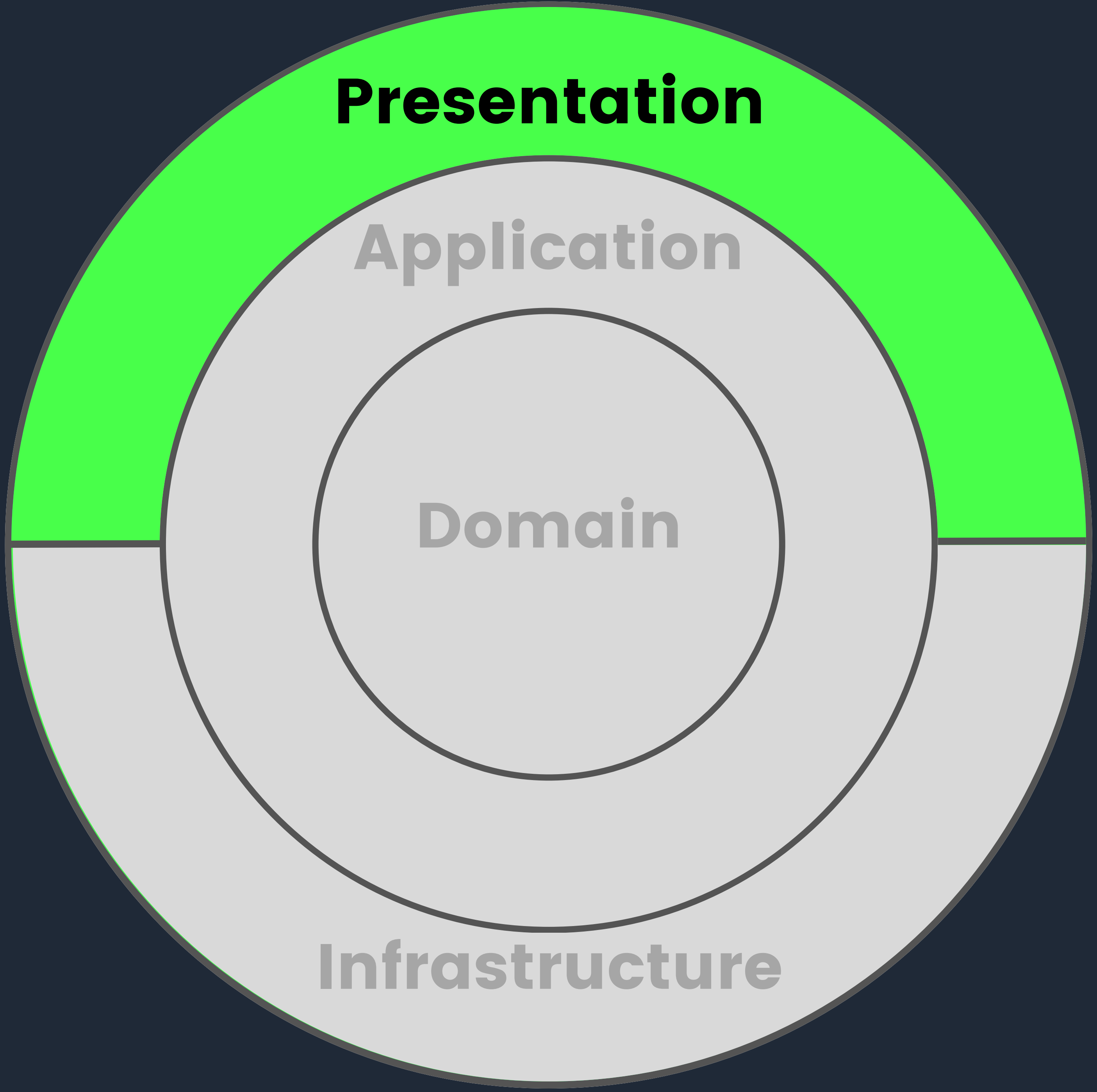
Infrastructure Layer

- EF Core
  - DbContext
  - Entity configurations
  - Repositories
- Optimistic concurrency
- Publishing Domain events



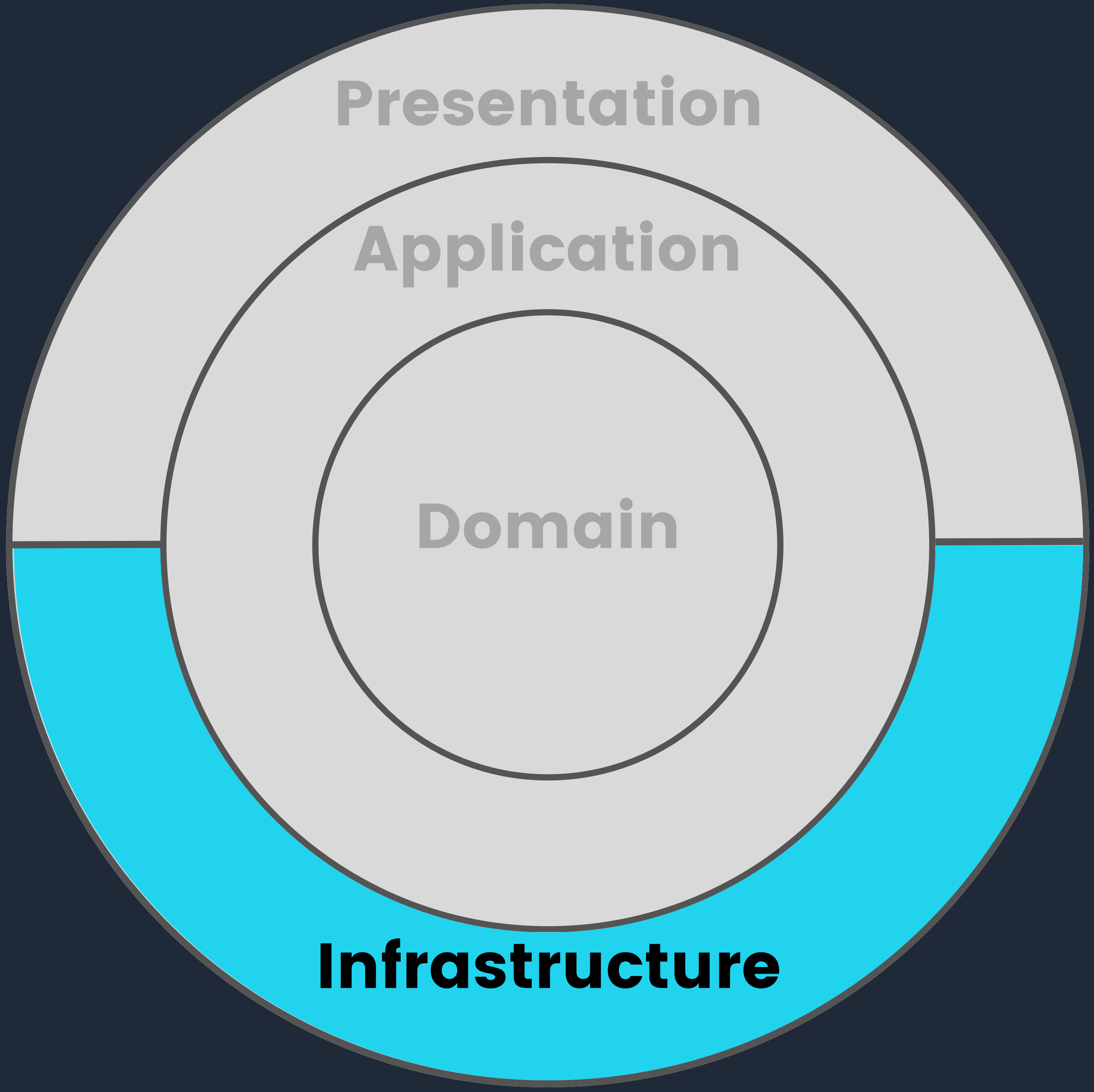
Presentation Layer

- Web API, .NET 7
- Controllers
- Middleware
- DI setup
- Docker Compose



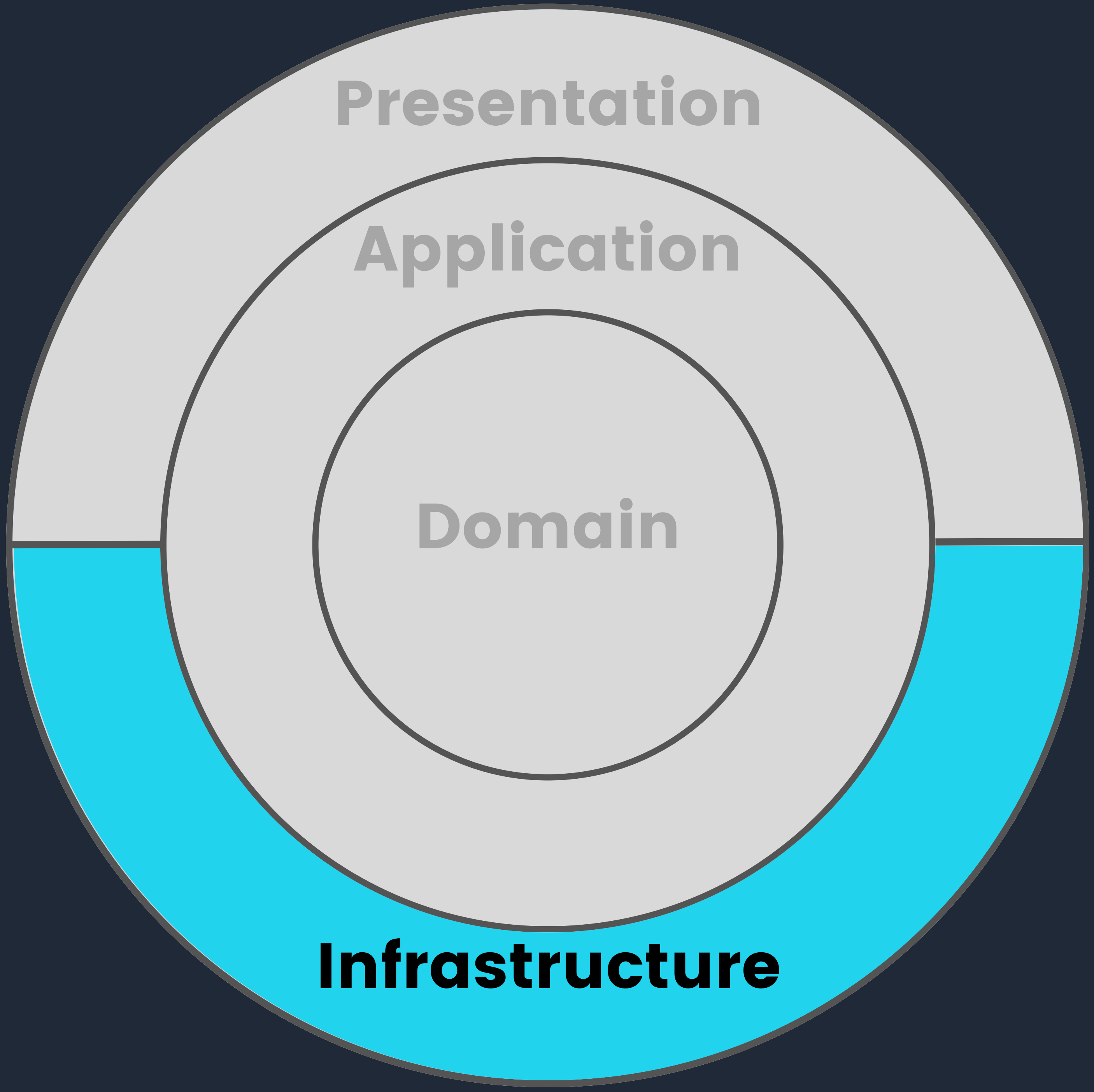
Authentication

- External Identity provider
- Keycloak
  - JWT Bearer auth
- .NET integration



Authorization

- Roles authorization
- Permissions authorization
- Resource-based authorization



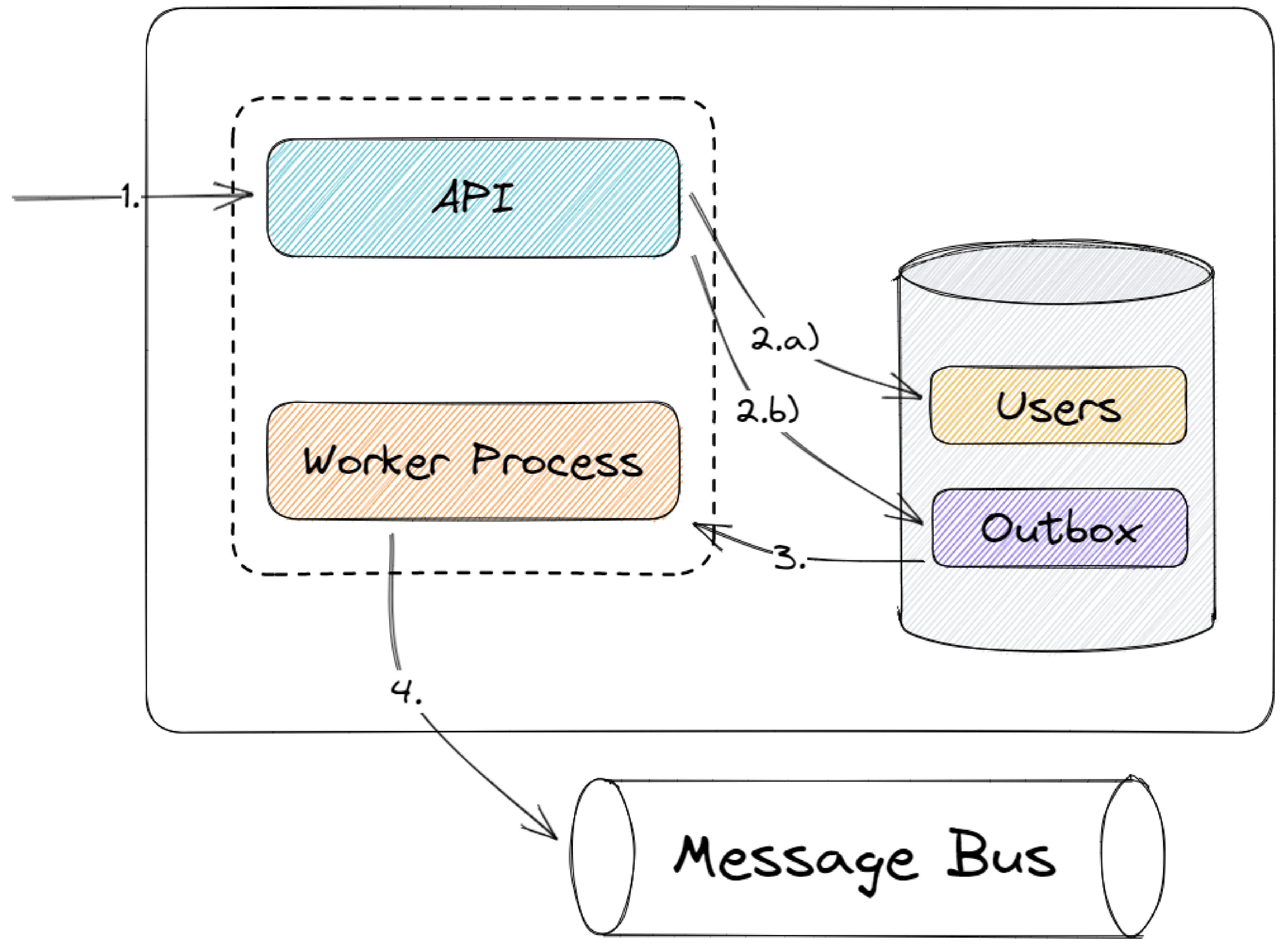
Advanced Topics

- Structured logging
- Distributed caching
- Health checks
- API Versioning
- Background jobs
  - Outbox pattern
- Minimal APIs

API Versioning

- Types of API Versioning
  - Query parameter
  - Header
  - URL
- Breaking changes
  - Agreed upon standard

Outbox Pattern



1. API request to register User

2.a) Save User to database
2.b) Save message to Outbox

3. Worker process polls the Outbox

4. Publish the Outbox message

Testing

- What should we test?
- Unit tests
  - Domain
  - Application
- Integration tests
- Functional tests
- Architecture tests