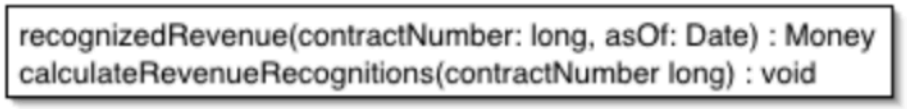
Transaction Script

Organizes business logic by procedures where each procedure handles a single request from the presentation.

For a full description see P of EAA page **110**



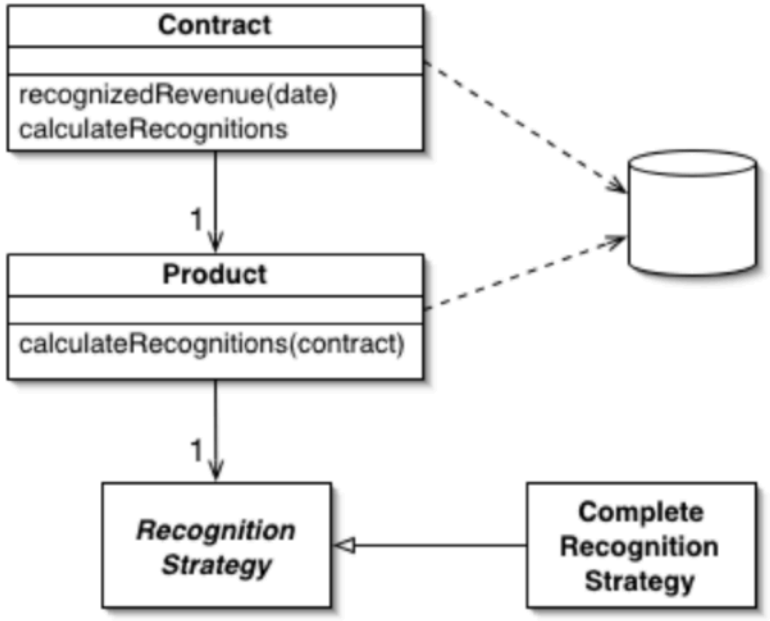
Most business applications can be thought of as a series of transactions. A transaction may view some information as organized in a particular way, another will make changes to it. Each interaction between a client system and a server system contains a certain amount of logic. In some cases this can be as simple as displaying information in the database. In others it may involve many steps of validations and calculations.

A Transaction Script organizes all this logic primarily as a single procedure, making calls directly to the database or through a thin database wrapper. Each transaction will have its own Transaction Script, although common subtasks can be broken into subprocedures.

Domain Model

An object model of the domain that incorporates both behavior and data.

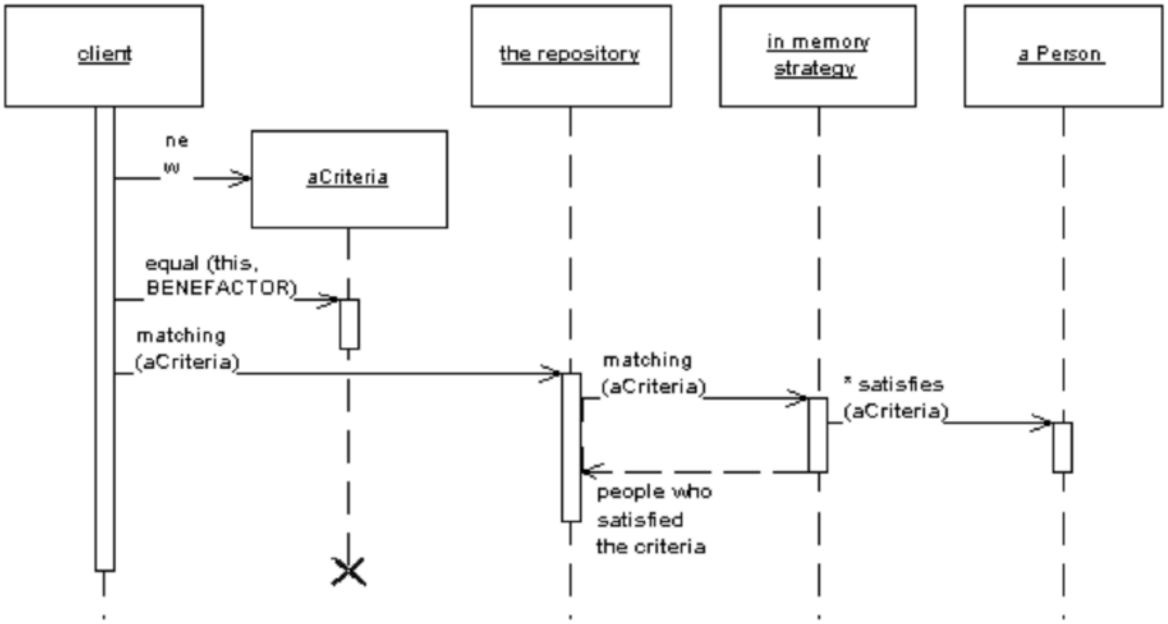
For a full description see P of EAA page **116**



At its worst business logic can be very complex. Rules and logic describe many different cases and slants of behavior, and it's this complexity that objects were designed to work with. A Domain Model creates a web of interconnected objects, where each object represents some meaningful individual, whether as large as a corporation or as small as a single line on an order form.

Repository

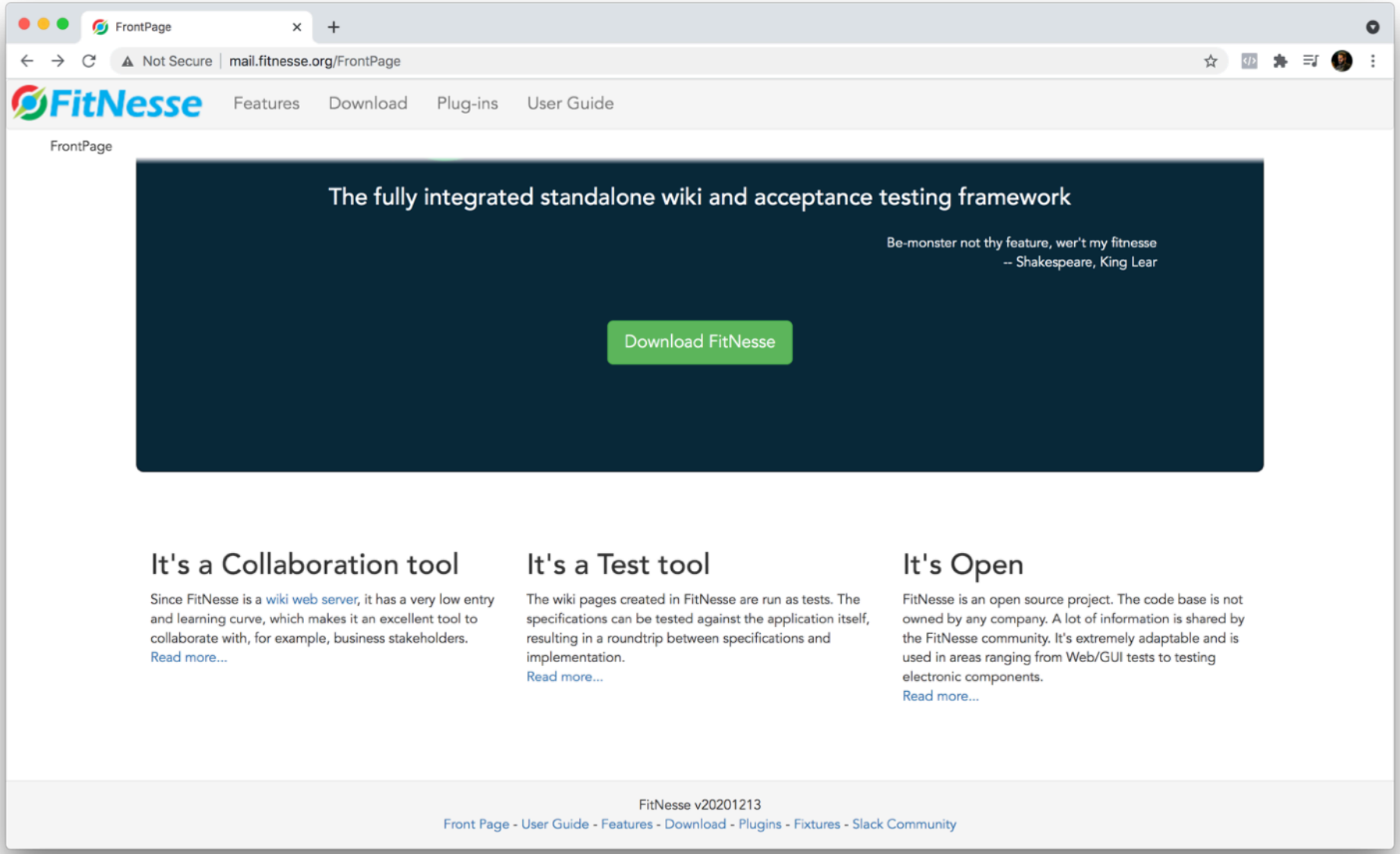
Mediates between the domain and data mapping layers using a collection-like interface for accessing domain objects.



A system with a complex domain model often benefits from a layer, such as the one provided by Data Mapper, that isolates domain objects from details of the database access code. In such systems it can be worthwhile to build another layer of abstraction over the mapping layer where query construction code is concentrated. This becomes more important when there are a large number of domain classes or heavy querying. In these cases particularly, adding this layer helps minimize duplicate query logic.

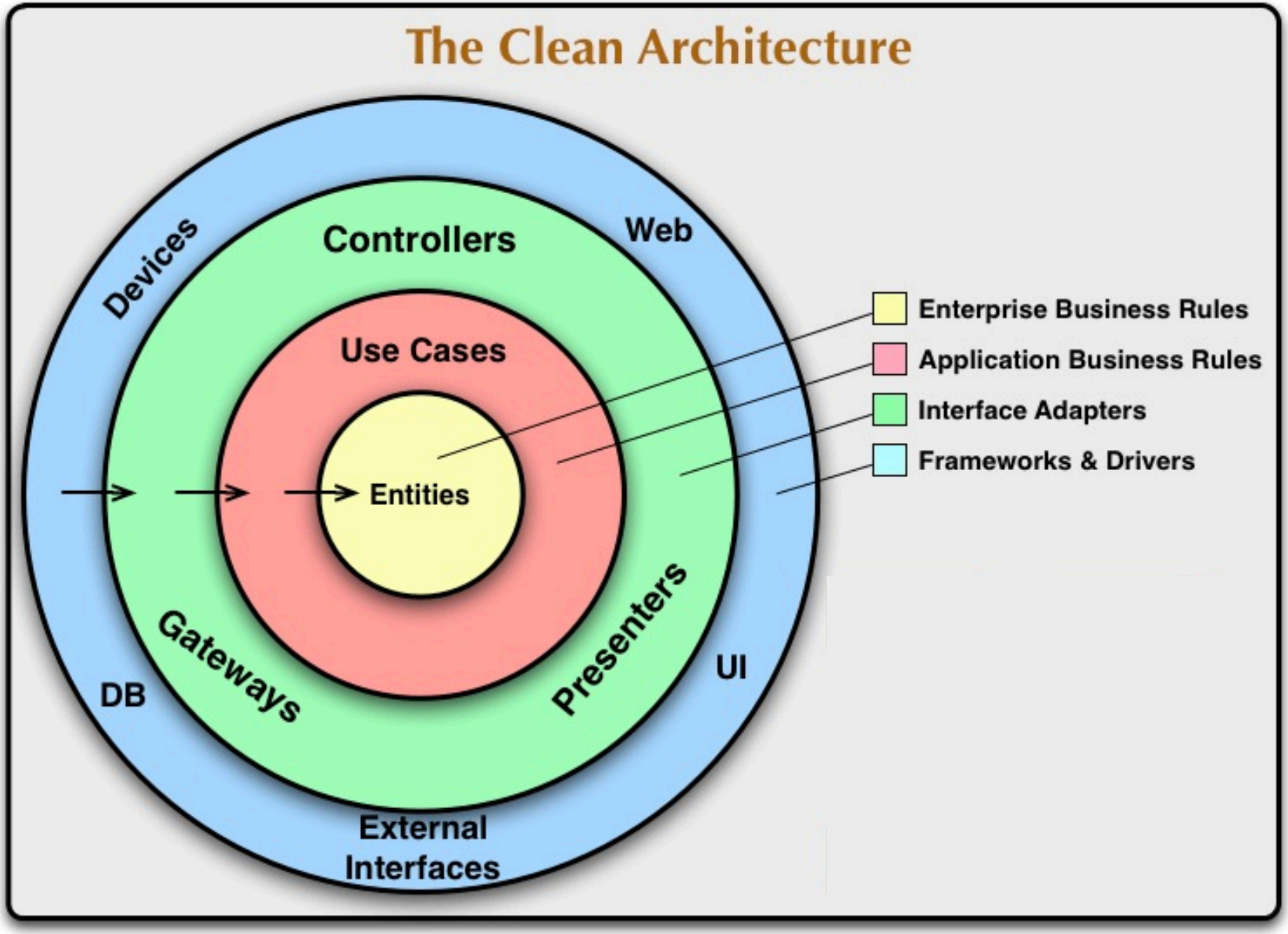

Clean Architecture

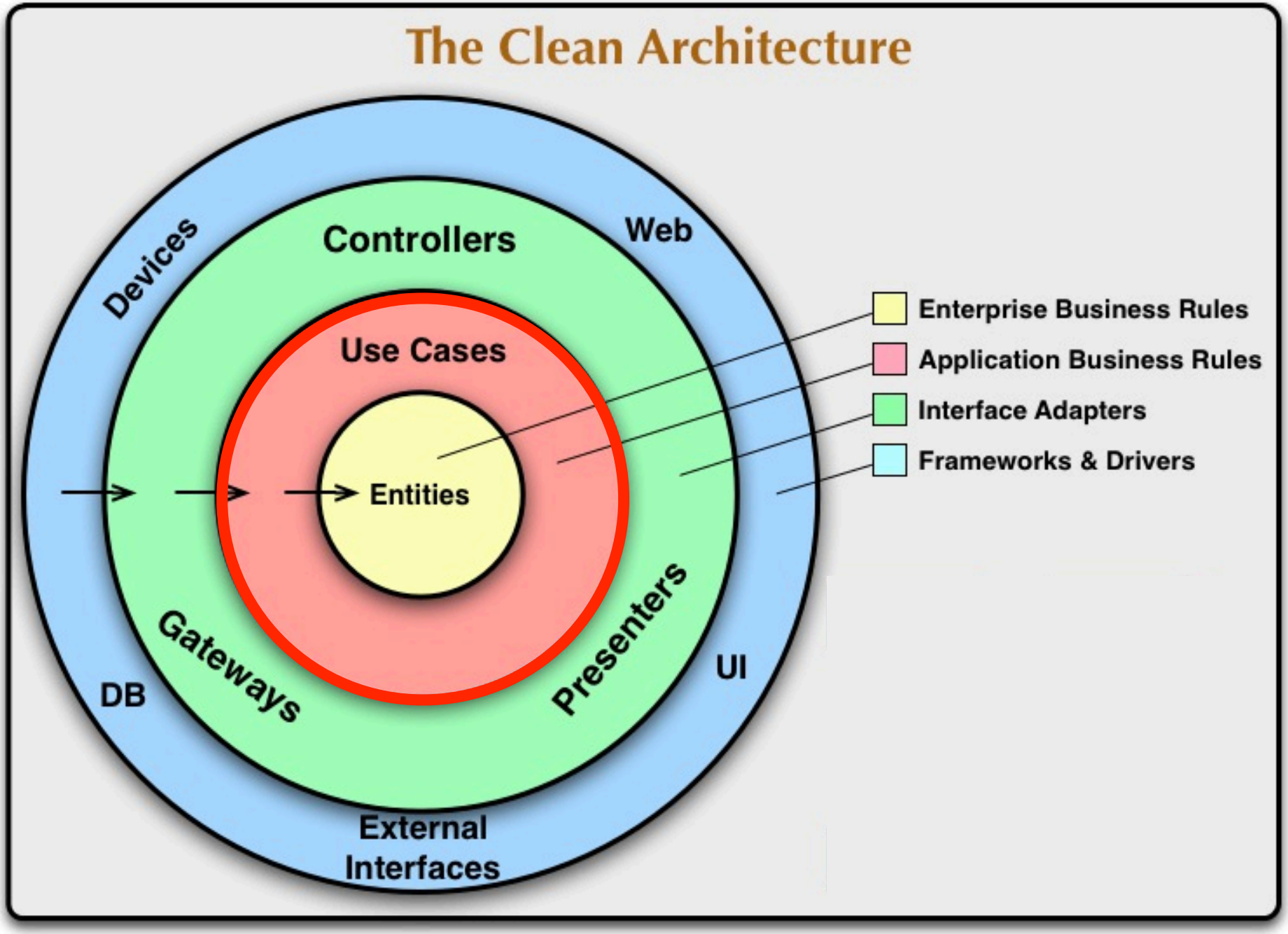
A Clean Architecture é um modelo que tem como objetivo o desacoplamento entre as regras de negócio, ou domínio, da aplicação e os recursos externos como frameworks e banco de dados


FrontPage
Not Secure | mail.fitnesse.org/FrontPage
FitNesse
Features
Download
Plug-ins
User Guide
FrontPage
The fully integrated standalone wiki and acceptance testing framework
Be-monster not thy feature, wer't my fitnesse
-- Shakespeare, King Lear
Download FitNesse
It's a Collaboration tool
Since FitNesse is a wiki web server, it has a very low entry and learning curve, which makes it an excellent tool to collaborate with, for example, business stakeholders.
Read more...
It's a Test tool
The wiki pages created in FitNesse are run as tests. The specifications can be tested against the application itself, resulting in a roundtrip between specifications and implementation.
Read more...
It's Open
FitNesse is an open source project. The code base is not owned by any company. A lot of information is shared by the FitNesse community. It's extremely adaptable and is used in areas ranging from Web/GUI tests to testing electronic components.
Read more...
FitNesse v20201213
Front Page - User Guide - Features - Download - Plugins - Fixtures - Slack Community

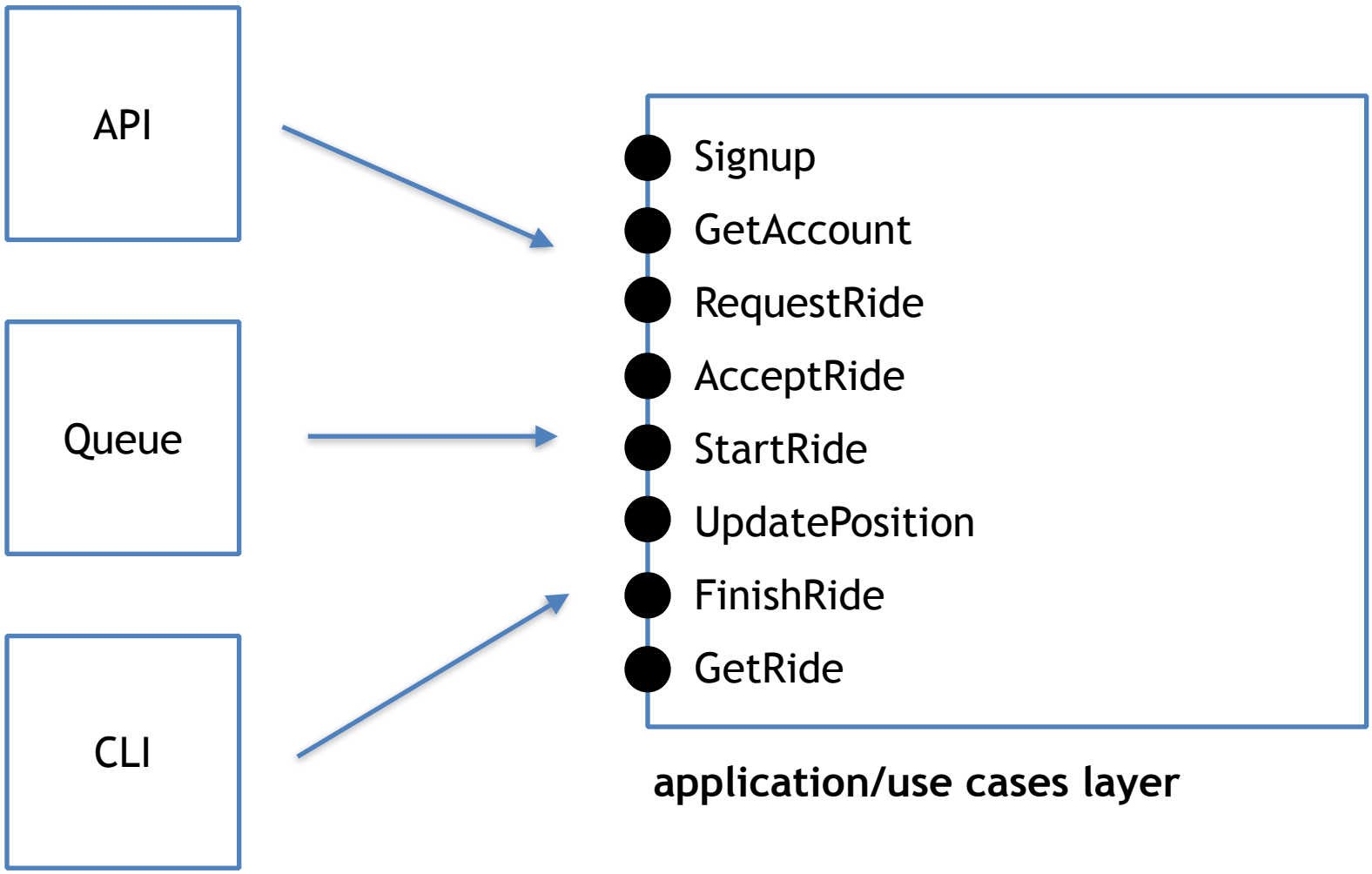
"The center of your application is not the database, nor is it one or more of the frameworks you may be using. The center of your application is the use cases of your application"

Robert Martin

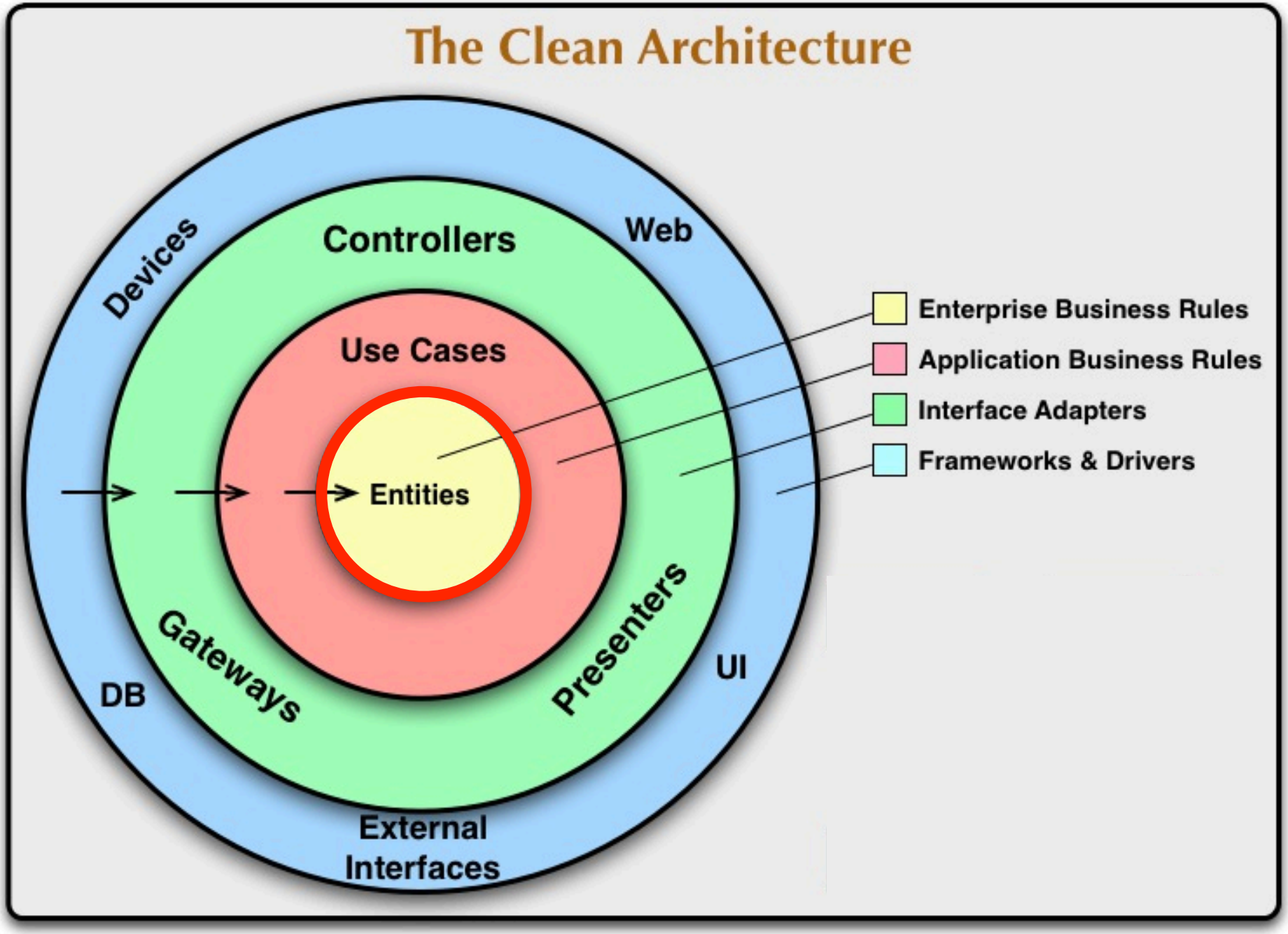

The Clean Architecture
Devices
Controllers
Web
Use Cases
Entities
Gateways
Presenters
DB
UI
External
Interfaces
Enterprise Business Rules
Application Business Rules
Interface Adapters
Frameworks & Drivers


The Clean Architecture
Devices
Controllers
Web
Use Cases
Entities
Gateways
Presenters
DB
UI
External
Interfaces
Enterprise Business Rules
Application Business Rules
Interface Adapters
Frameworks & Drivers

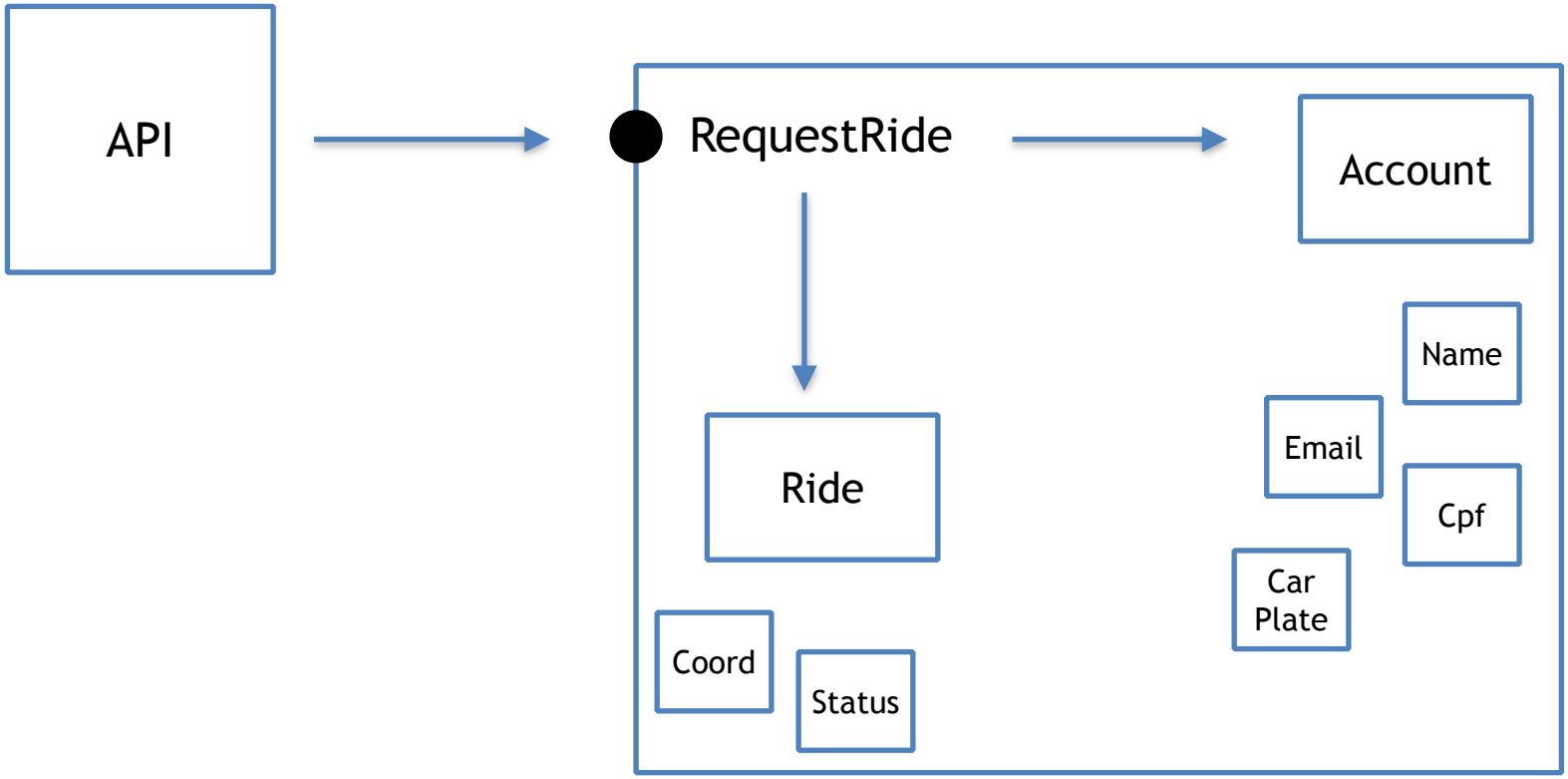
Os use cases expõe o comportamento demandado pelos drivers (atores), e orquestram as entidades e os recursos externos como banco de dados, APIs, Filas


API
Queue
CLI
Signup
GetAccount
RequestRide
AcceptRide
StartRide
UpdatePosition
FinishRide
GetRide
application/use cases layer


Caso de uso é diferente de CRUD?


The Clean Architecture
Devices
Controllers
Web
Use Cases
Entities
Gateways
Presenters
DB
UI
External
Interfaces
Enterprise Business Rules
Application Business Rules
Interface Adapters
Frameworks & Drivers

Entidades são responsáveis por abstrair as regras de negócio independentes, que podem ser desde um objeto com métodos até mesmo um conjunto de funções



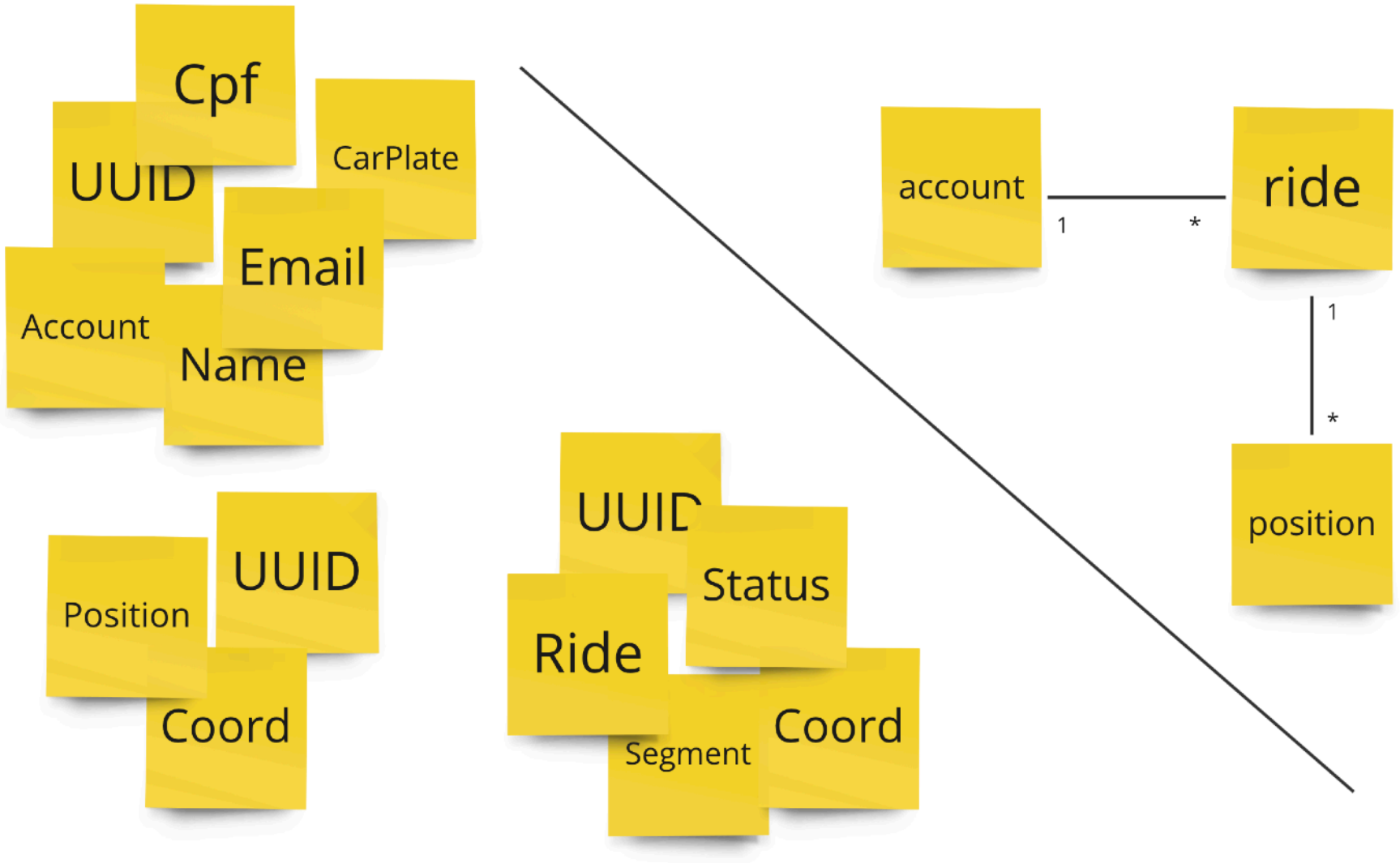
domain/entity layer

O que são regras de negócio independentes?

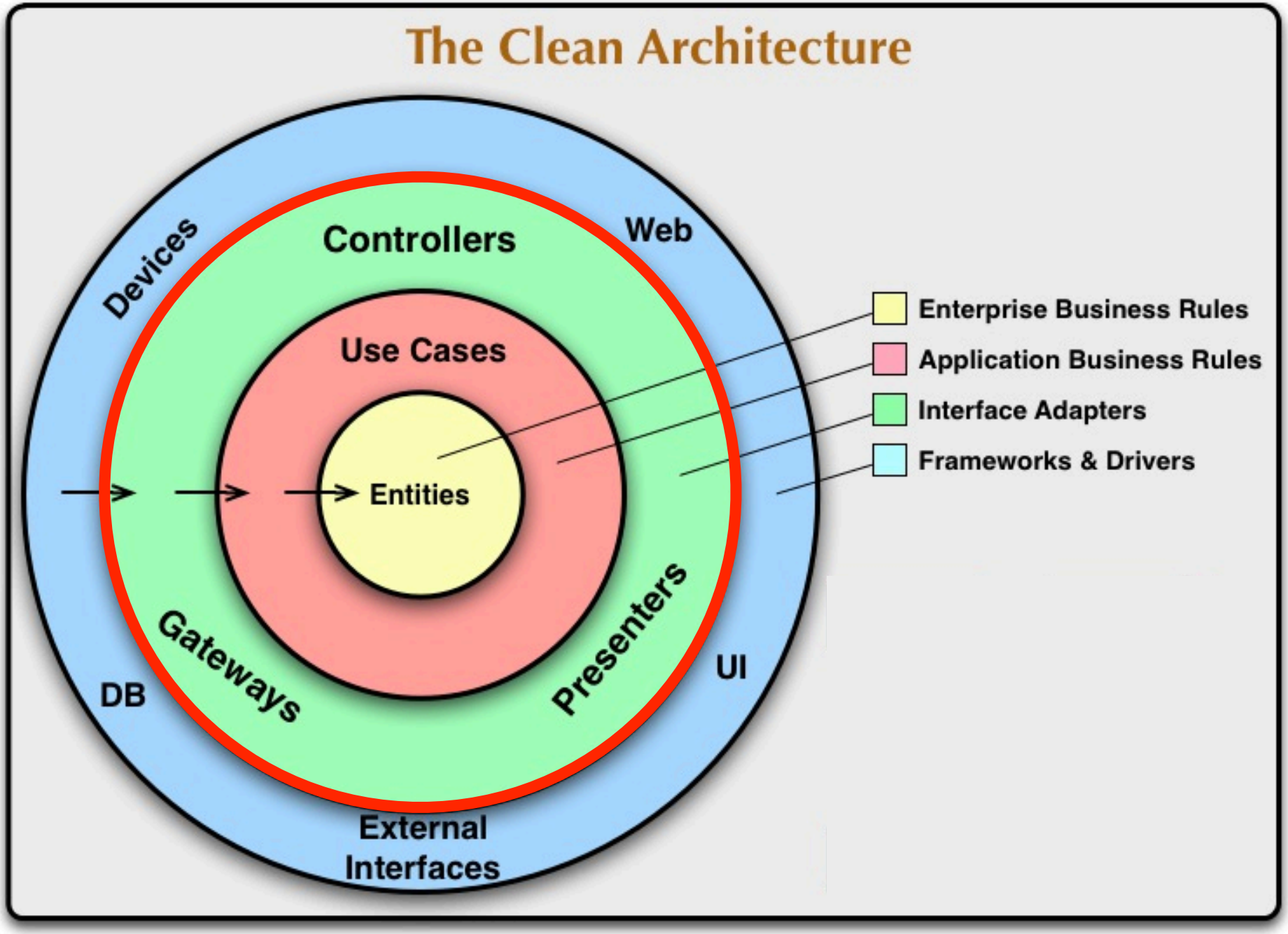
- A placa do carro é válida?
- Qual é a distância entre duas coordenadas?
- Quanto é a tarifa da corrida?
- Como é a mudança de status de uma corrida?

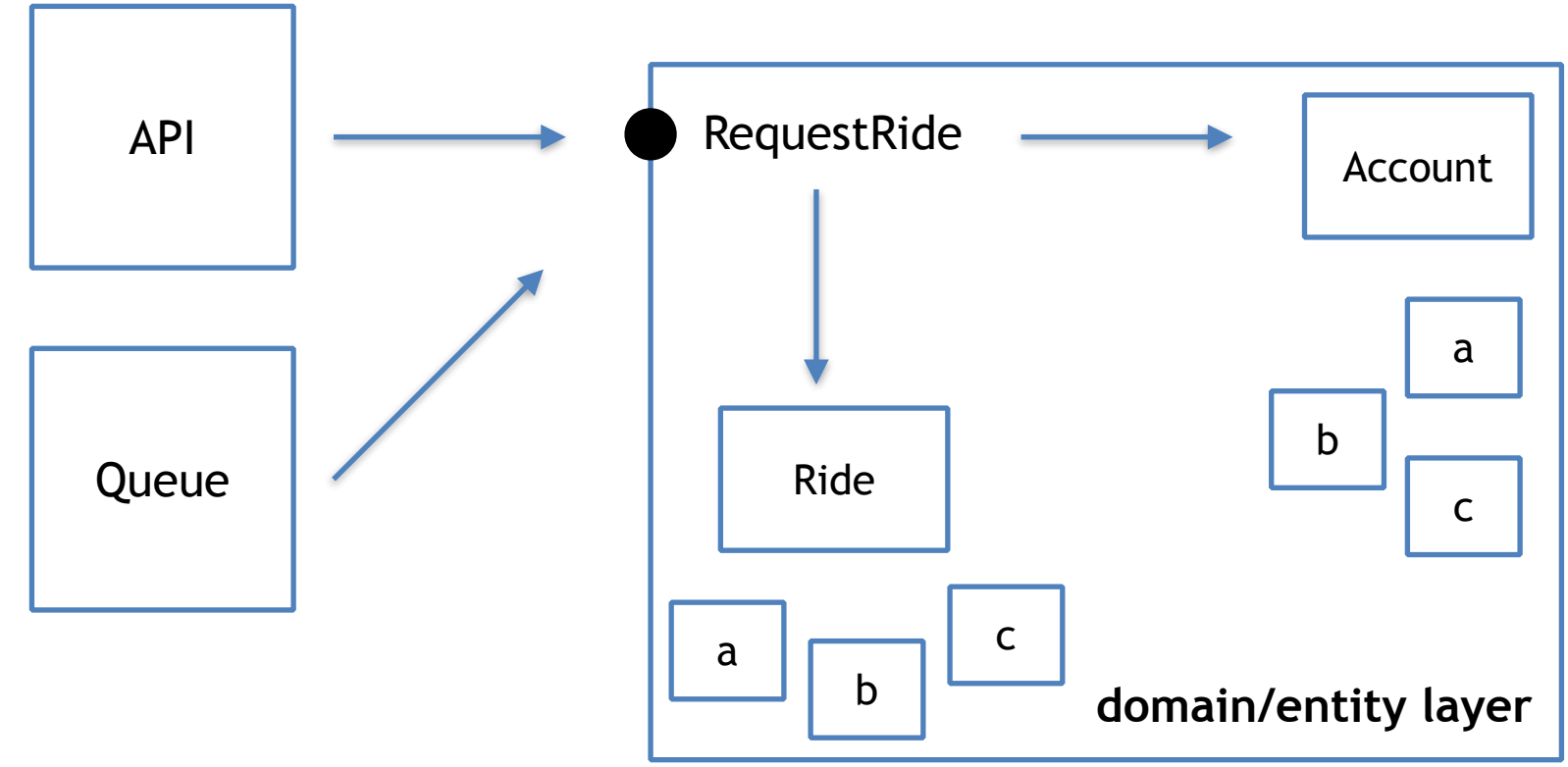
Essas entidades não são as mesmas que utilizamos em um ORM

Domain Objects vs. ORM Objects




O problema do domínio anêmico


The Clean Architecture
Devices
Web
Controllers
Use Cases
Entities
Gateways
Presenters
DB
UI
External
Interfaces
Enterprise Business Rules
Application Business Rules
Interface Adapters
Frameworks & Drivers



infra/interface adapters layer

application/use case layer


Os interface adapters fazem a ponte entre
os casos de uso e os recursos externos

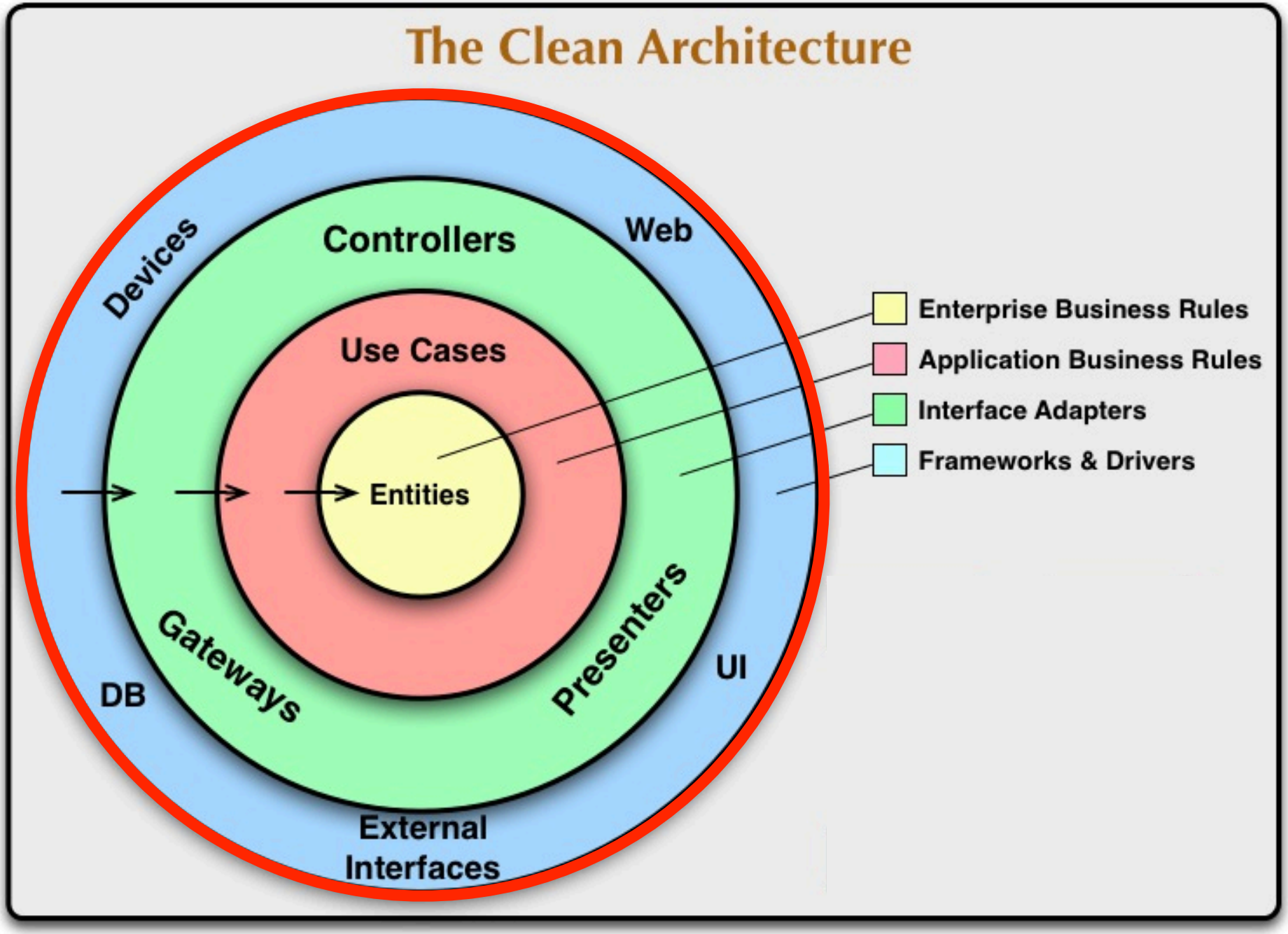
Tratamento de requisições e respostas HTTP, lidando com parâmetros

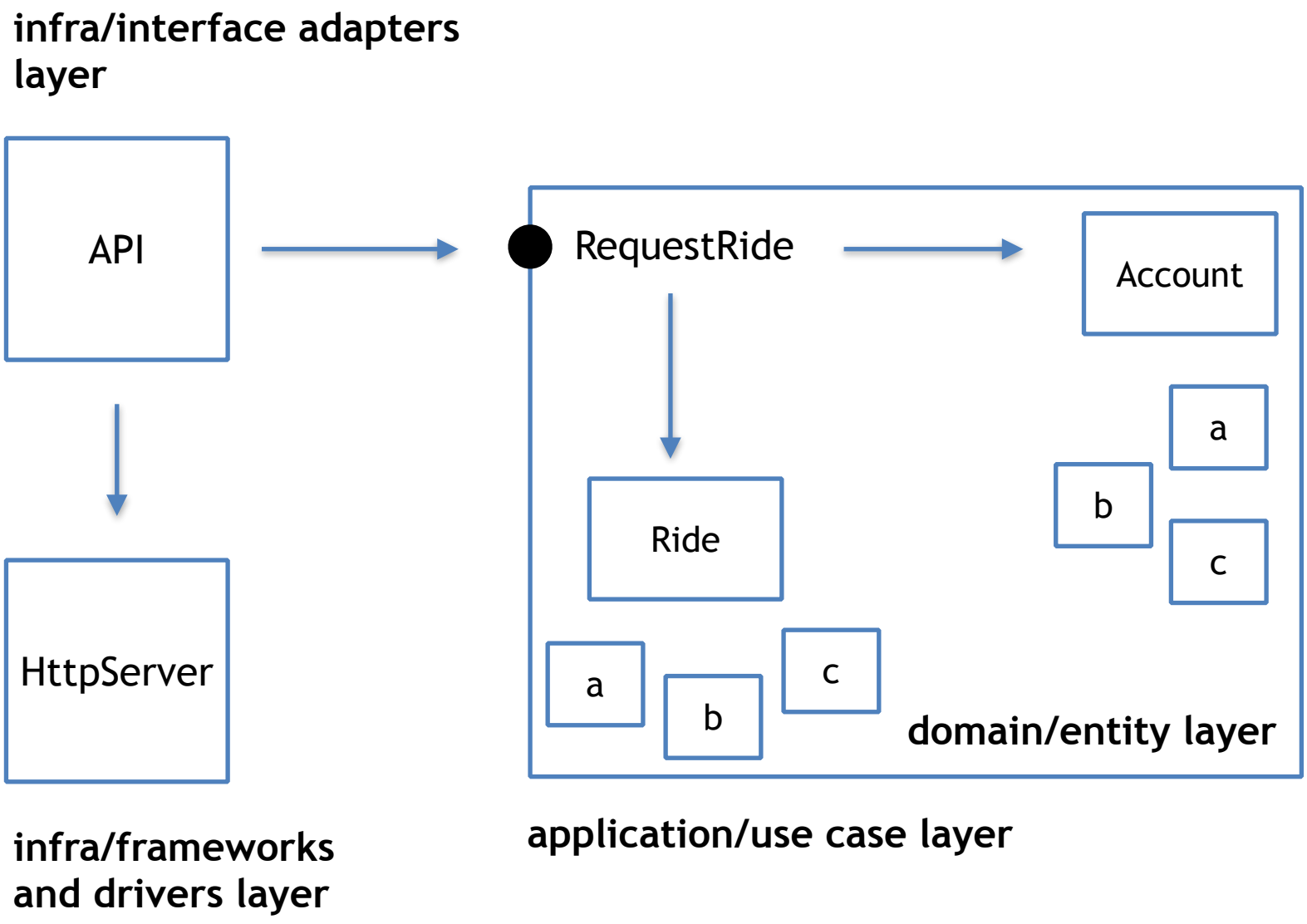
Acesso ao banco de dados, todo o código SQL pertence à esta camada

Integração com uma API externa

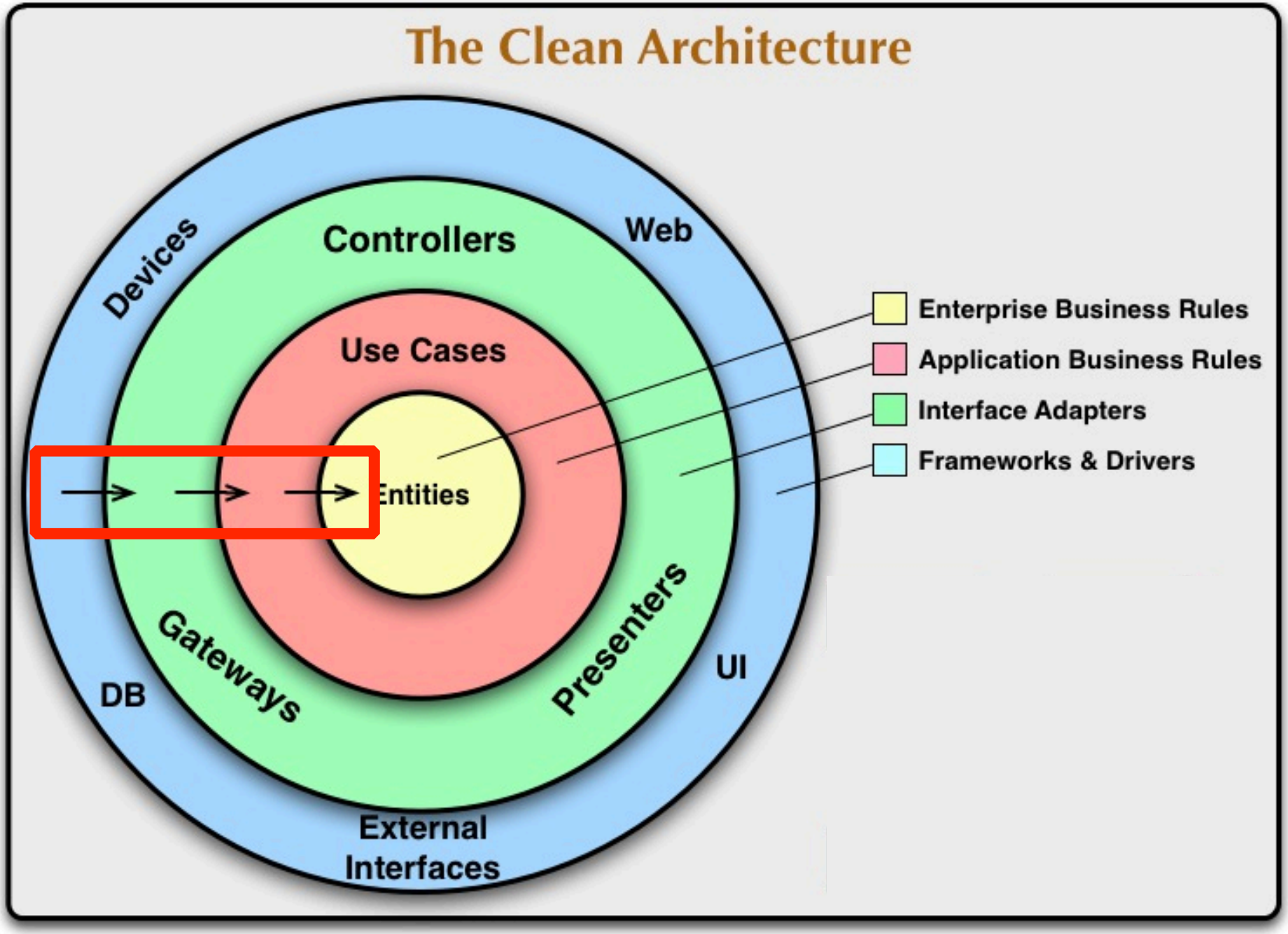
Escrita e leitura no sistema de arquivos

Conversão de dados para formatos específicos como CSV e PDF

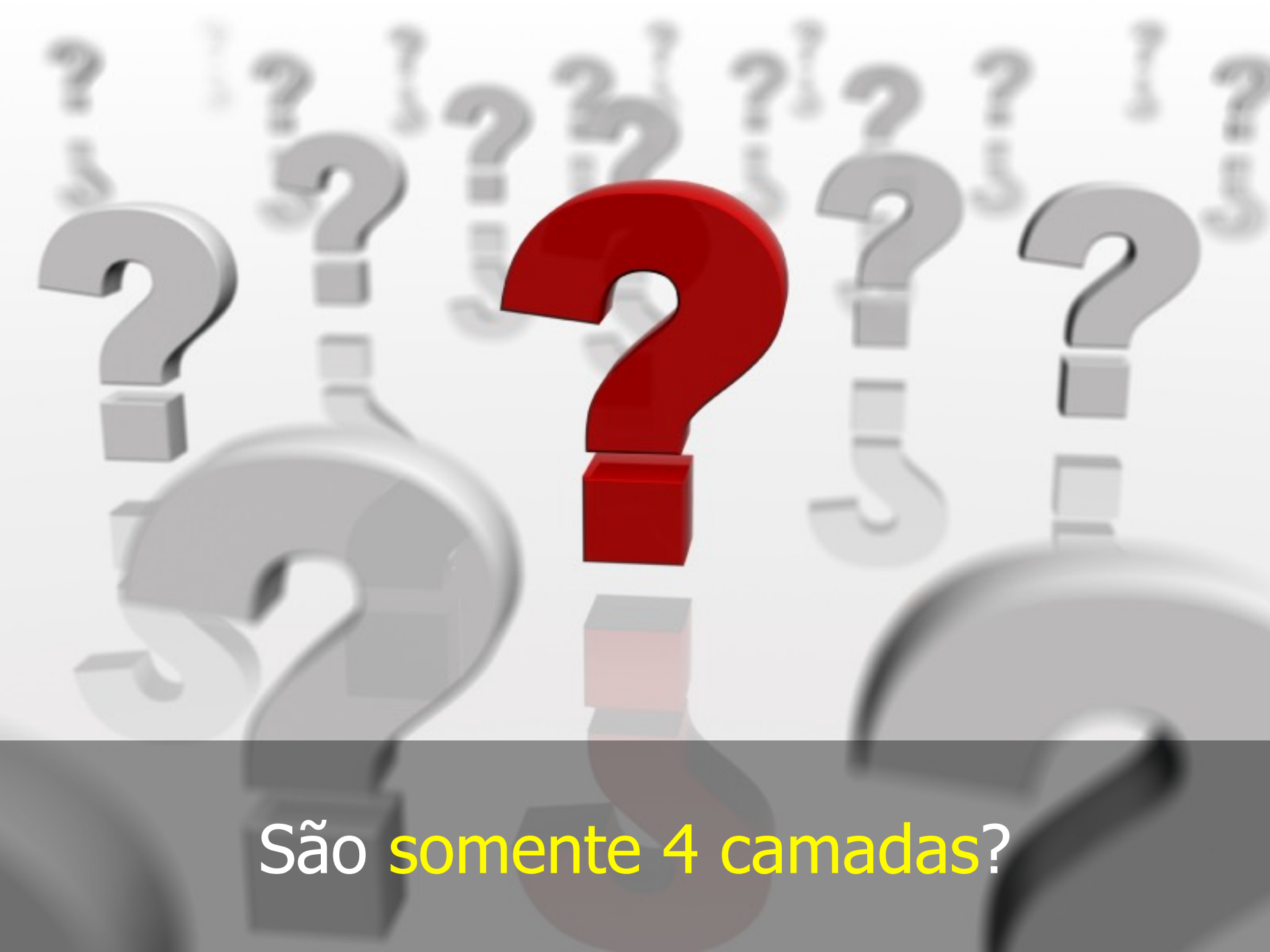

The Clean Architecture
Devices
Controllers
Web
Use Cases
Entities
Gateways
Presenters
DB
UI
External
Interfaces
Enterprise Business Rules
Application Business Rules
Interface Adapters
Frameworks & Drivers


infra/interface adapters layer
API
RequestRide
Account
a
b
c
Ride
a
b
c
domain/entity layer
HttpServer
infra/frameworks and drivers layer
application/use case layer

Por fim, os frameworks and drivers são o nível mais baixo de abstração, é a interação com a tecnologia, com os componentes que realizam a conexão com o banco de dados, as requisições HTTP, a interação com o sistema de arquivos ou o acessa aos recursos do sistema operacional


The Clean Architecture
Devices
Controllers
Web
Use Cases
Entities
Gateways
Presenters
DB
UI
External
Interfaces
Enterprise Business Rules
Application Business Rules
Interface Adapters
Frameworks & Drivers

Quem está dentro não conhece quem está fora, mas quem está fora conhece quem está dentro, as entidades não conhecem os use cases e esses não conhecem a implementação dos interface adapters, que não conhecem a implementação dos frameworks and drivers


São somente 4 camadas?

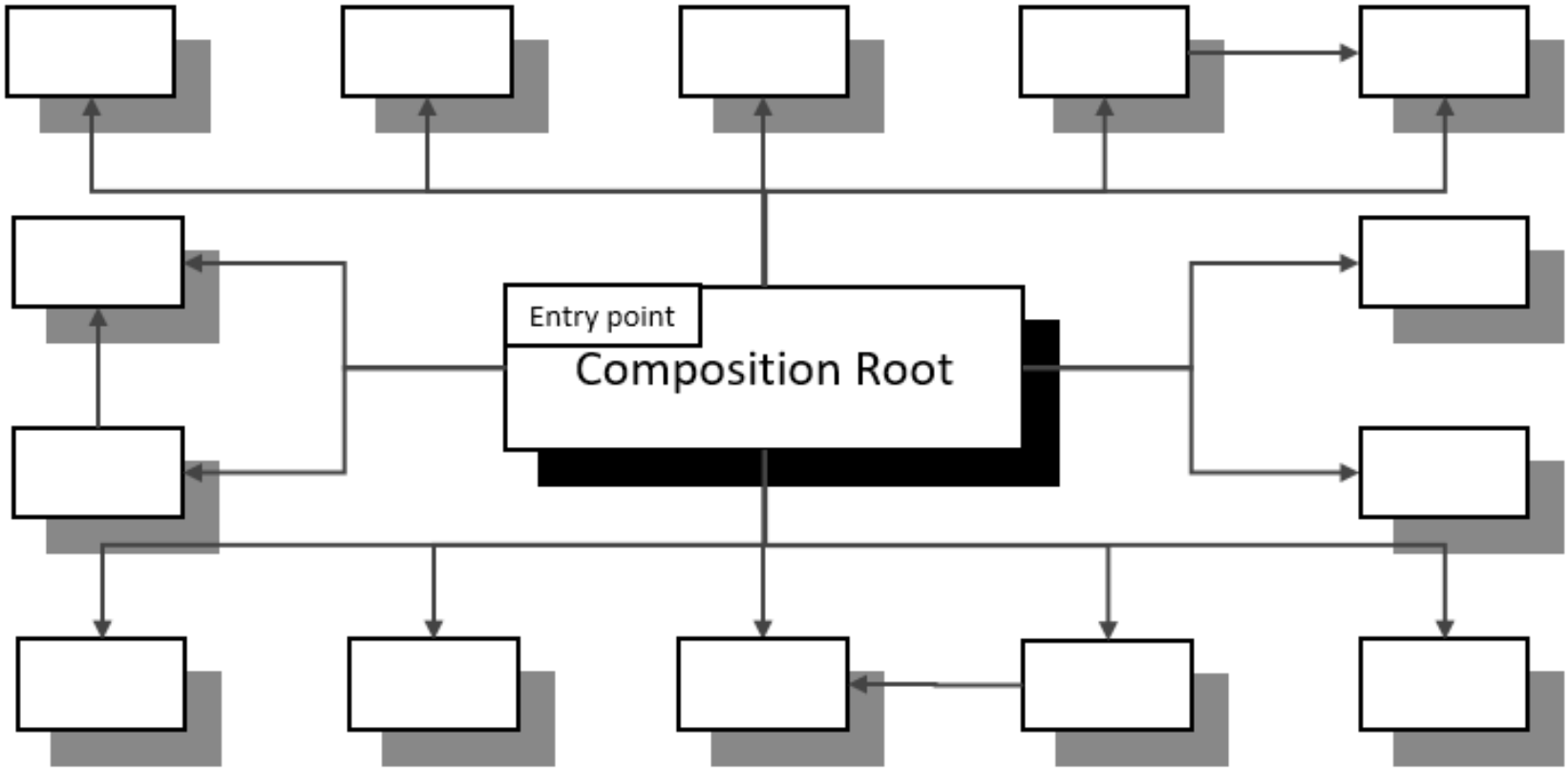
Uma camada é uma fronteira lógica entre um conjunto de componentes, que tem uma responsabilidade bem definida


As camadas não tem necessariamente
relação física com as pastas


Como incializamos a aplicação?

O main é o ponto de entrada da aplicação (HTTP, CLI, UI, Testes), é lá que as fábricas e estratégias são inicializadas e as injeções de dependência são realizadas durante a inicialização

When composing an application from many loosely coupled classes, **the composition should take place as close to the application's entry point as possible**. The Main method is the entry point for most application types. The Composition Root composes the object graph, which subsequently performs the actual work of the application


Entry point
Composition Root