Arquitetura do SommusGestor

Motivações, organização e características

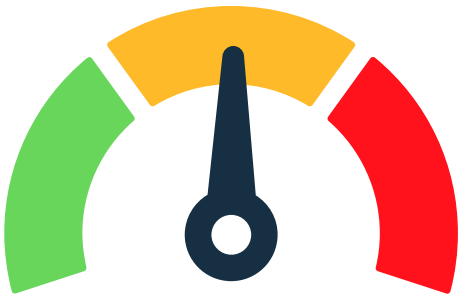
Dia 1 - Alinhando as expectativas

O que NÃO teremos neste treinamento?

Lições detalhadas sobre as diversas arquituras existentes.
Debate sobre qual arquitetura funciona ou não.
História das arquiteturas e seus fundamentos.
Implementação de códigos.

E o que teremos?

Breve introdução geral sobre arquiteturas.
Dica de estudo inicial.
Qual arquitetura o SommusGestor utiliza.
Como é a arquitetura do SommusGestor.

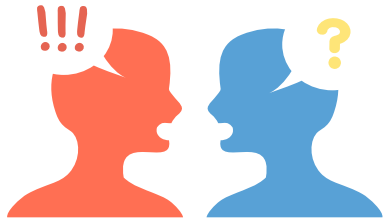


O que é a Arquitetura de Sofware?

“É uma estrutura que define como os diferentes componentes do software se **encaixam**, como se **comunicam** e como **funcionam juntos** para criar um sistema completo.”

Encaixar

Comunicar



Funcionar Junto

Uma boa analogia para pensar arquitetura

Arquitetura de uma Casa

Arquitetura de Software

Alguns comparativos

Organização;

Padrões e Princípios;

Comunicação entre componentes;

Escalabilidade e Manutenção;

Importância da Arquitetua

Organização e Estruturação;

Escalabilidade;

Manutenção;

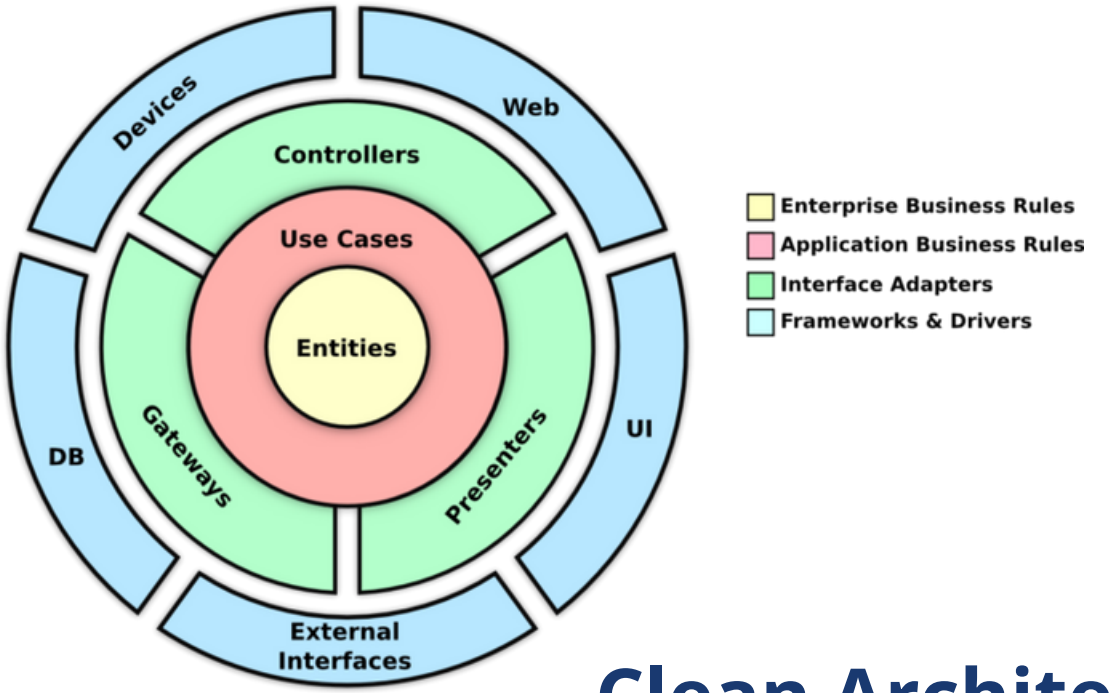
Reusabilidade;

Desempenho;

Segurança;

Dica de estudo

Após conhecer melhor a Arquitetura Limpa o meu entendimento mudou completamente. Tudo fez sentido a partir dali. Então deixo como recomendação, como passo inicial, para quando sentirem ser o momento de aperfeiçoar esse conhecimento.



Clean Architecture

Vamos ao que interessa?

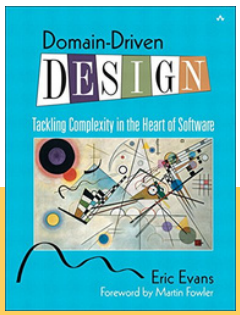
Arquitetura do SommusGestor

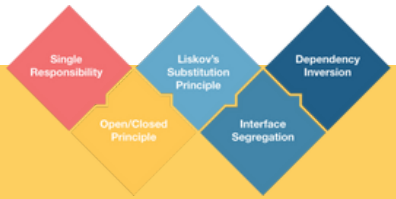
Arquitetura em camadas é a que melhor representa hoje a estrutura do SommusGestor.

A inspiração veio dos modelos citados nos livros sobre o “Domain Driven Design - DDD” mas de uma forma mais simplificada.

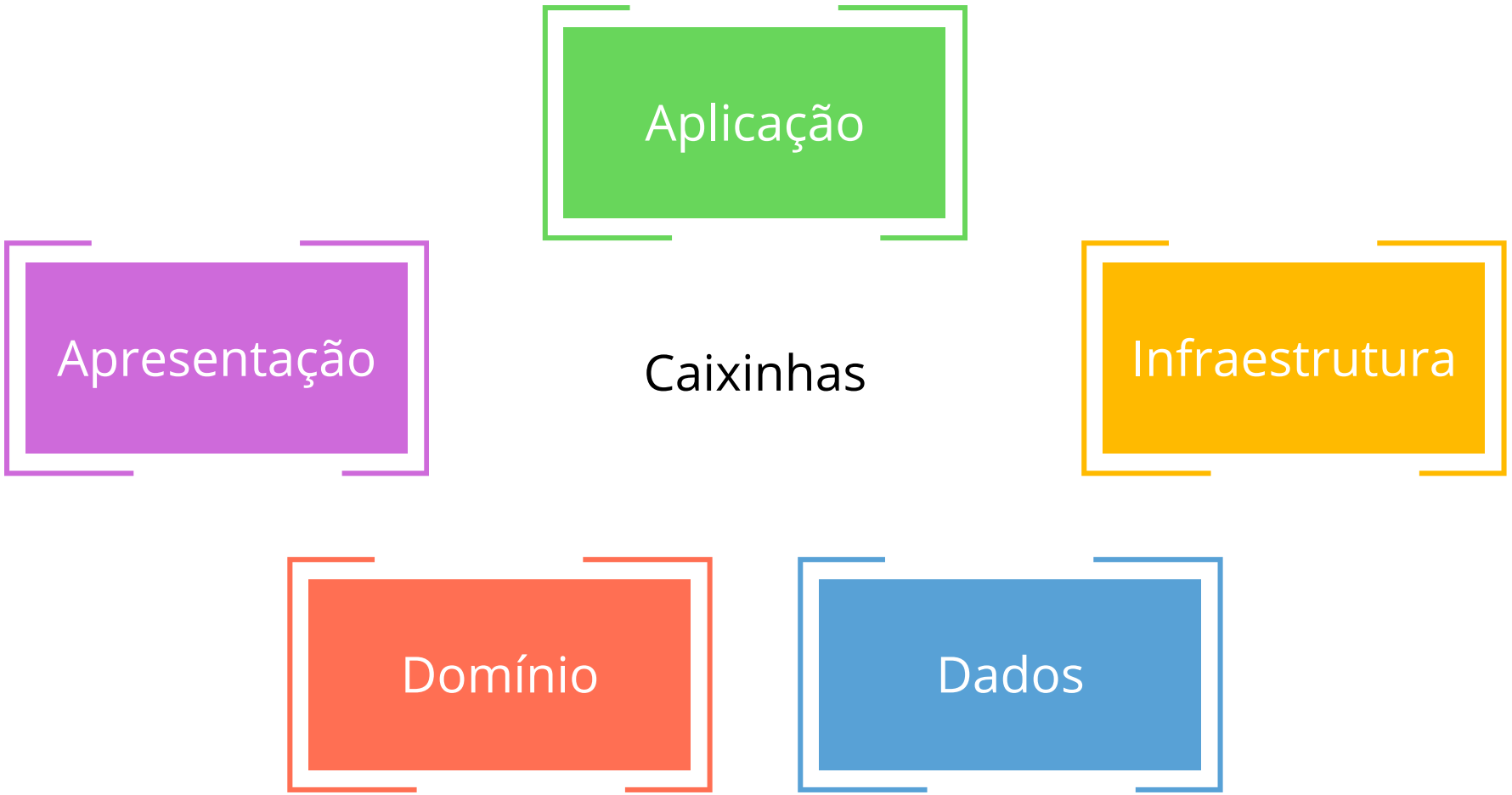
Mesmo o DDD não sendo uma arquitetura ele trouxe estratégias de organização que ajudaram a evoluir o projeto.

Contextos Delimitados, Camadas independentes e SOLID são alguns pilares que ajudam o projeto a se manter organizado.

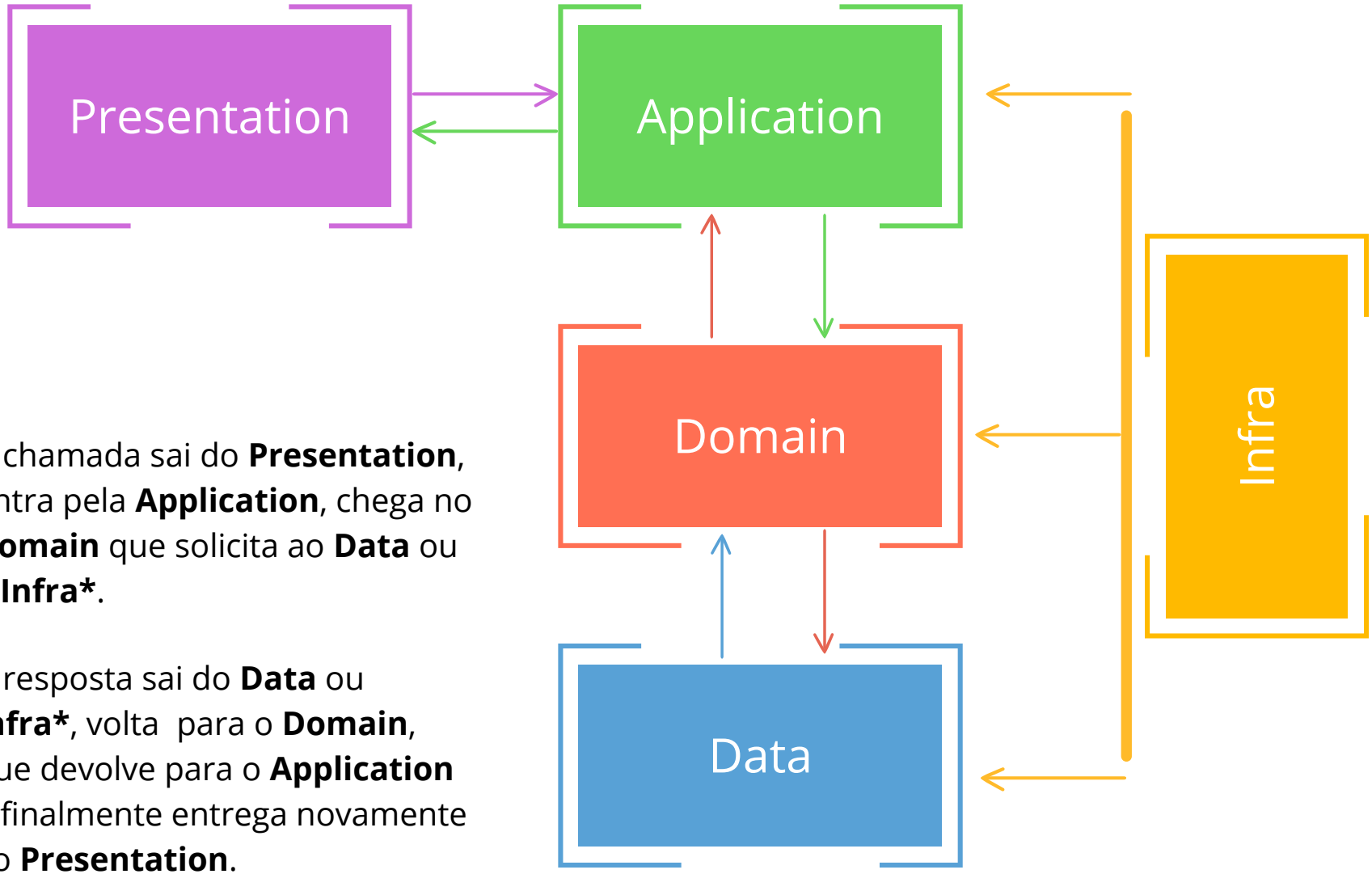




Camadas


Aplicação
Apresentação
Caixinhas
Infraestrutura
Domínio
Dados

Comunicação entre camadas

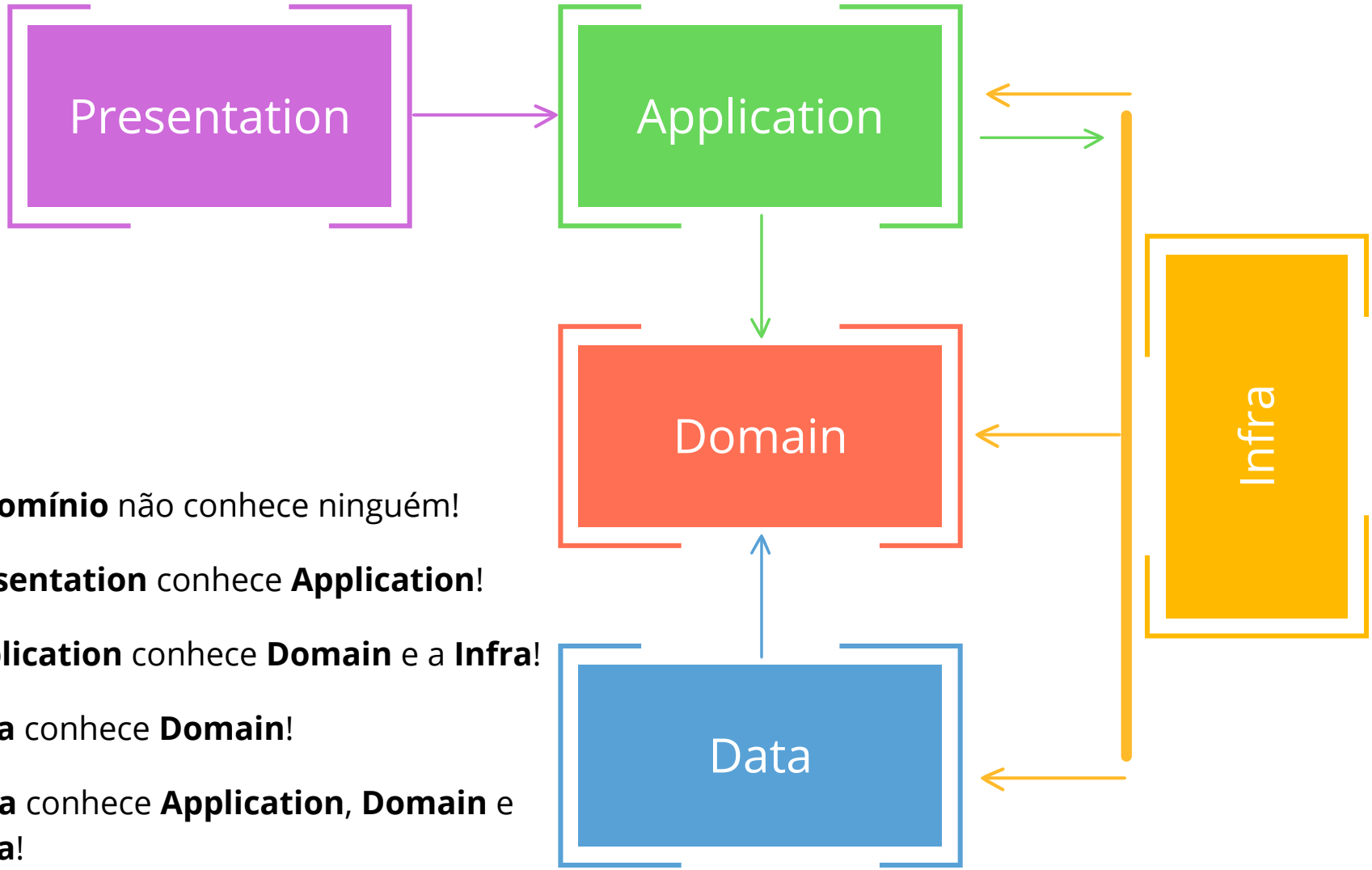


A chamada sai do **Presentation**, entra pela **Application**, chega no **Domain** que solicita ao **Data** ou a **Infra***.

A resposta sai do **Data** ou **Infra***, volta para o **Domain**, que devolve para o **Application** e finalmente entrega novamente ao **Presentation**.

***Comunicação especial, para a didática pode ser ignorada por hora.**

Quem conhece quem?



O **Domínio** não conhece ninguém!

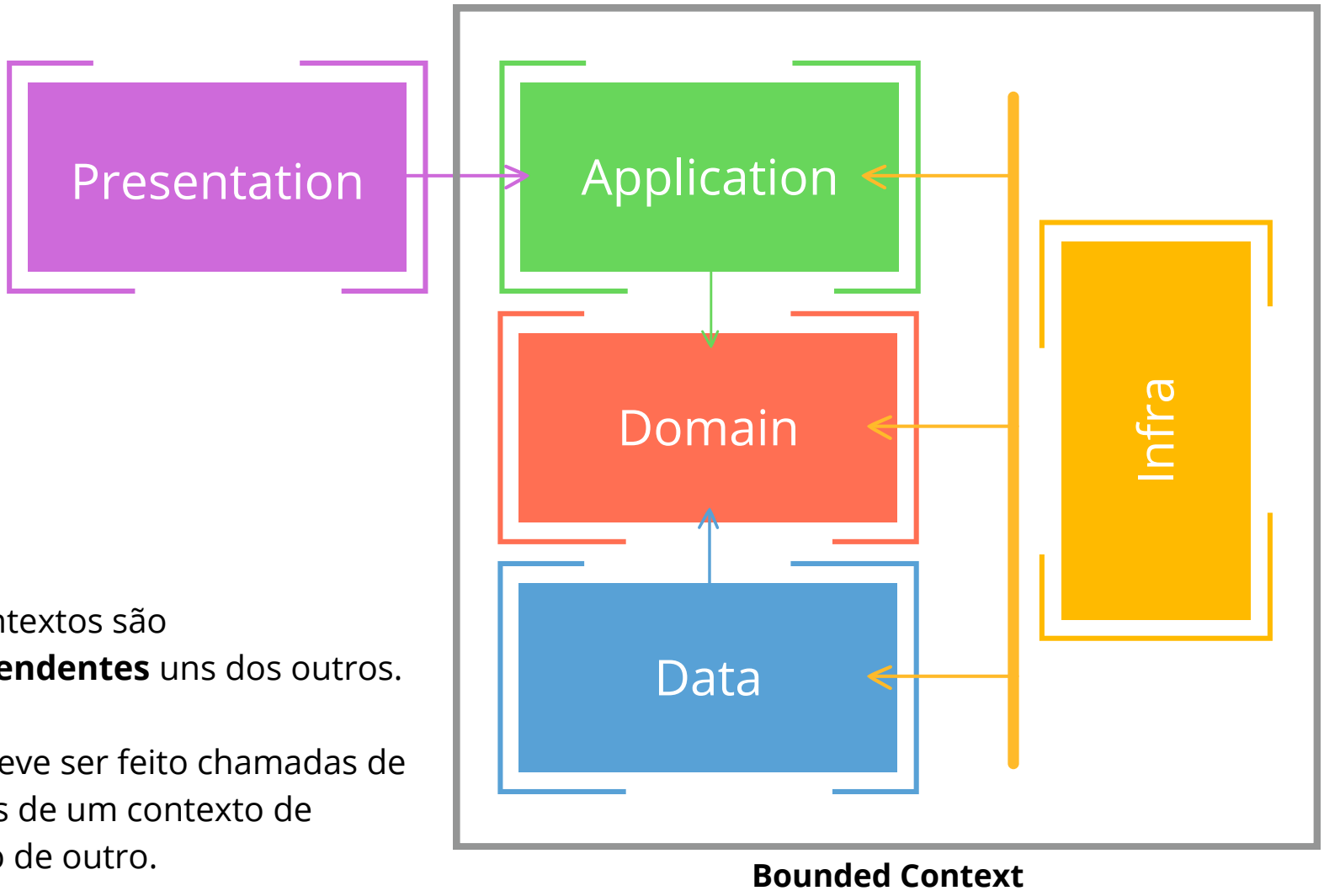
Presentation conhece **Application**!

Application conhece **Domain** e a **Infra**!

Data conhece **Domain**!

Infra conhece **Application**, **Domain** e **Data**!

Contextos Delimitados

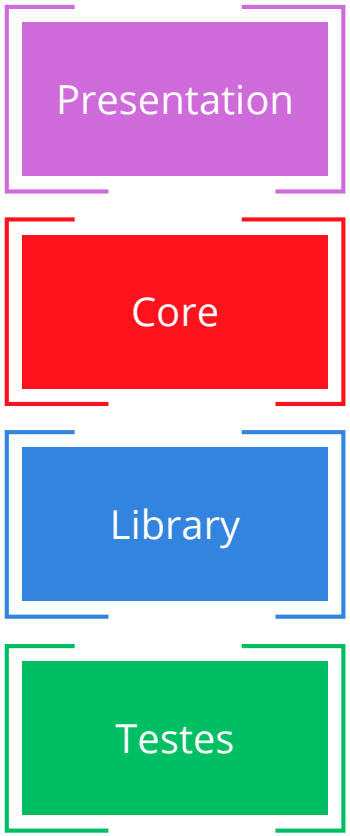


Os contextos são **independentes** uns dos outros.

Não deve ser feito chamadas de classes de um contexto de dentro de outro.

Projetos especiais

Alguns projetos dentro da Solução são especiais e fogem um pouco das regras apresentadas anteriormente. Isso foi definido para ser um facilitador, já que são regras que fazem sentido serem utilizadas em mais de um contexto e não deveriam ser reescritas em cada um deles.



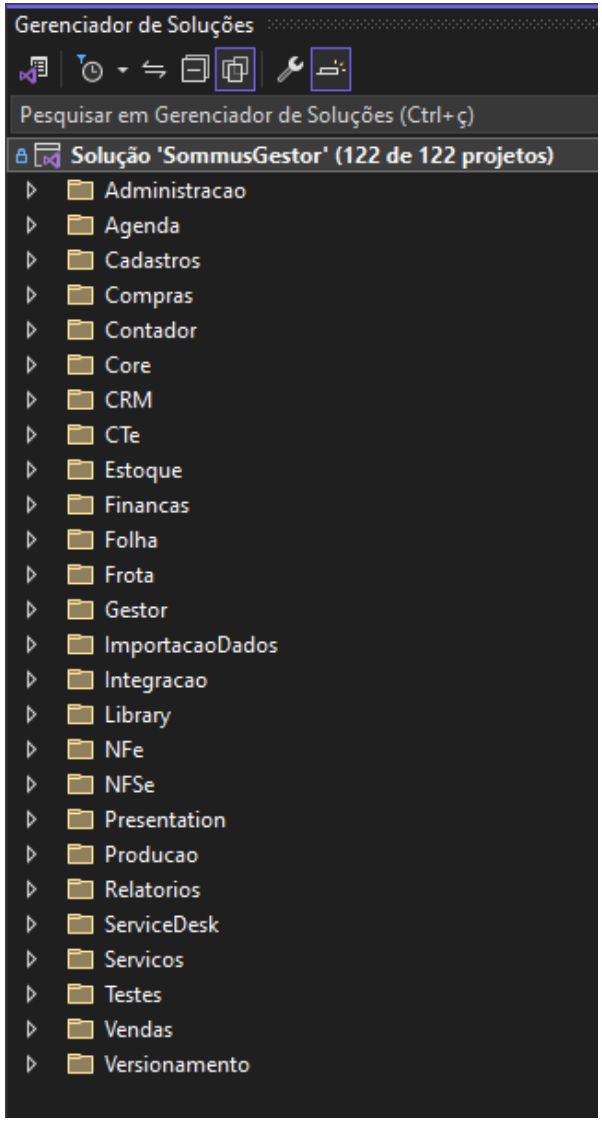
Contém os projetos Web e WebService (API)

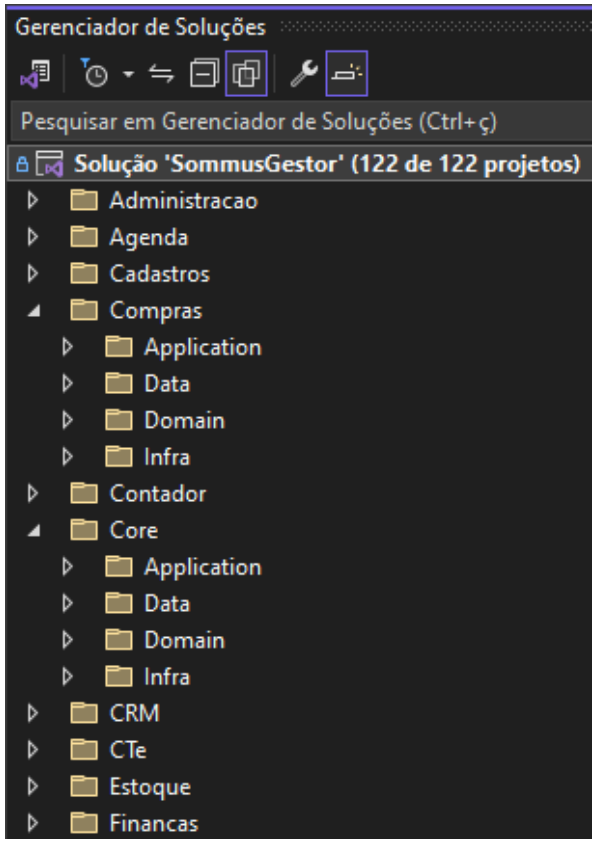
O contexto do Core centraliza implementações importantes para mais de um contexto.

Os projetos Library devem ser vistos como um código a parte. Imagine um pacote Nugget que é importado e utilizado. São independentes das regras de negócio e funcionam isoladamente.

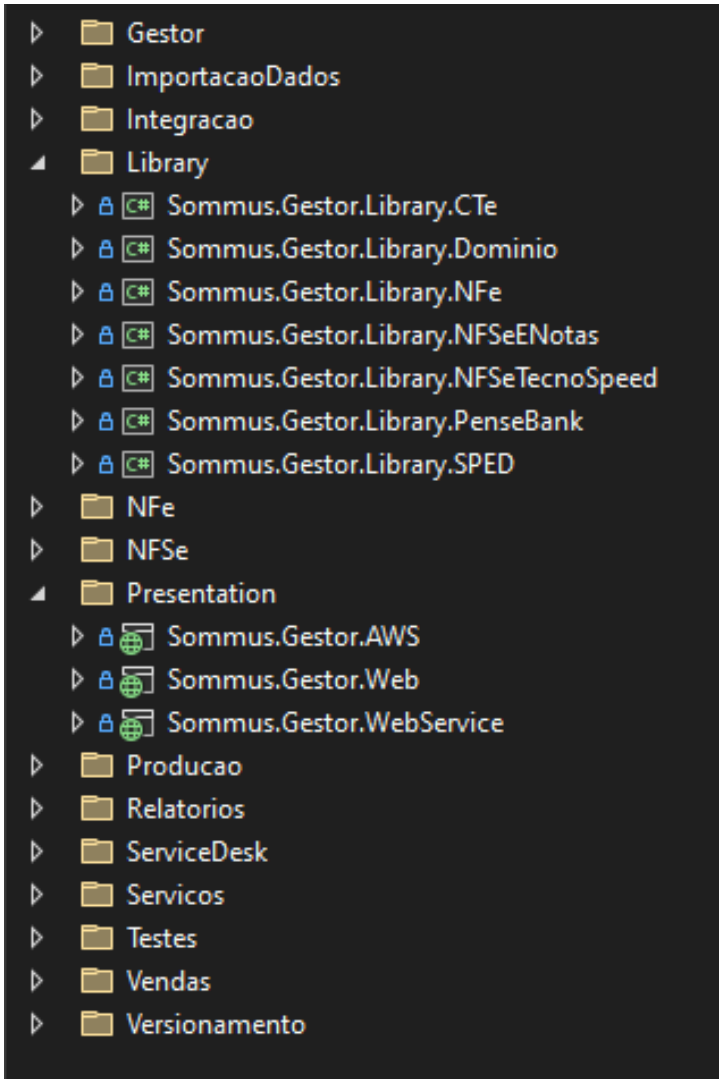
Contém os testes “E2E” que são os testes do Selenium e alguns projetos especiais de testes que cobrem áreas específicas.

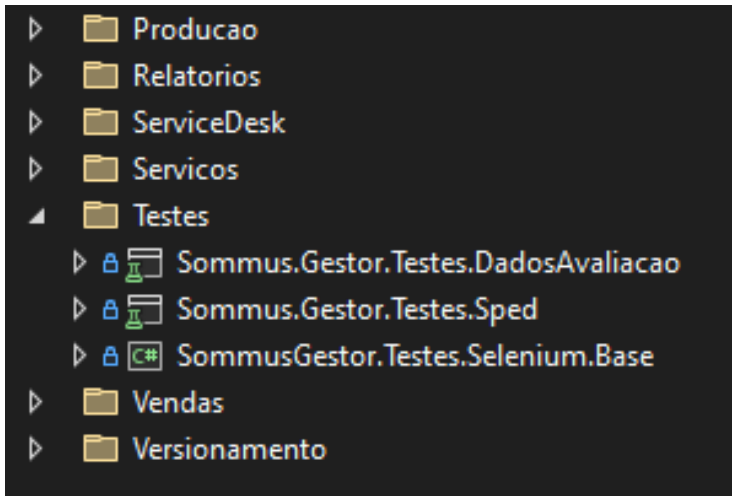
Estrutura de Pastas





Estrutura de Pastas





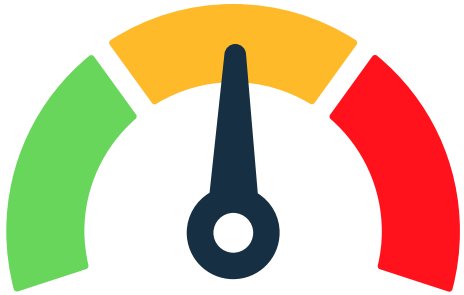
Fez sentido?

Dia 2 - Alinhando as expectativas

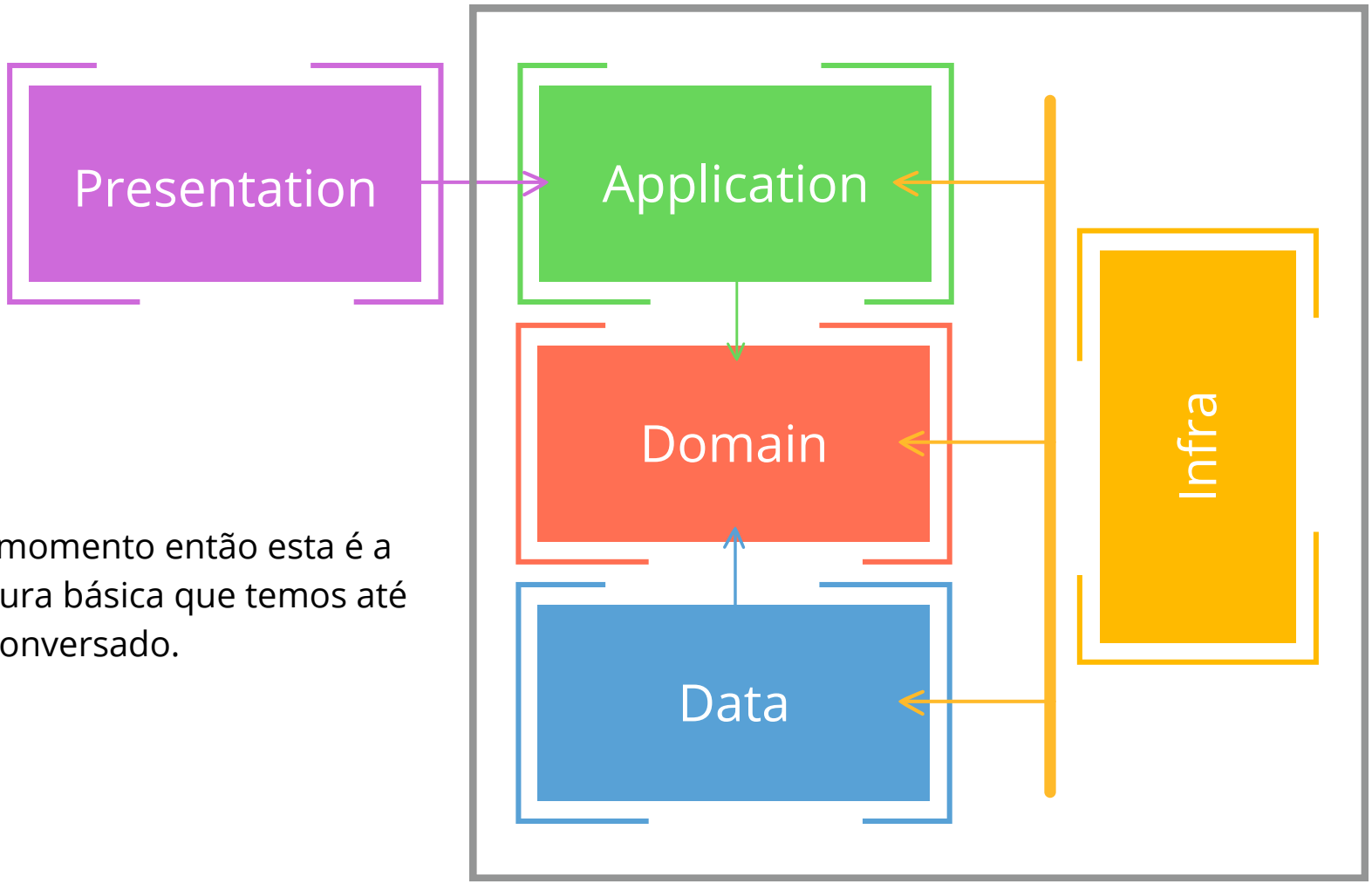
O que vamos conversar hoje?

Vamos descer um nível na visão das camadas.

Mostrar o SommusGestor percorrendo as camadas através de depuração de código.



Relembrando



Até o momento então esta é a estrutura básica que temos até aqui conversado.

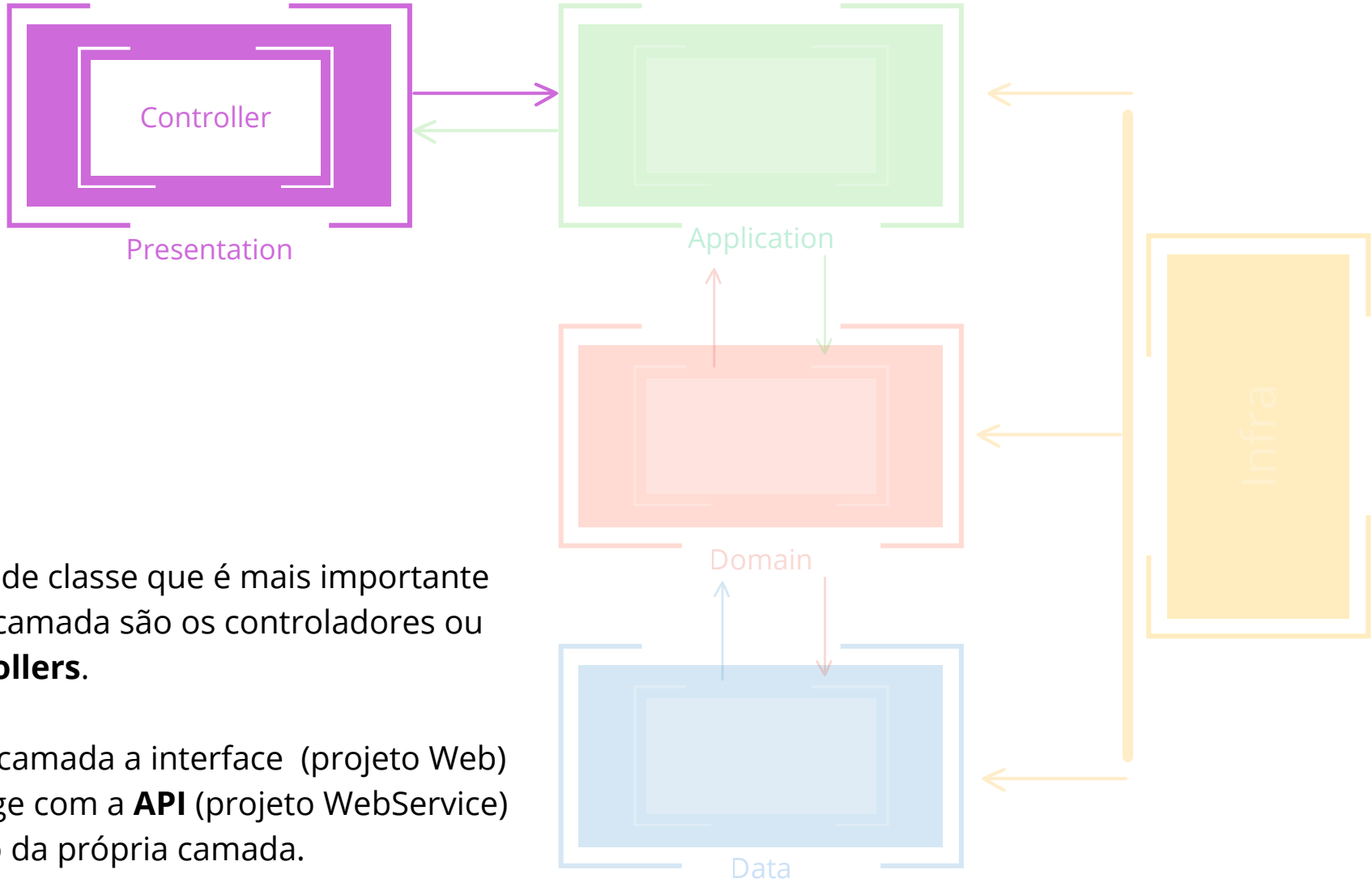
Camada Presentation

A camada **Presentation** (ou Apresentação) no contexto do DDD refere-se à parte da aplicação responsável pela interação com o usuário final, incluindo interfaces gráficas, APIs de serviço web, ou qualquer outro meio pelo qual os usuários interagem com o sistema.

No SommusGestor temos 3 projetos nesta camada:

- Projeto Web
- Projeto WebService
- Projeto AWS

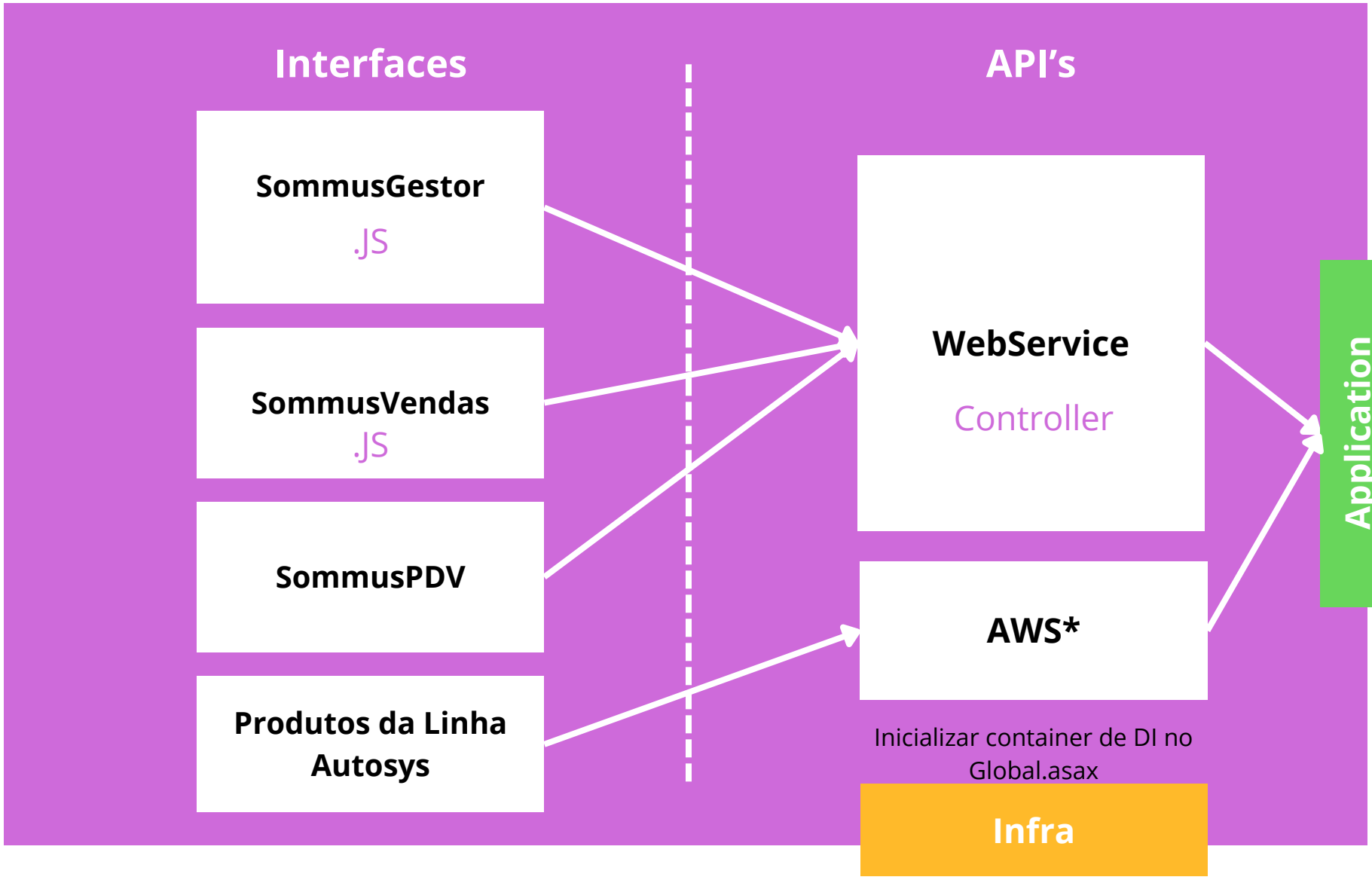
Classe principal da Presentation



O tipo de classe que é mais importante nesta camada são os controladores ou **Controllers**.

Nesta camada a interface (projeto Web) interage com a **API** (projeto WebService) dentro da própria camada.

Presentation



***Projeto Api AWS é o responsável principalmente pelo reset online dos produtos da linha Autosys.**

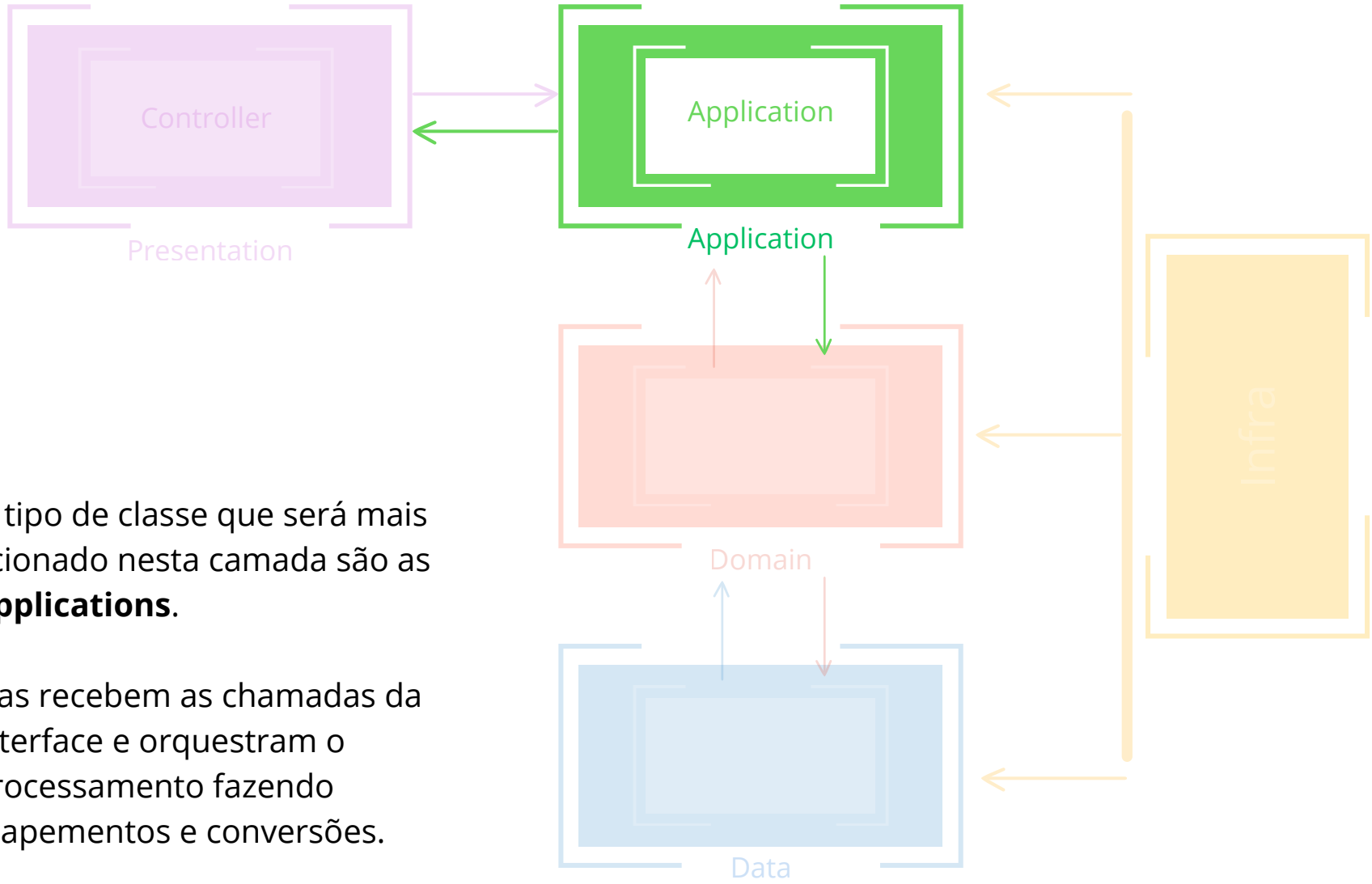
Camada Application

A camada de **Application** (ou Aplicação) no contexto do Domain-Driven Design (DDD) é responsável por coordenar as interações entre a camada de apresentação (Presentation) e a camada de domínio (Domain). Essa camada é onde residem as regras de aplicação e as lógicas de coordenação das operações do sistema.

No SommusGestor temos 2 tipos de projetos nesta camada:

- Projeto Application
- Projeto Application.Test

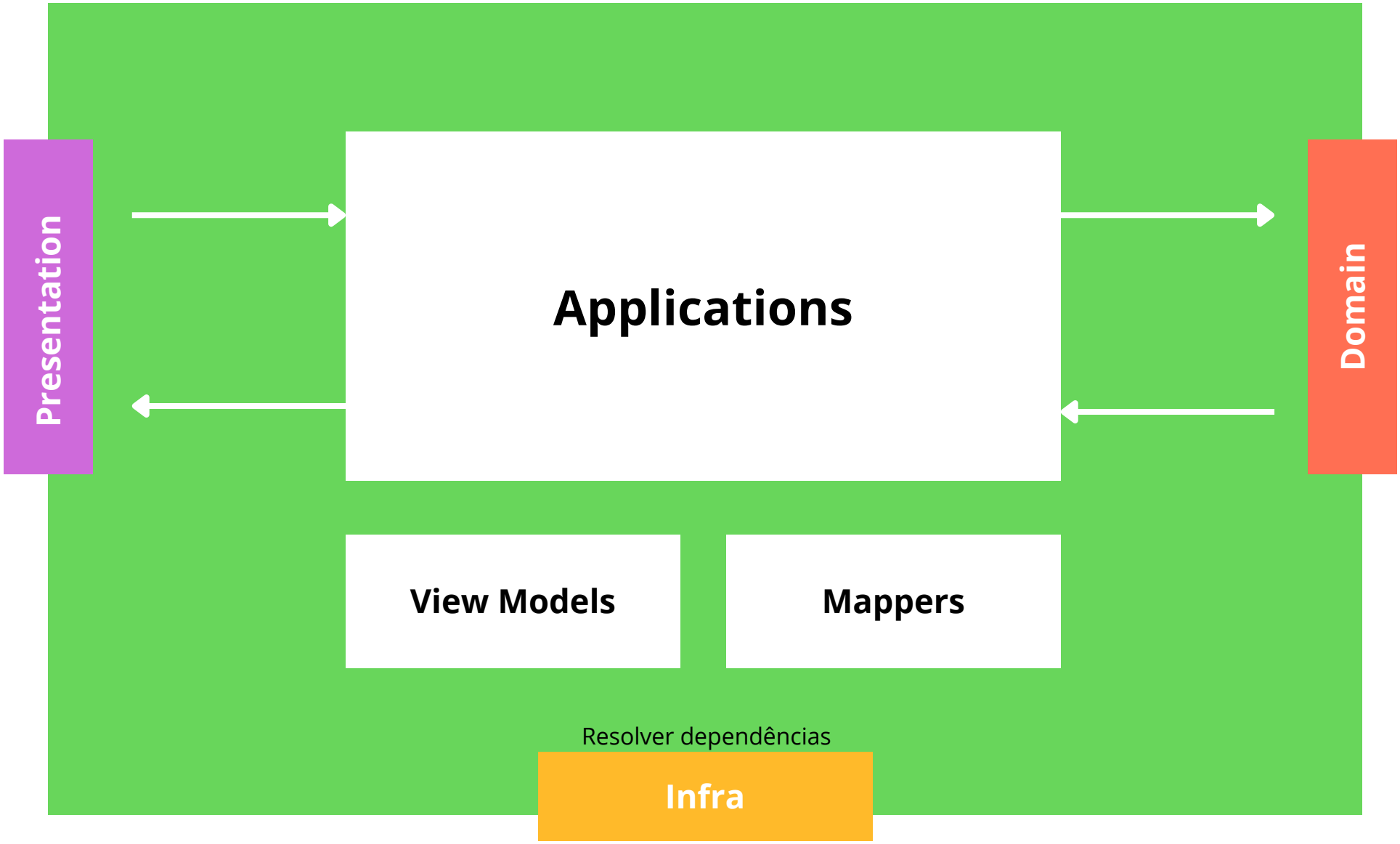
Classe principal da Application



O tipo de classe que será mais acionado nesta camada são as **Applications**.

Elas recebem as chamadas da interface e orquestram o processamento fazendo mapementos e conversões.

Application



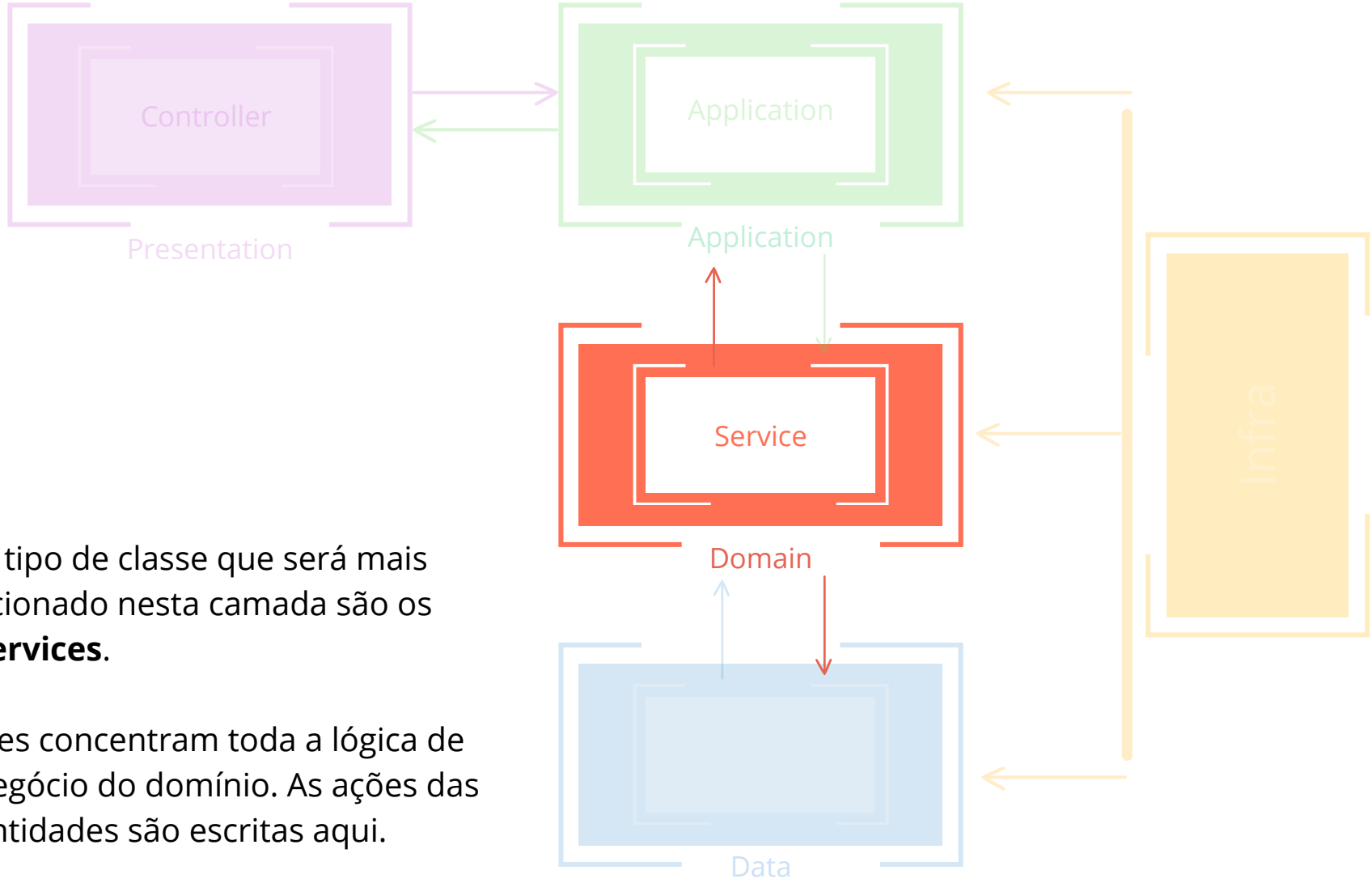
Camada Domain

A camada de **Domain** (ou Domínio) no Domain-Driven Design (DDD) é o núcleo da aplicação, onde reside a lógica de negócio e as entidades que representam os conceitos fundamentais do problema sendo resolvido. Essa camada é onde as regras de negócio são expressas e implementadas de forma a refletir com precisão o conhecimento do domínio em questão.

No SommusGestor temos 1 tipo de projeto nesta camada:

- Projeto Domain

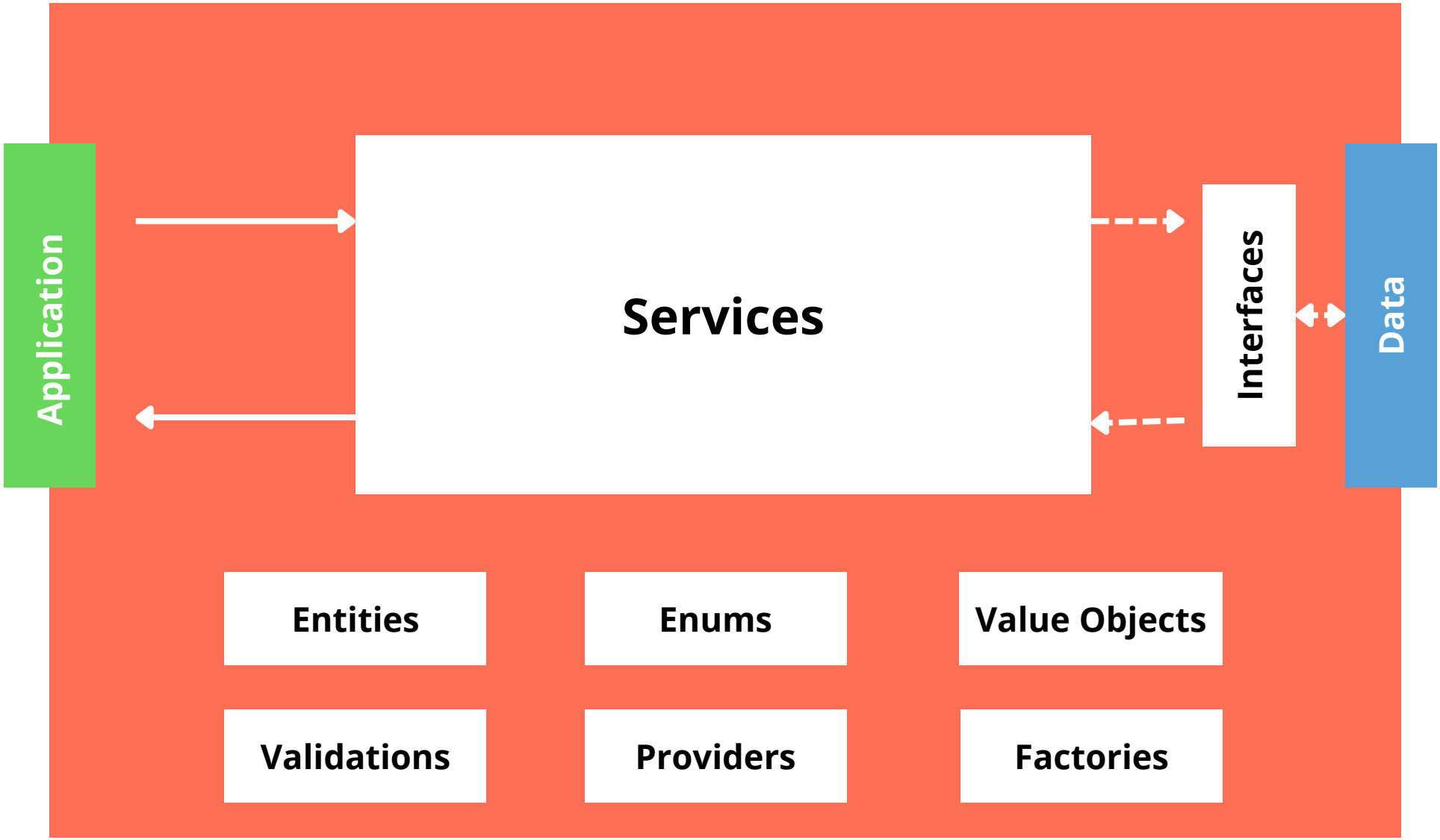
Classe principal do Domain



O tipo de classe que será mais acionado nesta camada são os **Services**.

Eles concentram toda a lógica de negócio do domínio. As ações das entidades são escritas aqui.

Domain


Application
Services
Interfaces
Data
Entities
Enums
Value Objects
Validations
Providers
Factories

Camada Data

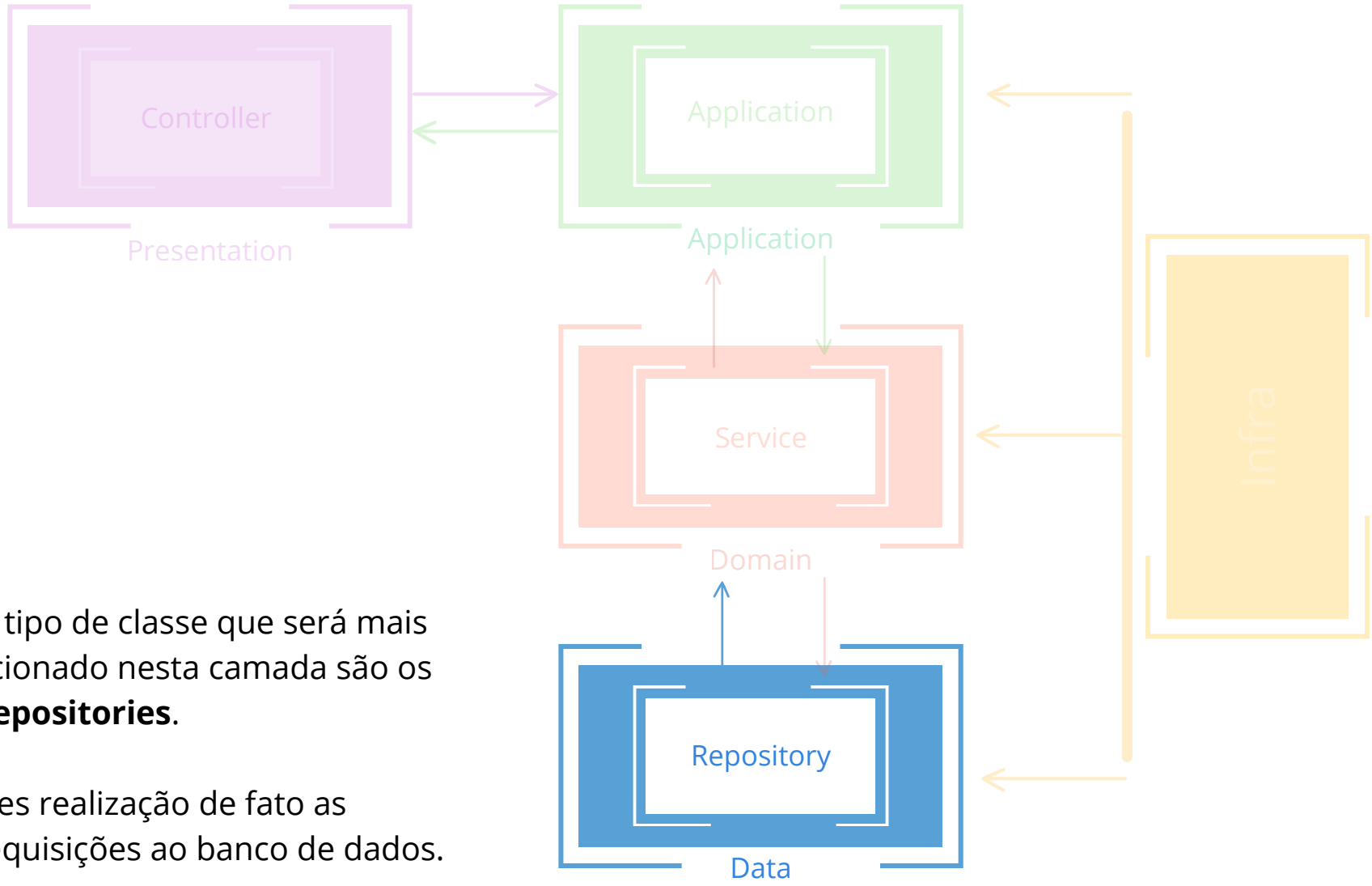
A camada de **Data** (ou Dados) no contexto do Domain-Driven Design (DDD) é responsável por lidar com a persistência e o acesso aos dados da aplicação. Esta camada é onde as operações de leitura e gravação em bancos de dados ou outros sistemas de armazenamento de dados são realizadas.

No SommusGestor temos 1 tipo de projeto nesta camada:

- Projeto Data.ADO

Não utilizamos ORM's para manipular o banco. Todas as queryes são escritas por nós.

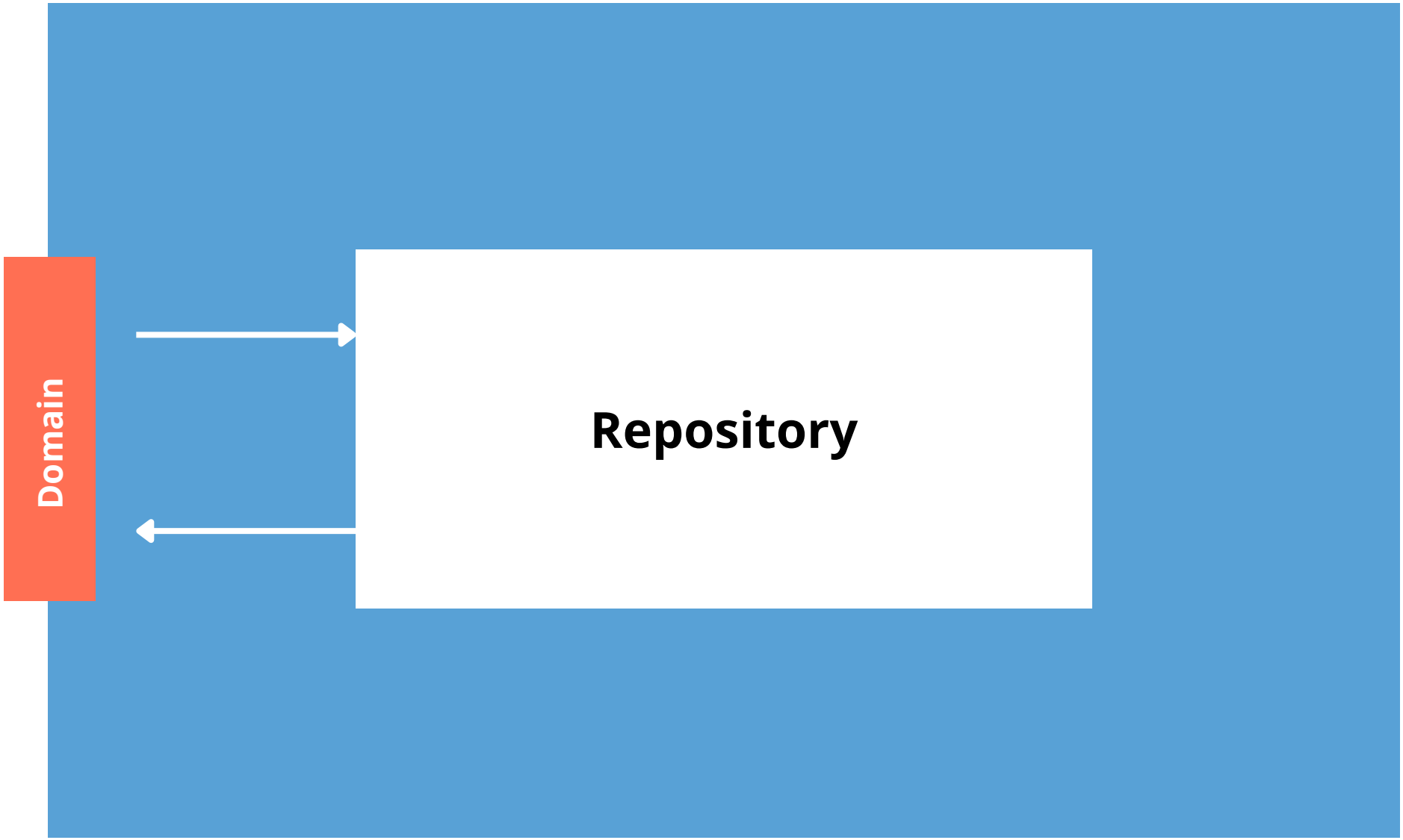
Classe principal do Data



O tipo de classe que será mais acionado nesta camada são os **Repositories**.

Eles realização de fato as requisições ao banco de dados.

Data



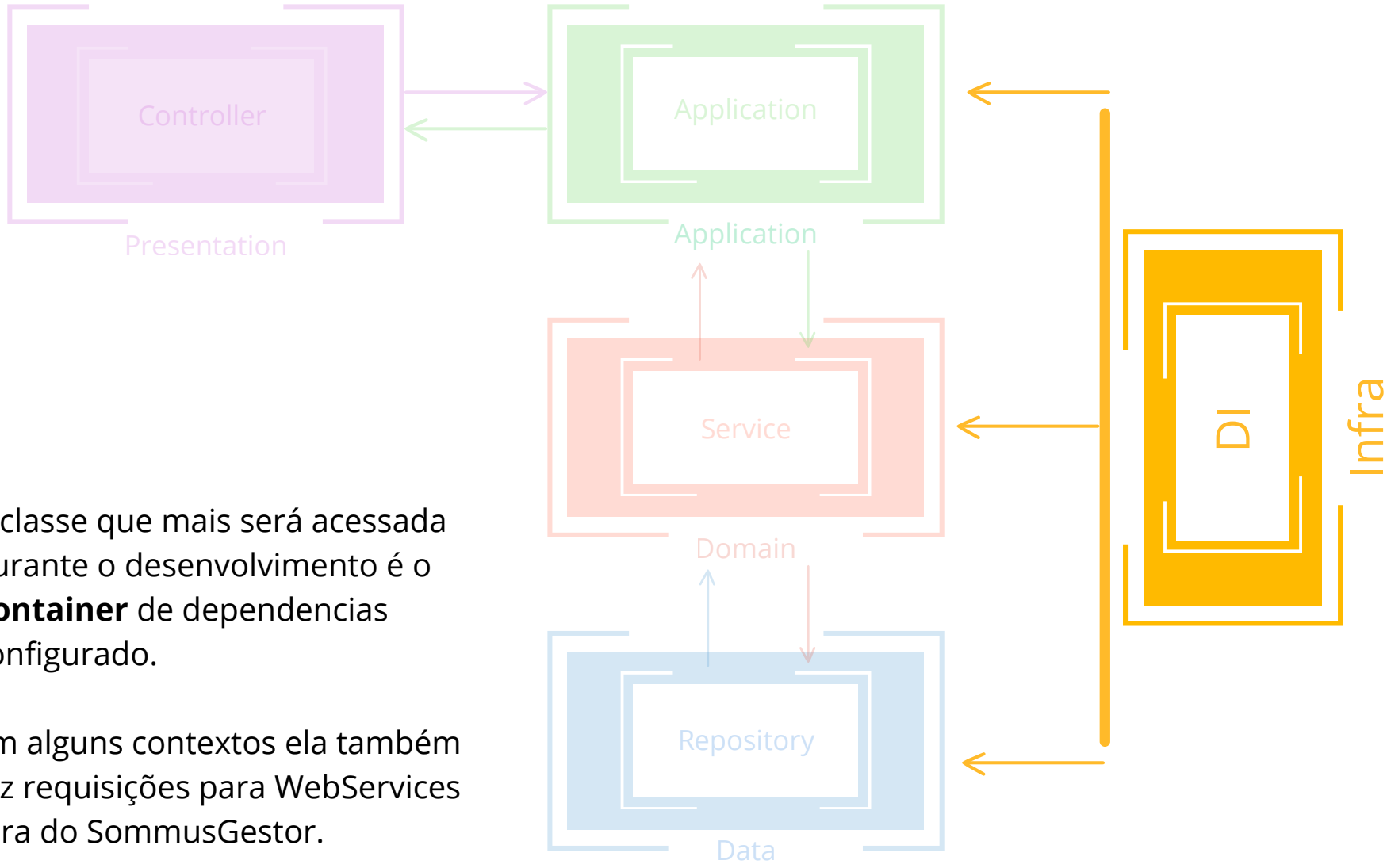
Camada Infra

A camada de **Infra** (ou Infraestrutura) no contexto do Domain-Driven Design (DDD) é uma camada adicional. Esta camada trata de aspectos técnicos que não estão diretamente relacionados à lógica de negócios, mas são necessários para o funcionamento da aplicação.

No SommusGestor temos 1 tipo de projeto nesta camada:

- Projeto Infra

Classe principal da Infra



A classe que mais será acessada durante o desenvolvimento é o **Container** de dependencias configurado.

Em alguns contextos ela também faz requisições para WebServices fora do SommusGestor.

Infra

Dependency Injection

Comunicação externa

Jobs do Hangfire

Compressão de arquivo

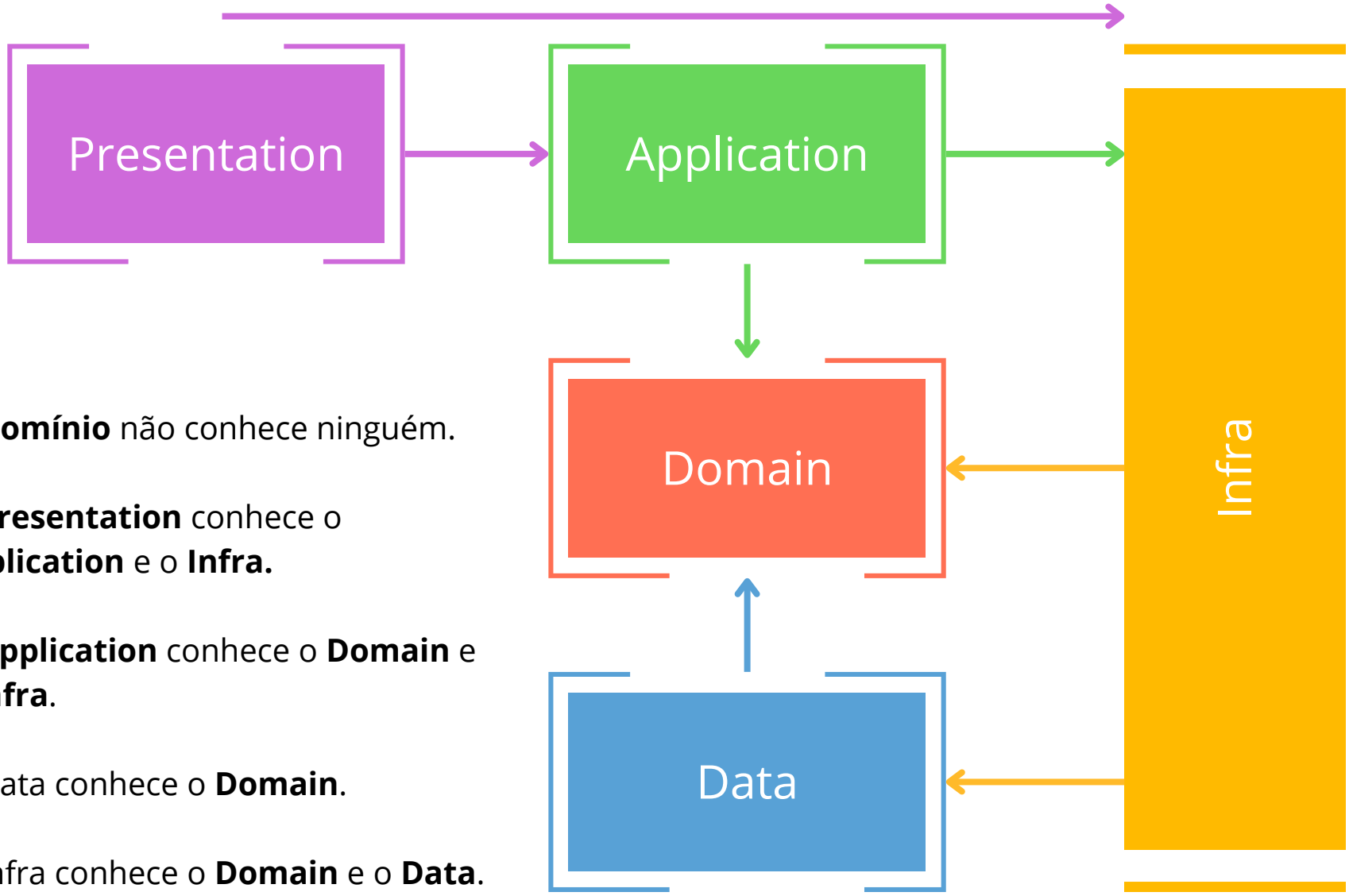
Conhece o Domain para Resolver as dependências da comunicação externa e implementar classes concretas dela.

Domain

Conhece o Data para Resolver as dependências dos Repositórios.

Data

Quem conhece quem?



O **Domínio** não conhece ninguém.

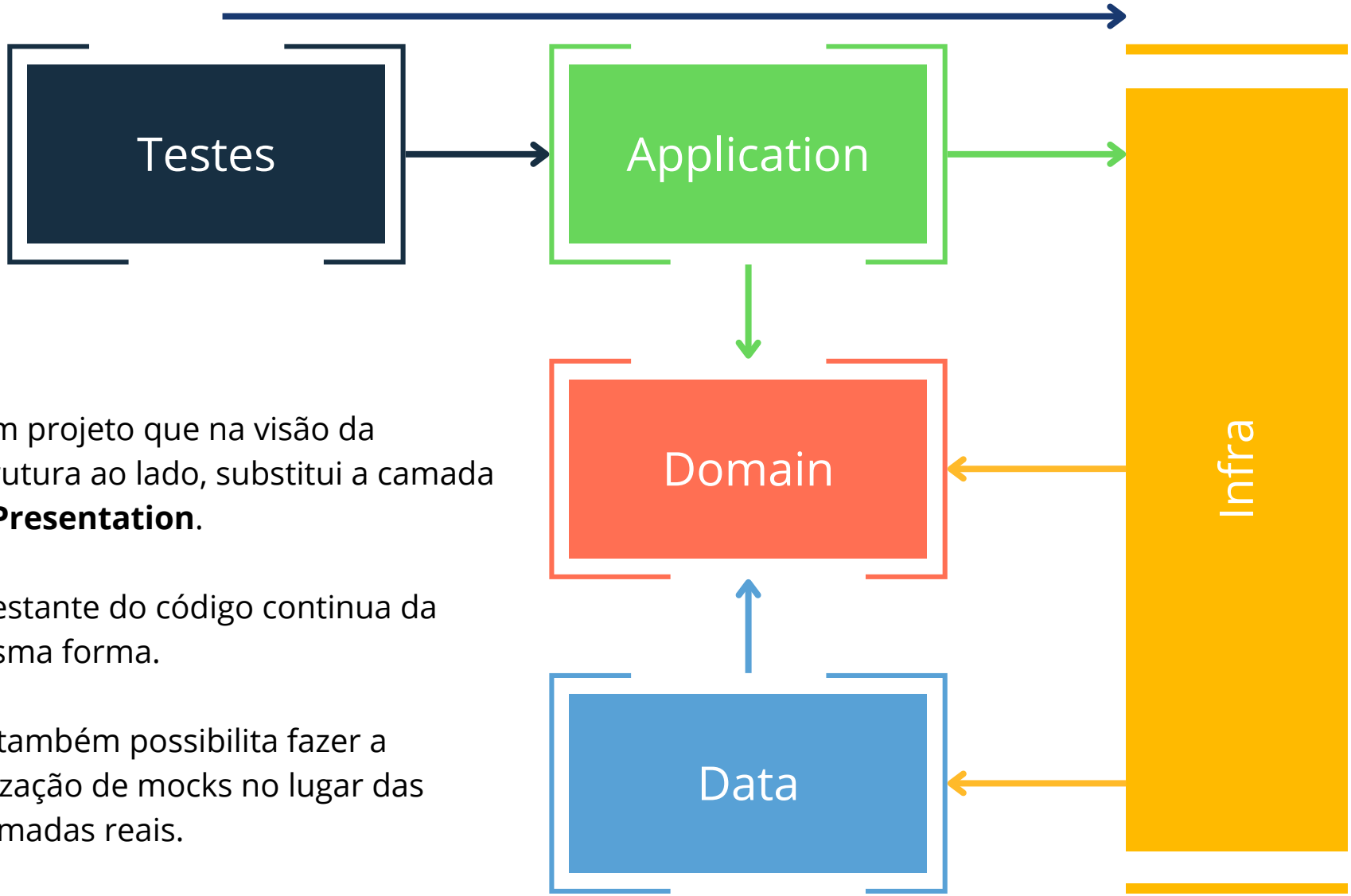
O **Presentation** conhece o **Application** e o **Infra.**

O **Application** conhece o **Domain** e o **Infra**.

O Data conhece o **Domain**.

O Infra conhece o **Domain** e o **Data**.

E os Testes?

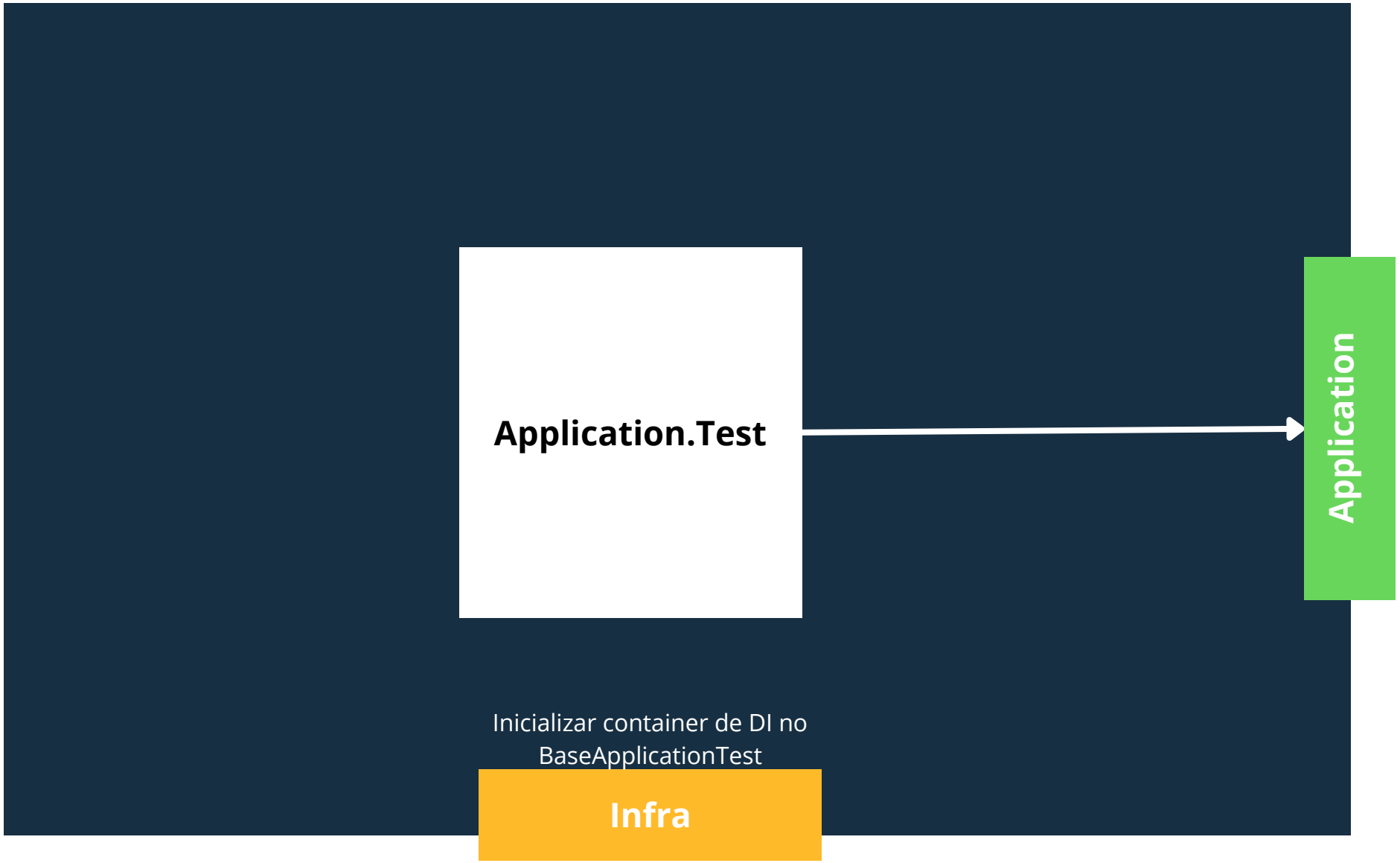


É um projeto que na visão da estrutura ao lado, substitui a camada da **Presentation**.

O restante do código continua da mesma forma.

Ele também possibilita fazer a utilização de mocks no lugar das chamadas reais.

Application.Test



Vamos ver as camadas em funcionamento?