PUC Minas
Virtual

Estilo em Camadas

O que iremos aprender nessa unidade?

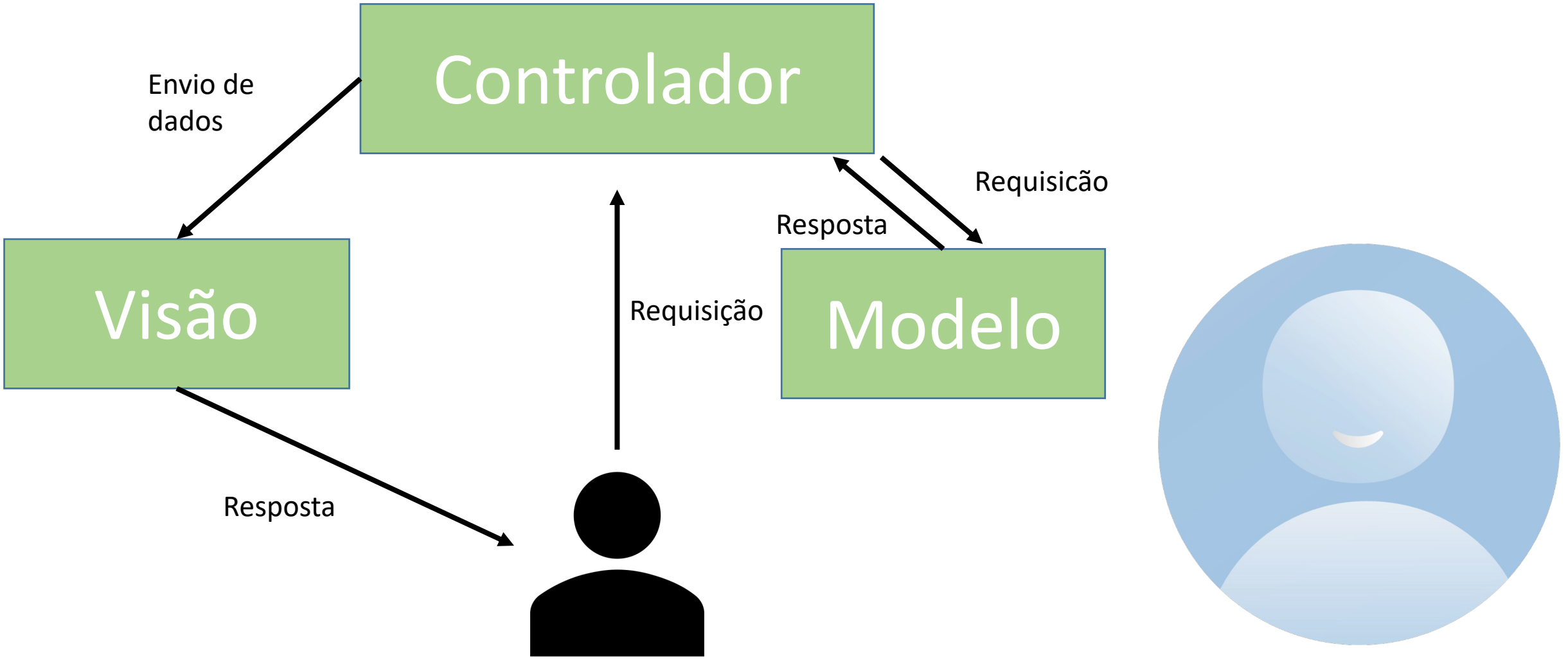
Marco Mendes

PUC Minas
Virtual

Padrão Arquitetural MVC

Marco Mendes

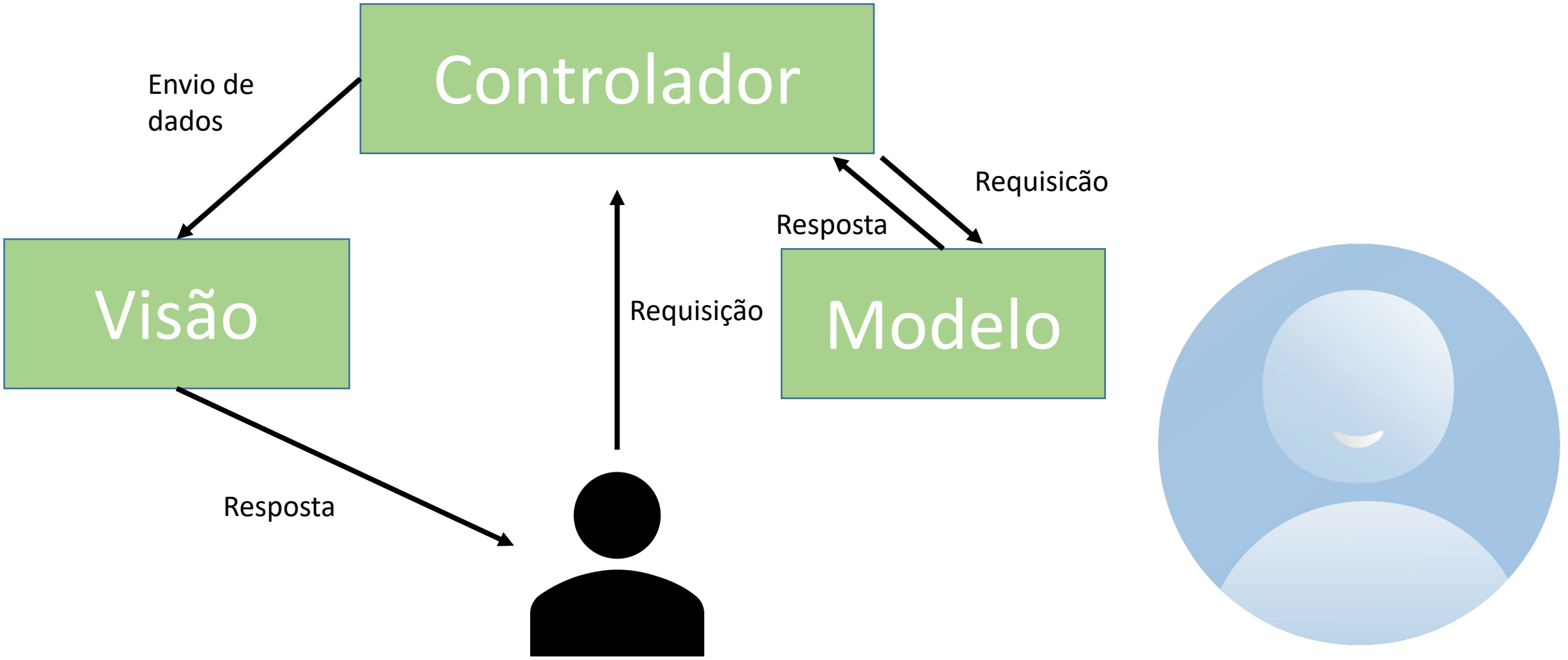
MVC – Model View Controller



Ordem de chamadas no MVC

1.
   - Usuário invoca algum controlador
   - Controlador valida requisição e seleciona o modelo a ser chamada
2.
   - Modelo executa regras de negócio, prepara e retorna dados para controlador
3.
   - Controlar entrega dados para Visão através de uma projeção do modelo
   - Visão desenha os dados para os clientes da aplicação

MVC – Considerações



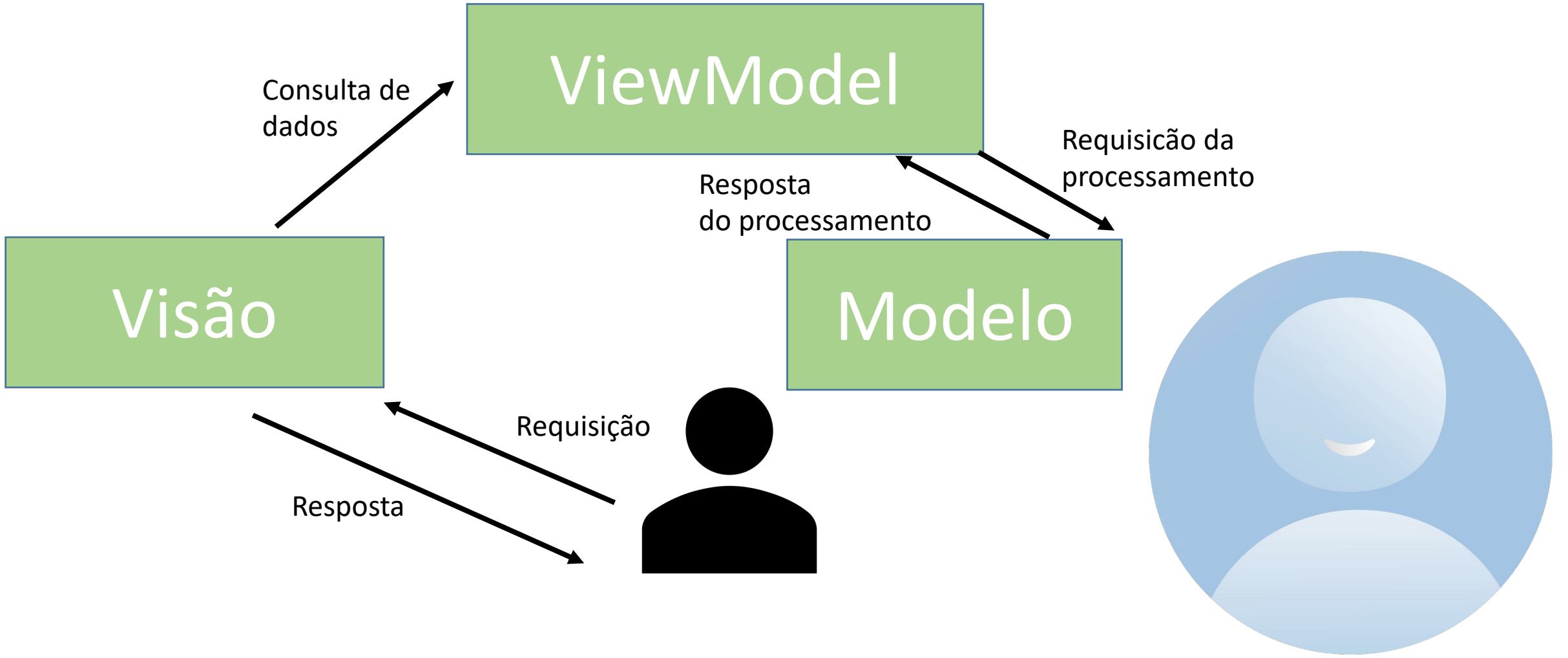
Atividade Prática

PUC Minas
Virtual

Padrão Arquitetural MVVM

Marco Mendes

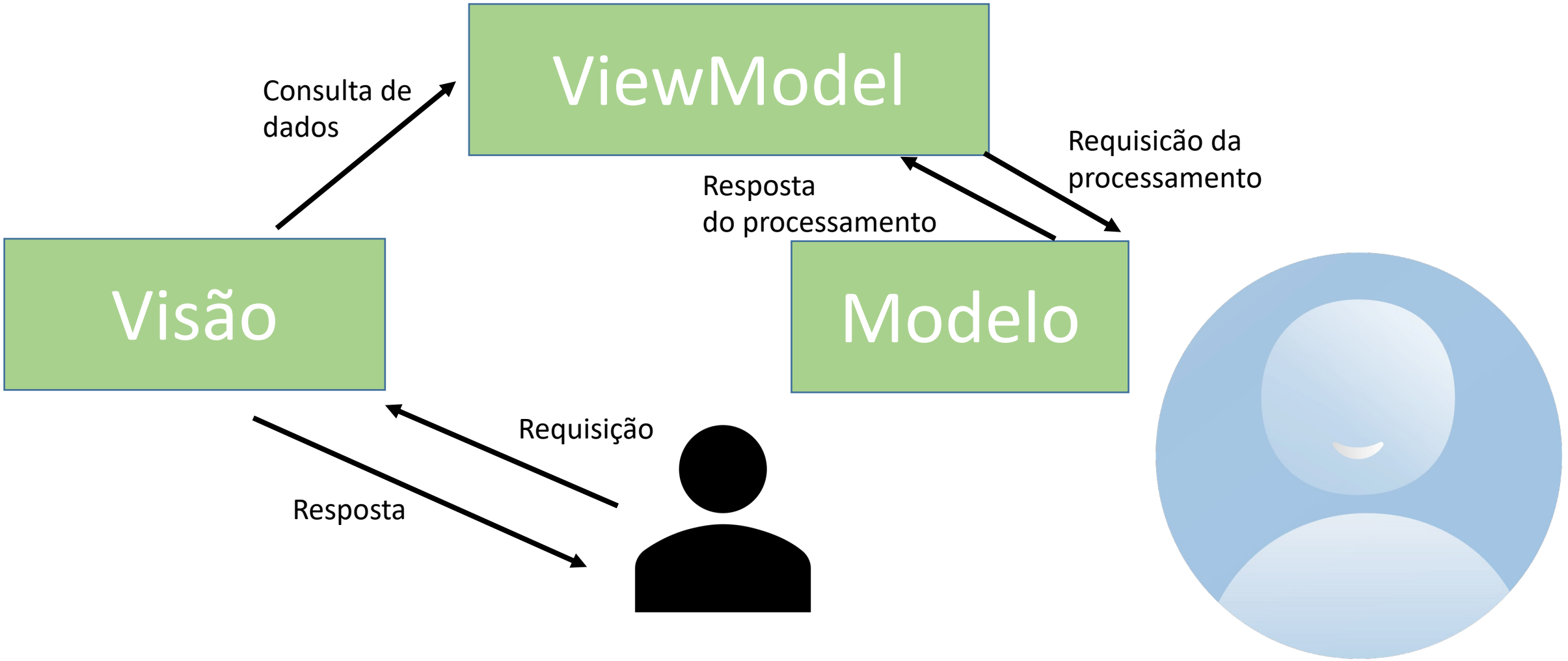
MVVM – Model View View Model



Ordem de chamadas no MVC

1.
   - Usuário invoca algum controlador
   - Controlador valida requisição e seleciona o modelo a ser chamada
2.
   - Modelo executa regras de negócio, prepara e retorna dados para controlador
3.
   - Controlar entrega dados para Visão através de uma projeção do modelo
   - Visão desenha os dados para os clientes da aplicação

MVVM – Considerações



Atividade Prática

PUC Minas
Virtual

Padrão Arquitetural DDD

Marco Mendes

DDD – Desenho Dirigido por Domínios

- Abordagem para o desenvolvimento de software que enfatiza a importância do **domínio central** e da **lógica do domínio**.

DDD

- Envolve a criação de abstrações de software chamadas modelos de domínio que encapsulam lógica de negócios complexa e vinculam as condições reais do aplicativo de um produto ao código.
- O DDD também defende a modelagem baseada na realidade dos negócios como relevante para os casos de uso e descreve áreas problemáticas independentes como contextos limitados, cada um dos quais se correlaciona com um microsserviço.

DDD – Domain Driven Design

O objetivo do DDD é criar sistemas ideais de objetos por meio de uma colaboração criativa entre especialistas técnicos e de domínio para refinar iterativamente um modelo conceitual

A linguagem ubíqua no DDD

A linguagem ubíqua no DDD (Domain Driven Design) é uma linguagem comum e compartilhada entre todas as pessoas envolvidas no desenvolvimento do software, incluindo desenvolvedores, especialistas de negócio e outros stakeholders. Essa linguagem é composta de termos e conceitos específicos que representam os conceitos do domínio da aplicação.

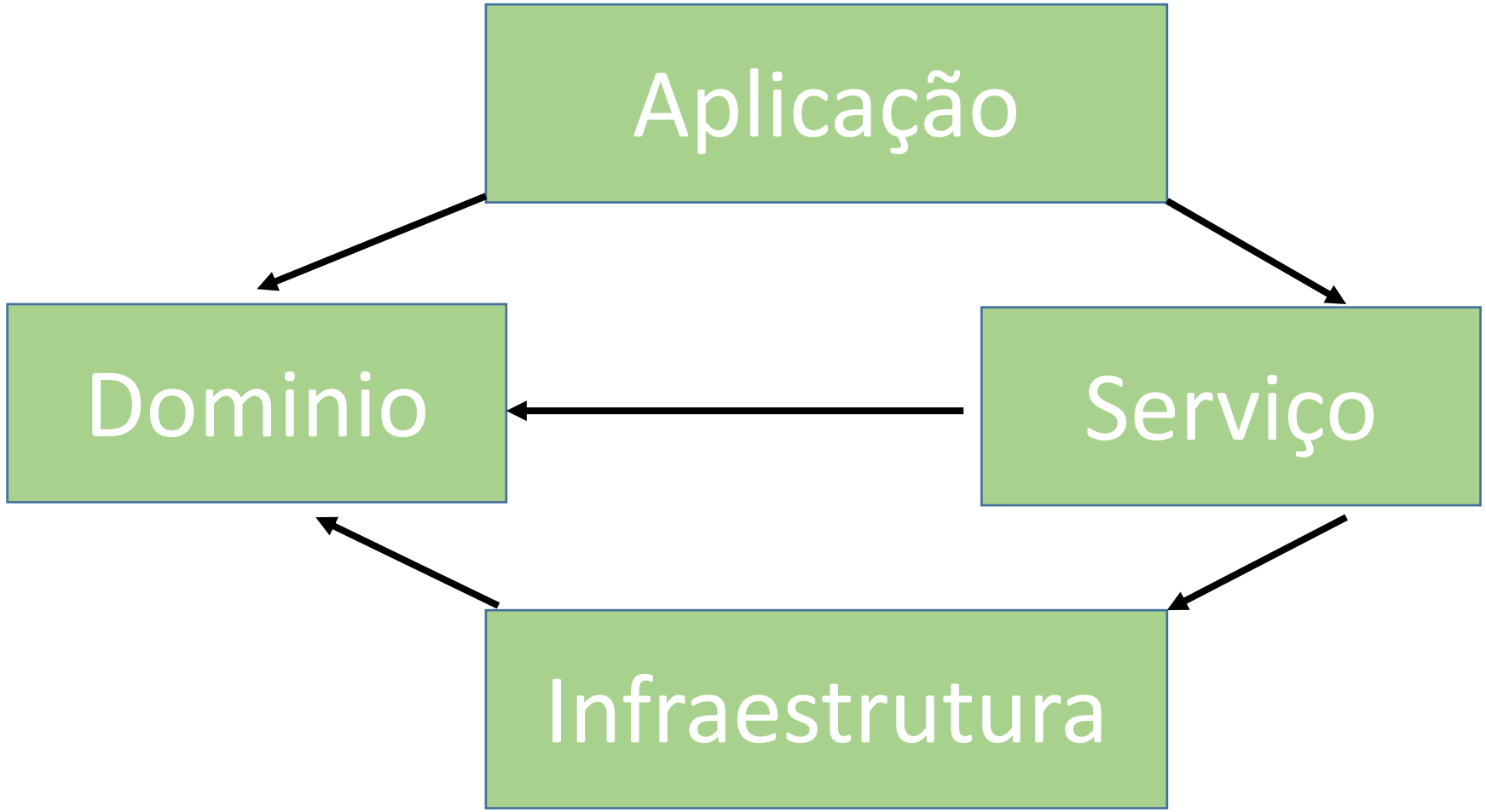
A linguagem ubíqua no DDD

A ideia por trás da linguagem ubíqua é que, ao utilizar uma linguagem comum, todos os envolvidos no desenvolvimento do software possam se comunicar de forma mais clara e precisa. Isso ajuda a evitar mal-entendidos e permite que todos trabalhem em conjunto para criar uma aplicação que reflita de forma fiel as necessidades e requisitos do negócio.

Exemplo de linguagem ubíqua

- **Correntista**: pessoa física ou jurídica que possui uma conta corrente no banco.
- **Conta corrente**: conta bancária utilizada pelo correntista para realizar transações financeiras, como depósitos, saques, transferências, entre outros.
- **Saldo**: valor atual disponível na conta corrente, considerando todas as transações já realizadas.
- **Extrato**: documento que apresenta todas as transações realizadas em uma conta corrente em um determinado período.
- **Limite de crédito**: valor máximo que o correntista pode utilizar de crédito em sua conta corrente.
- **Débito**: operação que retira um valor da conta corrente do correntista, como uma compra com cartão de crédito.
- **Crédito**: operação que adiciona um valor à conta corrente do correntista, como um depósito ou transferência recebida.

DDD – Visão Geral



DDD – Camada de Domínio

- Ela é responsável por representar conceitos, informações e situações referentes aos negócios.
- O estado que reflete a situação de negócios é controlado e usado aqui, embora os detalhes técnicos sobre como armazená-lo sejam delegados à infraestrutura.
- Essa camada é a essência de negócio do software sendo construído.

DDD – Camada de Infraestrutura

- Mantém a lógica técnica do produto, tais como persistência, segurança da informação, interoperabilidade e outros temas técnicos.
- Organiza os repositórios com a lógica específica de persistência de dados.

DDD – Camada de Serviço

- Fornece uma visão dos serviços da aplicação.
- Aqui são implementadas fluxos de trabalho e regras globais ao sistema.
- Essa camada também usa o domínio para persistir informações contra os bancos de dados.

DDD – Camada de Aplicação

- Camada de aplicação: responsável pelo projeto principal. Aqui serão implementados os controladores e a exposição de APIs.
- Tem a função de receber todas as requisições e direcioná-las a algum Serviço para executar uma determinada ação.

DDD – Estrutura simplificada de pastas

```
.
├── application
│   └── services
├── domain
│   ├── entities
│   ├── repositories
│   └── value-objects
├── infrastructure
│   └── repositories
└── interfaces
    └── controllers
```

Atividade Prática

PUC Minas
Virtual

Padrão Arquitetural Arquitetura Limpa

Marco Mendes

Arquitetura Limpa

- Conjunto de conceitos e princípios que visam a construção de sistemas com alta coesão, baixo acoplamento e separação de responsabilidades

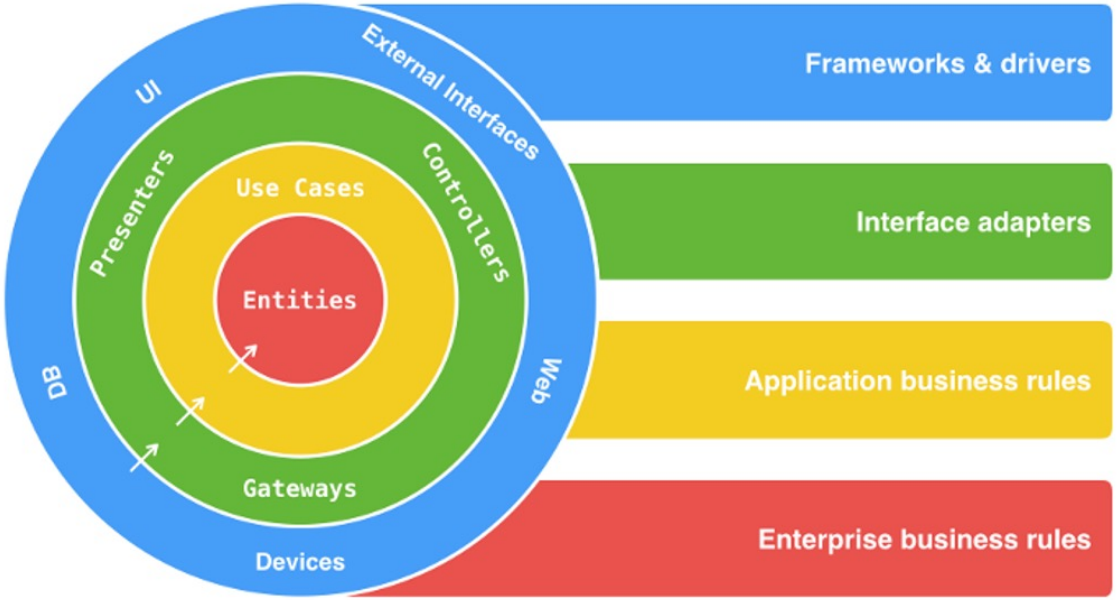
Arquitetura Limpa

- A regra principal da Arquitetura Limpa é a Regra da Dependência, que estabelece que as dependências de um código fonte devem sempre apontar para dentro, ou seja, as camadas mais internas não devem depender das camadas mais externas

Arquitetura Limpa

- Propõe a separação das preocupações, isolamento das regras de negócio, independência de frameworks, testabilidade, independência de UI, independência de banco de dados e independência de qualquer coisa externa.
- Outra regra importante é o Princípio do Reuso Comum, que diz que não devemos entregar para o usuário mais do que ele realmente precisa e devemos manter juntas somente as classes que o usuário irá utilizar.
- O papel do arquiteto é enxergar os pontos de mudança e criar limites/divisões para separar as partes do sistema de forma coesa.

Arquitetura Limpa



DDD – Estrutura simplificada de pastas

```
.
├── app
│   └── usecases
│       ├── consultar
│       ├── creditar
│       ├── debitar
│       └── definirLimiteCredito
├── domain
│   ├── entities
│   ├── repositories
│   └── value-objects
└── infrastructure
    ├── controllers
    ├── presenters
    ├── repositories
    └── server
```

Arquitetura Limpa versus DDD

- Quanto aos Objetivos:
  - *Arquitetura Limpa: Foca na organização do código para criar sistemas escaláveis, mantendo a separação de preocupações e facilitando a manutenção e evolução do software.*
  - *DDD: Foca na modelagem do domínio do problema, alinhando o design do software com o conhecimento do negócio e a linguagem utilizada pelos especialistas no domínio.*

Arquitetura Limpa versus DDD

- Quanto aos princípios:
  - *Arquitetura Limpa: Baseia-se em princípios de design de software, como SOLID e inversão de dependências. O código é dividido em camadas, sendo a camada mais interna responsável pela lógica de negócio e as camadas externas responsáveis pela comunicação com o mundo externo (UI, banco de dados, etc.).*
  - *DDD: Baseia-se em princípios específicos para a modelagem de domínios, como Entidades, Agregados, Repositórios, Serviços de Domínio e Objetos de Valor. O DDD também promove o uso de uma linguagem ubíqua, compartilhada por desenvolvedores e especialistas no domínio para garantir uma compreensão comum do problema*

Arquitetura Limpa versus DDD

- Quanto a abordagem:
  - Arquitetura Limpa: O foco é na organização e separação do código em diferentes camadas e componentes. A Arquitetura Limpa tem um padrão de camadas concêntricas: *Entidades (regras de negócio), Casos de Uso (interação com a lógica de negócio), Adaptadores (conversão de formatos) e Estruturas (integração com o mundo externo*).
  - DDD: O foco é na modelagem do domínio, criando um modelo rico e expressivo que reflita o conhecimento do negócio. O DDD também se preocupa com a divisão do sistema em contextos limitados, que são partes do domínio com limites bem definidos, facilitando o desenvolvimento e manutenção do sistema.