Arquitetura MVC

João Pedro Oliveira Batisteli

Introdução e Contextualização

- Até agora discutimos arquiteturas lógicas.

- Chamamos de arquitetura lógica à organização das classes de um sistema orientado a objetos (SSOO) em *subsistemas*.

- Já arquitetura física mostra coisas como o estilo arquitetónico, serviços, protocolos, bases de dados, interfaces de usuários, gateways de API, etc.

Introdução e Contextualização

- Anteriormente discutimos sobre arquitetura em camadas que são tipicamente encontradas em sistemas de informação.

- **A camada da apresentação (camada 1) não deve conter "inteligência", mas sim implementar apenas lógica da apresentação.**

- Outro padrão arquitetural normalmente utilizado para alcançar essa separação de responsabilidades entre a lógica da apresentação e a lógica da aplicação é o **Model-View-Controller** (MVC).

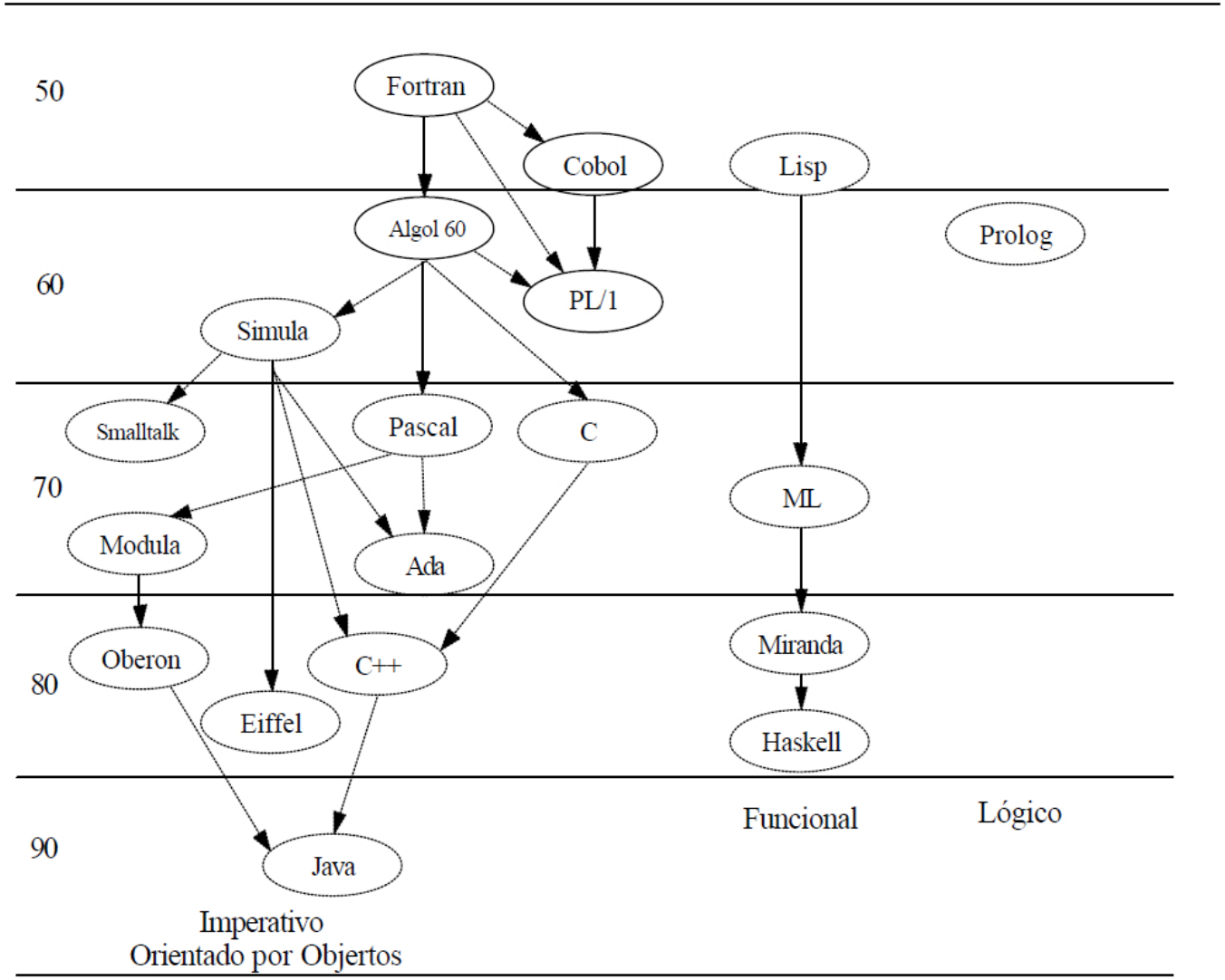
Introdução e Contextualização

- O padrão arquitetural MVC foi proposto no final da década de 70 e, em seguida, usado na implementação de Smalltalk-80.

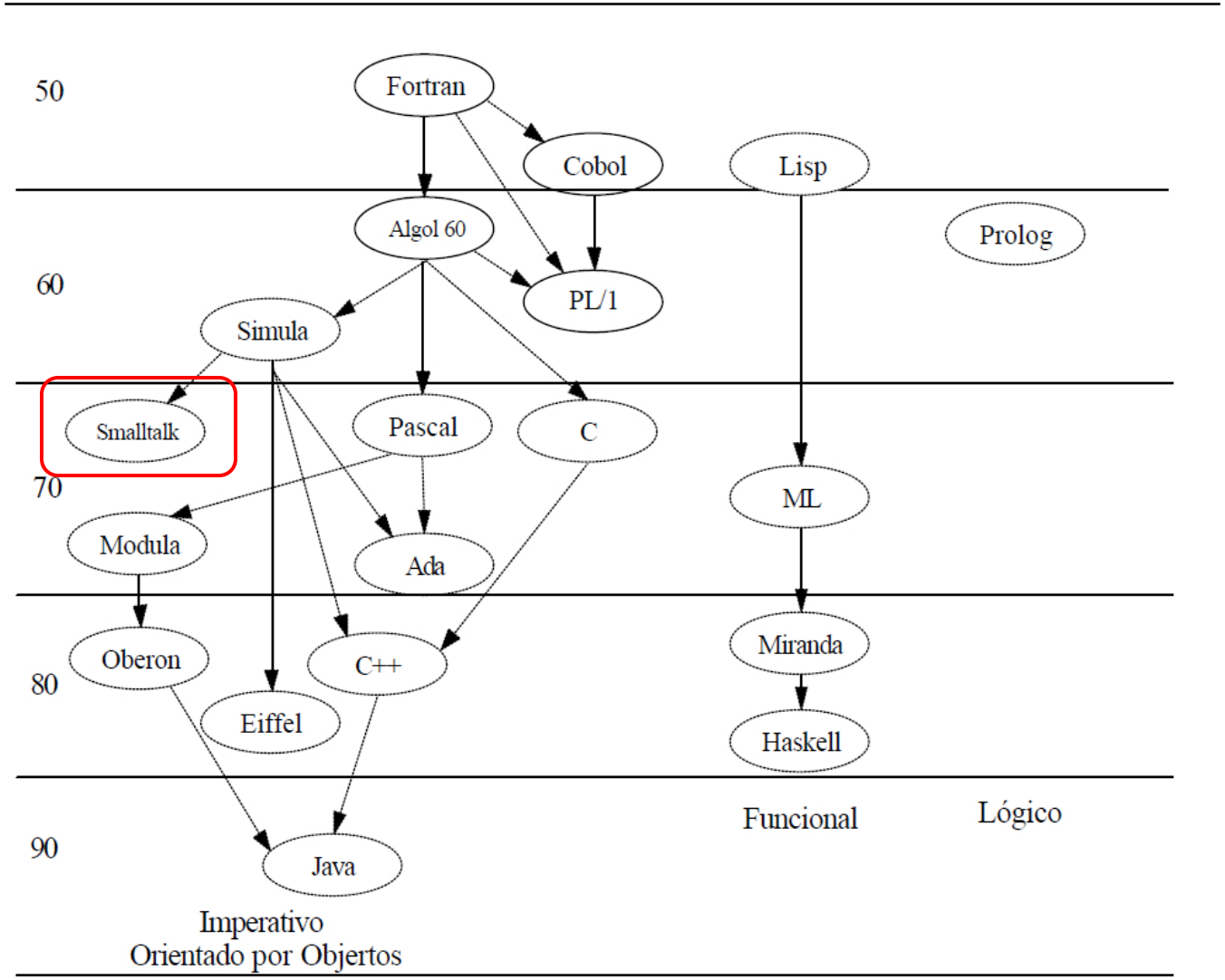
- Artigo "*Applications Programming in Smalltalk-80: How to use Model-View-Controller*"

- Além de utilizarem conceitos de orientação a objetos, programas em Smalltalk foram pioneiros no uso de interfaces gráficas, com janelas, botões, scroll bars, mouse, etc.

Introdução e Contextualização



Introdução e Contextualização



Introdução e Contextualização

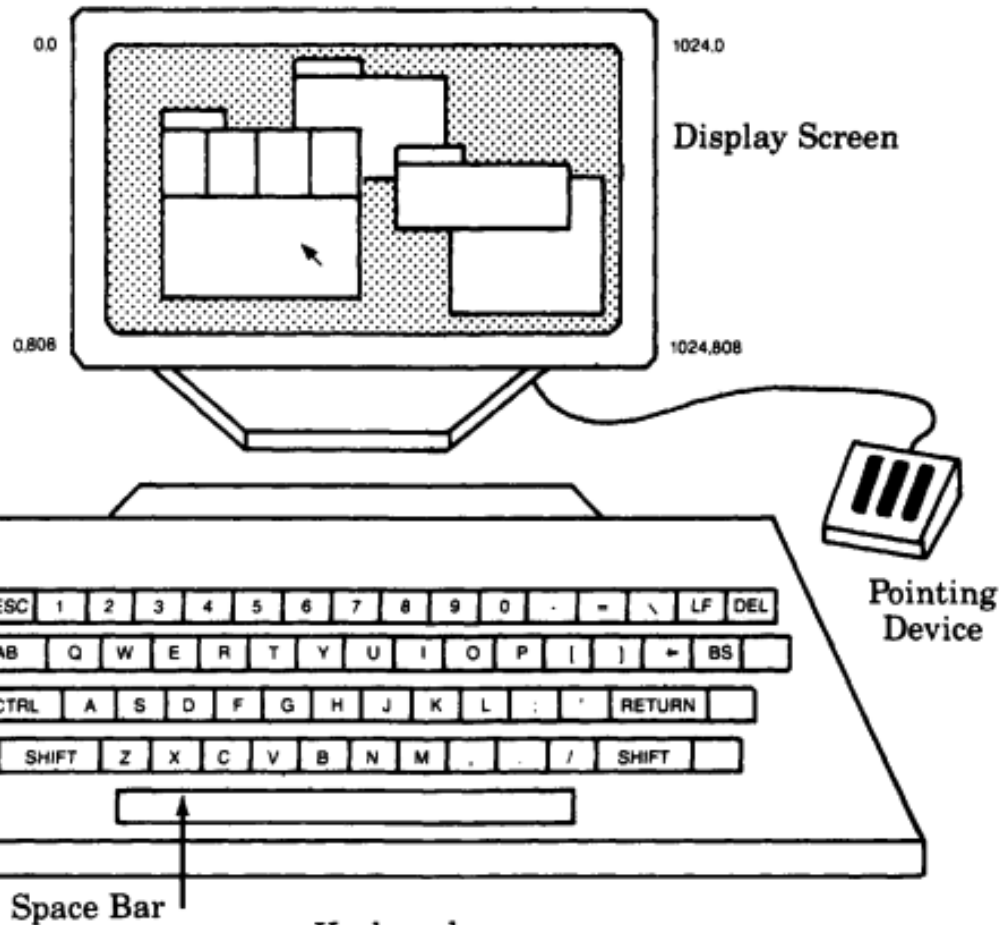
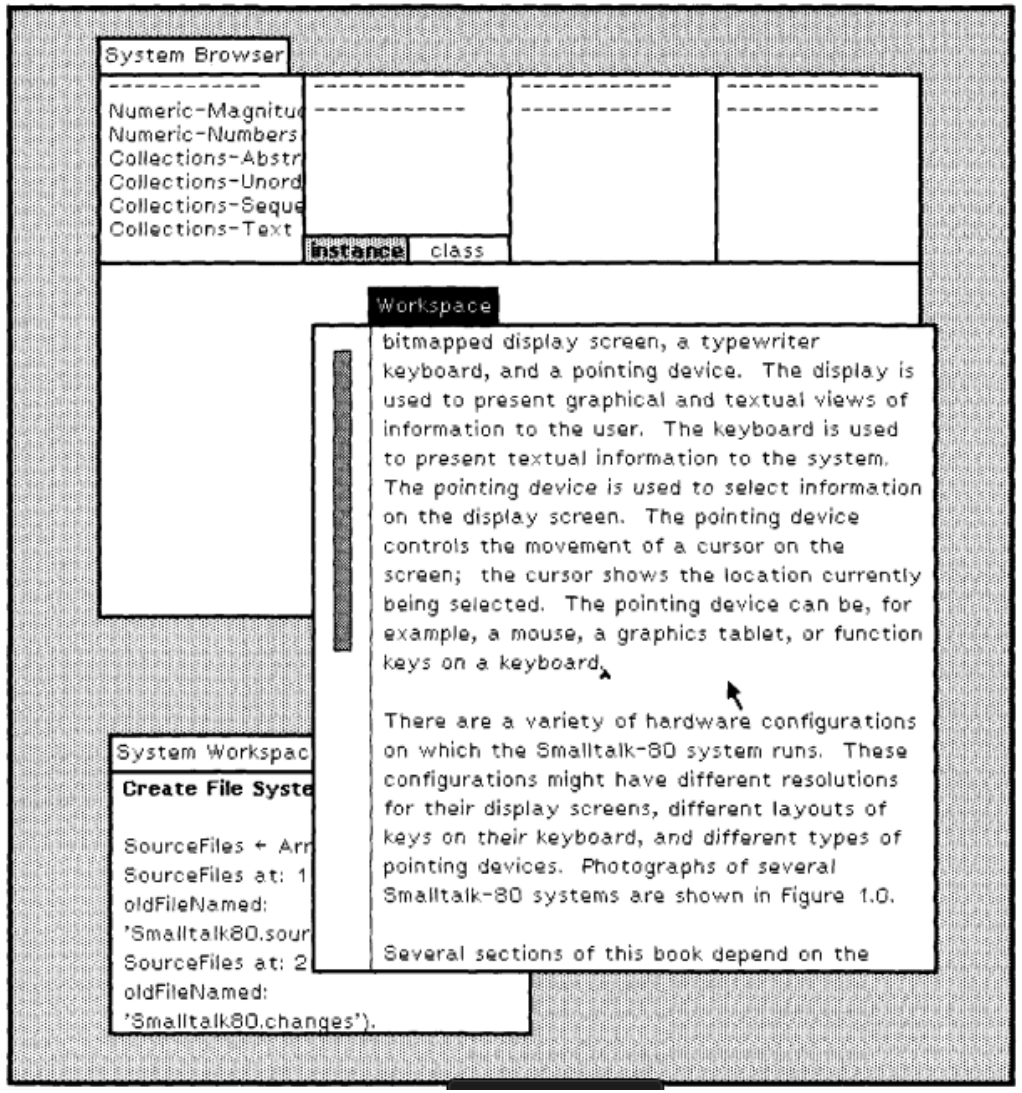
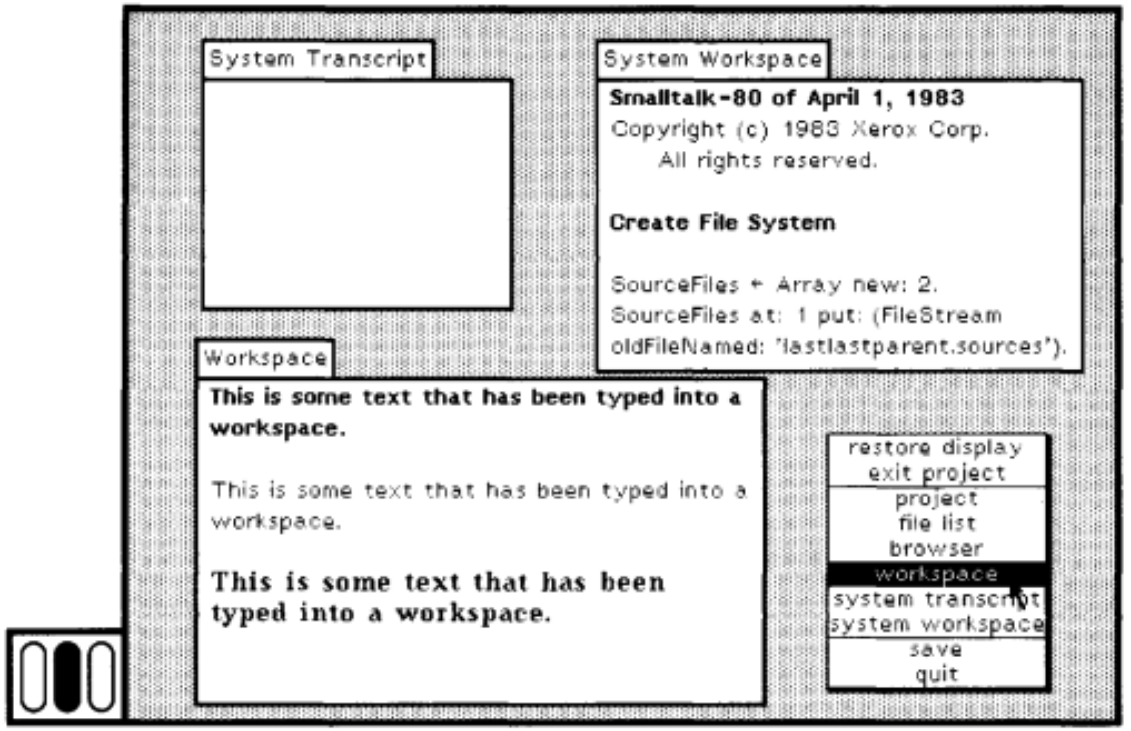


Figure 1.2

Introdução e Contextualização





Introdução e Contextualização

- Devido a possibilidade de construção de interfaces gráficas, o Samalltalk teve que adotar uma estratégia para controlar esse componente.

- MVC foi o padrão arquitetural escolhido pelos projetistas de Smalltalk para implementação de interfaces gráfica.

- Esse padrão pode ser usado em qualquer aplicação iterativa.

Por que o SmallTalk não adotou uma arquitetura em camadas (2 ou 3)?

Smalltalk x Arquitetura em Camadas

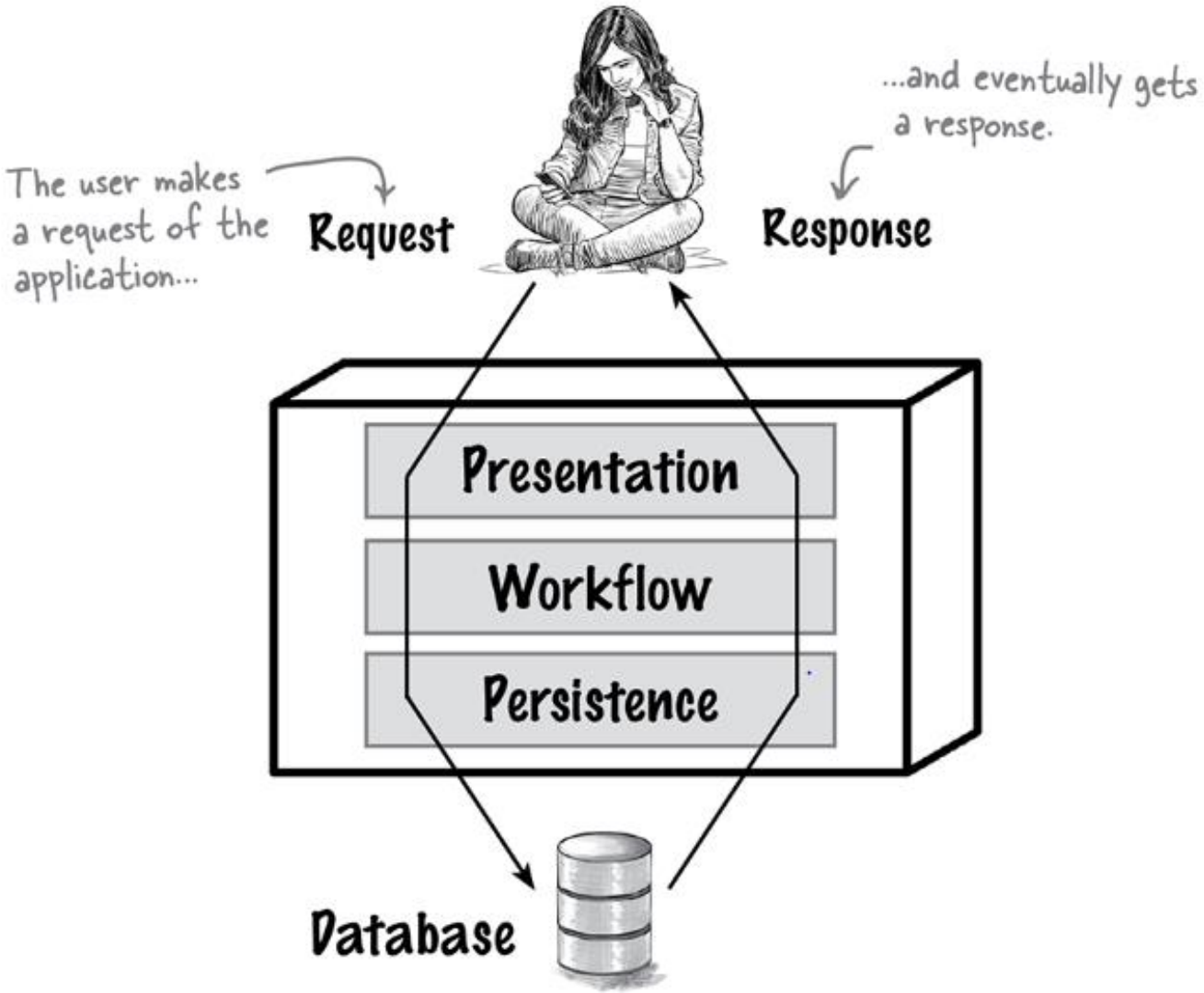
- Anos 70-80 os computadores pessoais estavam em ascensão.
- Limitação do modelo em camadas
  - Arquiteturas em 2 ou 3 camadas eram voltadas para **sistemas de informação tradicionais** (dados + lógica de negócio + apresentação).
  - As camadas tinham comunicação **sequencial e rígida** (pouco flexível).
- **Interfaces gráficas exigiam**:
  - **Interatividade em tempo real** (usuário ↔ sistema).
  - Ciclo de atualização dinâmico entre entrada do usuário, lógica e tela.
  - Maior acoplamento entre eventos e representação visual.

Smalltalk x Arquitetura em Camadas

- **Solução do Smalltalk**:
  - Criar um padrão que permitisse **sincronização contínua** entre dados, lógica e interface.
  - Daí surge o **MVC**, onde cada parte tem responsabilidades bem definidas, mas se comunicam por notificações e eventos, não por camadas fixas.

- O **modelo em camadas (da época de 70-80) era insuficiente para lidar com a interatividade gráfica**, e o **MVC foi proposto para responder às demandas de interfaces dinâmicas do Smalltalk-80**.

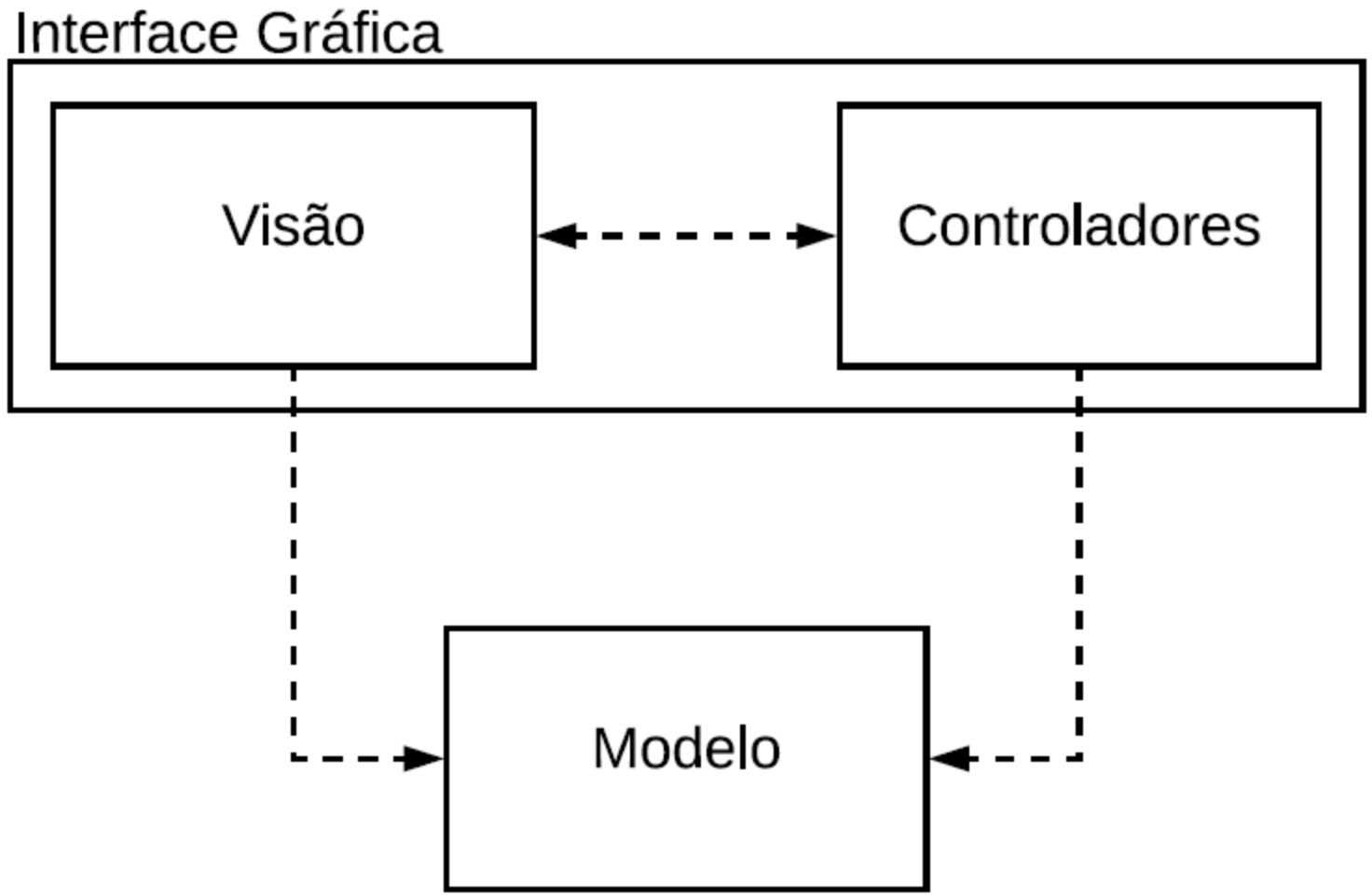
Smalltalk x Arquitetura em Camadas



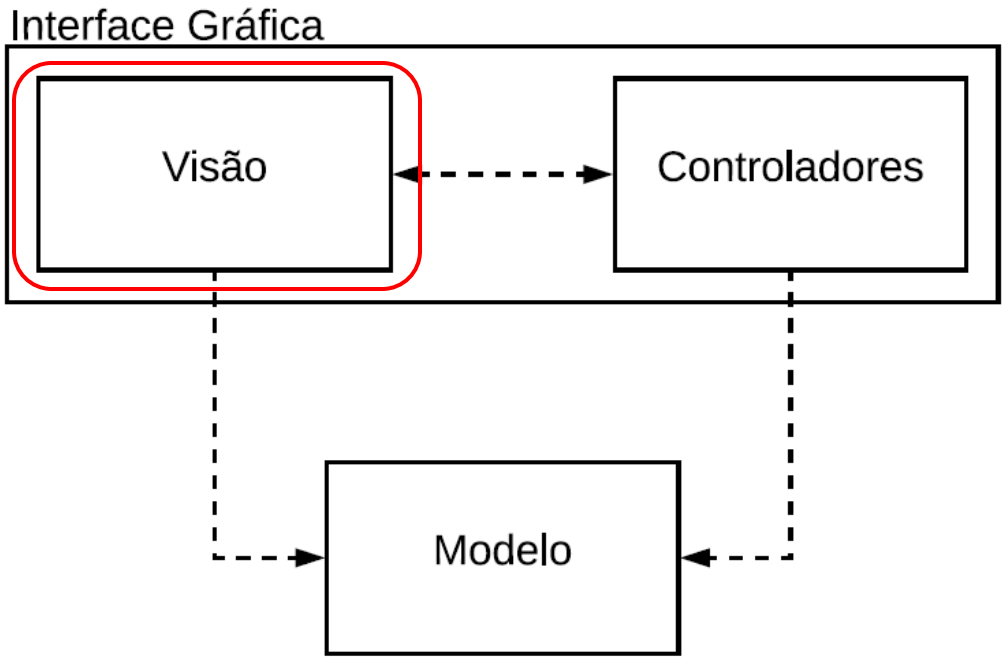
MVC

- MVC define que as classes de um sistema devem ser organizadas em três grupos:
  - Model (ou Modelo)
  - View (ou Visão)
  - Controller (ou Controlador(a))

MVC

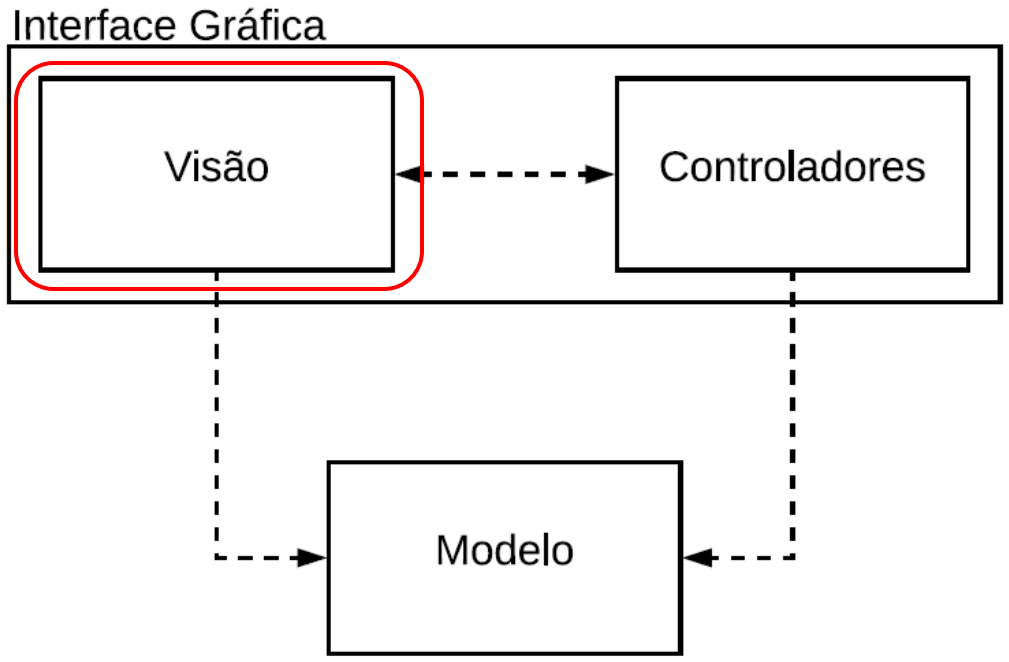


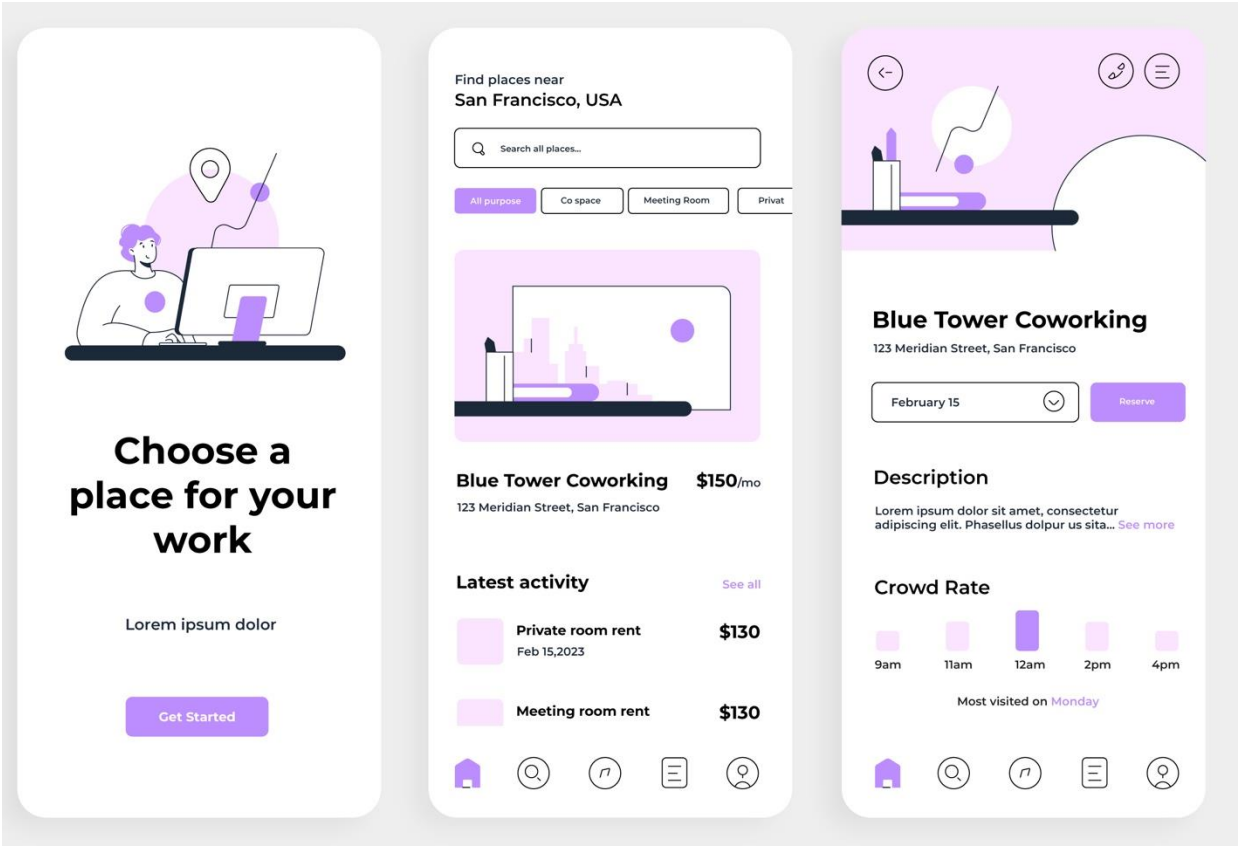
MVC - Visão



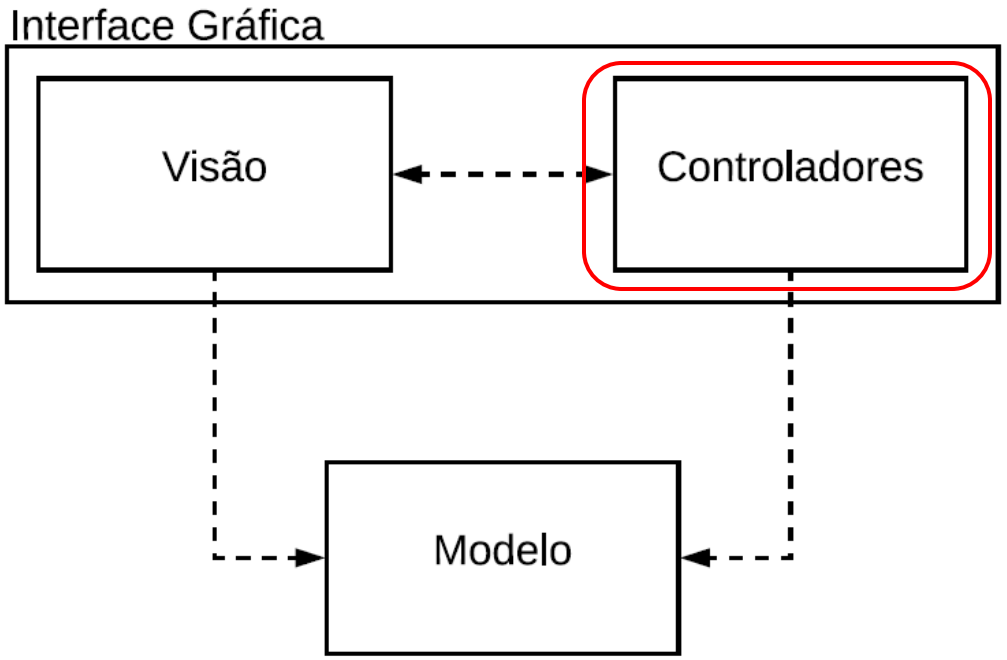
- Responsável pela apresentação das interfaces gráficas do sistema.
- **Ex:** janelas, botões, menus, barras de rolagem, etc

MVC - Visão


Interface Gráfica
Visão
Controladores
Modelo

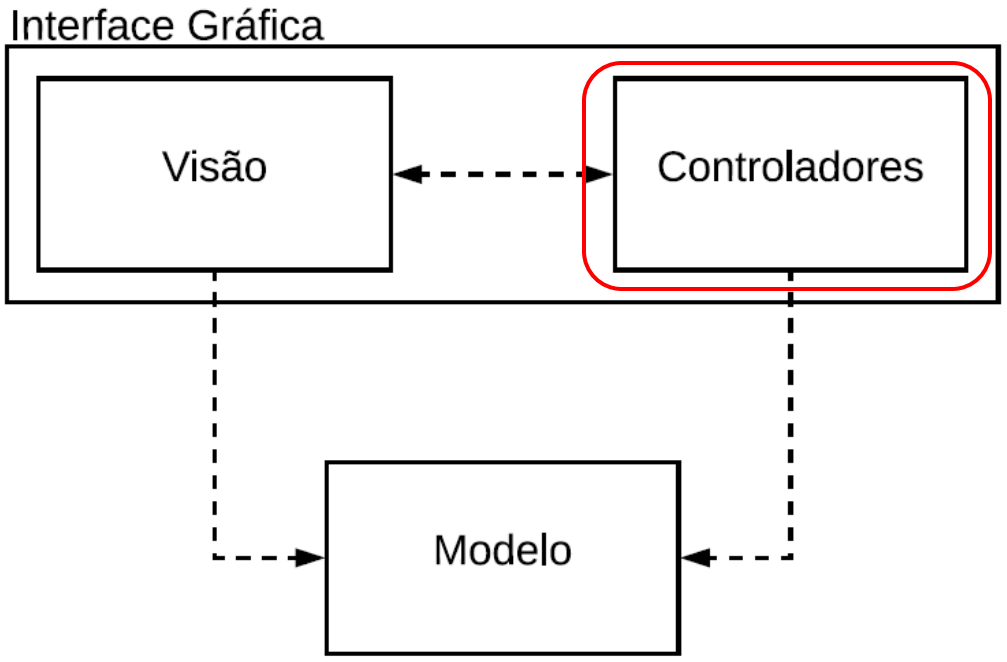

Choose a place for your work
Lorem ipsum dolor
Get Started
Find places near
San Francisco, USA
Search all places...
All purpose
Co space
Meeting Room
Privat
Blue Tower Coworking
$150/mo
123 Meridian Street, San Francisco
Latest activity
See all
Private room rent
Feb 15,2023
$130
Meeting room rent
$130
Blue Tower Coworking
123 Meridian Street, San Francisco
February 15
Reserve
Description
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus dolpur us sita... See more
Crowd Rate
9am
11am
12am
2pm
4pm
Most visited on Monday

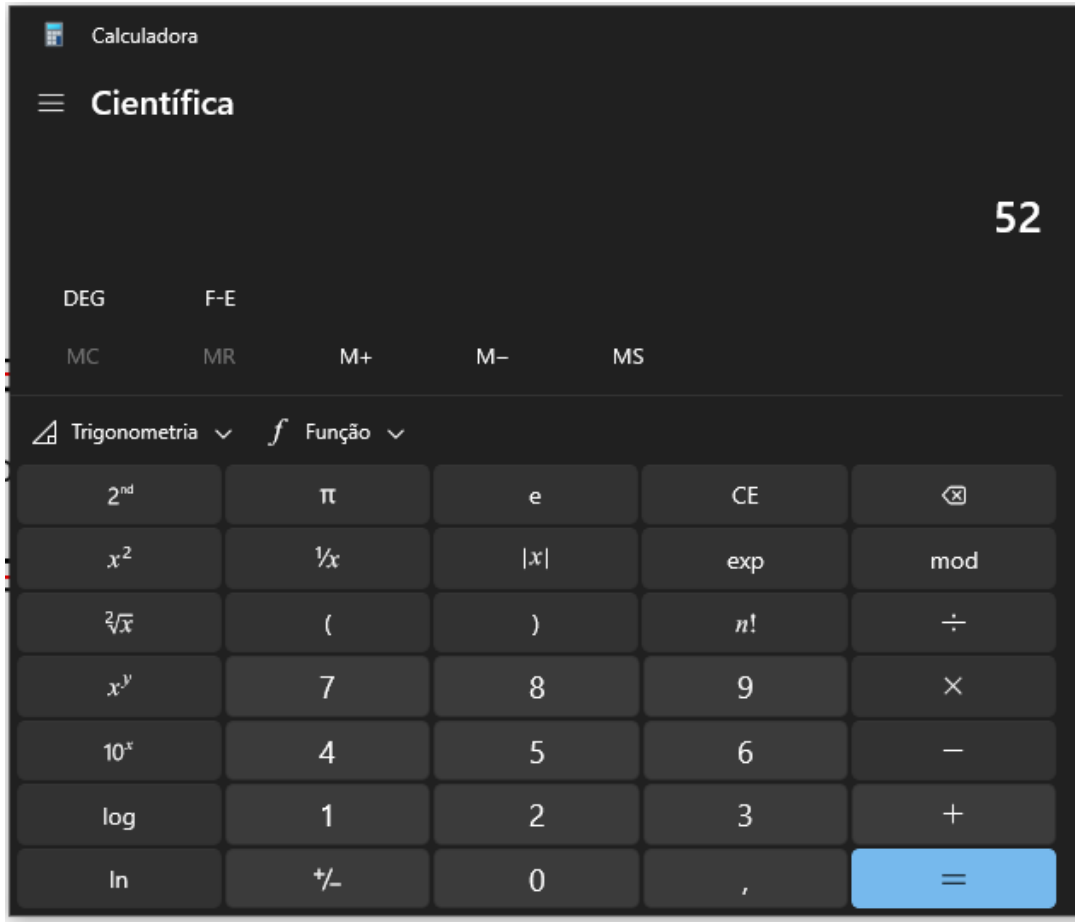
MVC - Controladores



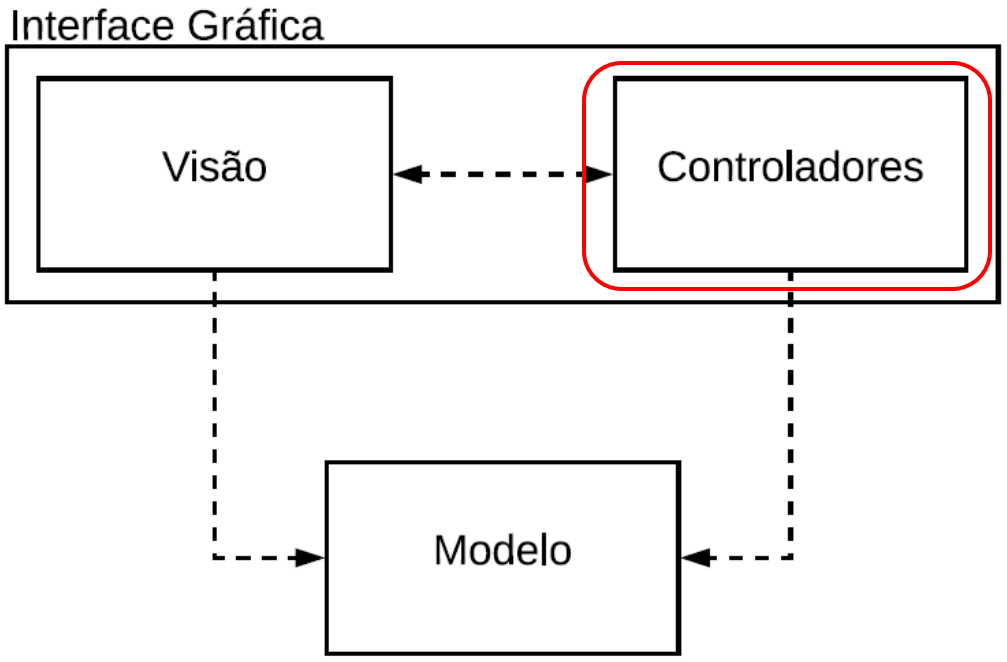
- Classes que tratam e interpretam eventos gerados por dispositivos de entrada.
- **Ex:** Mouse, teclado, toques, etc.
- Controladores podem solicitar uma alteração no estado do Modelo ou da Visão.

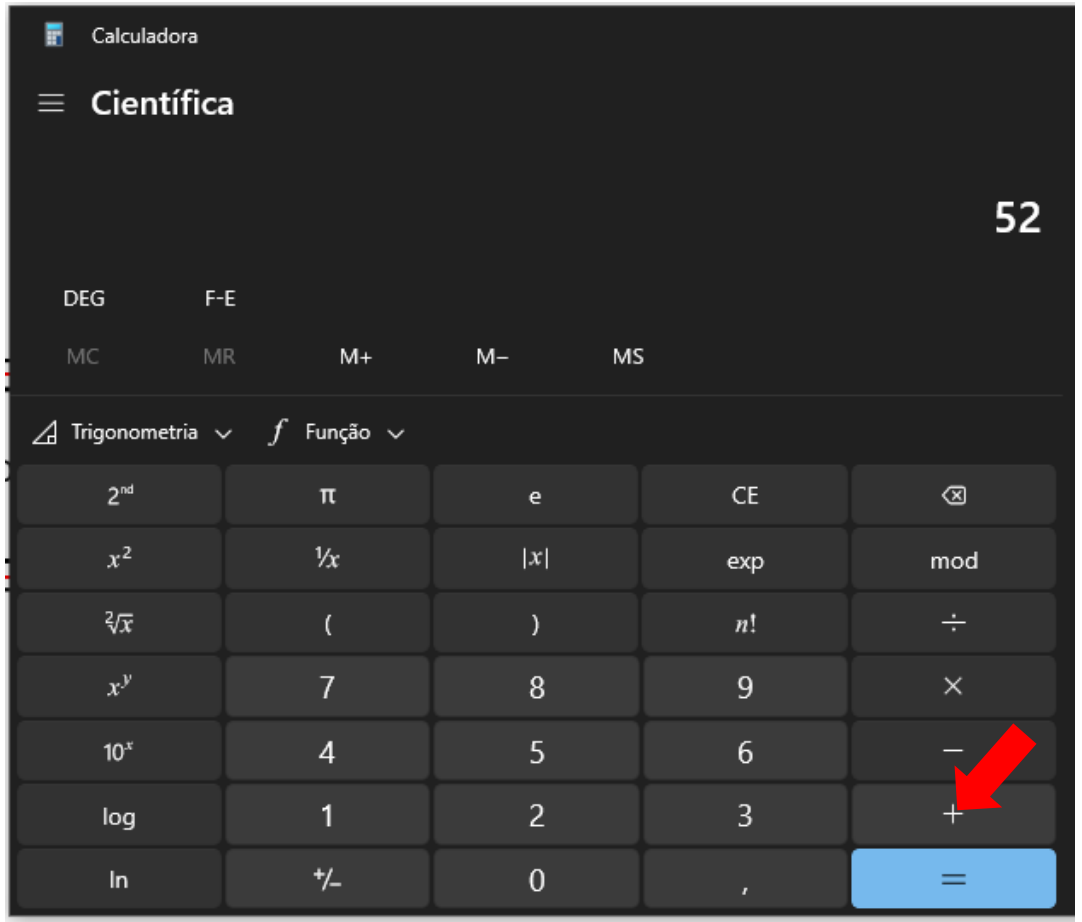
MVC - Controladores





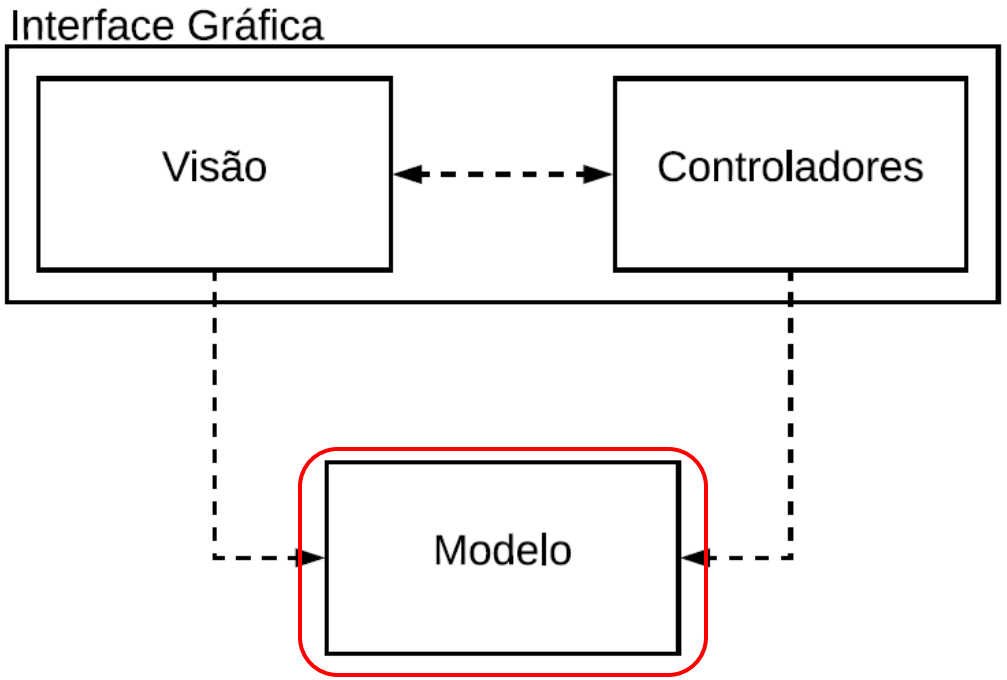
MVC - Controladores





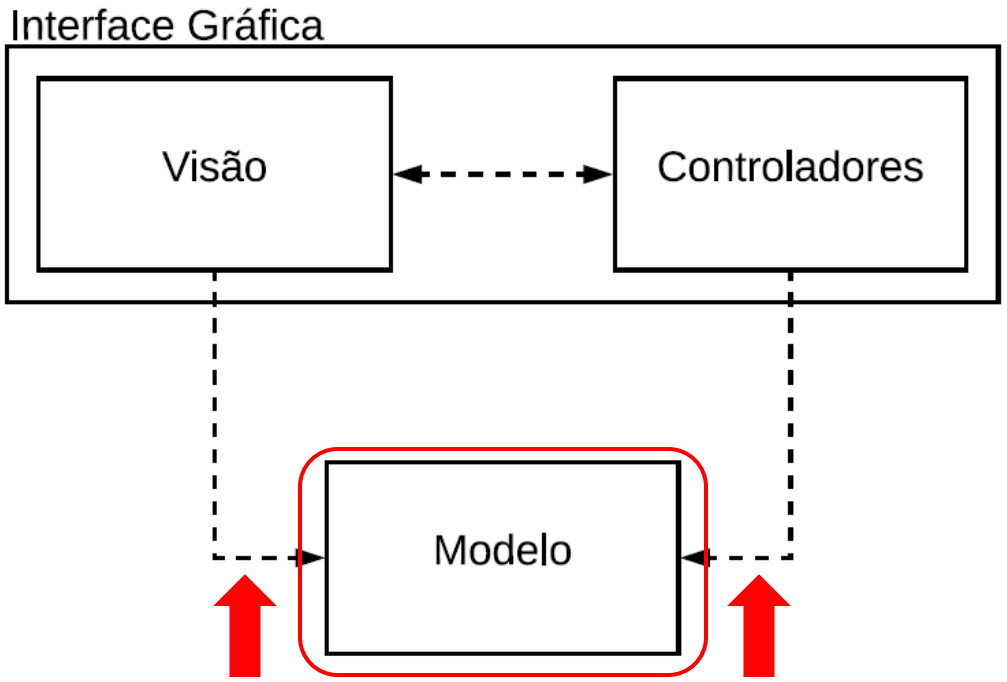
- Ao pressionar o botão “**+**”, uma classe **Controladora** deve capturar esse evento e chamar um método do **Modelo.**

MVC - Modelo



- Classes que armazenam os dados manipulados pela aplicação e que têm a ver com o domínio do sistema em construção.

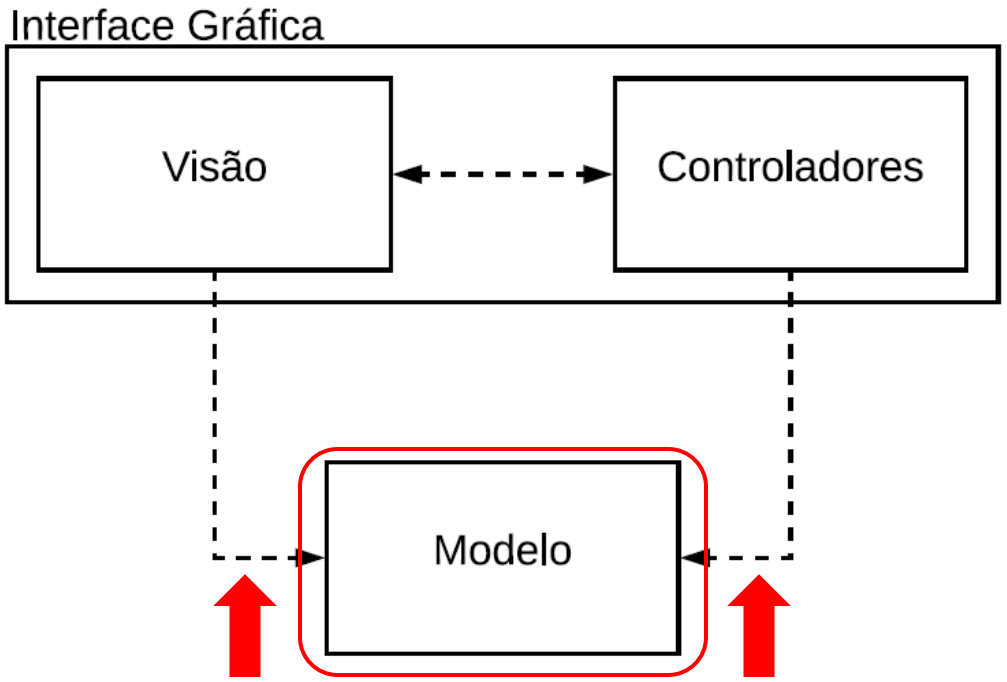
MVC - Modelo



Ex:

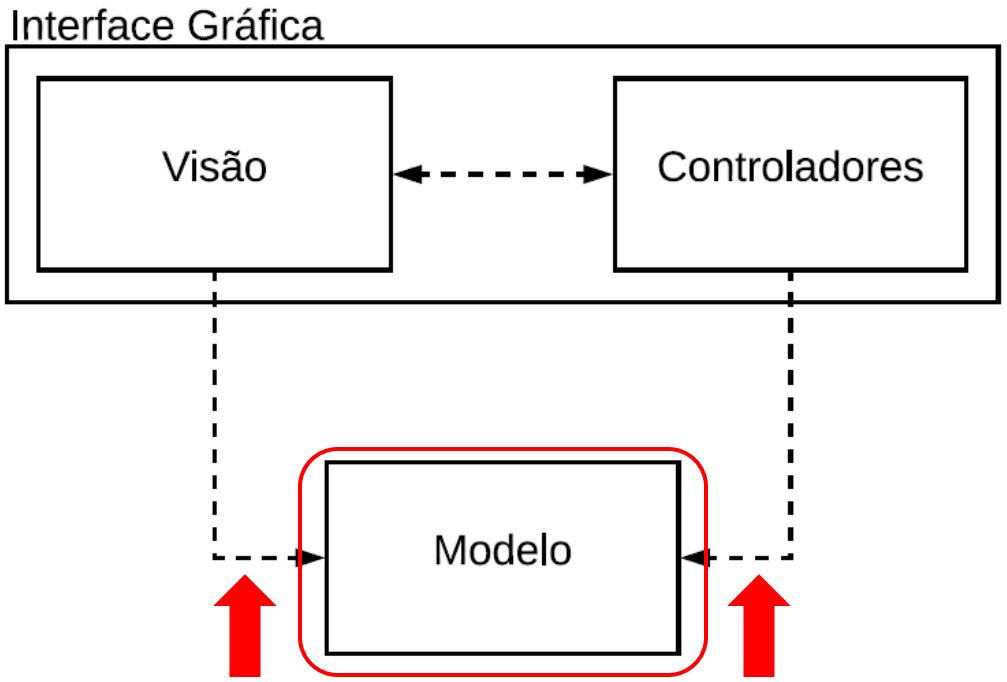
- Uma classe **ContaBancaria** com o atributo **saldo** e os métodos **sacar()** e **depositar()**.
- A classe **ContaBancaria** não sabe:
  - Se o saldo será exibido em uma tela, num gráfico ou num relatório.
  - Quem acionou a operação (um botão de interface, uma API ou um script).
- Ela “só” sabe cuidar da lógica do domínio (como atualizar/fornecer o saldo corretamente).

MVC - Modelo



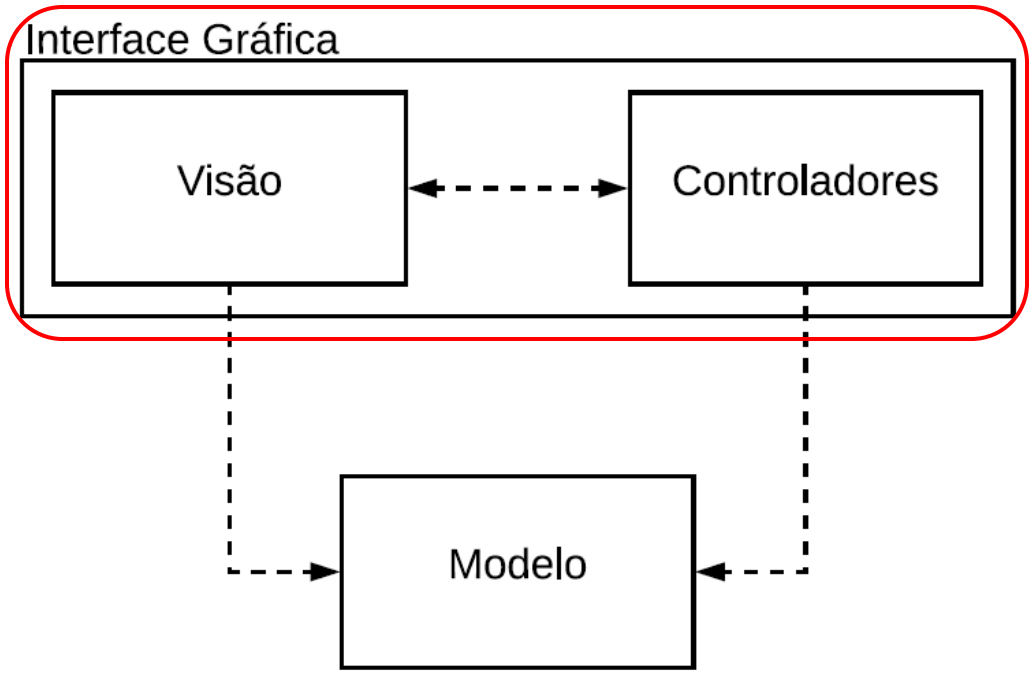
Porque isso é importante?

MVC - Modelo



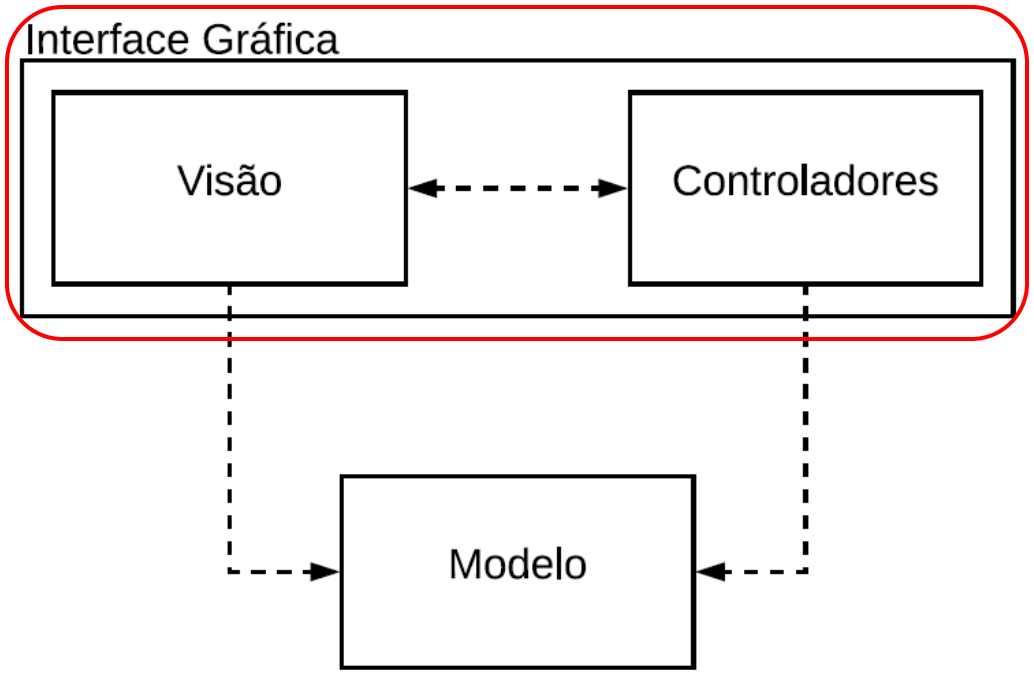
- **Garante independência**: o Modelo pode ser usado em diferentes aplicações (desktop, web, mobile) sem precisar mudar.
- A independência gera Flexibilidade.
- Quem decide como mostrar ou reagir é a View(Visão) e o Controller(Controlador).

MVC – Interface Gráfica



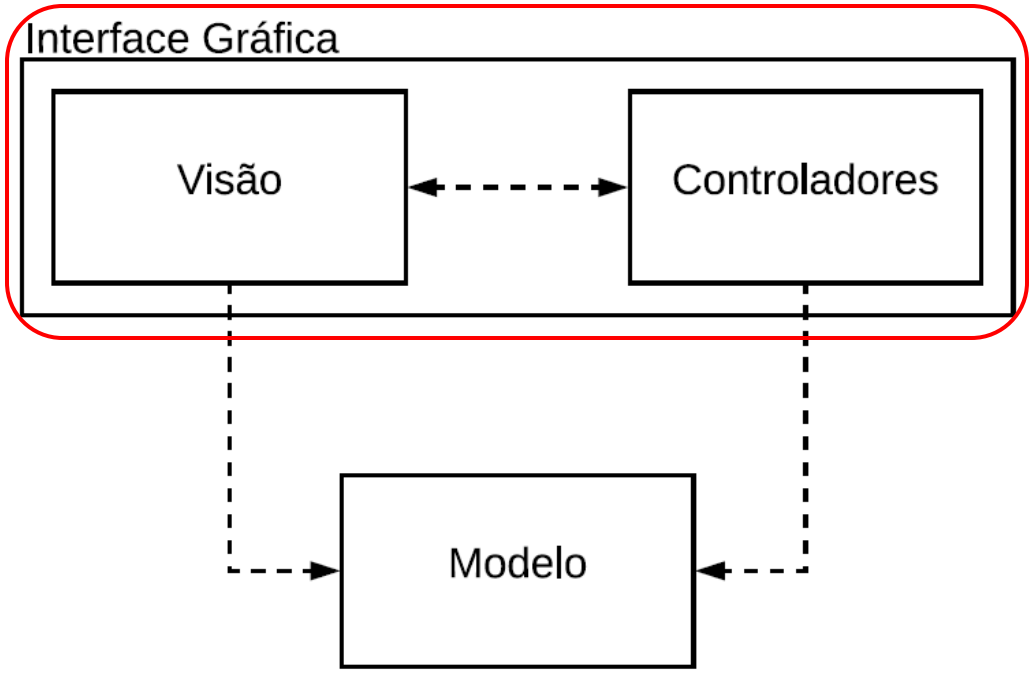
- Em uma arquitetura MVC a interface gráfica é formada por objetos de visão e por controladores.
- Divisão da reponsabilidade da interface!
- O Modelo é responsável por manter e manipular o estado da aplicação, aplicando as regras de negócio e disponibilizando os dados para serem exibidos na interface.

MVC – Interface Gráfica



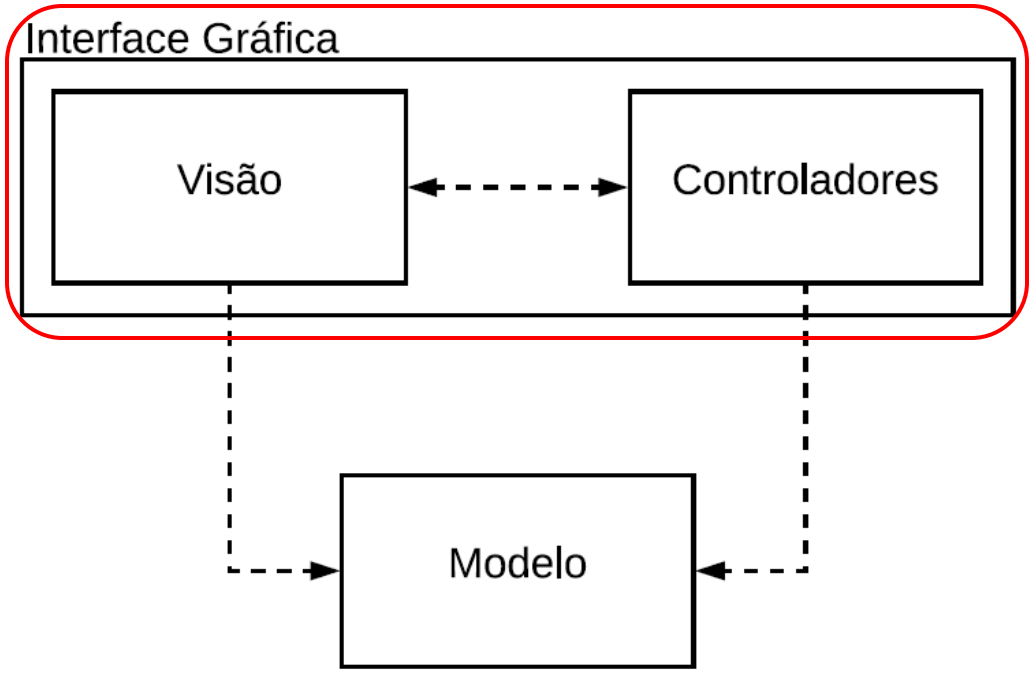
- Em uma arquitetura MVC a interface gráfica é formada por objetos de visão e por controladores.
- Divisão da reponsabilidade da interface!
- O Modelo é responsável por manter e manipular o estado da aplicação, aplicando as regras de negócio e disponibilizando os dados para serem exibidos na interface.

MVC – Interface Gráfica



- Em muitos sistemas não existe uma distinção clara entre Visão e Controladores
- Inclusive, a maioria das versões do Smalltalk não separa esses dois componentes.

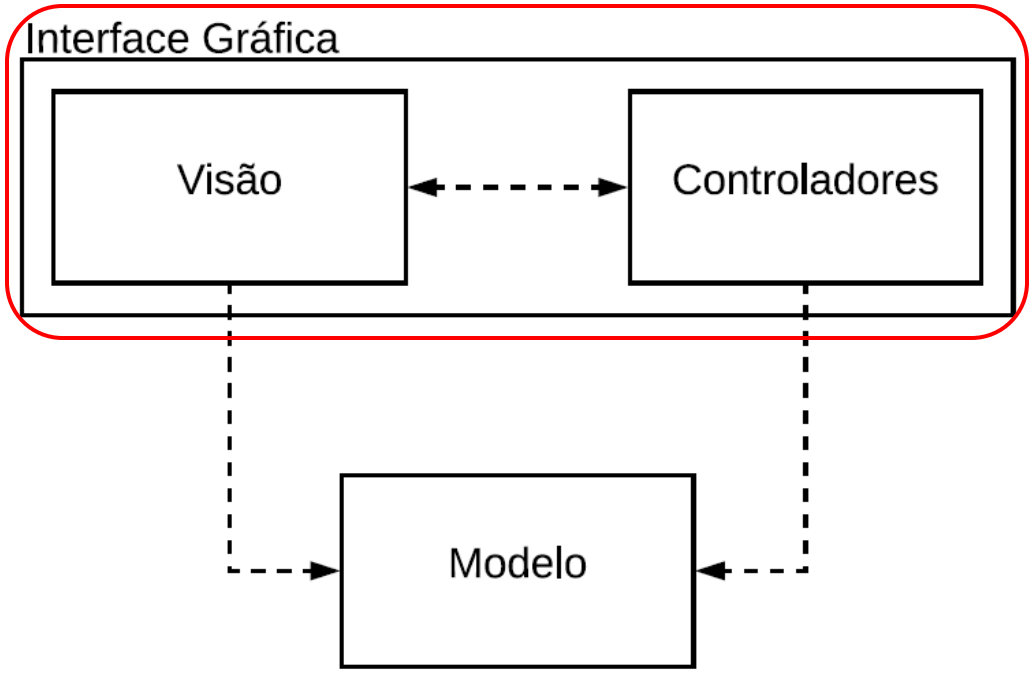
MVC – Interface Gráfica



(Visão + Controladores) + *Modelo* =

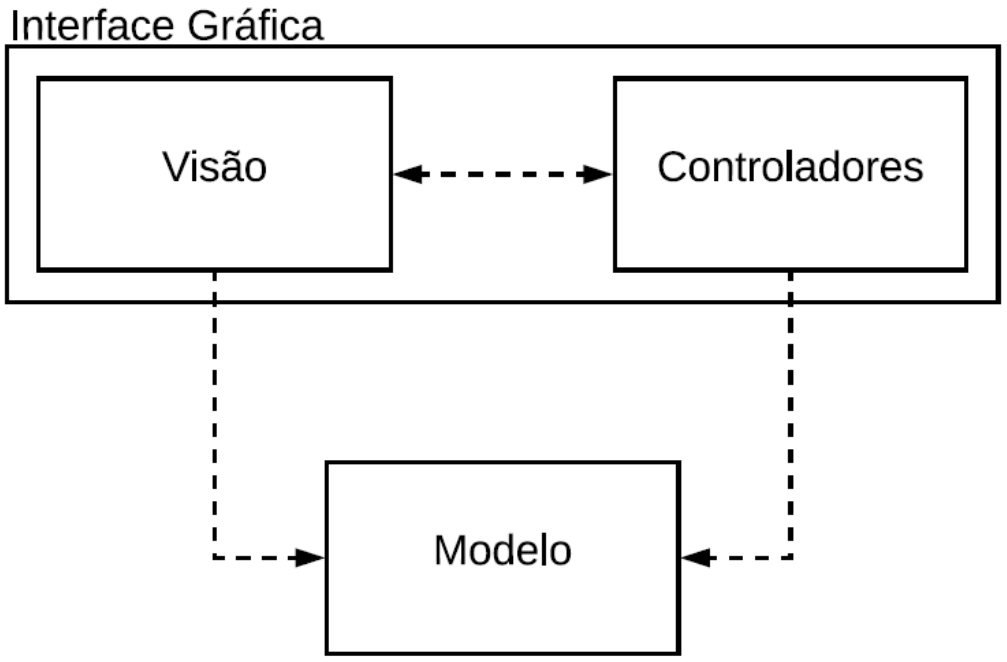
Interface Gráfica + *Modelo*

MVC – Interface Gráfica

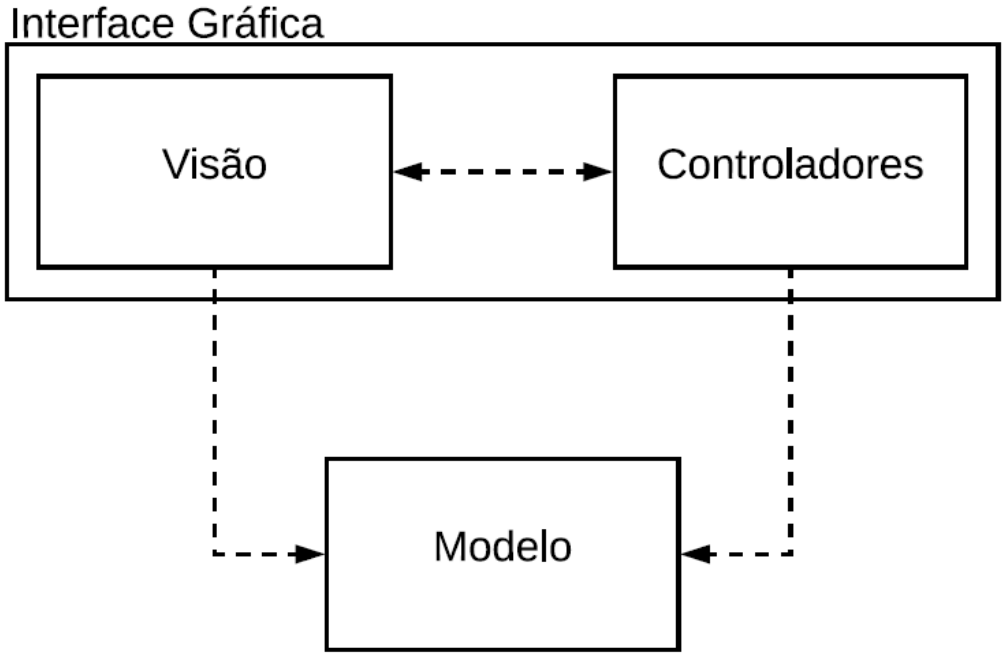


- A Interface Gráfica pode depender do Modelo. Porém, classes de Modelo não têm dependências para classes da Interface Gráfica.

MVC – Metáfora

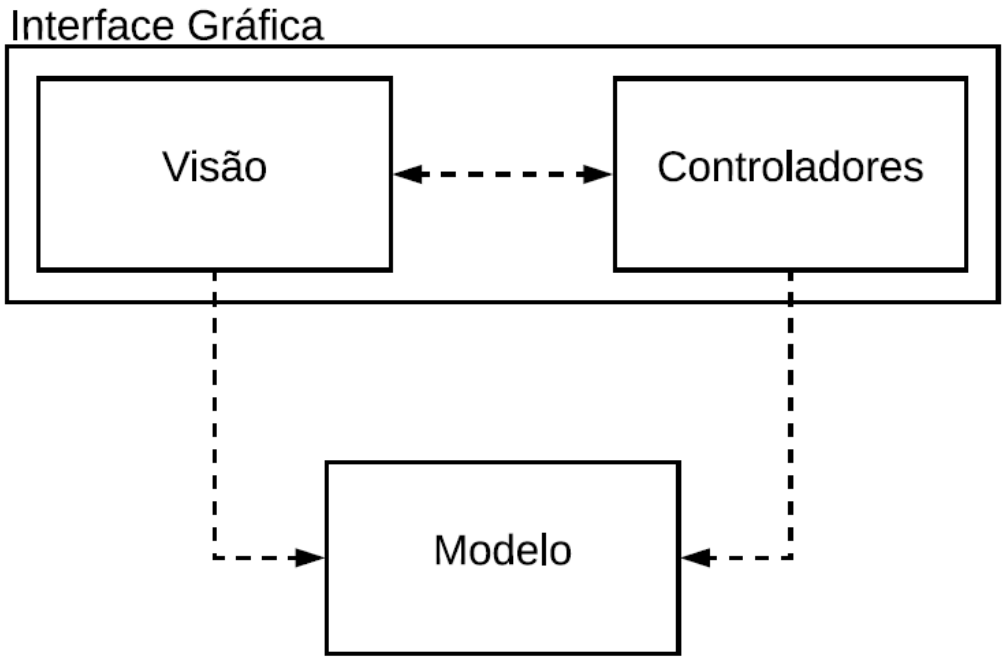


- **Modelo (o roteiro da peça)**
  - Contém toda a história: os personagens, os diálogos e as regras do enredo.
  - Não se preocupa em como a plateia verá a peça, só guarda **o conteúdo e a lógica da narrativa**.
- **Visão (o palco e os atores)**
  - São responsáveis por **mostrar a história ao público**.
  - Vestem figurinos, usam cenários e dão forma visual ao roteiro.
  - Não decidem o que acontece, apenas **encenam** o que está no roteiro.
- **Controlador (o diretor da peça)**
  - Interpreta os sinais do público (aplausos, risadas, vaias) e ajusta a condução da apresentação.
  - Diz aos atores **como agir** diante de cada situação.
  - Faz a ponte entre o público e a encenação.



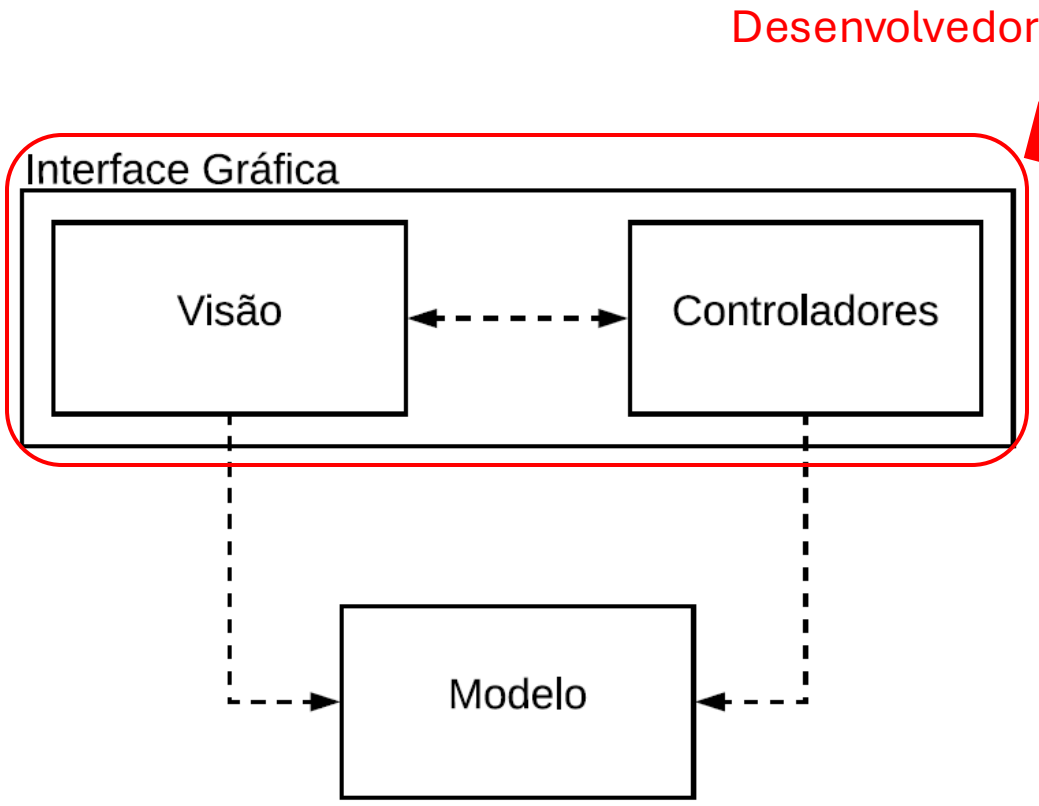
Quais as vantagens da arquitetura MVC?

MVC – Vantagens



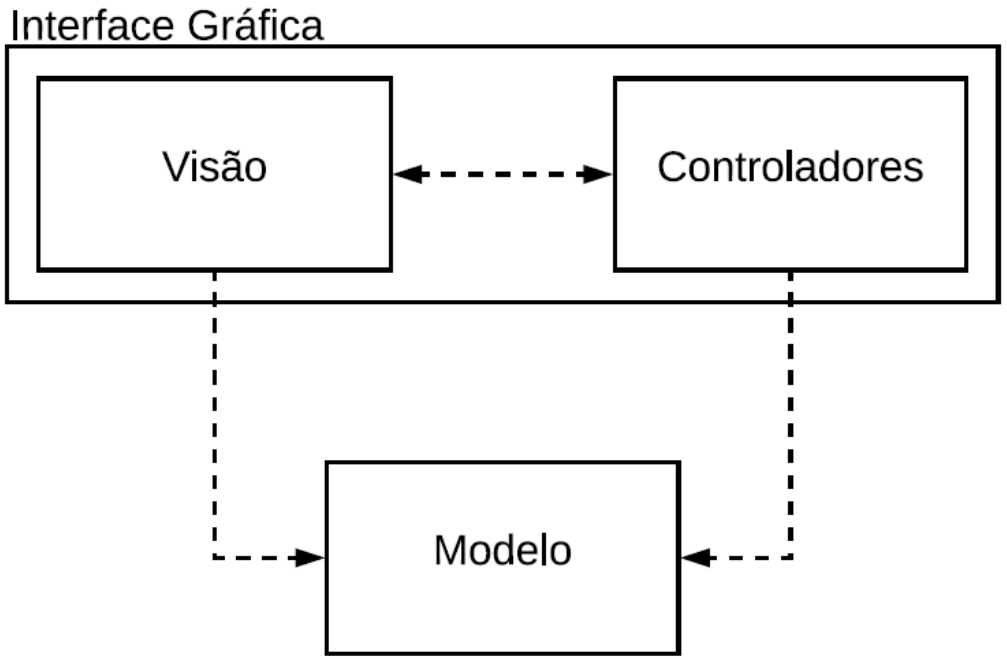
- MVC favorece a especialização do trabalho de desenvolvimento.

MVC – Vantagens



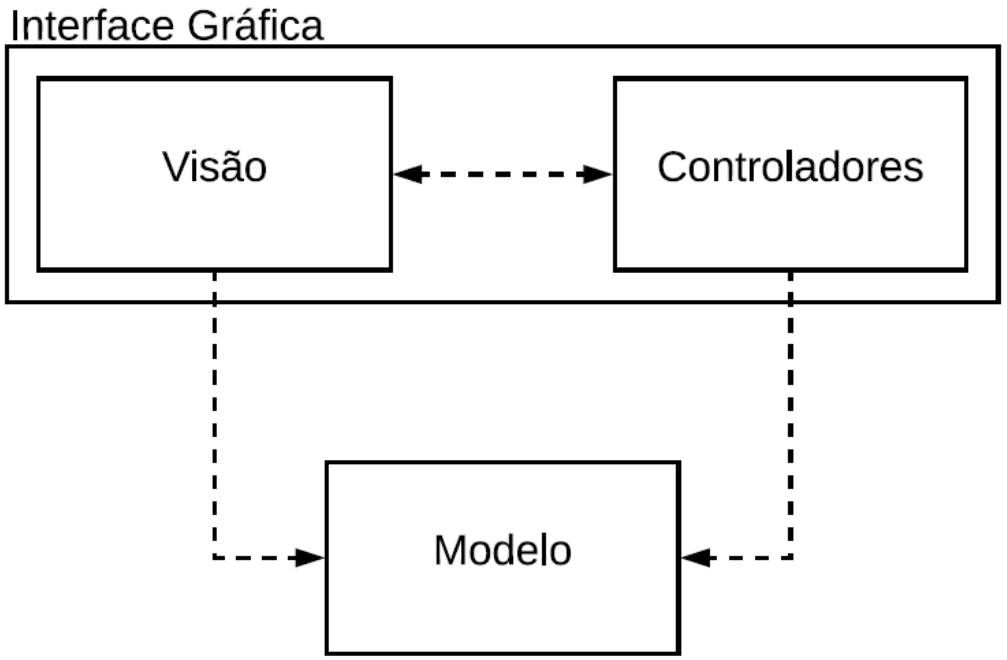
- MVC favorece a especialização do trabalho de desenvolvimento.
- Ex: Desenvolvedores especialistas na implementação de interfaces gráficas (*desenvolvedores front-end*).

MVC – Contexto de Desenvolvimento



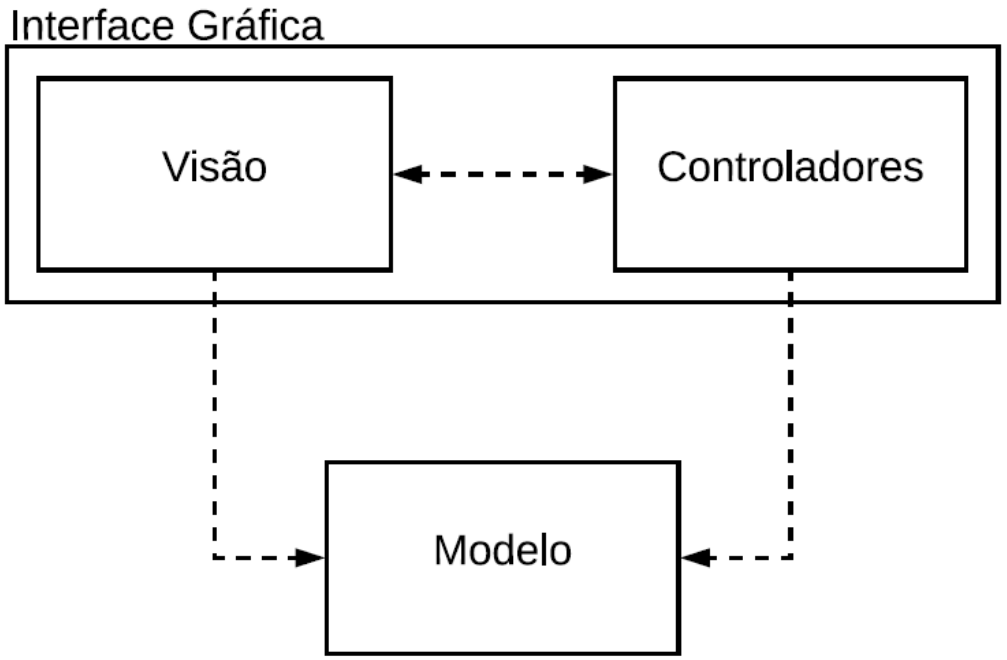
- Visão (View)
  - Representa a interface com o usuário.
  - Hoje: telas em HTML/CSS, componentes React/Vue/Angular, widgets móveis.
  - Normalmente front-end developers trabalham aqui.
- Controlador (Controller)
  - Interpreta os eventos do usuário (cliques, requisições, comandos) e os traduz em ações sobre o Modelo.
  - No front-end moderno, parte desse papel pode ser feito por JavaScript/TS ou frameworks (event handlers).
  - **Mas em aplicações web clássicas (ex: Spring MVC, Django), o Controller também fica no back-end, recebendo requisições HTTP.**
- Modelo (Model)
  - Mantém o estado e as regras de negócio.
  - Inclui persistência em banco de dados, lógica de domínio, validações.
  - Tipicamente responsabilidade de back-end developers.

MVC – Vantagens



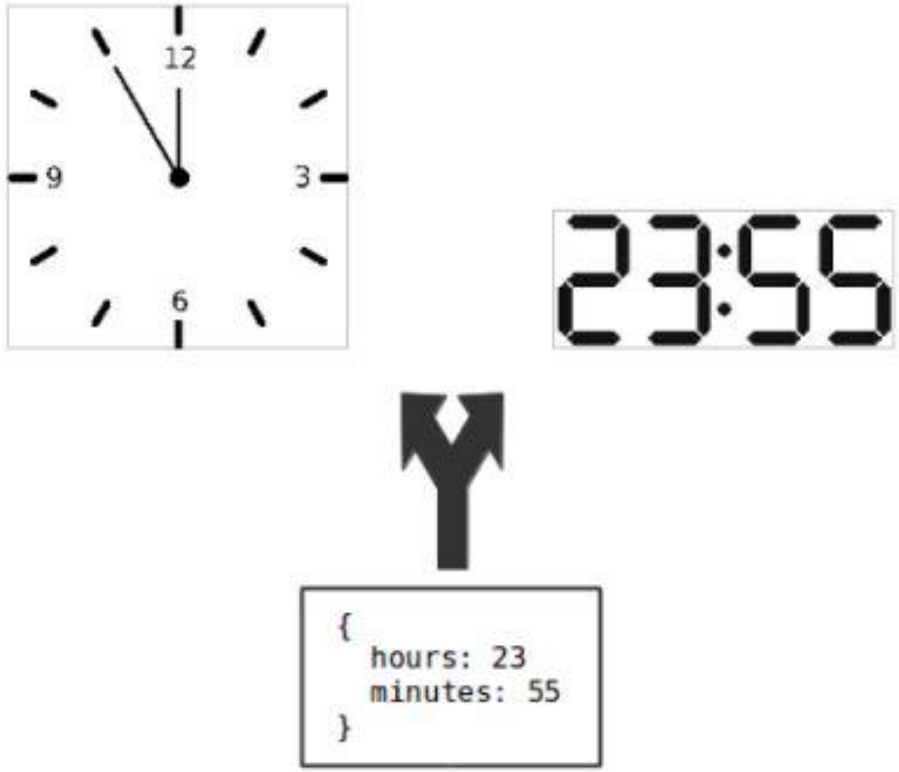
- Desenvolvedores de classes de Modelo não precisam conhecer e implementar código de interface com usuários.

MVC – Vantagens



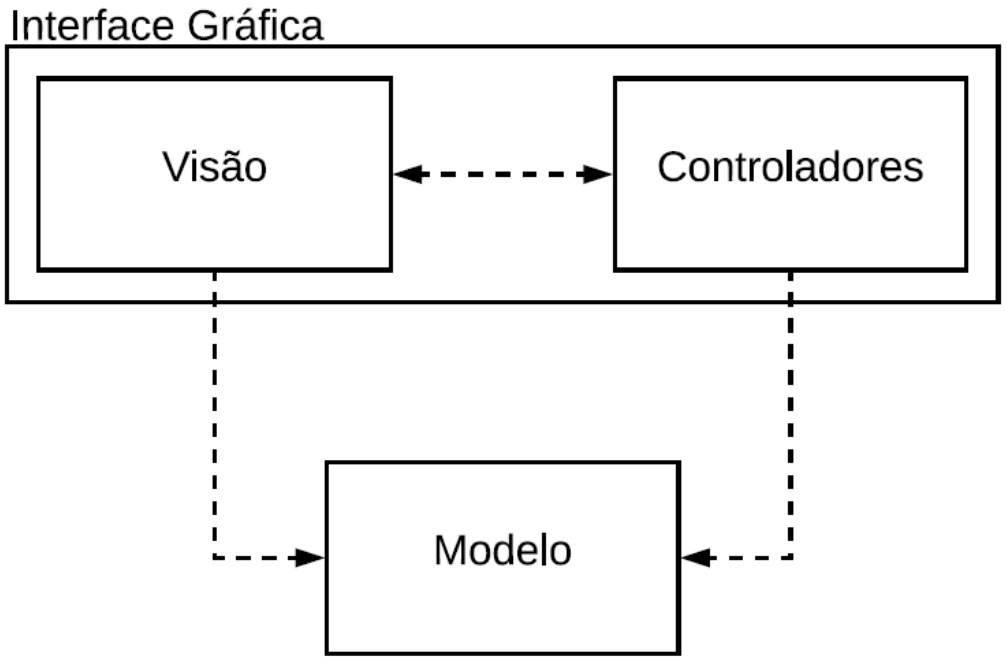
- Reutilização de classes de Modelo por diferentes Visões

Exemplo



- Um objeto de Modelo armazena dois valores (hora e minutos).
- Duas visões diferentes, usando os mesmos valores

MVC – Vantagens



- Favorece a **testabilidade.**
  - É mais fácil testar objetos não relacionados com a implementação de interfaces gráficas.
  - Ao separar objetos de apresentação de objetos de Modelo, fica mais fácil testar esses últimos

MVC – Ponto chave

- A arquitetura MVC **desacopla** a interface do usuário da funcionalidade e do conteúdo de informação do sistema.

- Se os dados sofrerem alterações a partir das apresentações, então o modelo do sistema é alterado, assim como os controladores que estão associados a cada visão, fazendo a atualização da apresentação.

Qual a diferença entre MVC e três camadas?

MVC x Três Camadas

- Para obter a resposta precisamos entender o contexto histórico das arquiteturas e a evolução tecnológica das ferramentas utilizadas no desenvolvimentos dos sistemas.

MVC

- MVC surgiu no final da década de 70, para ajudar na construção de interfaces gráficas.

- O foco no momento eram aplicações Desktop.
  - Aplicações que incluem uma interface com janelas, botões, caixas de texto, etc.
  - **Ex:** Aplicações como Word, Excel e Powerpoint.

MVC

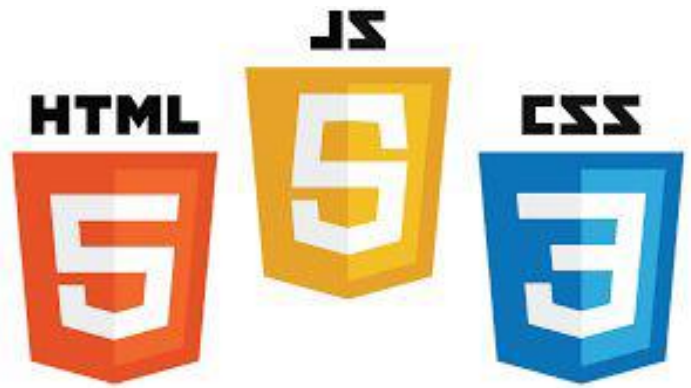
- Na década de 90, as tecnologias de redes, sistemas distribuídos e bancos de dados se tornaram comuns.

- Isso possibilitou a construção de aplicações distribuídas com três camadas.

- **Nesse caso, MVC pode ser usado na implementação da camada de interface, que pode, por exemplo, ser uma aplicação nativa em Windows, implementada usando-se linguagens como Visual Basic ou Java.**

MVC

- No início dos anos 2000, a Web se popularizou e a interface das aplicações migrou para HTML e, depois, para HTML e JavaScript.



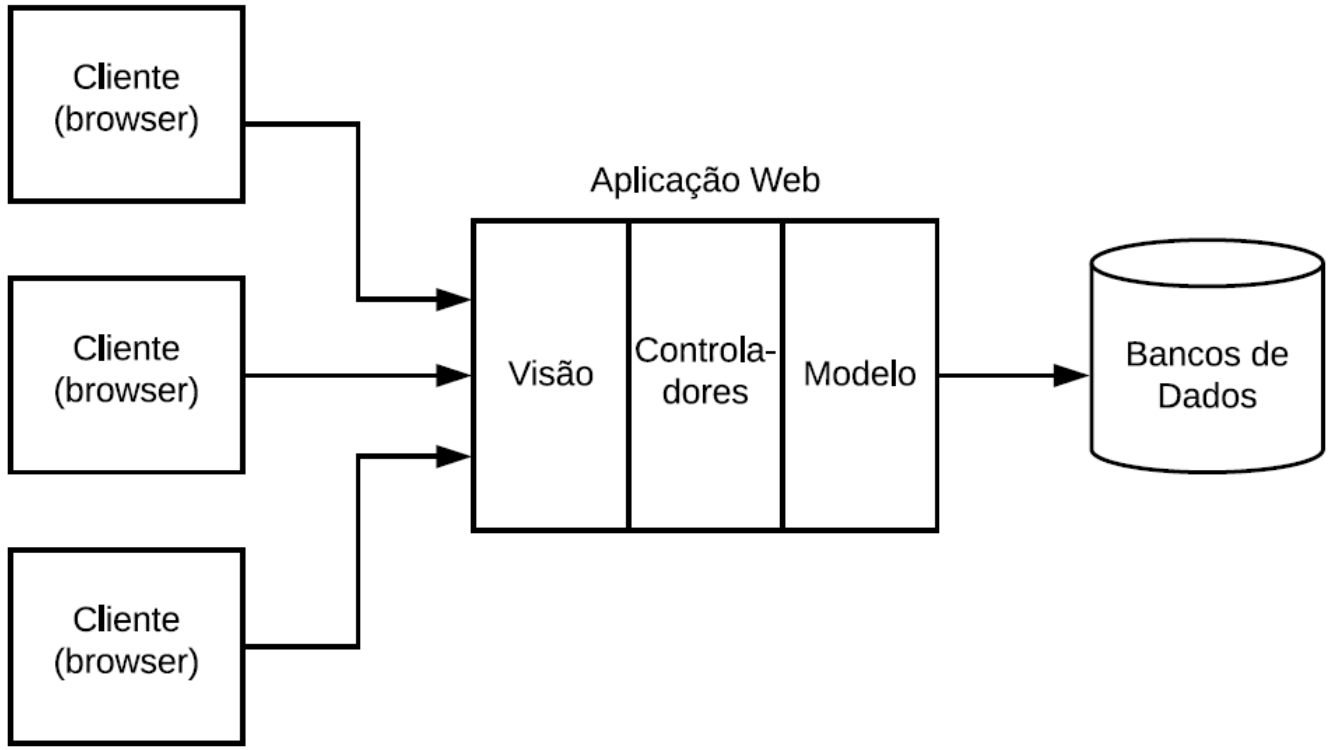
MVC

- A confusão entre os termos MVC e três camadas surgiu nessa época.

- Isso se deu devido ao aparecimento de frameworks para implementação de sistemas Web que se denominaram frameworks MVC.

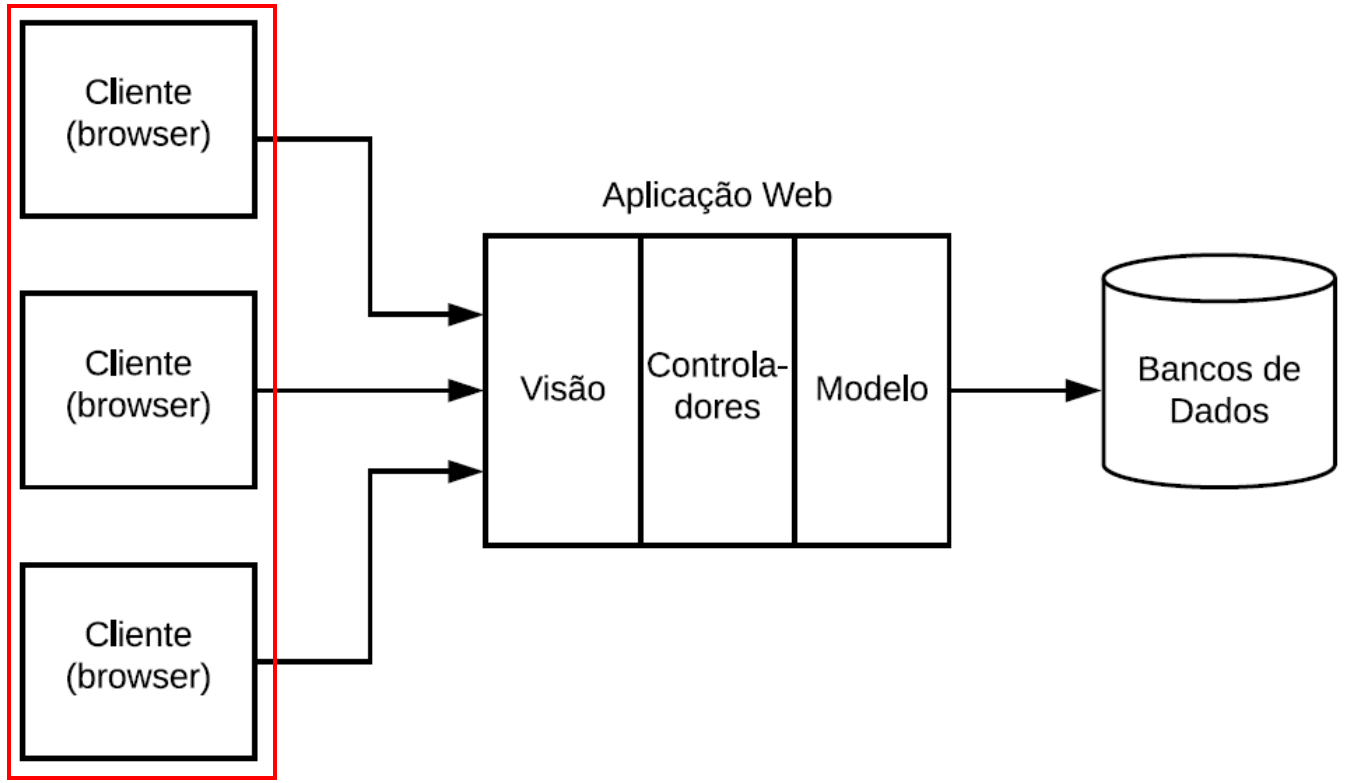
MVC em sistemas web

- Eles forçam a organização de um sistema Web em três partes.
- **Visão**: composta por páginas HTML.
- **Controladores:** processam uma solicitação e geram uma nova visão como resposta.
- **Modelo**: camada que persiste os dados em um banco de dados.



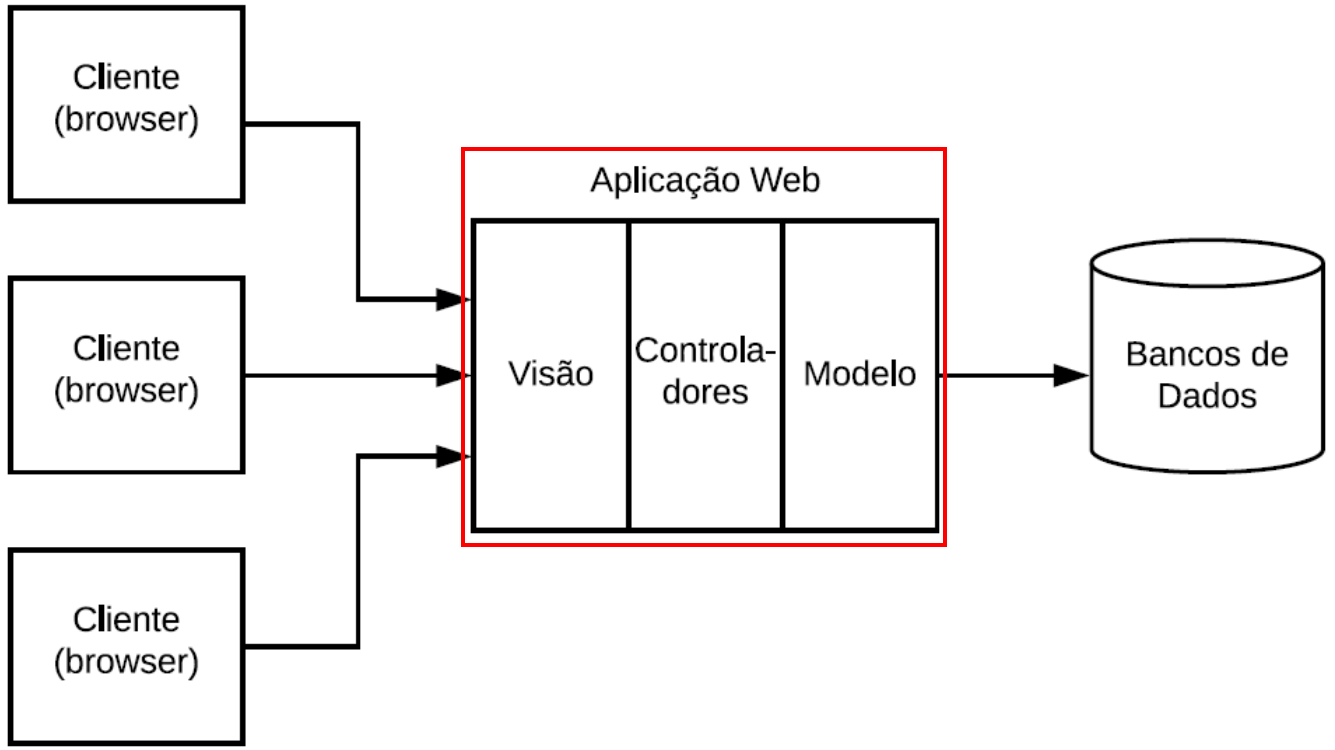
MVC em sistemas web

- Distribuída
- Temos browsers que exibem páginas Web (interface HTML, JavaScript, etc)



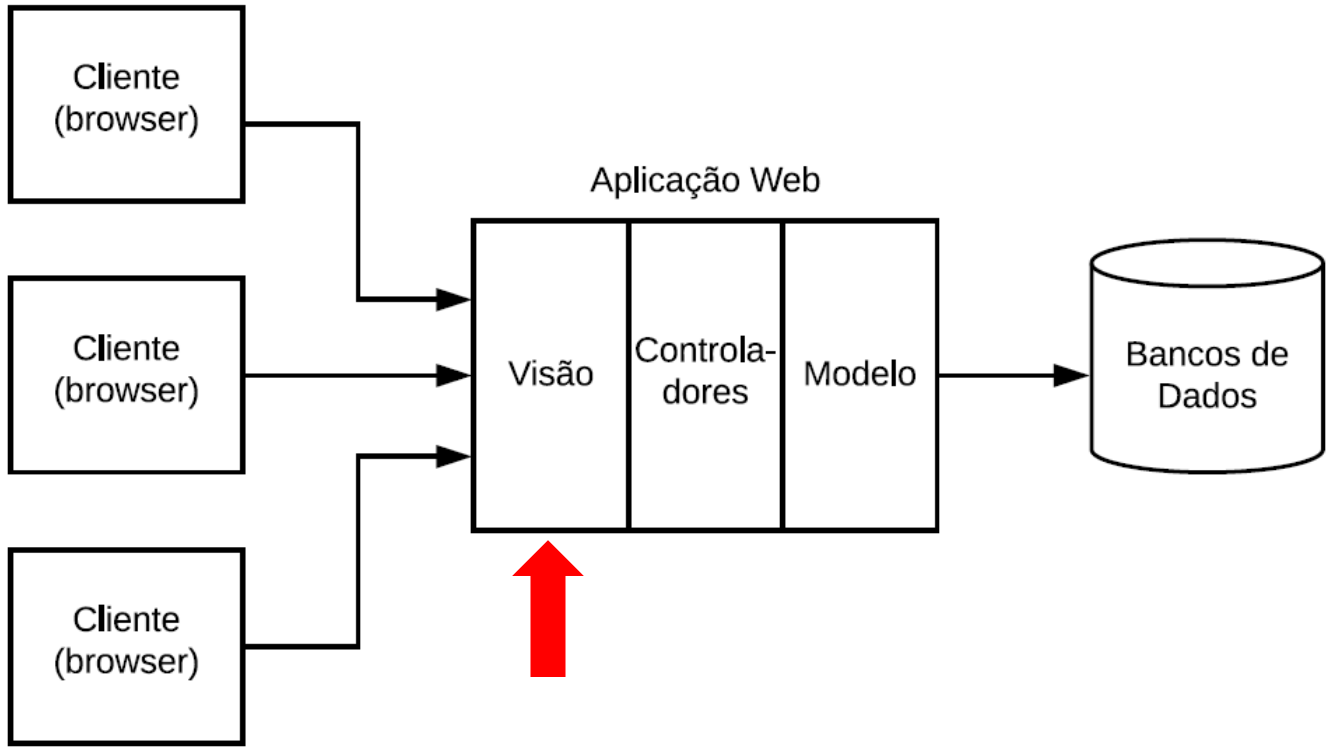
MVC em sistemas web

- MVC está na aplicação WEB
  - Executa no servidor web.
  - Primeiro componente é o browser
  - Segundo componente é o servidor em si.
  - MVC é utilizado para organizar essa aplicação.



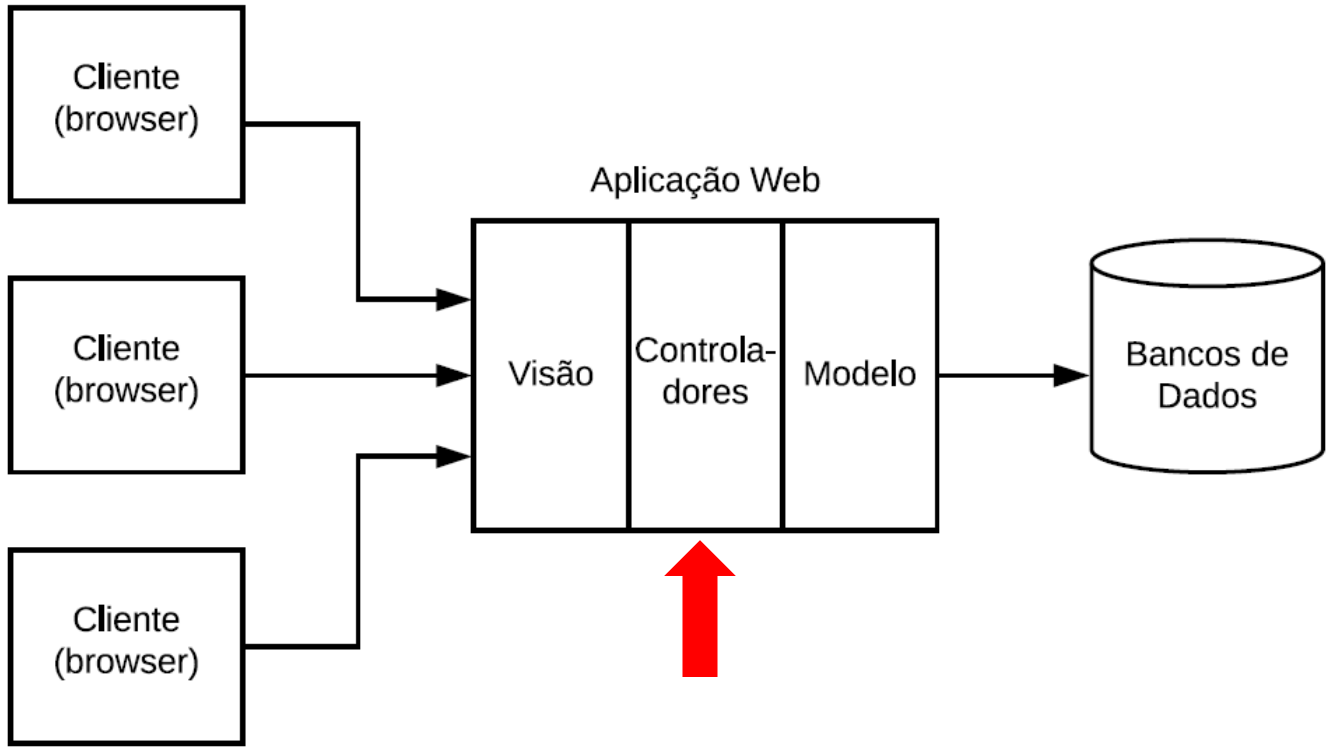
MVC em sistemas web

- **Visão:**
  - Paginas HTML, CSS, JavaScript, enviados para o browser para o cliente ver e interagir.



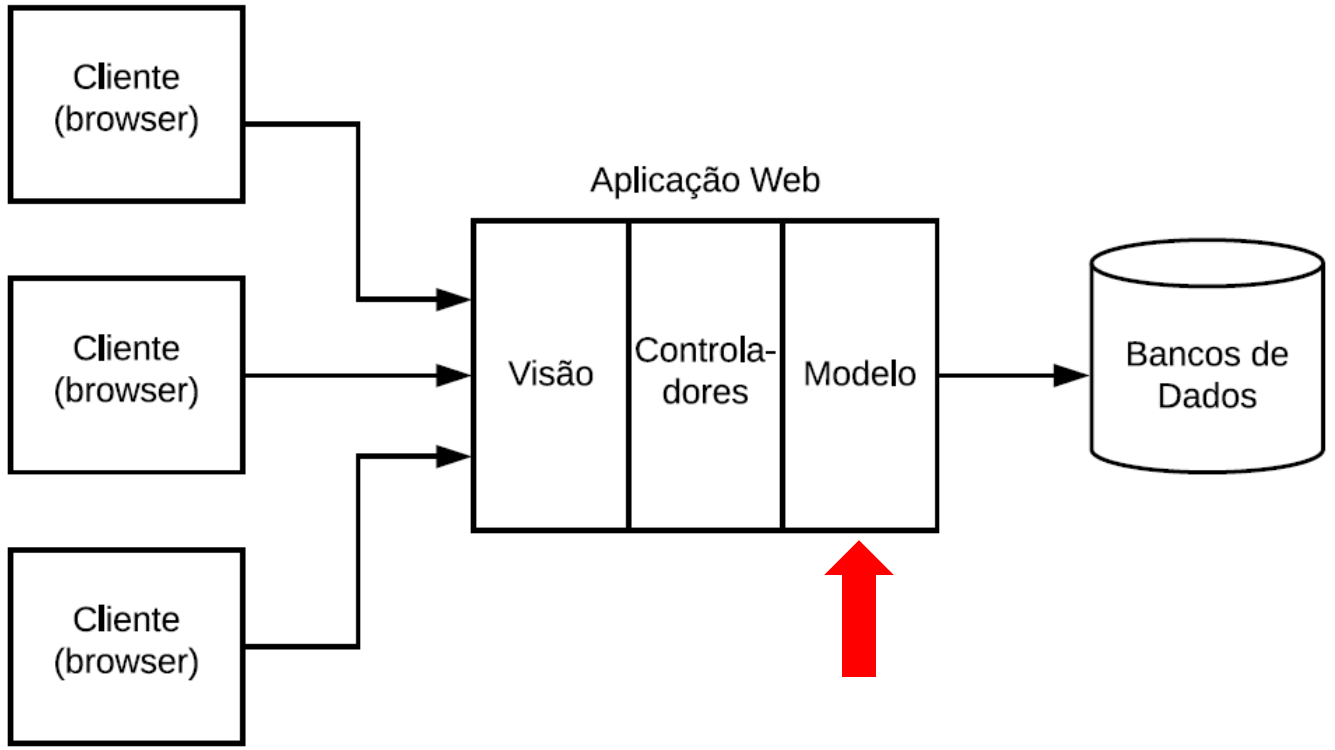
MVC em sistemas web

- **Controladores:**
  - Fornecem os dados para paginas HTML, recebem dados de entradas, fornecem informações para páginas de saída.



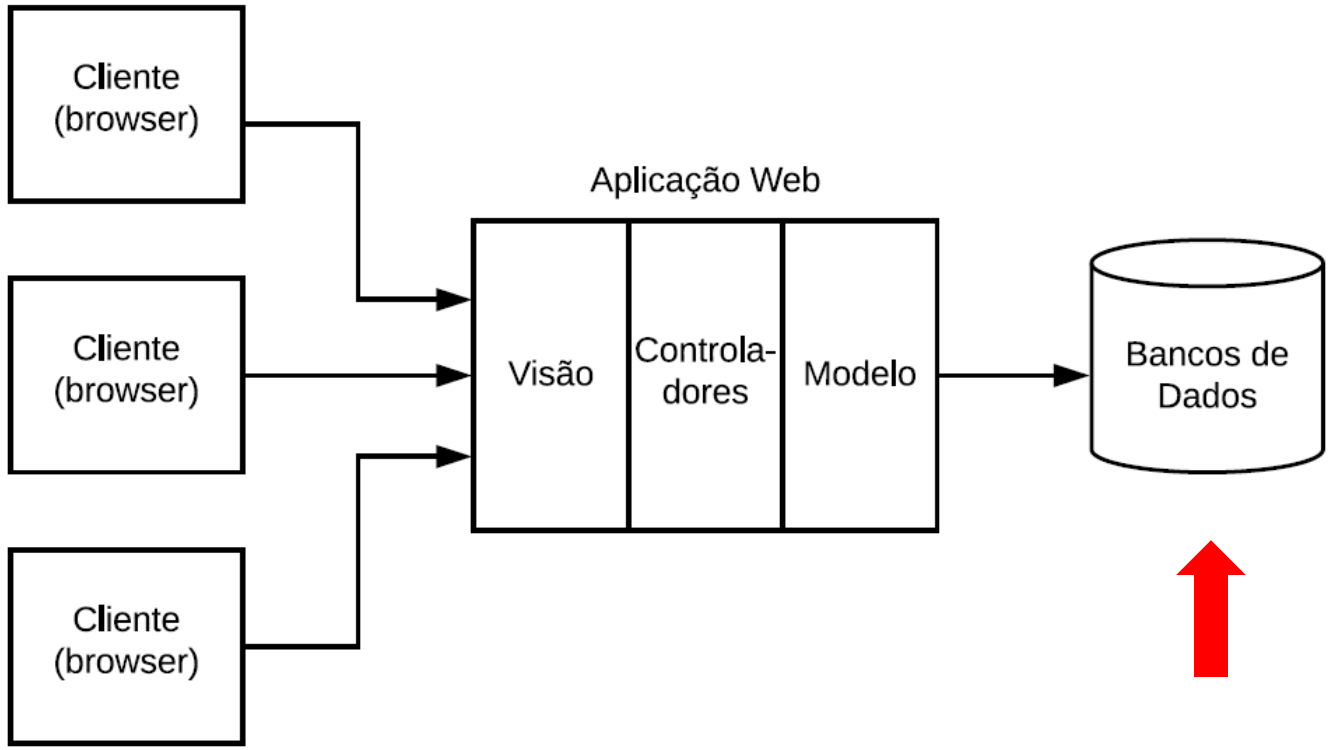
MVC em sistemas web

- **Modelo**
  - Persiste os dados em um banco de dados.



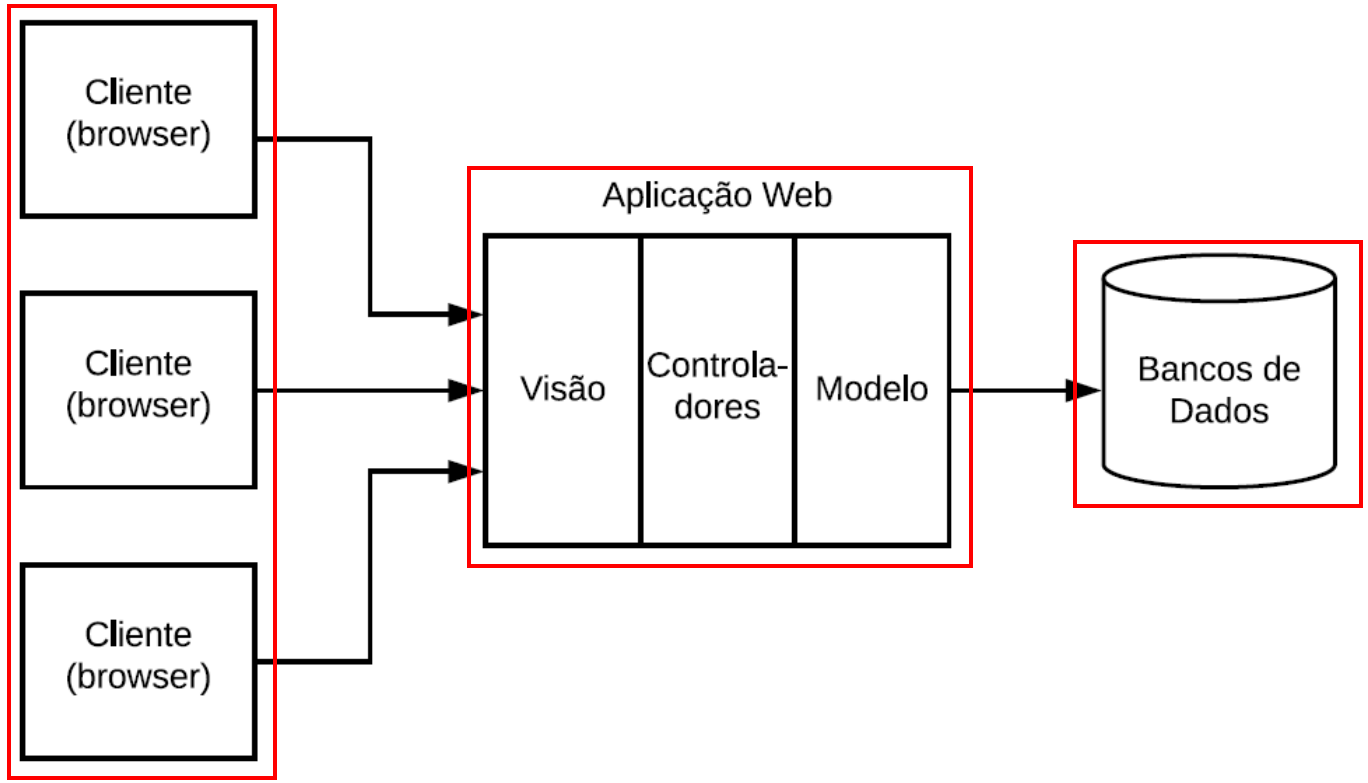
MVC em sistemas web

- O banco de dados é o terceiro componente.



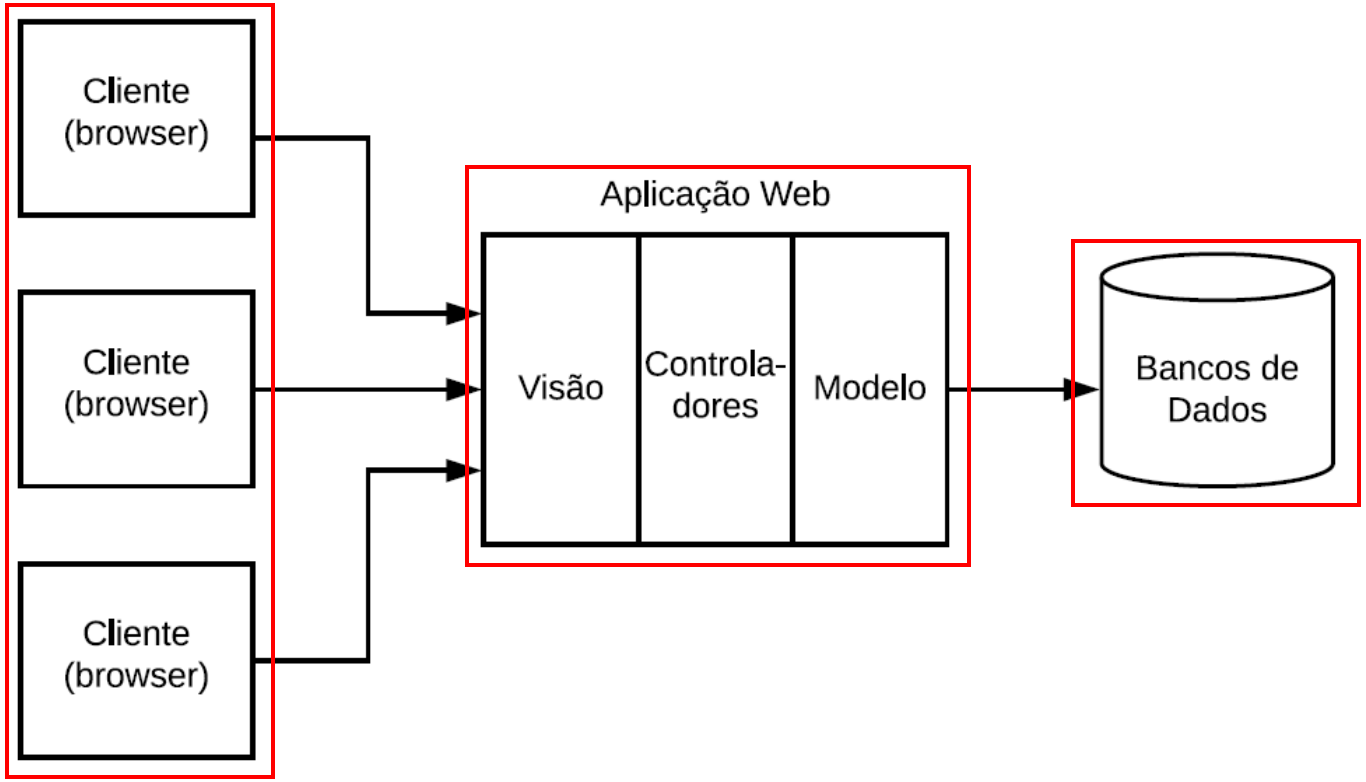
MVC em sistemas web

- **Parece uma arquitetura com 3 camadas!**

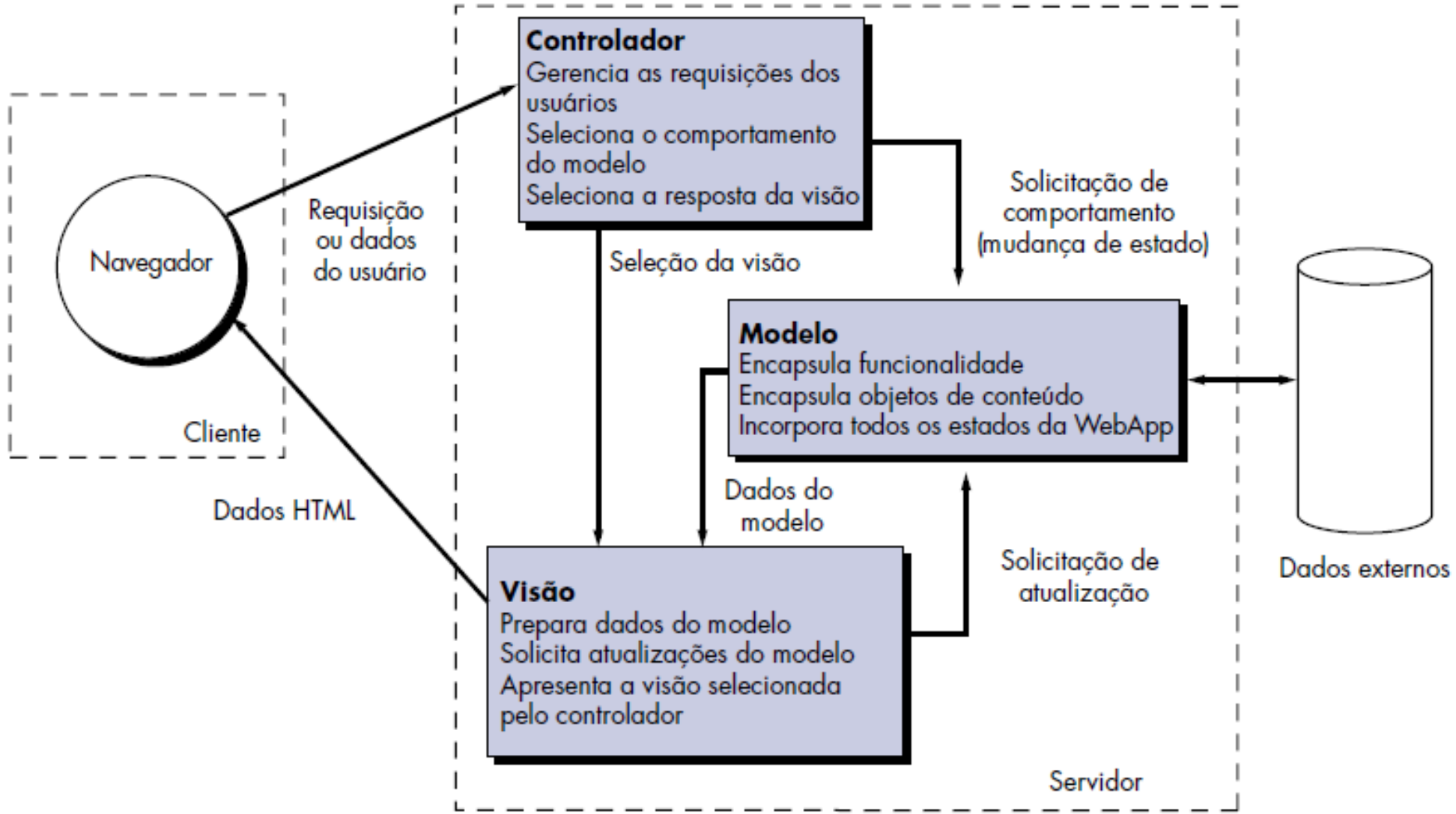


MVC em sistemas web

- Os sistemas WEB são parecidos com sistemas de três camadas.
- Porém, os frameworks Web mais populares optaram por usar termos típicos de MVC para nomear seus componentes



Sistema Web Usando MVC



MVC

- Existem duas vertentes de sistemas MVC:

1. A vertente clássica, que surgiu com Smalltalk-80;
2. A vertente Web, que se tornou comum na década de 90 e início dos anos 2000, lembrando bastante sistemas três camadas.

MVC - Variações

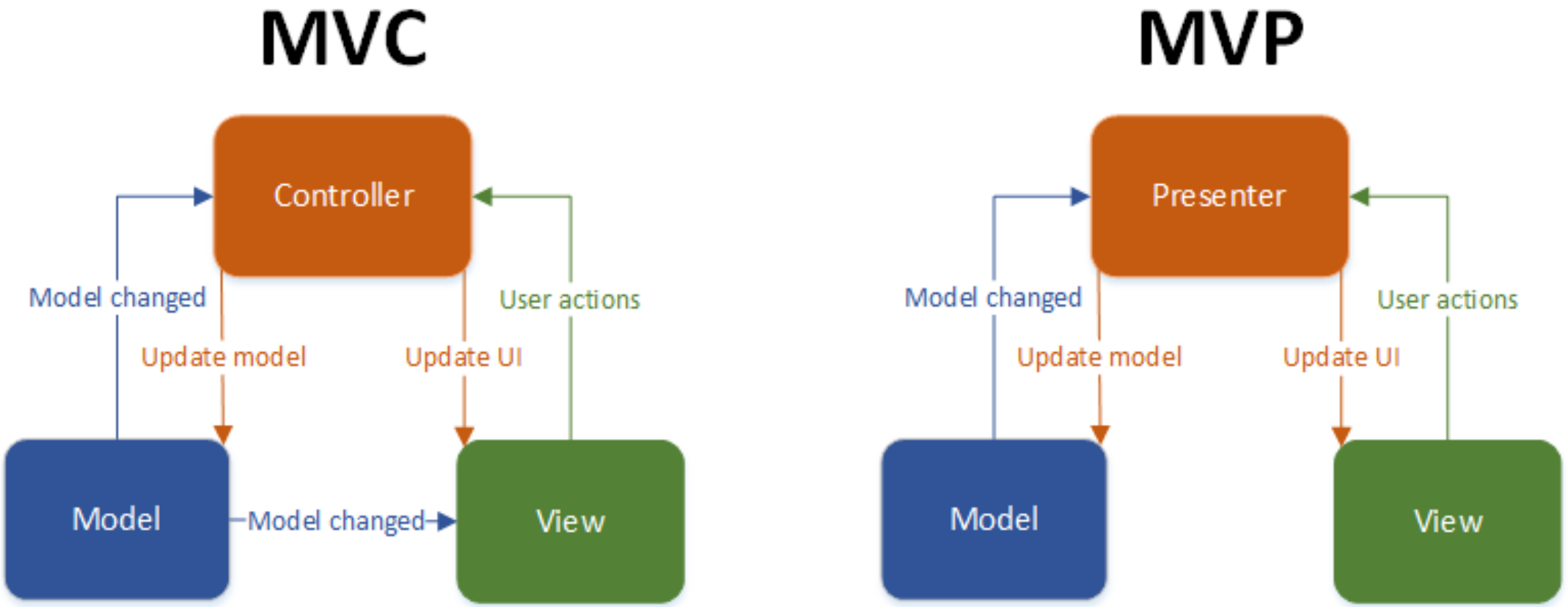
- Ao longo dos anos surgiram variações do MVC, algumas delas sendo:
  - MVP (Model–View–Presenter)
  - MVVM (Model–View–ViewModel)
  - HMVC (Hierarchical MVC)
  - MVA (Model–View–Adapter)
  - MVU (Model–View–Update)

- Cada variação nasceu para resolver limitações do MVC clássico em cenários específicos (desktop, web, mobile, arquiteturas reativas).

MVP

- **Model (Modelo)**
  - Igual ao MVC: representa dados e regras de negócio.
  - Não sabe nada sobre a interface.
- **View (Visão)**
  - Responsável apenas por **exibir informações** e **encaminhar interações**.
  - É “burra”: não contém lógica, apenas delega tudo para o Presenter.
- **Presenter**
  - Substitui o papel do Controller.
  - Atua como um **intermediário inteligente**: recebe inputs da View, consulta/atualiza o Model, e retorna dados formatados para a View.
  - Contém a lógica de **como os dados devem ser exibidos**.

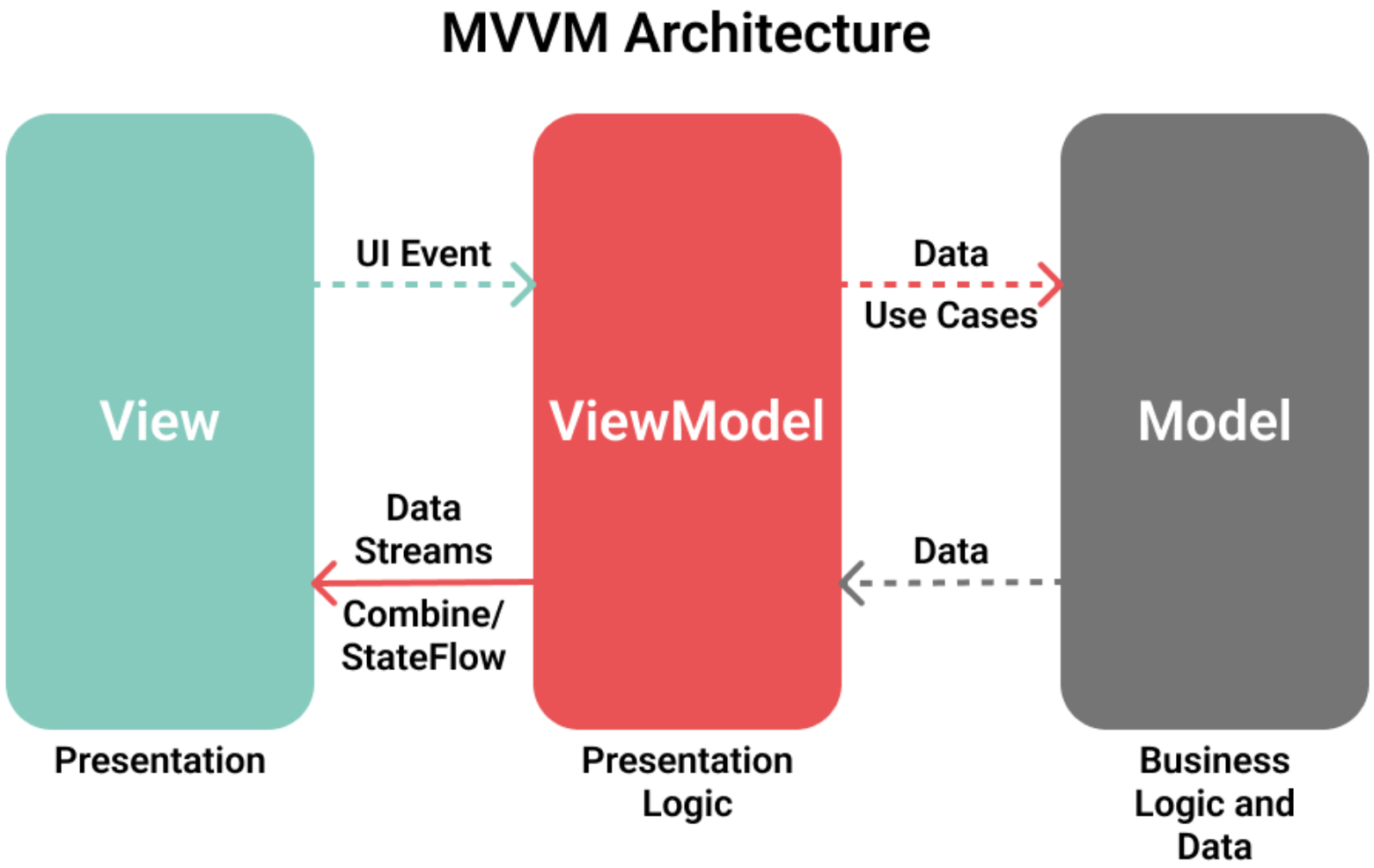
MVP


MVC
Controller
Model changed
Update model
Update UI
User actions
Model
Model changed
View
MVP
Presenter
Model changed
Update model
Update UI
User actions
Model
View

MVVM

- **Model (Modelo)**
  - Igual às outras variações: representa dados e regras de negócio.
  - Não tem conhecimento da interface.
- **View (Visão)**
  - Responsável apenas por **renderizar a interface** (botões, listas, gráficos, formulários).
  - Usa **data binding** para se conectar ao ViewModel.
  - Não precisa chamar métodos diretamente nem formatar dados.
- **ViewModel**
  - Intermediário entre a View e o Model.
  - Expõe **propriedades observáveis** (ex: Saldo, NomeCliente) e **comandos** (ex: DepositarCommand).
  - Contém a lógica de **apresentação**, mas não depende da tecnologia de UI.
  - Quando o Model muda, o ViewModel notifica a View, que se atualiza automaticamente.

MVVM


MVVM Architecture
View
ViewModel
Model
UI Event
Data
Use Cases
Data
Streams
Combine/
StateFlow
Data
Presentation
Presentation
Logic
Business
Logic and
Data

MVVM – Fluxo de Iteração

1. Usuário interage com a **View** (ex: digita num campo, clica em botão).
2. A ação está **vinculada (binded)** a um comando ou propriedade do **ViewModel**.
3. O ViewModel **atualiza o Model** ou reage à ação.
4. O Model muda → notifica o ViewModel → **a View é atualizada automaticamente** pelo binding.