Arquitetura de Sistemas

TIPOS E CONCEITOS

Conceito

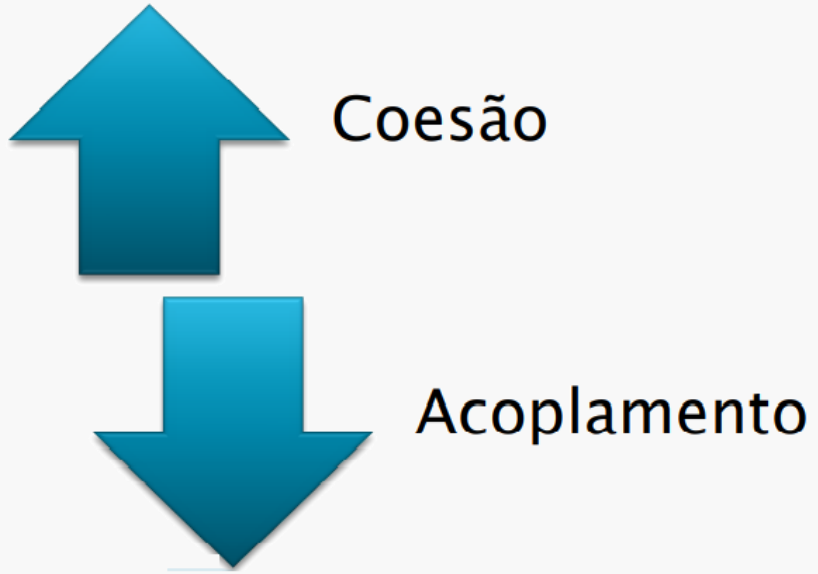
- Arquitetura de software é a organização ou a estrutura dos componentes significativos do sistema que interagem por meio de interfaces
- É composta de:
  - Componentes de software
  - Suas propriedades visíveis externamente
  - O relacionamento entre os componentes
- Forma a espinha dorsal para se construir softwares efetivos

Porque é importante?

- **Comunicação entre os *stakeholders***
  - A arquitetura representa uma abstração que pode ser entendida por todas as partes interessadas
  - Serve como base para entendimento, comunicação, negociação e consenso
- **Decisões de projeto tempestivas**
  - A arquitetura representa o ponto mais precoce onde as decisões de projeto podem ser analisadas
- **Abstração "transferível" de um sistema**
  - É possível reutilizar a aplicação de uma arquitetura ao longo de vários sistemas diferentes, mas que exibam características semelhantes

O que é uma boa arquitetura?

- Uma boa arquitetura de software deve ter os seus componentes projetados com **baixo acoplamento e alta coesão**



Acoplamento

- É o grau de dependência de um determinado módulo do programa em relação a outros módulos
- O acoplamento forte entre classes significa que elas precisam conhecer detalhes internos umas das outras
- Quanto menos acoplamento (interconexões entre classes) melhor!

Desvantagens do forte Acoplamento

- Mudanças em um módulo causam um efeito em cascata de mudanças em outros módulos
- A construção de um módulo se torna mais complicada devido à inter-dependência com outros módulos
- O reuso é prejudicado
- Os testes tornam-se mais difíceis de ser realizados

Exemplo

```
public class Pedido {
    public String produto;
    public int quantidade;
    public double preço;
    ...
}
```

```
public class Venda {
    ...
    public double calcularValor(Pedido pedido) {
        double valor = pedido.preço * pedido.quantidade;
    }
}
```

Código fortemente acoplado

```
public class Venda {
    ...
    public double calcularValor(Pedido pedido) {
        double valor = pedido.calcularTotal();
    }
}
```

Código fracamente acoplado

Coesão

- É a medida do quão fortemente relacionadas são as responsabilidades de um módulo
- Queremos ter classes
  - Com a menor complexidade possível
  - Com responsabilidades claramente definidas
  - Que não executam um grande volume de trabalho
- Queremos ter a máxima coesão possível

Vantagens da alta Coesão

- Módulos de sistemas coesos são mais simples de se entender
- A manutenção do sistema torna-se mais fácil, pois as mudanças são isoladas apenas ao módulo que interessa
- A capacidade de reuso aumenta

Exemplo

Código pouco coeso

```
public class Programa {

    public void desenharTela() {
        //implementação
    }

    public class reservarProduto() {
        //implementação
    }

    public class gravarNoBD() {
        // implementação
    }
```

Código de Apresentação

Código de Negócio

Código de Acesso a Dados

Arquitetura em Camadas

- Uma forma de organizar a arquitetura é através de camadas de software
  - Cada camada provê um conjunto de funcionalidades em determinado nível de abstração
  - Tipicamente, uma camada de mais alta abstração depende de uma camada de mais baixa abstração, e não o contrário
  - Uma mudança em determinada camada, desde que seja mantida sua interface, não afeta as outras camadas

Arquitetura em Camadas

Vantagens

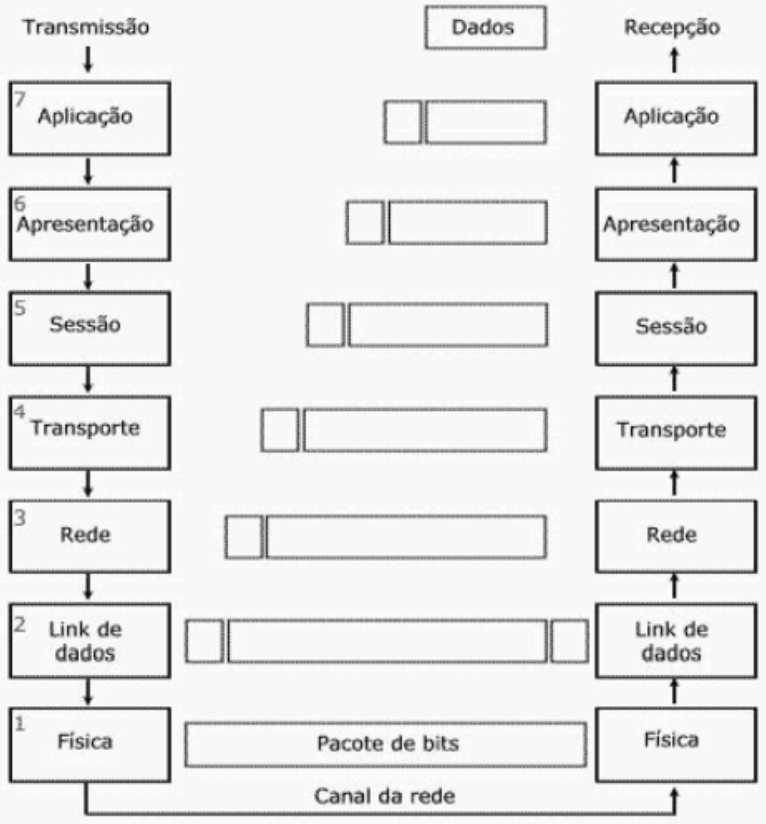
- Separação de responsabilidades
- Decomposição de complexidade
- Encapsulamento de implementação
- Maior reuso e extensibilidade

Desvantagens

- Podem penalizar a performance do sistema
- Aumento do esforço e complexidade de desenvolvimento

Arquitetura em Camadas

Exemplo:
Modelo OSI



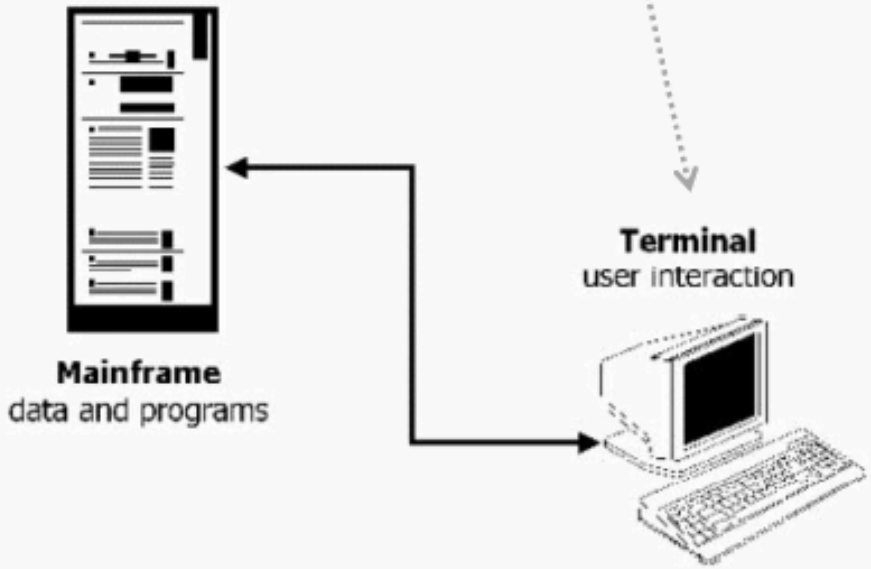
Evolução das arquiteturas em Camadas

- **Arquitetura Monolítica**
  - Programa e dados armazenados em uma única grande máquina – não havia camadas
  - Acesso através de terminais "burros"
- **Arquitetura em duas camadas (*two-tier*)**
  - Década de 80: surgem os PC's baratos
  - Aplicação rodava na máquina cliente que interagia com um SGBD (servidor de dados)
  - "*Fat client*" continha toda a lógica de apresentação, negócio e acesso a dados

Arquitetura Monolítica

Máquina de grande porte

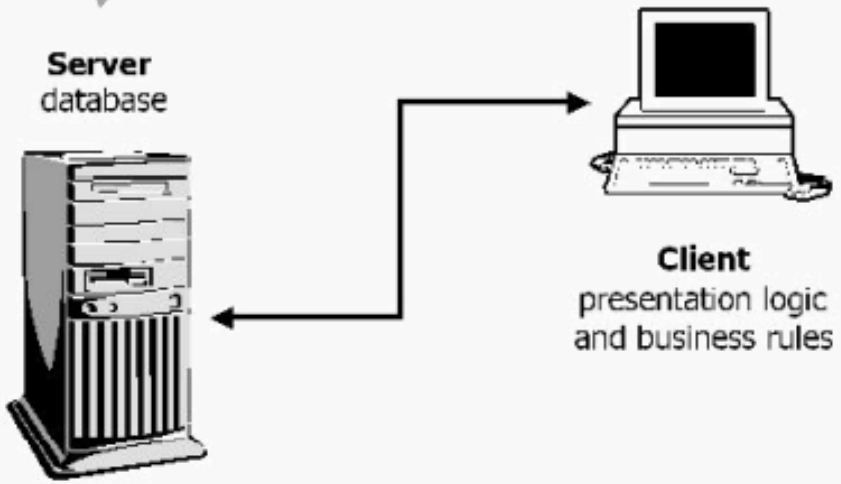
Terminal “burro”



Two-tier Architecture

SGBD

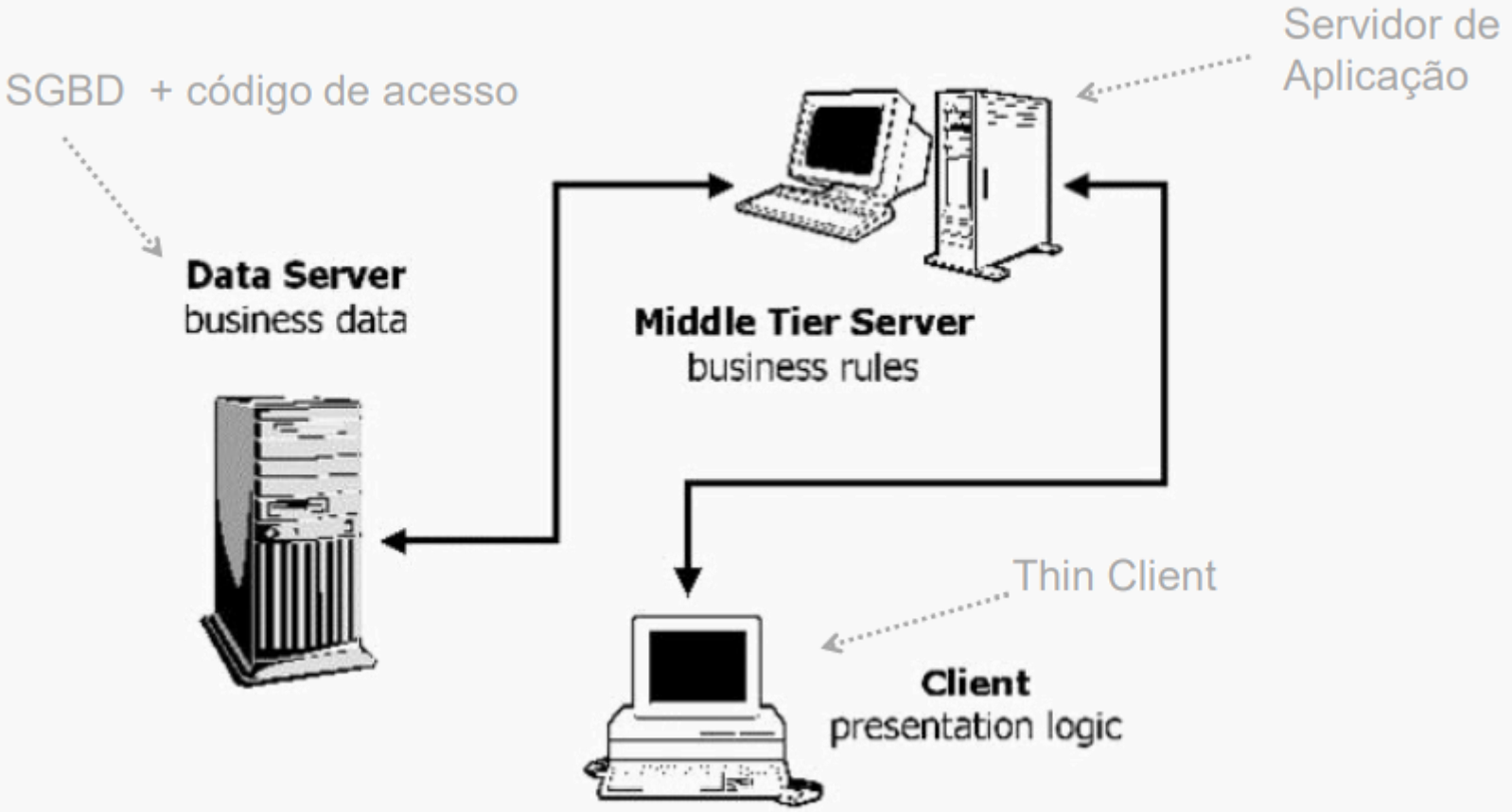
Fat Client



Arquiteturas em três camadas

- Para minimizar o impacto de mudanças nas aplicações, decidiu-se separar a camada de negócio da camada de interface gráfica, gerando três camadas:
  - **Camada de Apresentação**
  - **Camada da Lógica do Negócio**
  - **Camada de Acesso a Dados**
- Arquitetura conhecida como *Three-tier Architecture*

Arquiteturas em três camadas



Arquiteturas em três camadas

Vantagens

- É mais fácil de modificar ou substituir qualquer camada sem afetar as outras
- Separar a lógica de aplicação da lógica de acesso a dados melhora o balanceamento de carga
- É mais fácil assegurar políticas de segurança na camada do servidor sem interferir nos clientes

Arquiteturas em três camadas

Desvantagens

- Quanto mais camadas houver na arquitetura, maior é a tendência da performance diminuir
- O rastreamento de ponta-a-ponta em sistemas complexos com muitas camadas é uma tarefa complicada
- Requer um maior esforço de desenvolvimento

Camada de Apresentação

- Contém o código para a apresentação da aplicação (entrada e saída de dados)
  - As classes de fronteira se localizam aqui
- A camada de apresentação é altamente depende de ambiente
  - Páginas WEB (HTML, JavaScript, CSS, JSP, Applet, etc.)
  - Aplicações desktop (Windows/Linux Applications, etc.)
  - Menus baseados em texto (sistemas legados, aplicações móveis, etc.)

Camada da Lógica do Negócio

- Coordena a aplicação, processa comandos, toma decisões lógicas, faz avaliações e implementa as regras de negócio
- É **inerente** ao domínio (negócio) da aplicação
- Vários protocolos podem ser utilizados para ligar esta camada às outras duas
  - Sockets, HTTP, TCP/IP, etc. (Apresentação)
  - JDBC, LDAP, ODBC, etc. (Dados)

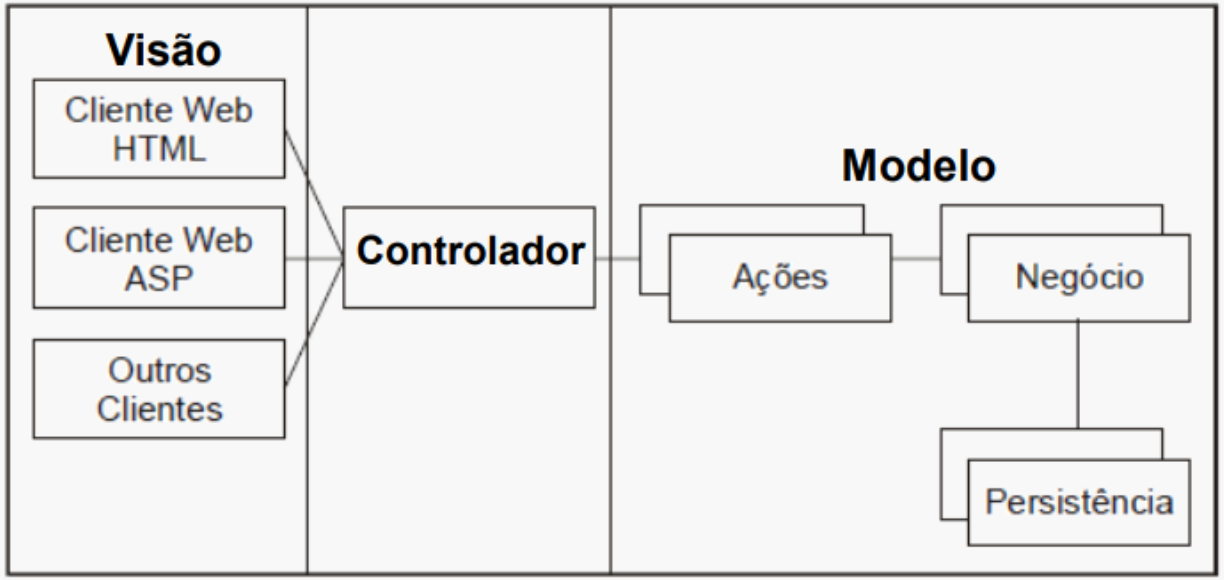
Camada de Acesso a Dados

- Contém o código responsável por armazenar e recuperar dados de uma base de dados ou sistema de arquivos
- Normalmente há uma sub-camada (interface) dentro desta camada que abstrai o mecanismo de persistência
  - O famoso padrão DAO (*Data Access Object*) é utilizado aqui.

Exemplo – camadas e seus protocolos/tecnologias

Arquitetura MVC(Model-View-Controller)

- Principal padrão de arquitetura em três camadas utilizado no mercado



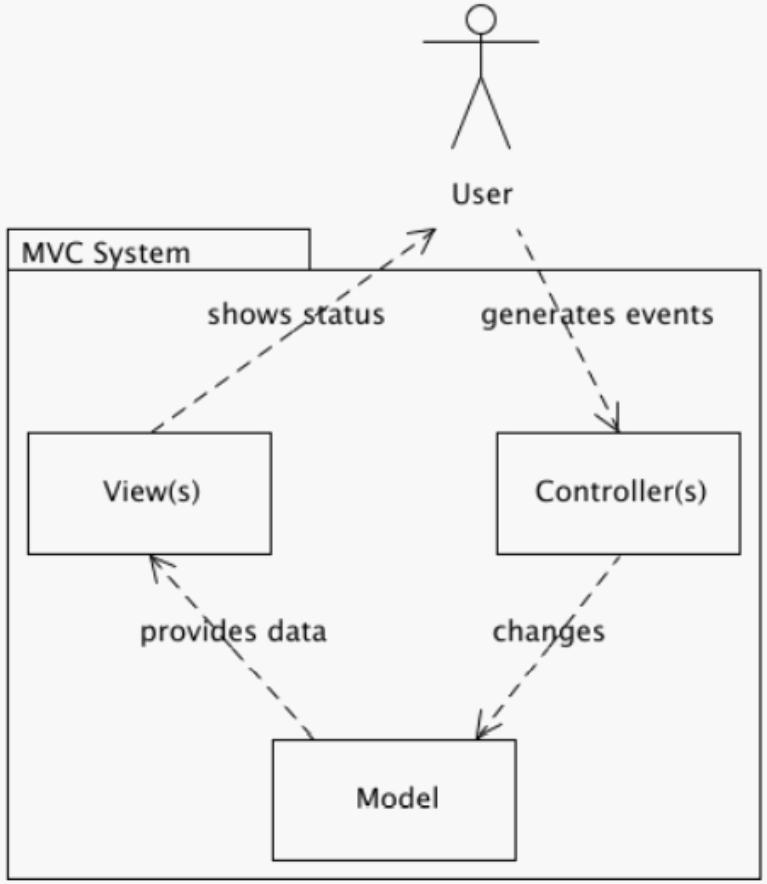
Camada de Visão

- É a camada de interface com o usuário
- Responsável por receber a entrada de dados e apresentar os resultados
- Não está preocupada em como ou onde a informação foi obtida, apenas exibe a informação
- Inclui elementos de exibição no cliente
  - HTML, XML, ASP, Applets, etc.
- Pode requerer dados diretamente da camada de Modelo

Camada de Modelo

- Responsável por modelar os dados e o comportamento por trás das regras de negócio
- Se preocupa com o armazenamento, manipulação e geração dos dados
- Objetos do Modelo são normalmente reusáveis, distribuídos, persistentes e portáveis para várias plataformas

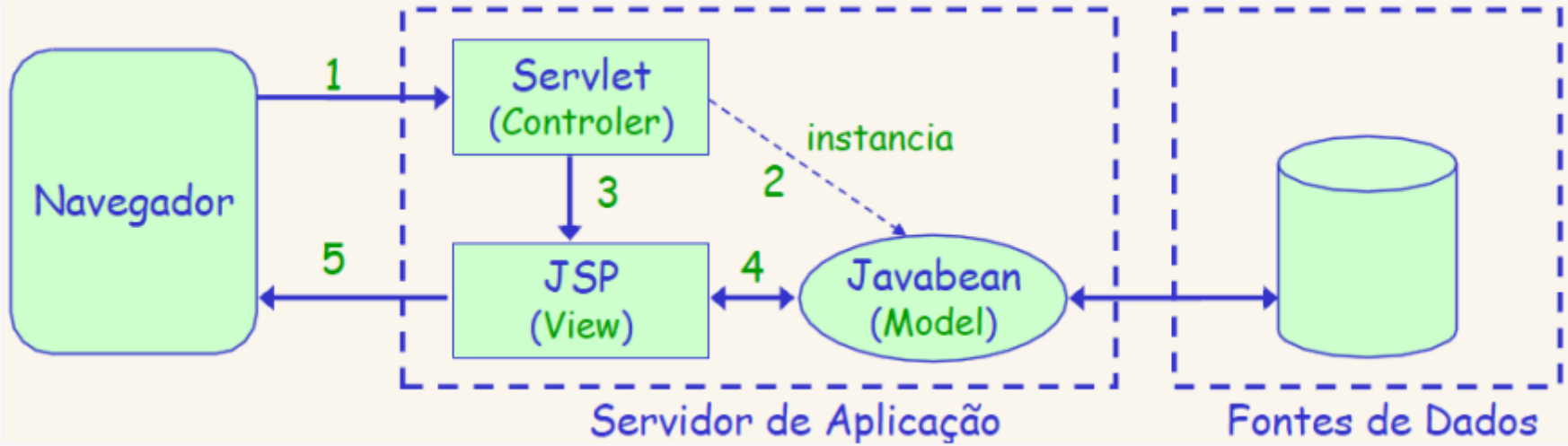
Interações entre as camadas MVC



MVC versus Three-tier Arquitecture

- A arquitetura em três camadas “pura” é linear – toda comunicação deve passar pela camada intermediária
- A arquitetura MVC é triangular – nem toda comunicação passa pelo Controlador
  - A Visão despacha atualizações para o Controlador
  - O controlador atualiza o modelo
  - **A Visão é atualizada diretamente pelo Modelo**

Arquitetura MVC na WEB(Model 2)

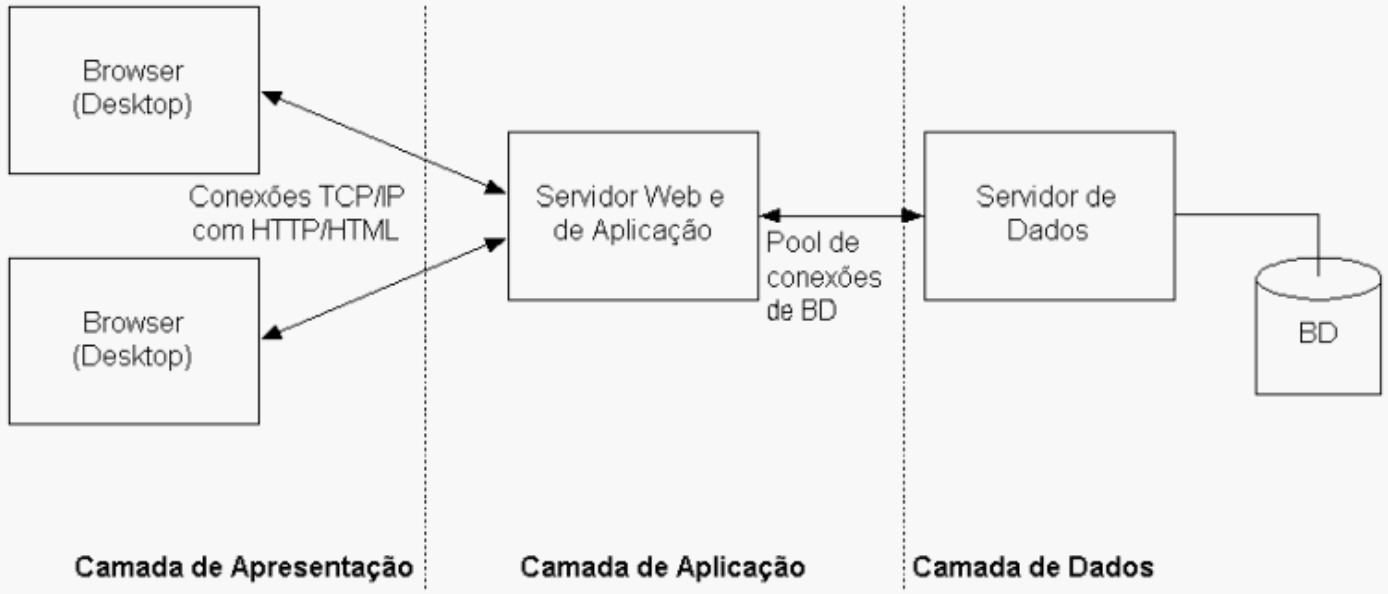


Arquitetura WEB

- Com o advento da WEB, o *browser* passou a ser utilizado como cliente universal
  - Evitamos instalar qualquer software em desktops, facilitando a manutenção
- O número e nome das camadas variam
  - No mínimo, costuma-se ter três camadas (pequenos volumes)
  - Mas pode haver "N" camadas, dependendo da necessidade (*N-tier architecture*)

Arquitetura WEB(3 camadas)

Para projetos mais simples podemos ter as camadas Web e Aplicação no mesmo local



Arquitetura WEB(n camadas)

Mas podemos ter mais camadas (flexibilidade)