# ARQUITETURA DE SW

## Introdução

SECTRAS
Sistemas de Ensino em Ciências e Tecnologias

Análise e Desenvolvimento de Sistemas

Prof. Ms. Renato Leite
renato@sectras.edu.br

# OBJETIVOS

- O que é Arquitetura de Software?
- Evolução da Arquitetura de Software
- O papel do arquiteto de software

# O QUE É ARQUITETURA DE SW

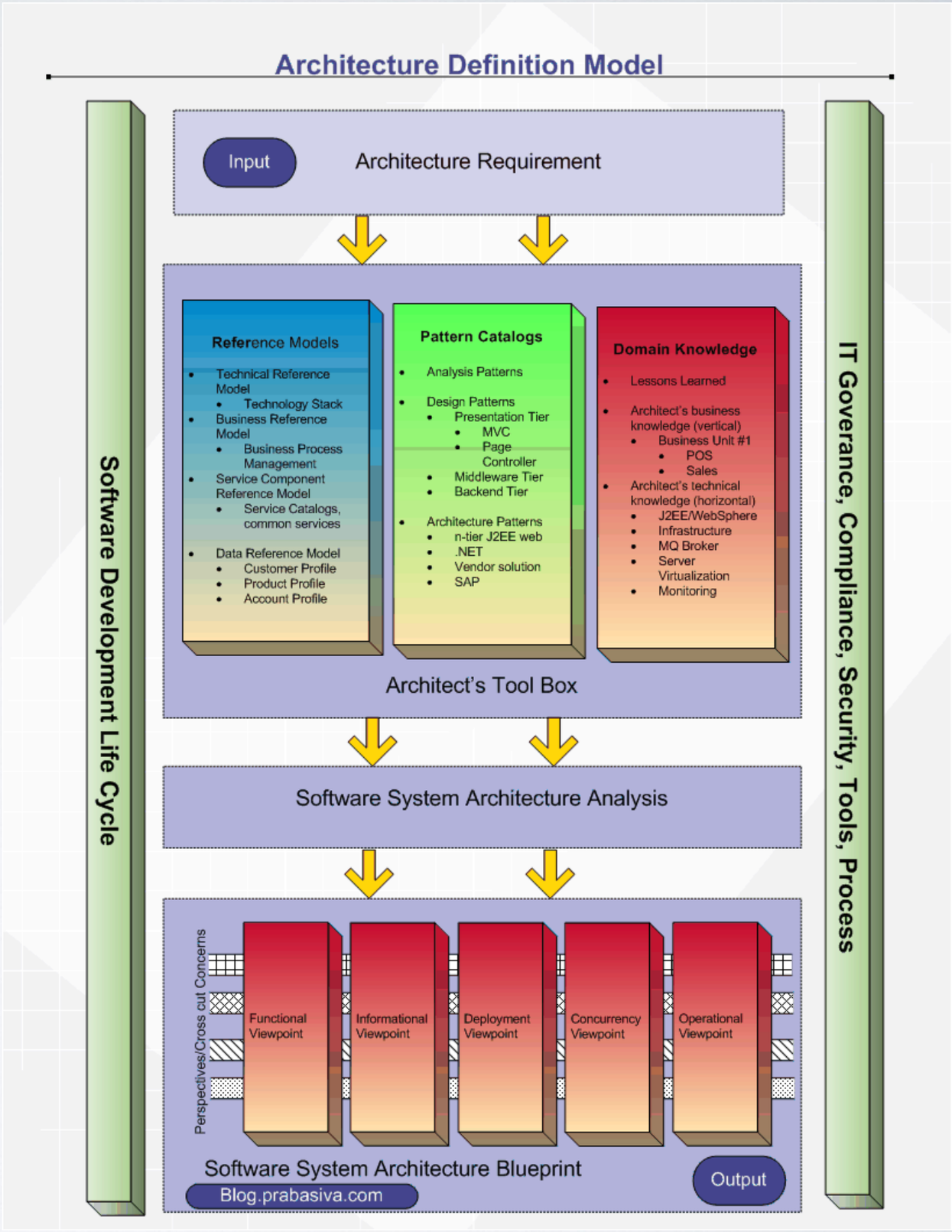
- Segundo Bass, Clements e Kazman (2012) em Software Architecture in Practice, "**a arquitetura de software é a estrutura do sistema, composta por componentes de software, suas interações e propriedades que determinam seu comportamento**".

- Já Sommerville (2019) em Software Engineering define arquitetura de software como "**o conjunto de decisões sobre a organização de um sistema que afetam diretamente sua confiabilidade, desempenho, segurança e capacidade de manutenção**".

- Função vs. Desempenho - é possível conciliar?

  - O que é um **soft**ware?

# O QUE É ARQUITETURA DE SW



# ENGENHARIA DE SOFTWARE

- Especificação
- Desenvolvimento
- Validação
- Manutenção

- **Em que fase está a arquitetura?**

# FUNÇÃO OU DESEMPENHO?

- Arquitetura de software define a estrutura interna de um sistema e sua organização.

- O uso de padrões arquiteturais simplifica a replicação e manutenção de software.

- A arquitetura do software evolui gradualmente conforme o sistema recebe novas funcionalidades.

- O software deve ser **flexível, extensível, portável e reutilizável** para garantir sua longevidade.

# O QUE DEFINE ARQUITETURA?

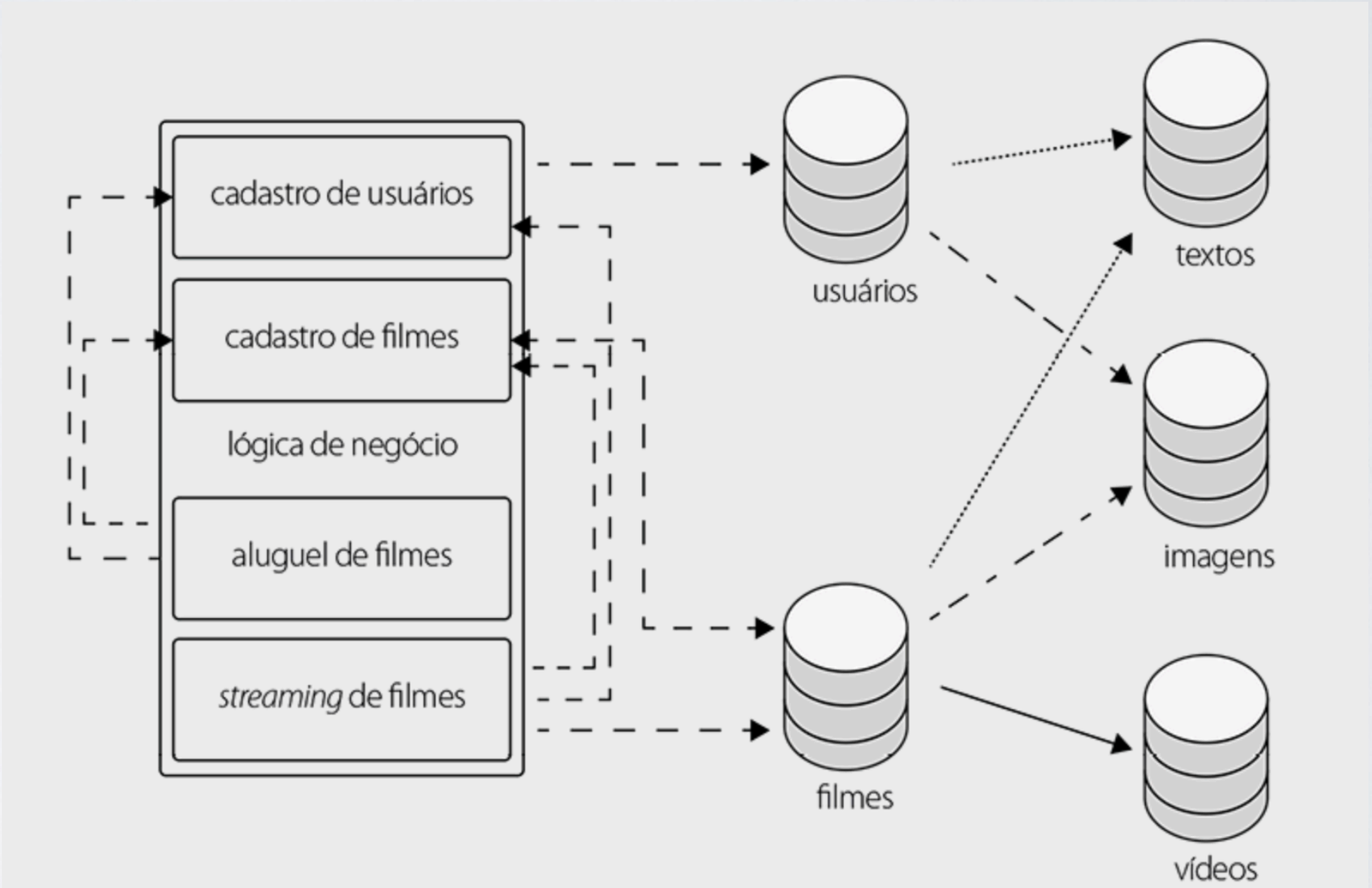
- Segundo Wolf (1992), "Arquitetura = (Elementos + Organização + Decisões)”

# O QUE DEFINE?

- Elementos:
  - de processamento – são eles que operam os dados, transformando suas entradas em saídas logicamente necessárias.
  - de dados – os dados que serão a matéria-prima a ser utilizada pelo sistema e processada.
  - de conexão – "amarram" os elementos na estrutura. Eles conectam qualquer forma de elemento entre si, de modo a formar um conjunto funcional operante.

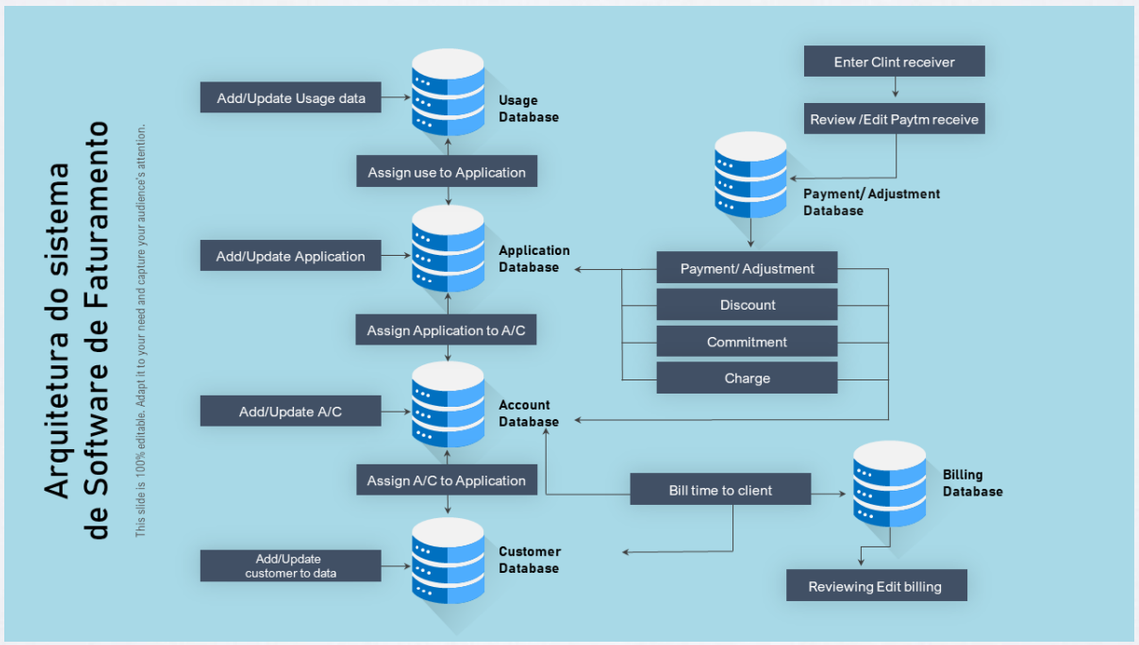
# O QUE DEFINE?

- Organização:
- Fluxo de informação

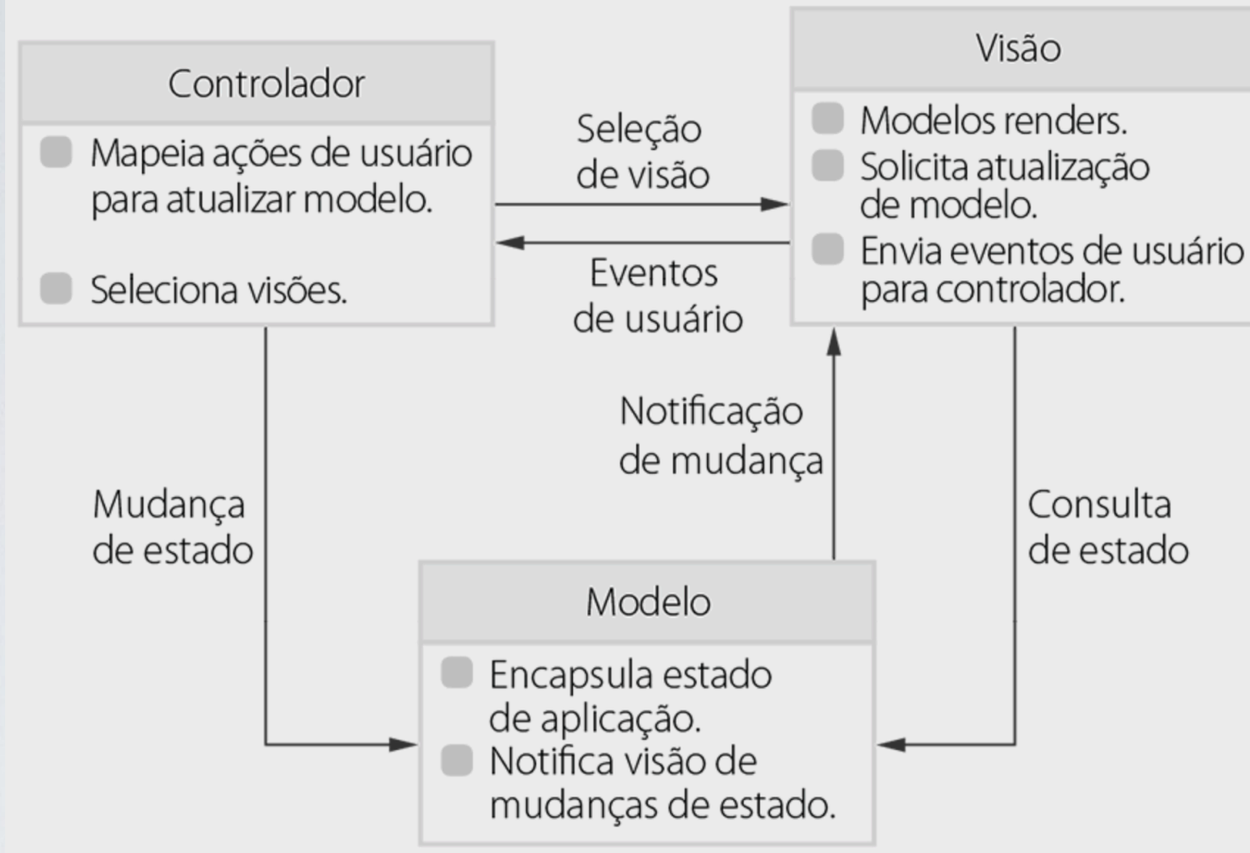


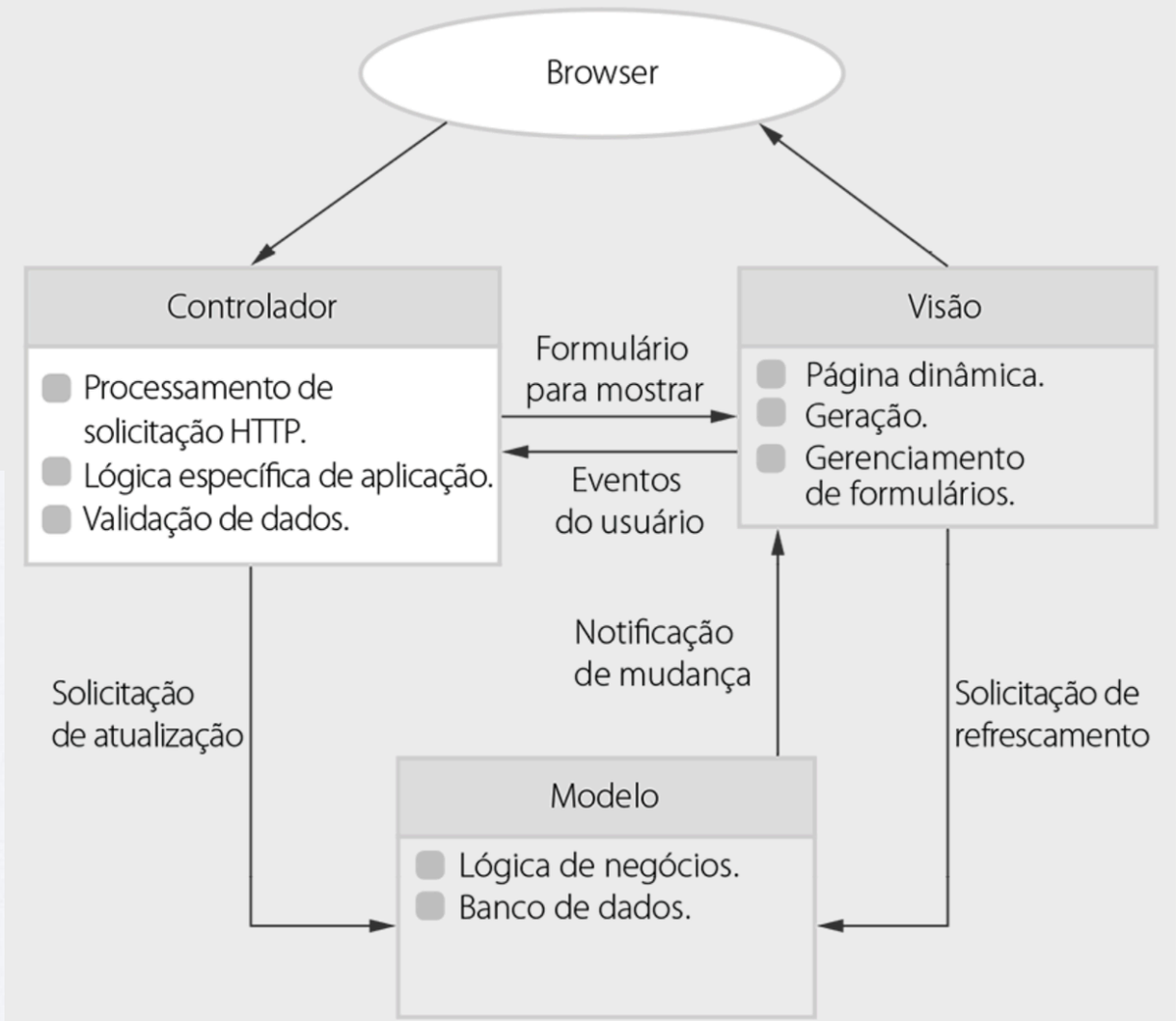
# PADRÕES ARQUITETURAIS

- São referências obtidas a partir de experiências bem sucedidas em sistemas já implementados;
- São diagramas e tabelas que apresentam de forma abstrata o que compõem uma arquitetura e suas descrições;



# PADRÃO MVC



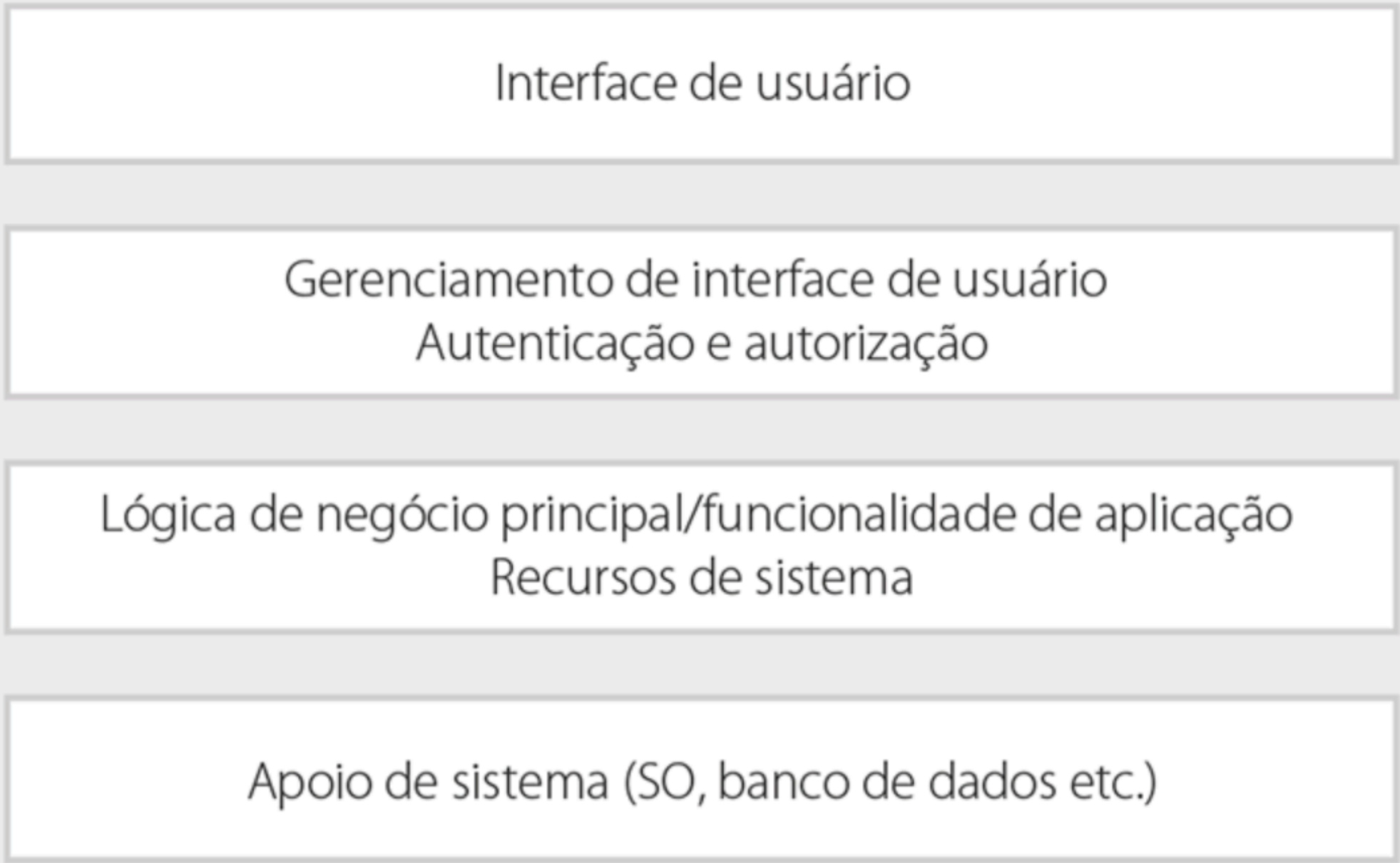

*Fonte:* Sommerville (2011, p. 110).

# PADRÃO MVC

- O componente modelo gerencia o sistema de dados e as operações associadas a esses dados.
- O componente visão define e gerencia como os dados são apresentados ao usuário.
- O componente controlador gerencia a interação do usuário (por exemplo, teclas, cliques do mouse etc.) e a passa para a visão e o modelo.

- É usado quando existem várias maneiras de se visualizar e interagir com dados, ou quando são desconhecidos os futuros requisitos de interação e apresentação de dados.
- Vantagens: Permite que os **dados sejam alterados de forma independente** de sua apresentação, e vice-versa. Apoia a **apresentação** dos mesmos dados de maneiras diferentes, com as alterações feitas em **uma representação aparecendo em todas elas**.
- Desvantagens: Quando o modelo de dados e as interações são simples, **pode envolver código adicional** e complexidade de código.

# PADRÃO LAYERED




*Fonte:* Sommerville (2011, p. 111).

# PADRÃO LAYERED

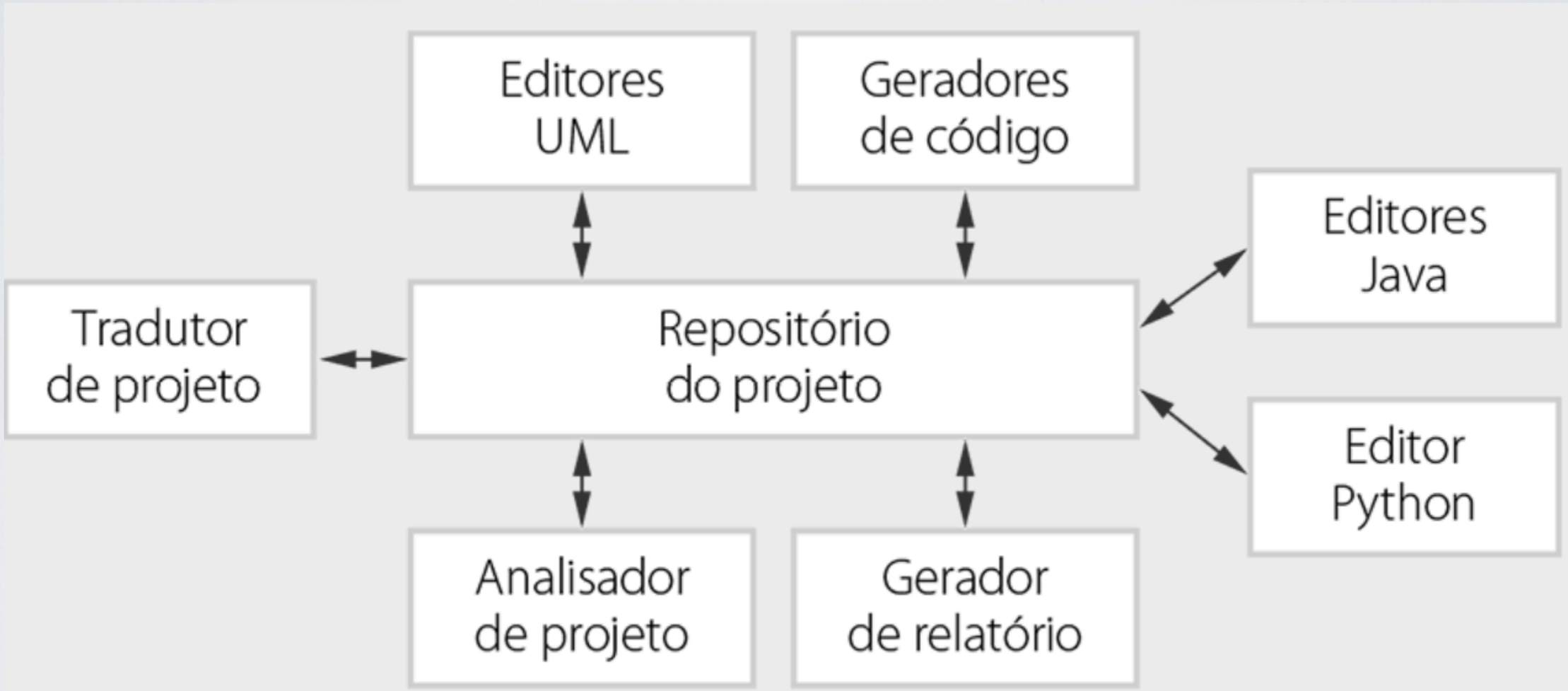
- Organiza o sistema em camadas com a funcionalidade relacionada associada a cada uma delas. Uma camada fornece serviços à camada acima dela; assim, os níveis mais baixos de camadas representam os principais serviços suscetíveis de serem usados em todo o sistema.

- É usado na construção de novos recursos em cima de sistemas existentes; quando o desenvolvimento está espalhado por várias equipes, com a responsabilidade de cada equipe em uma camada de funcionalidade; quando há um requisito de proteção multinível.

- Vantagens: Desde que a interface seja mantida, **permite a substituição** de camadas inteiras. **Recursos redundantes** (por exemplo, autenticação) podem ser fornecidos em cada camada para aumentar a confiança do sistema.

- Desvantagens: Na prática, costuma ser **difícil** proporcionar uma clara **separação** entre as camadas, e uma de alto nível pode ter de interagir diretamente com camadas de baixo nível, em vez da imediatamente abaixo dela. O **desempenho** pode ser um problema por causa dos múltiplos níveis de interpretação de uma solicitação de serviço, uma vez que são processados em cada camada.

# PADRÃO REPOSITORY




*Fonte:* Sommerville (2011, p. 112).

# PADRÃO REPOSITORY

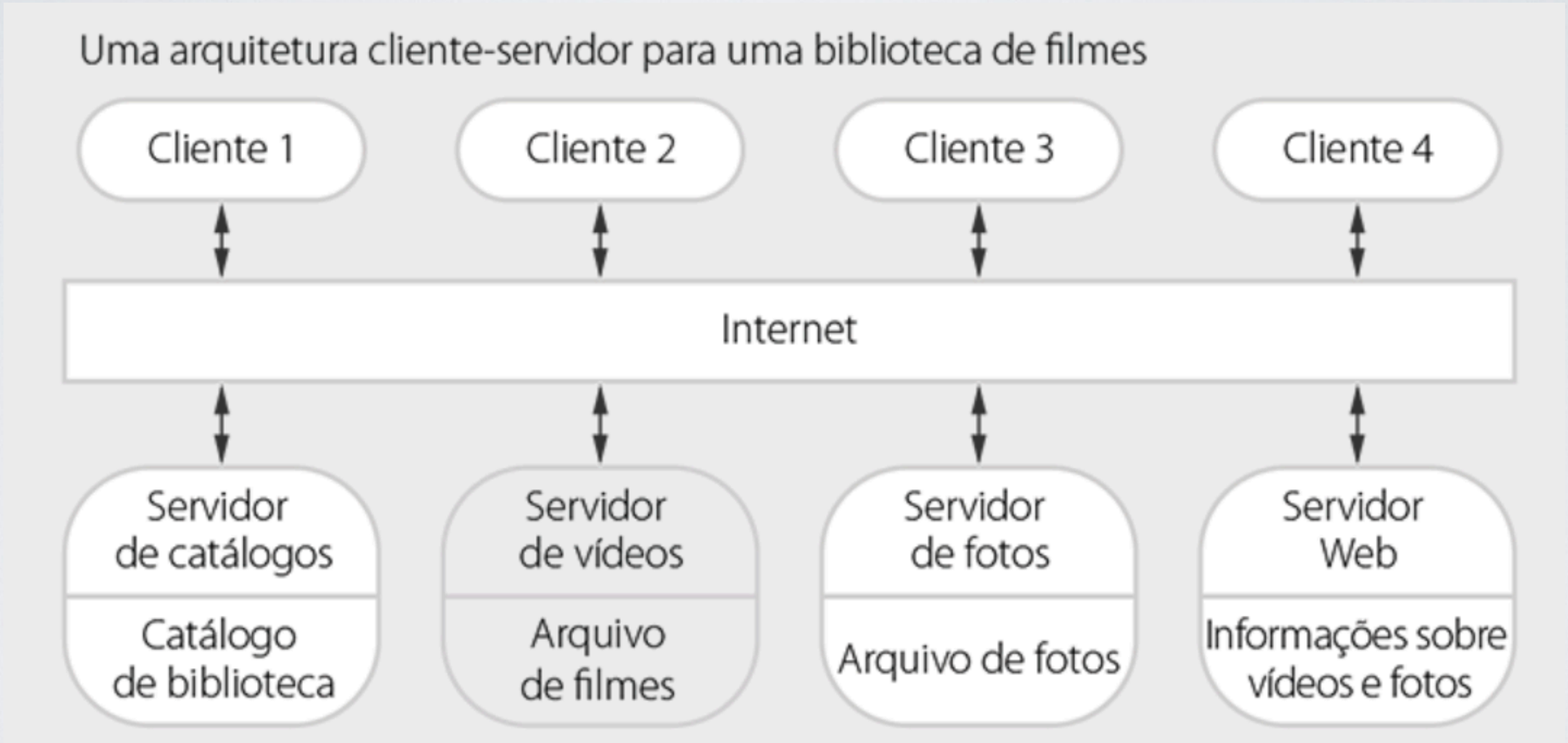
- Todos os dados em um sistema são gerenciados em um repositório central, acessível a todos os componentes do sistema. Os componentes não interagem diretamente, apenas por meio do repositório.

- Você deve usar esse padrão quando tem um sistema no qual grandes volumes de informações são gerados e precisam ser armazenados por um longo tempo. Você também pode usá-lo em sistemas dirigidos a dados, nos quais a inclusão dos dados nos repositórios dispara uma ação ou ferramenta

- Vantagens: Os componentes podem ser independentes - eles não precisam saber da existência de outros componentes. As alterações feitas a um componente podem se propagar para os demais. Gestão centralizada do dado.

- Desvantagens: O repositório é um **ponto único de falha**, **distribuir** o repositório a partir de vários computadores **pode ser difícil**.

# PADRÃO CLIENT-SERVER



# PADRÃO CLIENT-SERVER

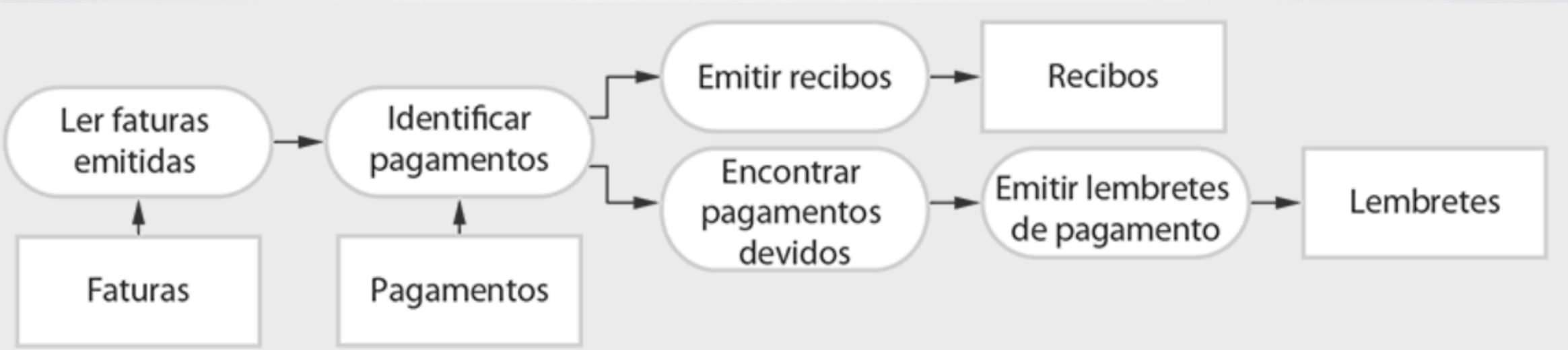
- Em uma arquitetura cliente-servidor, a funcionalidade do sistema está organizada em serviços – cada serviço é prestado por um servidor. Os clientes são os usuários desses serviços e acessam os servidores para fazer uso deles.

- É usado quando os dados em um banco de dados compartilhado precisam ser acessados a partir de uma série de locais. Como os servidores podem ser replicados, também pode ser usado quando a carga em um sistema é variável.

- Vantagem: os **servidores podem ser distribuídos** por meio de uma rede. A funcionalidade geral (por exemplo, um serviço de impressão) pode estar disponível para todos os clientes e não precisa ser implementada por todos os serviços.

- Desvantagem: **ponto único de falha** suscetível a ataques de negação de serviço ou falha do servidor. O **desempenho**, bem como o sistema, pode ser **imprevisível**, pois depende da rede. Pode haver problemas de gerenciamento se os servidores forem propriedade de diferentes organizações.

# PADRÃO DUCT-FILTER




*Fonte:* Sommerville (2011, p. 115).

# PADRÃO DUCT-FILTER

- O processamento dos dados em um sistema está organizado de modo que cada componente de processamento (filtro) seja discreto e realize um tipo de transformação de dados. Os dados fluem (como em um duto) de um componente a outro para processamento.

- Comumente, é usado em aplicações de processamento de dados (tanto as baseadas em lotes como as baseadas em transações) em que as entradas são processadas em etapas separadas para gerarem saídas relacionadas.

- Vantagens: O reuso da transformação é de fácil compreensão e suporte. Estilo de workflow corresponde à estrutura de muitos processos de negócios. Evolução por adição de transformações é simples. Pode ser implementado tanto como um sistema sequencial quanto concorrente.

- Desvantagens: O formato para transferência de dados tem de ser **acordado** entre as transformações de comunicação. Cada transformação deve analisar suas entradas e gerar as saídas para um formato acordado. Isso aumenta o overhead do sistema e pode significar a **impossibilidade** de **reuso** de transformações funcionais que adotam estruturas incompatíveis de dados.

# PAPEL DO ARQUITETO DE SOFTWARE

- Conforme Richards & Ford (2020) em Fundamentals of Software Architecture, os principais papéis do arquiteto incluem:

  - **Definir a estrutura do software**, determinando seus módulos e interações.

  - **Escolher os estilos** arquiteturais mais adequados para o projeto.

  - Avaliar trade-offs técnicos, como **custo vs. escalabilidade**.

  - **Garantir a qualidade do software**, estabelecendo boas práticas e padrões.

  - **Trabalhar junto aos times de desenvolvimento**, ajudando na implementação correta da arquitetura.

# BIBLIOGRAFIA


- GALLOTTI, G. M. A. Arquitetura de Software. 1ª Ed. São Paulo: Pearson, 2016. [acervo digital]

- JUNIOR, A. K. Computação em Nuvem. 1ª Ed. Curitiba: Contentus, 2020. [acervo digital]

- ZANETTI, M. Arquitetura de TI e modelos de negócios. 1ª Ed. Curitiba: Contentus, 2020. [acervo digital]

# DÚVIDAS?