Instituto Federal de Educação, Ciência e Tecnologia da Bahia - IFBA
Departamento de Ciência da Computação
Tecnólogo em Análise e Desenvolvimento de Sistemas

Projeto de arquitetura

André L. R. Madureira <andre.madureira@ifba.edu.br>
Doutorando em Ciência da Computação (UFBA)
Mestre em Ciência da Computação (UFBA)
Engenheiro da Computação (UFBA)

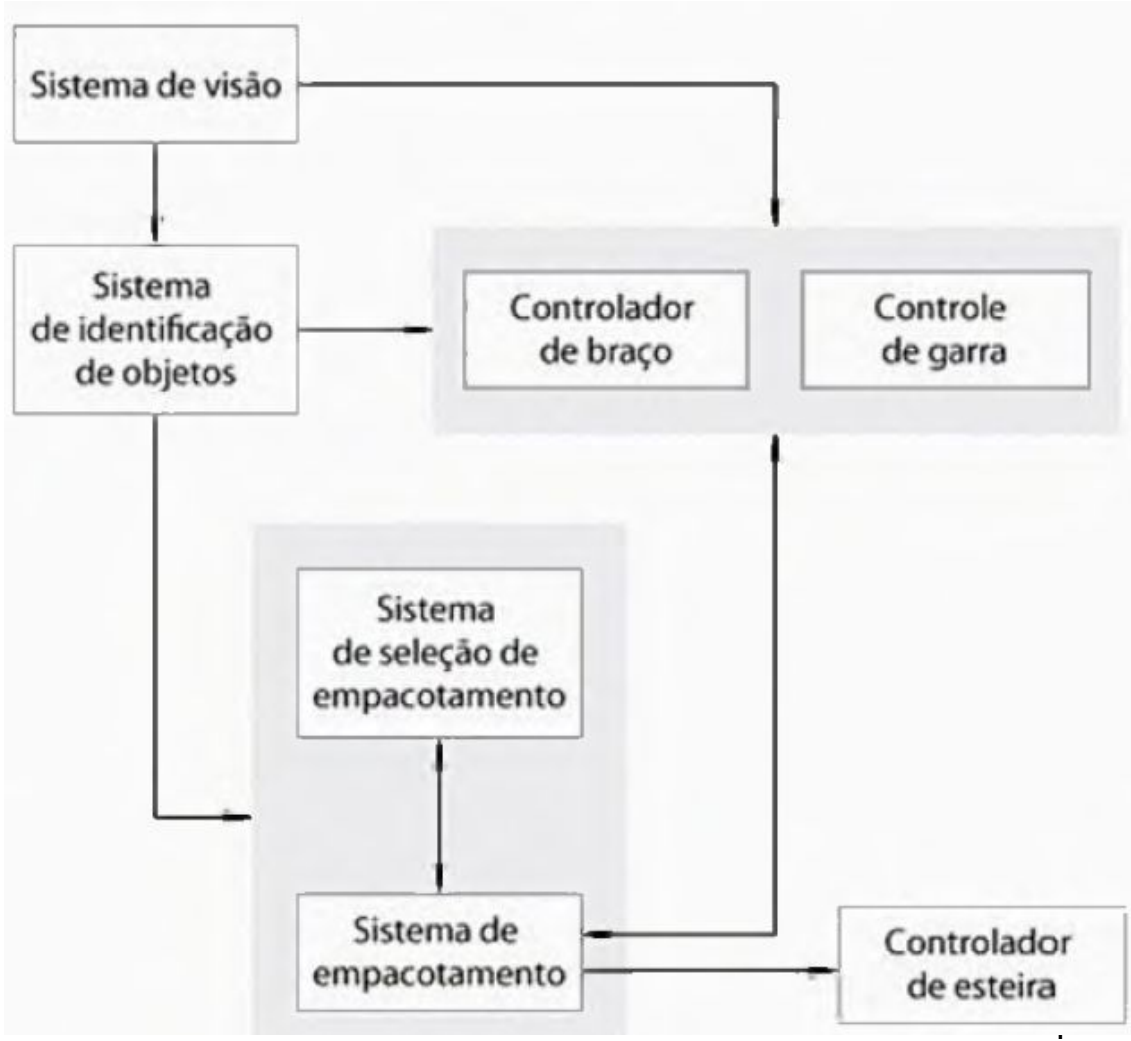
Projeto de arquitetura

- Primeira atividade do desenvolvimento do sistema
- Foco na organização e estrutura geral do sistema
  - **Objetivo**: satisfazer os requisitos funcionais e não funcionais
- **Resultado**: modelo de arquitetura, descrevendo a organização do sistema como um conjunto de componentes
  - **Importância**: a arquitetura afeta o desempenho, robustez, capacidade de distribuição e de manutenibilidade do sistema (BOSCH, 2000)

Modelagem da arquitetura

- A arquitetura de um sistema é modelada por meio de diagramas de blocos simples
  - Caixa representa um componente
  - Caixas dentro de caixas indicam que o componente foi decomposto em subcomponentes
  - As setas significam que os dados e/ou sinais de controle são passados de um componente a outro na direção das setas

Exemplo de modelo de arquitetura para sistema de empacotamento



Vantagens do Projeto de Arquitetura

- Segundo Bass et al. (2003), projetar e documentar explicitamente a arquitetura de um sistema traz vantagens:

Comunicação com *stakeholders*:
A arquitetura como uma apresentação de alto nível do sistema para os *stakeholders*

Análise de sistema:
A construção da arquitetura exige análise prévia do sistema, pois decisões de projeto de arquitetura afetam a implementação dos requisitos

Reúso em larga escala:
A arquitetura do sistema geralmente é a mesma para sistemas com requisitos semelhantes (apoio ao reuso de software em grande escala)

Documentação do sistema:
A arquitetura do sistema é um modelo completo do mesmo, que mostra os seus componentes, interfaces e conexões

Projeto de arquitetura através de decisões

- *“Existe uma arquitetura genérica que pode atuar como um modelo para o sistema que está sendo projetado?”*
- *“Que padrões ou estilos de arquitetura podem ser usados?”*
- *“Como os componentes estruturais do sistema serão decompostos em subcomponentes?”*
- *“Que estratégia será usada para controlar o funcionamento dos componentes do sistema?”*
- Outros aspectos de projeto e design relevantes . . .

Escolha da arquitetura de distribuição

- A escolha da arquitetura de distribuição é uma decisão importante que afeta o desempenho e a confiabilidade do sistema
  - **Para sistemas embutidos e sistemas para computadores pessoais, geralmente existe apenas um processador**
    - Logo, você não terá de projetar uma arquitetura distribuída
  - **No entanto, a maioria dos sistemas de grande porte são sistemas distribuídos**
    - O software executa em vários computadores diferentes

Escolha do padrão de arquitetura

- A arquitetura de um sistema de software pode se basear em um determinado **padrão ou estilo de arquitetura**
  - **Padrão de arquitetura**: é uma descrição abstrata de boas práticas experimentadas e testadas em diferentes sistemas e ambientes
    - Descreve uma organização de sistema bem-sucedida em sistemas anteriores
    - **Ex**: cliente-servidor, arquitetura em camadas

Escolha do padrão de arquitetura

- **Você deve conhecer os padrões de arquitetura comuns**
  - Saber ONDE e COMO eles podem ser usados
  - Quais são seus PONTOS FORTES E FRACOS

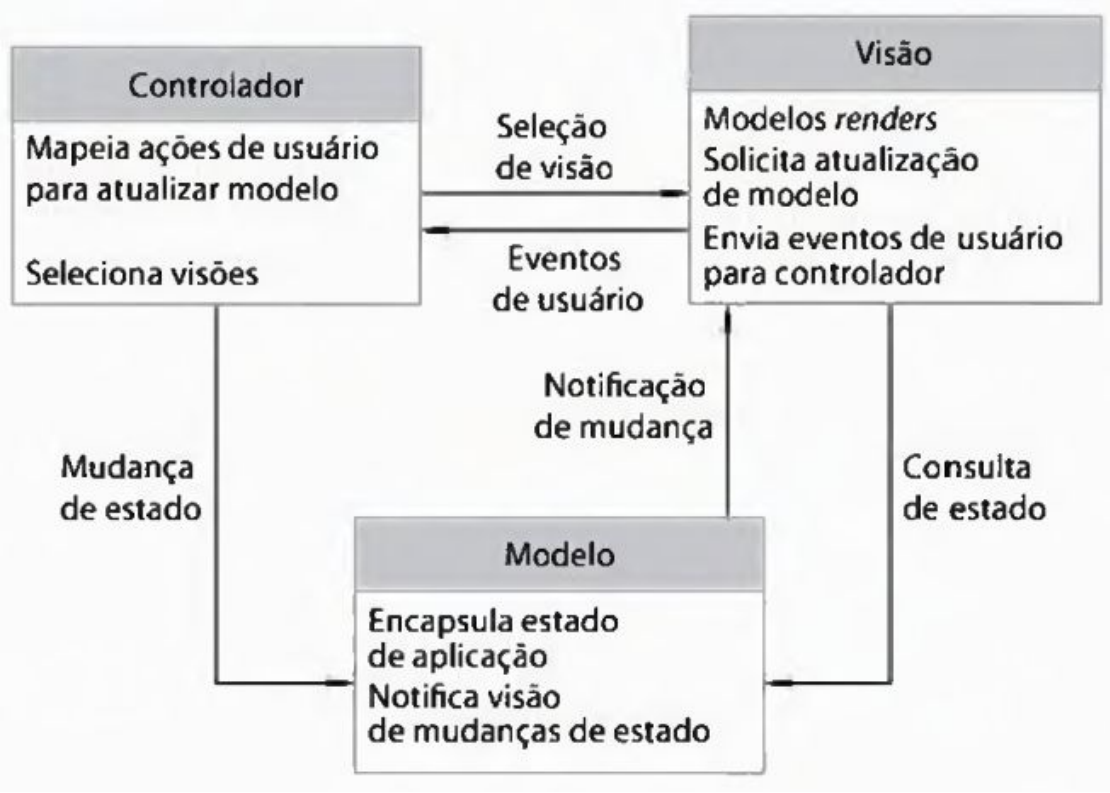
Padrões de arquitetura

- Os padrões mais utilizados são:
  - **MVC** (Model-View-Controller)
  - **Arquitetura em camadas**
  - **Arquitetura de repositório**
  - **Arquitetura cliente-servidor**
  - **Arquitetura de duto e filtro**

Padrão MVC (model-view-controller)

- Separa a apresentação, modelo de dados e lógica do sistema
- Arquitetura baseada em três componentes que interagem entre si:
  - **Model (modelo):** gerencia o sistema de dados e as operações associadas a esses dados
  - **View (visão)**: define como os dados são apresentados ao usuário e como ele interage com o sistema (e.g., GUI ou CLI).
  - **Controller (controlador):** define a lógica do sistema (mapeia as ações do usuário na ***view*** para atualizações do ***model***)

Padrão MVC (model-view-controller)



Padrão MVC (model-view-controller)

- **Quando usar este padrão?**
  - Existem várias maneiras de se visualizar e interagir com dados
  - São desconhecidos os futuros requisitos de interação e apresentação de dados

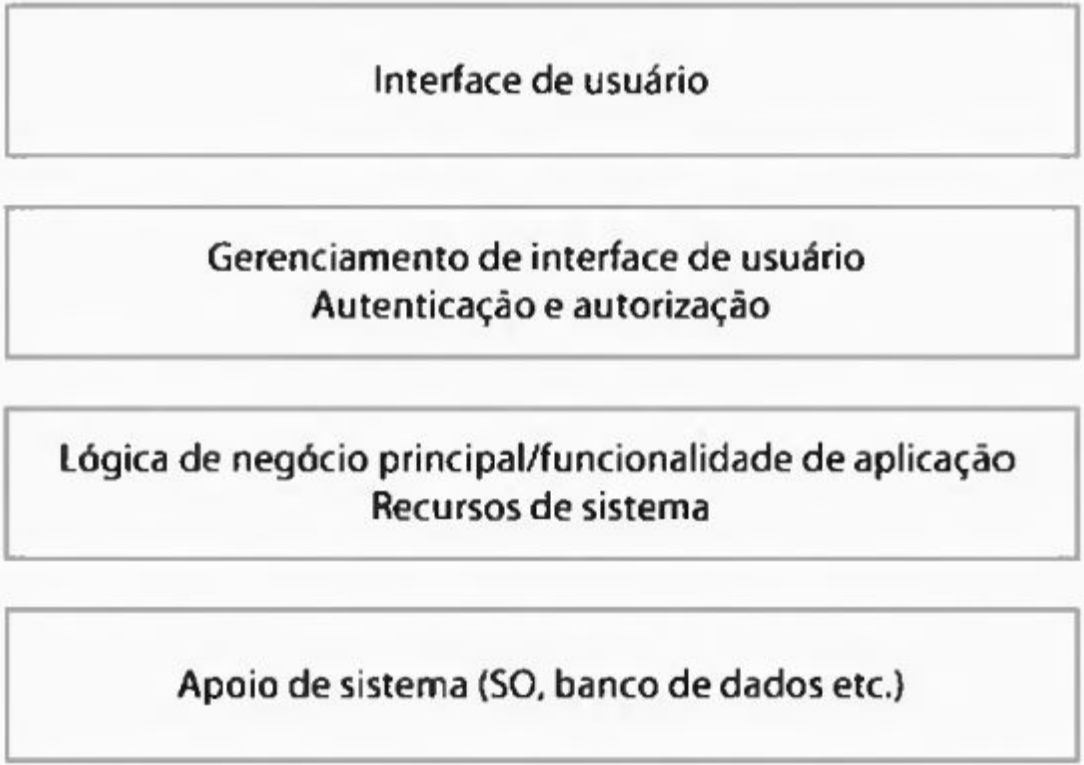
Padrão MVC (model-view-controller)

- **Vantagens**:
  - Permite a alteração dos dados de forma independente de sua representação
  - Alterações feitas em uma representação aparecem em todas as outras apresentações
- **Desvantagens**:
  - Código adicional necessário para criar cada *view*
  - Complexidade de código

Arquitetura em camadas

- A funcionalidade do sistema é organizada em camadas separadas
  - Uma camada fornece serviços à camada acima dela
  - Camadas mais baixas representam os principais serviços suscetíveis de serem usados em todo o sistema
- Uma camada pode ser substituída por outra equivalente, desde que a interface com as camadas adjacentes seja preservada
  - Mesmo quando a interface muda, apenas a camada adjacente é afetada

Arquitetura em camadas



Arquitetura em camadas

- **Quando usar este padrão?**
  - Construção de novos recursos em cima de sistemas existentes
  - Quando o desenvolvimento está espalhado por várias equipes
    - Cada equipe responsável por uma camada de funcionalidade
  - Quando há um requisito de proteção multinível

Arquitetura em camadas

- **Vantagens**:
  - Permite a substituição de camadas inteiras *(preservando interface)*
  - Recursos redundantes (ex: autenticação) podem ser fornecidos em cada camada para aumentar a confiança do sistema
- **Desvantagens**:
  - Dificuldade na separação em camadas
    - Comunicação intercamadas
  - Baixo desempenho (múltiplos níveis de interpretação de uma solicitação de serviço)

Arquitetura de repositório

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

Exemplo de arquitetura de repositório para uma IDE

Arquitetura de repositório

- **Quando usar este padrão?**
  - Compartilhar grandes quantidades de dado, que precisam ser armazenados por um longo tempo.
  - Você também pode usá-lo em **sistemas dirigidos a dados**
    - A inclusão dos dados no repositório dispara uma ação ou ferramenta

Arquitetura de repositório

- **Vantagens**:
  - Componentes independentes
    - Componentes não sabem da existência de outros componentes
  - Gerenciamento consistente de dados (*tudo está em um único local*)
- **Desvantagens**:
  - Repositório é um **ponto único de falha**
    - Problemas no repositório podem afetar todo o sistema
  - Ineficiências na comunicação através do repositório

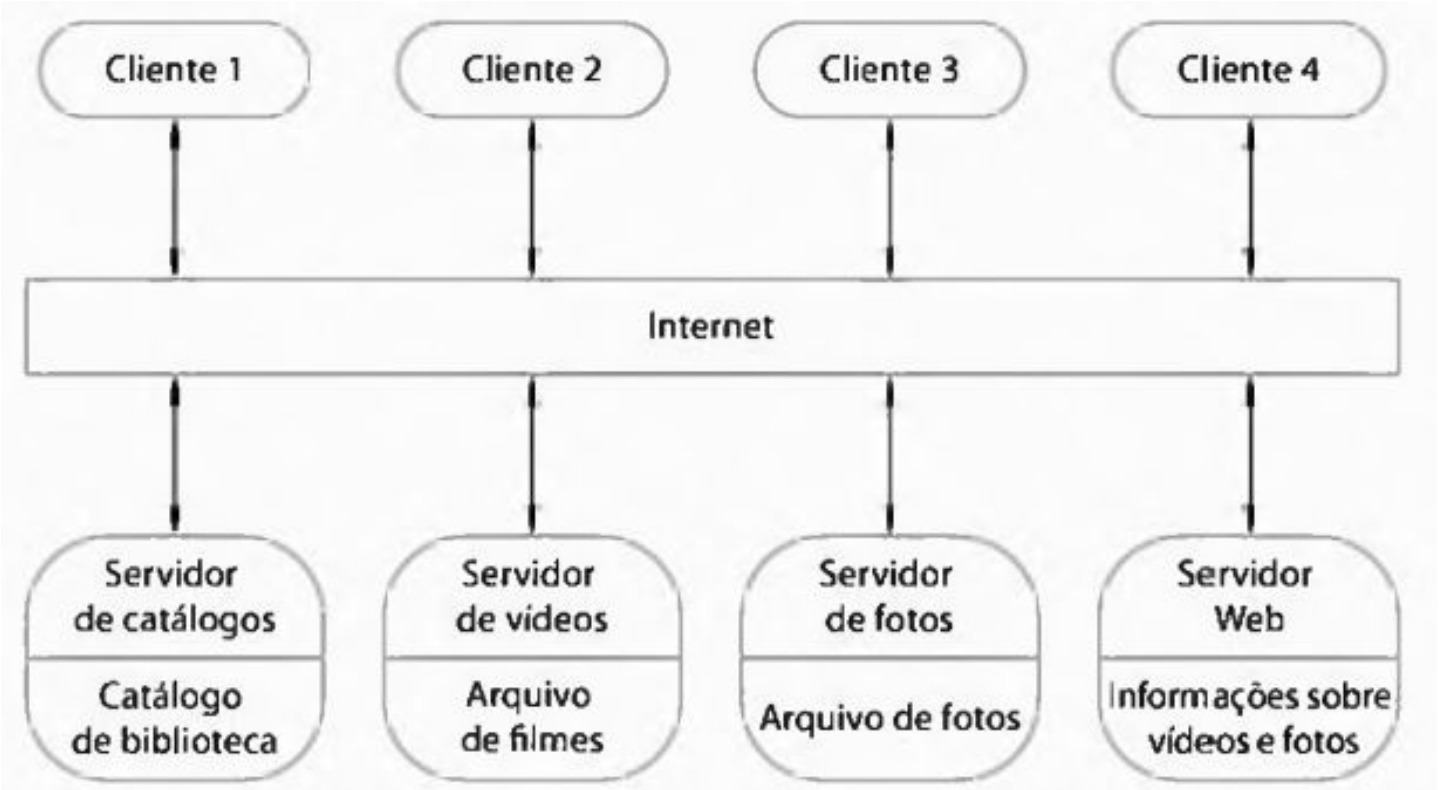
Arquitetura cliente-servidor

- Funcionalidade do sistema organizada em serviços
  - Cada serviço é prestado por um servidor
    - **Ex**: servidores de impressão, servidores de arquivos
- Os clientes são os usuários desses serviços
  - ***"Um cliente faz uma solicitação e espera pela resposta do servidor"***
  - Normalmente há várias instâncias de um programa cliente executando simultaneamente em computadores diferentes
- Cliente e servidores se comunicação através de uma rede

Arquitetura cliente-servidor

- Os servidores NÃO precisam conhecer
  - A identidade dos clientes
  - Quantos clientes estão acessando seus serviços
- Os clientes PODEM PRECISAR conhecer
  - Os nomes dos servidores disponíveis
  - Os serviços que eles fornecem
- A comunicação cliente-servidor é normalmente iniciada pelo cliente

Exemplo de Arquitetura cliente-servidor (*Serviço de Streaming Multimedia*)



Arquitetura cliente-servidor

- **Quando usar este padrão?**
  - Quando os dados em um banco de dados compartilhado precisam ser acessados a partir de uma série de locais
    - **Ex**: clientes espalhados em diferentes localizações
  - Quando a carga em um sistema é variável
    - Períodos com maior ou menor número de solicitações
    - Servidores podem ser replicados para distribuir a carga entre eles (***load balancing***)

Arquitetura cliente-servidor

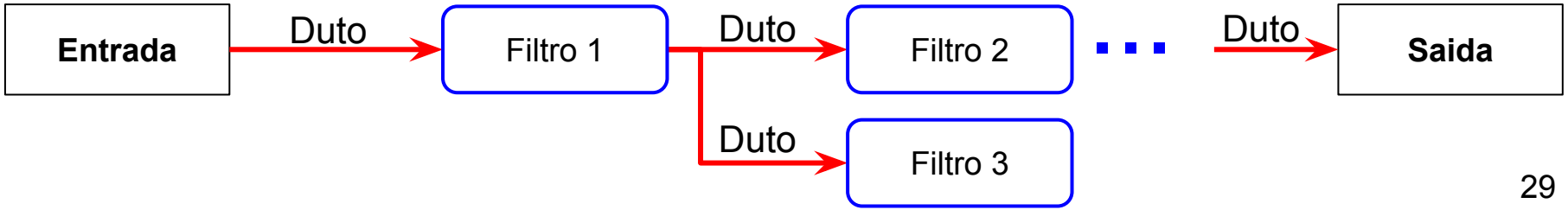
- **Vantagens**:
  - Servidores distribuídos através de uma rede (**arquitetura distribuída**)
  - Funcionalidade disponível para todos os clientes
  - Alta capacidade para atender solicitações (***load balancing***)
- **Desvantagens**:
  - Desempenho imprevisível pois depende da rede
  - Problemas de gerenciamento em servidores dentro de diferentes organizações
    - **Ex**: Google <-> Facebook

Arquitetura de duto e filtro

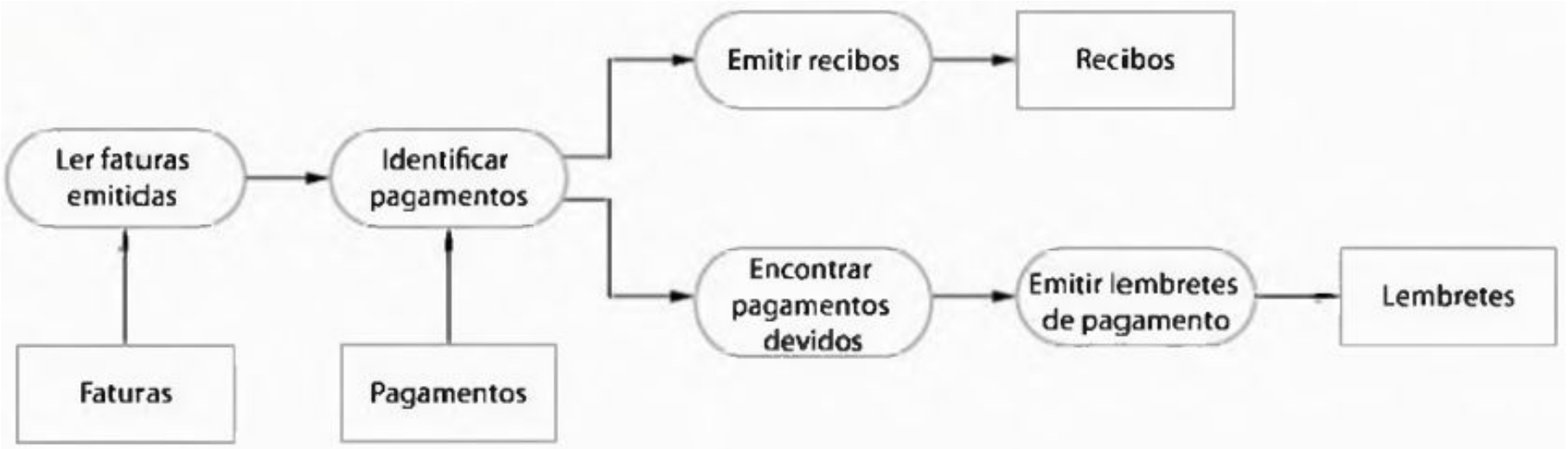
- Sistema baseado em transformações de dados (entrada -> saida)
  - Componentes de processamento discretos (**filtros**) transformam dados de entrada em saidas
- Cada componente implementa uma etapa de transformação
  - Os dados fluem de um componente para outro (como em **dutos**)
  - Componentes executam sequencialmente ou em paralelo

Arquitetura de duto e filtro

- Sistema baseado em transformações de dados (entrada -> saida)
  - Componentes de processamento discretos (**filtros**) transformam dados de entrada em saidas
- Cada componente implementa uma etapa de transformação
  - Os dados fluem de um componente para outro (como em **dutos**)
  - Componentes executam sequencialmente ou em paralelo



Exemplo de Arquitetura de duto e filtro (*Sistema de Ponto De Venda – PDV*)



Arquitetura de duto e filtro

- **Quando usar este padrão?**
  - Usado em aplicações de processamento de dados em que as entradas geram saídas relacionadas, através do processamento em etapas separadas
    - **Ex**: aplicações baseadas em lotes, ou em transações

Arquitetura de duto e filtro

- **Vantagens**:
  - Reuso de componentes (transformações)
  - Facilidade para manutenção e evolução do sistema
  - Permite implementação de sistemas sequenciais ou concorrentes
- **Desvantagens**:
  - Definição de interfaces entre componentes (entrada -> saida)
    - *Overhead* no sistema
    - Impossibilidade no reúso de componentes que usam estruturas de dados incompatíveis

Escolha do padrão de arquitetura

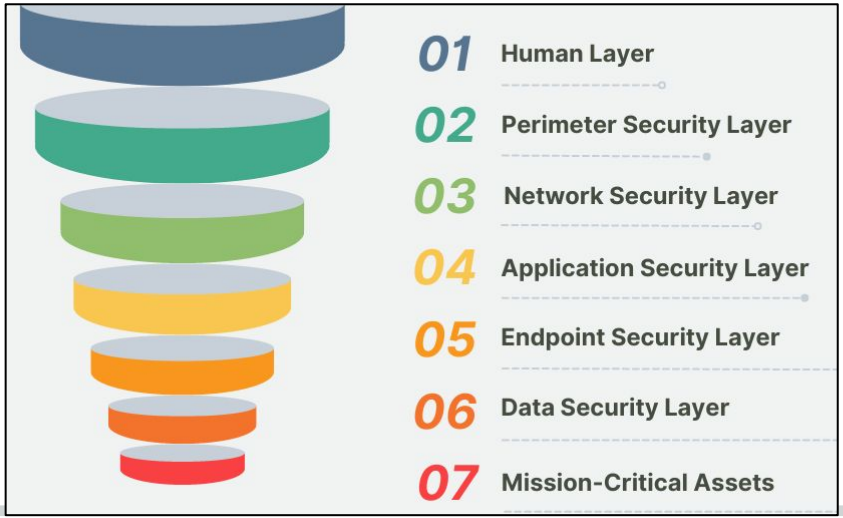
- **A arquitetura escolhida vai depender dos requisitos não funcionais do seu sistema**
  - Desempenho
  - Proteção
  - Segurança
  - Disponibilidade
  - Manutenibilidade

Arquitetura com foco em **Desempenho**

- **Se o desempenho for um requisito crítico**, a arquitetura deve:
  - Colocar as operações críticas (funções, classes, etc) dentro de um pequeno número de componentes,
    - Todos esses componentes devem estar no mesmo computador, em vez de distribuídos pela rede
    - Isso pode significar o uso de alguns componentes relativamente grandes (código grande e complexo).
  - Você também pode considerar uma arquitetura que permita que a execução *multithread* em diferentes processadores

Arquitetura com foco em **Proteção**

- **Deve ser usada uma estrutura em camadas para a arquitetura**
  - Os ativos mais críticos devem estar protegidos em camadas mais internas, com alto nível de proteção



Arquitetura com foco em **Segurança**

- **As operações relacionadas à segurança estão localizadas em um único componente ou em um pequeno número de componentes**
  - Reduz os custos e problemas de validação de segurança
  - Torna possível fornecer sistemas de proteção relacionados que podem desligar o sistema de maneira segura em caso de falha

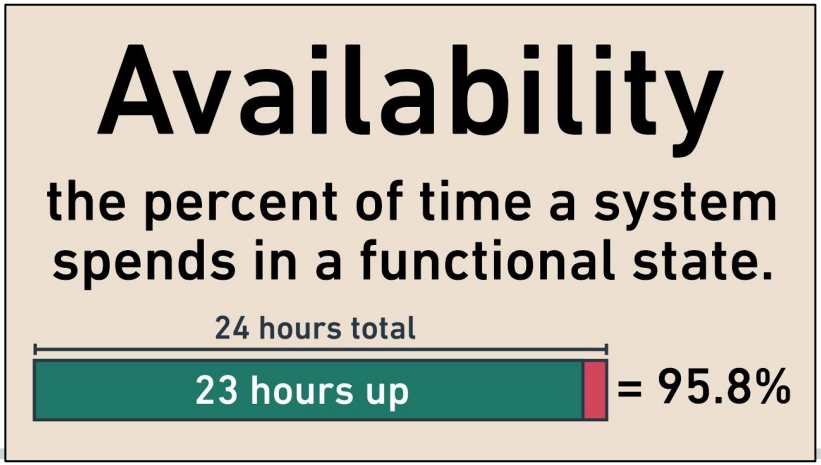
ATENÇÃO:
Segurança e proteção não são a mesma coisa.

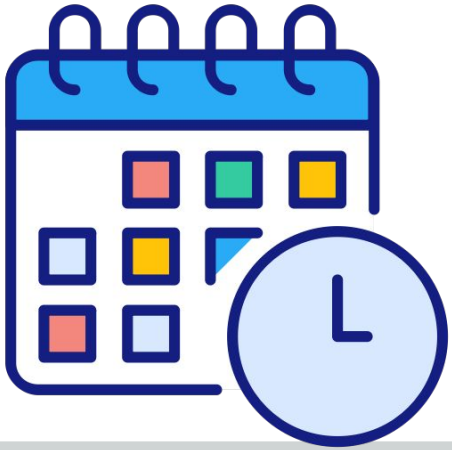
Segurança x Proteção

- **Segurança e proteção não são sinônimos**
  - **Segurança:** Associada à capacidade do sistema de evitar falhas e problemas internos
    - **Ex**: defeitos no hardware ou software de bomba de insulina
  - **Proteção:** Associada a evitar problemas externos
    - **Ex**: tentativas de invasão, vazamento de dados, etc

Arquitetura com foco em **Disponibilidade**

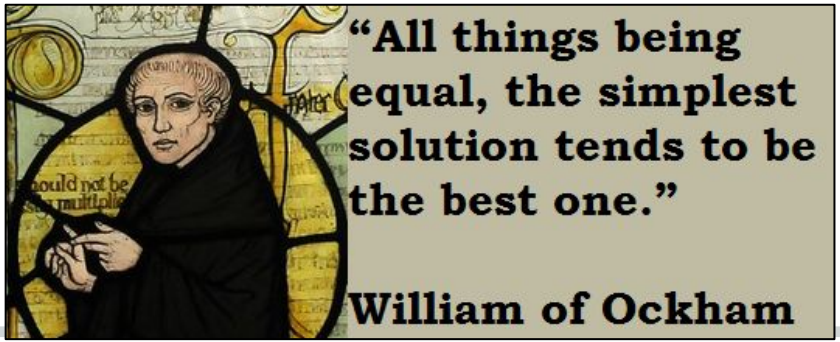
- **Disponibilidade:** *“quantidade de tempo que o sistema permanece funcionando”*
- **Como:** Incluir componentes redundantes, de modo que seja possível substituir e atualizar componentes sem parar o sistema

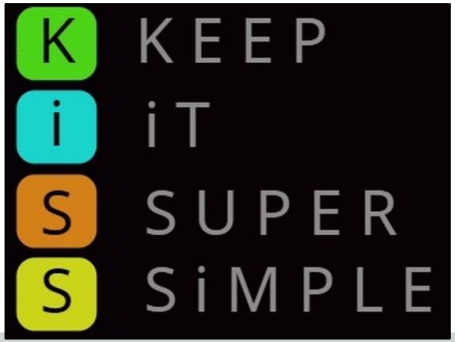




Arquitetura com foco em **Manutenibilidade**

- **Componentes autocontidos de baixa granularidade que podem ser rapidamente alterados**
  - Código simples, com poucas dependencias entre componentes
  - Estruturas de dados não devem ser compartilhadas entre componentes ou programas (*threads*, processos no SO)





Arquiteturas 100% otimizadas para todos os requisitos são uma impossibilidade

- **Não é possível obter uma arquitetura 100% otimizada para todas as propriedades e requisitos não funcionais**
  - **Ex:**
    - **Arquitetura de alto desempenho**: prevê a utilização de componentes grandes, altamente integrados
    - **Arquitetura de alta manutenibilidade:** recomenda o uso de componentes pequenos, fracamente integrados

Arquiteturas 100% otimizadas para todos os requisitos são uma impossibilidade

- **Não é possível obter uma arquitetura 100% otimizada para todas as propriedades e requisitos não funcionais**
  - **Solução:** utilizar diferentes padrões ou estilos de arquitetura para diferentes partes do sistema
    - **Ex**:
      - Melhor desempenho onde necessário
      - Foco em manutenibilidade no restante

Referencial Bibliográfico

- SOMMERVILLE, Ian. **Engenharia de Software**. 6. ed. São Paulo: Addison-Wesley, 2003.

- PRESSMAN, Roger S. **Engenharia de Software**. São Paulo: Makron Books, 1995.

- JUNIOR, H. E. **Engenharia de Software na Prática**. Novatec, 2010.