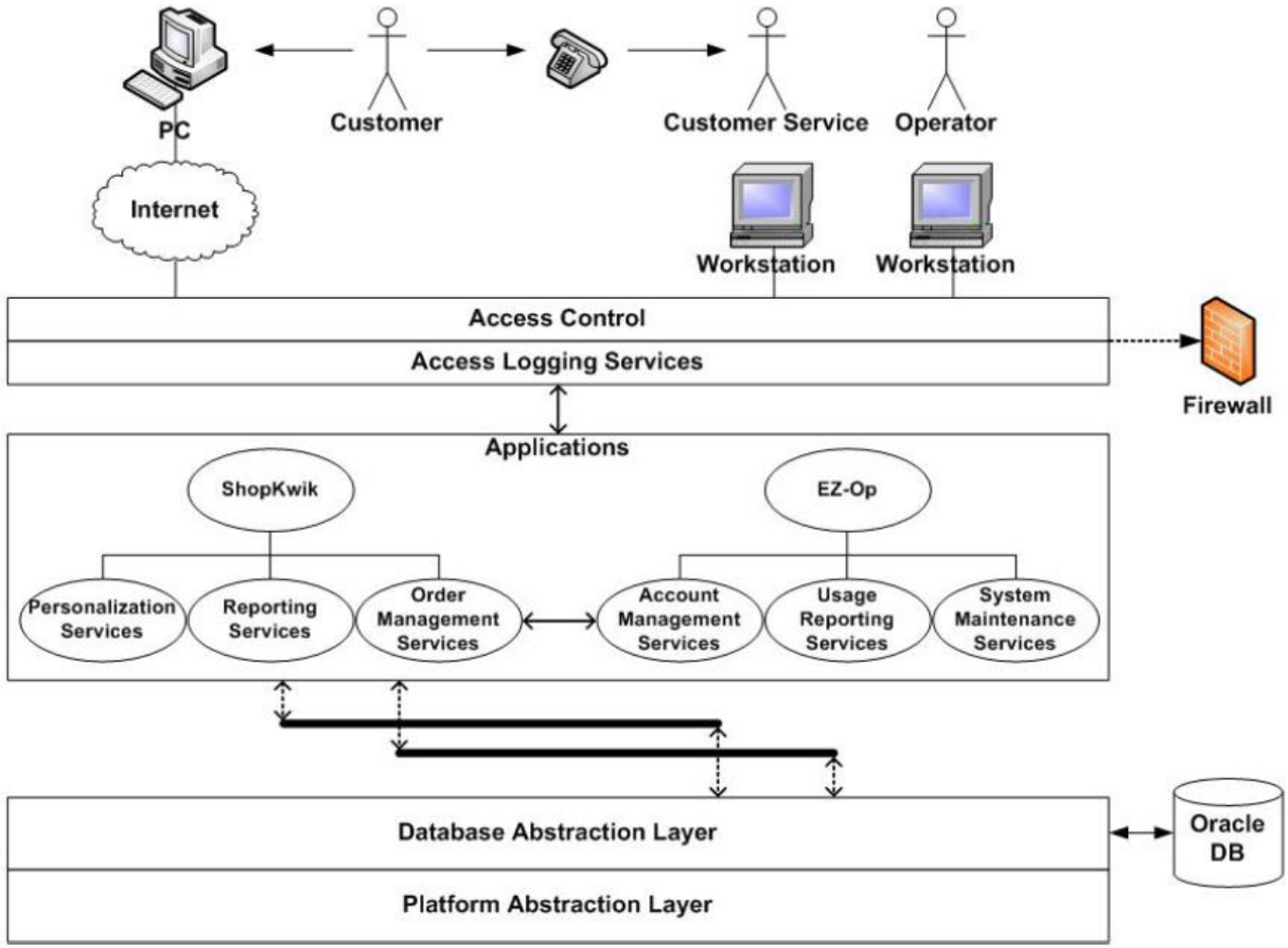
Arquitetura de Software

Profa. Dra. Jacilane Rabelo

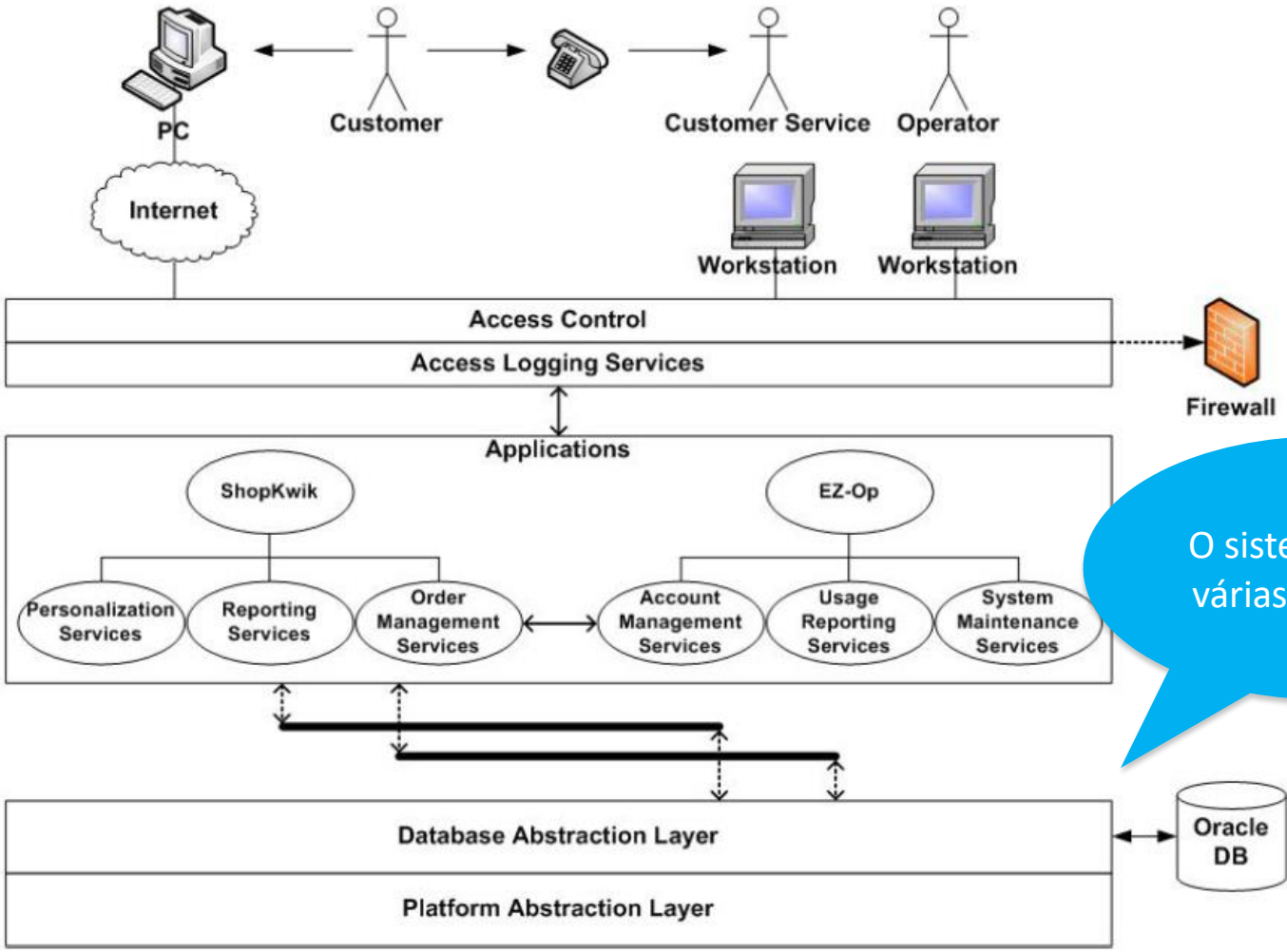
Baseado em material do Prof. Hyggo de Almeida e Profa. Dra. Anna Beatriz Marques

O QUE É ARQUITETURA DE SOFTWARE?



PODEMOS VER A PARTIR RAMA?

Os elementos são objetos, tarefas, processos, programas, dados?



O sistema possui várias camadas?

O que é Arquitetura de Software?
Muitas definições...

- “...conjunto de elementos arquiteturais (ou de projeto) que têm uma forma particular.” **Perry and Wolf, 1992**
- “...estrutura(s) do sistema, que compreende(em) os componentes de software, as propriedades externamente visíveis desses componentes, e os relacionamentos entre eles.” **Clements et al., 1997**
- "...especificação abstrata do sistema consistindo primeiramente de componentes funcionais descritos em termos de seus comportamentos, interfaces e interconexões entre componentes." **Hayes-Roth, 1994**
- "Arquitetura de software é o estudo da estrutura em larga escala e desempenho de sistemas de software" **Lane, 1990**
- "A arquitetura de um sistema complexo é seu estilo e método de projeto e construção" **Hayes-Roth, 1995**
- “...estrutura de suporte de um sistema“ **Rechtin, 1992**
- "...estrutura de sistema que consiste de módulos ativos, um mecanismo para permitir interação entre estes módulos e um conjunto de regras que governam a interação" **Boasson, 1995**

O que é Arquitetura de Software?
Muitas definições...

- “...conjunto de elementos arquiteturais (ou de projeto) que têm uma forma particular.” **Perry and Wolf, 1992**
- “...estrutura(s) do sistema, que compreende(em) os componentes de software, as propriedades externamente visíveis desses componentes, e os relacionamentos entre eles.” **Clements et al., 1997**
- "...especificação abstrata do sistema consistindo primeiramente de componentes funcionais descritos em termos de seus comportamentos, interfaces e interconexões entre componentes." **Hayes-Roth, 1994**
- "Arquitetura de software é o estudo da estrutura em larga escala e desempenho de sistemas de software" **Lane, 1990**
- "A arquitetura de um sistema complexo é seu estilo e método de projeto e construção" **Hayes-Roth, 1995**
- “...estrutura de suporte de um sistema“ **Rechtin, 1992**
- "...estrutura de sistema que consiste de módulos ativos, um mecanismo para permitir interação entre estes módulos e um conjunto de regras que governam a interação" **Boasson, 1995**

O que é Arquitetura de Software?
Resumindo...

- Elementos em comum...
  - Descrição dos componentes principais
  - Relacionamentos e interações entre componentes
  - Omite informação sobre o conteúdo dos componentes não relacionada a suas interações
  - O comportamento dos componentes é uma parte da arquitetura enquanto possa ser discernido do ponto de vista de outro componente
  - A arquitetura define uma lógica por trás dos componentes e da estrutura

O QUE É ARQUITETURA DE SOFTWARE?

- O principal desafio é:

Como organizar um sistema para simultaneamente fornecer os serviços funcionais desejadas (requisitos funcionais) e garantir a qualidade das dos serviços (requisitos não-funcionais).

Estrutura: define a forma dos elementos que se relacionam.

- As visões a serem consideradas são:

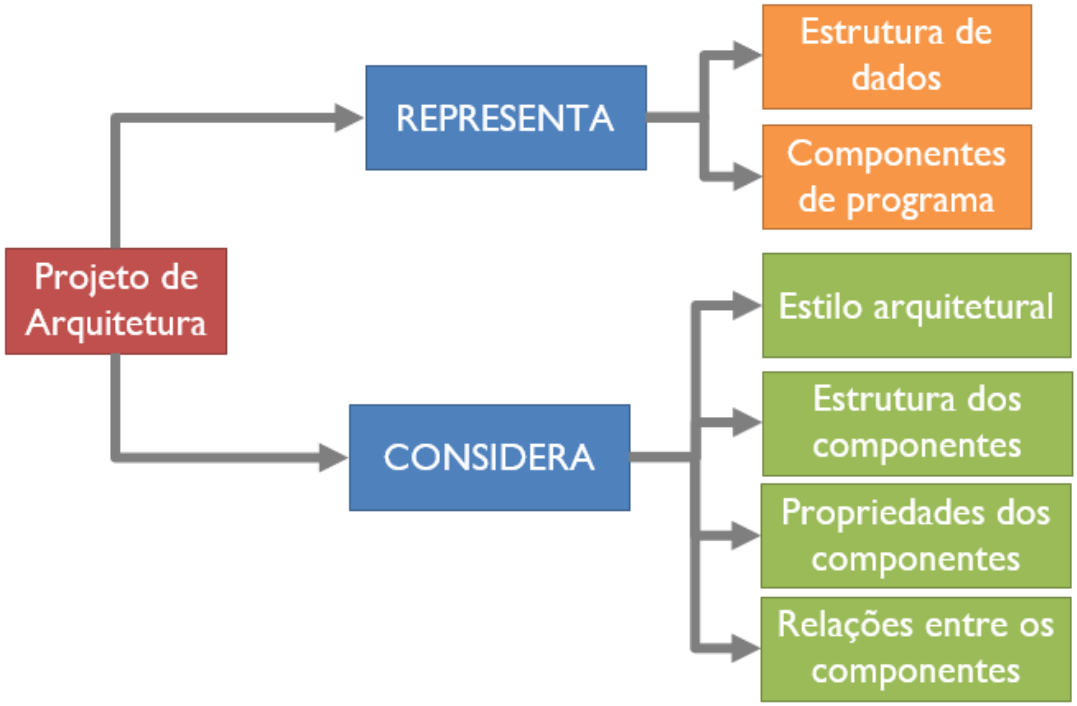
Comportamento: define as funcionalidades que os elementos fornecem.

Execução: como a funcionalidade fornecida pelos elementos é executada.

Análise: quais as propriedades não-funcionais que os elementos devem satisfazer

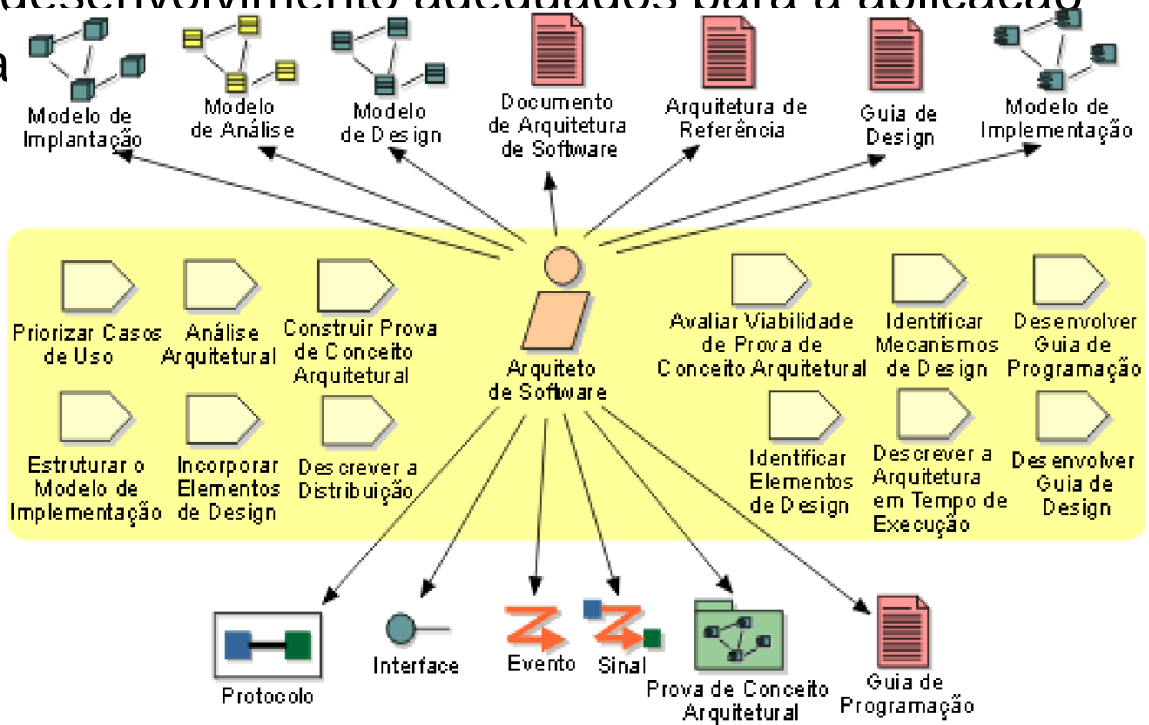
O QUE É ARQUITETURA DE SOFTWARE?

- É o conjunto das **principais decisões de design tomadas** durante o **desenvolvimento e futura evolução** de um sistema.



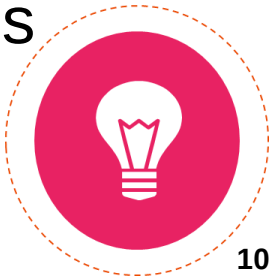
PAPEL DO ARQUITETO DE SOFTWARE

- Conhecimento profundo:
  - dos requisitos das aplicações;
  - das tecnologias disponíveis para apoio à construção da arquitetura e
  - dos processos de desenvolvimento adequados para a aplicação a ser desenvolvida



POR QUE ARQUITETURA DE SOFTWARE É IMPORTANTE?

- A arquitetura fornece uma **representação que facilita a comunicação** entre todos os envolvidos.
- A arquitetura destaca desde o início as **decisões de projeto** que terão um profundo **impacto no trabalho de engenharia de software** que se segue.
- A arquitetura constitui um **modelo relativamente pequeno** e intelectualmente **compreensível** de como o sistema é estruturado e como seus componentes trabalham em conjunto.



Dos Requisitos para a Arquitetura

Transição

- RNF são chamados de Atributos de Qualidade (AQ);
- AQ são os principais considerados para escolher o padrão arquitetural adotado;
- RF são agrupados por similaridade e modularizados em partes específicas para aumentar a manutenibilidade;

Transição

- Uma forma de auxiliar na manutenção da arquitetura e documentar a transição, é através do uso de MATRIZ DE RASTREABILIDADE.
- Rastreabilidade pode ser entre:
  - RF e RF;
  - RF e RNF;
  - RF e Arquitetura;
  - RNF e Arquitetura;

Atributos de Qualidade

Atributos de qualidade

- Arquitetura e funcionalidade
  - Se a funcionalidade fosse o único atributo buscado no desenvolvimento de um software...
  - ... sua arquitetura seria sempre monolítica: uma só função, um só componente, uma só classe...
  - Outros atributos: mutabilidade (*modifiability*), usabilidade, desempenho...
  - ... influenciam na determinação da arquitetura do software
  - É com base em atributos de qualidade interessantes para o sistema que se determina a sua arquitetura

Atributos de qualidade

- Classes de atributos
  - **Qualidades de sistema:** disponibilidade, mutabilidade, desempenho, usabilidade...
  - **Qualidades de negócio:** tempo de produção (*time to market*), custo e benefício...
  - **Qualidades de arquitetura:** *buildability*, integridade conceitual...

Atributos de qualidade

Qualidades de sistema

- Disponibilidade
- Mutabilidade
- Desempenho
- Segurança
- Testabilidade
- Usabilidade

Atributos de qualidade

Qualidades de sistema

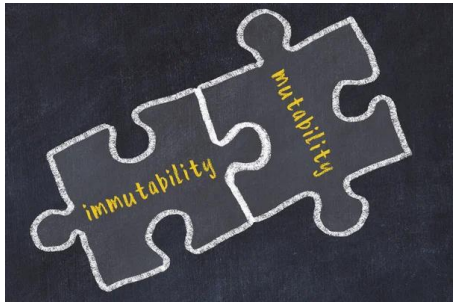
- Disponibilidade
  - Relacionada a falhas no sistema e suas consequências
  - Um sistema está em *falha* quando não funciona mais de acordo com a sua especificação
  - Uma falha é observável do ponto de vista externo
  - Medida de disponibilidade:

$$\text{disp} = \frac{\text{tempo médio para falhar}}{\text{tempo médio para falhar} + \text{tempo médio para reparar}}$$

Atributos de qualidade

Qualidades de sistema

- Mutabilidade
  - Relacionado ao custo de mudanças
  - O que pode mudar?
    - Implementação de funcionalidades
    - Plataforma na qual o sistema é executado (*hardware*, SO,...)
      - Portabilidade
    - O ambiente no qual opera (protocolos, rede, outros sistemas)
    - Capacidade (nº de usuários, nº de operações simultâneas)
      - Escalabilidade
  - Quem pode mudar?
    - Desenvolvedor, usuário final, administrador...



Atributos de qualidade

Qualidades de sistema

- Mutabilidade
  - Quando pode mudar?
    - Implementação (código fonte)
    - Construção – *build* (escolha de bibliotecas)
    - Configuração
    - Execução (parametrização)

Atributos de qualidade

Qualidades de sistema

- Desempenho
  - Relacionado a tempo!
  - Eventos ocorrem e o sistema tem que responder aos mesmos
  - A medida de desempenho é:
    - Quanto tempo leva o sistema para responder a um evento?
  - Evento???
    - Interrupções, mensagens, requisições do usuário, inicialização...
  - Exemplo:
    - Respostas a requisições do usuário não podem durar mais que 10 milisegundos!

Atributos de qualidade

Qualidades de sistema

- Segurança
  - Habilidade do sistema de impedir acesso não autorizado...
  - ... ainda garantindo acesso autorizado!
  - Segurança pode ser visto como uma composição de:
    - **Não repudiação:** uma transação não pode ser negada por nenhuma das partes envolvidas
    - **Confidencialidade:** proteção a acesso não autorizado
    - **Integridade:** de dados e serviços
    - **Garantia:** partes de uma transação são quem dizem que são
    - **Disponibilidade:** sistema disponível para uso sem falhas
    - **Auditoria:** caso ocorra falha, o sistema consegue recuperar-se sem perdas aos usuários

Atributos de qualidade

Qualidades de sistema

- Testabilidade
  - Facilidade com que podem ser demonstradas as faltas de um software através de testes
  - No mínimo, 40% do custo de desenvolvimento de software com “boa engenharia” é atribuído a testes...
  - ... se o arquiteto consegue reduzir este custo, o lucro é bem maior!

Atributos de qualidade

Qualidades de sistema

- Usabilidade
  - Quão fácil é para o usuário realizar uma tarefa desejada usando o sistema?
  - Que tipo de suporte o sistema provê para o usuário?
  - Áreas:
    - **Aprendizado das características do sistema:** o que o sistema pode fazer para ajudar no aprendizado do usuário?
    - **Uso eficiente do sistema:** o que o sistema pode fazer para que o usuário o utilize mais eficientemente?
    - **Minimização do impacto de erros:** o que o sistema pode fazer para minimizar o impacto de um erro cometido pelo usuário?
    - **Adaptação do sistema às necessidades do usuário:** como o usuário (ou o próprio sistema) pode adaptar o sistema para tornar as tarefas mais fáceis
    - **Aumento de confiança e satisfação:** o que o sistema faz para dar ao usuário a confiança de que ele está executando a tarefa corretamente?

Atributos de qualidade

Qualidades de negócio

- Tempo de produção (*time-to-market*)
  - Se há pressão competitiva ou janela de oportunidade restrita...
  - ... tempo de produção é essencial!
  - Em geral, reduzido com o uso componentes pré-construídos
    - Componentes COTS – *Commercial Off-The-Shelf*

Atributos de qualidade

Qualidades de negócio

- Custo e benefício
  - Orçamento não pode ser excedido
  - Diferentes arquiteturas levarão a diferentes custos de desenvolvimento
  - Arquiteturas mais flexíveis são mais caras!!!

Atributos de qualidade

Qualidades de negócio

- Tempo de vida projetado
  - Se o sistema é projetado para ter um longo ciclo de vida...
  - ... mutabilidade, escalabilidade e portabilidade se tornam extremamente importantes
  - Porém... isto influencia no custo!
  - Por outro lado, tais características diminuem custos de manutenção
  - Sistemas projetados para um curto ciclo de vida podem ser mais brandos em relação a estas características!
    - Será??? Como prever o ciclo de vida?

Atributos de qualidade

Qualidades de negócio

- Mercado alvo
  - Considerando softwares de propósito geral, a quantidade de plataformas de execução determina o mercado em potencial
    - Exemplo: seu sistema de controle de estoque pode executar:
      - Em Windows, Linux e Mac?!
      - Em rede ou standalone?!
      - Em PC ou dispositivo móvel?
    - Portabilidade é chave! Usabilidade e desempenho também!
  - Solução utilizada
    - Linhas de produto
      - Núcleo em comum + características específicas

Atributos de qualidade

Qualidades de negócio

- Agenda de divulgação
  - Liberação do produto como um todo???
  - Liberação de funcionalidade base e depois liberação de funcionalidades adicionais?
    - Escalabilidade
    - Flexibilidade
  - Facilidade de expansão e contração
    - Diferentes usuários terão diferentes necessidades
    - Exemplo: Eclipse

Atributos de qualidade

Qualidades de negócio

- Integração com sistemas legados
  - Integração com sistemas e tecnologias existentes
  - Impacto direto na arquitetura
    - Deve funcionar de acordo com a especificação de terceiros
    - Deve se adequar à arquitetura de terceiros
    - Pode haver incompatibilidades!

Atributos de qualidade

Qualidades de arquitetura

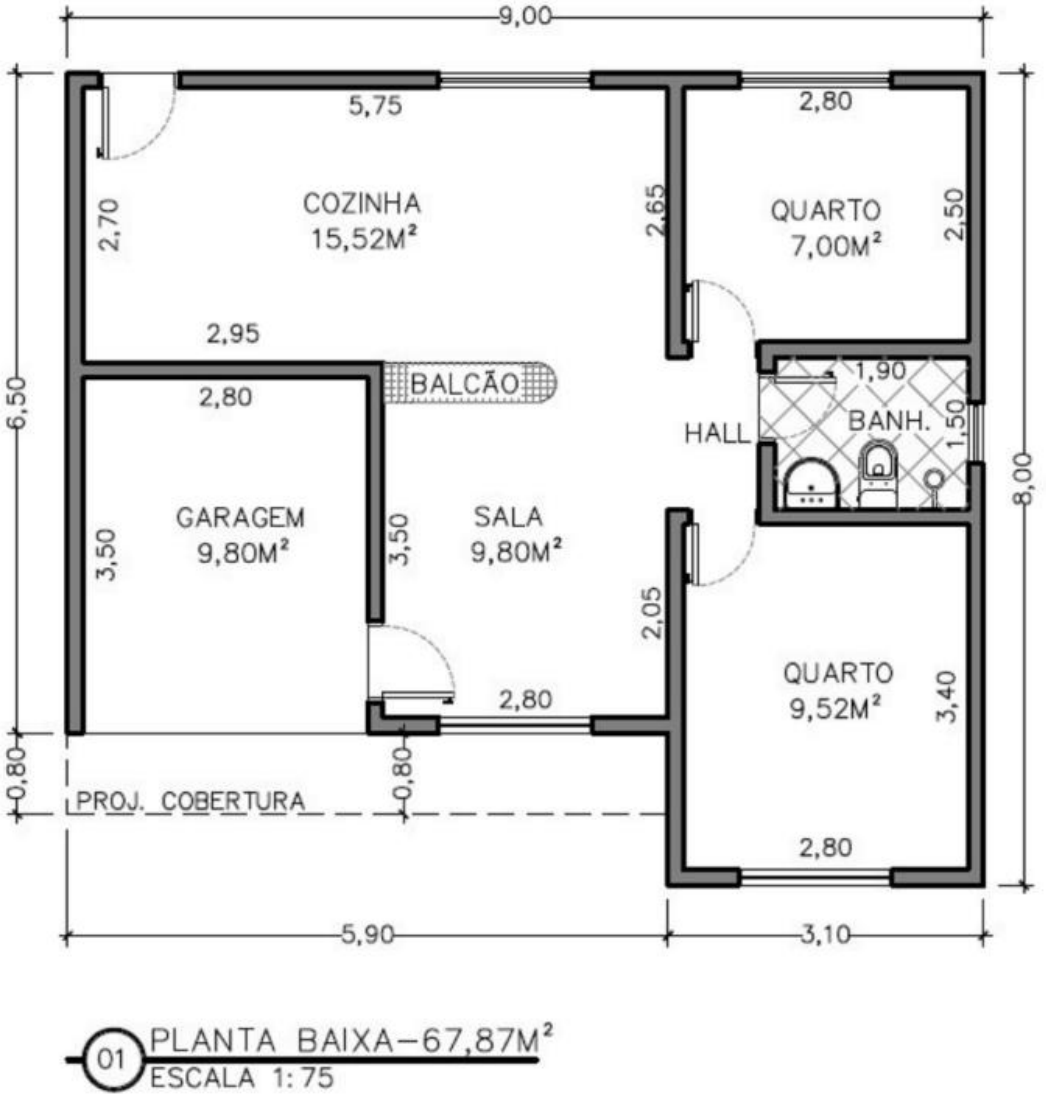
- *Buildability*
  - Facilidade do sistema ser construído, maximizando o paralelismo de desenvolvimento e manutenção
- Integridade conceitual
  - Visão que unifica o projeto arquitetura em todos os níveis
- Corretude e completude
  - Análise e verificação formal dos requisitos
  - Garante que a arquitetura contempla os requisitos
  - Complementar aos testes

Próximos passos

- Após definir os requisitos funcionais, requisitos não funcionais e requisitos de domínio:
  - Estruture um Documento de Requisitos correspondente para que, a partir dele, possa efetuar **decisões arquiteturais sobre estilo arquitetural** e alocação de RF e RNF na arquitetura proposta.
  - Elabore a matriz de rastreabilidade entre RF x RF; RF x RNF; RF x RFUF (requisitos funcionais de usabilidade)

O que é arquitetura de software?

- Quero que vocês observem **com bastante atenção** a imagem da planta baixa
- Vejam cada área, cada limite, cada instrução dada nesse modelo que é comumente utilizado por arquitetos



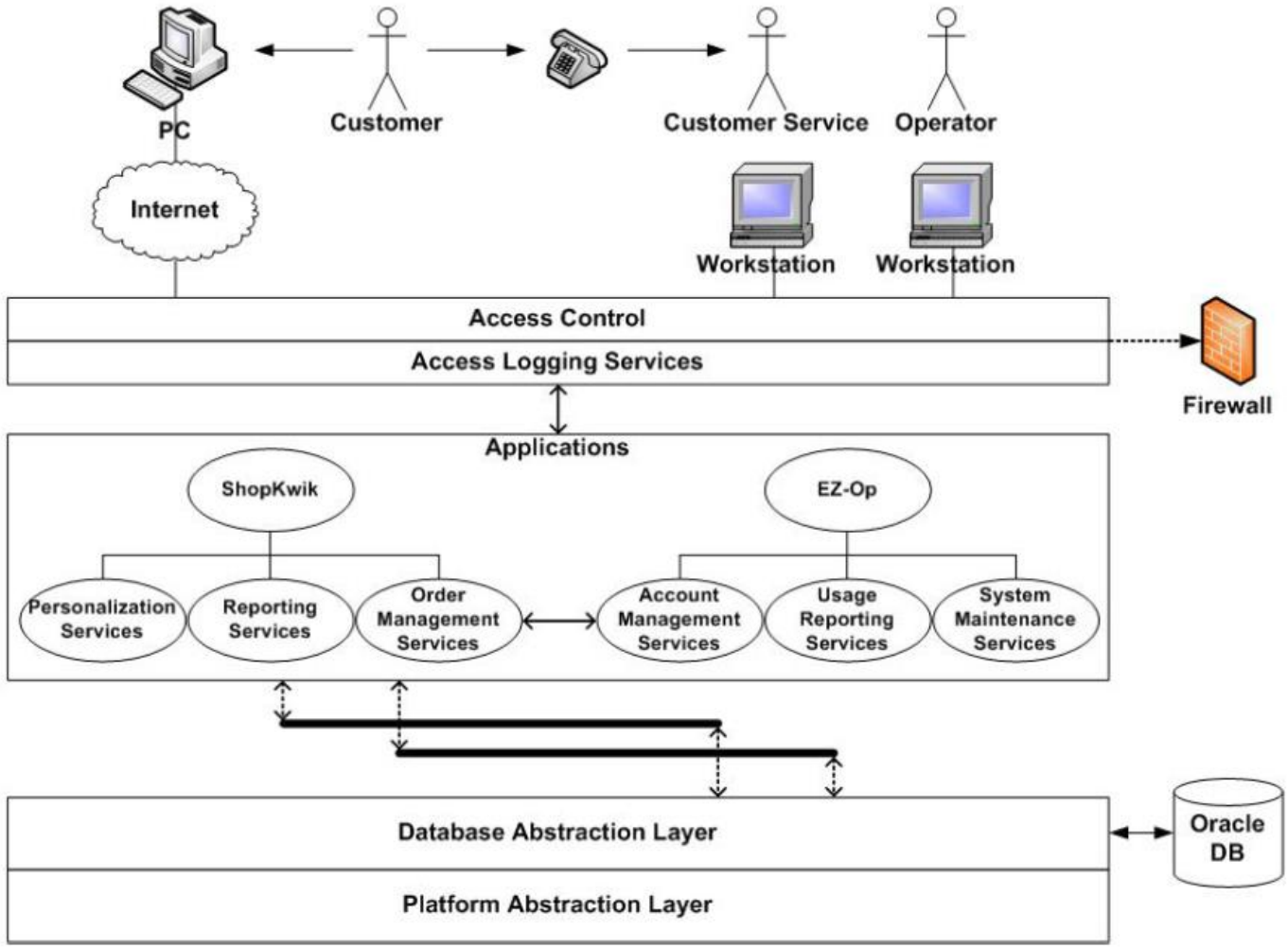
O que é arquitetura de software?

- Depois de um tempo observando, quero que vocês respondam para si mesmos algumas perguntas:
  - **Vocês conseguiram identificar do que se trata essa planta ?**
  - **Vocês conseguiram achar a porta de entrada?**
  - **Vocês conseguiram identificar facilmente os cômodos ?**
  - **Vocês conseguiram identificar entradas e saídas entre os cômodos?**
  - **Vocês conseguiram identificar áreas que só serão acessadas se você obrigatoriamente passar por uma outra área antes?**
  - **Vocês conseguiram ter uma ideia de tamanho/porte ?**

O que é arquitetura de software?

- Se vocês responderam rapidamente “**SIM**” para todas essas perguntas, podemos dizer que o objetivo dessa planta baixa foi atingida, pois o grande desafio dos arquitetos é tentar estruturar uma ideia macro para uma futura execução

O que é arquitetura de software?



Estilos arquiteturais

- Um estilo/padrão arquitetural expressa:
  - Uma organização estrutural
  - Um conjunto pré-definido de subsistemas e suas responsabilidades
  - Inclui regras e diretrizes para organizar o relacionamento entre os subsistemas

- São “templates” para arquiteturas concretas

Estilos arquiteturais

- Divisão em Camadas (Layers)
- Cliente-Servidor
- Pipes and Filters
- Model-View-Controller
- Broker

Estilos arquiteturais

Divisão em camadas

- Estrutura aplicações que podem ser decompostas em grupos de subáreas
  - Cada grupo está em um nível de abstração
- Exemplo: Modelo OSI (Open System Interconnection)

Fornecer protocolos comuns para as aplicações

Estruturar a informação

Dar suporte para "diálogos" e sincronização

Dividir a mensagem em pacotes e garantir a entrega

Escolher uma rota do remetente ao destinatário

Detectar e corrigir erros nas seqüências de bits

Transmitir bits: velocidade, bit-code, conexão, etc.

Estilos arquiteturais

Divisão em camadas

- Problema
  - Sistema com várias camadas de abstração
  - Camadas de níveis superiores dependem das camadas de níveis inferiores
  - Partes do sistema devem poder ser trocadas
    - Trocar a camada de comunicação
    - Trocar a camada de acesso a dados
- Podem existir várias camadas em um mesmo nível de abstração dependendo de camadas inferiores
  - Interface gráfica standalone (Swing) X Interface WEB

Estilos arquiteturais

Divisão em camadas

- Solução
  - Estruture o sistema em camadas
  - Sobreponha as camadas em níveis de abstração
    - Serviços da camada X são compostos de serviços da camada X-1

Estilos arquiteturais

Divisão em camadas

- Conseqüências
  - Pontos positivos
    - Reúso das camadas
    - Dependências tendem a permanecer “locais”
- Pontos negativos
  - Cascateamento de alterações para as camadas superiores quando o comportamento de uma camada inferior muda

Estilos arquiteturais

Divisão em camadas

- A aplicação que vocês irão projetar poderia utilizar o padrão Divisão em Camadas?

- Faça o desenho desta arquitetura da sua aplicação

- Quais seriam as vantagens e desvantagens?

Estilos arquiteturais

Cliente-Servidor

- Baseado em programas servidores e programas clientes
- Cliente
  - Estabelece a conexão, envia mensagens para o servidor e aguarda mensagens de resposta.
- Servidor
  - Aguarda mensagens, executa serviços e retorna resultados.

Estilos arquiteturais

Cliente-Servidor

- Os primeiros sistemas cliente-servidor surgiram por causa de limitações nas arquiteturas de compartilhamento de arquivos
  - Introdução de um banco de dados substituindo o servidor de arquivos
  - Reduziu o tráfego de rede, uma vez que somente as consultas (e suas respostas) trafegavam, ao invés de arquivos

Estilos arquiteturais

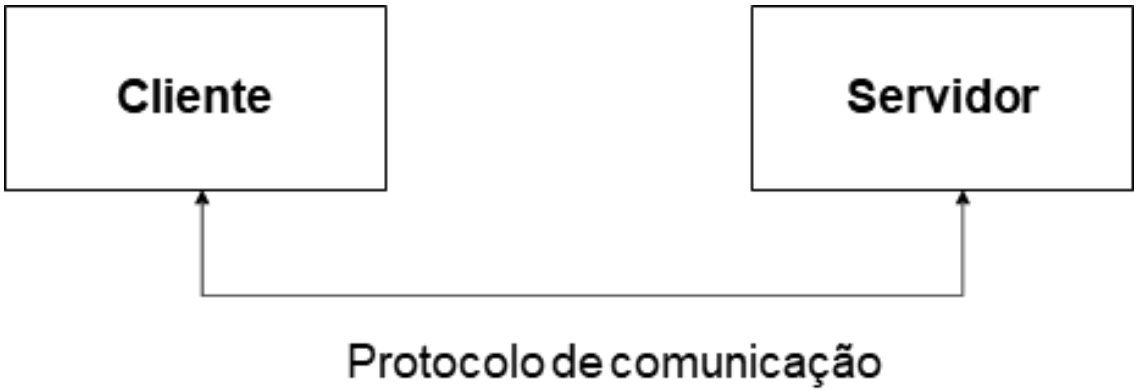
Cliente-Servidor

- Atualmente muitas aplicações possuem arquitetura Cliente-Servidor
  - Navegador WEB X Servidor WEB
  - Programa JAVA X Gerenciador de Banco de Dados

Estilos arquiteturais

Cliente-Servidor

- Estrutura



Estilos arquiteturais

Cliente-Servidor

- Vantagens
  - Utilização dos recursos do servidor
  - Escalabilidade
  - Aumentando a capacidade computacional do servidor

- Desvantagens
  - Introduz complexidade
  - Custos de comunicação

Estilos arquiteturais

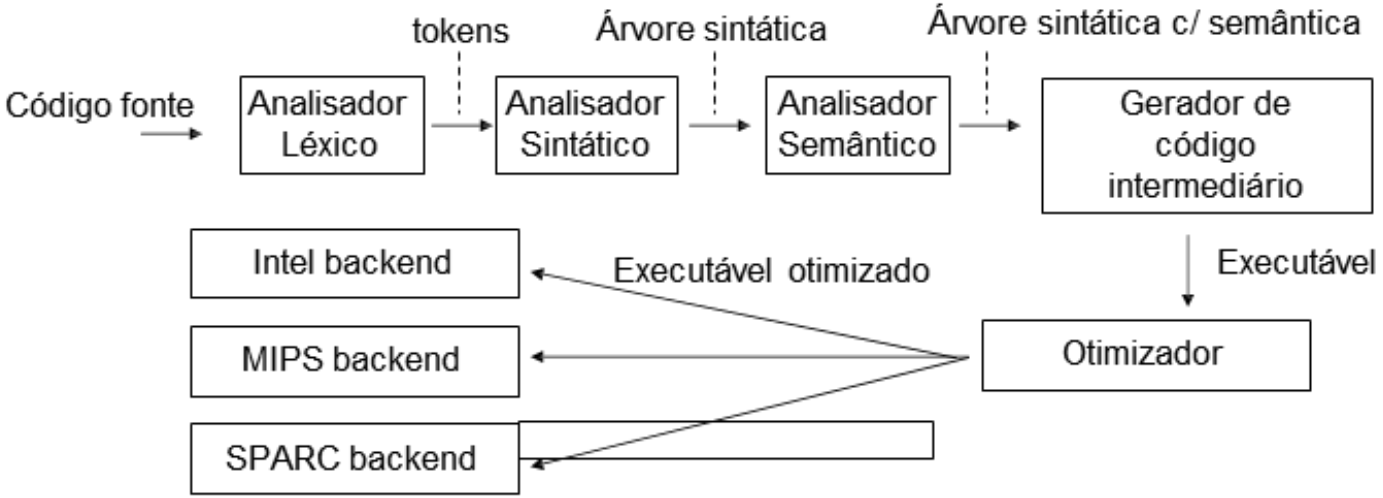
Cliente-Servidor

- Descreva a arquitetura de um sistema de troca de mensagens instantâneas usando o padrão cliente - servidor

Estilos arquiteturais

Pipes and filters

- Problema
  - Desenvolvimento de um sistema que processa ou transforma um “stream” de dados.
- Exemplo:
  - Compilador



Estilos arquiteturais

Pipes and filters

- Como dividir a tarefa entre vários desenvolvedores?
- E se for possível trocar etapas no processo?
- E recombinar etapas?

Estilos arquiteturais

Pipes and filters

- Solução
  - Dividir a tarefa entre várias etapas sequenciais
  - Saída de uma etapa é a entrada da etapa seguinte
  - Cada etapa de processamento é implementada por um filtro (filter)
    - Consome e entrega os dados incrementalmente, ao invés de consumir todos os dados de uma só vez
  - Cada “pipe” implementa o fluxo dos dados entre os filtros

Estilos arquiteturais

Pipes and filters

- Solução
  - Filter
    - Receber o dado da entrada
    - Processar o dado
    - Colocar o dado na saída
  - Pipe
    - Transferir o dado
    - Realizar “buffer”
    - Sincronizar os filtros “vizinhos”
  - Outros participantes
    - Fonte de dados
    - Saída de dados

Estilos arquiteturais

Pipes and filters

- Conseqüências
  - Não é preciso criar arquivos intermediários (mas é possível)
  - Flexibilidade na troca de filtros
  - Flexibilidade na recombinação
  - Eficiência no processamento em paralelo
    - Vários filtros consumindo e produzindo dados em paralelo.
- Ponto negativo
  - Gerenciamento de erros
    - Ausência de um estado global compartilhado

Estilos arquiteturais

Model-View-Controller

- A aplicação é dividida em 3 componentes
  - Model – contém a funcionalidade principal e os dados
  - View – exibe a informação aos usuários
  - Controller – gerenciam a entrada do usuário, deixando o modelo transparente

- Interface do usuário = View + Controller

Estilos arquiteturais

Model-View-Controller

- Quando usar?
  - Necessidade de várias interfaces com o usuário
    - WEB, Swing
  - Necessidade de várias visões dos dados
    - Planilhas, Tabelas, Gráficos
  - Mudanças nos dados devem ser refletidas na interface

Estilos arquiteturais

Model-View-Controller

- Estrutura
  - Model
    - Funcionalidade principal da aplicação
    - Registrar controllers e views
    - Notificar controllers e views registrados de alterações

Estilos arquiteturais

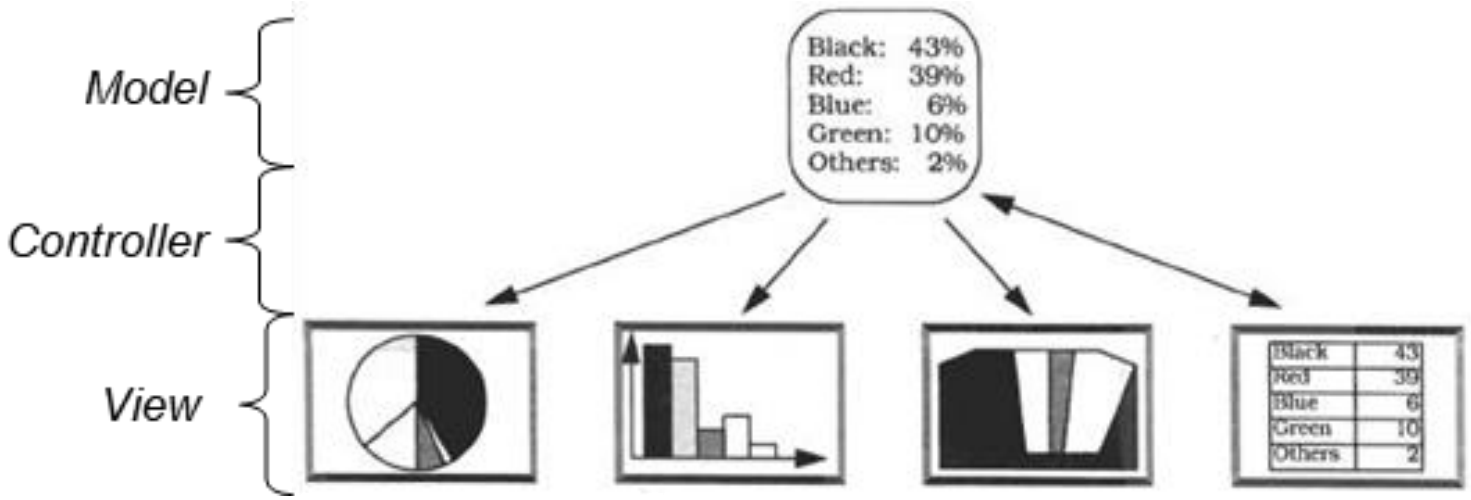
Model-View-Controller

- Estrutura
  - View
    - Criar e inicializar o controlador associado
    - Exibir informação ao usuário
    - Implementar o procedimento de atualização
    - Recuperar dados do modelo
  - Controller
    - Receber a entrada de dados/requisições do usuário
    - Transformar as requisições dos usuários em requisições ao modelo
    - Implementar o procedimento de atualização (se necessário)

Estilos arquiteturais

Model-View-Controller

- Exemplo: Pesquisa de opinião
  - Usuários interagem através da interfaces
  - Mudanças devem refletir nas outras interfaces



Estilos arquiteturais

Model-View-Controller

- Conseqüências
  - Pontos positivos
    - Múltiplas “views” de um mesmo modelo
    - “views” sincronizadas
    - Organização clara de abstrações
  - Pontos negativos
    - Aumento da complexidade
    - “Controllers” e “Views” tendem a ser bastante acoplados

Estilos arquiteturais

Model-View-Controller

- Estruture uma parte da aplicação que você irá projetar utilizando o estilo Model-View-Controller.

- Quantos requisitos funcionais estão sendo atendidos?

- Quais as vantagens e desvantagens?

Estilos e padrões arquiteturais

Exercício

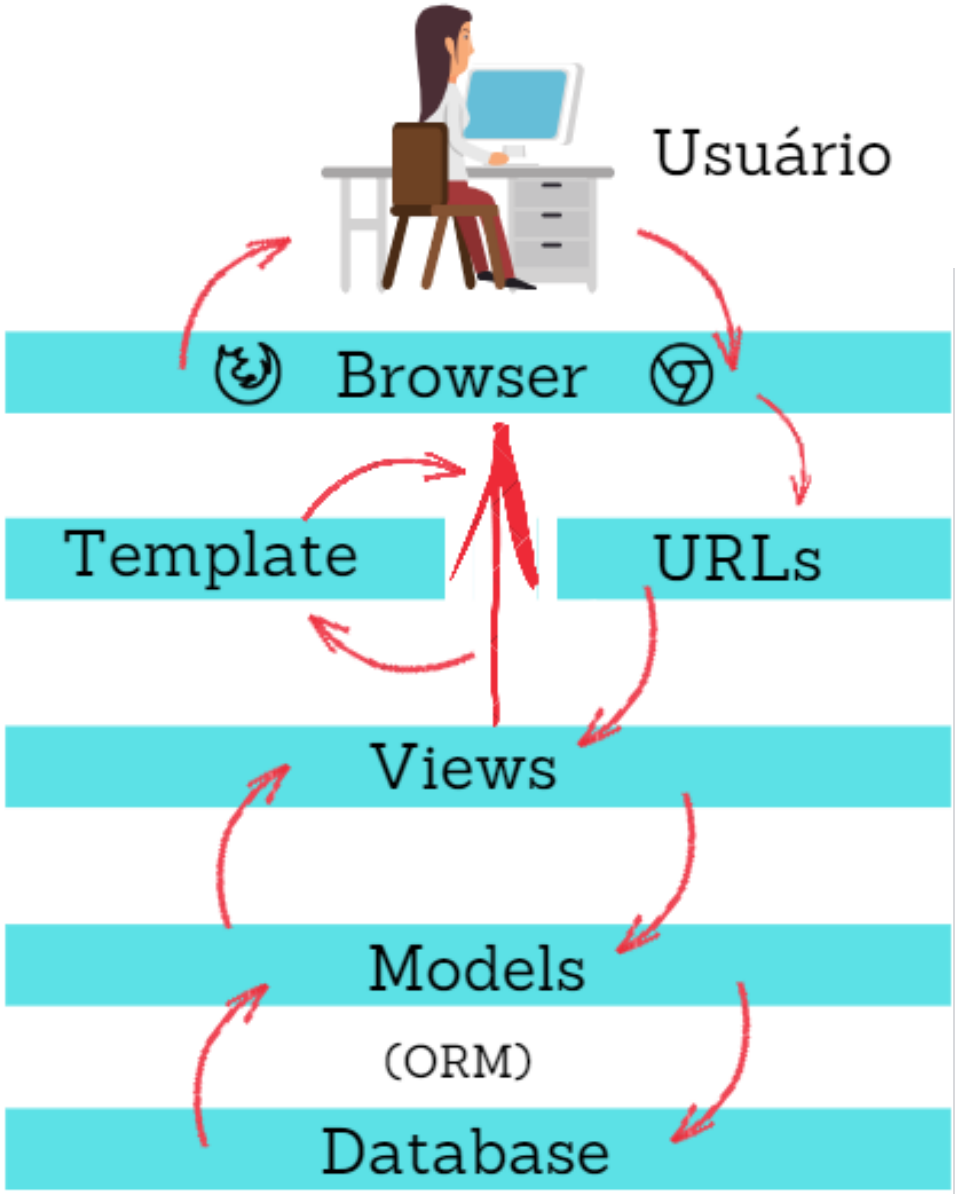
- Quais estilos arquiteturais podem ser aplicados na aplicação que você está projetando? Como?
- Se algum estilo não puder ser aplicado, justifique o porquê.
  - Arquitetura em camadas
  - Cliente-Servidor
  - Model-View-Controller
  - Pipers-and-Filters

MVT (Model View Template)

- Um projeto Django possui como padrão de projeto o MTV (Model, Template, View)
  - Model: Mapeamento do banco de dados para o projeto;
  - Template: Páginas para visualização de dados. Normalmente, é aqui que fica o HTML que será renderizado nos navegadores;
  - View: Lógica de negócio. É aqui que determinamos o que irá acontecer em nosso projeto.

MVT (Model View Template)

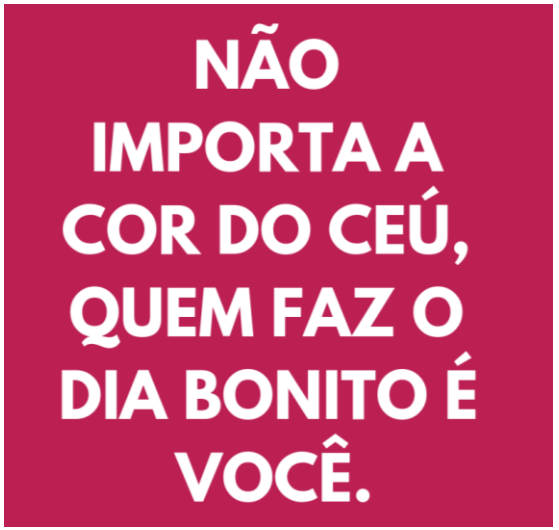
- Um projeto Django possui como padrão de projeto o MTV (Model, Template, View)
  - Model: Mapeamento do banco de dados para o projeto;
  - Template: Páginas para visualização de dados. Normalmente, é aqui que fica o HTML que será renderizado nos navegadores;
  - View: Lógica de negócio. É aqui que determinamos o que irá acontecer em nosso projeto.



O usuário faz uma requisição pelo browser, utilizando uma rota, é executado um método das *Views,* que utiliza os *Models* para acessar o banco de dados e retornar as informações. Estas informações são renderizadas pela camada de *Template* e, finalmente, é renderizado para o usuário pelo navegador.

- Resumidamente, a equipe de desenvolvimento do framework entende que a camada **view descreve quais dados serão apresentados ao usuário**, não a forma (aparência) que eles serão exibidos. Portanto, no padrão MTV, u**ma view é uma função que retorna algo para uma solicitação**, porque ela define apenas quais dados serão apresentados, não como serão mostrados.
- Além disso, é sensato separar o conteúdo da apresentação (por questões de organização e padronização do código). É aí que entra **camada template**. Com os dados retornados pela view, a template fica responsável por definir a forma que esses dados serão apresentados, **normalmente em páginas HTML.**

Padrões Arquiteturais



Estilos e padrões arquiteturais

- Estilos arquiteturais
  - Definem meios de selecionar e apresentar blocos de construção de arquitetura (*Shaw*)

- Padrões arquiteturais
  - Projetos de alto nível, testados e validados, de blocos de construção de arquitetura (*Buschman*).

Estilos e padrões arquiteturais

Classificação

- Invocação/Retorno (*Call/Return*)
  - Programa principal/Subrotina (*Main Program/Subroutine*)
  - Invocação remota de procedimento (*Remote Procedure Call - RPC*)
  - Camadas (*Layered*)
- Componentes independentes (*Independent Components*)
  - Comunicação de processos (*Communicating Processes*)
  - Baseado em eventos
  - Arquitetura orientada a serviços (SOA)
- Máquina virtual (*Virtual Machine*)
  - Interpretador (*Interpreter*)
  - Baseado em regras (*Rule-based*)
- Centrado em dados (*Data-Centered*)
  - Repositório (*Repository*)
  - Quadro negro (*Blackboard*)
- Fluxo de dados (*Data-Flow*)
  - Seqüencial (*Batch Sequential*)
  - Tubos e filtros (*Pipe and Filter*)
  - *Map-Reduce*

Estilos e padrões arquiteturais

Classificação

Invocação/Retorno (*Call/Return*)

Programa principal/Subrotina (*Main Program/Subroutine*)
Invocação remota de procedimento (*Remote Procedure Call - RPC*)
Camadas (*Layered*)

Componentes independentes (*Independent Components*)

Comunicação de processos (*Communicating Processes*)
Baseado em eventos
Arquitetura orientada a serviços (SOA)

Máquina virtual (*Virtual Machine*)

Interpretador (*Interpreter*)
Baseado em regras (*Rule-based*)

Centrado em dados (*Data-Centered*)

Repositório (*Repository*)
Quadro negro (*Blackboard*)

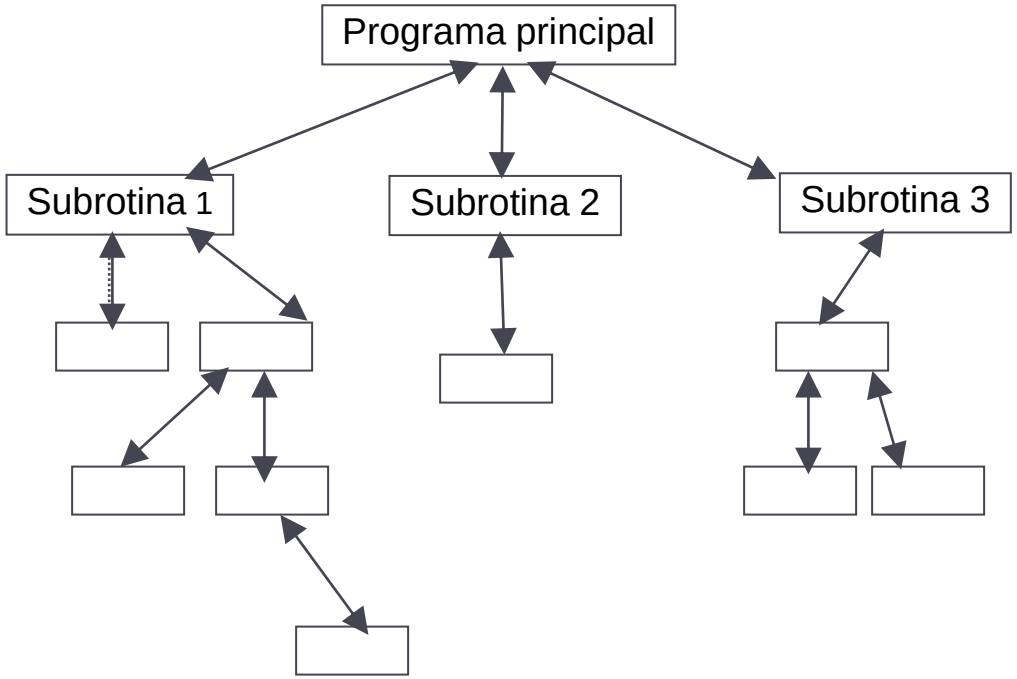
Fluxo de dados (*Data-Flow*)

Seqüencial (*Batch Sequential*)
Tubos e filtros (*Pipe and Filter*)
Map-Reduce

Estilos e padrões arquiteturais

Invocação/Retorno (Call/Return)

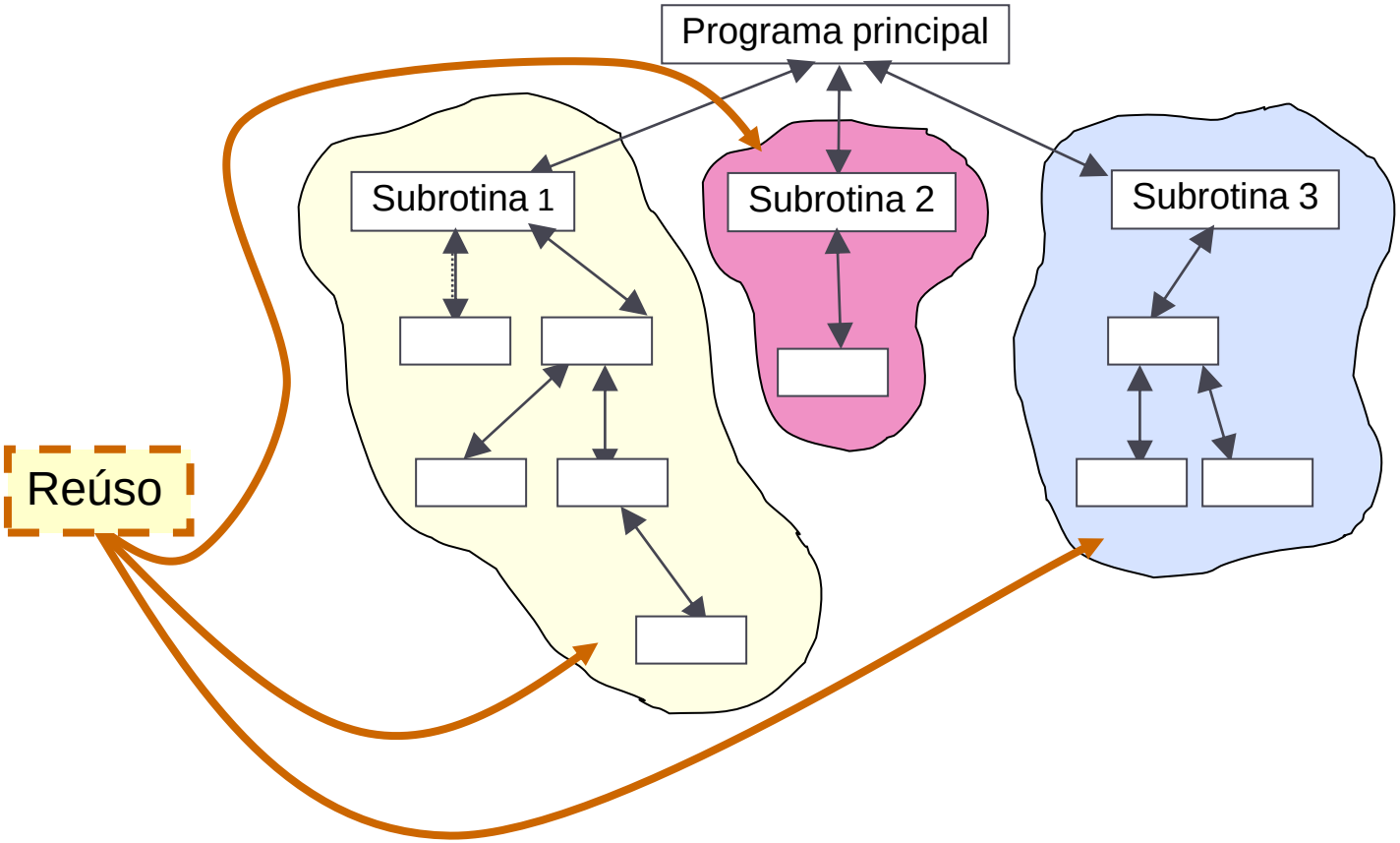
- Programa principal/Subrotina (*Main Program/Subroutine*)



Estilos e padrões arquiteturais

Invocação/Retorno (Call/Return)

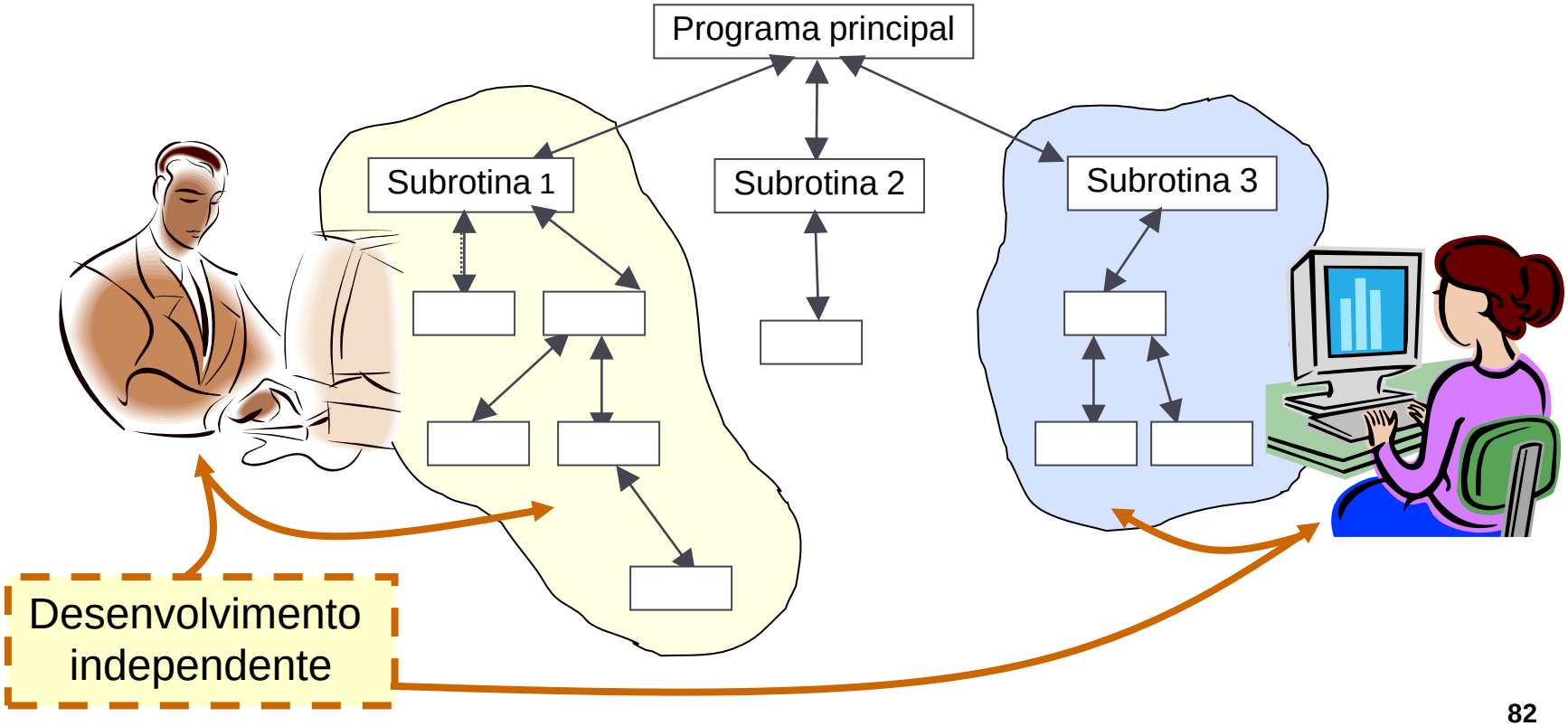
- Programa principal/Subrotina (*Main Program/Subroutine*)
  - **Objetivos**



Estilos e padrões arquiteturais

Invocação/Retorno (Call/Return)

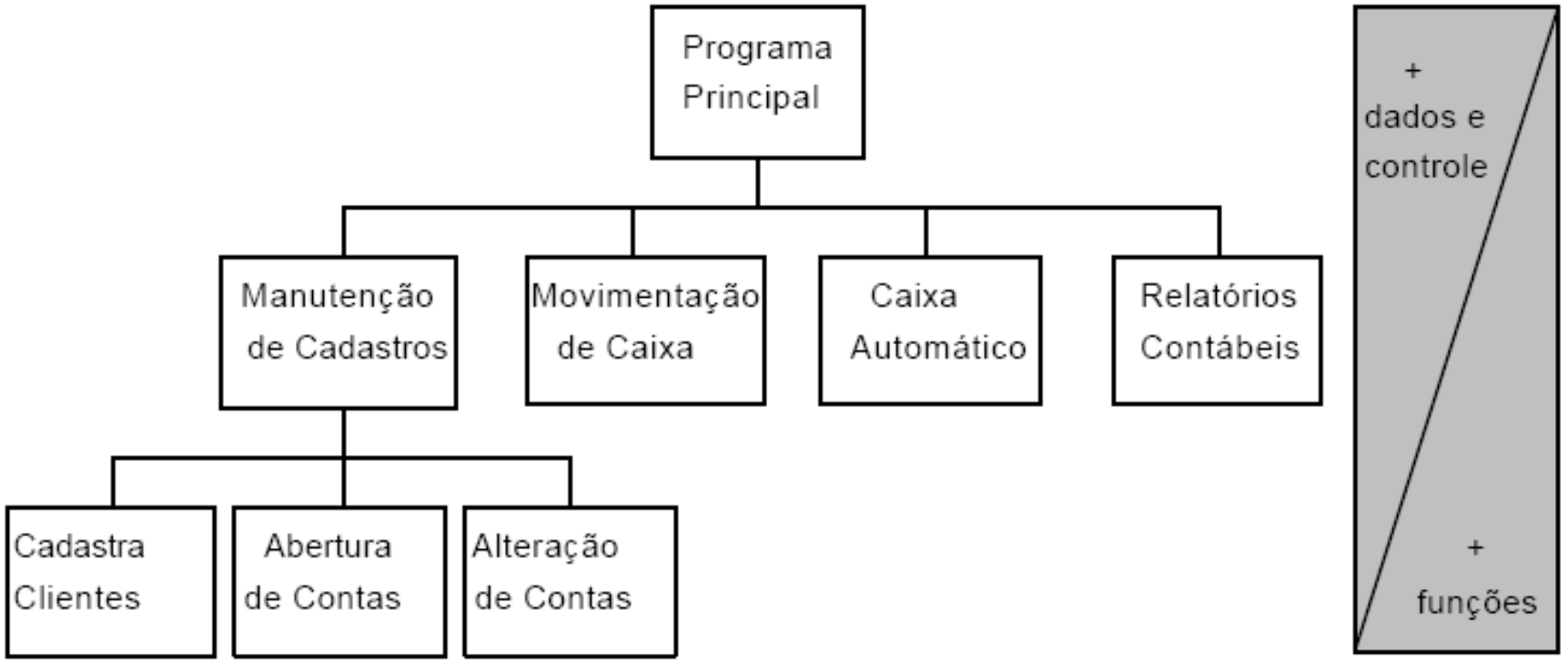
- Programa principal/Subrotina (*Main Program/Subroutine*)
  - **Objetivos**



Estilos e padrões arquiteturais

Invocação/Retorno (Call/Return)

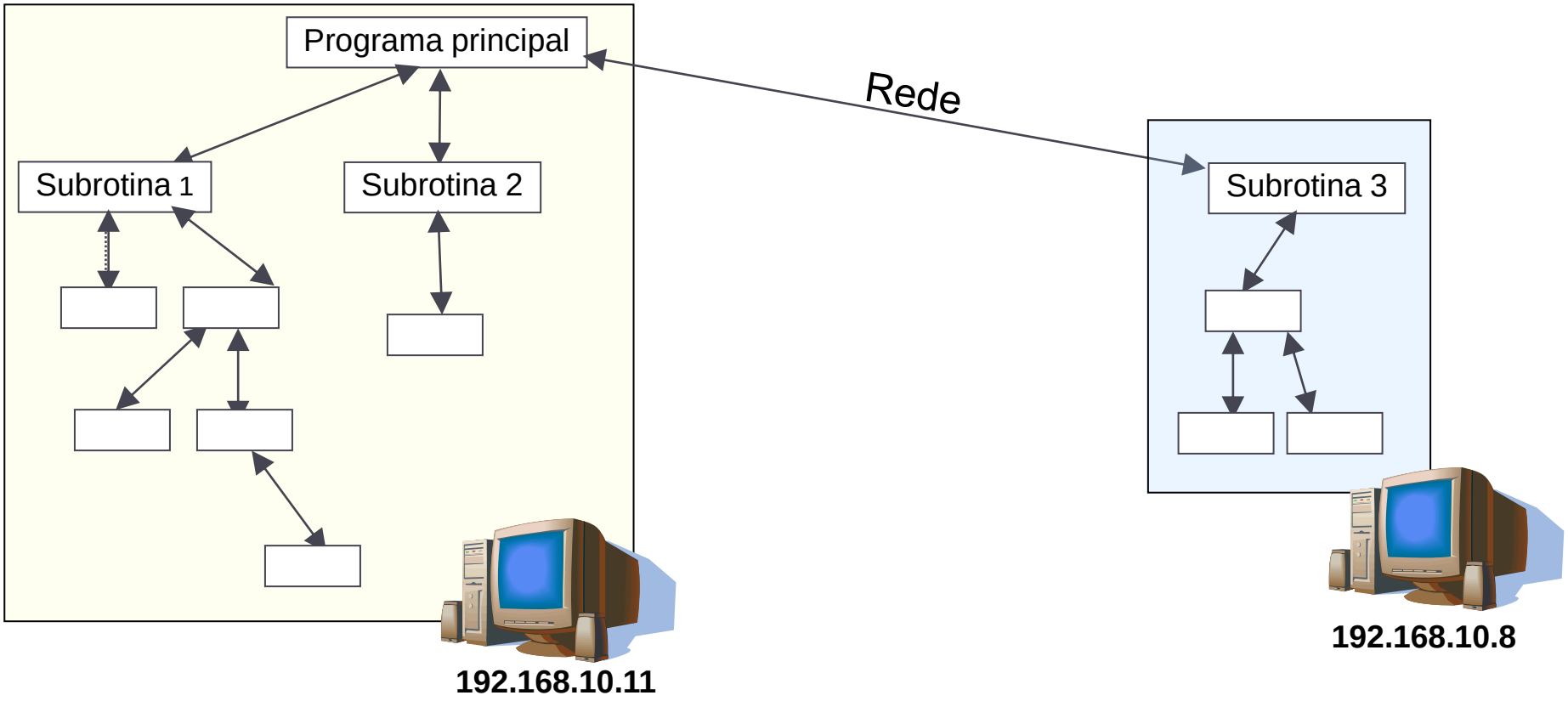
- Programa principal/Subrotina (*Main Program/Subroutine*)



Estilos e padrões arquiteturais

Invocação/Retorno (Call/Return)

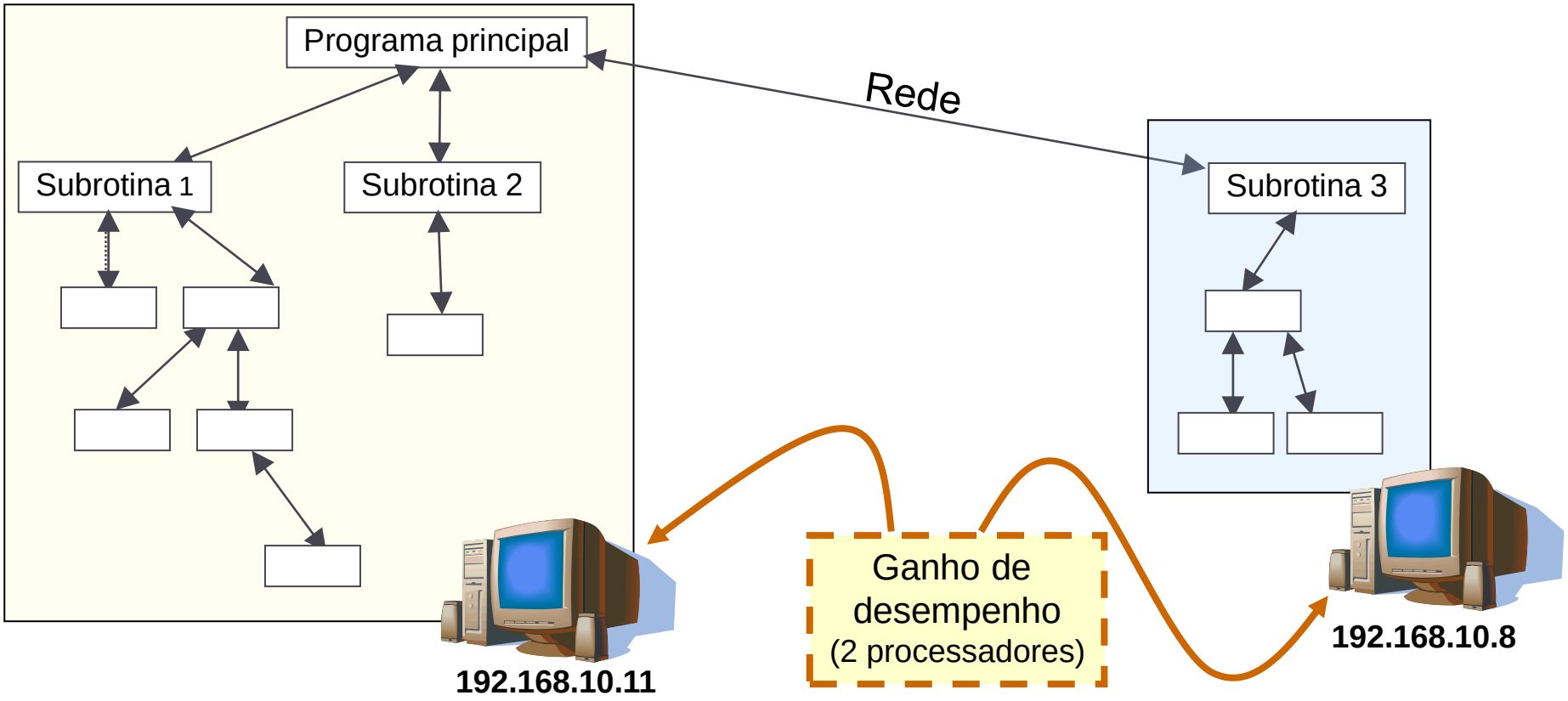
- Invocação remota de procedimento (*RPC*)



Estilos e padrões arquiteturais

Invocação/Retorno (Call/Return)

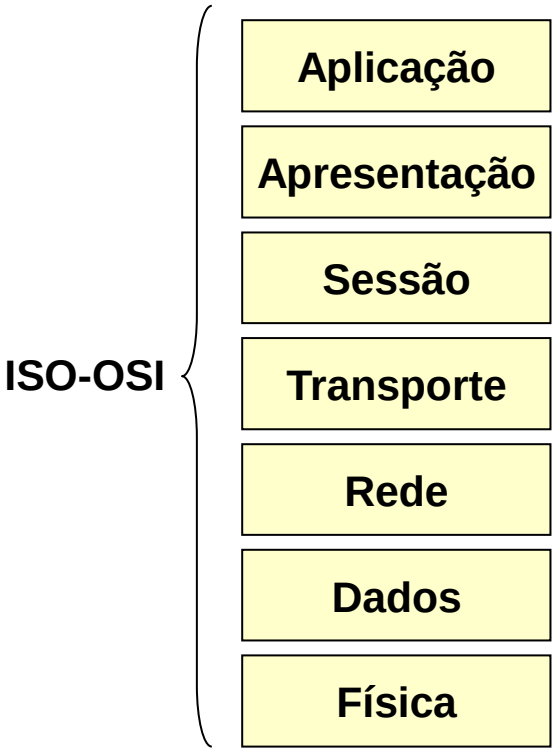
- Invocação remota de procedimento (*RPC*)

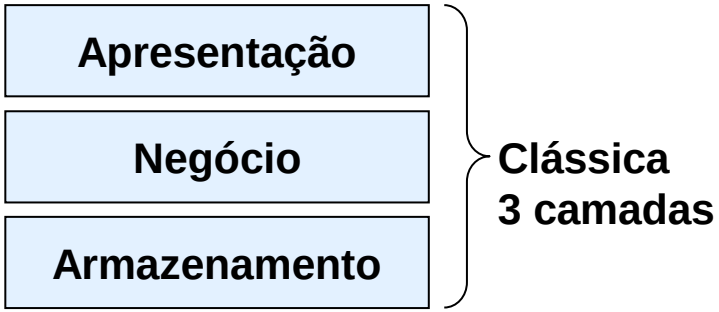


Estilos e padrões arquiteturais

Invocação/Retorno (Call/Return)

- Camadas (*Layered*)





Estilos e padrões arquiteturais

Invocação/Retorno (Call/Return)

- Camadas (*Layered*)
  - Camadas se comunicam apenas com outras adjacentes

Estilos e padrões arquiteturais

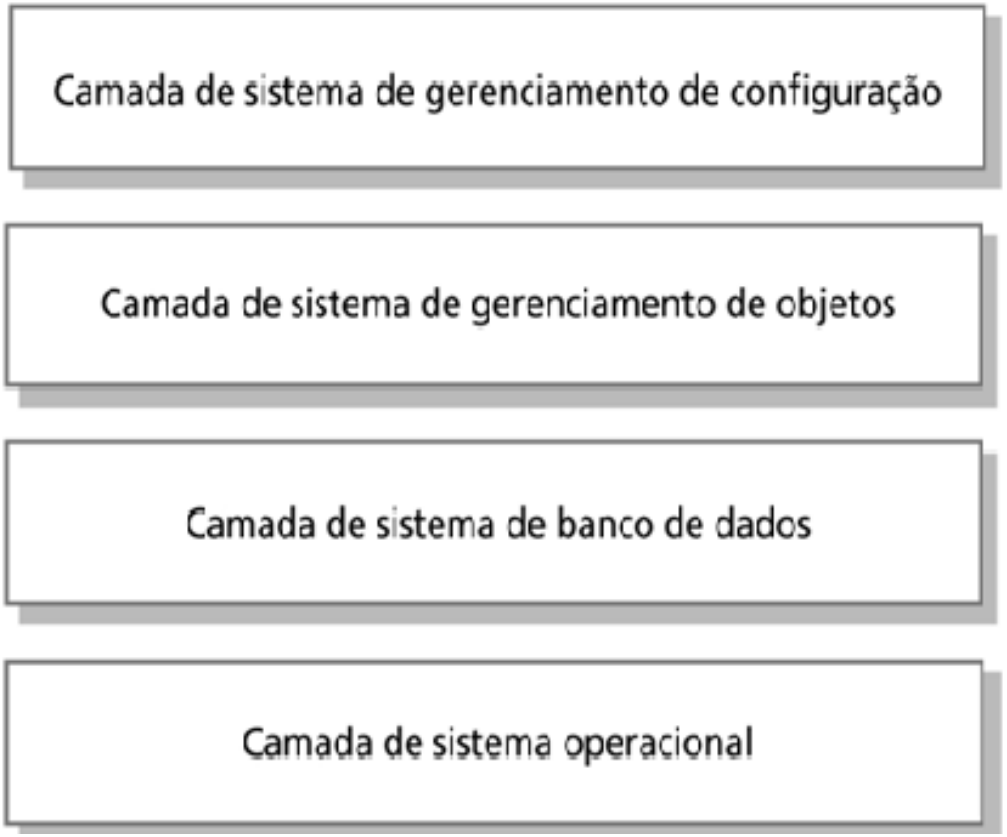
Invocação/Retorno (Call/Return)

- Camadas (*Layered*)
  - Alterações locais não são propagadas

Estilos e padrões arquiteturais

Invocação/Retorno (Call/Return)

- Exemplo: Sistema de gerenciamento de versões



Estilos e padrões arquiteturais

Classificação

Invocação/Retorno (*Call/Return*)

Programa principal/Subrotina (*Main Program/Subroutine*)
Invocação remota de procedimento (*Remote Procedure Call - RPC*)
Camadas (*Layered*)

Componentes independentes (*Independent Components*)

Comunicação de processos (*Communicating Processes*)
Baseado em eventos
Arquitetura orientada a serviços (SOA)

Máquina virtual (*Virtual Machine*)

Interpretador (*Interpreter*)
Baseado em regras (*Rule-based*)

Centrado em dados (*Data-Centered*)

Repositório (*Repository*)
Quadro negro (*Blackboard*)

Fluxo de dados (*Data-Flow*)

Seqüencial (*Batch Sequential*)
Tubos e filtros (*Pipe and Filter*)
Map-Reduce

Estilos e padrões arquiteturais

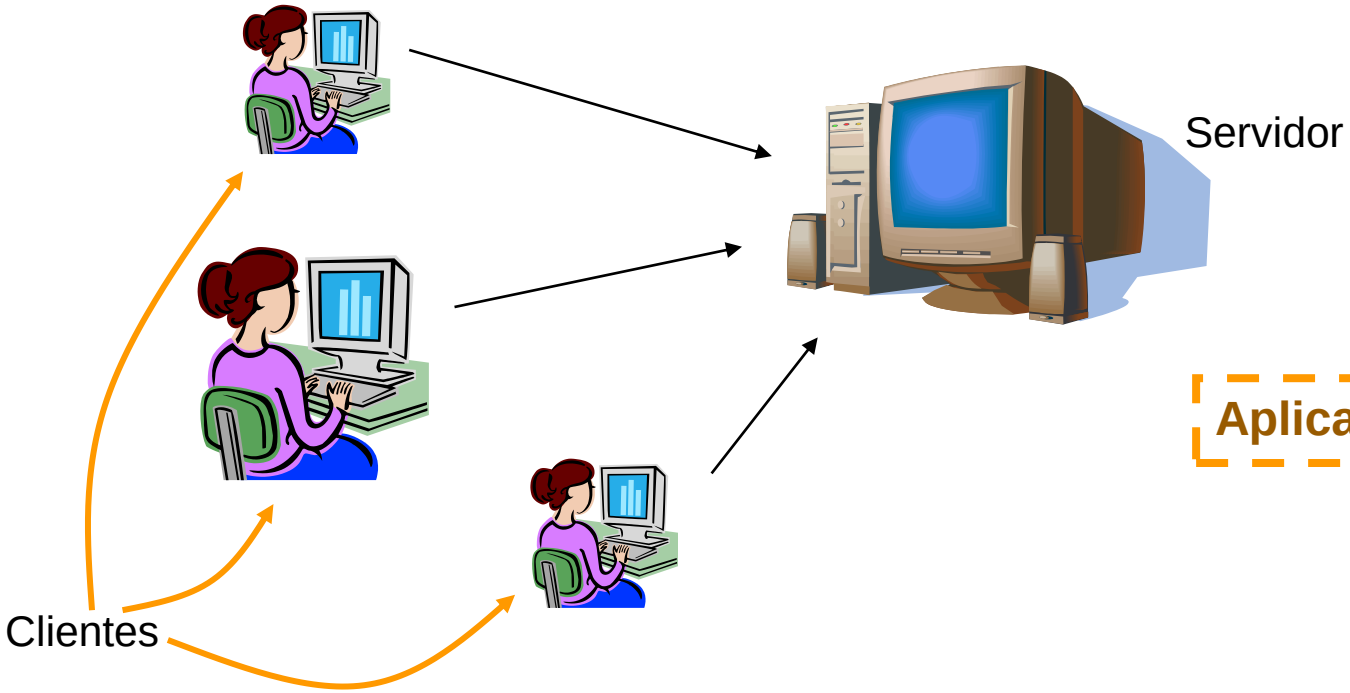
Componentes independentes

- Comunicação de processos (*Communicating Processes*)
  - Baseado na comunicação via troca de mensagens entre processos
    - Em geral, via rede
  - Cliente - Servidor
  - Ponto a ponto (*Peer to Peer* – P2P)

Estilos e padrões arquiteturais

Componentes independentes

- Comunicação de processos (*Communicating Processes*)
  - Cliente – Servidor

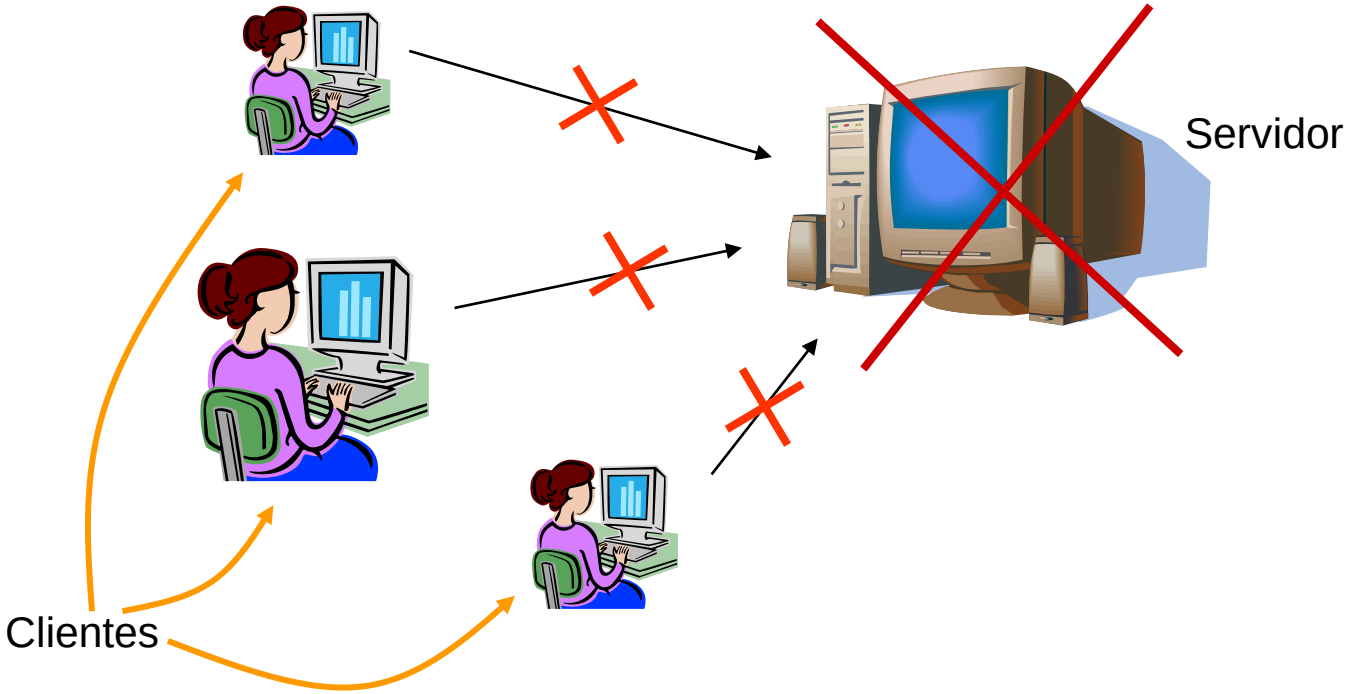


Aplicação: Internet

Estilos e padrões arquiteturais

Componentes independentes

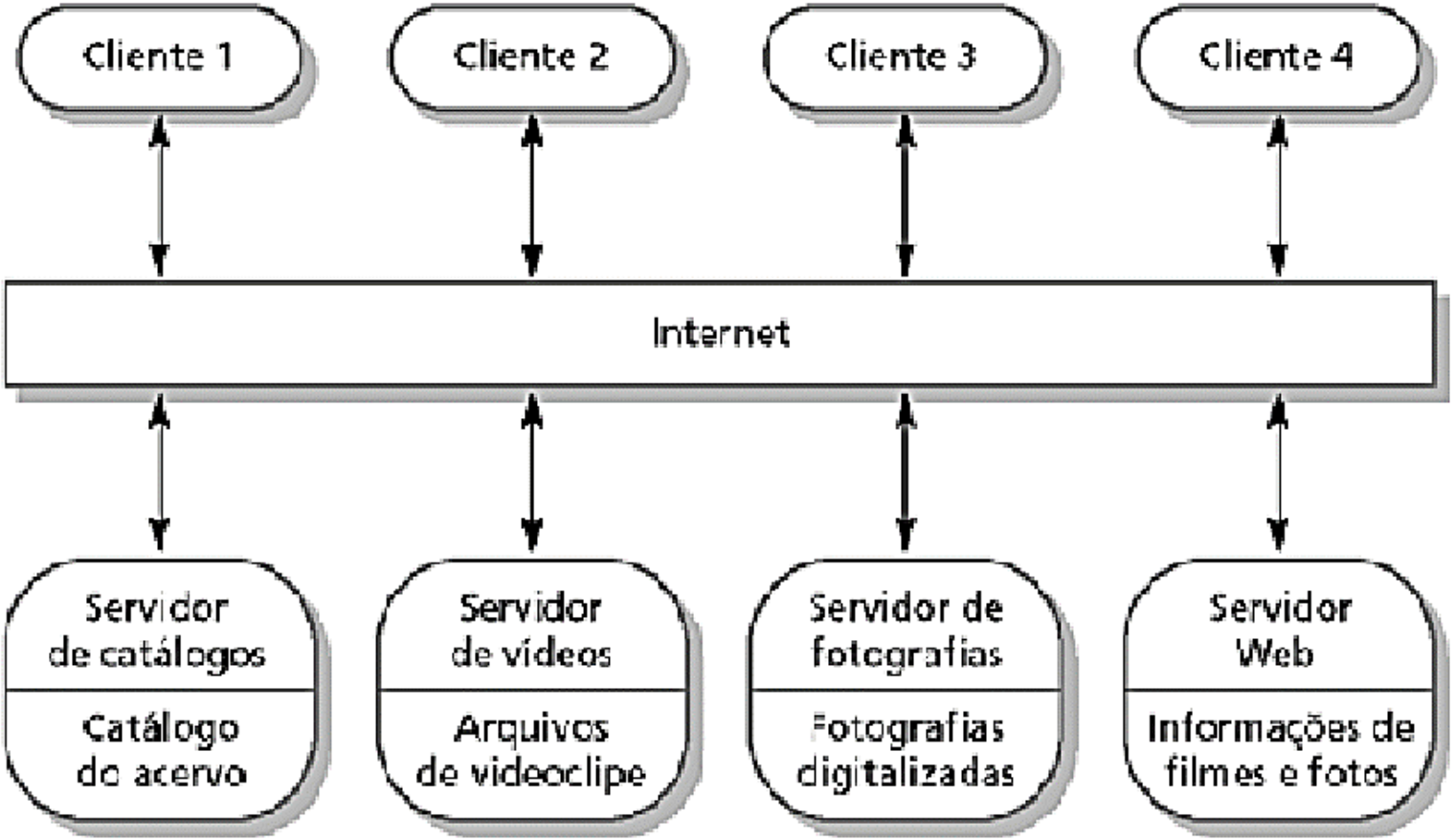
- Comunicação de processos (*Communicating Processes*)
  - Cliente – Servidor
    - Problema: servidor é ponto de falha!



Estilos e padrões arquiteturais

Componentes independentes

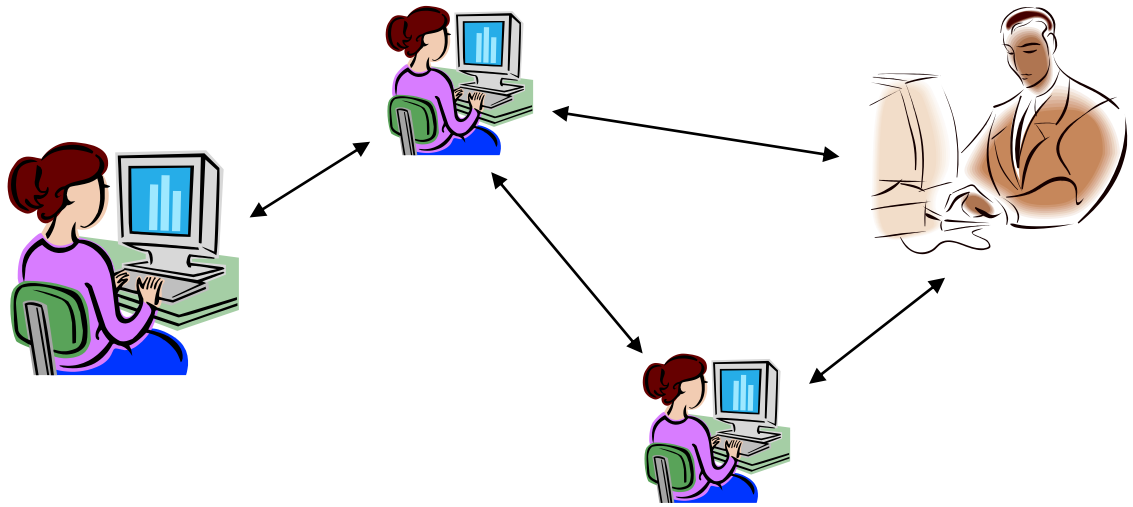
- Exemplo: Biblioteca de filmes e fotografias



Estilos e padrões arquiteturais

Componentes independentes

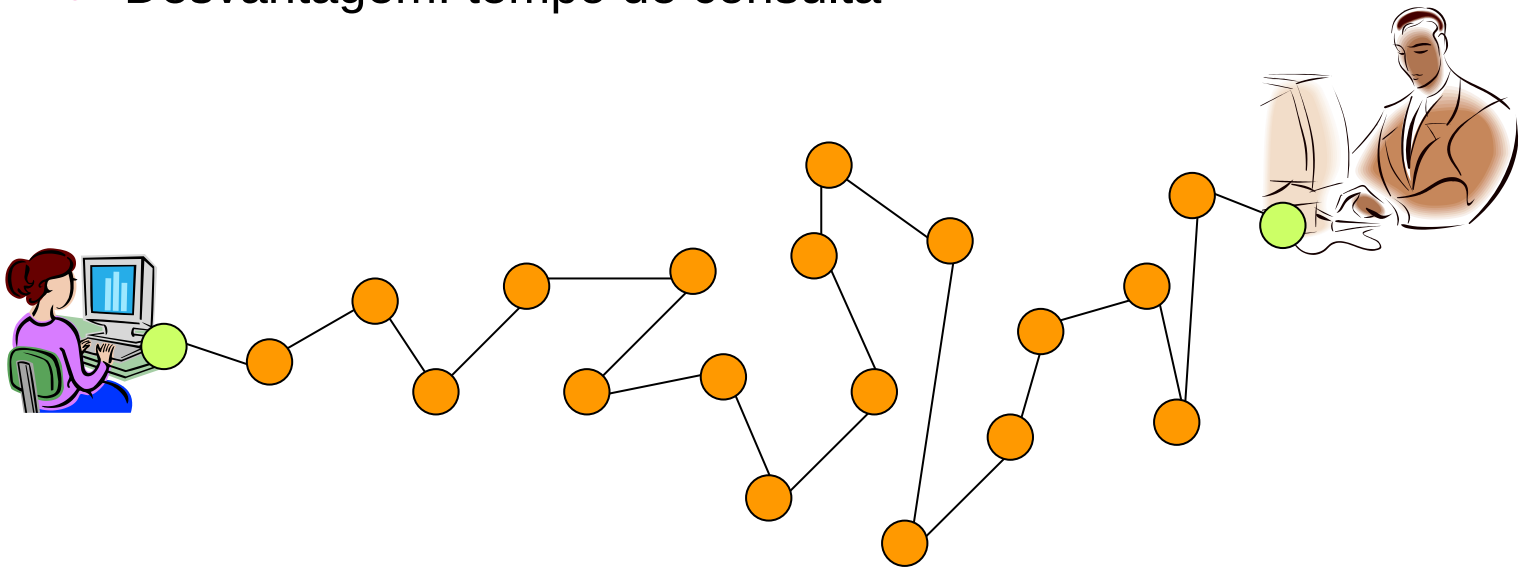
- Comunicação de processos (*Communicating Processes*)
  - Ponto a Ponto (P2P)
    - Não há distinção entre nós
    - Cada nó mantém seus próprios dados e endereços conhecidos
    - Cada nó é “cliente e servidor ao mesmo tempo”



Estilos e padrões arquiteturais

Componentes independentes

- Comunicação de processos (*Communicating Processes*)
  - Ponto a Ponto (P2P)
    - Vantagem: não há ponto de falha
    - Desvantagem: tempo de consulta



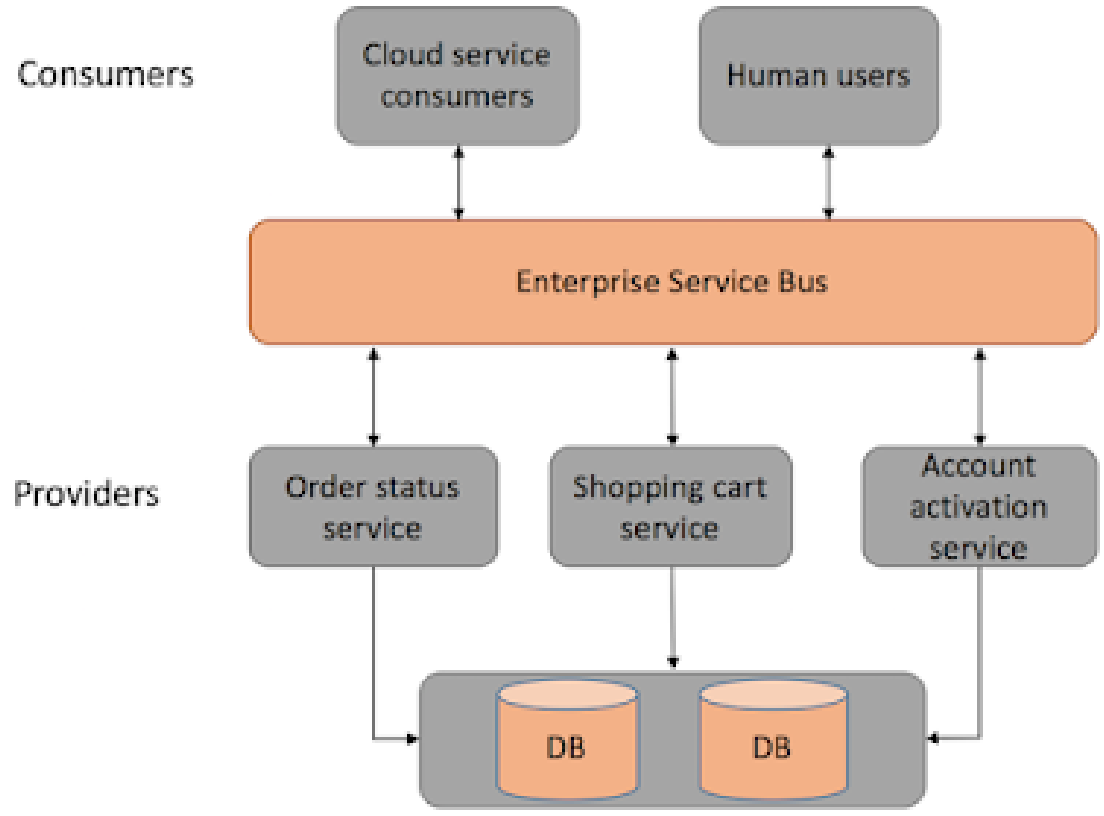
Estilos e padrões arquiteturais

Componentes independentes

- Arquitetura Orientada a Serviços (*Service-Oriented Architecture – SOA*)
  - Uma coleção de componentes distribuídos que fornecem e/ou consomem serviços
  - Componentes (consumidores ou fornecedores) podem usar diferentes linguagens de implementação e plataformas.
    - Cada componente possui uma interface que descreve os serviços que solicitam e os serviços que fornecem.

Estilos e padrões arquiteturais

Componentes independentes



- Fornecedor de serviço: fornecem um ou mais serviços por meio de uma interface. Podem ser consumidores também;
- Consumidor de serviço: solicita serviços diretamente ou por meio de um intermediário;
- ESB: elemento intermediário que gerencia as mensagens entre consumidores e fornecedores.

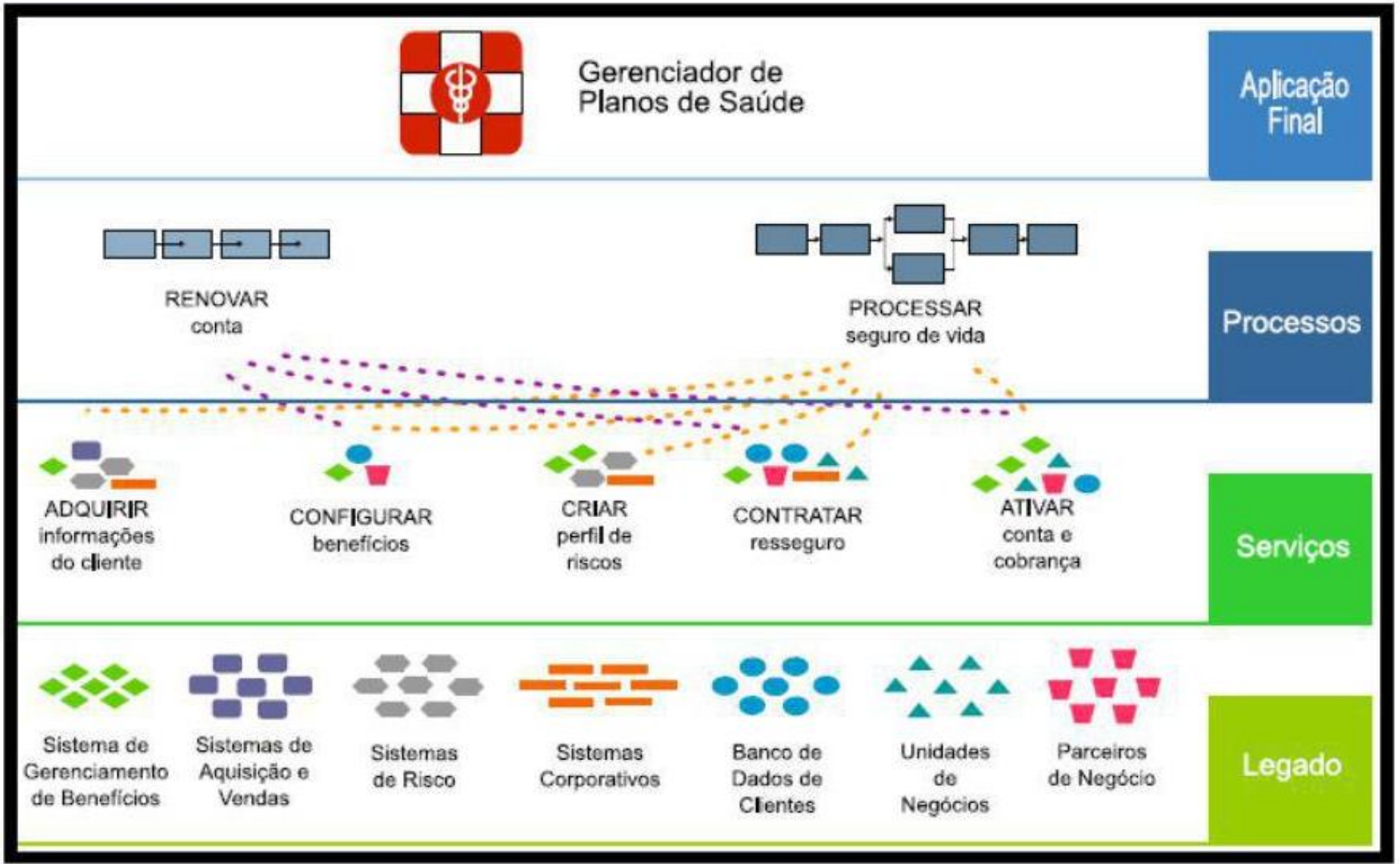
Estilos e padrões arquiteturais

Componentes independentes

- Arquitetura Orientada a Serviços (*Service-Oriented Architecture – SOA*)
  - Desafios:
    - Complexidade de design e implementação (ligação dinâmica)
    - Sobrecarga de desempenho no middleware
    - Os serviços podem ter atualizações inesperadas
  - Benefícios:
    - Interoperabilidade
    - Reconfiguração dinâmica

Estilos e padrões arquiteturais

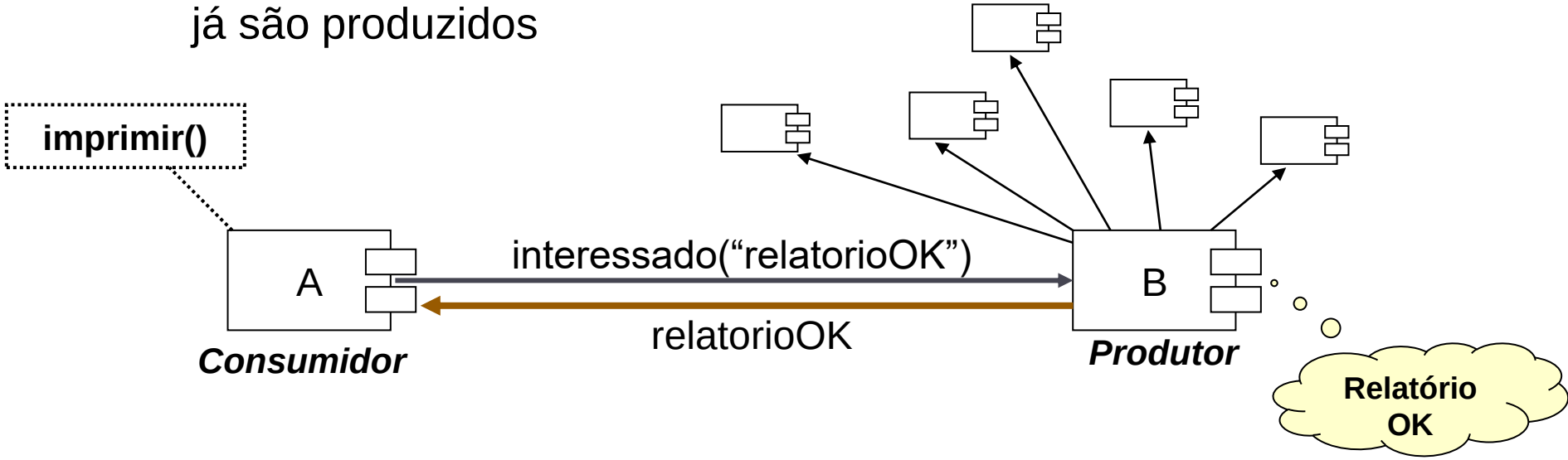
Componentes independentes



Estilos e padrões arquiteturais

Componentes independentes

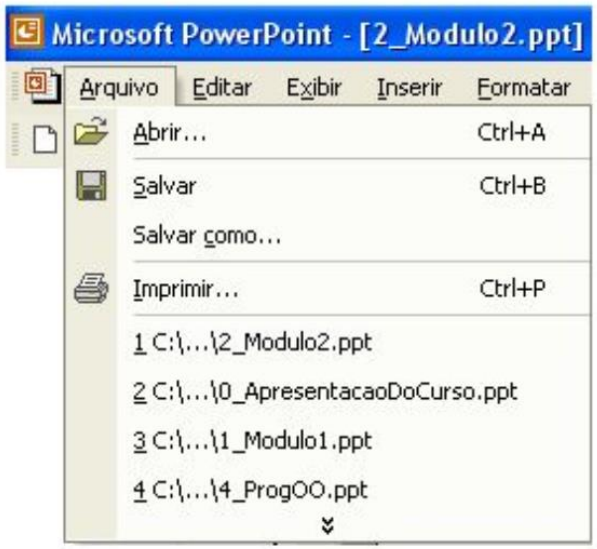
- Baseado em eventos
  - Produtores e consumidores de eventos
  - Consumidores se registram nos Produtores
  - Produtores notificam consumidores registrados
  - Escalabilidade: adição de novos observadores para eventos que já são produzidos



Estilos e padrões arquiteturais

Componentes independentes

- Aplicação comum
  - Interface gráfica



Estilos e padrões arquiteturais

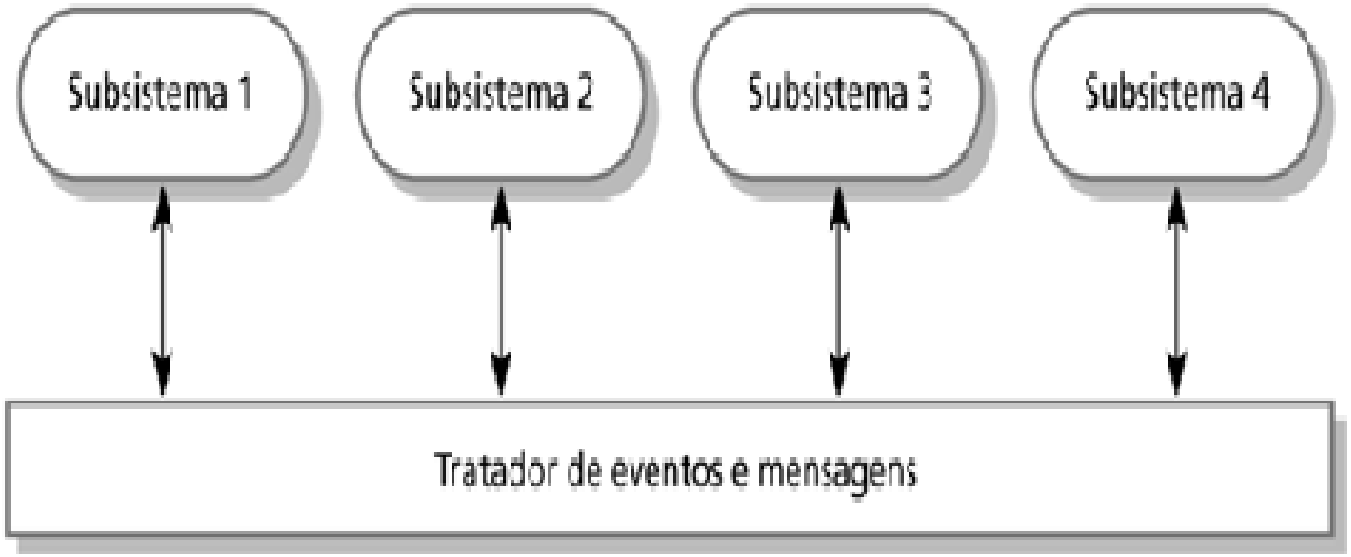
Componentes independentes

- Baseado em eventos
  - Produtores e consumidores são independentes
  - Execução via procedimentos disparados via mudança de estados
  - Escalabilidade no número de interessados
  - Publisher / Subscriber
    - Eventos são transmitidos a todos os componentes.
    - Qualquer componente interessado pode respondê-los

Estilos e padrões arquiteturais

Componentes independentes

- Exemplo:



Estilos e padrões arquiteturais

Classificação

Invocação/Retorno (*Call/Return*)

- Programa principal/Subrotina (*Main Program/Subroutine*)
- Invocação remota de procedimento (*Remote Procedure Call - RPC*)
- Camadas (*Layered*)

Componentes independentes (*Independent Components*)

- Comunicação de processos (*Communicating Processes*)
- Baseado em eventos
- Arquitetura orientada a serviços (SOA)

Máquina virtual (*Virtual Machine*)

- Interpretador (*Interpreter*)
- Baseado em regras (*Rule-based*)

Centrado em dados (*Data-Centered*)

- Repositório (*Repository*)
- Quadro negro (*Blackboard*)

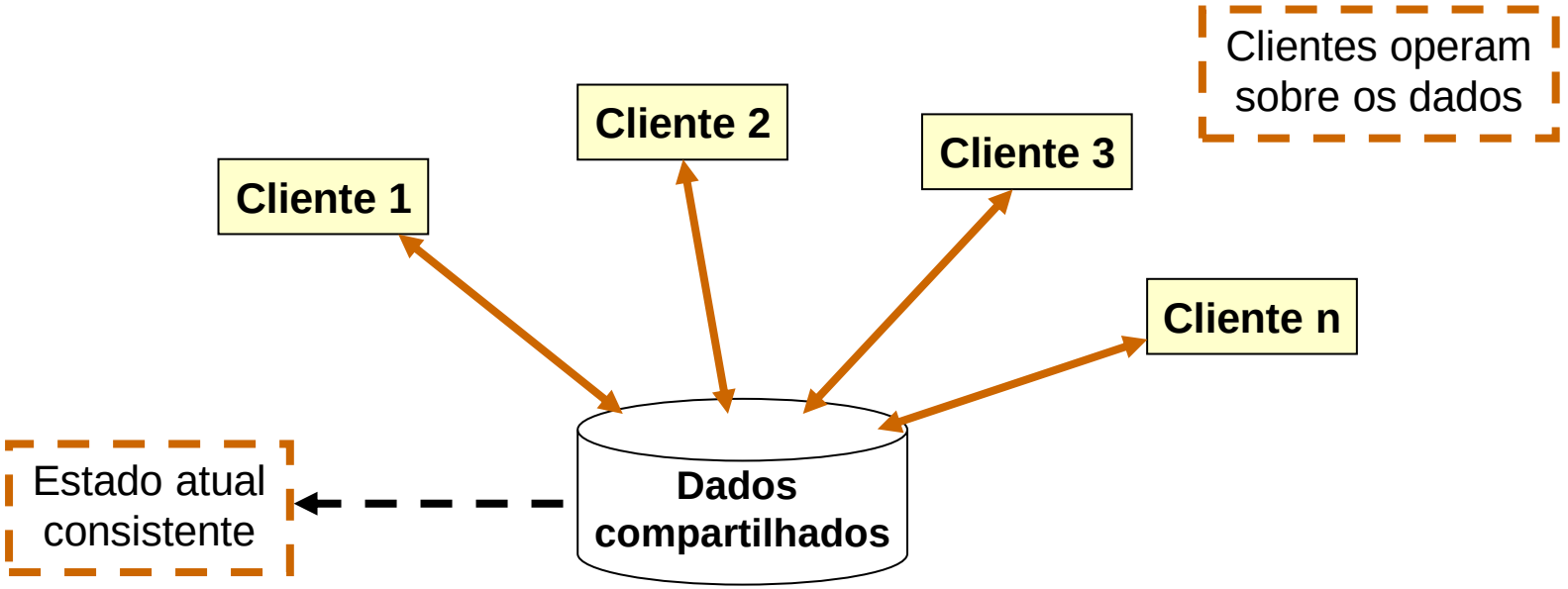
Fluxo de dados (*Data-Flow*)

- Seqüencial (*Batch Sequential*)
- Tubos e filtros (*Pipe and Filter*)
- *Map-Reduce*

Estilos e padrões arquiteturais

Centrado em dados (Data-centered)

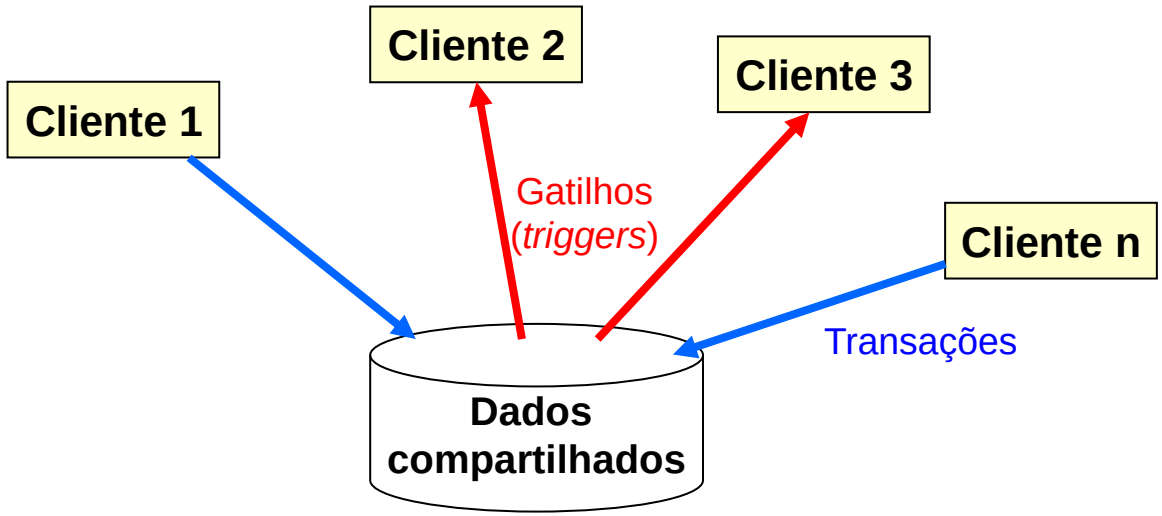
- Repositório (*Repository*)
  - Integridade, escalabilidade (novos clientes, novos dados)



Estilos e padrões arquiteturais

Centrado em dados (Data-centered)

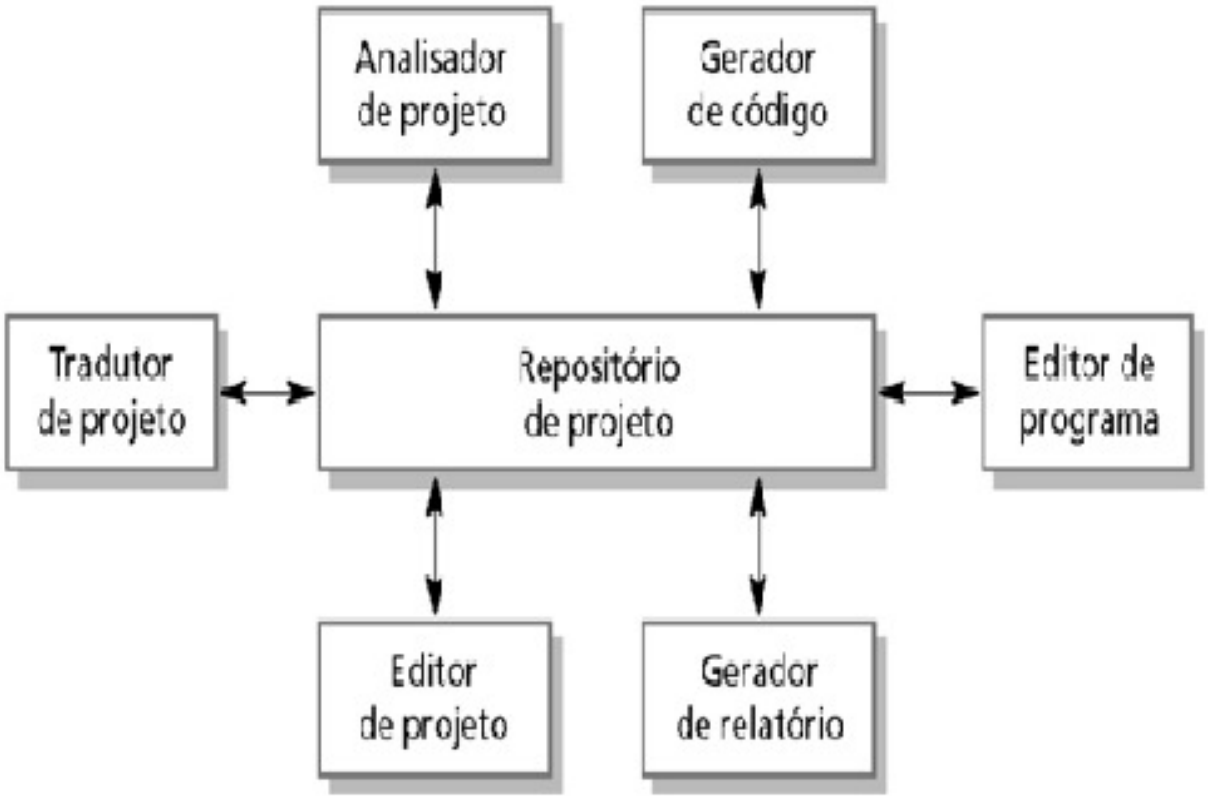
- Repositório (*Repository*)
  - Exemplo: banco de dados tradicional



Estilos e padrões arquiteturais

Centrado em dados (Data-centered)

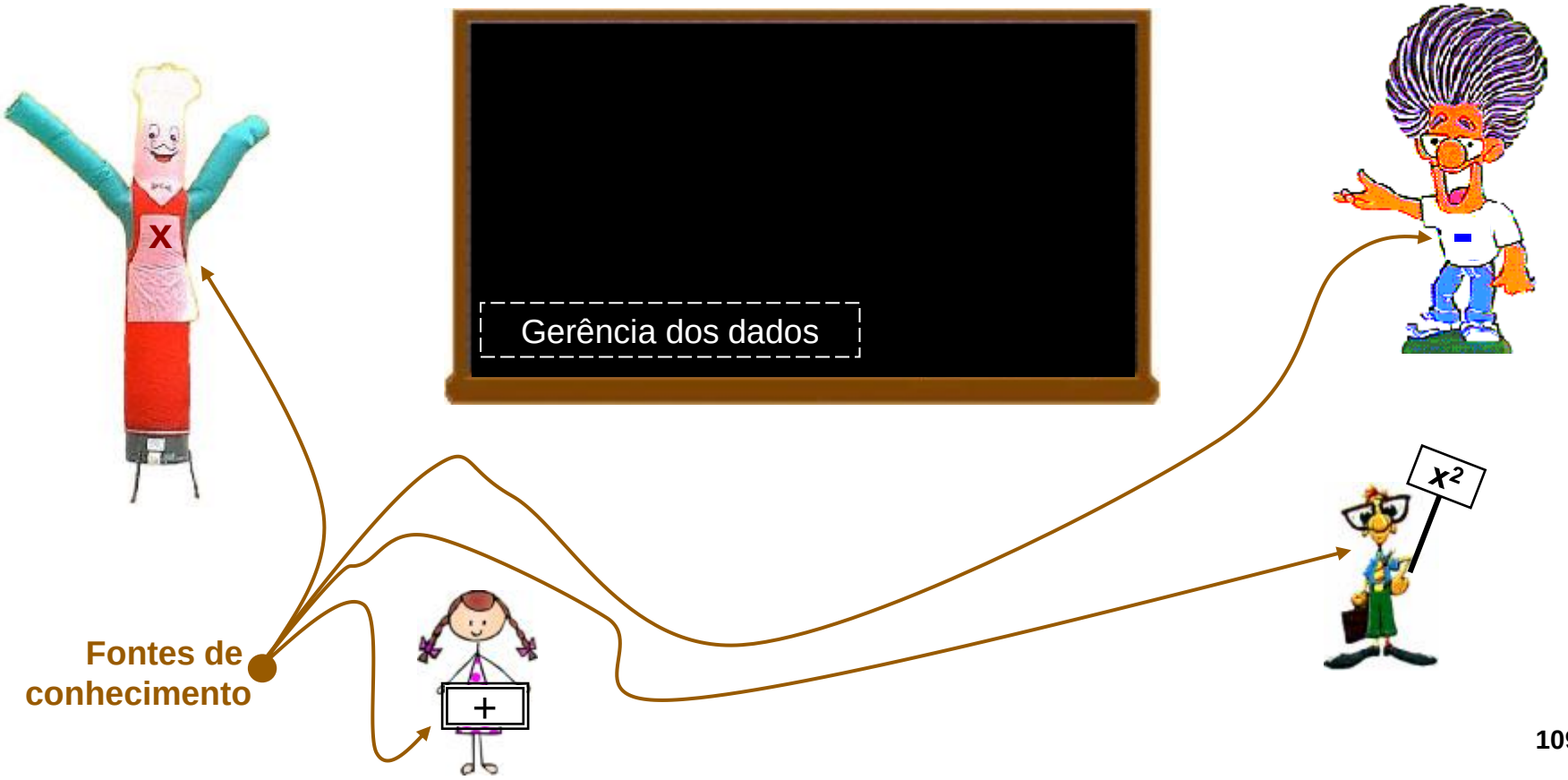
- Exemplos: Arquitetura de uma ferramenta case



Estilos e padrões arquiteturais

Centrado em dados (Data-centered)

- Quadro negro (*Blackboard*)



Estilos e padrões arquiteturais

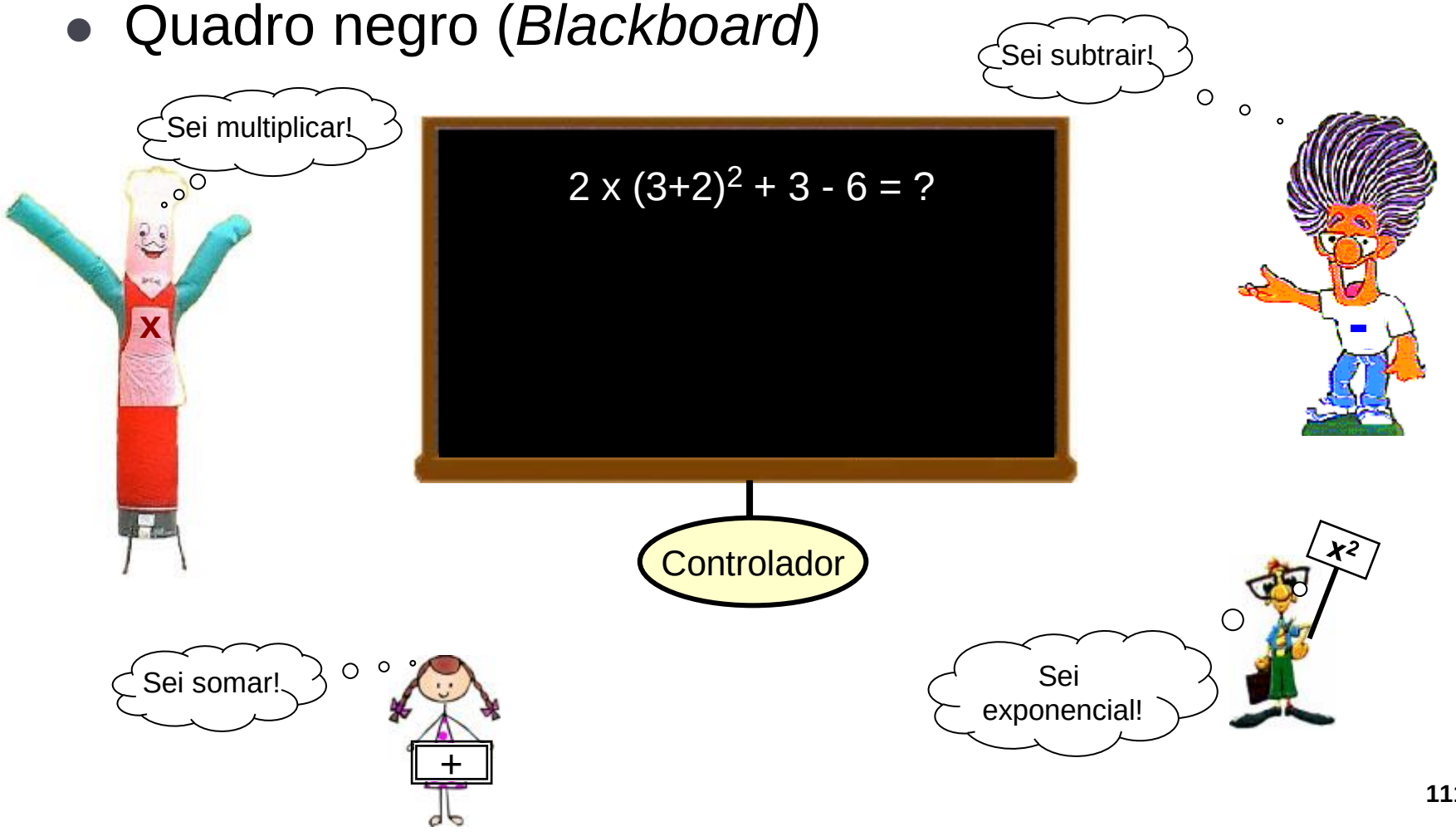
Centrado em dados (Data-centered)

- Quadro negro (*Blackboard*)
  - O sistema é dividido em
    - blackboard: aramazena dados – o vocabulário
    - base de conhecimento: subsistemas independentes, cada qual resolvendo aspectos específicos do problema
    - componente de controle: monitora mudanças no blackboard e decide as ações

Estilos e padrões arquiteturais

Centrado em dados (Data-centered)

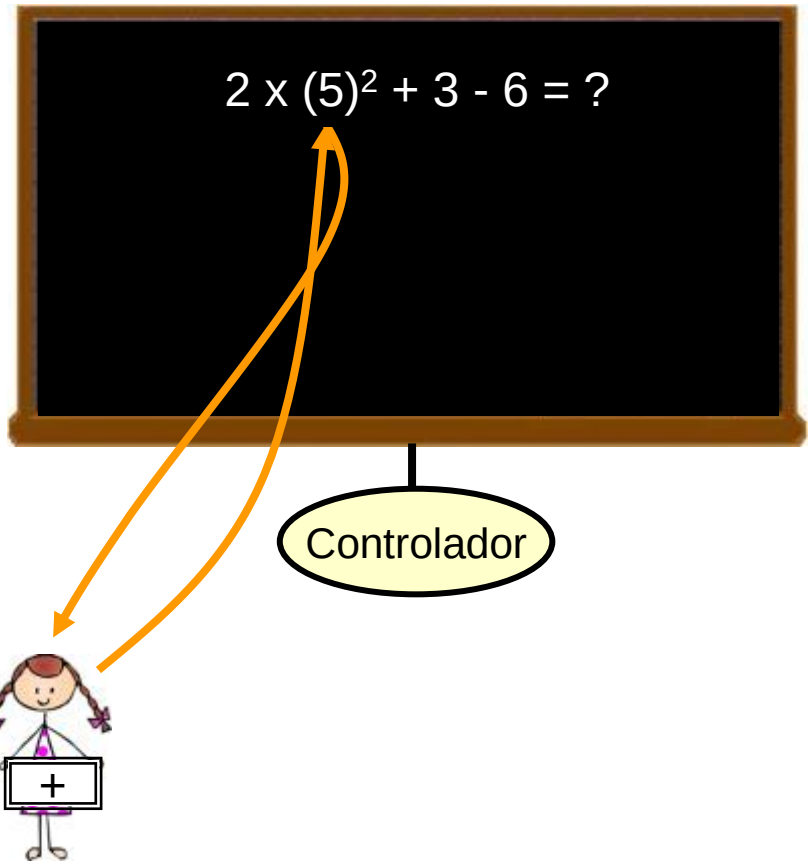
- Quadro negro (*Blackboard*)



Estilos e padrões arquiteturais

Centrado em dados (Data-centered)

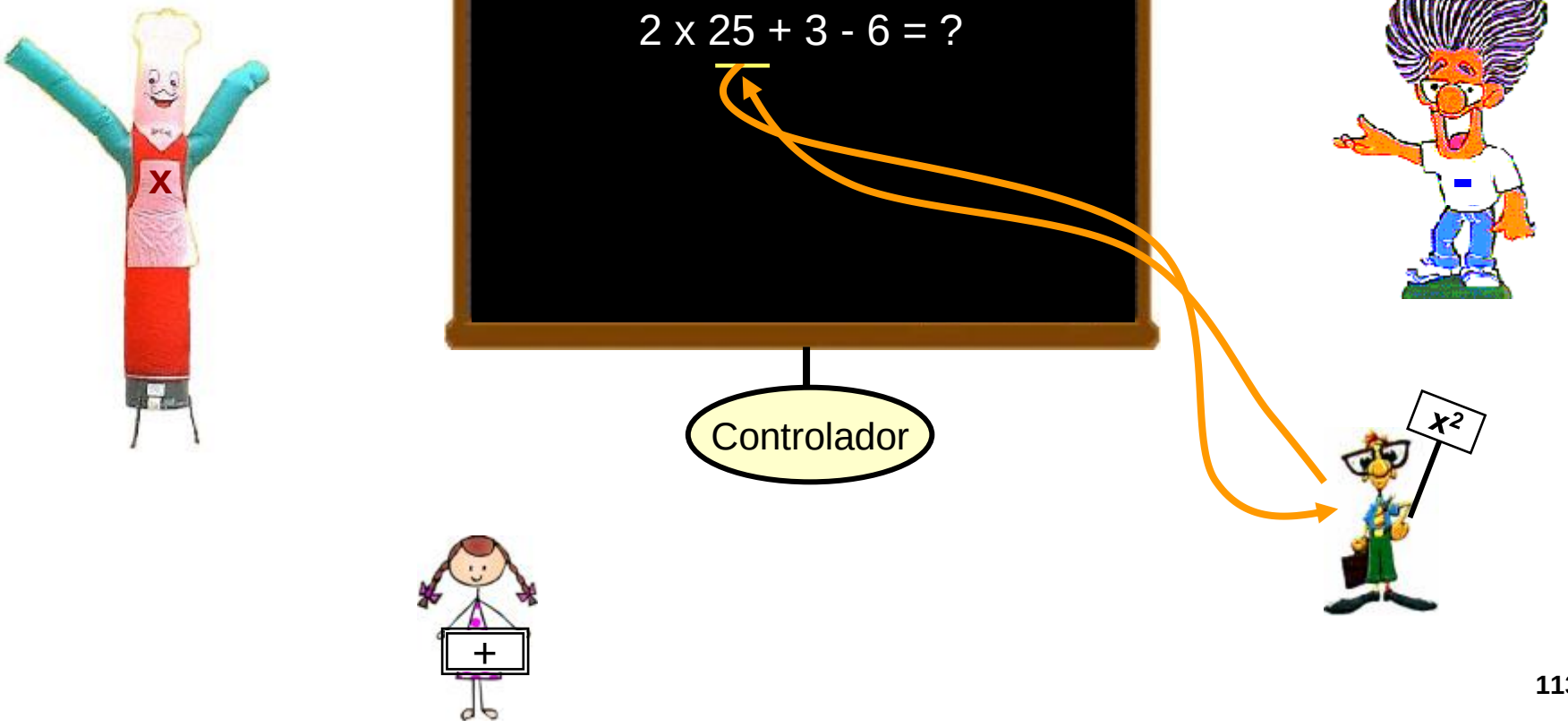
- Quadro negro (*Blackboard*)



Estilos e padrões arquiteturais

Centrado em dados (Data-centered)

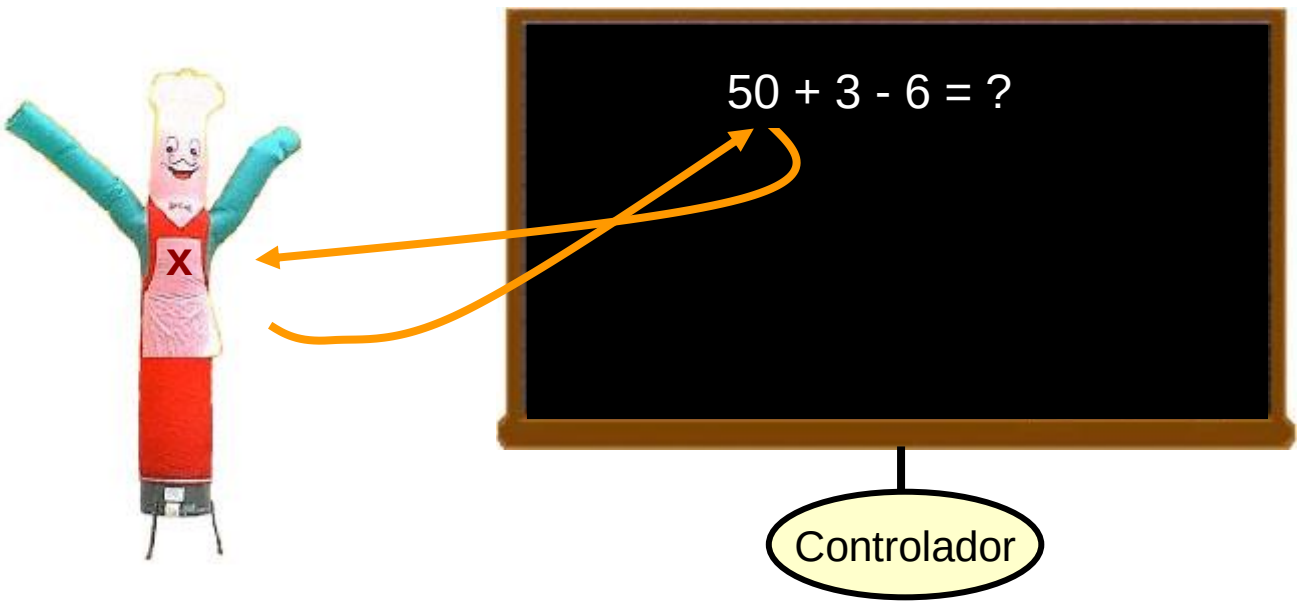
- Quadro negro (*Blackboard*)



Estilos e padrões arquiteturais

Centrado em dados (Data-centered)

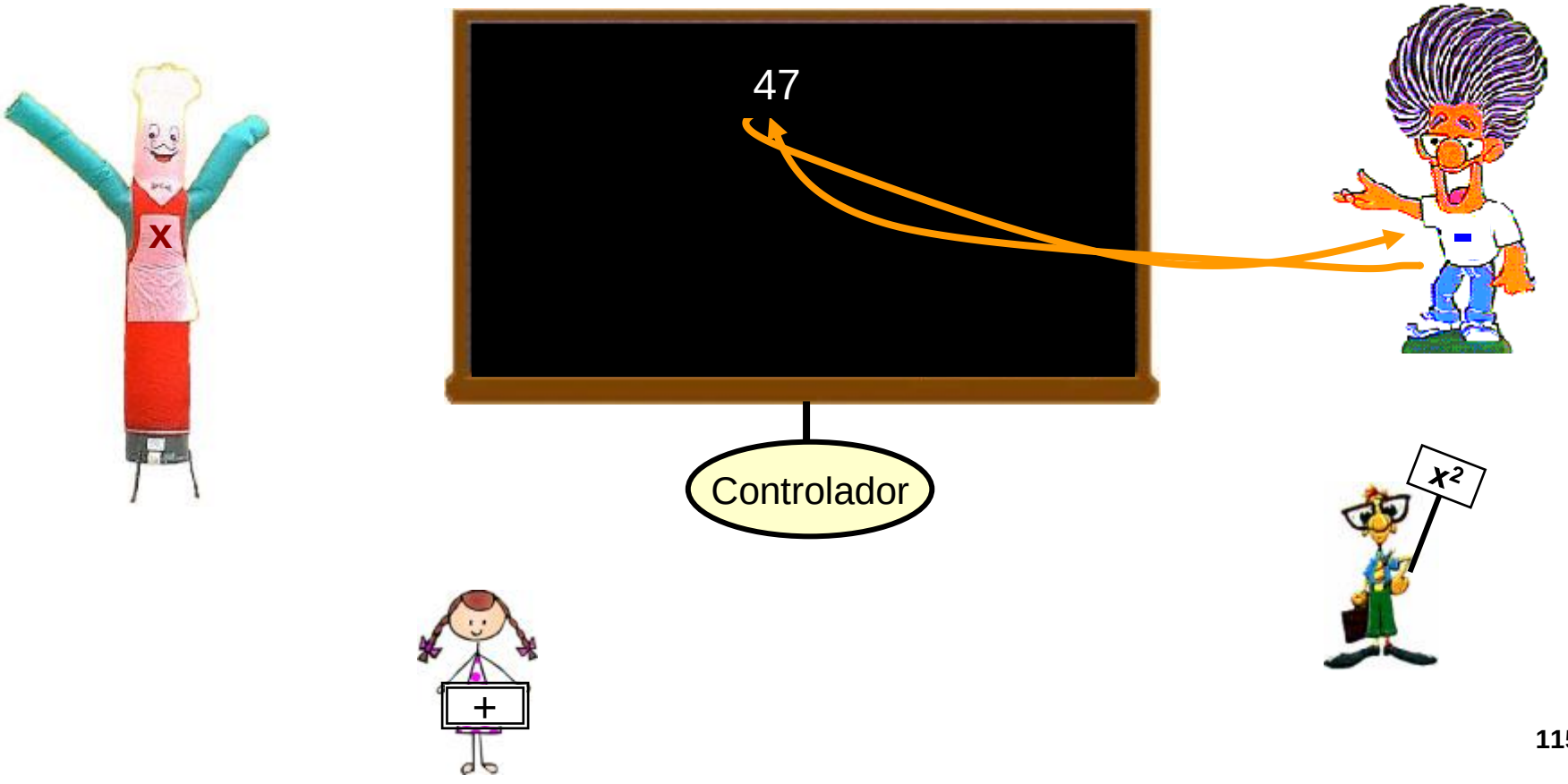
- Quadro negro (*Blackboard*)



Estilos e padrões arquiteturais

Centrado em dados (Data-centered)

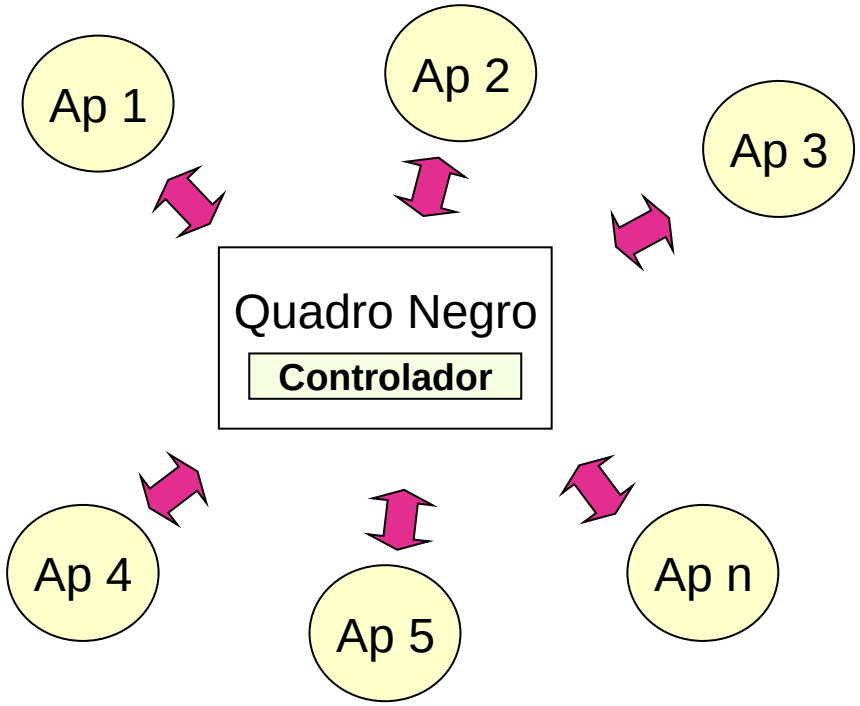
- Quadro negro (*Blackboard*)



Estilos e padrões arquiteturais

Centrado em dados (Data-centered)

- Quadro negro (*Blackboard*)



Estilos e padrões arquiteturais

Centrado em dados (Data-centered)

- Quadro negro (*Blackboard*)
  - Sistemas complexos
    - Resolução Distribuída de Problemas - RDP
  - Aplicações independentes
    - Escalabilidade
  - Ponto de falha!!!
    - Quadro negro
  - Arquitetura usada no paradigma de *agentes*

Estilos e padrões arquiteturais

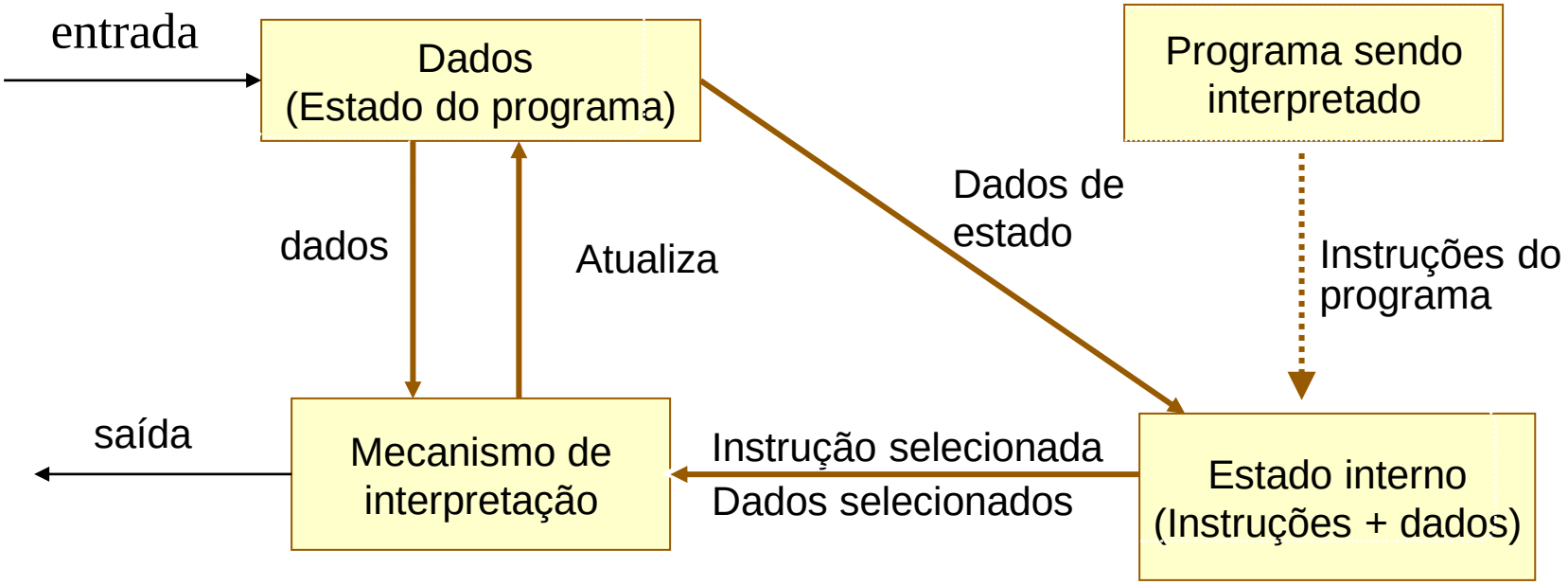
Classificação

- Invocação/Retorno (*Call/Return*)
  - Programa principal/Subrotina (*Main Program/Subroutine*)
  - Invocação remota de procedimento (*Remote Procedure Call - RPC*)
  - Camadas (*Layered*)
- Componentes independentes (*Independent Components*)
  - Comunicação de processos (*Communicating Processes*)
  - Baseado em eventos
  - Arquitetura orientada a serviços (SOA)
- Máquina virtual (*Virtual Machine*)
  - Interpretador (*Interpreter*)
  - Baseado em regras (*Rule-based*)
- Centrado em dados (*Data-Centered*)
  - Repositório (*Repository*)
  - Quadro negro (*Blackboard*)
- Fluxo de dados (*Data-Flow*)
  - Seqüencial (*Batch Sequential*)
  - Tubos e filtros (*Pipe and Filter*)
  - *Map-Reduce*

Estilos e padrões arquiteturais

Máquina virtual (Virtual Machine)

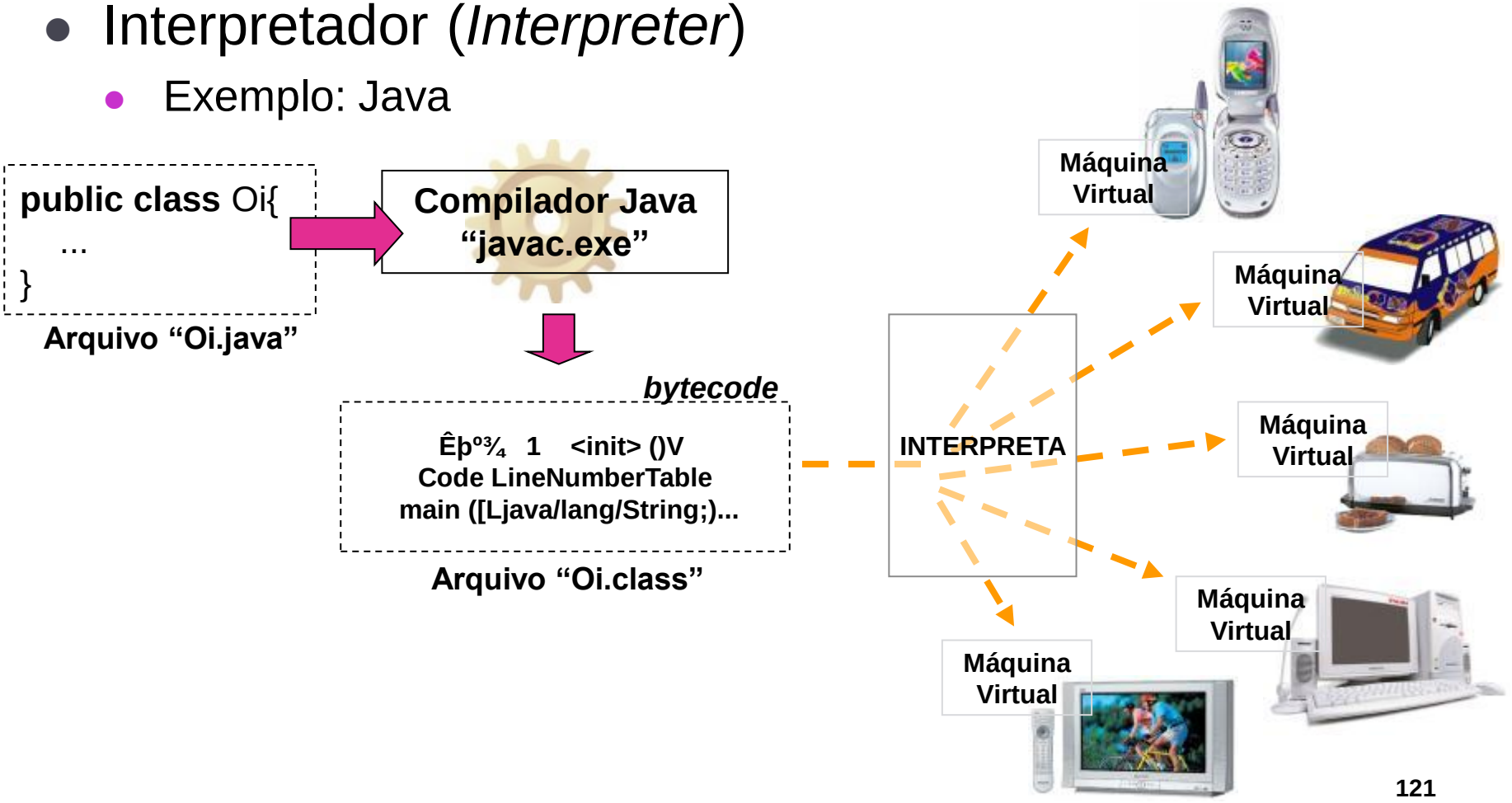
- Interpretador (*Interpreter*)
  - Simular funcionalidade não nativa para obter portabilidade



Estilos e padrões arquiteturais

Máquina virtual (Virtual Machine)

- Interpretador (*Interpreter*)
  - Exemplo: Java



Estilos e padrões arquiteturais

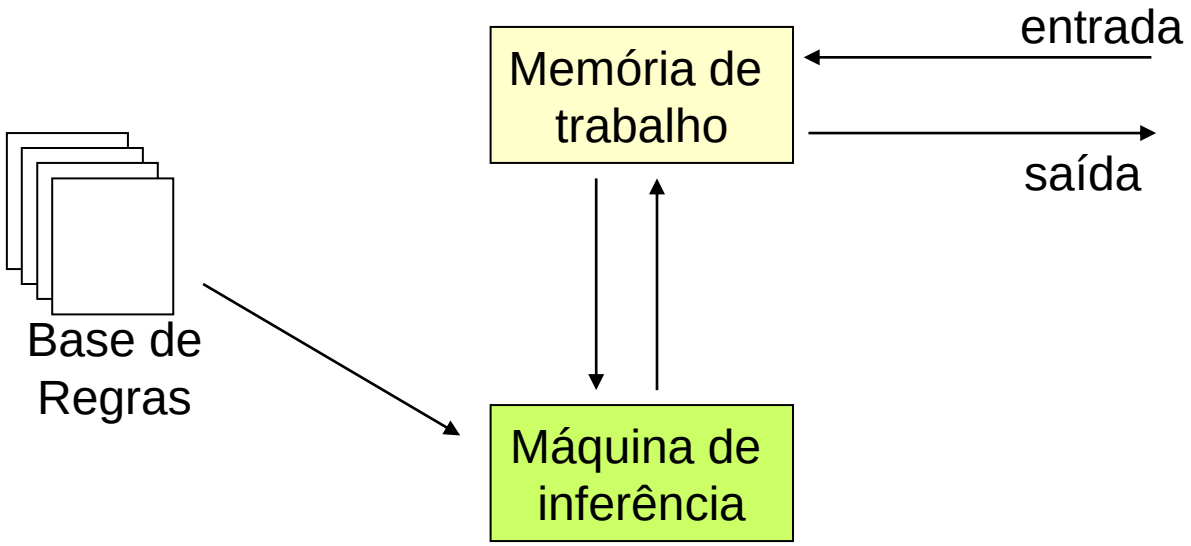
Máquina virtual (Virtual Machine)

- Interpretador (*Interpreter*)
  - Problema
    - Desempenho
  - Algumas pesquisas apontam que algumas das linguagens interpretadas já conseguem ser mais rápidas que C
    - Java, por exemplo
  - Máquina virtual nativa – Intel®

Estilos e padrões arquiteturais

Máquina virtual (Virtual Machine)

- Baseado em regras (*Rule-based*)
  - Conjunto de regras sobre um estado
  - Definição do estado atual com base em dados de entrada
  - Regras alteram o estado



Estilos e padrões arquiteturais

Máquina virtual (Virtual Machine)

- Baseado em regras (*Rule-based*)
  - Exemplos: Prolog, Sistemas Especialistas

SE “HORA=21:00”
ENTÃO “AÇÃO=LANCHE”

SE “HORA=22:00”
ENTÃO “AÇÃO=LIBERAR”

SE “HORA<19:00”
ENTÃO “AÇÃO=ESPERAR”

SE “HORA=19:00”
ENTÃO “AÇÃO=COMEÇAR”

Base de Regras

Memória de trabalho

HORA = ?
AÇÃO = ?

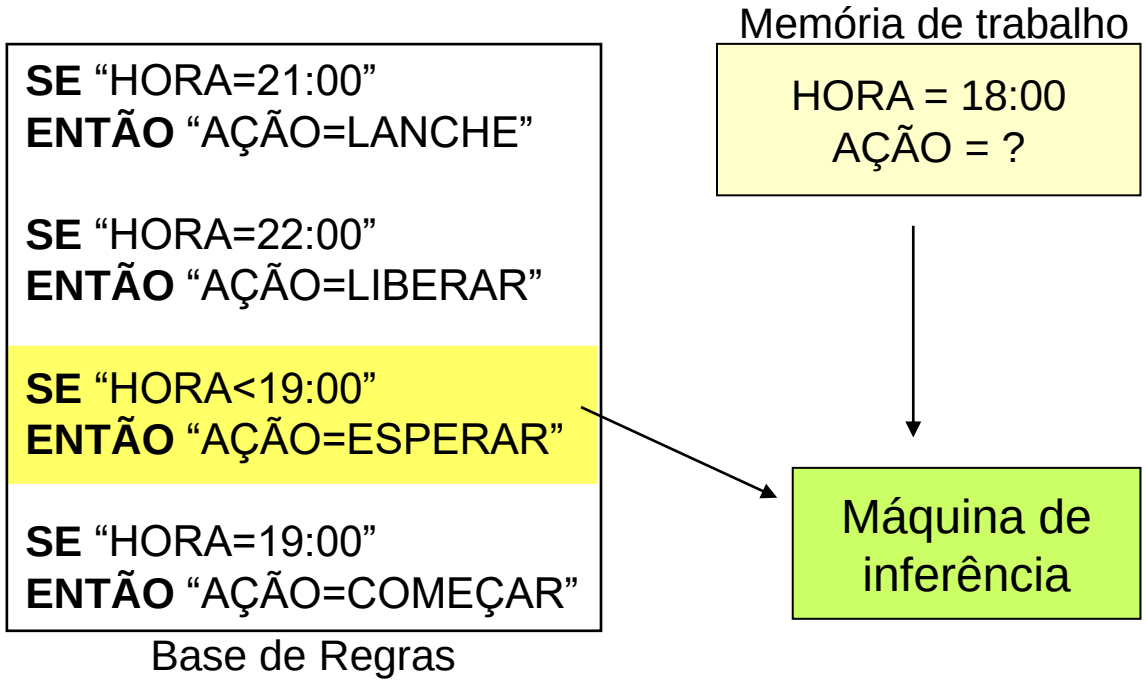
HORA=18:00

Máquina de inferência

Estilos e padrões arquiteturais

Máquina virtual (Virtual Machine)

- Baseado em regras (*Rule-based*)
  - Exemplos: Prolog, Sistemas Especialistas



Estilos e padrões arquiteturais

Máquina virtual (Virtual Machine)

- Baseado em regras (*Rule-based*)
  - Exemplos: Prolog, Sistemas Especialistas

SE “HORA=21:00”
ENTÃO “AÇÃO=LANCHE”

SE “HORA=22:00”
ENTÃO “AÇÃO=LIBERAR”

SE “HORA<19:00”
ENTÃO “AÇÃO=ESPERAR”

SE “HORA=19:00”
ENTÃO “AÇÃO=COMEÇAR”

Base de Regras

Memória de trabalho

HORA = 18:00
AÇÃO = ESPERAR

Máquina de inferência

Estilos e padrões arquiteturais

Máquina virtual (Virtual Machine)

- Baseado em regras (*Rule-based*)
  - Exemplos: Prolog, Sistemas Especialistas

SE “HORA=21:00”
ENTÃO “AÇÃO=LANCHE”

SE “HORA=22:00”
ENTÃO “AÇÃO=LIBERAR”

SE “HORA<19:00”
ENTÃO “AÇÃO=ESPERAR”

SE “HORA=19:00”
ENTÃO “AÇÃO=COMEÇAR”

Base de Regras

Memória de trabalho

HORA = 18:00
AÇÃO = ESPERAR

→ AÇÃO=ESPERAR

Máquina de inferência

Estilos e padrões arquiteturais

Classificação

Invocação/Retorno (*Call/Return*)

- Programa principal/Subrotina (*Main Program/Subroutine*)
- Invocação remota de procedimento (*Remote Procedure Call - RPC*)
- Camadas (*Layered*)

Componentes independentes (*Independent Components*)

- Comunicação de processos (*Communicating Processes*)
- Baseado em eventos
- Arquitetura orientada a serviços (SOA)

Máquina virtual (*Virtual Machine*)

- Interpretador (*Interpreter*)
- Baseado em regras (*Rule-based*)

Centrado em dados (*Data-Centered*)

- Repositório (*Repository*)
- Quadro negro (*Blackboard*)

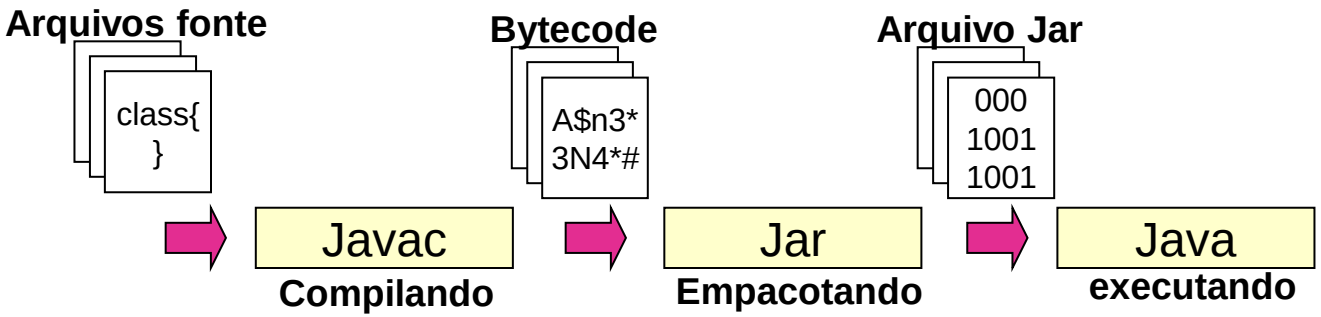
Fluxo de dados (*Data-Flow*)

- Seqüencial (*Batch Sequential*)
- Tubos e filtros (*Pipe and Filter*)
- *Map-Reduce*

Estilos e padrões arquiteturais

Fluxo de dados (Data Flow)

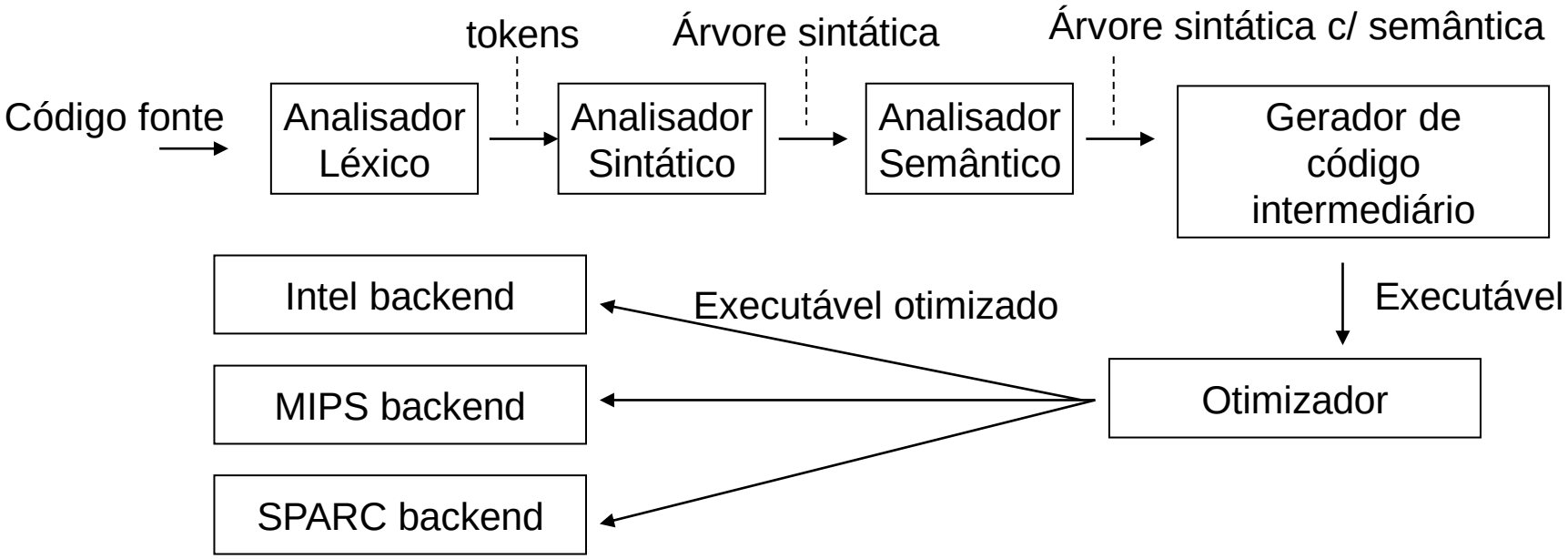
- Sequencial (*Batch Sequential*)
  - Programas independentes executados em seqüência
    - Um após o outro
    - Dado transmitido por completo entre um programa e outro



Estilos e padrões arquiteturais

Fluxo de dados (Data Flow)

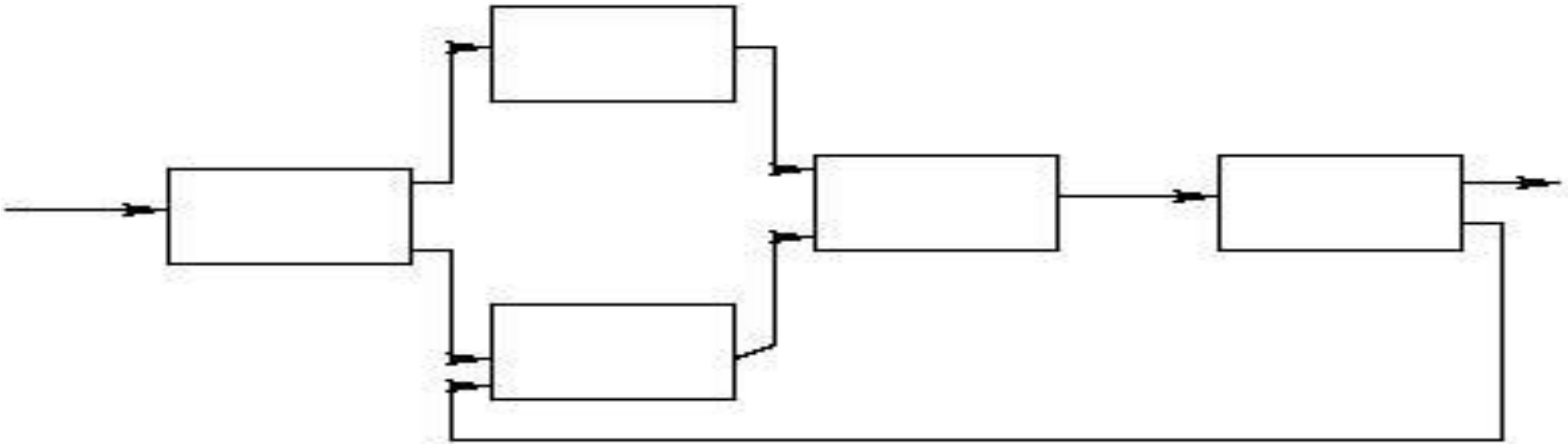
- Tubos e filtros (*Pipe and Filter*)
  - Já vimos na última aula
  - Exemplo: compilador



Estilos e padrões arquiteturais

Fluxo de dados (Data Flow)

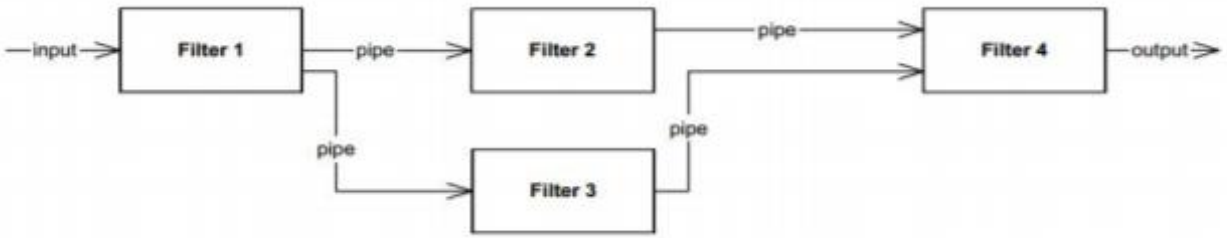
- Tubos e filtros (*Pipe and Filter*)
  - Não precisa ser seqüencial



Estilos e padrões arquiteturais

Fluxo de dados (Data Flow)

Exemplo do padrão ***Pipe-Filter*** (PF):

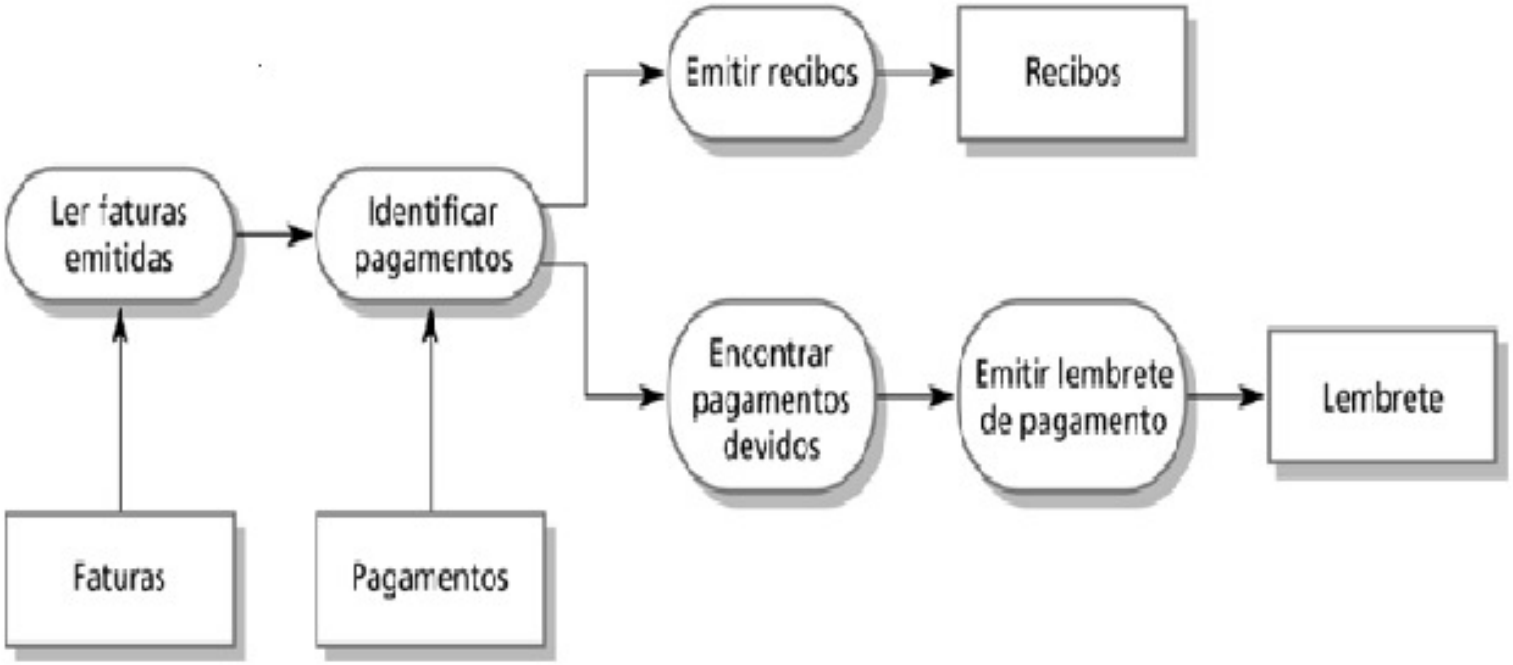


Descrição	Possibilita a transformação de fluxo de dados de forma consecutiva
Utilidade	Melhora o reuso da implementação dos filtros devido a baixa dependência entre eles; favorece a paralelização do processamento dos dados; simplifica o raciocínio sobre o comportamento geral do sistema
Elementos	Filtros → componente que transforma dados de entrada e comunica as transformações como um novo conjunto de dados de saída. Pipe → Conecta a saída de um filtro com a entrada de outro. Responsável pela comunicação do streaming de dados.
Relações	Relação de *attachment* (ligação) que linka os pontos de saída dos filtros com os pontos de entrada do pipe, e os pontos de entrada dos filtros com os pontos de saída do pipe
Propriedades	Dados são transformados pelos filtros e comunicados pelas pipes.
Restrições	Os dados de saída do filtro emissor devem ser compatíveis com os dados de entrada do filtro receptor.

Estilos e padrões arquiteturais

Fluxo de dados (Data Flow)

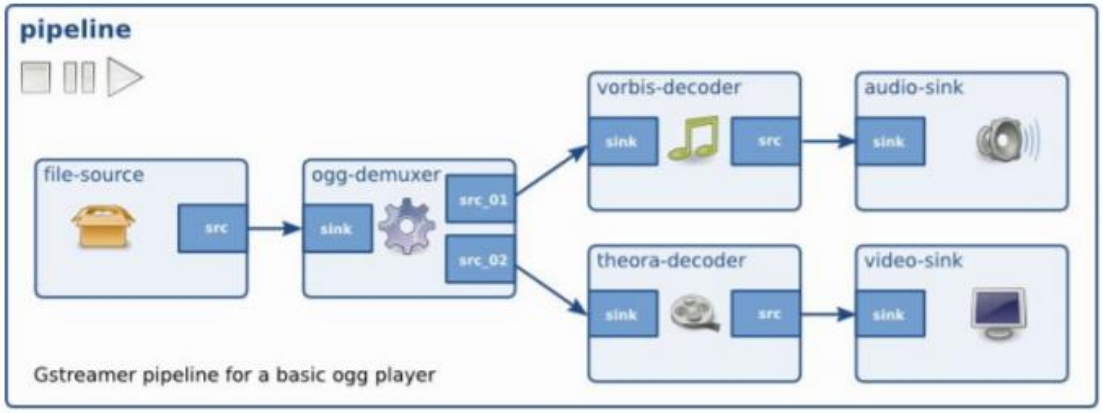
- Exemplo: Sistema de processamento de faturas



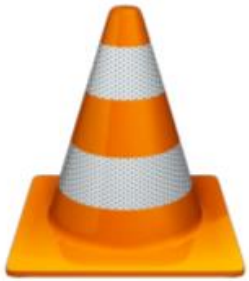
Estilos e padrões arquiteturais

Fluxo de dados (Data Flow)

Exemplo do padrão ***Pipes-and-Filters*** (PF):



Reprodutores de vídeo em diferentes formatos



Estilos e padrões arquiteturais

Exercício – Fazer baseado no Projeto que está sendo desenvolvido na disciplina

- Quais estilos arquiteturais podem ser aplicados na aplicação que você está projetando? Como?
- Se algum estilo não puder ser aplicado, justifique o porquê.

Invocação/Retorno (*Call/Return*)
- Programa principal/Subrotina (*Main Program/Subroutine*)
- Invocação remota de procedimento (*Remote Procedure Call - RPC*)
- Camadas (*Layered*)

Componentes independentes (*Independent Components*)
- Comunicação de processos (*Communicating Processes*)
- Baseado em eventos
- Arquitetura orientada a serviços (SOA)

Máquina virtual (*Virtual Machine*)
- Interpretador (*Interpreter*)
- Baseado em regras (*Rule-based*)

Centrado em dados (*Data-Centered*)
- Repositório (*Repository*)
- Quadro negro (*Blackboard*)

Fluxo de dados (*Data-Flow*)
- Seqüencial (*Batch Sequential*)
- Tubos e filtros (*Pipe and Filter*)
- *Map-Reduce*

Seleção de estilos e padrões arquiteturais

Seleção de estilos

- Como selecionar um estilo arquitetural?
  1. Identificar os principais elementos da arquitetura
  2. Identificar o estilo arquitetural dominante
  3. Considerar responsabilidades adicionais associadas com a escolha do estilo
  4. Modificar o estilo para atingir objetivos adicionais

Fonte:

Carnegie Mellon
Software Engineering Institute

Seleção de estilos

1. Identificar os principais elementos da arquitetura
   - Cada elemento arquitetural tem um estilo arquitetural dominante que reflete as qualidades importantes que devem ser alcançadas no contexto daquele elemento
   - A escolha do estilo arquitetural dominante é baseada nos principais elementos arquiteturais
   - Os atributos de qualidade sobre cada elemento arquitetural podem acarretar a utilização ou não de um estilo

Fonte:

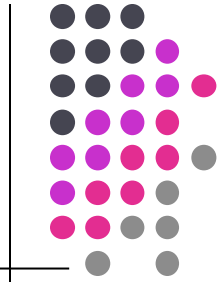
Carnegie Mellon
Software Engineering Institute

Seleção de estilos

2. Identificar o estilo arquitetural dominante
   - O estilo dominante pode ser modificado para alcançar objetivos particulares
   - Se nenhum estilo conhecido parece ser apropriado, o arquiteto deve projetar e documentar um novo estilo
   - As decisões sobre escolhas baseadas em atributos de qualidade dentro de um estilo devem ser documentadas

Fonte:
Carnegie Mellon
Software Engineering Institute

Seleção de estilos

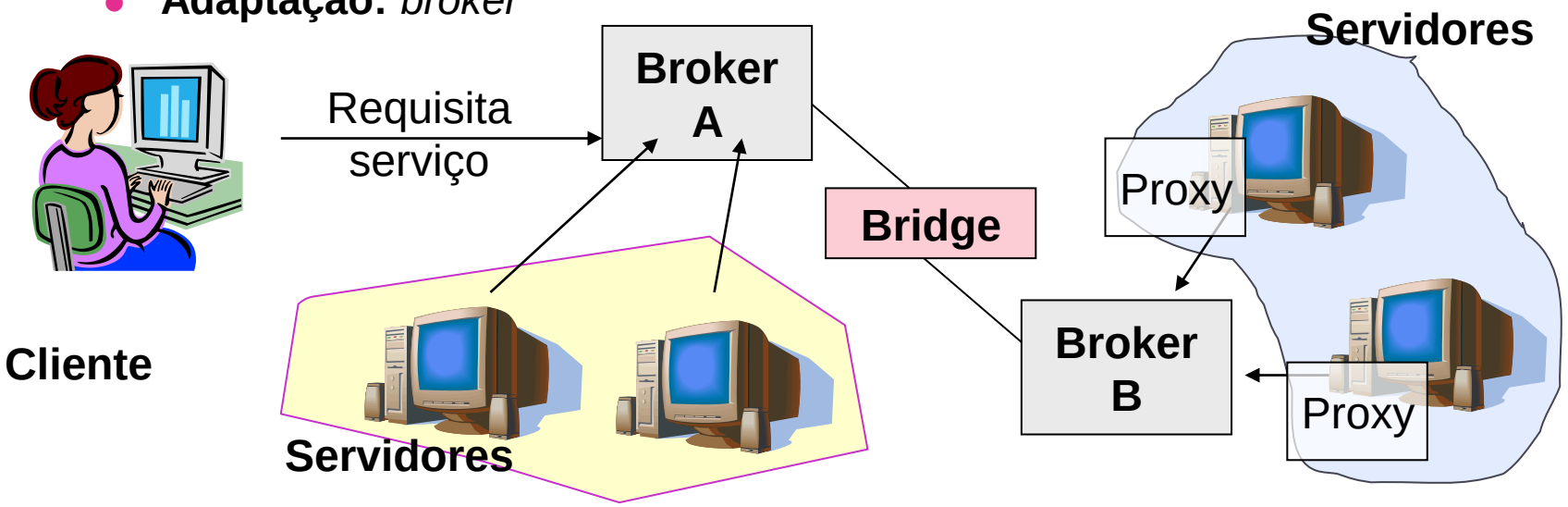


3. Considerar responsabilidades adicionais associadas com a escolha do estilo
   - A escolha de um estilo arquitetural introduzirá responsabilidades adicionais
   - Por exemplo:
     - Se o estilo é “Quadro negro”, então deve-se gerenciar os mecanismos para o controle do quadro negro
     - Se o estilo é “cliente-servidor”, deve-se gerenciar os protocolos de interação
   - Responsabilidades adicionais devem ser atribuídas a elementos arquiteturais existentes ou a novos elementos criados para este fim

Fonte:

Seleção de estilos

4. Modificar o estilo para atingir objetivos adicionais
   - Pode-se alterar o estilo arquitetural caso este necessite ser adaptado devido a atributos de qualidade ou até mesmo funcionalidade
   - Exemplo: cliente-servidor
     - **Adaptação:** *broker*

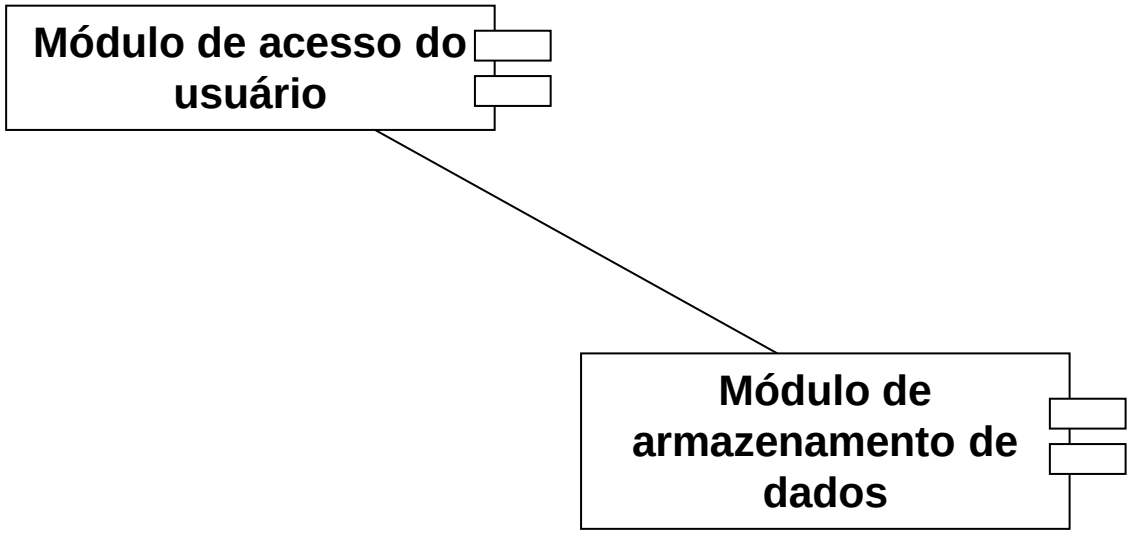


Fonte:

Seleção de estilos

Exemplo

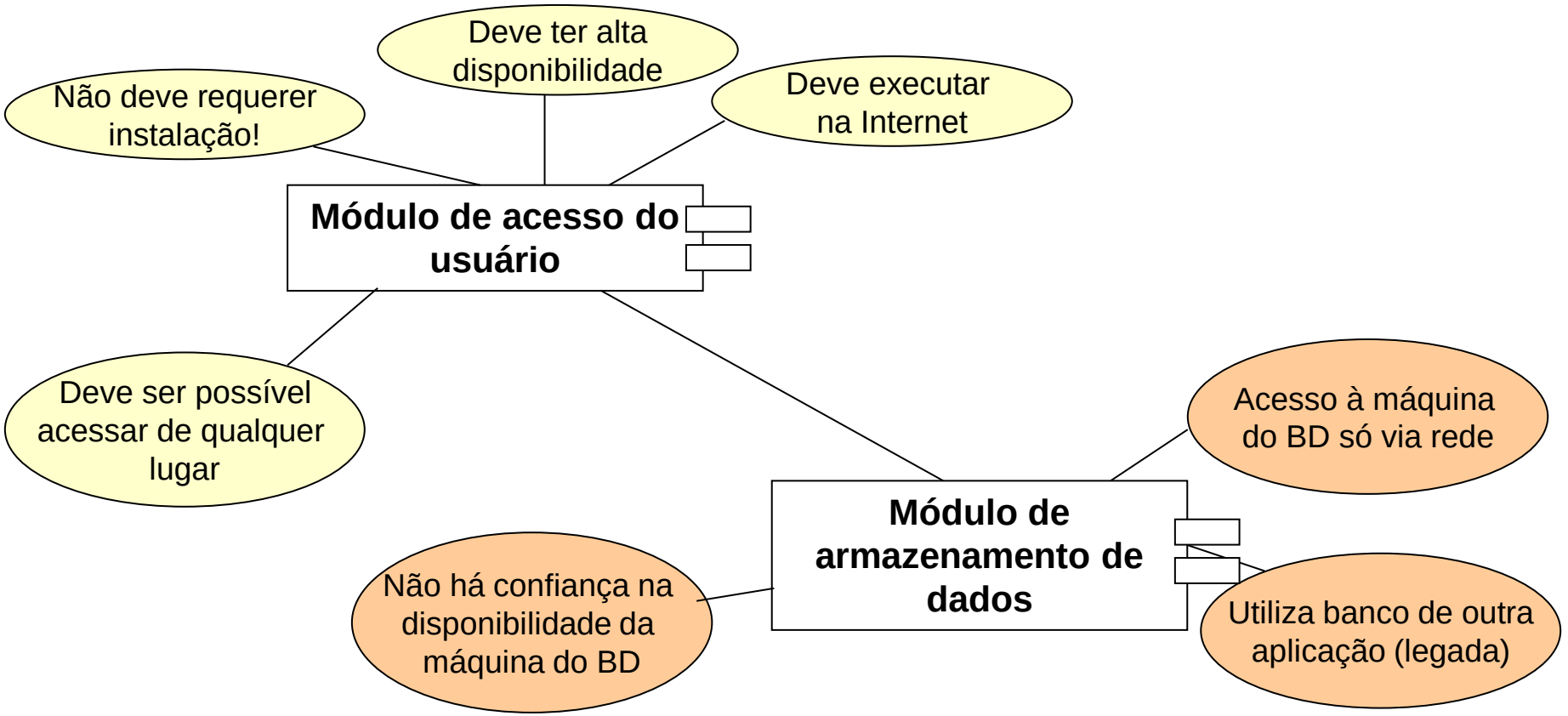
1. Identificar os principais elementos da arquitetura
   - Sistema: acadêmico



Seleção de estilos

Exemplo

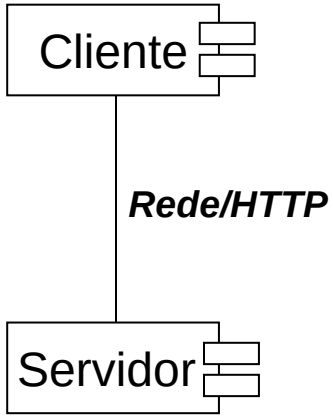
2. Identificar o estilo arquitetural dominante



Seleção de estilos

Exemplo

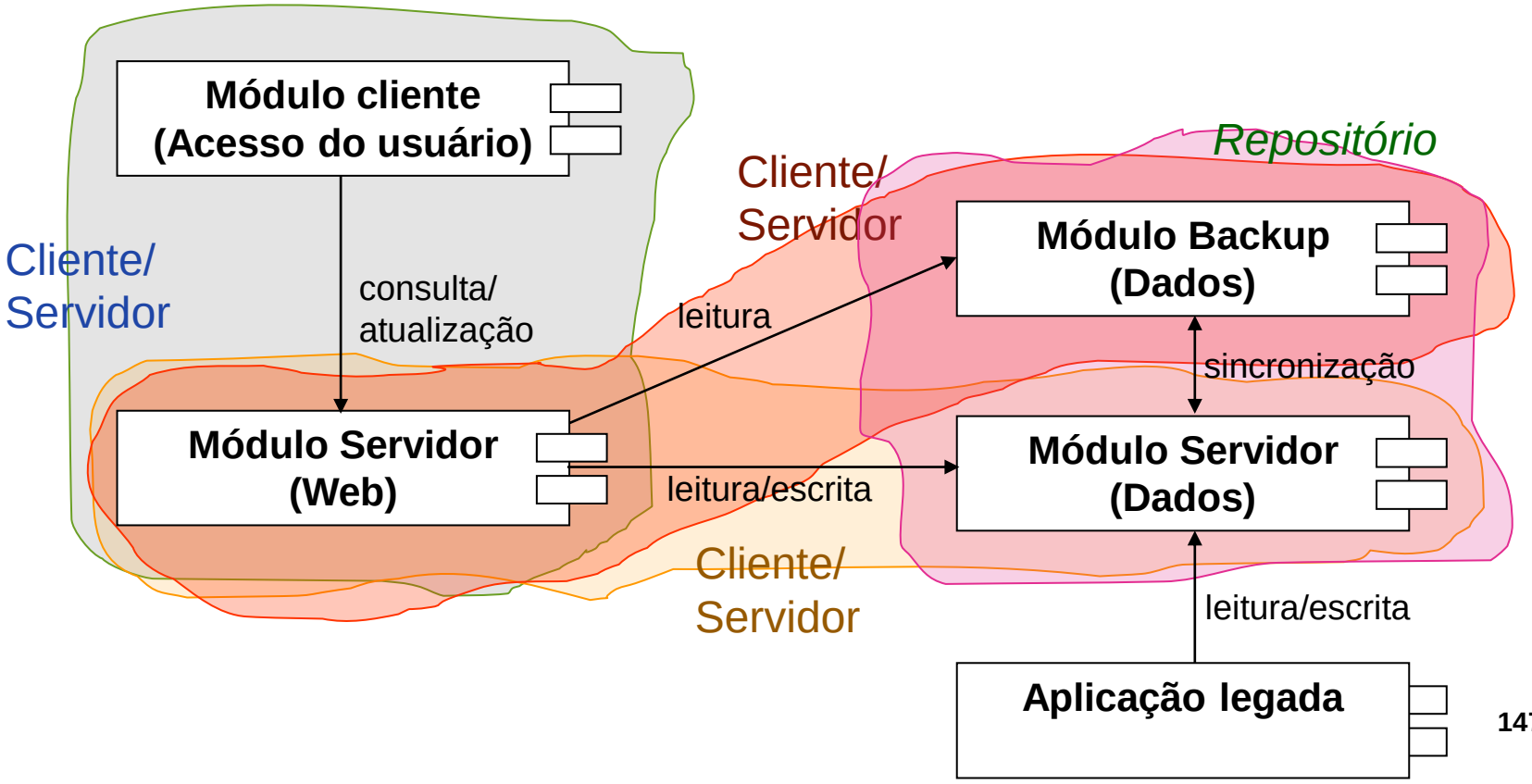
3. Considerar responsabilidades adicionais associadas com a escolha do estilo
   - Estilo dominante escolhido: Cliente-servidor
   - Estilo secundário: Repositório
   - Responsabilidades: considerar protocolo de comunicação



Seleção de estilos

Exemplo

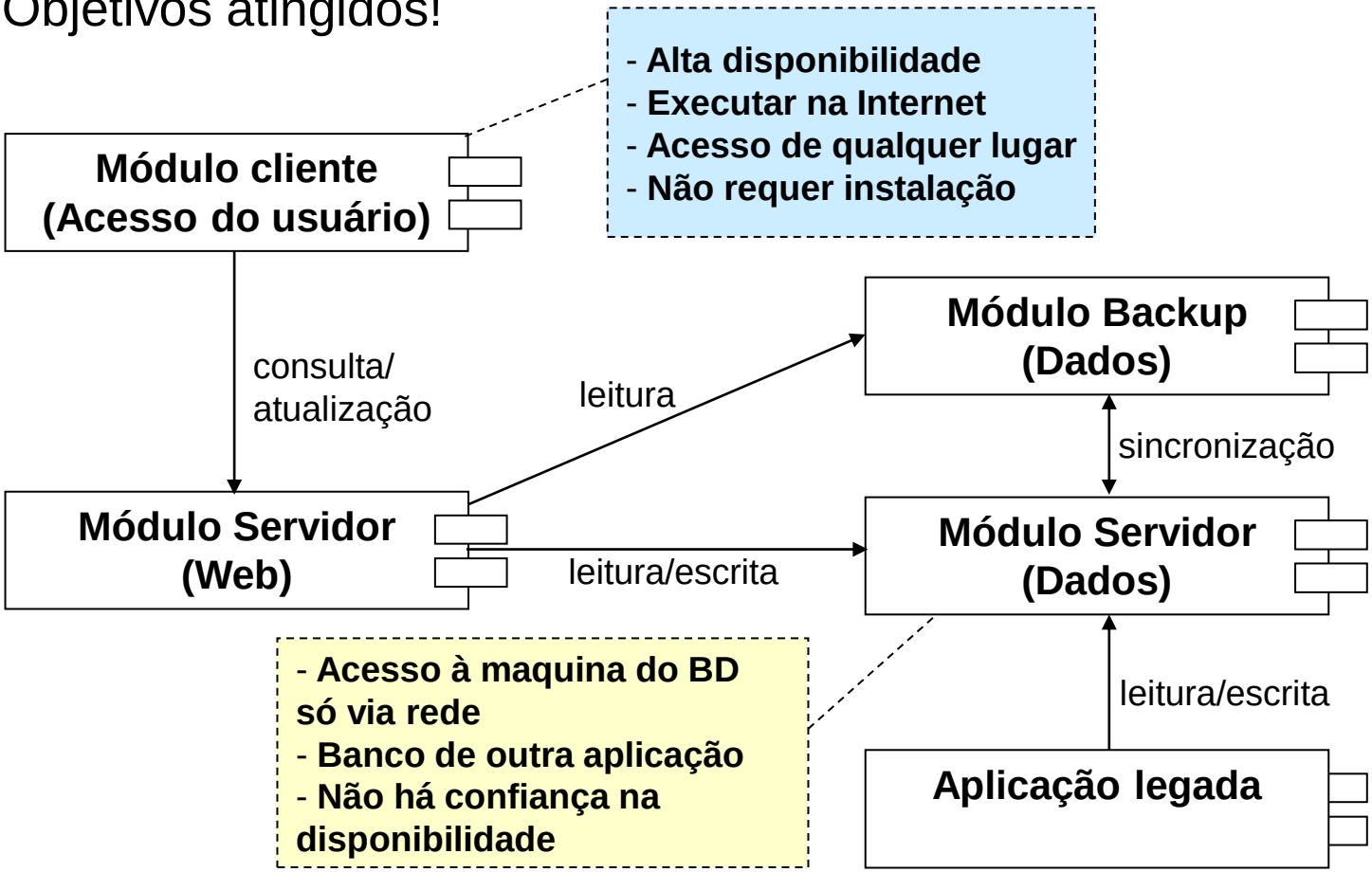
4. Modificar o estilo para atingir objetivos adicionais
   - Cliente servidor de 2 camadas e repositório com *backup*



Seleção de estilos

Exemplo

4. Modificar o estilo para atingir objetivos adicionais
   - Objetivos atingidos!



Seleção de estilos

Exercício

- Considerando os atributos de qualidade, qual ou quais estilos arquiteturais você utilizaria para o seu projeto?

Invocação/Retorno (*Call/Return*)

Programa principal/Subrotina (*Main Program/Subroutine*)
Invocação remota de procedimento (*Remote Procedure Call - RPC*)
Camadas (*Layered*)

Componentes independentes (*Independent Components*)

Comunicação de processos (*Communicating Processes*)
Baseado em eventos

Máquina virtual (*Virtual Machine*)

Interpretador (*Interpreter*)
Baseado em regras (*Rule-based*)

Centrado em dados (*Data-Centered*)

Repositório (*Repository*)
Quadro negro (*Blackboard*)

Fluxo de dados (*Data-Flow*)

Seqüencial (*Batch Sequential*)
Tubos e filtros (*Pipe and Filter*)

Atributos de qualidade

Exercício

- Construa uma matriz que relaciona os estilos arquiteturais (linhas) e atributos de qualidade que sua aplicação deve atender.
- Utilize, nas células, as letras F para dizer que aquele estilo favorece aquele atributo de qualidade, e P para quando um estilo prejudica um atributo de qualidade;
- Se não houver impacto de um no outro, deixe em branco;

Qualidades de sistema: disponibilidade, mutabilidade, desempenho, segurança, testabilidade e usabilidade
Qualidades de negócio: tempo de produção, custo e benefício, tempo de vida projetado, mercado alvo, agenda de divulgação, integração com sistemas legados
Qualidades de arquitetura: *buildability*, integridade conceitual, corretude e completude

Projeto Parte 2 – Seleção de estilos e padrões arquiteturais

- O que deve ser ENTREGUE:
  - Documento da arquitetura de acordo com template fornecido (adicionando as seções necessárias para decisões de arquitetura)
- O que deve ser APRESENTADO:
  - Três decisões arquiteturais
    - Explicar a decisão
    - Apresentar a imagem que represente a parte da arquitetura
    - Justificar a decisão

Atributos de qualidade

Exercício

- Construa uma matriz que relaciona os estilos arquiteturais (linhas) e atributos de qualidade que sua aplicação deve atender.
- Utilize, nas células, as letras F para dizer que aquele estilo favorece aquele atributo de qualidade, e P para quando um estilo prejudica um atributo de qualidade;
- Se não houver impacto de um no outro, deixe em branco;

Qualidades de sistema: disponibilidade, mutabilidade, desempenho, segurança, testabilidade e usabilidade
Qualidades de negócio: tempo de produção, custo e benefício, tempo de vida projetado, mercado alvo, agenda de divulgação, integração com sistemas legados
Qualidades de arquitetura: *buildability*, integridade conceitual, corretude e completude

Projeto Parte 2 – Seleção de estilos e padrões arquiteturais

- O que deve ser ENTREGUE:
  - Documento da arquitetura de acordo com template fornecido (adicionando as seções necessárias para decisões de arquitetura)
- O que deve ser APRESENTADO:
  - Três decisões arquiteturais
    - Explicar a decisão
    - Apresentar a imagem que represente a parte da arquitetura
    - Justificar a decisão

Projeto Parte 2 – Reutilização de Software

- **Cada grupo deve reutilizar e evoluir um projeto base disponível no GitHub sobre o tema (ou um sistema próprio).**

- **A partir desta implementação base, o grupo deve desenvolver uma Linha de Produtos de Software (LPS) com novas características. [Colocar o Mapa de Produto e também o modelo FODA)**

- **O código disponível no GitHub pode ser reusado, mas o grupo deve deixar claro qual foi a parte reusada e qual foi a parte desenvolvida pelo grupo.**

DATA DE ENTREGA

- **Entrega:**
  - **15/10/2023 (até 23h59)**
- **Aula**
  - **17/10/2023 (Feedback por equipe da parte 2 do Projeto (sala de aula)**