Arquitetura de Software

Estilos Arquiteturais

Nairon Neri Silva

Sumário

- Como definir Arquiteturas?
- Estilos Arquiteturais
- Elementos de um Estilo
- Padrões e Estilos Arquiteturais
- Aspectos do Processo de Definição da Arquitetura de Software
- Quando Criar a Arquitetura?
- Categorias de Problemas
- Definindo Camadas e Abstrações
- Técnicas Auxiliares para a Definição da Arquitetura de Software
- Processo de Definição da Arquitetura

Como Definir Arquiteturas?

- Dois tipos de elementos podem ser utilizados para a definição de uma arquitetura:
  - **Componentes** → blocos de construção de um sistema – partes do software ou provedores de funcionalidade
  - **Serviços** → são providos pelos componentes para os atores ou uns para os outros

- Um conjunto de componentes oferece as funcionalidades de um sistema

Estilos Arquiteturais

- Estilos arquiteturais de um software equivalem aos estilos arquiteturais de uma casa

- Referem-se ao “formato” geral do sistema

- A escolha do estilo arquitetural apropriado é muito importante, pois as demais decisões são tomadas no contexto do referido estilo

Estilos Arquiteturais

- Um sistema pode ser definido segundo
  - Um estilo de “fluxo”, como o ***Pipes and Filters***
  - Um estilo interativo, como o ***Model-View-Controller***

- Um estilo define características e regras que formam a arquitetura

- A escolha do estilo apropriado é a chave para o sucesso do sistema

Elementos de um Estilo

1. Os blocos básicos de construção
2. Os conectores entre os blocos básicos (comunicação)
3. Regras que especificam como os serviços podem ser combinados e utilizados em conjunto
4. A soluções intrínsecas ao estilo
5. Contexto e situações problema nas quais o referido estilo é mais útil

Padrões e Estilos Arquiteturais

Estilo Arquitetural	Padrão
From Mud to Structure	*(1) Em camadas,* *(2) Pipes and Filters,* *(3) Blackboard*
Distributed Systems	*(4) Broker*
Interactive Systems	*(5) Model-View-Controller e* *(6) Presentation-Abstraction-Control*
Adaptable Systems	*(7) Microkernel e* *(8) Reflection*

Aspectos do Processo de Definição da Arquitetura de Software

- **Tempo**
  - Quando criar a arquitetura do software?

- **Categoria de problemas**
  - O domínio da solução influencia as decisões arquiteturais

- **Camadas e abstrações**
  - Pensar em diferentes níveis de abstração é uma habilidade fundamental na construção da arquitetura

Quando Criar a Arquitetura?

- No início do processo de desenvolvimento

- Se o desenvolvimento é iterativo e incremental:
  - A arquitetura inicia a ser elaborada nas iterações iniciais em paralelo com algum projeto (baixo nível) e codificação
  - Cada iteração pode incluir mais refinamentos da arquitetura em conjunto com projetos e codificação
  - A cada iteração a arquitetura fica mais completa e estável

Categorias de Problemas

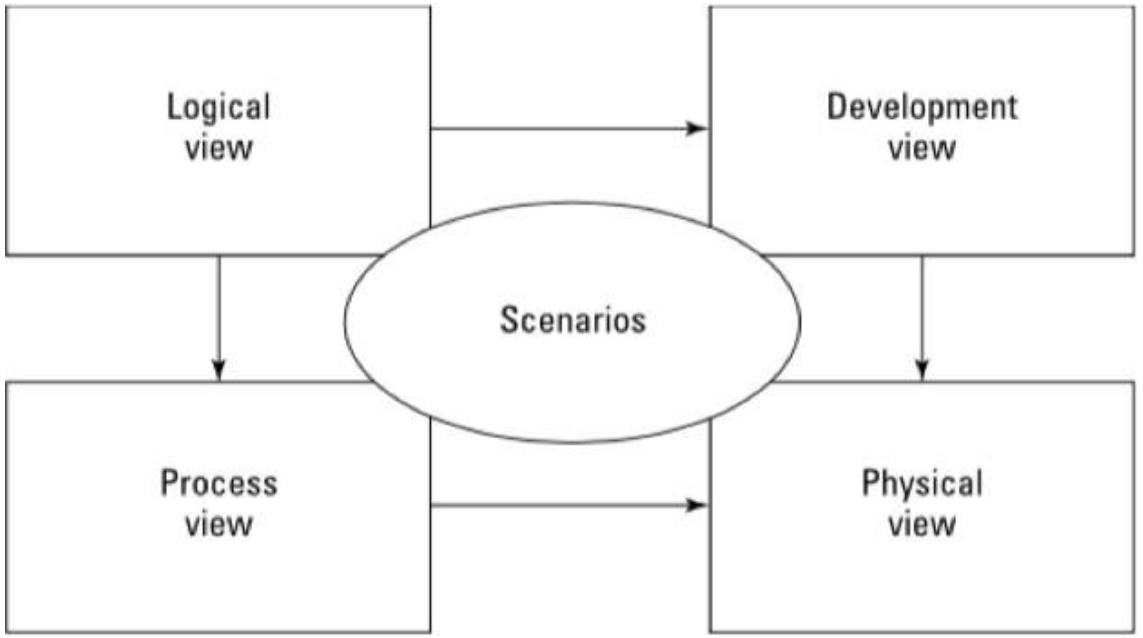
- Quando se desenvolve a arquitetura, você resolve problemas de vários domínios computacionais

- No sistema de folha de pagamento:
  - Domínio de banco de dados – para armazenar o histórico de pagamentos dos funcionários
  - Domínio de sistema interativo – para a leitura das horas trabalhadas pelos funcionários (calcular o pagamento)

- Solucionar problemas reais envolve vários domínios

Definindo Camadas e Abstrações

- Apresentamos o modelo 4+1 de visões arquiteturais
  1. Visão lógica
  2. Visão de processos
  3. Visão física
  4. Visão de desenvolvimento
  5. + a visão de cenários (ou casos de uso)



- Cada visão apresenta uma abstração (ou ponto de vista diferente) da mesma arquitetura

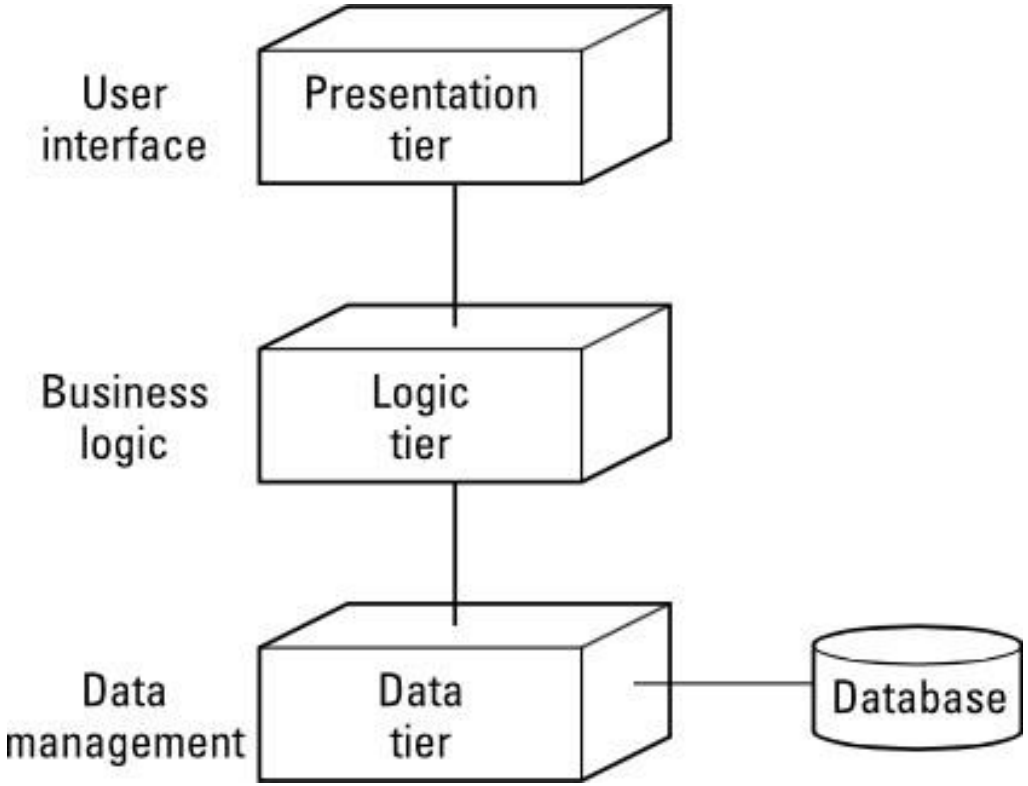
Definindo Camadas e Abstrações

- Algumas vezes seu sistema ou o seu ambiente possui **camadas** explicitas de funcionalidade

- Por exemplo: se está sendo desenvolvido um sistema de comunicação, você precisa estar familiar com o modelo OSI
  - No modelo OSI, as (sete) camadas representam a visão lógica da arquitetura

Definindo Camadas e Abstrações

- Em alguns casos, as camadas podem estar associadas aos elementos físicos (computadores)



Definindo Camadas e Abstrações

- Uma **abstração** é uma forma de descrever algo em termos gerais – deixando os detalhes para alguma implementação específica

- Por exemplo: a arquitetura em três camadas, os elementos são distribuídos em três grupos de acordo com sua funcionalidade abstrata
  1. Apresentação
  2. Lógica do negócio
  3. Armazenamento persistente

Técnicas Auxiliares para a Definição da Arquitetura de Software

1. **Abstração**
   - Habilidade de extrair o que é comum e geral à dadas entidades
2. **Encapsulamento**
   - Agrupar informações para preservar as fronteiras da abstração
3. **Ocultação de Informação**
   - Esconder as informações que um cliente não precisa saber
4. **Modularização**
   - Gerência da complexidade quebrando o sistema em partes
5. ***Separation of Concerns (Separação de preocupações)***
   - Responsabilidades não relacionadas precisam ficar separadas
6. **Acoplamento e Coesão**
   - Acoplamento (-): como os diferentes módulos se relacionam
   - Coesão (+): a medida de quanto um módulo é autossuficiente

Técnicas Auxiliares para a Definição da Arquitetura de Software

7. **Suficiência e Completude**
   - Componentes possuem características suficientes para uma interação útil e eficiente com os demais
8. **Separação das Políticas e Implementação**
   - Implementações não se prendem ao contexto → (+) reuso
9. **Separação das Interfaces da Implementação**
   - Clientes acessam interfaces separadas da implementação
10. **Único Ponto de Referência**
    - Definição única dos itens da arquitetura de software
11. **Dividir para Conquistar**
    - Dividir o problema em partes menores, simplificando a solução

Definindo a Arquitetura

Processo de Definição da Arquitetura

- **Passo 1**: selecione um componente a ser refinado
  - O primeiro componente que você irá selecionar é o **sistema completo**
  - Para o componente que você está refinando:
    1. Defina os objetivos e metas do componente
    2. Utilize como entrada os requisitos e a declaração do problema

Processo de Definição da Arquitetura

- **Passo 2**: identifique os requisitos do componente e os requisitos para as suas interações
  - Que outras partes do sistema ou exterior ele interage?
  - Os casos de uso ajudam a entender as interconexões e os serviços que este necessita de outras partes do sistema
  - Esquematize o fluxo de informações (de alto nível) entre esses componentes
  - Pense em:
    - Que partes do componente são responsáveis por partes da arquitetura
    - Os passos de processamento necessários
    - (para a OO) *brainstorm* das classes que compõem o sistema

Processo de Definição da Arquitetura

- Cartões CRC (*Class-Responsibility-Collaboration*) podem ser usados no registro dos componentes
  - Para cada componente escreva um cartão CRC (cenários)
  - Exemplo do sistema de folha de pagamento:

Name: Hour Store	
Responsibilities: • Remember hours worked during pay period	Collaborators: • Hours register • Payment calculator

Processo de Definição da Arquitetura

- **Passo 3**: pesquise por um estilo arquitetural ou padrão que se encaixe aos requisitos e interações identificadas no passo 2
  - Se não encontrar nenhum que case perfeitamente com o seu problema, tente buscar por problemas similares
  - Obs.: teremos mais subsídio no decorrer da disciplina sobre isso

Processo de Definição da Arquitetura

- **Passo 4**: aplique o padrão que se adequou ao seu problema a organização das classes e componentes
  - Cada padrão descreve uma estrutura e define como se dará as interações entre as classes ou componentes

Processo de Definição da Arquitetura

- **Passo 5**: repita os passos de 2 a 4 para cada um dos (sub)componente identificados no processo
  - Ao escolher o próximo componente a ser refinado evite se basear no seu interesse – escolha o próximo componente mais importante para a arquitetura
  - Escolha componentes com funcionalidades críticas
  - Ou componentes possivelmente mais difíceis de projetar

Documente o seu Trabalho

- Tome notas enquanto você define a arquitetura, registre:
  - as decisões chave (e vantagens de cada escolha)
  - opções rejeitadas (e as razões para terem sido excluídas)
  - os pressupostos (a serem validados com o cliente)

- Esboce como as partes trabalham juntas

- Ao final do processo, redija o documento de arquitetura

Exercício Prático

Provavelmente você nunca tenha visto um documento de arquitetura de software, sendo assim, acesse o link abaixo e verifique:

http://repositorio.aee.edu.br/bitstream/aee/1106/3/TCC2_2018_2_GabrielLeiteDias_MatheusLimadeAlbuquerque_Apendice2.pdf

Faça uma pesquisa por “Documento de arquitetura de software” e verifique outros exemplos.

Arquitetura de Software

Estilos Arquiteturais - Detalhamento

Nairon Neri Silva

Sumário

- Introdução
- Pipes and Filters
- Abstração de Dados e Organização Orientada a Objetos
- Baseada em Eventos
- Sistemas em Camadas
- Repositórios
- Interpretadores orientados a Tabela
- Outras Arquiteturas Familiares

Introdução

- Em "*An Introduction to Software Architecture*", artigo de 1994, David Garlan e Mary Shaw definiram:

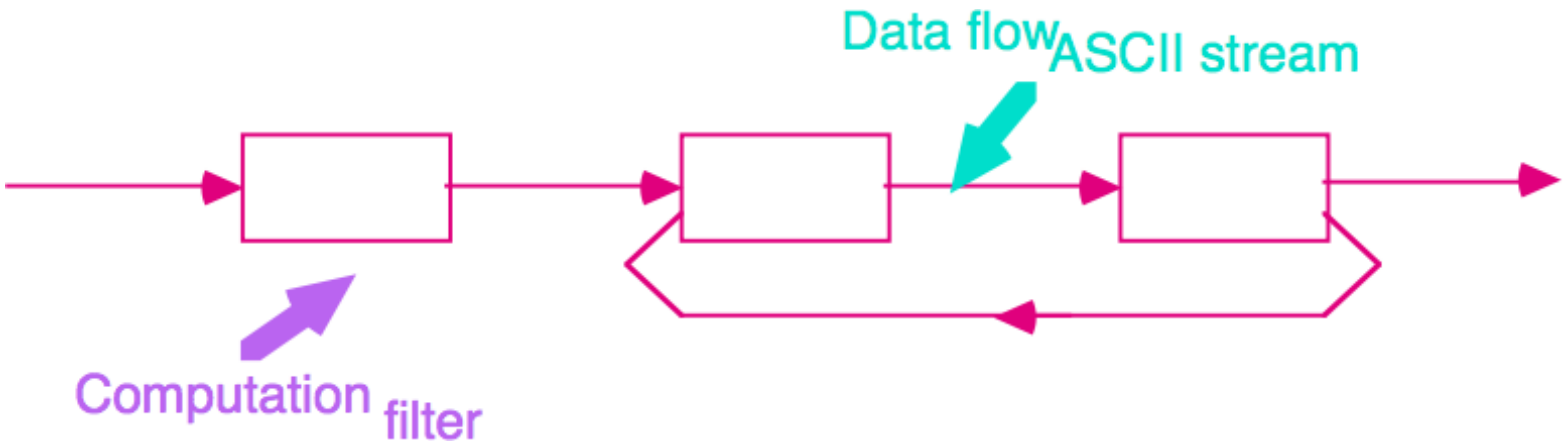
> *"Um estilo arquitetural, então, define uma família de tais sistemas em termos um padrão de organização estrutural. Mais especificamente, um estilo de arquitetura determina o vocabulário de componentes e conectores que podem ser usados em instâncias desse estilo, juntamente com um conjunto de restrições sobre como eles podem ser combinados."*

Pipes and Filters

Dutos e Filtros

Proposta

- Cada componente (filtro) possui uma série de entradas e uma série de saídas
- Recebe um “fluxo” de entrada – realiza algum processamento – gera um “fluxo” de saída



Invariantes do Estilo

1. Os “filtros” devem ser entidades independentes

2. Os “filtros” não devem ter conhecimento do seu antecessor, nem do seu sucessor

3. As especificações dos filtros devem apenas se restringir ao que é recebido nos dutos de entrada e assegurar o que aparecerá nos dutos de saída

Especializações do Estilo

- ***Pipelines*** – restringe a topologia a uma sequência linear de filtros

- ***Bounded pipes*** – (dutos limitados) restringe a quantidade de dados no duto

- ***Typed pipes*** – requerem que os dados que trafegam no duto possuam um tipo bem definido

Observações

- Uma variação indesejada ocorre quando cada filtro processa toda a entrada como uma única entidade
  - Acaba gerando um sistema em lotes (*batch sequential*)
  - A proposta é que seja utilizado em um “fluxo” de dados

Exemplos

1. Programas escritos em *Unix shell*
   - Instrução “pipe”

2. Compiladores de código
   - (a) análise léxica, (b) *parsing*, (c) análise semântica e (d) geração de código

Vantagens

1. Permitem pensar um sistema complexo como uma composição de “filtros” específicos
2. Útil para aplicações de processamento de informação que interagem pouco com os usuários
3. Suportam o reuso – combinação de “filtros”
4. É fácil de adicionar, recombinar ou trocar (flexibilidade)
5. Permite algumas análises especializadas
    - Tais como: *throughput* e análise de *deadlock*
6. Suportam execução concorrente (vários filtros em paralelo)
7. Processamento eficiente

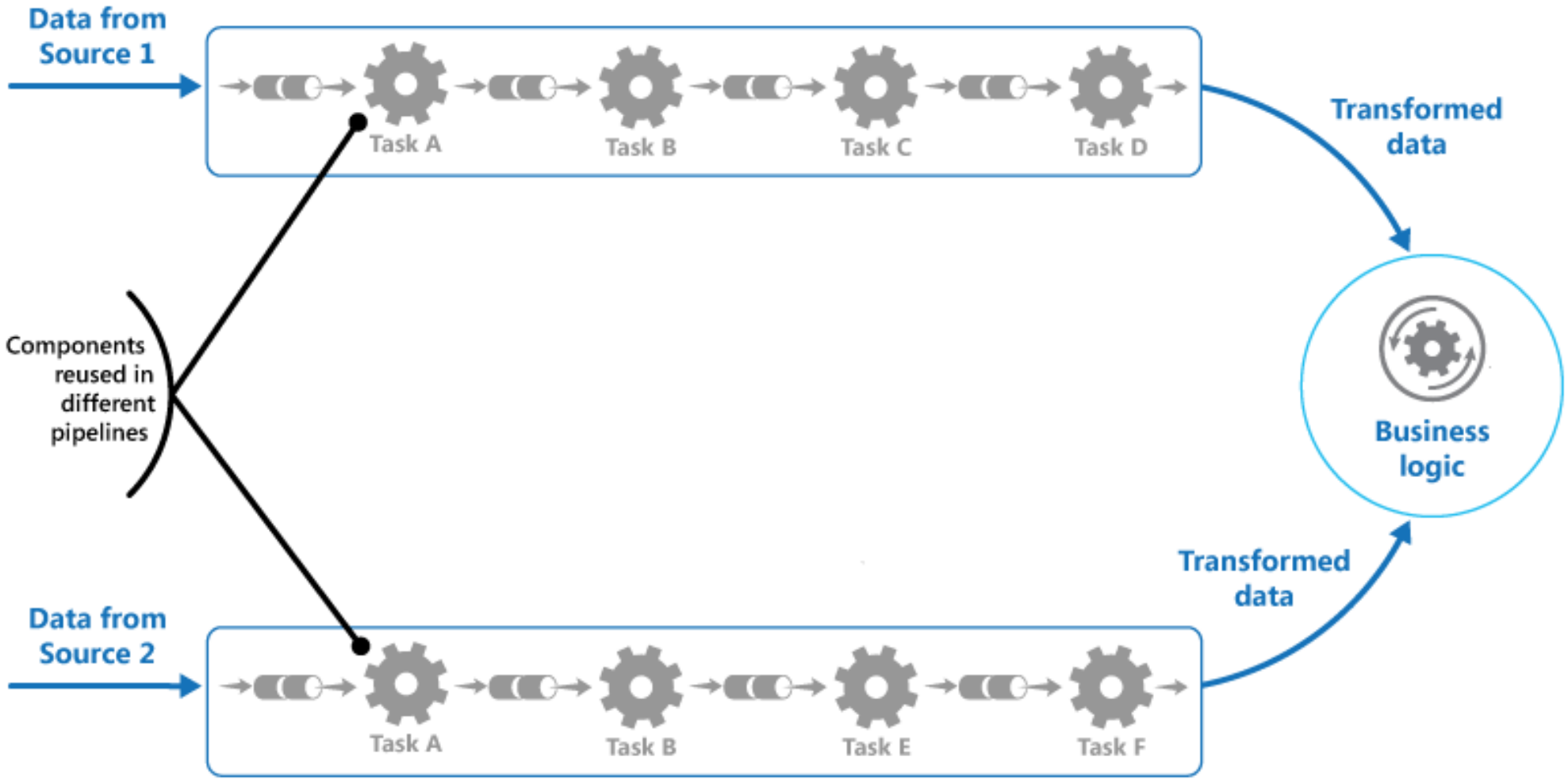
Desvantagens

1. Podem levar a uma organização do processamento em lote
2. Como os filtros são intrinsecamente independentes, são orientados a realizar uma transformação completa da entrada na saída
    - Não são adequados para sistemas interativos
3. Dificuldade de manter correspondência entre dois fluxos distintos, mas relacionados
4. Pode ter trabalho adicional para ajustar os dados às necessidades individuais de cada fluxo
5. Não existe compartilhamento de dados e gerenciamento de erros

Exemplo – Shell do Linux

$ ls | grep b | sort -r | tee arquivo.out | wc –l

No exemplo acima o comando "ls" lista o conteúdo do diretório, no entanto devido ao Pipe ele não envia o resultado para tela e sim para o comando "greb b", que por sua vez filtra os nomes de arquivos que contém a letra "b". O segundo Pipe envia a saída do comando "grep b" para "sort -r", que classifica os nomes em ordem crescente. A saída do comando "sort -r" é passada para o comando "tee", que divide os dados em dois, como uma conexão em T, fazendo com que as informações processadas por "sort -r" sejam escritas no arquivo "arquivo.out", e por fim o comando "wc -l" conta as linhas do arquivo "arquivo.out". Portanto, o resultado será a quantidade de arquivos que contém a letra b impressa na tela e o nomes desses arquivos em "arquivo.out".

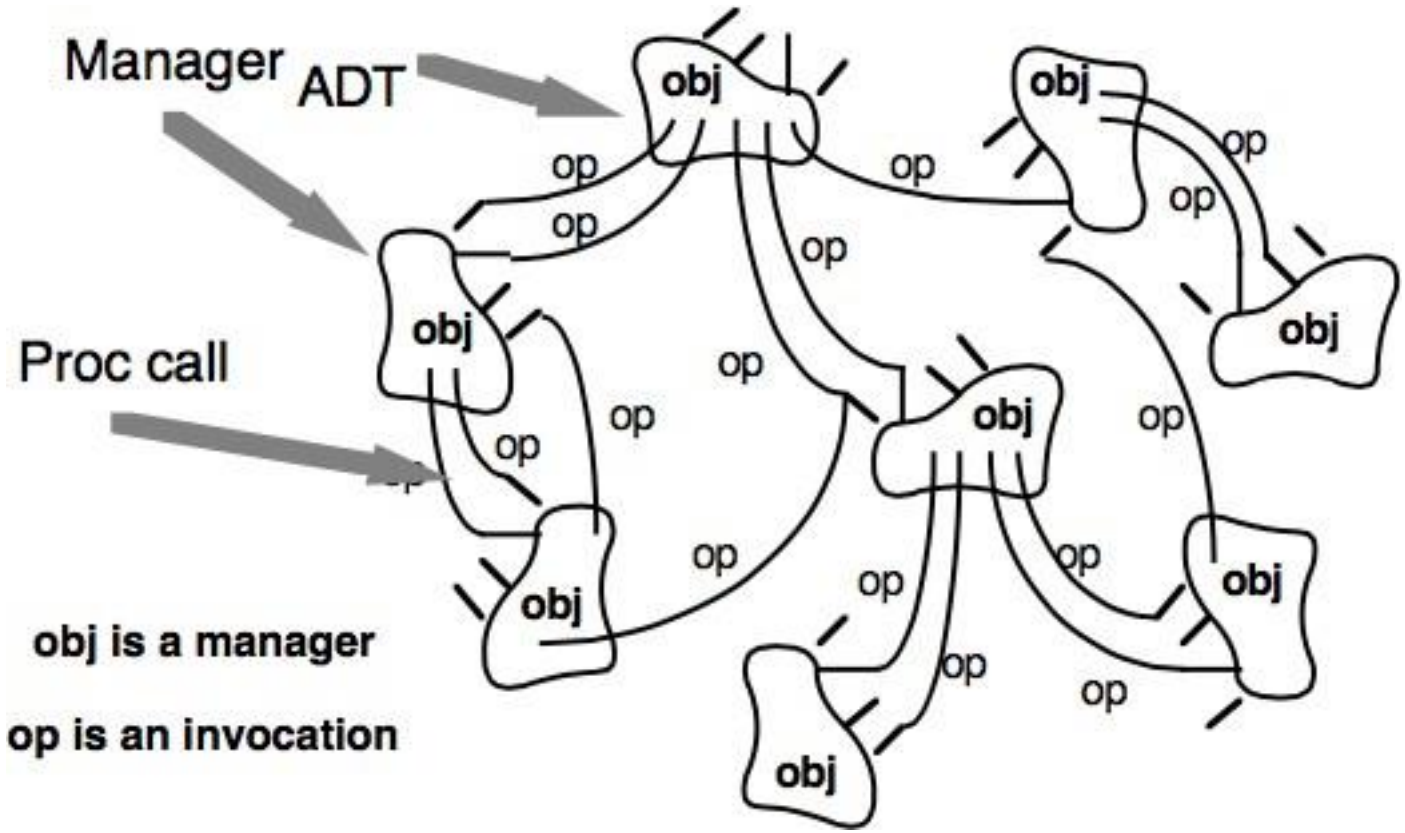


Fonte: https://docs.microsoft.com/pt-br/azure/architecture/patterns/pipes-and-filters

Abstração de Dados e Organização Orientada a Objetos

Introdução

- Nesse estilo, uma representação de dados e as suas operações primitivas são encapsuladas em um tipo abstrato de dados, chamado objeto
- Objetos interagem uns com os outros através da invocação de funções ou procedimentos (métodos)
- Aspectos importantes:
  1. Objetos são responsáveis por manter sua integridade
  2. Sua representação é oculta aos outros objetos


Manager ADT
Proc call
obj
op
obj is a manager
op is an invocation

Observações

- Ampla adoção e vantagens conhecidas

- Vantagens:
  - Objetos podem processar tarefas concorrentes
  - Objetos possuem múltiplas interfaces e são fracamente acoplados
  - Linguagens orientadas a objetos são amplamente usadas.
- Desvantagens:
  - Para que os objetos interajam, é preciso que ambos se conheça, o que gera dependência
  - A alteração de um objeto pode afetar todos os demais que dependem dele – pode gerar efeito em cascata

Baseada em Eventos

Invocação implícita

Introdução

- Geralmente, os componentes interagem através da invocação explícita de métodos

- Alternativa: invocação implícita (integração reativa ou *broadcast* seletivo)

Mecanismo

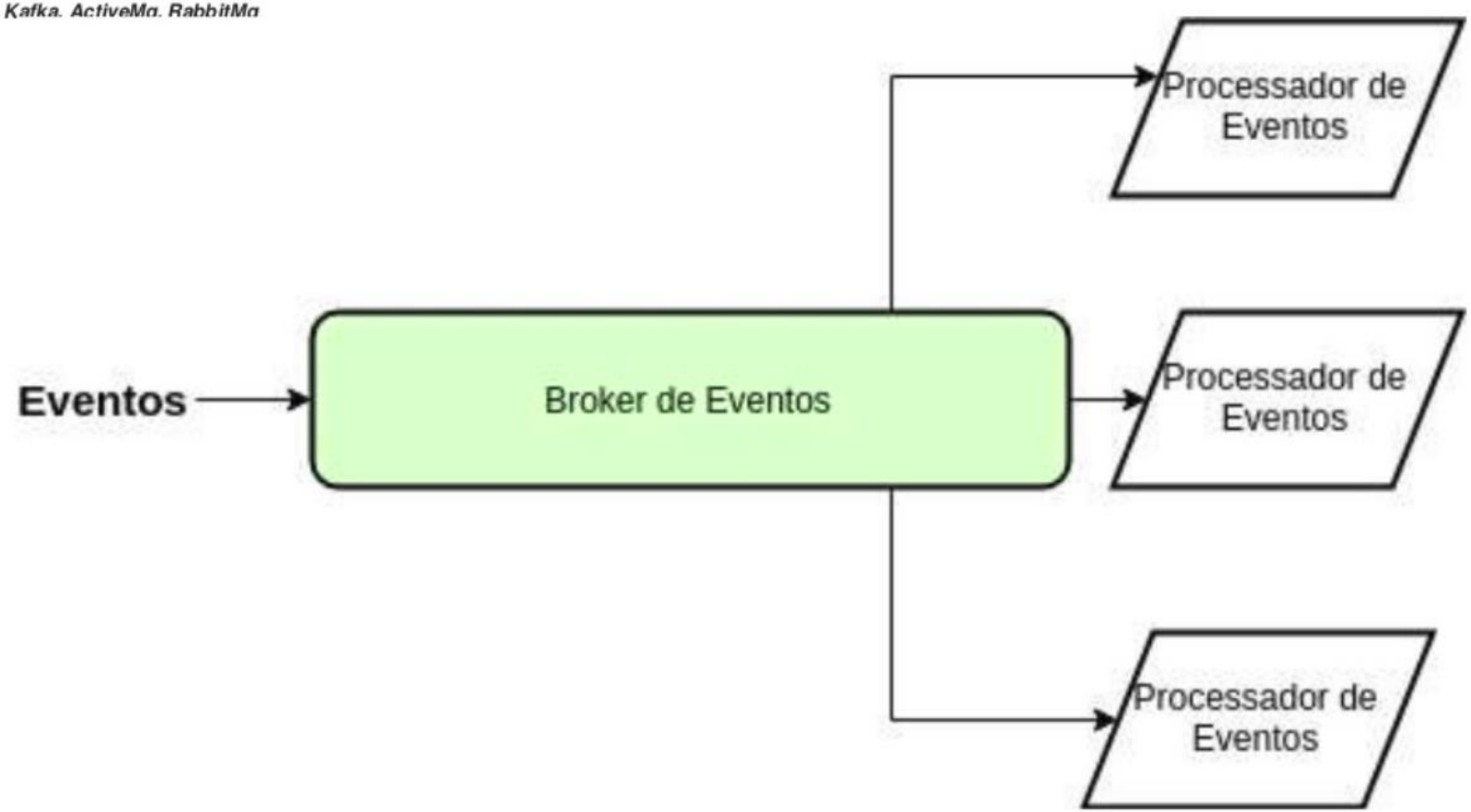
1. Um componente registra o interesse em um evento específico, associando um procedimento

2. Um componente anuncia (*broadcast*) que um dado evento ocorreu

3. O anúncio do evento causa a invocação implícita do procedimento associado

Broker

Kafka, ActiveMq, RabbitMq



https://medium.com/devs-javagirl/desenvolvimento-de-sistemas-e-arquitetura-baseada-em-eventos-3a9894f6a70a

Invariantes

- O anunciante de um evento não sabe quais outros componentes serão afetados pelo evento

- Sistemas com invocação implícita também permite a invocação explícita (complementar)

Exemplos

1. Sistemas de gerenciamento de banco de dados para garantir as restrições de consistência

2. Editores dirigidos pela sintaxe para suportar a checagem semântica incremental

Vantagens e Desvantagens

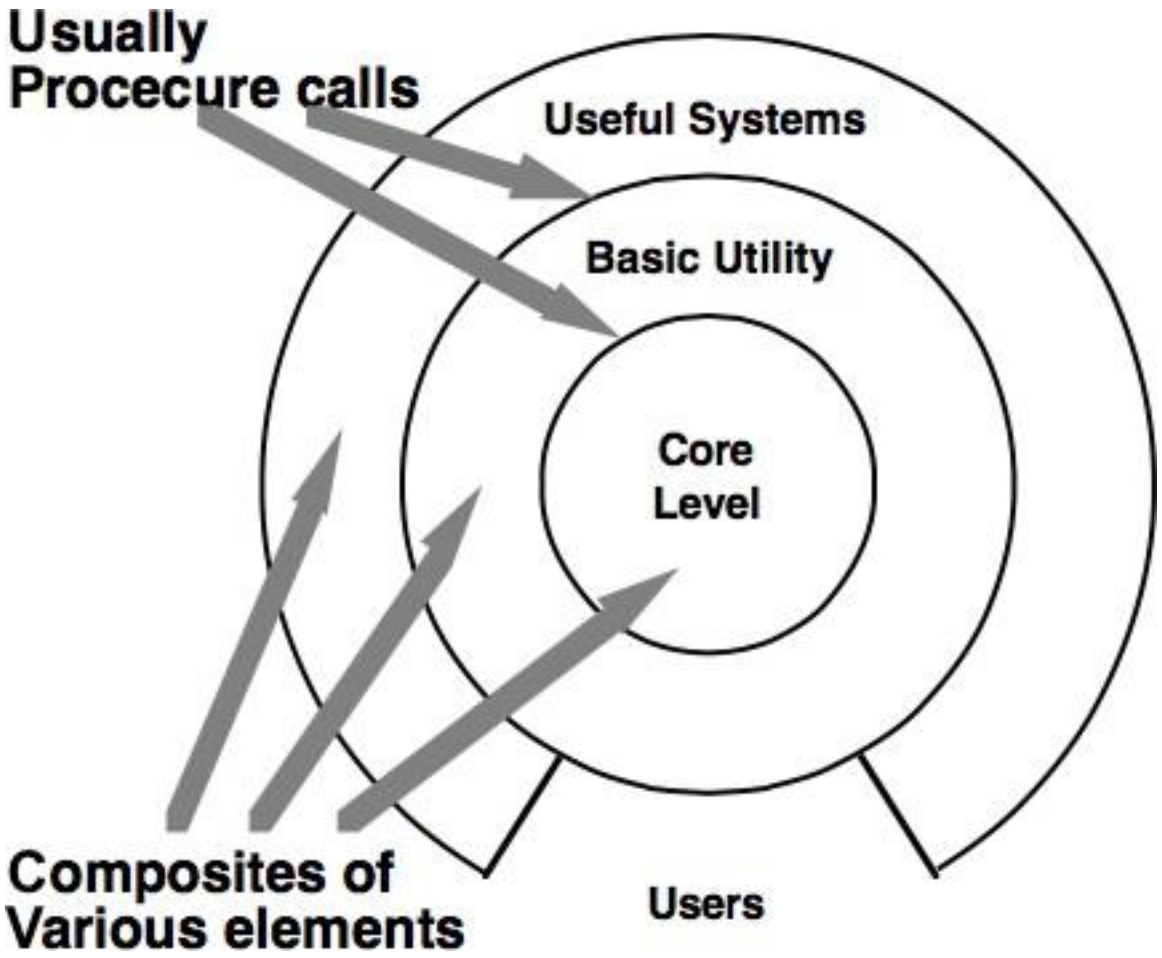
- Vantagens:
  - Forte suporte ao reuso
  - Facilita a evolução do sistema

- Desvantagens:
  - Um componente que anuncia um evento não tem controle (ordem/tempo) sobre a execução associada
  - A troca de dados precisa ocorrer também de forma indireta (repositório compartilhado)

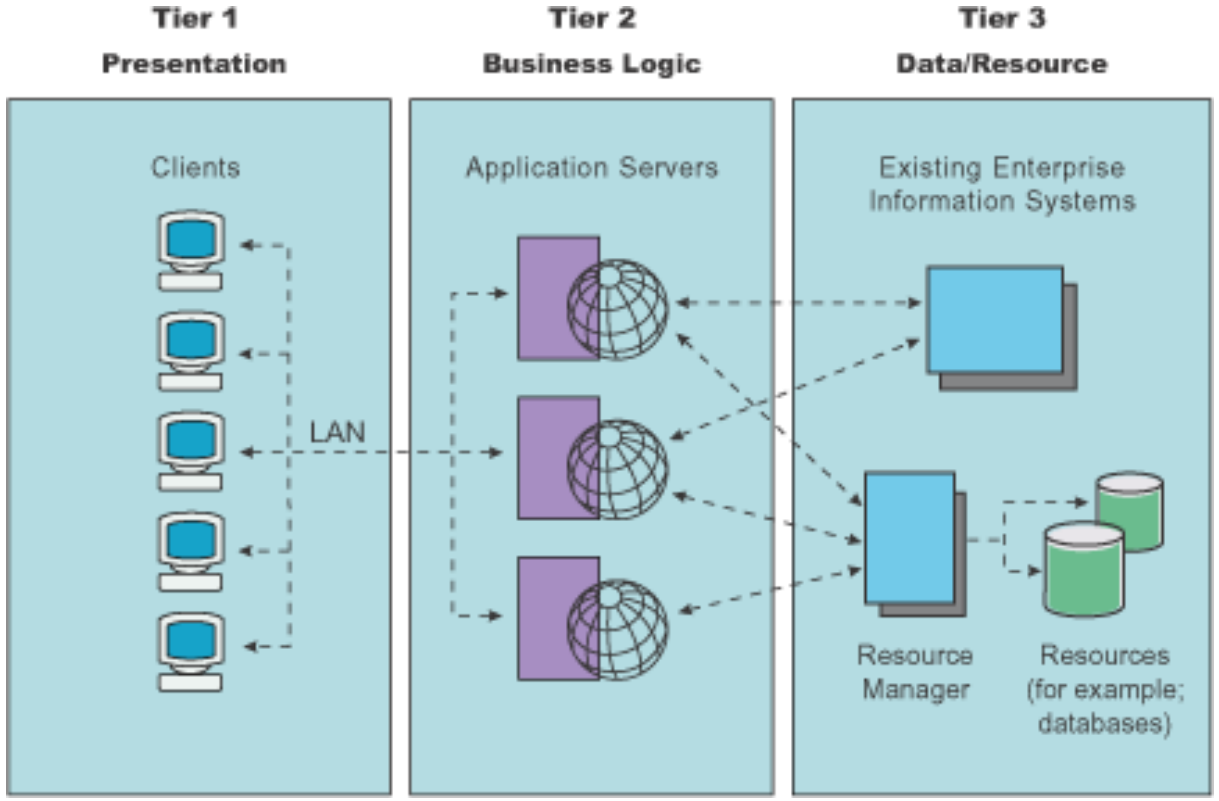
Sistemas em Camadas

Introdução

- São organizados hierarquicamente
- Cada camada provê serviços para a camada superior e consome os serviços da camada inferior
- Geralmente, camadas mais internas ficam ocultas, exceto para a camada imediatamente superior


Usually
Procecure calls
Useful Systems
Basic Utility
Core
Level
Composites of
Various elements
Users


CAMADA DE APRESENTAÇÃO
(MVC, MVP, MGM, etc)
Formulários, Páginas WEB
CAMADA DE SERVIÇOS
(FAÇADE)
Serviços de Consulta, Alterações, Ações Específicas
CAMADA DE DOMÍNIO
Classes de negócio, tipos, constantes específicas ao negócio, implementação dos serviços
CAMADA DE BASE DE DADOS
(DAO, MAPEAMENTO O/R etc)
Classes que tratam diretamente com o meio de armazenamento
CAMADA GLOBAL
Constantes, Tipos, Enumerações, Configurações, ClassHelpers, etc – Tudo que é comum à aplicação
DIREÇÃO DE ACESSIBILIDADE


Tier 1
Presentation
Tier 2
Business Logic
Tier 3
Data/Resource
Clients
LAN
Application Servers
Existing Enterprise Information Systems
Resource Manager
Resources (for example; databases)

Exemplos

1. Protocolos de comunicação em camadas
2. Sistemas de banco de dados
3. Sistemas operacionais
4. Sistemas em 3 camadas - MVC

Vantagens/Desvantagens

- Vantagens:
  - Possibilita trabalhar em níveis crescentes de abstração
  - Suportam facilmente a aplicação de melhoramentos
  - Suportam o reuso

- Desvantagens:
  - Nem todos os sistemas se adequam a essa organização
  - Requisitos de performance podem levar a quebra das regras de organização das camadas

Repositórios

Estrutura de *blackboard*

Introdução

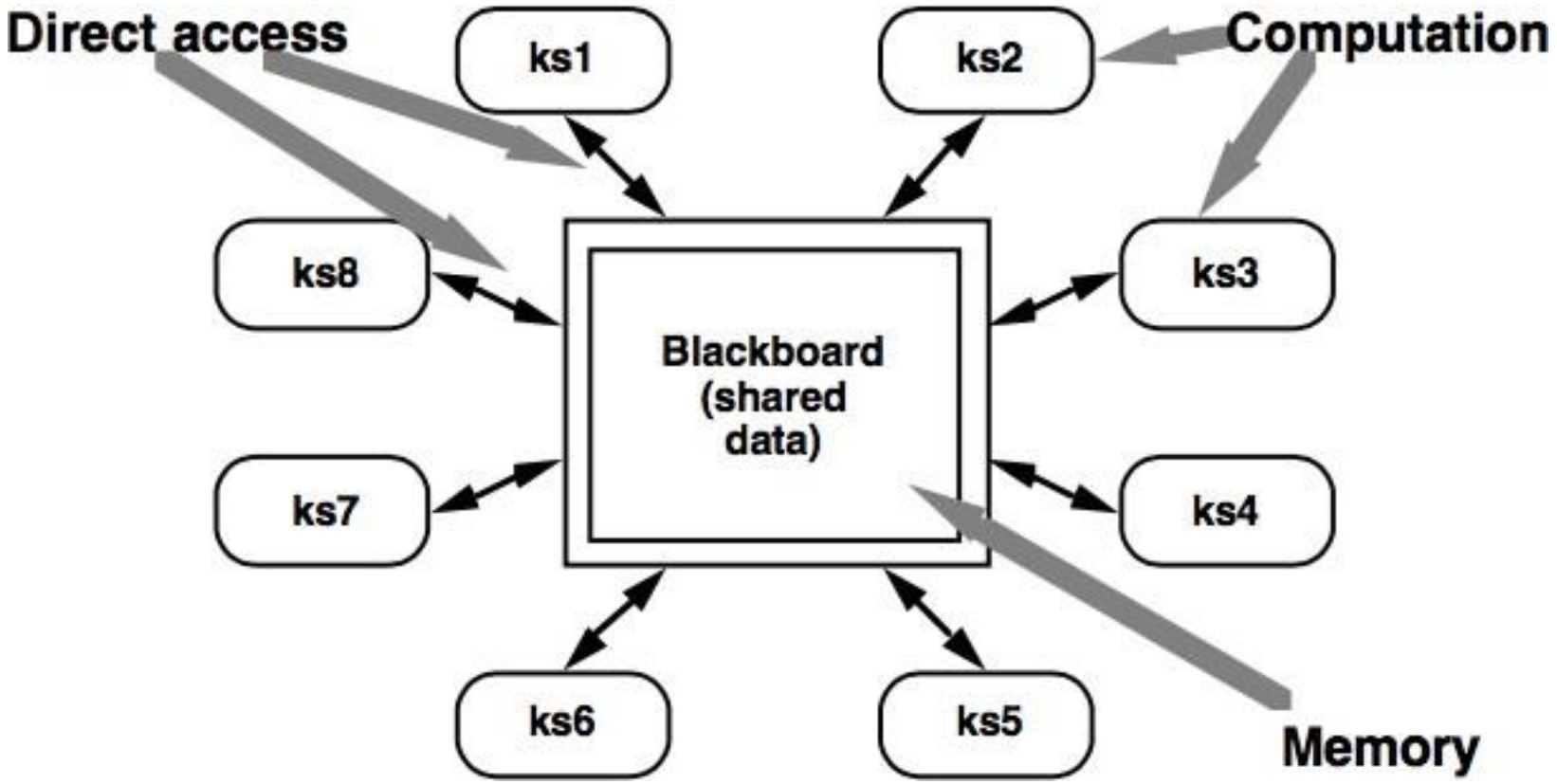
- Há dois tipos distintos de componentes:
  - Uma “estrutura de dados” central (repositório) representando o estado corrente do sistema
  - Uma coleção de componentes independentes que operam com essa “estrutura de dados” central

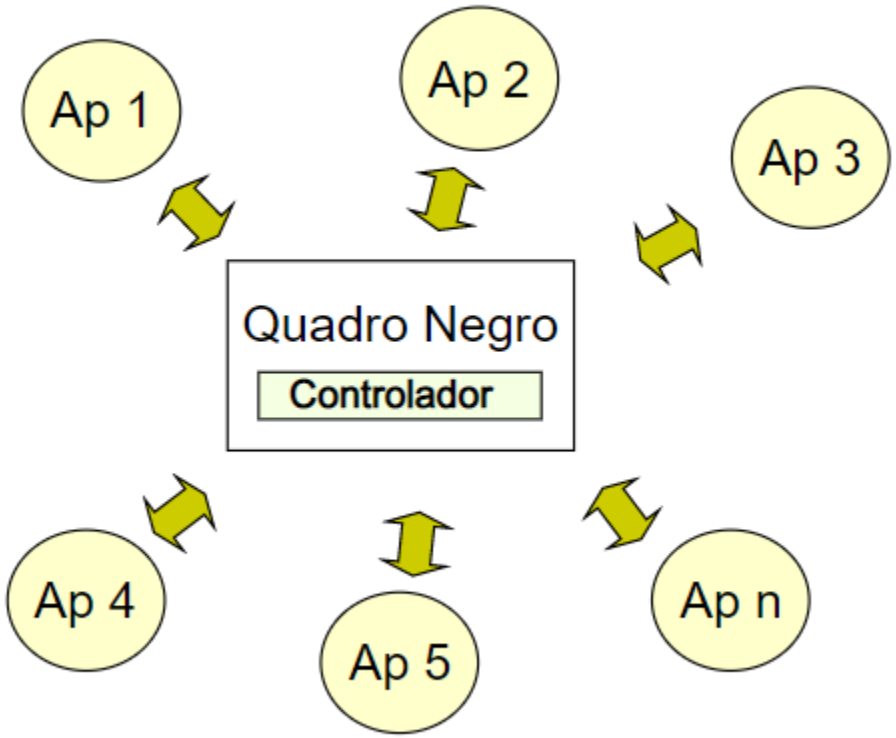
Componentes

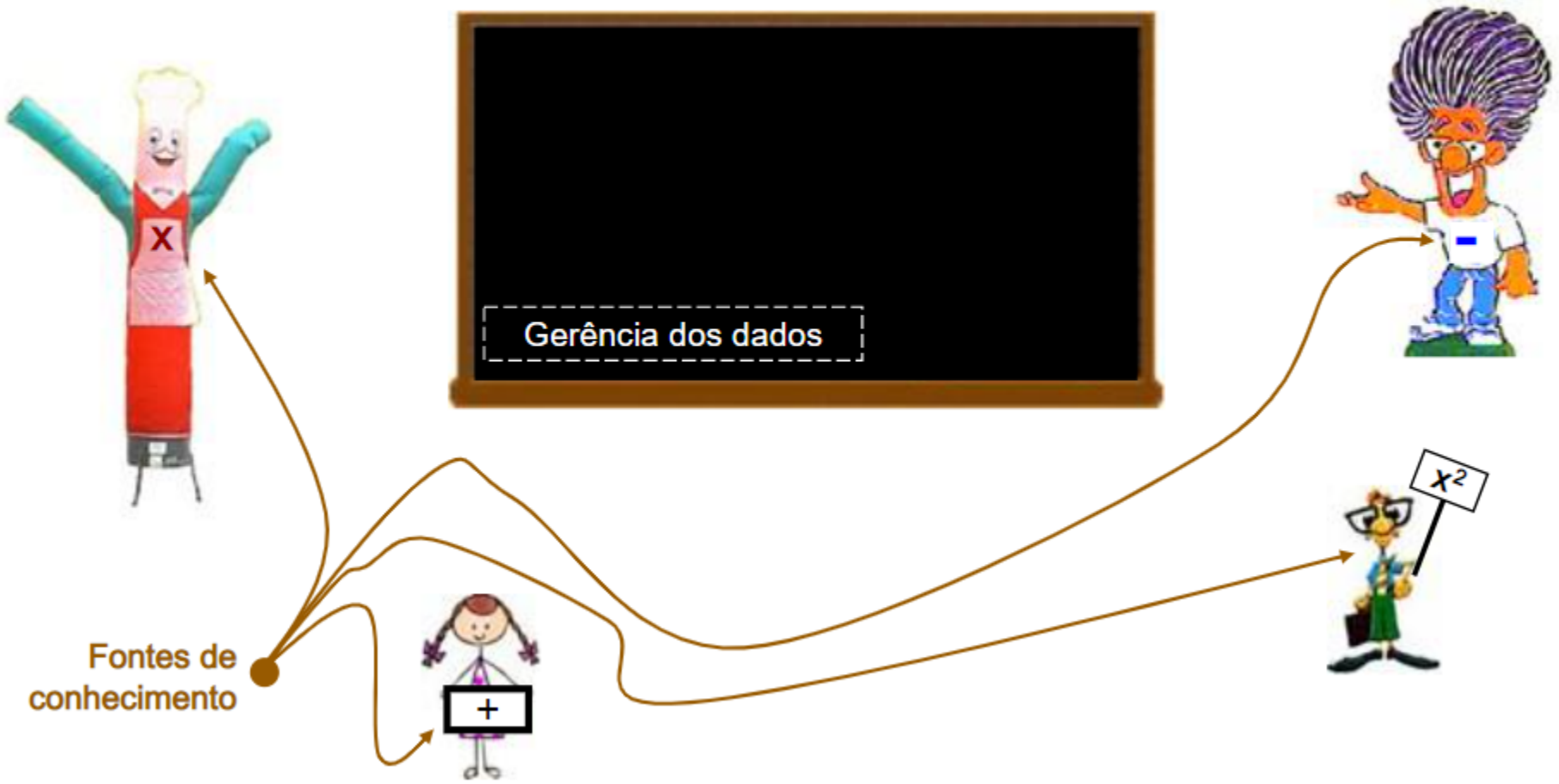
1. Fontes de conhecimento
   - Fontes separadas e independentes – acessadas através do *blackboard* – cada qual resolvendo aspectos específicos do problema

2. A estrutura de dados blackboard
   - Armazena o estado do sistema (dados)

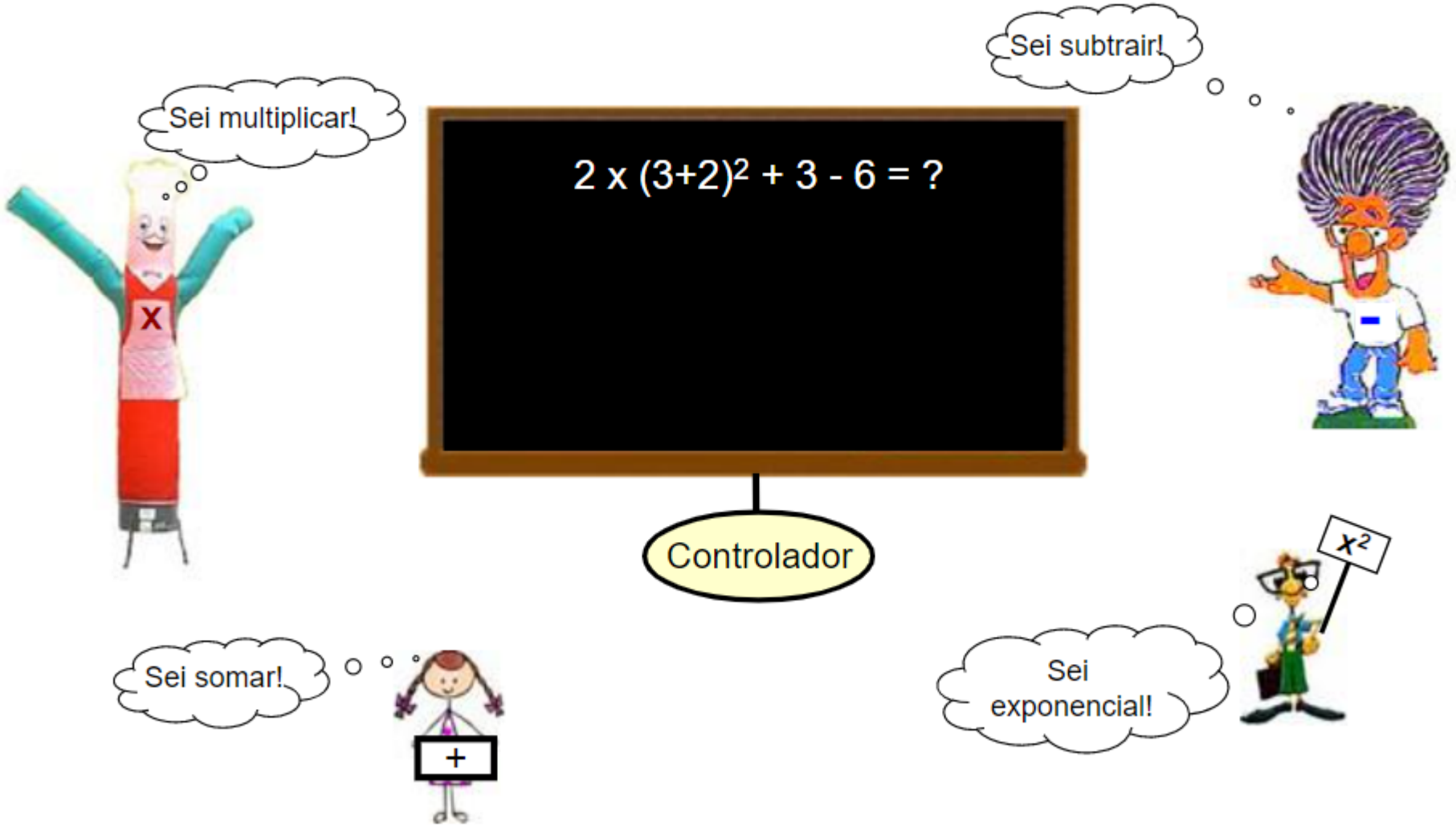
3. Controle
   - Monitora mudanças no *blackboard* e decide ações


Direct access
ks1
ks2
Computation
ks8
ks3
Blackboard
(shared
data)
ks7
ks4
ks6
ks5
Memory

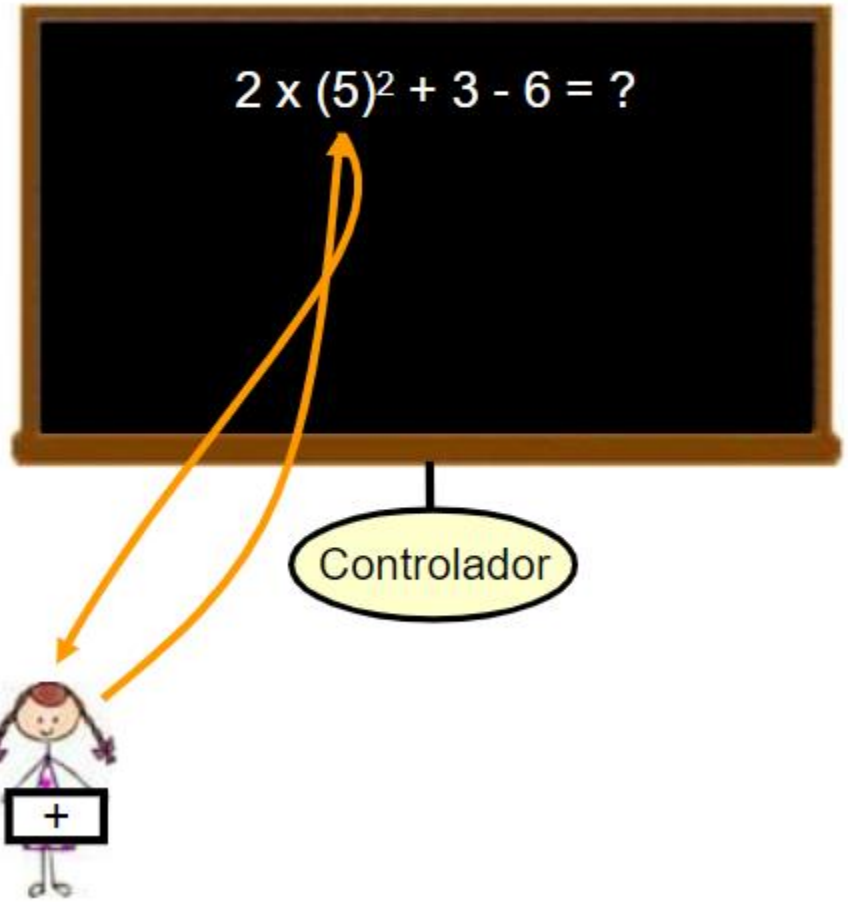

Ap 1
Ap 2
Ap 3
Quadro Negro
Controlador
Ap 4
Ap 5
Ap n



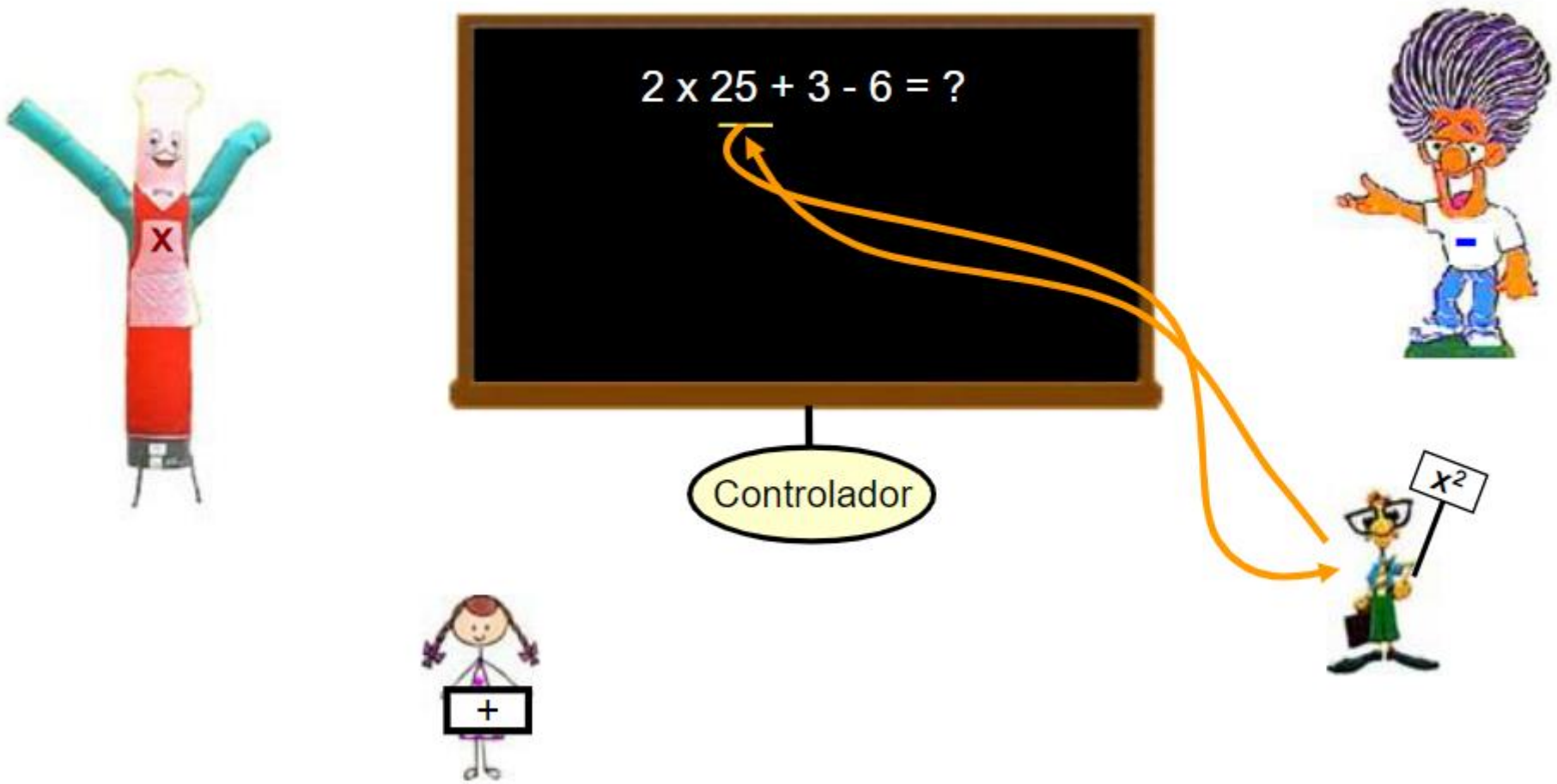
Fonte: Slides "Estilos Arquiteturais" - Professora Silvia Regina Vergilio



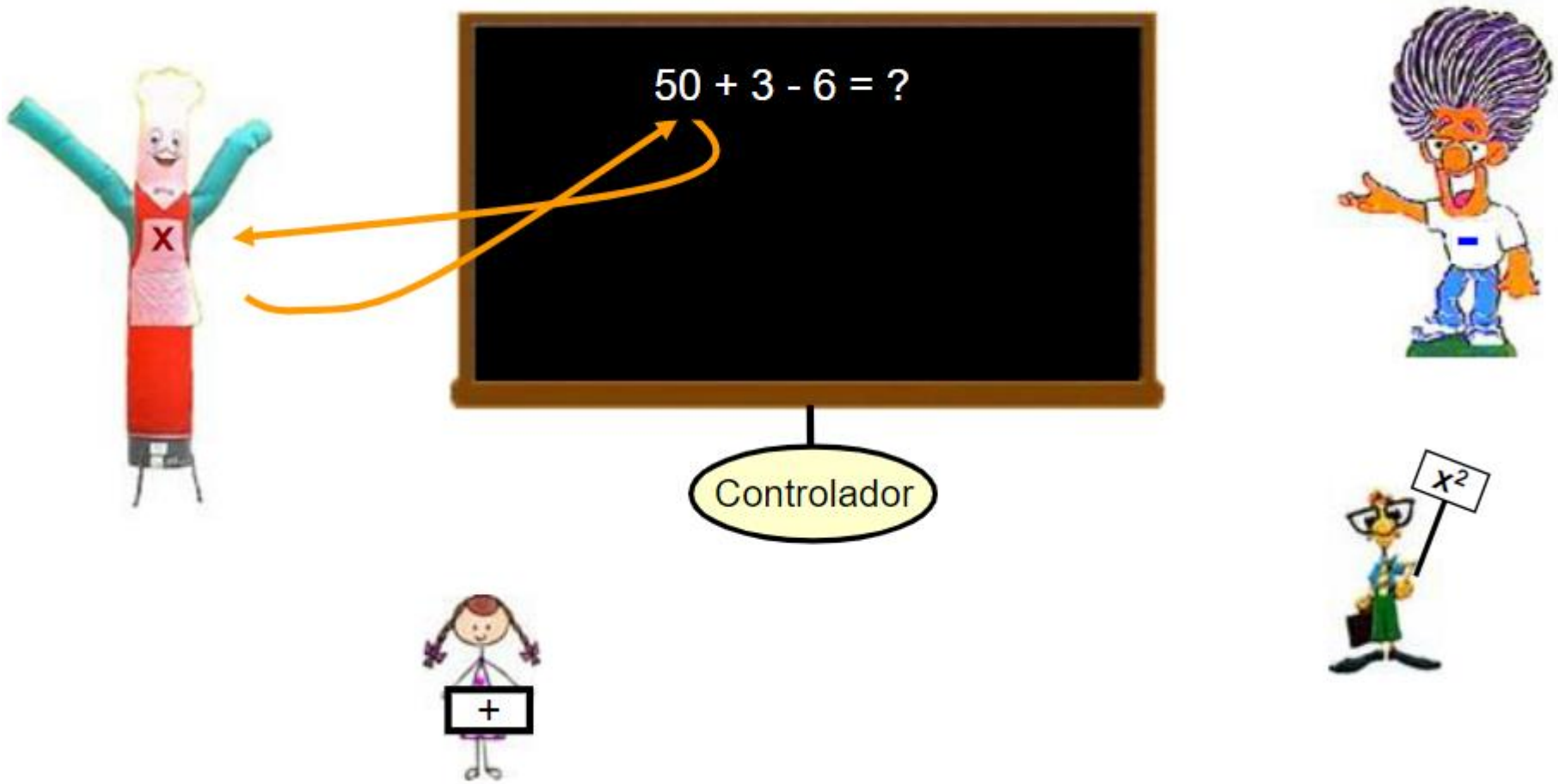
Fonte: Slides "Estilos Arquiteturais" - Professora Silvia Regina Vergilio



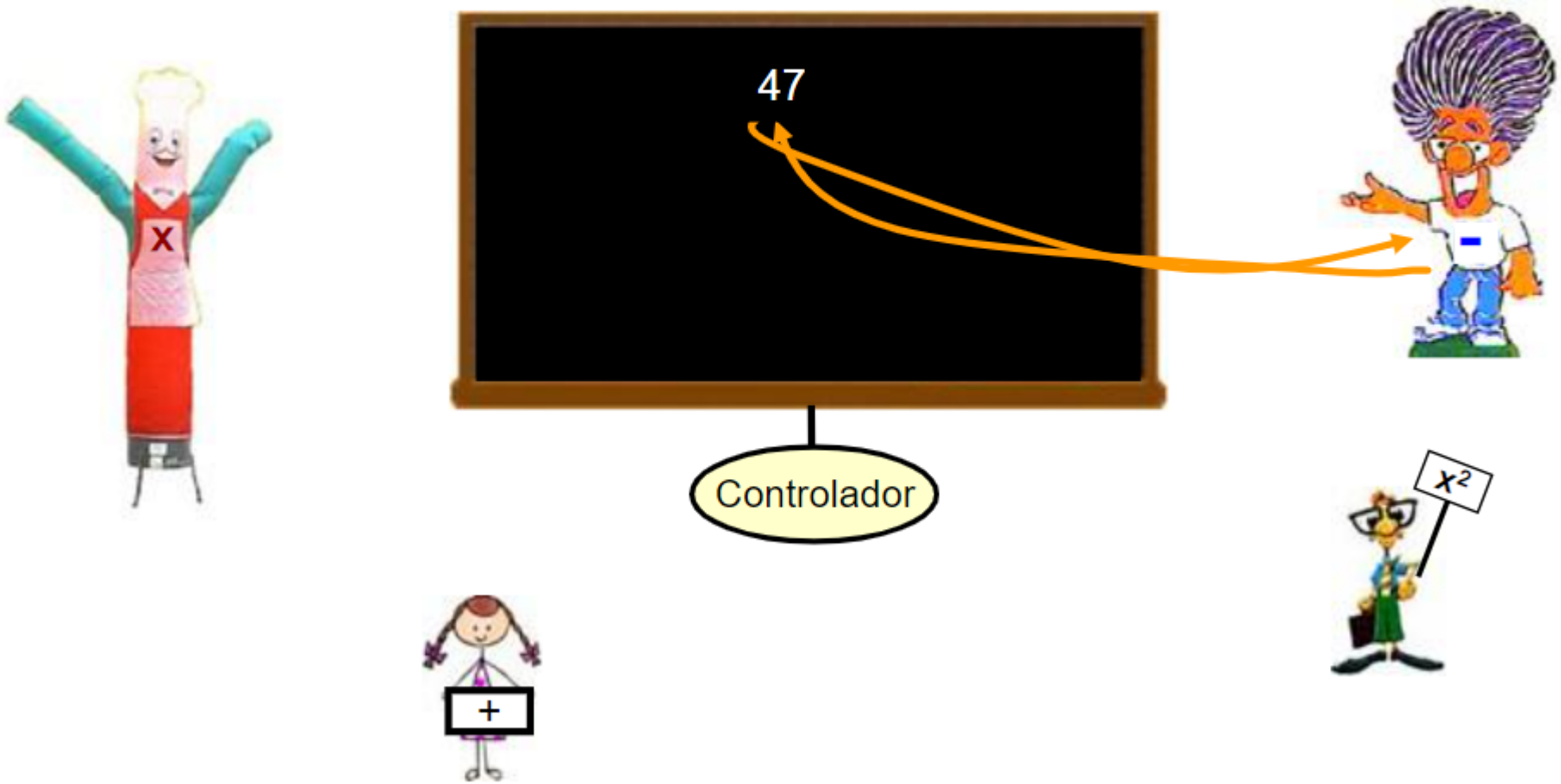
Fonte: Slides "Estilos Arquiteturais" - Professora Silvia Regina Vergilio



Fonte: Slides "Estilos Arquiteturais" - Professora Silvia Regina Vergilio



Fonte: Slides "Estilos Arquiteturais" - Professora Silvia Regina Vergilio



Fonte: Slides "Estilos Arquiteturais" - Professora Silvia Regina Vergilio

Usados em:

- Sistemas que não possuem estratégias de soluções determinísticas conhecidas e são baseados em soluções aproximadas ou parciais

- Problemas que podem ser decompostos em subproblemas e abrangem muitos domínios de conhecimento

Exemplos

- Sistemas de reconhecimento de fala e padrões
- Sistemas com acesso compartilhado a dados por “agentes” fracamente acoplados
- Sistemas compiladores (tabelas de símbolo, *abstract syntax tree*, etc.)

Vantagens

- Reúso de conhecimentos
- Suporta mudanças e manutenção
- Ajuda a resolver problemas de experimentação

Desvantagens

- Nenhuma boa solução é garantida
- Dificuldade em estabelecer uma boa estratégia de controle
- Baixa eficiência e alto esforço de desenvolvimento
- Não suporta paralelismo

Interpretadores orientados a Tabela

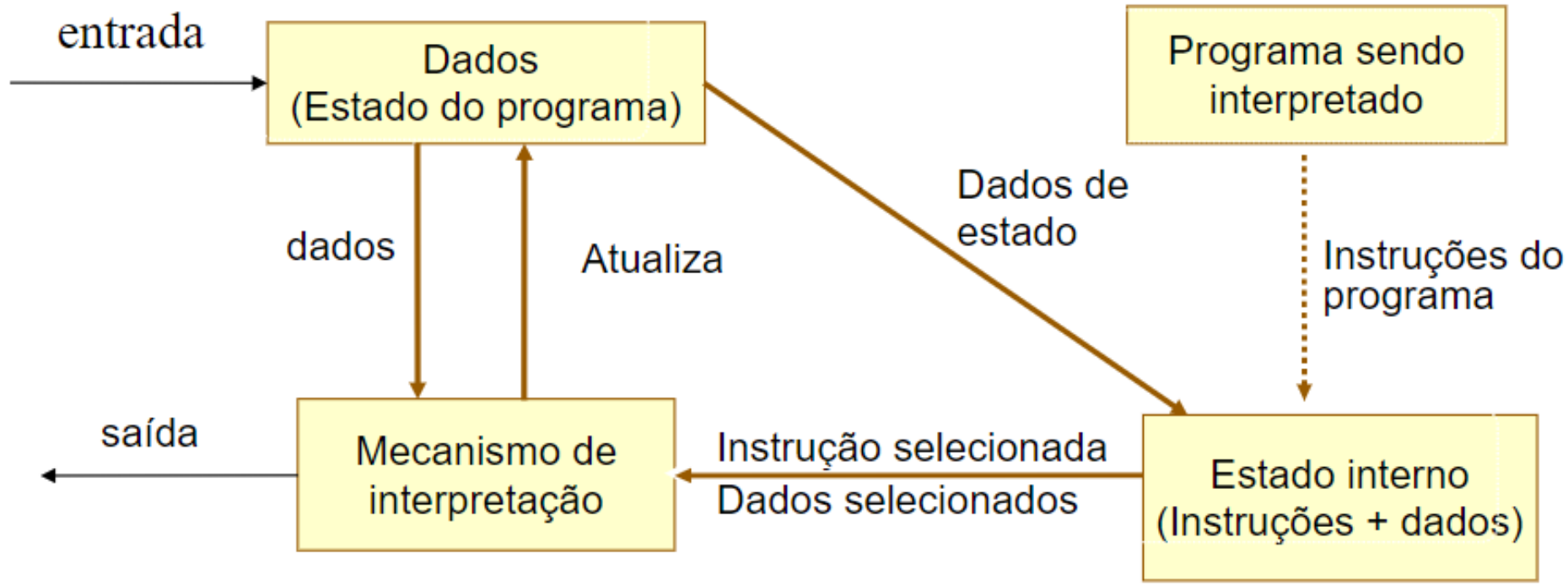
Máquina Virtual (Virtual Machine)

Virtual Machine

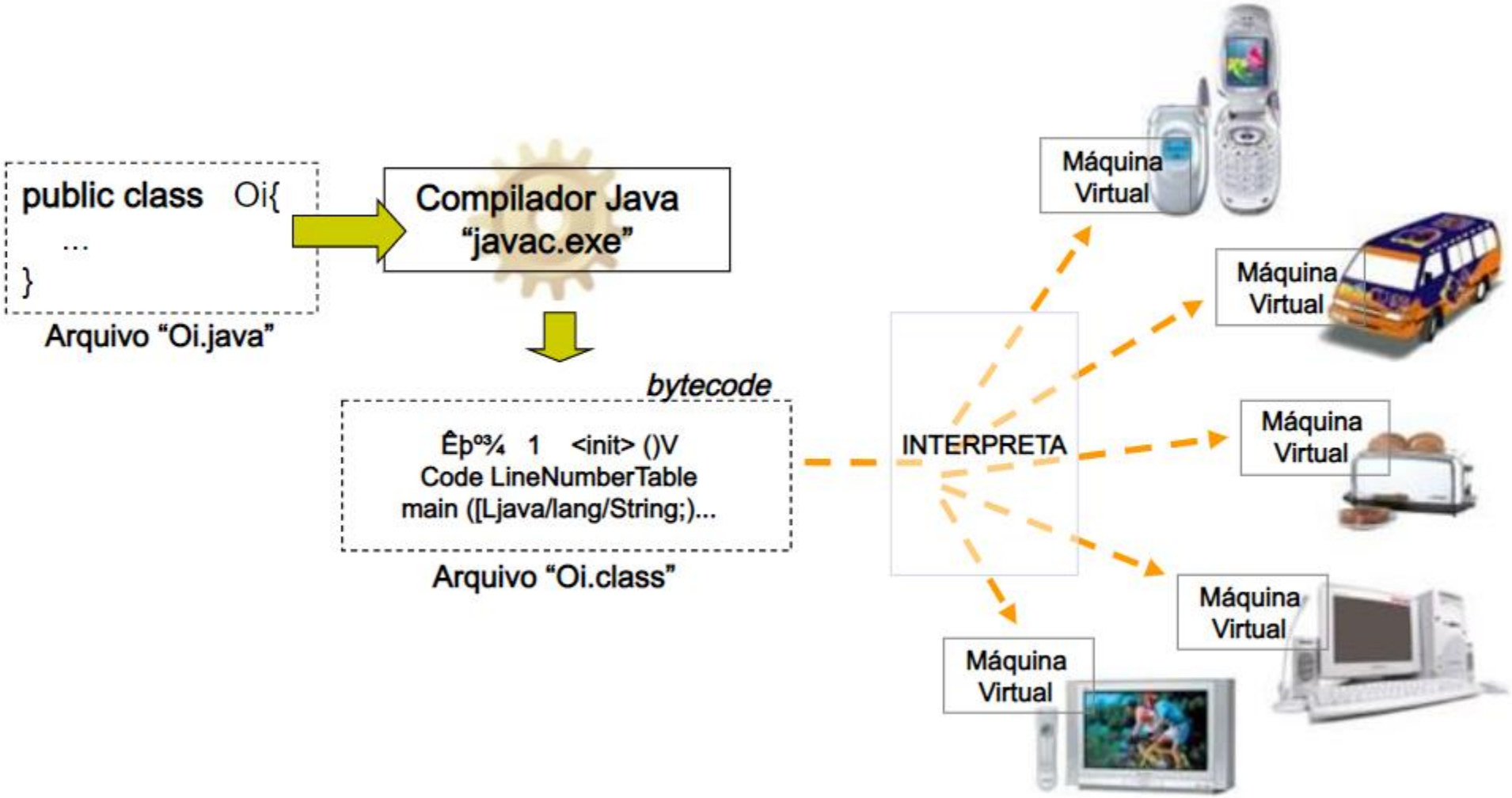
- Na organização de um interpretador uma máquina virtual é produzida em software

- Quatro componentes básicos:
  1. Motor de interpretação
  2. Memória que contém o código a ser interpretado
  3. Representação do estado do interpretador
  4. Representação do estado corrente do “programa” sendo interpretado

- A máquina tenta preencher a lacuna que existe entre o que o programa precisa e o que o hardware disponibiliza



Fonte: Slides "Estilos Arquiteturais" - Professora Silvia Regina Vergilio



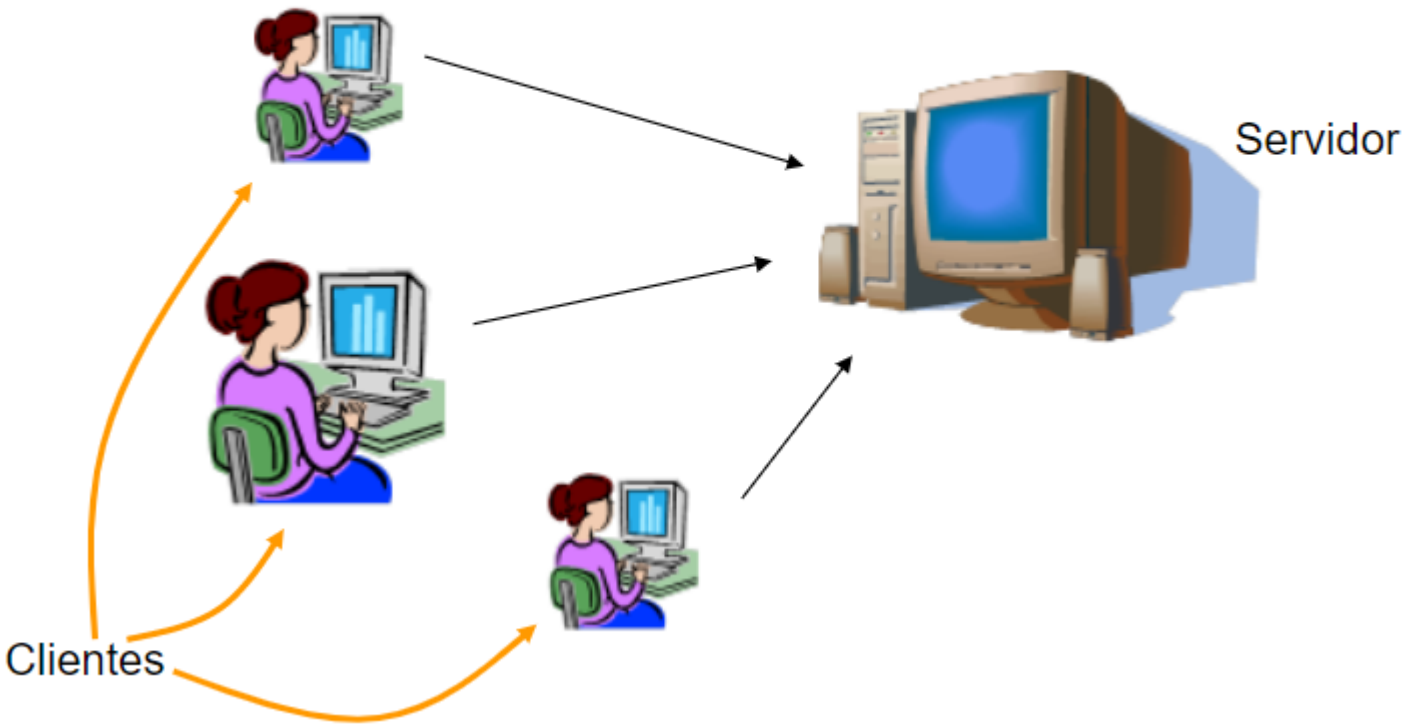
Fonte: Slides "Estilos Arquiteturais" - Professora Silvia Regina Vergilio

Cliente - Servidor

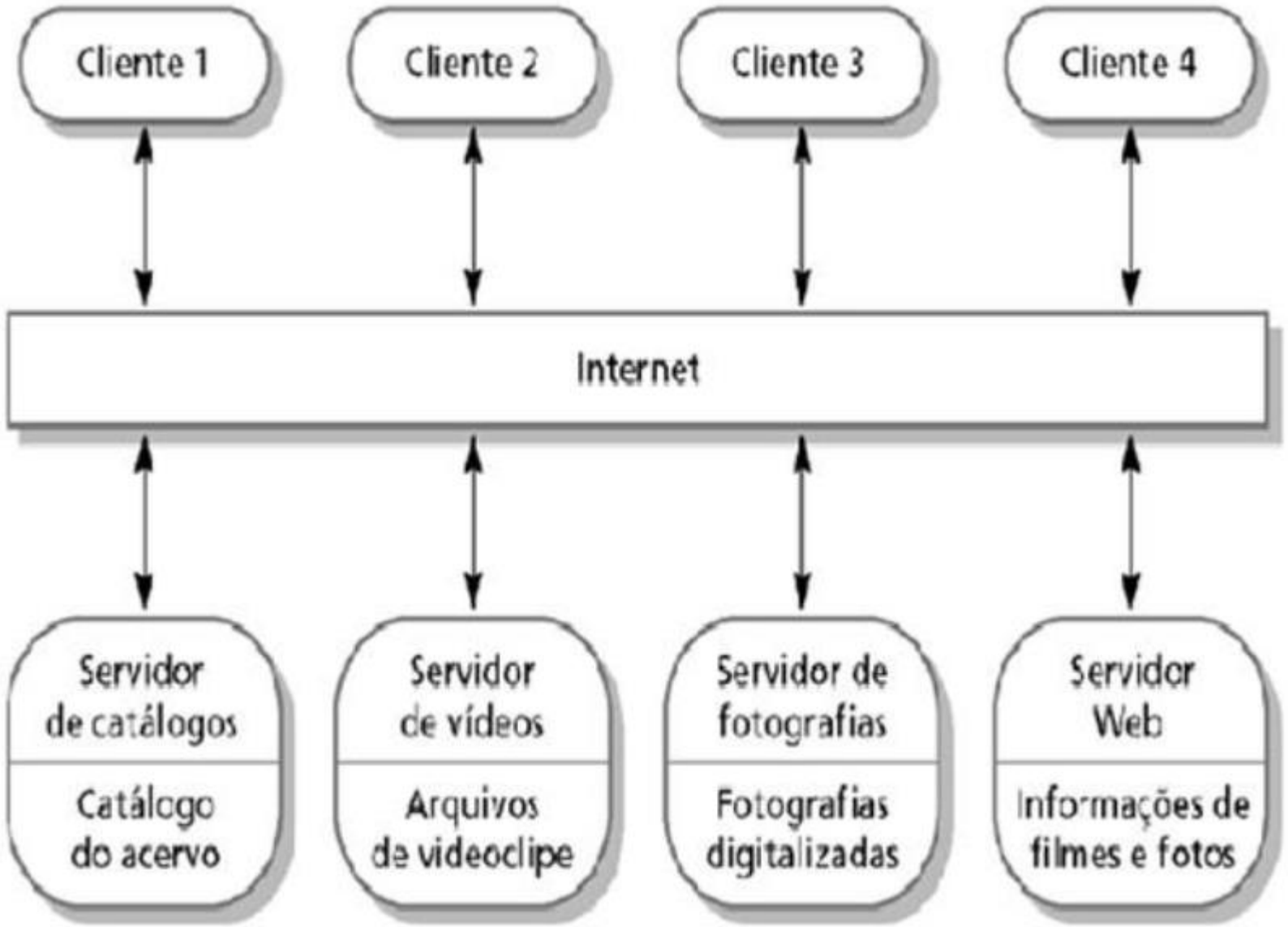
Cliente-Servidor

- Mostra como dados e processamento são distribuídos por uma variedade de componentes:
  - Servidores independentes que fornecem serviços tais como impressão, transferência de arquivos, gerenciamento de dados, etc..
  - Clientes utilizam esses serviços

- Clientes e servidores normalmente se comunicam através de uma rede
  - Diversas tecnologias de comunicação são possíveis



Fonte: Slides "Estilos Arquiteturais" - Professora Silvia Regina Vergilio


Cliente 1
Cliente 2
Cliente 3
Cliente 4
Internet
Servidor de catálogos
Catálogo do acervo
Servidor de vídeos
Arquivos de videoclipe
Servidor de fotografias
Fotografias digitalizadas
Servidor Web
Informações de filmes e fotos

Cliente-Servidor

- Vantagens:
  - Separação de interesses
  - Inerentemente distribuído: pode haver balanceamento de carga, tolerância a falhas
  - É fácil adicionar novos servidores ou atualizar servidores existentes
  - Escalabilidade: aumentando a capacidade computacional do servidor

Cliente-Servidor

- Desvantagens:
  - Gerenciamento redundante em cada servidor
  - Nenhum registro central de nomes e serviços – pode ser difícil descobrir quais servidores e serviços estão disponíveis
  - Requisições e respostas casadas
  - Falhas no servidor

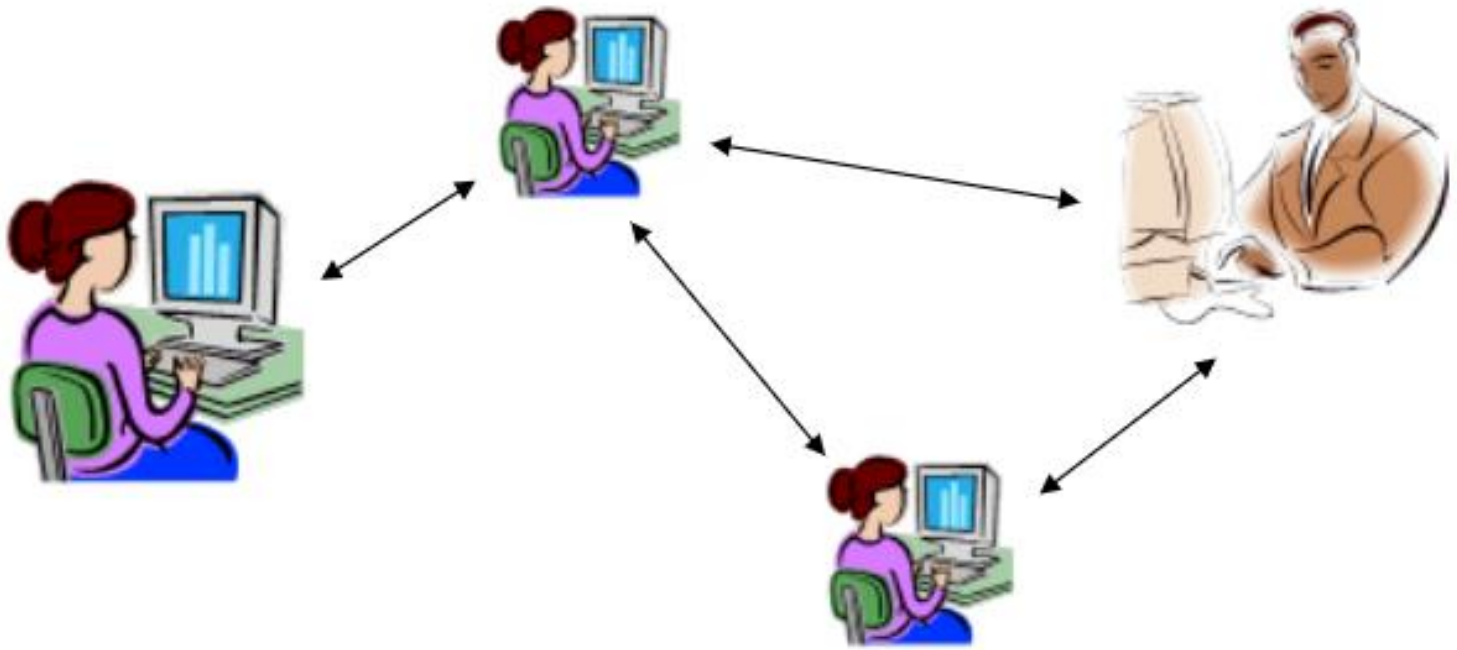
Ponto a Ponto (P2P)

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”

Vantagem: Reduz problemas de falhas

Desvantagem: Aumenta o tempo de consulta

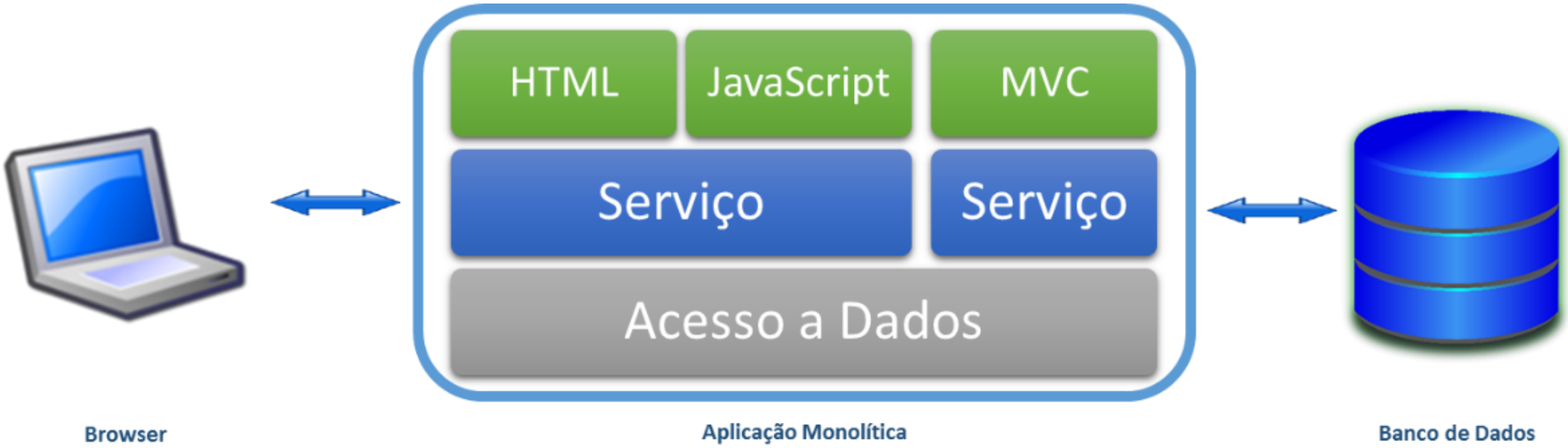


Fonte: Slides "Estilos Arquiteturais" - Professora Silvia Regina Vergilio

Monolítica

Monolítica

- Todas as funções do negócio estão implementadas em um único processo
- Ao longo do tempo o sistema vai crescendo e se tornando cada vez mais complexo
- Dificuldade para manutenção e escalabilidade

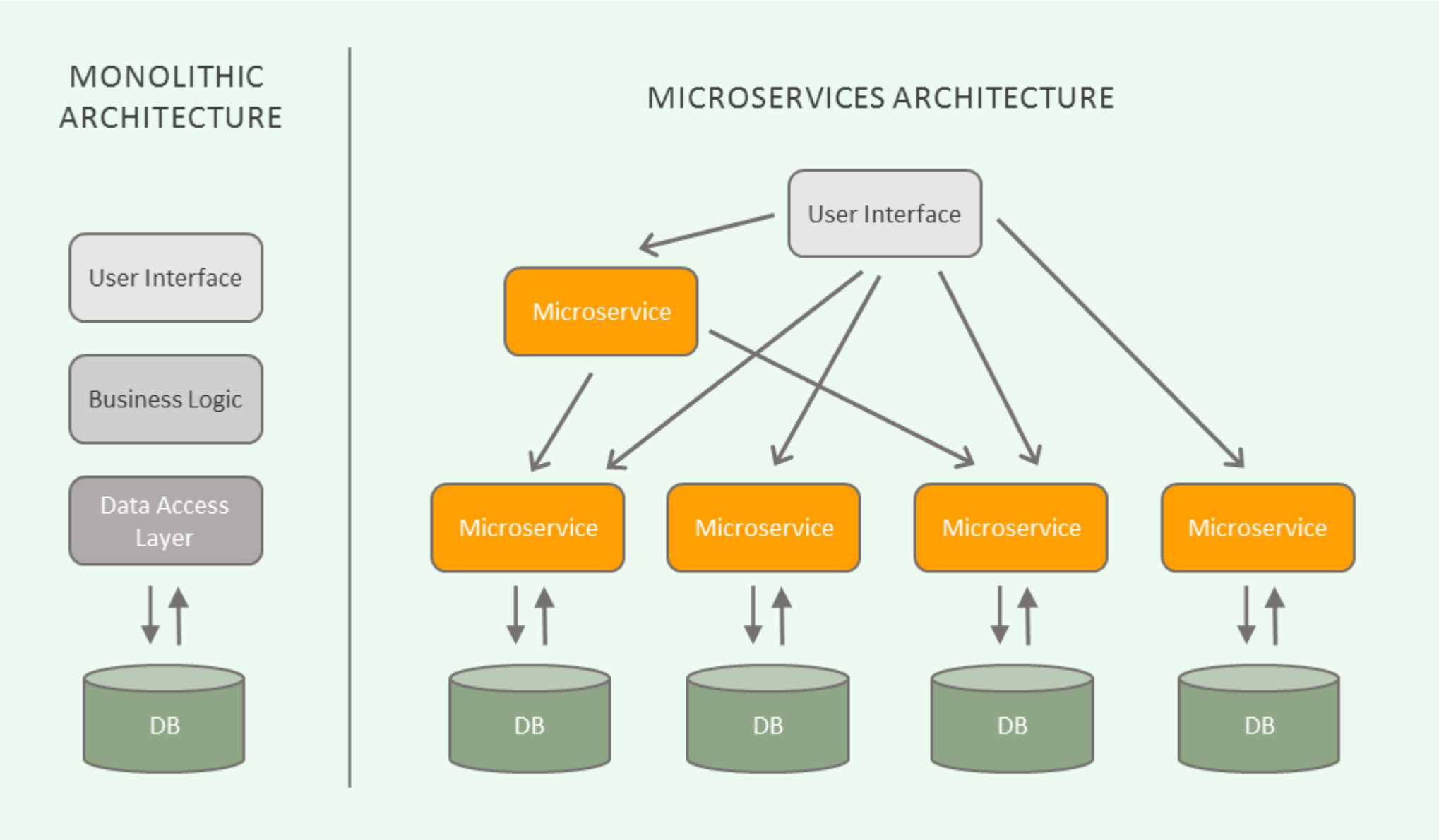


Fonte: https://www.opus-software.com.br/micro-servicos-arquietura-monolitica/

Microserviços

Microserviços

- É utilizada para desenvolver uma aplicação como um conjunto de pequenos serviços, cada um funcionando em seu próprio processo
- Cada serviço é desenvolvido em torno de um conjunto de regras de negócio específico, e é implementado de forma independente
- Com essas características, consegue-se quebrar as barreiras impostas pela arquitetura monolítica.



Fonte: https://elo7.dev/microservicos-rest//

Outras Arquiteturas Familiares

Outros Estilos

1. Processos distribuídos – caracterizados pelos protocolos de comunicação entre processos
2. Programa principal/sub-rotinas
3. Arquiteturas específica de domínio (ex.: veículos)
4. Sistemas de transição de estados
5. Sistemas de controle de processos
6. Arquiteturas heterogêneas...

Exercício

Pesquise e cite um exemplo real de cada estilo arquitetural abaixo:

a) Pipes and Filters
b) Baseada em Eventos
c) Sistemas em Camadas (MVC)
d) Repositório Blackboard
e) Cliente-Servidor
f) Ponto a Ponto (P2P)
g) Microserviços

Para cada exemplo, escreva uma breve descrição e inclua o que for necessário (*imagens, diagramas...*) para complementar que o seu exemplo se encaixa dentro do estilo arquitetural.

Importante: Os exemplos devem ser diferentes dos citados no material.

Fontes de Consulta

- Slides "Estilos Arquiteturais" - Professora Silvia Regina Vergilio
- https://www.opus-software.com.br/micro-servicos-arquietura-monolitica/