Sistemas Distribuídos e Programação Concorrente

Padrões Arquiteturais

Prof. MSc. Rodrigo Daniel Malara

UNIARA ONLINE

O que será abordado

- O que é um Padrão?
- Catálogo de Padrões de Projeto Arquiteturais
  - Multi-Layer ou Multi-Camadas
  - Multi-Tier ou Multi-Níveis
  - Cliente-Servidor
  - Shared Data – Dados compartilhados
  - Model-View-Controller MVC
  - Service Oriented Architecture SOA

O que é um Padrão?

Um padrão arquitetônico estabelece uma relação entre:

- *Um contexto. Uma situação recorrente e comum no mundo que gera um problema.*
- *Um problema. O problema, apropriadamente generalizado, surge no contexto dado.*

O que é um Padrão?

- *Uma solução. Uma resolução arquitetônica bem sucedida para o problema, apropriadamente abstraída. A solução para um padrão é determinada e descrita por*:
  - Um conjunto de tipos de elementos (por exemplo, repositórios de dados, processos e objetos)
  - Um conjunto de mecanismos de interação ou conectores (por exemplo, chamadas de método, eventos ou barramentos de mensagens)
  - Um layout topológico dos componentes
  - Um conjunto de restrições semânticas que abrangem topologia, comportamento de elementos e mecanismos de interação

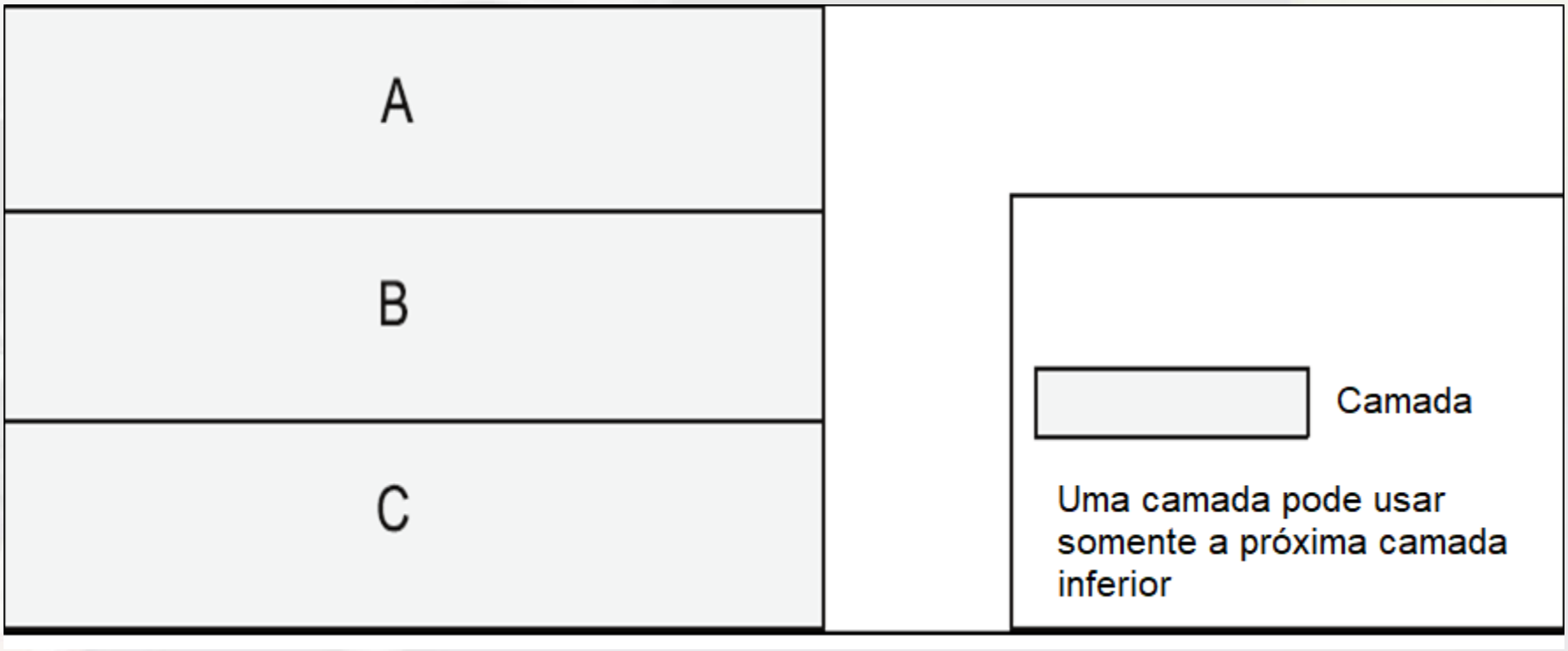
Padrão Arquitetural

Multi-Layer

Padrão Multi-Layer

- **Contexto**:
  - Sistemas complexos
  - Necessidade de desenvolver e evoluir partes do sistema de forma independente.
  - Desenvolvedores do sistema precisam de uma separação clara e bem documentada das preocupações, para que os módulos do sistema possam ser desenvolvidos e mantidos de forma independente.
- **Problema**: O software precisa ser segmentado de tal forma que os módulos possam ser desenvolvidos e evoluídos separadamente com pouca interação entre as componentes, suportando portabilidade, modificabilidade e reutilização.
- **Solução**: Para alcançar essa separação de preocupações:
  - Dividir o software em unidades chamadas camadas.
  - Cada camada é um agrupamento de módulos que oferece um conjunto coeso de serviços.
  - O uso deve ser unidirecional.
  - Cada partição é exposta através de uma interface pública.

Exemplo de padrão Multi-Layer



Solução do padrão Multi-Layer

- **Visão geral**: O padrão em camadas define camadas (agrupamentos de módulos que oferecem um conjunto coeso de serviços) e uma relação unidirecional permitida de uso entre as camadas.
- **Elementos**: Camada, uma espécie de módulo. A descrição de uma camada deve definir quais módulos a camada contém.
- **Relações**: Permitido usar. O design deve definir quais são as regras de uso da camada e quaisquer exceções permitidas.

Solução do padrão Multi-Layer

- **Restrições**:
  - Cada modulo de software é alocado exatamente em uma camada.
  - Há pelo menos duas camadas (mas geralmente há três ou mais).
  - As relações permitidas de uso não devem ser circulares (ou seja, uma camada inferior não pode usar uma camada acima).
- **Fraquezas**:
  - A adição de camadas adiciona custo e complexidade a um sistema.
  - As camadas contribuem com uma penalidade de desempenho.

Padrão Multi-Layer - Exemplos

- Modelo ISO/OSI (Open Systems Interconnect)
  - Arquitetura para interconexão em rede de dispositivos
- Arquitetura interna de sistemas operacionais
  - Principalmente ao lidar com dispositivos de E/S
  - Windows, Linux, etc
- Arquitetura proposta para aplicações desenvolvidas usando o Java Enterprise Edition – JEE

Camada de Apresentação – Interface com o Usuário	
Camada de Negócio – Funcionalidades realizadas pelo sistema	
Camada de Persistência Armazenamento de dados	Camada de Integração entre Sistemas

Padrão Arquitetural

Multi-Tier

Padrão Multi-Tier

Multi-Tier ou Multi-Níveis

- Diferenciar o Multi-Layer (ou "Em Camadas") do Multi-Tier (Multi-Níveis)
- A tradução de Tier e Layer em português pode ser a mesma: camadas
  - Mas esses termos representam conceitos diferentes
  - Tier também pode ser traduzido como Nível
- Layers são uma forma de organizar seu código.
  - Exemplo: apresentação, negócios, dados
  - No Multi-Layer não fica definido em que computador cada camada será executada
  - Layers são divisões lógicas do código de acordo com suas funções.
- Tiers no entanto é sobre onde o código executa.
  - Tiers são os locais onde as Layers são colocadas e onde elas rodam.
  - Tier portanto é o deploy físico onde as Layers executam.

Layers ou Camadas → Camadas Lógicas

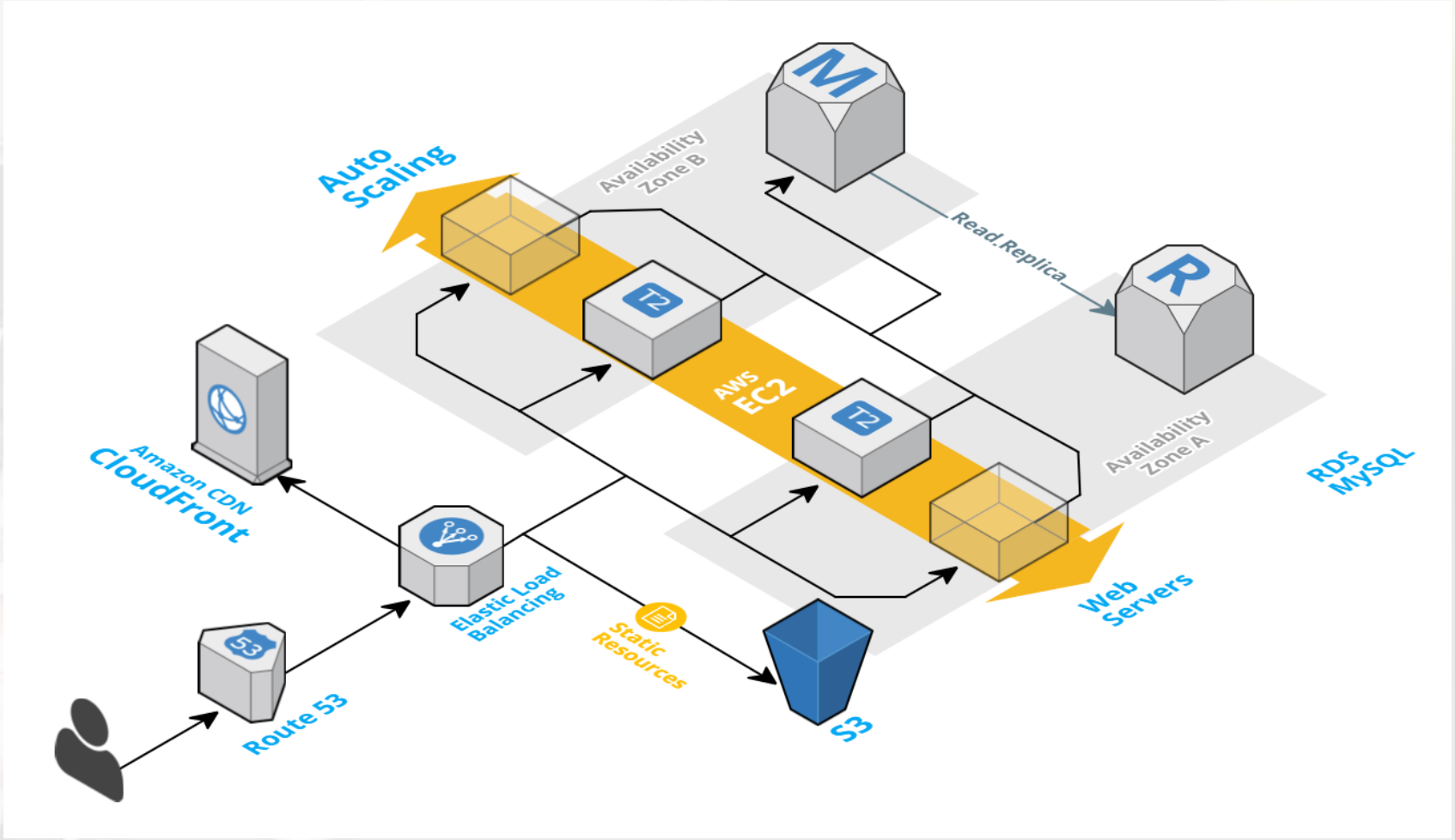
Tiers ou Níveis → Camadas Físicas

Padrão Multi-Tier

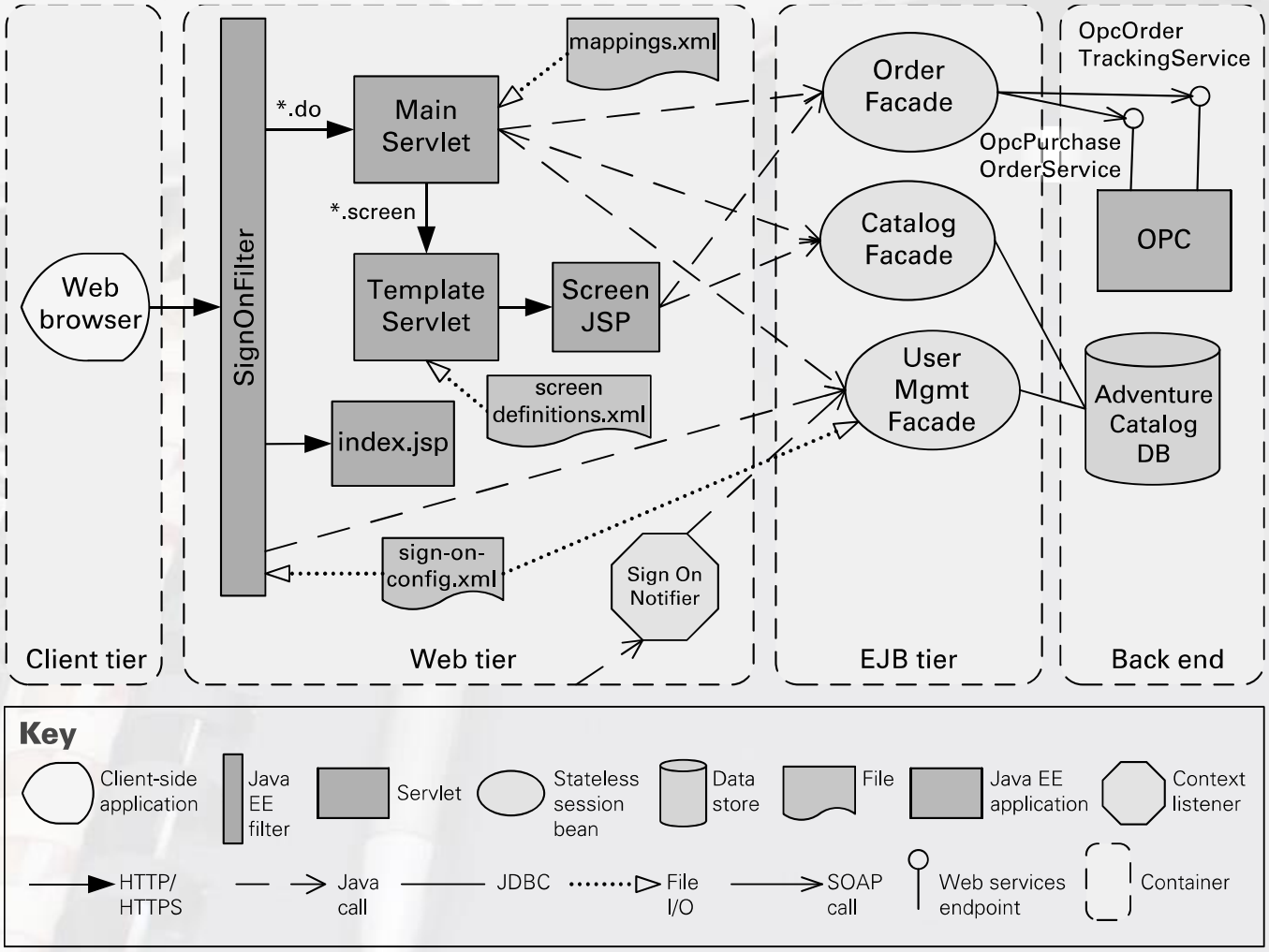
- **Contexto**: Em uma implantação distribuída, muitas vezes há a necessidade de distribuir a infraestrutura de um sistema em subconjuntos distintos.
- **Problema**: Como podemos dividir o sistema em uma série de estruturas de execução computacionalmente independentes — grupos de software e hardware — conectados por algumas mídias de comunicação?
- **Solução**: As estruturas de execução de muitos sistemas são organizadas como um conjunto de agrupamentos lógicos de componentes.

  Cada agrupamento é chamado de nível.

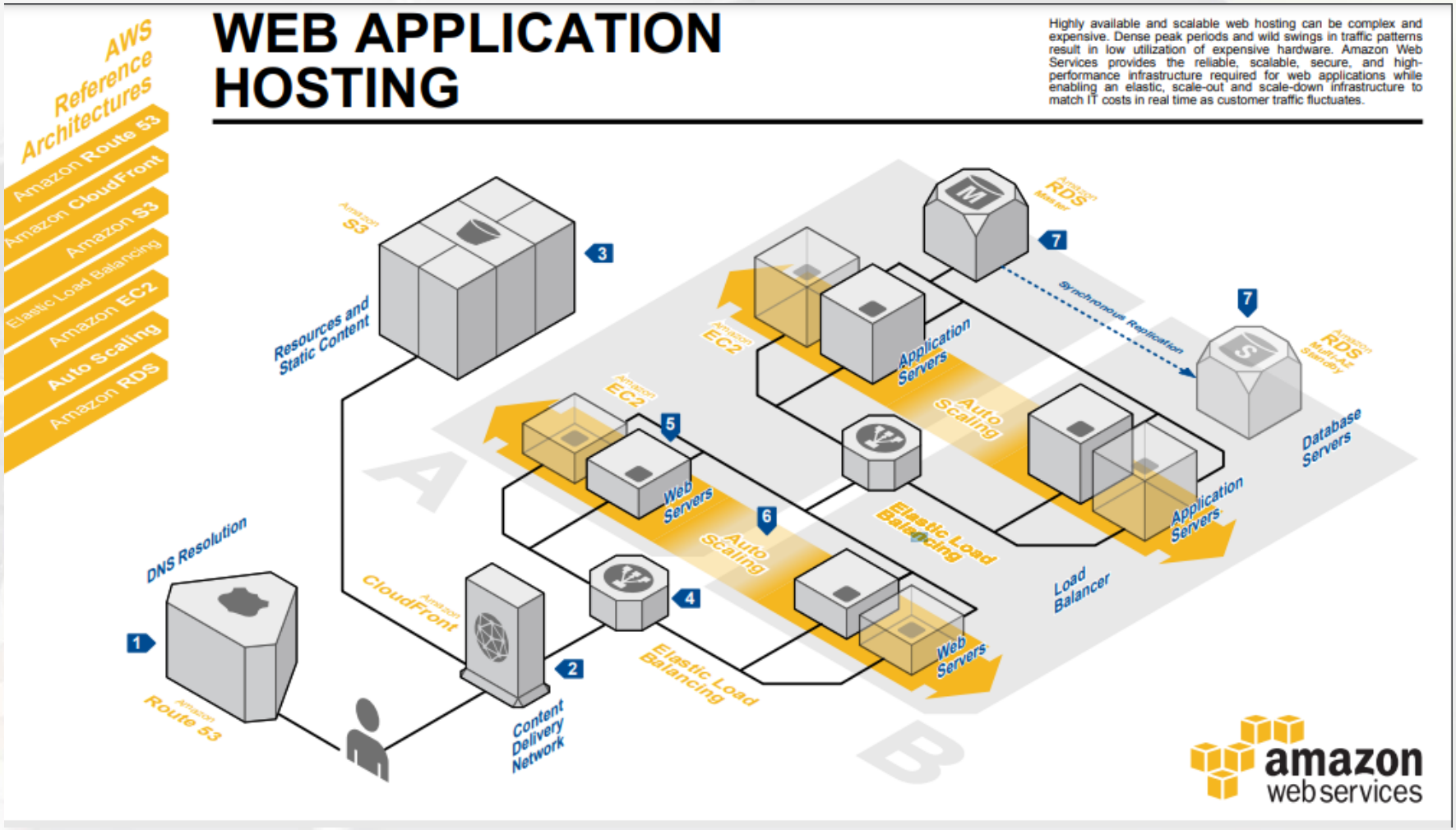
Exemplo Multi-Tier – 3 Níveis



Exemplo Multi-Tier – 4 Níveis



Exemplo Multi-Tier – 4 Níveis



Multi-Tier - Solução

- **Visão geral:** As estruturas de execução de muitos sistemas são organizadas como um conjunto de agrupamentos lógicos de componentes. Cada agrupamento é chamado de nível.
- **Elementos**:
  - *Tier,* que é um agrupamento lógico de componentes de software.
- **Relações**:
  - *Faz parte de:* agrupar componentes em níveis.
  - *Comunica-se com,* para mostrar como os níveis e os componentes que eles contêm interagem entre si.
  - *Alocado para,* no caso de que os níveis mapeiam para plataformas de computação.
- **Restrições**: Um componente de software pertence exatamente a um nível.
- **Fraquezas**: Custo de implantação elevado e complexidade substanciais.

Multi-Tier

- É um padrão de arquitetura importante para sistemas distribuídos
  - Bastante aplicável para provedores de computação em Nuvem
  - Procure por ‘arquiteturas de referência’ para o provedor desejado
- Atingimento de qualidades sistêmicas como alta-disponibilidade e escalabilidade utilizando arquiteturas multi-níveis em nuvem com custos proporcionais ao uso dos recursos computacionais
- Mais detalhes em Saiba Mais

Padrão Arquitetural

Client-Server

Client-Server Pattern

- Também conhecido como Cliente-Servidor é um dos padrões arquiteturais mais utilizados na atualidade.
- **Contexto:**
  - Existem recursos e serviços compartilhados que um grande número de clientes distribuídos deseja acessar, e para os quais desejamos controlar o acesso e/ou a qualidade do serviço.
- **Problema:**
  - Ao gerenciar um conjunto de recursos e serviços compartilhados, podemos promover modificabilidade e reutilização, fatorando serviços comuns e tendo que modificá-los em um único local, ou um pequeno número de locais.
  - Queremos melhorar a escalabilidade e a disponibilidade centralizando o controle desses recursos e

    serviços, enquanto distribuímos os recursos

    em vários servidores físicos.

Client-Server Pattern

- **Solução:**
  - Os clientes interagem solicitando a realização de serviços aos servidores.
  - Servidores prestam um conjunto de serviços a clientes
  - Alguns servidores também podem atuar como clientes ao mesmo tempo.
  - Pode haver um servidor central ou vários distribuídos para maior tolerância a falhas.

Client-Server - Solução - 1

- **Visão geral**: Os clientes iniciam interações com servidores, invocando serviços conforme necessário e aguardando os resultados dessas solicitações.
- **Elementos**:
  - *Client,* um componente que invoca serviços disponibilizados por servidores.
  - *Server: é* um componente que presta serviços aos clientes. Os serviços são disponibilizados através de portas de comunicação.
- *Conector de requisição/resposta:*

  Deve funcionar para chamadas locais

  ou remotas e dados criptografados.

Client-Server – Solução - 2

- **Relações**: A relação de *conexão* associa clientes com servidores.
- **Restrições**:
  - Os clientes estão conectados aos servidores por meio de conectores de solicitação/resposta.
  - Componentes do servidor podem ser clientes de outros servidores.
- **Fraquezas**:
  - O servidor pode ser um gargalo de desempenho.
  - O servidor pode ser um único ponto de falha.
  - Decisões sobre onde localizar funcionalidade (no cliente ou no servidor) são muitas vezes complexas e caras de mudar depois que um sistema foi construído.

Padrão Client-Server – Exemplos

- Sistemas Gerenciadores de Bancos de Dados
  - Oracle
  - MySQL
  - SQL Server
  - …
- Servidores Web
  - Apache HTTPD
- Servidores de E-mails
  - Sendmail
- Servidores de arquivos
  - SFTP
- E muitos outros

Padrão Arquitetural

Shared Data

Padrão Shared-Data

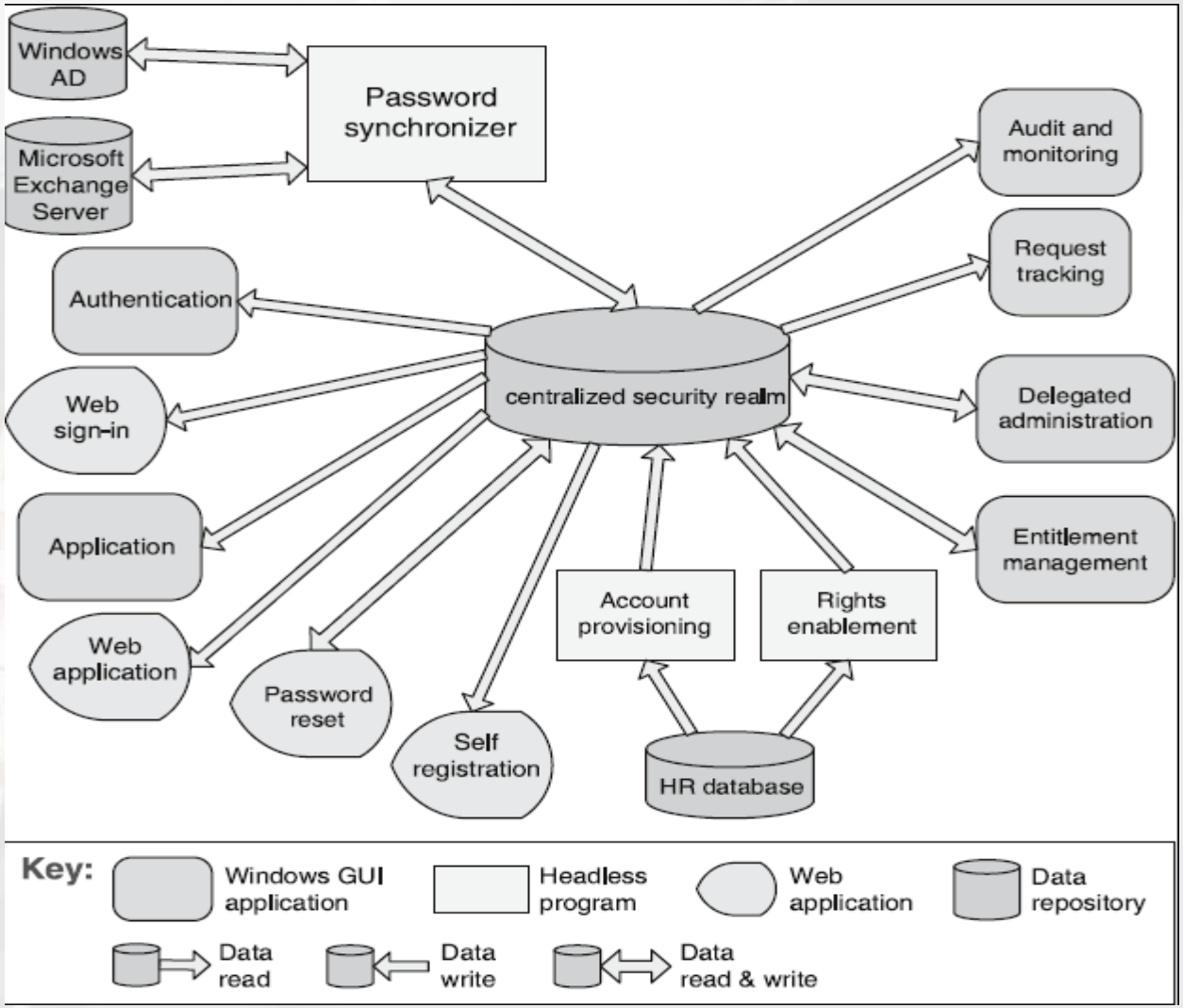
Ou Dados Compartilhados

- **Contexto:** Vários componentes computacionais precisam compartilhar e manipular grandes quantidades de dados. Esses dados não pertencem exclusivamente a nenhum desses componentes.
- **Problema:** Como os sistemas podem armazenar e manipular dados persistentes que são acessados por vários componentes independentes?

Padrão Shared-Data

- **Solução:**
  - No padrão de dados compartilhados, a interação é dominada pela troca de dados persistentes entre múltiplos componentes e pelo menos um armazenamento de dados compartilhados.
  - A troca pode ser iniciada pelos componentes ou pelo data store.
  - O tipo de conector é *leitura e escrita de dados*.

Exemplo Shared Data



Solução - Shared Data - 1

- **Visão geral**: A comunicação entre os acessórios de dados é mediada por um armazenamento de dados compartilhado. O controle pode ser iniciado pelos acessórios de dados ou pelo armazenamento de dados. Os dados são persistentes pelo armazenamento de dados.
- **Elementos**:
  - *Shared-data store ou Depósito de Dados Compartilhados:* As preocupações incluem tipos de dados armazenados, propriedades orientadas ao desempenho dos dados, distribuição de dados e número de clientes permitidos.
  - *Data accessor component ou componente de acesso aos dados*.
  - *Data reading and writing connector ou conector de leitura e escrita de dados*.

Solução - Shared Data - 2

- **Relações**: A relação de conexão determina quais elementos que acessam dados estão conectados aos elementos que armazenam dados.
- **Restrições**: Os elementos que acessam dados interagem apenas com o(s) depósito(s) de dados.

Solução - Shared Data - 2

- **Fraquezas**:
  - O armazenamento de dados compartilhados pode ser um gargalo de desempenho.
  - O meio de armazenamento de dados compartilhados pode ser um ponto único de falha – single point of failure (SPOF).
  - Produtores e consumidores de dados podem estar fortemente acoplados.
  - Acessos concorrentes prejudicam a performance de acesso e a integridade dos dados.
    - Integridade: Em Bancos de Dados Relacionais utilize Transações
    - Considere o uso dos níveis de isolamento entre Transações para 'trade-off' entre performance e confiabilidade
    - Mais detalhes em Saiba Mais

Padrão Arquitetural

Model-View-Controller

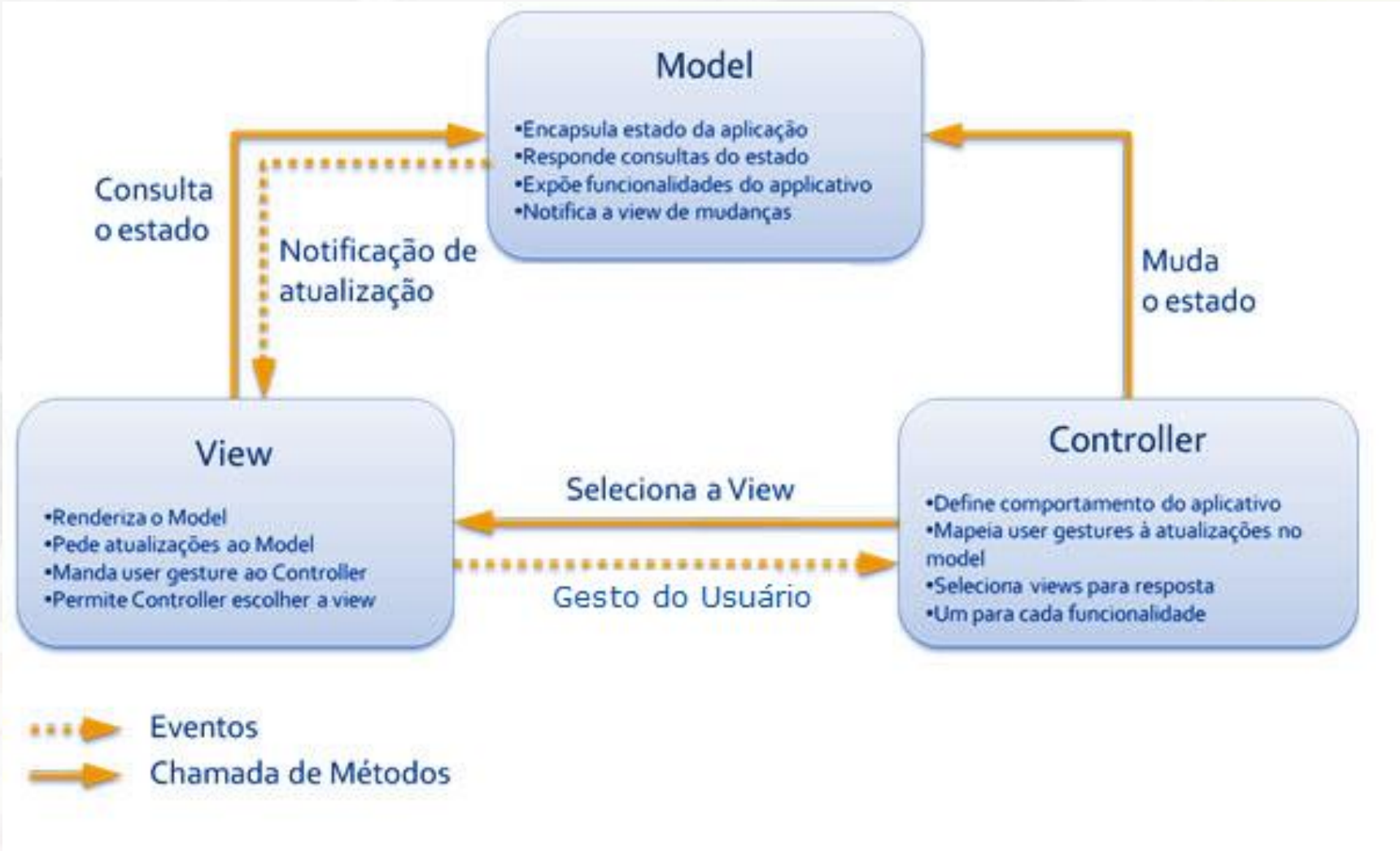
Padrão Model-View-Controller

- **Contexto**:
  - . A interface com o usuário é tipicamente a parte mais frequentemente modificada de um aplicativo interativo
  - Os usuários geralmente desejam olhar para dados de diferentes perspectivas, como um gráfico de barras ou um gráfico de tortas.
  - Essas representações devem refletir tanto o estado atual dos dados.
- **Problema**:
  - Como a funcionalidade da interface do usuário pode ser mantida separada da funcionalidade do aplicativo e ainda assim ser responsiva à entrada do usuário ou a alterações nos dados do aplicativo subjacente?
  - E como várias visualizações da interface do usuário podem ser criadas, mantidas e coordenadas quando os dados do aplicativo subjacente mudam?

Padrão Model-View-Controller

- **Solução**: O padrão MVC (Model-View-Controller, controlador de visualização de modelo) separa a funcionalidade do aplicativo em três tipos de componentes:
  - Um modelo, que contém os dados do aplicativo
  - Uma exibição, que exibe alguma parte dos dados subjacentes e interage com o usuário
  - Um controlador, que media entre o modelo e a visão e gerencia as notificações de mudanças de estado

MVC Example


Model
•Encapsula estado da aplicação
•Responde consultas do estado
•Expõe funcionalidades do applicativo
•Notifica a view de mudanças
Consulta o estado
Notificação de atualização
Muda o estado
View
•Renderiza o Model
•Pede atualizações ao Model
•Manda user gesture ao Controller
•Permite Controller escolher a view
Seleciona a View
Gesto do Usuário
Controller
•Define comportamento do aplicativo
•Mapeia user gestures à atualizações no model
•Seleciona views para resposta
•Um para cada funcionalidade
Eventos
Chamada de Métodos

MVC - Solução - 1

- **Visão geral**: O padrão MVC divide a funcionalidade do sistema em três componentes: um modelo, uma visualização e um controlador que faz a mediação entre o modelo e a visualização.
- **Elementos**:
  - O modelo é uma representação dos dados ou estado do aplicativo, e contém (ou fornece uma interface para) lógica de aplicação.
  - A visualização é um componente de interface do usuário que ou produz uma representação do modelo para o usuário ou permite alguma forma de entrada do usuário, ou ambos.
  - O controlador gerencia a interação entre o modelo e a visão, traduzindo as ações do usuário em alterações no modelo ou alterações na visão.

MVC - Solução - 2

- **Relações**: A relação *notifica* conecta instâncias de modelo, visualização e controle, notificando elementos a respeito de mudanças de estado relevantes.
- **Restrições**:
  - Deve haver pelo menos uma instância cada um de modelo, visualização e controlador.
  - O componente modelo não deve interagir diretamente com o controlador.
- **Fraquezas**:
  - A complexidade pode não valer a pena para interfaces simples de usuário.
  - As abstrações de modelo, visualização e controlador podem não ser boas para alguns toolkits de interface de usuário.

Padrão MVC – Exemplos

- Viabilizado por frameworks de desenvolvimento
  - Web
    - Spring MVC
    - Stripes Framework
    - Asp .NET MVC
  - Desktop
    - Java Swing
    - Eclipse RCP Java
    - Griffon Java
    - senCille para Delphi/Embarcadero

Padrão Arquitetural

Service Oriented Architecture - SOA

Service Oriented Architecture Pattern

Arquitetura Orientada a Serviços

- **Contexto**:
  - Uma série de serviços são oferecidos (e descritos) pelos prestadores de serviços e consumidos pelos consumidores de serviços.
  - Os consumidores de serviços precisam ser capazes de entender e usar esses serviços sem qualquer conhecimento detalhado de sua implementação.
- **Problema**:
  - Como podemos apoiar a interoperabilidade de componentes distribuídos executados em diferentes plataformas e escritos em diferentes linguagens de implementação,

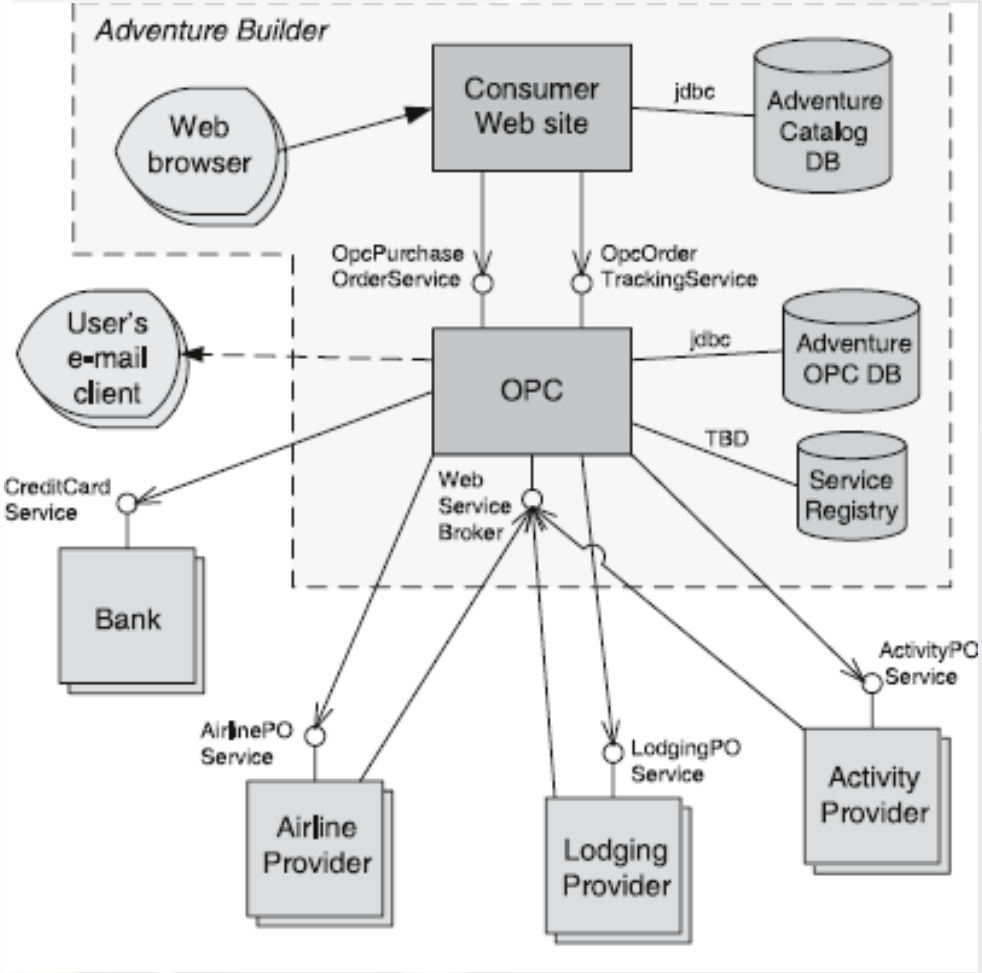
    fornecidos por diferentes organizações e

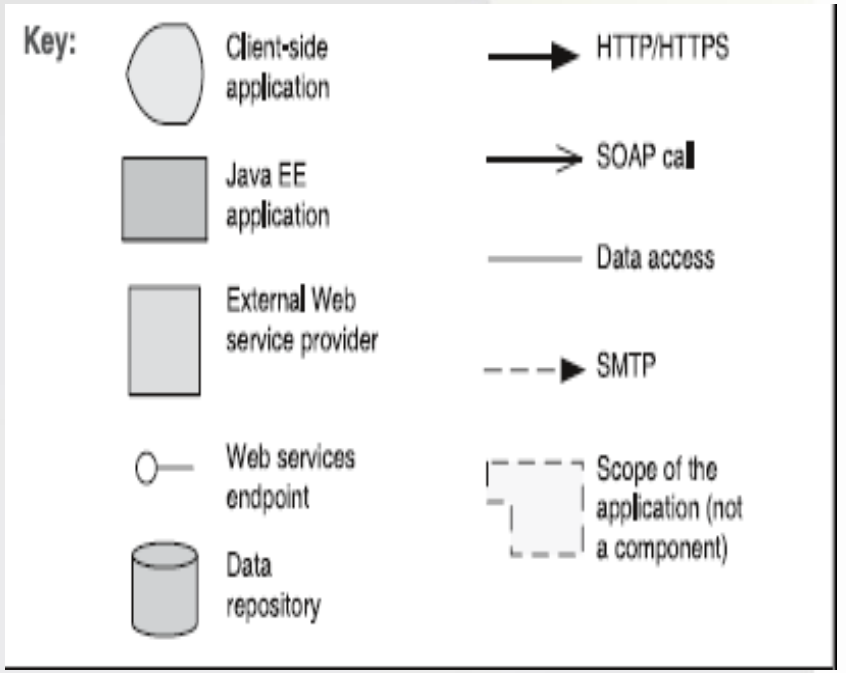
    distribuídos pela Internet?

Service Oriented Architecture Pattern

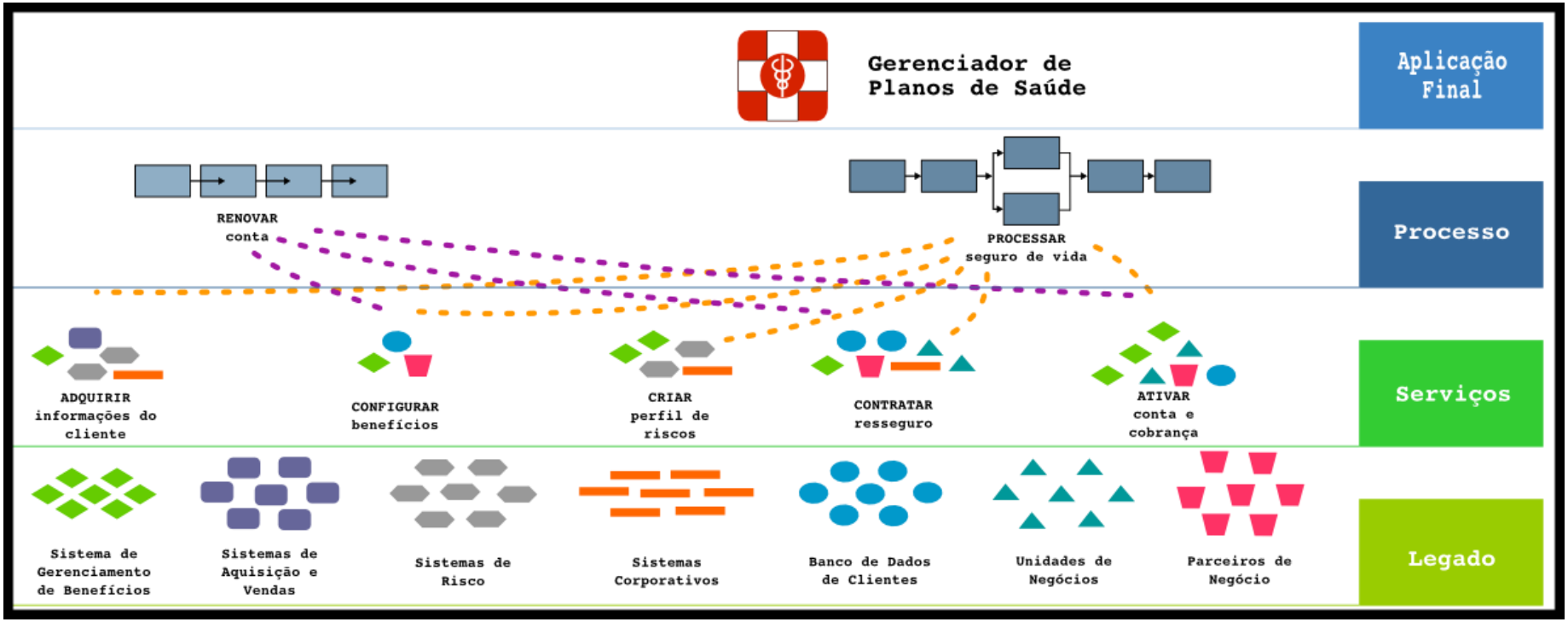
- **Solução**: O padrão SOA (Service-Oriented Architecture, arquitetura orientada a serviços) descreve uma coleção de componentes distribuídos que fornecem e/ou consomem serviços.

Exemplo da Arq. Orientada a Serviços





Exemplo da Arq. Orientada a Serviços



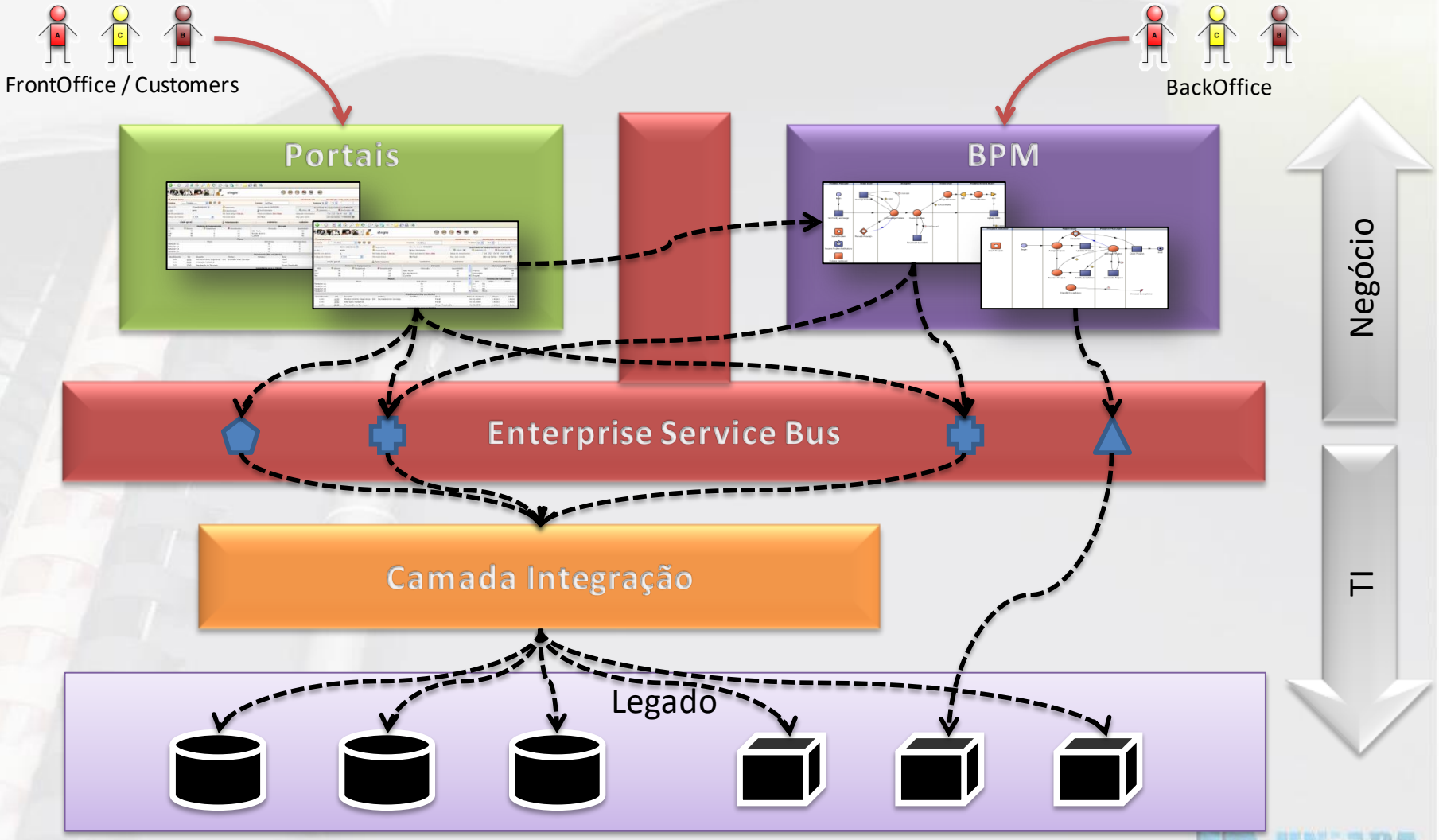
Padrão SOA - Solução - 1

- **Visão geral**: A computação é alcançada por um conjunto de componentes que fornecem e/ou consomem serviços em uma rede.
- **Elementos**:
  - Componentes:
    - *Service providers,* ou provedores de serviços que fornecem um ou mais serviços através de interfaces publicadas.
    - *Service consumers,* ou consumidores de serviços que invocam serviços diretamente ou através de um intermediário.
    - *Service providers* podem ser *service consumers*.

Padrão SOA - Solução - 2

- Componentes:
  - *Enterprise Service Bus (ESB),* ou barramento de serviços corporativo que é um elemento intermediário que pode encaminhar e transformar mensagens entre prestadores de serviços e consumidores.
  - *Registry of services,* ou registro de serviços que podem ser usados pelos provedores para registrar seus serviços e pelos consumidores para descobrir serviços em tempo de execução.
  - *Orchestration Server,* ou Servidor de Orquestração que coordena as interações entre consumidores de serviços e provedores com base em linguagens para processos de negócios e fluxos de trabalho.

Padrão SOA - Solução - 2



Orquestração

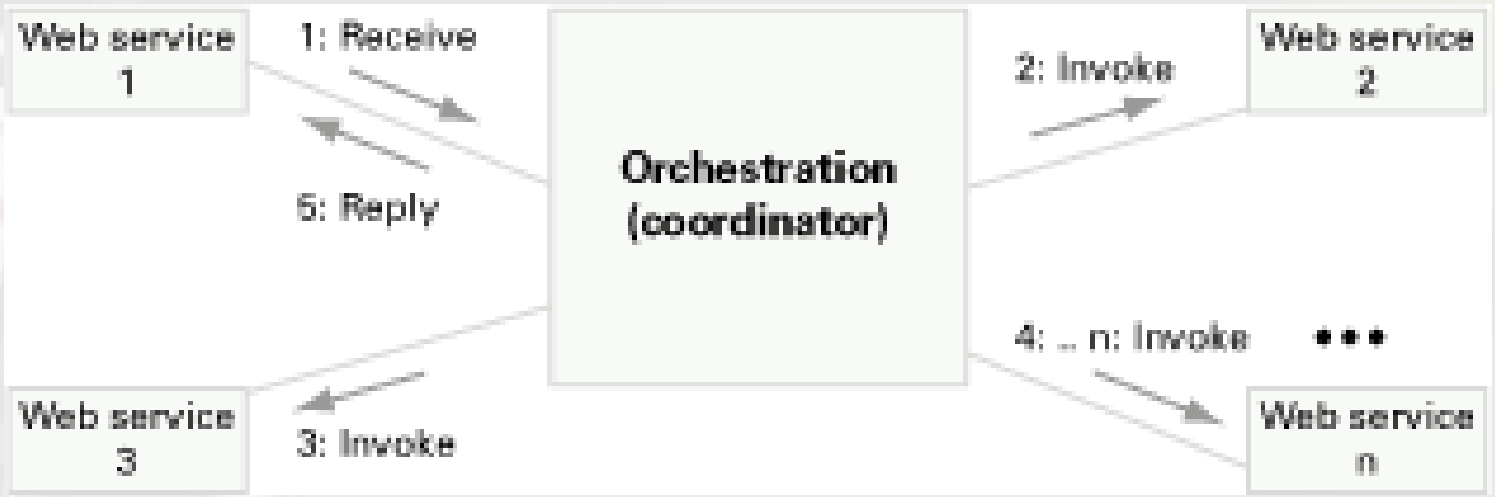
- Normalmente usado em processos de negócios privados
- Um processo central (que pode ser outro serviço Web) assume o controle dos Web Services envolvidos
- Coordena a execução de diferentes operações nos Web Services envolvidas na operação.

Orquestração

- Os Web Services envolvidos não "sabem" (e não precisa saber) que eles estão envolvidos em um processo de composição e que estão tomando parte em um processo de negócios de nível superior.

- Apenas o coordenador central da orquestração está ciente deste objetivo, de modo que a orquestração é centralizada com definições explícitas de operações e a ordem de chamada de Web Services

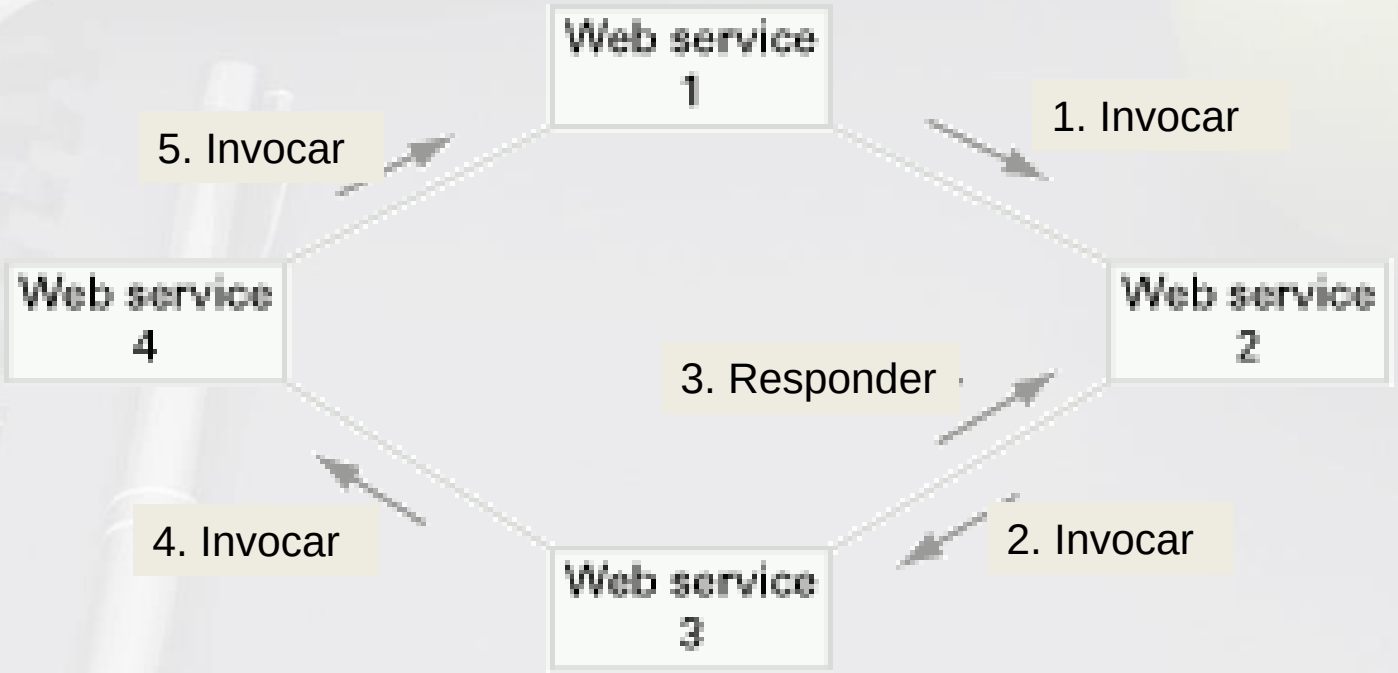
Orquestração



Coreografia

- Não depende de um coordenador central
- Cada serviço Web envolvidos na coreografia sabe exatamente quando deve executar suas operações e com quem interagem
- Esforço de colaboração com foco na troca de mensagens em processos de negócios públicos
- Todos os participantes da coreografia precisam estar cientes do processo de negócio, operações a executar, mensagens de troca, e o momento de troca de mensagens.

Coreografia



Orquestração vs Coreografia

- A partir da perspectiva de compor Web Services para executar processos de negócios, a orquestração é um paradigma mais flexível e tem as seguintes vantagens sobre coreografia:
  - A coordenação dos processos componentes é gerenciado centralmente por um coordenador conhecido.
  - Web services podem ser incorporados sem estar ciente de que eles estão participando de um processo de negócio maior.
  - Cenários alternativos podem ser postas em prática no caso de ocorrerem falhas.

Padrão SOA - Solução - 3

- Conectores:
  - *SOAP connector,* ou conector SOAP: usa o protocolo SOAP para comunicação síncrona entre serviços web, normalmente sobre HTTP.
    - Simple Object Access Protocol aplica a comunicação baseada em XML, Esquemas XML e descoberta de serviços
  - *REST connector,* ou conector REST: Se baseia nas operações básicas de solicitação/resposta do protocolo HTTP.
    - Representational State Transfer: Utiliza HTTP, seus verbos e caminhos e comunicação baseada em JSON (Javascript Object Notation)
  - *Asynchronous messaging connector, ou conector de mensageria assíncrona:* usa middlewares orientados a mensagens para oferecer trocas de

    mensagens assíncronas ponto-a-ponto

    ou publish-subscribe (publicar-assinar).

Padrão SOA - Solução - 4

- **Relações**: Conexão dos diferentes tipos de componentes disponíveis aos respectivos conectores
- **Restrições**: Os consumidores de serviços estão conectados aos provedores de serviços, mas componentes intermediários (por exemplo, ESB, registro, servidor de orquestração) podem ser usados.
- **Fraquezas**:
  - Sistemas baseados em SOA são tipicamente complexos de serem construídos.
  - Você não controla a evolução dos serviços independentes.
  - Há uma sobrecarga de desempenho associada ao middleware, e os serviços podem ser gargalos de desempenho, e normalmente não fornecem garantias de desempenho.

Padrão SOA – Exemplos

Exemplos de Produtos Baseados em SOA

- JBoss Enterprise SOA Platform
- WebMethods Integration Server
- Fuse ESB
- Oracle Fusion Applications, Architecture e Middleware
- Oracle SOA Suite
- Oracle WebLogic Server
- Oracle Beehive
- Petals ESB
- SAP Composite Application Framework
- SAP NetWeaver

Padrão Arquitetural

Pipe and Filter

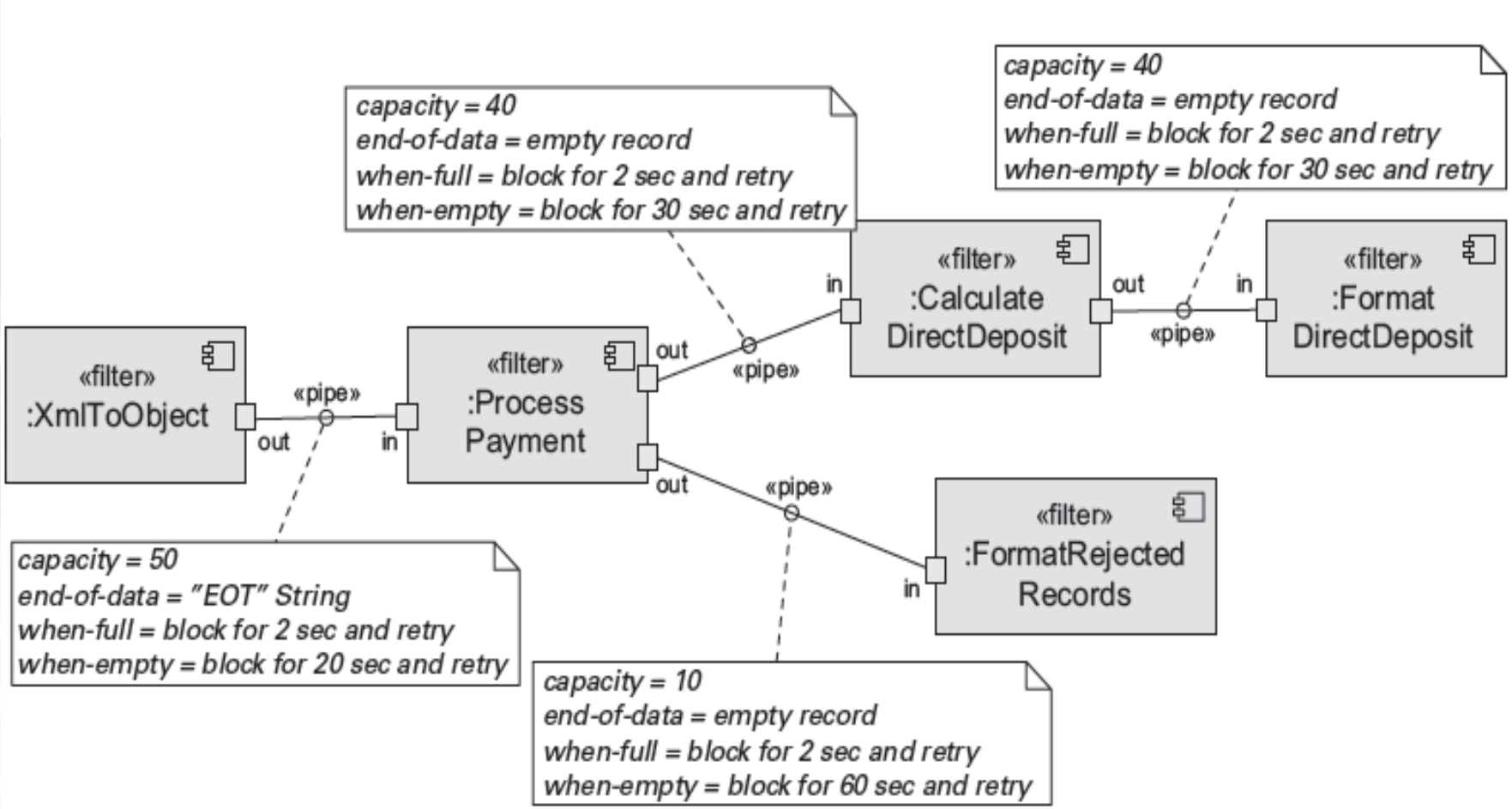
Padrão Pipe and Filter

- Padrão “Tubos” E “Filtros”
- **Contexto:**
  - Pode ser necessário transformar fluxos de dados discretos, da entrada à saída.
  - Vários tipos de transformações podem ocorrer repetidamente
  - É desejável criá-las como partes independentes e reutilizáveis.
- **Problema:**
  - Tais sistemas precisam ser divididos em componentes reutilizáveis e vagamente acoplados com mecanismos de interação simples e genéricos.
  - Dessa forma podem facilmente reutilizados.
  - Os componentes, sendo independentes, podem ser executados em paralelo.

Padrão Pipe and Filter

- **Solução:**
  - O padrão de interação é caracterizado por sucessivas transformações de fluxos de dados.
  - Os dados chegam às portas de entrada de um filtro, são transformados e, em seguida, são passados através de sua porta de saída através de um pipe para o próximo filtro.
  - Um único filtro pode consumir dados ou produzir dados para uma ou mais portas.

Exemplo Pipe and Filter



Solução - Pipe and Filter - 1

- **Visão geral**: Os dados são transformados das entradas externas de um sistema para suas saídas externas através de uma série de transformações realizadas por seus filtros conectados por conectores.
- **Elementos**:
  - *Filter,* que é um componente que transforma os dados lidos em suas portas de entrada para dados escritos em sua porta de saída(s).
  - *Pipe,* que é um conector que transmite dados das portas de saída de um filtro para as portas de entrada de outro filtro. Um tubo tem uma única fonte para sua entrada e um único alvo para sua saída. Um tubo preserva a sequência de itens de dados, e não altera os dados que passam.

Solução - Pipe and Filter - 2

- **Relações**: A relação de fixação associa a saída de filtros com a entrada de tubos e vice-versa.
- **Restrições**:
  - Os conectores conectam as portas de saída de um filtro às portas de entrada de outro filtro.
  - Os filtros conectados devem concordar com o tipo de dados que estão sendo passados ao longo do conector.

Padrão Pipe and Filter – Exemplos

- Spring Cloud Data Flow - Spring Cloud Data Flow fornece ferramentas para criar topologias complexas para pipelines de dados de streaming e lote
- Apache Flink [Java] - sistema para processamento de fluxo de dados de alta produtividade e baixa latência que suporta computação estatal, semântica de janelas orientada por dados e processamento de fluxo iterativo.
- Apache Heron [Java] - um mecanismo de processamento de fluxo em tempo real, distribuído e tolerante a falhas do Twitter.
- Faust [Python] - biblioteca de processamento de fluxo, portando as ideias de Kafka Streams para Python
- Hazelcast Jet [Java] - Um mecanismo de processamento de dados distribuído de propósito geral, construído em cima da Hazelcast.

Padrão Pipe and Filter – Exemplos

- SABER [Java/C] - Motor de processamento de fluxo híbrido baseado em GPU para Windows
- Trill [.NET/C#] - Trill é um mecanismo de análise de streaming de alta performance da Microsoft Research.
- Wallaroo [Python] - Uma estrutura rápida de processamento de fluxo. Wallaroo facilita a reação aos dados em tempo real.
- LightSaber [C++] - Motor de processamento de fluxo baseado em janelas multi-núcleo. O LightSaber usa geração de código para agregação eficiente de janelas.
- Kuiper [Golang] - Um software de análise/streaming de dados IoT leve de borda implementado pela Golang, e pode ser executado

  em todos os tipos de dispositivos de borda restritos a

  recursos.

Padrão Arquitetural

Broker

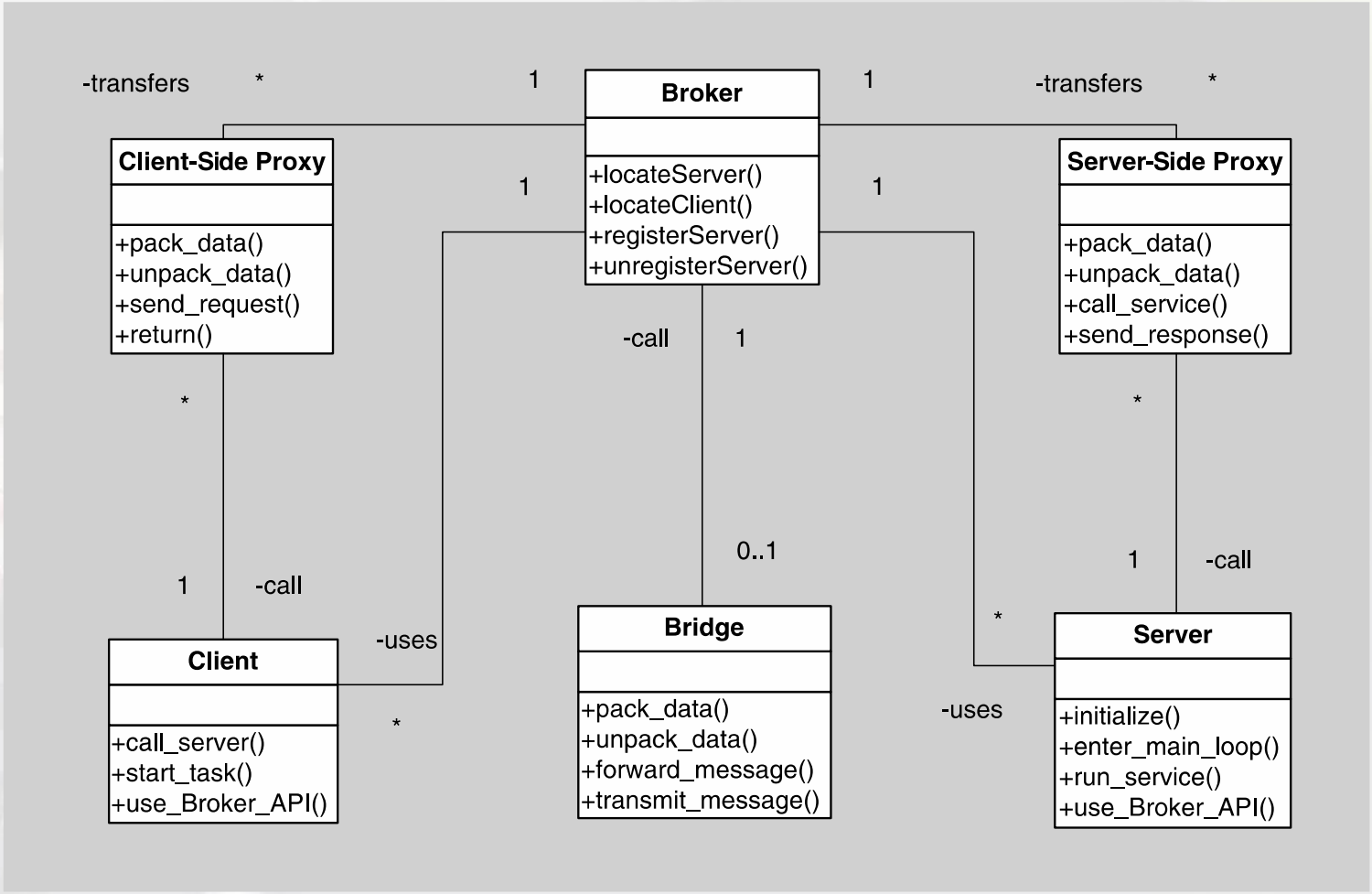
Padrão Broker

- **Contexto**:
  - Sistemas construídos a partir de uma coleção de serviços distribuídos em vários servidores.
  - A implementação desses sistemas é complexa
    - Você precisa se preocupar com como os sistemas interoperam
    - Como eles se conectarão uns com os outros
    - Como eles trocarão informações
    - Como será a disponibilidade dos serviços componentes.
- **Problema**:
  - Como estruturar o software distribuído para que:
    - Os usuários de serviços não precisem saber a natureza e a localização dos provedores de serviços?
    - Como facilitar a mudança dinâmica das vinculações entre usuários e provedores?

Padrão Broker

- **Solução**: O padrão Broker (ou ‘corretor’)
  - Separa os usuários de serviços (clientes) dos provedores de serviços (servidores)
  - Insere um intermediário, chamado de Broker.
  - Quando um cliente precisa de um serviço, ele consulta o Broker ‘corretor’ através de uma interface de serviço.
  - Em seguida, o Broker encaminha a solicitação de serviço do cliente para um servidor, que processa a solicitação.

Exemplo do Padrão Broker



Broker – Solução - 1

- Visão geral: O padrão do corretor define um componente de tempo de execução, chamado de corretor, que media a comunicação entre vários clientes e servidores.
- Elementos:
  - *Cliente,* um solicitante de serviços
  - *Server,* um provedor de serviços
  - *Broker,* um intermediário que localiza um servidor apropriado para atender a solicitação de um cliente, encaminha a solicitação ao servidor e devolve os resultados ao cliente
  - *Client-side proxy,* um intermediário que gerencia a comunicação real com o corretor, incluindo conversão, envio e desconversão de mensagens
  - *Server-side proxy,* um intermediário que gerencia a comunicação real com o corretor, incluindo conversão, envio e desconversão de mensagens

Broker – Solução - 2

- **Relações**: A relação de conexão associa clientes (e, opcionalmente, proxies do lado do cliente) e servidores (e, opcionalmente, proxies do lado do servidor) com o Broker.
- **Restrições**: O cliente só pode se conectar a um broker (potencialmente através de um proxy do lado do cliente). O servidor só pode ser conectado a um corretor (potencialmente através de um proxy do lado do servidor).
- **Fraquezas**:
  - Os Brokers adicionam uma camada de indireção (ou com intermediários) e, portanto, latência, entre clientes e servidores, e essa camada pode ser um gargalo de comunicação.
  - O Broker pode ser um único ponto de falha.
  - Um Broker adiciona complexidade inicial grande.
  - Um Broker pode ser alvo de ataques de segurança.
  - Um Broker pode ser difícil de testar.

Padrão Broker – Exemplos

- Invocações Remotas de Métodos em Objetos Distribuídos
  - Java RMI
  - Corba RMI IOOP
  - Microsoft DCOM
  - Java Enterprise Java Beans (EJB)

Padrão Arquitetural

Peer-to-Peer

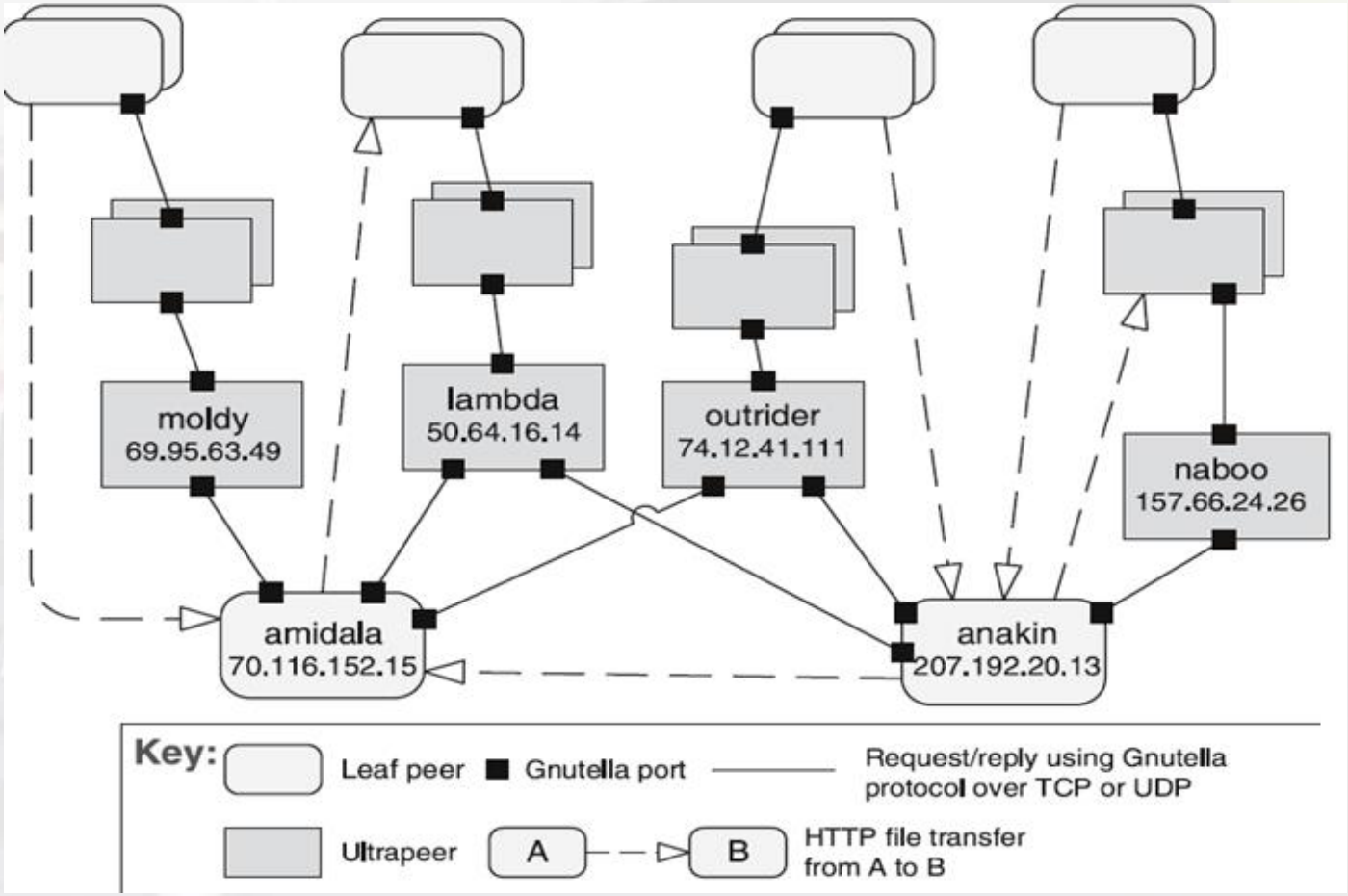
Padrão Peer-to-Peer

- **Contexto:** Entidades computacionais distribuídas — cada uma das quais é considerada igualmente importante em termos de iniciar uma interação e cada uma delas fornece seus próprios recursos — precisam cooperar e colaborar para fornecer um serviço a uma comunidade distribuída de usuários.
- **Problema:** Como um conjunto de entidades computacionais distribuídas “igualmente" pode ser conectado entre si através de um protocolo comum para que possam organizar e compartilhar seus serviços com alta disponibilidade e escalabilidade?

Padrão Peer-to-Peer

- **Solução:** No padrão peer-to-peer (P2P), os componentes interagem diretamente como pares. Todos os pares são "iguais" e nenhum par ou grupo de pares pode ser crítico para a saúde do sistema. A comunicação peer-to-peer é tipicamente uma interação de solicitação/resposta sem a assimetria encontrada no padrão cliente-servidor.

Exemplo Peer-to-Peer



Solução - Peer-to-Peer - 1

- **Visão geral**: A computação é alcançada por pares cooperando que solicitam serviços e fornecem serviços uns aos outros em uma rede.
- **Elementos**:
  - *Peer,* que é um componente independente em execução em um nó de rede. Componentes especiais de pares podem fornecer roteamento, indexação e capacidade de pesquisa por pares.
  - *Conector de requisição/resposta,* que é usado para se conectar à rede de pares, procurar outros pares e invocar serviços de outros pares.
- **Relações**: A relação associa pares com seus conectores. Anexos podem mudar em tempo de execução.

Solução - Peer-to-Peer - 2

- **Restrições**: As restrições são as seguintes:
  - O número máximo de conexões permitidos a qualquer par?
  - O número de redes que separam os pares?
  - Quais pares sabem sobre quais outros pares?
  - Algumas redes P2P são organizadas com topologias estrela, nas quais os pares só se conectam a super-nós.
- **Fraquezas**:
  - Gerenciar segurança, consistência de dados, disponibilidade de dados/serviços, backup e recuperação são todos mais complexos.
  - Pequenos sistemas peer-to-peer podem não ser capazes de alcançar consistentemente metas de qualidade, como desempenho e disponibilidade.

Padrão Arquitetural

Publish-Subscribe

Padrão Publish-Subscribe

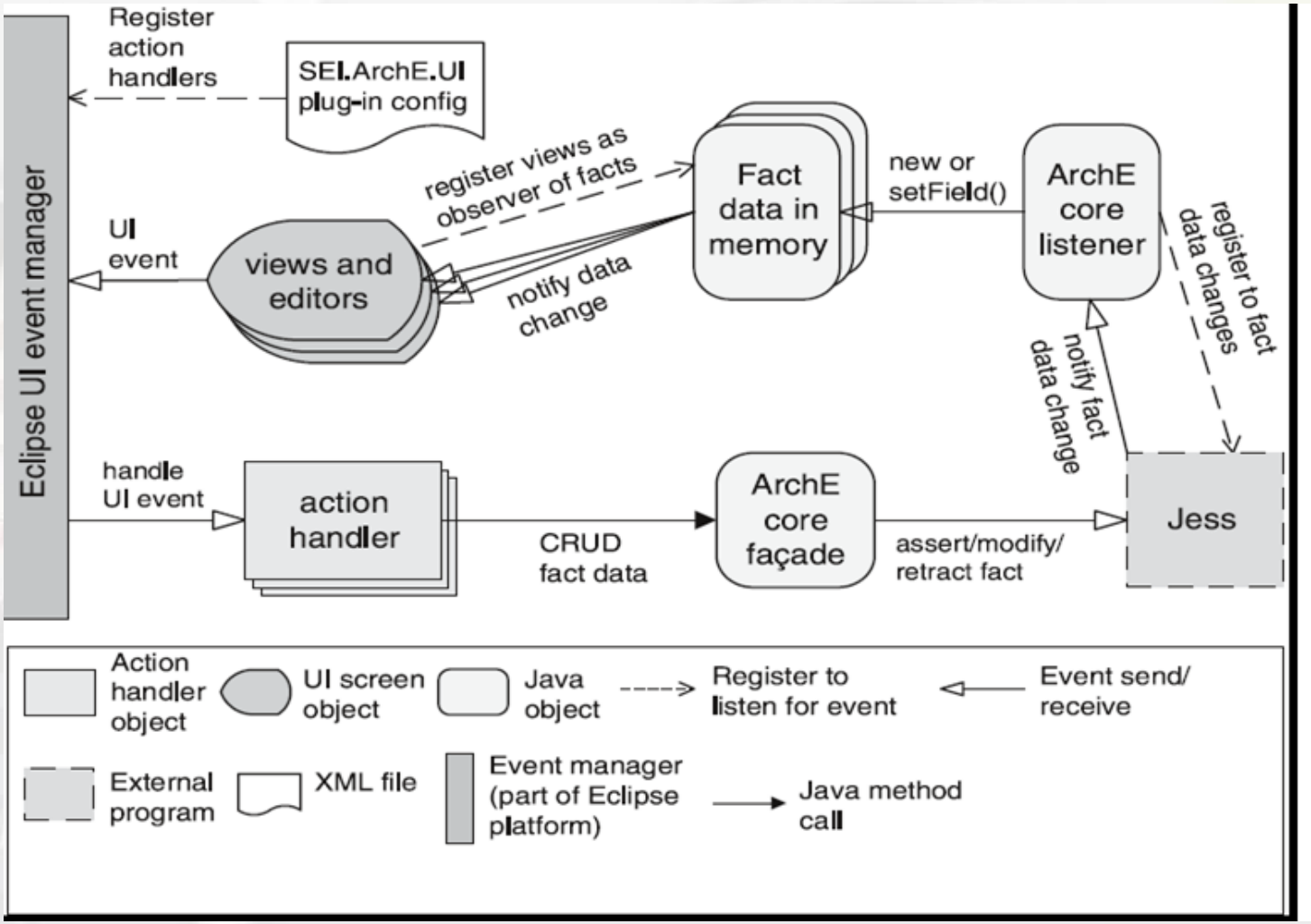
- Ou em português: Publicar-Assinar
- **Contexto:** Há uma série de produtores independentes e consumidores de dados que devem interagir. O número preciso e a natureza dos produtores de dados e consumidores não são predeterminados ou fixos, nem os dados que compartilham.

- **Problema:** Como podemos criar mecanismos de integração que apoiem a capacidade de transmitir mensagens entre os produtores e consumidores para que eles desconheçam a identidade uns dos outros, ou potencialmente até mesmo sua existência?

Padrão Publish-Subscribe

- **Solução:**
  - No padrão publish-subscribe, os componentes interagem através de mensagens ou eventos enviados.
  - Os componentes podem assinar um conjunto de eventos.
  - Os componentes de publicação colocam eventos no barramento anunciando-os;
  - O conector, em seguida, fornece esses eventos para os componentes assinantes que registraram interesse nesses eventos.

Publish-Subscribe Exemplo



Publish-Subscribe - Solução – 1

- **Visão geral**: Os componentes publicam e assinam eventos. Quando um evento é anunciado por um componente, a infraestrutura do conector despacha o evento para todos os assinantes registrados.
- **Elementos**:
  - *Qualquer componente* com pelo menos uma porta de publicação ou subscrição.
  - *The publish-subscribe connector*, que terá a função de *anunciar* e *ouvir* para componentes que desejam publicar e assinar eventos.
- **Relações**: A relação de conexão associa componentes ao conector de publicação por meio de uma inscrição pela qual componentes anunciam eventos e quais componentes estão registrados para receber eventos.

Publish-Subscribe - Solução - 2

- **Restrições**:
  - Todos os componentes estão conectados a um distribuidor de eventos que pode ser visto como um conector de barramento ou um componente.
  - As portas de publicação são anexadas para anunciar e as portas de subscrição são anexadas para ouvir.
- **Fraquezas**:
  - Normalmente aumenta a latência e tem um efeito negativo sobre a escalabilidade e previsibilidade do tempo de entrega de mensagens.
  - Menor controle sobre o envio e a recepção
  - Atente-se para escolher um Middleware com garantia de entregas das mensagens.

Padrão Publish-Subscribe – Exemplos

- Middleware Orientado a Mensagens
  - Apache ActiveMQ
  - Apache Artemis
  - Apache Kafka
  - Apache Qpid
  - Apache Pulsar
  - Amazon Web Services (AWS) Amazon MQ
  - Amazon Web Services (AWS) Kinesis
  - Eclipse Mosquitto MQTT Broker (Eclipse Foundation)
  - Fuse Message Broker (enterprise ActiveMQ)
  - Google Cloud Pub/Sub (Google)

Padrão Arquitetural

Map-Reduce

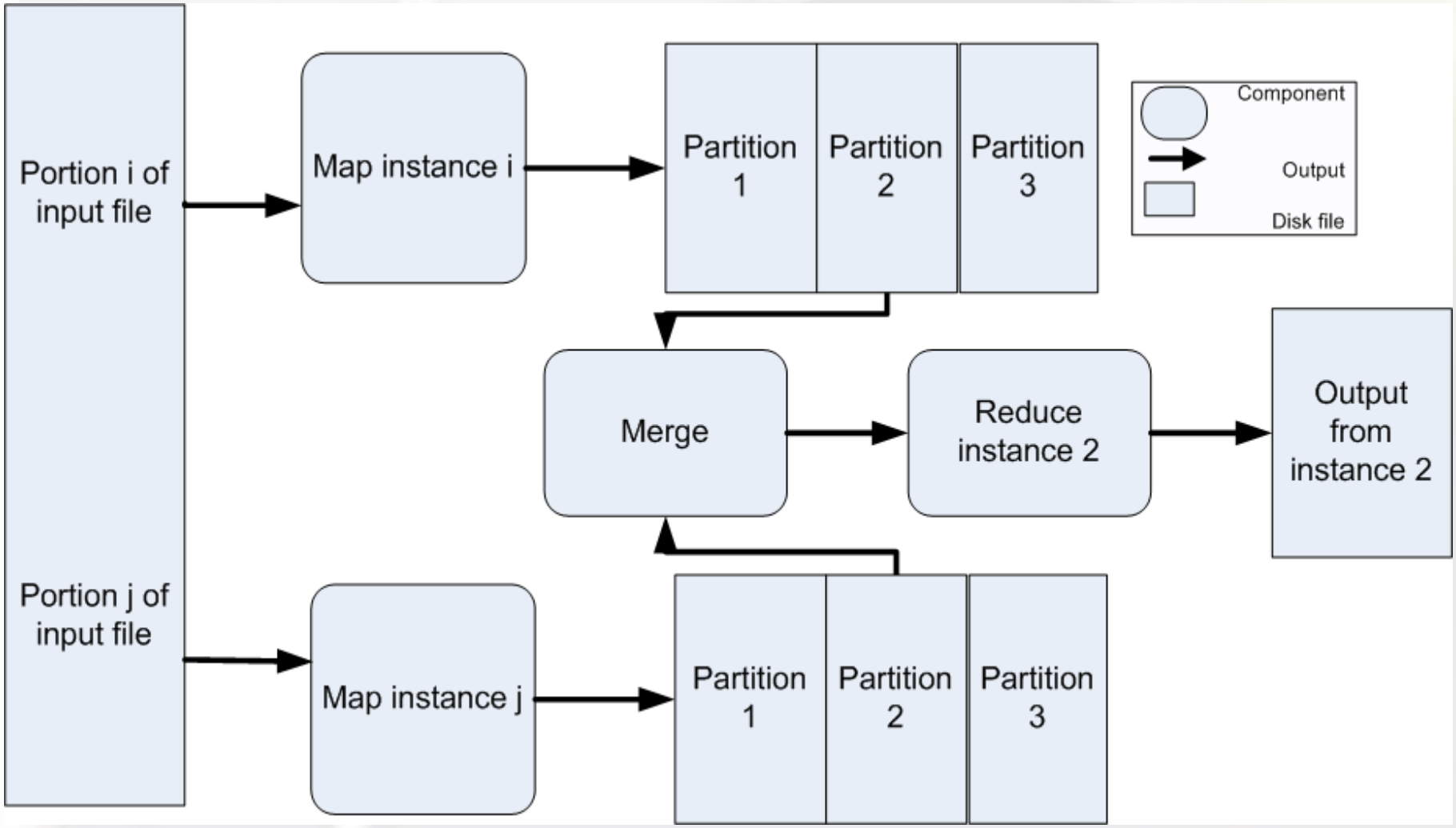
Padrão Map-Reduce

- Map-Reduce (ou Mapear e Reduzir) foi publicado em 2004 por Jeffery Dean e Sanjay Ghemawat no artigo "MAP-REDUCE: PROCESSO SIMPLIFICADO DE DADOS EM GRANDES CLUSTERS“.
  - Abordagem do Google para coletar e analisar dados de sites para pesquisa.
- **Contexto:** Organizações têm uma necessidade premente de analisar rapidamente enormes volumes de dados que geram ou acessam, em escala de petabytes.
- **Problema:** Para muitos aplicativos com conjuntos de dados ultra-grandes, classificar os dados e, em seguida, analisar os dados agrupados é suficiente. O problema que o padrão mapear e reduzir resolve é executar eficientemente uma espécie distribuída e paralela classificação e agrupamento de conjuntos de dados muito grandes e fornecendo um meio simples para o programador especificar a análise a ser feita.

Padrão Map-Reduce

- **Solução:** O padrão map-reduce tem três requisitos:
  - Uma infraestrutura especializada cuida da alocação de software para os nódulos de hardware em um ambiente de computação massivamente paralelo e lida com a classificação dos dados conforme necessário.
  - Um componente especificado pelo programador chamado mapa que filtra os dados para recuperar esses itens a serem combinados.
  - Um componente especificado pelo programador chamado reduce que combina os resultados do mapa

Exemplo do Map-Reduce



Map-Reduce - Solução - 1

- **Visão geral**: O padrão map-reduce fornece uma estrutura para analisar um grande conjunto distribuído de dados que será executado em paralelo, em um conjunto de processadores. Essa paralelização permite baixa latência e alta disponibilidade. O mapeamento realiza a extração e transforma partes da análise e a redução realiza o carregamento dos resultados.

Map-Reduce - Solução - 2

- **Elementos**:
  - Mapear é uma função com múltiplas instâncias implantadas em vários processadores que executa a parte da análise relacionada com a extração e transformação dos dados.
  - Reduzir é uma função que pode ser implantada como uma única instância ou como várias instâncias entre processadores para executar a porção de carga de dos dados transformados.
  - A infraestrutura é a estrutura responsável pela implantação das instâncias de mapeamento e redução, transferindo os dados entre elas e detectando e recuperando de falhas.

Map-Reduce - Solução - 3

- **Relações**:
  - *Implantar em* é a relação entre uma instância da função de mapeamento ou redução e o processador no qual ela será instalado.
  - *Instanciar, monitorar e controlar* é a relação entre a infraestrutura e as instâncias de mapeamento e de redução.
- **Restrições**:
  - Os dados a serem analisados devem existir como um conjunto de arquivos.
  - As funções de mapeamento são *stateless* (sem estado) e não se comunicam entre si.
  - A única comunicação entre as instâncias de mapeamento e redução são os dados enviados das instâncias de mapeamento para para as instâncias de redução pares <chave, valor>.

Map-Reduce - Solução - 4

- **Fraquezas**:
  - Se você não tiver grandes conjuntos de dados, a sobrecarga do map-reduce não é justificada.
  - Se você não puder dividir seus dados definidos em subconjuntos de tamanho semelhante, as vantagens do paralelismo são perdidas.
  - Operações que requerem múltiplas reduções são complexas de orquestrar.

Map-Reduce - Ferramentas

- Apache CouchDB
- Apache Hadoop
- Infinispan
- Riak
- Amazon EMR
- Azure Data Factory
- MapReduce for Google App Engine

Padrões e Táticas Arquiteturais

Relações entre táticas e padrões

- Padrões são construídos a partir de táticas; se um padrão é uma molécula, uma tática é um átomo.
- MVC, por exemplo, utiliza as táticas:
  - Aumente a coerência semântica
  - Encapsulamento
  - Use um intermediário
  - Use a vinculação do tempo de execução

Taticas Aumentam os Padrões

- Os padrões resolvem um problema específico, mas são neutros ou têm fraquezas em relação a outras qualidades.
- Considere o padrão do Broker
  - Pode ter gargalos de desempenho
  - Pode ter um único ponto de falha
- Usando táticas como
  - Aumentar recursos ajudará o desempenho
  - Manter várias cópias ajudará com a disponibilidade

Táticas e Interações

- Cada tática tem vantagens (sua razão de ser) e negativos – efeitos colaterais.
- O uso de táticas pode ajudar a aliviar os negativos.
- Mas nada é de graça...

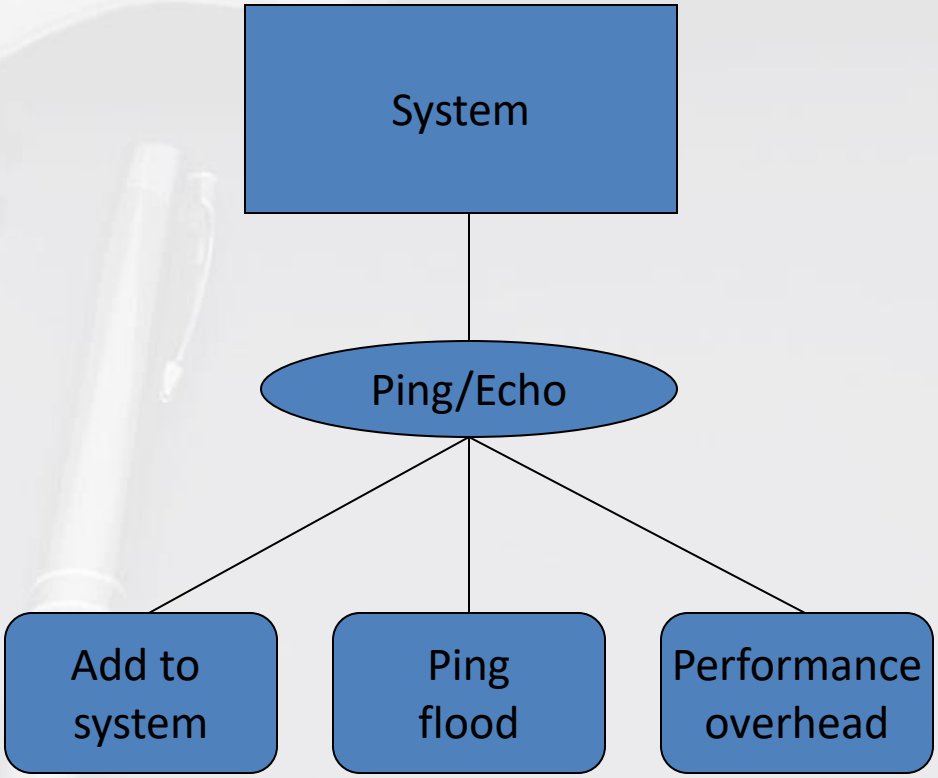
Táticas e Interações - 2

Uma tática comum para detectar falhas é ping/Echo.

Efeitos colaterais comuns do Ping/Echo são:

- Segurança: como evitar um ataque de inundação de ping?
- Desempenho: como garantir que a sobrecarga de desempenho do ping/echo seja pequena?
- Modificabilidade: como adicionar ping/eco à arquitetura existente?

Táticas e Interações - 3



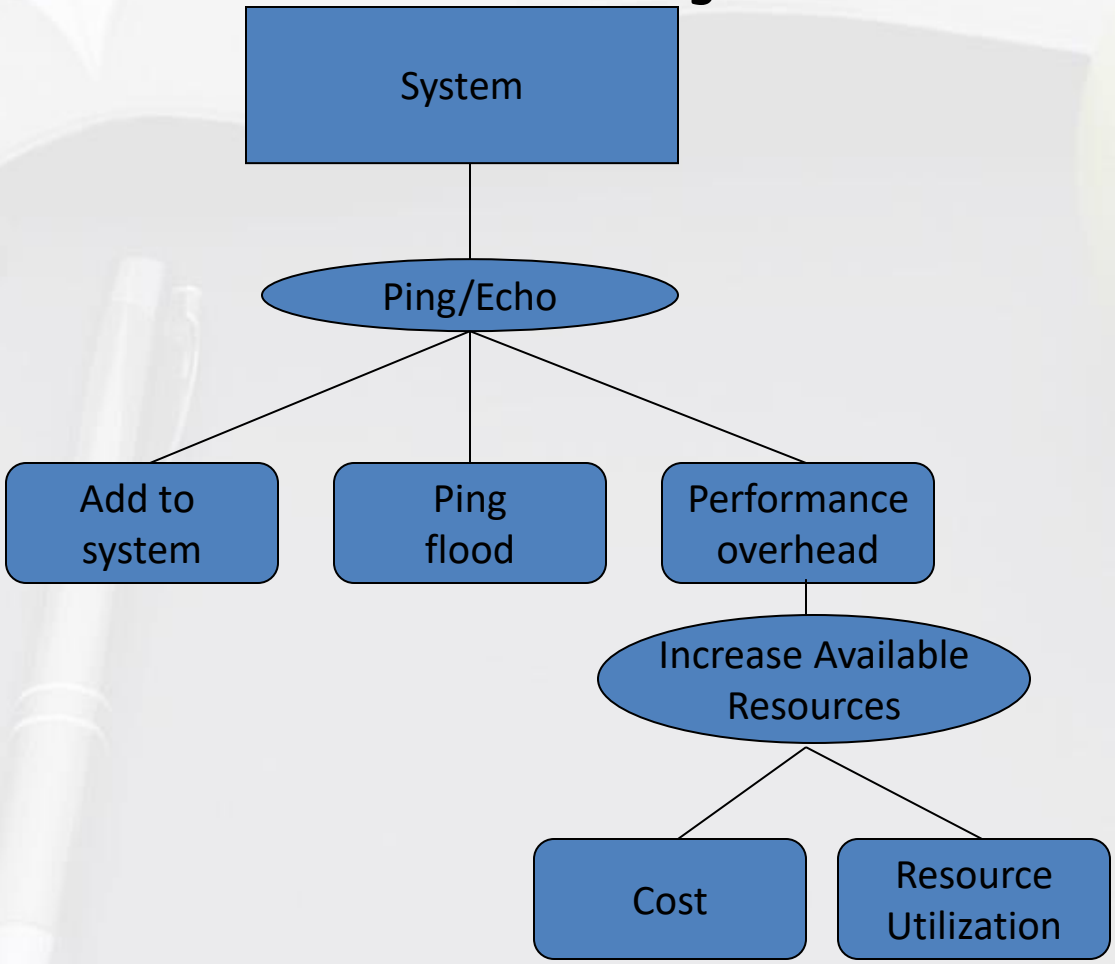
Táticas e Interações - 4

Uma tática para lidar com o efeito colateral de desempenho é "Aumentar os recursos disponíveis".

Os efeitos colaterais comuns do Aumento de Recursos Disponíveis são:

- Custo: aumento de recursos custam mais
- Desempenho: como utilizar os recursos de aumento de forma eficiente?

Táticas e Interações - 5



Táticas e Interações - 6

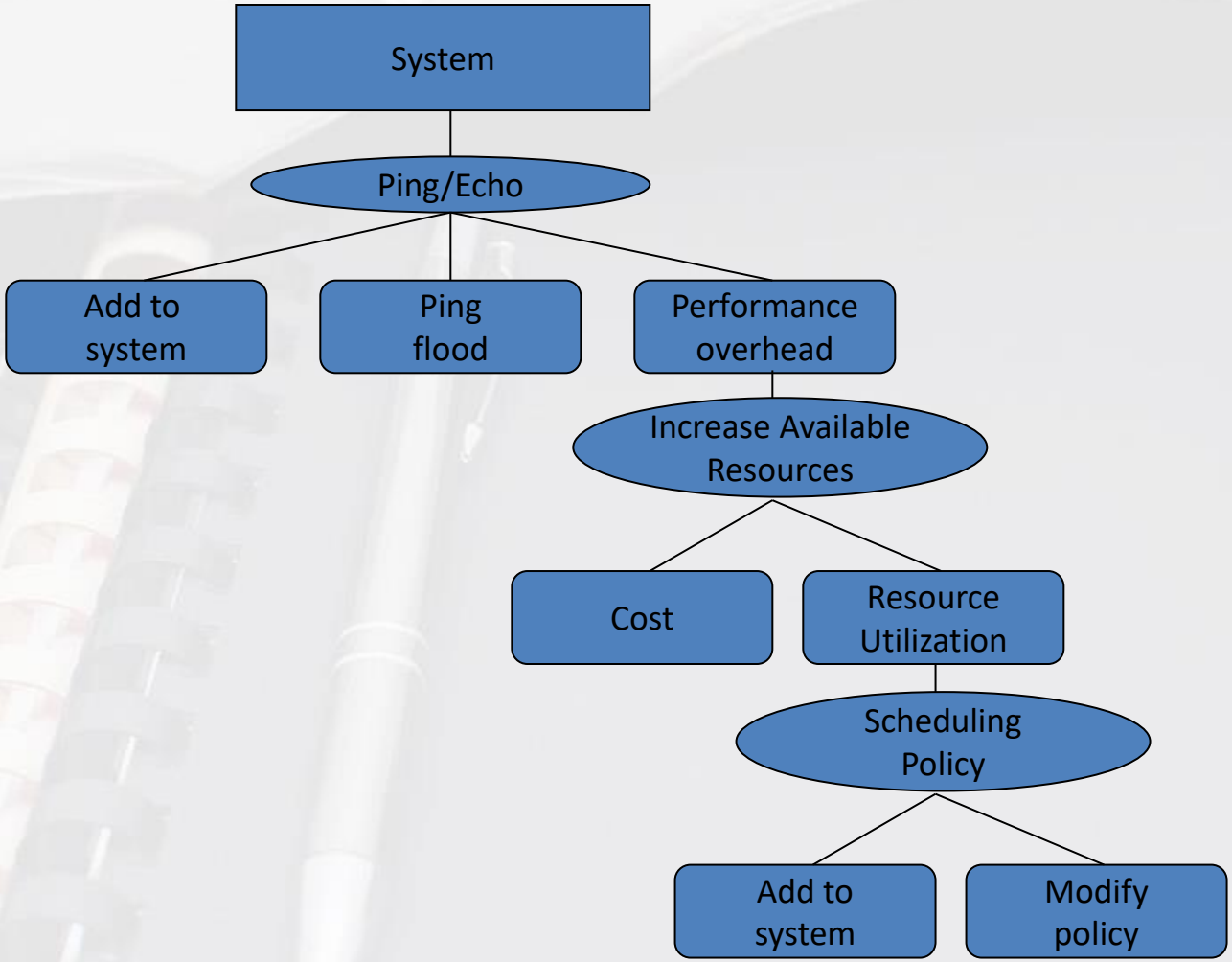
Uma tática para lidar com o uso eficiente do efeito colateral dos recursos é a "Política de Escalonamento".

Os efeitos colaterais comuns da Política de Agendamento são:

- Modificabilidade: como adicionar a política de escalonamento à arquitetura existente
- Modificabilidade: como alterar a política

  de escalonamento no futuro?

Táticas e Interações - 7



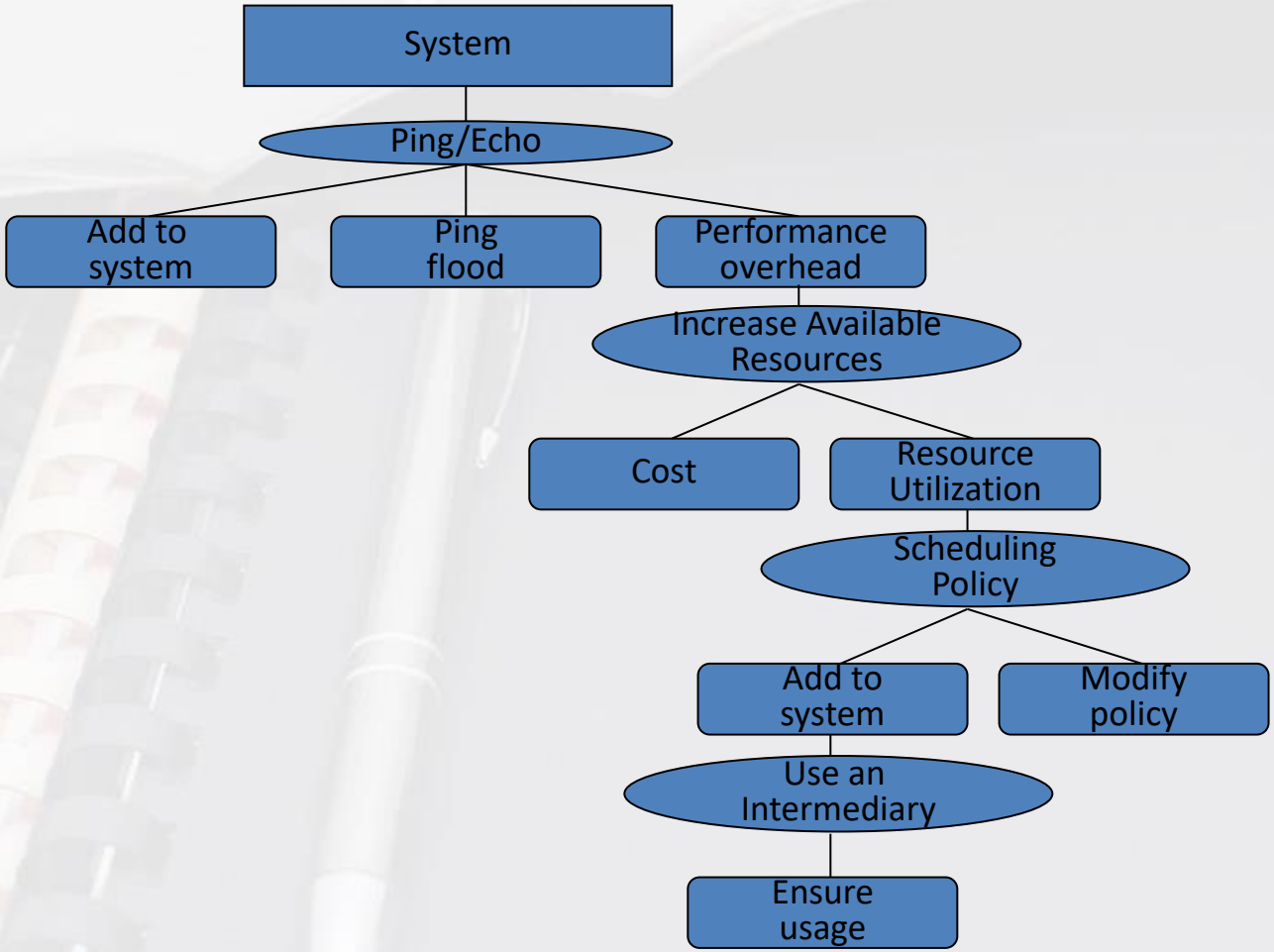
Táticas e Interações - 8

Uma tática para abordar a adição do agendador ao sistema é “Use um Intermediário”.

Os efeitos colaterais comuns do “Uso de um Intermediário” são:

- Modificabilidade: como garantir que toda comunicação passe pelo intermediário?

Táticas e Interações - 9



Táticas e Interações – 10.

Uma tática para abordar a preocupação de que toda comunicação passa pelo intermediário é "Restringir caminhos de comunicação".

Os efeitos colaterais comuns dos “Caminhos de Comunicação Restritos” são:

- desempenho: como garantir que a sobrecarga de desempenho do intermediário não seja excessiva?

Nota: este problema de design tornou-se agora recursivo!

Como esse processo termina?

- Cada uso da tática introduz novas preocupações.
- Cada nova preocupação faz com que novas táticas sejam adicionadas.
- Estamos em uma progressão infinita?
- Não. Eventualmente, os efeitos colaterais de cada tática tornam-se pequenos o suficiente para ignorar.

Resumo

- Um padrão arquitetural
  - é um pacote de decisões de projeto que é encontrado repetidamente na prática,
  - tem propriedades conhecidas que permitem reutilização, e
  - descreve uma classe de arquiteturas.
- Táticas são mais simples do que padrões
- Os padrões são subespecificados em relação aos sistemas reais, então eles têm que ser aumentados com táticas.
  - O aumento termina quando os requisitos para um sistema específico são satisfeitos.

Referências Bibliográficas

- BASS, L. et al. **Software Architecture in Practice**. Addison Wesley, Upper Saddle River, 2013, 3ª ed.

- GALLOTTI, G.M.A. **Arquitetura de Software**. Editora Pearson, São Paulo, 2016.

- SOMMERVILLE, Ian. **Engenharia de Software**. 10. Ed. – São Paulo: Pearson Education do Brasil, 2018.