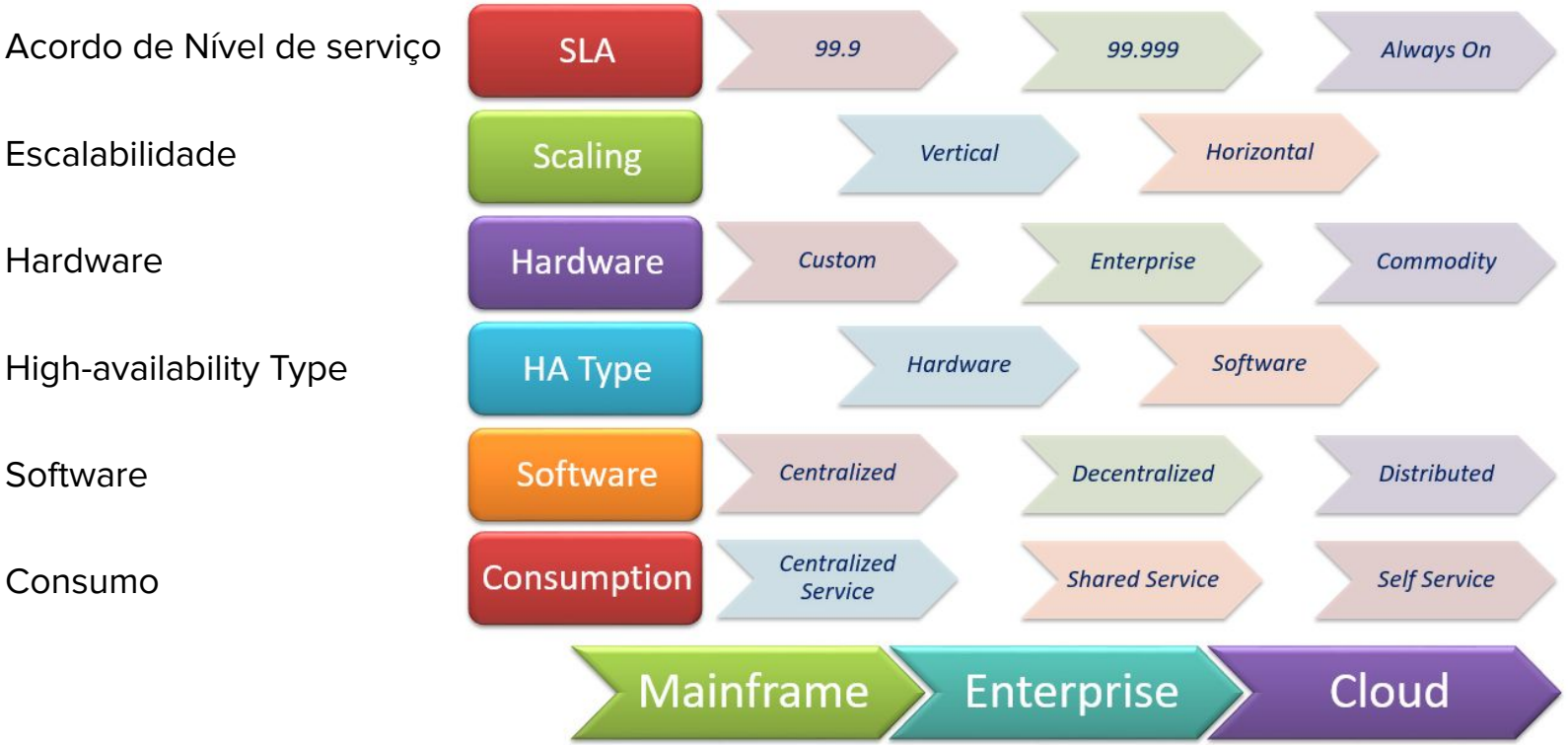
Evolução dos modelos computacionais



Desempenho

Dois dos principais objetivos do desenho de aplicações paralelas são:

- **Desempenho**: a capacidade de reduzir o tempo de resolução do problema à medida que os recursos computacionais aumentam.
- **Escalabilidade**: a capacidade de aumentar o desempenho à medida que a complexidade do problema aumenta

Escalabilidade

Escalabilidade: capacidade de manipular uma porção crescente de trabalho de forma uniforme, ou estar preparado para crescer. Podem ser classificadas em Escalabilidade horizontalmente (scale out) ou Escalabilidade vertical (scale up)

Tempo de Execução

O tempo de execução de um programa pode ser definido como o tempo que decorre desde que o primeiro processador inicia a execução até o último processador terminar.

Decomposto em tempo de computação, de comunicação e no tempo ocioso:

$$T_{exec} = T_{comp} + T_{comn} + T_{ocioso}$$

Tempo de computação: é o tempo despendido na computação

Tempo de comunicação: é o tempo que o algoritmo despende a enviar e receber mensagens

Tempo ocioso: surge quando um processador fica sem tarefas

Computação distribuída

A computação distribuída é o uso simultâneo de mais de um computador, conectado por uma **conexão de rede**, para resolver um problema.

Distribuir computação em vários computadores é uma ótima abordagem quando esses computadores são observados para interagir uns com os outros em toda a rede distribuída para resolver um problema maior em razoavelmente menos latência.

O objetivo de habilitar sistemas distribuídos inclui a capacidade de enfrentar um problema maior ou mais longo para processar por um computador individual.

Computação distribuída

A computação distribuída é realizada em um sistema distribuído, que é considerado um grupo de computadores que:

não apostam um relógio físico comum ou uma memória compartilhada,

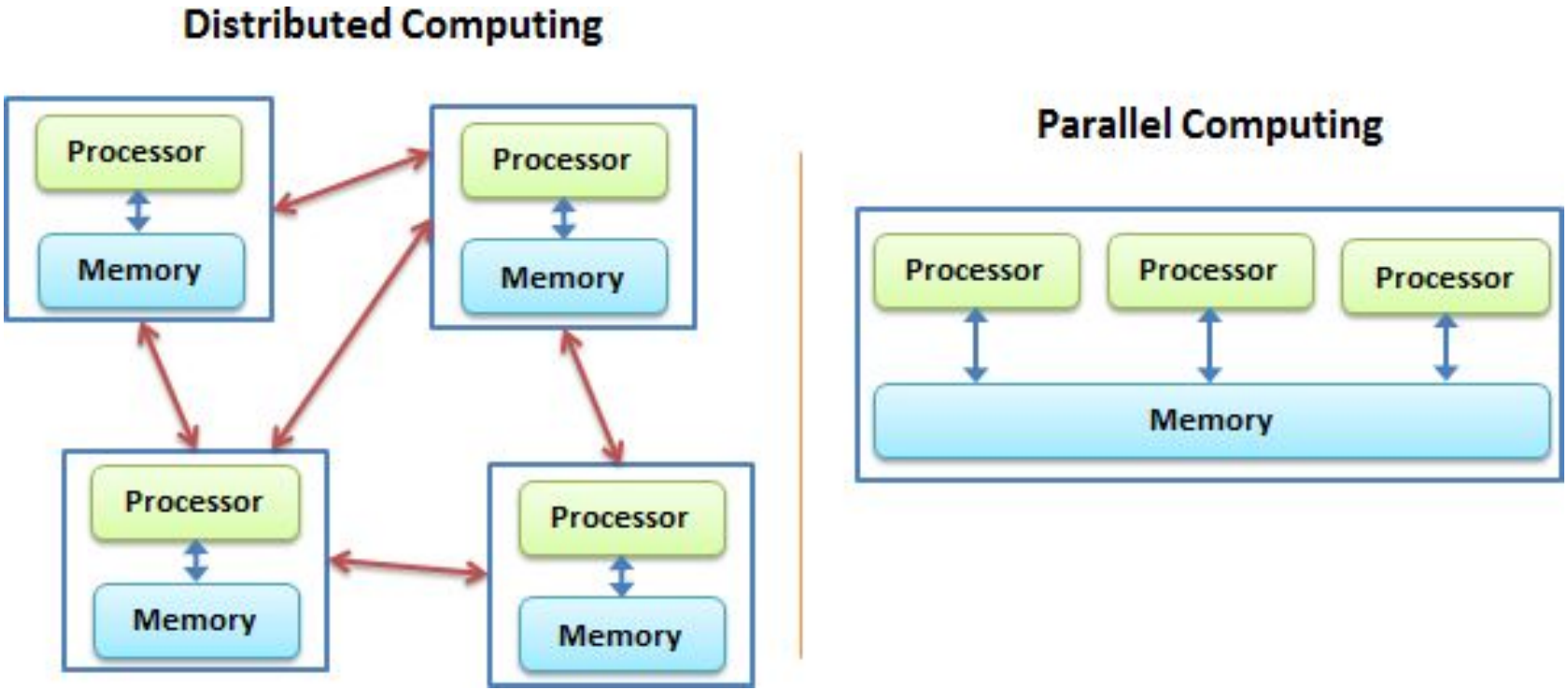
interagem com as informações trocadas por uma rede de comunicação (inter/intra),

com cada computador tendo sua própria memória,

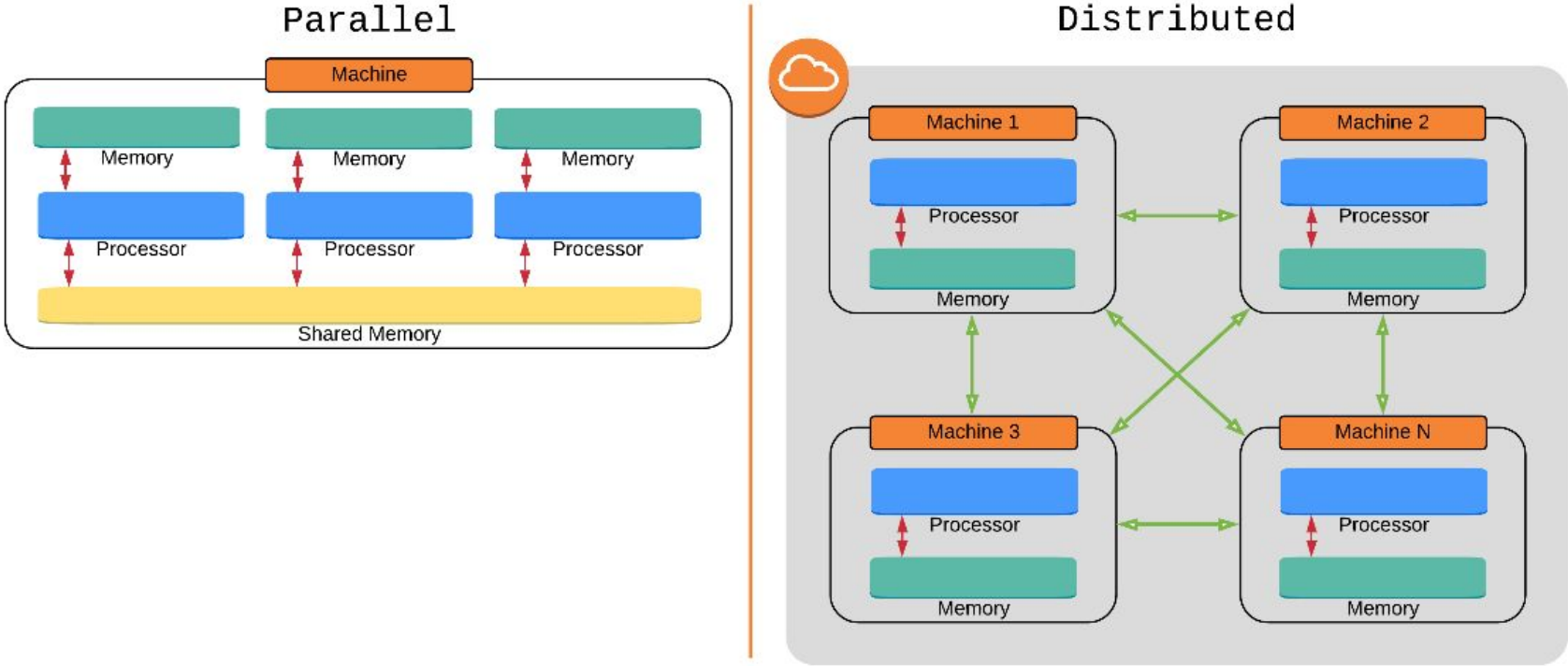
e roda em em seu próprio sistema operacional.

Normalmente, os computadores são ***semi autônomos*****, fracamente acoplados e cooperam para resolver um problema coletivamente**.

Computação distribuída x paralela



Computação paralela x distribuída



Por que fazer distribuído (não só paralelo)?

Escalabilidade: Como os sistemas distribuídos não têm os problemas associados à memória compartilhada, com o aumento do número de processadores, eles são obviamente considerados mais escaláveis do que sistemas paralelos.

Por que fazer distribuído (não só paralelo)?

Confiabilidade: O impacto da falha de qualquer subsistema ou de um computador na rede de computadores define a confiabilidade de um sistema.

Definitivamente, os sistemas distribuídos demonstram um aspecto melhor nesta área em comparação com os sistemas paralelos.

Por que fazer distribuído (não só paralelo)?

Compartilhamento de dados: O compartilhamento de dados fornecido por sistemas distribuídos é semelhante ao compartilhamento de dados fornecido por bancos de dados distribuídos.

Assim, várias organizações podem ter sistemas distribuídos com os aplicativos integrados para troca de dados.

Por que fazer distribuído (não só paralelo)?

Compartilhamento de recursos: Se existe um recurso de propósito especial e caro ou um processador, que não pode ser dedicado a cada processador no sistema, tal recurso pode ser facilmente compartilhado em sistemas distribuídos.

Por que fazer distribuído (não só paralelo)?

Heterogeneidade e modularidade: Um sistema deve ser flexível o suficiente para aceitar um novo processador heterogêneo a ser adicionado a ele e um dos processadores a ser substituído ou removido do sistema sem afetar a capacidade global de processamento do sistema.

Os sistemas distribuídos são observados como mais flexíveis nesse sentido.

Por que fazer distribuído (não só paralelo)?

Construção geográfica: A colocação geográfica de diferentes subsistemas de uma aplicação pode ser inerentemente colocada como distribuída.

O processamento local pode ser forçado pela baixa largura de banda de comunicação mais especificamente dentro de uma rede sem fio.

Por que fazer distribuído (não só paralelo)?

Econômico: Com a evolução dos computadores modernos, redes de alta largura de banda e estações de trabalho estão disponíveis a baixo custo, o que também favorece a computação distribuída por razões econômicas.

Considerações sobre Projetos de Sistemas Distribuídos

Sem relógio global: Sendo distribuídos em todo o mundo, não se pode esperar que os sistemas distribuídos tenham um relógio comum, e isso dá uma chance para a *assincronia intrínseca* entre os processadores que realizam a computação.

A coordenação do sistema distribuído geralmente depende de uma ideia compartilhada do momento em que ocorrem os programas ou estados.

No entanto, com sistemas distribuídos, sem relógio global, é um desafio alcançar a precisão com que os computadores da rede podem sincronizar seus relógios para refletir o momento em que a execução esperada do programa aconteceu.

Essa limitação espera que os sistemas da rede se **comuniquem através de mensagens** em vez de eventos baseados no tempo.

Considerações sobre Projetos de Sistemas Distribuídos

Distribuição geográfica: Espera-se que os sistemas individuais que participam do sistema distribuído sejam conectados através de uma rede, anteriormente por meio de uma Rede de Ampla Área (WAN), e agora com uma Rede de Estações de Trabalho/Cluster of Workstations (NOW/COW).

- NOWs são constituídas por estações de trabalho interligadas por tecnologia tradicional de rede (e.g., Ethernet)
- COW são redes de estações dedicadas ao processamento paralelo e podem ser vistas como NOW dedicada ao processamento paralelo e distribuído.

Espera-se que um sistema distribuído interno seja configurado dentro de uma conectividade LAN.

Considerações sobre Projetos de Sistemas Distribuídos

Sem memória compartilhada: Uma característica importante e fundamental da computação distribuída e o modelo de comunicação por mensagens é não ter memória compartilhada, o que também infere a inexistência de um relógio físico comum.

Considerações sobre Projetos de Sistemas Distribuídos

Independência e heterogeneidade: Os processadores do sistema distribuído são fracamente acoplados para que tenham suas próprias capacidades individuais em termos de velocidade e método de execução com sistemas operacionais versáteis.

Não se espera que eles façam parte de um sistema dedicado; no entanto, eles cooperam uns com os outros expondo os serviços e/ou executando as tarefas em conjunto como subtarefas.

Considerações sobre Projetos de Sistemas Distribuídos

Mecanismo de fail-over: muitas vezes vemos os sistemas de computador falharem, e espera-se do projeto definir o comportamento esperado com a consequência de possíveis falhas.

Os sistemas distribuídos são considerados falhos na integração, bem como nos subsistemas individuais.

Uma falha na rede pode resultar no isolamento de um indivíduo ou de um grupo de computadores no sistema distribuído; no entanto, eles ainda podem estar executando os programas que eles devem executar.

Na realidade, os programas individuais podem não ser capazes de detectar tais falhas de rede ou intervalos de tempo. Da mesma forma, a falha de um determinado computador, um sistema sendo encerrado abruptamente com um programa abrupto ou falha no sistema, pode não ser imediatamente reconhecida pelos outros sistemas/componentes da rede com a qual o computador falho geralmente se comunica.

Considerações sobre Projetos de Sistemas Distribuídos

Preocupações de segurança: sistemas distribuídos sendo criados em uma Internet compartilhada são propensos a ataques e vulnerabilidades mais não autorizados.

Middleware

Do ponto de vista etimológico, **middle** em inglês significa meio e o sufixo **ware** é usado para denotar conjunto ou para transformar a palavra na forma coletiva. Desta forma, em uma tradução simplificada, middleware denota as tecnologias intermediárias. Intermediárias entre o que? Intermediárias entre quem?

Uma solução de Middleware fica entre a aplicação que o usuário enxerga e as fontes de informações.

Exemplos: Servidores de Aplicação Java EE, Business intelligence, Mensageria, etc.

Middleware

O middleware é o software que se encontra entre o sistema operacional e os aplicativos nele executados (azure.microsoft.com: *what is midleware?*)

Se aplica a uma camada de software que fornece uma abstração de programação, assim como o **mascaramento da heterogeneidade** das redes, do hardware, dos sistemas operacionais e das linguagens de programação subjacentes. O CORBA (Common Object Request Broker), é um exemplo. Alguns middlewares, como o Java RMI (Remote Method Invocation), suportam apenas uma linguagem de programação.

Exemplos comuns de middleware incluem middleware de banco de dados, middleware de servidor de aplicativos, middleware orientado a mensagens, middleware de web e monitores de processamento de transações.

Concorrência

Concorrência: a presença de múltiplos usuários em um sistema. Por existirem diversos elementos em um SD, é possível que múltiplas requisições ou múltiplos acessos sejam realizados ao mesmo recurso, no mesmo instante de tempo.

Em ambiente concorrente, cada recurso deve ser projetado para manter sempre a **consistência**.

Em uma rede de computadores, a execução concorrente de programas é a norma. Posso fazer meu trabalho em meu computador, enquanto você faz o seu em sua máquina, compartilhando recursos como páginas Web ou arquivos, quando necessário.

Exemplo: dois usuários estão no mesmo momento acessando a mesma tabela no Banco de Dado

RESTful

REST (Representational State Transfer) é uma arquitetura de desenvolvimento que trabalha com protocolo Web. Já **o RESTful** é um serviço web que utiliza o REST quando implementamos *Web Services*, ou seja, sistemas que utilizam os princípios REST são chamados de RESTful.

A arquitetura REST foi originalmente projetada para servir ao protocolo HTTP usado pela World Wide Web.

RESTful: REpresentational State Transfer

As características de um sistema REST são definidas por seis regras de design:

Cliente-servidor: deve haver uma separação entre o servidor que oferece um serviço e o cliente que o consome.

Sem monitoração de Estado (Stateless): cada solicitação de um cliente deve conter todas as informações exigidas pelo servidor para executar a solicitação. Em outras palavras, o servidor não pode armazenar informações fornecidas pelo cliente em uma solicitação e usá-lo em outra solicitação.

Cacheable: o servidor deve indicar ao cliente se as solicitações podem ser armazenadas em cache ou não.

Sistema em camadas: a comunicação entre um cliente e um servidor deve ser padronizada de forma a permitir que os intermediários respondam a solicitações em vez do servidor final, sem que o cliente tenha que fazer algo diferente.

Interface uniforme: o método de comunicação entre um cliente e um servidor deve ser uniforme.

Código sob demanda: os servidores podem fornecer código executável ou scripts para que os clientes executem em seu contexto. Essa restrição é a única que é opcional.

Internet das coisas (IoT - Internet of Things)

É um conceito que se refere à interconexão digital de objetos cotidianos com a internet.

O conceito de Internet das Coisas, ou Internet of Things (IoT) é o de uma enorme rede de dispositivos conectados, mas não limitada ao seu computador, smartphone, tablet ou set-top box, entre outros, são gadgets que dependem da internet para funcionar apropriadamente, assim como equipamentos de grande porte como servidores de grandes empresas.

O foco da IoT é voltado para todos os demais equipamentos do dia a dia de um indivíduo, instituição, empresa ou mesmo de uma cidade inteira, aqueles que você não imaginaria num primeiro momento que podem se beneficiar da rede.

Contêineres

Os containers proporcionam uma maneira padrão de empacotar código, configurações e dependências de seu aplicativo em um único objeto. Eles compartilham um sistema operacional instalado no servidor e são executados como processos isolados de recursos. Isso permite fazer implantações rápidas, confiáveis e consistentes, independentemente do ambiente. *Fonte: https://aws.amazon.com/pt/containers/*

Como eles não incluem sistema operacional completo, os contêineres exigem recursos computacionais mínimos, além de serem rápidos e fáceis de instalar.

Micro-serviços

O termo “Arquitetura de Microsserviços (Microservice Architecture)” surgiu nos últimos anos para descrever uma maneira específica de desenvolver software como suítes de serviços com deploy independente. *(Martin Fowler)*.

Microsserviços são uma abordagem de arquitetura para a criação de aplicações. O que diferencia a arquitetura de microsserviços das abordagens monolíticas tradicionais é como ela decompõe a aplicação por funções básicas. Cada função é denominada um serviço e pode ser criada e implantada de maneira independente. Isso significa que cada serviço individual pode funcionar ou falhar sem comprometer os demais. (*redhat*)

Vídeo

Assista o vídeo abaixo sobre Microsserviços. Aproximadamente 15 minutos.

O que são Microsserviços?
https://hipsters.tech/o-que-sao-microsservicos-microservices/

Troca de Mensagens

A troca de mensagens é uma forma comunicação entre processos que consiste em enviar mensagens a destinatários, sob forma de invocação de funções, sinais ou pacotes de dados.

A comunicação por troca de mensagens em uma rede de computadores pode ser afetada por atrasos, sofrer uma variedade de falhas e ser vulnerável a ataques contra a segurança.

Troca de Mensagens

As mensagens são objetos de dados cuja estrutura e aplicação são definidas pelas próprias aplicações que a usarão. Sendo a troca de mensagens feita através de primitivas explícitas de comunicação:

- **send**(destino, mensagem) envio da mensagem para o destino
- **receive**(origem, mensagem) recebimento da mensagem enviada pela origem

Troca de Mensagens: forma de comunicação

Direta: há indicação do processo receptor (no **send**) e do emissor (no **receive**).

```
send(process, msg)

receive(process, msg)
```

Indireta: no **send** há o envio para uma porta ou mailbox sem o conhecimento de qual será o receptor. No **receive** há obtenção da mensagem guardada no mailbox, possivelmente desconhecendo a identidade do processo emissor

```
send(mailbox, msg)

receive(mailbox, msg)
```

Troca de Mensagens: forma de sincronização

Síncrono ou Bloqueante

Send: espera até que a mensagem seja recebida e confirmada pelo receptor.

Receive: aguarda a disponibilidade da mensagem.

Assíncrona ou Não Bloqueante

Send: envia a mensagem mas não espera até que a mensagem seja recebida/confirmada pelo receptor.

Receive: se a mensagem estiver disponível, recebe a mensagem, caso contrário continua o processamento retornando uma indicação de que a mensagem não estava disponível.

Serviços Web

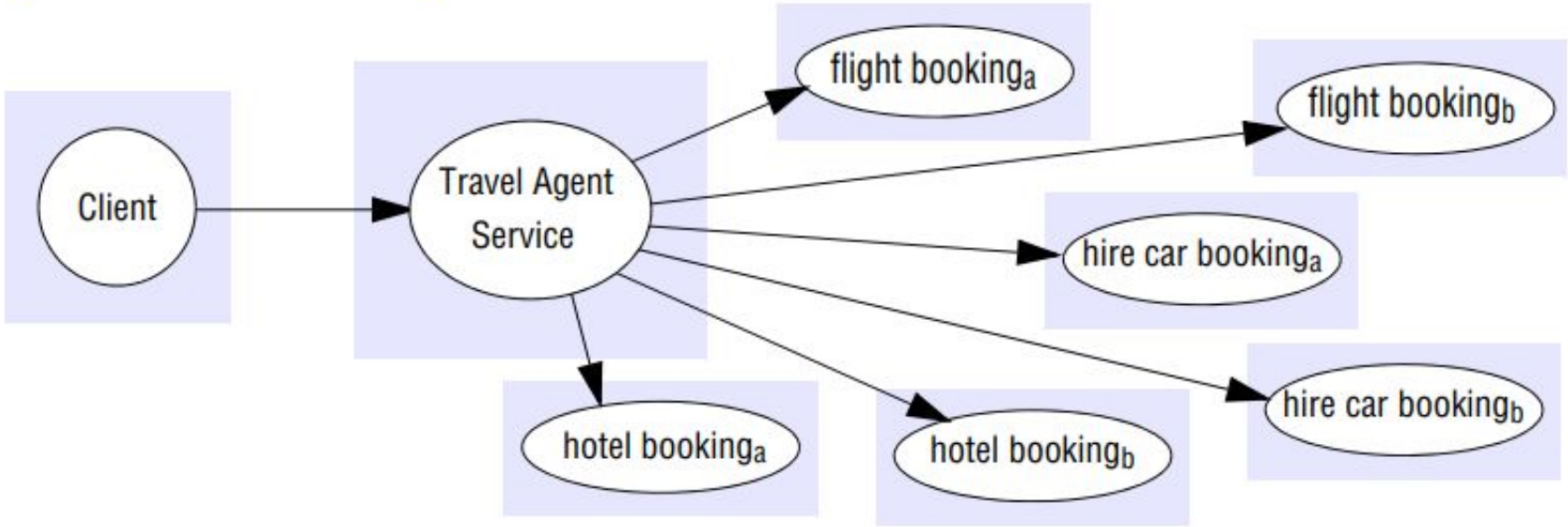
Um serviço Web (*web service*) fornece uma interface de serviço que permite aos clientes interagirem com servidores de uma maneira mais geral do que acontece com os navegadores Web.

Os clientes acessam operações de um serviço Web por meio de requisições e respostas formatadas em XML e, normalmente, transmitidas por HTTP.

Os serviços Web são cada vez mais importante nos sistemas distribuídos: eles suportam atividades conjunta na Internet global, incluindo a área fundamental de integração de empresa para empresa (B2B) e também a emergente cultura de *“mashup”*, permitindo que desenvolvedores criem software inovador em cima da base de serviços já existentes.

Serviços Web

Figure 9.2 The 'travel agent service' combines other web services



Serviços Web

Um Web Service pode ser definido como uma coleção de protocolos abertos e padrões para troca de informações entre sistemas ou aplicativos.

Nem todos os serviços executados por um aplicativo podem ser descritos como um Web Service.

Um serviço pode ser tratado como um Web Service se:

- O serviço é detectável através de uma pesquisa simples ele usa um formato XML padrão para mensagens está disponível em redes Internet/intranet.
- É um serviço auto descritivo através de uma sintaxe XML simples
- O serviço está aberto para, e não vinculado a, qualquer sistema operacional/linguagem de programação

SOAP e RESTFUL são os dois principais tipos de Web Services.

Serviços Web: Arquitetura

Como parte de uma arquitetura de Web Services, existem três funções principais:

- **Provedor de serviços (Provider)**: é o programa que implementa o serviço acordado para o Web Service e expõe o serviço através da Internet/intranet para outros aplicativos interagir com.
- **Solicitador de serviço (Requestor):** é o programa que interage com o Web Service exposto pelo provedor de serviços. Ele faz uma invocação para o serviço da Web através da rede para o provedor de serviços e troca de informações.
- **Registro de serviço (Registry)**: atua como o diretório para armazenar referências aos serviços da Web.

Referências

Raja Malleswara Rao Pattamsetti, **Distributed Computing in Java 9**, https://www.packtpub.com/product/distributed-computing-in-java-9/9781787126992

COULOURIS, G. et all. Sistemas Distribuídos: Conceitos e Projetos. 5ª edição. Bookmann, 2013.

Italo Santa, Design de Sistemas Distribuídos — Escalonamento Vertical e Horizontal, https://medium.com/, acesso em 2021.