

# WebServices e MBaaS

- Aula 01 - Introdução
- Mark Joselli
- [mark.joselli@pucpr.br](mailto:mark.joselli@pucpr.br)
- <https://www.linkedin.com/in/mjoselli/>

A photograph of a man with dark hair and glasses, wearing a dark jacket over a blue shirt. He is holding a microphone and gesturing with his hands while speaking. A name tag around his neck has 'SSBRMES' printed on it.

# Quem sou eu?

- Professor na PUC-PR
- Consultor pela Mark Joselli Consultoria
- CFO da Boa Pisada
- Pesquisador de GPGPU, games e mobile
- Engenheiro com Mestrado e Doutorado em computação

# Temas de estudo

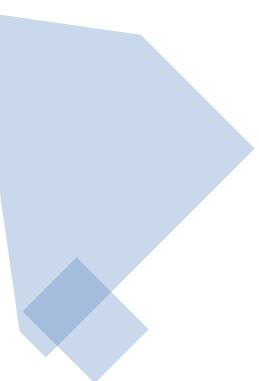
- Arquitetura de Web Services.
- SOA
- WSDL e SOAP
- Padrão REST
- Monolith x SOA x Microservices x Serverless
- Banco de dados
  - SQL x NoSQL x NewSQL (GraphQL)
- Projeto e implementação de MBaaS

# Objetivos

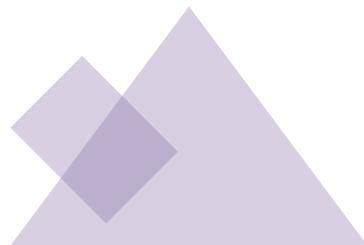
- Apresentar os conceitos de Serviços Web
- Conhecer os padrões SOA,
- Conhecer os tipos de bancos de dados
- MBaaS

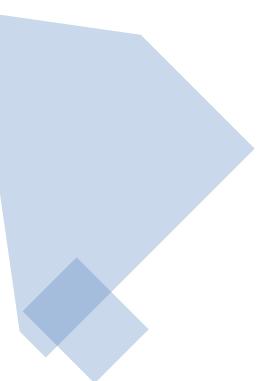
# Avaliação

- Implementar um App com uso de serviços e MBaaS
- Podem ser feitos em duplas

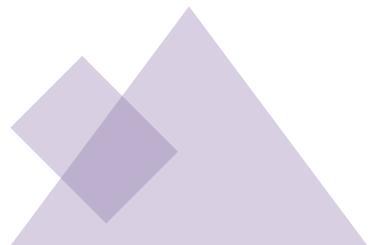


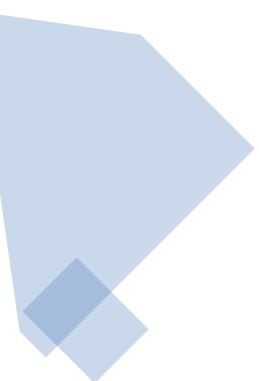
# Plano de aula

- Aula 1 – Introdução de Webservices e consumo;
  - Aula 2 – Padrão Rest/Rotas e Tipos de serviços
  - Aula 3 – Banco de dados e MBaaS
  - Aula 4 – Autenticação (EAD)
  - Aula 5 – Serverless
  - Aula 6 – Projeto (EAD)
- 

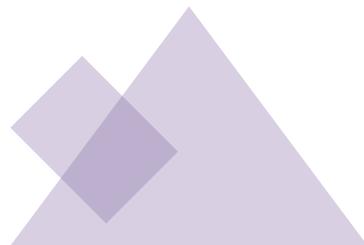


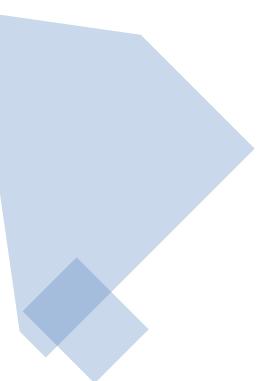
# Agenda de hoje

- Serviços
  - SOA
  - WebServices
  - SOAP
  - WSDL
  - REST
- 

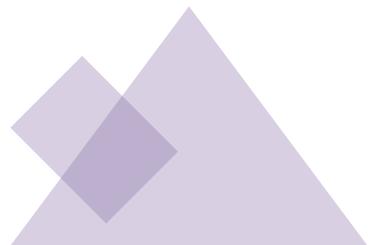


# Objetivos de hoje

- Apresentar os conceitos de Serviços Web
- 



# Criação da World Wide Web

- 12 de março de 1989 por Tim Berners-Lee no CERN
  - Sistema capaz de gerenciar documentos utilizando endereços e protocolos
- 

**Welcome to the Universe of HyperText**

**File View**

**WorldWideWeb**

- Info
- Navigate
- Document
- Edit
- Find
- LINKS
- Style
- Print
- Page layout
- Windows
- Services
- Hide
- Quit

**Home**

Access to this information is provided as part of the [WorldWideWeb](#) project. The WWW project does not take responsibility for the accuracy of information provided by others.

**How to proceed**

References to other information are represented like [this](#). Double-click on it to jump to related information.

**General CERN Information sources**

Now choose an area in which you would like to start browsing. The system currently has access to three sources of information. With the index(es), you should use the keyword search option on your browser.

[CERN Information](#): A general keyword index of information made available by the computer centre, including CERN, Cray and IBM help files, "Whitelists", and the Computer Newsletter (CNL). (This is the same data on CERNVM which is also available on CERNVM with the VM [END](#) command.)

[Yellow Pages](#): A keyword index to the CERN telephone book by function.

You can access the Internet news scheme ([See information for news users](#)). News articles are distributed typically CERN-wide or worldwide, and have a finite lifetime.

may be of general interest at CERN include

[CERN](#), [Meyrin](#), [Fermilab](#), [SLAC](#), [CERN](#), [Geneva](#), [SPC](#), [LEP](#), [AAEP](#), [GEKU](#), [Divezone](#), [Versoix](#), [Annemasse](#), [St. Julien](#), [Yonn](#), [Anneycy \(LAPP\)](#), [Parfa](#).

0 5 km

**HyperMedia Browser/Editor**

An exercise in global information availability

by Tim Berners-Lee

1990.01, CERN, Distribution restricted, see the terms. TEST VERSION ONLY

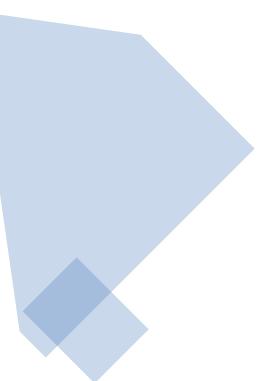
**Text:** Text which is not constrained to be linear.

**media:** Information which is not constrained linear... or to be text.

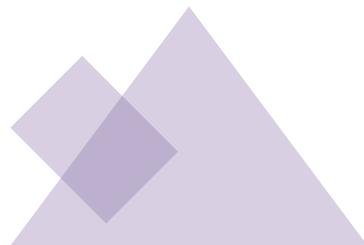
This is the first version of the NeXTStep WorldWideWeb application with the http://www library. Bug reports to [WWW-Bug@info.cern.ch](mailto:WWW-Bug@info.cern.ch) check the list of known bugs in the web too.

This was the original prototype for the World-Wide Web. Many servers for other platforms now exist. Read the web for details.

You should configure the newsreader code in this application to know where your local news (NNTP) server is. Type in a terminal window:

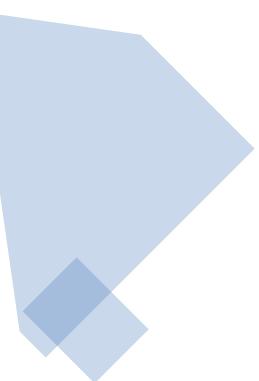


# Serviços

- No dia a dia, frequentemente usamos serviços para nos ajudar.
    - Serviços são ações valiosas que ajudam a atender uma demanda.
    - Por exemplo, pedir comida, pedir um uber...
  - Software também faz uso de serviços.
  - Na indústria de software, implantar um serviço significa fornecer uma funcionalidade que outros softwares podem usar.
- 

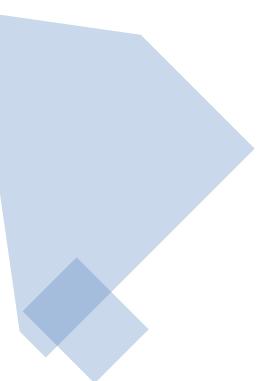
# O que é um serviço?

- Do dicionário:
  - Na economia, um serviço é o equivalente a um bem não material.
  - A prestação de serviços é uma atividade econômica que não resulta em propriedade.
  - Podemos dizer que é um processo que gera benefícios ao facilitar uma mudança de clientes,
    - uma mudança em suas posses físicas ou uma mudança em seus bens intangíveis.

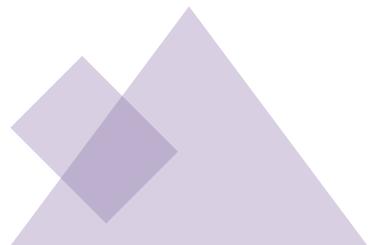


# O que é um serviço?

- Um serviço lida com um processo de negócios, uma tarefa técnica ou fornece dados de negócios.
    - Cálculo de uma cotação do seguro.
    - Acessar um banco de dados.
    - Detalhes necessários para a construção de uma GUI.
  - Um serviço pode acessar outro serviço e responder a diferentes tipos de solicitantes.
  - Um serviço é relativamente genérico.
    - Alterações em um solicitante requerem poucas ou nenhuma alteração no serviço.
    - Alterações na lógica interna de um serviço exigem poucas ou nenhuma alteração no solicitante
- 

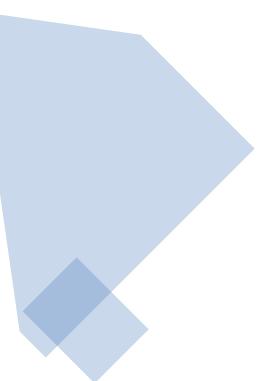


# Propriedades dos serviços

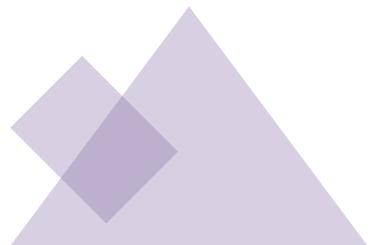
- Um serviço representa uma atividade de negócios com um resultado específico.
  - Um serviço é autônomo.
    - Projetado para manter o baixo acoplamento (loose coupling).
  - Um serviço é uma caixa preta para seus consumidores.
    - Apenas sua interface precisa ser entendida.
    - Pode lidar com interações dentro e fora de sua empresa, geograficamente distribuídas pelo mundo.
  - Um serviço pode consistir em outros serviços subjacentes.
- 

# Arquitetura baseada em serviços

- Uma forma de organizar softwares para que as empresas possam responder rapidamente às mudanças nas exigências do mercado.
- A arquitetura de um sistema conecta serviços.
  - Pequenas unidades personalizadas de software que são executadas em uma rede.
  - Os desenvolvedores disponibilizam serviços em uma rede para permitir que os usuários os combinem e os reutilizem.
  - Os serviços se comunicam passando dados em um formato bem definido, compartilhado ou coordenando atividades entre outros serviços.



# Arquitetura baseada em serviços

- Os serviços são frequentemente associados a duas funções:
    1. o solicitante de serviço, que é o software que solicita o serviço (cliente)
    2. o provedor de serviços, que atende solicitações (servidor)
  - A arquitetura orientada a serviços examina como construir, usar e combinar serviços.
  - Em vez de criar grandes suítes de software que fazem tudo, a arquitetura orientada a serviços trabalha construindo camadas e projetando uma arquitetura que suporte seu uso.
- 

# A TECNOLOGIA WEB COMO PLATAFORMA PARA DIVULGAÇÃO DE INFORMAÇÕES

## Paradigma Cliente Servidor

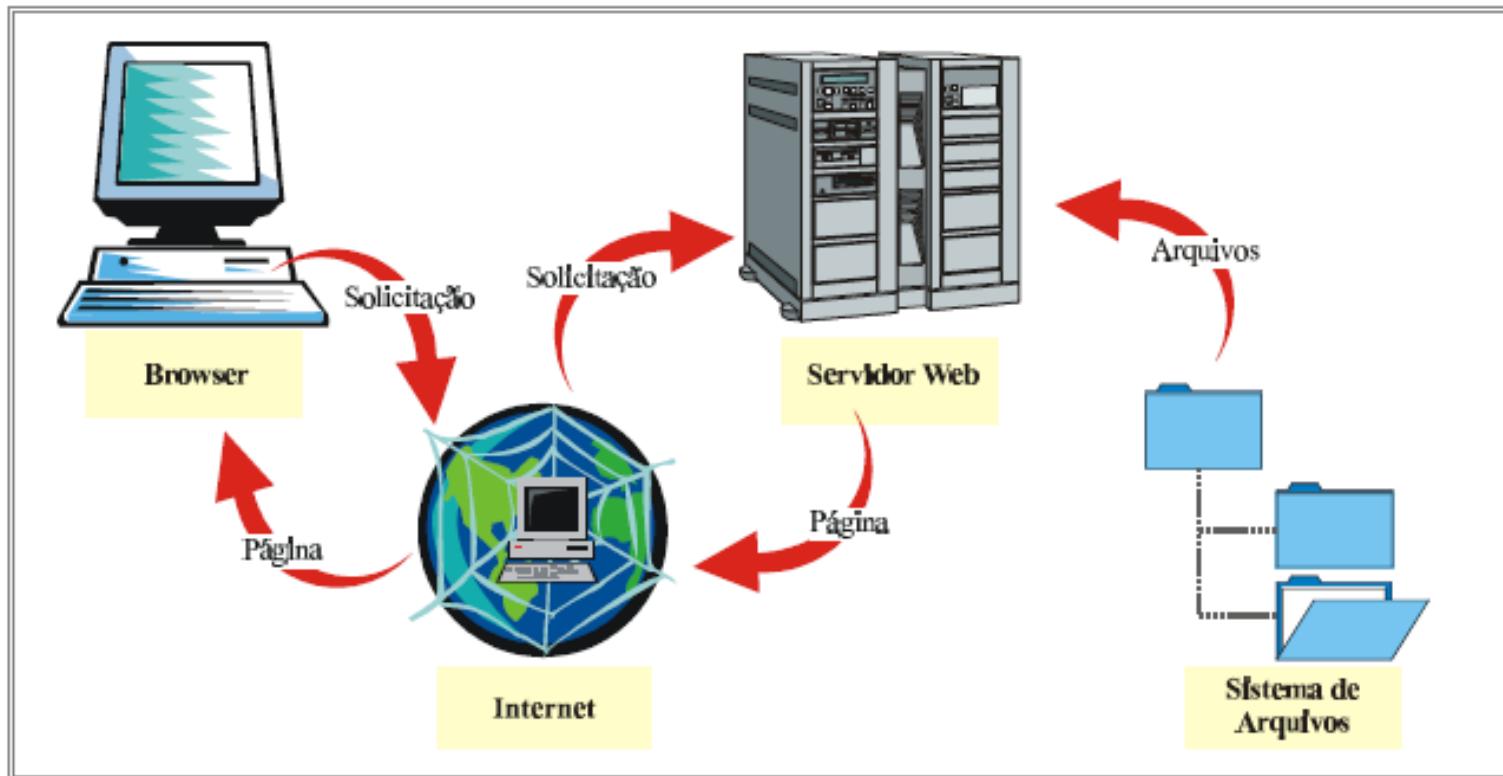


Figura 2 Funcionamento da tecnologia *Web* para acesso a um *Web Site* “tradicional”

# Começo do desenvolvimento de serviços

- Negócios e governo descobriram o valor da computação.
- Os requisitos de negócios foram levantados e programados.
- As aplicações eram projetadas para departamentos específicos/necessidades de negócios.
- As aplicações eram projetadas como uma entidade, combinando a lógica da interface do usuário, processamento de negócios e acesso a dados.

# Problema

Cada aplicação é auto-contida.  
Uma visão da interação do usuário.

Difícil encontrar pontos de integração  
Por causa do seu design, atualizações de um tipo de lógica  
exigem testar vários tipos de comportamento.  
Elas são mais difíceis de entender, pois a lógica é geralmente  
corrigida em vez de reescrita.

# Interoperabilidade sempre foi tentada!

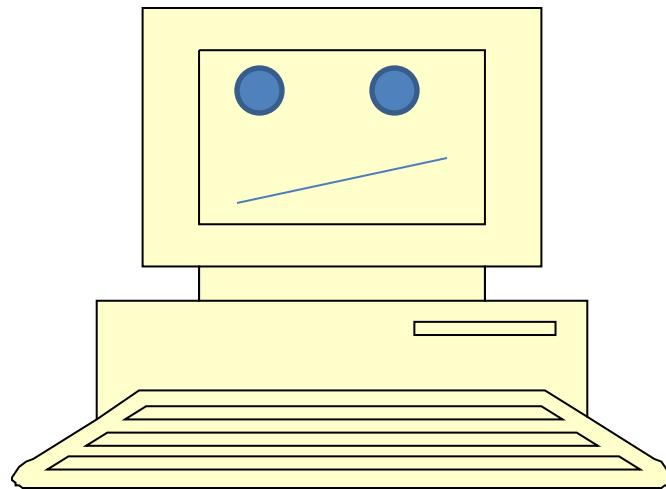




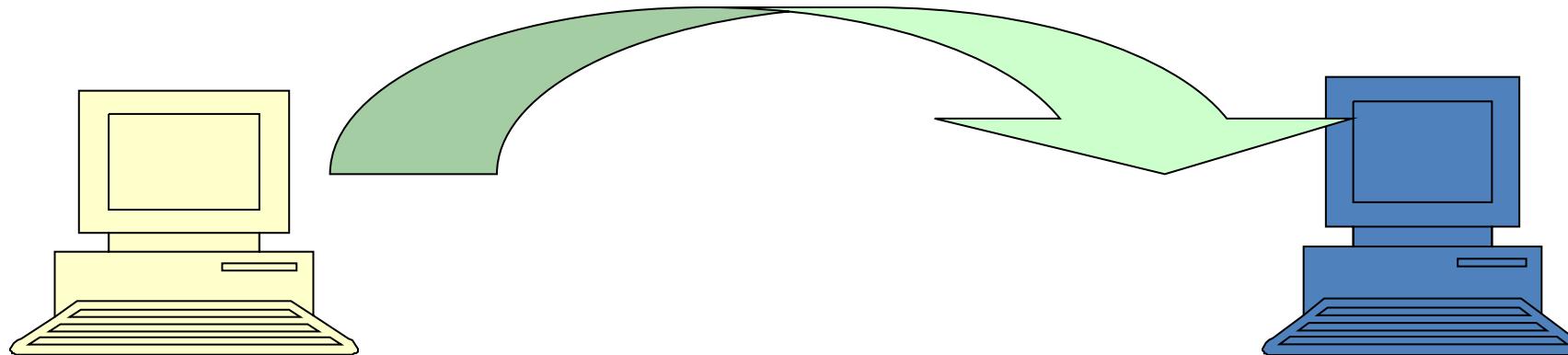
# Webservice

- “Um sistema de software projetado para suportar comunicação interoperável maquina-a-maquina através de uma rede.”
- Os serviços web podem implicar compensações.
- A facilidade de utilização dos serviços existentes deve ser equilibrada em relação às qualidades dos serviços, que não estão sob o controle dos desenvolvedores.
- Nesses casos, os requisitos não funcionais tornam-se muito importantes. Estes podem incluir:
  - Tempo de resposta
  - Capacidade de suporte
  - Disponibilidade do serviço

# Cenário de motivação para uso de web services



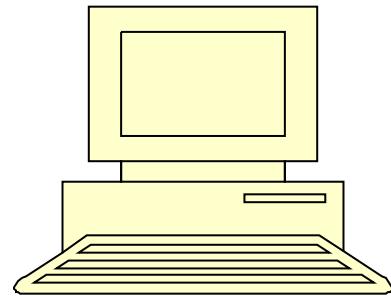
# métodos GET e POST



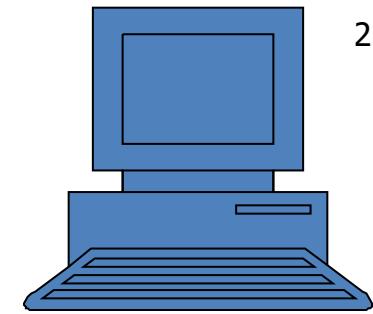
cliente web

servidor web

# Requisição Cliente X Servidor

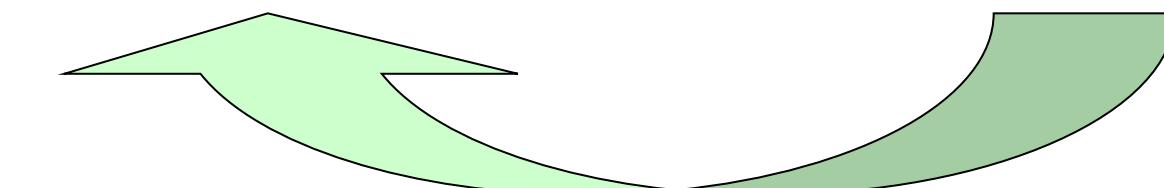


cliente web



servidor web

Formatos



HTML/CSS/JS  
Fragmentos de HTML  
JSON  
XML  
etc...

Resposta do Servidor ao Cliente

# XML/HTML/Json

- Sistemas web possuem muitos tipos diferentes de formatos para armazenar e expressar conteúdo
- 3 formatos comuns vão ser explorados durante nosso curso.

# HTML

```
<!DOCTYPE html>
<html>
  <head>
    <title>Page Title</title>
  </head>
  <body>
    <h1>This is a Heading</h1>
    <p>This is a paragraph.</p>
  </body>
</html>
```

# XML

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<receita nome="pão">
<titulo>Pão simples</titulo>
<ingredientes>
    <ingrediente>Farinha</ingrediente>
    <ingrediente>Fermento</ingrediente>
    <ingrediente>Água</ingrediente>
    <ingrediente>Sal</ingrediente>
</ingredientes>
</receita>
```

# Json

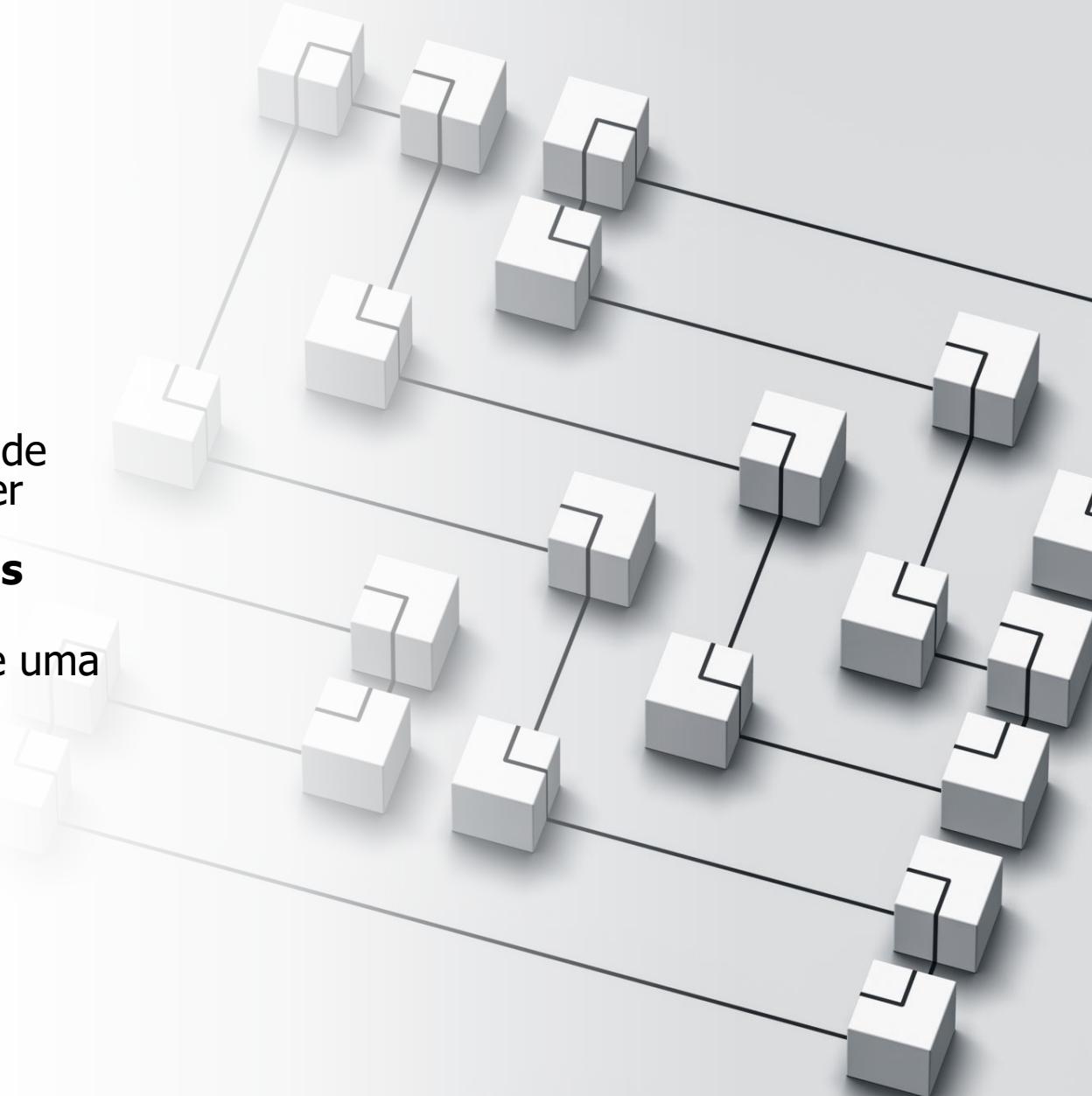
```
{ "Aluno": [  
    { "nome": "João", "notas": [ 8, 9, 7 ] },  
    { "nome": "Maria", "notas": [ 8, 10, 7 ] },  
    { "nome": "Pedro", "notas": [ 10, 10, 9 ] }  
]
```

# Web Service **não** é:

Site para Web Genérico  
que disponibilize um  
serviço

## Web Service é:

- Arquitetura para criação de aplicações que podem ser acessadas a partir de **diferentes plataformas (interoperabilidade)**;
- **Identificado** através de uma **URL**;
- Aplicação que **recebe e retorna** mensagens em formatos padrão:
  - XML;
  - XML/SOAP;
  - XML/RSS;
  - JSON...

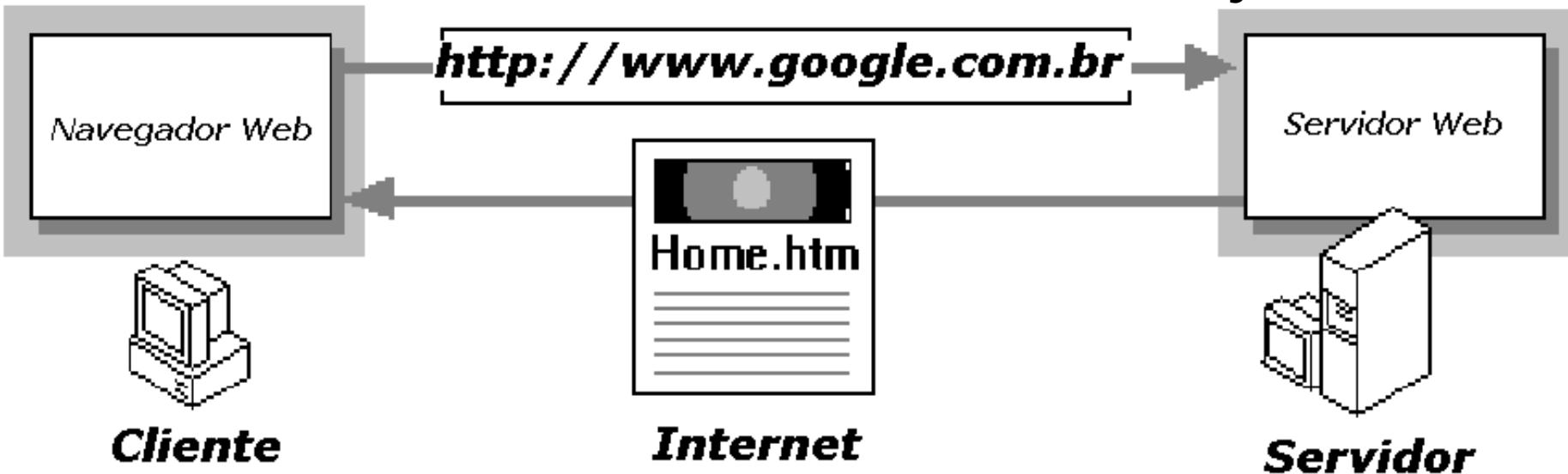


# Características de Web Services

- Utilizam o protocolo HTTP como transporte
- Podem ser utilizados em ambientes protegidos com firewall sem a abertura de portas adicionais
- São baseado em padrões abertos mantidos pelo W3C
- Permite interoperabilidade com diversas plataformas e linguagens
- Oferece baixo acoplamento das aplicações

# Etapas do HTTP

- ✓ 1 - **CONEXÃO** - Estabelecer a comunicação;
- ✓ 2 - **REQUISIÇÃO** - Fazer o pedido;
- ✓ 3 - **RESPOSTA** - Dar a resposta;
- ✓ 4 - **FECHAMENTO** - Concluir a comunicação.



# HTTP Requests

- Uma requisição HTTP consiste em uma linha com a solicitação, cabeçalhos, uma linha em branco e, às vezes, um corpo de mensagens.
  - A linha de solicitação inclui o método de solicitação, solicitação URI e protocolo.
  - Solicitações de clientes podem ter cabeçalhos de diferentes tipos.
  - Uma linha em branco segue os cabeçalhos.
  - Os corpos de mensagens podem ser documentos HTML, JSON, parâmetros codificados ou outro conteúdo semelhante.

# HTTP Responses

- Uma resposta de HTTP consiste em uma linha de status, cabeçalhos, uma linha em branco e, às vezes, um corpo de mensagens.
  - A linha de status inclui a versão do protocolo e o código de status HTTP. O código de status HTTP informa o cliente sobre o status da solicitação.
  - Solicitações de clientes podem ter cabeçalhos de diferentes tipos.
  - Uma linha em branco segue os cabeçalhos.
  - Os corpos de mensagens podem ser documentos HTML, JSON, parâmetros codificados ou outro conteúdo semelhante.

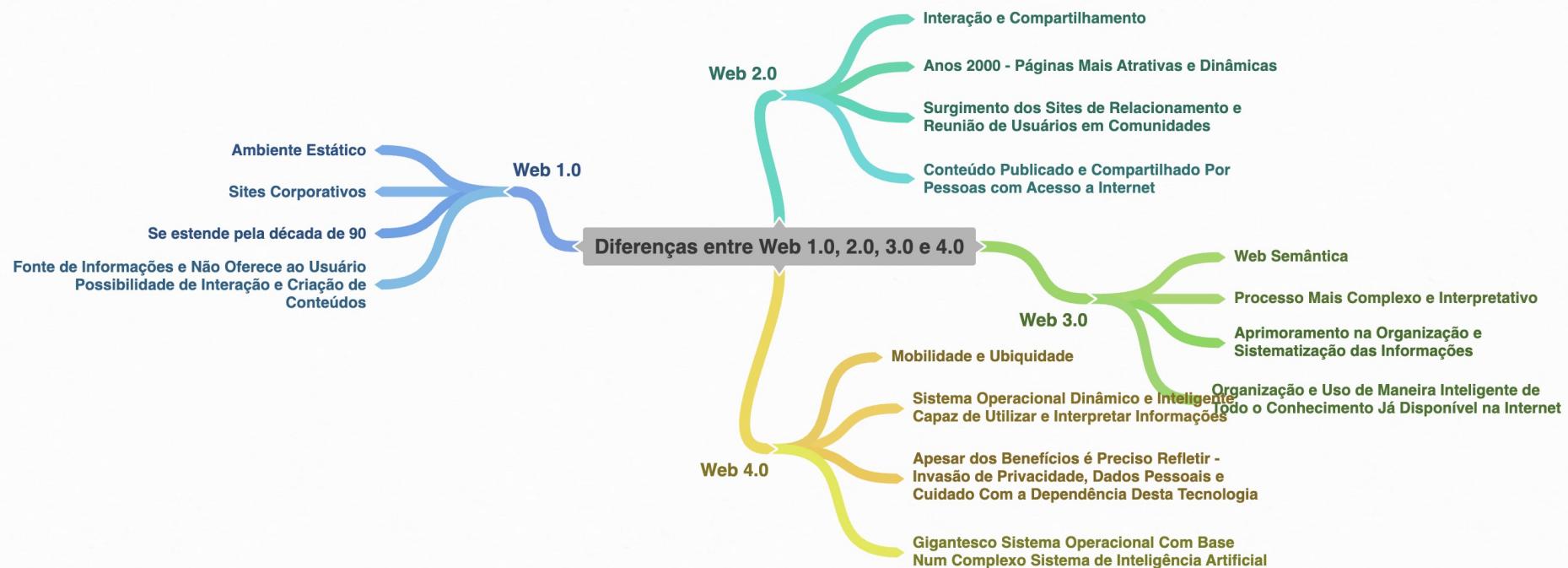
# Exemplo request-response

## Request

```
GET /reservation?number=17021&name=mark HTTP/1.1
Host: camping.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X x.y;
rv:10.0) Gecko/20100101 Firefox/10.0
Accept: */*
```

## Response

```
HTTP/1.1 200 OK
Date: Wed, 17 May 2021 19:15:56 GMT Content-Length:
10571
Content-Type: text/html; charset=utf-8 Last-Modified:
Wed, 17 May 2021 19:12:21 GMT
<!DOCTYPE html> <html lang="en"> ... </html>
```



# URIs e URLs

- Universal Resource Identifiers (URIs) são endereços usados para identificar recursos.
- Universal Resource Locators (URLs) são um subconjunto de URIs, que são usados para localizar recursos.
- Ambos identificam o recurso, mas as URLs também dizem ao protocolo como localizar e acessar o recurso.
- Os URLs fornecem o protocolo e o nome de domínio ou endereço IP da máquina em que o recurso é armazenado e a localização do recurso na máquina.
- Todas as URLs são URIs, mas nem todas as URIs são URLs.

- Número IP: 200.192.168.1
- Host: <http://www.google.com.br>



http://www.

# Endereços IP

- Identificação de um computador na Internet
  - Endereço IP ⇔ Número de telefone
  - Em geral: a cada computador ou máquina na Internet corresponde um único endereço IP
- Os endereços IP são constituídos por 4 números separados por pontos. Têm a forma: xxx.xxx.xxx.xxx, onde xxx é um número entre 0 e 255
- Quando um utilizador fica *online* é atribuído um endereço IP temporário ao computador, que dura enquanto estiver activa a ligação.
- Sistema de endereçamento numérico
  - **Problema:** fraca maneabilidade/legibilidade
  - **Solução:** Sistema de endereçamento por **nomes**-palavras (**DNS**) baseado no sistema de endereçamento numérico.

# DNS - Domain Name System

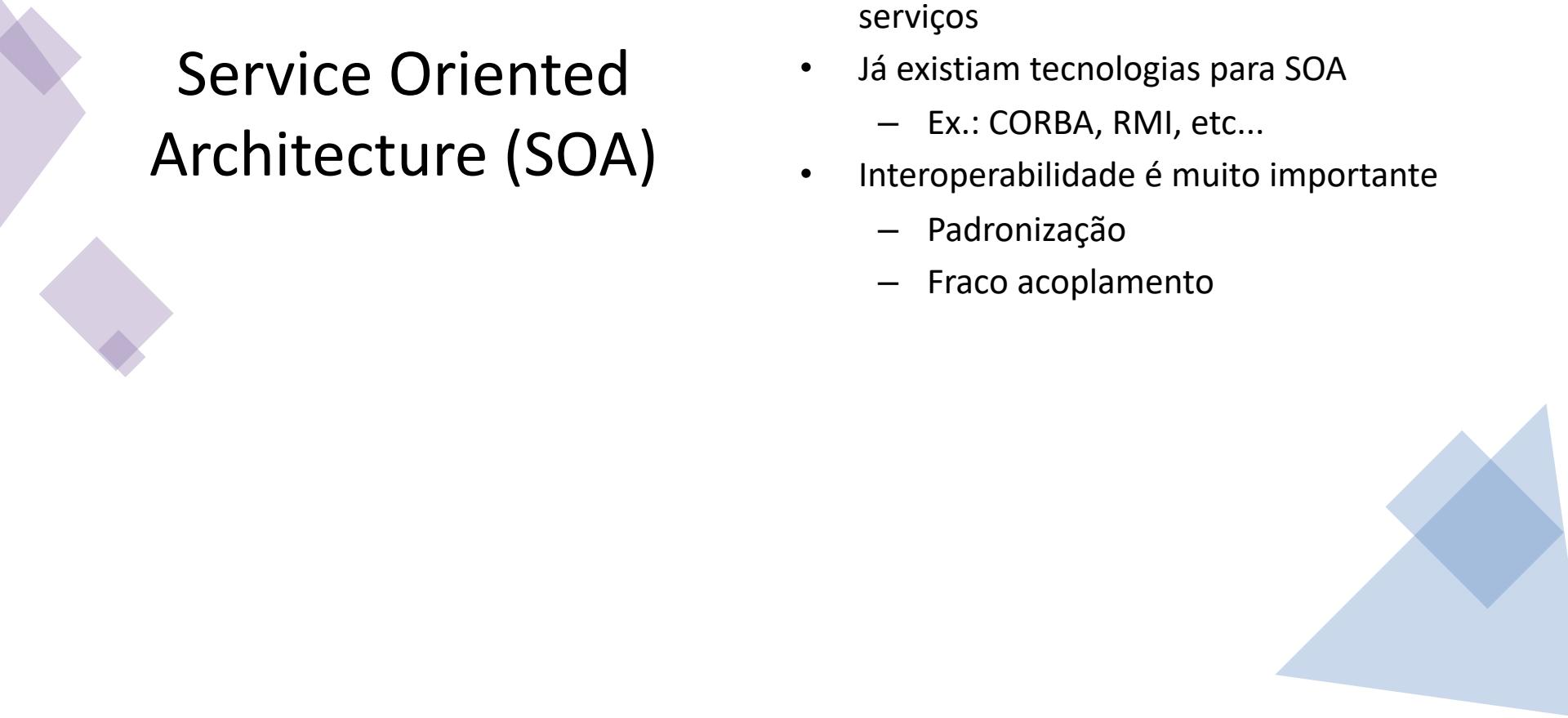
- Exemplos: www. Pucpr . Br
  - Dominio principal: br
  - Subdominio: pucpr
  - Host: www

# DNS - Domain Name System

- **Domínios:** **Entidades:**
- **com** comerciais
- **edu** educativas
- **gov** governamentais
- **int** internacionais
- **mil** militares
- **net** da Internet
- **org** organizações sem fins lucrativos
- **pt** Portugal
- **br** Brasil
- **es** Espanha
- **us** Estados Unidos
- **fr** França
- **uk** Reino Unido
- **au** Austrália
- **de** Alemanha
- **...**

# Visão Tecnológica

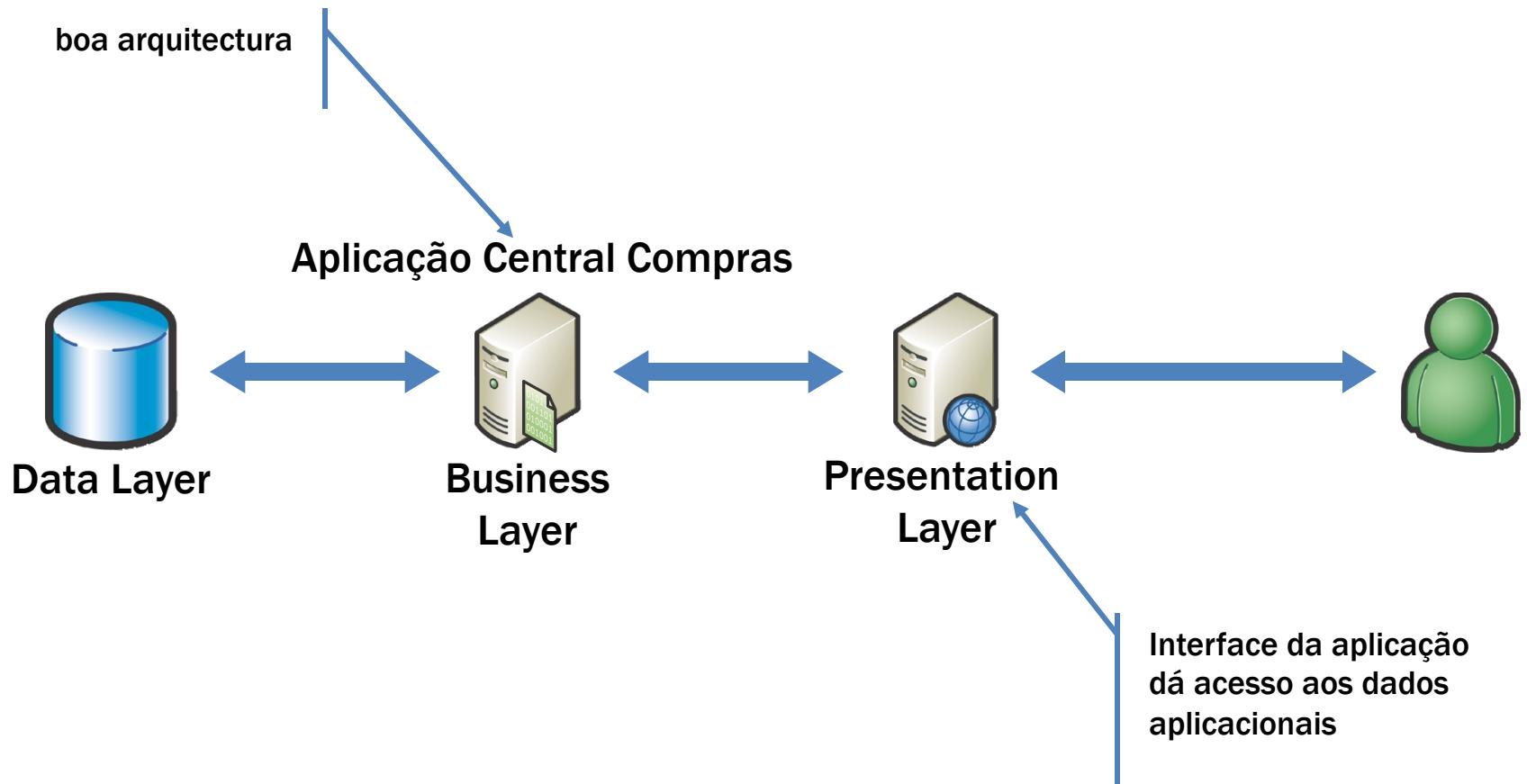
- Web Services é uma tecnologia de chamada remota de objetos
- Permite a infra-estrutura para criação de aplicações distribuídas (web ou não)
- Permitem a criação de pequenos módulos de código reutilizáveis e disponibilizados para construção de aplicações “LEGO”
- Utiliza protocolos Web como meio de transporte e comunicação
- Alto grau de abstração em relação a linguagens de programação e plataformas de hardware / software



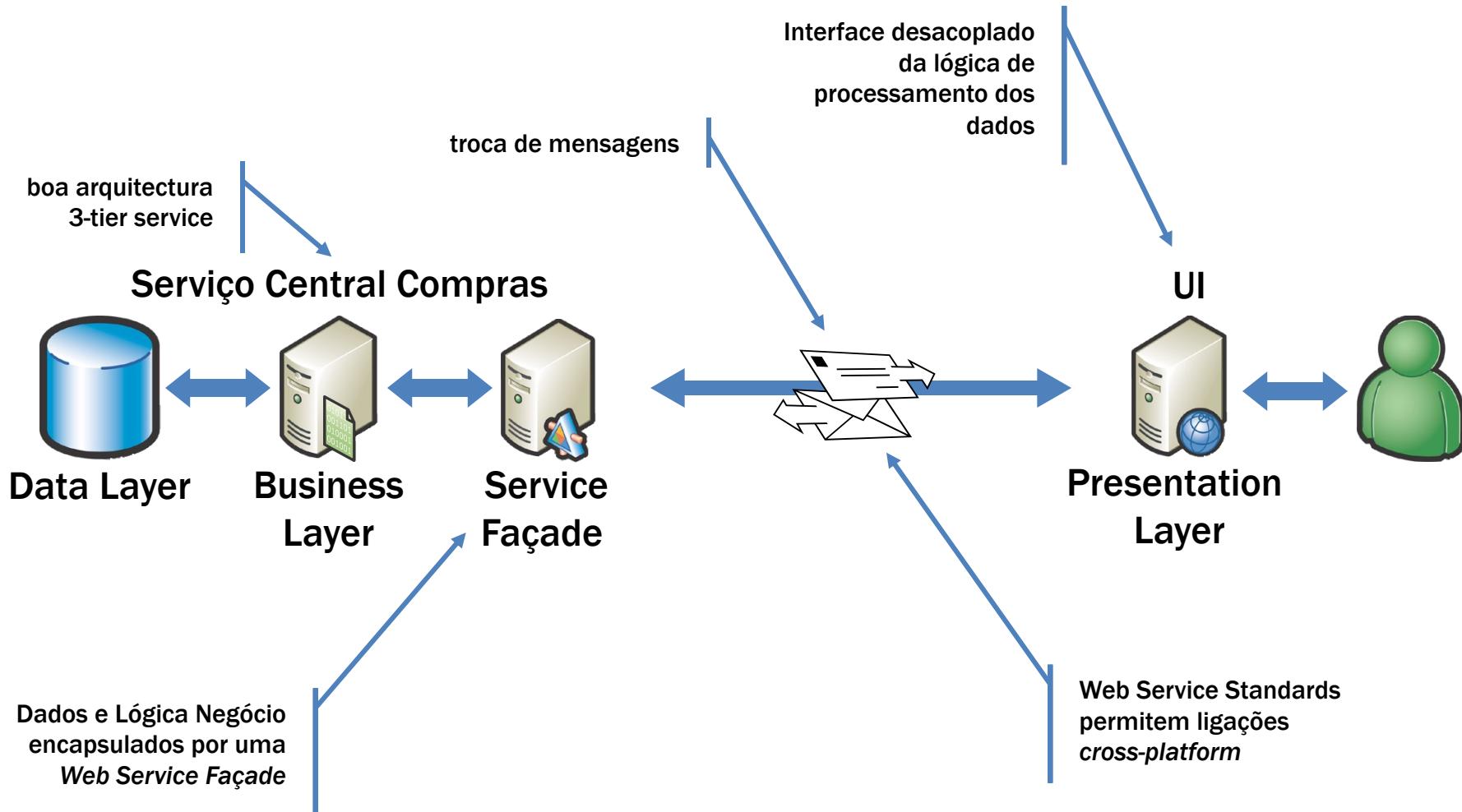
# Service Oriented Architecture (SOA)

- SOA é uma arquitetura que representa funcionalidades do software como serviços
- Já existiam tecnologias para SOA
  - Ex.: CORBA, RMI, etc...
- Interoperabilidade é muito importante
  - Padronização
  - Fraco acoplamento

# Exemplo: Aplicação 3 camadas



# Service Oriented Architecture



## Automação de Processos de Negócio

Serviços Contabilidade

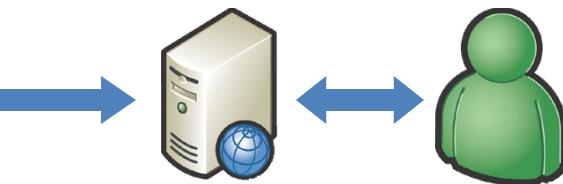


Integração com diferentes plataformas

Serviço Central Compras



UI Processo Compras



Business  
Process  
Automation

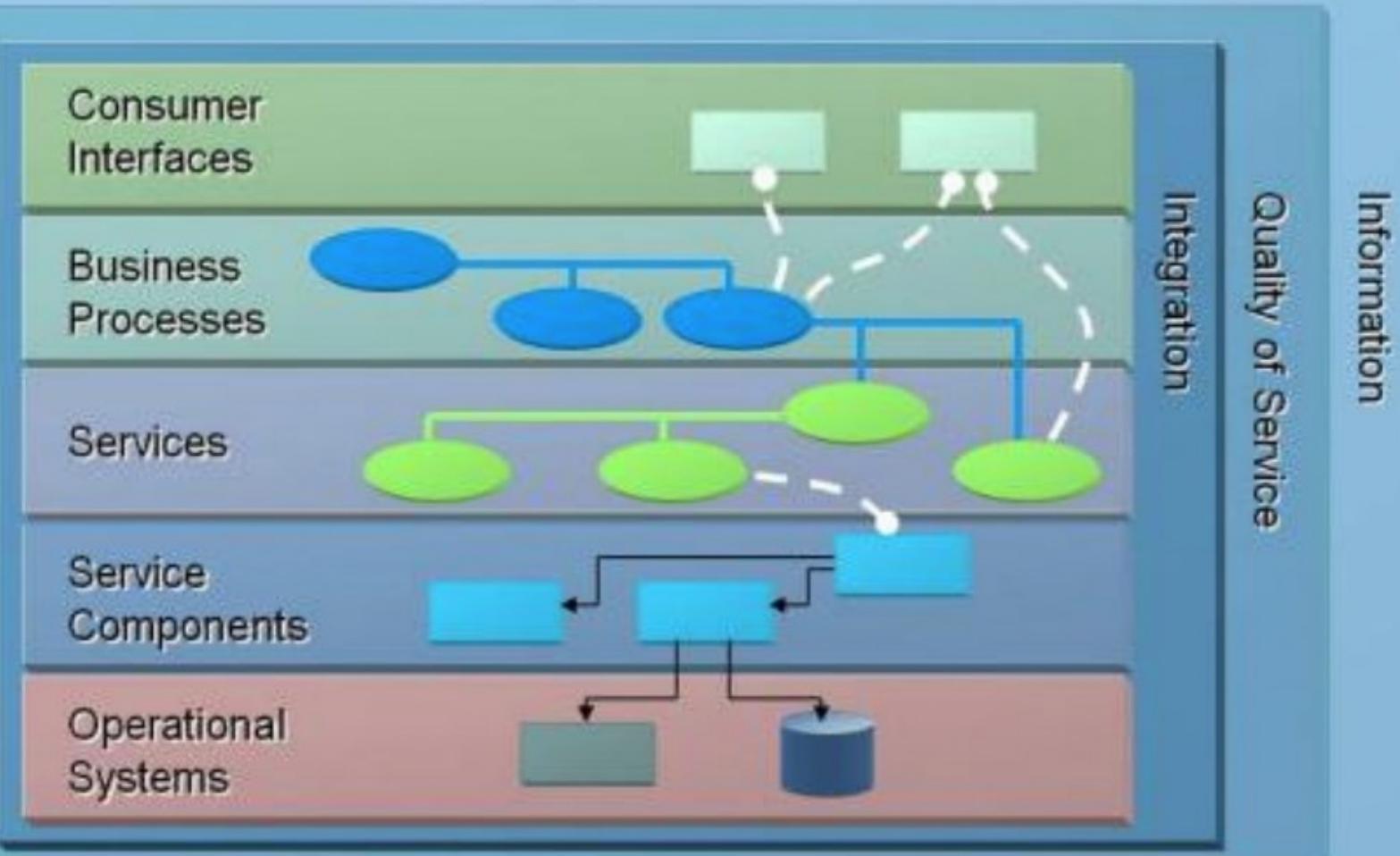
Serviço RH



Múltiplos serviços  
combinados num único  
processo de negócio

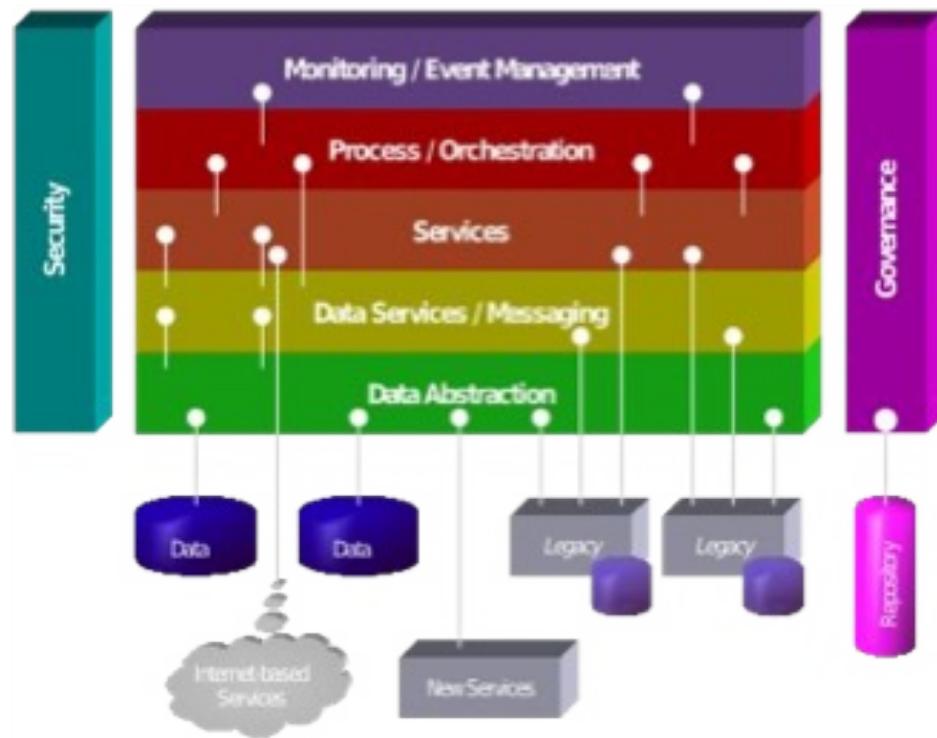
# Service Oriented Architecture (SOA)

- Estilo de arquitetura de software cujo princípio fundamental está baseado em funcionalidades implementadas por aplicações e disponibilizadas na forma de serviços.
- Camadas
  - Interface de Usuário
  - Processos de Negócios
  - Serviços
  - Componentes de Serviços
  - Sistemas Operacionais

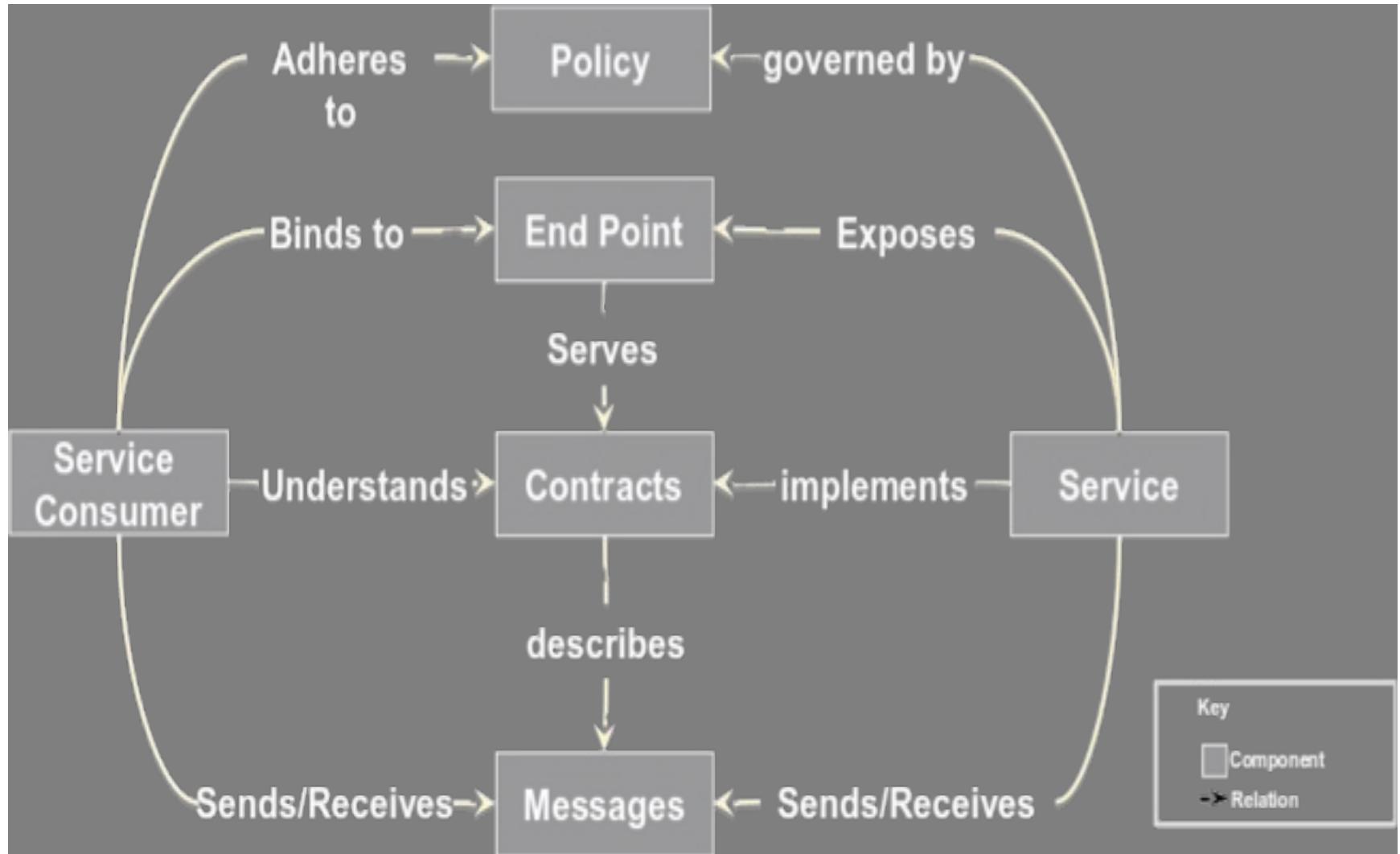


# Hierarquia dos serviços

- Os serviços tendem a formar naturalmente uma arquitetura em camadas.
  - A camada de abstração de dados recupera e escreve para bancos de dados subjacentes.
  - Os serviços de dados transformam esses dados e fornecem filas de mensagens.
  - Os serviços oferecem tarefas de negócios de baixo nível, podem ser combinados para executar tarefas de "alto nível" pela camada de processo/orquestração.
  - Os níveis superiores realizam integração e monitoramento de todo o sistema.
  - Os serviços de segurança e governança funcionam em todas as camadas.



# Interações com o serviço



# Padrões de contrato de serviços

- Serviços dentro do mesmo sistema deve estar em conformidade com o mesmo padrões de contrato.
  - Serviços compartilham esquema e contrato, não a classe.
  - A compatibilidade de serviço é baseada na política
- Um contrato de serviço é uma promessa do propósito e funcionalidade de um serviço.
  - Sua interface pública.
  - A natureza e a quantidade de conteúdo que ele publicará.
  - Como os serviços expressam a sua funcionalidade?
  - Como os tipos/modelos de dados são definidos?
  - Como as políticas são afirmadas e anexadas?

# Princípios dos serviços

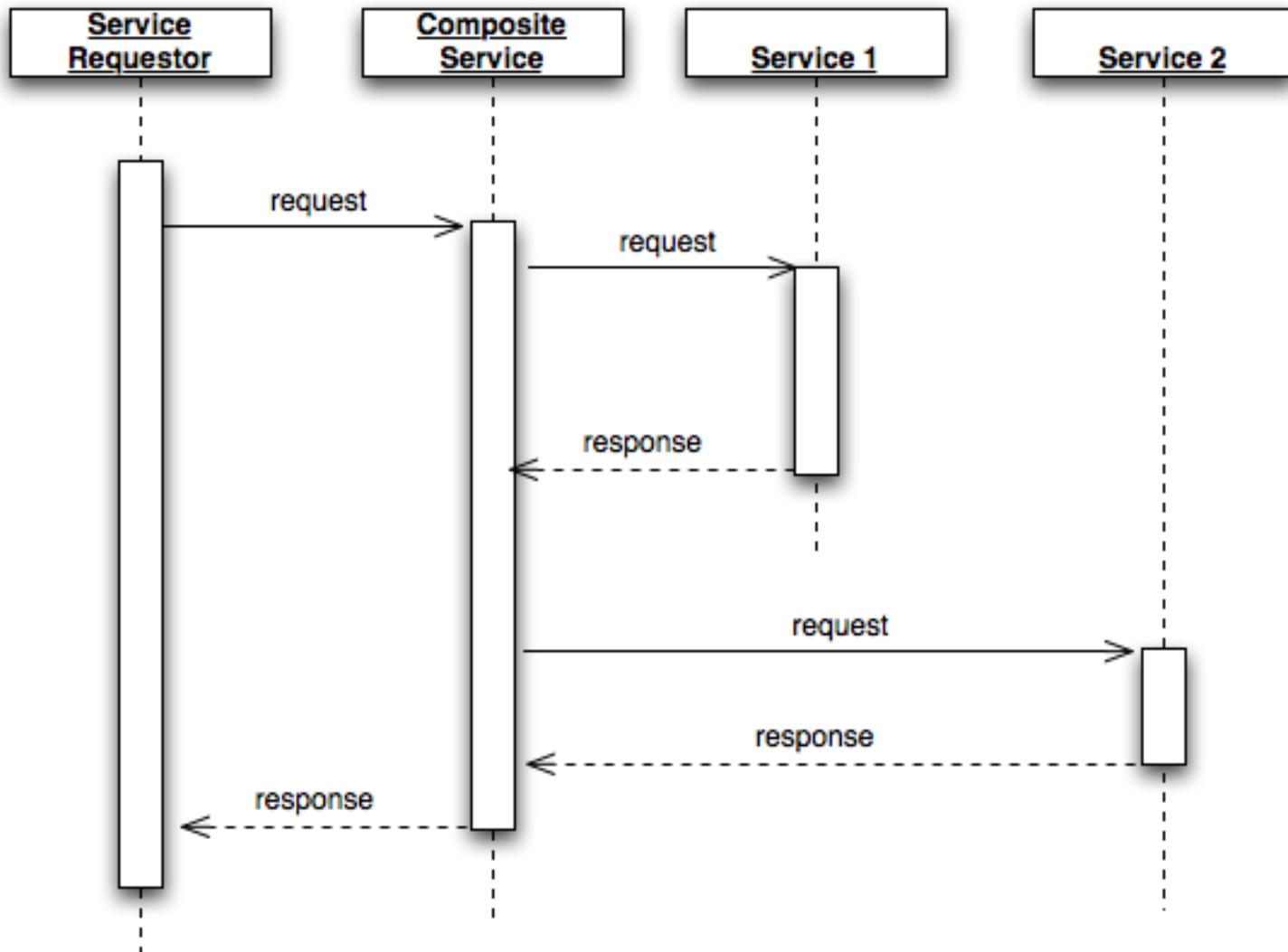
- Para criar serviços úteis e reutilizáveis e, por extensão, a arquitetura orientada serviços (SOA),
  - existem melhores práticas, diretrizes e princípios que foram desenvolvidos que delineiam as propriedades desejadas que os serviços deveriam ter.
- Essas propriedades desejadas para serviços são...

# Modular and Loosely Coupled

- Os serviços devem ser módulares e com acoplamento fraco.
- Isso permite que os serviços sejam reutilizáveis e combináveis
  - ou seja, os serviços podem ser misturados e combinados se forem modulares.
- Na programação orientada a objetos, o acoplamento fraco é alcançado expondo apenas os elementos relevantes de uma classe ou componente ao seu cliente.
- No SOA, as solicitações são feitas passando a comunicação para o serviço de forma a se alinhar com sua interface.
- O serviço realiza as operações necessárias e, em seguida, repassa uma comunicação contendo o resultado do serviço ou uma confirmação de que a solicitação foi atendida.

# Composable

- Os serviços devem ser usados em combinação, a fim de criar aplicativos utilizáveis ou outros serviços.
- Para conseguir essa propriedade, os serviços devem ser modulares.
- Assim como os objetos podem ser combinados na programação orientada a objetos para fornecer o comportamento desejado
- os serviços devem ser capazes de se combinarem para fornecer um objetivo final desejado em SOA.
- Existe o Business Process Execution Language (BPEL) para o WSDL



# Platform and Language Independent

- Os serviços devem ser independentes da plataforma e independentes da linguagem.
- Por exemplo, um serviço codificado em Java pode ser usado por um solicitante de serviço codificado em Ruby.
- Para alcançar a independência de plataforma e idiomas, devem ser seguidos padrões de comunicação e protocolos.
  - Por exemplo, serviços na Internet são frequentemente solicitados com um arquivo JSON ou solicitação HTTP.

# Self-Describing

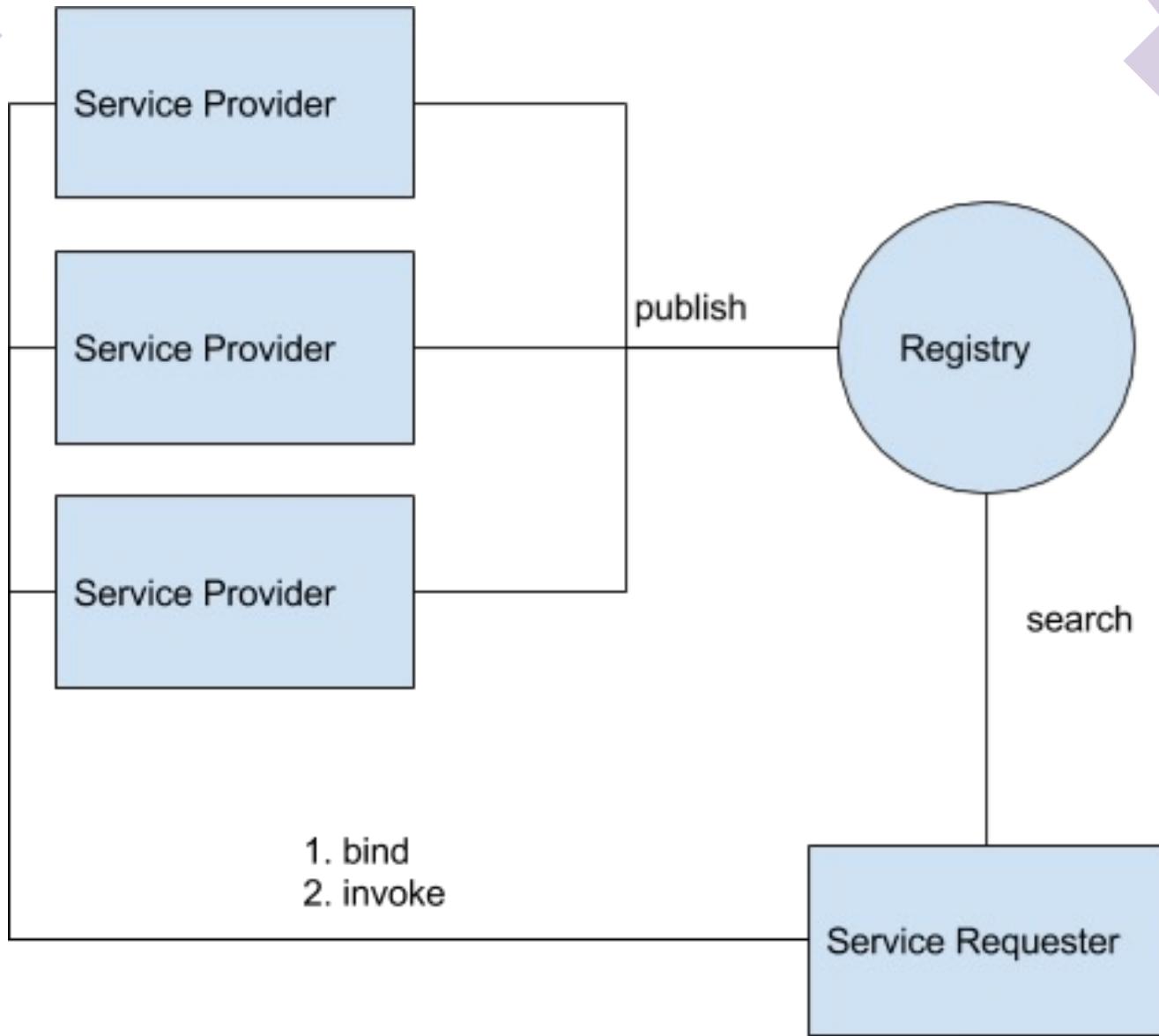
- Um serviço deve descrever como interagir com ele.
- Em outras palavras, um serviço deve descrever suas próprias interfaces.
- Isso inclui
  - Os parâmetros de entrada do serviço
  - a saída que serviço retorna.
- Existem normas formais para descrever serviços, incluindo a linguagem de descrição do serviço web (WSDL).

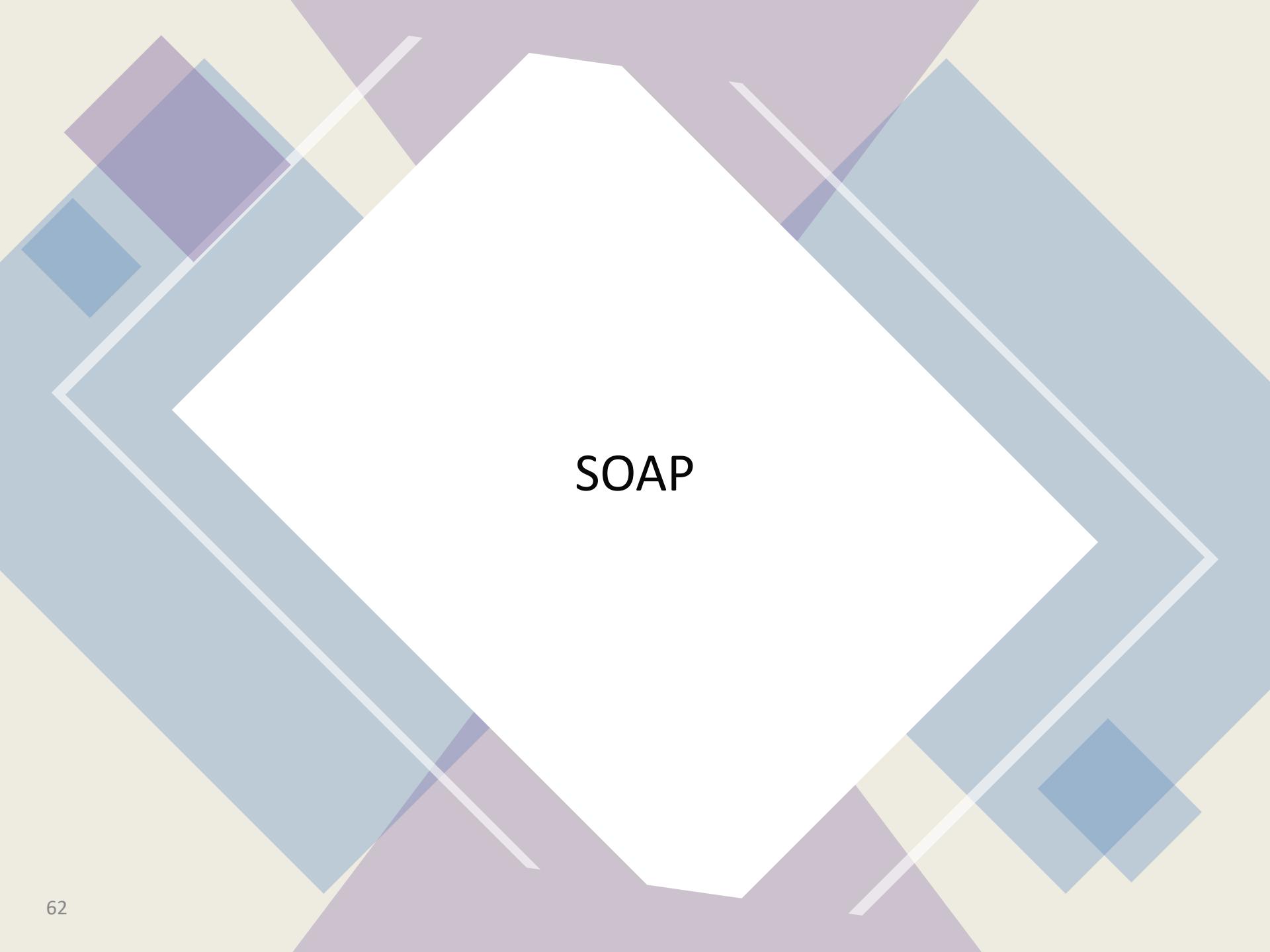
# Self-Advertising

- Os serviços devem se tornar conhecidos aos clientes.
- As organizações internas podem criar catálogos de serviços
- Já aplicações distribuídas usando webservices têm certos padrões
  - Um deles é o Universal Description, Discovery e Integration (UDDI) para conectar provedores de serviços com potenciais solicitantes de serviços.

# Universal Description, Discovery, and Integration (UDDI)

- O UDDI permite que os provedores de serviços publiquem seus serviços em um registro UDDI.
- Uma vez publicados, os potenciais solicitantes de serviços podem pesquisar o registro e descobrir o serviço de que precisam.
- Isso é feito pesquisando elementos da descrição WSDL ou outras descrições dos serviços ou de seus provedores.
- Reunir prestadores de serviços e solicitantes de serviços é uma parte essencial para criar um ecossistema de serviços eficaz.





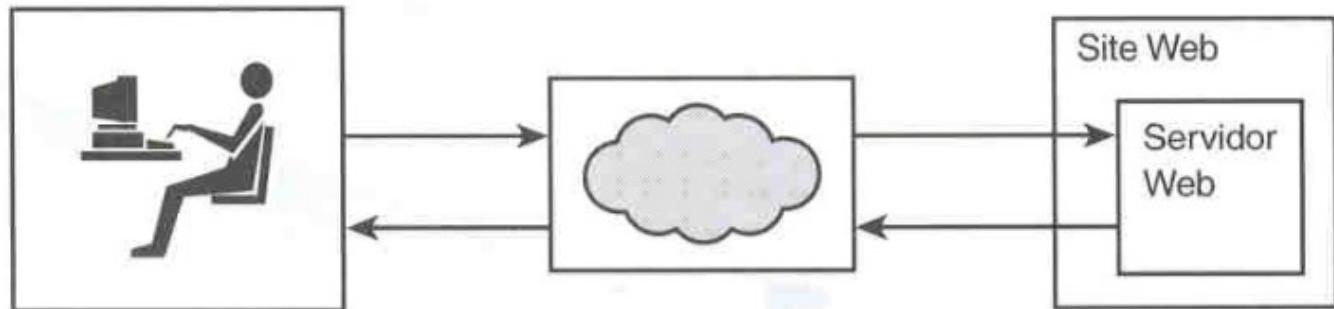
SOAP

# Ser humano

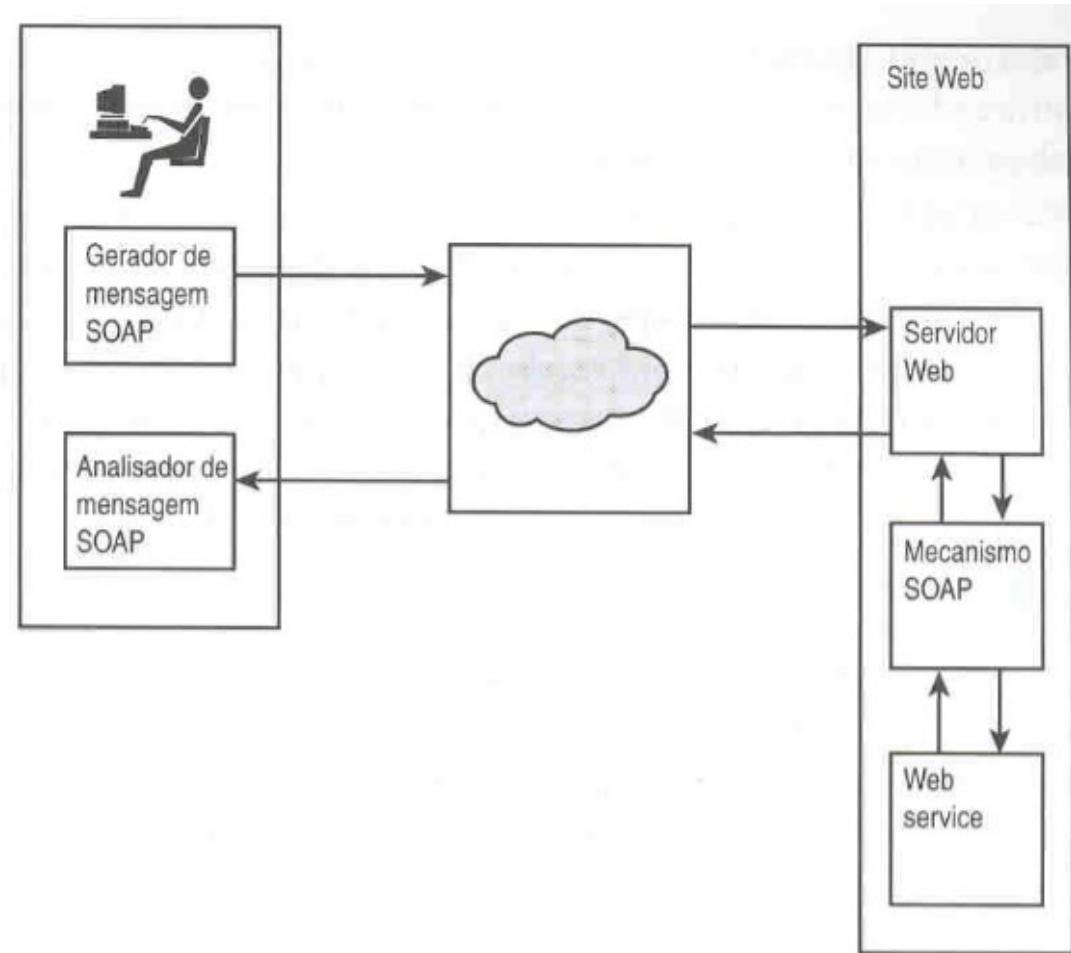
---

**FIGURA 1.1**

Um navegador interage com um servidor Web para fazer requisições.



# Web Service também é acessível via URL!



# SOAP

---

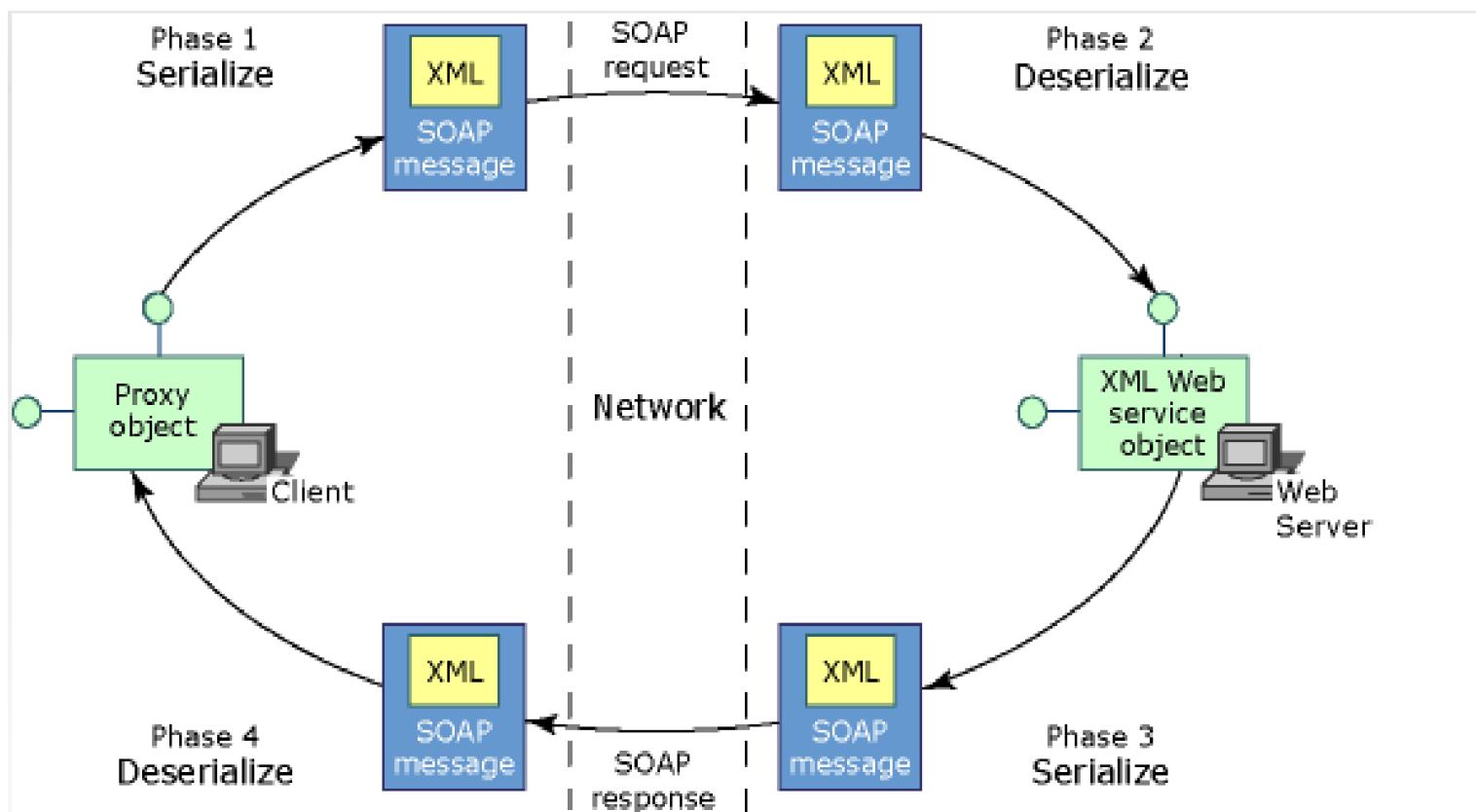
- O SOAP (Simple Object Access Protocol) baseia-se numa invocação remota de um método e para tal necessita especificar o endereço do componente, o nome do método e os argumentos para esse método.
- Estes dados são formatados em XML com determinadas regras e enviados normalmente por HTTP para esse componente.
- Não define ou impõe qualquer semântica, quer seja o modelo de programação, quer seja a semântica específica da implementação.

# Padrões SOAP

---

- **XML** (eXtensible Markup Language):
  - Metalinguagem;
  - Linguagem para construir linguagens.
- **SOAP** (Simple Object Access Protocol):
  - Gramática de envio e resposta;
  - Descrição do formato da mensagem;
- Duas partes:
  - Cabeçalho e Payload.

# Funcionamento do SOAP



# Estrutura da Mensagem SOAP

- Envelope
  - Define o inicio e o final da mensagem
  - é obrigatório
- Header (Cabeçalho)
  - Traz atributos opcionais da mensagem utilizada para o seu processamento
  - é opcional
- Body (Corpo)
  - Possui o conteúdo da mensagem em formato XML
  - é obrigatório



# Exemplo de SOAP requisição

**POST /InStock HTTP/1.1**

**Host: www.example.org**

**Content-Type: application/soap+xml; charset=utf-8**

**Content-Length: nnn**

```
<?xml version="1.0"?>
<soap:Envelope
    xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
    soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

    <soap:Header>
        <t:Transaction xmlns:t="URI" soap:mustUnderstand="1" > 5
        </t:Transaction>
    </soap:Header>

    <soap:Body xmlns:m="http://www.example.org/stock">
        <m:GetStockPrice>
            <m:StockName>IBM</m:StockName>
        </m:GetStockPrice>
    </soap:Body>

</soap:Envelope>
```

# Exemplo de SOAP resposta

**POST /InStock HTTP/1.1**

**Host: www.example.org**

**Content-Type: application/soap+xml; charset=utf-8**

**Content-Length: nnn**

```
<?xml version="1.0"?>
<soap:Envelope
    xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
    soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

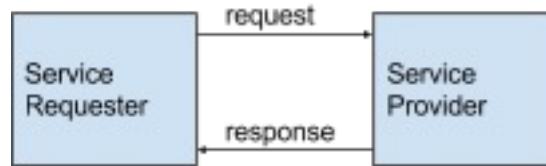
    <soap:Header>
        <t:Transaction xmlns:t="URI" soap:mustUnderstand="1" > 5
        </t:Transaction>
    </soap:Header>

    <soap:Body xmlns:m="http://www.example.org/stock">
        <m:GetStockPrice>
            <m:StockName>IBM</m:StockName>
        </m:GetStockPrice>
    </soap:Body>

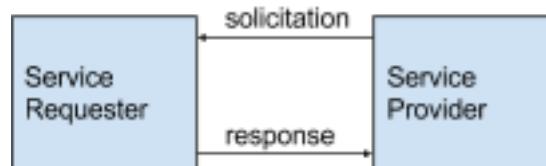
</soap:Envelope>
```

# Padrões SOAP de mensagens

- Request-response:



- Solicit-response



- One-way



- Notification

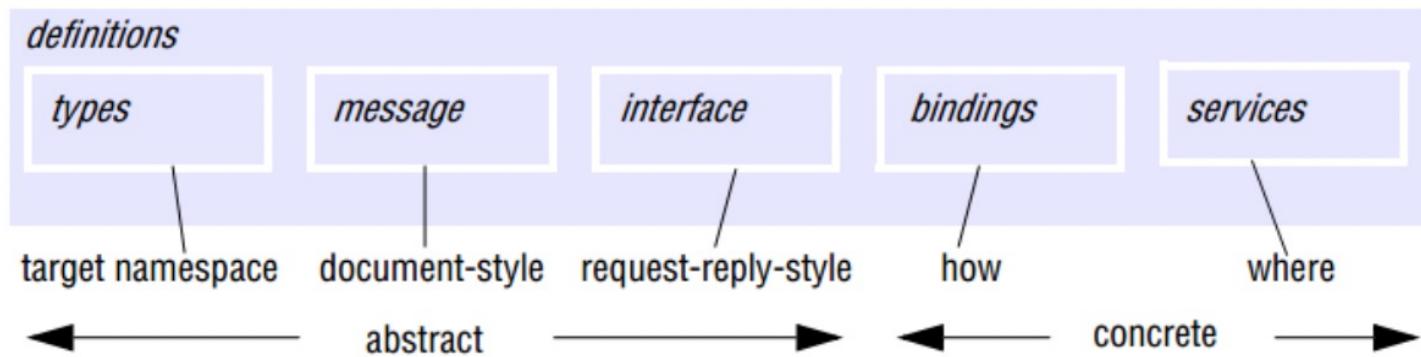


# WSDL

- O contrato do serviço deve reger todos serviços que você oferece em um "inventário".
  - Consiste em uma expressão funcional padronização - definindo a interface, entrada e saída (WSDL).
  - Um modelo de dados (esquema XML) - definindo formatos.
  - Documento de política - definindo termos de uso.
- Contratos de serviço garantem que os serviços sejam consistente, confiáveis e governáveis.
  - As normas devem ser aplicadas corretamente. Serviços com contratos evitam ambiguidade.

# WSDL

- Linguagem de descrição de um web service baseada em XML
- Define os seguintes objetos:
  - Tipos de dados suportados pelo serviço
  - Padrão de mensagens de entrada e saída
  - Protocolos de comunicação permitidos pelo Web Service (binding)
  - Serviços e portas de comunicação onde operações são disponibilizadas pelo Web Service
  - Definições sobre schemas e namespaces utilizados no contexto do Web Service

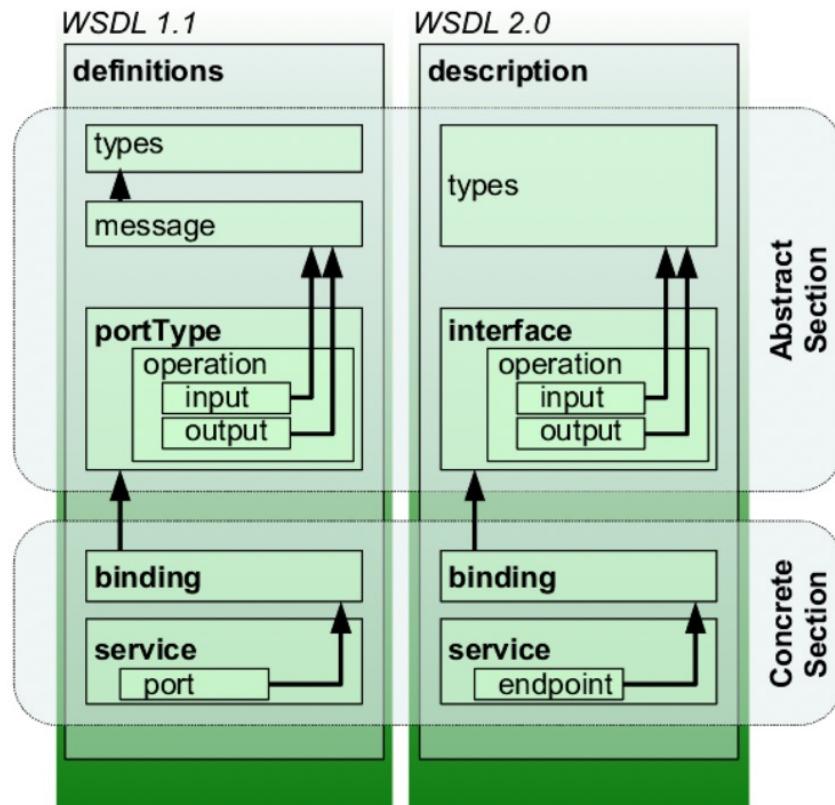


# Exemplo WSDL

```
<message name="getTermRequest">
  <part name="term" type="xs:string"/>
</message>

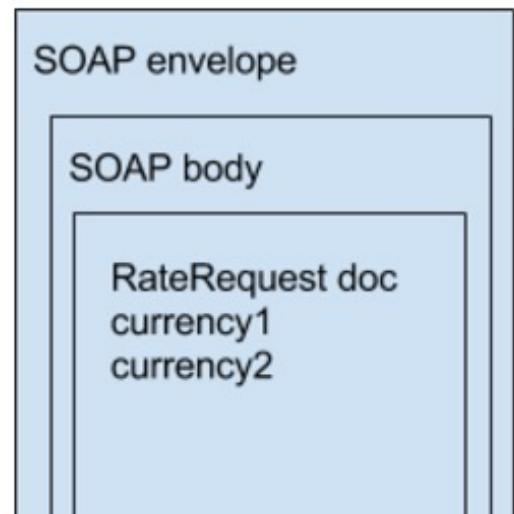
<message name="getTermResponse">
  <part name="value" type="xs:string"/>
</message>

<portType name="glossaryTerms">
  <operation name="getTerm">
    <input message="getTermRequest"/>
    <output message="getTermResponse"/>
  </operation>
</portType>
```



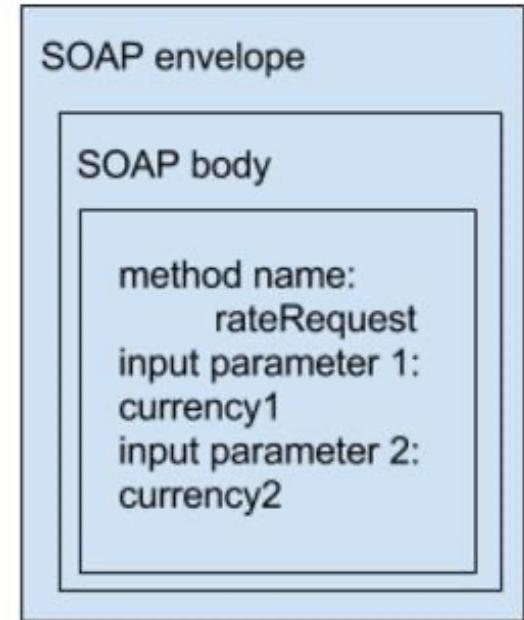
# Document x RPC

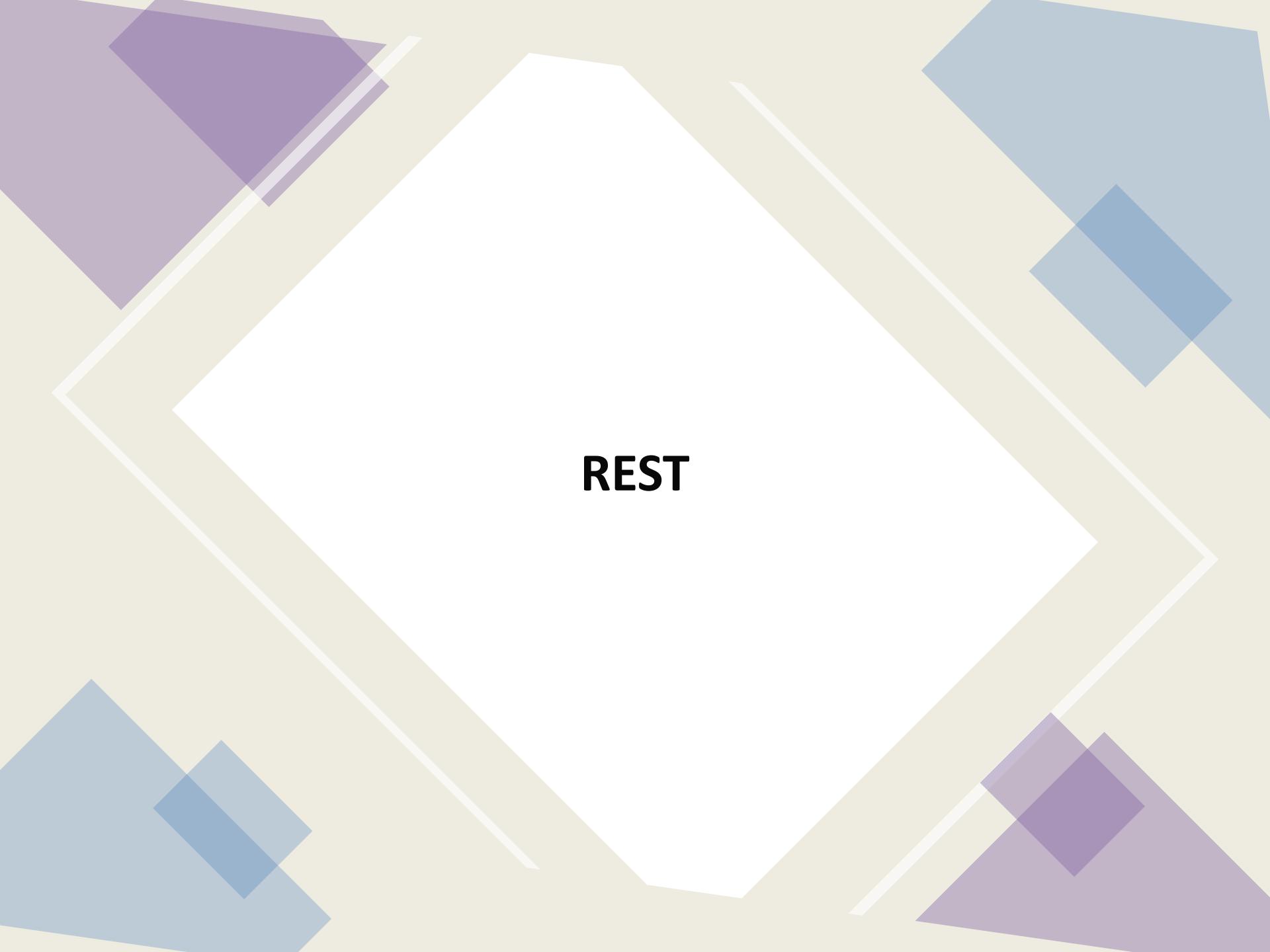
- Existem dois modelos de estilo de comunicação que são usados para traduzir uma ligação WSDL para um corpo de mensagens SOAP.
- São eles: Document & RPC
- A vantagem de usar um modelo de estilo Document é que você pode estruturar o corpo SOAP da maneira que quiser, desde que o conteúdo do corpo da mensagem SOAP seja qualquer instância XML arbitrária.
- O estilo Document também é referido como estilo Orientado a mensagem.
- A desvantagem de usar o estilo Document é que ele traz alguma complexidade na forma de XSD para definir os tipos de dados mais ricos.



# Document x RPC

- No entanto, com um modelo de estilo RPC, a estrutura do corpo de solicitação SOAP deve conter tanto o nome da operação quanto o conjunto de parâmetros de método.
- O modelo de estilo RPC assume uma estrutura específica para a instância XML contida no corpo da mensagem.
- A desvantagem básica de usar o estilo RPC é que ele não suporta tipos de dados mais ricos



The background features a minimalist abstract design composed of several large, overlapping geometric shapes. These shapes include triangles and trapezoids in shades of purple (#9966CC), blue (#6680BD), and beige (#CCCC99). The shapes are arranged in a way that creates a sense of depth and movement, radiating from the center of the frame.

**REST**

# REST - Representational State Transfer

- É um estilo arquitetural para construção de web services escaláveis que define um conjunto de regras.
- Regras REST
  - Cliente e Servidor
  - Sem estado (stateless) → servidor não armazena sessão do cliente
  - Interface Uniforme → permite que qualquer componente que entenda o protocolo de aplicação HTTP use o serviço
  - Permite cache (cacheable)
  - Sistema em camadas

# Regras REST – Interface uniforme

- Identificação de recursos via URI:
  - <http://graph.facebook.com/536510179872296>
- Manipulação de recursos via representações
- Mensagens auto descritivas
- Hipertexto como o mecanismo de estado da aplicação
- Benefícios
  - Simplifica a implementação
  - Prove desacoplamento da arquitetura
  - Permite que cada parte (cliente e servidor) evoluam independentemente

# Regras REST – Interface uniforme

- Identificacao de recursos via URI
- Requisicao
  - <https://api.punkapi.com/v2/beers/1>
- Resposta (JSON)

```
{"id":1,  
"name":"Buzz",  
"tagline":"A Real Bitter Experience.", "first_brewed":"09/2007",  
"description":"A light, crisp and bitter IPA brewed with English and American hops. A  
small batch brewed only once."...}
```

# REST

- ❑ Cada url deve representar um recurso;
- ❑ Ideal para CRUD (Criar, Ler, Atualizar e Excluir);
- ❑ Geralmente, **via método GET/POST**, cada recurso deve ser diferenciável;
- ❑ Uso de padrão de chamadas: GET, PUT, POST e DELETE
- ❑ Uso de cache
- ❑ **NÃO TEM WSDL!**
- ❑ Retorno livre:
  - XML;
  - JSON;
  - Etc.
- <http://www.recipepuppy.com/api/?i=onions,garlic&q=omelet&p=3>

# Regras REST – Interface uniforme

## Hipertexto como o mecanismo de estado da aplicação (HATEOAS)

```
GET /account/12345 HTTP/1.1
```

Resposta p/ saldo de 100,00

```
HTTP/1.1 200 OK
<?xml version="1.0"?>
<account>
  <account_number>12345</account_number>
  <balance currency="usd">100.00</balance>
  <link rel="deposit" href="/account/12345/deposit" />
  <link rel="withdraw" href="/account/12345/withdraw" />
  <link rel="transfer" href="/account/12345/transfer" />
  <link rel="close" href="/account/12345/close" />
</account>
```

```
GET /account/12345 HTTP/1.1
```

Resposta p/ saldo de -25,00

```
HTTP/1.1 200 OK
<?xml version="1.0"?>
<account>
  <account_number>12345</account_number>
  <balance currency="usd">-25.00</balance>
  <link rel="deposit" href="/account/12345/deposit" />
</account>
```

# Regras REST – Permite cache

- As respostas podem ser armazenadas em cache (No cliente ou em Proxies reversos)
- As respostas devem definir se permitem ou não o cache
- Benefícios
  - Aumenta eficiência (Menos requisições)
  - Aumenta performance (Menor latência entre cliente e servidor)
  - Permite maior escalabilidade

# Regras REST – Sistema em camadas

Permite uma arquitetura composta em camadas

Cada componente não pode ver além da camada seguinte

Servidores intermediários podem atuar provendo escalabilidade

- Servidores de Cache
- Balanceadores de carga

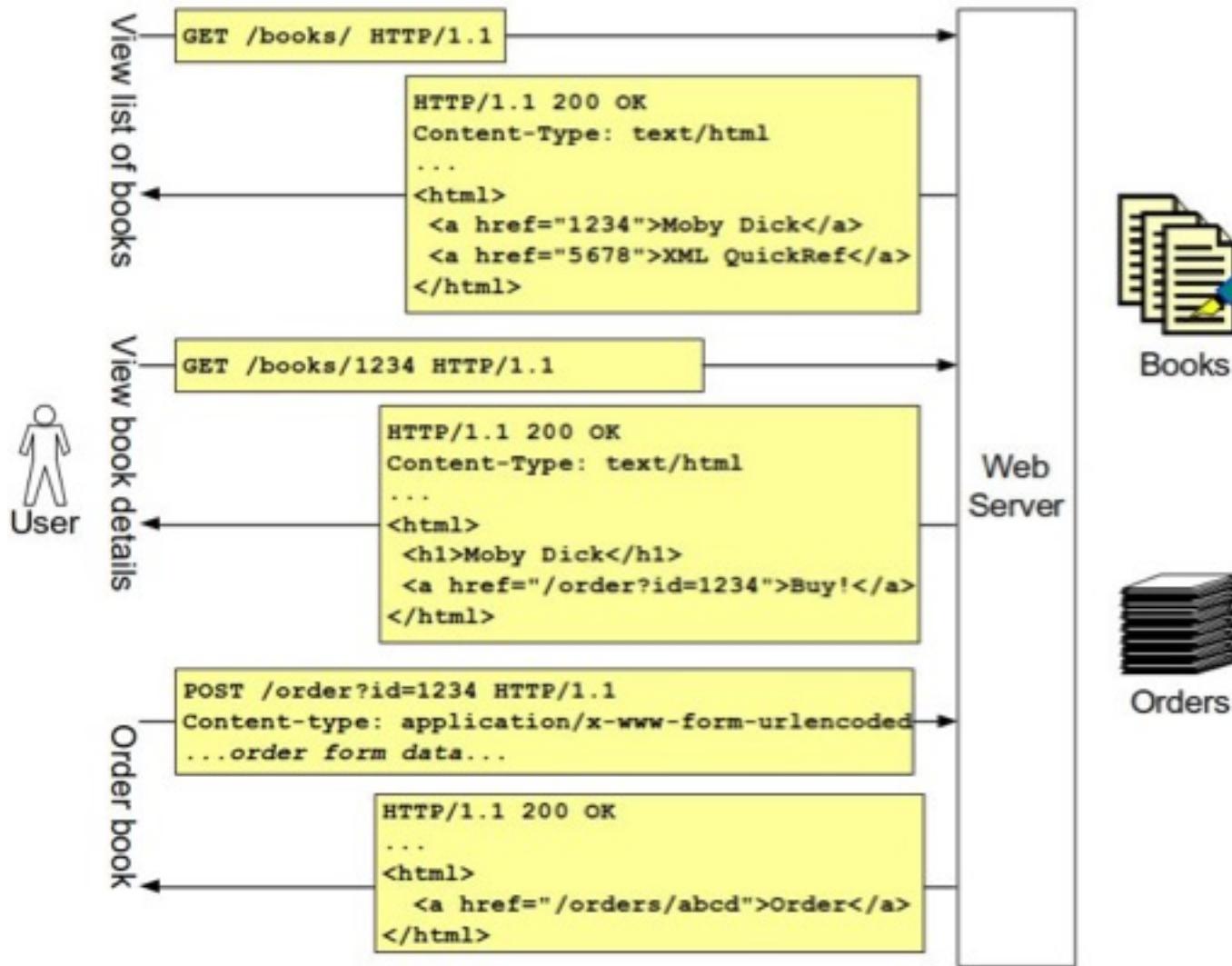
Benefícios

- Simplifica o cliente
- Permite maior escalabilidade (load balance, Cache)



SOAP x Rest

# REST



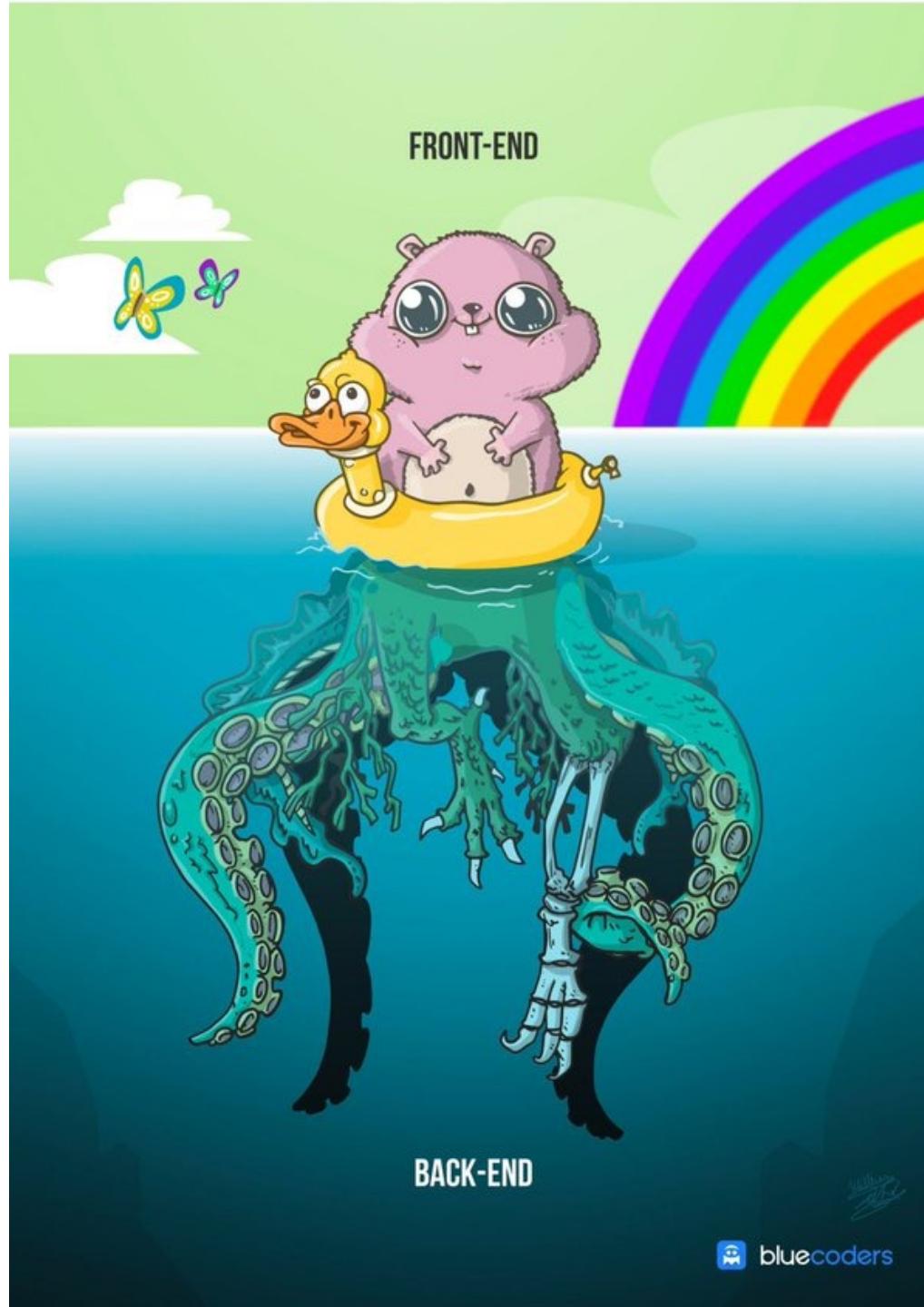
# SOAP



Item	SOAP	REST
Estrutura	Protocolo baseado em XML	Protocolo baseado no estilo arquitetural
Comunicação de Dados	Utiliza um padrão de mensagens baseado em XML	Utiliza XML ou JSON para envio e recebimento dos dados
Comunicação	Invoca serviços por meio de HTTP e RPC (remote procedure call)	Baseado 100% em HTTP e nas URIs
Formato da Mensagem	Resultado codificado no padrão SOAP	Resultado facilmente interpretado por um humano
Compatibilidade com Javascript	Complexa	Simplificada
Desempenho	Médio	Alto

# **BACKEND**

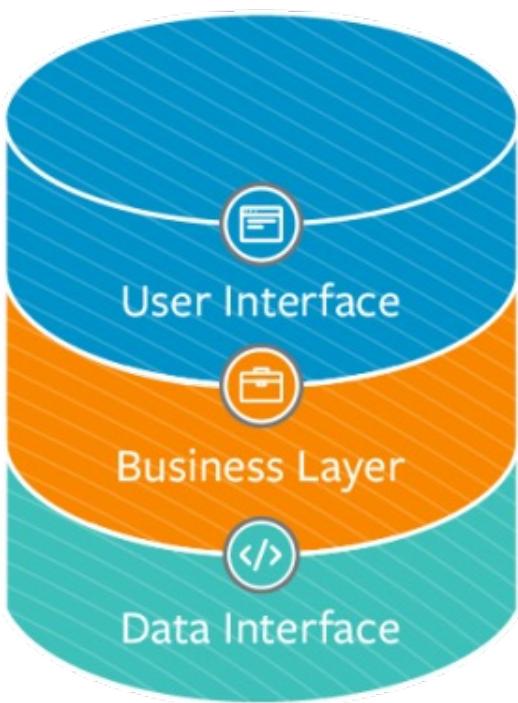
FRONT-END



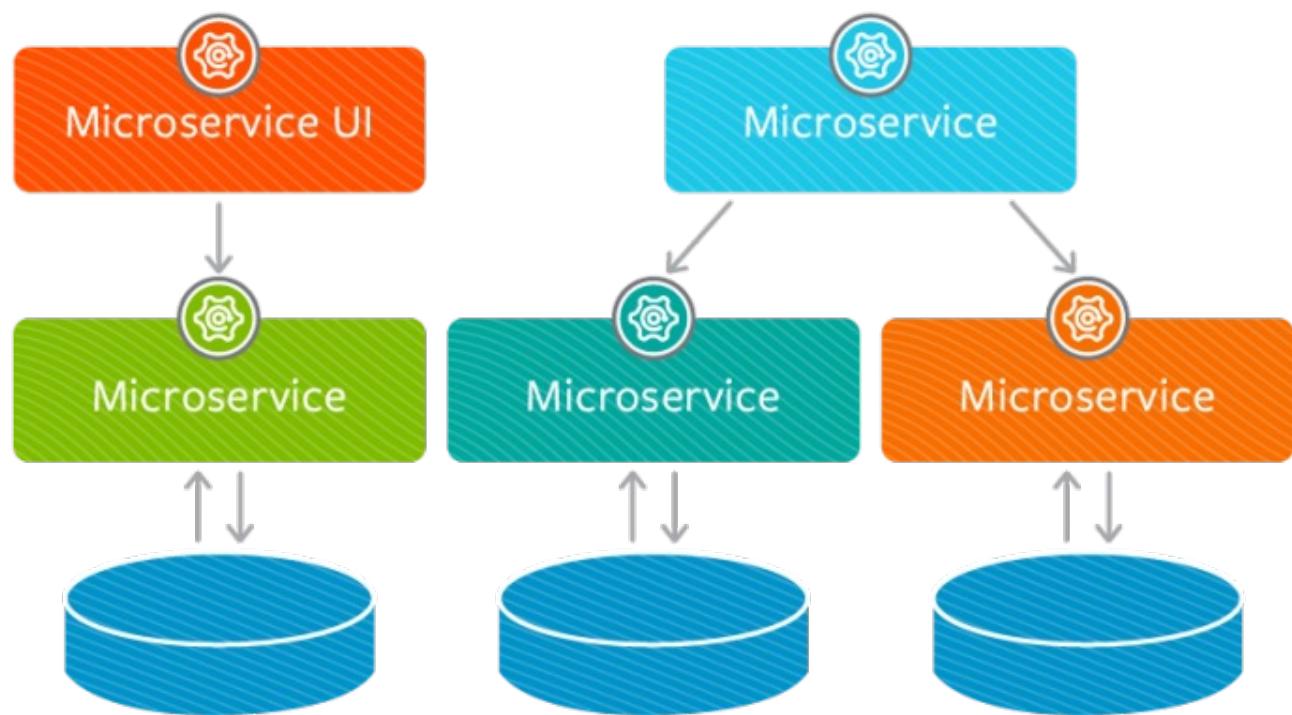
# REST

- ◆ Cada url deve representar um recurso;
- ◆ Ideal para CRUD (Criar, Ler, Atualizar e Excluir);
- ◆ Geralmente, via método GET/POST, cada recurso deve ser diferenciável;
- ◆ Uso de padrão de chamadas: GET, PUT, POST e DELETE
- ◆ Uso de cache
- ◆ NÃO TEM WSDL!
- ◆ Retorno livre: XML; JSON; Etc.

# Monolithic Architecture

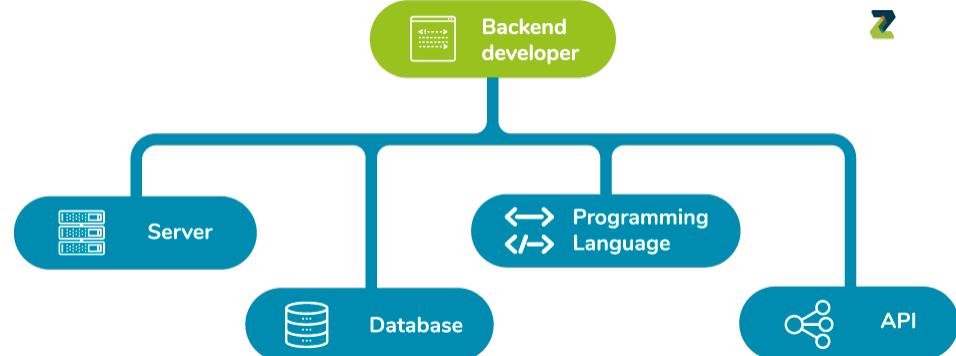


# Microservices Architecture



# O que se precisa de um backend

- Receber e responder requisições
- Interagir com um Database
- Fazer autenticação
- Automatizar processos
- Escalar
- Push Notifications, Analytics, ML...



# Alguns Frameworks Interessantes

- ◆ Swift: Kitura, Smoke ou Vapor
- ◆ Java: Spring (Boot) ou Hibernate
- ◆ Javascript: Node.js, Express.js, Loopback ou Sails.js
- ◆ Python: Django ou Flask
- ◆ PHP: Zend, Cake, Laravel ou Symphony
- ◆ Ruby (on Rails)
- ◆ .NET Core (C# ou F#)
- ◆ ...

# **Swift Server Frameworks**

# Swift Frameworks



# Swift Frameworks



😄 Linguagem Swift  
😊 Comunidade Crescente

Muito Nova 😕

Pouca documentação 😕

Comunidade pequena 😕

# Kitura

- IBM estava por trás do desenvolvimento;
- Baseado em Express.js (também parece bastante com Rails, php,...)
- Integração com o Docker e serviços cloud da IBM





# Video aulas no YouTube

- 4 vídeos:
  - Como instalar e Criando um Hello Server: <https://youtu.be/o8FRCVURBGM>
  - Criando um servidor com Model: <https://youtu.be/NwKcfQDBy0Q>
  - Colocando o model no Banco de Dados: <https://youtu.be/3O2PALj1Psc>
  - Autenticação:<https://www.youtube.com/watch?v=7YmWRy5Bk-E>

# **FLASK**



# Flask

web development,  
one drop at a time

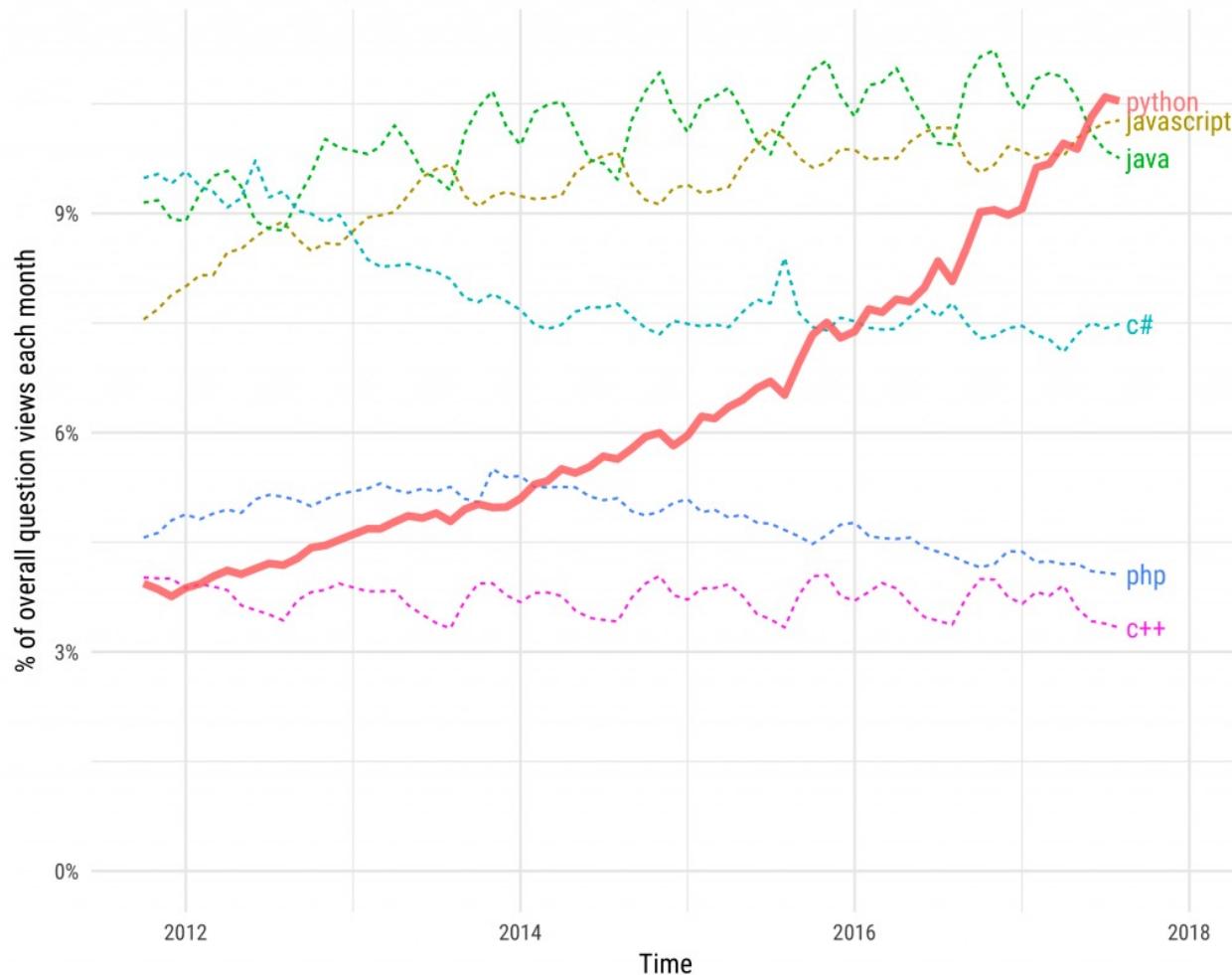
# Flask

- ◆ Microframework python baseado em microservices
- ◆ Facil codar e configurar
- ◆ Bem documentado e minimalista
- ◆ Opensource
- ◆ Padrão RESTful



## Growth of major programming languages

Based on Stack Overflow question views in World Bank high-income countries



# Vídeo aulas no YouTube

◆ 4 vídeos:

- ◆ Como instalar e Criando um Hello Server:  
<https://www.youtube.com/watch?v=SU3u8ALai04&list=PLkdMZhFG653SuzRknHF9qJszFR67XLzPf>
- ◆ Criando um banco de dados:  
<https://www.youtube.com/watch?v=Y2dtdPnxcbo&list=PLkdMZhFG653SuzRknHF9qJszFR67XLzPf>
- ◆ Autenticação:<https://www.youtube.com/watch?v=oJwzVM2eh8U&list=PLkdMZhFG653SuzRknHF9qJszFR67XLzPf>

# **Spring**



spring

# Por que Spring?



<https://spring.io/why-spring>



# Spring Framework Modules



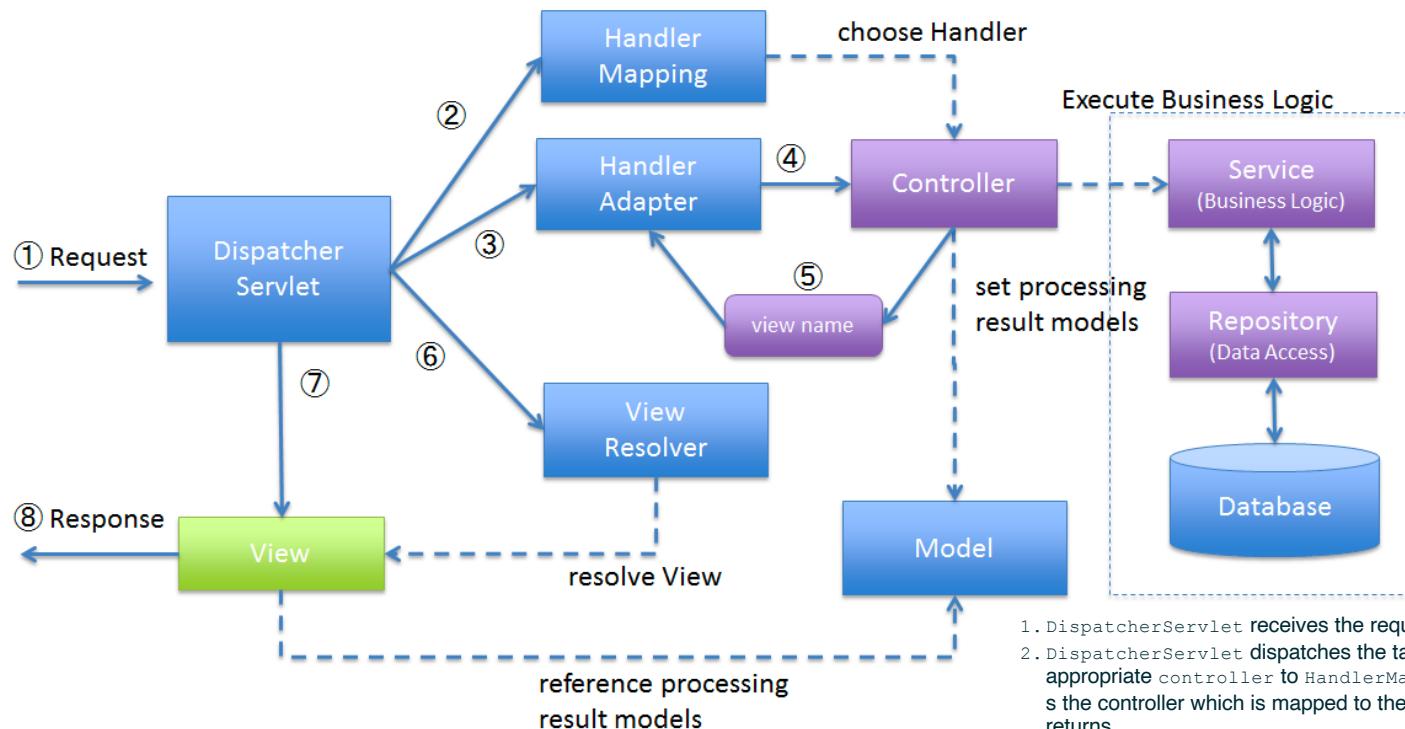
**Dependency Injection (DI)**

**Inversion of Control (IoC)**

**Aspect Oriented Programming (AOP)**

**Model View Controller (MVC)**

**Transaction Management**



1. DispatcherServlet receives the request.
2. DispatcherServlet dispatches the task of selecting an appropriate controller to HandlerMapping. HandlerMapping selects the controller which is mapped to the incoming request URL and returns the (selected Handler) and Controller to DispatcherServlet.
3. DispatcherServlet dispatches the task of executing of business logic of Controller to HandlerAdapter.
4. HandlerAdapter calls the business logic process of Controller.
5. Controller executes the business logic, sets the processing result in Model and returns the logical name of view to HandlerAdapter.
6. DispatcherServlet dispatches the task of resolving the View corresponding to the View name to ViewResolver. ViewResolver returns the View mapped to View name.
7. DispatcherServlet dispatches the rendering process to returned View.
8. View renders Model data and returns the response.



# Softwares/Frameworks utilizados durante o curso (pode ser modificado)

---

- Postman
  - <https://www.postman.com/>
- IntelliJ: Java/Spring Boot
  - <https://www.jetbrains.com/idea/>

# Exercício

- Procurar na web serviços de Webservice;
  - <https://github.com/public-apis/public-apis>
- Consumir um webservice em um App (em iOS ou Android ou Hybrid)
  - Recomendado: Alamofire (iOS) e Retrofit(Android)
- Publicar link do github.