

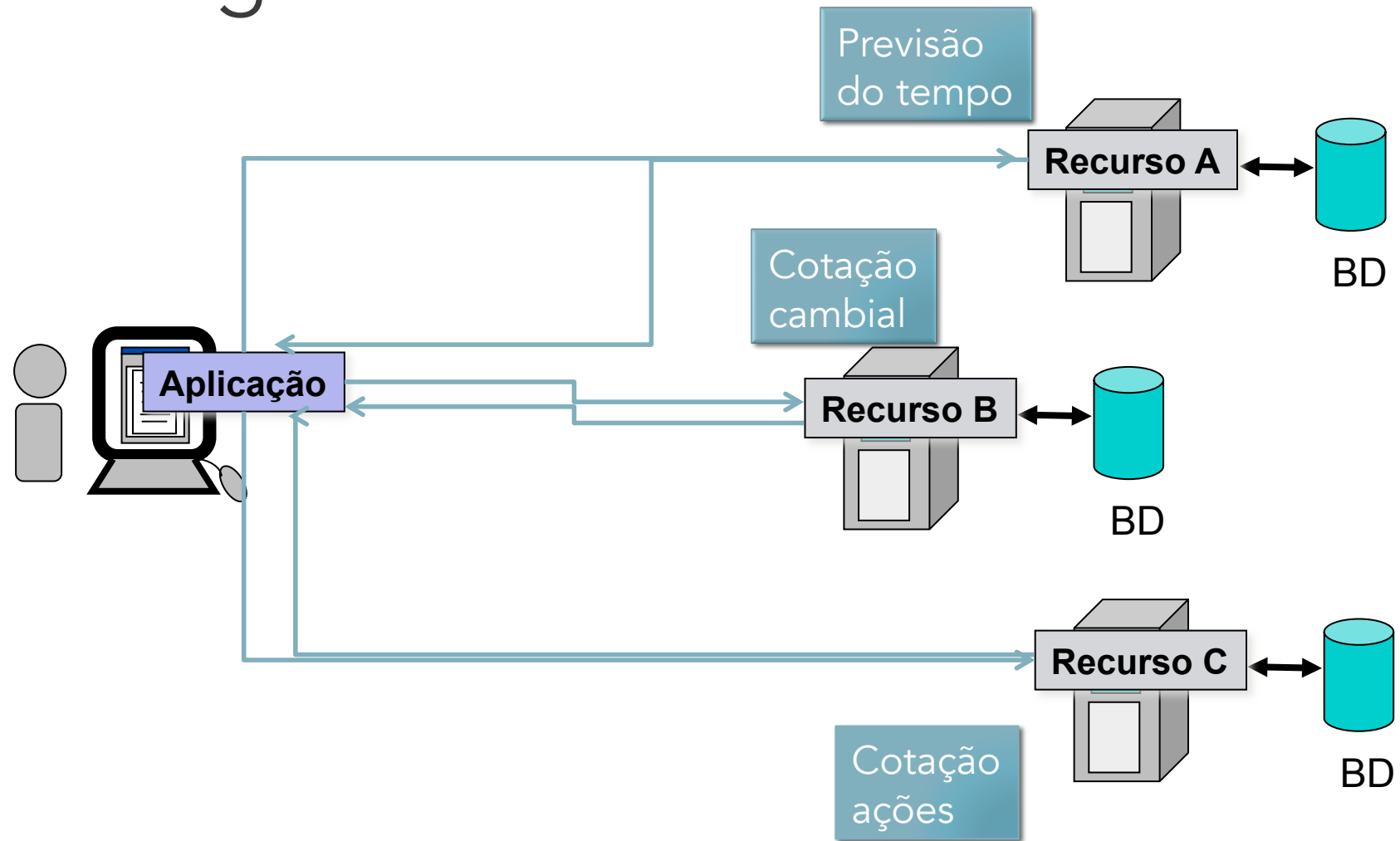
DESENVOLVIMENTO WEB 2

Jair C Leite

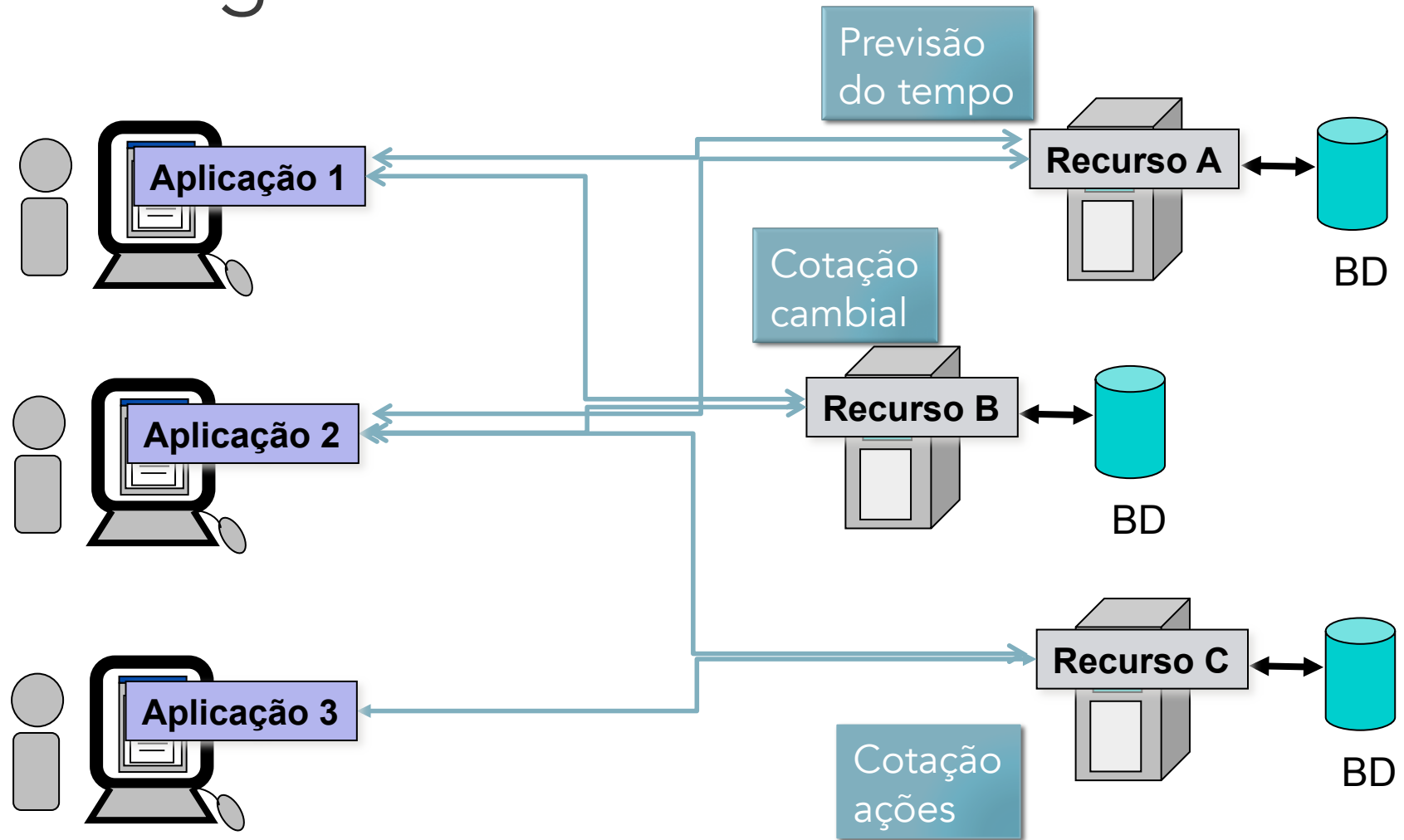


WEB SERVICES

Visão geral



Visão geral



Desafios

- Como possibilitar que consumidores e provedores de recursos computacionais
 - Consigam **conversar** entre si?
 - Sejam desenvolvidos em diferentes **linguagens de programação**?
 - Rodem em diferentes **sistemas operacionais**?
 - **Descubram** onde estão os recursos?

Web Services – conceito

- Abordagem independente de plataforma para oferecer serviços computacionais na Internet utilizando tecnologias Web
- Funciona como interface entre consumidores e provedores

Web Services

Características

- Independente de plataforma
 - Sistema operacional
 - Linguagem de programação
 - Plataforma de hardware



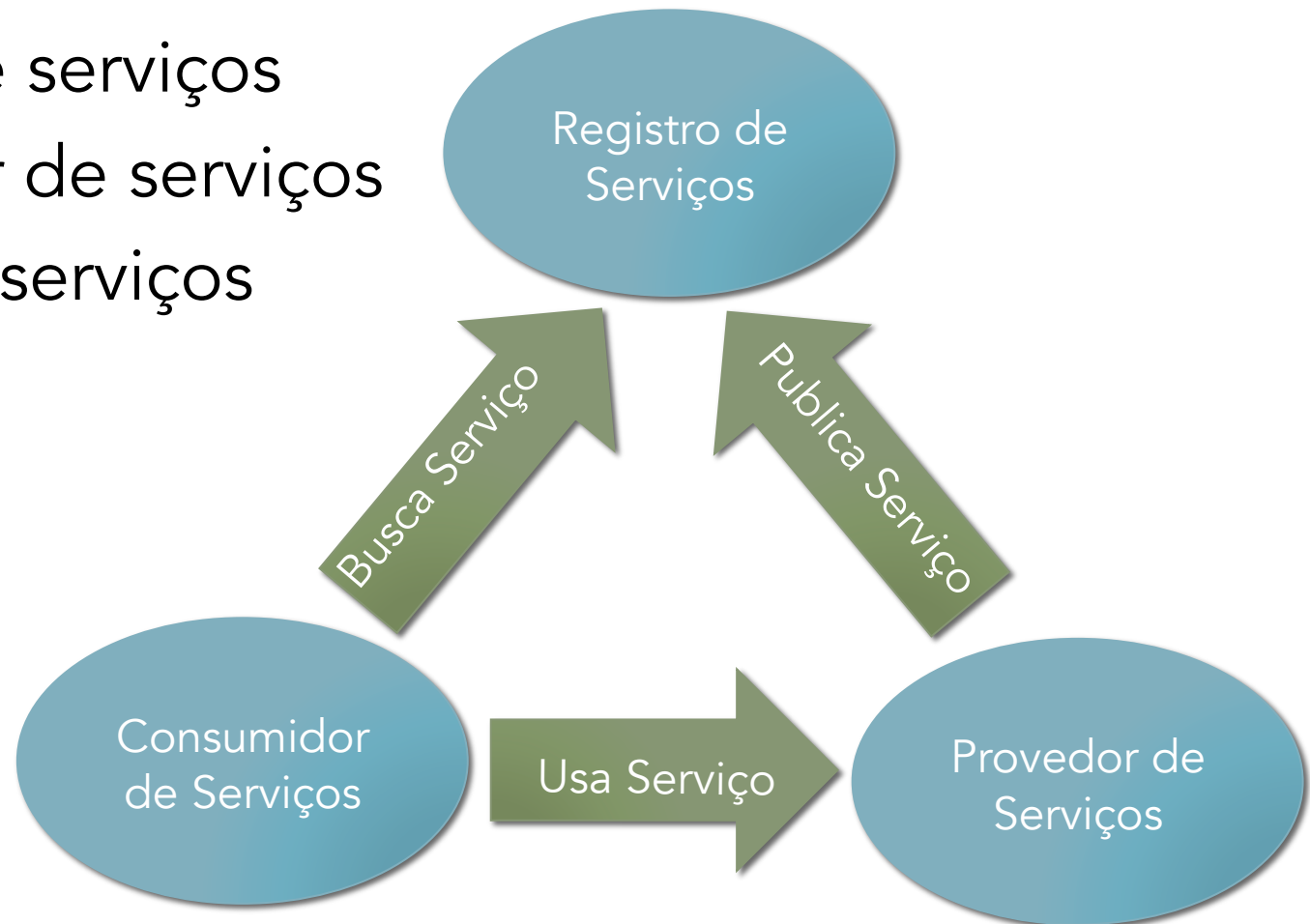
Web Services – rede de serviços

- Web services são aplicações
 - Autocontidas, modulares, distribuídas e dinâmicas
 - Descritas, publicadas, localizadas e acessadas em rede
 - para criar produtos, processos e rede de suprimentos



Papeis

- Provedor de serviços
- Consumidor de serviços
- Registro de serviços



Tecnologias Web Services

- XML + HTTP de duas formas
- SOA
 - SOAP
 - WSDL
 - UDDI
- REST
 - HTTP
 - XML, JSON, HTML

Vantagens

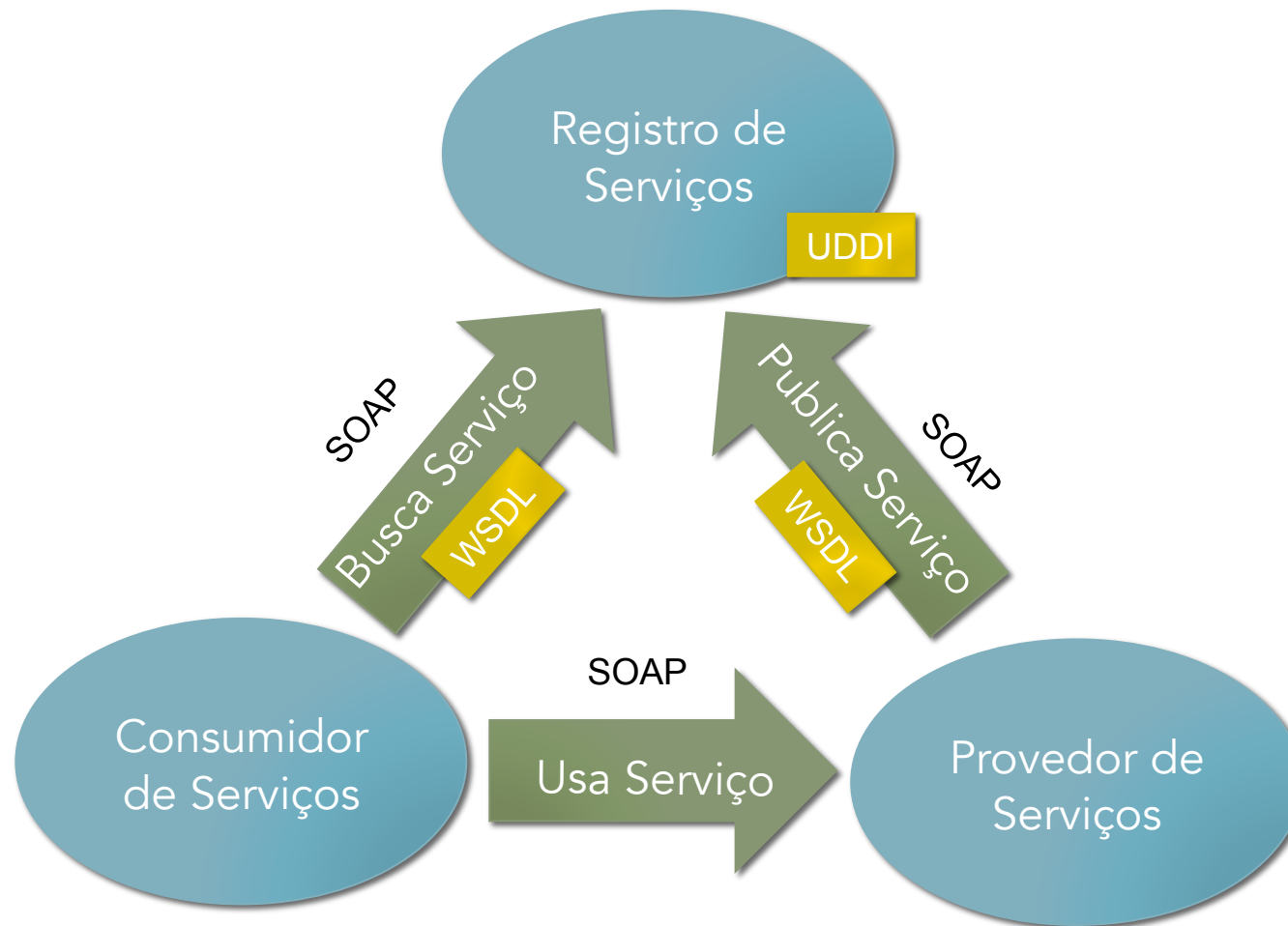
- Exposição de serviços na rede
- Interoperabilidade e integração
- Baixo acoplamento
- Protocolos padronizados
- Comunicação de baixo custo

SOA

Padrões SOA

- WSDL
 - Web Services Description Language
 - utiliza XML para descrever Web services.
 - recomendação do W3C
- SOAP
 - Simple Object Access Protocol
 - protocolo baseado em XML para acessar Web Services.
 - recomendação do W3C
- UDDI
 - Universal Description, Discovery, and Integration.
 - especificação para registro distribuído de serviços.
 - framework aberto e independente de plataforma.
 - pode comunicar via SOAP, CORBA, e Java RMI Protocol.
 - usa WSDL para descrever interfaces para web services.

Papeis e tecnologias



Vantagens

- Interoperabilidade
- Integração
- Reutilização
- Seguranças aos dados
- Redução de custos e tempo

REST

REST API

- Representational State Transfer
- Estilo arquitetural para aplicações em rede
- Utiliza protocolo cliente-servidor stateless: tipicamente HTTP
- Trata objetos no servidor como recursos que podem ser criados ou destruídos
- Podem utilizar com qualquer linguagem de programação

Desenvolvido por Roy Fielding
em seu doutorado

Os 6 Princípios RESTful

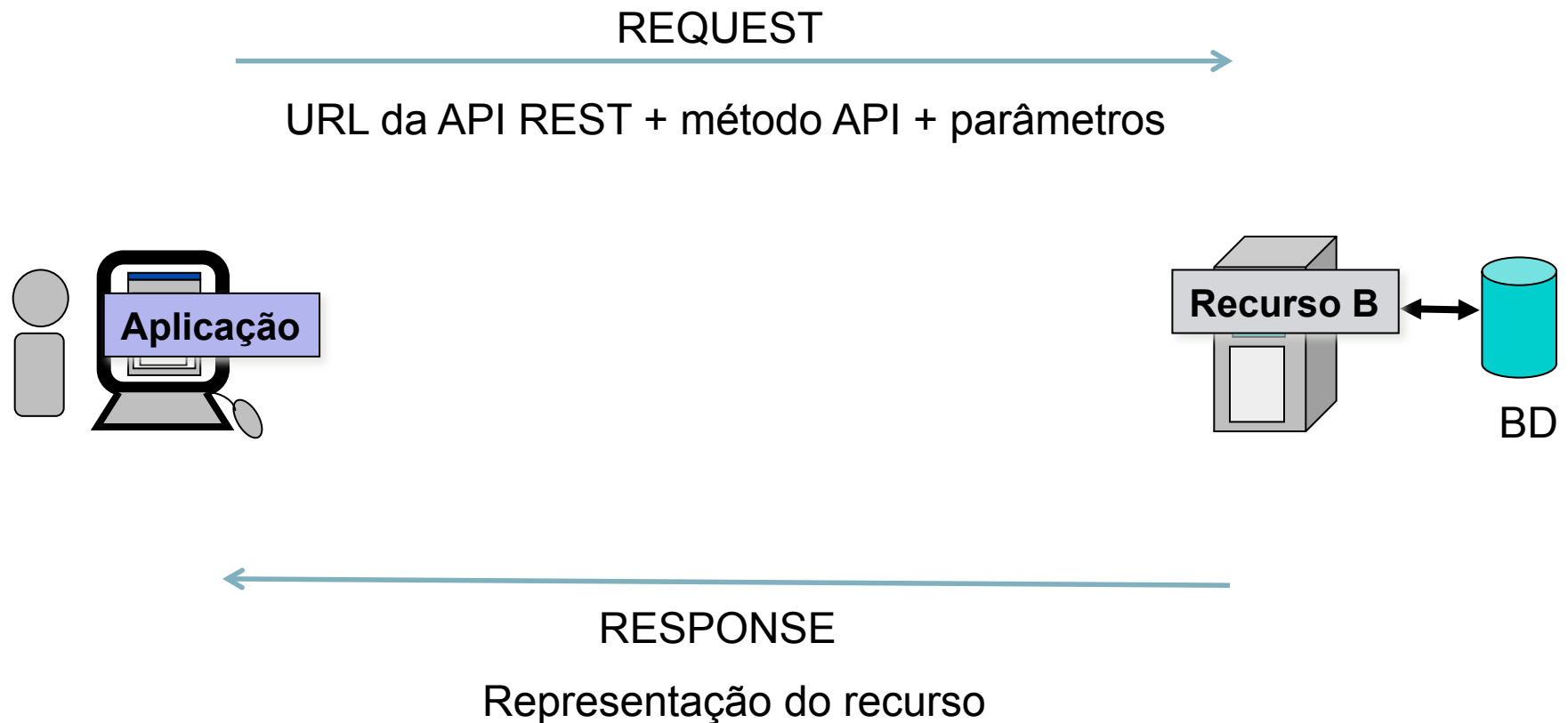
- Cliente-servidor
 - Promove, simplicidade, interoperabilidade, portabilidade, escalabilidade
- Stateless
 - Requisição deve ser auto-contida e servidor não deve guardar informações de requisições anteriores
 - O Estado da sessão é responsabilidade do cliente
- Cacheable
 - Resposta a requisições podem ser armazenadas ou não em cache no cliente para uso posterior
- Sistema em camadas
 - O estilo em camadas permite componentes vendo apenas outros componentes da camada inferior
- Código sob demanda
 - A funcionalidade do cliente pode ser obtida por requisição de scripts a serem executadas
- Interface uniforme
 - Identificação dos recursos, utilização dos recursos através de representações, mensagens auto-descritivas, hipermedia como motor dos estados da aplicação

<https://restfulapi.net/rest-architectural-constraints/>

Recursos

- Importante abstração de REST
- Recursos podem ser:
 - Dados, texto, imagens, funcionalidades (serviços), coleção de recursos
- O recurso é **desacoplado de sua representação** de forma que um mesmo recurso pode ser representado em diversos formatos, JSON, XML, text, PDF, JPEG, PNG
- O **estado** de um recurso é a sua representação em um instante ou intervalo de tempo

Request/Response



Métodos HTTP

- GET
 - Recuperar dados de um recurso específico
- POST
 - Submeter dados a serem processados em um recurso específico
- PUT
 - Atualizar um recurso específico
- DELETE
 - Eliminar um recurso específico

HEAD – Mesmo que GET, sem o corpo

OPTIONS – Retorna os métodos HTTP

PATCH – similar a PUT atualiza recursos parciais

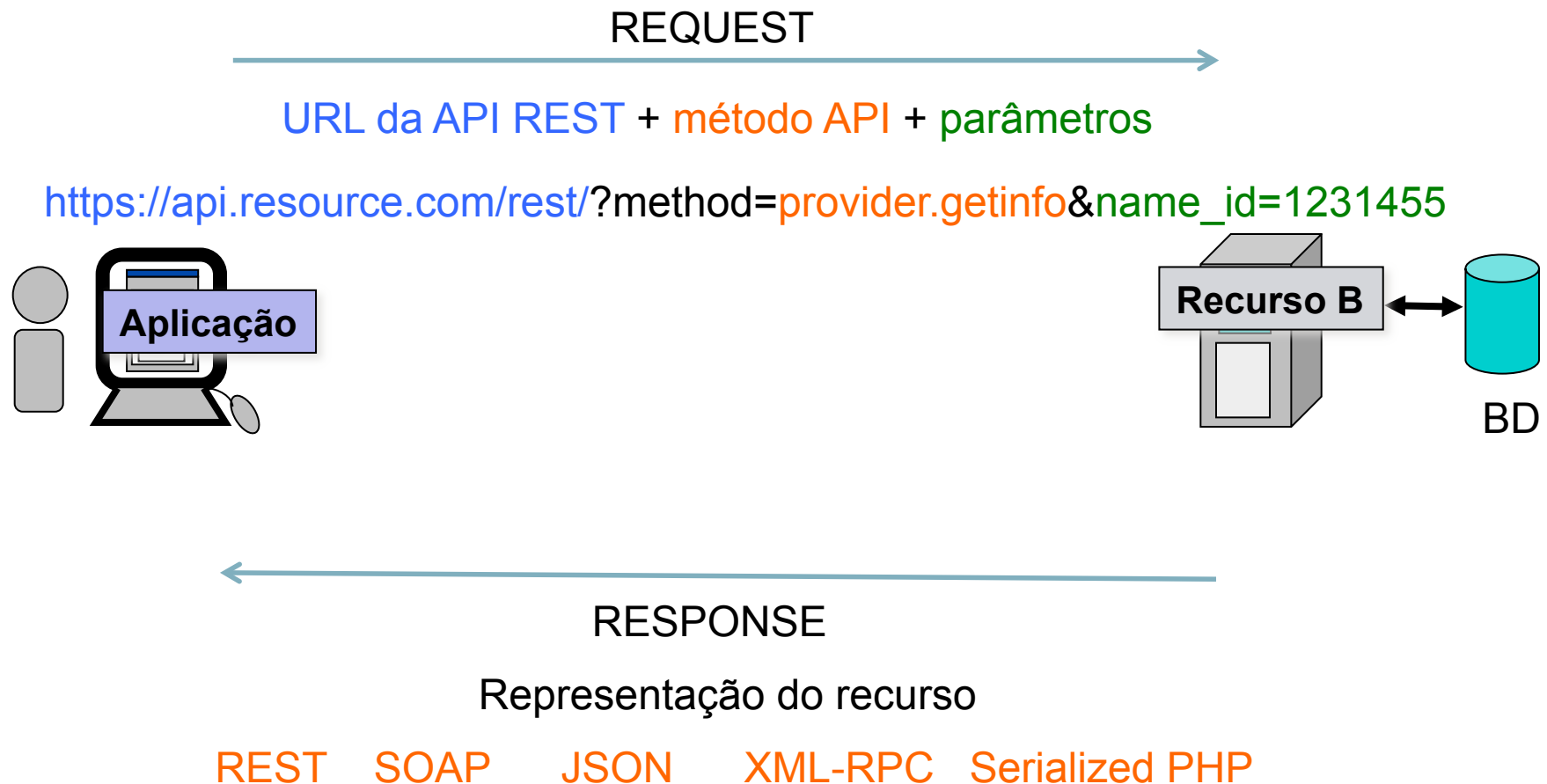
HTTP e CRUD

HTTP METHOD	CRUD
POST	Create
GET	Read
PUT	Update/Replace
PATCH	Partial Update/Modify
DELETE	Delete

Estrutura geral de Request

- Estrutura geral da URL de uma API
 - <http://host:port/version/service/...>
- Exemplos:
 - GET
 - <http://www.appdomain.com/users>
 - <http://www.appdomain.com/users?size=20&page=5>
 - <http://www.appdomain.com/users/123>
 - POST
 - <http://www.appdomain.com/users>
 - <http://www.appdomain.com/users/123/accounts>
 - PUT
 - <http://www.appdomain.com/users/123>
 - <http://www.appdomain.com/users/123/accounts/456>
 - DELETE
 - <http://www.appdomain.com/users/123>
 - <http://www.appdomain.com/users/123/accounts/456>

Exemplo



UTILIZANDO API REST

Clientes para APIs REST

- Linha de comando
 - CURL – Command URL
- Plugins para browsers
 - Ex. ARC - Advanced Rest Client (Chrome), Postman for Chrome
- Aplicações Clientes
 - Ex, Postman, Insomina

API commands

- <https://api.github.com/users/zellwk/repos?sort=pushed>
- API resource endpoint
 - api.github.com
- Resource hierarchy
 - users/zellwk/repos
- Query string
 - ?sort=pushed

Usando CURL

- Instale o CURL no seu computador, se ainda não tiver
- Digite o comando a seguir para obter (GET) um recurso
 - `curl https://api.github.com/users/zellwk/repos\?sort\=pushed`
- Para enviar algo ao recurso usando o POST
 - `curl -x POST -u "username:password" https://api.github.com/user/repos`

Usando um API cliente

The screenshot displays the Postman API client interface. At the top, there's a navigation bar with buttons for 'New', 'Import', 'Runner', and 'My Workspace'. Below this, a search bar and tabs for 'History' and 'Collections' are visible. The main workspace shows a GET request to 'api.github.com/users'. The request is configured with the method 'GET' and the URL 'api.github.com/users'. The response is displayed in the 'Body' tab, showing a JSON object for a user named 'mojombo'. The status is '200 OK', the time is '1074 ms', and the size is '26.98 KB'. The JSON response is formatted in 'Pretty' mode.

```
1 [
2   {
3     "login": "mojombo",
4     "id": 1,
5     "node_id": "MDQ6VXNlcnJE=",
6     "avatar_url": "https://avatars0.githubusercontent.com/u/1?v=4",
7     "gravatar_id": "",
8     "url": "https://api.github.com/users/mojombo",
9     "html_url": "https://github.com/mojombo",
10    "followers_url": "https://api.github.com/users/mojombo/followers",
11    "following_url": "https://api.github.com/users/mojombo/following{/other_user}",
12    "gists_url": "https://api.github.com/users/mojombo/gists{/gist_id}",
13    "starred_url": "https://api.github.com/users/mojombo/starred{/owner}/{repo}",
14    "subscriptions_url": "https://api.github.com/users/mojombo/subscriptions",
15    "organizations_url": "https://api.github.com/users/mojombo/orgs",
16    "repos_url": "https://api.github.com/users/mojombo/repos",
17    "events_url": "https://api.github.com/users/mojombo/events{/privacy}",
18    "received_events_url": "https://api.github.com/users/mojombo/received_events",
19    "type": "User",
20    "site_admin": false
21  },
22  {
```

flickr**Sign Up**

Explore

Create

Get Pro



Photos, people, or groups

The App Garden

[Create an App](#) | **API Documentation** | [Feeds](#) | [What is the App Garden?](#)

The Flickr API is available for non-commercial use by outside developers. Commercial use is possible by prior arrangement.

Read these first:

- [Developer Guide](#)
- [Overview](#)
- [Encoding](#)
- [User Authentication](#)
- [Dates](#)
- [Tags](#)
- [URLs](#)
- [Buddyicons](#)
- [Flickr APIs Terms of Use](#)

API Methods

activity

- [flickr.activity.userComments](#)
- [flickr.activity.userPhotos](#)

auth

- [flickr.auth.checkToken](#)
- [flickr.auth.getFrob](#)
- [flickr.auth.getFullToken](#)
- [flickr.auth.getToken](#)

auth.oauth

- [flickr.auth.oauth.checkToken](#)

Exemplo – Flickr API

The screenshot shows the Flickr API documentation page. The top navigation bar includes the Flickr logo, a 'Sign Up' button, and links for 'Explore', 'Create', and 'Get Pro'. A search bar on the right contains the text 'Photos, people, or groups'. The main content area is titled 'The App Garden' and includes links for 'Create an App' and 'API Documentation'. A paragraph states: 'The Flickr API is available for non-developers. Commercial use is po...'. Below this, a section titled 'Read these first:' lists several links: 'Developer Guide', 'Overview', 'Encoding', 'User Authentication', 'Dates', 'Tags', 'URLs', 'Buddyicons', and 'Flickr APIs Terms of Use'. Three overlapping callout boxes highlight specific sections: 'Photo Upload API' (listing 'Uploading Photos', 'Replacing Photos', 'Example Request', and 'Asynchronous Uploading'), 'Request Formats' (listing 'REST', 'XML-RPC', and 'SOAP'), and 'Response Formats' (listing 'REST', 'XML-RPC', 'SOAP', 'JSON', and 'PHP'). To the right of these boxes, the 'API Methods' section is partially visible, with sub-sections for 'activity' (listing 'flickr.activity.userComments' and 'flickr.activity.userPhotos'), 'auth' (listing 'flickr.auth.checkToken', 'flickr.auth.getFrob', 'flickr.auth.getFullToken', and 'flickr.auth.getToken'), and 'auth.oauth' (listing 'flickr.auth.oauth.checkToken').

flickr Sign Up Explore Create Get Pro Photos, people, or groups

The App Garden

Create an App API Documentation

The Flickr API is available for non-developers. Commercial use is po...

Read these first:

- [Developer Guide](#)
- [Overview](#)
- [Encoding](#)
- [User Authentication](#)
- [Dates](#)
- [Tags](#)
- [URLs](#)
- [Buddyicons](#)
- [Flickr APIs Terms of Use](#)

Photo Upload API

- [Uploading Photos](#)
- [Replacing Photos](#)
- [Example Request](#)
- [Asynchronous Uploading](#)

Request Formats

- [REST](#)
- [XML-RPC](#)
- [SOAP](#)

Response Formats

- [REST](#)
- [XML-RPC](#)
- [SOAP](#)
- [JSON](#)
- [PHP](#)

API Methods

activity

- [flickr.activity.userComments](#)
- [flickr.activity.userPhotos](#)

auth

- [flickr.auth.checkToken](#)
- [flickr.auth.getFrob](#)
- [flickr.auth.getFullToken](#)
- [flickr.auth.getToken](#)

auth.oauth

- [flickr.auth.oauth.checkToken](#)

REST API COM NODEJS/ EXPRESSJS

Usando Express JS como servidor de API

- O papel de um servidor de aplicação para uma API é definir os métodos para atender requisições
- Estrutura básica de roteamento é:
 - `app.METHOD(PATH, HANDLER)`
- Onde
 - `app` é uma instância de `express`.
 - `METHOD` é um método de HTTP request.
 - `PATH` é o recurso
 - `HANDLER` é a function JS a ser executada quando o método for solicitado

Template para roteamento em servidor express JS

- `app.get('/', function (req, res) {`
- `res.send('Hello World!')`
- `}`
- Respond to POST request on the root route (/), the application's home page:
- `app.post('/', function (req, res) {`
- `res.send('Got a POST request')`
- `}`
- Respond to a PUT request to the /user route:
- `app.put('/user', function (req, res) {`
- `res.send('Got a PUT request at /user')`
- `}`
- Respond to a DELETE request to the /user route:
- `app.delete('/user', function (req, res) {`
- `res.send('Got a DELETE request at /user')`
- `}`

Criando API com Node/Express JS

- Exemplo de servidor para dados no formato JSON

```
var express = require('express');  
var app = express();  
var fs = require("fs");
```

```
app.get('/listUsers', function (req, res) {  
  fs.readFile( __dirname + "/" + "users.json", 'utf8', function (err, data) {  
    console.log( data );  
    res.end( data );  
  });  
})
```

```
var server = app.listen(8081, function () {  
  var host = server.address().address  
  var port = server.address().port  
  console.log("Exemplo executando em http://%s:%s", host, port)  
})
```

Usando a API

- No browser, digite
 - localhost:8081/listUsers
- Para acrescentar um usuário, coloque no código

```
var user = {  
  "user4" : {  
    "name" : "mohit",  
    "password" : "password4",  
    "profession" : "teacher",  
    "id": 4  
  }  
}  
(continua...)
```

Usando a API - continuação

```
app.post('/addUser', function (req, res) {  
  // First read existing users.  
  fs.readFile(__dirname + "/" + "users.json", 'utf8',  
    function (err, data) {  
      data = JSON.parse( data );  
      data["user4"] = user["user4"];  
      console.log( data );  
      res.end( JSON.stringify(data));  
    });  
})
```

- Use um cliente para com **POST** para
 - localhost:8081/addUser

Usando a API – obtendo um dado por id

- O código abaixo estende o anterior para definir um método para obter um usuário pelo seu ID

```
app.get('/:id', function (req, res) {  
  // First read existing users.  
  fs.readFile( __dirname + "/" + "users.json", 'utf8', function  
    (err, data) {  
    var users = JSON.parse( data );  
    var user = users["user" + req.params.id]  
    console.log( user );  
    res.end( JSON.stringify(user));  
  });  
})
```

- Use o cliente e, usando GET, indique
 - localhost:8081/2

Usando a API com DELETE

- Para o exemplo anterior, acrescente o método

```
app.delete('/deleteUser', function (req, res) {  
  // First read existing users.  
  fs.readFile( __dirname + "/" + "users.json", 'utf8', function  
  (err, data) {  
    data = JSON.parse( data );  
    delete data["user2"];  
  
    console.log( data );  
    res.end( JSON.stringify(data));  
  });  
})
```
- Use o cliente e, usando **DELETE**, indique
 - localhost:8081/deleteUser

Referencias