Existe vida além do REST?

GET /whoami

Ravan Scafi

Back-end Developer na Leroy Merlin Brasil

Co-organizador do Meetup do Laravel SP

@ravanscafi

API?

O que é uma API?

Interface

é um elemento que proporciona uma ligação física ou lógica entre dois sistemas ou partes de um sistema que não poderiam ser conectados diretamente.

Parte 1

Recursos vs Operações

Quando Pensamos em APIs...

REST

REpresentational **S**tate **T**ransfer

REST

Baseado em Endpoints

```
GET /api/users/rscafi HTTP/1.1
Host: meusite.dev
```

REST

Separado por Recursos (Substantivos)

```
GET /api/users/rscafi HTTP/1.1
Host: meusite.dev
```

REST

Verbos HTTP indicam a ação

```
GET /api/users/rscafi HTTP/1.1
Host: meusite.dev
```

REST

Relação

Cliente-Servidor

REST

Segue convenções HTTP

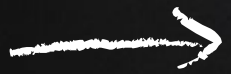
REST

Controle por Hypermedia

(HATEOAS – RMM)

```
GET /api/users/rscafi HTTP/1.1
Host: meusite.dev

{
    "id": "rscafi",
    "name": "Ravan Scafi",
    "website": "http://ravan.me",
    "_links": {
        "self": {
            "href": "http://meusite.dev/api/users/rscafi"
        }
    }
}
```



REST

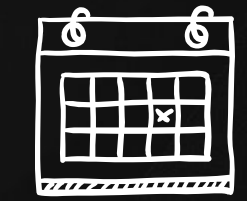
Endpoints cacheáveis

REST

Uniforme

REST

Feito para durar décadas



REST

Problemas

(Spoiler: Parte 1!)

REST - Problemas

Recomendações nem sempre são seguidas

REST - Problemas

Excesso de Roundtrips dependendo da operação

REST - Problemas

Nem sempre os Verbos HTTP disponíveis falam com clareza

Exemplo - Slack

Um usuário pode ser "kickado", "banido"

ou pode "deixar" um canal

Exemplo - Slack

```
DELETE /users/rscafi HTTP/1.1
Host: api.slack.com
```

Exemplo - Slack

```
DELETE /users/rscafi HTTP/1.1
Host: api.slack.com
Content-Type: application/json

{“status”: “kicked”}
```

Exemplo - Slack

```
DELETE /users/rscafi HTTP/1.1
Host: api.slack.com
Content-Type: application/json

{“status”: “kicked”,
“kick_channel”: “random”}
```

Exemplo - Slack

```
DELETE /channels/random/users/rscafi HTTP/1.1
Host: api.slack.com
```

Exemplo - Slack

```
DELETE /channels/random/users/rscafi HTTP/1.1
Host: api.slack.com
Content-Type: application/json

{“status”: “kicked”}
```

REST - Problemas

Operações "BULK"

fogem do Padrão

O QUE FAZER?

RPC

Remote Procedure Call

RPC

Baseado em Endpoints

```
GET /api/getUser HTTP/1.1
Host: meusite.dev
Content-Type: application/json

{“id”: “rscafi”}
```

RPC

Similar à chamada de funções

```
GET /api/getUser HTTP/1.1
Host: meusite.dev
Content-Type: application/json

{“id”: “rscafi”}
```

"Equivalência" em PHP

```php
<?php

// definição
function getUser ($id) {
    //
}

// chamada
getUser('rscafi');
```

RPC

Popularizado pelo SOAP

(Simple Object Access Protocol)

SOAP - Exemplo

```xml
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
        xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Header>
    <ns1:RequestHeader
         soapenv:actor="http://schemas.xmlsoap.org/soap/actor/next"
         soapenv:mustUnderstand="0"
         xmlns:ns1="https://www.google.com/apis/ads/publisher/v201605">
      <ns1:networkCode>123456</ns1:networkCode>
      <ns1:applicationName>DfpApi-Java-2.1.0-dfp_test</ns1:applicationName>
    </ns1:RequestHeader>
  </soapenv:Header>
  <soapenv:Body>
    <getAdUnitsByStatement xmlns="https://www.google.com/apis/ads/publisher/v201605">
      <filterStatement>
        <query>WHERE parentId IS NULL LIMIT 500</query>
      </filterStatement>
    </getAdUnitsByStatement>
  </soapenv:Body>
</soapenv:Envelope>
```

RPC

Porém o SOAP perdeu espaço para as APIs REST

REST - Exemplo

```
GET /ads?parentId=null&limit=500 HTTP/1.1
HOST: api.example.com
```

RPC

Mas não existe somente o SOAP!

RPC baseado em JSON - Exemplo

```
POST /getAdUnitsByStatement HTTP/1.1
HOST: api.example.com
Content-Type: application/json

{"filter": "WHERE parentId IS NULL LIMIT 500"}
```

RPC

Foco em Operações

Exemplo - Slack

```
POST /api/channels.kick HTTP/1.1
Host: slack.com
Content-Type: application/json
{
  "token": "xxxx-xxxxxxxxx-xxxx",
  "channel": "random",
  "user": "rscafi"
}
```

RPC

Relação entre (Micro) Serviços

RPC

A partir de uma Definição de serviços, código é gerado

RPC

Complexidade Abstraída

Google

10.000.000.000

de chamadas RPC

POR SEGUNDO!

gRPC

Framework "universal", open source e performático

gRPC

HTTP/2

gRPC

Protocol Buffers

gRPC

Linguagem de Definição de Serviços

gRPC

```
service Greeter {
  rpc SayHello (HelloRequest) returns (HelloReply) {}
}

message HelloRequest {
  string name = 1;
}

message HelloReply {
  string message = 1;
}
```

gRPC

Gera código para as maiores linguagens

gRPC

```
protoc --proto_path=examples/protos \
  --php_out=examples/php \
  --grpc_out=examples/php \
  --plugin=protoc-gen-grpc=bins/opt/grpc_php_plugin \
  ./examples/protos/helloworld.proto
```

gRPC

Cria stubs de Classes, onde os métodos correspondem a uma rota RPC

gRPC

```php
$request = new Helloworld\HelloRequest();
$request->setName($name);

list($reply, $status) = $client->SayHello($request)->wait();

$message = $reply->getMessage();
```

RPC

Problemas

RPC - Problemas

Abstrai complexidade (!)

RPC - Problemas

Pode complicar mais do que se não estivesse presente

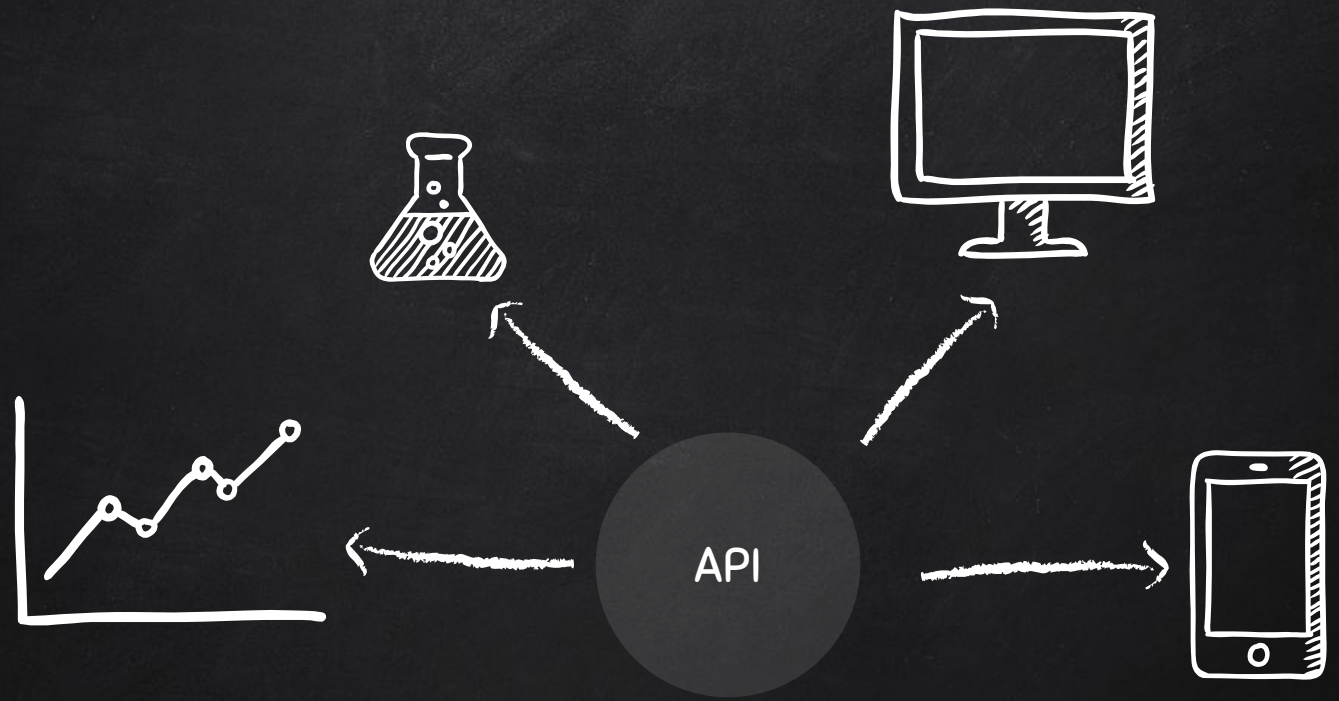
REST vs RPC

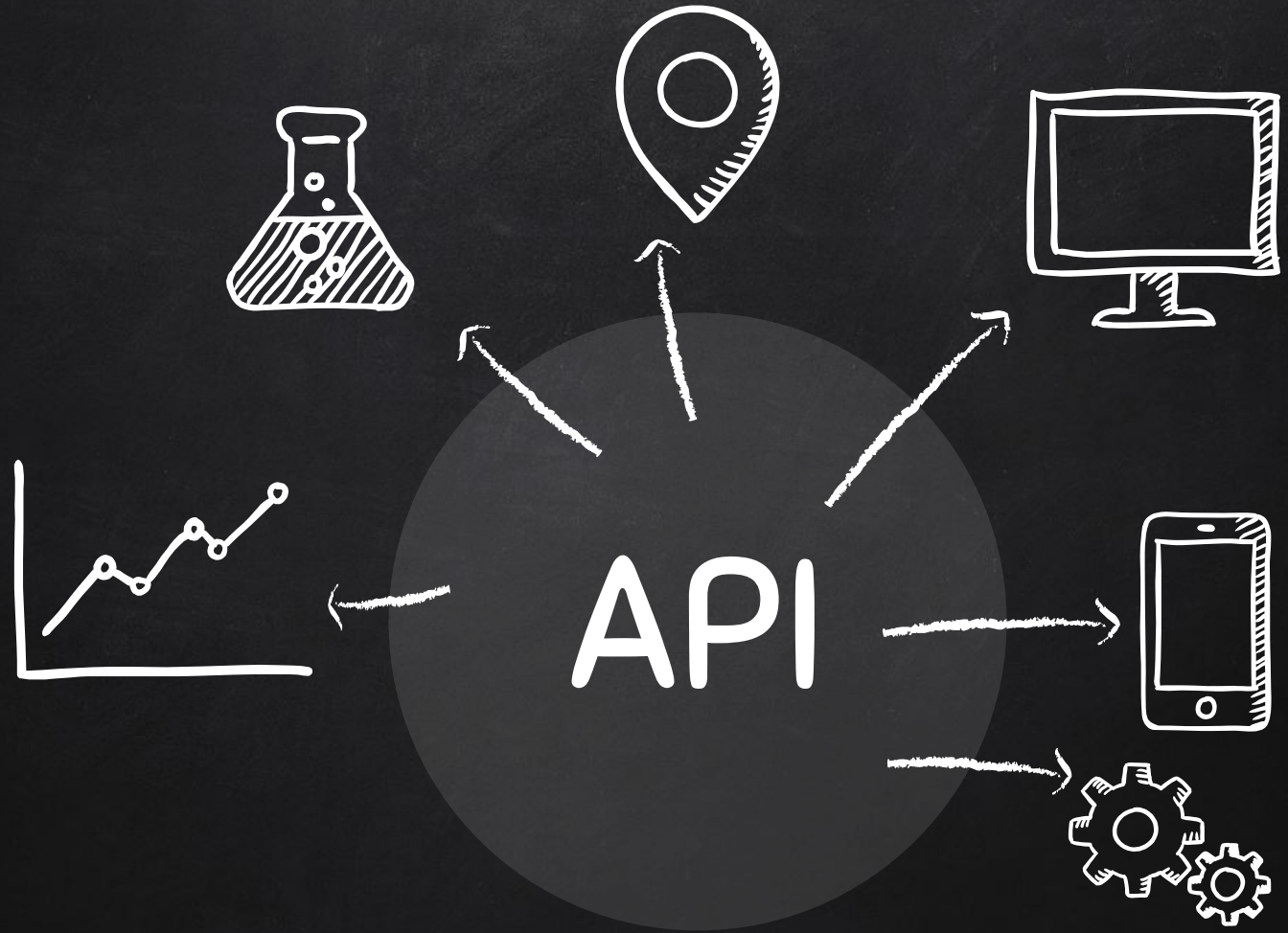
	Foco	Complexidade	Comunicação
REST	Recursos	Exposta	Sistemas
RPC	Operações	Abstraída	Serviços

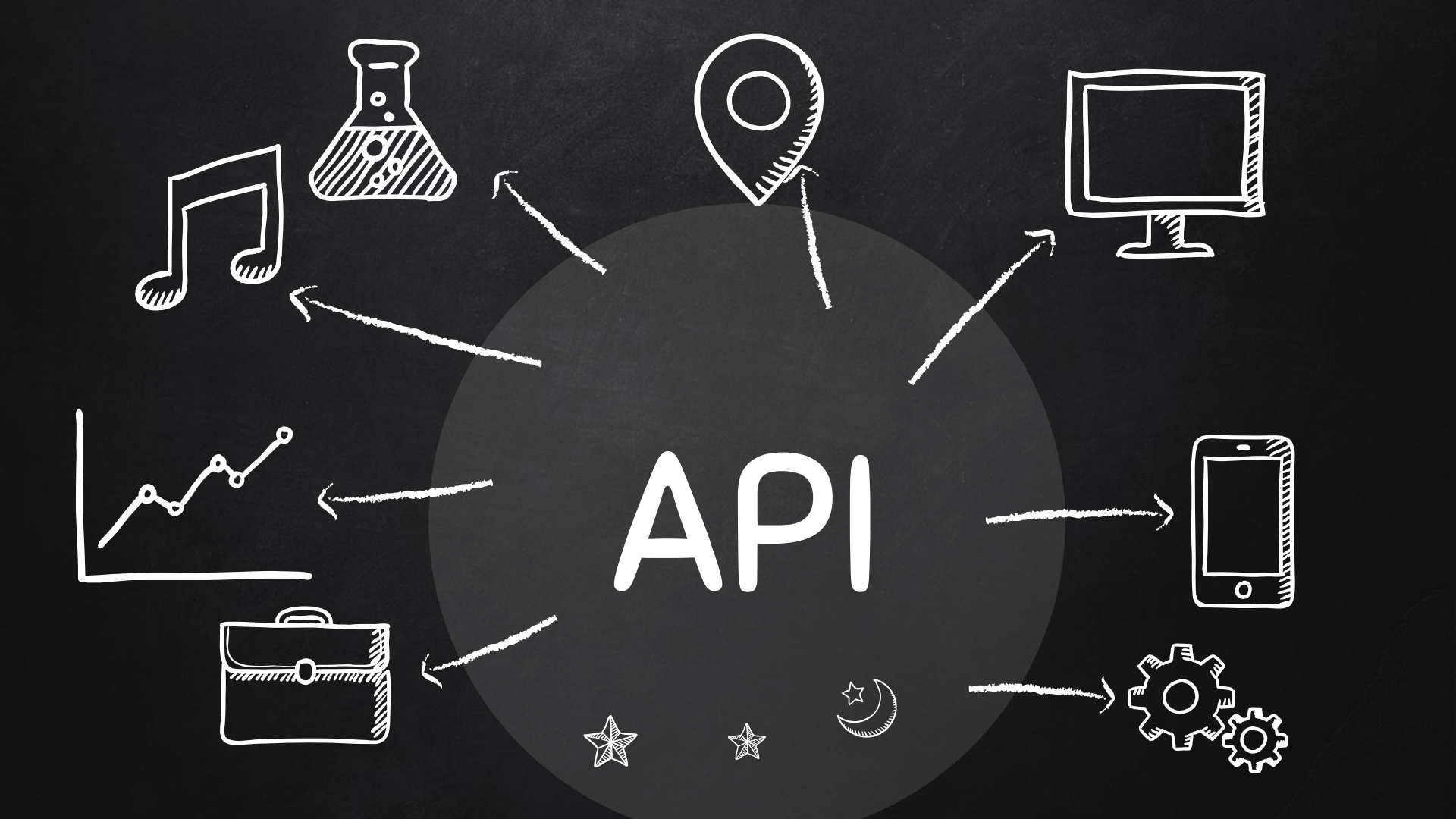
Parte 2

Clientes

Quem vai consumir esses dados?


API


API


API

Como evitar isso?

BFF

Backend For Frontend

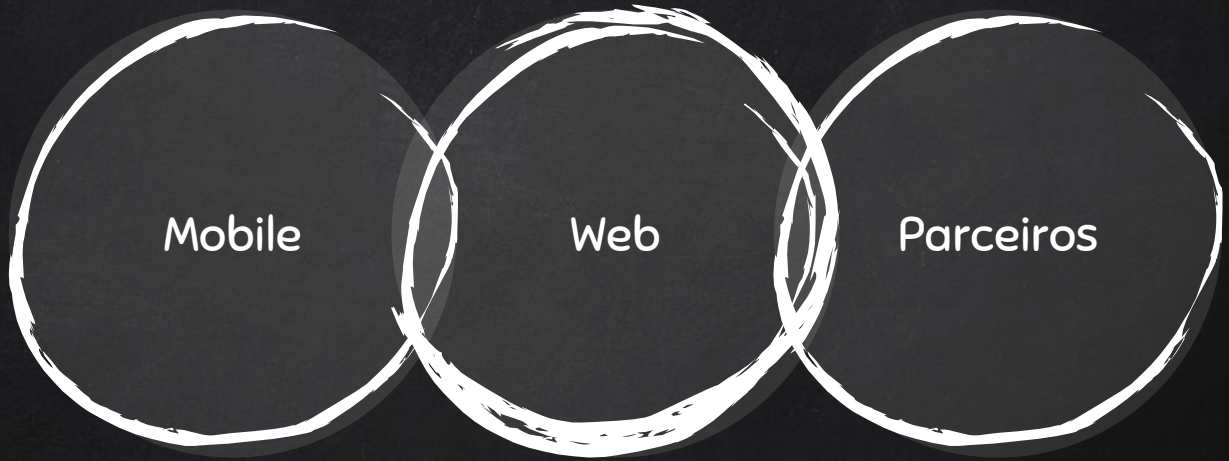
BFF

Abordagem criada pelo SoundCloud

BFF

Clientes da API eram muito diferentes entre si

Clientes do SoundCloud

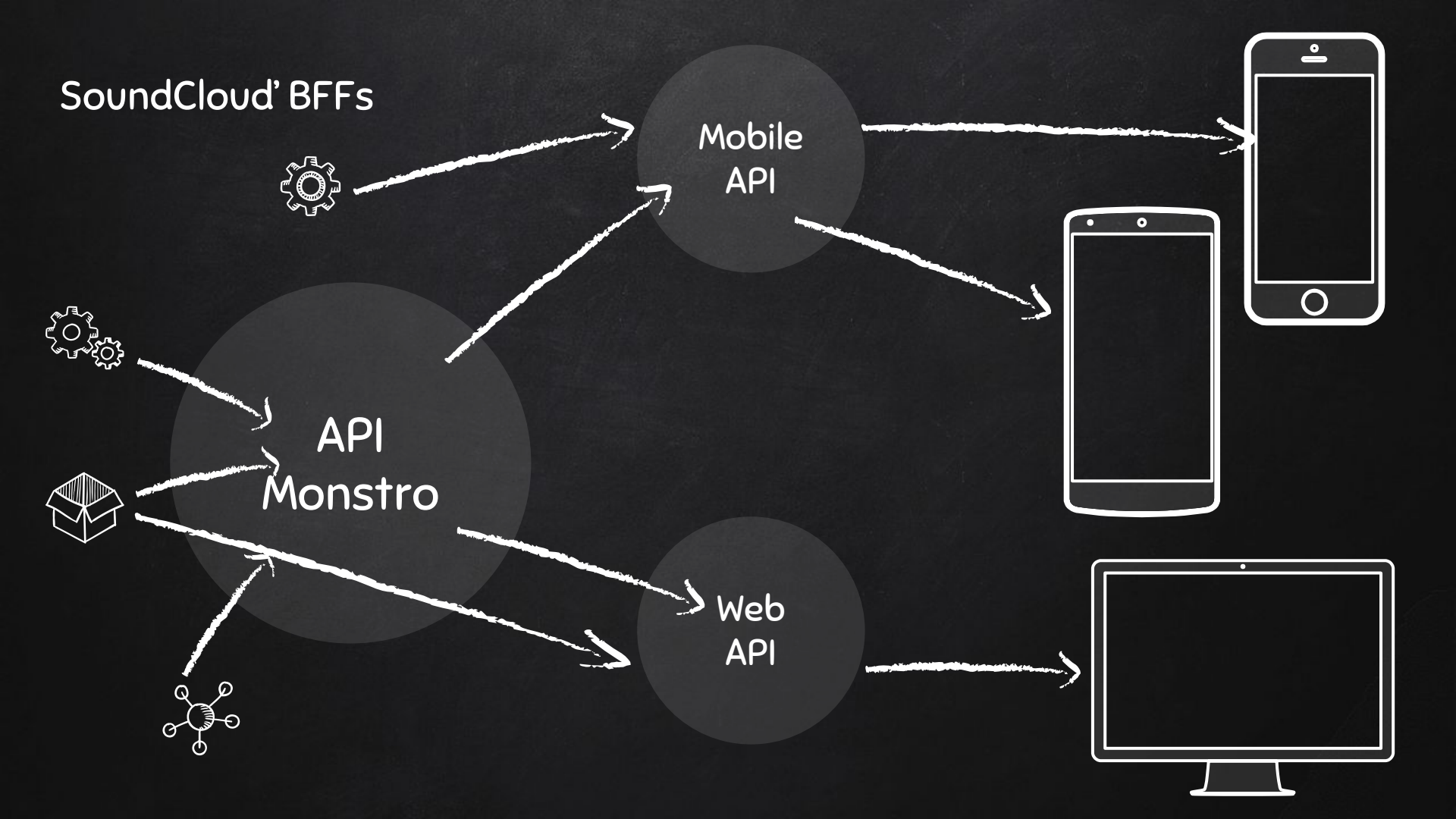


BFF

Cada equipe mantém uma API para atender às suas necessidades

BFF

Menos burocracia para evoluir as APIs


SoundCloud' BFFs
Mobile API
API Monstro
Web API

BFF

Problemas

BFF - Problemas

APIs fragmentadas e em linguagens diferentes

BFF - Problemas

Código duplicado!

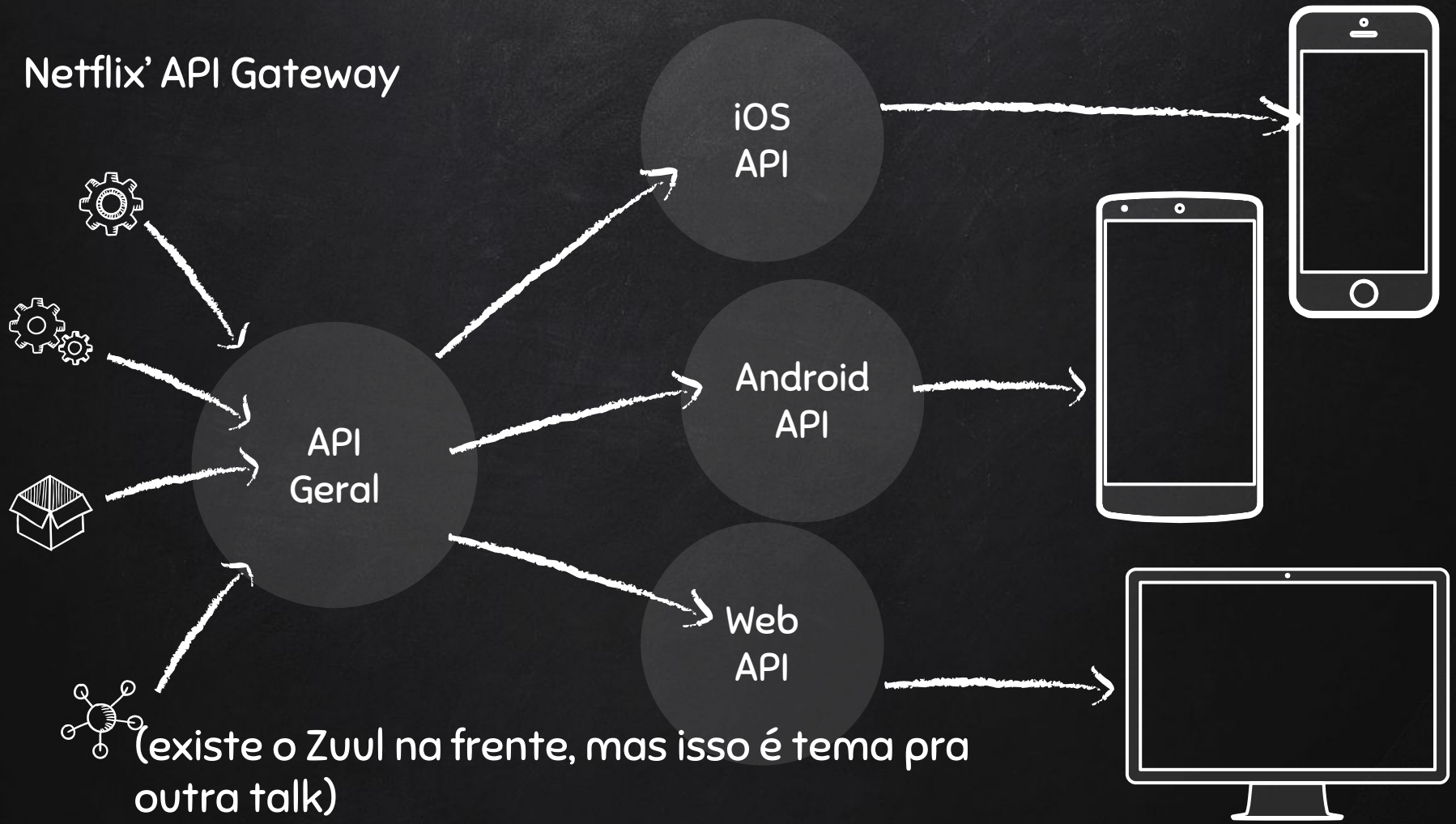
BFF - Problemas

Maior complexidade por cliente

Mas e o REST?

Então é ele o problema?

Netflix' API Gateway



(existe o Zuul na frente, mas isso é tema pra outra talk)

Até funciona,

mas...

REST

Problemas

Parte 2

REST - Problemas

Includes vs Endpoints

em recursos relacionados

REST - Problemas

Overfetching / Underfetching

REST - Problemas

HATEOAS não diz nada sobre formato de Requisições e Respostas

REST - Problemas

Não possui uma Especificação formal

REST - Problemas

API pode rapidamente ficar gigantesca para atender a todos os clientes

REST - Problemas

É difícil evoluir os dados sem saber o que os clientes de fato consomem

REST - Problemas

Quanto mais complexa a API fica,

menos eficaz é o Cache

GraphQL

Graph Query Language

GraphQL

Queries + Mutations + Documentação

GraphQL

Única rota servindo de “tunel” para as queries

GraphQL

```
POST /graphql HTTP/1.1
Host: meusite.dev
Content-Type: application/graphql

{
  users {
    name
  }
}
```

GraphQL

Dados da API são descritos

GraphQL

```
type User {
  name: String
  website: String
  hobbies: [Hobby]
}
```

GraphQL

Um Request pede por dados específicos

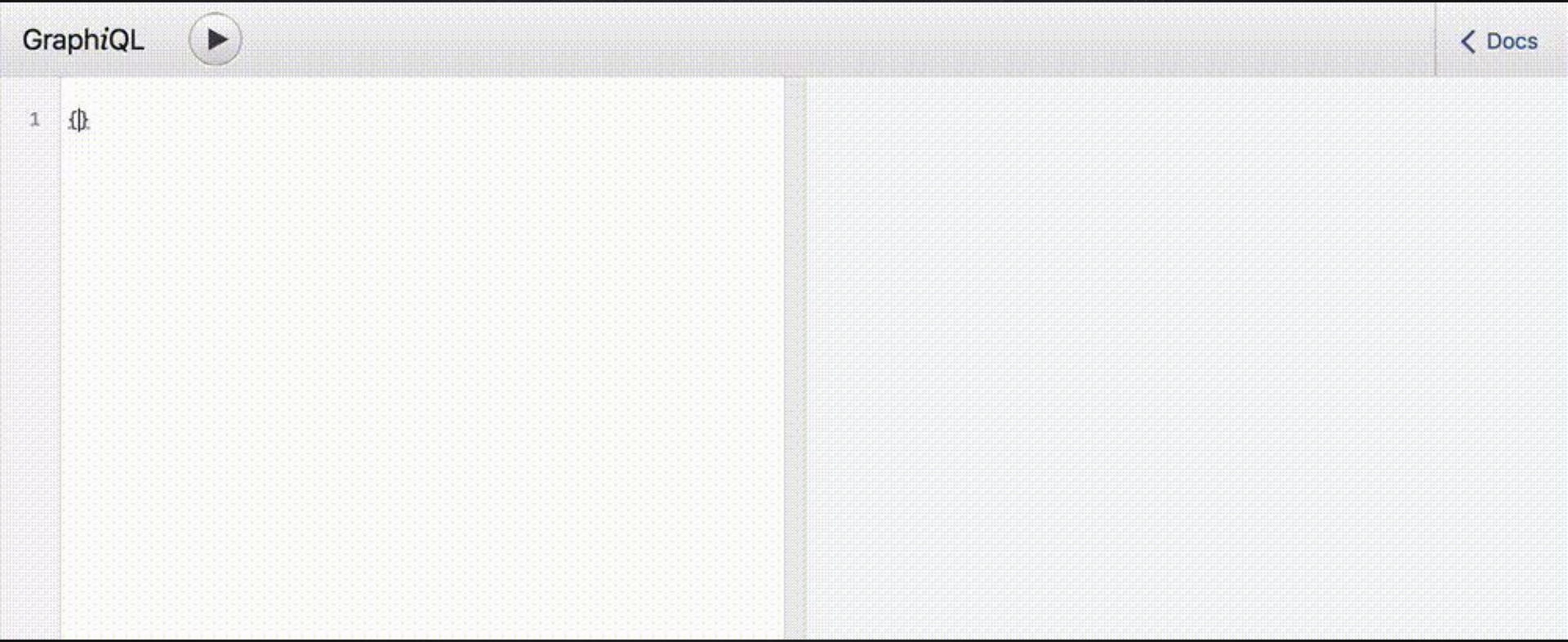
```
{
  user(name: "Ravan Scafi") {
    website,
    hobbies {
        name
    }
  }
}
```

GraphQL

O retorno é exatamente o que foi requisitado

```
{
  "user": {
    "website": "http://ravan.me",
    "hobbies": [
        {"name": "PHP"},
        {"name": "Viajar"}
    ]
  }
}
```

GraphiQL



GraphQL

Mutations enviam dados para o servidor

GraphQL

Especificação “completa” e documentada

GraphQL

Dados podem evoluir com (maior) facilidade

GraphQL

Problemas

(achou que tinha bala de prata?)

GraphQL - Problemas

Técnicas de cache são difíceis

GraphQL - Problemas

Implementação pode ser complexa

GraphQL - Problemas

Próprias (e novas) convenções

GraphQL - Problemas

Upload de arquivos é “hacky”

REST vs GraphQL

	Foco	Implementação	Clientes	Cache
REST	Resiliência	Evolutiva	Parecidos	Robusto
GraphQL	Performance	Deve atender a especificação	Diferentes	Difícil

Conclusões

REST

Atende bem em CRUDs e onde as convenções HTTP possam ser aplicadas.

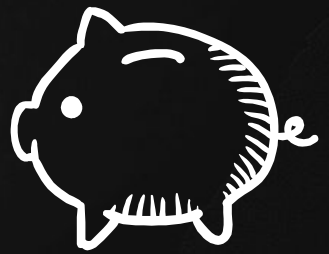
RPC

Atende bem entre serviços e quando o foco é em operações.

GraphQL

Garante performance de Clientes e permite evolução mais rápida.

Dicas de Ouro



Você pode usar mais de uma solução!

(mas Por favor, não na mesma API)

Versionamento é sempre difícil

NÃO EXISTE BALA DE PRATA.

Entenda seus dados e seus Usuários/Clientes

Empatia

é a chave

Empatia

É a capacidade de se identificar com outra pessoa, de sentir o que ela sente, de querer o que ela quer, de apreender do modo como ela apreende etc.

Obrigado!

@ravanscafi

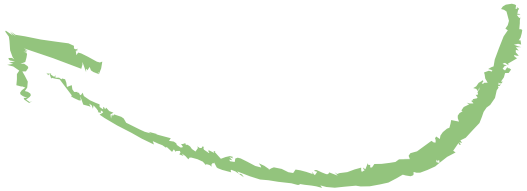
LEROY MERLIN

A casa da sua casa.

Estamos Contratando!

Dev BackEnd, Dev FrontEnd, DevOps

gguitte@leroymerlin.com.br



Agradecimentos

Agradecimentos especiais a todos do SlidesCarnival que fizeram e disponibilizaram o template da apresentação gratuitamente.