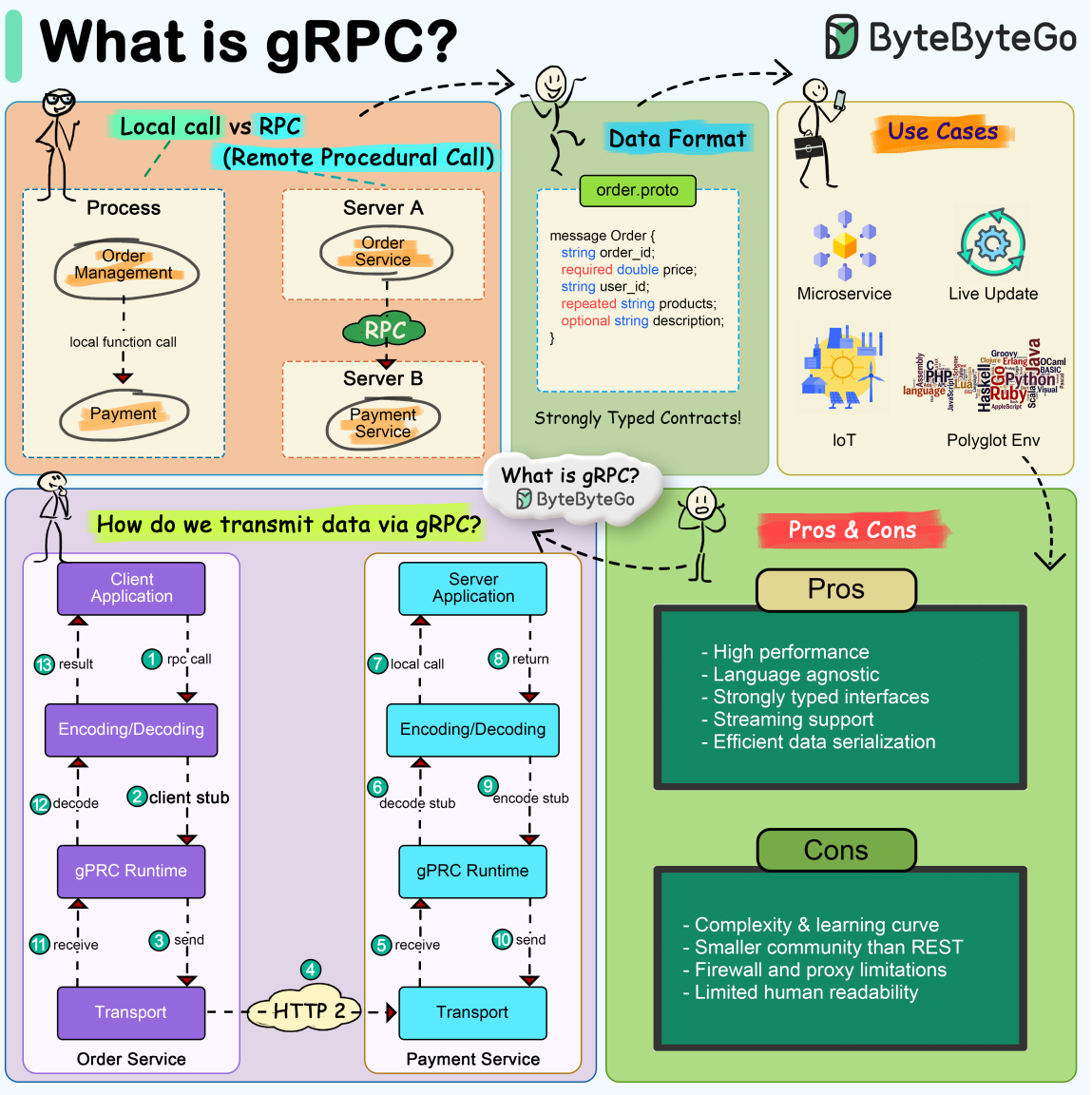


5954025 - Sistemas Distribuídos

Aula 07 - Comunicação via HTTP: gRPC

Prof. Dr. Denis M. L. Martins
DCM | FFCLRP | USP

Objetivos de Aprendizagem

- Conhecer a arquitetura interna do gRPC.
- Entender como o gRPC se baseia no HTTP/2 para multiplexação e comunicação.
- Avaliar trade-offs entre REST/JSON e gRPC/Protobuf.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Microsserviços

Definição: Um estilo arquitetural onde uma aplicação é estruturada como um conjunto de serviços pequenos, independentes e **fracamente acoplados**.

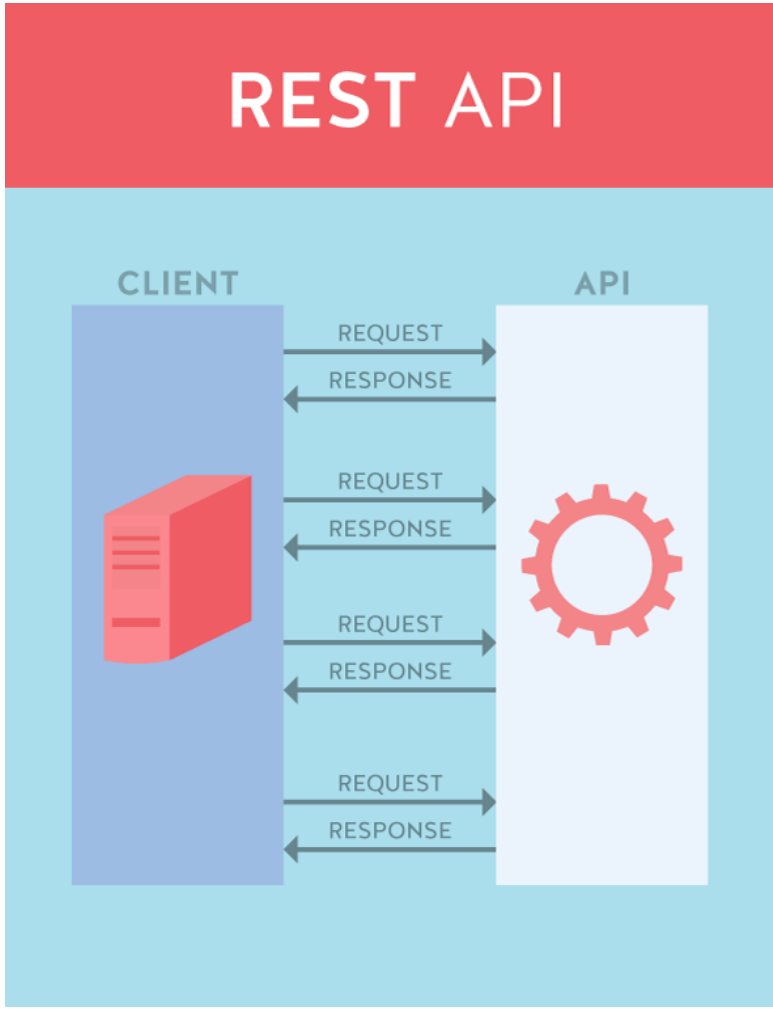
Princípios Fundamentais

1. **Desacoplamento (Loose Coupling):** Serviços comunicam-se via APIs bem definidas, permitindo evolução independente.
2. **Deploy Independente:** Mudanças em um serviço não exigem recompilação ou redeploy do sistema inteiro.
3. **Especialização de Dados:** Cada serviço gerencia seu próprio modelo de dados (Database per Service).

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Comunicação via HTTP

- Microsserviços são frequentemente programados para se comunicar via REST.
- REST não é um protocolo, mas um **estilo arquitetural**.
- **Stateless:** Cada requisição deve conter todas as informações necessárias para ser processada.
- **Client-Server:** Separação clara entre quem consome a API e quem fornece os recursos.
- **Cacheable:** Respostas devem indicar se podem ser armazenadas em cache.
- **Uniform Interface:** Padronização de como clientes e servidores interagem (URIs, métodos HTTP).



> Fonte da imagem ao lado: SystemDesign.us Blog

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

REpresentational State Transfer (REST)

Em REST, os dados são tratados como **recursos** identificados por URIs:

```
GET    /api/alunos/123         # Leitura
POST   /api/alunos             # Criação
PUT    /api/alunos/123         # Atualização completa
DELETE /api/alunos/123         # Exclusão
```

JSON (JavaScript Object Notation) é o formato de troca de dados mais utilizado em APIs REST:

- **Legibilidade:** Texto humano-legível facilita *debugging* e documentação.
- **Universalidade:** Suportado nativamente por JavaScript, Python, Java, Go, etc.

```
{
    "id": "123",
    "email": "ana@exemplo.com",
    "idade": 25
}
```


Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Limitação do Tradicional REST/JSON

- **Overhead:** Serialização JSON é verbosa. Grande consumo de banda.
- **Contrato Fraco:** Falta tipagem forte entre o cliente e o servidor.
- **Problema:** Em microsserviços internos (Service-to-Service), precisamos de mais eficiência.
- **Exemplo:** Um sistema com 50 microsserviços REST precisa enviar milhões de pequenos JSONs, gastando muita banda desnecessariamente.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Remote Procedure Call (RPC)

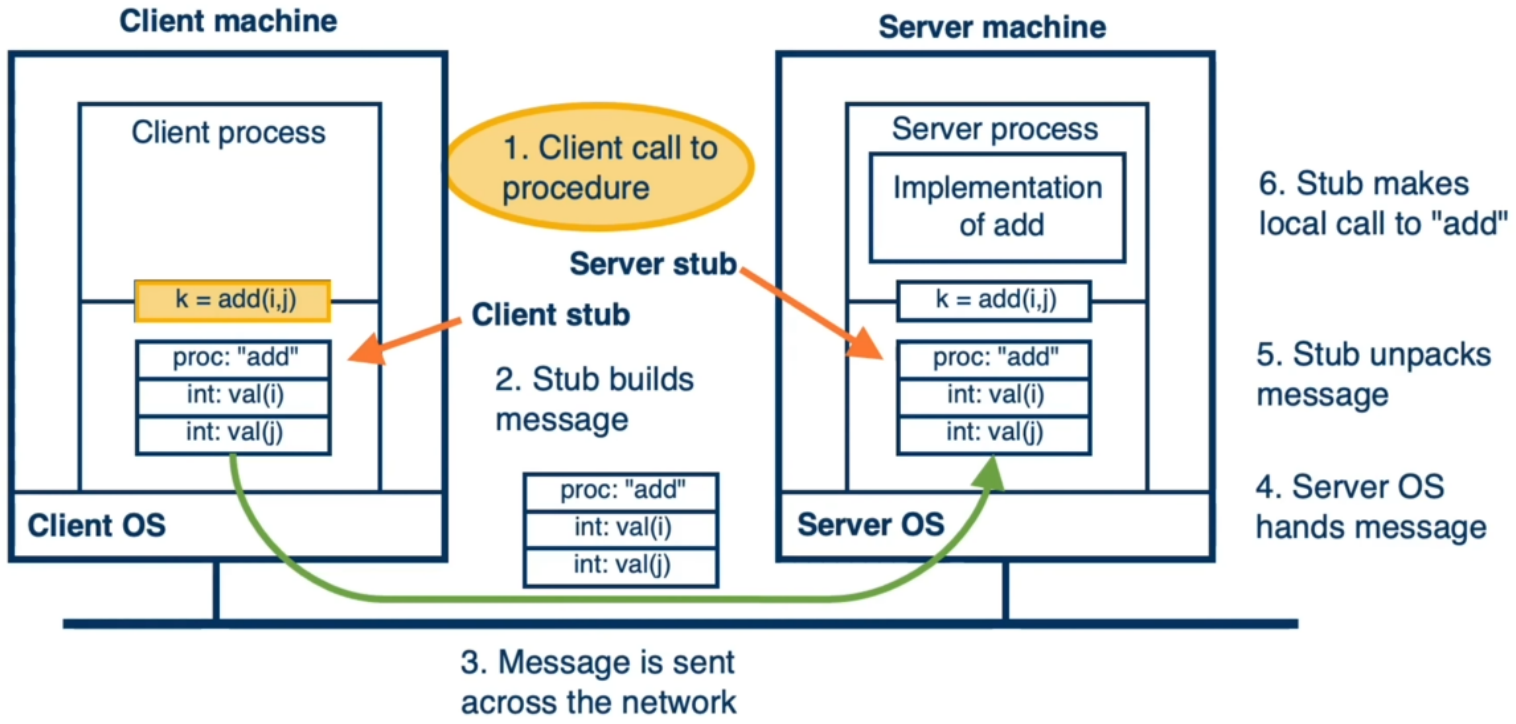
Máquina 1: Programa Cliente

```
def mult(a, b):
    return a * b

def main():
    l = mult(5, 7)
    k = add(1, 5)
```

Máquina 2: Programa Servidor

```
def add(i, j):
    return i + j
```

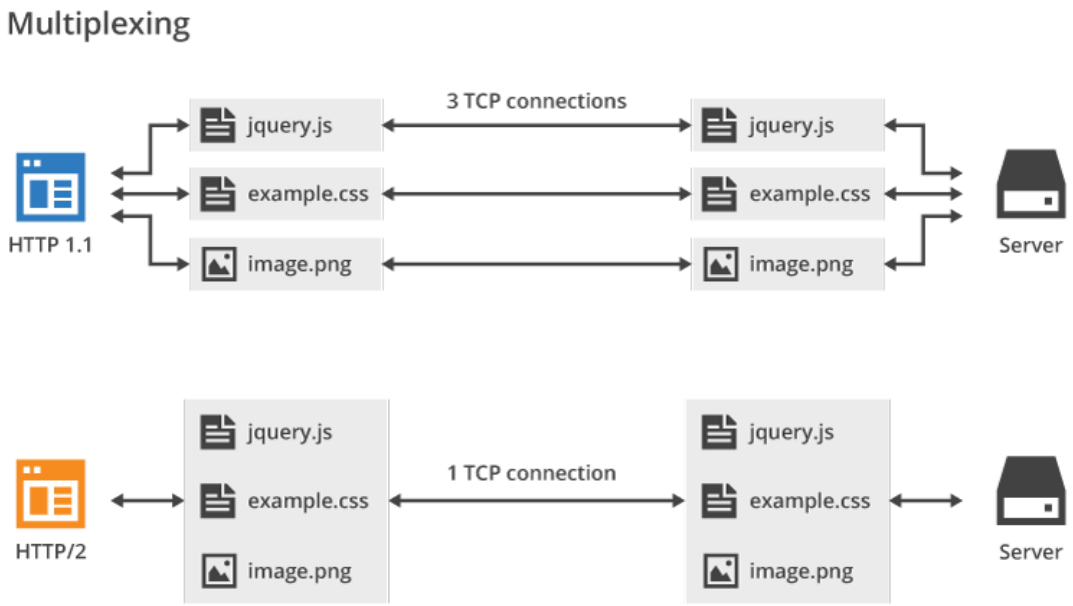


RPC por Chirs Wirz. Note que cliente e servidor podem estar em linguagens de programação diferentes.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Por que precisamos de gRPC?

- HTTP/1.1 sofre com *Head-of-Line Blocking*: paralelismo limitado.
- Necessidade de parsing complexo para ignorar comentários ou espaços extras; vulnerável a ataques de injeção de linha.
- APIs modernas exigem **latência baixa** e **streaming** (ex.: microsserviços).
- gRPC oferece RPCs tipados, multiplexação e compressão de cabeçalhos.



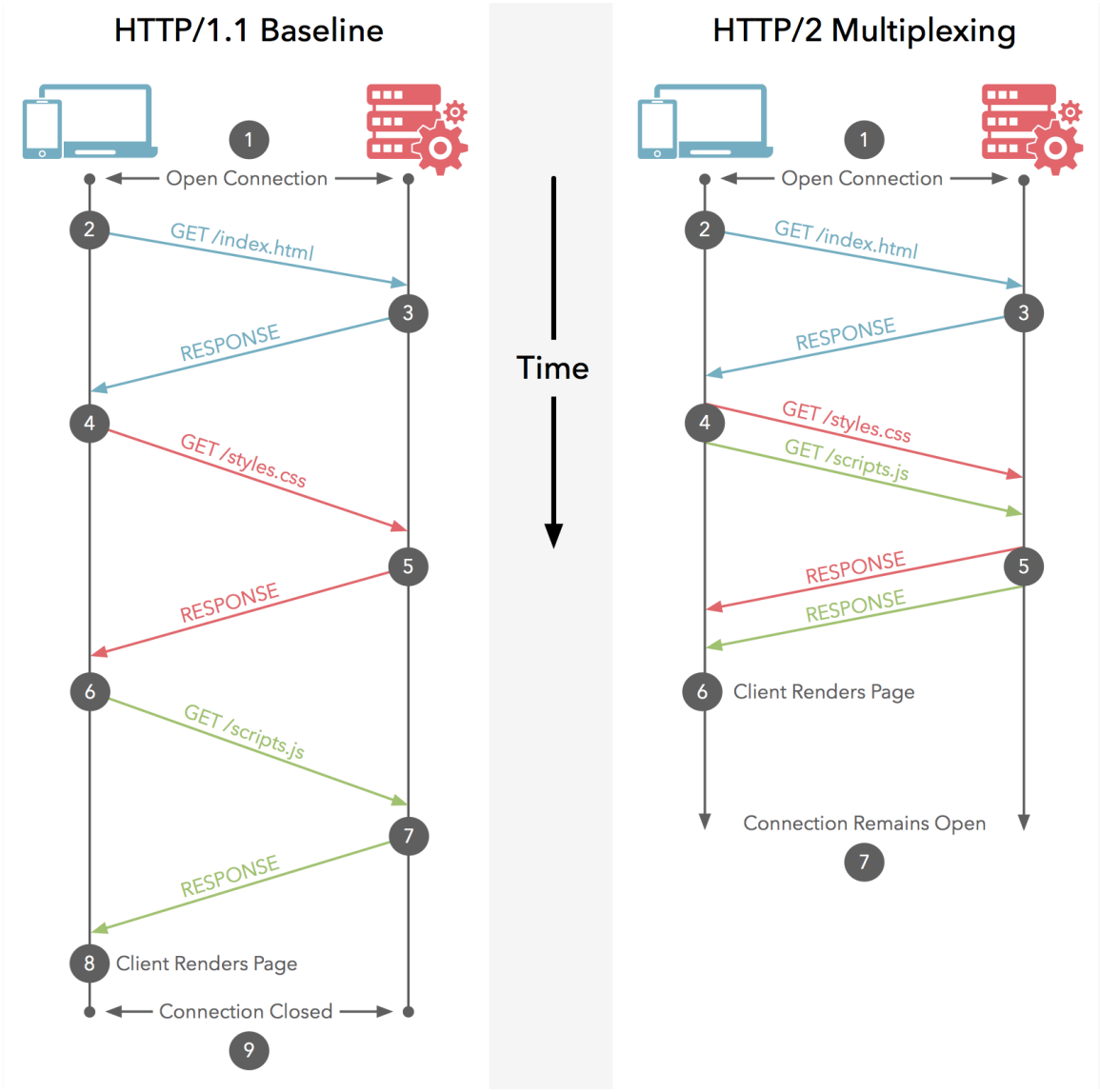
> Fonte da imagem ao lado: Cloudfare

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

HTTP/2

- **Formato binário estruturado**: Parsing determinístico e rápido. O protocolo define exatamente onde os dados começam e terminam.
- Redução da latência de processamento no lado do servidor (Server-side).
- Capacidade de enviar múltiplos streams independentes sobre uma única conexão TCP

> Mais informação em HTTP/2 por Julia Evans
>
> Fonte da imagem ao lado: DZone



Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Problema na Comunicação de Dados

Como representar dados estruturados para que possam ser transmitidos ou persistidos entre *diferentes* componentes de software (ex: serviços)?

Problema na Comunicação de Dados: Exemplo

- **Contexto**: Desenvolvimento de uma agenda de contatos simples em arquitetura cliente-servidor.
  - **Estrutura**: Nome, ID, Email, Telefone.
  - **Requisito**: Capacidade de leitura e escrita desses dados em um *stream* ou arquivo persistente.
- Quais **alternativas** no contexto de comunicação via HTTP?

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Problema na Comunicação de Dados: Exemplo (cont.)

- **Contexto**: Desenvolvimento de uma agenda de contatos simples em arquitetura cliente-servidor.
  - **Estrutura**: Nome, ID, Email, Telefone.
  - **Requisito**: Capacidade de leitura e escrita desses dados em um *stream* ou arquivo persistente.
- **Alternativas**:
  - **Python Pickling**: Serialização nativa da linguagem. Baixa interoperabilidade (com C++ ou Java, por exemplo).
  - **XML**: Linguagem baseada em marcação (DOM). Consumo excessivo de espaço (*space-intensive*).
  - **Ad-Hoc**: Criar seu próprio parser e codificador. Boa sorte...

> Exemplo baseado em https://protobuf.dev/getting-started/pythontutorial/

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

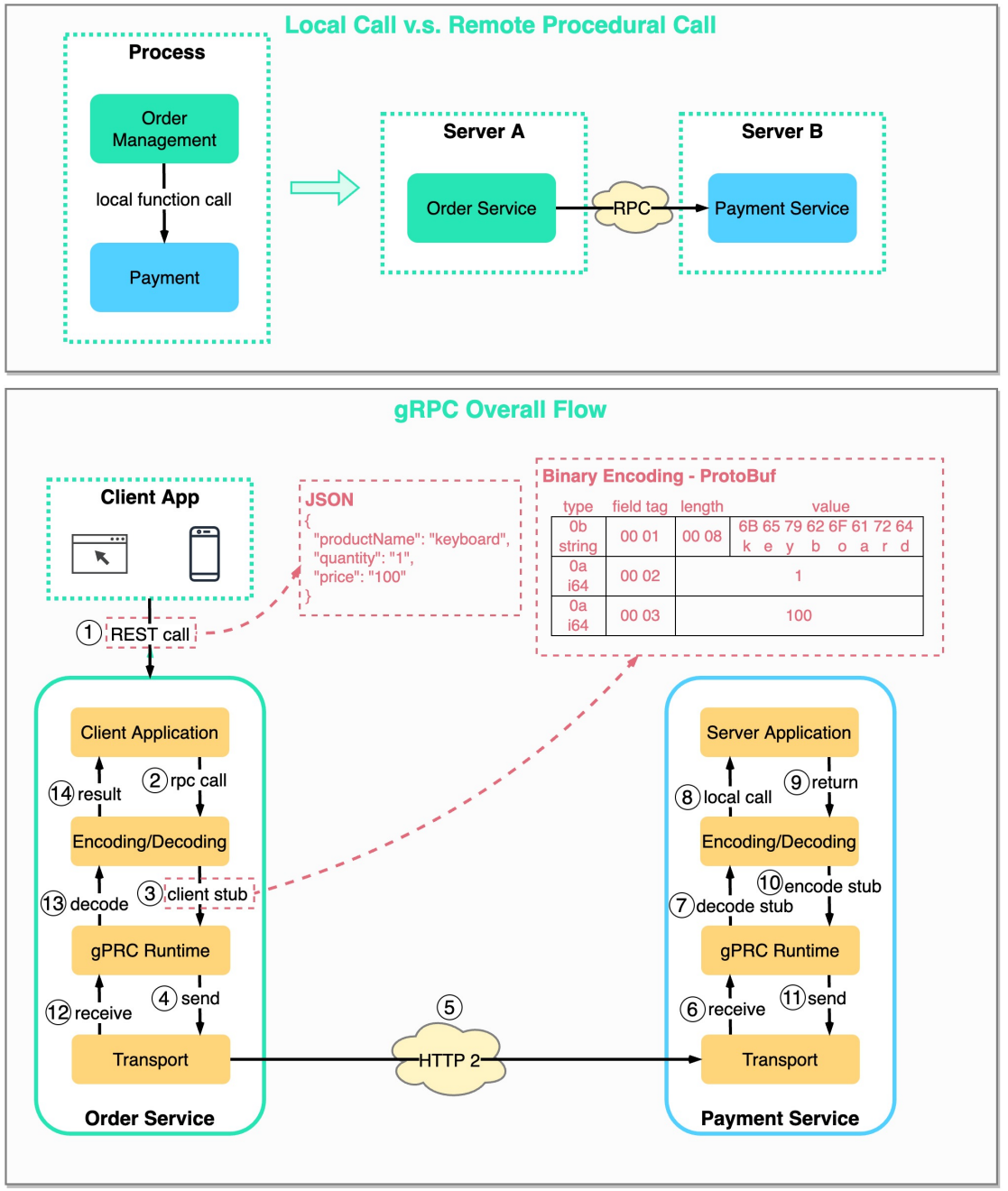
Desiderata

Um mecanismo *agnóstico* em relação à linguagem e plataforma para serializar dados estruturados em um formato altamente *eficiente*.

Visão geral do gRPC

- gRPC (Google Remote Procedure Call): Um framework de comunicação.
- Permite chamar métodos em um servidor remoto como se fossem locais.
- **Transporte**: HTTP/2.
- **Serialização padrão**: Protocol Buffers.
- **Modelos de chamada**: Unary, Server Streaming, Client Streaming, Bidirectional.

> Fonte da imagem ao lado: bytebytego



Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Protocol Buffers (Protobuf)

- Protocol Buffers: Mecanismo de serialização de dados estruturados.
  - Agnóstico em relação à linguagem e à plataforma.
  - Define a estrutura (*schema*) dos dados em um arquivo **.proto**.
- **Benefício Principal:** Binário e muito compacto. Mais rápido e menor que JSON/XML.
- Garante **tipagem forte**: o contrato é rigoroso (client sabe exatamente o que esperar).

```
edition = "2023";

package addressbook;

message Person {
  string name = 1;
  int32 id = 2;
  string email = 3;

  enum PhoneType {
    PHONE_TYPE_MOBILE = 1;
    PHONE_TYPE_WORK = 2;
  }

  message PhoneNumber {
    string number = 1;
    PhoneType type = 1 [default = PHONE_TYPE_MOBILE];
  }

  repeated PhoneNumber phones = 4;
}

message AddressBook {
  repeated Person people = 1;
}
```


Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Protocol Buffers (Protobuf)

- **Package declaration**: Previne colisões de nomes (n*aming conflicts*) entre diferentes projetos ou módulos.
- **message** é a unidade fundamental de dados. Encapsula um conjunto de campos tipados.
- **Tags e Otimização Binária**: Marcadores numéricos, como " = 1", associados a cada campo, identificam o rótulo único (tag) que o campo utilizará durante a codificação binária.

```
edition = "2023";

package addressbook;

message Person {
  string name = 1;
  int32 id = 2;
  string email = 3;

  enum PhoneType {
    PHONE_TYPE_MOBILE = 1;
    PHONE_TYPE_WORK = 2;
  }

  message PhoneNumber {
    string number = 1;
    PhoneType type = 1 [default = PHONE_TYPE_MOBILE];
  }

  repeated PhoneNumber phones = 4;
}

message AddressBook {
  repeated Person people = 1;
}
```


Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Como o gRPC usa HTTP/2

Multiplexação: Permite enviar múltiplos *streams* em uma única conexão TCP.

- Compressão eficiente: Compacta cabeçalhos (HPACK), economizando banda.
- Controle de fluxo por *flow-control windows*.

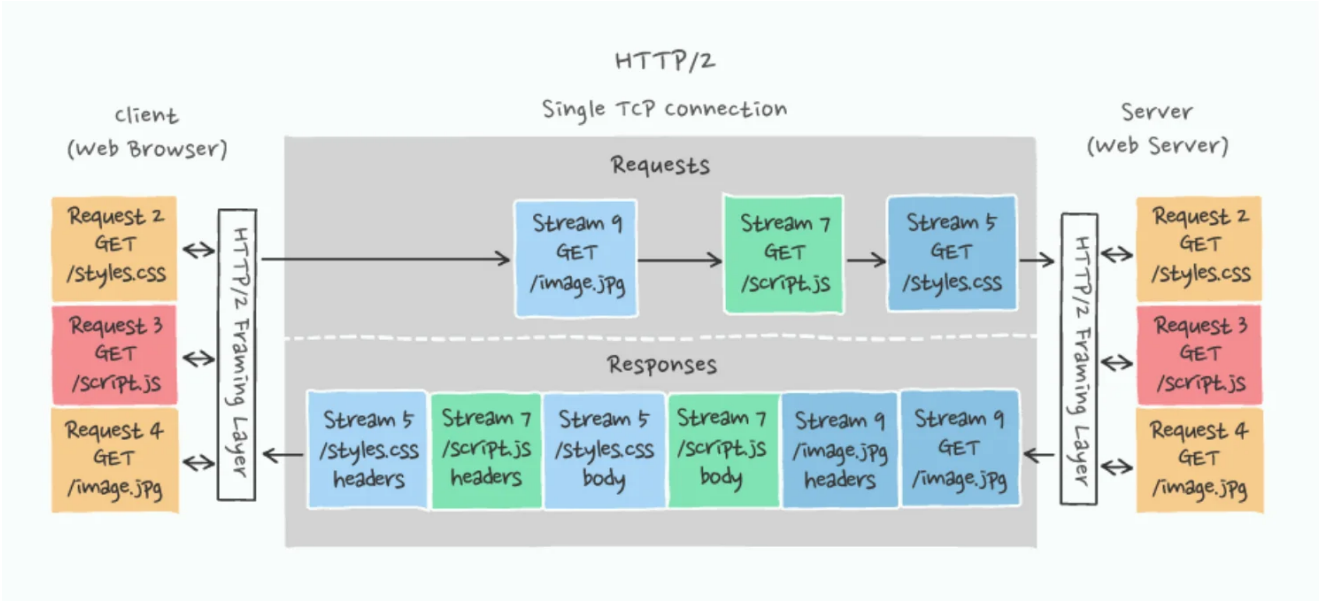


Fig. .1: Requests e responses entre cliente e servidor sobre TCP e HTTP/2. Fonte: https://ably.com/topic/http-2-vs-http-3

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Métodos de Streaming RPC

- **1. Unary RPC:** A chamada clássica (Cliente $\rightarrow$ Servidor, Resposta única).
- **2. Server Streaming:** Cliente envia 1x. Servidor devolve um fluxo de dados contínuo.
- **3. Client Streaming:** Cliente envia um fluxo contínuo de dados. Servidor responde uma vez com o resultado final.
- **4. Bidirectional Streaming:** Fluxo constante e simultâneo nas duas direções.

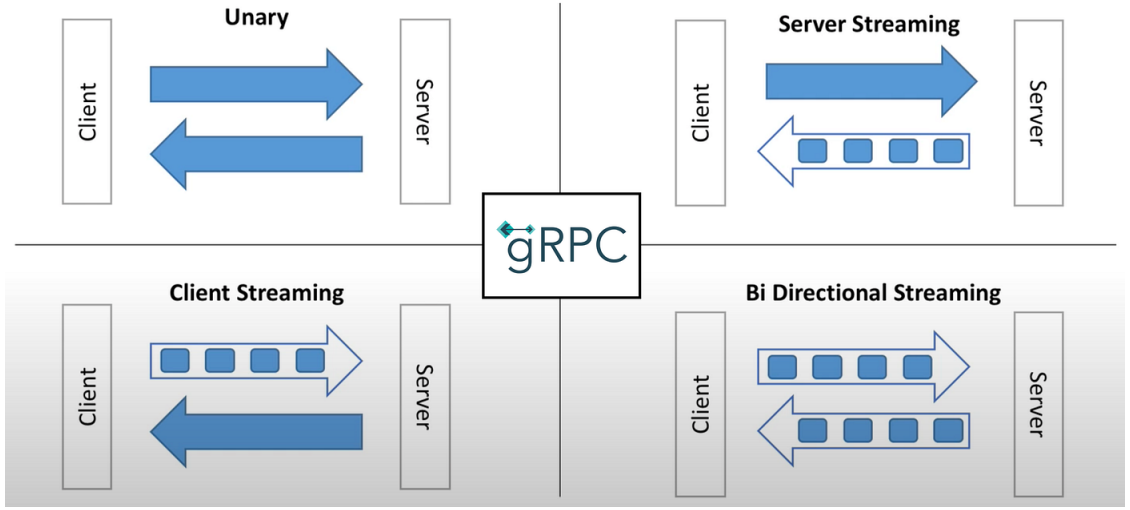


Fig. .1: Métodos de streaming em gRPC. Fonte: towardsdev.com/4-types-of-grpc-apis-with-go-and-java-example-dc4db630fc83

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Modo híbrido: REST + API Gateway + gRPC

- REST para requisições externas via API Gateway.
- Comunicação interna via gRPC.

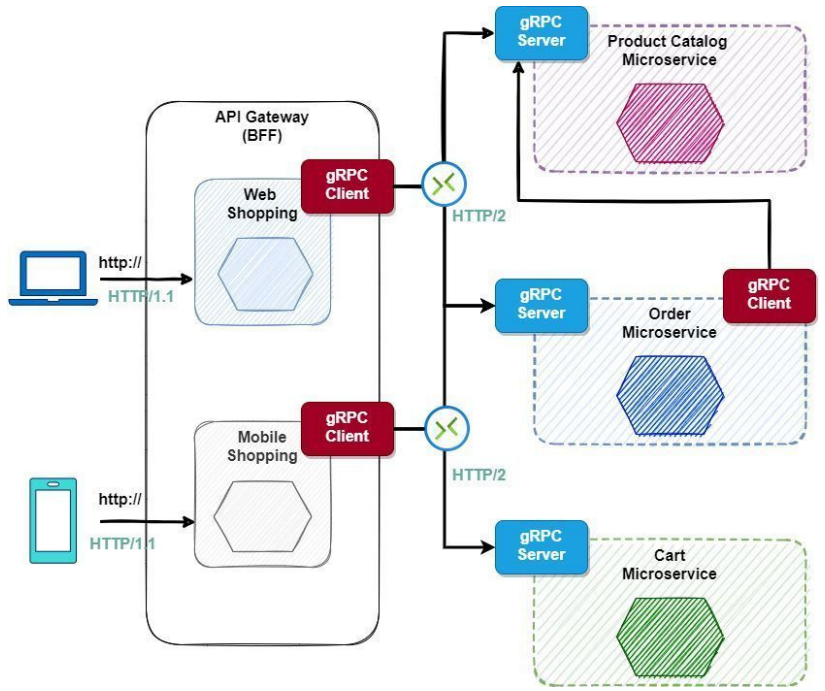


Fig. .1: REST + gRPC em microsserviços. Fonte: https://www.linkedin.com/posts/vintageglobal_grpc-microservices-apigateway-activity-6846382683614916608-aaK3

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Conclusão e Próximos Passos

- **REST/HTTP:** Melhor para APIs **Client-to-Gateway** (externa). Simples, fácil consumo por navegadores e terceiros.
- **gRPC:** Ideal para comunicação **Service-to-Service** (interna). Alta performance, exige contrato estrito.
- **Protobuf:** Define um contrato binário de alto desempenho.
- **HTTP/2:** Base para baixa latência e multiplexação.
- **Comunicação Híbrida em Microsserviços:** Usar gRPC *dentro* do seu backend de microsserviços e expor uma camada REST ou GraphQL no *Edge/API Gateway*.
  - **Exemplo:** O front-end (cliente) fala REST com o API Gateway, que por sua vez usa gRPC para conversar otimamente com os serviços internos.
- **Material Adicional**: Protocol Buffers Crash Course | 99% of Developers Don't Get gRPC | gRPC Dicionário do Programador |

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Exercícios Teóricos

1. **(Conceito)** Qual é o principal benefício de usar Protocol Buffers (Protobuf) em comparação com JSON?
2. **(Definição)** O que significa dizer que um sistema utiliza "tipagem forte" no contexto do gRPC?
3. **(Identificação)** Qual protocolo de transporte deve ser usado pelo gRPC para garantir a eficiência e o streaming avançado?
4. **(Comparação)** Um time decide criar uma API pública que será consumida por diversos aplicativos legados (que só entendem JSON). Onde, dentro da arquitetura de microsserviços, seria mais apropriado usar gRPC e onde deveria ser usado REST? Justifique a separação.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Lab Prático 1

Siga o tutorial do Google Codelabs Getting Started with gRPC-Python.

Para outras linguagens de programação, siga os tutoriais em https://grpc.io/ e https://protobuf.dev/getting-started/

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Lab Prático 2

Objetivo: Implementar a leitura e atualização de dados simples em um serviço centralizado.

Você está construindo o módulo de perfil de usuário de uma plataforma. Este módulo deve interagir com um serviço remoto para buscar informações e atualizar campos específicos.

Componentes:

- Mensagem **`UserRequest`**: Deve conter apenas o **`user_id`** (string).
- Mensagem **`UserResponse`**: Deve conter **`user_id`**, **`name`** e **`email`**.
- Serviço **`UserService`** com dois métodos:
  - **`GetUserDetails(UserRequest)`**: Retorna um único **`UserResponse`**. Simule buscar um usuário (pode ser um dicionário estático) baseado no ID fornecido. Se o usuário não existir, lance uma exceção apropriada.
  - **`UpdateEmail(UserRequest, UserResponse)`**: Recebe o ID e retorna uma confirmação de sucesso. Imprima um *log* simulando a atualização bem-sucedida no banco de dados.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Questão para Reflexão

Em um cenário onde múltiplos micro-serviços escritos em linguagens diferentes precisam trocar dados, quais são os principais desafios ao escolher gRPC? Como você mitigaria esses problemas?

Dúvidas e Discussão

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br