Fullstack Development

API Architectures and Design #1

Content

- **What is API?**
- **API Architecture Styles**
- RESTful API design
- API Security
- API Testing

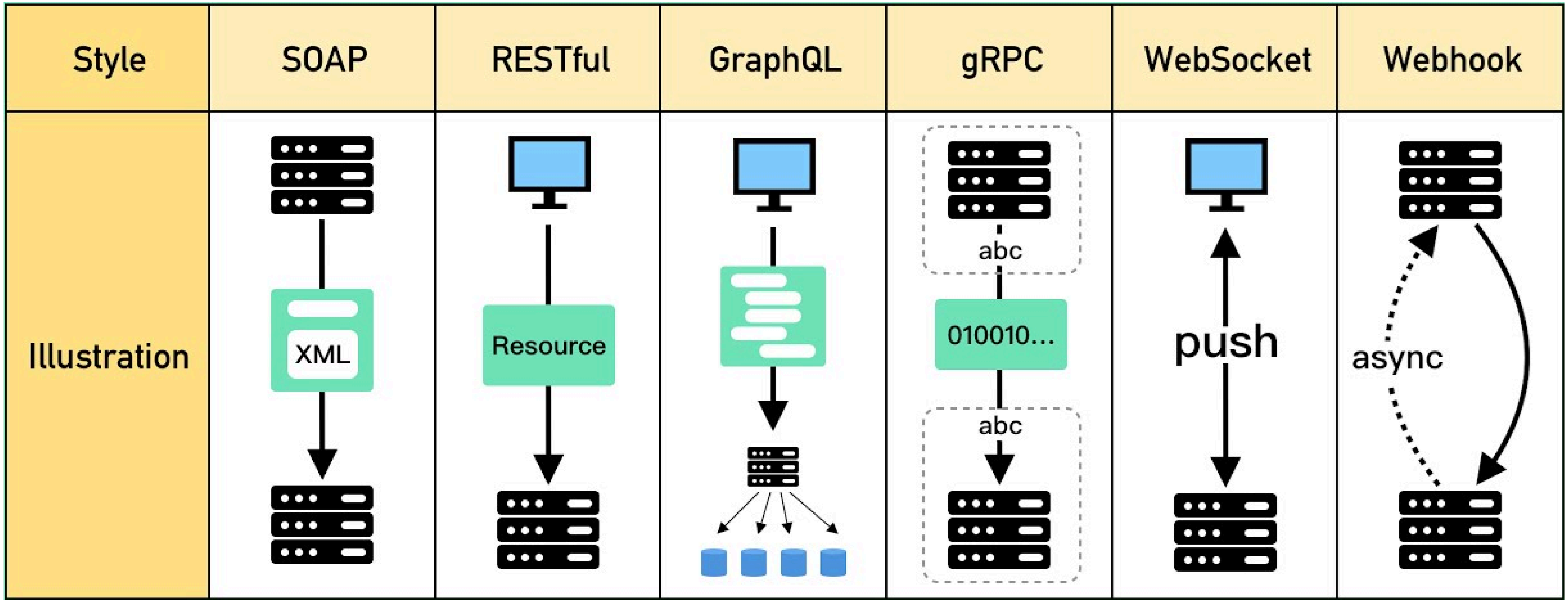
What is API?

Application Programming Interfaces

- **Web service**
- Provides communication and integration between software systems
- Data exchange between components,
- Remote function calls

API Architecture Styles

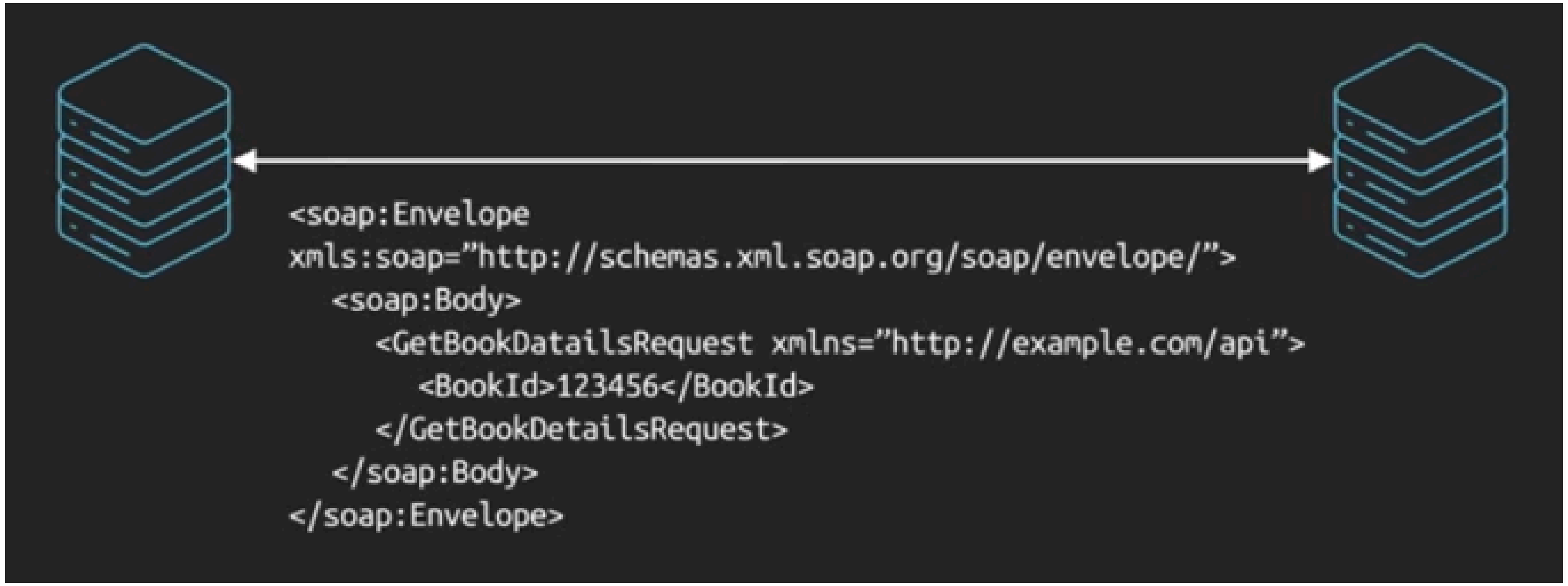
Top 6 most popular styles:



Style	SOAP	RESTful	GraphQL	gRPC	WebSocket	Webhook
Illustration	XML	Resource		abc 010010... abc	push	async

SOAP

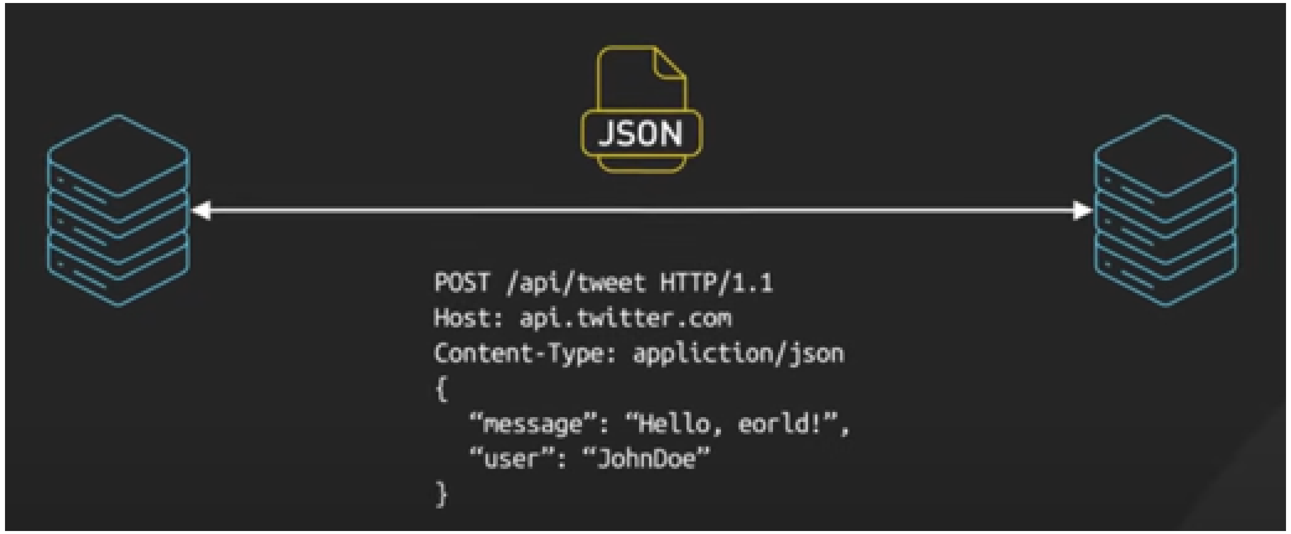
- **Simple Object Access Protocol**
- XML-based for enterprise application, mature and comprehensive
- Commonly used in financial services



```
<soap:Envelope
xmls:soap=”http://schemas.xml.soap.org/soap/envelope/”>
  <soap:Body>
    <GetBookDatailsRequest xmlns=”http://example.com/api”>
      <BookId>123456</BookId>
    </GetBookDetailsRequest>
  </soap:Body>
</soap:Envelope>
```

RESTful

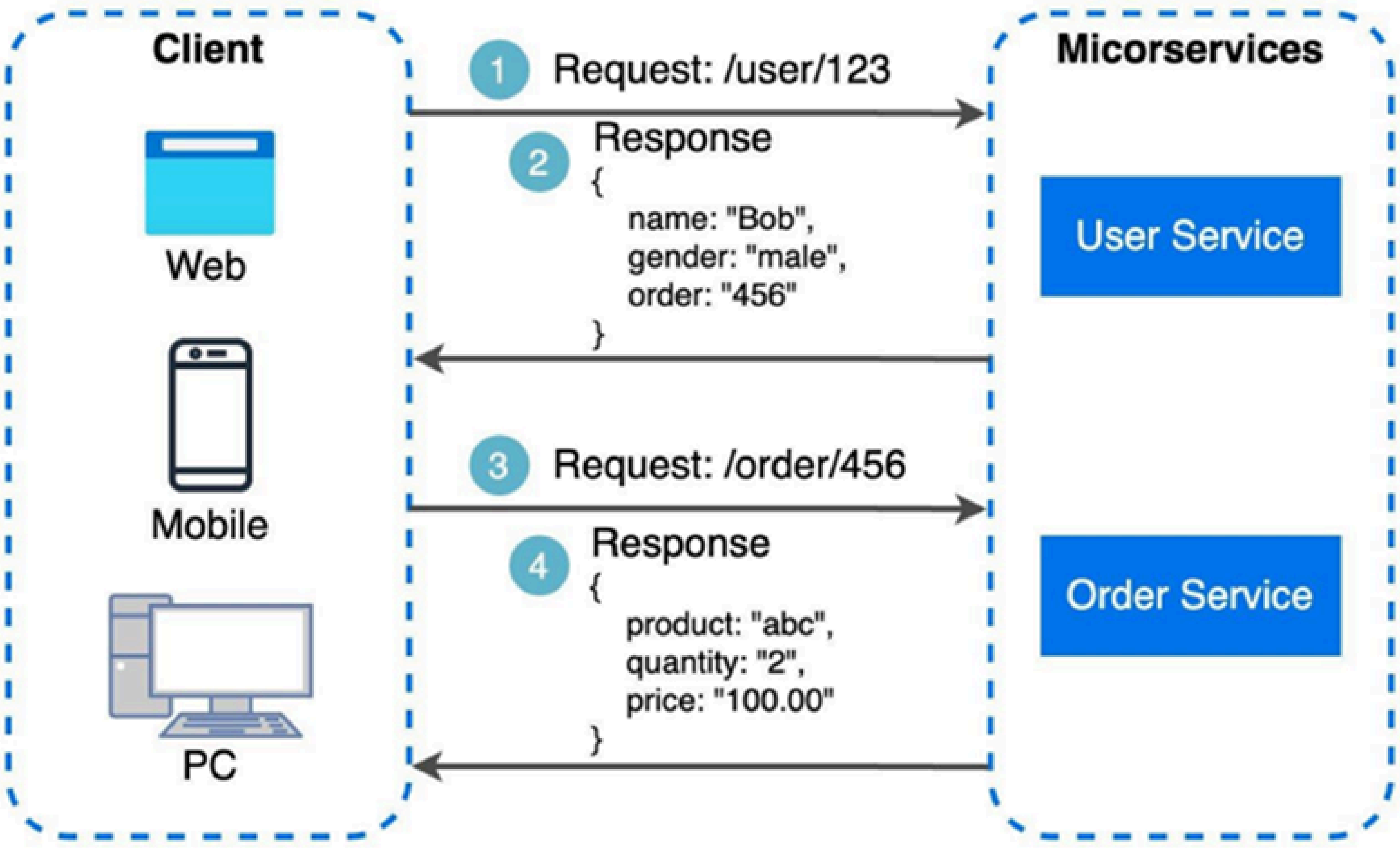
- **REpresentational State Transfer**
- Build on top of `HTTP methods`, popular and easy to implement
- Resource-based, rely on multiple endpoints, return fixed data structures
- Not suitable for `real-time` applications or highly connected model



```
POST /api/tweet HTTP/1.1
Host: api.twitter.com
Content-Type: appliction/json
{
  “message”: “Hello, eorld!”,
  “user”: “JohnDoe”
}
```

{ REST }

RESTful Example



Client

Web

Mobile

PC

1 Request: /user/123

2 Response

```
{
    name: "Bob",
    gender: "male",
    order: "456"
}
```

3 Request: /order/456

4 Response

```
{
    product: "abc",
    quantity: "2",
    price: "100.00"
}
```

Micorservices

User Service

Order Service

RESTful: E-Commerce

Method	Endpoint	Description
GET	/products	fetch a list of products
GET	/users/{userId}	fetch user details
GET	/orders/{orderId}	fetch order details
...		

For `user order history`, multiple requests are required

RESTful: Social Media Platform

Method	Endpoint	Description
GET	/users/{userId}/posts	fetch a user's posts
GET	/users/{userId}/followers	fetch a user's followers
GET	/posts/{postId}/comments	fetch comments on a posts
...		

Each endpoint returns a fixed set of data, requiring multiple requests to gather all necessary information

RESTful

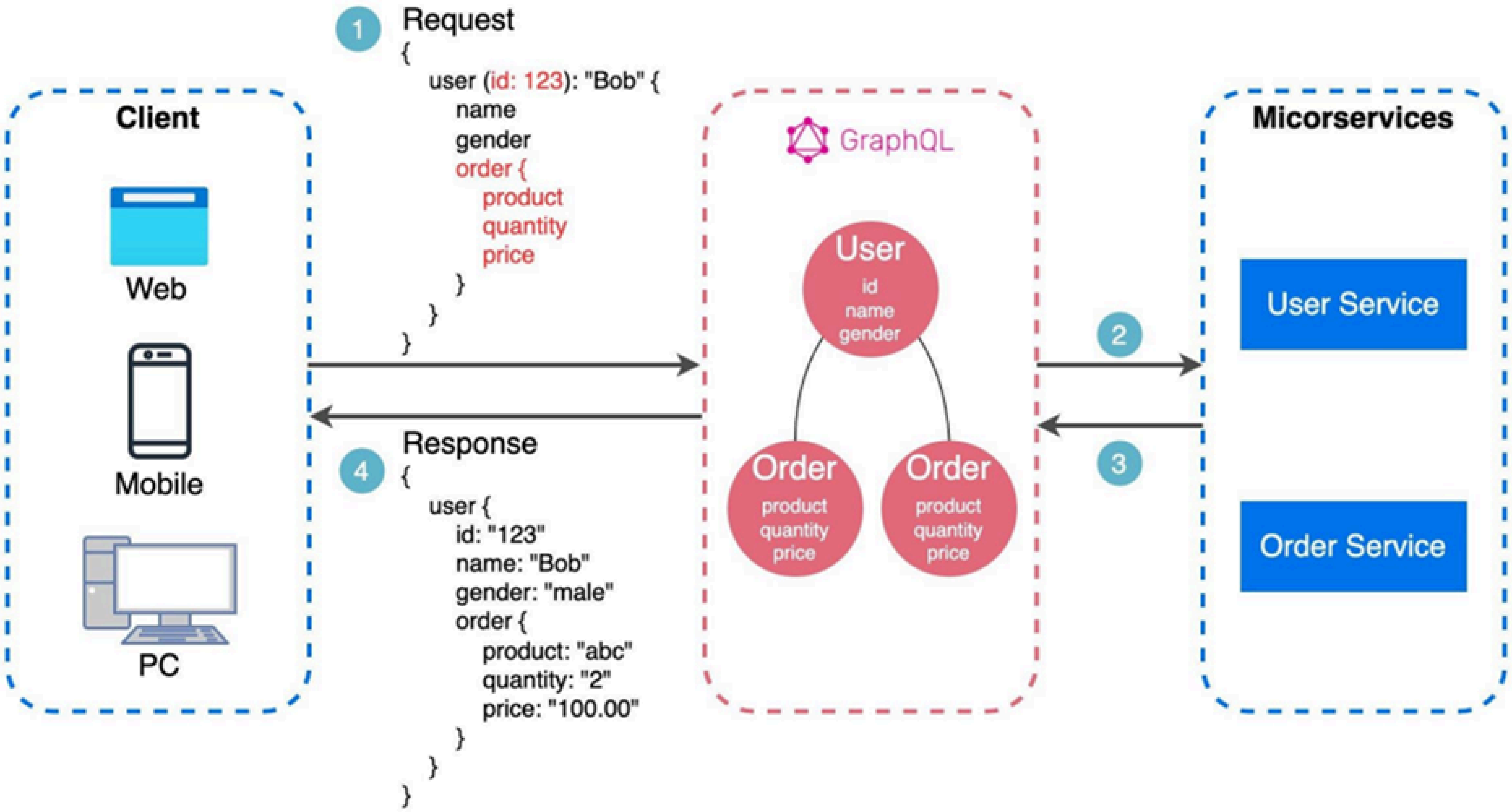
Use RESTful API when:

- API is simple and `CRUD-based`
- Want to use `HTTP caching`
- Building `public APIs` or microservices
- Prefer `minimal tooling` or need API to support older systems

GraphQL

- `Architecture` and `query language`
- Allows clients to specify exact data they need (no `over-/under-fetching` data)
- Strong typed API (predefined `schema`)
- Operates through a single endpoint (flexible and efficient)
- **Hard** to optimized `database query` and `cache`

GraphQL Example



Request

```
{
  user (id: 123): "Bob" {
    name
    gender
    order {
      product
      quantity
      price
    }
  }
}
```

Response

```
{
  user {
    id: "123"
    name: "Bob"
    gender: "male"
    order {
      product: "abc"
      quantity: "2"
      price: "100.00"
    }
  }
}
```

GraphQL: E-Commerce

```
query GetUserOrders {
  user(id: "userId") {
    id
    name
    gender
    orders {
      id
      total
      items {
        product
        quantity
        price
      }
    }
  }
}
```

GraphQL: Social Media

```
query GetUserPosts {
  user(id: "userId") {
    id
    name
    posts {
      id
      content
      comments {
        id
        author {
          id
          name
        }
        content
      }
    }
  }
}
```

GraphQL

Use GraphQL when:

- Frontend needs custom or `nested data structure`
- Want to reduce the `number of API calls`
- Building and `SPA` or `mobile app` with dynamic UI
- Need `flexibility` and scalability

GraphQL vs. RESTful Comparison

Feature	GraphQL ✅ 🚫 🟡	REST ✅ 🚫 🟡
Data Fetching	✅ Fetches only requested fields (efficient)	🚫 Returns entire resource (over-fetching possible)
Endpoint	✅ Single endpoint (/graphql)	🟡 Multiple endpoints (/users, /orders, etc.)
Performance	✅ Reduces network requests & payload size	🟡 Can require multiple requests for related data
Flexibility	✅ Clients define response structure	🚫 Server dictates response format
Versioning	✅ No need for versioning (schema evolves)	🚫 Requires API versioning (v1, v2, etc.)
Real-time Support	✅ Built-in via subscriptions	🚫 Requires WebSockets or polling
Complexity	🚫 Requires schema & resolver setup	✅ Simpler, follows RESTful principles
Caching	🚫 More complex (no native HTTP caching)	✅ Easier with HTTP methods (GET caching)
Use Case	✅ Best for dynamic frontends & complex queries	✅ Best for simple, resource-based APIs

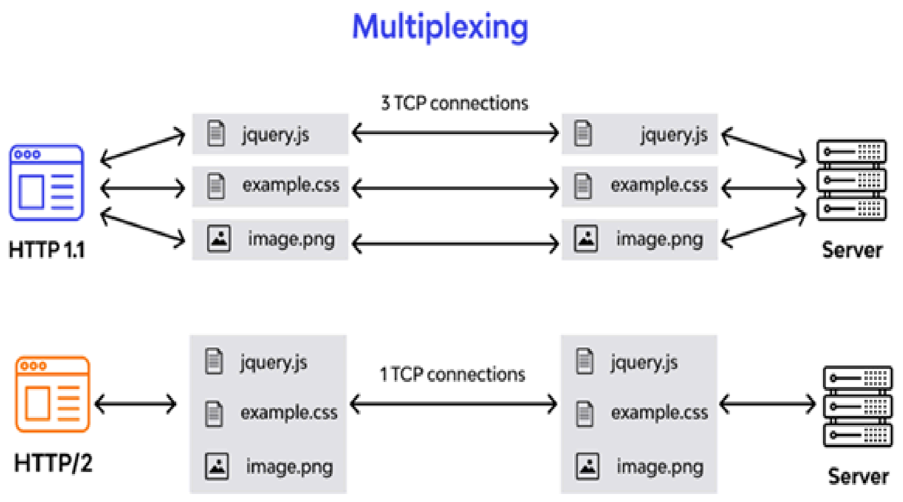
gRPC

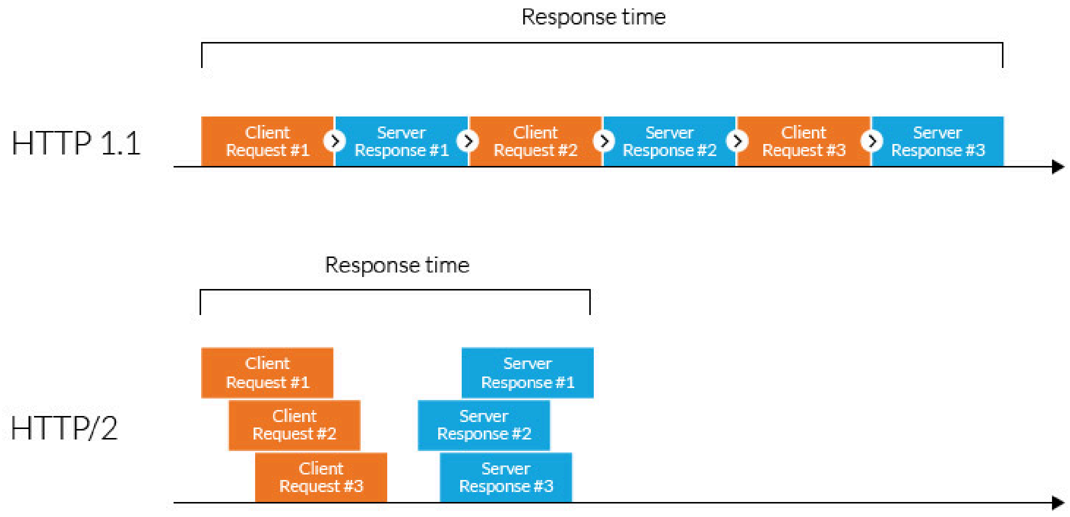
- Open source **Remote Procedure Call** framework created by Google
- High performance for microservices
- Using `Protocol Buffer` (aka. `Protobuf`) which is **binary encoded**

gRPC

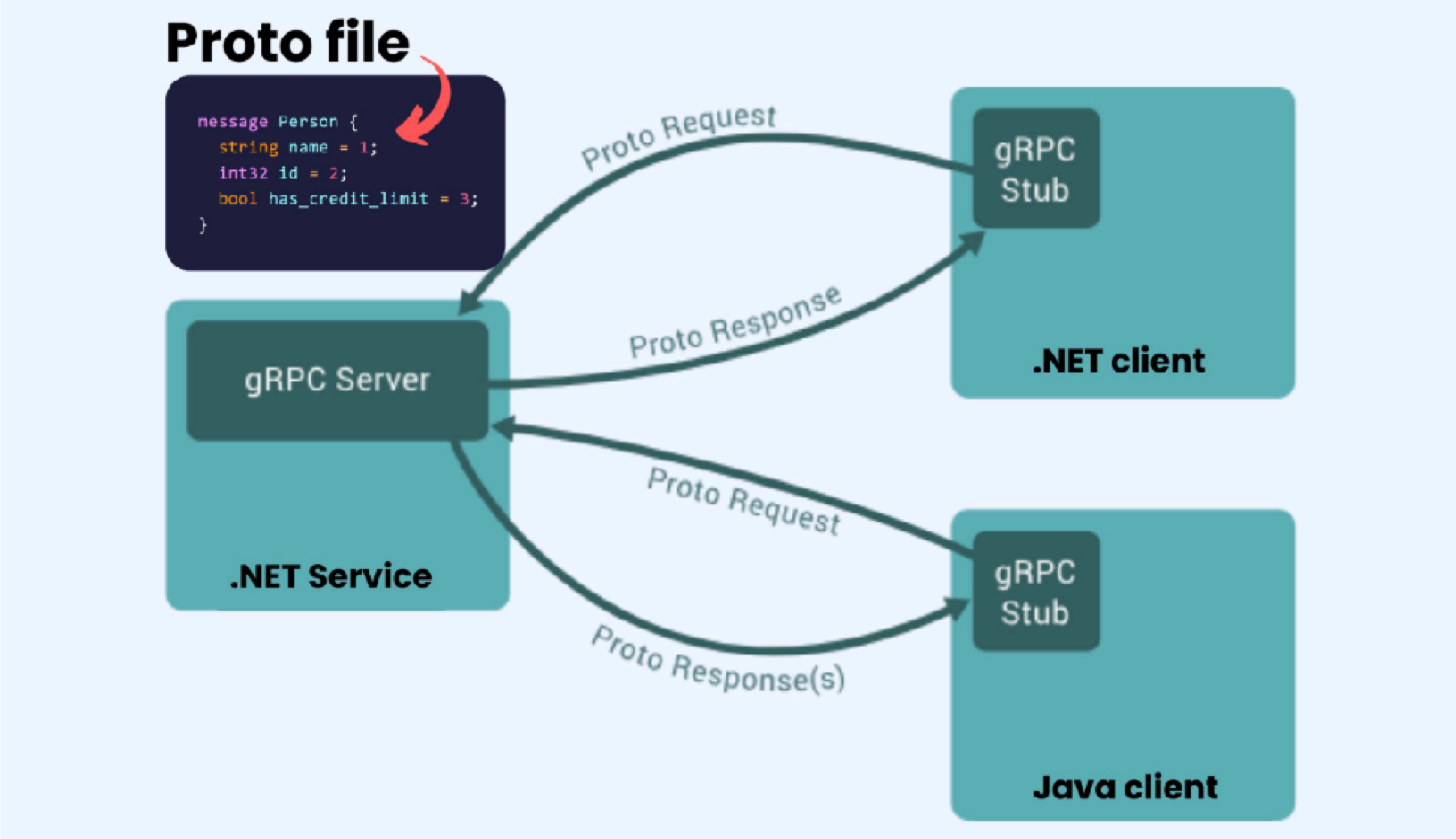
Why is gRPC very fast?

- Send data in binary encoded format (`Protobuf`)
- Run on HTTP/2 protocol



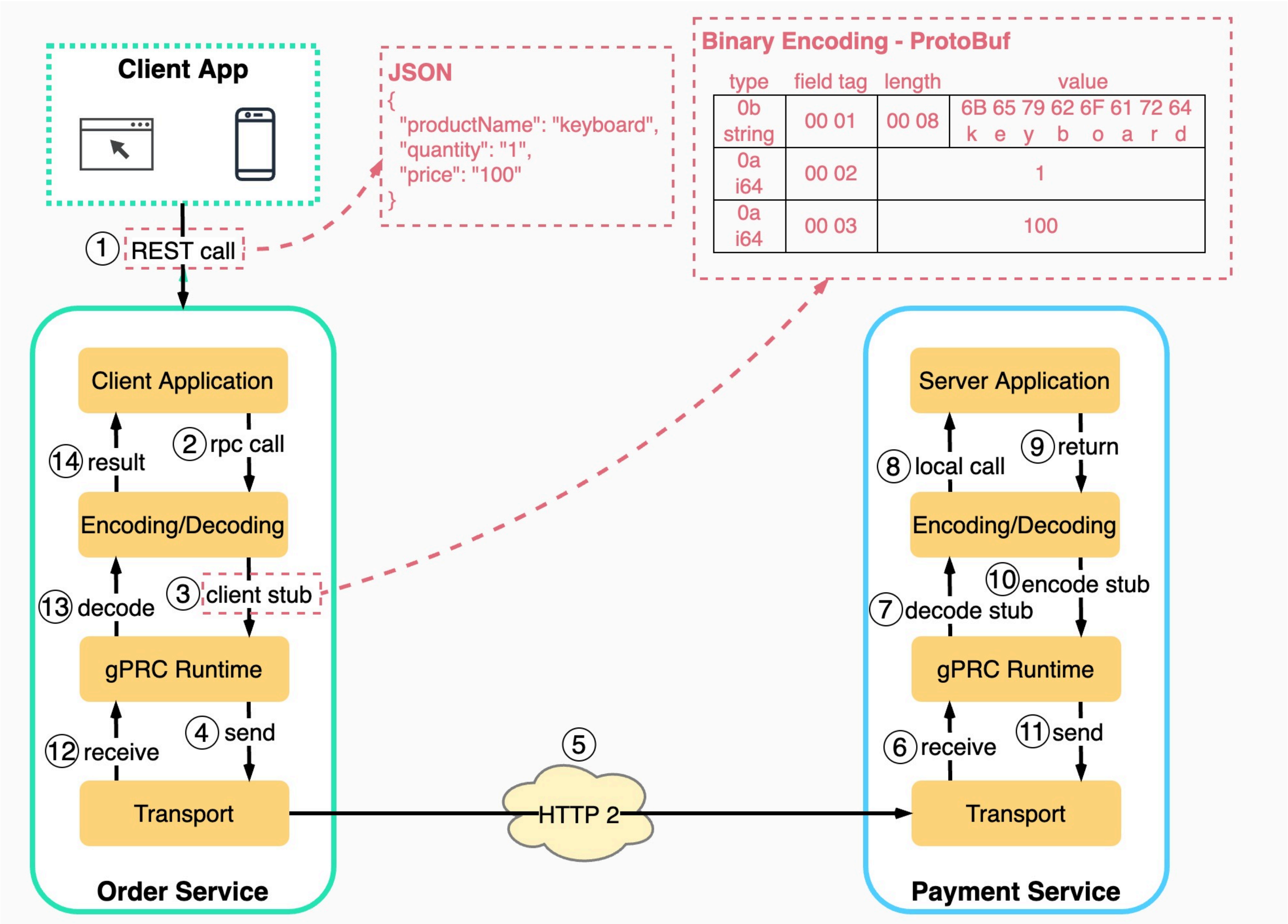


gRPC



```
message Person {
  string name = 1;
  int32 id = 2;
  bool has_credit_limit = 3;
}
```

gRPC - How does it work?



JSON

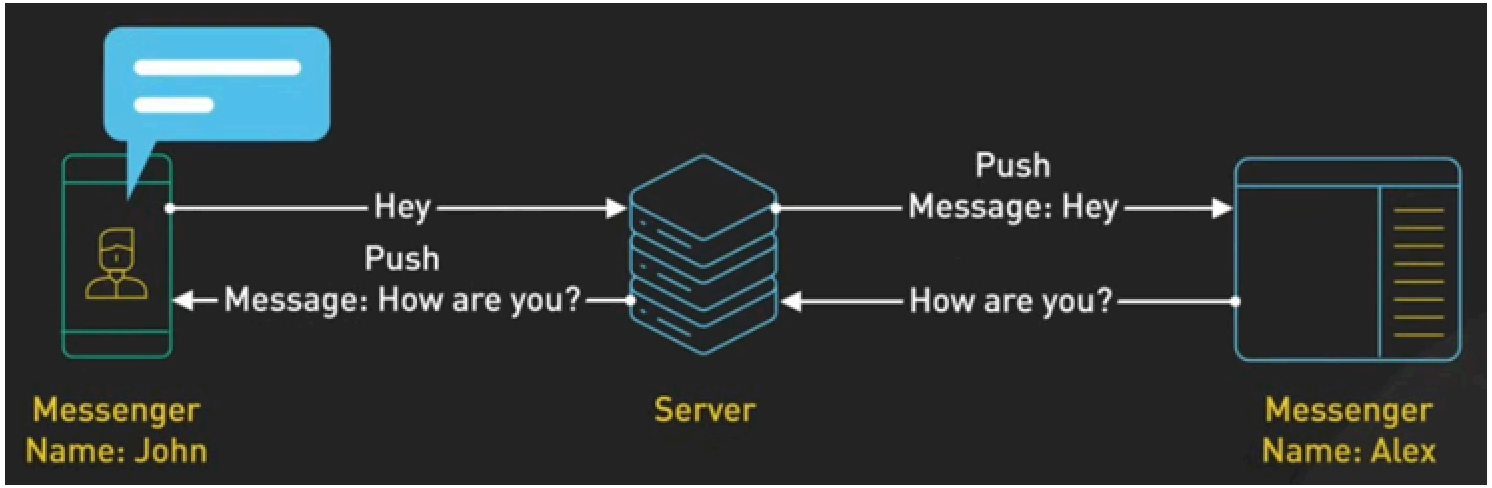
```
{
  "productName": "keyboard",
  "quantity": "1",
  "price": "100"
}
```

Binary Encoding - ProtoBuf

type	field tag	length	value
0b string	00 01	00 08	6B 65 79 62 6F 61 72 64 k e y b o a r d
0a i64	00 02	1	
0a i64	00 03	100	

WebSockets

- `Bi-directional` for `low-latency` data exchange over TCP connection
- `Real-time` and `persistent connection` (for Live chat, real-time gaming)



Websockets

WebSockets

JavaScript/TypeScript library

ws - A simple WebSocket

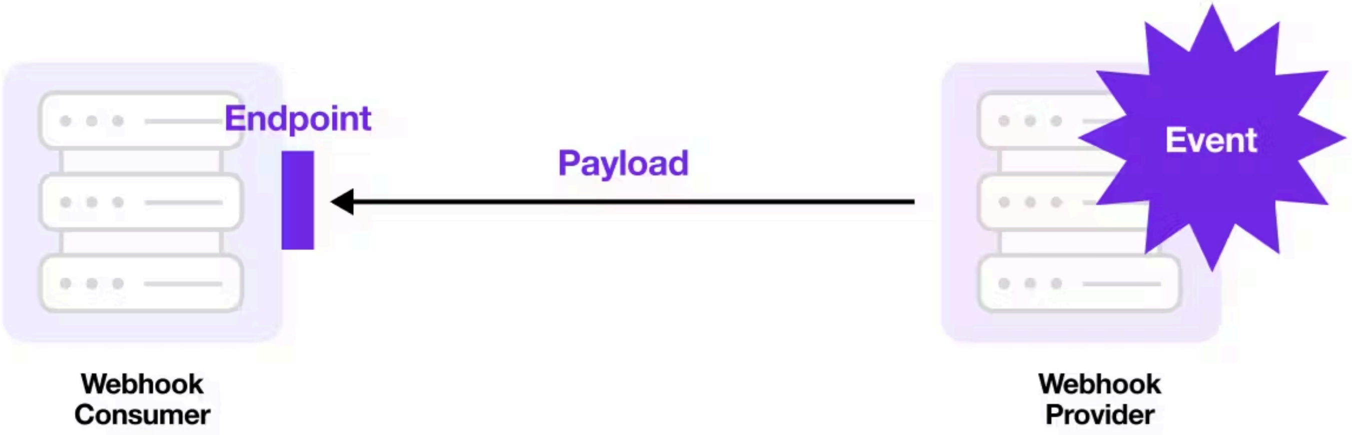
```
npm install ws
npm install --save @types/ws
```

Socket.IO

```
npm install socket.io
npm install --save-dev typescript ts-node @types/socket.io
```

Webhook

- `Asynchronous` for `even-driven` application
- `Webhook consumer` must register with `webhook provider`
    - When certain event occurs, `webhook provider` invokes `HTTP request` to `webhook consumer` at a `certain URL`
    - The `webhook consumer` then handles the request in certain way (e.g., notify user)



Webhook

Webhook

Stripe

- Payment service
- Allow developer to register multipel webhooks, select specific `payment-related` events

GitHub

- Able to notify developer about `code commits`, `pull requests`, and `issues`

Twilio

- Communication service
- Able to notify developers about `SMS` and `voice-related` events

References

- GraphQL Example: Apollo Server | Apollo Client
- REST vs. GraphQL: Choosing the right API for your project
- GraphQL vs. REST: Top 4 advantages & disadvantages
- How to integrate gRPC with React and TypeScript
- Using gRPC in React the Modern Way
- Chat App driven by WebSockets using Socket.IO and TypeScript