Section 1: Introduction to APIs

Introduction to API

Chances are that by being a part of this training, you're already familiar with the word API

You probably already know that API stands for **application programming interface**

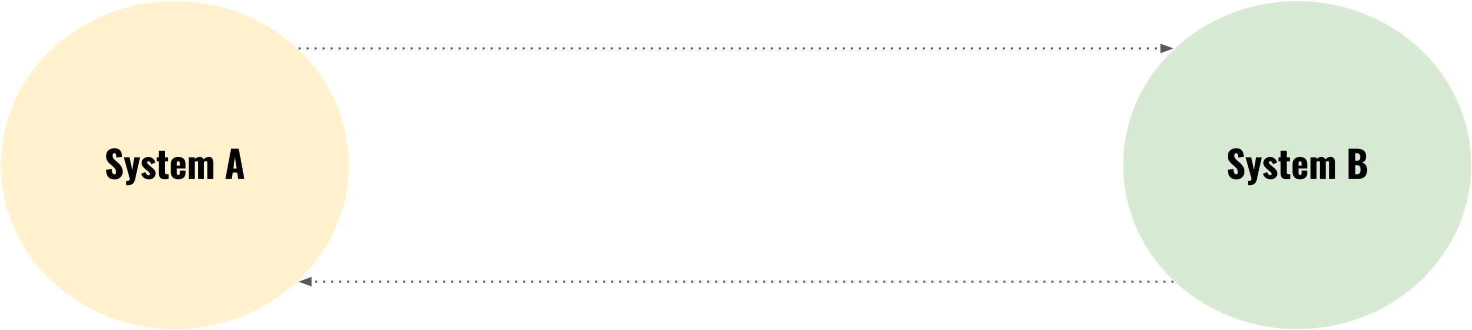
But there is more to it than just definitions!

The focus of this section will be to cover what these basics actually mean in more detail, as well as why they matter

What are APIs?

Just like humans interact with other humans through - a language and interface (mouth and ears)
Humans talk to computers through buttons, mouse, clicks

An API defines the way in which **computer** systems **interact**



What are APIs?

Just like there are ways in which humans can talk to each other (email, call, text, notes)

APIs are the way how computer systems talk to each other

Examples: A code to read a file from disk uses API exposed by Operating System.
The two systems that want to talk are: **your code** and the **OS**

Examples: You want to insert some data in a database, its happens via APIs exposed by DBs
The two systems that want to talk are: **your code** and **DB**

Examples: You upload a photo on Instagram
The two systems that want to talk are: **your browser** and **Instagram Server**

Let’s try to Read a File in our System using Pandas Library

Step 1	Download a CSV file from Internet
Step 2	Install Pandas Library
Step 3	Glance over the simple code to see what we are doing
Step 4	Run the program
Question	Was that an API call?

Code is available here is you want to try

Was that an API Call?

Yes, it was!

But there was no internet, request, response, etc

Whenever two systems are talking to each other, it will be via an API.
This is an important definition - because a lot of people think that API is only when internet is involved, which is kind of true but not the whole story. (More on that later)

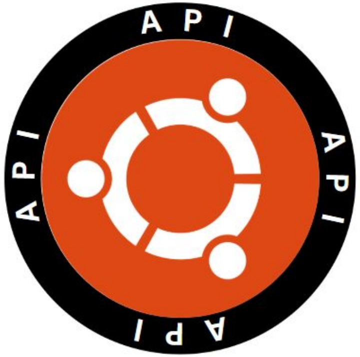
Where exactly was that API call?

Where was the API call?

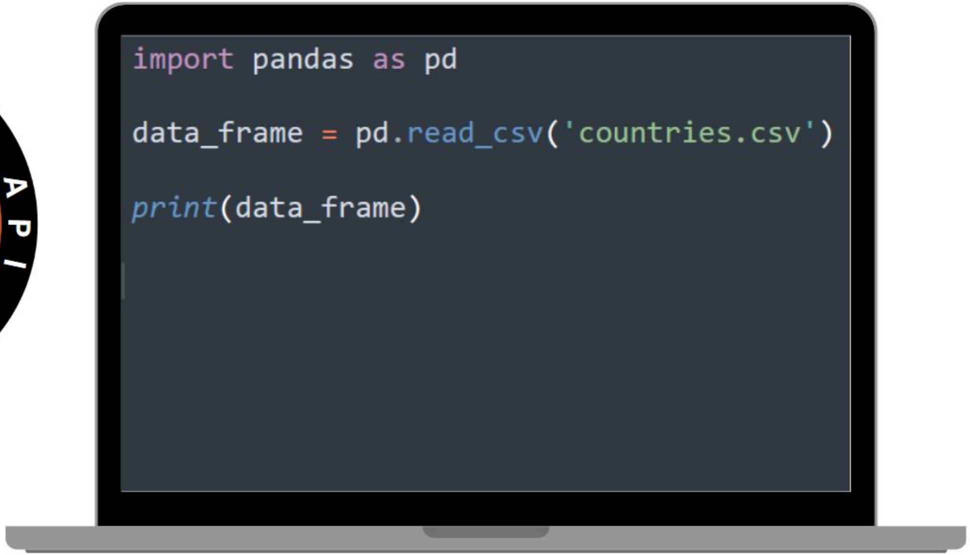
Systems

Remember the definition: APIs are the interfaces using which 2 systems can talk

System A



System B

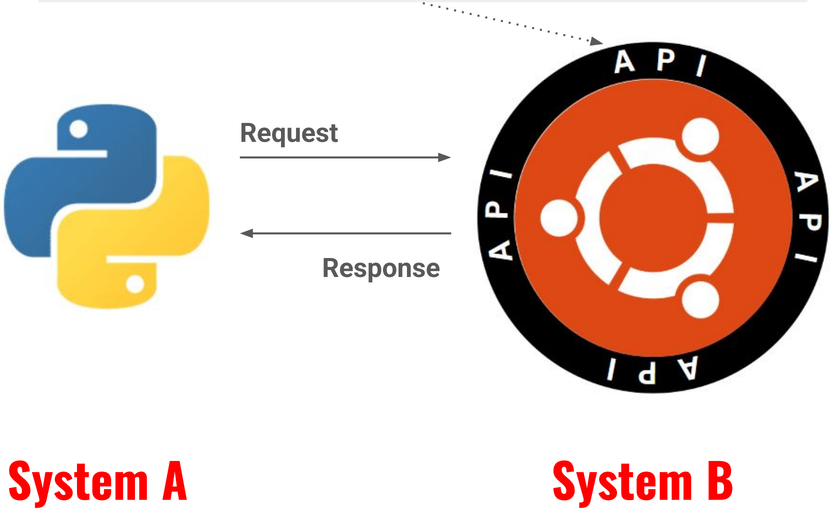


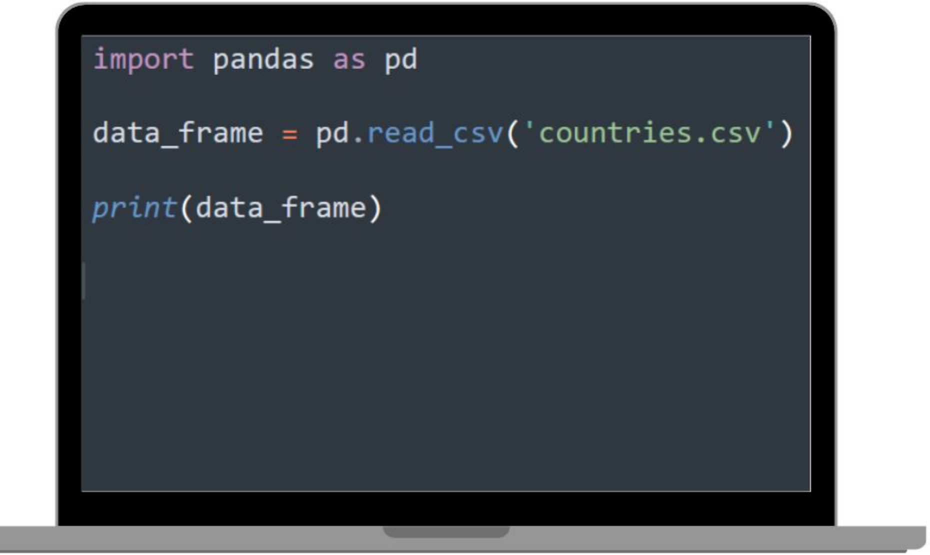
```
import pandas as pd

data_frame = pd.read_csv('countries.csv')

print(data_frame)
```

Where was the API call?

API

Ubuntu OS exposes an API that other systems (codes) can call to interact with files





```
import pandas as pd

data_frame = pd.read_csv('countries.csv')

print(data_frame)
```

API: Application Programming Interface

Application

(Payments, Auth, Social)

A (Application): There is an **application** (a program or service) that provides certain functionalities

API: Application Programming Interface

Program 1

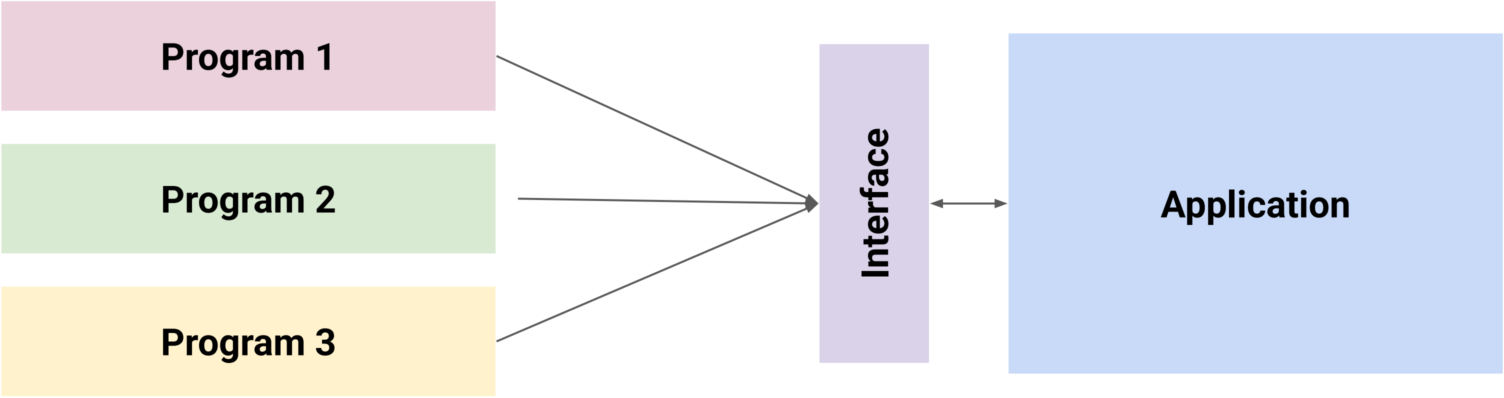
Program 2

Program 3

Application

P (Programming): Other **programs** (or systems) wants to access and use this application's functionalities

API: Application Programming Interface



I (Interface): This access happens via a defined **interface** (a set of rules, protocols, or methods)

But we are interested in specific kind of API: Web API

We can find APIs in the libraries we use from language package managers (e.g., an encryption library that provides a method like function encrypt(input: string): string) and technically in the code we write ourselves, even if it’s never intended for use by anyone else

But there’s one special type of API that is built to be exposed over a **network** and used **remotely** by lots of different people

And it’s these types that are the focus of this training, often called “**web APIs**”

Major Difference Between a Web API and Non-web API

It’s not about internet or web, it’s about network!

A quick side-track

What is network?

Why do we need network?

Network Protocols

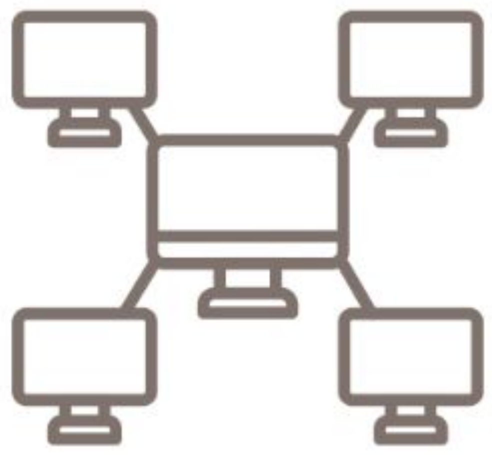
Side Track

Network

What is network in our day to day life?
People who we know. People who we talk to.
People from whom we can get some work done

Network in the world of computers is exactly the same!

Computer networks are systems of interconnected devices that can communicate with each other to get some work done



Side Track

Why do we need network?

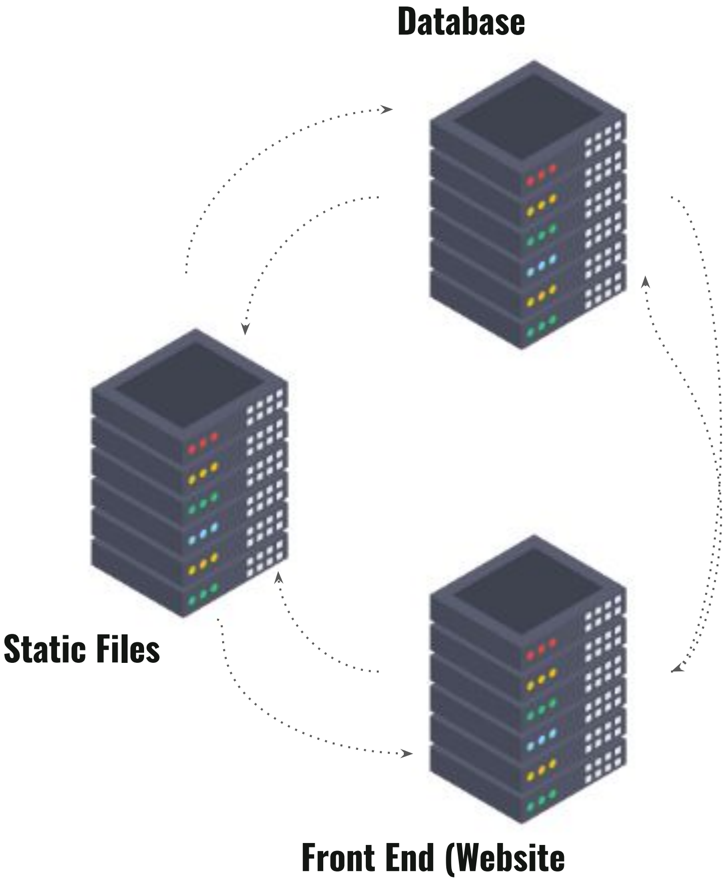
Not everything can be one 1 machine

Network provides a means for system/servers to talk to each other, much like our mobile networks

There are multiple modes of communication like TCP, HTTP, etc

Just like we have different modes like call, text, video call, emails etc and based on our needs we choose one mode

For web APIs - HTTP is the mode we select



Side Track

What is Client and Server?

In the world of computers, anyone who is requesting any data or information is called a **Client**

Anyone who has the data/information/functions is called a Server/machines/computer.

Side Track

What is Client and Server?

In the world of computers, anyone who is requesting any data or information is called a **Client**

Anyone who has the data/information/functions is called a Server/machines/computer

A client can be a person/software/code

Side Track

What is Client and Server?

In the world of computers, anyone who is requesting any data or information is called a **Client**

Anyone who has the data/information/functions is called a Server/machines/computer

A client can be a person/software/code

A server can be a machine, database, web server

Side Track

What is Client and Server?

In the world of computers, anyone who is requesting any data or information is called a **Client**

Anyone who has the data/information/functions is called a Server/machines/computer

A client can be a person/software/code

Clients makes a request

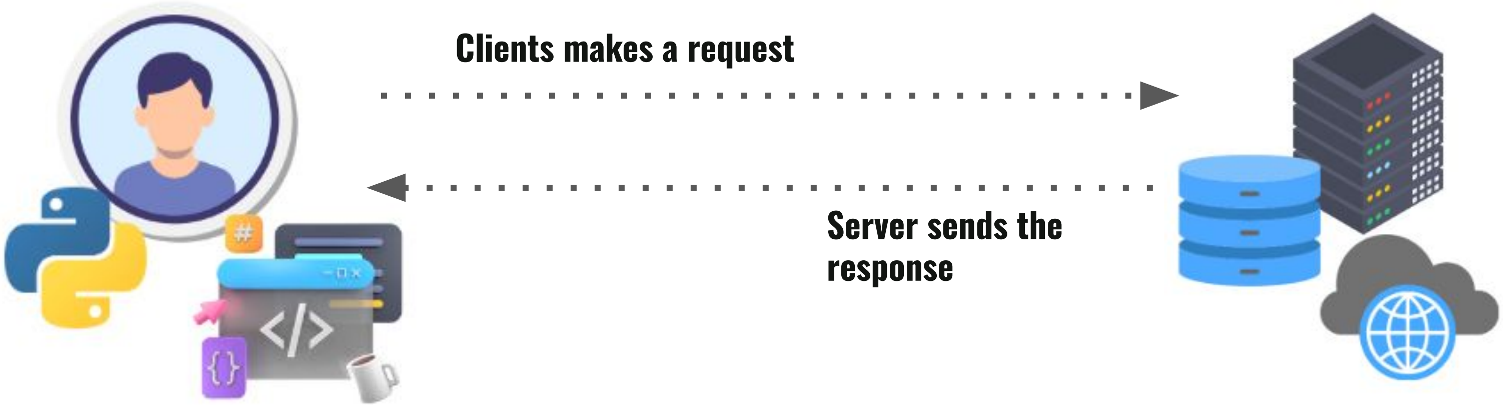
A server can be a machine, database, web server

Side Track

What is Client and Server?

In the world of computers, anyone who is requesting any data or information is called a **Client**

Anyone who has the data/information/functions is called a Server/machines/computer



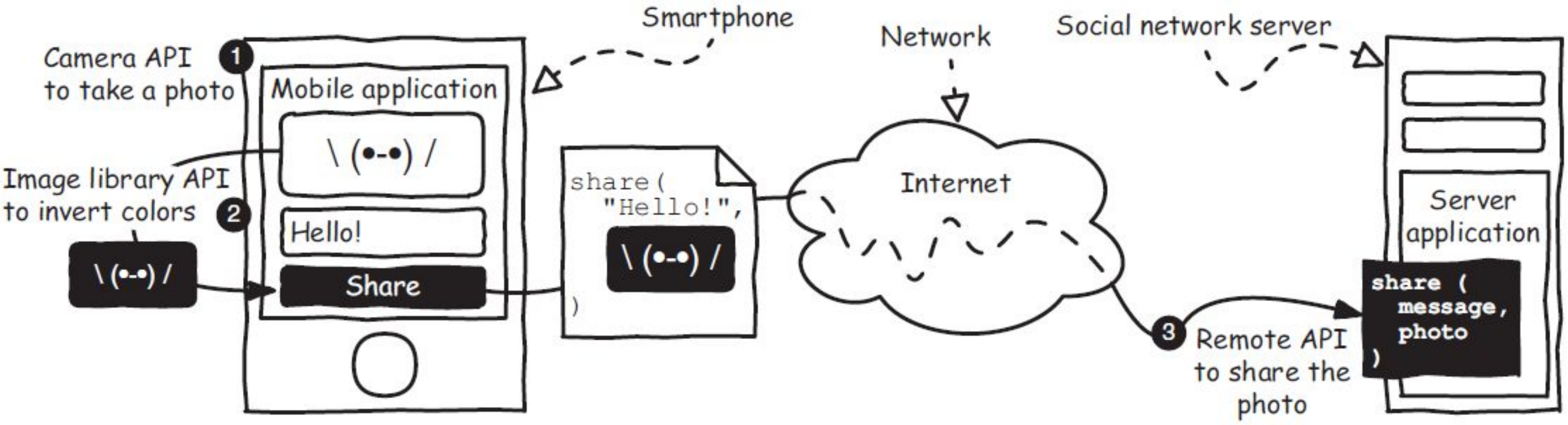
Clients makes a request

Server sends the response

A client can be a person/software/code

A server can be a machine, database, web server

APIs are everywhere: Some are Web and some are non-Web



Non Web APIs
(Operate within network and don’t talk over HTTP)

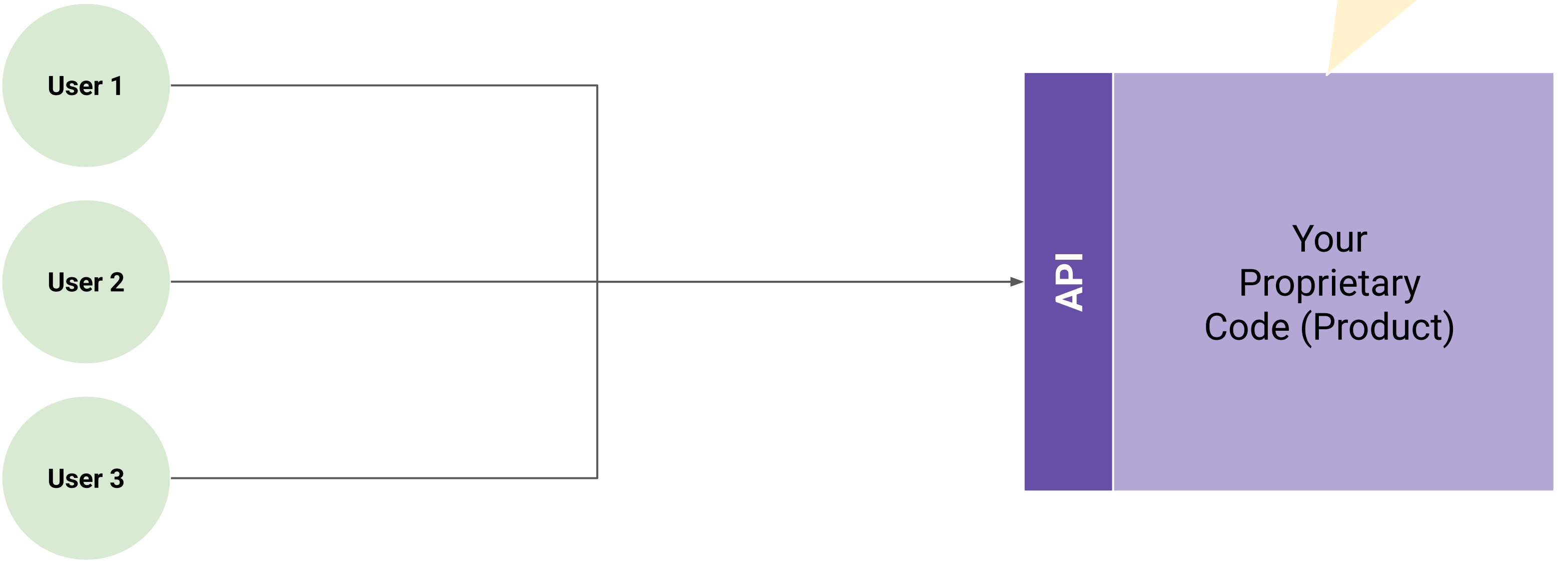
Network
(Not necessarily internet)

Web APIs
(Operate over network and talk over HTTP)

Motivation Behind Web APIs

Why has Web APIs become so popular in the last 10 years?

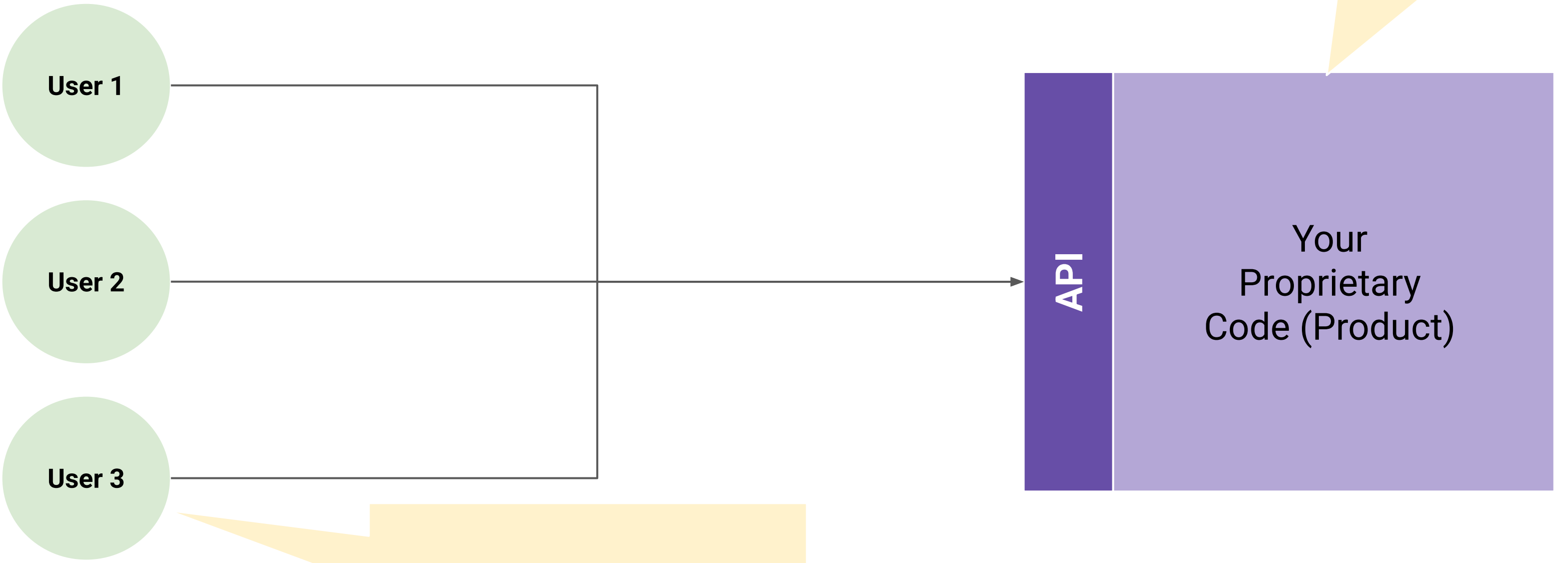
Idea behind creating a Web API



You have developed something cool!

Proprietary product. Unauthorized disclosure, reproduction or other use prohibited

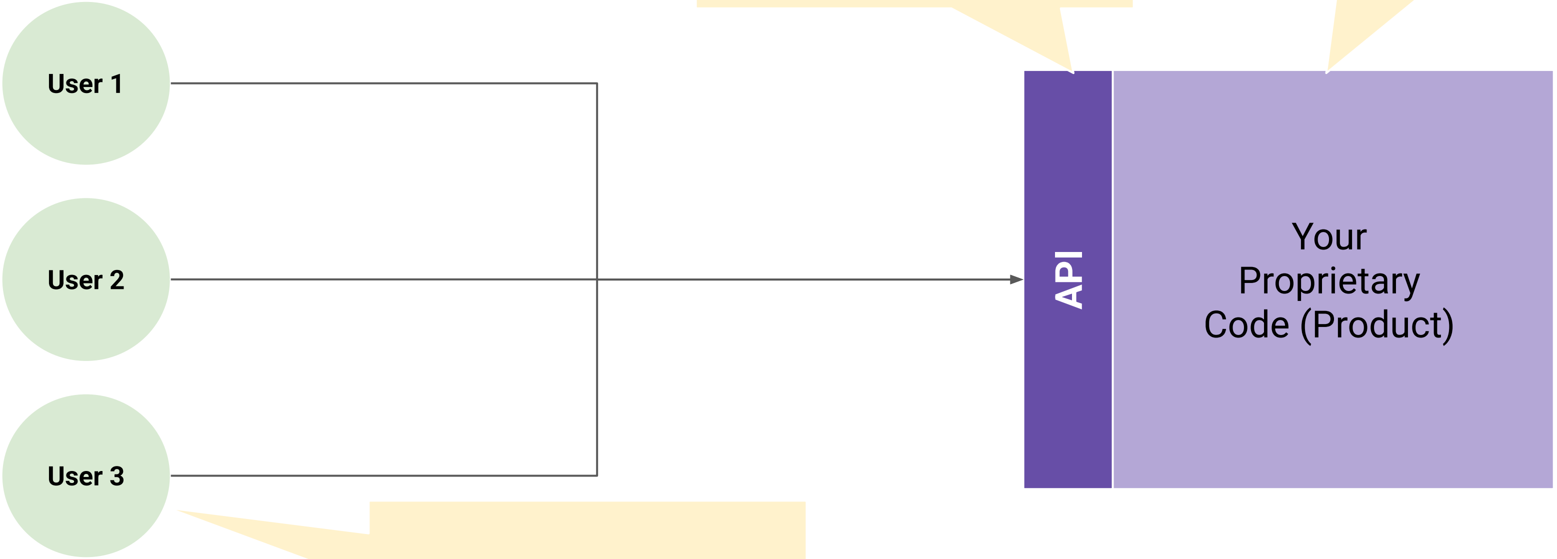
Idea behind creating a Web API



You have developed something cool!

Lot of users are interested in using your product

Idea behind creating a Web API



You have developed something cool!

So you expose your product as an API

Lot of users are interested in using your product

Proprietary product. Unauthorized disclosure, reproduction or other use prohibited

The Rise of Web APIs: Why has this model become so popular?

I hope the basic definition of a web API is clear: One system makes an API call to another system to get some work done (retrieve data, do some calculation, process payment, send message, etc)

So why has Web API become so popular?

Fast Moving Business
Last decade has seen unprecedented growth in the number of startups, mobile apps, web apps - today there is hardly something that cannot be done via app. From ticket booking, to shopping, payment, movies, appointments.

In the cutting throat tech world - **first movers advantage** is one of the biggest advantage. Everyone wants to go to the market first. Which means the time from development to production is very small.

Due to this dearth of time - there has been a mentality shift among organization. It's not about doing everything themselves. It's about doing something that our organization is good at and then leveraging other organizations who are good at other things

The Rise of Web APIs

Fast Moving Business
Take an example of payments - every app needs a payment feature (Swiggy, Bookmyshow, irctc, Flipkart, Ola) - but if all of them go and start building their own payment application - there development time will increase significantly

Instead, they focus on their core competency and leverage other organizations like Razorpay, UPI, PayPal, Stripe etc to **integrate** the payment feature

There are 100s of examples where one company concentrate on their product while leveraging other companies product via an **API**

- **Geolocation and Maps:** Ola, Uber, Lyft uses Google and Apple maps product as an API
- **Email and communication:** Many companies uses Twilio, MSG91, SendGrid for OTP and Email notificatio**ns**
- **Authentication:** Almost all apps use Authentication API of Google, Facebook, Twitter
- **AI**: Companies have started using LLM (Chatgpt), IBM Watson (speech to text) etx

Imagine you have an idea which itself is complex to implement. But now you have to take care of other things like notifications, payments, authentication, etc.
That's the pain that web APIs have removed from our lives and exactly why it has become backbone of modern technology stack

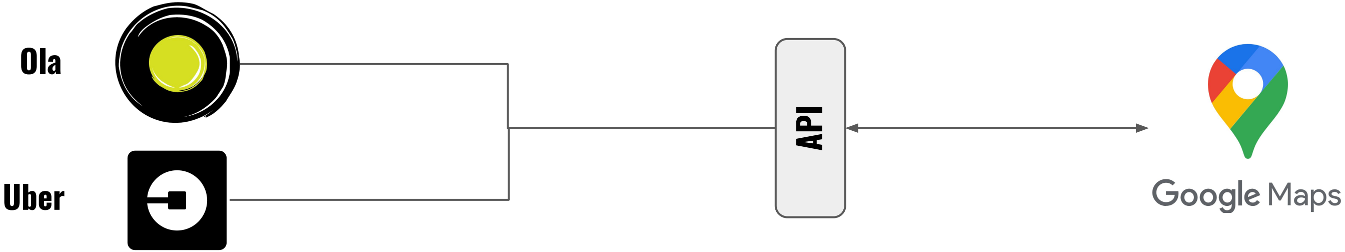
The Rise of Web APIs

Interoperability and Integration (Hidden Complexity)
You can argue that the previous point that we discussed - can those features not be shipped (sold) as a code or software suite?

Theoretically yes, but imagine the complexity that will fall on the products to be used. Today there are unlimited forms of clients and devices - servers, desktops, mobile, different OS, etc. They will have to ship in all the platforms.

But with API that complexity goes away. In Fact that is the single most important thing that APIs do. The client and server does not care about any implementation (language) details. We will see how later.

Multiple apps - one API. That’s the fundamental of API design



Let’s Look At Some Common APIs Use Cases

<<5:00>>

TakeMyTrip

You are impressed by MakeMyTrip business model and you want to implement your own

The first thing you would need is a way to collect details from all aviation orgs like Indigo, Air India, Vistara, etc

And how do you do that?

All of them have an API exposed that you can call and get the details in response

Q: Who is the client?

Q: Who is the server?

Q: What does request look like?

Q: What about response?

TakeMyTrip

You are impressed by MakeMyTrip business model and you want to implement your own

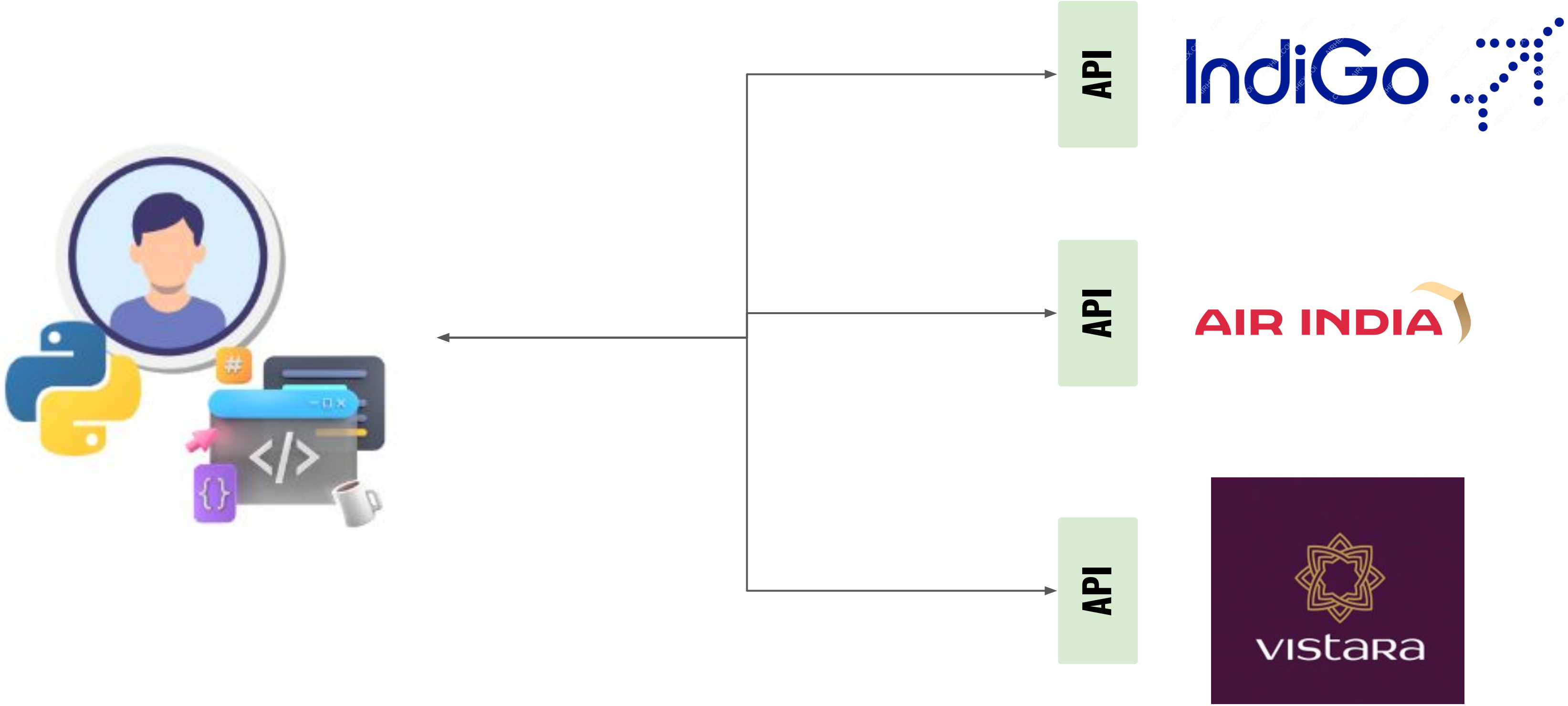
The first thing you would need is a way to collect details from all aviation orgs like Indigo, Air India, Vistara, etc

And how do you do that?

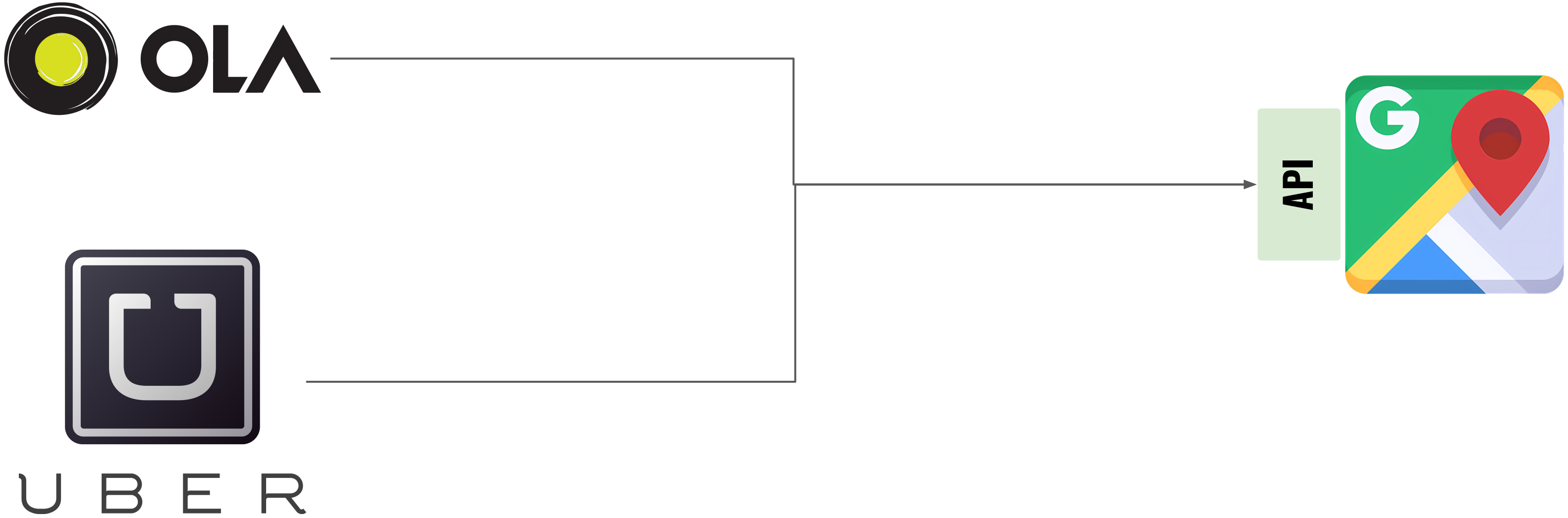
All of them have an API exposed that you can call and get the details in response

Q: Who is the client?	**Q: Who is the server?**	**Q: What does request look like?**	**Q: What about response?**
Q: TakeMyTrip	**Q: Indigo, Vistara, etc**	**Q: date, from, to, others**	**Q: list of flights, timings, cost, etc**

MakeMyTrip



Google/Apple Maps



API

API is Anything as a service!

Geolocation
as service

Social media
as service

Large Language Model
as service

Payments
as service

aws

Infrastructure
as service

zoom

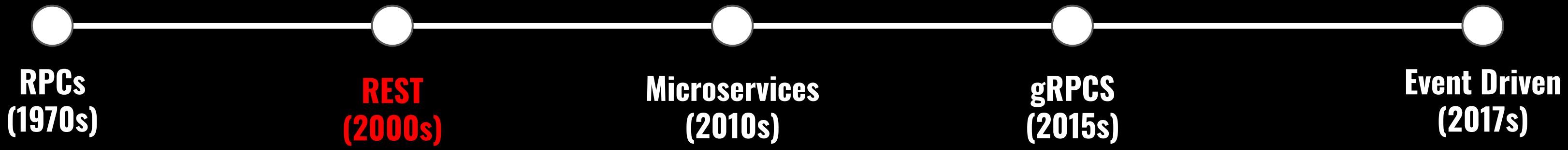
Conferencing
as service

NSE
National Stock Exchange Of India

Trading
as service

History of APIs

A brief time-travel



RPCs (1970s)

REST (2000s)

Microservices (2010s)

gRPCS (2015s)

Event Driven (2017s)

Now that we are sold on the idea that APIs are indeed important, we have to understand that depending on what we are trying to implement - there are different choice of API architecture that we can choose from.

There is no better or worse and there is no right or wrong (it's not like gRPCs are better than REST)
It's a choice we have to make based on requirements and situations

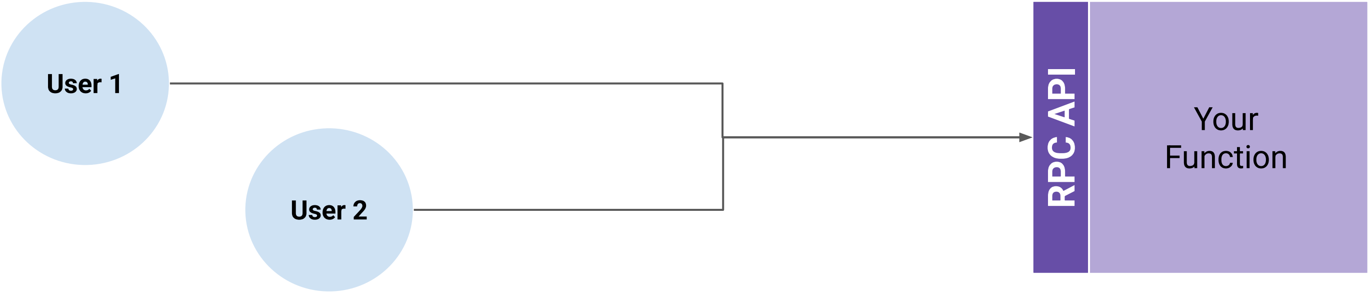
RPCs: First Kind of APIs (1970s)

First Kinds of API ever: RPC (1970s)

I am sure all of you have written a **function** while programming any application

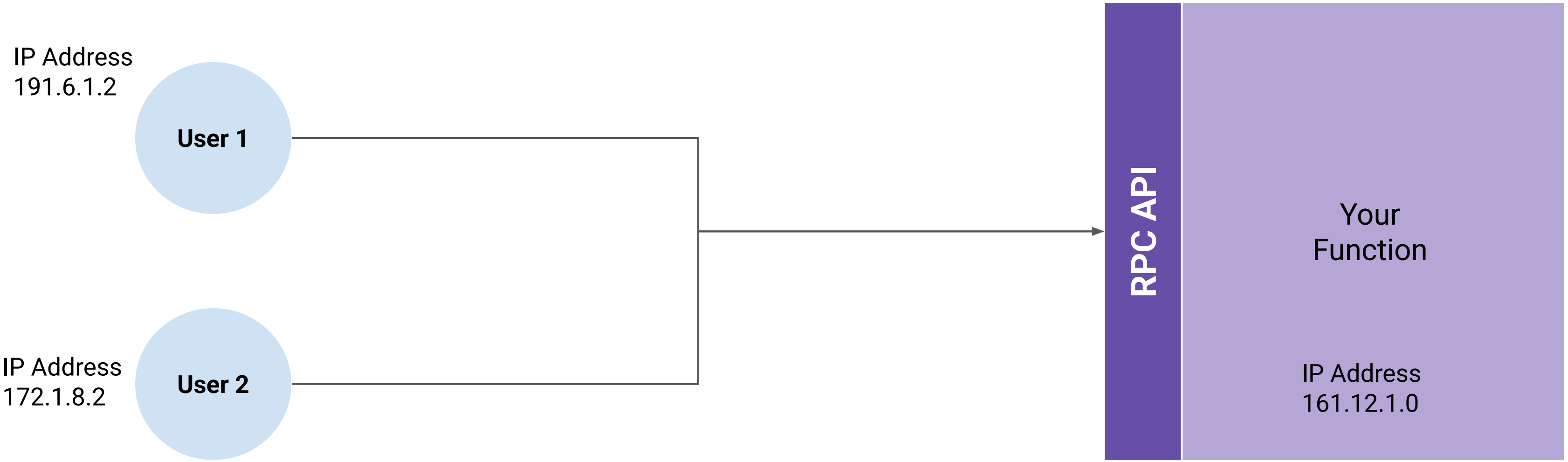
We write a function which takes in some input and produces some output, and then we call (execute) the function when needed

RPCs are exactly like this, with the only change that its others who call (execute) the function and not you!



RPC: A formal introduction

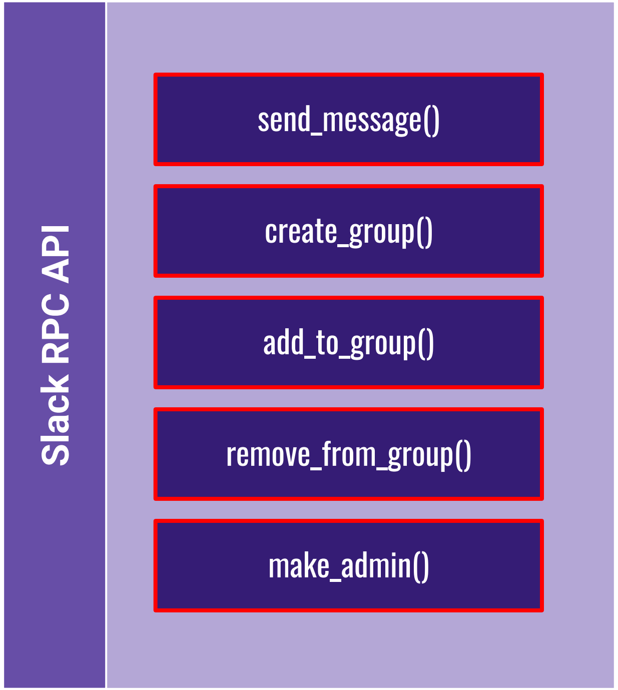
RPC-based APIs execute a unit of code, the procedure, over the network as if it were being executed locally



Proprietary product. Unauthorized disclosure, reproduction or other use prohibited

RPC: A formal introduction

The client is given a list of available procedures (functions) that may be invoked on the server

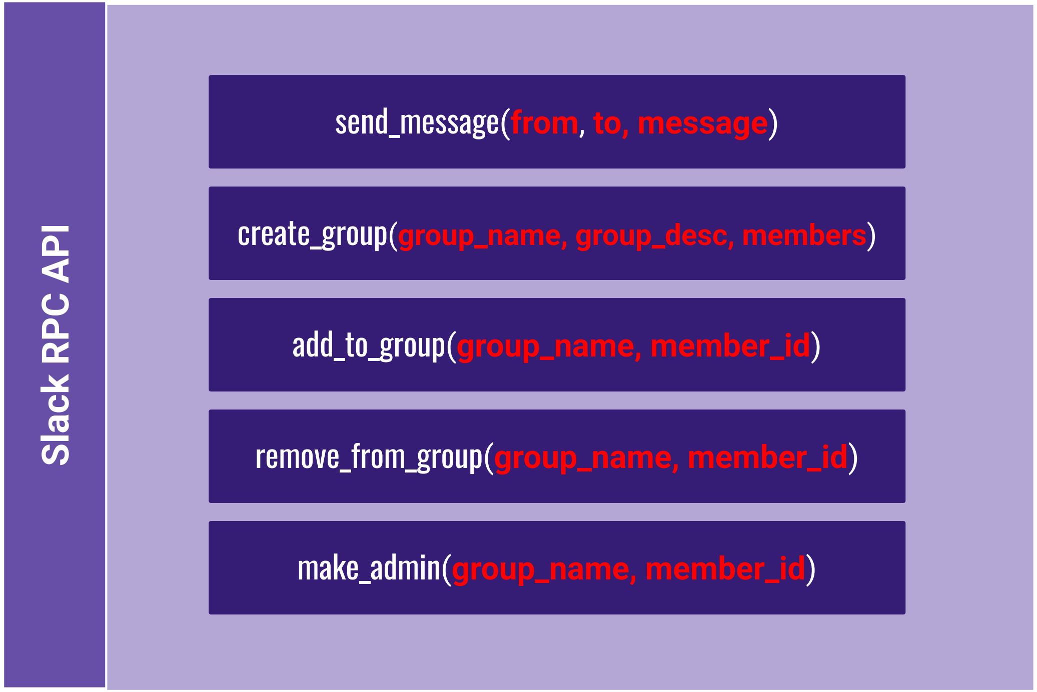


Slack RPC API

- send_message()
- create_group()
- add_to_group()
- remove_from_group()
- make_admin()

RPC: A formal introduction

Each procedure defines a typed and **ordered parameter list** and the structure of a response structure



Slack RPC API

send_message(from, to, message)

create_group(group_name, group_desc, members)

add_to_group(group_name, member_id)

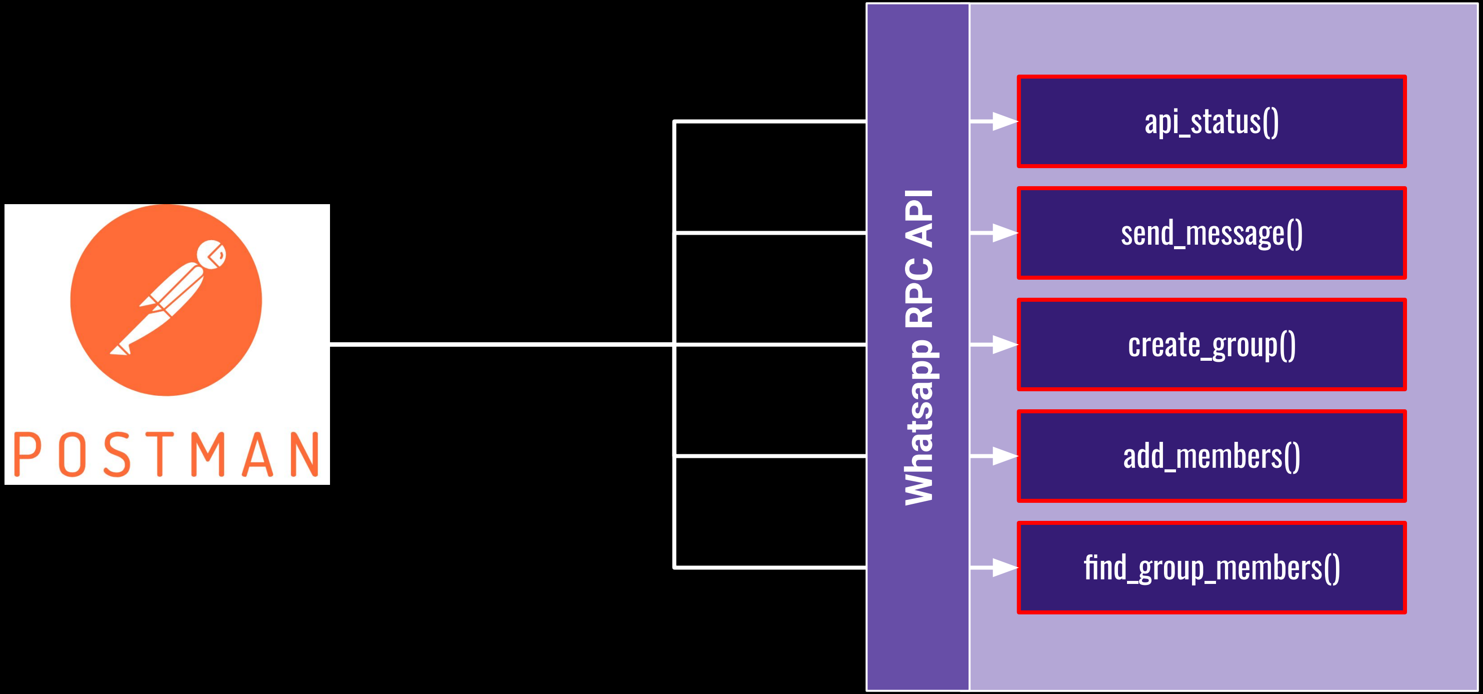
remove_from_group(group_name, member_id)

make_admin(group_name, member_id)

Proprietary product. Unauthorized disclosure, reproduction or other use prohibited

Hands On

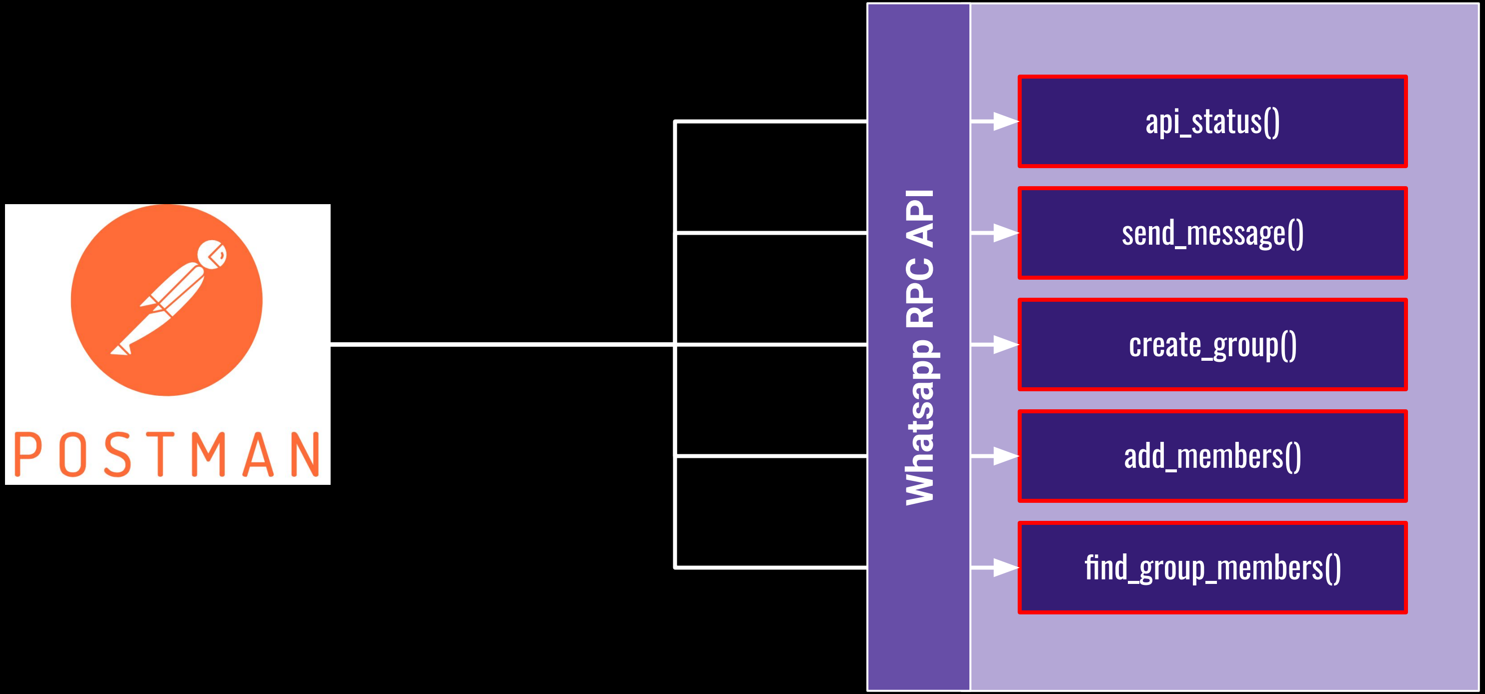
Prerequisite: Ubuntu, Python 3, Postman



Proprietary product. Unauthorized disclosure, reproduction or other use prohibited

Hands On

Let’s try to implement Whatsapp like API in RPC: link



Some Drawbacks of RPCs

The critical aspect of APIs like this (RPCs) is the primary focus on the **actions** being performed

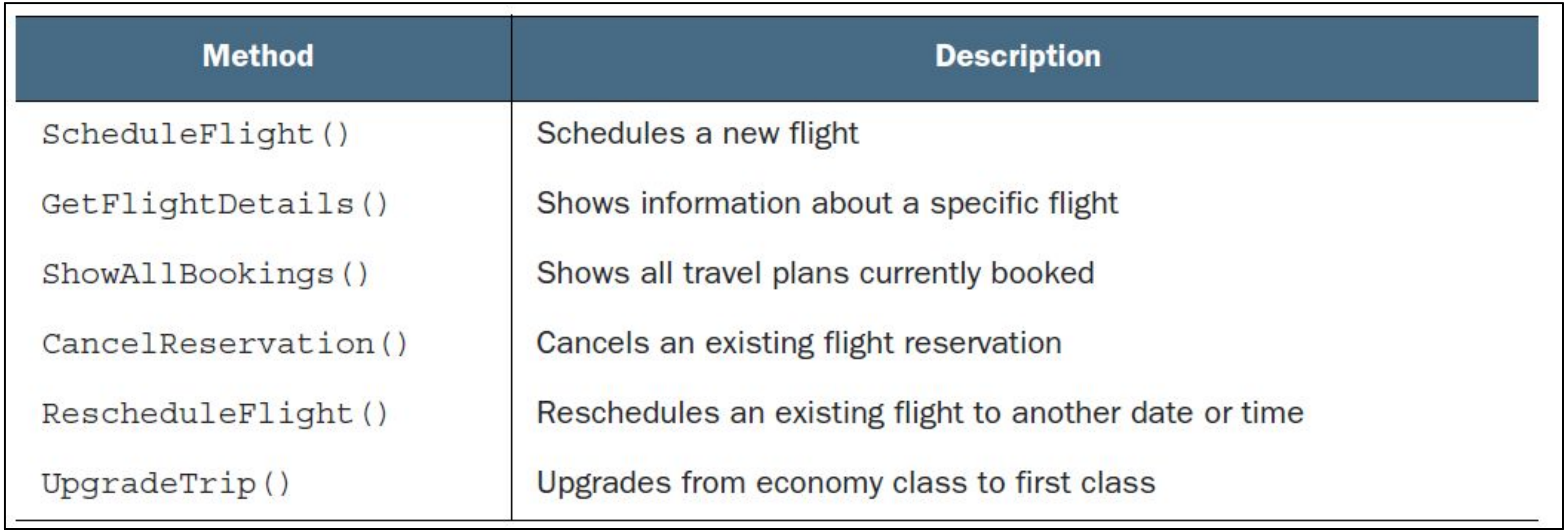
For example, calculating the weather (**predictWeather**(postalCode=...)) or
Encrypting data (**encrypt**(data=...)) or sending an email (**sendEmail**(to=...)), each with an
emphasis on “**doing**” something

One of the major challenge of RPC based APIs is the **long list of actions** that can be done!
This long list makes the process of documenting, sharing and understanding APIs very difficult

The second challenge is - when the list of parameter is very large - it can be tedious to maintain the correct sequence of parameters.
Example: **registerUser**(firs_name, last_name, email, mobile, address, username, password, DOB, gender, occupation, company_name)

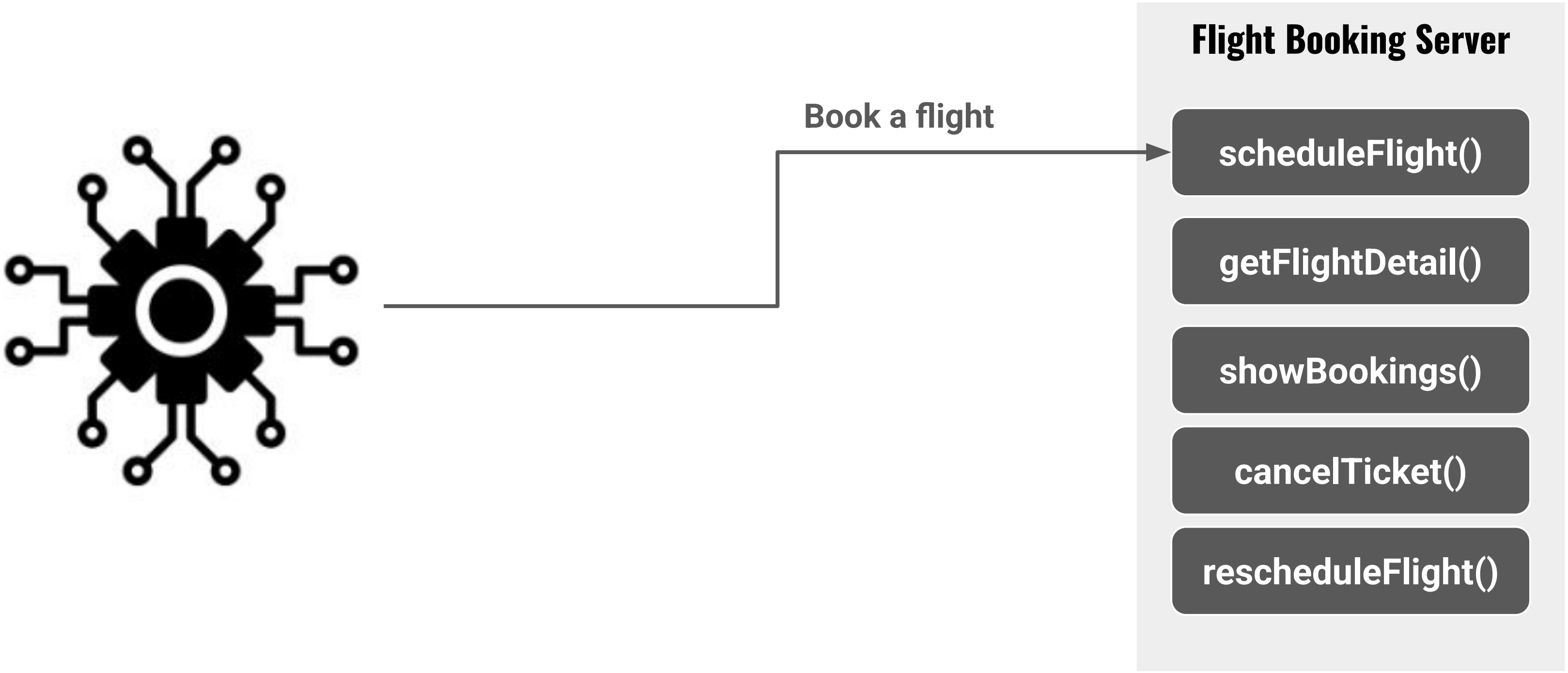
So Many Actions to Remember: Flight Booking API

These are the list of RPC (remote procedure calls) we might want to make for a general flight booking app like MakeMyTrip or Cleartrip

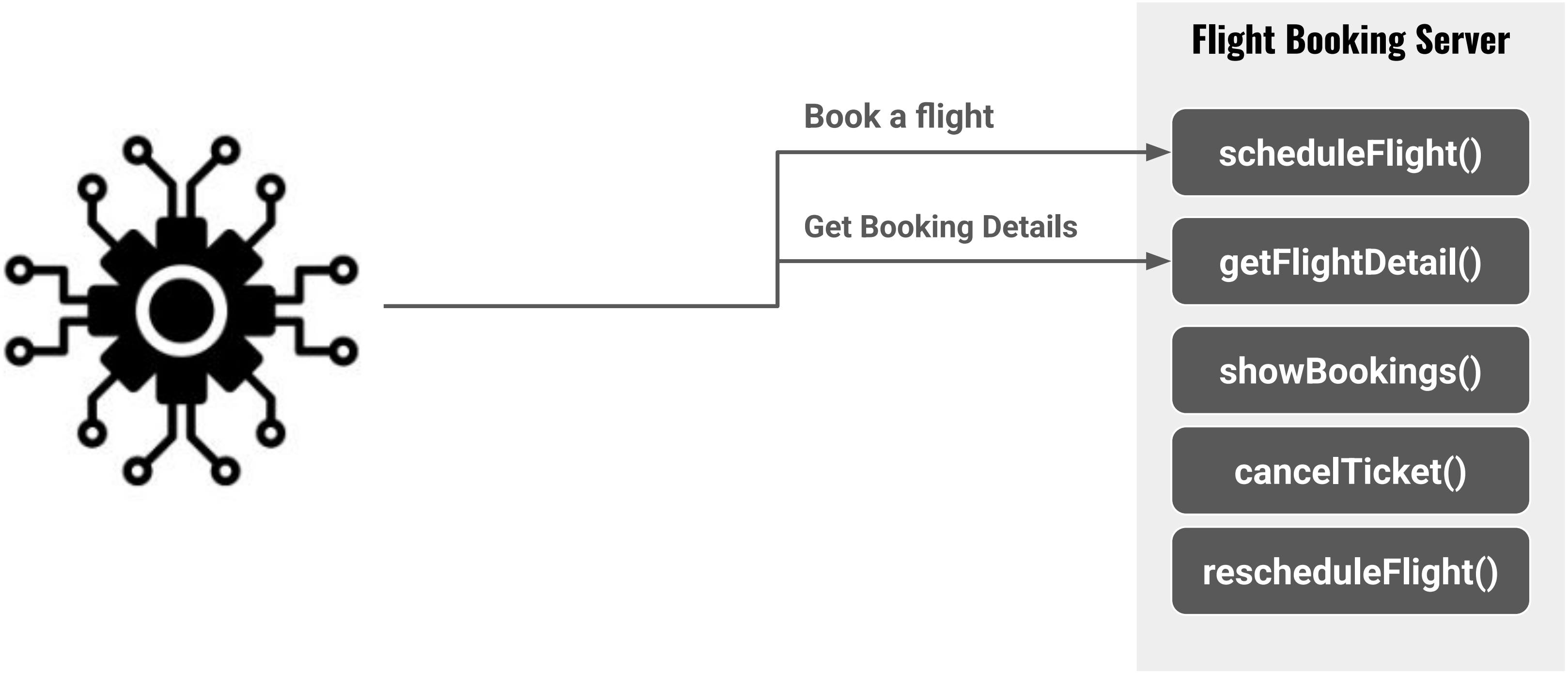


Method	Description
`ScheduleFlight()`	Schedules a new flight
`GetFlightDetails()`	Shows information about a specific flight
`ShowAllBookings()`	Shows all travel plans currently booked
`CancelReservation()`	Cancels an existing flight reservation
`RescheduleFlight()`	Reschedules an existing flight to another date or time
`UpgradeTrip()`	Upgrades from economy class to first class

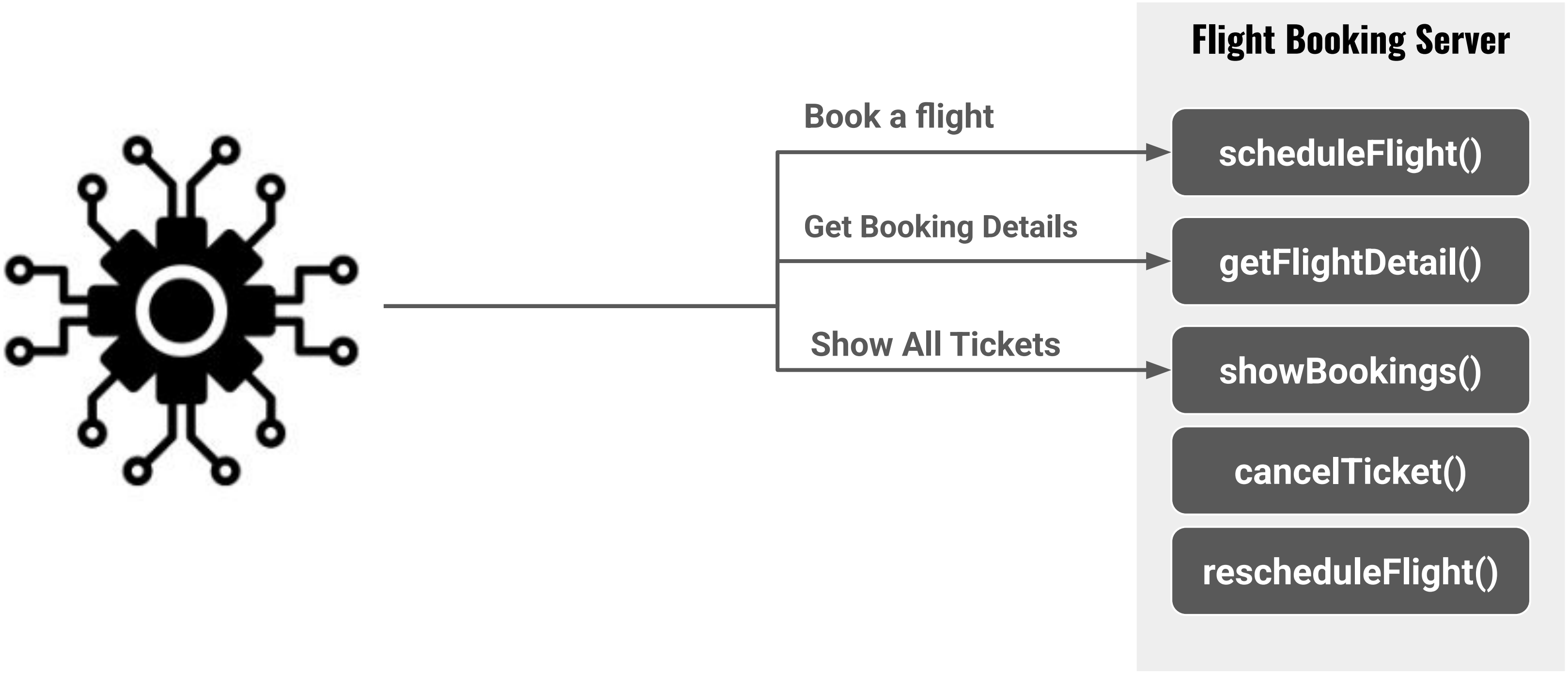
Flight Booking API: scheduleFlight() RPC



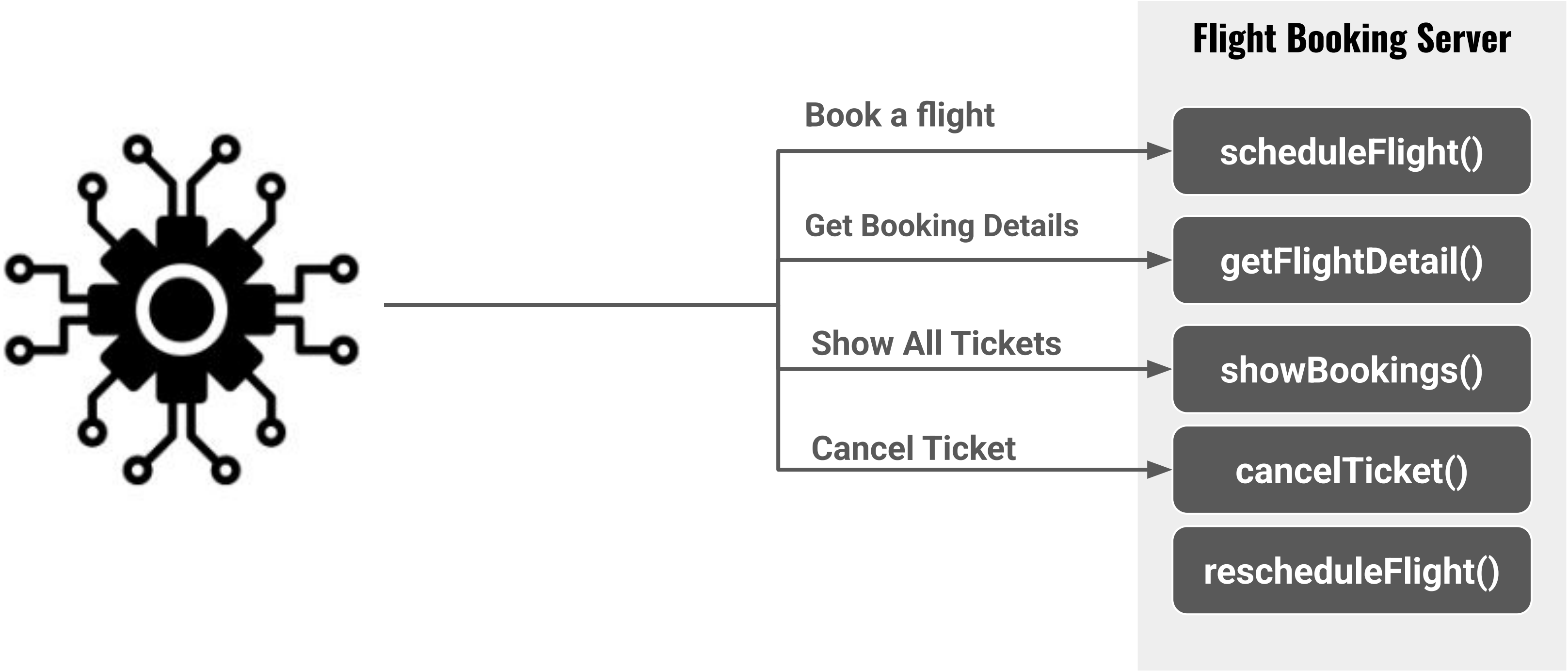
Flight Booking API: getFlightDetail() RPC



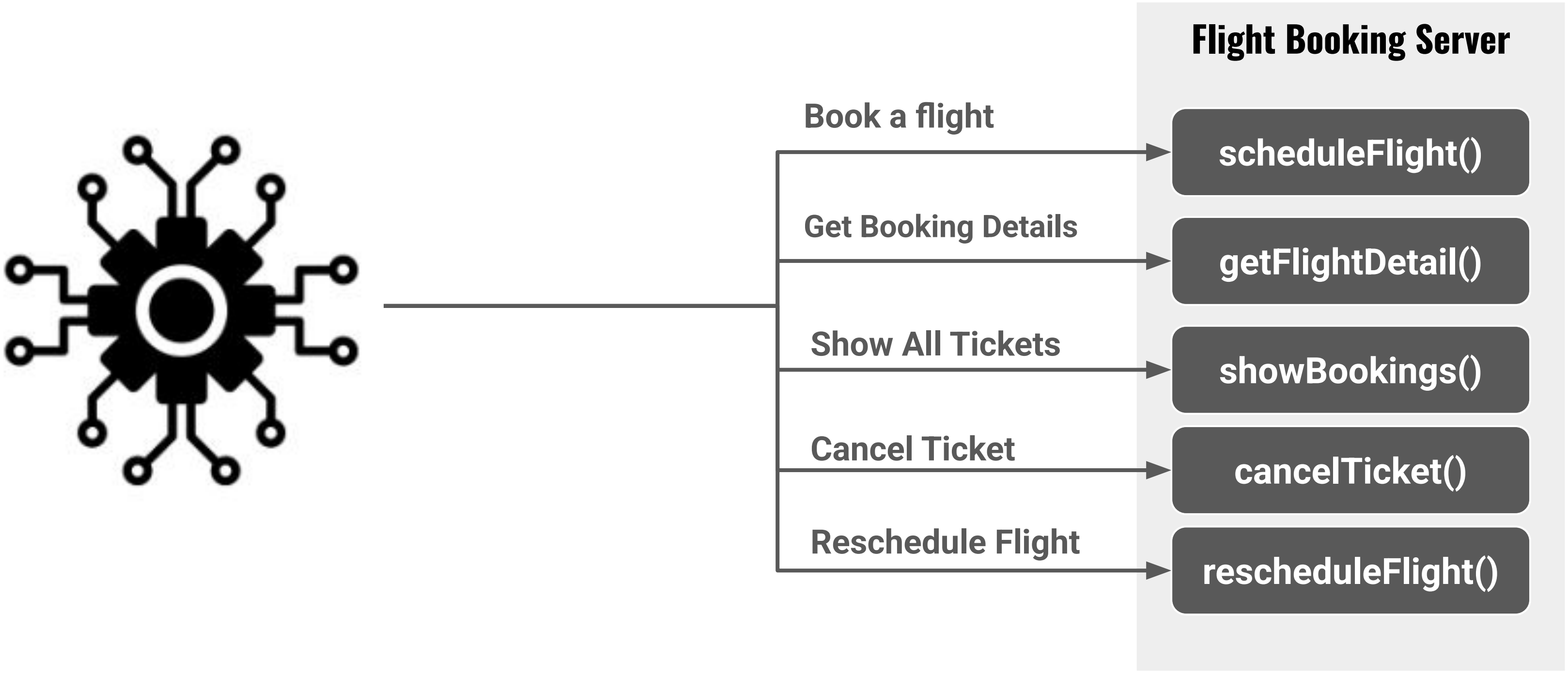
Flight Booking API: showBookings() RPC



Flight Booking API: cancelTicket() RPC



Flight Booking API: reschedule() RPC



Flight Booking Server

Book a flight

scheduleFlight()

Get Booking Details

getFlightDetail()

Show All Tickets

showBookings()

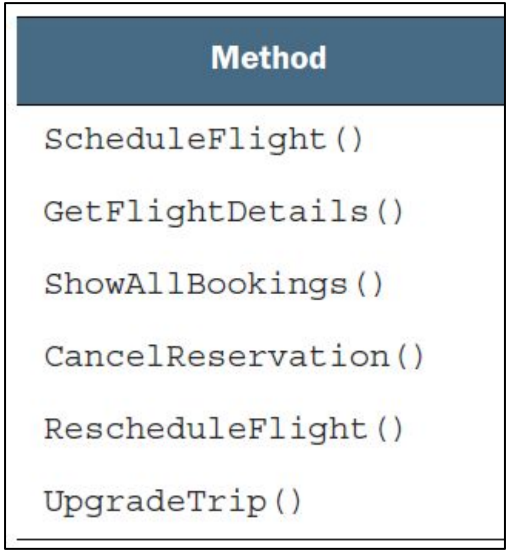
Cancel Ticket

cancelTicket()

Reschedule Flight

rescheduleFlight()

Flight Booking API



Method
ScheduleFlight()
GetFlightDetails()
ShowAllBookings()
CancelReservation()
RescheduleFlight()
UpgradeTrip()

Each of these RPCs is pretty descriptive, but there’s no escaping the requirement that we memorize these API methods

For example, sometimes a method talks about a “flight” (e.g., RescheduleFlight()) and other times operates on a “reservation” (e.g., CancelReservation())

We need to remember whether the way to see all of our bookings is ShowBookings(), ShowAllBookings(), ListBookings(), or ListAllBookings()

But what can we do to solve this? The answer comes in the form of standardization by orienting the methods around resources!

Day 2 & 3

Enter REST API

A brief introduction

More Details Later

Resource Oriented APIs: REST

Resource-oriented APIs rely on the idea of “**resources**,” which are the key concepts we store and interact with

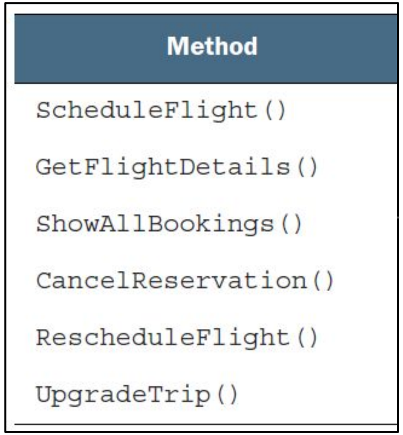
Resources are the **entities** of the **domain**. For example, for a Flight Booking API, **reservation** is one resource. For a bank - **customer** is one resource, **account** can be another resource, **employees** can be another resource, **transaction** is one more resource

The reason why resource oriented API are easy to understand is that once we identify a resources, there is only limited actions you can do on them (**create**, **read, update, delete**) popularly known as **CRUD** operations

Let’s apply the resource oriented process to our Flight Booking API

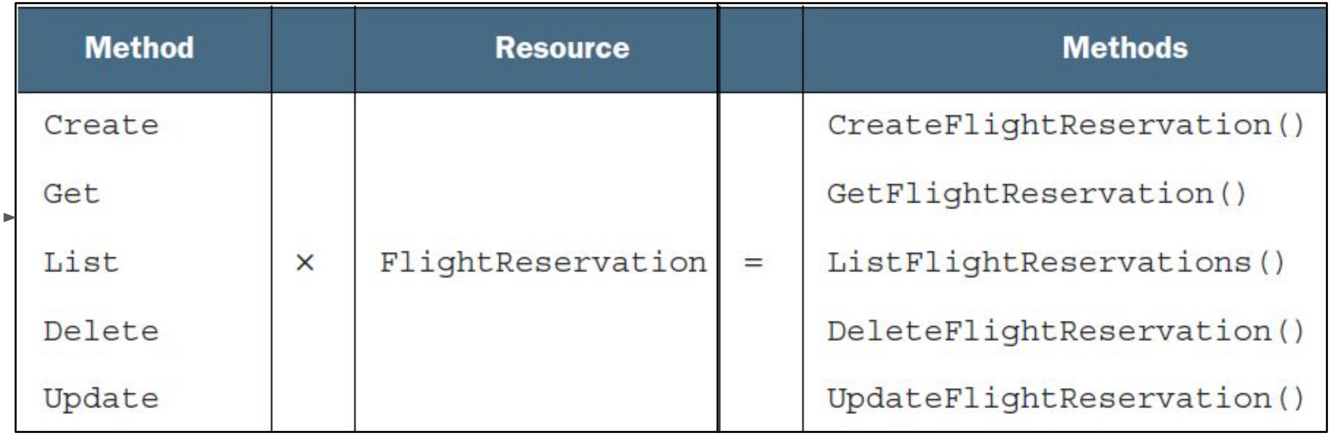
Resource Oriented APIs

Instead of the variety of different RPC methods shown below, we could come up with a single resource (e.g., FlightReservation) and get equivalent functionality with the set of standard methods



Method
ScheduleFlight()
GetFlightDetails()
ShowAllBookings()
CancelReservation()
RescheduleFlight()
UpgradeTrip()

RPCs



Method		Resource		Methods
Create	×	FlightReservation	=	CreateFlightReservation()
Get				GetFlightReservation()
List				ListFlightReservations()
Delete				DeleteFlightReservation()
Update				UpdateFlightReservation()

Resources

Resource Oriented APIs: A Bank

The same list of methods applies for almost any resources we can imagine. Let's take example of 3 resources for a bank: **Accounts, Customers, Employees**

Bank Account

Method		Resource		Actions
Create	X	**Bank Account**	=	Create an Account
Read (Get)				Read an Account
Update				Update an Account
Delete				Delete an Account

Resource Oriented APIs: A Bank

The same list of methods applies for almost any resources we can imagine. Let’s take example of 3 resources for a bank: Accounts, Customers, Employees

Customers

Method		Resource		Actions
Create	**X**	**Customers**	**=**	Create a Customer
Read (Get)				Get a Customer
Update				Update a Customer
Delete				Delete a Customer

Resource Oriented APIs: A Bank

The same list of methods applies for almost any resources we can imagine. Let’s take example of 3 resources for a bank: Accounts, Customers, Employees

Employee

Method		Resource		Actions
Create	**X**	**Employee**	**=**	Create a Employee
Read (Get)				Get a Employee
Update				Update a Employee
Delete				Delete a Employee

<<15:00>>

Activity

- Create a 5 resources for a company of your choice like Ola or Uber or NPCI (at least 5)
- Create the actions possible on those resources using Resource Oriented approach

Solution

RPC vs REST

When it comes to choice - there is neither right or wrong, nor better or worse

Depending on our requirement, we have to choose one!

Aspect	RPC	REST
Internal vs External clients	Better for internal client and implementation within the org	Better when API is to be exposed over to public as its easily understood and adopted due to resources
Speed	Very fast as it does not have to rely on HTTP methods. Suitable for ultra high speeds.	Slightly slower due to HTTP headers and resource based payloads
Industry adoption	Not first choice for public APIs	Very widely adopted
Developer Experience	Long list of actions and comma separated arguments list make it cumbersome to use	With standard resources, methods and HTTP standards - it is very easy to use and integrate

Let’s Design A Payment API

<<3:00>>

Let’s Design A Payment API

Imagine you are told to design a Payments API

Where will you begin?

Take 2 minutes to think about where to begin? What do you need to start?

<<3:00>>

Let’s Design A Payment API

Imagine you are told to design a Payments API
Where will you begin?

A natural starting point is to find out all the features required in the API
Can you try to jot down some features that comes to your mind?

Let’s Design A Payment API

Imagine you are told to design a Payments API
Where will you begin?

A natural starting point is to find out all the features required in the API

How do you document and list those features? There are so many things to think about in a feature.
In what scenario does the feature come in, who will use it, what are possible results or actions, etc

Let’s Design A Payment API

Imagine you are told to design a Payments API
Where will you begin?

A natural starting point is to find out all the features required in the API

How do you document those features? There are so many things to think about in a feature. In what scenario does the feature come in, who will use it, what are possible results or actions, etc

Is there a general framework/process we can follow to begin our design process?
Yes! JTBD Framework - Let’s dive in

Where to Start? Job To Be Done (JTBD)

JTBD method provides a very good starting point where the fundamental aspects of API can be captured

JTBD method formulated as a method of taking the **viewpoint of the customer** when designing a product or service

It starts by identifying the needs of customers, **the job,** and then defining how a product or service will fill that need

In JTBD, jobs are more than just functions that need to be performed. Jobs are really about the desired outcome or accomplishment

What are Job(User) Stories?

Customers and users don’t care about APIs, microservices, serverless, or the flavor of frontend framework used

They want a solution to a problem. They care about outcomes

Job stories capture the jobs to be done for any product, including the customer motivations, events, and expectations for a new product, service, or API

Job stories seek to identify the problems that customers have and the eventual outcome they wish to achieve

Job stories will also help in details necessary to create acceptance criteria for automated tests

It is important to note that job stories shouldn’t contain implementation specifics. Instead, they should elaborate on what needs to happen to deliver the outcome

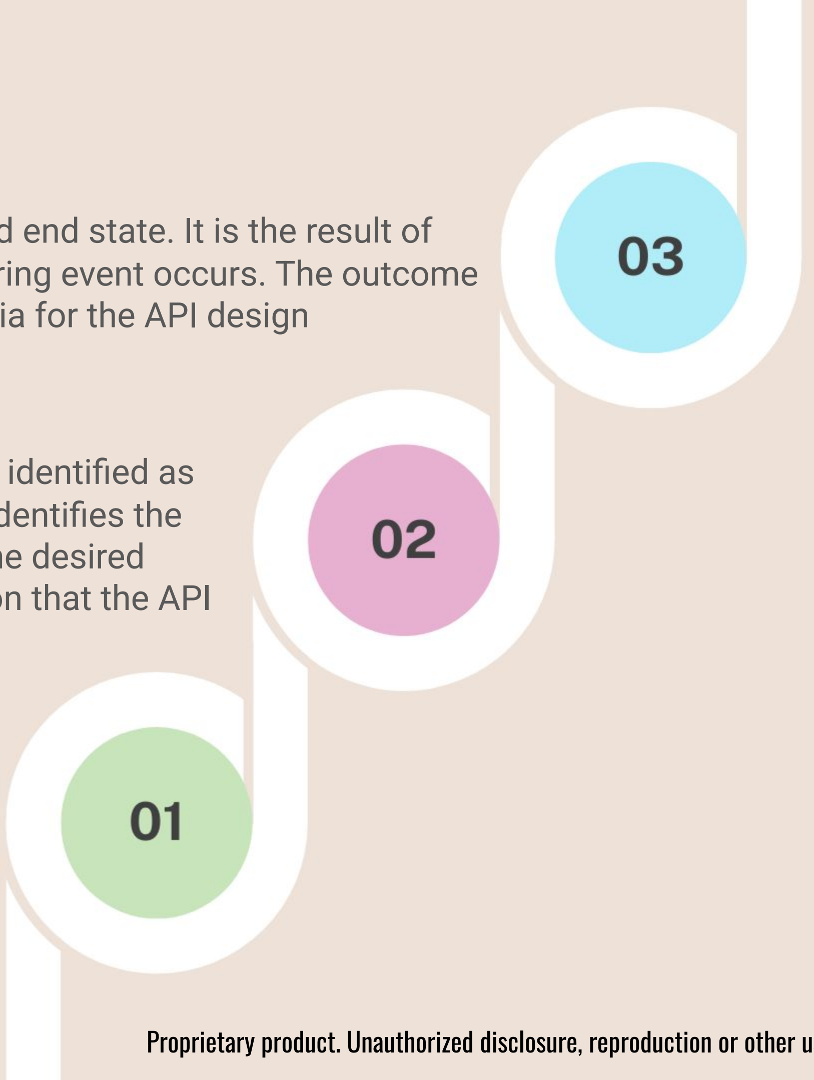
This allows us to capture business requirements in a customer-centric way

Components of a Job Story

So I can: The ***outcome*** is the desired end state. It is the result of applying the capability when the triggering event occurs. The outcome drives the acceptance criteria for the API design

I want to: The ***capability*** is what the customer has identified as the action that needs to be taken. The capability identifies the important role that the API will play to deliver the desired outcome. It is also used to deconstruct the operation that the API will deliver

When: The ***triggering*** event indicating for when an API will be used



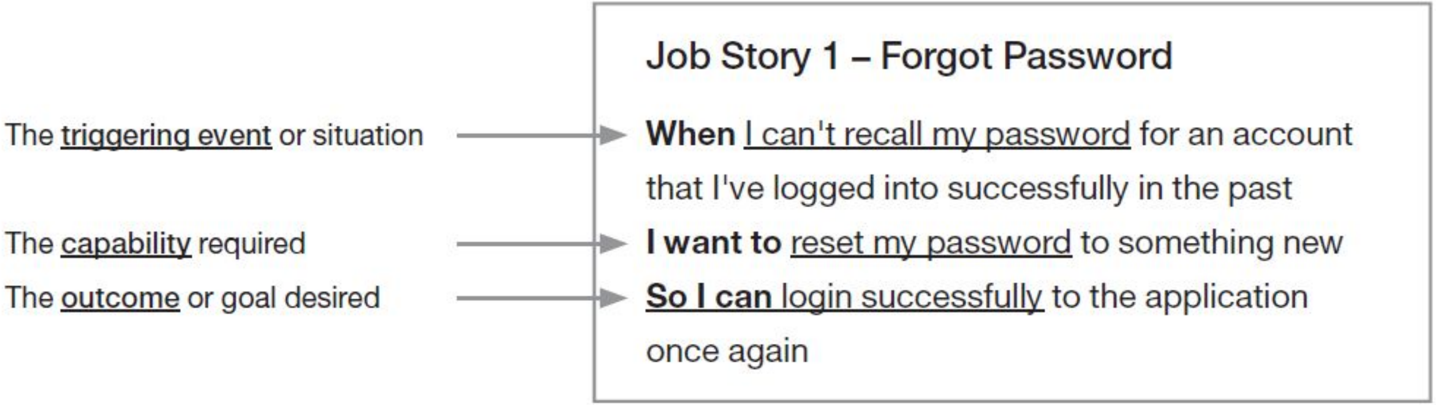
An Example of Job Story

In almost any product you build, there is always a scenario of User’s forgetting their password

So the product has to has a ‘digital capability’ to reset the password.

So, how do we write the Job story of this digital capability?

Note that this is one of many digital capabilities the API must offer to meet the needs of the target customers



The triggering event or situation

The capability required

The outcome or goal desired

Job Story 1 – Forgot Password

When I can't recall my password for an account that I've logged into successfully in the past

I want to reset my password to something new

So I can login successfully to the application once again

Few Points about writing Job Stories for APIs: Short is Sweet

Don’t make it too detailed - because details will be added later as task

Take an example of a job story for a digital capability of **add to cart**

When	I find a product I want to buy
I want to	provide the quantity, color, and style of the product
So that I can	add it to my shopping cart and see the current subtotal, shipping costs, and estimated sales tax

When	I find a product I want to buy
I want to	add the product to my shopping cart
So that I can	include it in my order
Additional Details	The following fields will be required when adding an item to a cart: quantity, color, and style. The shopping cart will then show the current subtotal, shipping costs, and estimated sales tax

Few Points about writing Job Stories for APIs: Don’t Make It Feature Centric

Job stories should be focused on Outcomes and not features of the job

Take an example of a job story for a digital capability of add to cart

When	I find a product I want to buy
I want to	add the product to my shopping cart by clicking a yellow button
So that I can	include it in my order

When	I find a product I want to buy
I want to	add the product to my shopping cart
So that I can	include it in my order
Additional Details	The button to add a product to a cart should be yellow The label should say “Add to Cart.”

Job Stories for an E-commerce application

To explore the JTBD process, let’s first try to apply it using an e-commerce application

And then as DIY - you can do it for the Payments API

Job Stories for our E-commerce

ID	When	I want to	So I can
1	I am going to the website	Login to my account	Place an order
2	When I go to home page	See suggestions of books curated for me	Make a selection of what to buy
3	I need latest books	Sort book by release date	Keep up with the latest watercooler talk
4	I want to find a book that will be entertaining or teach me something new	Search for a book by topic or keyword	Browse related books
5	I encounter an unfamiliar book	View a book’s details and reviews	Determine if the book is of interest to me
6	I find one or more books that I wish to buy	Place an order	Buy the books and have them shipped to my preferred address
7	I am uncertain of when my order will arrive	View the status of an order	Confirm the date that the order will arrive

<<15:00>>

Let’s go back to Payments API

We are in the same situation - You are asked to begin design of the Payments API
Can you use JTBD framework to begin the creation of feature list?

Job Stories for a Payment based App

ID	When	I want to	So I can
1	**When** I want to use the Payment App	**I want to** create a user account	**So I can** start sending and receiving payments
2	**When** I want to securely access my account	**I want to** authenticate using my username and password	**So I can** manage my payments and user settings
3	**When** my personal or account information changes	**I want to** update my user details	**So I can** ensure my account is accurate and up-to-date
4	**When** I no longer want to use the platform	**I want to** delete my user account	**So I can** remove my data and discontinue the service
5	**When** I need to review all payments	**I want to** retrieve a list of all payments	**So I can** monitor financial activities and reconcile records
6	**When** I need to make payment someone	**I want to** initiate a payment	**So I can** settle my balance
7	**When** the payment is complete	**I want to** come to the app & check it	**So I can** ensure money is transferred
8	**When** i want to check specific details of payment	**I want to** get all the details of that payment	**So I can** confirm its accuracy and settle disputes

Job Stories for a Payment based App

ID	When	I want to	So I can
9	**When** I receive a payment	**I want to** get notified	**So I can** be informed about money received
10	**When** a payments get delayed	**I want to** to get notified when status updates	**So I can** check whether payment succeeded or failed
11	**When** there are any offers	**I want to** be notified	**So I can** utilize them and get benefits
12	**When** money is debited but not credited	**I want to** raise a complaint	**So I can** get the refund and resolve issue
13	**When** a complaint is raised	**I want to** check the status of the complaint	**So I can** be informed about resolutions and next steps
14	**When** there is update to the complaint	**I want to** be notified	**So I can** updated with latest status

Identifying API Boundaries

Introduction

Now we have list of stories (features) to be implemented using the APIs

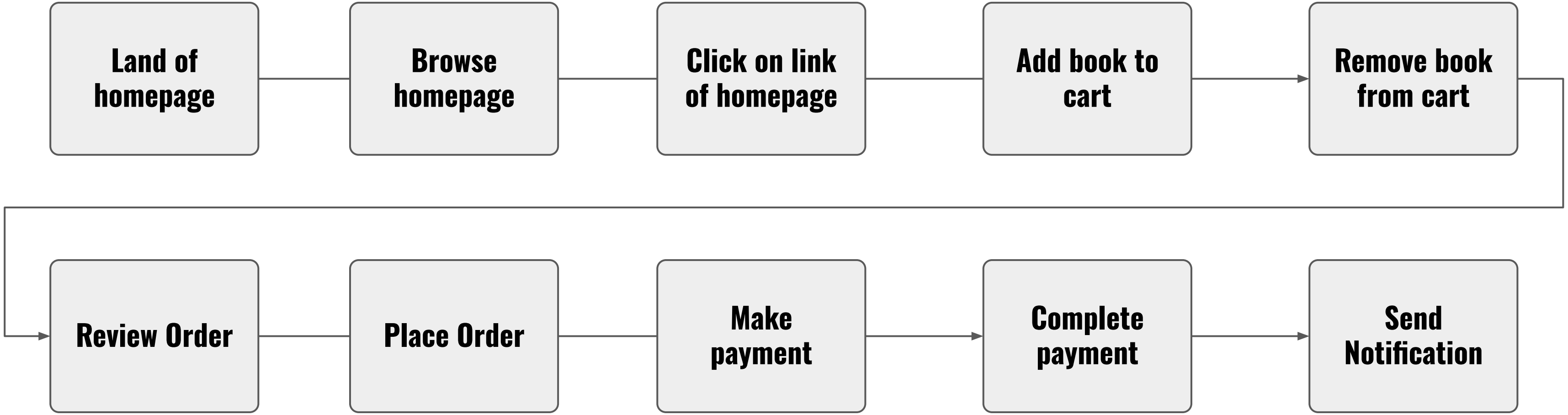
The next step is to segrate the features into **logical groupings** like (Orders, Notification, Login, Browsing, etc)

One way of doing this is to picture the end to end lifecycle or path through which a customer goes through

And then create boundaries and distribute the work among different teams

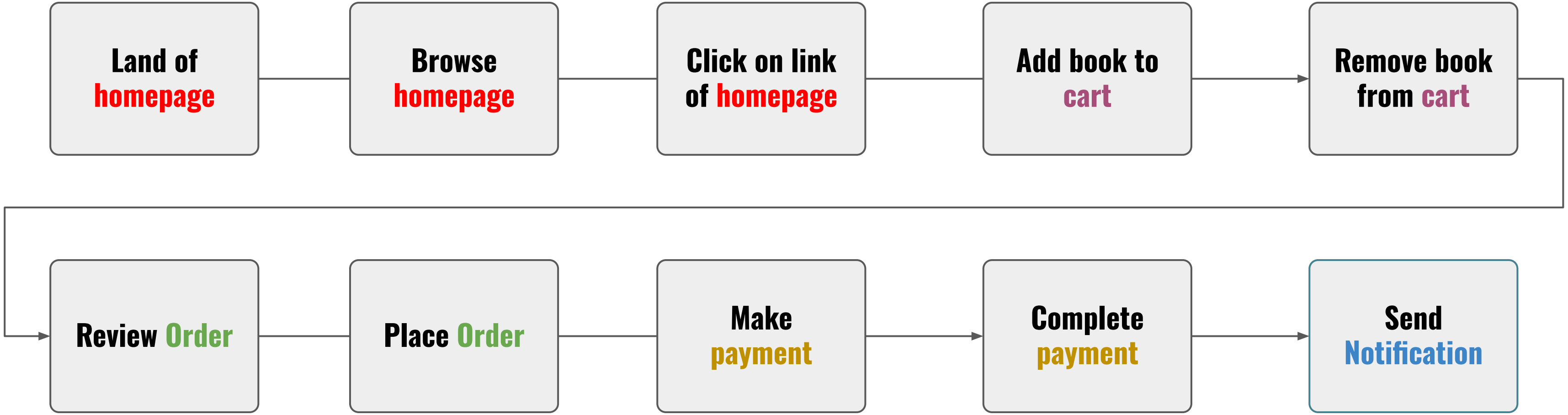
How to identify the boundaries?

One way to find the API boundary is to create a series of steps that a user is going to take while using an API



How to identify the boundaries?

By examining the language used throughout the step, developers find hints about possible API boundaries


Land of homepage

Browse homepage

Click on link of homepage

Add book to cart

Remove book from cart

Review Order

Place Order

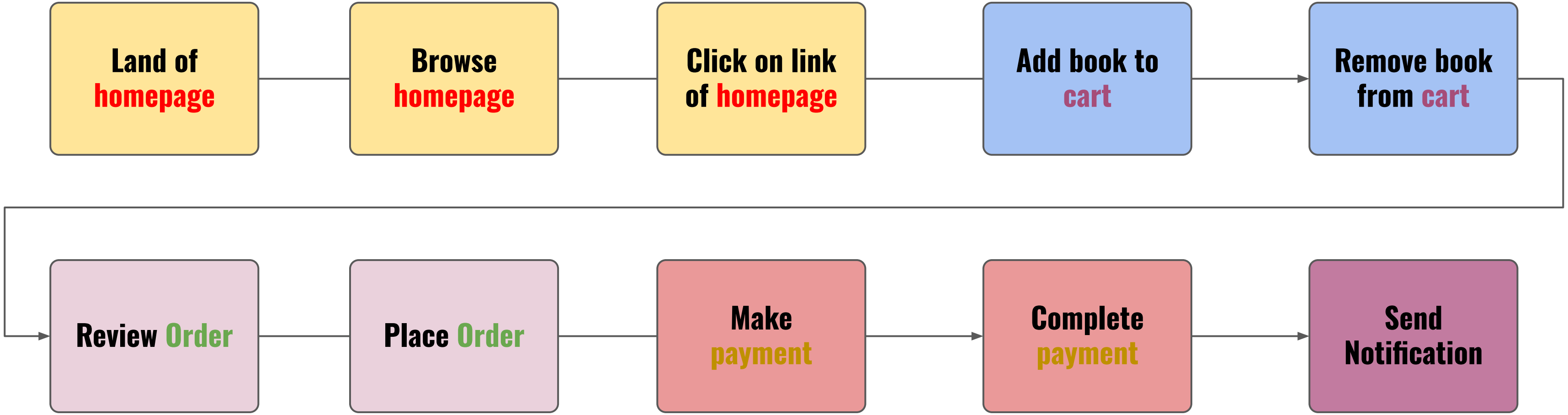
Make payment

Complete payment

Send Notification

How to identify the boundaries?

By examining the language used throughout the step, developers find hints about possible API boundaries

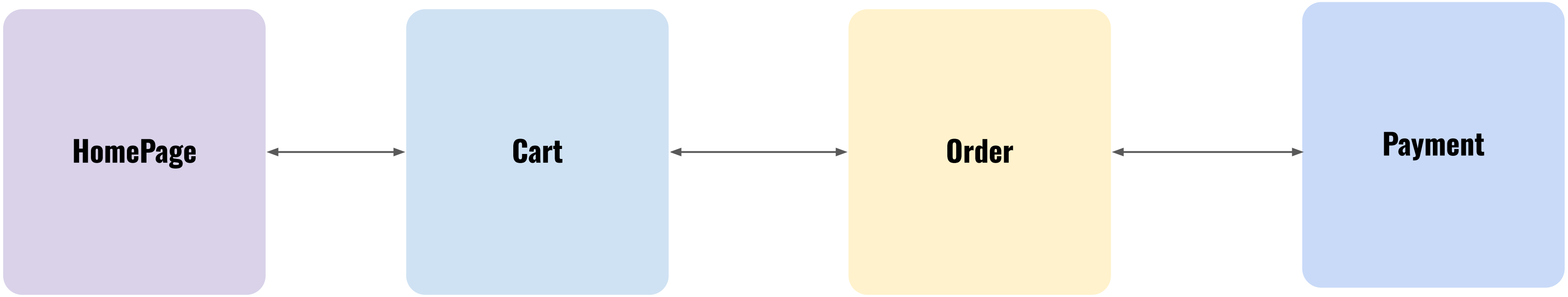


Proprietary product. Unauthorized disclosure, reproduction or other use prohibited

How to identify the boundaries?

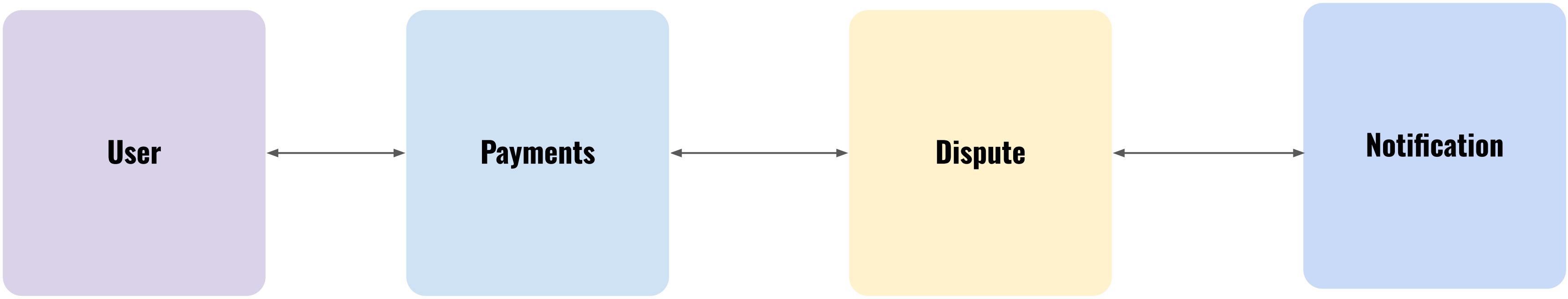
Once boundaries are identified, they become different components which interact as Web API (over a network)

Different components often become different team who work parallelly to implement the final product



Can you Identify Boundaries for our Payments API?

Can you Identify Boundaries for our Payments API?



User

Payments

Dispute

Notification

We are all set to start the actual development of API

And one of the first step in that process to select the Architecture style

API Architecture Pattern

Now that we know the process, it's time to dive in the actual development process.
We are presented with multiple architecture pattern that we have to choose from based on certain conditions

RPC

REST

Microservices

Event Driven

Introduction to this Section

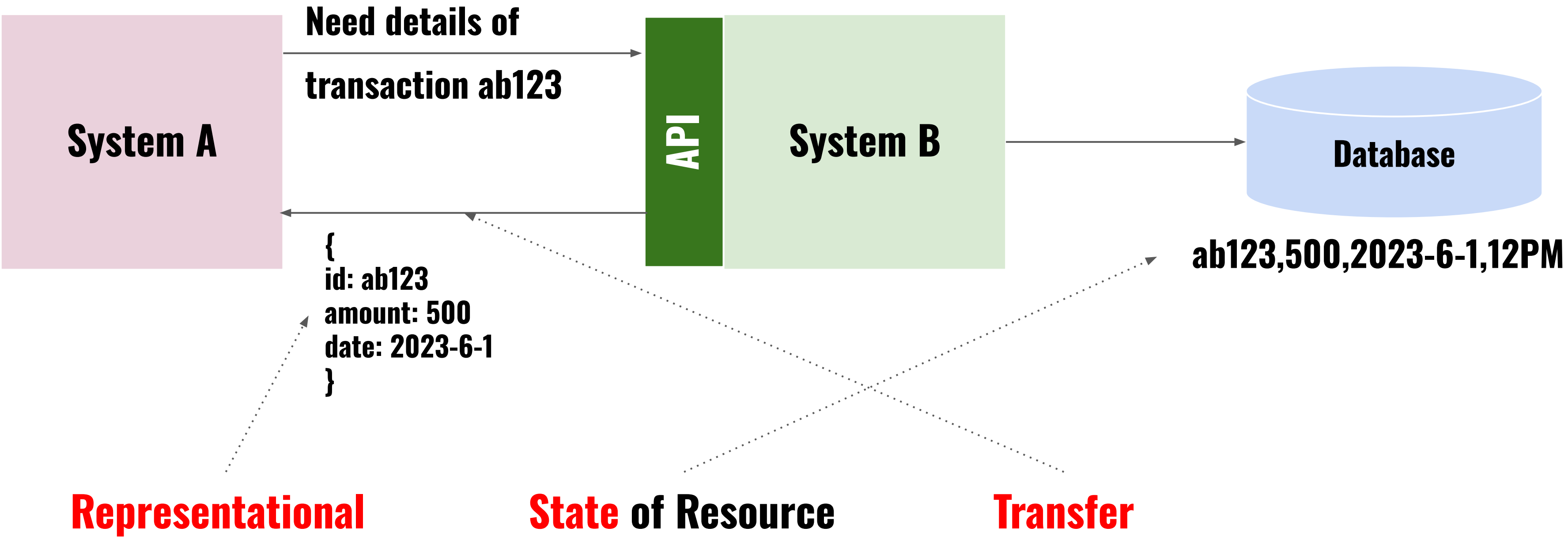
Representational state transfer (REST) describes an architectural style for applications that communicate over a network

The key idea in REST is about transfer of “resource” in a particular “representation”

The **representation** here means the **format of data** the client is requesting the data in: JSON, XML, text, etc

Recall from our previous lectures: resources are entities of a domain. For a payment domain resources are payers, payees, transactions, etc
For e-commerce example of resources: orders, customers, products, etc

Dissection of the word REST



System A

Need details of

transaction ab123

API

System B

Database

```
{
id: ab123
amount: 500
date: 2023-6-1
}
```

ab123,500,2023-6-1,12PM

Representational

State of Resource

Transfer

Introduction to this Section

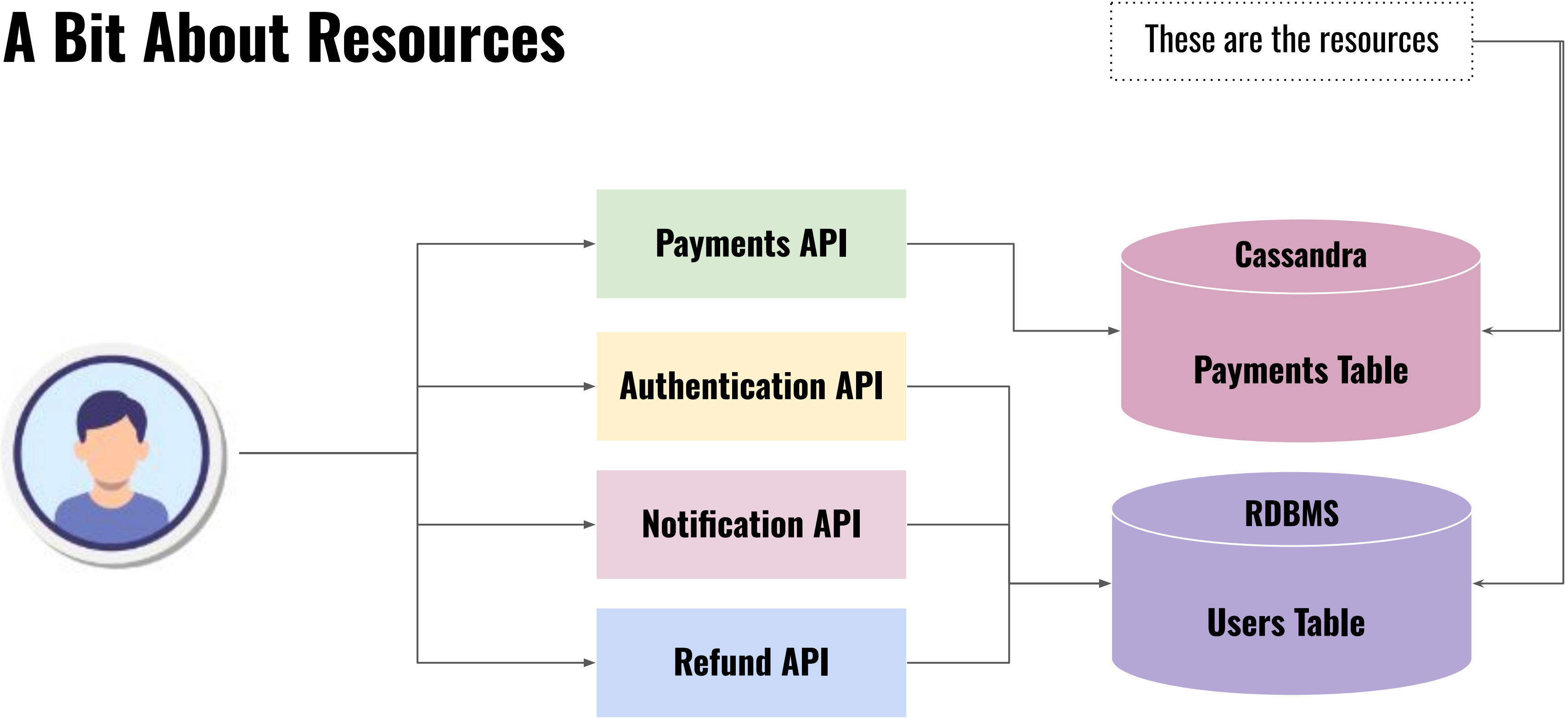
Today, REST is by far the most popular choice for building web APIs

REST is just a specification which tells you what you **should** be doing but it does not enforces any errors if you don't

In this section we will learn to apply the design principles of REST by designing the orders API of the CoffeeMesh platform. We will add more complexities of payments, delivery, etc

We will learn the concept of a resource, HTTP Protocols, Verbs, status codes, URL parameters, API payloads, etc

A Bit About Resources



A Bit About Resources

Store as however you want.
Serve it how client wants

Clients can mention what format/representation they want the data in the “content-type” header

It doesn’t mean that API has to implement the complete list of representation. The representation gets decided beforehand

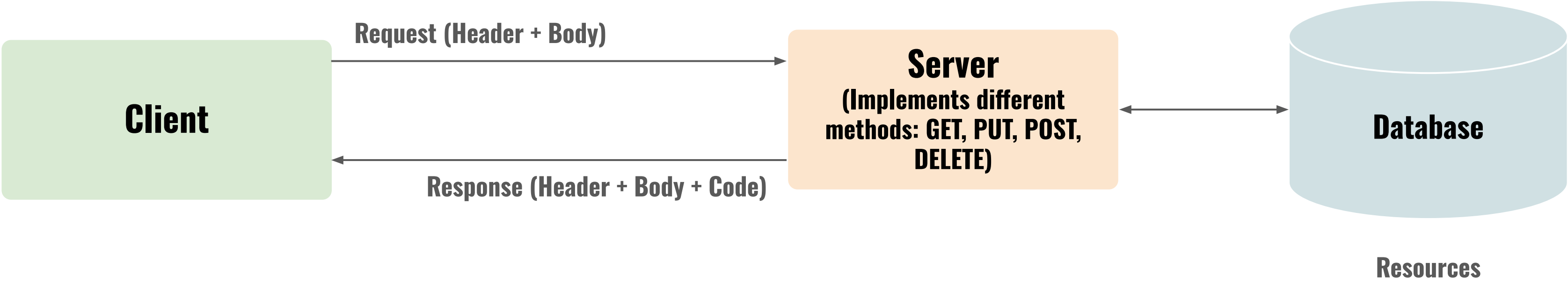
Hands On

A simple API that supports both JSON and XML representation

Link

Components of REST API

Before we begin - let's get the nomenclature out of our way


Request (Header + Body)
Client
Server
(Implements different methods: GET, PUT, POST, DELETE)
Database
Response (Header + Body + Code)
Resources

CRUD operations by using resource URLs with HTTP methods

Structured resource URLs with HTTP methods

Using HTTP methods and status codes is associated with a mature API design

HTTP methods are special keywords used in HTTP requests to indicate the type of action we want to perform in the server

Proper use of HTTP methods makes our APIs more structured, elegant, understandable and easy to use

Definition

HTTP request methods are **keywords** used in HTTP requests to indicate the **type of action** we wish to perform. For example, the **GET** method **retrieves** the details of a **resource,** while the **POST** method **creates** a new **resource.** The most important HTTP methods for REST APIs are **GET, POST, PUT, PATCH, and DELETE.** HTTP **methods** are also known as **verbs**

Semantics of HTTP Methods

GET	Returns information about the requested resource
POST	Creates a new resource
PUT	Performs a full update by replacing a resource
PATCH	Updates specific properties of a resource
DELETE	Deletes a resource

HTTP Methods

CRUD Operations using HTTP Methods

Methods	Actions	
GET	• Use GET for reading resources. • GET requests never, ever change the state of the resource • GET method has a read-only semantic	R
POST	• Used for creating new resources • Creating new user, order, payment, bill, etc	C
PUT	• Used for replacing an existing resources • Updating a name of user, updating orders, etc	U
DELETE	• Used for deleting an existing resources • Deleting a user, order, etc	D

PUT vs PATCH

Technically, both PUT and PATCH is used to update a resource

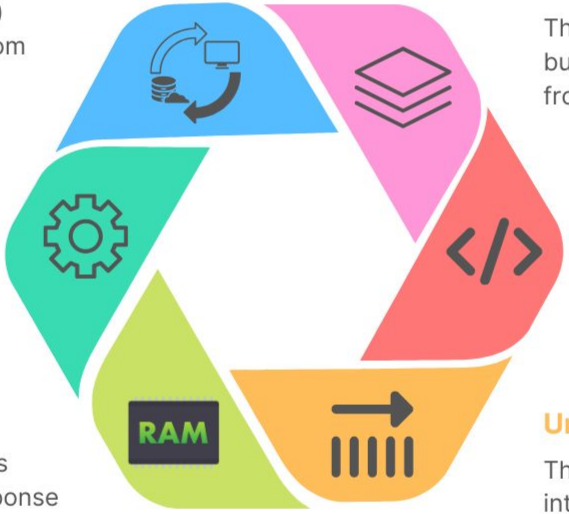
However, there usage is dependent of type of update

If the resource is being completely overwritten then we use PUT.
Example: Resetting a profile to default value

If one or more parameters of the resource are updated then we use patch.
Example: Updating payment status

Architectual Specifications of REST applications

What makes any API a REST API



Client Server Architecture

The user interface (UI) must be decoupled from the backend

Layered system

The API may be architected in layers, but such complexity must be hidden from the user

Statelessness

The server must not manage states between requests

Code on demand

The server can inject code into the user interface on demand

Cacheability

Requests that always return the same response must be cacheable

Uniform interface

The API must provide a consistent interface for accessing and manipulating resources

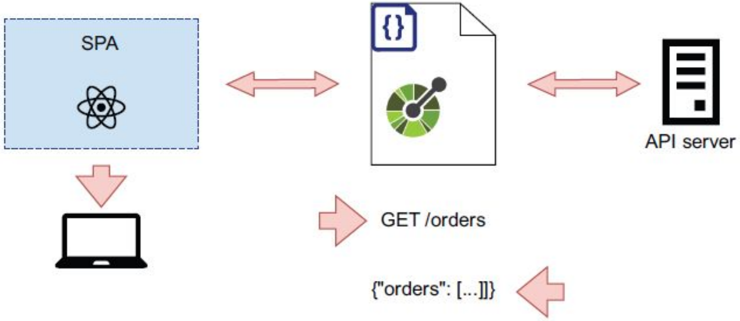
The specifications/constraints of REST application specify how a server should process and respond to a client request. Again, these are specifications and not bindings

Separation of concerns: The client-server architecture principle

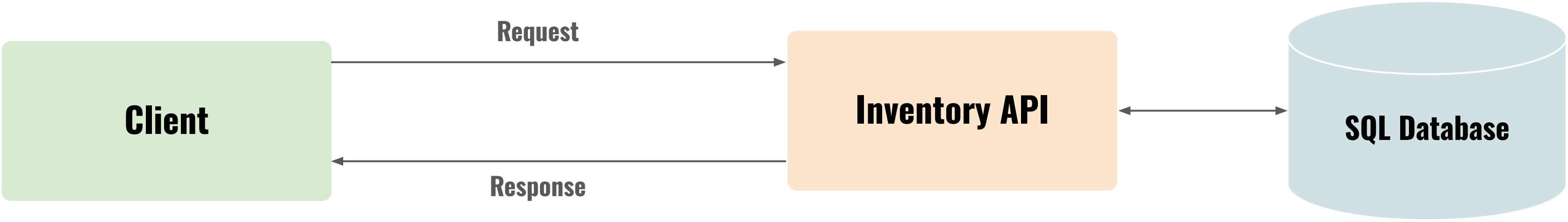
REST relies on the principle of separation of concerns where client and server are loosely coupled

The client and the server **act and evolve independently**, and the interaction between them is only in the form of **requests** and **responses**

This allows server-side components to evolve independently from the Client. The server may offer new resources or additional representation formats without negatively impacting the client



Separation of concerns: Example



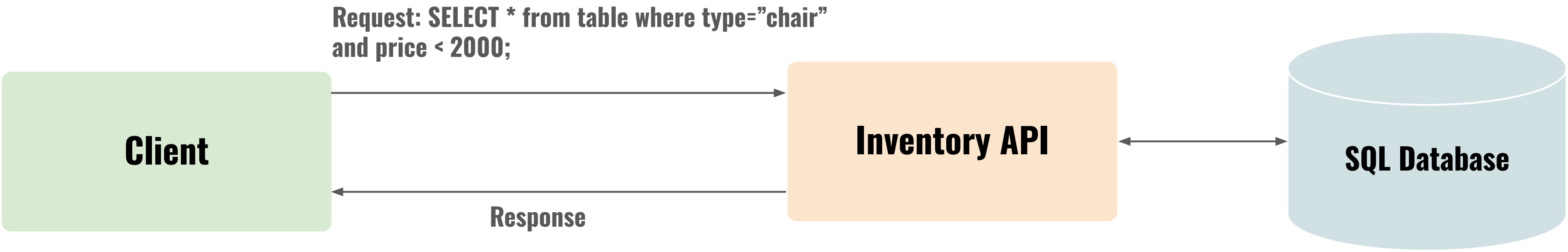
Let’s say we have to create an inventory API for our e-commerce application.

Simply put, we need this API to fetch the inventory details (if item is there, cost, quantity available, etc)

Let’s see how it can be implemented in two ways:

- **Tightly coupled**: where if one component changes - the other component has to change
- **Loosely coupled**: where change in one component does not impact other components and they can evolve independently

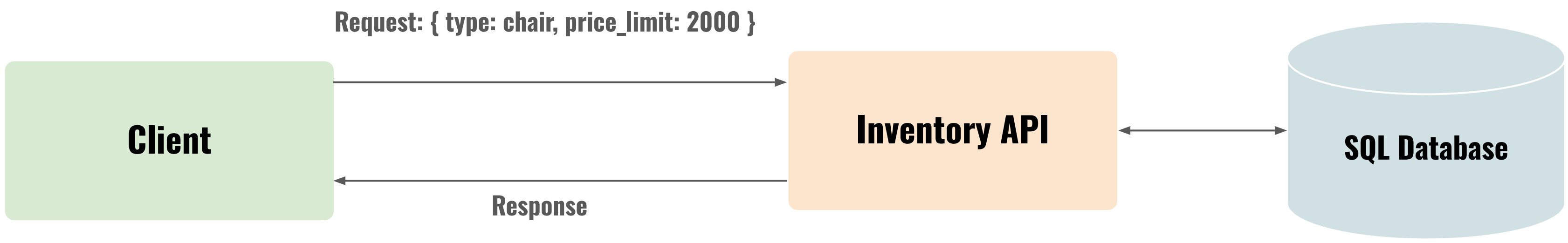
Separation of concerns Violated (Tightly coupled)



The API is created in such a way that it accepts the complete SQL query (or partial SQL query) and then executes it in the DB and sends response back

What happens if the DB changes? From SQL to noSQL?
The client code has to change, which means clients and backed cannot evolve independently
Imagine if 5-7 services (order, payment, notification) depend of inventory API - all of them have to change now

Separation of concerns Not-Violated (Loosely coupled)



Instead the API should accept condition as request parameters as shown above and then create the SQL query itself in the API (not by client)

What happens if the DB changes? From SQL to noSQL?
The client code does not change event a bit. Only inventory API has to change from using SQL queries to noSQL queries. All the 5-7 microservices using inventory API do not have to change

REST APIs are Stateless

But before we go there, a Quick Detour

Horizontal vs. Vertical Scalability

Make it scalable: The statelessness principle

In REST, every request to the server must contain all the information necessary to process it

The server does not remember anything about the previous requests to the API, so all necessary information to process the request must be provided by the client on each request

Removing state management from server components makes it easier to scale the backend horizontally

This allows us to deploy multiple instances of the server, and because none of those instances manages the API client’s state, the client can communicate with any of them

Note: This isn’t about storing server-side state, it's more about storing client side state

The statelessness principle violated: Login Example

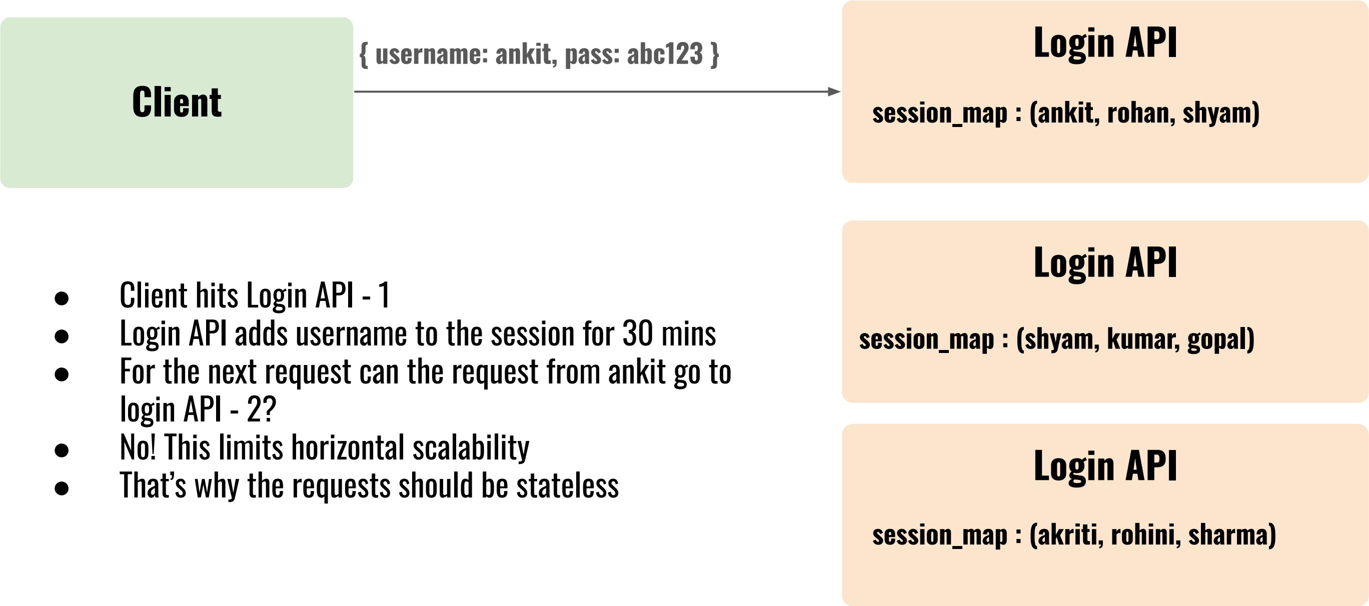
All of us have seen that once we login to a website - our login information (state) is preserved, meaning once we login, we don't have to login every time page refreshes

One way to implement is that the login API creates a session ID and stores it in memory (stateful) and the client does not have a provide the username and password in each requests

This is a violation as the stateless principle says - clients should always provide entire information in each request because servers are not suppose to store or remember anything from previous request as it hinders scalability

Let’s see how

The statelessness principle violated: Login Example



Client

{ username: ankit, pass: abc123 }

Login API

session_map : (ankit, rohan, shyam)

Login API

session_map : (shyam, kumar, gopal)

Login API

session_map : (akriti, rohini, sharma)

- Client hits Login API - 1
- Login API adds username to the session for 30 mins
- For the next request can the request from ankit go to login API - 2?
- No! This limits horizontal scalability
- That’s why the requests should be stateless

Hide the Complexity: The layered system principle

In a REST architecture, clients must have a unique point of entry to your API and must not be able to tell whether they are connected directly to the end server or to an intermediary layer such as a load balancer

You can deploy different components of a server side application in different servers, or you can deploy the same component across different servers for redundancy and scalability

But this complexity should be hidden from the user by exposing a single endpoint that encapsulates access to your services

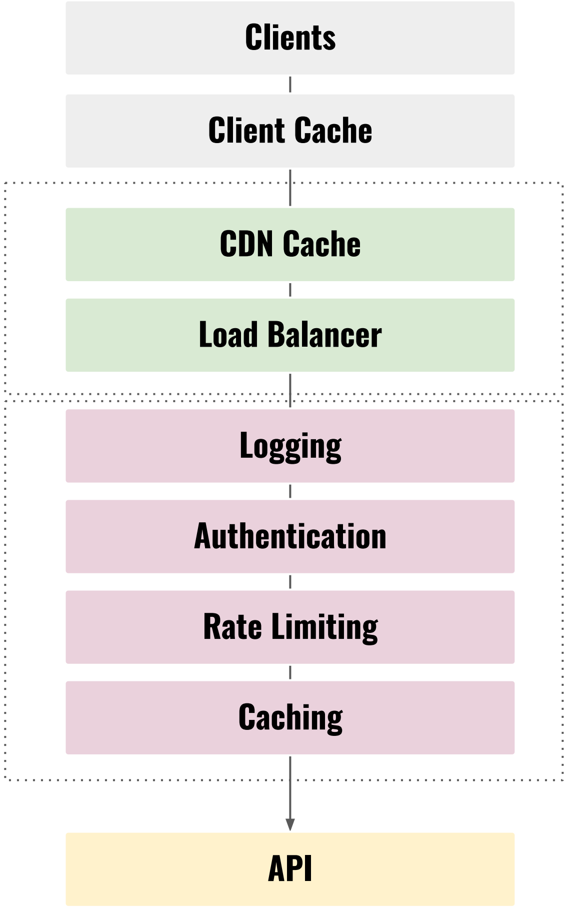
The client is agnostic as to how many layers, if any, there are between the client and the actual server responding to the request

This is a key principle of REST API, allowing for client-side caching, caching servers, reverse proxies, and authorization layering—all transparent to the client sending the request

Hide the Complexity: The layered system principle

Client should not be built on the assumption it is communicating directly with the server

There may be multiple middleware layers between the client and server that offer caching, logging, access controls, load balancing, and other infrastructure needs, as shown in figure


Clients
Client Cache
CDN Cache
Load Balancer
Logging
Authentication
Rate Limiting
Caching
API