

# Node.js runtime | HTTP

Week 2.2

# What are we learning today

## Foundation

Foundation Javascript, async nature of JS

Node.js and its runtime

Databases (NoSQL/SQL)

Mongo and Postgres deep dive

Typescript beginner to advance

## Backend

Backend communication protocols

Express basic to advance

ORMs

Middlewares, routes, status codes, global catches

Zod

MonoRepos, turborepo

Serverless Backends

OpenAPI Spec

Autogenerated clients

Authentication using external libraries

Scaling Node.js, performance benchmarks

Deploying npm packages

## Frontend

Reconcilers and Frontend frameworks

React beginner to advance

Internals of state, Context API

State management using recoil

CSS you need to know of, Flexbox, basic styling

Frontend UI frameworks, Deep dive into Tailwind

Containerization, Docker

Next.js

Custom hooks

In house auth using next auth

## Basic Devops

Docker end to end

Deploying to AWS servers

Newer clouds like fly/Remix

Nginx and reverse proxies

## Projects

GSoC Project setting up and issue solving

Building Paytm/Wallet End to End

# Node.js and its runtime

What is ECMAScript

What is Javascript?

What is Node.js?

What is Bun?



# What is ECMAScript

## What is Javascript?

## What is Node.js?

## What is Bun?

tc39.es/ecma262/#sec-numbers-and-dates

Search...

Table of Contents

> 7 Abstract Operations

> 8 Syntax-Directed Operations

> 9 Executable Code and Execution Contexts

> 10 Ordinary and Exotic Objects Behaviours

> 11 ECMAScript Language: Source Text

> 12 ECMAScript Language: Lexical Grammar

> 13 ECMAScript Language: Expressions

> 14 ECMAScript Language: Statements and Declarat...

> 15 ECMAScript Language: Functions and Classes

> 16 ECMAScript Language: Scripts and Modules

> 17 Error Handling and Language Extensions

> 18 ECMAScript Standard Built-in Objects

> 19 The Global Object

> 20 Fundamental Objects

> 21 Numbers and Dates

> 21.1 Number Objects

> 21.1.1 The Number Constructor

> 21.1.2 Properties of the Number Constructor

> 21.1.2.1 Number.EPSILON

> 21.1.2.2 Number.isFinite ( *number* )

> 21.1.2.3 Number.isInteger ( *number* )

> 21.1.2.4 Number.isNaN ( *number* )

> 21.1.2.5 Number.isSafeInteger ( *number* )

> 21.1.2.6 Number.MAX\_SAFE\_INTEGER

> 21.1.2.7 Number.MAX\_VALUE

> 21.1.2.8 Number.MIN\_SAFE\_INTEGER

> 21.1.2.9 Number.MIN\_VALUE

> 21.1.2.10 Number.NaN

> 21.1.2.11 Number.NEGATIVE\_INFINITY

> 21.1.2.12 Number.parseFloat ( *string* )

> 21.1.2.13 Number.parseInt ( *string*, *radix* )

> 21.1.2.14 Number.POSITIVE\_INFINITY

21.1.2.3 Number.isInteger ( *number* )

This function performs the following steps when called:

1. Return `isIntegralNumber(number)`.

21.1.2.4 Number.isNaN ( *number* )

This function performs the following steps when called:

1. If *number* is not a `Number`, return `false`.

2. If *number* is `NaN`, return `true`.

3. Otherwise, return `false`.

NOTE This function differs from the global `isNaN` function (19.2.3) in that it does not convert its argument to a `Number` before determining whether it is `NaN`.

21.1.2.5 Number.isSafeInteger ( *number* )

NOTE An integer *n* is a "safe integer" if and only if the `Number` value for *n* is not the `Number` value for any other integer.

This function performs the following steps when called:

1. If `isIntegralNumber(number)` is `true`, then

a. If  $\text{abs}(\text{B}(\text{number})) \leq 2^{53} - 1$ , return `true`.

2. Return `false`.

21.1.2.6 Number.MAX\_SAFE\_INTEGER

NOTE Due to rounding behaviour necessitated by precision limitations of IEEE 754-2019, the `Number` value for every integer greater than `Number.MAX_SAFE_INTEGER` is shared with at least one other integer. Such large-magnitude integers are therefore not `safe`, and are not guaranteed to be exactly representable as `Number` values or even to be distinguishable from each other. For example, both `9007199254740992` and `9007199254740993` evaluate to the `Number` value

# ECMAScript

ECMAScript is a scripting language specification on which [JavaScript](#) is based. [Ecma International](#) is in charge of standardizing ECMAScript.

**<https://tc39.es/ecma262/#sec-numbers-and-dates>**



# What is ECMAScript

## What is Javascript?

### What is Node.js?

### What is Bun?

#### 1. ECMAScript: The Specification

- **ECMAScript** is a scripting language specification standardized by Ecma International in the ECMA-262 and ISO/IEC 16262 documents. It serves as the guideline or the 'rules' for scripting language design.
- **Versions:** ECMAScript versions (like ES5, ES6/ES2015, ES2017, etc.) are essentially updates to the specification, introducing new features and syntaxes. For example, ES6 introduced arrow functions, classes, and template literals.

#### 2. JavaScript: The Implementation

- **JavaScript** is a scripting language that conforms to the ECMAScript specification. It's the most widely known and used implementation of ECMAScript.
- **Beyond ECMAScript:** JavaScript includes additional features that are not part of the ECMAScript specification, like the Document Object Model (DOM) manipulation, which is crucial for web development but is not defined by ECMAScript.

# What is ECMAScript

## What is Javascript?

### What is Node.js?

### What is Bun?

Date, var, const, let, function

setTimeout  
fs.readFile

#### 1. ECMAScript: The Specification

- **ECMAScript** is a scripting language specification standardized by Ecma International in the ECMA-262 and ISO/IEC 16262 documents. It serves as the guideline or the 'rules' for scripting language design.
- **Versions:** ECMAScript versions (like ES5, ES6/ES2015, ES2017, etc.) are essentially updates to the specification, introducing new features and syntaxes. For example, ES6 introduced arrow functions, classes, and template literals.

#### 2. JavaScript: The Implementation

- **JavaScript** is a scripting language that conforms to the ECMAScript specification. It's the most widely known and used implementation of ECMAScript.
- **Beyond ECMAScript:** JavaScript includes additional features that are not part of the ECMAScript specification, like the Document Object Model (DOM) manipulation, which is crucial for web development but is not defined by ECMAScript.



# What is ECMAScript

## What is Javascript?

## What is Node.js?

## What is Bun?

### Common JS Browser engines

1. V8 - Used by google chrome/chromium - C

<https://github.com/v8/v8>



2. SpiderMonkey - Used by Firefox - C + Rust

<https://spidermonkey.dev/>



1. **V8 Engine:** V8 is an open-source JavaScript engine developed by the Chromium project for Google Chrome and Chromium web browsers. It's written in C++ and is responsible for compiling JavaScript code into native machine code before executing it, which greatly improves performance.

**What is ECMAScript**  
**What is Javascript?**  
**What is Node.js?**  
**What is Bun?**

**Some smart people took out the V8 engine  
Added some Backend things (filesystem reads) on top  
to create a new runtime to compete with BE languages  
like Java.**

**JS was never meant to be run in the backend  
Eventually became very popular and is a popular  
choice of runtime on the backend**



**What is ECMAScript**  
**What is Javascript?**  
**What is Node.js?**  
**What is Bun?**

**Other than the fact that JS is single threaded,  
Node.js is slow (multiple reasons for it)  
Some smart people said they wanted to re-write  
the JS runtime for the backend and introduced Bun**

**It is a significantly faster runtime**

**It is written in Zig**

**<https://github.com/oven-sh/bun>**

**We will be focusing on Node.js**  
**Specifically, how to write Backend applications using**  
**Javascript**

# What can you do with Node.js?

1. Create clis
2. Create a video player
3. Create a game
4. Create an **HTTP Server**



# What is an HTTP Server?

# What is an HTTP Server?

## HTTP

**Hyper text transfer Protocol**

1. A protocol that is defined for machines to communicate
2. Specifically for websites, it is the most common way for your website's frontend to talk to its backend

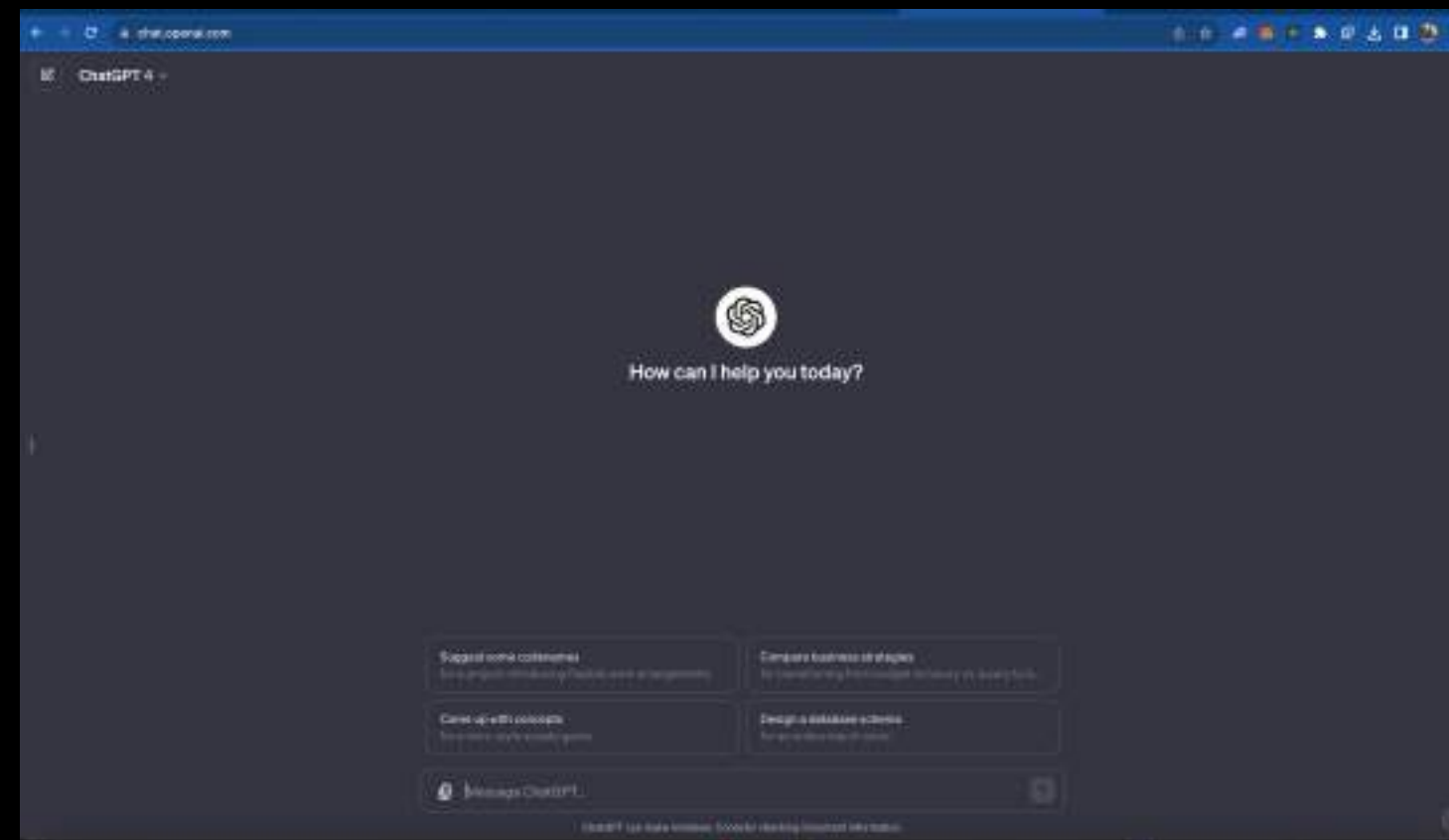
# What is an HTTP Server?

First lets understand what are **Frontends** and **Backends**



# What is an HTTP Server?

Frontend/Clients (HTML/CSS/JS)



India

Backends (Node.js)



California



# How do frontends talk to backends - Wires/routers





# What is an HTTP Server?

**Some code that follows the HTTP Protocol  
And is able to communicate with clients (browsers/mobile apps...)**

**Think of it to be similar to the call app in your phone  
Which lets you communicate with your friends**





# HTTP Protocol

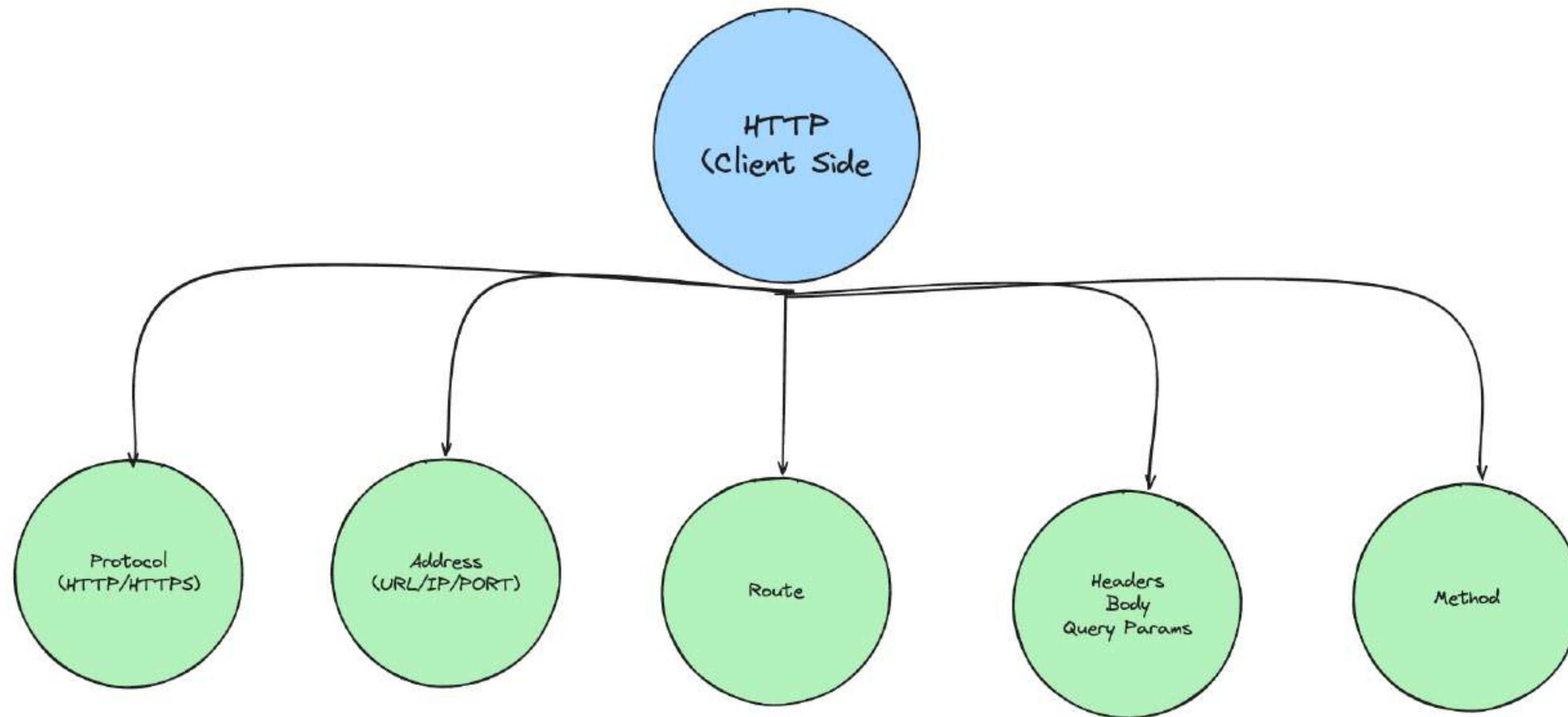
**In the end, its the client throwing some information at a server  
Server doing something with that information  
Server responding back with the final result**

**Think of them as functions, where**

- 1. Arguments are something the client sends**
- 2. Rather than calling a function using its name, the client uses a URL**
- 3. Rather than the function body, the server does something with the request**
- 4. Rather than the function returning a value, the server responds with some data**

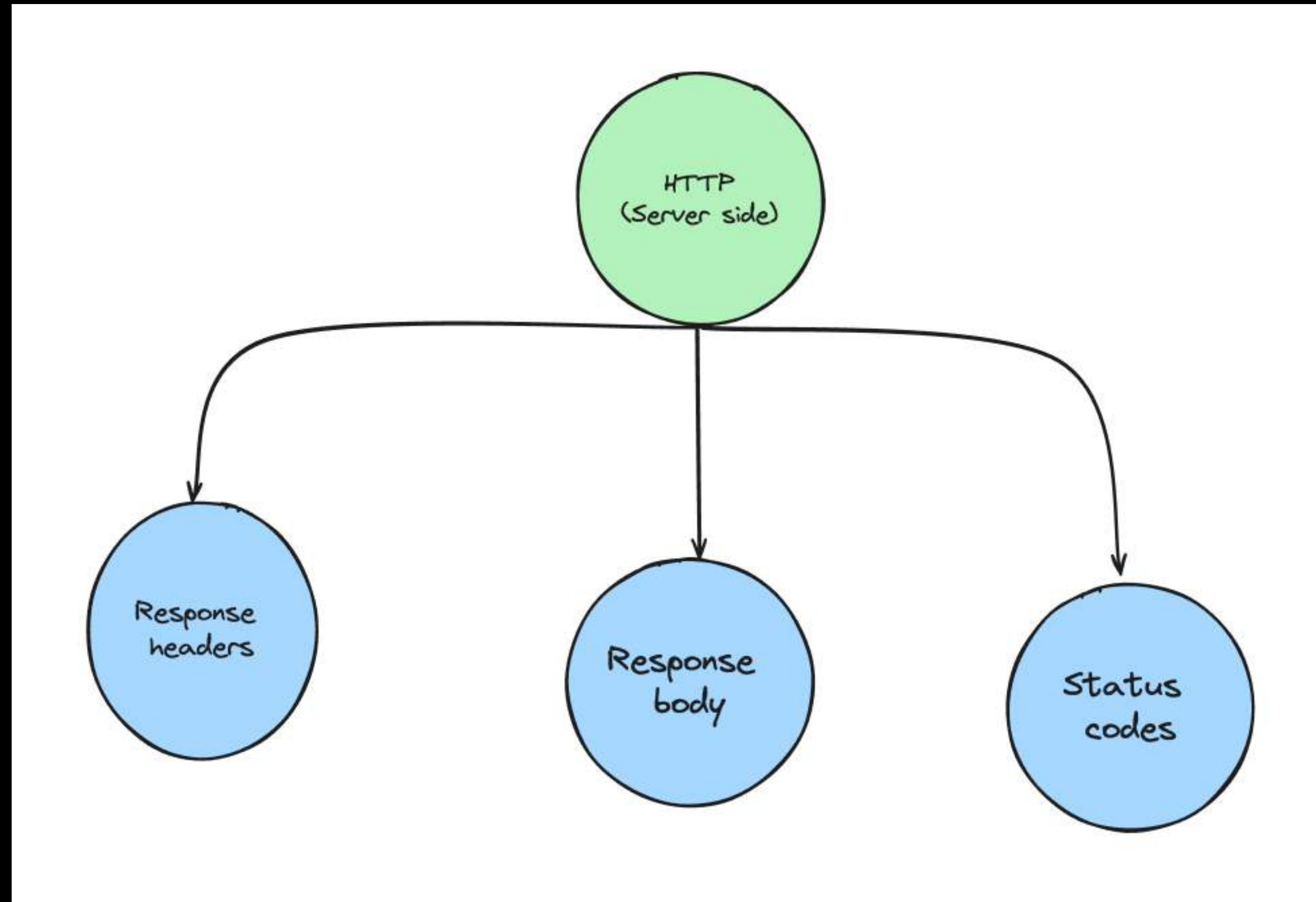
# HTTP Protocol

Things client needs to worry about



# HTTP Protocol

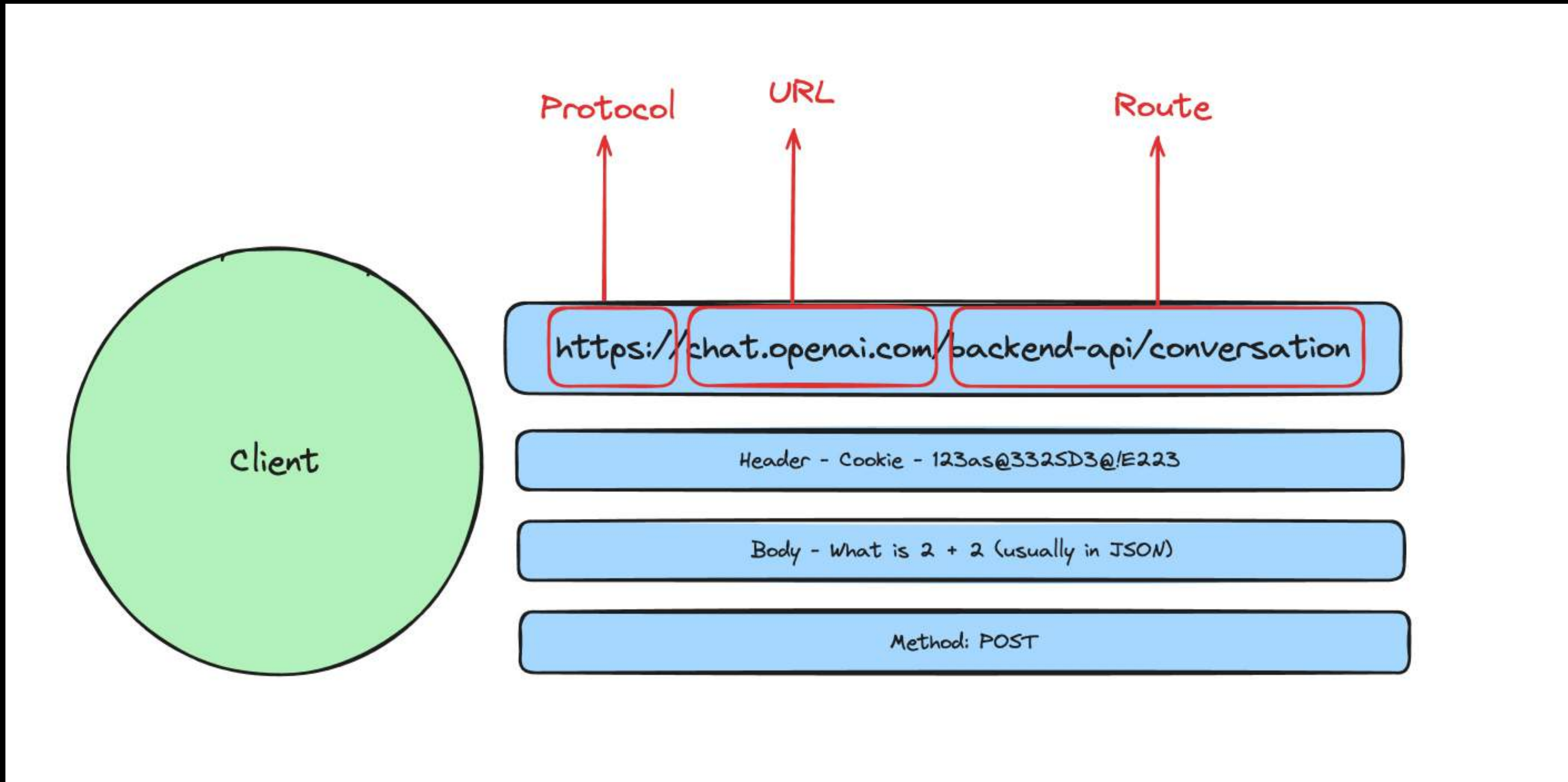
Things server needs to worry about





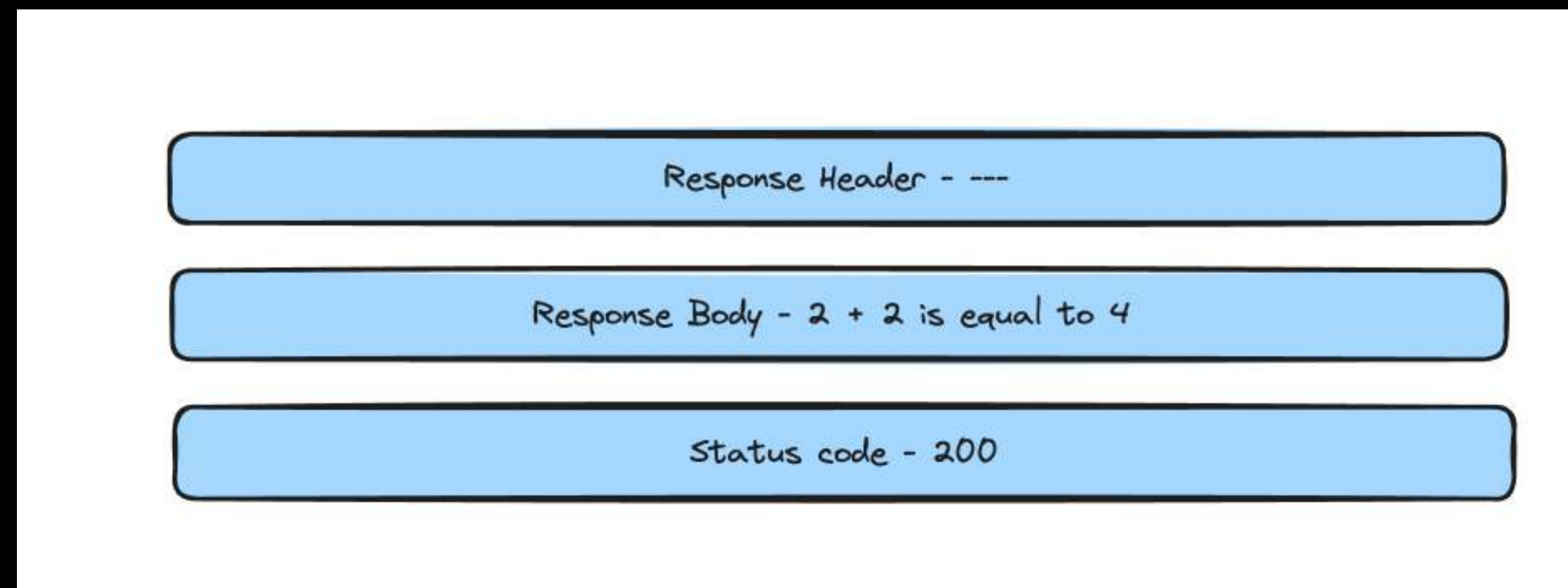
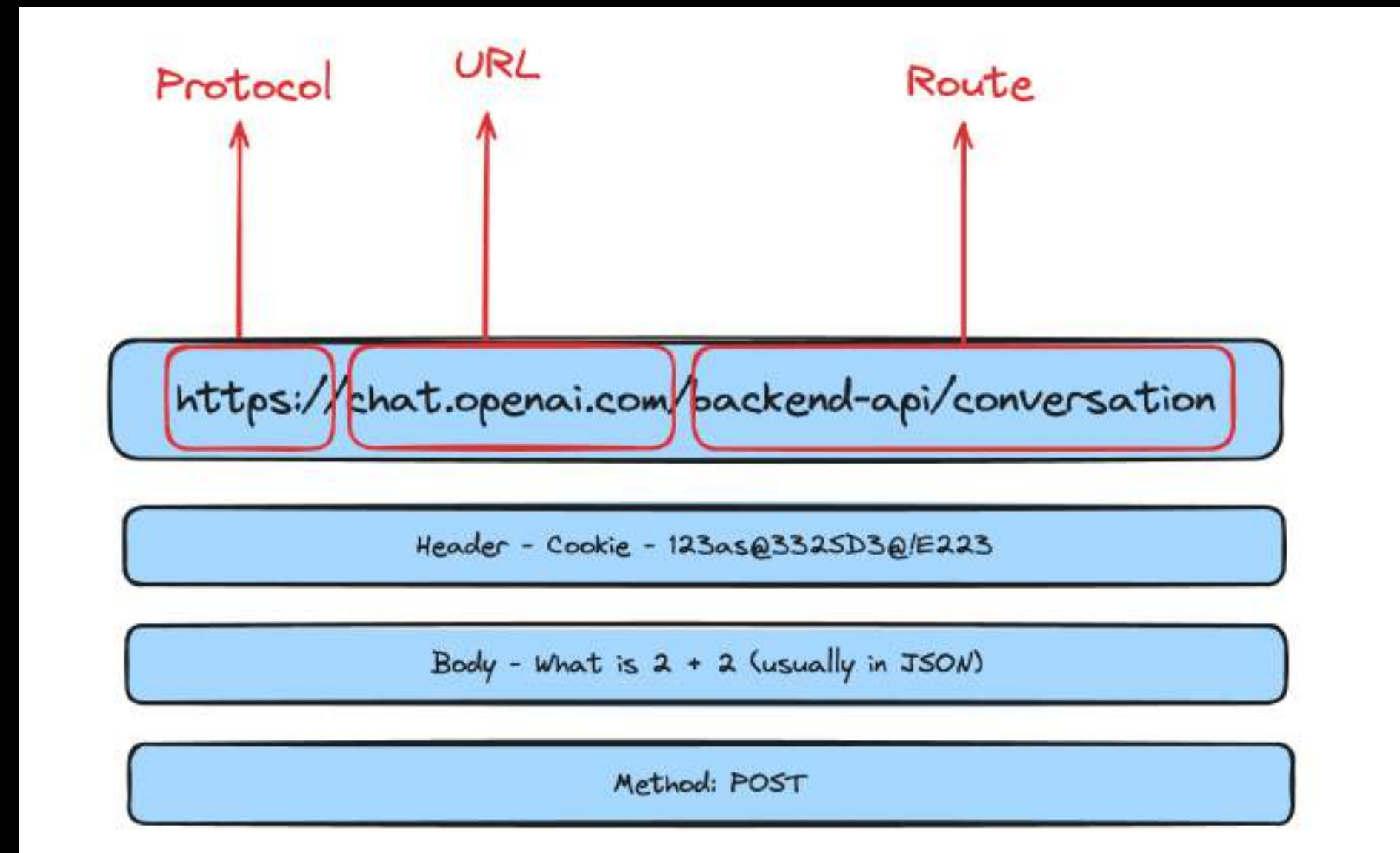
# HTTP Protocol

Usually communication would happen like this



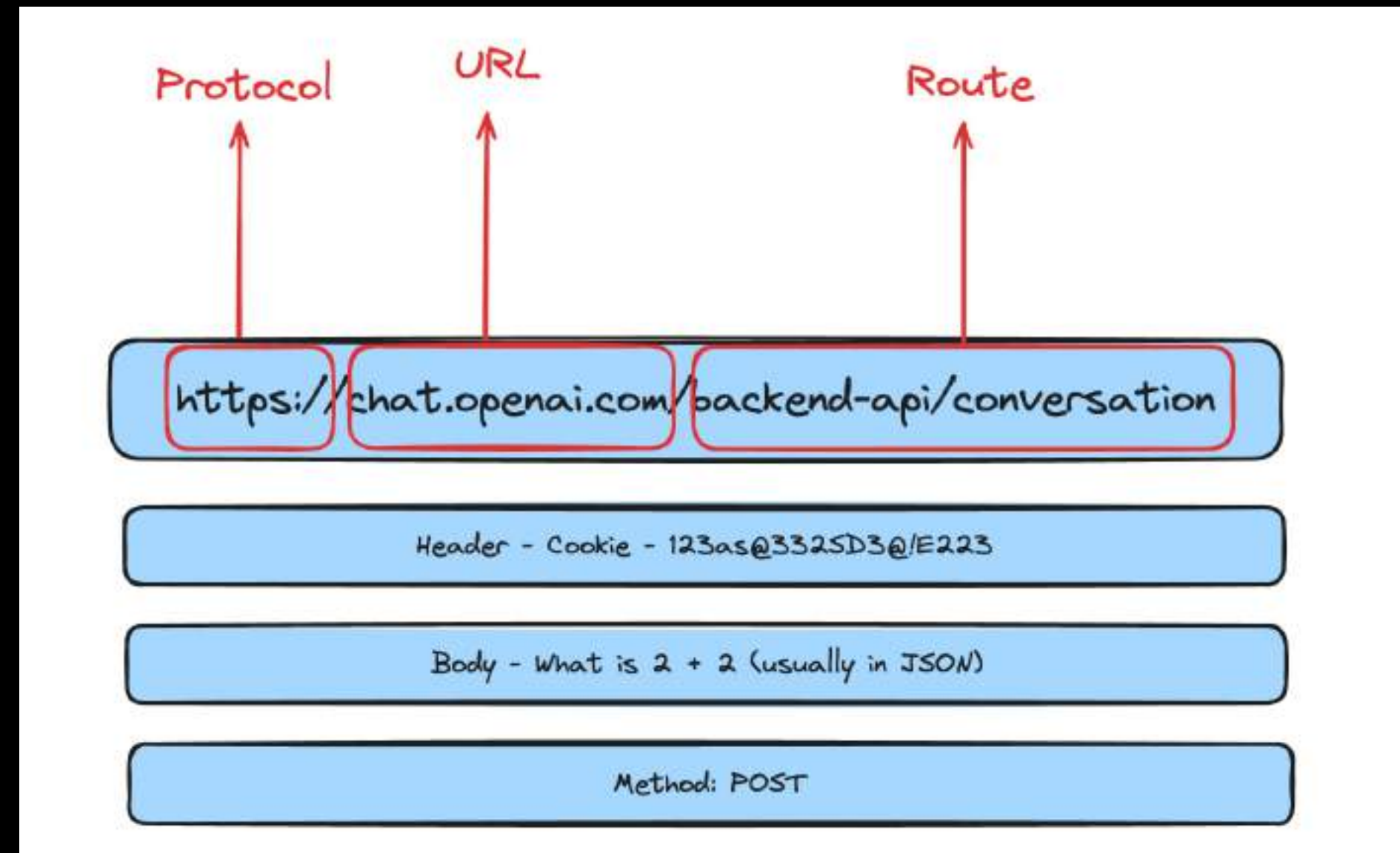
# HTTP Protocol

Usually communication would happen like this



# HTTP Protocol

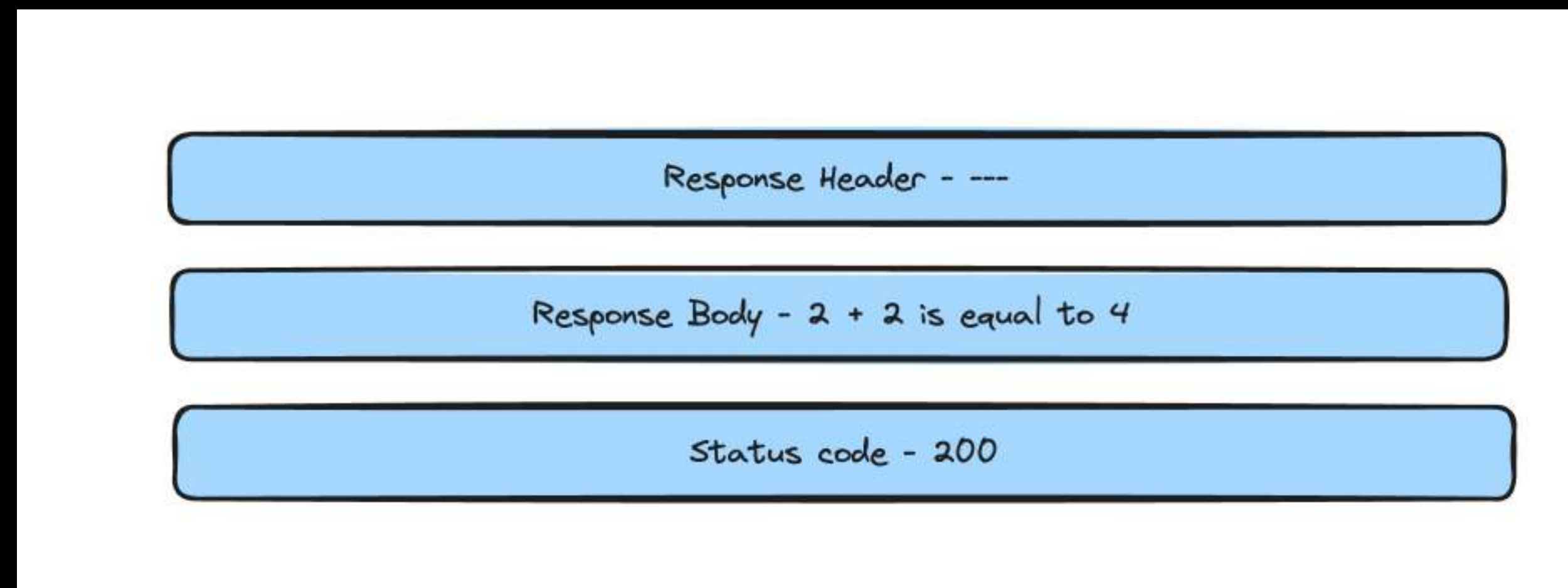
Usually communication would happen like this



Client

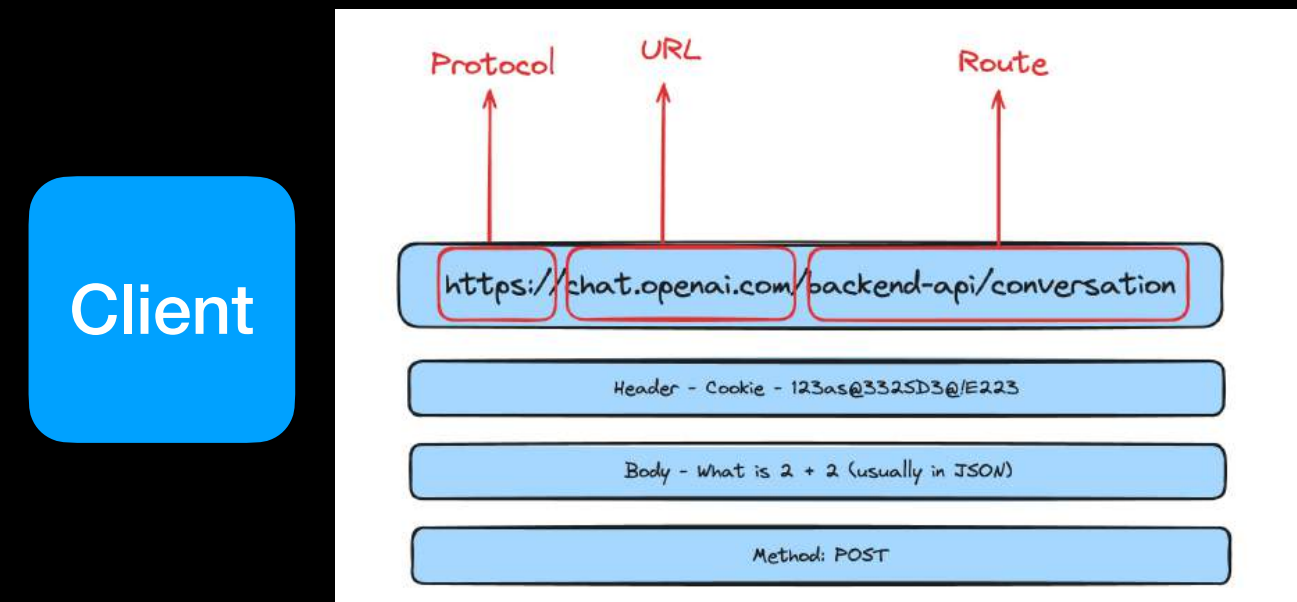


Server



# HTTP Protocol

Things that happen in your browser  
after you fire this request  
(we will get to how to fire request to a backend server later)



1. Browser parses the URL
2. Does a DNS Lookup (converts google.com to an IP)
3. Establishes a connection to the IP (does handshake...)

## What is DNS resolution

URLs are just like contacts in your phone

In the end, they map to an IP

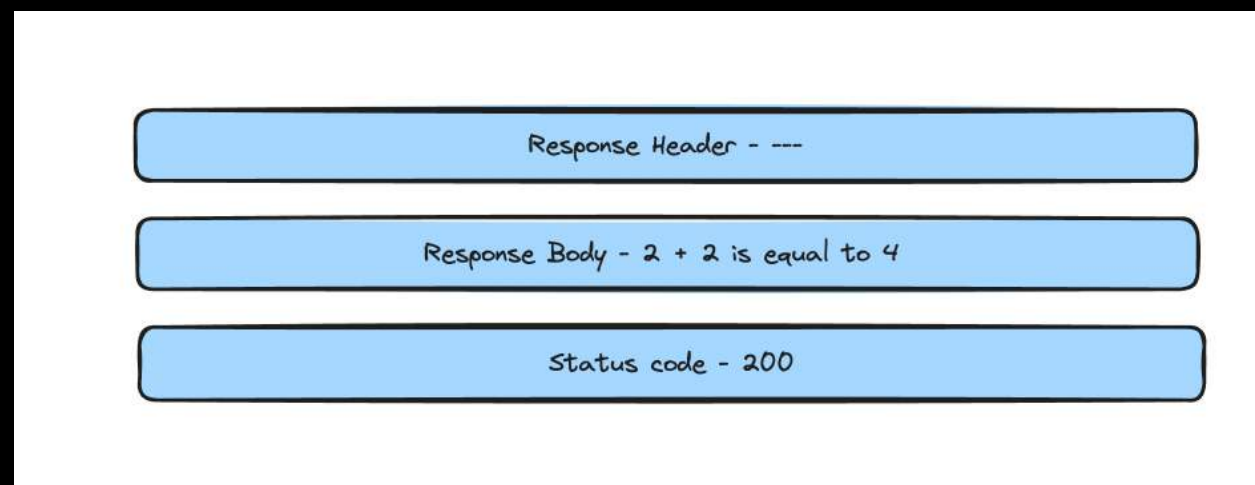
If you ever buy a URL of your own, you will need to point it to the IP of your server



# HTTP Protocol

Things that happen on your server after the request is received

Server






1. You get the inputs (route, body, headers)
2. You do some logic on the input, calculate the output
3. You return the output body, headers and status code

# HTTP Protocol

Lets see it in action

☐ Blocked requests ☐ 3rd-party requests

Name	✕ Headers	Payload	Preview	Response	Initiator	Timing	Cookies
 conversation	▼ General						
 r	<div>Request URL: https://chat.openai.com/backend-api/conversation</div> <div>Request Method: POST</div> <div>Status Code:  200 OK</div> <div>Remote Address: 104.18.37.228:443</div> <div>Referrer Policy: strict-origin-when-cross-origin</div>						
	▼ Response Headers						
	Access-Control-Allow-true						

2 / 10 requests | 23.4 kB / 26.6 kB trans

# HTTP Protocol

**What are the common methods you can send to your BE server?**

- 1. GET**
- 2. POST**
- 3. PUT**
- 4. DELETE**

# HTTP Protocol

**What are the common status codes the backend responds with?**

- 1. 200 - Everything is ok**
- 2. 404 - Page/route not found**
- 3. 403 - Authentication issues**
- 4. 500 - Internal server error**



# HTTP Protocol

**Why do we need status codes? Why can't we just return in the body something like success: true/false**  
**Why do we need so many types of request methods? Why can't just one work?**  
**Why do we need body/headers/query params, why can't just one work?**

**These are standard practises, you don't need all of it, but it is what is mentioned in the spec and hence is good to follow**

# HTTP Protocol

**Question at this point -**

**How do I create a HTTP server of my own?**

**How to I expose it over the internet like [chatgpt.com](https://chatgpt.com)**

# HTTP Protocol

Do you remember the fs library?  
We used to to read from a file

```
JS index.js > ...  
  
1  let a = 1;  
2  console.log(a);  
3  
4  ✓ fs.readFile("a.txt", "utf-8", (err, data) => {  
5    console.log("data read from the file is ");  
6    console.log(data);  
7  })  
8  
9  let ans = 0;  
10 ✓ for (let i = 0; i<100; i++) {  
11   ans = ans + i;  
12 }  
13 console.log(ans);|
```

# HTTP Protocol

**Similarly, there are many libraries that let you create HTTP Servers  
The most famous one is express**

**A great exercise to do is to create an HTTP server from scratch in C/C++  
It is out of scope for this course, but if you're looking for a challenge**

# HTTP Protocol

Library that we are using - **Express**

Lets create a simple HTTP Server

```
1  const express = require('express' 4.18.2 )
2  const app = express()
3  const port = 3000
4
5  app.get('/', (req, res) => {
6    res.send('Hello World!')
7  })
8
9  app.listen(port, () => {
10    console.log(`Example app listening on port ${port}`)
11  })
```

<https://expressjs.com/en/starter/hello-world.html>