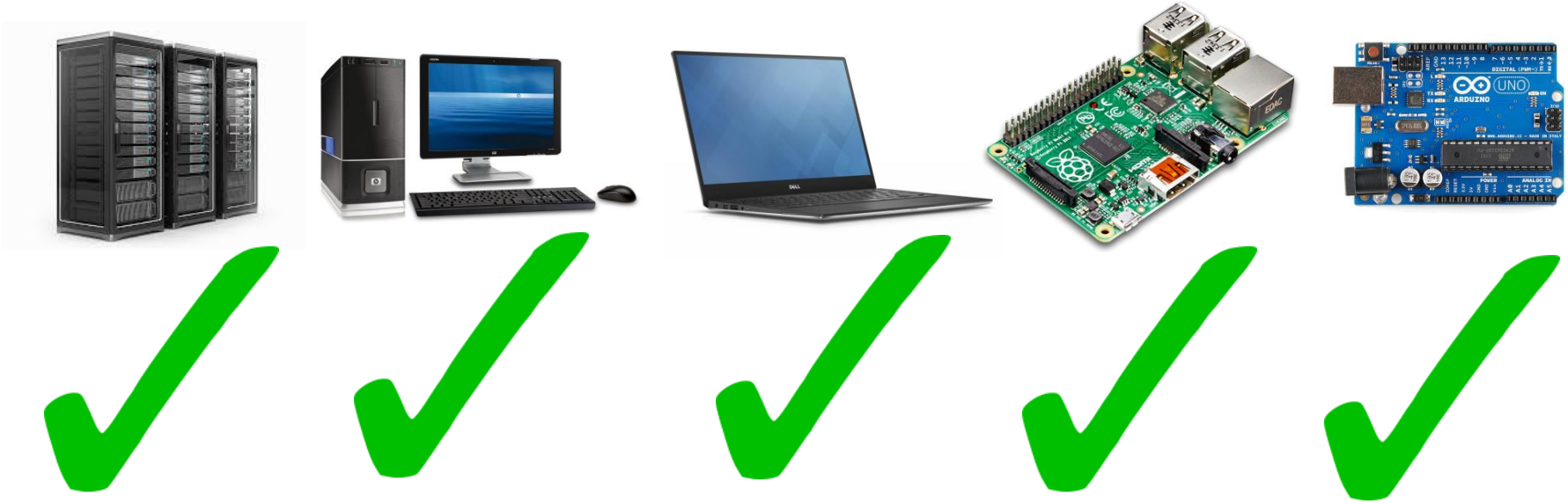
HTTP and Servers

101 on how the internet works

What is a Server?



- Any “Computer” with a way to store data, run processes, and can connect to other computers can be a Server
- Raspberry Pi is a “Full Computer”
- Difference between a Raspberry Pi and my Laptop is the performance it can handle

How do I run a server

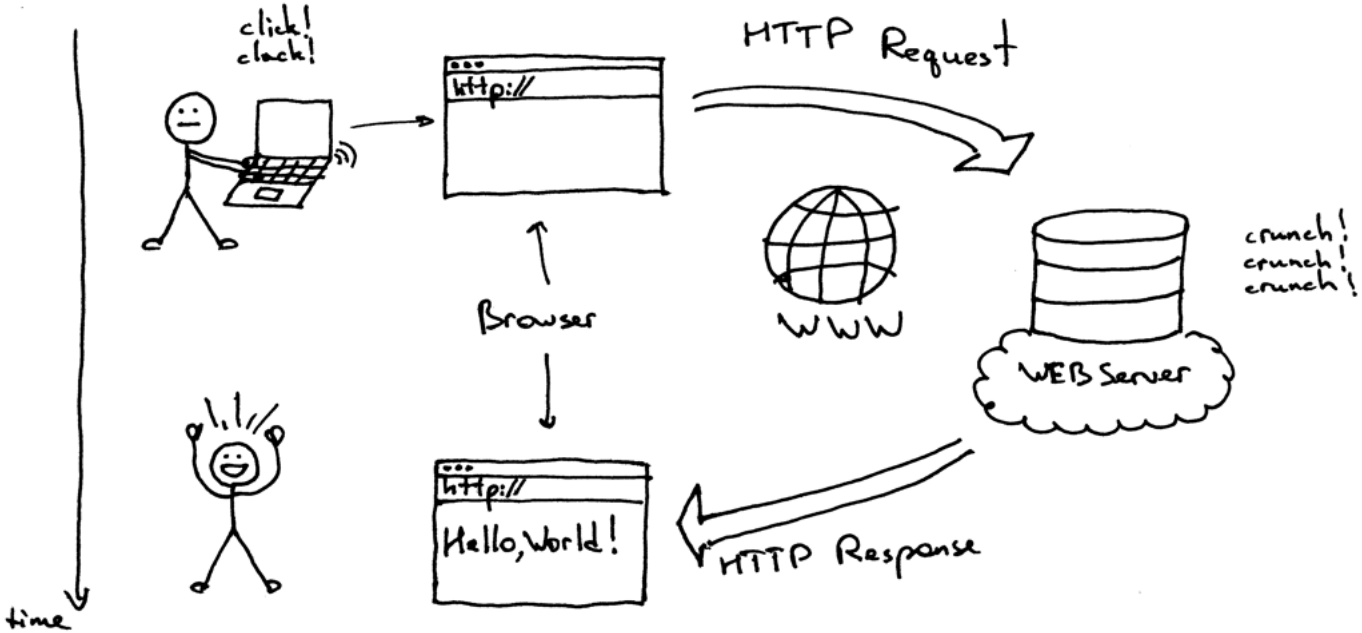
- You should use a Server Framework that handle request at a high level.
- People have spent years working on them, they are good, so use them.

- …But today you will learn how to create your own server on a single socket that parses the text of the string.
- Hey, someone needs to understand how it works

How do I talk to the server

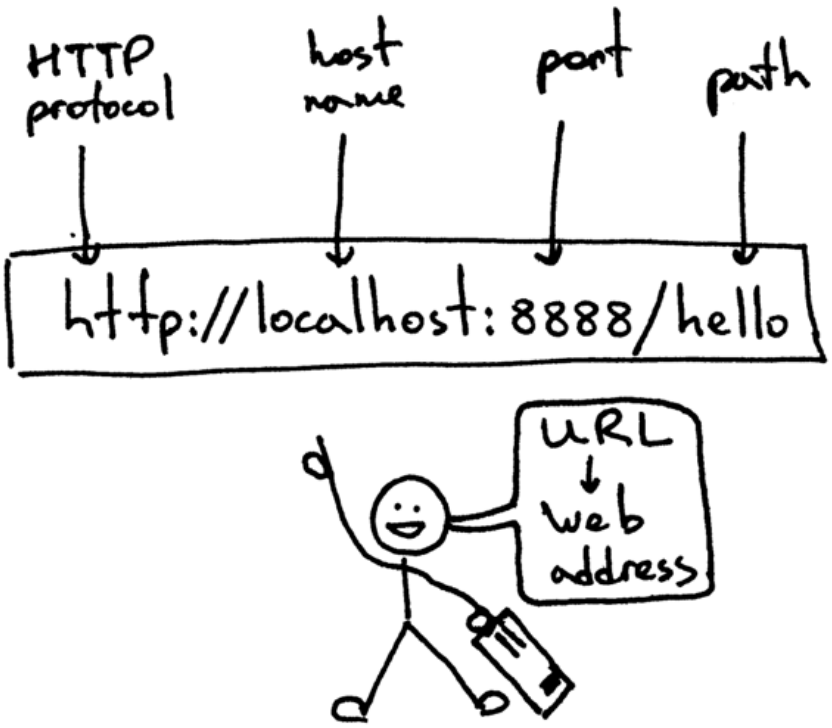
- You can send anything you want at a server if you have the IP address, but there are standard **Protocols** that are agreed on otherwise I wouldn't know what format the information you are sending is sent in
- HTTP
  - Stands for Hypertext Transfer Protocol
  - Way of sending text to servers and getting a response back from them
  - This literally is what the Internet/world runs off of

How you are use to talking to servers



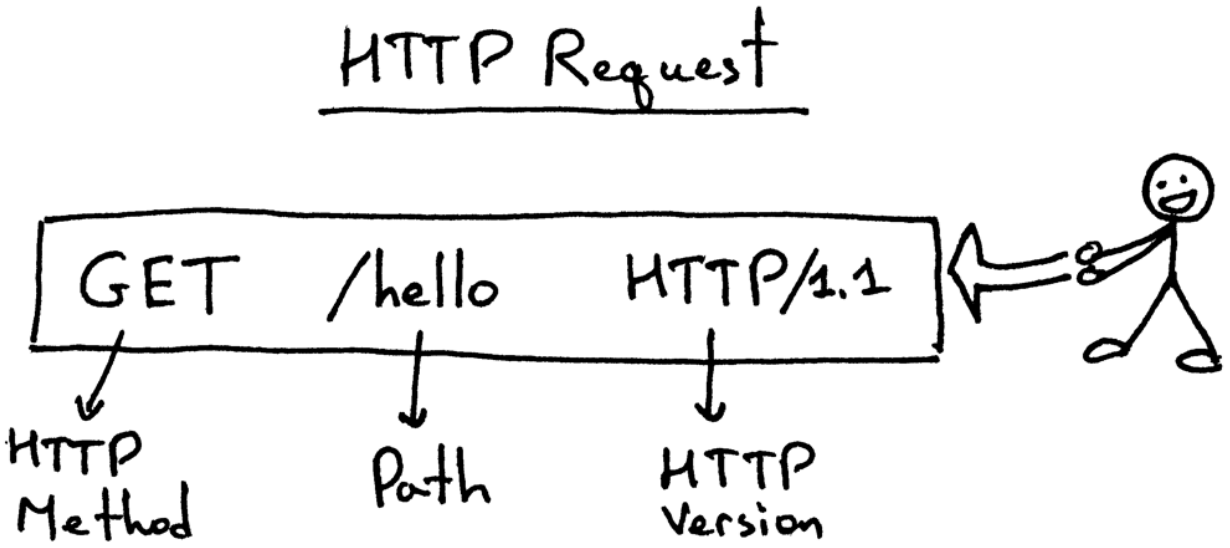
- Main idea is a **Client** sends a **Server** a **HTTP Request** where the server does its thing and sends back a **HTTP Response** to the **Client**
- This **HTTP Response** can be the HTML to a website, info that the server executed its job correctly or hit an error, it could send you back a joke in plain text, its all up to the server (also known as back-end code)

How to message the server



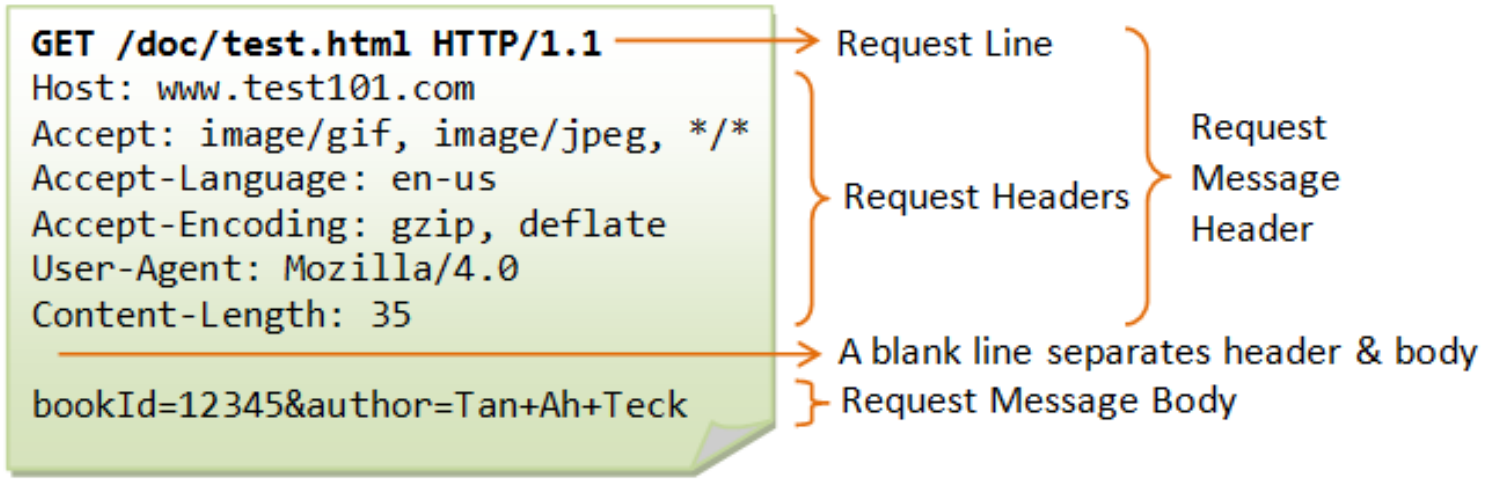
- We use a URL
- This just tell what server to find and where in the server to send the message
- Can be used in a Web Browser, Dev tool (Postman), or a custom made client request from your Raspberry Pi

The Request



- Many methods
  - We will stick to just GET and POST
  - Path is structured like Linux file location
  - HTTP 1.1 is old and still going strong

The Request - Header



- Server first reads Request Line
- Header information is where we can send details in a “Key-Value” fashion
- Everything is separated by new lines

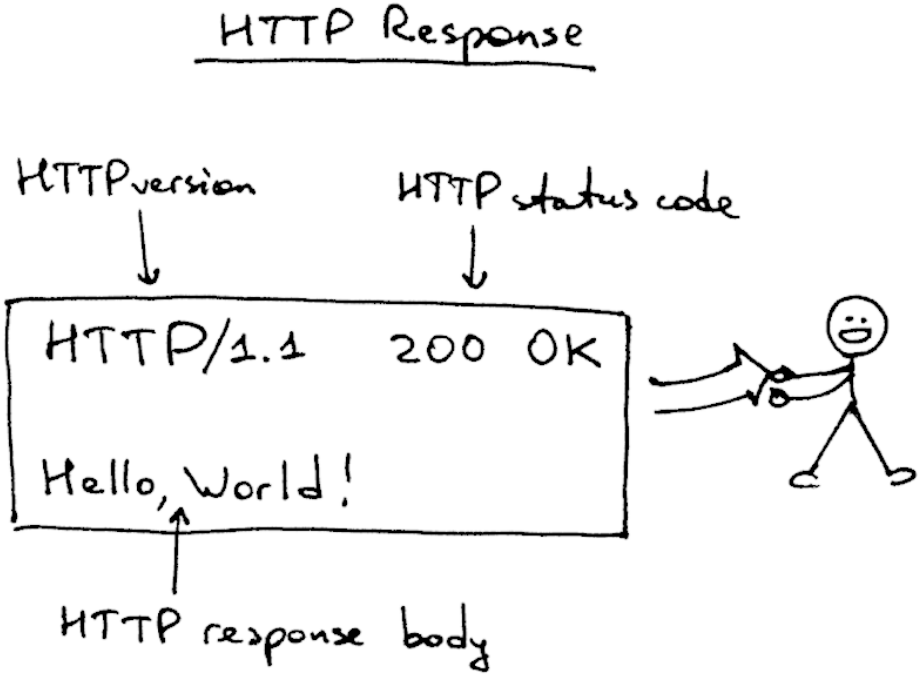
The Request – Verb types

- Verbs indicate **DESIRED** actions, its still up to server to handle, but there are some set standards
- GET
  - All information is in URL/Request Line
  - http://myServer:8888/path/with/info/about/the/get/call
  - Server might have parameters set depending what you put on a specific part of a path
- POST
  - Lets server know you are sending a **Body** with the request in JSON format
  - This is used to send larger amount of data

JSON? Ok quick 2 minute detour

- JavaScript Object Notation
  - Way of holding data in a “Key-Value” fashion
  - Similar to Structs in C/C++
  - Var example = { “Name” : “Spencer”,
    “Age” : 21,
    “isFunny” : false }
  - example.Name == “Spencer”
  - example.Age == 21
  - example.isFunny == false
  - “Name”, “Age”, “isFunny” are the **Keys**
  - “Spencer”, 21, false are the **Value** and can be any type

The Response

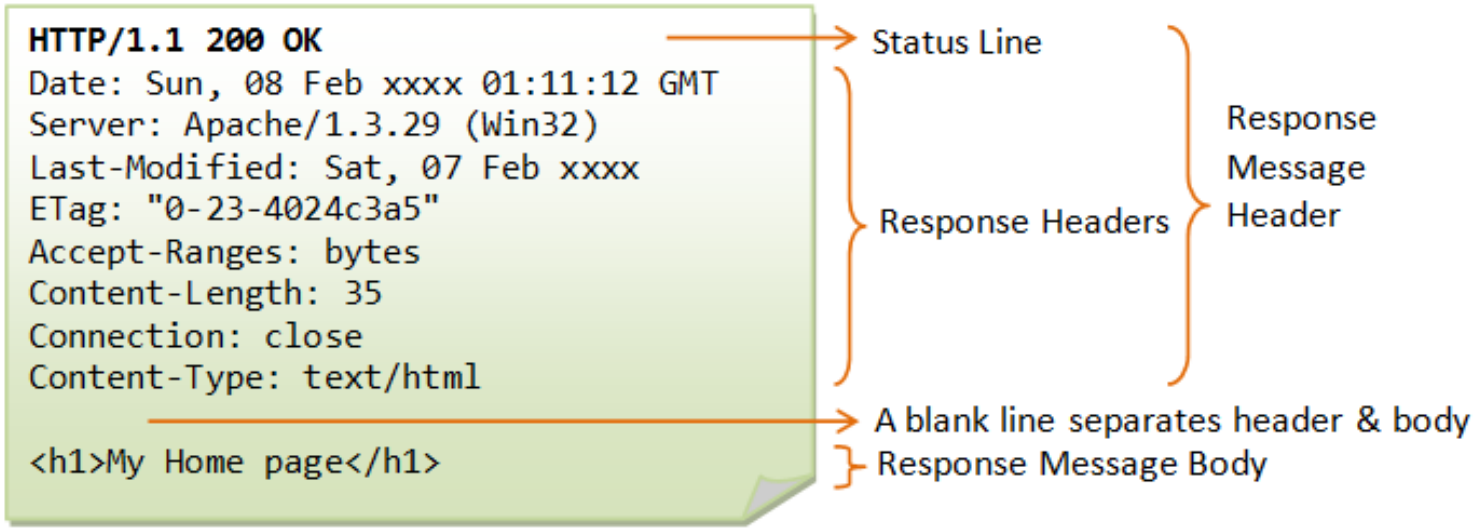


- The server will decide what it will do and send back a response
  - If it doesn't the client will be in a infinite loop waiting

The Response – Status Code

- Status Codes are one of those things where you writing your server can send back any number. There a **very standard** number system… so don't go crazy
- Status codes to know:
  - **200 OK** - "I got your request and did something" but doesn't mean it "Correctly" did it
  - **400 Bad Request** – The Client screwed up what it sent
  - **404 Not Found** - it couldn't find the path you sent it
  - **500 Internal Server Error** – The Server screwed up what it was trying to do

The Response – Looks like the Request



- Same format as the Request
- Can you smell the string parsing yet?