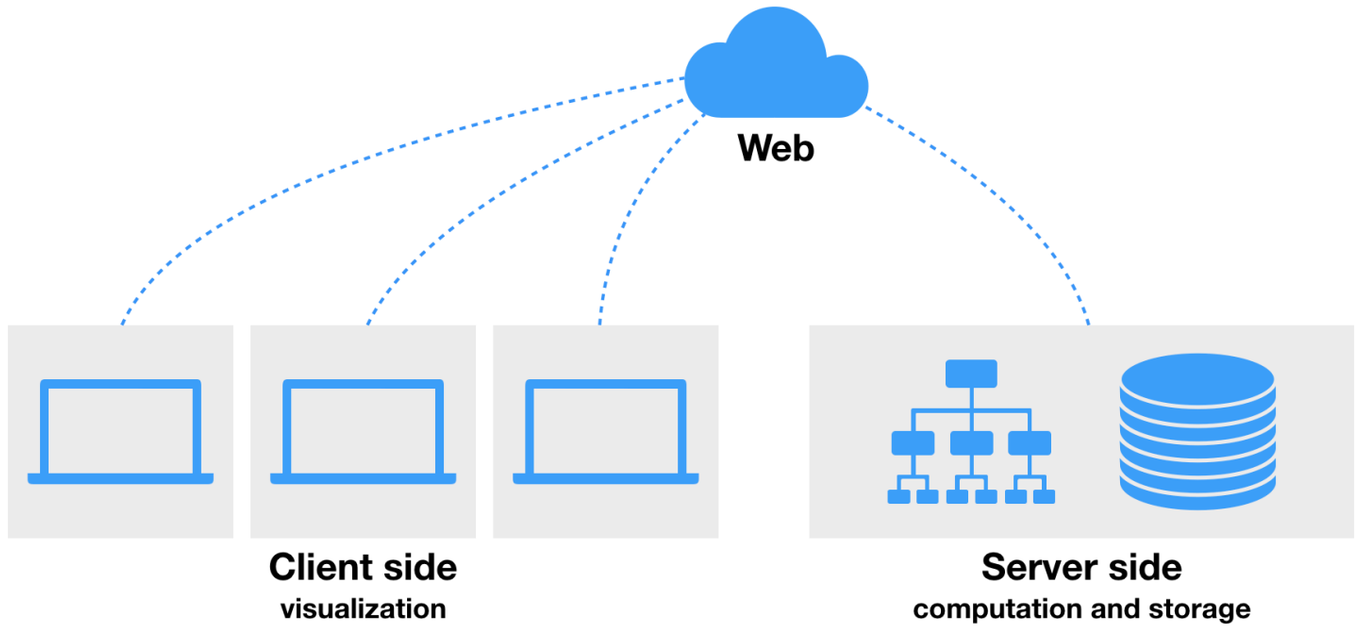
Lesson - 10

Client-Server Architecture. HTTP Protocol



SkillUp by Dobrea Vladislav

Lesson Plan

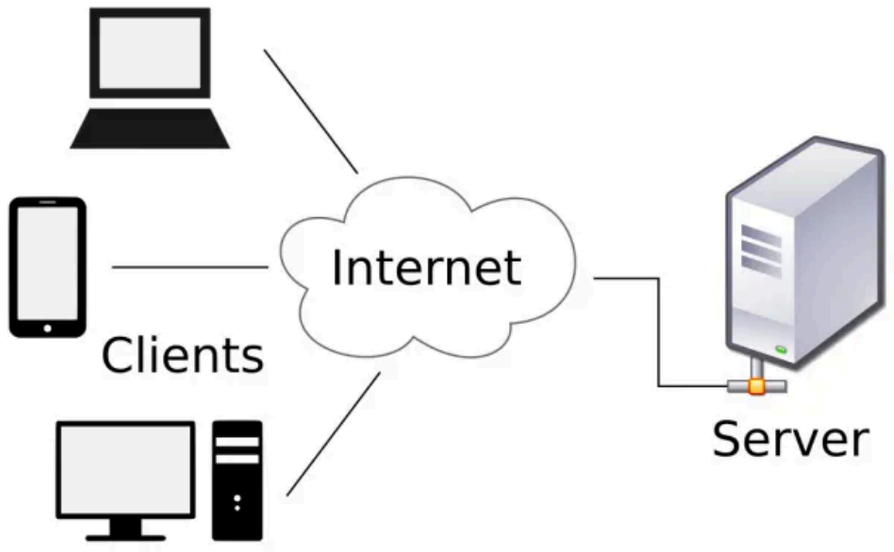
1. **Review HW**
2. **Client-Server Architecture**
3. **HTTP Protocol**
4. **Netlify**

Client - Server Model

The **Client-server model** is a distributed application structure that partitions task or workload between the providers of a resource or service, called servers, and service requesters called clients. In the client-server architecture, when the client computer sends a request for data to the server through the internet, the server accepts the requested process and deliver the data packets requested back to the client.

The **client-server model** describes how a server provides resources and services to one or more clients.

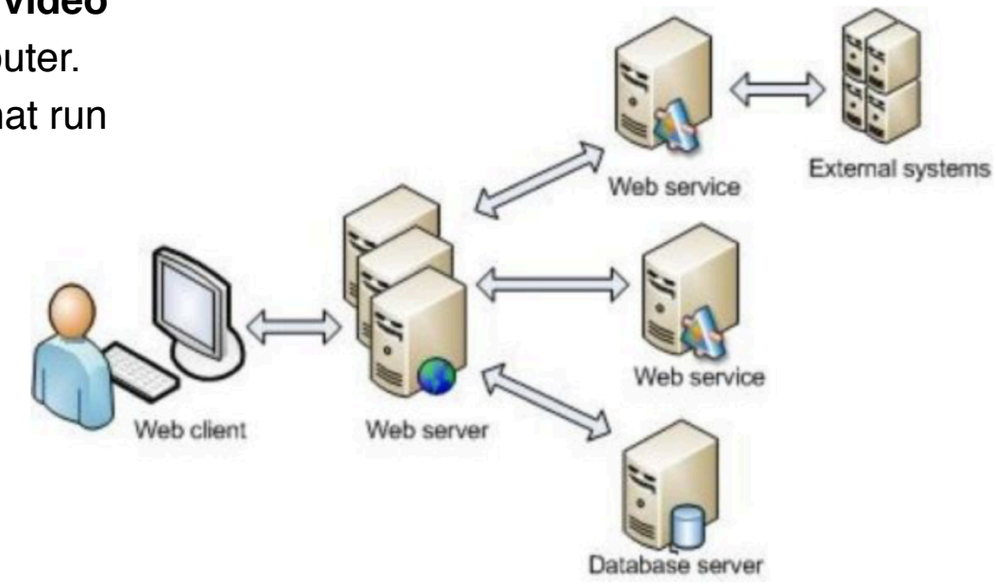
Clients do not share any of their resources. Examples of Client-Server Model are Email, World Wide Web, etc.



What is Client?

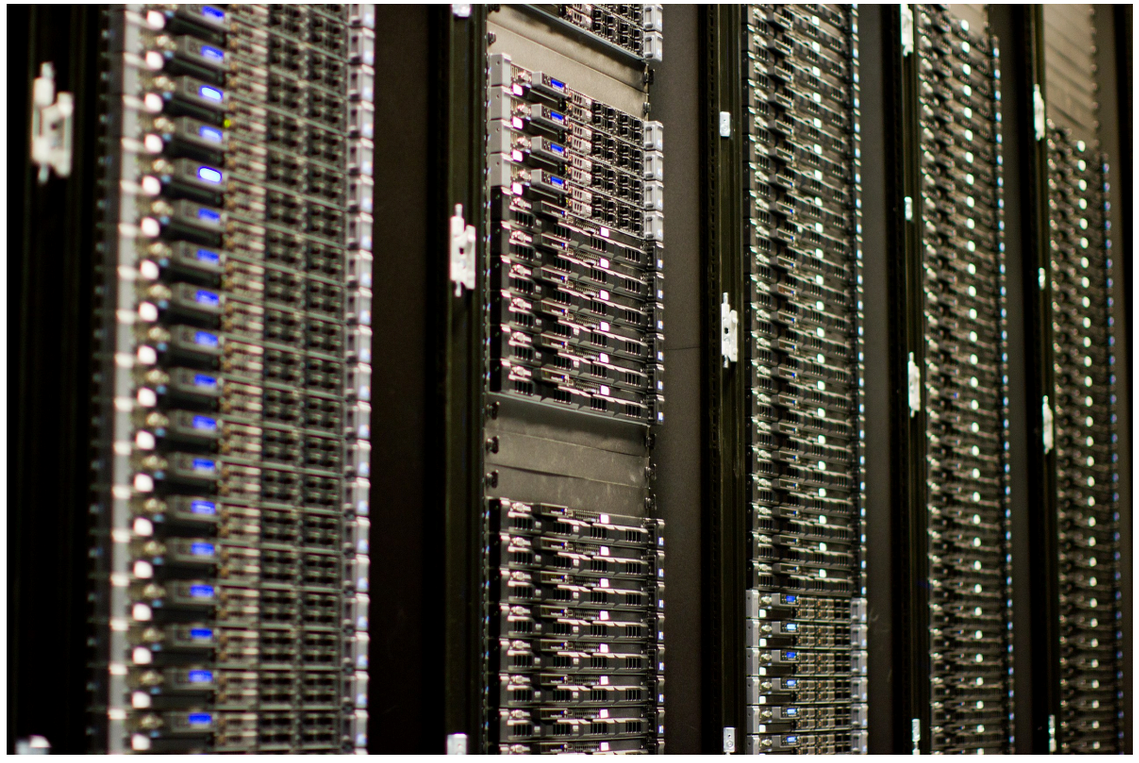
A client is a computer or a program that, as part of its operation, relies on sending a request to another program or a computer hardware or software that accesses a service made available by a server.
Client communicates with server through a specific protocol.

For example, web browsers are clients that connect to web servers and retrieve web pages for display. **Email clients** retrieve email from mail servers. **Online chat** uses a variety of clients, which vary on the chat protocol being used. **Multiplayer video games** or online video games may run as a client on each computer.
The term "client" may also be applied to computers or devices that run the client software or users that use the client software.



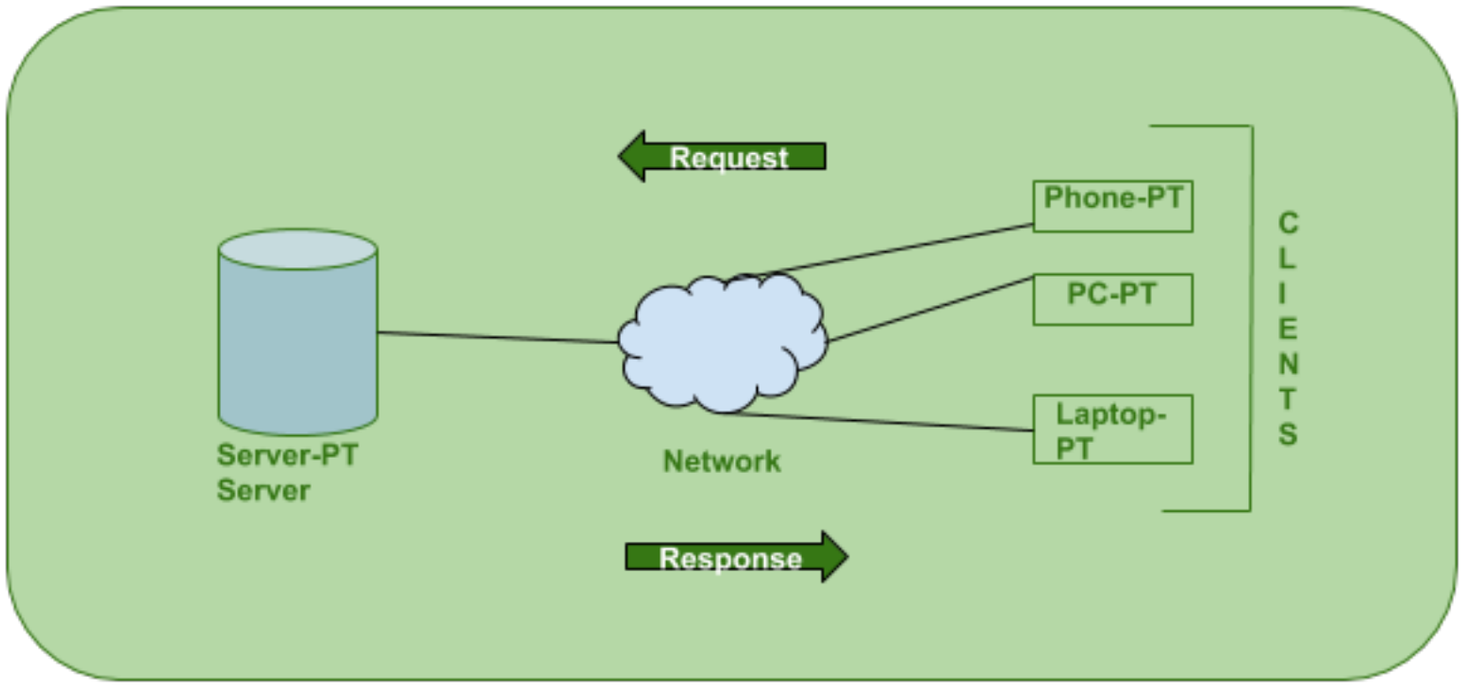
What it Server?

In computing, a **server** is a computer program or a device that provides functionality for other programs or devices, called "clients". This architecture is called the client–server model, and a single overall computation is distributed across multiple processes or devices. Servers can provide various functionalities, often called "services", such as sharing data or resources among multiple clients, or performing computation for a client.



How Client-Server model works?

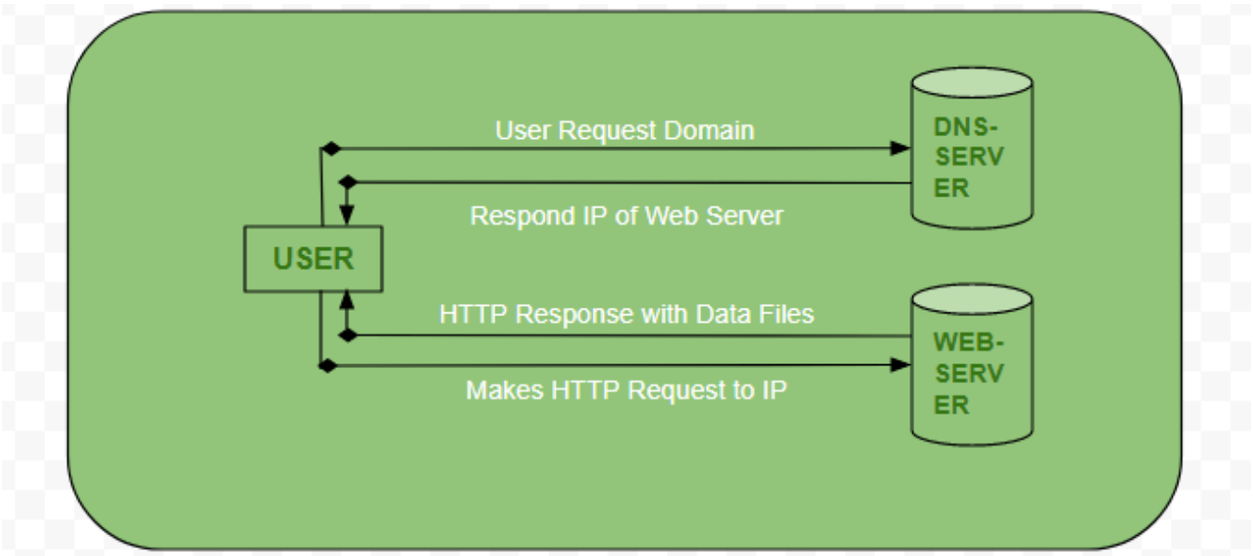
On a very high level overview it's that simple:
Client initiates a **request** to **Server** and receives a **response.**



How Client interacts with Server

There are few steps to follow to interacts with the servers a client.

- User enters the **URL**(Uniform Resource Locator) of the website or file. The Browser then requests the **DNS**(DOMAIN NAME SYSTEM) Server.
- **DNS Server** lookup for the address of the **WEB Server**.
- **DNS Server** responds with the **IP address** of the **WEB Server**.
- Browser sends over an **HTTP/HTTPS** request to **WEB Server's IP** (provided by **DNS server**).
- Server sends over the necessary files of the website.
- Browser then renders the files and the website is displayed. This rendering is done with the help of **DOM** (Document Object Model) interpreter, **CSS** interpreter and **JS Engine** collectively known as the **JIT** or (Just in Time) Compilers.



Advantages of Client-Server

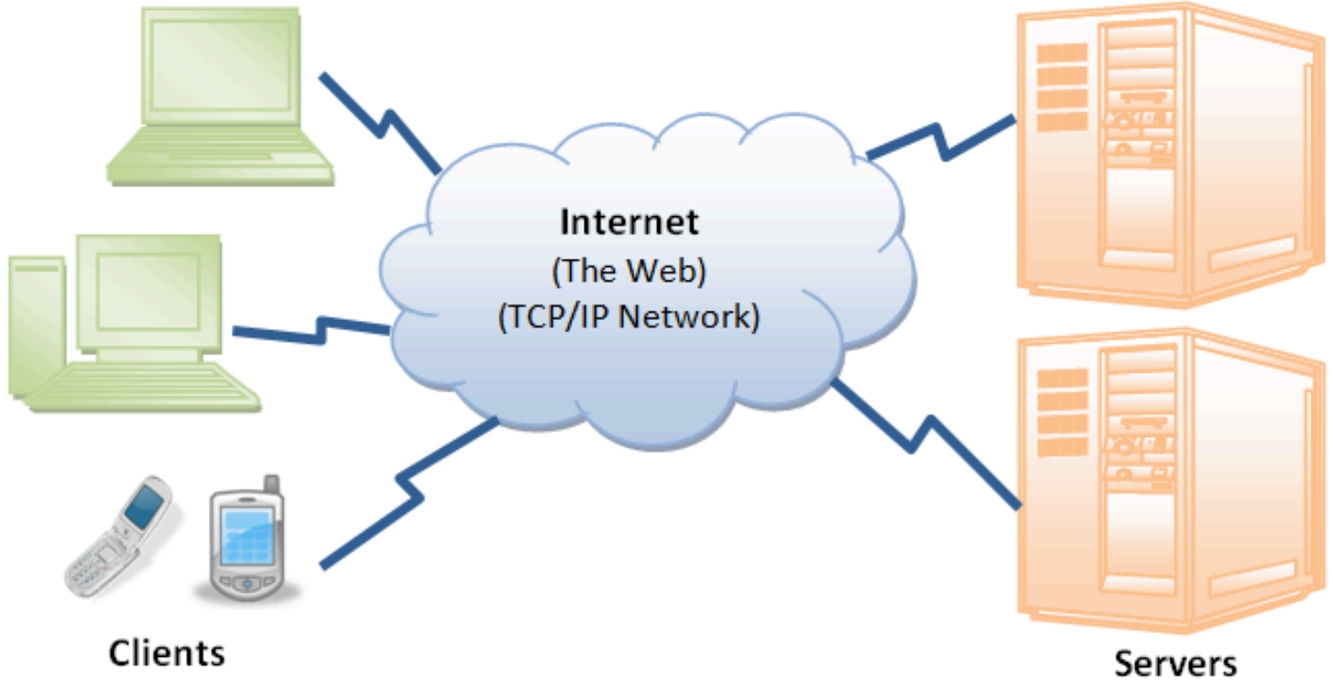
- **Centralized:** Centralized back-up is possible in client-server networks, i.e., all the data is stored in a server.
- **Security:** These networks are more secure as all the shared resources are centrally administered.
- **Performance:** The use of the dedicated server increases the speed of sharing resources. This increases the performance of the overall system.
- **Scalability:** We can increase the number of clients and servers separately, i.e., the new element can be added, or we can add a new node in a network at any time.

Disadvantages of Client-Server

- **Traffic Congestion** is a big problem in Client/Server networks. When a large number of clients send requests to the same server may cause the problem of Traffic congestion.
- It does not have a robustness of a network, i.e., when the server is down, then the client requests cannot be met.
- A client/server network is very decisive. Sometimes, regular computer hardware does not serve a certain number of clients. In such situations, specific hardware is required at the server side to complete the work.
- Sometimes the resources exist in the server but may not exist in the client. For example, If the application is web, then we cannot take the print out directly on printers without taking out the print view window on the web.

Communication

Many applications are running concurrently over the Web, such as web browsing/surfing, e-mail, file transfer, audio & video streaming, and so on. In order for proper communication to take place between the client and the server, these applications must agree on a specific application-level **protocol** such as HTTP, FTP, SMTP, POP, and etc.

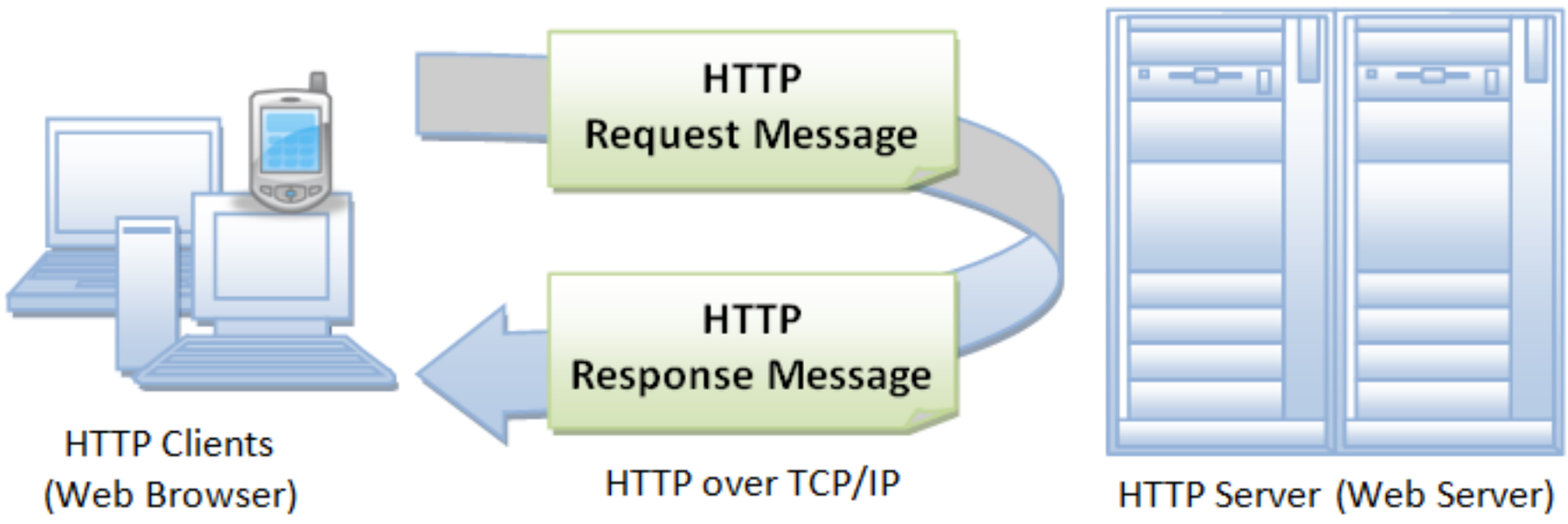


HTTP - Most popular protocol

HTTP (Hypertext Transfer Protocol) is perhaps the most popular application protocol used in the Internet (or The WEB).

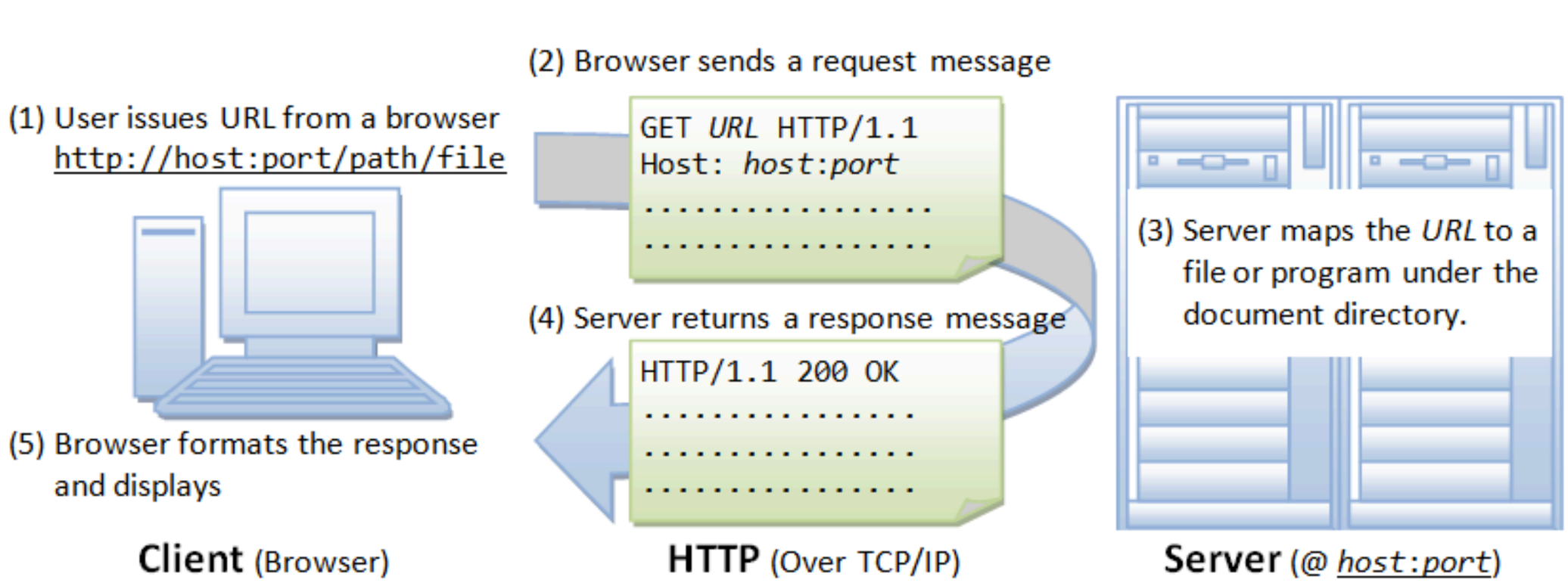
HTTP is an asymmetric request-response client-server protocol as illustrated. An HTTP client sends a request message to an HTTP server. The server, in turn, returns a response message. In other words, HTTP is a pull protocol, the client pulls information from the server (instead of server pushes information down to the client).

HTTP is a stateless protocol. In other words, the current request does not know what has been done in the previous requests.



Browser

Whenever you issue a URL from your browser to get a web resource using HTTP, e.g. http://www.nowhere123.com/index.html, the browser turns the URL into a request message and sends it to the HTTP server. The HTTP server interprets the request message, and returns you an appropriate response message, which is either the resource you requested or an error message. This process is illustrated below



URL - Uniform resource locator

A URL (Uniform Resource Locator) is used to uniquely identify a resource over the web. URL has the following syntax:

Example: *protocol://hostname:port/path-and-file-name*

Here are 4 parts in a URL:

1. Protocol: The application-level protocol used by the client and server, e.g., HTTP, FTP, and telnet.
2. Hostname: The DNS domain name (e.g., www.nowhere123.com) or IP address (e.g., 192.128.1.2) of the server.
3. Port: The TCP port number that the server is listening for incoming requests from the clients.
4. Path-and-file-name: The name and location of the requested resource, under the server document base directory.

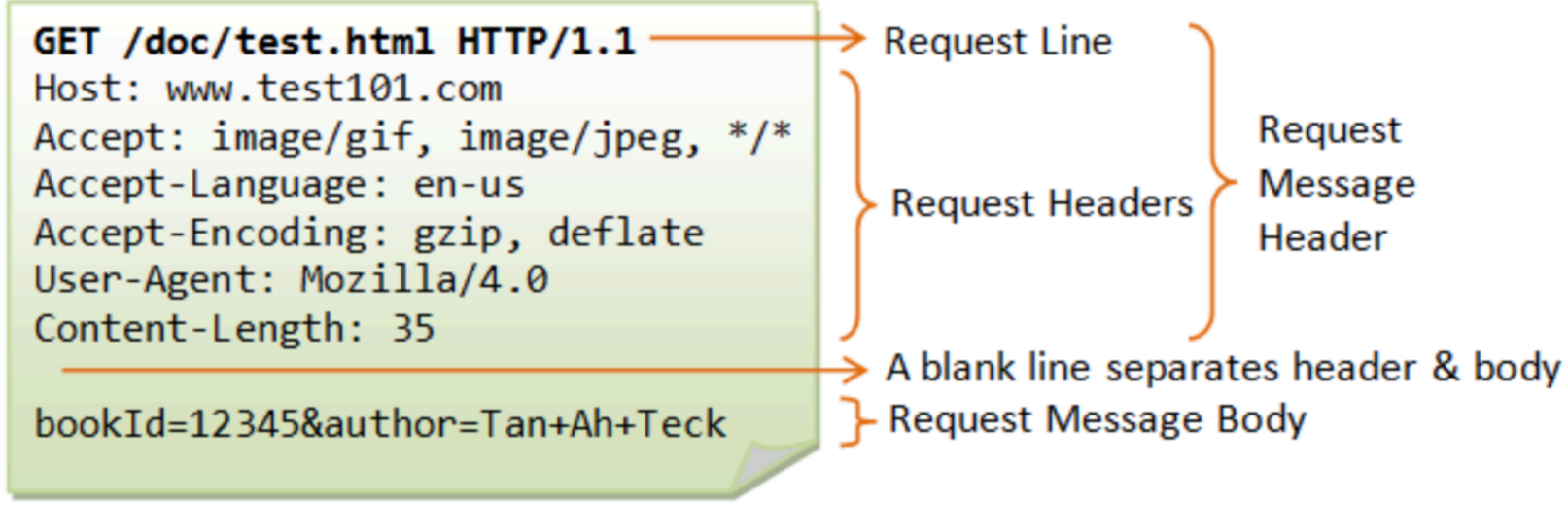
URL To HTTP Request Message

As mentioned, whenever you enter a URL in the address box of the browser, the browser translates the URL into a request message according to the specified protocol; and sends the request message to the server.

For example, the browser translated the URL `http://www.nowhere123.com/docs/index.html` into the following request message:

```
GET /docs/index.html HTTP/1.1
Host: www.nowhere123.com
Accept: image/gif, image/jpeg, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
(blank line)
```

Request Message Detailed



Response Message

When this request message reaches the server, the server can take either one of these actions:

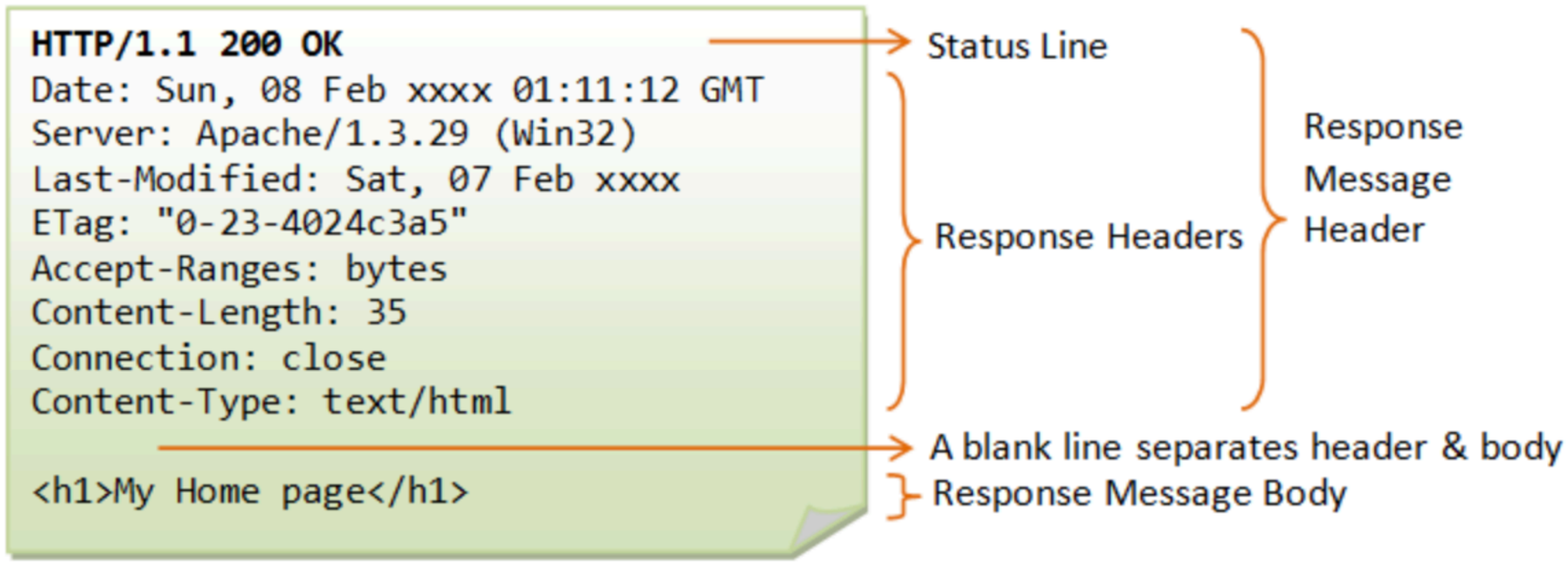
1. The server interprets the request received, maps the request into a file under the server's document directory, and returns the file requested to the client.
2. The server interprets the request received, maps the request into a program kept in the server, executes the program, and returns the output of the program to the client.
3. The request cannot be satisfied, the server returns an error message.

An example of the HTTP response message is as shown:

```
HTTP/1.1 200 OK
Date: Sun, 18 Oct 2009 08:56:53 GMT
Server: Apache/2.2.14 (Win32)
Last-Modified: Sat, 20 Nov 2004 07:16:26 GMT
ETag: "10000000565a5-2c-3e94b66c2e680"
Accept-Ranges: bytes
Content-Length: 44
Connection: close
Content-Type: text/html
X-Pad: avoid browser bug

<html><body><h1>It works!</h1></body></html>
```

Response Message Detailed



HTTP Request Methods

HTTP protocol defines a set of request methods. A client can use one of these request methods to send a request message to an HTTP server. The methods are:

- **GET**: A client can use the GET request to get a web resource from the server.
- **HEAD**: A client can use the HEAD request to get the header that a GET request would have obtained. Since the header contains the last-modified date of the data, this can be used to check against the local cache copy.
- **POST**: Used to post data up to the web server.
- **PUT**: Ask the server to store the data.
- **DELETE**: Ask the server to delete the data.
- **TRACE**: Ask the server to return a diagnostic trace of the actions it takes.
- **OPTIONS**: Ask the server to return the list of request methods it supports.
- **CONNECT**: Used to tell a proxy to make a connection to another host and simply reply the content, without attempting to parse or cache it. This is often used to make SSL connection through the proxy.
- Other extension methods.

GET Request Method

GET is the most common HTTP request method. A client can use the GET request method to request (or "get") for a piece of resource from an HTTP server. A GET request message takes the following syntax:

- The keyword GET is case sensitive and must be in uppercase.
- request-URI: specifies the path of resource requested, which must begin from the root "/" of the document base directory.
- HTTP-version: Either HTTP/1.0 or HTTP/1.1. This client negotiates the protocol to be used for the current session. For example, the client may request to use HTTP/1.1. If the server does not support HTTP/1.1, it may inform the client in the response to use HTTP/1.0.
- The client uses the optional request headers (such as Accept, Accept-Language, and etc) to negotiate with the server and ask the server to deliver the preferred contents (e.g., in the language that the client preferred).
- GET request message has an optional request body which contains the query string (to be explained later).

POST Request Method

POST request method is used to "post" additional data up to the server (e.g., submitting HTML form data or uploading a file). Issuing an HTTP URL from the browser always triggers a GET request. To trigger a POST request, you can use an HTML form with attribute method="post" or write your own network program. For submitting HTML form data, POST request is the same as the GET request except that the URL-encoded query string is sent in the request body, rather than appended behind the request-URI.

POST vs GET for Submitting Form Data

As mentioned in the previous section, POST request has the following advantage compared with the GET request in sending the query string:

- The amount of data that can be posted is unlimited, as they are kept in the request body, which is often sent to the server in a separate data stream.
- The query string is not shown on the address box of the browser.

Note that although the password is not shown on the browser's address box, it is transmitted to the server in clear text, and subjected to network sniffing. Hence, sending password using a POST request is absolutely not secure.

POST vs GET

	GET	POST
BACK button/Reload	Harmless	Data will be re-submitted (the browser should alert the user that the data are about to be re-submitted)
Bookmarked	Can be bookmarked	Cannot be bookmarked
Cached	Can be cached	Not cached
Encoding type	application/x-www-form-urlencoded	application/x-www-form-urlencoded or multipart/form-data. Use multipart encoding for binary data
History	Parameters remain in browser history	Parameters are not saved in browser history
Restrictions on data length	Yes, when sending data, the GET method adds the data to the URL; and the length of a URL is limited (maximum URL length is 2048 characters)	No restrictions
Restrictions on data type	Only ASCII characters allowed	No restrictions. Binary data is also allowed
Security	GET is less secure compared to POST because data sent is part of the URL Never use GET when sending passwords or other sensitive information!	POST is a little safer than GET because the parameters are not stored in browser history or in web server logs
Visibility	Data is visible to everyone in the URL	Data is not displayed in the URL

Netlify

A cloud computing company that offers hosting and serverless backend services for web applications and static websites.

Learning Resources

1. **Client-Server Model**
    1. **https://www.javatpoint.com/computer-network-client-and-server-model**
    2. **About DNS https://www.javatpoint.com/computer-network-dns**
    3. **Good website about computer networks in general**
       https://www.javatpoint.com/computer-network-architecture
2. **HTTP Basics**
    1. **https://www.ntu.edu.sg/home/ehchua/programming/webprogramming/http_basics.html**
3. **Netlify**
    1. **https://www.netlify.com/**

Home Work

1. Read Learning resources
2. Netlify
    1. Create a Netlify account (login via GitHub)
    2. Create a new site, and connect it to GitHub repo project (that you did as a HW for this lesson)