

# Biomedical Wearable Technologies for Healthcare and Wellbeing

## HTTP

---

A.Y. 2023-2024  
Giacomo Cappon



# Outline

---

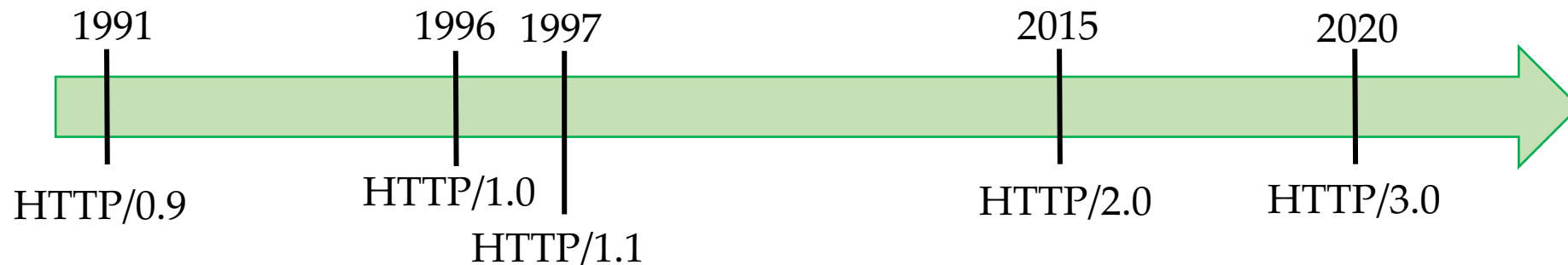
➤ HyperText Transfer Protocol (HTTP)

➤ Web applications

# The HyperText Transfer Protocol (HTTP)

---

- **HyperText Transfer Protocol (HTTP):** the main web communication protocol for exchanging information between servers and clients.
- Simple **request-response protocol**, proper of the application layer, running over TCP.
- It specifies what messages clients may send to servers and what responses they get back in return.
- Most of Web currently use HTTP/1.1 and HTTP/2.0, which share the same syntax.

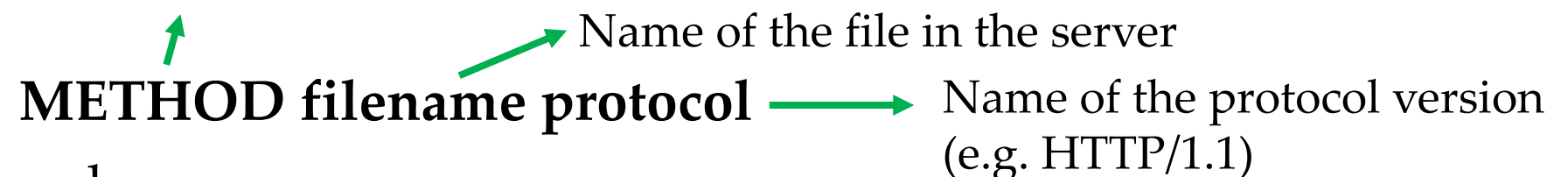



# Request structure (HTTP/1.1)

---

- Each **request** consists of one or more lines of ASCII text

Case-sensitive name of the method  
(type of the request)

- First line:  
**METHOD filename protocol**  

- Request headers:  
**Host: servername**   
**Other optional headers**
- Empty line
- Request body (payload):  
**Optional message body**

# HTTP methods

---

- The names of methods are case-sensitive.

Method	Description
GET	Read a Web page
HEAD	Read a Web page's header
POST	Append to a Web page
PUT	Store a Web page
DELETE	Remove a Web page
TRACE	Echo the incoming request
CONNECT	Connect through a proxy
OPTIONS	Query options for a web page
PATCH	Change the state of the resource

# Examples of request headers

---

- *Host*: the server's DNS name
- *User-Agent*: it allows the client to inform the server about the user agent making the request, e.g., the browser implementation
- *Accept*: it allows to tell the server what the client is willing to accept
- *If-Modified-Since*: time and date of last page modification, it allows the client to check the validity of cached pages (the client stores recently requested contents in a cache memory, similarly to what the server does).
- *Authorization*: needed for pages that are protected. The client might need to demonstrate it has the right to see the requested page.
- *Cache-Control*: it specifies directives that must be obeyed by all caching mechanisms along the request/response chain (e.g., *no-cache*).
- *Content-Type*: the type of the content in the request body.
- ...

# Response structure (HTTP/1.1)

- A response message is sent by a server to a client as a reply to its former request message.

Three-digit integer code representing the result of the server's attempt to understand and satisfy the request.

- Status line:

**Protocol status\_code reason\_phrase** →

Optional. If the status code indicated a problem, the *reason phrase* might displayed the further information about the nature of the problem.

- Response headers:

**Optional headers**

- Empty line

- Request body (payload):

**Optional message body**

Status code	Description
1xx	Informational: the request was received and is being processed,
2xx	Successful: the request was successfully received, understood, and accepted.
3xx	Redirecting: further action needs to be taken in order to complete the request.
4xx	Client error: the request contains bad syntax or cannot be fulfilled.
5xx	Server error: The server failed to fulfill an apparently valid request.

# Examples of response headers

---

- *Date*: date and time the message was sent
- *Content-Language*: the natural language used in the page
- *Content-Type*: the type of the returned content (e.g., HTML, image, plain text, ...)
- *Content-Length*: the content length in bytes
- *Last-Modified*: time and date the page was last changed
- *Expires*: Time and date when the page stops being valid
- *ETag*: tag for the content of the page (used for caching)
- *Location*: redirect the recipient to a location other than the requested URL for completion of the request or identification of a new resource
- ...



# GET

---

- **GET:** It requests the server to send a page or an object. It only retrieves data.

GET /wiki/Hypertext\_Transfer\_Protocol HTTP/1.1

Host: en.wikipedia.org

Accept-Language: en

## Request example:

```
GET / HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-GB,en;q=0.5
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
```

## Response example:

```
HTTP/1.1 200 OK
Date: Mon, 23 May 2005 22:38:34 GMT
Content-Type: text/html; charset=UTF-8
Content-Length: 155
Last-Modified: Wed, 08 Jan 2003 23:11:55 GMT
Server: Apache/1.3.3.7 (Unix) (Red-Hat/Linux)
ETag: "3f80f-1b6-3e1cb03b"
Accept-Ranges: bytes
Connection: close

<html>
  <head>
    <title>An Example Page</title>
  </head>
  <body>
    <p>Hello World, this is a very simple HTML document.</p>
  </body>
</html>
```

# Conditional GET and Partial GET

---

- The GET can become **conditional** if the request message includes an *If-Modified-Since*, *If-Unmodified-Since*, *If-Match*, *If-None-Match*, or *If-Range* header field.
  - The content is transferred only under the circumstances described by the conditional header field(s).
  - The conditional GET reduces unnecessary network usage by allowing cached entities to be refreshed without requiring the transfer of data already held by the client.
- The GET can become **partial** if the request message includes a *Range* header field.
  - Only part of the entity is transferred.
  - The partial GET method reduces unnecessary network usage by allowing partially-retrieved entities to be completed without transferring data already held by the client.

# POST

---

- **POST:** It asks to upload data to a server.
- If the request is accepted, then the server does something with the data that depends on the URL, and send a response page indicating the result.
  - POST is commonly used when forms are submitted, or when uploading files to servers, e.g., to extend a database appending some new data.

## POST HTML Form Example

```
POST /echo/post/form HTTP/1.1
Host: reqbin.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 23

key1=value1&key2=value2
```

## POST JSON Data Example

```
POST /echo/post/json HTTP/1.1
Host: reqbin.com
Accept: application/json
Content-Type: application/json
Content-Length: 81

{
  "Id": 78912,
  "Customer": "Jason Sweet",
  "Quantity": 1,
  "Price": 18.00
}
```

# Other important methods

---

- **PUT**: it asks to write a content in the server to the specified URL.
- **DELETE**: it asks to delete the content in the server at the specified URL.
- **PATCH**: it requests that the target resource modifies its state according to the instructions defined in the request body.
- **HEAD**: it just asks for the message header, without the actual page. It can be used to collect information for indexing purposes, or to test a URL for validity.

# POST vs PUT vs PATCH

---

- Difference between POST and PUT?
  - **POST**: the client asks to the server to accept the entity enclosed in the request as a new subordinate of the resource identified by the request URL.
  - **PUT**: the client asks to the server to store the enclosed entity under the request URL. → If the request URL refers to an already existing resource, the server replaces the existing entity with the new enclosed entity. If the URL does not point to an existing resource, the server can create the resource with that URL.
- Difference between PUT and PATCH?
  - **PUT** supplies a modified version of the resource to be allocated at the request URL.
  - **PATCH** supplies a set of instructions to modify the resource at the requested URL.

# Other (less) important methods

---

BONUS

- **TRACE:** it asks the server to send back the request. Useful when requests are not processed correctly to check what the server received.
- **CONNECT:** it lets a user make a connection to a Web server through an intermediate device, such as a Web cache.
- **OPTIONS:** it queries the server for a page and obtain the methods and headers that can be used with that page.

# Properties of the HTTP methods

---

- **Safe methods:** A request method is safe if a request with that method has no intended effect on the server.
  - GET, HEAD, OPTIONS, and TRACE are safe, since they only read data from the server.
  - POST, PUT, DELETE, PATCH, CONNECT are not safe, since they may modify the server state.
- **Idempotent methods:** A request method is idempotent if multiple identical requests with that method have the same intended effect as a single such request.
  - PUT and DELETE, and safe methods are idempotent.
  - POST, PATCH and CONNECT are not necessarily idempotent. Sending an identical POST request multiple times may further modify the state of the server.
- **Cacheable methods:** A request method is cacheable if responses to requests with that method may be stored by the client in the cache for future reuse.
  - Only GET, HEAD, and POST are cacheable.

# Summary of properties of the HTTP methods

---

METHOD	REQUEST HAS BODY	RESPONSE HAS BODY	IS SAFE	IS IDEMPOTENT	IS CACHEABLE
GET	Optional	Yes	Yes	Yes	Yes
HEAD	Optional	No	Yes	Yes	Yes
POST	Yes	Yes	No	No	Yes
PUT	Yes	Yes	No	Yes	No
DELETE	Optional	Yes	No	Yes	No
TRACE	No	Yes	Yes	Yes	No
CONNECT	Optional	Yes	No	No	No
OPTIONS	Optional	Yes	Yes	Yes	No
PATCH	Yes	Yes	No	No	No



# Example: inspecting the HTTP requests of Chrome

The screenshot shows a web browser with the address bar displaying `gcappon.github.io/teaching/`. The page content includes a profile for Giacomo Cappon, an Assistant Professor at DEI, University of Padova, and details about his teaching courses. The Chrome DevTools Network tab is open, showing a list of requests. The 'teaching/' request is selected, and its details are visible on the right. Annotations with green boxes and arrows point to specific parts of the interface:

- Request headers**: Points to the 'Headers' tab in the Network panel.
- Requested page and contents**: Points to the 'teaching/' request in the list.
- Response headers**: Points to the 'Response Headers' section in the details panel.

**Page Content:**

**Giacomo Cappon**  
Assistant Professor @ DEI, University of Padova, YOLO committer  
Padova, Italy  
Email  
ResearchGate  
Github  
Google Scholar  
PubMed  
ORCID

**Teaching**

**Biomedical Wearable Technologies for Healthcare and Wellbeing, 2023/24**  
Masters course, *University of Padova, Department of Information Engineering, 2024*  
Professor for the course unit Biomedical Wearable Technologies for Healthcare and Wellbeing of the Bioengineering course (Master's degree), School of Engineering, University of Padova (6 credits/CFU, SSD ING/INF-06) (~120 students).

**Biomedical Wearable Technologies for Healthcare and Wellbeing, 2022/23**  
Masters course, *University of Padova, Department of Information Engineering, 2023*  
Contract professor for the course unit Biomedical Wearable Technologies for Healthcare and Wellbeing of the Bioengineering course (Master's degree), School of Engineering, University of Padova (3 credits/CFU, SSD ING/INF-06).

**Network Tab Details:**

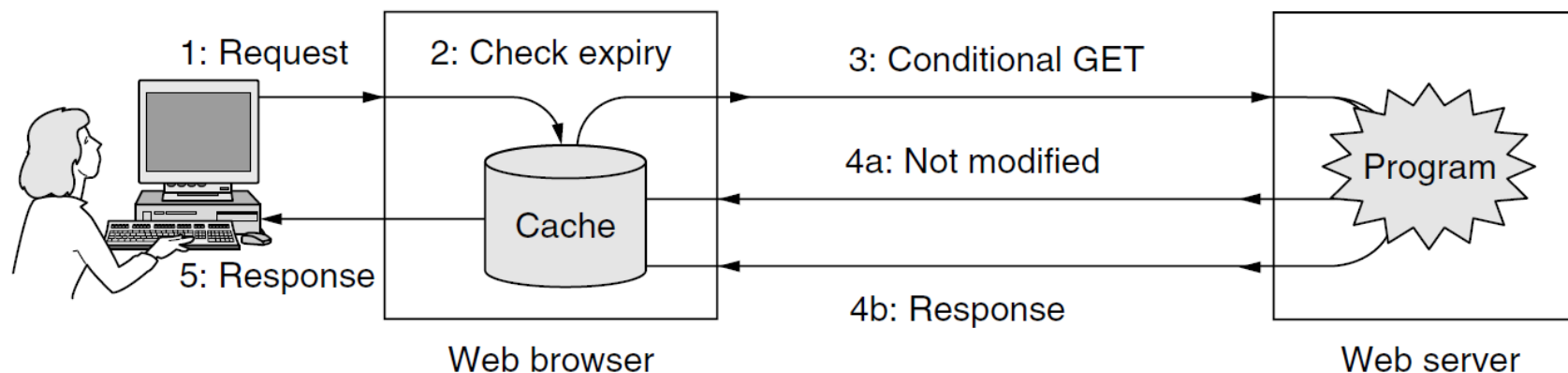
Name	Headers	Preview	Response	Initiator
teaching/	Request URL: <code>https://gcappon.github.io/teaching/</code> Request Method: <code>GET</code> Status Code: <code>200 OK</code> Remote Address: <code>185.199.110.153:443</code> Referrer Policy: <code>strict-origin-when-cross-origin</code>			

**Response Headers:**

Header	Value
Accept-Ranges	bytes
Access-Control-Allow-Origin	*
Age	0
Cache-Control	max-age=600
Content-Encoding	gzip
Content-Length	2891
Content-Type	text/html; charset=utf-8
Date	Thu, 01 Feb 2024 12:49:10 GMT
Etag	W/"64d79e9b-293c"
Expires	Thu, 01 Feb 2024 12:59:10 GMT

# Caching

- People often return to Web pages that they have viewed before, and related Web pages often have the same embedded resources.
- **Caching:** save on the client pages that are fetched for possible subsequent use.
- HTTP headers can be used to let the client know when a page can be safely re-used. → reduced network traffic
  - **Page validation:** The cache is consulted, and if it has a copy of a page for the requested URL that is known to be fresh (i.e., still valid), there is no need to fetch it anew from the server. The validity time is obtained by the *Expires* header of the original GET request.
  - **Conditional GET:** The client checks the time a cached page was last updated from the *Last-Modified* header. It can send it to the server using the *If-Modified-Since* header to ask for the page only if it has been changed in the meantime.



# Web communication protocols

---

- HTTP is not the only Web communication protocol.
- Different protocols have been defined for different purposes.

Name	Used for	Example
http	Hypertext (HTML)	<a href="http://www.ee.uwa.edu/~rob/">http://www.ee.uwa.edu/~rob/</a>
https	Hypertext with security	<a href="https://www.bank.com/accounts/">https://www.bank.com/accounts/</a>
ftp	FTP	<a href="ftp://ftp.cs.vu.nl/pub/minix/README">ftp://ftp.cs.vu.nl/pub/minix/README</a>
file	Local file	<a href="file:///usr/suzanne/prog.c">file:///usr/suzanne/prog.c</a>
mailto	Sending email	<a href="mailto:JohnUser@acm.org">mailto:JohnUser@acm.org</a>
rtsp	Streaming media	<a href="rtsp://youtube.com/montypython.mpg">rtsp://youtube.com/montypython.mpg</a>
sip	Multimedia calls	<a href="sip:eve@adversary.com">sip:eve@adversary.com</a>
about	Browser information	<a href="about:plugins">about:plugins</a>

# Outline

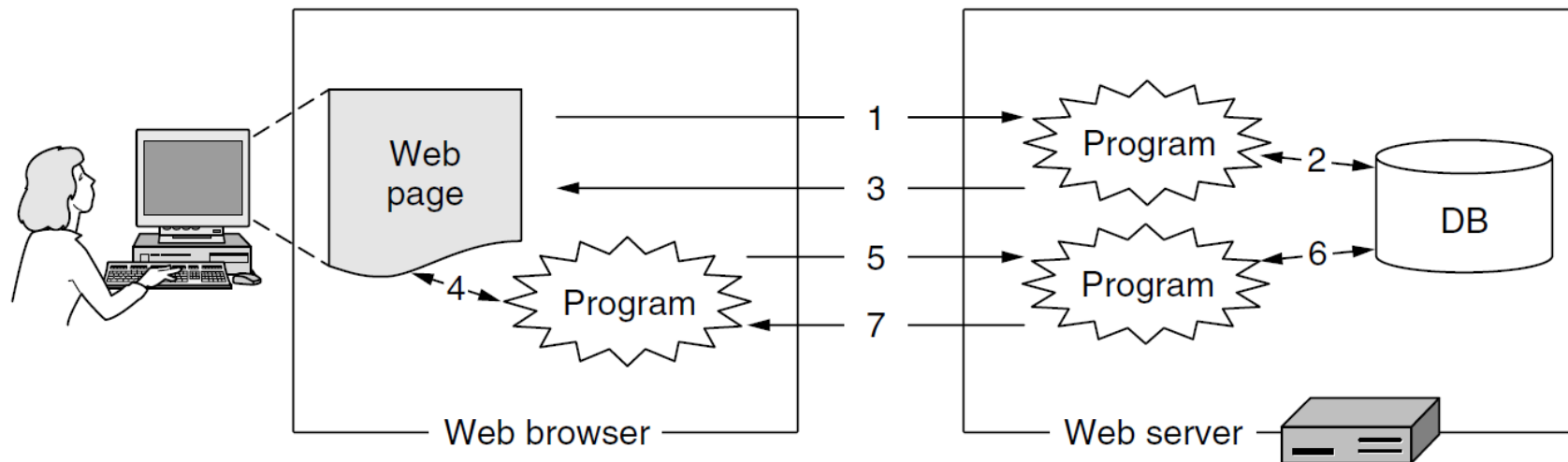
---

➤ HyperText Transfer Protocol (HTTP)

➤ Web applications

# Web application

- **Web application:** a Web page or a collection of Web pages, delivered over HTTP, which use server-side or client-side processing, i.e., programs, to provide an "application-like" experience within a Web browser.
- Web applications access data that are stored in databases in the server and use Web protocols (e.g., HTTP) to exchange data with the client. They use the Web browser to display a user interface by which the user can use the app (making queries, inserting data, view retrieved data, etc.).
- Advantage of Web applications: The users do not need to install separate application programs, and data can be accessed from different computers and backed up by the service operator.
- Examples of Web apps: Gmail, Google Docs, Trello, online banking apps, online retail sales apps, ...



To act as applications, Web pages can no longer be static, but they need to generate **dynamic content**.

# Server-side dynamic Web page generation

- A simple situation in which server-side processing is needed is a **Web form**.
- When the user compiles the form and click 'Submit order', the client sends a **POST request** to a specific URL pointing to a **program** in the Web server.
- The POST request will include the input data of the form, properly formatted, in the request body.
- The server receives the data and passes them to the program.
- The program process the data and generates an HTML page, whose content depends on the inserted data.
- The server returns the HTML page to the client's browser, in the body of the POST response.

### Widget Order Form

Name

Street address

City  State  Country

Credit card #  Expires  M/C ☐ Visa ☐

Widget size Big ☐ Little ☐ Ship by express courier ☐

Thank you for ordering an AWI widget, the best widget money can buy!

How can the server invoke the program?

Using specific APIs (Application Programming Interfaces)



# Common Gateway Interface

---

- **Common Gateway Interface (CGI):** an interface specification that allows Web servers to talk to back-end programs and scripts that can accept input (e.g., from forms) and generate HTML pages in response.
- Programs may be written in whatever language is convenient (e.g., Python).
- By convention, programs invoked via CGI live in a directory called **cgi-bin**, which is visible in the URL.
- The server maps a request to this directory to a program name and executes that program as a separate process. It provides any data sent with the request as input to the program.
- The output of the program gives a Web page that is returned to the browser.

```
<html>
<head> <title> AWI CUSTOMER ORDERING FORM </title> </head>
<body>
<h1> Widget Order Form </h1>
<form ACTION="http://widget.com/cgi-bin/order.cgi" method=POST>
<p> Name <input name="customer" size=46> </p>
<p> Street address <input name="address" size=40> </p>
<p> City <input name="city" size=20> State <input name="state" size =4>
Country <input name="country" size=10> </p>
<p> Credit card # <input name="cardno" size=10>
Expires <input name="expires" size=4>
M/C <input name="cc" type=radio value="mastercard">
VISA <input name="cc" type=radio value="visacard"> </p>
<p> Widget size Big <input name="product" type=radio value="expensive">
Little <input name="product" type=radio value="cheap">
Ship by express courier <input name="express" type=checkbox> </p>
<p><input type=submit value="Submit order"> </p>
Thank you for ordering an AWI widget, the best widget money can buy!
</form>
</body>
</html>
```

# PHP: Hypertext Preprocessor

- An alternative approach to CGI is to embed little scripts inside HTML pages and have them be executed by the server itself to generate the page.
- A popular language for writing these scripts is **PHP (PHP: Hypertext Preprocessor)**.
- Web pages embedding PHP scripts have a .php extension.

1. The Web form HTML file **contains a POST** to action.php. The POST sends formatted input data to the server (e.g. 'name=Amelia&age=32').
2. The server process **action.php** with the data provided by the POST. The **php script** is run.
3. action.php generates a **HTML page**, which is provided in response to the client's browser.

## form.html

```
<html>
<body>
<form action="action.php" method="post">
<p> Enter your name: <input type="text" name="name"> </p>
<p> Enter your age: <input type="text" name="age"> </p>
<input type="submit">
</form>
</body>
</html>
```

## response.html

```
<html>
<body>
<h1> Reply: </h1>
Hello Amelia.
Prediction: next year you will be 33
</body>
</html>
```

## action.php

```
<html>
<body>
<h1> Reply: </h1>
Hello <?php echo $name; ?>.
Prediction: next year you will be <?php echo $age + 1; ?>
</body>
</html>
```



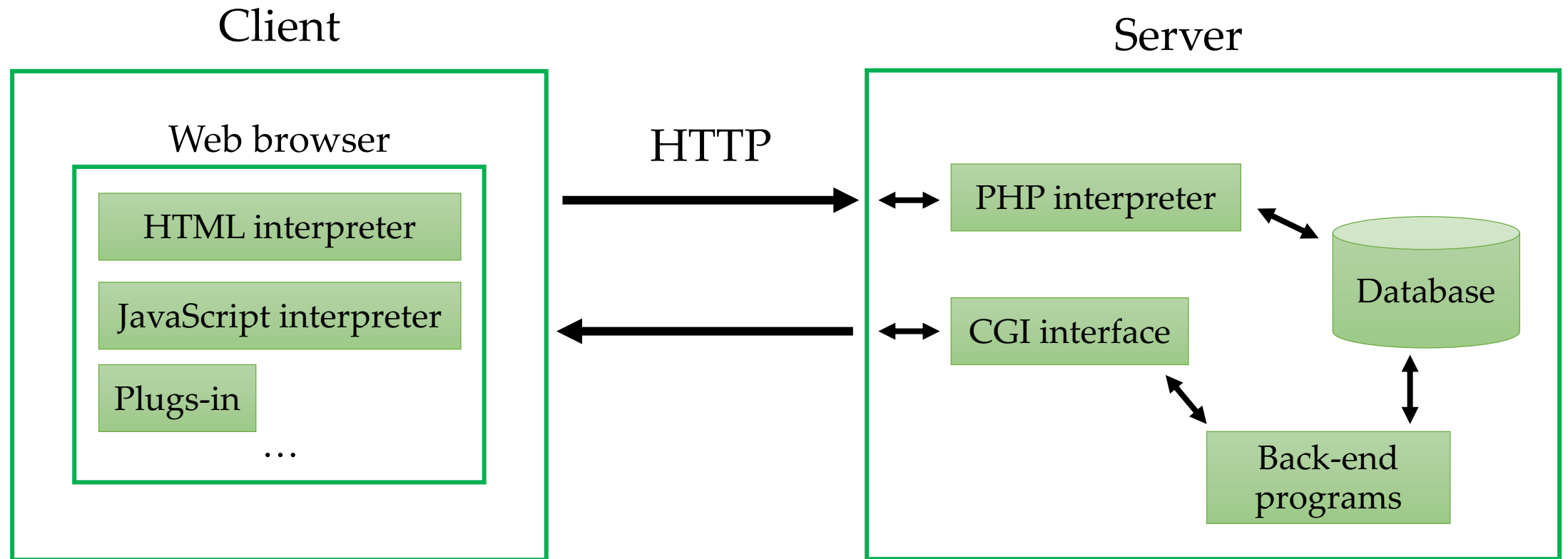
# Client-side dynamic Web page generation

---

- Client-side processing is necessary to have applications with a more responsive interface that quickly responds to user commands and interact with the user.
- It is necessary to have scripts embedded in HTML pages that are executed on the client machine.
- The most popular language for client-side scripting is **JavaScript**.
- When the browser interprets the HTML page, it also interprets the JavaScript code contained in the page.
- All the work is done locally, inside the browser. There is no contact with the server.

```
<html>
<head>
<script language="javascript" type="text/javascript">
function response(test form) {
var person = test form.name.value;
var years = eval(test form.age.value) + 1;
document.open();
document.writeln("<html> <body>");
document.writeln("Hello " + person + ".<br>");
document.writeln("Prediction: next year you will be " + years +
".");
document.writeln("</body> </html>");
document.close(); }
</script>
</head>
<body>
<form>
Please enter your name: <input type="text" name="name">
<p>Please enter your age: <input type="text" name="age"><p>
<input type="button" value="submit"
onclick="response(this.form)">
</form>
</body>
</html>
```

# Web application architecture



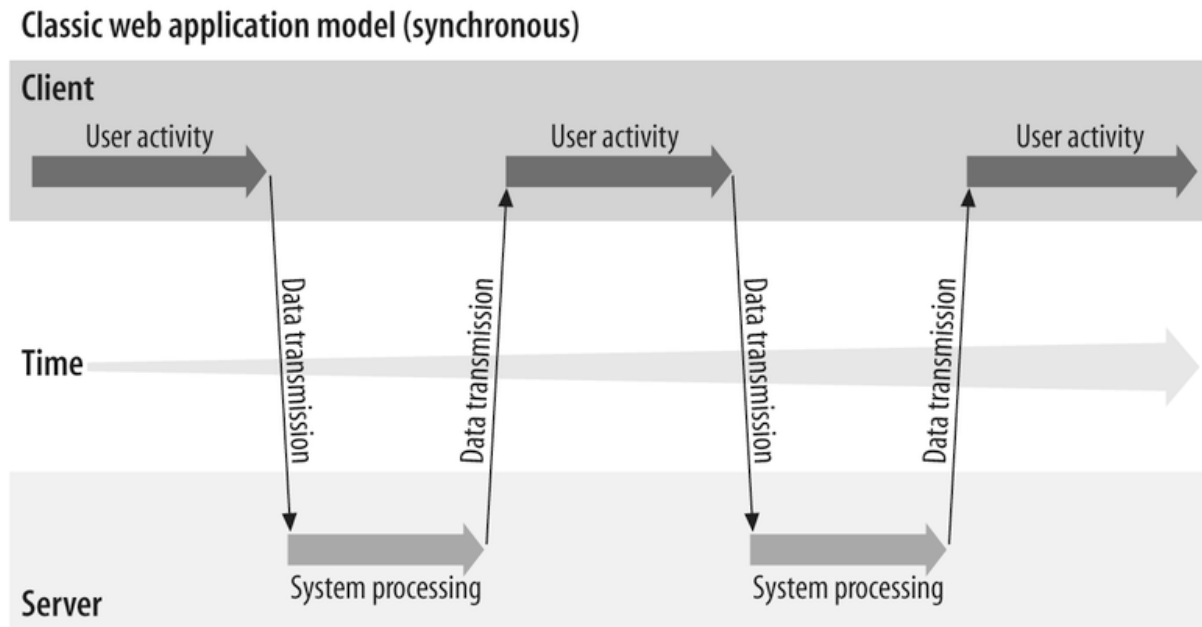
➤ Alternatives to JavaScript: VBScript, applets Java

➤ Alternatives to PHP: JavaServer Pages, Active Server Pages .NET

# Limitation of classic Web application model (synchronous)

---

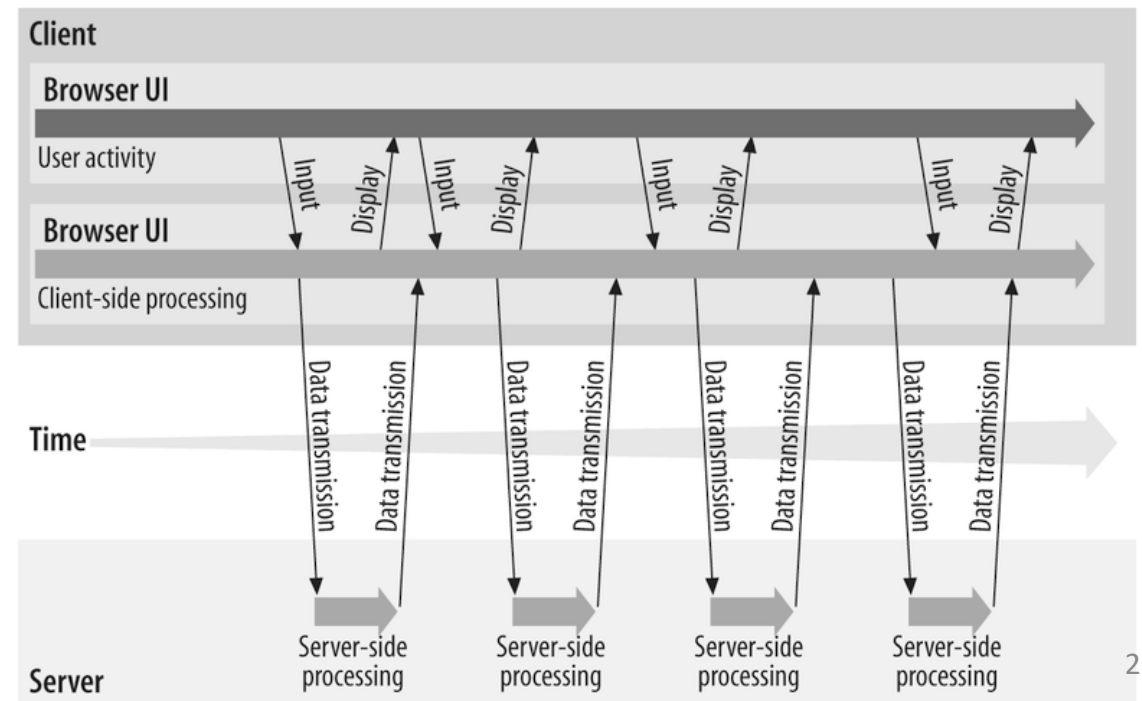
- Every update of the Web page requiring server-side processing, requires the re-load of the entire Web page (by HTTP request/response), even if only a small part of the page has to be updated.



# Asynchronous JavaScript and XML (AJAX)

- This limitation is overcome by **Asynchronous JavaScript and XML (AJAX)**, a set of techniques for Web application development that decouples the server-client interchange of data from the interchange of Web pages.
- Asynchronous applications: applications that can send and retrieve data from a server asynchronously (i.e., in the background) without interfering with the display of the existing page.
- AJAX allows Web applications to change content dynamically without the need to reload the entire page.
- The data are exchanged with the server in a structured data format such as XML or JSON.
- Advantages of AJAX:
  - It increases the responsiveness of Web applications
  - It decreases network traffic

Ajax web application model (asynchronous)



# Structured data formats

---

- Asynchronous Web applications need to exchange data with the server.
- HTML is not suitable for representing data because it mixes content with formatting. It is indeed concerned about the presentation of the content.
  - It is difficult to automatically retrieve data within an HTML file
- Two popular structured data formats that facilitate the automatic structured data processing:
  - Extensible Markup Language (XML)
  - JavaScript Object Notation (JSON)

# Extensible Markup Language (XML)

---

- A language for specifying structured content.
- Introduced in 1998.
- It defines a set of rules for encoding documents in a format that is both human-readable and machine-readable.
- Unlike HTML, there are no defined tags for XML. Each user can define their own tags.
- Extension .xml

Example of XML code that define a book list

```
<?xml version="1.0" ?>
<book list>
<book>
<title> Human Behavior and the Principle of Least Effort
</title>
<author> George Zipf </author>
<year> 1949 </year>
</book>
<book>
<title> The Mathematical Theory of Communication </title>
<author> Claude E. Shannon </author>
<author> Warren Weaver </author>
<year> 1949 </year>
</book>
<book>
<title> Nineteen Eighty-Four </title>
<author> George Orwell </author>
<year> 1949 </year>
</book>
</book list>
```

# JavaScript Object Notation (JSON)

---

- A format that uses human-readable text to store and transmit data objects consisting of attribute–value pairs and arrays.
- Introduced in 2001.
- JSON is language-independent. It was derived from JavaScript, but many modern programming languages include code to generate and parse JSON-format data.
- Extension .json
- JSON data types: number, string, Boolean, null (empty value)
- JSON data structures
  - Object
  - Array

# JSON data structures

---

- **JSON object:** a collection of name–value pairs where the names (also called keys) are strings. The order of pairs is not relevant. The object is surrounded by curly braces.

```
{
  "employee": {
    "id": 1,
    "name": "Admin",
    "location": "USA"
  }
}
```

Objects can contain array and arrays can contain objects.

- **JSON array:** an ordered list of zero or more elements, each of which may be of any type. Arrays use square bracket notation with comma-separated elements.

```
{
  "employees": [
    {
      "id": 1,
      "name": "Admin",
      "location": "India"
    },
    {
      "id": 2,
      "name": "Author",
      "location": "USA"
    },
    {
      "id": 3,
      "name": "Visitor",
      "location": "USA"
    }
  ]
}
```



# References

---

- Tanenbaum, Wetherall – Computer Networks – Fifth Edition
  - Chapter 7 – The application layer