

COM1025: Web and Database Systems

The Internet and JavaScript

(Slides credited to Dr. Manos Panaousis)

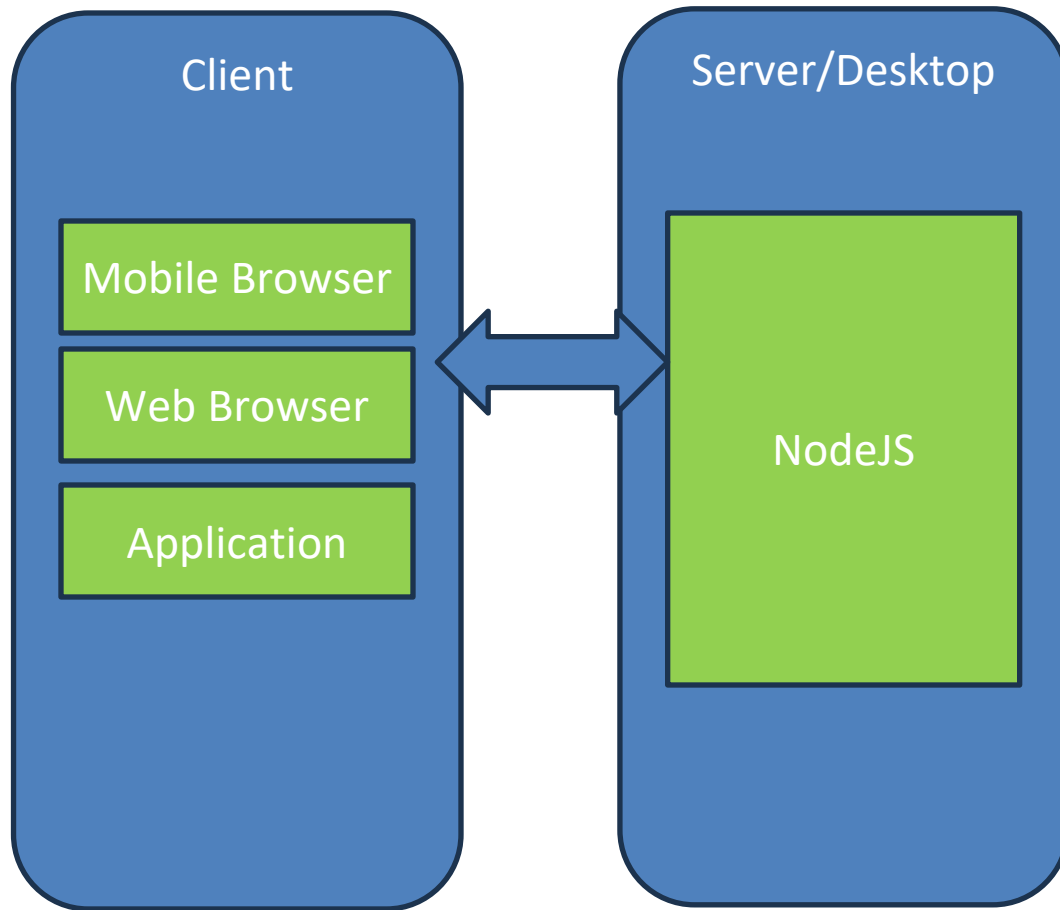
Week 8

JavaScript

Defining Some Terms

- **JavaScript:** JavaScript is a programming language primarily used for creating interactive and dynamic content within web browsers. **You won't be required to write JavaScript that runs in the web browser, for this module.**
- **Node.js:** : Node.js is not a programming language but a runtime environment that allows you to run JavaScript code on the server. **This is the environment we will be using for this module**

Where Does Node.JS Run

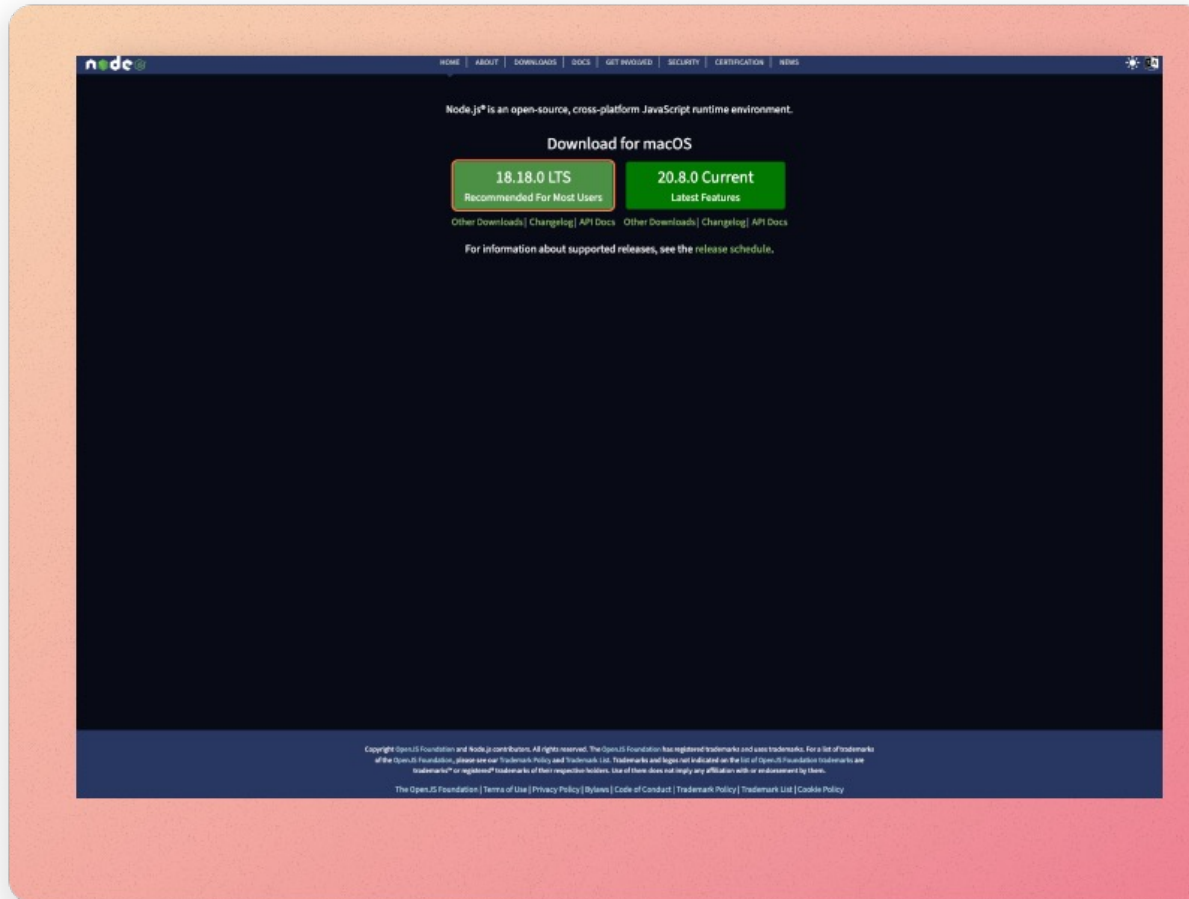


- runs on a server or desktop computer
- It can handle HTTP requests and responses
 - i. making it well-suited for building web applications

Who Uses Node.JS?



Installing Node.JS



To run JavaScript in the NodeJS environment, you should install NodeJS on your computer. Go to the official Node.js website at <https://nodejs.org> and download the installer

Creating Our First Node.js Program

```
1 console.log("hello world");  
2
```

index.js

```
→ node-app node index.js  
hello world  
→ node-app █
```

Our terminal

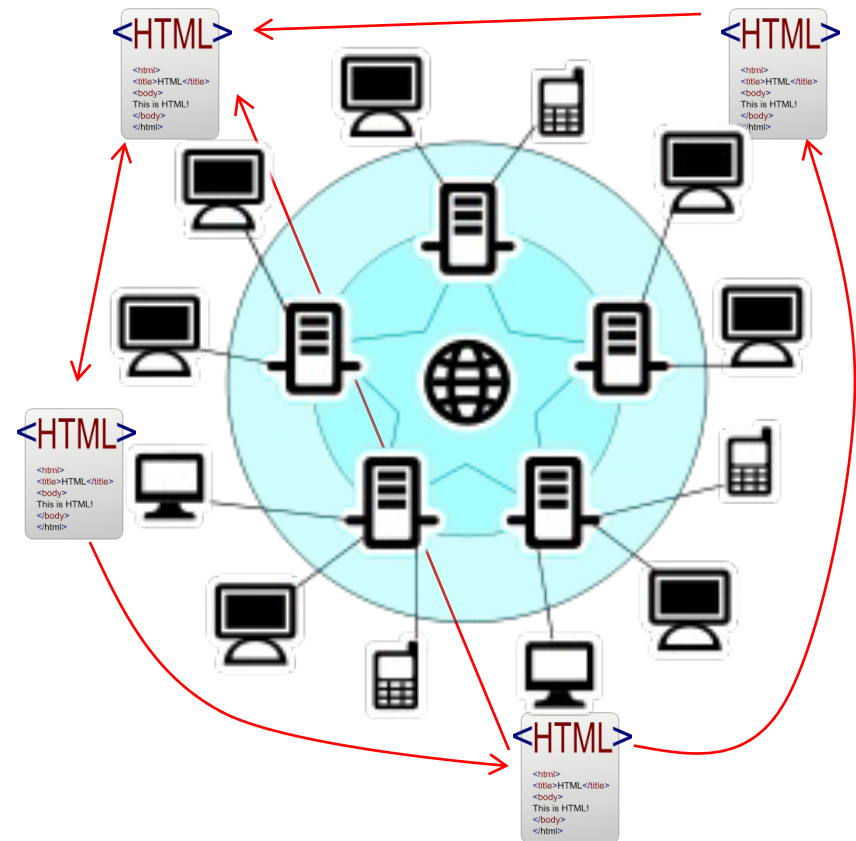
Basic JavaScript Concepts

[Check the lab notes for an introduction to JavaScript concepts](#)

Computer Networking

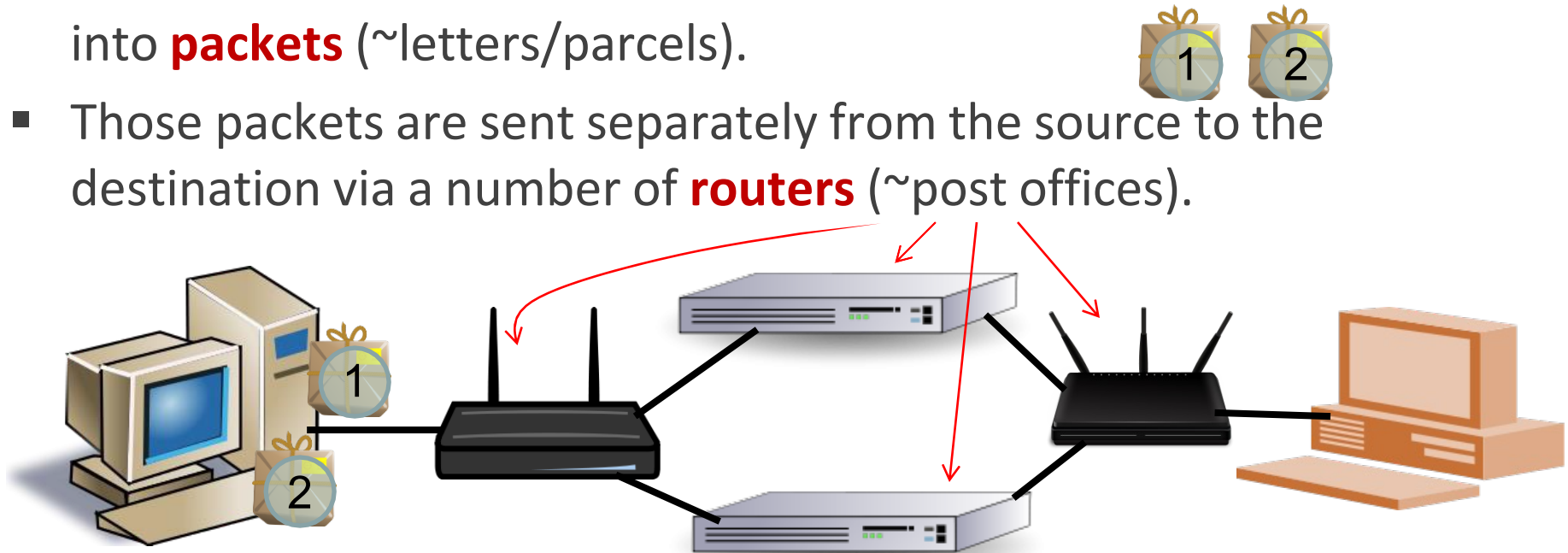
From the Internet to the Web

- Computers (PCs, servers, laptops, smart phones, tablets, ...)
- Computer networks
 - ✓ A **particular** interconnected network of computer networks = Internet
- **“particular”** =
A special set of networking protocols (TCP/IP)



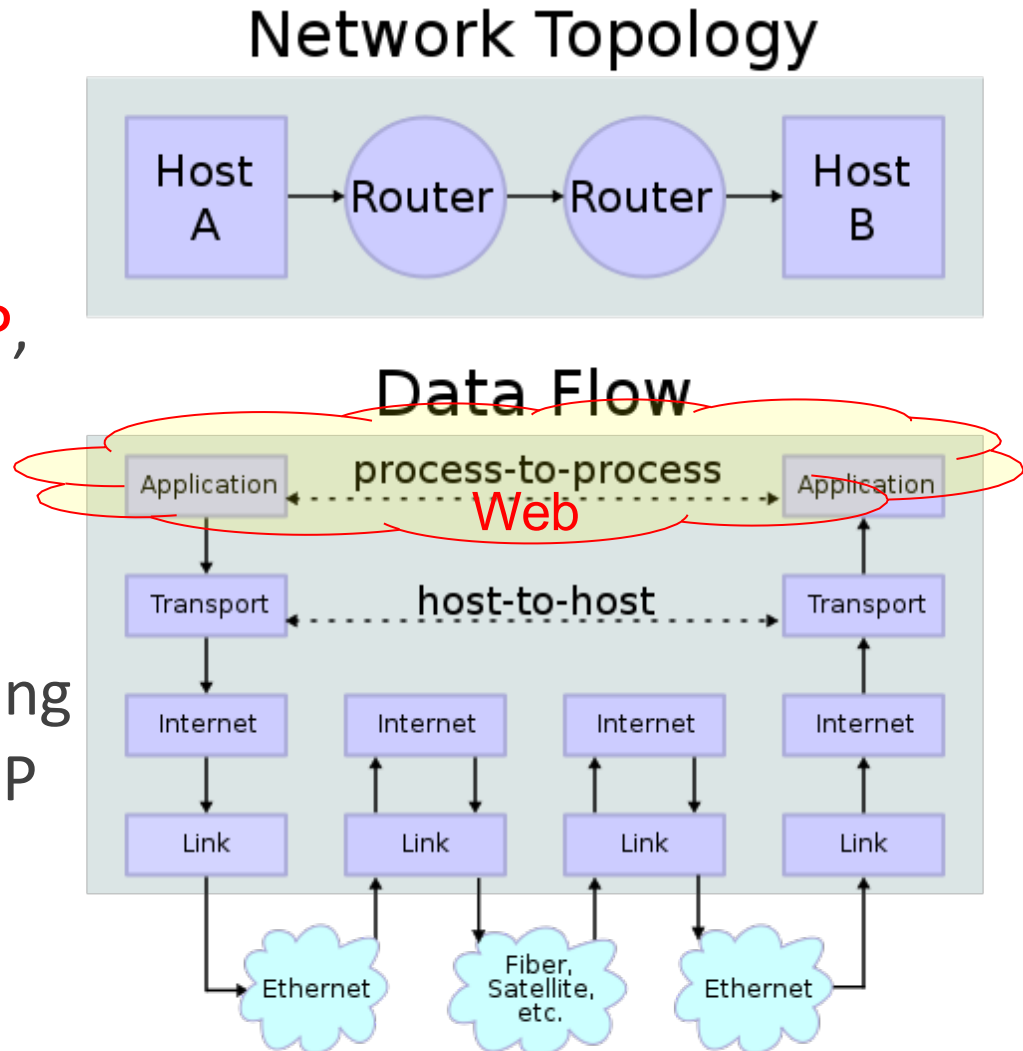
How does the Internet work?

- Each network interface of a networked computer on the Internet (host) has a unique address (~street number).
 - ✓ One computer may have $n > 1$ network interfaces so $n > 1$ addresses (e.g. a PC has both LAN and WiFi connections).
- Data from one host (source) to another (destination) is divided into **packets** (~letters/parcels).
- Those packets are sent separately from the source to the destination via a number of **routers** (~post offices).



Four abstract layers of the Internet

1. **Application layer:**
HTTP(S), FTP, SMTP,
DNS, ...
2. **Transport layer:** **TCP**,
UDP
3. **Internet layer:** **IP**
4. **Link layer:** the
underlying networking
architecture below IP
(Ethernet, 802.11)



■ UDP:

- ✓ **Anything where you don't care too much if you get all data always**
- ✓ Tunnelling/VPN (lost packets are ok - the tunnelled protocol takes care of it)
- ✓ Media streaming (lost frames are ok)
- ✓ Online games that don't care if you get *every* update
- ✓ Local broadcast mechanisms (same application running on different machines "discovering" each other)
- ✓ **(Speed)**

■ TCP:

- ✓ **Almost anything where you have to get all transmitted data**
- ✓ Web
- ✓ SSH, FTP, telnet
- ✓ SMTP, sending mail
- ✓ IMAP/POP, receiving mail
- ✓ **(Reliability)**

- The communications protocol sending data packets from one host to the other via routing is called the **Internet Protocol (IP)**.
 - ✓ Two editions: IPv4 and IPv6.
- The unique address of each host is called its **IP address**.
 - ✓ IPv4: 32-bit, normally written as 4-segment dot-decimal format a.b.c.d, where each segment is an 8-bit integer between 0 and 255: e.g. 172.16.254.1.
 - ✓ IPv6: 128-bit, new-generation IP address, but wide development is still to come,
 - ❖ <https://www.google.com/intl/en/ipv6/statistics.html#tab=per-country-ipv6-adoption>

How do I get an IP

- **DHCP (Dynamic Host Configuration Protocol)**
- **Static Configuration**

Check your own IP

```
C:\Users\yyang>ipconfig

Windows IP Configuration

Ethernet adapter Ethernet:

    Media State . . . . . : Media disconnected
    Connection-specific DNS Suffix  . :

Ethernet adapter Ethernet 2:

    Media State . . . . . : Media disconnected
    Connection-specific DNS Suffix  . :

Ethernet adapter Local Area Connection* 11:

    Media State . . . . . : Media disconnected
    Connection-specific DNS Suffix  . :

Wireless LAN adapter Local Area Connection* 2:

    Media State . . . . . : Media disconnected
    Connection-specific DNS Suffix  . :

Wireless LAN adapter Local Area Connection* 12:

    Media State . . . . . : Media disconnected
    Connection-specific DNS Suffix  . :

Wireless LAN adapter Wi-Fi:

    Connection-specific DNS Suffix  . : connect
    Link-local IPv6 Address . . . . . : fe80::c8f7:f788:4cd5:7285%5
    IPv4 Address. . . . . : 192.168.1.6
    Subnet Mask . . . . . : 255.255.255.0
    Default Gateway . . . . . : 192.168.1.1
```


- IP addresses are difficult to remember!
 - ✓ How can one remember meaningless IP addresses?
 - ✓ And many different IP addresses?
 - ✓ And much longer 128-bit IPv6 addresses?

- IP addresses are difficult to remember!
 - ✓ How can one remember meaningless IP addresses?
 - ✓ And many different IP addresses?
 - ✓ And much longer 128-bit IPv6 addresses?
- More user-friendly domain names are introduced as aliases of IP addresses.
 - ✓ DNS (domain name system) servers convert domain names to IP addresses.
 - ✓ Multiple domain names may point to the same IP address.
 - ✓ Syntax: `hostname.sub_domain.....top_level_domain`
 - ✓ Example 1: `surreylearn.surrey.ac.uk` \Rightarrow 131.227.132.44

- IP addresses are difficult to remember!
 - ✓ How can one remember meaningless IP addresses?
 - ✓ And many different IP addresses?
 - ✓ And much longer 128-bit IPv6 addresses?
- More user-friendly domain names are introduced as aliases of IP addresses.
 - ✓ DNS (domain name system) servers convert domain names to IP addresses.
 - ✓ Multiple domain names may point to the same IP address.
 - ✓ Syntax: `hostname.sub_domain.....top_level_domain`
 - ✓ Example 1: `surreylearn.surrey.ac.uk` \Rightarrow `131.227.132.44`
 - ✓ Example 2: `localhost` \Rightarrow `127.x.x.x` (on many systems just `127.0.0.1`) = the current machine itself (Local debug)

Traceroute

```
C:\WINDOWS\system32\cmd.exe

-d          Do not resolve addresses to hostnames.
-h maximum_hops  Maximum number of hops to search for target.
-j host-list  Loose source route along host-list (IPv4-only).
-w timeout    Wait timeout milliseconds for each reply.
-R          Trace round-trip path (IPv6-only).
-S srcaddr   Source address to use (IPv6-only).
-4          Force using IPv4.
-6          Force using IPv6.

C:\Users\yyang>tracert google.com

Tracing route to google.com [216.58.204.46]
over a maximum of 30 hops:

  1  11 ms    1 ms     <1 ms   vodafone.connect [192.168.1.1]
  2   8 ms    8 ms     9 ms   host-212-158-250-34.dslgb.com [212.158.250.34]
  3  10 ms    7 ms     7 ms   63.130.105.130
  4   8 ms    8 ms     8 ms   72.14.216.237
  5   *       *       *      Request timed out.
  6  26 ms   11 ms   10 ms   172.253.68.22
  7  19 ms    9 ms   11 ms   74.125.242.83
  8   9 ms    8 ms   10 ms   72.14.237.53
  9  10 ms    9 ms    9 ms   216.239.57.236
 10  10 ms   11 ms   14 ms   108.170.246.161
 11  10 ms    9 ms    9 ms   108.170.238.119
 12  11 ms   14 ms   18 ms   lhr25s12-in-f46.1e100.net [216.58.204.46]

Trace complete.

C:\Users\yyang>
```

HTTP

- The Hypertext Transfer Protocol (HTTP) is designed to enable communications between clients and servers.
- HTTP is a **two-party communication** protocol:
 - ✓ Party 1 – The **client** @ Computer 1: A **web browser** (also called a “user agent”) that requests web pages
 - ✓ Party 2 – The **server** @ Computer 2: A program that handles requests of web pages from multiple clients
- Steps:
 - ✓ A client submits an **HTTP request** to the server.
 - ✓ The server returns an **HTTP response** to the client.
 - ✓ The response contains **status information** about the request and may also contain the **requested content**.
- HTTP uses the Transmission Control Protocol (TCP)

Web browser vs. Web (HTTP) server

- The program running at the server side is normally called a **web server** or an **HTTP server**

- ✓ Two most widely used web (HTTP) servers

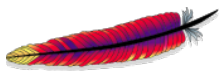
- ❖ **Apache HTTP Server**: **cross-platform**, used in our labs and for your coursework **Microsoft IIS** (Internet Information Services, formerly known as Internet Information Server): Windows platforms only

- ❖ **NodeJS** (**We will use this one**)

- **Security?**

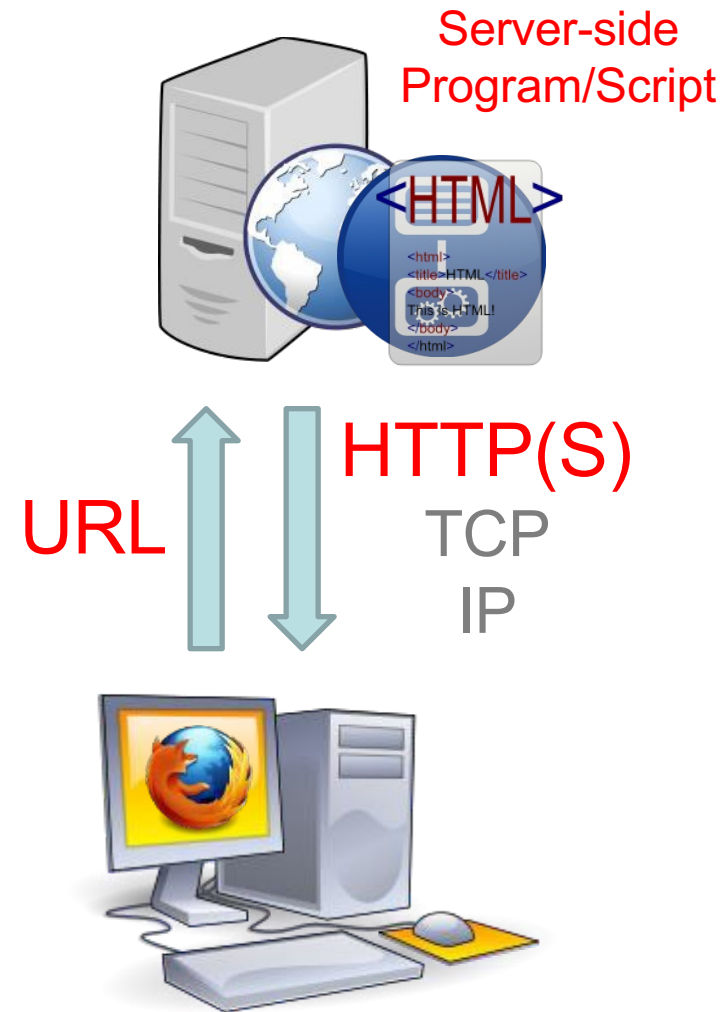
- It has a secure edition **HTTPS (HTTP Secure)**

- ✓ HTTPS allows both **end-to-end encryption** of all communications between the web server and the web browser, and also server/client authentication
- ✓ It uses a cryptographic protocol called TLS/SSL



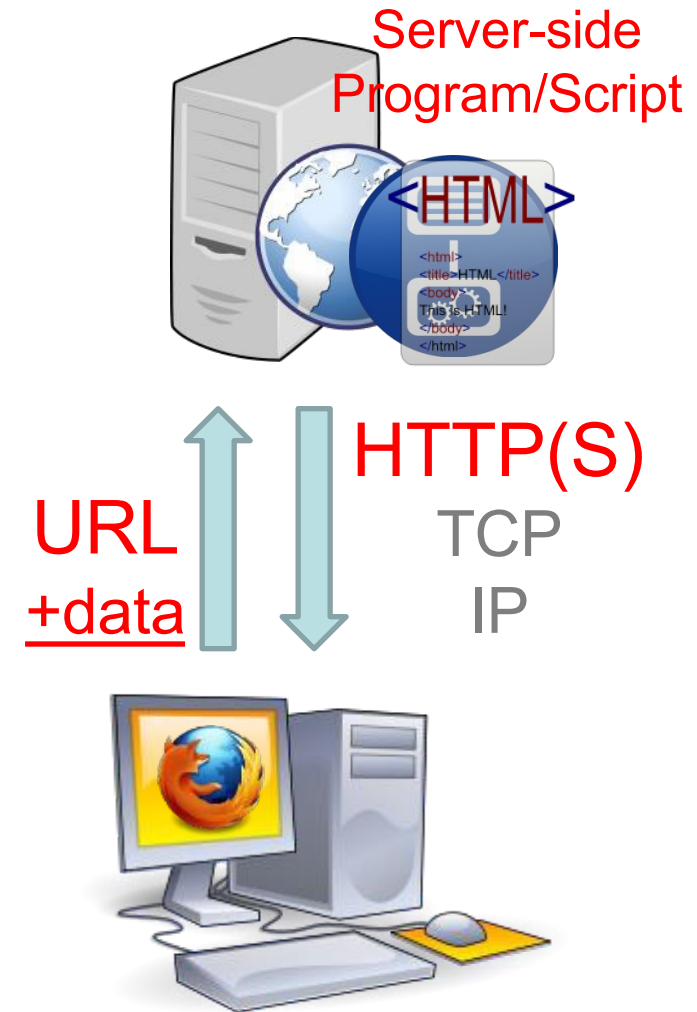
From static to dynamic web pages

- A requested web page is not necessarily a static one
- It can be a **server-side program/script** residing on the web server
- Why do we need dynamic programs/scripts?
 - ✓ Handle dynamic data.
 - ✓ Interact with users.
 - ✓ Do more with less code!
 - ✓ ...



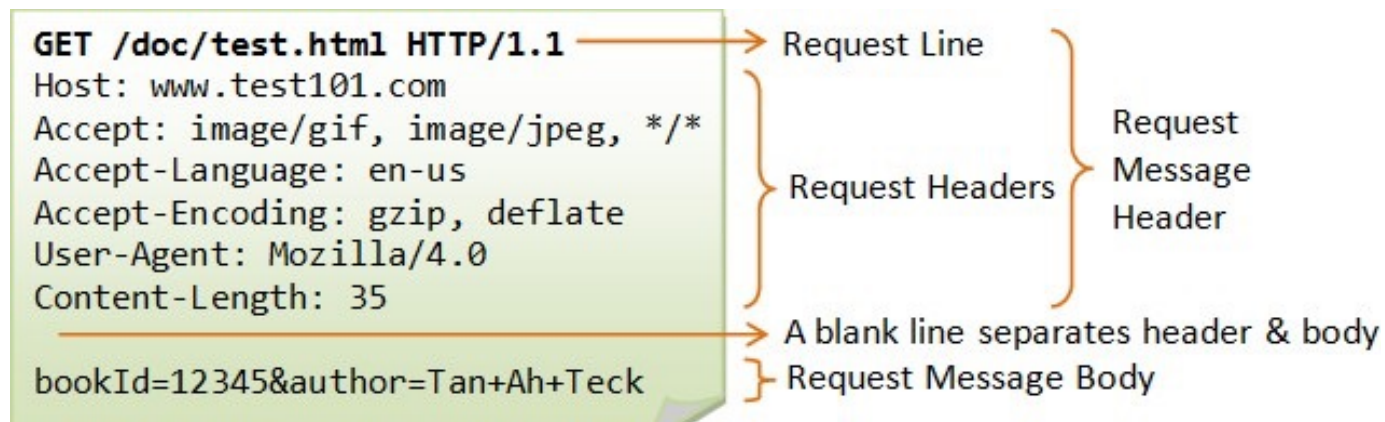
Request a dynamic web page

- User \Rightarrow Web browser
 - ✓ URL <http://...> (**optionally + data**)
- Web browser \Rightarrow Web server
 - ✓ URL <http://...> (**optionally + data**)
- Web server:
 - ✓ URL \Rightarrow Local file path
 - ✓ Execute the local file as a program and record the output (HTML code)
- Web server \Rightarrow Web browser
 - ✓ HTML code generated above
- Web browser \Rightarrow User
 - ✓ The received HTML code shown (in a web browser) as a web page



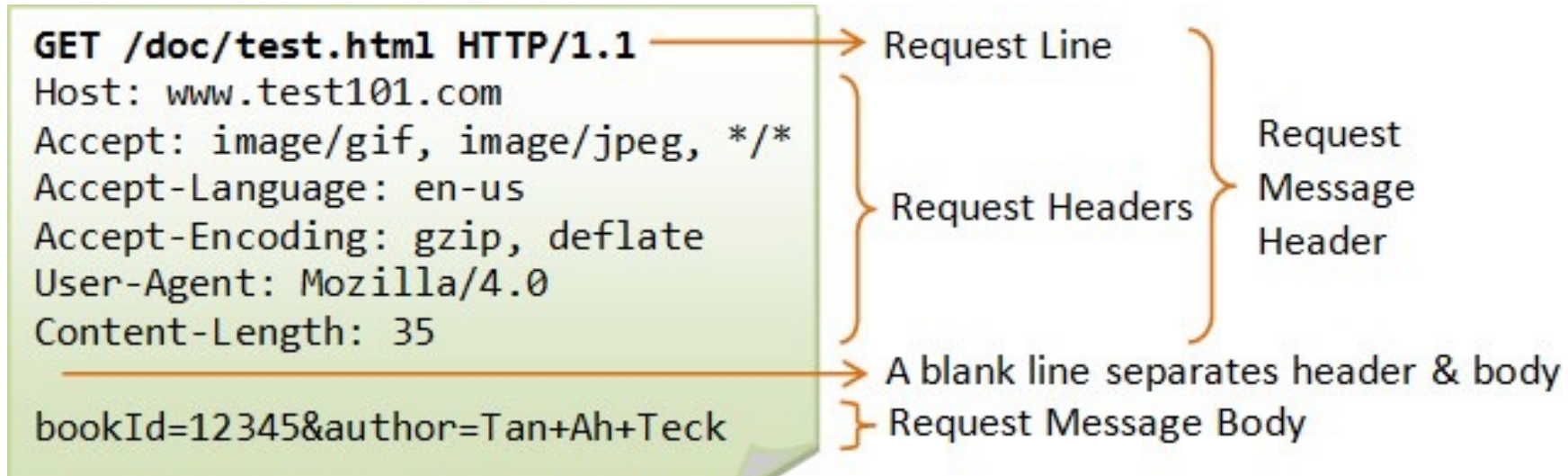
Sending a client request

- Once the connection is established, the **user-agent** (e.g. web browser, a crawler) can send the request
- A **client request** consists of **text directives** divided into:
 - ✓ The **first line** contains a **request method** (GET, POST) followed by its parameters:
 - ❖ **the path of the document**, i.e. an **absolute URL** without the protocol or domain name
 - ❖ **the HTTP protocol version**



Sending a client request

- Subsequent lines represent **an HTTP header**, giving the server information about **what type of data is appropriate** (e.g. what language), or other data altering its behavior (e.g. not sending an answer if it is already cached).
- Then, the **Request Message** itself is included



Example of a request

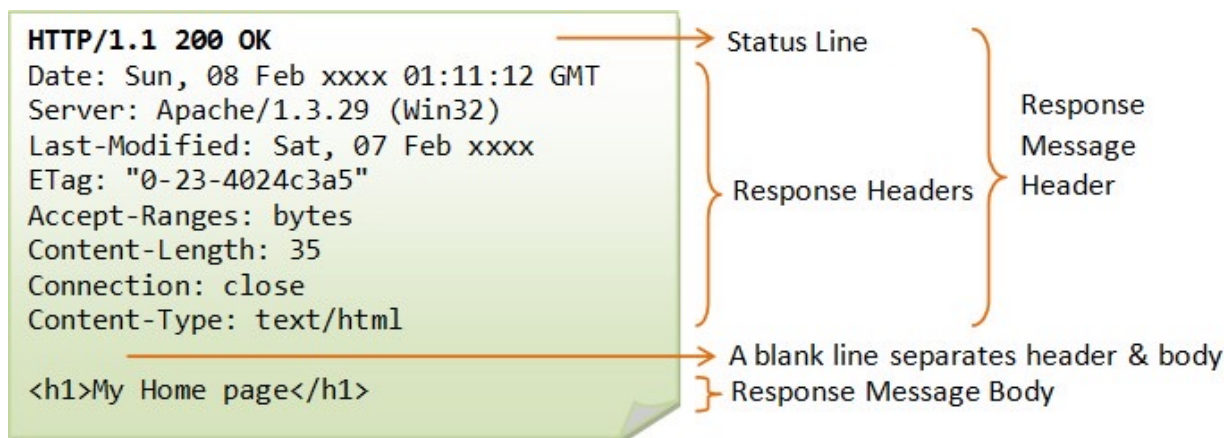
Fetching the root page of developer.mozilla.org, i.e. <http://developer.mozilla.org/>, and telling the server that the user-agent would prefer the page in French, if possible:

```
1 GET / HTTP/1.1
2 Host: developer.mozilla.org
3 Accept-Language: fr
4
```

Observe that final empty line, this separates the data block from the header block. As there is no `Content-Length` provided in an HTTP header, this data block is presented empty, marking the end of the headers, allowing the server to process the request the moment it receives this empty line.

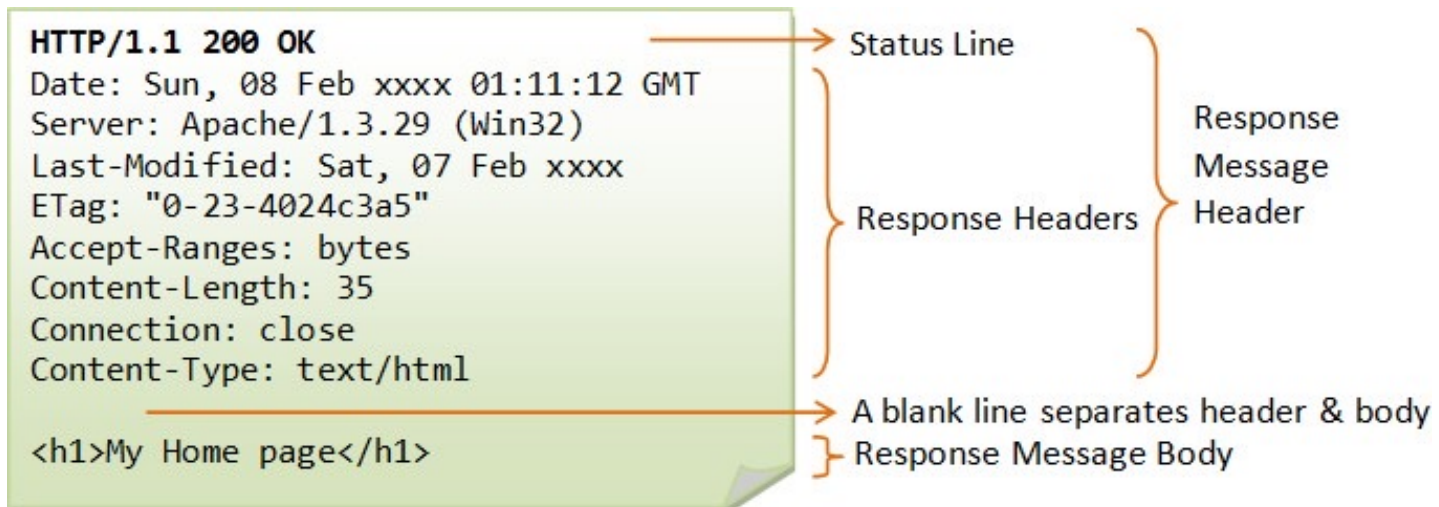
Structure of a server response

- The web server processes the request, and ultimately **returns a response**
- A server response is formed of text directives, divided into the following blocks:
 - ✓ The first line, the status line, consists of an **acknowledgment of the HTTP version used**, followed by a **status request** (and its brief meaning in human-readable text).
 - ✓ Subsequent lines represent specific **HTTP headers**, giving the client information about the data sent (e.g. type, data size, compression algorithm used, hints about caching).



Structure of a server response

- A server response is formed of text directives, divided into the following blocks:
 - ✓ Similarly to the block of HTTP headers for a client request, these HTTP headers form a **block ending with an empty line**.
 - ✓ The final block is a data block, which contains the **optional data**.



Beyond URL: data transmission

- Client (web browser) \Rightarrow Web server
 - ✓ An **HTTP request** contains: **URL** (what web page?), **data** (what data is sent to the requested web page?)
- Two main ways (called HTTP request methods or verbs in HTTP terms) for transmitting data from client to server:
 - ✓ GET
 - ❖ **Requests** data from a specified resource.
 - ❖ Data transmitted as part (query string) of the URL:
http://.../...?data1=value1&data2=value2...#fragment_ID
 - ✓ POST
 - ❖ **Submits** data to be processed to a specified resource.
 - ❖ Data transmitted as the message body of an HTTP request.

The GET method

- the **query string** (name/value pairs) is sent **in the URL** of a GET request:
 - ✓ e.g., `/test/demo_form.php?name1=value1&name2=value2`
 - ❖ GET method sends the **encoded user information** appended to the page request
 - ❖ The page and the encoded information are separated by the **?** Character
- The GET method produces a long string that **appears in your server logs**
- GET requests should be used only to retrieve data
 - ✓ GET requests should **never** be **used** when dealing with **sensitive data**
- Note: GET **can't** be used to send binary data, like images or word documents, to the server (string exclusively)
- To be covered: PHP provides **\$_GET associative array** to access all the sent information using GET method
- GET requests remain in the browser history and they **can be bookmarked**



The POST method

- the **query string** (name/value pairs) is sent **in the HTTP message body** of a POST request:

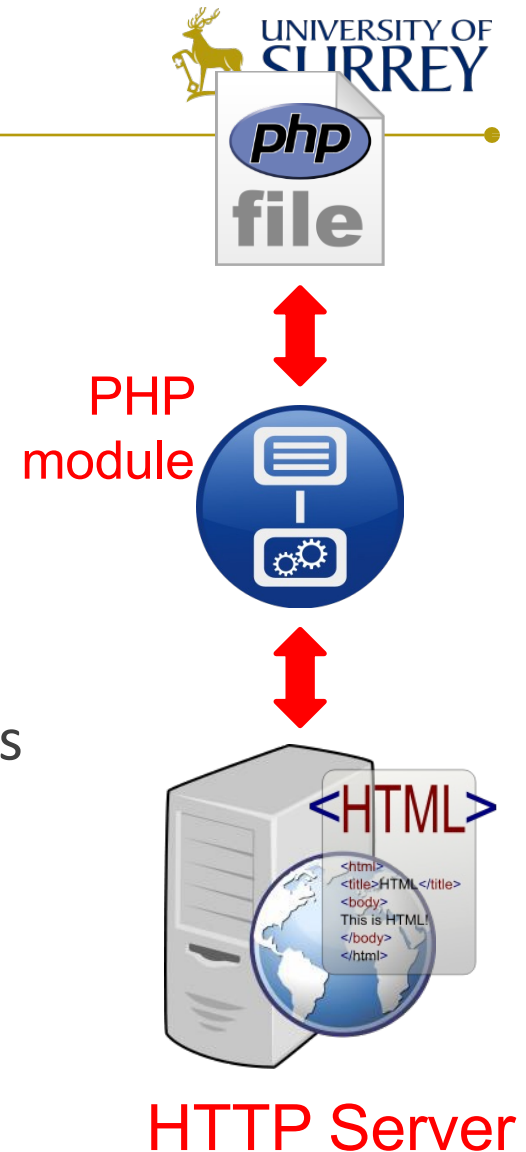
```
POST / HTTP/1.1
Host: foo.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 13

say=Hi&to=Mom
```

- It has no limit on data size (unlike GET method that is capped by max-length of URL)
- The **POST method** **transfers information** via HTTP so its security depends on HTTP protocol:
 - ✓ By using Secure HTTP you can make sure that your information is secure
- To be covered:** PHP provides **\$_POST associative array** to access all the sent information using the POST method
- POST requests do not remain in the browser history and they **cannot be bookmarked**

File extension matters!

- How can the web server know if a requested file should be treated as a server-side program/script?
 - ✓ **File extension matters!**
 - ✓ When the extension is “**htm**” or “**html**” the web server knows what the client requests is a **static web page**
 - ✓ When the extension is a different one (e.g. “**php**” or “**asp**”), the web server knows a server-side program/script needs invoking



Response status codes

3. Redirection messages:

- ❖ e.g. [301 Moved Permanently](#): indicates that the resource requested has been definitively moved to the URL given by the **Location** headers

```
1 GET /index.php HTTP/1.1
2 Host: www.example.org
```

Server response 

```
1 HTTP/1.1 301 Moved Permanently
2 Location: http://www.example.org/index.asp
```

4. Client error responses:

- ❖ e.g. [404 Not Found](#): the server can not find requested resource
 - This response code is probably the most famous one due to its frequent occurrence on the web
- ❖ e.g. [403 Forbidden](#): indicates that the server understood the request but refuses to authorise it

5. Server error responses:

- ✓ e.g. 500 Internal Server Error
 - ❖ Wrong implementation of server side script



The Web/HTTP is stateless

- HTTP protocol itself is **stateless** (**memoryless**)
 - ✓ So is the Web!
 - ✓ Each HTTP request is processed without knowledge of any prior requests
 - ✓ A stateless protocol **does not** require the server to **retain information** or status about each user for the duration of multiple requests
- **No need to dynamically allocate storage** to deal with conversations in progress
- It **minimises time/bandwidth** that would otherwise be spent reestablishing a connection for each request
- If a client session dies in mid-transaction, no part of the system needs to be responsible for cleaning up the present state of the server



The Web/HTTP is stateless

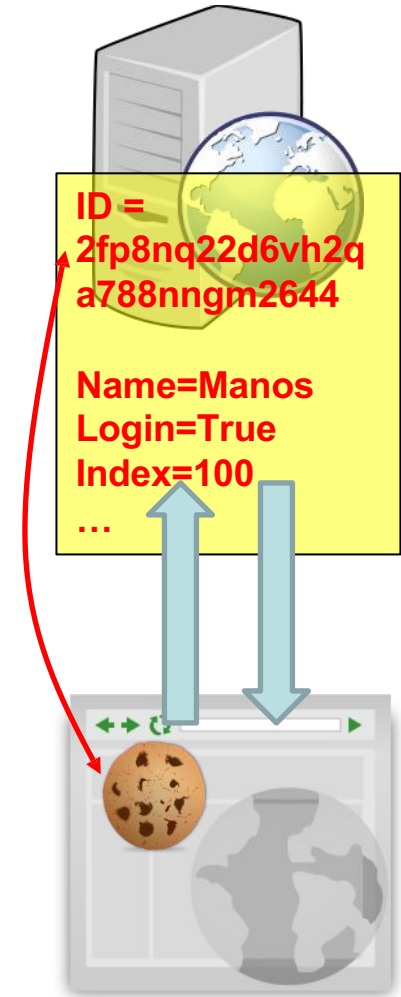
- In many applications, we want the web server to have memory of the client's visiting **history**
 - ✓ E.g., For a registered user of a website, he/she should be able to log in once for the whole website
- So we need some mechanism to make the Web (HTTP) **stateful** (= **memoryful**).
 - ✓ Save information about **historical visits** to the web server
 - ❖ So the web server keeps a memory of the past
- What **information** do we need **to save**?
 - ✓ Data submitted via HTML forms
 - ✓ Data submitted via URLs (query strings, may not be via a form)
 - ✓ **Who** visited **what**, **when** and **from where** and **how**?
 - ❖ Maximally, everything about the visit history of every visitor!



- Q: How do we save such information?
- A: Multiple ways
 - ✓ As **web (HTTP) cookies** (client-side data, short- and long-term)
 - ✓ As **session variables** (server-side data, short-term)
 - ✓ In **databases** (server-side, long-term)
 - ✓ In **data files** (server-side data, long-term)
- We will cover everything!

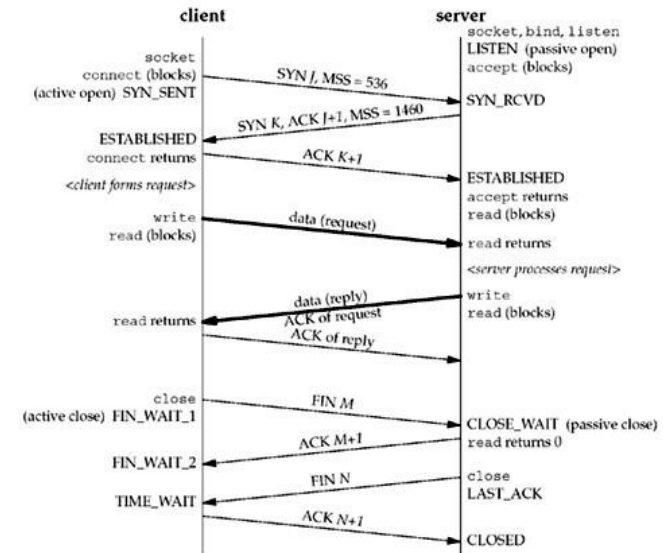
Sessions

- A (client-side) **Web/HTTP session** lasts **from the first time the user visits a web server until the web browser closes**
- Q: How is the **session ID** maintained during a Web session?
 - ✓ Stored as a (client-side) **session cookie**
 - ✓ **Communicated as a URL parameter** as part of the query string of each URL
- Note: Each **client-side session** has a **unique ID** and can be associated with a **number of session variables** stored at the **server side**



A typical HTTP session (3 phases)

1. The client **establishes** a TCP connection:
 - ✓ The client must know the **IP address** and the **port number** of the desired server
2. The client sends its request to the server, and waits for the answer
3. The server processes the request, sending back its answer, providing a **status code** and returned data

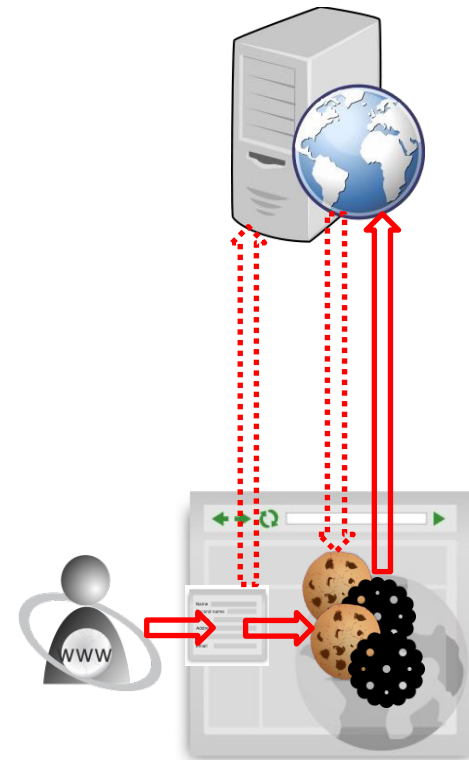


Code	Description	Code	Description
200	OK	400	Bad Request
201	Created	401	Unauthorized
202	Accepted	403	Forbidden
301	Moved Permanently	404	Not Found
303	See Other	410	Gone
304	Not Modified	500	Internal Server Error

Cookies

HTTP Cookies

- An HTTP cookie is a **small piece of text file** that is downloaded onto 'terminal equipment' after sent by the server to the user's web browser when the user accesses a website:
 - ✓ It remembers stateful information for the stateless HTTP protocol
- It allows the website to **recognise the user's device** and store some information about the user's **preferences** or **past actions**
- When receiving an HTTP request, a server can send a Set-Cookie header with the HTTP response
 - ✓ `Set-Cookie: <cookie-name>=<cookie-value>`

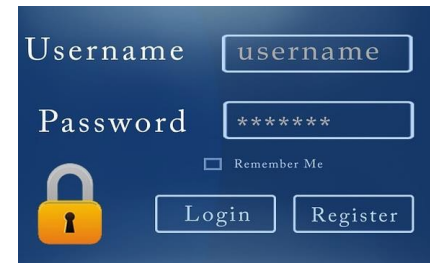


HTTP cookies

- Cookies can be used for a variety of reasons:
 - ✓ **PERSONALISATION** - **remember your preferences** on a site:
 - ❖ e.g. whether you read the oldest or newest comments first; the volume on the video player.
 - ✓ **TRACKING** - **understand how** you are **using** the site:
 - ❖ e.g. to tell what the most popular news story of the day is; to record how you responded to a new design or version of the site.
 - ✓ **SESSION MANAGEMENT** - for **logging in to a service** or to make sure you're **logged in securely**:
 - ❖ these cookies may contain information such as your email address and your name – the information you gave when you signed up
 - ❖ the website you signed up to **is the only site** that can access this information




(((BREAKING NEWS)))




Username

Password

☐ Remember Me





Cookies are sent with every request, so they can **worsen performance** (especially for mobile data connections)

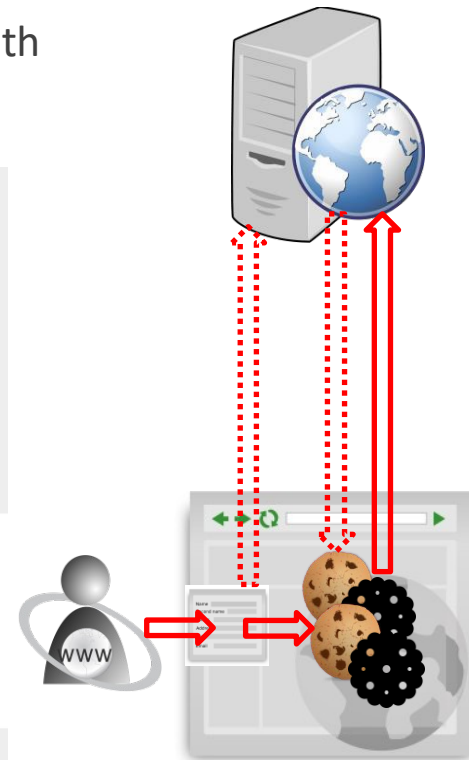
Creating cookies

- Cookie is composed by server-side program/script (e.g., PHP) and sent to the client via “Set-Cookie” in HTTP response
- The cookie can be [stored by the browser](#), and then the cookie is sent with requests made to the same server inside a [Cookie](#) HTTP header

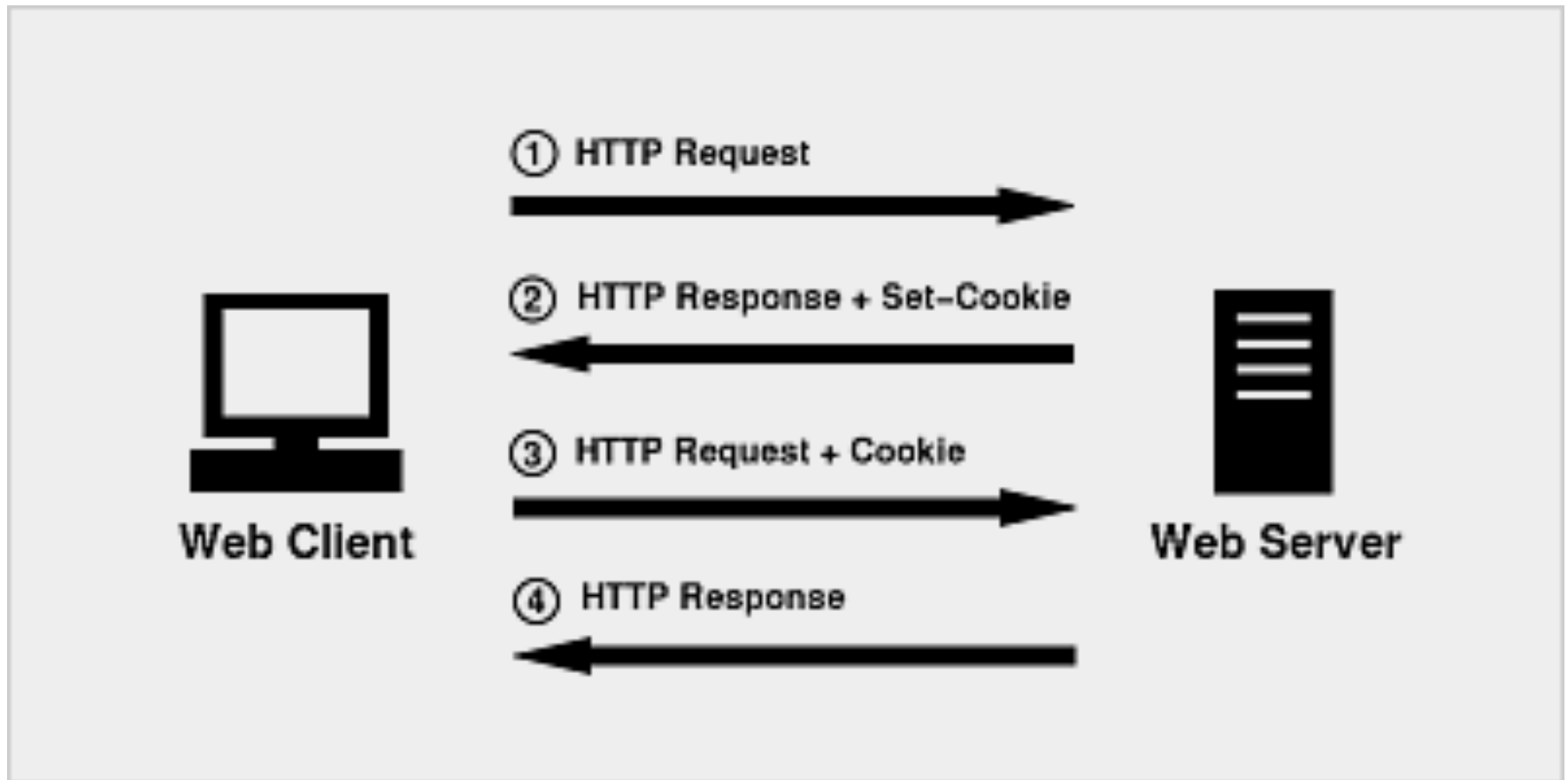
```
1 HTTP/1.0 200 OK
2 Content-type: text/html
3 Set-Cookie: yummy_cookie=choco
4 Set-Cookie: tasty_cookie=strawberry
5
6 [page content]
```

Now, with every new request to the server, the browser will send back all previously stored cookies to the server using the [Cookie](#) header.

```
1 GET /sample_page.html HTTP/1.1
2 Host: www.example.org
3 Cookie: yummy_cookie=choco; tasty_cookie=strawberry
```



HTTP cookies



Session and Permanent cookies

- The cookie created above is a *session cookie*
 - ✓ it is deleted when the client shuts down, because it didn't specify an Expires or Max-Age directive.
 - ✓ However, web browsers may use **session restoring**, which makes most session cookies permanent, as if the browser was never closed.
- Instead of expiring when the client closes, *permanent cookies* expire at a specific date (Expires) or after a specific length of time (Max-Age).

```
1 | Set-Cookie: id=a3fWa; Expires=Wed, 21 Oct 2015 07:28:00 GMT;
```

📌 **Note:** When an expiry date is set, the time and date set is relative to the client the cookie is being set on, not the server.

- Cookies are stored for **a particular domain**:
 - ✓ They can be made accessible to a single domain name (e.g. `www.foo.com`) or all its subdomain names (e.g. `*.foo.com`).
- Cookies have **an expiration time**:
 - ✓ Default: expire once the web browser closes = **session cookies**
- Secure Cookies (*): Cookies made valid for HTTPS only
 - ✓ A secure cookie will only be sent to the server when a request is made using SSL and the HTTPS protocol
 - ✓ However, **confidential or sensitive information should never be stored or transmitted in HTTP Cookies** as the entire mechanism is inherently insecure and this doesn't mean that any information is encrypted



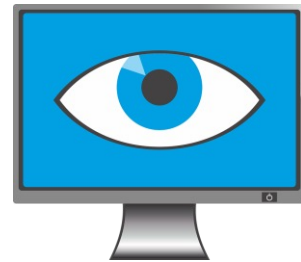
- HttpOnly Cookies (*):
 - ✓ Cookies available to server only
 - ✓ HttpOnly is an **additional flag** included in a **Set-Cookie HTTP response header**
 - ✓ Using the HttpOnly flag when generating a cookie helps mitigate the risk of client side script accessing the protected cookie.
 - ❖ To prevent cross-site scripting ([XSS](#)) attacks, HttpOnly cookies are inaccessible to JavaScript's [Document.cookie](#) API



```
Set-Cookie: id=a3fWa; Expires=Wed, 21 Oct 2015 07:28:00 GMT; Secure; HttpOnly
```

Privacy?

- Contrary to a common belief, **cookies do not contain software programs**, so cannot install anything on a computer
- Cookies generally **do not contain any information** that would identify a person (**Personal Identifiable Information - PII**)
 - ✓ Usually they contain a string of text or "unique identifier" acting like a label
 - ✓ When a website sees the string of text it set in a cookie, it knows the browser is one it has seen before
- However, cookies **may violate the user's privacy** as it gives the remote server **access to local data**
- Cookies **can be disabled** in the web browser by the user



The GDPR, Cookie Consent

- The EU General Data Protection Regulation (**GDPR**) has now been in force
- The **GDPR** is an over-arching piece of legislation dealing with all aspects of the processing of personal information
- Cookies are mentioned once in the **GDPR**, in [Recital 30](#):
 - ✓ *“Natural persons may be associated with online identifiers...such as internet protocol addresses, [cookie identifiers](#) or other identifiers.... This may leave traces which, in particular when combined with unique identifiers and other information received by the servers, may be used to create profiles of the natural persons and identify them.”*

End of Week 8