بسم الله الرحمن الرحيم

تکنولۆژی کامپیوتر

جلسەی هفتم

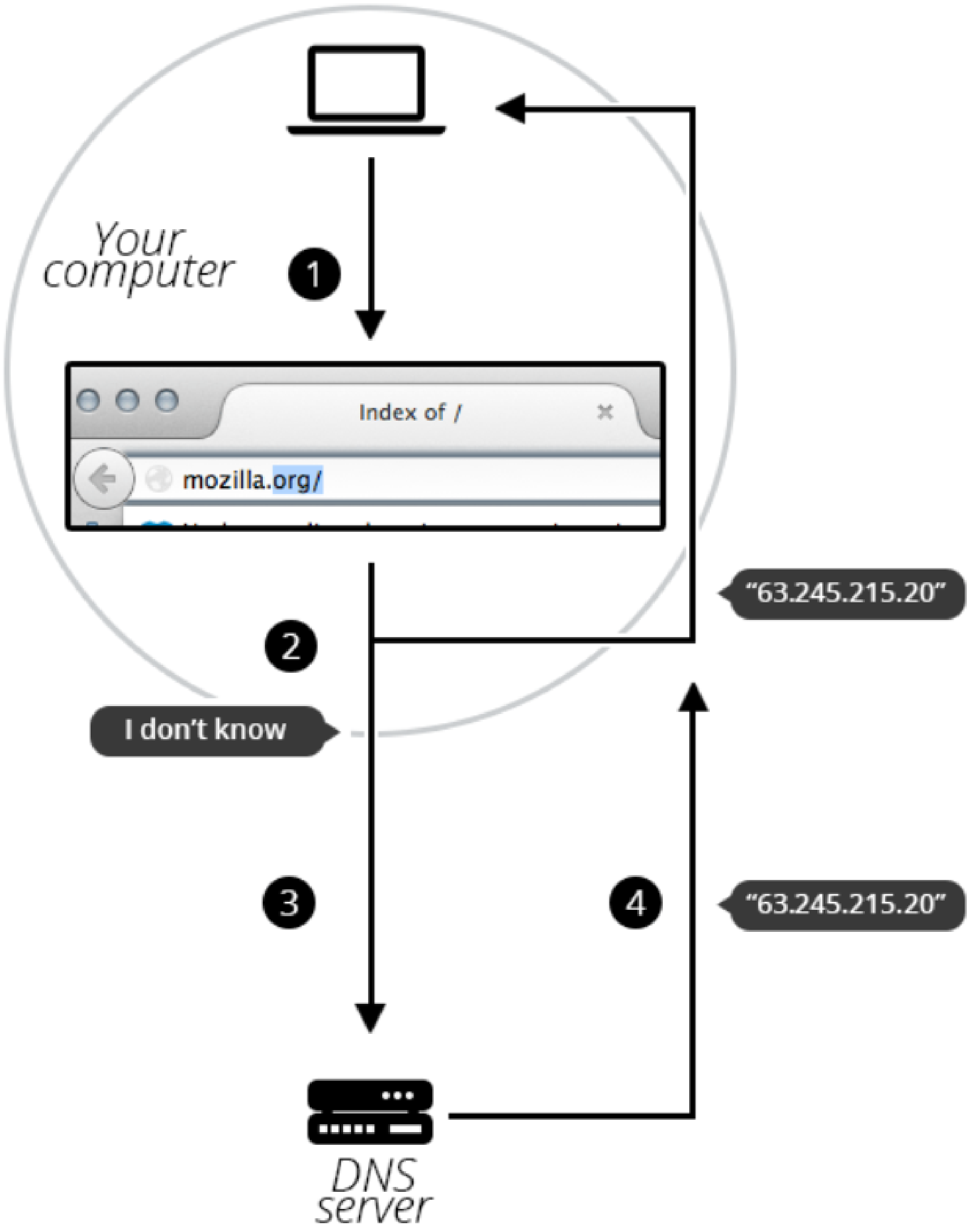
برنامەنویسی وب

Original Images / Texts of this lecture: https://developer.mozilla.org/en-US/docs

جلسه‌ی گذشته

اچ تی تی پی

DNS



Your computer
1
Index of /
mozilla.org/
2
I don't know
"63.245.215.20"
3
4
"63.245.215.20"
DNS server

TCP

- **Reliable Data Transfer**: Lost or corrupt segments are detected and retransmitted.
- **Ordered Delivery**: Sequence numbers ensure data is reassembled in the correct order.
- **Flow Control**: The receiving end can tell the sender how much data it can handle at once.
- **Congestion Control**: TCP tries to sense network congestion and adjust the sending rate, helping to avoid overwhelming the network.
- **Connection-Oriented**: The handshake before data transfer ensures both ends agree on parameters (sequence numbers, MSS, etc.).

URL

- http://www.example.com:80/path/to/myfile.html?key1=value1&key2=value2#SomewhereInTheDocument

URL

Protocol / Scheme

$\nearrow$

- http://www.example.com:80/path/to/myfile.html?key1=value1&key2=value2#SomewhereInTheDocument

URL

$$\begin{tikzcd}\text{Authority / Host} \\ \text{www.example.com} \arrow[u]\end{tikzcd}$$

- http://www.example.com:80/path/to/myfile.html?key1=value1&key2=value2#SomewhereInTheDocument

URL

Port

$\uparrow$

- http://www.example.com:80/path/to/myfile.html?key1=value1&key2=value2#SomewhereInTheDocument

URL

Path

- http://www.example.com:80/path/to/myfile.html?key1=value1&key2=value2#SomewhereInTheDocument

URL

- http://www.example.com:80/path/to/myfile.html?key1=value1&key2=value2#SomewhereInTheDocument

$\downarrow$

Query / Params

URL

- http://www.example.com:80/path/to/myfile.html?key1=value1&key2=value2#SomewhereInTheDocument

$\swarrow$

Anchor / Fragment / Location Hash

HTTP (Hypertext Transfer Protocol)

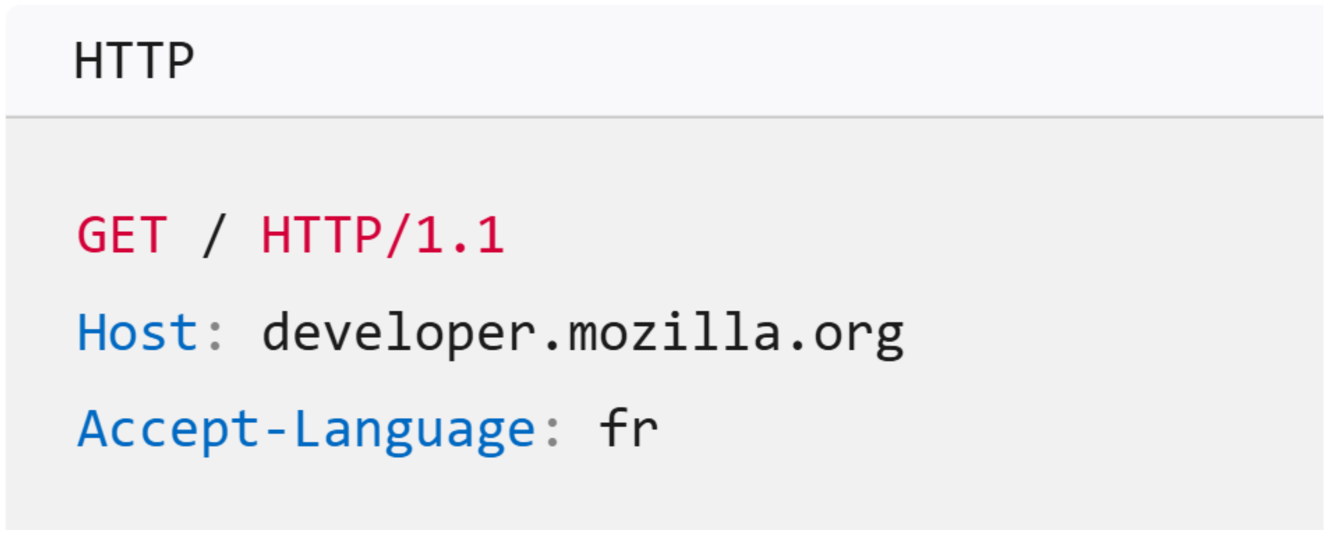
- HTTP 0.9, 1.0, 1.1, 2.0, 3.0
- Some details of evolution of HTTP in:
  - *https://developer.mozilla.org/en-US/docs/Web/HTTP/Evolution_of_HTTP*

HTTP Flow

1. **Open a TCP connection:** The TCP connection is used to send a request, or several, and receive an answer. The client may open a new connection, reuse an existing connection, or open several TCP connections to the servers.
   - *HTTP default port: 80*
   - *HTTPS default port: 443*

HTTP Flow

2. **Send an HTTP message:** HTTP messages (before HTTP/2) are human-readable. With HTTP/2, these simple messages are encapsulated in frames, making them impossible to read directly, but the principle remains the same. For example:



HTTP

```
GET / HTTP/1.1
Host: developer.mozilla.org
Accept-Language: fr
```

HTTP Flow

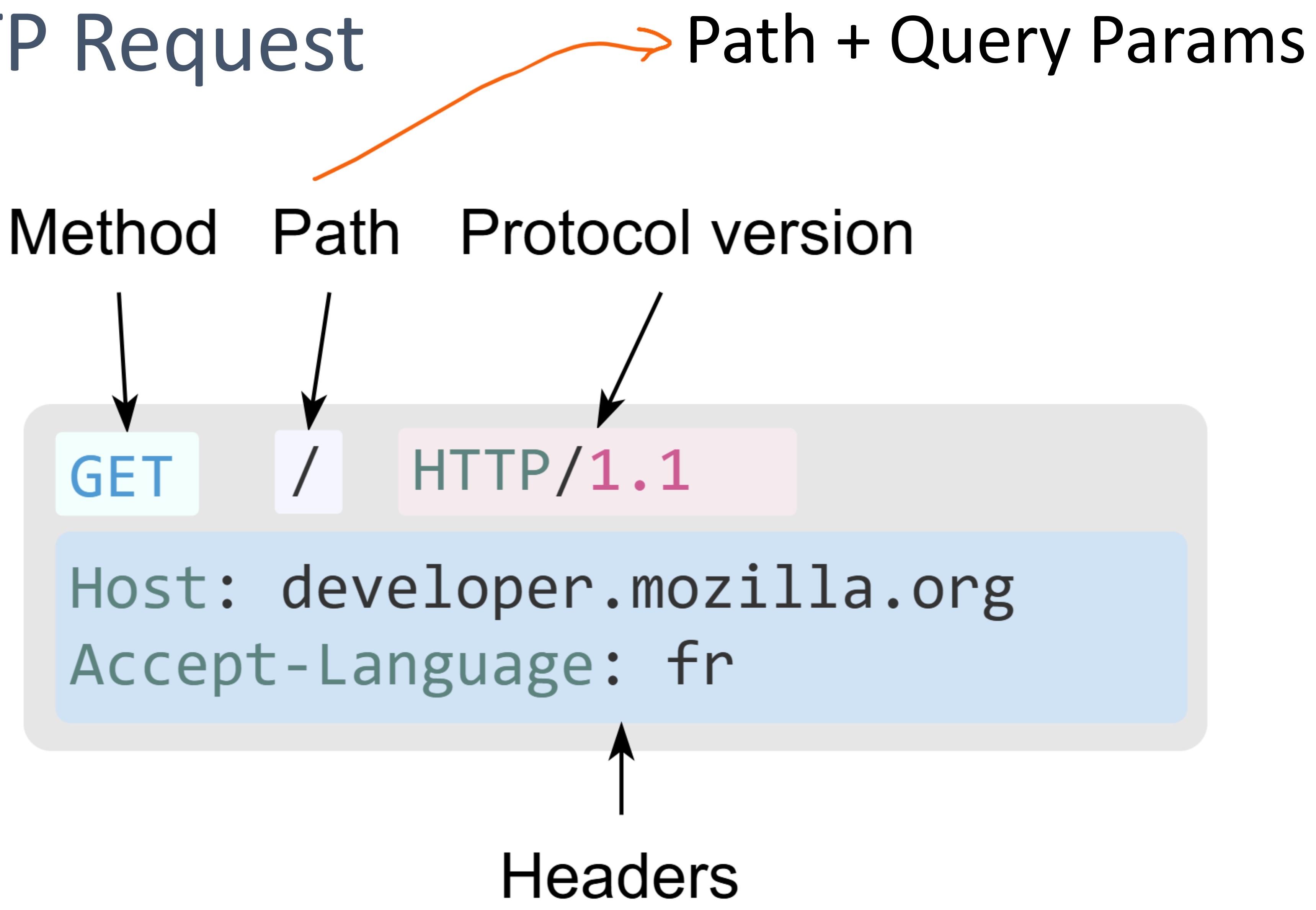
3. Read the response sent by the server, such as:

```
HTTP

HTTP/1.1 200 OK
Date: Sat, 09 Oct 2010 14:28:02 GMT
Server: Apache
Last-Modified: Tue, 01 Dec 2009 20:18:22 GMT
ETag: "51142bc1-7449-479b075b2891b"
Accept-Ranges: bytes
Content-Length: 29769
Content-Type: text/html

<!doctype html>… (here come the 29769 bytes of the requested web page)
```

HTTP Flow

4. Close or reuse the connection for further requests.

HTTP Request



```
GET / HTTP/1.1
Host: developer.mozilla.org
Accept-Language: fr
```

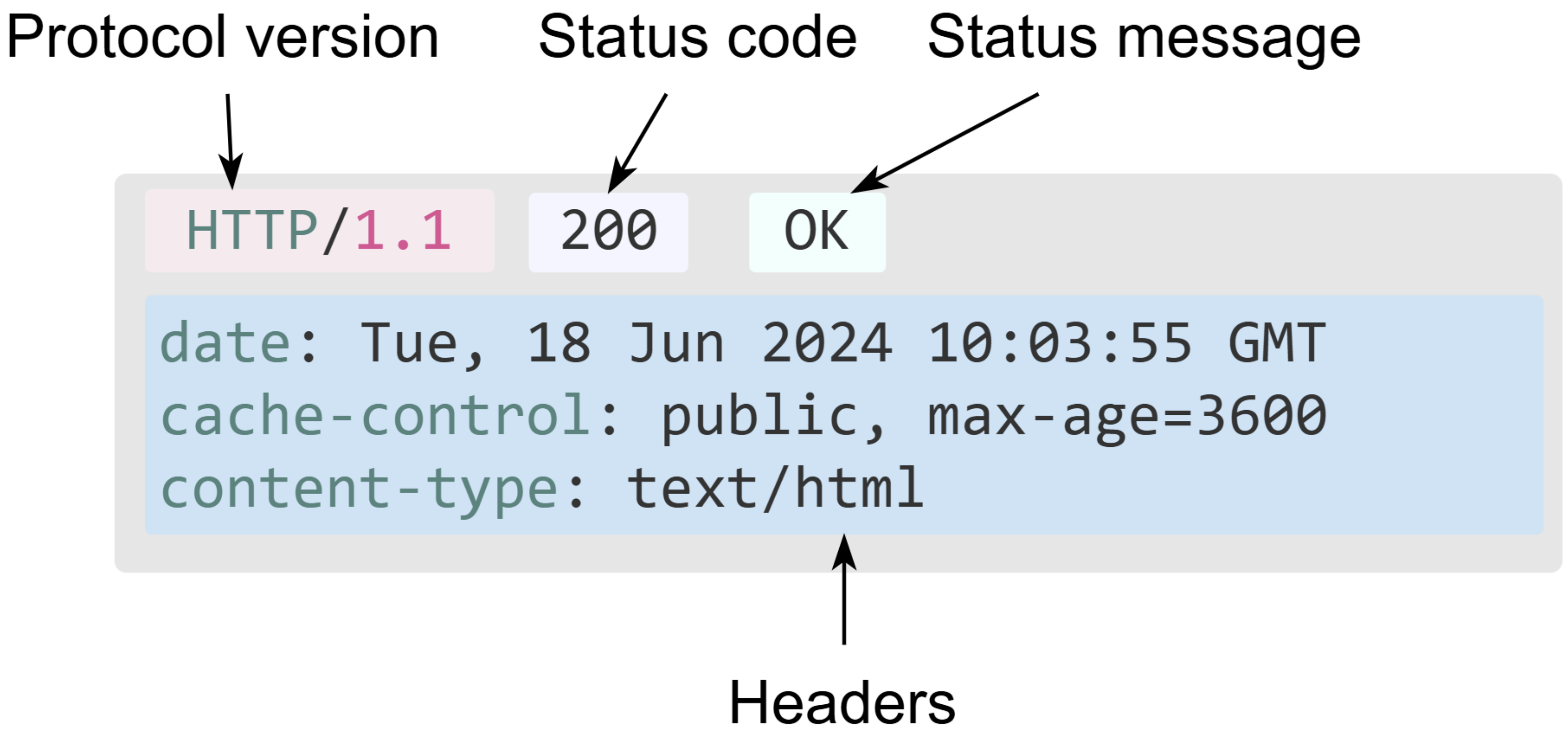
HTTP Request

- Requests consist of the following elements:
  - *An HTTP method, usually a verb like GET, POST, or a noun like OPTIONS or HEAD that defines the operation the client wants to perform. Typically, a client wants to fetch a resource (using GET) or post the value of an HTML form (using POST), though more operations may be needed in other cases.*
  - *The path of the resource to fetch; the URL of the resource stripped from elements that are obvious from the context, for example without the protocol (http://), the domain (here, developer.mozilla.org), or the TCP port (here, 80).*
  - *The version of the HTTP protocol.*
  - *Optional headers that convey additional information for the servers.*
  - *A body, for some methods like POST, similar to those in responses, which contain the resource sent.*

Methods

- GET: Retrieve a resource (no body typically).
- POST: Send data to the server (e.g., form submissions, file uploads).
- PUT: Update or replace a resource.
- PATCH: Partial update of a resource.
- DELETE: Delete a resource.
- HEAD: Same as GET but returns no body (just headers).
- OPTIONS: Query the server for supported HTTP methods or capabilities.

HTTP Response



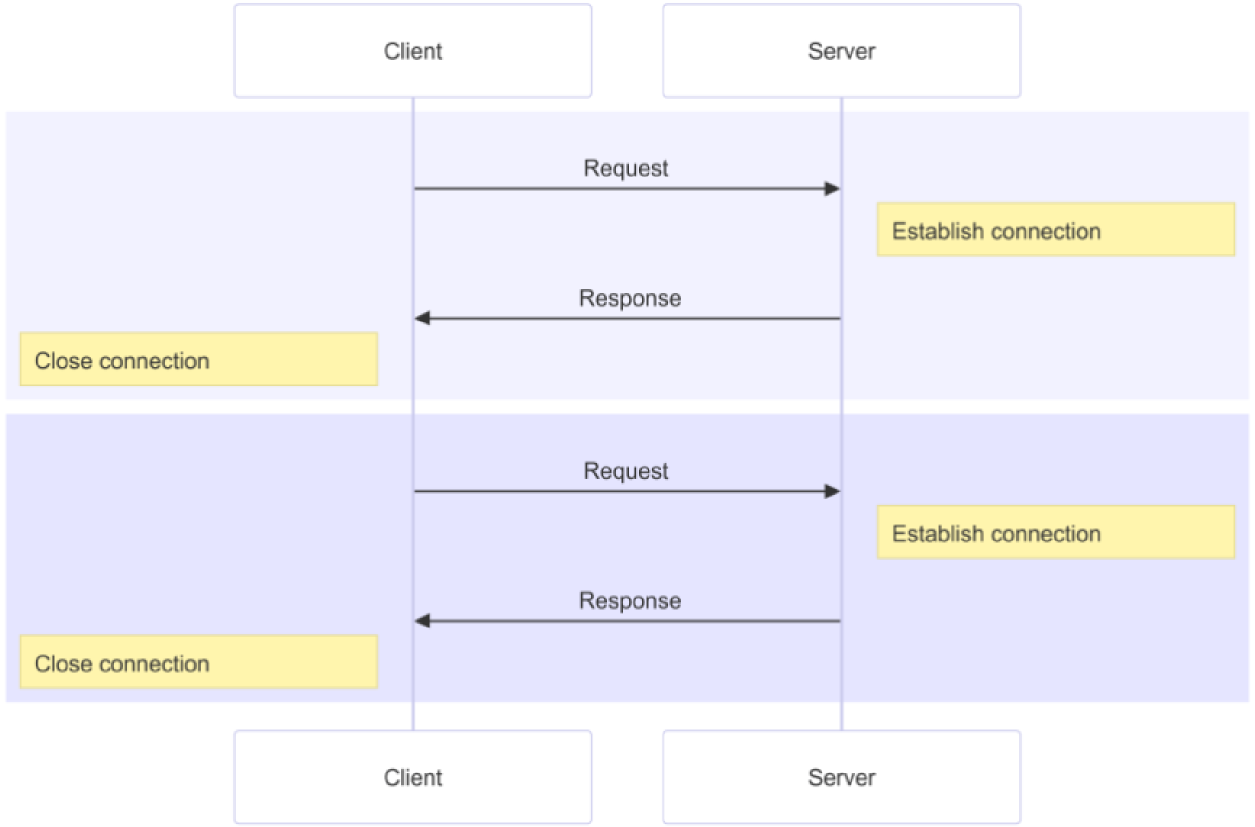
```
HTTP/1.1 200 OK
date: Tue, 18 Jun 2024 10:03:55 GMT
cache-control: public, max-age=3600
content-type: text/html
```

HTTP Response

- Responses consist of the following elements:
  - *The version of the HTTP protocol they follow.*
  - *A status code, indicating if the request was successful or not, and why.*
  - *A status message, a non-authoritative short description of the status code.*
  - *HTTP headers, like those for requests.*
  - *Optionally, a body containing the fetched resource.*

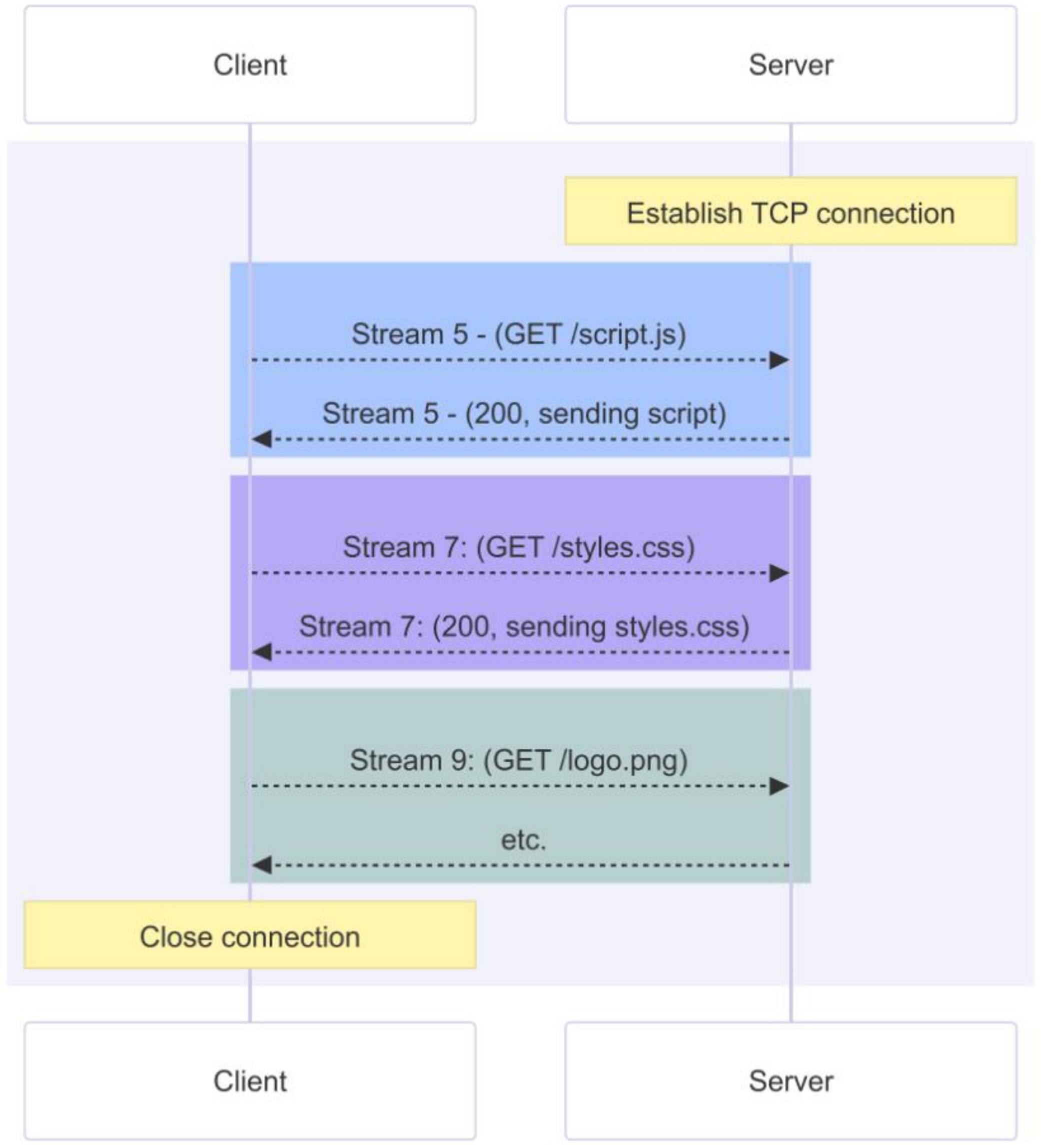
HTTP 1.1 vs HTTP 2

- In HTTP/1.1 if you want to make two requests in parallel, you have to open two connections:



HTTP 1.1 vs HTTP 2

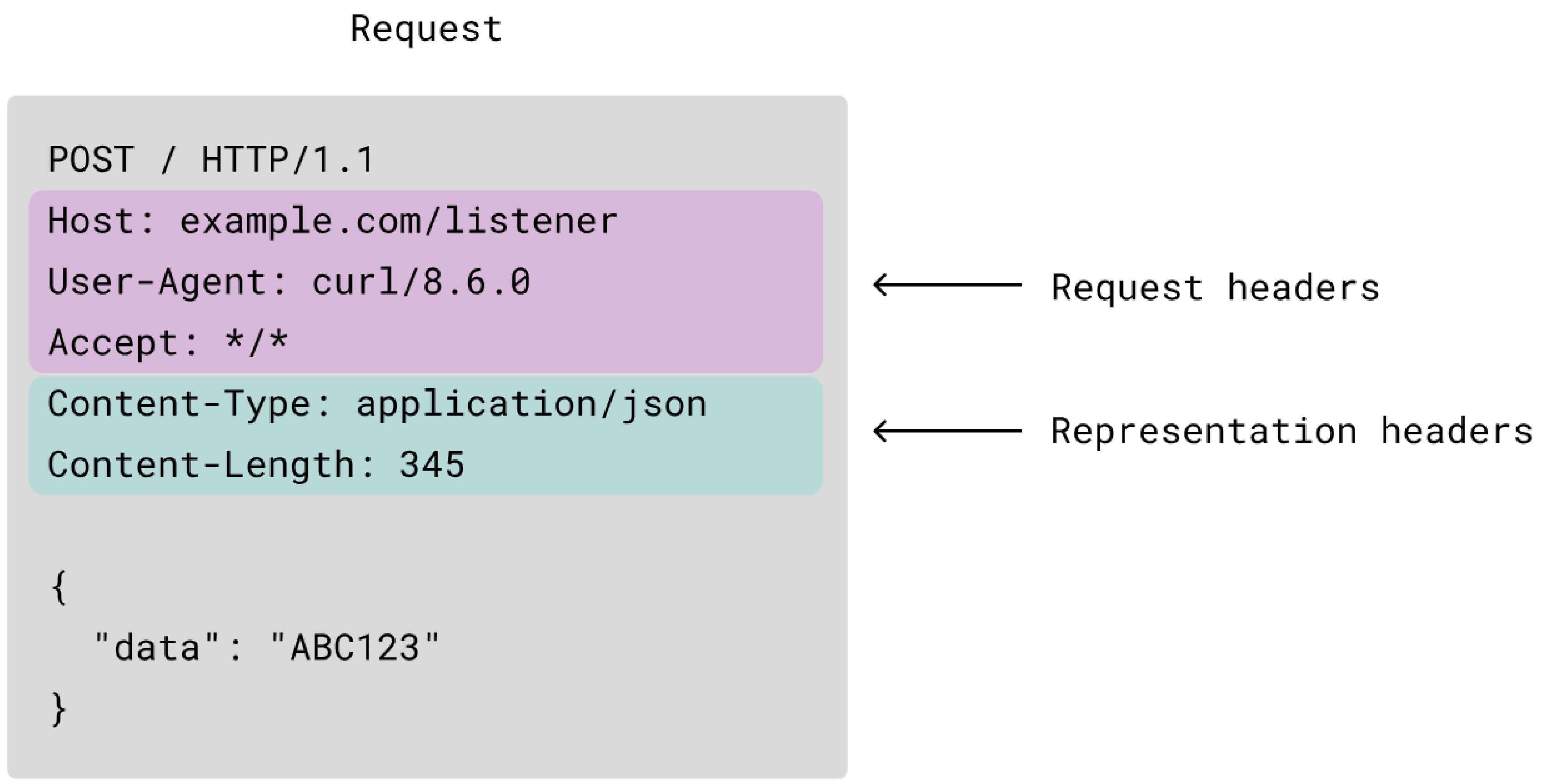
- HTTP/2 allows you to use a single TCP connection for multiple requests and responses at the same time
- Requests are not necessarily sequential: stream 9 doesn't have to wait for stream 7 to finish, for instance.


Client
Server
Establish TCP connection
Stream 5 - (GET /script.js)
Stream 5 - (200, sending script)
Stream 7: (GET /styles.css)
Stream 7: (200, sending styles.css)
Stream 9: (GET /logo.png)
etc.
Close connection
Client
Server

HTTP 1.1 vs HTTP 2

- One notable change to messages in HTTP/2 are the use of pseudo-headers. Where HTTP/1.x used the message start-line, HTTP/2 uses special pseudo-header fields beginning with :. In requests, there are the following pseudo-headers:
  - ***:method** - the HTTP method.*
  - ***:scheme** - the scheme portion of the target URI, which is often HTTP(S).*
  - ***:authority** - the authority portion of the target URI.*
  - ***:path** - the path and query parts of the target URI.*
- In responses, there is only one pseudo-header, and that's the **:status** which provides the code of the response.

HTTP Request Headers



```
POST / HTTP/1.1
Host: example.com/listener
User-Agent: curl/8.6.0
Accept: */*
Content-Type: application/json
Content-Length: 345

{
  "data": "ABC123"
}
```

Representation Headers

- A representation header (or 'representation metadata') is an HTTP header that describes how to interpret the data contained in the message.
  - *Content-Length*
    - The HTTP Content-Length header indicates the size, in bytes, of the message body sent to the recipient.
  - *Content-Range: <unit> <range-start>-<range-end>/<size>*
    - https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Range
  - *Content-Type: <type>/<subtype>[;<parameter>=<value>]*
    - https://developer.mozilla.org/en-US/docs/Web/HTTP/MIME_types

Representation Headers

- A representation header (or 'representation metadata') is an HTTP header that describes how to interpret the data contained in the message.
  - *Content-Encoding*
    - lists the encodings and the order in which they have been applied to a resource
  - *Content-Location*
  - *Content-Language*

HTTP Request Body

- The request body is the part of a request that carries information to the server. Only PATCH, POST, and PUT requests have a body

HTTP Request Body

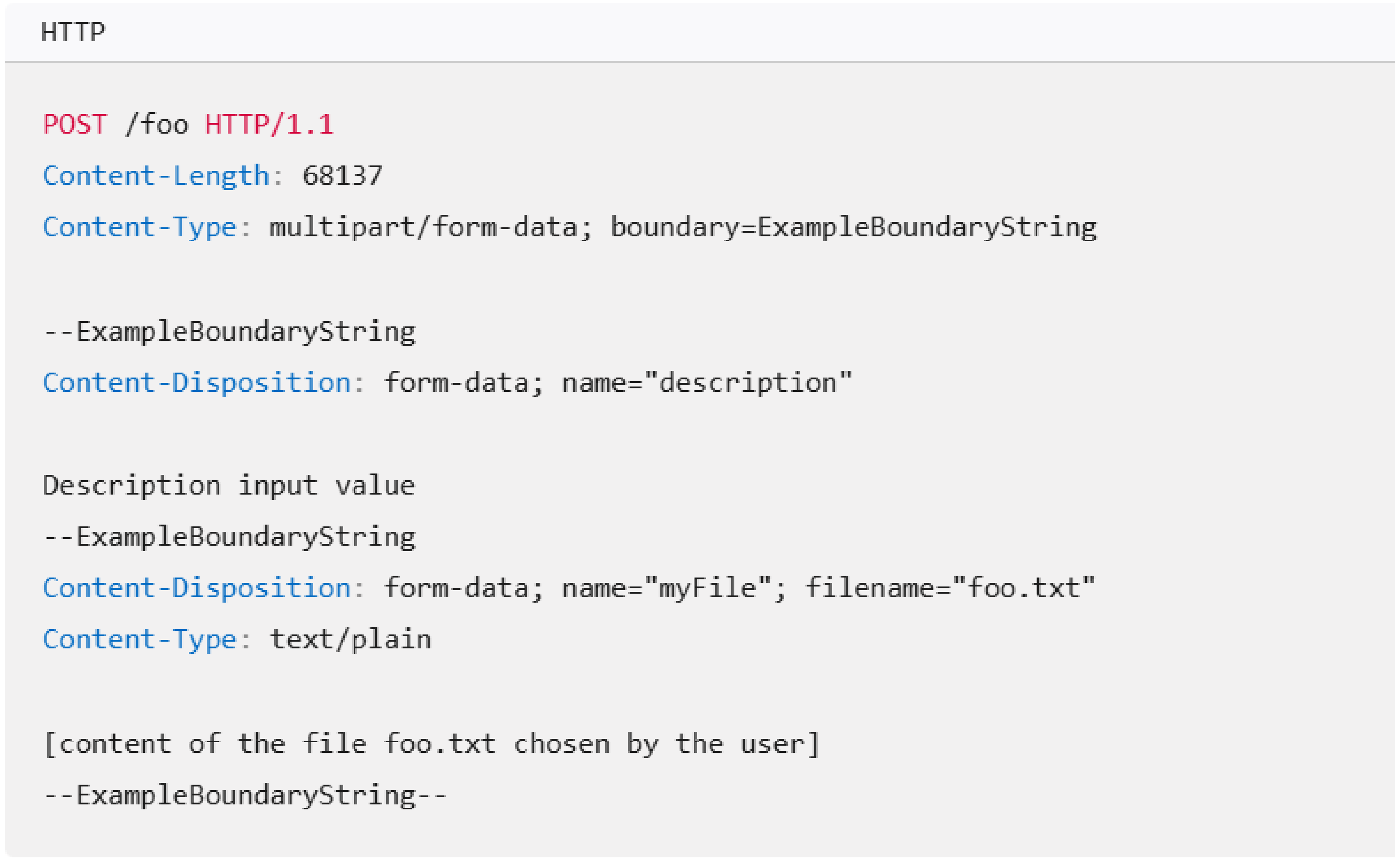
- Form submission example:

HTTP

```
POST /submit HTTP/1.1
Host: example.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 15

comment=Hello!
```

HTTP Request Body

- Body in multipart forms



HTTP

```
POST /foo HTTP/1.1
Content-Length: 68137
Content-Type: multipart/form-data; boundary=ExampleBoundaryString

--ExampleBoundaryString
Content-Disposition: form-data; name="description"

Description input value
--ExampleBoundaryString
Content-Disposition: form-data; name="myFile"; filename="foo.txt"
Content-Type: text/plain

[content of the file foo.txt chosen by the user]
--ExampleBoundaryString--
```

HTTP Request Body

- Body in a REST API using JSON

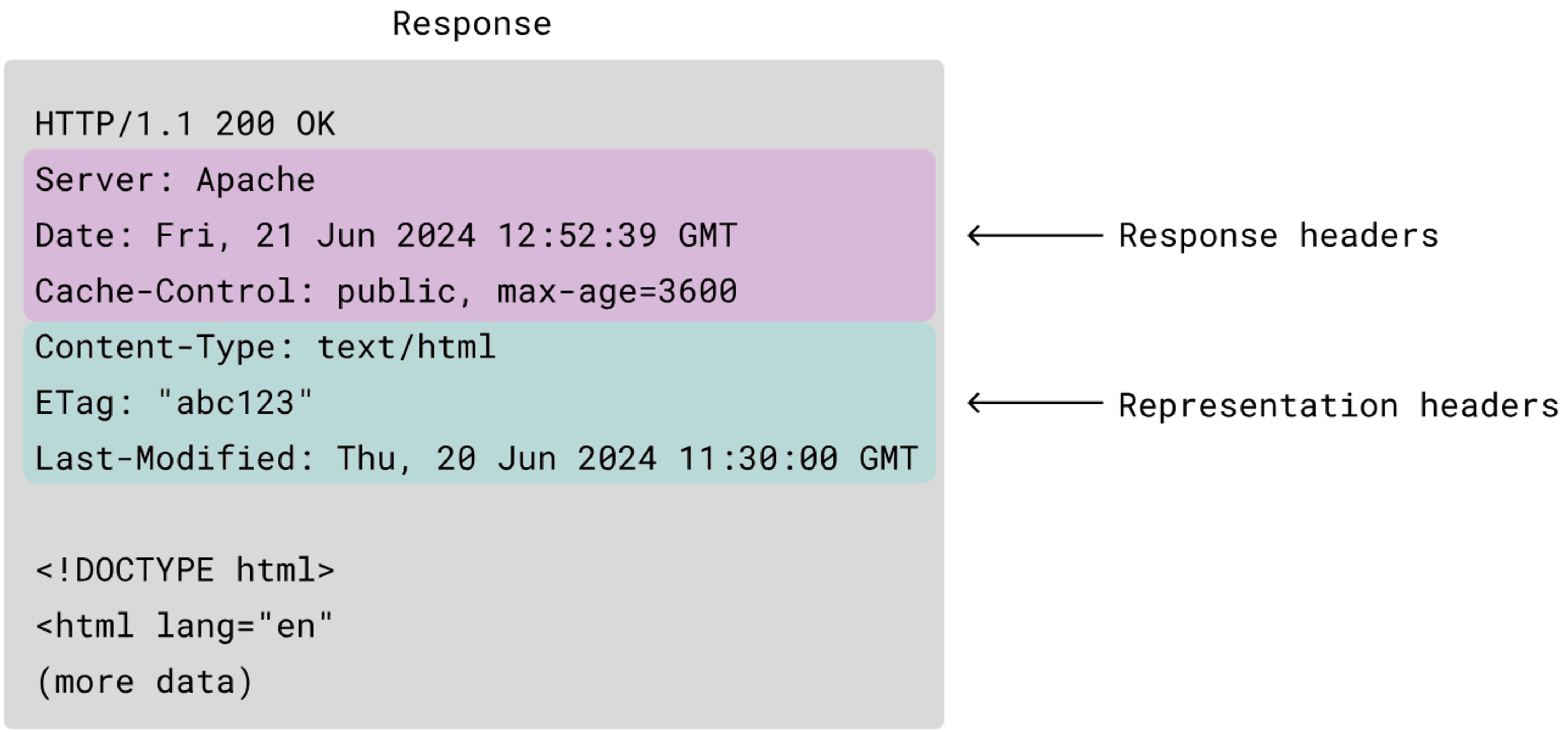
```
POST /submit HTTP/1.1
Content-Type: application/json

{
  "message": "New user created",
  "user": {
    "id": 123,
    "firstName": "Paul",
    "lastName": "Klee",
    "email": "p.klee@example.com"
  }
}
```

HTTP Request Headers

HTTP

```
GET /home.html HTTP/1.1
Host: developer.mozilla.org
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9; rv:50.0) Gecko/20100101 Firefox/50.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Referer: https://developer.mozilla.org/testpage.html
Connection: keep-alive
Upgrade-Insecure-Requests: 1
If-Modified-Since: Mon, 18 Jul 2016 02:36:04 GMT
If-None-Match: "c561c68d0ba92bbeb8b0fff2a9199f722e3a621a"
Cache-Control: max-age=0
```

HTTP Response Headers



```
HTTP/1.1 200 OK
Server: Apache
Date: Fri, 21 Jun 2024 12:52:39 GMT
Cache-Control: public, max-age=3600
Content-Type: text/html
ETag: "abc123"
Last-Modified: Thu, 20 Jun 2024 11:30:00 GMT

<!DOCTYPE html>
<html lang="en"
(more data)
```

HTTP Status Code

- Informational responses (100 – 199)
- Successful responses (200 – 299)
- Redirection messages (300 – 399)
- Client error responses (400 – 499)
- Server error responses (500 – 599)
- Details: https://developer.mozilla.org/en-US/docs/Web/HTTP/Status

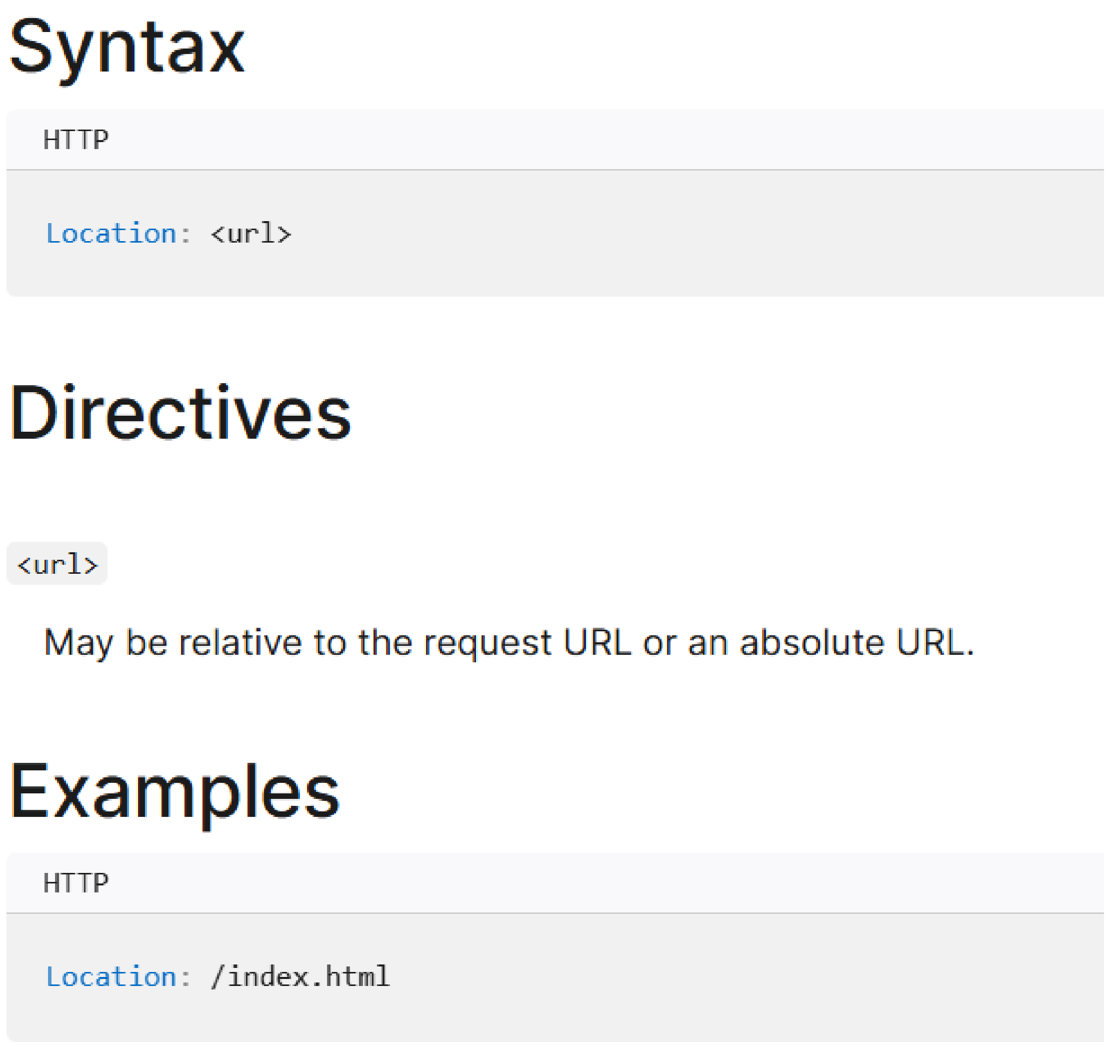
HTTP Status Code

- Successful responses (200 – 299)
  - *200 OK*
  - *201 Created*
  - *204 No Content*
  - *206 Partial Content*

HTTP Status Code

- Redirection messages (300 – 399)
  - *301 Moved Permanently*
    - The URL of the requested resource has been changed permanently. The new URL is given in the response.
  - *302 Found*
    - This response code means that the URI of requested resource has been changed temporarily. Further changes in the URI might be made in the future, so the same URI should be used by the client in future requests.
  - *304 Not Modified*
    - This is used for caching purposes. It tells the client that the response has not been modified, so the client can continue to use the same cached version of the response.
  - *307 Temporary Redirect*
    - The server sends this response to direct the client to get the requested resource at another URI with the same method that was used in the prior request. This has the same semantics as the 302 Found response code, with the exception that the user agent must not change the HTTP method used: if a POST was used in the first request, a POST must be used in the redirected request.
  - *308 Permanent Redirect*

HTTP Redirection



Syntax

HTTP

```
Location: <url>
```

Directives

`<url>`

May be relative to the request URL or an absolute URL.

Examples

HTTP

```
Location: /index.html
```

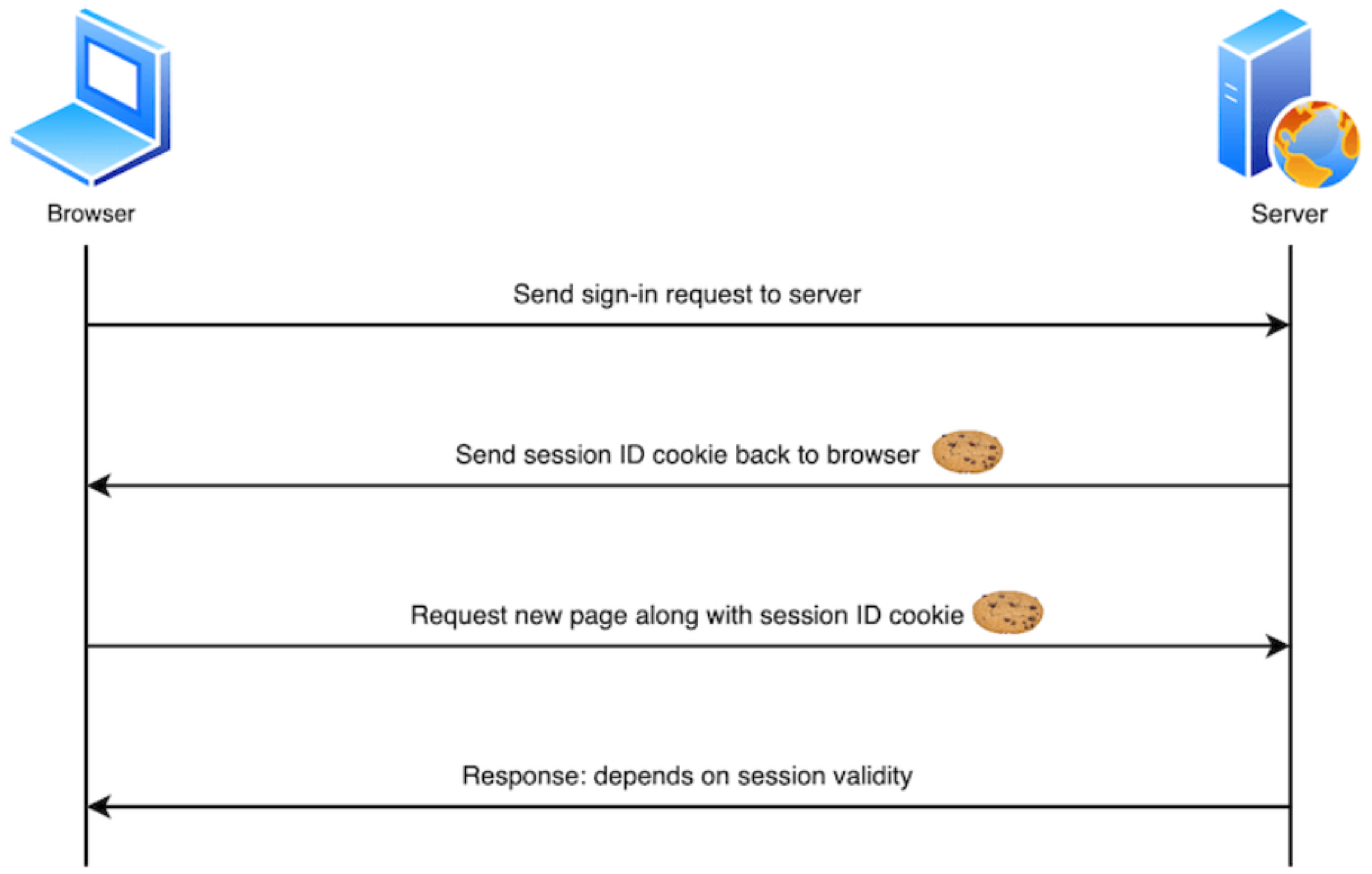
HTTP Status Code

- Client error responses (400 – 499)
  - *400 Bad Request*
  - *401 Unauthorized*
  - *403 Forbidden*
  - *404 Not Found*
  - *405 Method Not Allowed*
  - *429 Too Many Requests*

HTTP Status Code

- Server error responses (500 – 599)
  - *500 Internal Server Error*
  - *502 Bad Gateway*
  - *503 Service Unavailable*
  - *504 Gateway Timeout*

Cookies and Sessions

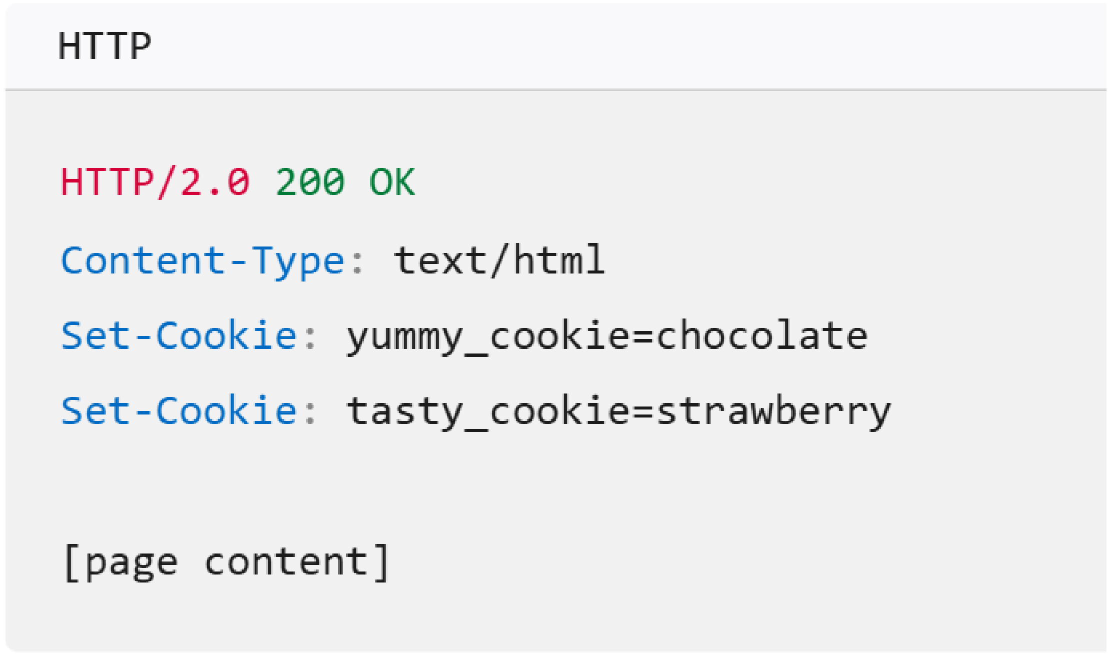

Browser
Server
Send sign-in request to server
Send session ID cookie back to browser
Request new page along with session ID cookie
Response: depends on session validity

Cookies and Sessions

- Cookies are mainly used for three purposes:
  - ***Session management**: User sign-in status, shopping cart contents, game scores, or any other user session-related details that the server needs to remember.*
  - ***Personalization**: User preferences such as display language and UI theme.*
  - ***Tracking**: Recording and analyzing user behavior.*

Set Cookie by HTTP Response

- Header
  - *Set-Cookie: <cookie-name>=<cookie-value>*



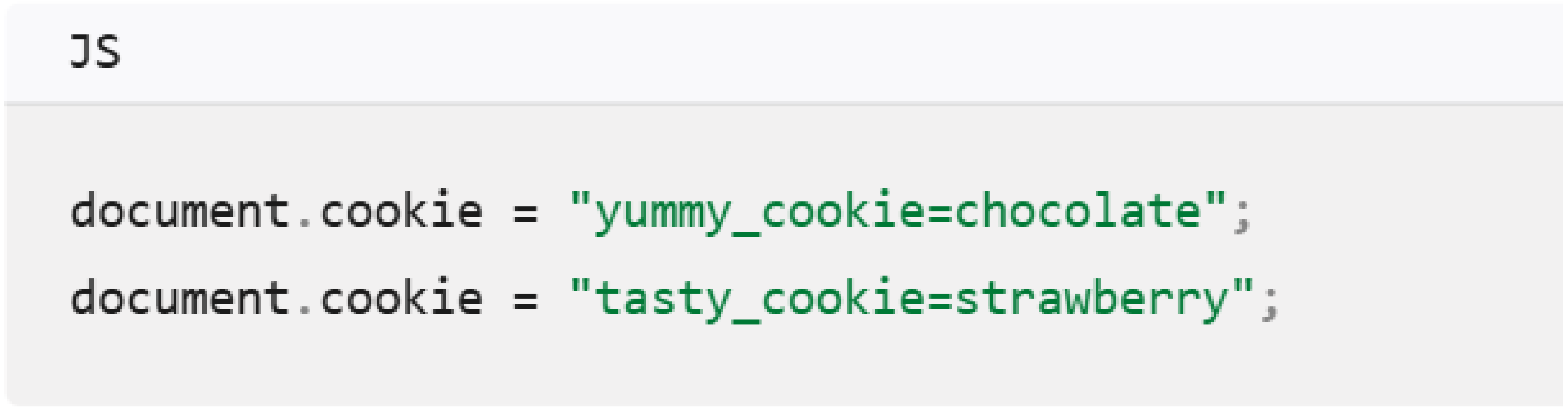
HTTP

```
HTTP/2.0 200 OK
Content-Type: text/html
Set-Cookie: yummy_cookie=chocolate
Set-Cookie: tasty_cookie=strawberry

[page content]
```

Life Time of Cookie

- Set-Cookie: id=a3fWa; Expires=Thu, 31 Oct 2021 07:28:00 GMT;
- Set-Cookie: id=a3fWa; Max-Age=2592000
- **Session cookies**
  - *cookies without a Max-Age or Expires attribute*
  - *They are deleted when the current session ends*

Updating Cookie via JavaScript



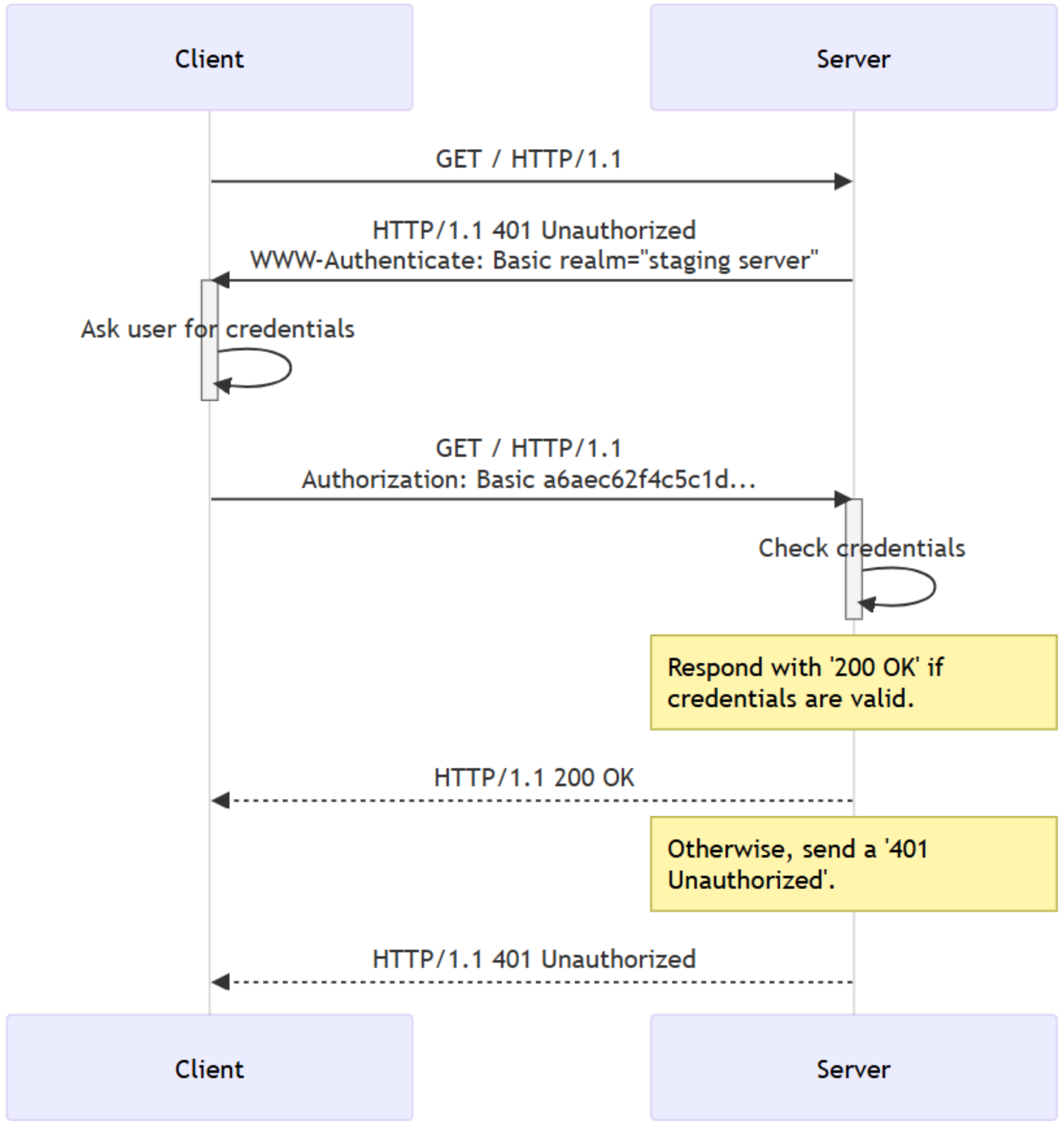
JS

```
document.cookie = "yummy_cookie=chocolate";
document.cookie = "tasty_cookie=strawberry";
```

Cookies and Security!

- HttpOnly Cookies
- Secure Cookies
- Read this: https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies#security

Authentication


Client
Server
GET / HTTP/1.1
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Basic realm="staging server"
Ask user for credentials
GET / HTTP/1.1
Authorization: Basic a6aec62f4c5c1d...
Check credentials
Respond with '200 OK' if credentials are valid.
HTTP/1.1 200 OK
Otherwise, send a '401 Unauthorized'.
HTTP/1.1 401 Unauthorized
Client
Server

HTTP Caching

- Read: https://developer.mozilla.org/en-US/docs/Web/HTTP/Caching

جلسه‌ی جدید

HTML - HyperText Markup Language

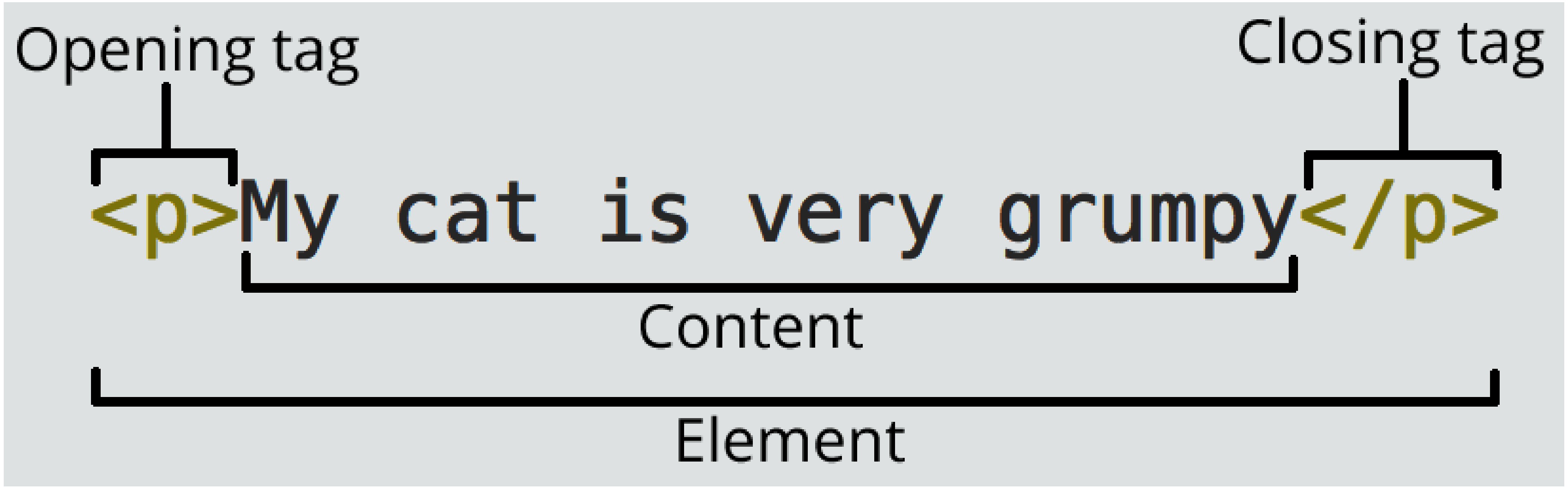
تمرین با اچ‌تی‌تی‌پی

شروع HTML

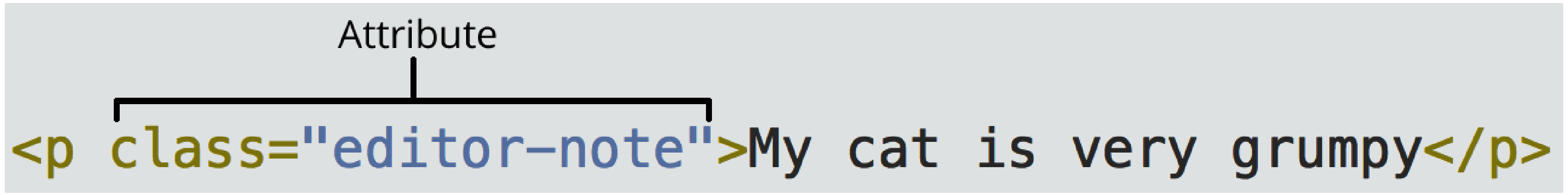
- HTML
- CSS
- JS

Note! Browser Support…

Anatomy of an HTML element



Anatomy of an HTML element



```
<p class="editor-note">My cat is very grumpy</p>
```

- Attributes that set a value always have:
  - *A space between it and the element name (or the previous attribute, if the element already has one or more attributes).*
  - *The attribute name followed by an equal sign.*
  - *The attribute value wrapped by opening and closing quotation marks.*

Nesting elements

HTML

```
<p>My cat is <strong>very</strong> grumpy.</p>
```

HTML

```
<p>My cat is <strong>very grumpy.</p></strong>
```

Void elements

```html
<img src="images/firefox-icon.png" alt="My test image" />
```

Comments

- Anything in HTML between <!-- and --> is an HTML comment.
- The browser ignores comments as it renders the code. In other words, they are not visible on the page - just in the code.
- <!-- this is a comment in HTML -->

What about < and > characters?

- < : &lt;
- > : &gt;
- &: &amp;
- More in: https://www.degraeve.com/reference/specialcharacters.php

What Tags we have?

- A good reference: https://www.w3schools.com/TAGs/ref_byfunc.asp

First HTML Document

```
<!doctype html>
<html lang="en-US">
  <head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width" />
    <title>My test page</title>
  </head>
  <body>
    <img src="" alt="My test image" />
  </body>
</html>
```

First HTML Document

- <!doctype html>
  - *The doctype is a required preamble. In the mists of time, when HTML was young (around 1991/92), doctypes were meant to act as links to a set of rules that the HTML page had to follow to be considered good HTML, which could mean automatic error checking and other useful things. However, these days, they don't do much and are basically just needed to make sure your document behaves correctly. That's all you need to know for now.*
- <html></html>
  - *the <html> element. This element wraps all the content on the entire page and is sometimes known as the root element. It also includes the lang attribute, setting the primary language of the document.*

First HTML Document

- <head></head>
  - *This element acts as a container for all the stuff you want to include on the HTML page that isn't the content you are showing to your page's viewers. This includes things like keywords and a page description that you want to appear in search results, CSS to style our content, character set declarations, and more.*
- <meta charset="utf-8">
  - *This element sets the character set your document should use to UTF-8 which includes most characters from the vast majority of written languages. Essentially, it can now handle any textual content you might put on it. There is no reason not to set this, and it can help avoid some problems later on.*

First HTML Document

- <meta name="viewport" content="width=device-width">
  - *This viewport element ensures the page renders at the width of viewport, preventing mobile browsers from rendering pages wider than the viewport and then shrinking them down.*
- <title></title>
  - *This sets the title of your page, which is the title that appears in the browser tab the page is loaded in. It is also used to describe the page when you bookmark/favorite it.*
- <body></body>
  - *This contains all the content that you want to show to web users when they visit your page, whether that's text, images, videos, games, playable audio tracks, or whatever else.*

HTML Headings

- H1
- H2
- H3
- H4
- H5
- h6

Lists

- ul: unordered list
- ol: ordered list
- li:list item

New line

- br
- hr

Emphasis and importance

- em: emphasis (make text italic)
- strong: show important (make text bold)

But we have also presentation tags:

- b:bold
- i: italic
- u: underlined

Links

- a tag
- Attributes
  - *href*
    - Relative
      - *From root*
    - Absolute
    - Email links
  - *title*
  - *target="_blank"*

Images

```
<img
  src="images/dinosaur.jpg"
  alt="The head and torso of a dinosaur skeleton;
          it has a large head with long sharp teeth"
  width="400"
  height="341"
  title="A T-Rex on display in the Manchester University Museum" />
```

Global Attributes

- Class, Style
- Id
- Title
- Dir: ltr|rtl|auto
- Lang: en|fa|…
- Tabindex: number

Tables

- table
- tr
- td, th
- Colspan and rowspan attributes

Forms

- Form
  - *Action*
  - *method*
- Button
- Input
  - *Name*
  - *Value*
  - *Id*
  - *type*
    - text
    - password
    - button
    - submit