

# Web Programming

# **Web Servers & Protocols**

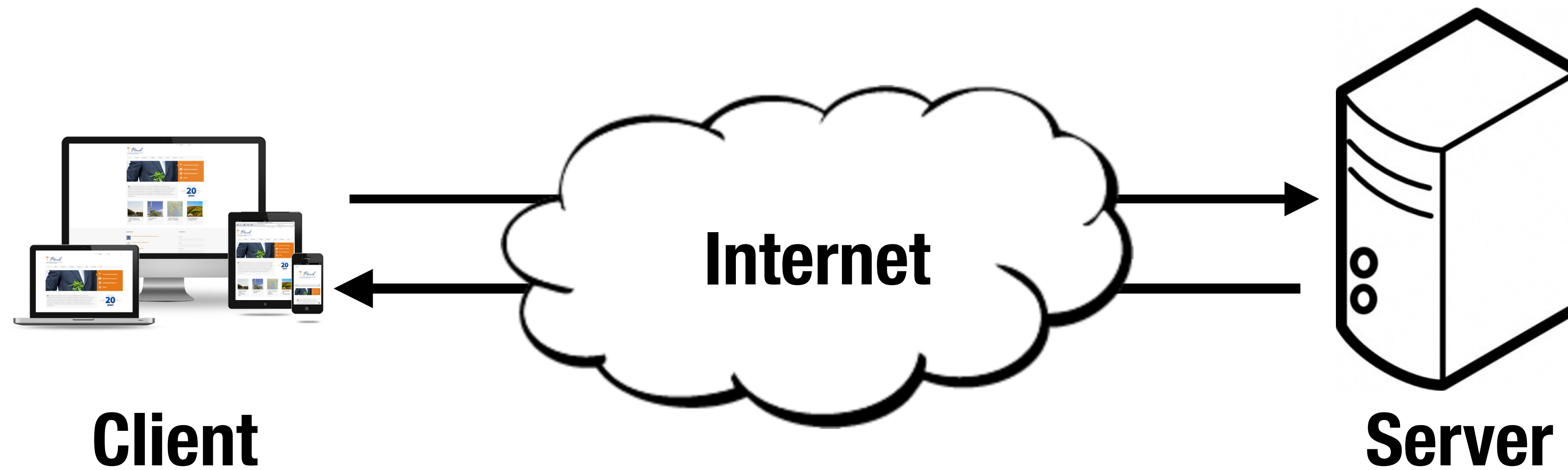
**Leander Jehl** | University of Stavanger

# Internet vs. Web

- Internet
  - Collection of computers and devices connected by equipment that allows them to communicate with each other
- World Wide Web
  - Collection of software and protocols
  - Most people use the Internet *through* the web

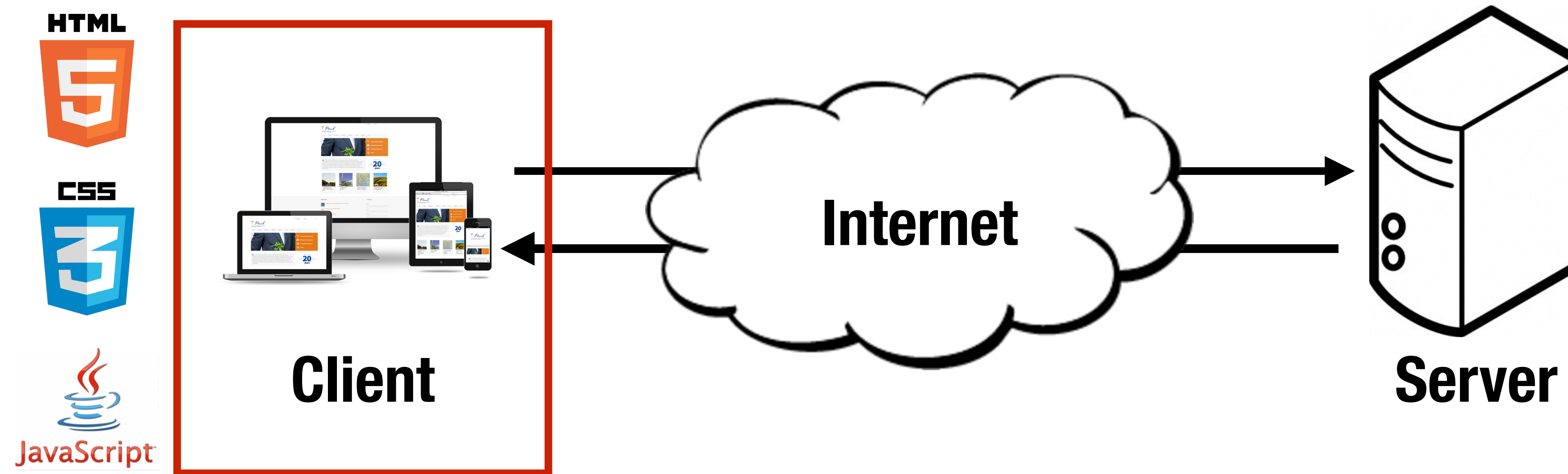
# Web

- Client-server architecture

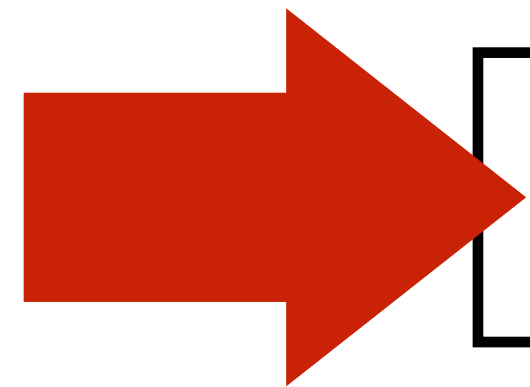


# So far in this course

- Only client-side



# Network layers



## Application

Dictates the method used to send data.  
E.g., HTTP, FTP, POP3, SMTP, SSL, ...

## Transport

Dictates the format of data sent, exactly where it is sent to, and maintaining data integrity.  
E.g., TCP, UDP

## Internet

Purely transports data across the network.  
E.g., IP, ICMP, IGMP

## Physical

The physical/logical network components used to interconnect network nodes.  
E.g., Ethernet, Wi-Fi, ...

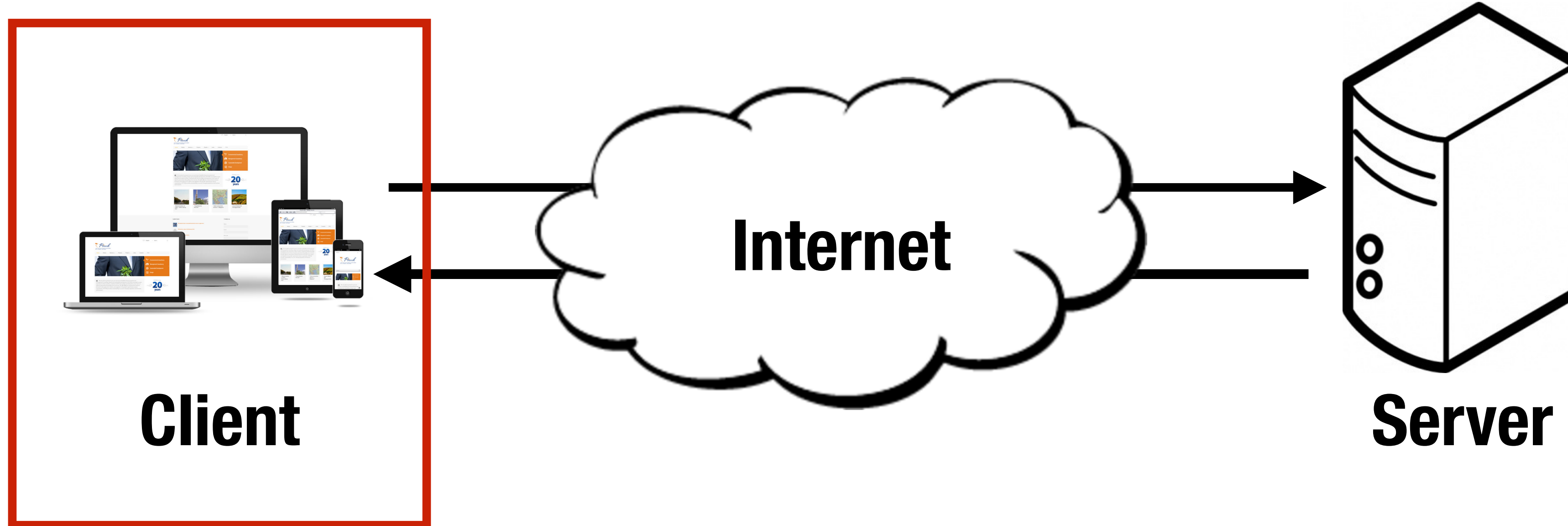
# IP addresses

- Internet Protocol address
- Unique numerical label assigned to each device that is connected to the Internet
  - IPv4: 32-bit number (4x 1byte)
    - **172.16.254.1**
  - IPv6: 128 bits (8 groups of 4-hex digits)
    - **2001:0db8:0a0b:12f0:0000:0000:0000:0001**

# Domain names

- Hostname: domain name assigned to a host computer
  - Combination of the host's local name with the parent's domain name
    - **www.stavanger.kommune.no**
- Translated to an IP address via the Domain Name System (DNS) resolver
  - or via a local hosts file

# Web clients





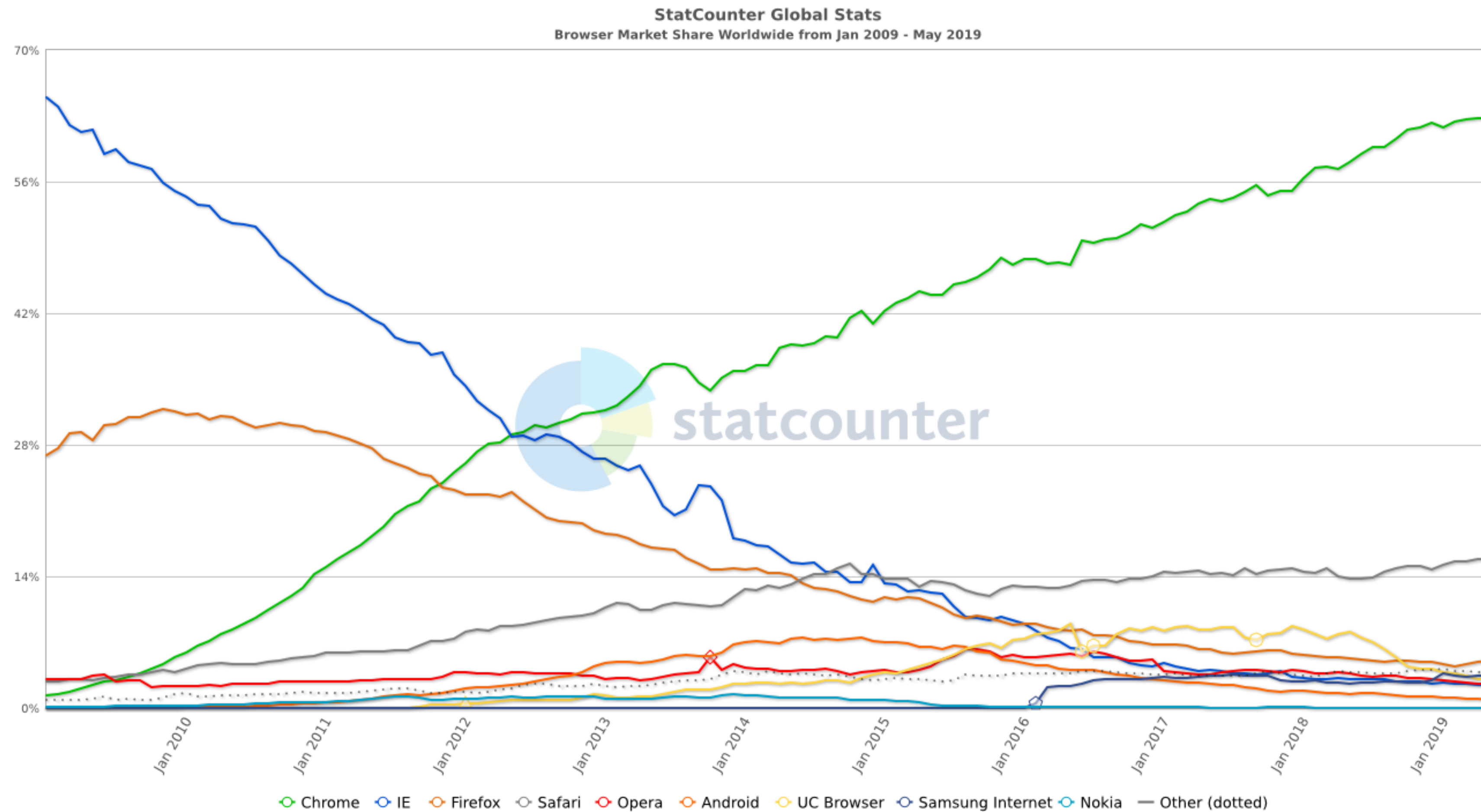
# Web browsers

- Programs running on client machines
- Initiates the communication by requesting a document (resource)
- Displays the returned document

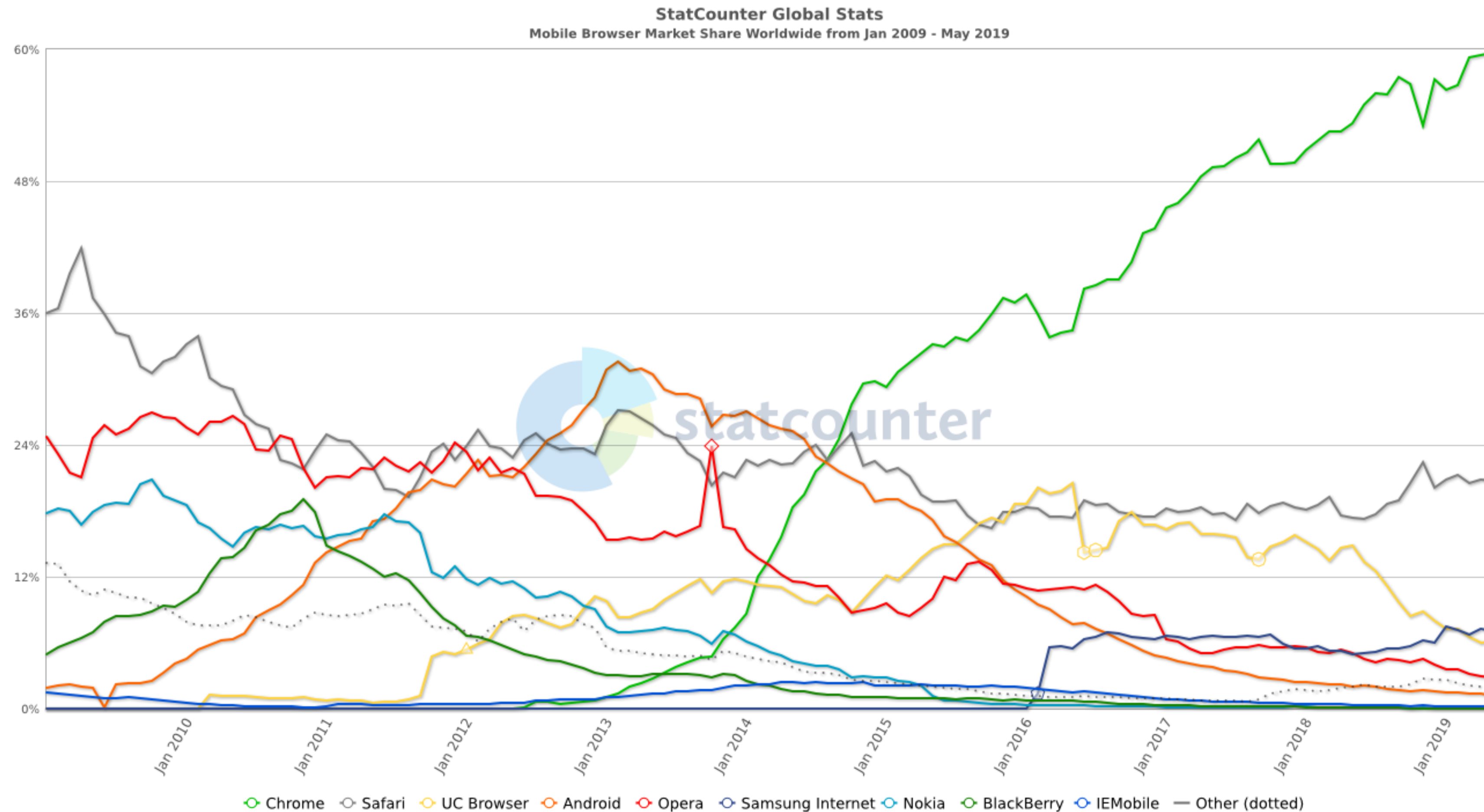
# Web browsers

- Programs running on client machines
- Initiates the communication by requesting a document (resource)
- Displays the returned document
- Javascript engine
- Media player
- Requests handling, networking, caching
- Local storage

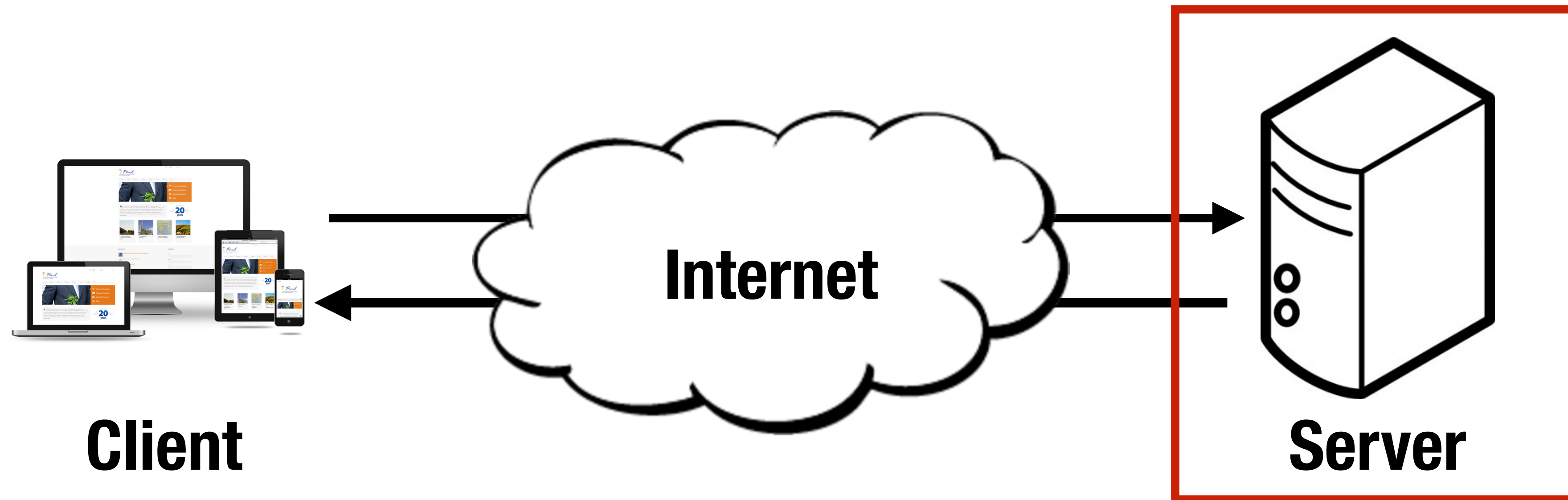
# Web browsers market share - desktop



# Web browsers market share - mobile



# Web servers



# Web servers

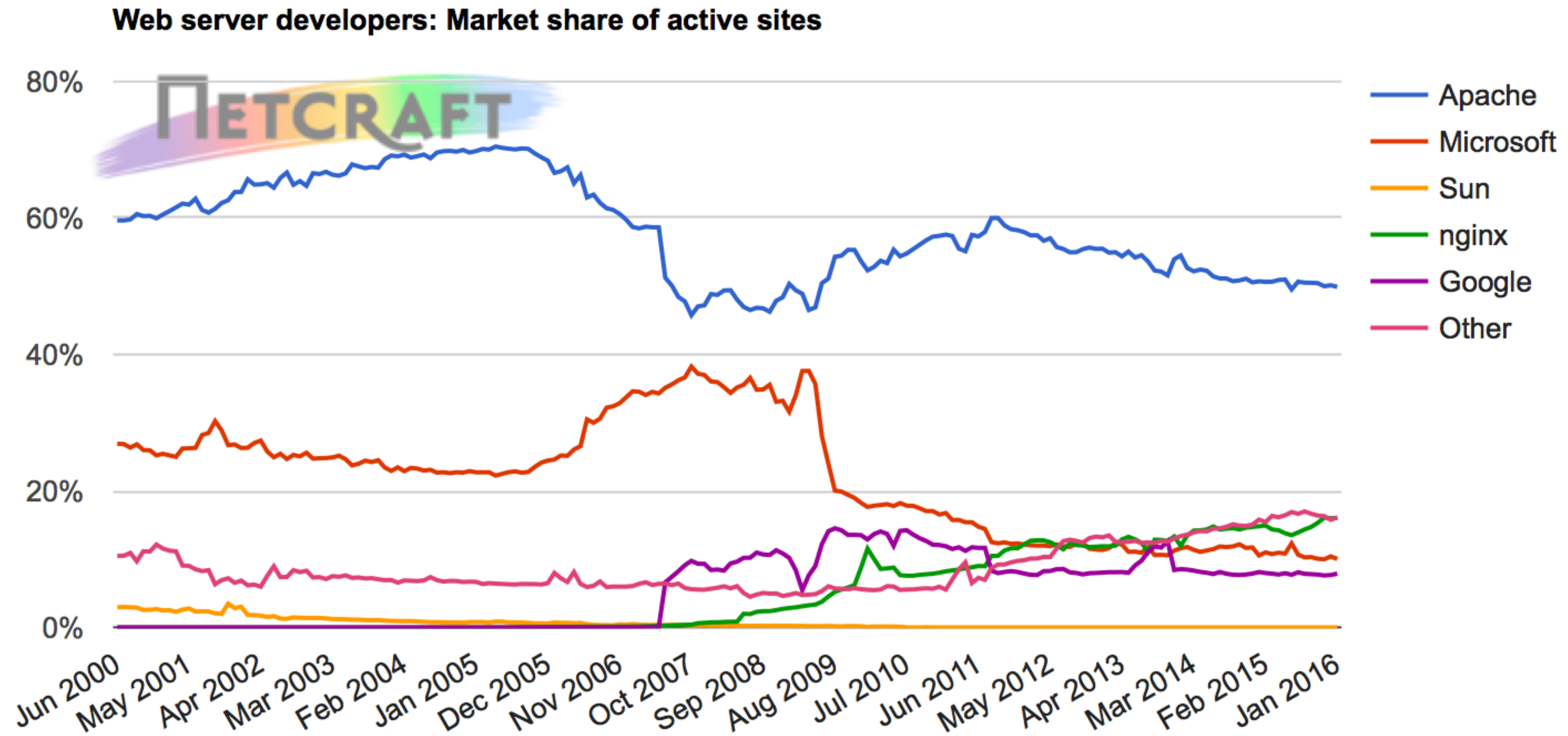
- Programs that provide documents (resources) upon client requests
  - Files stored in the document root
  - Interact with databases through server-side scripts
- **Many servers host more than a single site, called virtual hosting**
  - Requires a dedicated IP per domain name served
  - Alternatively, port-based virtual hosting (rarely used)

# Most popular web servers

- Apache
  - Fast, reliable, open source
  - One of the best available options for Unix-based systems, has also been ported to Windows
- IIS
  - Reasonably good, provides similar services to Apache
  - Supplied as part of Windows (but not turned on by default)
- nginx
  - For Unix-based systems (with proof-of-concept for Win)
  - Focus on high performance and low memory usage



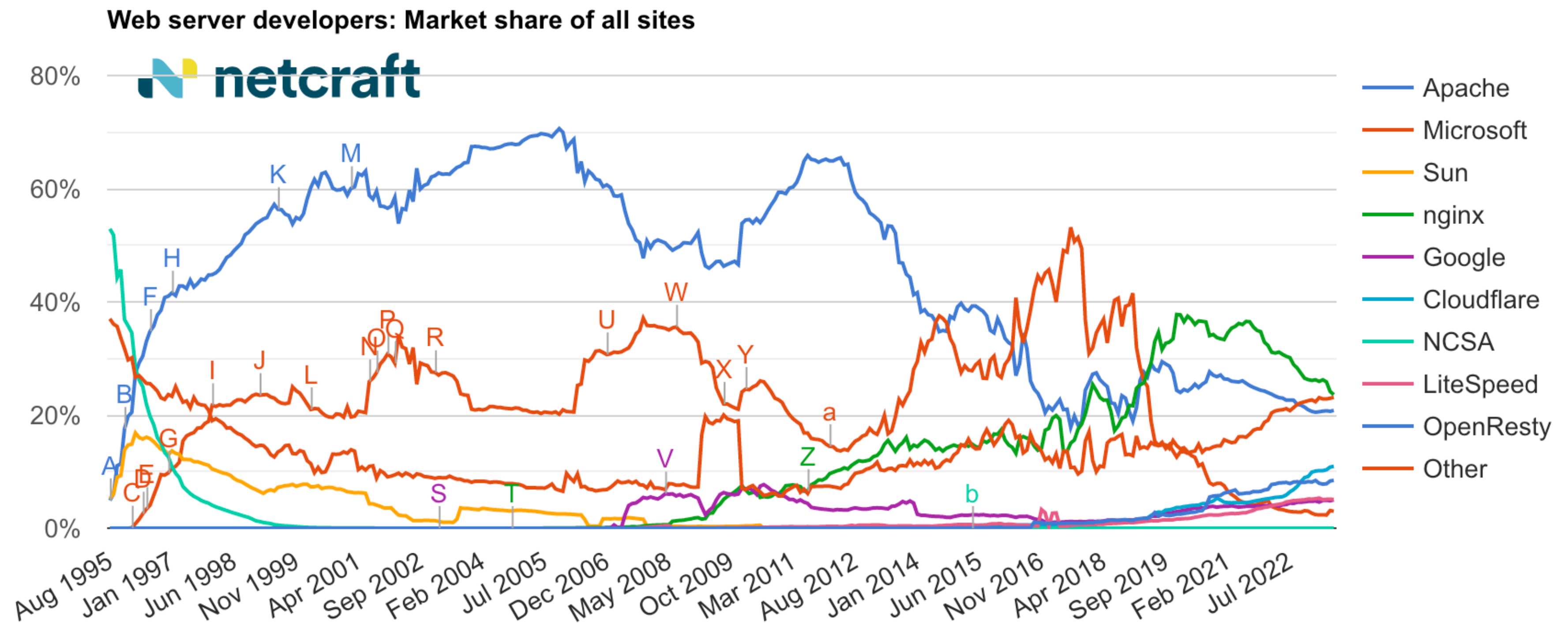
# Web server market share



source: <http://news.netcraft.com/archives/2016/01/26/january-2016-web-server-survey.html>



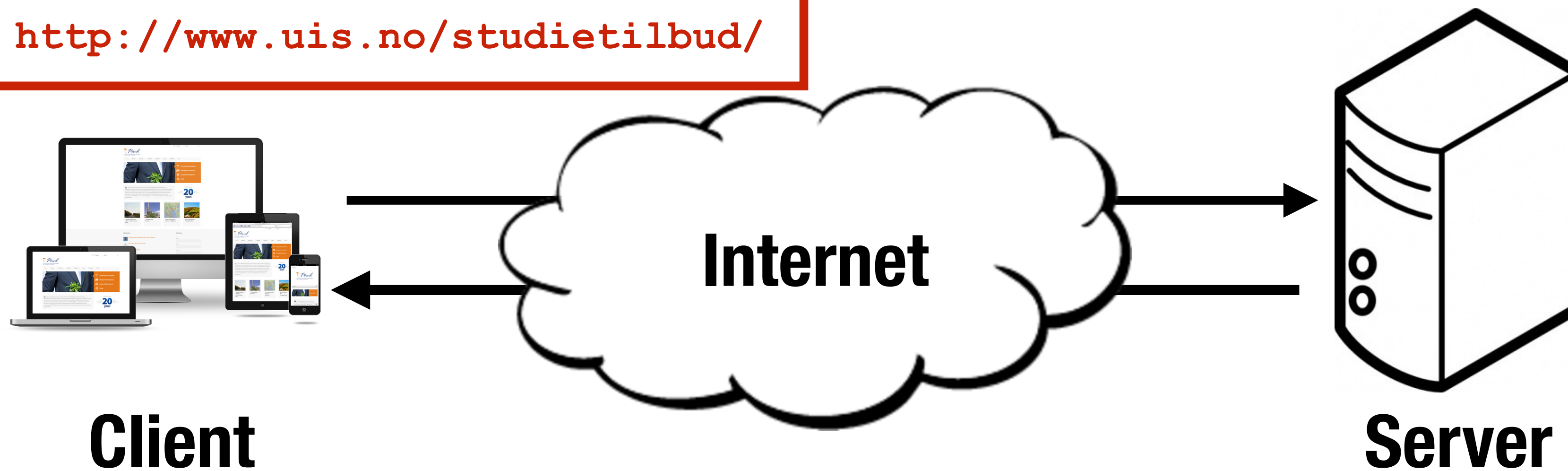
# Web server market share



source: <https://www.netcraft.com/blog/july-2023-web-server-survey/>

# URLs (Web addresses)

`http://www.uis.no/studietilbud/`



# URLs

- Uniform Resource Locator (URL)
- Provides reference to a resource
- That is, a **web address**
- Scheme:

**http : //www.uis.no / studietilbud /**

communication  
protocol

host (domain name  
or IP address)

full path to document  
(resource)

# Communication protocol

- Most commonly reference to web pages (http), but can also refer to
  - Documents on the client machine (file)  
`file:///Users/kbalog/work/teaching/DAT310/examples/css.html`
  - File transfer (ftp)  
`ftp://apache.uib.no/pub/apache/lucene/solr/4.10.0`
  - Database access (jdbc)  
`jdbc:mysql://localhost/mydatabase`

# Port

- HTTP default port is 80. Port only needs to be supplied if the web server is configured differently (e.g., on port 8000)
  - `http://domain.com:8000/some_document.html`

# URLs

- Url

protocol :// host / path

- With port

protocol :// host : port / path

- With user

protocol :// user @ host : port / path

- Url with query and fragment

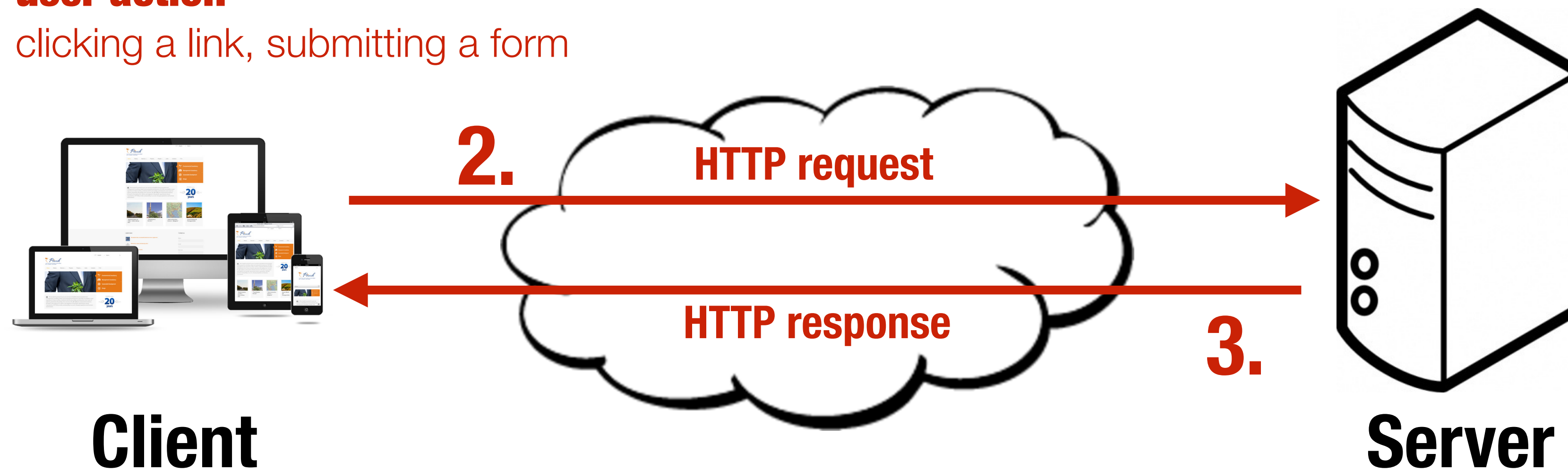
protocol :// host / path ? query # fragment

# URLs

- Can not contain spaces
- Can not contain special characters (; , & )
- URL encoding is used to escape these
  - % followed by a 2-digit hexadecimal ASCII code
  - E.g., space => %20, & => %26
  - Conversion functions are available
    - JavaScript: **encodeURIComponent ()** , **decodeURI ()**
    - PHP: **urlencode ()** , **urldecode ()**

# HTTP

- 1. user action**
  - clicking a link, submitting a form





# HTTP

- Hypertext Transfer Protocol
  - Current moving from version 2 to 3
- Default for all Web transactions
- Consist of two phases: request and response
- Each consist of two parts: header and body
  - Header: information about the communication
  - Body: data of the communication (if any)

# HTTP request

- Format

1. HTTP method, path to document, HTTP version
2. Header fields
3. Blank line
4. Message body

# HTTP request

1. HTTP method, path to document, HTTP version

```
GET /om-uis/kontakt/ HTTP/1.1
```

- Request methods
  - GET, HEAD, POST, PUT, DELETE
  - GET and POST are the most frequently used

# HTTP request methods

- GET
  - Returns the contents of the specified document
  - Request has no body part
  - Can be bookmarked
  - Has length restrictions
    - About 2000 characters in practice
  - Submitting forms using GET
    - Variables (name-values pairs are sent in the URL)
    - **`http://.../index.php?page=booking&step=1`**
  - Should never be used when dealing with sensitive data

# HTTP request methods

## - POST

- Executes the specified document, using the enclosed data
- Data is sent in the body of the request
- No restrictions on data length
- Cannot be bookmarked
- Submitting forms using POST
  - Variables are sent in the HTTP body of the request  
**page=booking&step=1**
  - (When form values are sent, the content-type is set to application/x-www-form-urlencoded)

# HTTP request

## 2. Header fields

- Host is required

```
Host: www.uis.no
```

- What type of document is accepted

```
Accept: text/html
```

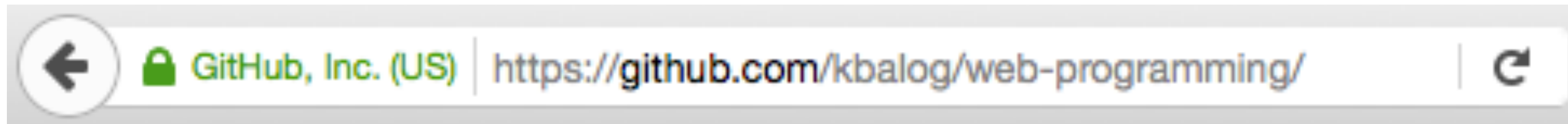
```
Accept: text/*
```

- If the request has a body, the length of the body in bytes is required

```
Content-length: 128
```

# HTTP requests from the browser

- New URL in address bar: GET request



- Link: GET request

```
<a href="http://github.com/web-programming">Link</a>
```

- Form submission: GET or POST request
  - Default is GET

```
<form action="somepage">...</form>
```

- POST:

```
<form action="somepage" method="POST">...</form>
```

# HTTP response

- Format

1. Status line
2. Header fields
3. Blank line
4. Response body



# HTTP response

```
HTTP/1.1 200 OK
Date: Mon, 15 Sep 2014 08:59:42 GMT
Server: Apache/2.2.23 mod_ssl/2.2.23 OpenSSL/0.9.8x PHP/5.4.14
Status: 200 OK
Last-Modified: Mon, 15 Sep 2014 08:59:45 GMT
Content-Language: no_NO, no
Content-Type: text/html; charset=utf-8
```

```
<!DOCTYPE html>
<html lang="no" class="no-js">
<head>
[...]
```

# HTTP response

HTTP/1.1 200 OK

Status line

Date: Mon, 15 Sep 2014 08:59:42 GMT

Server: Apache/2.2.23 mod\_ssl/2.2.23 OpenSSL/0.9.8x PHP/5.4.14

Status: 200 OK

Last-Modified: Mon, 15 Sep 2014 08:59:45 GMT

Content-Language: no\_NO, no

Content-Type: text/html; charset=utf-8

<!DOCTYPE html>

<html lang="no" class="no-js">

<head>

[...]

# HTTP response

Header fields

HTTP/1.1 200 OK

Date: Mon, 15 Sep 2014 08:59:42 GMT

Server: Apache/2.2.23 mod\_ssl/2.2.23 OpenSSL/0.9.8x PHP/5.4.14

Status: 200 OK

Last-Modified: Mon, 15 Sep 2014 08:59:45 GMT

Content-Language: no\_NO, no

Content-Type: text/html; charset=utf-8

<!DOCTYPE html>

<html lang="no" class="no-js">


<head>

[...]

# HTTP response

```
HTTP/1.1 200 OK
Date: Mon, 15 Sep 2014 08:59:42 GMT
Server: Apache/2.2.23 mod_ssl/2.2.23 OpenSSL/0.9.8x PHP/5.4.14
Status: 200 OK
Last-Modified: Mon, 15 Sep 2014 08:59:45 GMT
Content-Language: no_NO, no
Content-Type: text/html; charset=utf-8

```



```
<!DOCTYPE html>
<html lang="no" class="no-js">
<head>
[...]
```

# HTTP response

```
HTTP/1.1 200 OK
Date: Mon, 15 Sep 2014 08:59:42 GMT
Server: Apache/2.2.23 mod_ssl/2.2.23 OpenSSL/0.9.8x PHP/5.4.14
Status: 200 OK
Last-Modified: Mon, 15 Sep 2014 08:59:45 GMT
Content-Language: no_NO, no
Content-Type: text/html; charset=utf-8
```

```
<!DOCTYPE html>
<html lang="no" class="no-js">
<head>
[...]
```

Response body

# Essential header fields

- Status code
- Content-type

# HTTP status codes

```
HTTP/1.1 200 OK
Date: Mon, 15 Sep 2014 08:59:42 GMT
Server: Apache/2.2.23 mod_ssl/2.2.23 OpenSSL/0.9.8x PHP/5.4.14
Status: 200 OK
Last-Modified: Mon, 15 Sep 2014 08:59:45 GMT
Content-Language: no_NO, no
Content-Type: text/html; charset=utf-8

<!DOCTYPE html>
<html lang="no" class="no-js">
<head>
[...]
```

# HTTP status codes

- First digit
  - **1**: Informational
  - **2**: Success
  - **3**: Redirection
  - **4**: Client error
  - **5**: Server error



# HTTP status codes

**Steve Losh**  
@stevelosh



HTTP status ranges in a nutshell:

1xx: hold on  
2xx: here you go  
3xx: go away  
4xx: you fucked up  
5xx: I fucked up

 Reply  Retweet  Favorite  More

4,382  
RETWEETS

1,917  
FAVORITES



8:20 AM - 28 Aug 13

# Content-type

```
HTTP/1.1 200 OK
Date: Mon, 15 Sep 2014 08:59:42 GMT
Server: Apache/2.2.23 mod_ssl/2.2.23 OpenSSL/0.9.8x PHP/5.4.14
Status: 200 OK
Last-Modified: Mon, 15 Sep 2014 08:59:45 GMT
Content-Language: no NO, no
Content-Type: text/html charset=utf-8
```

```
<!DOCTYPE html>
<html lang="no" class="no-js">
<head>
[...]
```

# MIME

- Multipurpose Internet Mail Extensions
  - Originally developed to specify the format of documents sent via e-mail
- Determines the format of documents transmitted over the Web
  - Browser can choose the appropriate procedure to process the received content
- MIME specification format:
  - **type/subtype**

# MIME (2)

- A list of MIME specifications is stored in the Web server configuration file
  - Based on file extensions
- Browsers also maintain a conversion table
  - Only used when the server does not specify a MIME type
- Experimental subtypes are prefixed with x-
  - E.g., **application/x-gzip**

# Example MIME types

- A list of MIME specifications is stored in the Web server configuration file
  - Based on file extensions

<code>.css</code>	<code>text/css</code>
<code>.doc</code>	<code>application/msword</code>
<code>.gif</code>	<code>image/gif</code>
<code>.html</code>	<code>text/html</code>
<code>.js</code>	<code>application/x-javascript</code>
<code>.txt</code>	<code>text/plain</code>
<code>.qt</code>	<code>video/quicktime</code>
<code>.mp3</code>	<code>audio/mpeg</code>

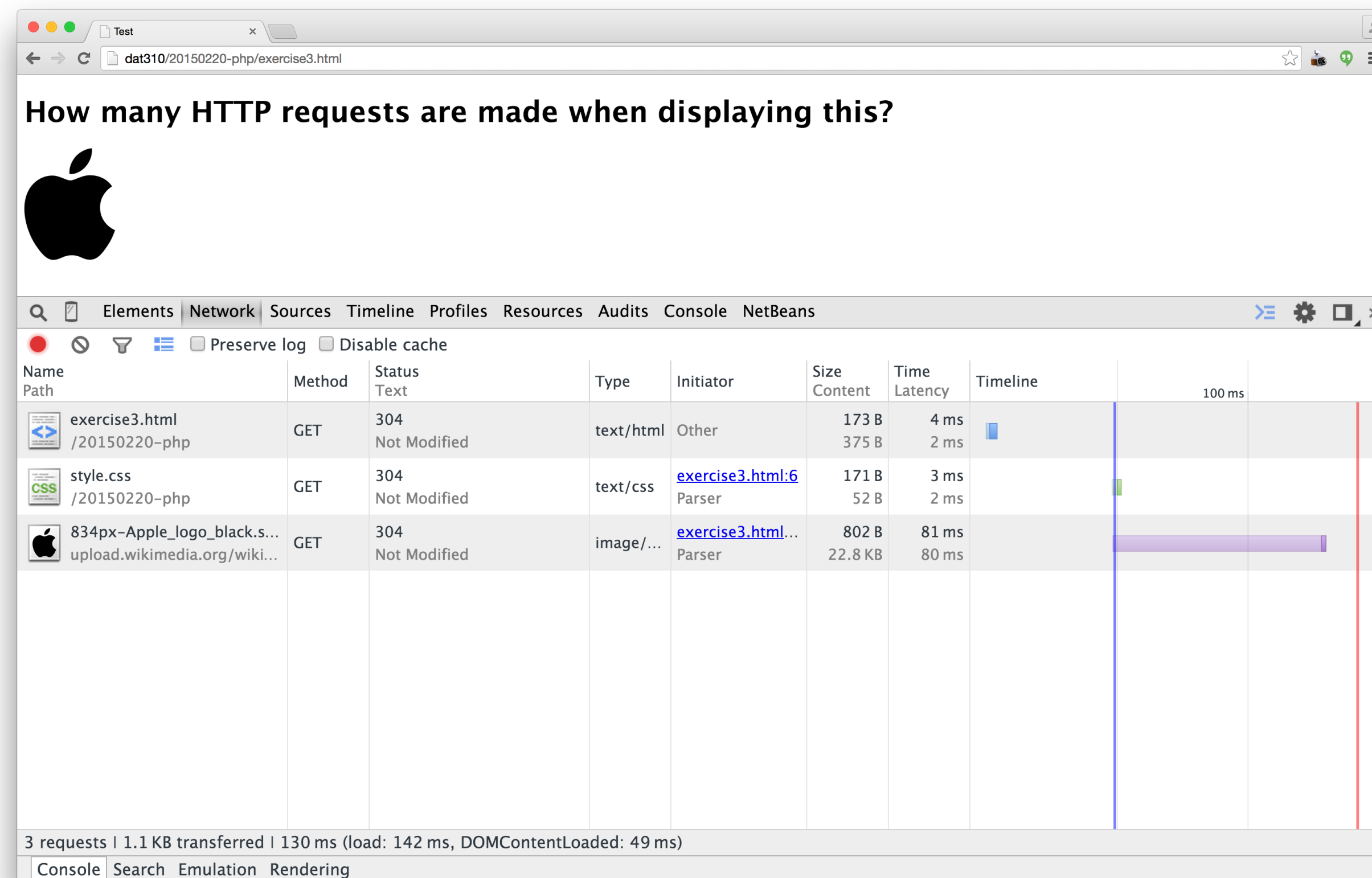
# Exercise

- How many HTTP requests are made by the browser when displaying this page?

```
<!DOCTYPE html>
<html>
<head lang="en">
  <meta charset="UTF-8">
  <title>Test</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <h1>How many HTTP requests are made when displaying this?</h1>
  
</body>
</html>
```

# Tracking HTTP requests (client-side)

- Network tab under Web developer tools



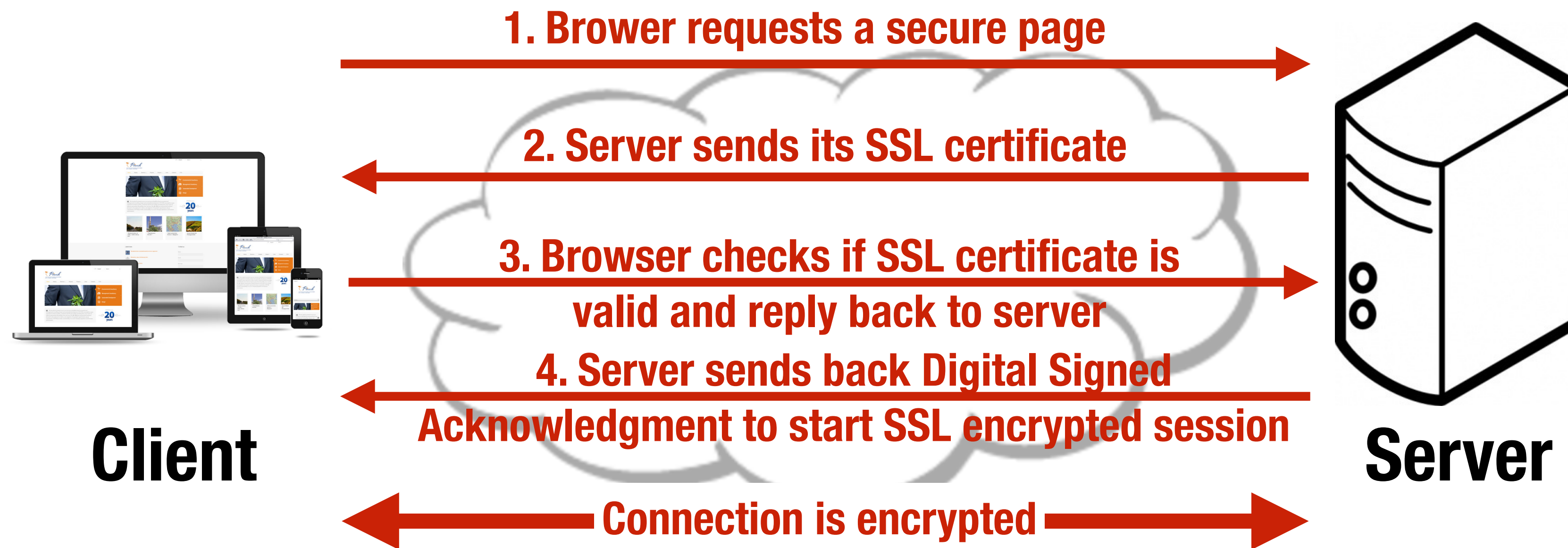
**HTTPS**



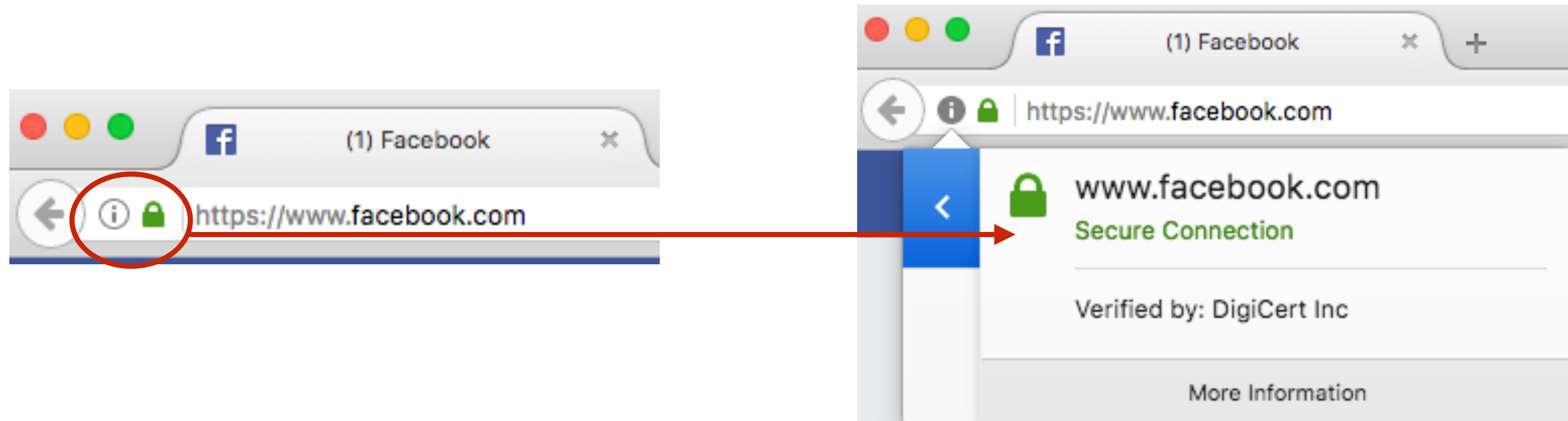
# HTTPS

- Hyper Text Transfer Protocol **Secure** (https://)
  - Uses port 443 instead of the default port 80
- Data is transmitted securely on internet with the help of SSL/TLS
  - **Secure Socket Layer** (SSL) encrypts data between client and server using 128/256 bit key encryption
  - SSL/TSL also operates on the *Application* network layer
- Site requires an SSL certificate
  - Has to be issued by a recognized Certificate Authority (CA)
    - In cryptographic terms, the CA acts as *trusted party*
  - Consists of a public and private key

# How HTTPS works



# Find out info about the SSL certificate



# Browsers alert if they receive an invalid certificate



## Secure Connection Failed

svn.boost.org uses an invalid security certificate.

The certificate is not trusted because the issuer certificate is unknown.

(Error code: sec\_error\_unknown\_issuer)

- This could be a problem with the server's configuration, or it could be someone trying to impersonate the server.
- If you have connected to this server successfully in the past, the error may be temporary, and you can try again later.

[Or you can add an exception...](#)

# **Making HTTP requests in Python**

# Overview



# Using the requests package

📄 examples/python/http/request1.py

```
import requests

r = requests.get("http://wiki.ux.uis.no/")

print(r.status_code)
print(r.headers)
print(r.content)
```

- Documentation: [docs.python-requests.org](https://docs.python-requests.org)

# Using the requests package

🔗 [examples/python/http/request2.py](https://github.com/psf/requests/blob/master/examples/python/http/request2.py)

```
import urllib.request

u = urllib.request.urlopen("http://wiki.ux.uis.no/")

# u can be read as a file

# Read the entire page
print (u.read())

# Alternatively: read it line-by-line
for line in u:
    print(line)
```

- Doc: <https://docs.python.org/3.5/library/urllib.request.html>

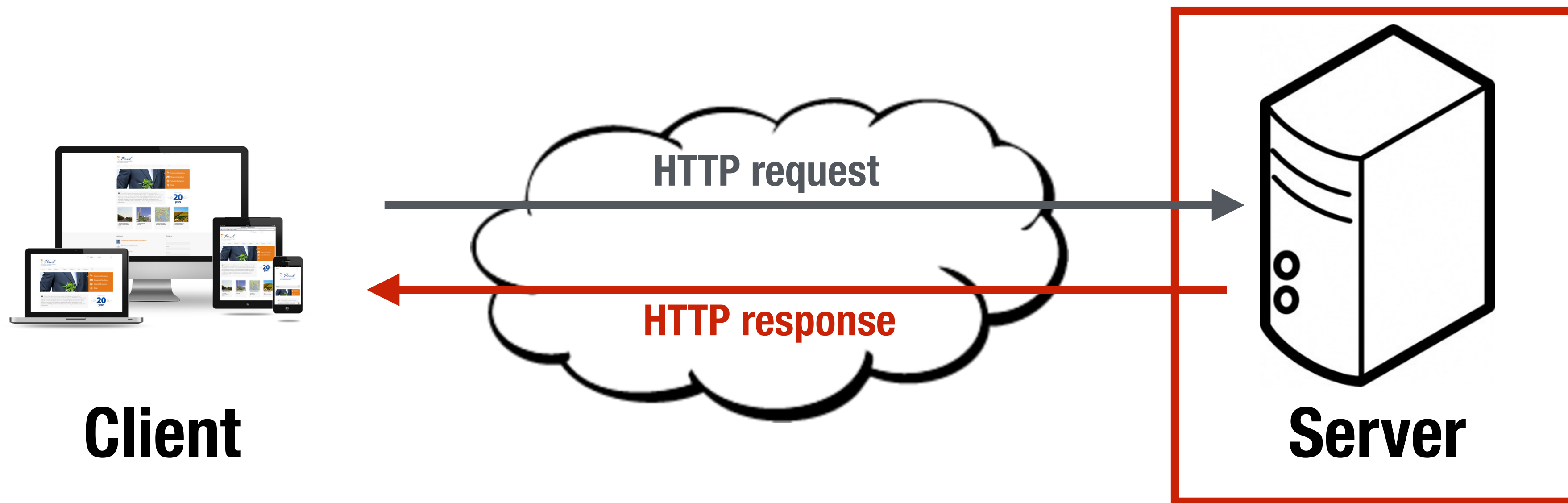


# Web scraping

- Web scraping is used for extracting data from websites using HTTP (or a web browser)
  - First fetching a webpage
  - Then, extracting data from it (parsing, searching, etc.)
  - Data is typically stored in a local database
- Frequent usages include web indexing, data mining, product price or review comparison, reputation management, etc.
- Many other Python libraries for scraping, see, e.g.,
  - <https://elitedatascience.com/python-web-scraping-libraries>

# **Simple HTTP server in Python**

# Overview



# Simple HTTP server

📄 examples/python/http/server.py

```
from http.server import BaseHTTPRequestHandler, HTTPServer

class myHTTPServer_RequestHandler(BaseHTTPRequestHandler):

    def do_GET(self):
        # Send response status code
        self.send_response(200)

        # Send headers
        self.send_header('Content-type', 'text/html')
        self.end_headers()

        message = "Hello world!"
        # Write message content as utf-8 data
        self.wfile.write(bytes(message, "utf8"))
        return

def main():
    server_address = ('127.0.0.1', 8080)
    httpd = HTTPServer(server_address, myHTTPServer_RequestHandler)
    print("running server...")
    httpd.serve_forever()
```

# Sending requests using cURL

- cURL is a library and command line tool for transferring data using various protocols
- We use the command line curl tool to make HTTP requests
- GET request

```
curl http://localhost:8080
```

- POST request

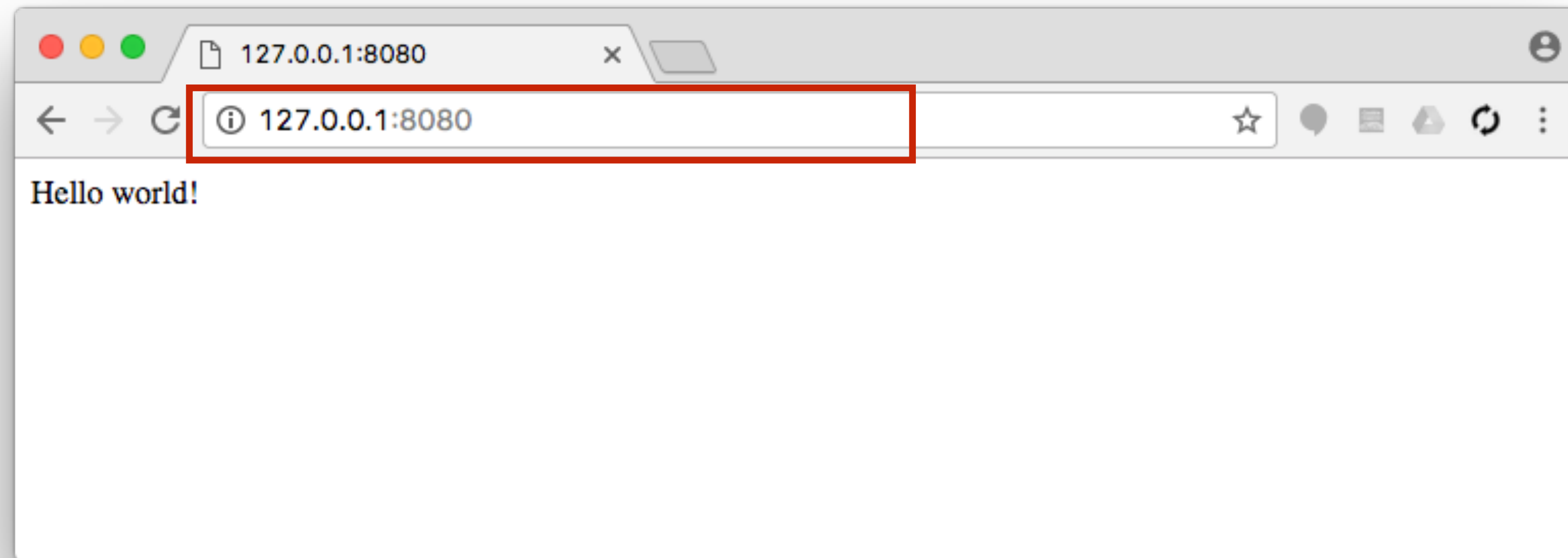
- without data

```
curl -X POST http://localhost:8080
```

- with data

```
curl --data "param1=value1&param2=value2" http://localhost:8080
```

# Sending requests from a browser



# Exercises



[github.com/dat310-2024/info/tree/master/](https://github.com/dat310-2024/info/tree/master/exercises/python/http)  
**exercises/python/http**