

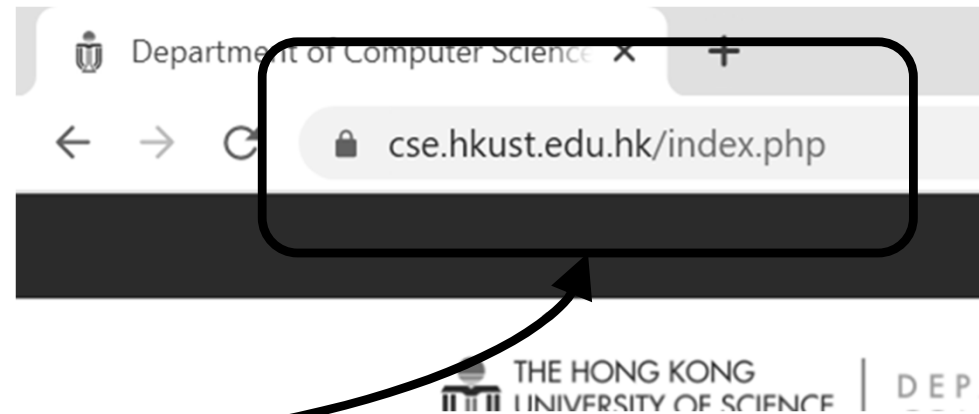
COMP4021
Internet Computing

Basic HTTP Process

Gibson Lam and David Rossiter

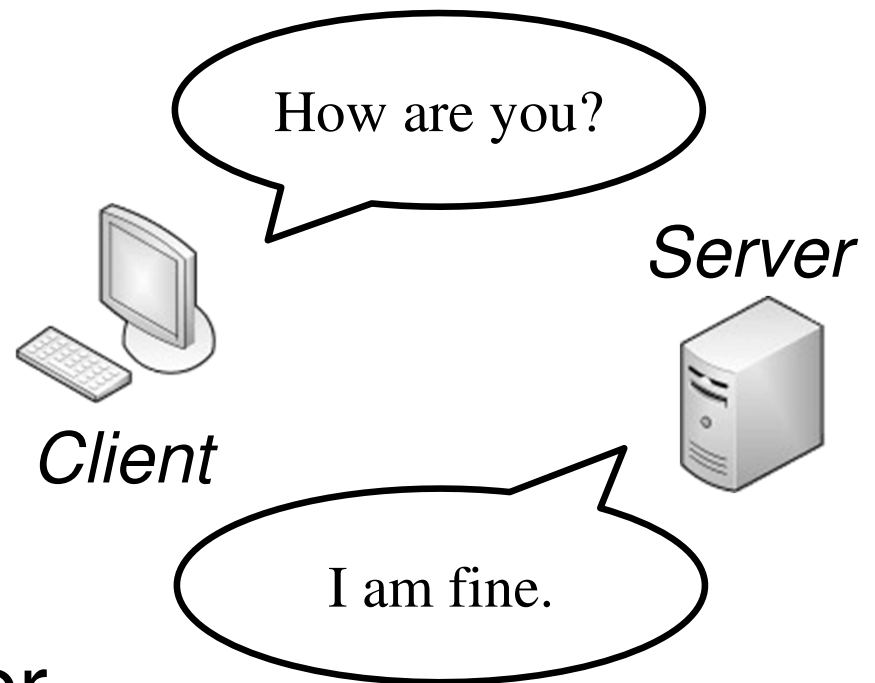
The HTTP Process

- When you want to go to a web page on a browser, you simply type the web address, i.e. the URL, and the browser will load the page for you
- In this presentation, we will look at what happens between your browser, also called the client, and the server
- We call this the *HTTP process*



The HTTP Protocol

- The HTTP protocol, or just HTTP, stands for **H**ypertext **T**ransfer **P**rotocol
- It describes how the client and the server should talk to each other
- The protocol uses plain text that means you can read and understand the content of the communication




Getting a Web Page

- To get a web page:
 1. You enter the URL into the browser
 2. The browser connects to the server
 3. The browser sends an HTTP request to the server
 4. The server returns an HTTP response
 5. The browser processes the response and then may make additional requests to the server

1. Entering the URL

- You need a URL to get a web page
- It has the following format:

`https://cse.hkust.edu.hk/index.html`


Protocol *Hostname* *Path and File*

- In the above example, it uses HTTPS, which is the secure version of HTTP
- A HTTP address starts with `http://`, while HTTPS starts with `https://`

HTTP Vs HTTPS

- HTTPS is **H**ypertext **T**ransfer **P**rotocol **S**ecure, i.e. HTTP secure
- HTTPS and HTTP have the same functions and work in a similar way
- The difference is that HTTPS is encrypted so that others cannot read its content
- Most websites nowadays uses HTTPS for its secure communication

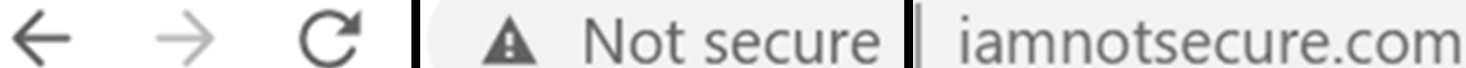
The Protocol on Chrome

- Chrome typically hides the protocol text (http:// or https://) from the URL:



A screenshot of a Chrome browser's address bar. It features navigation icons (back, forward, refresh) on the left. To the right of these icons is a small padlock icon, followed by the text 'cse.hkust.edu.hk/index.php'. A black arrow points from the 'lock' icon in the list below to this padlock icon in the address bar.

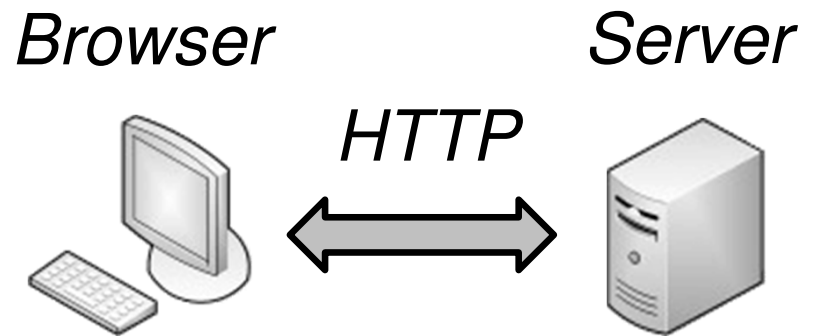
- But you know which one it is by the 'lock' shown on the left of the URL
- If HTTP is used, it will show this:



A screenshot of a Chrome browser's address bar. It features navigation icons (back, forward, refresh) on the left. To the right of these icons is a warning icon (triangle with an exclamation mark) followed by the text 'Not secure | iamnotsecure.com'. A black rounded rectangle highlights the 'Not secure' text.

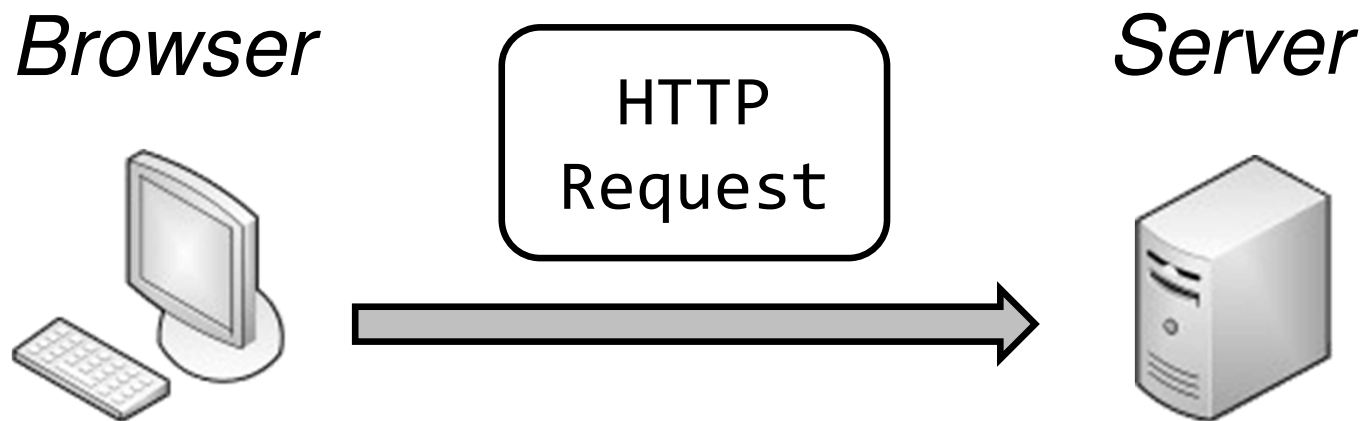
2. Connecting to the Server

- To get a web page, the browser first connects to the server
- Basic HTTP usually connects through port 80 on the server (the 'door number' on the server), while HTTPS uses port 443
- As you will see later, you can use any port number, such as 3000 or 8000, if you want to



3. Sending an HTTP request

- After connecting to the server, the browser then sends an *HTTP request* to it

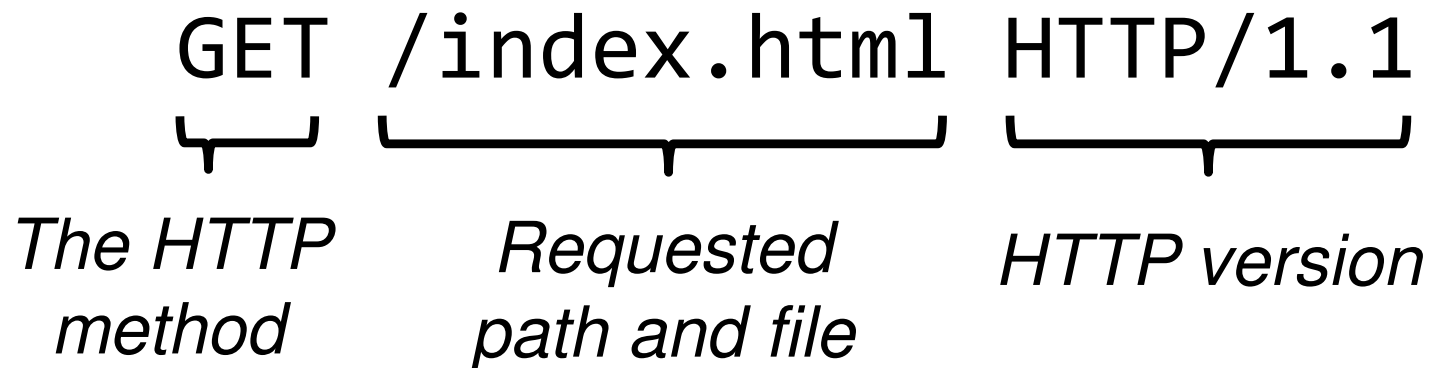


- HTTP requests are text-based so you can easily read and understand their content

The HTTP request

- Here is an example of the first line of an HTTP request :

`GET /index.html HTTP/1.1`



The diagram shows the components of the HTTP request line `GET /index.html HTTP/1.1`. Brackets are placed under each part: a bracket under `GET`, a bracket under `/index.html`, and a bracket under `HTTP/1.1`. Below each bracket is a label: *The HTTP method* under `GET`, *Requested path and file* under `/index.html`, and *HTTP version* under `HTTP/1.1`.

The HTTP method *Requested path and file* *HTTP version*

- In addition to the above text, a request can have optional 'headers', which contain useful information for the server

Request Methods

- GET, for getting a requested file, is one of the request methods
- Here are two other commonly used ones:
 - POST sending some data to the server in addition to asking for the requested file
 - HEAD asking for the information of the requested file only, but **not** getting the actual file

A Complete HTTP Request

- Here is a HTTP request sent by Chrome, which includes many HTTP headers:

GET / HTTP/1.1

HTTP headers {
Host: cse.hkust.edu.hk
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 ...
Accept: text/html, ...
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US
...*More headers not shown*...

The Root File

- In the previous example, the request asks for the root file:

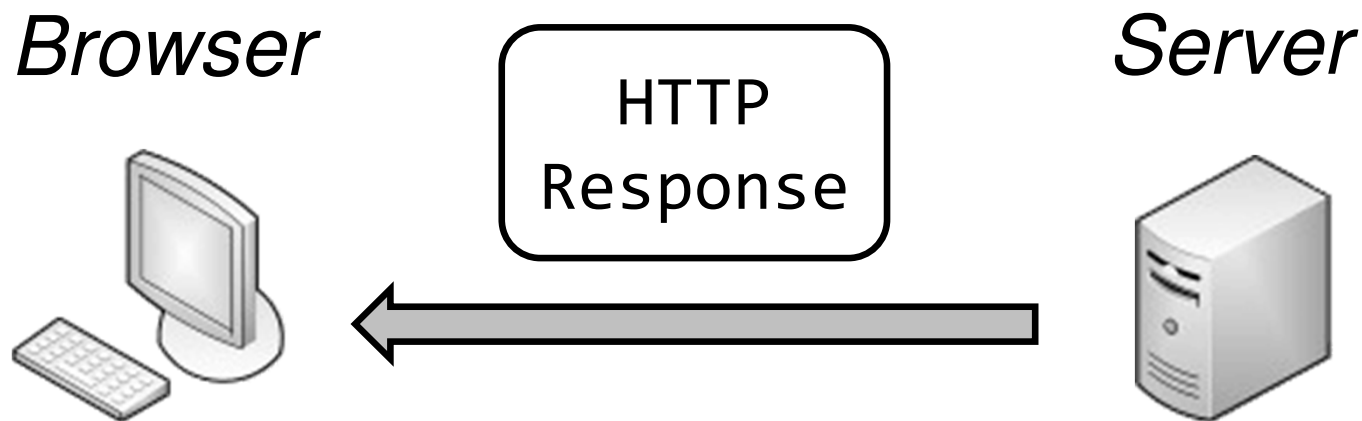
GET / HTTP/1.1
└─┘

The request has a root path without a filename so it is asking for the root file

- The file to be returned is determined by the server, which is typically index.html

4. Returning an HTTP Response

- After getting the request, the server returns an HTTP response to the browser



- The HTTP response contains some text-based information but it may also contain binary content from the requested file

The HTTP Response

- Here is an example of the first line of an HTTP response:

HTTP/1.1 200 OK

HTTP version Response code

- Similar to HTTP requests, there are optional headers returned by the server
- The requested file is then sent after a blank line, as shown in the next slide

A Complete HTTP Response

- Here is an example response with headers:

HTTP/1.1 200 OK

Date: Sat, 12 Mar 2022 16:32:17 GMT

Server: Apache

Cache-Control: private

Keep-Alive: timeout=5, max=100

Connection: Keep-Alive

Transfer-Encoding: chunked

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

HTTP headers

*Blank
line*

<!DOCTYPE ...

Content of the requested file

The Response Codes

- There are many HTTP response codes, also called the status codes
- Here are some common response codes:
 - 200 OK
 - 400 Bad Request
 - 404 Not Found
 - 500 Internal Server Error
- The previous example is an 'OK' response, i.e. the request is a successful one

Content Types

- When a server returns a file, it tells you what the type of file, using the 'content type' header
- The header contains a MIME type and some other information, for example:

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

*The server tells you the
returned file is an HTML file*

More MIME Types

- A MIME type indicates the content type of a file or some data
- Here are some common MIME types:

text/plain

Plain text

image/png

PNG image

image/svg+xml

SVG image

text/javascript

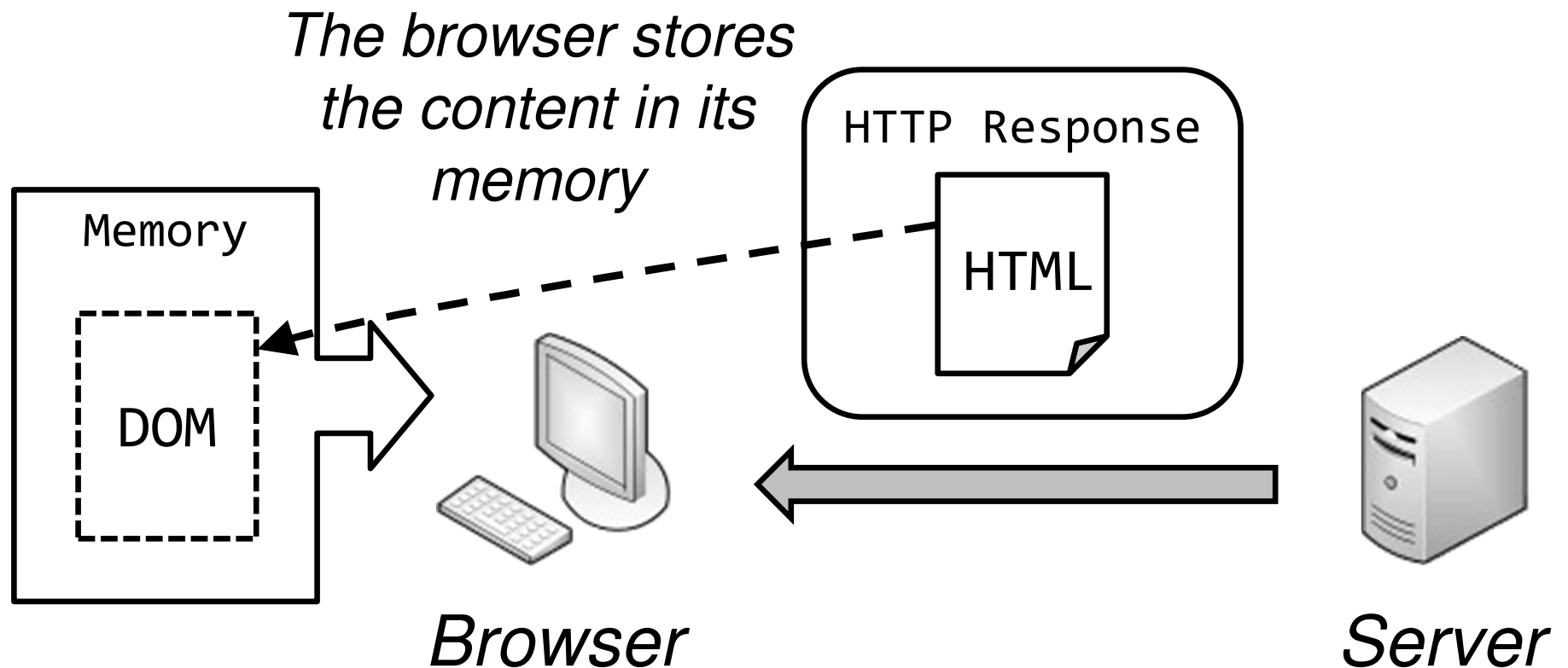
JavaScript file

application/json

JSON data

5. Processing the Response

- If the returned file is an HTML file, the browser will convert its content into a memory structure, i.e. the DOM



Linking to Other Files

- A web page may contain links to many files, for example:
 - The page uses CSS files, i.e. `<link ...>`
 - The page contains images, i.e. ``
 - The page uses JavaScript files, i.e. `<script ...></script>`and so on
- To get all these files, the browser needs to make more HTTP requests to the server

Examples of Linked Files

- Here is the HTML file returned by the CSE website
- You can see the browser needs to make many more requests for these files!

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <meta content="IE=edge" http-equiv="X-UA-C
  <meta name="viewport" content="width=device
  <title>Department of Computer Science and
  <link rel="canonical" href="https://cse.hk
  <link rel="icon" type="image/png" href="/f
  <link rel="stylesheet" href="https://fonts
  <link rel="stylesheet" href="/css/app.css?
  <script src="https://oss.maxcdn.com/html5s
  <script src="https://oss.maxcdn.com/respon
  <script src="/scripts/jquery/jquery-1.12.4
  <script src="/scripts/script.js"></script>
  <script src="/scripts/onload.js"></script>
  <link rel="stylesheet" media="all" href="/
  <link rel="stylesheet" media="all" href="/
```

...

Making Another Request

- For example, if the browser sees this link:

```
<script src="/scripts/script.js"></script>
```

it will send another request to the server,
i.e.:

```
GET /scripts/script.js HTTP/1.1  
...
```

- The result of the request is shown on the next slide

Response of the Script File

- Here is the response from the server:

Date: Sat, 12 Mar 2022 17:50:42 GMT

Server: Apache

Last-Modified: Mon, 16 Jul 2012 03:00:00 GMT

ETag: "2034f228-29f-4c4e9a0244c00"

Accept-Ranges: bytes


Content-Length: 671

Keep-Alive: timeout=5, max=100

Connection: Keep-Alive

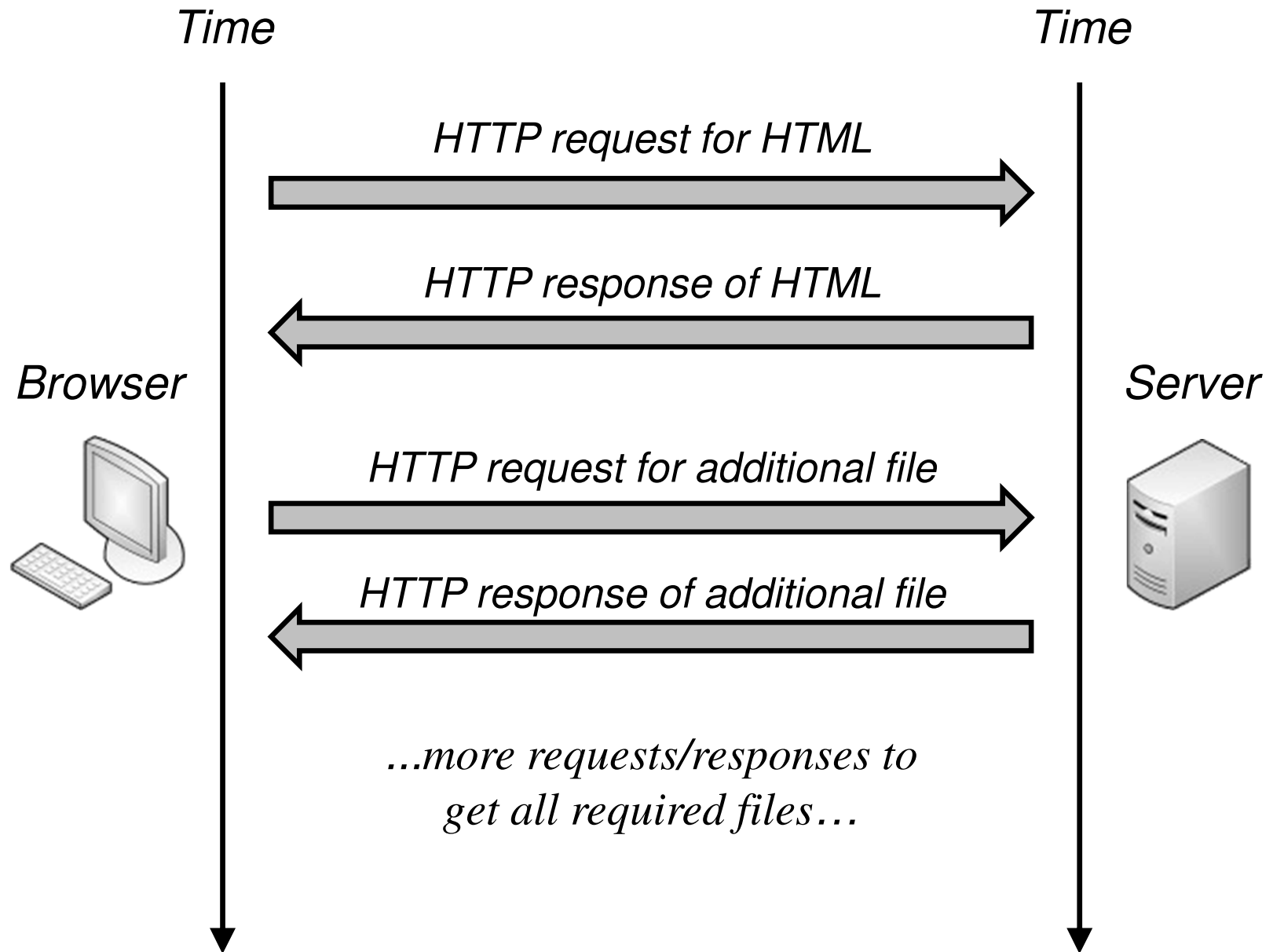
Content-Type: application/x-javascript

*This time the
content is not
an HTML file*



```
function msend(user,place) { ...
```


Summary



Getting the Headers in Chrome

- You can see the HTTP headers in Chrome in the *Network* tab from its DevTools

