Web Security Basics

Web Security Basics

1. Web Architecture
2. Web Server
3. HTTP Protocol
4. Cookies

Web Security Goals

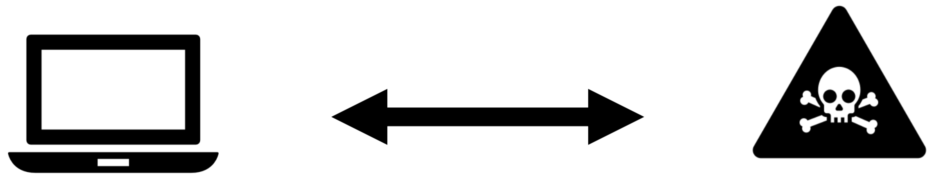
Safely browse the web in the face of attackers

Visit a web sites (including malicious ones!) without incurring harm

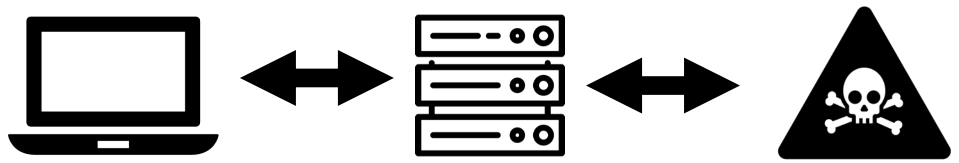
1. **Site A** cannot steal data from your device, install malware, access camera, etc.
2. **Site A** cannot affect session on **Site B** or eavesdrop on **Site B**

Attack Models

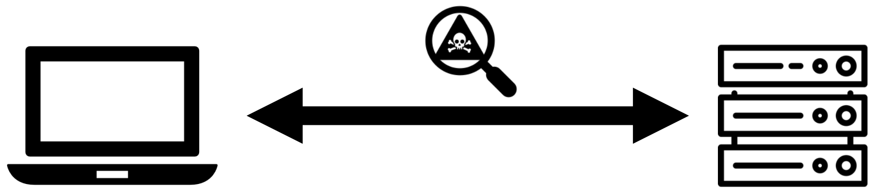
Malicious Website



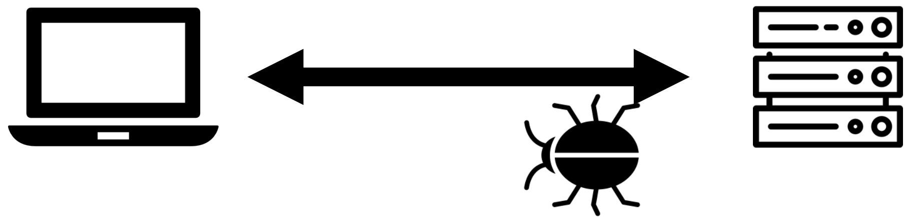
Malicious External Resource



Network Attacker



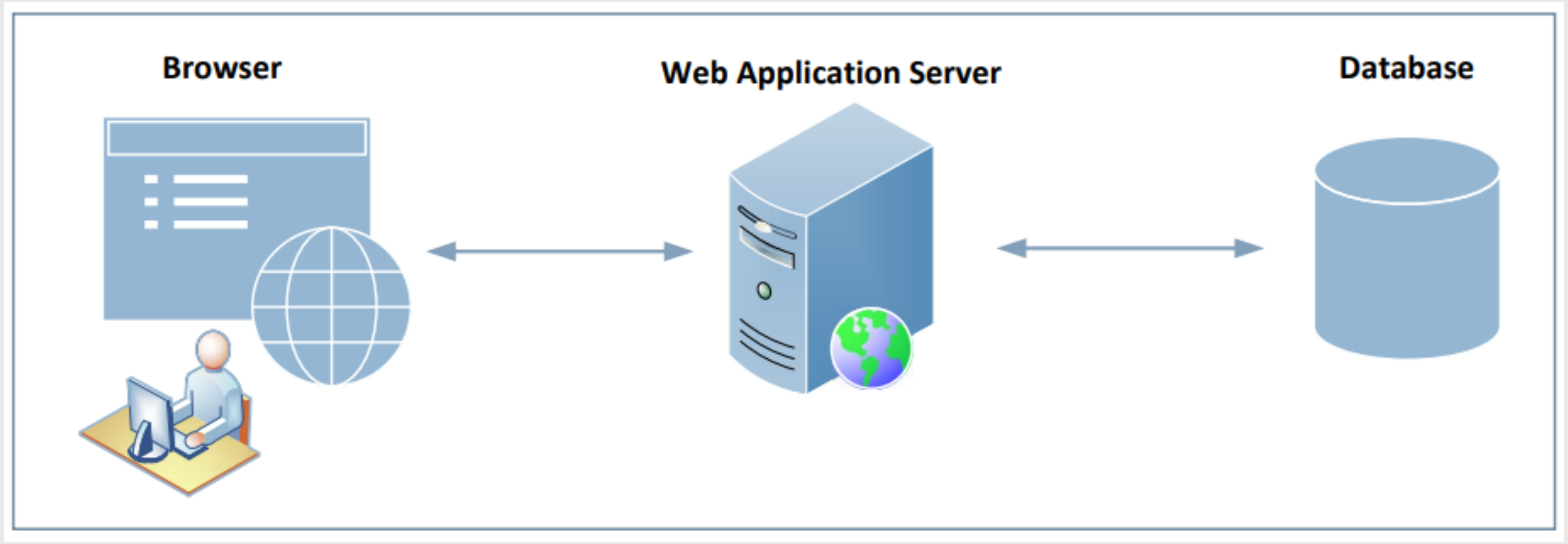
Malware Attacker



Web Security Basics

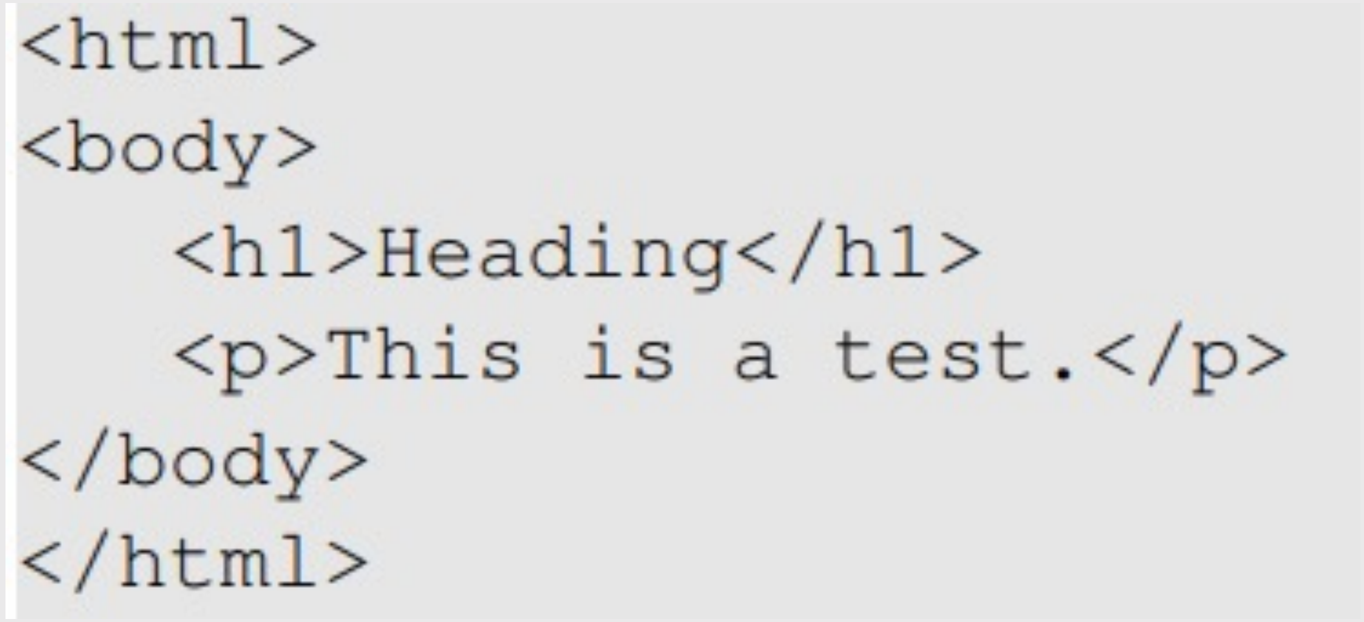
1. Web Architecture
2. Web Server
3. HTTP Protocol

Web Architecture



HTML

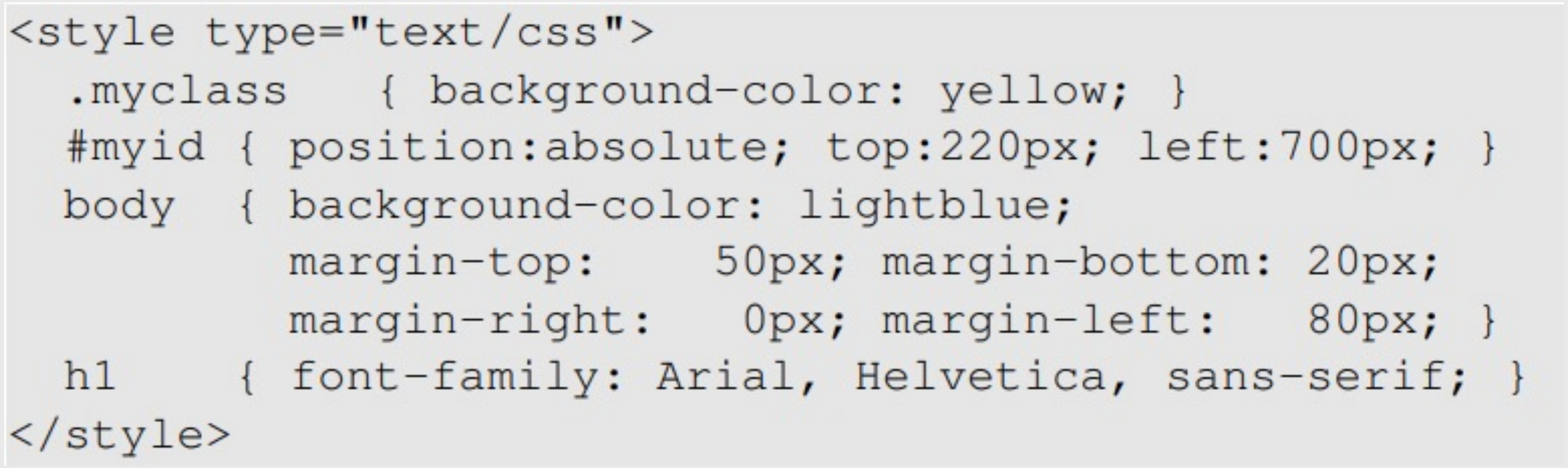
- Hypertext Markup Language
- For creating web pages
- Example



```
<html>
<body>
   <h1>Heading</h1>
   <p>This is a test.</p>
</body>
</html>
```

CSS: Cascading Style Sheets

- Specify the presentation style
- Separate content from the presentation style
- Example



```
<style type="text/css">
  .myclass   { background-color: yellow; }
  #myid { position:absolute; top:220px; left:700px; }
  body  { background-color: lightblue;
          margin-top:    50px; margin-bottom: 20px;
          margin-right:   0px; margin-left:   80px; }
  h1    { font-family: Arial, Helvetica, sans-serif; }
</style>
```

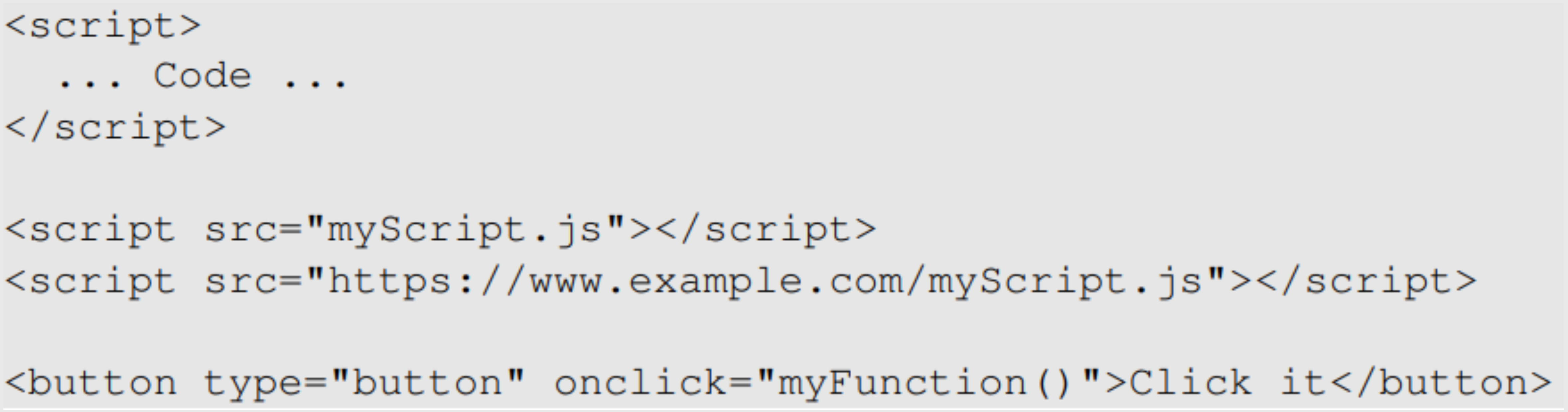
Dynamic Content

- Angular
- React
- Vue.js
- **JavaScript**

- AJAX (Asynchronous JavaScript and XML)

JavaScript

- Also known as ECMAScript
- Scripting language for web pages
- Different ways to include JavaScript code

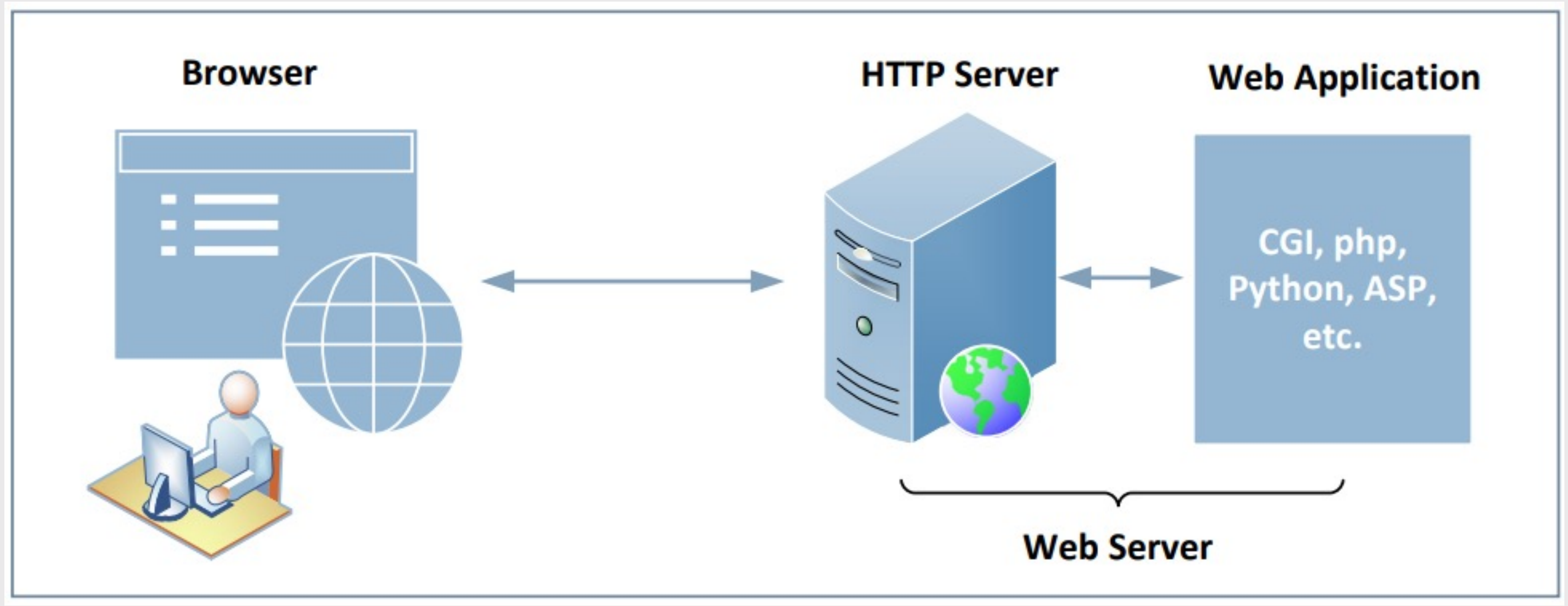


```
<script>
  ... Code ...
</script>

<script src="myScript.js"></script>
<script src="https://www.example.com/myScript.js"></script>

<button type="button" onclick="myFunction()">Click it</button>
```

Web Security Basics

1. Web Architecture
2. Web Server
3. HTTP Protocol

HTTP Server & Web Application Server



Web Security Basics

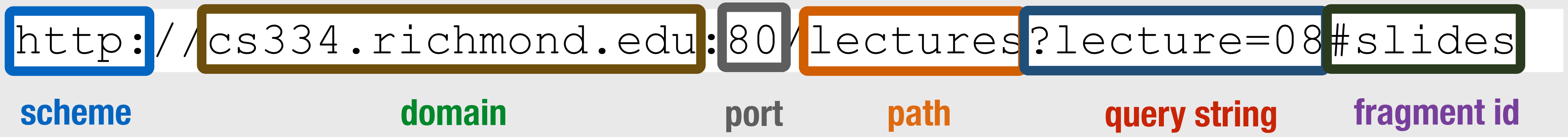
1. Web Architecture
2. Web Server
3. HTTP Protocol
4. Cookies

HTTP Protocol

Protocol from 1989 that allows fetching resources from a server

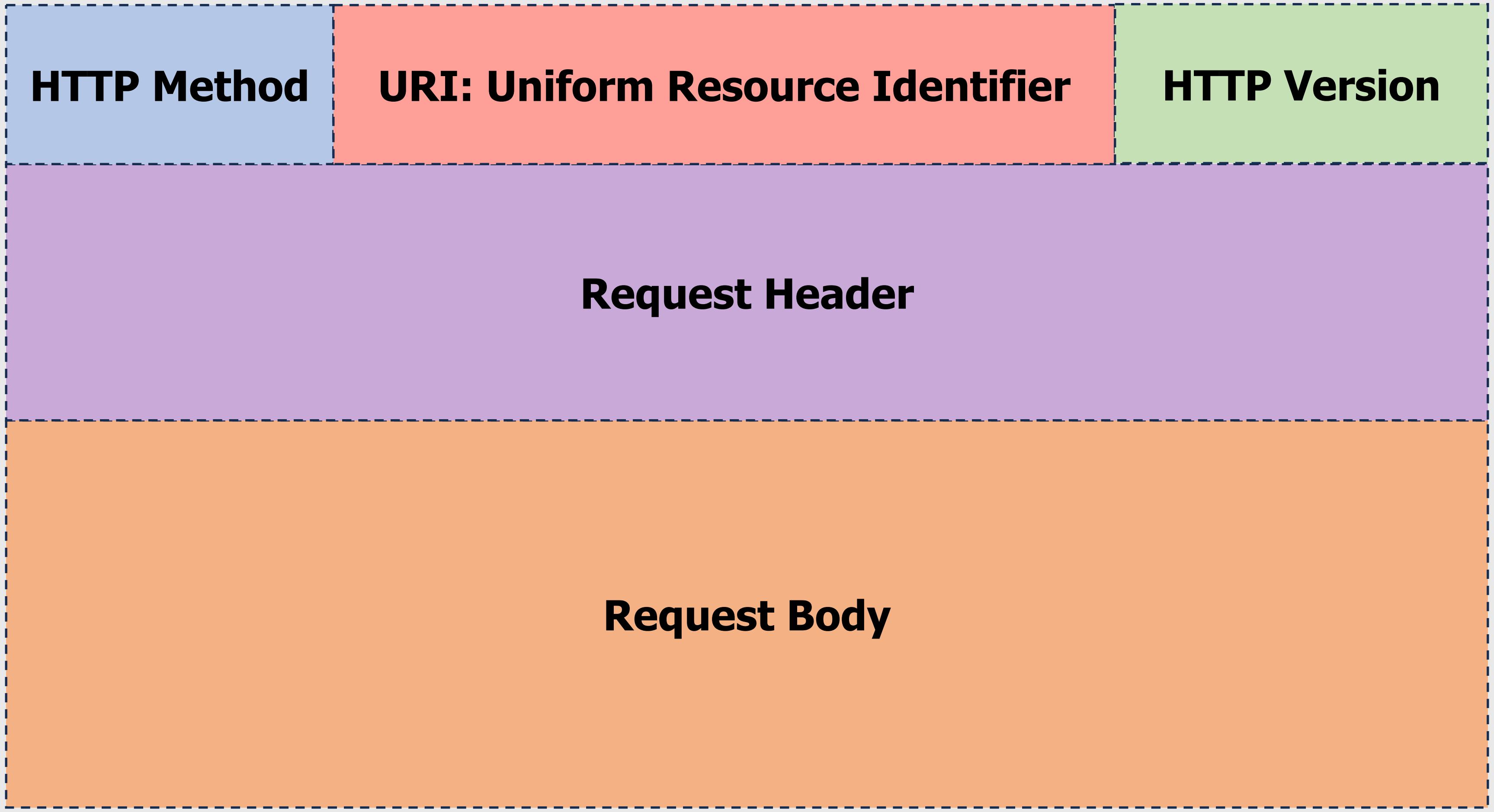
- Two messages: request and response
- Stateless protocol beyond a single request + response

Every resource has a uniform resource location (URL):

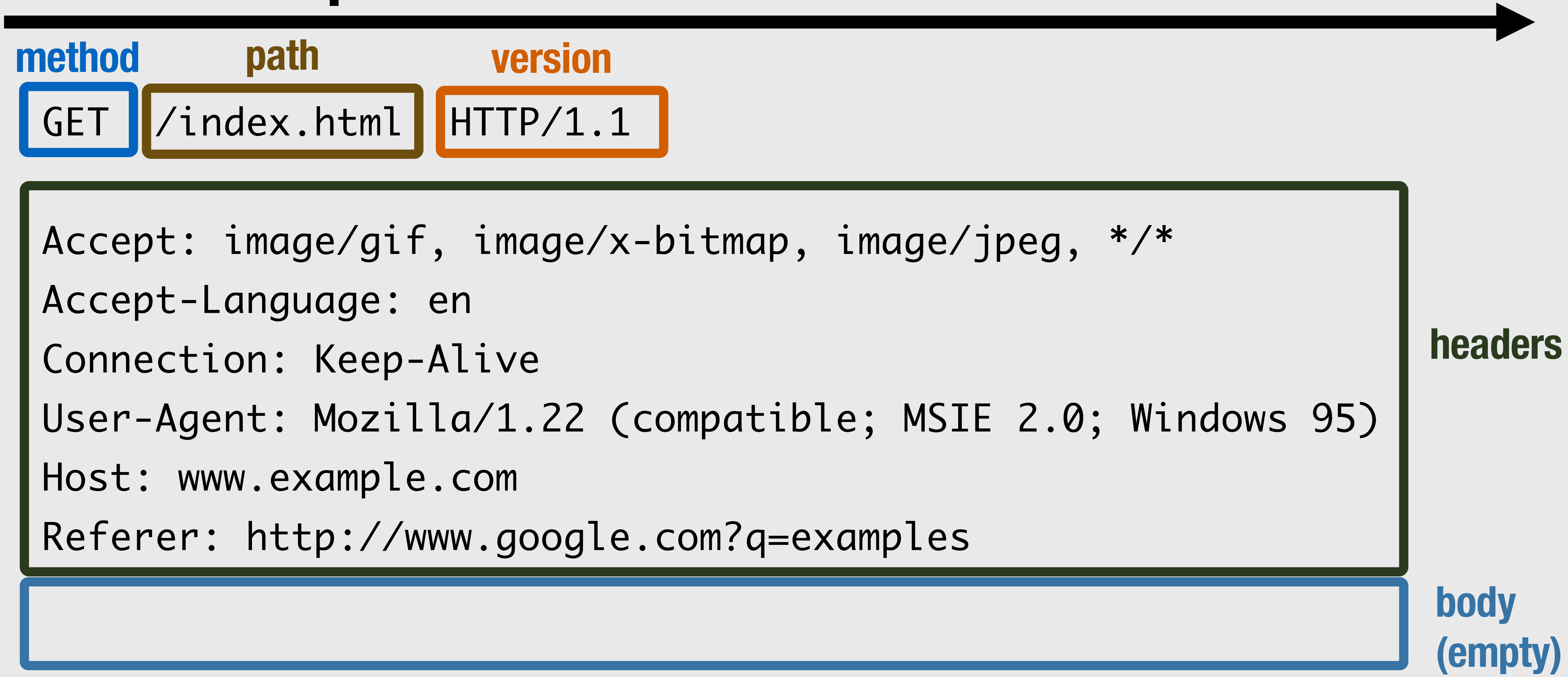


```
http://cs334.richmond.edu:80/lectures?lecture=08#slides
```

HTTP Request

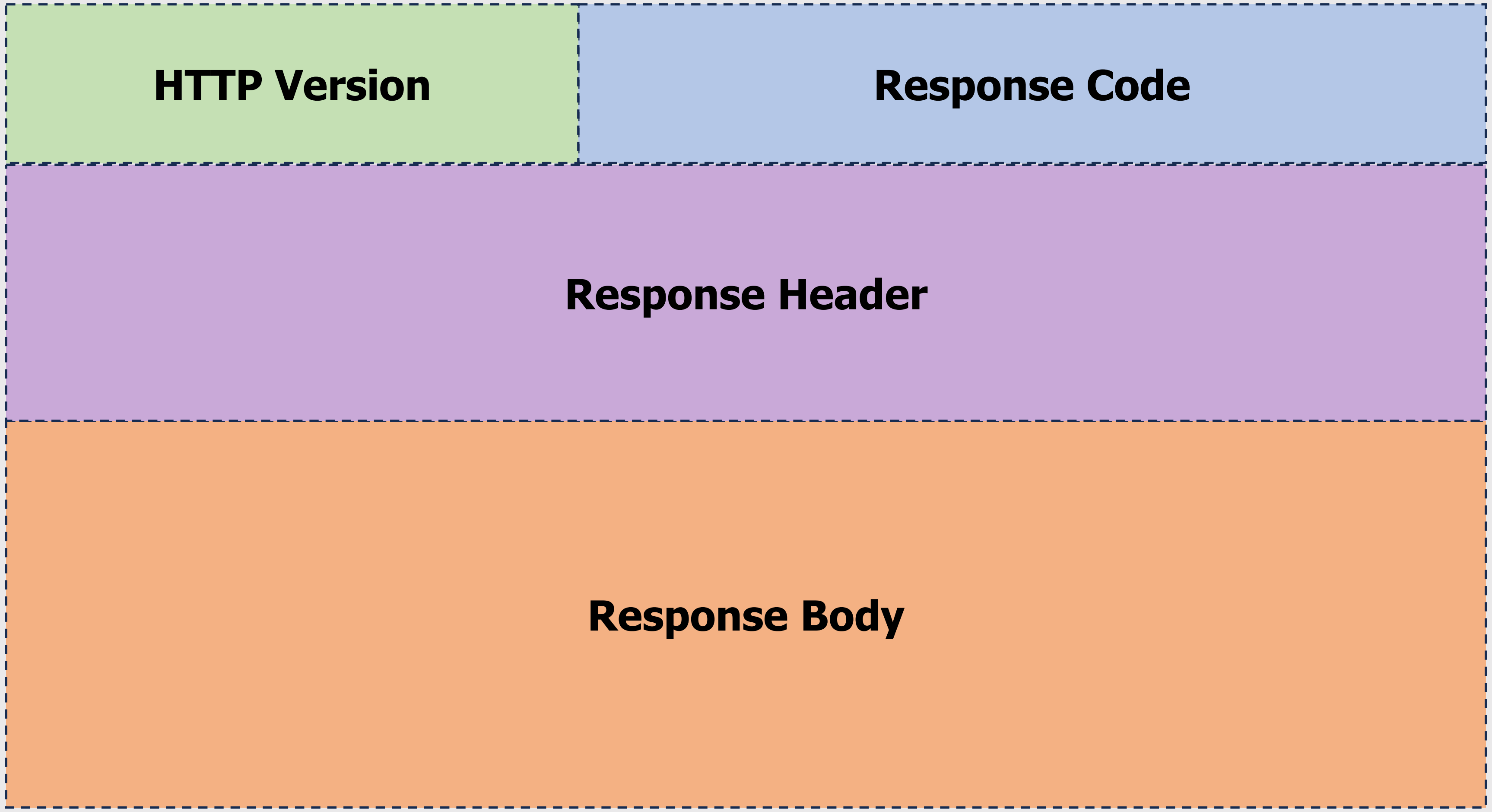


HTTP Request

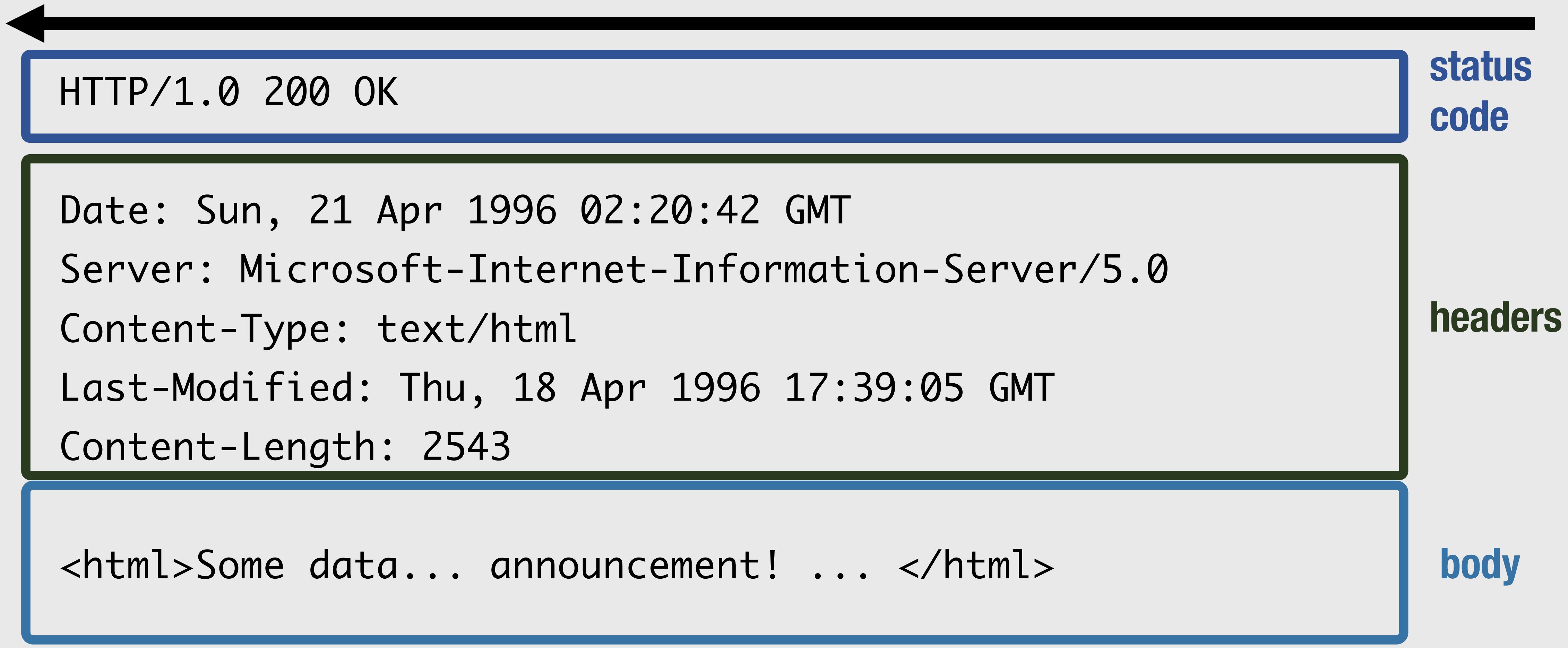


```
GET /index.html HTTP/1.1
Accept: image/gif, image/x-bitmap, image/jpeg, */*
Accept-Language: en
Connection: Keep-Alive
User-Agent: Mozilla/1.22 (compatible; MSIE 2.0; Windows 95)
Host: www.example.com
Referer: http://www.google.com?q=examples
```

HTTP Response



HTTP Response



```
HTTP/1.0 200 OK
```

```
Date: Sun, 21 Apr 1996 02:20:42 GMT
Server: Microsoft-Internet-Information-Server/5.0
Content-Type: text/html
Last-Modified: Thu, 18 Apr 1996 17:39:05 GMT
Content-Length: 2543
```

```
<html>Some data... announcement! ... </html>
```

HTTP Request

method path version

```
POST /index.html HTTP/1.1
```

headers

```
Accept: image/gif, image/x-bitmap, image/jpeg, */*
Accept-Language: en
User-Agent: Mozilla/1.22 (compatible; MSIE 2.0; Windows 95)
Host: richmond.edu
Referer: http://www.google.com?q=cs334
```

body

```
Class: Computer Security
Organization: University of Richmond
```

HTTP Methods

GET: Get the resource at the specified URL (does not accept message body)

POST: Create new resource at URL with payload

PUT: Replace target resource with request payload

PATCH: Update part of the resource

DELETE: Delete the specified URL

HTTP Methods

Not all methods are created equal — some have different security protections

`GET`s should not change server state

In practice, some servers do perform side effects

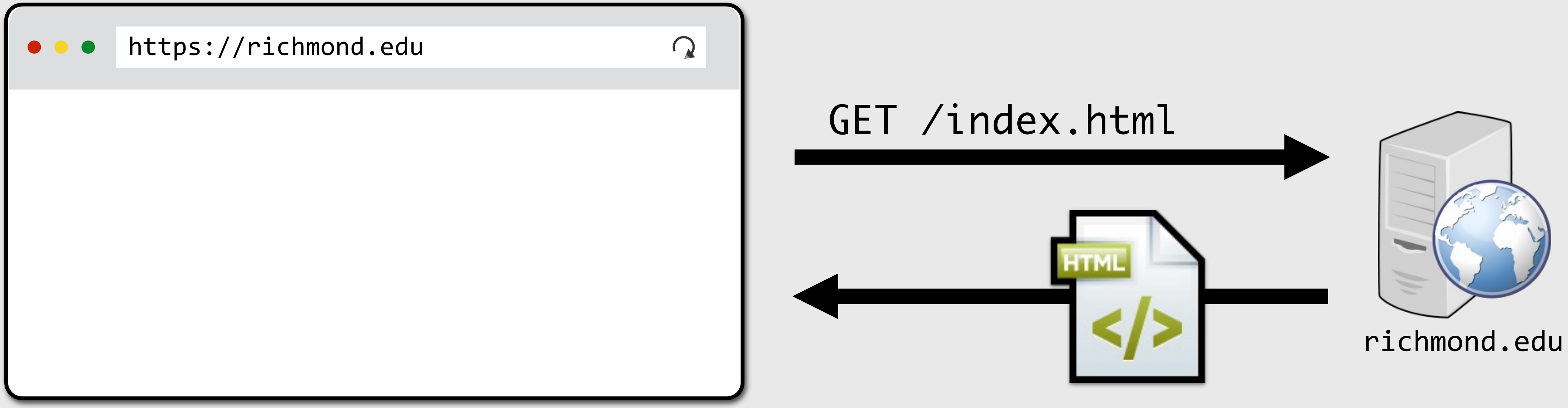
- Old browsers don't support **`PUT`**, **`PATCH`**, and **`DELETE`**
- Most requests with a side affect are **`POST`**s today
- Real method hidden in a header or request body

🙅 Never do…

```
GET
http://bank.com/transfer?fromAcct=ABC&toAcct=XYZ&amount=1000
```

HTTP → Website

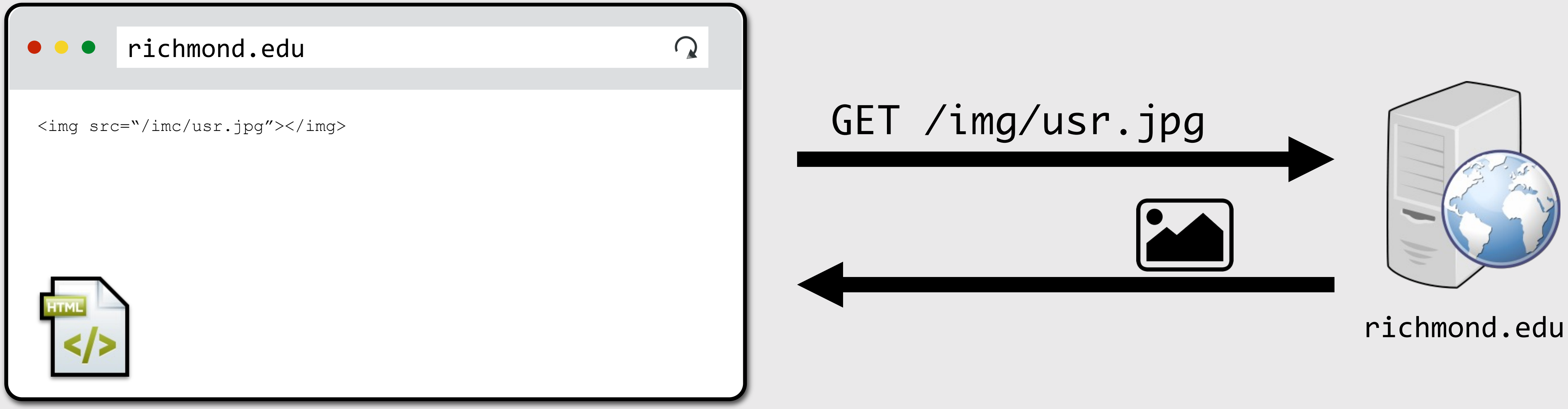
When you load a site, your web browser sends a **GET** request to that website



Loading Resources

Root HTML page can include additional resources like images, videos, fonts

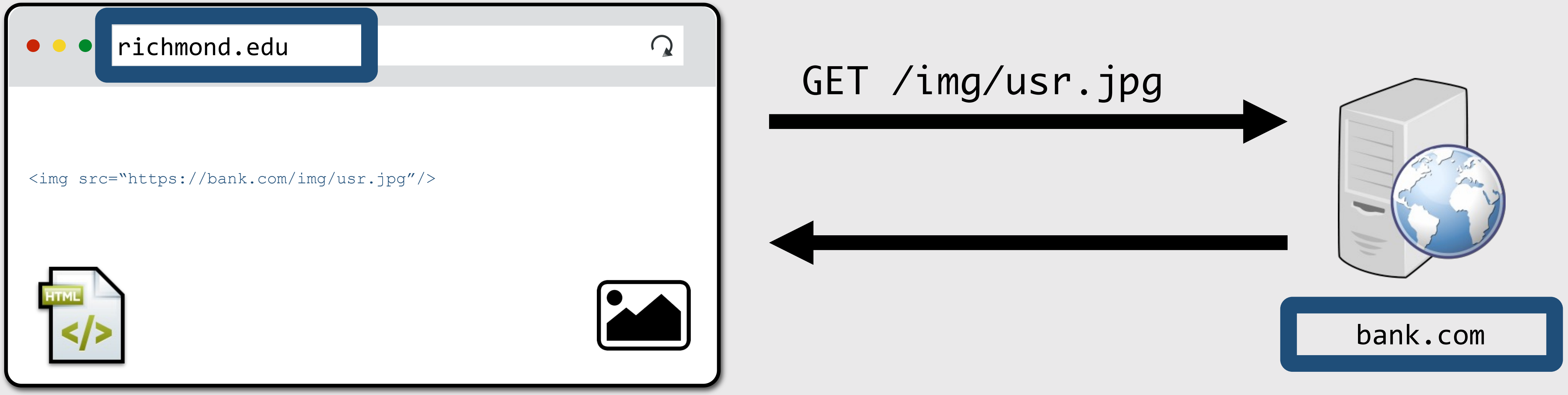
After parsing page HTML, your browser requests those additional resources



External Resources

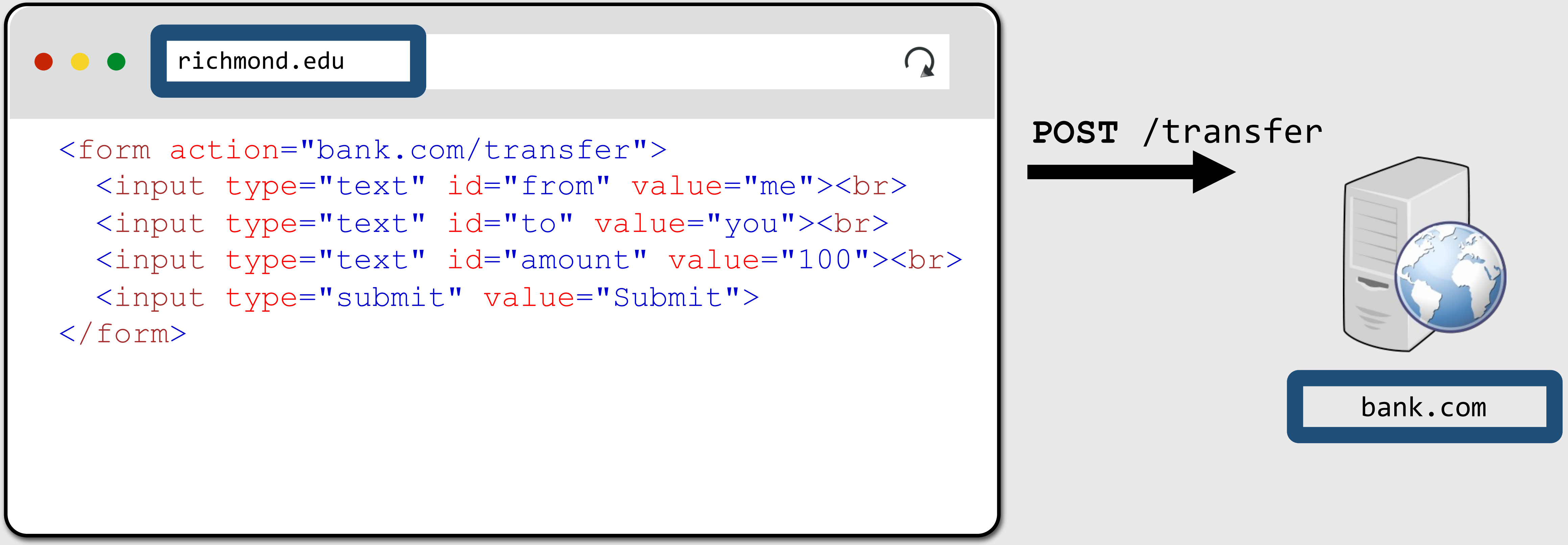
There are no restrictions on where you can load resources like images

Nothing prevents you from including images on a different domain



POST to external

You can also submit forms to any URL similar to how you can load resources



```
<form action="bank.com/transfer">
  <input type="text" id="from" value="me"><br>
  <input type="text" id="to" value="you"><br>
  <input type="text" id="amount" value="100"><br>
  <input type="submit" value="Submit">
</form>
```

Javascript

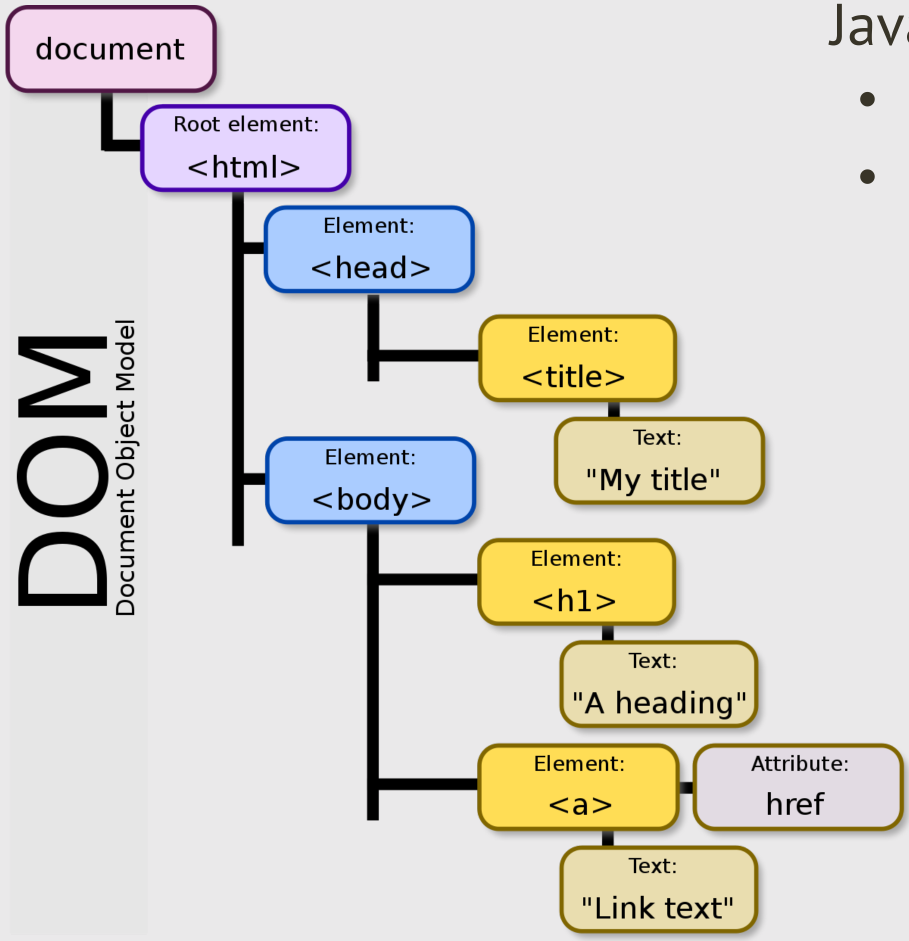
Historically, HTML content was static or generated by the server and returned to the web browser to simply render to the user

Today, websites also deliver scripts to be run inside of the browser

```
<button onclick="alert("The date is" + Date())">
  Click me to display Date and Time.
</button>
```

Javascript can make additional web requests, manipulate page, read browser data, local hardware — exceptionally powerful today

Document Object Model (DOM)



Javascript can read and modify page by interacting with DOM

- Object Oriented interface for reading/writing page content
- Browser takes HTML -> structured data (DOM)

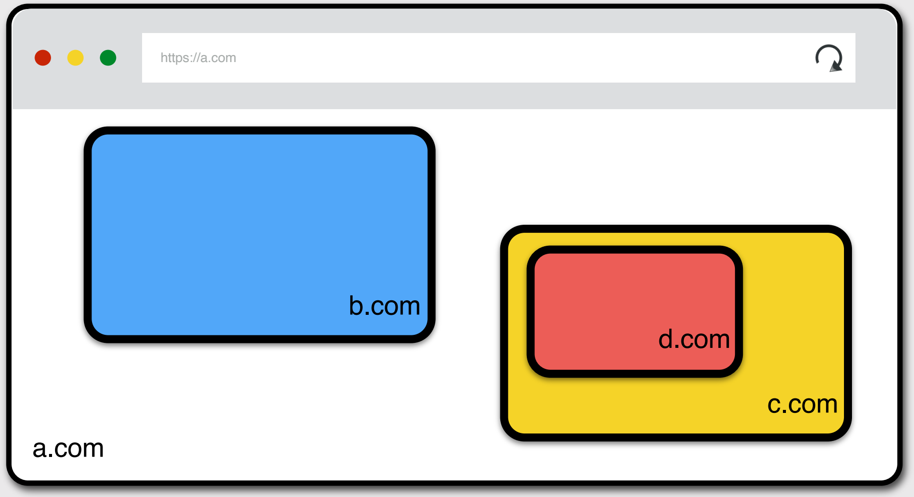
```
<p id="today"></p>

<script>
  document.getElementById('today').innerHTML = Date()
</script>
```

iFrames

Beyond loading individual resources, websites can also load other *websites* within their window

- Frame: rigid visible division
- iFrame: floating inline frame

Allows delegating screen area to content from another source (e.g., ad)



Browser Execution Model

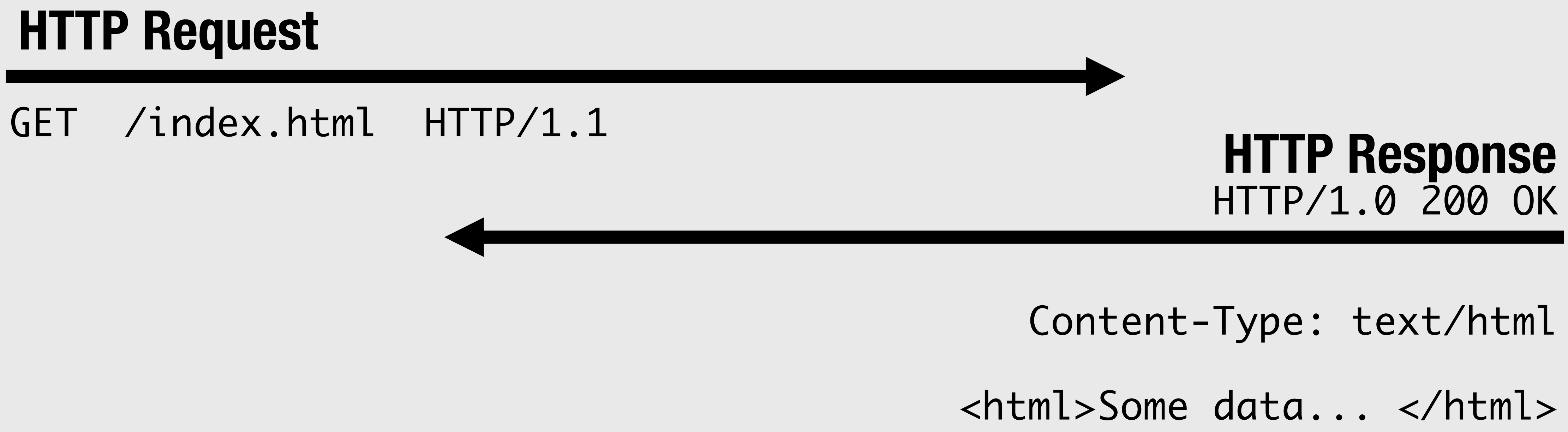
Each browser window….

- Loads content of root page
- Parses HTML and runs included Javascript
- Fetches additional resources (e.g., images, CSS, Javascript, iframes)
- Responds to events like onClick, onMouseover, onLoad, setTimeout
- Iterate until the page is done loading (which might be never)

Web Security Basics

1. Web Architecture
2. Web Server
3. HTTP Protocol
4. Cookies

HTTP is Stateless



HTTP Request

```
GET  /index.html  HTTP/1.1
```

HTTP Response

```
HTTP/1.0 200 OK

Content-Type: text/html

<html>Some data... </html>
```

If HTTP is stateless, how do we have website sessions?

HTTP Cookies

HTTP cookie: a small piece of data that a server sends to the web browser

The browser *may* store and send back in future requests to that site

Session Management

Logins, shopping carts, game scores, or any other session state

Personalization

User preferences, themes, and other settings

Tracking

Recording and analyzing user behavior

Setting Cookie

HTTP Response

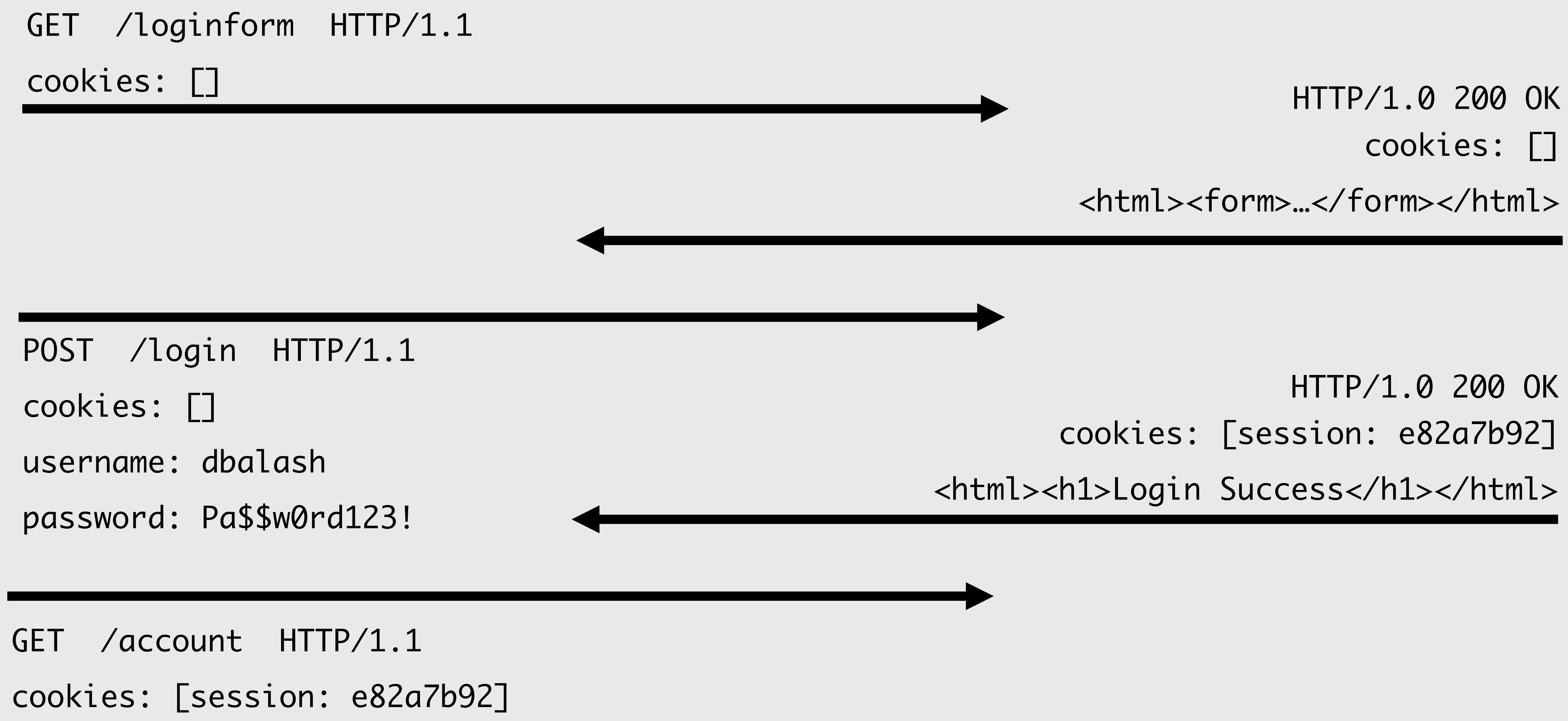
```
HTTP/1.0 200 OK
Date: Sun, 21 Apr 1996 02:20:42 GMT
Server: Microsoft-Internet-Information-Server/5.0
Connection: keep-alive
Content-Type: text/html
Set-Cookie: trackingID=3272923427328234
Set-Cookie: userID=F3D947C2
Content-Length: 2543

<html>Some data... whatever ... </html>
```

Sending Cookie

HTTP Request

```
GET  /index.html  HTTP/1.1

Accept: image/gif, image/x-bitmap, image/jpeg, */*
Accept-Language: en
Connection: Keep-Alive
User-Agent: Mozilla/1.22 (compatible; MSIE 2.0; Windows 95)
Cookie: trackingID=3272923427328234
Cookie: userID=F3D947C2
Referer: http://www.google.com?q=examples
```

Login Session



```
GET  /loginform  HTTP/1.1
cookies: []
```

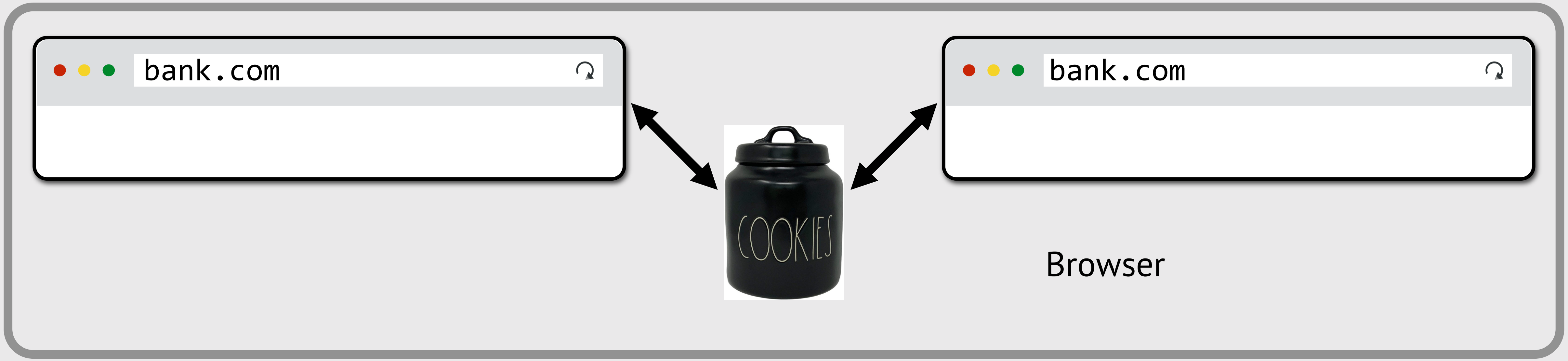
```
HTTP/1.0 200 OK
cookies: []
<html><form>…</form></html>
```

```
POST  /login  HTTP/1.1
cookies: []
username: dbalash
password: Pa$$w0rd123!
```

```
HTTP/1.0 200 OK
cookies: [session: e82a7b92]
<html><h1>Login Success</h1></html>
```

```
GET  /account  HTTP/1.1
cookies: [session: e82a7b92]
```

Shared Cookie Jar

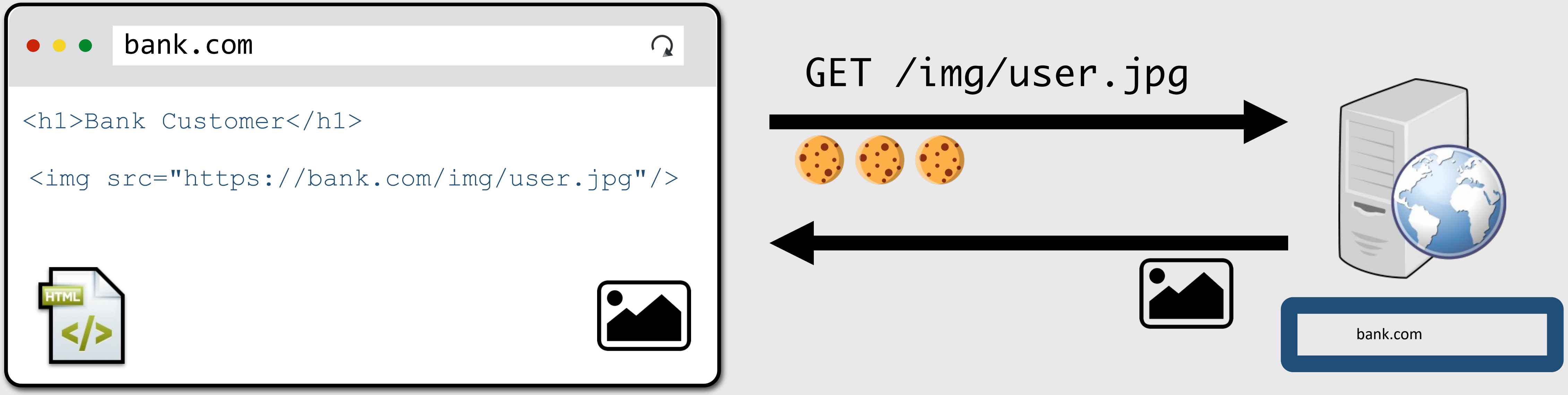


Both tabs share the same origin and have access to each others' cookies

(1) Tab 1 logins into bank.com and receives a cookie
(2) Tab 2's requests also send the cookies received by Tab 1 to bank.com

Cookies are always sent

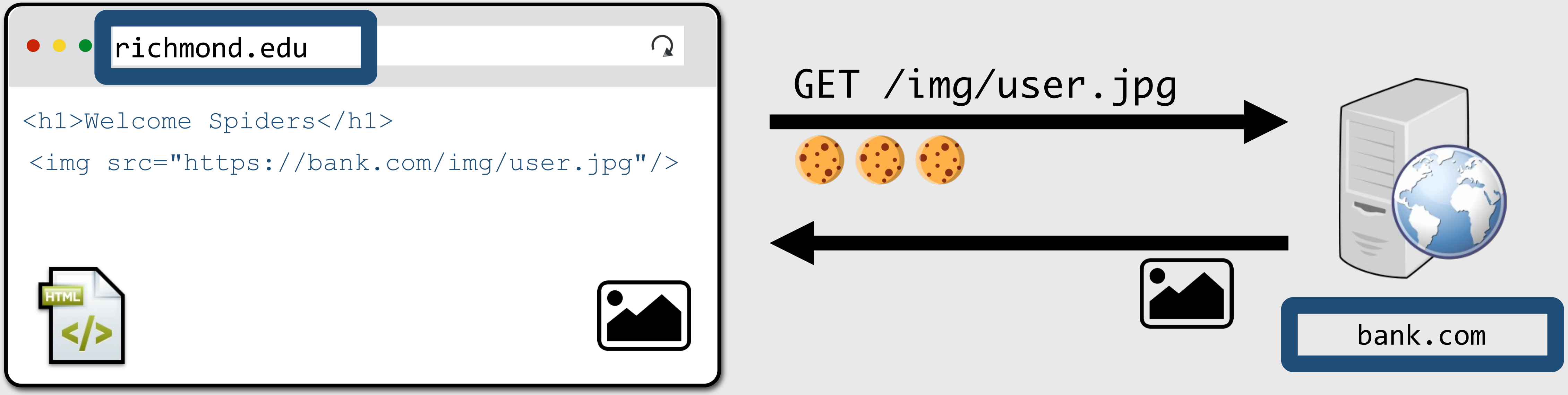
Cookies set be a domain are always sent for any request to that domain



```
<h1>Bank Customer</h1>

<img src="https://bank.com/img/user.jpg"/>
```

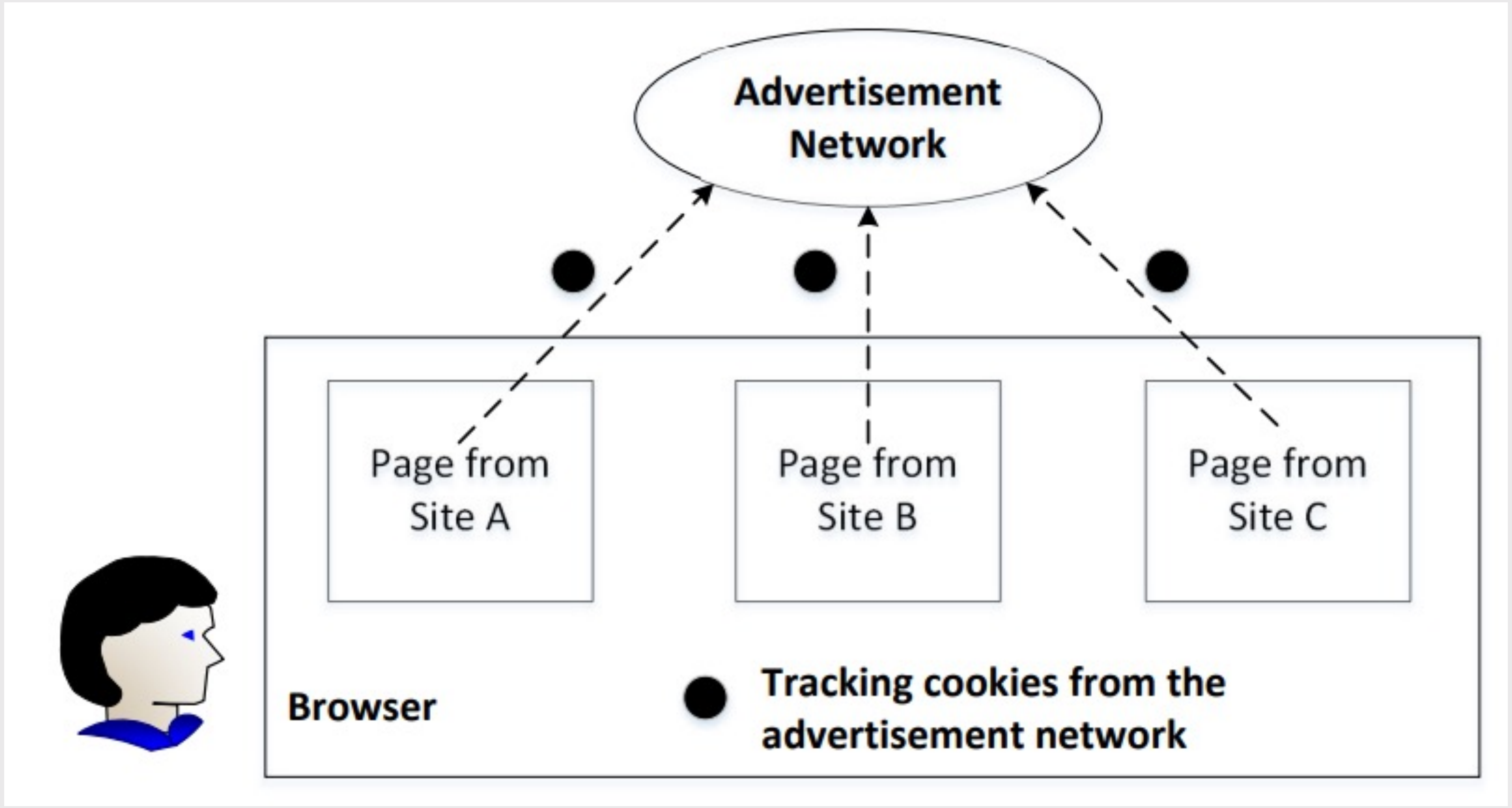
Cookies are always sent

Cookies set be a domain are always sent for any request to that domain



```
<h1>Welcome Spiders</h1>
<img src="https://bank.com/img/user.jpg"/>
```

Tracking Using Cookies



```
<img src="advertisement network’s website" width="1" height="1"/>
```

Prevent Tracking

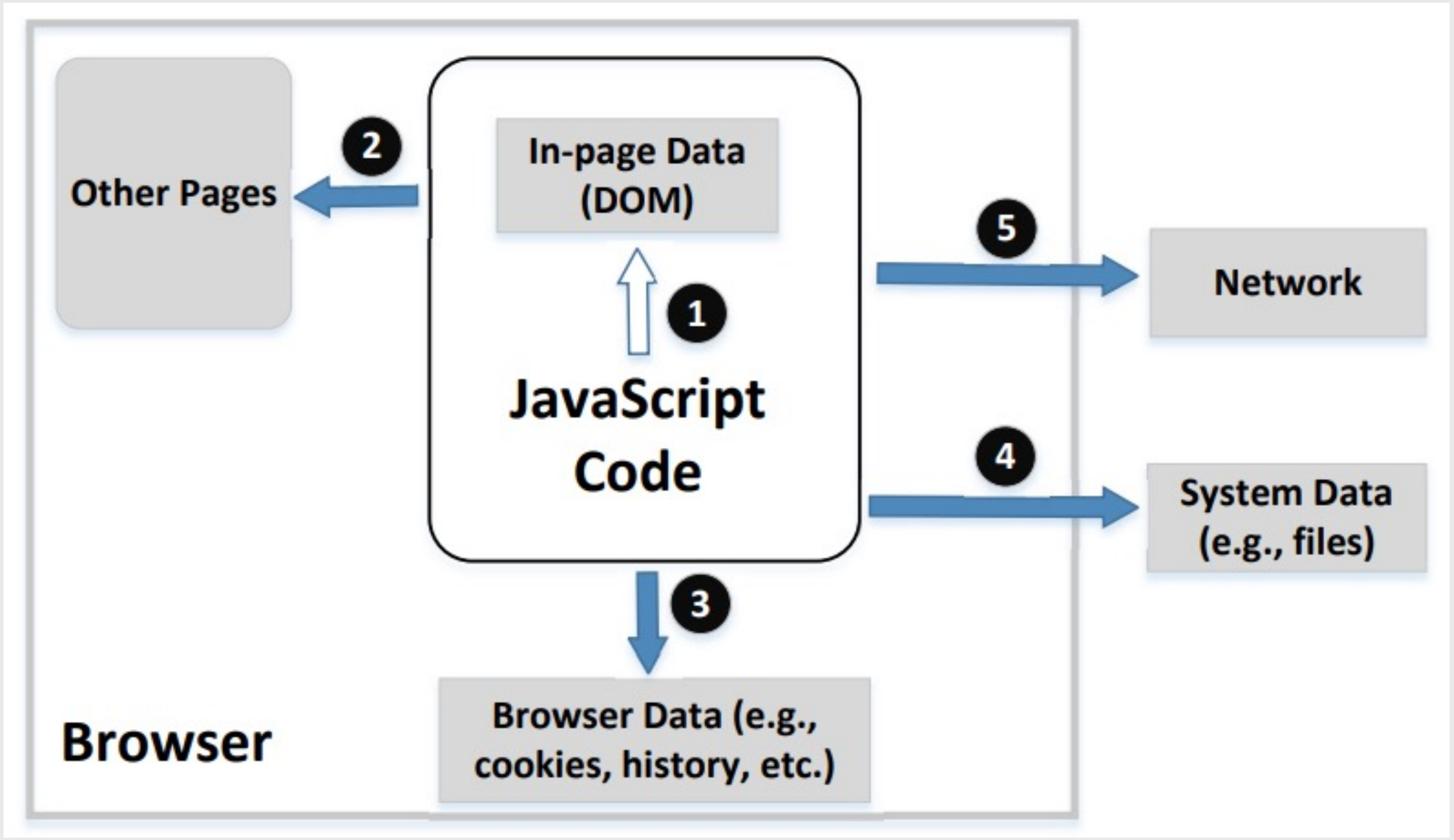
- Using anonymous mode in browsing

- Block third-party cookies
    - First-party cookies are essential for browsing
    - Third-part cookies are mainly used for advertisement, information collection, etc.

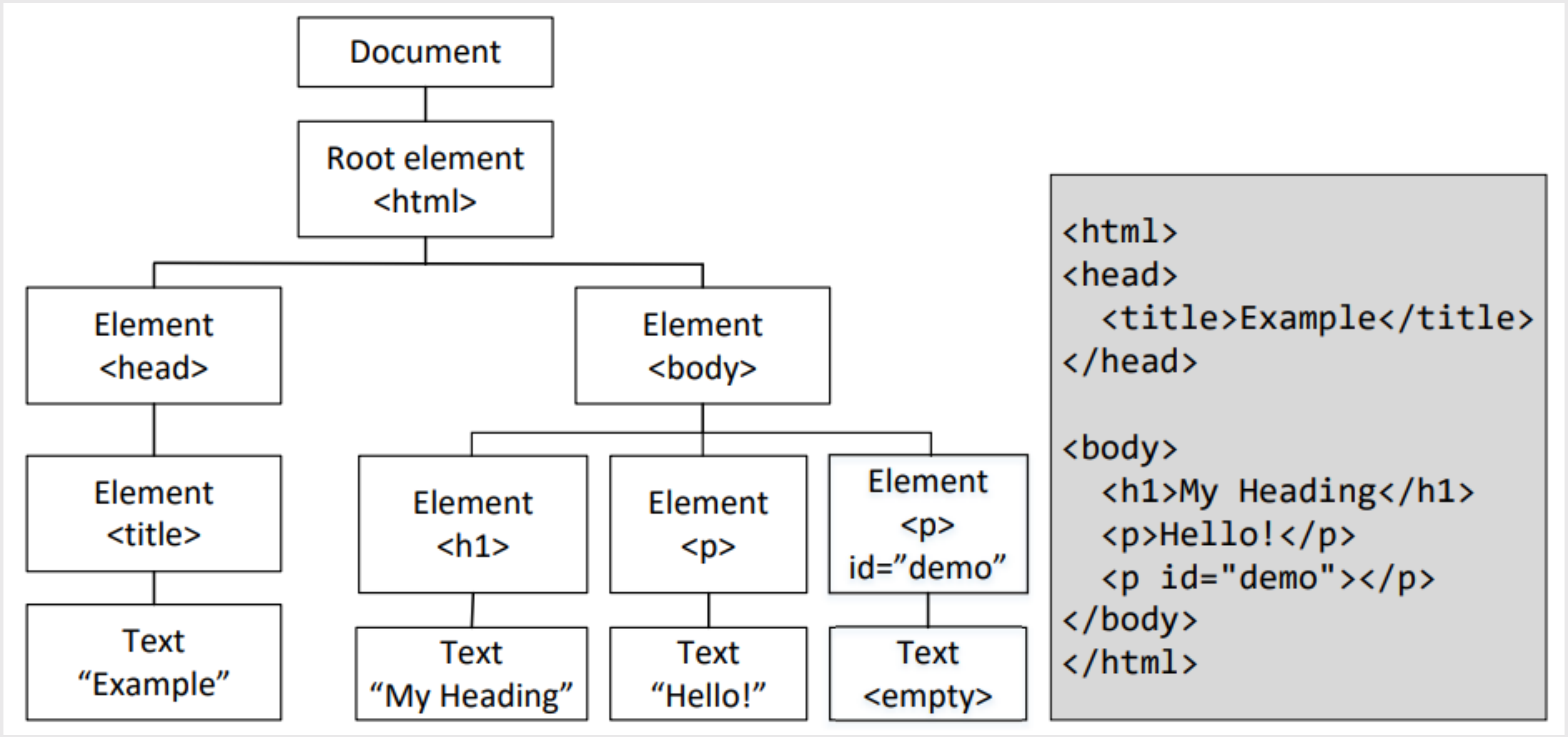
Web Security Basics

1. Web Architecture
2. Web Server
3. HTTP Protocol
4. Cookies
5. JavaScript and Sandboxing

Protection Needs


Other Pages
2
In-page Data (DOM)
1
JavaScript Code
5
Network
4
System Data (e.g., files)
3
Browser Data (e.g., cookies, history, etc.)
Browser

Access Page Data and DOM



```
<html>
<head>
  <title>Example</title>
</head>

<body>
  <h1>My Heading</h1>
  <p>Hello!</p>
  <p id="demo"></p>
</body>
</html>
```

```
document.getElementById('demo').innerHTML = 'Hello World'
```

Access File System

- JavaScript cannot directly access local file system
- User needs to grant permission via file selection

```
<input type="file" id="file-selector">
```

File selection: **grant permissions by selection**

```
var files = document.getElementById(’file-selector’).files;
```

Get the file handlers