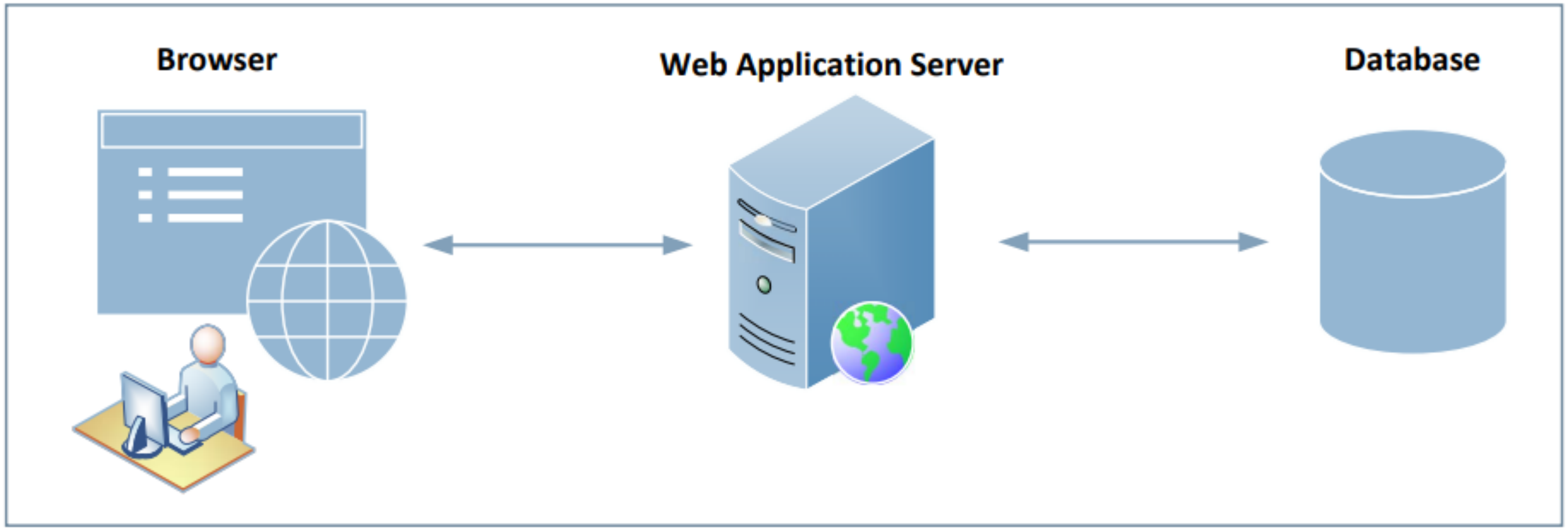
Web Security Basics

Outline

- The web architecture
- Web server
- HTTP protocol, cookies
- JavaScript and sandboxing

The Web Architecture



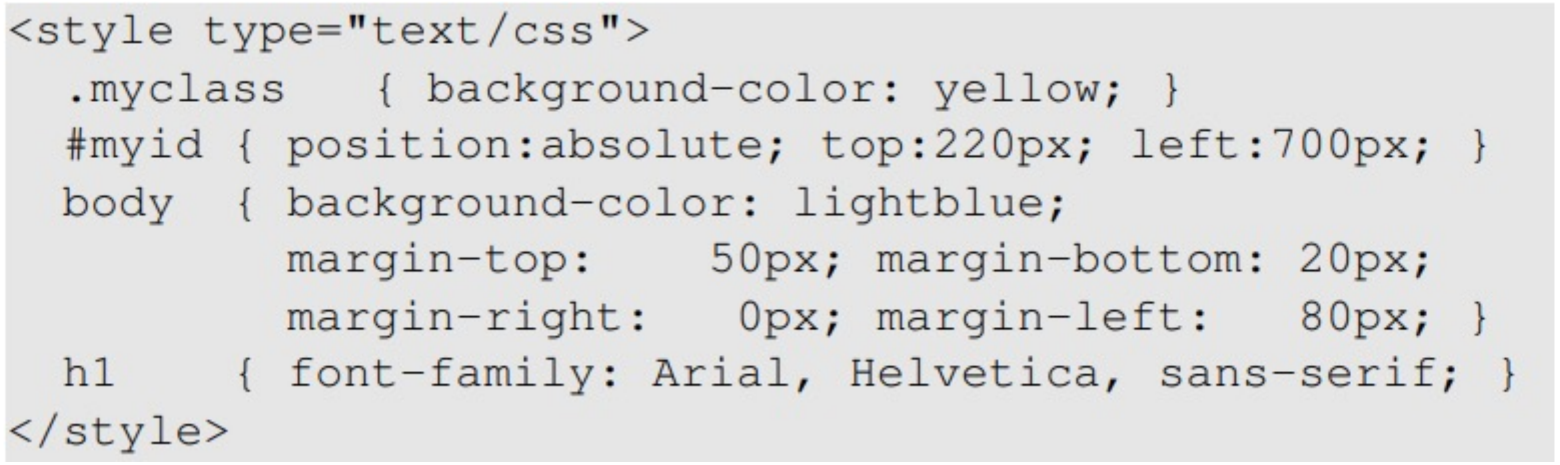
HTML

- 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

- Adobe Flash
- Microsoft Silverlight
- ActiveX
- Java applets
- **JavaScript**

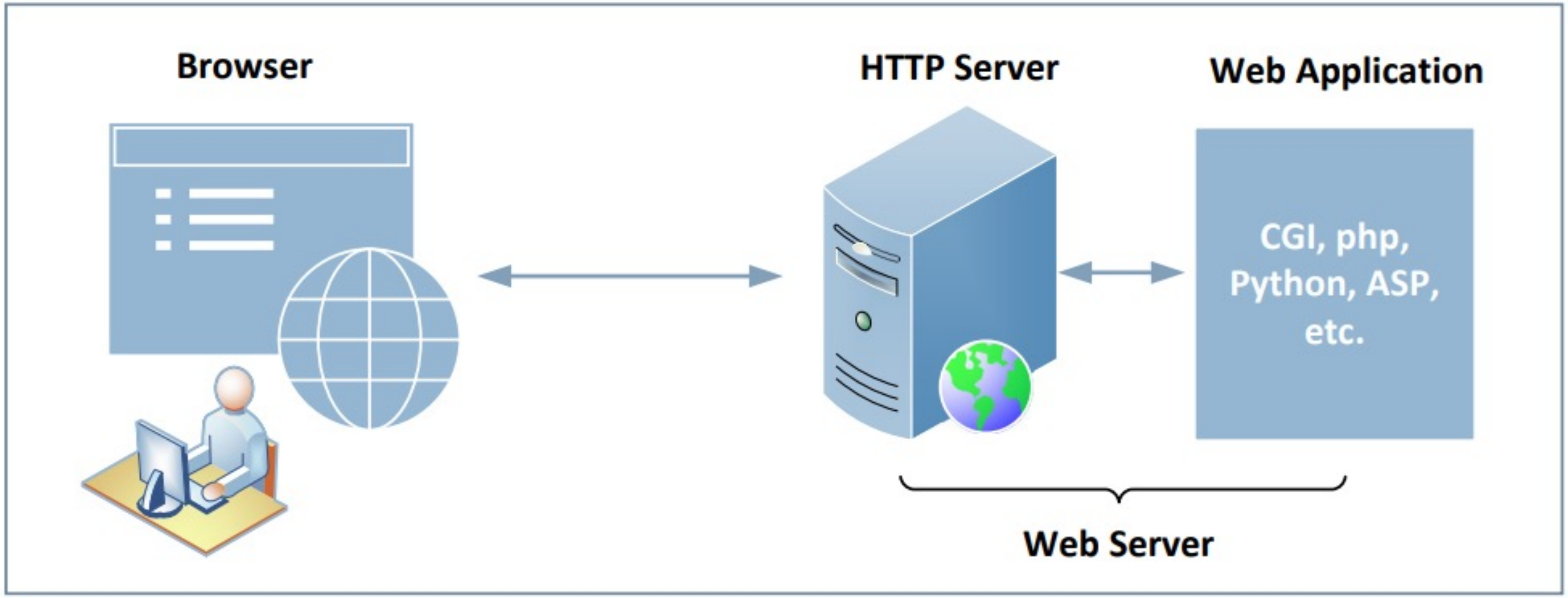
JavaScript

- 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>
```

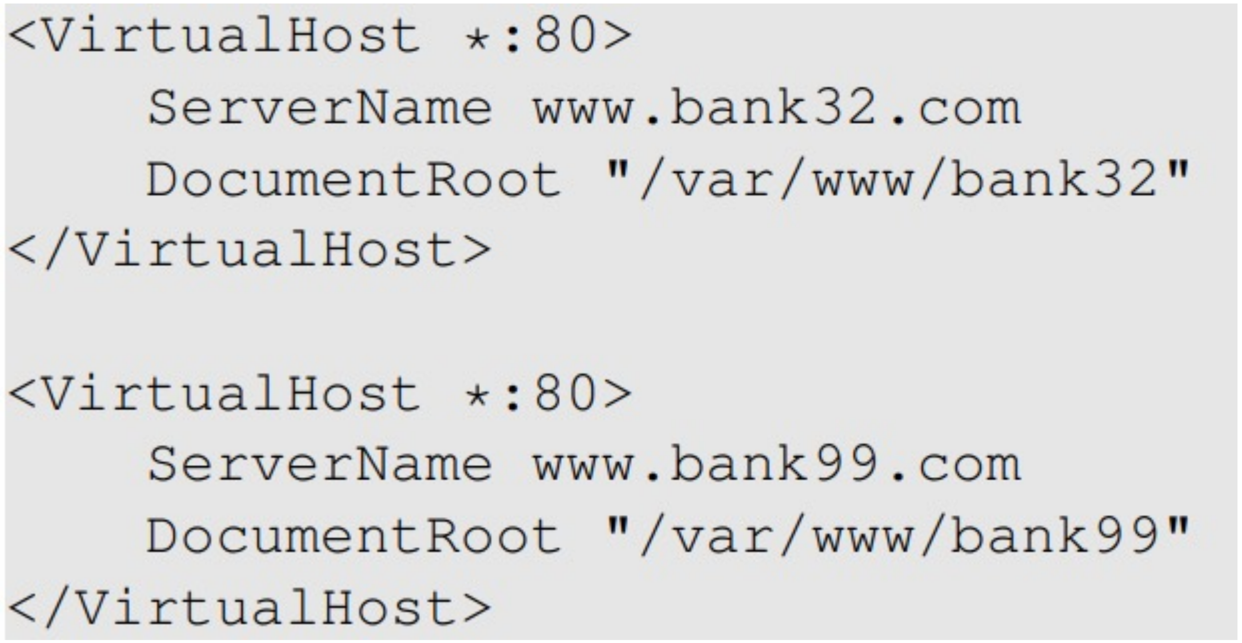
WEB SERVER

HTTP Server & Web Application Server



Case Study: Apache Server

Configuration: Virtual Hosting



```
<VirtualHost *:80>
    ServerName www.bank32.com
    DocumentRoot "/var/www/bank32"
</VirtualHost>

<VirtualHost *:80>
    ServerName www.bank99.com
    DocumentRoot "/var/www/bank99"
</VirtualHost>
```

How HTTP Server Interacts with Web Applications

- CGI: The Common Gateway Interface
  - Starts the CGI program in a new process
- FastCGI: a variation of the CGI, faster
- Modules: directly execute script-based programs

PHP Example

Inline Approach

```
<!doctype html>
<html>
<body>
<h1>PHP Experiment</h1>
<h2>Current time is
<?php echo date("Y-m-d h:i:sa") ?>
</h2>
</body>
</html>
```

Template Approach

```
<?php
 $title = "PHP Experiment";
 $time = date("Y-m-d h:i:sa")
?>

<!doctype html>
<html>
<body>
<h1><?=$title?></h1>
<h2>Current time is <?=$time?></h2>
</body>
</html>
```

HTTP PROTOCOL

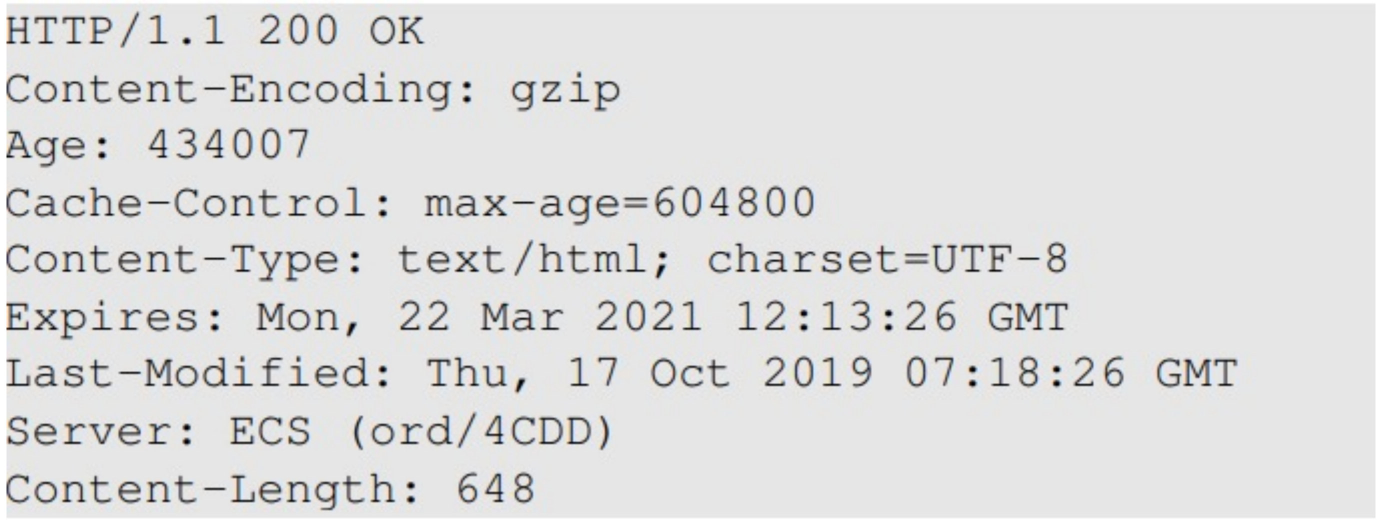
HTTP: Interacting with Server (1)

- An example of HTTP request

```
GET /index.html HTTP/1.1                    ①
Host: www.example.com                       ②
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:83.0) ...
Accept: text/html,application/xhtml+xml,application/xml; ...
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1
```

HTTP: Interacting with Server (2)

- An example of HTTP response



```
HTTP/1.1 200 OK
Content-Encoding: gzip
Age: 434007
Cache-Control: max-age=604800
Content-Type: text/html; charset=UTF-8
Expires: Mon, 22 Mar 2021 12:13:26 GMT
Last-Modified: Thu, 17 Oct 2019 07:18:26 GMT
Server: ECS (ord/4CDD)
Content-Length: 648
```

HTTP 404 (not found)
A Joke: I had trouble finding my new classroom. Room 404, classroom not found

GET versus POST Requests

- Main difference
  - how they send data to the server

- GET request

```
GET /post_form.php?foo=hello&bar=world HTTP/1.1
Host: www.example.com
Cookie: SID=xsdfgergbghedvrbeadv
```

- Post request

```
POST /post_form.php HTTP/1.1
Host: www.example.com
Cookie: SID=xsdfgergbghedvrbeadv
Content-Length: 19
foo=hello&bar=world
```

Cookies

- Web server is stateless
  - does not maintain a long-term connection with the client
- HTTP Cookies: used to save information on the client side
  - Browser save cookies
  - Attach cookies in every request

Setting Cookies

- Server: setting cookies

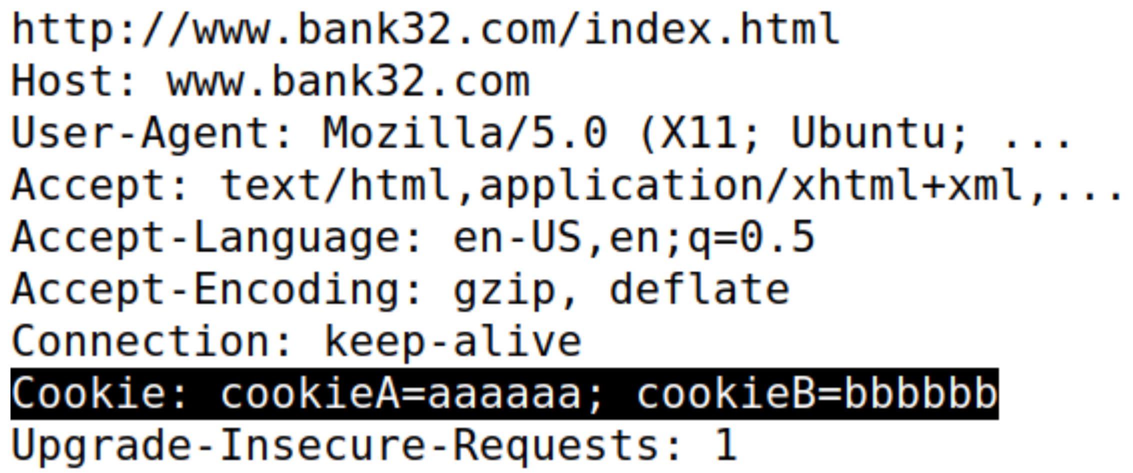
```
<?php
  setcookie('cookieA', 'aaaaaa');
  setcookie('cookieB', 'bbbbbb', time() + 3600);

  echo "<h2>Cookies are set</h2>"
?>
```

- HTTP response

```
GET: HTTP/1.1 200 OK
Date: Wed, 25 Aug 2021 20:40:15 GMT
Server: Apache/2.4.41 (Ubuntu)
Set-Cookie: cookieA=aaaaaa
cookieB=bbbbbb; expires=Wed, 25-Aug-2021 21:40:15 GMT; Max-Age=3600
Content-Length: 28
Keep-Alive: timeout=5, max=99
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8
```

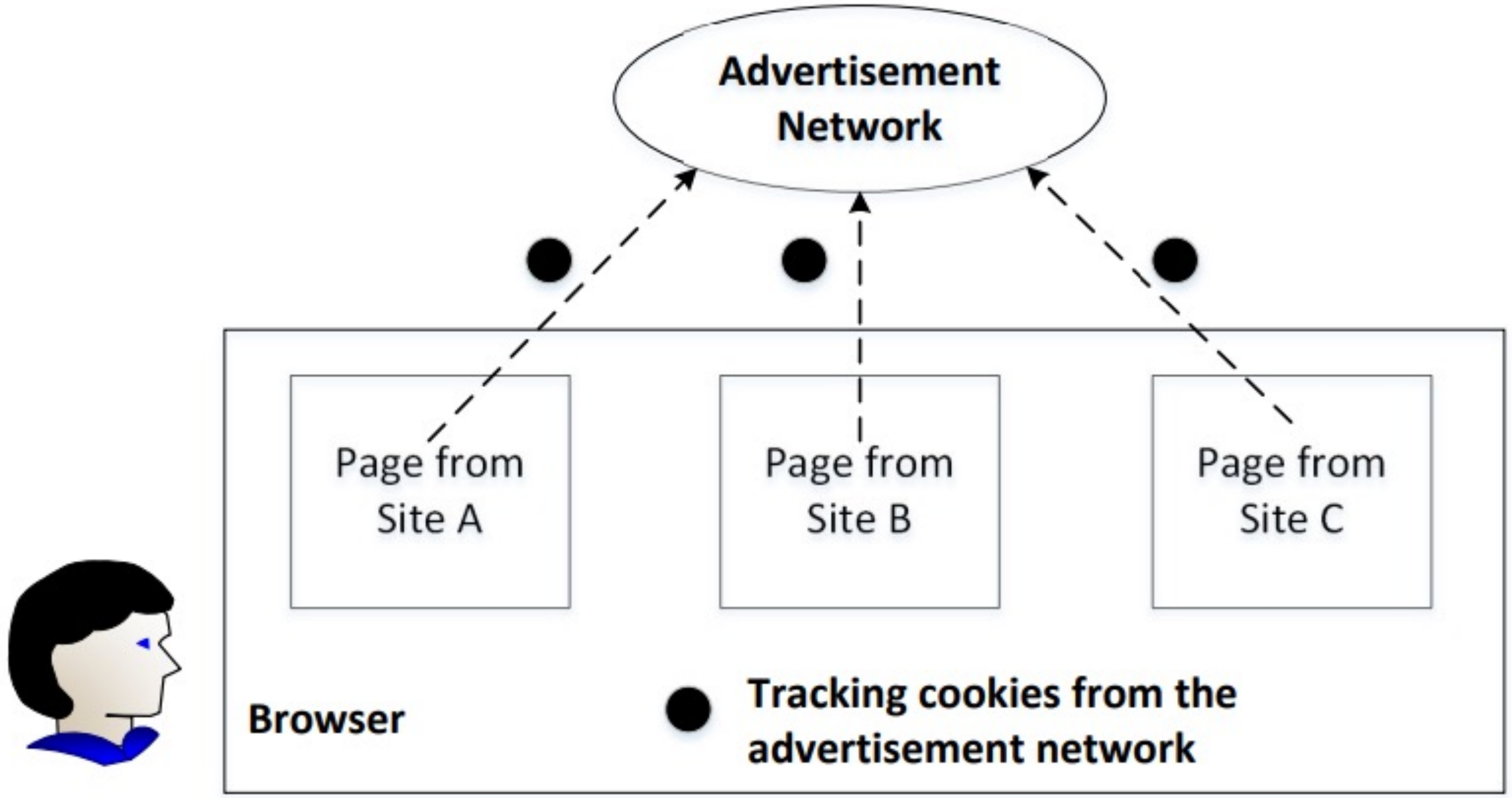
Attaching Cookies

- Browser: attach all the cookies belonging to the target server



```
http://www.bank32.com/index.html
Host: www.bank32.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; ...
Accept: text/html,application/xhtml+xml,...
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Cookie: cookieA=aaaaaa; cookieB=bbbbbb
Upgrade-Insecure-Requests: 1
```

Tracking Using Cookies



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

Prevent Tracking

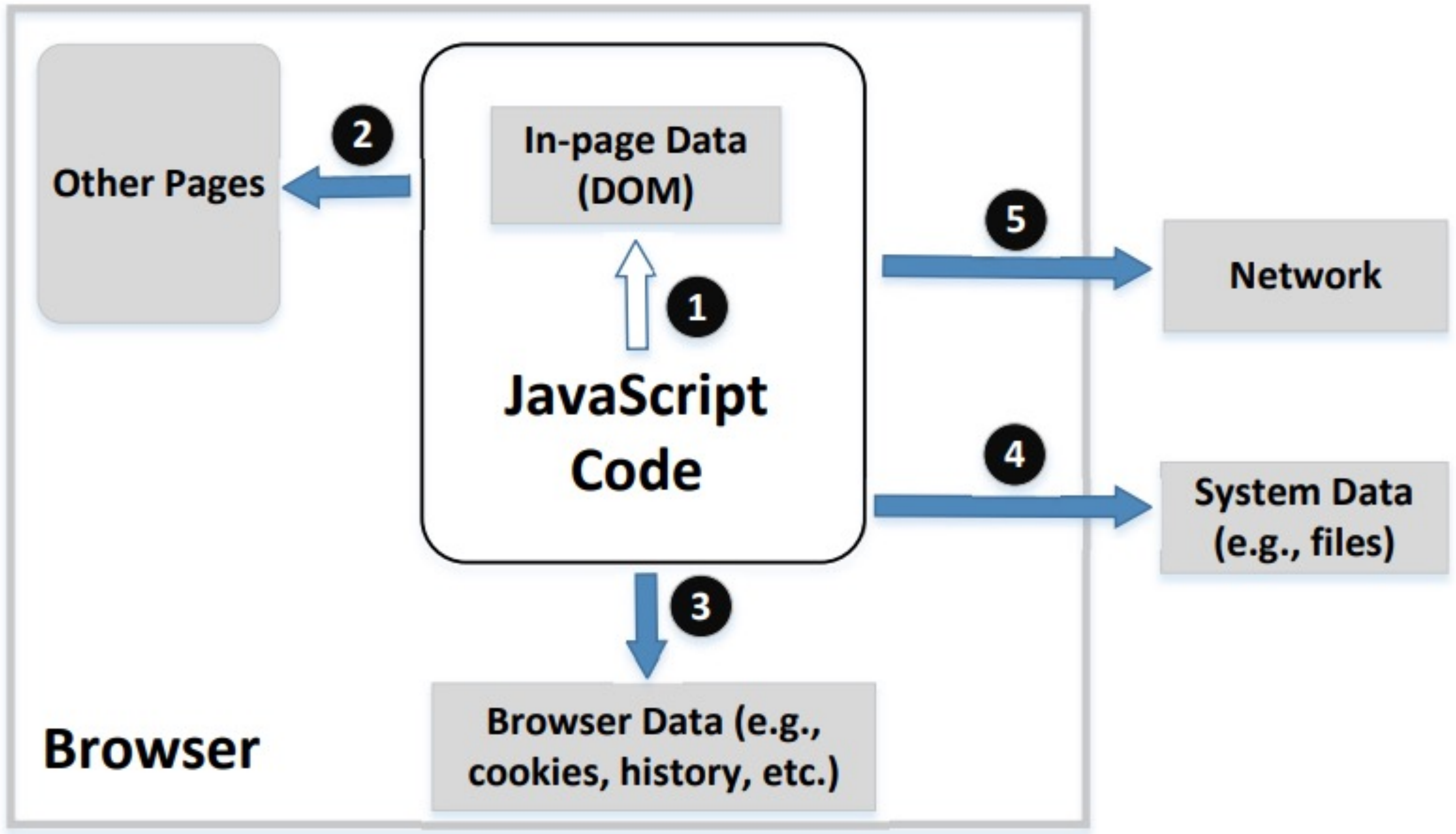
- Using anonymous mode in browsing
- Block third-party cookies
  - First-party cookies are essential for browsing
  - Third-part cookies are mainly used for advertisement, information collection, etc.

Session Cookies

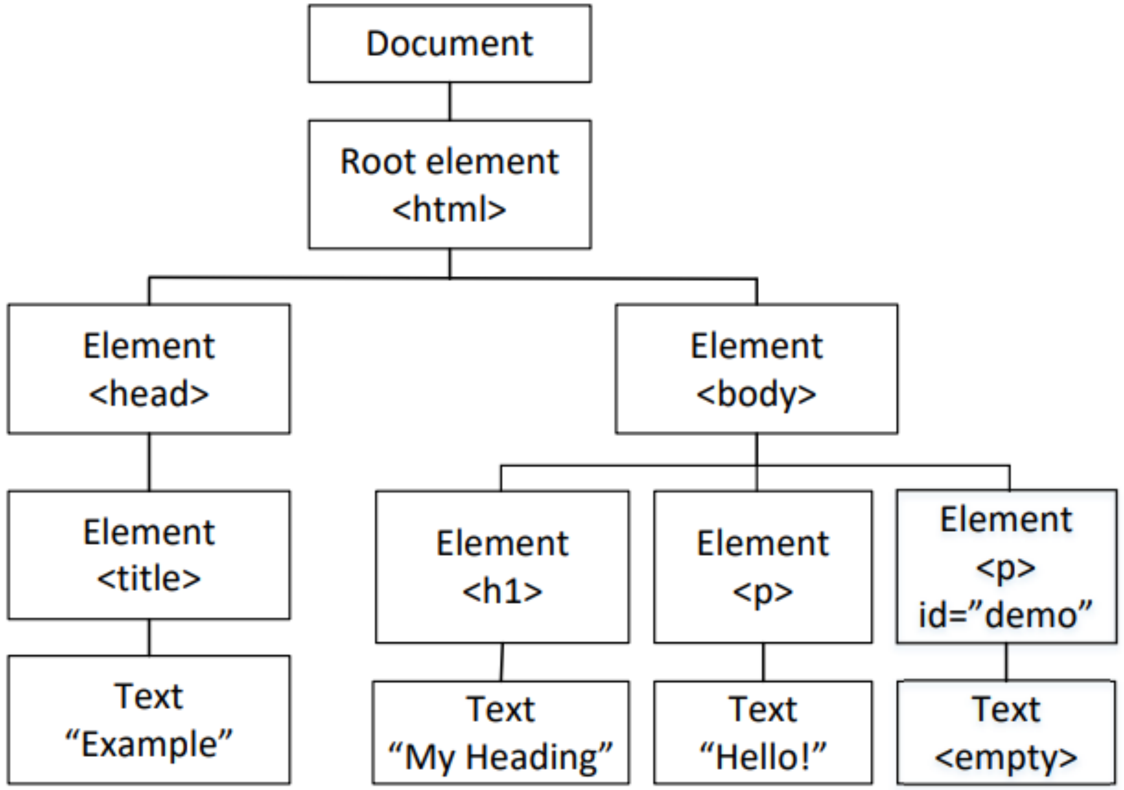
- A cookie: store session ID
- The session ID identifies a session
- Session data are typically maintained on the server
- Session is typically created after user login
  - Have the session ID = have the access
  - Security sensitive
  - ID: Random number

JAVASCRIPT AND SANDBOX

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

Access Network and Ajax

- Three communication mechanisms
  - Normal HTTP
  - Ajax
  - WebSocket
- Security policies are different

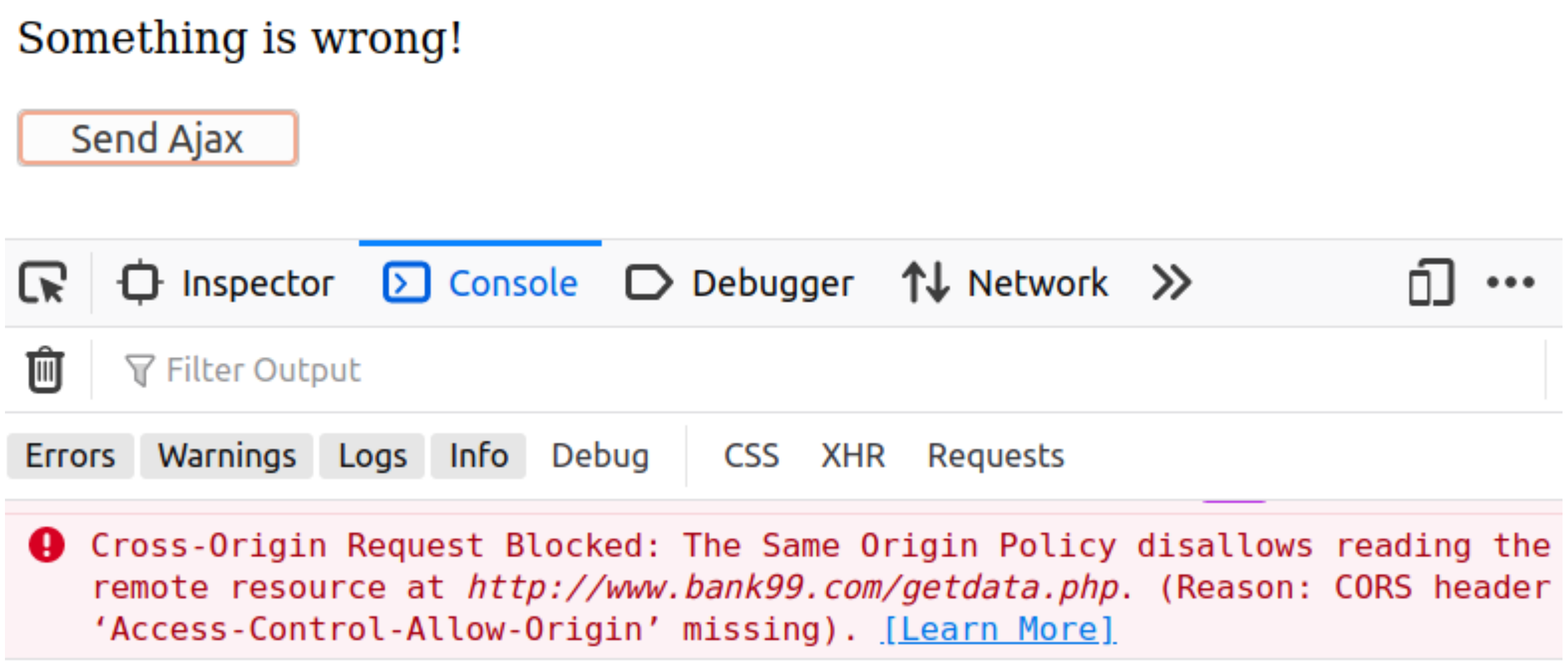
Ajax Example

- Asynchronous JavaScript and XML (Ajax)
- Slowly being superseded by the Fetch API
- Ajax example

```
function send_ajax()
{
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {      ②
    if (this.readyState == 4 && this.status == 200) {
      document.getElementById("demo").innerHTML = this.responseText;
    }
  };
  xhttp.open("GET", "http://www.bank32.com/getdata.php", true); ③
  xhttp.send();
}
```

Same Origin Policy on Ajax

- Page from www.bank32.com trying to access www.bank99.com (using Ajax)


Something is wrong!

```
Cross-Origin Request Blocked: The Same Origin Policy disallows reading the
remote resource at http://www.bank99.com/getdata.php. (Reason: CORS header
'Access-Control-Allow-Origin' missing). [Learn More]
```

What is Blocked: Request or Response?

- Request was sent out, response came back
- Browser blocks Ajax code from accessing the response

```
$ sudo tcpdump -i br-3f00b5edf2b0 -n -v
10.9.0.1.42580 > 10.9.0.5.80: ... HTTP, length: 316
   GET /getdata.php HTTP/1.1
   Host: www.bank32.com
   Origin: http://www.bank99.com
   Referer: http://www.bank99.com/ajax.html

10.9.0.5.80 > 10.9.0.1.42580: ... HTTP, length: 224
   HTTP/1.1 200 OK
   Server: Apache/2.4.41 (Ubuntu)
   Content-Type: text/html; charset=UTF-8

   Data from Bank32!
```

Why Blocking the Response?

- Cross-Origin access compromise privacy
  - Same-origin policy is enforced
- Example: Ajax code in Facebook page
  - allowed to access the user's Facebook data
  - not allowed to access the user's Google data

Relaxing the Restriction

- The same-origin policy is too restrictive
- CORS (Cross-Origin Resource Sharing)
  - Whitelist provided by server: grant permissions
- CORS policy on www.bank99.com

```
<?php
 header("Access-Control-Allow-Origin: http://www.bank32.com");

 echo "Time from Bank99: ".date("h:i:sa")
?>
```

WebSocket

- Ajax uses HTTP: half-duplex
  - Browser sends request, server responds
  - No “push” mechanism (from server)
- WebSocket is full-duplex
  - Both browser/server can send data (without request)

Security Policy on WebSocket

- Browser does not restrict data from WebSocket
  - Different from Ajax
  - Access control on client side
- Access control is conducted on server side
  - Check the “Origin” of the request

Cable Haunt Attack

- Discovered: January 2020
- Affected many Broadcom-based cable modems
- These modems run a WebSocket-based server program
  - JavaScript code can interact with the server: a door is open
  - Attacker exploits a buffer overflow vulnerability on the server