

# How the Web Works

CMSC 23200, Spring 2025, Lecture 11

---

Grant Ho

University of Chicago, 04/29/2025

(Slides adapted from Blasé Ur, Vern Paxson, and Zakir Durumeric)

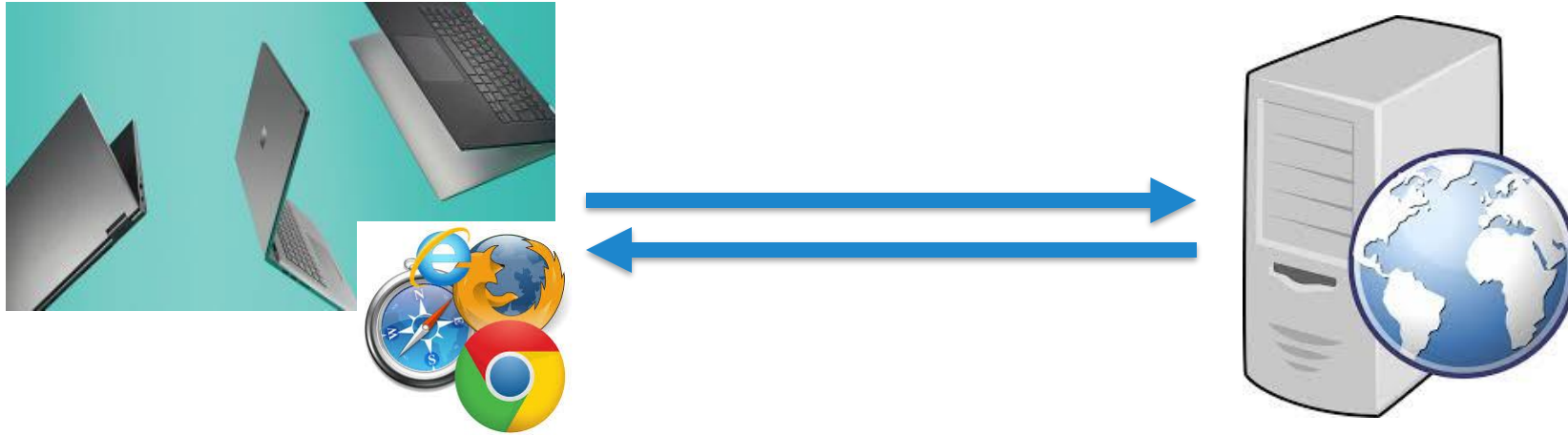
# Logistics

- Discussion Section #4 this Wednesday (04/30)
- Assignment 4 due Friday at 11:59pm (5/2)
  - Start early!
  - See my Ed post for some tips & debugging notes

# Outline

- Web Overview
- Navigating the Web
- Webpage Structure & Contents
- Web Security Threat Models
- Same Origin Policy

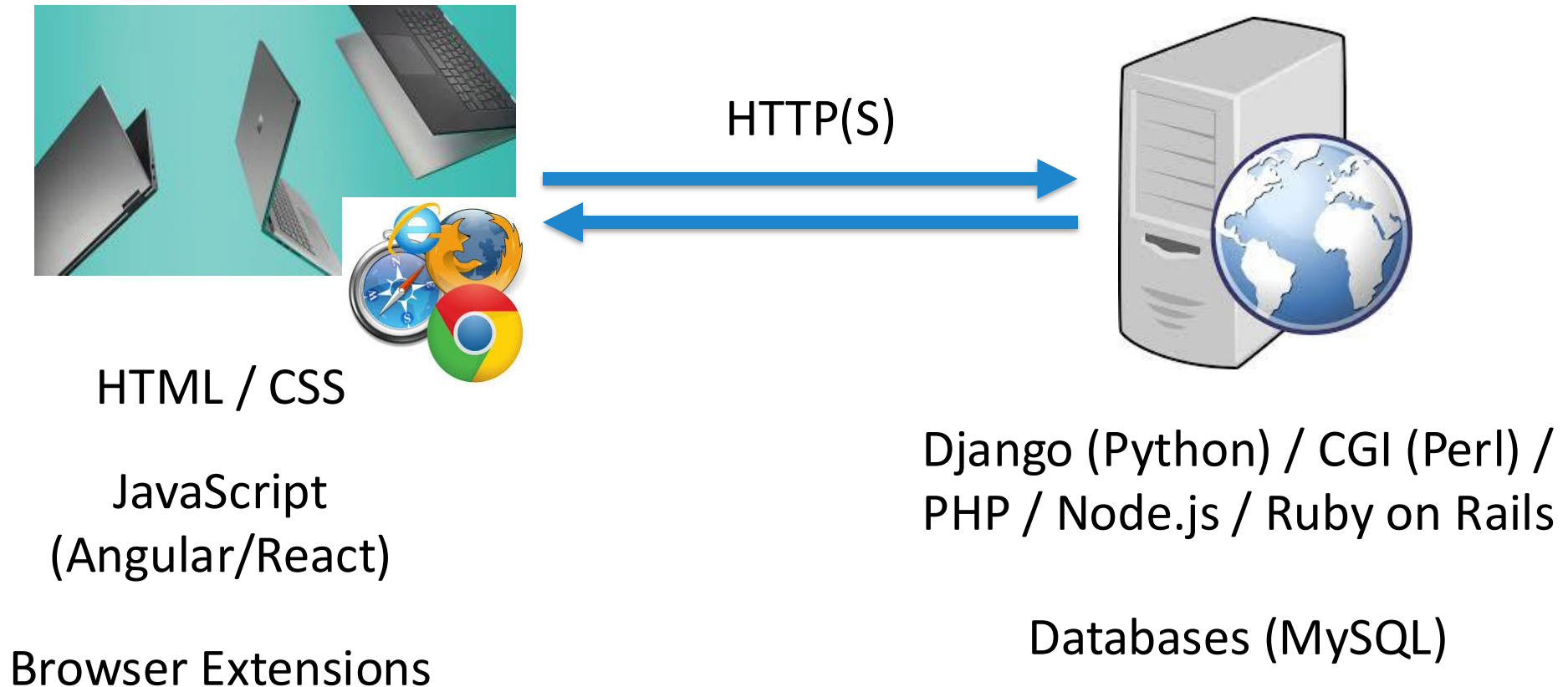
# Web Browsing



- Previously: Networking = how do route desired packets between clients  $\leftrightarrow$  servers on the Internet
- The web: structured content (desired packets) on the Internet hosted by web servers and typically accessed by web browsers (clients)

# A 10,000 Foot View of Technologies

- Where things run:



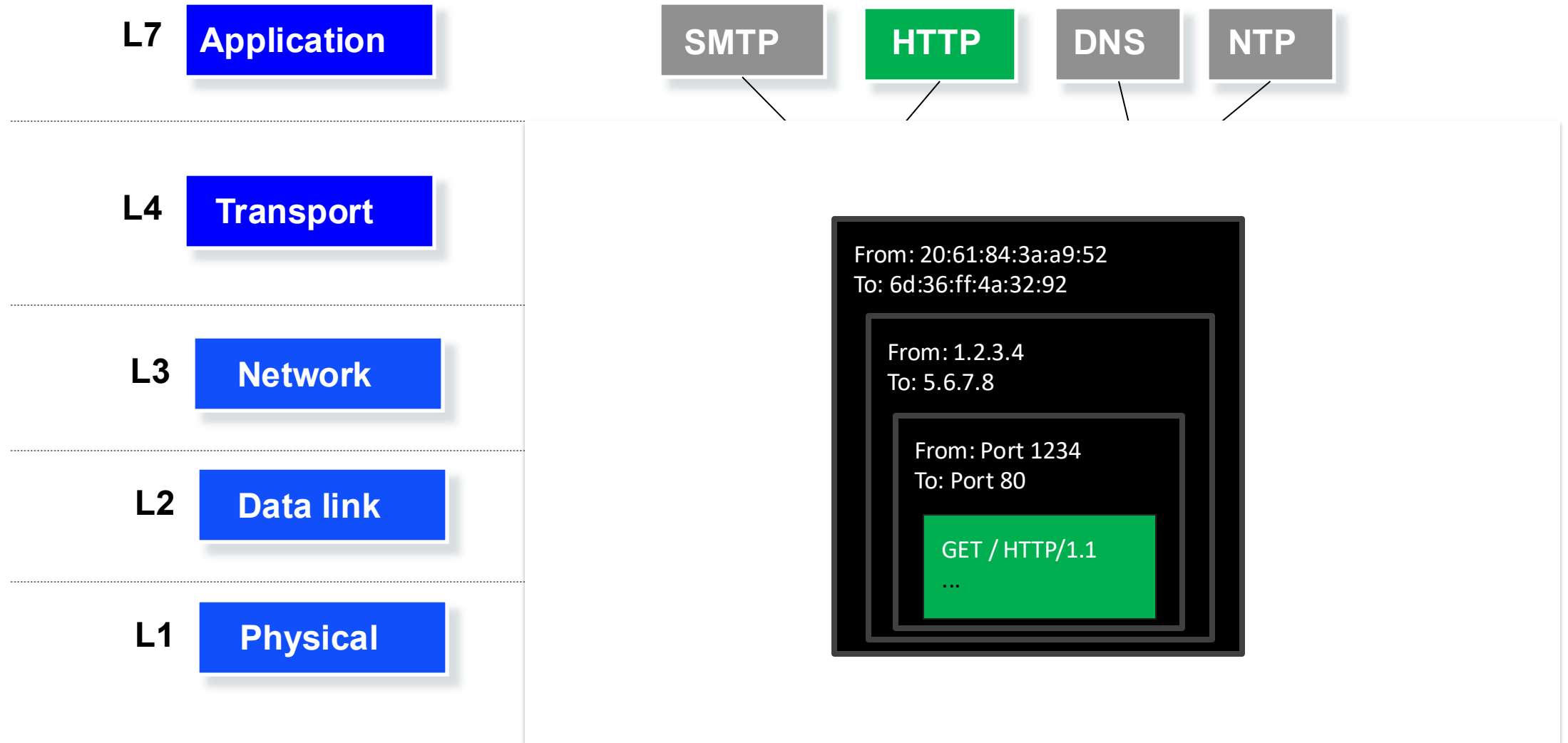
# Outline

- Web Overview
- Navigating the Web
- Webpage Structure & Contents
- Web Security Threat Models
- Same Origin Policy

# HTTP (Hypertext Transfer Protocol)

- ASCII protocol from 1989 that allows fetching resources (e.g., HTML file) from a server over TCP
  - Two messages: **request** (client -> server) and **response** (server -> client)
  - **Stateless** protocol beyond a single request + response
- Every resource has a uniform resource location (URL)

# HTTP: Application Layer





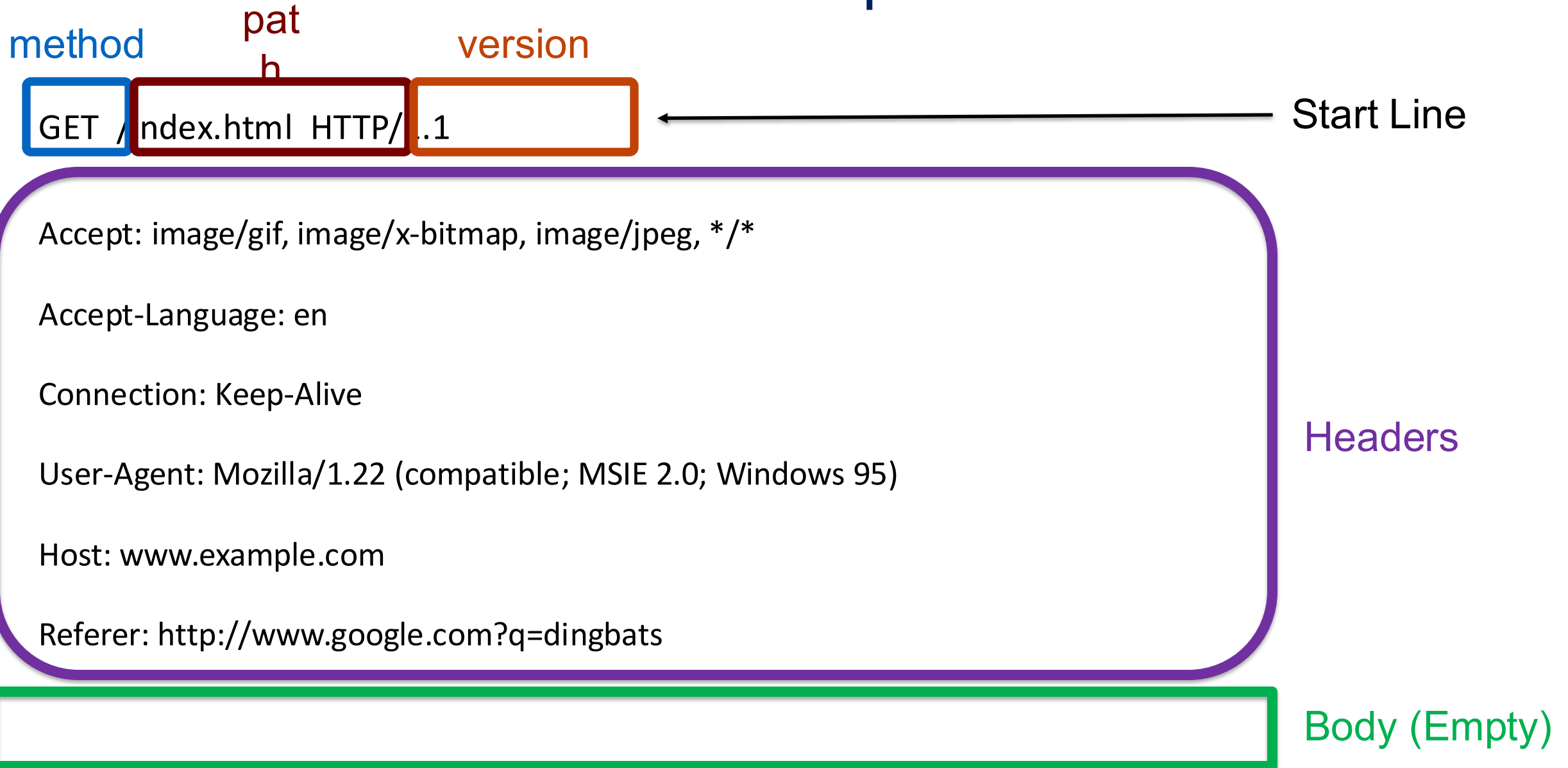
# The Anatomy of a URL (Web Resource Address)

- <https://www.uchicago.edu/fun/funthings.html?query=music&year=2024#topsection>
  - Scheme (Protocol): https
  - Hostname: www.uchicago.edu
  - Path: /fun/funthings.html
  - Parameters: (key=value pairs, follow “?” and delimited with “&”)
  - Named anchor: #topsection (used only by client/browser)

# HTTP Request: Client Msg to Server

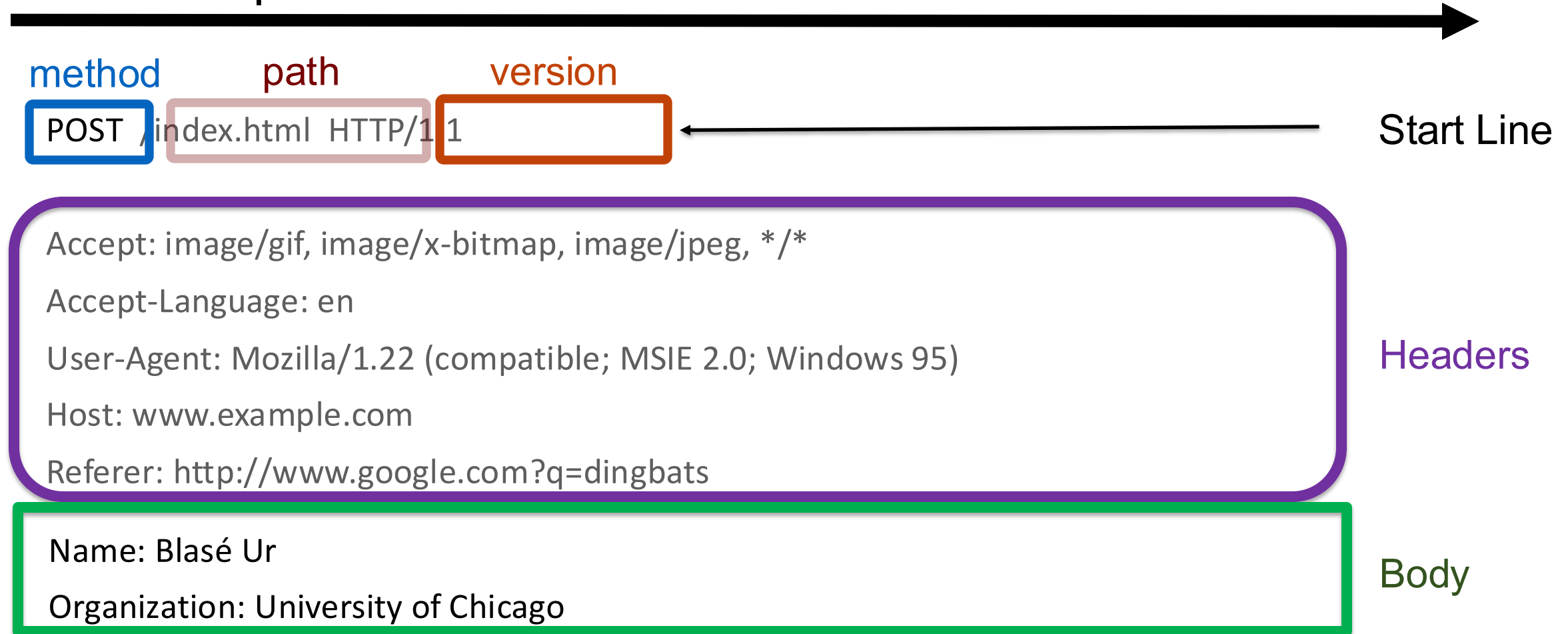
- Start line: method, target (path), protocol version
  - `GET /index.html HTTP/1.1`
  - Method: `GET`, `PUT`, `POST`, `HEAD`, `OPTIONS`
- HTTP Headers (Key: Value pairs)
  - Host, User-agent, Referer, many others
  - <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers>
- Body (not needed for certain methods, e.g., GET)
- In Firefox: F12, “Network” to see HTTP requests

# HTTP Request



# HTTP GET vs. POST

## HTTP Request



# HTTP Request Methods

- **GET:** Get the resource at the specified URL & encode data at the end of the URL (does not accept message body)
  - **POST:** Create new resource at URL with payload (body)
- 
- **PUT:** Replace target resource with request payload
  - **PATCH:** Update part of the resource
  - **DELETE:** Delete the specified URL

# HTTP Request Methods

- Not all methods are created equal — some have different security protections
- **GETs** 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 effect are **POSTs** today
  - Real method hidden in a header or request body

– 🙈♀ **Never do...**

– **GET** `http://bank.com/transfer?fromAcct=X&toAcct=Y&amount=1000`

# HTTP Response: Server Msg to Client

- Status: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Status>
  - 200 (OK)
  - 404 (not found)
  - 301 (moved permanently)
  - 302 (moved temporarily)
- HTTP Headers
- Body

# HTTP Response

HTTP Response



HTTP/1.0 200 OK

status  
code

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

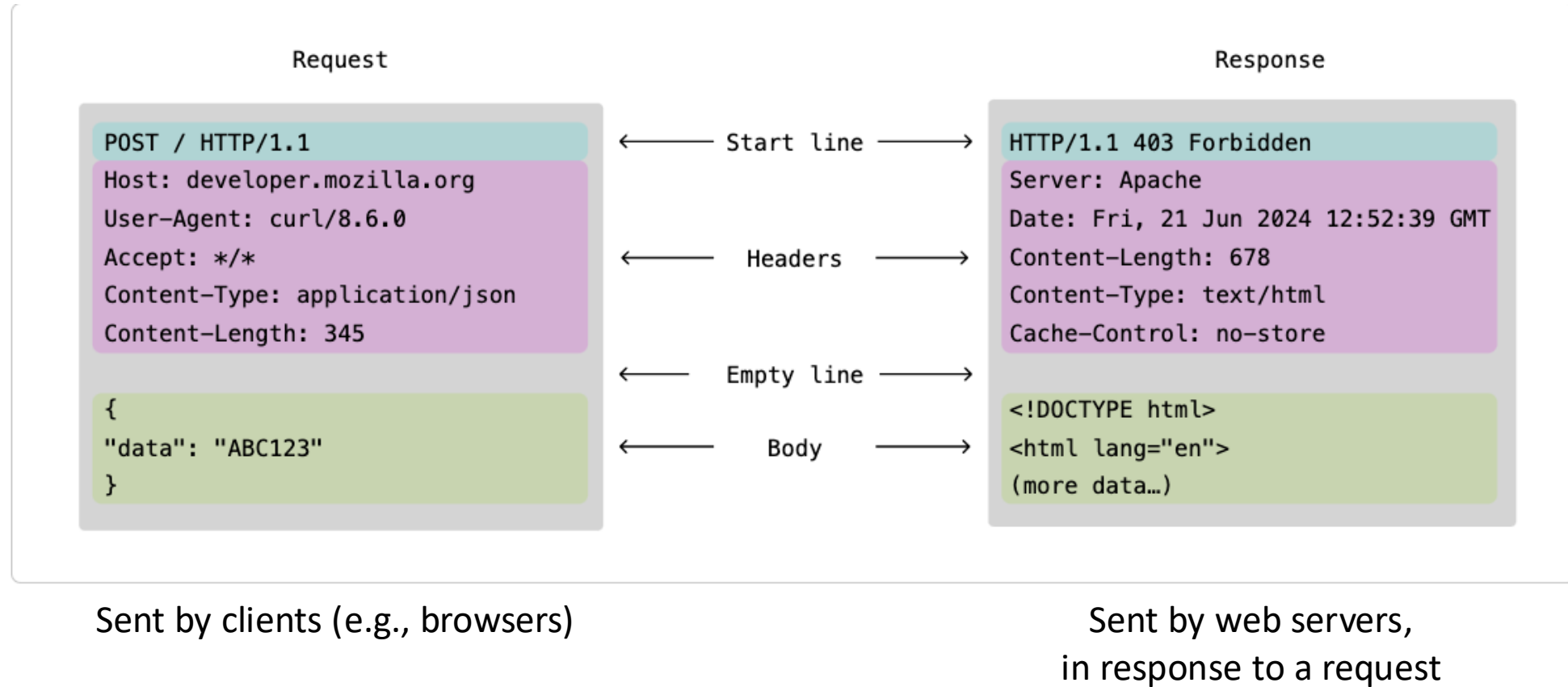
headers

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

body



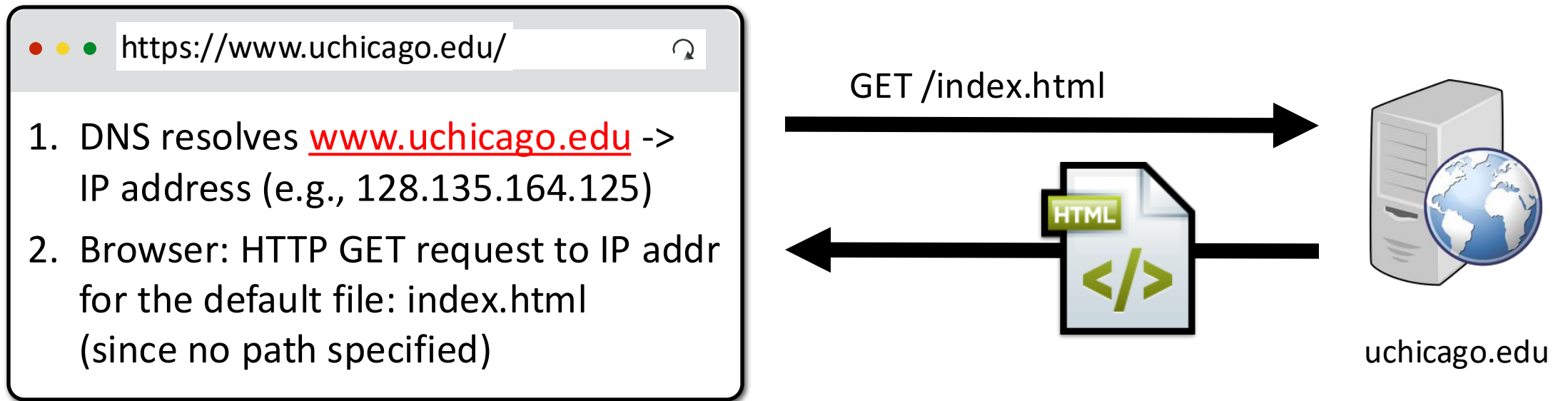
# HTTP: Request & Response



From <https://developer.mozilla.org/en-US/docs/Web/HTTP/Messages>

# HTTP → Website

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



# 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 servers send to clients

- Enables persistent state / web browsing sessions
- The client (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



# Keeping State Using Cookies

- Server Sends: Set-Cookie HTTP header
- Client Sends w/ Each Request: Cookie HTTP header
  - *Cookie: name=value; name2=value2; name3=value3*
- Cookies are **automatically sent** with all requests your web browser makes
- Cookies are *bound to an origin* (only sent to servers w/ matching origin)

# 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>

Server uses  
“Set-Cookie”  
HTTP Header

# Sending Cookies

## 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=dingbats

Cookies are  
**automatically  
sent** with all  
requests your  
browser  
makes!

# Authorization Tokens = Cookies

- You log into a website, and it presents you an authorization token (typically a hash of some secret)
- Subsequent HTTP requests automatically embed this authorization token
- Session cookies (until you close your browser) vs. persistent cookies (until the expiration date)
- View cookies: “Application” tab in Chrome developer tools, “Storage” in Firefox



# Login Session w/ Cookies

GET /loginform HTTP/1.1

cookies: []



HTTP/1.0 200 OK

cookies: []

POST /login HTTP/1.1

cookies: []



username: Blase

password: chicago4life

<html><form>...</form></html>

HTTP/1.0 200 OK

cookies: [session: e82a7b92]



<html><h1>Login Success</h1></html>

GET /account HTTP/1.1

cookies: [session: e82a7b92]



GET /img/user.jpg HTTP/1.1

cookies: [session: e82a7b92]



# HTTPS: Protecting HTTP Data Over the Network

- Simply an extension where HTTP data sent over TLS!
  - That is, TCP payload = HTTP request and response are encrypted
- Which CAs (certificate authorities) does your browser trust?
  - Firefox: Options → Privacy & Security → (all the way at the bottom) View Certificates

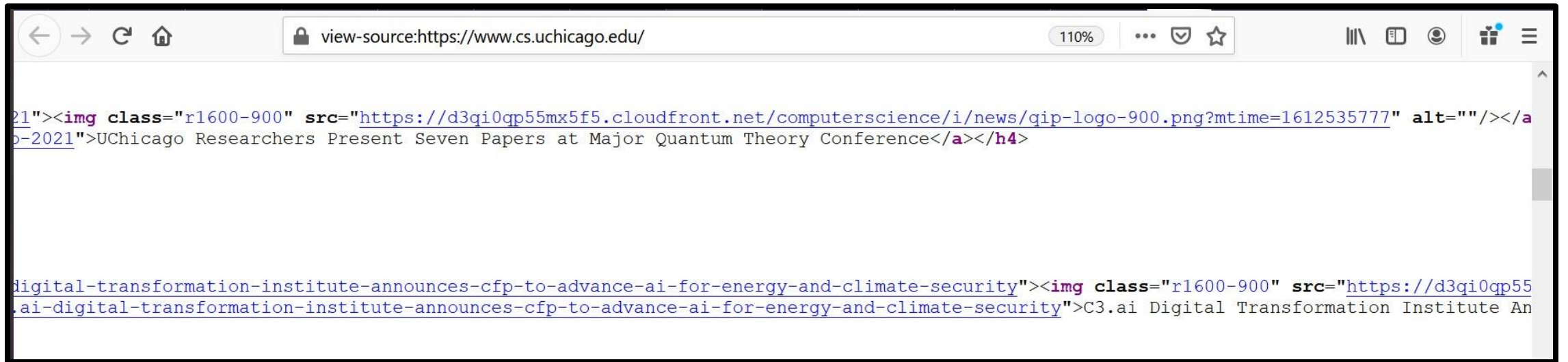
# Outline

- Web Overview
- Navigating the Web
- Structure & Contents of Webpages
- Web Security Threat Models
- Same Origin Policy

# The Anatomy of a Webpage

- HTML (hypertext markup language)
  - Language to create structured documents (webpages)
  - Uses tags `<>` to define elements on the page
    - All sorts of formatting tags: `<div><p>Hi</p></div>` `<br />`
    - Links: `<a href="blaseur.com">Click here</a>`
    - Pictures: ``
    - Forms
    - Audio/video

# The Anatomy of a Webpage



view-source:https://www.cs.uchicago.edu/

# CSS (Cascading Style Sheets)

Language used for describing the presentation (“style”) of a document

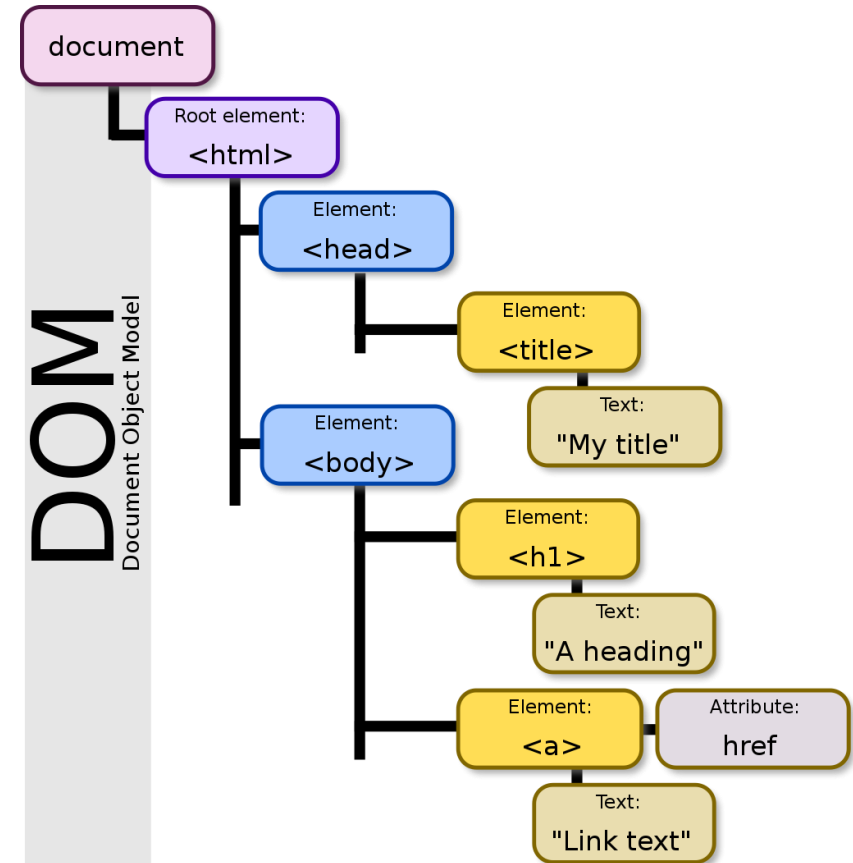
```
index.css
```

```
p.serif {  
  font-family: "Times New Roman", Times, serif;  
}  
p.sansserif {  
  font-family: Arial, Helvetica, sans-serif;  
}
```

# DOM (document object model)

Cross-platform model for representing and interacting with objects in HTML

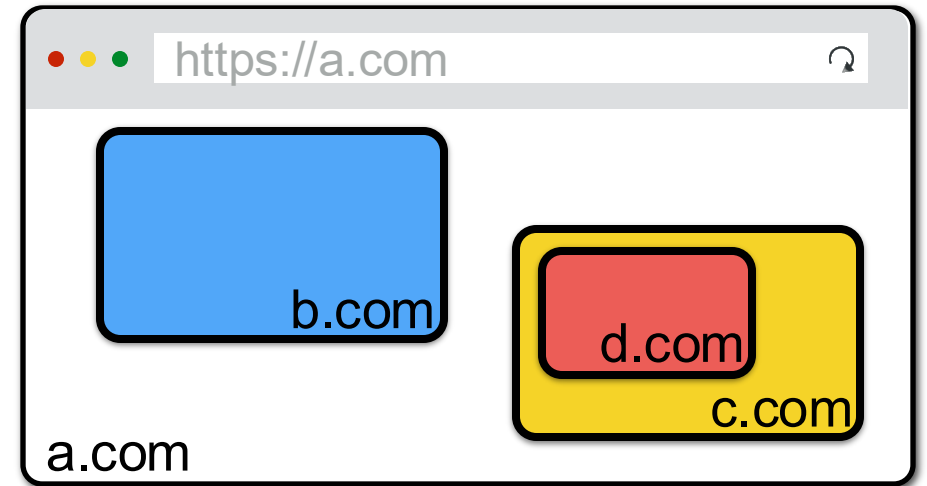
- Represent a document (webpage) as a tree object w/ properties (HTML elements)
- Browser takes HTML -> structured data (DOM)



# Inline Frames (iFrames)

- Beyond loading individual resources, websites can also load other *websites* within their window
  - iFrame: floating inline frame
  - Allows delegating screen area to content from another source (e.g., ads)
  - Frame isolation: inner & outer pages cannot modify each other's content

```
<iframe src="URL"></iframe>
```





# Creating Interactive Pages

- JavaScript!
  - The core idea: Let's run code on the client's computer
- Incredibly powerful scripting language that's interpreted/compiled & run inside of the browser:
  - Math, variables, control structures
  - Modify the DOM
  - Access browser data & hardware
  - Issue network requests for data (e.g., through AJAX)
  - Can be multi-threaded (web workers)

# Common Javascript Libraries

- JQuery (easier to specify access to DOM)
  - `$(".test").hide()` hides all elements with class="test"
- JQueryUI
- Bootstrap
- Angular / React
- Google Analytics (*sigh*)

# Importing Javascript Libraries

```
673
674         </ul>
675     </div>
676 </div>
677 </div>
678 <div class="row">
679     <div class="footer_copy">
680         <p>&#169; 2021 <span class="url fn org">The University of Chicago</span></p>
681     </div>
682 </div>
683 </div>
684 <a id="back-to-top" href="#" class="back-to-top" role="button"></a>
685 </footer>
686
687 <script defer src="/js/libs/modernizr.js?updated=20191205080224"></script>
688 <script src="https://ajax.googleapis.com/ajax/libs/jquery/2.1.4/jquery.min.js"></script>
689 <script src="https://ajax.googleapis.com/ajax/libs/jqueryui/1.11.4/jquery-ui.min.js"></script>
690 <script>window.jQuery || document.write('<script src="/js/libs/jquery/2.1.4/jquery.min.js"></script><script src="/js/libs
691 <script defer src="/js/core-min.js?updated=20191205080225"></script>
692
693 <!--[if lte IE 8]><script src="/js/libs/selectivizr.js"></script><![endif]-->
694 <!--[if lte IE 9]><script src="/js/ie_fixes/symbolset.js"></script><![endif]-->
695 <!--<script src="https://cdnjs.cloudflare.com/ajax/libs/jquery.lifestream/0.3.7/jquery.lifestream.min.js"></script> -->
696
697
698
699
700
701 <script async src="https://www.googletagmanager.com/gtag/js?id=UA-3572058-1"></script>
702 <script>window.dataLayer = window.dataLayer || [];function gtag(){dataLayer.push(arguments);}gtag('js', new Date());
703 gtag('config', 'UA-3572058-1');gtag('config', 'UA-187440939-1');</script>
704
705 </body>
706 </html>
707
```

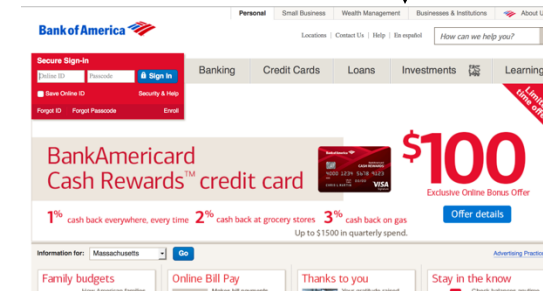
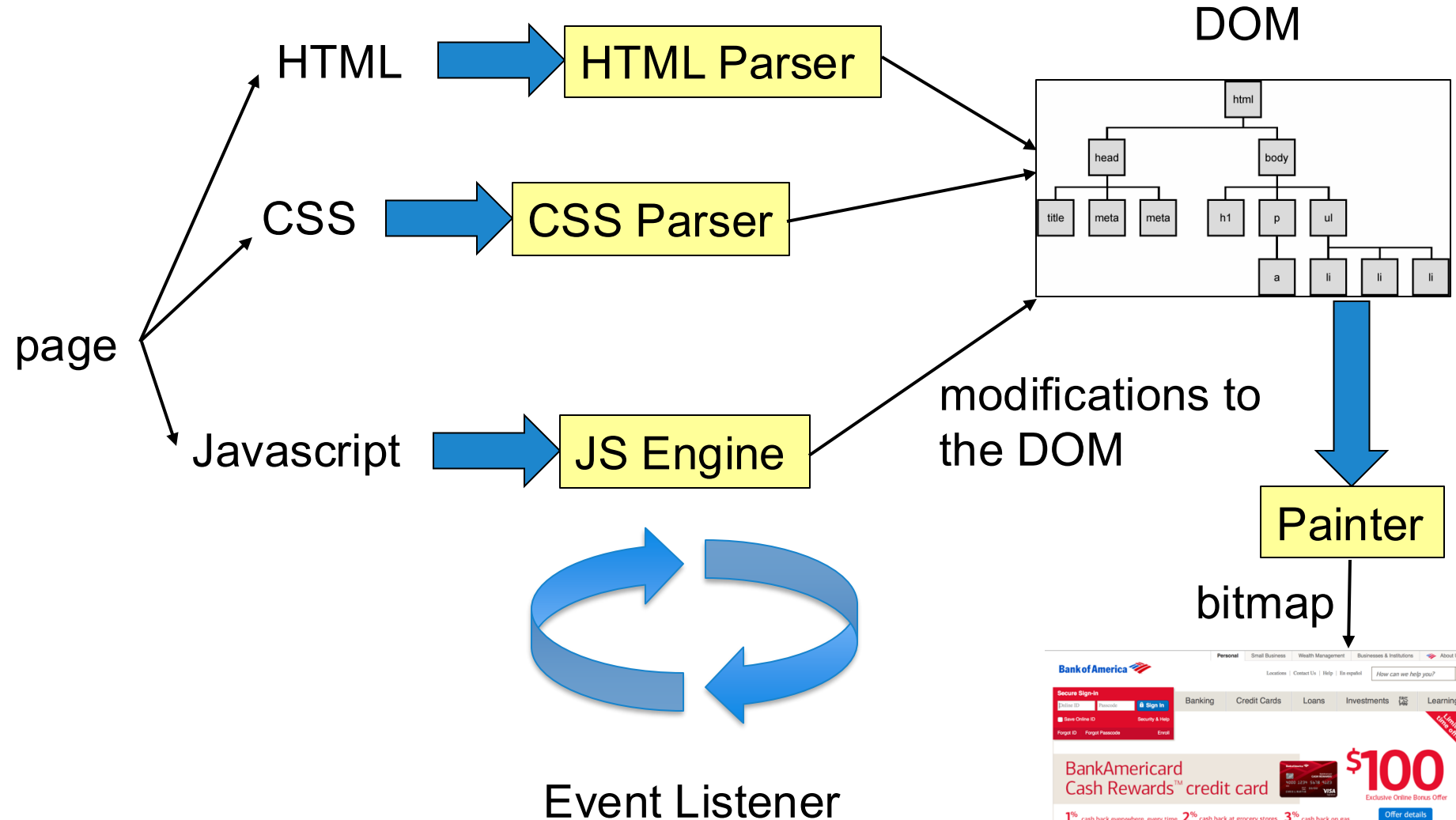
# Sub-resource Integrity

- Sub-resource integrity (SRI): Validate that the resource your website loads on clients matches what you expect
  - New “integrity” attribute for certain HTML tags
- Website creator computes hash of expected resource, and adds SRI integrity attribute to resource tags in their HTML
  - `cat FILENAME.js | openssl dgst -sha384 -binary | openssl base64 -A`
  - `<script src=“https://example.com/FILENAME.js” integrity=“sha384-oqVuAfXRKap...x4JwY8wC”></script>`

# Basic Browser Execution Model (Page Rendering)

- 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)

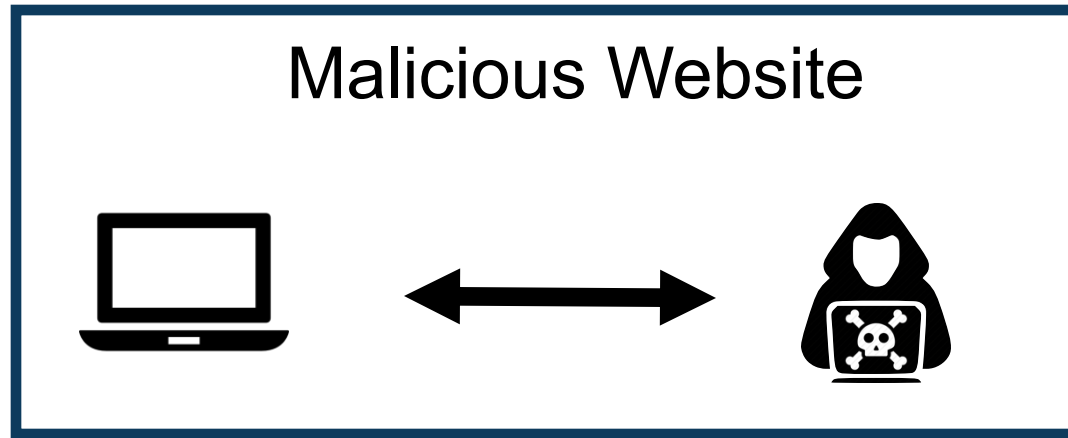
# Page Rendering



# Outline

- Web Overview
- Navigating the Web
- Webpage Structure & Contents
- Web Security Threat Models
- Same Origin Policy

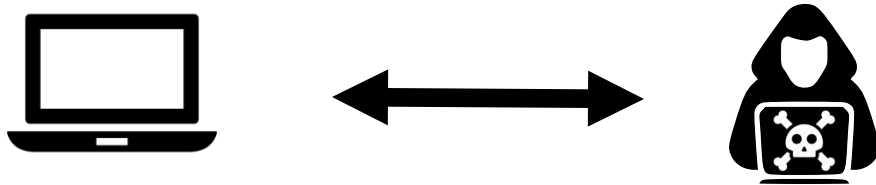
# Web Attack Models



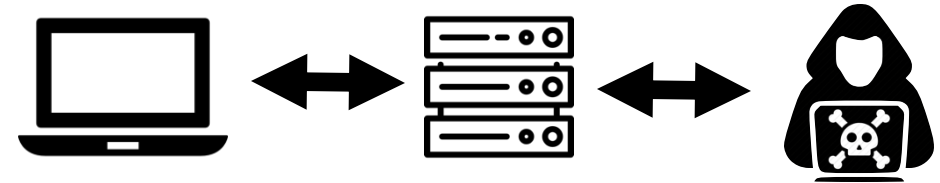


# Web Attack Models

Malicious Website



Malicious External Resource



# Desirable security goals

- **Integrity**: a malicious website should not be able to tamper with integrity of our computers or our information on other web sites
- **Confidentiality**: malicious web sites should not be able to learn confidential information from our computers or other web sites
- **Privacy**: malicious web sites should not be able to spy on us or our online activities
- **Availability**: malicious parties should not be able to keep us from accessing our web resources

# Security on the web

- Risk #1: we don't want a malicious site to be able to trash files/programs on our computers
  - Browsing to [awesomevids.com](#) (or [evil.com](#)) should not infect our computers with malware (malicious software), read or write files on our computers, etc.

# Security on the web

- Risk #1: we don't want a malicious site to be able to trash files/programs on our computers
  - Browsing to [awesomevids.com](#) (or [evil.com](#)) should not infect our computers with malware, read or write files on our computers, etc.
- Defenses: Javascript is [sandboxed](#); try to avoid security bugs in browser code; privilege separation; automatic updates.

# Security on the web

- Risk #2: we don't want a malicious site to be able to spy on or tamper with our information or interactions with other websites
  - Browsing to **evil.com** should not let **evil.com** spy on our emails in Gmail or buy stuff with our Amazon accounts

# Security on the web

- Risk #2: we don't want a malicious site to be able to spy on or tamper with our information or interactions with other websites
  - Browsing to **evil.com** should not let **evil.com** spy on our emails in Gmail or buy stuff with our Amazon accounts
- Defense: the *same-origin policy*
  - A security policy grafted on after-the-fact, and enforced by web browsers

# Security on the web

- Risk #3: we want data stored on a web server to be protected from unauthorized access

# Security on the web

- Risk #3: we want data stored on a web server to be protected from unauthorized access
- Defense: server-side security  
(e.g., web-app security, as well as access control, software security, firewalls, etc.)



# Outline

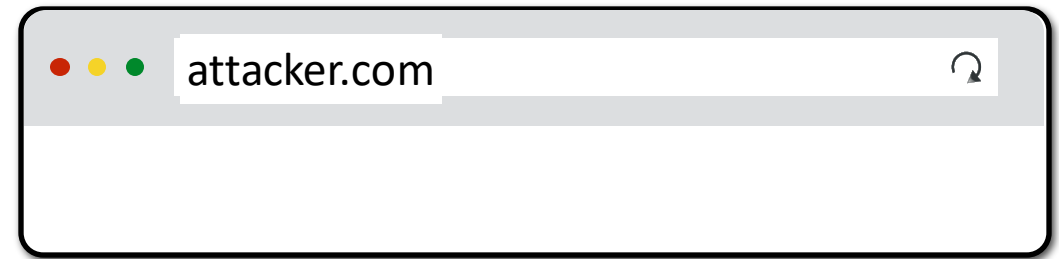
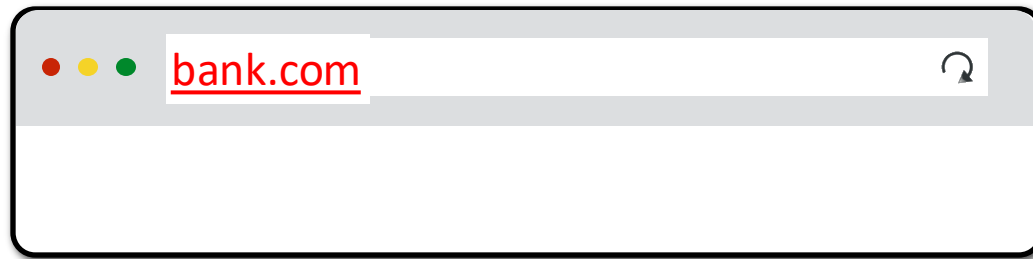
- Web Overview
- Navigating the Web
- Webpage Structure & Contents
- Web Security Threat Models
- Same Origin Policy

# Same-Origin Policy (SOP): Core Web Defense

- **Goal:** prevent one website from tampering with other unrelated websites (malicious DOM access)
  - Enforced by the web browser
- **Origin [DOM]:** exact triplet of (URI scheme, host name, port)
- **SOP:** Content, such as scripts, from different origins cannot interact with each other
  - Javascript inherits origin of the frame that loaded it

# Bounding Origins — Windows

- Every Window and Frame has an origin
- Origins are blocked from accessing other origin's objects



`attacker.com` cannot...

- *read or write* content from **bank.com** tab
- *read or write* **bank.com**'s cookies
- *detect* that the other tab has **bank.com** loaded

# Assessing SOP

Originating document	Accessed document
<a href="http://wikipedia.org/a/">http://wikipedia.org/a/</a>	<a href="http://wikipedia.org/b/">http://wikipedia.org/b/</a>
<a href="http://wikipedia.org/">http://wikipedia.org/</a>	<a href="http://www.wikipedia.org/">http://www.wikipedia.org/</a>
<a href="http://wikipedia.org/">http://wikipedia.org/</a>	<a href="https://wikipedia.org/">https://wikipedia.org/</a>
<a href="http://wikipedia.org:81/">http://wikipedia.org:81/</a>	<a href="http://wikipedia.org:82/">http://wikipedia.org:82/</a>
<a href="http://wikipedia.org:81/">http://wikipedia.org:81/</a>	<a href="http://wikipedia.org/">http://wikipedia.org/</a>

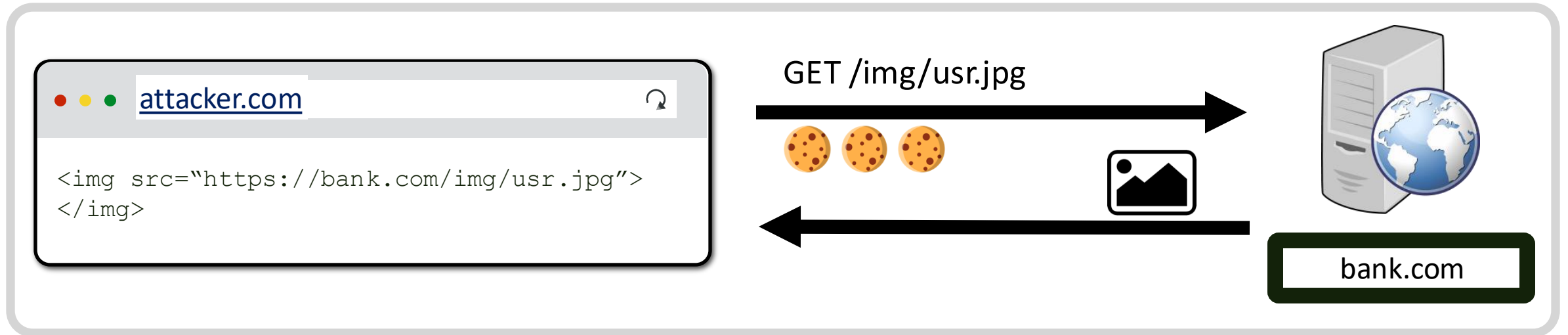


except



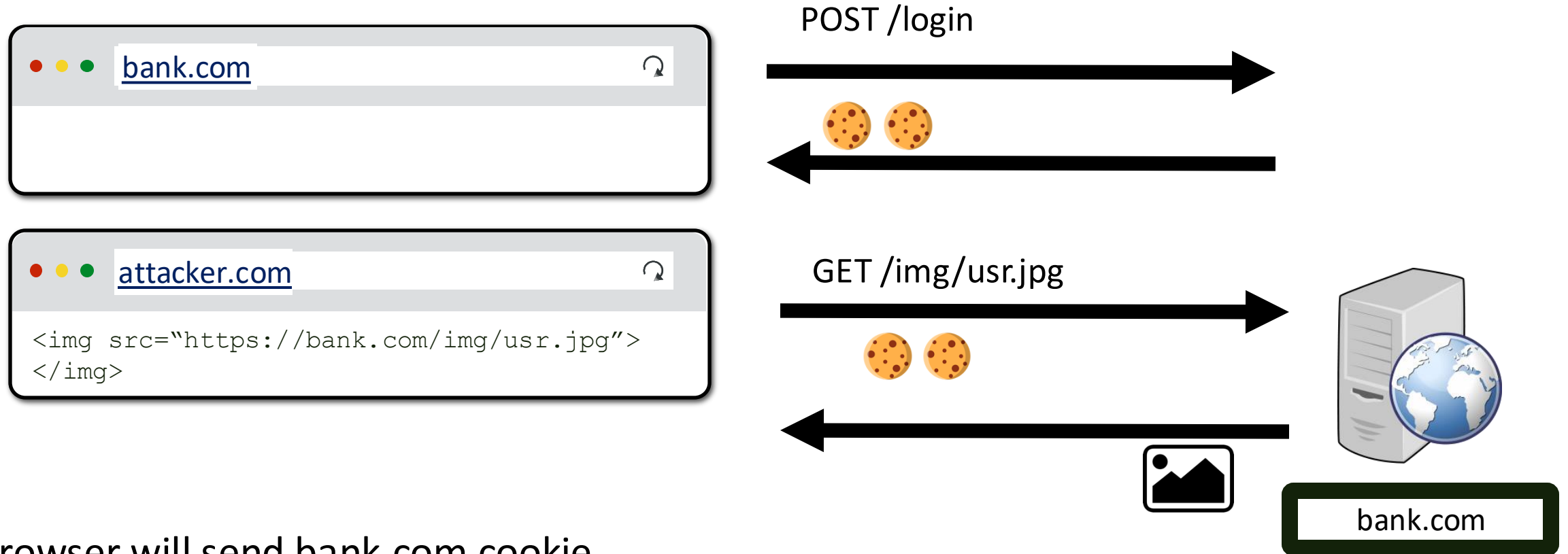
# SOP for HTTP Responses

- Pages can *make requests* across origins



SOP does not prevent attacker.com from making the HTTP request to the server

# Origins and Cookies

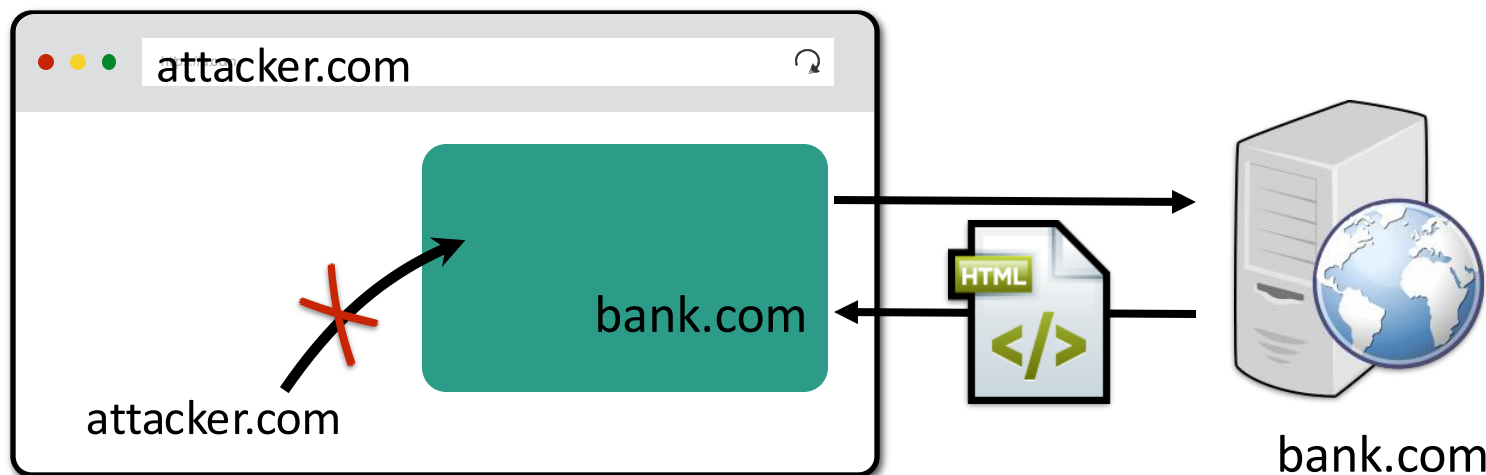


Browser will send bank.com cookie

SOP blocks attacker.com from inspecting bank.com's image and cookie

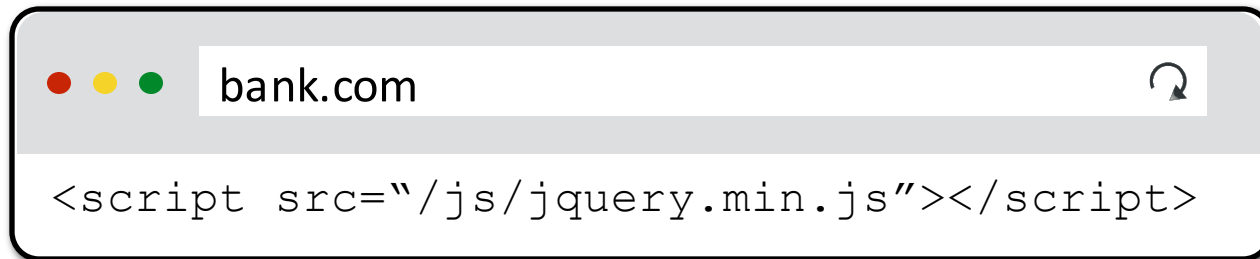
# SOP for Other HTTP Resources

- **Images:** Browser renders cross-origin images, but SOP prevents page from inspecting individual pixels. Can check size and if loaded successfully.
- **CSS, Fonts:** Similar — can load and use, but not directly inspect
- **Frames:** Can load cross-origin HTML in frames, but cannot inspect or modify the frame content. Cannot check success for Frames.



# Script Execution

Scripts can be loaded from other origins. Scripts execute with the privileges of their parent frame/window's origin. Parent can call functions in script.



**You can load library from CDN and use it to alter your page**



**If you load a malicious library, it can also steal your data (e.g., cookies)**



# Relaxing SOP

# Frames - Domain Relaxation



These frames  
cannot access  
each other's DOM!

# Domain Relaxation

You can change your `document.domain` to be a **super-domain**

`a.domain.com` → `domain.com`      **OK**

`b.domain.com` → `domain.com`      **OK**

`a.domain.com` → `com`      **NOT OK**

`a.doin.co.uk` → `co.uk`      **NOT OK**

# Cross-Origin Resource Sharing (CORS)

Let's say you have a web application running at **app.company.com** and you want to access JSON data by making requests to **api.company-internal.com**.


By default, this wouldn't be possible — app.company.com and api.company-internal.com are different origins!


# CORS (Relaxes SOP)

- Cross-Origin Resource Sharing
  - HTTP Headers that specify when other origins can make a request for data on a different origin
- Server on “a.com” can use CORS headers in its HTTP response:
  - **Access-Control-Allow-Origin: https://b.com**
  - **Access-Control-Allow-Methods: POST, GET, OPTIONS**
  - **Access-Control-Allow-Headers: X-PINGOTHER, Content-Type**
  - ...

# When CORS is Needed

## What requests use CORS?

This [cross-origin sharing standard](#)  can enable cross-origin HTTP requests for:

- Invocations of the [XMLHttpRequest](#) or [Fetch APIs](#), as discussed above.
- Web Fonts (for cross-domain font usage in `@font-face` within CSS),  
[so that servers can deploy TrueType fonts that can only be loaded cross-origin and used by web sites that are permitted to do so.](#) 
- [WebGL textures](#).
- Images/video frames drawn to a canvas using [drawImage\(\)](#).
- [CSS Shapes from images](#).

This is a general article about Cross-Origin Resource Sharing and includes a discussion of the necessary HTTP headers.

From <https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS>