Web Security I

CSE 565: Fall 2024
Computer Security

Xiangyu Guo (xiangyug@buffalo.edu)

University at Buffalo

Acknowledgement

- We don’t claim any originality of the slides. The content is developed heavily based on
  - Slides from Prof Dan Boneh’s lecture on Computer Security (https://cs155.stanford.edu/syllabus.html)
  - Slides from Prof Ziming Zhao’s past offering of CSE565 (https://zzm7000.github.io/teaching/2023springcse410565/index.html)

Announcement

- In-Class Midterm on **Oct 17**.
- HW2 & Proj 2 will be released in Thu 10/03

Review of last week

- Access Control
  - Purpose: limit access for *authenticated* users
  - Access Control Matrix: ACL vs Capabilities
  - POSIX File Permission Mode

- Major Access Control Models:
  - Discretionary (DAC): OS.
  - Mandatory (MAC): Military, Government. OS.
  - Role-Based (RBAC): Corporate management. DBMS.
  - Attribute-Based (ABAC): Most complex and covers all previous. Web/Cloud Service. Finantial/Healthcare service.

Today's topic

Today

- Web Security Model
- Basics of HTTP
- Cookies & Sessions
- The Same-Origin Policy

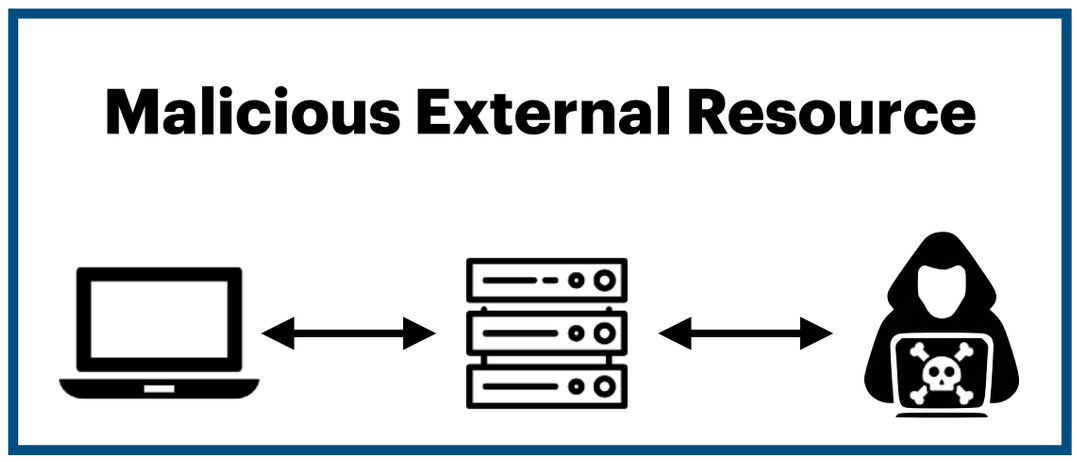
Later Lectures: Web Attacks

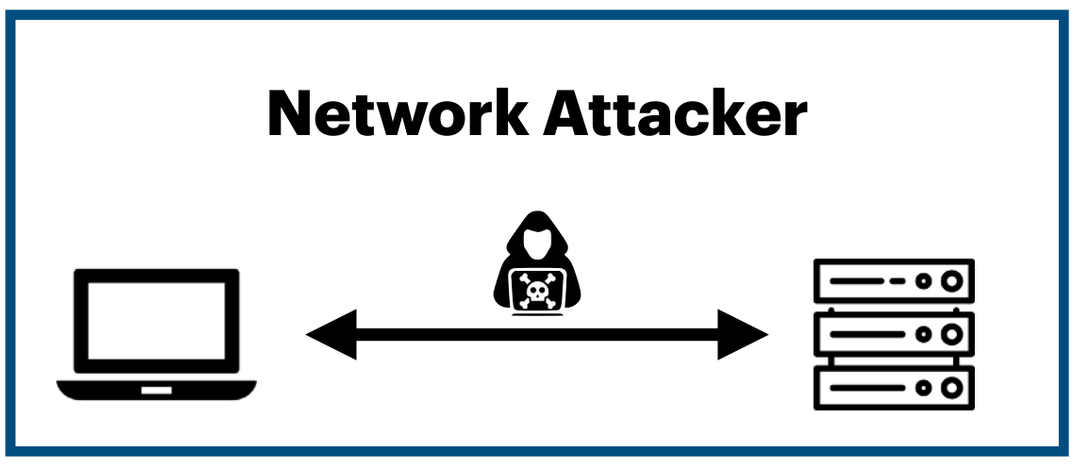
- Cross-Site Request Forgery (CSRF)
- Cross-Site Scripting (XSS)
- Injection
  - Path traversal
  - Command Injection
  - SQL Injection

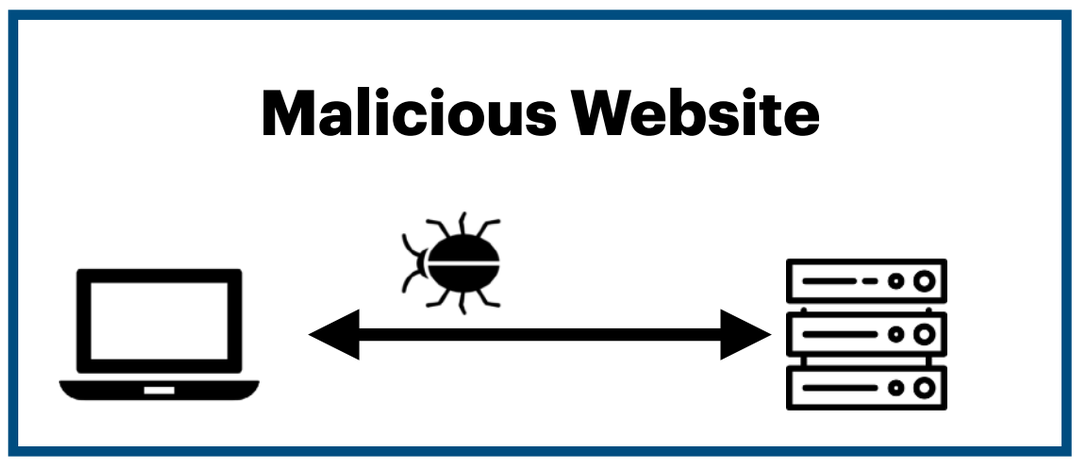
Web Security Goals

- **Safely** browse the web in the face of attackers
- Visit websites (including malicious ones!) without incurring harm
  - **Site A** cannot steal data from your device, install malware, access camera, etc.
  - **Site A** cannot affect session on **Site B** or eavesdrop on **Site B**
- Support secure high-performance web apps (e.g., Google Meet)

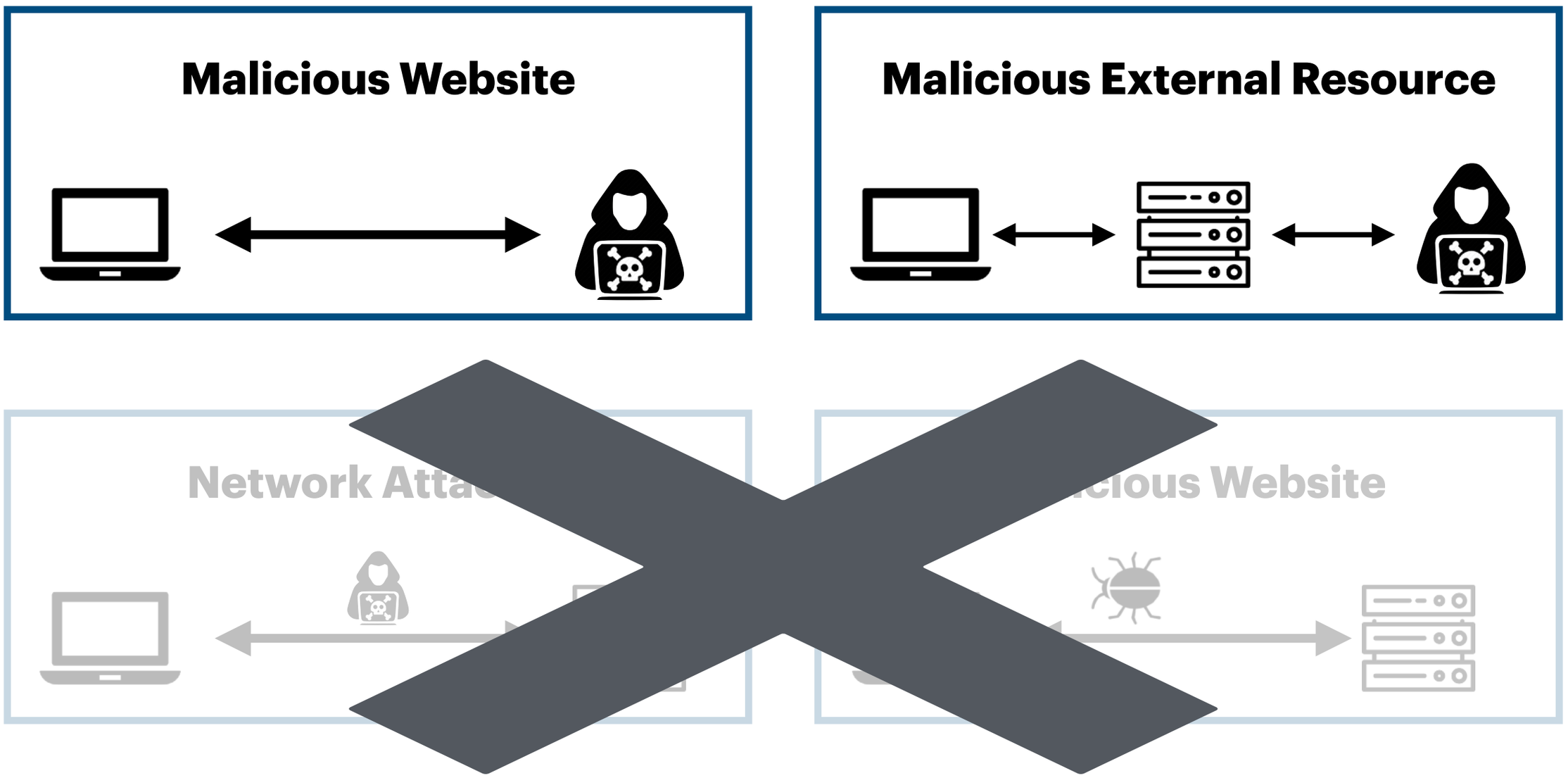
Web Attack Models







Web Attack Models



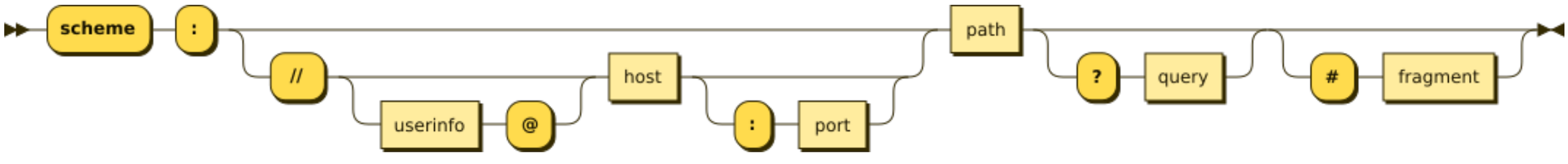
HTTP Basics

HTTP Protocol

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

http://example.com:80/slides?slide=08#web

scheme **domain** **port** **path** **query** **fragment**



HTTP Request

method **path** **version**

GET /index.html HTTP/1.1

Accept: image/gif, image/x-bitmap, image/jpeg, */*

Accept-Language: en-US

Connection: Keep-Alive

User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)

Host: www.example.com

Referer: http://www.google.com?q=buffalo+cse

headers

body (empty)

HTTP Response

HTTP / 1.0 200 OK

status code

Date: Tue, 01 Oct 2024 14:48:42 GMT

Server: Microsoft-Internet-Information-Server/5.0

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

Last-Modified: Thu, 17 Oct 2019 07:18:26 GMT

Content-Length: 648

headers

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

body

HTTP Request: GET vs. POST

method **path** **version**

POST /index.html HTTP/1.1

Accept: image/gif, image/x-bitmap, image/jpeg, */*

Accept-Language: en-US

User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)

Host: www.example.com

Referer: http://www.google.com?q=buffalo+cse

headers

Name: Alice

Organization: University at Buffalo

body

HTTP Methods

- **GET**: Get the resource at the specified URL (does not accept message body)
- **POST**: Create new resource at URL with payload

major ones

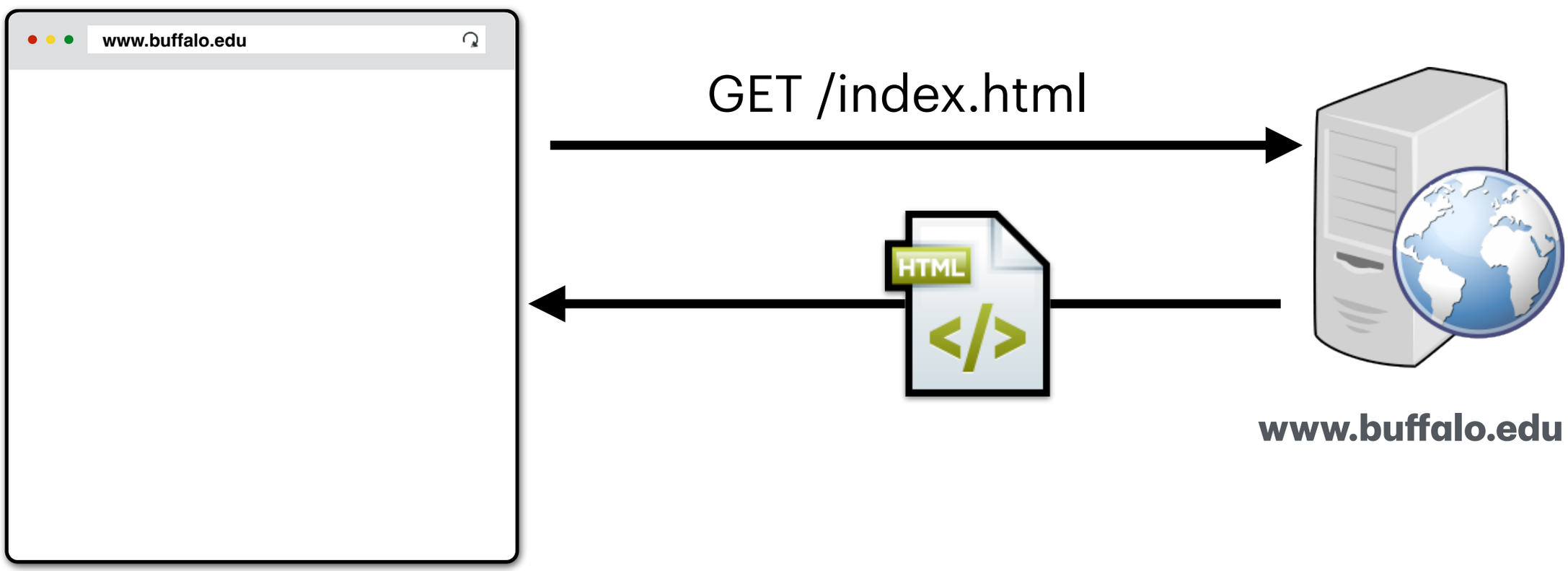
- **PUT**: Replace target resource with request payload
- **PATCH**: Update part of the resource
- **DELETE**: Delete the specified URL
- **HEAD**, **OPTIONS**, **CONNECT**, ...

HTTP Methods

- Not all methods are created equal — some have different security protections
- **GET**s should not change server state;
  - Idempotent & Cacheable
  - However 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

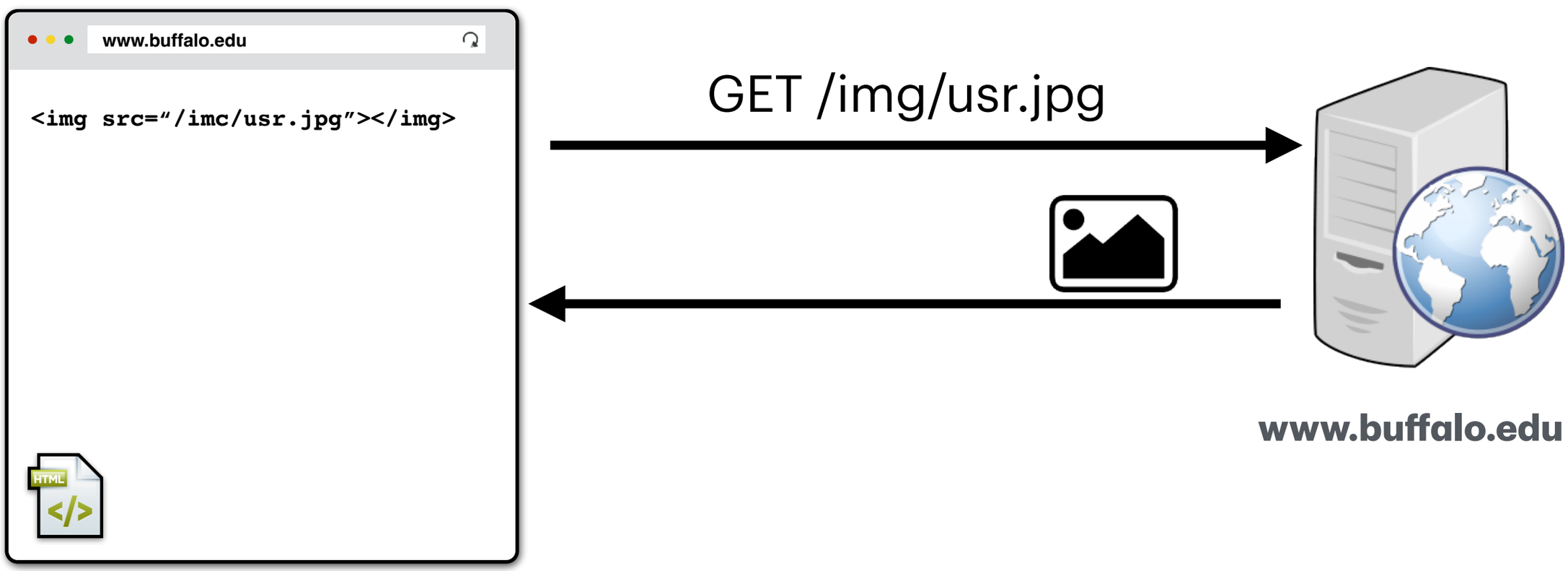
HTTP ⟹ Website

Browser loads a website by sending a **GET** request to the website



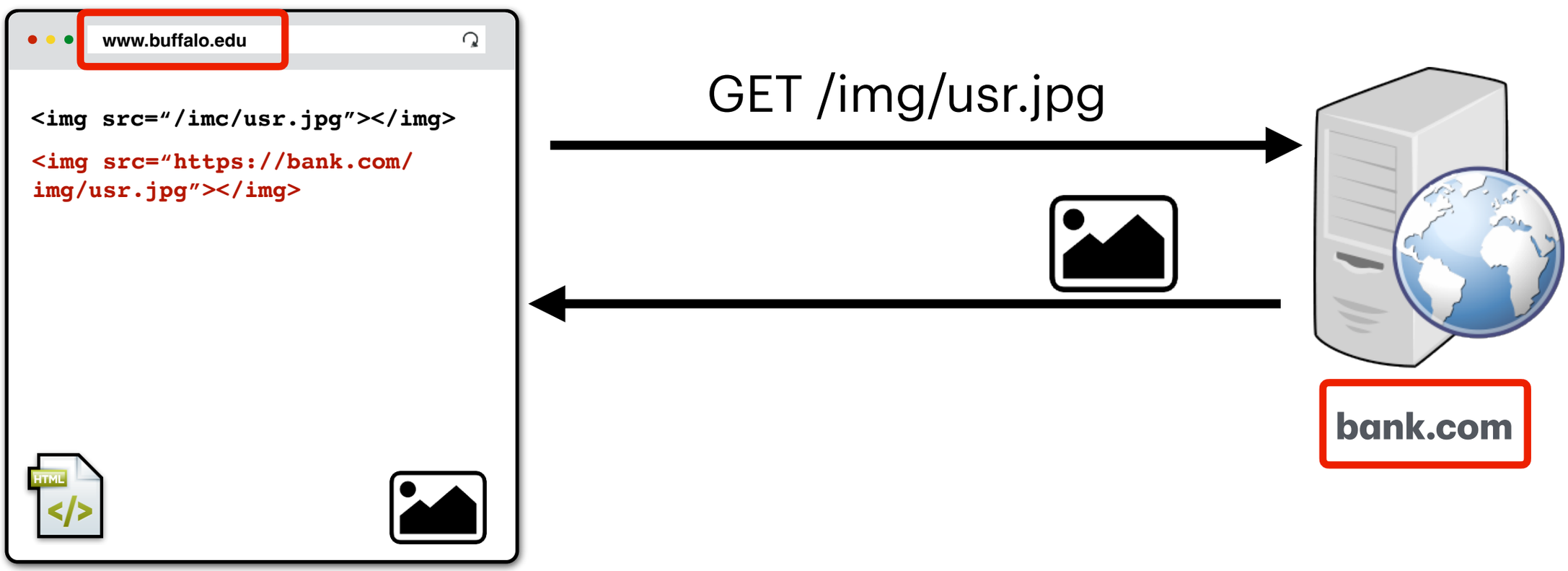
Loading Resources

- Root HTML page can include additional resources like images, videos, fonts
- After parsing page HTML, your browser requests those additional resources



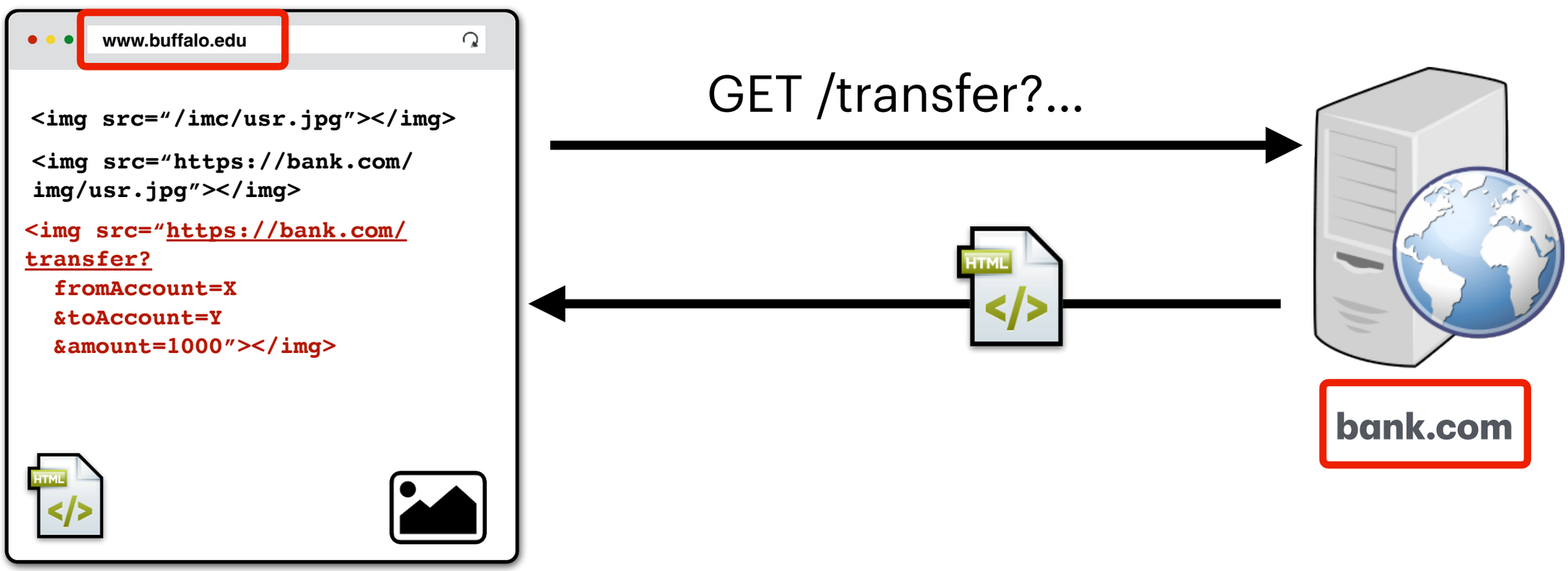
External Resources

- There are no restrictions on where you can load resources like images
- Nothing prevents you from including images on a *different* domain



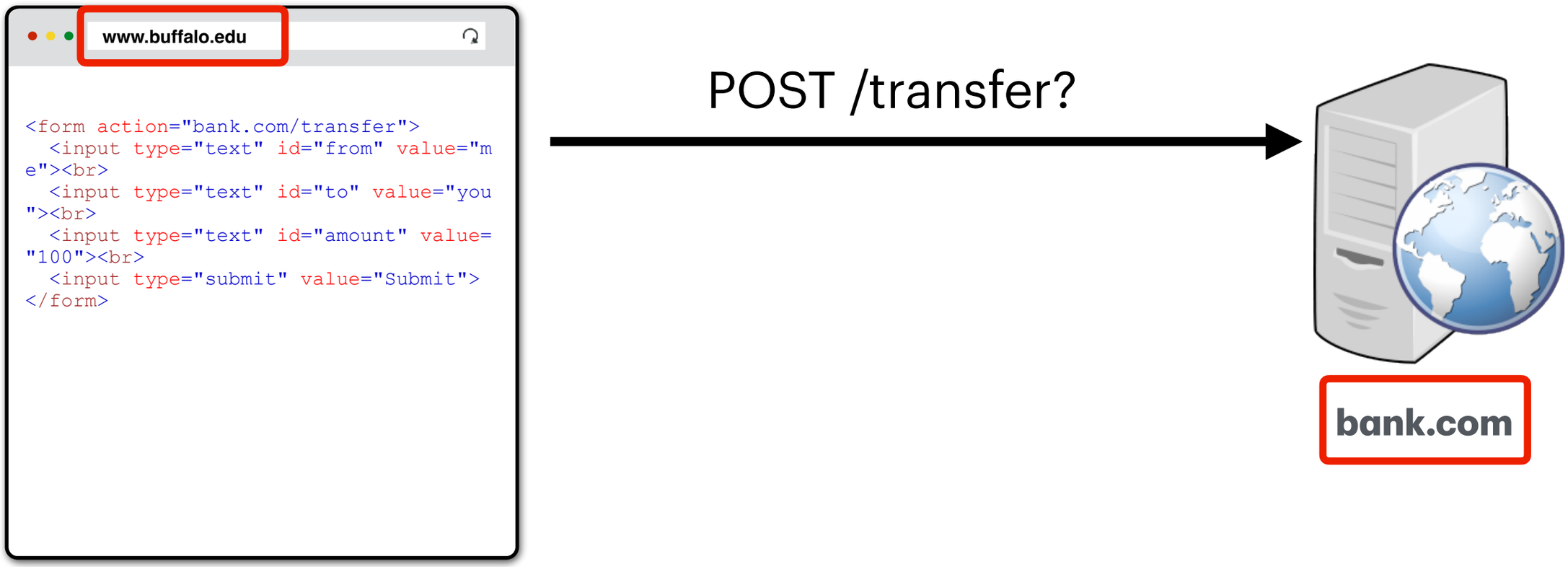
Client does not verify resource

- The browser doesn't know what will be returned when they make a request to a web server!



Not only GETs!

- You can also submit (**POST**) forms to any URL similar to how you can load resources



Javascript

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

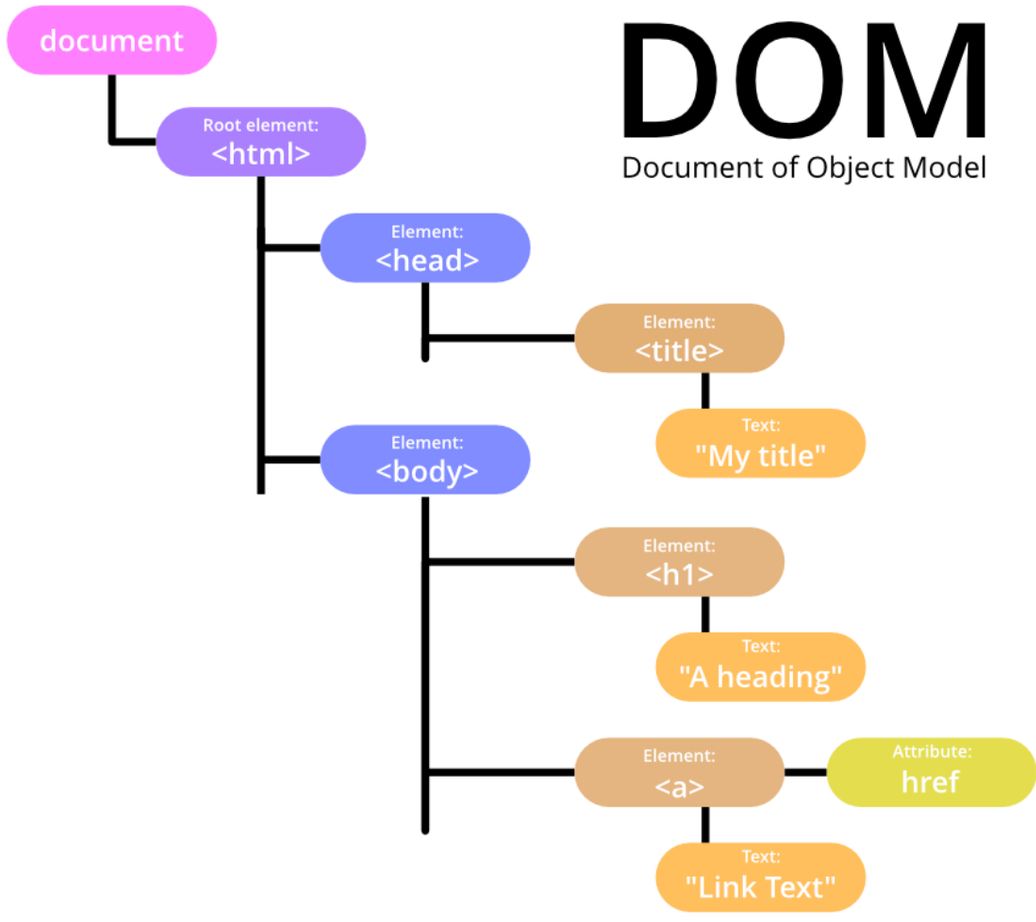
- Today, websites also deliver scripts to be run inside of the browser

```
<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)

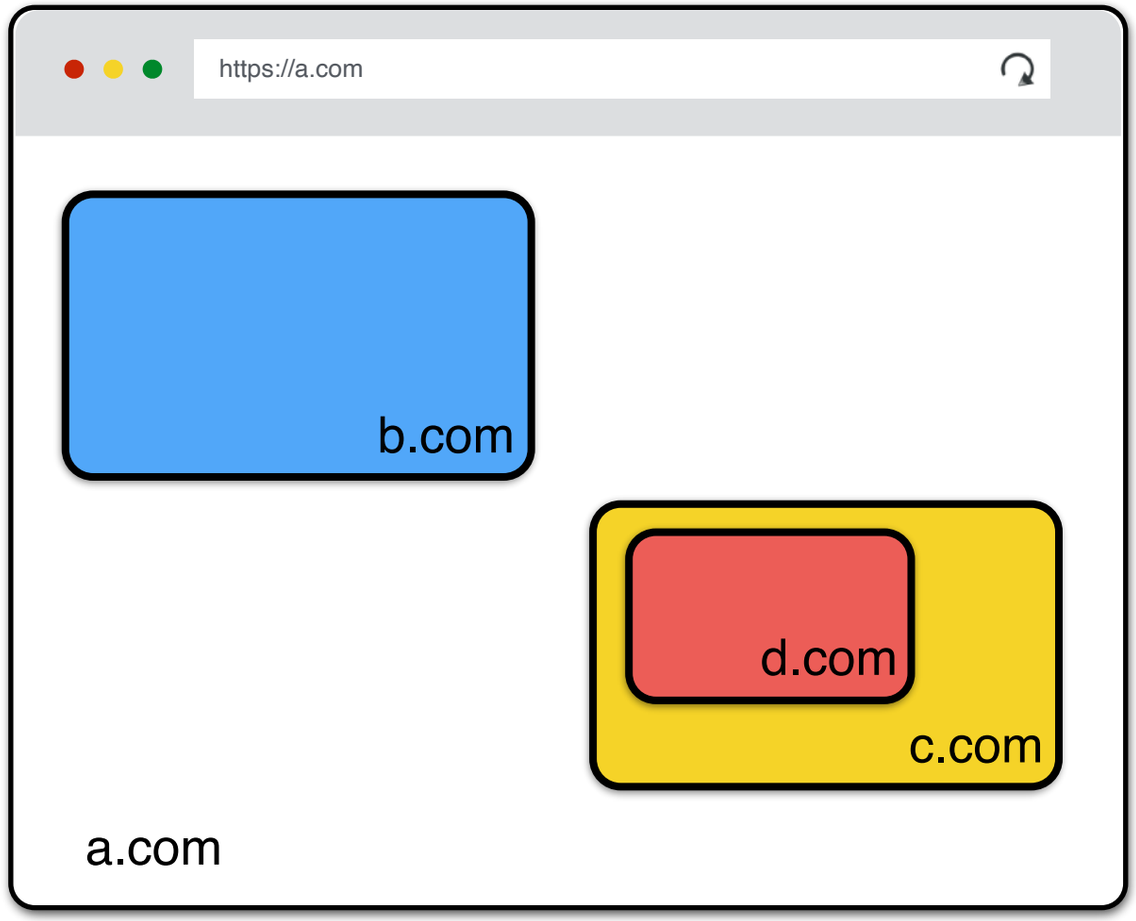



```
<p id=“demo"></p>

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

(i)Frames

- 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)

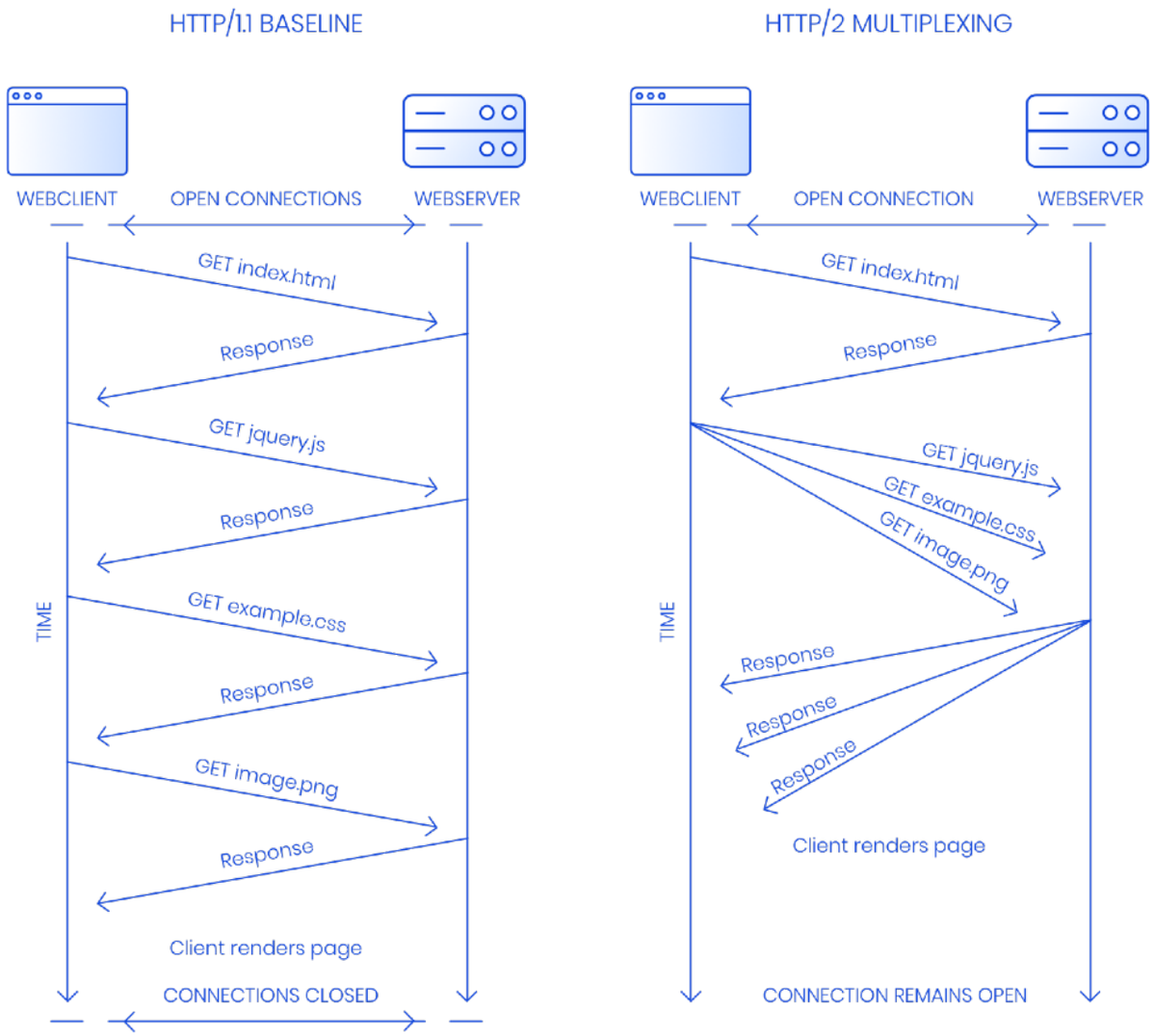


Basic Execution Model

- 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)

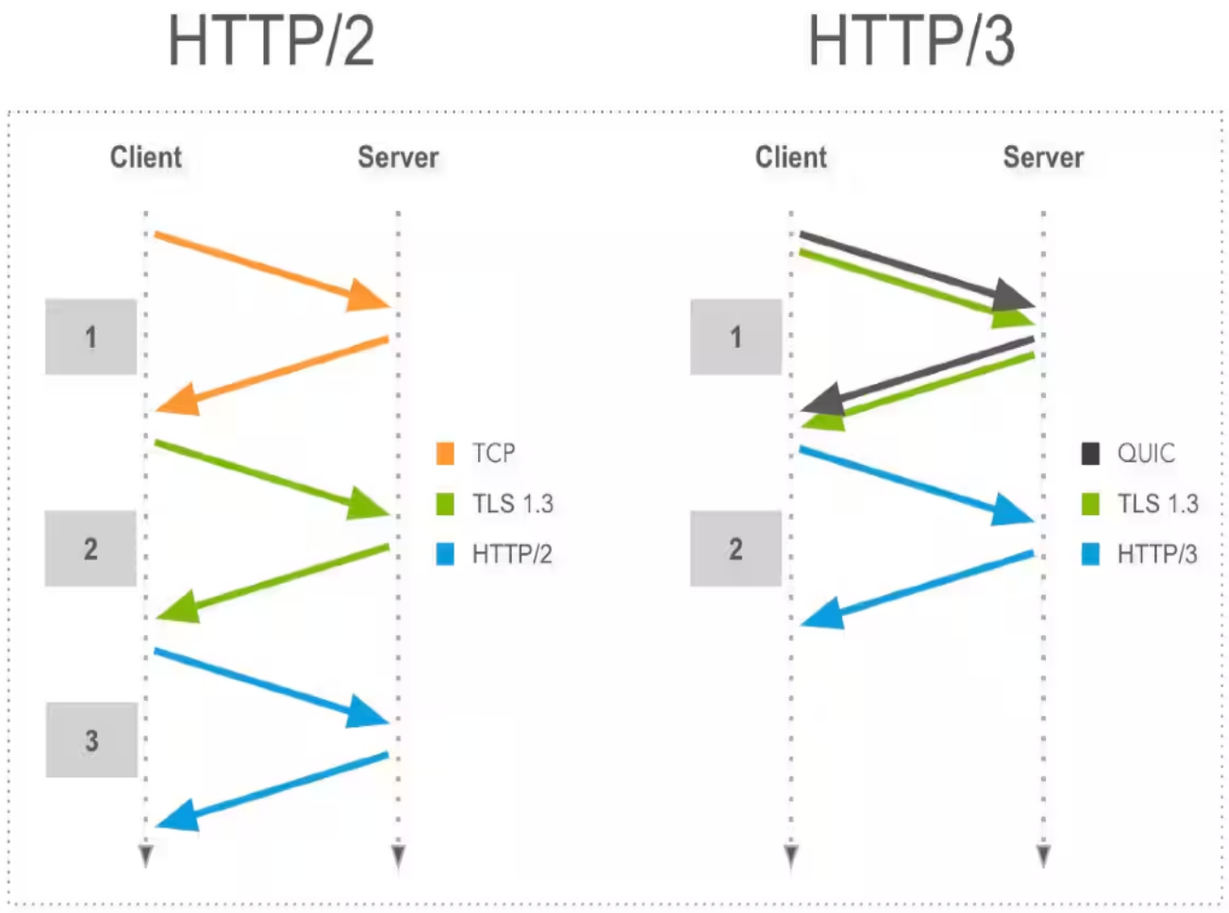
Evolution: HTTP/2

- Major revision of HTTP released in 2015
- Based on Google SPDY Protocol
- No major changes in how applications are structured
- Major changes (mostly performance):
  - Allows pipelining requests for multiple objects
  - **Multiplexing** multiple requests over one TCP connection
  - Header Compression
  - Server push



Evolution: HTTP/3

- Published in 2022
- Now used on 30.9% websites & Supported by most web browsers
- Backward compatible in format
- Major change:
  - Use **QUIC** to replace TCP as transportation layer protocol



Cookies & Sessions

HTTP is Stateles

HTTP Request

GET /index.html HTTP/1.1

HTTP Response

HTTP/1.0 200 OK

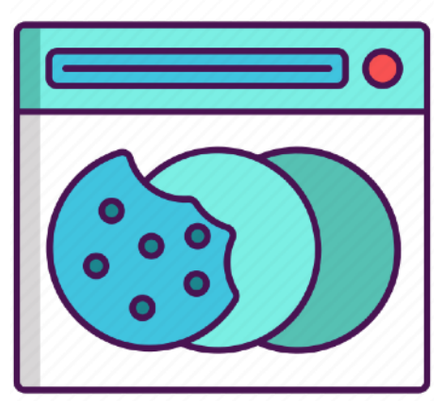
Content-Type: text/html

<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 Cookies

HTTP Response

HTTP/1.0 200 OK

Date: Tue, 01 Oct 2024 02:20:42 GMT

Server: Microsoft-Internet-Information-Server/5.0

Connection: keep-alive

Content-Type: text/html

Set-Cookie: trackingID=327292342732823 4

Set-Cookie: userID=F3D947C2

Content-Length: 2543

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

Setting Cookies

HTTP Response

HTTP/1.0 200 OK

Date: Tue, 01 Oct 2024 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 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

Login Session

GET /loginform HTTP/1.1
cookies:[]

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

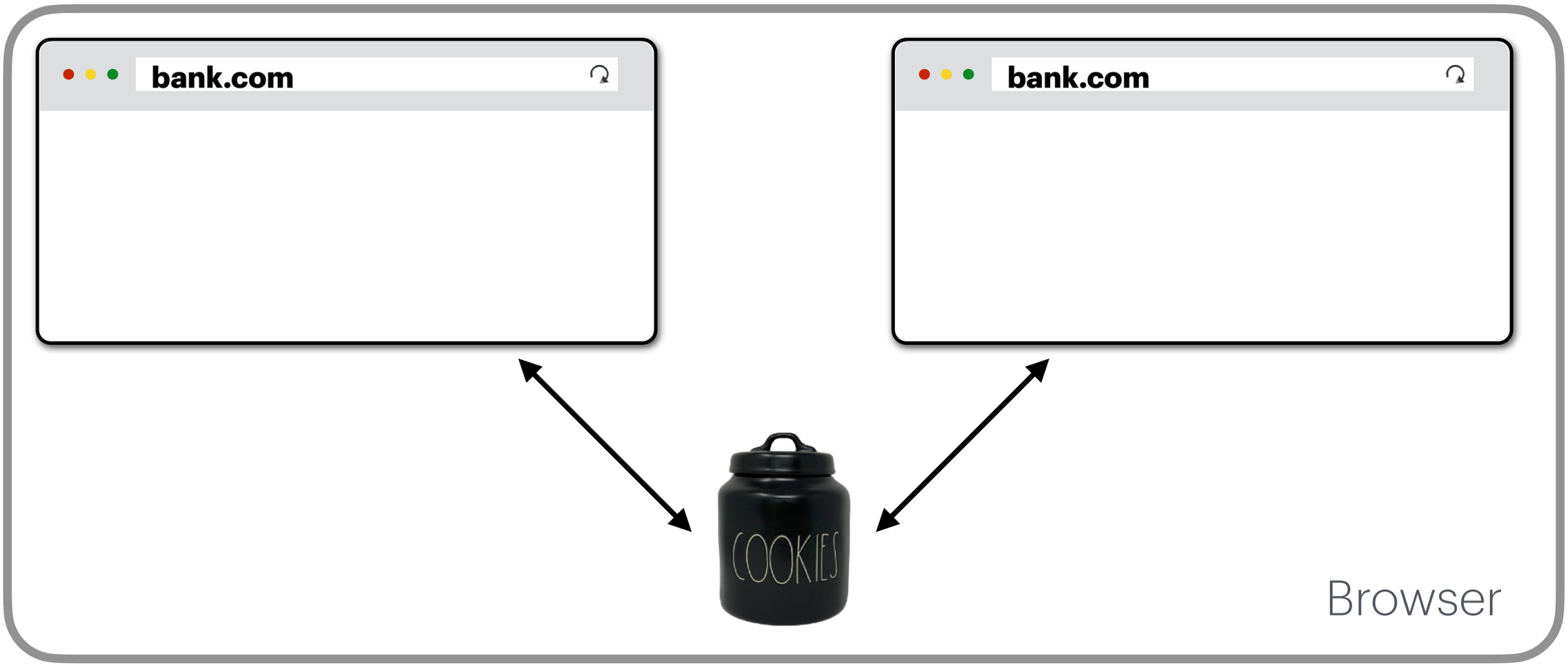
POST /login HTTP/1.1
cookies: []
username: Alice
password: secretpwd

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]

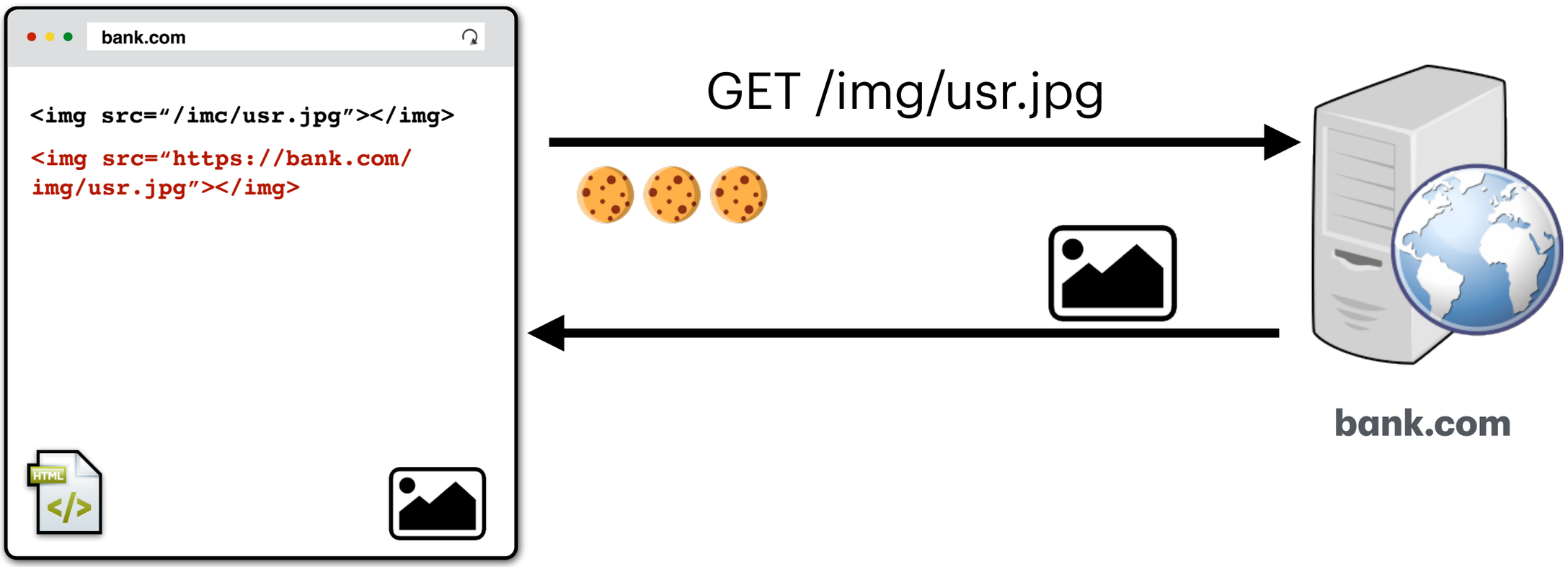
Shared Cookie Jar



- 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

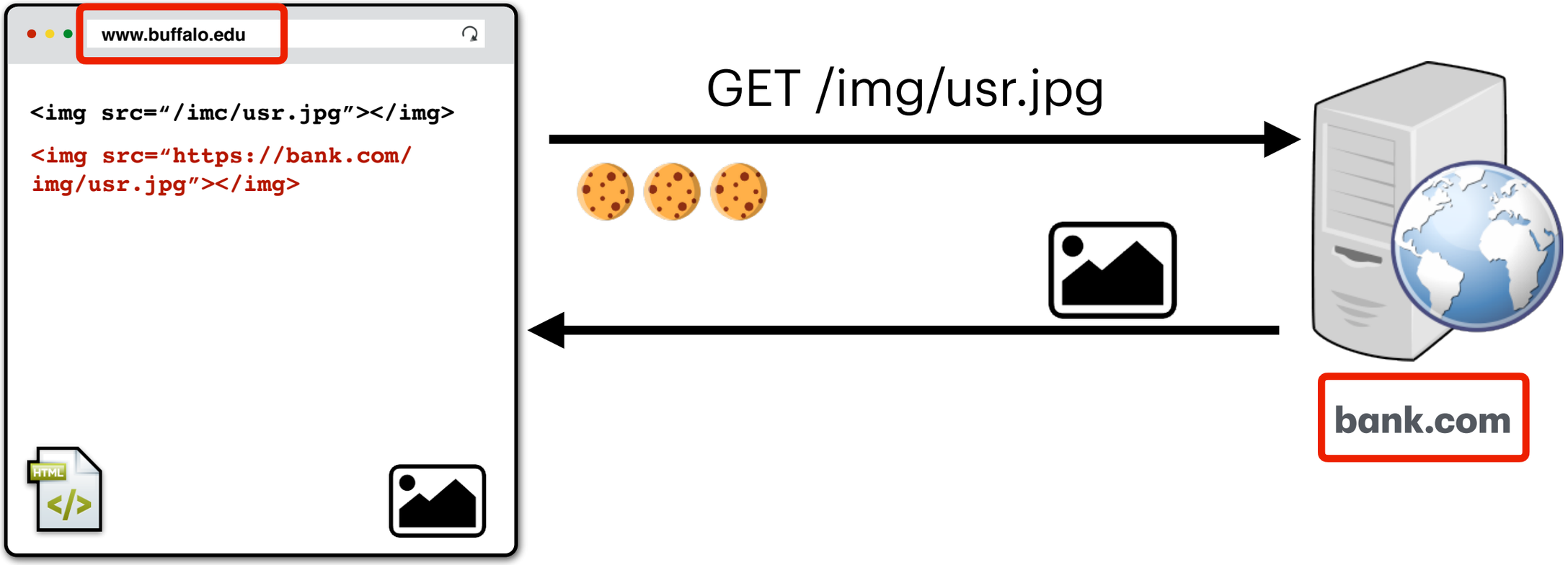
Cookies are *always* sent

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



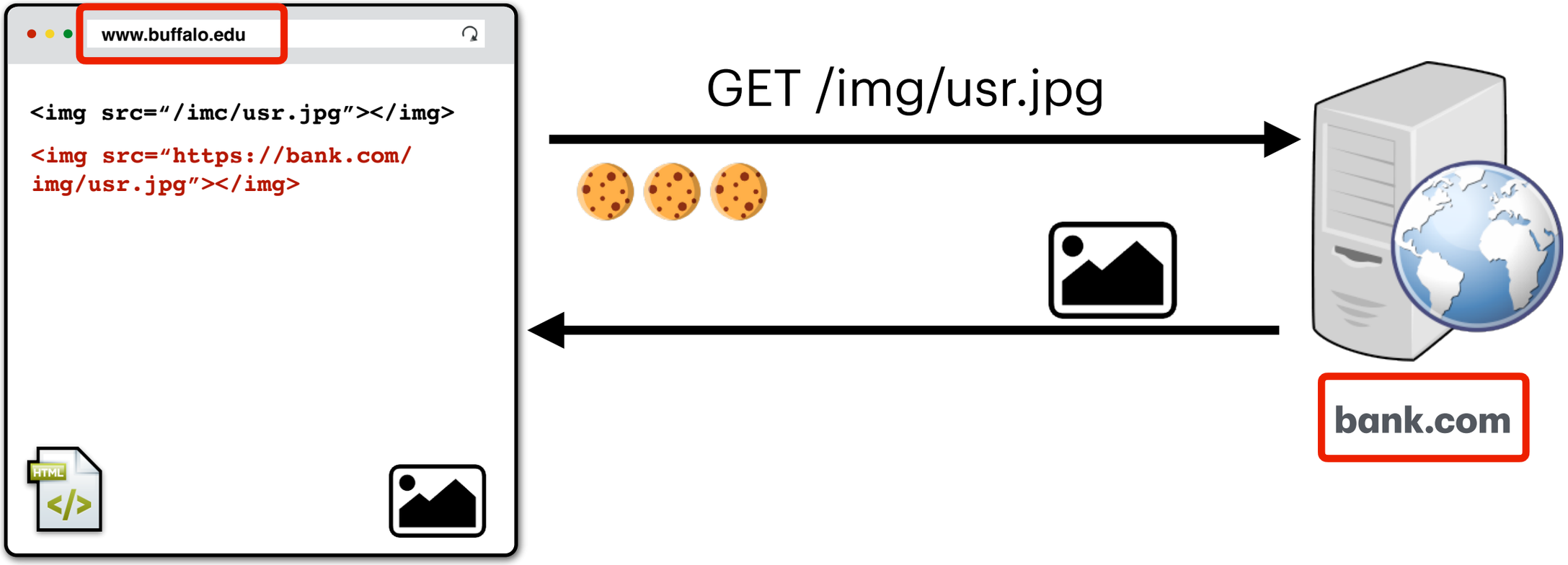
Cookies are *always* sent

- Cookies set be a domain are **always** sent for any request *to that domain* … even if the request is from a different domain



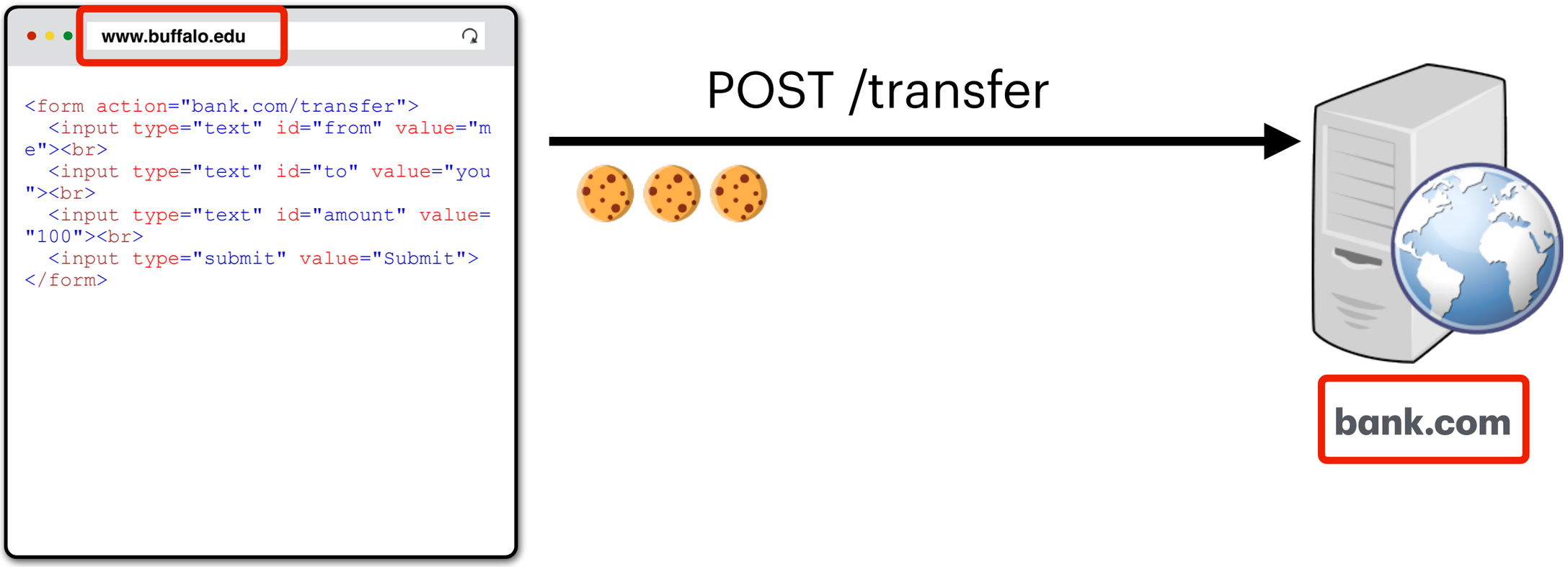
Cookies are *always* sent

- Cookies set be a domain are **always** sent for any request *to that domain* … even if the request is from a different domain
  - Can this be abused? Next lecture: CSRF attacks



POSTs also send cookies

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



The Same Origin Policy

Web Isolation

- **Safely browse the web**: Visit a web sites (including malicious ones!) without incurring harm
  - Site A cannot steal data from your device, install malware, access camera, etc.
  - Site A cannot affect session on Site B or eavesdrop on Site B
- **Support secure high-performance web apps**
  - Web-based applications (e.g., Google Meet) should have the same or better security properties as native desktop applications

Web Isolation

- **Safely browse the web**: Visit a web sites (including malicious ones!) without incurring harm
  - Site A cannot steal data from your device, install malware, access camera, etc.
  - Site A cannot affect session on Site B or eavesdrop on Site B
- **Support secure high-performance web apps**
  - Web-based applications (e.g., Google Meet) should have the same or better security properties as native desktop applications

Recall: UNIX Security Model

- **Subjects (Who?)**
  - Users, processes
- **Objects (What?)**
  - Files, directories
  - Files: sockets, pipes, hardware devices, kernel objects, process data
- **Access Operations (How?)**
  - Read, Write, Execute

Web Security Model

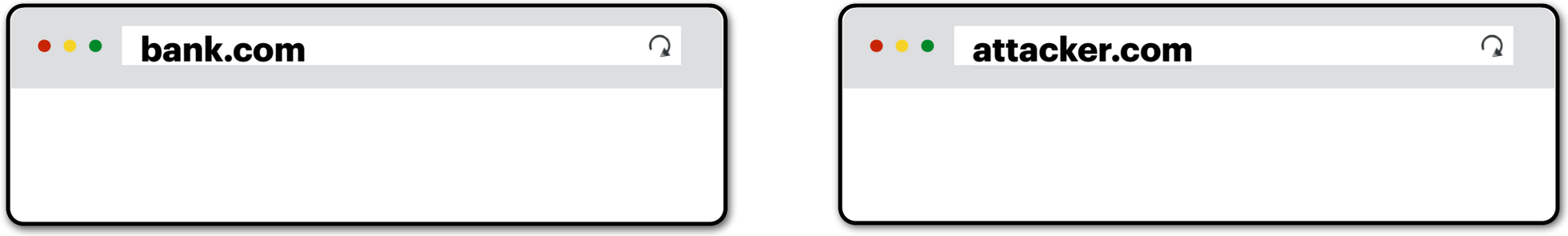
- **Subjects (Who?)**
  - "Origins" — a unique **scheme://domain:port**
- **Objects (What?)**
  - DOM tree, DOM storage, cookies, javascript namespace, HW permission
- **Same Origin Policy (SOP)**
  - **Goal**: Isolate content of different origins
    - **Confidentiality**: script on evil.com should not be able to *read* bank.ch
    - **Integrity**: evil.com should not be able to *modify* the content of bank.ch

Origins Examples

- **Origin** defined as **scheme://domain:port**
- All of these are different origins — ***cannot*** access one another
  - http://buffalo.edu
  - http://www.buffalo.edu
  - http://buffalo.edu:8080
  - https://cse.buffalo.edu
- These origins are the same — ***can*** access one another
  - https://buffalo.edu
  - https://buffalo.edu:80
  - https://buffalo.edu/cse

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

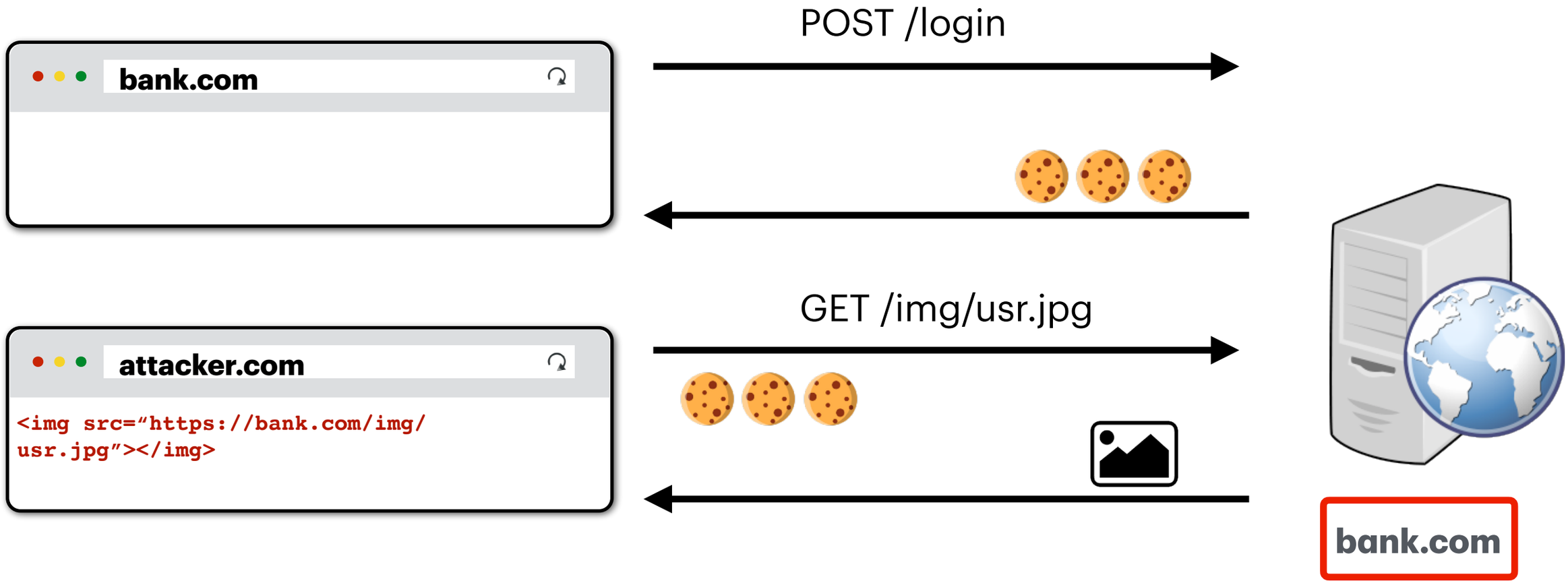
Bounding Origins - Frames

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

attacker.com cannot...

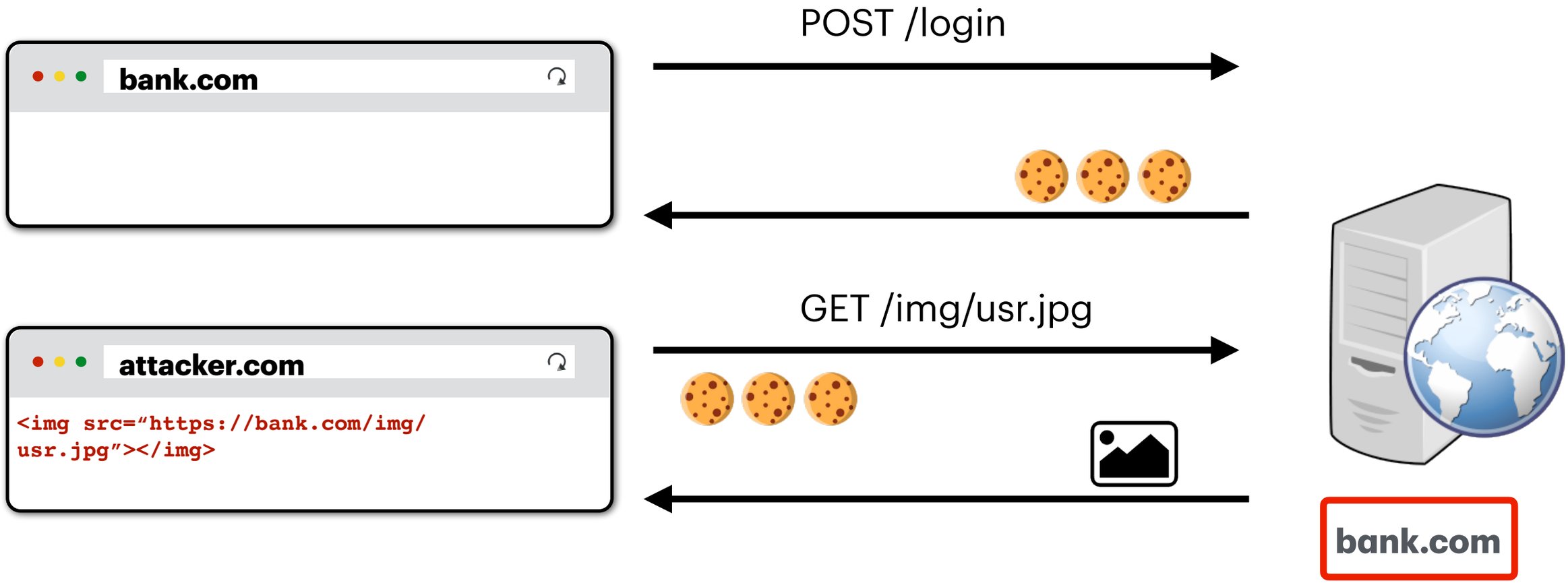
- *read* or *write* content from bank.com frame
- access bank.com's cookies
- *detect* that bank.com has loaded

Origins and Cookies



- Browser will always send bank.com cookie
- SOP blocks attacker.com from **reading** bank.com's cookie

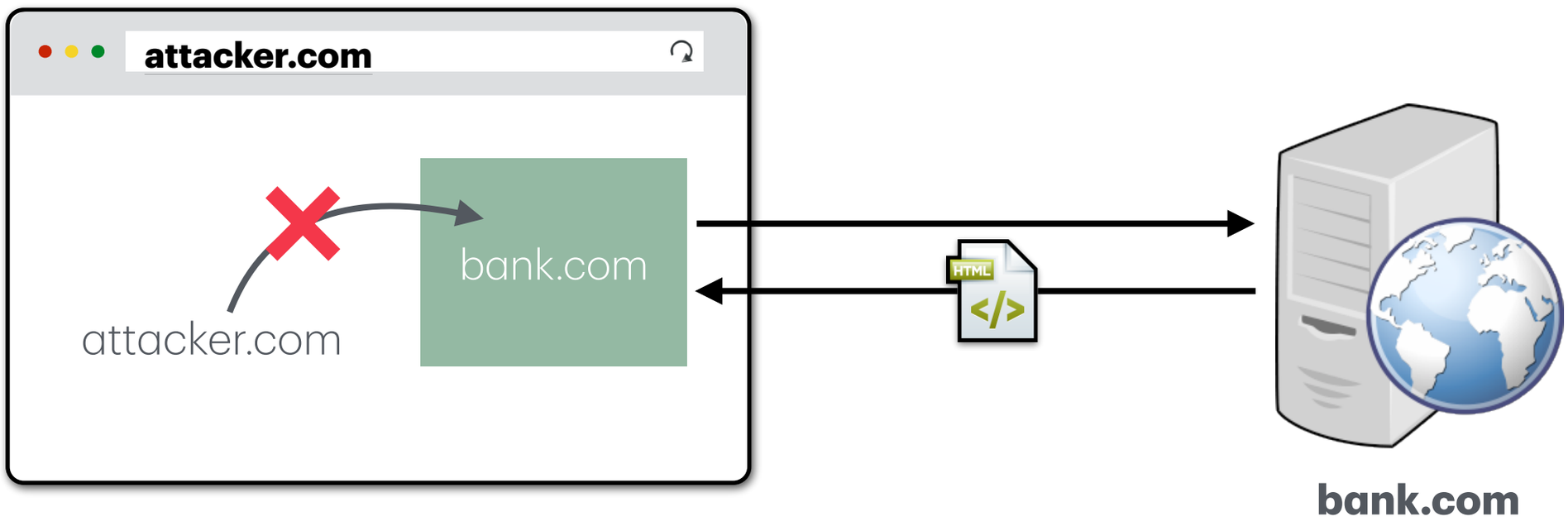
SOP for HTTP Response



- Pages *can* make requests ***across*** origins
- SOP does ***not*** prevent attacker.com from making the request.

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 not *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)

Questions?