CS 188

Scalable Internet Services

John Rothfels

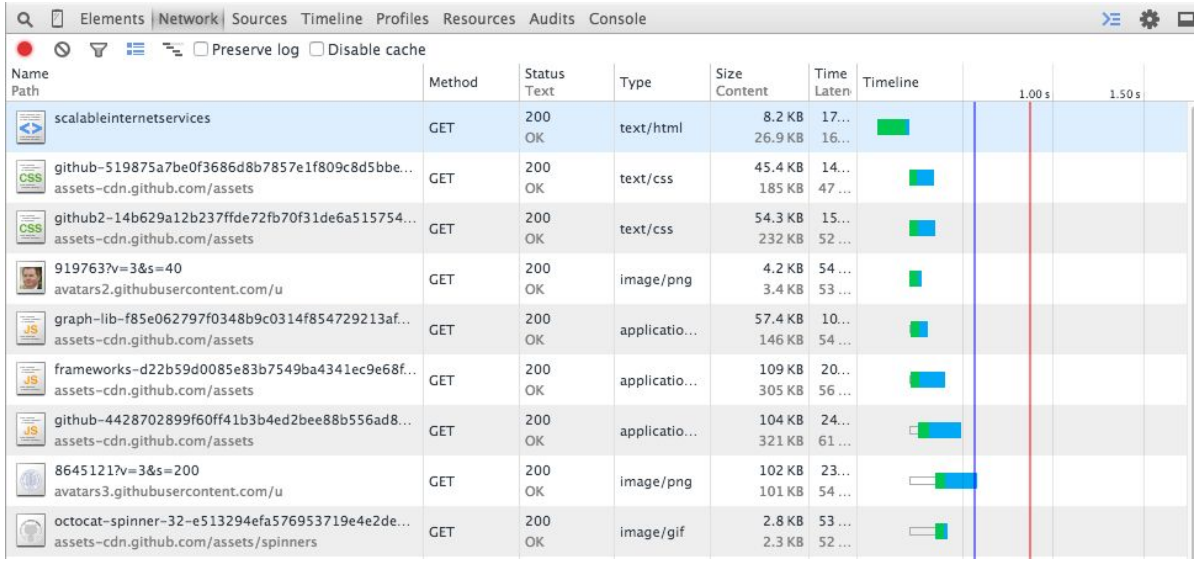
October 1, 2019

Today's Agenda

- The Lifecycle of a Web Request

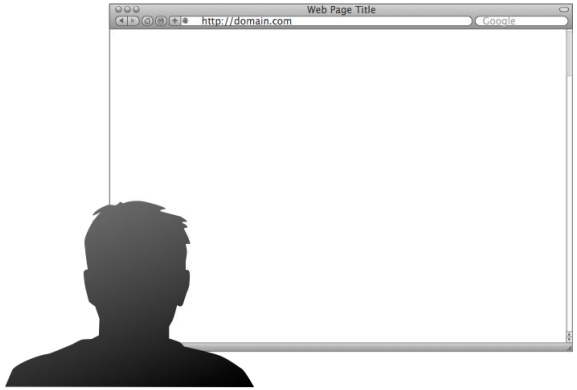
- Understand the browser's technology stack
  - HTTP
  - HTML, CSS

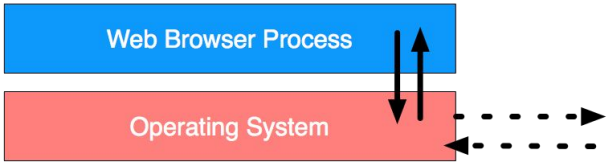
For Today



After today, you should have all the basic tools you need to understand output like this

The Lifecycle of a Request





Starting with the basics

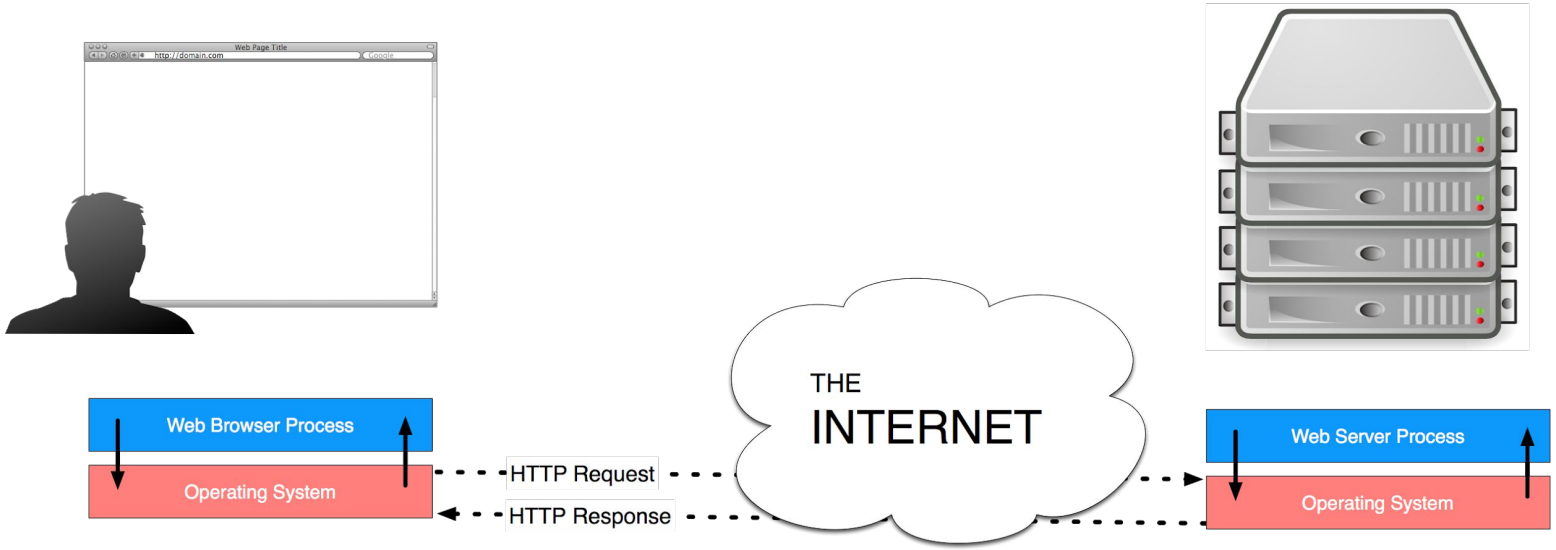
- A web browser is a process
- It runs on an operating system
- It responds to user input, renders graphics, and uses the network.

The Lifecycle of a Request

Starting with the basics

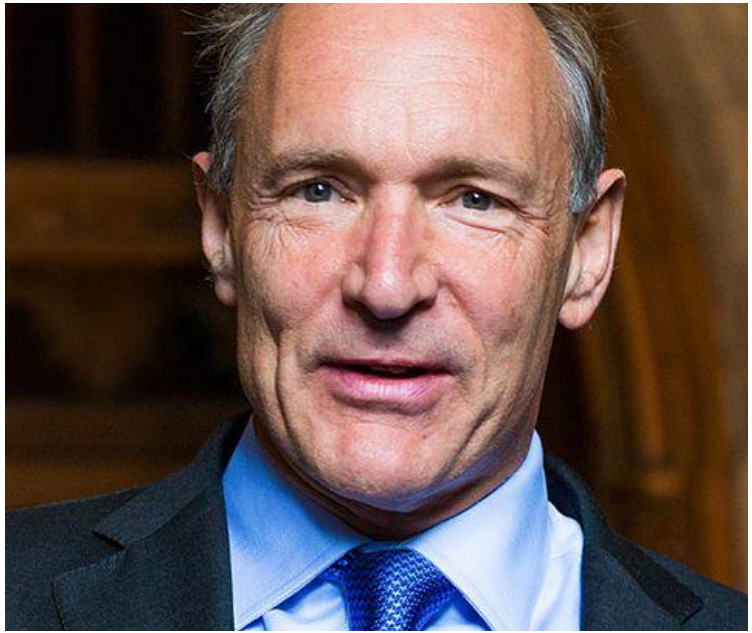
- A web server is a process
- It runs on an operating system
- It interacts with the network and the filesystem.

The Lifecycle of a Request

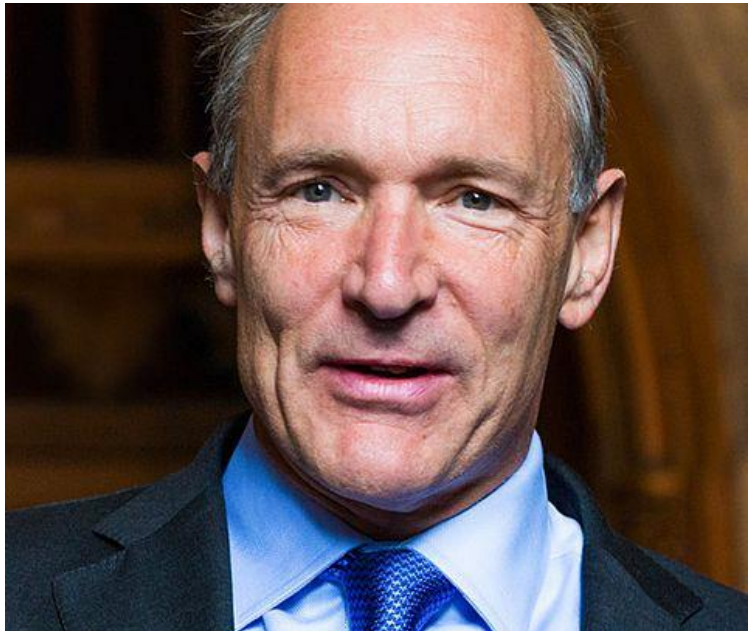


These two processes communicate over the internet.

HTTP



HTTP



Tim Berners-Lee

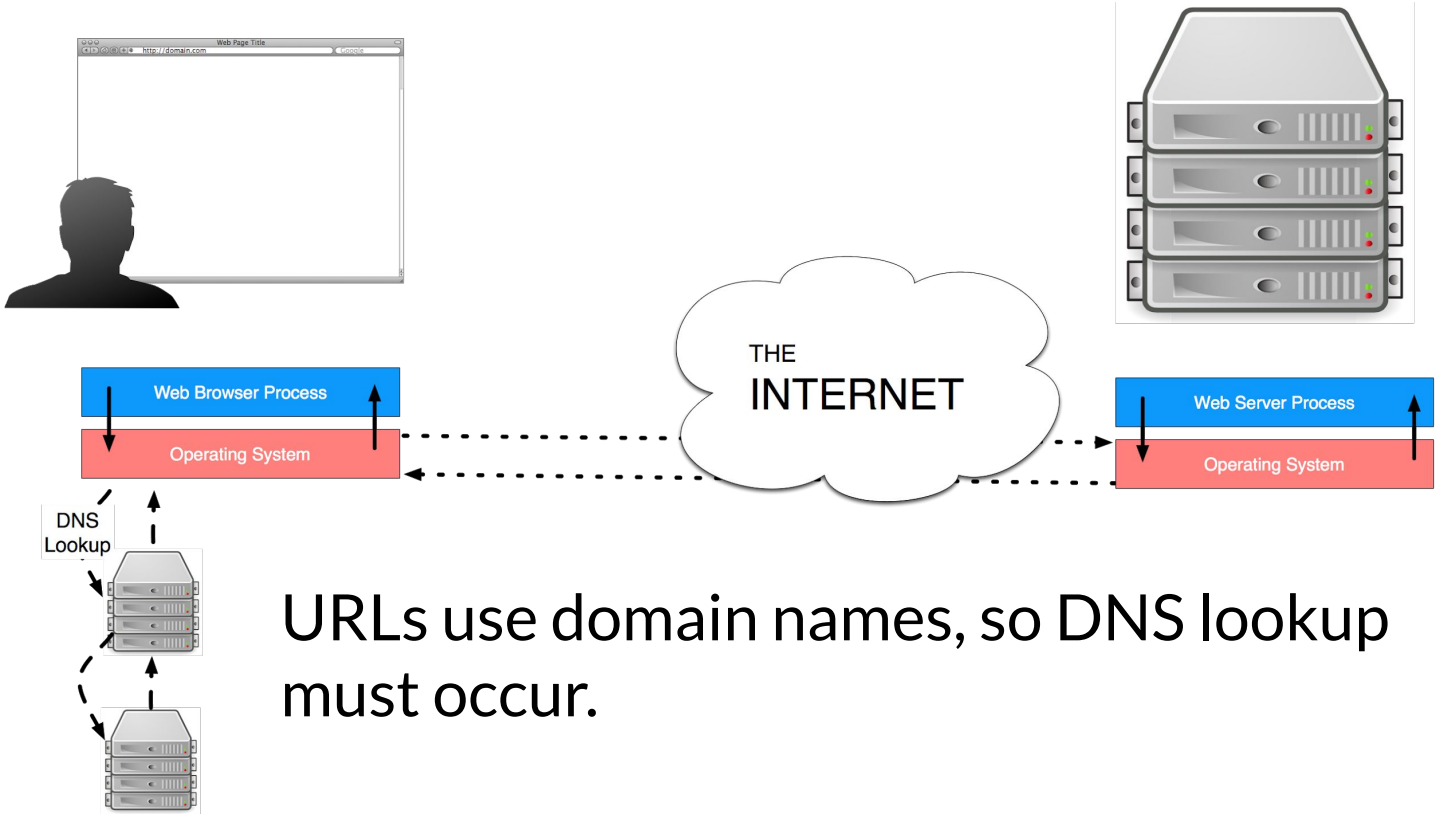
Marc Andreessen

HTTP

- In 1990, the internet has existed for ~20 years, email for ~8 years.
- Tim Berners-Lee is at CERN, and believes that two already-existing technologies could be combined to good effect: the internet and HyperText
  - HyperText is the idea of linking documents together with HyperLinks.
  - `<a href="http://mycoolsite.com/page2" />`
  - Berners-Lee creates the first versions of HTTP and HTML

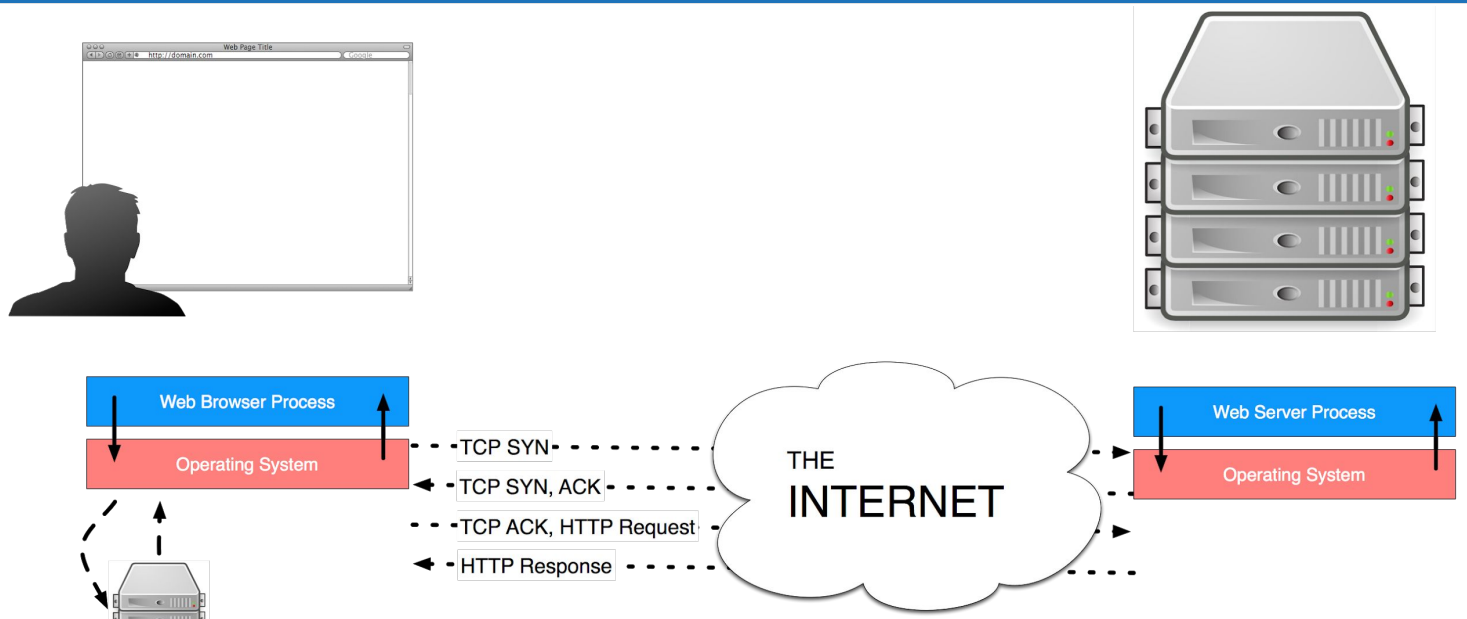
- In 1993, Mosaic is created at the NCSA center at UIUC
- In 1994, Marc Andreessen leaves UIUC and founds Netscape with Jim Clark.
- In 1998, AOL acquires Netscape

The Lifecycle of a Request



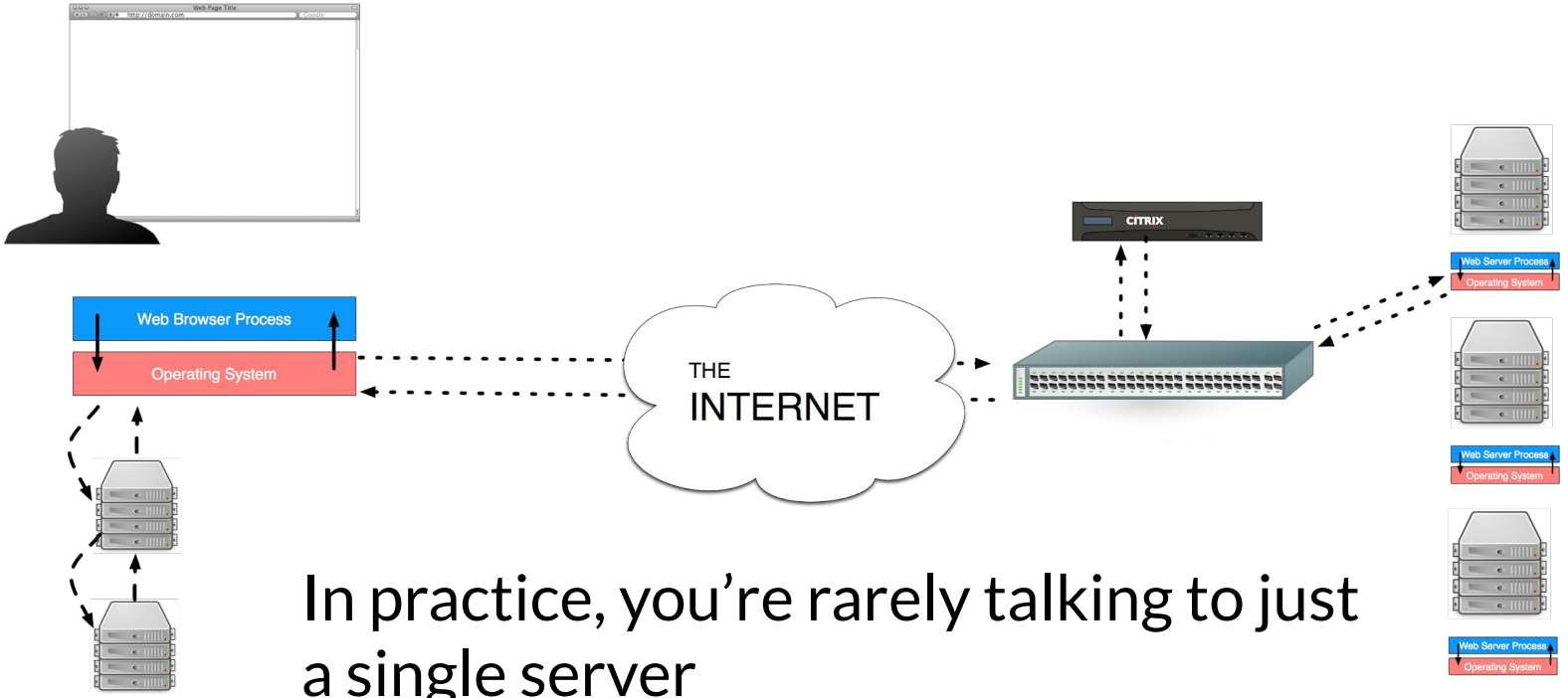
URLs use domain names, so DNS lookup must occur.

The Lifecycle of a Request



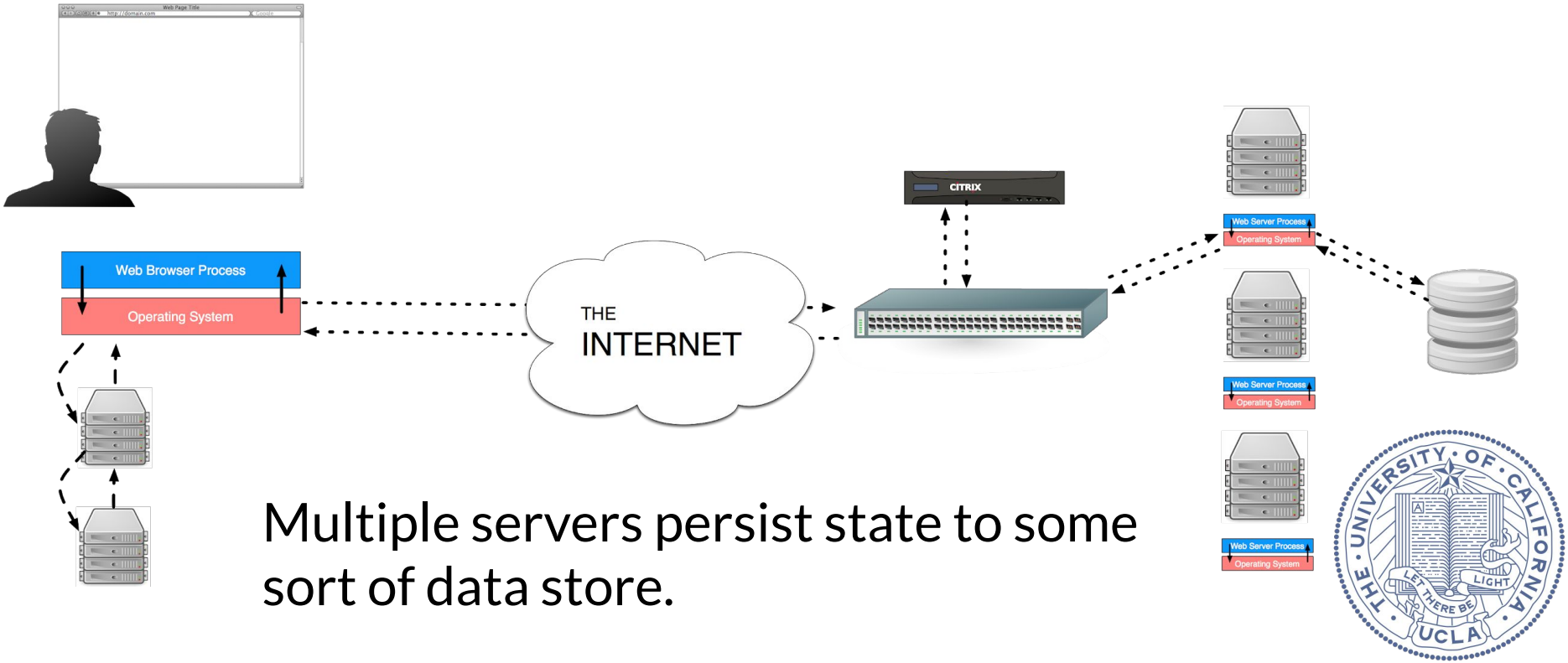
HTTP is built on trusty TCP, so there's the requisite setup

The Lifecycle of a Request



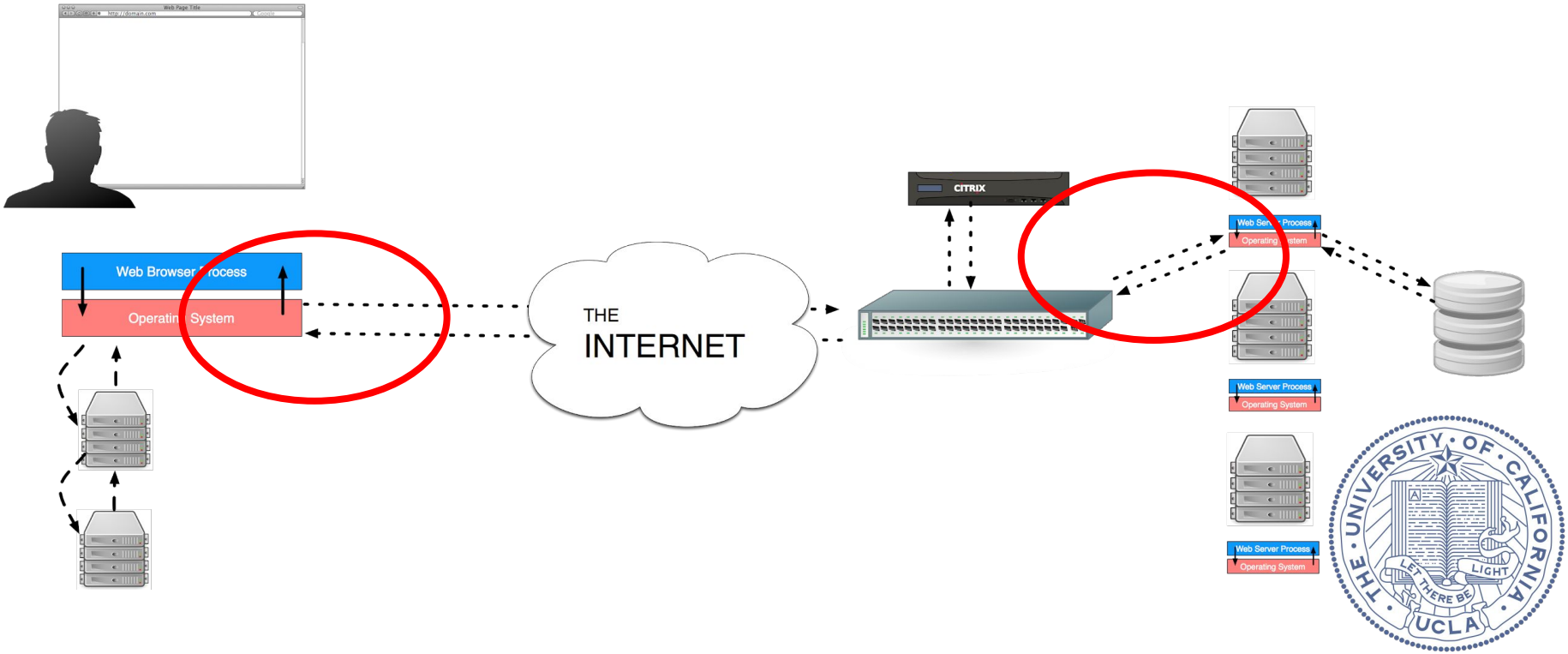
In practice, you're rarely talking to just a single server

The Lifecycle of a Request



Multiple servers persist state to some sort of data store.

HTTP


Web Browser Process
Operating System
THE
INTERNET
CITRIX
Web Server Process
Operating System
Web Server Process
Operating System
Web Server Process
Operating System
THE UNIVERSITY OF CALIFORNIA
LET THERE BE LIGHT
UCLA

HTTP


PROPRIETE CERN
PROPRIETE CERN
Information Management: A Proposal


THE UNIVERSITY OF CALIFORNIA
LET THERE BE LIGHT
UCLA

HTTP Explained

- Simple ASCII protocol

- In its original, simplest version
  - Open a TCP socket (standard is port 80)
  - Send over a request
  - Response comes back
  - Close the TCP socket

- Originally designed to exchange HTML, extended to send anything
  - Originally designed for web browser to server interaction, today very widely used for server to server APIs

HTTP Explained

- Request:
  - **Verb**: What do you want to do? GET something?
  - **Resource**: What is logical path of the resource you want?
  - **HTTP version**: specify version to aid in compatibility
  - **Headers**: Standard ways to request optional behavior
  - **Body**: A data payload

- Response comes back:
  - **HTTP Version:** specify version to aid in compatibility
  - **Status code:** Success? Failure? Other?
  - **Headers**: Standard ways to convey metadata
  - **Body**: A data payload

HTTP Explained

Verb
Resource
Version
Headers
Status
Body

Request:

```
GET /about/ HTTP/1.1
Accept: text/html
```

Response:

```
HTTP/1.1 200 OK
Content-Type: text/html

<!DOCTYPE html>
<html class="example" lang="en">
...
```

HTTP Verbs

- **`GET`**
  - Get a copy of the resource
  - Should have no side-effects
  - Example: **`GET /users/sign_out`** shouldn't be done.

- **`POST`**
  - Sends data to the server. Generally creates a new resource
  - Commonly used for form submissions.
  - Should be assumed to have side-effects.
  - Not idempotent.
    - "Do you want to post your form submission again?"

HTTP Verbs

- **`PUT`**
  - Sends data to the server. Generally updates an existing resource.
  - Has side effects, but is idempotent.
    - Does this make sense?
  - The difference between **`PUT`** and **`POST`** is subtle. We will cover it more later in the course while discussing REST.

- **`DELETE`**
  - Destroys a resource.
  - Has side effects, is idempotent.
  - This is the right way to do a user signout:
    - **`DELETE /session/<id>`**

HTTP Verbs

- **`HEAD`**
  - Just like **`GET`**, but only returns the headers.
  - What would this be useful for?
    - (Hint: we will be discussing this topic in greater length later)

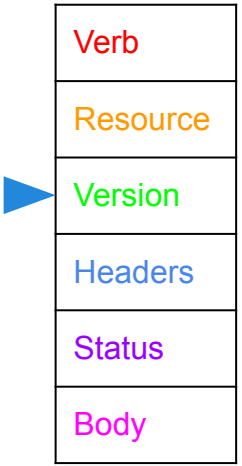
- These are how the verbs should be used, not everyone understands them or uses them correctly.
  - Example: **`GET /blog_postings/5?action=hide`**

- What problems can you think of happening through the misuse of HTTP verbs?

HTTP Verbs

- Less commonly used verbs:
  - **`TRACE`**
  - **`OPTIONS`**
  - **`CONNECT`**

HTTP Version



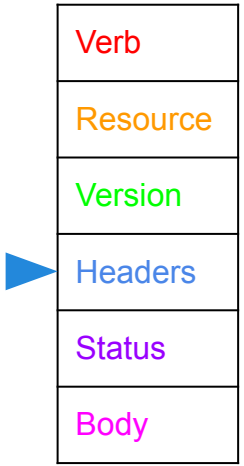
The HTTP version is included with each request/response for ease of protocol development.

Everyone today uses HTTP 1.1, with some support for SPDY/HTTP 2.0

Timeline:

- 1991, HTTP 0.9: Single line protocol. No headers.
- 1996, HTTP 1.0: Headers added, more than just HyperText.
- 1999, HTTP 1.1: Connection keep-alive, more caching mechanisms, everything else we enjoy today
- 2015: HTTP 2.0 finalized. Future lecture with details.

HTTP Headers

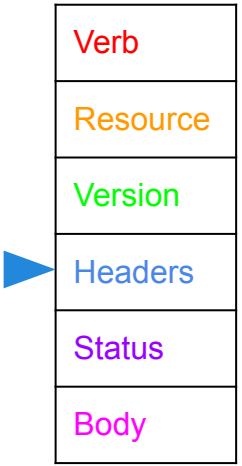


HTTP headers are where a lot of the power lies.

- `Accept:` indicates the preferred format of the resource
  - `Accept: text/html`
    - Give me the resource as a web page
  - `Accept: application/json:`
    - Give me the resource as a JSON document
  - `Accept: application/json,application/xml :`
    - Give me the resource as a JSON document, but if you can't do that then I will accept XML.

- `Accept-Encoding:` indicates the content should be compressed
  - `Accept-Encoding: bzip2,gzip`
    - Compress the data using bzip2, and if you don't support that, then use gzip

HTTP Headers

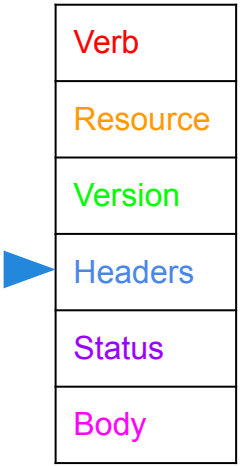


- `Host:` indicates the DNS host the requestor is trying to reach
  - Why is this important? Why isn't the host implied?

- `Accept-Language:` indicates the preferred languages (in order)
  - `Accept-Language: es,en-US`
    - I prefer spanish, but will accept US english.
    - Without this header, every website intended for an international audience would need a massive "choose your language" list on its homepage.

- `User-Agent`: indicates what type of browser or device is connecting.
  - Too often, web applications make decisions based on this.
  - Ridiculousness ensues. Current Chrome User-Agent:
    - `Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_5) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/37.0.2062.124 Safari/537.36`

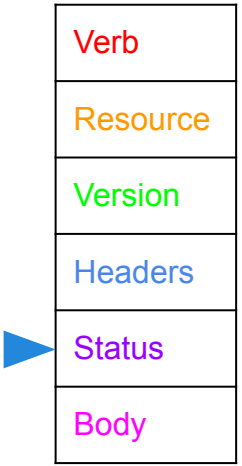
HTTP Headers



There are many more headers than we can cover here.

- Some useful headers that we will discuss later in the course regarding caching:
  - `ETag, Date, Last-Modified, Cache-Control, Age`
- Some useful headers we will later discuss later in the course regarding security:
  - `Strict-Transport-Security, X-Frame-Options, X-Forwarded-Proto`
- `Transfer-Encoding: chunked` can be used to serve up resources whose length is not known
- Headers that begin with `X-` are not part of the specification, but may be commonly used and "standardized"

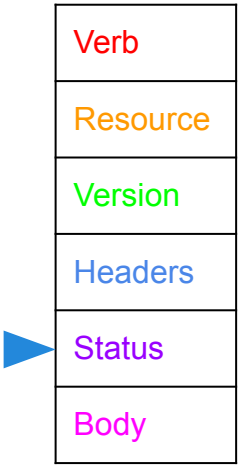
HTTP Headers



HTTP Status is provided in the response to indicate the outcome of the request.

- The first digit of a status code classifies it:
  - 1XX - Informational
  - 2XX - Successful
  - 3XX - Redirecting
  - 4XX - Client Error
  - 5XX - Server Error

HTTP Headers



A few of the most common response codes:

- `200 OK`: The go-right case.
- `301 Moved Permanently`: The client should use the provided new URL moving forward.
  - The new URL will be provided in the `Location` header.
- `302 Found`: You should go to a different URL to get this resource, but it hasn't permanently moved there.
- `403 Forbidden`: The request is not authorized
- `404 Not Found`: Specified resource could not be found
- `418 I'm a Teapot`: April fools day joke
- `500 Internal Server Error`: Something crashed.
- `503 Service Unavailable`: Temporary failure.

HTTP Body

The HTTP body is where content is delivered.

When used in a request, the HTTP body can have key-value pairs for a `POST` or a `PUT`.

Example:

```
POST /comments HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Content-Length: 32

username=andrew&comment=Hello%20World
```

HTTP Body

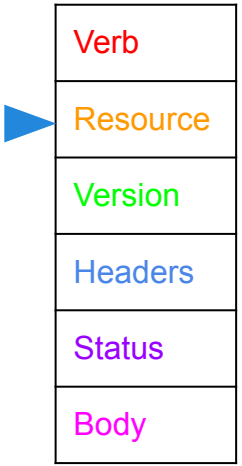
When used in a response, the body most commonly has the data of the resource.

```
GET /about/ HTTP/1.1
Accept: text/html

HTTP/1.1 200 OK
Content-Type: text/html

<!DOCTYPE html>
<html>
  <head>...
```

HTTP Resource



- The resource specifies the thing you are referring to via a logical hierarchy.
  - This is not the file path location.
  - Good: `http://www.amazon.com/gp/product/1565925092/`
  - Bad: `http://www.cocacola.com/index.jsp?page_id=4251`

- Anything past the question mark is called a query string
  - Query strings are intended to assist in locating the resource
  - Parameters are assigned using equals, concatenated using ampersand
  - Example:
    - http://www.amazon.com/search?term=dogs&new=1

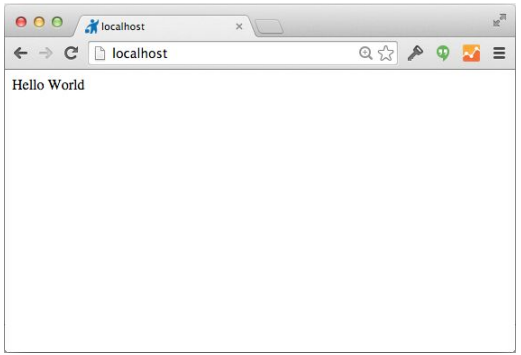
Demoing the protocol

```
%sudo nc -l 80 #after this, tell chrome to go to localhost


GET / HTTP/1.1
Host: localhost
Connection: keep-alive
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_5) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/37.0.2062.124 Safari/537.36
Accept-Encoding: gzip,deflate,sdch
Accept-Language: en-US,en;q=0.8
```

Demoing the protocol

```
%echo "HTTP/1.1 200
OK\nContent-Type:
text/html\n\n<html><body>Hello
World</body></html>" | sudo nc -l
80
```



Demoing the protocol

```
%echo "GET /about/
HTTP/1.1\nAccept: text/html \n\n" |
nc www.google.com 80
```

```
HTTP/1.1 200 OK
Vary: Accept-Encoding
Content-Type: text/html
Last-Modified: Tue, 26 Aug 2014 10:35:15 GMT
Date: Tue, 07 Oct 2014 02:42:23 GMT
Expires: Tue, 07 Oct 2014 02:42:23 GMT
Cache-Control: private, max-age=0
X-Content-Type-Options: nosniff
Server: sffe
X-XSS-Protection: 1; mode=block
Alternate-Protocol: 80:quic,p=0.01
Transfer-Encoding: chunked

3aae
<!DOCTYPE html>
<html class="google" lang="en">
```

Demoing the protocol

```
%echo "GET /about/
HTTP/1.1\nAccept-Language:
es\nAccept: text/html \n\n" | nc
www.google.com 80
```

```
HTTP/1.1 200 OK
Vary: Accept-Encoding
Content-Type: text/html
Last-Modified: Tue, 26 Aug 2014 10:35:15 GMT
Date: Tue, 07 Oct 2014 02:38:47 GMT
Expires: Tue, 07 Oct 2014 02:38:47 GMT
Cache-Control: private, max-age=0
X-Content-Type-Options: nosniff
Server: sffe
X-XSS-Protection: 1; mode=block
Alternate-Protocol: 80:quic,p=0.01
Transfer-Encoding: chunked

3954
<!DOCTYPE html>
<html class="google" lang="es">
```

Beyond one Request per Connection

Thinking of HTTP as one request per TCP connection can be a useful simplification for reasoning about your application

- In practice this is inefficient. **Why?**

Beyond one Request per Connection

Thinking of HTTP as one request per TCP connection can be a useful simplification for reasoning about your application

- In practice this is inefficient. **Why?**
  - Initial round trips to setup connection
  - TCP connections start out low bandwidth

Let's take a look at both of these...

Beyond one Request per Connection

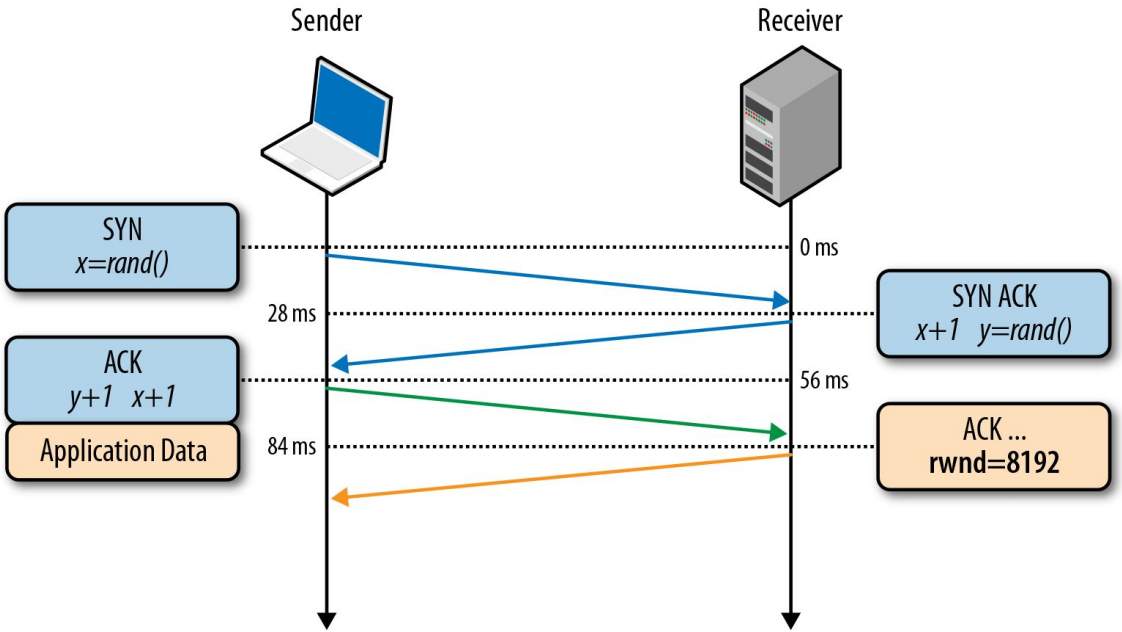
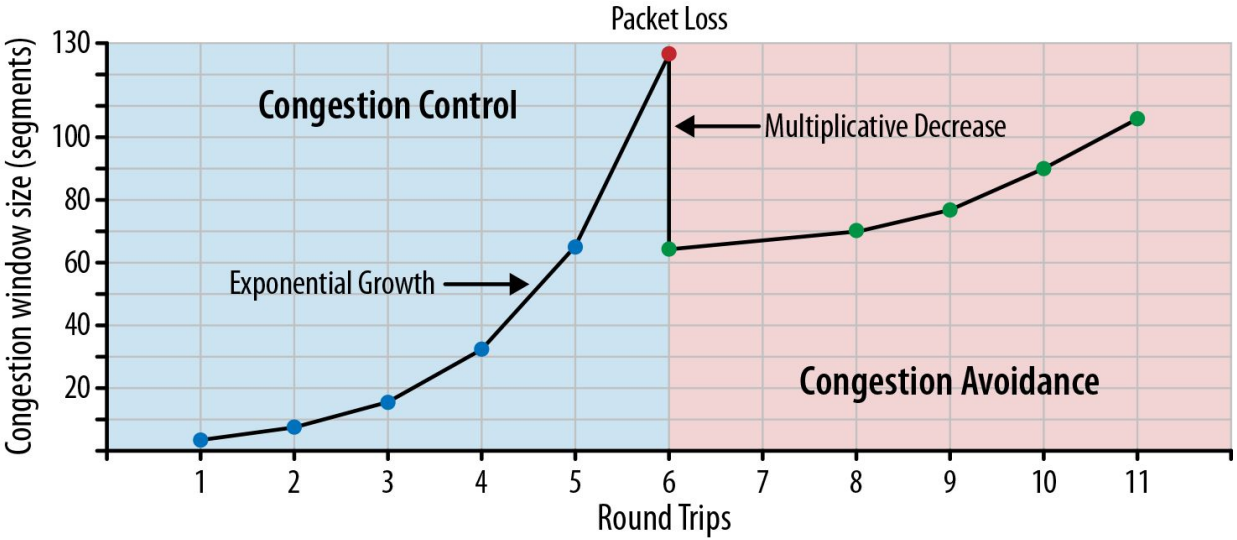


Image Source: “High Performance Browser Networking,” by Ilya Grigorik

Each TCP connection that is established requires a round trip for setup

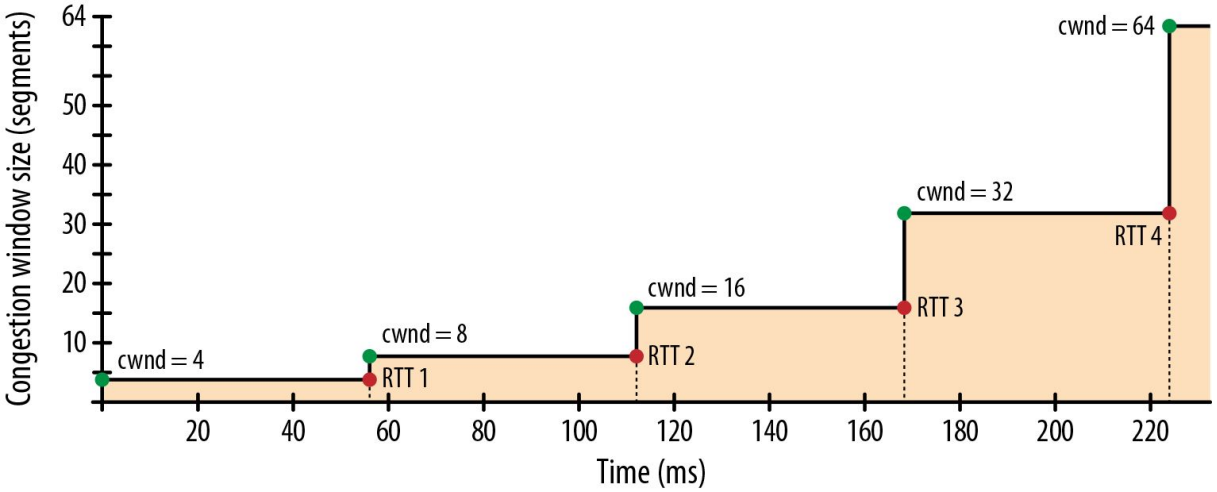
Beyond one Request per Connection



The early phases of a TCP connection are bandwidth constrained

Image Source: “High Performance Browser Networking,” by Ilya Grigorik

Beyond one Request per Connection



By repeatedly reopening TCP connections, we only use the most constrained part.

Image Source: “High Performance Browser Networking,” by Ilya Grigorik

Beyond one Request per Connection

HTTP 1.1 introduced connection keep-alive to address this.

- HTTP header exists that is "`Connection: keep-alive`", but it is the default behavior as of 1.1
- Simple mechanics:
  - Don't close the socket. Server will expect multiple requests to arrive on the same socket.
  - After receiving one response, send another request

Beyond one Request per Connection

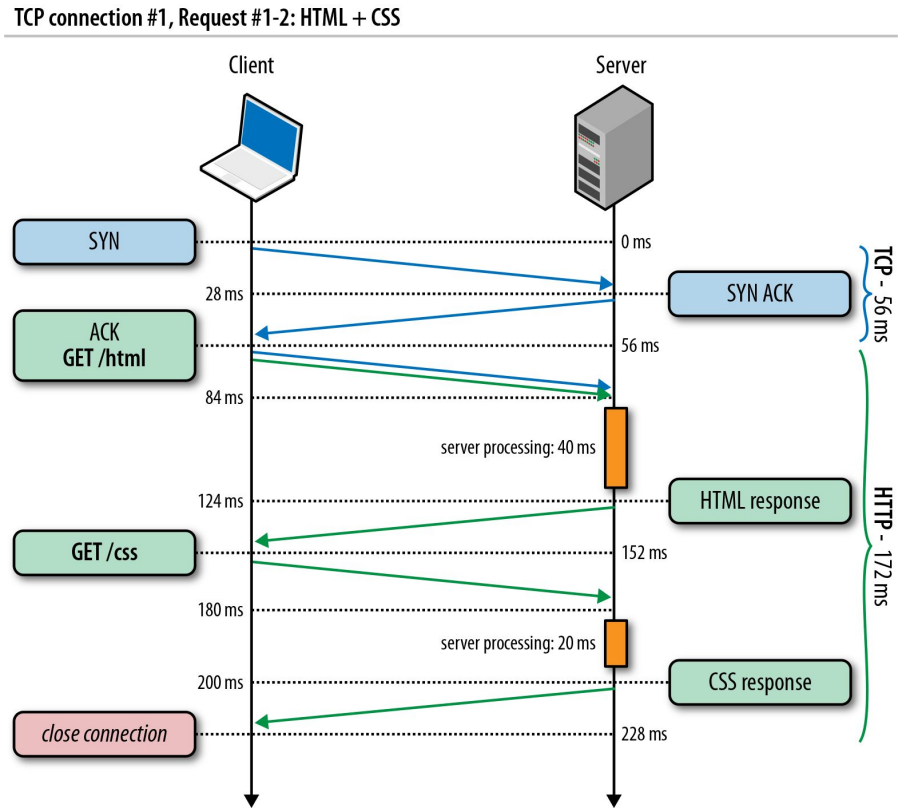


Image Source: “High Performance Browser Networking,” by Ilya Grigorik

Beyond one Request per Connection

Why not take this a step further?

- Frequently we need many resources from the same server
- Why do we wait for responses before sending subsequent requests?
- Can’t we save time on roundtrips if we send our requests without waiting for each response?

Beyond one Request per Connection

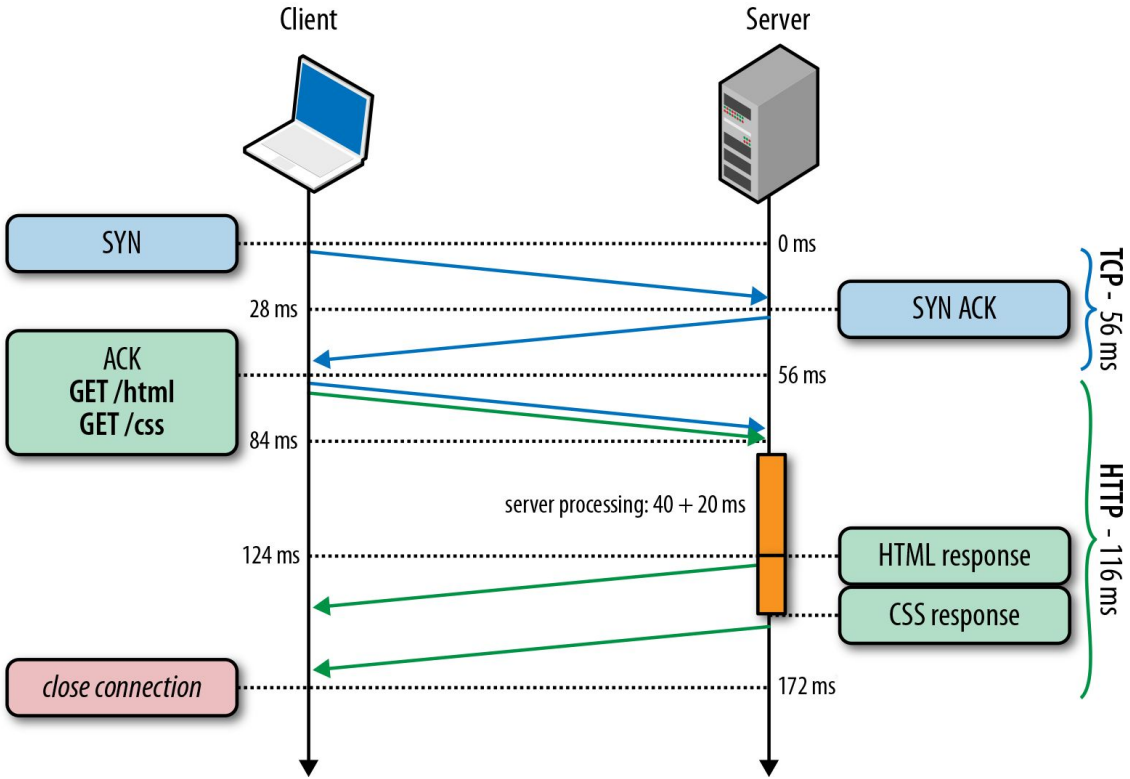


Image Source: “High Performance Browser Networking,” by Ilya Grigorik

Beyond one Request per Connection

This is called “HTTP Pipelining”

- It works, but has lots of problems
  - Head of line blocking
  - Failure mechanics
- Due to problems, while support for it exists widely, it is generally not turned on.
- Expect to see this make a comeback in HTTP 2.0

Beyond one Request per Connection

So a browser that needs N resources can establish anywhere between 1 and N TCP connections to get these.

- How many should it use?
  - Too many and you can DoS the site
  - Too few and you prohibit any parallelism
- **Whats the answer?**

Beyond one Request per Connection

So a browser that needs N resources can establish anywhere between 1 and N TCP connections to get these.

- How many should it use?
  - Too many and you can DoS the site
  - Too few and you prohibit any parallelism
- **Whats the answer?**
  - **Six.**

Beyond one Request per Connection

Six connections per server is today's "standard"

- Not part of any actual standard.

If you want to have more, you can use a trick called "domain sharding"

- Set up www1.foo.com, www2.foo.com, www3.foo.com, that all point to the same server
- The browser will treat these as separate servers and open six connections per domain

HTML Explained

Overview of HTML

- Syntax
- Essential tags
- Relationship to CSS

HTML Explained: Syntax

- HyperText Markup Language
  - A markup language takes flat text and annotates it to add structure.
  - HTML uses SGML syntax:

```
<h1>Endangered Species</h1>
<p class="intro">Why we <i>need</i> to act.</p>
```

- Areas of text are marked up using tags enclosed in <tag>...</tag>
- Tags are nestable
- Tags have key-value attributes

(Markup languages aren't really "languages", rather they are structured data.)

HTML Explained: Essential Tags

HTML documents consist of an all-encompassing `<html>` tag. The logical tree of tags found inside this is referred to as the Document Object Model (DOM)

Within this, the document is divided into the `<head>` and the `<body>`:

```
<html>
    <head>
        <title>My HTML Page</title>
    </head>
    <body> Hello World </body>
</html>
```

HTML Explained: Essential Tags

The HTML header generally contains information about the page that isn't directly rendered:

- CSS
- JavaScript
- Metadata (`title`, `refresh`, etc.)

We will cover CSS in greater detail later this lecture.

HTML Explained: Essential Tags

The body includes most of the content that is seen by the user of the web browser. Tags can be used for formatting and styling, but this is increasingly the role of CSS.

- `h1` - Header
- `p` - Paragraph
- `ul, ol, li` - Unordered and ordered lists
- `table, tr, td` - Tabular information
- `span` - An inline grouping mechanism
- `div` - A block-level grouping mechanism

HTML Explained: Essential Tags

The body will also include Anchor tags. The anchor tag is one of the original, central innovations of the world wide web.

```
You can search <a href="http://www.google.com">here</a>.
```

The links that are created by anchor tags make HyperText.

HTML Explained: Essential Tags

HTML forms

- Author specifies a series of input elements to be presented to the user
- The form is submitted to the specified `action`, generally as a `POST`
  - `Method` doesn't have to be `POST`
  - `Encoding defaults "application/x-www-form-urlencoded"`

```
<form action="/communities" method="post">
  <label for="cn">Name</label><br>
  <input type="text" name="community" id="cn">
  <input type="submit" name="commit" value="Submit">
</form>
```

HTML Explained

HTML elements have some important attributes:

- `id` - A unique identifier can be assigned to a DOM element.
- `class` - Multiple classes can be assigned to DOM elements. There is a many-to-many relationship between classes and DOM elements.

```
<span class="alert,loud" id="flash_message">Error.</span>
```

Classes and IDs can be used to refer to DOM elements by CSS and JavaScript.

Intro to CSS

- Inside the HTML head, we can define styles using the `<style>` tag.
  - (As we will later see, we generally don't style with style tags)

```
<html>
    <head>
        <style>
            h1 {color: blue;}
        </style>
    </head>
    <body>
        <h1> Hello World </h1>
    </body>
</html>
```

Intro to CSS

- We can style DOM elements in a variety of ways

```
<html>
    <head>
        <style>
            #header {color: blue;}
        </style>
    </head>
    <body>
        <span id="header"> Hello World </span>
    </body>
</html>
```

Intro to CSS

- We can style DOM elements in a variety of ways

```
<html>
    <head>
        <style>
            .alerting {color: red;}
        </style>
    </head>
    <body>
        <span class="alerting"> Hello World </span>
    </body>
</html>
```

Intro to CSS

- We can style DOM elements in a variety of ways

```
<html>
    <head>
        <style>
        </style>
    </head>
    <body>
        <span style="color: red;"> Hello World </span>
    </body>
</html>
```

Intro to CSS

- If we can style in many ways, we can style in contradictory ways
  - What color will the text be rendered below?

```
<html>
    <head>
        <style>
            span {color: blue;}
            .a {color: yellow;}
            #b {color: green;}
        </style>
    </head>
    <body>
        <span class="a" id="b" style="color: red;"> Hello World </span>
    </body>
</html>
```

Intro to CSS

- The precedence order is somewhat complex, but in general…
  - More specific has higher precedence than less specific
    - Example: A style applied to an id-specified `img` tag takes precedence over one applied to all `img` tags
  - Styles specified in the markup itself via the style attribute are higher than those specified in separate CSS files
  - The `!important` annotation can be used to increase the precedence of a CSS rule.

Intro to CSS

Styling is not generally done in-line, either on an element or elsewhere in the markup using the <style> tag

Instead, we serve these styles as separate resources and indicate their location to the browser with a link tag:

- `<link rel="stylesheet" href=”styling.css” type="text/css" >`

Why is this better?

Intro to CSS

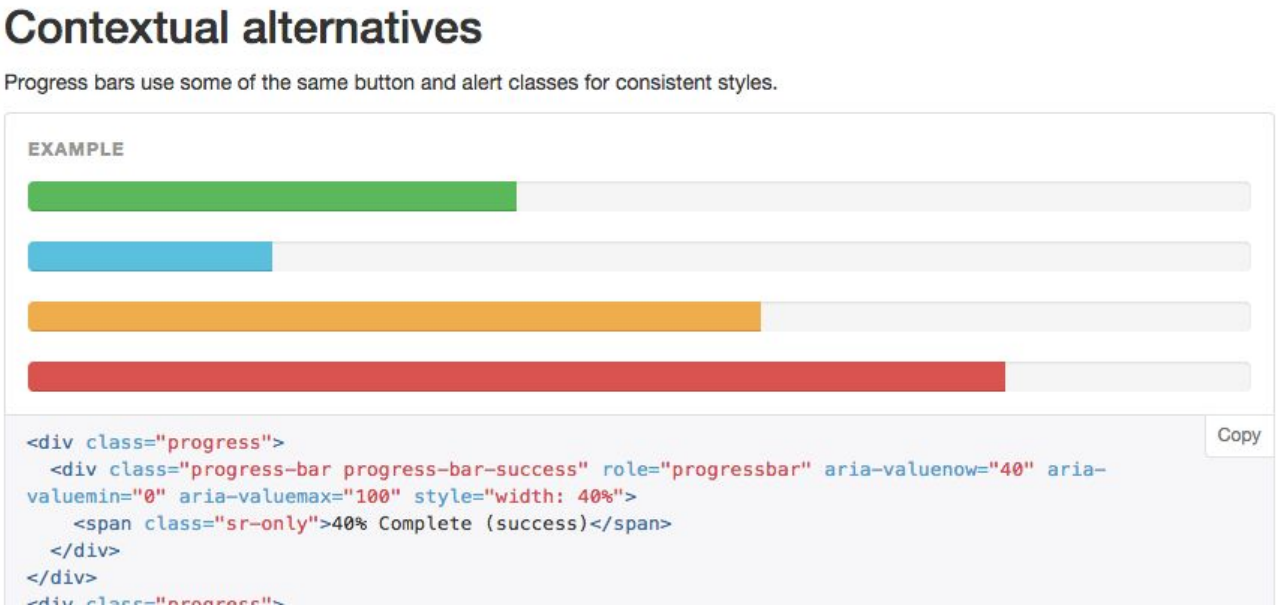
CSS styling can give you limitless control over the appearance of a web application.

While you will use CSS to style your project in this class, CSS really falls outside the core of the course

- Leaning on libraries like Twitter Bootstrap is recommended

Intro to CSS

Learn more about Bootstrap: http://getbootstrap.com/



Contextual alternatives

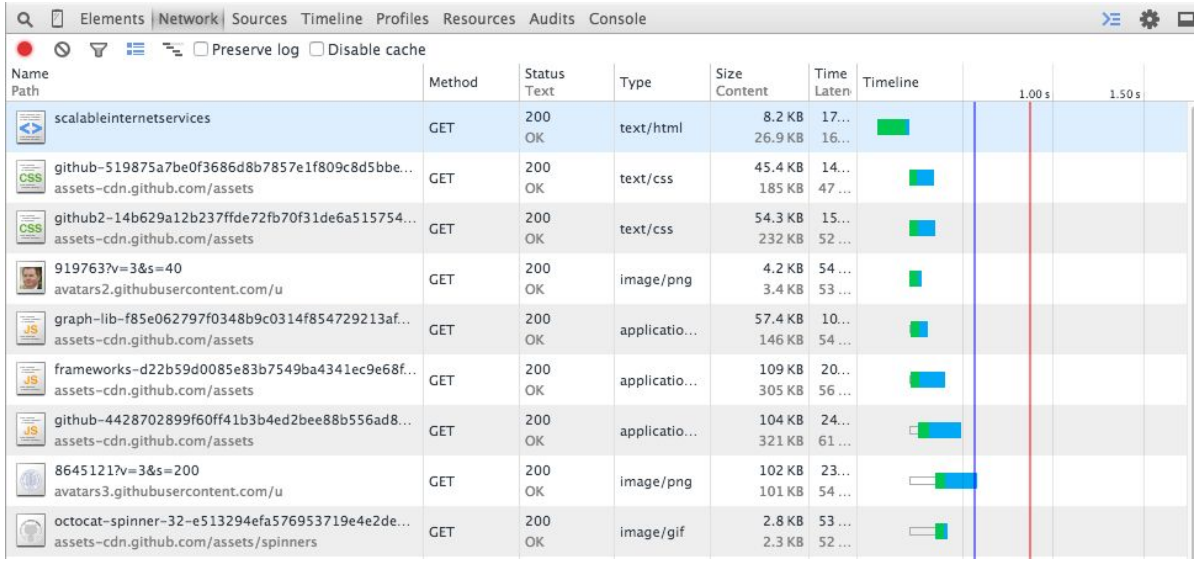
Progress bars use some of the same button and alert classes for consistent styles.

EXAMPLE

Copy

```
<div class="progress">
  <div class="progress-bar progress-bar-success" role="progressbar" aria-valuenow="40" aria-
valuemin="0" aria-valuemax="100" style="width: 40%">
    <span class="sr-only">40% Complete (success)</span>
  </div>
</div>
```

Conclusion



After today, you should have all the basic tools you need to understand output like this

For Next Time...

For Thursday:

- Read and do ch. 1 and 2 from Rails book, get Rails setup
- Continue Ruby Code Academy (finish it by next week)
    - http://www.codecademy.com/en/tracks/ruby/