Web Programming

Robert M. Dondero, Ph.D.
Princeton University

Objectives

- We will cover:
  - The technologies that are at the foundation of web programming...
  - The hypertext transfer protocol (HTTP)
  - The hypertext markup language (HTML)

HTTP and HTML

- **Who**: Tim Berners-Lee
- **When**: 1990
- **Where**: CERN
- **Why**: Allow CERN researchers to share documents

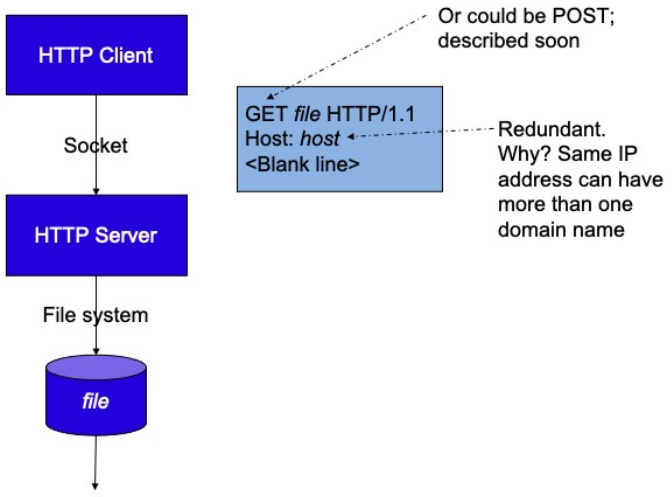
Agenda

- **HTTP**
- HTTP in COS 333
- HTML
- The World Wide Web

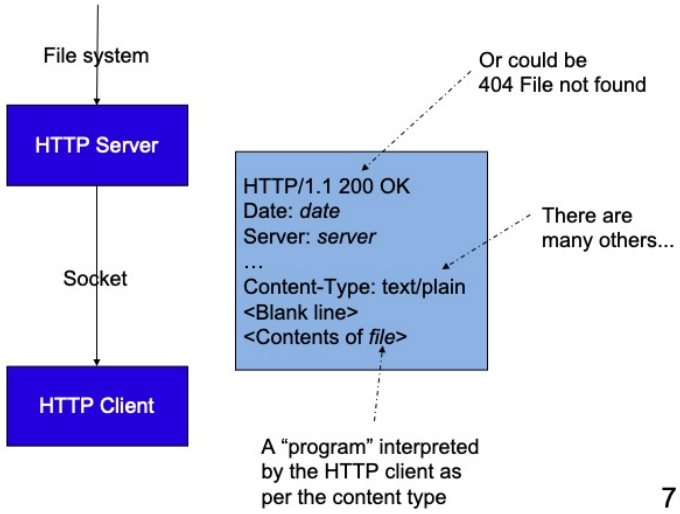
HTTP

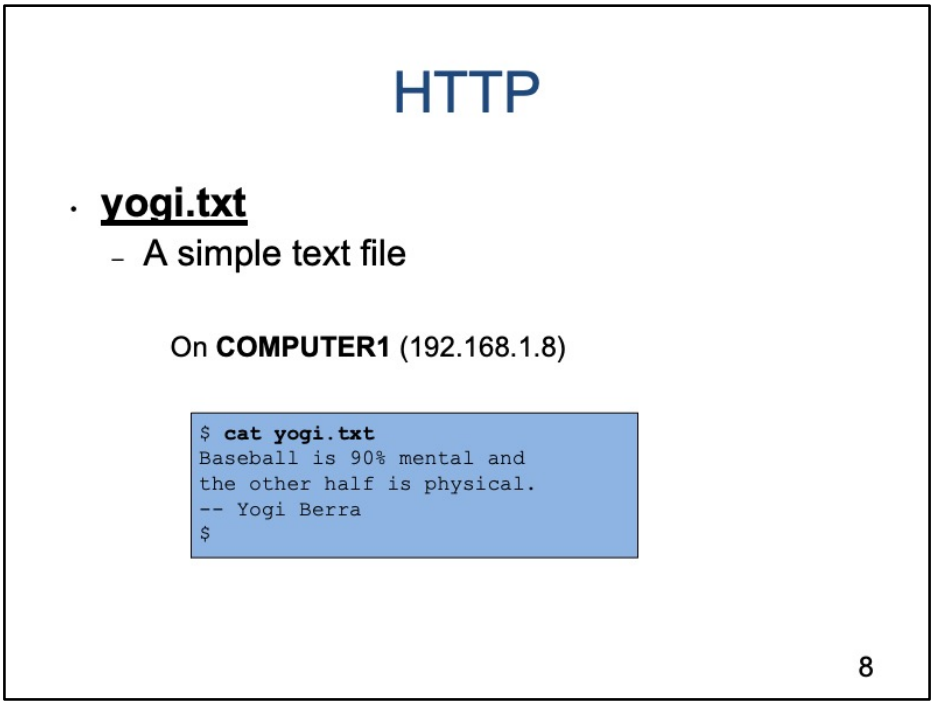
- ***Hypertext Transfer Protocol (HTTP)***
  - A client/server protocol
  - Server: HTTP (Web) server
  - Client: browser (usually, but not necessarily)
- HTTP client requests a file
- HTTP server provides a file

HTTP


HTTP Client
Socket
HTTP Server
File system
file
Or could be POST; described soon
GET file HTTP/1.1
Host: host
<Blank line>
Redundant. Why? Same IP address can have more than one domain name

HTTP


File system
HTTP Server
Socket
HTTP Client
Or could be
404 File not found
HTTP/1.1 200 OK
Date: *date*
Server: *server*
…
Content-Type: text/plain
<Blank line>
<Contents of *file*>
There are
many others...
A “program” interpreted
by the HTTP client as
per the content type



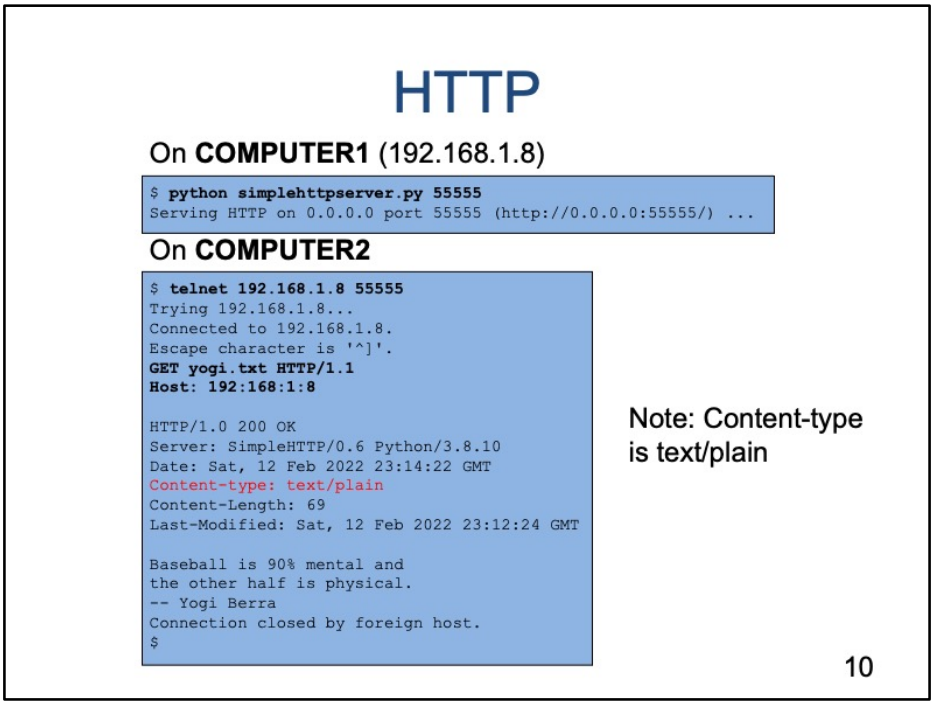
HTTP

An example of using HTTP…

[see slide]

HTTP

- **simplehttpserver.py**
  - A simple HTTP server
  - Enhancement of “standard” Python HTTP server
  - Assume that it runs on **COMPUTER1** (192.168.1.8)
  - Assume that it has access to **yogi.txt**

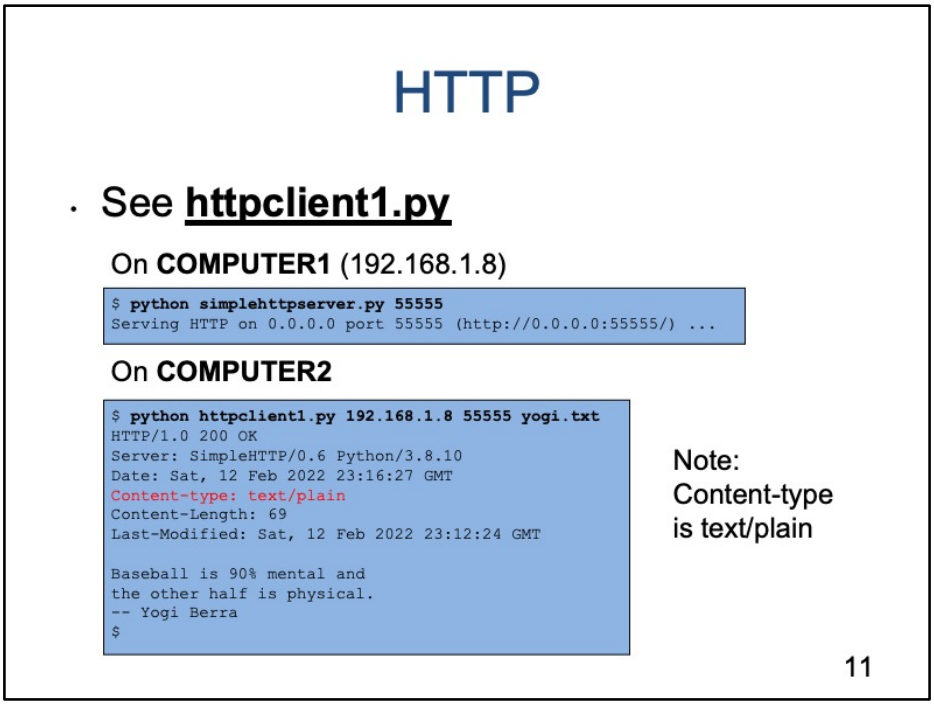


HTTP

From a different computer, we can contact the HTTP server to fetch the file

[see slide]

The server could be simplehttpserver.py
The client could be **telnet**



HTTP

[see slide]

The server could be simplehttpserver.py
The client could be **httpclient1.py**

Code notes: httpclient1.py:

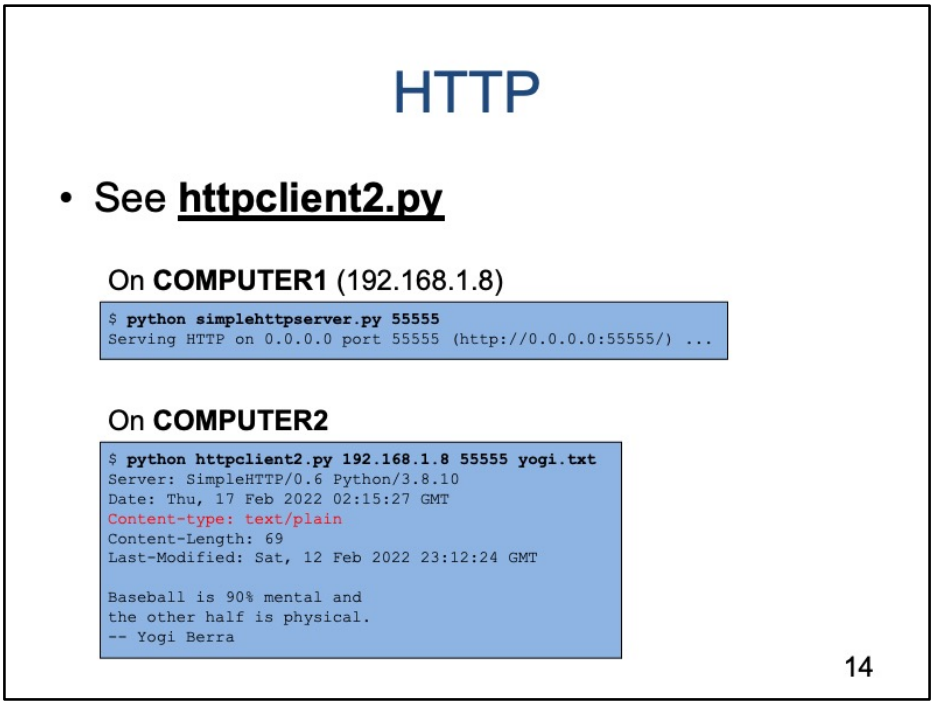
- Create socket
- Create flo for writing
  - Header encoding is iso-8859-1
- Write three lines
  - Must use MS Windows end-of-line marks!
- Create flo for reading
  - Read all header lines
  - Close the flo
- Create flow for reading
  - Read file content
  - Close the flo
- Close socket (via with statement)

Aside: URLs

- ***Uniform Resource Locator (URL)***
  - Format: `protocol://host:port/file`
  - **`protocol`**
    - We'll use `http` now, `https` later
    - Others: `file`, `ftp`, `mailto`, …
    - See http://en.wikipedia.org/wiki/URI_scheme
  - **`host`**
    - IP address or domain name of HTTP server
    - Recall *Network Programming* lecture

Aside: URLs

- Uniform resource locator (cont.)
  - Format: `protocol://host:port/file`
  - **`port`**
    - The port at which the HTTP server is listening
    - Recall *Network Programming* lecture
    - For HTTP, default port is 80
    - For HTTPS, default port is 443
  - **`file`**
    - The path name of the file that the HTTP server should deliver
    - Default path name is specified in HTTP server settings
      - Often index.html or index.php



HTTP

The server could be simplehttpserver.py
The client could be **httpclient2.py**
Which uses a URL to identify the server host, port, and file

[see slide]

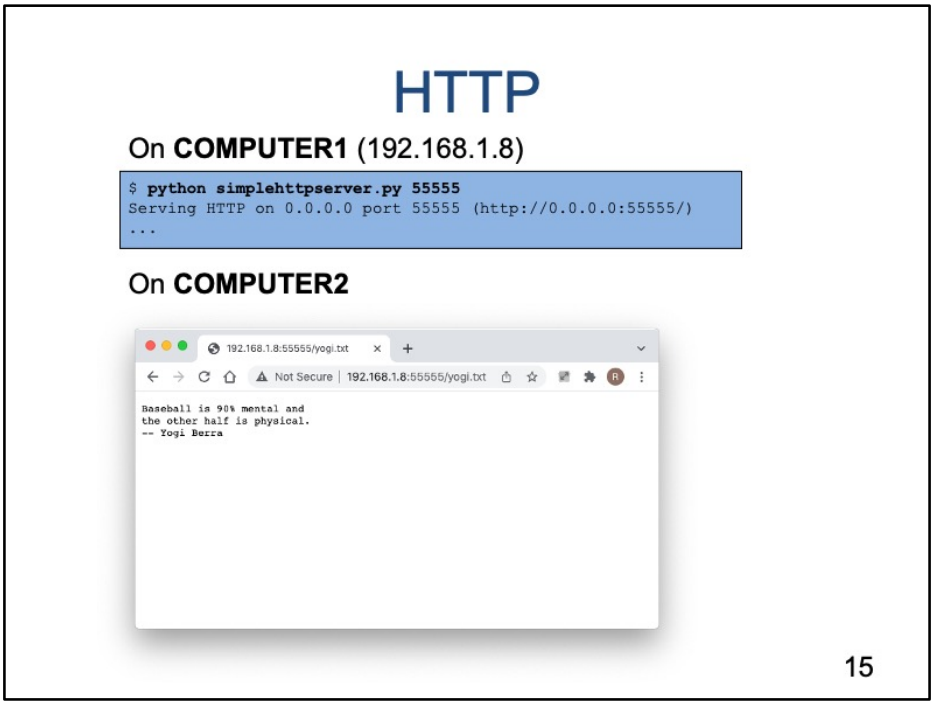
Code notes:

`urlopen()`

- Accepts URL
- Creates socket
- Creates HTTP request, and writes it to socket
- Reads HTTP response headers from socket
- Returns flo

Can ask flo for HTTP response headers via flo.read()

Can read HTTP response body by reading from flo



HTTP

The server could be simplehttpserver.py
The client could be a **browser**

[see slide]

Browsers are (elaborate) HTTP clients
Browsers use URLs to identify hosts and ports of HTTP servers, and files to retrieve

HTTP

- **Review…**

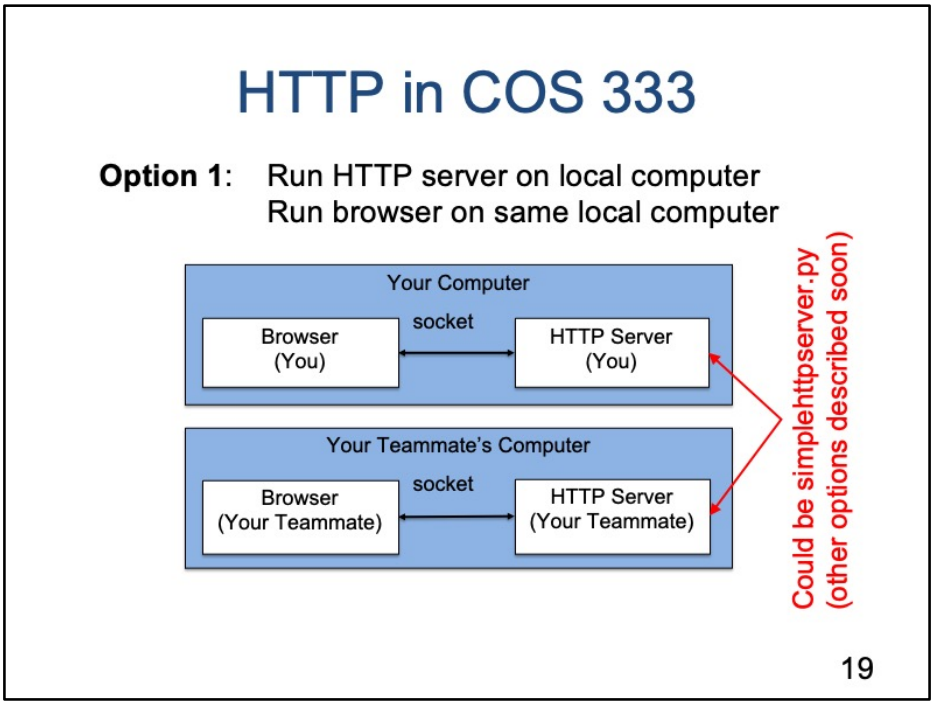
- **Question**: How to issue HTTP request?
- **Answer 1**: Telnet
- **Answer 2**: Your own program
- **Answer 3**: Browser
  - Enter appropriate URL

Agenda

- HTTP
- **HTTP in COS 333**
- HTML
- The World Wide Web

HTTP in COS 333

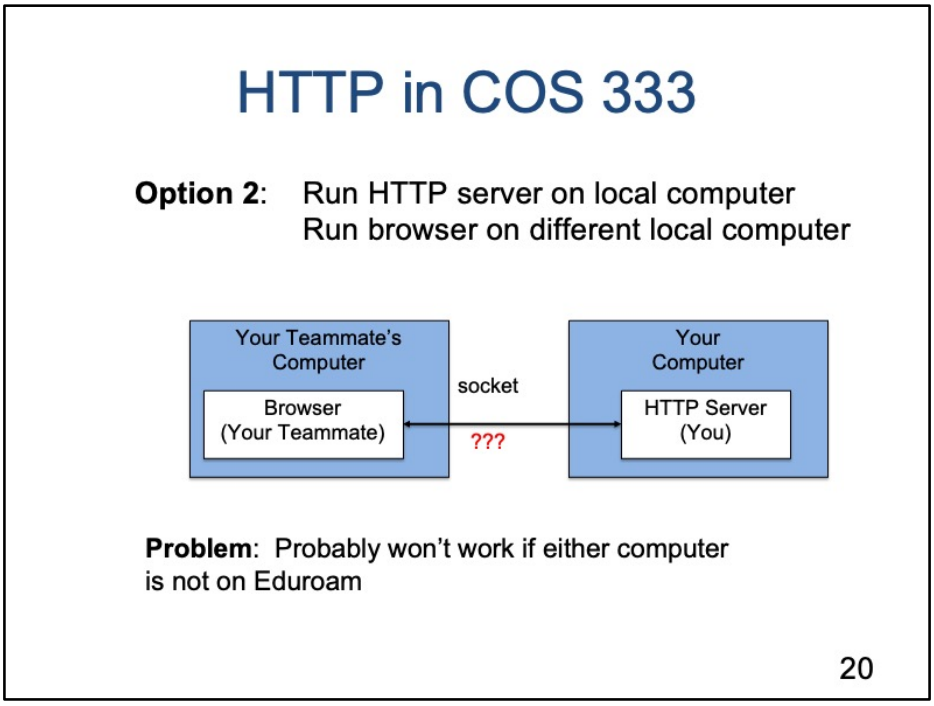
- In the context of Assignment 3...



HTTP in COS 333

[see slide]

You run HTTP server & browser on your computer
Your teammate runs HTTP server & browser on his/her computer

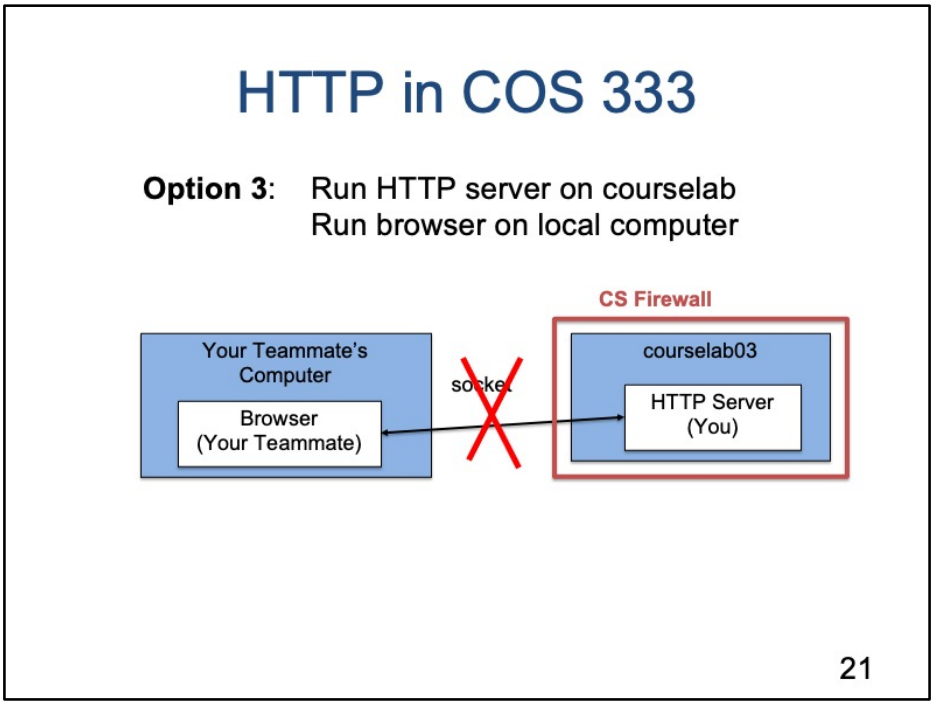


HTTP in COS 333

[see slide]

You run HTTP server on your computer
You tell your teammate:
    IP addr of your computer
        Use `ifconfig` (Mac/Linux) or `ipconfig` (MS Windows) to find out
    Port at which HTTP server is listening
Your teammate runs browser on his/her computer
Could switch roles

Problem:
    Probably not OK if either computer is not on Eduroam
Solution:
    Reconfigure router(s)
    (Not recommended)

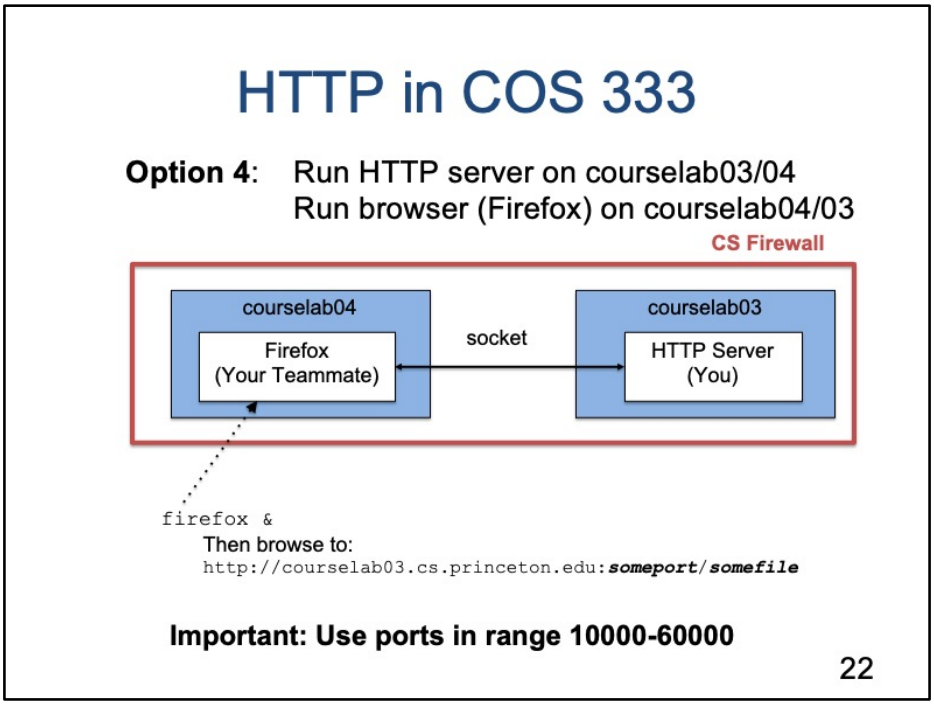


HTTP in COS 333

[see slide]

You run HTTP server on courselab03
You tell your teammate
    Which computer is running HTTP server
    Port at which server is listening
Your teammate runs browser on his/her computer
Could use courselab04; could switch roles

Problem: firewall
    Wikipedia: "In computing, a ***firewall*** is a network security system that monitors and controls incoming and outgoing network traffic based on predetermined security rules."
    Wikipedia: "A firewall typically establishes a barrier between a **trusted internal network** and **untrusted external network**, such as the Internet."
    Browser and HTTP server cannot penetrate CS firewall



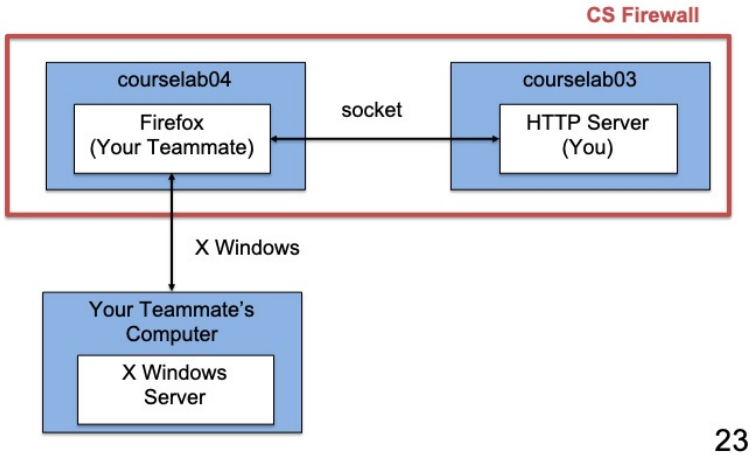
HTTP in COS 333

[see slide]

You run HTTP server on courselab03
You tell your teammate:
    Which computer is running HTTP server
    Port at which server is listening
        Important: use ports in range 10000-60000
Your teammate runs Firefox on courselab04
Could switch computers; could switch roles

HTTP in COS 333

Option 4 note: X Windows can cross firewall



HTTP in COS 333

- Suggestion:
  - Use option 1 (browser & server on same local computer) during development
  - If possible, use option 2 (browse & server on different local computers) to test network comm
- Requirement:
  - Use option 4 (browser & server on different courselab computers) to test network comm, and to make sure that HTTP server runs on courselab

Agenda

- HTTP
- HTTP in COS 333
- **HTML**
- The World Wide Web

HTML

- Some HTTP content types:
  - **text/plain**, **text/html**, image/gif, image/jpeg, audio/mp4, **application/xml**, **application/json**, …
- Complete list of HTTP content types:
  - http://en.wikipedia.org/wiki/Internet_media_type
- The most popular HTTP content type:
  - **text/html**

HTML

- ***Hypertext Markup Language (HTML)***
  - A language for expressing documents
- HTML document
  - Consists of plain text
- HTML text
  - Contains unadorned text and ***markup***

HTML

- ***HTML markup***
  - ***Elements*** (by example):
    - ****some text****
      - A normal element
      - Delimited by a ***start tag*** and an ***end tag***
    - <a **href="someurl"**>some text
      - An element with an ***attribute***

HTML

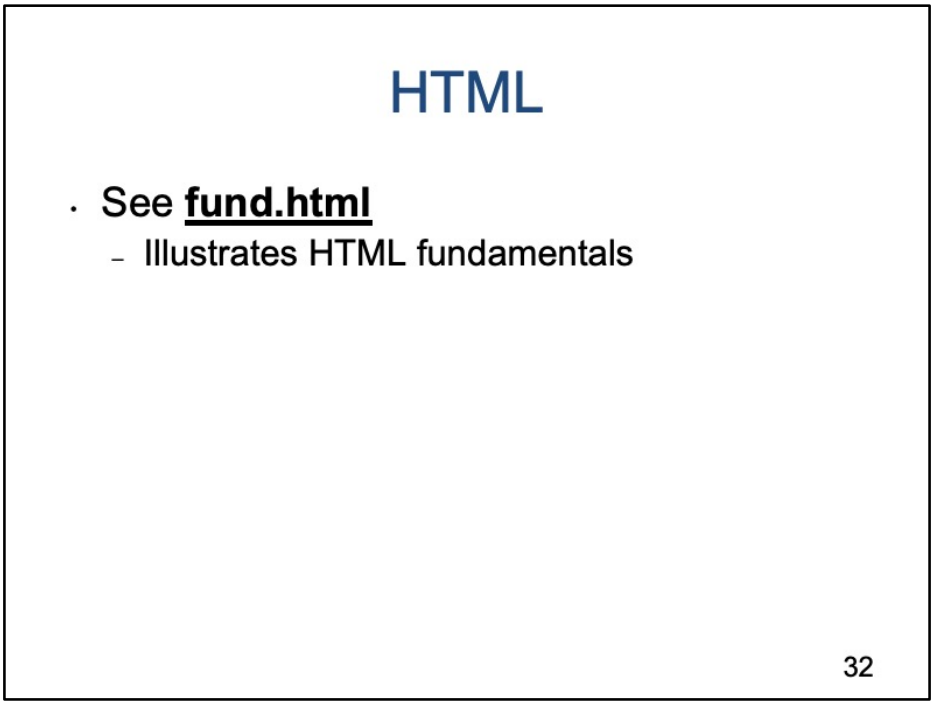
- ***HTML markup*** (cont.)
  - ***Elements*** (by example):
    - `<strong></strong>`
      - An ***empty element***
    - `<hr>`
      - A ***void element***
      - An element that must be empty and must consist of a start tag only
      - Disallowed by some “relatives” of HTML
    - `<hr/>`
      - A ***self-closing tag***
      - A void element
      - Allowed by HTML and all “relatives”

HTML

- ***HTML markup*** (cont.)
  - ***Processing instructions***
    - `<!DOCTYPE html>`
      - A ***DOCTYPE*** processing instruction
        - » First line of document
        - » Identifies this document as an HTML 5 document
    - `<!-- This is a comment -->`
      - A ***comment***

HTML

- ***HTML markup*** (cont.)
  - Uses finite specified set of elements
  - Uses finite specified set of attributes for each element
  - Describes presentation
    - How the text should be presented/rendered
  - Tags and attribute names are case insensitive
- We'll use HTML 5
  - See Appendix for history



HTML

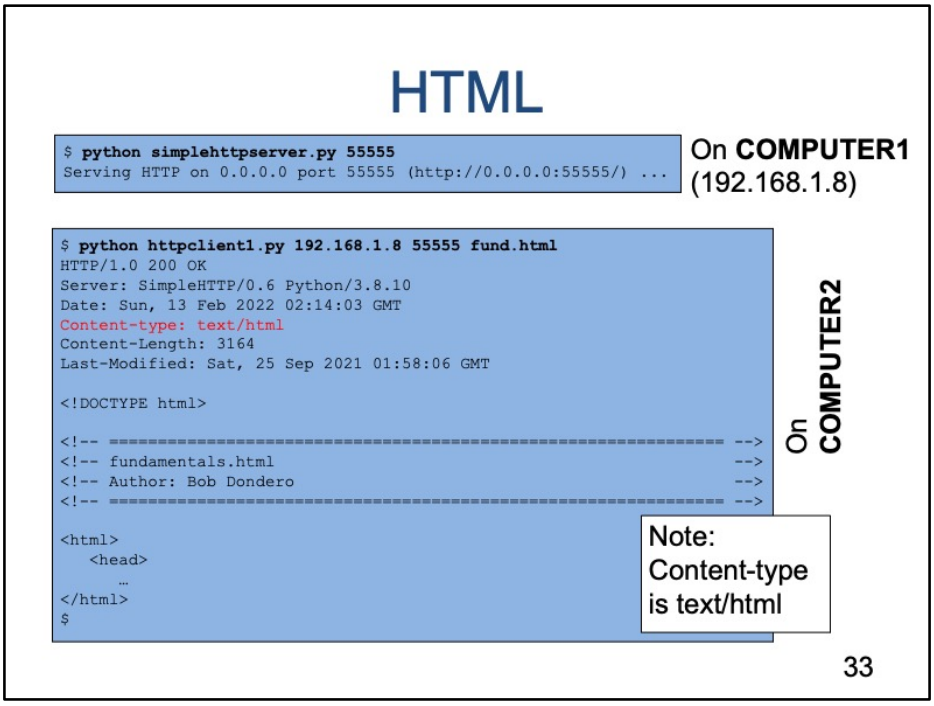
An example of an HTML document
Illustrates HTML fundamentals

[see slide]

Code notes:

- Document structure
  - DOCTYPE processing instruction
  - Html element
  - Head element
  - Body element
- Comments
- Heading elements
- Horizontal rule element
- Container elements
- Paragraphs
- Physical character formatting elements
- Logical character formatting elements
- Character entities

Lists
Tables

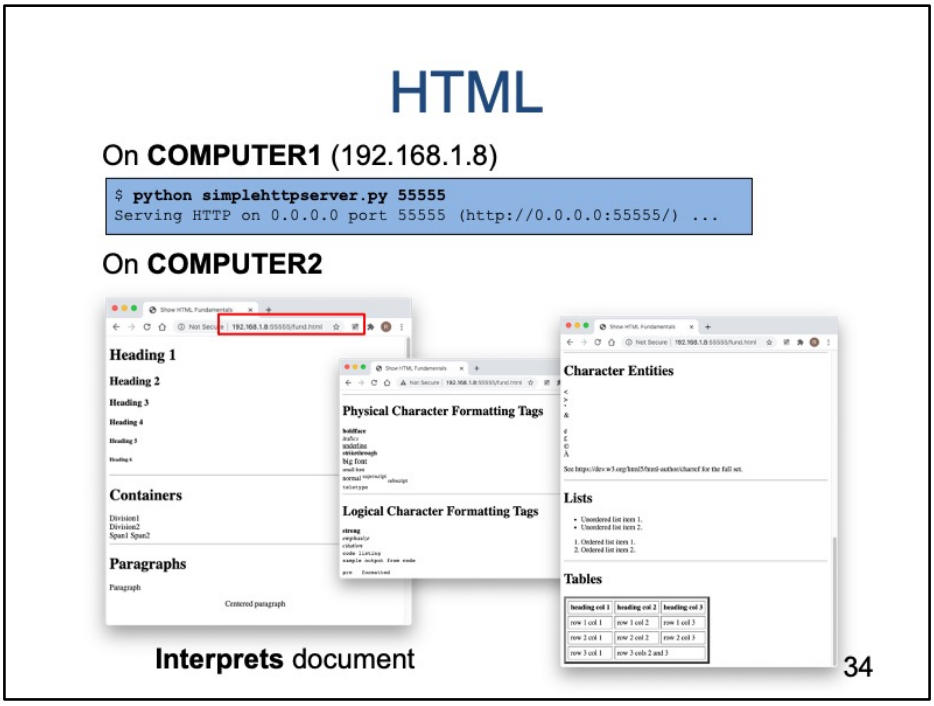


HTML

[see slide]

The server could be simplehttpserver.py on 192.168.1.8
at port 55555
The client could be **httpclient1.py**

httpclient1.py simply writes the document to stdout
Note that the content type of the HTTP response is text/html



HTML

[see slide]

The server could be simplehttpserver.py at port 5555
The client could be a **browser**

Because the content type is text/html, the browser **interprets** the document

HTML

[see slide]

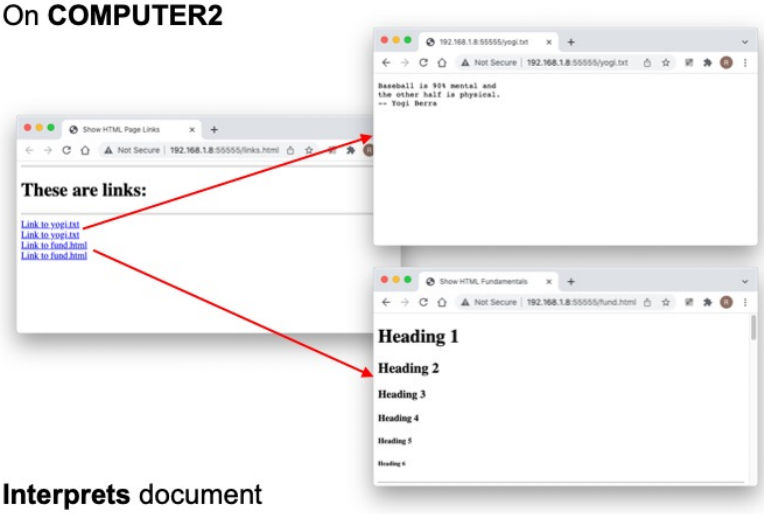
Code notes:

`<a href="someurl">…</a>`

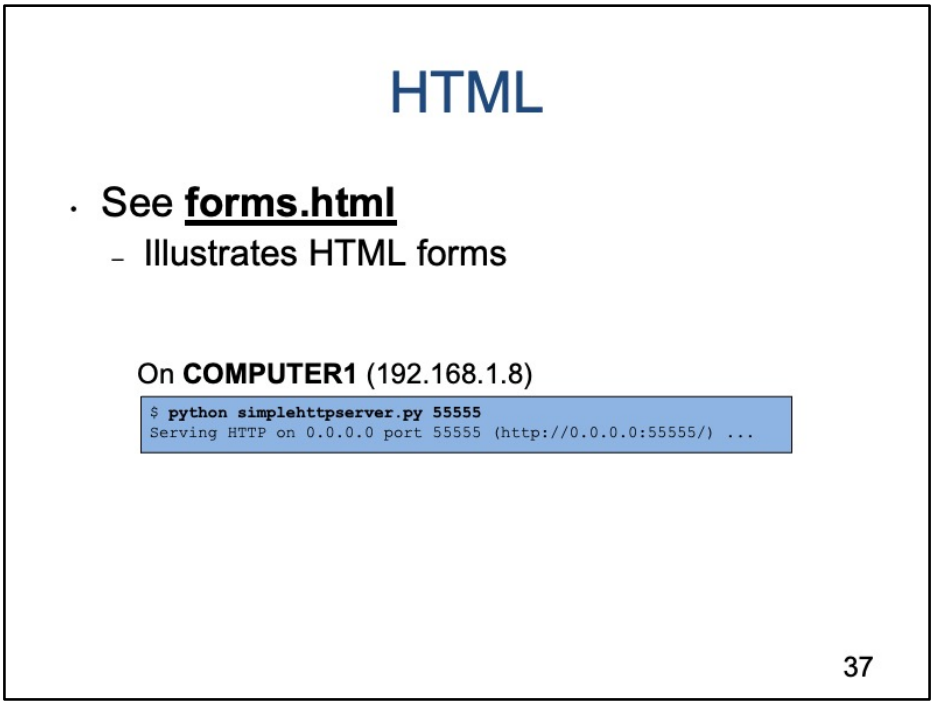
Defines a ***page link*** (alias ***link***)
Browser renders as hyperlink
User clicks on link => browser sends request to server at specified host at specified port for specified file
Abbreviated URL => protocol, host, and port default to protocol, host, and port that caused delivery of *this* page

HTML

On **COMPUTER2**



Interprets document



HTML

[see slide]

Code notes:

`<form action="someurl">…</form>`

Defines a ***form***

Browser does not render

`<input type="submit" value="somelabel">`

Often nested within `form` element

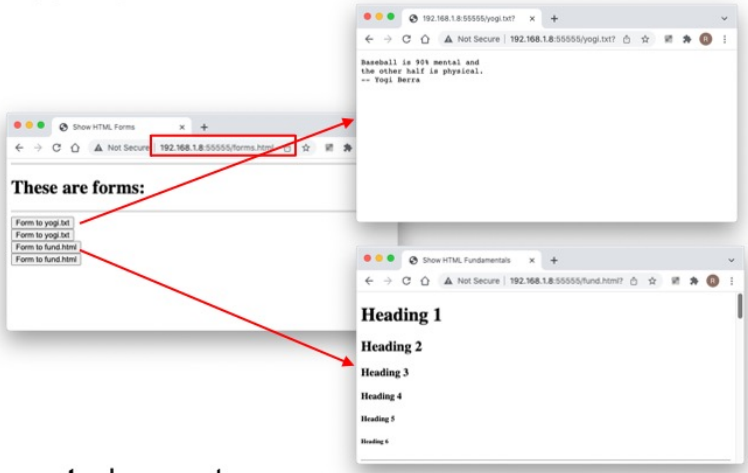
Browser renders as a button whose label is `somelabel`

User clicks on button => browser sends request specified by form `action` attribute

Can use abbreviated URLs

HTML

On **COMPUTER2**



Interprets document

HTML

- **Review**…

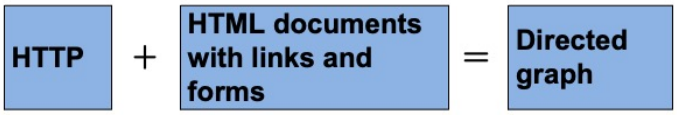
- **Question**: How to issue HTTP request?
- **Answer 1**: Telnet
- **Answer 2**: Your own program
- **Answer 3**: Browser
  - Enter URL
  - Click on HTML link
  - Click on HTML form input element of type submit

Agenda

- HTTP
- HTTP in COS 333
- HTML
- **The World Wide Web**

The World Wide Web

- Note:
  - Link or form could specify another HTML doc
  - And so…



The World Wide Web

- The ***World Wide Web***
  - A directed graph
    - Nodes: HTML docs
    - Edges: links and forms
  - HTML pages are:
    - Identified by URLs
    - Communicated via HTTP
      - Requested by browsers
      - Sent by HTTP servers
      - Received and interpreted by browsers

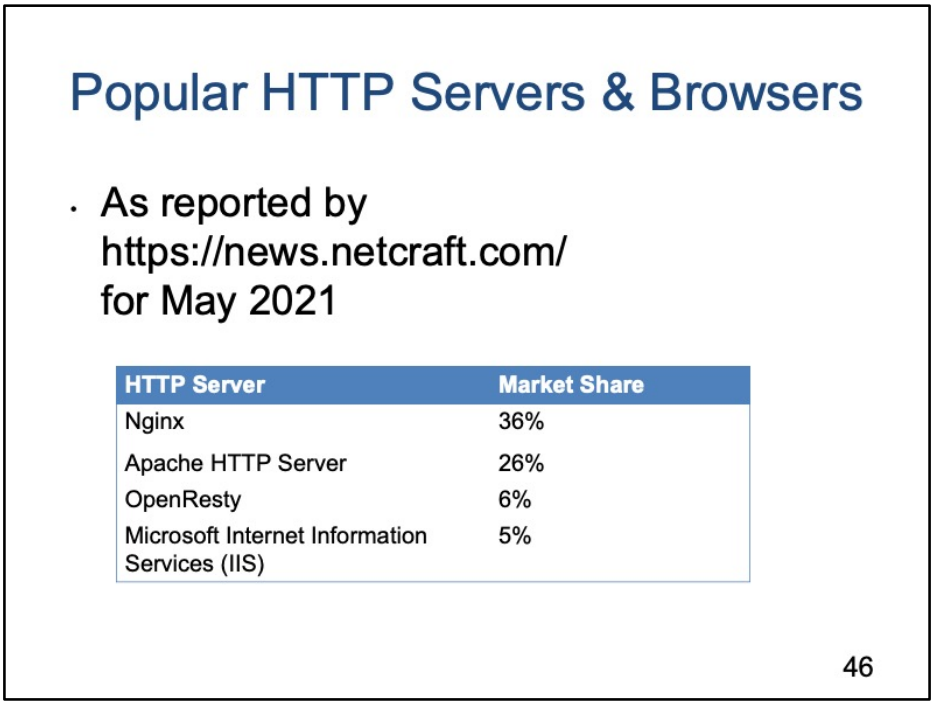
Aside: Examining Uninterpreted HTML

- Each browser provides a way to examine raw HTML document
  - Chrome & firefox: right click → view page source
  - Good learning/debugging tool

Summary

- We have covered:
  - The technologies that are at the foundation of web programming...
  - The hypertext transfer protocol (HTTP)
  - The hypertext markup language (HTML)
- See also:
  - **Appendix 1**: Popular HTTP Servers & Browsers
  - **Appendix 2**: HTML History

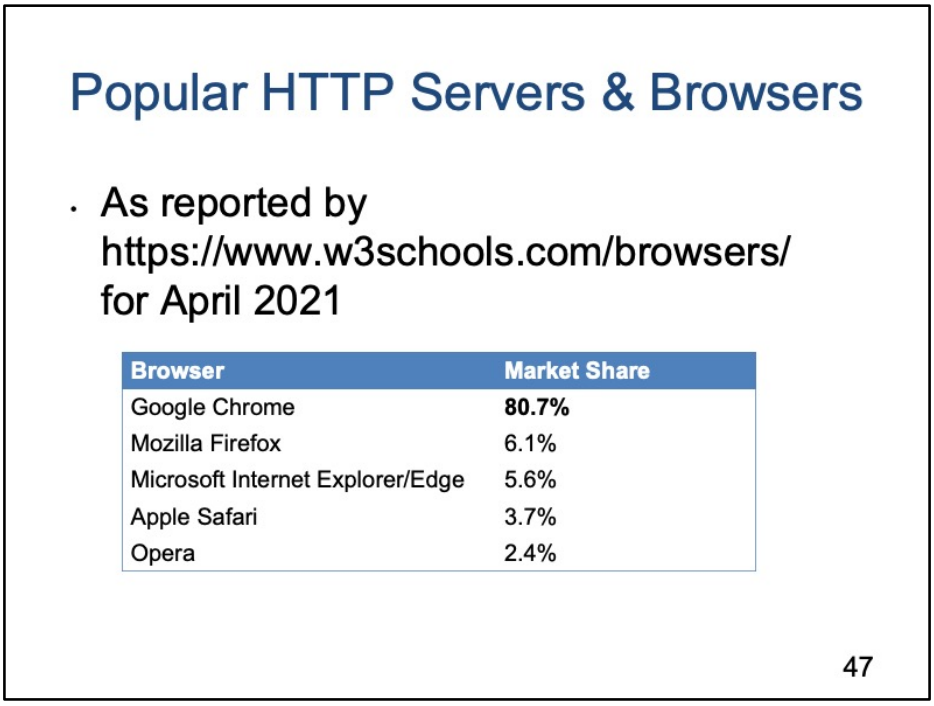
Appendix 1: Popular HTTP Servers & Browsers



Popular HTTP Servers and Browsers

[see slide]

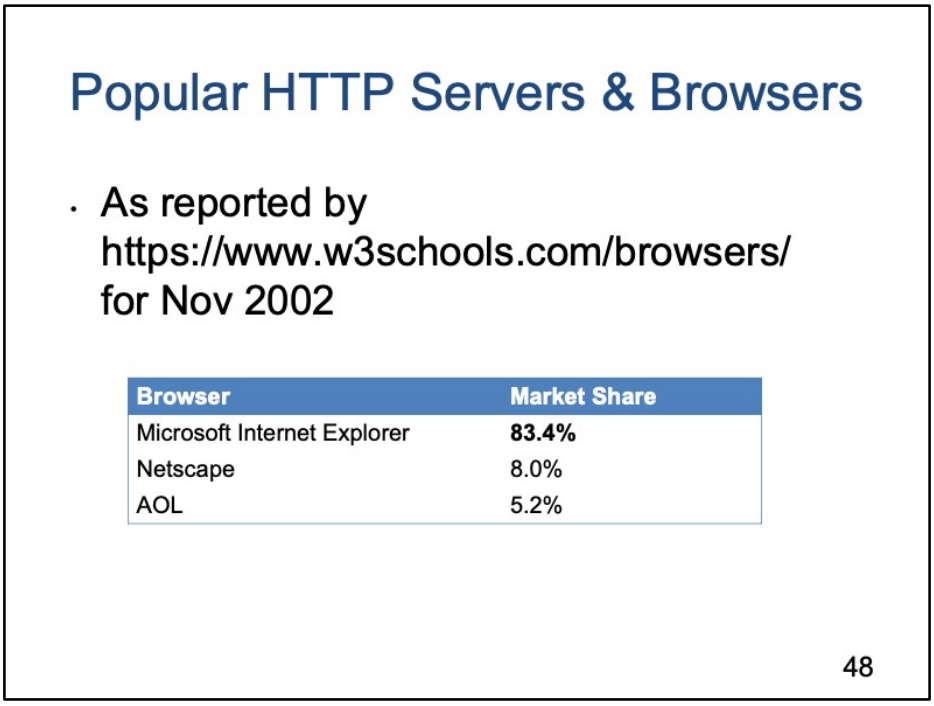
Princeton's CS Department runs an Apache HTTP Server
It serves the Survey and ProjectFinder apps



Popular HTTP Servers and Browsers

[see slide]

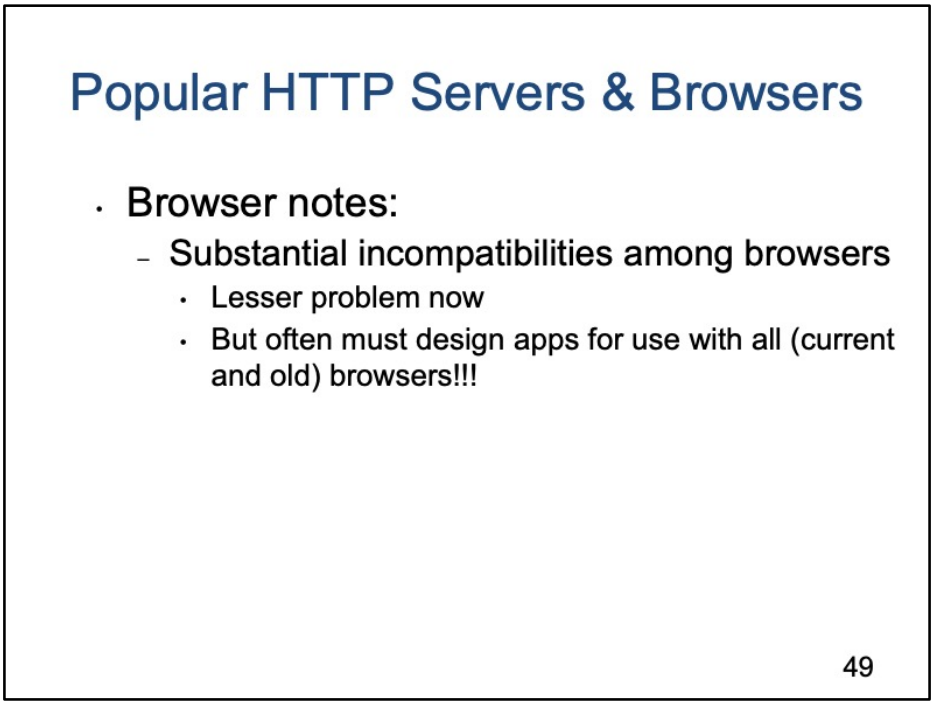
Google Chrome is amazingly dominant
Microsoft IE/Edge has only a 5.6% market share



Popular HTTP Servers and Browsers

[see slide]

Microsoft used to have a 83.4% market share!



Popular HTTP Servers and Browsers

Browser notes:

There are (were!) substantial incompatibilities among browsers

When Microsoft dominated the market, it didn't feel obliged to conform to standards

Lesser problem now

But often must design apps for use with all (current and old) browsers!!!

Appendix 2:
HTML History

HTML History

- ***Structured Generalized Markup Language (SGML)***
  - A language for expressing documents
- SGML document
  - Consists of plain text
- SGML text
  - Contains ***markup***

HTML History

- ***SGML markup***
  - `<tag>...</tag>`
  - `<tag attribute="value" …>... </tag>`
  - `<tag />`
- Tags and attributes can be anything you want!

HTML History

- ***Document type definition (DTD)***
  - A specification of allowable tags and attributes (and much more)
- Typically:
  - SGML user group (e.g., pharm industry, drug regulatory agencies) composes DTD
  - SGML users (e.g., pharm companies) compose documents that conform to the DTD
- First line of SGML doc specifies DTD

HTML History

- HTML
  - 1990
  - Intended to be an application of SGML, but…
  - At the time no clear parsing guidelines were established, so…
  - Many HTML documents are not valid SGML documents

HTML History

- HTML 2.0
  - 1995
  - First version to be standardized
  - First line of document:
    - `<!DOCTYPE html PUBLIC "-//IETF//DTD HTML 2.0//EN">`

HTML History

- HTML 3.2
  - 1997
  - More of an SGML application, but...
  - Burdened by need for backward compatibility
    - Still had many legacy features that differ from SGML's requirements
  - First line of document:
    - `<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">`

HTML History

- HTML 4.0
  - 1997
  - Two versions:
    - Strict: deprecated elements are forbidden
    - Transitional: deprecated elements are allowed
  - With strict DTD, an SGML application
    - Conforms to ISO 8879 – SGML

HTML History

- First line of document:
  - `<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">`
  - `<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">`

HTML History

- HTML 5
  - 2014
  - Abandons any attempt to define HTML as SGML application
  - Explicitly defines its own syntax rules
  - More closely match existing implementations and documents
  - First line of document:
    - `<!DOCTYPE html>`

HTML History

- We'll use HTML 5
  - But we'll keep it simple
    - This course is not about HTML