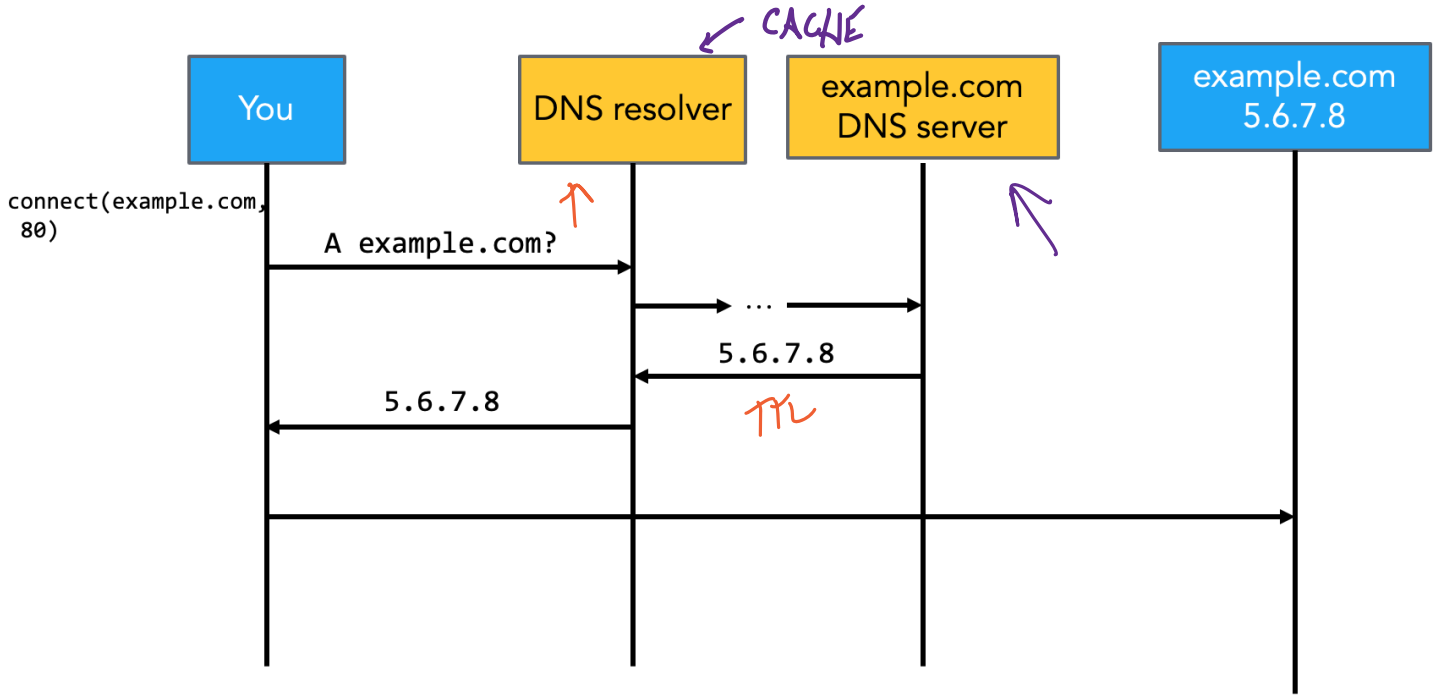
Lecture 19

Today
- DNS record types
- HTTP (browsers, webservers)

Announcements
- TCP milestone II meetings this week
- HW4 (short): out sometime this week
- TCP gearup III: Thursday, 5pm in CIT 165
  => How to test everything

Lecture 19: HTTP I

Warmup: if example.com's DNS server goes down, can another DNS server still resolve example.com?



=> Result will be cached until TTL expires
=> Good practice to have a backup authoritative server (SHOULD have at least 2)

In practice: DNS Record types

Not just about IP addresses!

RR Type	Purpose	Example
A	IPv4 Address	`128.148.56.2`
AAAA	IPv6 Address	`2001:470:8956:20::1`
CNAME	Specifies an alias ("Canonical name")	`systems.cs.brown.edu. 86400 IN CNAME systems-v3.cs.brown.edu.` `systems-v3.cs.brown.edu. 86400 IN A 128.148.36.51`
NS	DNS servers for a domain	`cs.brown.edu. 86400 IN NS br1.brown.edu`
MX	Mail servers	`MX <priority> <ip> (eg. MX 10 1.2.3.4)`
SOA	Start of authority	`Information about who owns a zone`
PTR	Reverse IP lookup	`7.34.148.128.in-addr.arpa. 86400 IN PTR quanto.cs.brown.edu.`
SRV	How to reach specific services (eg. host, port)	`_minecraft._tcp.example.net 3600 SRV <priority> <weight> <port> <server IP>`
TXT	Arbitrary string	. . .

Example TXT records

```
% dig brown.edu TXT @1.1.1.1
;; ANSWER SECTION:
brown.edu. 3600 IN TXT "google-site-verification=9NxSC7hDW1F0dOB7Hp5sj1b4Zh3YuOLsWALzm0W5e3M"
brown.edu. 3600 IN TXT "docusign=084c221f-92f9-4237-beb8-c25279422ee0"
brown.edu. 3600 IN TXT
"atlassian-domain-verification=Hs1rCYbV9DaUuZH78ks2h7V68IDvaPFkmvb6vrxSU/3HXmysLgY5FKVuj1PY0sWT"
brown.edu. 3600 IN TXT
"cisco-ci-domain-verification=122c4cac85430ffd529f4c1f7dbc9a90b3d511cd9486b79c39bf04929a3d24aa"
brown.edu. 3600 IN TXT "airtable-verification=5535dab3fd5abf3d31fb73a24ce236a4"
brown.edu. 3600 IN TXT "miro-verification=72ea693488717c97da4e2a48c89fdfee7e673b54"
brown.edu. 3600 IN TXT "v=spf1 ip4:128.148.0.0/16 ip4:209.235.66.235/32 ip4:74.122.104.0/22"
       " ip4:35.163.186.146 ip4:208.117.49.214 ip4:198.2.128.0/18 ip4:148.105.8.0/21
       ip4:198.105.13.0/24" " ip4:206.107.42.249 ip4:206.107.42.254 ip4:206.107.42.9
       ip4:35.169.11.239 ip4:198.187.196.100 include:_spf.google.com include:sendgrid.net
include:_spf.qualtrics.com include:aspmx.pardot.com ~all"
brown.edu. 3600 IN TXT "github-verification=ekYLSgdWns5XA28K32JpNpD53dWpQEHQAmg5y7BY"

 [ . . . ]
```

TXT records: just arbitrary strings, usually used to validate something about the owner of the domain

Example: SPF: list of valid senders of email for this domain

For Brown, this includes:

- Brown's own IPs
- Gmail's IPs (since use gmail)
- Other apps that are allowed to send mail on behalf of Brown (e.g., Qualtrics survey platform)

(Nick is posting the slide-based version for this introductory part because he thinks it more accurately captures the flow of the lecture vs. the in-class handout version)

HTTP: Hypertext Transfer Protocol

HTTP

"Application protocol for distributed, collaborative hypermedia information systems"

- Fundamental protocol behind "the web"
- Now part of most things we do on the Internet—so much more than web pages

But what is hypertext?

Hypertext

70 languages

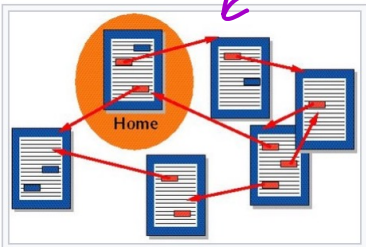
Article Talk

Read Edit View history Tools

From Wikipedia, the free encyclopedia

For the concept in semiotics, see Hypertext (semiotics).

Hypertext is text displayed on a computer display or other electronic devices with references (hyperlinks) to other text that the reader can immediately access.[1] Hypertext documents are interconnected by hyperlinks, which are typically activated by a mouse click, keypress set, or screen touch. Apart from text, the term "hypertext" is also sometimes used to describe tables, images, and other presentational content formats with integrated hyperlinks. Hypertext is one of the key underlying concepts of the World Wide Web,[2] where Web pages are often written in the Hypertext Markup Language (HTML). As implemented on the Web, hypertext enables the easy-to-use publication of information over the Internet.

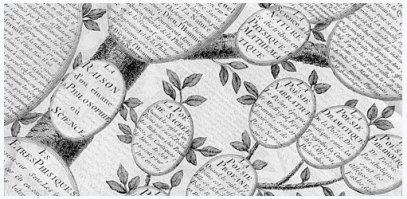


Documents that are connected by hyperlinks

Etymology [edit]

> "(...)'Hypertext' is a recent coinage. 'Hyper-' is used in the mathematical sense of extension and generality (as in 'hyperspace,' 'hypercube') rather than the medical sense of 'excessive' ('hyperactivity'). There is no implication about size— a hypertext could contain only 500 words or so. 'Hyper-' refers to structure and not size."
>
> —Theodor H. Nelson, *Brief Words on the Hypertext*, 23 January 1967

The English prefix "hyper-" comes from the Greek prefix "ὑπερ-" and means "over" or "beyond"; it has a common origin with the prefix "super-" which comes from Latin. It signifies the overcoming of the previous linear constraints of written text.



Information mapping

Topics and fields

Business decision mapping · Data visualization

"As we may think", Vannevar Bush (1945)

"The human mind…operates by association. With one item in its grasp, it snaps instantly to the next … in accordance with some intricate web of trails carried by the cells of the brain"

Defines the "Memex": *"a device in which an individual stores all his books, records, and communications, and which is mechanized so that it may be consulted with exceeding speed and flexibility"*

*HTTP: a protocol for distributing hypertext media (*and now so much more)*

"WEB" OF PAGES!

Enables the World Wide Web (WWW): a distributed database of pages linked through HTTP

… now synonymous with with "The Internet" itself!

A bit of history

- 1990: First HTTP implementation
  - Tim Berners-Lee, CERN
- 1991: HTTP/0.9: Fetching pages
- 1992: HTTP/1.0:
  Client/server information, simple caching
- 1996: HTTP/1.1
  - Extensive caching support
  - Host identification
  - Pipelined, persistent connections, …

STILL MOST WIDELY SUPPORTED!

The first webserver

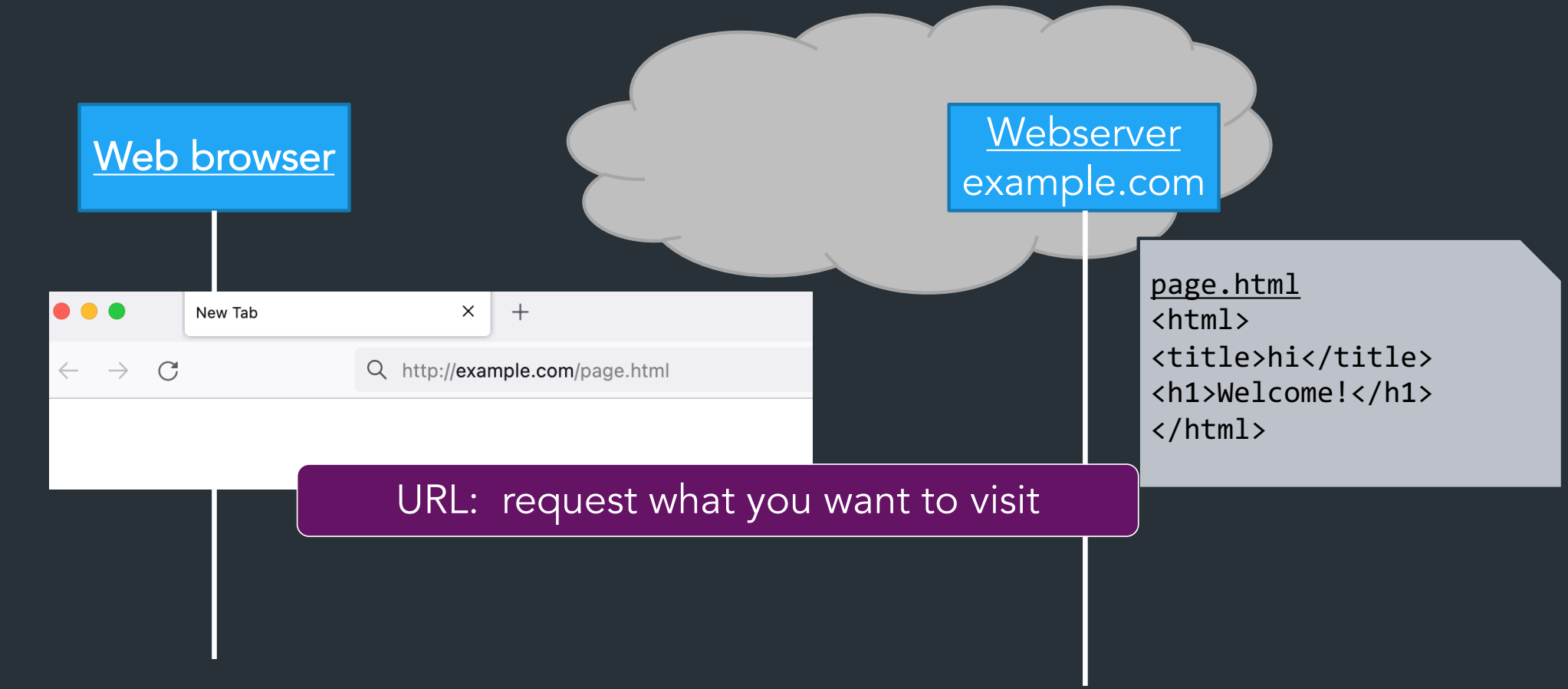
(Lots of development after this (with varying levels of success -- we'll talk more about this next class!)

So what does HTTP actually do?

HTTP: a protocol for distributing hypertext media
*(*and now so much more)*

=> Ways to publish content that clients can fetch

=> Protocol to look up content (eg. from a browser)



=> URL: Uniform resource locator: refers to some specific piece of content a site

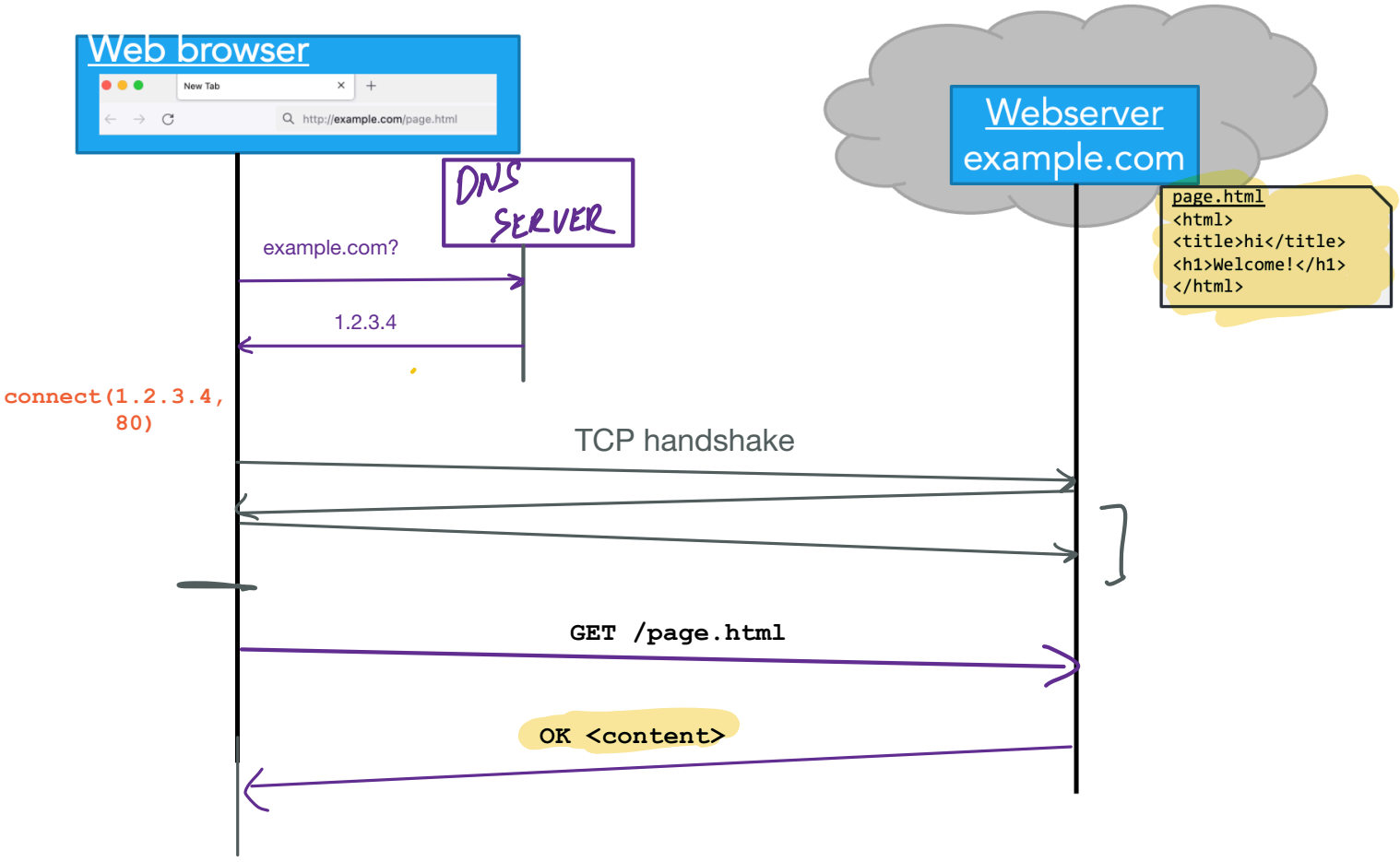
=> Not just connecting to an application, but a specific resource!!

How it works: the big picture

Key facts
- Uses TCP
- Port 80 (by default)

Type URL into browser => go to http://example.com/page.html

But first: what are all the steps we need to do beforehand?



Components of a request

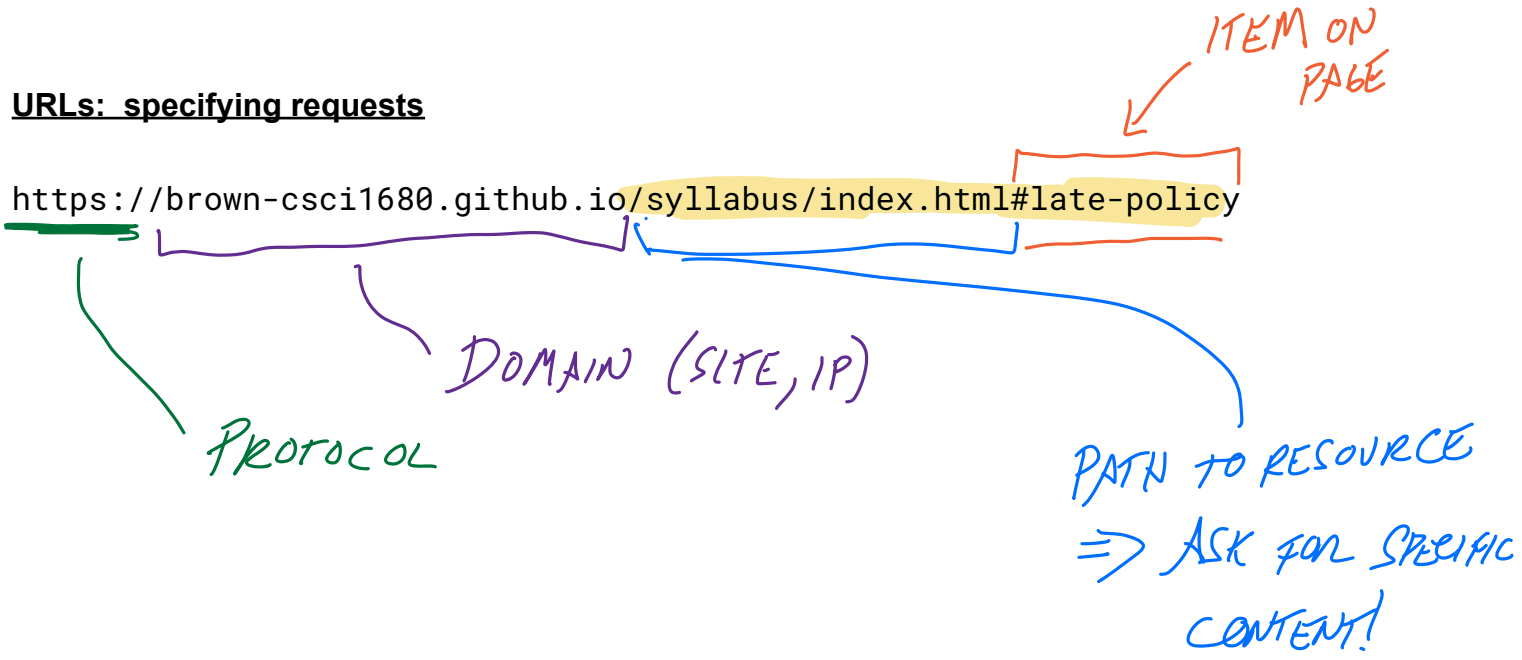
Here's a summary of the terms we discussed throughout the lecture:

- Content: any object (HTML, images, JSON)

- Clients: send requests, receive responses
- Servers: store content, or fetch it (e.g., from a database) or generate it (via some backend app code) somehow

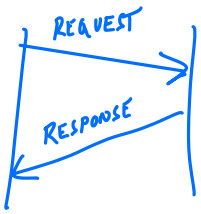
- Request: asking for a resource (page.html)
- Response: content + status code

URLs: specifying requests



Server uses URL parameters to know what resource to fetch

HTTP components



Anatomy of a request

```
> nc cs.brown.edu 80
GET /courses/index.html HTTP/1.0
Host: cs.brown.edu
Content-Type: text/html
Accept-Language: en

HTTP/1.1 200 OK
Date: Thu, 24 Mar 2011 12:58:46 GMT
Server: Apache/2.2.9 (Debian) mod_ssl/2.2.9 OpenSSL/0.9.8g
Last-Modified: Thu, 24 Mar 2011 12:25:27 GMT
ETag: "840a88b-236c-49f3992853bc0"
Accept-Ranges: bytes
Content-Length: 9068
Vary: Accept-Encoding
Connection: close
Content-Type: text/html


<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
...
```

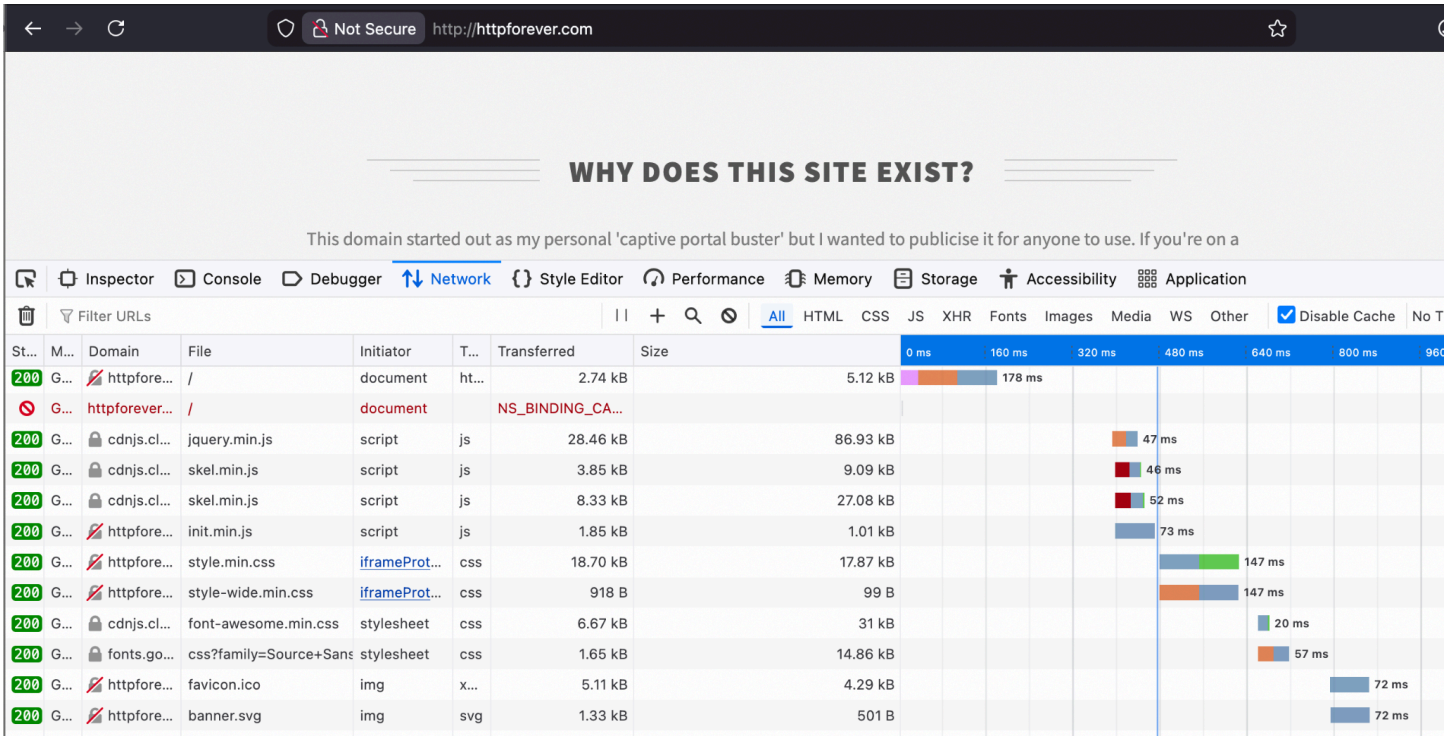

Components of a request

- **Method**: GET, POST, PUT, ...
  => What operation you're doing on the server
  - GET: fetch data
  - POST: I want to make a change on the server (e.g., submit a form)
- **Resource:** /courses/index.html
  - Target of the request (e.g., item to fetch, endpoint to submit something, ...)
- **Headers**: metadata about request or response
  - Examples:
  - User-agent: browser version
  - Language, content-type
  - Cookies (next page)
  - Info about how/if data can be cached (more on this later...)
- **Body**: the actual content
- **Status code:** indicates success or error (extra notes at end, or look up "HTTP status codes"

How do websites (and browsers) work?

- Visit some URL (e.g., http://httpforever.com)
- Browser fetches some initial page (usually, index.html if not specified)
- Initial page contains URLs for other resources (images, Javascript, CSS stylesheets, etc.)
  => Browser also fetches these => a recursive process!
- Pages have links that go to other pages => repeat process when clicking a link

Can view requests in your browser using "Developer Tools" (or right-click => "Inspect element")
=> To see a timeline of requests, and the content of requests, go to the "Network" tab!



Department of Computer Science

Welcome to the Brown University Computer Science Department Web. Information here is organized into broad categories, which are summarized in the icon bar, above. If you are visiting for th or exploring, the rest of this page offers some details about what you'll find.

If you are visiting us in person, you'll need directions to the CIT building. If not, perhaps you just need our address, phone, fax or other vital statistics.

Calendar of Events

Talks, conferences and soirees both at Brown and elsewhere are described.

Programs of Study

Undergraduate concentration requirements and the masters and phd programs are described, accompanied by the relevant forms, brochures and pointers to related information elsewhere.

Research Groups

Active research areas in computer science at Brown include graphics, geometric computing, object-oriented databases, artificial intelligence and robotics. Each group maintains a home pag describing their research and activities and links to relevant publications.

Publications

The Department publishes brochures, technical reports, a newsletter, *conduit!*, and, for locals, house rules.

Courses

Many courses taught using the Department's facilities have home pages, which provide information useful to students taking them.

amazon

Delivering to Providence 02912
Update location

All | Search Amazon

EN

Hello, sign in
Account & Lists

Returns
& Orders

0 Cart

All | Holiday Deals | Medical Care | Groceries | Best Sellers | Amazon Basics | Prime | Registry | New Releases | Today's Deals | Customer S | Fashion

Sign in

New customer? Start here.

Early Black Friday deals

Save up to 50% on Amazon smart home devices

Limited-time offer

Gear up for game day

Shop all teams

Try on Coach styles for free

Shop Coach with Prime Try Before You Buy

Top Deal

Up to 50% off Deal

Ring Doorbells, Cameras and Bundles

See all deals

Sign in for the best experience

Sign in securely

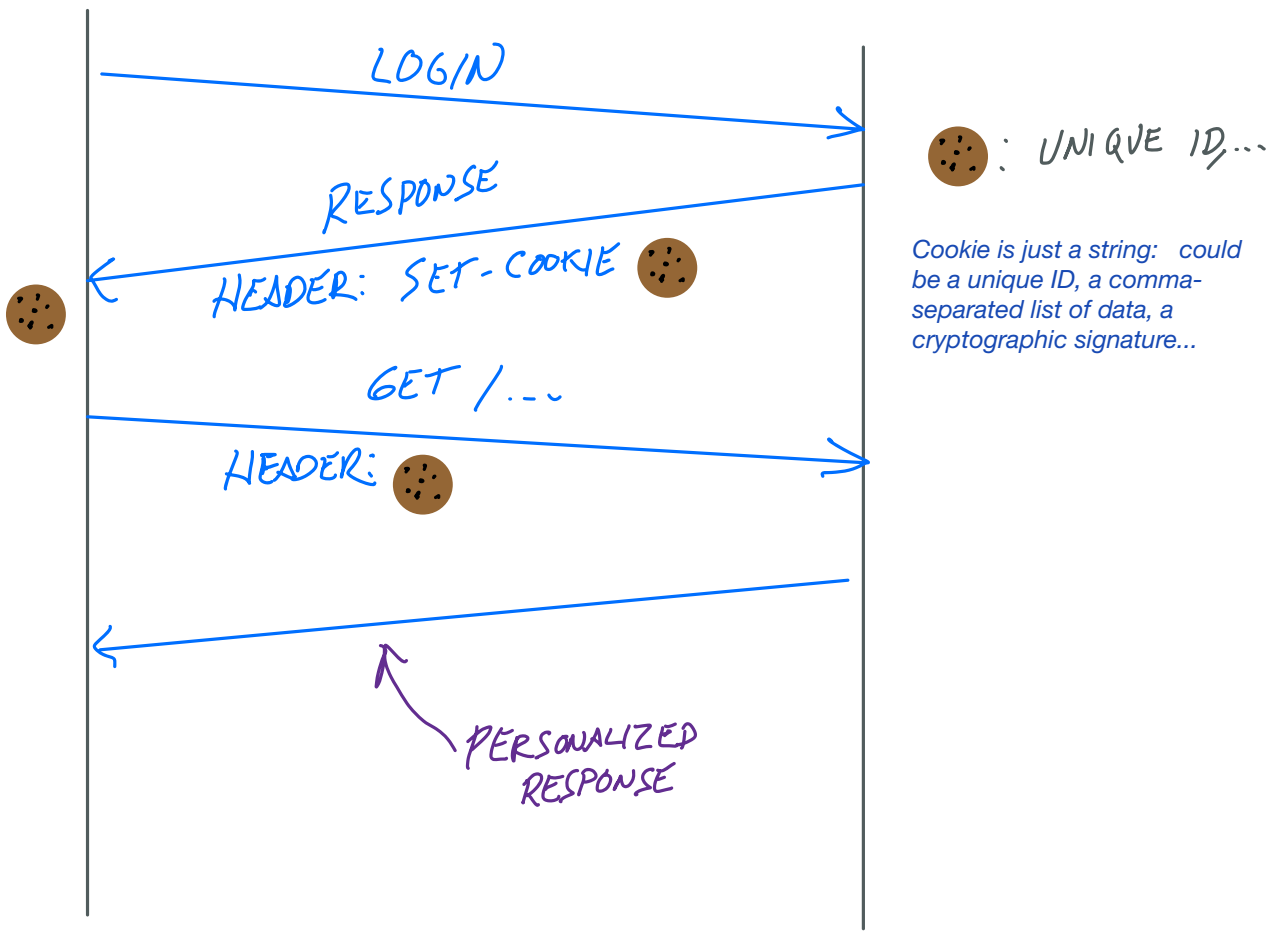
HTTP is stateless

- Strategy: want to keep server simple, don't want server to need to remember lots of state about a client *(compare to Snowcast: one TCP connection per connected client)*

But how to implement applications that need state (e.g., anything requiring login, shopping cart, etc.)???

One way: server asks client to set a cookie: some piece of information that tells the server how to look up its state
- When a client makes another request for that server (*), send cookies with request (as headers)

=> With cookies, don't need to have persistent state on server for each client
=> Server can use what's in the cookie or, use it to look up a value!



***: Extra note outside of scope for this course**: "same server" more precisely means "same Origin", which is any request for the same domain and port number, subject to some restrictions to prevent sharing cookie data with other entities. Lots of important security considerations here! For more on this, read about the "same-origin policy" (or take CS1660))*

Ways to serve data

Roughly, two types of content:
- Static content: e.g., files on disk
- Dynamic content: generated on the fly for that client (e.g., fetch from database, run some code, ...)

Extra notes if you want to read more...

How to find stuff: URLs

protocol://[name@]hostname[:port]/directory/resource?k1=v1&k2=v2#tag

- Name: can identify a client
- *Hostname:* FQDN or IP address
- *Port number: defaults to common protocol port (eg. 80, 22)*
- *Directory:* path to the resource
- *Resource:* name of the object
- After that, various delimiters to specify further, common examples:
  - ?*parameters* are passed to the server for execution
  - #*tag* allows jumps to named tags within document

Steps in HTTP(1.0) Request

- Open TCP connection to server
- Send request
- Receive response
- TCP connection terminates
  - How many RTTs for a single request?
- You may also need to do a DNS lookup first!

HTTP Responses

Status codes to indicate something about the result

- 1xx: Information e.g, 100 Continue
- 2xx: Success e.g., 200 OK
- 3xx: Redirection e.g., 302 Found (elsewhere),
- 4xx: Client Error e.g., 403 Forbidden, 404 Not Found
- 5xx: Server Error e.g, 503 Service Unavailable