CSCI-1680
HTTP

Nick DeMarinis

Based partly on lecture notes by Rodrigo Fonseca, Scott Shenker and John Jannotti

Administrivia

- You should have done your milestone II meeting
- You have one week from today to finish TCP.
  Do not wait until the end.
- Final project info: Thursday

HTTP: Hypertext Transfer Protocol

HTTP

"Application protocol for distributed, collaborative hypermedia information systems"

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

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

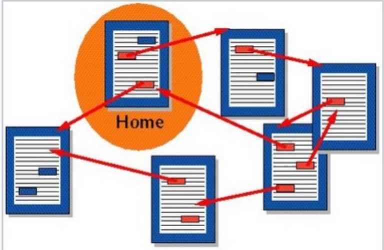
70 languages

Article Talk

Read Edit View history Tools

From Wikipedia, the free encyclopedia

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

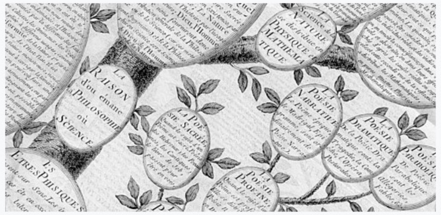


Documents that are connected by hyperlinks

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.

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



Information mapping

Topics and fields

Business decision mapping · Data visualization

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.

“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”

<u>"As we may think", Vannevar Bush (1945)</u>

"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

Contents [hide]

From Wikipedia, the free encyclopedia

(Redirected from Http)

The **Hypertext Transfer Protocol** (**HTTP**) is an application layer protocol in the Internet protocol suite model for distributed, collaborative, hypermedia information systems.[1] HTTP is the foundation of data communication for the World Wide Web, where hypertext documents include hyperlinks to other resources that the user can easily access, for example by a mouse click or by tapping the screen in a web browser.

Development of HTTP was initiated by Tim Berners-Lee at CERN in 1989 and summarized in a simple document describing the behavior of a client and a server using the first HTTP version, named 0.9.[2] That version was subsequently developed, eventually becoming the public 1.0.[3]

Development of early HTTP Requests for Comments (RFCs) started a few years later in a coordinated effort by the Internet Engineering Task Force (IETF) and the World Wide Web Consortium (W3C), with work later moving to the IETF.

HTTP/1 was finalized and fully documented (as version 1.0) in 1996.[4] It evolved (as version 1.1) in 1997 and then its specifications were updated in 1999, 2014, and 2022.[5]

Its secure variant named HTTPS is used by more than 85% of websites.[6] HTTP/2, published in 2015, provides a more efficient expression of HTTP's semantics "on the wire". As of April 2023, it is used by 39% of websites[7] and supported by almost all web browsers (over 97% of users).[8] It is also supported by

HTTP

International standard	RFC 1945 HTTP/1.0 RFC 9110 HTTP Semantics RFC 9111 HTTP Caching RFC 9112 HTTP/1.1 RFC 9113 HTTP/2 RFC 7541 HTTP/2: HPACK Header Compression RFC 8164 HTTP/2: Opportunistic Security for

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

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

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

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

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

… now synonymous with with "The Internet" itself!

Tim Berners-Lee

- 1990: First HTTP implementation
  - Tim Berners-Lee, CERN

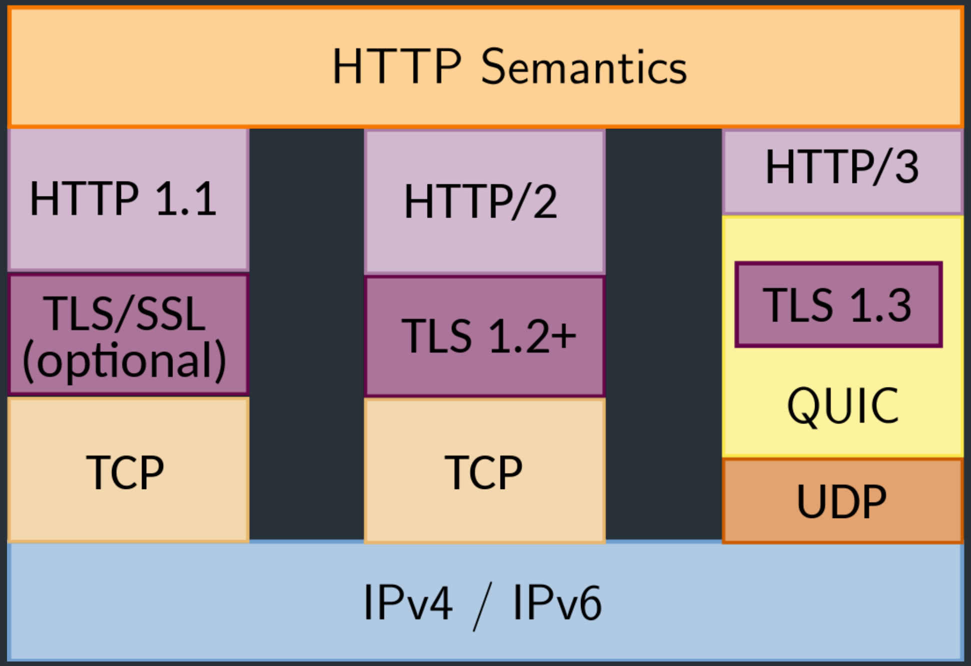
- 1991: HTTP/0.9: Fetching pages

- 1992: HTTP/1.0:
  Client/server information, simple caching

- 1996: HTTP/1.1
  - Extensive caching support
  - Host identification
  - Pipelined, persistent connections, …

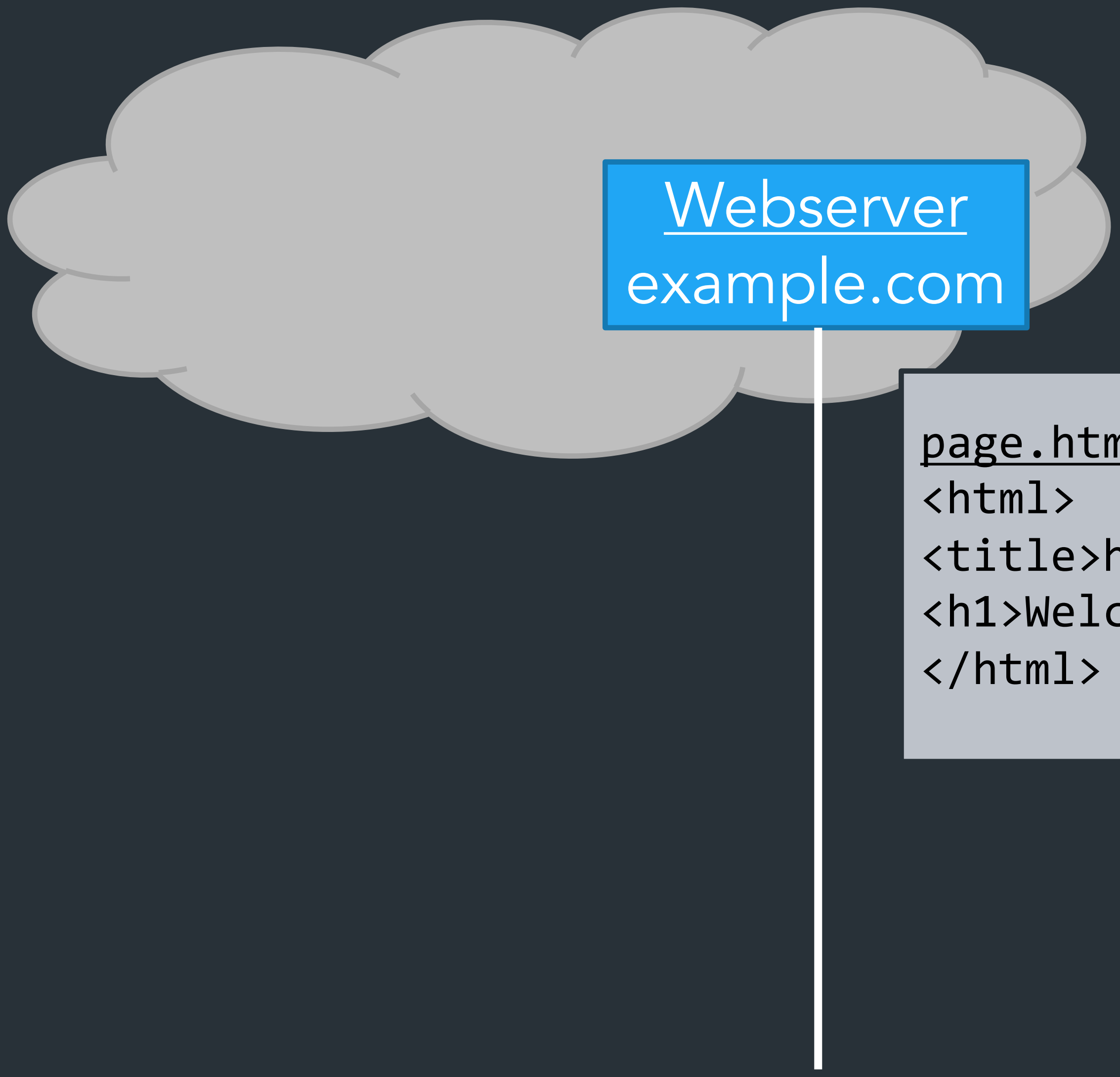
The first webserver

- <u>2015</u>: HTTP/2
  - Main goal: reduce latency
- <u>2022</u>: HTTP/3
  - Still: reduce latency
  - Integrates security via TLS
  - Replace transport layer with QUIC
  - Already supported in >94% of browsers



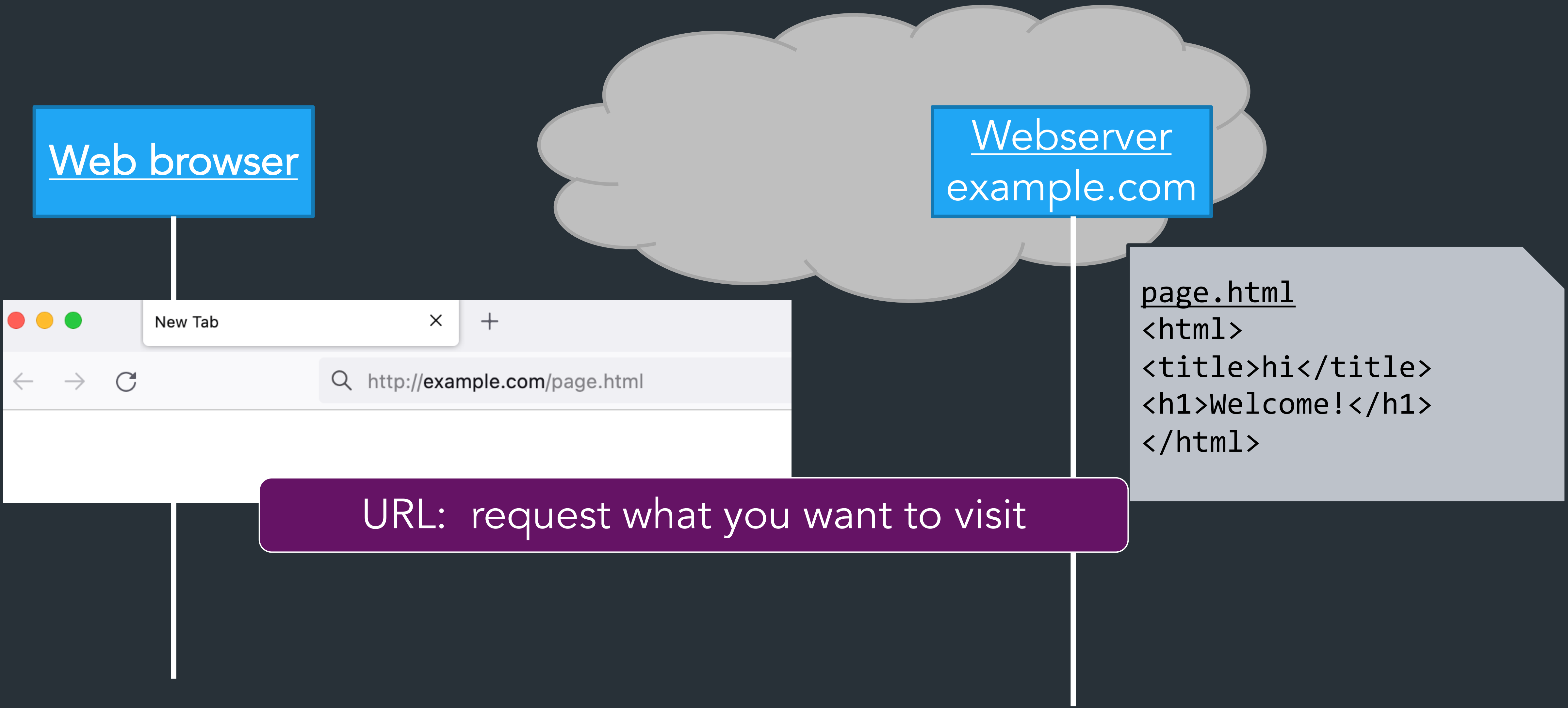
http://httpwg.org/specs/rfc7540.html

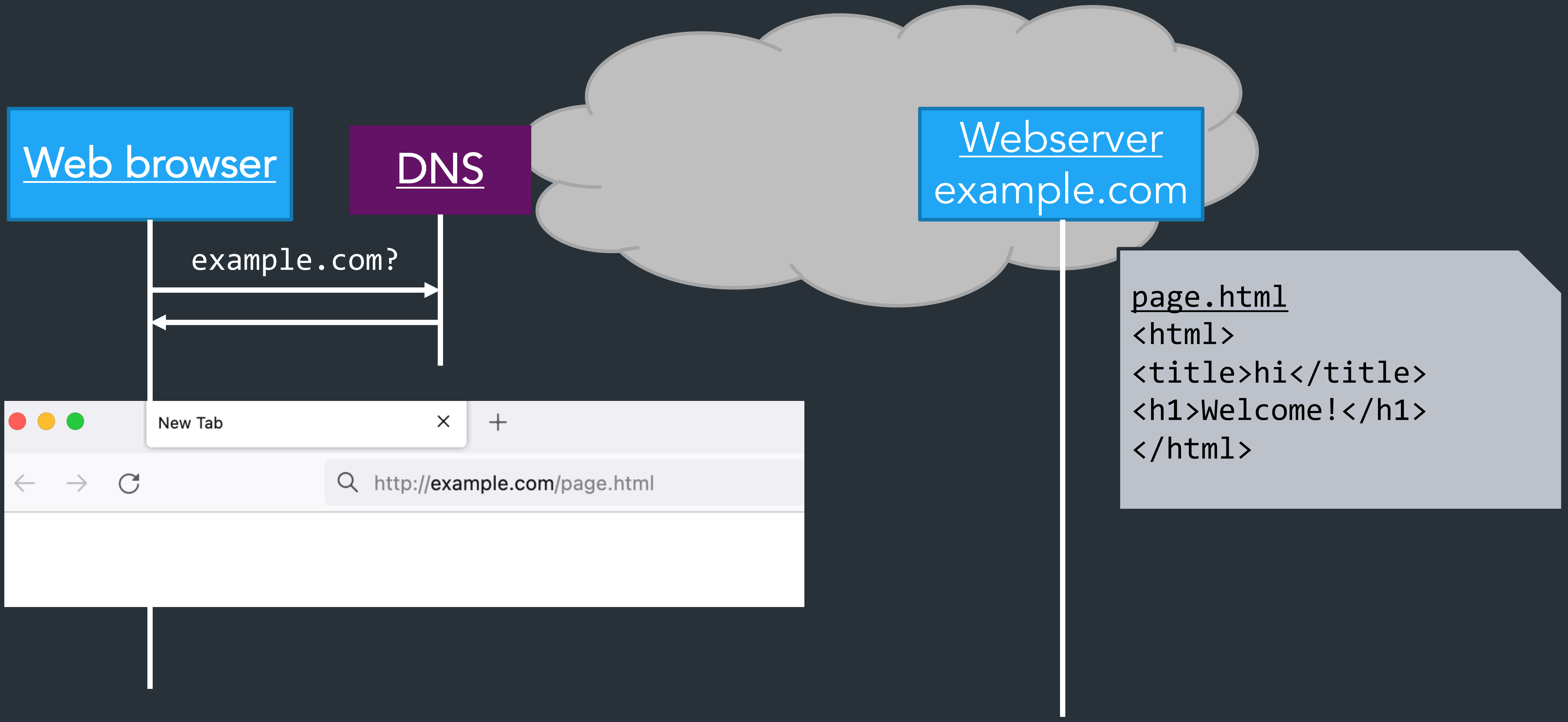
How does “the web” work?



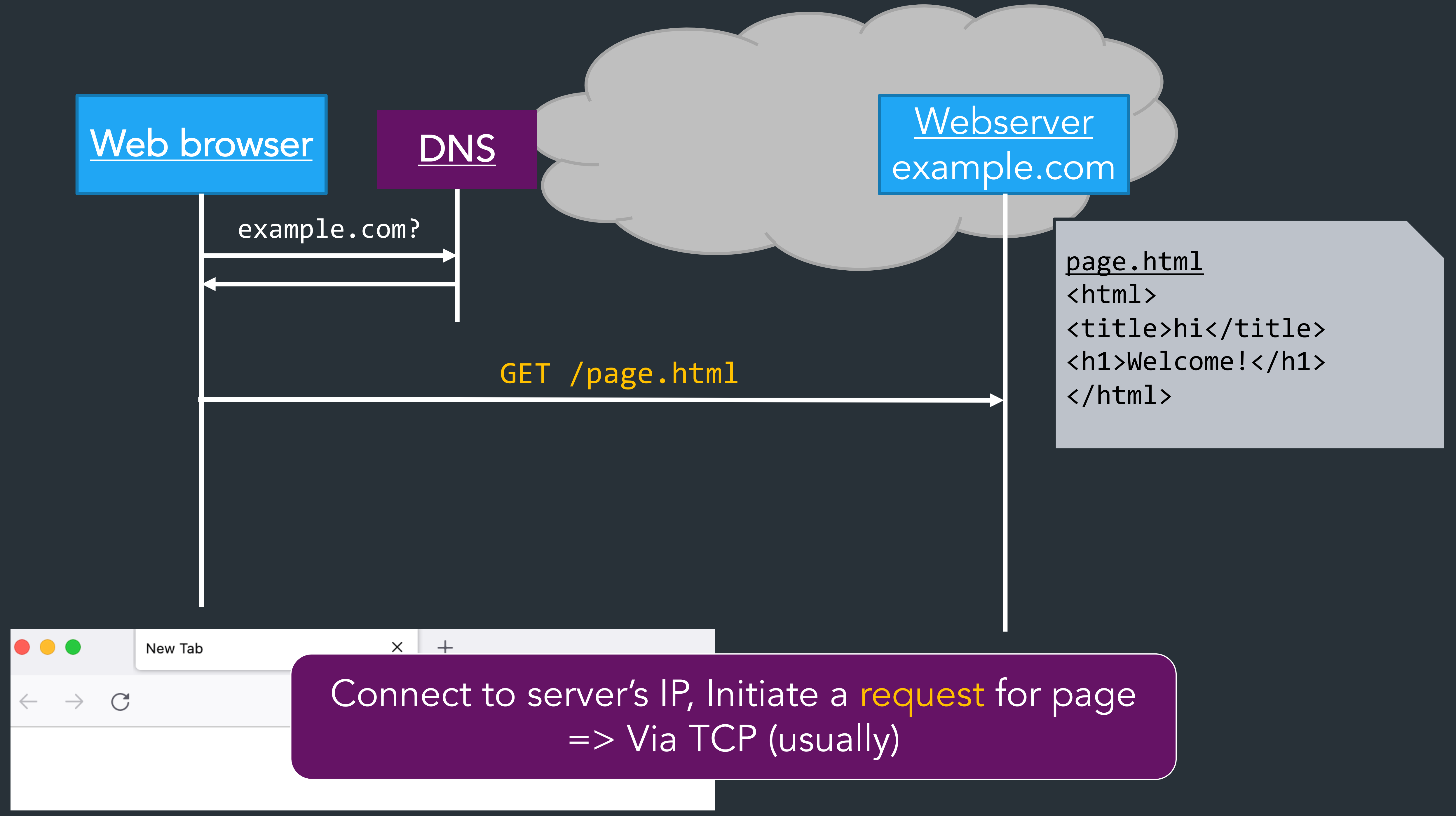
page.html

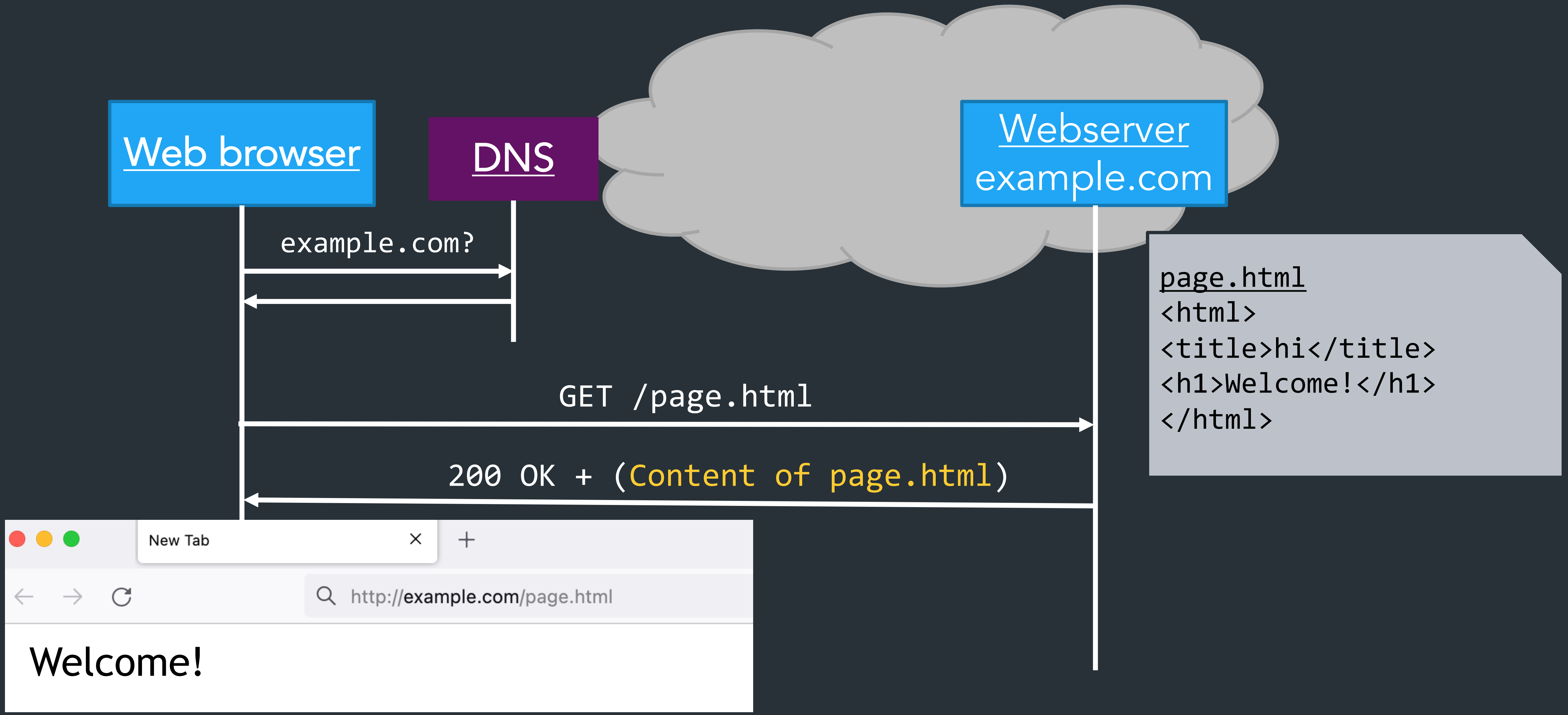
```
<html>
<title>hi</title>
<h1>Welcome!</h1>
</html>
```


Web browser
Webserver
example.com
New Tab
http://example.com/page.html
page.html
<html>
<title>hi</title>
<h1>Welcome!</h1>
</html>
URL: request what you want to visit



First step: look up where server is on Internet
=> Usually, via DNS


Web browser
DNS
Webserver
example.com
example.com?
GET /page.html
page.html
<html>
<title>hi</title>
<h1>Welcome!</h1>
</html>
New Tab
Connect to server's IP, Initiate a request for page
=> Via TCP (usually)



Server returns response (in this case, with HTML)

Why so successful?

Anyone can host a website!

… just need a domain and a server

Why so successful?

Anyone can host a website!

… just need a domain and a server

Clients can easily find arbitrary pages, pages can easily link to others => content can grow very quickly

HTTP components

Content: objects (HTML, images, JSON, …)

Clients: send requests, receive response

Servers: store content, or generate it

HTTP components

Content: objects (HTML, images, JSON, …)

Clients: send requests, receive response

Servers: store content, or generate it

Proxies/Middleboxes

- Placed between clients and servers
- Do extra stuff: caching, anonymization, logging, transcoding, filtering access

=> Important for scaling, modern browsing…
more on this later

How to find stuff?

- So far: DNS: names for one or more hosts
  - eg. cs.brown.edu

How do we ask for a specific *resource* from this host?

URL: Uniform Resource Locator

How to find stuff?

- So far: DNS: names for one or more hosts
  - eg. cs.brown.edu

How do we ask for a specific *resource* from this host?

URL: Uniform Resource Locator

URLs: how we find stuff

https://cs.brown.edu/courses/csci1680/f23/policies/#late-policy

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

HTTP: the protocol

- Client-server protocol
- Protocol (but not data) in ASCII (before HTTP/2)
- Stateless
- Server typically listens on port 80 (or 443, with TLS)

- Server sends response, may close connection (client may ask it to say open)

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!

```
> telnet www.cs.brown.edu 80
Trying 128.148.32.110...
Connected to www.cs.brown.edu.
Escape character is '^]'.
GET / HTTP/1.0

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">
```

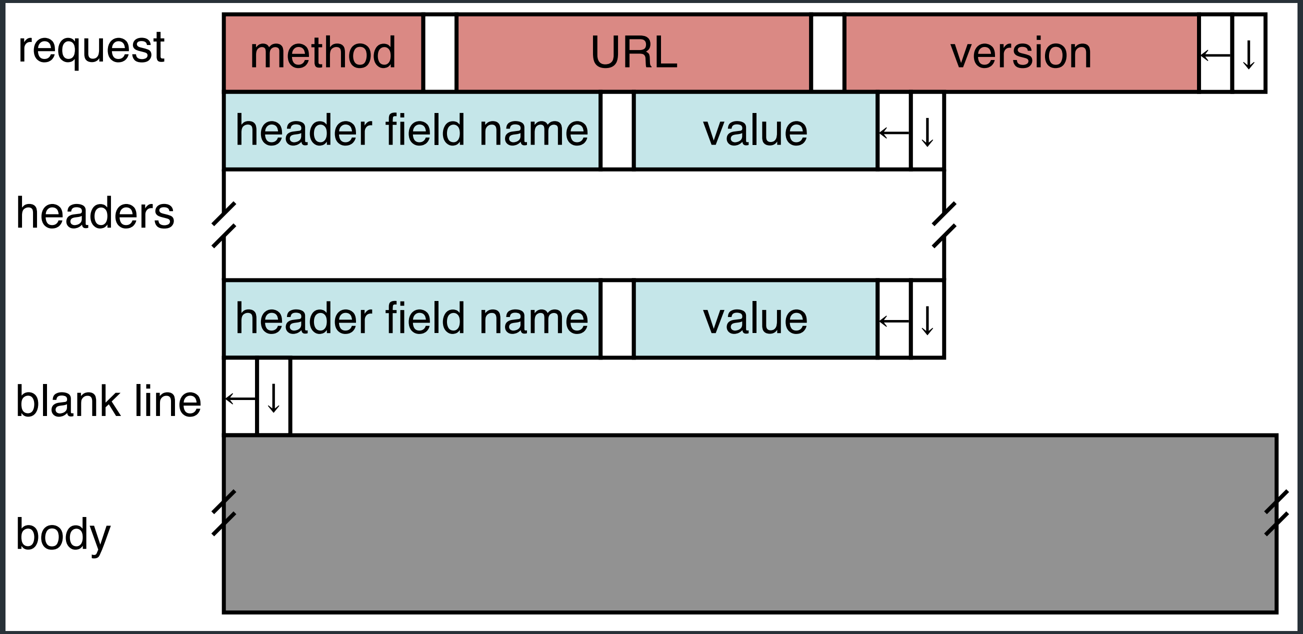
HTTP Request

Method:

- GET: current value of resource, run program
- POST: update a resource, provide input for a program. . .

Headers: useful info about request

- E.g., desired language, text encoding



Sample Browser Request

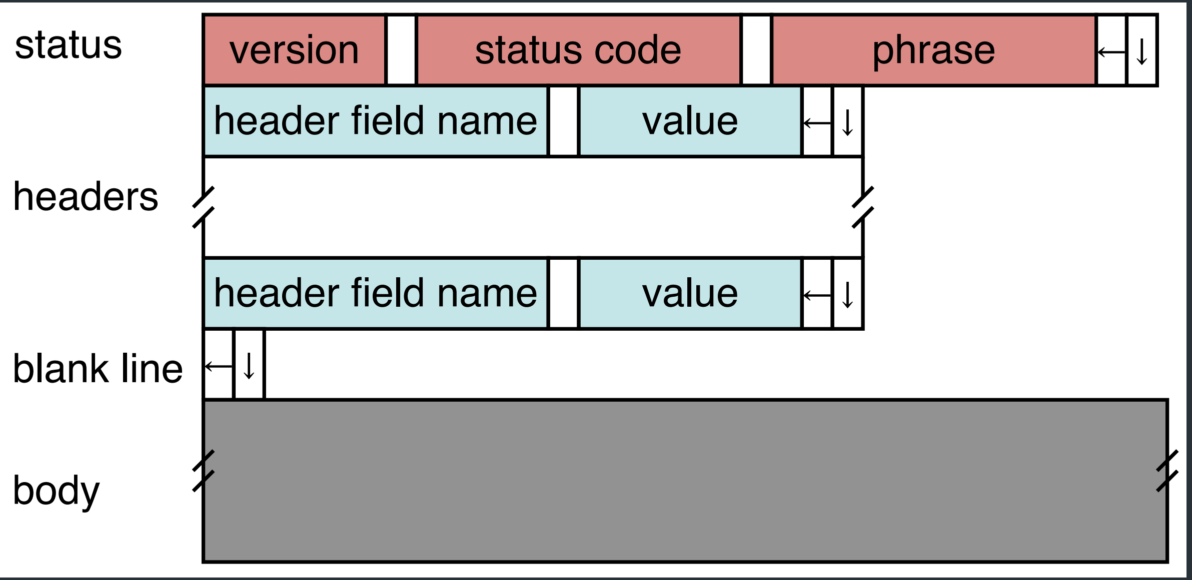
```
GET / HTTP/1.1
Host: localhost:8000
User-Agent: Mozilla/5.0 (Macinto ...
Accept: text/xml,application/xm ...
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
(empty line)
```

In your browser: right click => Inspect element => Network

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



HTTP is Stateless

- Each request/response treated independently
- Servers not required to maintain state

HTTP is Stateless

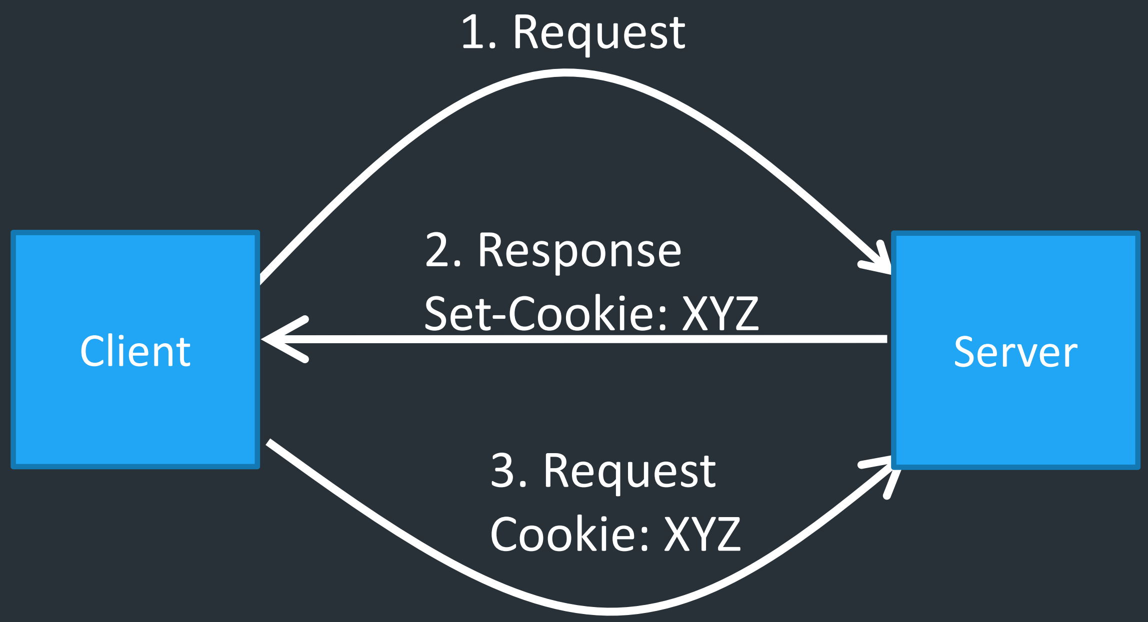
- Each request/response treated independently
- Servers not required to maintain state

But…

- Most applications need persistent state
- E.g., shopping cart, web-mail, usage tracking, (most sites today!)

HTTP Cookies

- Client-side state maintenance
  - Client stores small state on behalf of server
  - Sends request in future requests to the server
  - Cookie value is meaningful to the server (e.g., session id)
- Can provide authentication



Anatomy of a Web Page

- HTML content
- A number of additional resources
  - Images
  - Scripts
  - Frames
- Browser makes one HTTP request for each object
  - Course web page: 14 objects
  - Modern web pages: hundreds of objects

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.

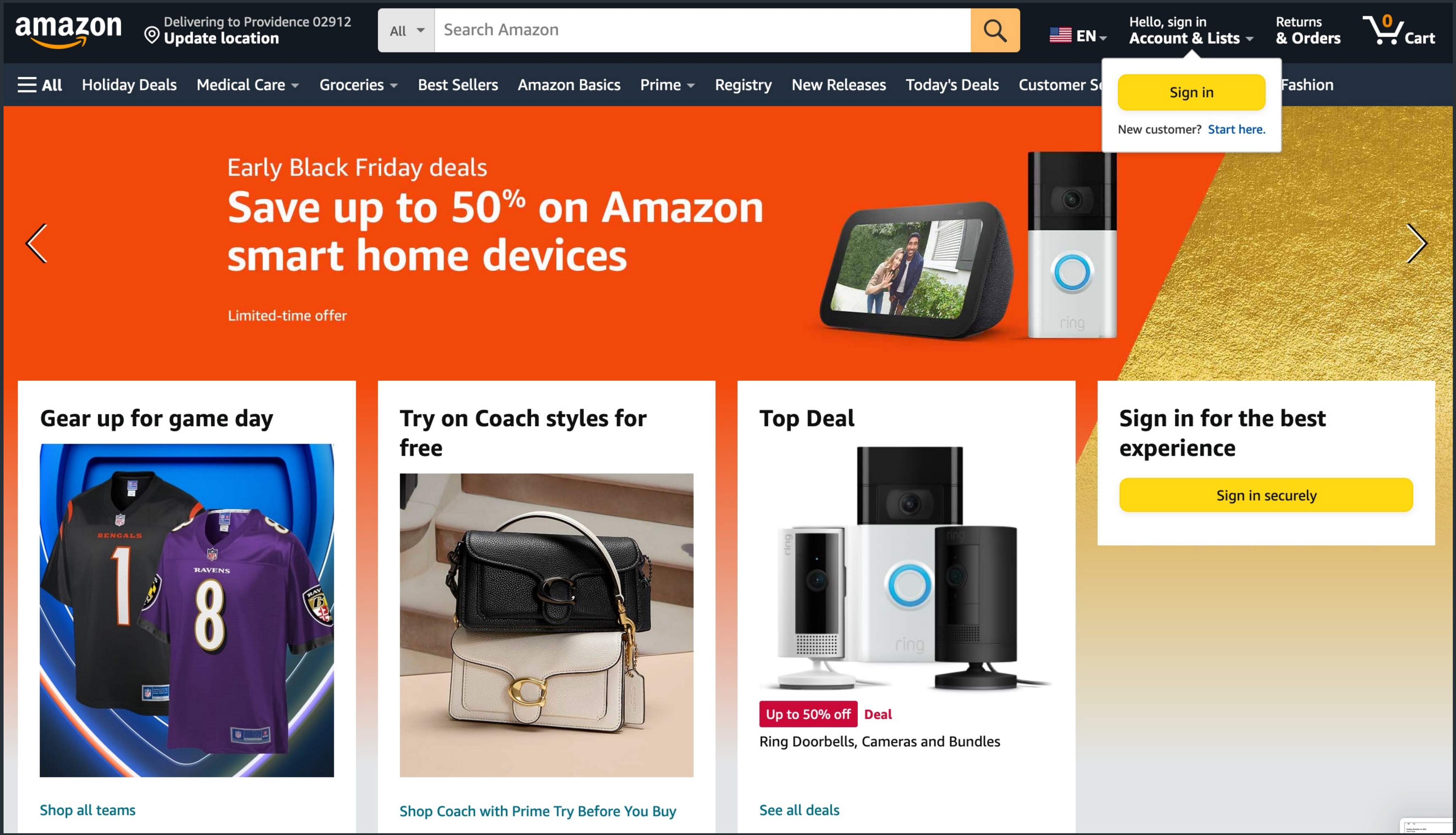
Calendar of Events
Talks, conferences and soirees both at Brown and elsewhere are described.

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 Se
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

Modern web pages and HTTP

- Web APIs: HTTP response/requests are a standard way to ask for *anything*
- *Modern web pages: use Javascript to make lots of requests without reloading page*
  - *And can use APIs for all kinds of other stuff*

Example: Github public API

```
$ curl https://api.github.com/users/ndemarinis
{
  "login": "ndemarinis",
  "id": 1191319,
  "node_id": "MDQ6VXNlcjExOTEzMTk=",
  "avatar_url": "https://avatars.githubusercontent.com/u/1191319?v=4",
  "gravatar_id": "",
  "url": "https://api.github.com/users/ndemarinis",
  "type": "User",
  "site_admin": false,
  "name": "Nick DeMarinis",
  "blog": "https://vty.sh",
  "twitter_username": null,
  "public_repos": 10,
 . . .
}
```

HTTP

```
> telnet www.cs.brown.edu 80
Trying 128.148.32.110...
Connected to www.cs.brown.edu.
Escape character is '^]'.
GET / HTTP/1.0

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">
```

HTTP: What matters for performance?

Depends on type of request

- Lots of small requests (objects in a page)
- Some big requests (large download or video)

Small Requests

- Latency matters
- RTT dominates
- Major steps:
  - DNS lookup (if not cached)
  - Opening a TCP connection
  - Setting up TLS (optional, but now common)
  - Actually sending the request and receiving response

How can we reduce the number of connection setups?

- Keep the connection open and request all objects serially
  - Works for all objects coming from the same server
  - Which also means you don't have to "open" the window each time

Persistent connections (HTTP/1.1)

Browser Request

```
GET / HTTP/1.1
Host: localhost:8000
User-Agent: Mozilla/5.0 (Macinto ...
Accept: text/xml,application/xm ...
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
```

Small Requests (cont)

- Second problem is that requests are serialized
  - Similar to stop-and-wait protocols!
- Two solutions
  - Pipelined requests (similar to sliding windows)
  - Parallel Connections
    - Browsers implement this differently—see “Inspect element”
  - How are these two approaches different?

HTTP/2

- Adds more options to trade off:
- Multiplexed streams on same connection
  - Plus stream weights, dependencies
- No head of line blocking!
  - But what happens if there is packet loss?

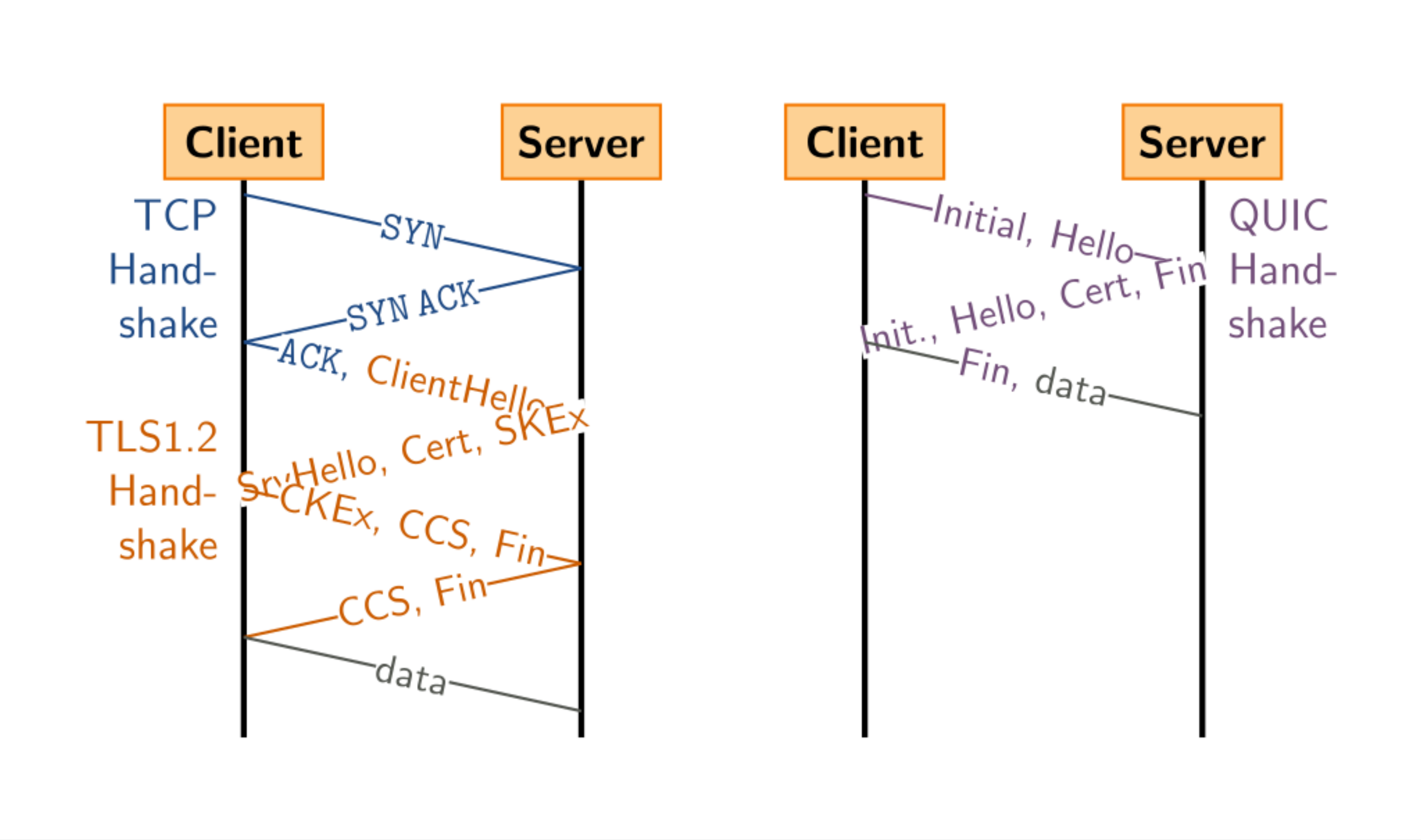
https://www.twilio.com/blog/2017/10/http2-issues.html

HTTP/3

- Mapping of HTTP semantics onto QUIC
  - E.g., QUIC already implements multiple streams, and HTTP doesn’t need to do it
- QUIC: Another transport-layer protocol, intended to replace TCP
  - RFC9000
  - Same goals as TCP, but…
  - Integrates security by default (TLS, next class)
  - Supports multiple streams at once
  - Various tricks to reduce message size and latency

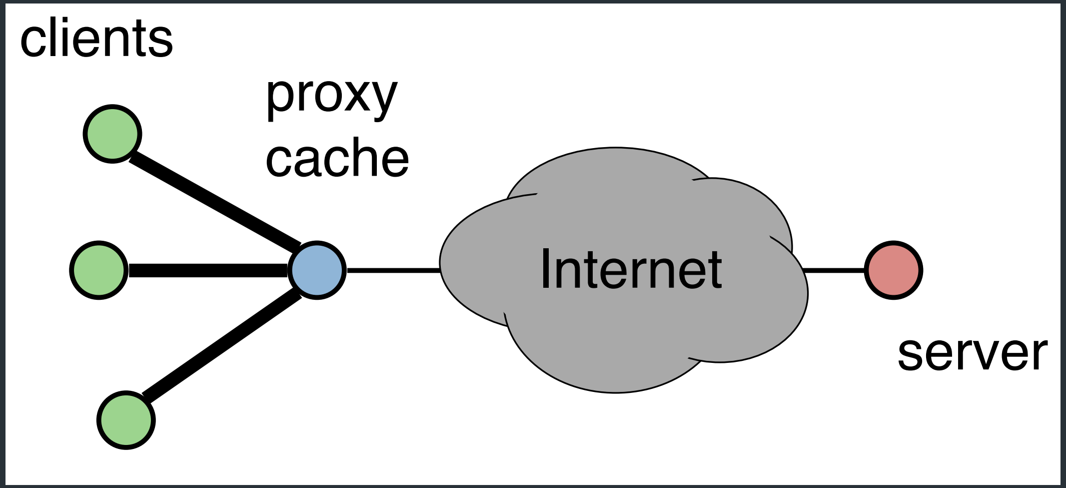
- By moving multiplexing into the transport layer, can do so in a way that benefits HTTP (no head of line blocking!)

Comparison: QUIC's handshake



Larger Objects

- Problem is throughput in bottleneck link
- Solution: HTTP Proxy Caching
  - Also improves latency, and reduces server load

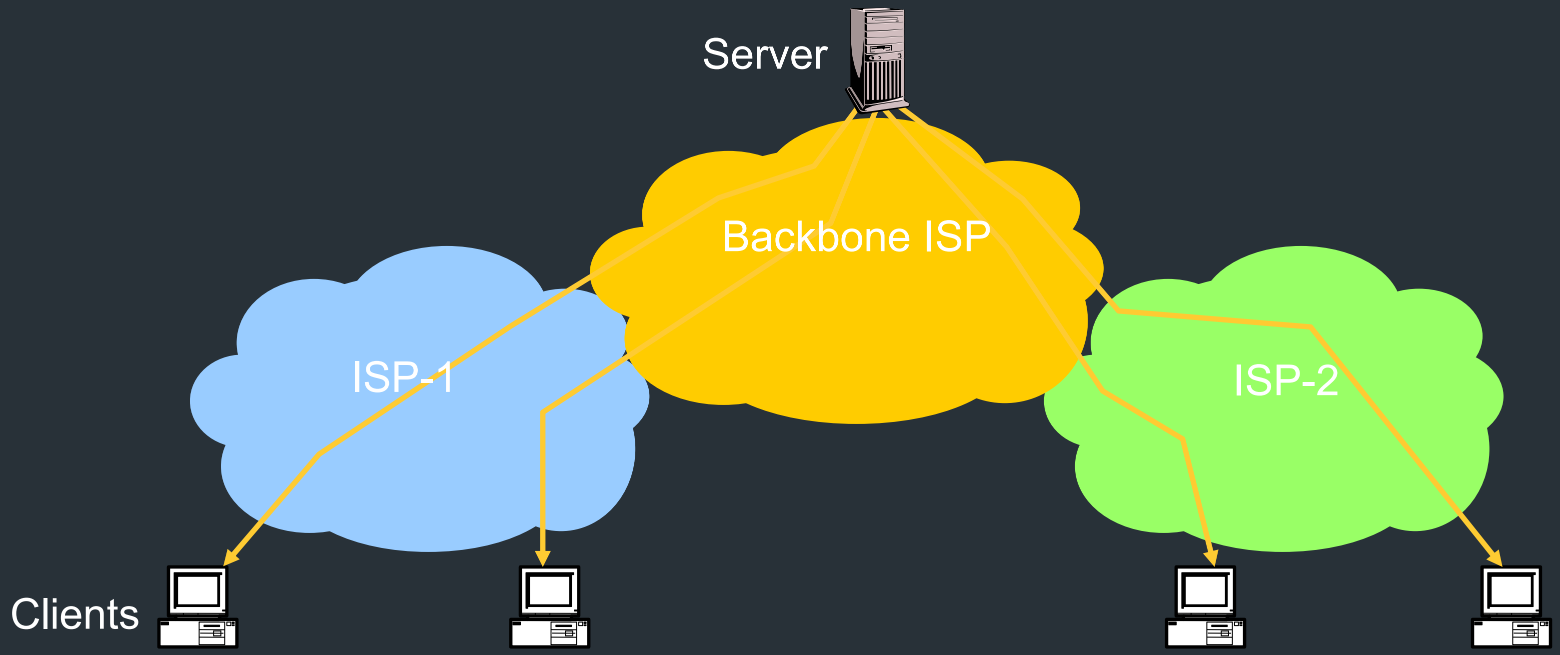


How to Control Caching?

- Server sets options
  - **Expires** header
  - No-Cache header
- Client can do a conditional request:
  - Header option: if-modified-since
  - Server can reply with 304 NOT MODIFIED

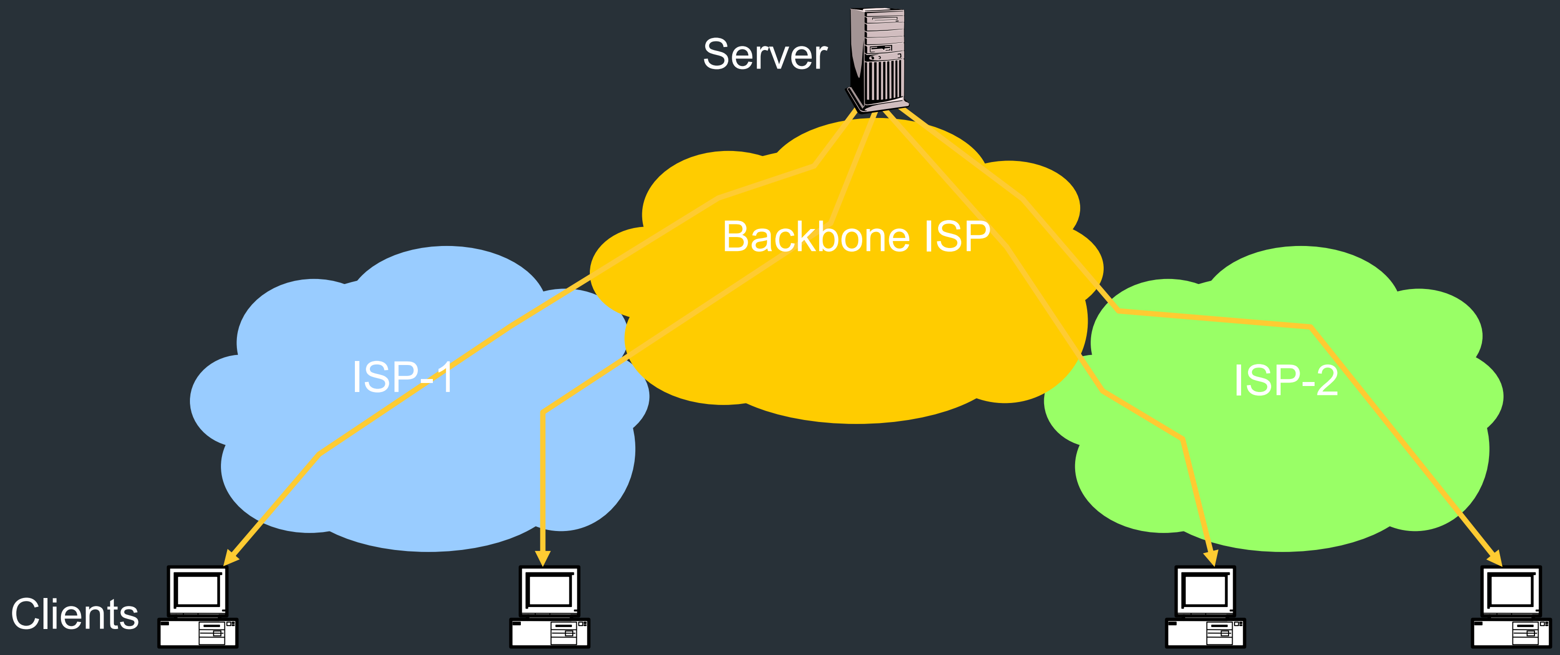
Caching

- Where to cache content?
  - Client (browser): avoid extra network transfers
  - Server: reduce load on the server
  - Service Provider: reduce external traffic



Caching

- Why caching works?
  - Locality of reference:
    - Users tend to request the same object in succession
    - Some objects are popular: requested by many users



How well does caching work?

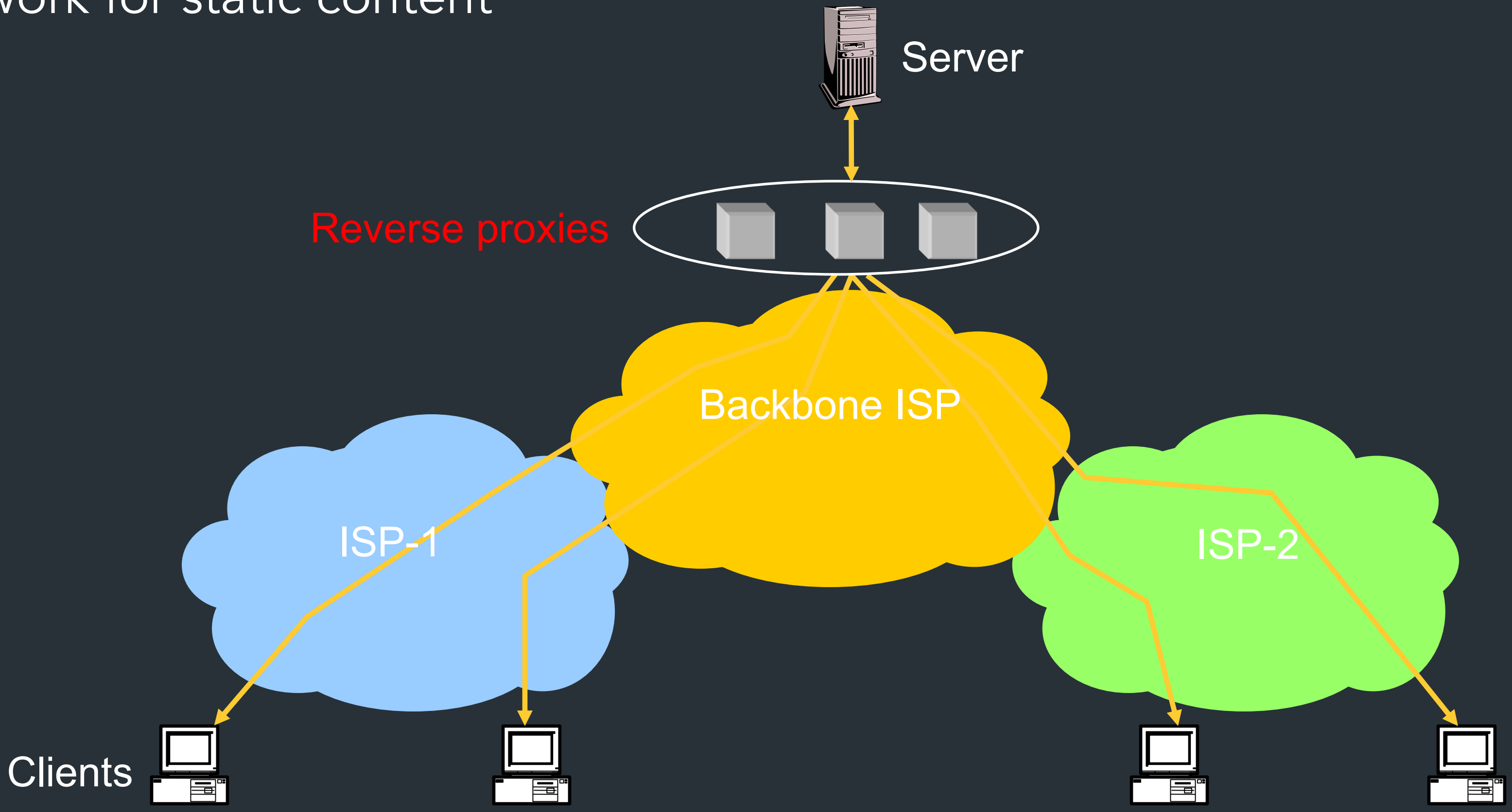
- Very well, up to a point
  - Large overlap in requested objects
  - Objects with one access place upper bound on hit ratio
  - Dynamic objects not cacheable*
- Example: Wikipedia
  - About 400 servers, 100 are HTTP Caches (Squid)
  - 85% Hit ratio for text, 98% for media

* But can cache portions and run special code on edges to reconstruct

Reverse Proxies

Close to the server

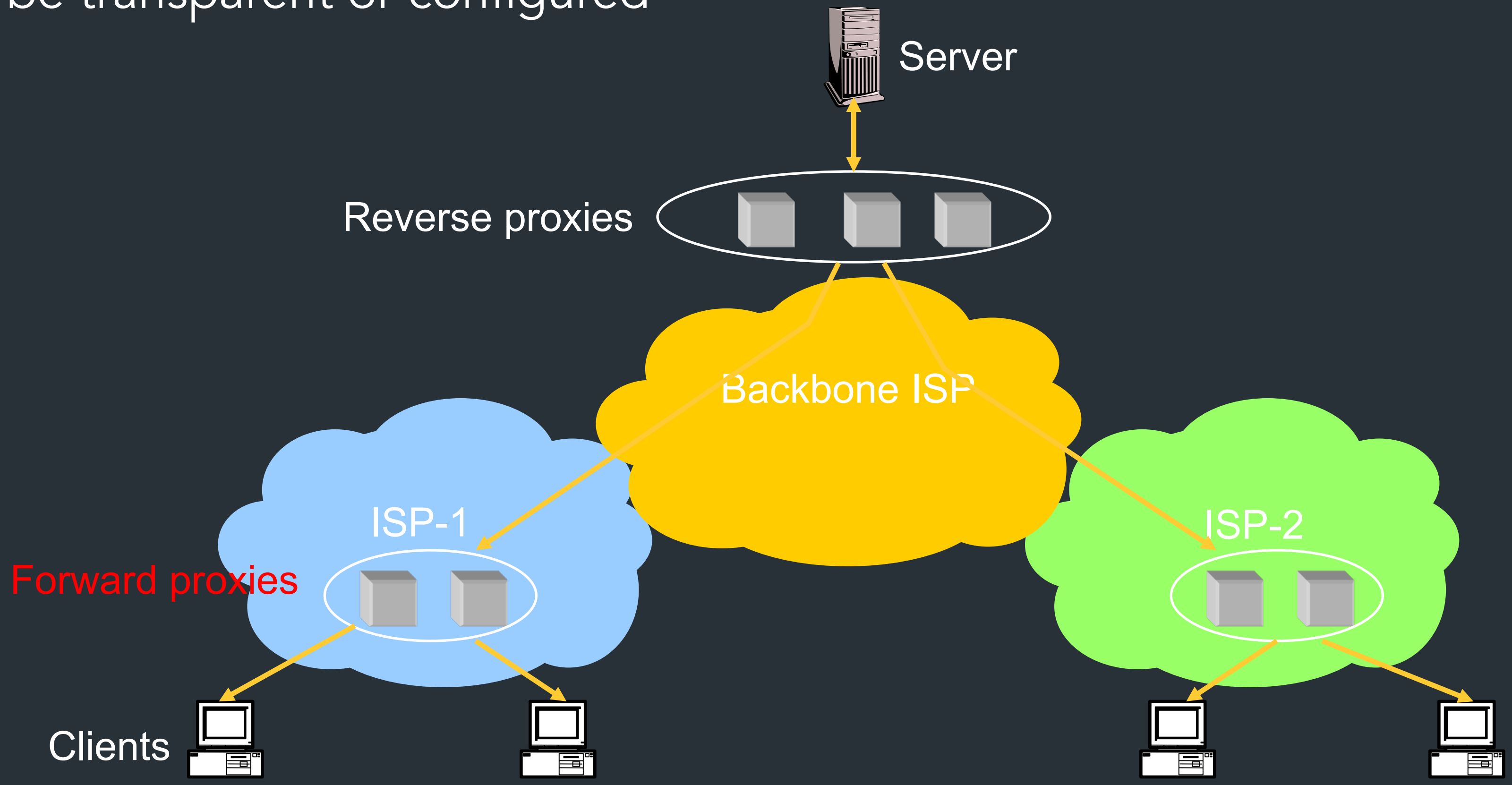
- Also called Accelerators
- Only work for static content



Forward Proxies

Typically done by ISPs or Enterprises

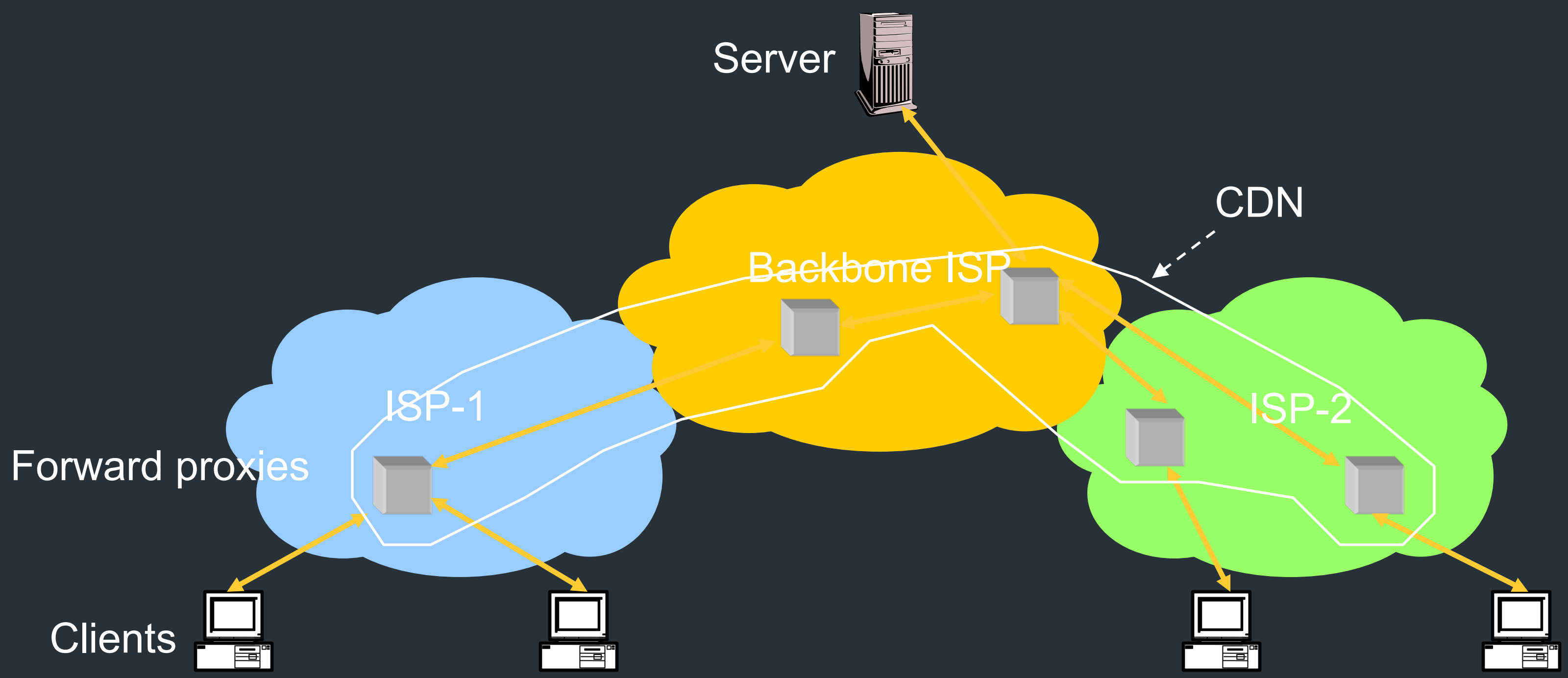
- Reduce network traffic and decrease latency
- May be transparent or configured



Content Distribution Networks

- Integrate forward and reverse caching
  - One network generally administered by one entity
  - E.g. Akamai
- Provide document caching
  - Pull: result from client requests
  - Push: expectation of high access rates to some objects
- Can also do some processing
  - Deploy code to handle some dynamic requests
  - Can do other things, such as transcoding

Example CDN



How Akamai works

Akamai has cache servers deployed close to clients

- Co-located with many ISPs

- Challenge: make same domain name resolve to a proxy close to the client
- Lots of DNS tricks. BestBuy is a customer
  - Delegate name resolution to Akamai (via a CNAME)

DNS Resolution

```
dig www.bestbuy.com
;; ANSWER SECTION:
www.bestbuy.com.  3600     IN       CNAME    www.bestbuy.com.edgesuite.net.
www.bestbuy.com.edgesuite.net. 21600 IN     CNAME    a1105.b.akamai.net.
a1105.b.akamai.net.       20       IN       A        198.7.236.235
a1105.b.akamai.net.       20       IN       A        198.7.236.240
;; AUTHORITY SECTION:
b.akamai.net.             1101     IN       NS       n1b.akamai.net.
b.akamai.net.             1101     IN       NS       n0b.akamai.net.
;; ADDITIONAL SECTION:
n0b.akamai.net.           1267     IN       A        24.143.194.45
n1b.akamai.net.           2196     IN       A        198.7.236.236
```

- n1b.akamai.net finds an edge server close to the client's local resolver
    - Uses knowledge of network: BGP feeds, traceroutes. *Their secret sauce…*

Example

From Brown

```
dig www.bestbuy.com
;; ANSWER SECTION:
www.bestbuy.com.  3600  IN    CNAME www.bestbuy.com.edgesuite.net.
www.bestbuy.com.edgesuite.net. 21600 IN  CNAME a1105.b.akamai.net.
a1105.b.akamai.net.     20    IN    A     198.7.236.235
a1105.b.akamai.net.     20    IN    A     198.7.236.240
```

– Ping time: 2.53ms

From Berkeley, CA

```
a1105.b.akamai.net.     20    IN    A     198.189.255.200
a1105.b.akamai.net.     20    IN    A     198.189.255.207
```

– Ping time: 3.20ms

```
dig www.bestbuy.com
;; QUESTION SECTION:
;www.bestbuy.com. IN A

;; ANSWER SECTION:
www.bestbuy.com. 2530 IN CNAME www.bestbuy.com.edgekey.net.
www.bestbuy.com.edgekey.net. 85 IN CNAME e1382.x.akamaiedge.net.
e1382.x.akamaiedge.net. 16 IN A 104.88.86.223

;; Query time: 6 msec
;; SERVER: 192.168.1.1#53(192.168.1.1)
;; WHEN: Thu Nov 16 09:43:11 2017
;; MSG SIZE  rcvd: 123

traceroute to 104.88.86.223 (104.88.86.223), 64 hops max, 52 byte packets
 1  router (192.168.1.1)  2.461 ms  1.647 ms  1.178 ms
 2  138.16.160.253 (138.16.160.253)  1.854 ms  1.509 ms  1.462 ms
 3  10.1.18.5 (10.1.18.5)  1.886 ms  1.705 ms  1.707 ms
 4  10.1.80.5 (10.1.80.5)  4.276 ms  6.444 ms  2.307 ms
 5  lsb-inet-r-230.net.brown.edu (128.148.230.6)  1.804 ms  1.870 ms  1.727 ms
 6  131.109.200.1 (131.109.200.1)  2.841 ms  2.587 ms  2.530 ms
 7  host-198-7-224-105.oshean.org (198.7.224.105)  4.421 ms  4.523 ms  4.496 ms
 8  5-1-4.bear1.boston1.level3.net (4.53.54.21)  4.099 ms  3.974 ms  4.290 ms
 9  * ae-4.r00.bstnma07.us.bb.gin.ntt.net (129.250.66.93)  4.689 ms  4.109 ms
10  ae-6.r24.nycmny01.us.bb.gin.ntt.net (129.250.4.114)  8.863 ms  10.205 ms  10.477 ms
11  ae-1.r08.nycmny01.us.bb.gin.ntt.net (129.250.5.62)  9.298 ms
    ae-1.r07.nycmny01.us.bb.gin.ntt.net (129.250.3.181)  10.008 ms  8.677 ms
12  ae-0.a00.nycmny01.us.bb.gin.ntt.net (129.250.3.94)  8.543 ms  7.935 ms
    ae-1.a00.nycmny01.us.bb.gin.ntt.net (129.250.6.55)  9.836 ms
13  a104-88-86-223.deploy.static.akamaitechnologies.com (104.88.86.223)  9.470 ms  8.483
ms  8.738 ms
```

```
dig www.bestbuy.com @109.69.8.51

e1382.x.akamaiedge.net. 12 IN A 23.60.221.144


traceroute to 23.60.221.144 (23.60.221.144), 64 hops max, 52 byte packets
 1  router (192.168.1.1)  44.072 ms  1.572 ms  1.154 ms
 2  138.16.160.253 (138.16.160.253)  2.460 ms  1.736 ms  2.722 ms
 3  10.1.18.5 (10.1.18.5)  1.841 ms  1.649 ms  3.348 ms
 4  10.1.80.5 (10.1.80.5)  2.304 ms  15.208 ms  2.895 ms
 5  lsb-inet-r-230.net.brown.edu (128.148.230.6)  1.784 ms  4.744 ms  1.566 ms
 6  131.109.200.1 (131.109.200.1)  3.581 ms  5.866 ms  3.238 ms
 7  host-198-7-224-105.oshean.org (198.7.224.105)  4.288 ms  6.218 ms  8.332 ms
 8  5-1-4.bear1.boston1.level3.net (4.53.54.21)  4.209 ms  6.103 ms  5.031 ms
 9  ae-4.r00.bstnma07.us.bb.gin.ntt.net (129.250.66.93)  3.982 ms  5.824 ms  4.514 ms
10  ae-6.r24.nycmny01.us.bb.gin.ntt.net (129.250.4.114)  9.735 ms  12.442 ms  8.689 ms
11  ae-9.r24.londen12.uk.bb.gin.ntt.net (129.250.2.19)  81.098 ms  81.343 ms  81.120 ms
12  ae-6.r01.mdrdsp03.es.bb.gin.ntt.net (129.250.4.138)  102.009 ms  110.595 ms  103.010
ms
13  81.19.109.166 (81.19.109.166)  99.426 ms  93.236 ms  101.168 ms
14  a23-60-221-144.deploy.static.akamaitechnologies.com (23.60.221.144)  94.884 ms  92.77
ms  93.281 ms
```

Other CDNs

- Akamai, Limelight, Cloudflare
- Amazon, Facebook, Google, Microsoft
- Netflix
- Where to place content?
- Which content to place? Pre-fetch or cache?