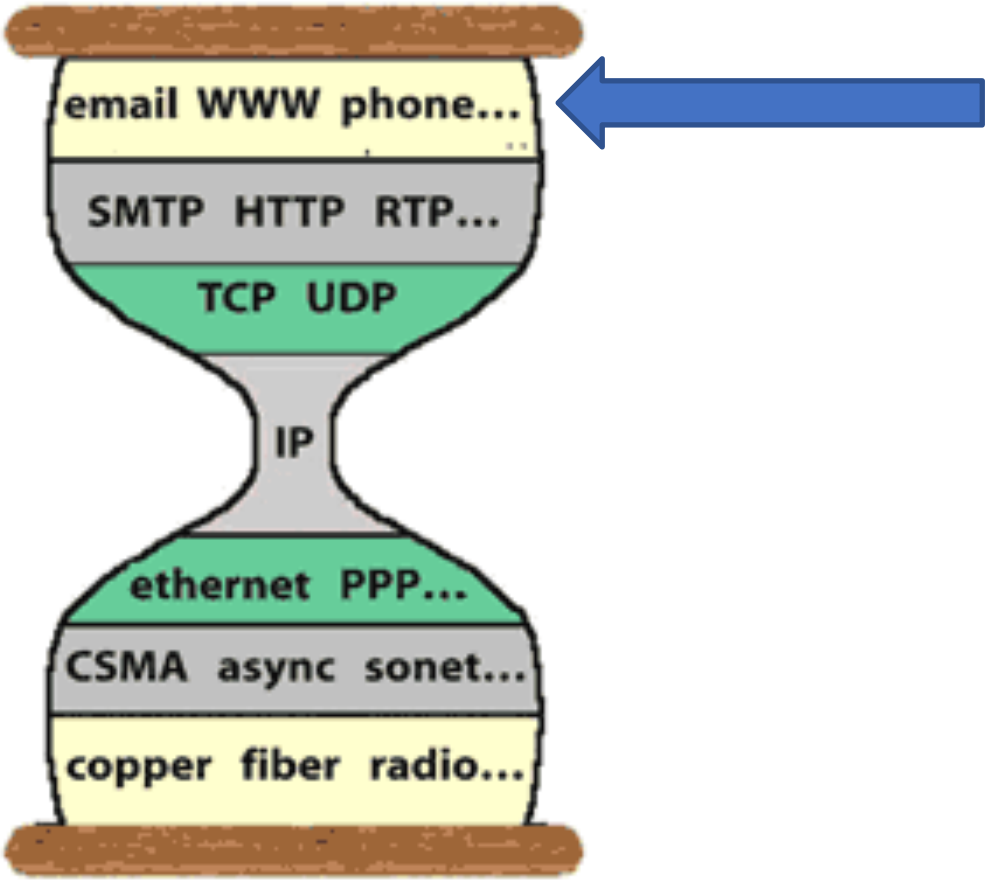
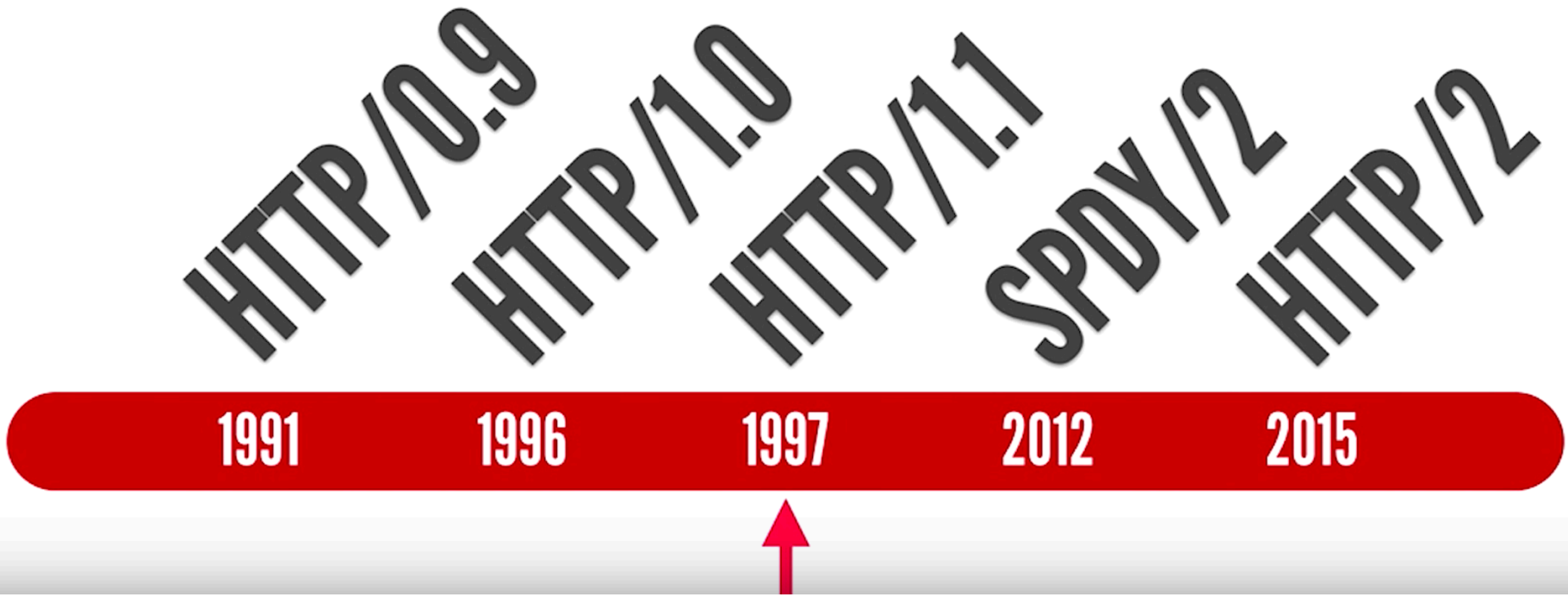
# Plan for today: More on Application Layer

- Non-persistent vs persistent HTTP, pipelining
- HTTP cookies
- HTTP caching
- HTTP/2: a brief introduction



# History HTTP




https://www.youtube.com/watch?v=r5oT_2ndjms

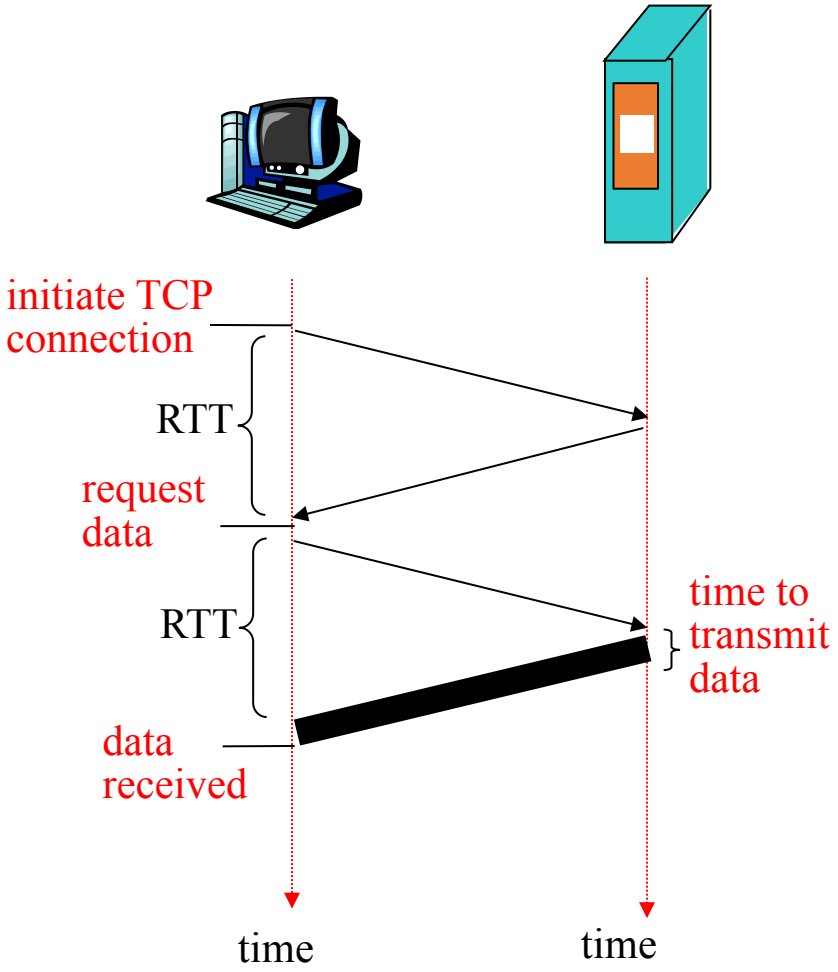
# Non-Persistent HTTP: Response Time

RTT: time between client sending a small packet to server and and receiving reply

Response time for fetching one object:

- one RTT to set up TCP connection
- one RTT for HTTP request and first byte of HTTP response to reach the client
- object transmission time

total = 2*RTT+transfer time



Photos from Yellowstone N

Alex

web.cs.ucla.edu/classes/spring16/cs118/assets/class/

# Photos from Yellowstone National Park

Yellowstone National Park

# http://web.cs.ucla.edu/classes/spring16/cs118/

```
<!DOCTYPE html>
<html>
  <head>
    <title>Photos from Yellowstone National Park</title>
  </head>

  <body>
    <h1>Photos from Yellowstone National Park</h1>

    <p>
      <img src="IMG_4083.jpg" width="400px" alt="Hot pool"/>
      <img src="IMG_4113.jpg" width="400px" alt="Geiser" />
    </p>

    <a href="https://en.wikipedia.org/wiki/Yellowstone_National_Park">Yellowstone
National Park</a>
  </body>
</html>
```

What would be overall timeline to open this web page (say, with RTT = 4 sec)?

# Response time: fetching 2 objects

User clicks URL www.foo.com/index.html

0. Host www.foo.com runs HTTP server process, waiting for incoming requests

1. HTTP client sends TCP connection open request to www.foo.com

2. HTTP server responds with "accept TCP connection"

3. HTTP client sends HTTP request message asking for object index.html

4. HTTP server receives request message, forms a reply containing requested object, and sends it.

as soon as the server receives ACK for the reply data, HTTP server closes TCP connection.

5. HTTP client receives response message (a html file), parsing the html file, finds 2 referenced jpeg objects

6. Steps 1-5 repeated for each of 2 jpeg objects

time

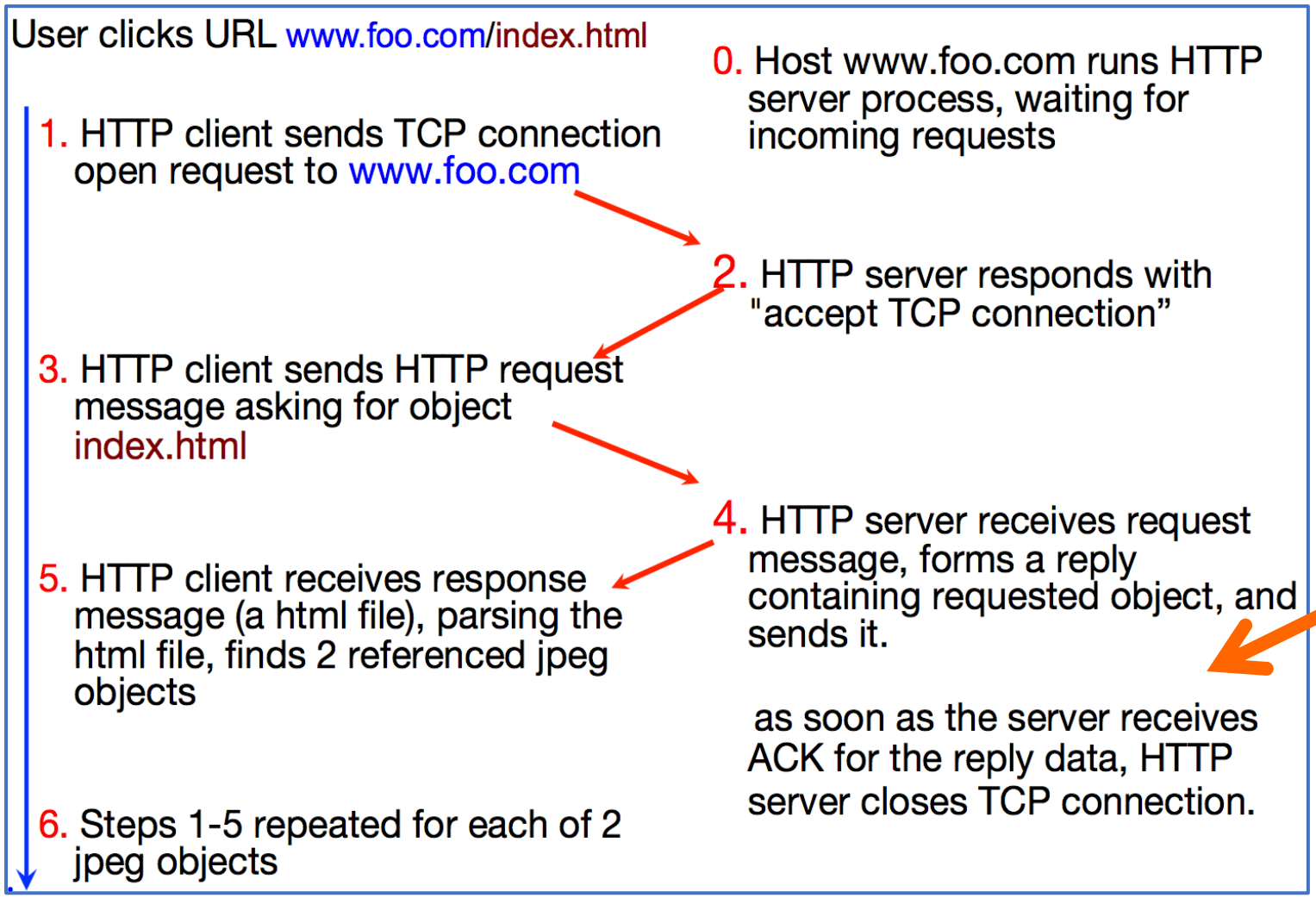
# Q: How to improve?

At most **one** object is sent over **one** TCP connection

After opening a TCP connection, use it to fetch send multiple objects (**Persistent HTTP**)

open multiple parallel TCP connections

# Persistent HTTP



# HTTP summary

## Nonpersistent HTTP

- At most **one** object is fetched over a single TCP connection.
- HTTP/1.0 uses nonpersistent TCP connections
  - Described in RFC 1945

    http://tools.ietf.org/html/rfc1945

## Persistent HTTP

- Multiple objects can be sent over a single TCP connection between client and server.
- HTTP/1.1 uses persistent connections
  - Described in RFC2616

Either non-persistent or persistent HTTP can set up multiple TCP connections in parallel to speed up data fetching

# Persistent HTTP: one more detail

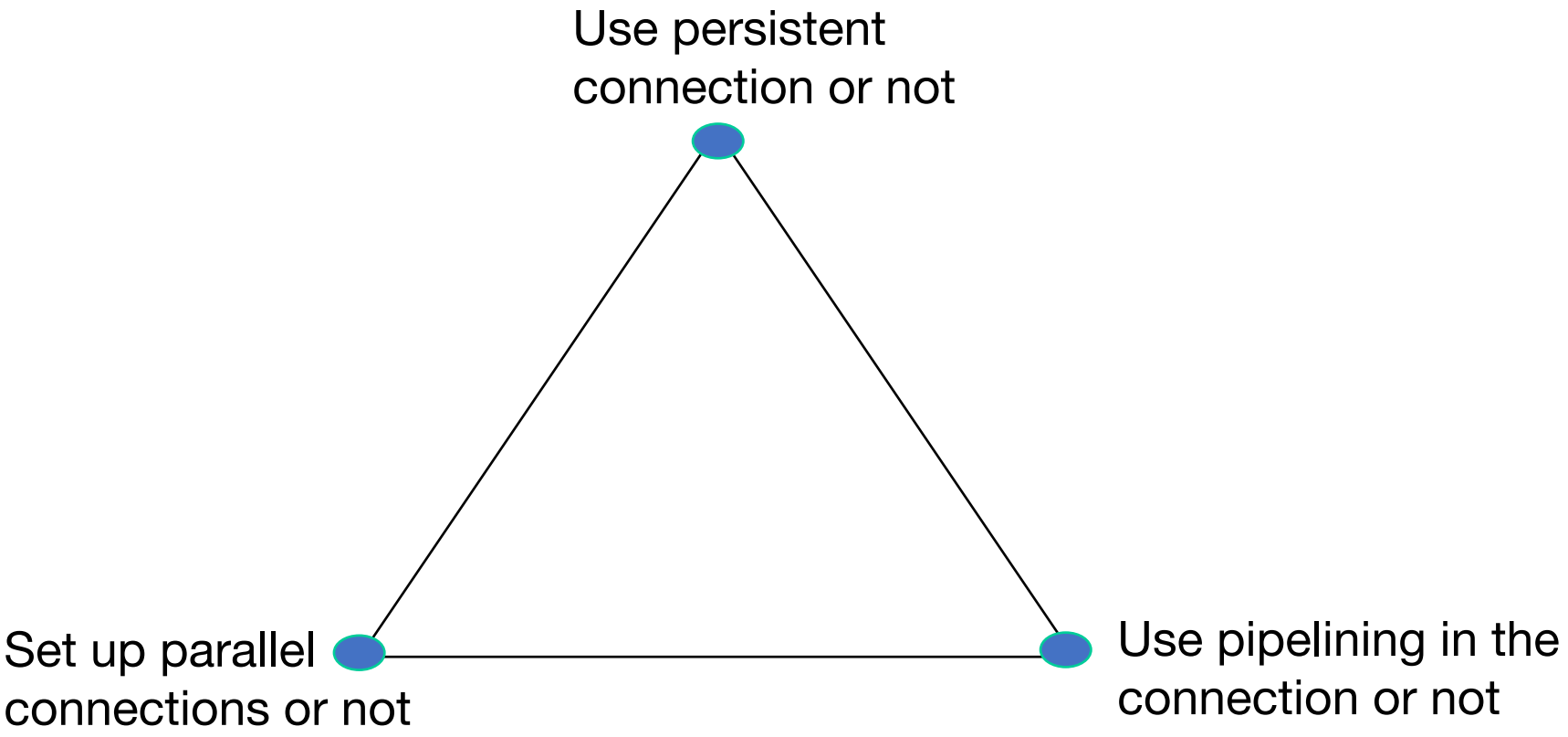
Persistent ***without*** pipelining:

- client issues new request after previous response *has* been received
- Total delay: one RTT for *each* object plus data transfer time

Persistent ***with*** pipelining:

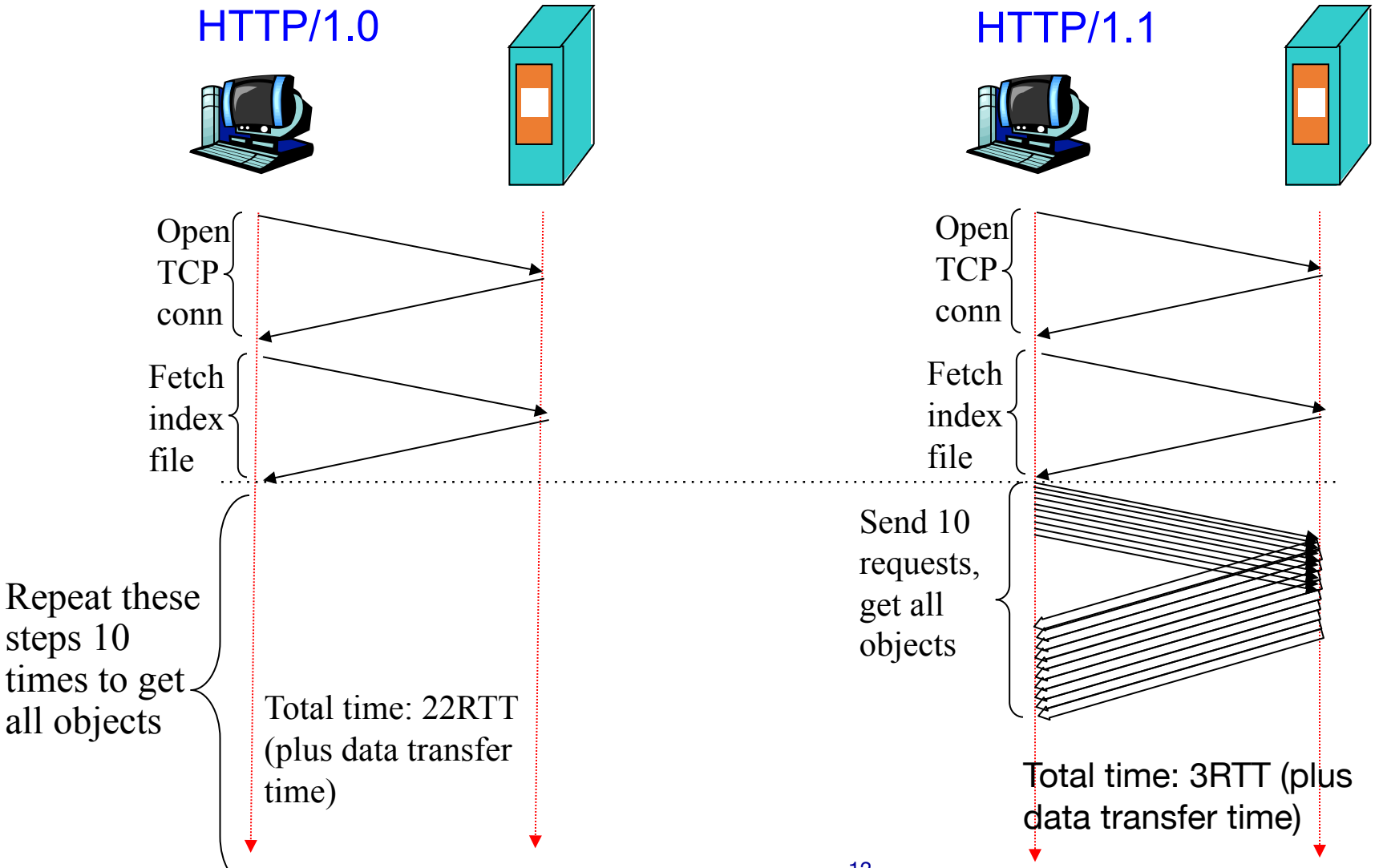
- client sends requests as soon as it sees a referenced object
- Total delay: one RTT plus data transfer time for all objects

# Three factors in HTTP fetching



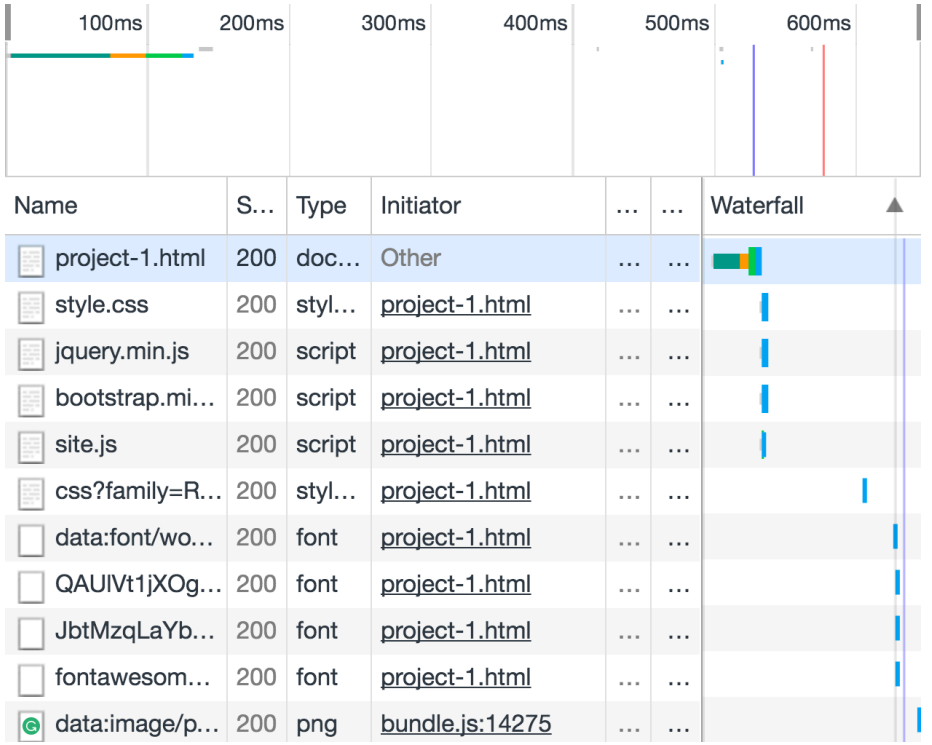
# HTTP/1.0 vs HTTP/1.1 w/ pipelining

Fetching one index.html file ***and*** 10 referenced jpeg files



# Timeline to Retrieve a Web Page

- http://web.cs.ucla.edu/classes/spring17/cs118/project-1.html
  - http://web.cs.ucla.edu/classes/spring17/cs118/css/style.css
  - https://ajax.googleapis.com/ajax/libs/jquery/3.1.1/jquery.min.js
  - http://web.cs.ucla.edu/classes/spring17/cs118/js/bootstrap.min.js
  - http://web.cs.ucla.edu/classes/spring17/cs118/js/site.js
  - https://fonts.googleapis.com/css?family=Raleway:400,700
  - … a few more




Is this HTTP/1.0 or HTTP/1.1?

What difference we would we with the other version?

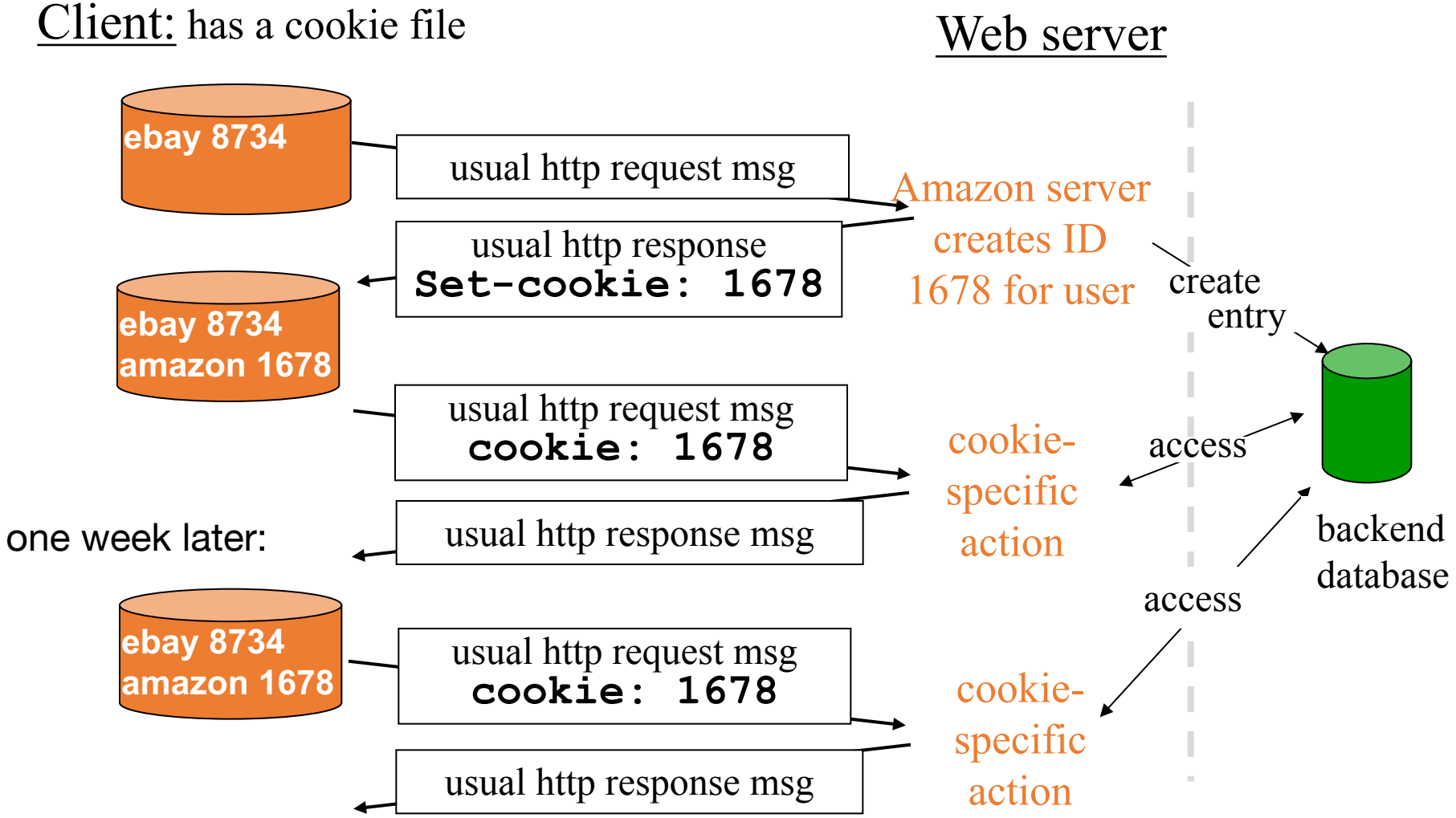
# Higher-Level HTTP Semantics: Client State

# You are Being Tracked!


Amazon/Gmail knows who you are

and what is your shopping history

# User-server interaction: cookies



# Cookies make use of these ingredients:

- Website:
  - If an HTTP request does not have a “cookie” field, include a “set-cookie” header line in HTTP response message
  - Storing in back-end database the cookie with info about the user’s requests
- Browser:
  - Save all received cookies in a local cookie file
  - When visiting a webpage: if a match cookie found, include it in “cookie” header line in HTTP request

# Cookies: usefulness versus privacy

Cookie can

- Bring convenience to you
- Bring recommendations

- Permit a website to learn a lot about you
- Advertising companies can obtain user info across multiple sites

# Third Party Cookie

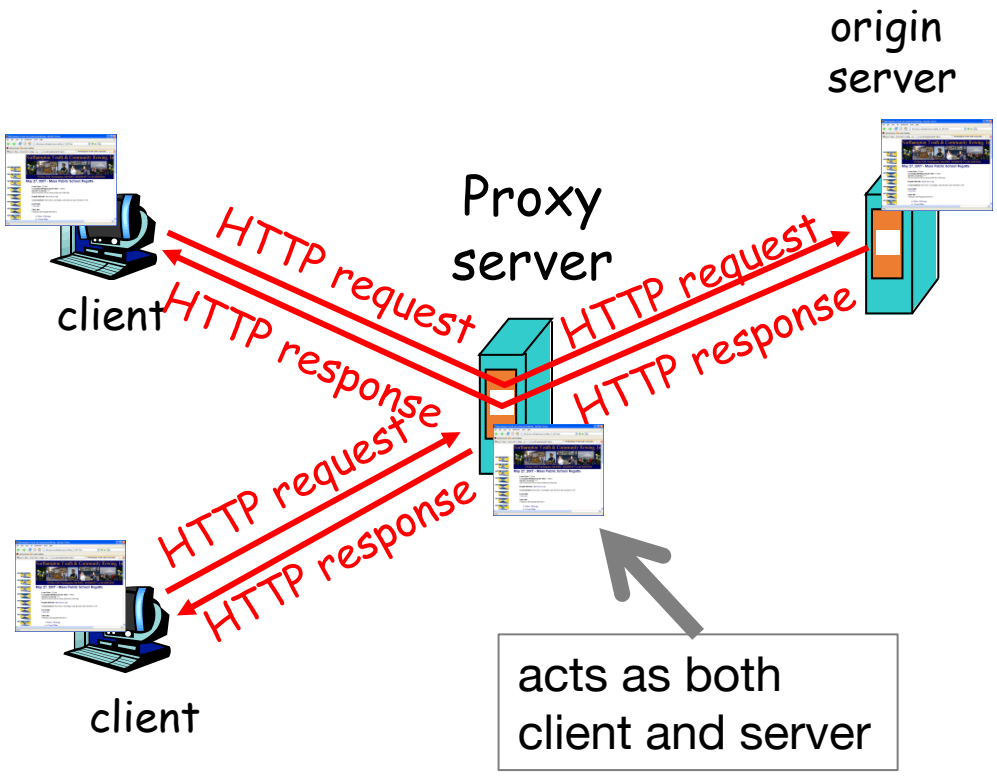
- A user visits www.example1.com, which contains an advert from ad.foxytracking.com, which, when downloaded, sets a cookie belonging to (ad.foxytracking.com).
- When the same user visits another website, www.example2.com, which also contains an advert from ad.foxytracking.com → user gets another cookie belonging to (ad.foxytracking.com).
- The advertiser can then use these cookies to build up a browsing history of the user across all websites hosting ads from this advertiser.

# Leveraging Statelessness and Data Centricity of HTTP

# HTTP Proxying

important

- User sets browser to enable web accesses via cache
- Browser sends all HTTP requests to cache
- Cache:
  - If requested object in cache: returns object
  - else requests object from the server, then returns object to client



# Why Web caching

- Reduce load on the origin server
  - enables “poor” content providers to effectively deliver content
- Reduce traffic on an institution’s access link
- Reduce response time for client request

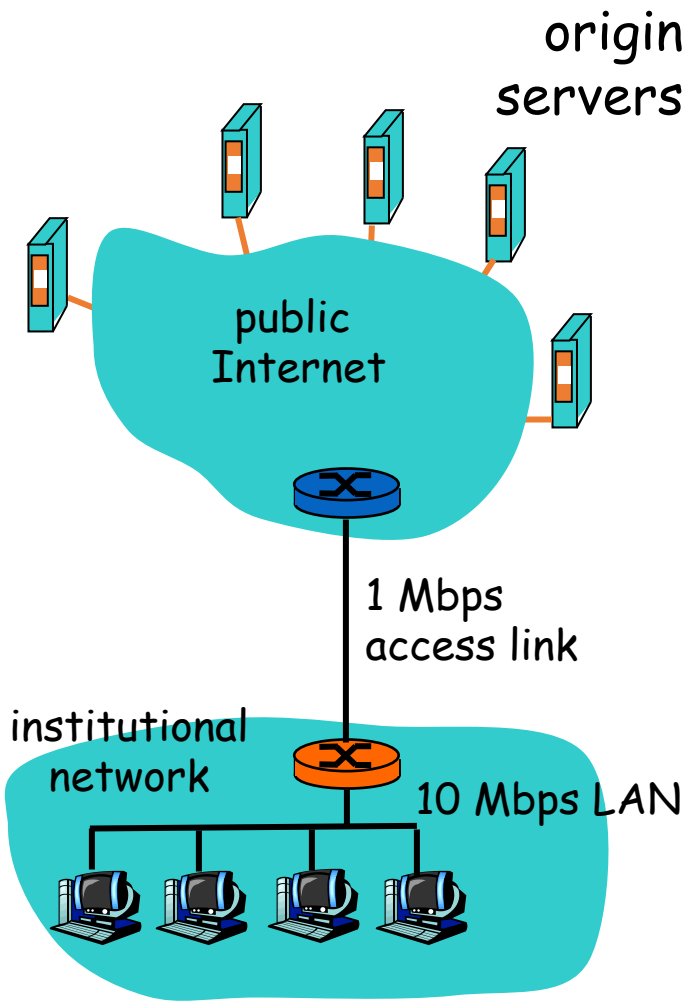
# Example: without caching

## Assumptions

- avg. web request rate = 10 reqs/sec
- average object size = 100,000 bits
- RTT from institutional router to any web server and back = 500msec

## Consequences

- utilization on LAN = 10%
- utilization on access link = 100%
  - Queuing delay at institutional router: may grow without bound
- Total delay = 500msec + long queuing delay



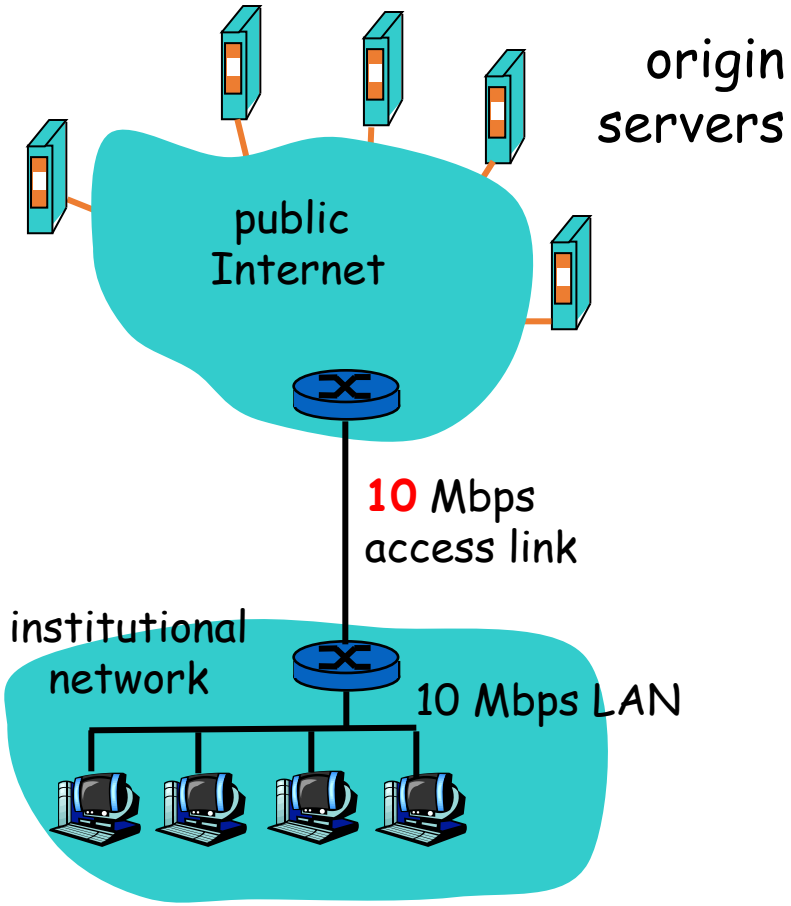
# Solution-1: buy more bandwidth

Increase bandwidth of access link to 10 Mbps

Consequences

- Can be costly
- utilization on LAN = 10%
- utilization on access link = 10%
- delay for each request

  ≈ 500msec+ (queueing delay negligible)



# Solution-2: Adding a local cache

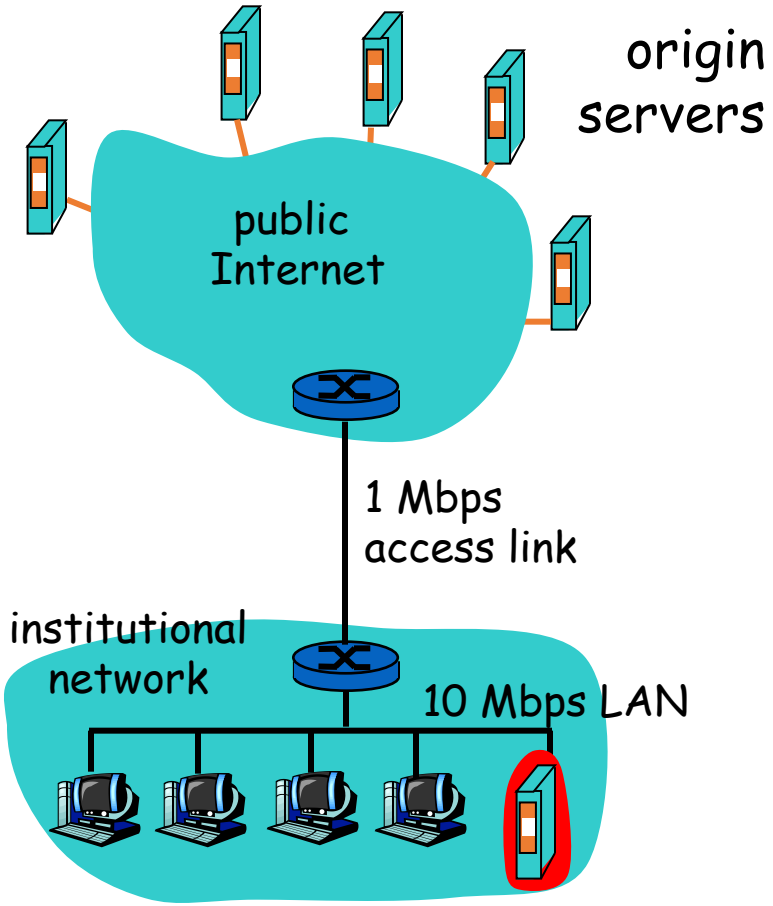
## Consequences

- assume hit rate = 0.4: 40% requests will be satisfied by the data in the cache (delay ≈ 10msec)
- Remaining 60% requests served by origin servers
- utilization of access link reduced to 60% → much smaller queueing delay (say 30 msec)
- avg delay for each packet

  = 0.6x(Internet delay + access delay) + 0.4x(LAN+cache delay)

  = 0.6x(530 msec) + 0.4 x (10 msec)

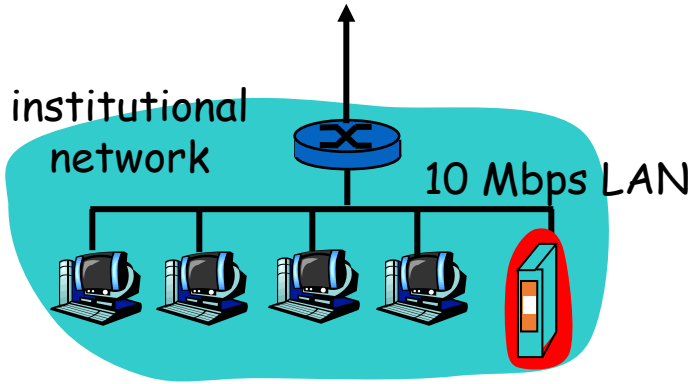
  = 322msec



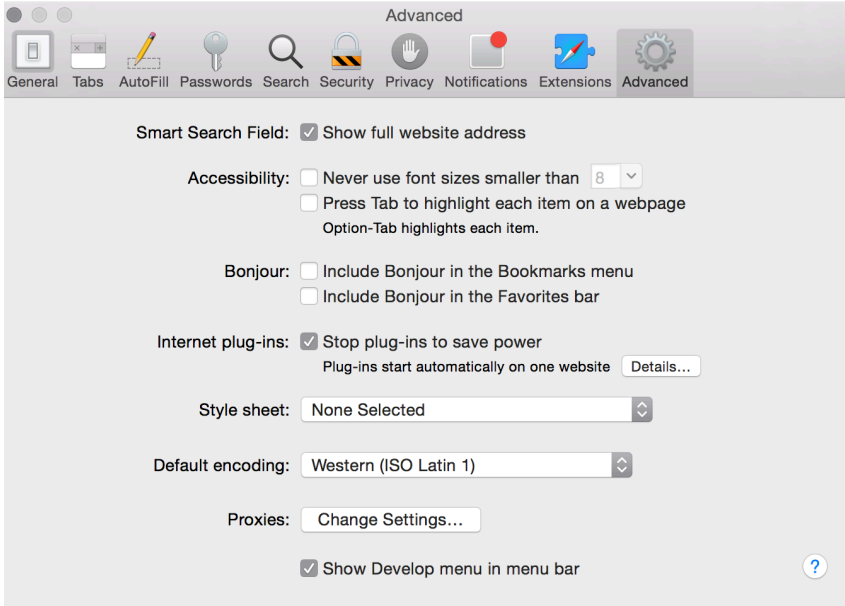

Institutional cache

# Having a local cache: not a free lunch

- Need to configure browsers to use the local cache
- What if the local cache fails?
- What if the cached content is obsolete?
- What about secure HTTP connections?




Institutional cache

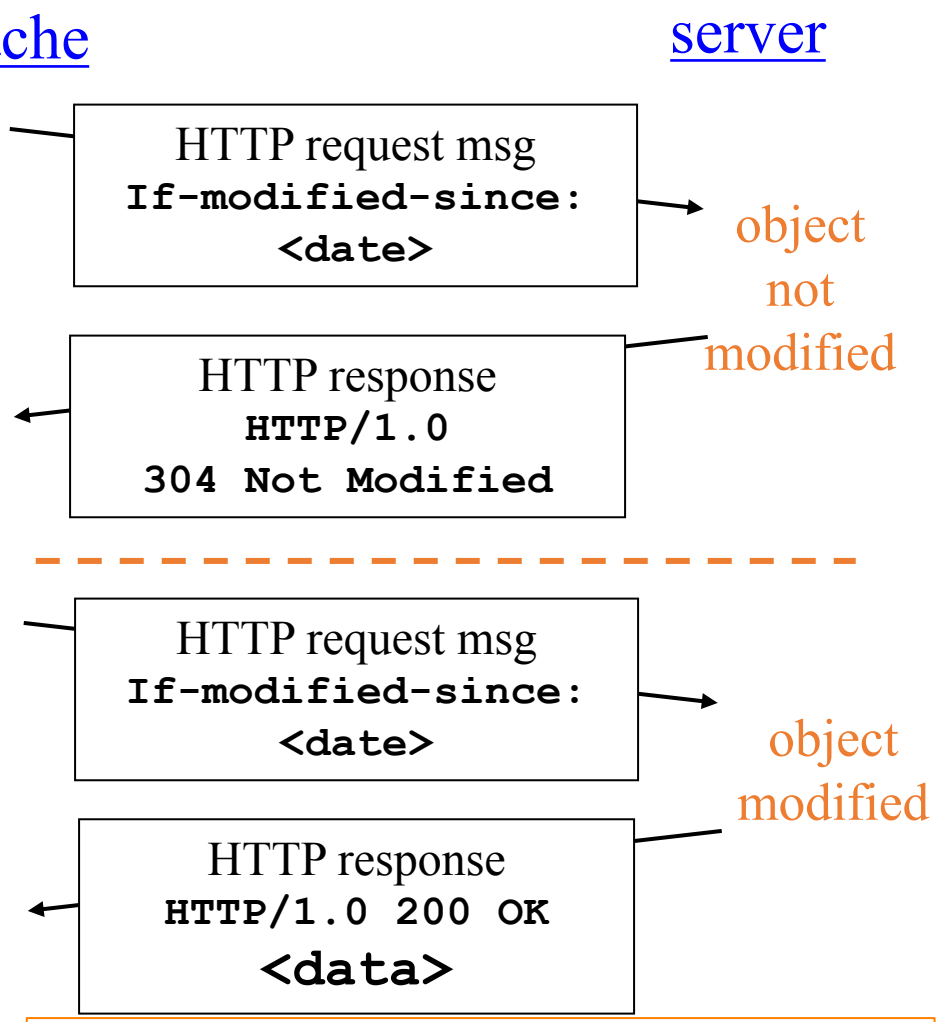


# HTTP Conditional GET

- Fetch content only if it has changed since previous fetch
- Cache: specify date of cached copy in HTTP request

  If-modified-since: <date>
- server: response contains no object if cached copy is up-to-date:

  HTTP/1.1 304 Not Modified




FYI: for http status code, see
http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html

# HTTPS and HTTP Proxying

- Does it work?

# How a Concept of HTTP Proxy is Used Today

- Role of HTTP proxies today is diminishing
  - Access links are getting faster and cheaper
  - Some universities/companies still use it
    - Why?
- However, concept of HTTP proxy is used a lot today in a slightly different form
  - How?

# HTTP/2
# (HTTP/1.1 is Not Fast Enough)

# Why HTTP/2

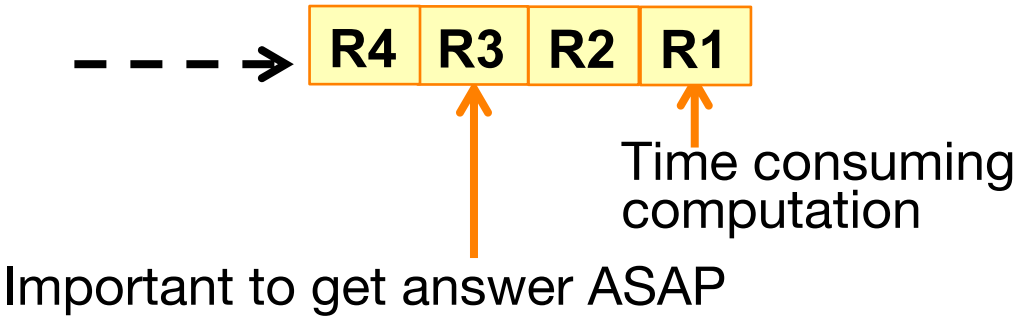
HTTP/1.1 with pipelining: not good enough

Some measurement numbers: an average Web application is now

- composed of more than 90 objects
- fetched from more than 15 distinct hosts
- totaling more than 1,300KB of transferred data
  - On average each object <15KB
    - Some can be very big, e.g. large image files
    - That means some others can be very small

# Some of HTTP/1.1's performance issues

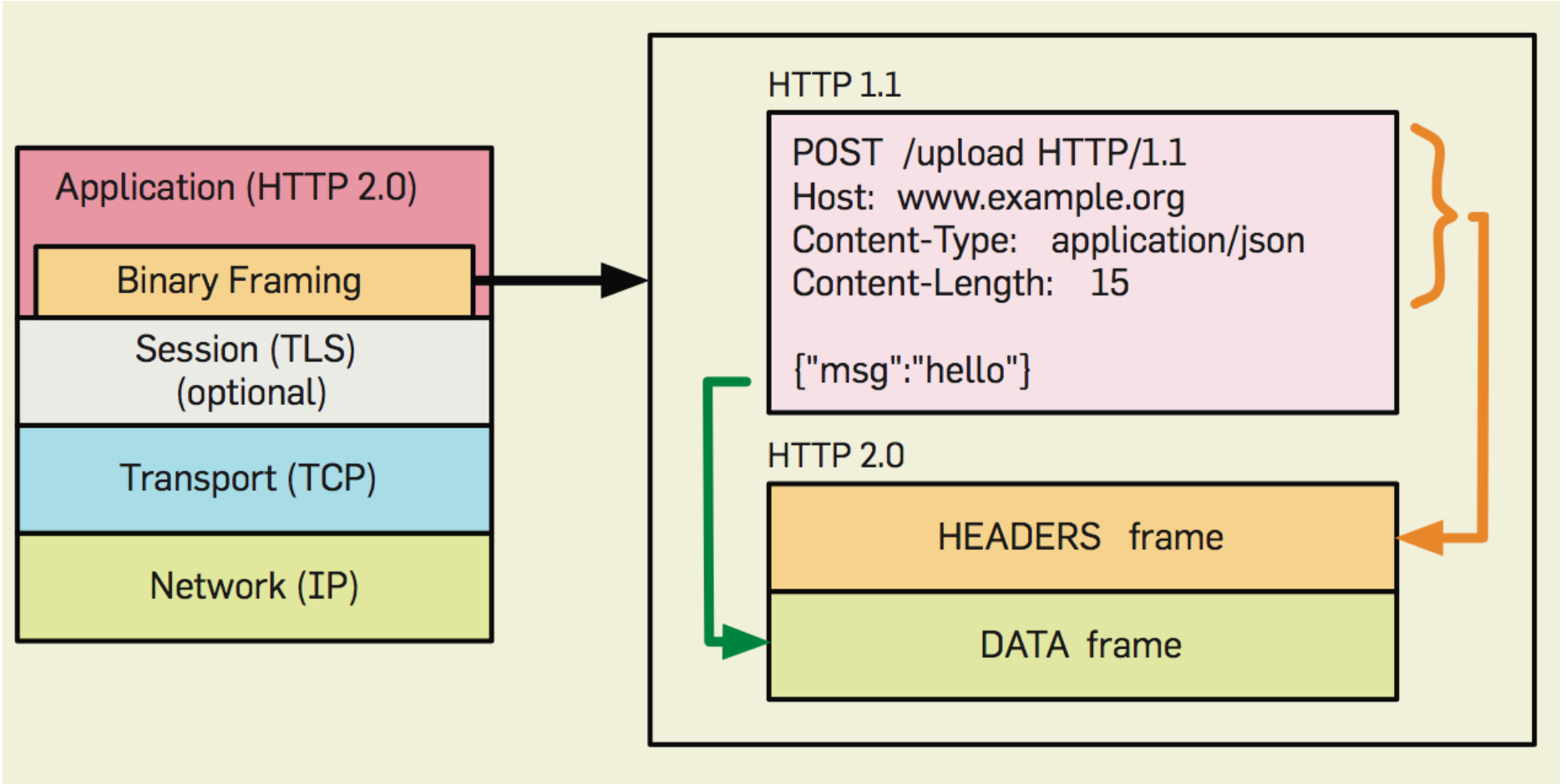
- Pipelining: not supported by default
  - Consequence: take one RTT to fetch each object
  - Work-around: app developers bundle multiple small objects into a big one
- Head-of-line blocking: HTTP/1.1 handles requests in strict sequential order
  - A request for a large file, or some dynamic computation, can take time, blocking all requests after it



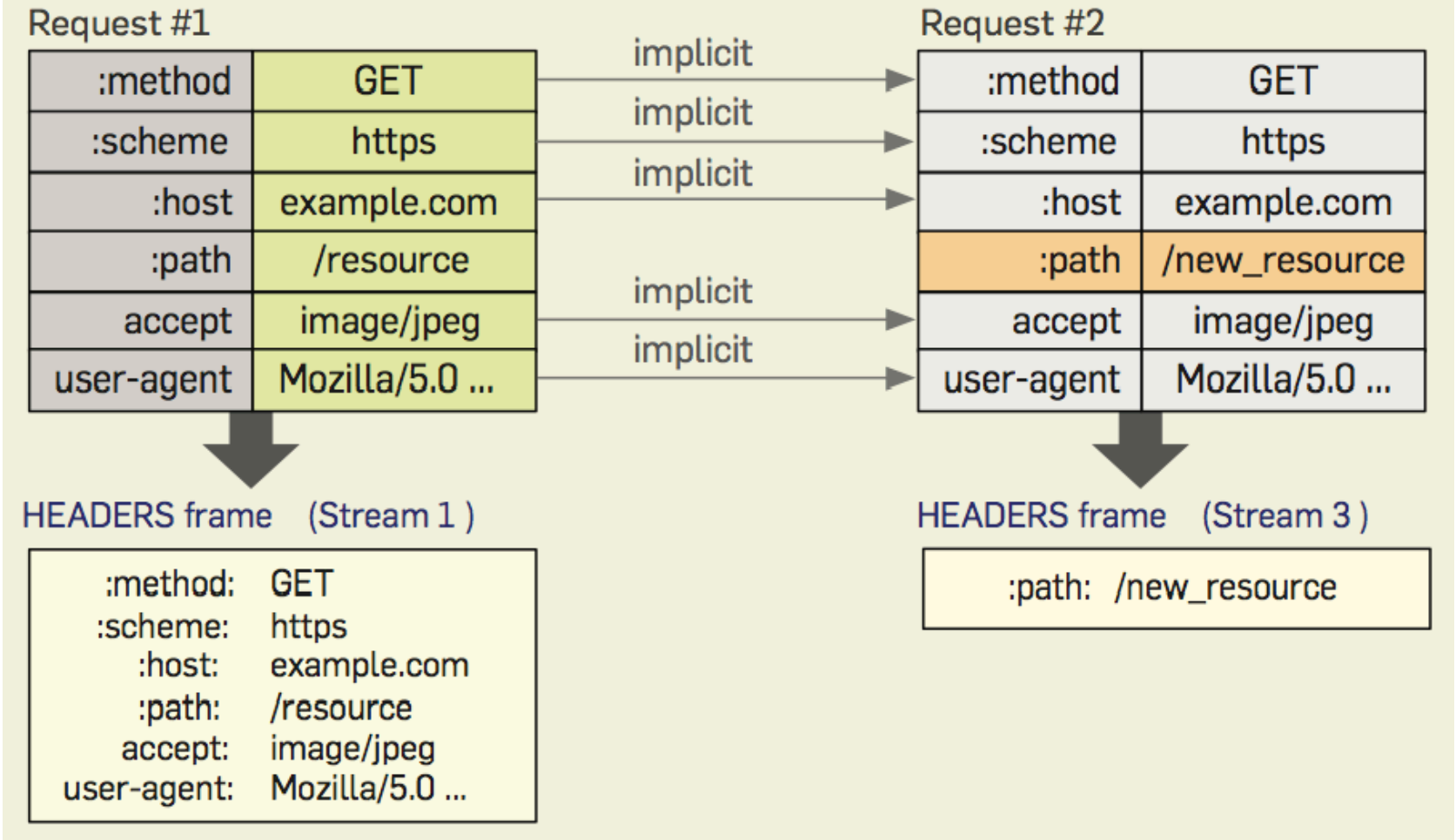
# Some of HTTP/1.1's performance issues

- Pipelining: not supported by default
  - Consequence: take one RTT to fetch each object
  - Work-around: app developers bundle multiple small objects into a big one
- Head-of-line blocking: HTTP/1.1 processes requests in strict sequential order
  - A request for a large file, or some dynamic computation, can take time, blocking all requests following it
  - Work-around: open multiple TCP connections
- Big HTTP header with repetitive information
  - No way to get around this

# HTTP/2: Binary Framing



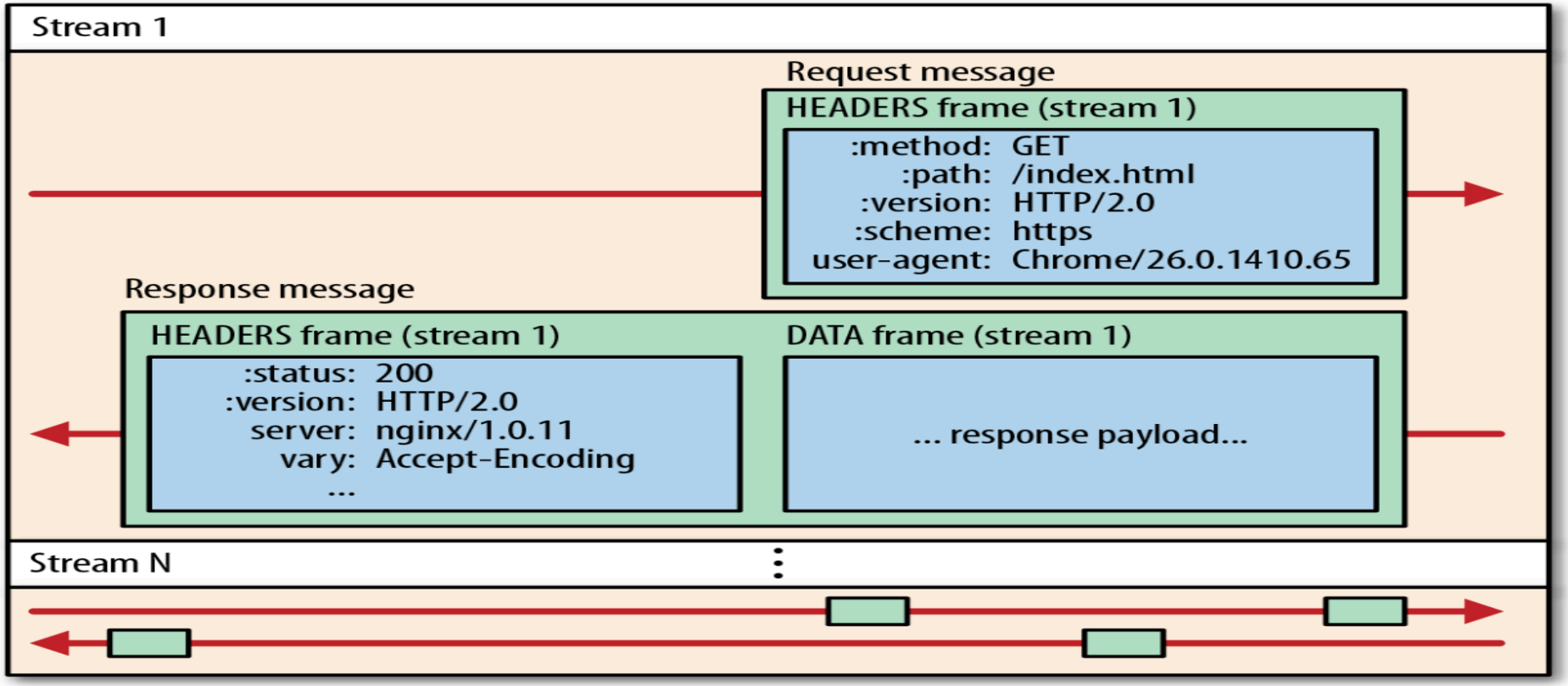
# HTTP/2: Header Compression




- Both browser & server keep a header table until the TCP connection closes

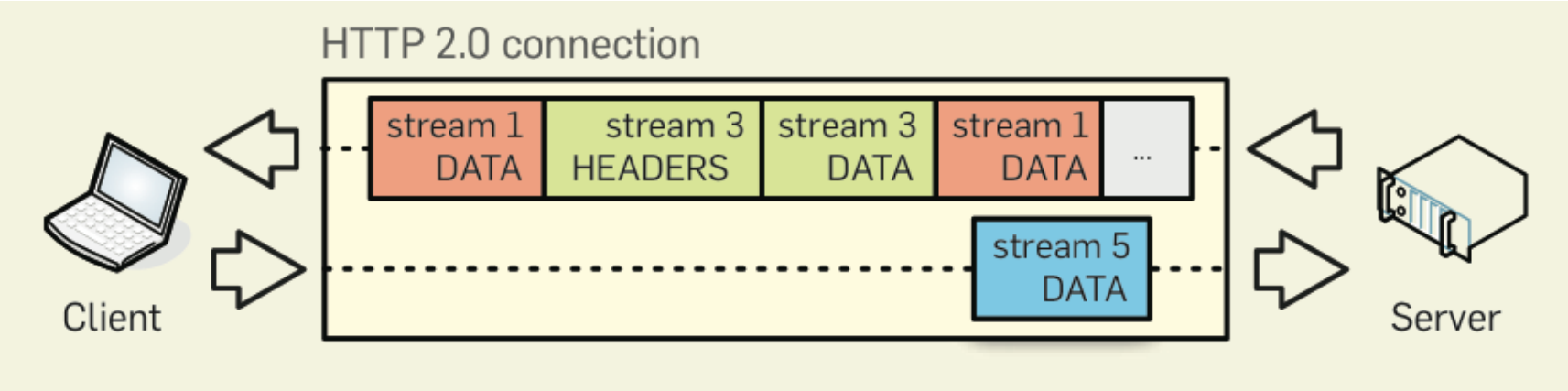
# HTTP/2.0: Frame, Message, Stream

- Frame: basic communication unit
- Message: an HTTP request or response (using one or multiple frames)
- Stream: a virtual channel with priority, carrying messages between client and server



# HTTP/2: Multiplexing

- Only need a single TCP connection between HTTP client and server
- The connection can carry multiple streams



# HTTP/2 Performance Improvements

- Reduce HTTP header overhead
  - Binary encoding
  - Header compression
- Single TCP connection between each client-server pair
  - Multiple streams
  - Each stream: deliver messages between HTTP client—server
  - Big messages are broken down to multiple frames
  - Frames from all streams can be interleaved

# QUIC
# (HTTP/2 is Not Fast Enough)

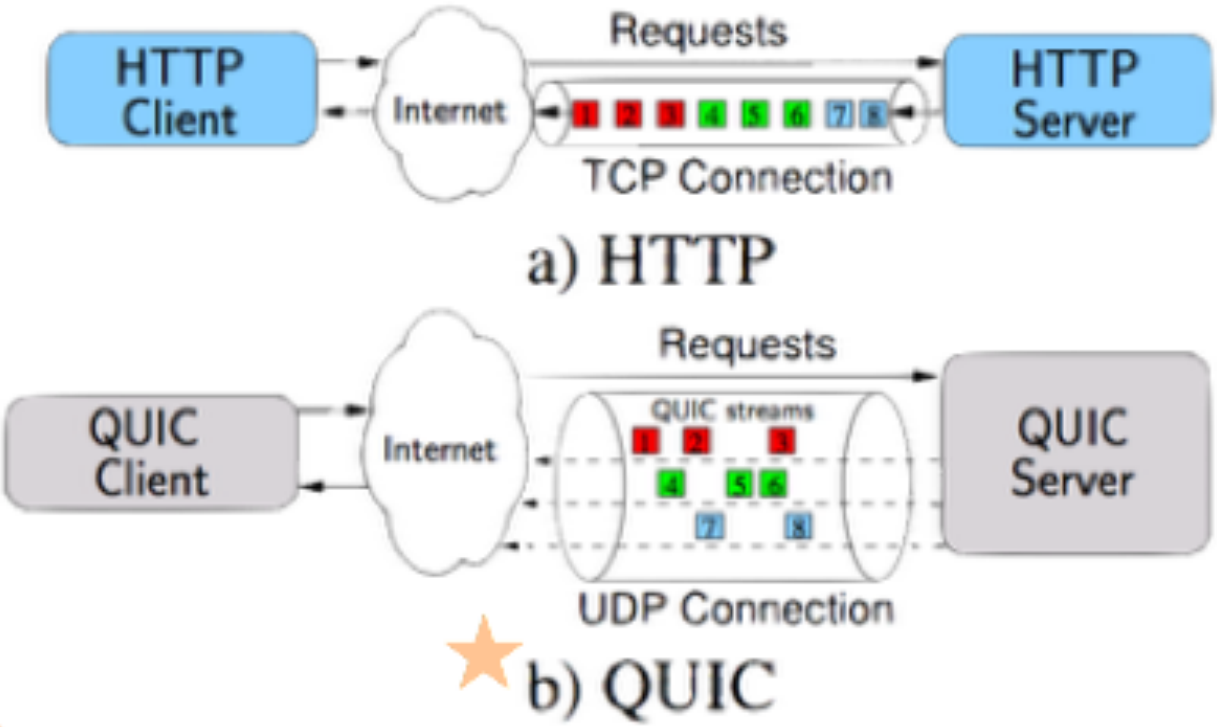
# Why QUIC

- HTTP/2 inherits TCP problem
  - Head-of-line blocking
  - Congestion control
- Require separate security layer (TLS)
  - Each connection must negotiate security parameters

- Key features of QUIC over TCP+TLS+HTTP2 include:
  - Runs over UDP (app-managed connection, congestion, and flow control)
  - Connection establishment latency (reuse security parameters from previous connections)
  - Improved congestion control (app-managed)
  - Multiplexing without head-of-line blocking
  - Forward error correction
  - Connection migration

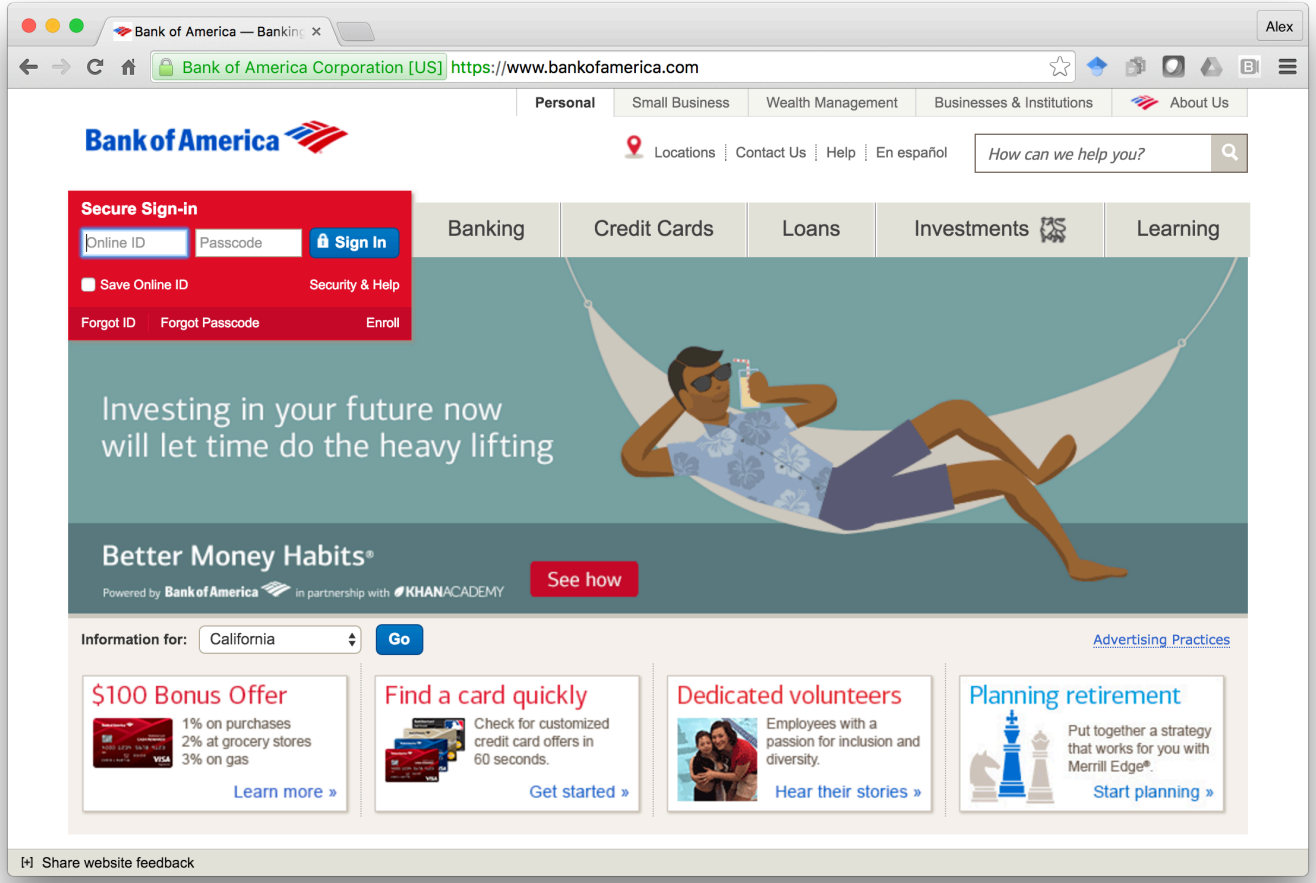
# A Few More Details on QUIC

- Use “streams”
  - No head-of-line blocking
- Compresses headers
- Server can “push” response, predicting client’s request
- Request prioritization
- Uses “connection identifier”

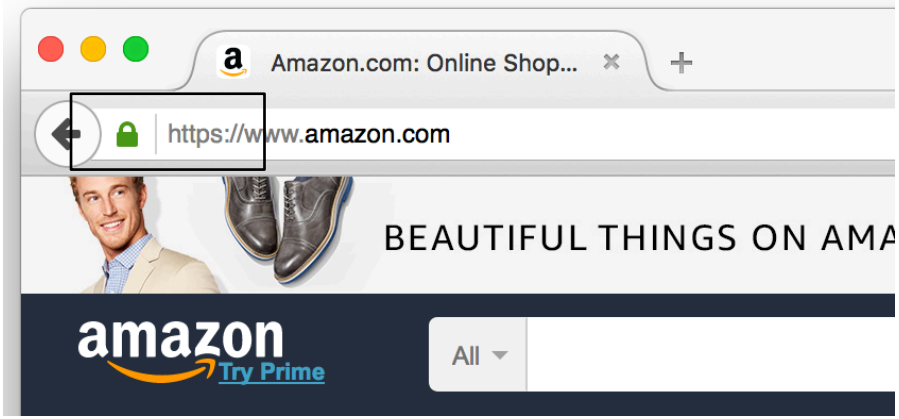


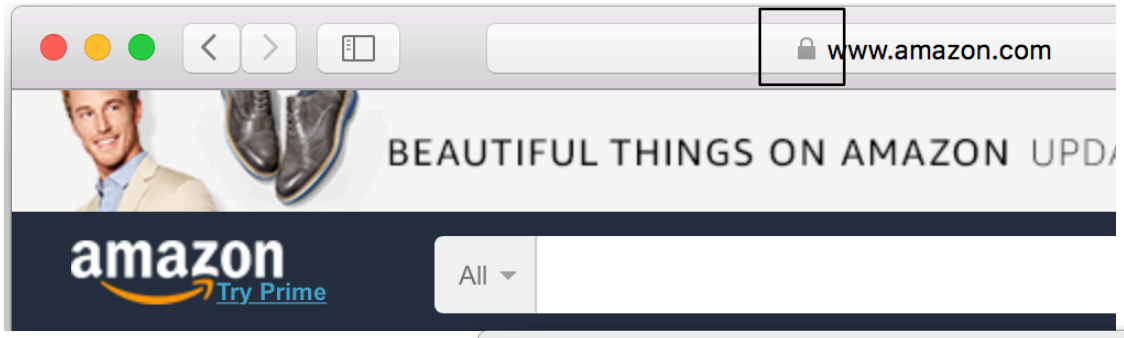
# HTTPS: Secure HTTP

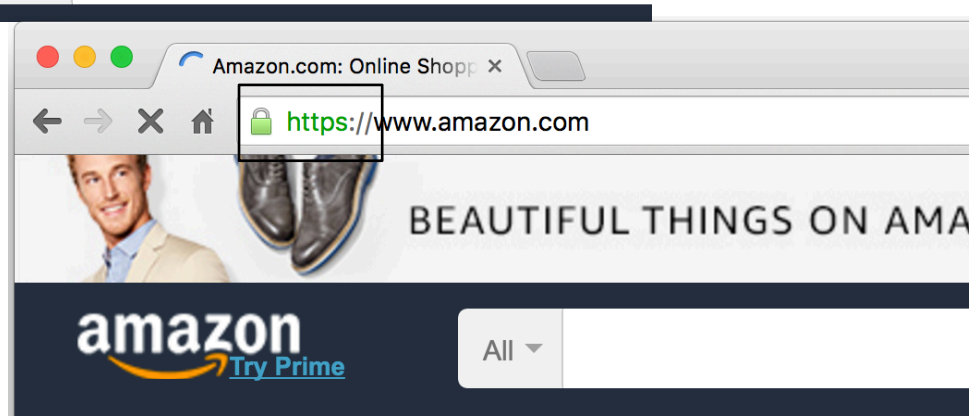
- Ensure that you're receiving data from place you think you do
- Ensure secrecy of information exchanges



# HTTPS: Sites with Domain Validation (DV) Certificates

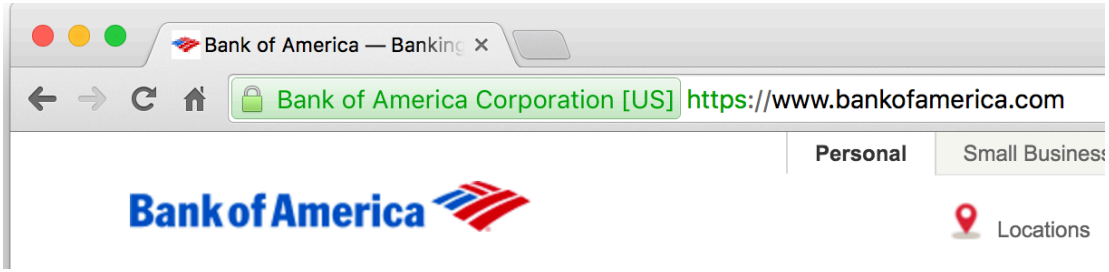


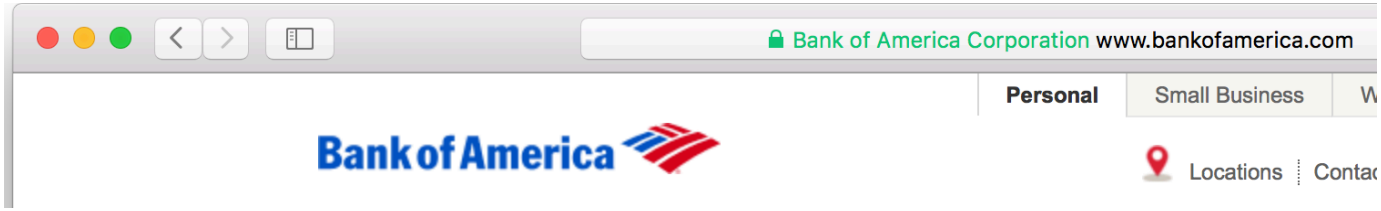



# HTTPS: Sites with Extended Validation (EV) Certificates

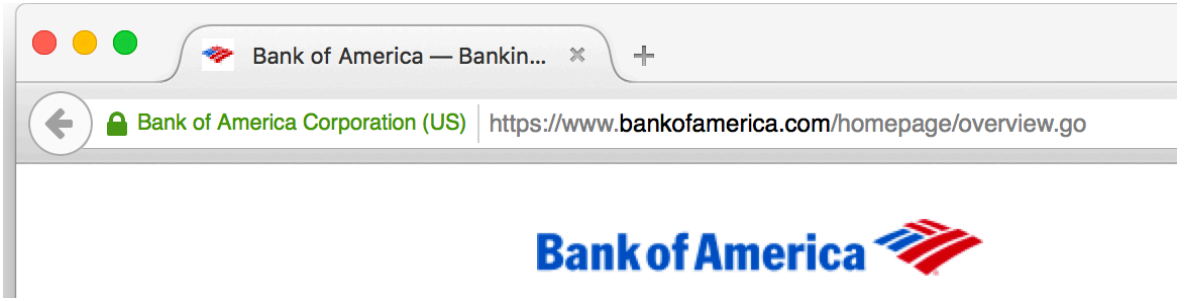
- the Certificate Authority (CA) checks the right of the applicant to use a specific domain name PLUS it conducts a THOROUGH vetting of the organization




Chrome



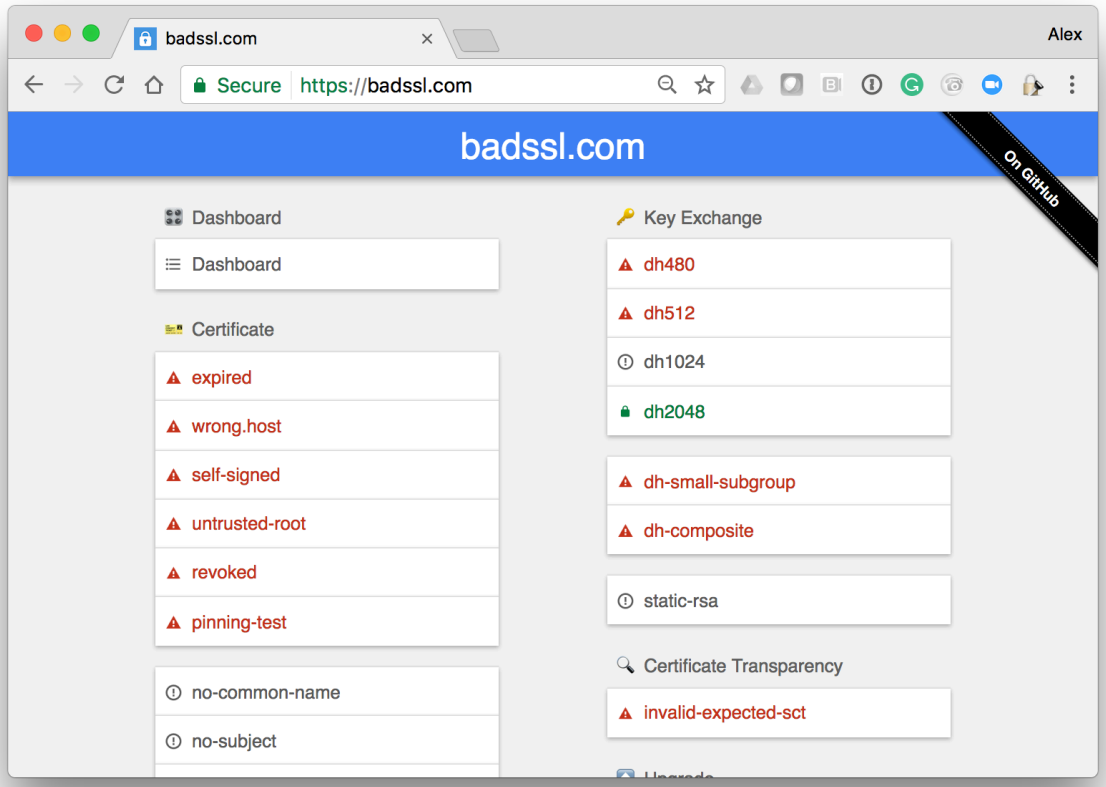

Safari




Firefox

# HTTPS (TLS) Certificate States

- Valid
- Expired
- Wrong host
- Untrusted root
- Revoked
- Pinned




- \+ more

Is your browser safe?

# Summary of today's lecture

The most important points to take home:

- Non-persistent HTTP: how long it takes to fetch a webpage (with multiple objects)
- Persistent HTTP: how long it takes to fetch a webpage, if
  - Pipelining, or
  - no-pipelining
- Using multiple parallel TCP connections to speed up data fetching
- Web caching: what are the benefits and issues
- HTTP/2, QUIC: future protocols to speed up web
- HTTPS: beware and be ready