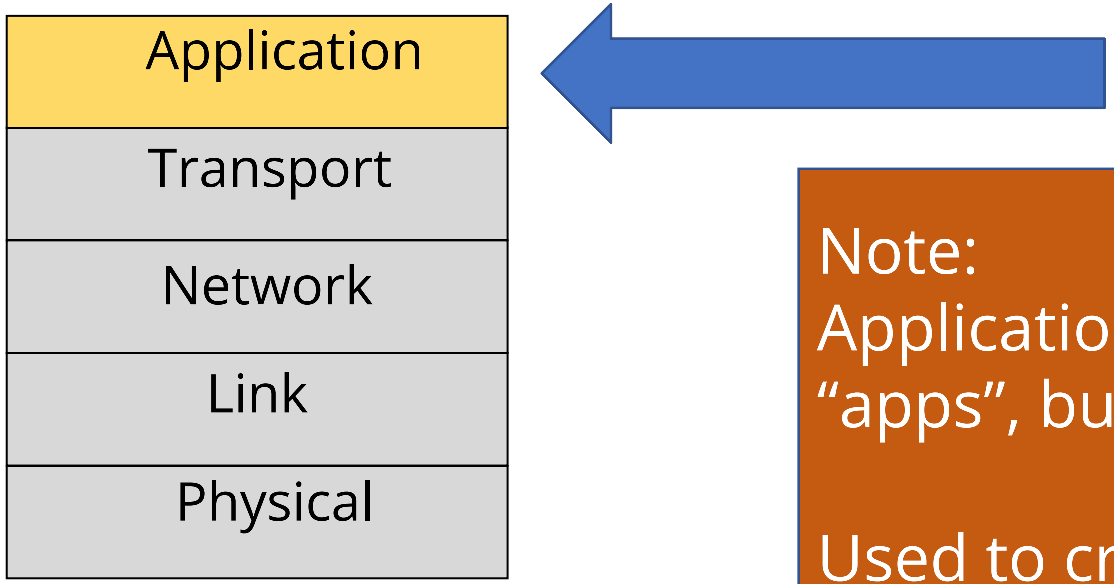
Applications!!!

Where we are in the Course

Remainder of Today: Real application layer protocols built over networks

- We made it! Know all the details now, so can have a useful and realistic model of high-level systems!

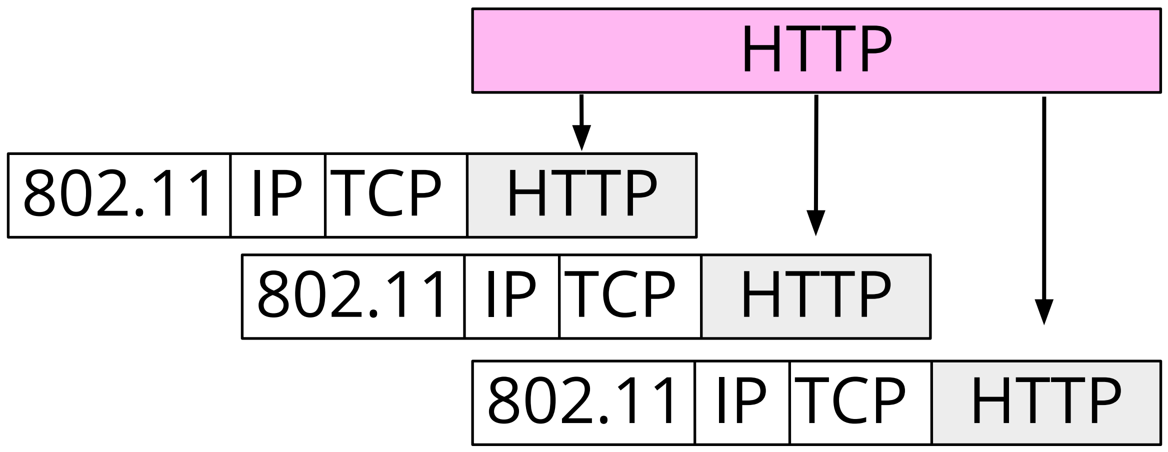


Note:
Application layer protocols are often part of "apps", but don't necessarily need a GUI…

Used to create a functioning application

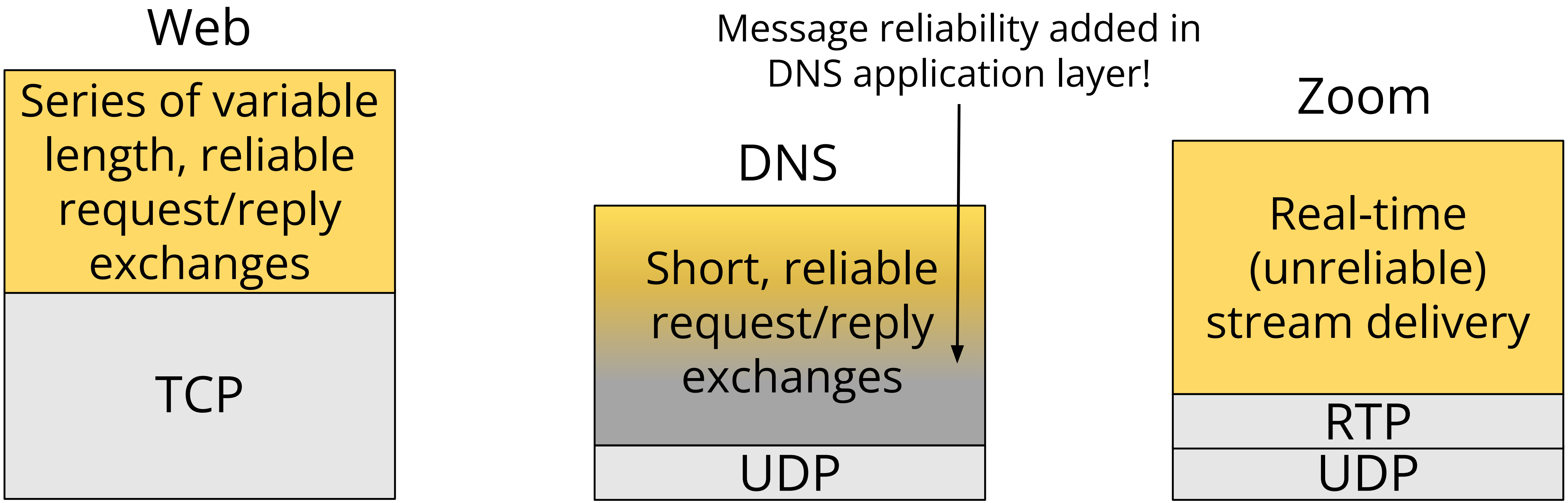
Recall

- Application layer messages are often split over multiple packets
  - Or may be aggregated in a packet …



Application Communication Needs

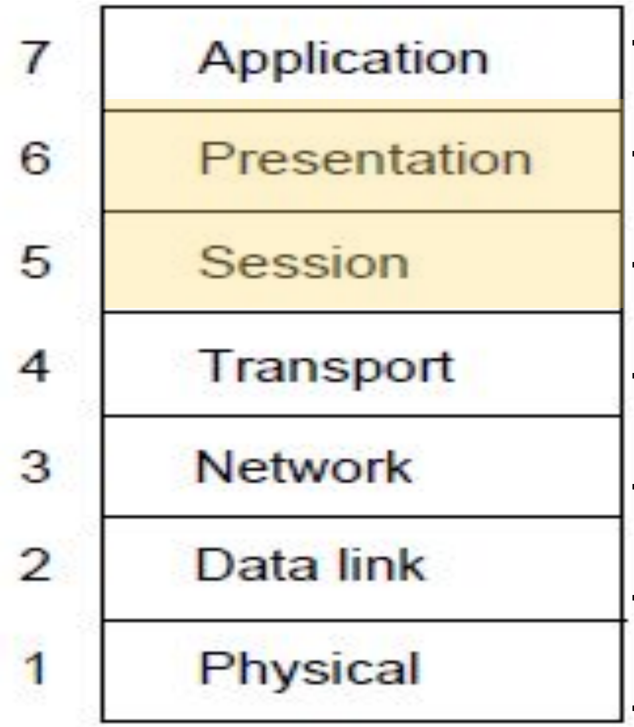
- Vary widely; must build on Transport services



OSI Session/Presentation Layers

- Remember this? Two relevant concepts …

Considered part of the application, not strictly layered!



– Provides functions needed by users
– Converts different representations
– Manages task dialogs
– Provides end-to-end delivery
– Sends packets over multiple links
– Sends frames of information
– Sends bits as signals

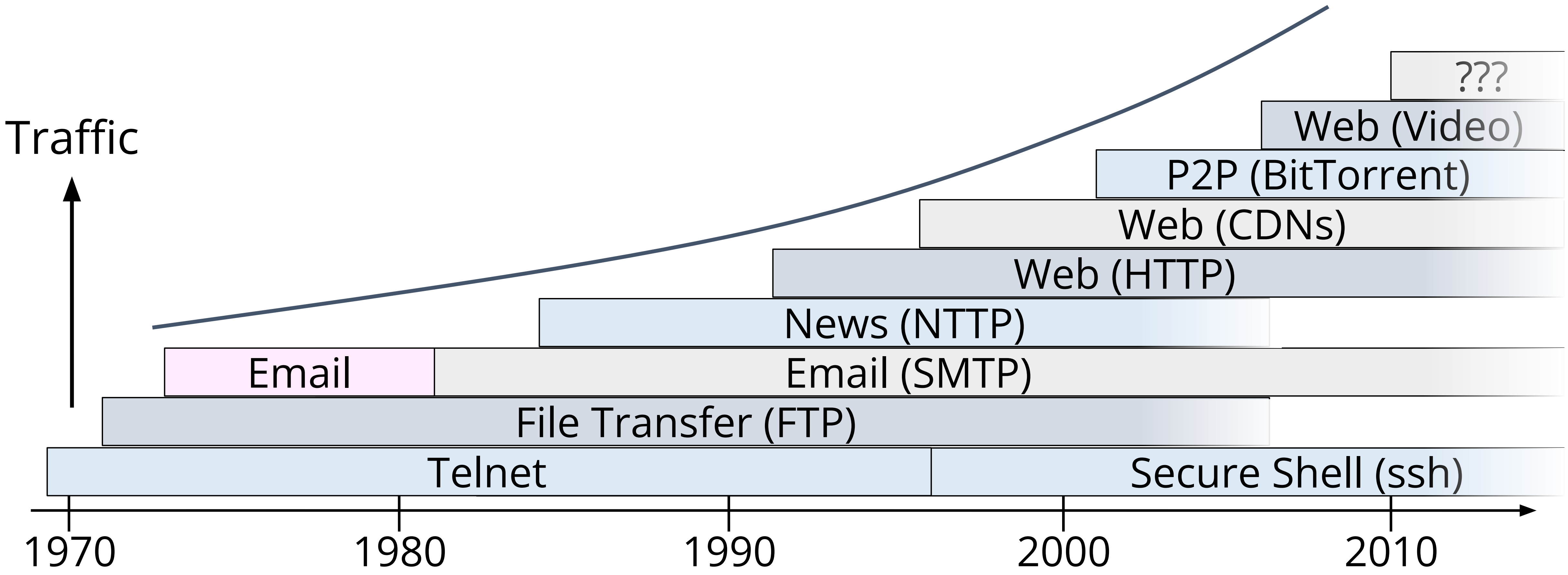
Session Concept

- A session is a series of related network interactions in support of an application task
  - Often informal, not explicit
- Examples:
  - Web page fetches multiple resources
  - Zoom call involves audio, video, chat

Presentation Concept

- Apps need to identify the type of content, and encode it for transfer
  - These are Presentation functions
- Examples:
  - Media (MIME) types, e.g., image/jpeg, identify content type
  - Transfer encodings, e.g., gzip, identify the encoding of content
  - Application headers are often simple and readable versus packed for efficiency

Evolution of Internet Applications



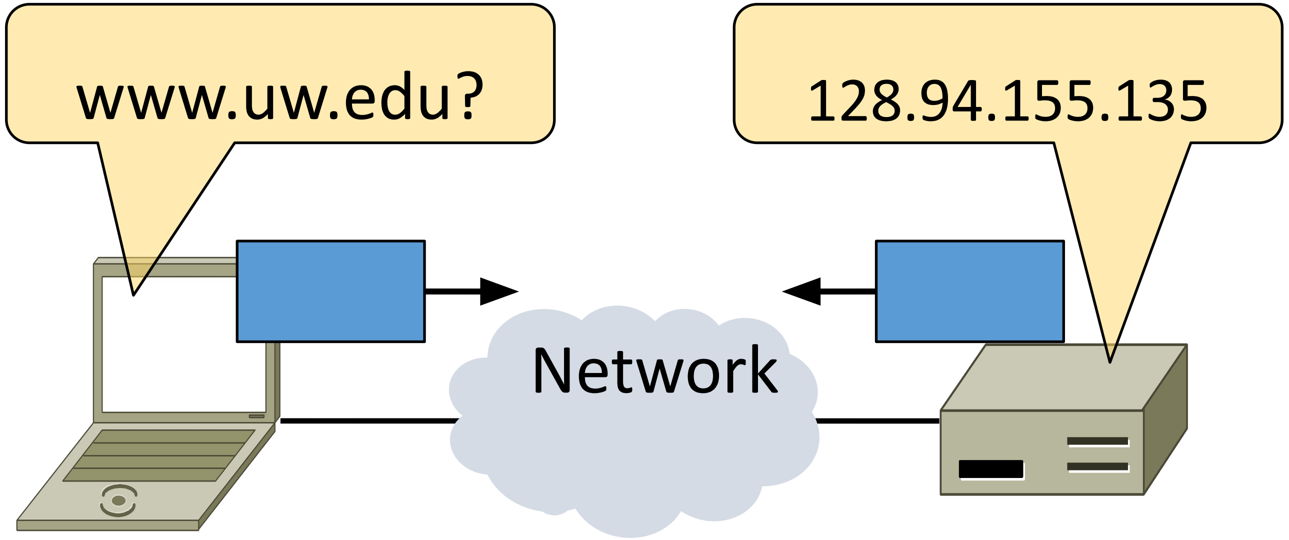
Evolution of Internet Applications (2)

- For a peek at the state of the Internet:
  - Akamai's State of the Internet Report (quarterly)
  - Cisco's Visual Networking Index
  - Mary Meeker's Internet Report
- Robust Internet growth, esp. video, wireless, mobile, cats
  - 82% of Internet traffic is video (2023)
  - Mobile traffic overtakes desktop (2016)
    - 54.4% in 2023
  - 53.1% of global Internet users are in Asia

Domain Name System

DNS

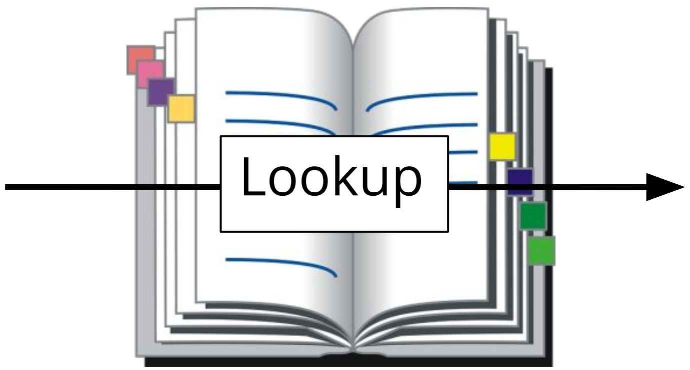
- Human-readable host names, and more…



Names and Addresses

- Names are higher-level identifiers for resources
- Addresses are lower-level locators for resources
  - Multiple levels, e.g. full name → email → IP address → Ethernet addr
- Resolution (or lookup) is mapping a name to an address

Name, e.g.
"Andy
Tanenbaum,"
or "flits.cs.vu.nl"



Directory

Address, e.g.
"Vrijie Universiteit,
Amsterdam"
or IPv4 "130.30.27.38"

Before the DNS – HOSTS.TXT

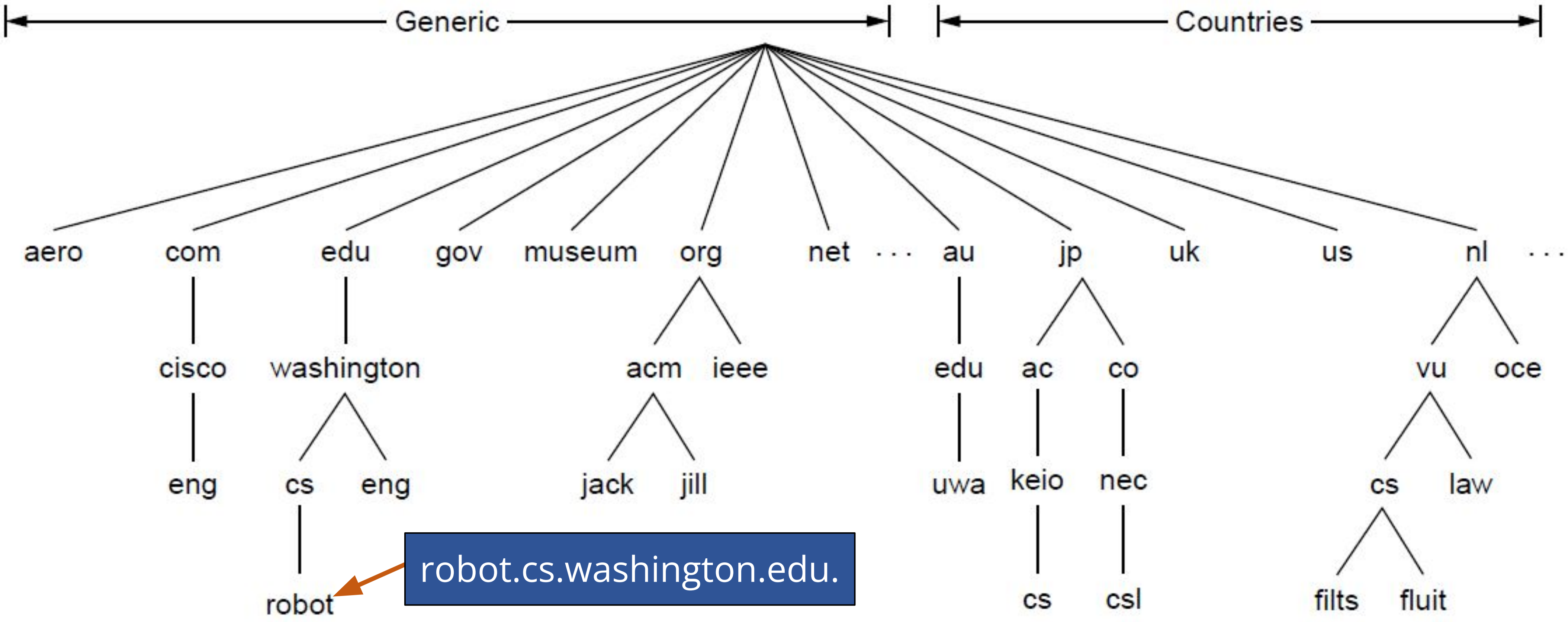
- Directory was a file HOSTS.TXT regularly retrieved for all hosts from a central machine at the NIC (Network Information Center)
- Names were initially flat
  - Became hierarchical (e.g., lcs.mit.edu) ~1985
- Not manageable or efficient as the ARPANET grew …

DNS

- A naming service to map between host names and their IP addresses (and more)
  - www.uwa.edu.au → 130.95.128.140
- Goals:
  - Easy to manage (esp. with multiple parties)
  - Efficient (good performance, few resources)
- Approach:
  - Distributed directory based on a hierarchical namespace
  - Automated protocol to tie pieces together

DNS Namespace

- Hierarchical, starting from "." (dot, typically omitted)



TLDs (Top-Level Domains)

- Run by ICANN (Internet Corp. for Assigned Names and Numbers)
  - Starting in ‘98; naming is financial, political, and international
- 700+ generic TLDs
  - Initially .com, .edu , .gov., .mil, .org, .net
  - Unrestricted (.com) vs Restricted (.edu)
  - Added regions (.asia, .kiwi), Brands (.apple), Sponsored (.aero) in 2012
- ~250 country code TLDs
  - Two letters, e.g., “.au”, plus international characters since 2010
  - Widely commercialized, e.g., .tv (Tuvalu)
  - Many domain hacks, e.g., instagr.am (Armenia), kurti.sh (St. Helena)

https://publicsuffix.org

ecompute v1.1

PUBLIC SUFFIX LIST

LEARN MORE | THE LIST | SUBMIT AMENDMENTS

A "public suffix" is one under which Internet users can (or historically could) directly register names. Some examples of public suffixes are `.com`, `.co.uk` and `pvt.k12.ma.us`. The Public Suffix List is a list of all known public suffixes.

The Public Suffix List is an initiative of Mozilla, but is maintained as a community resource. It is available for use in any software, but was originally created to meet the needs of browser manufacturers. It allows browsers to, for example:

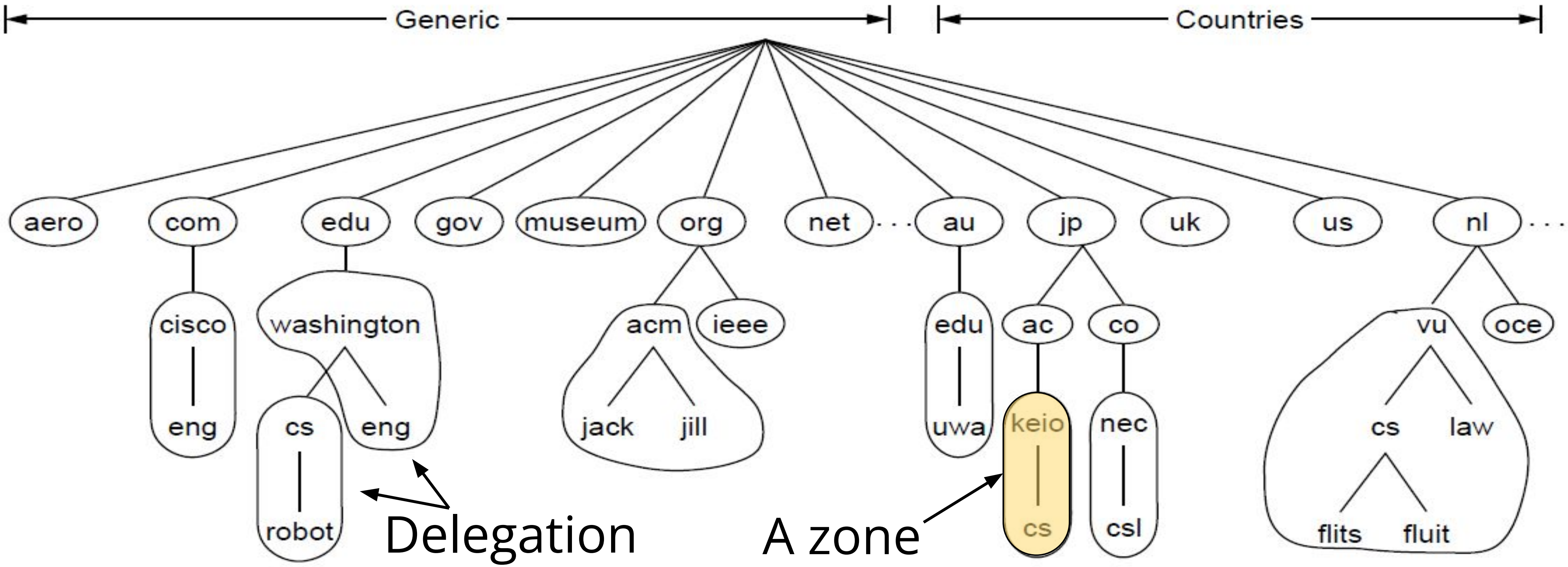
- Avoid privacy-damaging "supercookies" being set for high-level domain name suffixes
- Highlight the most important part of a domain name in the user interface
- Accurately sort history entries by site

We maintain a fuller (although not exhaustive) list of what people are using it for. If you are using it for something else, you are encouraged to tell us, because it helps us to assess the potential impact of changes. For that, you can use the psl-discuss mailing list, where we consider issues related to the maintenance, format and semantics of the list. Note: please do not use this mailing list to request amendments to the PSL's data.

It is in the interest of Internet registries to see that their section of the list is up to date. If it is not, their customers may have trouble setting cookies, or data about their sites may display sub-optimally. So we encourage them to maintain their section of the list by submitting amendments.

DNS Zones

- A <u>zone</u> is a contiguous portion of the namespace



DNS Zones (2)

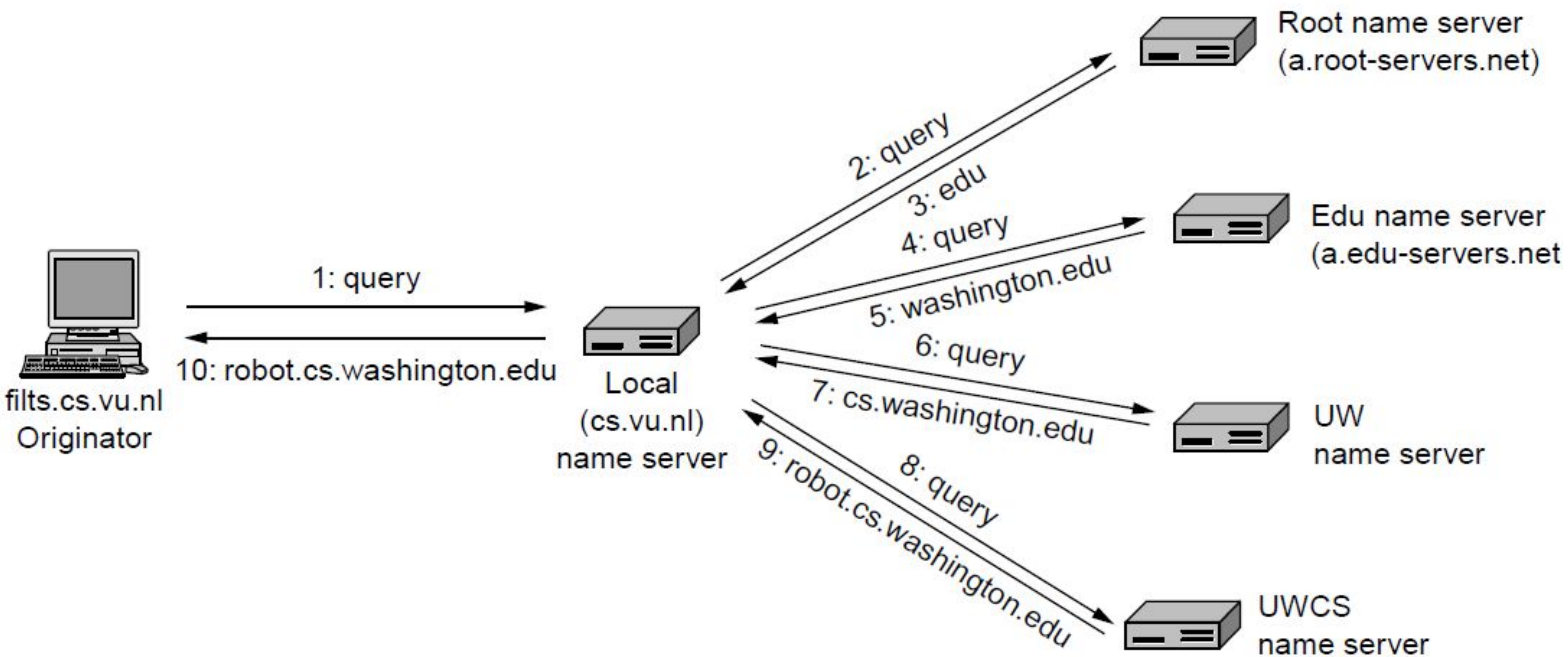
- Zones are the basis for distribution
  - EDU Registrar administers .edu
  - UW administers washington.edu
  - CSE administers cs.washington.edu
- Each zone has a <u>nameserver</u> to contact for information about it
  - Zone must include contacts for delegations, e.g., .edu knows nameserver for washington.edu

DNS Resolution

- DNS protocol lets a host resolve any host name (domain) to IP address
- If unknown, can start with the root nameserver and work down zones
- Let's see an example first …

DNS Resolution (2)

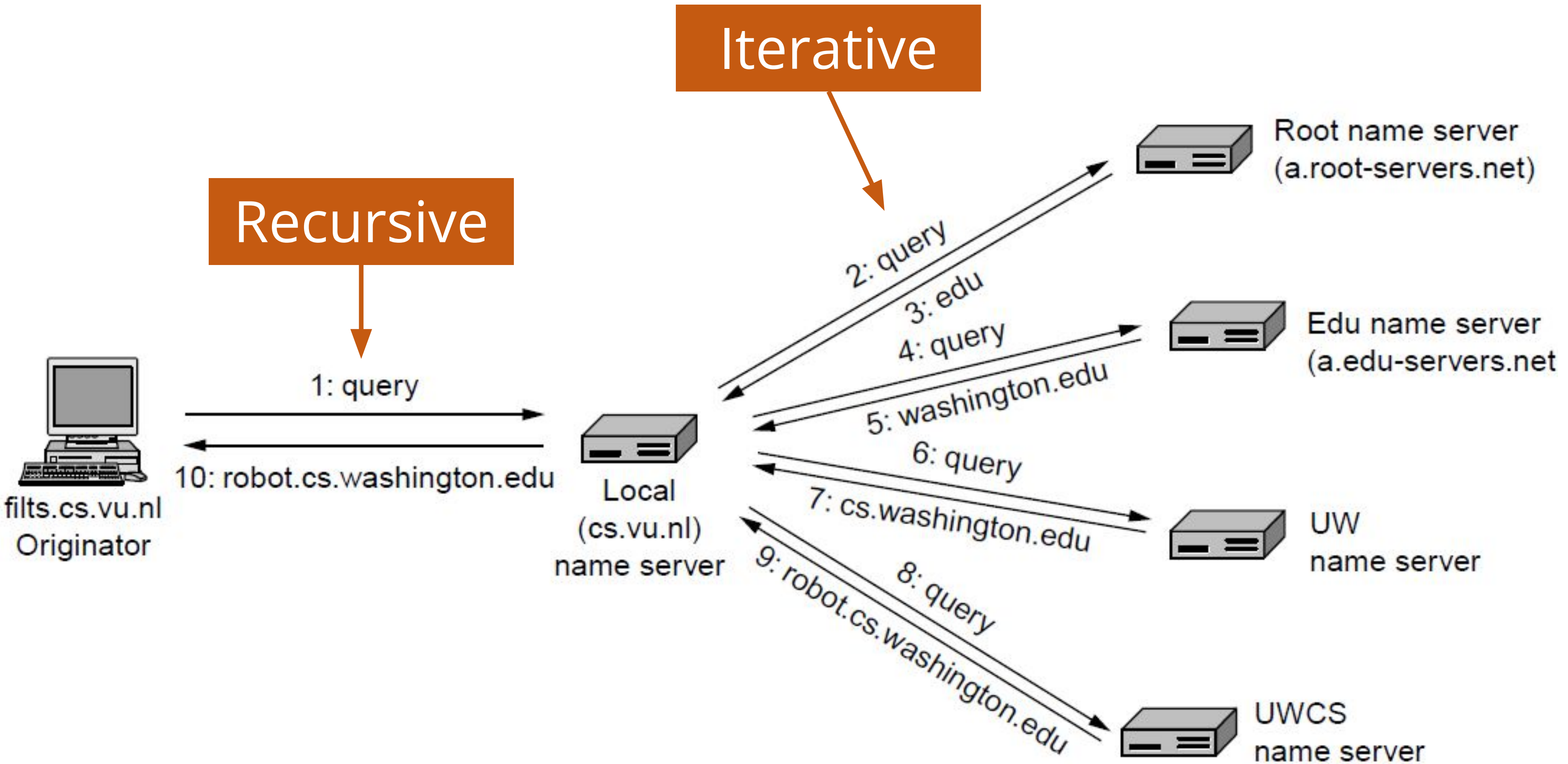
- flits.cs.vu.nl resolves robot.cs.washington.edu



Iterative vs. Recursive Queries

- Recursive query
  - Nameserver resolves and returns final answer
  - E.g., flits → local nameserver
- Iterative (Authoritative) query
  - Nameserver returns answer or who to contact for answer
  - E.g., local nameserver → all others

Iterative vs. Recursive Queries (2)



Iterative vs. Recursive Queries (3)

- Recursive query
  - Lets server offload client burden (simple resolver) for manageability
  - Lets server cache results for a pool of clients
- Iterative query
  - Lets server "file and forget"
  - Easy to build high load servers

Local Nameservers

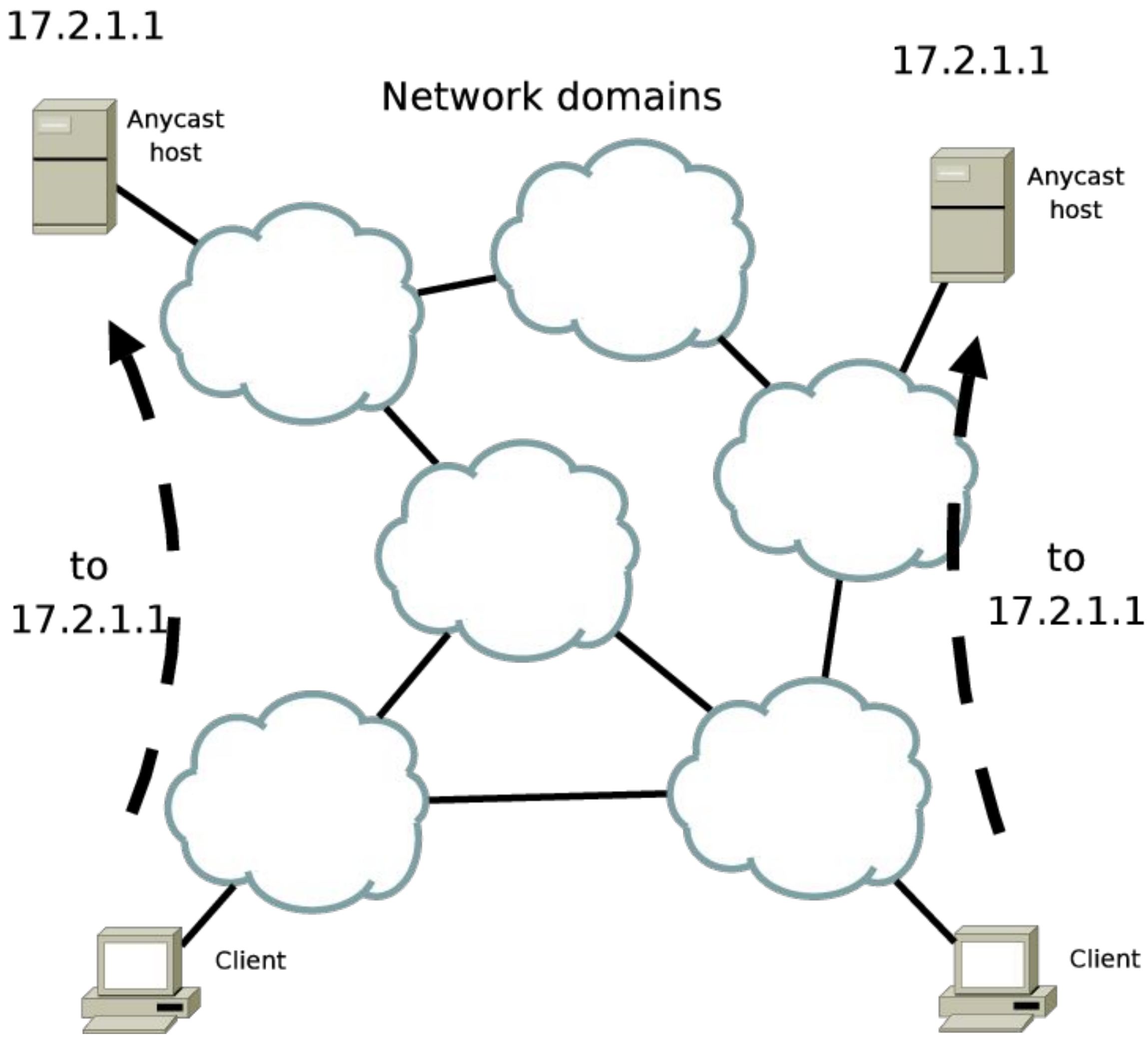
- Local nameservers often run by IT (enterprise, ISP)
  - But may be your host or AP
  - Or centralized alternatives e.g., Google public DNS (8.8.8.8) Cloudflare's public DNS (1.1.1.1)
- Clients need to be able to contact local nameservers
  - Typically configured via DHCP

Root Nameservers

- Root (dot) is served by 13 server names
  - a.root-servers.net to m.root-servers.net
  - All nameservers need root IP addresses
  - Handled via configuration file (named.ca)
- There are >250 distributed server instances
  - Highly reachable, reliable service
  - Most servers are reached by <u>IP anycast</u>
    - (Multiple locations advertise same IP! Routes take client to the closest one.)
  - Servers are IPv4 and IPv6 reachable

Brief aside: anycast

- Core idea: what if an IP address was ***shared*** across multiple machines?
- Mechanism: advertise the same AS ***at different locations*** *(e.g., IXPs)*
- Useful tool (hack?) for services distributed across the network

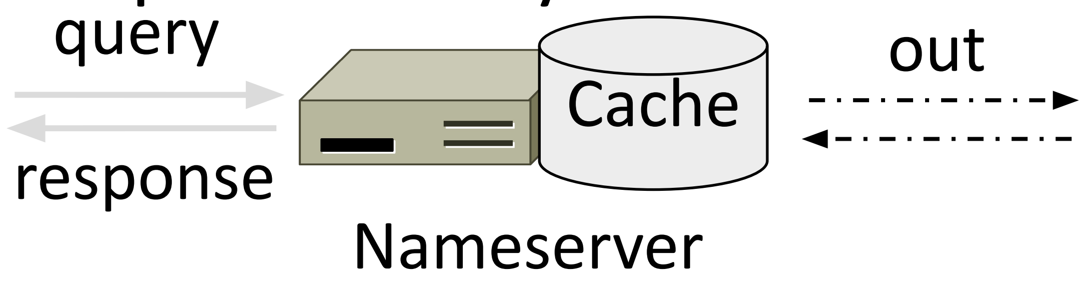


Demo:

- You can check out https://root-servers.org for info!

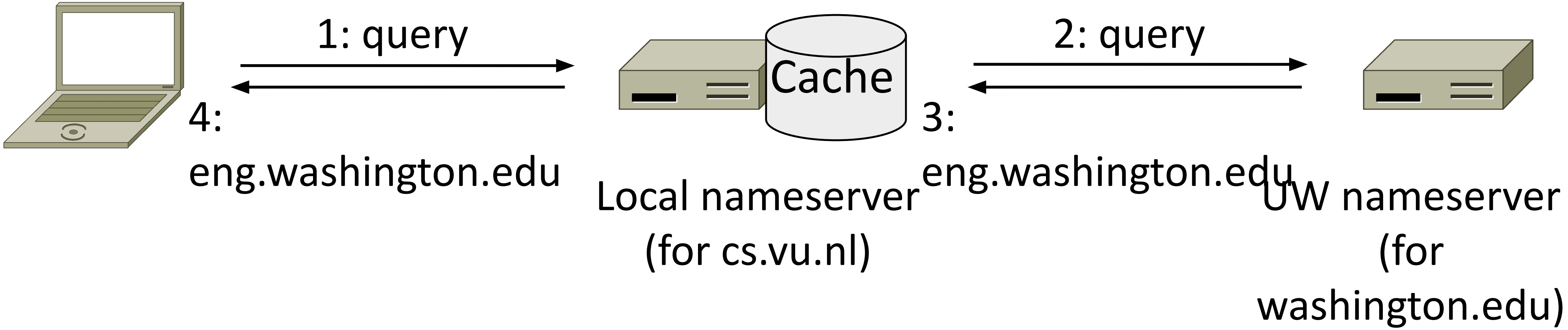
Caching

- Resolution latency needs to be low
- URLs don't have much churn
- Cache query/responses to answer future queries immediately
  - Including partial (iterative) answers
  - Responses carry a TTL for caching



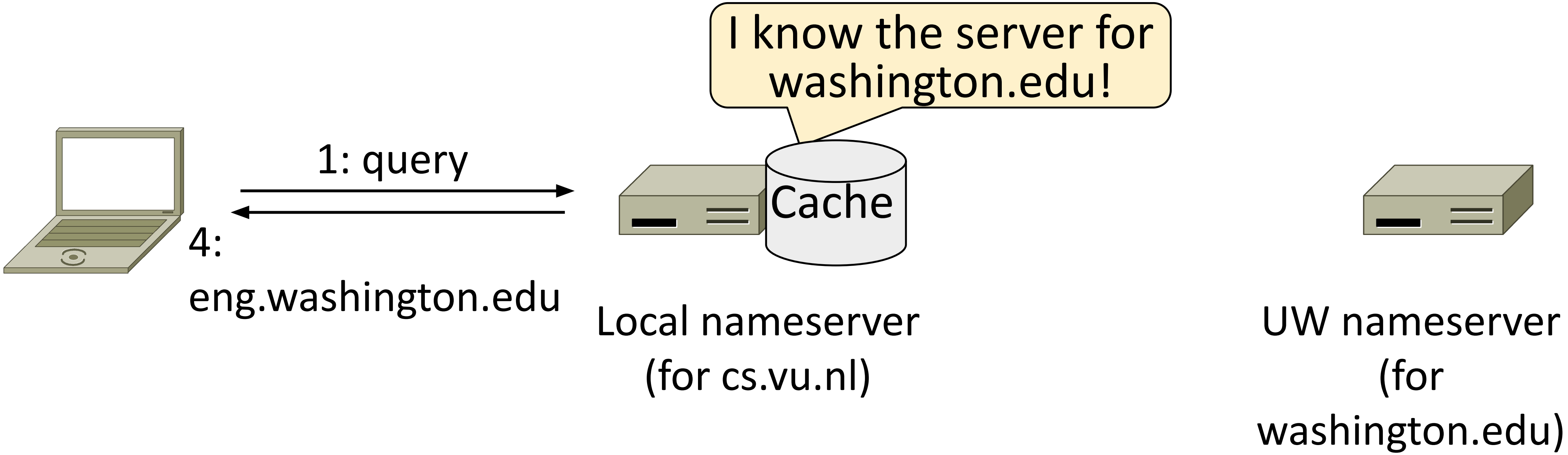
Caching (2)

- flits.cs.vu.nl looks up and stores eng.washington.edu



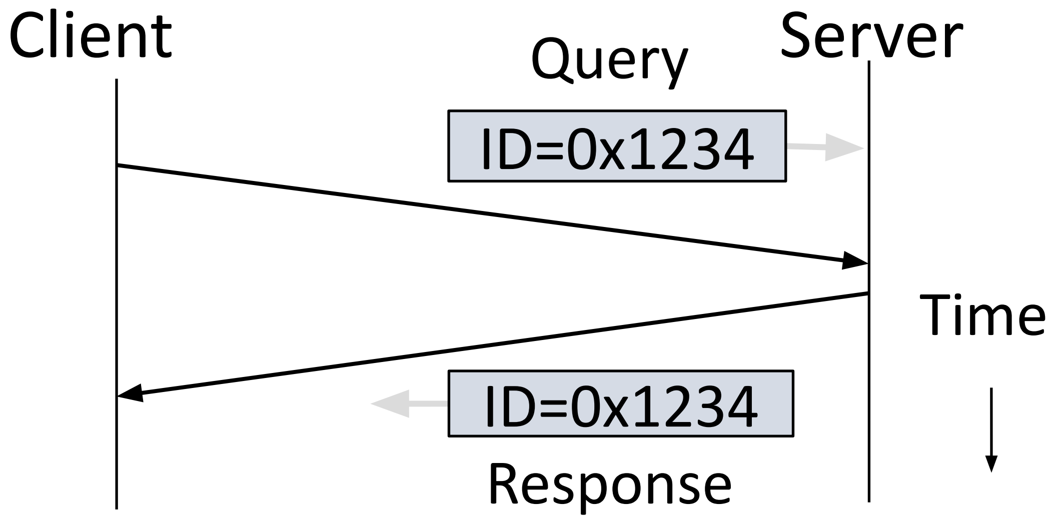
Caching (3)

- flits.cs.vu.nl now directly resolves eng.washington.edu



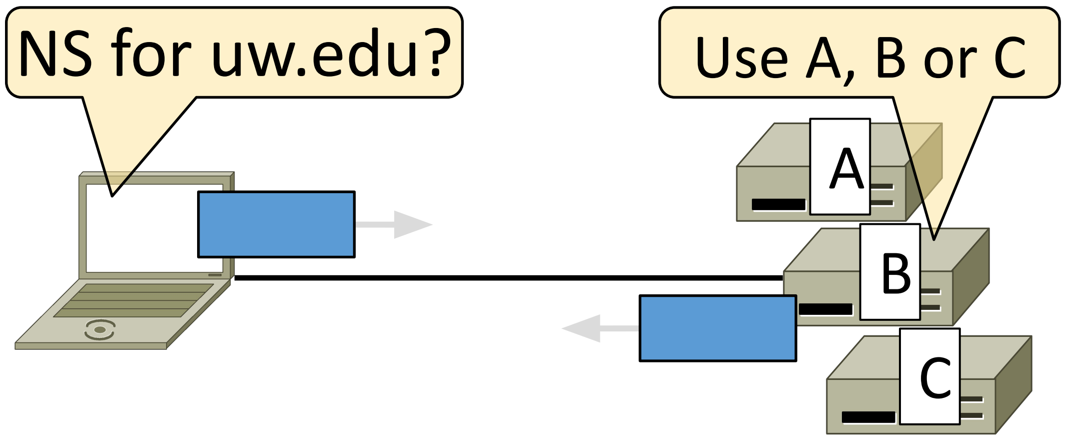
DNS Protocol

- Query and response messages
  - Built on UDP messages, port 53
  - ARQ for reliability; server is stateless!
  - Messages linked by a 16-bit ID field



DNS Protocol (2)

- Service reliability via replicas
  - Run multiple nameservers for domain
  - Return the list; clients use one answer
  - Helps distribute load too



DNS Resource Records

- A zone is comprised of DNS resource records that give information for its domain names

Type	Meaning
SOA	Start of authority, has key zone parameters
A	IPv4 address of a host
AAAA (“quad A”)	IPv6 address of a host
CNAME	Canonical name for an alias
MX	Mail exchanger for the domain
NS	Nameserver of domain or delegated subdomain

DNS Resource Records (2)

```
; Authoritative data for cs.vu.nl
cs.vu.nl.       86400  IN  SOA    star boss (9527,7200,7200,241920,86400)
cs.vu.nl.       86400  IN  MX     1 zephyr
cs.vu.nl.       86400  IN  MX     2 top
cs.vu.nl.       86400  IN  NS     star

star            86400  IN  A      130.37.56.205
zephyr          86400  IN  A      130.37.20.10
top             86400  IN  A      130.37.20.11
www             86400  IN  CNAME  star.cs.vu.nl
ftp             86400  IN  CNAME  zephyr.cs.vu.nl

flits           86400  IN  A      130.37.16.112
flits           86400  IN  A      192.31.231.165
flits           86400  IN  MX     1 flits
flits           86400  IN  MX     2 zephyr
flits           86400  IN  MX     3 top

rowboat                IN  A      130.37.56.201
                       IN  MX     1 rowboat
                       IN  MX     2 zephyr

little-sister          IN  A      130.37.62.23

laserjet               IN  A      192.31.231.216
```

Start of Authority

Name server

IP addresses
of computers

Mail gateways

Address Mappings

- What types of mappings are allowed in the DNS?
  - The Internet is practical… what is needed?
  - What do you all think?

- Single hostname to single address?
- Multiple hostnames to a single addresses?
- Single hostname to multiple addresses?

Address Mappings

- What types of mappings are allowed in the DNS?
  - The Internet is practical… what is needed?
  - What do you all think?

- Single hostname to single address?
- Multiple hostnames to a single addresses?
- Single hostname to multiple addresses?

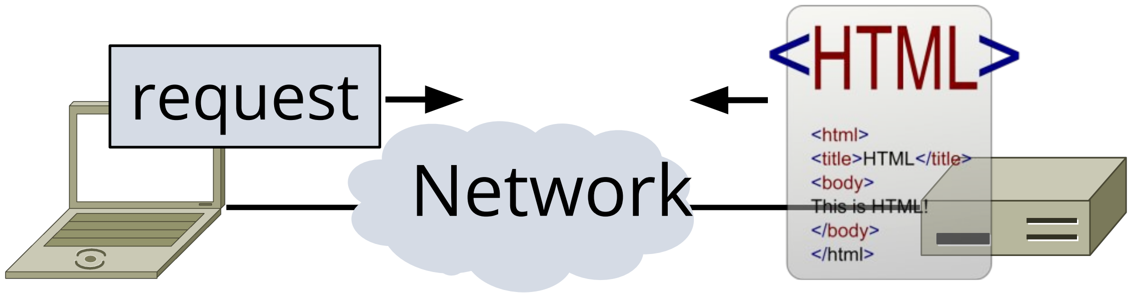
- Answer: All of the above!

DIG DEMO

HTTP

HTTP, (HyperText Transfer Protocol)

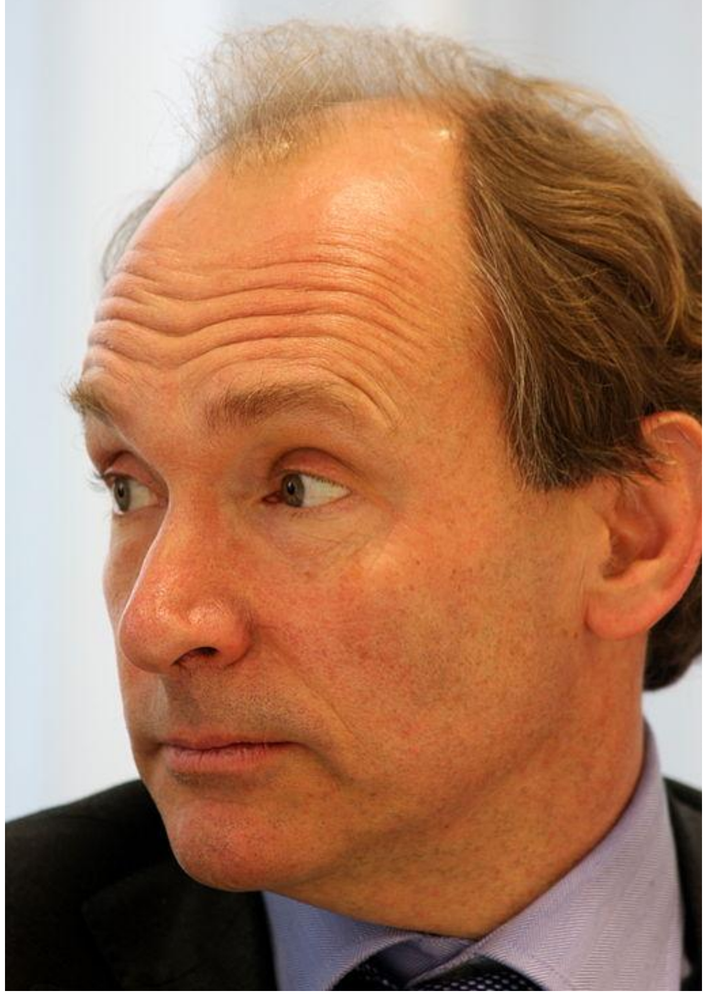
- Basis for fetching Web pages
- Now used for many non-web applications too (“App APIs”)



Sir Tim Berners-Lee (1955–)

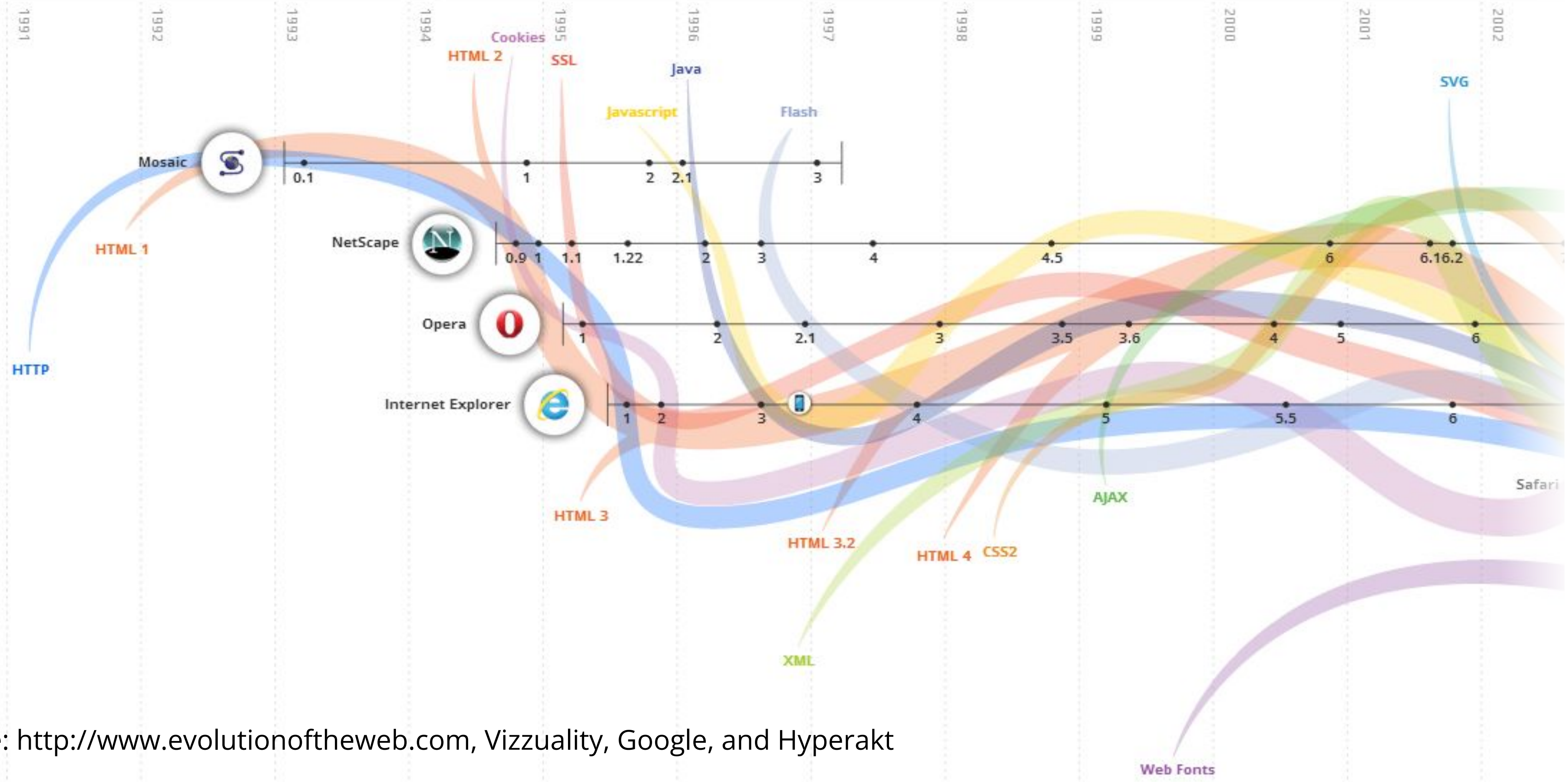
- Inventor of the Web
  - Dominant Internet app since mid 90s
  - He now directs the W3C

- Developed Web at CERN in ‘89
  - Browser, server and first HTTP
  - Popularized via Mosaic (‘93), Netscape
  - First WWW conference in ’94 …



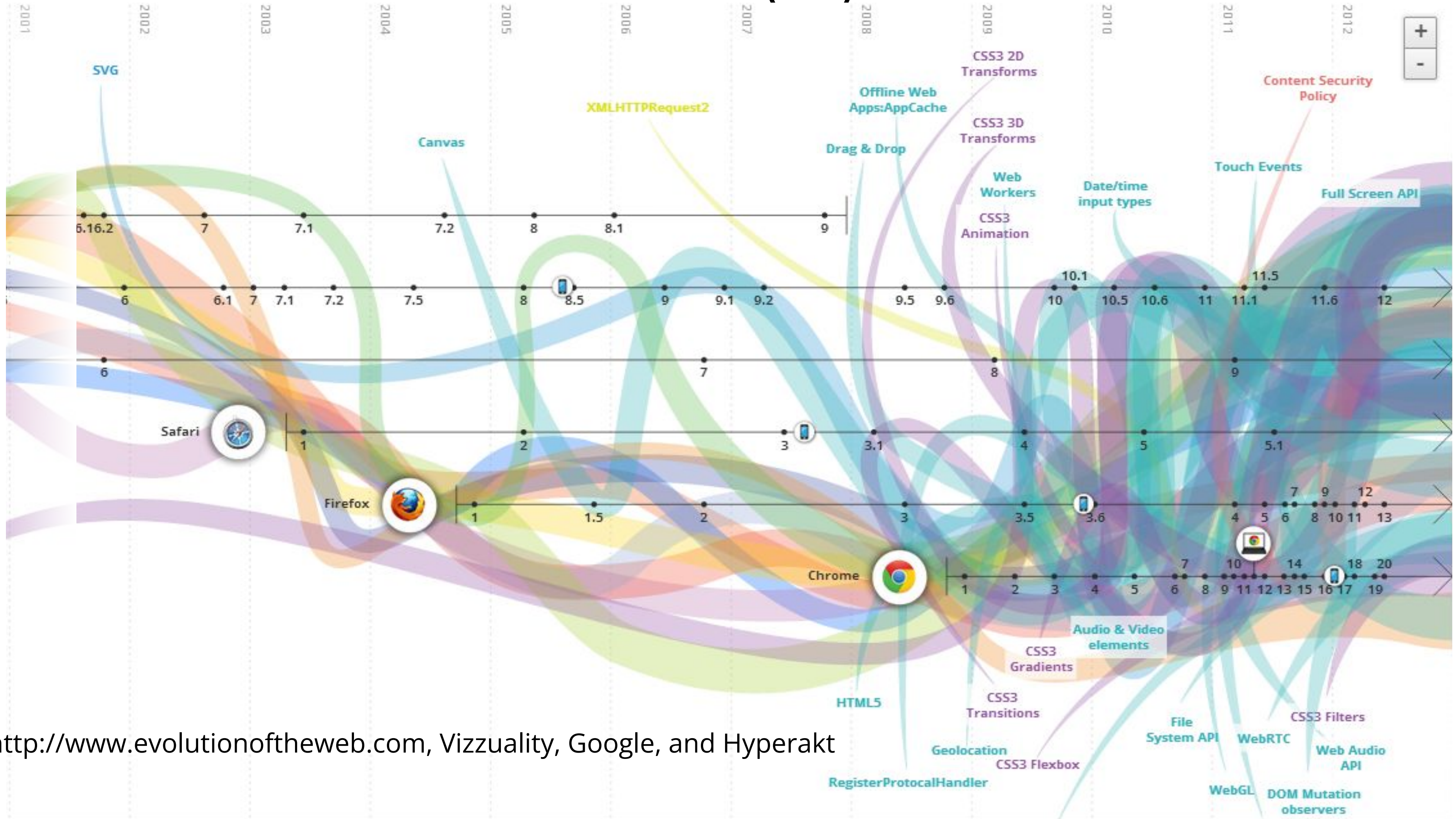
Source: By Paul Clarke, CC-BY-2.0, via Wikimedia Commons

Evolution of the Web



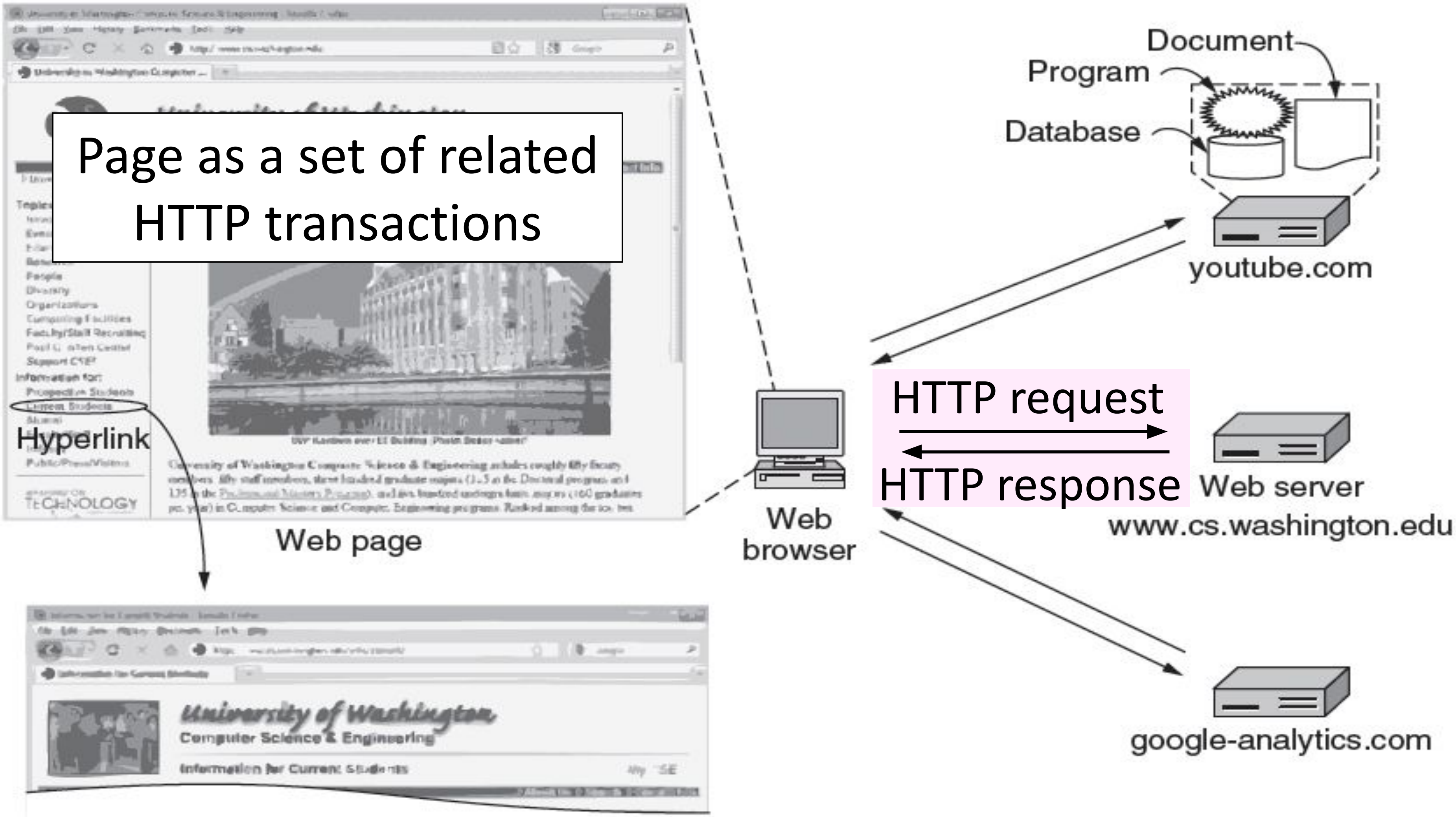
Source: http://www.evolutionoftheweb.com, Vizzuality, Google, and Hyperakt

Evolution of the Web (2)



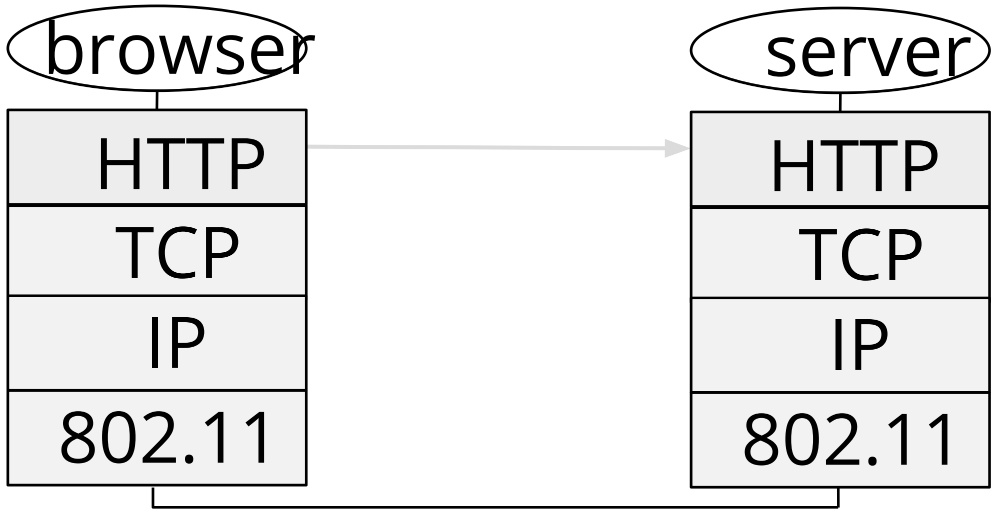
Source: http://www.evolutionoftheweb.com, Vizzuality, Google, and Hyperakt

Web Context


Page as a set of related HTTP transactions
Hyperlink
Web page
Web browser
HTTP request
HTTP response
Document
Program
Database
youtube.com
Web server
www.cs.washington.edu
google-analytics.com
University of Washington
Computer Science & Engineering
Information for Current Students

Web Protocol Context

- HTTP is a request/response protocol for fetching Web resources
  - HTTP/1 and HTTP/2 run on TCP
  - Typically port 80 (plain http) or 443 (https-- later)
- Part of browser/server or app



Fetching a Web page with HTTP

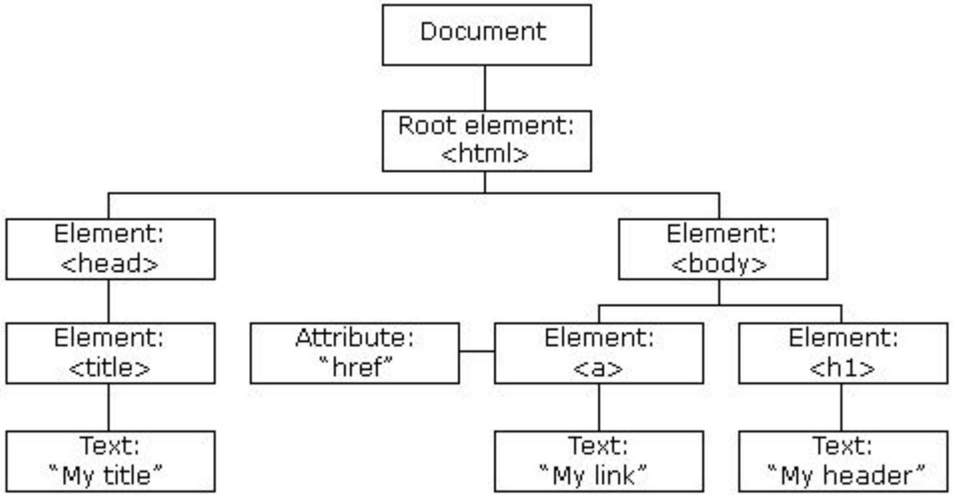
- Start with the page URL (Uniform Resource Locator):
  - $\underbrace{\text{http://}}_{\text{Protocol}}\underbrace{\text{en.wikipedia.org}}_{\text{Server}}\underbrace{\text{/wiki/Vegemite}}_{\text{Page on server}}$
- Steps:
  - 🏠 Resolve the server to IP address (DNS)
  - ⛓️Set up TCP connection to the server
  - ✋Send HTTP request for the page
    - ⏱️ (Await HTTP response for the page)
  - ✌️Execute/fetch embedded resources/render
  - 🧹Clean up any idle TCP connections

HTML

- Hypertext Markup Language (HTML)
  - Describes how web content should be handled and rendered
  - Key innovation was the “hyperlink”, an HTML element linking to other HTML elements using URLs
  - Also includes Cascading Style Sheets (CSS) for maintaining look-and-feel across a domain
  - Specific standards have been the subject of many “browser wars”

DOM (Document Object Model)

- Base primitive for web browsers interacting with HTML
- Use HTML (XML) to create a tree of elements
- Javascript code is embedded in the page and modifies the DOM based on:
    - User actions
    - Asynchronous Javascript
    - Other server-side actions



DOM Example

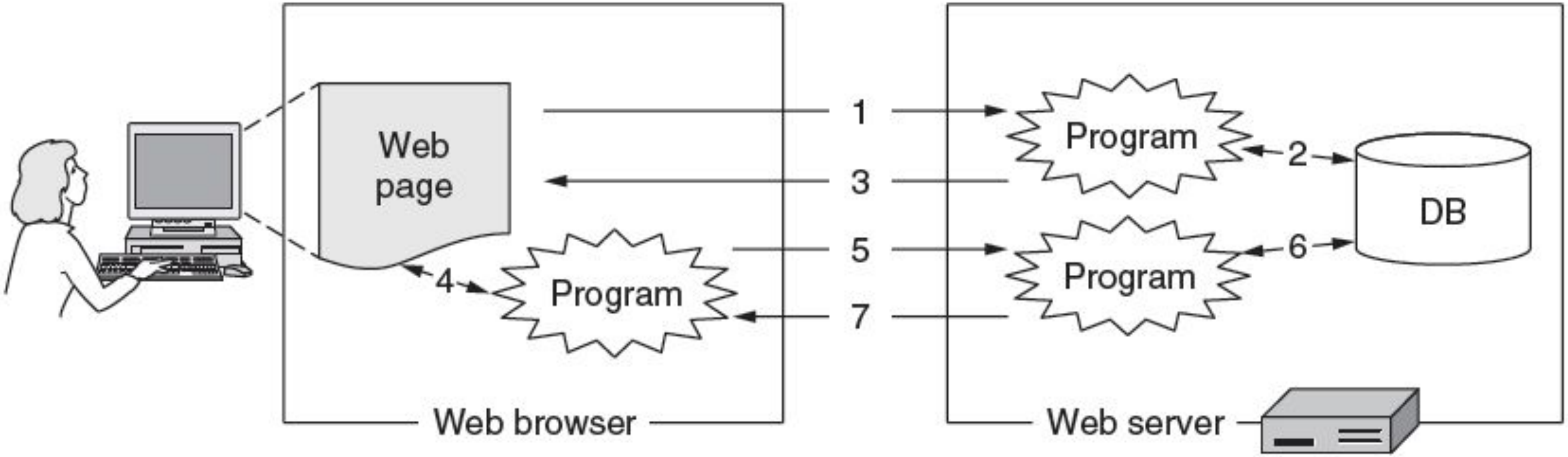
```html
<!DOCTYPE html>
<html>
<head>
  <link rel="stylesheet" href="styles.css">
</head>
<body>

<h1>This is a heading</h1>
<p>This is a paragraph.</p>

</body>
</html>
```

DOM Examples

- Go to browser and show DOM

Static vs Dynamic Web pages

- Static is just static files, e.g., image
- Dynamic has ongoing computation of some kind
  - e.g., Javascript on client, Java/Go/PHP/Rust/JS/etc… on server



HTTP Protocol

- Originally a simple protocol
  - ***But with many options added over time*** 😅
  - Text-based commands, headers
- Try it yourself:
  - As a “browser” fetching a URL
  - Run “telnet stackoverflow 80”
  - Type “GET /questions HTTP/1.0
    Host: stackoverflow.com” to server followed by a blank line
  - Server will return HTTP response with the page contents
    - (or other info)

HTTP Protocol (2)

- Commands used in the request

Fetch page

Upload data

Method	Description
GET	Read a Web page
HEAD	Read a Web page's header
POST	Append to a Web page
PUT	Store a Web page
DELETE	Remove the Web page
TRACE	Echo the incoming request
CONNECT	Connect through a proxy
OPTIONS	Query options for a page

Basically defunct

HTTP Protocol (3)

- Codes returned with the response

Yes! →

Code	Meaning	Examples
1xx	Information	100 = server agrees to handle client's request
2xx	Success	200 = request succeeded; 204 = no content present
3xx	Redirection	301 = page moved; 304 = cached page still valid
4xx	Client error	403 = forbidden page; 404 = page not found
5xx	Server error	500 = internal server error; 503 = try again later

ed

418 I'm a teapot

The HTTP `418 I'm a teapot` client error response code indicates that the server refuses to brew coffee because it is, permanently, a teapot. A combined coffee/tea pot that is temporarily out of coffee should instead return 503. This error is a reference to Hyper Text Coffee Pot Control Protocol defined in April Fools' jokes in 1998 and 2014.

Some websites use this response for requests they do not wish to handle, such as automated queries.

Status

```
418 I'm a teapot
```

Specifications

Specification
RFC 2324 # section-2.3.2

Browser compatibility

Report problems with this compatibility data on GitHub

arge

ns

	Chrome	Edge	Firefox	Opera	Safari	Chrome Android	Firefox for Android	Opera Android	Safari on iOS	Samsung Internet	WebView Android
418	✓ Yes	✓ 12	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✓ Yes

Tip: you can click/tap on a cell for more information.

Representational State Transfer (REST)

- Using HTTP for general network services
- An ideal for design of HTTP-based APIs
  - Called RESTful APIs
- 5 Core Tenants:
  - (1) Uniform Interface and (2) Client/Server
  - (3) Stateless (no state on server)
  - (4) Cachable (individual urls can be cached)
  - (5) Layered (no visibility under REST hood)

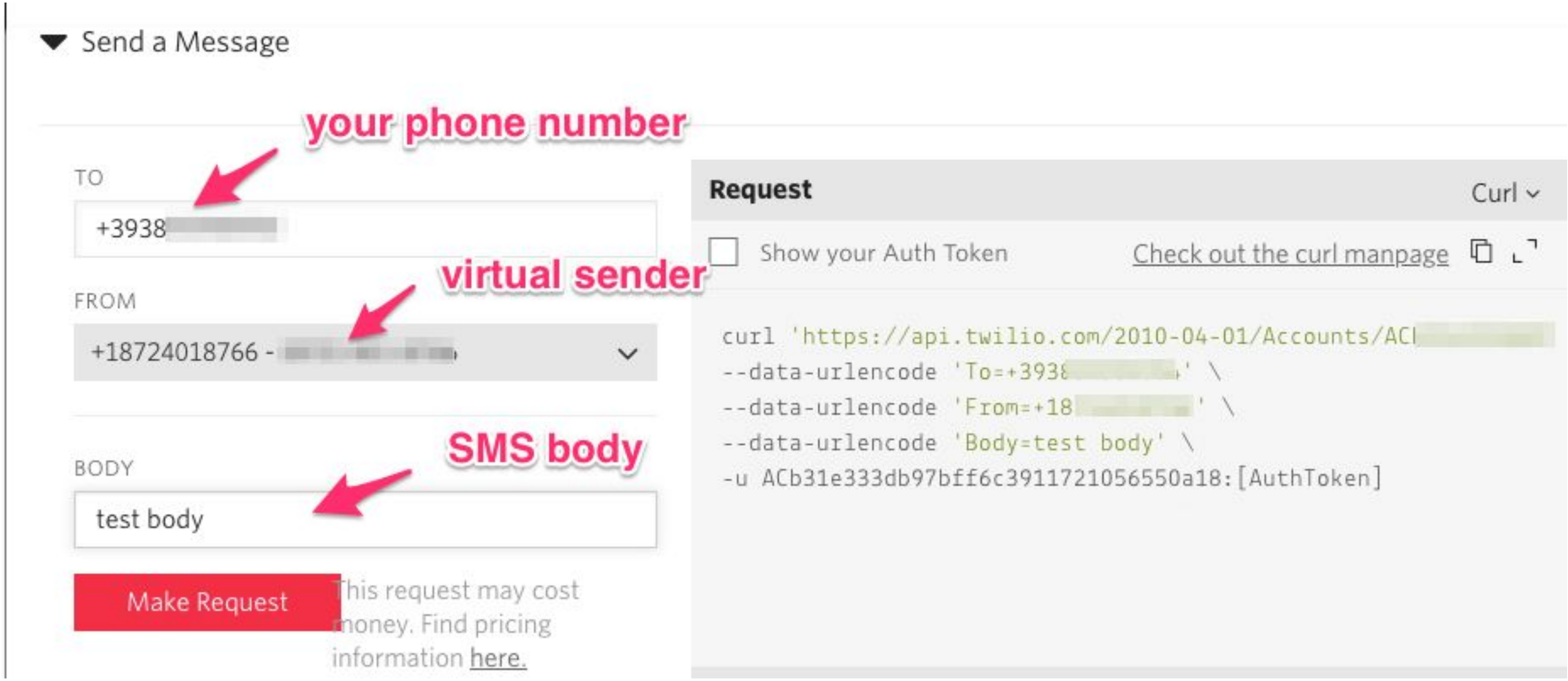
Representational State Transfer (REST)

- RESTful Interfaces use HTTP to provide a variety of other media (e.g., JSON)
  - For example, GET will always be safe and change nothing

HTTP methods

Uniform Resource Locator (URL)	GET	PUT	POST	DELETE
Collection, such as http://api.example.com/resources/	**List** the URIs and perhaps other details of the collection's members.	**Replace** the entire collection with another collection.	**Create** a new entry in the collection. The new entry's URI is assigned automatically and is usually returned by the operation.[17]	**Delete** the entire collection.
Element, such as http://api.example.com/resources/item17	**Retrieve** a representation of the addressed member of the collection, expressed in an appropriate Internet media type.	**Replace** the addressed member of the collection, or if it does not exist, **create** it.	Not generally used. Treat the addressed member as a collection in its own right and **create** a new entry within it.[17]	**Delete** the addressed member of the collection.

A REST API Example: SMS Gateway



Braid protocol

Braid builds distributed features into today's web. We are an open group in the IETF. Join us!

The **Braid Protocol** is an extension to HTTP that generalizes it from a state *transfer* to a state *synchronization* protocol.

Braid adds these features to HTTP:

1. *Versioning* to HTTP resources
2. *Subscriptions* to GET requests
3. *Patches* to Range Requests
4. *Merge-Types* to specify OT or CRDT behavior

Together, these features enable a web resource to synchronize automatically across multiple clients, servers and proxies, and support arbitrary simultaneous edits by multiple writers, under arbitrary network delays and partitions, while guaranteeing consistency using a OT, CRDT, or other algorithm.

HTTP Performance

PLT (Page Load Time)

- PLT was the key measure of web performance
  - From click until user sees page
  - Small increases in PLT decrease sales
  - Nowadays part of Search Engine Optimization (SEO) too
- PLT depends on many factors
  - Structure of page/content
  - HTTP (and TCP!) protocol
  - Network RTT and bandwidth

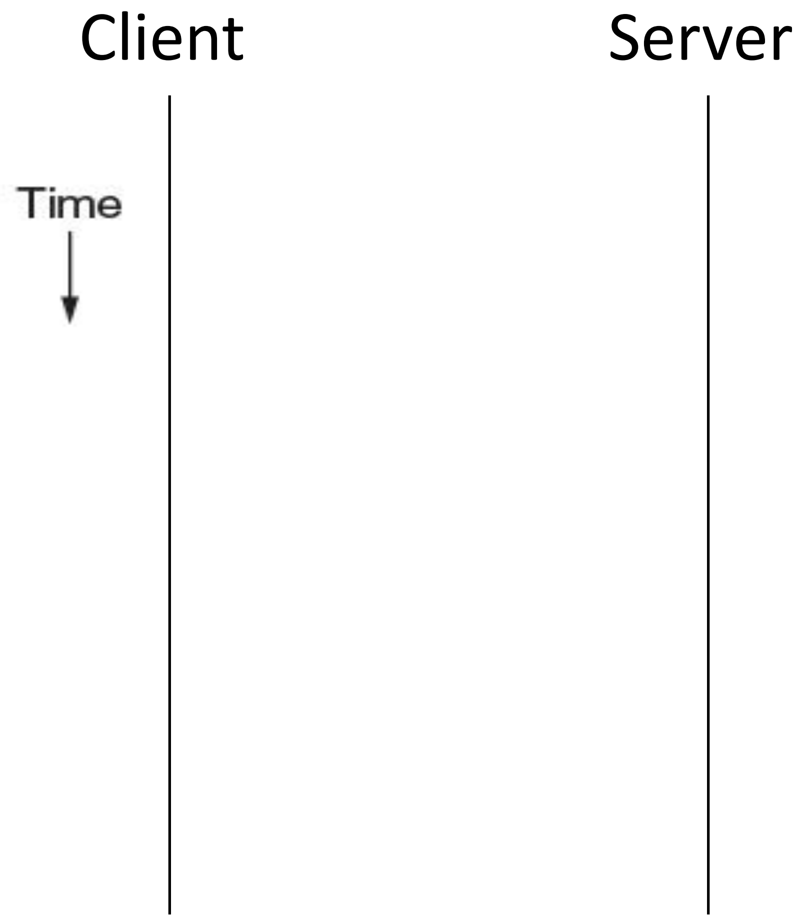
Fun Fact… Page Load Time is actually pretty tough to measure on a modern dynamic page. Now use related metrics around inspection of the rendered page itself!

Key user moments
(PLT is Dead… Long live PLT)

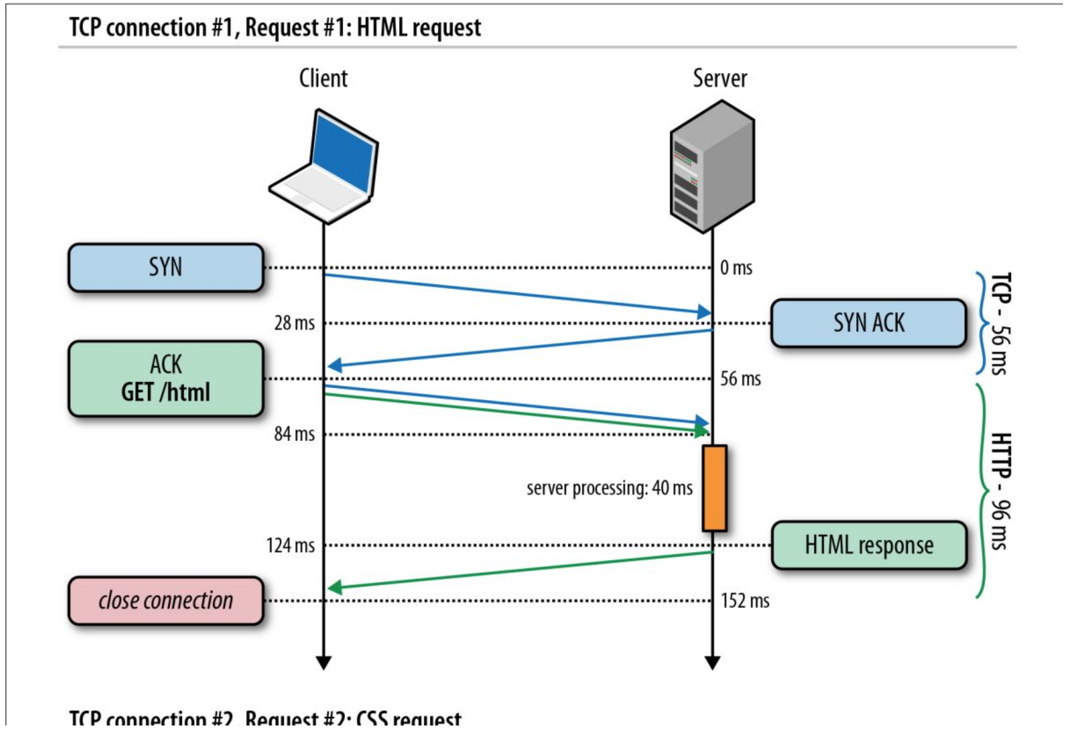
- First Contentful Paint (FCP) **"is it happening?"**
- First Meaningful Paint (FMP) **"is it useful?"**
- Time to Interactive (TTI) **"is it usable?"**

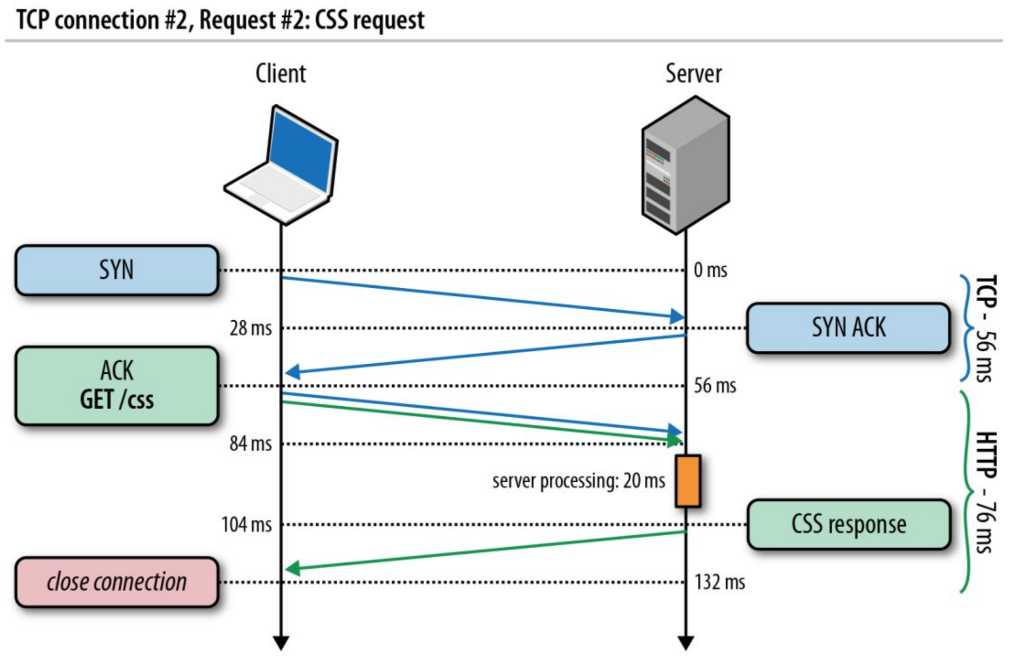
Early Performance

- HTTP/1.0 used one TCP connection to fetch one web resource
  - Made HTTP very easy to build
  - But gave fairly poor PLT …



Early Performance



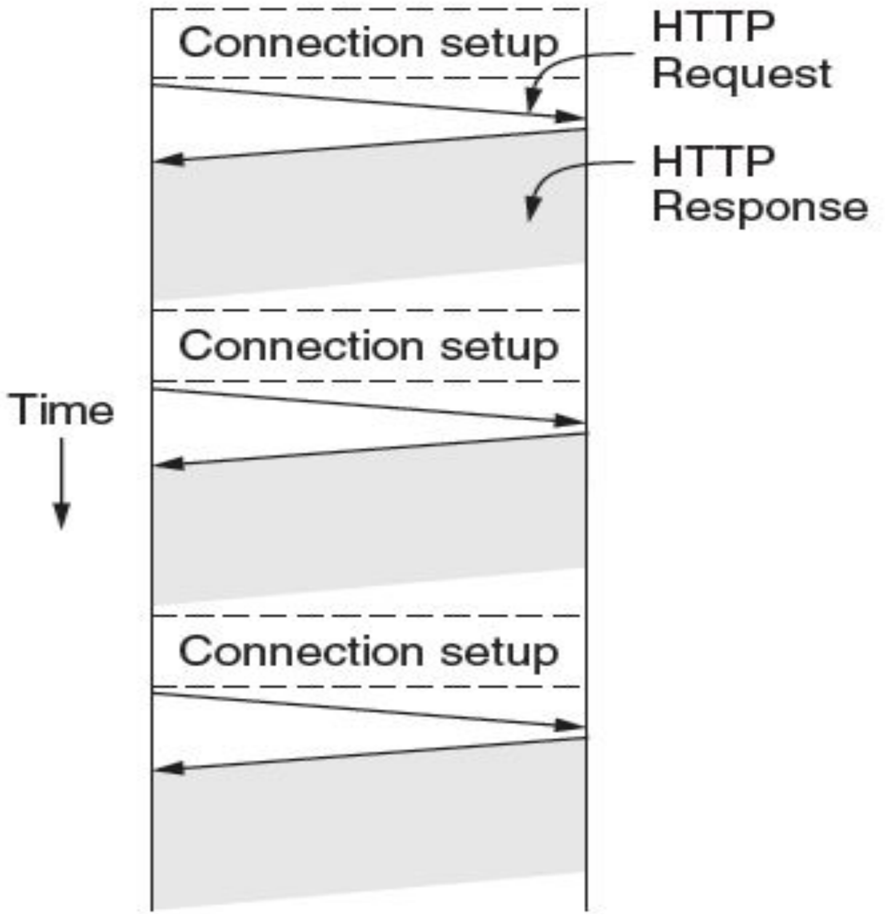


Remember: DOM Example

```html
<!DOCTYPE html>
<html>
<head>
  <link rel="stylesheet" href="styles.css">
</head>
<body>

<h1>This is a heading</h1>
<p>This is a paragraph.</p>

</body>
</html>
```

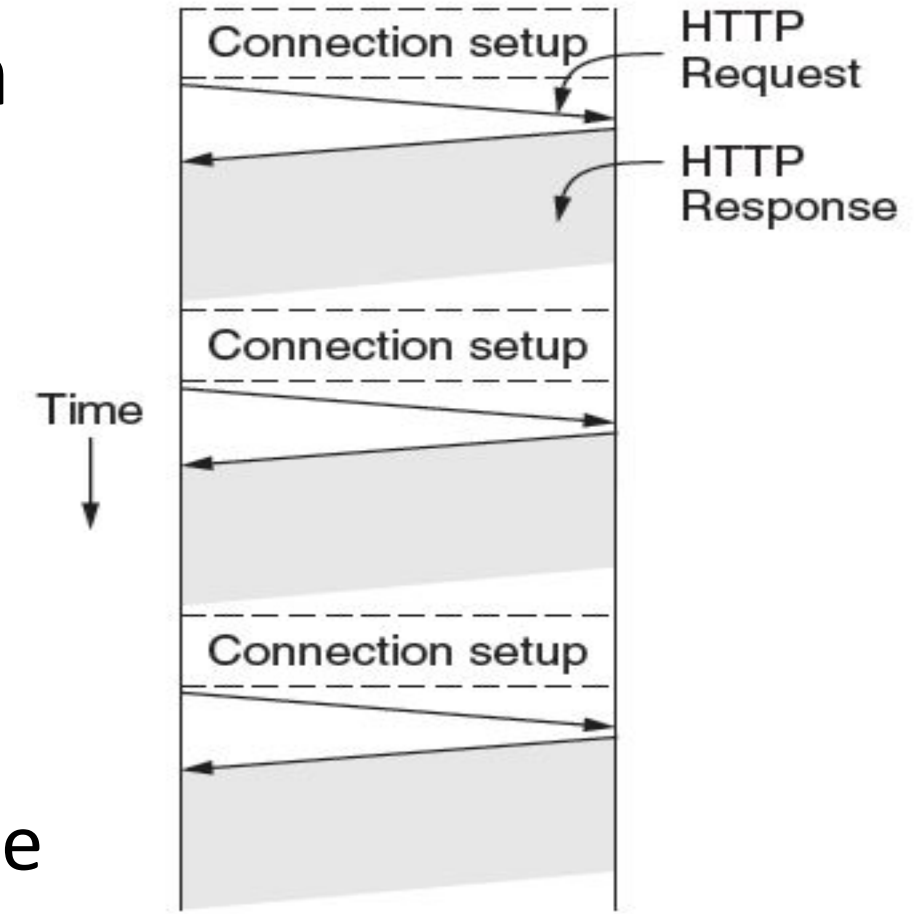
Early Performance (2)

- HTTP/1.0 used one TCP connection to fetch one web resource
  - Made HTTP very easy to build
  - But gave fairly poor PLT…

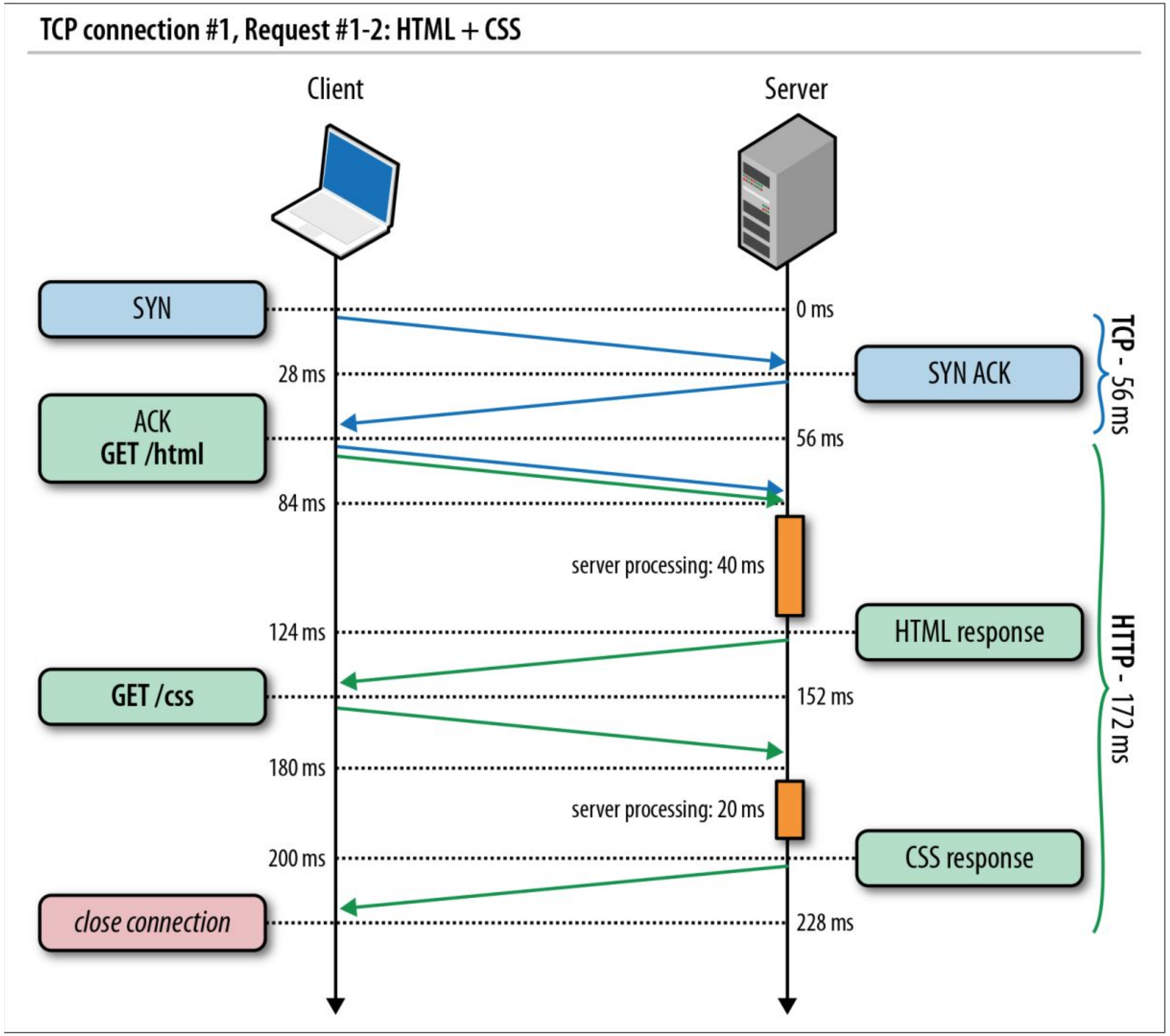


Early Performance (3)

- Many reasons why PLT is larger than necessary
  - Sequential request/responses, even when to different servers
  - Multiple TCP connection setups to the same server
  - Multiple TCP slow-start phases
- Network is not used effectively
  - Worse with many small resources / page



HTTP1


TCP connection #1, Request #1-2: HTML + CSS
Client
Server
SYN
0 ms
SYN ACK
28 ms
ACK
GET /html
56 ms
84 ms
server processing: 40 ms
HTML response
124 ms
GET /css
152 ms
180 ms
server processing: 20 ms
CSS response
200 ms
close connection
228 ms
TCP - 56 ms
HTTP - 172 ms

Ways to Decrease PLT

1. Reduce content size for transfer
   - Smaller images, gzip
2. Change HTTP to make better use of bandwidth
3. Change HTTP to avoid repeat sending of same content
   - Caching, and proxies
4. Move content closer to client
   - CDNs [later]

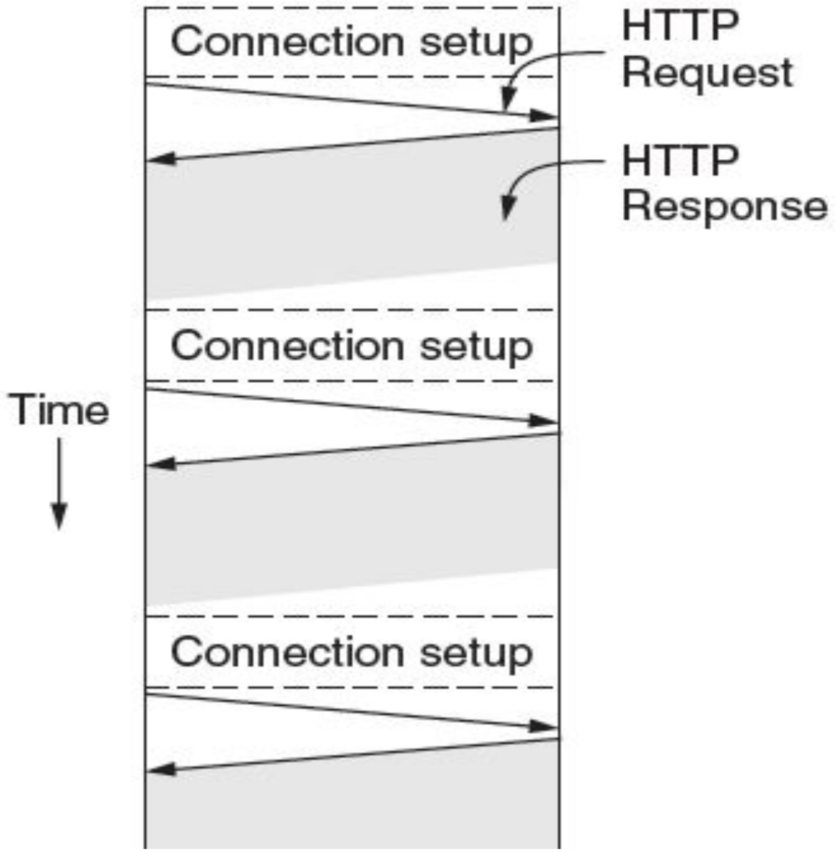
Parallel Connections

- One simple way to reduce PLT
  - Browser runs multiple (6, say) HTTP instances in parallel
  - Server is unchanged; already handled concurrent requests for many clients
- How does this help?
  - Single HTTP wasn't using network much …
  - So parallel connections aren't slowed much
  - Pulls in completion time of last fetch

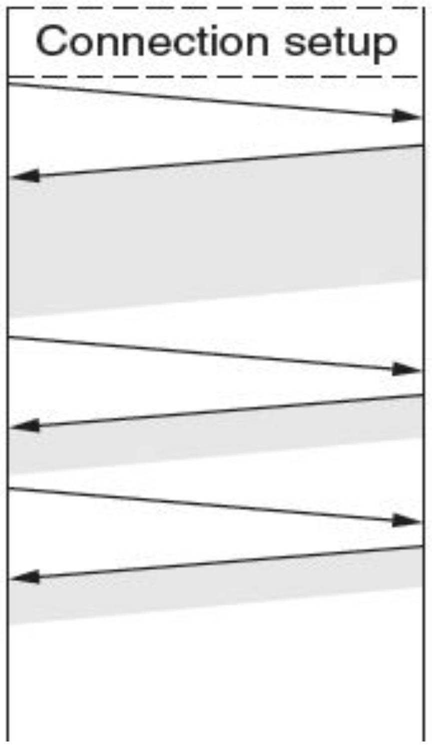
Persistent Connections

- Parallel connections compete with each other for network resources
  - 1 parallel client ≈ 6 sequential clients?
  - Exacerbates network bursts, and loss
- Persistent connection alternative
  - Make 1 TCP connection to 1 server
  - *Use it for multiple HTTP requests*

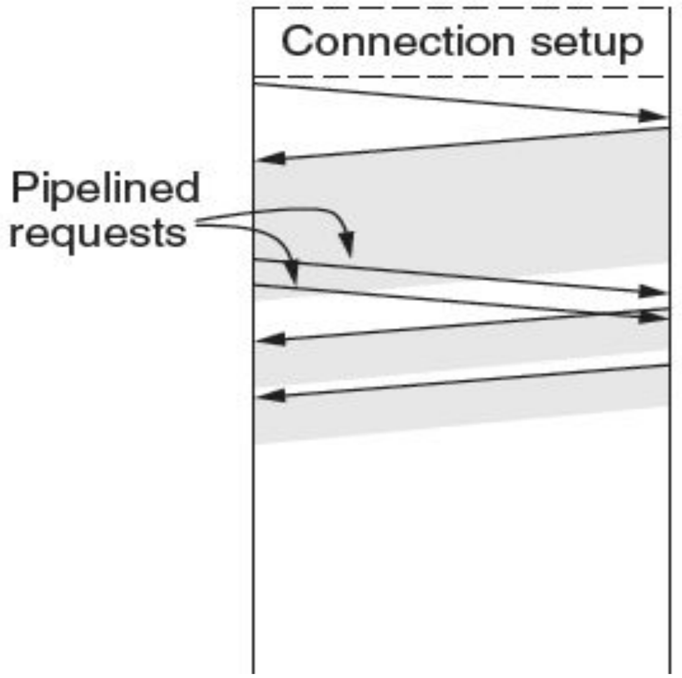
Persistent Connections (2)



One request per connection



Sequential requests per connection



Pipelined requests per connection

Persistent Connections (3)

- Widely used as part of HTTP/1.1
  - Supports optional pipelining
  - PLT benefits depending on page structure, but easy on network

HTTP Futures (Modern HTTP?)

HTTP 1.1

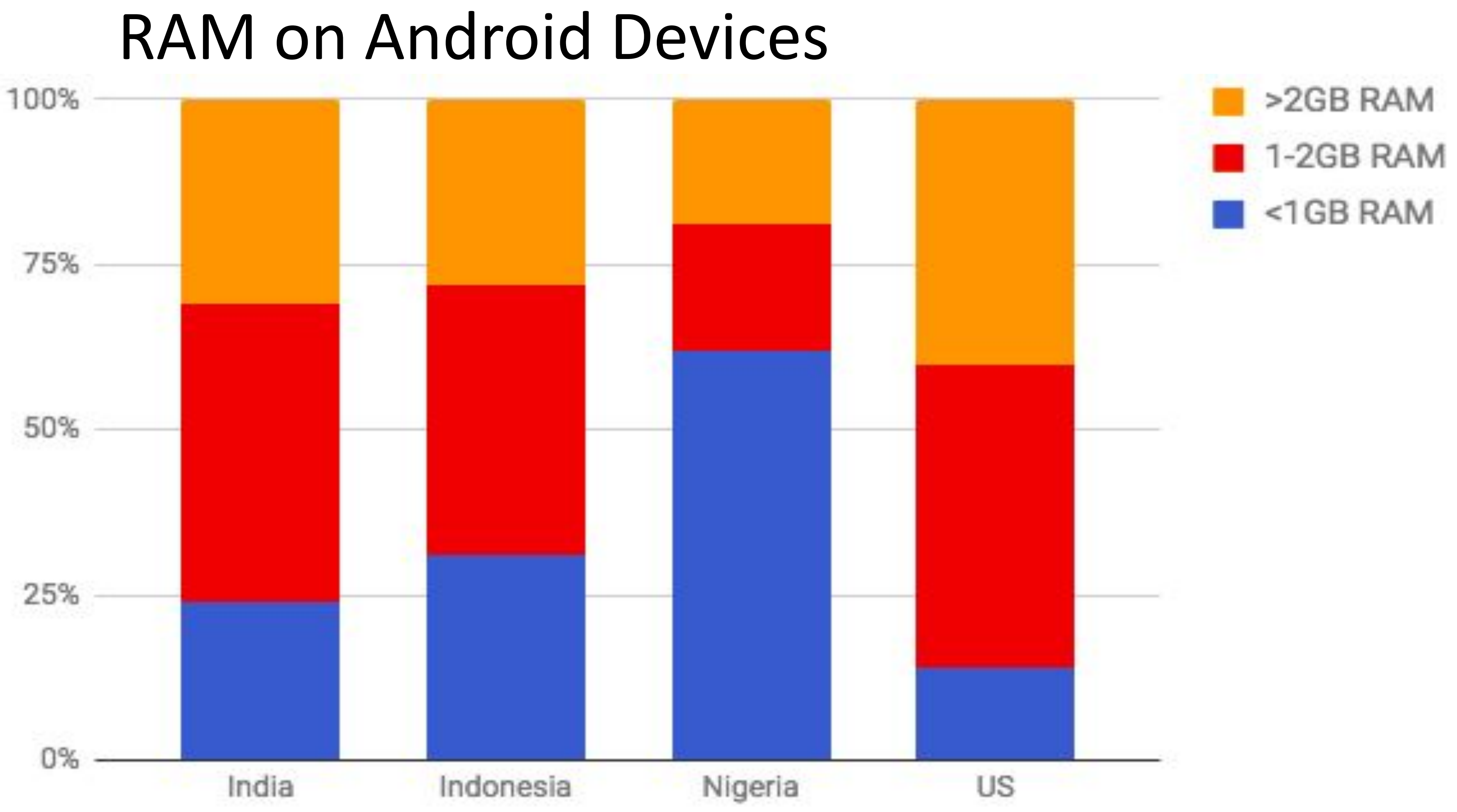
- This was it! Standard protocol until circa 2015.
- HTTP 1.1 everywhere for all web access
- Until some massive web company started noticing some trends….

Continued Growth

Country	Mobile-Only Internet Users
Egypt	70%
India	59%
South Africa	57%
Indonesia	44%
United States	25%

Thanks to Ben Greenstein @ google for slides

Continued Growth (2)



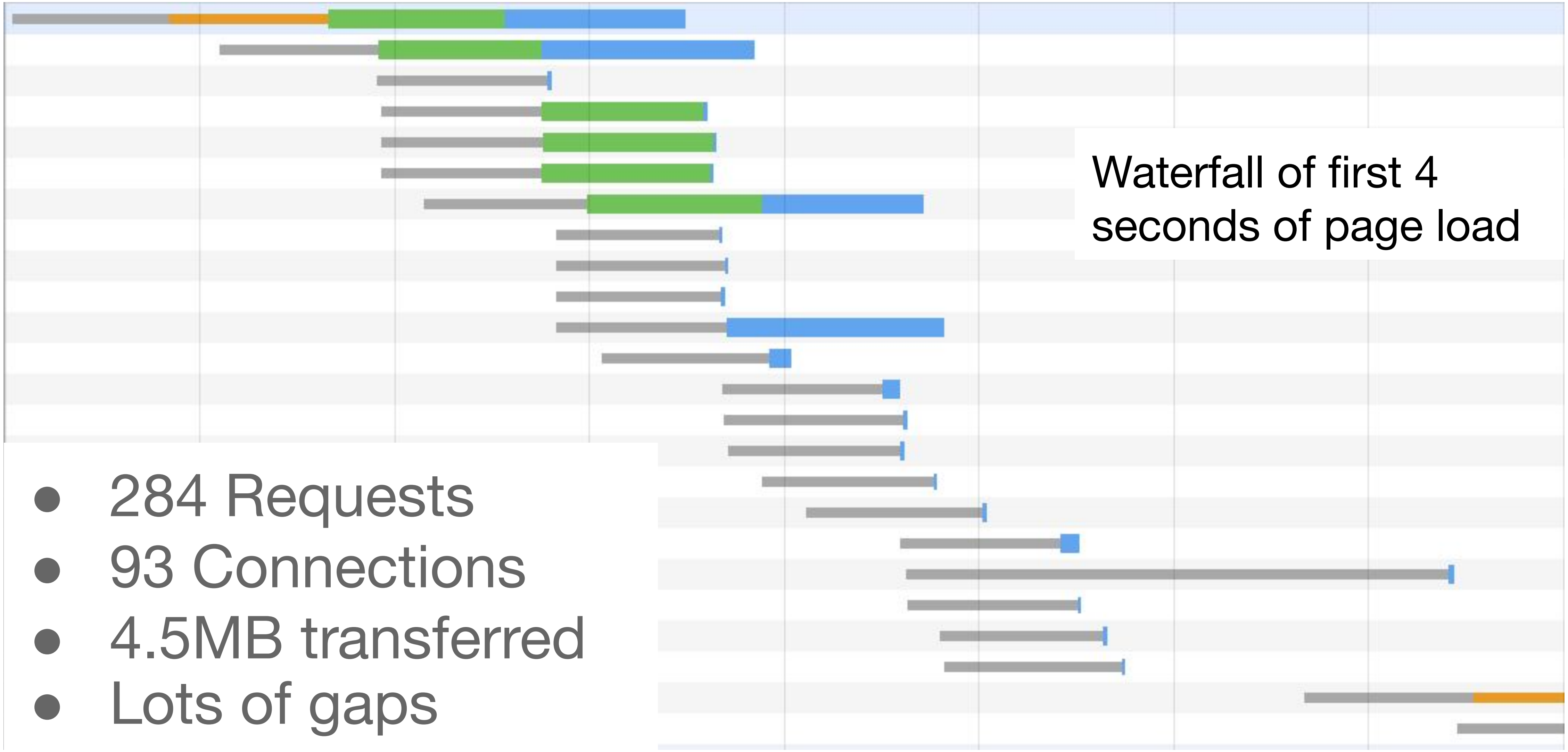
Continued Growth (3)

Tecno Y2
512MB RAM, 8GB ROM
1.3GHz dual-core Cortex-A7
2G & 3G only
4” (480x800)

Tecno W3
1GB RAM, 8GB ROM
1.3GHz dual-core Cortex-A7
2G & 3G only
5” (480x854)

Infinix Hot 4 Lite
1GB RAM, 16GB ROM
1.3GHz quad-core Cortex-A7
2G & 3G only
5.5” (720x1280)

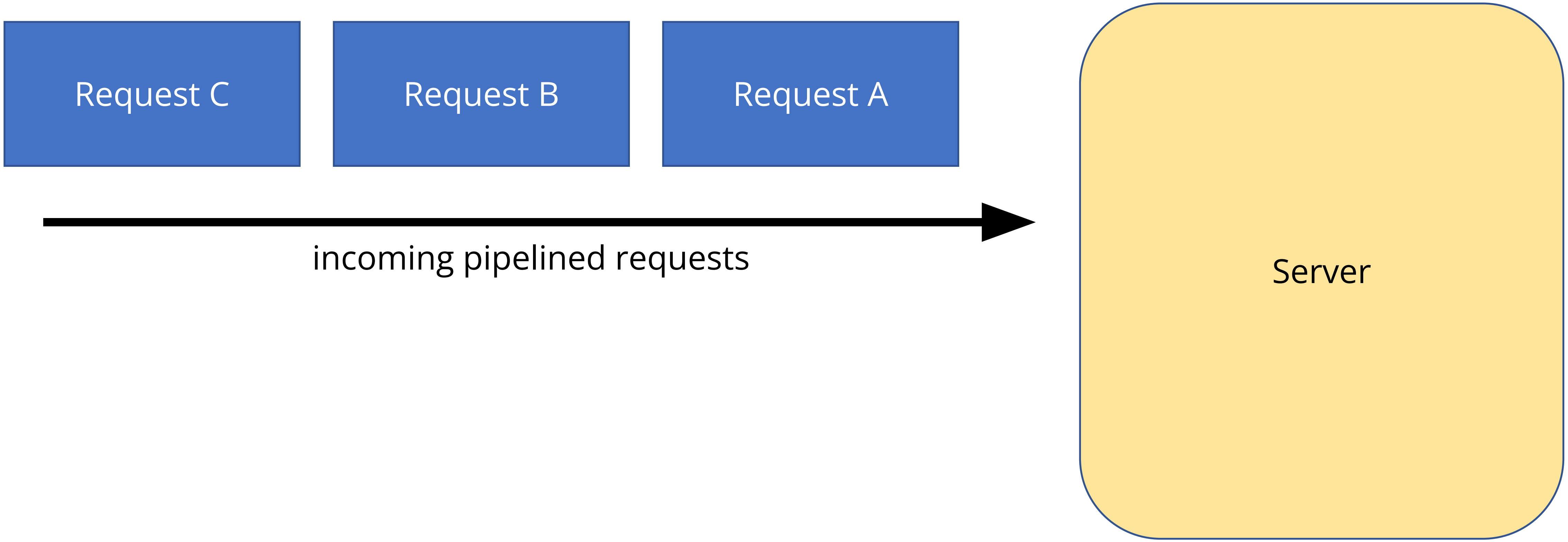
Source: Chrome logs 2019



Waterfall of first 4 seconds of page load

- 284 Requests
- 93 Connections
- 4.5MB transferred
- Lots of gaps

“head of line blocking”



HTTP Head-of-Line Blocking

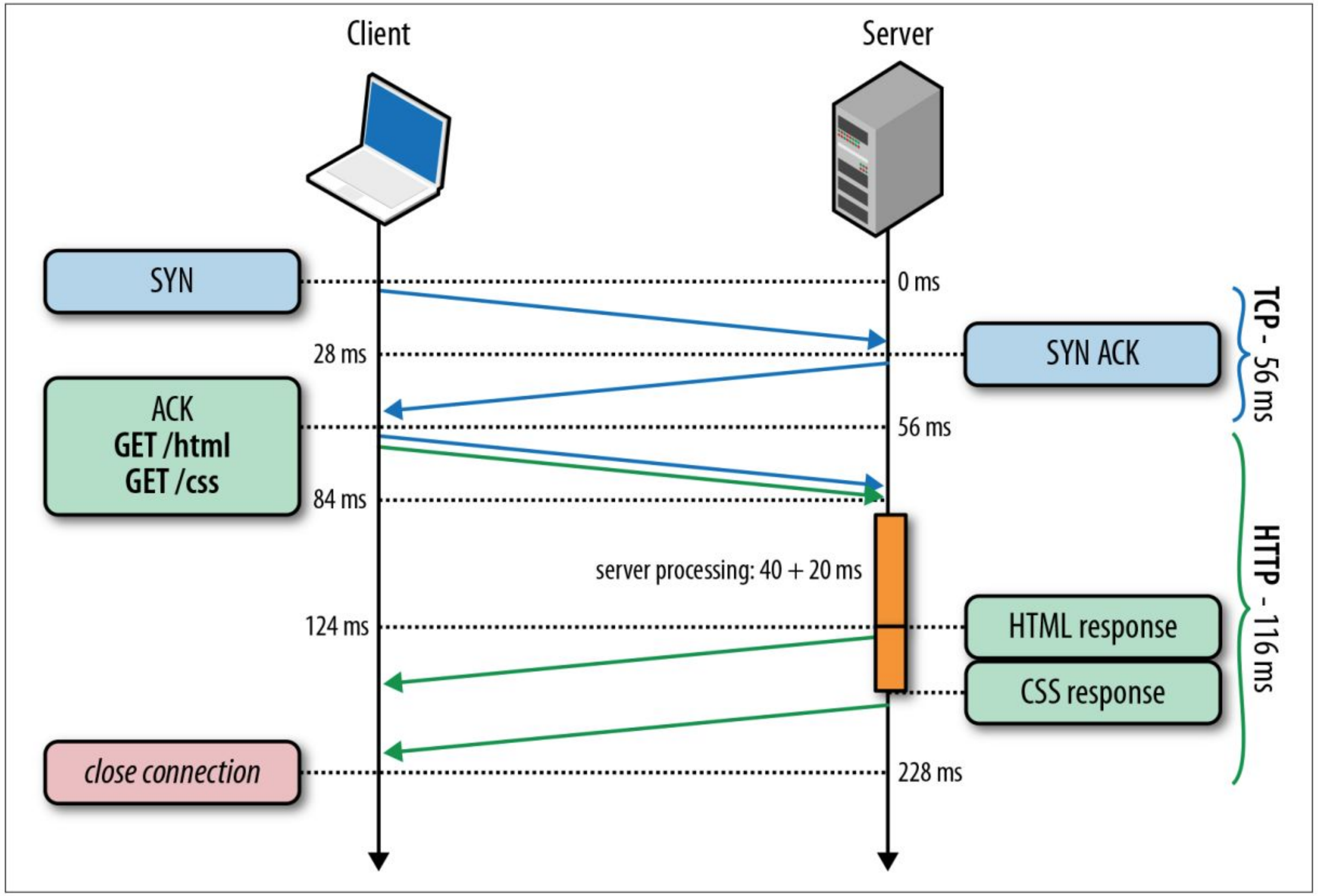
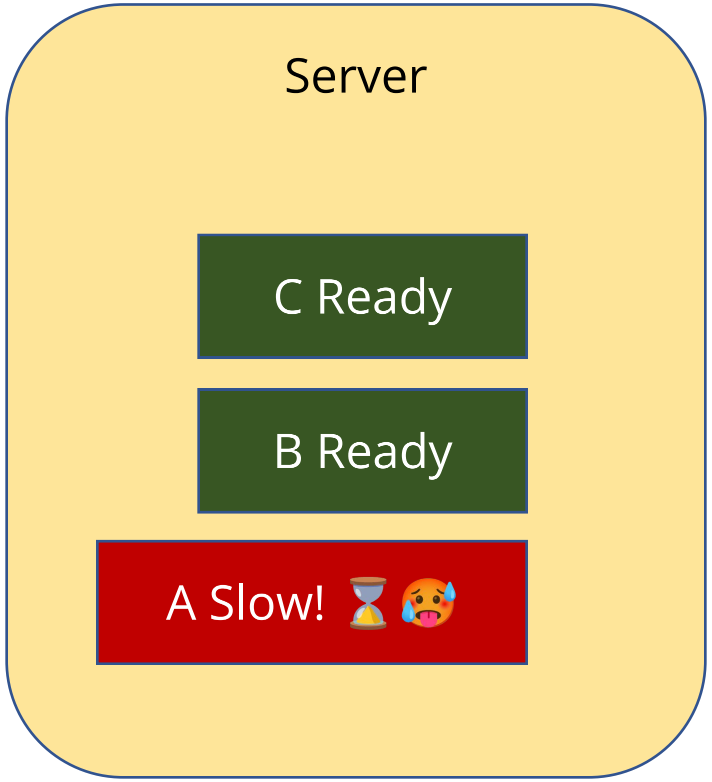


Figure 11-3. Pipelined HTTP requests with server-side FIFO queue

- Strict serialization of responses!
- We cannot interleave responses on the same connection
- We have to block for the full response to complete

“head of line blocking”

- In HTTP/1.1, the server could be ready to send responses B and C, but cannot do it until A ready!
  - No response reordering allowed
- Wastes capacity since link could be used for B & C, but sits idle while server prepares A!



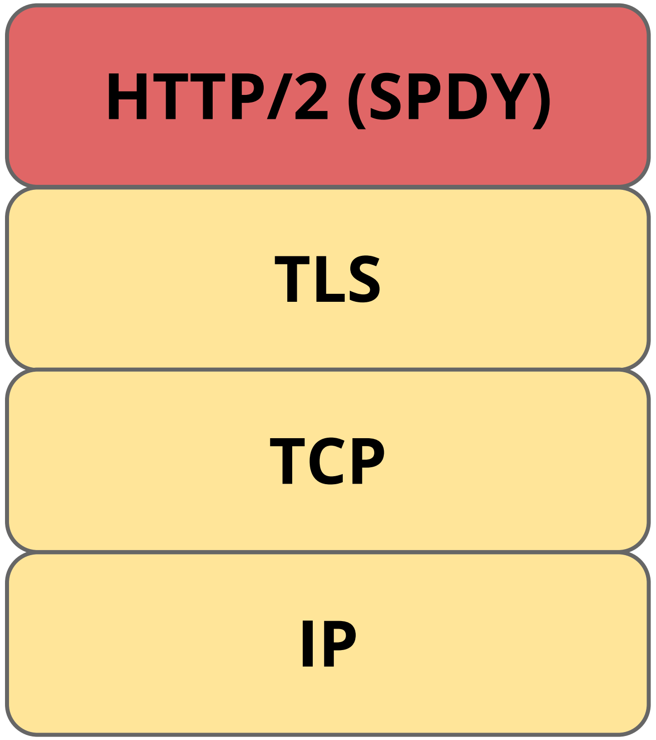
HTTP Changes

HTTP/1.0: TCP connection per request

HTTP/1.1: Persistence and pipelining

HTTP2/SPDY: Targeted performance specifically

- All happens below HTTP layer
- Prioritized stream multiplexing
- Header compression
- Server push
- Started as SPDY, standardized as HTTP/2 in 2015 after ~~every possible bikeshed~~ deep discussion

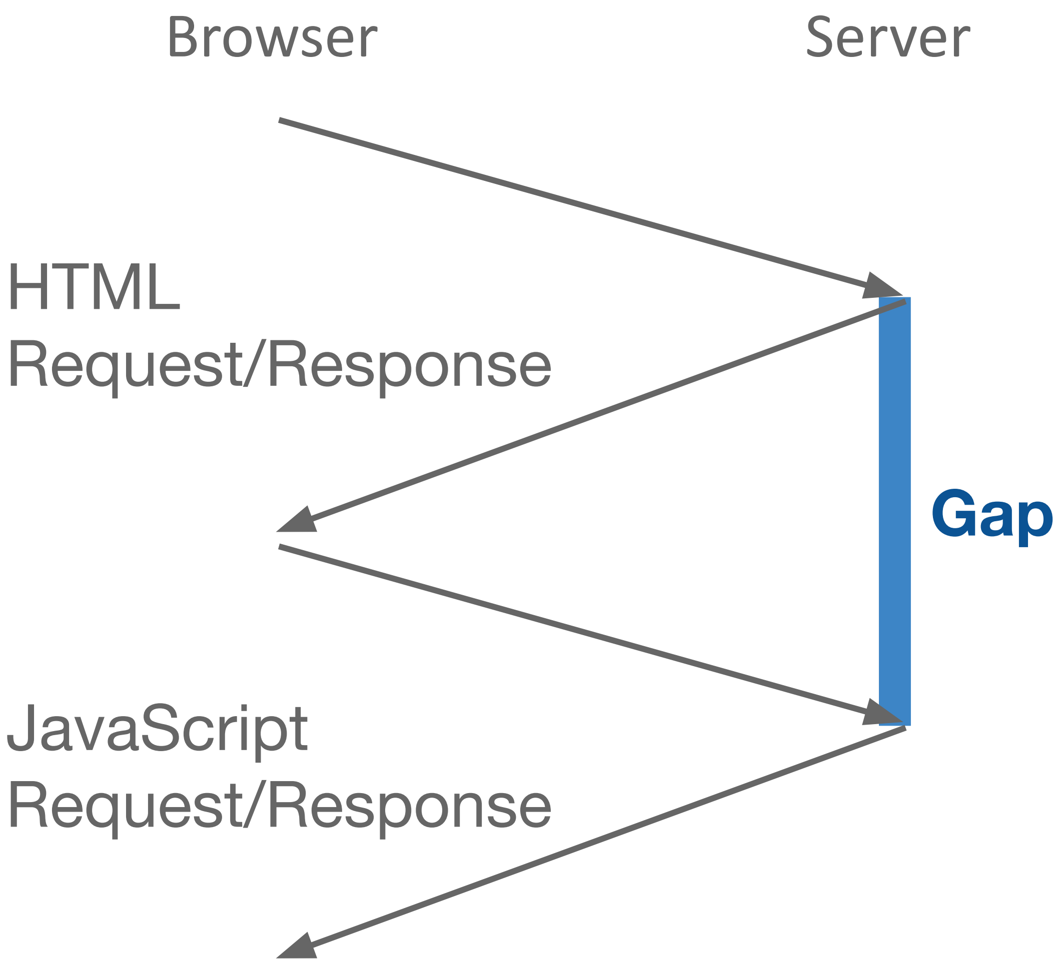


HTTP 2 Optimizations

- Prioritized Stream Multiplexing
  - HTTP 1.0: Each HTTP connection has own TCP
  - HTTP 1.1: Share one TCP connection to save setup
  - HTTP 2.0: Allow multiple **concurrent HTTP connections** in a single TCP flow to avoid head-of-line blocking
- Header Compression
  - HTTP Headers very wordy; Designed to be human readable
    - This is not great for pages making many requests.
  - Let’s compress them (usually gzip).
- Server Push
  - Server knows the client will eventually need a resource
  - Send it proactively before its even requested

Server Push: example resource loading gap

- Browser requests and receives HTML, encounters `<script src=”...”>`
- Similarly, JavaScript might src a dependent JavaScript file

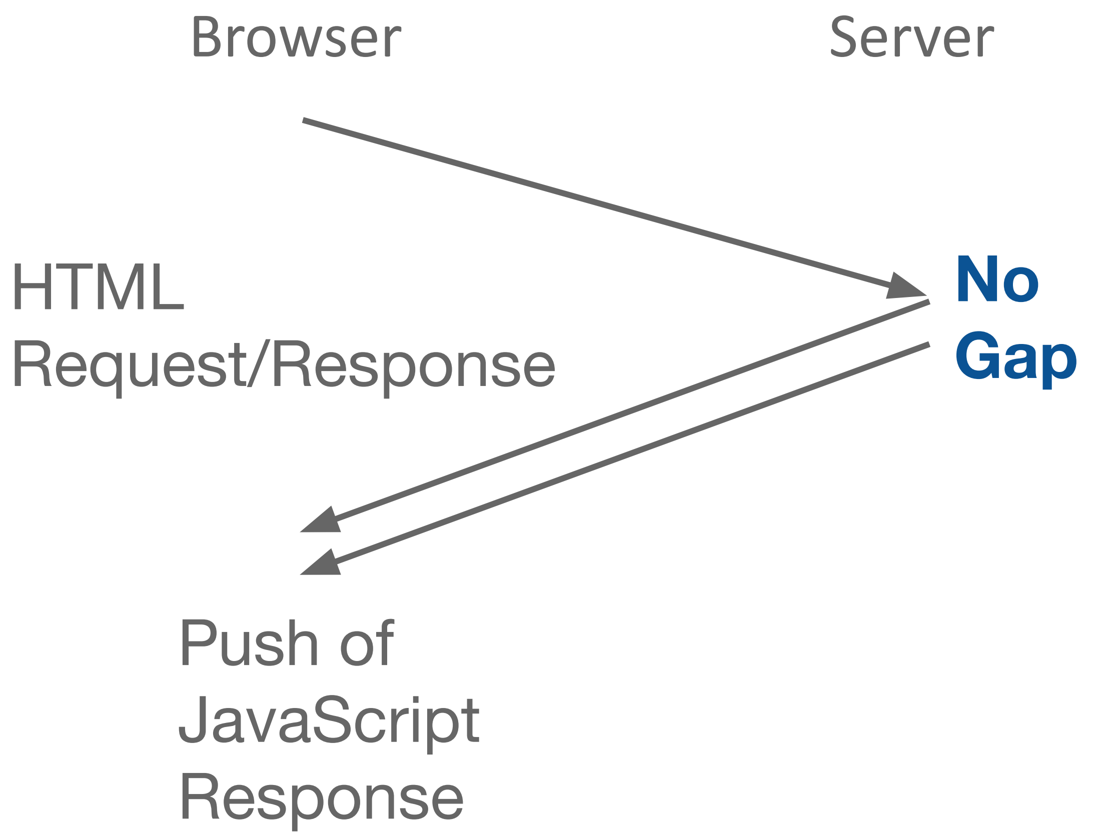


Server Push: example resource loading gap

Use **HTTP/2 server push** to close gaps

Or use **Link: rel=preload**

- Particularly useful for **hidden render blocking resources (HRBRs)**
    - Basically preload things that block rendering

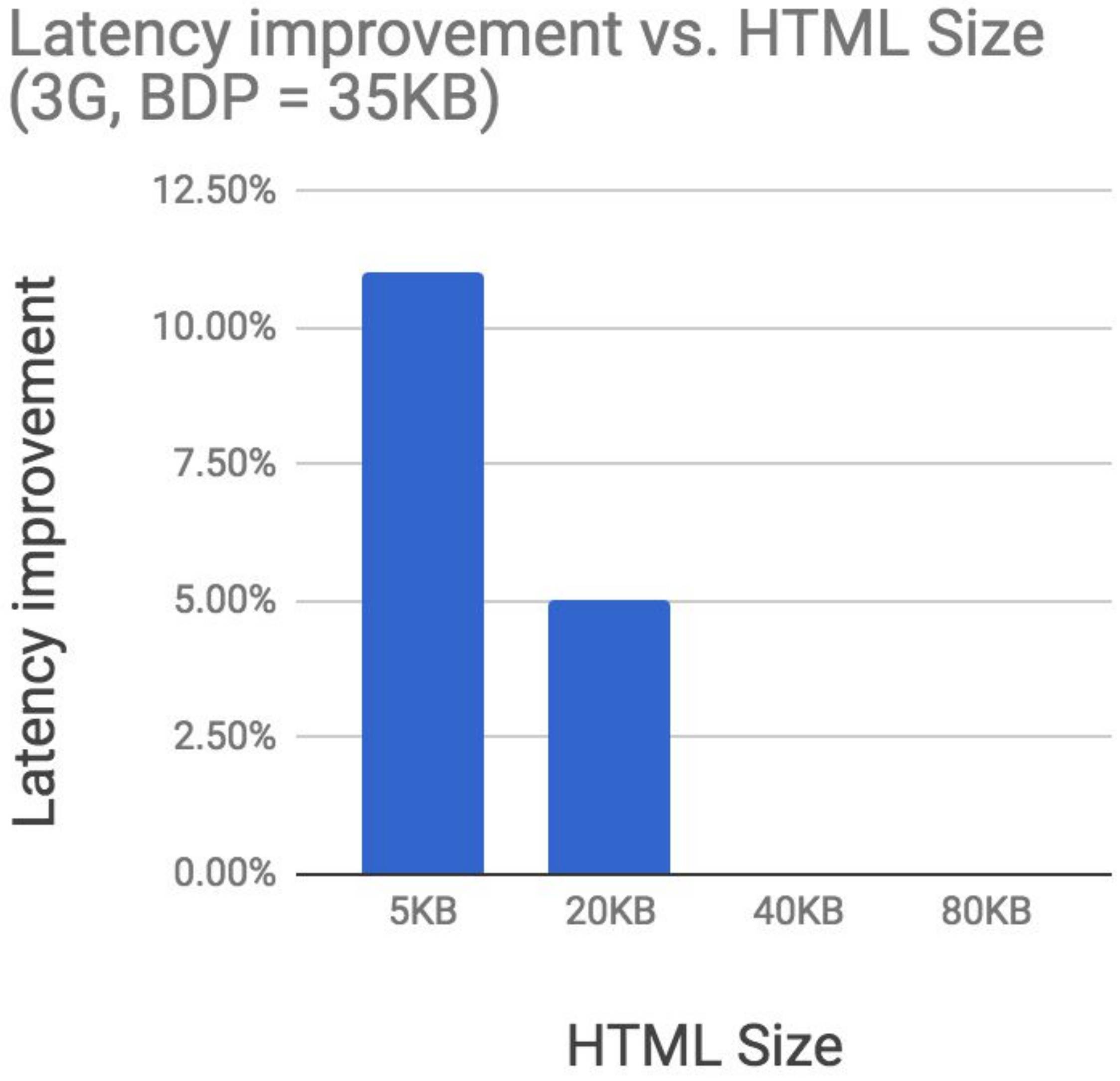


Simple server push lab experiment

Result: No benefit when HTML size > BD Product

Why? No gap even without push.

Opportunity only on high BDP networks, e.g., LTE, Satellite, and Cable



HTTP/2 is a solid improvement, but…

- Uses a single TCP connection as the underlying transport…
- What service model does TCP provide?
- What happens when a segment is lost?
  - Common occurrence in mobile and congested networks

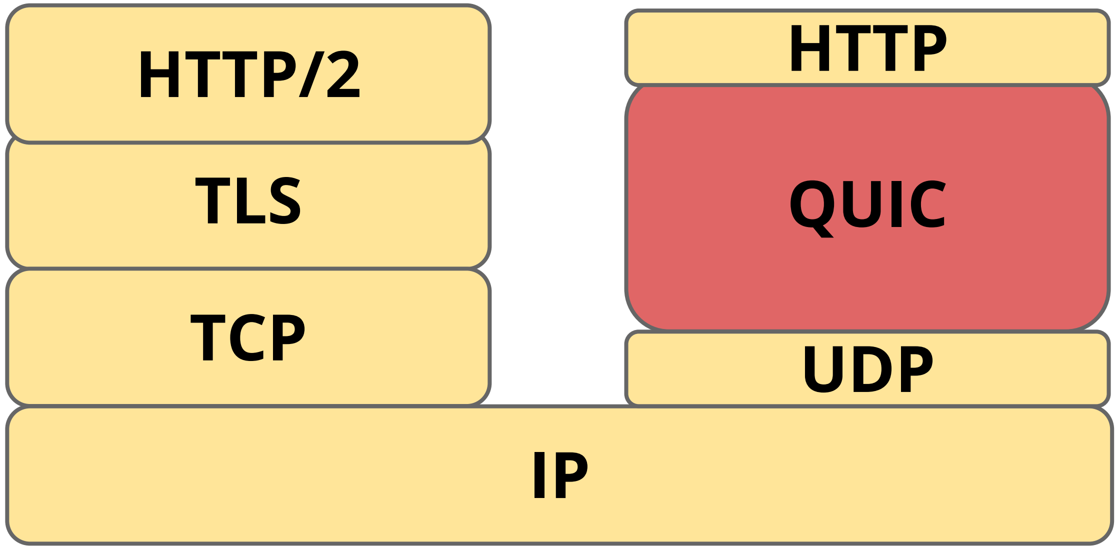
To keep the service model of an in-order stream of bytes, have to wait for the segment to be retransmitted before passing to the application layer!

Even if following received segments could contain a complete response!

Other issues too (will discuss), many challenges with mobile networks!

QUIC & HTTP/3

- Goal: make HTTP**S** transport even faster!
- Deployed at Google starting 2014
- IETF working group formed 2016
- Standardized as HTTP/3 in October 2018
- IETF QUIC RFC 9000 in 2021
- Now responsible for >30% of internet traffic by volume

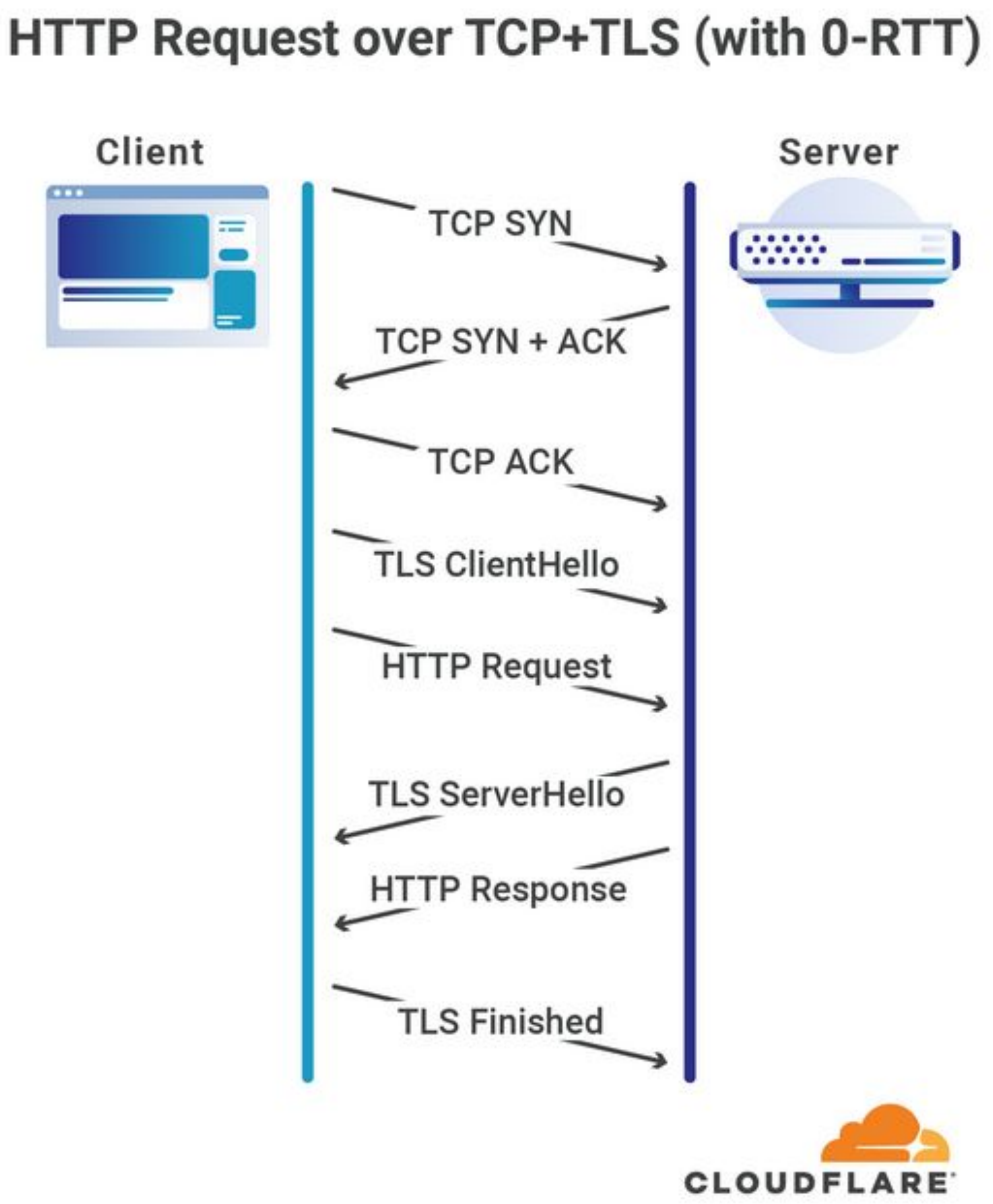


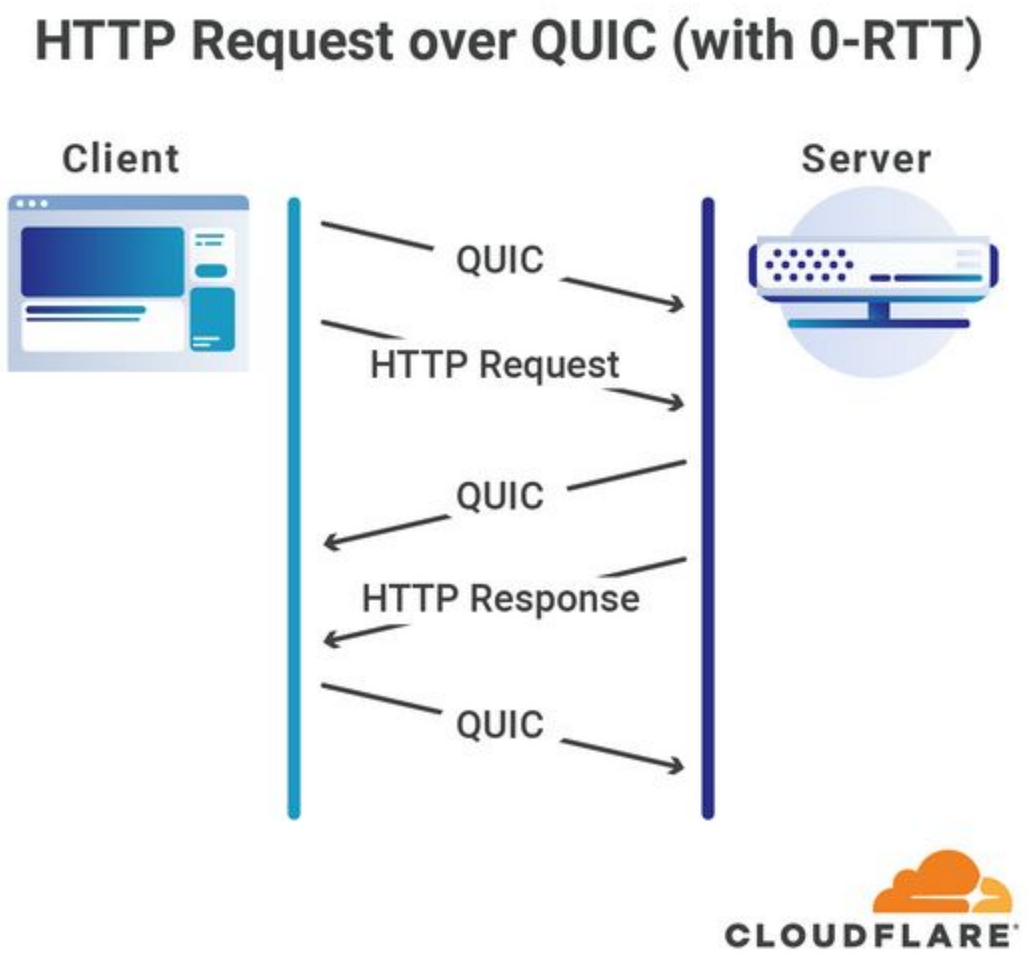
QUIC & HTTP/3 Innovations (1)

- Remove TCP, Switch to UDP
  - Error correction: Groups of packets contain a FEC packet which can be used to recreate lost packet.
  - Congestion control: Move congestion control to user space with pluggable implementations
  - BBR Implementation: all packets carry new sequence numbers, allows for precise roundtrip-time calculation.
  - Per-packet encryption (rather than flow)

QUIC & HTTP/3 Innovations (2)

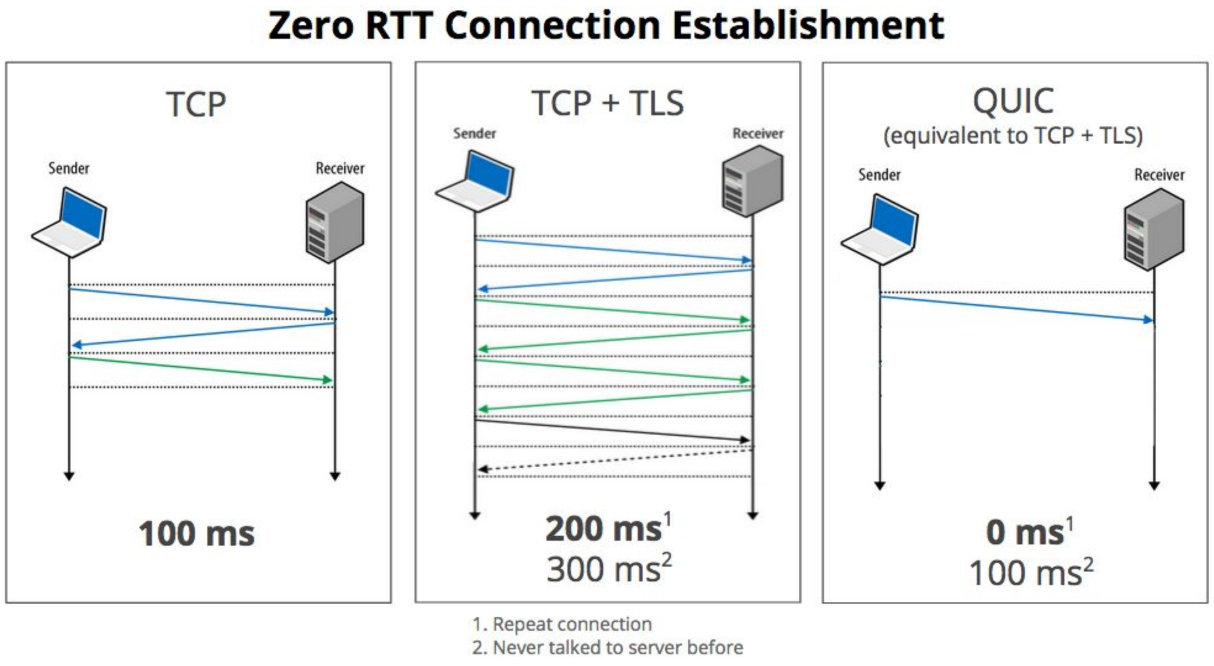
- Speed up connection establishment
  - Include TLS/Encryption in setup (TLS 1.3)
    - Next slide deck, sorry.
  - Similarly pack HTTP content into setup


HTTP Request over TCP+TLS (with 0-RTT)
Client
Server
TCP SYN
TCP SYN + ACK
TCP ACK
TLS ClientHello
HTTP Request
TLS ServerHello
HTTP Response
TLS Finished
CLOUDFLARE


HTTP Request over QUIC (with 0-RTT)
Client
Server
QUIC
HTTP Request
QUIC
HTTP Response
QUIC
CLOUDFLARE

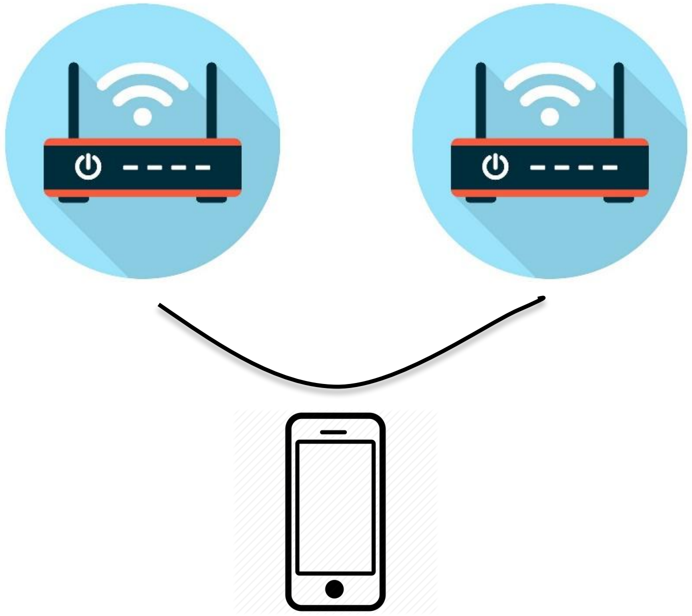
QUIC & HTTP/3 Innovations (3)

- Support mobility through 64-bit stream IDs
  - This means you can change IP address or ports but still keep your connection alive



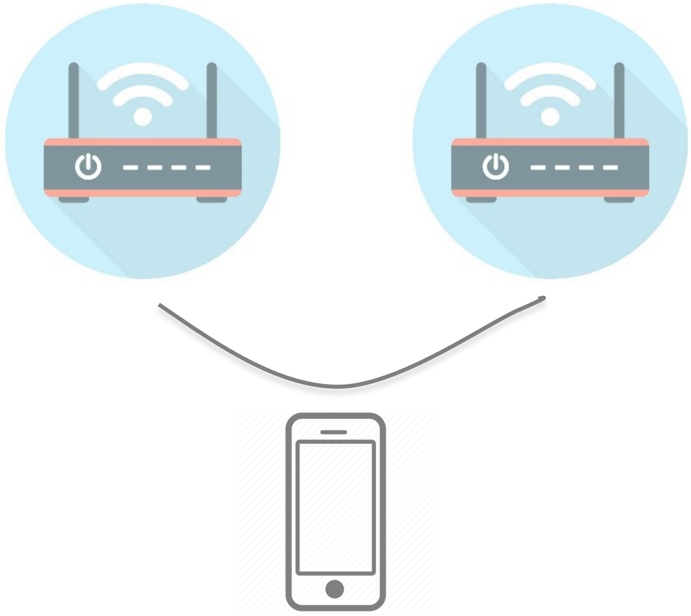
QUIC & HTTP/3: Problem of Mobility

- What happens to IP addresses and HTTP sessions when a user moves between wifi APs?

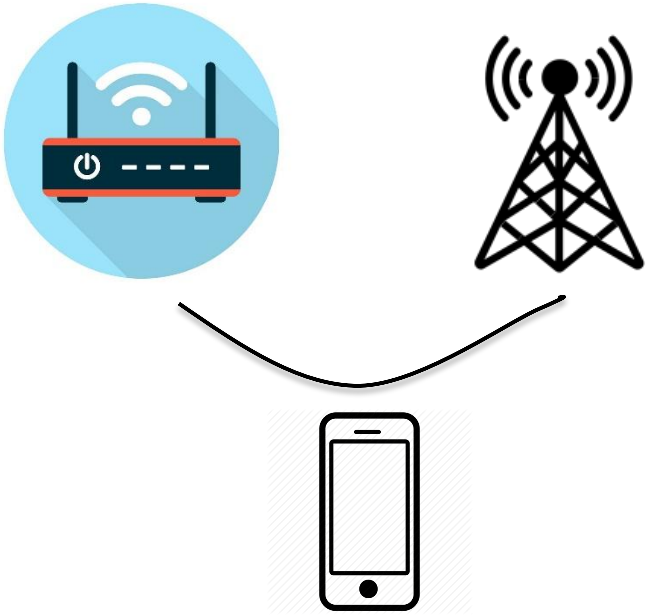


QUIC & HTTP/3: Problem of Mobility

- What happens to IP addresses and HTTP sessions when a user moves between wifi APs?

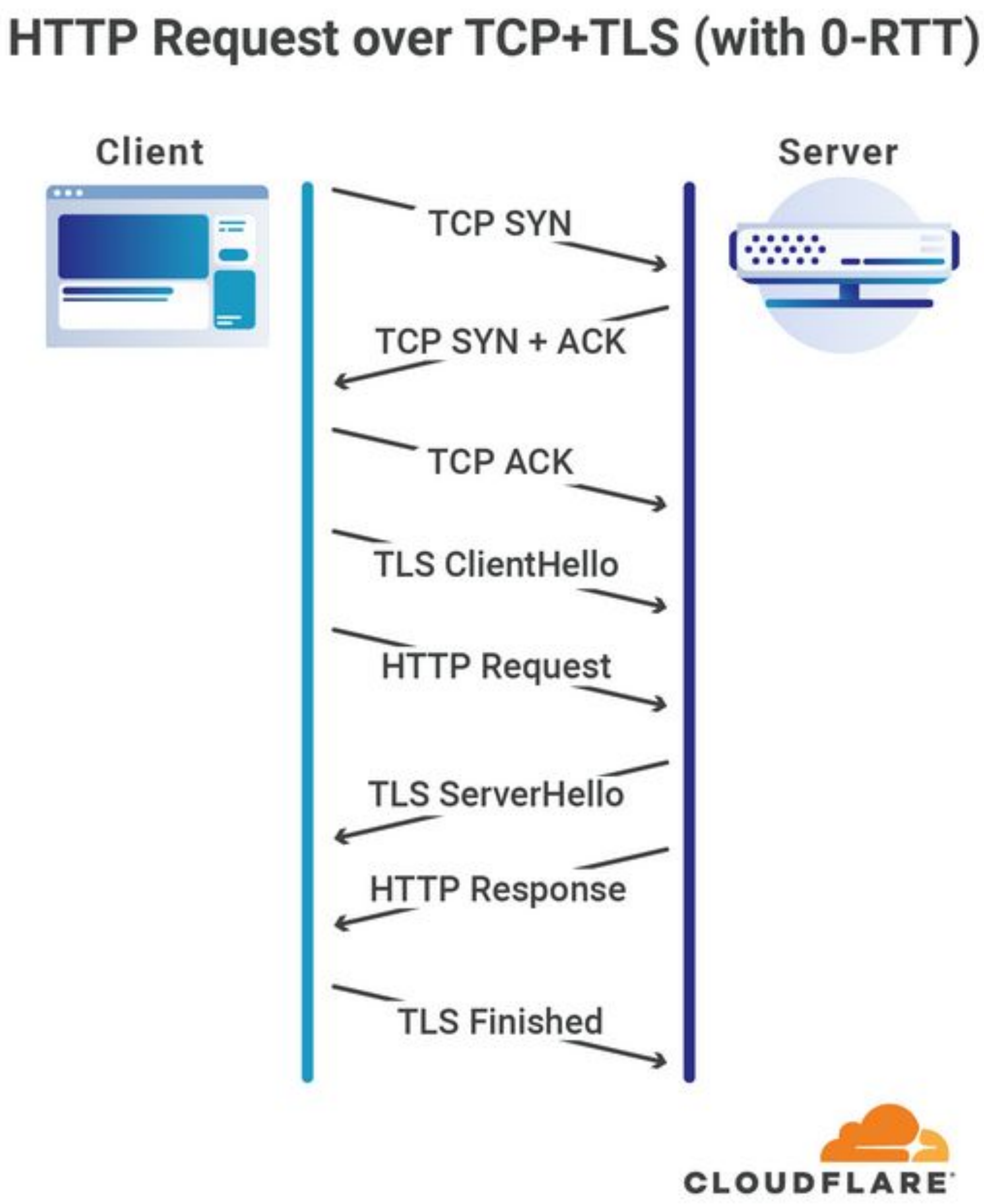


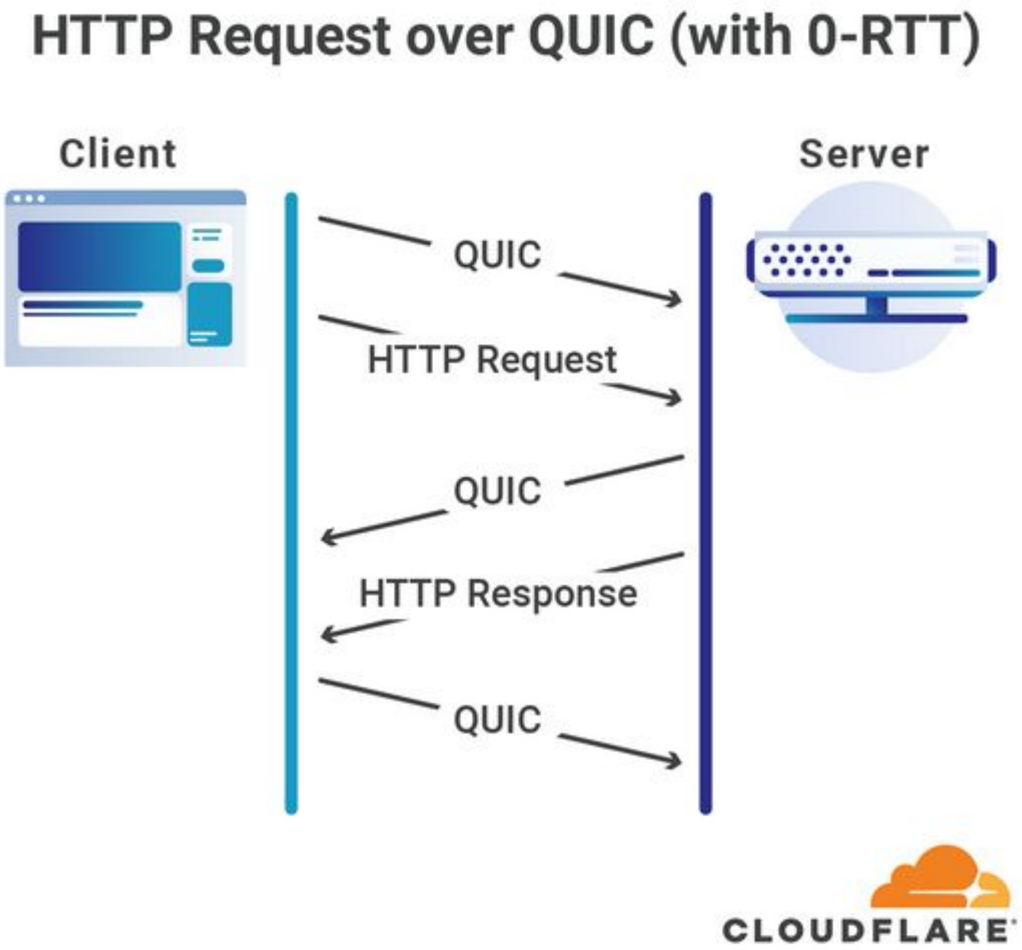
- What happens to IP addresses and HTTP sessions when a user moves between cellular and wifi?



IP Mobility

- Hard problem: IP addresses are supposed to identify nodes in the network but change as nodes move around.
- Proposed solutions:
  - IP Anchor: Place a server at an IP and tunnel traffic to user.
  - DNS Anchor: Have DNS server which rapidly updates as user moves between IP addresses
- All try to keep some global state constant:
  - IP or DNS Name


HTTP Request over TCP+TLS (with 0-RTT)
Client
Server
TCP SYN
TCP SYN + ACK
TCP ACK
TLS ClientHello
HTTP Request
TLS ServerHello
HTTP Response
TLS Finished
CLOUDFLARE


HTTP Request over QUIC (with 0-RTT)
Client
Server
QUIC
HTTP Request
QUIC
HTTP Response
QUIC
CLOUDFLARE

QUIC summary

- A transport specifically designed to work well with HTTPS
- Makes HTTPS faster, particularly in the tail
- Moves congestion control and stream handling into an application library
  - Client innovation without OS updates
- Now > 30% of total Internet traffic (2021)
  - 8.3% of websites (2025)
  - Meta (75%!), Google, Cloudflare, and Others

A REST vs GraphQL

A REST vs GraphQL

Best Practices

- **Naming matters.** As soon as a client starts to make use of a field, it becomes much harder to change the name that they're using. Names should be obvious and self-documenting for the benefit of newcomers to the code base, and flexible enough to accommodate future versions of your product.
- **Think in graphs, not endpoints.** Unlike traditional APIs, GraphQL exposes all of the data from a single endpoint. Your objective with GraphQL should be to create a unified, cohesive graph that allows users of your API to create subsets of that graph in order to build new product experiences. For example, by treating a search as an object, you might have that search object point to other search objects in order to suggest related searches to the end user.
- **Describe the data, not the view.** Make sure that your API isn't linked too closely with the demands of your initial interface. If you're building an API for an iOS app, you might later port it to other devices such as desktop computers or feature phones. When building queries, focus on the underlying data, rather than how it's represented in your application.
- **GraphQL is a thin interface.** GraphQL is intended to sit atop your existing and future systems. What it's not intended to do are things like authentication, authorization, caching, database query and optimization. By handling these concerns below the GraphQL service layer, your application becomes more resilient as your platform evolves and as you migrate between different services.
- **Hide implementation details.** While GraphQL should be thin, it should also be a seam between your data and the details of its storage and retrieval. If you're giving away too much data in a single response, then some client might use the extra information for its own purposes, which results in their code base breaking when the implementation changes behind the scenes.