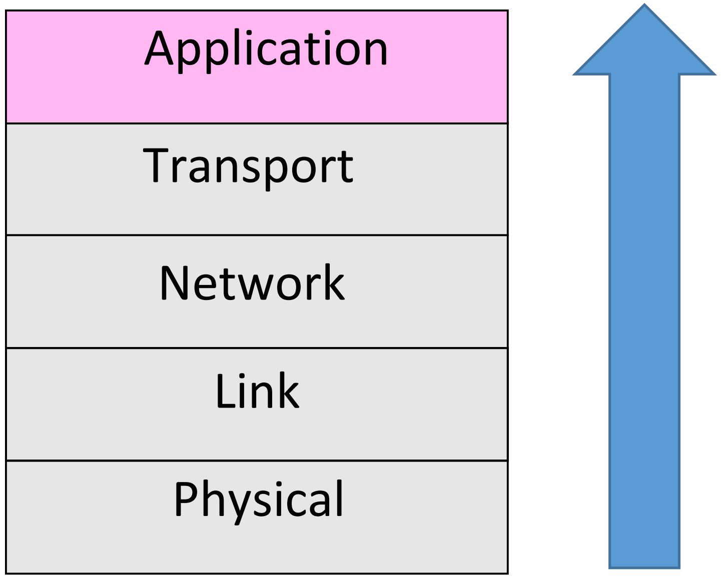
Applications!

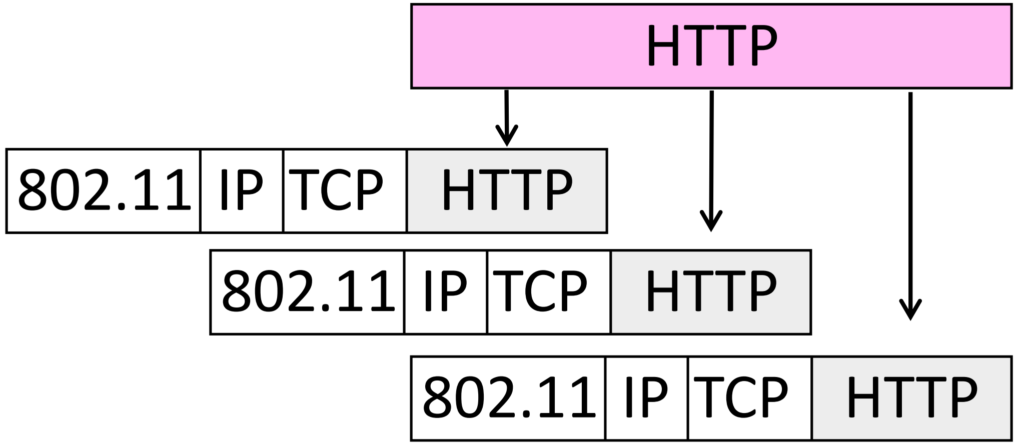
Where we are in the Course

- Application layer protocols are often part of "app"
  - But don't need a GUI, e.g., DNS



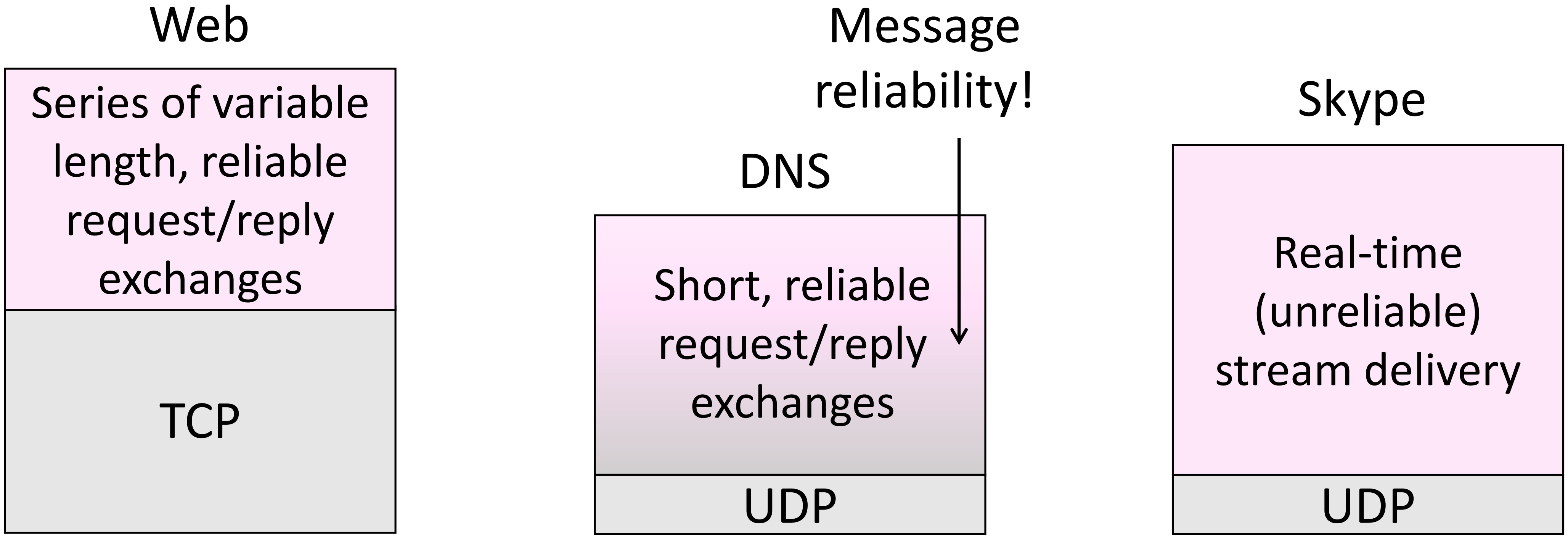
Recall

- 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 …

But consider part of the application, not strictly layered!

7	Application	– Provides functions needed by users
6	Presentation	– Converts different representations
5	Session	– Manages task dialogs
4	Transport	– Provides end-to-end delivery
3	Network	– Sends packets over multiple links
2	Data link	– Sends frames of information
1	Physical	– Sends bits as signals

Session Concept

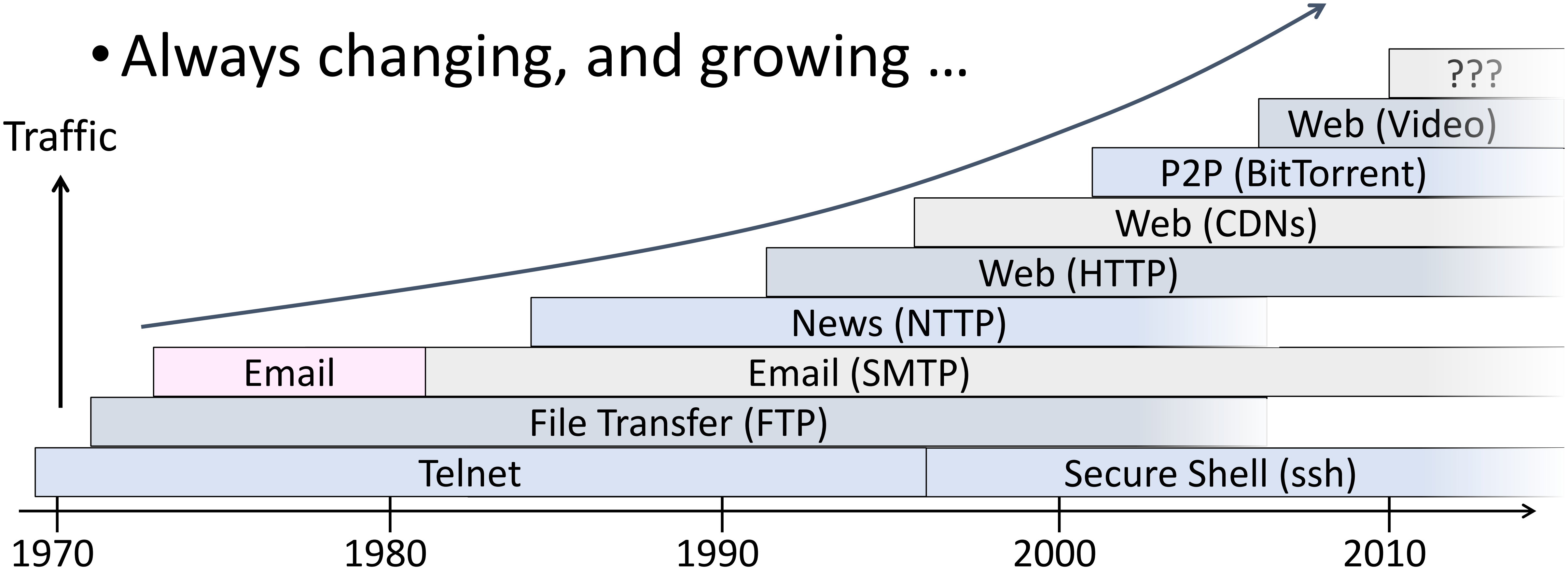
- A session is a series of related network interactions in support of an application task
  - Often informal, not explicit
- Examples:
  - Web page fetches multiple resources
  - Skype call involves audio, video, chat

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

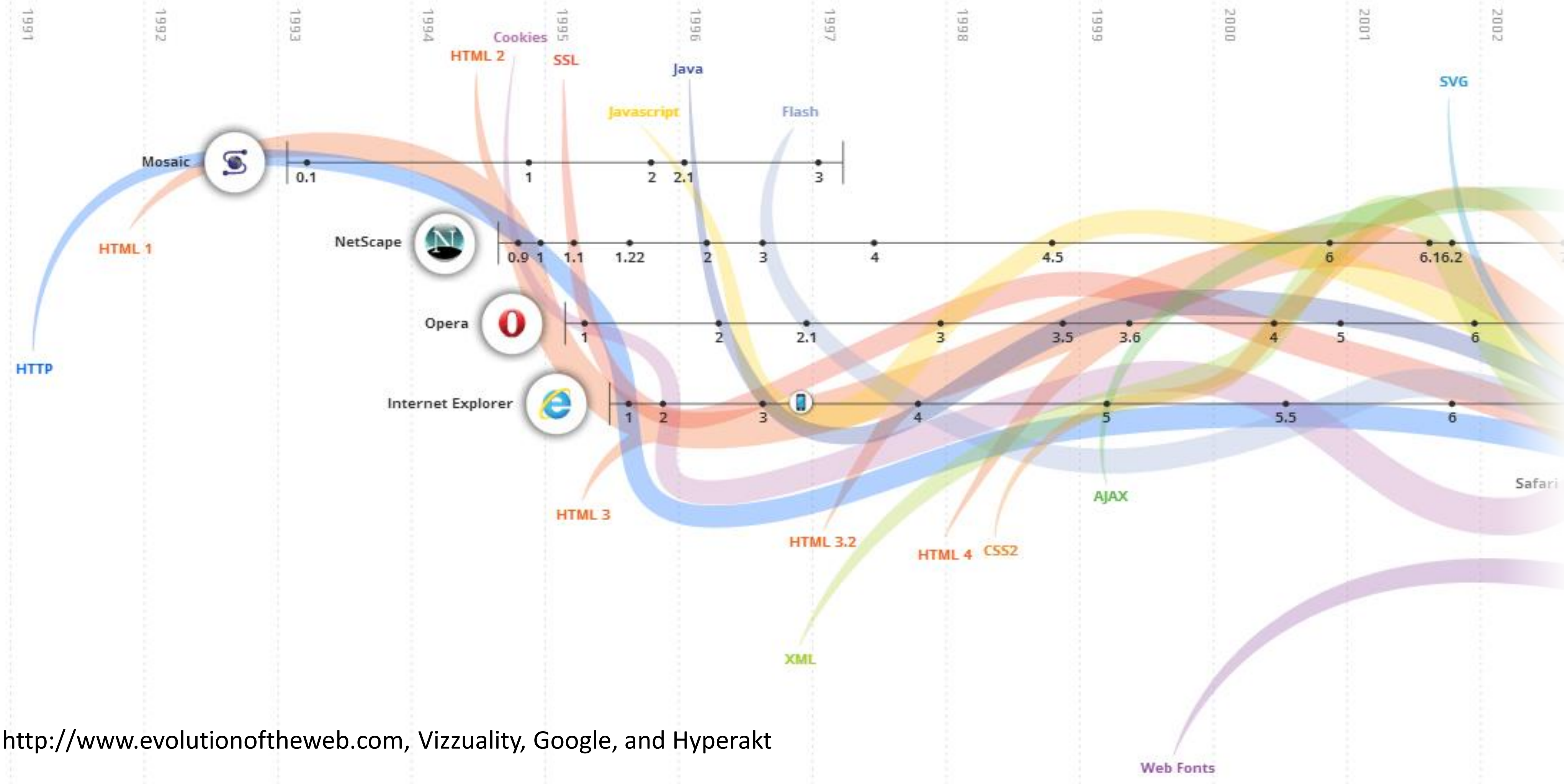
- Always changing, and growing …



Evolution of Internet Applications (2)

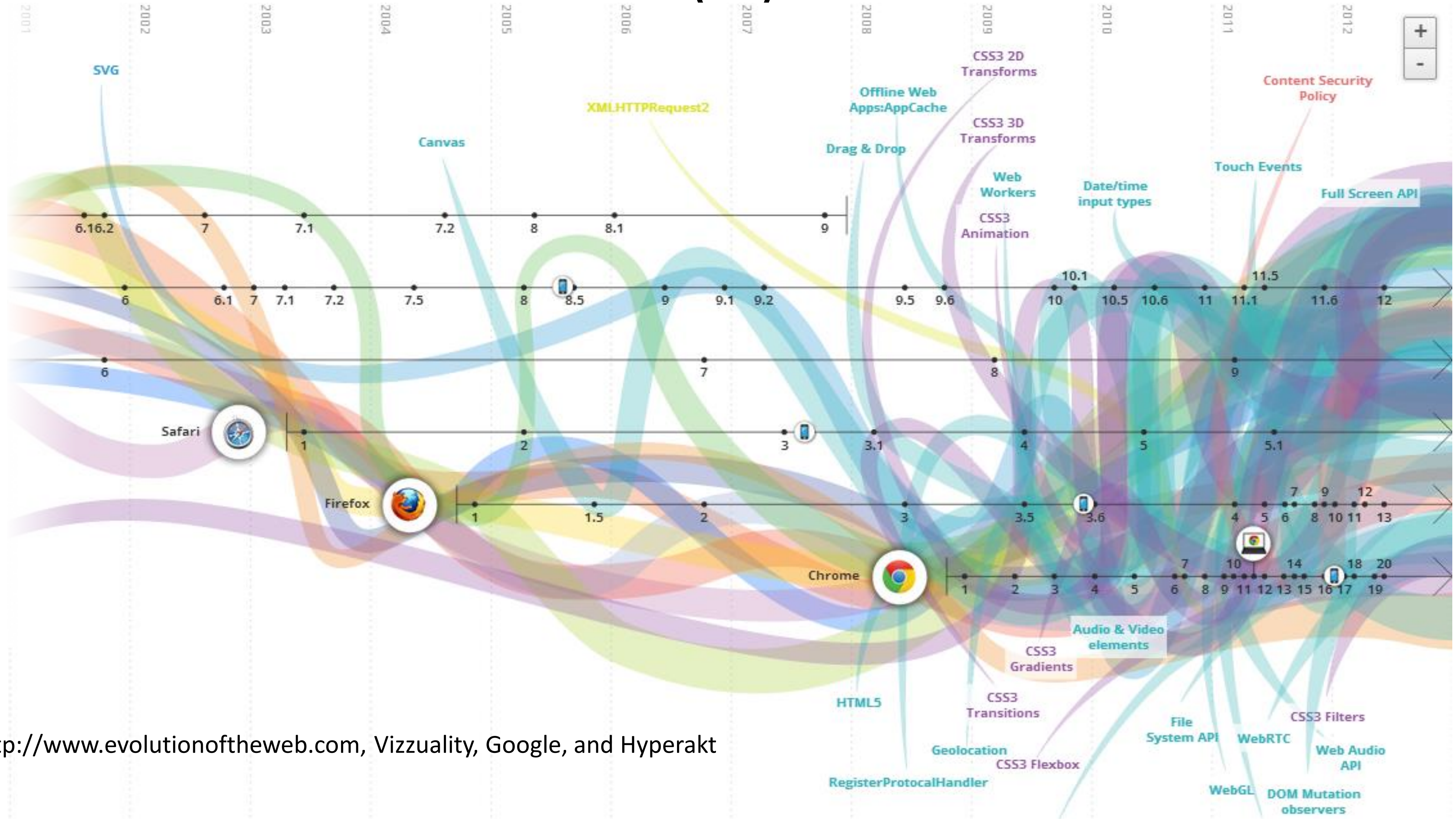
- For a peek at the state of the Internet:
  - Akamai’s State of the Internet Report (quarterly)
  - Cisco’s Visual Networking Index
  - Mary Meeker’s Internet Report
- Robust Internet growth, esp. video, wireless, mobile, cat
  - Most (70%) traffic is video (expected 80% in 2019)
  - Mobile traffic overtakes desktop (2016)
  - 15% of traffic is cats (2013)
  - Growing attack traffic from China, also U.S. and Russia

Evolution of the Web



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

Evolution of the Web (2)

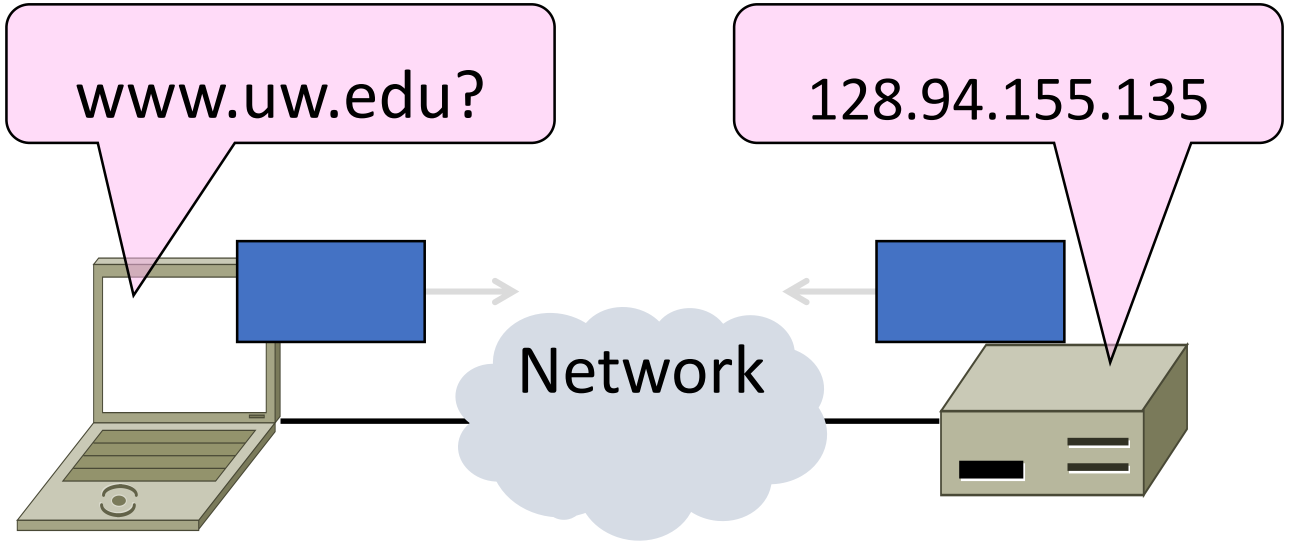


Source: http://www.evolutionoftheweb.com, Vizzuality, Google, and Hyperakt

Domain Name System

DNS

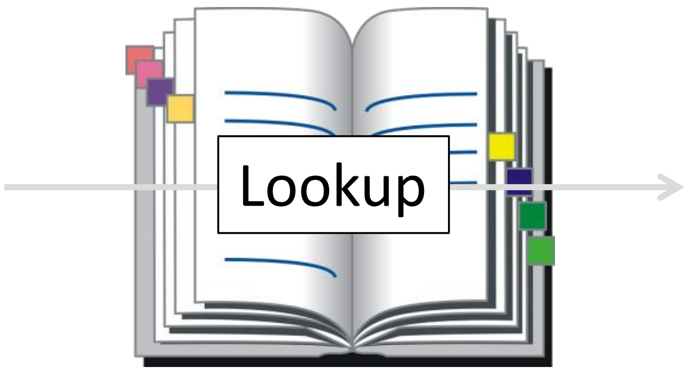
- Human-readable host names, and more



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

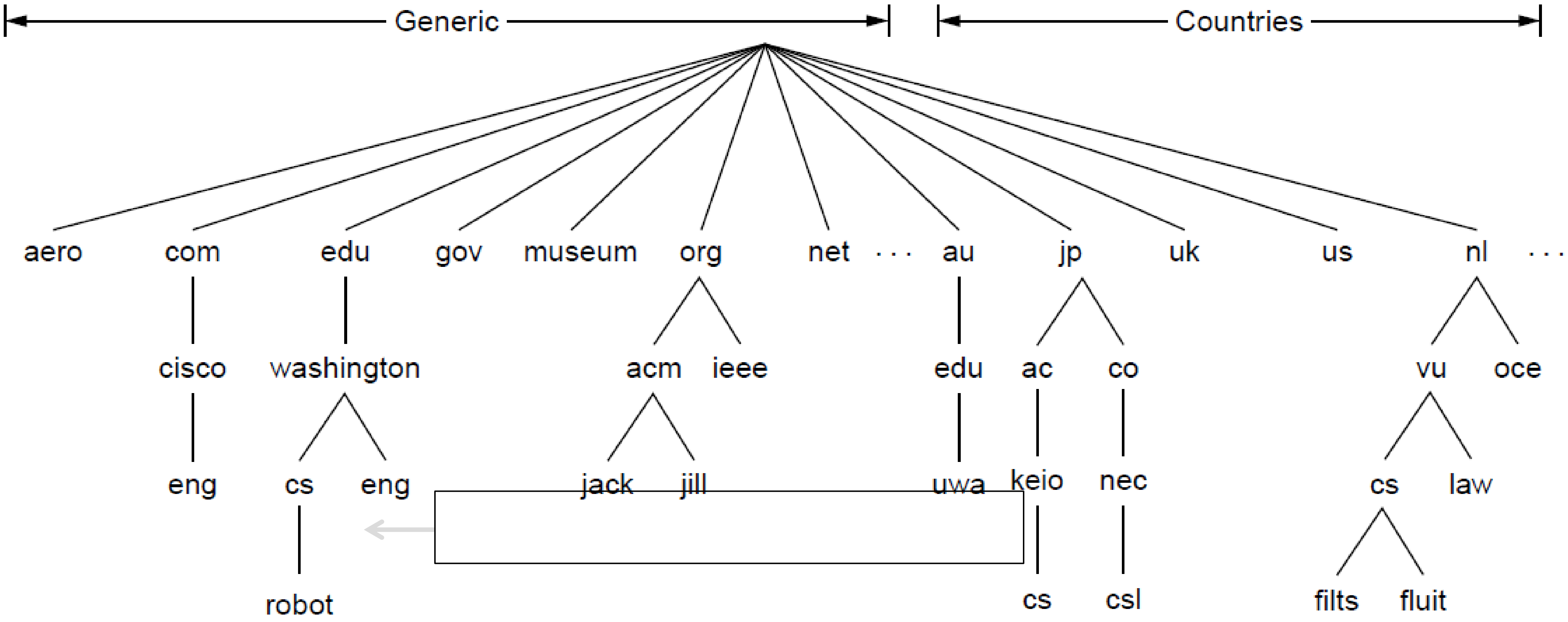
- Directory was a file HOSTS.TXT regularly retrieved for all hosts from a central machine at the NIC (Network Information Center)
- Names were initially flat, became hierarchical (e.g., lcs.mit.edu) ~85
- Not manageable or efficient as the ARPANET grew …

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)

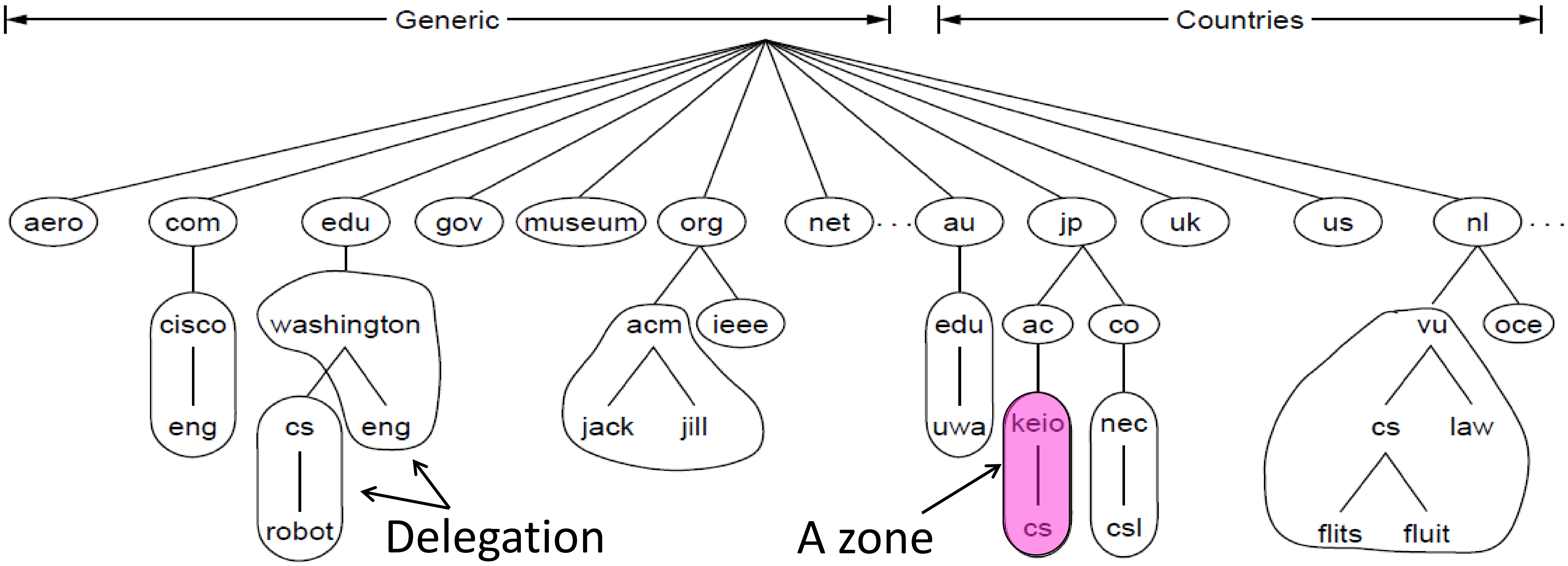


TLDs (Top-Level Domains)

- 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)

DNS Zones

- A zone 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 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
```

Name server

IP addresses of computers

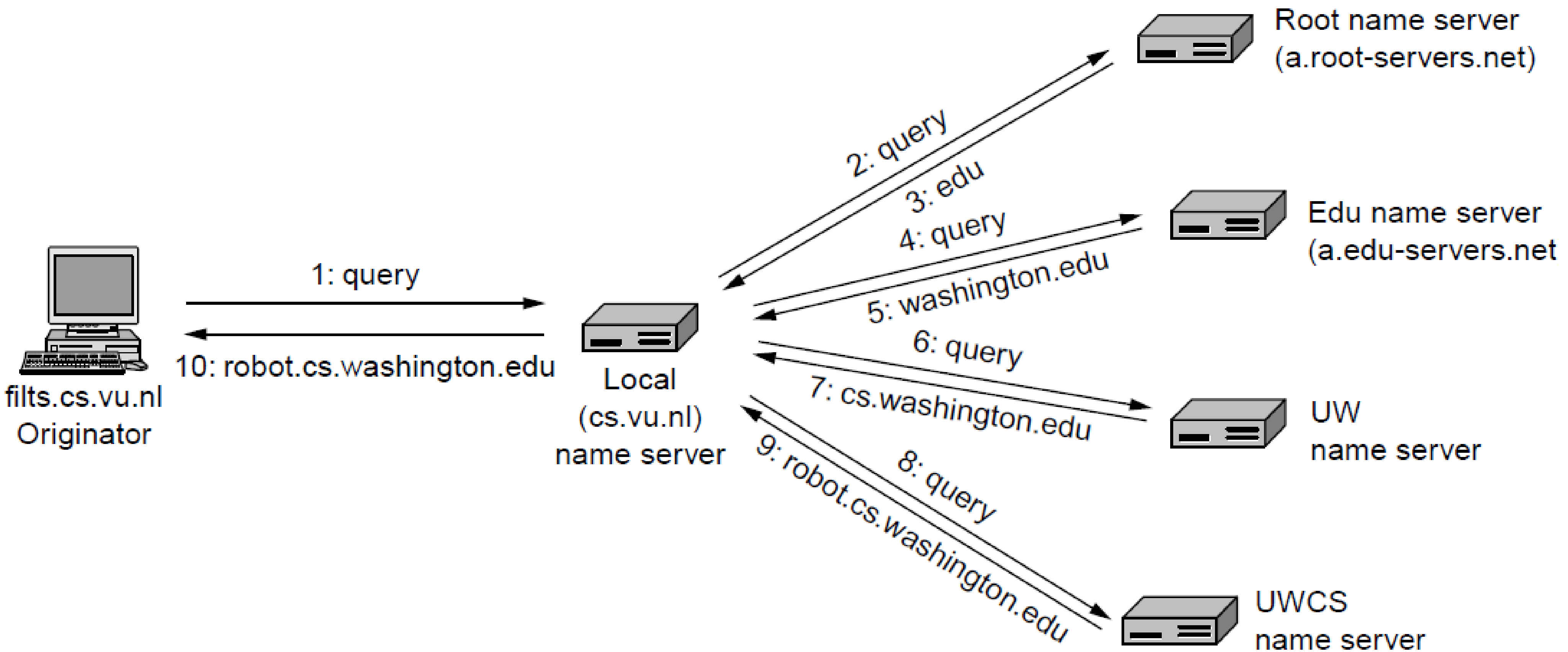
Mail gateways

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

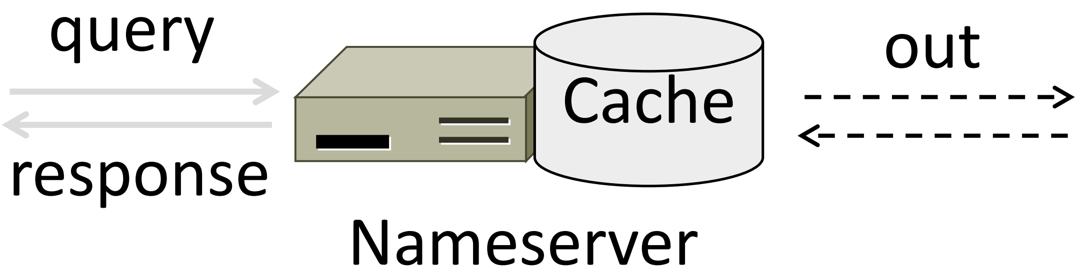
- Recursive query
  - Nameserver resolves and returns final answer
  - E.g., flits → local nameserver
- Iterative (Authoritative) query
  - Nameserver returns answer or who to contact for answer
  - E.g., local nameserver → all others

Iterative vs. Recursive Queries (2)

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

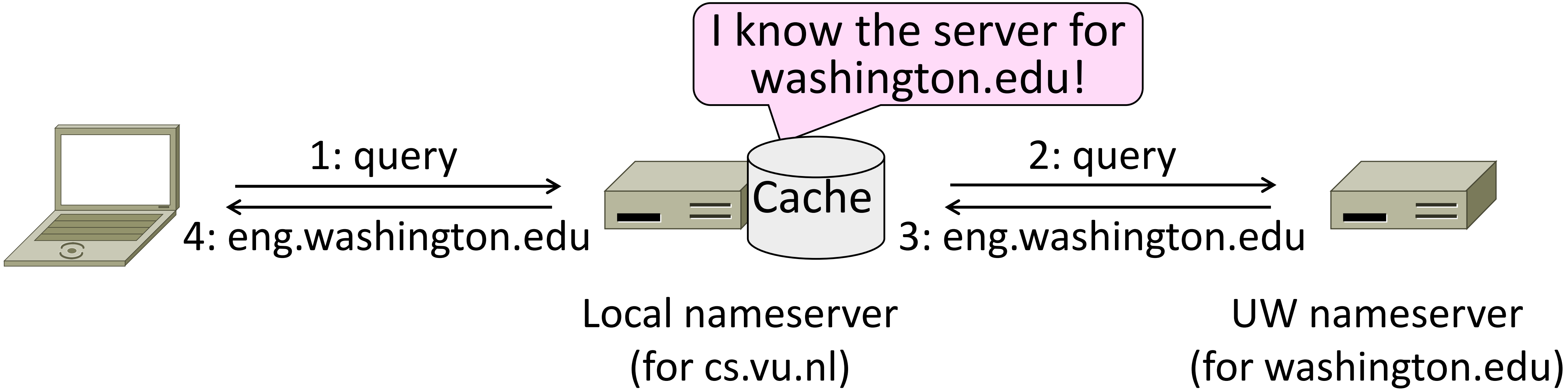
Caching

- Resolution latency should be low
  - Adds delay to web browsing
- Cache query/responses to answer future queries immediately
  - Including partial (iterative) answers
  - Responses carry a TTL for caching



Caching (2)

- flits.cs.vu.nl now resolves eng.washington.edu
  - And previous resolutions cut out most of the process



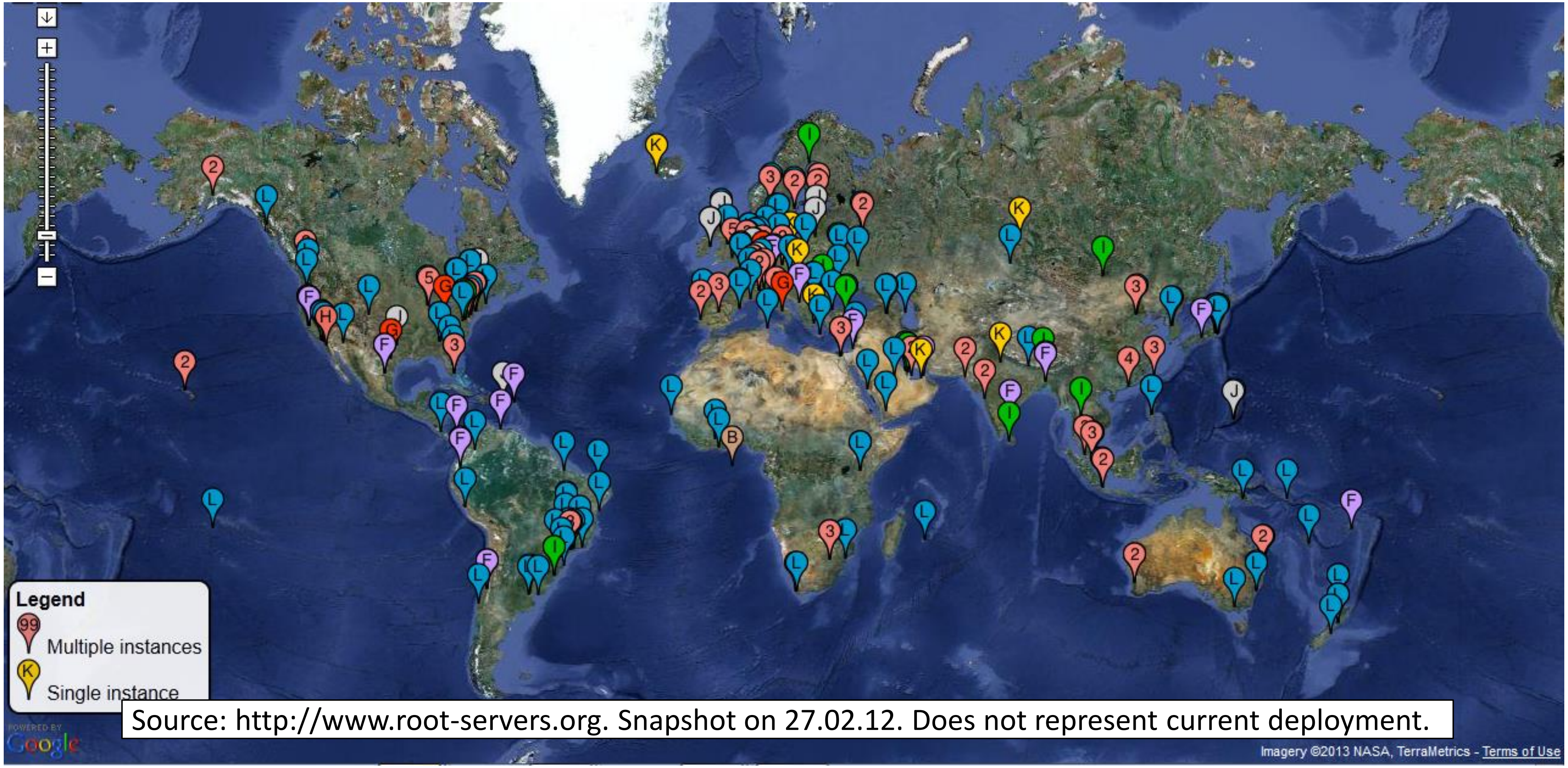
Local Nameservers

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

Root Nameservers

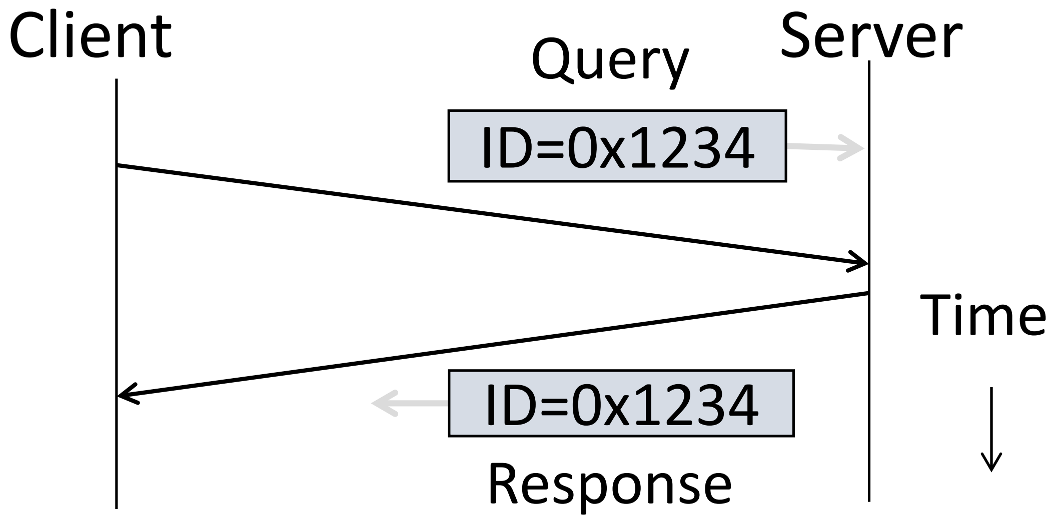
- Root (dot) is served by 13 server names
  - a.root-servers.net to m.root-servers.net
  - All nameservers need root IP addresses
  - Handled via configuration file (named.ca)
- There are >250 distributed server instances
  - Highly reachable, reliable service
  - Most servers are reached by <u>IP anycast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

Root Server Deployment



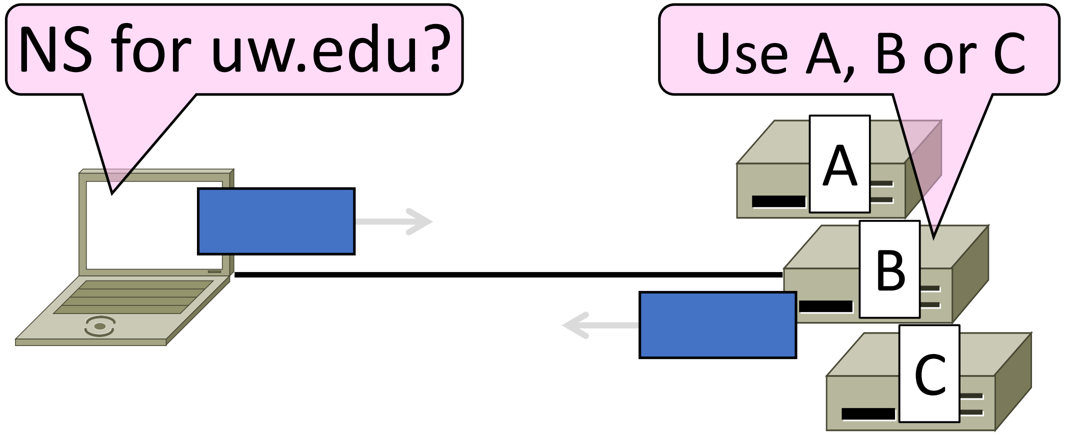
DNS Protocol

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



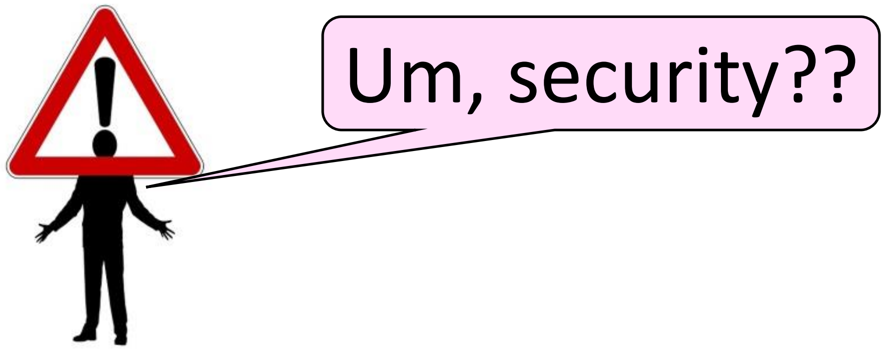
DNS Protocol (2)

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



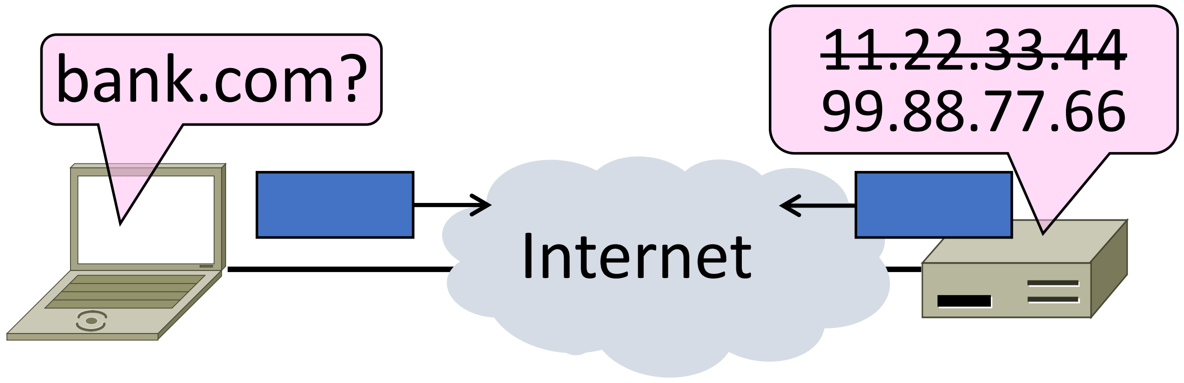
DNS Protocol (3)

- Security is a major issue
  - Compromise redirects to wrong site!
  - Not part of initial protocols ..
- DNSSEC (DNS Security Extensions)
  - Mostly deployed



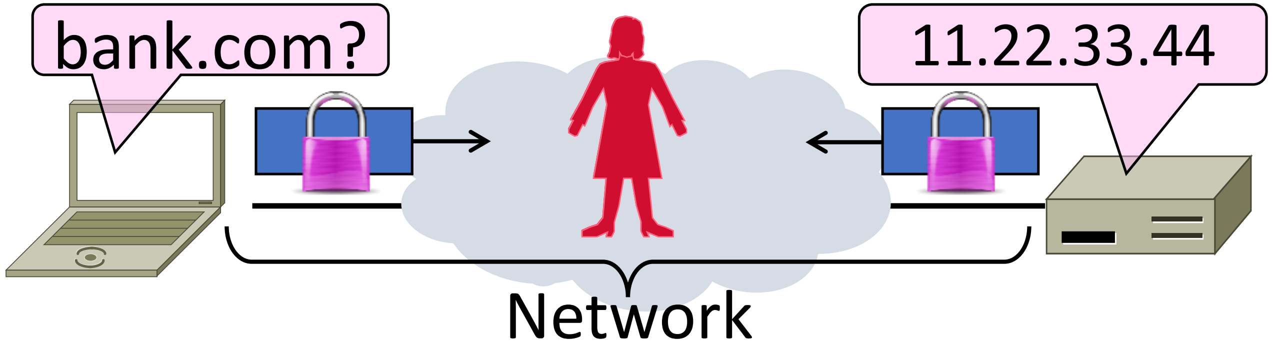
Goal and Threat Model

- Naming is a crucial Internet service
  - Binds host name to IP address
  - Wrong binding can be disastrous …



Goal and Threat Model (2)

- Goal is to secure the DNS so that the returned binding is correct
  - Integrity/authenticity vs confidentiality
- Attacker can tamper with messages on the network

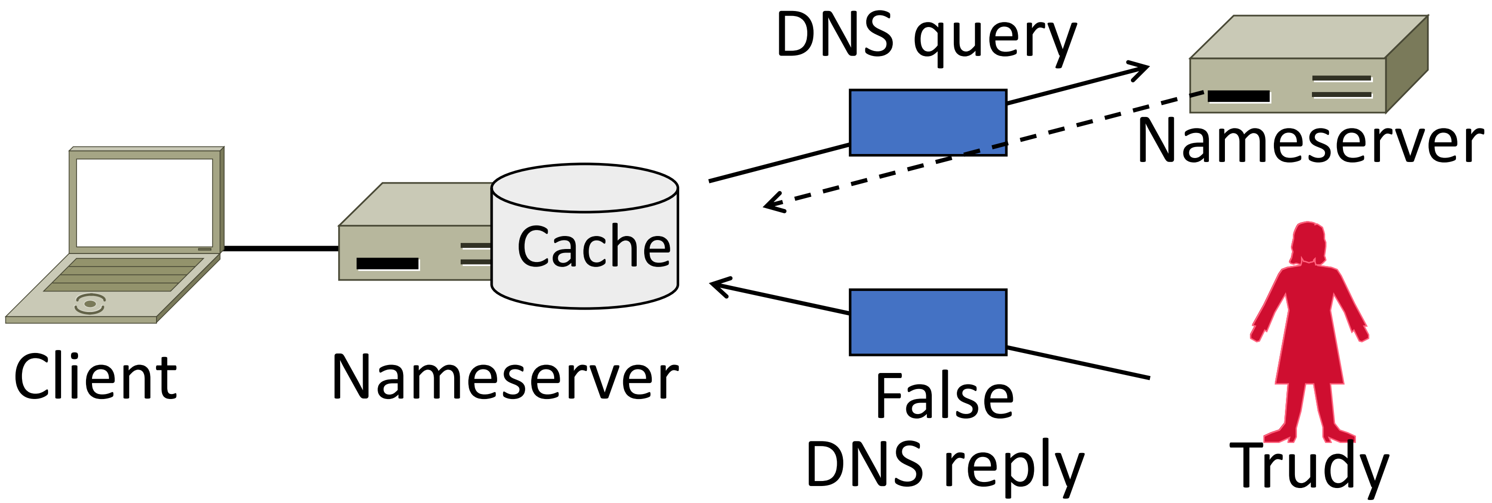


DNS Spoofing

- Hang on – how can attacker corrupt the DNS?
- Can trick nameserver into caching the wrong binding
  - By using the DNS protocol itself
  - This is called DNS spoofing

DNS Spoofing (2)

- To spoof, Trudy returns a fake DNS response that appears to be true
  - Fake response contains bad binding



DNS Spoofing (3)

- Lots of questions!
    1. How does Trudy know when the DNS query is sent and what it is for?
    2. How can Trudy supply a fake DNS reply that appears to be real?
    3. What happens when the real DNS reply shows up?
- There are solutions to each issue …

DNS Spoofing (4)

1. How does Trudy know when the query is sent and what it is for?

- Trudy can make the query herself!
  - Nameserver works for many clients
  - Trudy is just another client

DNS Spoofing (5)

2. How can Trudy supply a fake DNS reply that appears to be real?

- A bit more difficult. DNS checks:
  - Reply is from authoritative nameserver (e.g., .com)
  - Reply ID that matches the request
  - Reply is for outstanding query
- (Nothing about content though …)

DNS Spoofing (6)

2. How can Trudy supply a fake DNS reply that appears to be real?

- Techniques:
  - Put IP of authoritative nameserver as the source IP address
  - ID is 16 bits (64K). Send many guesses! (Or if a counter, sample to predict.)
  - Send reply right after query
- Good chance of succeeding!

DNS Spoofing (7)

3. What happens when real DNS reply shows up?

- Likely not be a problem
  - There is no outstanding query after fake reply is accepted
  - So real reply will be discarded

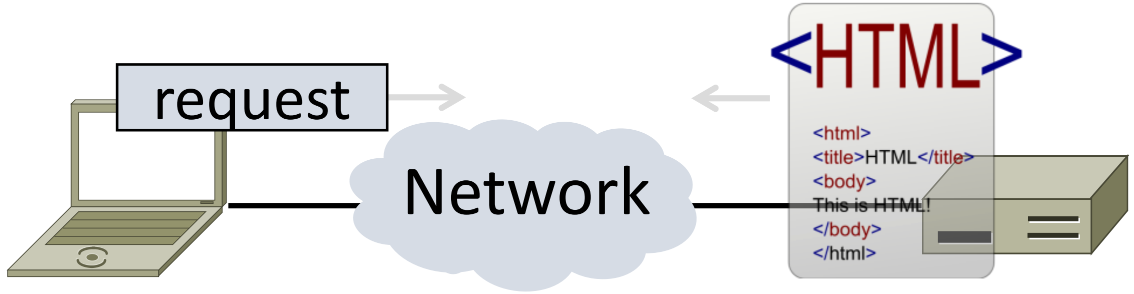
DNSSEC (DNS Security Extensions)

- Extends DNS with new record types
  - RRSIG for digital signatures of records
  - DNSKEY for public keys for validation
  - DS for public keys for delegation
  - First version in ‘97, revised by ’05
- Deployment requires software upgrade at both client and server
  - Root servers upgraded in 2010
  - Followed by uptick in deployment

HTTP

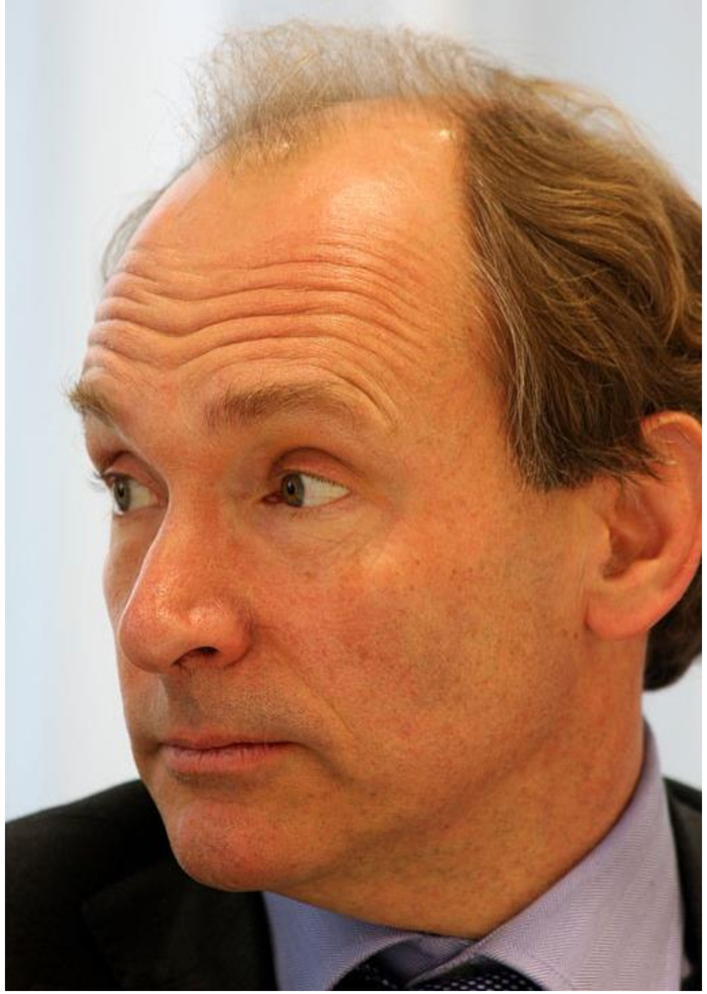
HTTP, (HyperText Transfer Protocol)

- Basis for fetching Web pages

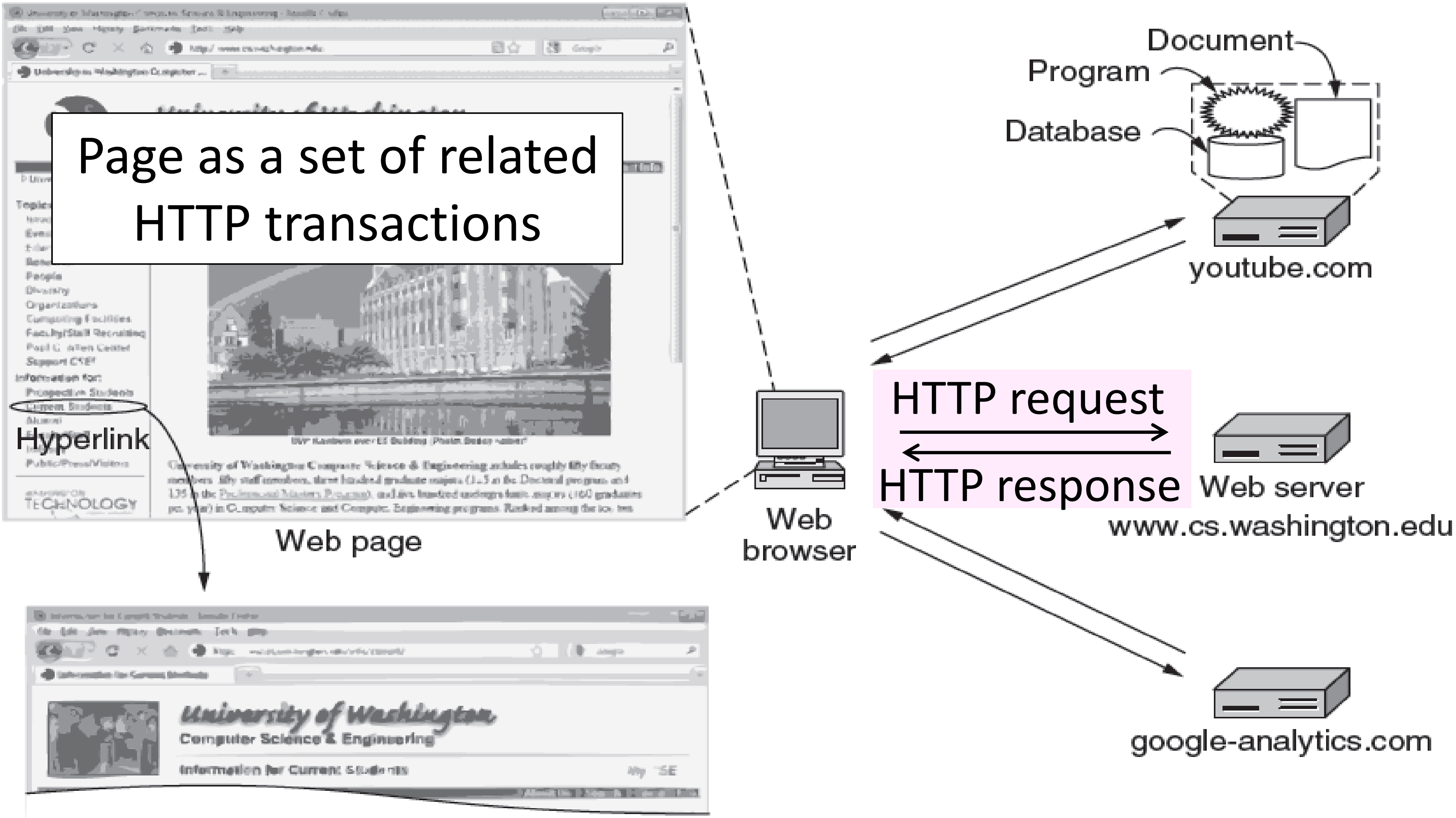


Sir Tim Berners-Lee (1955–)

- 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 …



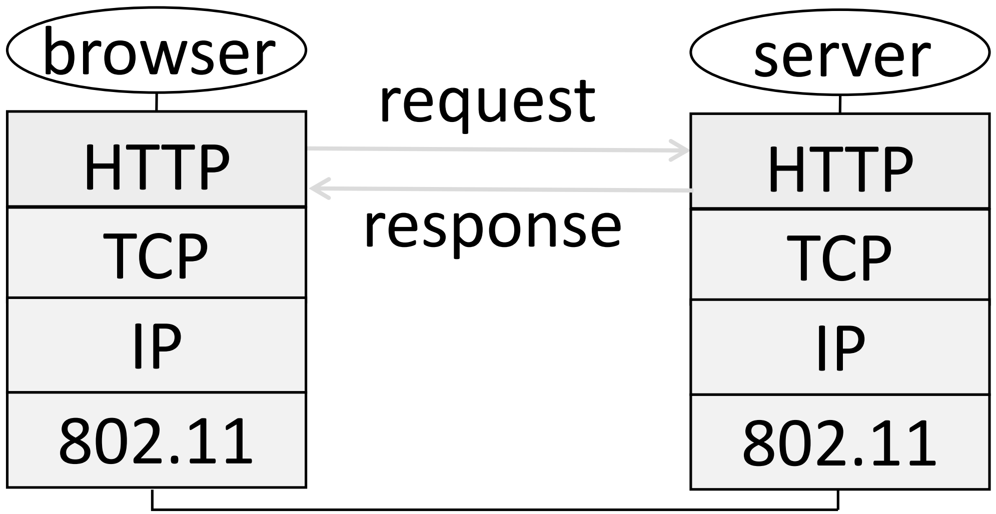
Web Context



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

Web Protocol Context

- HTTP is a request/response protocol for fetching Web resources
  - Runs on TCP, typically port 80
  - Part of browser/server app



Fetching a Web page with HTTP

- Start with the page URL:

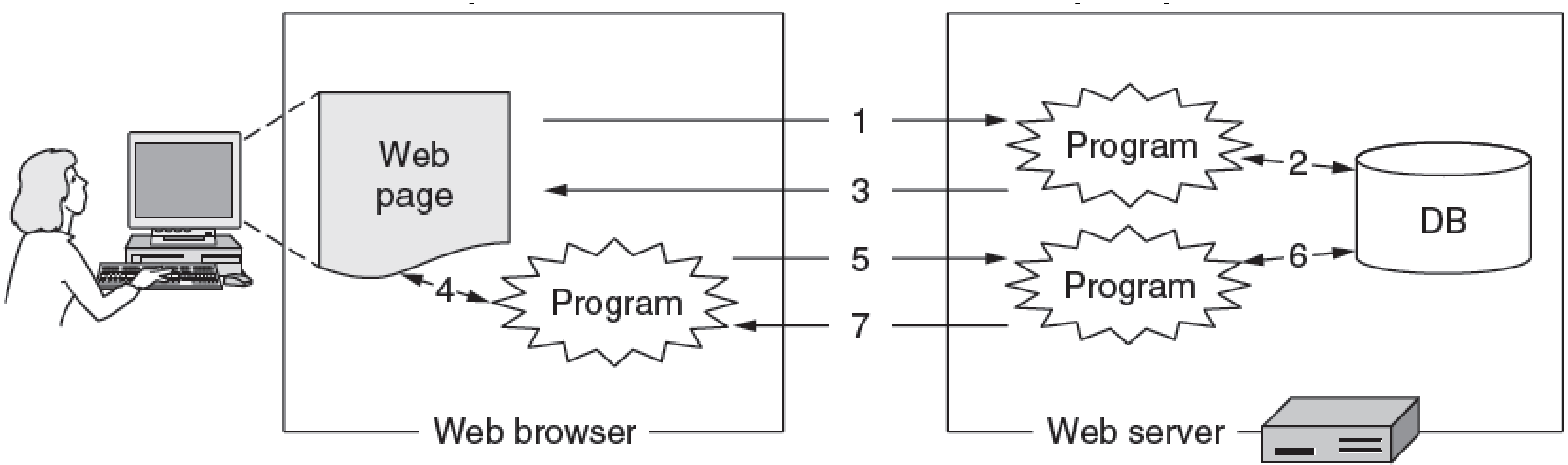
  $$\underbrace{\texttt{http:}}_{\text{Protocol}}//\underbrace{\texttt{en.wikipedia.org}}_{\text{Server}}/\underbrace{\texttt{wiki/Vegemite}}_{\text{Page on server}}$$

- 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

ADD DOM IN FUTURE

Static vs Dynamic Web pages

- Static web page is a file contents, e.g., image
- Dynamic web page is the result of program execution



HTTP Protocol

- Originally a simple protocol, with many options added over time
  - Text-based commands, headers
- Try it yourself:
  - As a "browser" fetching a URL
  - Run "telnet en.wikipedia.org 80"
  - Type "GET /wiki/Vegemite HTTP/1.0" 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

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

Fetch page

Upload data

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

Representational State Transfer (REST)

- Moving HTTP beyond websites
- An ideal for design of HTTP-based APIs
  - Called RESTful APIs
- 5 Core Tenants:
  - Stateless (no state on server)
  - Cachable (individual urls can be cached)
  - 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.

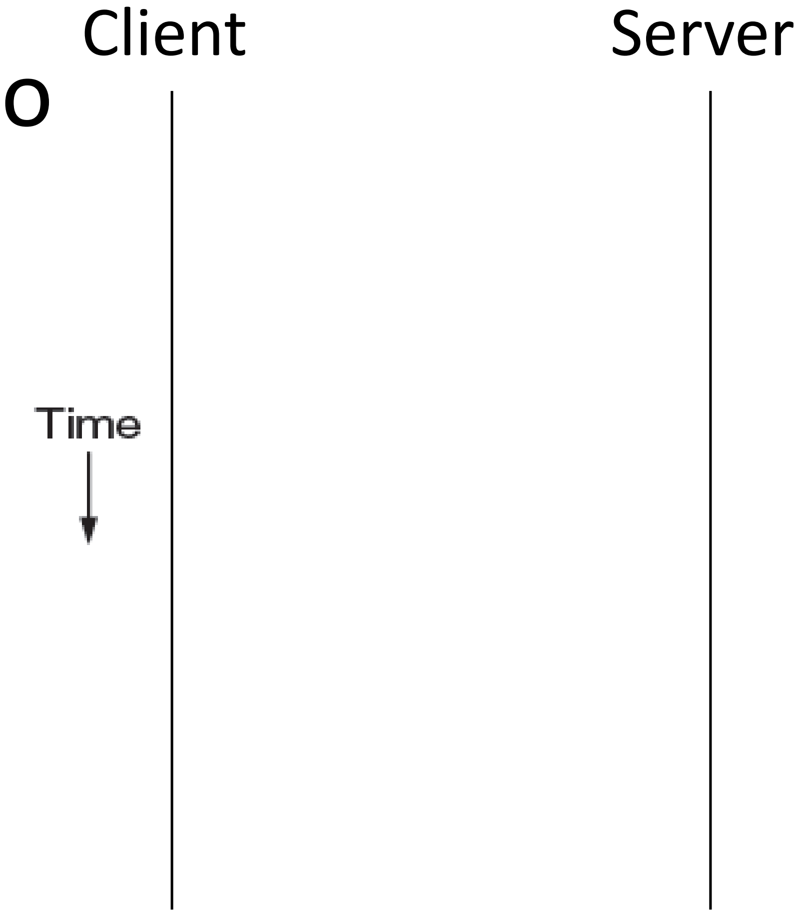
Performance

PLT (Page Load Time)

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

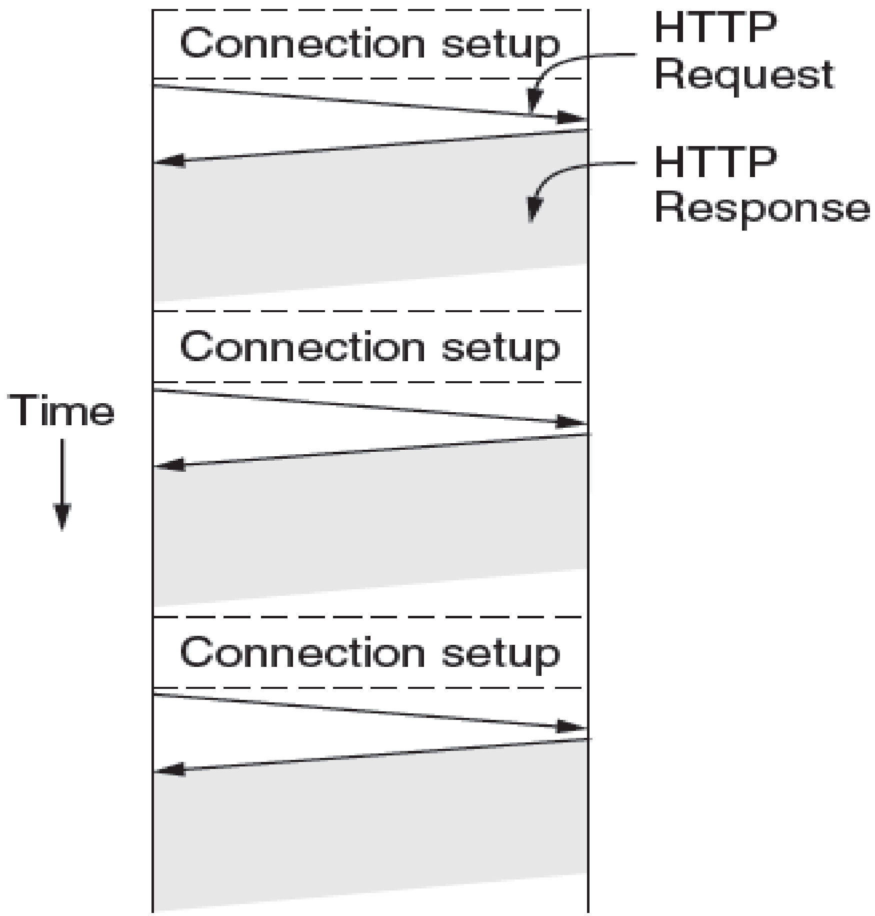
Early Performance

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



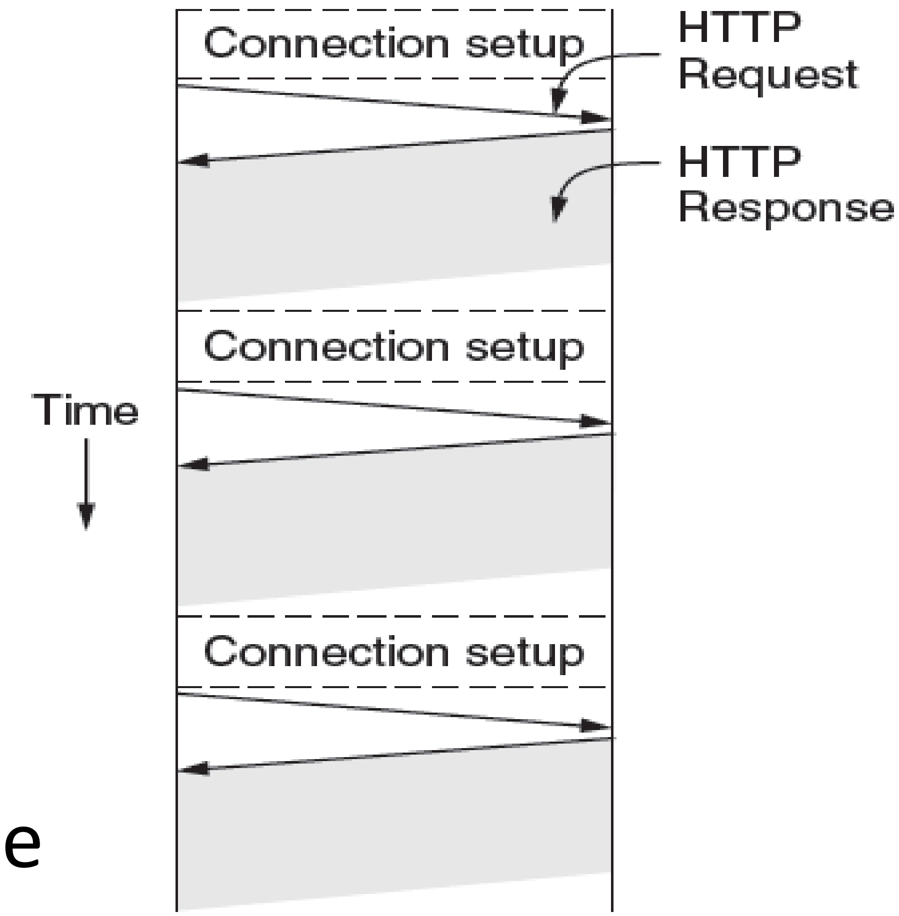
Early Performance (2)

- 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



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]

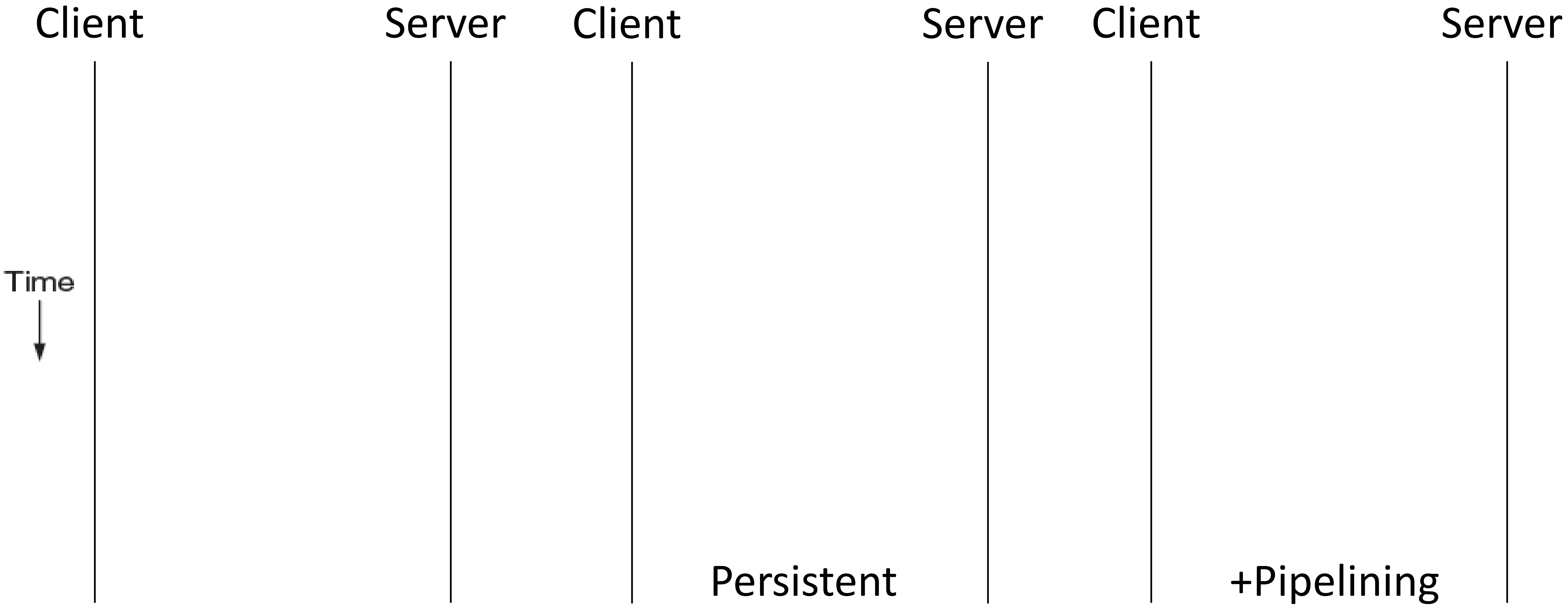
Parallel Connections

- One simple way to reduce PLT
  - Browser runs multiple (8, 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

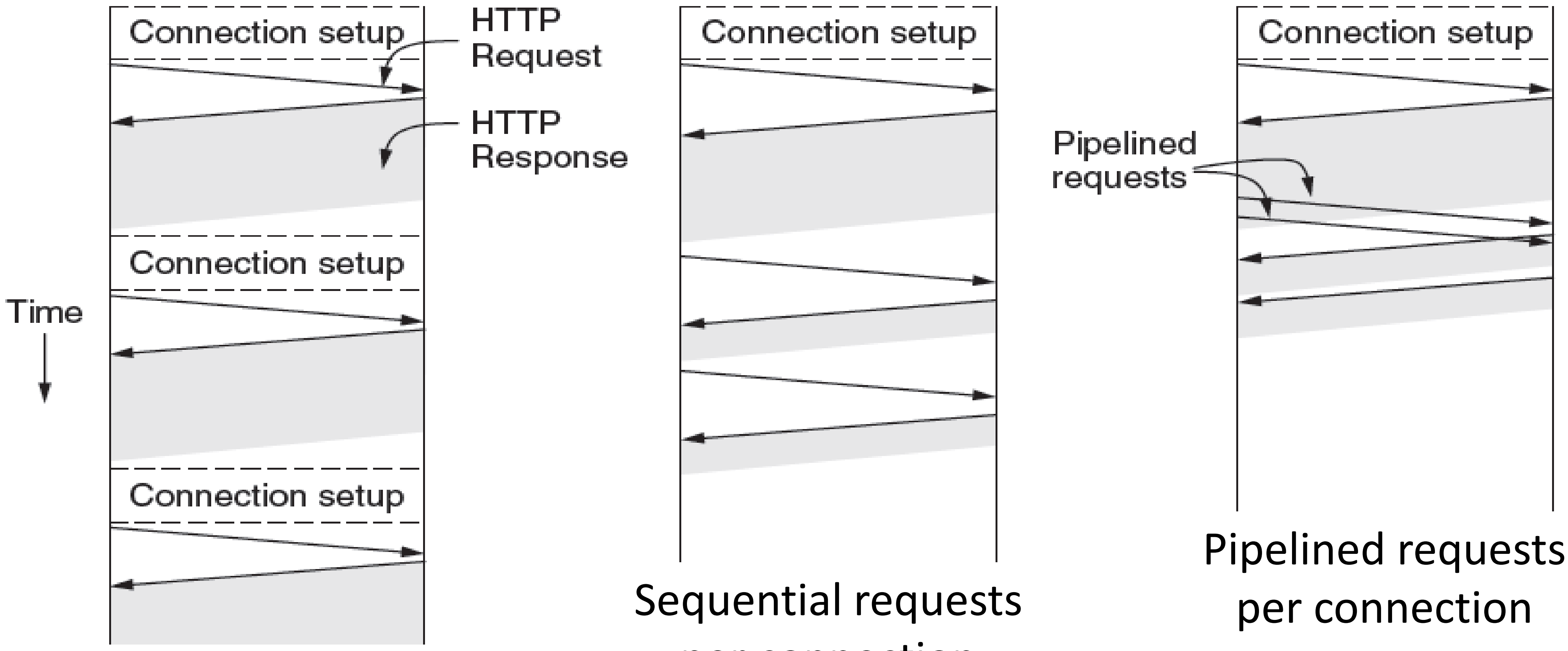
Persistent Connections

- Parallel connections compete with each other for network resources
  - 1 parallel client ≈ 8 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)



Persistent Connections (3)



One request per connection

Sequential requests per connection

Pipelined requests per connection

Persistent Connections (4)

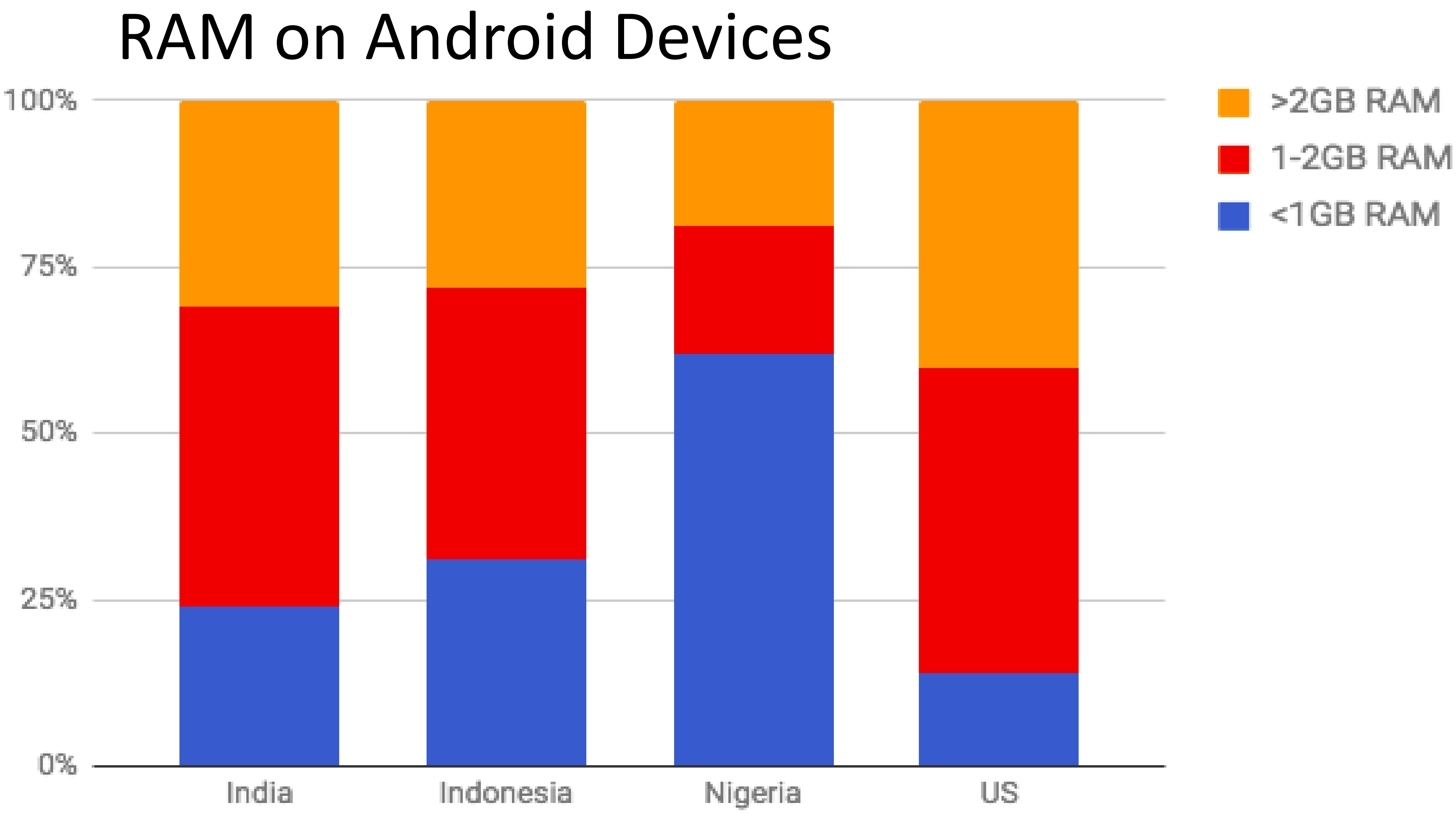
- Widely used as part of HTTP/1.1
  - Supports optional pipelining
  - PLT benefits depending on page structure, but easy on network

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)



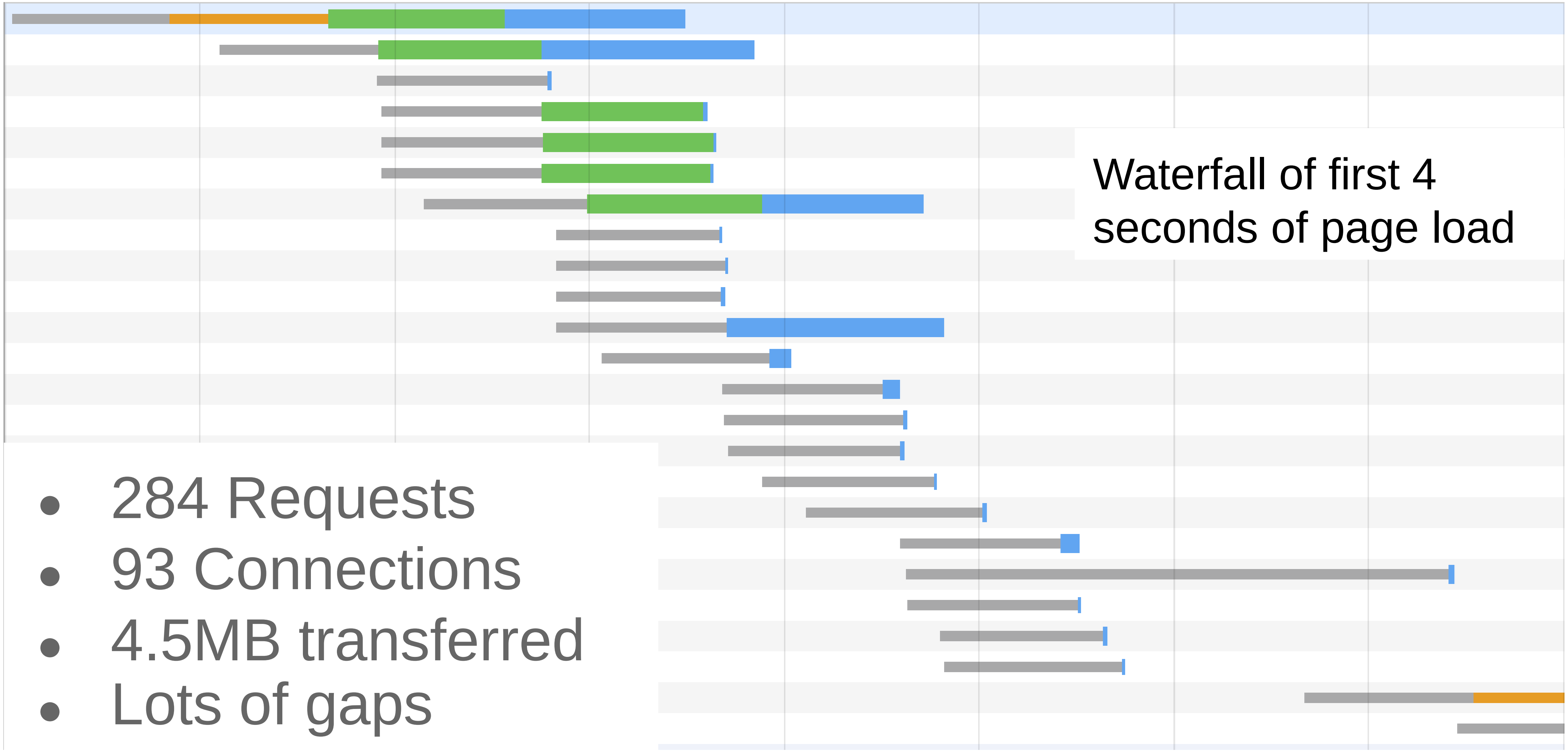
Continued Growth (3)

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)

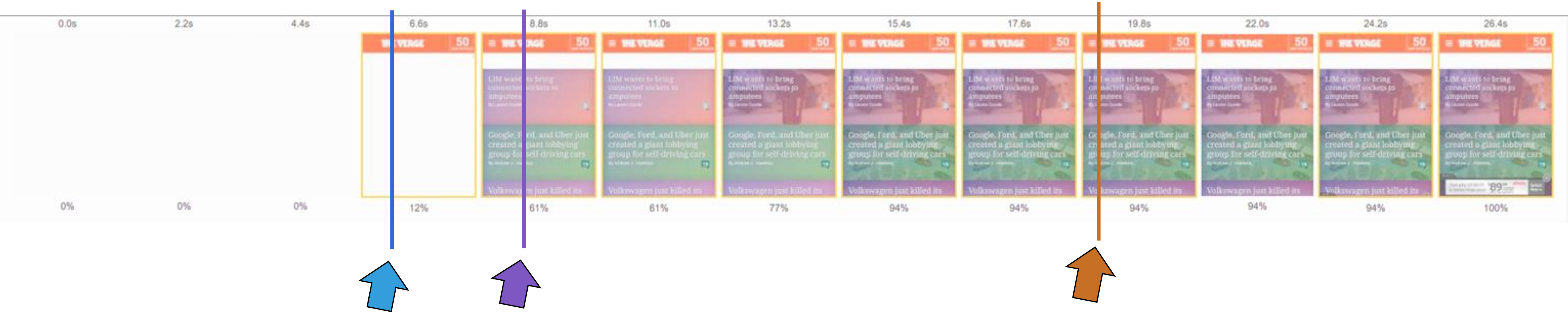
Source: Chrome logs



Waterfall of first 4 seconds of page load

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

Key user moments (Kill PLT)



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

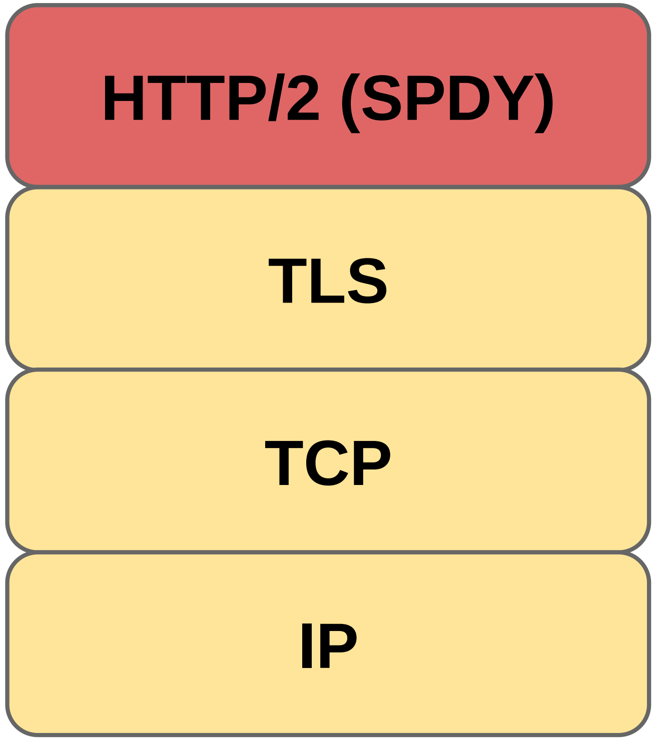
HTTP Changes

HTTP/1.0: TCP connection per request

HTTP/1.1: Persistence and pipelining

HTTP2/SPDY: Targeted performance specifically

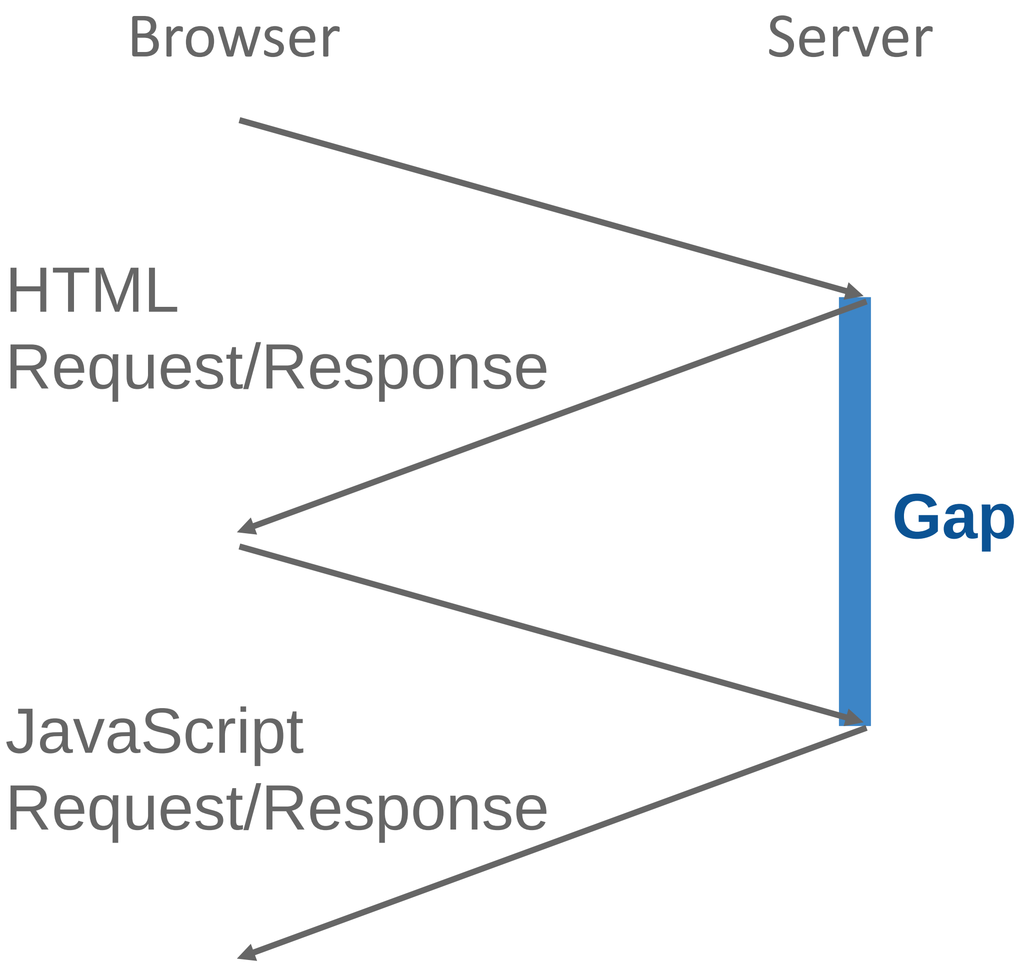
- All happens below HTTP layer
- Prioritized stream multiplexing
- Header compression
- Server push
- Started as SPDY, standardized as HTTP/2 in 2015 after ~~every possible bikeshed~~ deep discussion



Example resource loading gap

- Browser requests and receives HTML, encounters `<script src=”...”>`

- Similarly, JavaScript might src a dependent JavaScript file

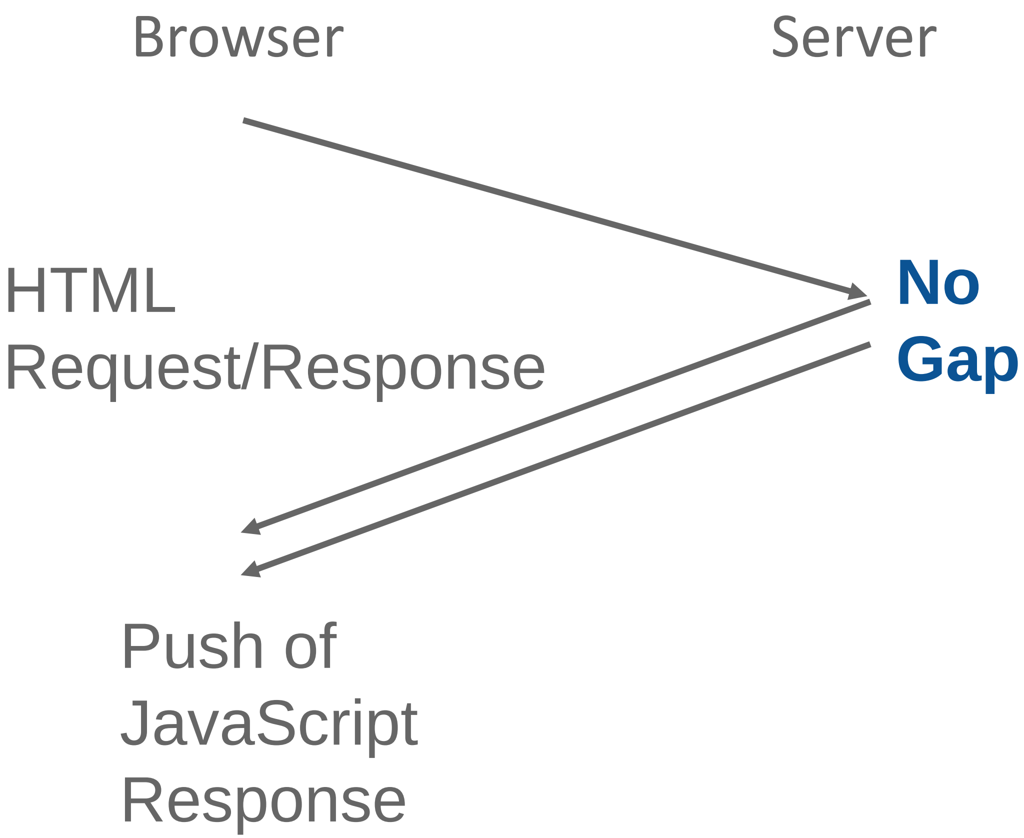


A resource loading gap

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

Or use **Link: rel=preload**

- Particularly useful for **hidden render blocking resources (HRBRs)**

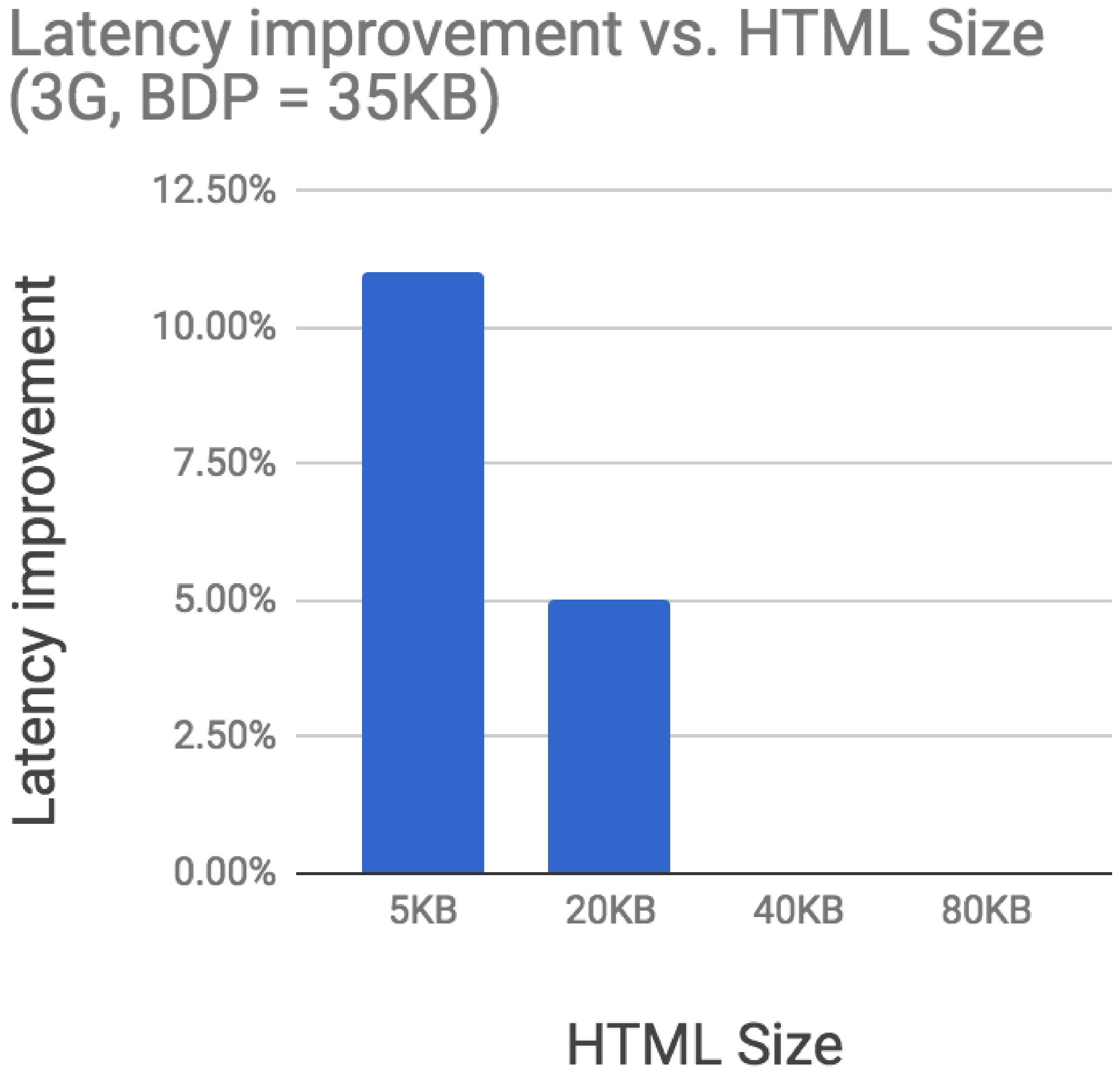


Simple server push lab experiment

Result: No benefit when HTML size > BDP

Why? No gap even without push.

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



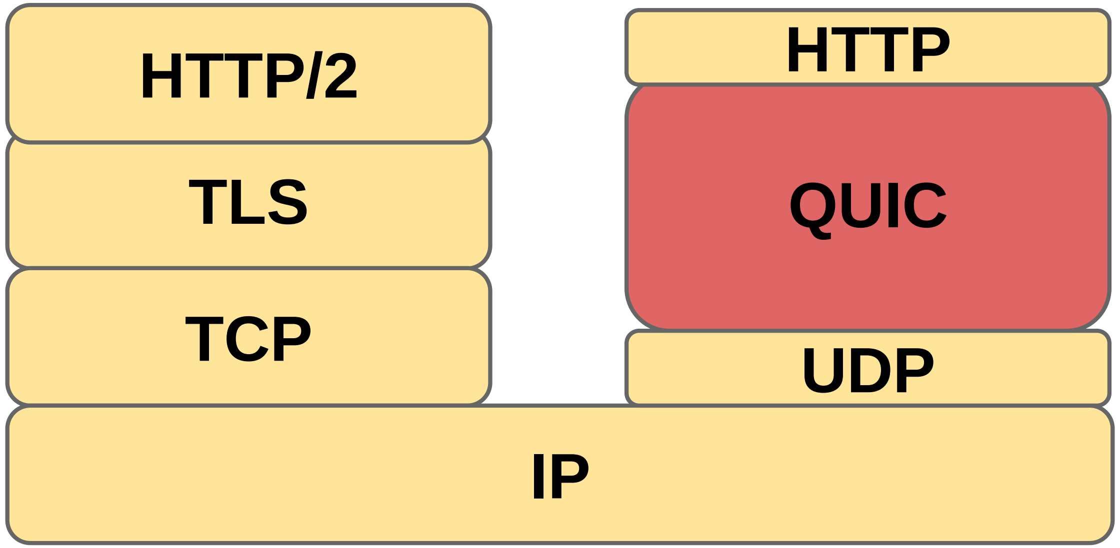
QUIC

Goal: make HTTPS transport **even faster!**

Deployed at Google starting **2014**

Runs over **UDP**

IETF working group formed in 2016

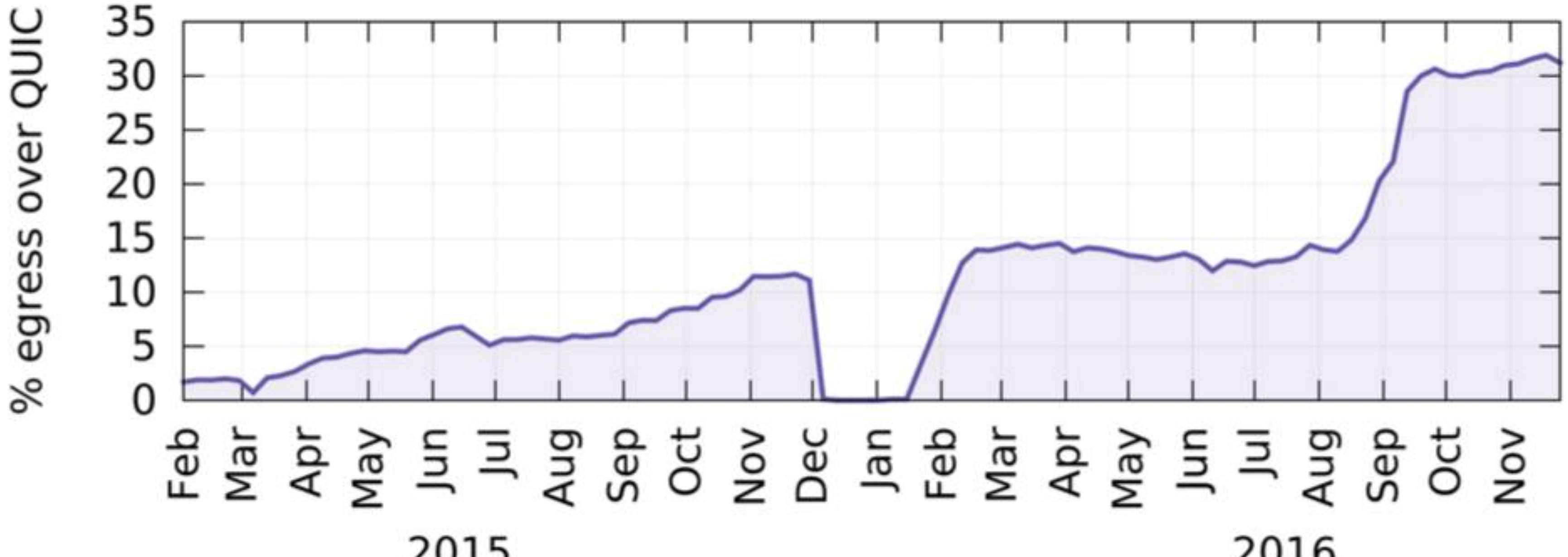


QUIC summary

Makes HTTPS faster, particularly in the tail

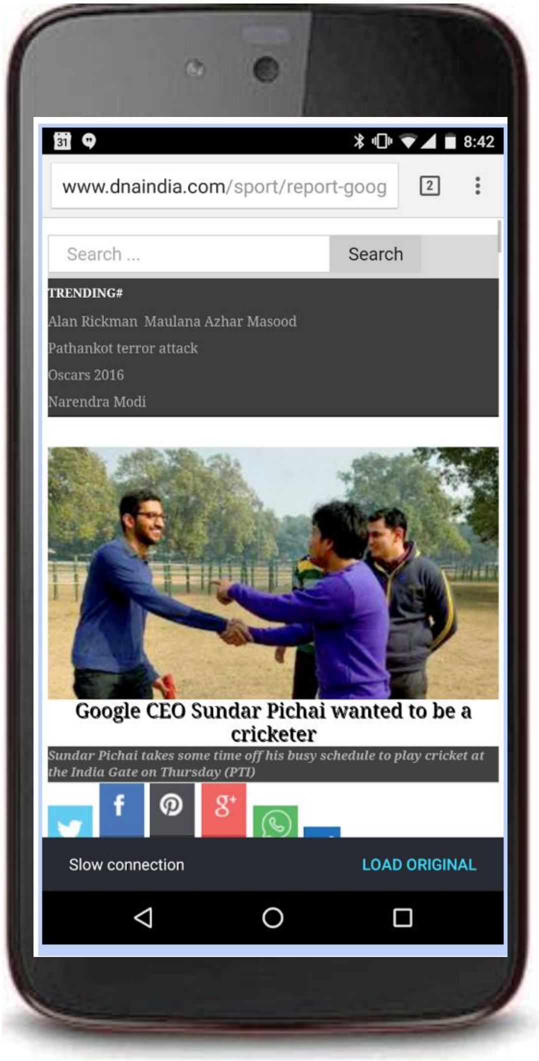
35% of Google's egress traffic (7% of the Internet)

Deploying at Google was 3+ years of hard work



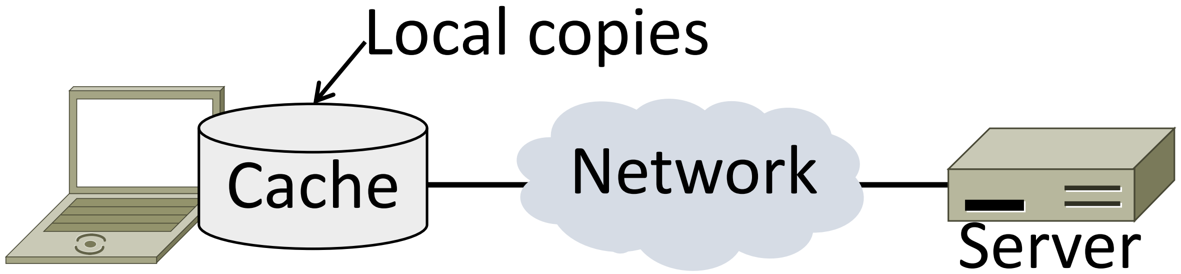
Going Farther

- Flywheel **proxy** service
  - **Compresses HTTP pages** by 60%.
  - **Transcodes** to **WebP**, **WebM**, **Brotli**
    Uses **HTTP/2** and **QUIC**
- Render the page on the server
  - 50% speedup, >90% compression
  - Trades **fidelity loss** for **speed**, so we do this only on **very slow networks**



Web Caching

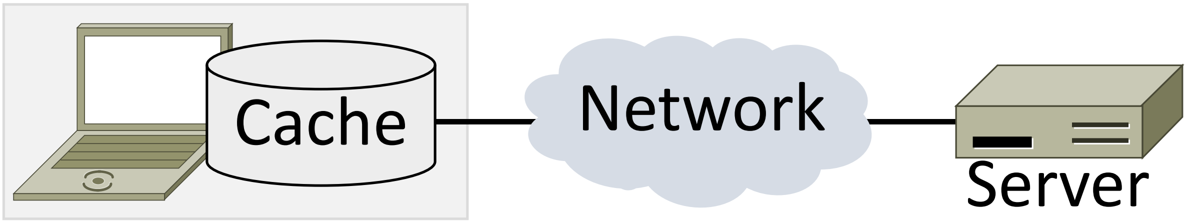
- Users often revisit web pages
  - Big win from reusing local copy!
  - This is caching



- Key question:
  - When is it OK to reuse local copy?

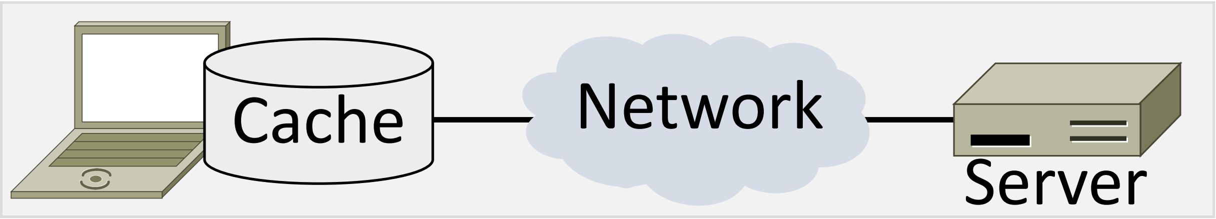
Web Caching (2)

- Locally determine copy is still valid
  - Based on expiry information such as “Expires” header from server
  - Or use a heuristic to guess (cacheable, freshly valid, not modified recently)
  - Content is then available right away



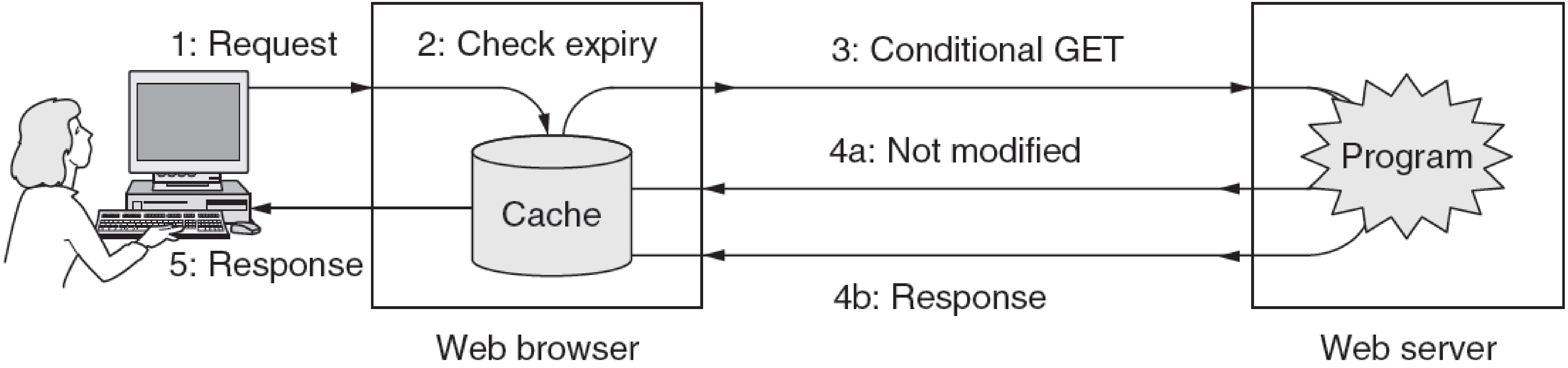
Web Caching (3)

- Revalidate copy with remote server
  - Based on timestamp of copy such as “Last-Modified” header from server
  - Or based on content of copy such as “Etag” server header
  - Content is available after 1 RTT



Web Caching (4)

- Putting the pieces together:

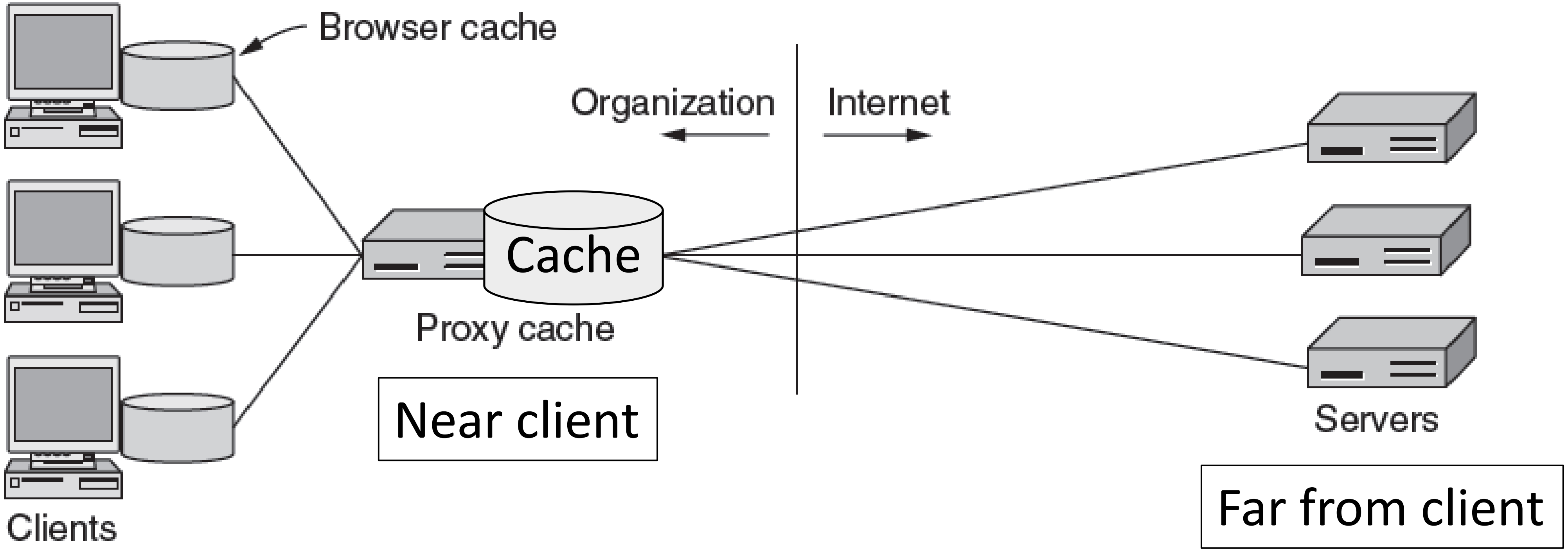


Web Proxies

- Place intermediary between pool of clients and external web servers
  - Benefits for clients include caching and security checking
  - Organizational access policies too!
- Proxy caching
  - Clients benefit from larger, shared cache
  - Benefits limited by secure / dynamic content, as well as "long tail"

Web Proxies (2)

- Clients contact proxy; proxy contacts server

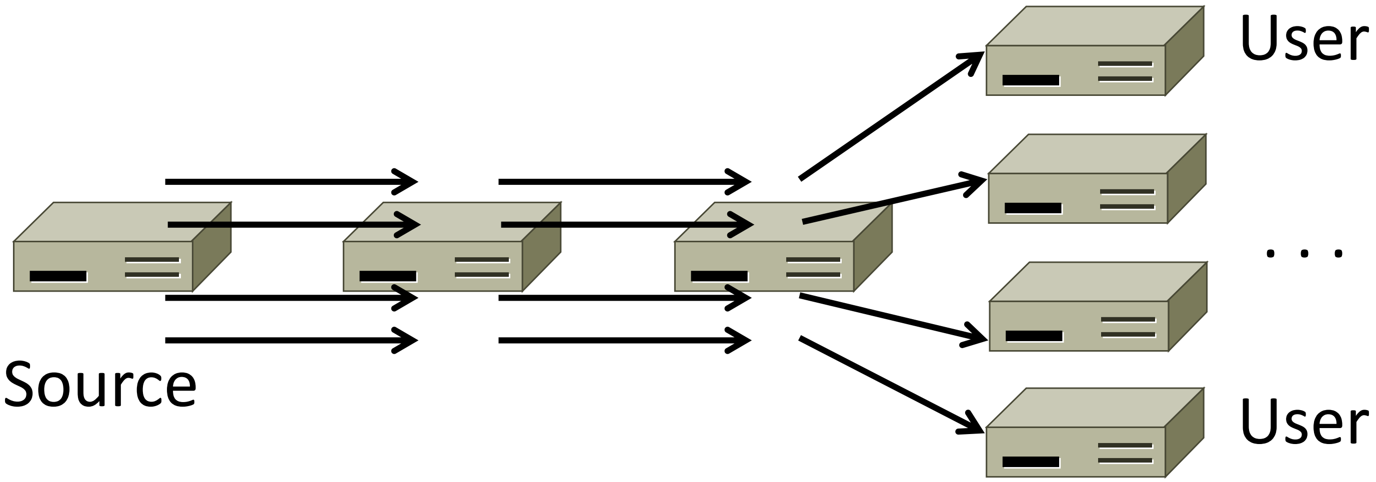


Content Delivery Networks

- As the web took off in the 90s, traffic volumes grew and grew. This:
    1. Concentrated load on popular servers
    2. Led to congested networks and need to provision more bandwidth
    3. Gave a poor user experience
- Idea:
    - Place popular content near clients
    - Helps with all three issues above

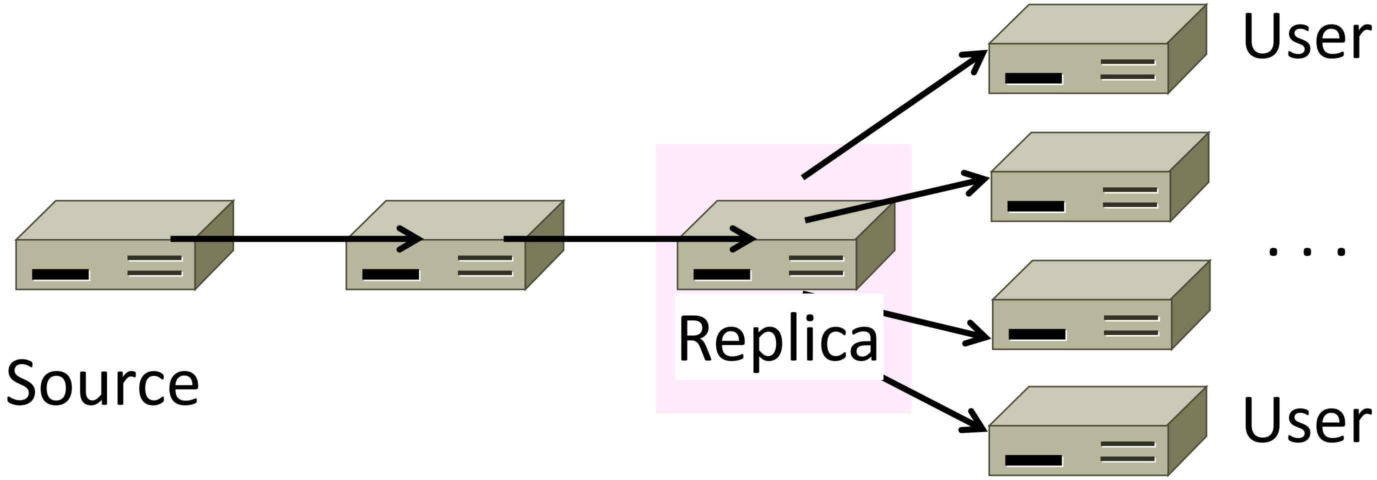
Before CDNs

- Sending content from the source to 4 users takes 4 x 3 = 12 "network hops" in the example



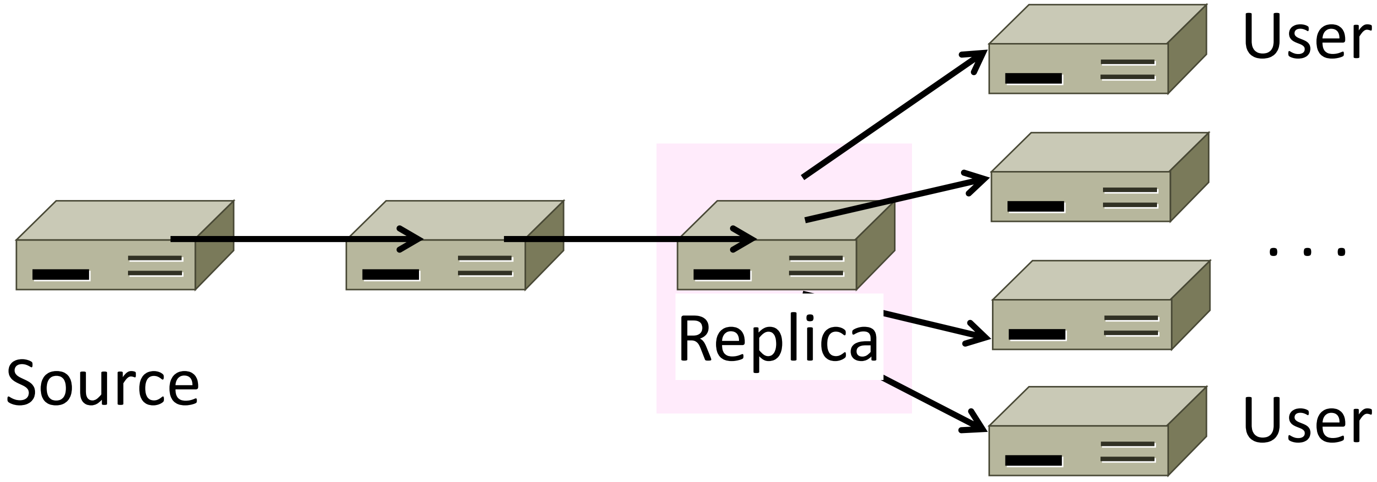
After CDNs

- Sending content via replicas takes only 4 + 2 = 6 "network hops"



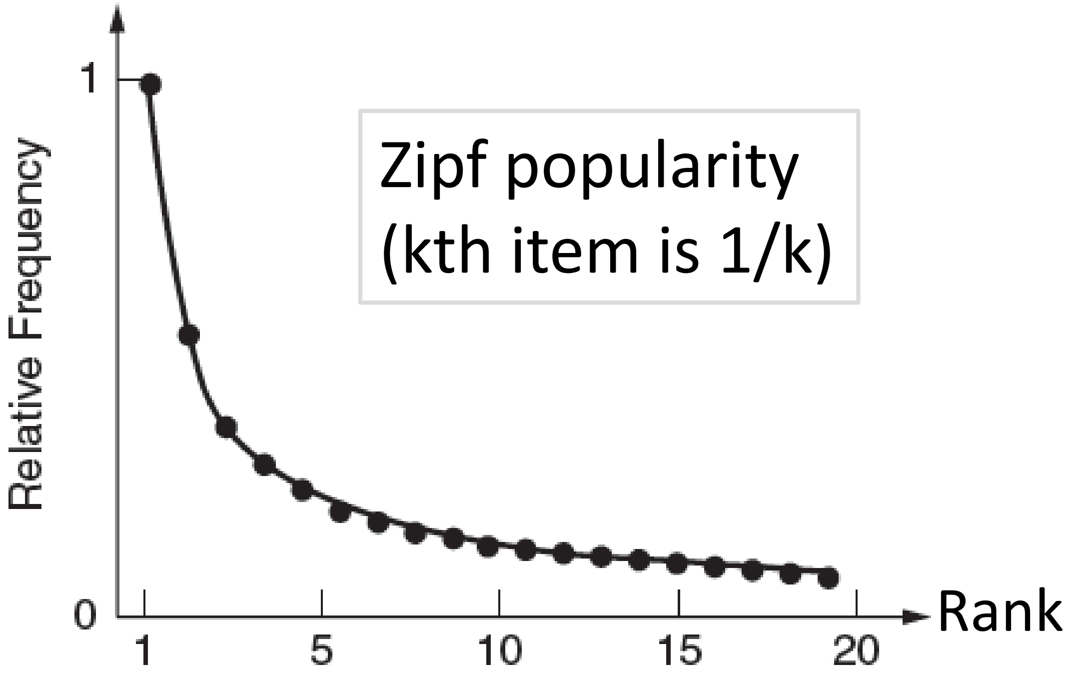
After CDNs (2)

- Benefits assuming popular content:
  - Reduces server, network load
  - Improves user experience (PLT)



Popularity of Content

- Zipf's Law: few popular items, many unpopular ones; both matter



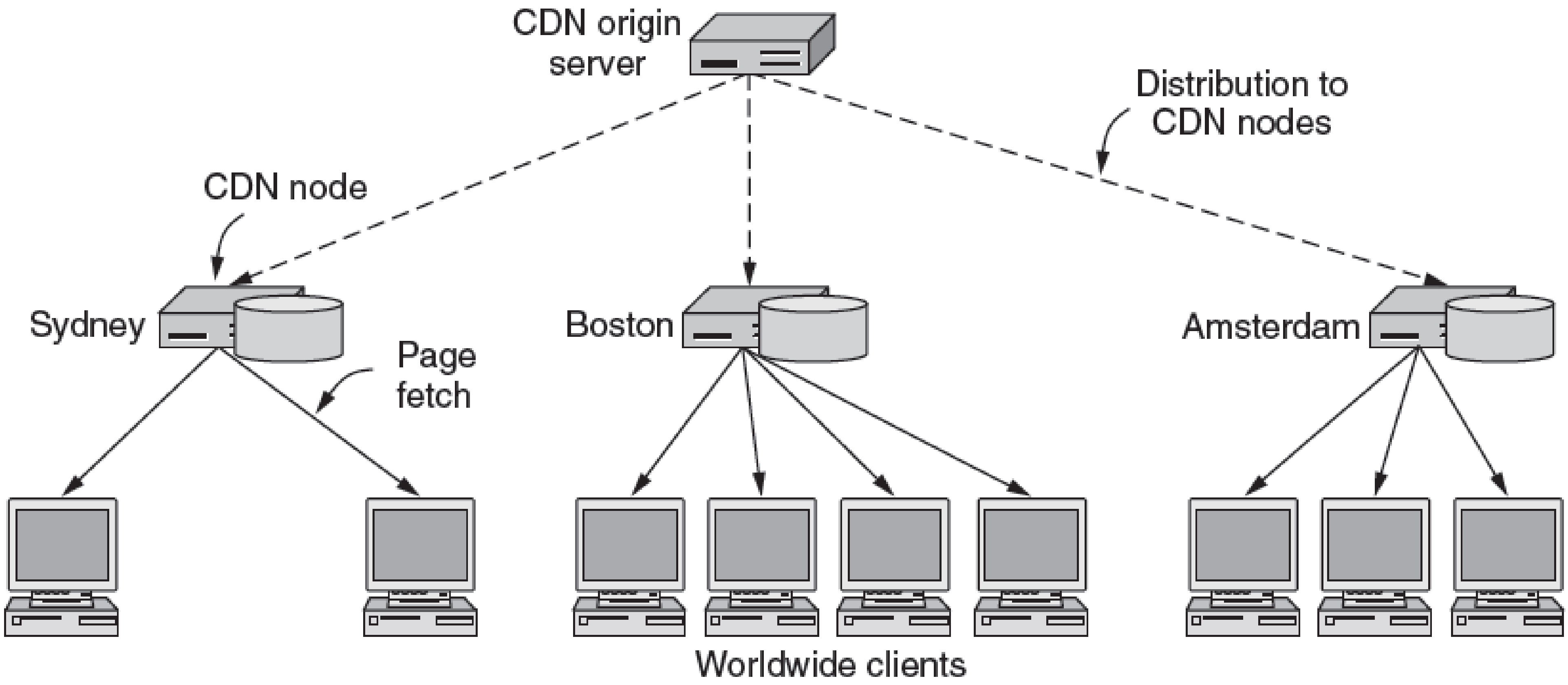
George Zipf (1902-1950)

Source: Wikipedia

How to place content near clients?

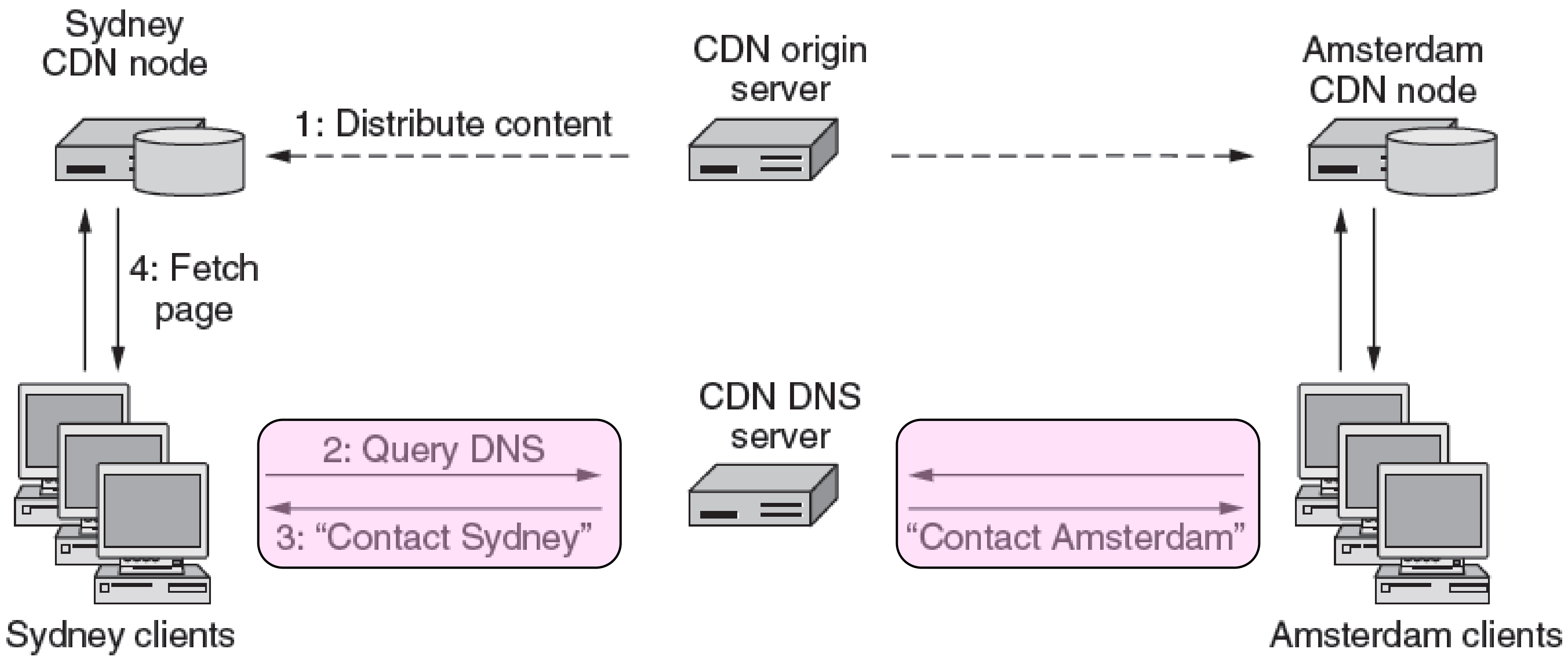
- Use browser and proxy caches
  - Helps, but limited to one client or clients in one organization
- Want to place replicas across the Internet for use by all nearby clients
  - Done by clever use of DNS

Content Delivery Network



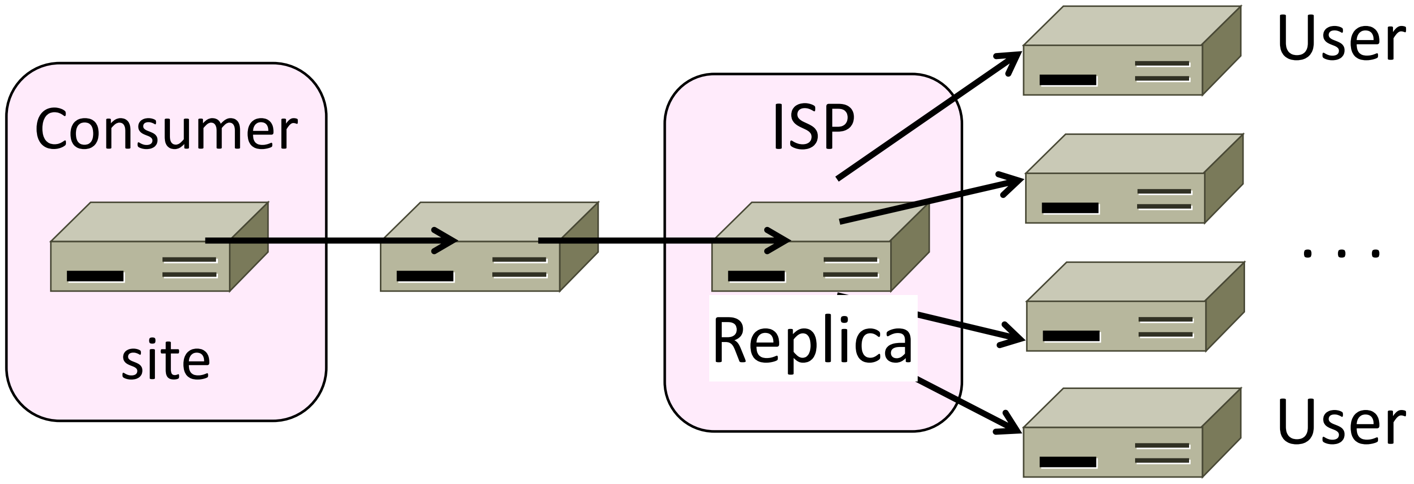
Content Delivery Network (2)

- DNS gives different answers to clients
  - Tell each client the nearest replica (map client IP)



Business Model

- Clever model pioneered by Akamai
  - Placing site replica at an ISP is win-win
  - Improves site experience and reduces ISP bandwidth usage



CDNs - Issues

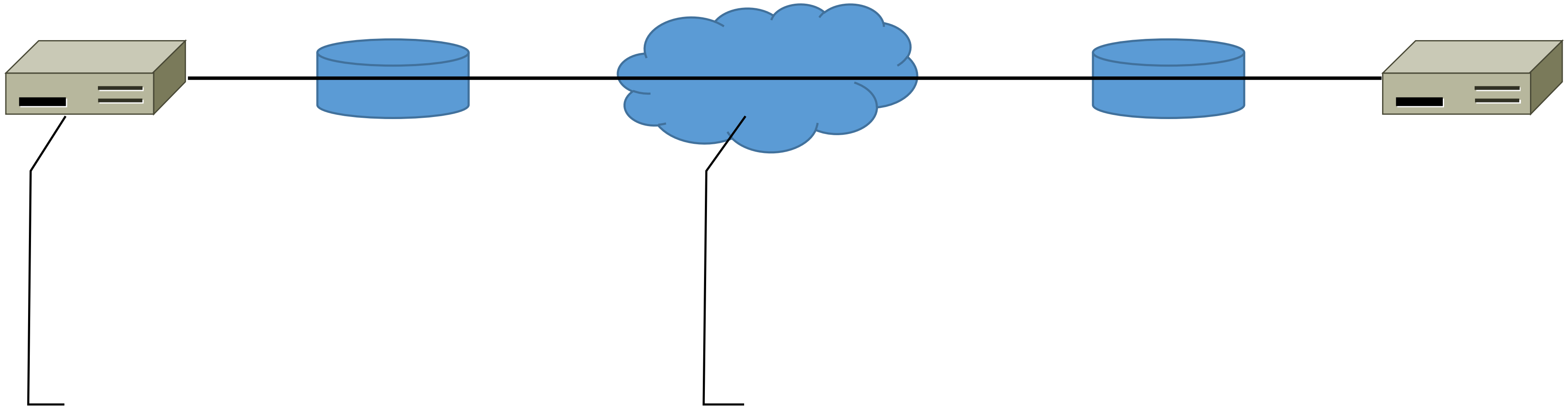
- Security
  - What about private information?
  - How to cache/forward encrypted content
    - Basically can’t!
- Net neutrality
  - I.org, FreeBasics -> Basically CDNs
    - But for reasons of price, not efficiency
  - Who decides who gets to place CDNs?

End-to-End principle

End-to-end Principle

- Broad networking principle
  - French CYCLADES network (after ARPA) first to implement
- Idea: The network cannot be trusted. Do it yourself.
  - “"Reliability and raw error rates are secondary. The network must be built with the expectation of heavy damage anyway. Powerful error removal methods exist.”

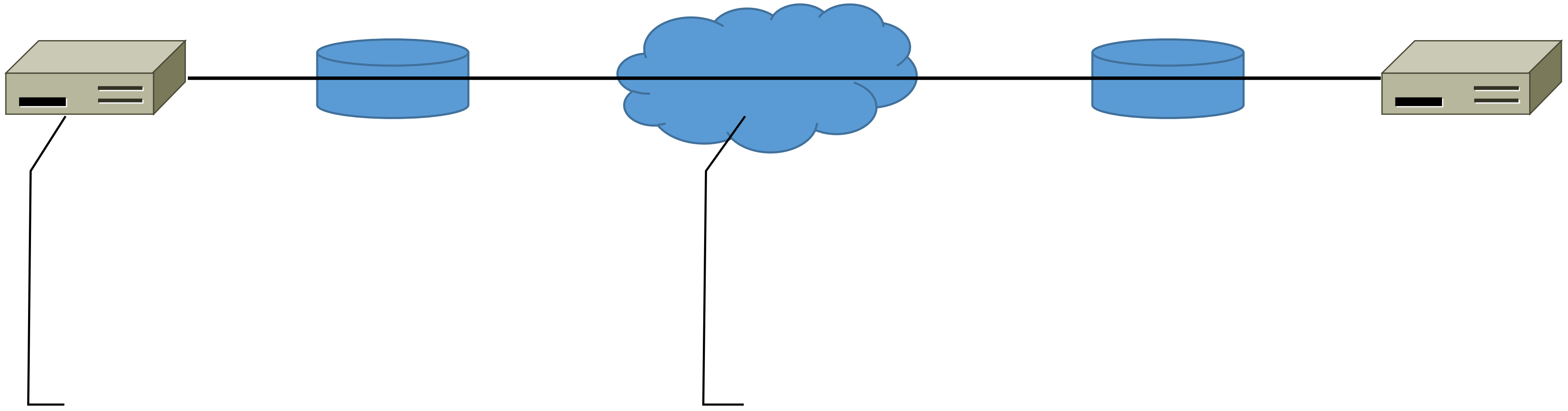
E2E Example: Error-correcting codes



IP:
Host detects
errors

802.11:
Link detects errors

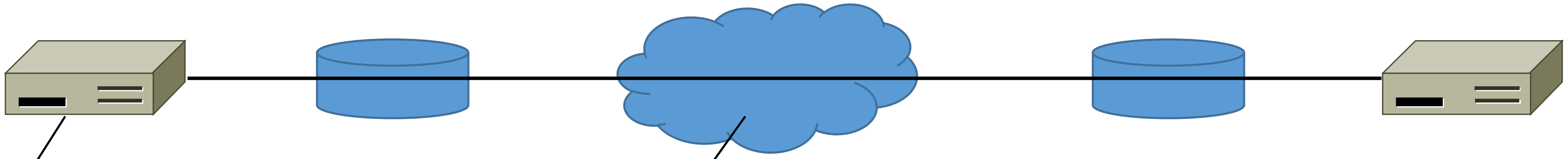
E2E Example: ARQ



TCP:
Host retransmits
on failure

802.11:
Link detects drops
and retransmits

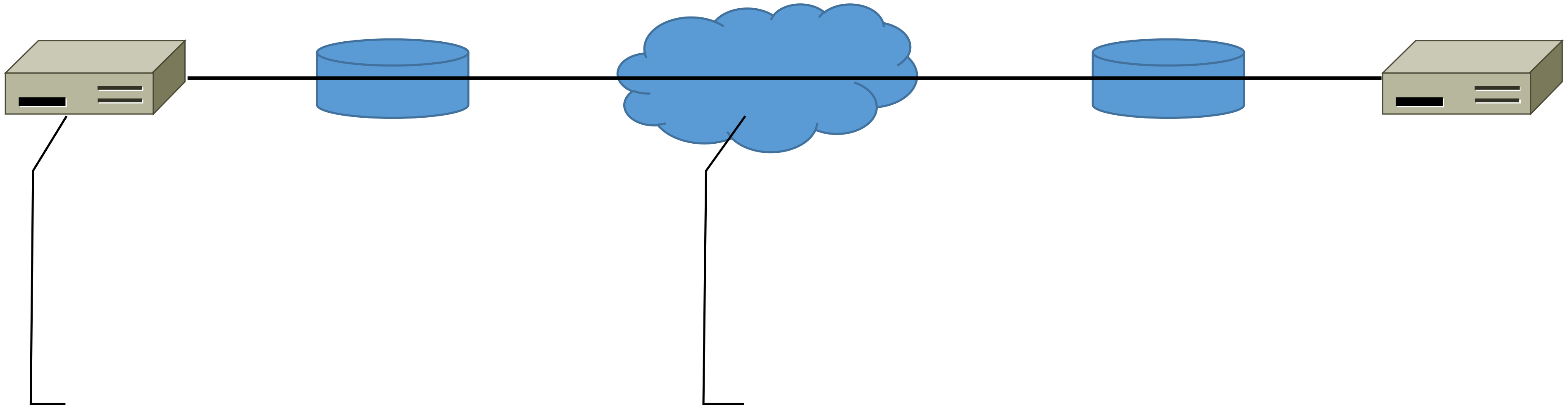
E2E Example: In-order delivery



TCP:
Host enforces in-order delivery

SS5:
Network enforces in-order delivery

E2E Example: Security



SSL:
Host encrypts content

GSM:
Network encrypts content

End-to-End

- What are the limitations of the End-to-End principle?