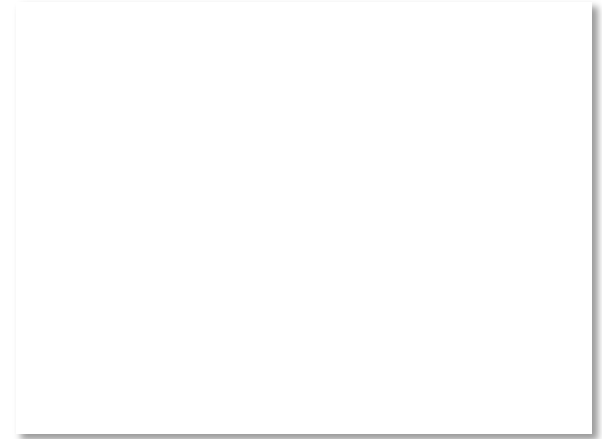
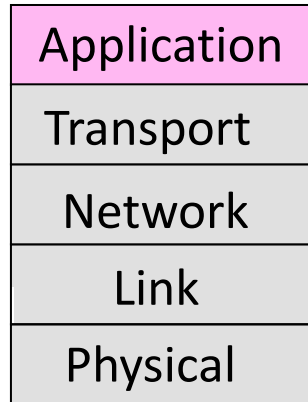


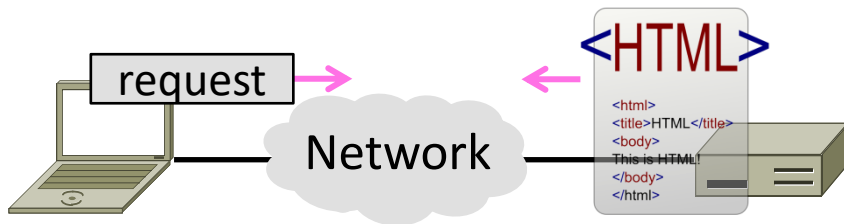
Where we are in the Course

- Starting the Application Layer!
 - Builds distributed “network services” (DNS, Web) on Transport services



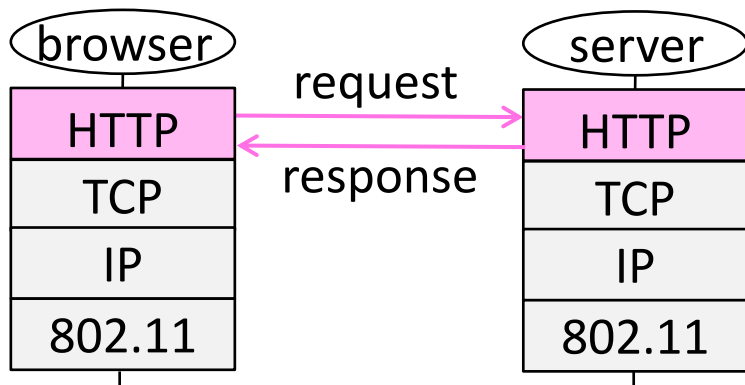
Topic

- HTTP, (HyperText Transfer Protocol)
 - Basis for fetching Web pages



Web Protocol Context

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



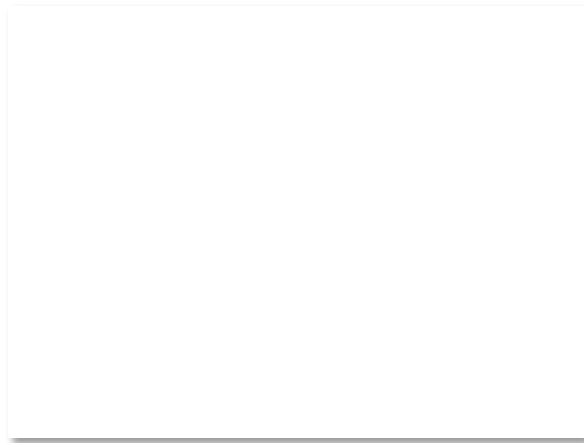
Fetching a Web page with HTTP

- Start with the page URL:

`http://en.wikipedia.org/wiki/Vegemite`

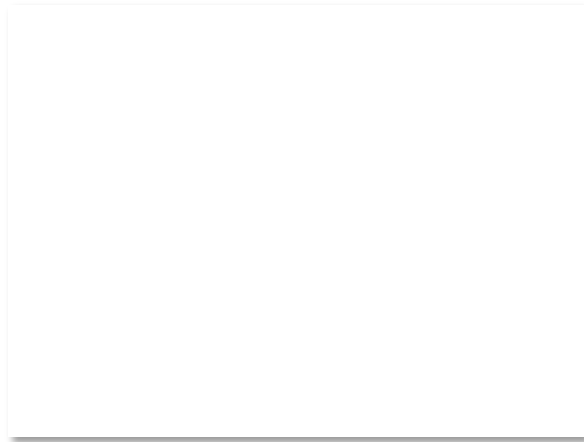
Protocol Server Page on server

- 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



HTTP Protocol

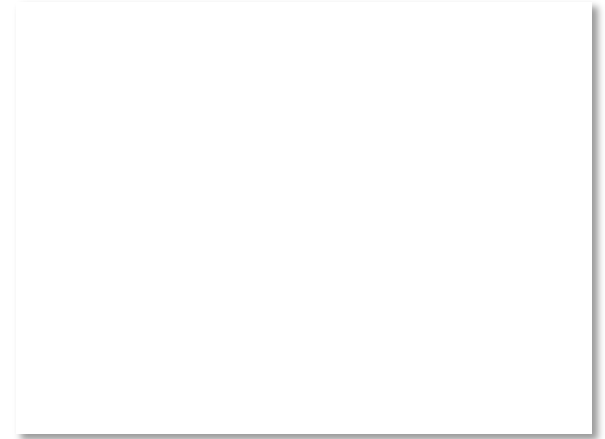
- Originally a simple protocol, with many options added over time
 - Text-based commands, headers
- Try it yourself:
 - As a “browser” fetching a URL
 - Run “telnet en.wikipedia.org 80”
 - Type “GET /wiki/Vegemite HTTP/1.0” to server followed by a blank line
 - Server will return HTTP response with the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



HTTP Protocol (2)

- Commands used in the request

	Method	Description
Fetch page →	GET	Read a Web page
	HEAD	Read a Web page's header
Upload data →	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



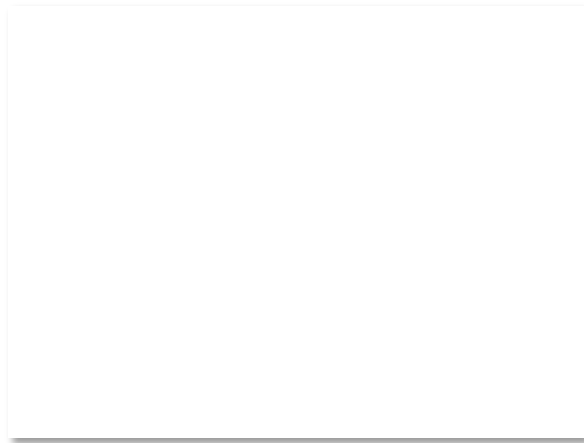
HTTP Protocol (3)

- 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

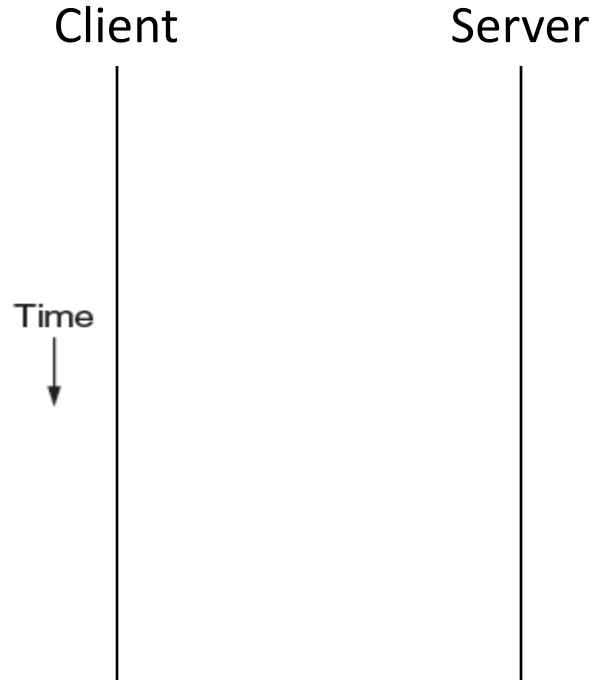
PLT (Page Load Time)

- 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



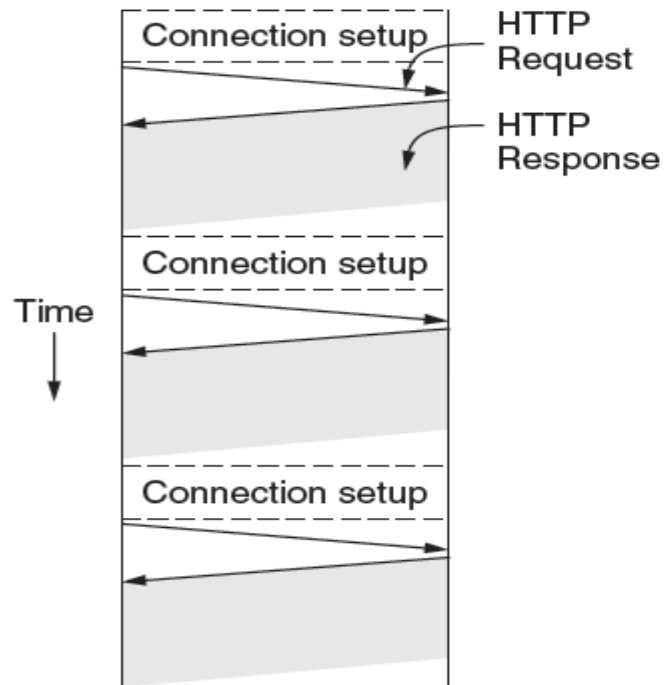
Early Performance

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



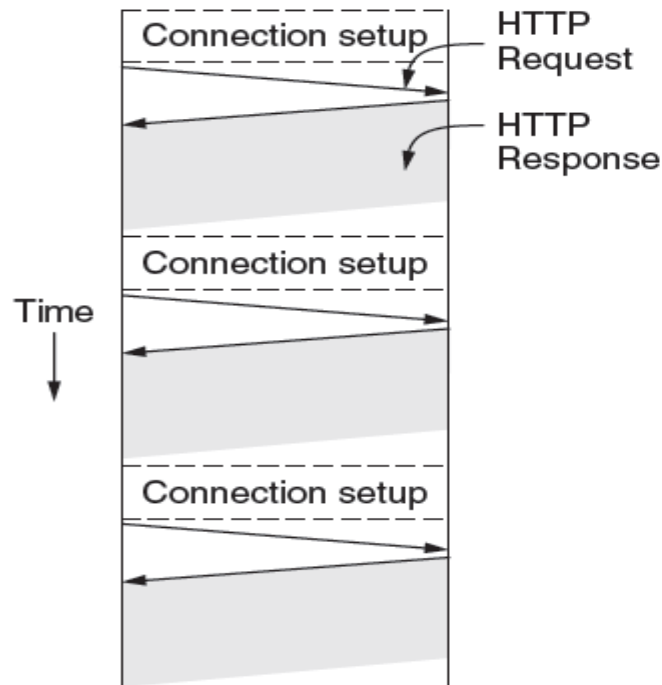
Early Performance (2)

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



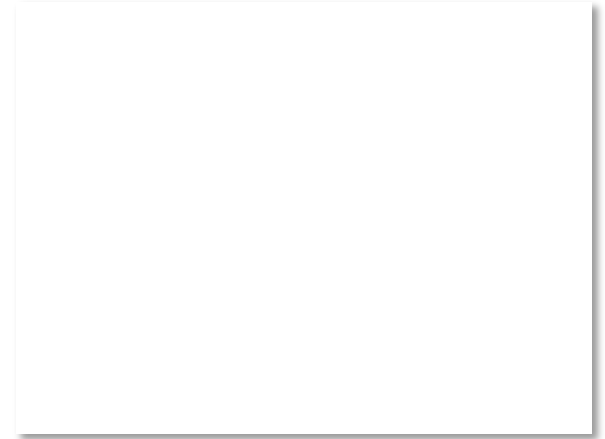
Early Performance (3)

- 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



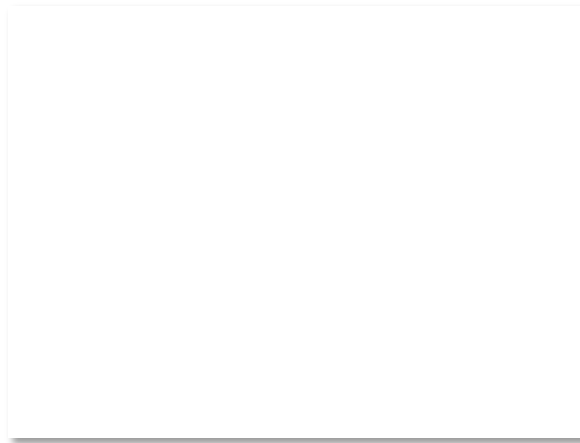
Parallel Connections

- 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

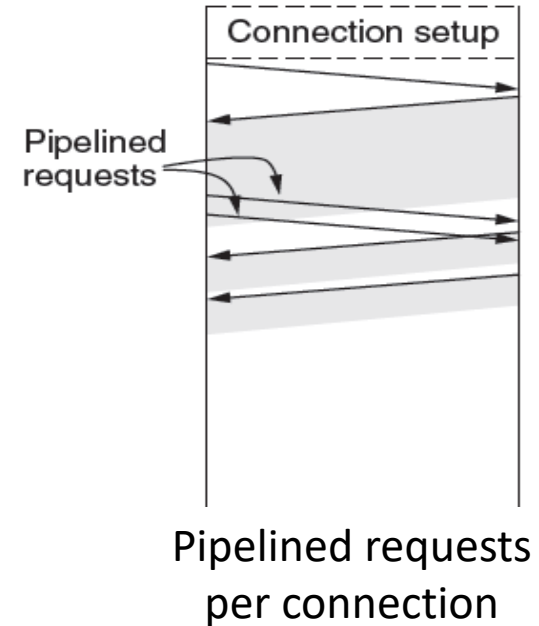
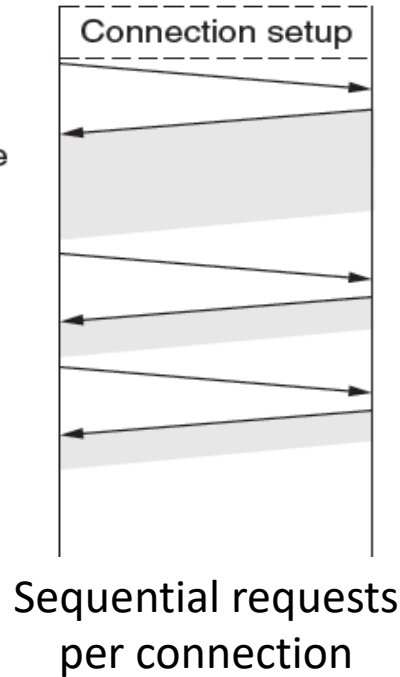
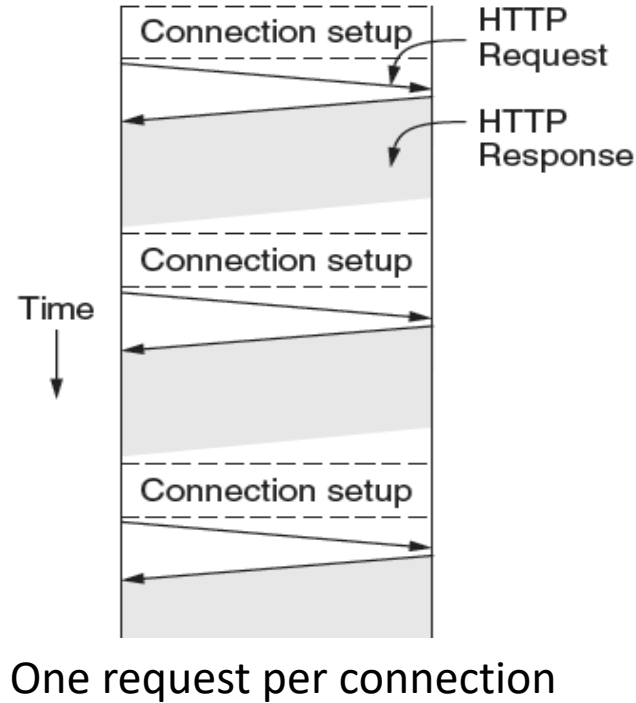


Persistent Connections

- Parallel connections compete with each other for network resources
 - 1 parallel client \approx 8 sequential clients?
 - Exacerbates network bursts, and loss
- Persistent connection alternative
 - Make 1 TCP connection to 1 server
 - Use it for multiple HTTP requests

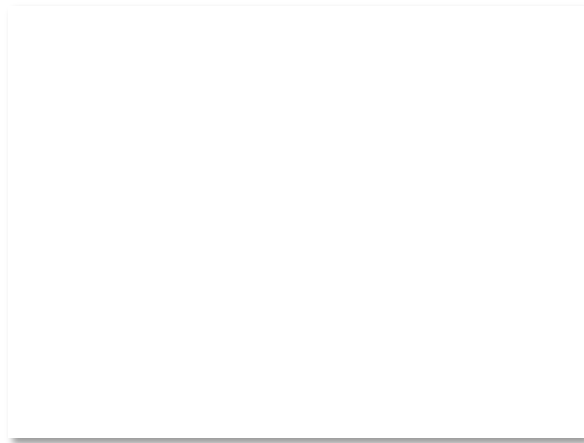


Persistent Connections (3)



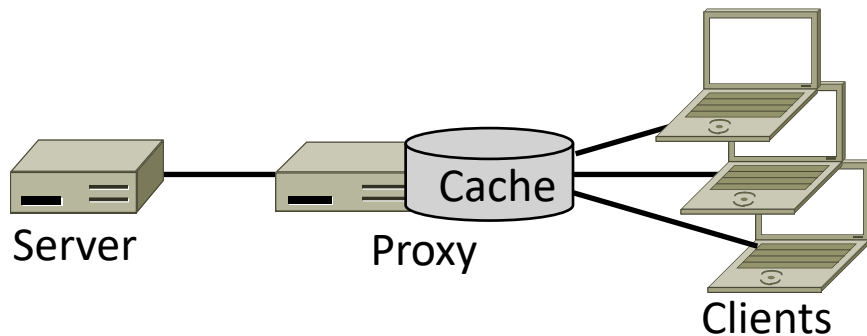
Persistent Connections (4)

- Widely used as part of HTTP/1.1
 - Supports optional pipelining
 - PLT benefits depending on page structure, but easy on network
- Issues with persistent connections
 - How long to keep TCP connection?
 - Can it be slower? (Yes. But why?)



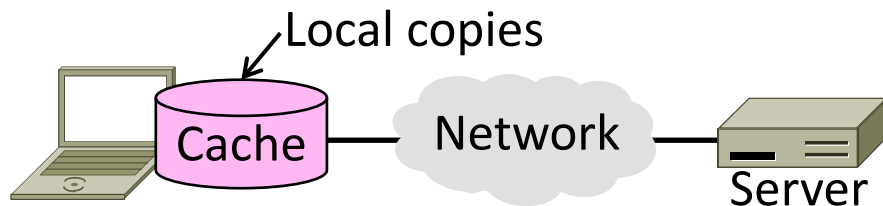
Topic

- HTTP caching and proxies
 - Enabling content reuse



Web Caching

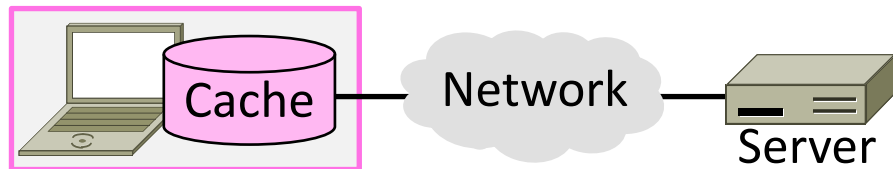
- Users often revisit web pages
 - Big win from reusing local copy!
 - This is caching



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

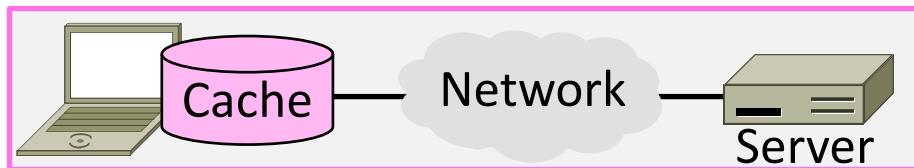
Web Caching (2)

- 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



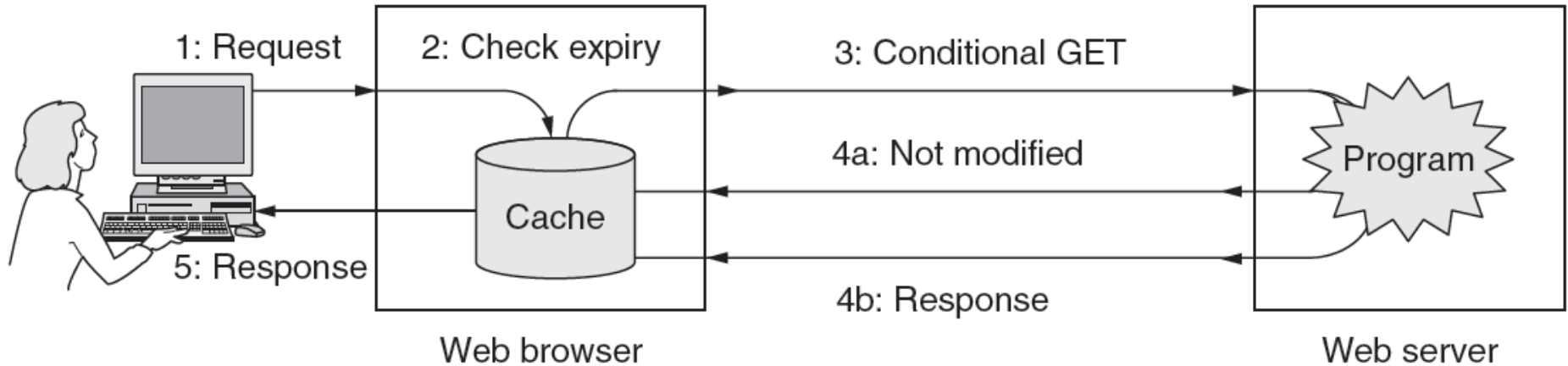
Web Caching (3)

- 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” header from server
 - Content is available after 1 RTT



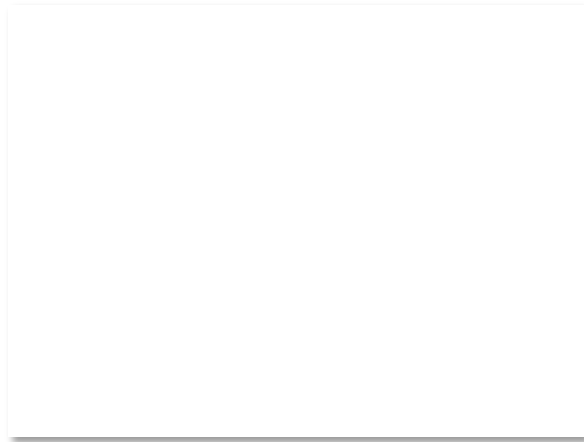
Web Caching (4)

- Putting the pieces together:



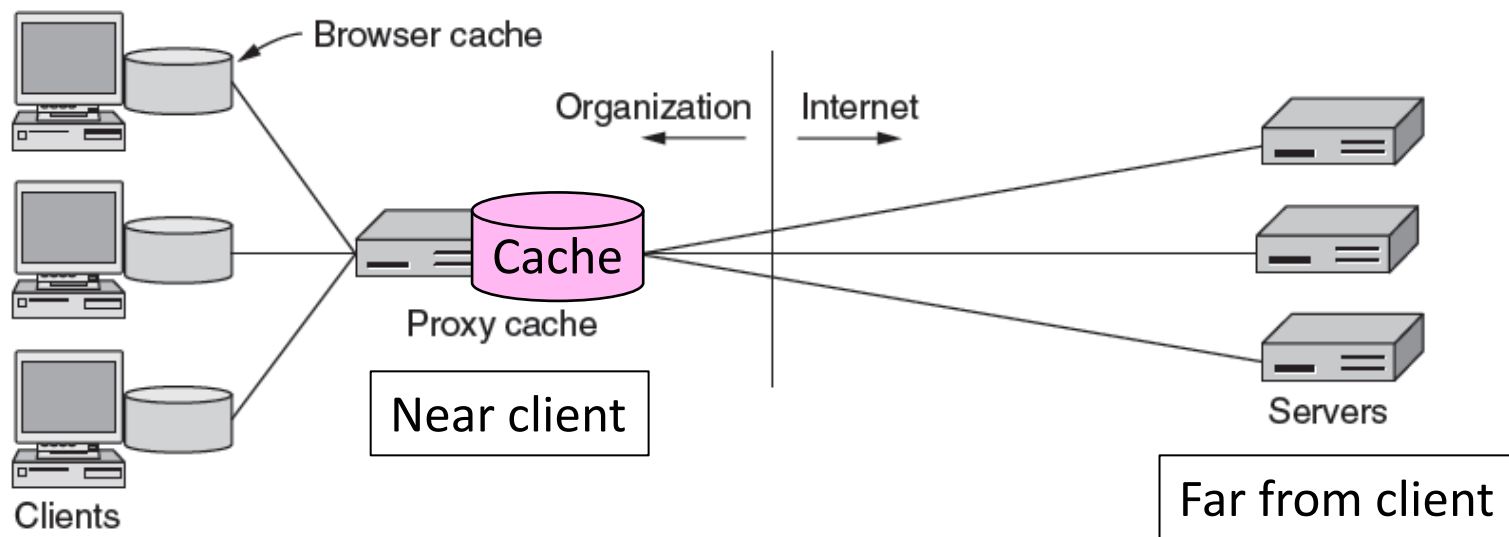
Web Proxies

- Place intermediary between pool of clients and external web servers
 - Benefits for clients include greater 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”



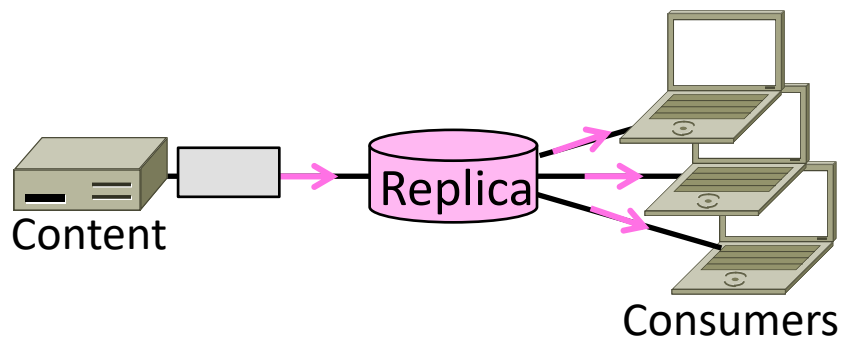
Web Proxies (2)

- Clients contact proxy; proxy contacts server



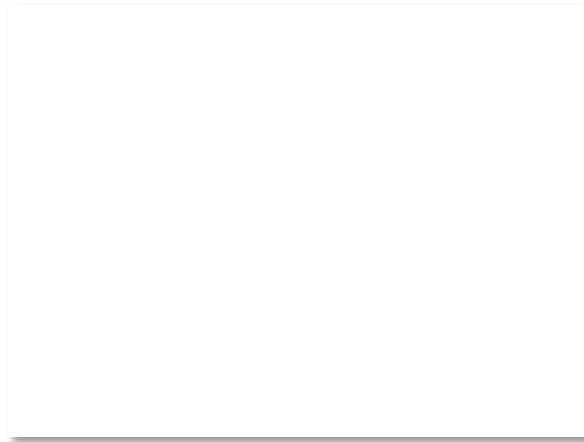
Topic

- CDNs (Content Delivery Networks)
 - Efficient distribution of popular content; faster delivery for clients



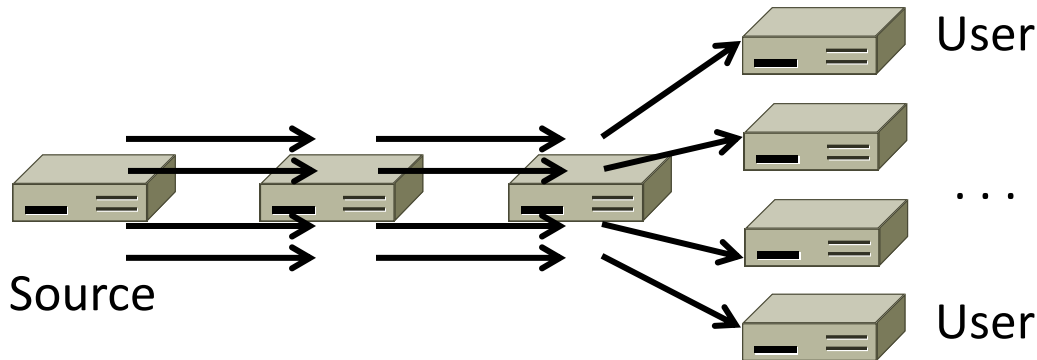
Context

- 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



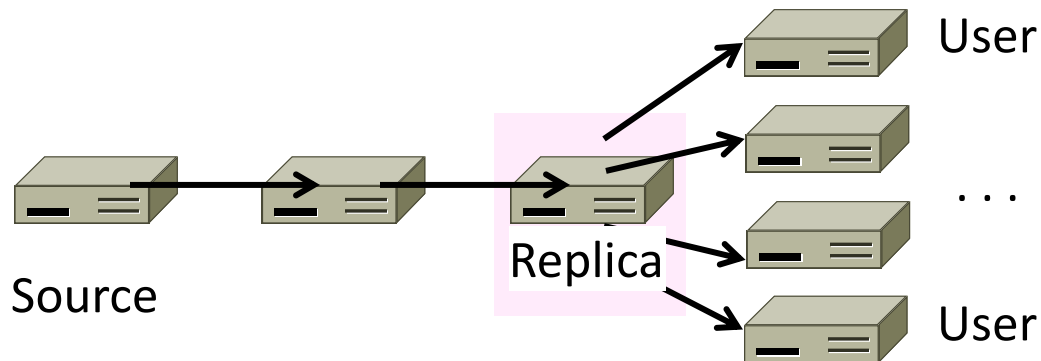
Before CDNs

- Sending content from the source to 4 users takes $4 \times 3 = 12$ “network hops” in the example



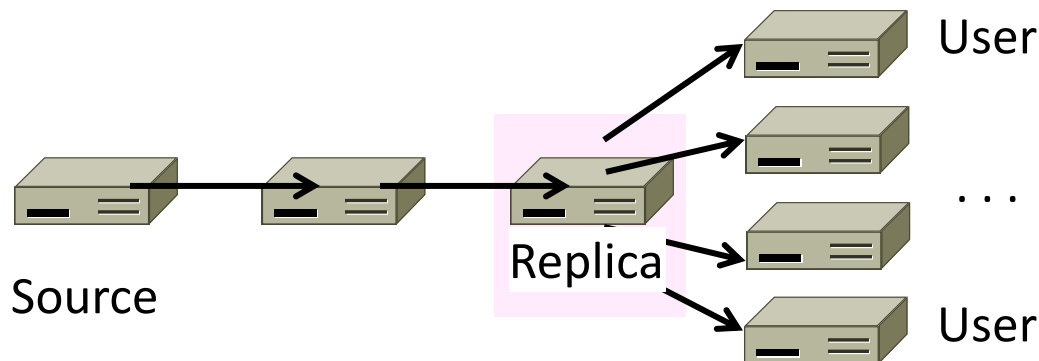
After CDNs

- Sending content via replicas takes only $4 + 2 = 6$ “network hops”



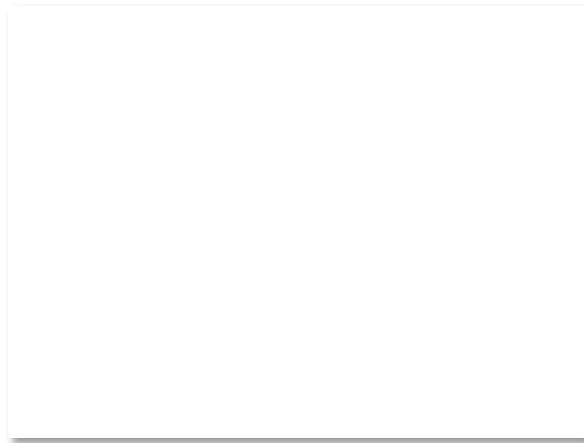
After CDNs (2)

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

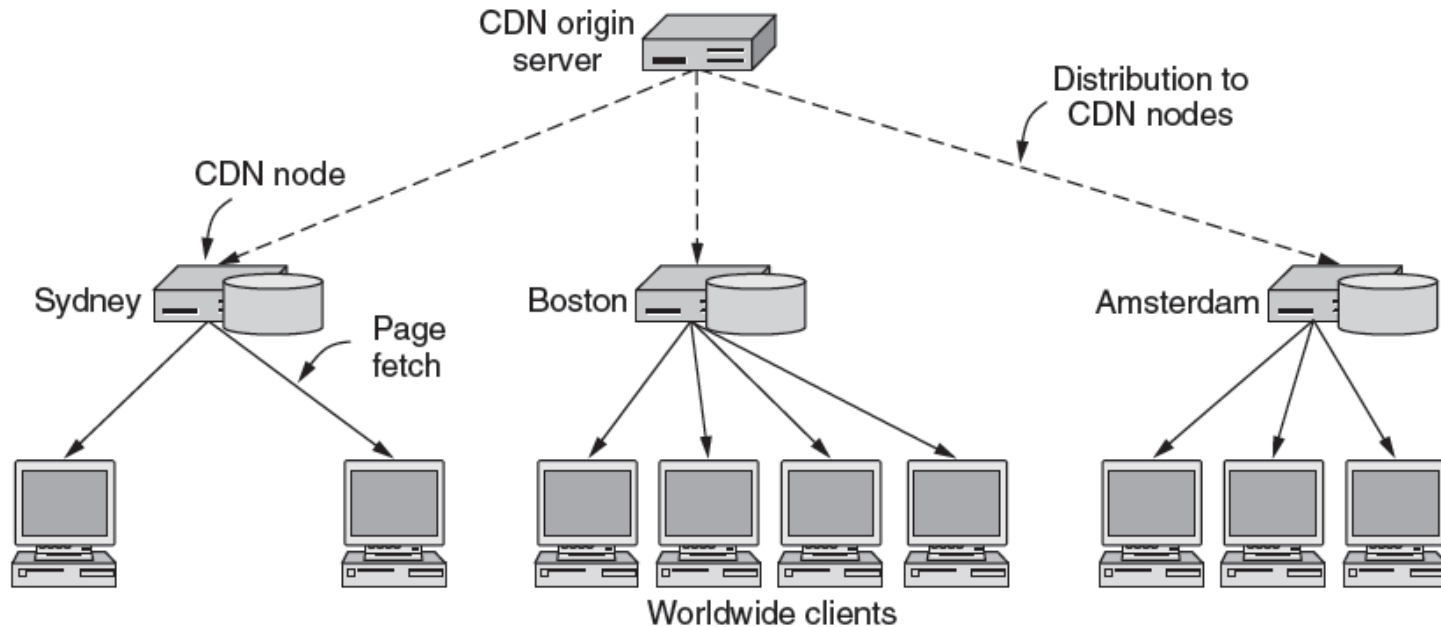


How to place content near clients?

- 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

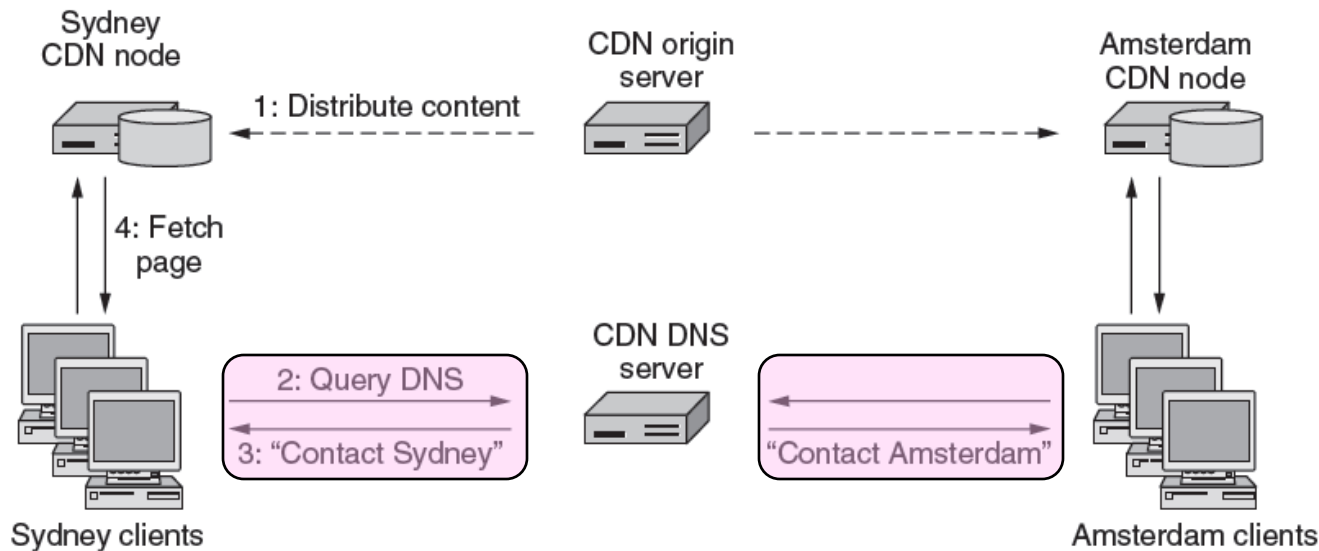


Content Delivery Network



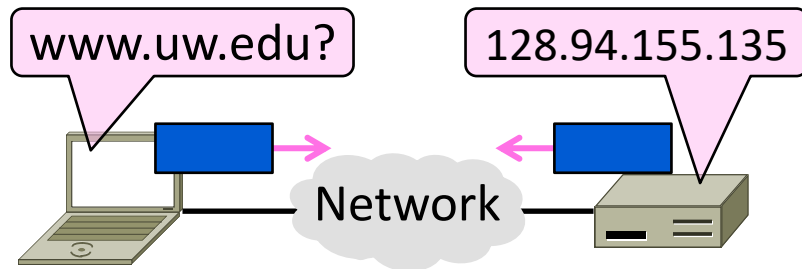
Content Delivery Network (2)

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



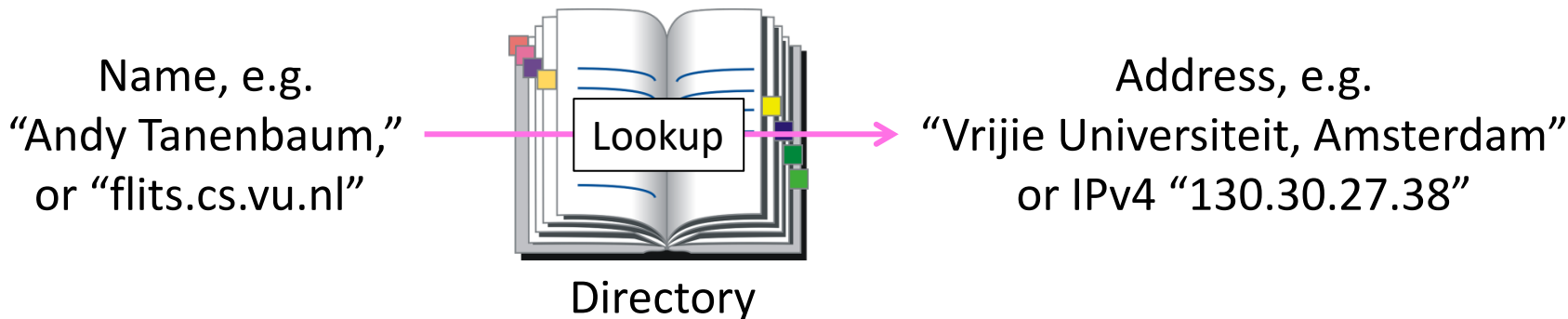
Topic

- The DNS (Domain Name System)
 - Human-readable host names, and more
 - Part 1: the distributed namespace



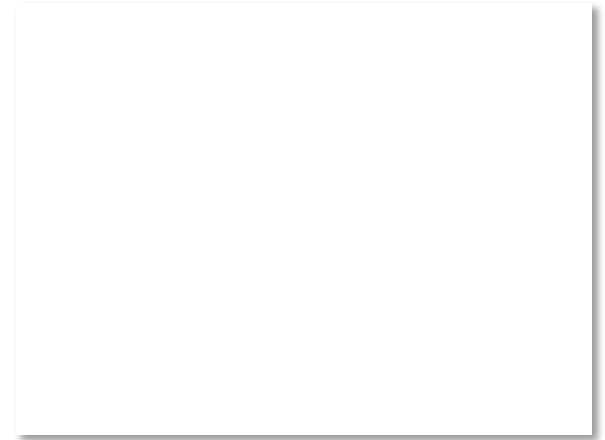
Names and Addresses

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



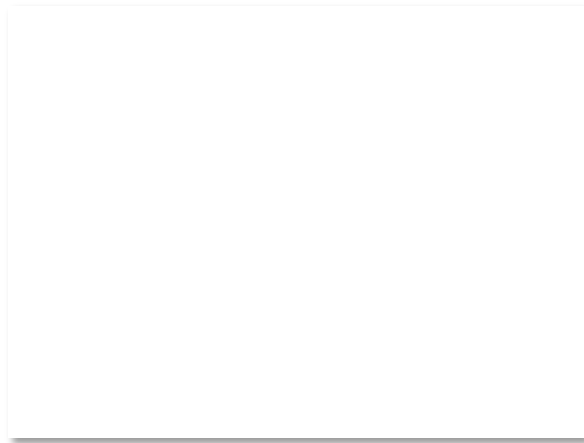
Before the DNS – HOSTS.TXT

- 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
- Neither manageable nor efficient as the ARPANET grew ...



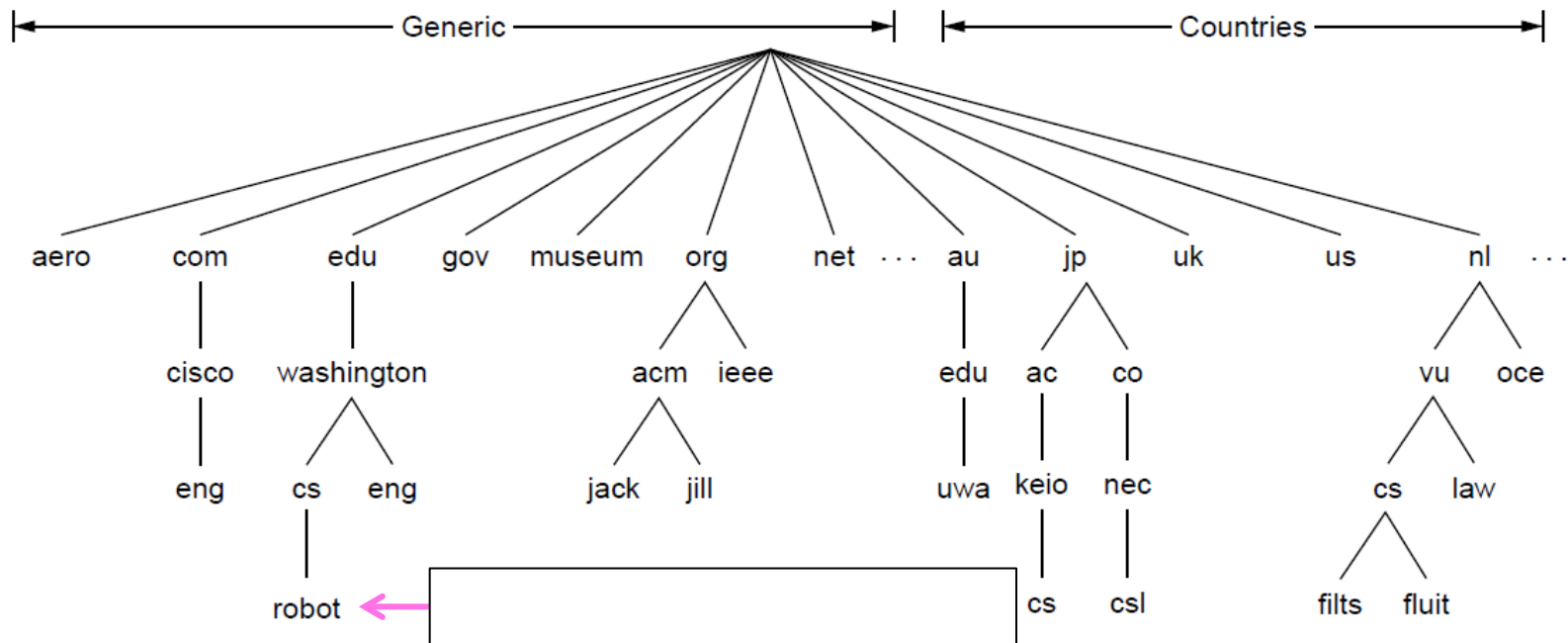
DNS

- 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)

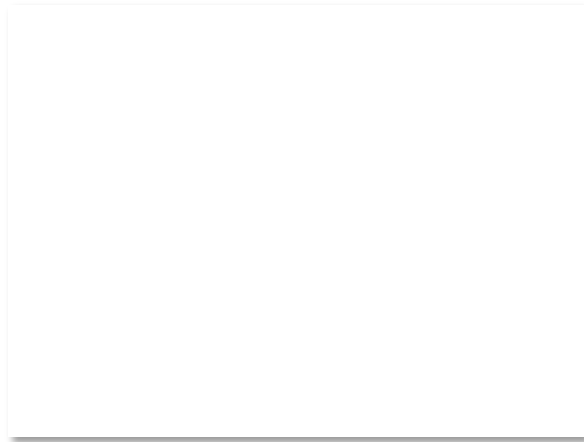


TLDs (Top-Level Domains)

- Run by ICANN (Internet Corp. for Assigned Names and Numbers)
 - Starting in '98; naming is financial, political, and international ☺
- 22+ generic TLDs
 - Initially .com, .edu , .gov., .mil, .org, .net
 - Added .aero, .museum, etc. from '01 through .xxx in '11
 - Different TLDs have different usage policies
- ~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), goo.gl (Greenland)

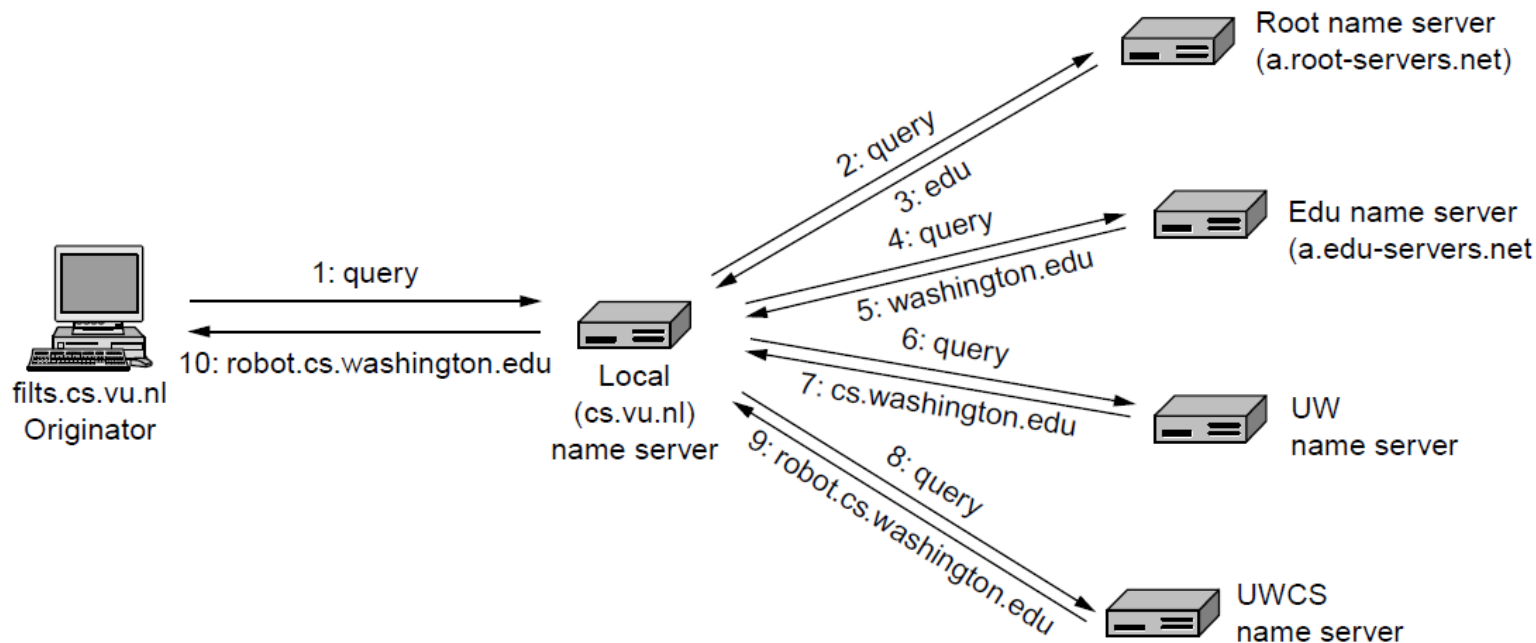
DNS Zones (2)

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



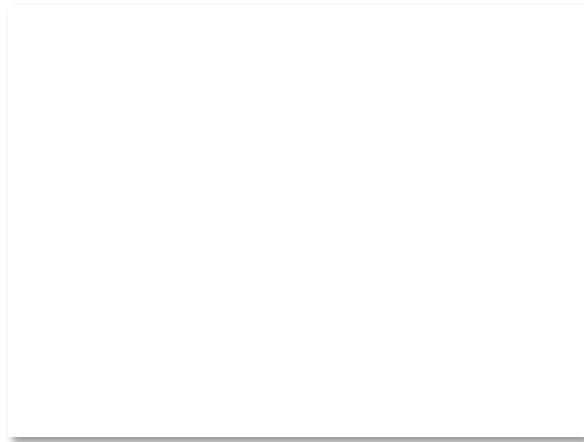
DNS Resolution (2)

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



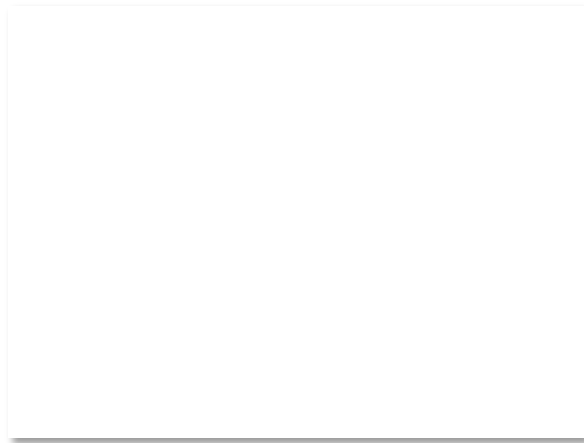
Iterative vs. Recursive Queries

- Recursive query
 - Nameserver completes resolution and returns the final answer
 - E.g., flits → local nameserver
- Iterative query
 - Nameserver returns the answer or who to contact next for the answer
 - E.g., local nameserver → all others



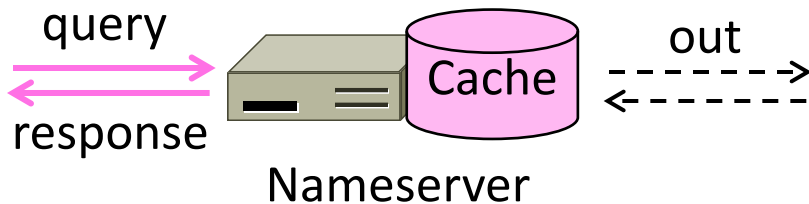
Iterative vs. Recursive Queries (2)

- 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



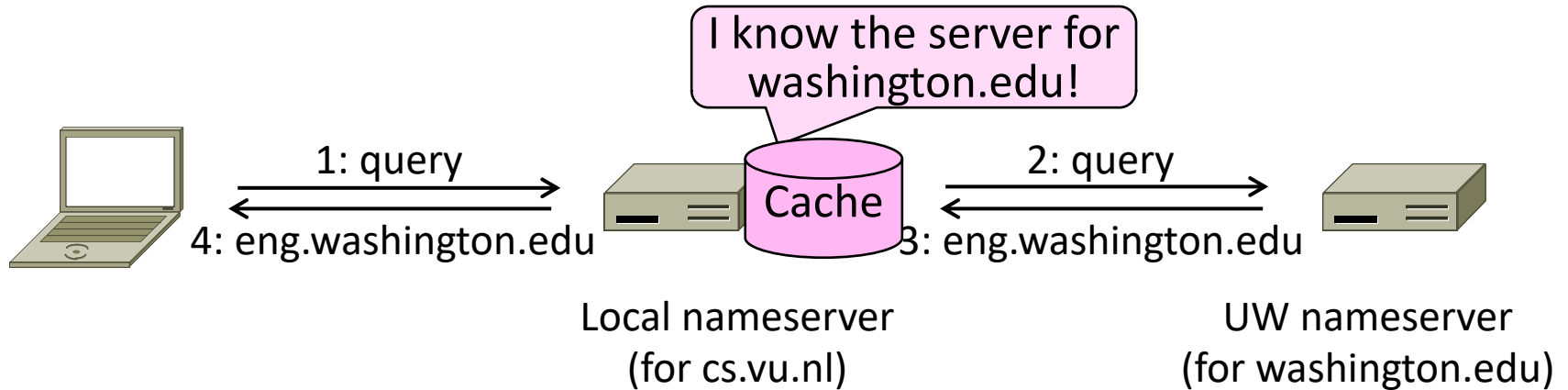
Caching

- 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



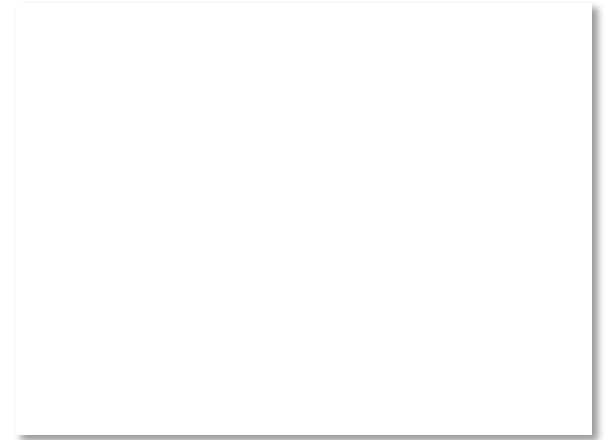
Caching (2)

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



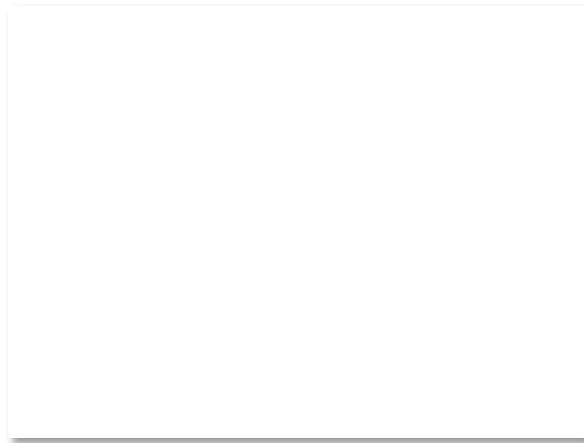
Local Nameservers

- Local nameservers typically 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 their local nameservers
 - Typically configured via DHCP

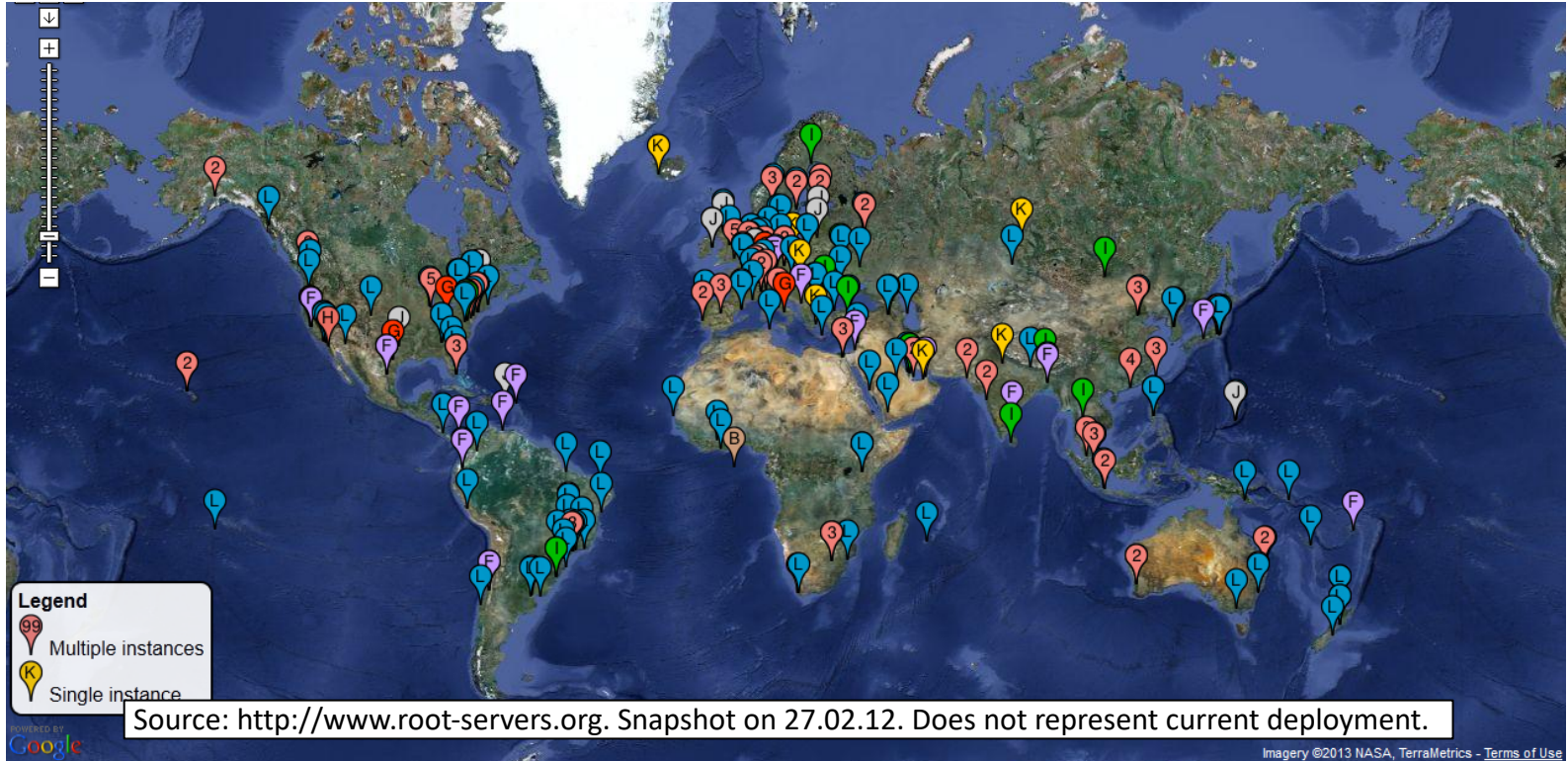


Root Nameservers

- 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 IP anycast (Multiple locations advertise same IP! Routes take client to the closest one. See §5.2.9)
 - Servers are IPv4 and IPv6 reachable

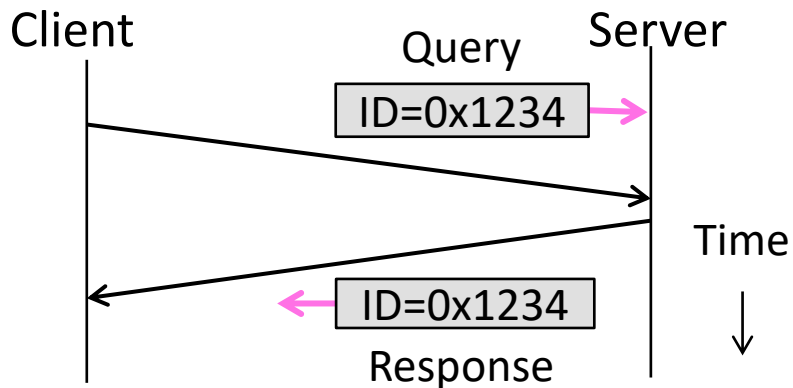


Root Server Deployment



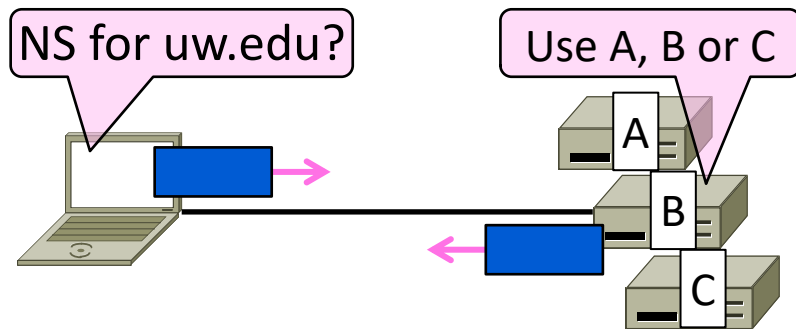
DNS Protocol

- 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)
 - Long under development, now partially deployed. We'll look at it later

