Application Layer

Anand Baswade

anand@iitbhilai.ac.in

Maintaining user/server state: cookies

Web sites and client browser use *cookies* to maintain some state between transactions

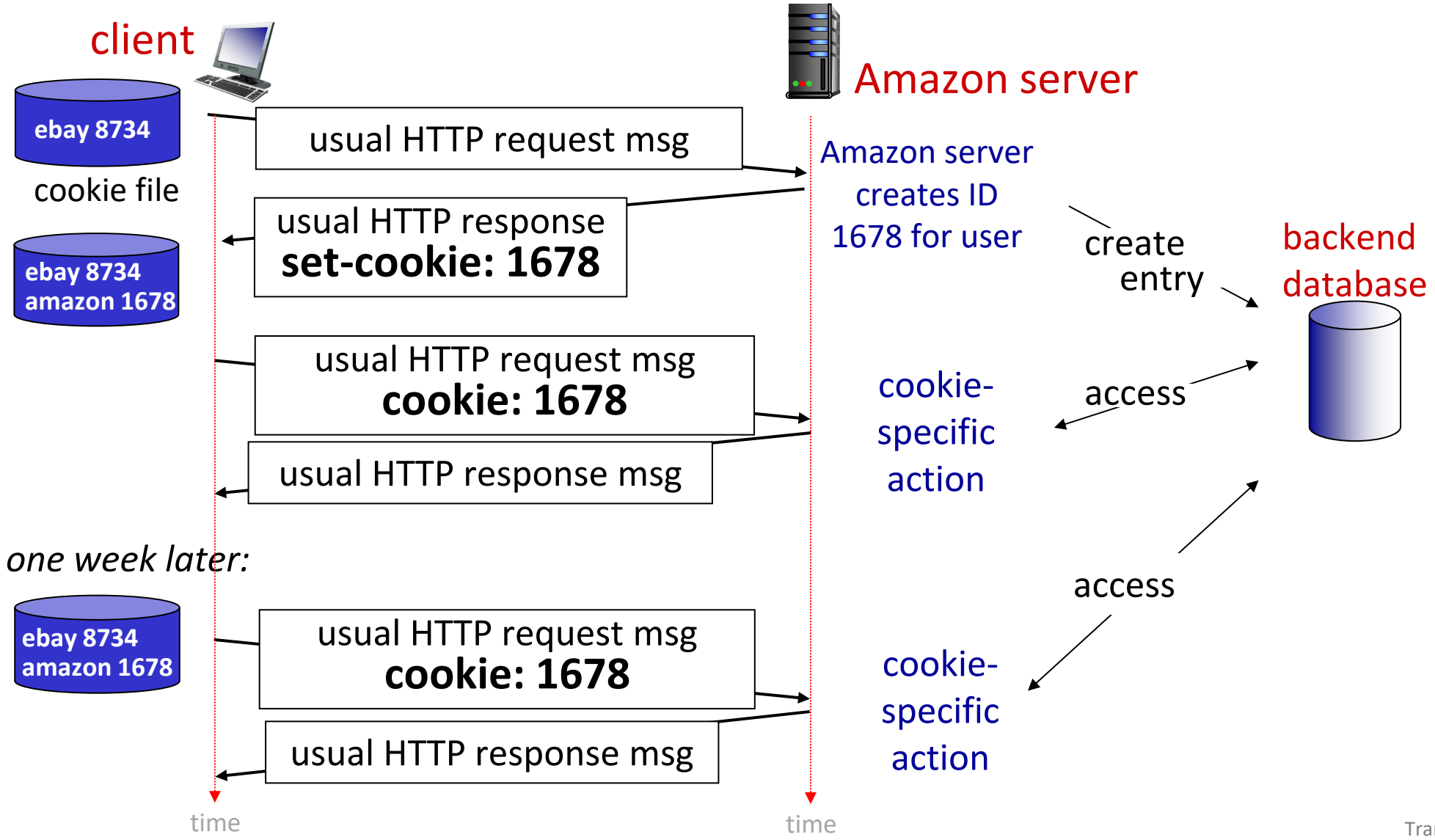
four components:

1) cookie header line of HTTP *response* message
2) cookie header line in next HTTP *request* message
3) cookie file kept on user's host, managed by user's browser
4) back-end database at Web site

Example:

- Susan uses browser on laptop, visits specific e-commerce site for first time
- when initial HTTP requests arrives at site, site creates:
  - unique ID (aka "cookie")
  - entry in backend database for ID
- subsequent HTTP requests from Susan to this site will contain cookie ID value, allowing site to "identify" Susan

Maintaining user/server state: cookies



HTTP cookies:

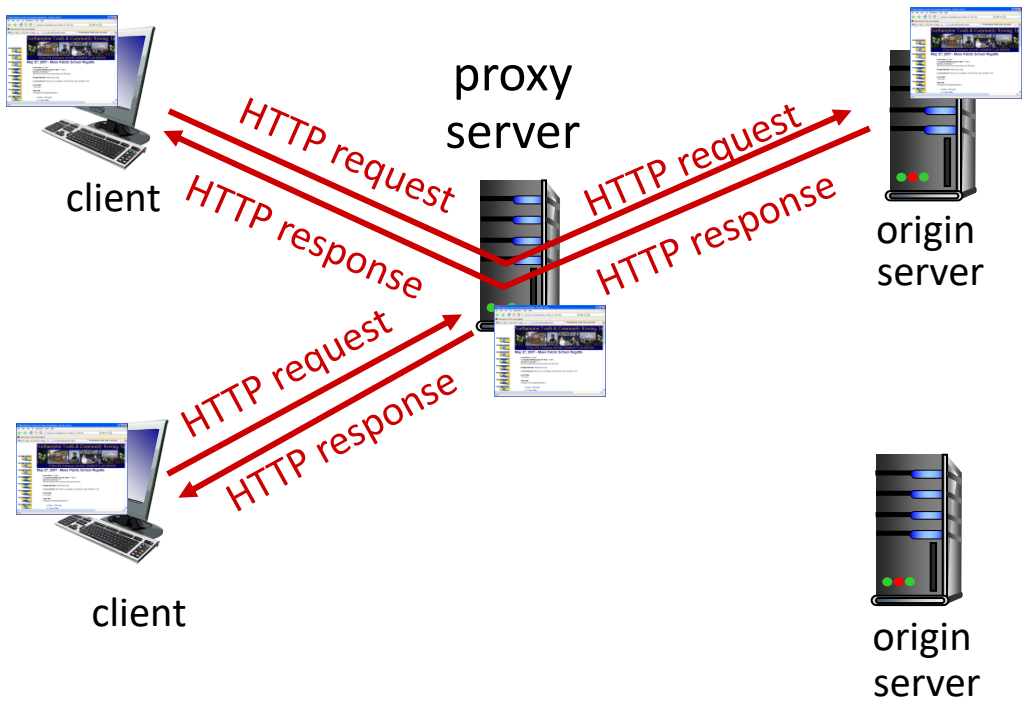
What cookies can be used for:

- authorization
- shopping carts
- recommendations
- user session state (Web e-mail)

Web caches (proxy servers)

Goal: satisfy client request without involving origin server

- user configures browser to point to a *Web cache*
- browser sends all HTTP requests to cache
  - *if* object in cache: cache returns object to client
  - *else* cache requests object from origin server, caches received object, then returns object to client



Web caches (proxy servers)

- Web cache acts as both client and server
  - server for original requesting client
  - client to origin server
- typically cache is installed by ISP (university, company, residential ISP)

Why Web caching?

- reduce response time for client request
  - cache is closer to client
- reduce traffic on an institution's access link
- Internet is dense with caches
  - enables "poor" content providers to more effectively deliver content

Browser caching: Conditional GET

Goal: don't send object if browser has up-to-date cached version

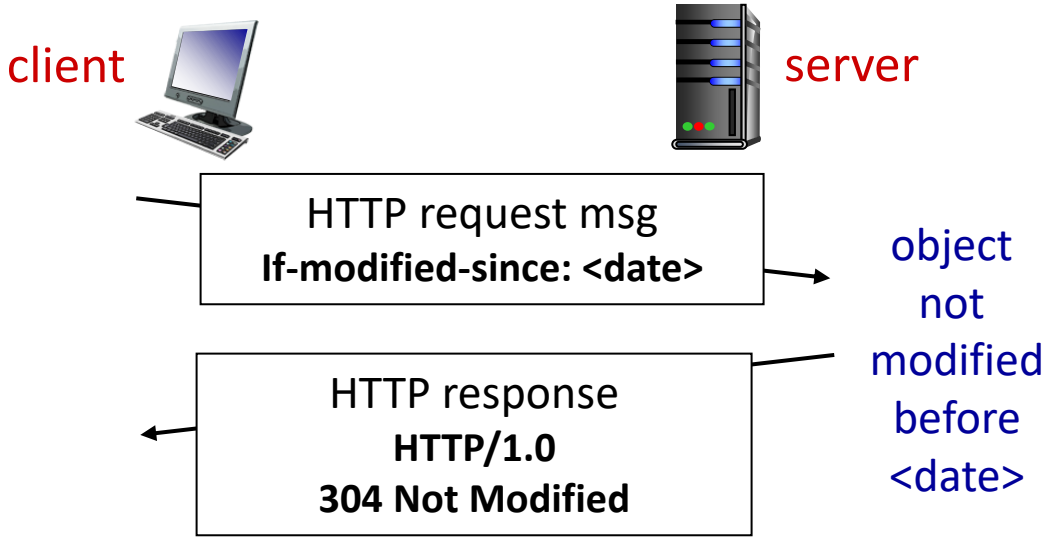
- no object transmission delay (or use of network resources)

- *client:* specify date of browser-cached copy in HTTP request

  If-modified-since: <date>

- *server:* response contains no object if browser-cached copy is up-to-date:

  HTTP/1.0 304 Not Modified



HTTP/2

Key goal: decreased delay in multi-object HTTP requests

HTTP1.1: introduced multiple, pipelined GETs over single TCP connection

- server responds *in-order* (FCFS: first-come-first-served scheduling) to GET requests
- with FCFS, small object may have to wait for transmission (head-of-line (HOL) blocking) behind large object(s)
- loss recovery (retransmitting lost TCP segments) stalls object transmission

HTTP/2

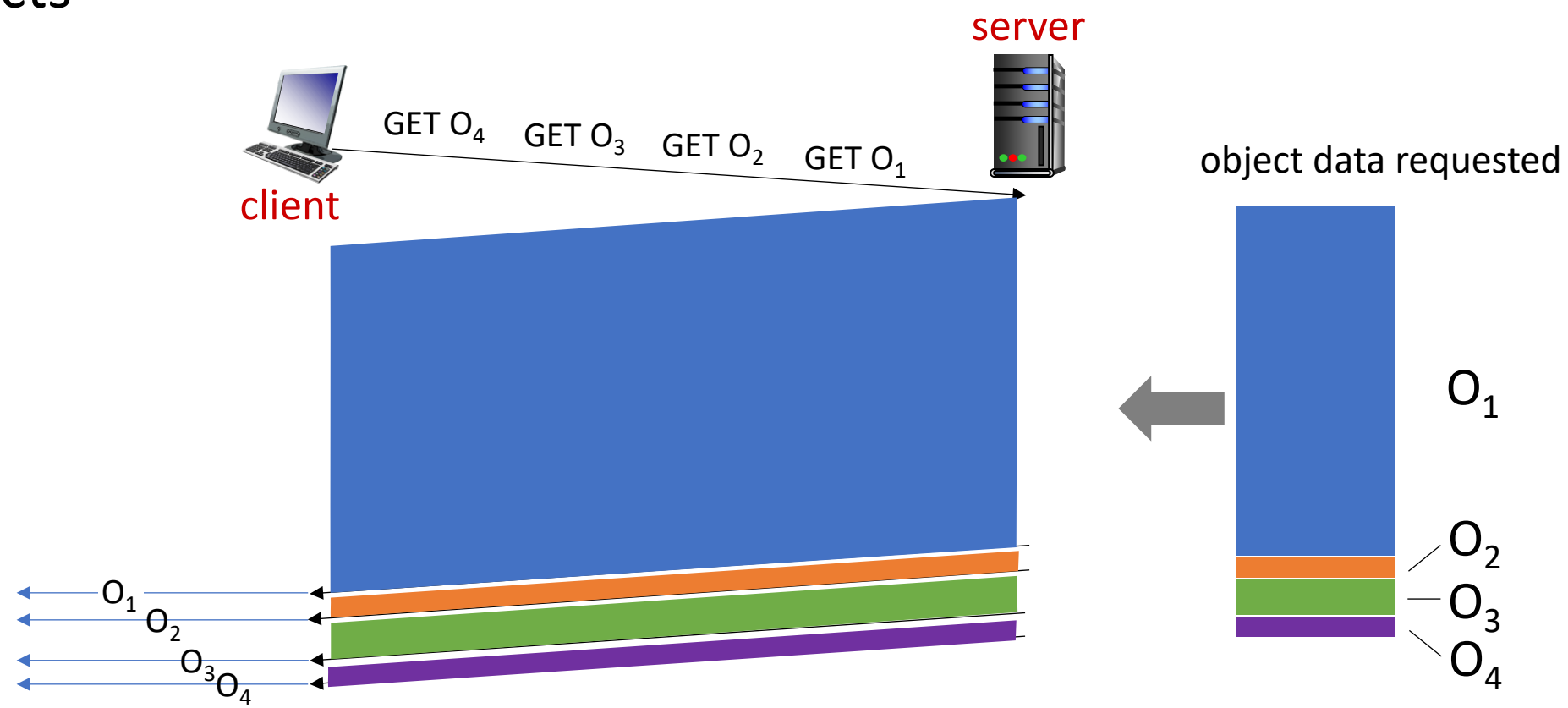
Key goal: decreased delay in multi-object HTTP requests

HTTP/2: [RFC 7540, 2015] increased flexibility at *server* in sending objects to client:

- methods, status codes, most header fields unchanged from HTTP 1.1
- transmission order of requested objects based on client-specified object priority (not necessarily FCFS)
- *push* unrequested objects to client
- divide objects into frames, schedule frames to mitigate HOL blocking

HTTP/2: mitigating HOL blocking

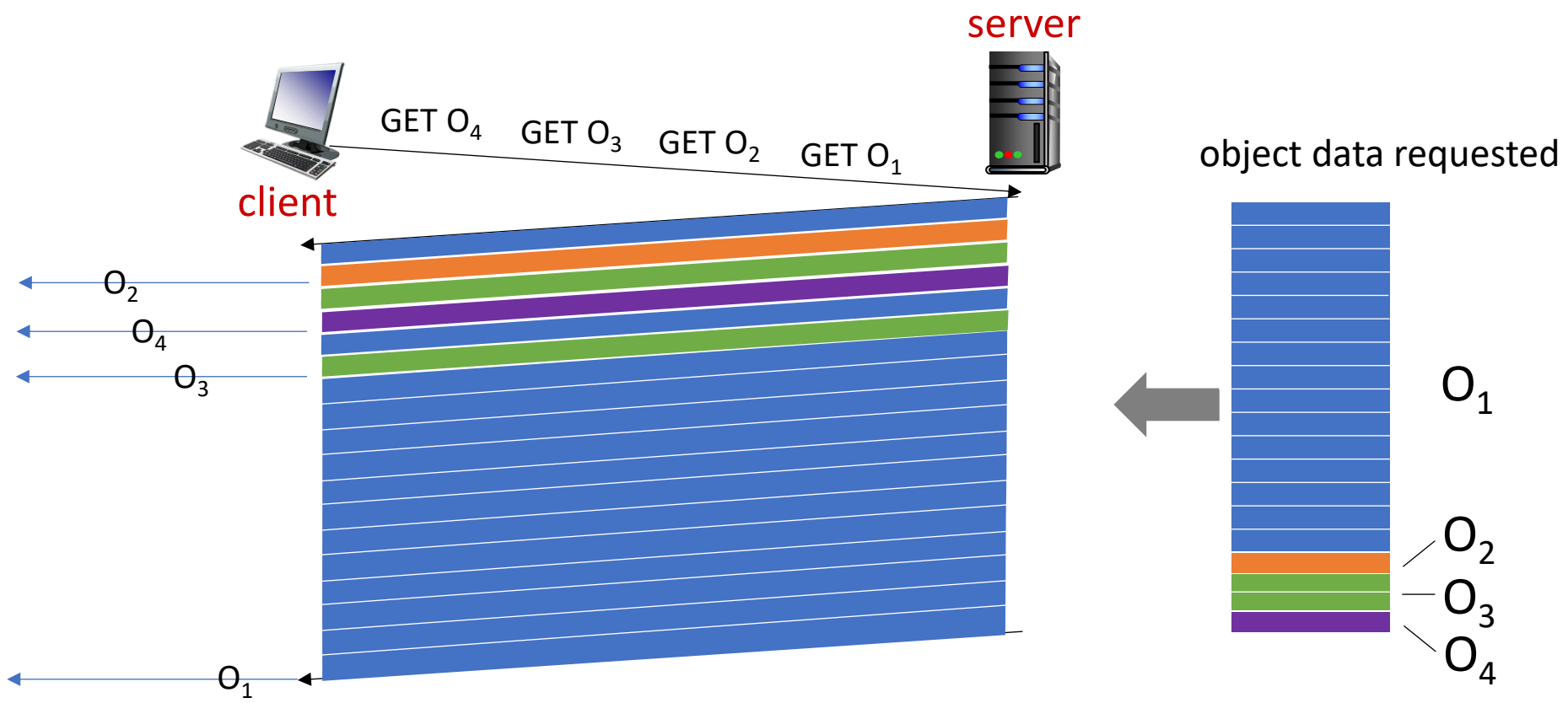
HTTP 1.1: client requests 1 large object (e.g., video file) and 3 smaller objects



objects delivered in order requested: O_2, O_3, O_4 wait behind O_1

HTTP/2: mitigating HOL blocking

HTTP/2: objects divided into frames, frame transmission interleaved



O_2, O_3, O_4 delivered quickly, O_1 slightly delayed

HTTP/2 to HTTP/3

HTTP/2 over single TCP connection means:

- recovery from packet loss still stalls all object transmissions
  - as in HTTP 1.1, browsers have incentive to open multiple parallel TCP connections to reduce stalling, increase overall throughput
- no security over vanilla TCP connection
- HTTP/3: adds security, per object error- and congestion-control (more pipelining) over UDP
  - more on HTTP/3 in transport layer

Application Layer: Overview

- Principles of network applications
- Web and HTTP
- **The Domain Name System DNS**
- E-mail, SMTP, IMAP
- P2P applications
- video streaming and content distribution networks
- socket programming with UDP and TCP

DNS: Domain Name System

people: many identifiers:

- name, passport #, Aadhar #

Internet hosts, routers:

- IP address (32 bit) - used for addressing datagrams
- "name", e.g., cs.umass.edu - used by humans

Q: how to map between IP address and name, and vice versa ?

Domain Name System:

- *distributed database* implemented in hierarchy of many *name servers*
- *application-layer protocol:* hosts, name servers communicate to *resolve* names (address/name translation)
  - note: core Internet function, *implemented as application-layer protocol*

DNS: services, structure

DNS services

- hostname to IP address translation
- Host aliasing
  - canonical, alias names
- Mail server aliasing
- load distribution
  - replicated Web servers: many IP addresses correspond to one name

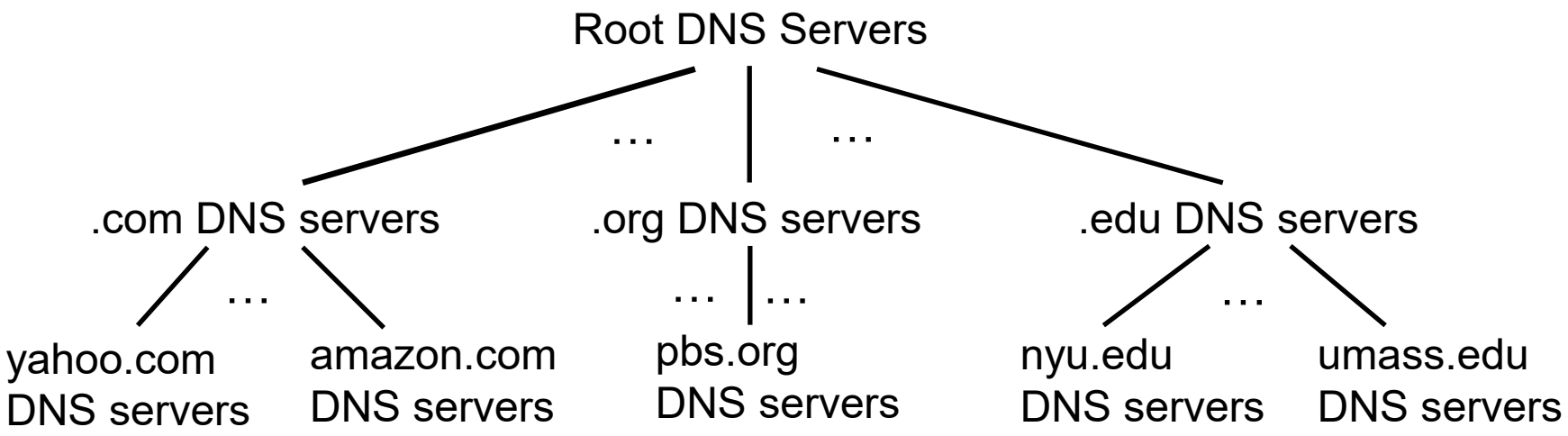
Q: Why not centralize DNS?

- single point of failure
- traffic volume
- distant centralized database
- maintenance

A: doesn‘t scale!

- Comcast DNS servers alone: 600B DNS queries per day

DNS: a distributed, hierarchical database



Root

Top Level Domain

Authoritative

Client wants IP address for www.amazon.com; 1st approximation:

- client queries root server to find .com DNS server
- client queries .com DNS server to get amazon.com DNS server
- client queries amazon.com DNS server to get IP address for www.amazon.com

DNS: root name servers

- official, contact-of-last-resort by name servers that can not resolve name
- *incredibly important* Internet function
  - Internet couldn't function without it!
  - DNSSEC – provides security (authentication and message integrity)
- ICANN (Internet Corporation for Assigned Names and Numbers) manages root DNS domain

13 logical root name "servers" worldwide each "server" replicated many times (~200 servers in US)