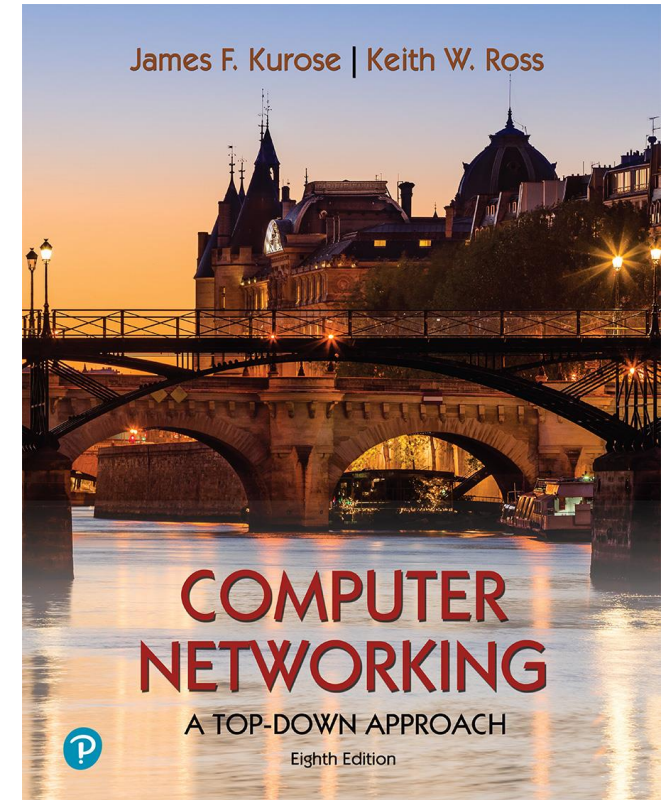# Computer Networks

Amir Mahdi Sadeghzadeh, Ph.D.

Chapter 2
Application Layer

Computer Networking: A Top-Down Approach

8th edition   n
Jim Kurose, Keith Ross
Pearson, 2020

Creating a network app

write programs that:

- run on (different) end systems
- communicate over network
- e.g., web server software communicates with browser software

no need to write software for network-core devices

- network-core devices do not run user applications
- applications on end systems allows for rapid app development, propagation
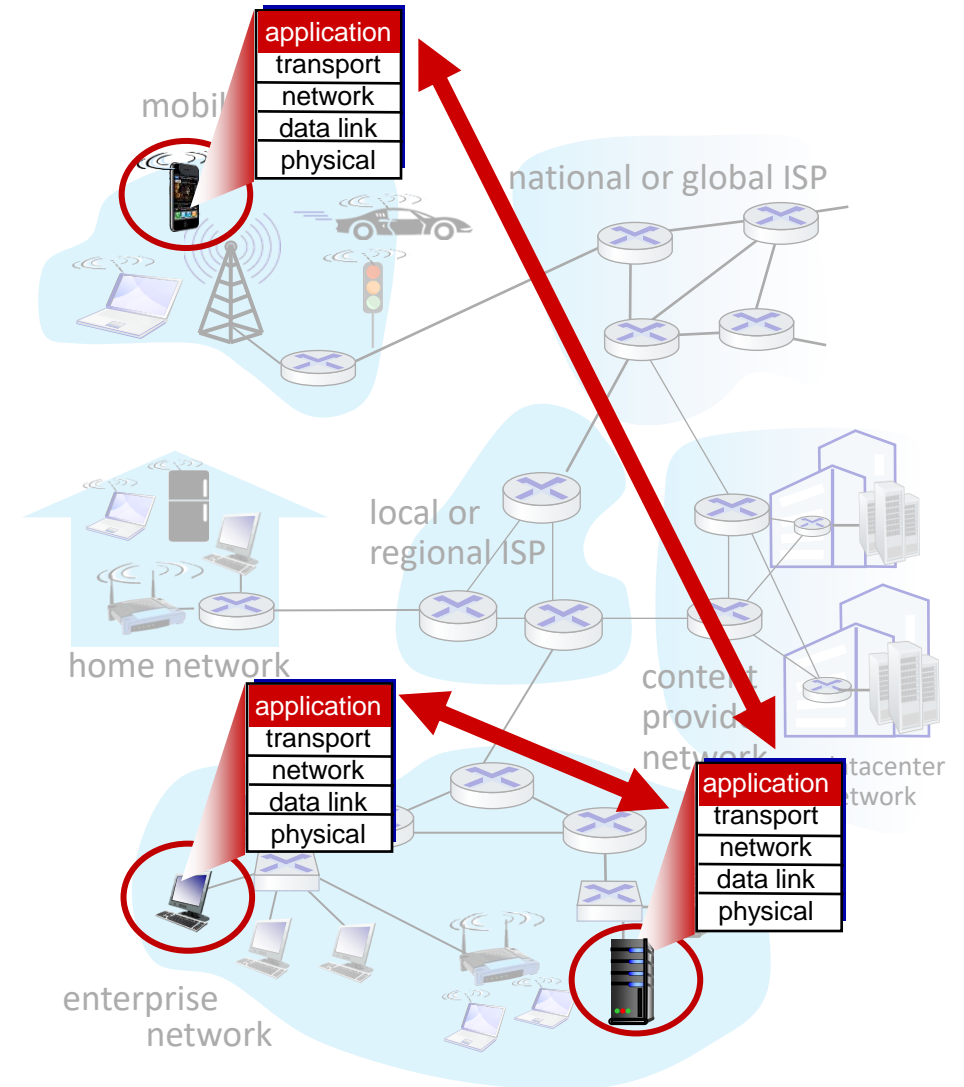
Sockets

- process sends/receives messages to/from its socket

- socket analogous to door

  - sending process shoves message out door

  - sending process relies on transport infrastructure on other side of door to deliver message to socket at receiving process

  - two sockets involved: one on each side
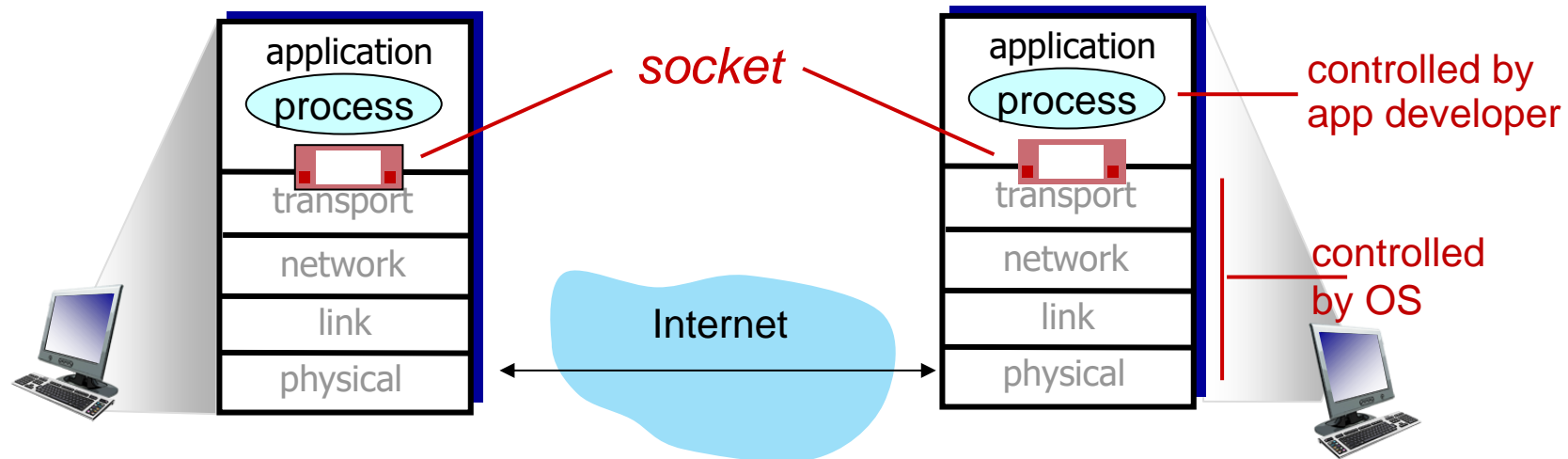
Application layer: overview

- Principles of network applications

- **Web and HTTP**

- E-mail, SMTP, IMAP

- The Domain Name System DNS

- P2P applications

- video streaming and content distribution networks

- socket programming with UDP and TCP

Web and HTTP

First, a quick review...

- web page consists of *objects,* each of which can be stored on different Web servers

- object can be HTML file, JPEG image, Java applet, audio file,...

- web page consists of *base HTML-file* which includes *several referenced objects, each* addressable by a *URL,* e.g.,
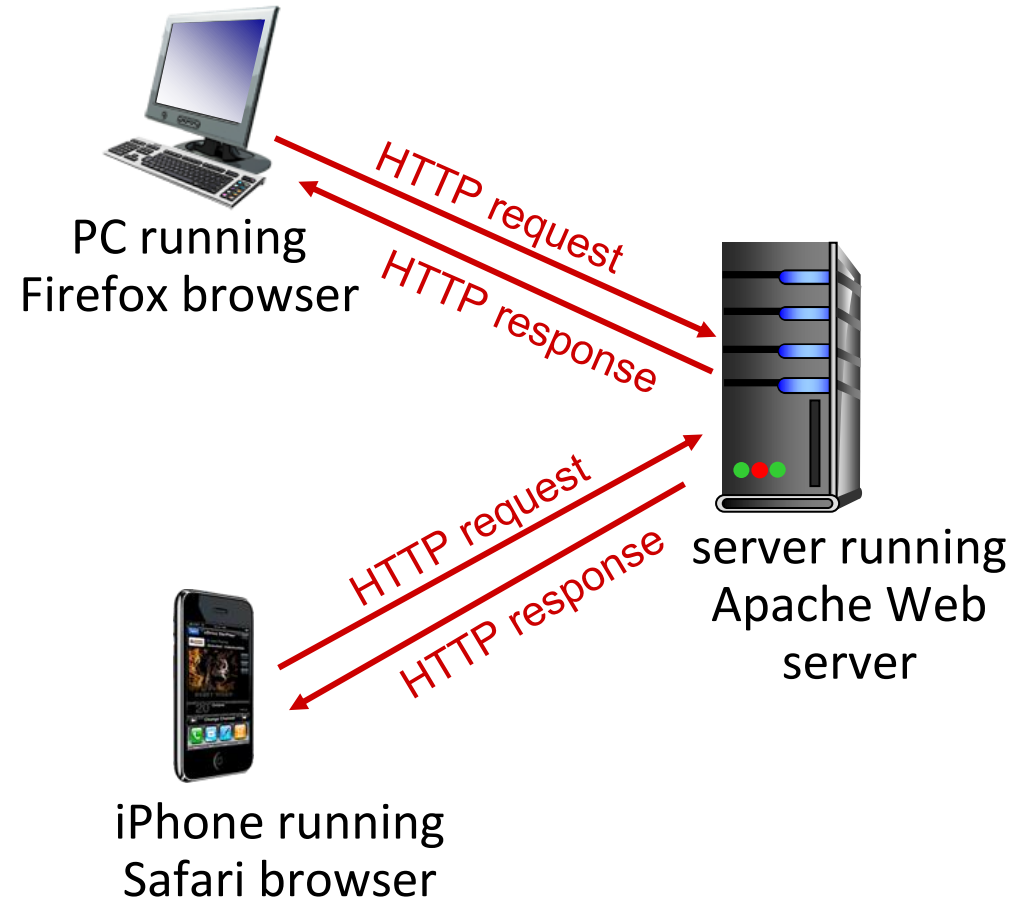
```
www.someschool.edu/someDept/pic.gif
```

host name   path name

HTTP overview

## HTTP: hypertext transfer protocol

- Web's application-layer protocol
- client/server model:
  - *client:* browser that requests, receives, (using HTTP protocol) and "displays" Web objects
  - *server:* Web server sends (using HTTP protocol) objects in response to requests

PC running
Firefox browser

HTTP request

HTTP response

server running
Apache Web
server

HTTP request

HTTP response

iPhone running
Safari browser

Web caches

Goal: satisfy client requests without involving origin server

- user configures browser to point to a (local) *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
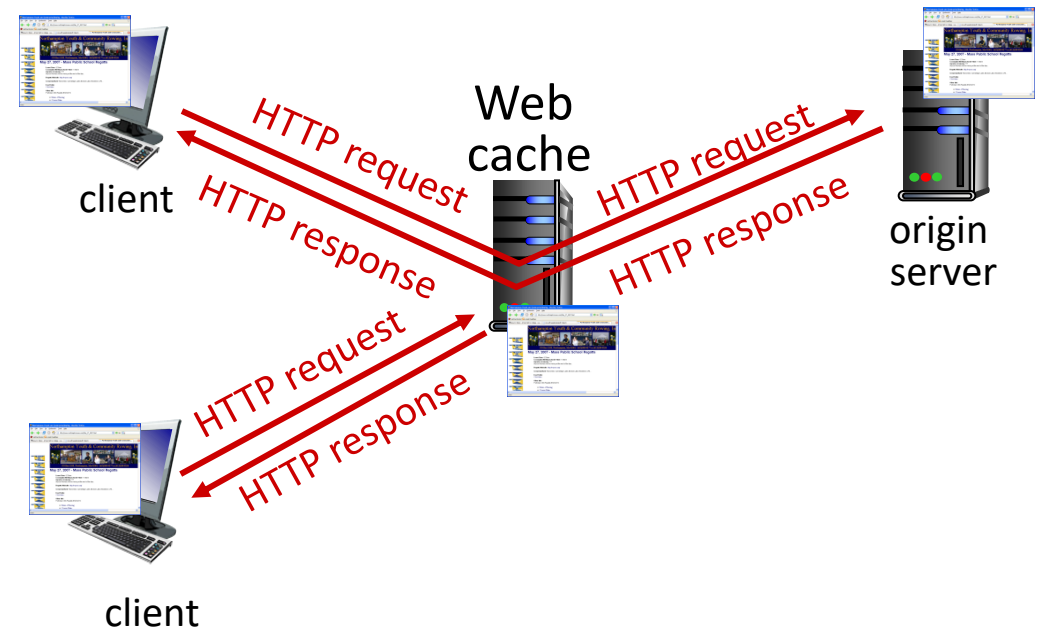
Web caches (aka proxy servers)

- Web cache acts as both client and server
  - server for original requesting client
  - client to origin server
- server tells cache about object's allowable caching in response header:

  ```
  Cache-Control: max-age=<seconds>
  ```

  ```
  Cache-Control: no-cache
  ```

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
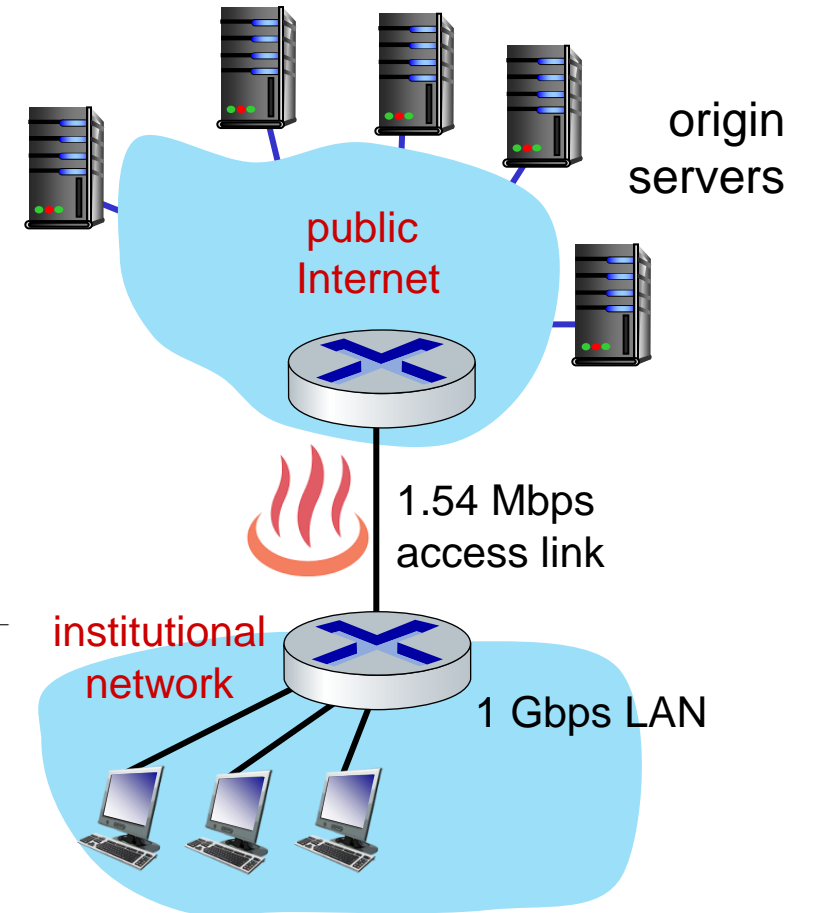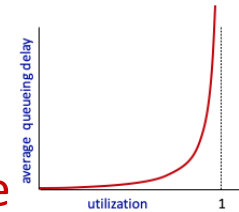 - enables "poor" content providers to more effectively deliver content

Caching example

Scenario:

- access link rate: 1.54 Mbps
- RTT from institutional router to server: 2 sec
- web object size: 100K bits
- average request rate from browsers to origin servers: 15/sec
  - avg data rate to browsers: 1.50 Mbps

Performance:

- access link utilization = .97

  problem: large queueing delays at high utilization!

- LAN utilization: .0015
- end-end delay = Internet delay +
  access link delay + LAN delay
  = 2 sec + minutes + usecs

origin servers

public Internet

1.54 Mbps access link

institutional network

1 Gbps LAN

Option 1: buy a faster access link

Scenario:

- access link rate: ~~1.54~~ **154 Mbps**
- RTT from institutional router to server: 2 sec
- web object size: 100K bits
- average request rate from browsers to origin servers: 15/sec
  - avg data rate to browsers: 1.50 Mbps

Performance:

- access link utilization = ~~.97~~ **.0097**
- LAN utilization: .0015
- end-end delay = Internet delay + access link delay + LAN delay
  = 2 sec + ~~minutes~~ + usecs → **msecs**

Cost: faster access link (expensive!)

origin servers

public Internet

154 Mbps
~~1.54 Mbps~~
access link

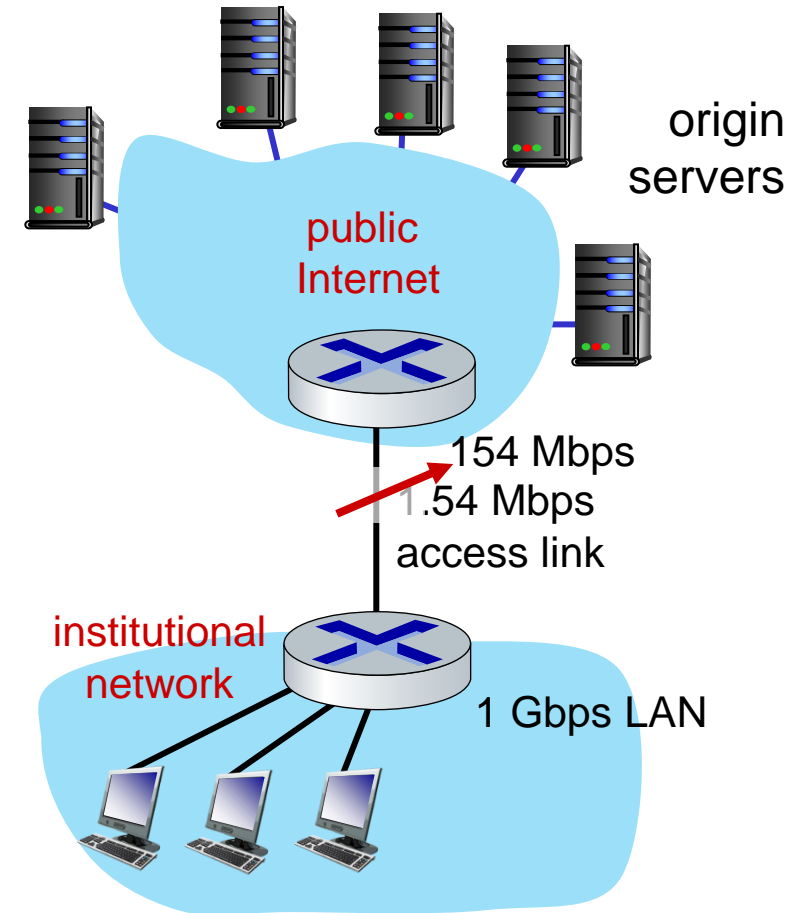institutional network

1 Gbps LAN

Option 2: install a web cache

Scenario:

- access link rate: 1.54 Mbps
- RTT from institutional router to server: 2 sec
- web object size: 100K bits
- average request rate from browsers to origin servers: 15/sec
  - avg data rate to browsers: 1.50 Mbps

Cost: web cache (cheap!)

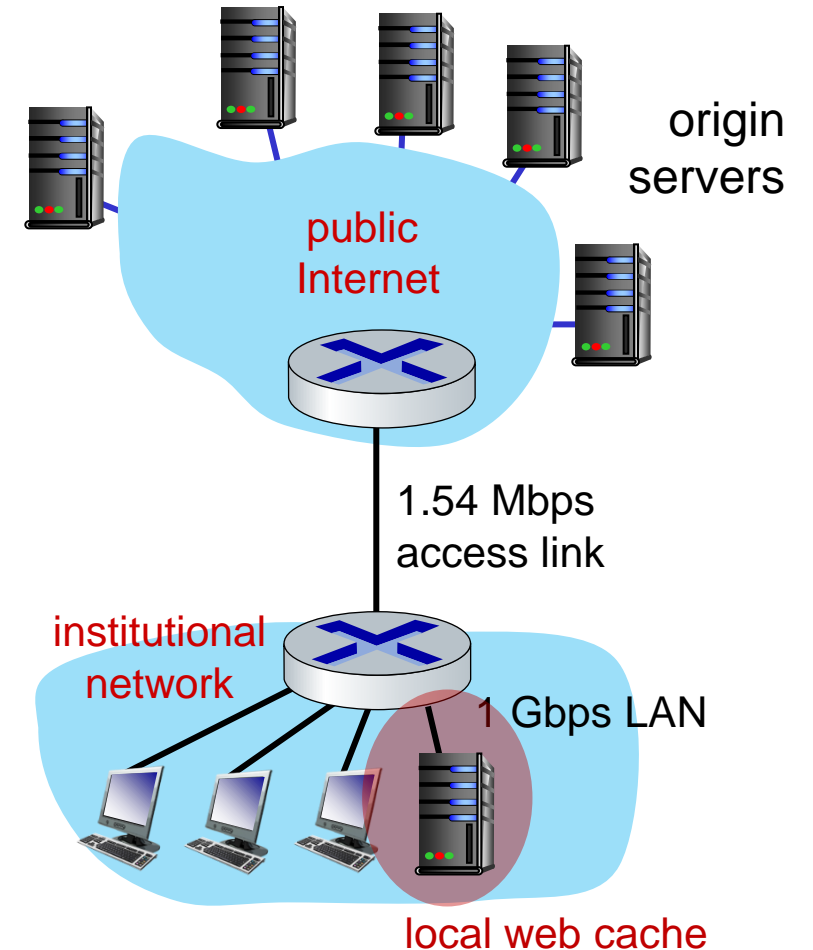*Performance:*

- LAN utilization: .?
- access link utilization = ?
- average end-end delay = ?

How to compute link utilization, delay?

origin servers

public Internet

1.54 Mbps access link

institutional network

1 Gbps LAN

local web cache
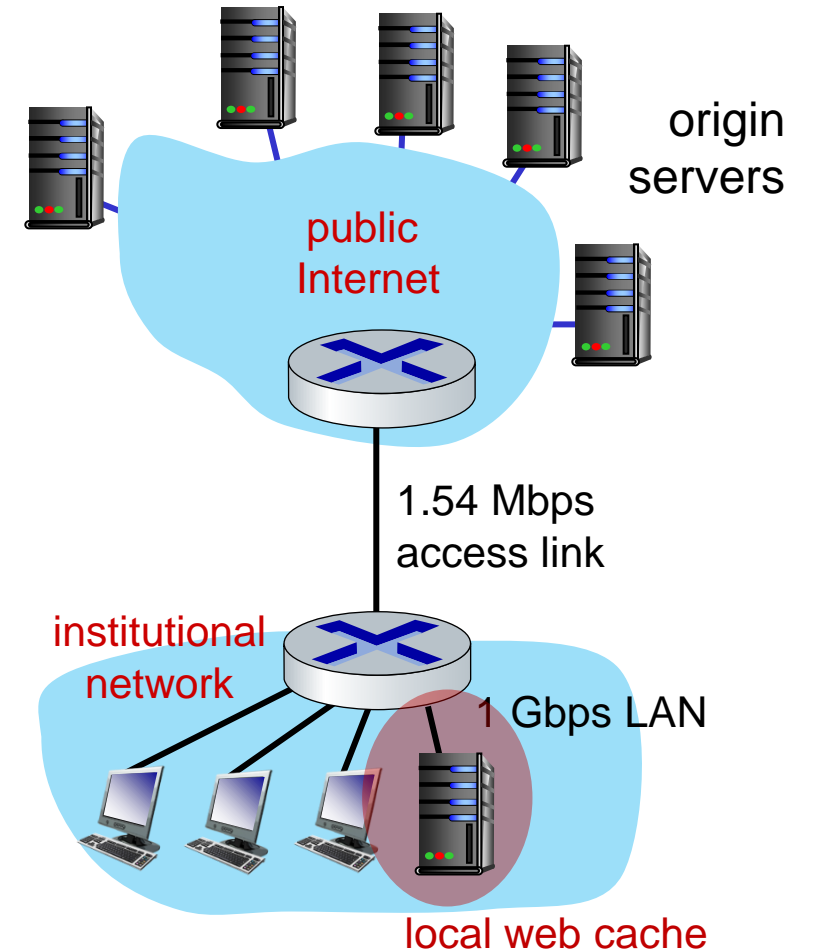
Calculating access link utilization, end-end delay with cache:

suppose cache hit rate is 0.4:

- 40% requests served by cache, with low (msec) delay

- 60% requests satisfied at origin

  - rate to browsers over access link

    = 0.6 * 1.50 Mbps = .9 Mbps

  - access link utilization = 0.9/1.54 = .58 means low (msec) queueing delay at access link

- average end-end delay:

  = 0.6 * (delay from origin servers)

      + 0.4 * (delay when satisfied at cache)

  = 0.6 (2.01) + 0.4 (~msecs) = ~ 1.2 secs

lower average end-end delay than with 154 Mbps link (and cheaper too!)

origin servers

public Internet

1.54 Mbps access link

institutional network

1 Gbps LAN

local web cache

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

client

server

HTTP request msg
If-modified-since: <date>

object
not
modified
before
<date>

HTTP response
**HTTP/1.0
304 Not Modified**

HTTP request msg
If-modified-since: <date>

object
modified
after
<date>

HTTP response
**HTTP/1.0 200 OK
<data>**
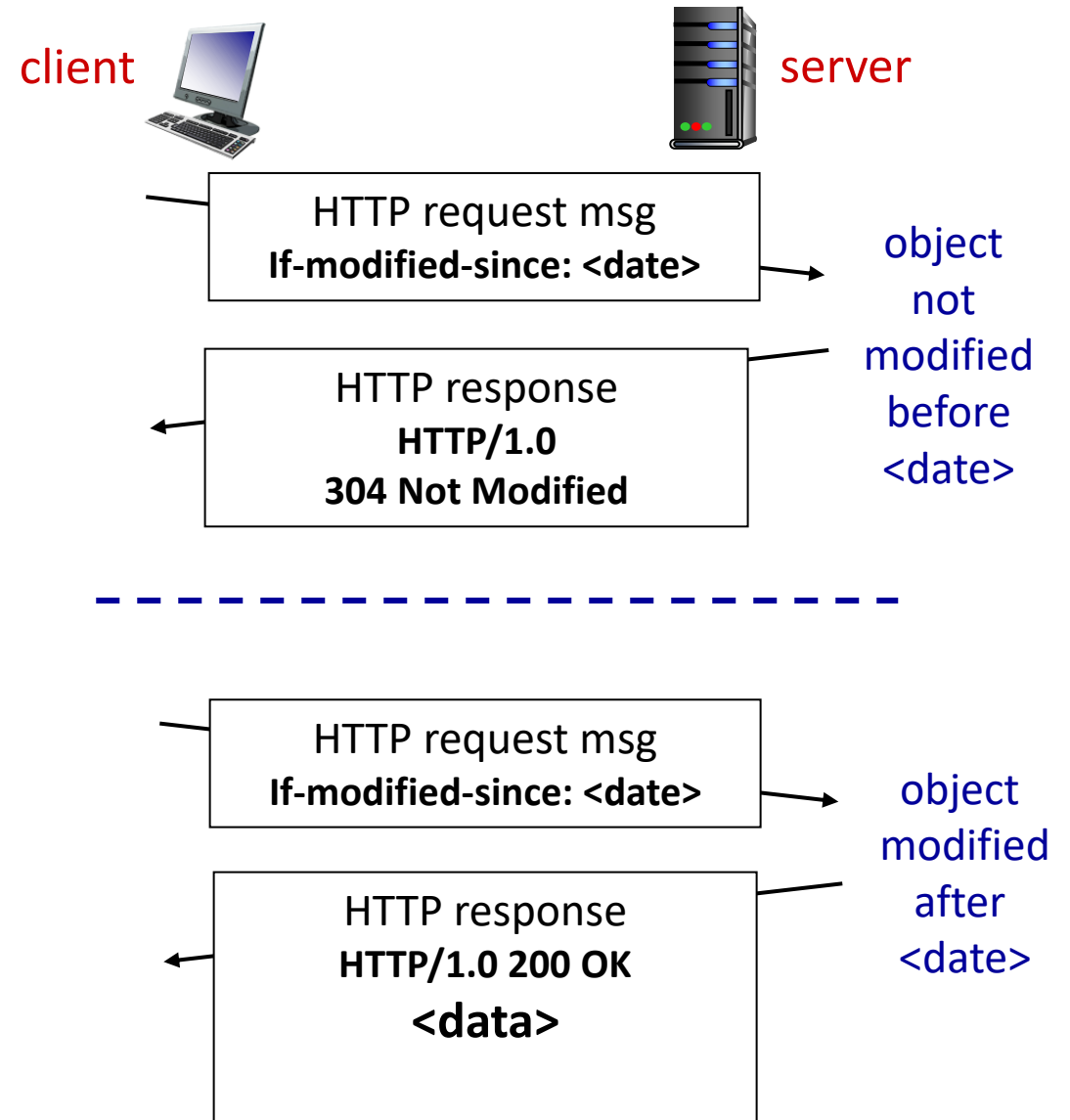
HTTP/2

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
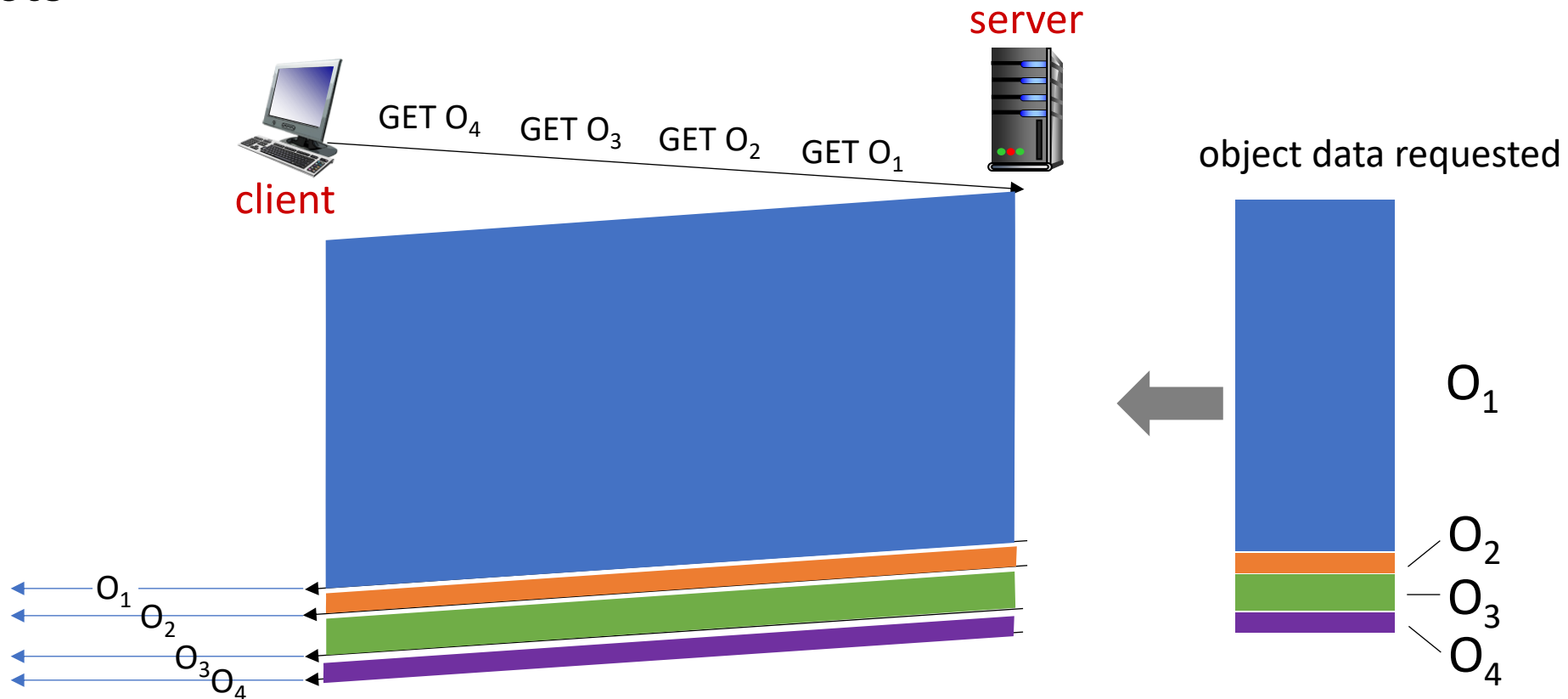
HTTP/2

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
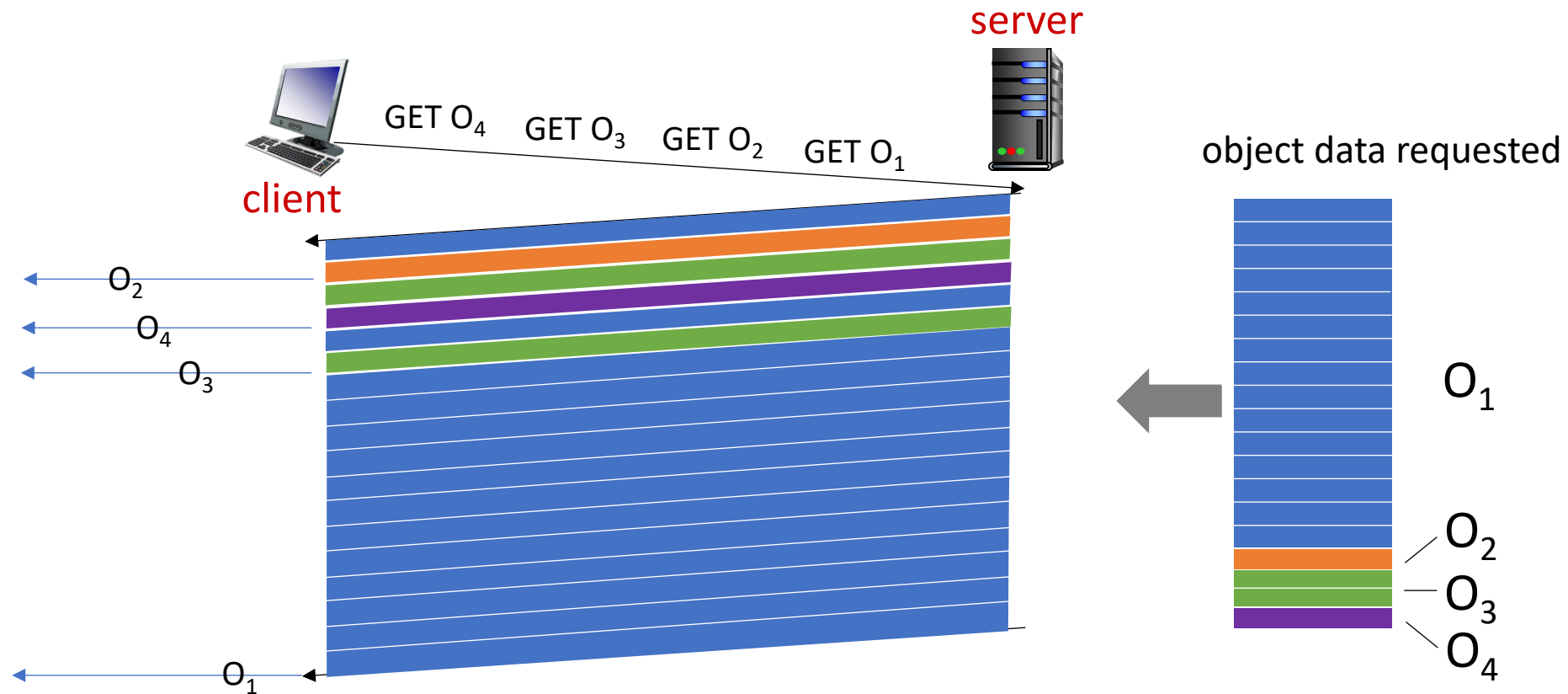
HTTP/2: mitigating HOL blocking

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
- E-mail, SMTP, IMAP
- **The Domain Name System DNS**

- P2P applications
- video streaming and content distribution networks
- socket programming with UDP and TCP

DNS: Domain Name System

people: many identifiers:
- SSN, name, passport #

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 (DNS):

- *distributed database* implemented in hierarchy of many *name servers*
- *application-layer protocol:* hosts, DNS servers communicate to *resolve* names (address/name translation)
  - *note:* core Internet function, implemented as application-layer protocol
  - complexity at network's "edge"

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/day
- Akamai DNS servers alone: 2.2T DNS queries/day

Thinking about the DNS

humongous distributed database:
- ~ billion records, each simple

handles many *trillions* of queries/day:
- *many* more reads than writes
- *performance matters:* almost every Internet transaction interacts with DNS - msecs count!
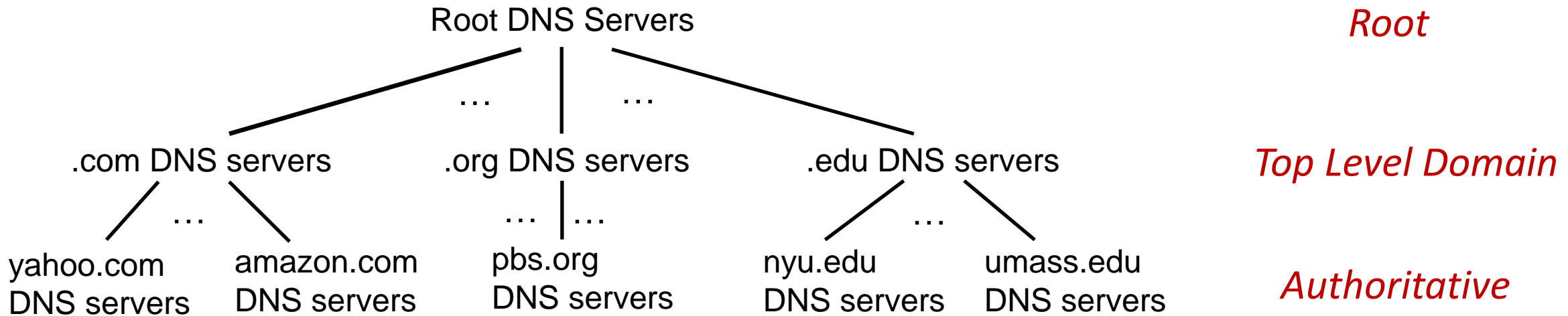
organizationally, physically decentralized:
- millions of different organizations responsible for their records

"bulletproof": reliability, security

DNS: a distributed, hierarchical database

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