Lecture 6: Application Layer

HTTP Protocol and Web proxies

COMP 411, Fall 2022

Victoria Manfredi

WESLEYAN
UNIVERSITY

Acknowledgements: materials adapted from Computer Networking: A Top Down Approach 7th edition: ©1996-2016, J.F Kurose and K.W. Ross, All Rights Reserved as well as from slides by Abraham Matta at Boston University and some material from Computer Networks by Tannenbaum and Wetherall.

Today

Announcements

- homework 2 due Friday by 5p
- loopback interface in Wireshark

Web and HTTP

- non-persistent vs. persistent connections
- request and response messages
- web caching
  - homework 3 and 4 will implement a version of this

HTTP Protocol

NON-PERSISTENT VS. PERSISTENT CONNECTIONS

HTTP connections

2 ways to use HTTP requests to get objects from web server

1. Non-persistent HTTP

- at most one object sent over TCP connection
  - connection then closed
- for each object, setup and use separate TCP connection
  - downloading multiple objects requires multiple connections
- HTTP/1.0

2. Persistent HTTP

- multiple objects can be sent over single TCP connection between client, server
- reuse same TCP connection to download multiple objects
- HTTP/1.1: by default

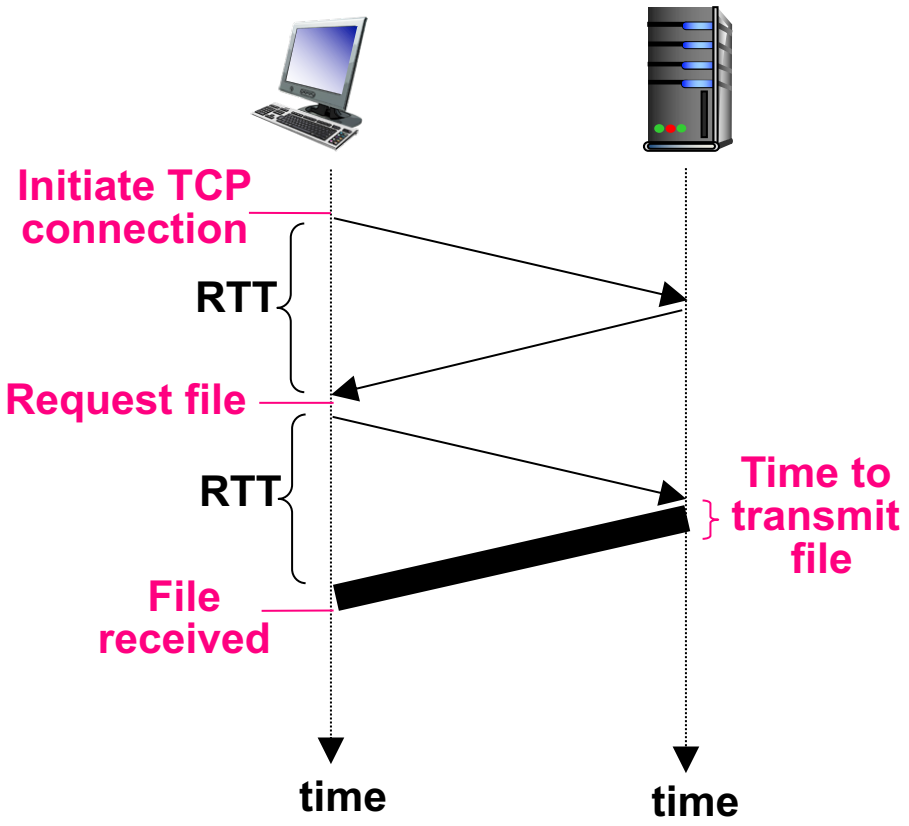
Non-persistent HTTP response time

Round-trip-time (RTT)

- time for small packet to travel from client to server and back

HTTP response time

- **1 RTT**
  - to initiate TCP connection
- **1 RTT**
  - for HTTP req and first few bytes of HTTP resp to return
- **file transmission time**



Delay and resource usage

- requires 2 RTTs + file tx time per object
- OS must work and allocate host resources for each TCP connection
- browsers often open parallel TCP connections to fetch objects

Persistent HTTP

Server leaves connection open after sending response

- subsequent HTTP messages sent over open connection
- client sends requests as soon as it encounters referenced object

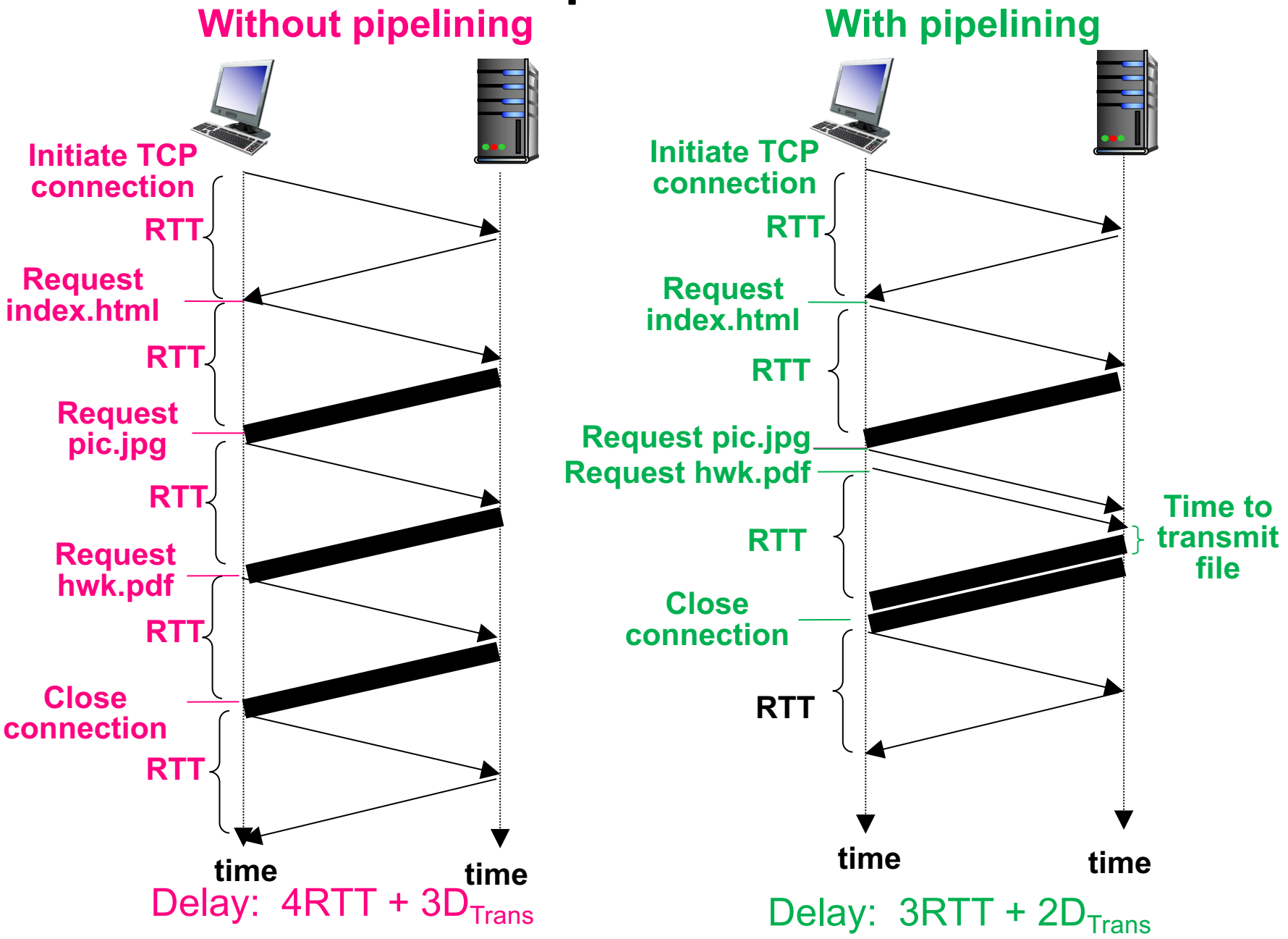
Persistent without pipelining

- client issues new request only when previous response received
- 1 RTT for each referenced object

Persistent with pipelining

- client issues new request as soon as it encounters referenced object
- as little as 1 RTT for all referenced objects
- default in HTTP/1.1

Persistent HTTP response time

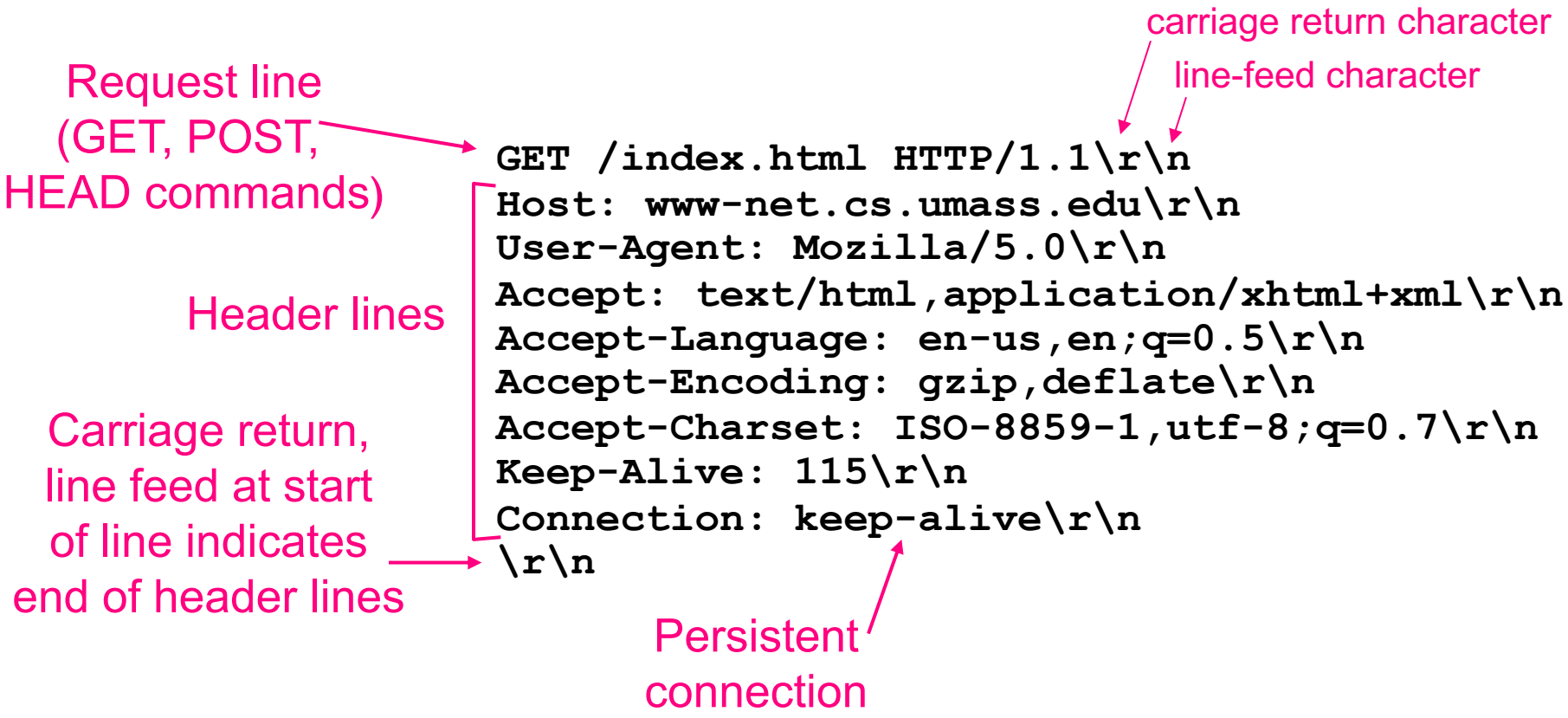


HTTP Protocol

REQUEST AND RESPONSE MESSAGES

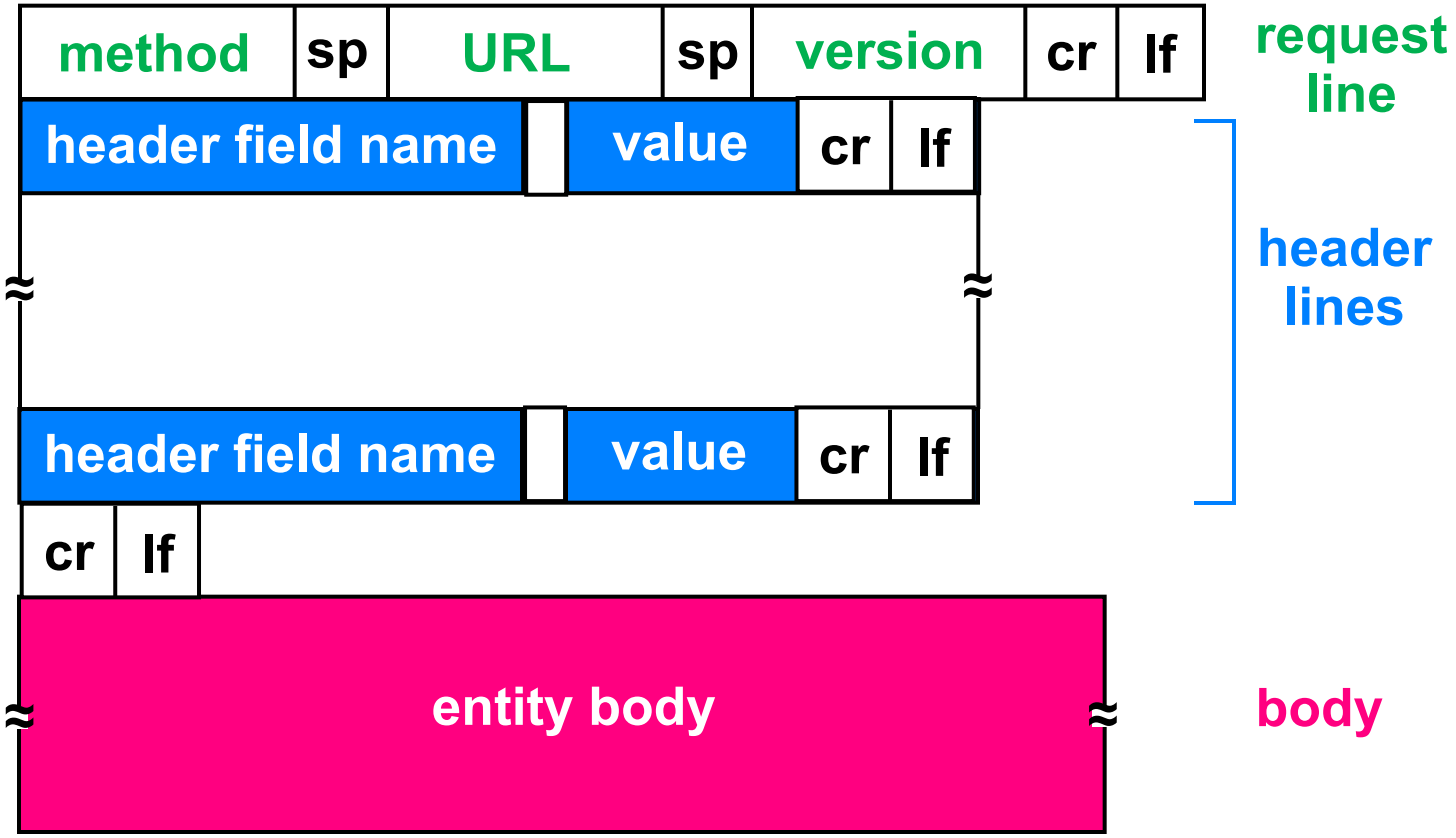
HTTP request message

ASCII (human-readable format)



Q: What info can server use to fingerprint you, without even using cookies?

HTTP request format



Uploading form input

POST method

- web page often includes form input
- input is uploaded to server in entity body

URL method

- uses GET method
- input is uploaded in URL field of request line:

```
www.somesite.com/animalsearch?monkeys&banana
```

HTTP response message

Status line (protocol status code status phrase)

Header lines

Use to determine end of message

Data, e.g., requested HTML file (may be split across multiple pkts

```
HTTP/1.1 200 OK\r\n
Date: Sun, 26 Sep 2010 20:09:20 GMT\r\n
Server: Apache/2.0.52 (CentOS)\r\n
Last-Modified: Tue, 30 Oct 2007 17:00:02
   GMT\r\n
ETag: "17dc6-a5c-bf716880"\r\n
Accept-Ranges: bytes\r\n
Content-Length: 2652\r\n
Keep-Alive: timeout=10, max=100\r\n
Connection: Keep-Alive\r\n
Content-Type: text/html; charset=ISO-8859-
   1\r\n
\r\n
data data data data data ...
```

HTTP response status codes

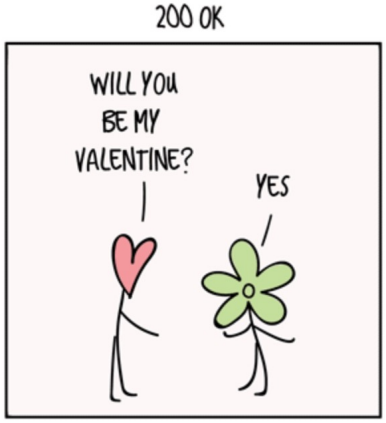
Status code

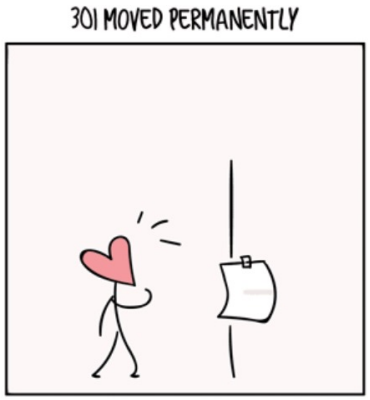
- appears in 1st line in server-to-client response message.

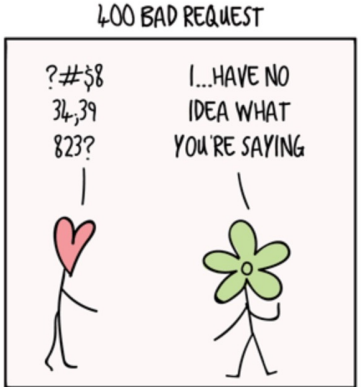
Some sample codes

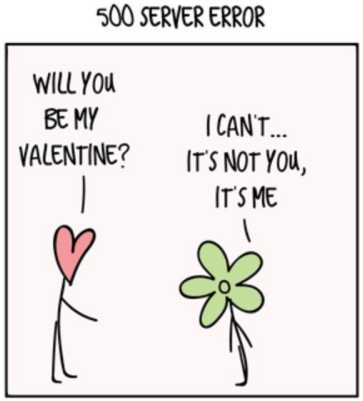
- 200 OK
  - request succeeded, requested object later in this msg
- 301 Moved Permanently
  - requested object moved, new location specified later in this msg (Location:)
- 400 Bad Request
  - request msg not understood by server
- 404 Not Found
  - requested document not found on this server
- 500 Server error
- 505 HTTP Version Not Supported

HTTP status codes as Valentine's Day cartoons









From https://medium.com/@hanilim/http-codes-as-valentines-day-comics-8c03c805faa0

Try out HTTP (client side) for yourself

1. Open tcp connection using netcat:

```
nc wesleyan.edu 80
```

Opens TCP connection to port 80 (default HTTP server port) at wesleyan.edu. Anything typed in will be sent to port 80 at wesleyan.edu

2. Type in a GET HTTP request:

```
GET /mathcs/index.html HTTP/1.1
Host: wesleyan.edu
```

By typing this in (hit carriage return twice), you send this minimal (but complete) GET request to HTTP server

3. Look at response message sent by HTTP server!

(or use Wireshark to look at captured HTTP request/response)

Netcat: useful for testing

Be a TCP server: listen for connections on port 51234

- nc -l 51234

Be a TCP client: connect to port 51234 on locahost

- nc localhost 51234
- type a string and press enter: you should see it show up at server
- type a string at server and press enter: you should see it at client

Look at connections you created

- netstat | grep 51234

Create a chat app with nc:

- nc -l 5000 on one machine with ip addr x
- nc x 5000 on another machine

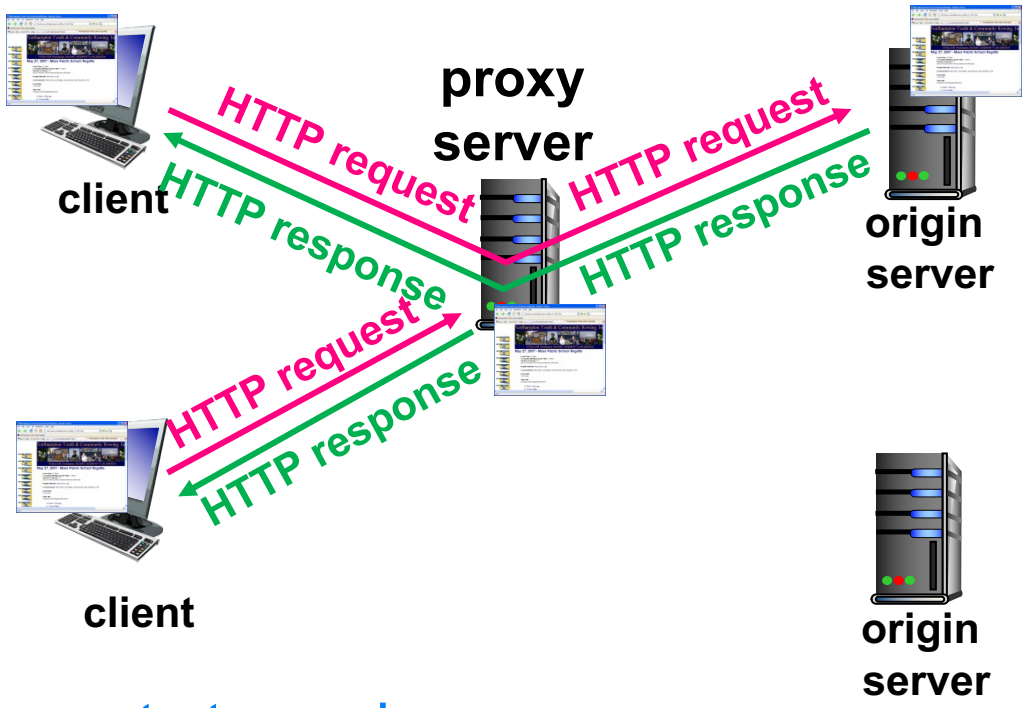
HTTP Protocol

WEB CACHING

Web caches (proxy server)

Goal

- satisfy client request without (really) involving origin server



How?

- user sets browser to perform web accesses via cache

Browser sends all HTTP requests to cache

- if object in cache
  - cache returns object
- else
  - cache requests object from origin server, then returns object to client

More about Web caching

Cache acts as both client and server

- server for original requesting client
- client to origin server

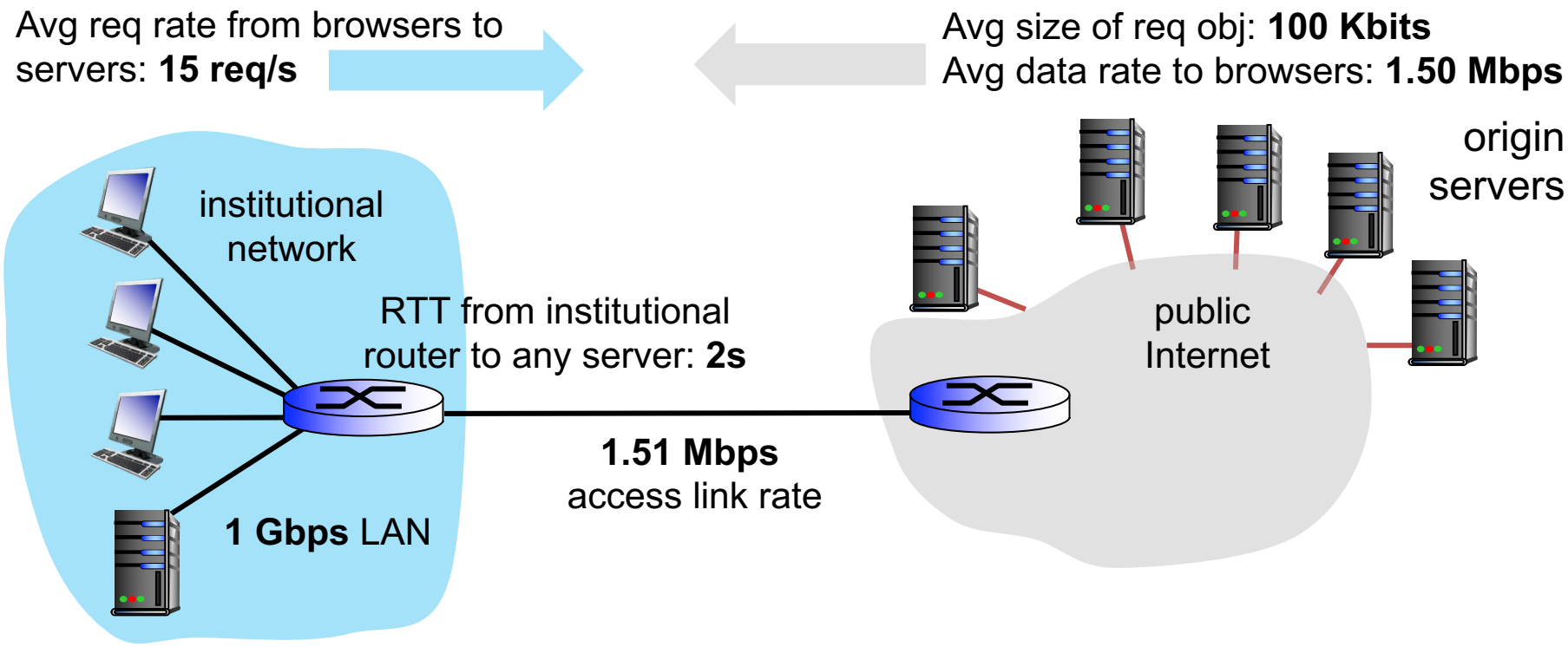
Typically cache installed by ISP

- university, company, residential ISP

Q: why use web caching?

- reduce response time for client request
- reduce traffic on institution's access link
- reduce load on origin servers
- Internet dense with caches
  - enables "poor" content providers to effectively deliver content (so too does P2P file sharing)

Example



LAN utilization: 1.5Mbps/1Gbps = 0.15%, assume ~ μsec

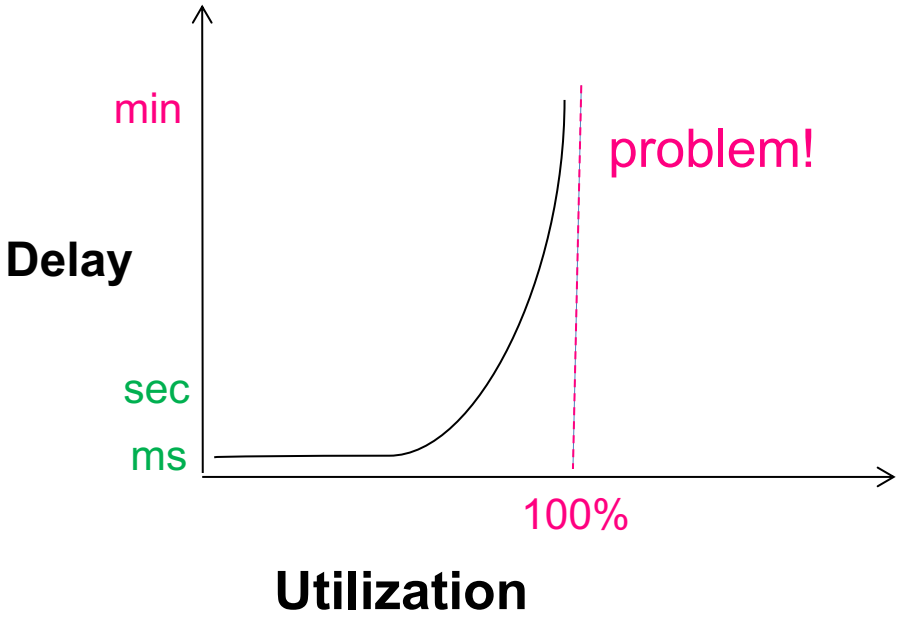
Access link utilization: 1.50/1.51 = 99%

Total delay = LAN delay + access delay + Internet delay

= μsec + minutes + 2s

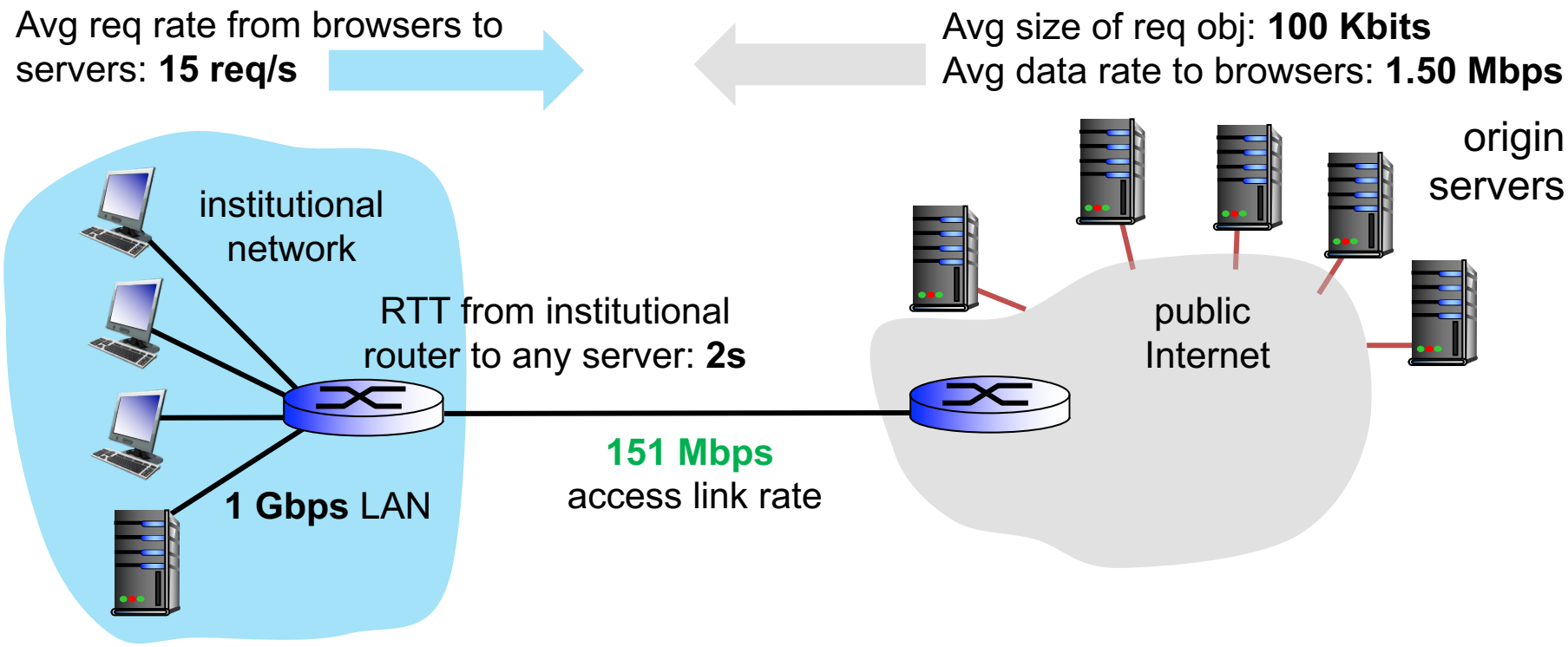
Delay as a function of utilization

Grows exponentially…



Why 99% access link utilization is bad!
What can we do?

Increase access link rate



LAN utilization: 1.5Mbps/1Gbps = 0.15%, assume ~ μsec

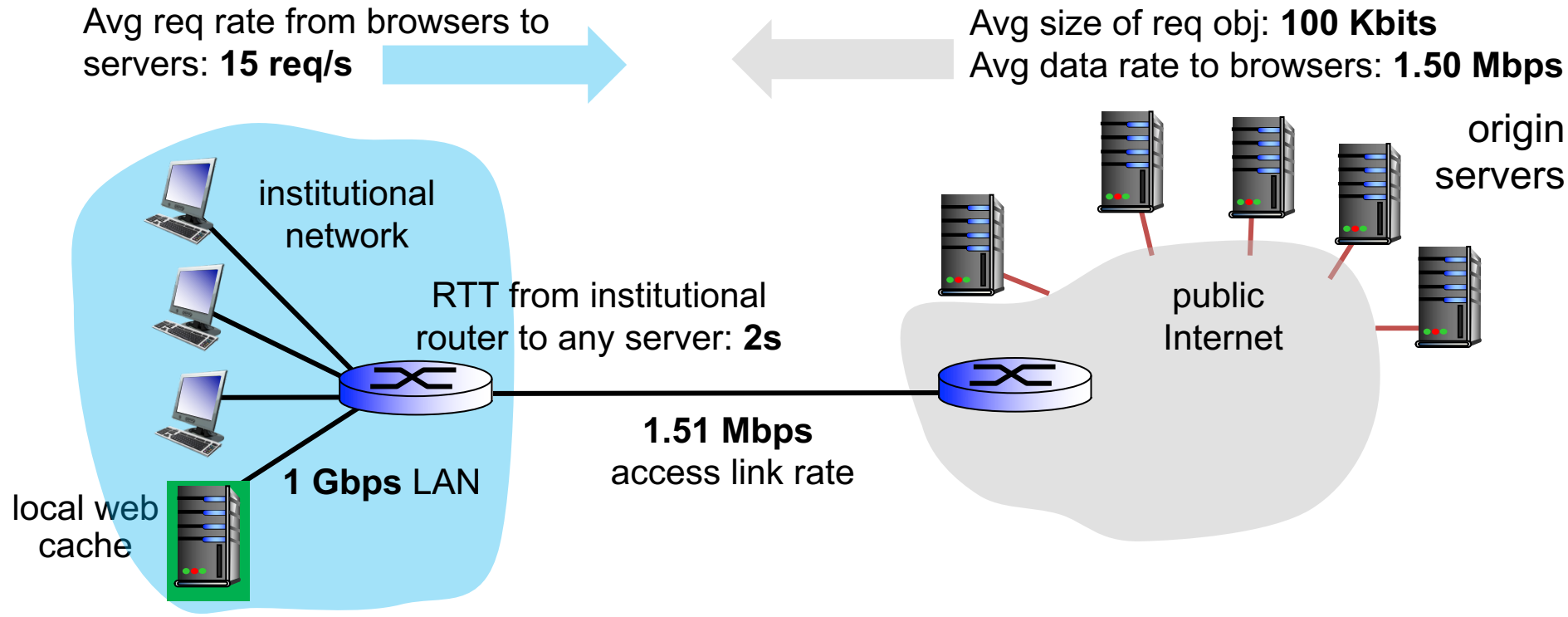
Access link utilization: 1.50/151 = 99%

Total delay = LAN delay + access delay + Internet delay

= μsec + ms + 2s

But, increasing access link rate is expensive!

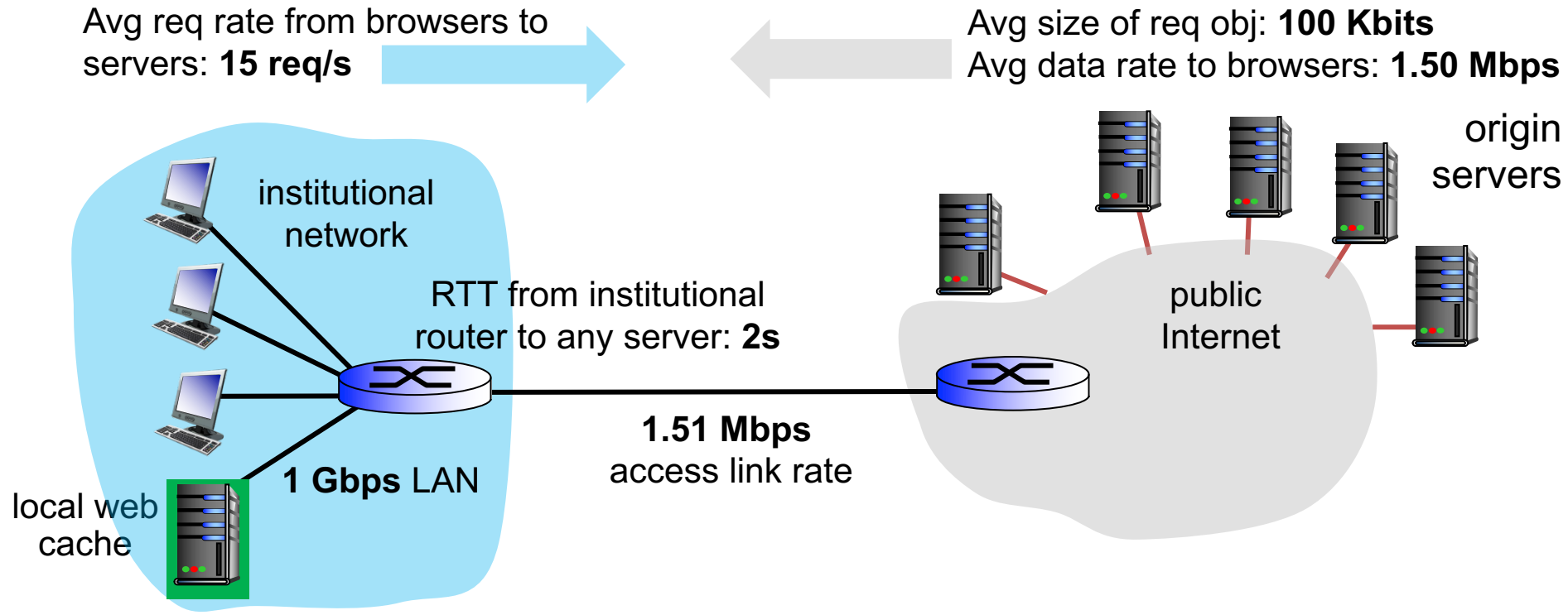
Install local cache



Web cache is cheap!

How to compute access link utilization and delay?

Access link utilization and delay with cache



Assume cache hit rate is **0.4**

- **40%** of requests satisfied at cache
- **60%** of requests satisfied at server
- thus, 60% of requests use access link

Data rate to browsers over access link

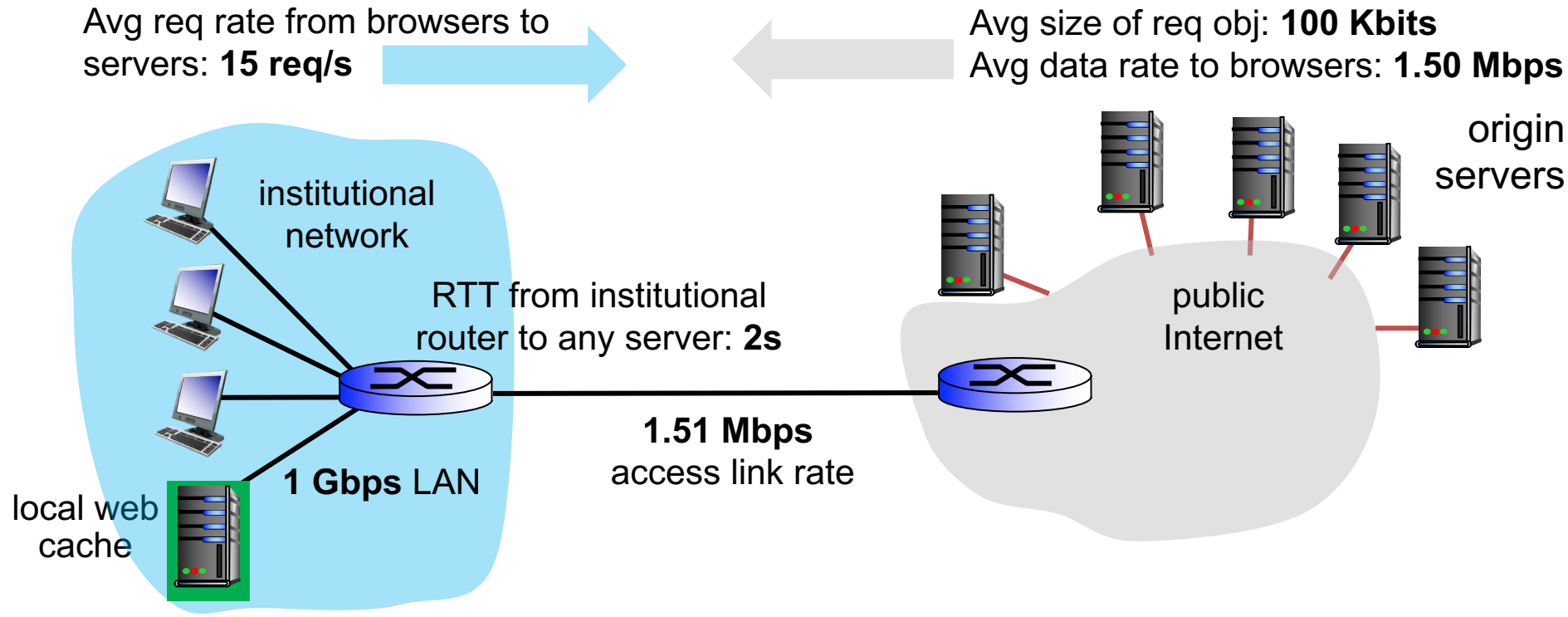
- **0.6** x 1.50 Mbps = 0.9 Mbps

Access link utilization

- 0.9 Mbps /1.51 Mbps = 60%

Assume access delay: **~700 msec**

Total delay with cache



Total delay

= 0.6 x (delay when satisfied by servers) + 0.4 x (delay when satisfied by cache)

= 0.6 x (LAN delay + access delay + Internet delay) + 0.4 x (LAN delay)

= 0.6 (**μsec** + **700 msec** + **2 sec**) + 0.4 (**μsec**)

= 0.6 (**2.7 sec**) + 0.4 (**μsec**) = **~1.6 sec**

Conditional GET

Goal

- don't send object if cache has up-to-date version
- no object transmission delay
- lower link utilization

Cache

- specify date of cached copy in HTTP request

```
If-modified-since:<date>
```

Server

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

```
HTTP/1.0 304 Not Modified
```