

# The Application Layer: HTTP

CS 352, Lecture 4, Spring 2020

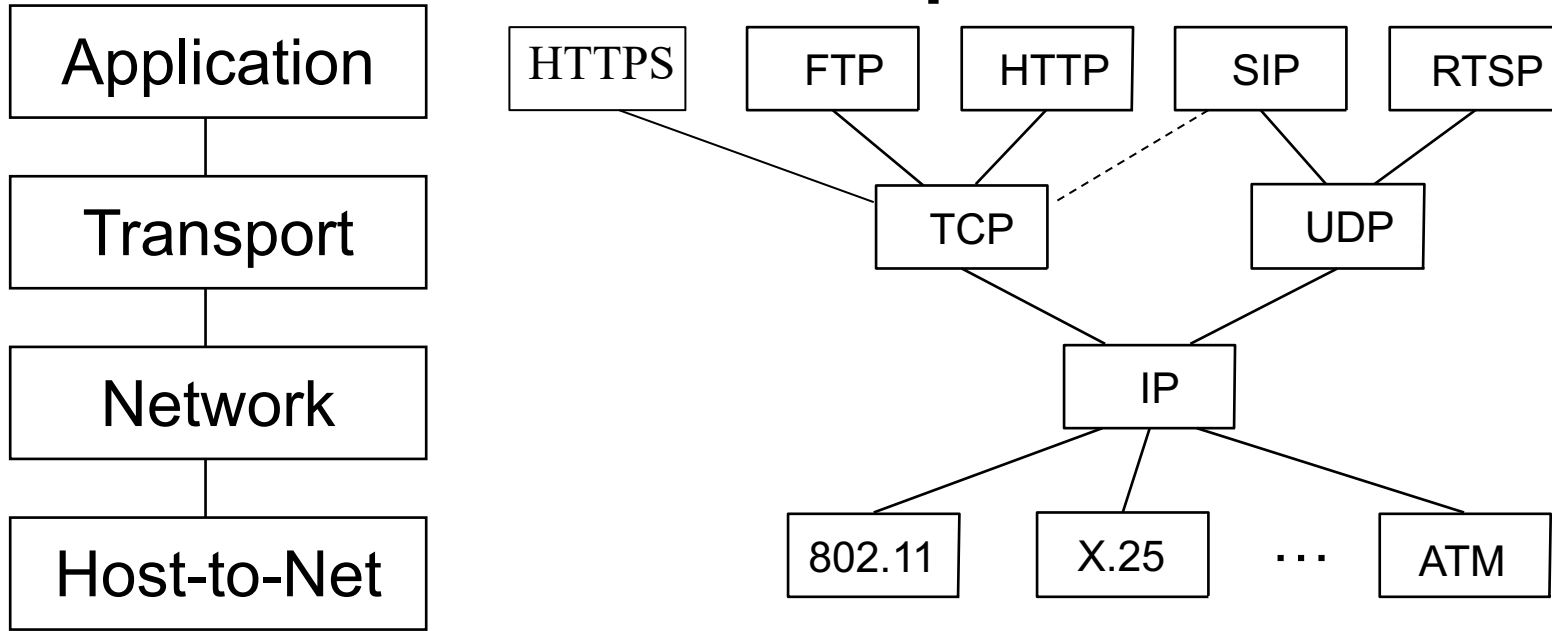
<http://www.cs.rutgers.edu/~sn624/352>

Srinivas Narayana

# Course announcements

- First quiz will go online on Sakai later today
  - Due Tuesday at 10 PM
  - Can take it from anywhere. 30 minutes time limit
  - Please don't reveal quiz questions to each other until Wednesday
    - including on Piazza
- All first week recitations done
  - Please contact me if you had any difficulties
- Find project partners
  - Project 0 (sample socket code) will be released later today on Sakai

# Review of concepts



- Layering and modularity; **application layer**
- 4-tuples ( $IP_s$ ,  $port_s$ ,  $IP_d$ ,  $port_d$ ), socket
- Client-server, peer to peer architectures
- Directory services: map name to IP with Domain Name System
  - Iterative and recursive queries

# DNS records

DNS: distributed db storing resource records (**RR**)

RR format: (**name**, **type**, **class**, **ttl**, **addr**)

Type=A

- ❖ **name** is hostname
- ❖ **value** is IP address

Type=AAAA

- ❖ **name** is hostname
- ❖ **value** is IPv6 address

• Type=NS

- **name** is domain (e.g. foo.com)
- **value** is hostname of authoritative name server for this domain

Type=CNAME

- ❖ **name** is alias name for some “canonical” (the real) name  
www.ibm.com is really  
serveeast.backup2.ibm.com
- ❖ **value** is canonical name

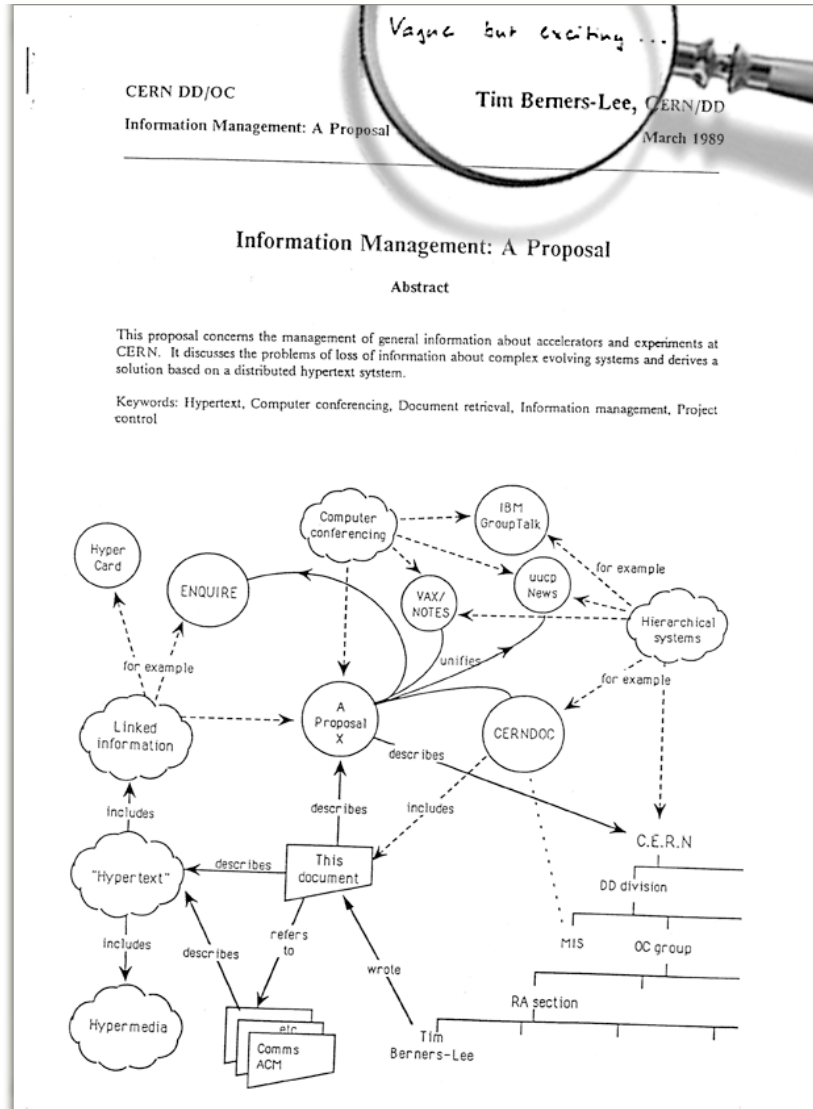
Type=MX

- ❖ **value** is name of mailserver associated with **name**

# Some themes from DNS

- Request/response nature of the protocol
- ASCII-based message structures
  - You can read and interpret the messages in natural text
  - Many protocols have binary-encoded messages
- Tricks for scaling:
  - Distribution
  - Hierarchy
  - Caching
- Many commonalities with our next application-layer protocol: HTTP
  - HyperText Transfer Protocol: the protocol of the web

# The Web: Humble origins



Tim Berners-Lee: a way to manage and access documents at CERN research lab

His boss is said to have written on his proposal: “vague, but exciting”

# Web and HTTP: Some terms

- Web page consists of **objects**
- Object can be HTML file, JPEG image, video stream chunk, audio file,...
- Web page consists of **base HTML-file** which includes several referenced objects
- Each object is addressable by a **URL (uniform resource locator)**
- Example URL:

`www.cs.rutgers.edu/~netid/index.php`

host name

path name

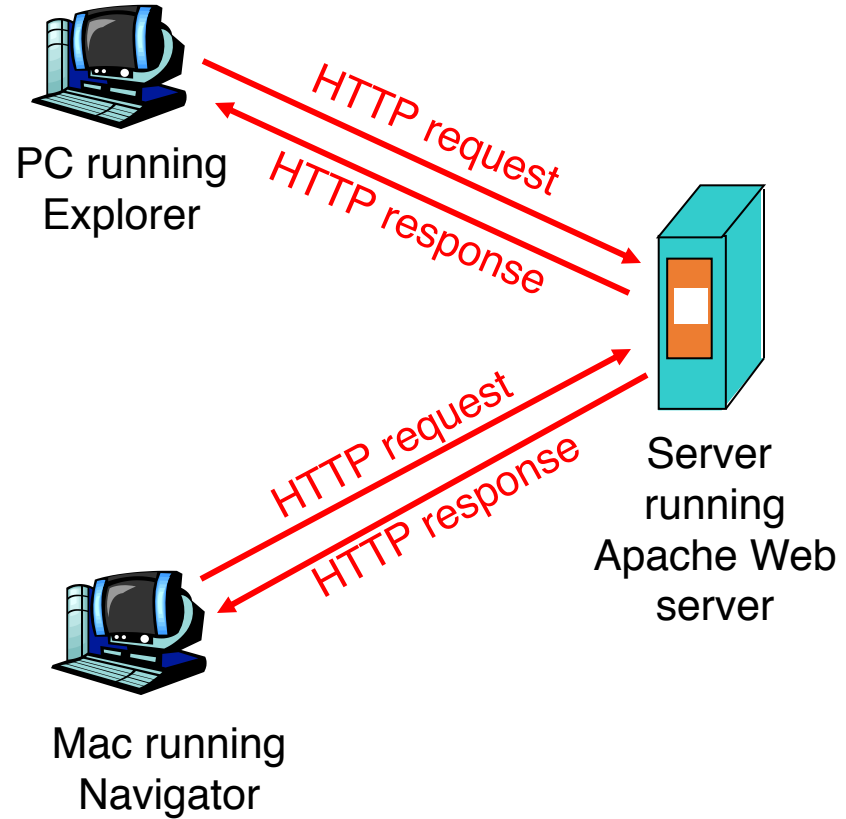
# HTTP Protocol Overview



# HTTP overview

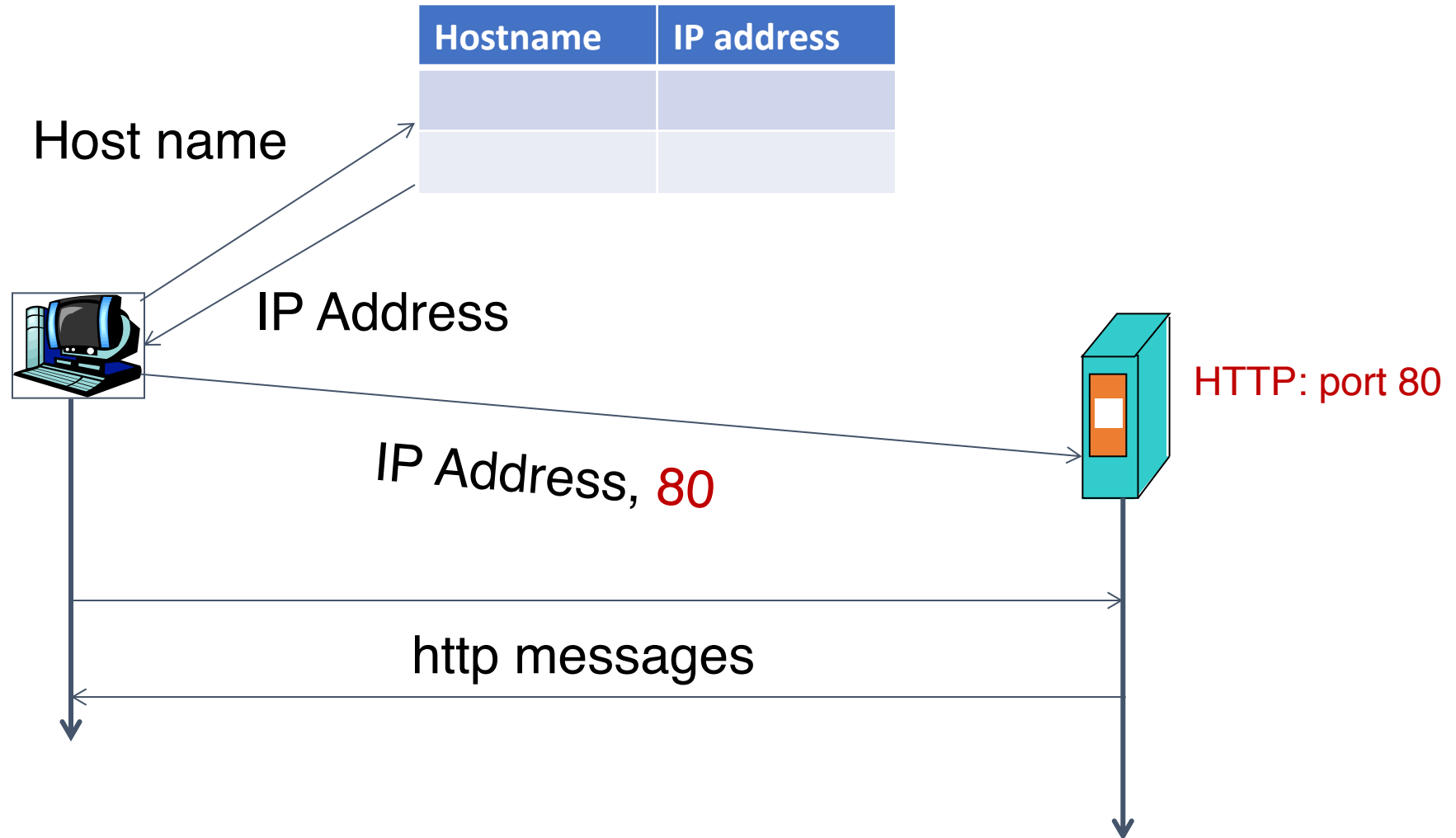
## HTTP: hypertext transfer protocol

- client/server model
  - *Client*: browser that requests, receives, “displays” Web objects
  - *Server*: Web server sends objects in response to requests
- HTTP 1.0: RFC 1945
- HTTP 1.1: RFC 2068



# Client server connection

DNS



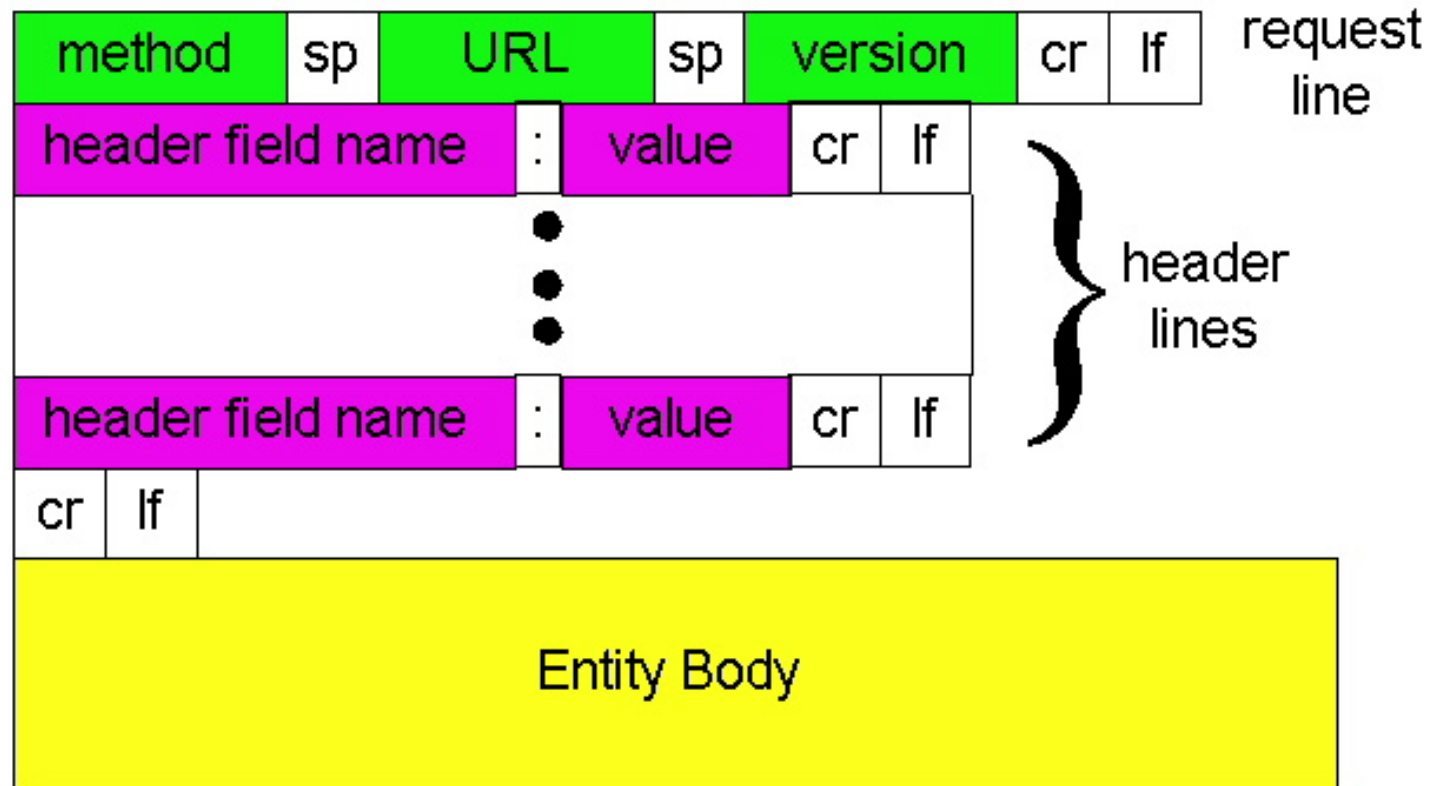
# HTTP messages: request message

- **HTTP request message:**
  - ASCII (human-readable format)

The diagram illustrates the structure of an HTTP request message. It consists of a request line, header lines, and a final carriage return/line feed. Annotations with arrows point to each part: 'request line (GET, POST, HEAD commands)' points to the first line; 'header lines' points to the subsequent four lines; and 'Carriage return, line feed indicates end of message' points to the final line.

```
request line  
(GET, POST,  
HEAD commands) → GET /somedir/page.html HTTP/1.1  
header lines → Host: www.someschool.edu  
User-agent: Mozilla/4.0  
Connection: close  
Accept-language: fr  
Carriage return,  
line feed → (extra carriage return, line feed)  
indicates end  
of message
```

# HTTP request message: general format



# Method types

- **GET**
  - Get the file specified in the path URL field in entity body
- **POST**
  - accept the entity enclosed in the entity body as a new subordinate of the resource identified by the URL field
- **HEAD**
  - asks server to leave requested object out of response
- **PUT**
  - uploads file in entity body to path specified in URL field
- **DELETE**
  - deletes file specified in the URL field

# Uploading form input: GET and POST

## POST method:

- Web page often includes form input
- Input is uploaded to server **in entity body**
- Posted content not visible in the URL
  - Free form content (ex: images) can be posted since entity body interpreted as data bytes

## GET method:

- Entity body is empty
- Input is uploaded **in URL field of request line**
- Example:
  - `http://site.com/form?first=jane&last=austen`

# Example: Client POST request

`POST /cgi-bin/rats.cgi HTTP/1.0`

`Referer: http://nes:8192/cgi-bin/rats.cgi`

`Connection: Keep-Alive`

`User-Agent: Mozilla/4.73 [en] (X11; U; Linux 2.2.12-20 i686)`

`Host: nes:8192`

`Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*`

`Accept-Encoding: gzip`

`Accept-Language: en`

`Accept-Charset: iso-8859-1,*,utf-8`

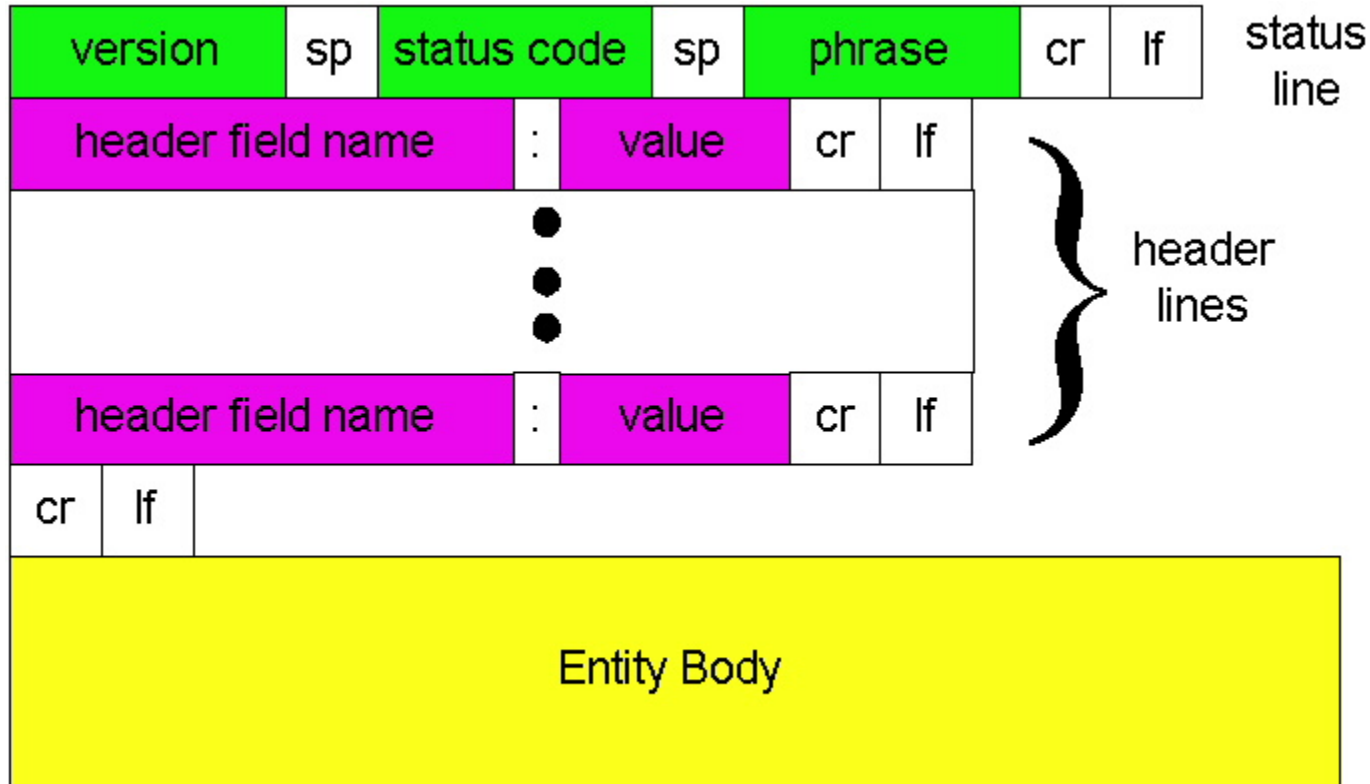
`Content-type: application/x-www-form-urlencoded`

`Content-length: 93`

`Account=cs111fall&First=Alice&Last=White&SSN=123456789&Bday=01011980&State=CreateAccount`

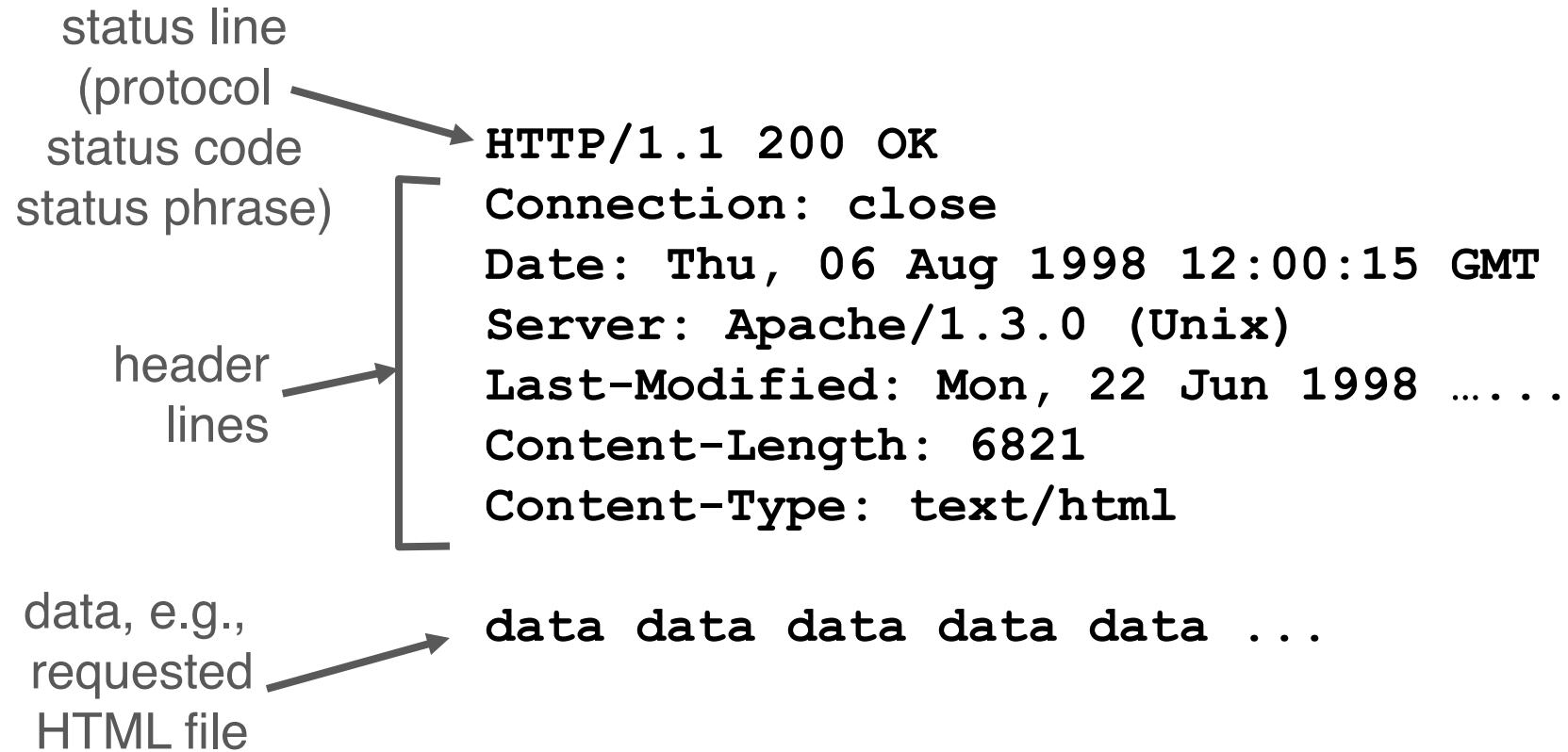
# HTTP response message: general format

Unlike HTTP  
request, No  
method  
name





# HTTP message: response message



# HTTP response status codes

In first line in server->client response message.

A few sample codes:

## **200 OK**

- request succeeded, requested object later in this message

## **301 Moved Permanently**

- requested object moved, new location specified later in this message (Location:)

## **400 Bad Request**

- request message not understood by server

## **404 Not Found**

- requested document not found on this server

## **505 HTTP Version Not Supported**

# Try out HTTP for yourself!

1. Telnet to your favorite Web server:

```
telnet web.mit.edu 80
```

Opens TCP connection to port 80 (default HTTP server port). Anything typed in sent to port 80 at web.mit.edu

2. Type in a GET HTTP request:

```
GET / HTTP/1.1  
Host: web.mit.edu
```

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

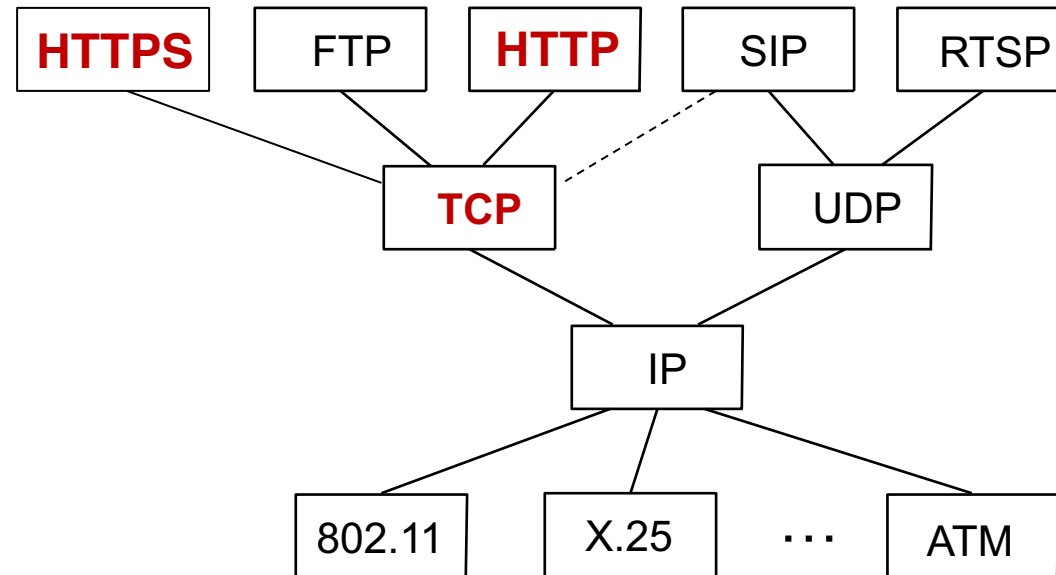
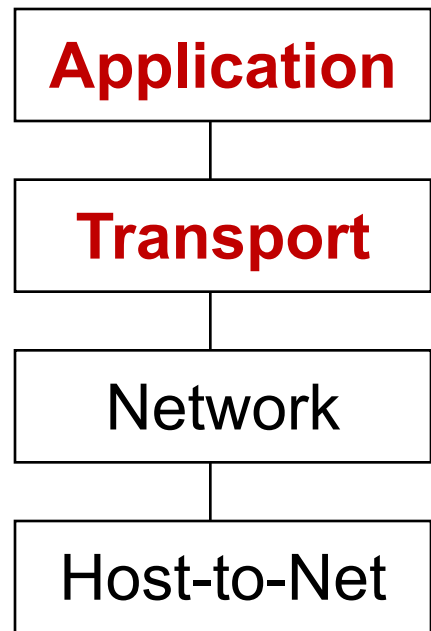
3. Look at response message sent by HTTP server!

# Additional details about HTTP

- Persistent vs. Nonpersistent HTTP connections
- Cookies (User-server state)
- Web caches

# Non/Persistent HTTP

# Recall the Internet protocol stack...



# HTTP connections

## Non-persistent HTTP

- At most one object is sent over a TCP connection.
- HTTP/1.0 uses nonpersistent HTTP

## Persistent HTTP

- Multiple objects can be sent over single TCP connection between client and server.
- HTTP/1.1 uses persistent connections in default mode

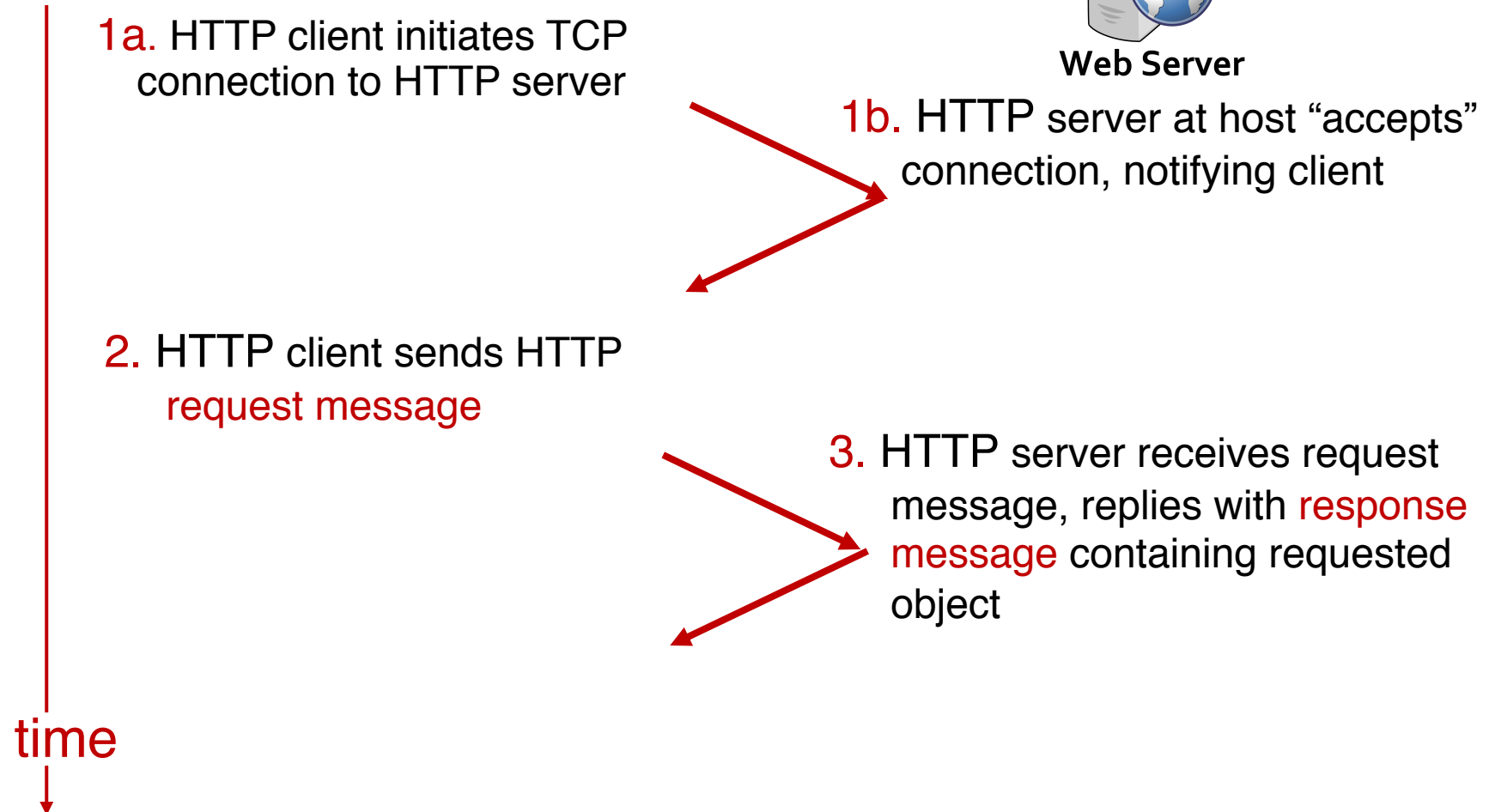
TCP is a kind of reliable communication service provided by the transport layer. It requires the connection to be “set up” before data communication.

# Non-persistent HTTP



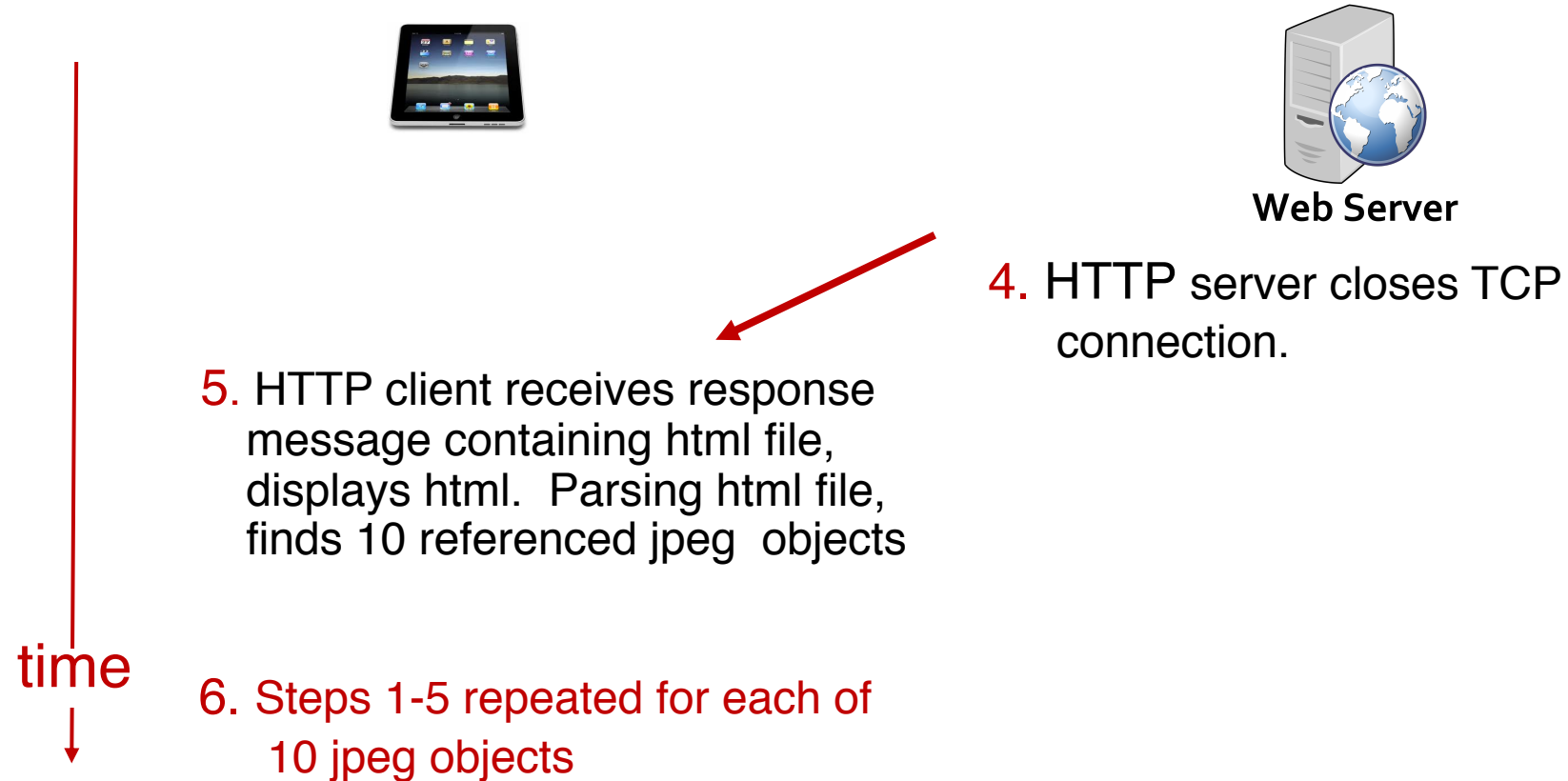
Web Server

Suppose user visits a page with text and 10 images.





# Non-persistent HTTP (contd.)



# HTTP Response time

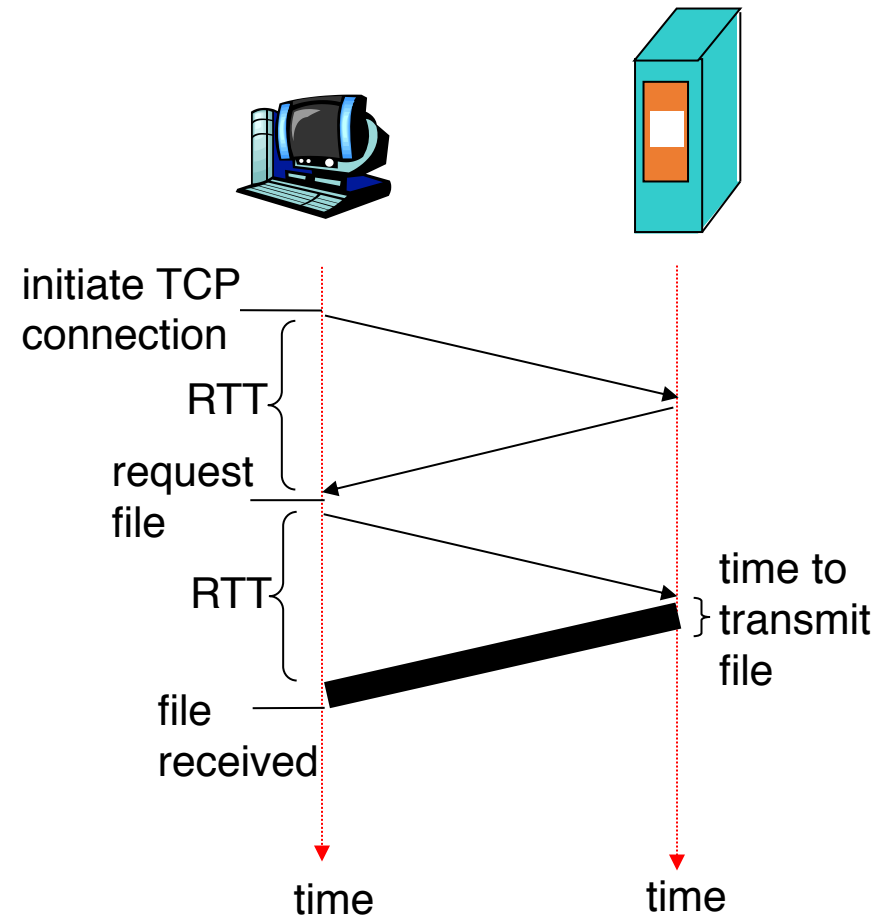
**Definition of RTT:** time to send a small packet to travel from client to server and back.

- Sum of propagation and queueing delays.

**Response time:**

- one RTT to initiate TCP connection
- one RTT for HTTP request and first few bytes of HTTP response to return
- file transmission time

**total =  $2RTT$  + transmit time**



# Persistent vs. Non-persistent

## Non-persistent HTTP issues:

- requires 2 RTTs per object
- Browsers can open parallel TCP connections to fetch referenced objects

## Persistent HTTP

- server leaves connection open after sending response
- subsequent HTTP messages between same client/server sent over open connection

# Remembering HTTP users

# HTTP: User data on servers?

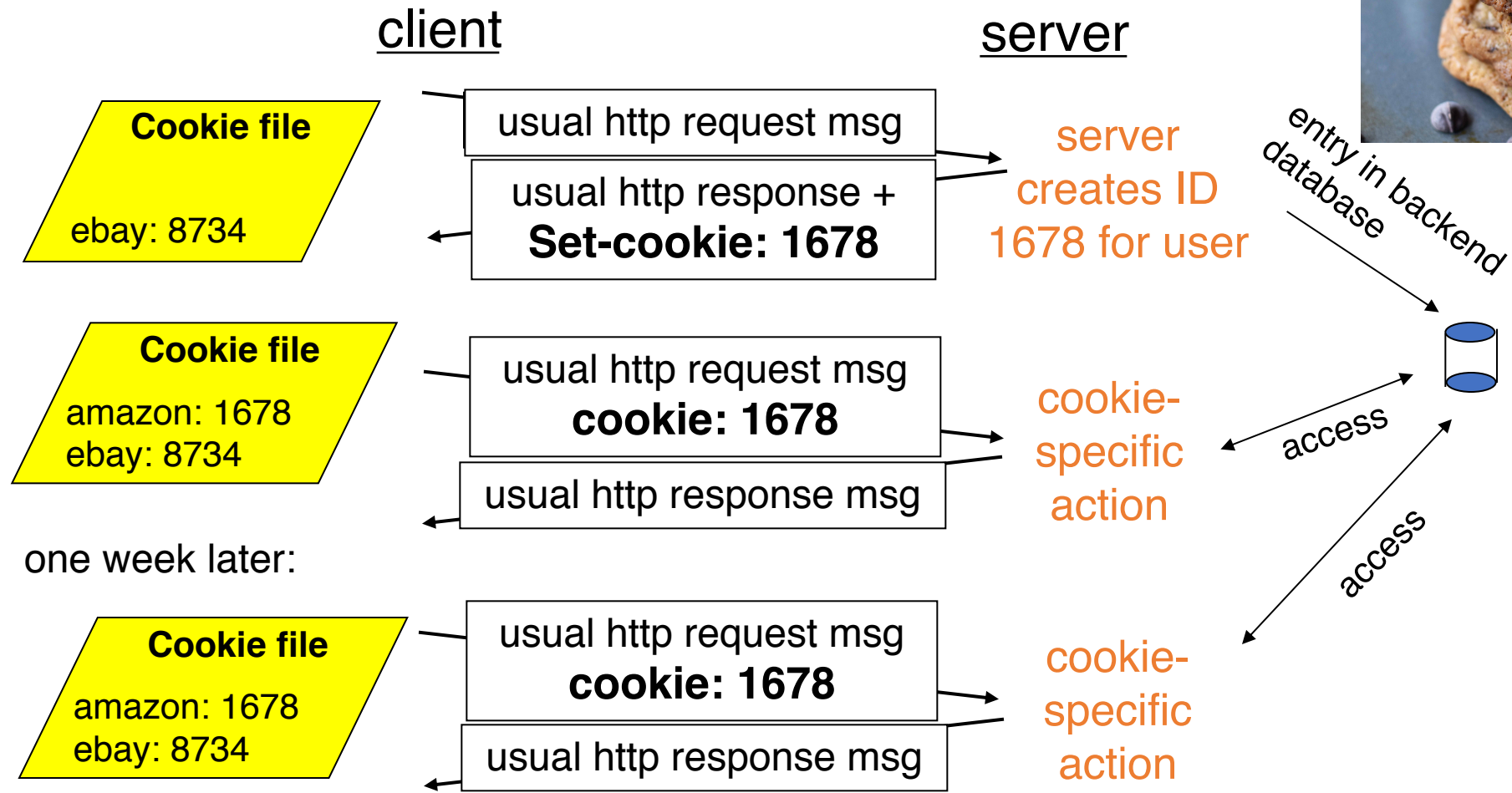
So far, HTTP is “stateless”

- The server maintains no memory about past client requests

But state, i.e., memory, about the user at the server be very useful!

- authorization
- shopping carts
- recommendations
- user session state

# Cookies: Keeping user memory



# How cookies work

## Four components:

1. cookie header line of HTTP **response** message
2. cookie header line in HTTP **request** message
3. cookie file kept on user endpoint, managed by user's browser
4. back-end database maps cookie to user data at Web endpoint

Client and server **collaboratively** track and remember the user's state.

# Cookies and Privacy

Aside

## Cookies and privacy

- cookies permit sites to learn a lot about you
- e.g., you may supply name and e-mail to sites





# Caching in HTTP

# Web caches

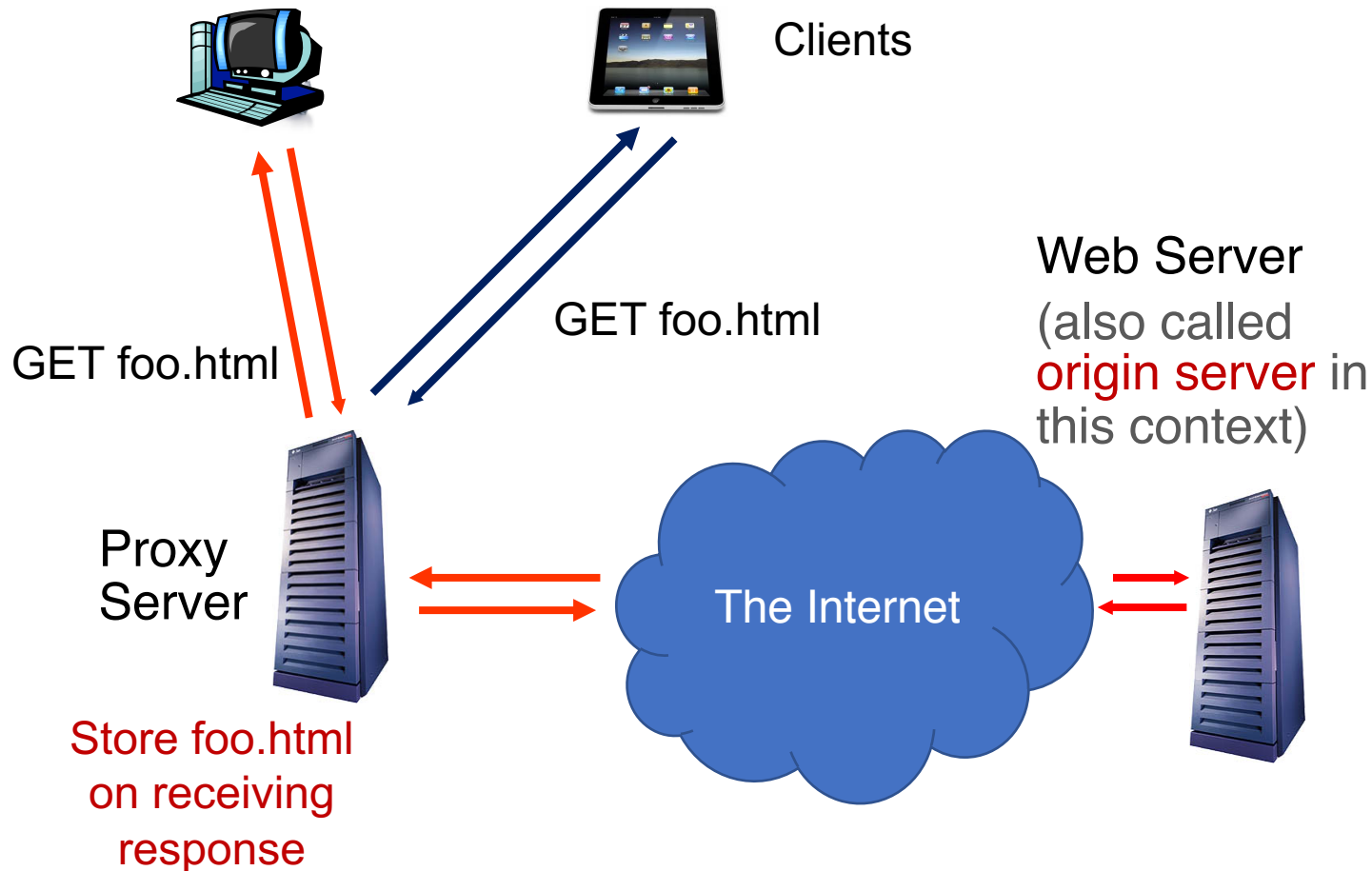
Web caches: Machines that remember web responses for a network

## Why cache web responses?

- Reduce response time for client requests
- Reduce traffic on an institution's access link

Caches can be implemented in the form of a **proxy server**

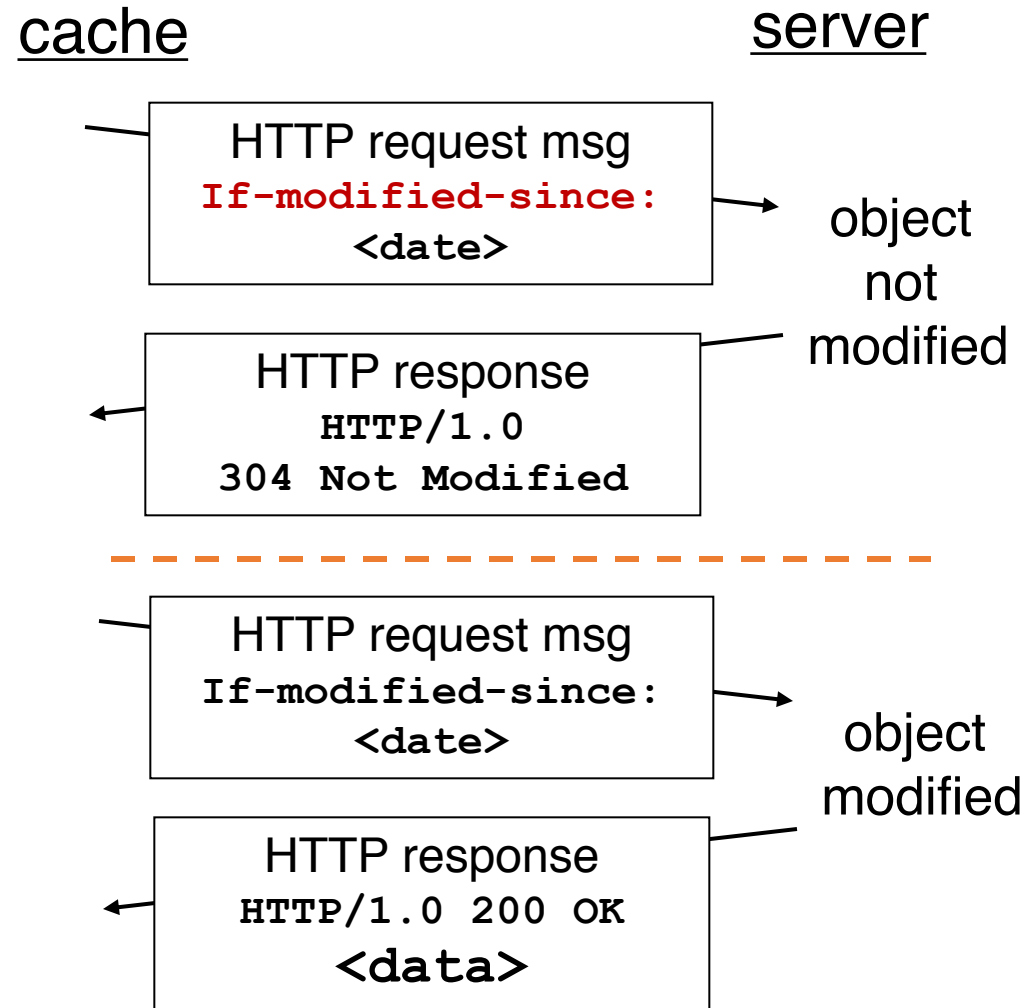
# Web caching using a proxy server



- You can configure a HTTP proxy on your laptop's network settings.
- If you do, your browser sends all HTTP requests to the proxy (cache).
- Hit: cache returns object
- Miss:
  - cache requests object from origin server
  - caches it locally
  - and returns it to client

# Web Caches: how does it look on HTTP?

- **Conditional GET**  
guarantees cache content is up-to-date while still saves traffic and response time whenever possible
- Date in the cache's request is the last time the server provided in its response header "**last modified**"



# Content Distribution Networks (CDN)

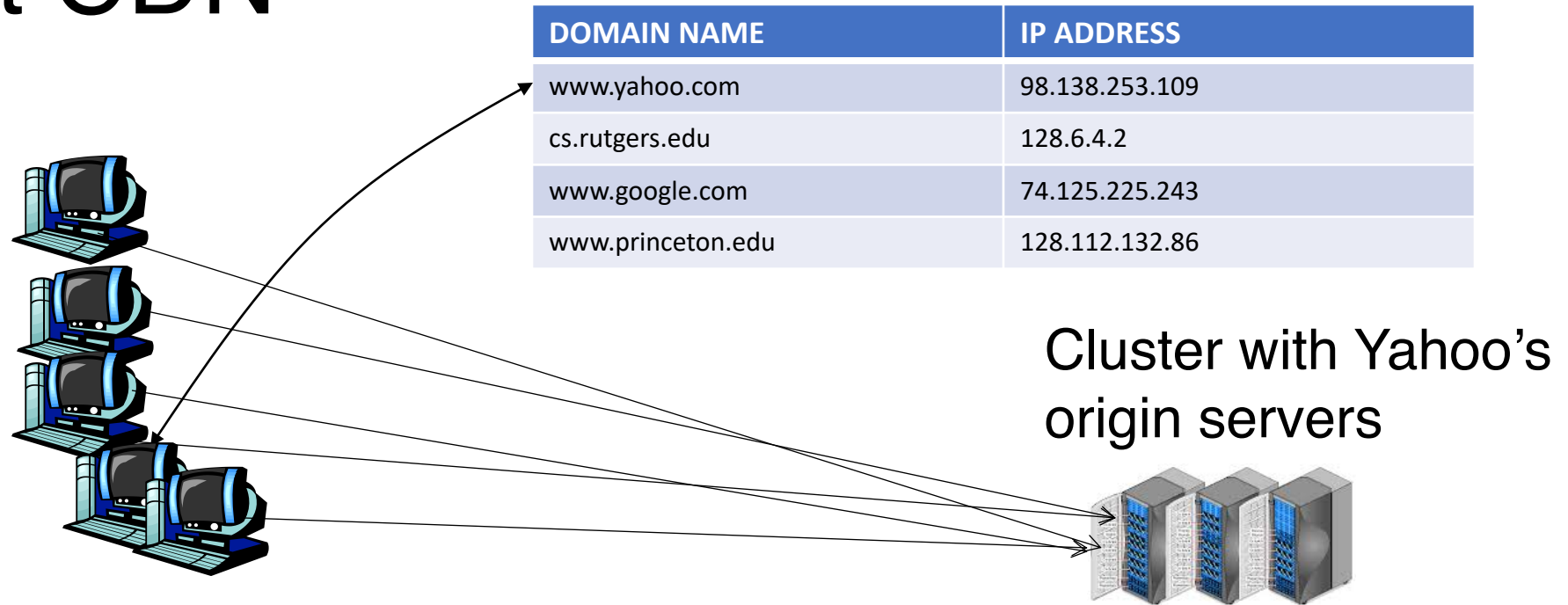
## A global network of web caches

- Provisioned by ISPs and network operators
- Or content providers, like Netflix, Google, ...

## Uses

- **Reduce bandwidth requirements** on content provider
- Reduce \$\$ to maintain origin servers
- Reduce traffic on a network's Internet connection, e.g., Rutgers
- Improve response time to user for a service

# Without CDN



- Huge bandwidth requirements
- Large propagation delays to reach users
- So, distribute content to geographically distributed cache servers.
- Often, **use DNS** to redirect request to users to copies of content

# CDN terms

- Origin server
  - Server that holds the authoritative copy of the content
- CDN server
  - A replica server owned by the CDN provider
- CDN name server
  - A DNS like name server used for redirection
- Client

# With CDN

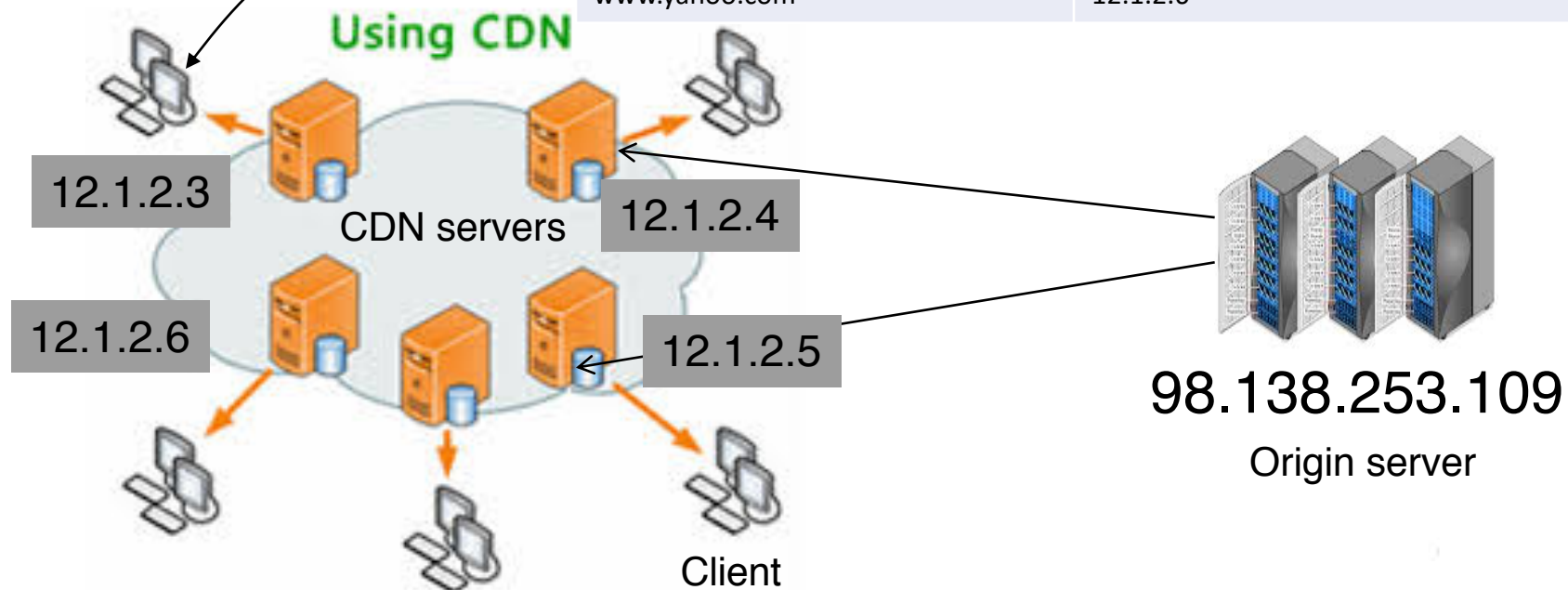
Scale through  
indirection to CDN  
name server.

| DOMAIN NAME       | IP ADDRESS            |
|-------------------|-----------------------|
| www.yahoo.com     | 124.8.9.8 (NS of CDN) |
| cs.rutgers.edu    | 128.6.4.2             |
| www.google.com    | 74.125.225.243        |
| www.princeton.edu | 128.112.132.86        |

## CDN Name Server (124.8.9.8)

| DOMAIN NAME   | IP ADDRESS |
|---------------|------------|
| www.yahoo.com | 12.1.2.3   |
| www.yahoo.com | 12.1.2.4   |
| www.yahoo.com | 12.1.2.5   |
| www.yahoo.com | 12.1.2.6   |

Custom logic  
to map ONE  
domain name  
to one of  
many IP  
addresses!





# Themes from HTTP

- Request/response nature of protocols
  - Headers determine the actions of all the parties of the protocol
- ASCII-based message structures
- Higher performance using caching
- Scaling using indirection
- These principles form of the basis of the web that we enjoy today!