# DNS

Why?

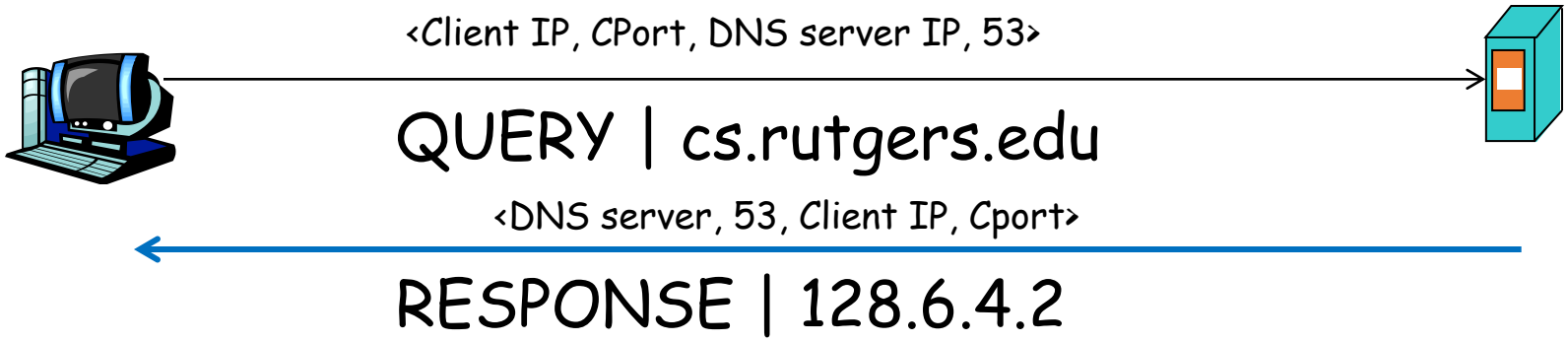
For any networked application, we need to know the IP address of a host given its name

# Domain Name System (DNS)

- Problem statement:
  - Average brain can easily remember 7 digits for a few names
  - On average, IP addresses have 12 digits
  - We need an easier way to remember IP addresses
- Solution:
  - Use names to refer to hosts
  - Just as a contact or telephone book (white pages)
  - Add a service (called DNS) to map between host names and binary IP addresses
  - We call this *Address Resolution*

# Simple DNS

| DOMAIN NAME | IP ADDRESS |
|---|---|
| WWW.YAHOO.COM | 98.138.253.109 |
| cs.rutgers.edu | 128.6.4.2 |
| www.google.com | 74.125.225.243 |
| www.princeton.edu | 128.112.132.86 |




- Simple but does not scale
- Every new host needs to be entered  in this table
- Performance? Failure?

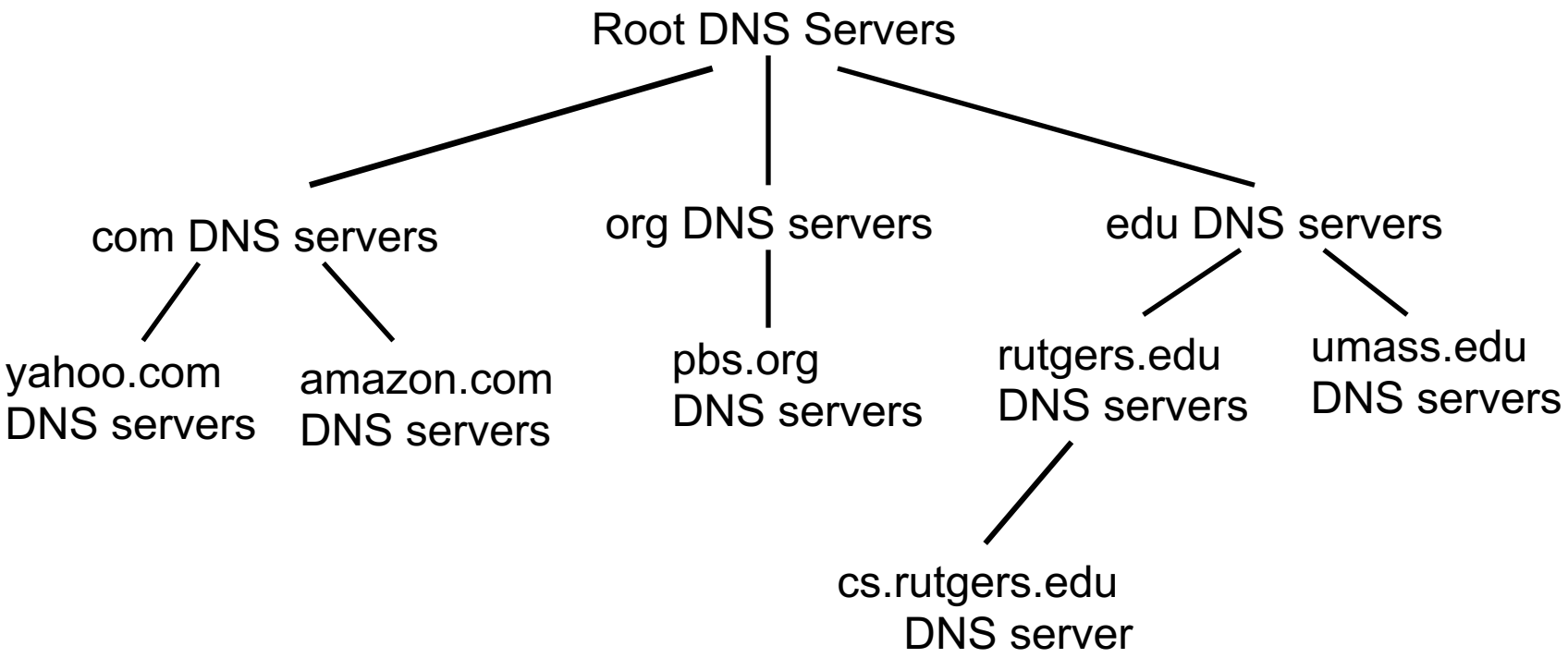
# DNS

Centralize DNS?

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

doesn't *scale!*

# Distributed, Hierarchical Database




RFC 1034

# DNS Protocol

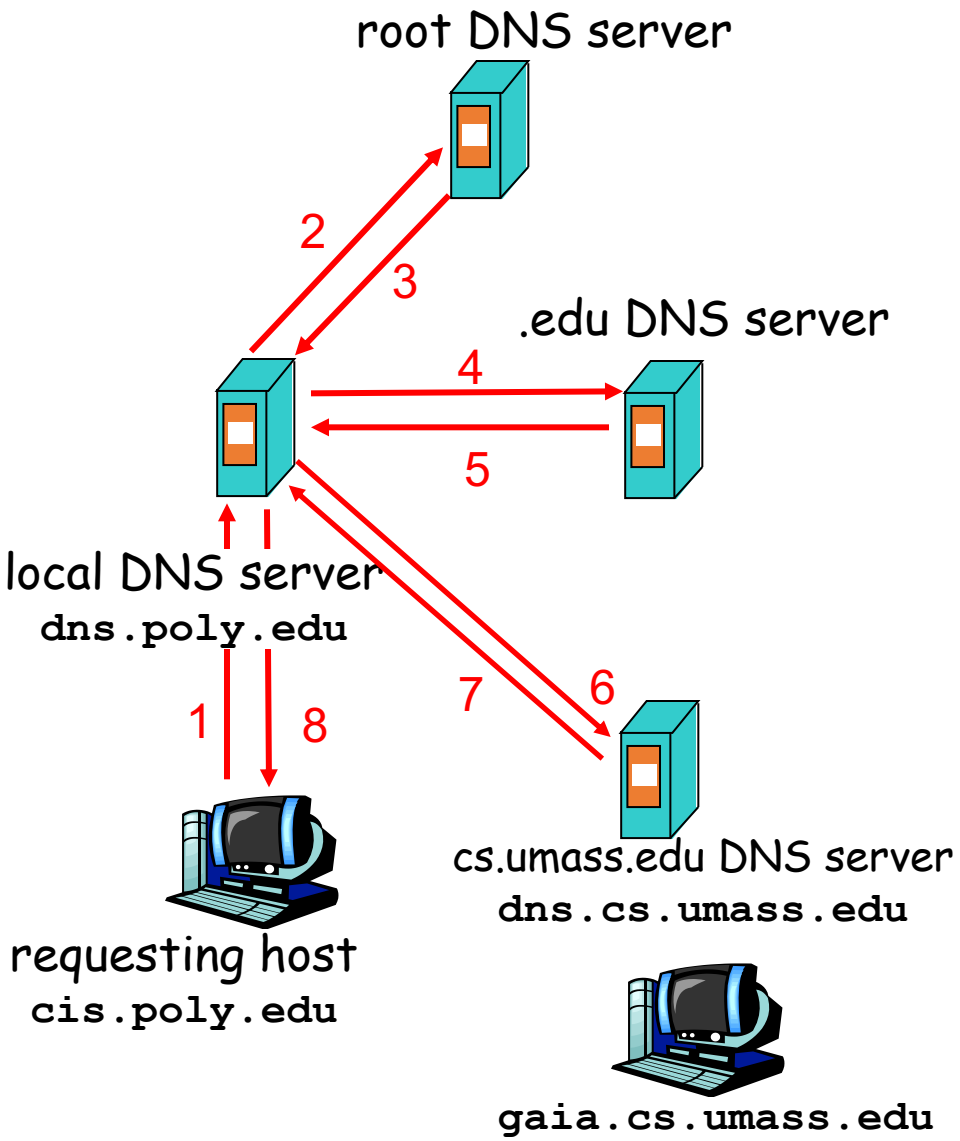
- Client and Server (CS Model)
- Client connects to Port 53
- DNS server address should be known
  - Either manually configured or automatically
- Two types of messages
  - Queries
  - Responses
- Type of Query methods
  - Standard query
    - Request IP address
  - Updates
    - Provide a binding of IP address to domain name
- Each type has a common message format that follows the header

# DNS Protocol

- When client wants to know an IP address for a host name
  - Client sends a DNS query to the primary name server in its zone
  - If name server contains the mapping, it returns the IP address to the client
  - Otherwise, the name server forwards the request to the root name server
  - The request works its way down the tree toward the host until it reaches a name server with the correct mapping

# Example

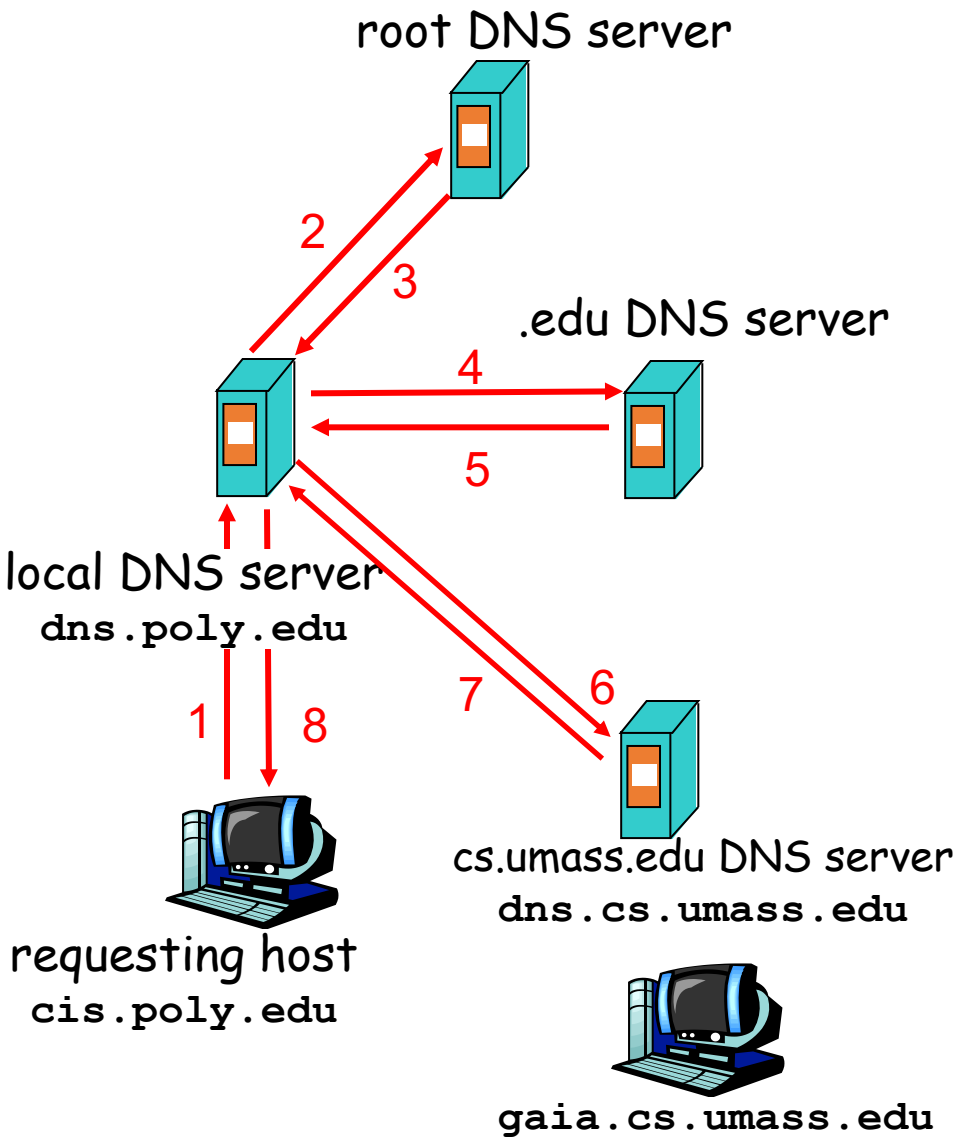
- Host at cis.poly.edu wants IP address for gaia.cs.umass.edu



# Query type

iterated query:

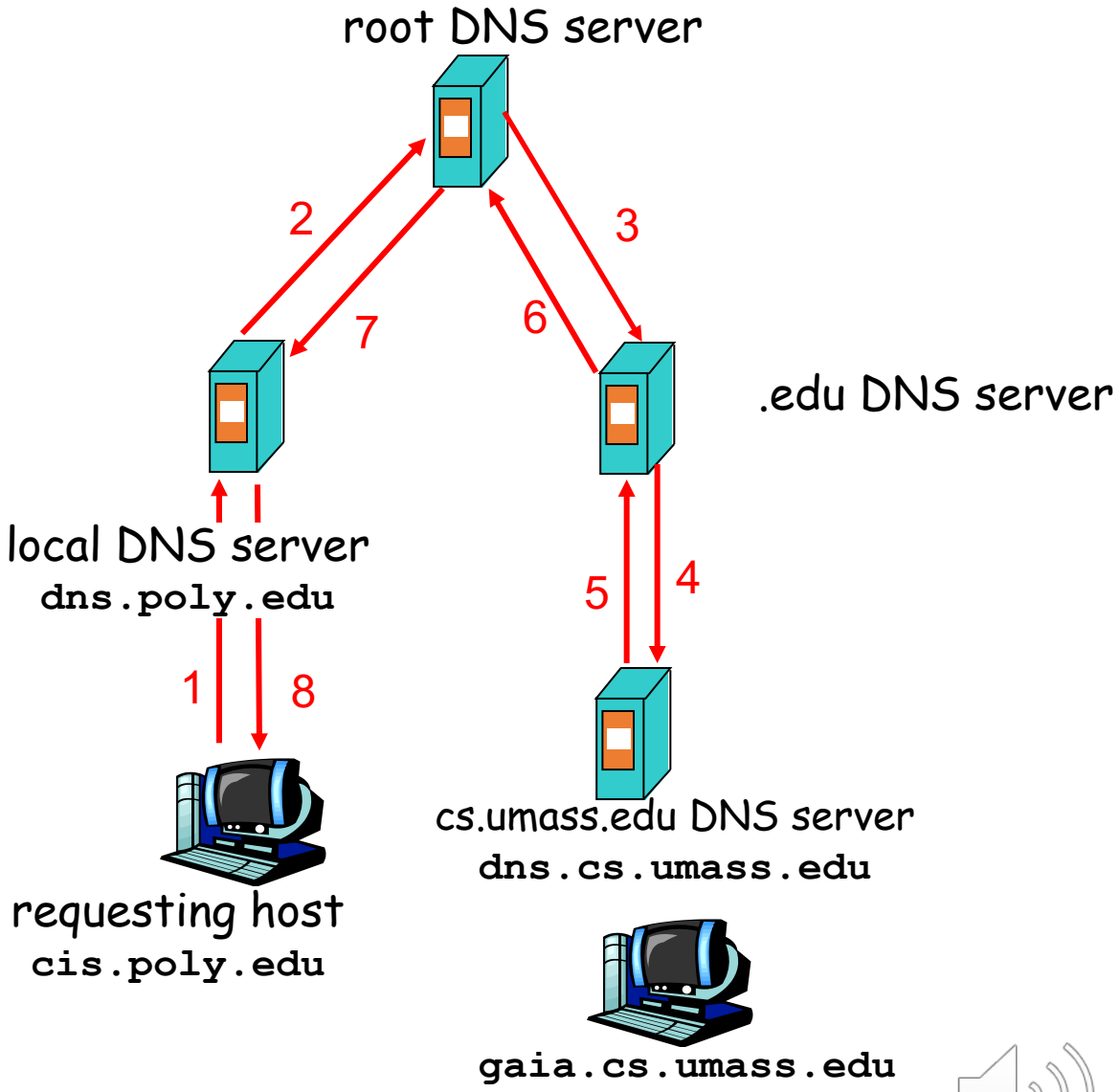
- contacted server replies with name of server to contact
- "I don't know this name, but ask this server"



# Query type

recursive query:

- puts burden of name resolution on contacted name server



# DNS: caching and updating records

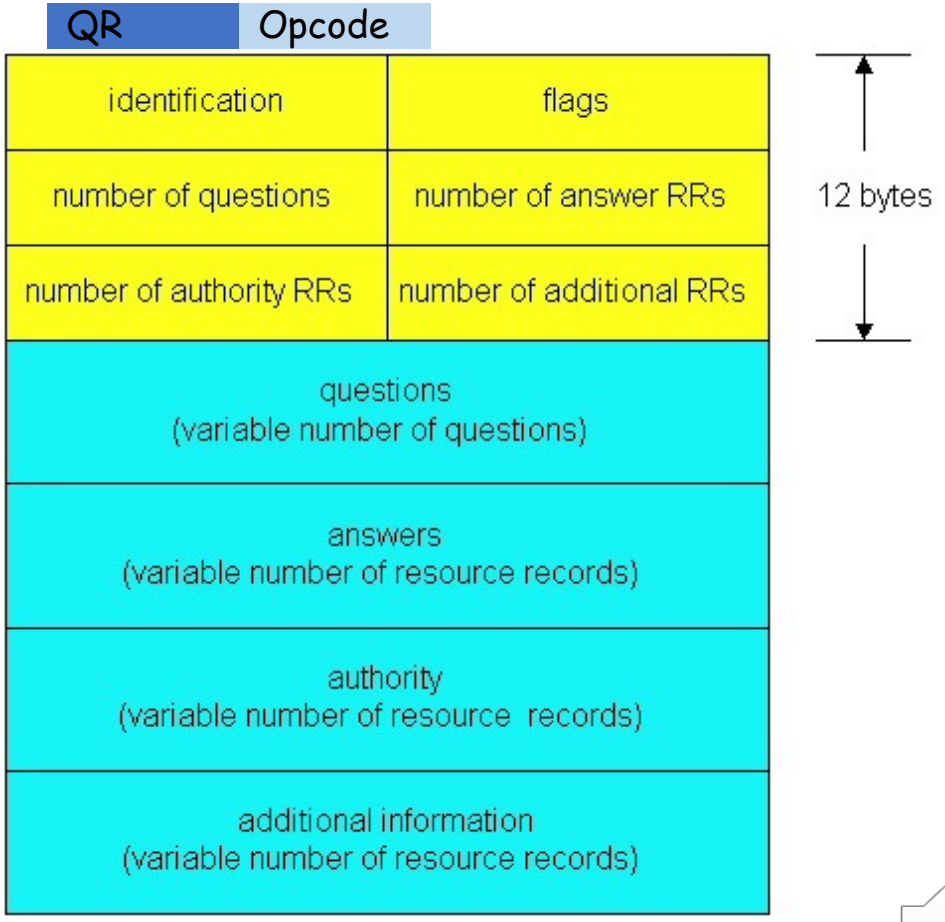
- once (any) name server learns mapping, it *caches* mapping
  - cache entries timeout (disappear) after some time
  - TLD (Top Level Domain) servers typically cached in local name servers
    - Thus root name servers not often visited

# DNS protocol, messages

DNS protocol : *query* and *reply* messages, both with same *message format*

msg header

- QR = 0 for Q, 1 for response
- Opcode= 0 standard
- identification: 16 bit # for query, reply to query uses same #
- flags:
  - Authoritative answer
  - recursion desired
  - recursion available
  - reply is authoritative



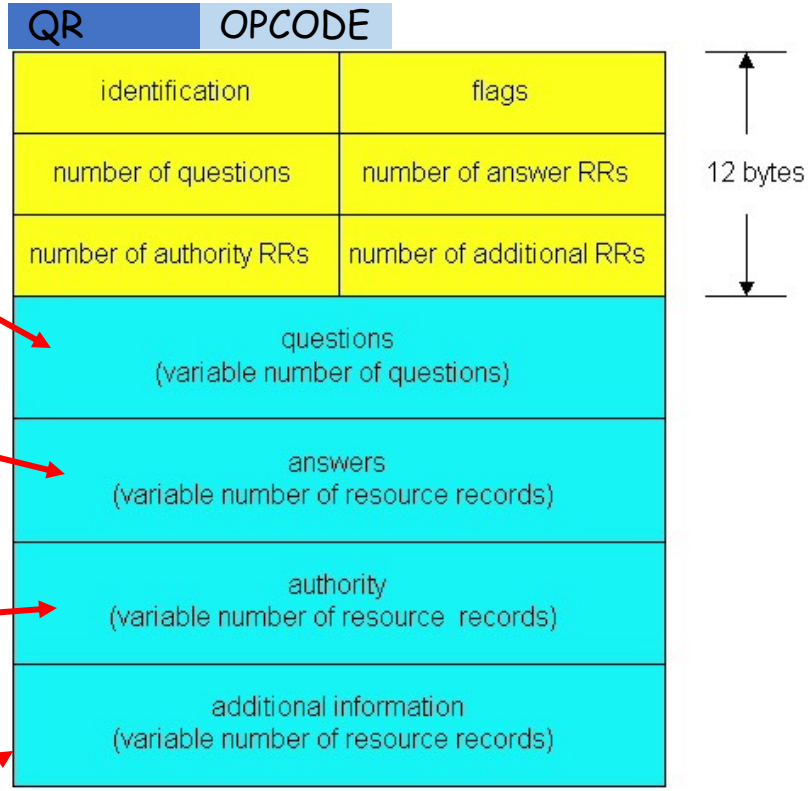
# DNS protocol, messages

Name, type fields for a query

RRs in response to query

records for authoritative servers Information about nameserver

additional "helpful" info that may be used



# 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=NS
  - **name** is domain (e.g. foo.com)
  - **value** is hostname of authoritative name server for this domain
- Type=AAAA
  - **name** is hostname
  - **value** is IPv6 address

# DNS Record example

RRs in response to query

| NAME | Design.cs.rutgers.edu |
|---|---|
| TYPE | A |
| CLASS | IN |
| TTL | 1 day(86400) |
| ADDRESS | 192.26.92.30 |

records for authoritative servers

Information about nameserver

| NAME | Cs.rutgers.edu |
|---|---|
| TYPE | NS |
| CLASS | IN |
| TTL | 1 day(86400) |
| NSD NAME | Ns-lcsr.rutgers.edu |

# Bootstrapping DNS

- How does a host contact the name server if all it has is the name and no IP address?
- IP address of at least 1 nameserver must be given in advance
  - or with another protocol (DHCP, bootp)
  - File **`/etc/resolv.conf`** in unix
  - `Start -> settings-> control panel-> network ->TCP/IP -> properties` in windows

# Themes

- Request/response nature of these protocols
- How Messages are structured
  - HTTP, SMTP, FTP - simple ASCII protocols
- Caching
- Name Lookup
  - Hierarchy structure

# HTTP

Hypertext Transfer Protocol

# Web and HTTP

First some jargon

- Web page consists of objects
- Object can be HTML file, JPEG image, Java applet, 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/undergraduate/pic.gif`

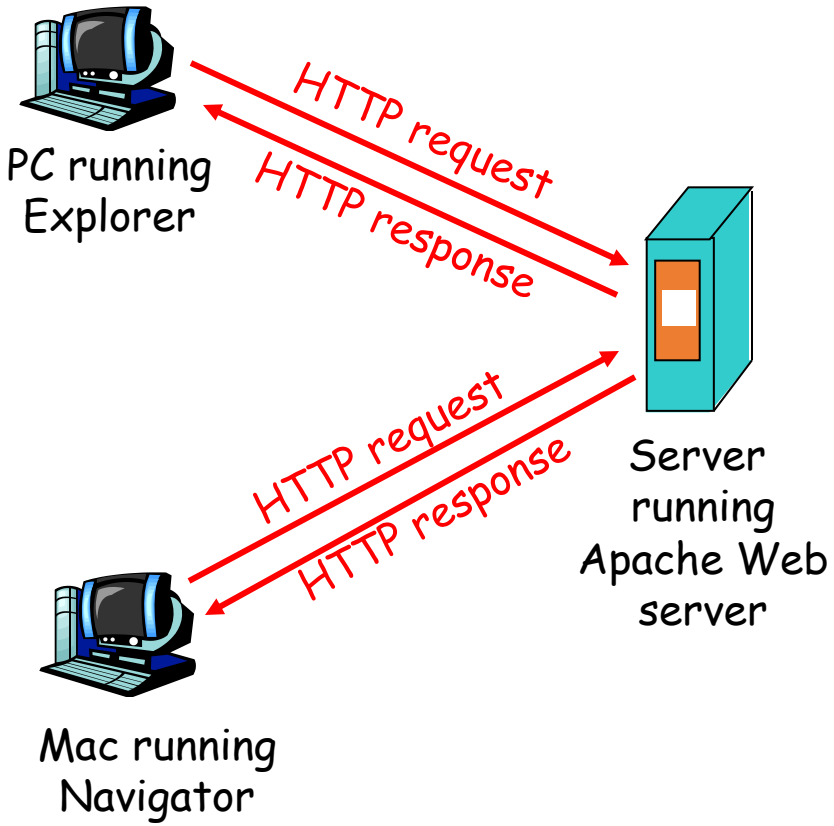
host name | path name

# 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



# HTTP messages: request message

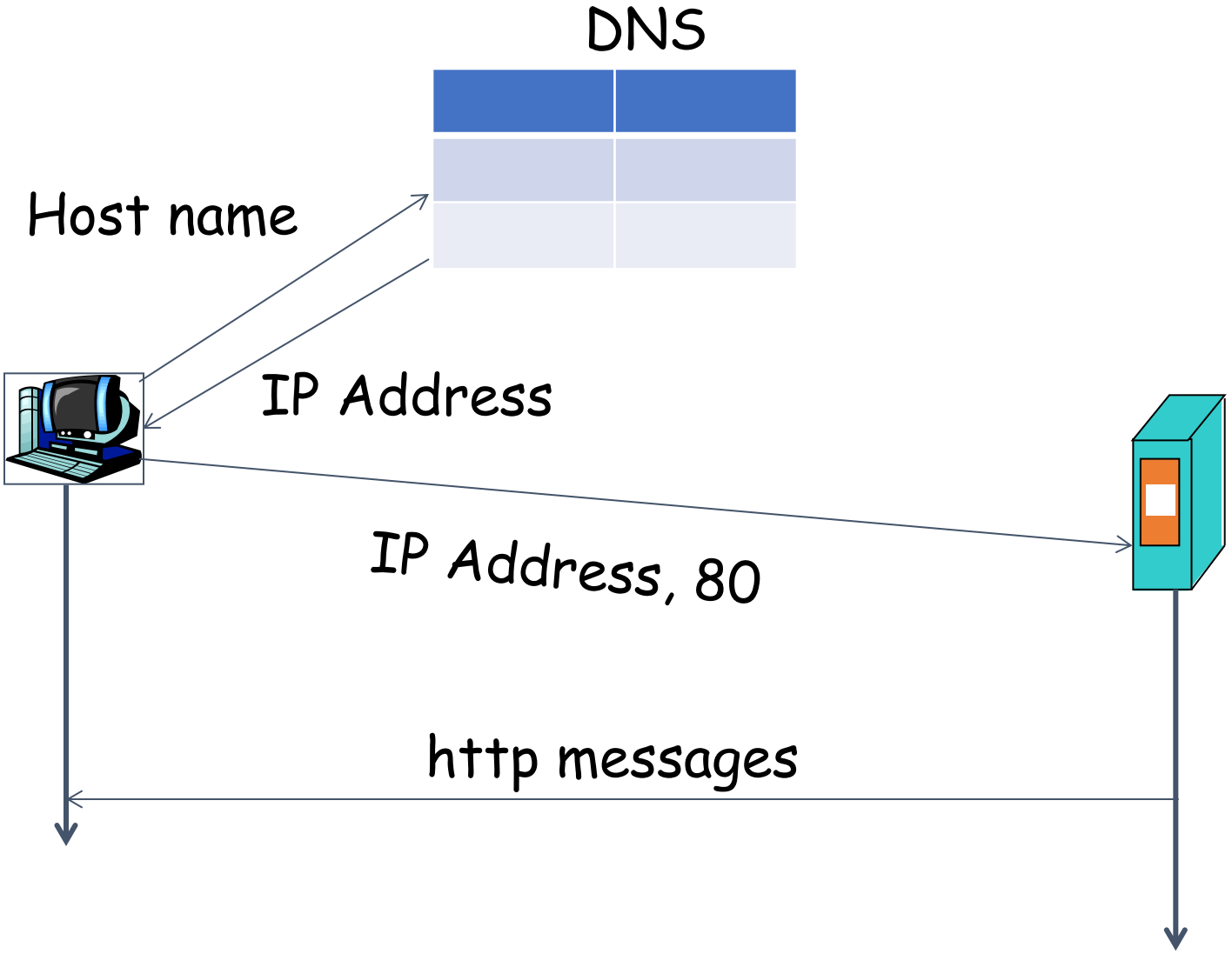
- HTTP request message:
  - ASCII (human-readable format)

request line (GET, POST, HEAD commands)

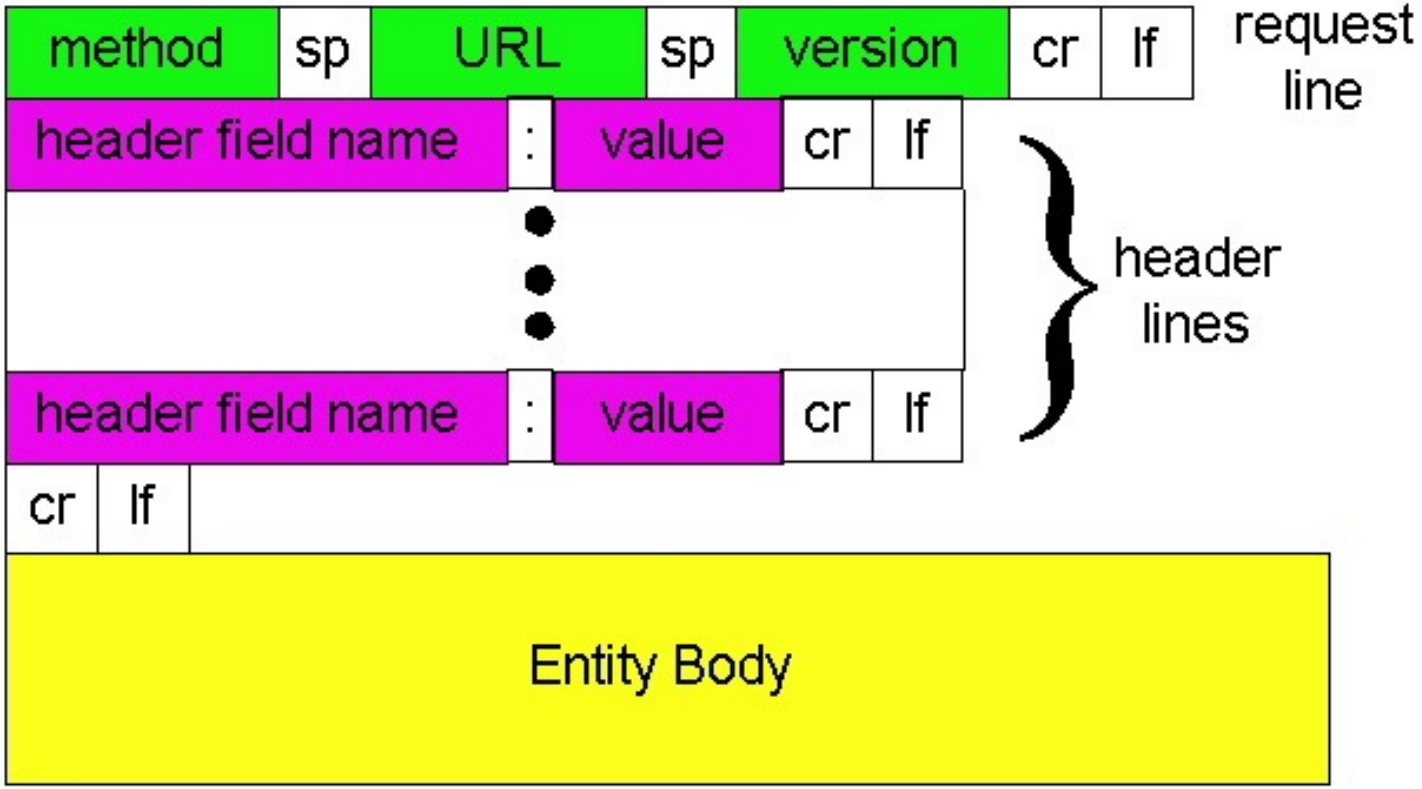
header lines

```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
User-agent: Mozilla/4.0
Connection: close
Accept-language:fr
```

# Client server connection



# HTTP request message: general format



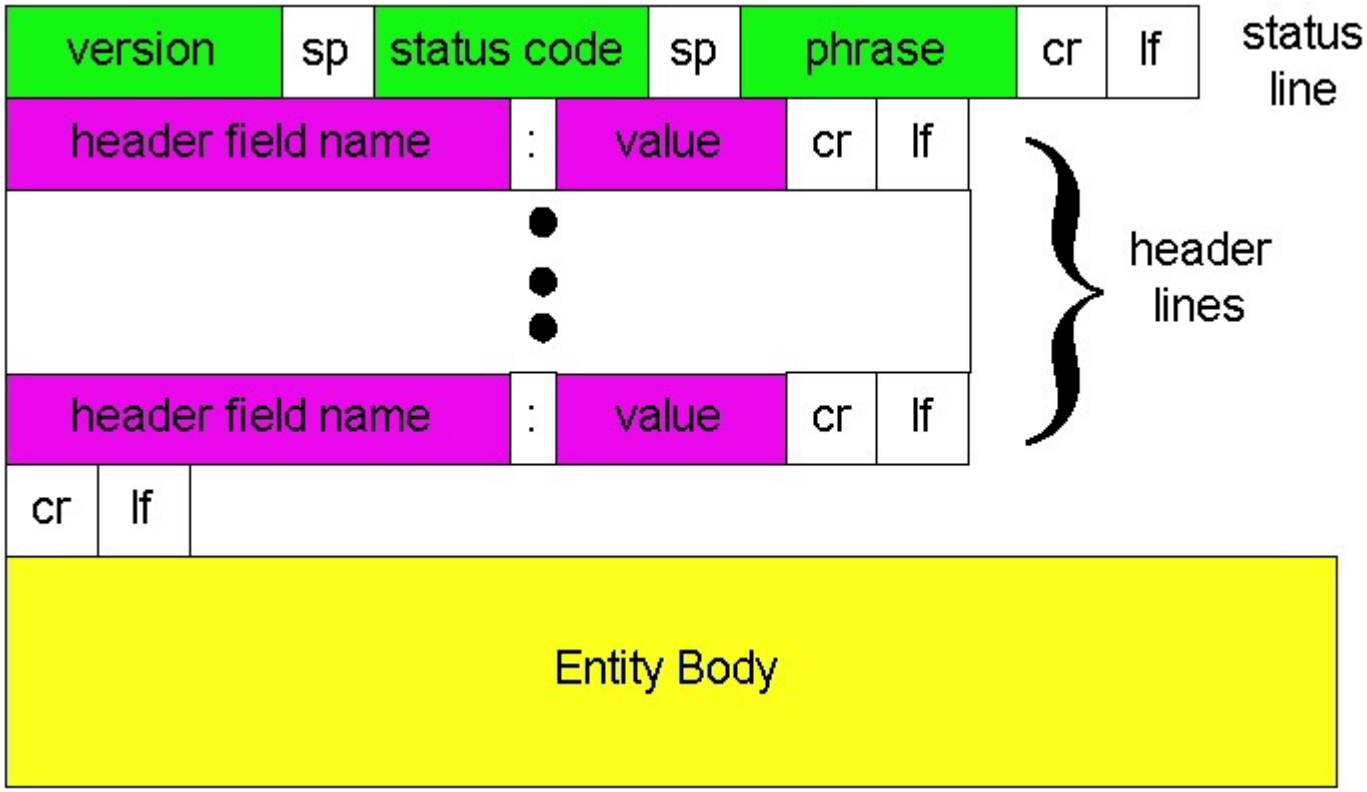

http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14

# Method types

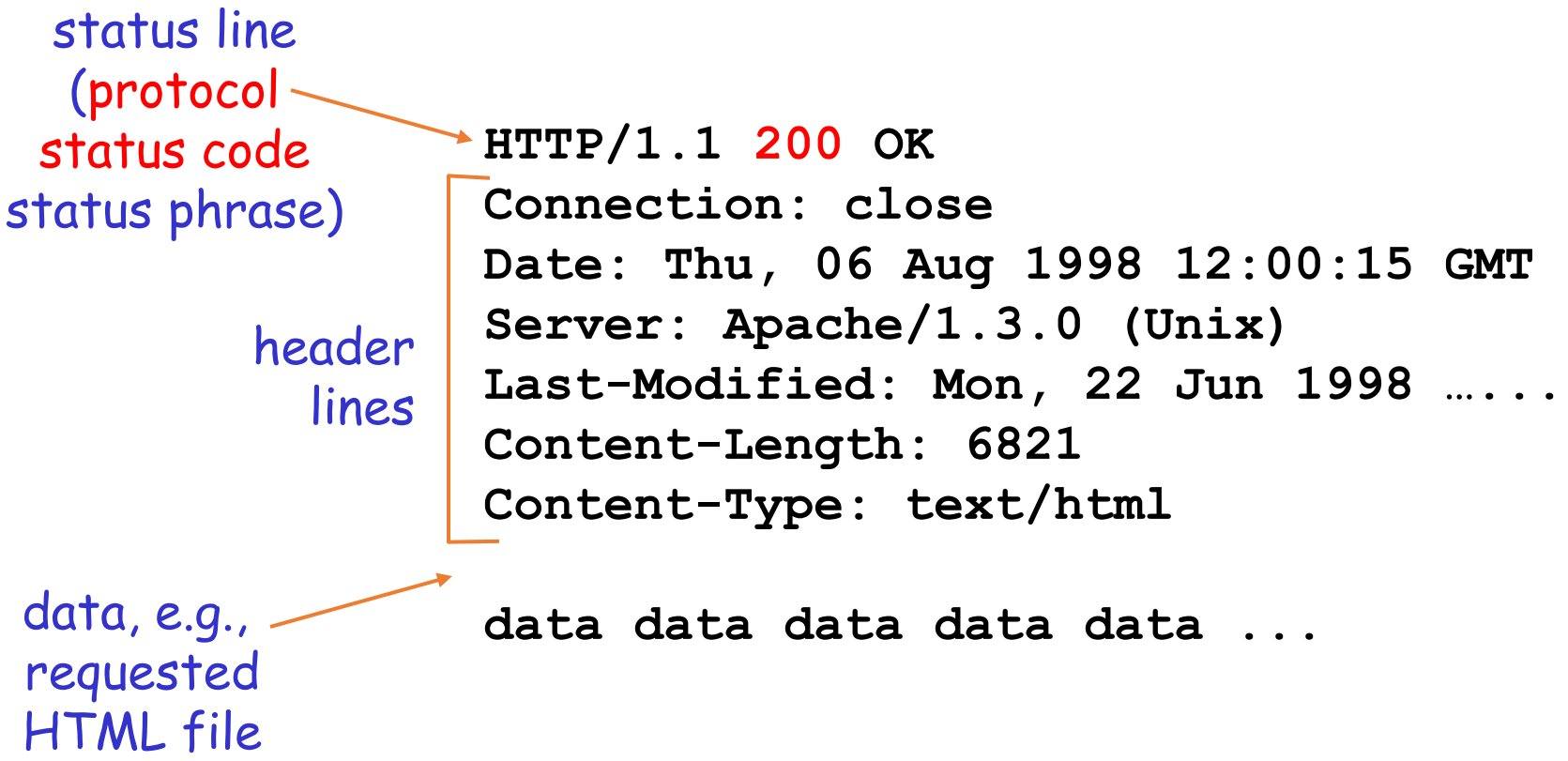
- GET
  - Get the file specified in the path URL field in entity body
- POST
  - accept the entity enclosed in the entity body as a new subordinate of the resource identified by the URL field
- PUT
  - uploads file in entity body to path specified in URL field
- DELETE
  - deletes file specified in the URL field

# 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**

# Additional about HTTP

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

# HTTP connections

## Nonpersistent HTTP

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

## Persistent HTTP

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

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

# Nonpersistent HTTP

Suppose user enters URL `www.someSchool.edu/someDepartment/home.index` (contains text, references to 10 jpeg images)

1a. HTTP client initiates TCP connection to HTTP server

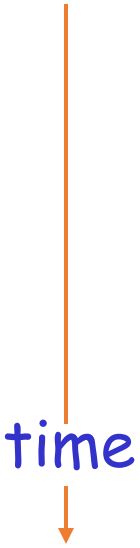
1b. HTTP server at host "accepts" connection, notifying client

2. HTTP client sends HTTP *request message*

3. HTTP server receives request message, replies with *response message* containing requested object

time

# Nonpersistent HTTP (cont.)




4. HTTP server closes TCP connection.

5. HTTP client receives response message containing html file, displays html. Parsing html file, finds 10 referenced jpeg objects

6. Steps 1-5 repeated for each of 10 jpeg objects

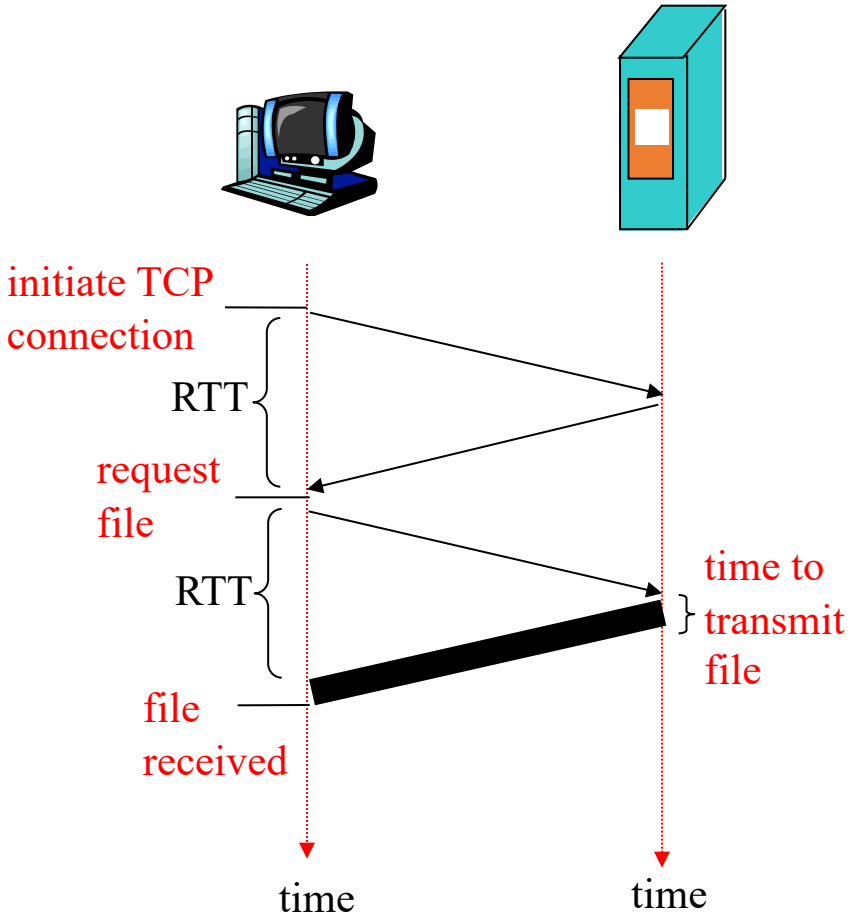
# HTTP: Response time

Definition of RTT (Round-Trip Time): time to send a small packet to travel from client to server and back.

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. Nonpersistent

Nonpersistent HTTP issues:

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

Persistent HTTP

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

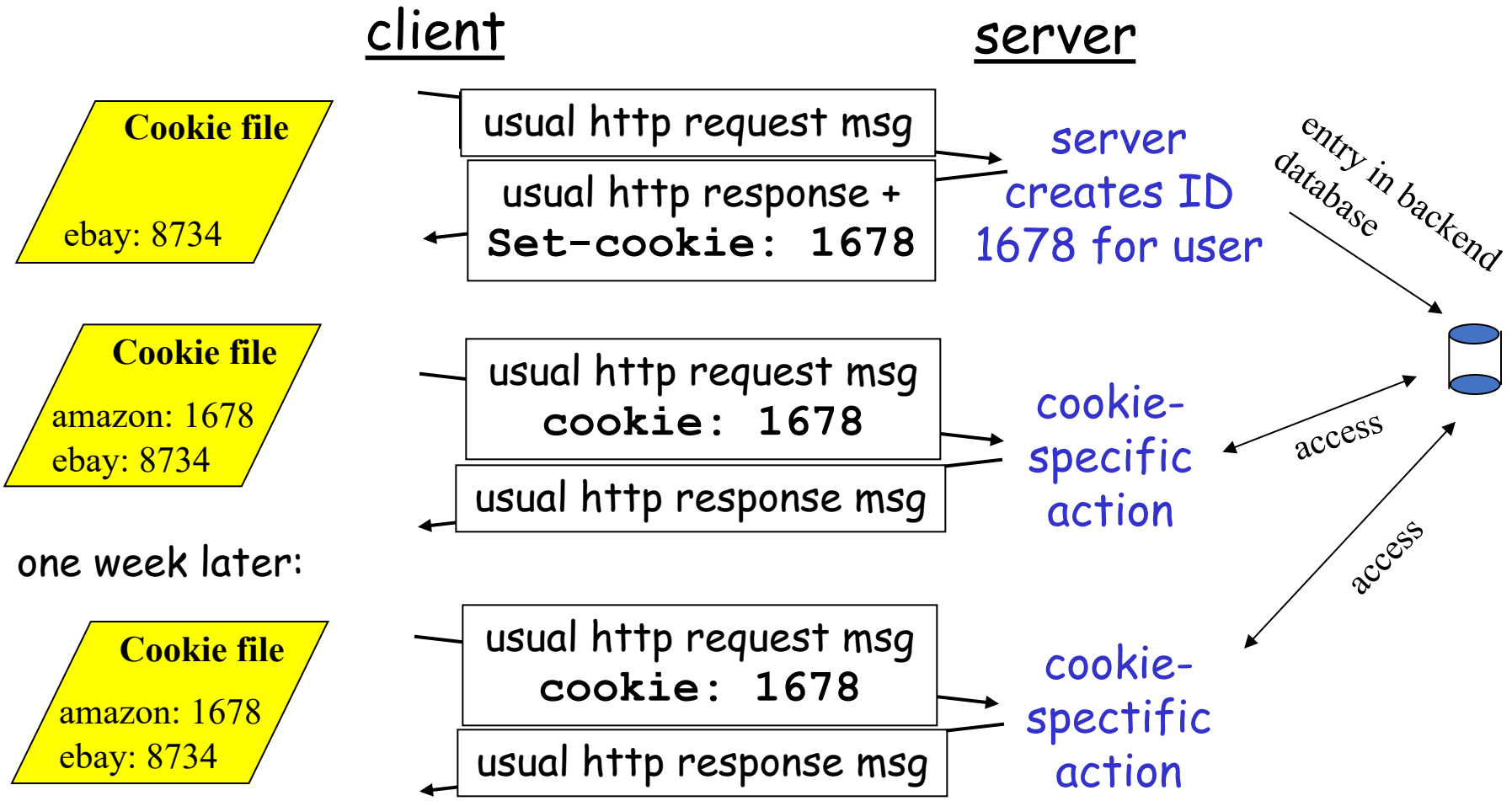
# HTTP: user-server state

HTTP is “stateless”

- server maintains no information about past client requests

- What state can bring:
  - authorization
  - shopping carts
  - recommendations
  - user session state

# Cookies: keeping "state"



# Cookies (continued)

Four components:

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

# Cookies (continued)

aside

**Cookies and privacy:**

- cookies permit sites to learn a lot about you