The Web (HTTP)

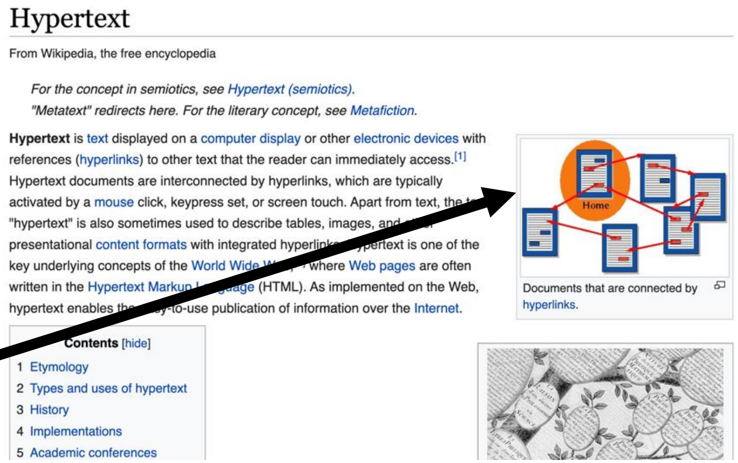
Lecture 6

http://www.cs.rutgers.edu/~sn624/352-F24

Srinivas Narayana

RUTGERS
UNIVERSITY | NEW BRUNSWICK

Web and HTTP: Terms



Hypertext

From Wikipedia, the free encyclopedia

For the concept in semiotics, see Hypertext (semiotics).

"Metatext" redirects here. For the literary concept, see Metafiction.

Hypertext is text displayed on a computer display or other electronic devices with references (hyperlinks) to other text that the reader can immediately access.[1] Hypertext documents are interconnected by hyperlinks, which are typically activated by a mouse click, keypress set, or screen touch. Apart from text, the term "hypertext" is also sometimes used to describe tables, images, and other presentational content formats with integrated hyperlinks. Hypertext is one of the key underlying concepts of the World Wide Web,[2] where Web pages are often written in the Hypertext Markup Language (HTML). As implemented on the Web, hypertext enables the easy-to-use publication of information over the Internet.

Documents that are connected by hyperlinks.

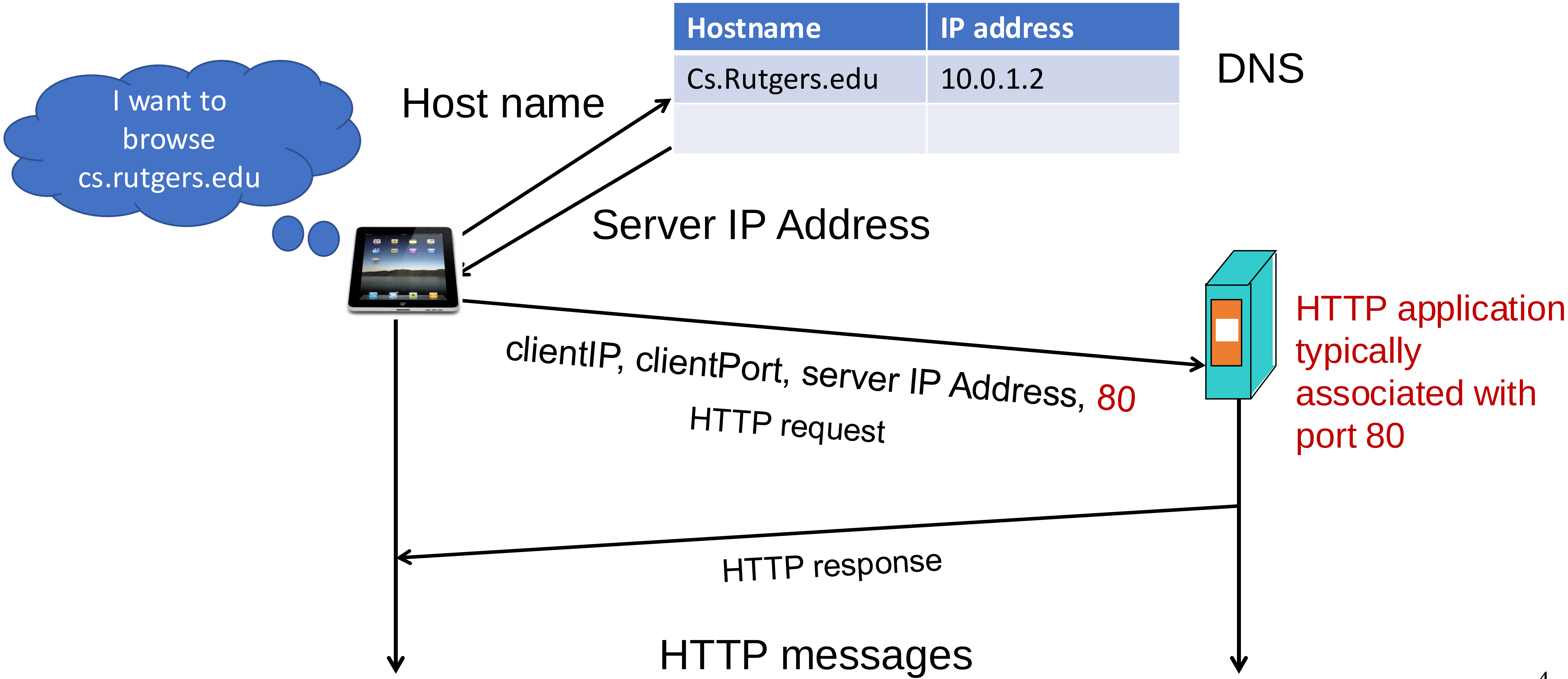
Contents [hide]

- HTTP stands for “HyperText Transfer Protocol”
- A web page consists of many objects
- Object can be HTML file, JPEG image, video stream chunk, audio file,…
- Web page consists of base HTML-file which embeds several objects
- Each object is addressable by a uniform resource locator (URL)
  - sometimes also referred to as uniform resource identifier (URI)
- Example URL:

$$\underbrace{\texttt{www.cs.rutgers.edu}}_{\text{Domain/host name}}\underbrace{\texttt{/\textasciitilde sn624/index.html}}_{\text{path}}$$

HTTP Protocol

Client server protocol



I want to browse cs.rutgers.edu

Host name

Hostname	IP address
Cs.Rutgers.edu	10.0.1.2

DNS

Server IP Address

clientIP, clientPort, server IP Address, 80

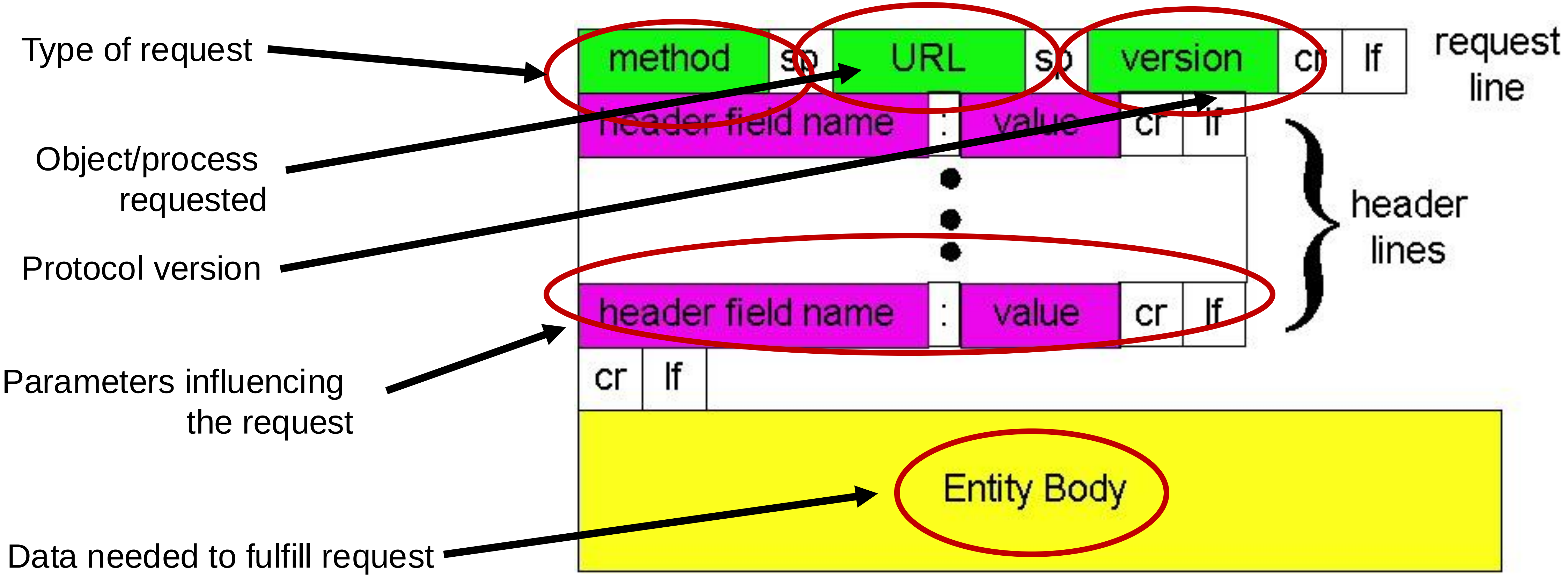
HTTP request

HTTP application typically associated with port 80

HTTP response

HTTP messages

HTTP Request: Message Format



Type of request

Object/process requested

Protocol version

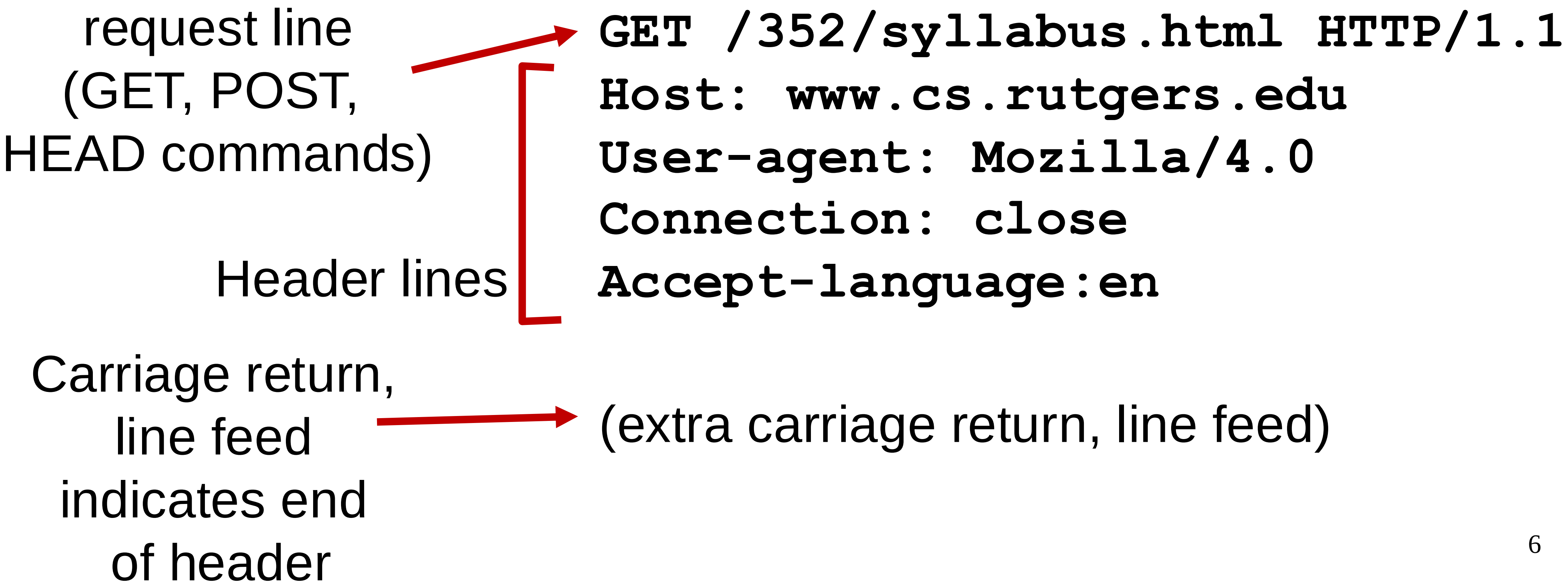
Parameters influencing the request

Data needed to fulfill request

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

HTTP messages: request message

- ASCII (human-readable format)



request line (GET, POST, HEAD commands)

```
GET /352/syllabus.html HTTP/1.1
Host: www.cs.rutgers.edu
User-agent: Mozilla/4.0
Connection: close
Accept-language:en
```

Header lines

Carriage return, line feed indicates end of header

(extra carriage return, line feed)

The URL

- Universal Resource Locator: a way to name objects on server
- But can also name an application process on the server!
- Examples:
  - Data storage from data entered in web forms
  - Login pages
  - Web carts
- Providing almost any service requires data handling by running code at the server
  - Not just rendering “static” resources

HTTP method types

- GET
  - Get the resource specified in the requested URL (could be a process)
- POST
  - Send entities (specified in the entity body) to a data-handling process at the requested URL
- HEAD
  - Asks server to leave requested object out of response, but send the rest of the response
  - Useful for debugging

- PUT
  - Update a resource at the requested URL with the new entity specified in the entity body
- DELETE
  - Deletes file specified in the URL

- and other methods

https://httpwg.org/specs/rfc9110.html#method.definitions

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
- URL must contain a restricted set of characters
- Example:
  - http://site.com/form?first=jane&last=austen

Difference between POST and PUT

- POST: the URL of the request identifies the resource that processes the entity body

- PUT: the URL of the request identifies the resource that is contained in the entity body

https://tools.ietf.org/html/rfc2616

Difference between HEAD and GET

- GET: return the requested resource in the entity body of the response along with response headers (we'll see these shortly)

- HEAD: return all the response headers in the GET response, but without the resource in the entity body

https://tools.ietf.org/html/rfc2616

Observing HTTP GET and POST

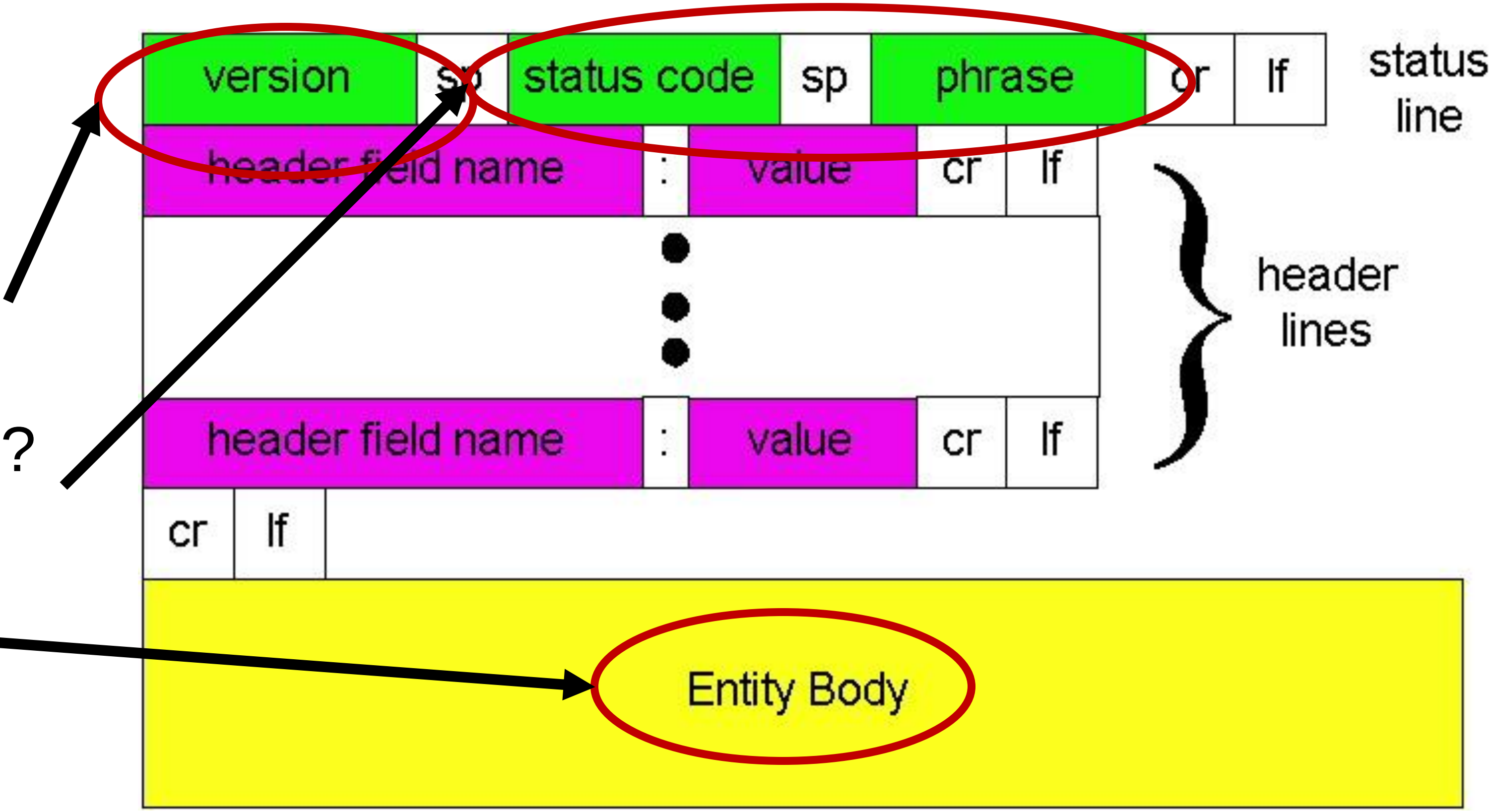
HTTP Response: General format

Unlike HTTP request,
No method name

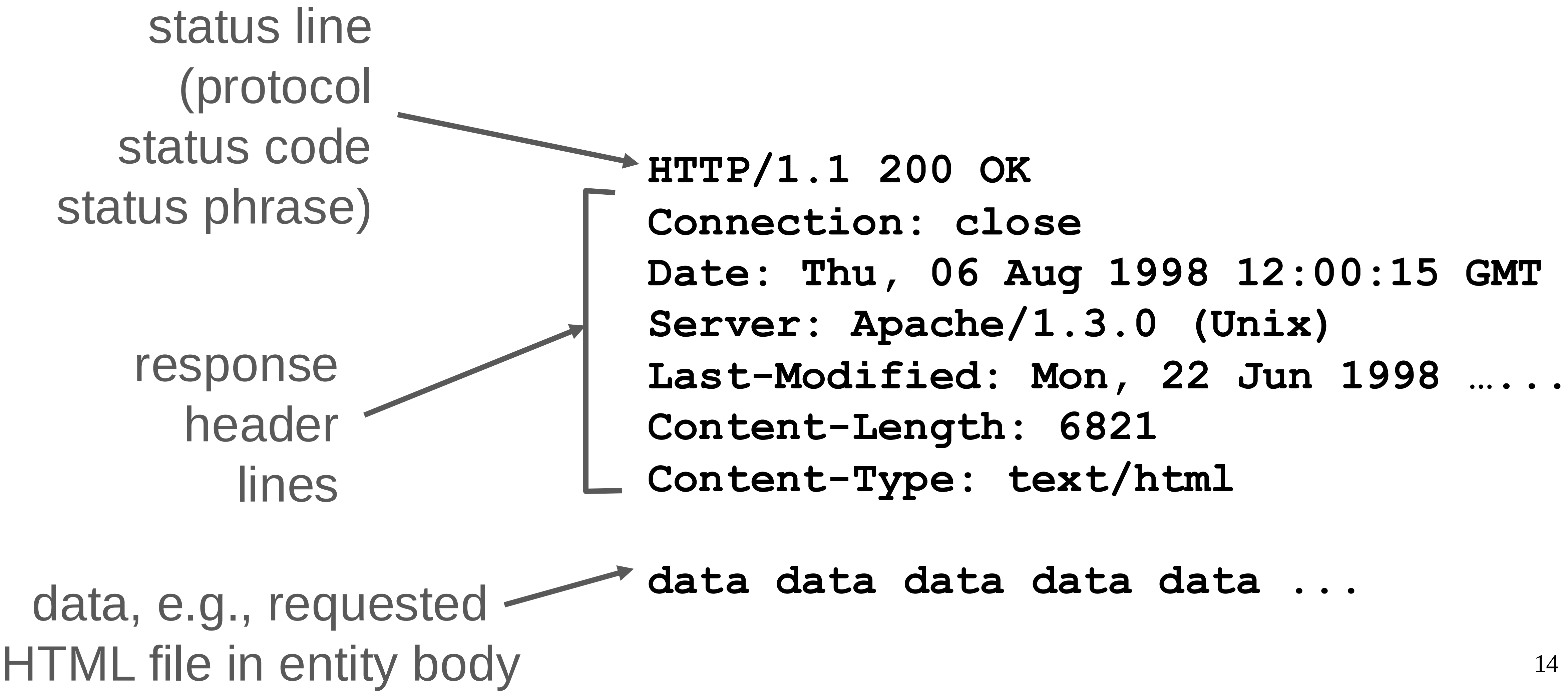
HTTP protocol version
used by server

Was request successful?
(or error condition)

Returned object data



HTTP message: response message



status line (protocol status code status phrase)

response header lines

data, e.g., requested HTML file in entity body

```
HTTP/1.1 200 OK
Connection: close
Date: Thu, 06 Aug 1998 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon, 22 Jun 1998 …...
Content-Length: 6821
Content-Type: text/html

data data data data data ...
```

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:)

`403 Forbidden`

- Insufficient permissions to access the resource

`404 Not Found`

- requested document not found on this server

`505 HTTP Version Not Supported`

Observing HTTP behaviors

- `wget google.com (or) curl google.com`

- `telnet example.com 80`
  - `GET / HTTP/1.1`
  - `Host: example.com`

(followed by two enter’s)

- Exercise: try
  - `telnet google.com 80`
  - `telnet web.mit.edu 80`

HTTP Persistence

HTTP connections

Non-persistent HTTP

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

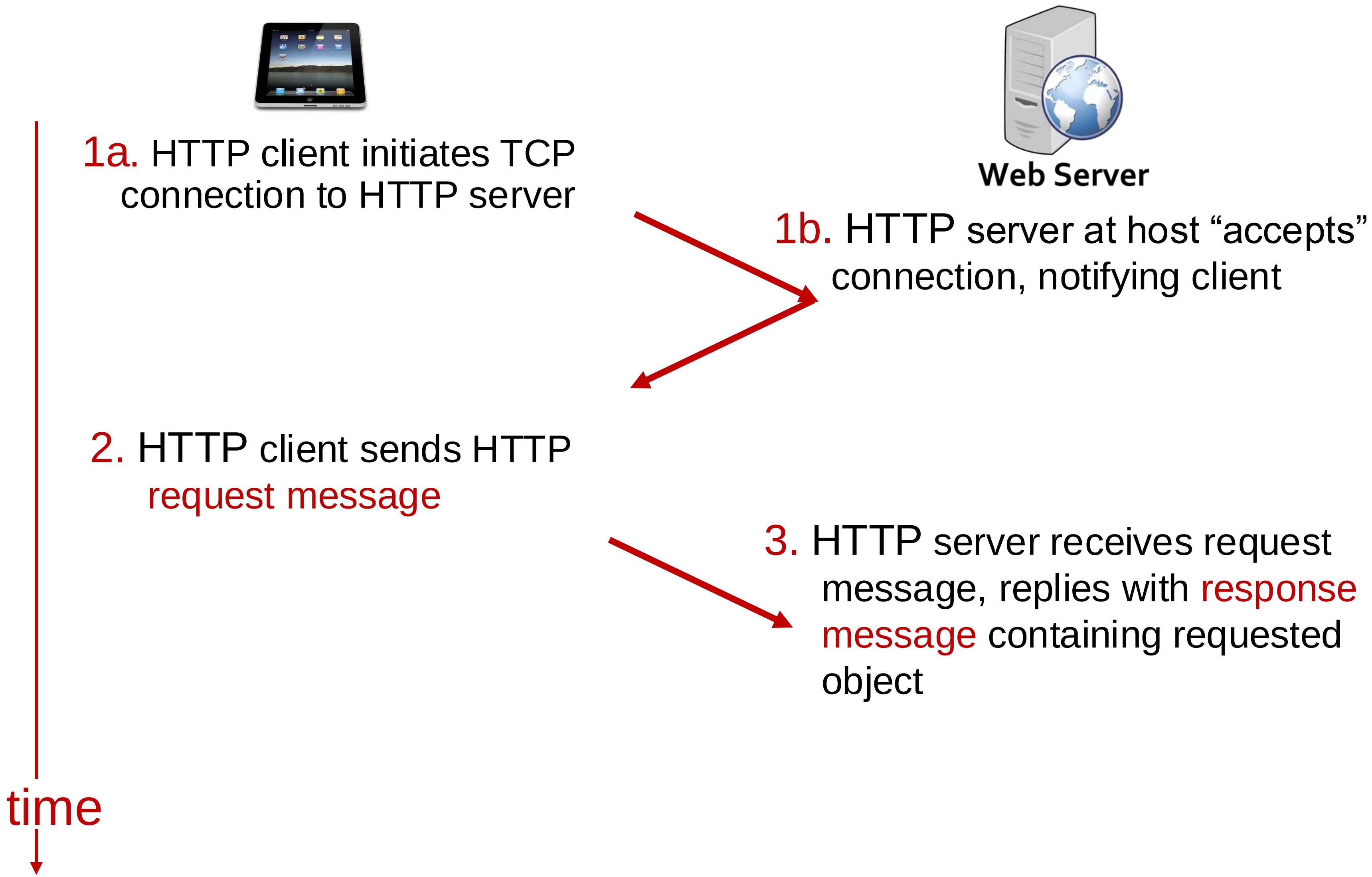
Persistent HTTP

- 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 some resources for the connection to be set up at the endpoints before data communication.

Non-persistent HTTP (HTTP/1.0)

Suppose a user visits a page with text and 10 embedded images.



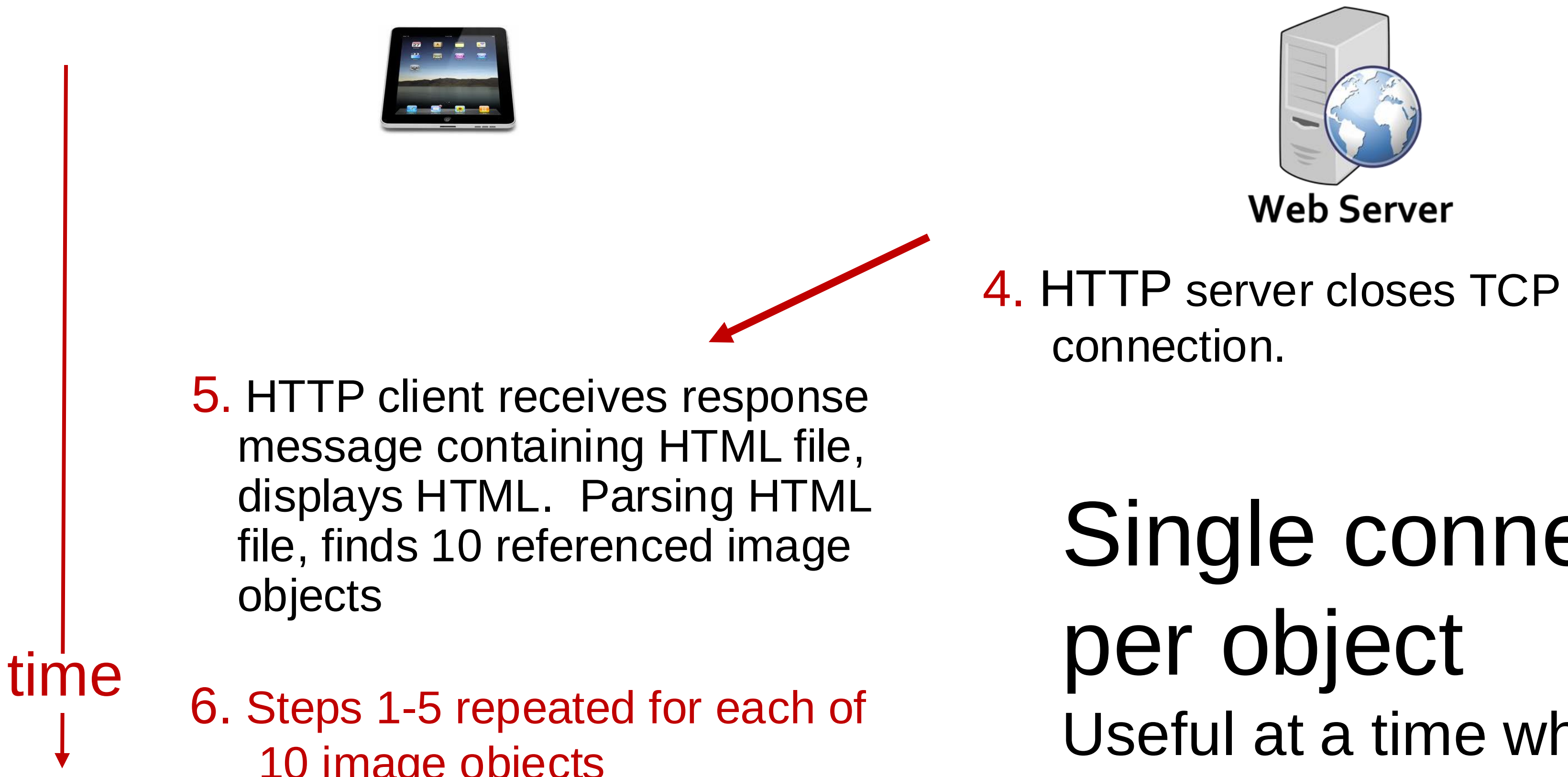
1a. HTTP client initiates TCP connection to HTTP server

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

Non-persistent HTTP (HTTP/1.0)



4. HTTP server closes TCP connection.

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

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

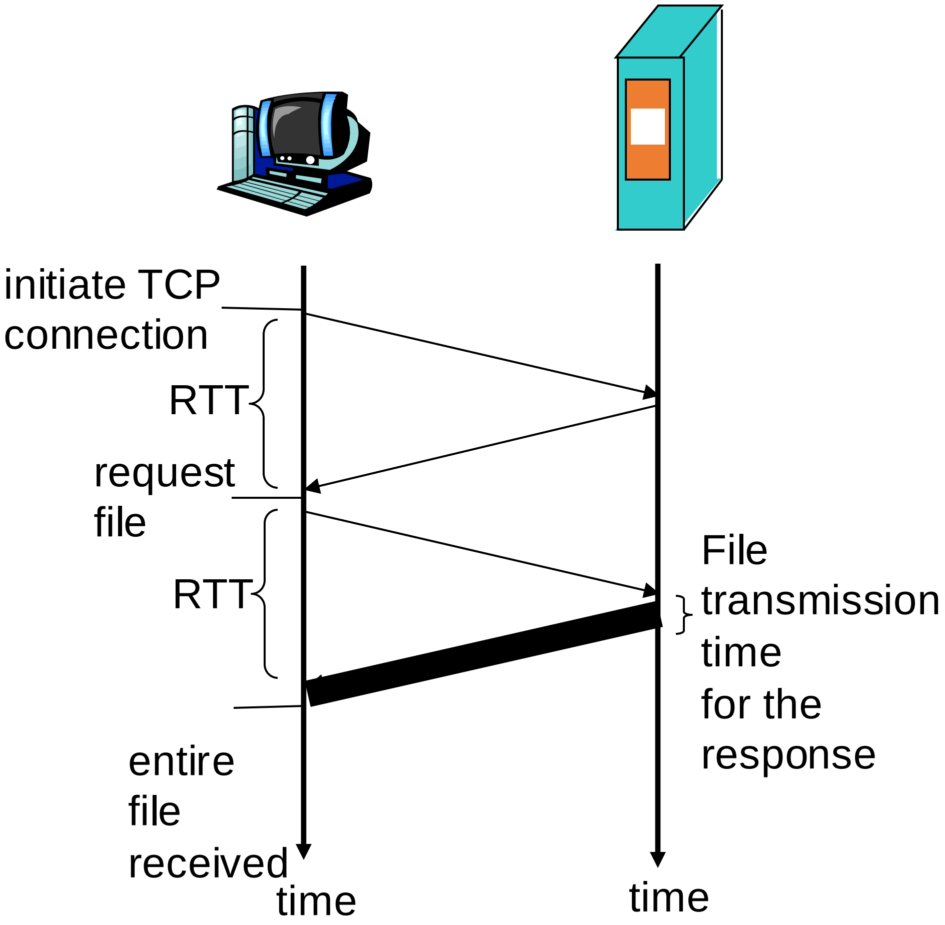
Single connection per object

Useful at a time when web pages contained 1 object: the base HTML file.

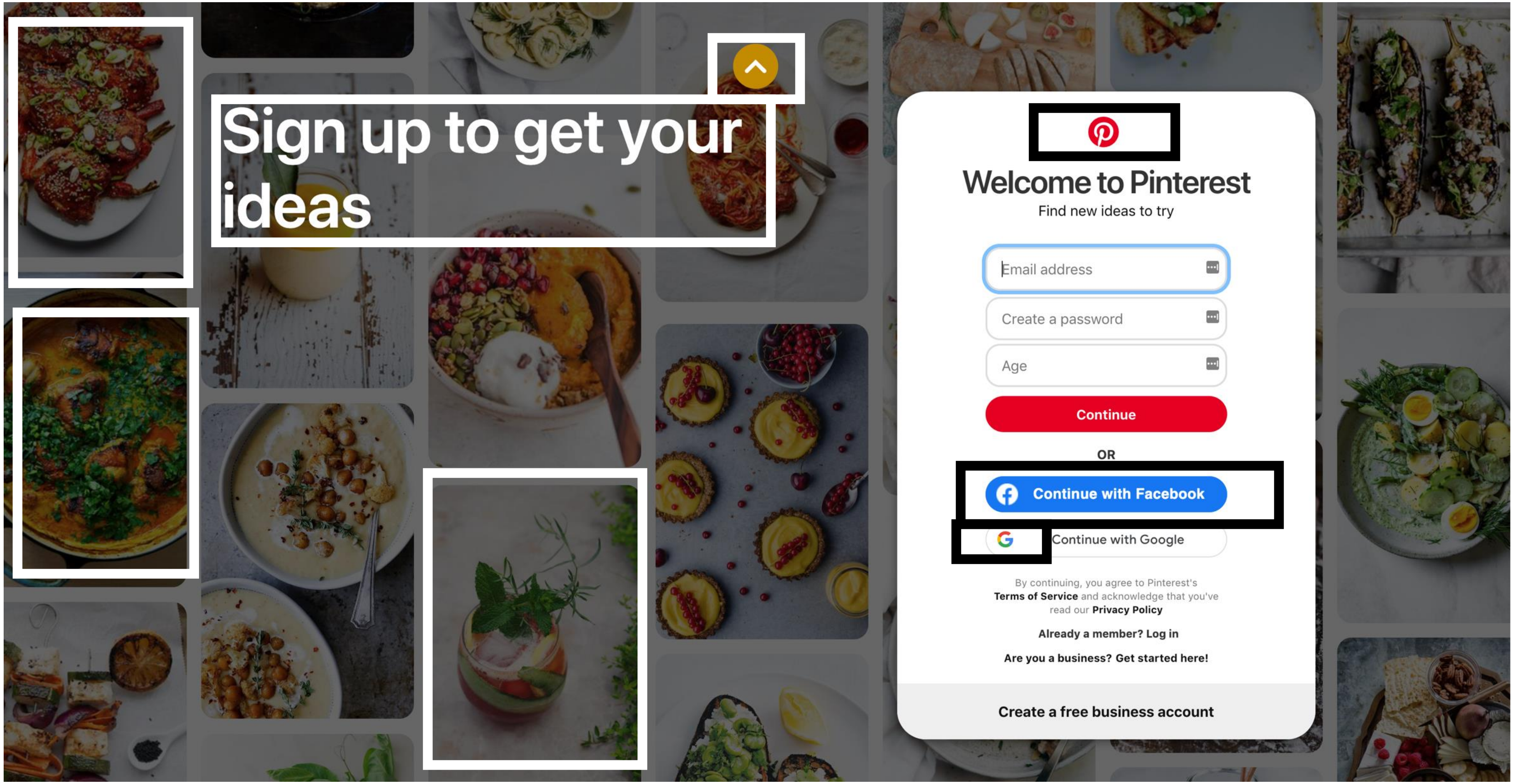
How long does it take to transfer an object with non-persistent HTTP?
i.e.: before your browser can load the (entire) object?

Non-persistent HTTP user response time

- Total delay = propagation + queueing + transmission
- Response time for the user
    - = sum of forward and backward total delays
- Round-Trip Time (RTT): total forward + backward delay for a “small” packet
    - Zero transmission delay
- Assumptions:
    - TCP initiation packet, response, HTTP requests are all “small” packets
    - No processing delays at the server
    - RTT is stable over time
- (2RTT + file transmission time) * #objects



Per-object overheads quickly add up



Sign up to get your ideas

Welcome to Pinterest

Find new ideas to try

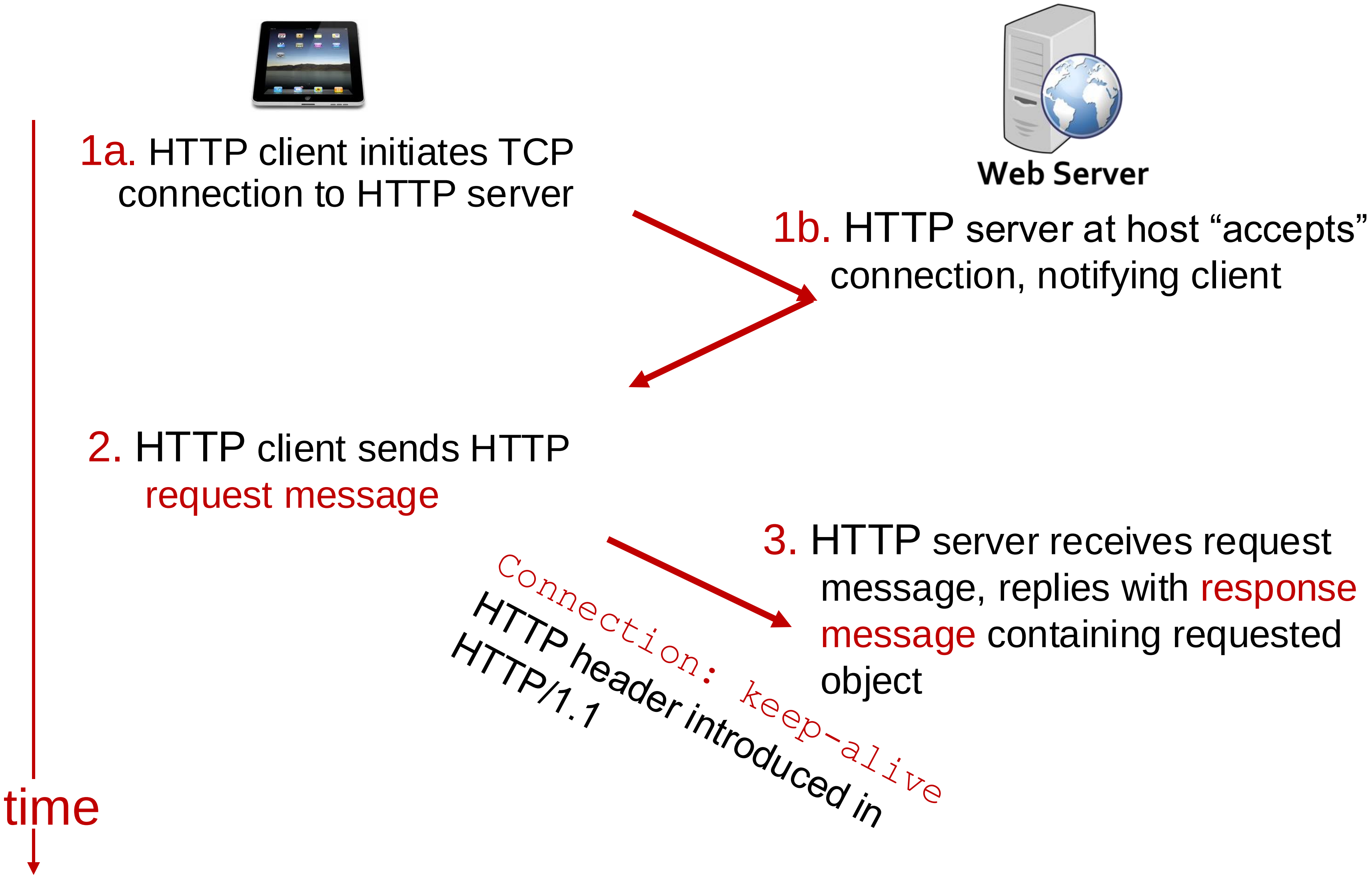
By continuing, you agree to Pinterest's **Terms of Service** and acknowledge that you've read our **Privacy Policy**

Modern web pages have 100s of objects in them.

Objects (e.g. images) may not be small.

Persistent HTTP (HTTP/1.1)

Suppose user visits a page with text and 10 images.



1a. HTTP client initiates TCP connection to HTTP server

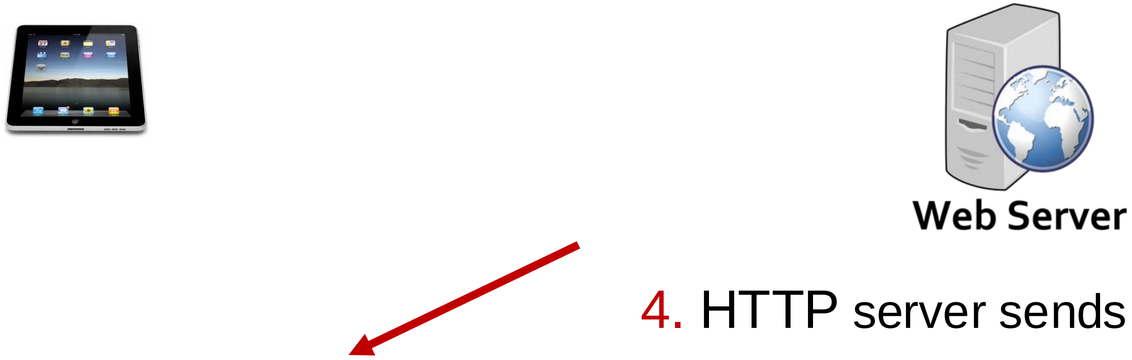
1b. HTTP server at host “accepts” connection, notifying client

2. HTTP client sends HTTP request message

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

`Connection: keep-alive` HTTP header introduced in HTTP/1.1

Persistent HTTP (HTTP/1.1)



4. HTTP server sends a response.

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

Server keeps the TCP connection alive.

time

The 10 objects can be requested over the same TCP connection.

i.e., save an RTT per object (otherwise spent opening a new TCP connection in HTTP/1.0)

Persistent HTTP user response time

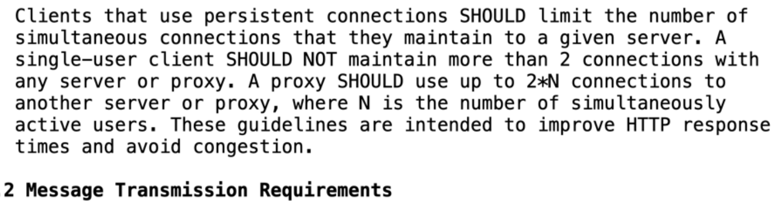
- Assume requests made one at a time (separate RTT per req)
- RTT + (RTT + file transmission time) * #objects
- Pipelining: send more than one HTTP request at a time
  - Extreme case: all requests in one (small) packet
  - RTT + (file transmission time) * #objects
  - In practice, dependencies between objects
- Compare with non-persistent:
  - (2RTT + file transmission time) * #objects
- Persistence (& pipelining) can save significant time, especially on high-RTT connections
- Other advantages of persistence: CPU savings, reduced network congestion, less memory (fewer connections)

Persistence vs. # of connections

- Persistence is distinct from the number of concurrent connections made by a client

- Your browser has the choice to open multiple connections to a server
  - HTTP spec suggests to limit this to a small number (2)

- Further, a single connection can have multiple HTTP requests in flight (pipelining) with persistent HTTP



Clients that use persistent connections SHOULD limit the number of simultaneous connections that they maintain to a given server. A single-user client SHOULD NOT maintain more than 2 connections with any server or proxy. A proxy SHOULD use up to 2*N connections to another server or proxy, where N is the number of simultaneously active users. These guidelines are intended to improve HTTP response times and avoid congestion.

8.2 Message Transmission Requirements

Remembering Users On the Web

HTTP: Remembering users

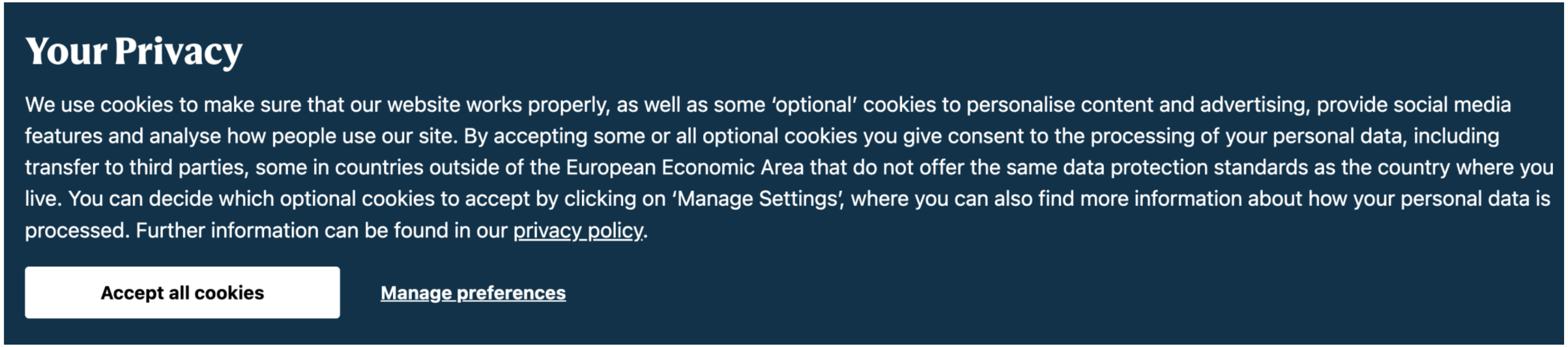
So far, HTTP mechanisms considered stateless

- Each request processed independently at the server
- The server maintains no memory about past client requests

However, state, i.e., memory, about the user at the server, is very useful!

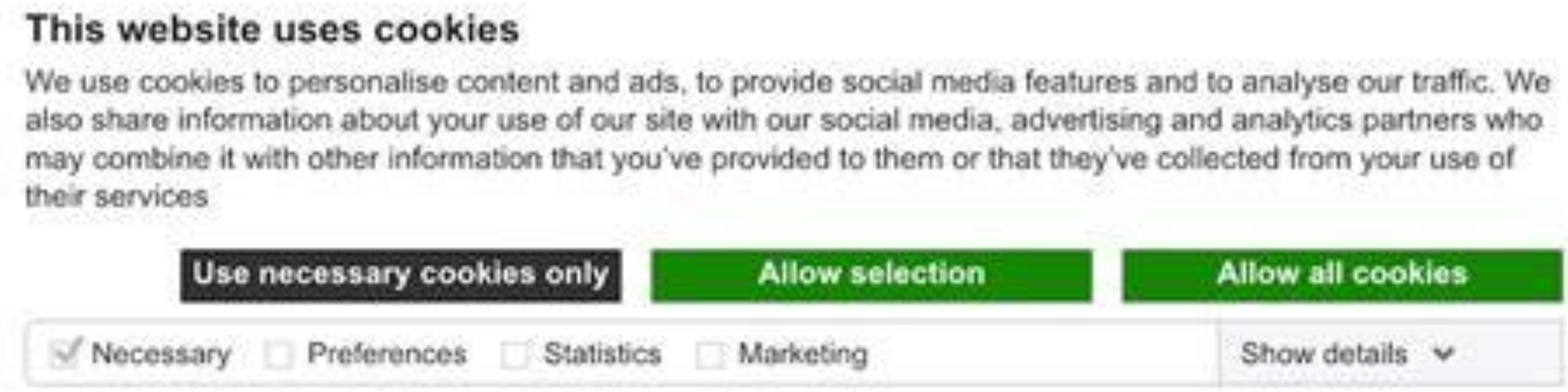
- User authentication (e.g., gmail)
- Shopping carts (e.g., Amazon)
- Video recommendations (e.g., Netflix)
- Any user session state in general

Familiar with these?



Your Privacy

We use cookies to make sure that our website works properly, as well as some ‘optional’ cookies to personalise content and advertising, provide social media features and analyse how people use our site. By accepting some or all optional cookies you give consent to the processing of your personal data, including transfer to third parties, some in countries outside of the European Economic Area that do not offer the same data protection standards as the country where you live. You can decide which optional cookies to accept by clicking on ‘Manage Settings’, where you can also find more information about how your personal data is processed. Further information can be found in our privacy policy.

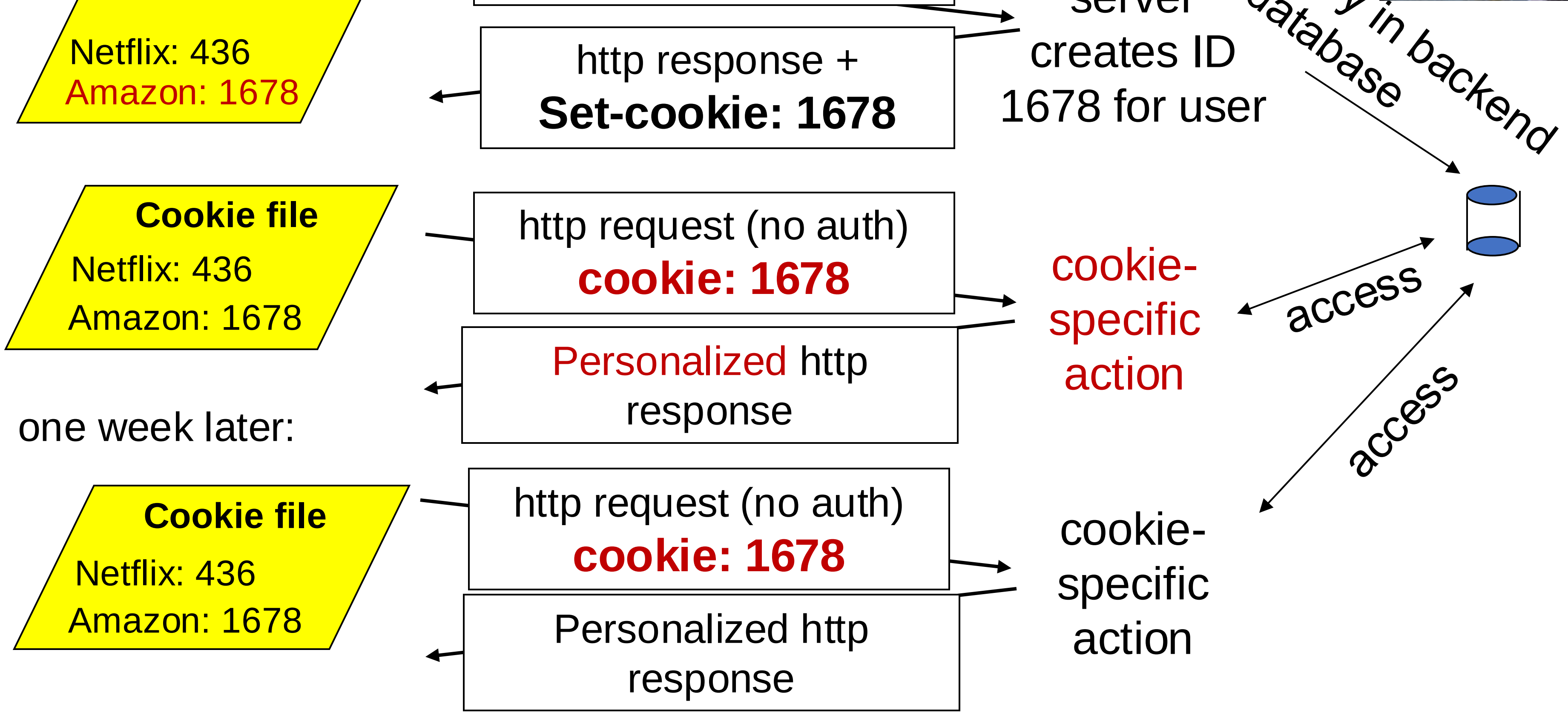


This website uses cookies

We use cookies to personalise content and ads, to provide social media features and to analyse our traffic. We also share information about your use of our site with our social media, advertising and analytics partners who may combine it with other information that you've provided to them or that they've collected from your use of their services

Cookies: Keeping user memory

Cookie is typically opaque to client.



How cookies work

Collaboration between client and server to track user state.

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

Cookies come with an expiration date (yet another HTTP header)

Cookies have many uses

- The good: Awesome user-facing functionality
  - Shopping carts, auth, … very challenging or impossible without it
- The bad: Unnecessary recording of your activities on the site
  - First-party cookies: performance statistics, user engagement, …
- The ugly: Tracking your activities across the Internet
  - Third-party cookies (played by ad and tracking networks) to track your activities across the Internet
  - personally identifiable information (PII)
  - Ad networks target users with ads; may sell this info
  - Scammers can target you too

PSA: Cookies and Privacy

- Disable and delete unnecessary cookies by default

- Suggested privacy-conscious browsers, websites, tools:

- DuckDuckGo (search)
- Brave (browser)
- AdBlock Plus (extension)
- ToR (distract targeting)
- … assuming it doesn’t break the functions of the site

https://gdpr.eu/cookies/