

CS 330: Network Applications & Protocols

Application Layer: HTTP

Galin Zhelezov
Department of Physical Sciences
York College of Pennsylvania

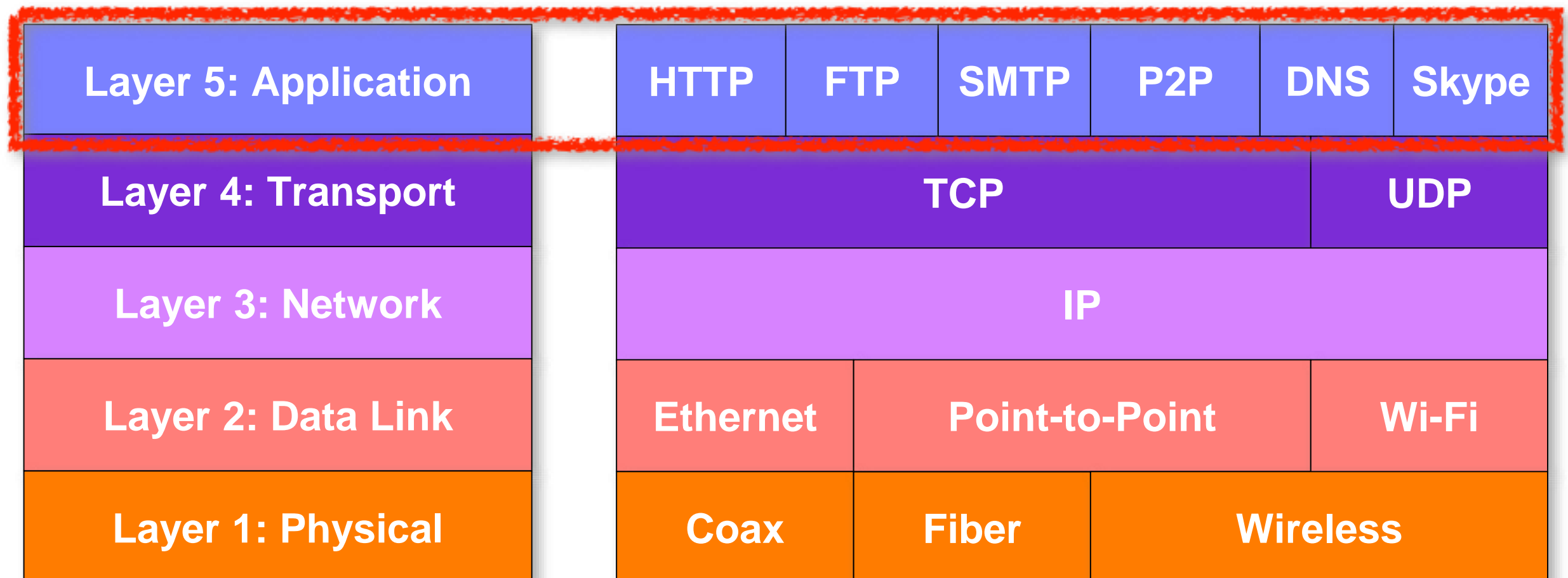


Overview of Application Layer

- **Network Application Architectures**
 - Protocol Layers
 - Client-Server vs. Peer-to-Peer
 - Process Communication
 - Transport Services
- **HyperText Transfer Protocol (HTTP)**
- **File Transfer and Email protocols (FTP, SMTP)**
- **Domain Name System (DNS)**
- **Peer-to-Peer Applications (P2P)**

Protocol Layers

- **Top-Down Approach**



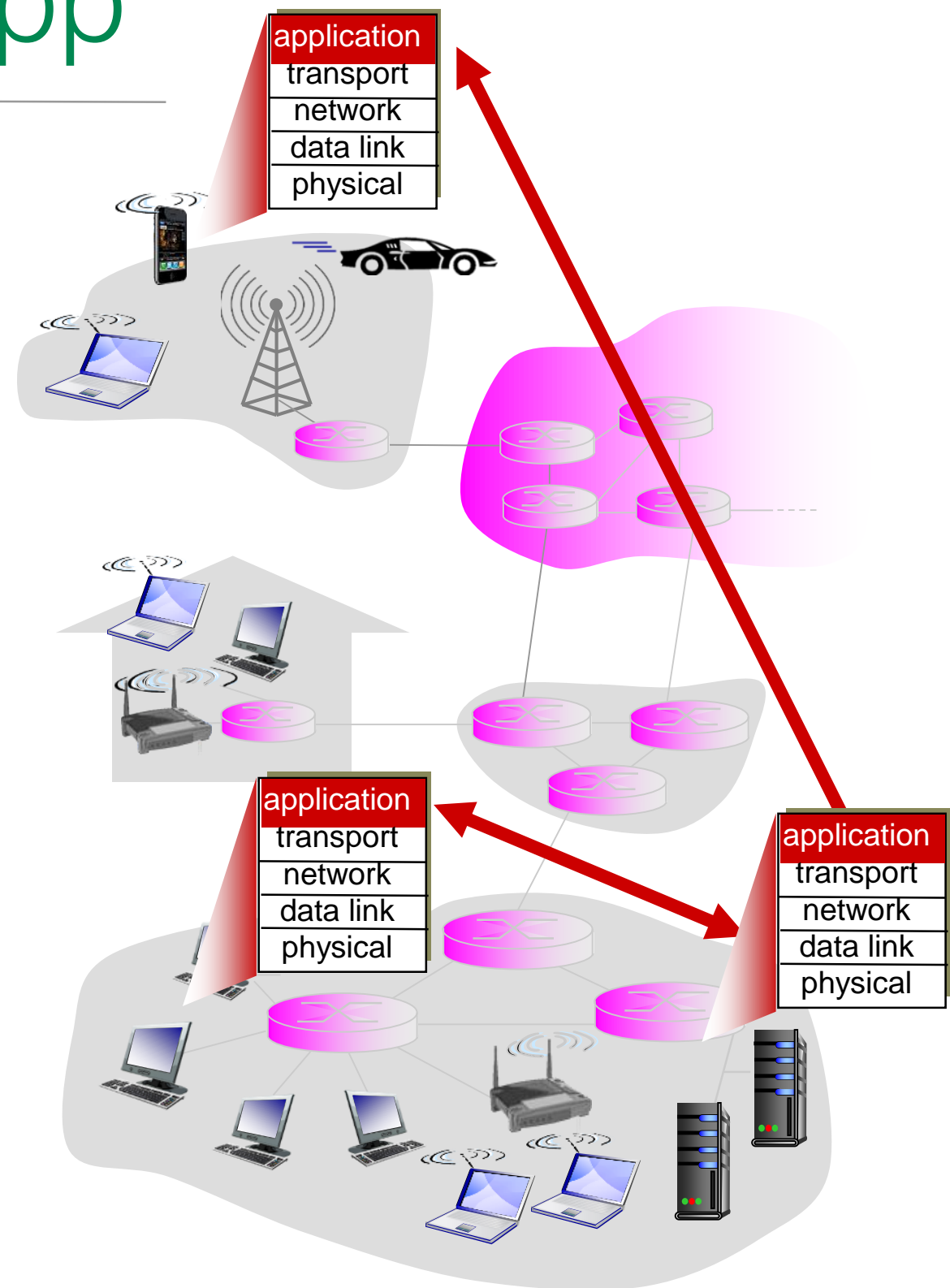
Example Applications

- **E-mail**
- **Web**
- **Text messaging**
- **Remote login**
- **P2P file sharing**
- **Multi-user network games**
- **Streaming stored video**
(YouTube, Hulu, Netflix)
- **Voice over IP (e.g. Skype)**
- **Real-time video conferencing**
- **Social networking**
- **Search**
- **...**

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



Network Application Architectures

- **Client-Server**
- **Peer-to-Peer**

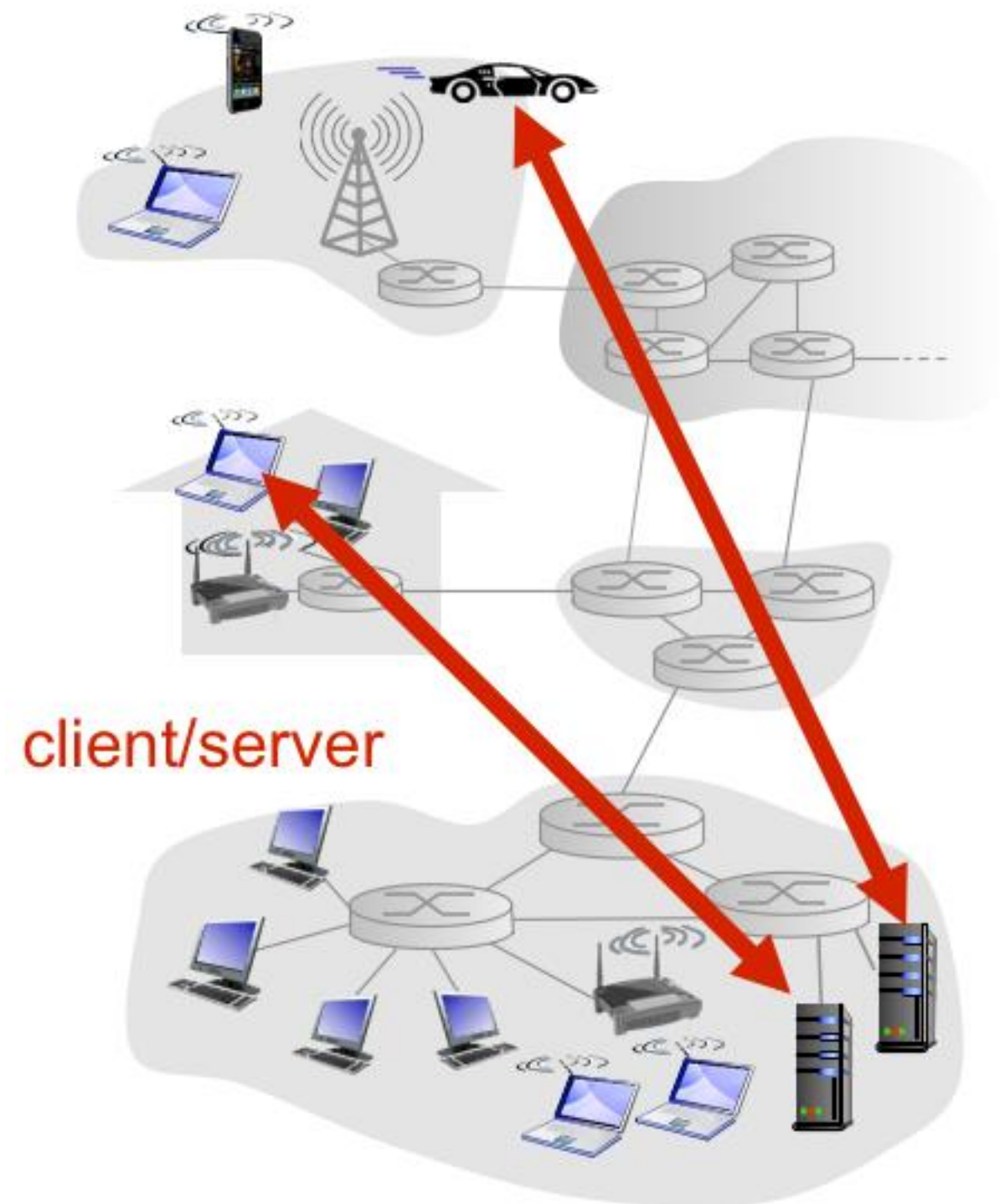
Client-Server Architecture

- **Clients:**

- Request service from server
- May be intermittently connected
- May have dynamic IP addresses
- Do not communicate directly with each other

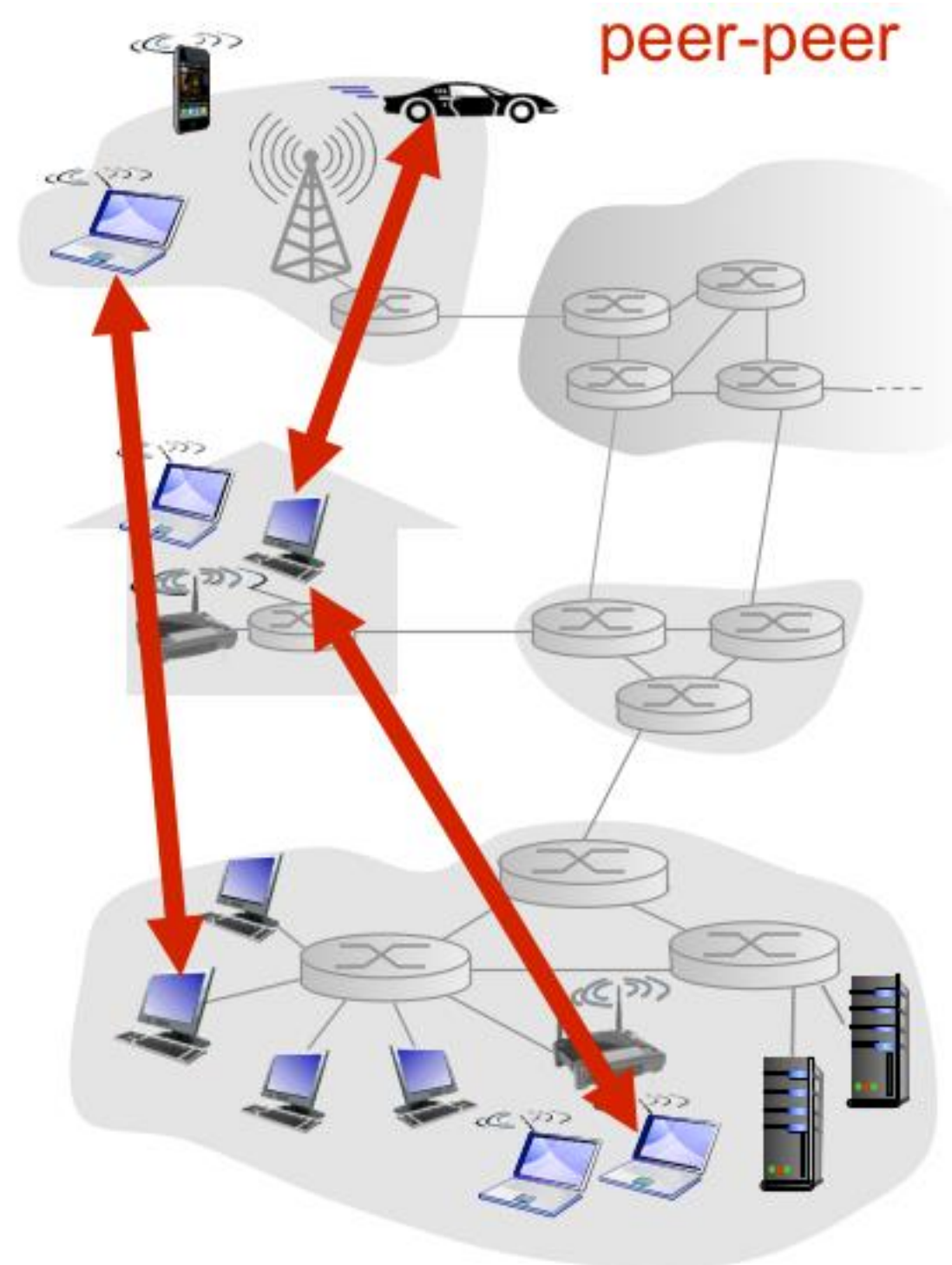
- **Server:**

- Provides a service to clients
- Always-on host
- Permanent IP address
- Data centers for scaling



Peer-to-Peer Architecture

- Does not require always-on servers
- Hosts communicate directly with each other
- Peers request service from other peers, and provide service in return to other peers
- Highly scalable
- Self scalability – new peers bring new service capacity, as well as new service demands
- Hosts are intermittently connected and may change IP addresses
- Difficult to authenticate - possibly insecure
- Hosts need incentive to share data



Process Communications

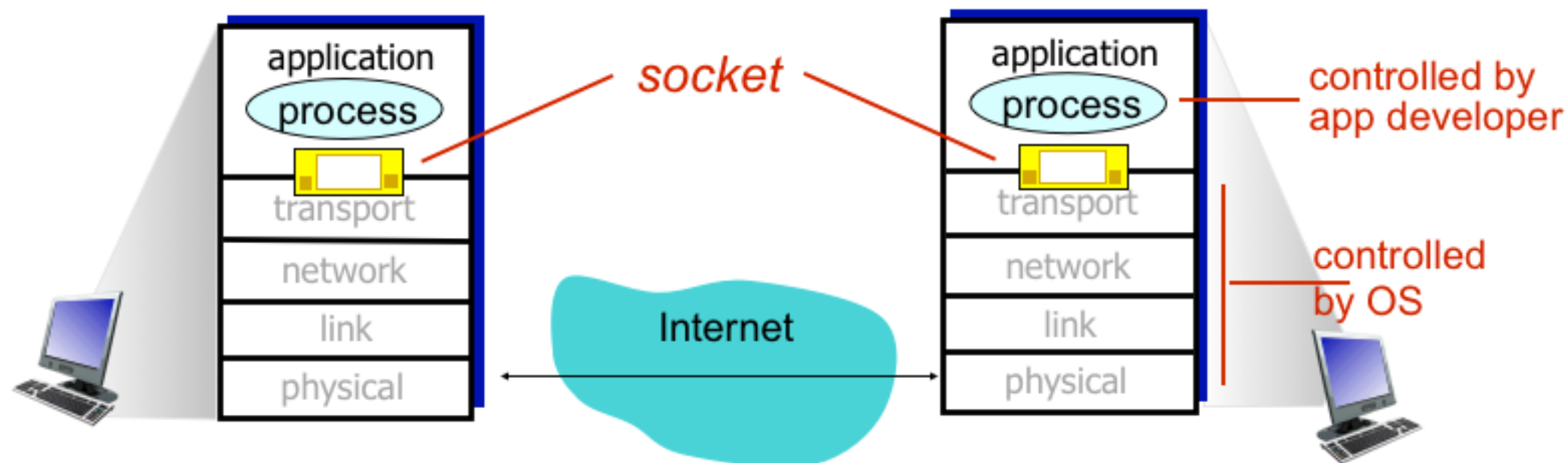
- **A process is program running on a host**
 - Client process - process that initiates communication
 - Server process - process that waits to be contacted

Note: Applications with P2P architectures have both client processes & server processes

- **Sockets provide a mechanism for inter-process communication (IPC)**
 - Inter-process communication on the same host
 - Operating system provides message passing
 - Inter-process communication on different hosts
 - Network provides message passing

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



Addressing Processes

- To receive messages, process must have some **identifier**
- Host device has unique 32-bit IP address (IPv4)
 - IP address alone is insufficient to address a process on a system
 - Many processes may be running on the same system
- Identifier includes both **IP address and port numbers associated with process on host**
 - Example port numbers:
 - HTTP server: port 80
 - SMTP Mail server: port 25
- To send an **HTTP** message to **cs.ycp.edu** web server:
 - IP address: 192.245.87.64
 - Port number: 80

Application Layer Protocol Defines

- **What types of messages are exchanged**
 - e.g. request, response
- **Message syntax:**
 - What fields are in messages
 - How fields are delineated
- **Message semantics:**
 - Meaning of information in fields
 - Rules for when and how processes send & respond to messages

What Transport Service Does an Application Need?

- **Data integrity**

- Some applications require 100% reliable data transfer (e.g. file transfer, web transactions)
- Other applications can tolerate some loss (e.g. audio)

- **Timing**

- Some applications require low delay to be “effective” (e.g. Internet telephony, interactive games)

- **Throughput**

- Some applications require minimum amount of throughput to be “effective” (e.g., multimedia)
- Other “elastic” applications make use of whatever throughput they get

- **Security**

- Encryption, data integrity, etc.

Transport service requirements: common apps

application	data loss	throughput	time sensitive
file transfer	no loss	elastic	no
e-mail	no loss	elastic	no
Web documents	no loss	elastic	no
real-time audio/video	loss-tolerant	audio: 5kbps-1Mbps video: 10kbps-5Mbps	yes, 100' s msec
stored audio/video	loss-tolerant	same as above	yes, few secs
interactive games	loss-tolerant	few kbps up	yes, 100' s msec
text messaging	no loss	elastic	yes and no

Internet Transport Protocol Services

TCP (Transmission Control Protocol)	UDP (User-Datagram Protocol)
Reliable data transfer	Unreliable data transfer
Packet sequence # required	Sequence # is optional
Every packet is acked	Not acked
Lost packets are retransmitted	No retransmission
May cause long delay	Quick and lossy
Connection-oriented service	Connection-less service
Good for reliable and delay-insensitive applications	Good for loss-tolerant and delay sensitive applications
Application examples: email, HTTP, FTP, remote terminal access	Application examples: Telephony, streaming multimedia

Securing TCP

TCP & UDP

- no encryption
- cleartext passwds sent into socket traverse Internet in cleartext

SSL

- provides encrypted TCP connection
- data integrity
- end-point authentication

SSL is at app layer

- apps use SSL libraries, that “talk” to TCP

SSL socket API

- cleartext passwords sent into socket traverse Internet encrypted

Internet apps: application, transport protocols

application	application layer protocol	underlying transport protocol
e-mail	SMTP [RFC 2821]	TCP
remote terminal access	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
file transfer	FTP [RFC 959]	TCP
streaming multimedia	HTTP (e.g., YouTube), RTP [RFC 1889]	TCP or UDP
Internet telephony	SIP, RTP, proprietary (e.g., Skype)	TCP or UDP

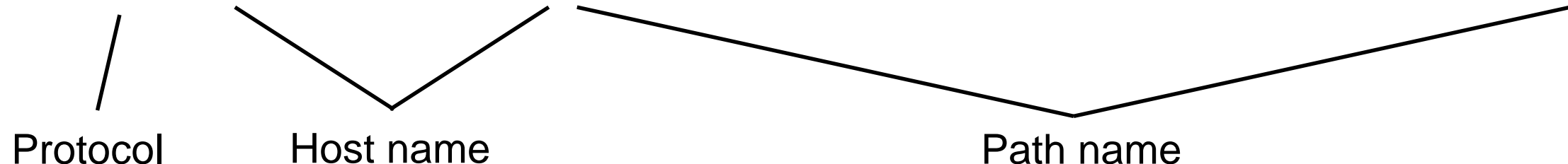
Overview of Application Layer

- **Network Application Architectures**
- **HyperText Transfer Protocol (HTTP)**
 - HTTP Overview
 - HTTP Communication
 - HTTP Request Message
 - HTTP Response Message
 - Cookies
 - Web Caching
- **File Transfer and Email protocols (FTP, SMTP)**
- **Domain Name System (DNS)**
- **Peer-to-Peer Applications (P2P)**

HTTP Overview

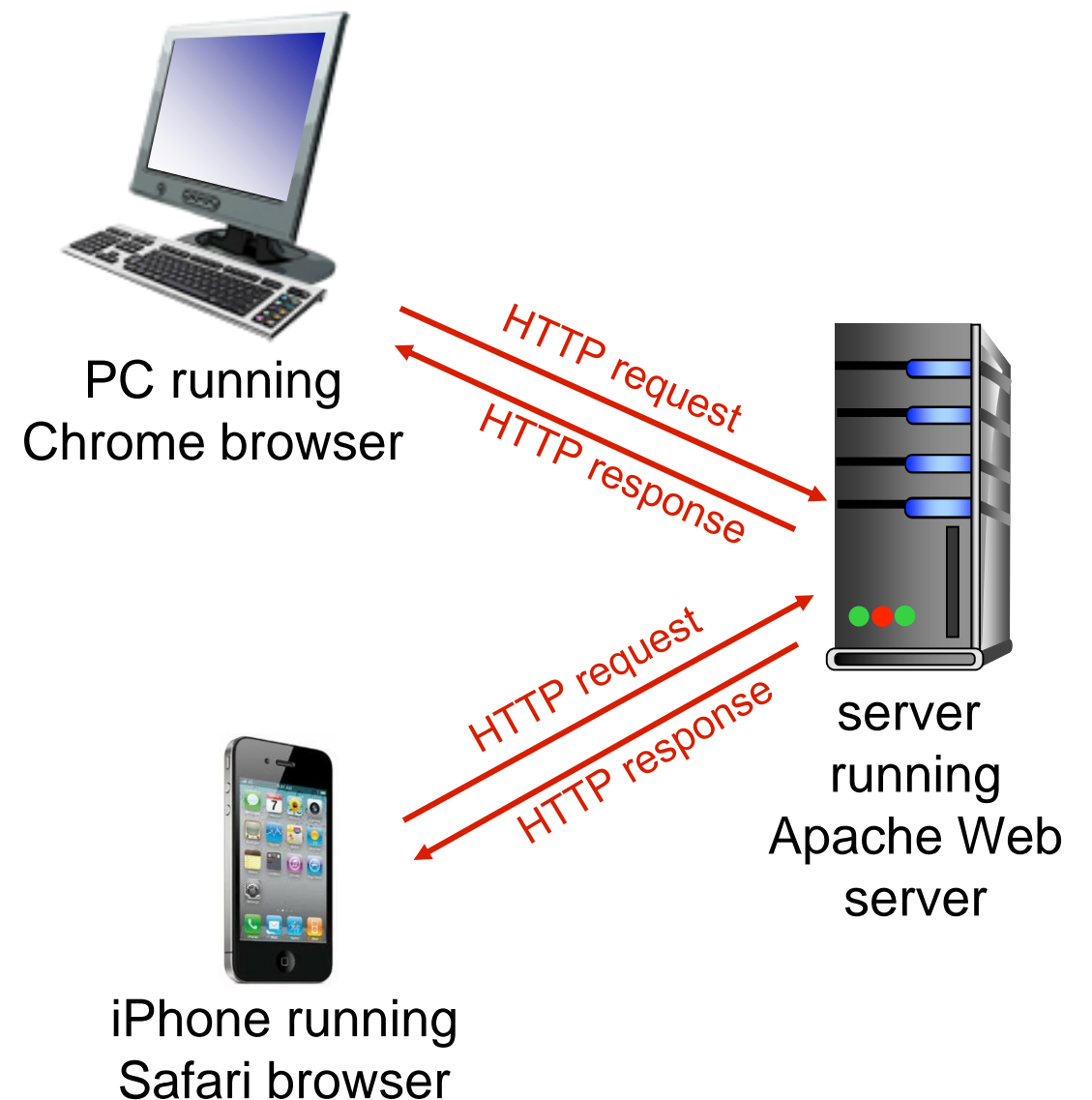
- **HTTP client - web browser (e.g. Chrome, Firefox, Safari)**
- **HTTP server - web server (Apache, Microsoft Internet Information Service (IIS))**
- **Web page consists of:**
 - A group of objects
 - HTML files, images, Java applets, audio files, etc.
 - A base HTML file which references objects
 - Each object is addressable by a URL

http://faculty.ycp.edu/~alice/classes/cs330/docs/cs330_syllabus.pdf



HTTP Overview (Cont.)

- **HTTP: hypertext transfer protocol**
- **Web's application layer protocol**
- **Uses 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



HTTP Communication

- **Uses TCP**

- Client initiates TCP connection (creates socket) to server, port 80
- Server accepts TCP connection from client
- HTTP messages exchanged between web browser (HTTP client) and web server (HTTP server)
- TCP connection closed

- **HTTP is “stateless”**

- Server maintains no information about past client requests

HTTP Connections

- **Non-persistent HTTP**

- At most one object is sent over a TCP connection
 - Open connection, get one object, close connection
- Downloading multiple objects requires multiple connections

- **Persistent HTTP**

- Multiple objects can be sent over single TCP connection between client and server
 - Server leaves the connection open after sending an object and closes on timeout

HTTP Request Message

- Two types of HTTP messages: **request, response**
- **HTTP Request Message**
 - ASCII (human-readable format)

Request line
(GET, POST,
HEAD commands)

GET /~alice/index.html HTTP/1.1\r\n

Header Lines

Host: faculty.ycp.edu\r\n

User-Agent: Firefox/3.6.10\r\n

Accept: text/html,application/xhtml+xml\r\n

Accept-Language: en-us,en;q=0.5\r\n

Accept-Encoding: gzip,deflate\r\n

Accept-Charset: ISO-8859-1,utf-8;q=0.7\r\n

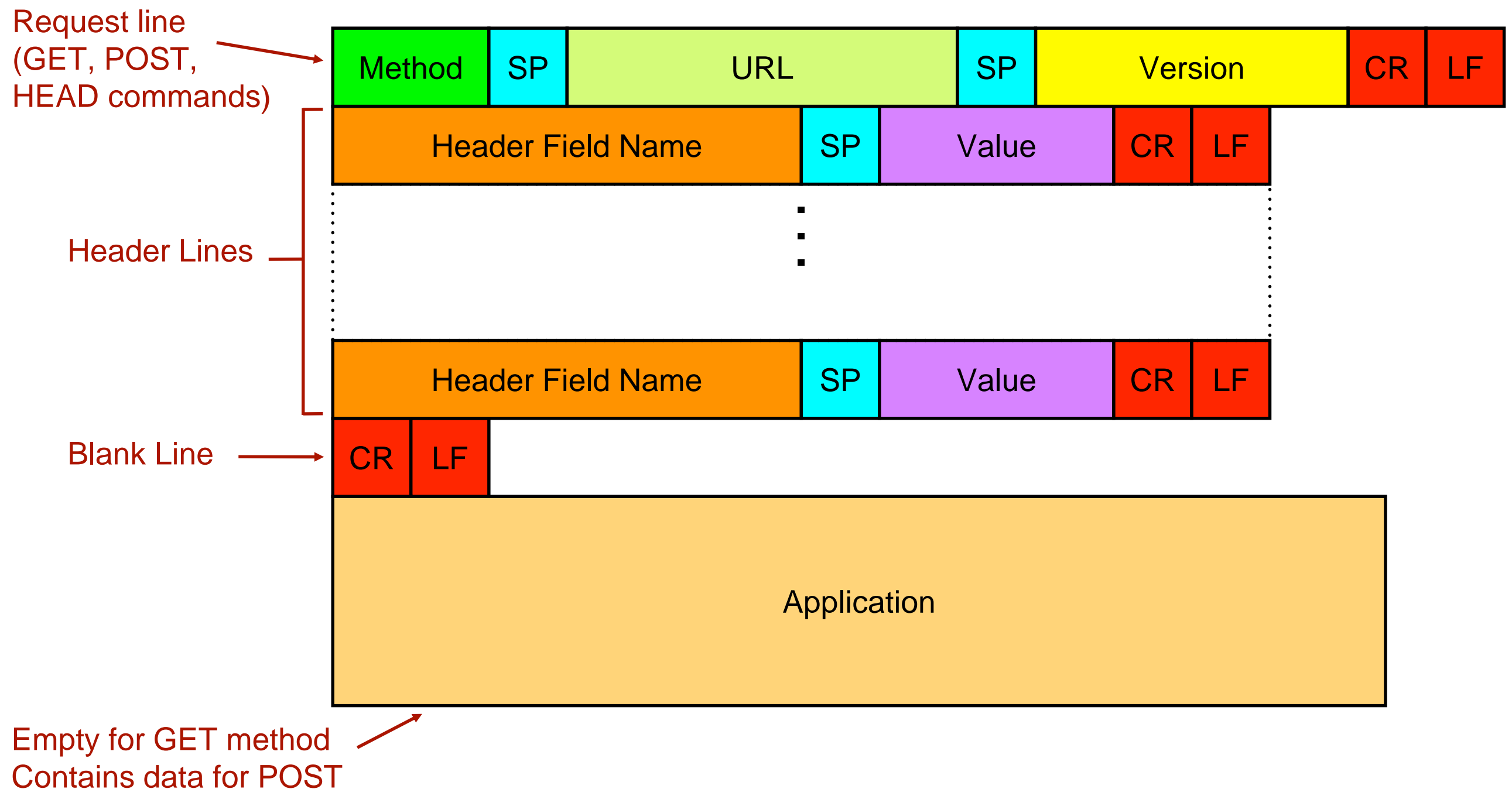
Keep-Alive: 115\r\n

Connection: keep-alive\r\n

\r\n

Carriage return,
line feed at start
of line indicates
end of header lines

HTTP Request Message: General Format



HTTP Methods

- **GET** - used to request an object from a server
 - Requested object is in URL field of HTTP request message
- **HEAD** - same as GET, but only sends header
 - Doesn't actually send requested object
 - Useful for testing/debugging
- **POST** - used to send information to a server when requesting an object
 - The object returned may depend on the information posted
 - Often used when filling out web forms
- **PUT** - uploads file in entity body to path specified in URL field
- **DELETE** - deletes file specified in the URL field

Method types

HTTP/1.0:

- **GET**
- **POST**
- **HEAD**
 - asks server to leave requested object out of response

HTTP/1.1:

- **GET, POST, HEAD**
- **PUT**
 - uploads file in entity body to path specified in URL field
- **DELETE**
 - deletes file specified in the URL field

Uploading Form Input

- **Two methods:**

- POST method:

- Web page often includes form input
 - Input is uploaded to server in the Entity portion of the HTTP request message

- URL method:

- Uses HTTP GET method
 - Input is uploaded in URL field of HTTP request message

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

HTTP Response Message

status line
(protocol status
code status
phrase)

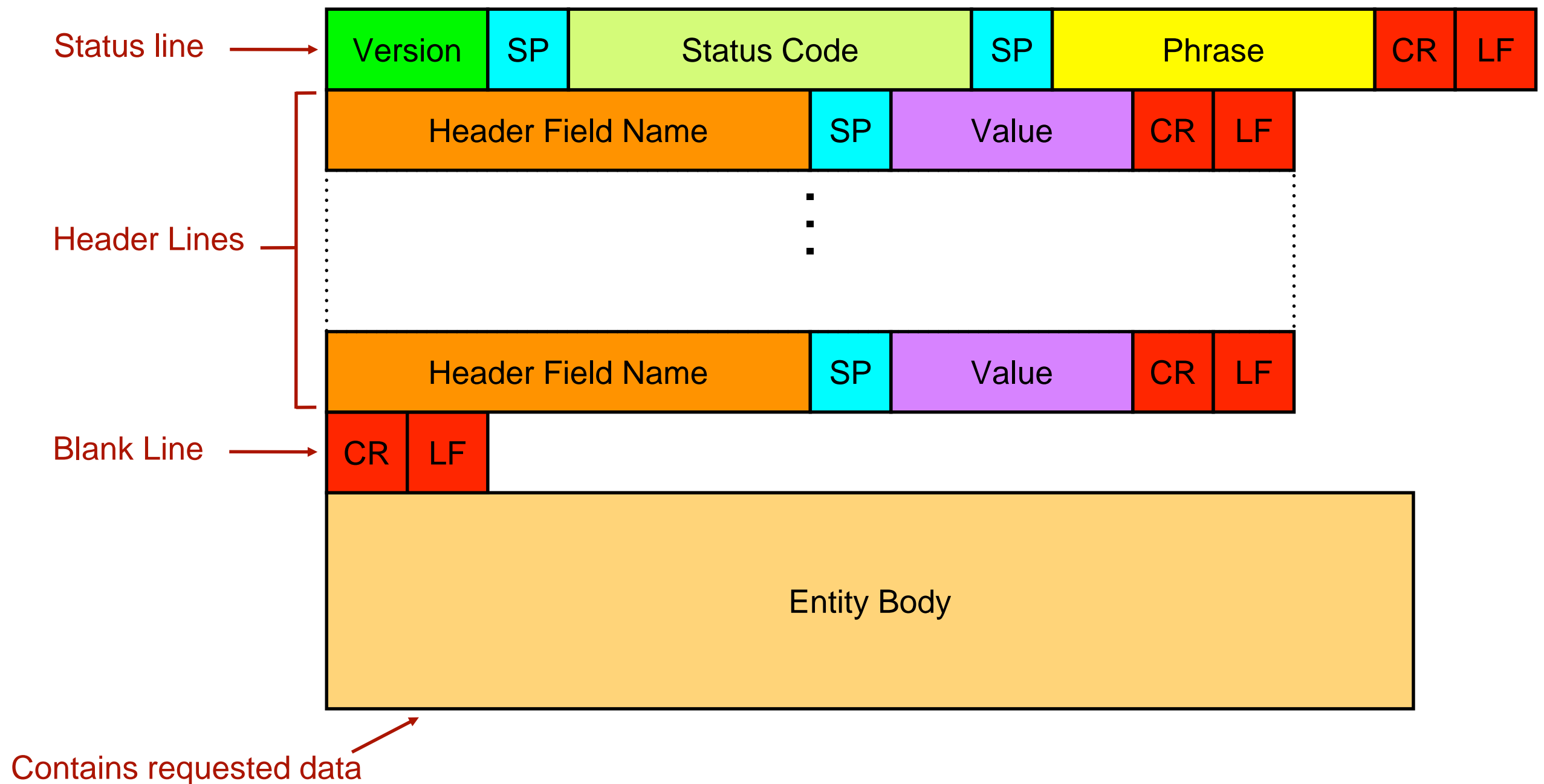
Header Lines

Carriage return,
line feed at start
of line indicates
end of header lines

data (e.g.
requested
HTML file)

```
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 Message: General Format



HTTP Response Status Codes

- Status code appears in first line in server-to-client response message

- Some sample codes:

200 OK

- Request succeeded, requested object later in this message

301 Moved Permanently

- Requested object moved, new location specified later in this msg (Location:)

400 Bad Request

- Request message not understood by server

404 Not Found

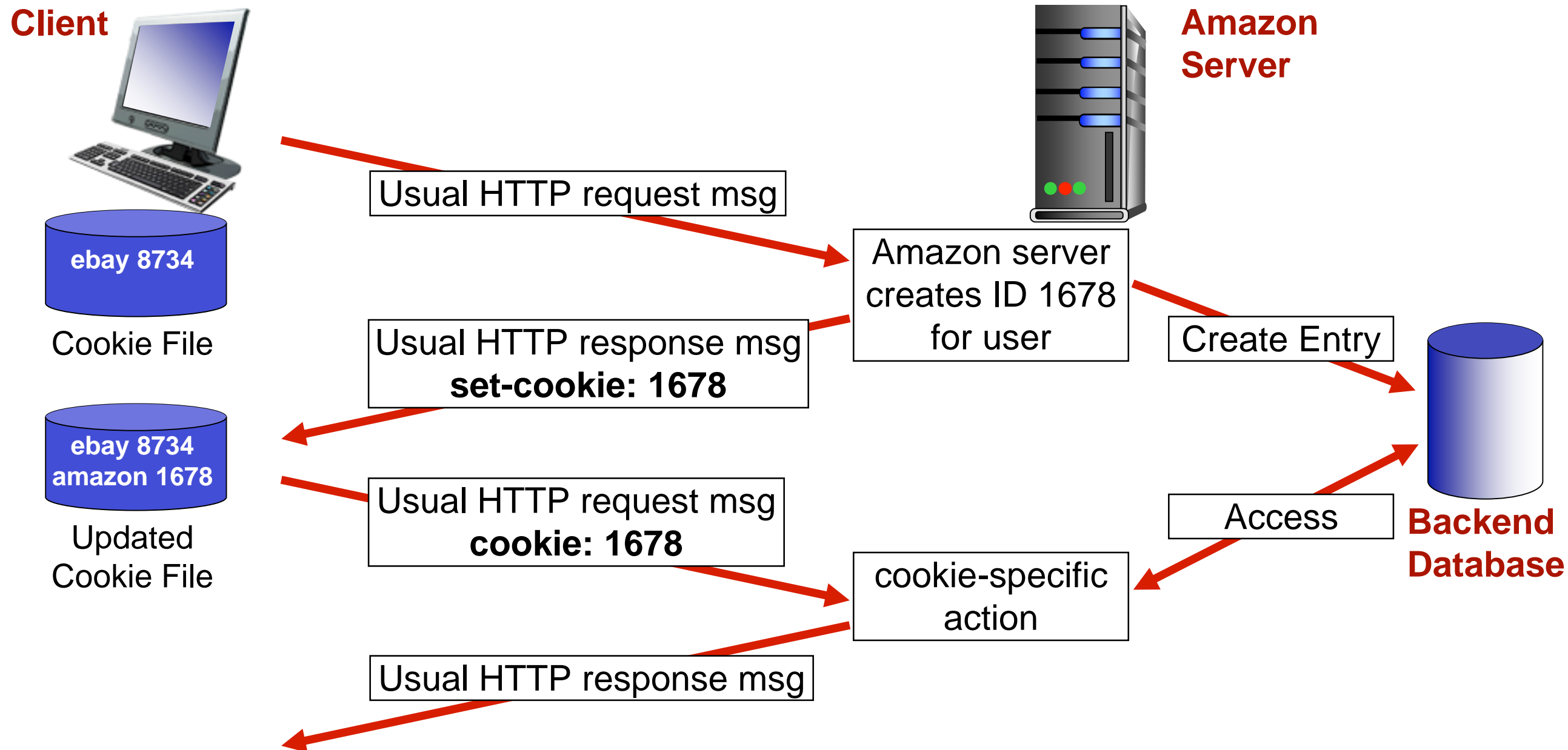
- Requested document not found on this server

505 HTTP Version Not Supported

User-Server State: Cookies

- **Cookies allow servers to remember previous information about user**
 - Stored in file on end user system since HTTP server is stateless
 - Many companies use cookies to identify a user
 - Content is dependent on the identity of the user
- **Four components to cookies:**
 - A cookie header line in the HTTP response message
 - A cookie header line in the HTTP request message
 - A cookie file kept on user's end system & managed by user's browser
 - A back-end database at web site

Cookies: Keeping “state”



Cookies

- **What cookies can be used for:**

- Authorization
- Shopping carts
- Recommendations
- User session state (web-based e-mail)

cookies and privacy:

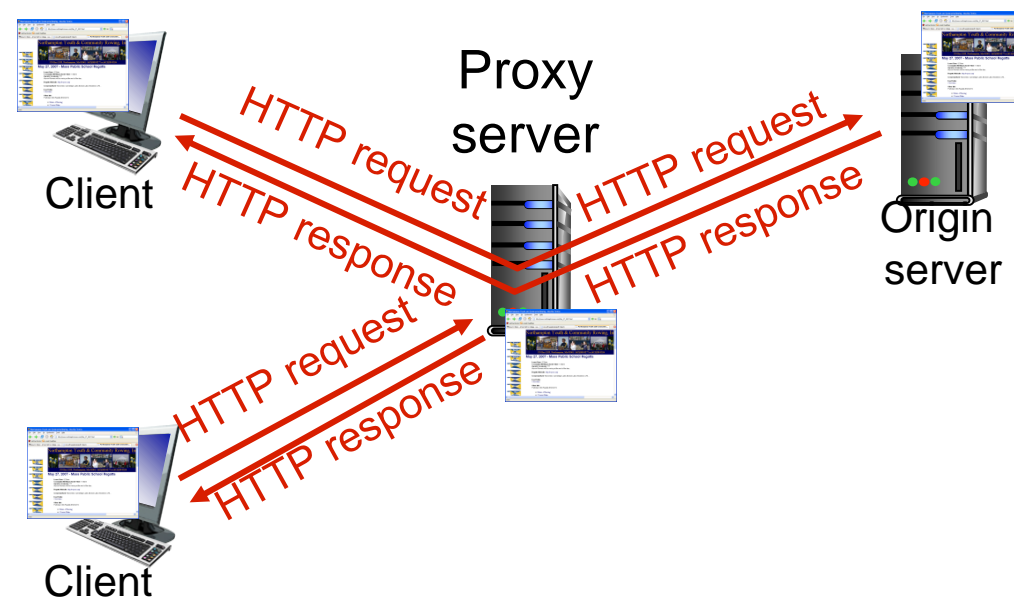
- cookies permit sites to learn a lot about you
- you may supply name and e-mail to sites

- **How to keep “state”:**

- Protocol endpoints: maintain state at sender/receiver over multiple transactions
- Cookies: http messages carry state

Web Caches (proxy server)

- **A network entity that satisfies requests on behalf of an origin web server**
 - All requests are sent to proxy server
 - Proxy server caches objects
 - Only new objects are requested from origin server



More About Web Caching

- **Cache acts as both client and server**
 - Server for requesting clients
 - Client to the origin server
- **Typically, cache is installed by ISP (university, company, residential ISP)**
- **Why Web caching?**
 - Reduce response time for client request
 - Reduce traffic on an institution's access link

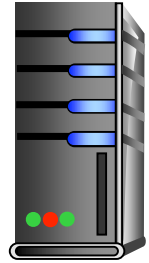
Conditional GET

- **Goal: don't send object if cache has up-to-date cached 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

Client



Server



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

Object not
modified

HTTP response
HTTP/1.0 304 Not Modified



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

Object
modified

HTTP response
HTTP/1.0 200 OK <data>