DATA COMMUNICATOIN NETWORKING

Instructor: Ouldooz Baghban Karimi

Course Book: Computer Networking, A Top-Down Approach
By: Kurose, Ross

Course Overview

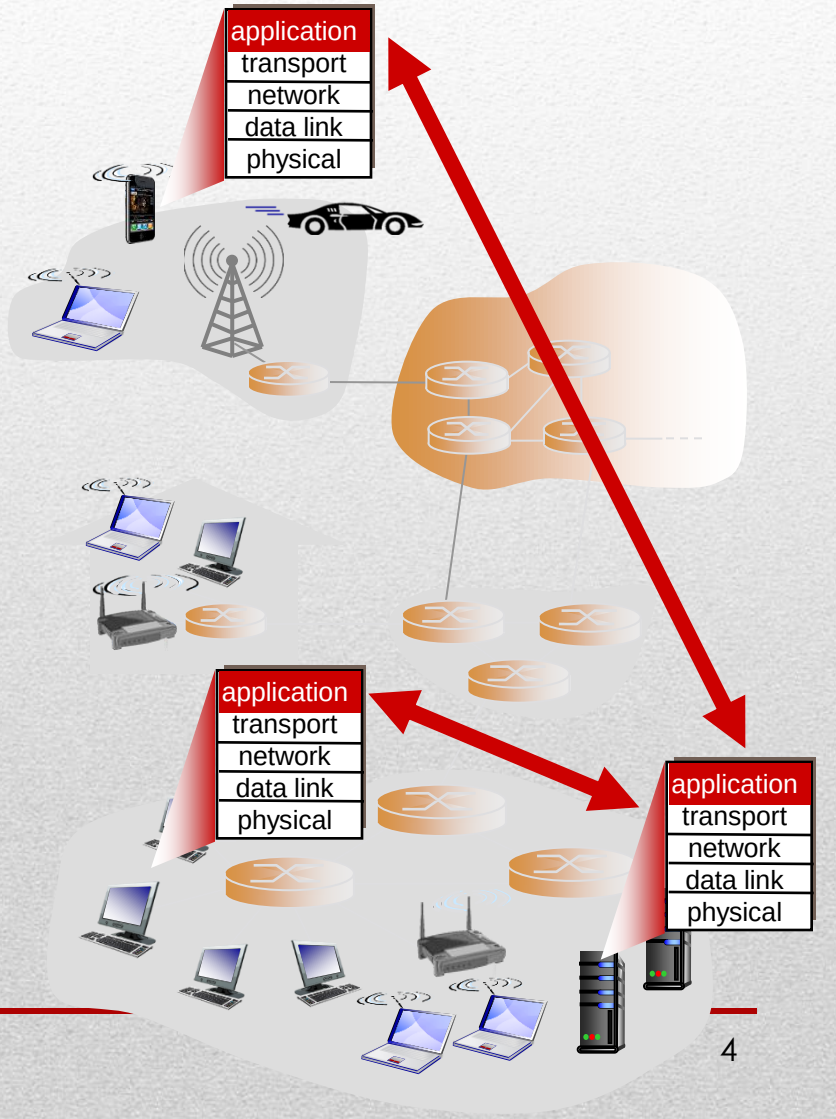
- **Basics of Computer Networks**
  - Internet & Protocol Stack
  - Application Layer
  - Transport Layer
  - Network Layer
  - Data Link Layer
- **Advanced Topics**
  - Case Studies of Computer Networks
  - Internet Applications
  - Network Management
  - Network Security

Application Layer

- **Conceptual & implementation aspects of network application protocols**
  - Client-Server
  - Peer to Peer
- **Applications & application level protocols**
  - HTTP
  - FTP
  - SMTP/POP3/IMAP
  - DNS
- **Create Network Applications**
  - Socket API

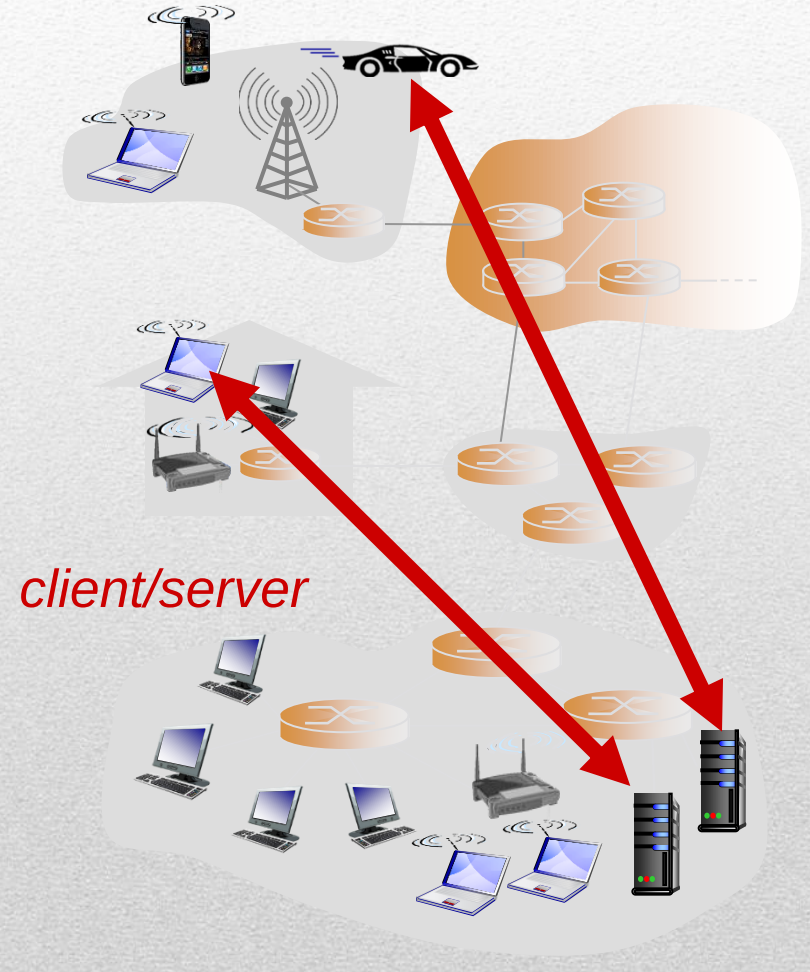
Creating a Network Application

- **Programs run on end systems**
  - Network core devices do not run user applications
  - Communication over network
- **Application Architectures**
  - Peer to peer
  - Client-Server



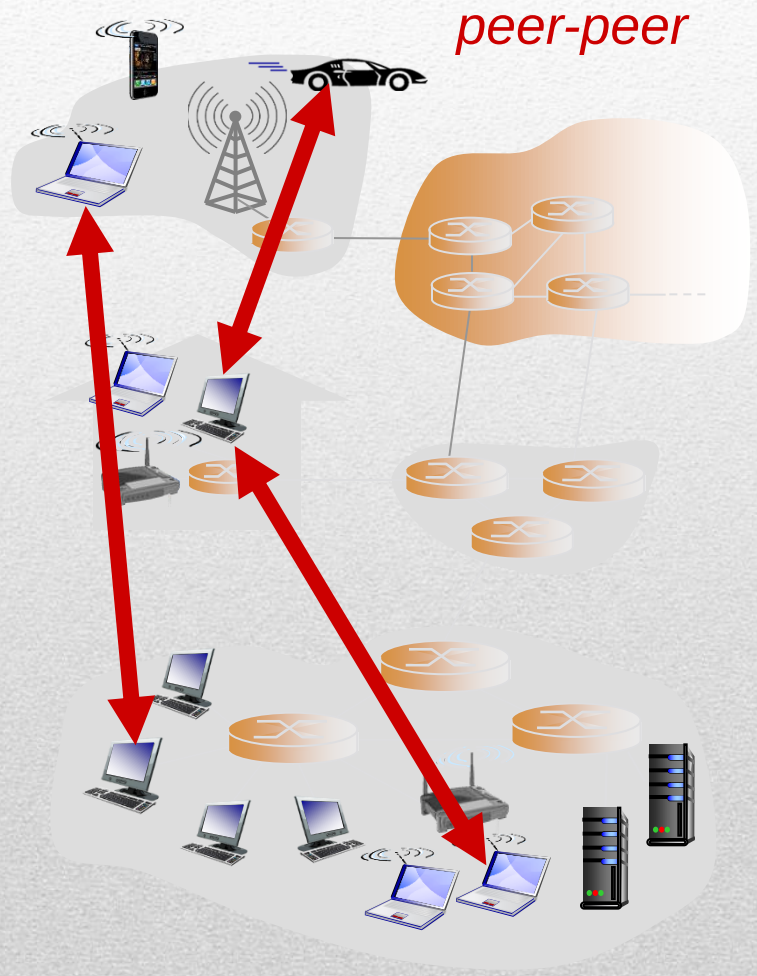
Client-Server Structure

- **Server**
  - Permanent IP address
  - Always on host
  - Data centers for scaling
- **Client**
  - Communicate with server
  - May be intermittently connected
  - May have dynamic IP addresses
  - Do not communicate directly with each other



Peer to Peer Architecture

- **Arbitrary end systems**
  - Direct communication
- **Peers request service from each other and provide service to each other in return**
  - Self scalability
    - Every new peer, a new service capacity
      - New service
      - New demand
- **Intermitted connection of peers**
  - Changed IP addresses
  - Complex management



Process Communication

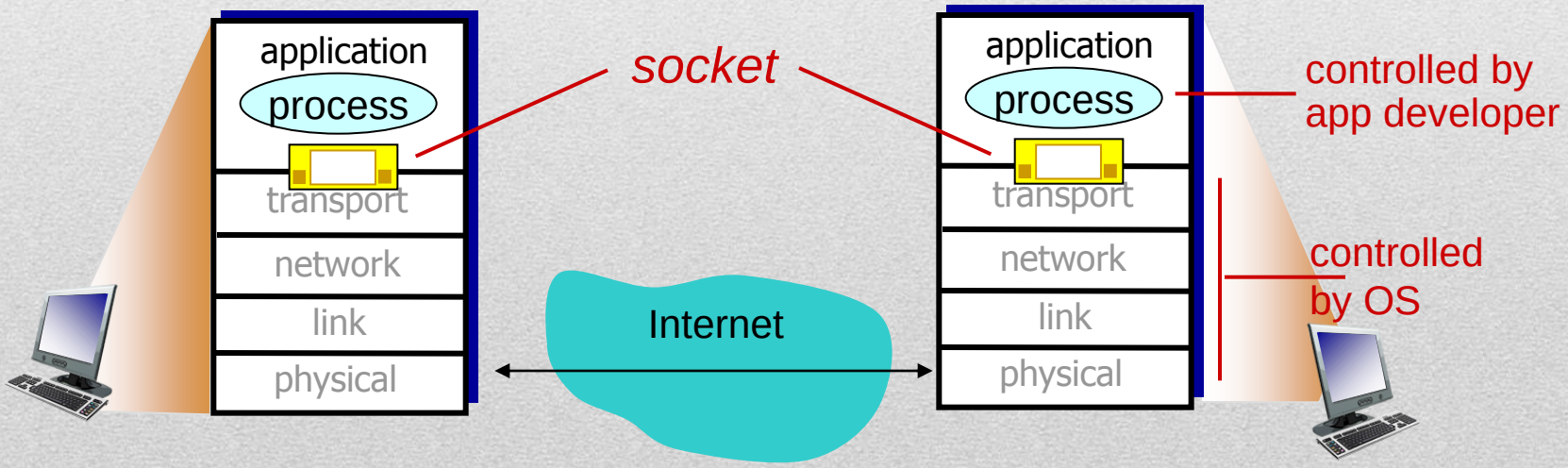
- **Process**
  - Program running within an end host
- **Process Communication**
  - Within the same host
    - Inter-process communication
    - Defined by OS
  - Unication
    - Defined by OS
  - In different hosts
    - Exchanging messages
- **P2P applications have client and server processes as well**

client process: process that initiates communication

server process: process that waits to be contacted

Sockets

- **Process send/receive messages to/from their sockets**
  - Sending process relies on transport infrastructure on other side of door to deliver message to socket at receiving process



Addressing Processes

- **IP Address of the host – Enough?**
  - IP address
  - Port numbers associated with the process
  - Example
    - HTTP: 80
    - Mail Server: 25

- **Example: HTTP message to gaia.cs.umass.edu**
  - IP address: 128.119.245.12
  - Port number: 80

Application Layer Protocol

- **Defines**
  - Types of messages exchanged
    - e.g. request, response
  - Message Syntax
    - Fields in the message and their values
  - Message Semantics
    - Meaning of the information in the fields
  - Rules for when and how to send and receive messages

- Open protocols:
  - defined in RFCs
  - allows for interoperability
  - e.g., HTTP, SMTP
- Proprietary protocols:
  - e.g., Skype

Transport Service Requirements

- **Data Integrity**
  - Reliable transfer (FTP)
  - Loss tolerant applications (Multimedia)
- **Timing**
  - Low delay
    - Online gaming
    - VoIP
- **Throughput**
  - Elastic
  - Inelastic: Multimedia
- **Security**

Transport Service Requirements

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**
  - Reliable transport
  - Flow control
  - Congestion Control
  - Connection Oriented
  - Does not Provide
    - Timing
    - Minimum throughput
    - Guarantees for security
- **UDP**
  - Unreliable data transfer
  - Does not provide
    - Timing
    - Minimum throughput
    - Security
    - Congestion Control
    - Flow Control

Why do we need UDP?

Internet Transport Protocol Services

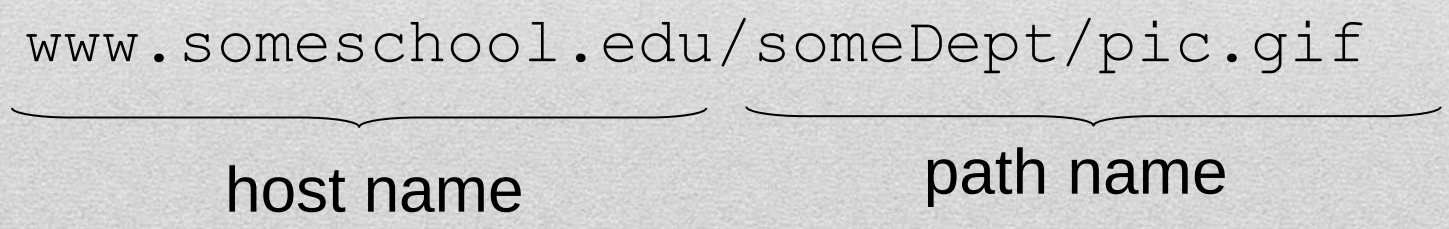
application	application layer protocol	underlying transport protocol
e-mail	SMTP [RFC 2821]	TCP
remote terminal access	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
file transfer	FTP [RFC 959]	TCP
streaming multimedia	HTTP (e.g., YouTube), RTP [RFC 1889]	TCP or UDP
Internet telephony	SIP, RTP, proprietary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

Securing TCP

- **TCP & UDP**
  - No encryption
  - Cleartext passwords sent into socket traverse internet in cleartext
    - Guarantees for security
- **SSL**
  - Provides encrypted TCP connection
  - Data integrity
  - End-point authentication
  - SSL Socket API
    - Apps use SSL libraries which talk to TCP
    - Cleartext passwords sent into socket traverse Internet encrypted

Web & HTTP

- **Web Page**
  - Consists of objects
    - HTML file
    - JPEG image
    - Audio file
  - Base HTML file which includes several referenced objects
    - Each object is addressable by a URL



HTTP: HyperText Transfer Protocol

- **Application layer protocol**
  - Client-server model
    - Client: browser that requests, receives and displays the Web objects
    - Server: Web server sends objects in response to requests

HTTP Connections

- **Uses TCP**
  - Client initiates TCP connection (creates socket) to server, port 80
  - Server accepts TCP connection from client
  - HTTP messages (application layer messages) exchanged between browser (HTTP Client) & web serer
  - TCP Connection Closed

HTTP is "stateless"

server maintains no information about past client requests

protocols that maintain "state" are complex!

past history (state) must be maintained

if server/client crashes, their views of "state" may be inconsistent, must be reconciled

HTTP Connections

- **Non persistent HTTP**
  - At most one object sent over TCP connection
    - After than connection is closed
  - Downloading multiple objects requires multiple connections

- **Persistent HTTP**
  - Multiple objects can be sent over single TCP connection between client and server

Non-persistent HTTP

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

1a. HTTP client initiates TCP connection to HTTP server (process) at www.someSchool.edu on port 80

1b. HTTP server at host www.someSchool.edu waiting for TCP connection at port 80. "accepts" connection, notifying client

2. HTTP client sends HTTP *request message* (containing URL) into TCP connection socket. Message indicates that client wants object someDepartment/home.index

3. HTTP server receives request message, forms *response message* containing requested object, and sends message into its socket

Non-persistent HTTP

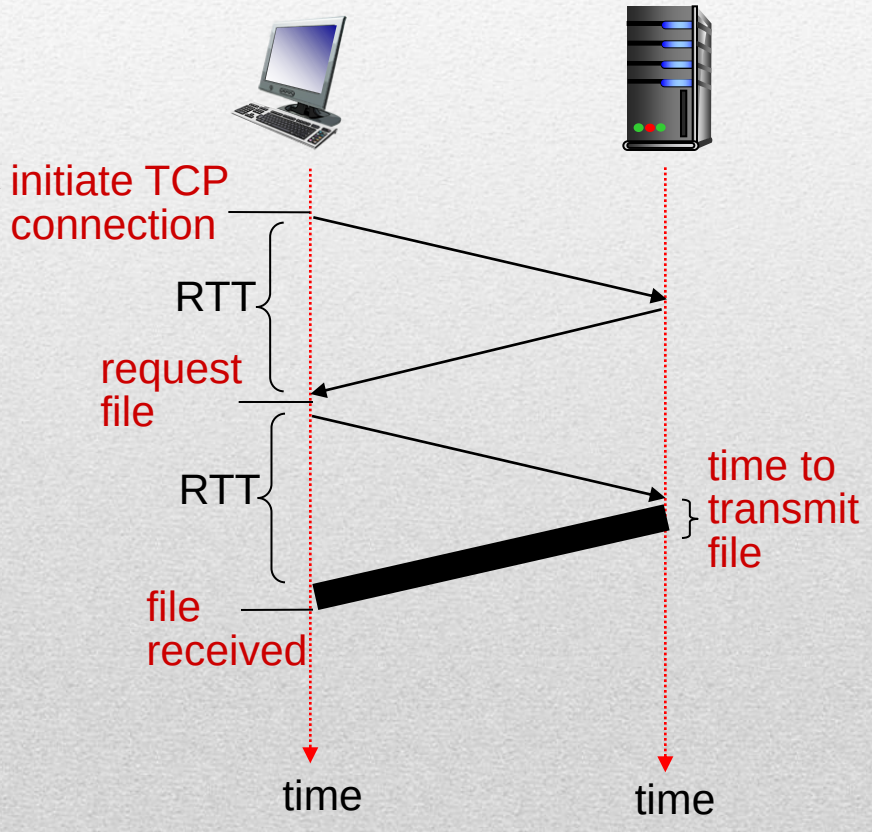
4. HTTP server closes TCP connection.

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

Non Persistent HTTP: Response Time

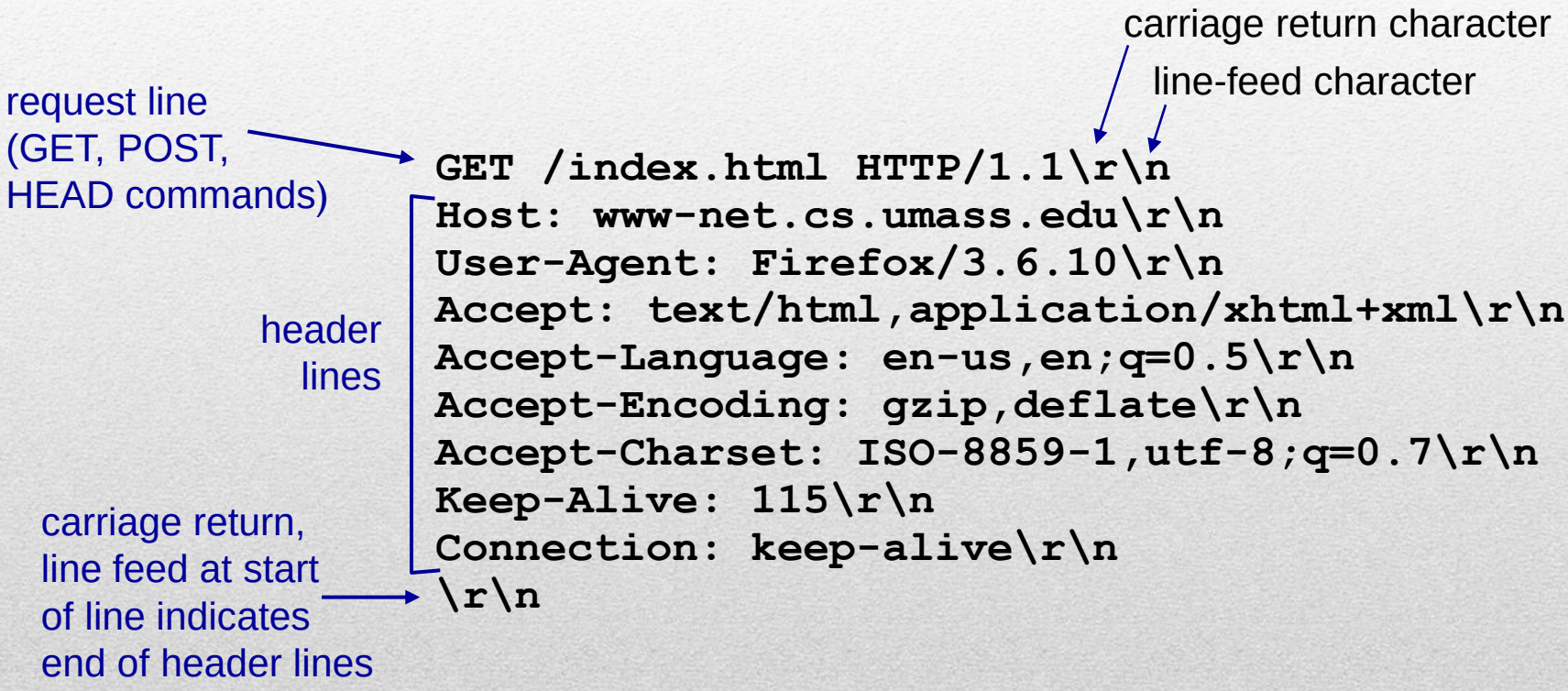
- **RTT**
  - Round Trip Time
    - Time for a packet to travel from client to server and back
- **HTTP Response Time:**
  - 1RTT: initiate TCP connection
  - 1 RTT: GTTP request & first few bytes of HTTP response to return file transmission time
  - Non persistent HTTP response time =
    2RTT+file transmission time



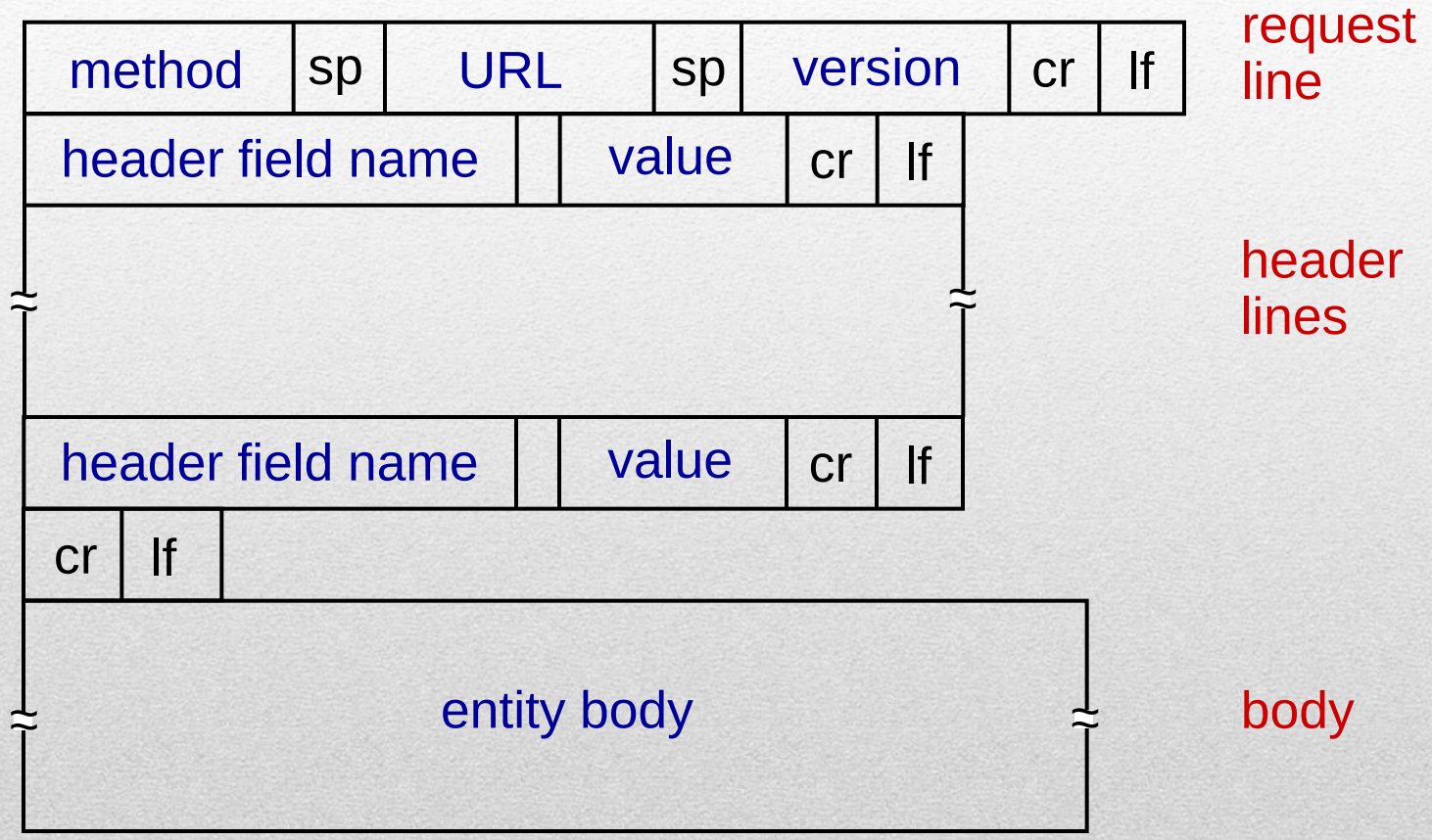
Persistent HTTP

- **Non persistent HTTP problems**
  - Requires 2 RTT per object
  - OS overhead for each TCP connection
  - Browsers often 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 connections
  - Client sends requests as soon as it encounters a referenced object
  - As little as one RTT for all the referenced objects

HTTP Request Message



HTTP Request Message: General Format



Uploading From 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
```

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

Method Types

status line (protocol status code status phrase)

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 Status Codes

Status Code appears in the first line in the server to client response

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

505 HTTP Version Not Supported