# Announcement

- Homework 1 was due 30 seconds ago…
- Link to the course applets

  http://wps.pearsoned.com/ecs_kurose_compnetw_6/216/55463/14198700.cw/

# Last Time

- Access networks: DSL, cable networks, Ethernet, wireless LAN, wide-area wireless access, *etc.*
- Packet switching and circuit switching
- Nodal delay = processing delay + queueing delay + transmission delay + propagation delay
- End-to-end throughput
- Internet Protocol Stack: application layer, transport layer, network layer, link layer, physical layer

# Chapter 2: outline

2.1 principles of network applications

2.2 Web and HTTP

2.3 FTP

2.4 electronic mail

- SMTP, POP3, IMAP

2.5 DNS

2.6 P2P applications

2.7 socket programming with UDP and TCP

# Some network apps

- 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
- …
- …

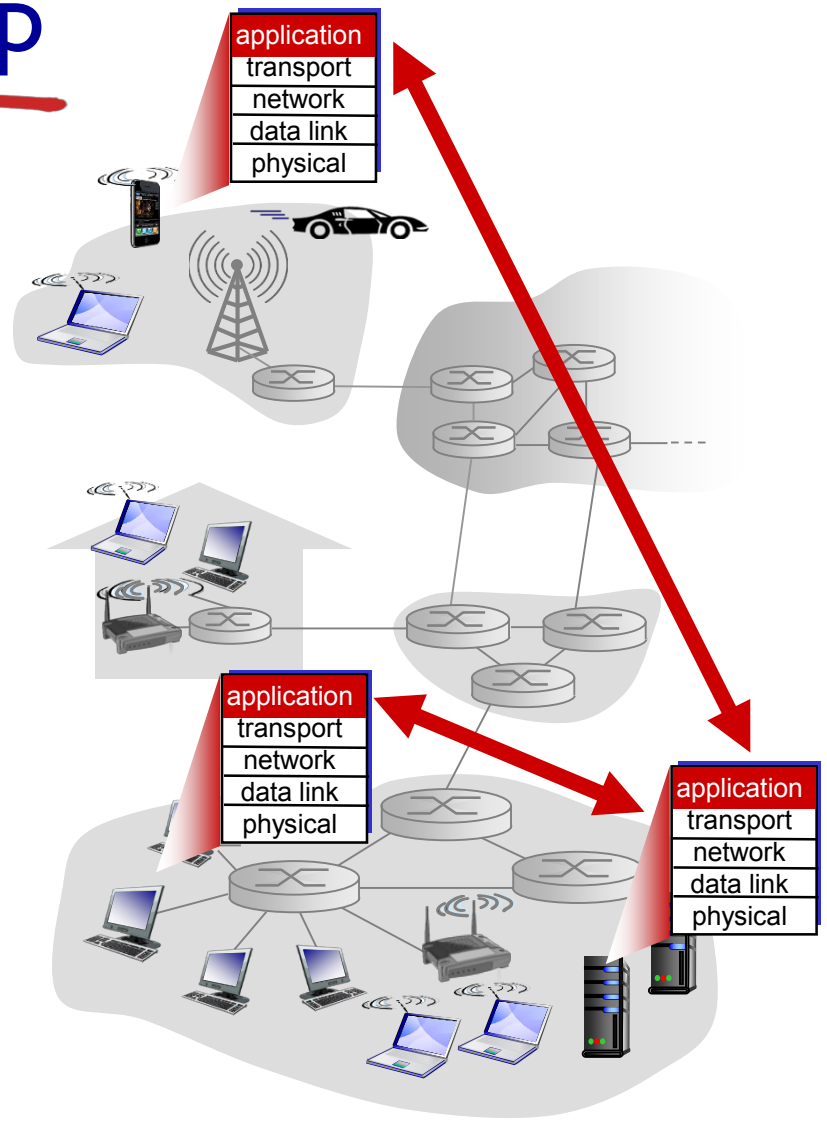
# Creating a network app

write programs that:

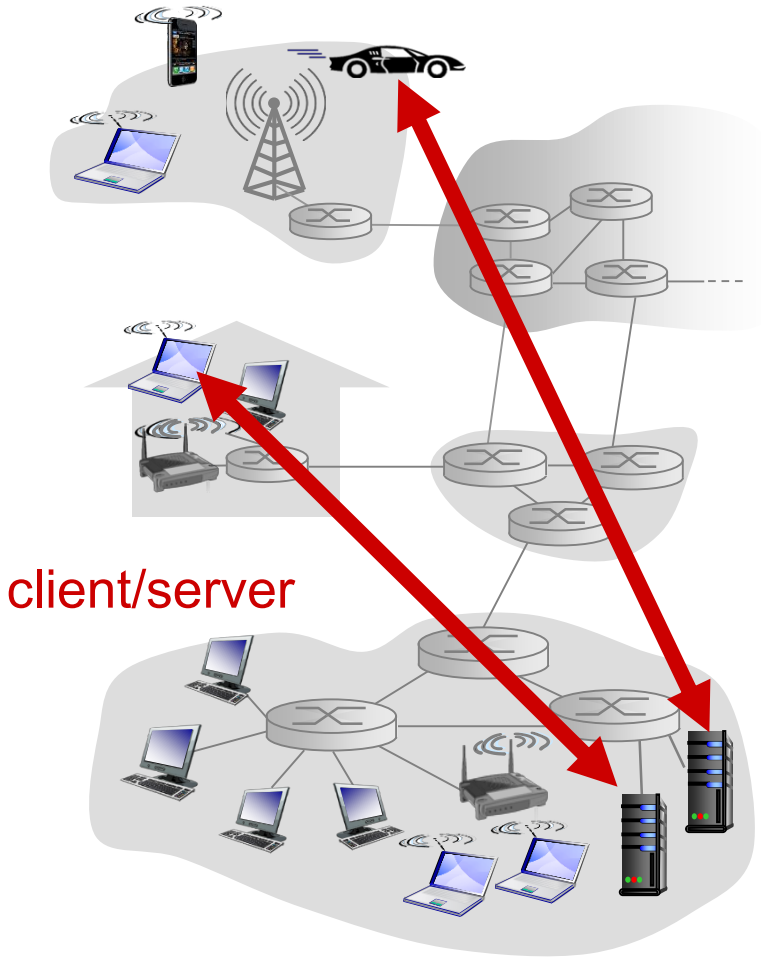
- run on (different) *end systems*
- communicate over network
- e.g., web server software communicates with browser software

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



# Client-server architecture




## server:

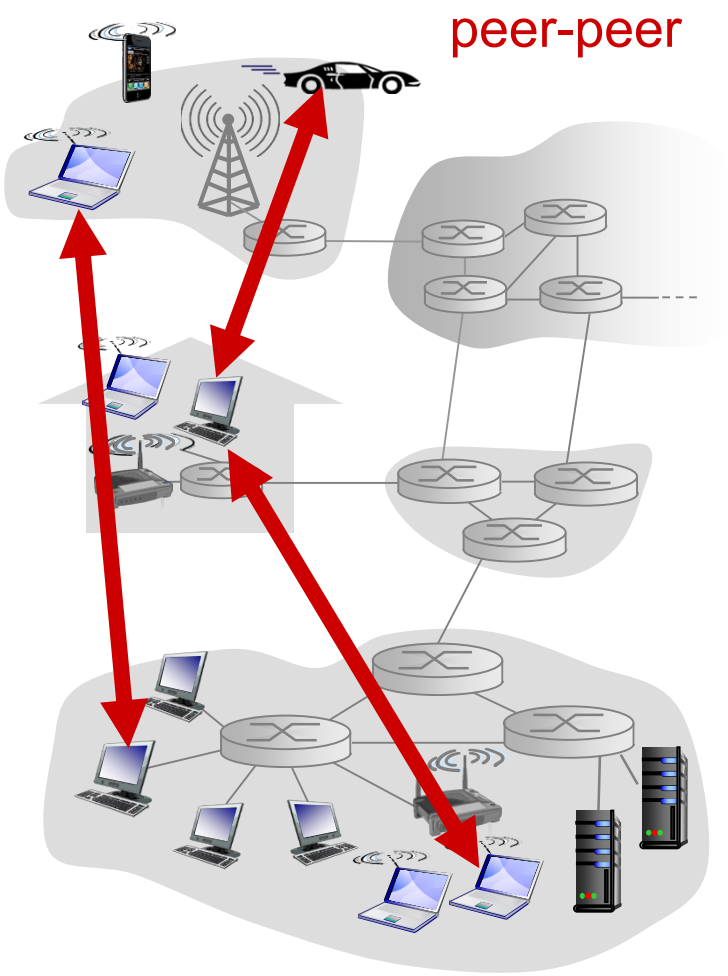
- always-on host
- permanent IP address
- data centers for scaling

## clients:

- communicate with server
- may be intermittently connected
- may have dynamic IP addresses
- do not communicate directly with each other

# P2P architecture

- *no* always-on server
- arbitrary end systems directly communicate
- peers request service from other peers, provide service in return to other peers
  - *self scalability* – new peers bring new service capacity, as well as new service demands
- peers are intermittently connected and change IP addresses
  - complex management



# Processes communicating

*process:* program running within a host

- within same host, two processes communicate using inter-process communication (defined by OS)
- processes in different hosts communicate by exchanging messages

clients, servers

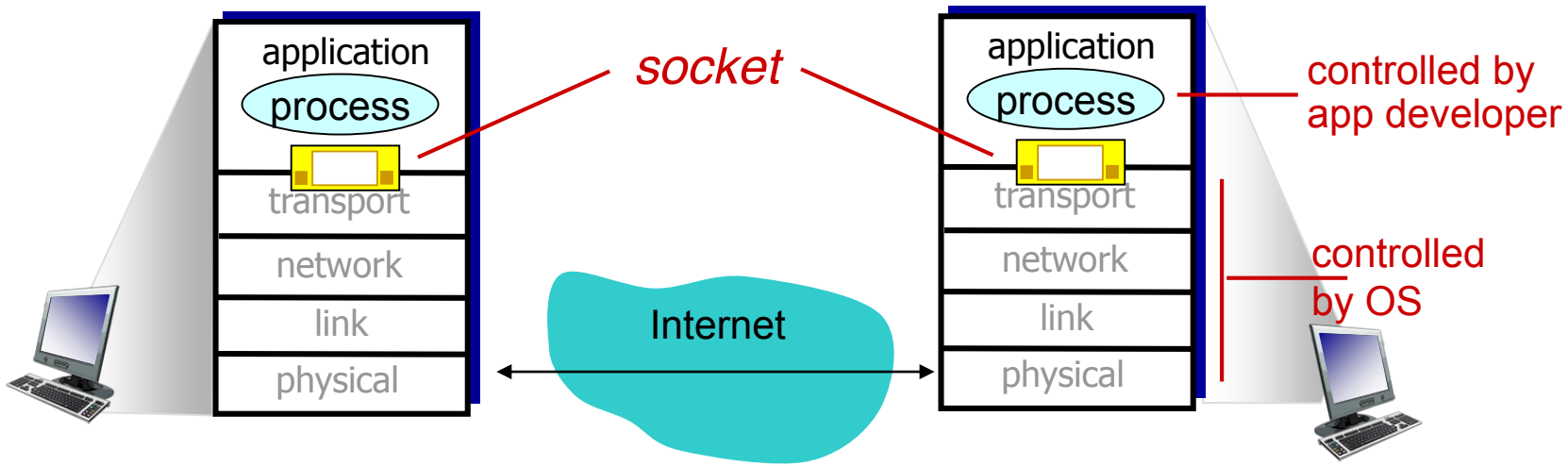
*client process:* process that initiates communication

*server process:* process that waits to be contacted

- aside: applications with P2P architectures have client processes & server processes

# 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 *identifier*
- host device has unique 32-bit IP address
- *Q:* does IP address of host on which process runs suffice for identifying the process?
  - *A:* no, *many* processes can be running on same host

- *identifier* includes both IP address and port numbers associated with process on host.
- example port numbers:
  - HTTP server: 80
  - mail server: 25
- to send HTTP message to gaia.cs.umass.edu web server:
  - IP address: 128.119.245.12
  - port number: 80
- more shortly…

# App-layer protocol defines

- types of messages exchanged,
  - e.g., request, response
- message syntax:
  - what fields in messages & how fields are delineated
- message semantics
  - meaning of information in fields
- rules for when and how processes send & respond to messages

open protocols:

- defined in RFCs
- allows for interoperability
- e.g., HTTP, SMTP

proprietary protocols:

- e.g., Skype

# What transport service does an app need?

## reliable data transfer

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

## timing

- some apps (e.g., Internet telephony, interactive games) require low delay to be "effective"

## throughput

- some apps (e.g., multimedia) require minimum amount of throughput to be "effective"
- other apps ("elastic apps") make use of whatever throughput they get

## security

- encryption, data integrity, …

# Internet transport protocols services

*TCP service:*

- *connection-oriented:* setup required between client and server processes
- *reliable transport* between sending and receiving process
- *flow control:* sender won't overwhelm receiver
- *congestion control:* throttle sender when network overloaded
- *does not provide:* timing, minimum throughput guarantee, security

*UDP service:*

- *unreliable data transfer* between sending and receiving process
- *does not provide:* reliability, flow control, congestion control, timing, throughput guarantee, security, or connection setup,

Q: why bother? Why is there a UDP?

# 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 |

# Chapter 2: outline

2.1 principles of network applications
- app architectures
- app requirements

2.2 Web and HTTP

2.3 FTP

2.4 electronic mail
- SMTP, POP3, IMAP

2.5 DNS

2.6 P2P applications

2.7 socket programming with UDP and TCP

# Web and HTTP

*First, a review…*

- *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,* e.g.,

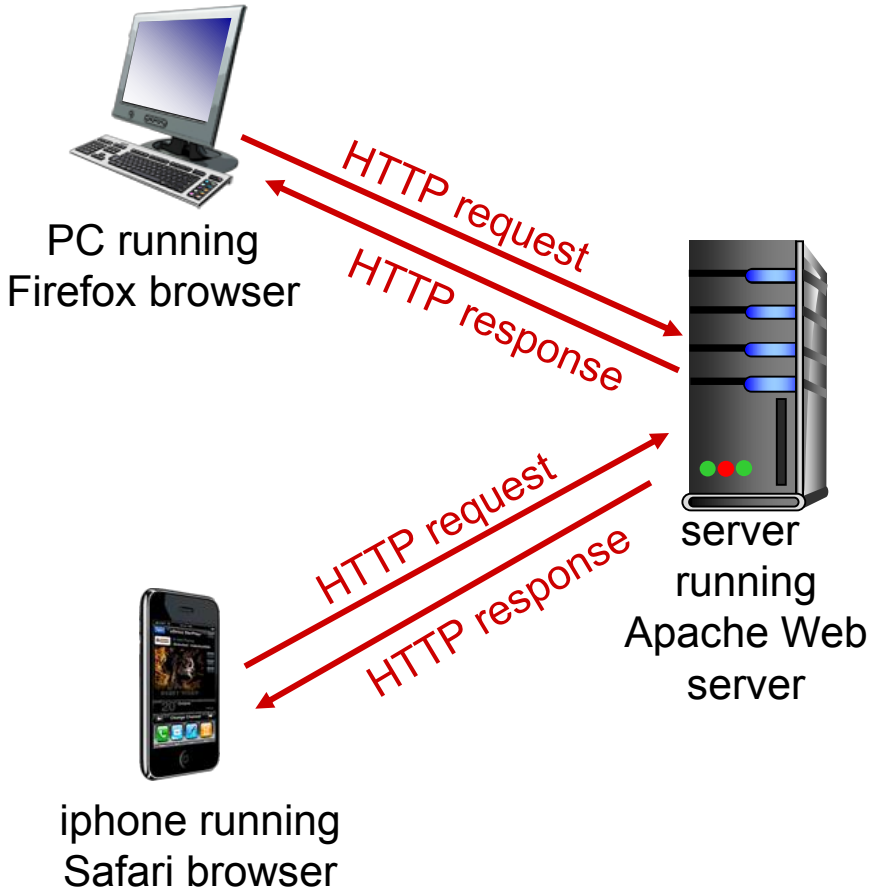
`www.someschool.edu/someDept/pic.gif`

host name | path name

# HTTP overview

## HTTP: hypertext transfer protocol

- Web's application layer protocol
- 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 overview (continued)

*uses TCP:*

- client initiates TCP connection (creates socket) to server, port 80
- server accepts TCP connection from client
- HTTP messages (application-layer protocol messages) exchanged between browser (HTTP client) and Web server (HTTP server)
- TCP connection closed

*HTTP is "stateless"*

- server maintains no information about past client requests

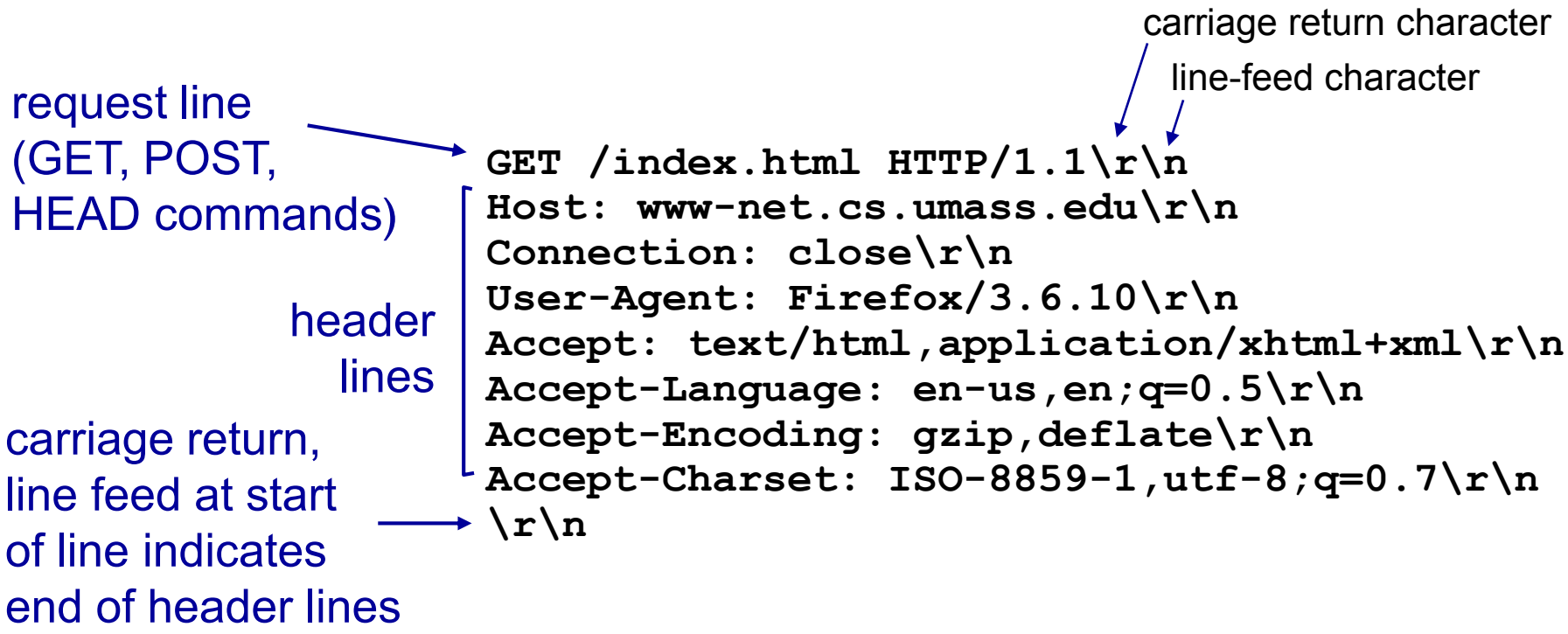
*aside*

protocols that maintain "state" are complex!

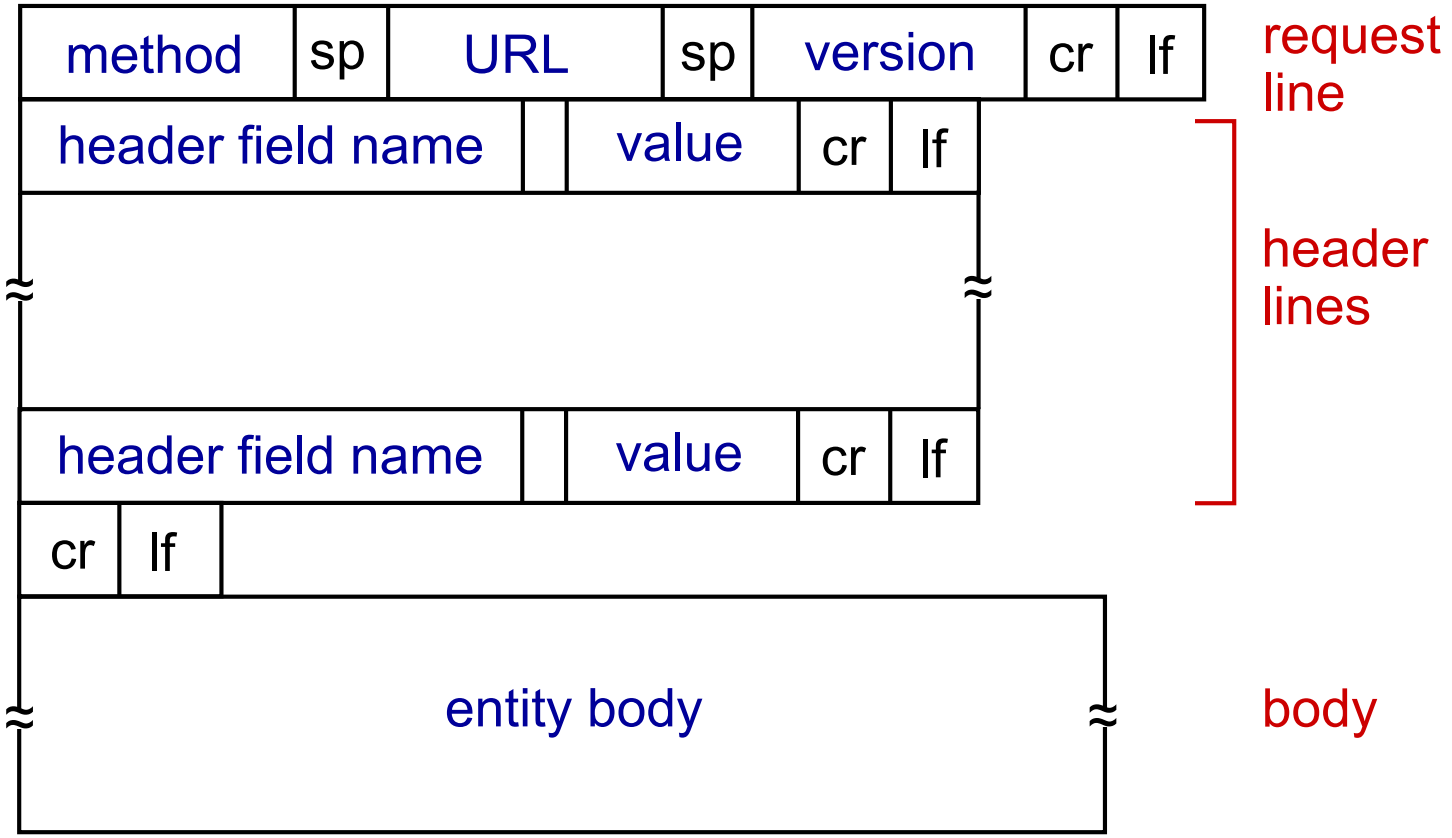
- past history (state) must be maintained
- if server/client crashes, their views of "state" may be inconsistent, must be reconciled

# HTTP request message

- two types of HTTP messages: *request, response*
- HTTP request message:
  - ASCII (human-readable format)



# HTTP request message: general format



# Uploading form 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

# HTTP response message

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 1st line in server-to-client response message.
- some sample codes:

**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**

# Trying out HTTP (client side) for yourself

1. Open a Command Prompt window, type

```
telnet plaza.ufl.edu 80
```

opens TCP connection to port 80 (default HTTP server port) at plaza.ufl.edu
anything typed in sent to port 80 at plaza.ufl.edu

2. Press Ctrl+], then press carriage return

3. Type in a GET HTTP request:

```
GET /hyue/ HTTP/1.1
Host: plaza.ufl.edu
```

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

4. Look at response message sent by HTTP server!