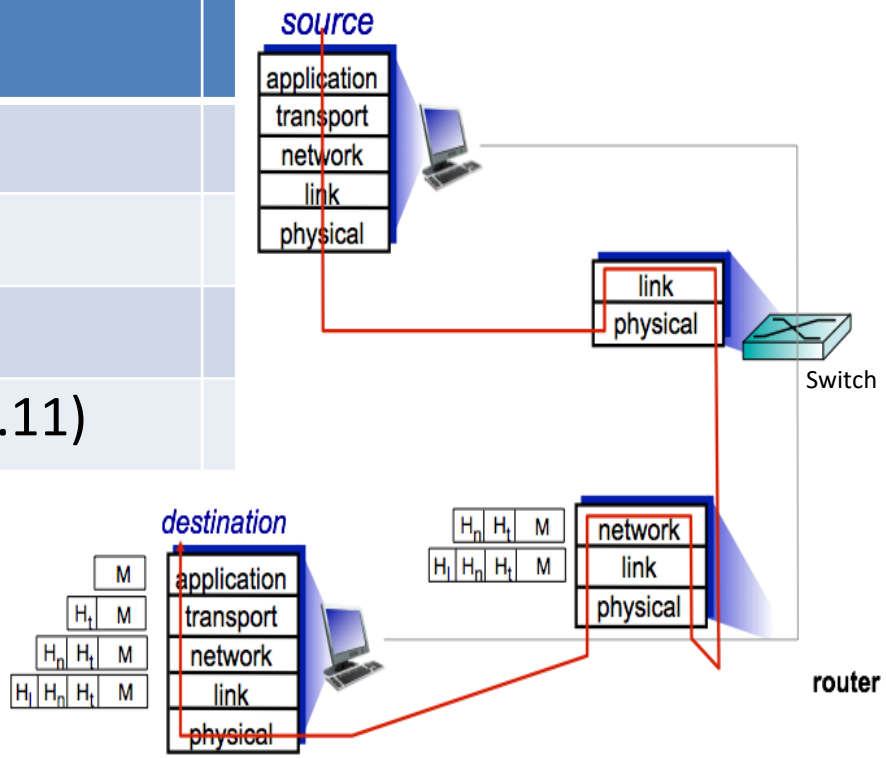
Computer networking (TDDE25): Part 2 …

Niklas Carlsson, Senior Associate Professor
https://www.ida.liu.se/~nikca89/

Network stack with protocol and address examples

Layer	Example Protocols
Application (AL)	HTTP, SMTP, DNS
Transport (TL)	TCP, UDP
Network (NL)	IPv4, IPv6
Link (LL)	Ethernet, WiFi (802.11)



Roadmap: Application layer

- Principles of Network Applications
  - Application Architectures
  - Application Requirements
- Web and HTTP
- FTP
- Electronic Mail
  - SMTP, POP3, IMAP
- DNS
- P2P Applications
- Socket Programming with UDP and TCP

Some Network 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
- …
- …

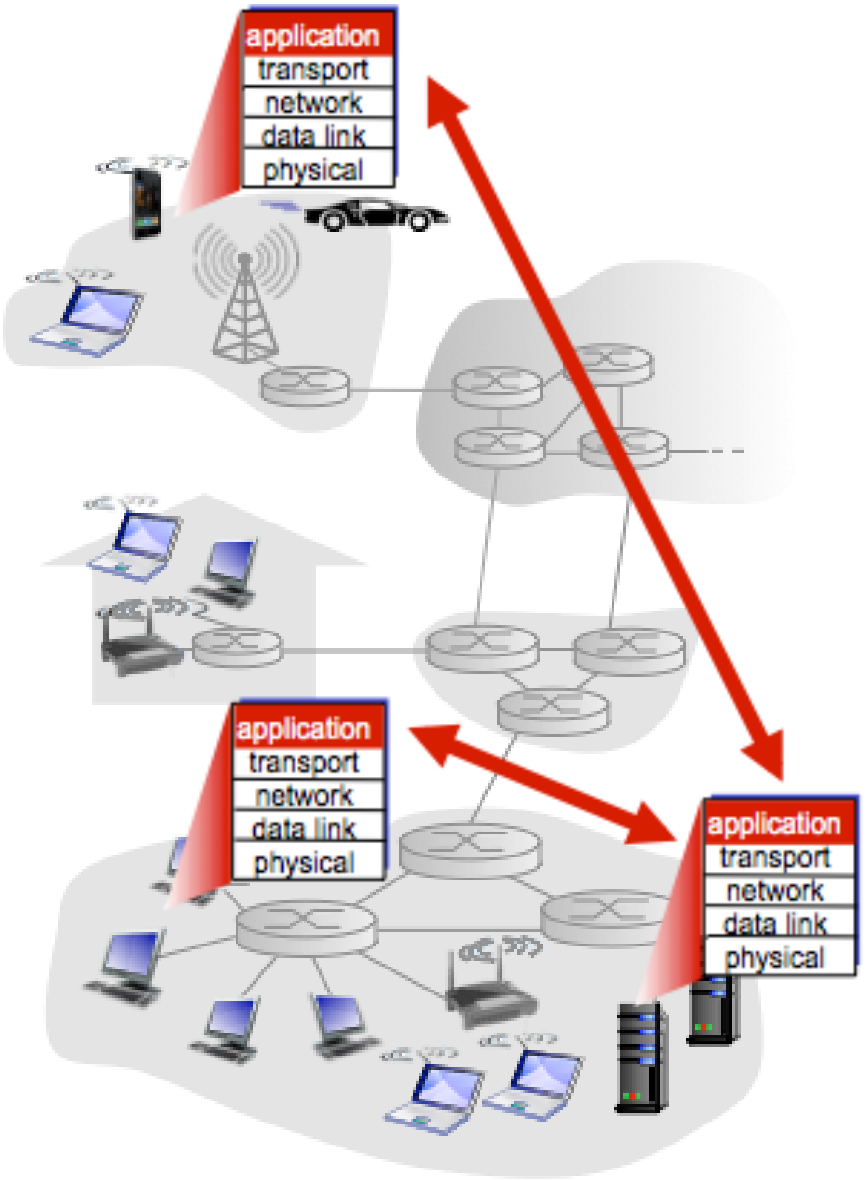
Creating a Network App

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

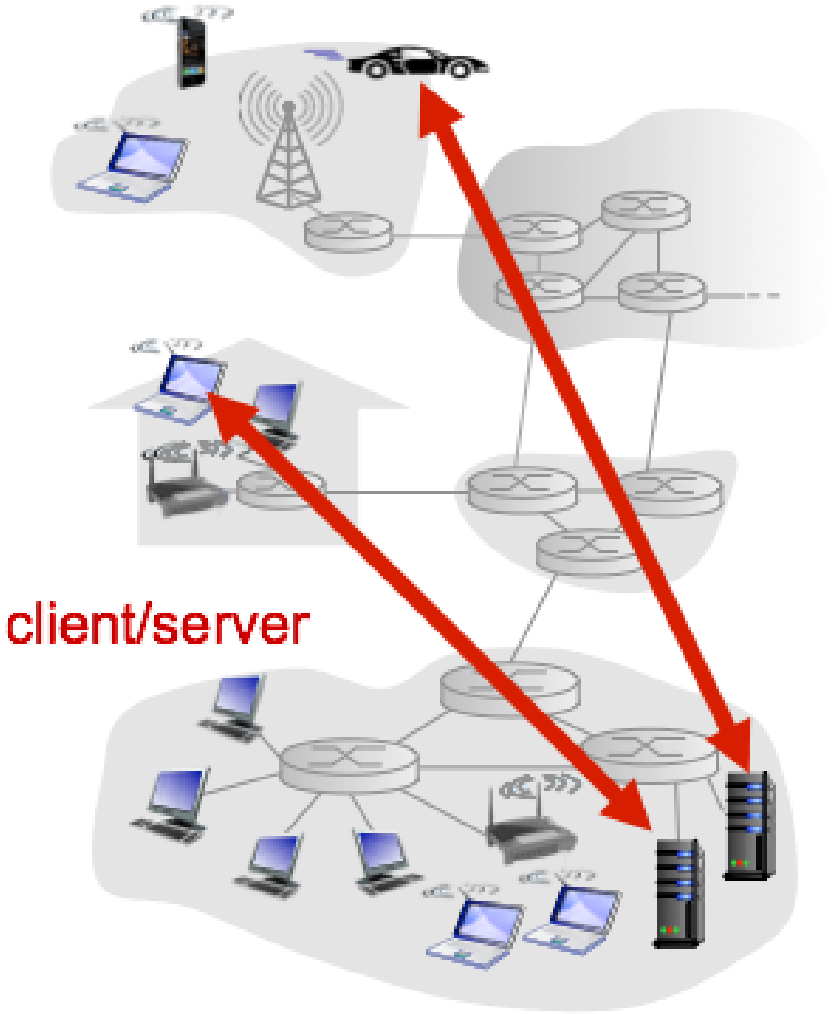


Application Architectures

possible structure of applications:

- client-server
- peer-to-peer (P2P)

Client-Server Architecture



server:

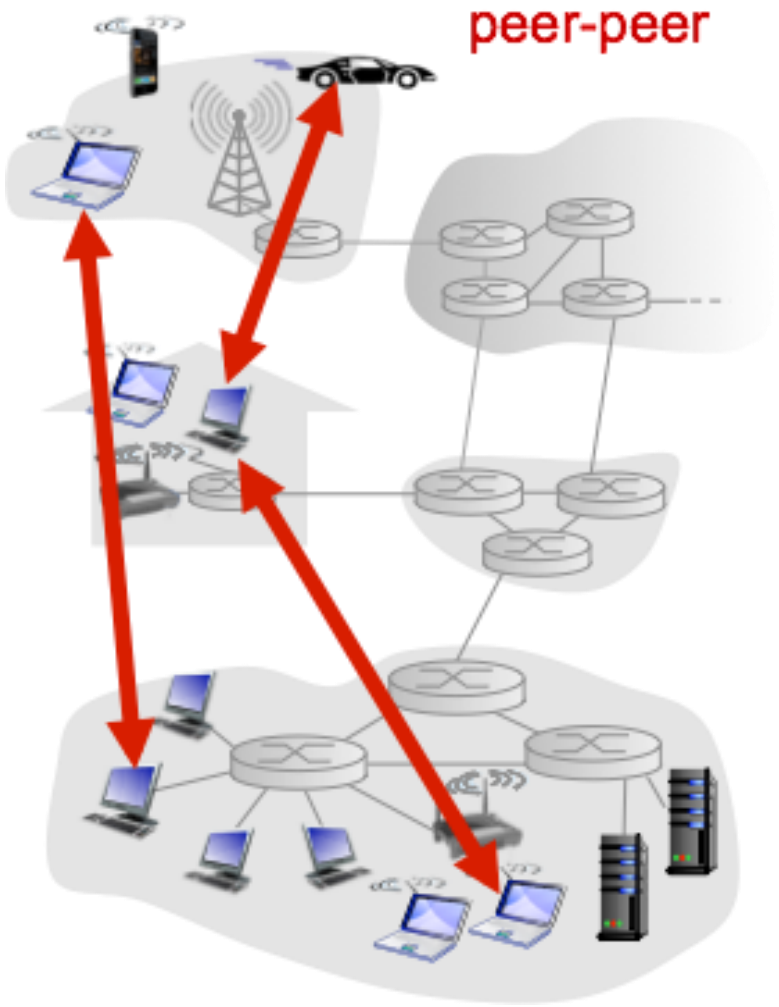
- always-on host
- permanent IP address
- data centers for scaling

clients:

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

P2P Architetcure

- *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.

- processes in different hosts communicate by exchanging messages

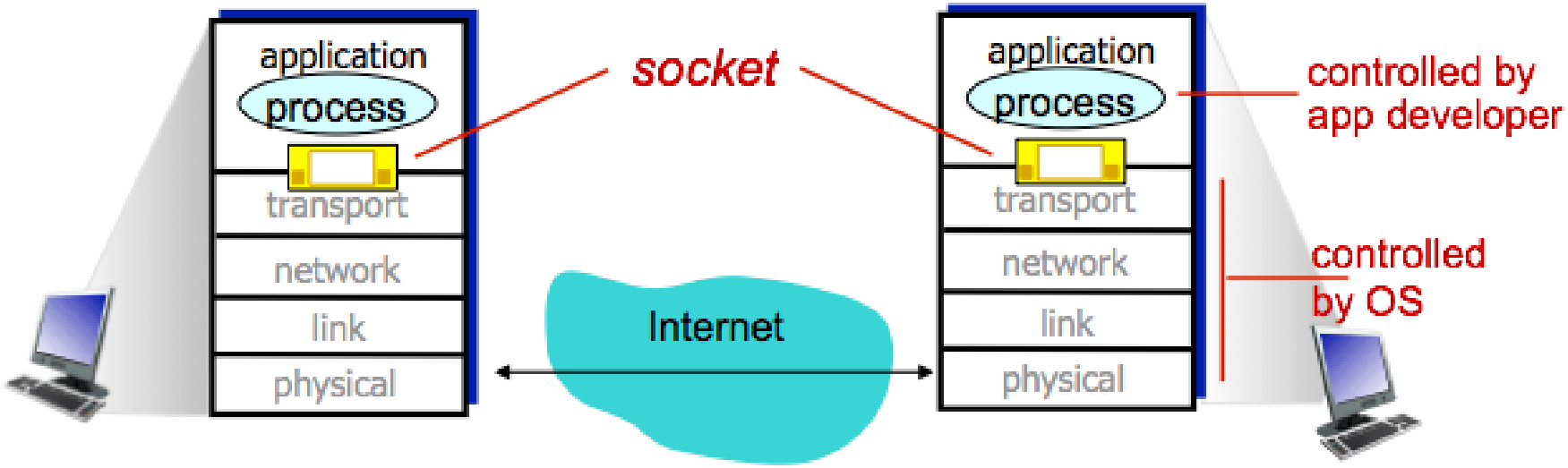
Client-server paradigm

client process: process that initiates communication

server process: process that waits to be contacted

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 Process

- 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?

data integrity

- 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, …

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	
interactive games	loss-tolerant	few kbps up	yes, few secs
text messaging	no loss	elastic	Yes (and no), 100's msec

Internet Transport Protocol Services

Transport Control Protocol

TCP service:

- *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
- *connection-oriented:* setup required between client and server processes

User Datagram Protocol

UDP service:

- *unreliable data transfer* between sending and receiving process
- *does not provide:* reliability, flow control, congestion control, timing, throughput guarantee, security, orconnection 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]	
remote terminal access	Telnet [RFC 854]	
Web	HTTP [RFC 2616]	
file transfer	FTP [RFC 959]	
streaming multimedia	proprietary (e.g., RealNetworks, youtube, netflix, spotify)	
Internet telephony	proprietary (e.g., Dialpad, skype)	

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	proprietary (e.g., RealNetworks, youtube, netflix, spotify)	TCP (or UDP)
Internet telephony	proprietary (e.g., Dialpad, skype)	UDP or TCP typically UDP

Securing TCP

TCP & UDP

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

SSL/TLS

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

SSL is at app layer

- Apps use SSL libraries, which "talk" to TCP

SSL socket API

- cleartext passwds sent into socket traverse Internet encrypted

Roadmap

- Principles of Network Applications
  - App Architectures
  - App Requirements
- Web and HTTP
- FTP
- Electronic Mail
  - SMTP, POP3, IMAP
- DNS
- P2P Applications
- Socket Programming with UDP and TCP

The Web and HTTP

HTML: Hypertext Markup Language

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

URL: Uniform Resource Locator

Hypertext Links & URLs

Linking to other URL's:

<A HREF="http://www.ida.liu.se/fred/resumepage.html">my resume</A>

Acquiring Images:

<IMG src="http://www.ninthwonder.com/~miko/counter.gif?name=idocsguide" ALT="counter">

Executing Applets:

```
<APPLET
CODE="http://www.ida.liu.se/tutorial/MyApplet.class"
WIDTH=200 HEIGHT=50>
    <PARAM NAME=TEXT VALUE="Hi There">
    <P>Hi There!<P>
</APPLET>
```

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

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

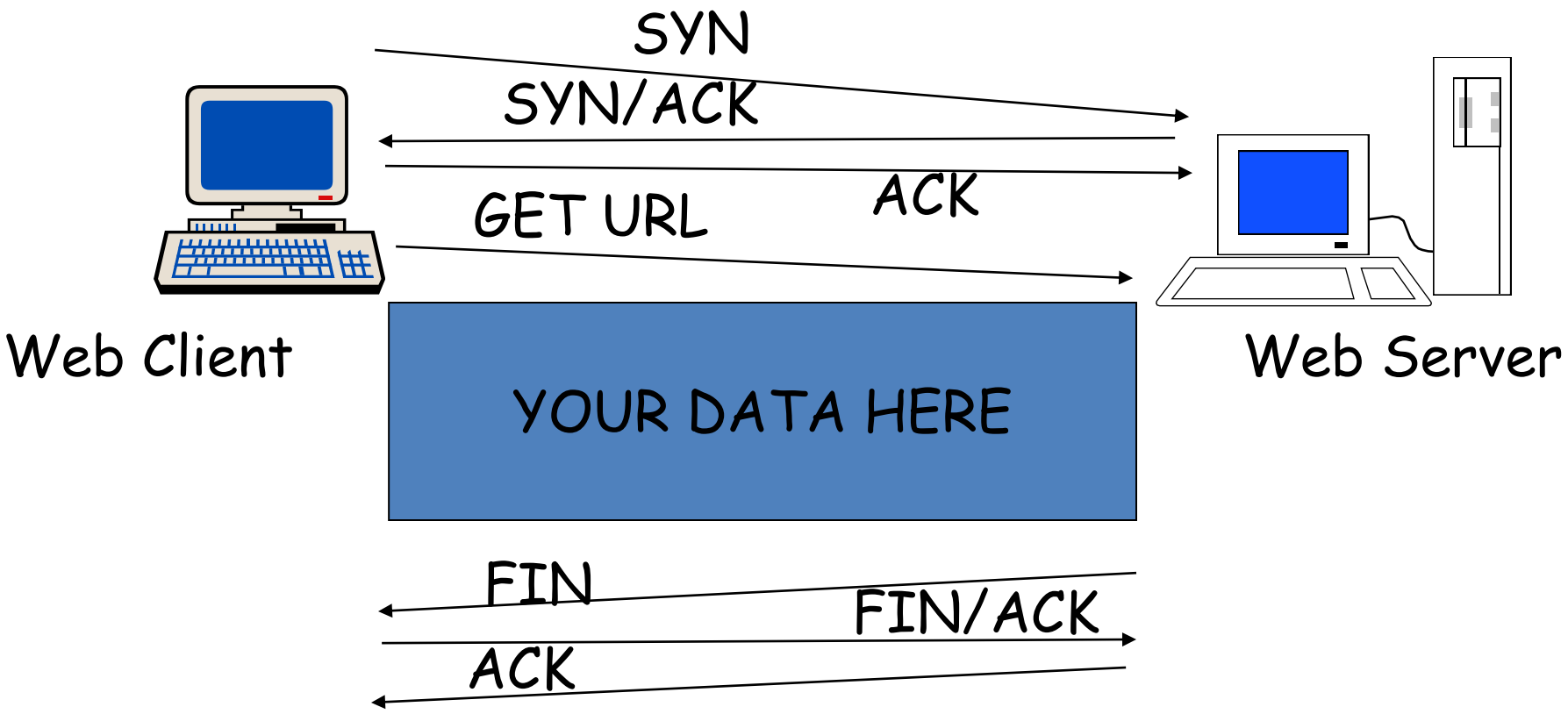
aside

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

Network View: HTTP and TCP

- TCP is a connection-oriented protocol



HTTP Connections

non-persistent HTTP

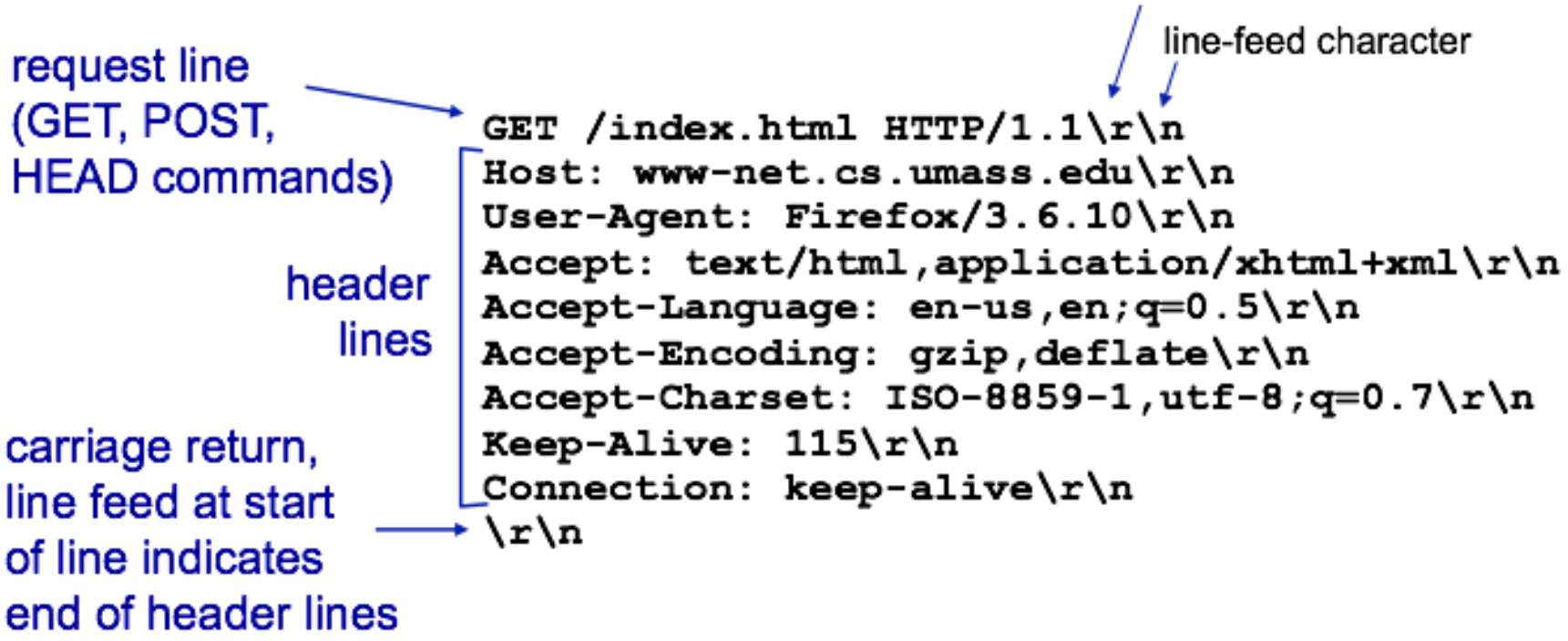
- at most one object sent over TCP connection
  - connection then closed
- downloading multiple objects required multiple connections

persistent HTTP

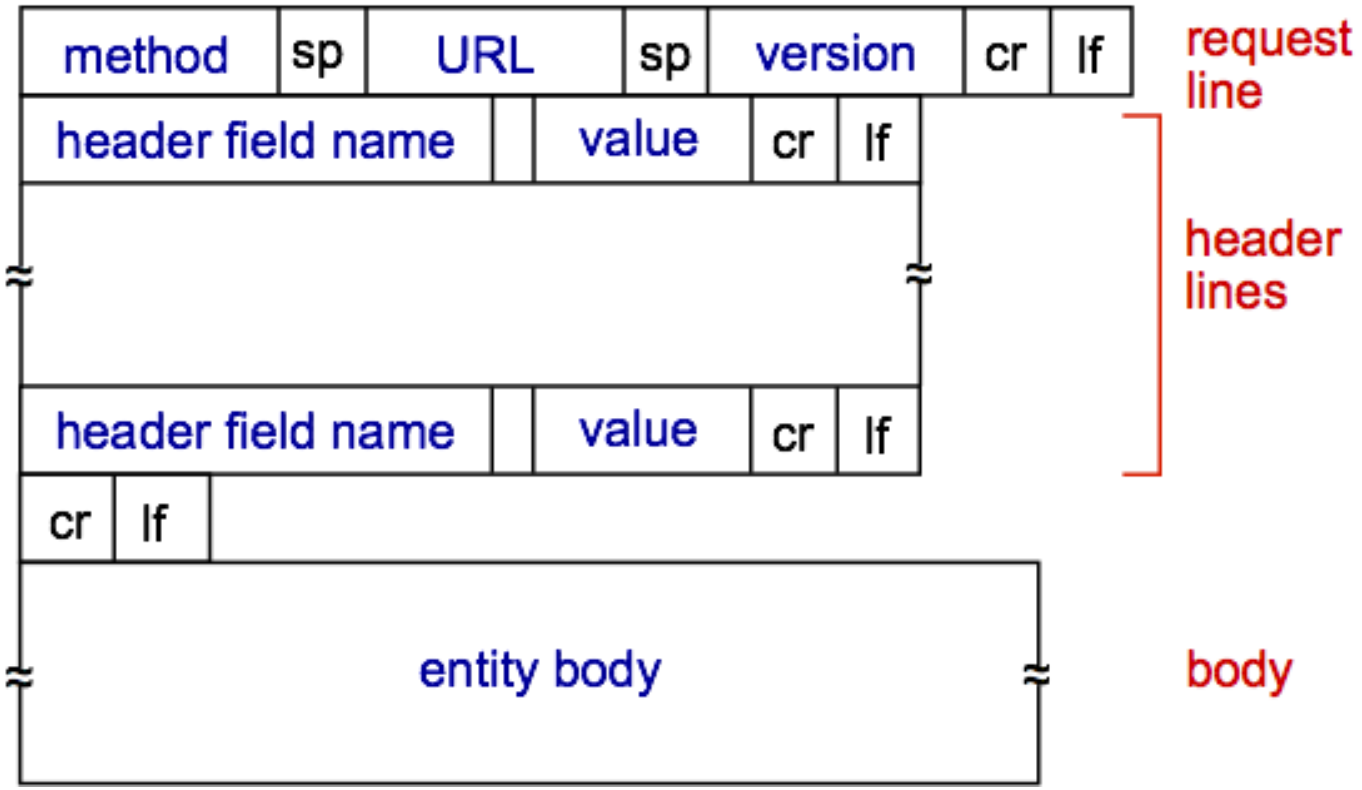
- multiple objects can be sent over single TCP connection between client, server

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

June 1997

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/2

2015: RFC published + supported by major browsers

Similar basic features as HTTP 1.1, but also performance related enhancements, including (but not limited to):

- Server push
- Multiplexing (to avoid head-of-line blocking)
- Header compression

Trying out HTTP (client side) for yourself

1. Telnet to your favorite Web server:

```
telnet www.eurecom.fr 80
```

Opens TCP connection to port 80 (default HTTP server port) at www.eurecom.fr. Anything typed in sent to port 80 at www.eurecom.fr

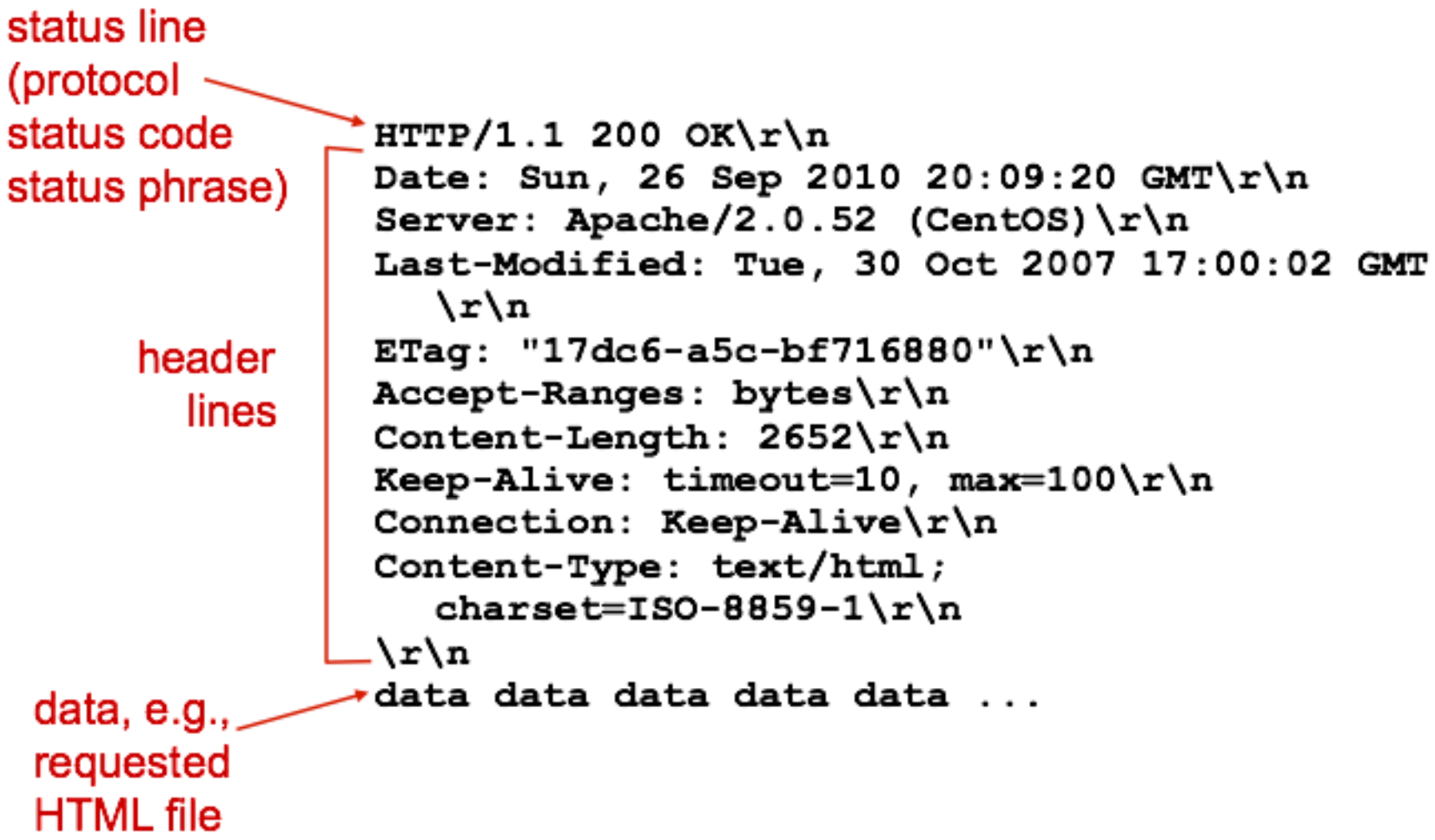
2. Type in a GET HTTP request:

```
GET /~ross/index.html HTTP/1.0
```

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

3. Look at response message sent by HTTP server!

HTTP Response Message



HTTP Response Status Codes

- 1XX: Informational (def'd in 1.0, used in 1.1)

  100 Continue, 101 Switching Protocols
- 2XX: Success

  200 OK, 206 Partial Content
- 3XX: Redirection

  301 Moved Permanently, 304 Not Modified
- 4XX: Client error

  400 Bad Request, 403 Forbidden, 404 Not Found
- 5XX: Server error

  500 Internal Server Error, 503 Service Unavailable, 505 HTTP Version Not Supported

HTTP Response Status Codes

- 1XX: Informational (def'd in 1.0, used in 1.1)

  100 Continue, 101 Switching Protocols

- 2XX: Success

  200 OK, 206 Partial Content

- 3XX: Redirection

  301 Moved Permanently, 304 Not Modified

- 4XX: Client error

  400 Bad Request, 403 Forbidden, 404 Not Found

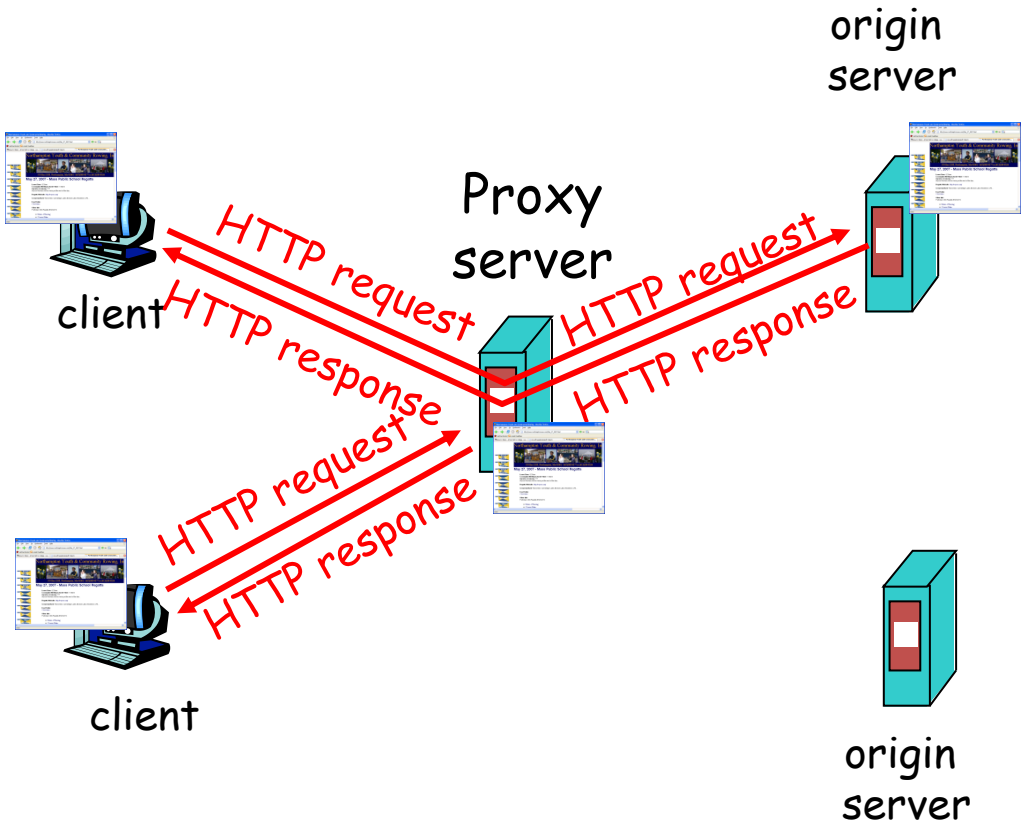
- 5XX: Server error

  500 Internal Server Error, 503 Service Unavailable, 505 HTTP Version Not Supported

Web caches (proxy server)

Goal: satisfy client request without involving origin server

- user sets browser: Web accesses via cache
- browser sends all HTTP requests to cache
  - object in cache: cache returns object
  - else cache requests object from origin server, then returns object to client



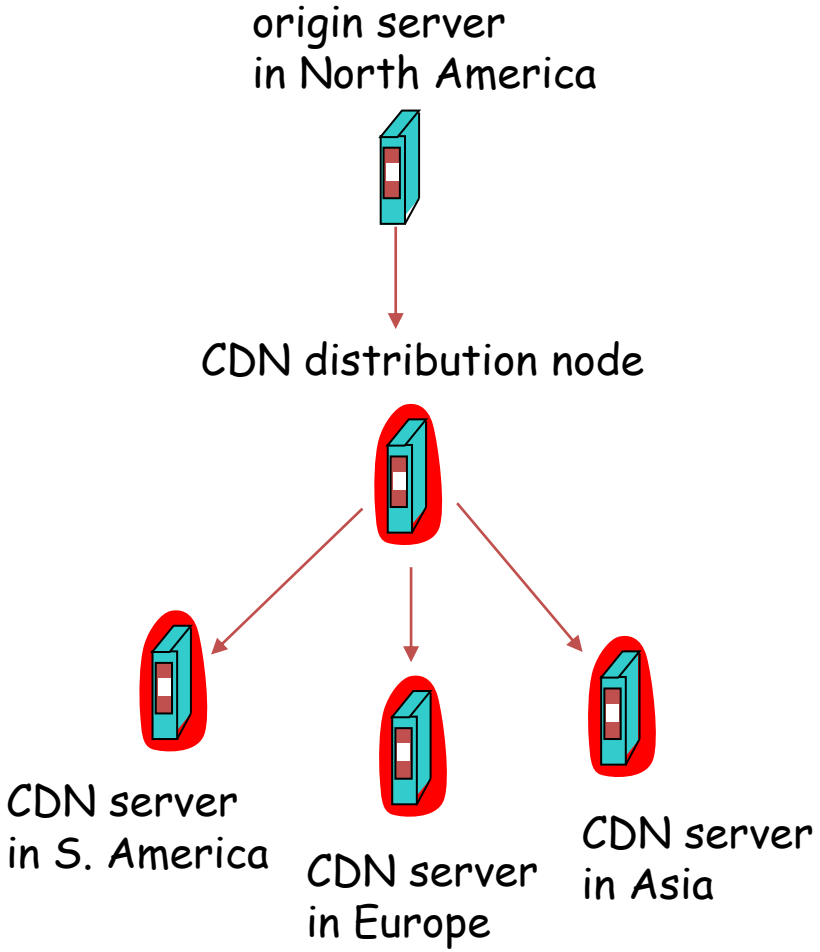
Content distribution networks (CDNs)

- The content providers are the CDN customers.

Content replication

- CDN company installs hundreds of CDN servers throughout Internet
  - in lower-tier ISPs, close to users
- CDN replicates its customers' content in CDN servers. When provider updates content, CDN updates servers

Different approaches …



Cookies: keeping “state”

Many major Web sites use cookies

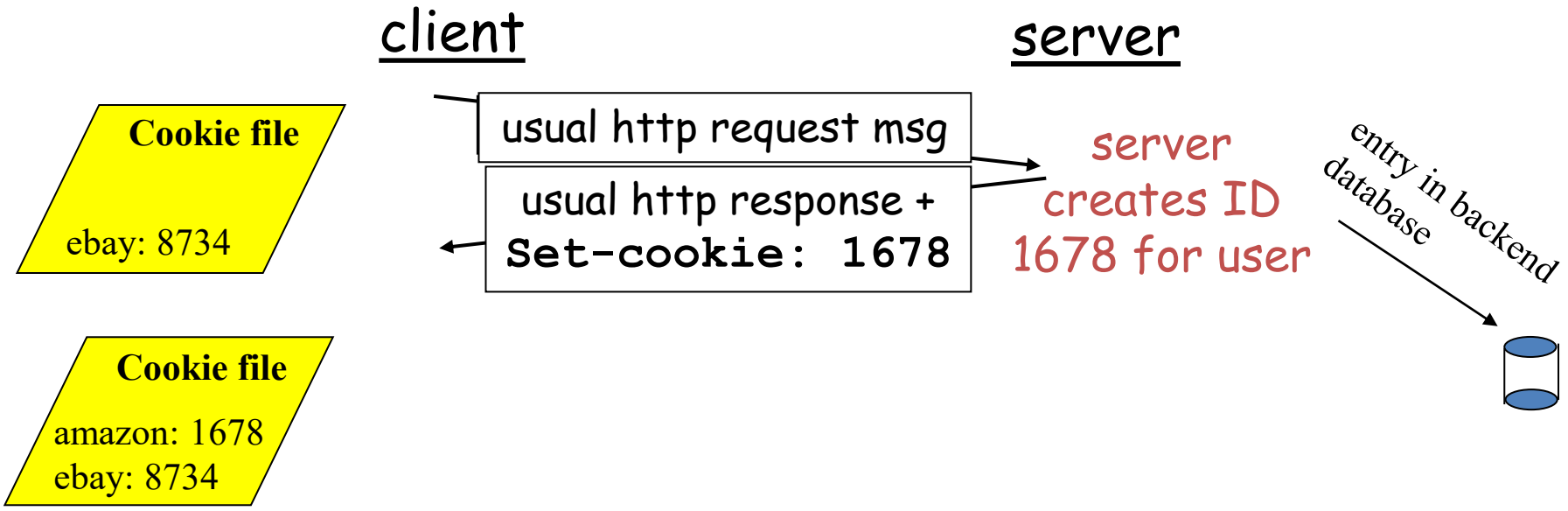
Four components:

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

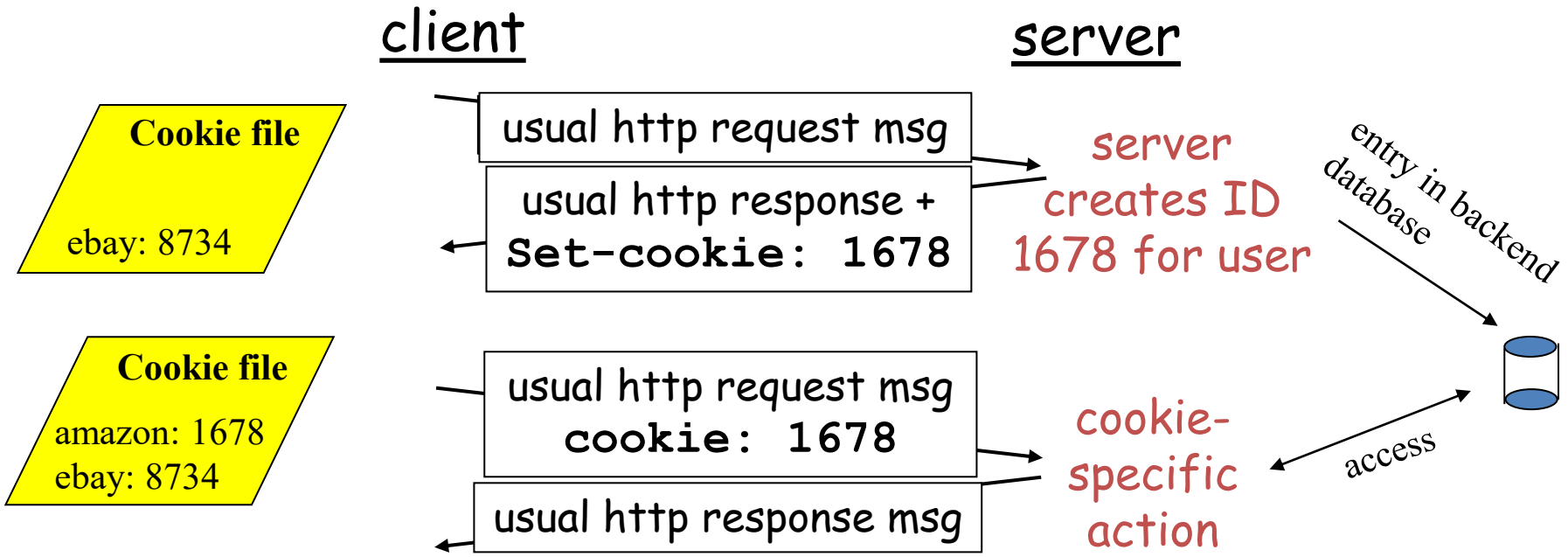
Example:

– User visits a specific e-commerce site …

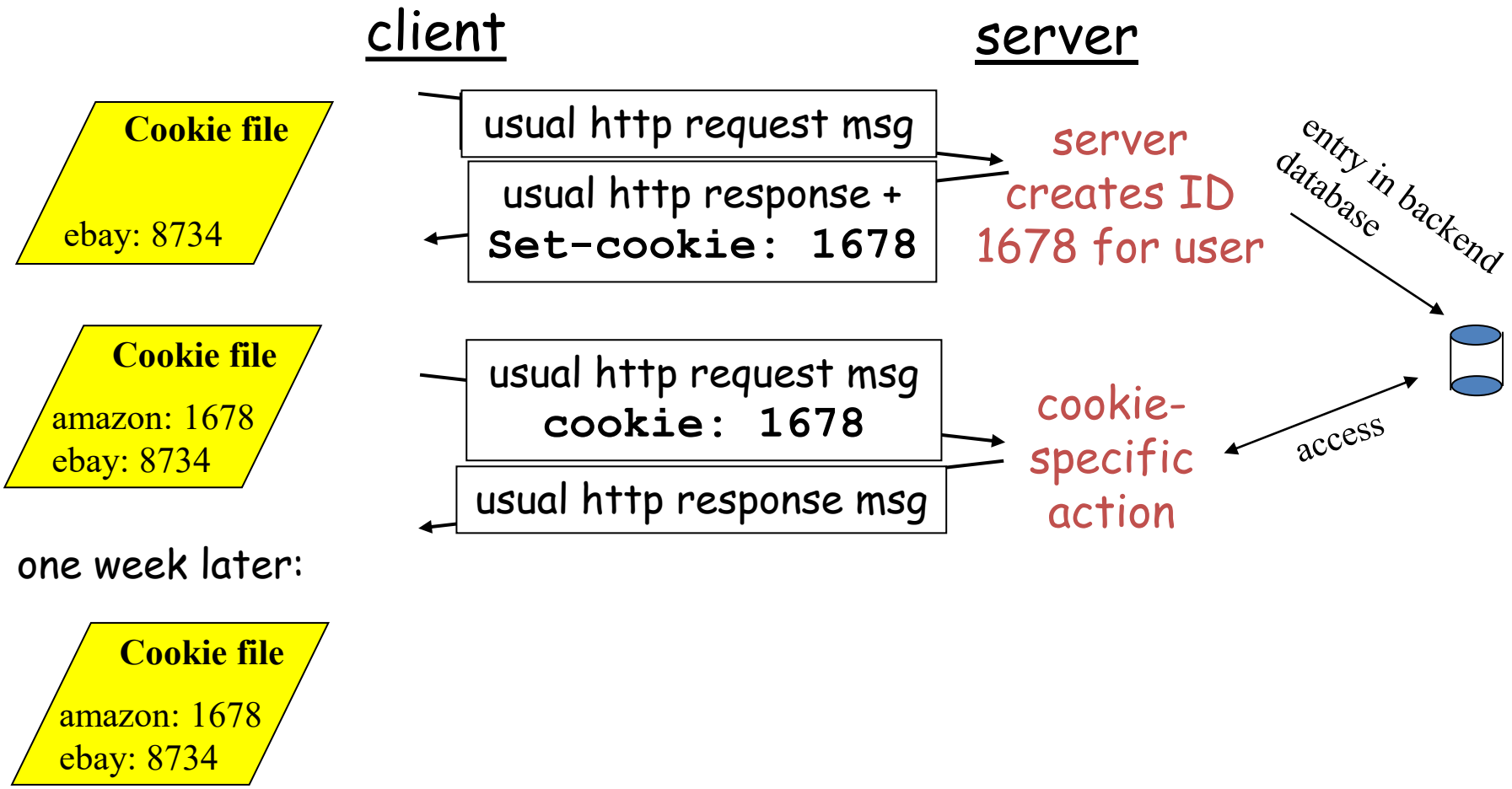
Cookies: keeping “state” (cont.)



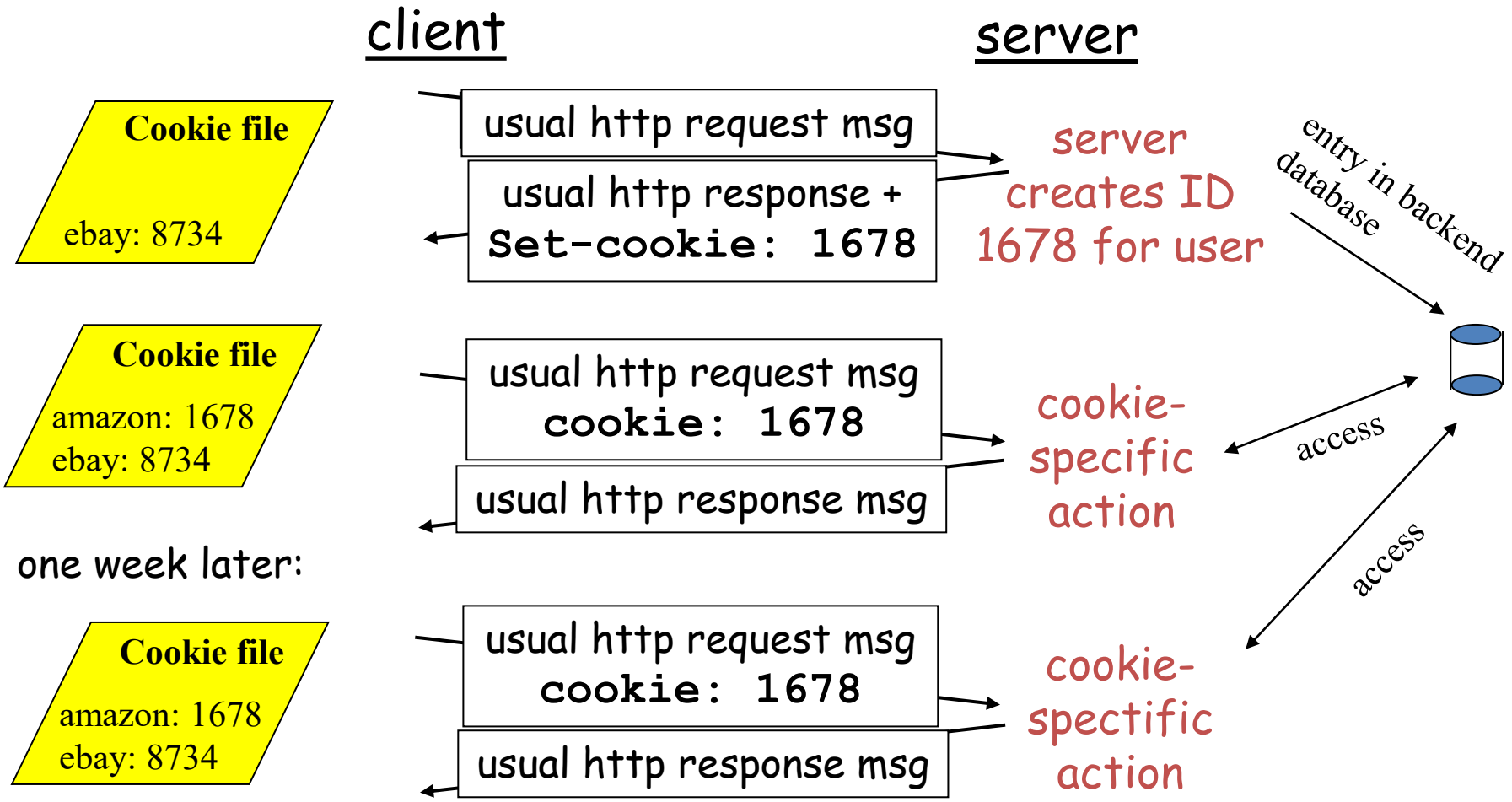
Cookies: keeping "state" (cont.)



Cookies: keeping "state" (cont.)



Cookies: keeping "state" (cont.)



Cookies (continued)

What cookies can bring:

- authorization
- shopping carts
- recommendations
- user session state (Web e-mail)

how to keep "state":

- protocol endpoints: maintain state at sender/receiver over multiple transactions
- cookies: http messages carry state

aside

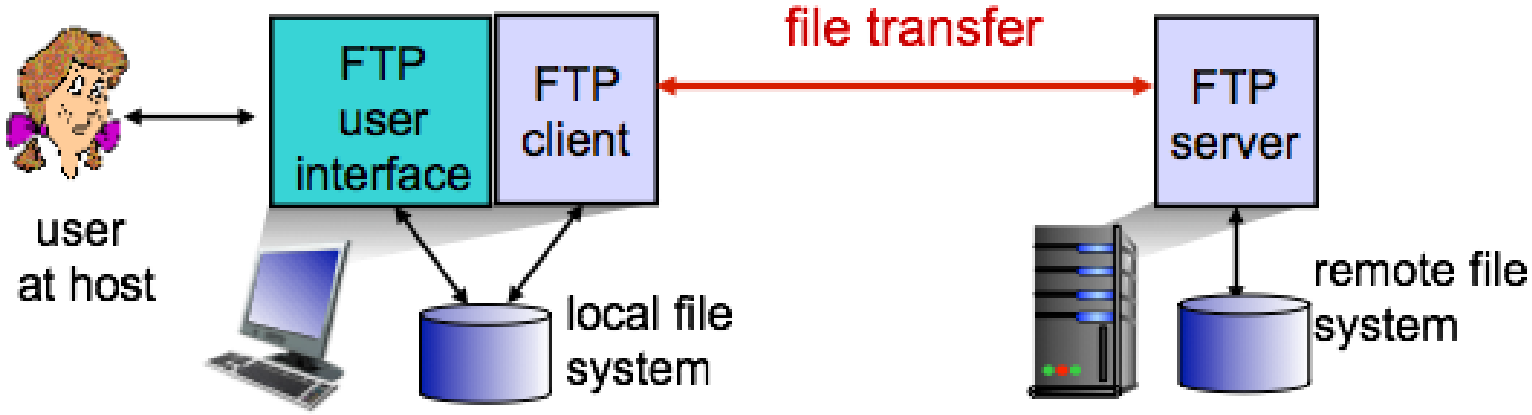
Cookies and privacy:

- cookies permit sites to learn a lot about you
- you may supply name and e-mail to sites
- search engines use redirection & cookies to learn yet more
- advertising companies obtain info across sites

Roadmap

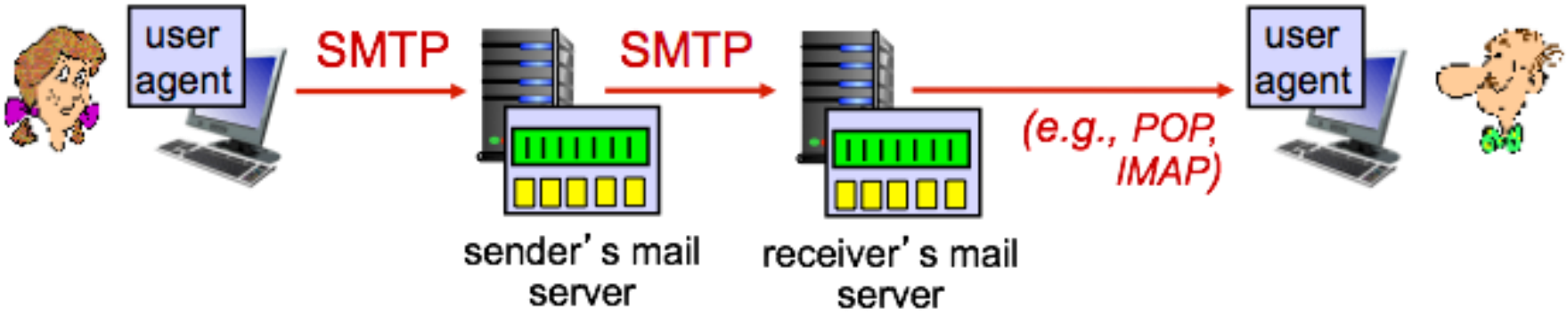
- Principles of Network Applications
  - App Architectures
  - App Requirements
- Web and HTTP
- FTP
- Electronic Mail
  - SMTP, POP3, IMAP
- DNS
- P2P Applications
- Socket Programming with UDP and TCP

FTP: File Transfer Protocol



- transfer file to/from remote host
- client/server model
  - *client:* side that initiates transfer (either to/from remote)
  - *server:* remote host
- ftp: RFC 959
- ftp server: port 21

Mail Access Protocols



- SMTP: delivery/storage to receiver's server
- mail access protocol: retrieval from server

Roadmap

- Principles of Network Applications
  - App Architectures
  - App Requirements
- Web and HTTP
- FTP
- Electronic Mail
  - SMTP, POP3, IMAP
- DNS
- P2P Applications
- Socket Programming with UDP and TCP

DNS: Domain Name System

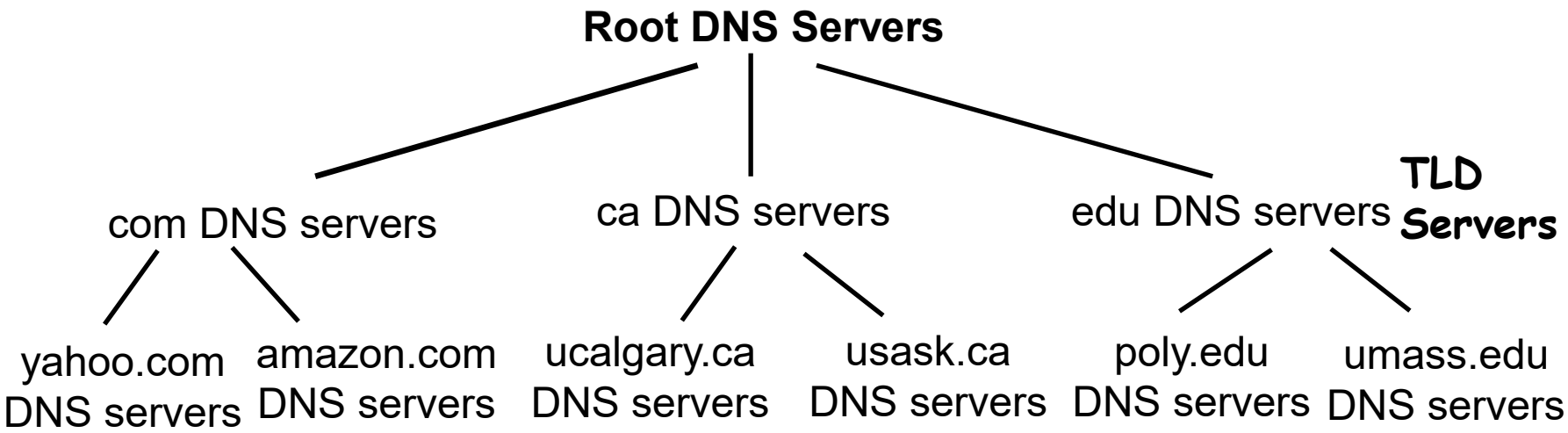
Internet hosts:

- IP address (32 bit) - used for addressing datagrams
- "name", e.g., www.yahoo.com - used by humans

DNS: provides translation between host name and IP address

- *distributed database* implemented in hierarchy of many *name servers*
- *distributed for scalability & reliability*

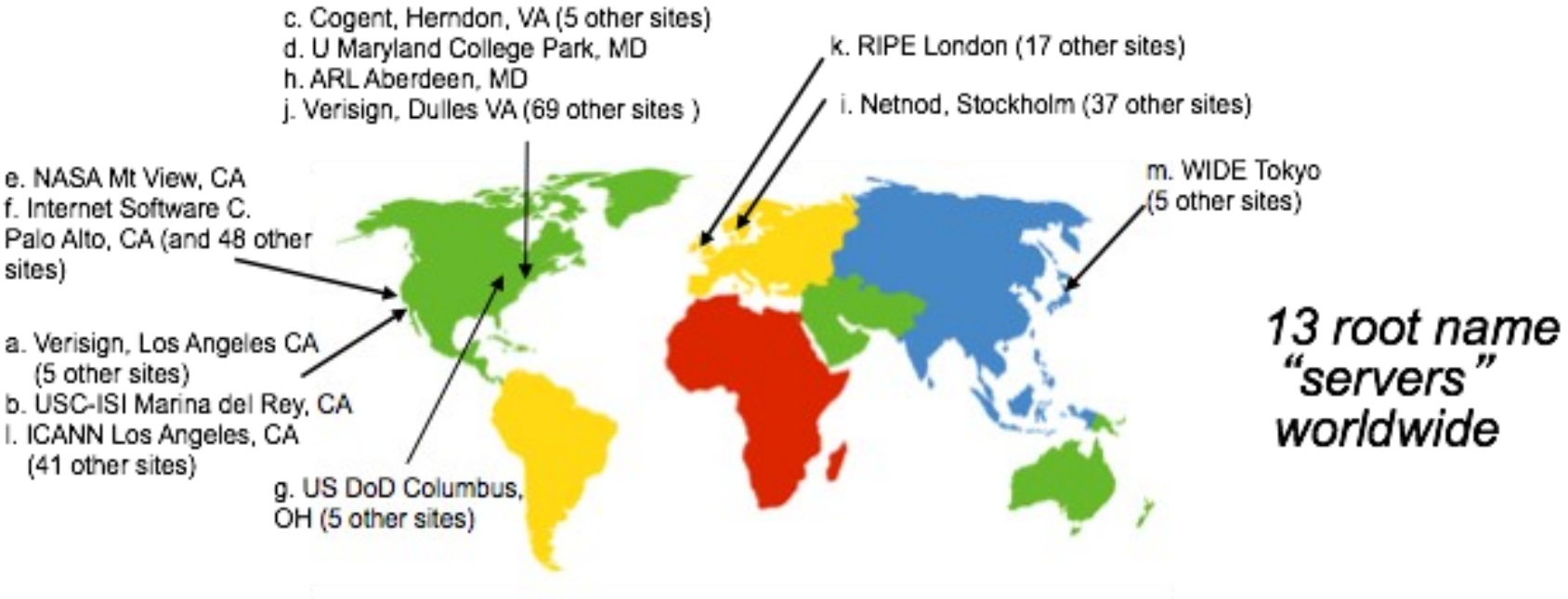
Distributed, Hierarchical Database



- Root servers and TLD servers typically do not contain hostname to IP mappings; they contain mappings for locating authoritative servers.

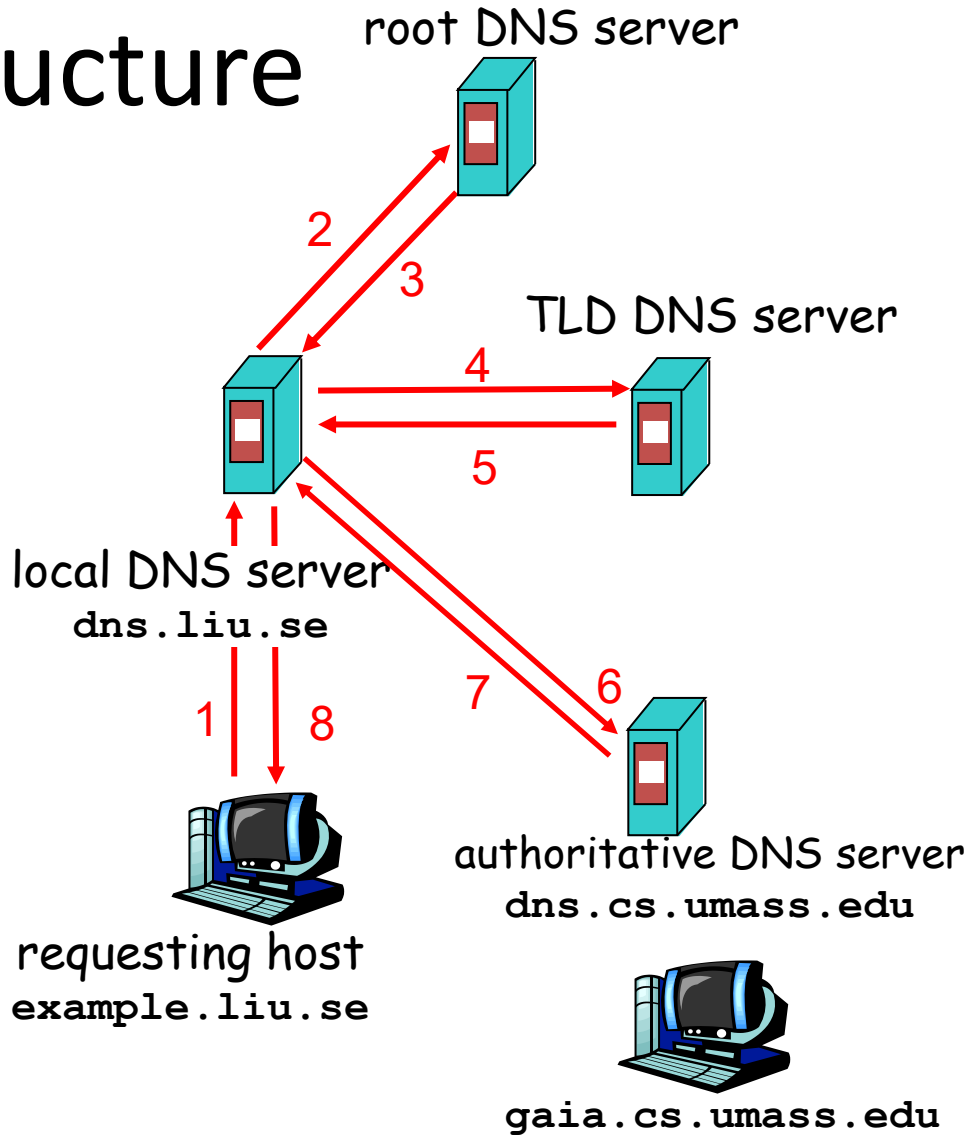
DNS: Root Name Servers

- contacted by local name server that can not resolve name
- root name server:
  - contacts authoritative name server if name mapping not known
  - gets mapping
  - returns mapping to local name server



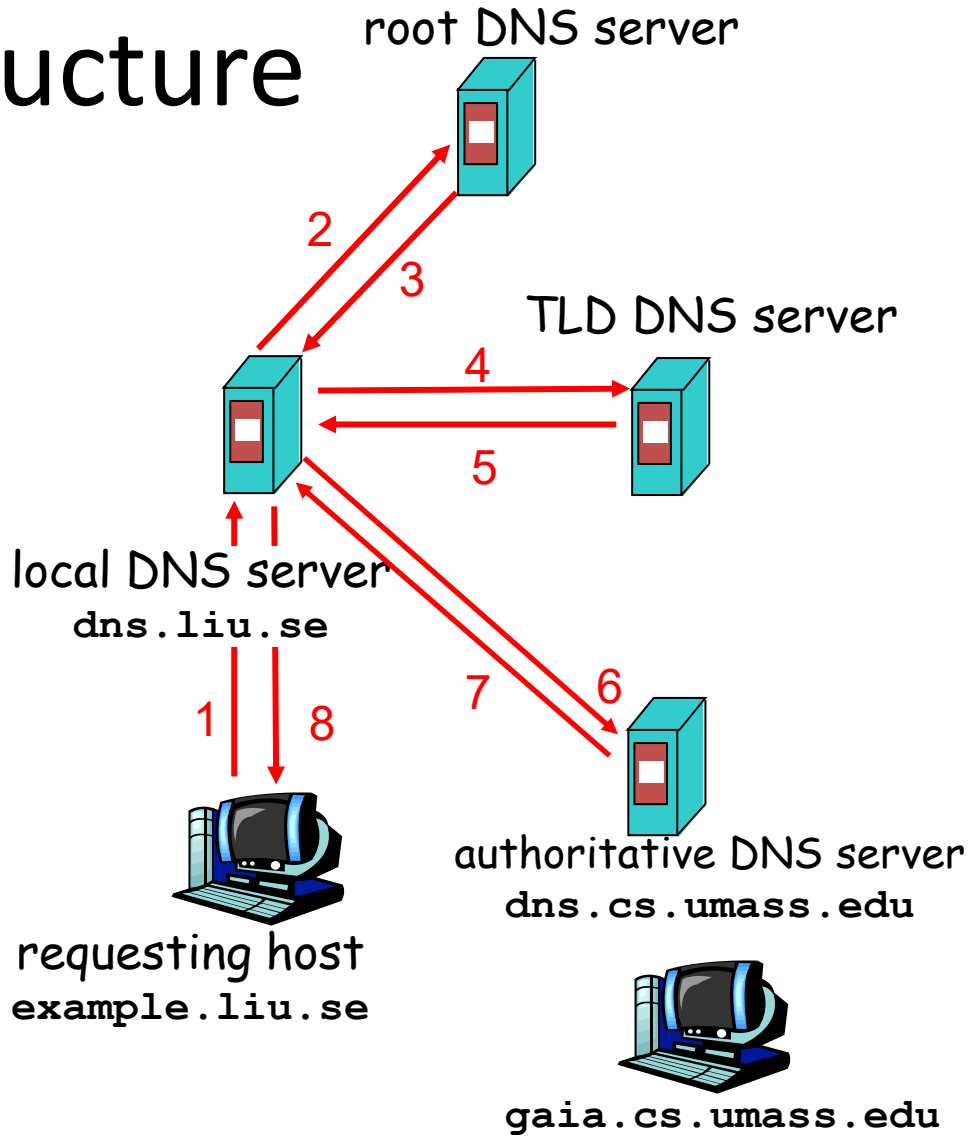
DNS Infrastructure

- Host at liu.se wants IP address for gaia.cs.umass.edu
- Infrastructure:
  - Client resolver
  - Local DNS server
  - Authoritative DNS Server
  - Root DNS Server
  - Top-Level Domain DNS Server
- Transport protocol?



DNS Infrastructure

- Host at liu.se wants IP address for gaia.cs.umass.edu
- Infrastructure:
  - Client resolver
  - Local DNS server
  - Authoritative DNS Server
  - Root DNS Server
  - Top-Level Domain DNS Server
- Transport protocol?
  - UDP (port: 53)



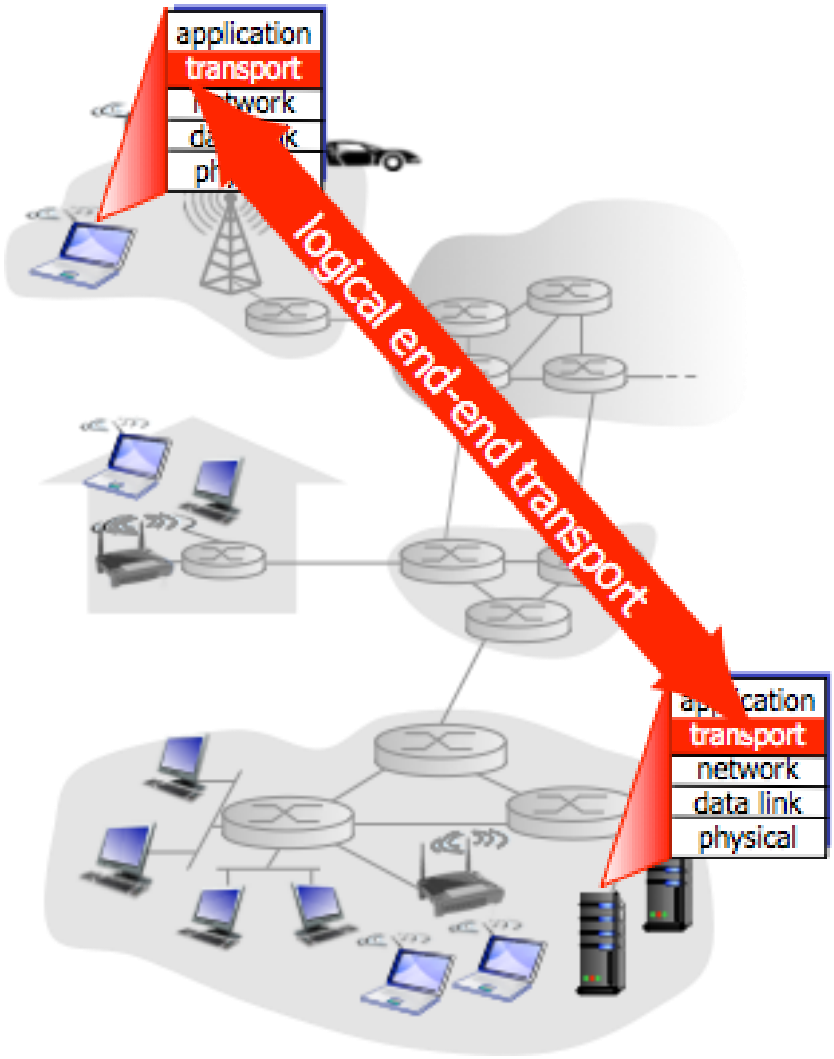
Network stack with protocol and address examples

Layer	Example Protocols	Example of address type used here
Application (AL)	HTTP, SMTP, DNS	www.ida.liu.se
Transport (TL)	TCP, UDP	
Network (NL)	IPv4, IPv6	
Link (LL)	Ethernet, WiFi (802.11)	

Transport Layer

Transport Services and Protocols

- provide *logical communication* between app processes running on different hosts
- transport protocols run in end systems
  - send side: breaks app messages into *segments*, passes to network layer
  - rcv side: reassembles segments into messages, passes to app layer
- more than one transport protocol available to apps
  - Internet: TCP and UDP

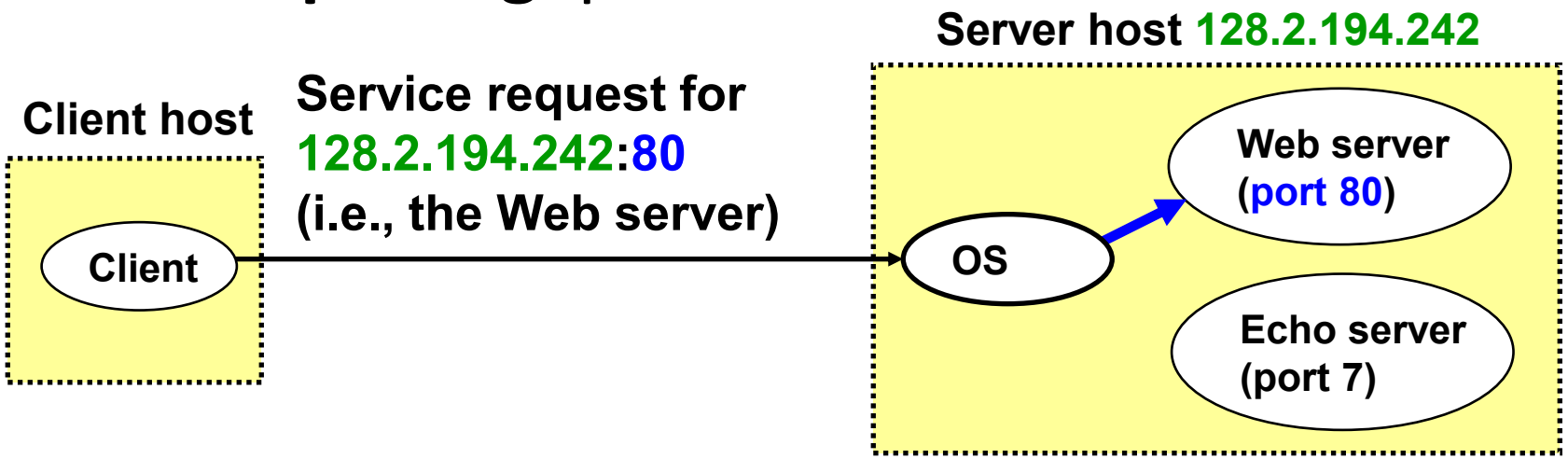


Transport vs. Network Layer

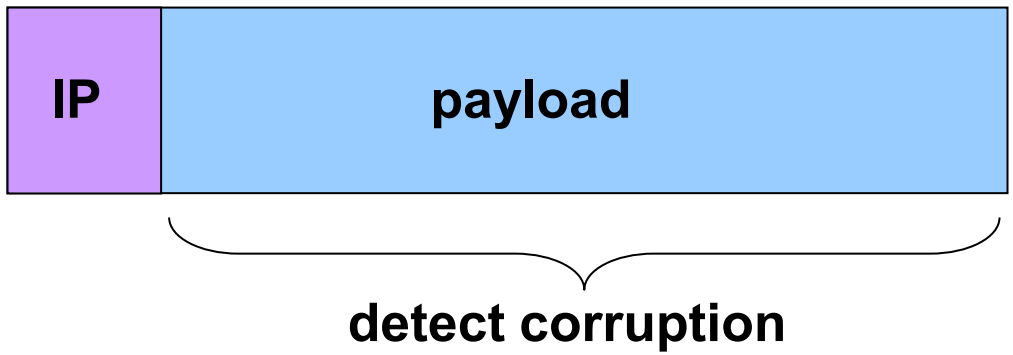
- *network layer:* logical communication between hosts
- *transport layer:* logical communication between processes
  - relies on, enhances, network layer services

Two Basic Transport Features

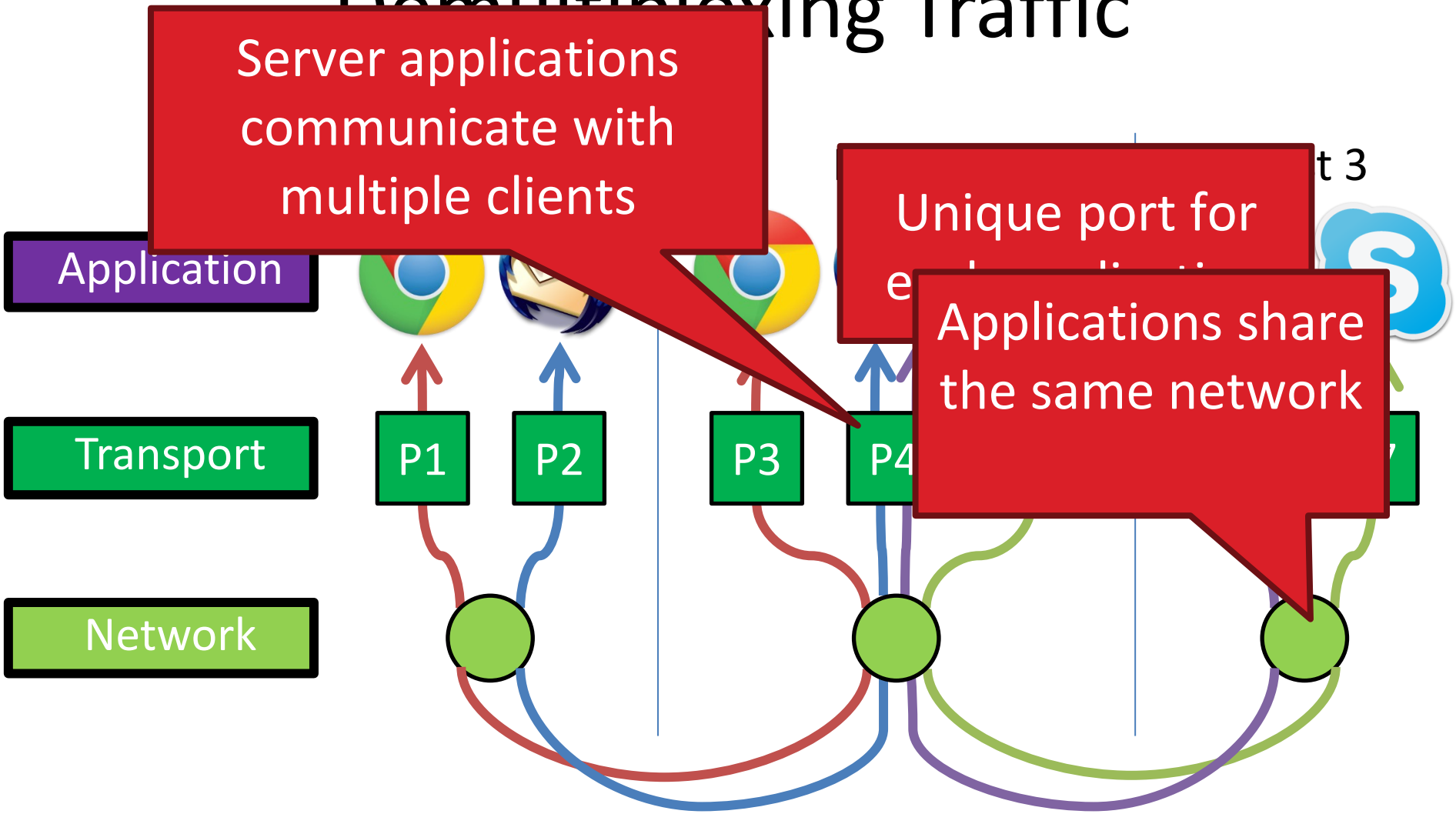
- **Demultiplexing:** port numbers



- **Error detection:** checksums



Demultiplexing Traffic



Endpoints identified by *<src_ip, src_port, dest_ip, dest_port>*

Two Main Transport Layers

- User Datagram Protocol (UDP)
  - Just provides demultiplexing and error detection
  - Header fields: port numbers, checksum, and length
  - Low overhead, good for query/response and multimedia
- Transmission Control Protocol (TCP)
  - Adds support for a "stream of bytes" abstraction
  - Retransmitting lost or corrupted data
  - Putting out-of-order data back in order
  - Preventing overflow of the **receiver buffer**
  - Adapting the sending rate to **alleviate congestion**
  - Higher overhead, good for most stateful applications

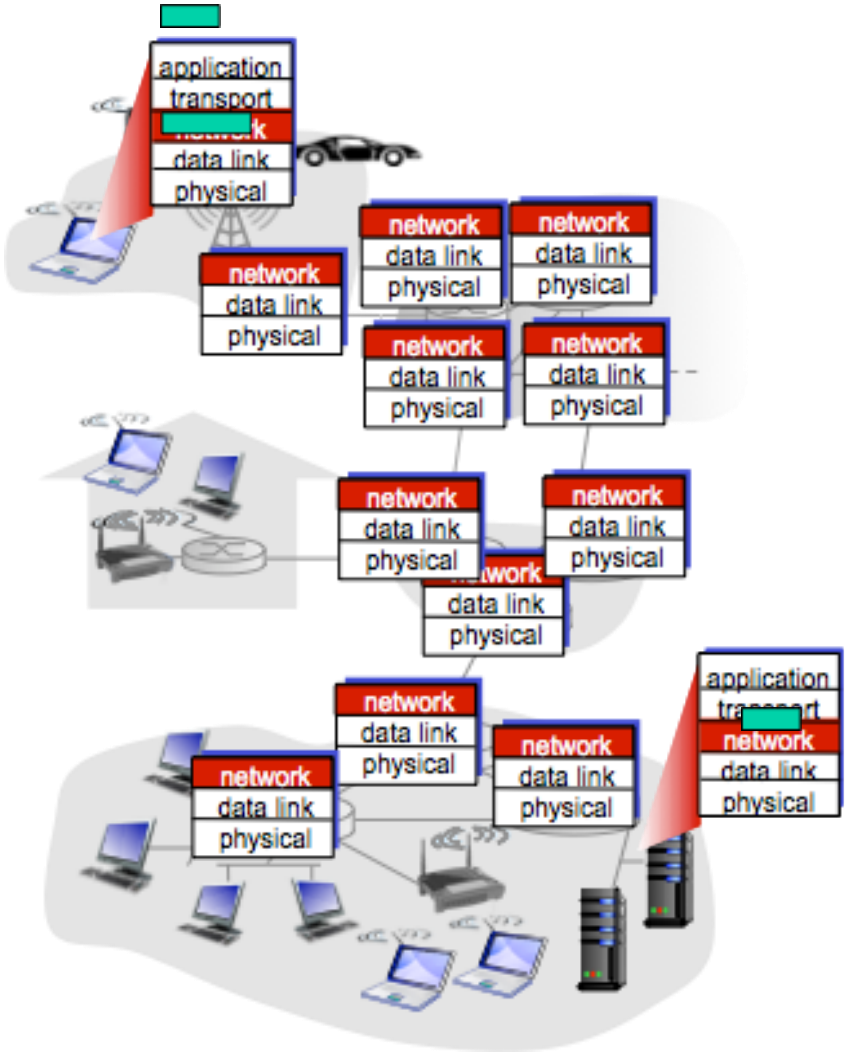
Network stack with protocol and address examples

Layer	Example Protocols	Example of address type used here
Application (AL)	HTTP, SMTP, DNS	www.ida.liu.se
Transport (TL)	TCP, UDP	port 80, port 12376
Network (NL)	IPv4, IPv6	
Link (LL)	Ethernet, WiFi (802.11)	

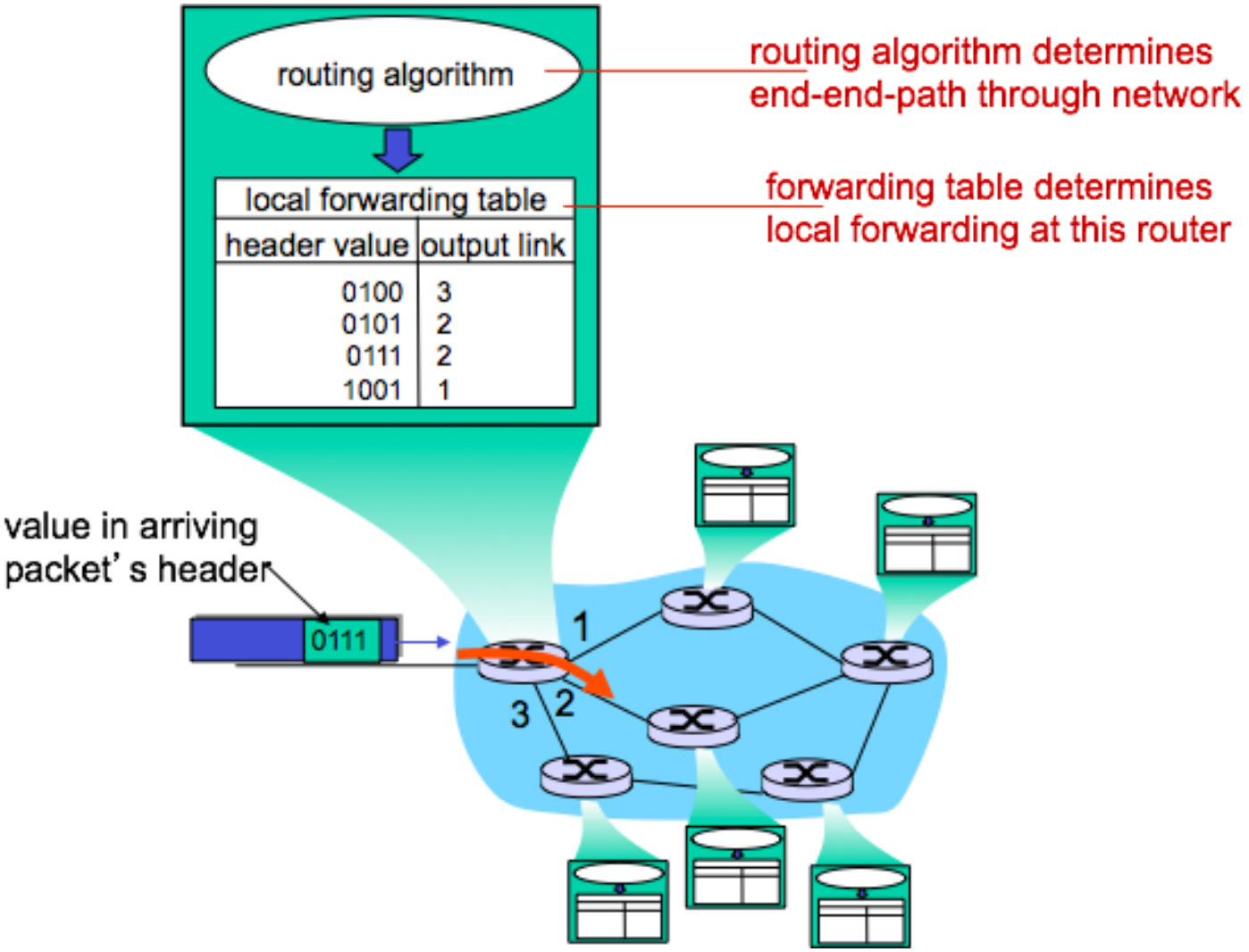
Network Layer

Network Layer

- transport segment from sending to receiving host
- on sending side encapsulates segments into datagrams
- on receiving side, delivers segments to transport layer
- network layer protocols in *every* host, router
- router examines header fields in all IP datagrams passing through it



Interplay between Routing and Forwarding

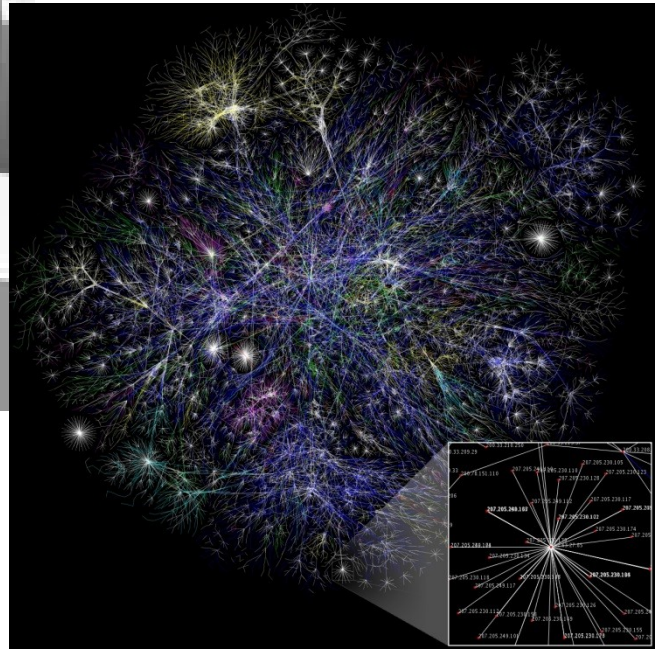


Network Layer Service Model

Q: What *service model* for "channel" transporting datagrams from sender to receiver?

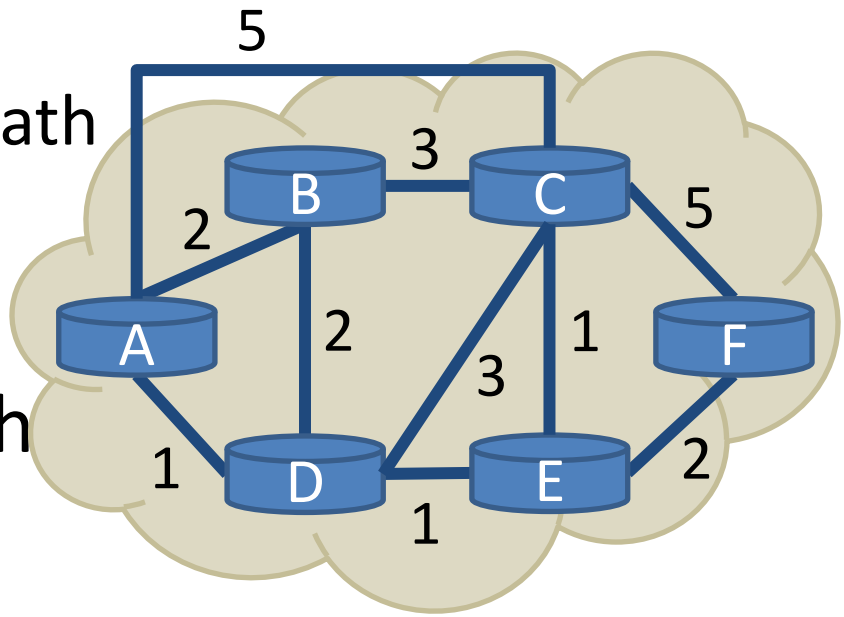
Network Architecture	Service Model	Guarantees ?				Congestion feedback
		Bandwidth	Loss	Order	Timing	
Internet	best effort	none	no	no	no	no (inferred via loss)

How do we find a path?

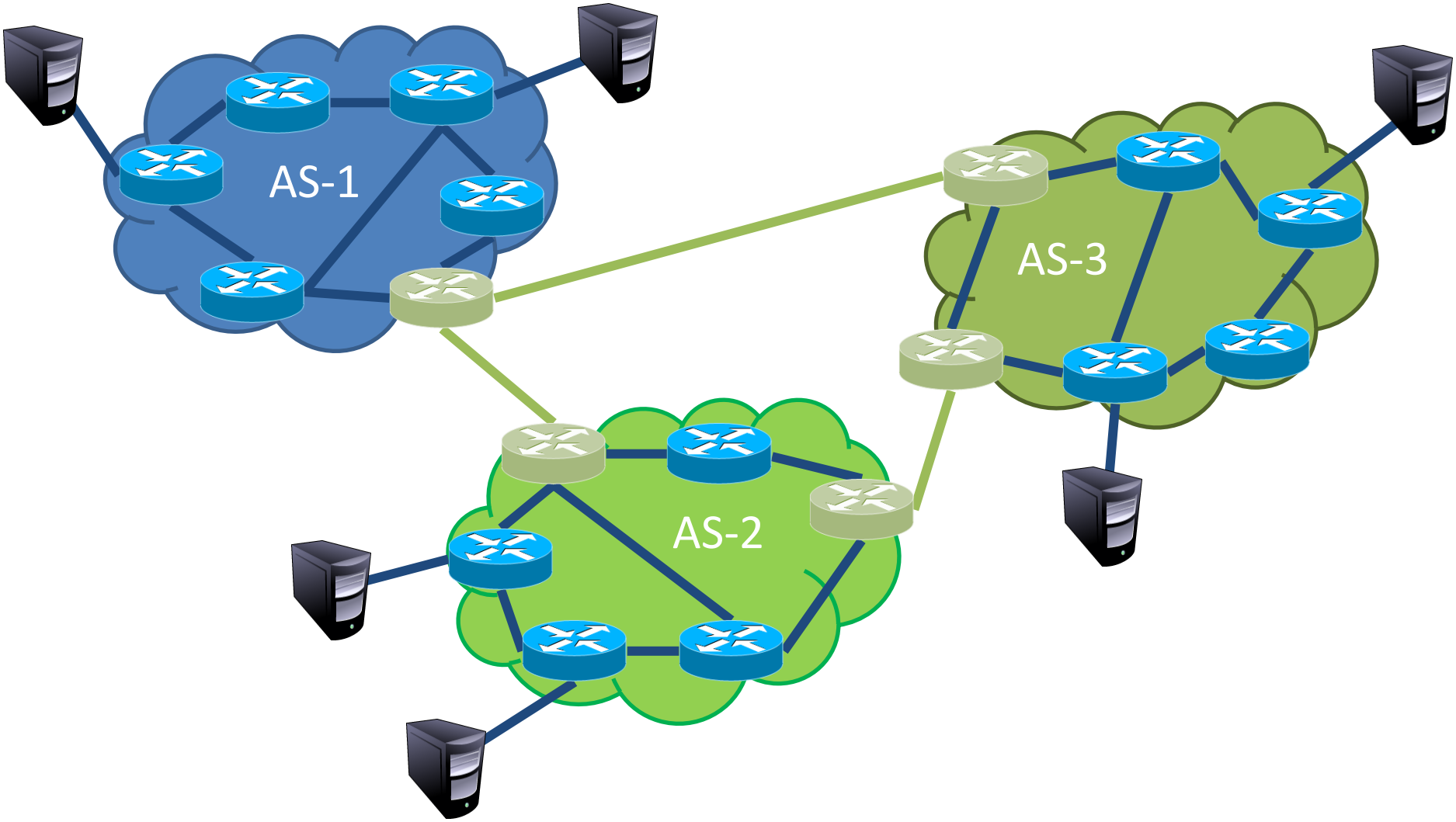


Routing on a Graph

- Goal: determine a "good" path through the network from source to destination
- What is a good path?
  - Usually means the shortest path
  - Load balanced
  - Lowest $$$ cost
- Network modeled as a graph
  - Routers → nodes
  - Link → edges
    - Edge cost: delay, congestion level, etc.

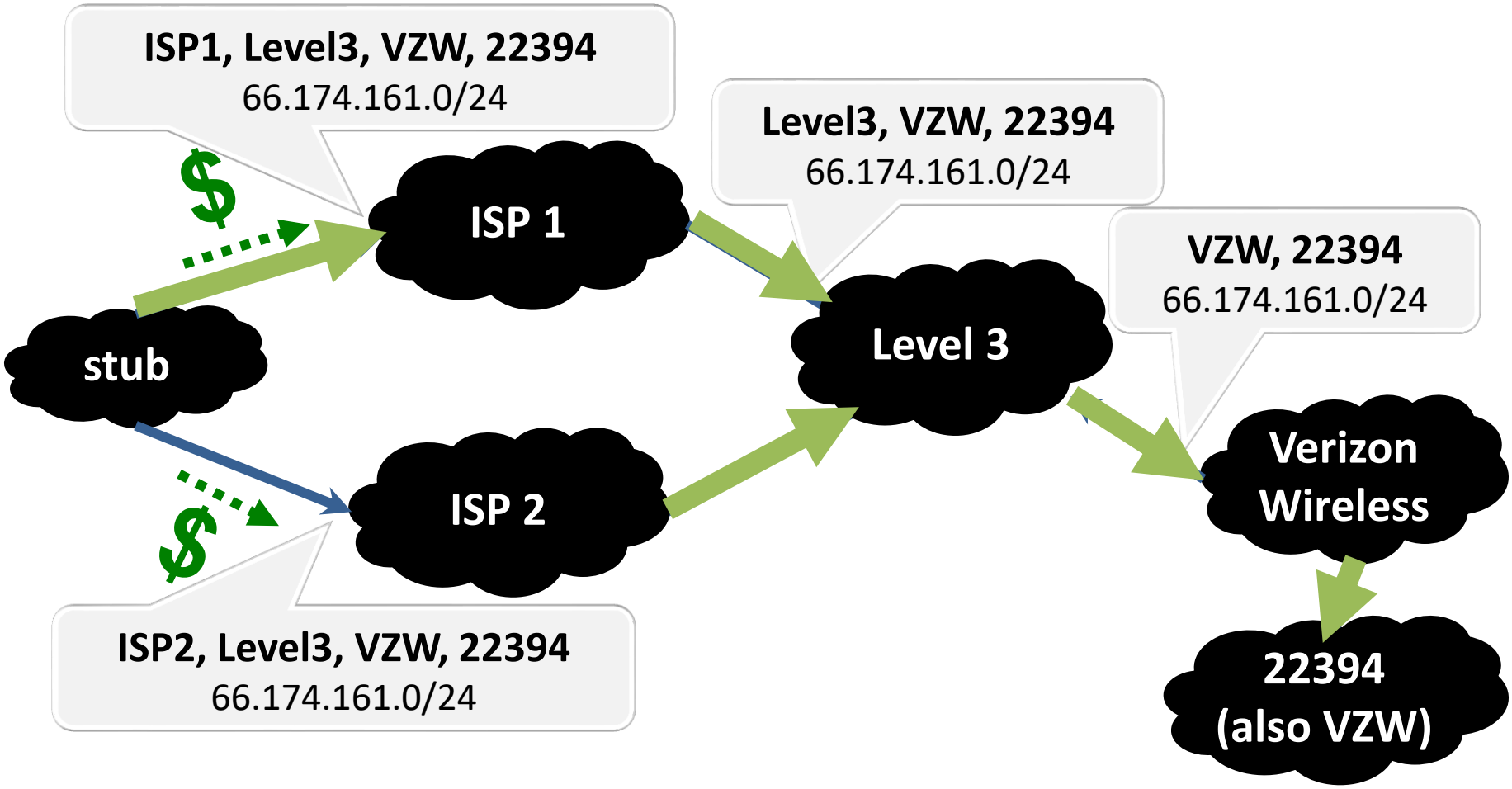


Inter vs intra domain routing



BGP: The Internet's Routing Protocol (3)

BGP sets up paths from ASes to destination IP prefixes.

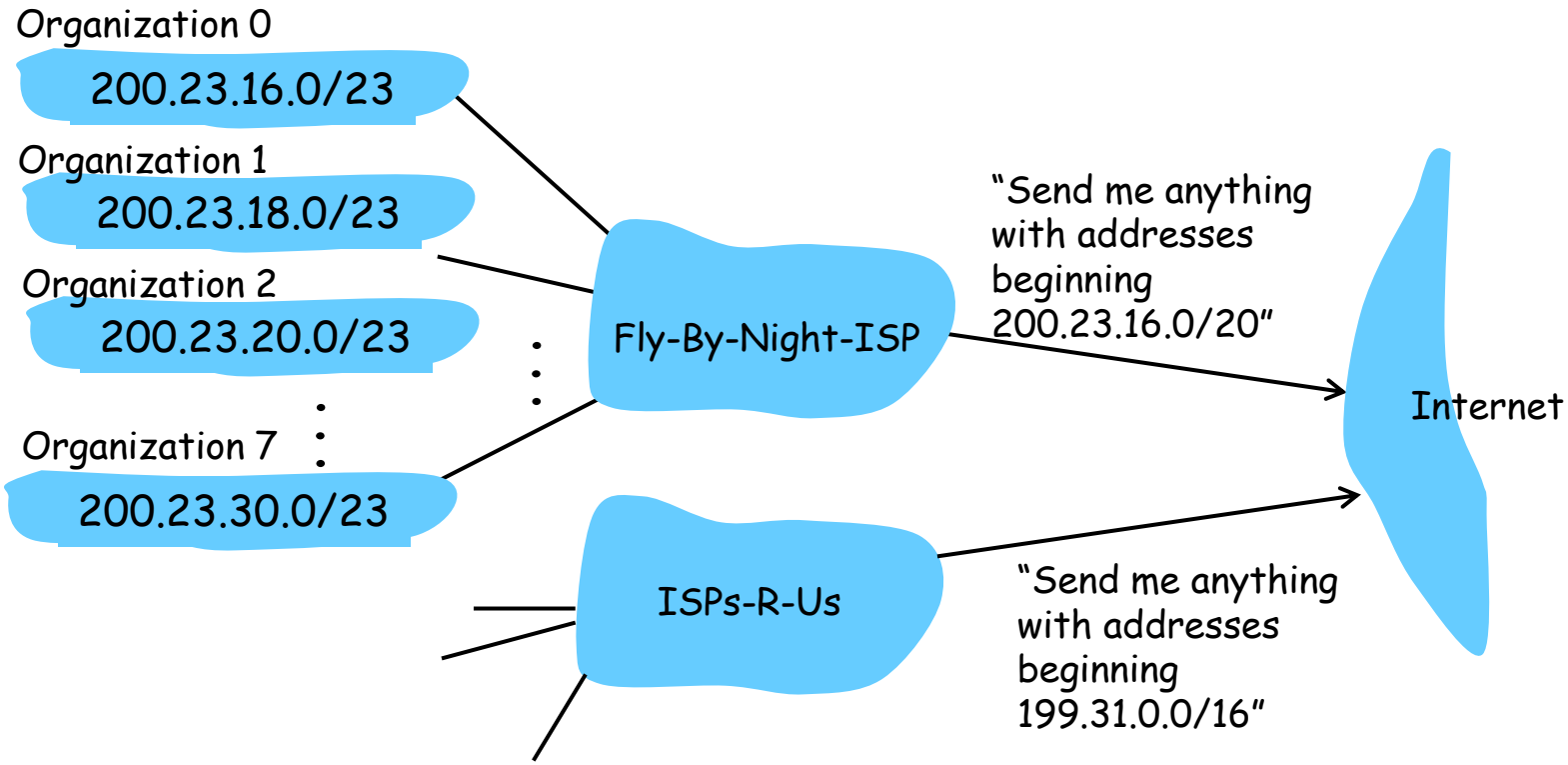


A model of BGP routing policies:

Prefer cheaper paths. Then, prefer shorter paths.

Hierarchical addressing: route aggregation

ISP has an address block; it can further divide this block into sub blocks and assign them to subscriber organizations.



Network stack with protocol and address examples

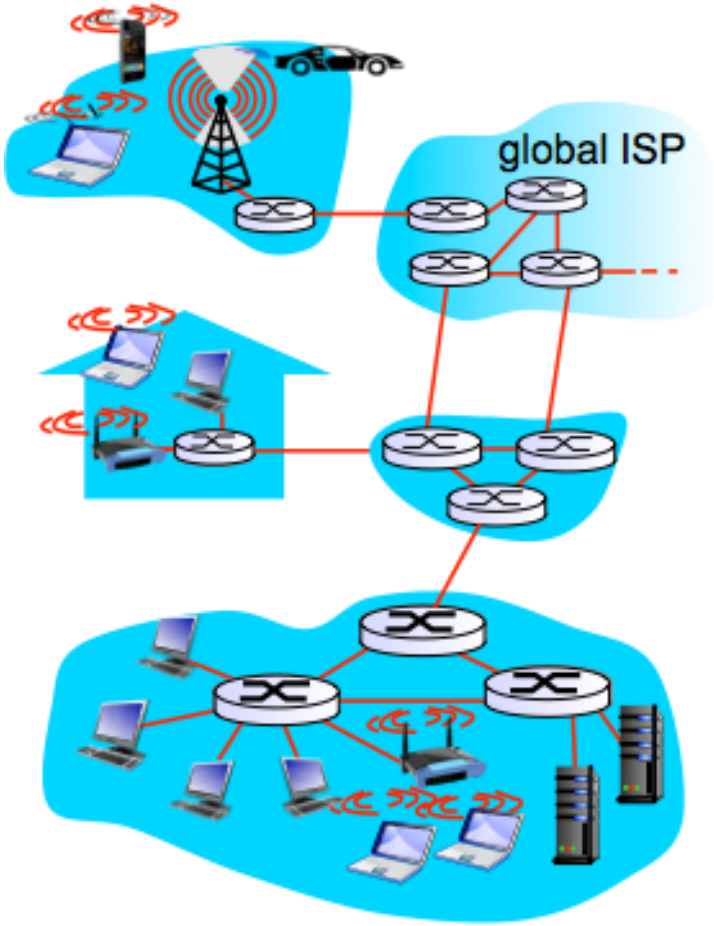
Layer	Example Protocols	Example of address type used here
Application (AL)	HTTP, SMTP, DNS	www.ida.liu.se
Transport (TL)	TCP, UDP	port 80, port 12376
Network (NL)	IPv4, IPv6	123.45.96.21
Link (LL)	Ethernet, WiFi (802.11)	

Link Layer

Link Layer

terminology:

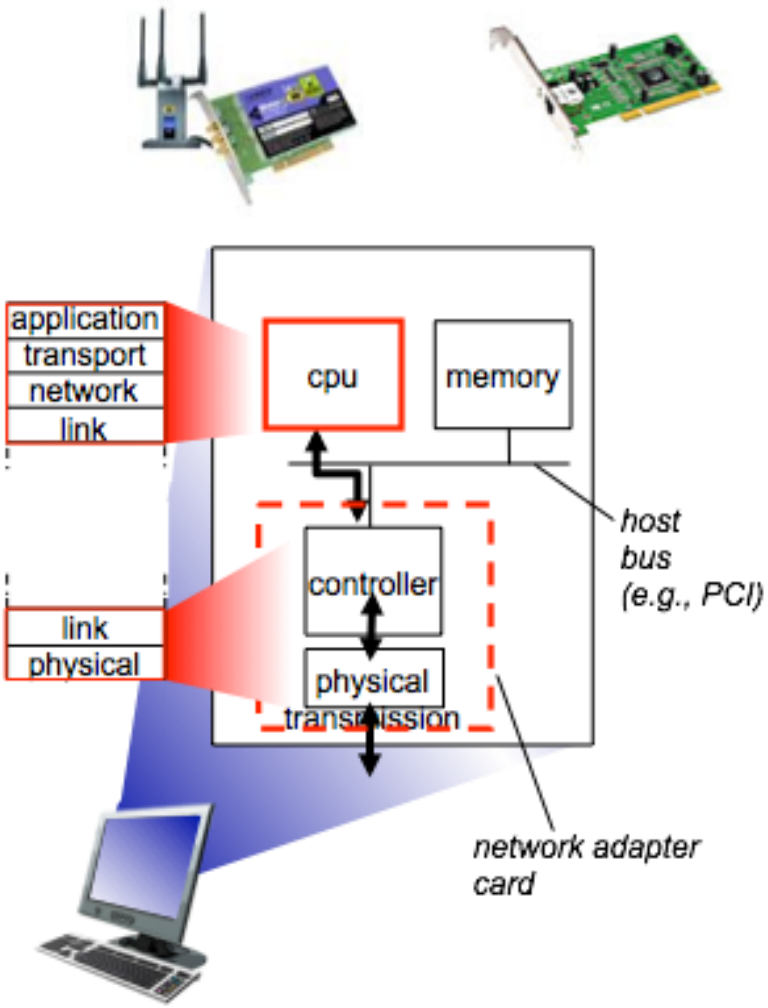
- hosts and routers: nodes
- communication channels that connect adjacent nodes along communication path: links
  - wired links
  - wireless links
  - LANs
- layer-2 packet: frame, encapsulates datagram



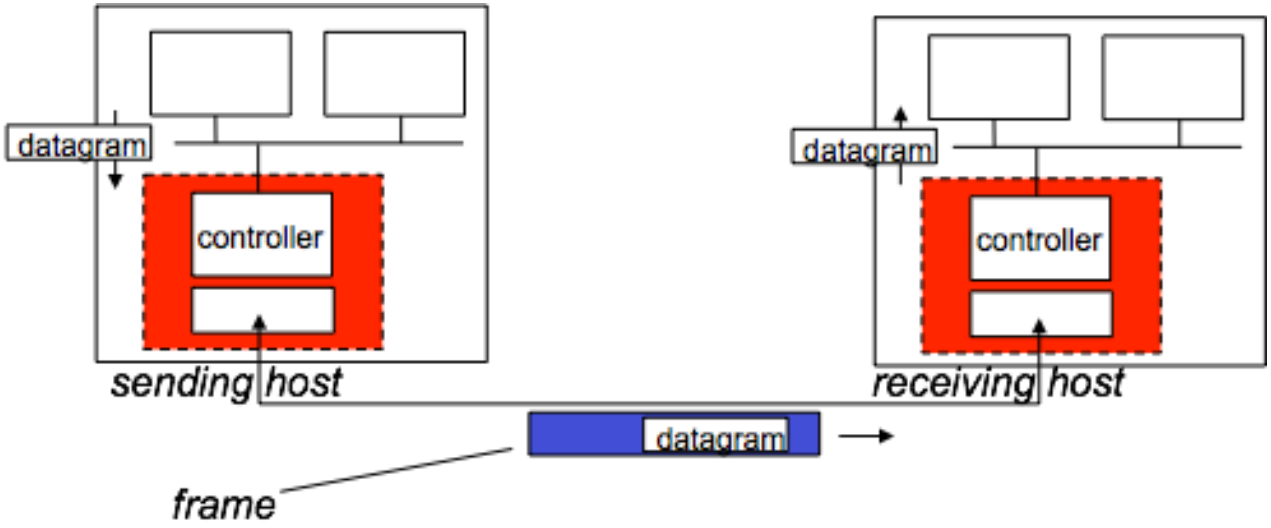
data-link layer has responsibility of transferring datagram from one node to *physically adjacent* node over a link

Where is the link layer implemented?

- in each and every host
- link layer implemented in "adaptor" (aka *network interface card* NIC) or on a chip
  - Ethernet card, 802.11 card; Ethernet chipset
  - implements link, physical layer
- attaches into host's system buses
- combination of hardware, software, firmware



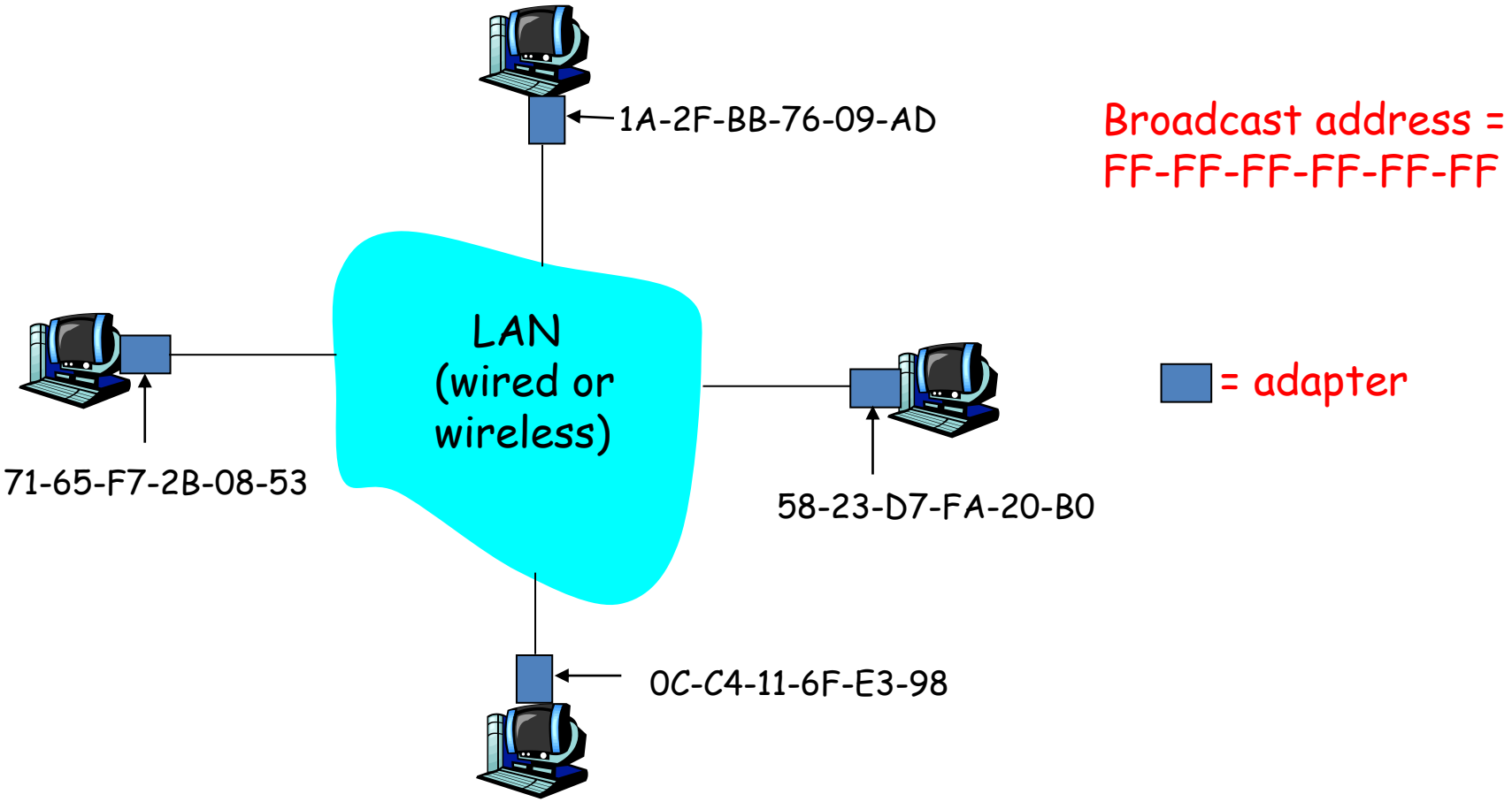
Adaptors Communicating



- sending side:
  - encapsulates datagram in frame
  - adds error checking bits, rdt, flow control, etc.
- receiving side
  - looks for errors, rdt, flow control, etc
  - extracts datagram, passes to upper layer at receiving side

MAC Addresses(2/3)

Each adapter on LAN has unique LAN address

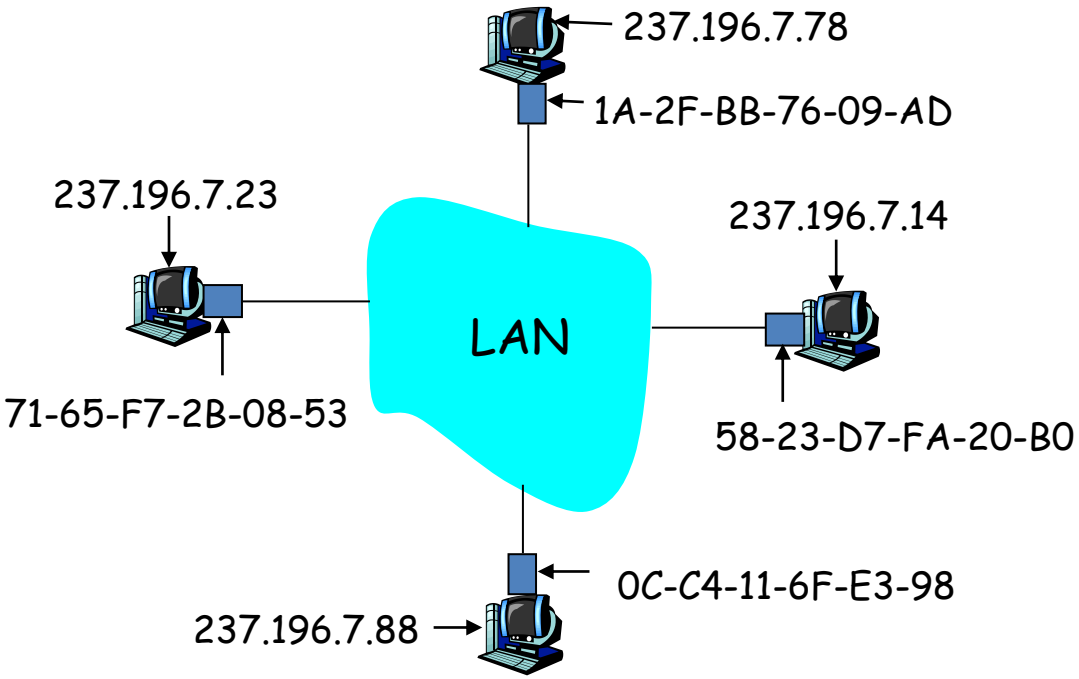


LAN Address (3/3)

- MAC address allocation administered by IEEE
- manufacturer buys portion of MAC address space
- MAC flat address provides portability
  - can move LAN card from one LAN to another
  - different than with IP addresses!

ARP: Address Resolution Protocol

Question: how to determine MAC address of B knowing B's IP address?



- Each IP node (Host, Router) on LAN has ARP table
- ARP Table: IP/MAC address mappings for some LAN nodes

  < IP address; MAC address; TTL>

  - TTL (Time To Live): time after which address mapping will be forgotten (typically 20 min)

Link Layer Services

- *framing, link access:*
  - encapsulate datagram into frame, adding header, trailer
  - channel access if shared medium
  - "MAC" addresses used in frame headers to identify source, dest
    - different from IP address!
- *reliable delivery between adjacent nodes*
  - seldom used on low bit-error link (fiber, some twisted pair)

Link Layer Services

- *flow control:*
  - pacing between adjacent sending and receiving nodes
- *error detection:*
  - errors caused by signal attenuation, noise.
  - receiver detects presence of errors:
    - signals sender for retransmission or drops frame
- *error correction:*
  - receiver identifies *and corrects* bit error(s) without resorting to retransmission
- *half-duplex and full-duplex*
  - with half duplex, nodes at both ends of link can transmit, but not at same time

Network stack with protocol and address examples

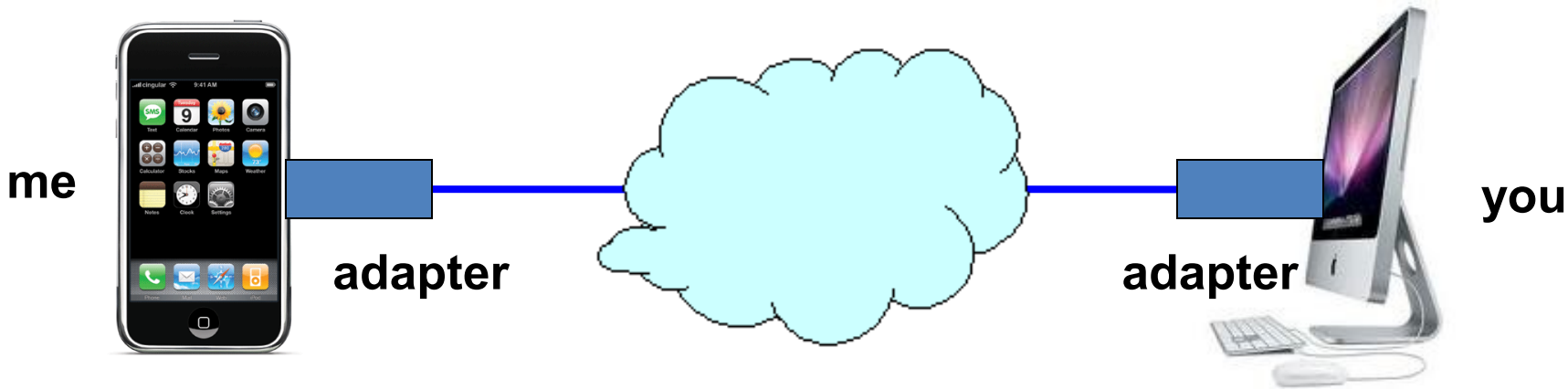
Layer	Example Protocols	Example of address type used here
Application (AL)	HTTP, SMTP, DNS	www.ida.liu.se
Transport (TL)	TCP, UDP	port 80, port 12376
Network (NL)	IPv4, IPv6	123.45.96.21
Link (LL)	Ethernet, WiFi (802.11)	AB:12:3A:45:1A:BB

Connecting the pieces

Three Kinds of Identifiers (+ports)

	Host Name	IP Address	MAC Address
Example	www.cs.princeton.edu	128.112.7.156	00-15-C5-49-04-A9
Size	Hierarchical, human readable, variable length	Hierarchical, machine readable, 32 bits (in IPv4)	Flat, machine readable, 48 bits
Read by	Humans, hosts	IP routers	Switches in LAN
Allocation, top-level	Domain, assigned by registrar (e.g., for .edu)	Variable-length prefixes, assigned by ICANN, RIR, or ISP	Fixed-sized blocks, assigned by IEEE to vendors (e.g., Dell)
Allocation, low-level	Host name, local administrator	Interface, by DHCP or an administrator	Interface, by vendor

Learning a Host's Address



- Who am I?
  - Hard-wired: MAC address
  - Static configuration: IP interface configuration
  - Dynamically learned: IP address configured by DHCP
- Who are you?
  - Hard-wired: IP address in a URL, or in the code
  - Dynamically looked up: ARP or DNS

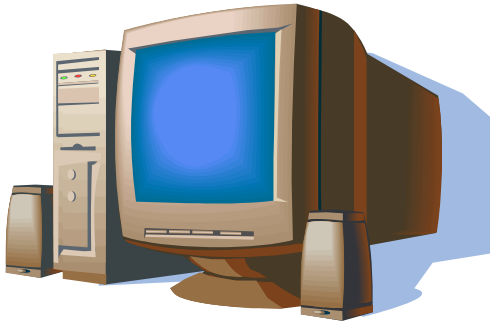
Mapping Between Identifiers

- Dynamic Host Configuration Protocol (DHCP)
  - Given a MAC address, assign a unique IP address
  - … and tell host other stuff about the Local Area Network
  - To automate the boot-strapping process
- Address Resolution Protocol (ARP)
  - Given an IP address, provide the MAC address
  - To enable communication within the Local Area Network
- Domain Name System (DNS)
  - Given a host name, provide the IP address
  - Given an IP address, provide the host name

Dynamic Host Configuration Protocol

arriving client

DHCP discover (broadcast)



DHCP server

DHCP offer

Host learns
IP address,
Subnet mask,
Gateway address,
DNS server(s),
and a lease time.

DHCP request (broadcast)

DHCP ACK

Network stack with protocol and address examples

Layer	Example Protocols	Example of address type used here
Application (AL)	HTTP, SMTP, DNS	www.ida.liu.se
Transport (TL)	TCP, UDP	port 80, port 12376
Network (NL)	IPv4, IPv6	123.45.96.21
Link (LL)	Ethernet, WiFi (802.11)	AB:12:3A:45:1A:BB

Courses about Computer Networks

- TDTS06 Computer Networks (6hp)
  - D program: Recommended elective …
- TDDE35 Large-scale Systems (11hp)
  - U program: Second year course covering computer networking, distributed systems, multicore, embedded systems, and a project
- TDTS21 Advance Networking (6p)
  - Pre-requirement: Introductory networking course; e.g., TDDE35 (U), TDTS04 (IP, C, …), TDTS06 (D, Y, …), TDTS11 (IT)
- Thesis opportunities
  - Companies often have projects
  - I have many additional research projects (on these and related topics, including novel multimedia streaming solutions, cloud, IoT, data analytics/mining, network security, social networking, …, but also other things (e.g., sports analytics))

Extra slides …

FTP: Commands and Responses

sample commands:

- sent as ASCII text over control channel
- **USER *username***
- **PASS *password***
- **LIST** return list of file in current directory
- **RETR filename** retrieves (gets) file
- **STOR filename** stores (puts) file onto remote host

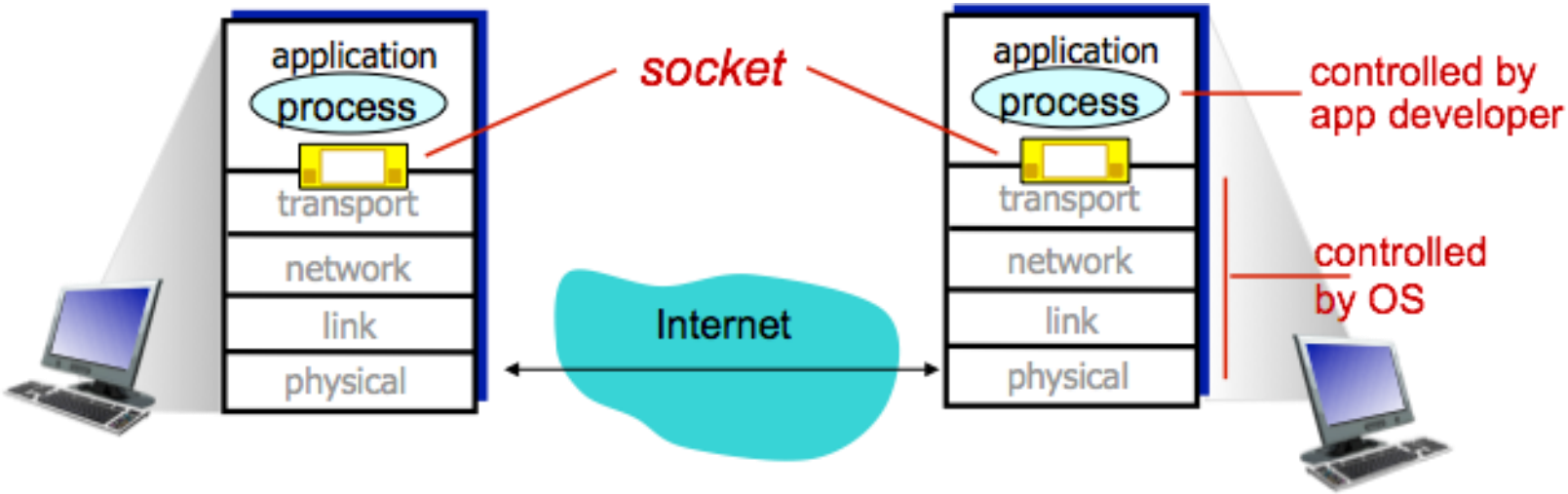
sample return codes

- status code and phrase (as in HTTP)
- **331 Username OK, password required**
- **125 data connection already open; transfer starting**
- **425 Can't open data connection**
- **452 Error writing file**

Socket Programming

goal: learn how to build client/server applications that communicate using sockets

socket: door between application process and end-end-transport protocol



Socket Programming

Two socket types for two transport services:

- *UDP:* unreliable datagram
- *TCP:* reliable, byte stream-oriented

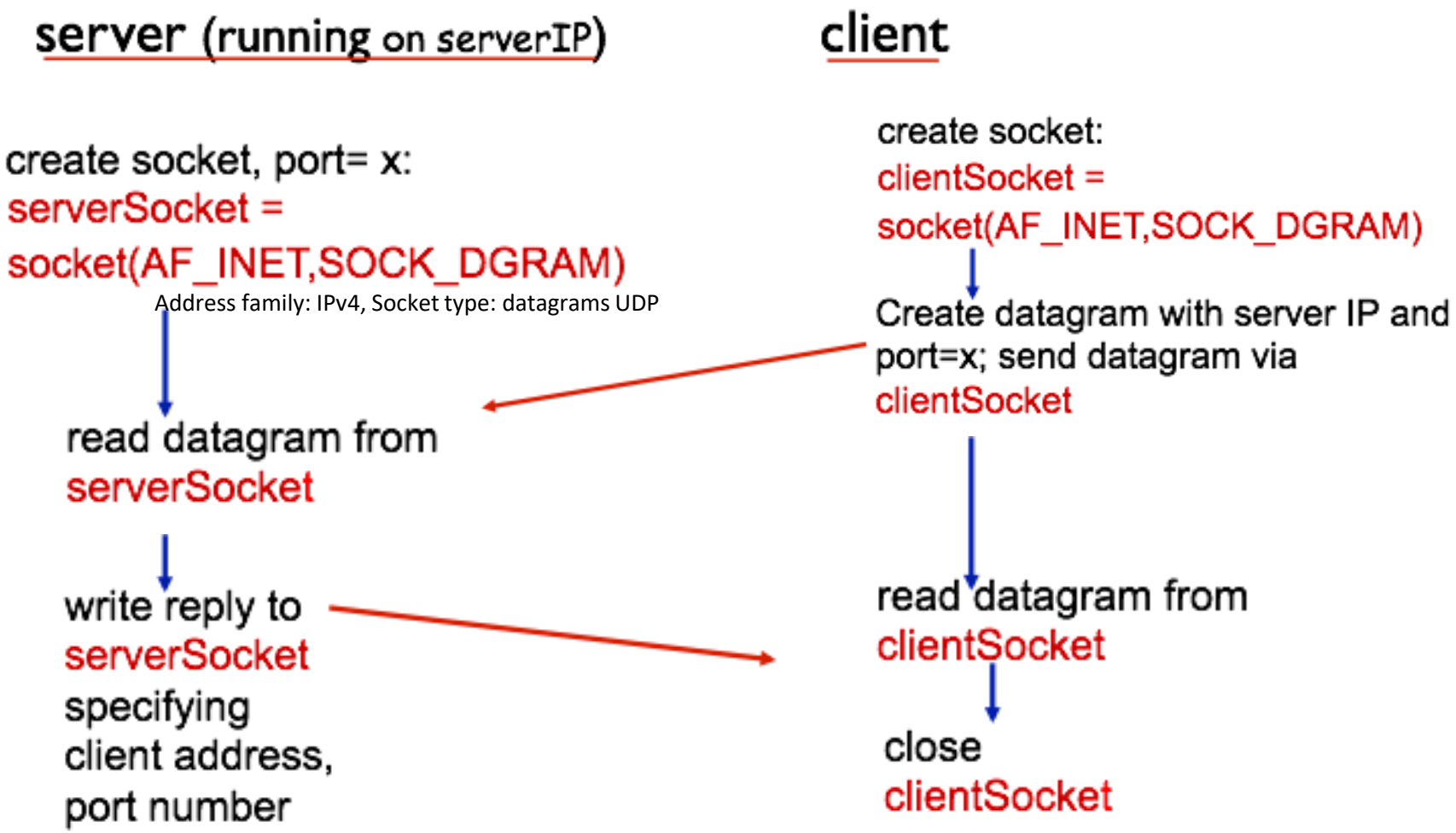
Application Example:

1. Client reads a line of characters (data) from its keyboard and sends the data to the server.
2. The server receives the data and converts characters to uppercase.
3. The server sends the modified data to the client.
4. The client receives the modified data and displays the line on its screen.

Socket Programming with UDP

- UDP: no “connection” between client & server
  - no handshaking before sending data
  - sender explicitly attaches IP destination address and port # to each packet
  - rcvr extracts sender IP address and port# from received packet
- UDP: transmitted data may be lost or received out-of-order
- Application viewpoint:
  - UDP provides unreliable transfer of groups of bytes (“datagrams”) between client and server

Client/Server Socket Interaction: UDP



Example App: UDP Client

Python UDPClient

include Python's socket library

```
from socket import *
serverName = 'hostname'
serverPort = 12000
```

create UDP socket for server

```
clientSocket = socket(socket.AF_INET,
                      socket.SOCK_DGRAM)
```

get user keyboard input

```
message = raw_input('Input lowercase sentence:')
```

Attach server name, port to message; send into socket

```
clientSocket.sendto(message,(serverName, serverPort))
```

read reply characters from socket into string

```
modifiedMessage, serverAddress =
                      clientSocket.recvfrom(2048)
```

print out received string and close socket

```
print modifiedMessage
clientSocket.close()
```

Example App: UDP Server

Python UDPServer

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET, SOCK_DGRAM)
serverSocket.bind(('', serverPort))
print "The server is ready to receive"
while 1:
    message, clientAddress = serverSocket.recvfrom(2048)
    modifiedMessage = message.upper()
    serverSocket.sendto(modifiedMessage, clientAddress)
```

create UDP socket

bind socket to local port number 12000

loop forever

Read from UDP socket into message, getting client's address (client IP and port)

send upper case string back to this client

Socket Programming with TCP

client must contact server

- server process must first be running
- server must have created socket (door) that welcomes client's contact

client contacts server by:

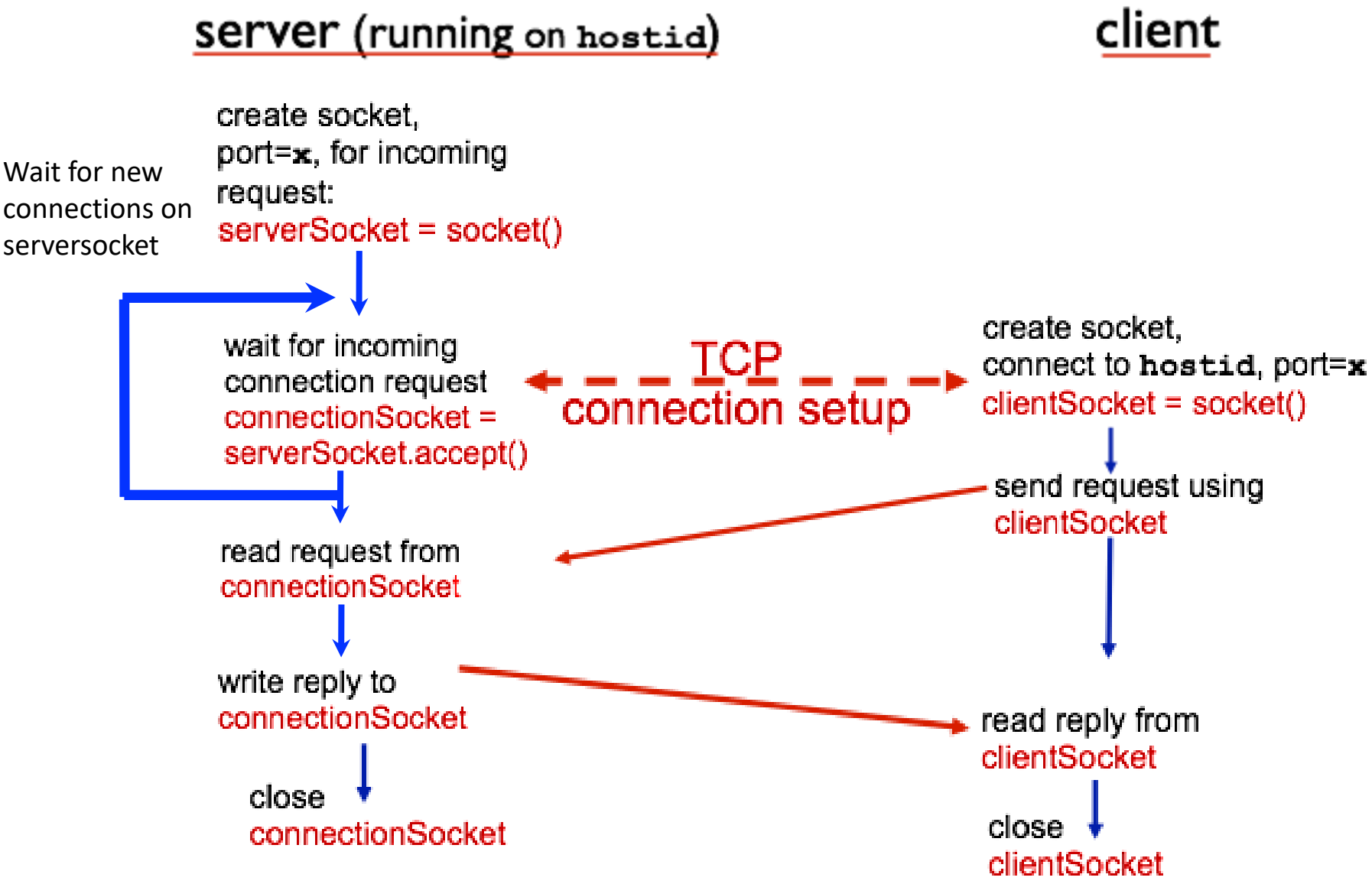
- Creating TCP socket, specifying IP address, port number of server process
- *when client creates socket:* client TCP establishes connection to server TCP

- when contacted by client, *server TCP creates new socket* for server process to communicate with that particular client
  - allows server to talk with multiple clients
  - source port numbers used to distinguish clients

application viewpoint:

TCP provides reliable, in-order byte-stream transfer ("pipe") between client and server

Client/Server Socket Interaction: TCP



Example App: TCP Client

Python TCPClient

```
from socket import *
serverName = 'servername'
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName,serverPort))
sentence = raw_input('Input lowercase sentence:')
clientSocket.send(sentence)
modifiedSentence = clientSocket.recv(1024)
print 'From Server:', modifiedSentence
clientSocket.close()
```

create TCP socket for server, remote port 12000

Address family: IPv4, Socket type: TCP

Connect to remote socket

No need to attach server name, port

# of bytes

Example App: TCP Server

Python TCPServer

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET,SOCK_STREAM)
serverSocket.bind(('',serverPort))
serverSocket.listen(1)
print 'The server is ready to receive'
while 1:
    connectionSocket, addr = serverSocket.accept()

    sentence = connectionSocket.recv(1024)
    capitalizedSentence = sentence.upper()
    connectionSocket.send(capitalizedSentence)
    connectionSocket.close()
```

create TCP welcoming socket

server begins listening for incoming TCP requests

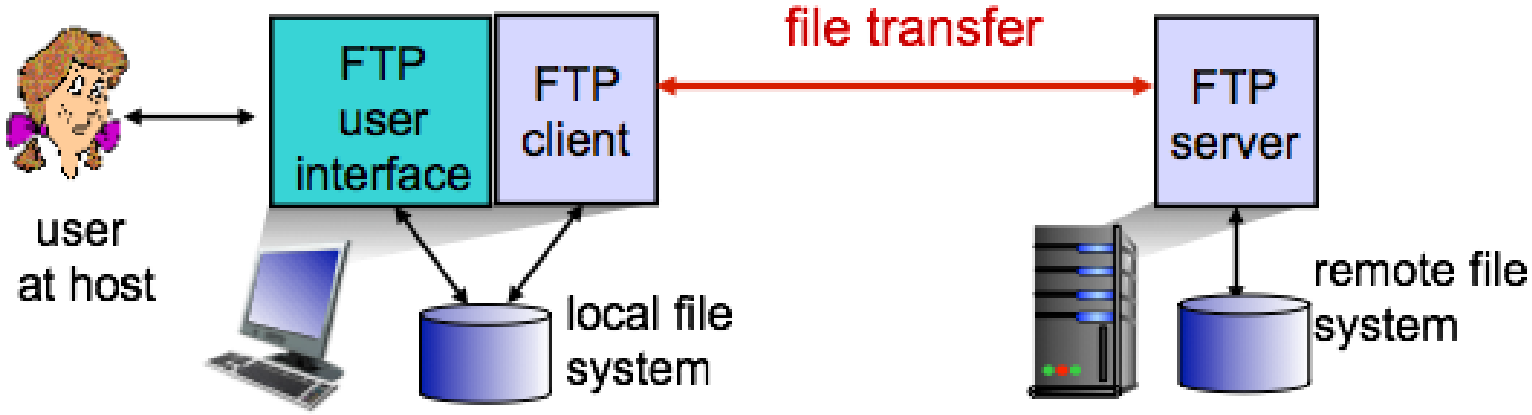
loop forever

server waits on accept() for incoming requests, new socket created on return

read bytes from socket (but not address as in UDP)

close connection to this client (but *not* welcoming socket)

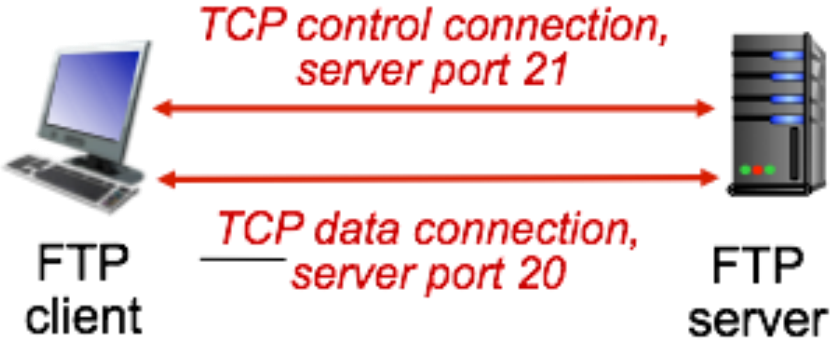
FTP: File Transfer Protocol



- transfer file to/from remote host
- client/server model
  - *client:* side that initiates transfer (either to/from remote)
  - *server:* remote host
- ftp: RFC 959
- ftp server: port 21

FTP: Separate Control/Data Connections

- FTP client contacts FTP server at port 21, using TCP
- client authorized over control connection
- client browses remote directory, sends commands over control connection
- when server receives file transfer command, *server* opens 2nd TCP data connection (for file) *to* client
- after transferring one file, server closes data connection



- server opens another TCP data connection to transfer another file
- control connection: *"out of band"*
- FTP server maintains "state": current directory, earlier authentication

Roadmap

- Principles of Network Applications
  - App Architectures
  - App Requirements
- Web and HTTP
- FTP
- Electronic Mail
  - SMTP, POP3, IMAP
- DNS
- P2P Applications
- Socket Programming with UDP and TCP

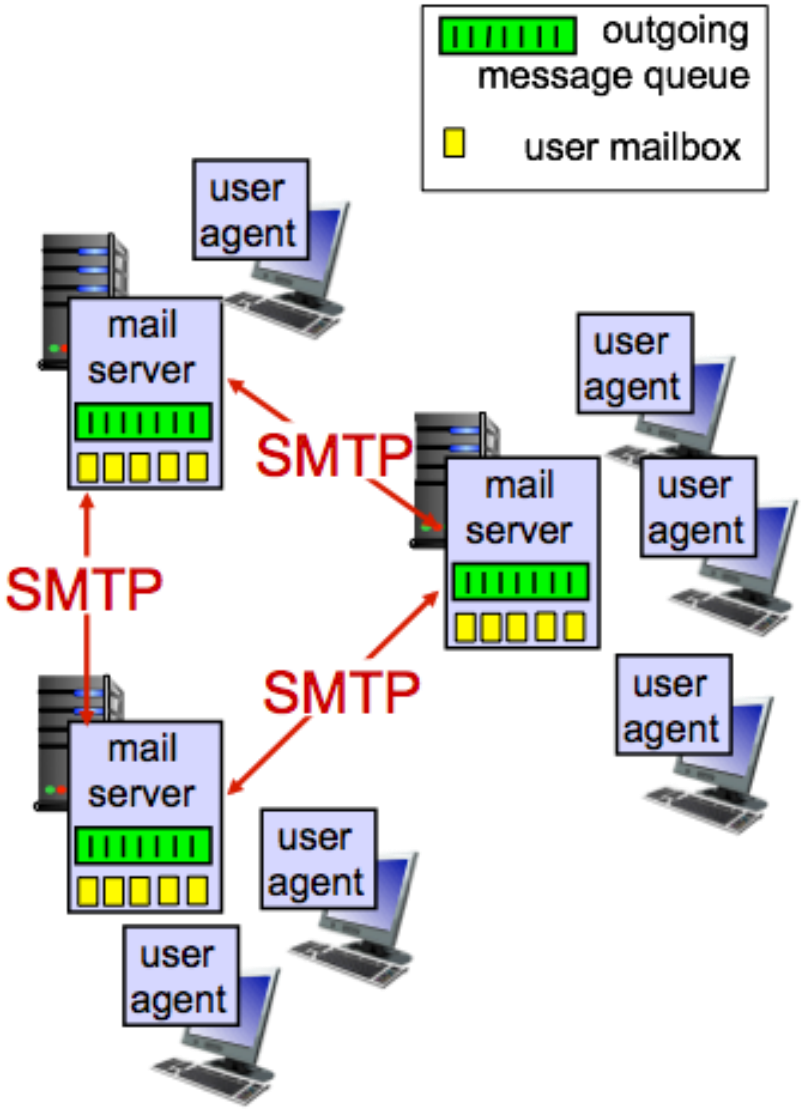
Electronic Mail

Three major components:

- user agents
- mail servers
- simple mail transfer protocol: SMTP

User Agent

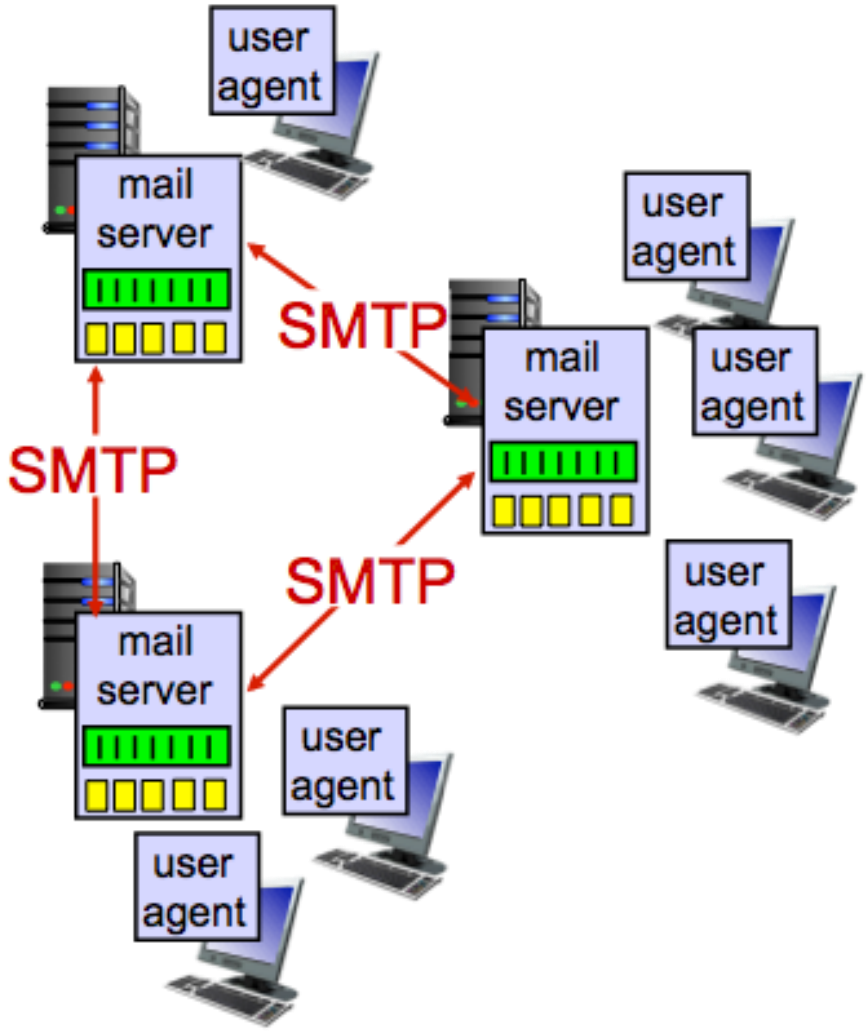
- a.k.a. "mail reader"
- composing, editing, reading mail messages
- e.g., Outlook, Thunderbird, iPhone mail client
- outgoing, incoming messages stored on server



Electronic Mail: Mail Servers

mail servers:

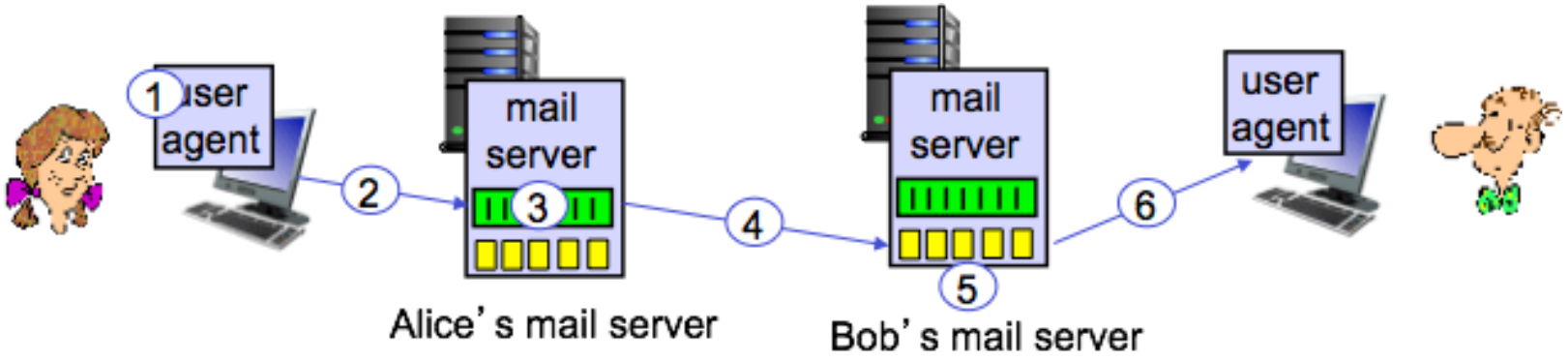
- *mailbox* contains incoming messages for user
- *message queue* of outgoing (to be sent) mail messages
- *SMTP protocol* between mail servers to send email messages
  - client: sending mail server
  - "server": receiving mail server



Electronic Mail: SMTP [RFC 2821]

- uses TCP to reliably transfer email message from client to server, port 25
- direct transfer: sending server to receiving server
- three phases of transfer
  - handshaking (greeting)
  - transfer of messages
  - closure
- command/response interaction (like HTTP, FTP)
  - commands: ASCII text
  - response: status code and phrase
- messages must be in 7-bit ASCI

Scenario: Alice Sends Message to Bob



1) Alice uses UA to compose message "to" `bob@someschool.edu`
2) Alice's UA sends message to her mail server; message placed in message queue
3) client side of SMTP opens TCP connection with Bob's mail server
4) SMTP client sends Alice's message over the TCP connection
5) Bob's mail server places the message in Bob's mailbox
6) Bob invokes his user agent to read message

UA: User agent

Sample SMTP Interaction

```
S: 220 hamburger.edu
C: HELO crepes.fr
S: 250  Hello crepes.fr, pleased to meet you
C: MAIL FROM: <alice@crepes.fr>
S: 250 alice@crepes.fr... Sender ok
C: RCPT TO: <bob@hamburger.edu>
S: 250 bob@hamburger.edu ... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Do you like ketchup?
C: How about pickles?
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 hamburger.edu closing connection
```

Both are mailservers!

S: Server
C: Client

Try SMTP Interaction!

- **`telnet servername 25`**
- see 220 reply from server
- enter HELO, MAIL FROM, RCPT TO, DATA, QUIT commands

above lets you send email without using email client (reader)

```
prompt$ telnet mail.liu.se 25
Trying 130.236.27.19...
Connected to mail.liu.se (130.236.27.19).
Escape character is '^]'.
220 HC3-2010.ad.liu.se Microsoft ESMTP MAIL Service ready at
Fri, 25 Sep 2015 07:51:45 +0200
HELO
250 HC3-2010.ad.liu.se Hello [130.236.180.74]
QUIT
221 2.0.0 Service closing transmission channel
Connection closed by foreign host.
prompt$
```

SMTP: Comparison with HTTP

- SMTP uses persistent connections
- SMTP requires message (header & body) to be in 7-bit ASCII
- SMTP server uses `CRLF.CRLF` to determine end of message

comparison with HTTP:

- HTTP: pull
- SMTP: push
- both have ASCII command/response interaction, status codes
- HTTP: each object encapsulated in its own response msg
- SMTP: multiple objects sent in multipart msg

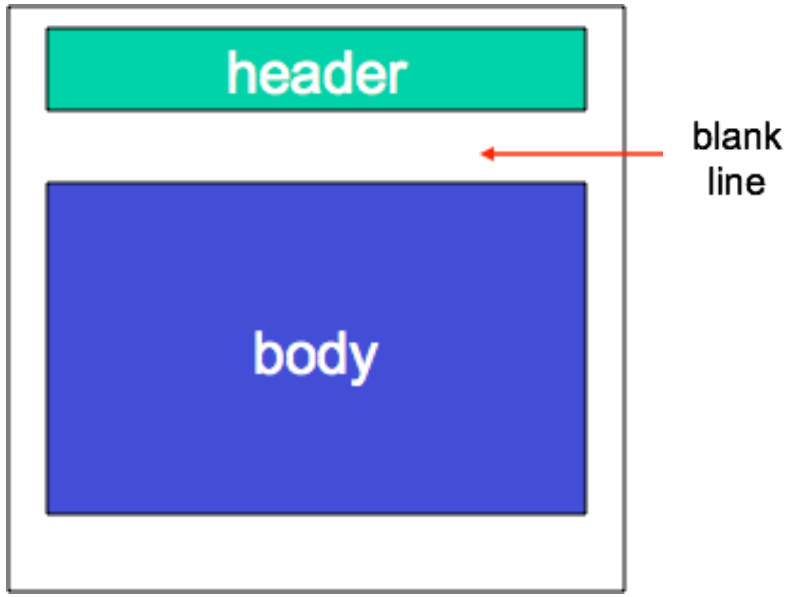
Mail Message Format

SMTP: protocol for exchanging email msgs

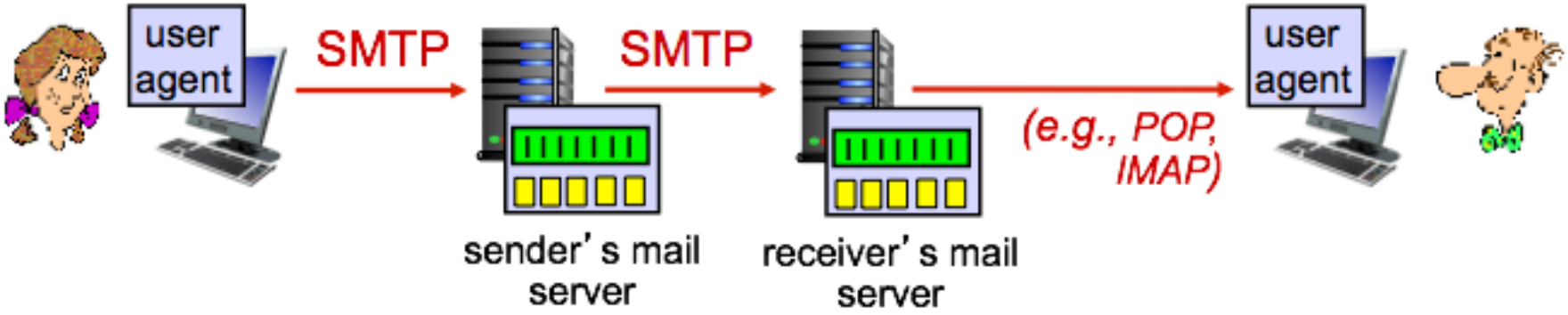
RFC 822: standard for text message format:

- header lines, e.g.,
  - To:
  - From:
  - Subject:

  different from SMTP MAIL FROM, RCPT TO: commands!
- Body: the "message"
  - ASCII characters only



Mail Access Protocols



- SMTP: delivery/storage to receiver's server
- mail access protocol: retrieval from server
  - POP: Post Office Protocol [RFC 1939]: authorization, download
  - IMAP: Internet Mail Access Protocol [RFC 1730]: more features, including manipulation of stored msgs on server
  - HTTP: gmail, Hotmail, Yahoo! Mail, etc.