CS 3800: Computer Networks

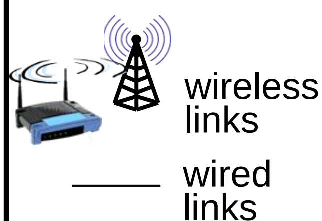
Lecture 1: Introduction

Instructor: John Korah

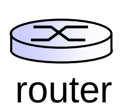
INTRODUCTION

The Internet: “nuts and bolts” view

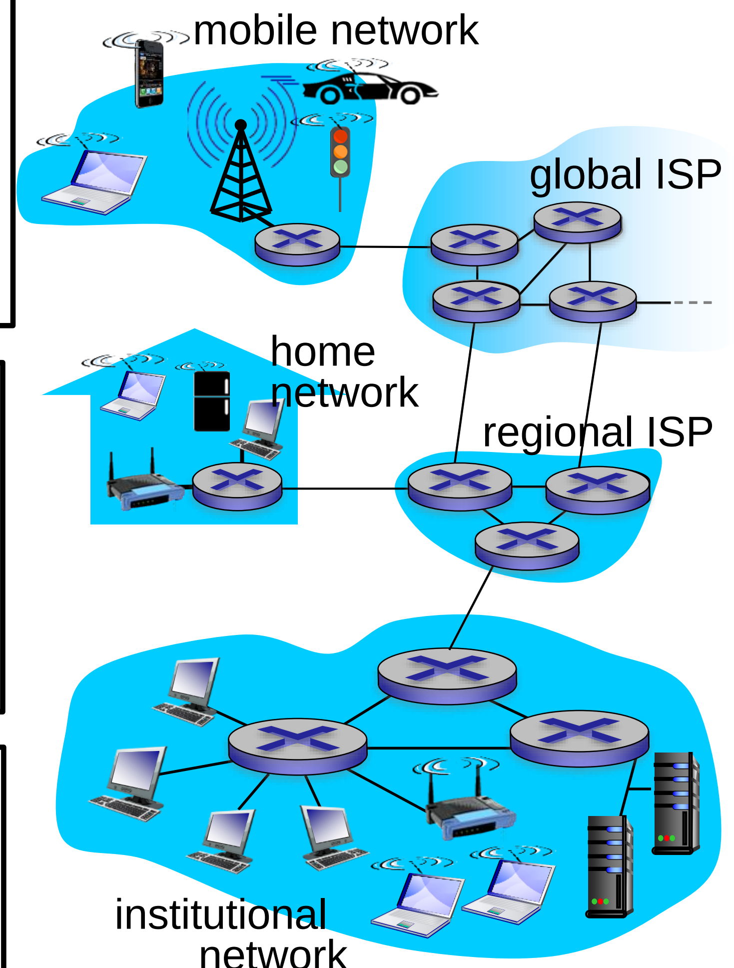
- billions of connected computing devices:
  - *hosts* = *end systems*
  - running *network apps*



- *communication links*
  - fiber, copper, radio, satellite
  - transmission rate: *bandwidth*

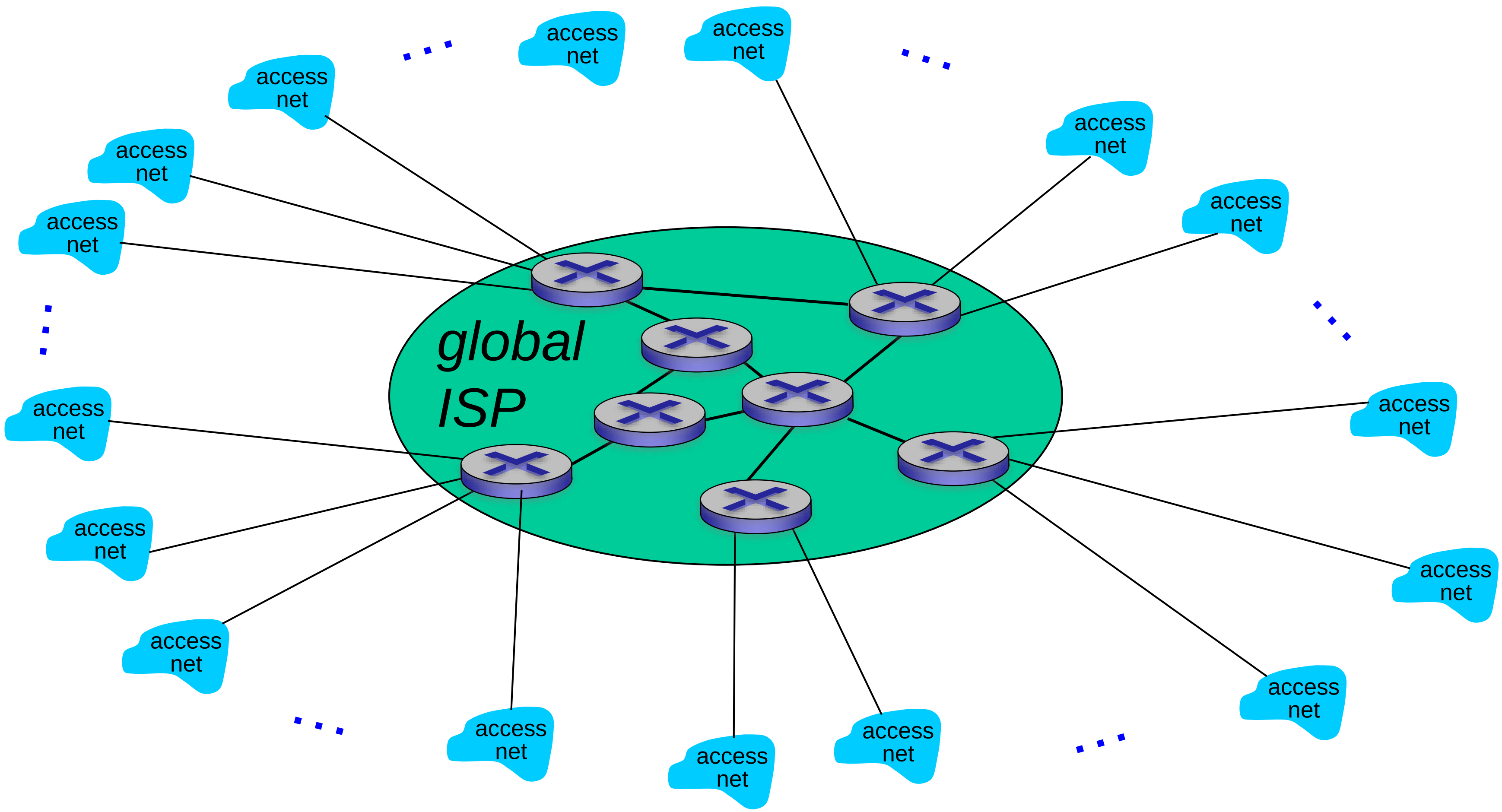


- *switches:* forward packets (chunks of data)
  - *routers* and *switches*



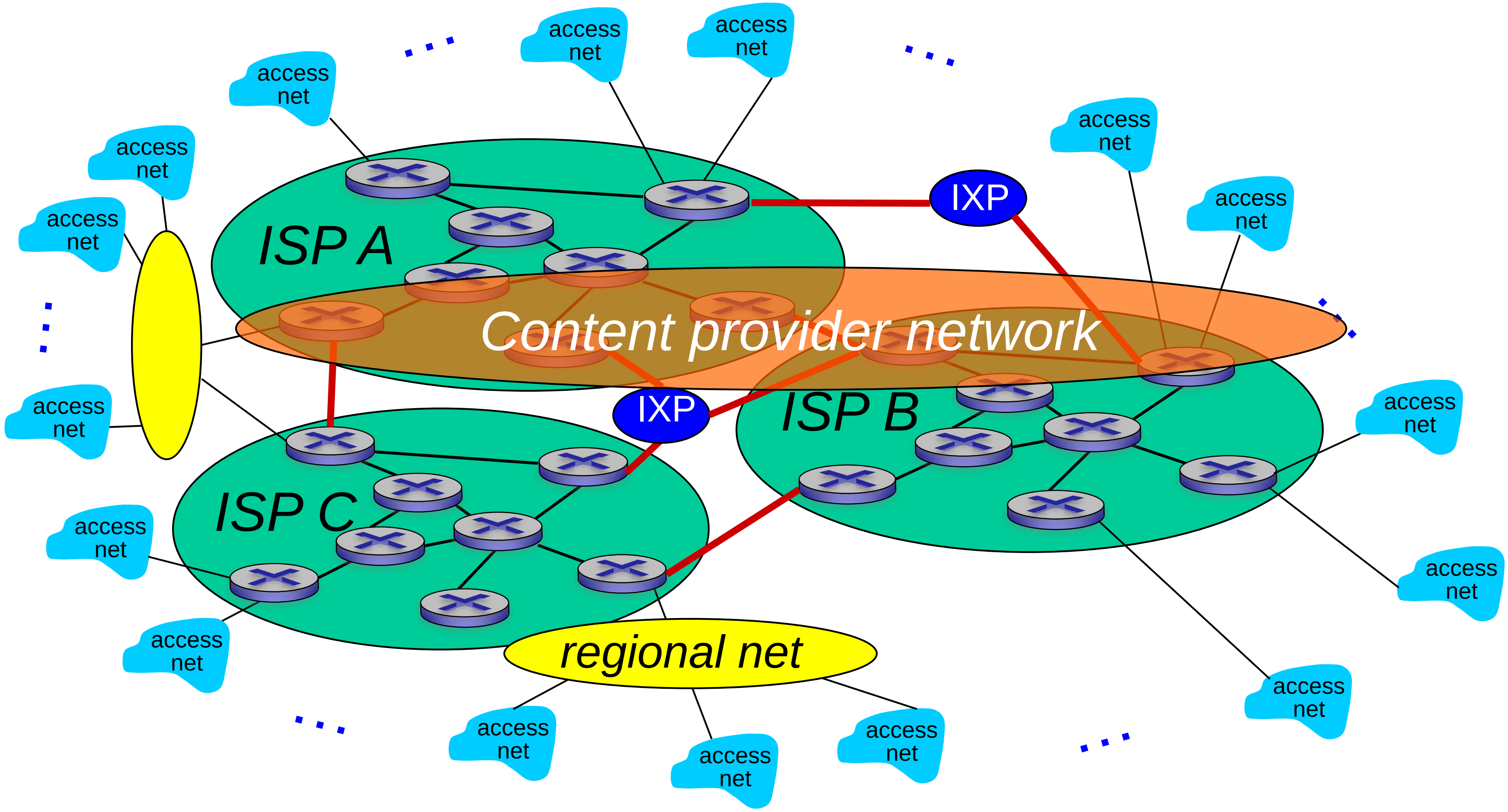
Internet structure: network of networks

Option: connect each access ISP to one global transit ISP?

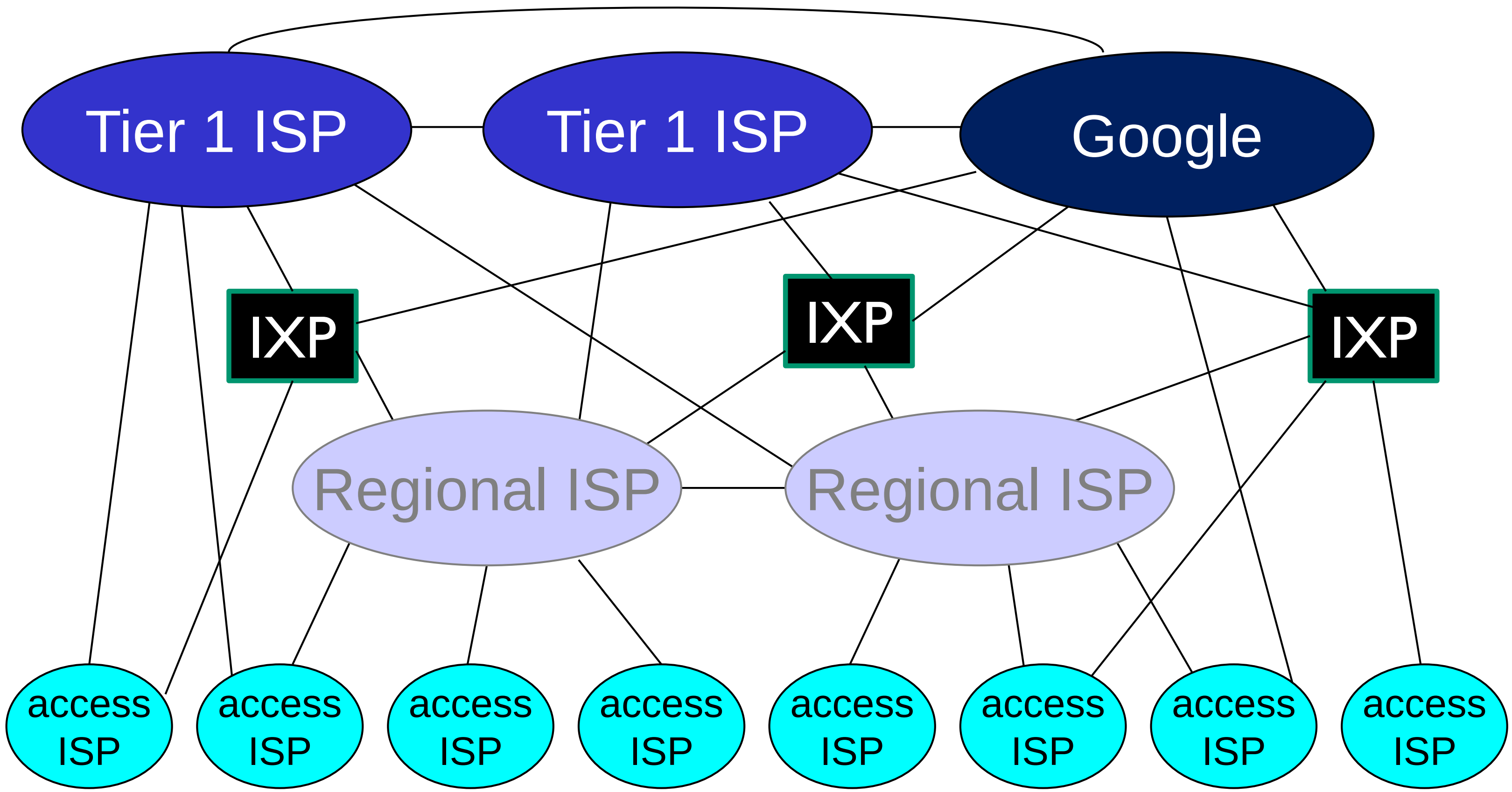


Internet structure: network of networks

… and content provider networks (e.g., Google, Microsoft, Akamai) may run their own network, to bring services, content close to end users

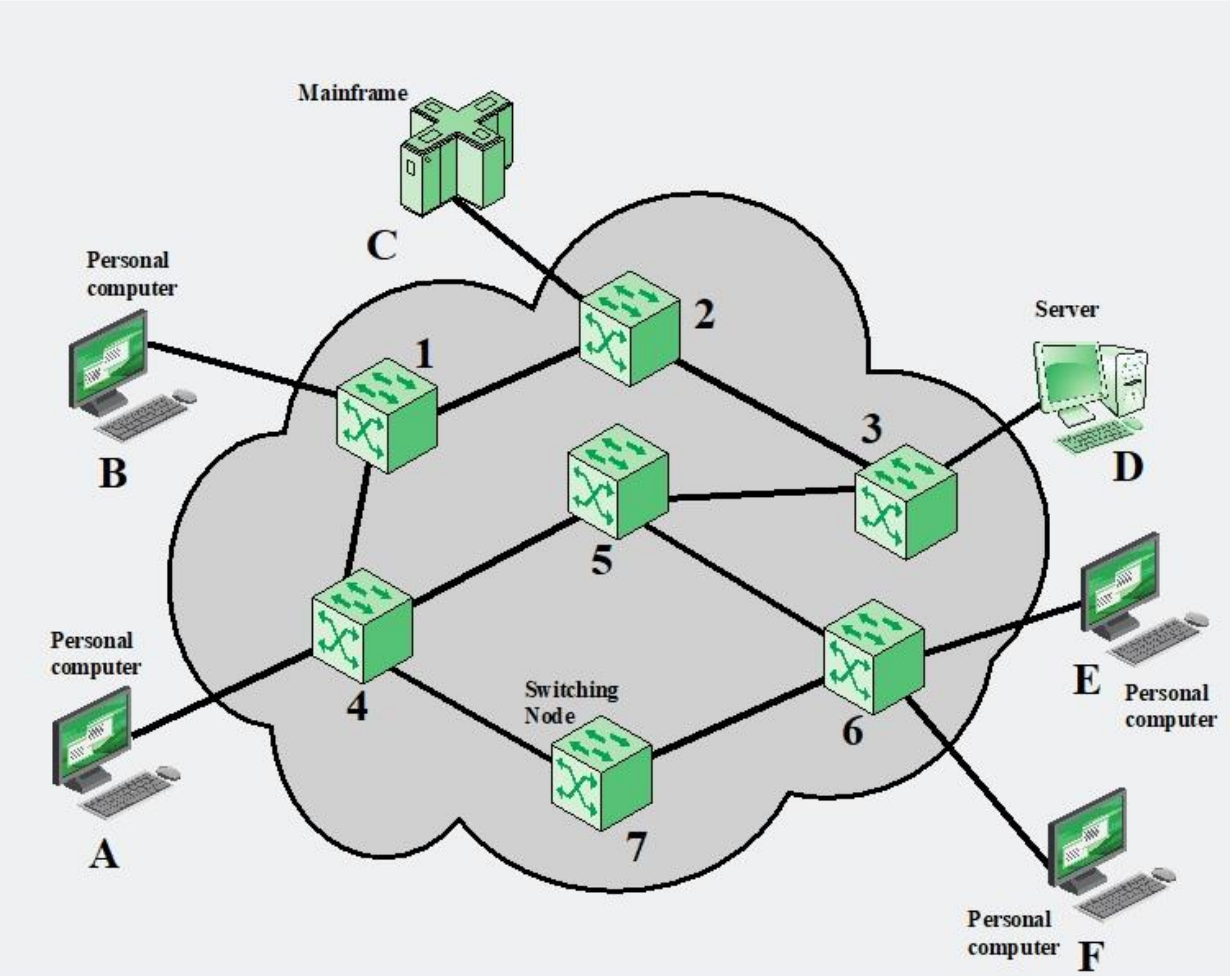


Internet structure: network of networks



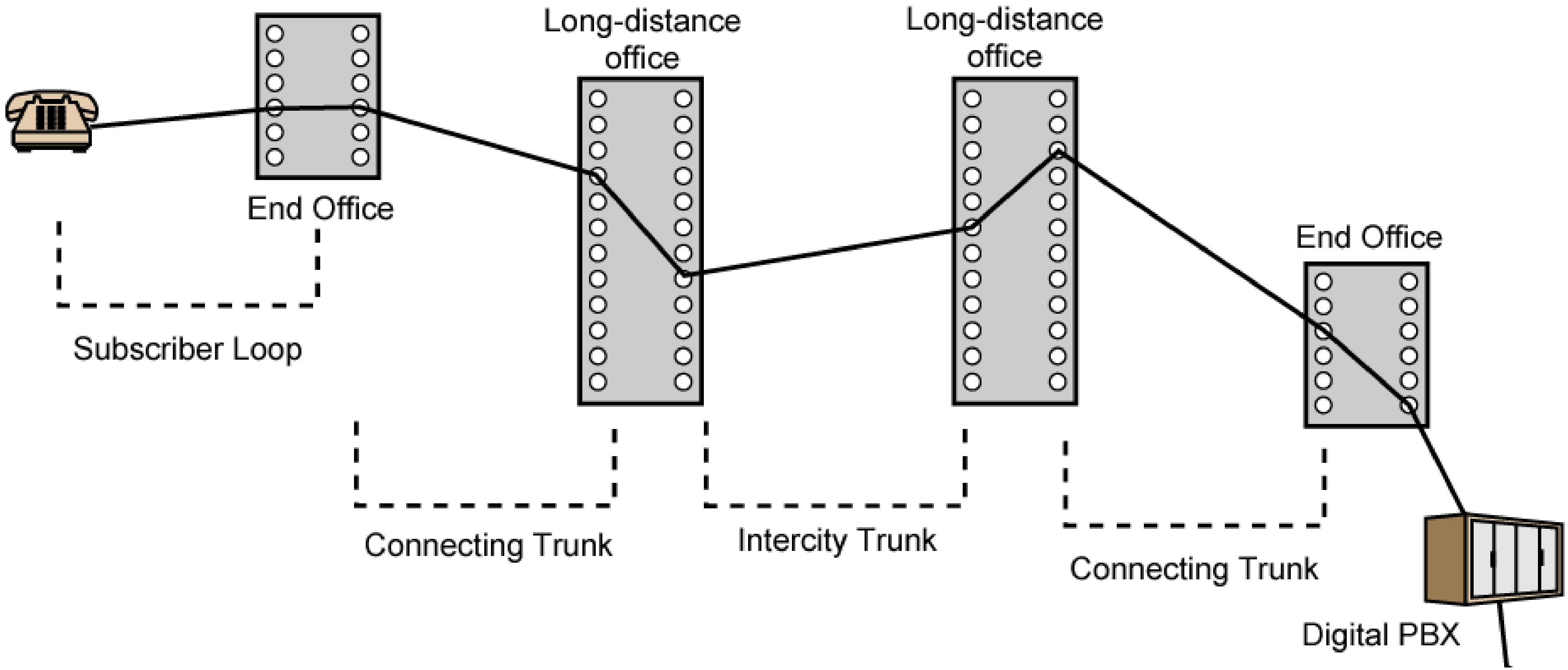
- at center: small # of well-connected large networks
  - "tier-1" commercial ISPs (e.g., Level 3, Sprint, AT&T, NTT), national & international coverage
  - content provider network (e.g., Google): private network that connects it data centers to Internet, often bypassing tier-1, regional ISPs

Switching Network



A – F: Hosts 1 – 7 : Switching nodes

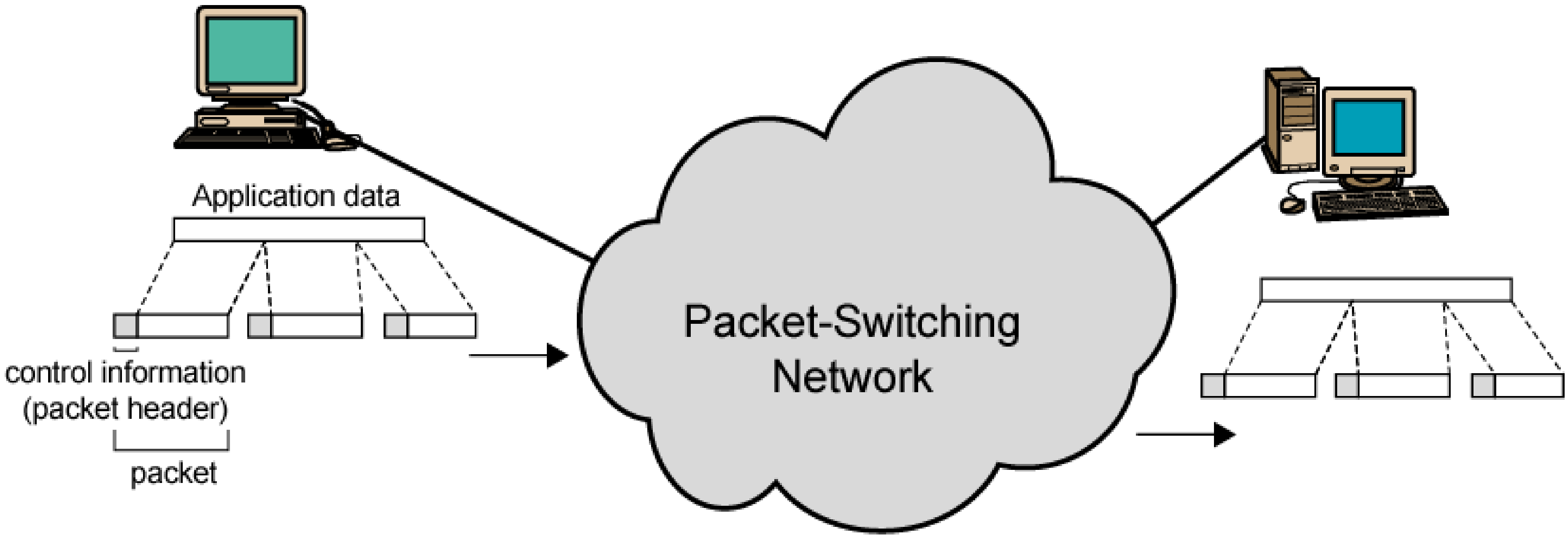
Circuit Switched Telephone Network



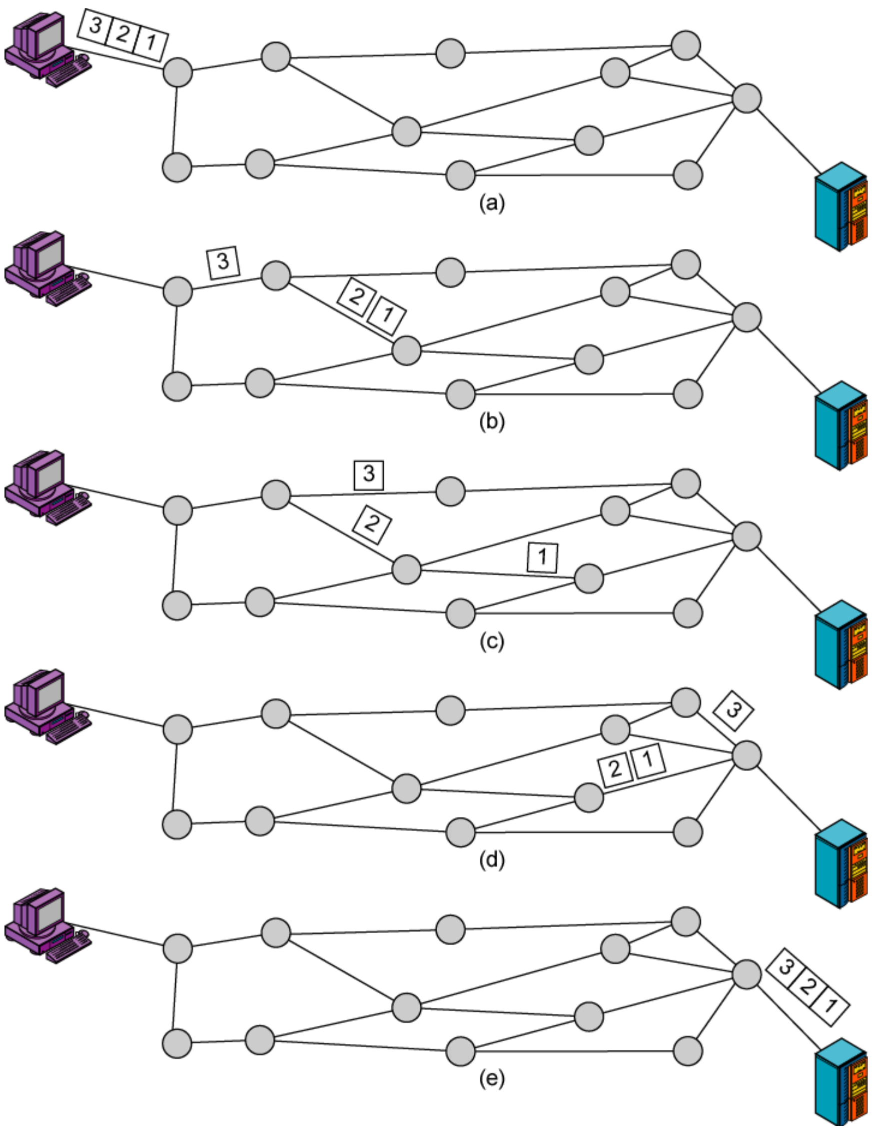
Packet Switching

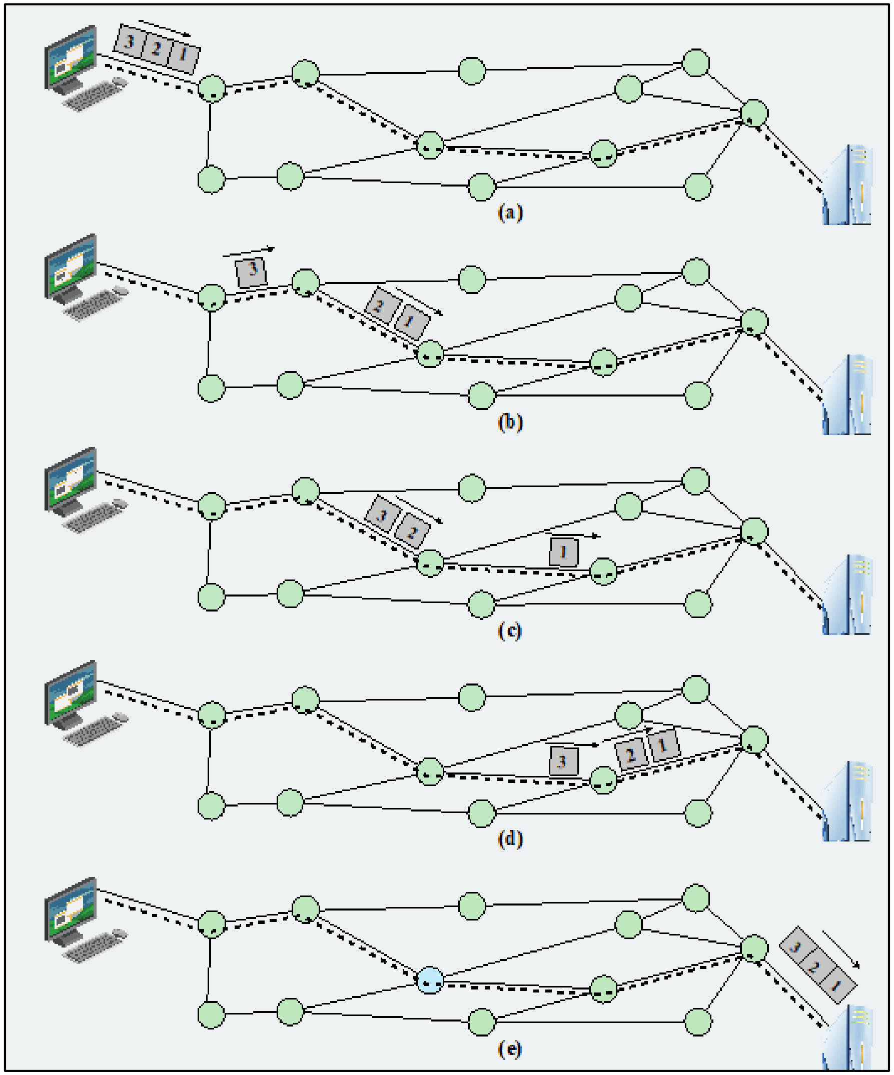
- Data transmitted in small packets
  - Longer messages split into series of packets
  - Each packet contains a portion of user data plus some control info
- Control info
  - Routing (addressing) info
- Packets are received, stored briefly (buffered) and past on to the next node
  - Store and forward

Packet Switching

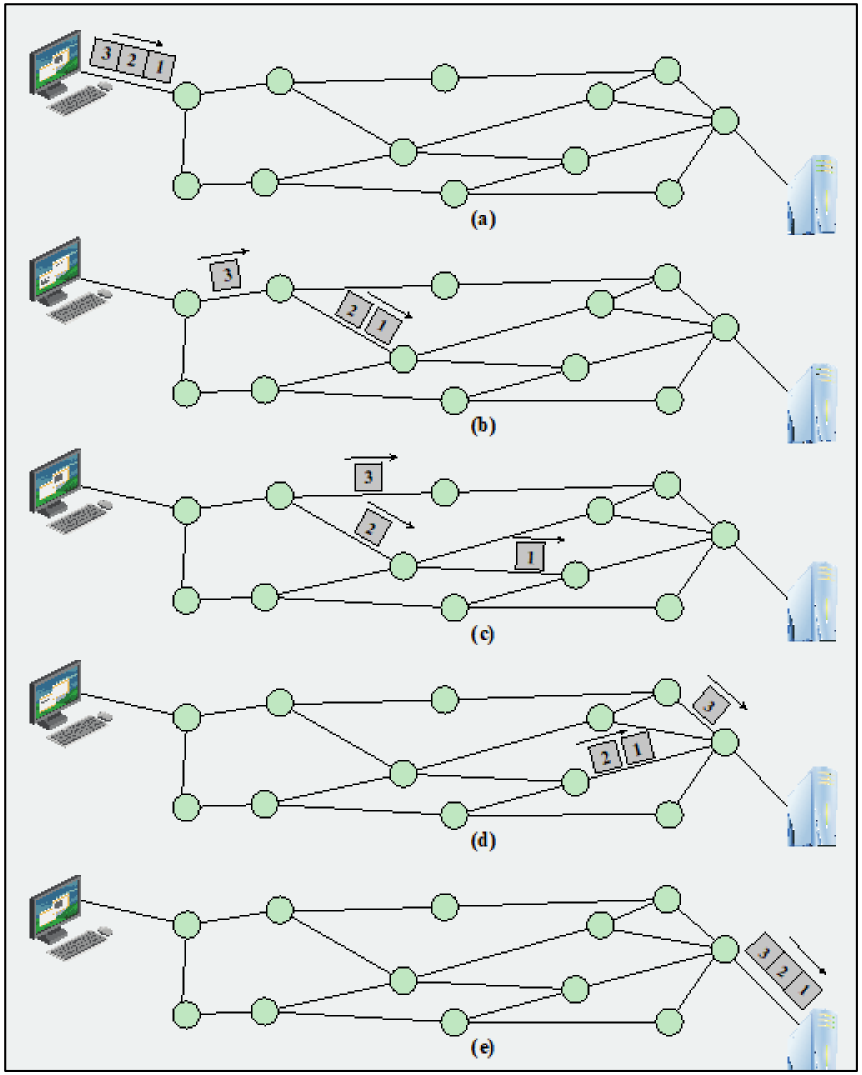


Packet Switching



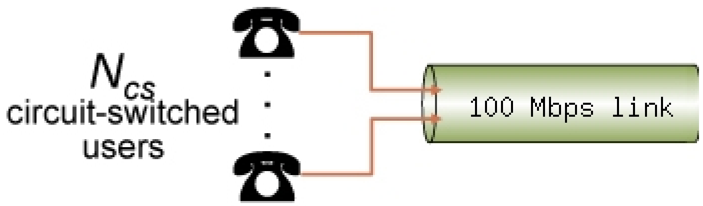


Circuit Switching

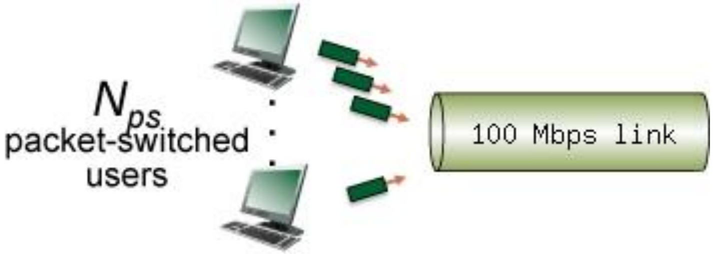


Packet Switching

Quantitative Comparison of Packet Switching and Circuit Switching



a) Circuit Switching



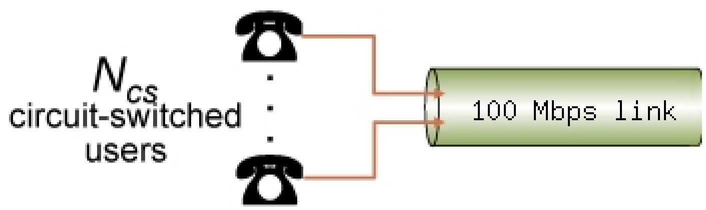
b) Packet Switching

Consider the two scenarios:

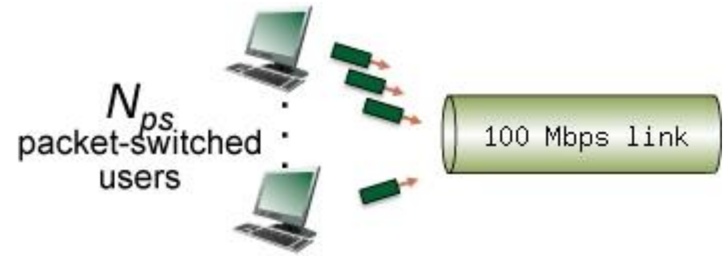
a) A circuit-switching scenario in which a set of users N_{cs}, each requiring a bandwidth of 10 Mbps, must share a link of capacity 100 Mbps.

b) A packet-switching scenario in which a set of users N_{ps} , sharing a 100 Mbps link, where each user again requires 10 Mbps when transmitting, but only needs to transmit 30 percent of the time.

Quantitative Comparison of Packet Switching and Circuit Switching



a) Circuit Switching



b) Packet Switching

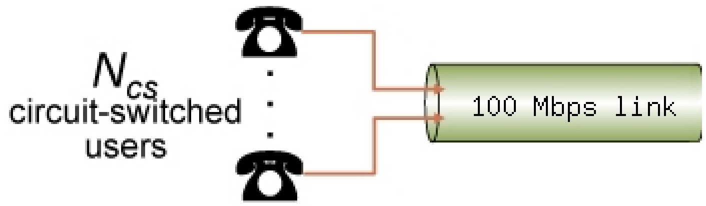
Consider the two scenarios:

a) A circuit-switching scenario in which a set of users N_{cs}, each requiring a bandwidth of 10 Mbps, must share a link of capacity 100 Mbps.

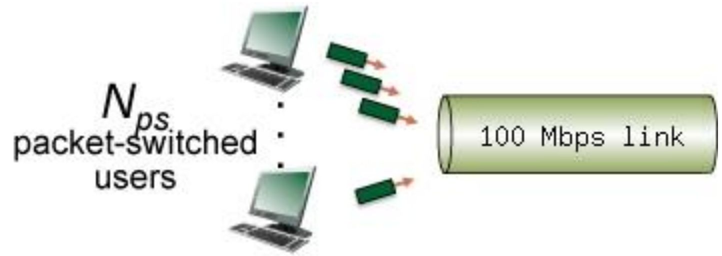
b) A packet-switching scenario in which a set of users N_{ps} , sharing a 100 Mbps link, where each user again requires 10 Mbps when transmitting, but only needs to transmit 30 percent of the time.

Q: When circuit switching is used, what is the maximum number of circuit-switched users that can be supported?

Quantitative Comparison of Packet Switching and Circuit Switching



a) Circuit Switching



b) Packet Switching

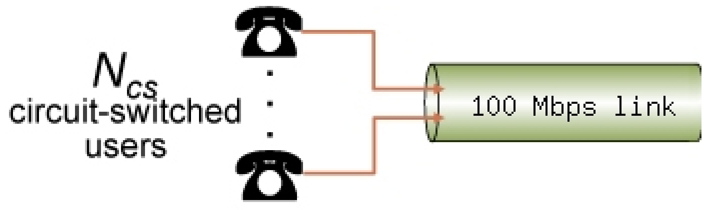
Consider the two scenarios:

a) A circuit-switching scenario in which a set of users N_{cs}, each requiring a bandwidth of 10 Mbps, must share a link of capacity 100 Mbps.

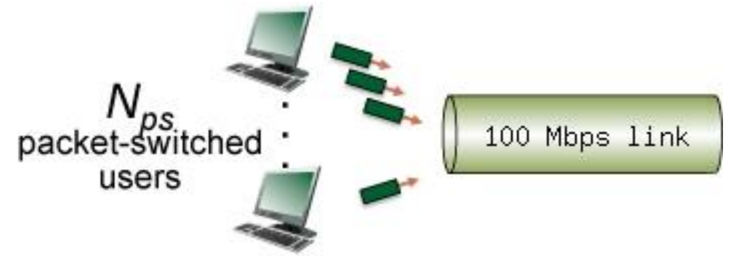
b) A packet-switching scenario in which a set of users N_{ps} , sharing a 100 Mbps link, where each user again requires 10 Mbps when transmitting, but only needs to transmit 30 percent of the time.

Q: Suppose there are 19 packet-switching users (i.e., $|N_{ps}| = 19$). Can this many users be supported under circuit-switching?

Quantitative Comparison of Packet Switching and Circuit Switching



a) Circuit Switching



b) Packet Switching

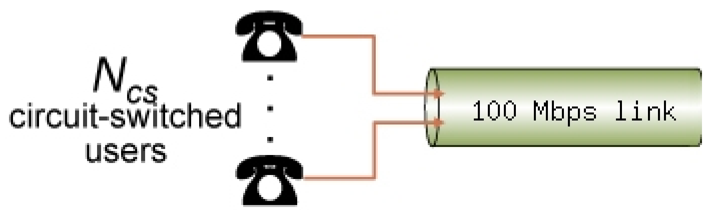
Consider the two scenarios:

a) A circuit-switching scenario in which N_{cs} users, each requiring a bandwidth of 10 Mbps, must share a link of capacity 100 Mbps.

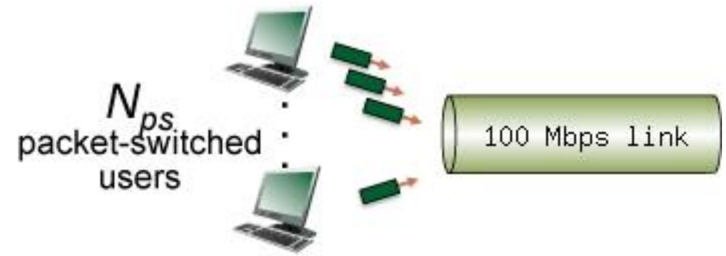
b) A packet-switching scenario with N_{ps} users sharing a 100 Mbps link, where each user again requires 10 Mbps when transmitting, but only needs to transmit 30 percent of the time.

Q: What is the probability that a given (*specific*) user is transmitting, and the remaining users are not transmitting? Assume $N_{ps} = 19$

Quantitative Comparison of Packet Switching and Circuit Switching



a) Circuit Switching



b) Packet Switching

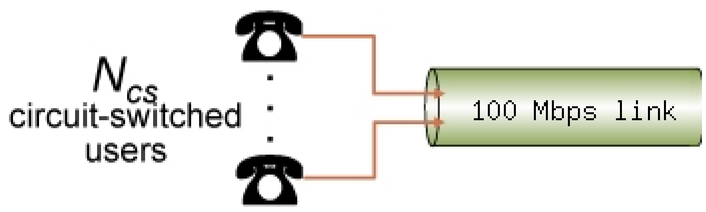
Consider the two scenarios:

a) A circuit-switching scenario in which N_{cs} users, each requiring a bandwidth of 10 Mbps, must share a link of capacity 100 Mbps.

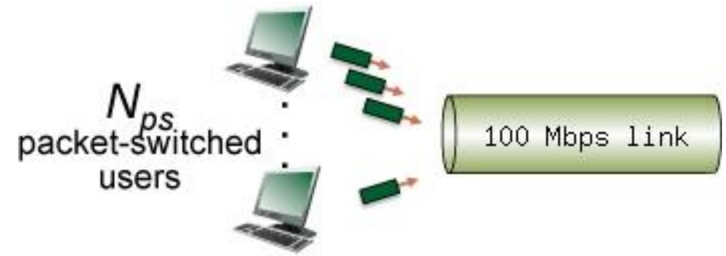
b) A packet-switching scenario with N_{ps} users sharing a 100 Mbps link, where each user again requires 10 Mbps when transmitting, but only needs to transmit 30 percent of the time.

Q: What is the probability that one user (*any* one among the 19 users) is transmitting, and the remaining users are not transmitting? When one user is transmitting, what fraction of the link capacity will be used by this user?

Quantitative Comparison of Packet Switching and Circuit Switching



a) Circuit Switching



b) Packet Switching

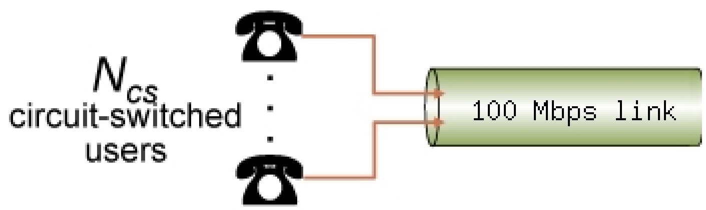
Consider the two scenarios:

a) A circuit-switching scenario in which N_{cs} users, each requiring a bandwidth of 10 Mbps, must share a link of capacity 100 Mbps.

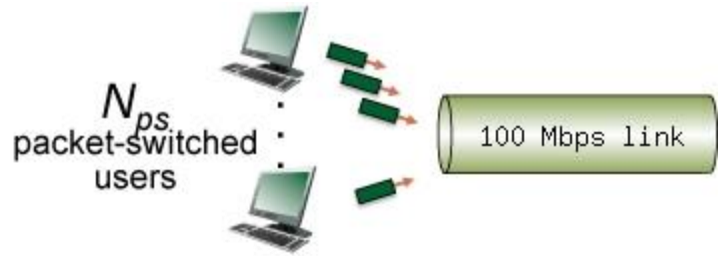
b) A packet-switching scenario with N_{ps} users sharing a 100 Mbps link, where each user again requires 10 Mbps when transmitting, but only needs to transmit 30 percent of the time.

Q: What is the probability that any 10 users (of the total 19 users) are transmitting and the remaining users are not transmitting?

Quantitative Comparison of Packet Switching and Circuit Switching



a) Circuit Switching



b) Packet Switching

Consider the two scenarios:

a) A circuit-switching scenario in which N_{cs} users, each requiring a bandwidth of 10 Mbps, must share a link of capacity 100 Mbps.

b) A packet-switching scenario with N_{ps} users sharing a 100 Mbps link, where each user again requires 10 Mbps when transmitting, but only needs to transmit 30 percent of the time.

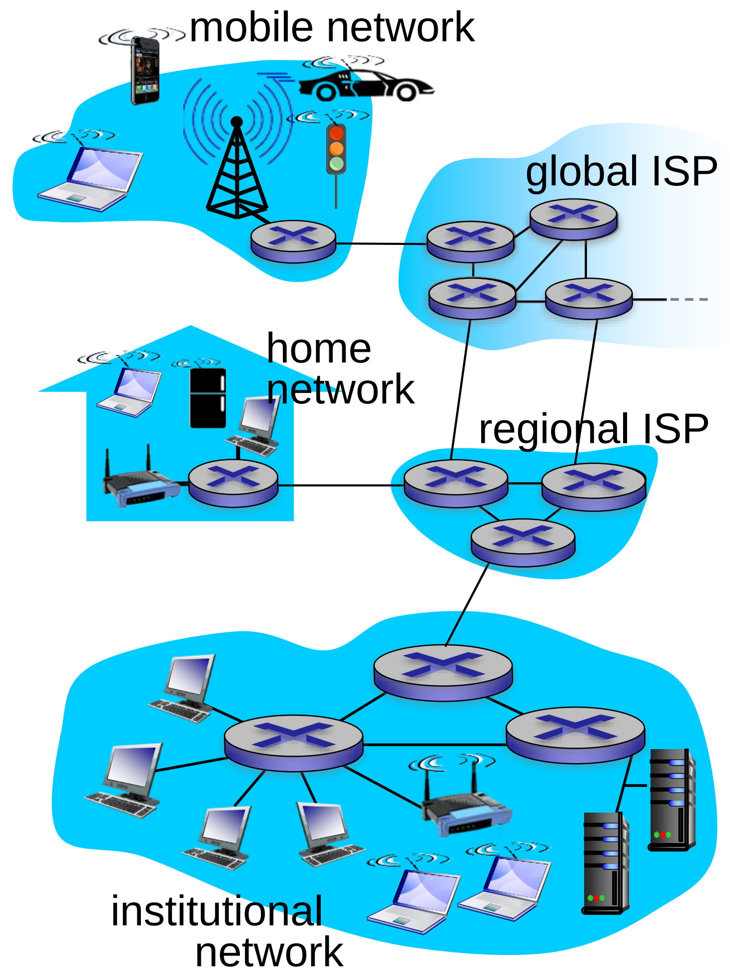
Q: What is the probability that *more* than 10 users are transmitting? Comment on what this implies about the number of users supportable under circuit switching and packet switching.

Packet Switching

- Advantages:
  - Line efficiency
    - Single node to node link can be shared by many packets over time
    - Packets queued and transmitted as fast as possible
  - Data rate conversion
    - Each station connects to the local node at its own speed
    - Nodes buffer data if required to equalize rates
  - Priorities can be used (quality of service)
- Disadvantages:
  - Congestion and packet delay – can lead to deterioration of service
  - Sophisticated protocols required for packet delivery and routing

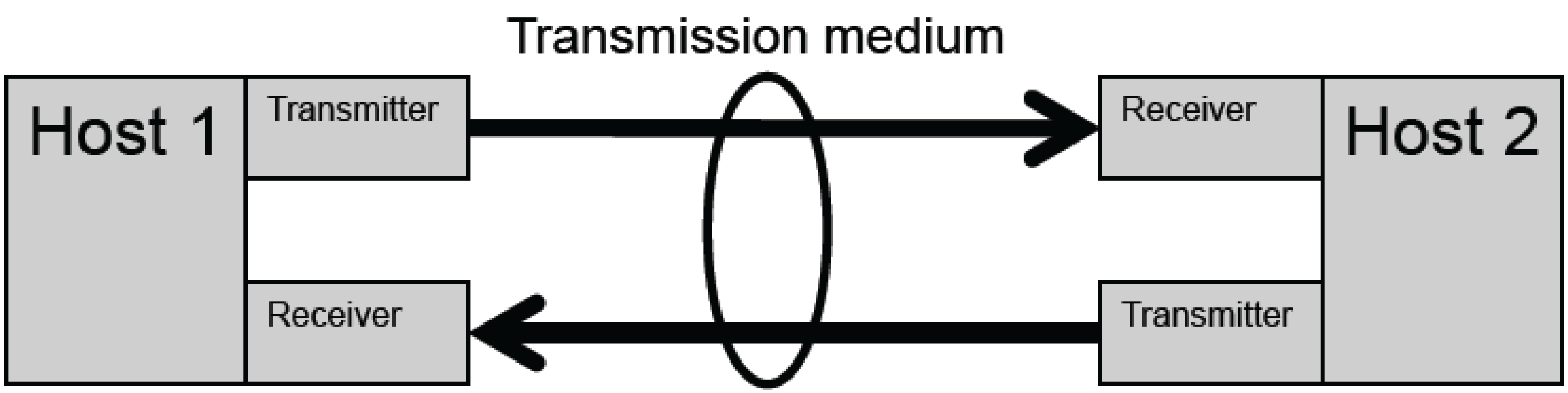
The Internet: "nuts and bolts" view

- *Internet:* "network of networks"
  - Interconnected ISPs
- End systems connect to Internet via access ISPs (Internet Service Providers)
  - residential, company and university ISPs
- Access ISPs in turn must be interconnected.
  - so that any two hosts can send packets to each other
- Resulting network of networks is very complex
  - evolution was driven by economics and national policies



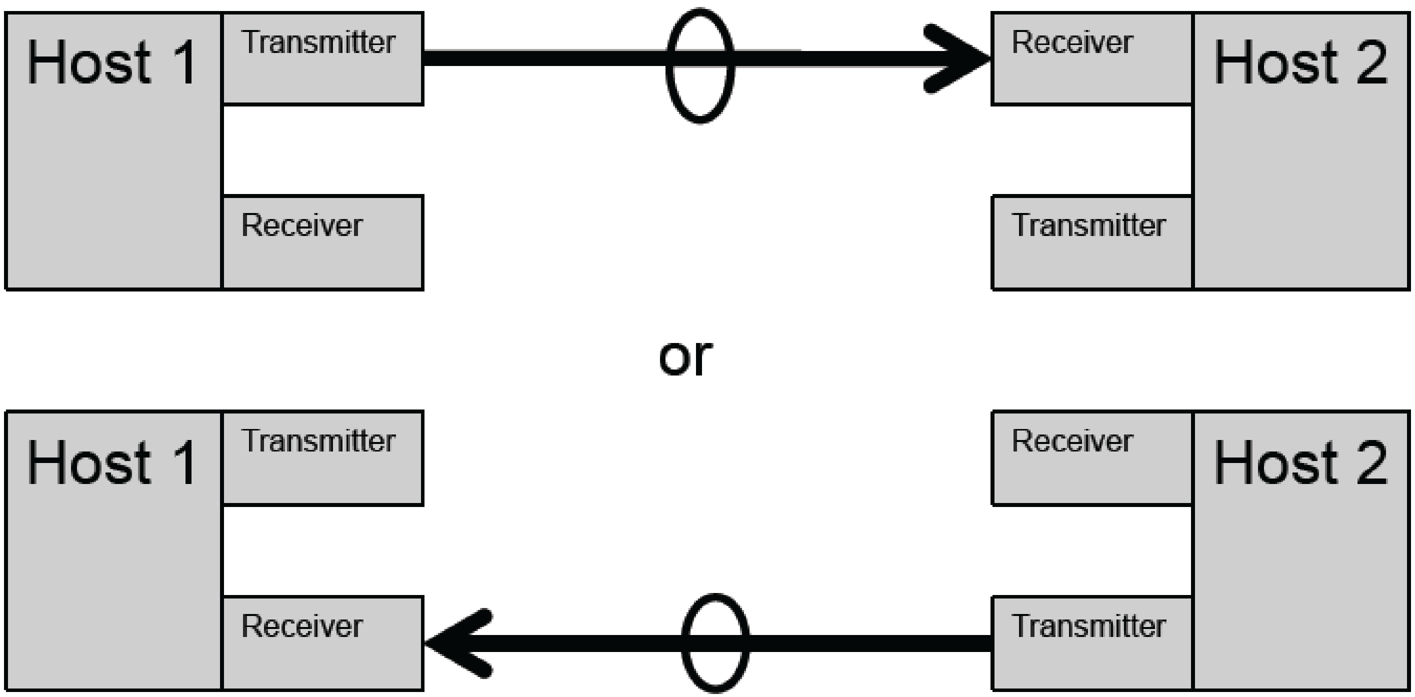
Links

- **Bit** – atomic unit of information
  - 1 or 0
- **Bandwidth** – rate of information communication
  - measured in bits/second
- **Physical link, transmission channel/medium**
  - medium for transmitting bits



Full Duplex

Links cont.



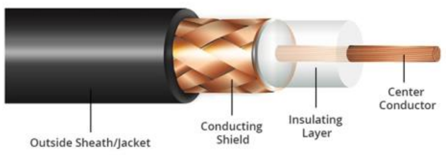
Half Duplex

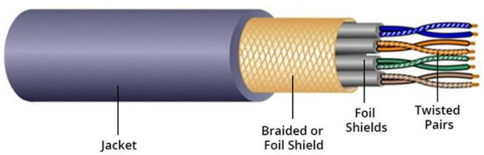
Transmission Media

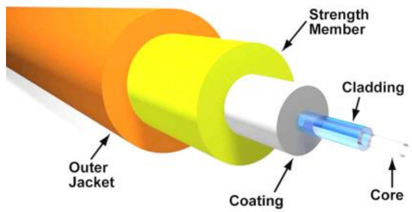
- **Transmission medium** or physical medium
  - A medium in which electromagnetic waves or light waves propagate
  - **Guided** – transmission signals propagate through a solid medium
  - **Unguided** – transmission signals propagate through free space

Guided Transmission Media

- *coaxial cable:*
  - bidirectional
  - multiple channels on cable
- *twisted pair (TP)*
  - two insulated copper wires
    - Category 5: 100 Mbps, 1 Gbps Ethernet
    - Category 6: 10Gbps
- *fiber optic cable:*
  - glass fiber carrying light pulses, each pulse a bit
  - high-speed operation (e.g., 10's-100's GBPS)







Source: https://community.fs.com

Bandwidth and Data Rate

- Bandwidth: The frequency band that carries the signals in a transmission medium.
  - Unit: hertz
  - The bandwidth of the transmitted signal as constrained by the transmitter and the nature of the transmission medium
- Data rate: The rate, in bits per second (bps), at which data can be communicated.
- Channel capacity: the maximum rate at which data can be transmitted over a given communication path, or channel, under given conditions.

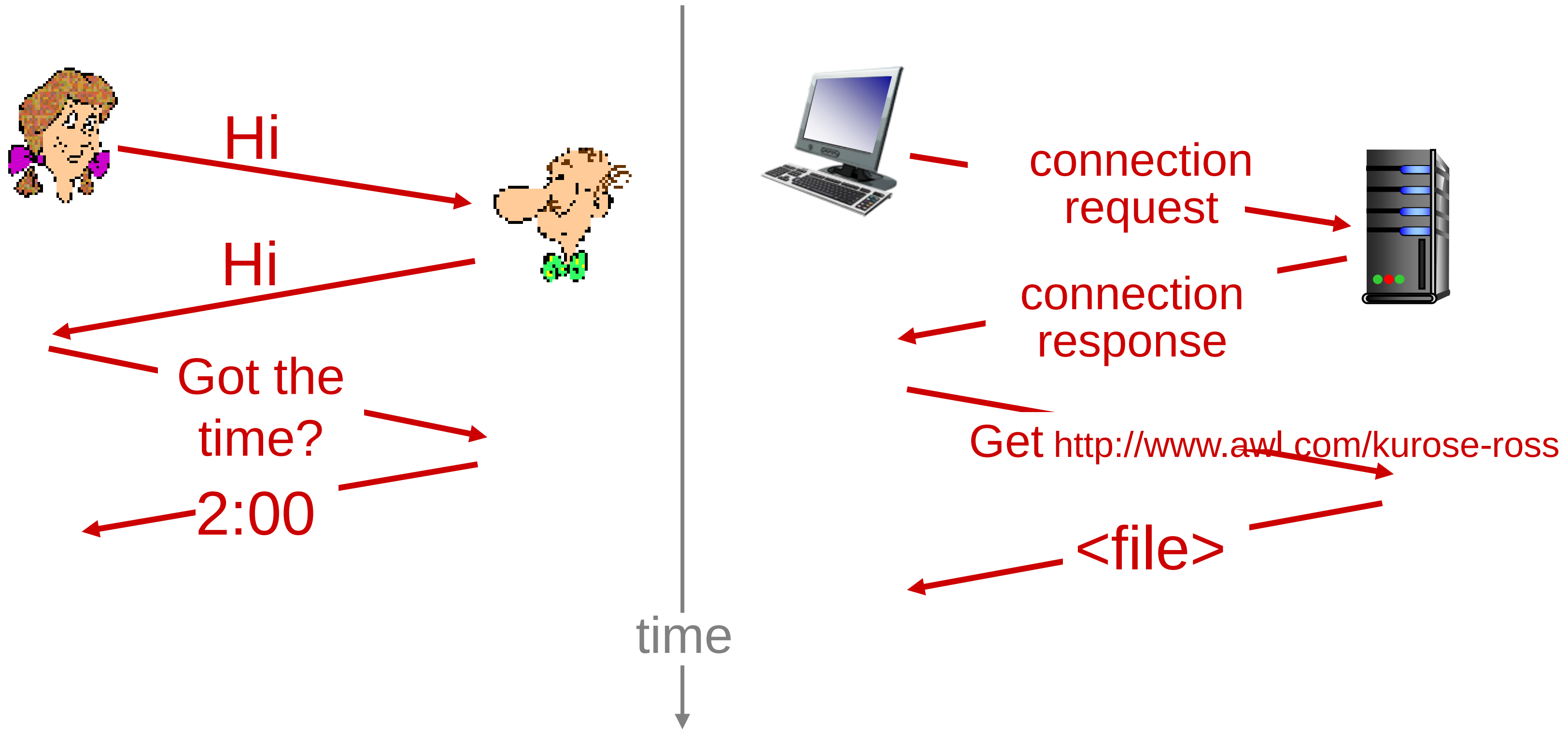
Greater the bandwidth of a transmission system, the higher the data rate that can be transmitted over that system.

Noise and Error Rates

- Received signal will consist of the transmitted signal, modified by the various distortions imposed by the transmission system, plus additional unwanted signals that are inserted somewhere between transmission and reception.
  - The latter, undesired signals are referred to as **noise**.
  - Noise is the major limiting factor in communications system performance.
- Doubling the bandwidth doubles the data rate.
  - Increasing bandwidth is expensive
  - In reality, due to noise the actual data rate is much lower.
- Error rate: The rate at which errors occur, where an error is the reception of a 1 when a 0 was transmitted or the reception of a 0 when a 1 was transmitted

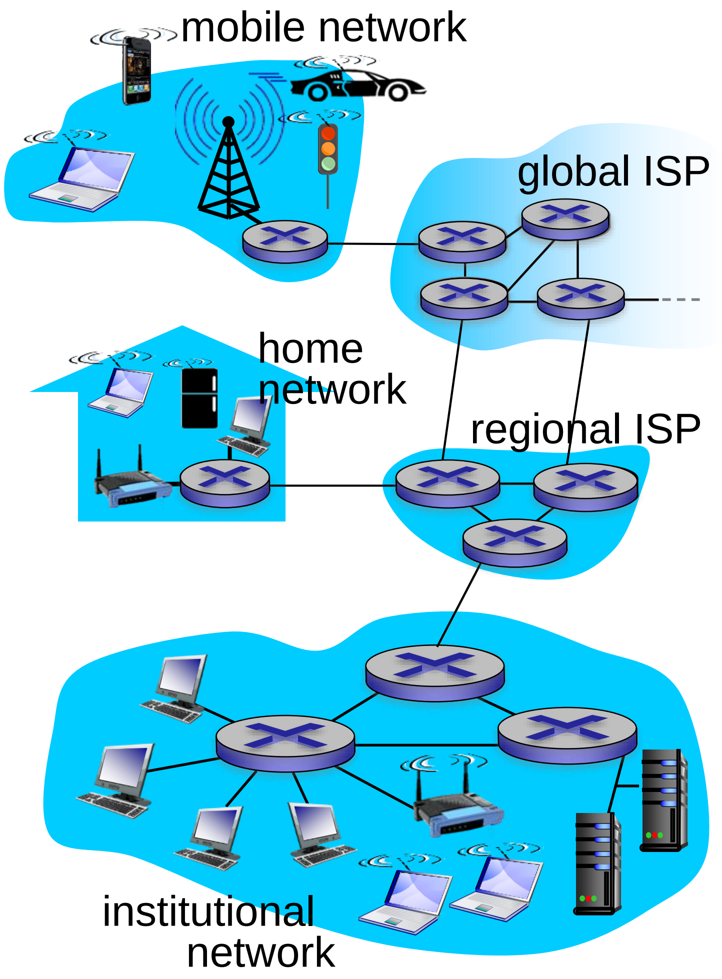
Format of Network Communications: Protocols

a human protocol and a computer network protocol:



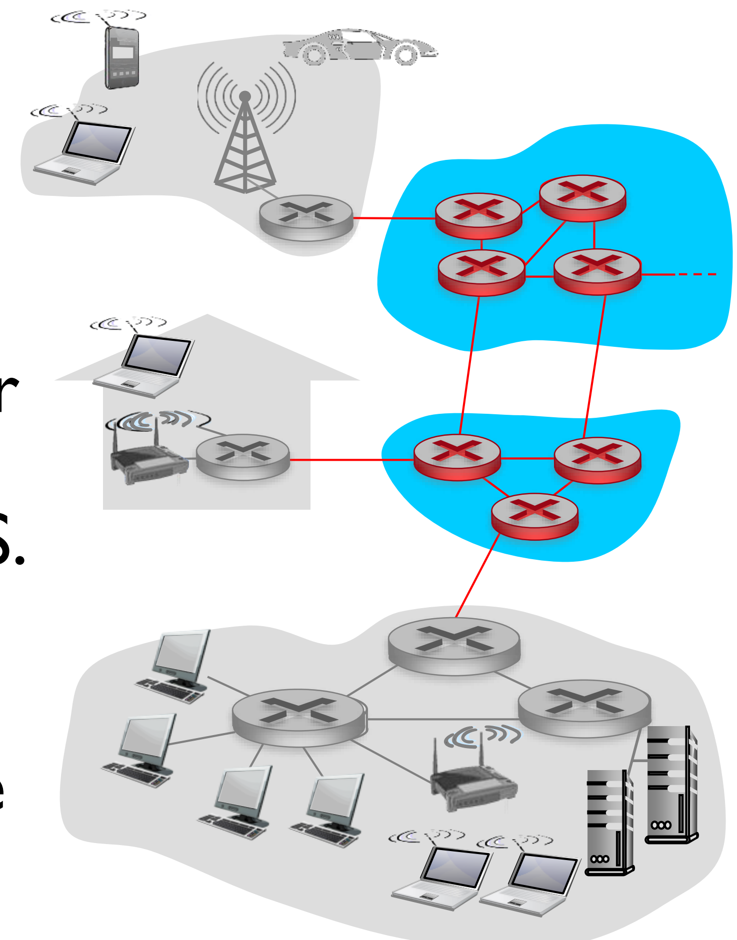
Internet Structure

- *Network edge:*
  - Private networks
  - hosts: clients and servers
  - servers often in data centers
- *Access networks:*
  - wired, wireless communication links
  - Link between Network Edge and Network Core between private and public networks
- *Network core:*
  - Public network
  - interconnected routers



Network core

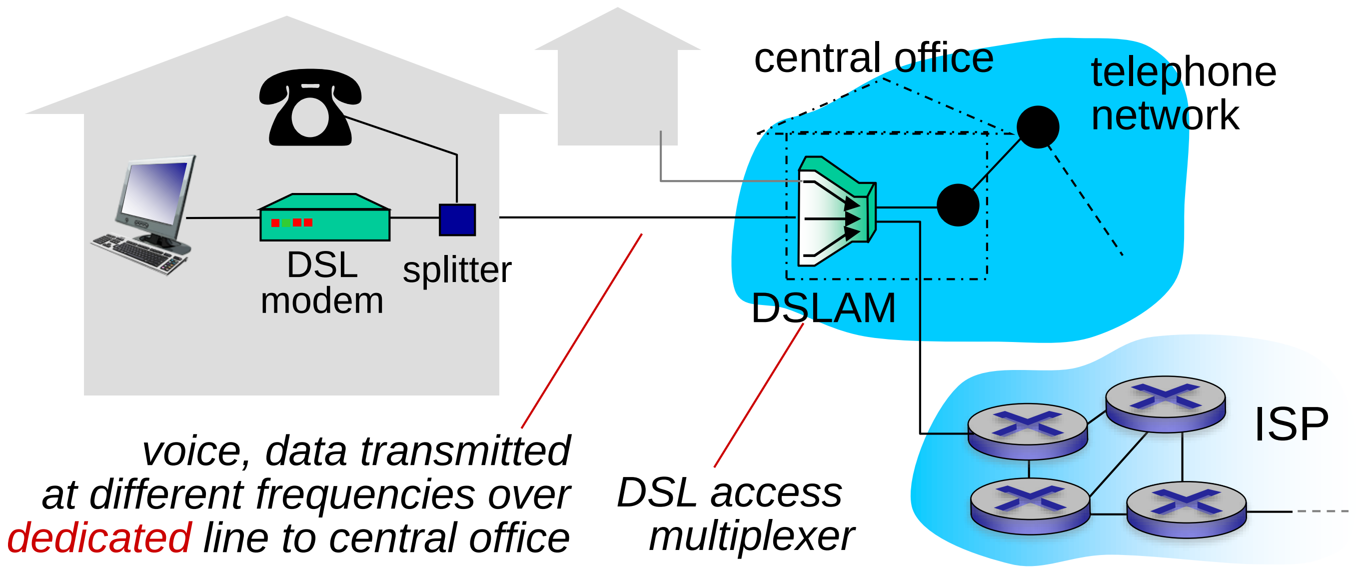
- The network core are the networks built by service providers for **public** consumption
- Primarily connected with fiber optic cables offering high bandwidth up to several TBPS.
- Various individual service provider networks interconnected constitute the network core



Network Edge

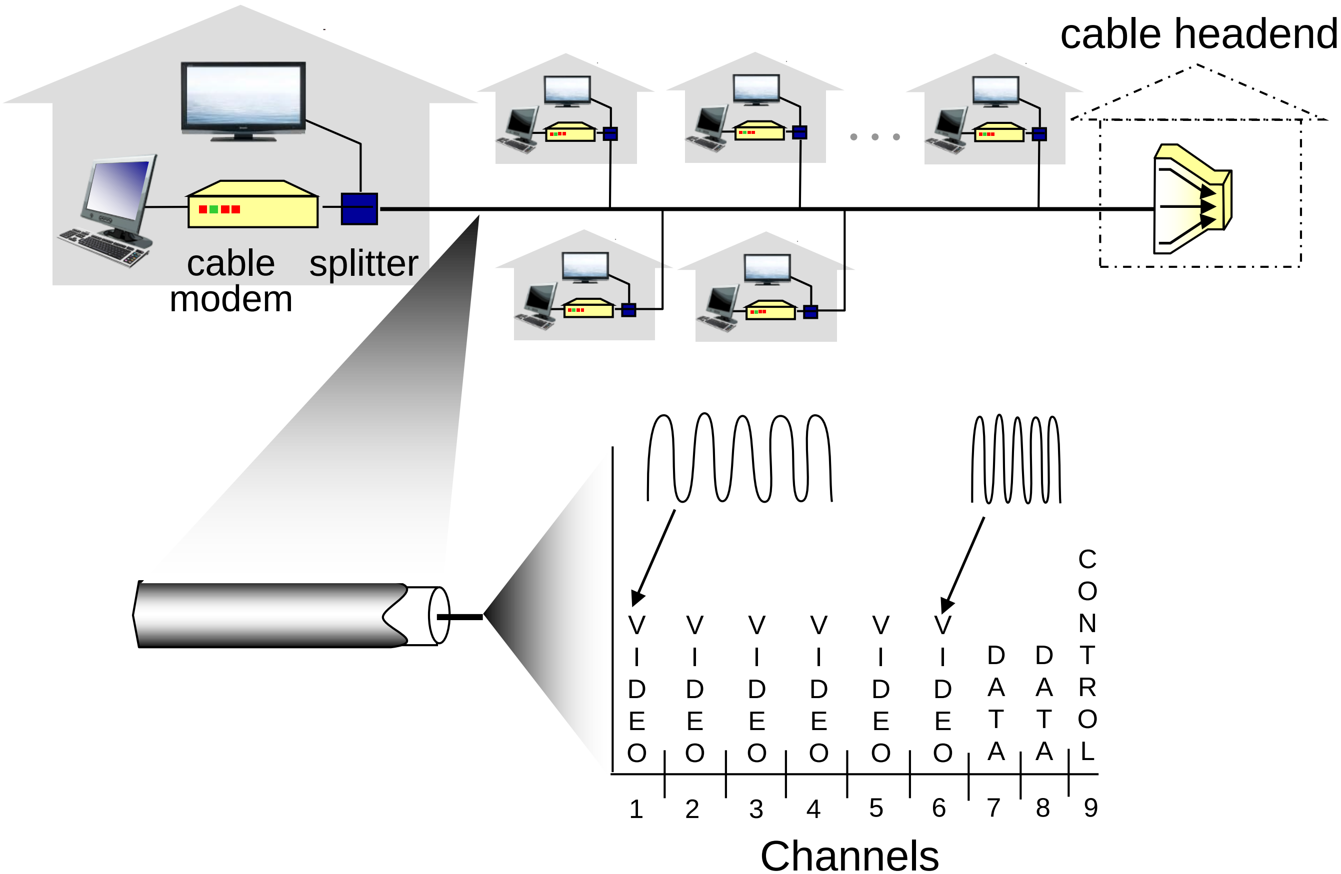
- The network edge are the networks in homes, businesses or institutions built for **private** consumption
- Primarily connected with copper cables carrying high frequency signals
  - up to 1Gbps bandwidth
- Wireless for mobility
- Large institutions may have a fiber optic cabled backbone network

Access network: Digital Subscriber Line (DSL)



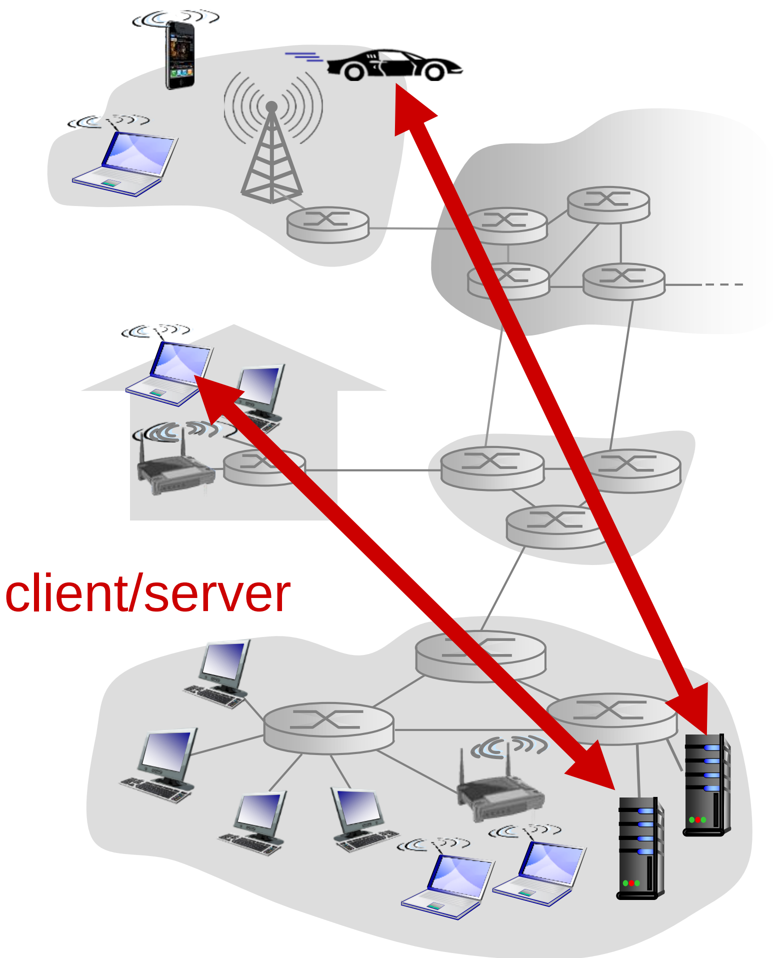
- Use *existing* telephone line
  - data over DSL phone line goes to Internet
  - voice over DSL phone line goes to telephone net
- < 2.5 Mbps upstream transmission rate (typically < 1 Mbps)
- < 24 Mbps downstream transmission rate (typically < 10 Mbps)

Access network: cable network



frequency division multiplexing: different channels transmitted in different frequency bands

Internet Services: Client-server architecture



server:

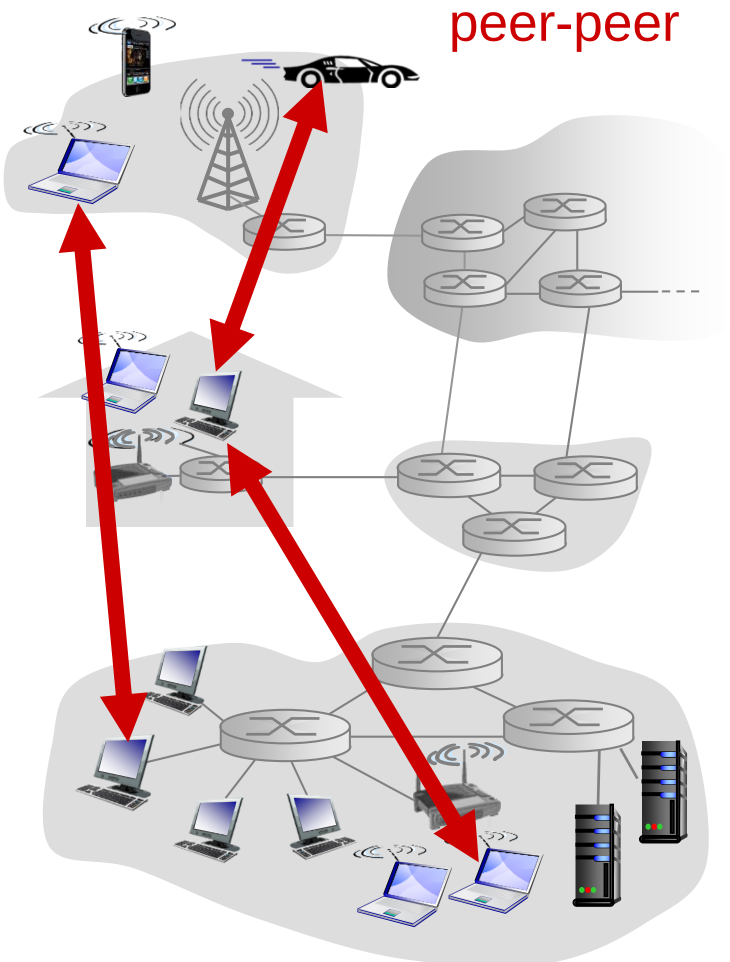
- always-on host
- data centers for scaling

clients:

- communicate with server
- may be intermittently connected
- do not communicate directly with each other

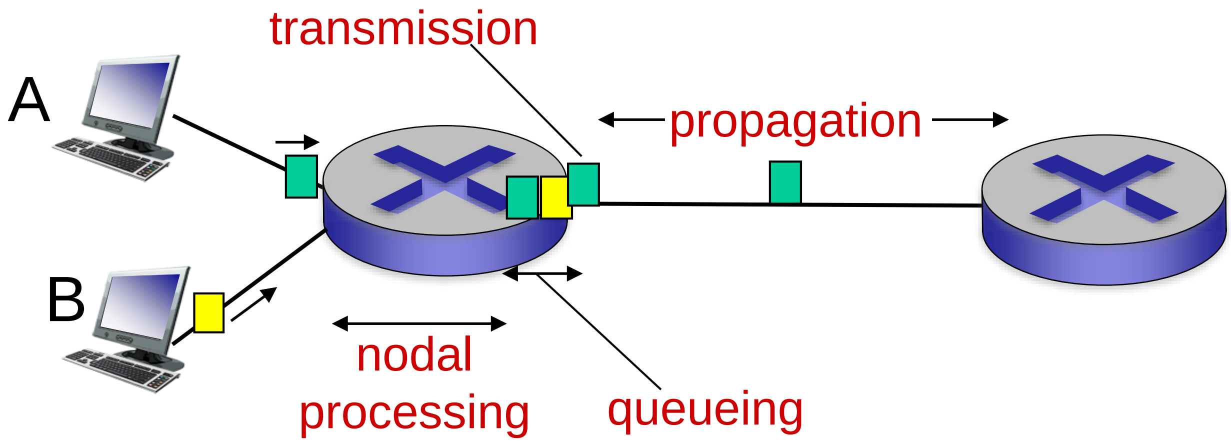
Internet Services: P2P architecture

- *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
- Examples ?



PACKET DELAYS

Four sources of packet delay



$$d_{\text{nodal}} = d_{\text{proc}} + d_{\text{queue}} + d_{\text{trans}} + d_{\text{prop}}$$

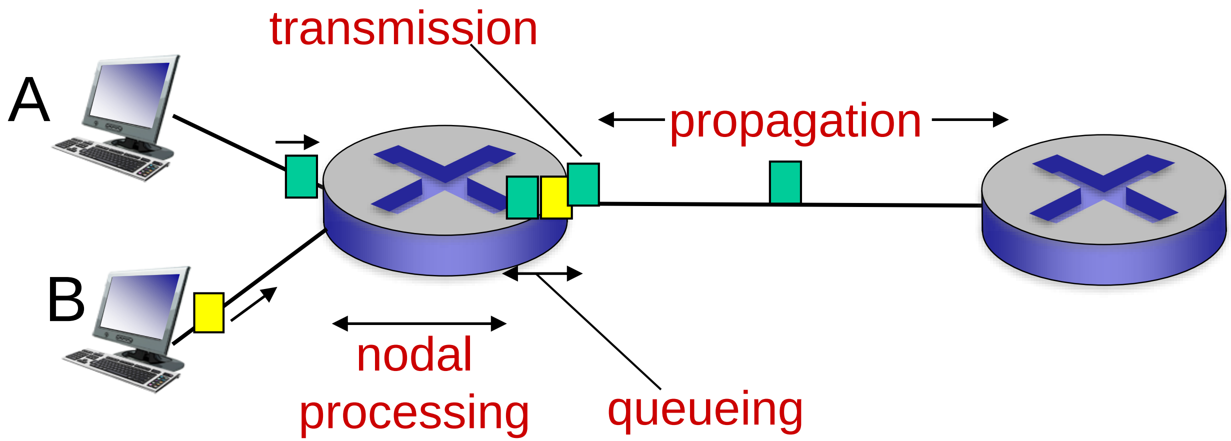
d_{proc}: nodal processing

- check bit errors
- determine output link
- typically < msec

d_{queue}: queueing delay

- time waiting at output link for transmission
- depends on congestion level of router

Four sources of packet delay



$$d_{nodal} = d_{proc} + d_{queue} + d_{trans} + d_{prop}$$

d_{trans}: transmission delay:

- L: packet length (bits)
- R: link *bandwidth (bps)*
- $d_{trans} = L/R$

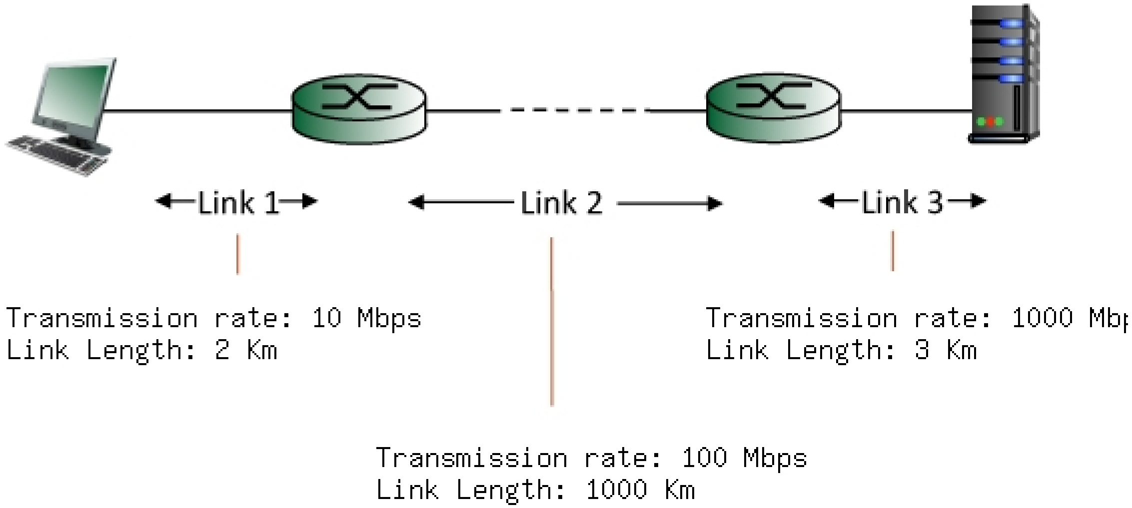
d_{trans} and d_{prop} *very* different

d_{prop}: propagation delay:

- d: length of physical link
- s: propagation speed (~2×10^8 m/sec)
- $d_{prop} = d/s$

- Check out the Java applet for an interactive animation on trans vs. prop delay here

Problem



- Find the end-to-end delay (including the transmission delays and propagation delays on each of the three links, but ignoring queueing delays and processing delays) from when the left host begins transmitting the first bit of a packet to the time when the last bit of that packet is received at the server at the right. The speed of light propagation delay on each link is 3x10**8 m/sec. Note that the transmission rates are in Mbps and the link distances are in Km. Assume a packet length of **12000** bits. Give your answer in milliseconds.

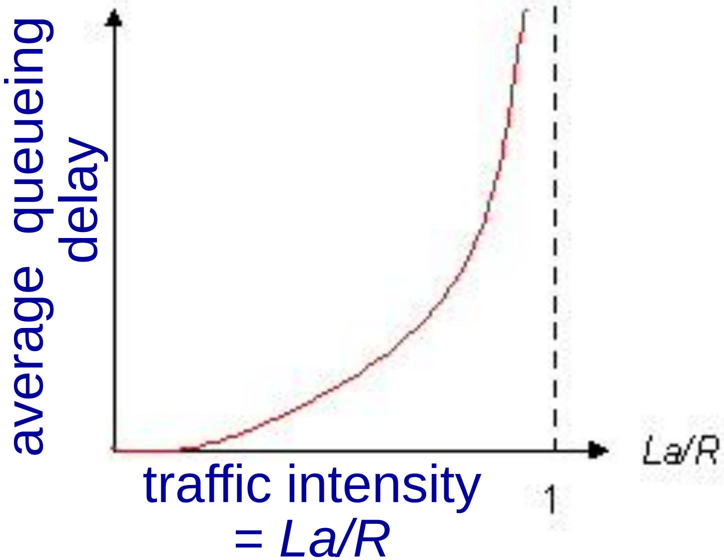
Queueing delay (revisited)

- *R:* link bandwidth (bps)
- *L:* packet length (bits)
- a: average packet arrival rate

- *What happens when*
  - *La/R* ~ 0
  - *La/R* → 1
  - *La/R* > 1

Queueing delay (revisited)

- *R:* link bandwidth (bps)
- *L:* packet length (bits)
- a: average packet arrival rate



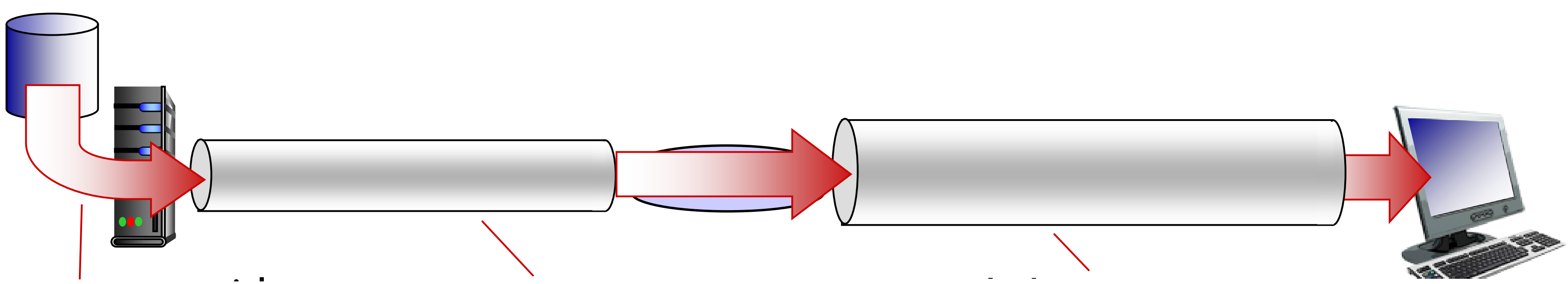
- *La/R* ~ 0: avg. queueing delay small
- *La/R* → 1: avg. queueing delay gradually grows large
- *La/R* > 1: more "work" arriving than can be serviced, average delay infinite!


La/R ~ 0


La/R -> 1

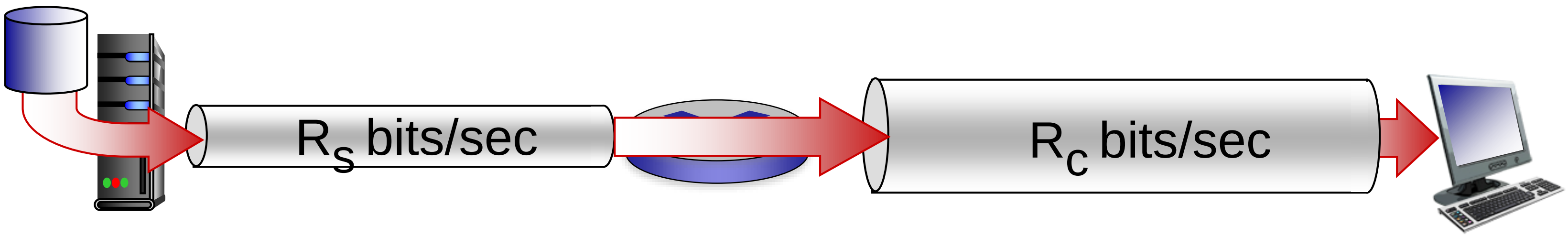
Throughput

- *Throughput:* rate (bits/time unit) at which bits transferred between sender/receiver
  - *instantaneous:* rate at given point in time
  - *average:* rate over longer period of time

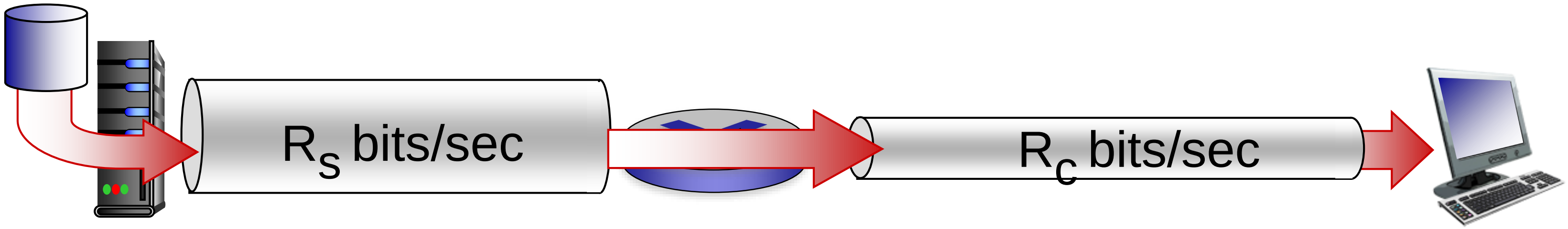


Throughput (more)

- $R_s < R_c$ What is average end-end throughput?



- $R_s > R_c$ What is average end-end throughput?



bottleneck link

link on end-end path that constrains end-end throughput

Key Features of a Protocol

A protocol is a set of rules or conventions that allow peer layers to communicate

The key features of a protocol are:

Syntax	• Format of data blocks
Semantics	• Control information for coordination and error handling
Timing	• Flow control and sequencing

Protocol "layers"

Networks are complex, with many "pieces":

- hosts
- routers
- links of various media
- applications
- protocols
- hardware, software

Question:

- How do we make it work?

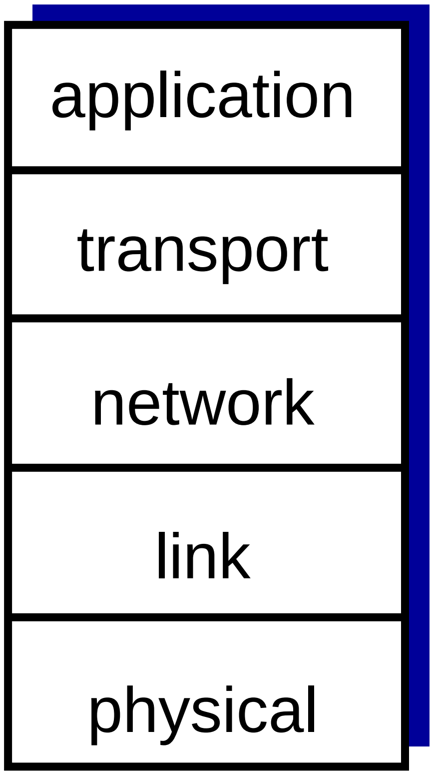
- We will employ a divide and conquer approach
- How do we divide the complexity?

Why layering?

dealing with complex systems:

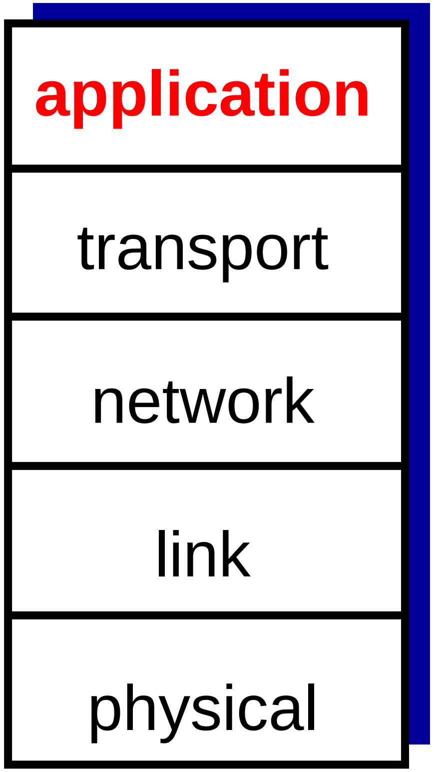
- explicit structure allows identification, relationship of complex system's pieces
  - layered *reference model* for discussion
- modularization eases maintenance, updating of system
  - change of implementation of layer's service transparent to rest of system
  - e.g., change in gate procedure doesn't affect rest of system
- layering considered harmful?

Internet protocol stack



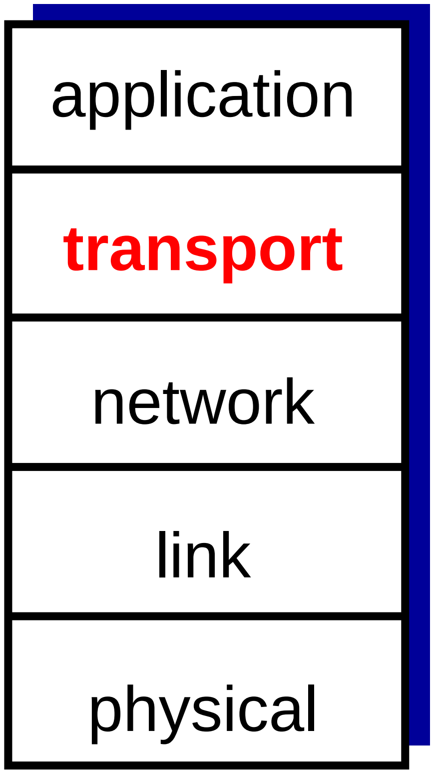
Application Layer

- *Responsibilities*
  - Exchange information between hosts
  - Data unit is called **message**
- Examples
  - HTTP
  - DNS
  - SMTP
  - FTP
  - BitTorrent



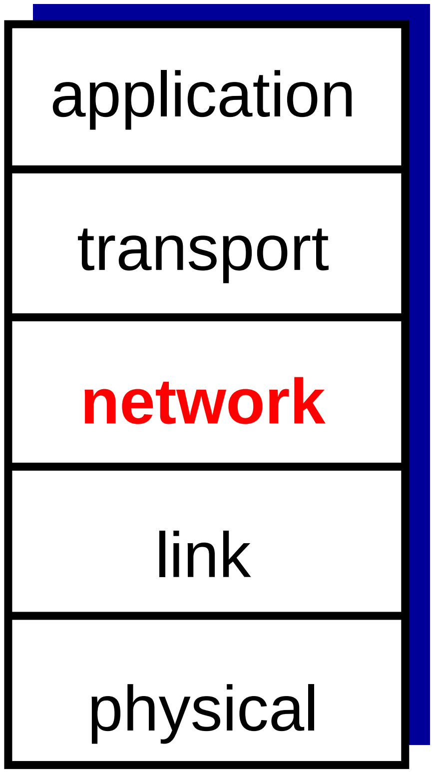
Transport Layer

- *Responsibilities*
  - Exchange packets between hosts
  - Error recovery
  - Congestion and flow control
  - Data unit is called **segment**
- Examples
  - Transmission Control Protocol (TCP)
  - User Datagram Protocol (UDP)



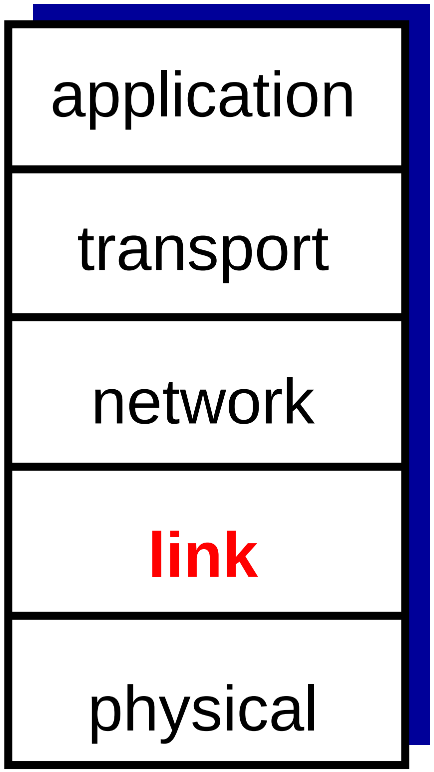
Network Layer

- *Responsibilities*
  - Route packets from source host to destination host
  - Data unit is called **datagram**
- Examples
  - Internet Protocol (IP)
  - Routing table update: Routing Information Protocol (RIP), Border Gateway Protocol (BGP)



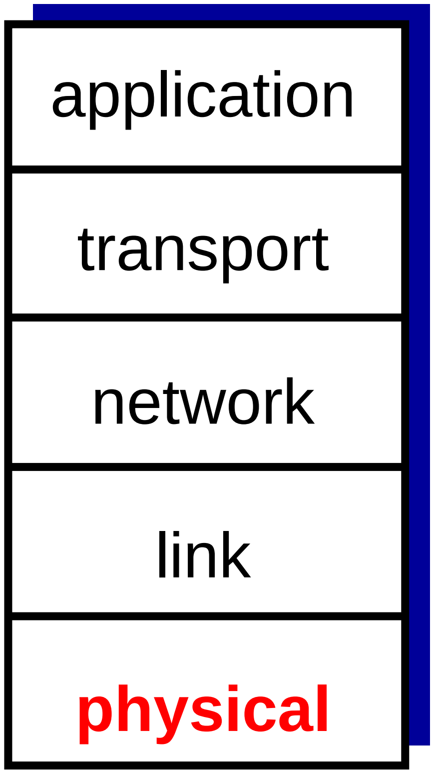
Link Layer

- *Responsibilities*
  - Deliver packets from one end of a transmission channel to the other (i.e., between two nodes [host or switching device])
  - Error recovery, flow control
  - Coordinate transmission channel sharing
  - Data unit is called **frame**
- Examples
  - Ethernet
  - WiFi

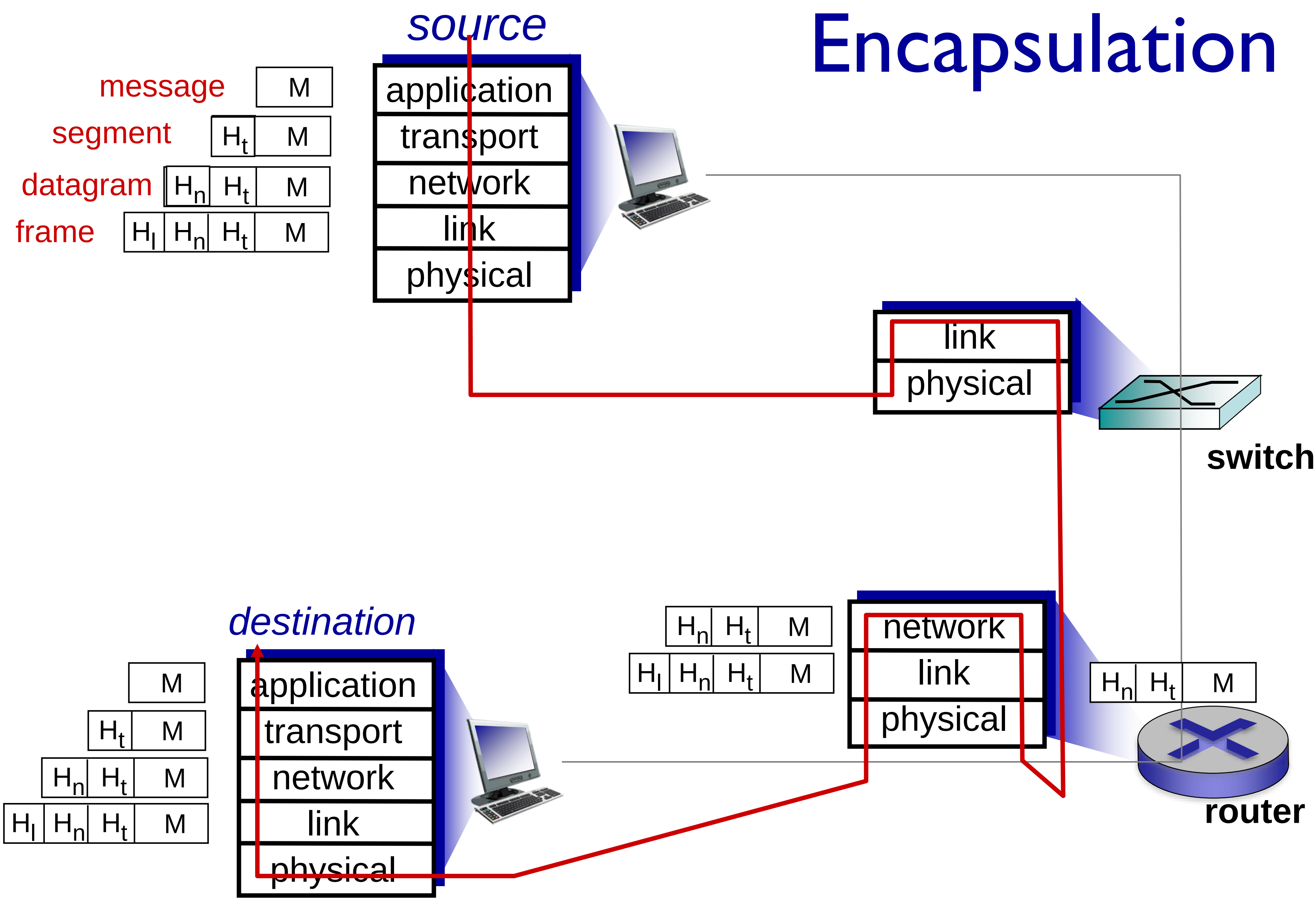


Physical Layer

- *Responsibilities*
- Define electrical or optical signals that represent bit sequences
- Data units are **bits**
- Examples
  - Cable Modem
  - ON-OFF Keying for fiber optics



Encapsulation



Addressing processes

- *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

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, 100s msec
stored audio/video	loss-tolerant	same as above	Yes: few secs
interactive games	loss-tolerant	few kbps – 10kbps	Yes: 100s msec
text messaging	no loss	elastic	Yes and no

Internet Transport Protocols

Transport Control Protocol (TCP) service:

- *Connection-oriented*
  - Setup required between client and server processes
- *Reliable transport* between sending and receiving process
  - Deal with packet drop
- *Flow control*
  - Change rate of sending packets so that receiver is not overwhelmed
- *Does not provide* security

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, or connection setup,

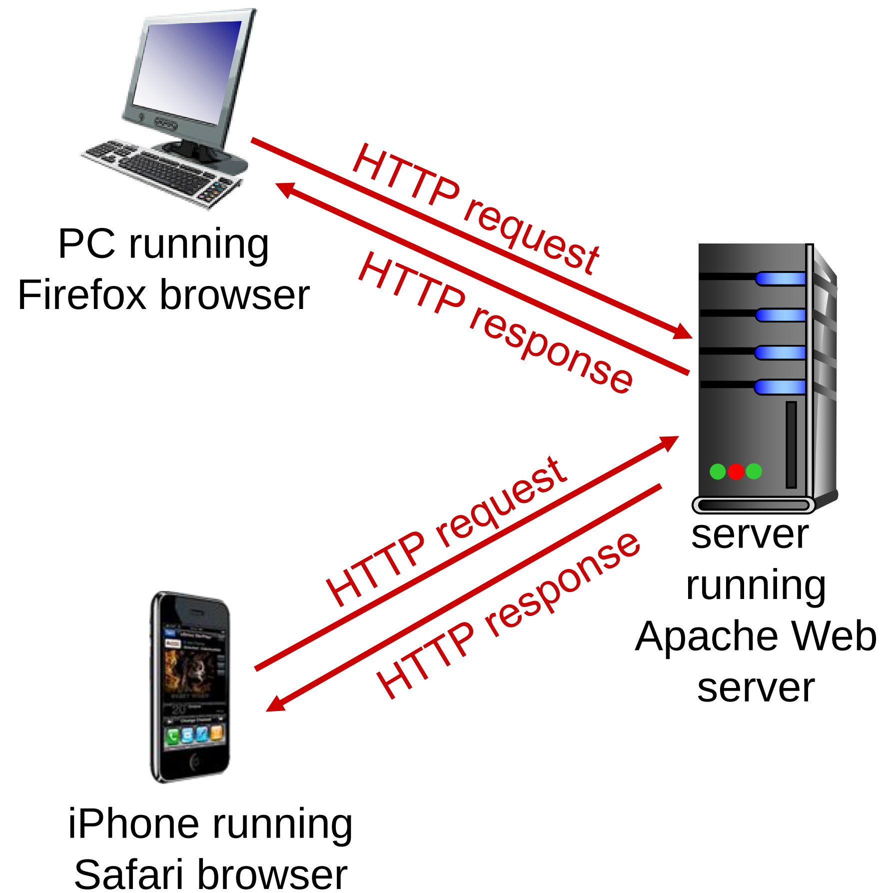
Q: why bother? Why is there a UDP?

APPLICATION LAYER

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 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

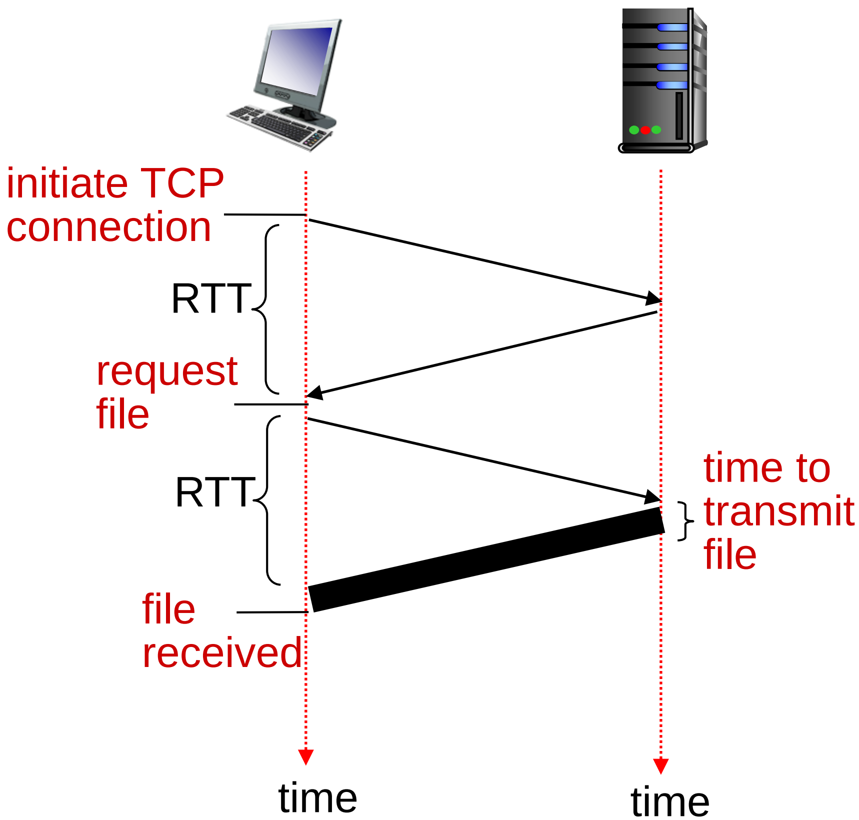
Non-persistent HTTP: response time

RTT (definition): time for a small packet to travel from client to server and back

HTTP response time:

- one RTT to initiate TCP connection
- one RTT for HTTP request and 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 issues:

- requires 2 RTTs 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 connection
- client sends requests as soon as it encounters a referenced object
- as little as one RTT for all the referenced objects

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 request message

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

carriage return character

line-feed character

request line (GET, POST, HEAD commands)

header lines

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

```
GET /index.html HTTP/1.1\r\n
Host: www-net.cs.umass.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
```

Method types

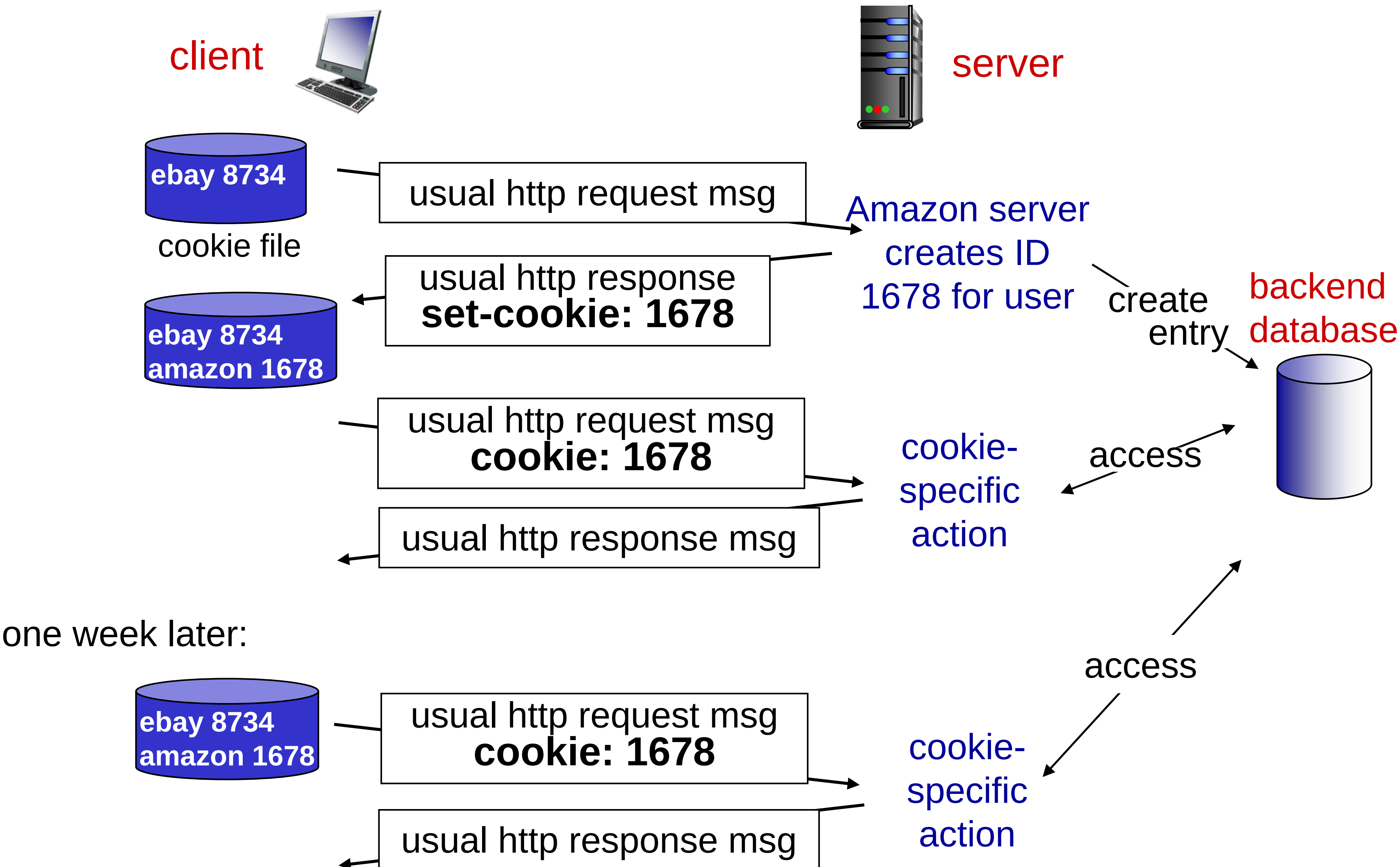
HTTP/1.0:

- 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

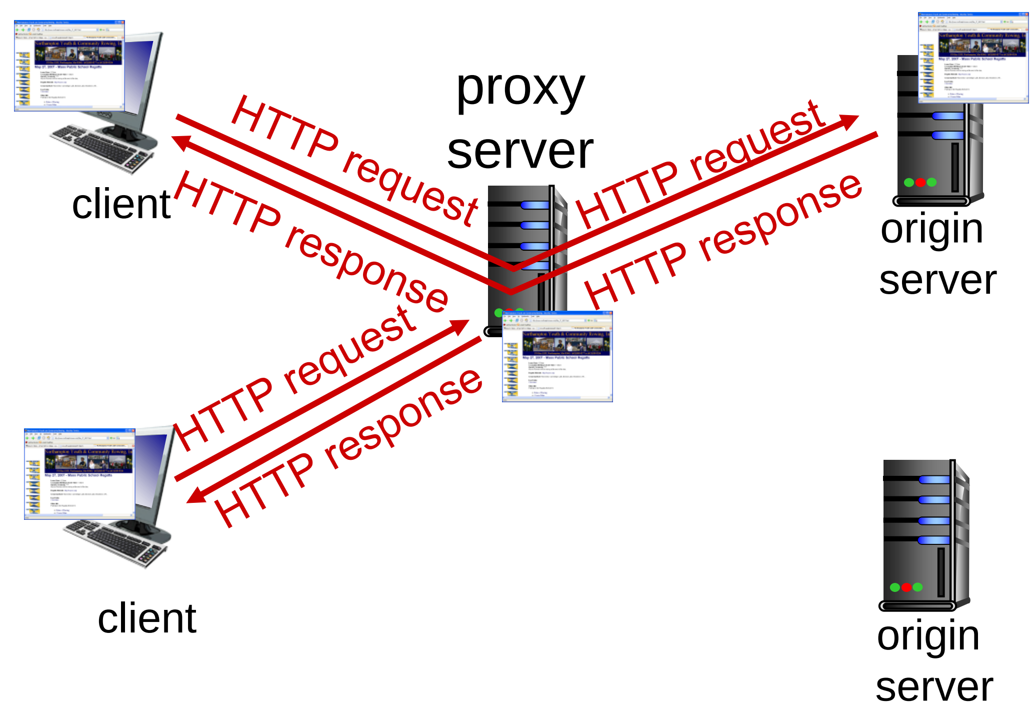
Cookies: keeping "state" (cont.)



Web caches (proxy server)

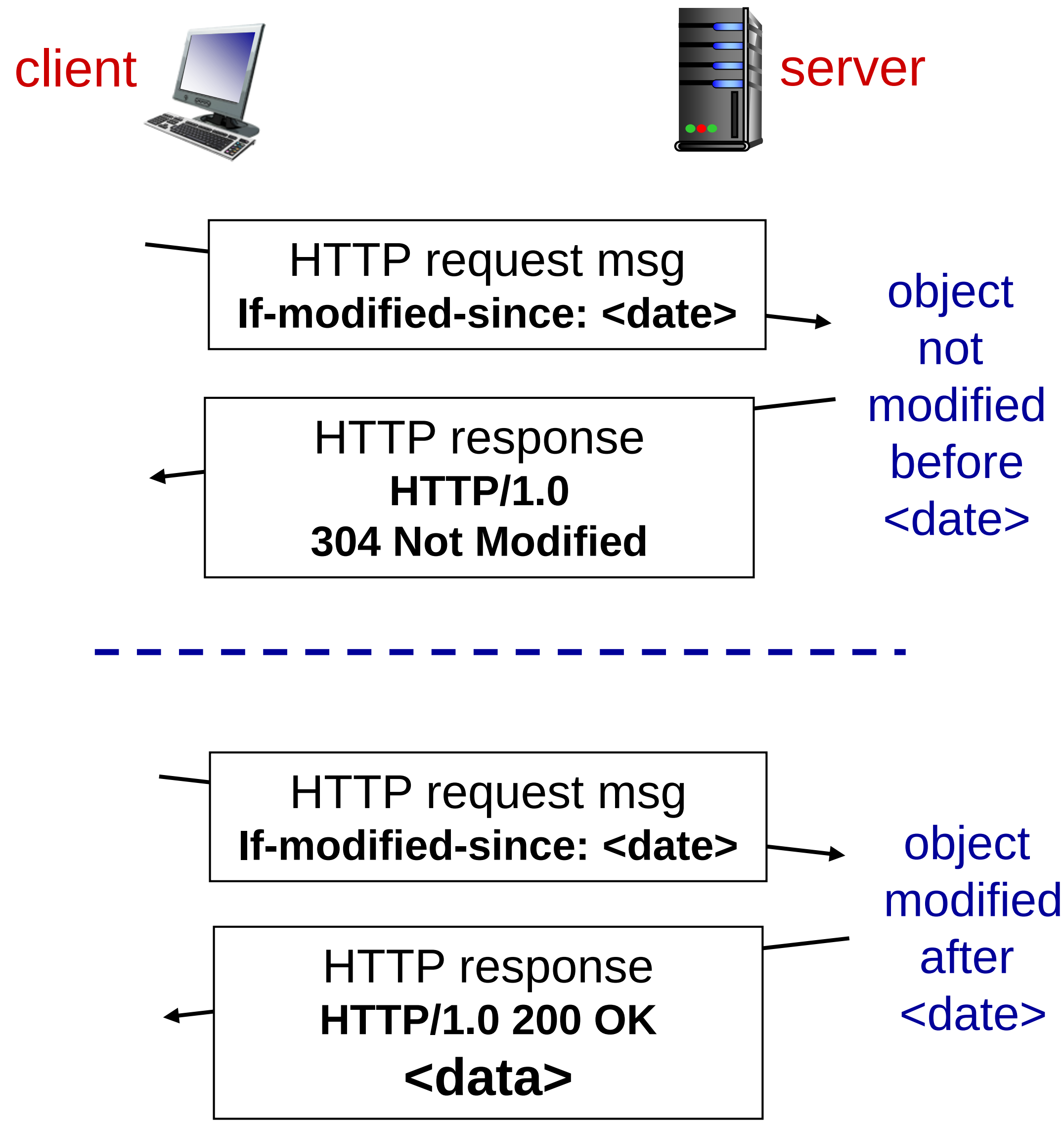
goal: satisfy client request without involving origin server

- 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



Conditional GET

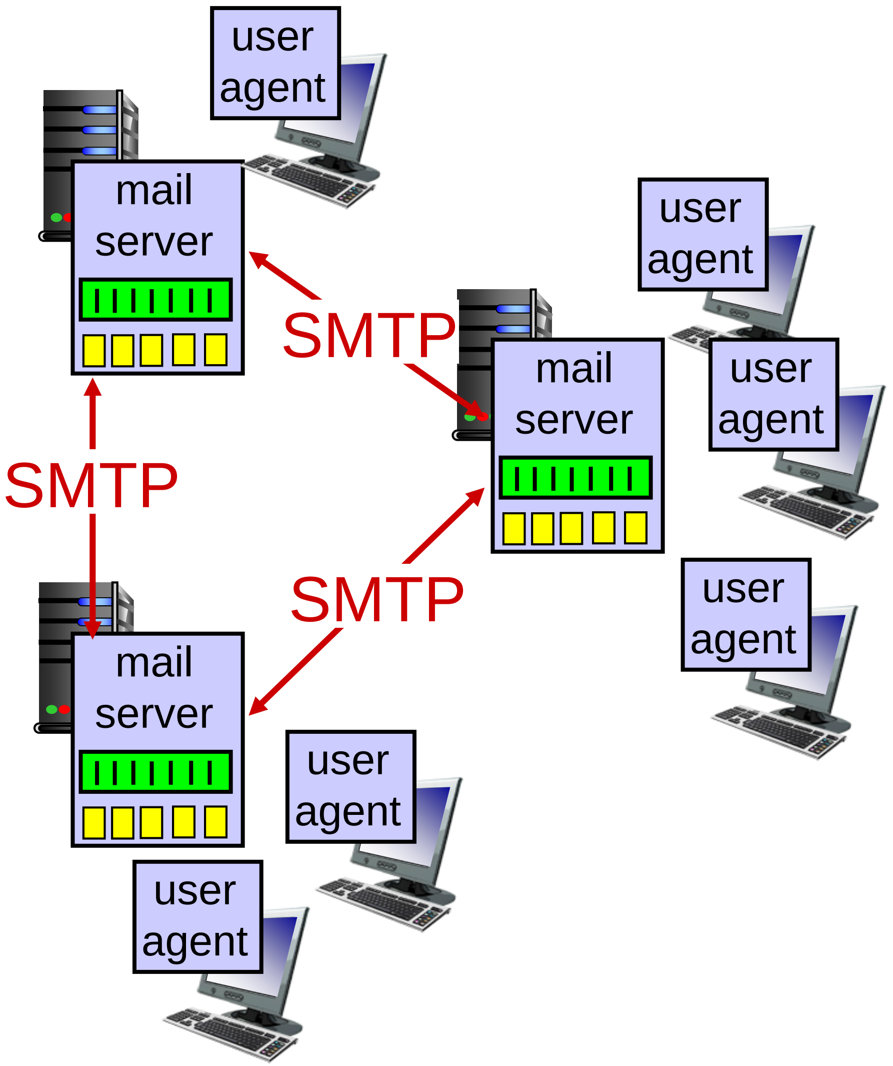
- *Goal:* don't send object if cache has up-to-date cached version
  - no object transmission delay
  - lower link utilization
- *cache:* specify date of cached copy in HTTP request
  `If-modified-since: <date>`
- *server:* response contains no object if cached copy is up-to-date:
  `HTTP/1.0 304 Not Modified`



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



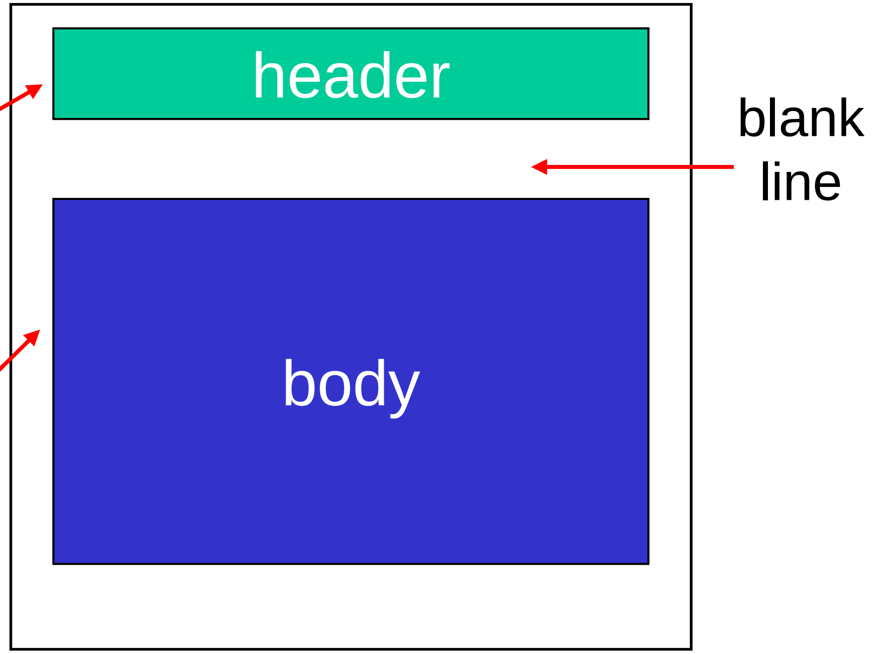
Mail message format

SMTP: protocol for exchanging email messages

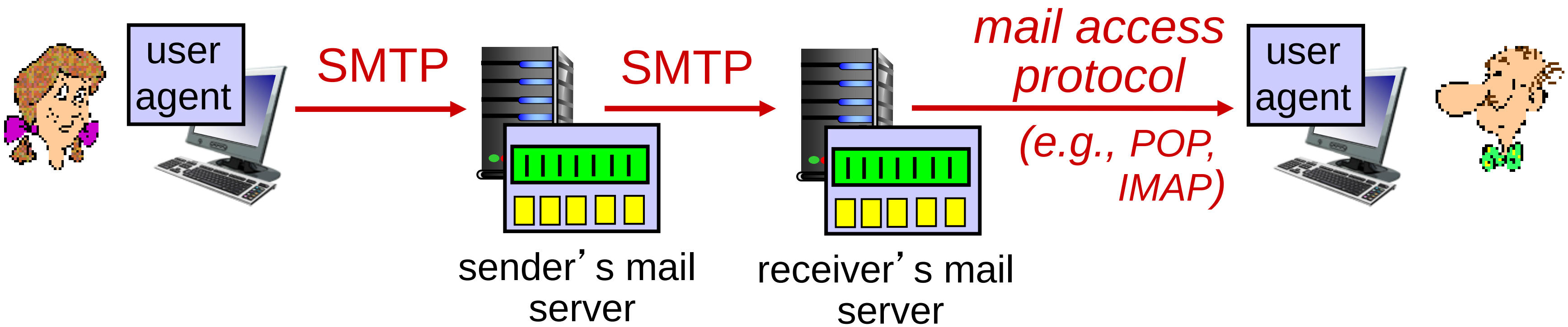
RFC 822: standard for text message format:

- 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 messages on server
  - HTTP: gmail, Hotmail, Yahoo! Mail, etc.

DNS: domain name service

people: many identifiers:

- SSN, name, passport #

Internet hosts, routers:

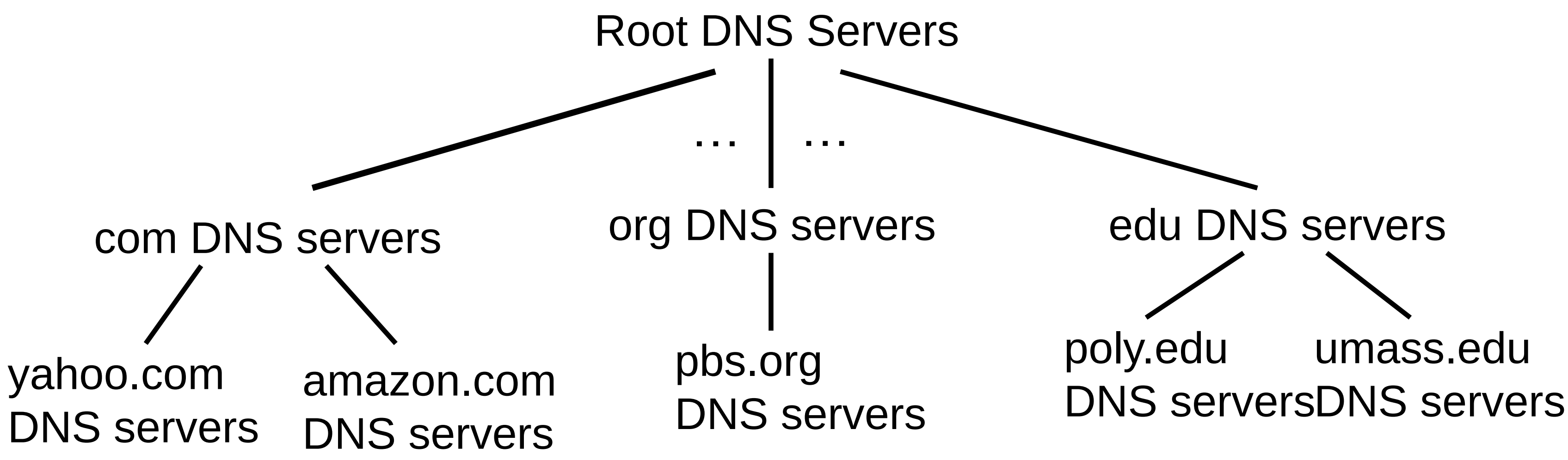
- IP address (32 bit) - used for addressing datagrams
- "name", e.g., www.yahoo.com - used by humans

Q: how to map between IP address and name, and vice versa ?

Domain Name System:

- *distributed database* implemented in hierarchy of many *name servers*
- *application-layer protocol:* hosts, name servers communicate to *resolve* names (address/name translation)
  - note: core Internet function, implemented as application-layer protocol
  - complexity at network's "edge"

DNS: a distributed, hierarchical database



client wants IP for www.amazon.com; 1st approximation:

- client queries root server to find com DNS server
- client queries .com DNS server to get amazon.com DNS server
- client queries amazon.com DNS server to get IP address for www.amazon.com

TLD, authoritative servers

top-level domain (TLD) servers:

- responsible for com, org, net, edu, aero, jobs, museums, and all top-level country domains, e.g.: uk, fr, ca, jp
- Network Solutions maintains servers for .com TLD
- Educause for .edu TLD

authoritative DNS servers:

- organization's own DNS server(s), providing authoritative hostname to IP mappings for organization's named hosts
- can be maintained by organization or service provider

Local DNS name server

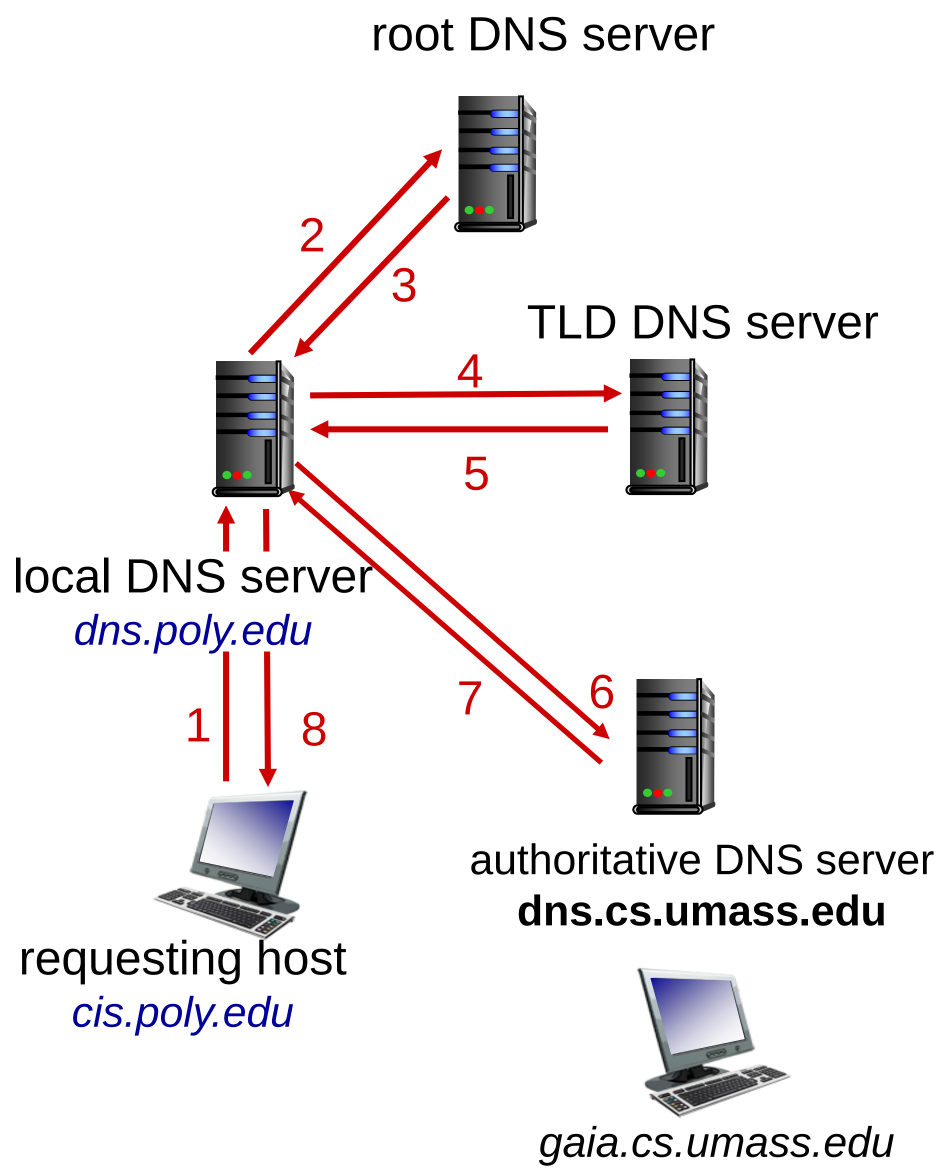
- does not strictly belong to hierarchy
- each ISP (residential ISP, company, university) has one
  - also called "default name server"
- when host makes DNS query, query is sent to its local DNS server
  - has local cache of recent name-to-address translation pairs (but may be out of date!)
  - acts as proxy, forwards query into hierarchy

DNS Example

- host at cis.poly.edu wants IP address for gaia.cs.umass.edu

iterated query:

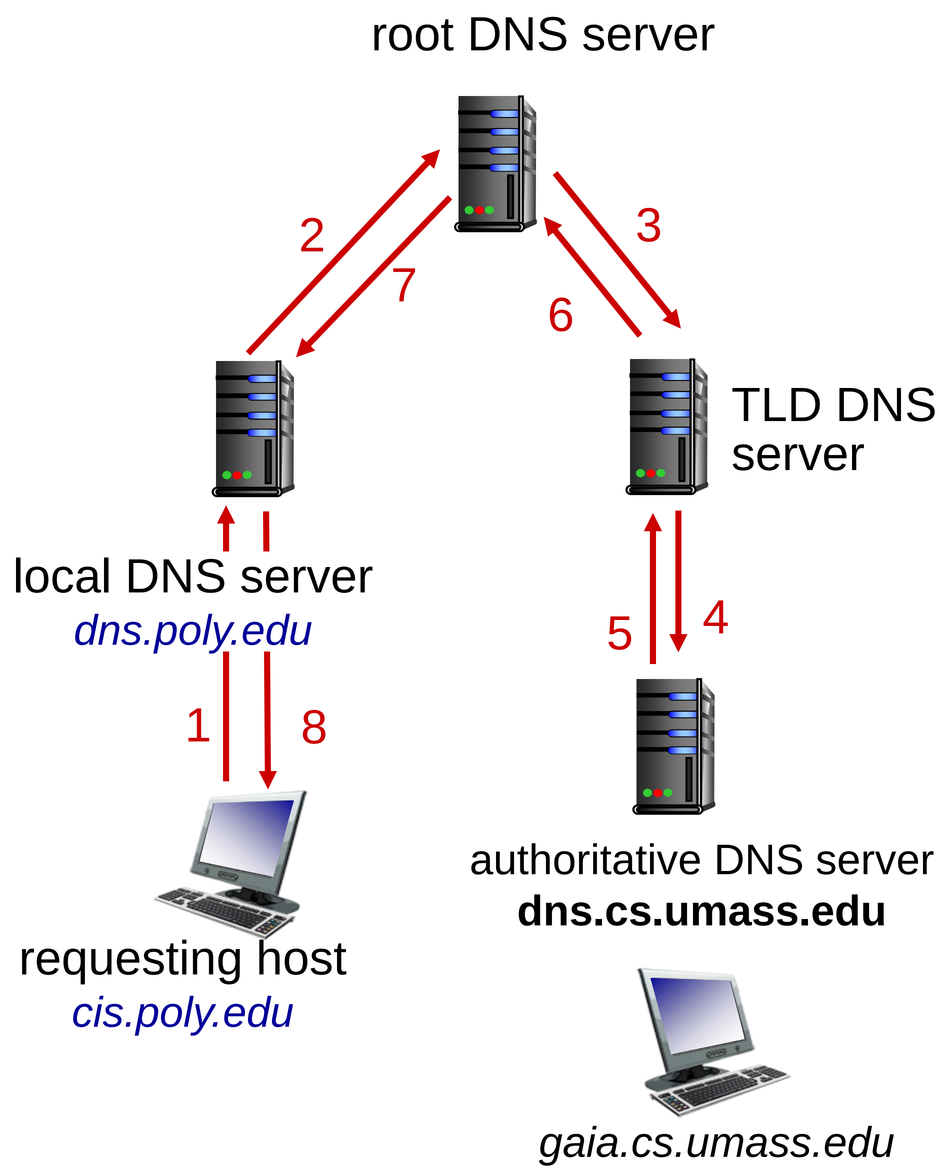
- contacted server replies with name of server to contact
- "I don't know this name, but ask this server"



DNS example

recursive query:

- puts burden of name resolution on contacted name server
- heavy load at upper levels of hierarchy?



DNS records

DNS: distributed database storing resource records (RR)

RR format: `(name, value, type, ttl)`

type=A

- **`name`** is hostname
- **`value`** is IP address

type=NS

- **`name`** is domain (e.g., foo.com)
- **`value`** is hostname of authoritative name server for this domain

type=CNAME

- **`name`** is alias name for some "canonical" (the real) name
- **`www.ibm.com`** is really **`servereast.backup2.ibm.com`**
- **`value`** is canonical name

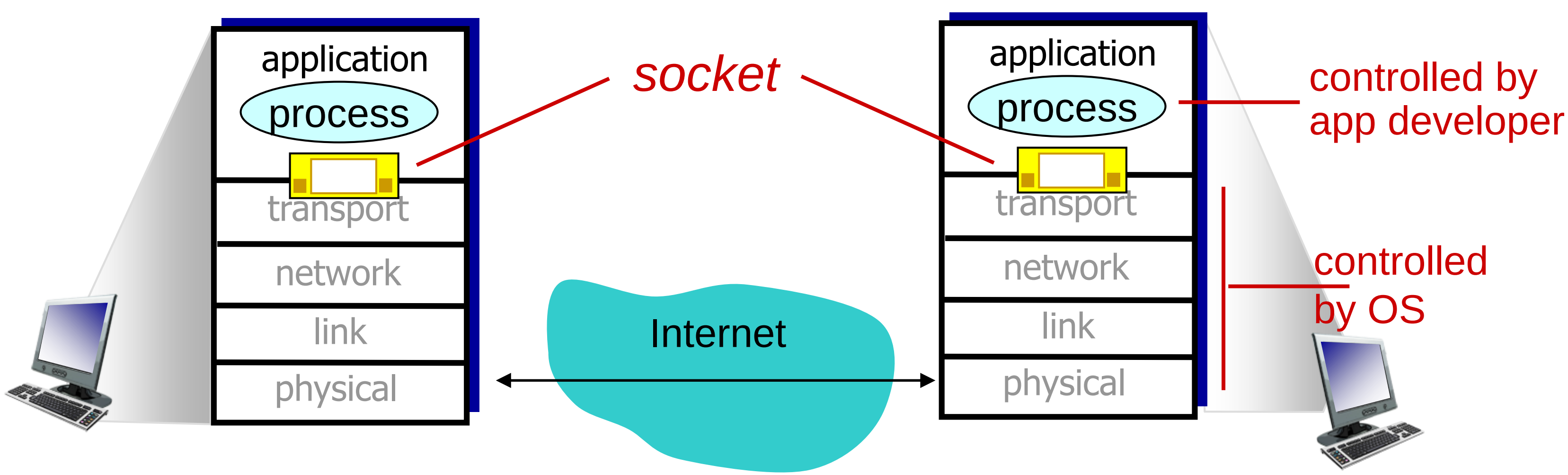
type=MX

- **`value`** is name of mailserver associated with **`name`**

SOCKET PROGRAMMING

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



Java Sockets Programming

- The package java.net provides support for sockets programming (and more).

- Typically you import everything defined in this package with:

```
import java.net.*;
```

Classes

`InetAddress`

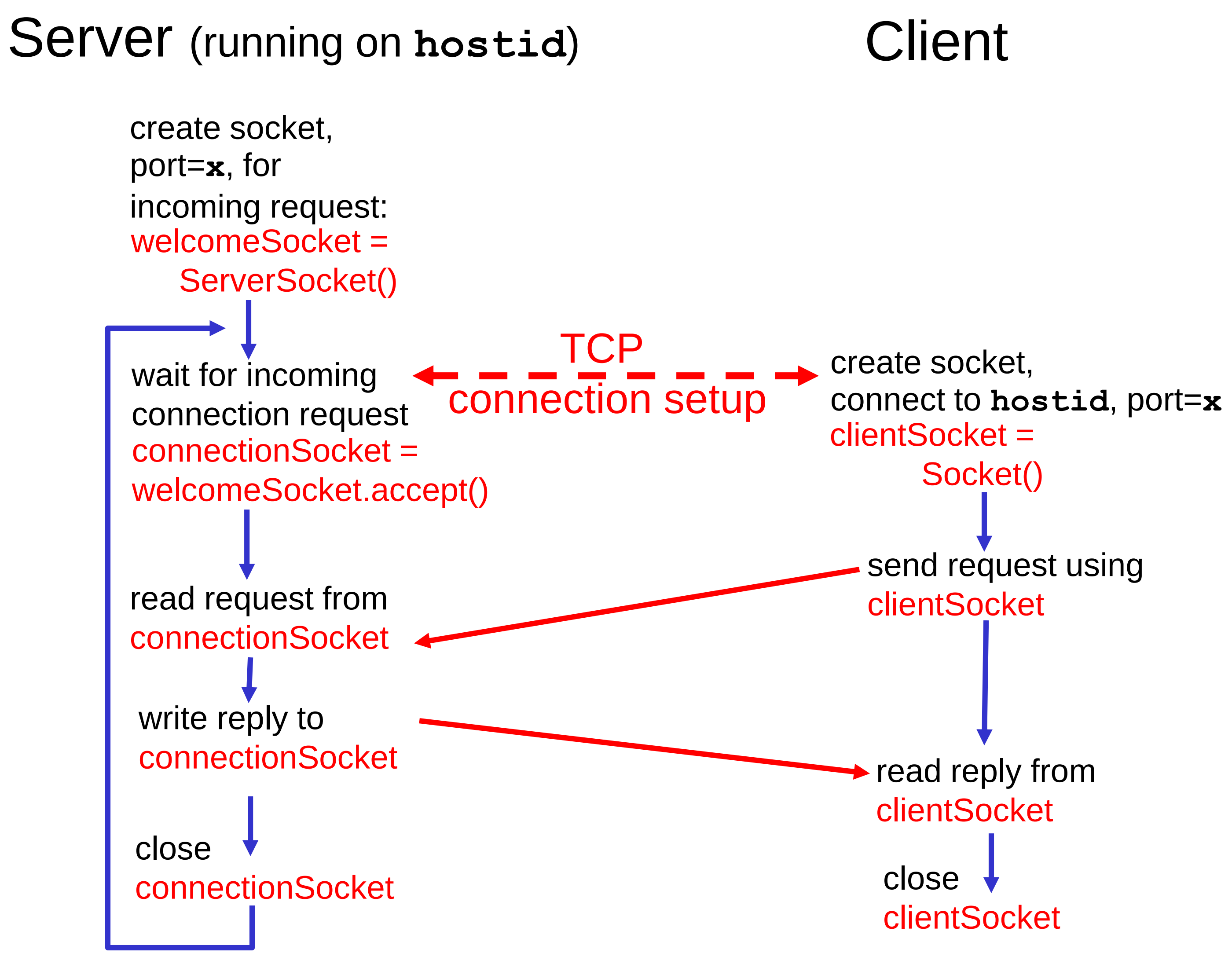
`Socket`

`ServerSocket`

`DatagramSocket`

`DatagramPacket`

Client/server socket interaction: TCP



Example: Java client (TCP)

```
import java.io.*;
import java.net.*;
class TCPClient {

    public static void main(String argv[]) throws Exception
    {
        String sentence;
        String modifiedSentence;

        BufferedReader inFromUser =
          new BufferedReader(new InputStreamReader(System.in));

        Socket clientSocket = new Socket("hostname", 6789);

        DataOutputStream outToServer =
          new DataOutputStream(clientSocket.getOutputStream());
```

Create input stream

Create client socket, connect to server

Create output stream attached to socket

Example: Java client (TCP), cont.

Create input stream attached to socket

```
BufferedReader inFromServer =
  new BufferedReader(new
  InputStreamReader(clientSocket.getInputStream()));

 sentence = inFromUser.readLine();
```

Send line to server

```
 outToServer.writeBytes(sentence + '\n');
```

Read line from server

```
 modifiedSentence = inFromServer.readLine();

 System.out.println("FROM SERVER: " + modifiedSentence);

 clientSocket.close();

}
}
```

Example: Java server (TCP)

```
import java.io.*;
import java.net.*;

class TCPServer {

  public static void main(String argv[]) throws Exception
   {
     String clientSentence;
     String capitalizedSentence;
```

Create welcoming socket at port 6789

```
     ServerSocket welcomeSocket = new ServerSocket(6789);

     while(true) {
```

Wait, on welcoming socket for contact by client

```
        Socket connectionSocket = welcomeSocket.accept();
```

Create input stream, attached to socket

```
       BufferedReader inFromClient =
         new BufferedReader(new
         InputStreamReader(connectionSocket.getInputStream()));
```

Example: Java server (TCP), cont

Create output stream, attached to socket

Read in line from socket

Write out line to socket

```
DataOutputStream  outToClient =
  new DataOutputStream(connectionSocket.getOutputStream());

clientSentence = inFromClient.readLine();

capitalizedSentence = clientSentence.toUpperCase() + '\n';

outToClient.writeBytes(capitalizedSentence);
          }
      }
  }
```

End of while loop,
loop back and wait for
another client connection

Example: Java client (UDP)

Create
input stream

Create
client socket

Translate
hostname to IP
address using DNS

```
import java.io.*;
import java.net.*;

class UDPClient {
    public static void main(String args[]) throws Exception
    {

      BufferedReader inFromUser =
        new BufferedReader(new InputStreamReader(System.in));

      DatagramSocket clientSocket = new DatagramSocket();

      InetAddress IPAddress = InetAddress.getByName("hostname");

      byte[] sendData = new byte[1024];
      byte[] receiveData = new byte[1024];

      String sentence = inFromUser.readLine();

      sendData = sentence.getBytes();
```

Example: Java client (UDP), cont.

Create datagram with data-to-send, length, IP addr, port

Send datagram to server

Read datagram from server

```
DatagramPacket sendPacket =
   new DatagramPacket(sendData, sendData.length, IPAddress, 9876);

clientSocket.send(sendPacket);

DatagramPacket receivePacket =
   new DatagramPacket(receiveData, receiveData.length);

clientSocket.receive(receivePacket);

String modifiedSentence =
    new String(receivePacket.getData());

System.out.println("FROM SERVER:" + modifiedSentence);
clientSocket.close();
}
}
```

Example: Java server (UDP)

Create datagram socket at port 9876

Create space for received datagram

Receive datagram

```
import java.io.*;
import java.net.*;

class UDPServer {
  public static void main(String args[]) throws Exception
    {

      DatagramSocket serverSocket = new DatagramSocket(9876);

      byte[] receiveData = new byte[1024];
      byte[] sendData  = new byte[1024];

      while(true)
        {

          DatagramPacket receivePacket =
             new DatagramPacket(receiveData, receiveData.length);
           serverSocket.receive(receivePacket);
```

Example: Java server (UDP), cont

```
String sentence = new String(receivePacket.getData());

InetAddress IPAddress = receivePacket.getAddress();

int port = receivePacket.getPort();

        String capitalizedSentence = sentence.toUpperCase();

sendData = capitalizedSentence.getBytes();

DatagramPacket sendPacket =
   new DatagramPacket(sendData, sendData.length, IPAddress,
                      port);

serverSocket.send(sendPacket);
  }
 }
}
```

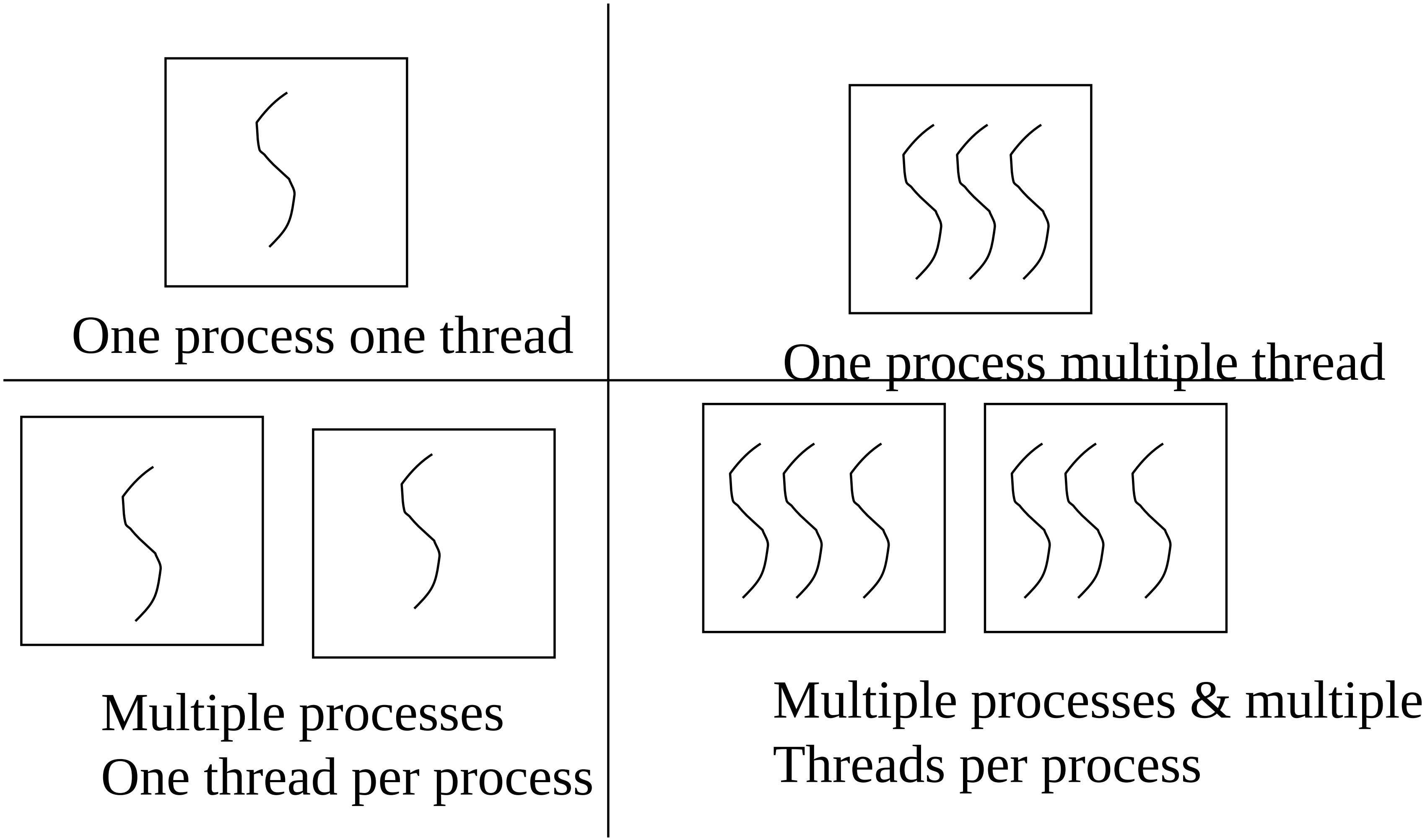
Get IP addr port #, of sender

Create datagram to send to client

Write out datagram to socket

End of while loop, loop back and wait for another datagram

Possible combination of thread and processes



Example

Step 1: Implement the "Runnable" interface

```
public class RunnableExample implements Runnable {

    public static void main(String[] args) {
        System.out.println("Inside : " + Thread.currentThread().getName());

        System.out.println("Creating Runnable...");
        Runnable runnable = new RunnableExample();

        System.out.println("Creating Thread...");
        Thread thread = new Thread(runnable);

        System.out.println("Starting Thread...");
        thread.start();
    }

    @Override
    public void run() {
        System.out.println("Inside : " + Thread.currentThread().getName());
    }
}
```

Step 2: Create the class object that needs to be executed in multithreaded fashion

Step 4: Create thread(s) and pass the object. Thread is suspended at this point

Step 5: Call start() to start the thread

Step 3: Make sure the class has a run() method

Using Runnable

Method	Meaning
getName	Obtain thread's name
getPriority	Obtain thread's priority
isAlive	Determine if a thread is still running
join	Wait for a thread to terminate
run	Entry point for the thread
sleep	Suspend a thread for a period of time
start	Start a thread by calling its run method

yield() and sleep()

- Sometimes a thread can determine that it has nothing to do
  - Sometimes the system can determine this. ie. waiting for I/O
- When a thread has nothing to do, it should not use CPU
  - This is called a busy-wait.
  - Threads in busy-wait are busy using up the CPU doing nothing.
    - Often, threads in busy-wait are continually checking a flag to see if there is anything to do.
- Threads in busy-wait should be moved from the Run queue to the Wait queue so that they do not hog the CPU
  - Use yield() or sleep(time)
  - Yield simply tells the scheduler to schedule another thread
  - Sleep guarantees that this thread will remain in the wait queue for the specified number of milliseconds.

The Static yield() Method

You can use the yield() method to temporarily release time for other threads.

```
public void run() {
  for (int i = 1; i <= lastNum; i++)
 {
    System.out.print(" " + i);
    Thread.yield();
  }
}
```

Every time a number is printed, the thread is yielded.

The Static sleep(milliseconds) Method

The sleep(long mills) method puts the thread to sleep for the specified time in milliseconds to allow other threads to execute:

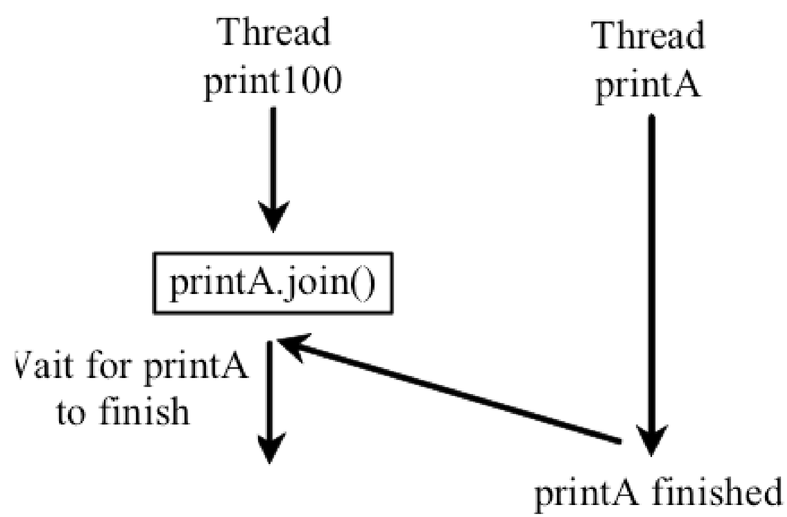
```
public void run() {
  for (int i = 1; i <= lastNum; i++) {
    System.out.print(" " + i);
    try {
      if (i >= 50) Thread.sleep(1);
    }
    catch (InterruptedException ex) {
    }
  }
}
```

Every time a number (>= 50) is printed, the thread is put to sleep for 1 millisecond.

The join() Method

You can use the join() method to force one thread to wait for another thread to finish. For example, suppose you modify the code in Lines 53-57 in TaskThreadDemo.java as follows:

```
public void run() {
  Thread thread4 = new Thread(
    new PrintChar('c', 40));
  thread4.start();
  try {
    for (int i = 1; i <= lastNum; i++) {
      System.out.print(" " + i);
      if (i == 50) thread4.join();
    }
  }
  catch (InterruptedException ex) {
  }
```



The numbers after 50 are printed after thread printA is finished.

The `synchronized` keyword

- To avoid race conditions, threads must be prevented from simultaneously entering certain part of the program, known as critical section.
  - In the previous scenario the critical section is the entire deposit method.
- You can use the synchronized keyword to synchronize the method so that only one thread can access the method at a time.
- One approach is to make Account thread-safe by adding the synchronized keyword in the deposit method in class Account as follows:

```
public synchronized void deposit() {
    ….
    newBalance= balance + 1;
    …..
    Balance = newBalance;
}
```

- A synchronized method acquires a lock before it executes.
  - In the case of an instance method, the lock is on the object for which the method was invoked.
  - In the case of a static method, the lock is on the class.

Synchronizing Code Blocks

- Enclose lines of code in a *synchronized* block

```
public void close() {
        System.out.printf("Closing accounting");
        long totalAmmount;
        synchronized (this) {
                totalAmmount=cash;
                cash=0;
        }
        System.out.println("The total amount is " + totalAmmount);
}
```

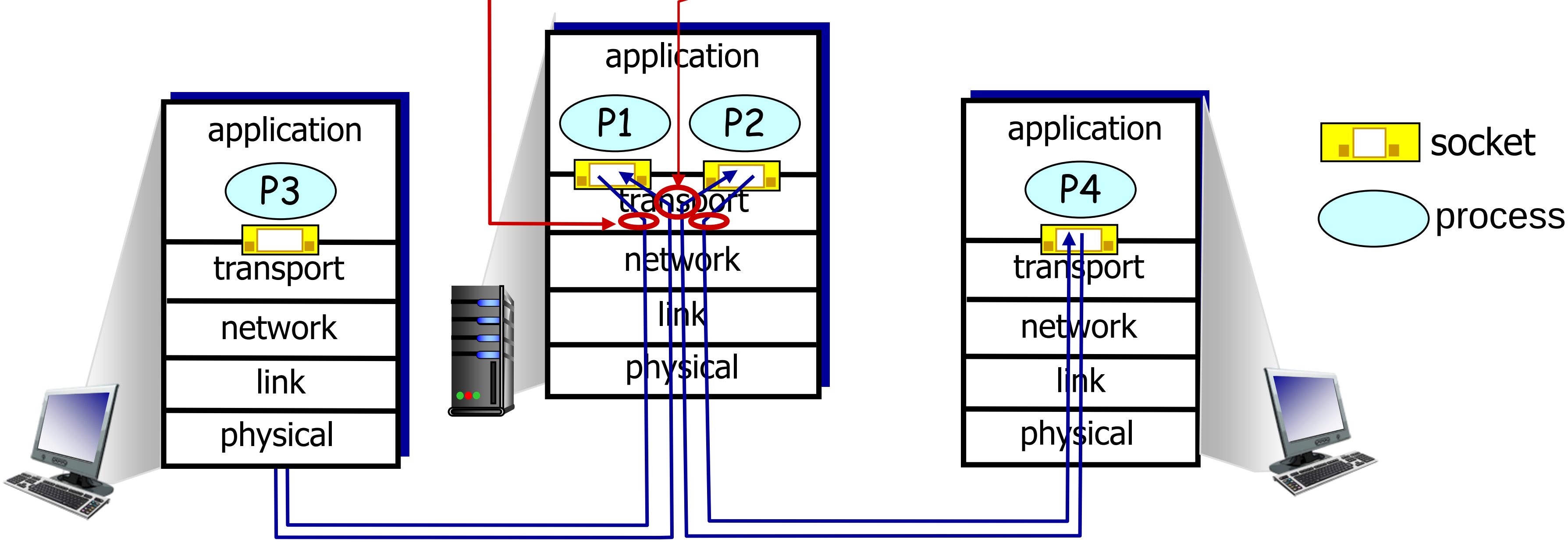
- More than one thread could try to execute this code, but one acquires the lock and the others "block" or wait until the first thread releases the lock

TRANSPORT LAYER

Multiplexing/demultiplexing

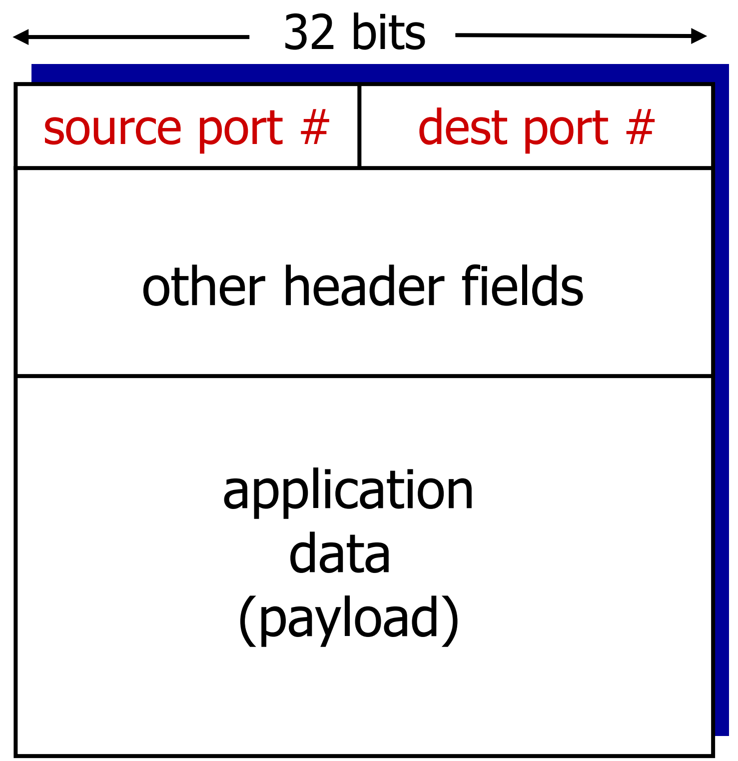
multiplexing at sender:
handle data from multiple sockets, add transport header (later used for demultiplexing)

demultiplexing at receiver:
use header info to deliver received segments to correct socket



How demultiplexing works

- host receives IP datagrams
  - each datagram has source IP address, destination IP address
  - each datagram carries one transport-layer segment
  - each segment has source, destination port number
- host uses *IP addresses & port numbers* to direct segment to appropriate socket



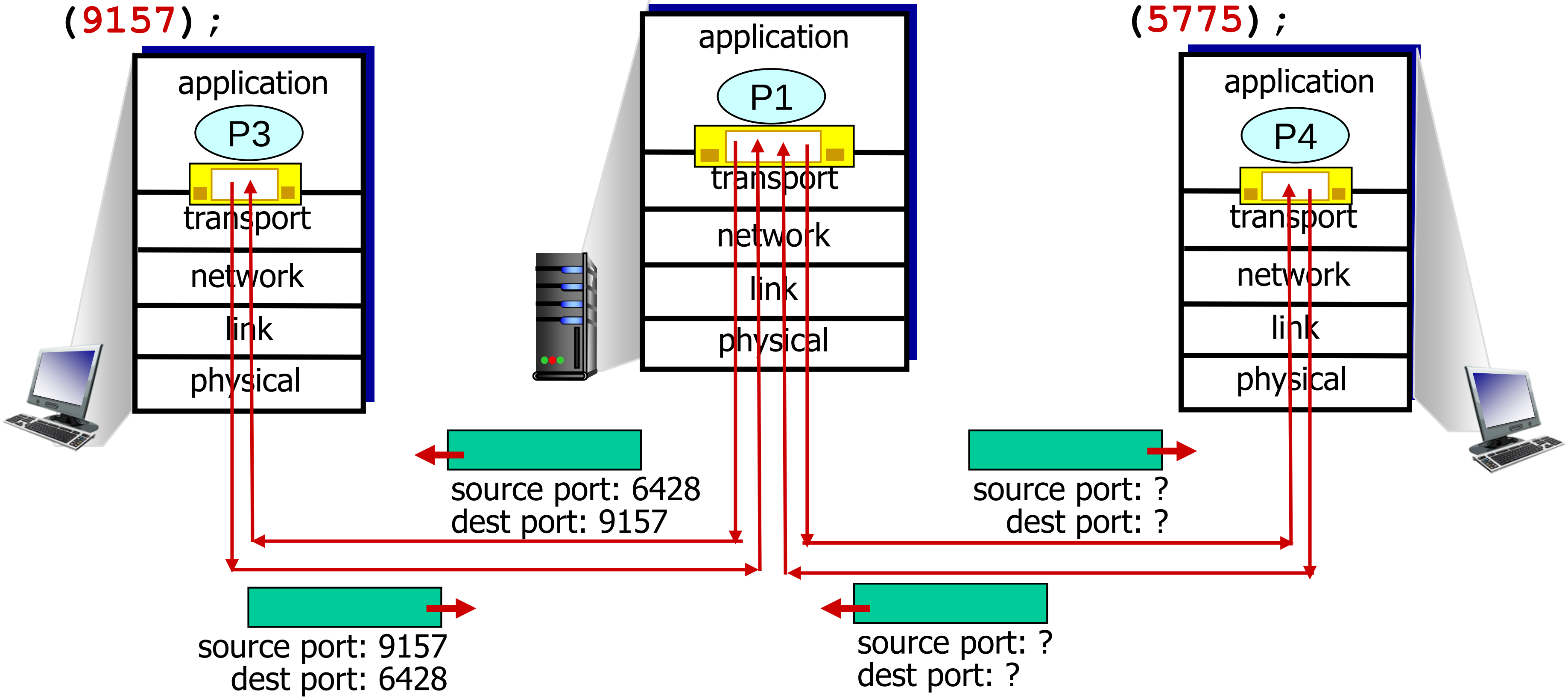
TCP/UDP segment format

Connectionless demux: example

```
DatagramSocket
 serverSocket = new
 DatagramSocket
 (6428);
```

```
DatagramSocket
 mySocket2 = new
 DatagramSocket
 (9157);
```

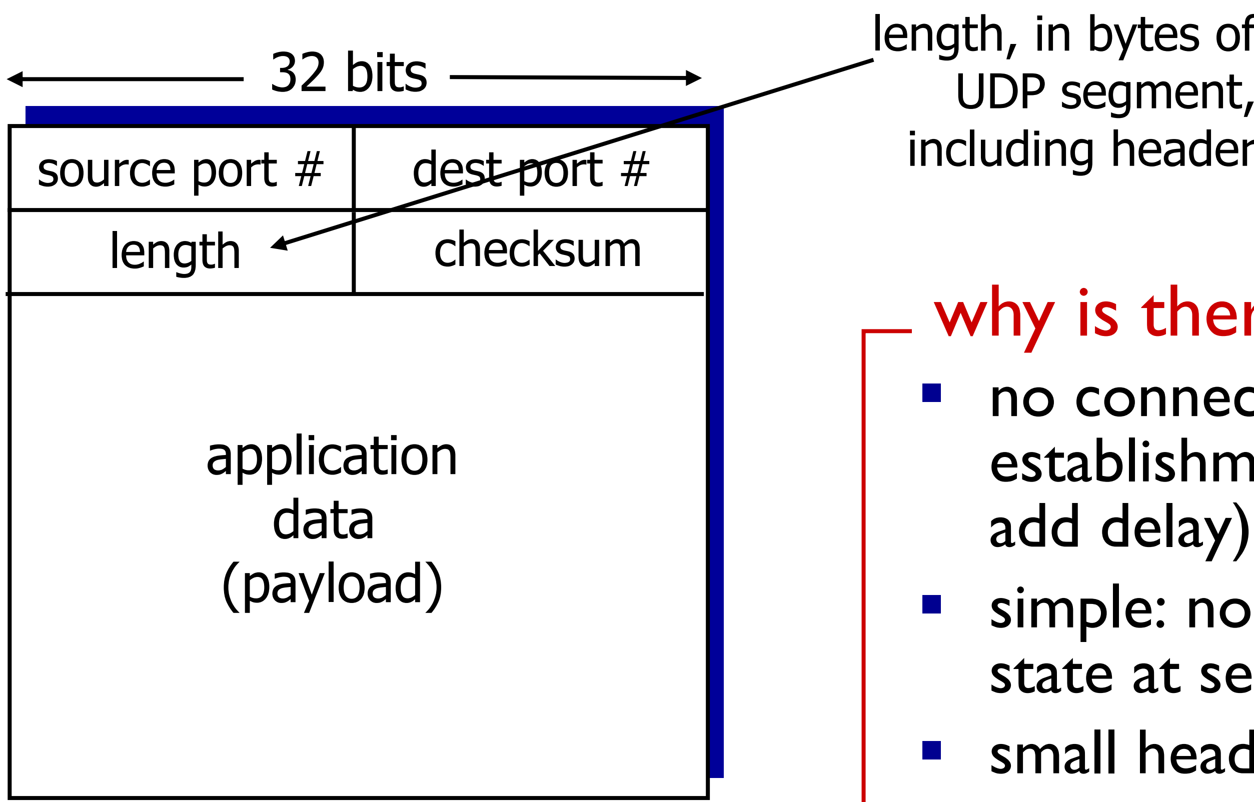
```
DatagramSocket
 mySocket1 = new
 DatagramSocket
 (5775);
```



Connection-oriented demux

- TCP socket identified by 4-tuple:
  - source IP address
  - source port number
  - dest IP address
  - dest port number
- demux: receiver uses all four values to direct segment to appropriate socket
- server host may support many simultaneous TCP sockets:
  - each socket identified by its own 4-tuple
- web servers have different sockets for each connecting client
  - non-persistent HTTP will have different socket for each request

UDP: segment header



UDP segment format

why is there a UDP?

- no connection establishment (which can add delay)
- simple: no connection state at sender, receiver
- small header size
- no congestion control: UDP can blast away as fast as desired

UDP checksum

Goal: detect "errors" (e.g., flipped bits) in transmitted segment

sender:

- treat segment contents, including header fields, as sequence of 16-bit integers
- checksum: addition (one's complement sum) of segment contents
- sender puts checksum value into UDP checksum field

receiver:

- compute checksum of received segment
- check if computed checksum equals checksum field value:
  - NO - error detected
  - YES - no error detected.

Internet checksum: example

example: add two 16-bit integers

		1	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0
		1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
wraparound	1	1	0	1	1	1	0	1	1	1	0	1	1	1	0	1	1
sum		1	0	1	1	1	0	1	1	1	0	1	1	1	1	0	0
checksum		0	1	0	0	0	1	0	0	0	1	0	0	0	0	1	1

Note: when adding numbers, a carryout from the most significant bit needs to be added to the result

* Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu/kurose_ross/interactive/