# RELATED CONCEPTS IN COMPUTER NETWORKS

Some slides have been taken from: *Computer Networking: A Top Down Approach Featuring the Internet*, 3rd edition. Jim Kurose, Keith Ross. Addison-Wesley, July 2004. All material copyright 1996-2004. J.F Kurose and K.W. Ross, All Rights Reserved.

1

Contents

- Computer Networks
- Internet protocol stack
- Application layer
- TCP & UDP
- Internet layer

What's the Internet?

- *Internet:* "network of networks"
  - Interconnected ISPs
- *protocols* control sending, receiving of msgs
  - e.g., TCP, IP, HTTP, Skype, 802.11
- *Internet standards*
  - RFC: Request for comments
  - IETF: Internet Engineering Task Force

mobile network

global ISP

home network

regional ISP

institutional network
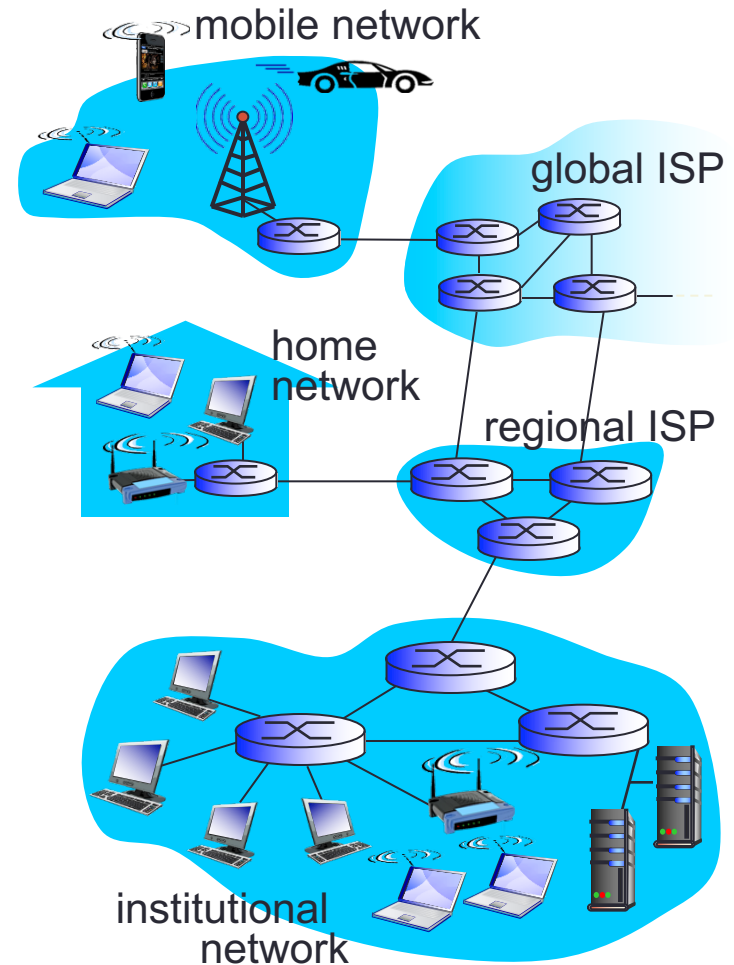
What's the Internet?

- *Infrastructure that provides services to applications:*
  - Web, VoIP, email, games, e-commerce, social nets, …
- *provides programming interface to apps*
  - hooks that allow sending and receiving app programs to "connect" to Internet
  - provides service options, analogous to postal service

mobile network

global ISP

home network

regional ISP

institutional network
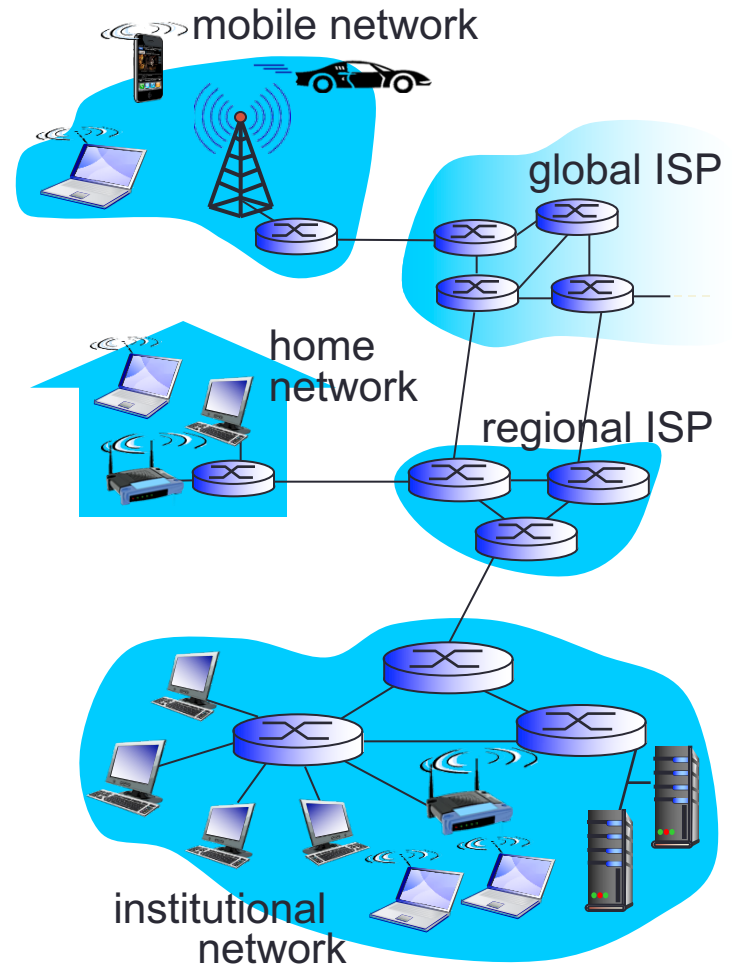
What's a protocol?

human protocols:

- "what's the time?"
- "I have a question"
- introductions

… specific msgs sent

… specific actions taken when msgs received, or other events
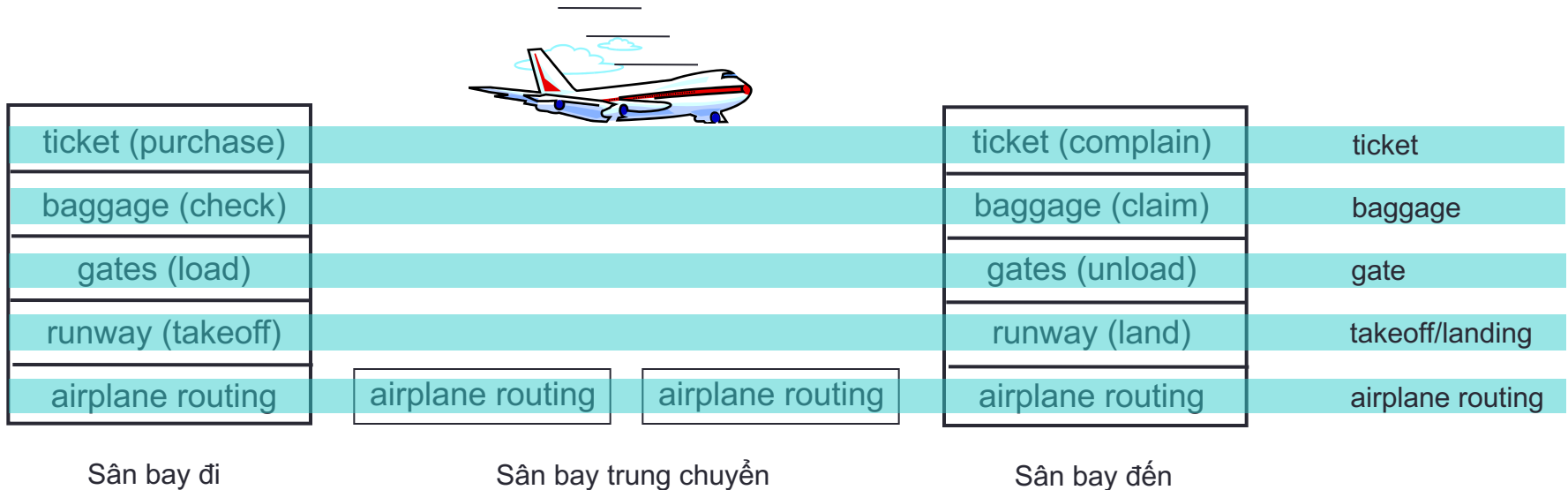
network protocols:

- machines rather than humans
- all communication activity in Internet governed by protocols

protocols define format, order of msgs sent and received among network entities, and actions taken on msg transmission, receipt

Vì sao phải phân tầng?

- Đối với các hệ thống phức tạp: nguyên lý *"chia để trị"*
- Cho phép xác định rõ nhiệm vụ của mỗi bộ phận và quan hệ giữa chúng
- Cho phép dễ dàng bảo trì và nâng cấp hệ thống
  - Thay đổi bên trong một bộ phận không ảnh hưởng đến các bộ phận khác
  - Như việc nâng cấp từ CD lên DVD player mà không phải thay loa.

Phân tầng các chức năng hàng không

ticket (purchase)			ticket (complain)	ticket
baggage (check)			baggage (claim)	baggage
gates (load)			gates (unload)	gate
runway (takeoff)			runway (land)	takeoff/landing
airplane routing	airplane routing	airplane routing	airplane routing	airplane routing

Sân bay đi   Sân bay trung chuyển   Sân bay đến

Tầng: Mỗi tầng có nhiệm vụ cung cấp 1 dịch vụ
- Dựa trên các chức năng của chính tầng đó
- Dựa trên các dịch vụ cung cấp bởi tầng dưới

OSI protocol stack

- **Tầng Ứng dụng (Application):** cung cấp các ứng dụng trên mạng (web, email, truyền file…)
- **Tầng Trình diễn (Presentation):** biểu diễn dữ liệu của ứng dụng, e.g., mã hóa, nén, chuyển đổi…
- **Tầng Phiên(Session):** quản lý phiên làm việc, đồng bộ hóa phiên, khôi phục quá trình trao đổi dữ liệu
- **Tầng Giao vận (Transport):** Xử lý việc truyền-nhận dữ liệu cho các ứng dụng chạy trên nút mạng đầu-cuối
- **Tầng Mạng (Network):** Chọn đường (định tuyến), chuyển tiếp gói tin từ nguồn đến đích
- **Tầng Liên kết dữ liệu (Data link):** Truyền dữ liệu trên các liên kết vật lý giữa các nút mạng kế tiếp nhau
- **Tầng Vật lý (Physical):** Chuyển dữ liệu (bit) thành tín hiệu và truyền

| Application |
| Presentation |
| Session |
| Transport |
| Network |
| Data link |
| Physical |
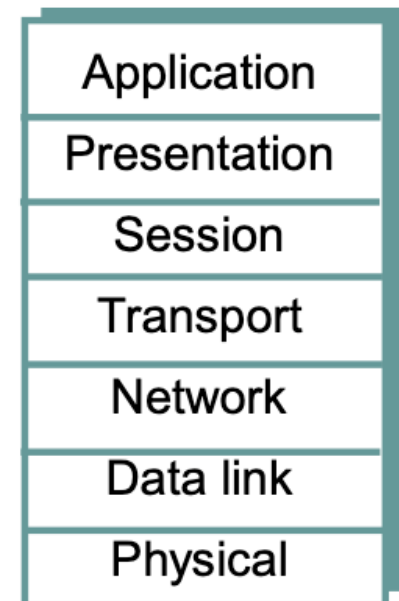
OSI protocol stack

- Application layer: defines communication between different parts of the same application
  - Presentation layer: application data representation, data encryption, compression, conversion…
  - Session layer: manages sessions, synchronization, recovery of data transmission process
- Transport layer: Transmits data between applications
- Network layer: Transmits data between distance network elements: Taking care of routing and forwarding data
- Data link layer: Transmits data between adjacent network elements.
- Physical layer: Transmits bits on the medium. Converting bits to physical form appropriate to the medium.

application
presentation
session
transport
network
data link
physical

TCP/IP protocol stack
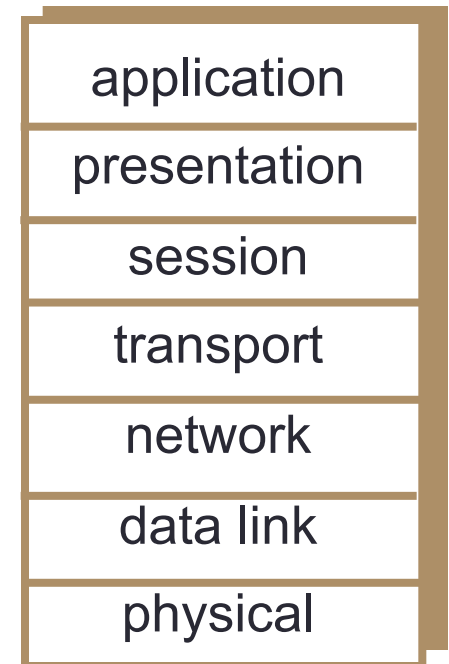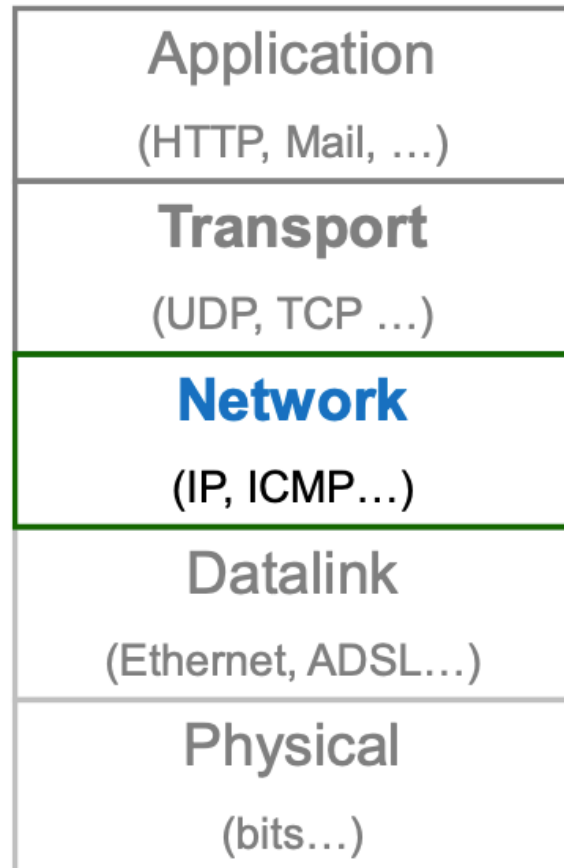
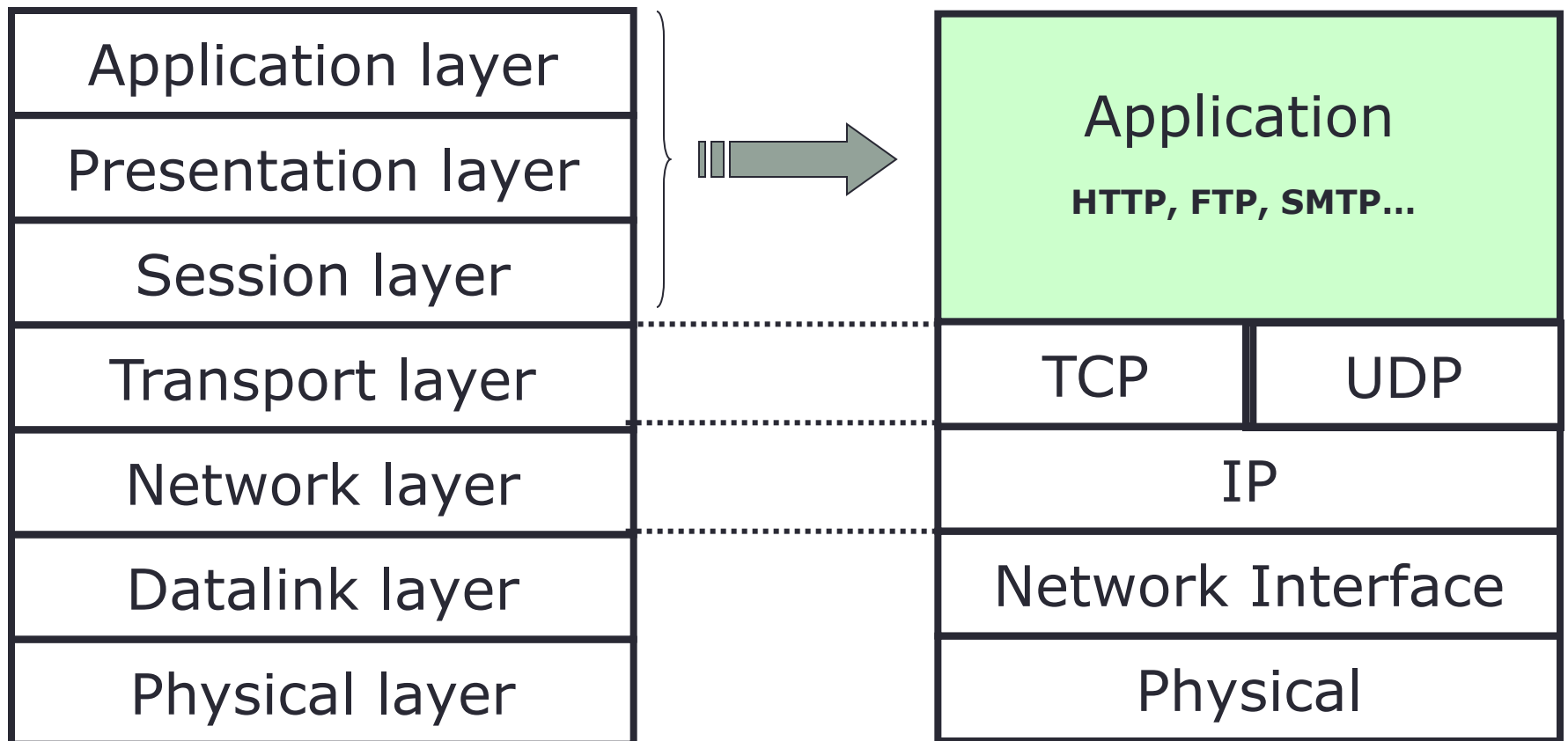|	
Application (HTTP, Mail, …)	Hỗ trợ các ứng dụng trên mạng
Transport (UDP, TCP …)	Điều khiển truyền dữ liệu giữa các tiến trình của tầng ứng dụng
Network (IP, ICMP…)	**Điều khiển truyền dữ liệu giữa các nút mạng qua môi trường liên mạng**
Datalink (Ethernet, ADSL…)	Hỗ trợ việc truyền thông cho các thành phần kế tiếp trên cùng 1 mạng
Physical (bits…)	Truyền và nhận dòng bit trên đường truyền vật lý

TCP/IP protocol stack

- application: supporting network applications
  - FTP, SMTP, STTP
- transport: host-host data transfer
  - TCP, UDP
- network: routing of datagrams from source to destination
  - IP, routing protocols
- link: data transfer between neighboring network elements
  - PPP, Ethernet
- physical: bits "on the wire"

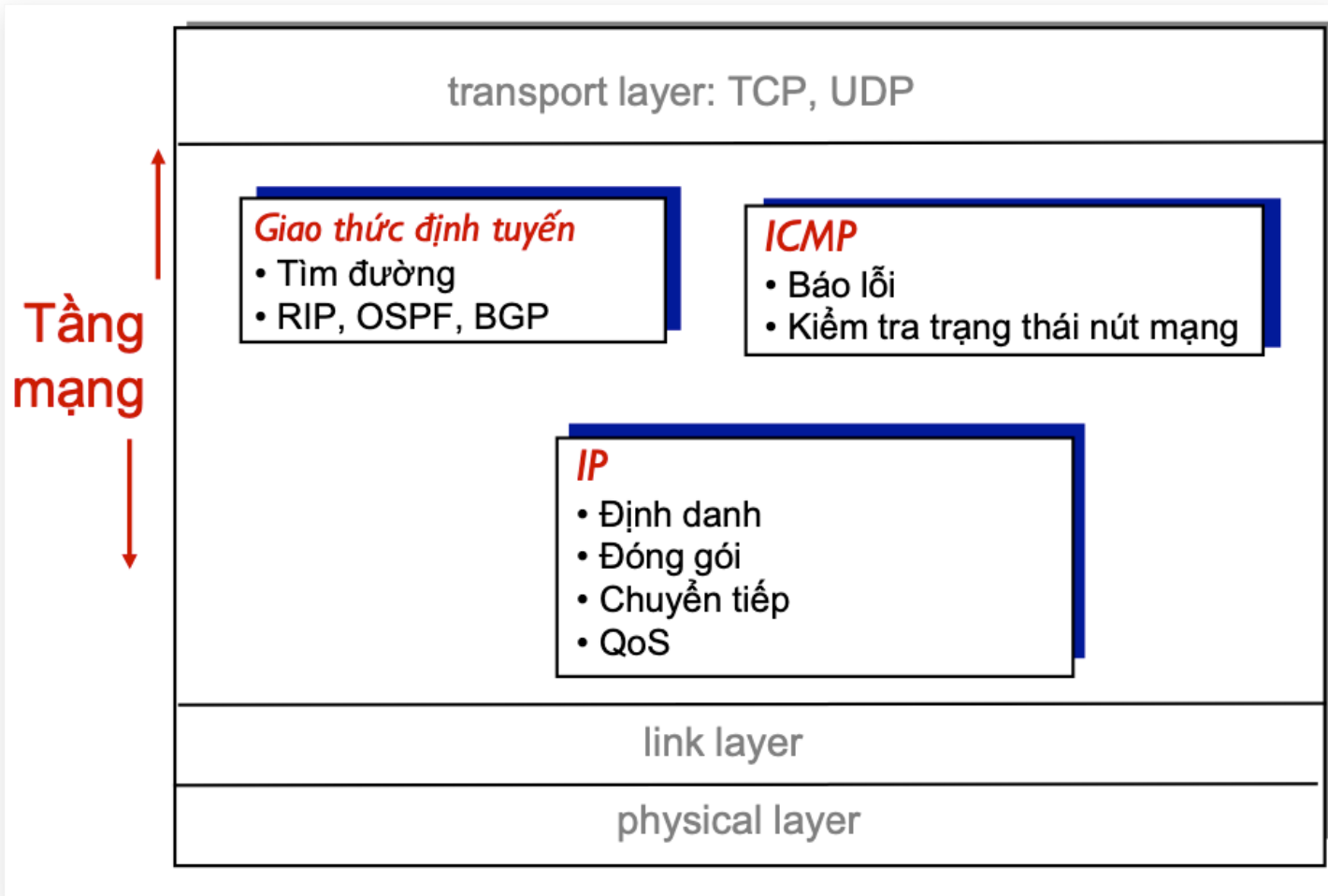| application |
|---|
| transport |
| network |
| link |
| physical |

OSI and TCP/IP models

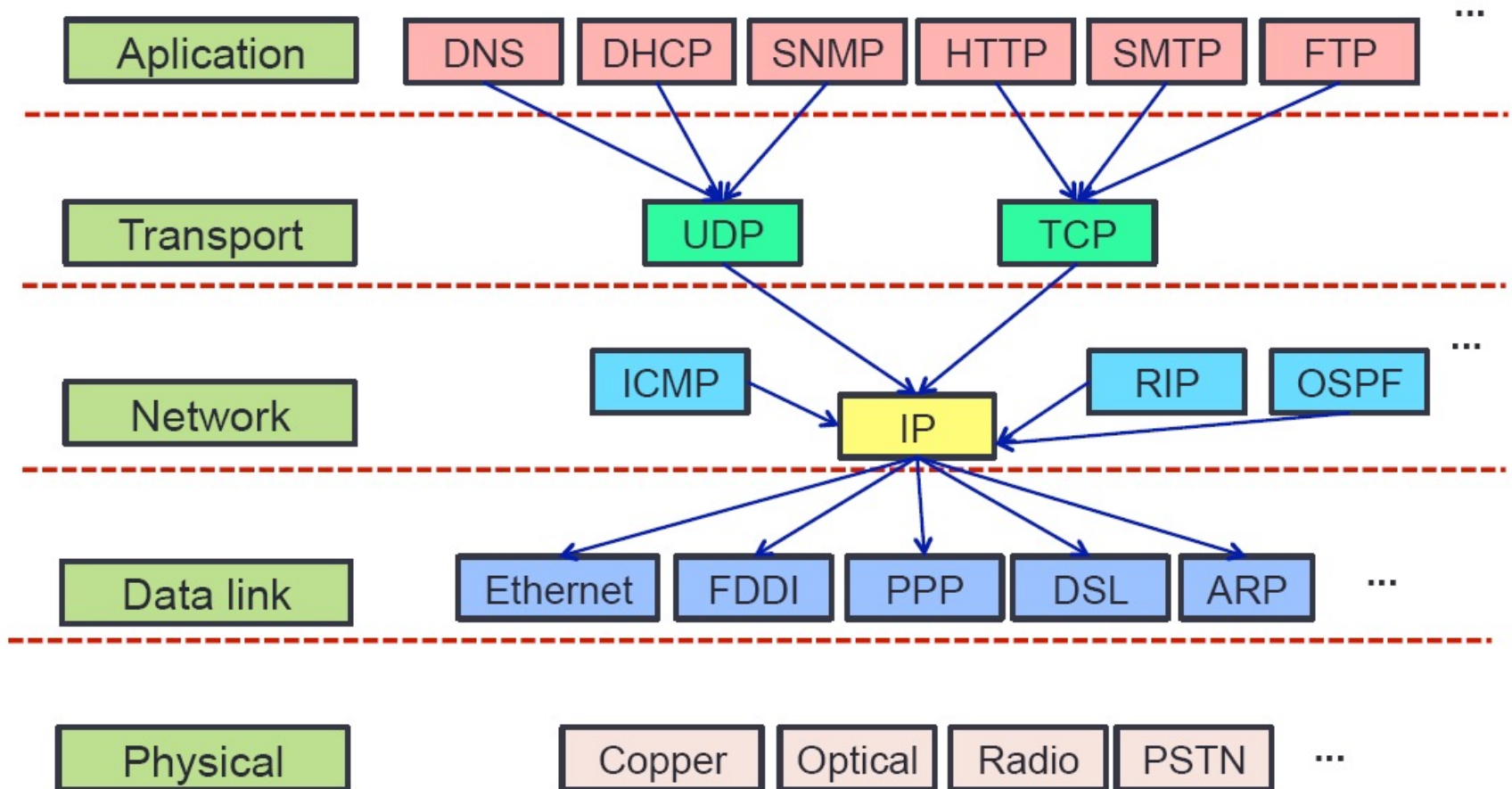| OSI model		TCP/IP model
Application layer		**Application** **HTTP, FTP, SMTP...**
Presentation layer		
Session layer		
Transport layer		TCP / UDP
Network layer		IP
Datalink layer		Network Interface
Physical layer		Physical
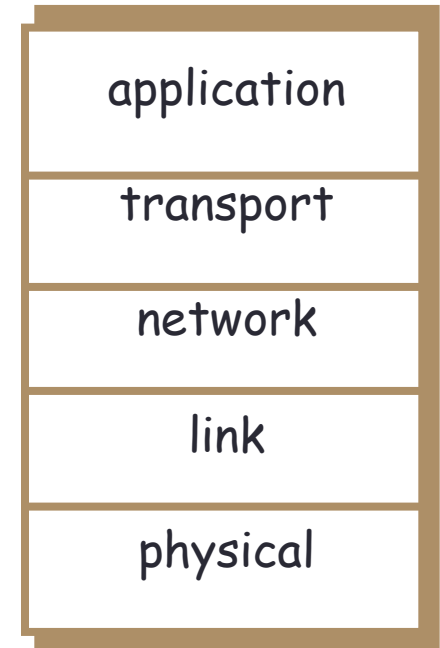
OSI and TCP/IP models

- OSI model: reference model
- TCP model: Internet model
  - Transport layer: TCP/UDP
  - Network layer: IP + routing protocols.

Giao thức tầng mạng

transport layer: TCP, UDP

Tầng mạng

Giao thức định tuyến
• Tìm đường
• RIP, OSPF, BGP

ICMP
• Báo lỗi
• Kiểm tra trạng thái nút mạng

IP
• Định danh
• Đóng gói
• Chuyển tiếp
• QoS

link layer

physical layer

14

Internet protocols mapping on TCP/IP

Encapsulation

source

message	M
segment	H_t M
datagram	H_n H_t M
frame	H_l H_n H_t M

application
transport
network
link
physical

link
physical

switch

destination

	M
	H_t M
	H_n H_t M
	H_l H_n H_t M

application
transport
network
link
physical

H_n H_t M
H_l H_n H_t M

network
link
physical

H_n H_t M

router
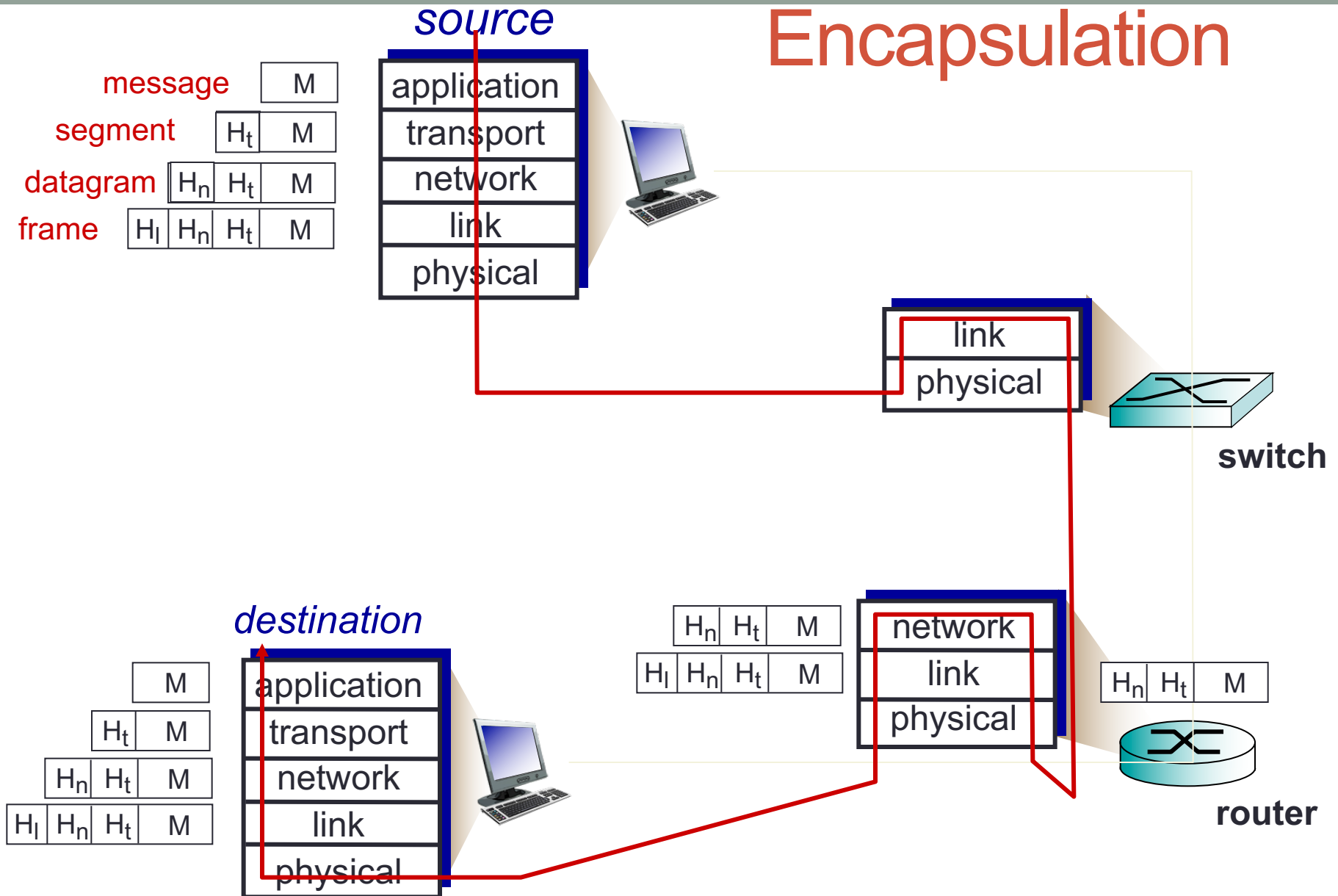
OSI and TCP/IP models

- Layering Makes it Easier

- Application programmer

  - Doesn't need to send IP packets

  - Doesn't need to send Ethernet frames

  - Doesn't need to know how TCP implements reliability

- Only need a way to pass the data down

  - Socket is the API to access transport layer functions

Application layer

- E-mail
- Web
- Instant messaging
- Remote login
- P2P file sharing
- Multi-user network games
- Streaming stored video clips

- Internet telephone
- Real-time video conference
- Massive parallel computing

Creating a network app

Write programs that

- run on different end systems and communicate over a network.
- e.g., Web: Web server software communicates with browser software

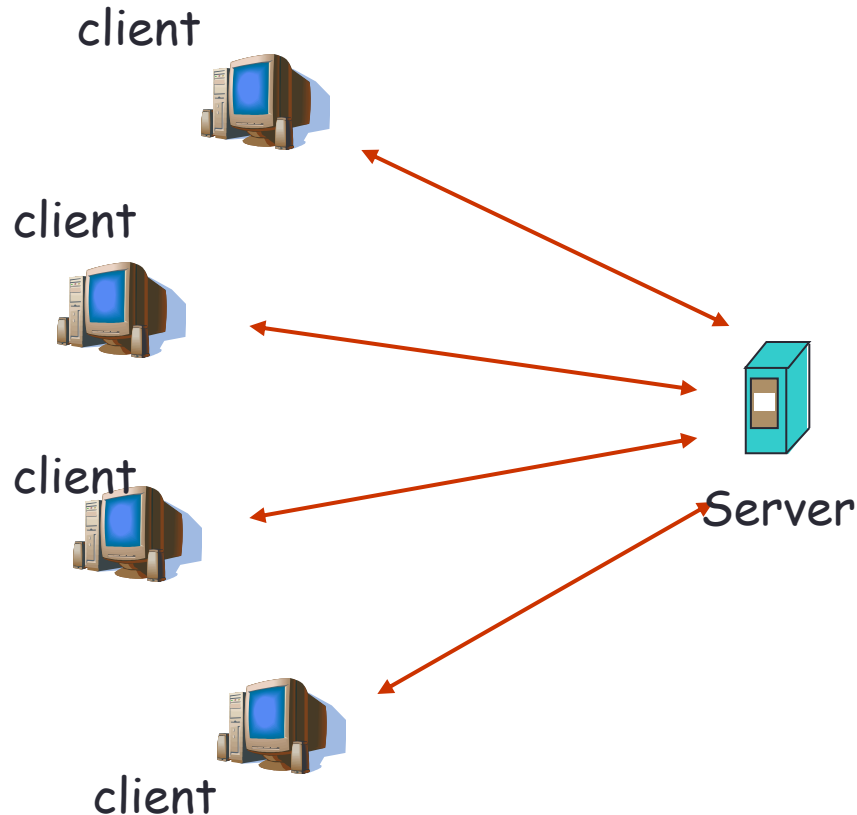**No software written for devices in network core**

- Network core devices do not function at app layer
- This design allows for rapid app development
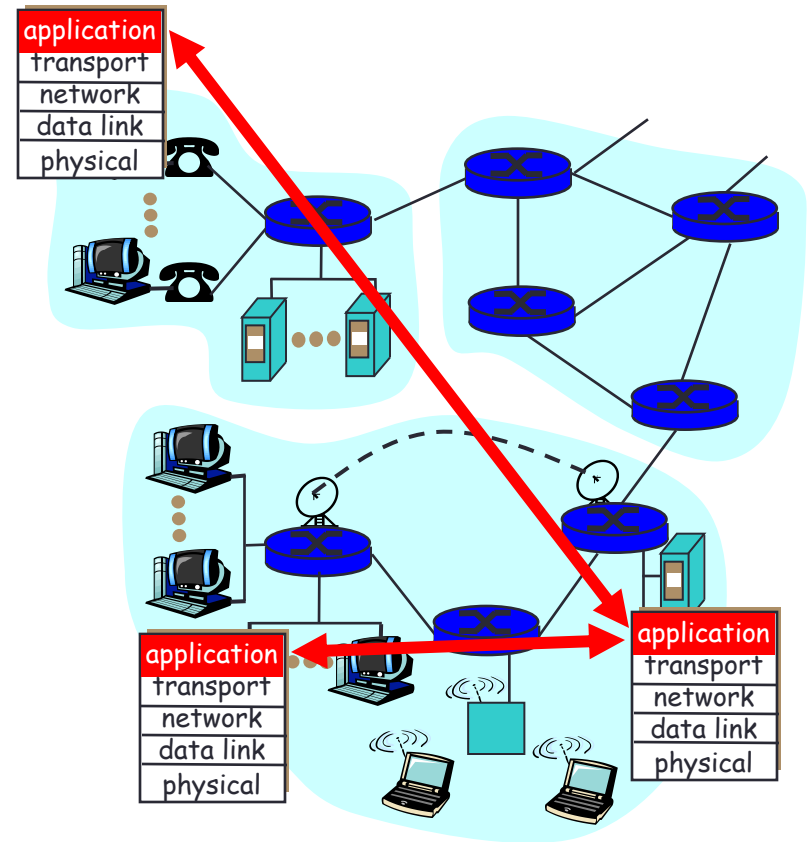
Application architectures

- Client-server
- Peer-to-peer (P2P)
- Hybrid of client-server and P2P

Client-server architecure

client

client

client

client

Server

server:
- "Always" online waiting for requests from clients
- Permanent IP address

clients:
- Request services from server
- May have dynamic IP addresses
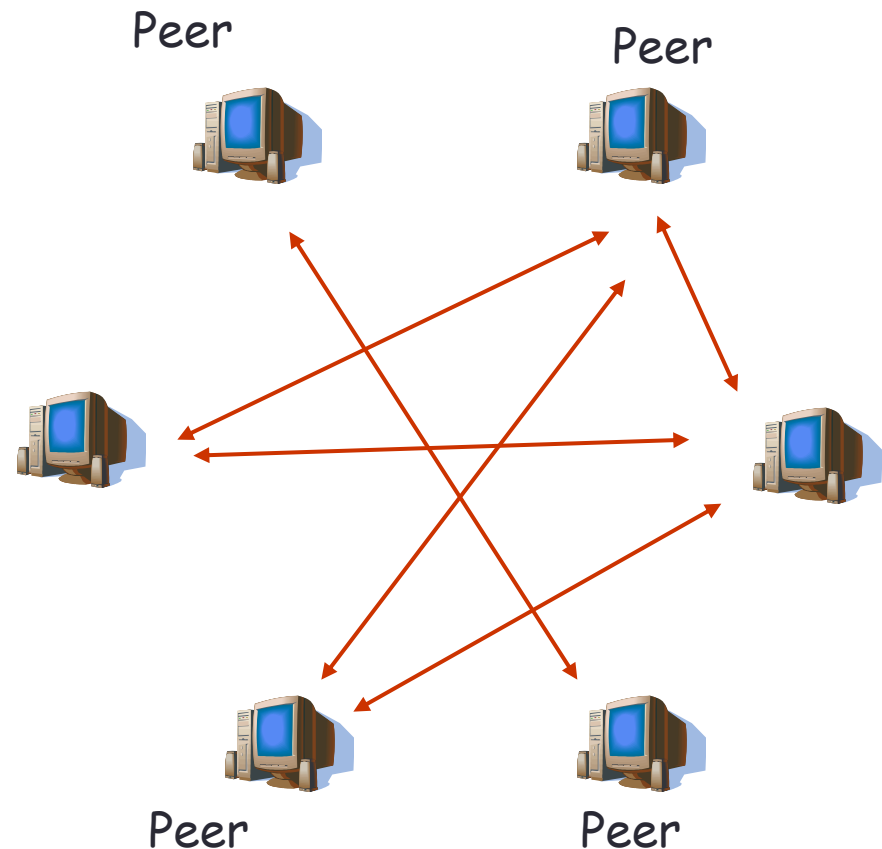- Do not communicate directly with each other

E.g: web, mail…

Pure P2P architecture

• No central server
• Peers have equal role
• Peers can communicate directly to each other
• Peers do not need to be always online
Example: Gnutella
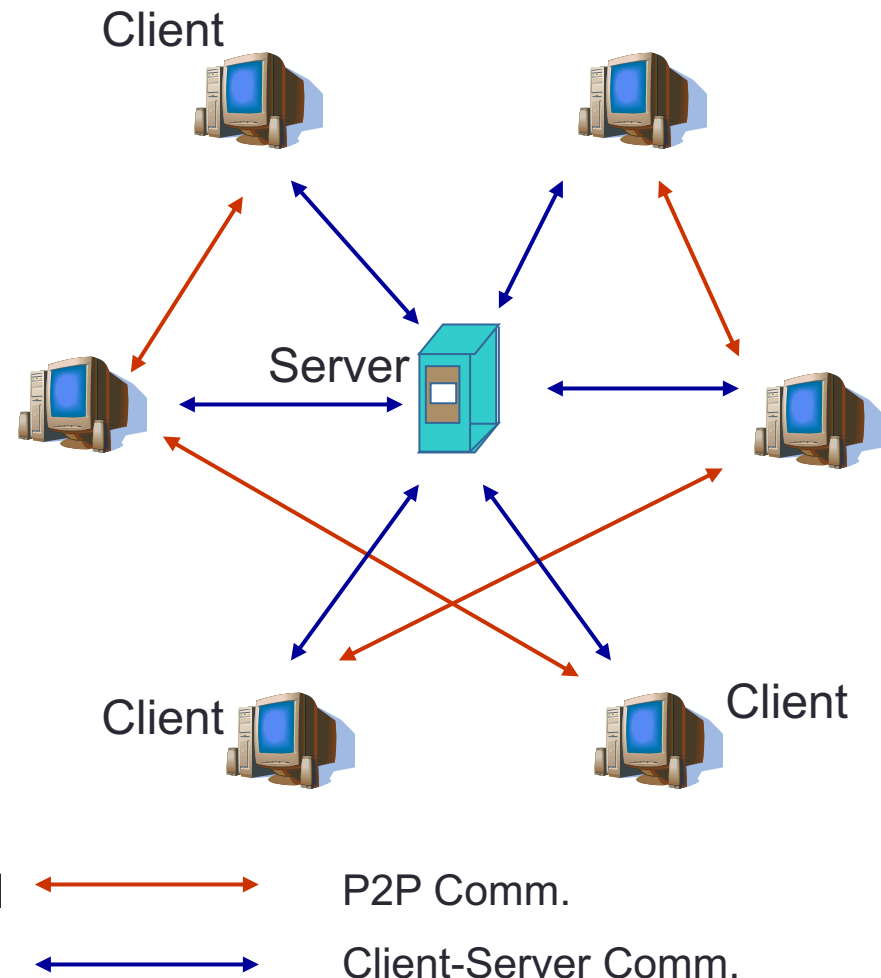
Highly scalable

But difficult to manage

Peer

Peer

Peer

Peer

Hybrid of client-server and P2P
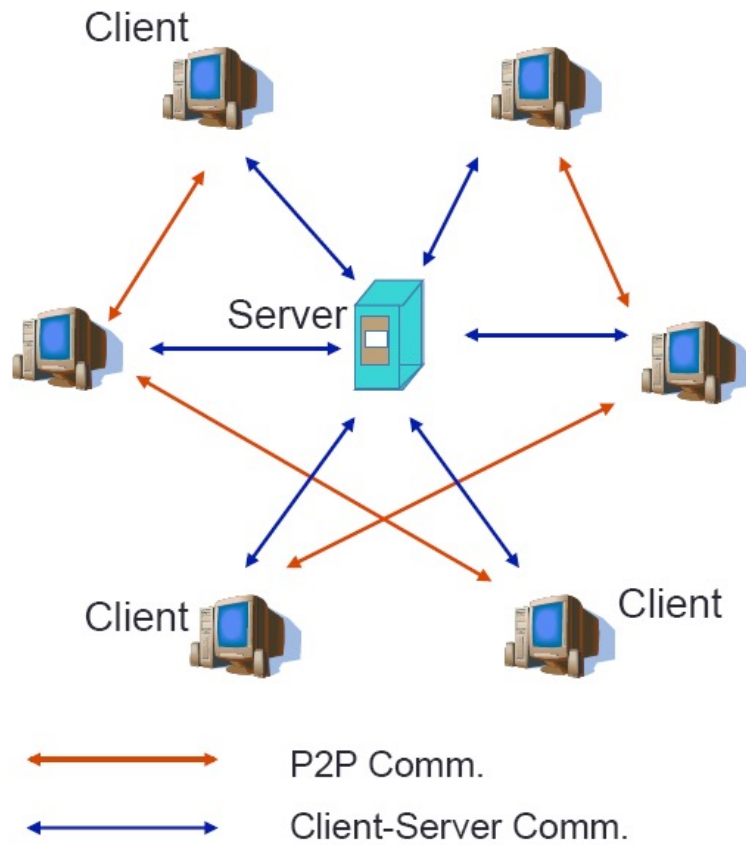
BitTorrent

- File transfer P2P
- File search centralized:
    - Peers register content at central server
    - Peers query same central server to locate content

Instant messaging

- Chatting between two users is P2P
- Presence detection/location centralized:
    - User registers its IP address with central server when it comes online
    - User contacts central server to find IP addresses of buddies

Client

Client

Server

Client

Client

← → P2P Comm.

← → Client-Server Comm.

Hybrid of client-server and P2P

Client

Server

Client   Client

→ P2P Comm.

→ Client-Server Comm.

- Central server manages user accounts, authentification, stores data for searching process …
- Clients communicate directly after authentication process.
- E.g. Skype
  - Server manages login process.
  - Messages, voices are transmitted directly between servers.

Processes communicating

Process: program running within a host.

- Within same host, two processes communicate using inter-process communication (defined by OS).
- Processes in different hosts communicate by exchanging messages
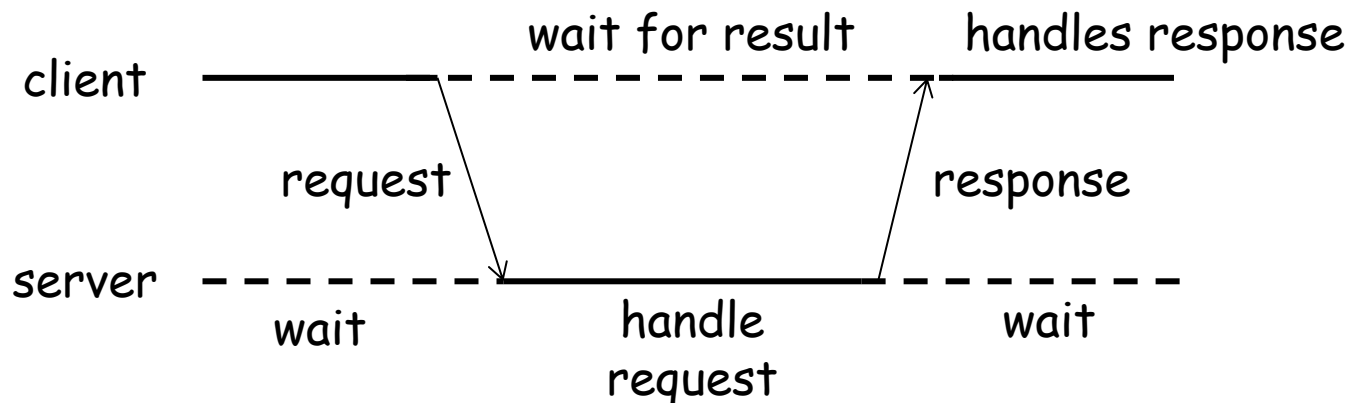
Client process: process that initiates communication

Server process: process that waits to be contacted

❑ Note: applications with P2P architectures have client processes & server processes
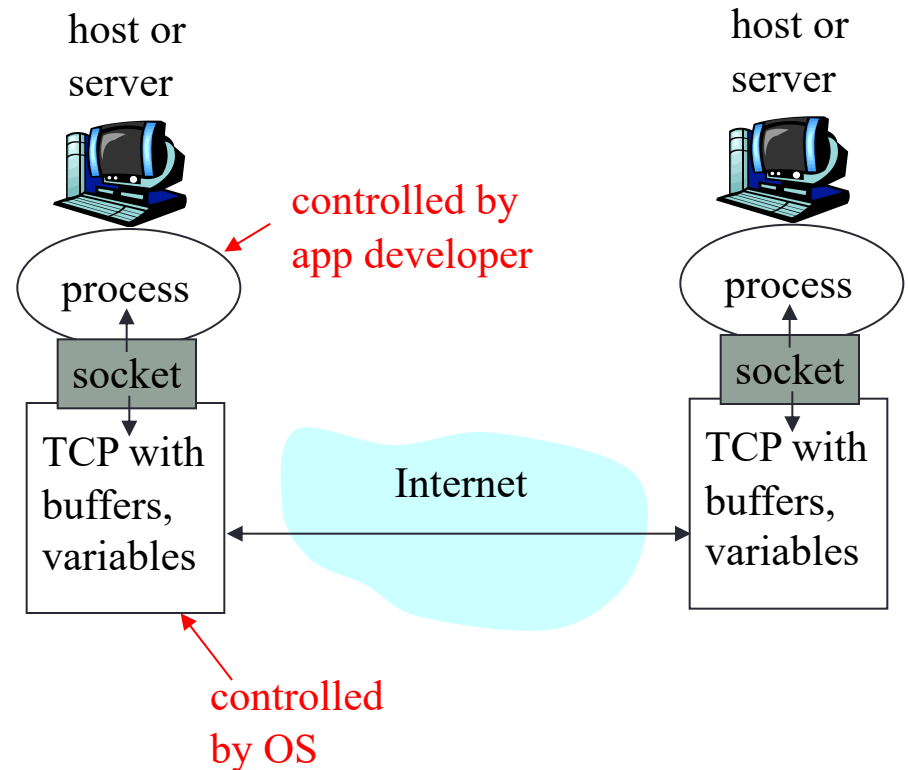
Processes communicating

- Client process: sends request
- Server process: replies response
- Typically: single server - multiple clients
- The server does not need to know anything about the client
- The client should always know something about the server
  - at least the socket address of the server

```
                        wait for result      handles response
client ─────────────╲ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─↗───────────────
                     ╲                    ╱
            request   ╲                  ╱   response
                       ╲                ╱
server ─ ─ ─ ─ ─ ─ ─ ─ ↓────────────── ─ ─ ─ ─ ─ ─ ─ ─
            wait           handle            wait
                          request
```

Sockets
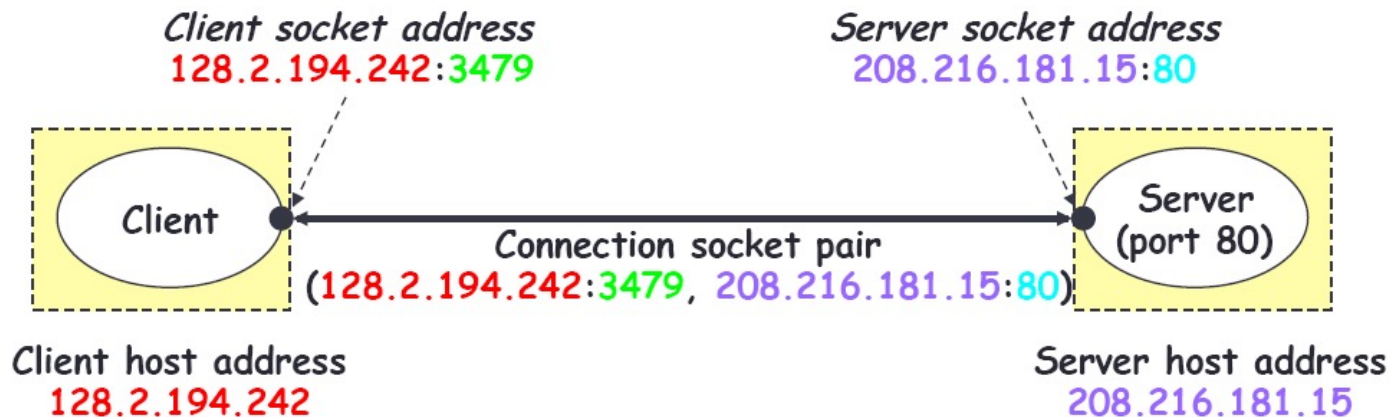
- process sends/receives messages to/from its socket
- Defined by
  - Port number
  - IP Address } Socket address
  - TCP/UDP

host or server

process

controlled by app developer

socket

TCP with buffers, variables

Internet

controlled by OS

host or server

process

socket

TCP with buffers, variables

Internet connections (TCP/IP)

- Address the machine on the network
  - By IP address
- Address the process/application
  - By the "port"-number
- The pair of *IP-address* + *port* – makes up a *"socket-address"*

Client socket address
128.2.194.242:3479

Server socket address
208.216.181.15:80

Client

Server
(port 80)

Connection socket pair
(128.2.194.242:3479, 208.216.181.15:80)

Client host address
128.2.194.242

Server host address
208.216.181.15

Note: *3479 is an ephemeral port allocated by the kernel*

Note: *80 is a well-known port associated with Web servers*

Internet connections (TCP/IP)

- Need to open two sockets of both sides
  - Client socket
  - Server socket
- Client application send/receive data to server through client socket
- Server application send/receive data to client through client socket
- Make two sockets talk to each other.

App-layer protocol defines

- Types of messages exchanged, eg, request & response messages
- Syntax of message types: what fields in messages & how fields are delineated
- Semantics of the fields, ie, meaning of information in fields
- Rules for when and how processes send & respond to messages

What transport service does an app need?

Data loss

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

Timing

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

Bandwidth

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

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
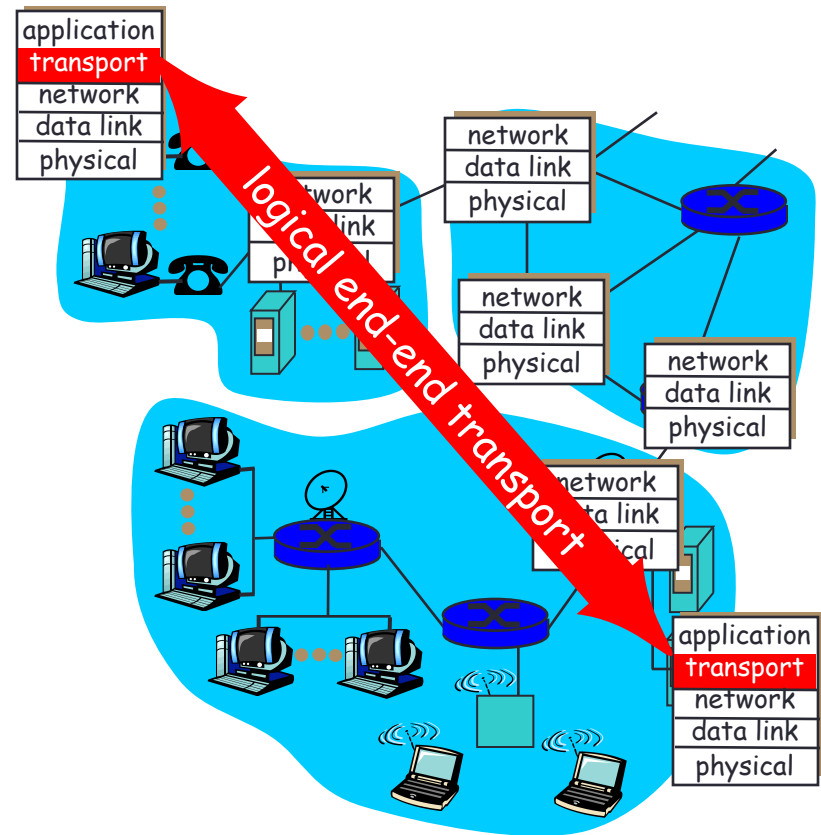
Internet transport protocols services

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

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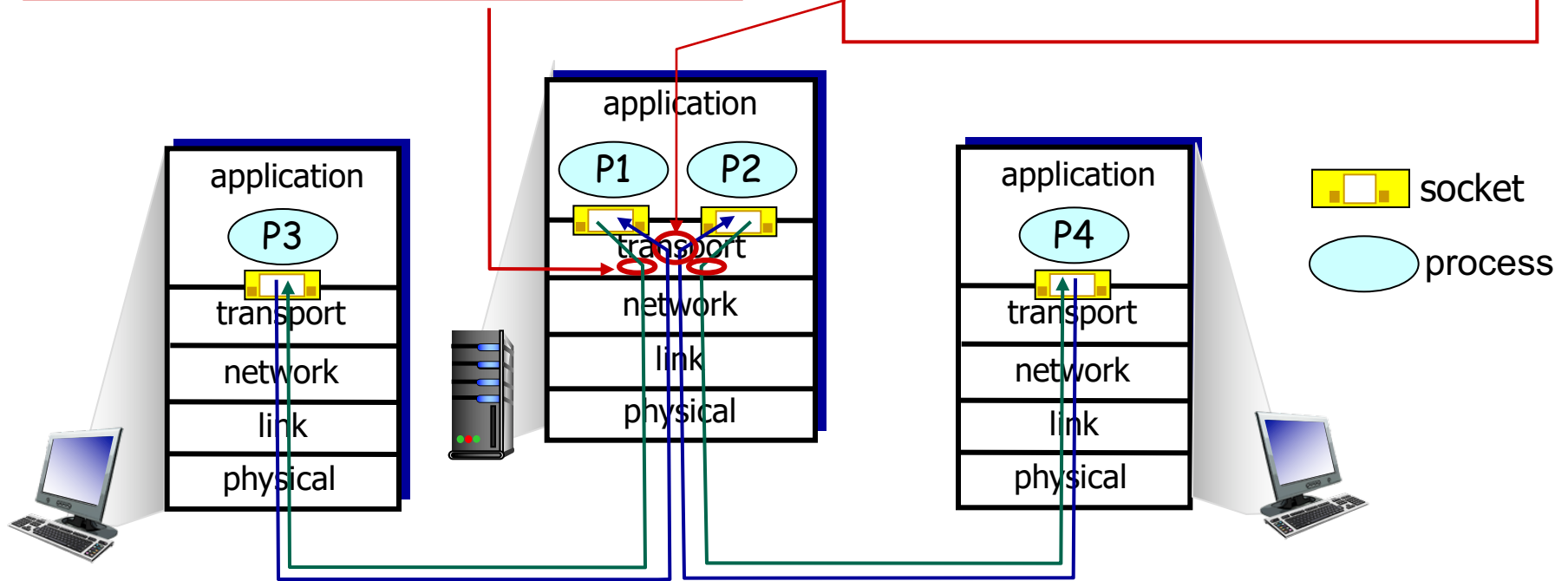
Multiplexing/demultiplexing
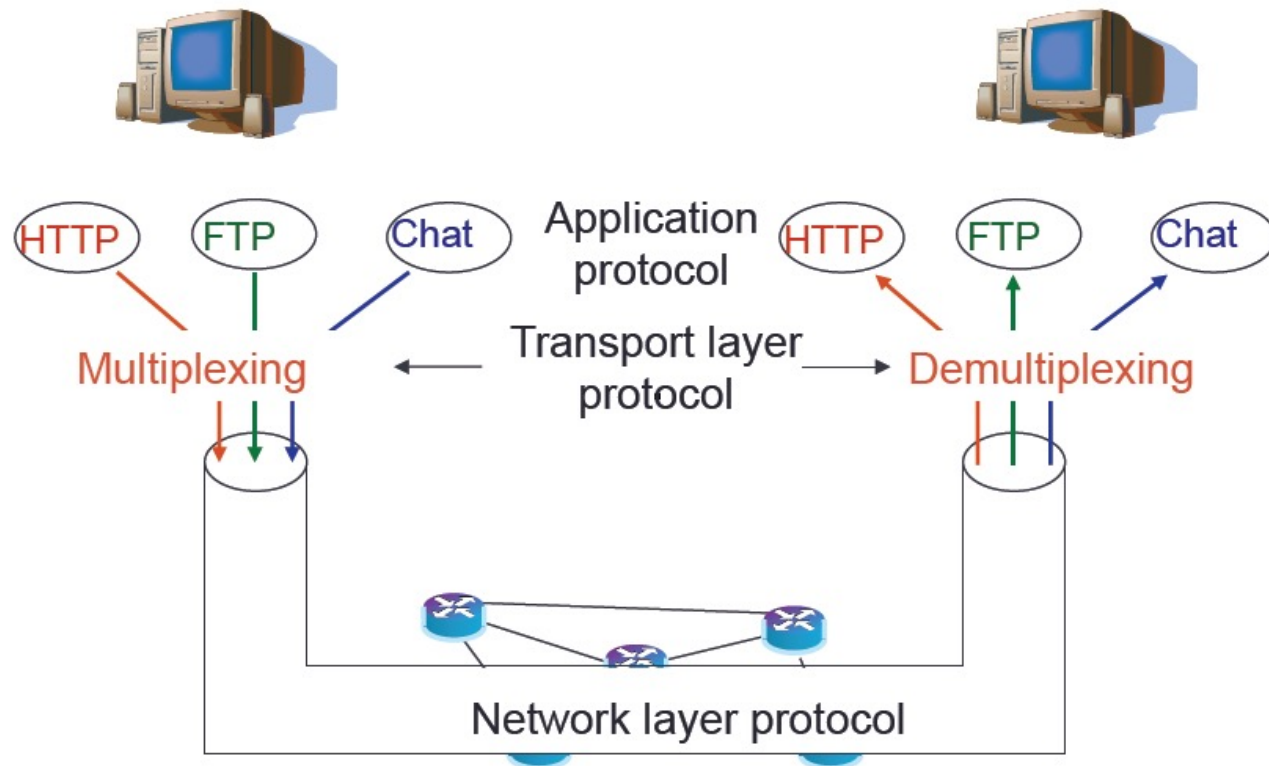
multiplexing at sender:
handle data from multiple
sockets, add transport header
(later used for demultiplexing)

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

application

P1   P2

application

P3

transport

network

link

physical

transport

network

link

physical

application

P4

transport

network

link

physical

socket

process

Transport layer Mux/Demux

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

32 bits

source port #	dest port #
other header fields	
application data (payload)	
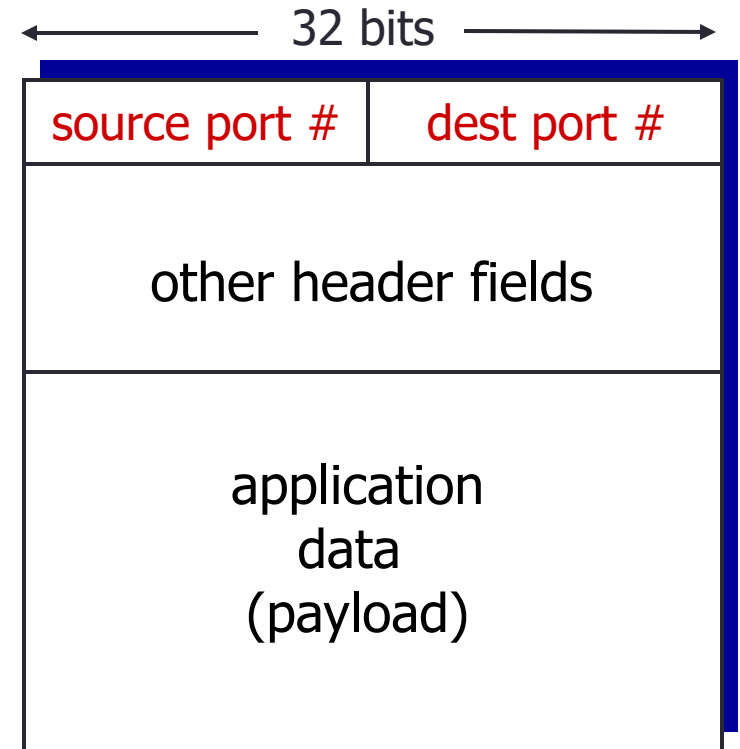
TCP/UDP segment format

UDP: User Datagram Protocol [RFC 768]

- "best effort" service, UDP segments may be:
  - lost
  - delivered out of order to app
- *connectionless:*
  - no handshaking between UDP sender, receiver
  - each UDP segment handled independently of others

Why is there a UDP?

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

UDP demultiplexing

- Create sockets with port numbers:

```
mySocket = socket(AF_INET,
  SOCK_DGRAM, 0)
```

- UDP socket identified by two-tuple:
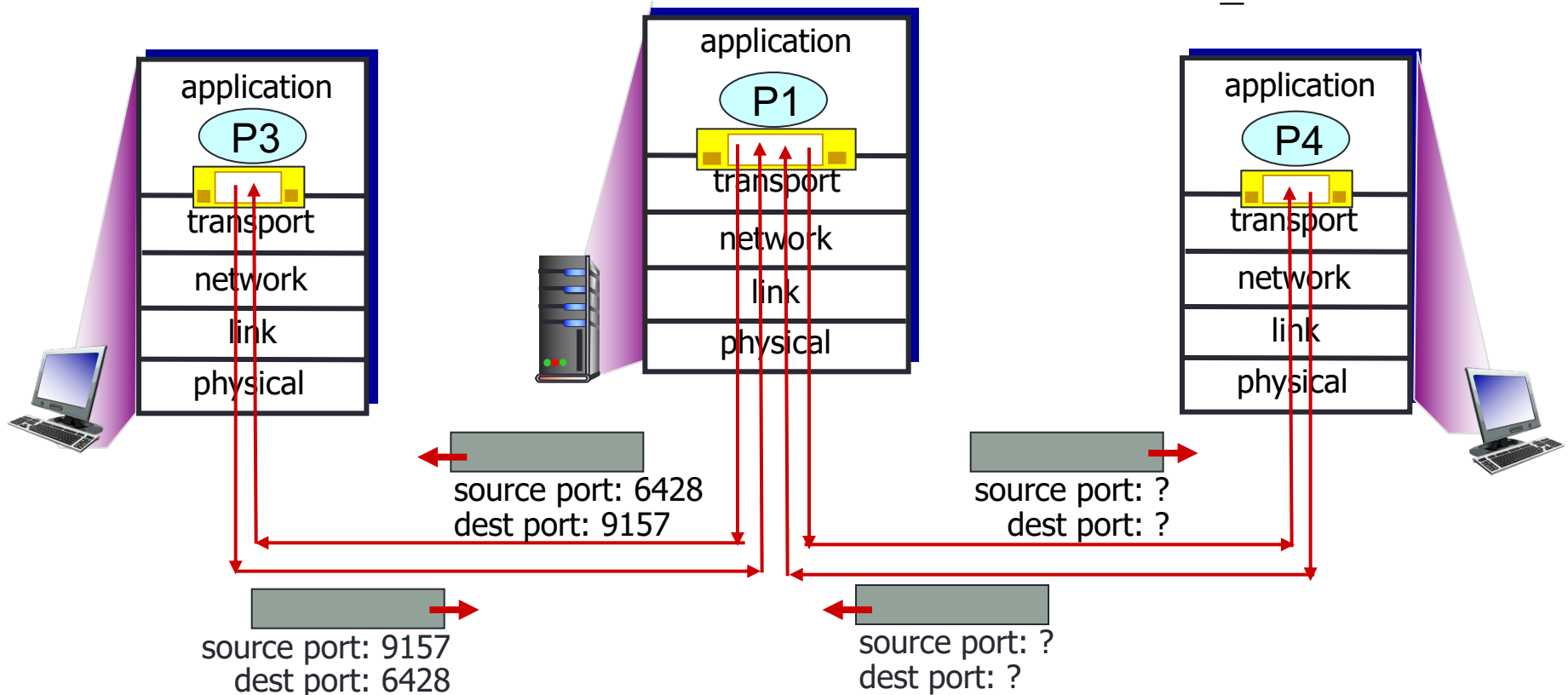
(dest IP address, dest port number)

- When host receives UDP segment:
  - checks destination port number in segment
  - directs UDP segment to socket with that port number
- IP datagrams with different source IP addresses and/or source port numbers directed to same socket

UDP demux


```
serverSocket =
    socket(AF_INET,
    SOCK_DGRAM, 0);

bind(…)
```

```
mySocket =
    socket(AF_INET,
    SOCK_DGRAM, 0);
```

```
mySocket =
    socket(AF_INET,
    SOCK_DGRAM, 0);
```

application

P3

transport

network

link

physical

application

P1

transport

network

link

physical

application

P4

transport

network

link

physical

source port: 6428
dest port: 9157

source port: ?
dest port: ?

source port: 9157
dest port: 6428

source port: ?
dest port: ?

40

# TCP: Overview   RFCs: 793, 1122, 1323, 2018, 2581

- point-to-point:
  - one sender, one receiver
- reliable, in-order *byte steam:*
  - no "message boundaries"
- pipelined:
  - TCP congestion and flow control set window size
- *send & receive buffers*

- full duplex data:
  - bi-directional data flow in same connection
  - MSS: maximum segment size
- connection-oriented:
  - handshaking (exchange of control msgs) init's sender, receiver state before data exchange
- flow controlled:
  - sender will not overwhelm receiver

application
writes data

socket
door

TCP
send buffer

application
reads data

socket
door

TCP
receive buffer

segment
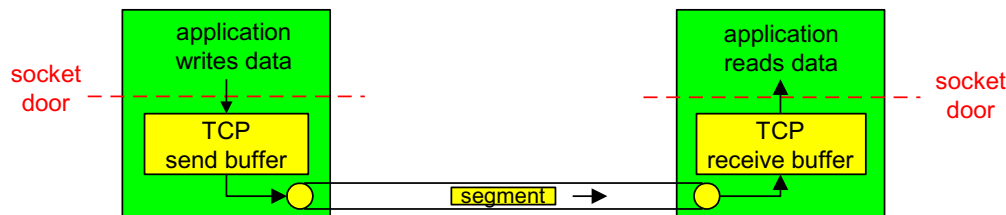
# TCP Connection Management: Setup

- Connection oriented protocol
  - 3-step connection opening
- Reliable protocol
  - Re-transmission on error
- Flow control
- Congestion control

A   B

SYN

ACK/SYN

ACK
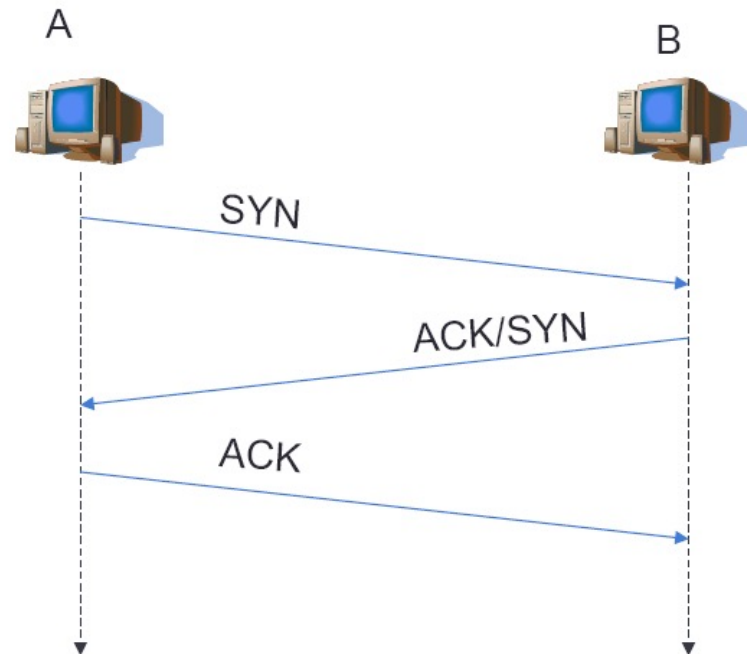
Connection-oriented demux

- TCP socket identified by 4-tuple:
  - source IP address
  - source port number
  - dest IP address
  - dest port number
- recv host 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

Connection-oriented demux: example
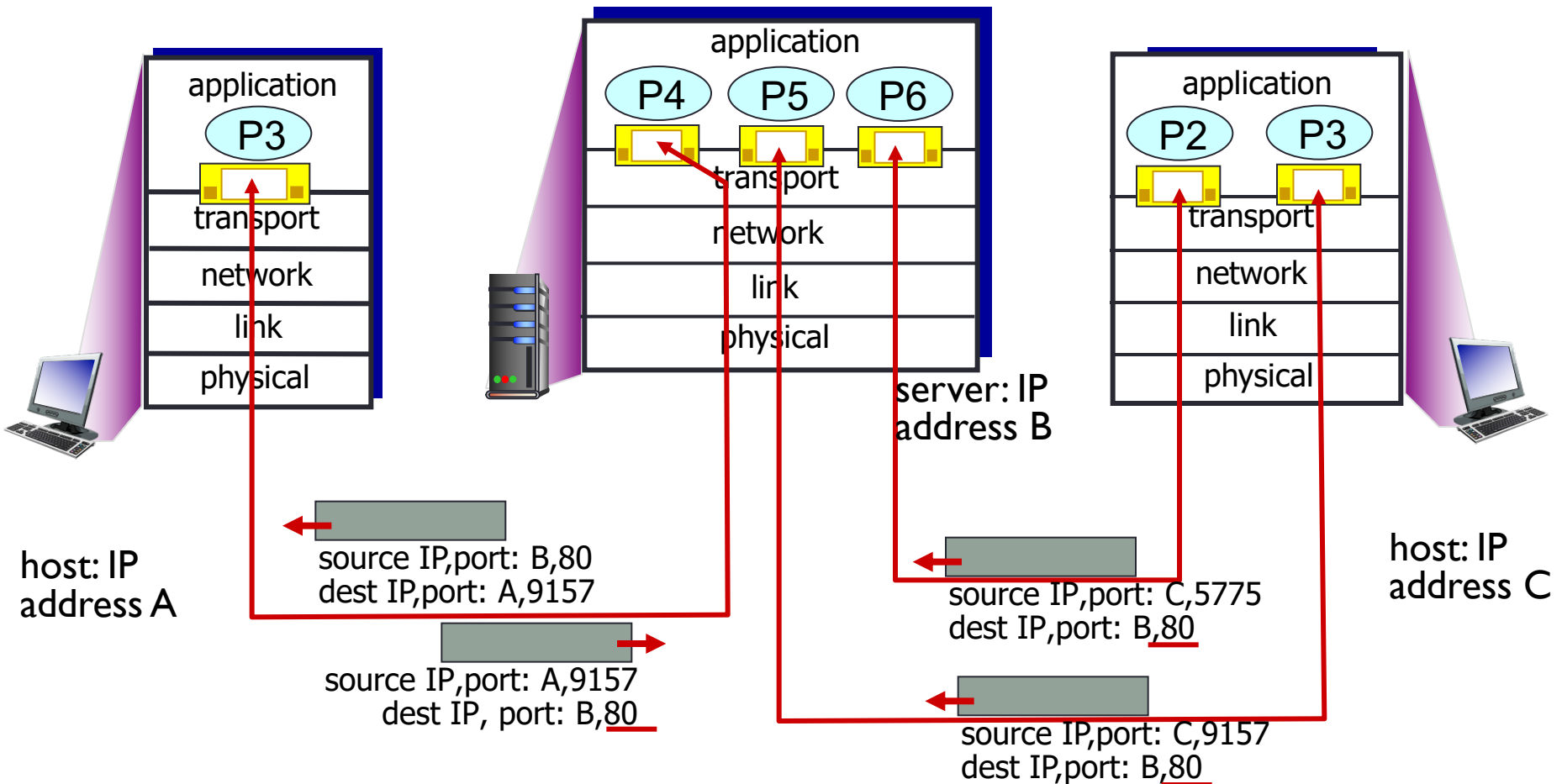
three segments, all destined to IP address: B,
dest port: 80 are demultiplexed to *different* sockets

45

Connection-oriented demux: example

threaded server

application

P4

transport

network

link

physical

server: IP
address B

application

P3

transport

network

link

physical

host: IP
address A

application

P2   P3

transport

network

link

physical

host: IP
address C

source IP,port: B,80
dest IP,port: A,9157

source IP,port: A,9157
dest IP, port: B,80

source IP,port: C,5775
dest IP,port: B,80

source IP,port: C,9157
dest IP,port: B,80
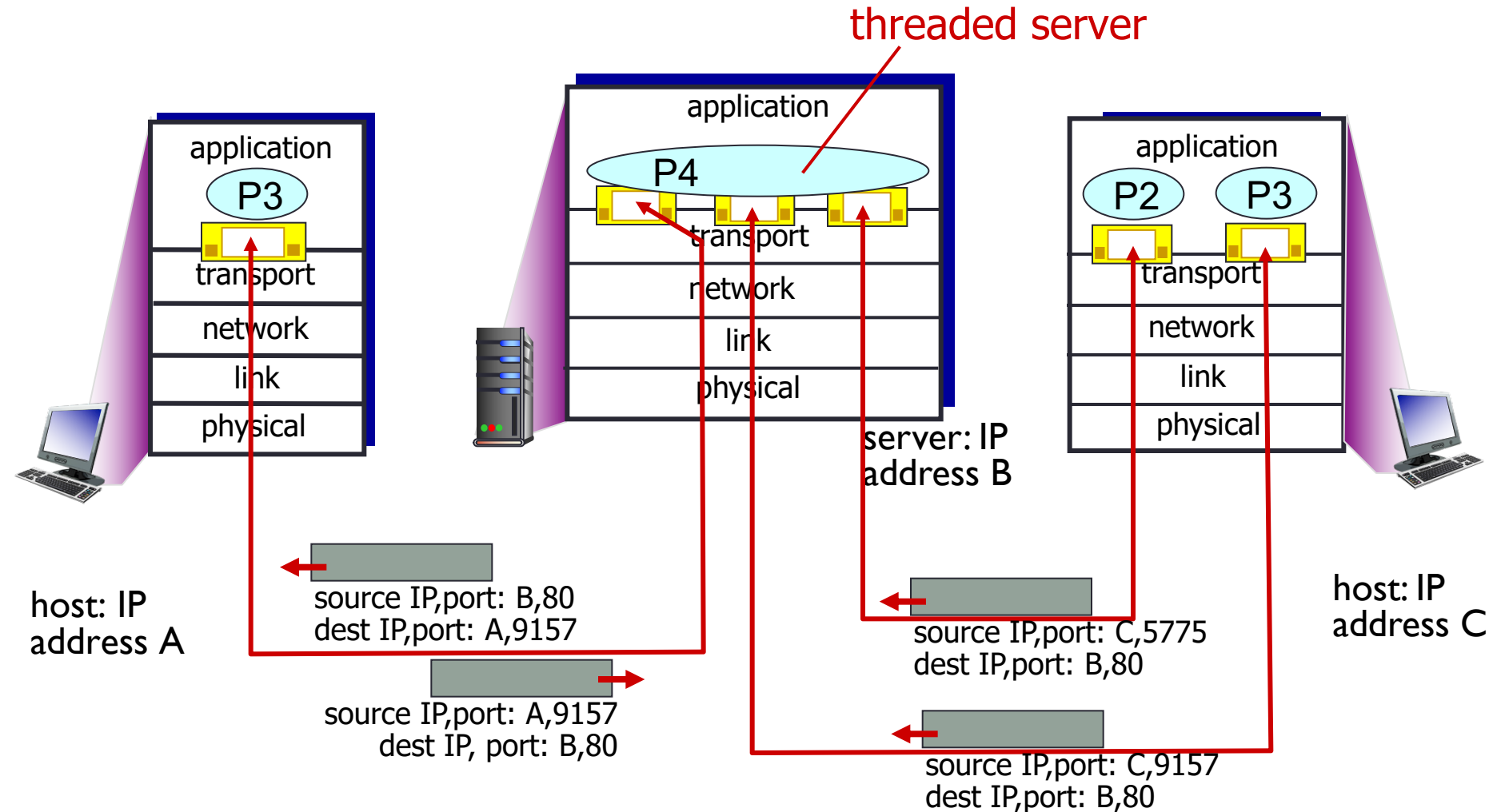
Network layer

- 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
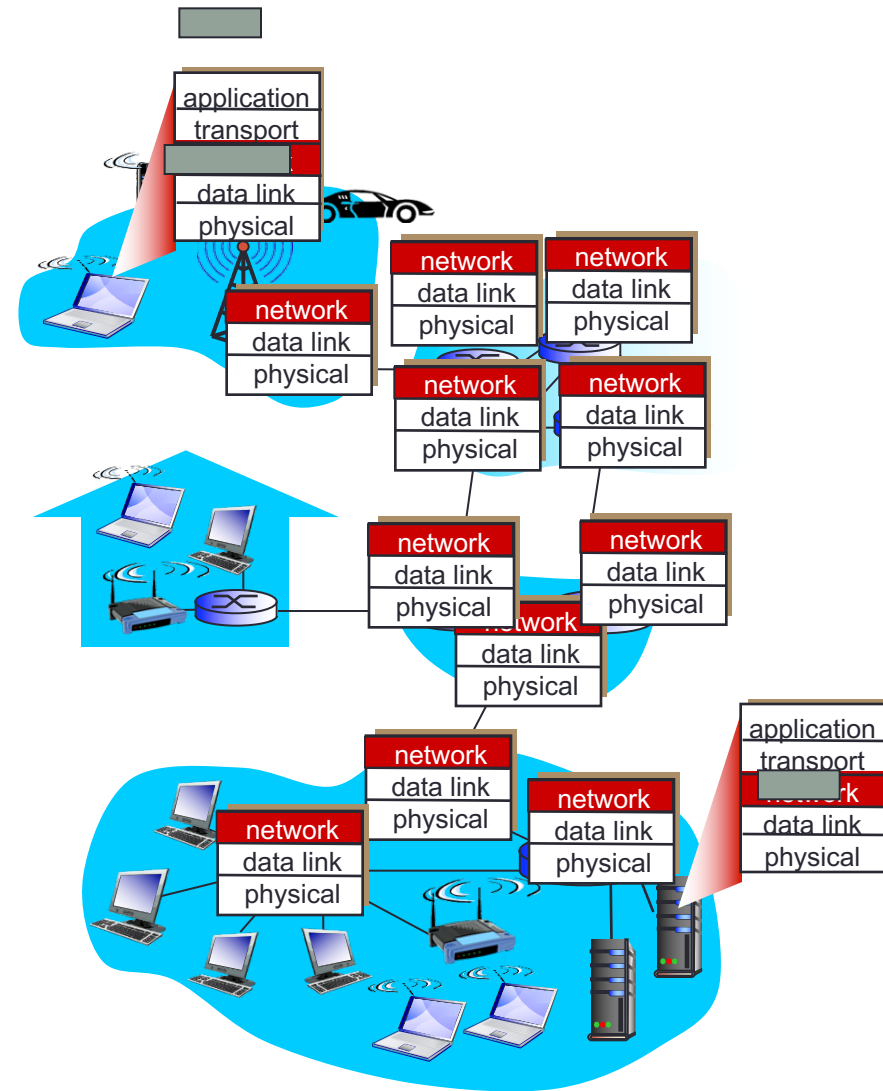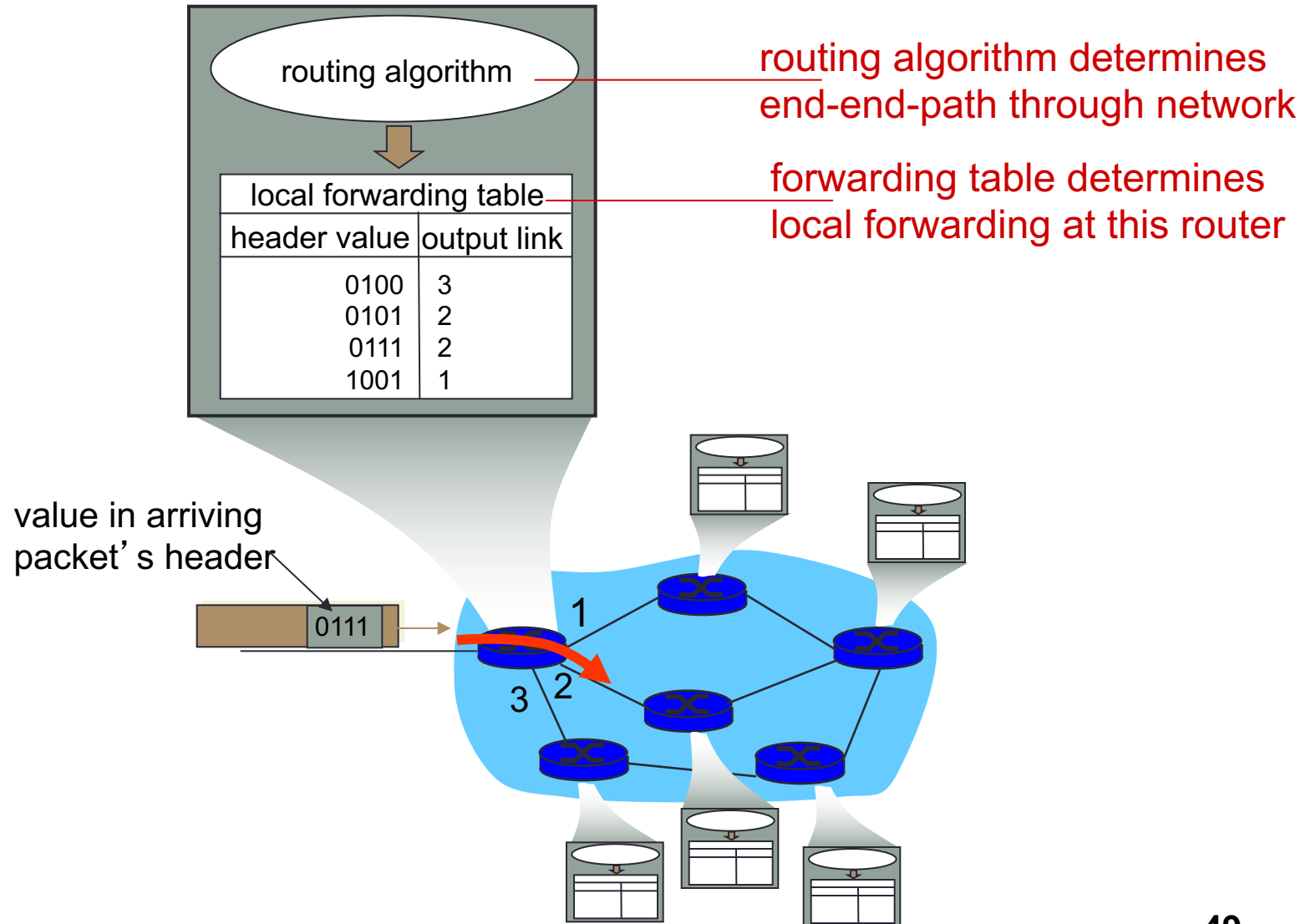
Two key network-layer functions

- *forwarding:* move packets from router's input to appropriate router output

- *routing:* determine route taken by packets from source to dest.
  - *routing algorithms*

analogy:

- *routing:* process of planning trip from source to dest

- *forwarding:* process of getting through single interchange

Interplay between routing and forwarding

routing algorithm determines end-end-path through network

forwarding table determines local forwarding at this router

local forwarding table	
header value	output link
0100	3
0101	2
0111	2
1001	1

value in arriving packet's header

0111

Why an internet layer?
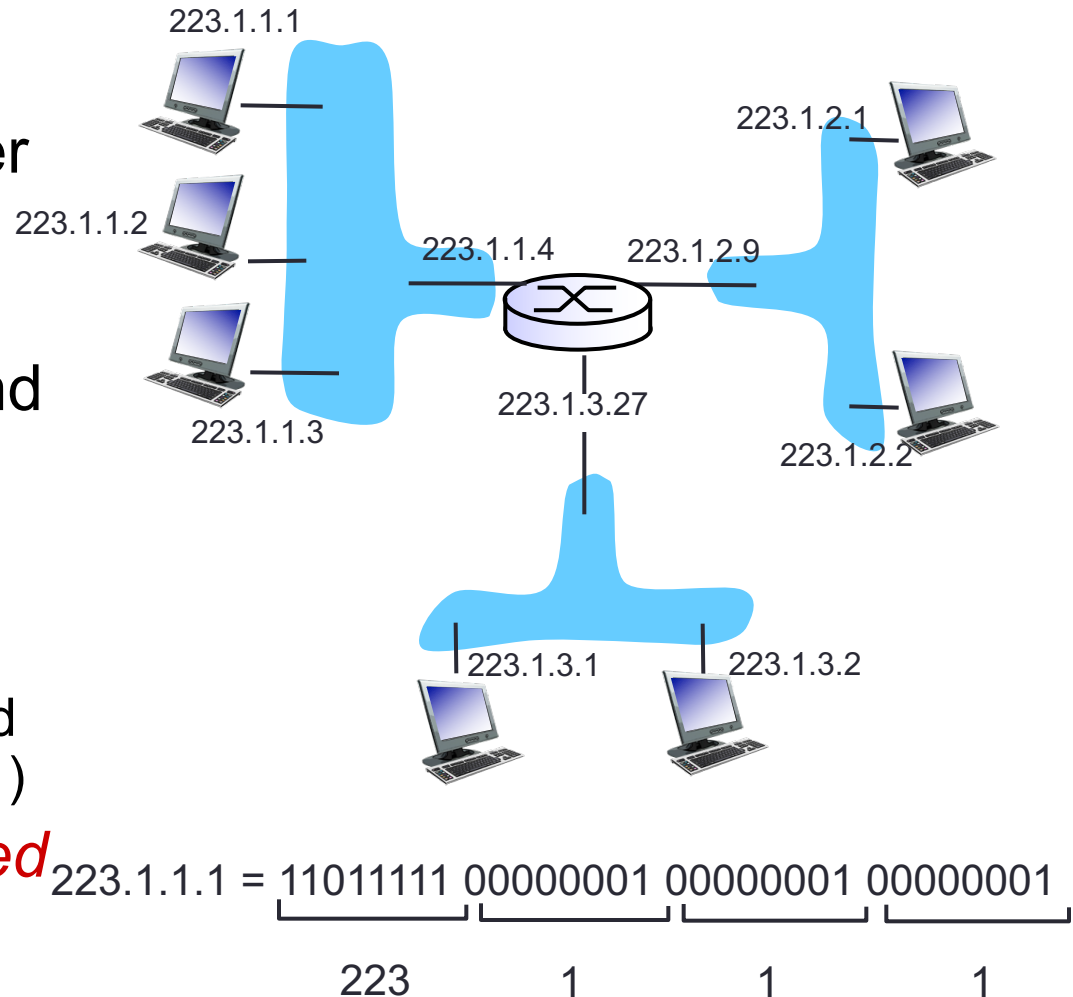
☐ **Why not one big flat LAN?**
  ○ Different LAN protocols
  ○ Flat address space not scalable

☐ **IP provides:**
  ○ Global addressing
  ○ Scaling to WANs
  ○ Virtualization of network isolates end-to-end protocols from network details/changes

email WWW phone…
SMTP HTTP RTP…
TCP UDP…
IP
ethernet PPP…
CSMA async sonet…
copper fiber radio…
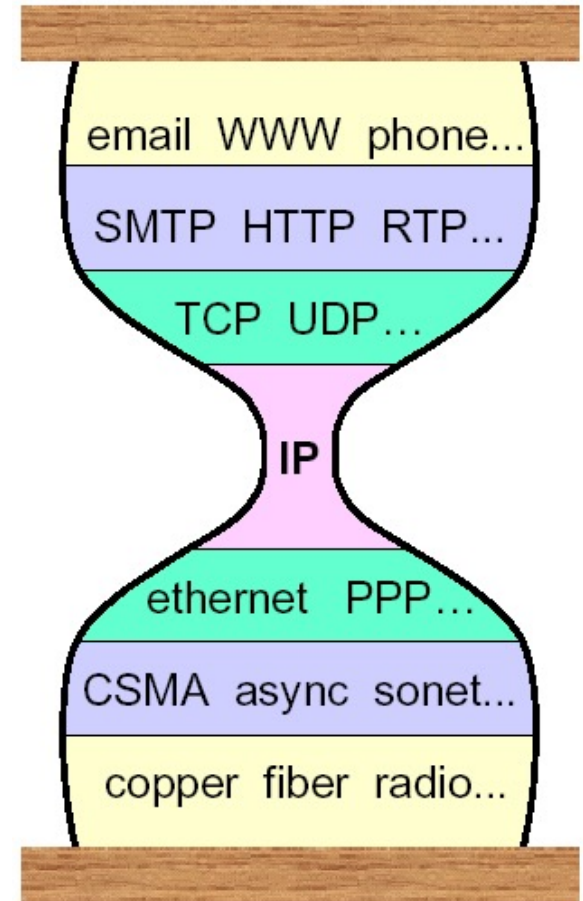
"hourglass model"
(Steve Deering)

IP addressing: introduction

- *IP address:* 32-bit identifier for host, router *interface*

- *interface:* connection between host/router and physical link
  - router's typically have multiple interfaces
  - host typically has one or two interfaces (e.g., wired Ethernet, wireless 802.11)

- *IP addresses associated with each interface*

223.1.1.1
223.1.1.2
223.1.1.3
223.1.1.4
223.1.2.9
223.1.2.1
223.1.2.2
223.1.3.27
223.1.3.1
223.1.3.2

223.1.1.1 = 11011111 00000001 00000001 00000001

223   1   1   1

IP addressing: "class-full"

class

|		
A	0 network ⎪ host	1.0.0.0 to 127.255.255.255
B	10 ⎪ network ⎪ host	128.0.0.0 to 191.255.255.255
C	110 ⎪ network ⎪ host	192.0.0.0 to 223.255.255.255
D	1110 ⎪ multicast address	224.0.0.0 to 239.255.255.255

←——— 32 bits ———→
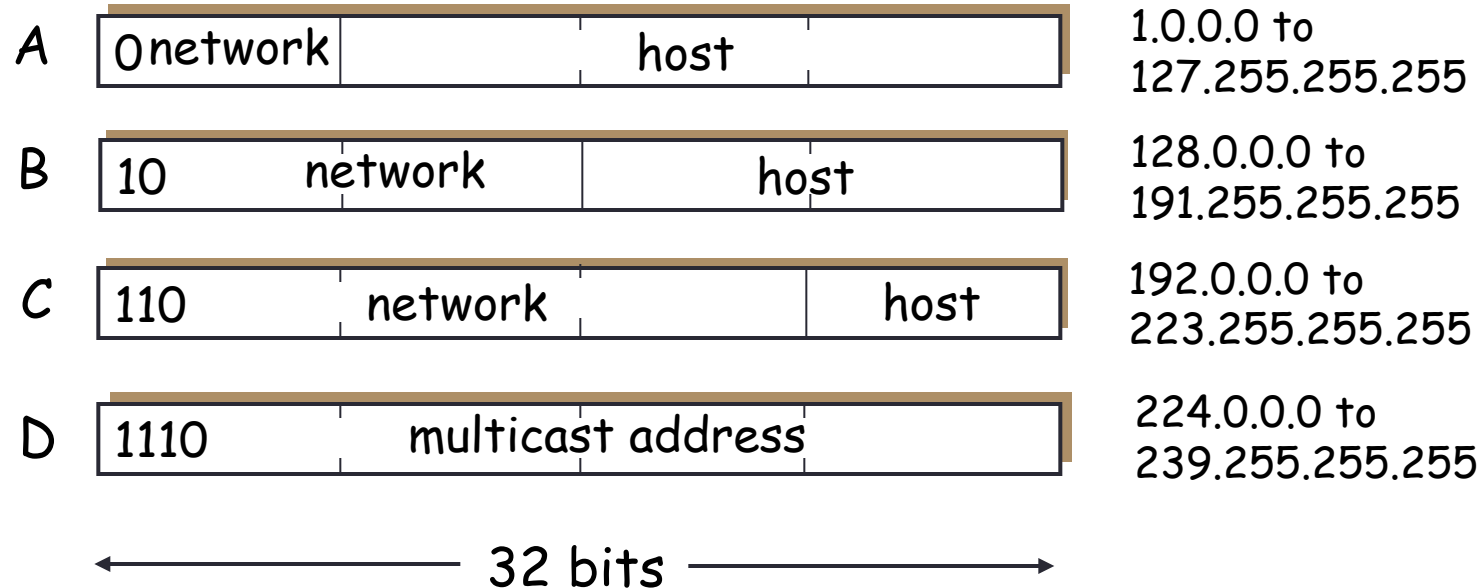
- Classful addressing:
  - inefficient use of address space, address space exhaustion
  - e.g., class B net allocated enough addresses for 65K hosts, even if only 2K hosts in that network

IP addressing: "class-less"

CIDR: Classless **I**nter**D**omain **R**outing

- subnet portion of address of arbitrary length
- address format: a.b.c.d/x, where x is # bits in subnet portion of address

subnet part ←————————————————————→ host part ←————→

11001000 00010111 0001000**0** 00000000

200.23.16.0/23
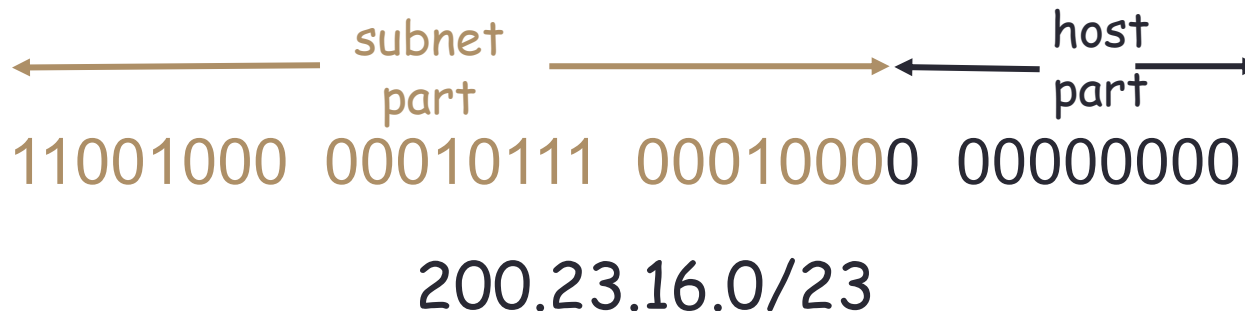
Address Allocation for Private Internets

- RFC1918

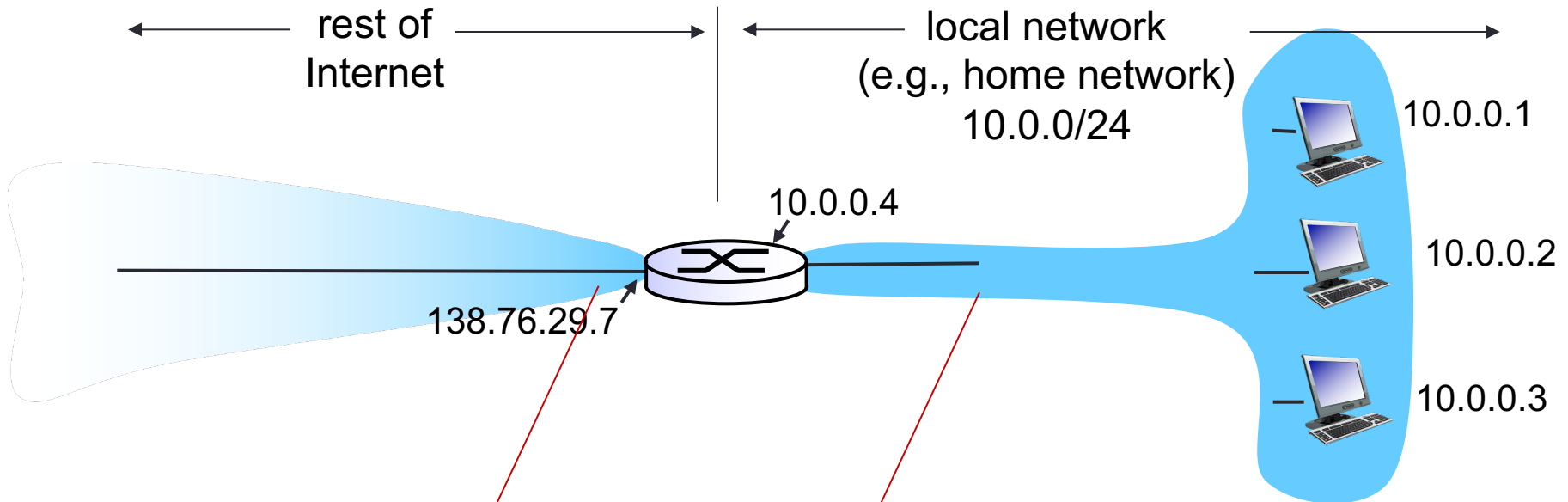|	
Private address	10.0.0.0/8 172.16.0.0/16 → 172.31.0.0/16 192.168.0.0/24 → 192.168.255.0 /24
Loopback address	127.0.0.0 /8
Multicast address	224.0.0.0 ～239.255.255.255

- Link local address: 169.254.0.0/16

IP

IP: Internet Protocol

- Forward data packet between distance network nodes (routers or hosts)
- Using routing table built by routing protocols such as OSPF, RIP …

- IP address
  - Is assigned to each network interface
  - IP v4: 32 bits
    - 133.113.215.10
  - IP v6: 128 bits
    - 2001:200:0:8803::53

- A host may have a domain name
  - Conversion IP <-> domain name: DNS
  - Ex: soict.hust.edu.vn <--> 202.191.56.65

NAT: network address translation

rest of Internet

local network (e.g., home network) 10.0.0/24

10.0.0.1

10.0.0.2

10.0.0.3

10.0.0.4

138.76.29.7

all datagrams *leaving* local network have *same* single source NAT IP address: 138.76.29.7,different source port numbers

datagrams with source or destination in this network have 10.0.0/24 address for source, destination (as usual)

NAT: network address translation

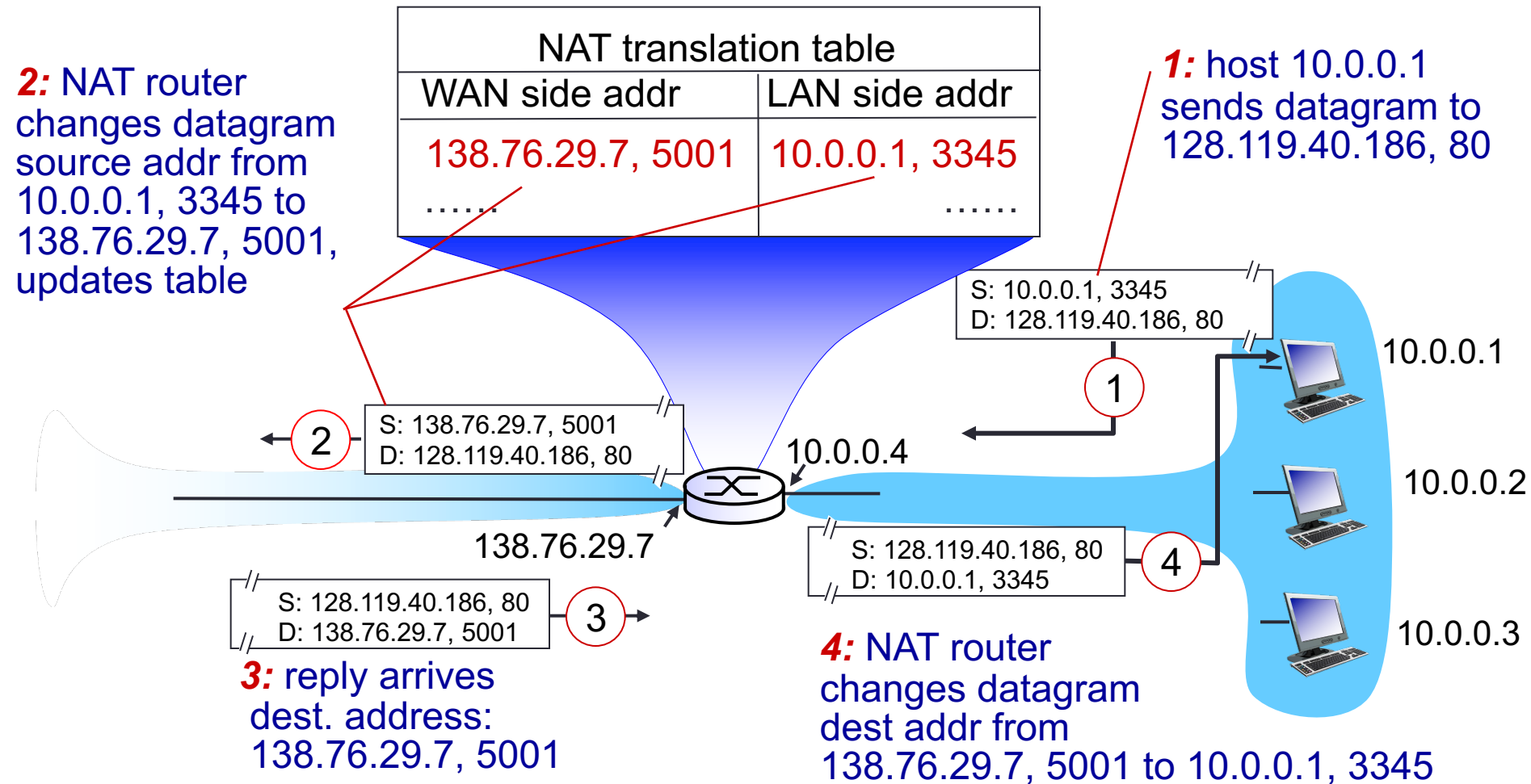*motivation:* local network uses just one IP address as far as outside world is concerned:

- range of addresses not needed from ISP: just one IP address for all devices
- can change addresses of devices in local network without notifying outside world
- can change ISP without changing addresses of devices in local network
- devices inside local net not explicitly addressable, visible by outside world (a security plus)

NAT: network address translation

implementation: NAT router must:

- *outgoing datagrams: replace* (source IP address, port #) of every outgoing datagram to (NAT IP address, new port #)

  …remote clients/servers will respond using (NAT IP address, new port #) as destination addr

- *remember (in NAT translation table)* every (source IP address, port #) to (NAT IP address, new port #) translation pair

- *incoming datagrams: replace* (NAT IP address, new port #) in dest fields of every incoming datagram with corresponding (source IP address, port #) stored in NAT table

NAT: network address translation

2: NAT router changes datagram source addr from 10.0.0.1, 3345 to 138.76.29.7, 5001, updates table

1: host 10.0.0.1 sends datagram to 128.119.40.186, 80

NAT translation table	
WAN side addr	LAN side addr
138.76.29.7, 5001	10.0.0.1, 3345
……	……

S: 10.0.0.1, 3345
D: 128.119.40.186, 80

1

10.0.0.1

S: 138.76.29.7, 5001
D: 128.119.40.186, 80

2

10.0.0.4

10.0.0.2

138.76.29.7

S: 128.119.40.186, 80
D: 10.0.0.1, 3345

4

S: 128.119.40.186, 80
D: 138.76.29.7, 5001

3

10.0.0.3

3: reply arrives dest. address: 138.76.29.7, 5001

4: NAT router changes datagram dest addr from 138.76.29.7, 5001 to 10.0.0.1, 3345

NAT: network address translation

- 16-bit port-number field:

  - 60,000 simultaneous connections with a single LAN-side address!

- NAT is controversial:

  - routers should only process up to layer 3

  - violates end-to-end argument

    - NAT possibility must be taken into account by app designers, e.g., P2P applications

 - address shortage should instead be solved by IPv6

NAT traversal problem

- client wants to connect to server with address 10.0.0.1
  - server address 10.0.0.1 local to LAN (client can't use it as destination addr)
  - only one externally visible NATed address: 138.76.29.7
- *solution1:* statically configure NAT to forward incoming connection requests at given port to server
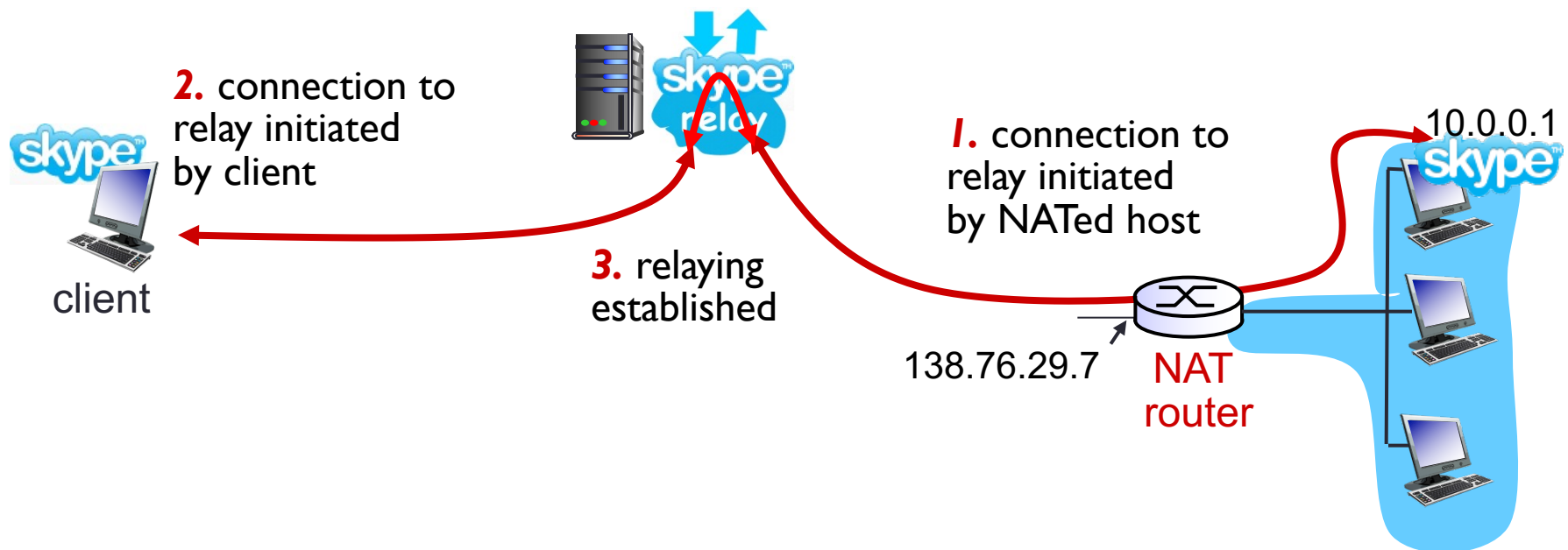 - e.g., (123.76.29.7, port 2500) always forwarded to 10.0.0.1 port 25000

client

?

10.0.0.1

10.0.0.4

138.76.29.7   NAT router
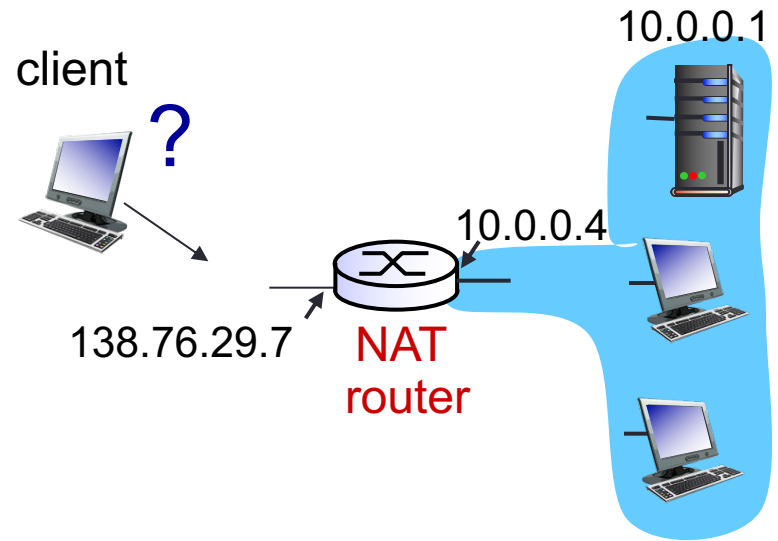
NAT traversal problem

- *solution 2:* Universal Plug and Play (UPnP) Internet Gateway Device (IGD) Protocol. Allows NATed host to:
  - ❖ learn public IP address (138.76.29.7)
  - ❖ add/remove port mappings (with lease times)

  i.e., automate static NAT port map configuration

10.0.0.1

IGD

NAT
router
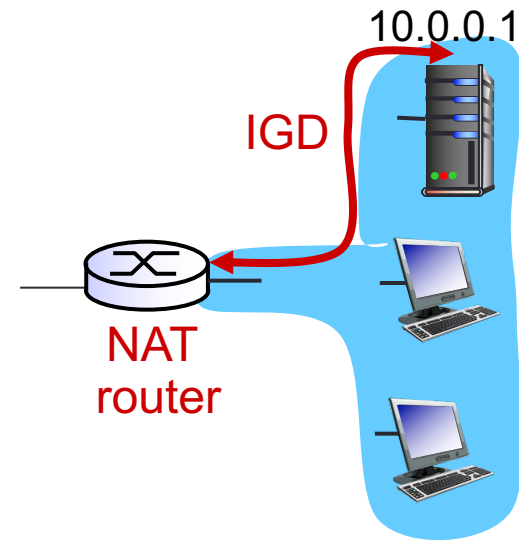
NAT traversal problem

- *solution 3:* relaying (used in Skype)
    - NATed client establishes connection to relay
    - external client connects to relay
    - relay bridges packets between to connections

2. connection to relay initiated by client

1. connection to relay initiated by NATed host

3. relaying established

client

10.0.0.1

138.76.29.7   NAT router

NAT traversal problem

- *solution 4:* NAT hole punching. Example: STUN protocol