RELATED CONCEPTS IN COMPUTER NETWORKS

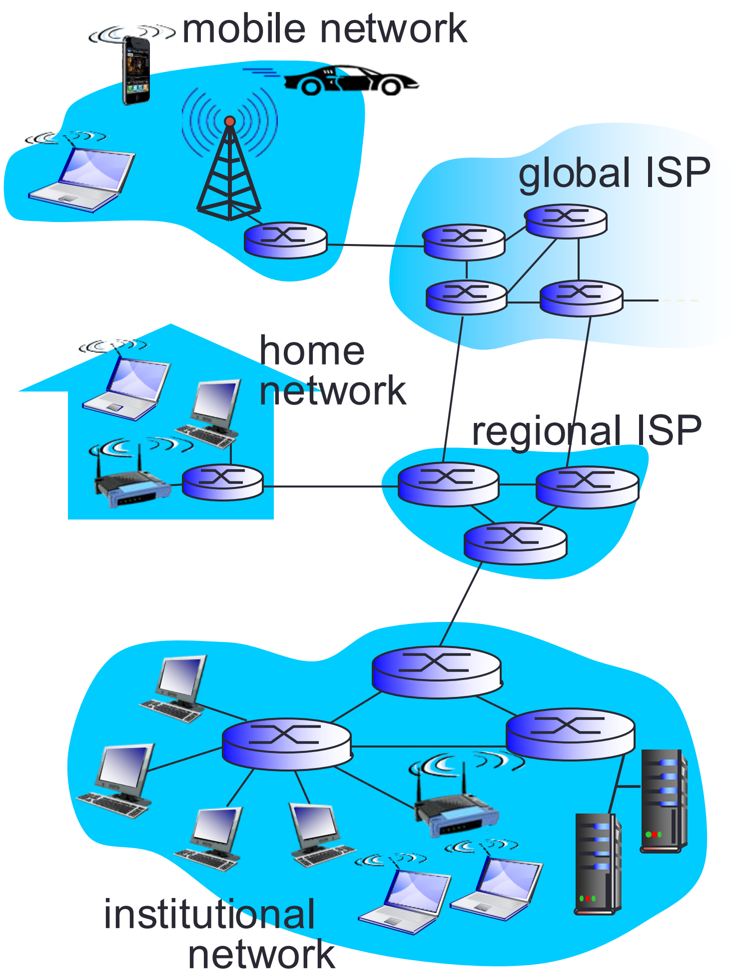
Some slides have been taken from: *Computer Networking: A Top Down Approach Featuring the Internet*, 3rd edition. Jim Kurose, Keith Ross. Addison-Wesley, July 2004.

Contents

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

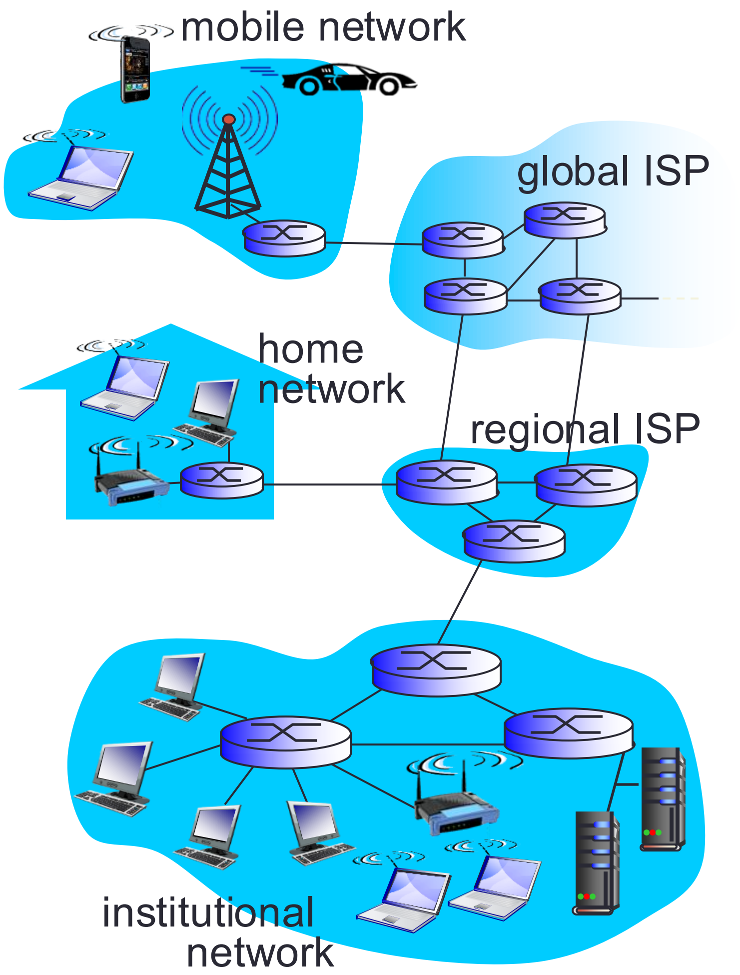
What's the Internet?

- *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



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



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:

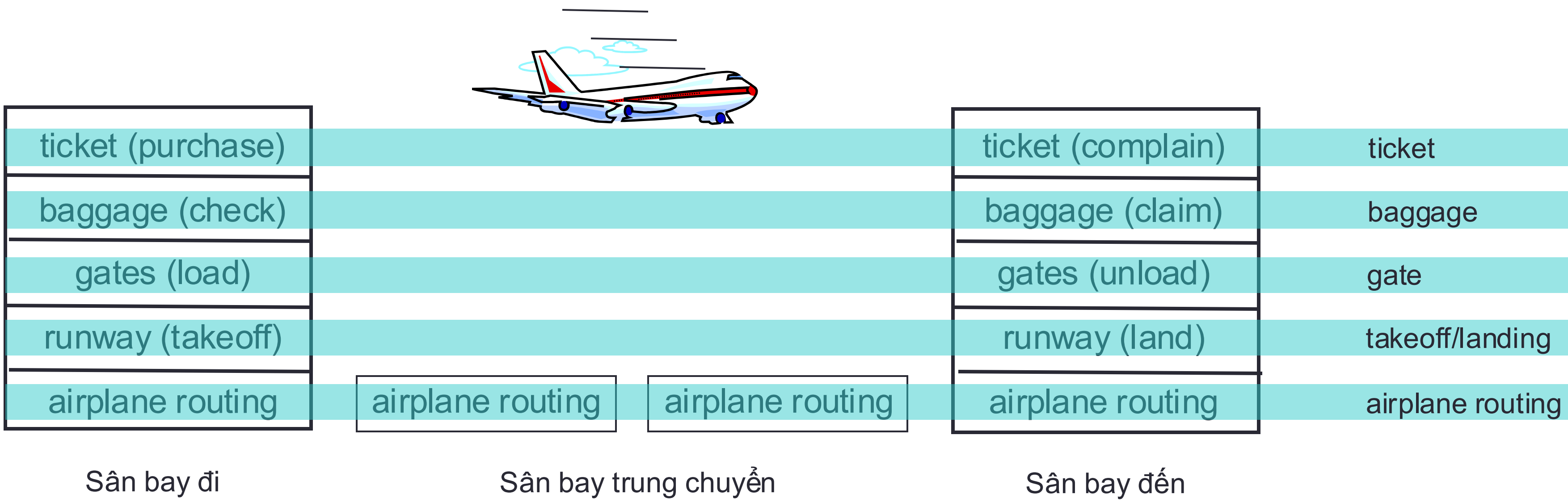
- machines rather than humans
- all communication activity in Internet governed by protocols

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

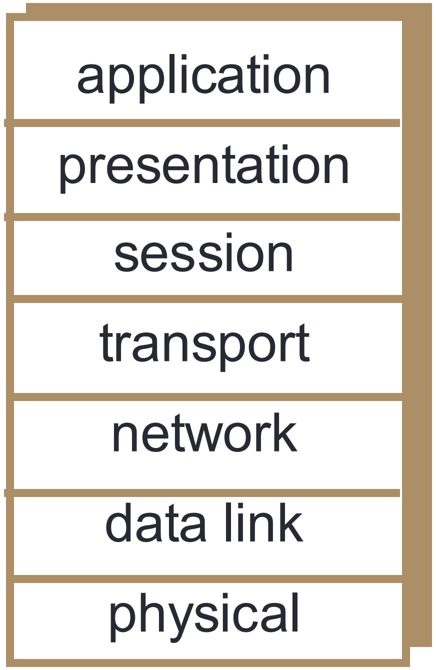


Tầng: Mỗi tầng có nhiệm vụ cung cấp 1 dịch vụ

- Dựa trên các chức năng của chính tầng đó
- Dựa trên các dịch vụ cung cấp bởi tầng dưới

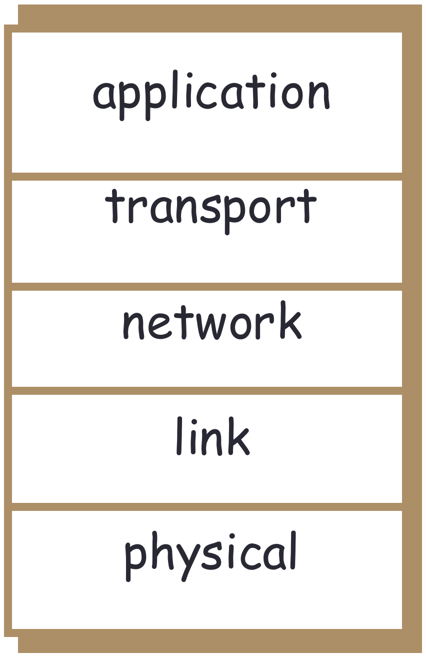
OSI protocol stack

- 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.

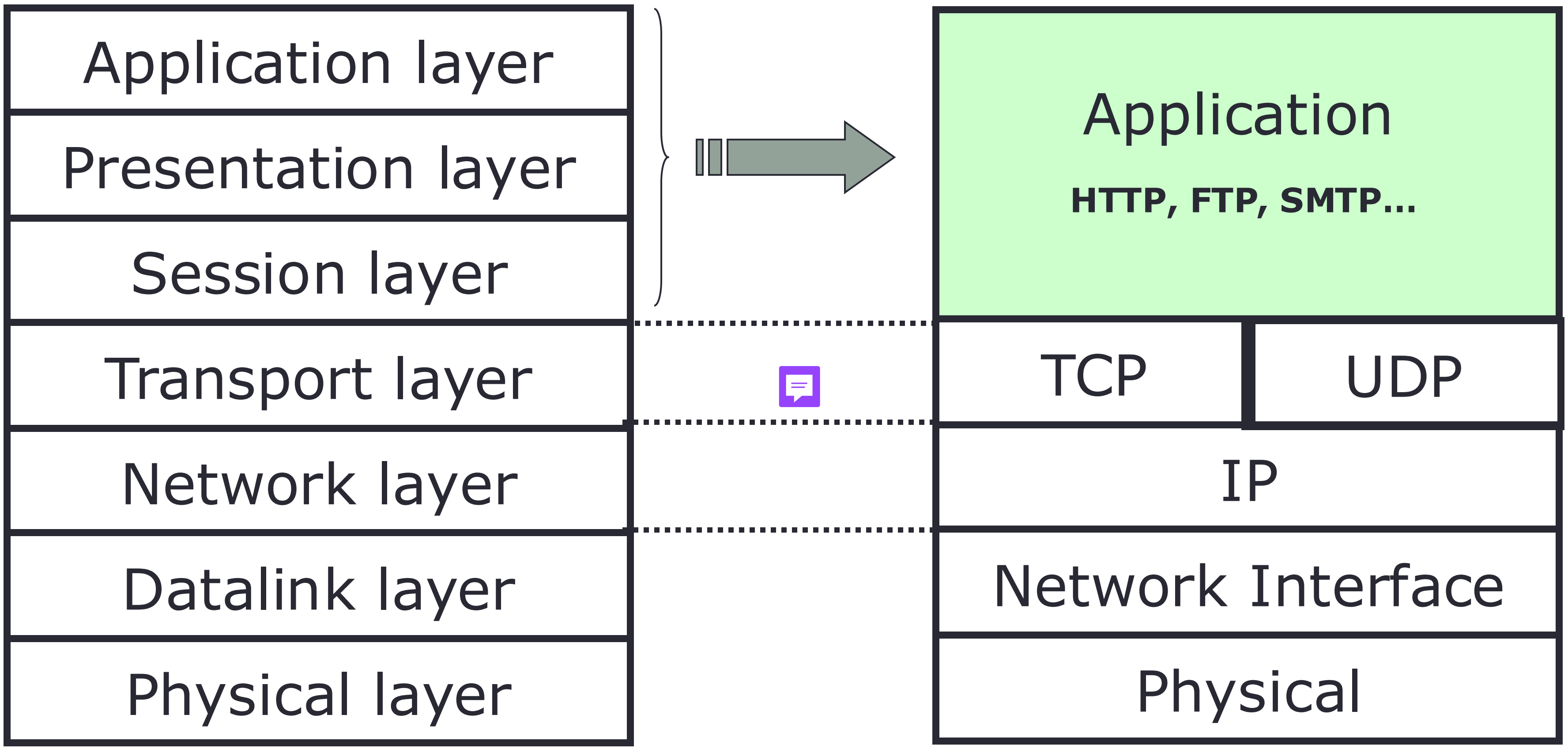


TCP/IP protocol stack

- 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”



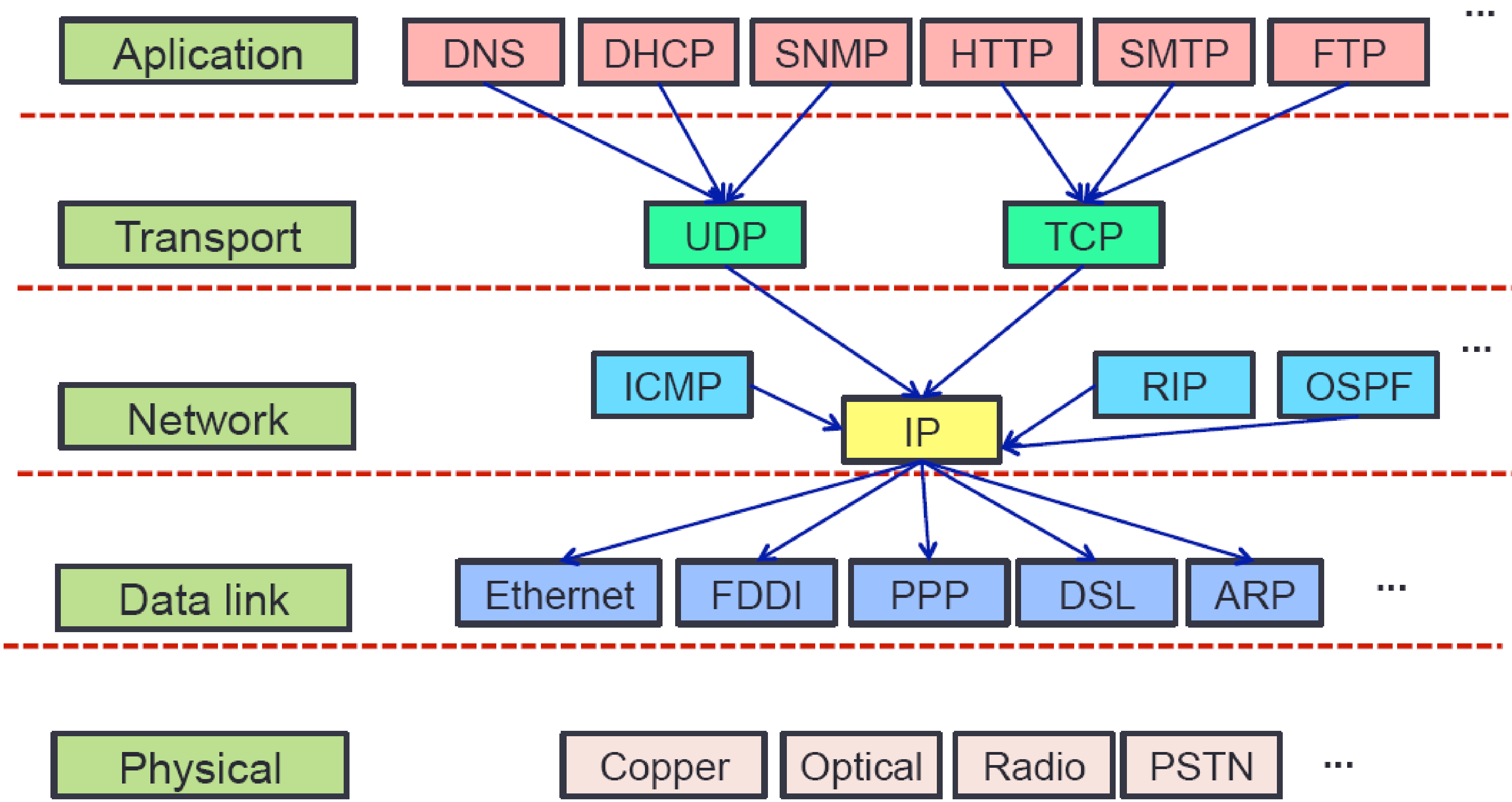
OSI and TCP/IP models



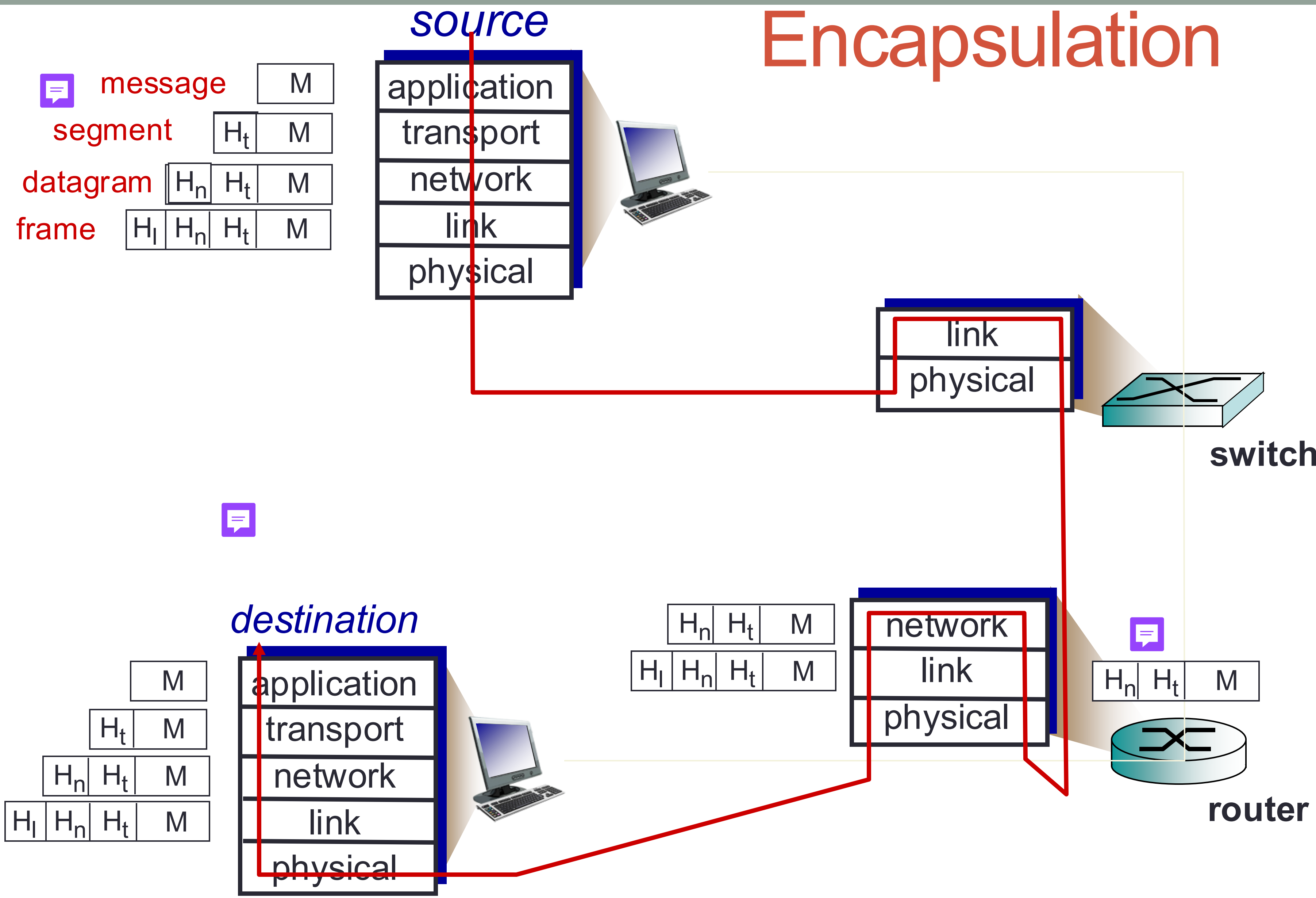
OSI and TCP/IP models

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

Internet protocols mapping on TCP/IP



Encapsulation



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

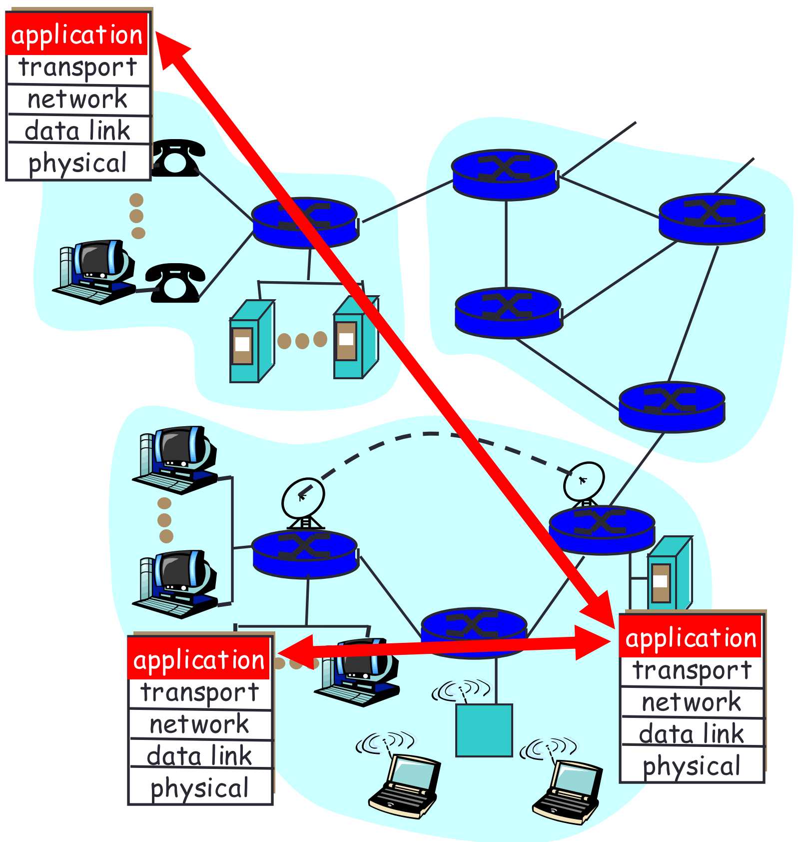
Creating a network app

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

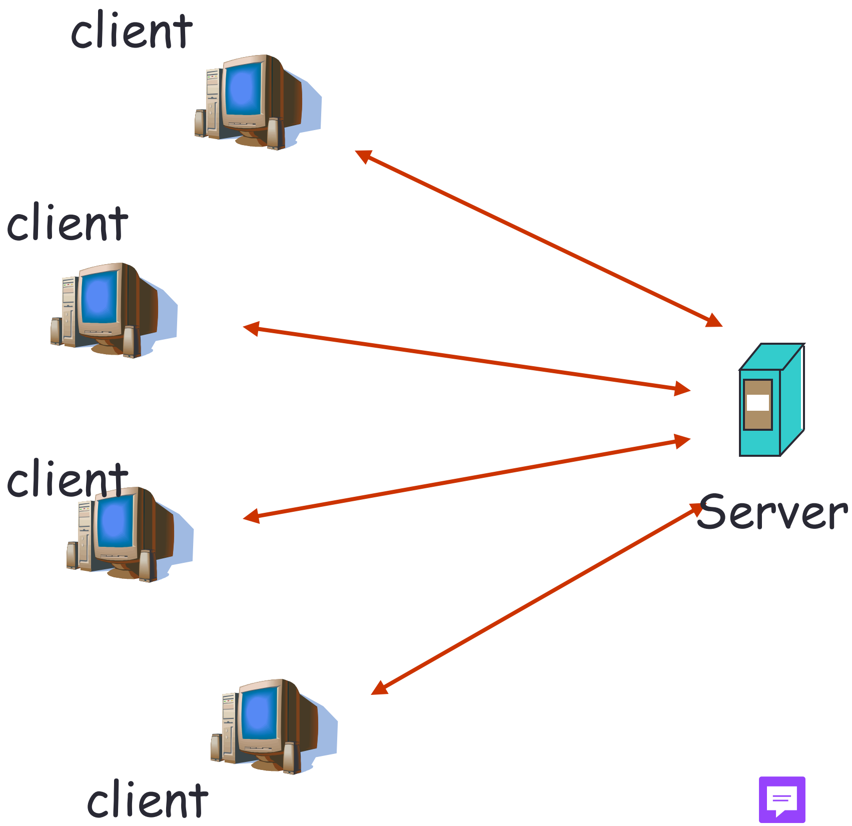
- Network core devices do not function at app layer
- This design allows for rapid app development



Application architectures

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

Client-server architecure



server:

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

clients:

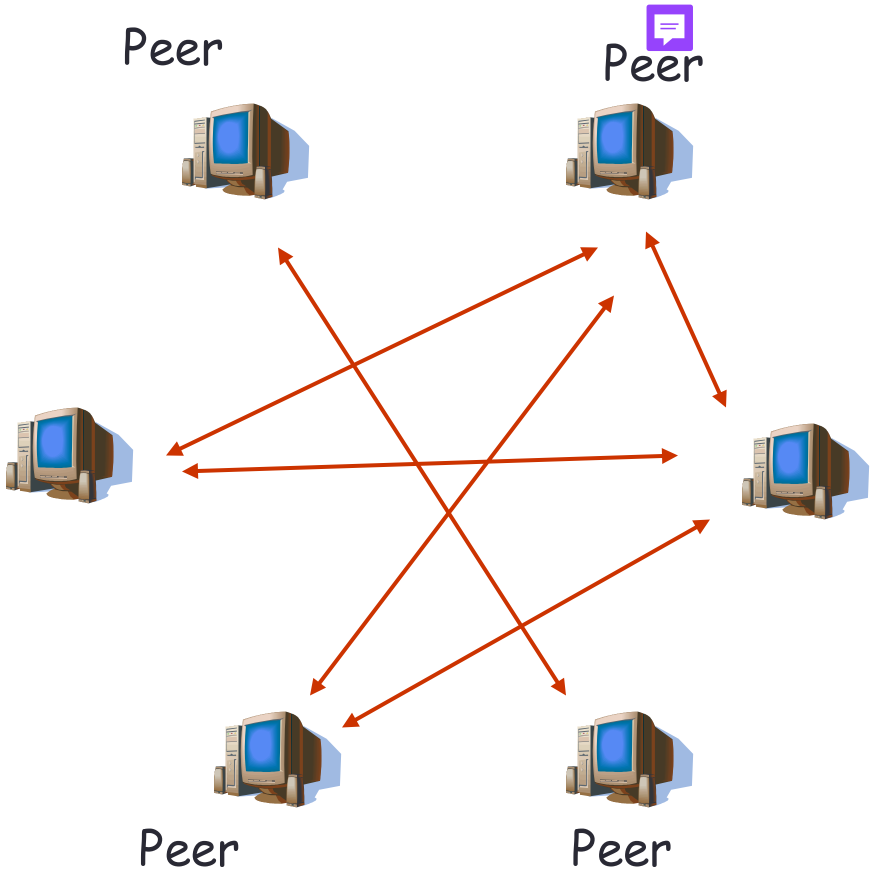
- Request services from server
- May have dynamic IP addresses
- Do not communicate directly with each other

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

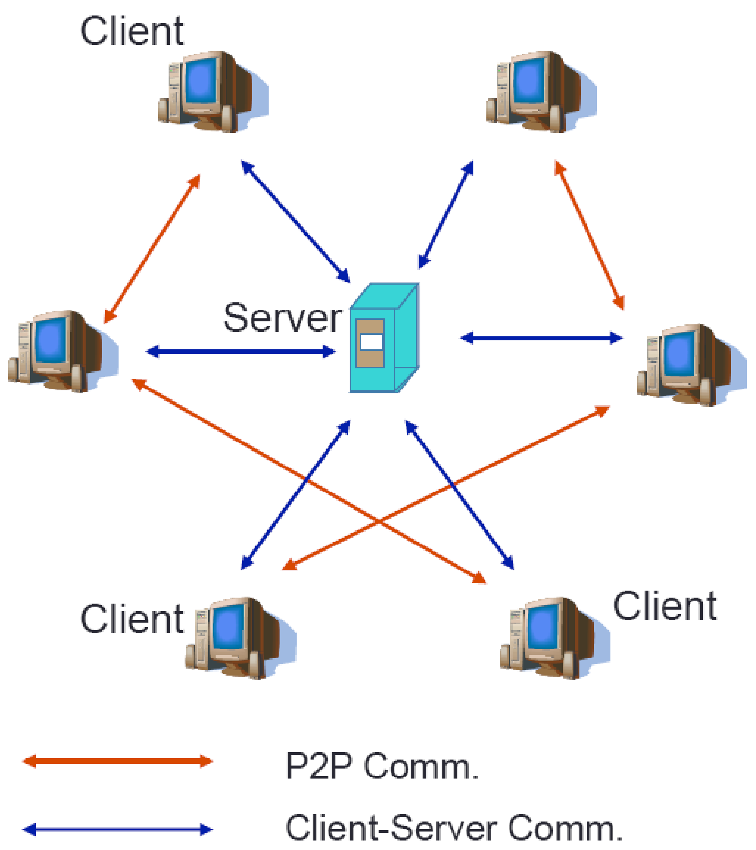
Example: Gnutella



Highly scalable

But difficult to manage

Hybrid of client-server and P2P



- 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

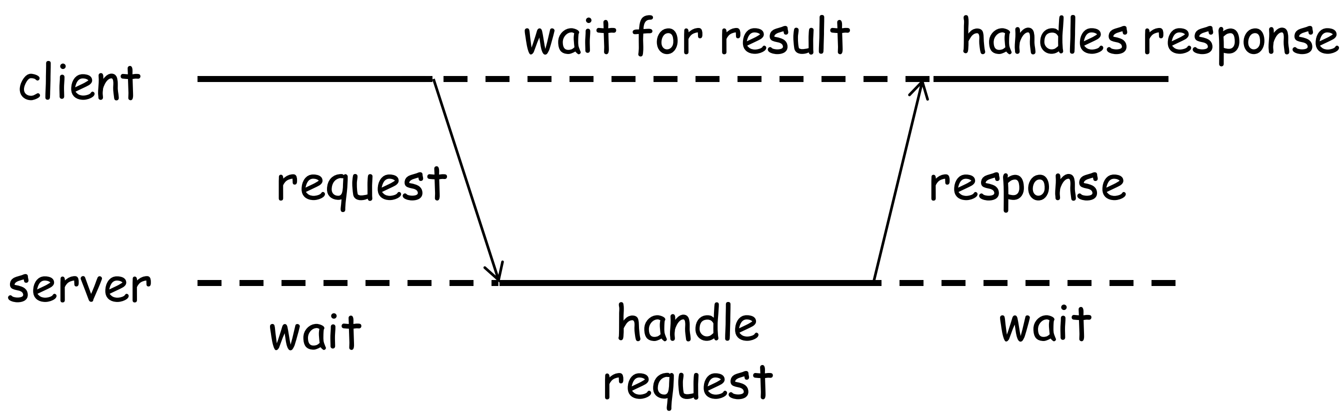
Client process: process that initiates communication

Server process: process that waits to be contacted

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

Processes communicating

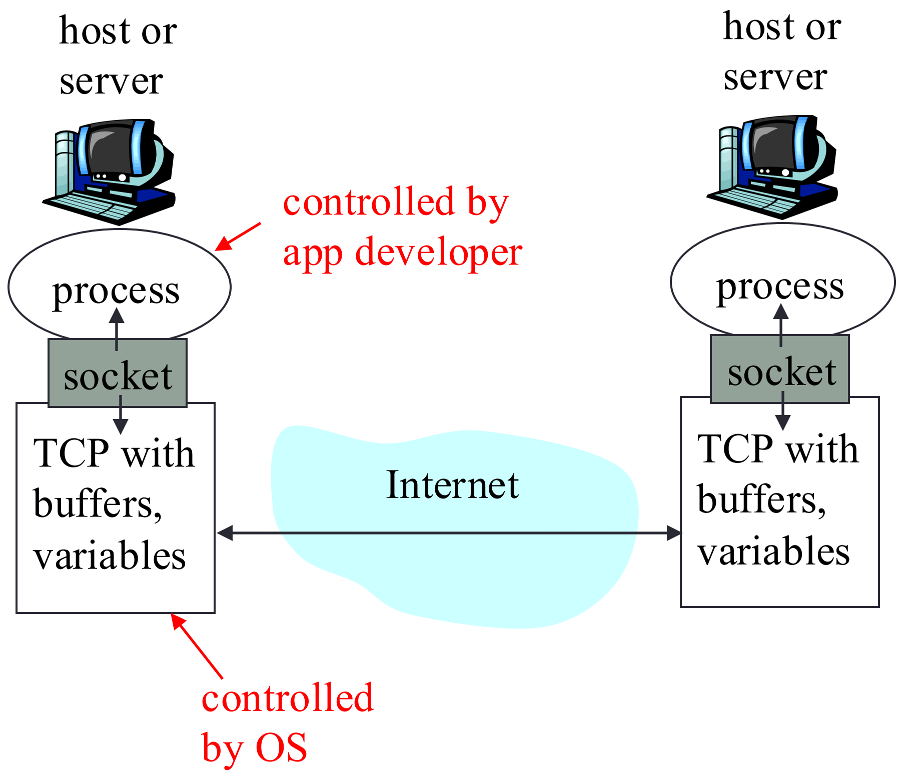
- Client process: sends request
- Server process: replies response
- Typically: single server - multiple clients
- The server does not need to know anything about the client
- The client should always know something about the server
  - at least the socket address of the server



Sockets

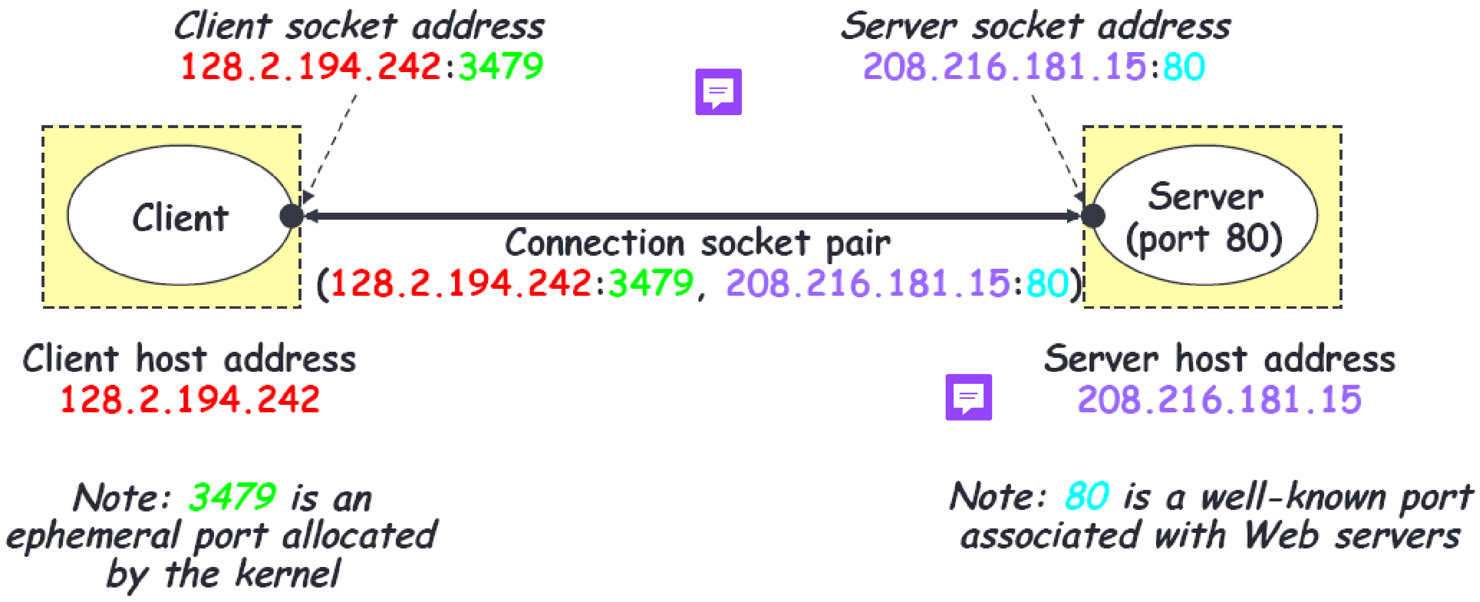
- process sends/receives messages to/from its socket
- Defined by
  - Port number
  - IP Address
  - TCP/UDP

Socket address



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*"



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

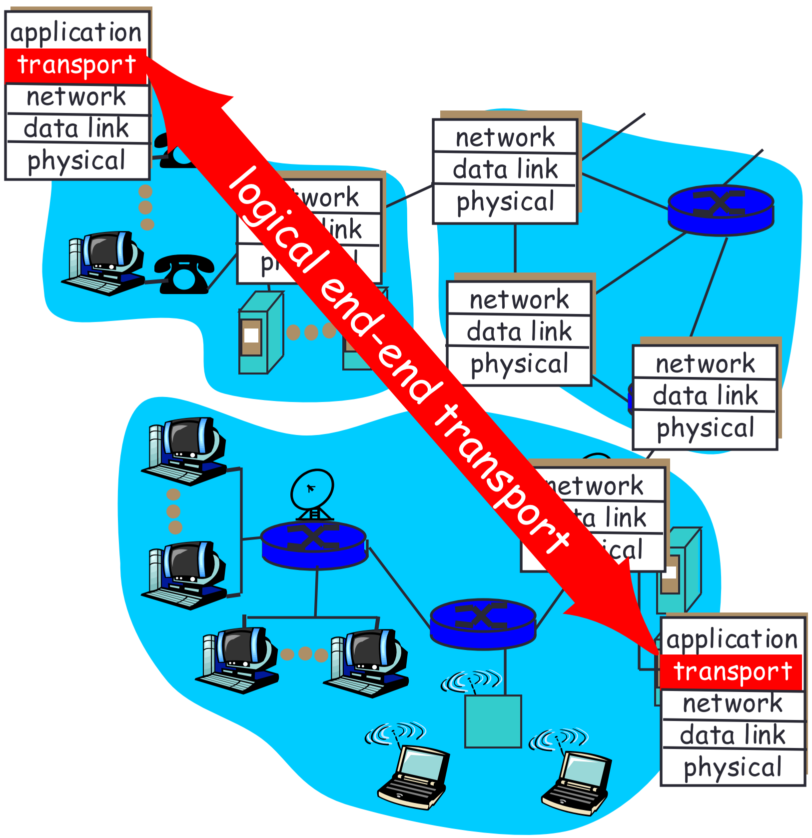
- some apps (e.g., Internet telephony, interactive games) require low delay to be "effective"

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

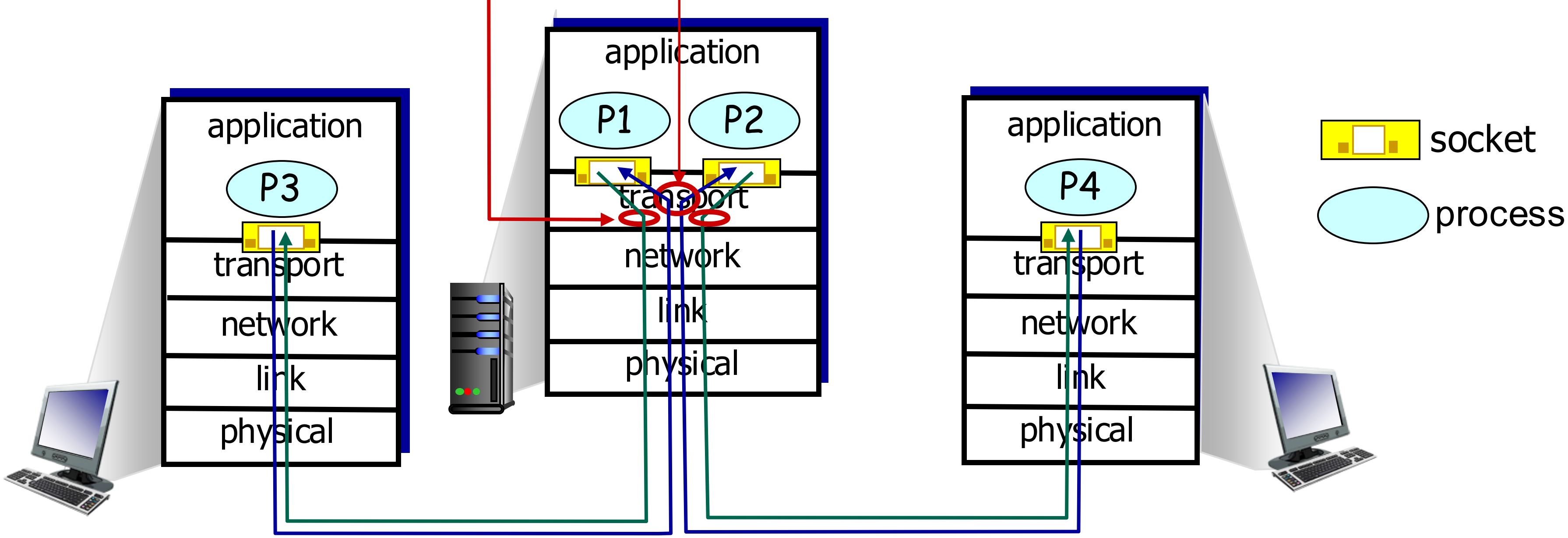
UDP service:

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

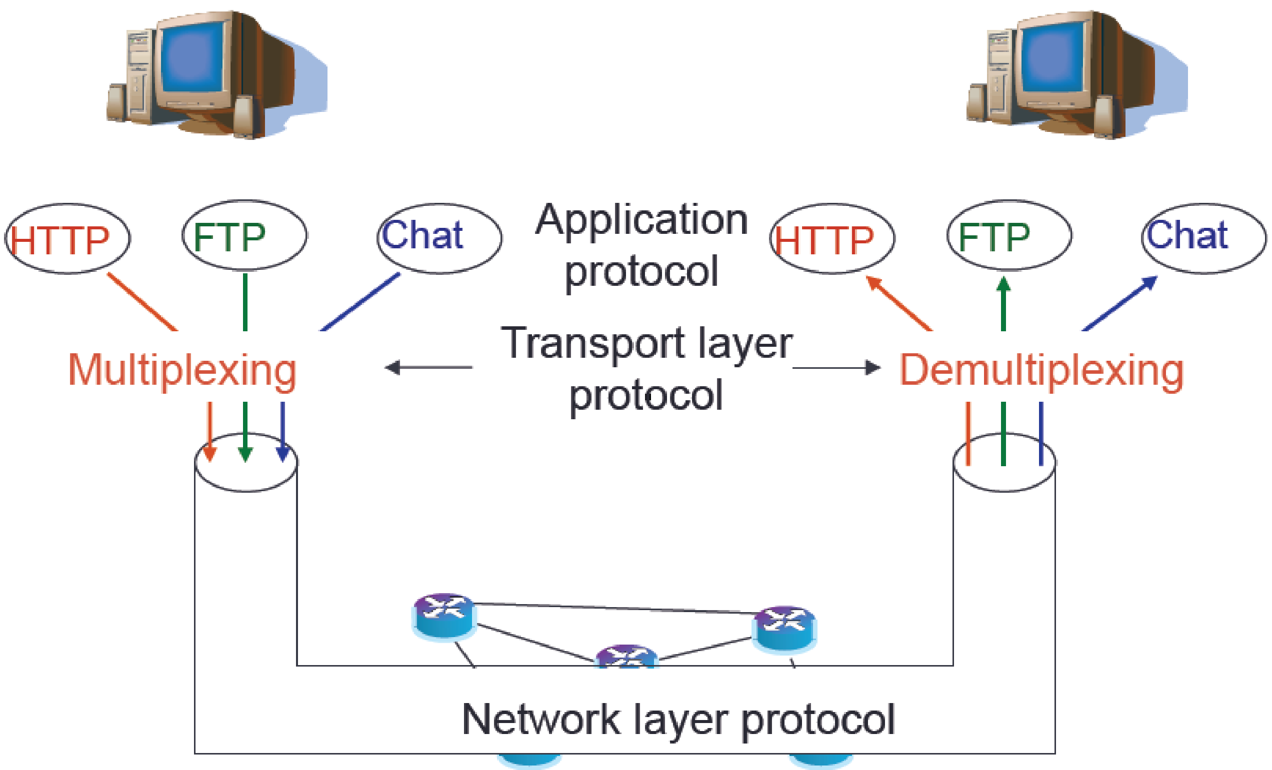
Multiplexing/demultiplexing

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

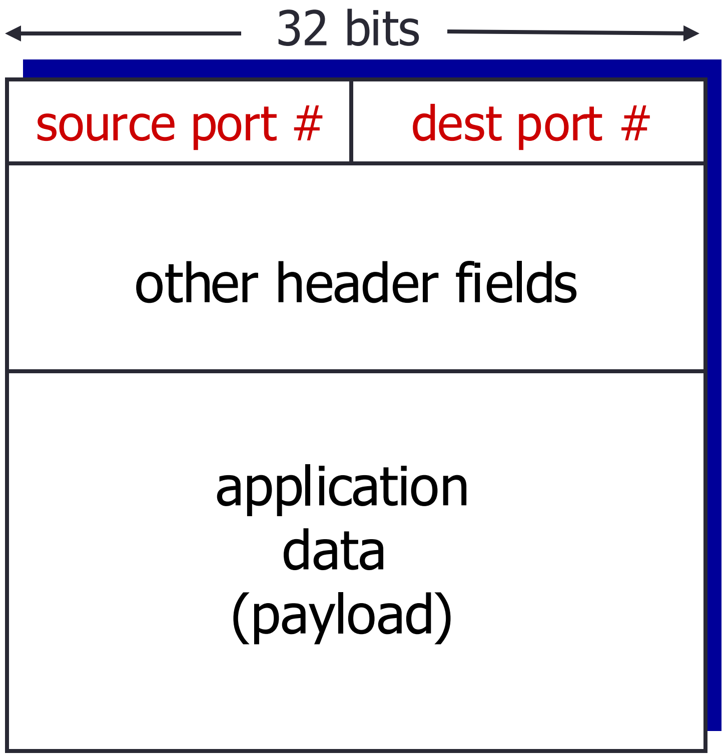


Transport layer Mux/Demux



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



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:

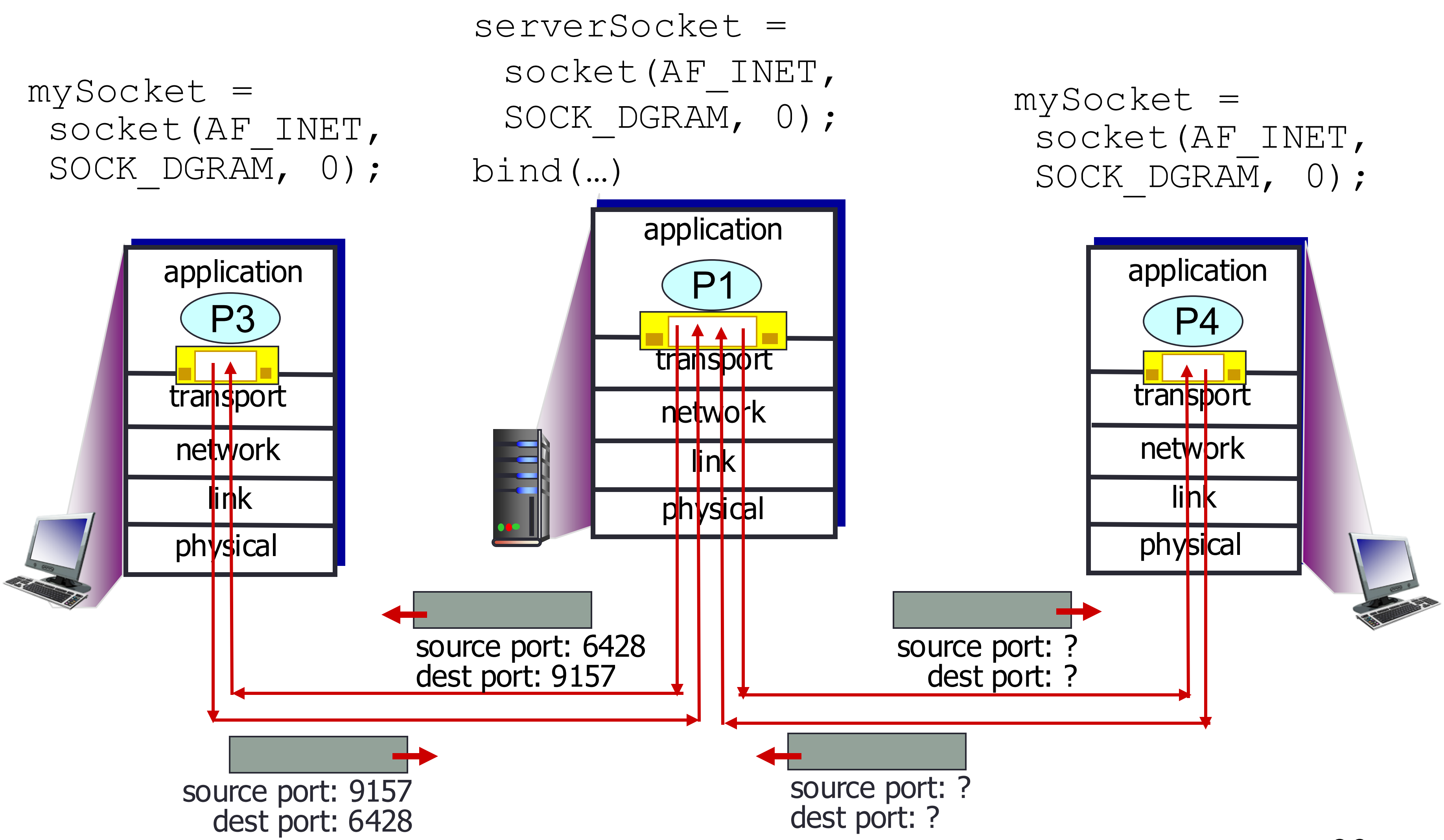
```
mySocket = socket(AF_INET,
  SOCK_DGRAM, 0)
```

- UDP socket identified by two-tuple:

(dest IP address, dest port number)

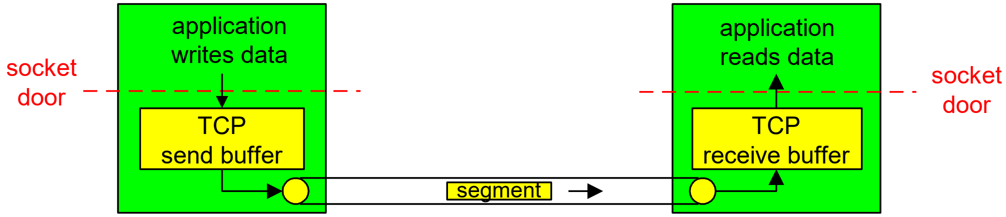
- When host receives UDP segment:
  - checks destination port number in segment
  - directs UDP segment to socket with that port number
- IP datagrams with different source IP addresses and/or source port numbers directed to same socket

UDP demux


mySocket =
socket(AF_INET,
SOCK_DGRAM, 0);
serverSocket =
socket(AF_INET,
SOCK_DGRAM, 0);
bind(...)
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: ?

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

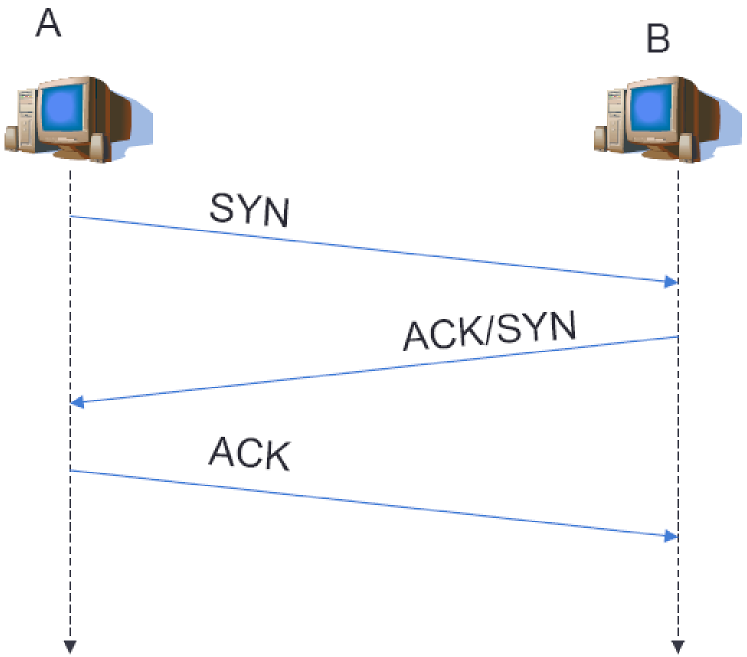
- point-to-point:
  - one sender, one receiver
- reliable, in-order *byte steam:*
  - no “message boundaries”
- pipelined:
  - TCP congestion and flow control set window size
- *send & receive buffers*



- 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

TCP Connection Management: Setup

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

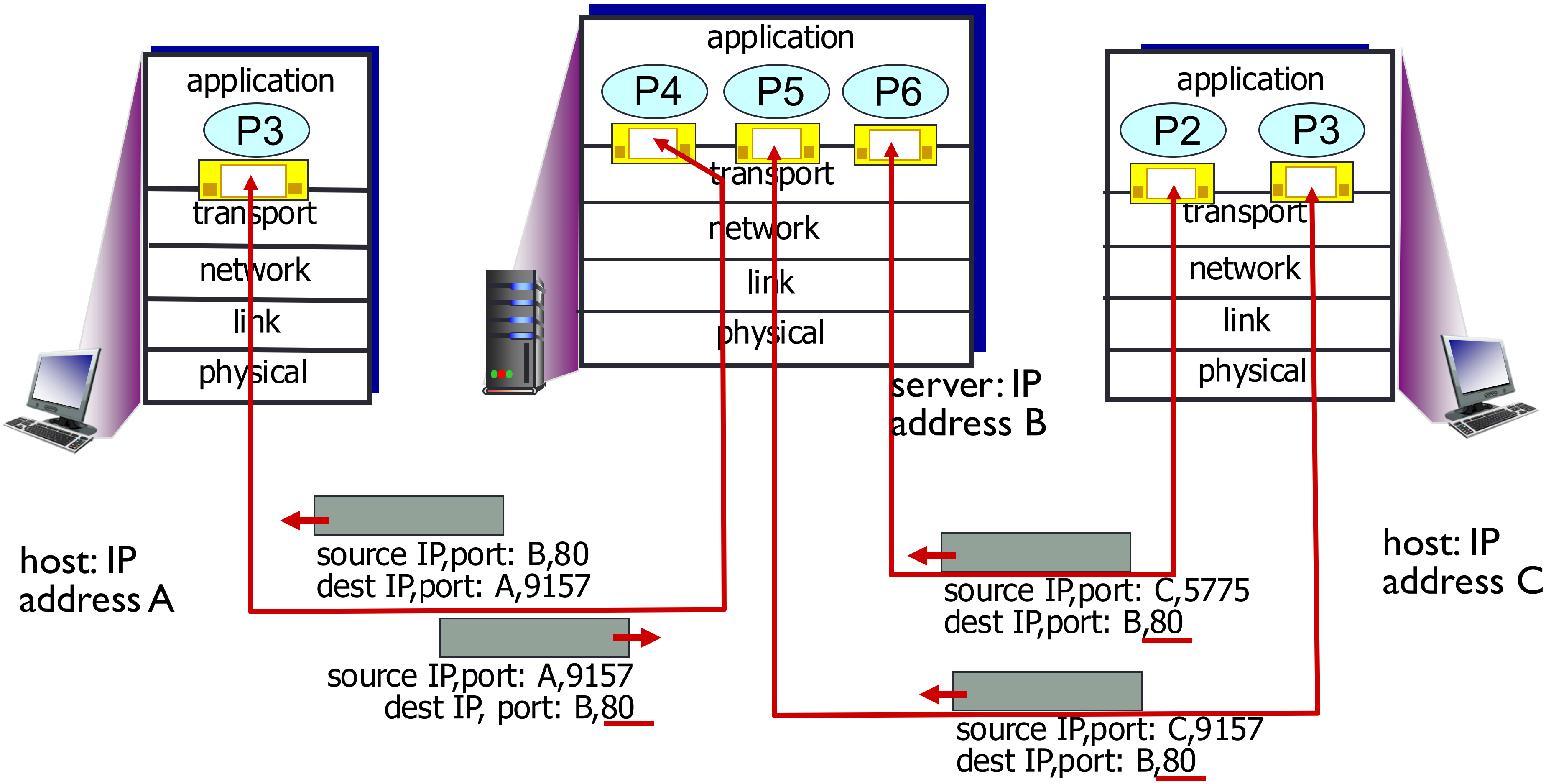


Connection-oriented demux

- 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

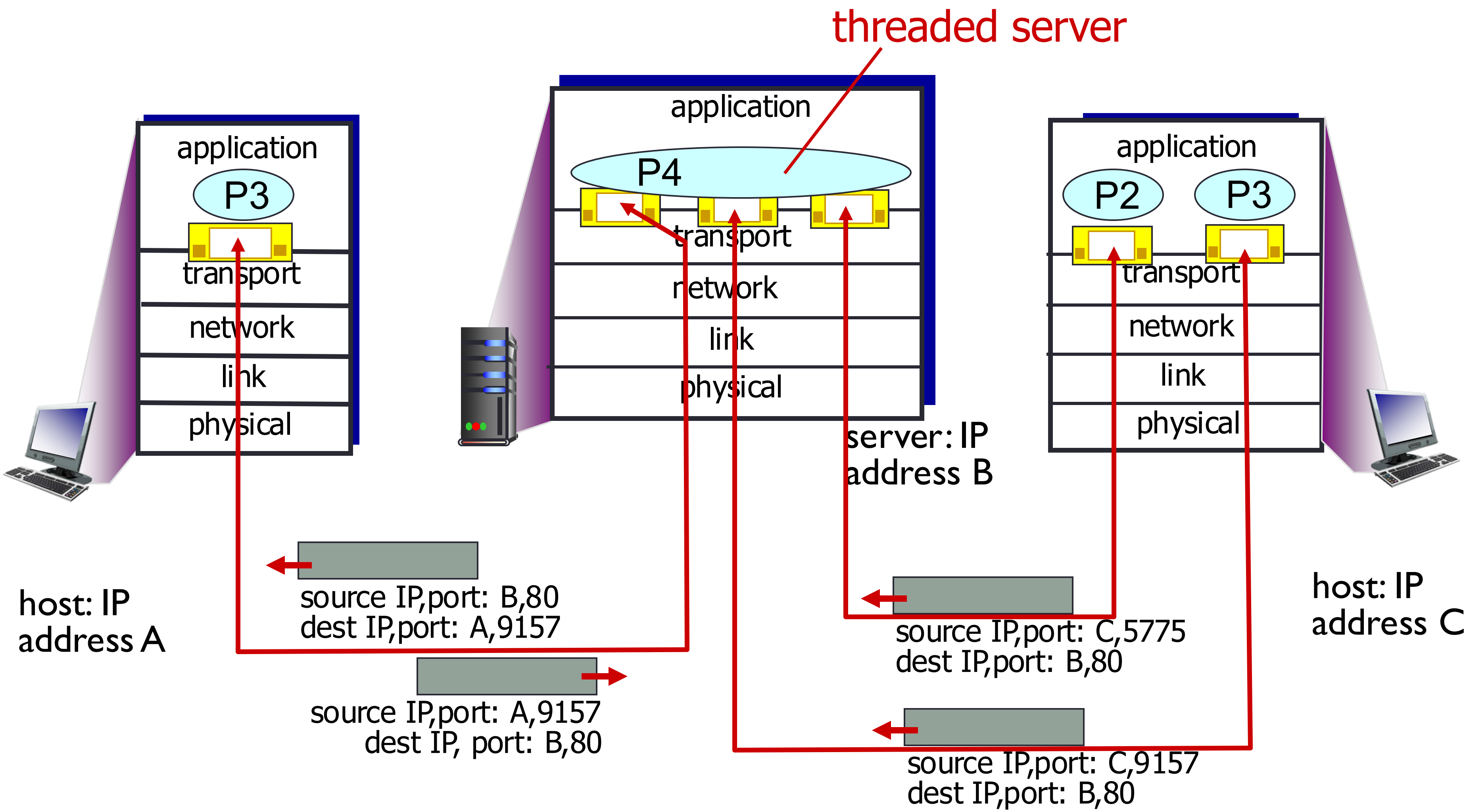
- Server host may support many simultaneous TCP sockets:
  - each socket identified by its own 4-tuple
- Web servers have different sockets for each connecting client

Connection-oriented demux: example



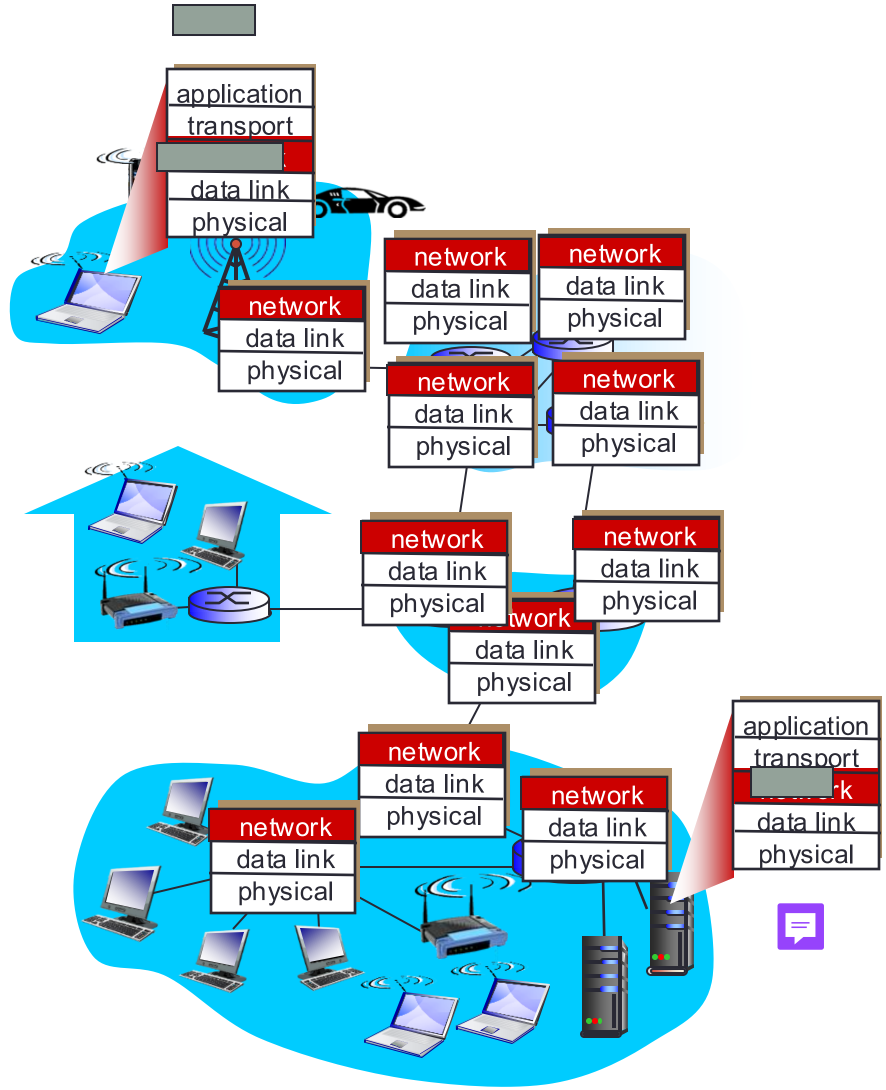
three segments, all destined to IP address: B,
dest port: 80 are demultiplexed to *different* sockets

Connection-oriented demux: example



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



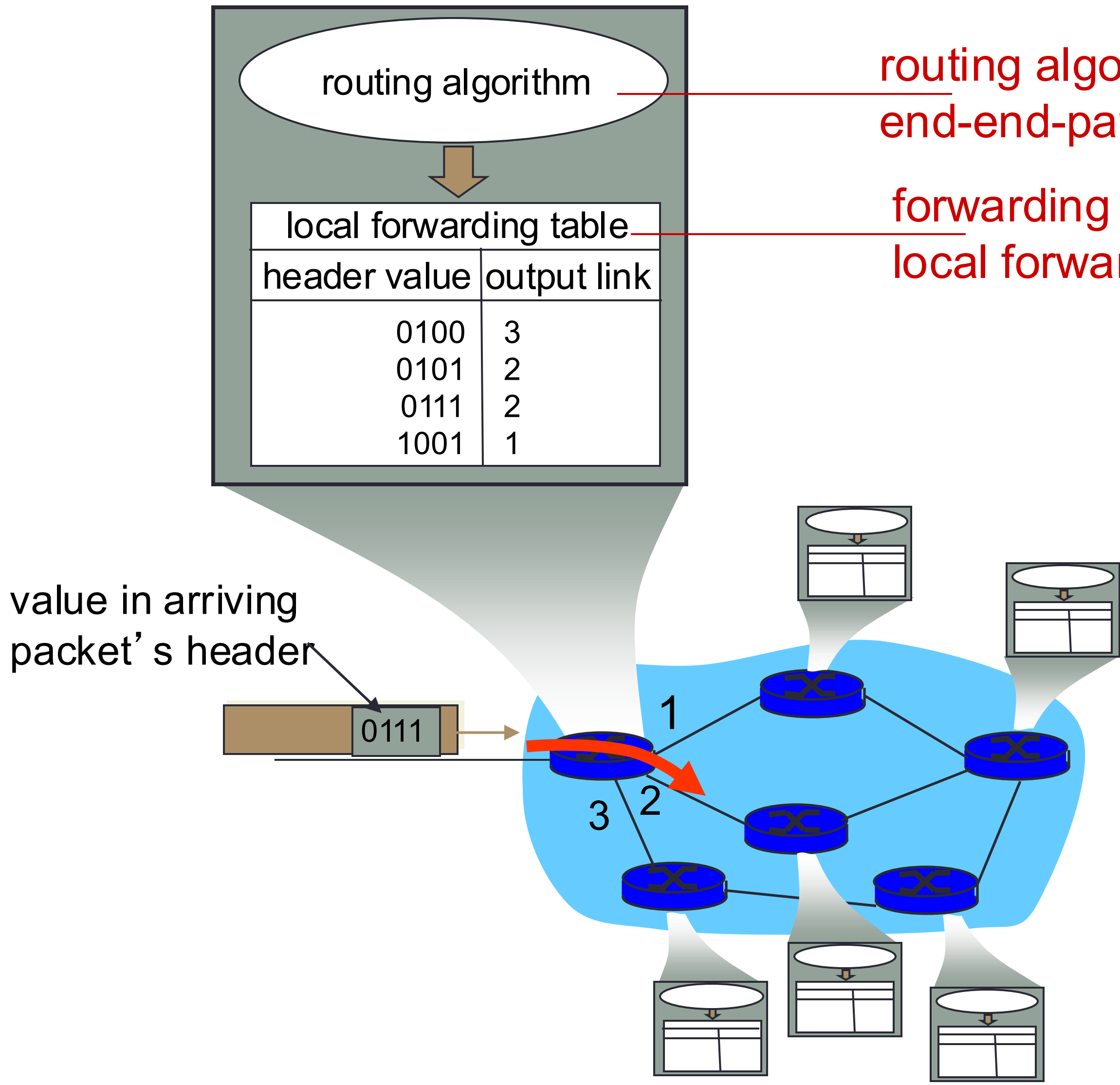
Two key network-layer functions

- *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

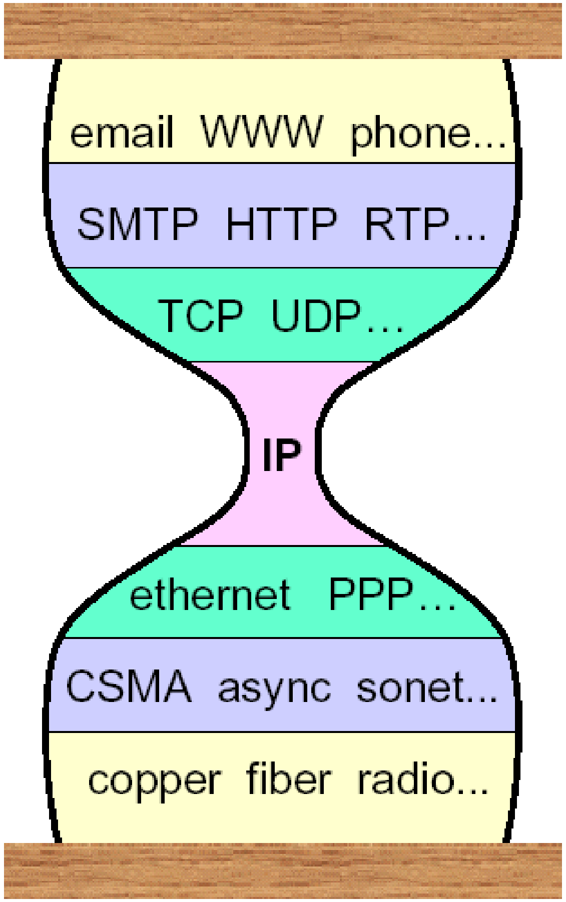


routing algorithm determines end-end-path through network

forwarding table determines local forwarding at this router

Why an internet layer?

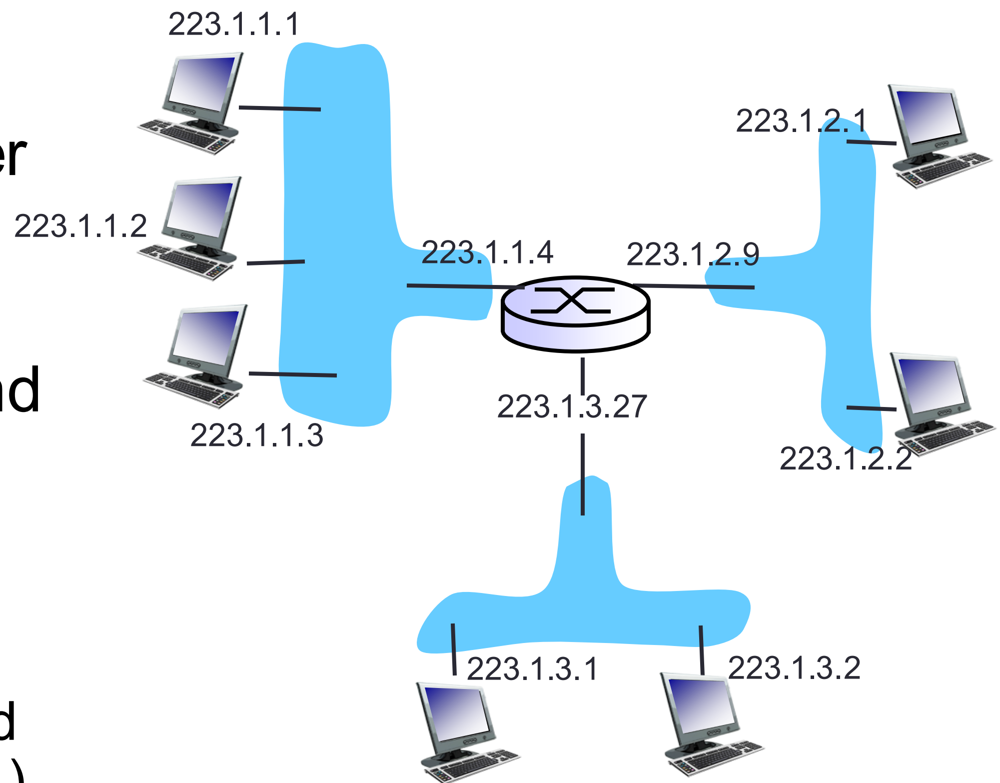
- Why not one big flat LAN?
  - Different LAN protocols
  - Flat address space not scalable
- IP provides:
  - Global addressing
  - Scaling to WANs
  - Virtualization of network isolates end-to-end protocols from network details/changes



"hourglass model"
(Steve Deering)

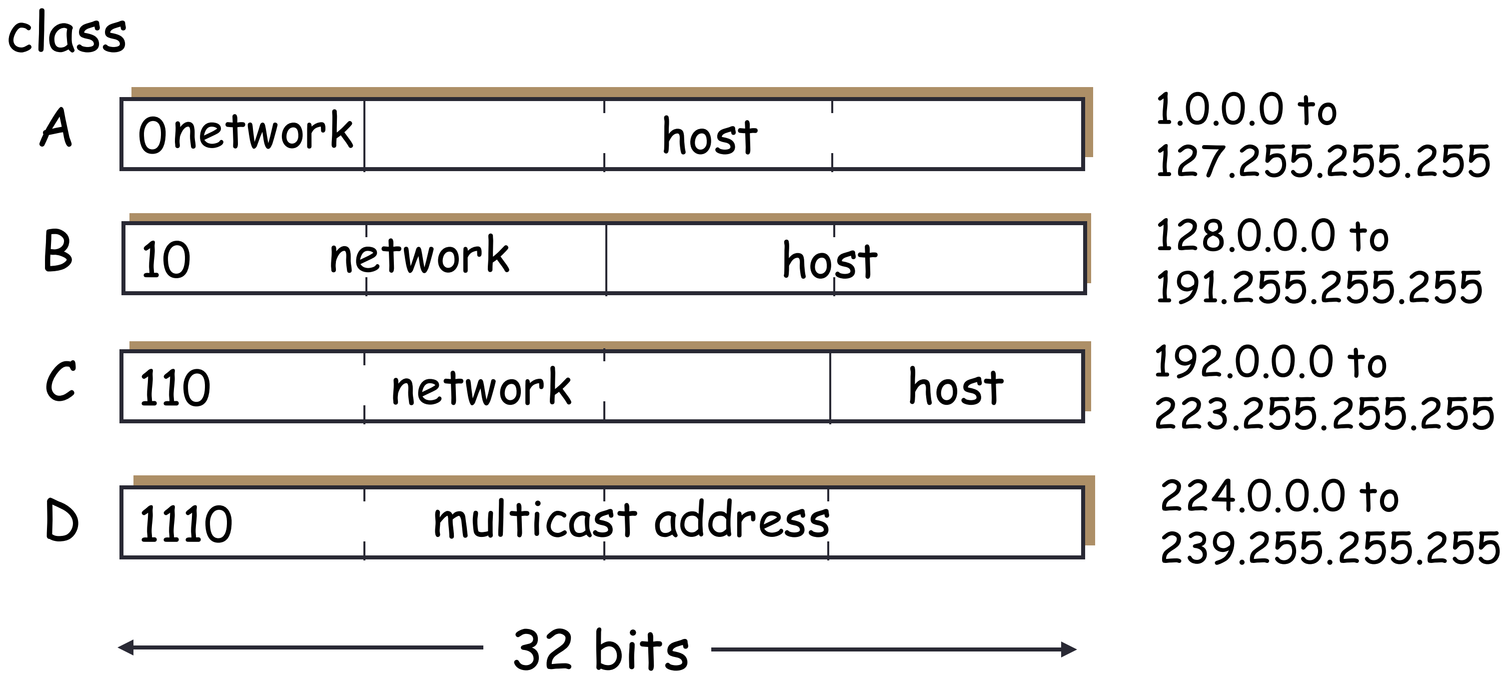
IP addressing: introduction

- *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*



$$223.1.1.1 = \underbrace{11011111}_{223}\ \underbrace{00000001}_{1}\ \underbrace{00000001}_{1}\ \underbrace{00000001}_{1}$$

IP addressing: “class-full”



- 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 InterDomain Routing

- 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 00010000 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

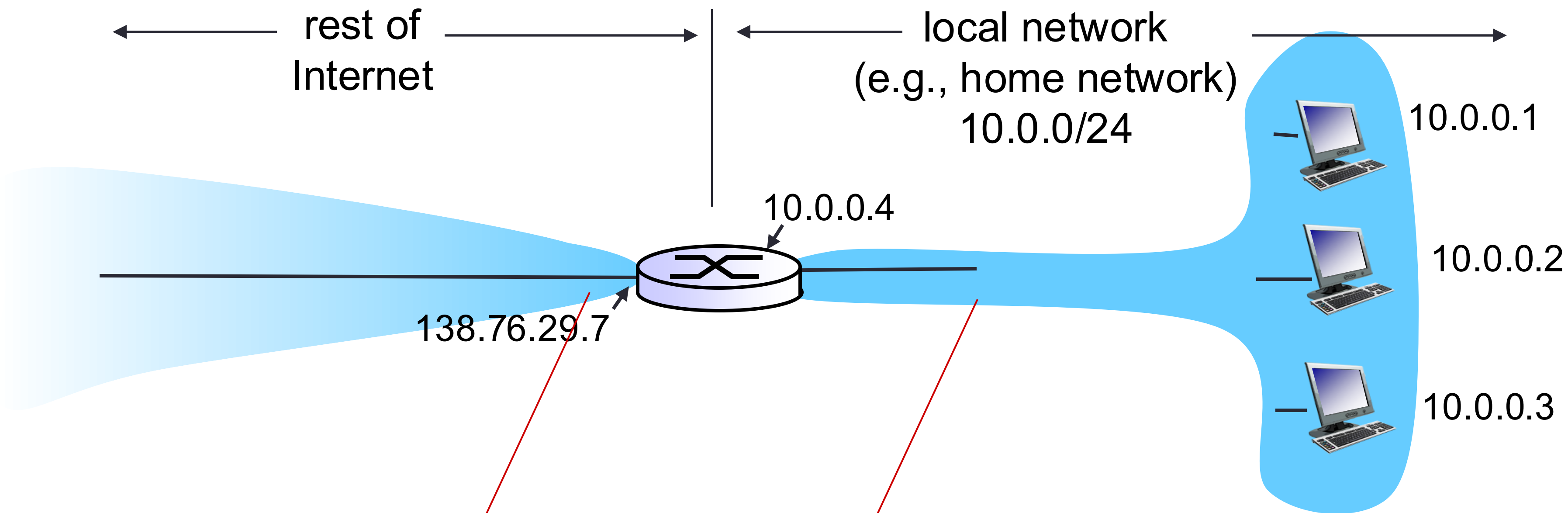
IP

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 <--> 202.191.56.68

NAT: network address translation



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:

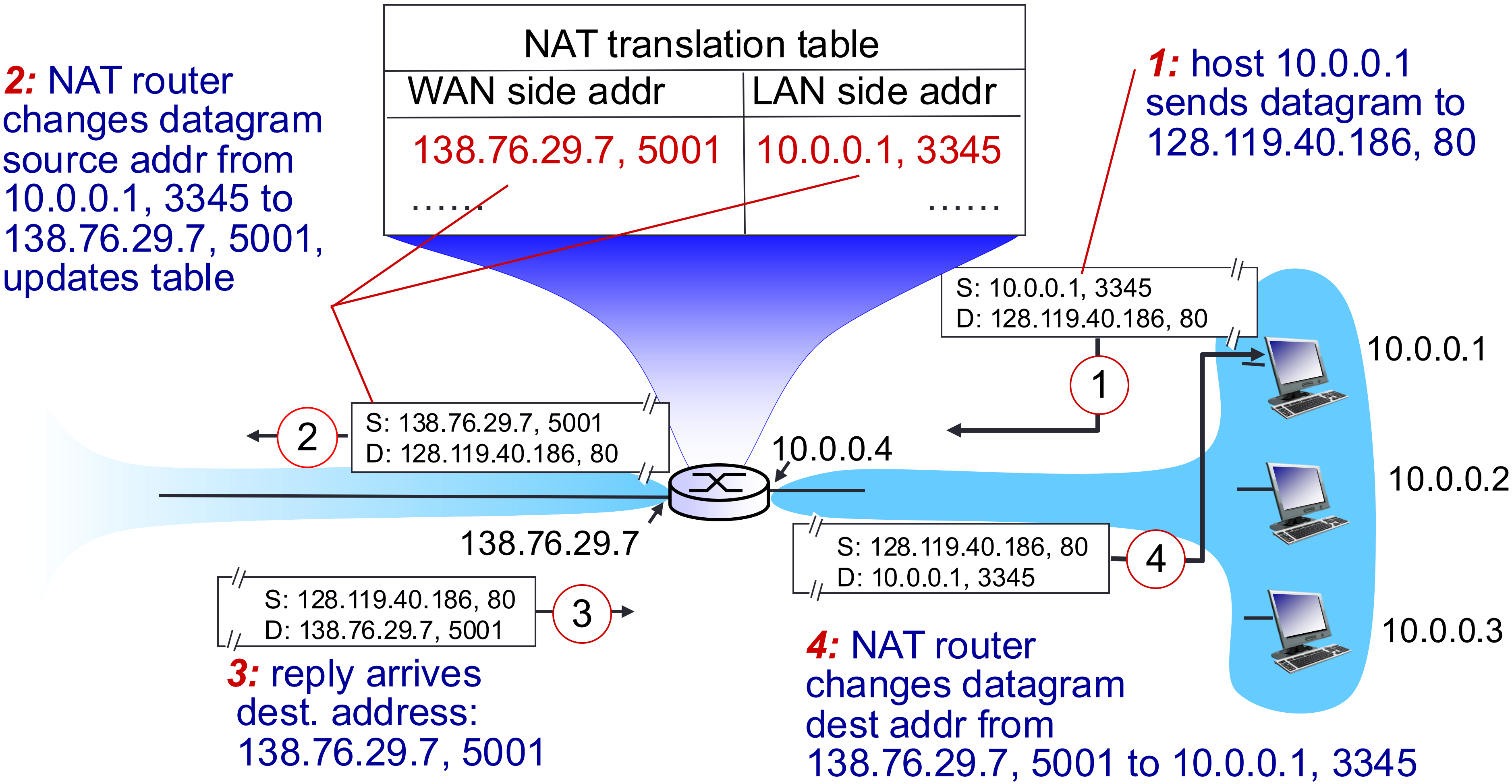
- *outgoing datagrams: replace* (source IP address, port #) of every outgoing datagram to (NAT IP address, new port #)

  . . . 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

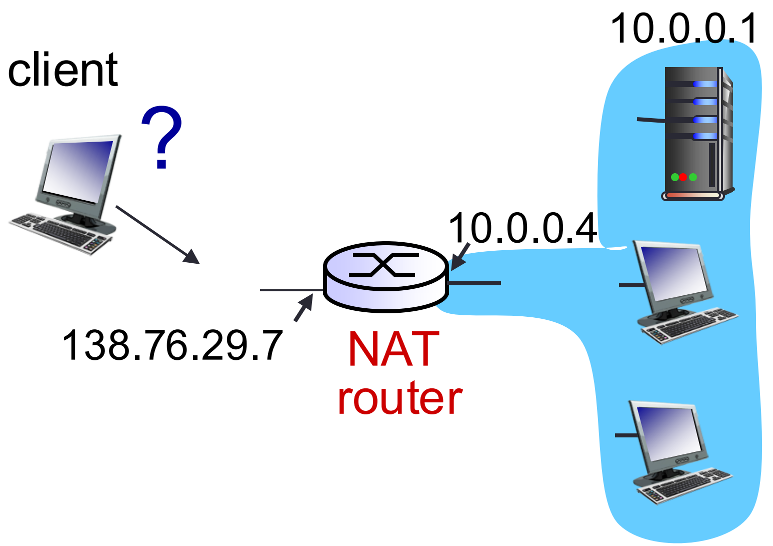


NAT: network address translation

- 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

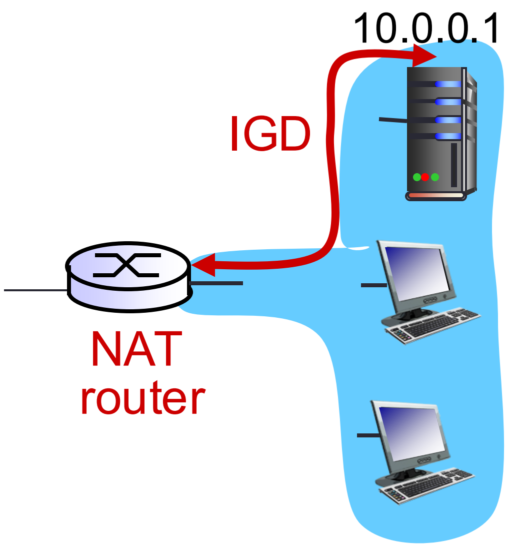
- client wants to connect to server with address 10.0.0.1
  - server address 10.0.0.1 local to LAN (client can’t use it as destination addr)
  - only one externally visible NATed address: 138.76.29.7
- *solution1:* statically configure NAT to forward incoming connection requests at given port to server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



NAT traversal problem

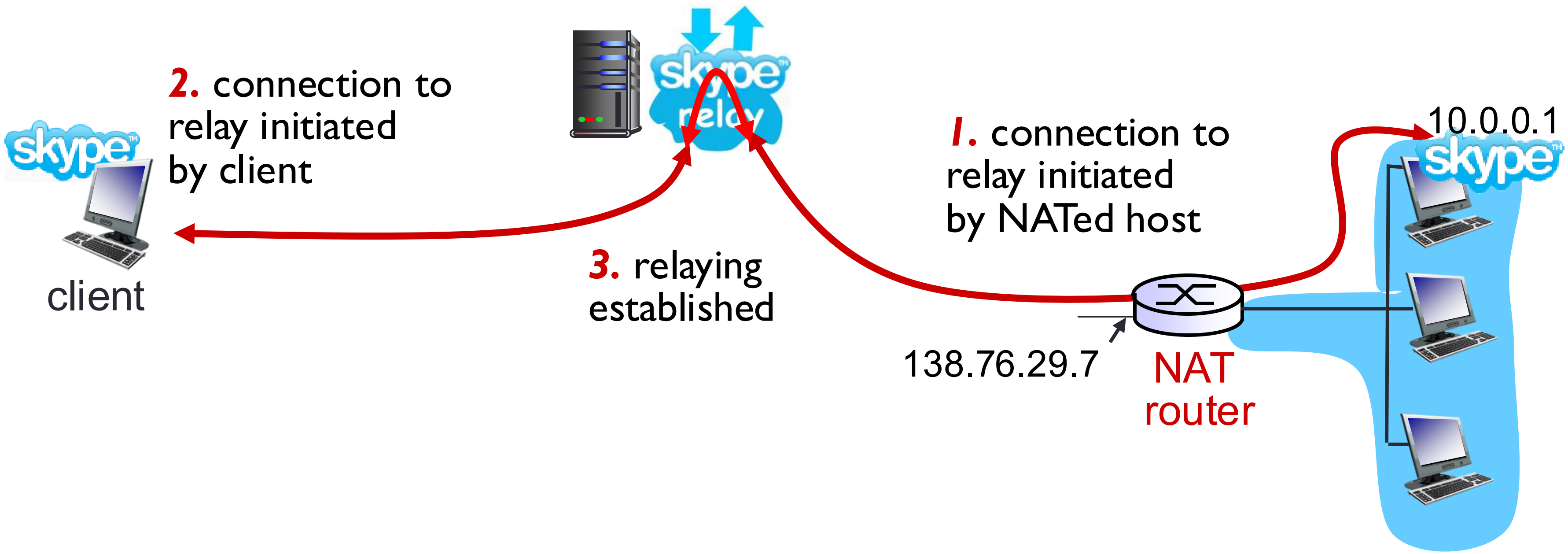
- *solution 2:* Universal Plug and Play (UPnP) Internet Gateway Device (IGD) Protocol. Allows NATed host to:
  - learn public IP address (138.76.29.7)
  - add/remove port mappings (with lease times)

i.e., automate static NAT port map configuration



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



NAT traversal problem

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