

# Chapter 1 Introduction

- What is Internet:
  - A nuts-and-bolts view
  - A service view
- Network edge:
  - Devices
  - Access networks
  - Physical media: Guided vs Unguided
- Network core:
  - Circuit switching, TDM, FDM
  - Packet switching, Store and forward
- (message switching)
- Internet structure
- **Delay**, loss, throughput
- **Protocol stacks:**
  - Protocol model of Internet
  - OSI model
  - Protocol data units
  - Encapsulation & Decapsulation

# Chapter 2 Application Layer

- Principles of network applications
  - Architecture: Client server vs P2P
  - Process, Socket, IP, port number
  - TCP service vs UDP service
- Web & HTTP
  - HTTP protocol,
  - HTTP connection: Persistent HTTP & Non-persistent HTTP, pipelining
  - Cookies
- Web cache
- EMAIL
  - SMTP vs. HTTP
  - Mail access protocols: POP3, IMAP
- DNS
  - Service, structure
  - DNS records
  - Name resolution, iterated query vs. recursive query

# Chapter 3 Transport Layer

- Transport-layer services, TCP vs UDP
- Multiplexing & demultiplexing
- UDP
- Reliable data transfer:
  - handle errors: error detection and feedback, stop and wait
  - handle duplicates:, seq number
  - handle loss: timeout, pipelining, Go-back-N vs. Selected repeated
- TCP:
  - segment structure, seq#, timer, RTT, ACK
  - retransmission, fast retransmit
  - flow control
  - 3-way handshake
- congestion control principles
  - Cause & cost
  - Approaches: end-end vs network assisted congestion control
- TCP congestion control:
  - AIMD
  - slow start, congestion avoidance, fast recovery
  - TCP Tahoe, TCP Reno

# Chapter 4: network layer – data plane

- Network layer
  - Functions, data plane, control plane
  - Services, best-effort
- Router:
  - Architecture and functions, **longest prefix match**
  - buffer management
  - Pkt Scheduling: FCFS, Priority, Round-bin, WFQ
- Internet Protocol/IP
  - datagram format
  - Addressing, IP address, interface, subnet, CIDR
  - DHCP
- NAT: network address translation
- IPv6, tunneling and encapsulation

# Chapter 5: network layer – control plane

- routing protocols
  - link state
  - distance vector
- intra-ISP routing:
  - RIP, EIGRP
  - OSPF
- Inter-ISP routing: **BGP**
- Internet Control Message Protocol

# Chapter 6 data link layer

- Link layer services
- error detection & correction
  - Parity checking, checksum, CRC
- multiple access links & protocols
  - Channel partition: TDMA, FDMA
  - Random access:
    - Slotted ALOHA, pure ALOHA
    - CSMA, CSMA/CD
  - Taking turns
- LANs
- MAC address
- Address resolution protocols: ARP
- Ethernet:
  - topology
  - frame structure
  - Services
  - Standards
  - Hub, switch, router



西北工业大学  
NORTHWESTERN POLYTECHNICAL UNIVERSITY



# Computer Networks

Lecturer: Prof. ZHANG Ying

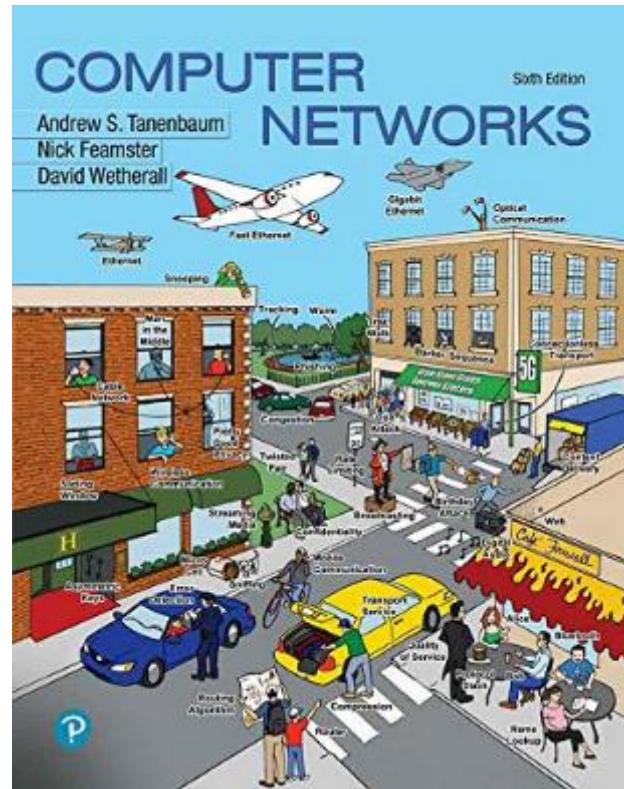
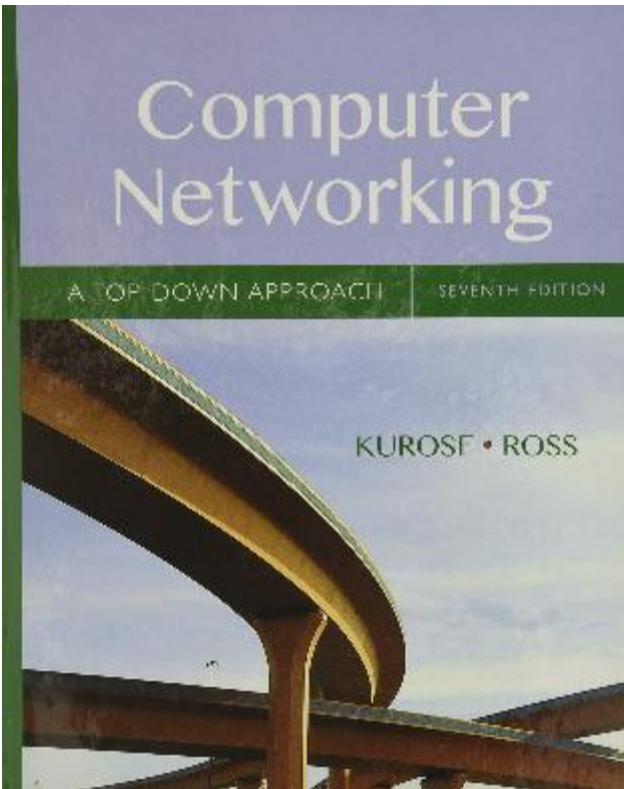
Fall semester 2022

[izhangying@nwpu.edu.cn](mailto:izhangying@nwpu.edu.cn)

# About

- Week 2 – week 11
  - 10 weeks
  - 20 classes, 2 sessions/class
  - 90 min/class, 45 min/session
- Tuesday & Thursday, Beijing Time
  - 10:30-12:10
- Enrollment: 17 students
- TA: Cheng Tiancong
- Stay tuned for WeChat group

# Literatures



- 《Computer Networking: A Top-Down Approach》 James F. Kurose, Keith W. Ross
  - 《Computer Networks》, Andrew S. Tanenbaum

# Policy

- Attendance
  - Your qualification of examination will be cancelled if you are absent from class without prior notice more than twice.
- Scoring (Tentative)
  - Assignment 25%
  - Midterm/Project 25%
  - Final 50%

# Chapter 1: Introduction

# Chapter Outline

1

what is the Internet

2

network edge

3

network core

4

delay, loss, throughput in networks

5

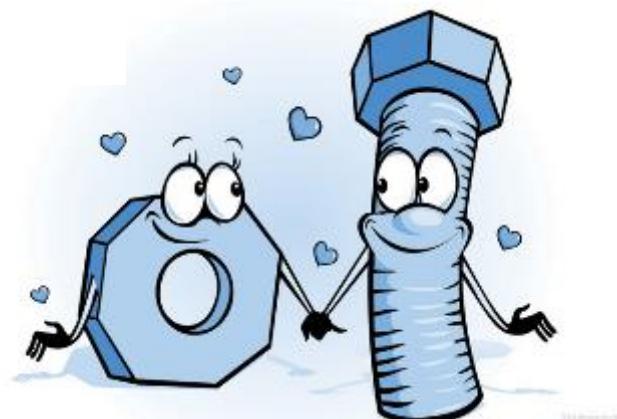
protocol layers, service models

6

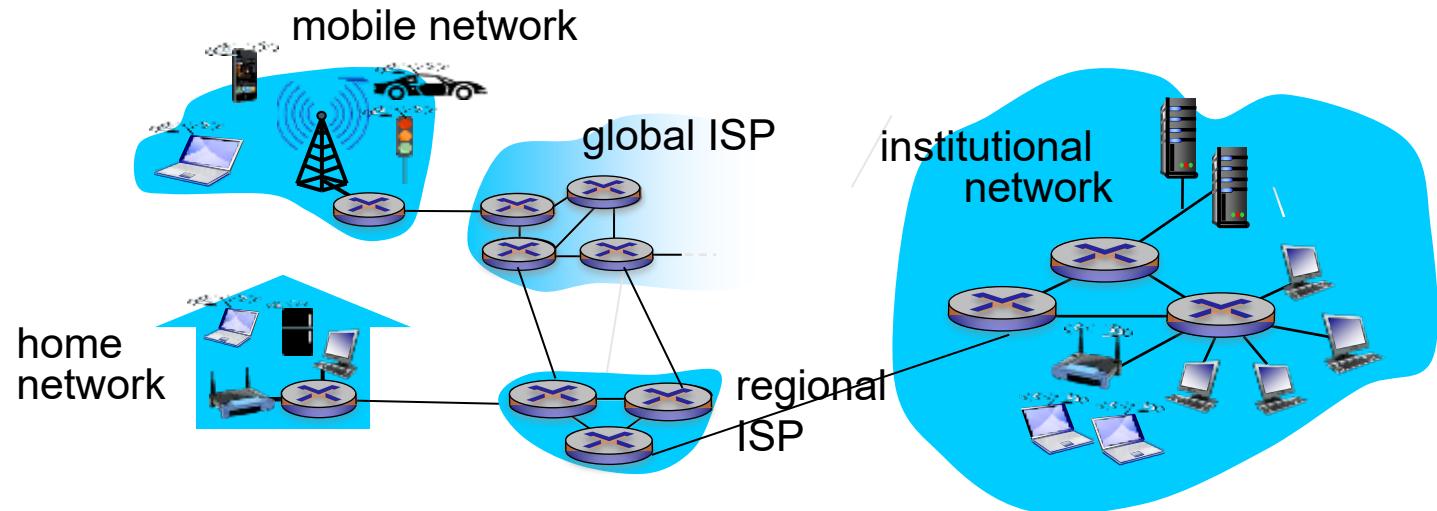
history

# What is Internet

- A Nuts-and-Bolts view
- A service view



# What's the Internet (I) : “nuts and bolts” view



Many connected computing devices

- *hosts = end systems*
- running *network apps*



communication links

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

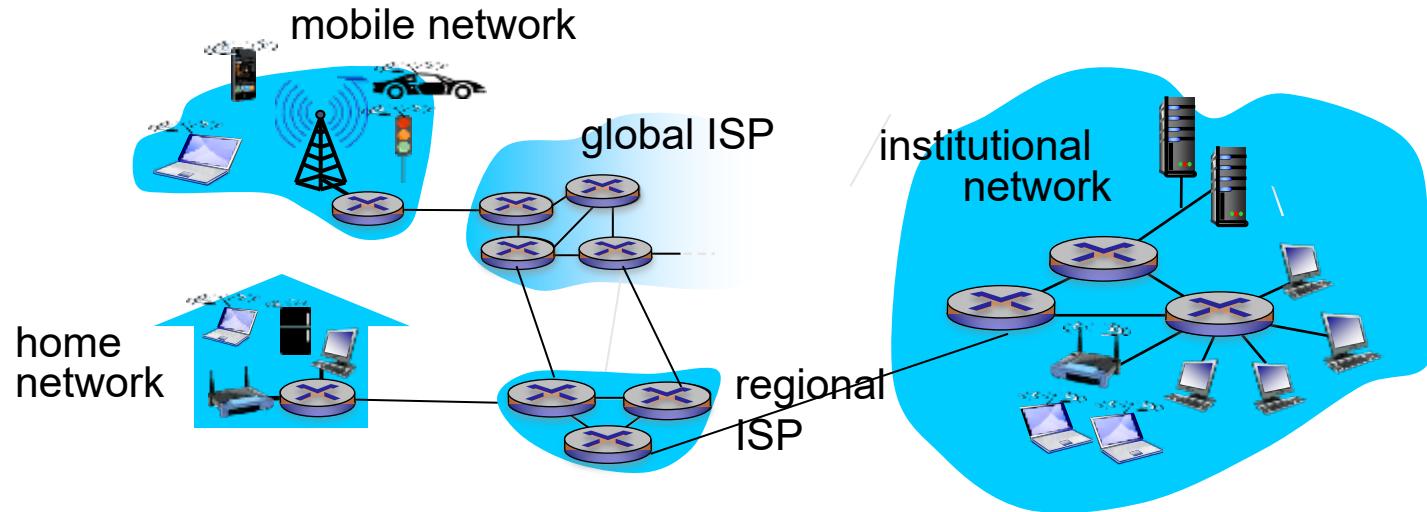


packet switching

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



# What's the Internet (I) : “nuts and bolts” view



## Internet:

- “network of networks”
- loosely hierarchical
- Interconnected ISPs

## protocols

- control sending, receiving of messages
- e.g., TCP, IP, HTTP, Skype, 802.11

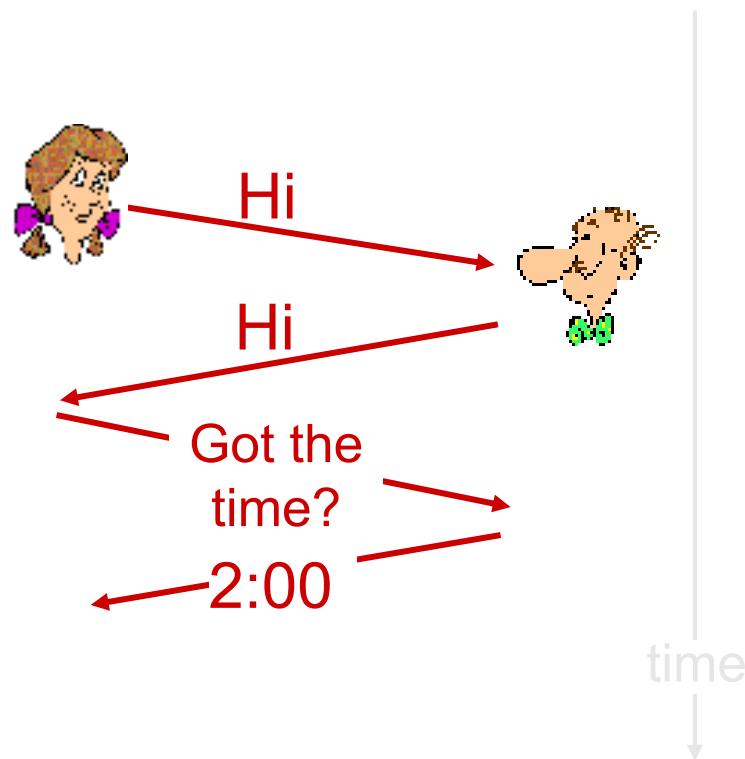
## Internet standards

- IETF: Internet Engineering Task Force
- RFC: Request for comments

# Internet is all about protocols

## *human protocols:*

- “what’s the time?” protocols

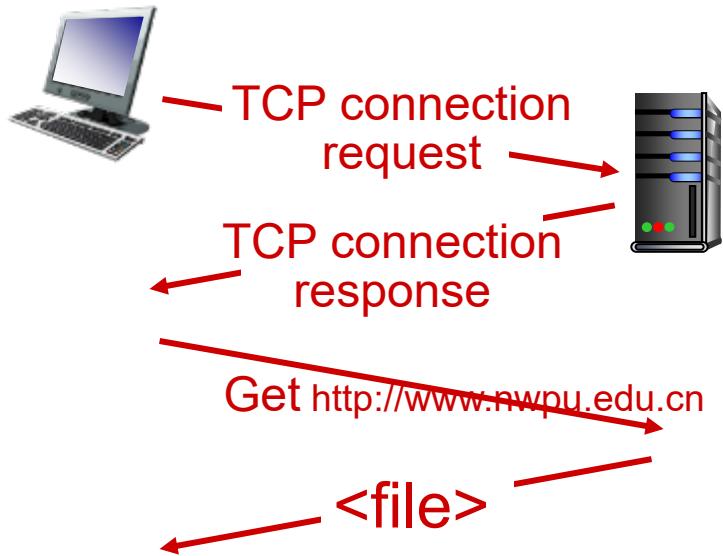


… specific messages sent  
… specific actions taken  
when messages received,  
or other events

# Internet is all about protocols

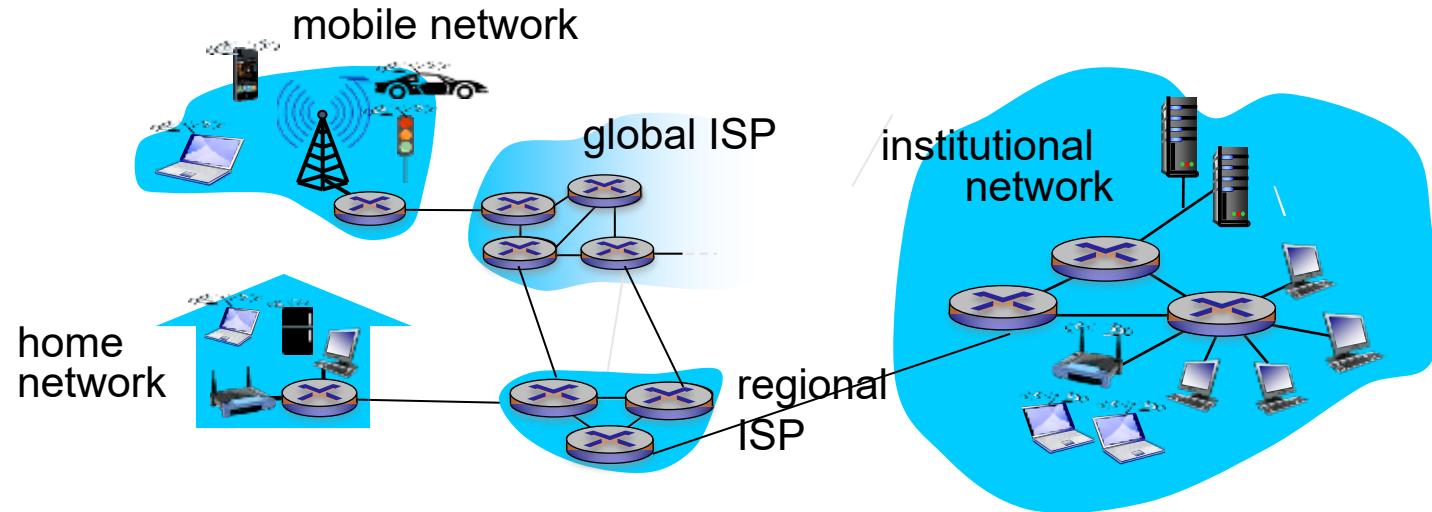
## *network protocols:*

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



*protocols define format, order of messages sent and received among network entities, and actions taken on message transmission, “receipt”*

# What's the Internet (II) : a service view



*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

# Two important things

**Multiple types of hardware**

e.g., PCs, smartphones, smart sensors..

**Multiple types of applications**

e.g., data, voice, videos, images, etc...

# Two important features

## Connectivity

- possible to exchange information (data, and various audio and video) between users, as if these users' computers can be directly connectable
- “virtual”: impossible to know exactly who and where the other is

## Sharing

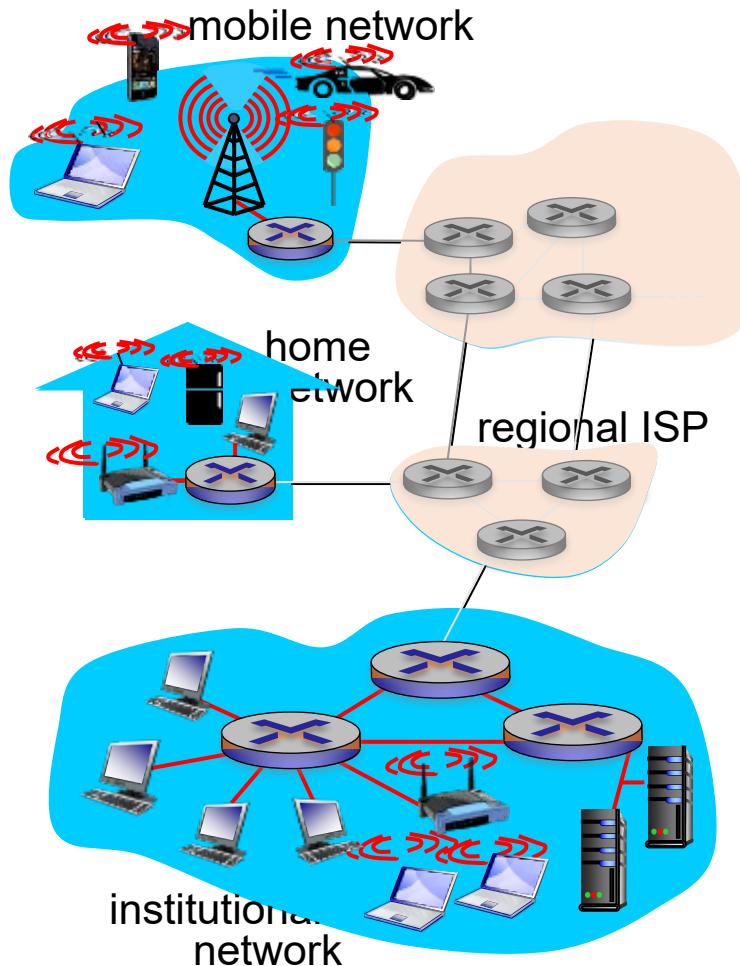
- resource sharing
  - Information sharing
  - Software sharing
  - Hardware sharing
- Due to the existence of the network, resources seem to be at the user's side and are easy to use.

# Chapter Outline

- 1 what is the Internet
- 2 network edge
  - End system, access networks, physical media
- 3 network core
- 4 delay, loss, throughput in networks
- 5 protocol layers, service models
- 6 history

# Network edge

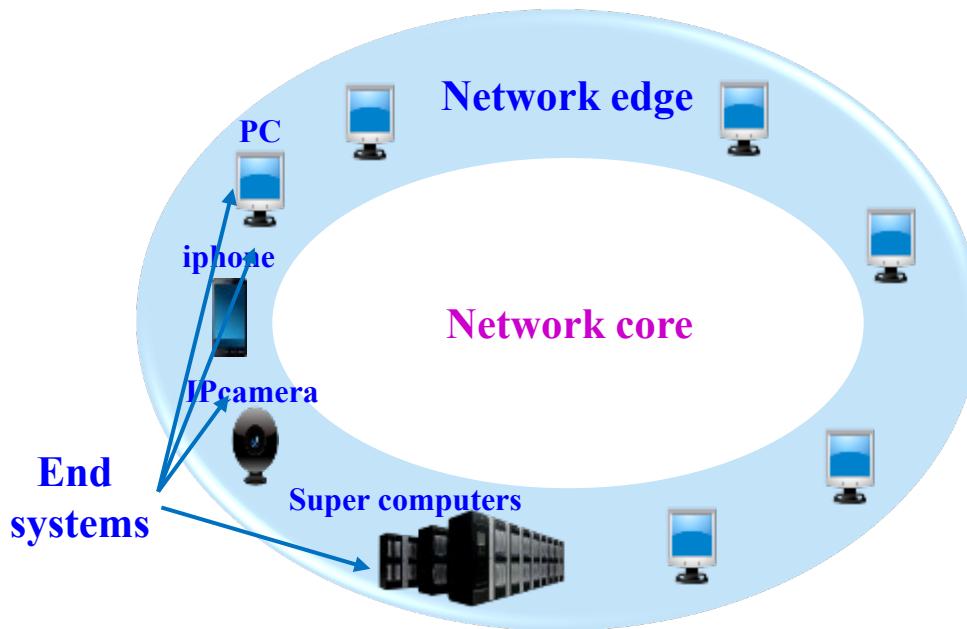
- Devices
- Access networks
- Physical media



# hosts/end systems

end systems can **vary widely in functionality**

- PC/smartphone/webcam
- High-performance computers
- different owner: individual or a unit



2.5 million servers – Gartner 2016

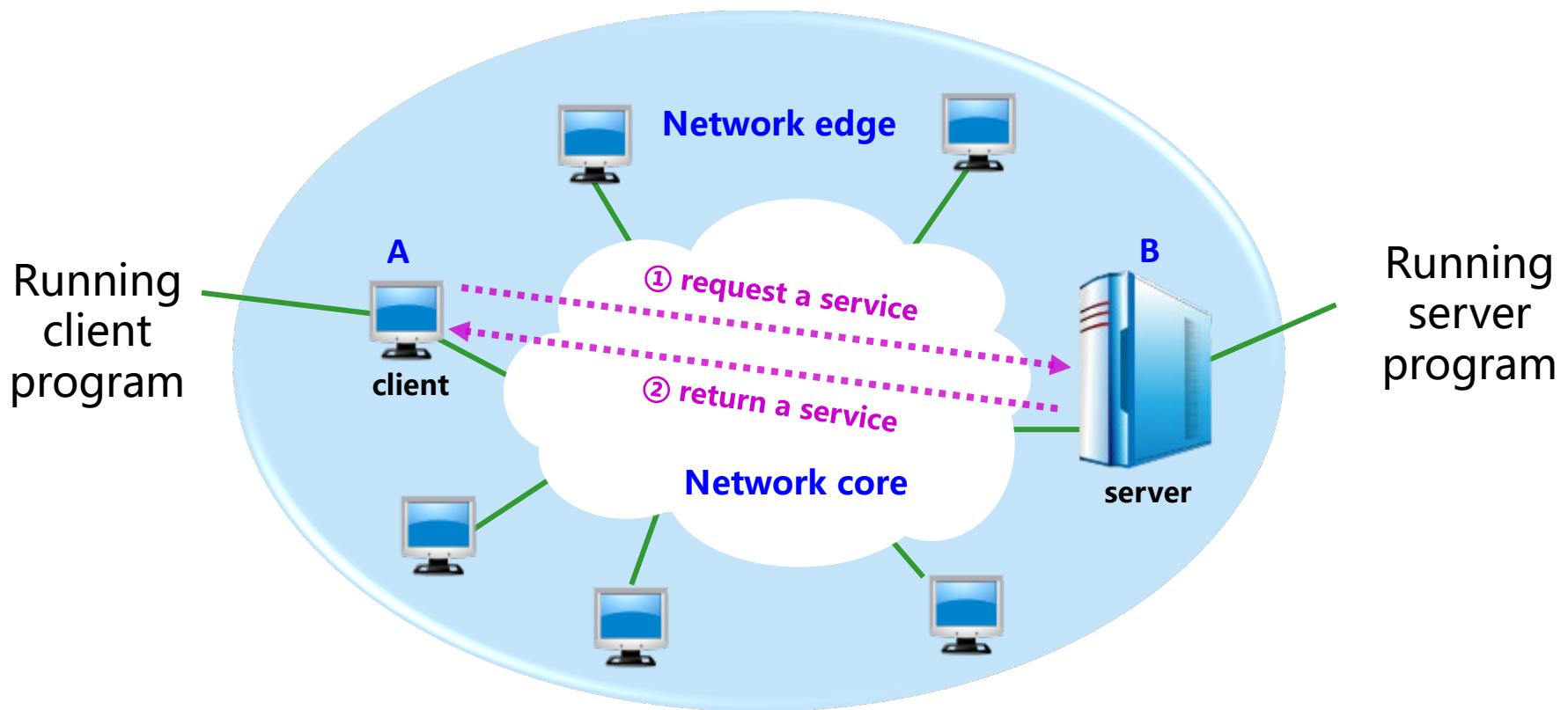
# Host A communicates with host B

- actually means: "A **program** running on **host A** communicates with **another program** running on **host B**".
- Two communication mode:

**Client/Server or C/S**

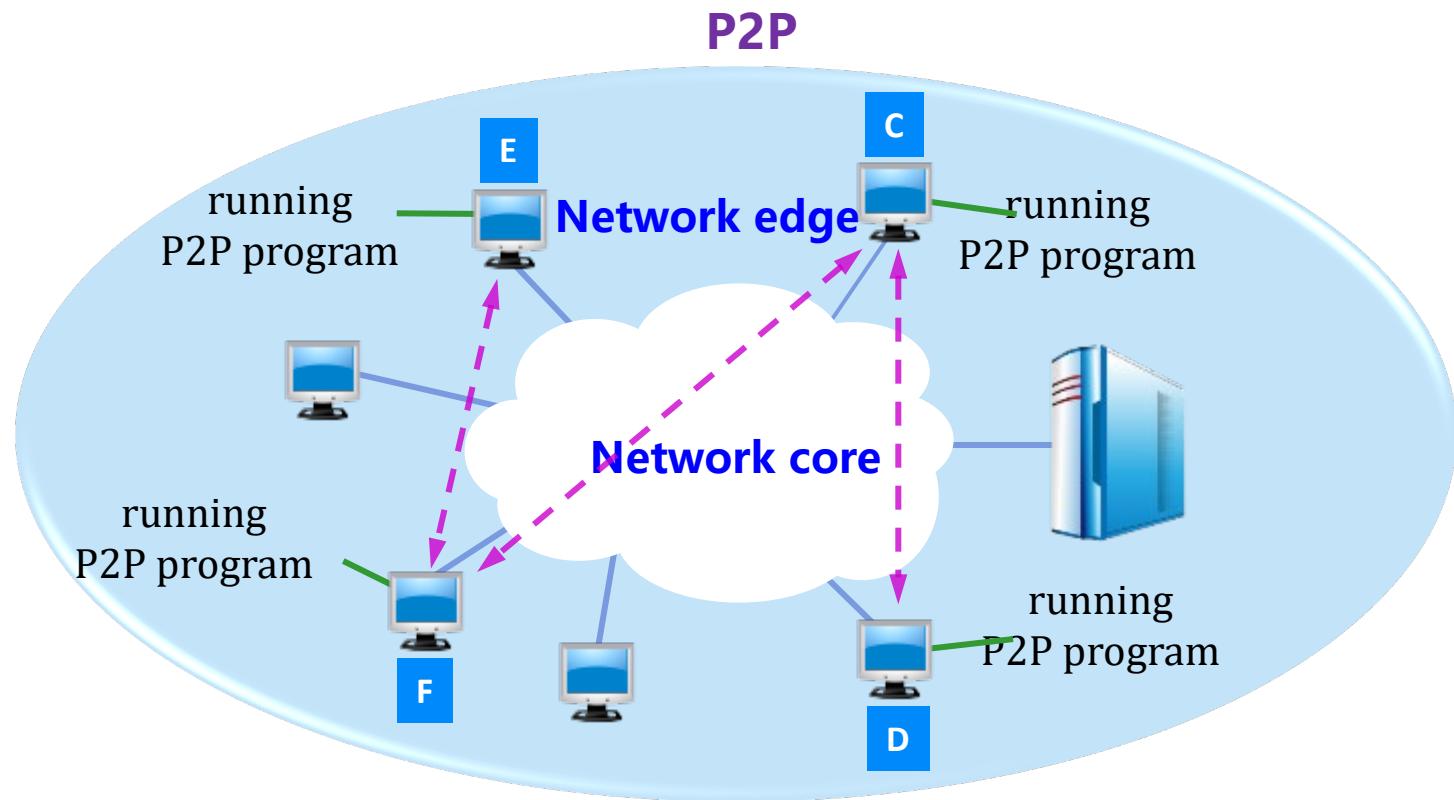
**Peer-to-Peer or P2P**

# Client – server mode



**Host A: request service from Host B  
Host B: return service to Host A**

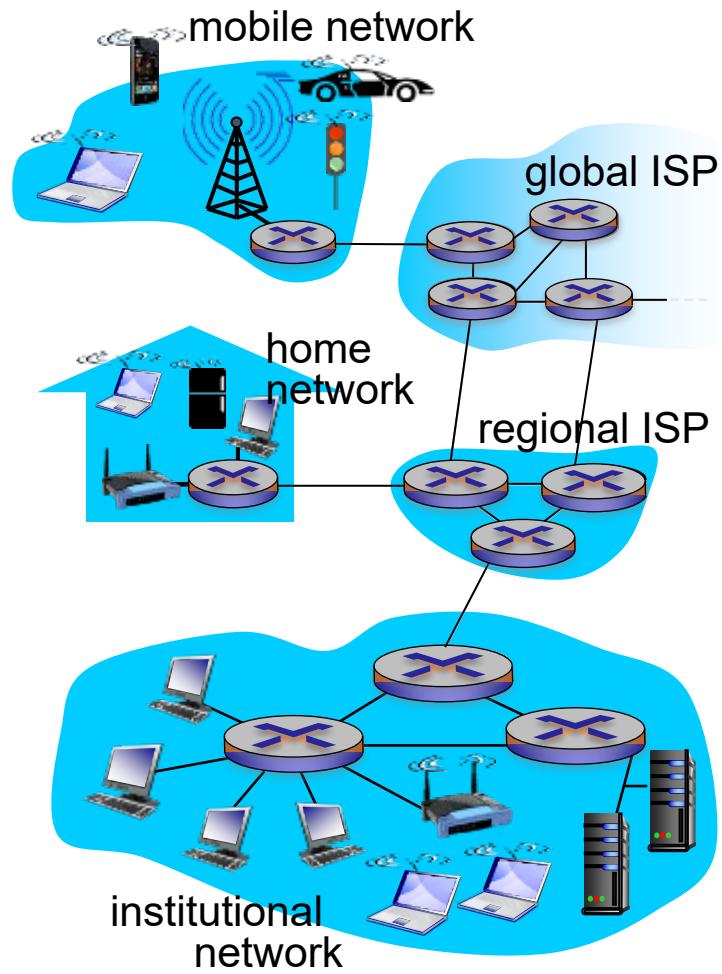
# Peer to peer mode



**each party has the same capabilities**

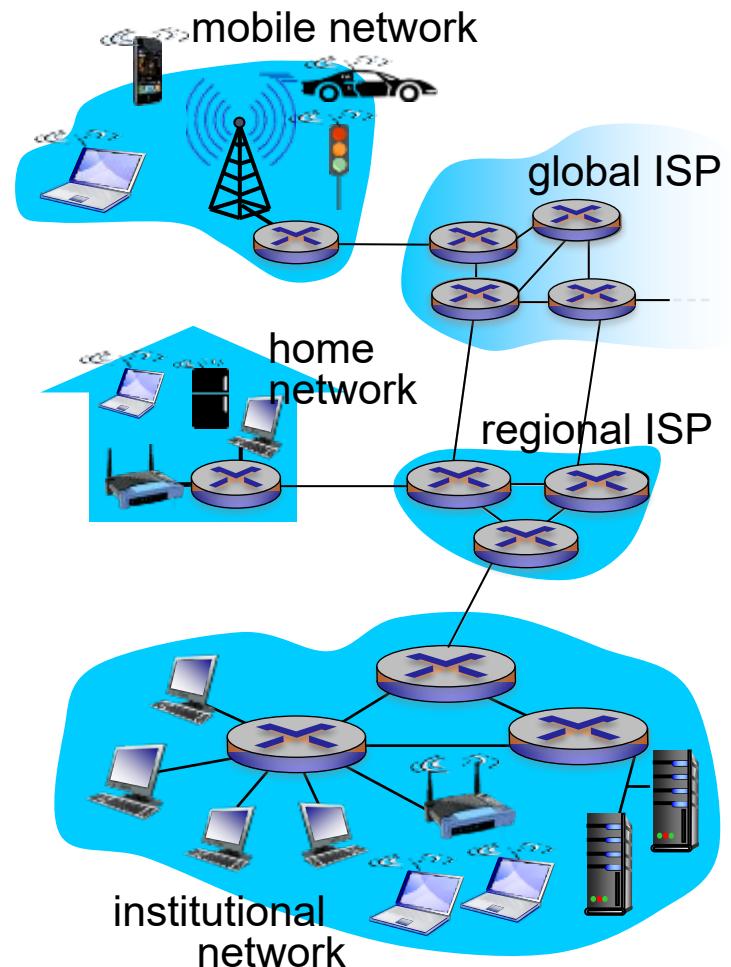
# Network edge

- Devices
- Access networks
- Physical media



## *access networks*

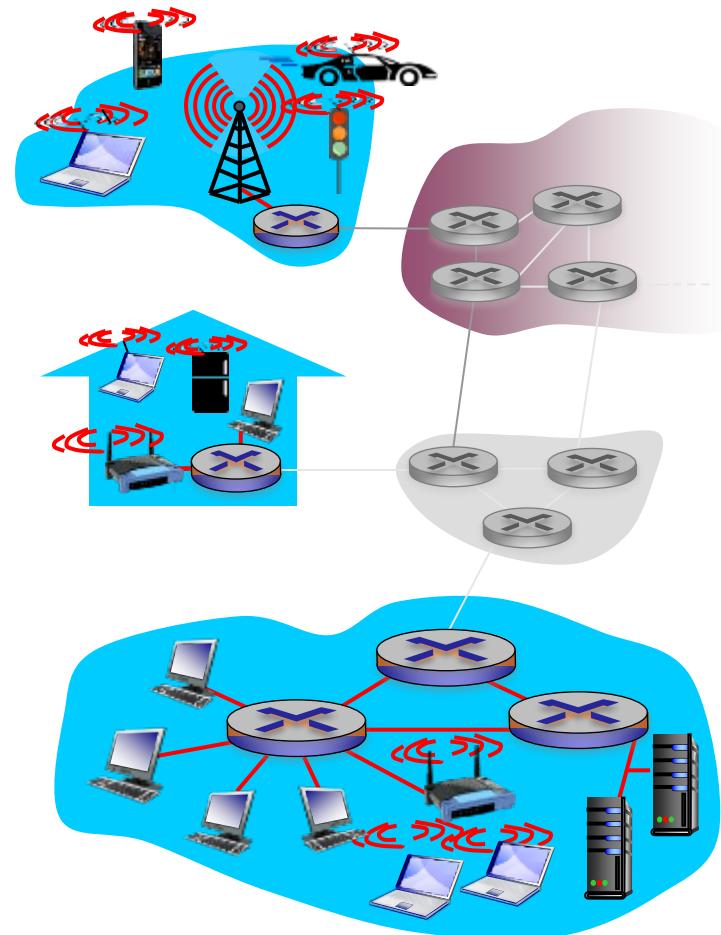
- *access network* is the network that physically connects an end system to the first router
- the 1<sup>st</sup> router is also named as the edge router
- Via physical media
  - Wired comm links
  - Wireless comm links



# Access networks

*How to connect end systems  
to edge router?*

- residential access nets
  - DSL, Cable, etc
- institutional access networks (school, company)
- mobile access networks



# DSL - digital subscriber line

- DSL offers high-speed internet service for homes and businesses

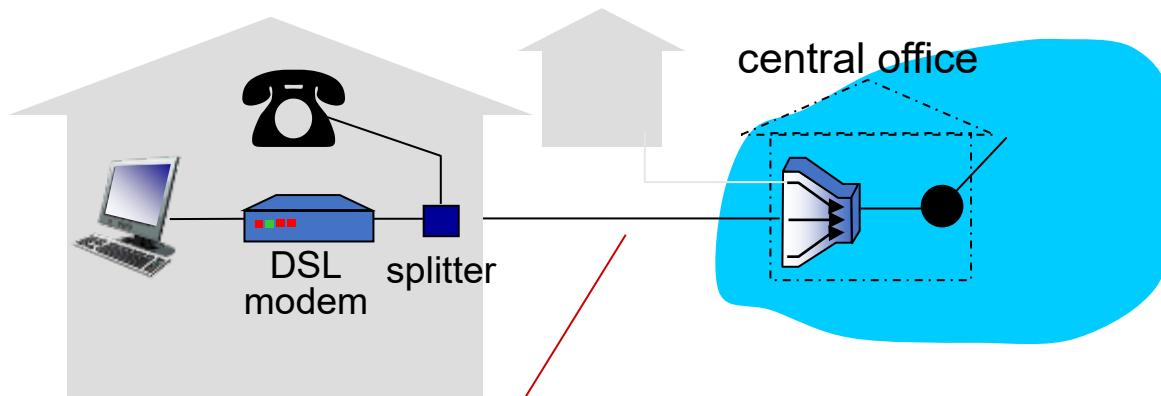


**verizon**<sup>✓</sup>

**Frontier**<sup>TM</sup>  
COMMUNICATIONS

| Provider                    | Est. Population Covered | Max Speed |
|-----------------------------|-------------------------|-----------|
| AT&T Internet               | 120,456,246             | 100 mbps  |
| EarthLink                   | 99,191,941              | 100 mbps  |
| Verizon High Speed Internet | 48,844,244              | 15 mbps   |
| CenturyLink                 | 48,602,571              | 140 mbps  |

# DSL



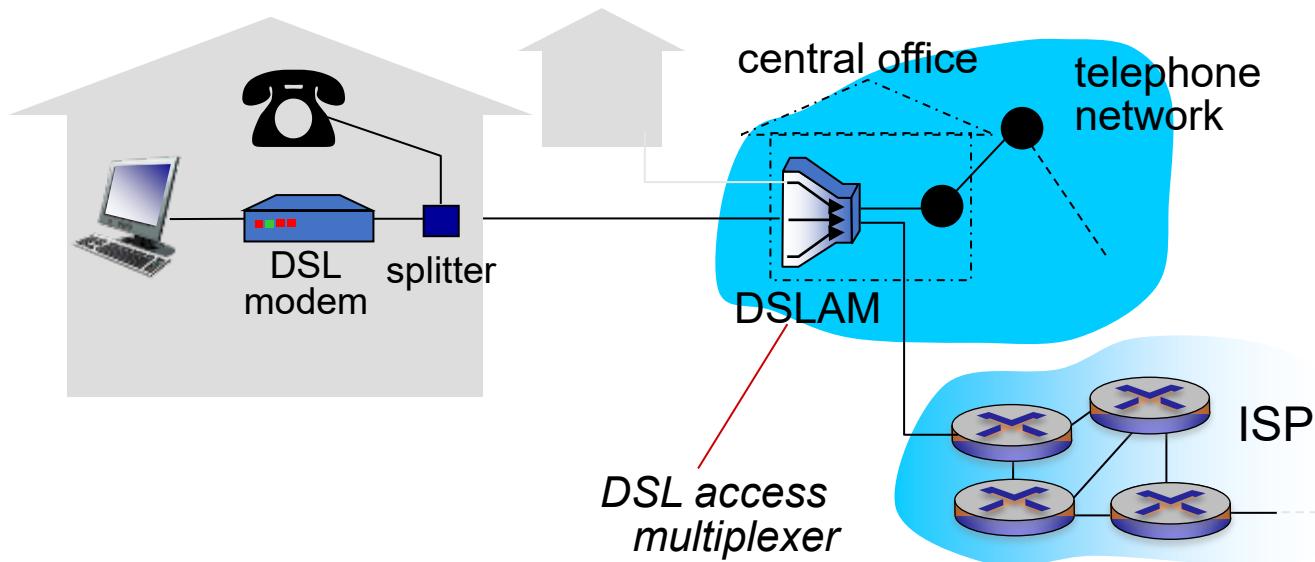
*voice, data transmitted  
at different frequencies over  
**dedicated** line to central office*

- Home side: your network & telephone service share the same phone line without disrupting either your voice or network connections



DSL modem

# DSL

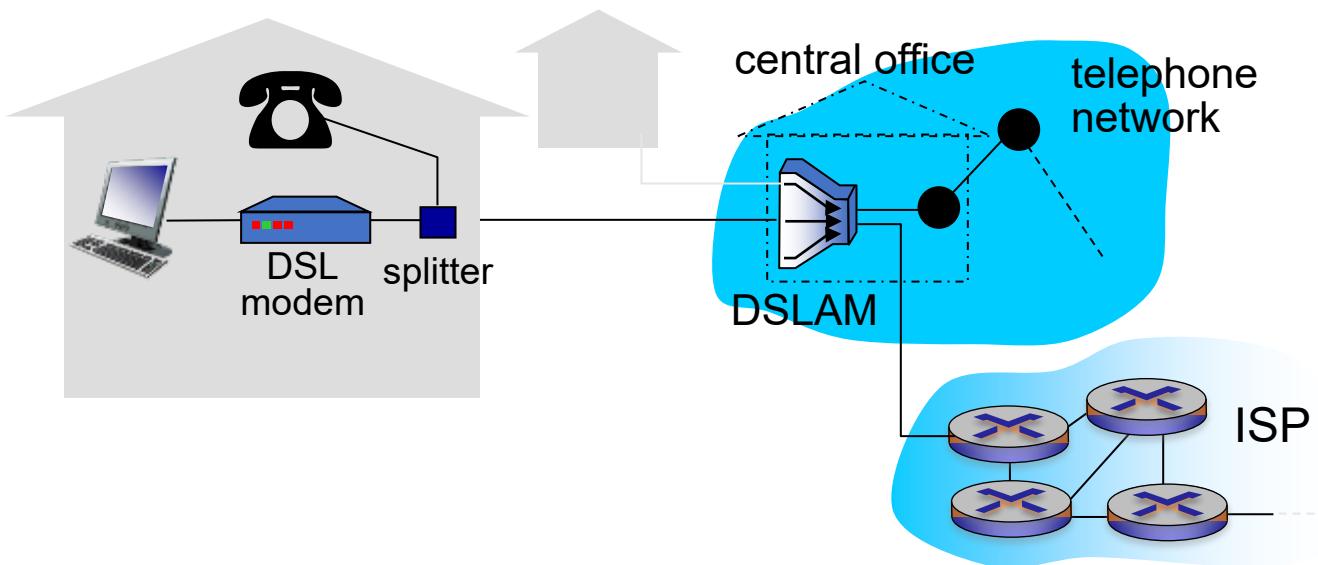


- Central office side : use DSLAM to separate data & voice
  - data over DSL phone line goes to Internet
  - voice over DSL phone line goes to telephone net



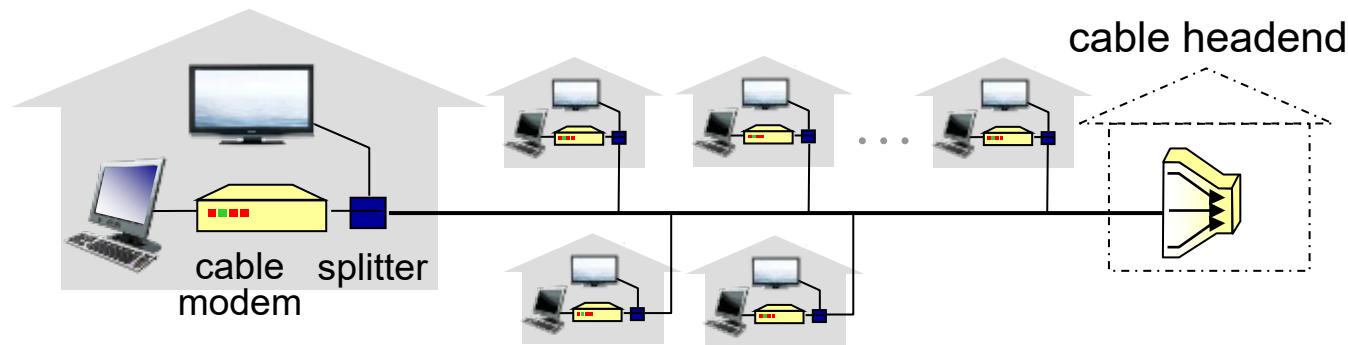
# DSL summary

- Use the same infrastructure as a telephone
- Dedicated access from home to central office
- DSL modem & DSLAM



# cable network

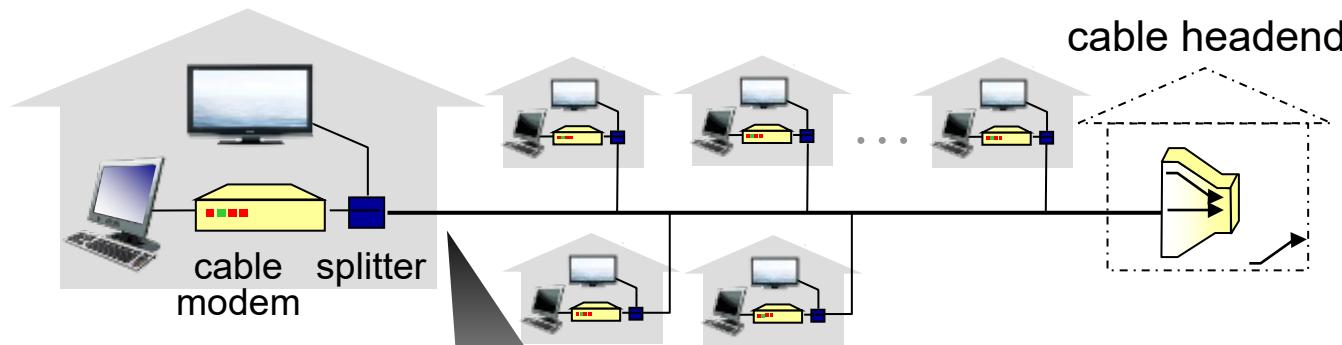
- **Cable Internet access/ cable Internet** is a form of broadband Internet access which uses the same infrastructure as a cable television



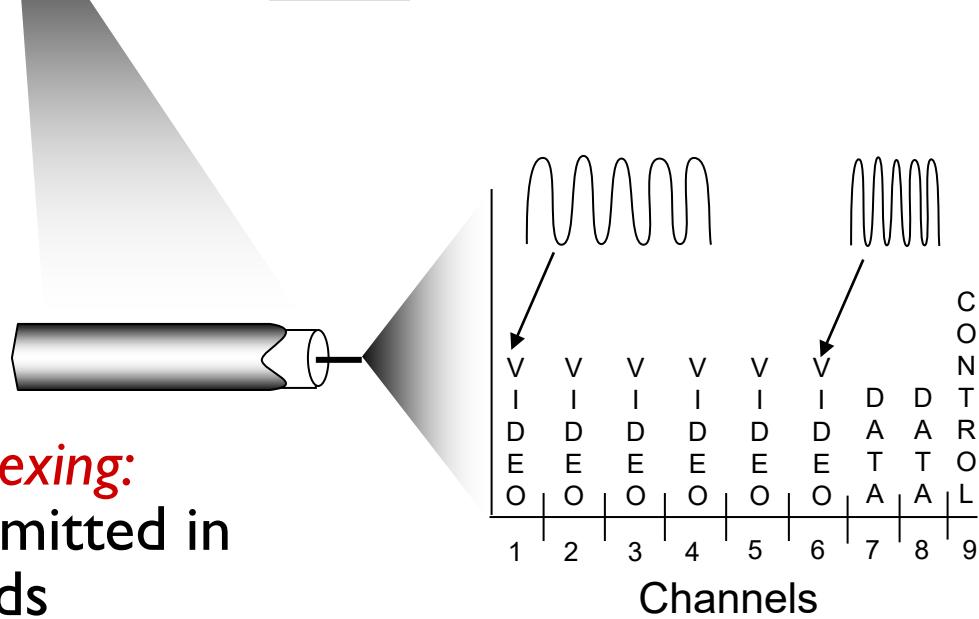
## Cable vs. DSL:

- Cable network: homes *share access network* to cable headend
- DSL: has dedicated access to central office

# cable network

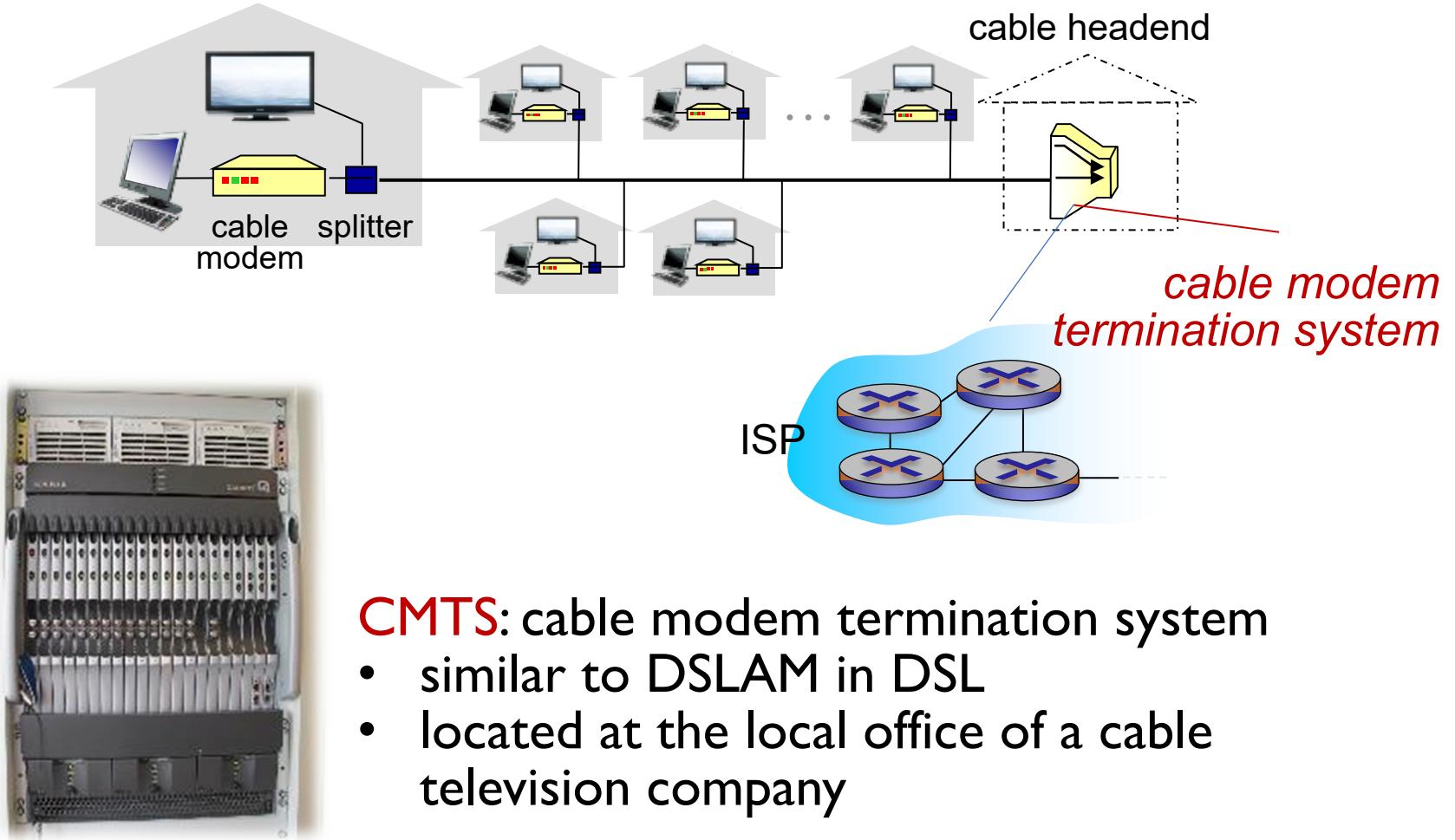


data, TV transmitted at  
different frequencies  
over shared cable  
distribution network



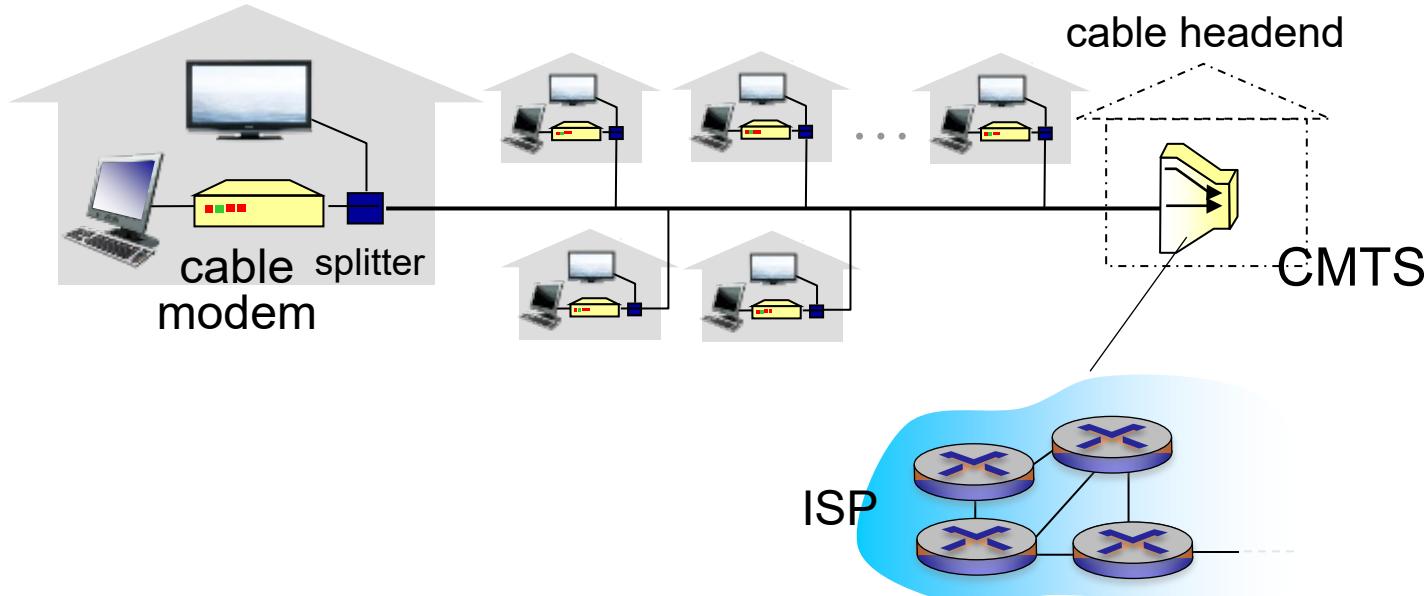
**frequency division multiplexing:**  
different channels transmitted in  
different frequency bands

# cable network



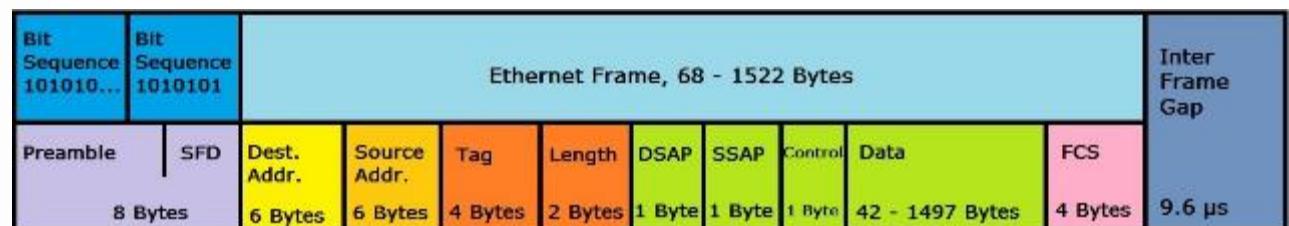
# Cable network summary

- Use the same infrastructure as **television**
- **shared** access from homes to central office
- cable modem & CMTS



# Ethernet

- Ethernet offers a standard way to link the computers via a wired connection to the network.
- It is an interface connecting multiple devices.
- Ethernet is used for Local Area Networks/LANs
- Ethernet transmits the data in frame form.



## Ethernet



Example of Ethernet is LAN  
(Local Area Network).

## Internet



Example of Internet is WAN  
(Wide Area Network).



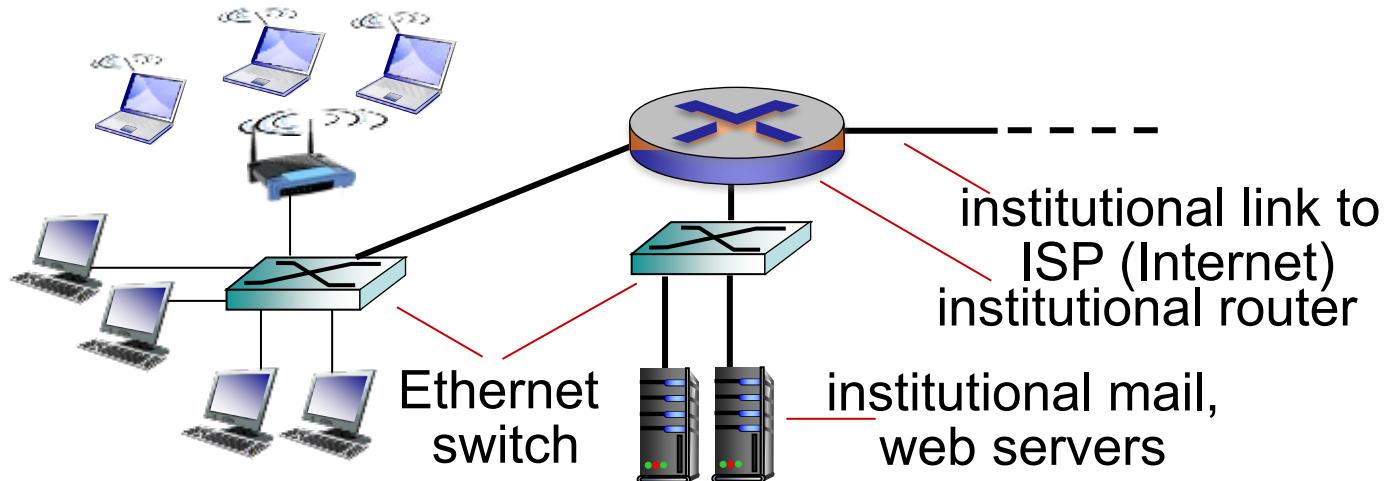
Ethernet is more secure  
because outside devices have  
no access to the network.



Internet is less secure as  
anyone can access the network  
and gain the information.

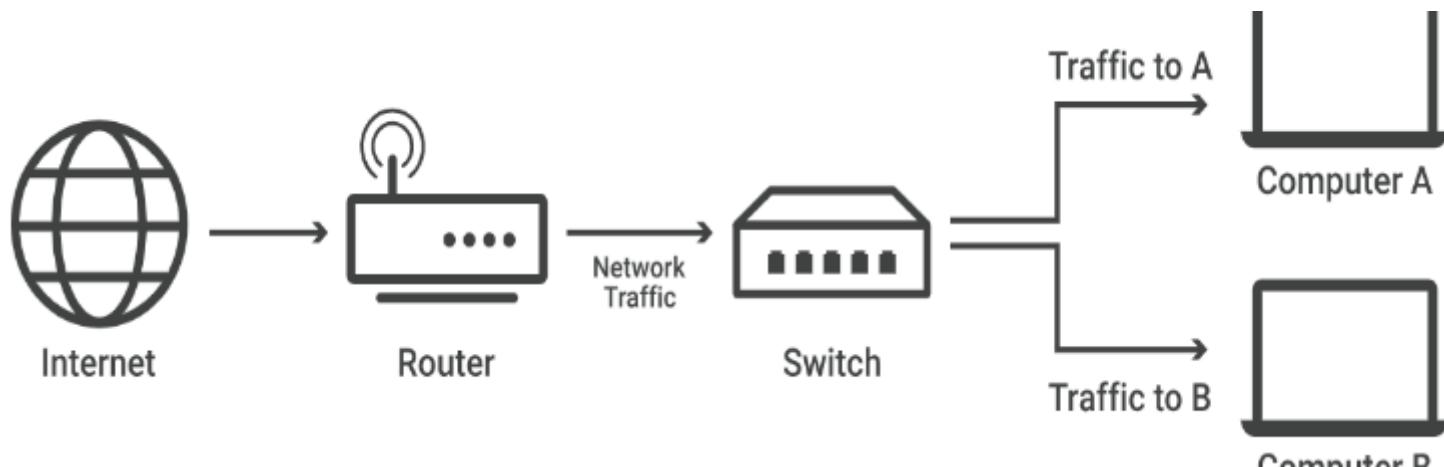
# Ethernet

- Originally typically used in companies, universities, etc.
- 10 Mbps, 100Mbps, 1Gbps, 10Gbps transmission rates
- today, end systems typically connect via coaxial into Ethernet **switch**



# Switch vs Router

- **switch** connects multiple devices to create a network
- **router** connects multiple switches, and their respective networks, to form an even larger network..



# Wireless access networks

- **WLAN**: wireless LANs connects end system to router via the **access point (AP)**
- standalone or integral AP



Cisco Aironet wireless access point



Linksys "WAP54G"  
802.11g wireless router

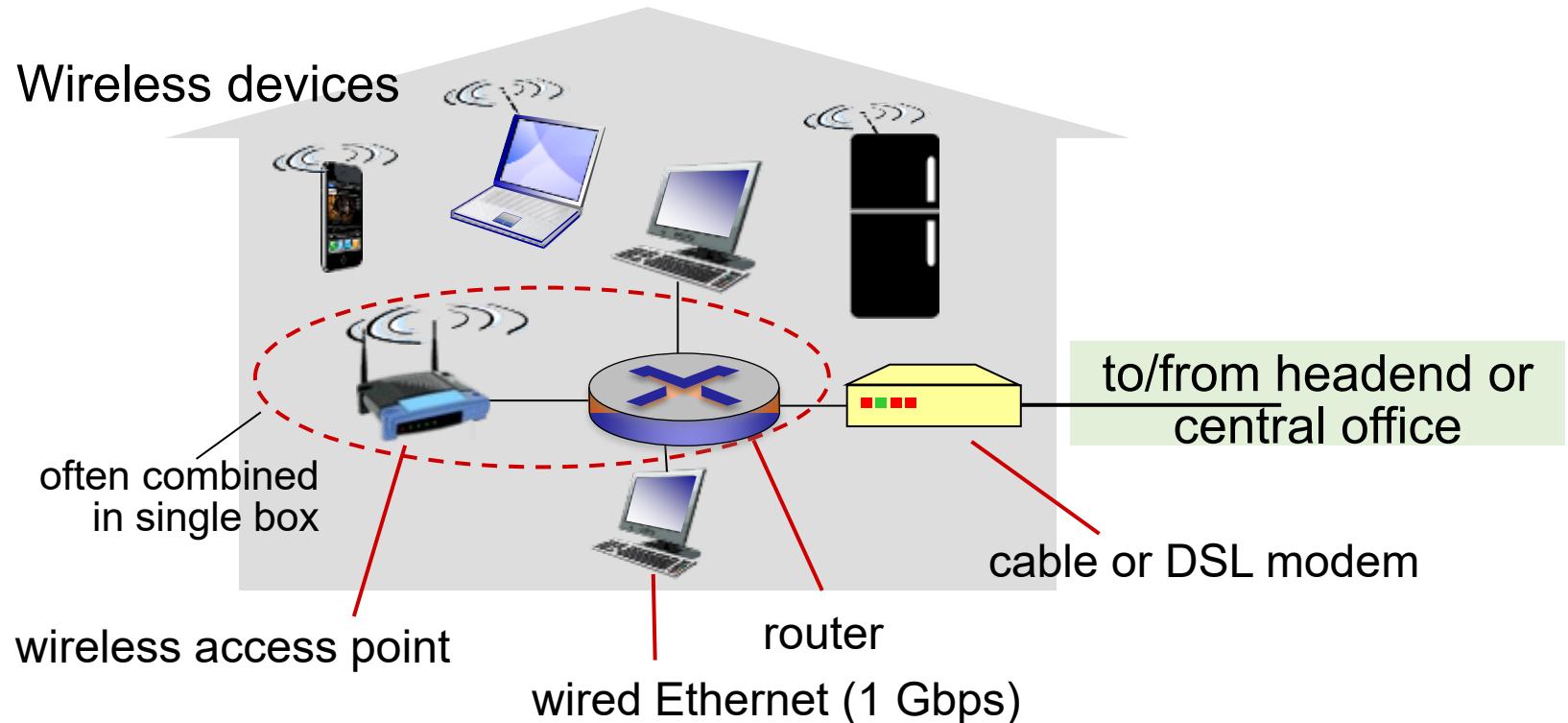
# WIFI

- **WIFI**: WLAN using IEEE 802.11 protocols.



# Access network: home network

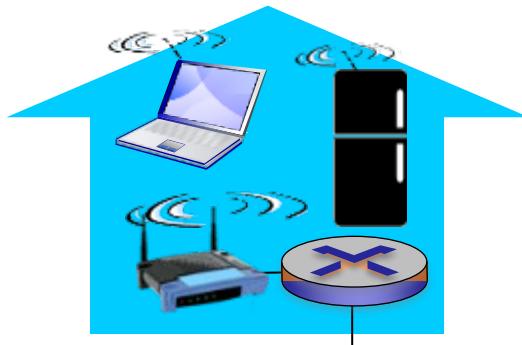
- Ethernet and Wifi: most popular LAN techniques



# Wireless access networks

## wireless LANs:

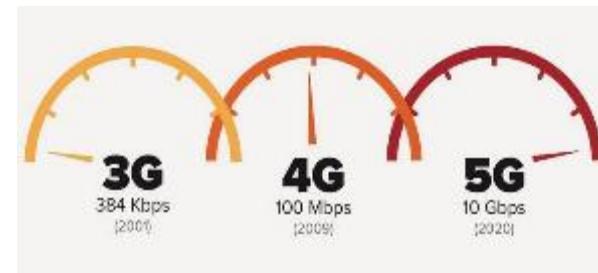
- within building (100 ft.)
- 802.11b/g/n (WiFi): 11, 54, 450 Mbps transmission rate



*to Internet*

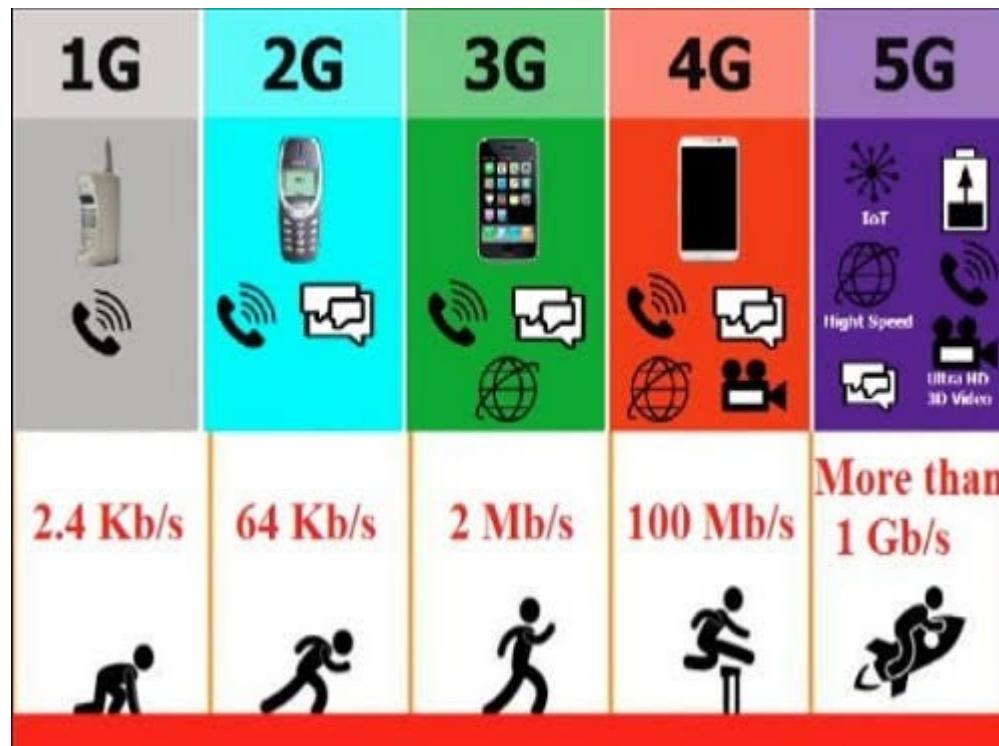
## wide-area wireless access

- provided by telco (cellular) operator,
- 3G, 4G, 5G



# 3G, 4G, 5G

- G : GENERATION



# TAKEAWAYS

- Internet overview
  - A Nuts-and-Bolts view
  - A service view
- Network Edge:
  - End system/hosts
  - Access Network
    - DSL and Cable network
    - LANs and Ethernet
    - wireless networks



西北工业大学

NORTHWESTERN POLYTECHNICAL UNIVERSITY



# Computer Networks

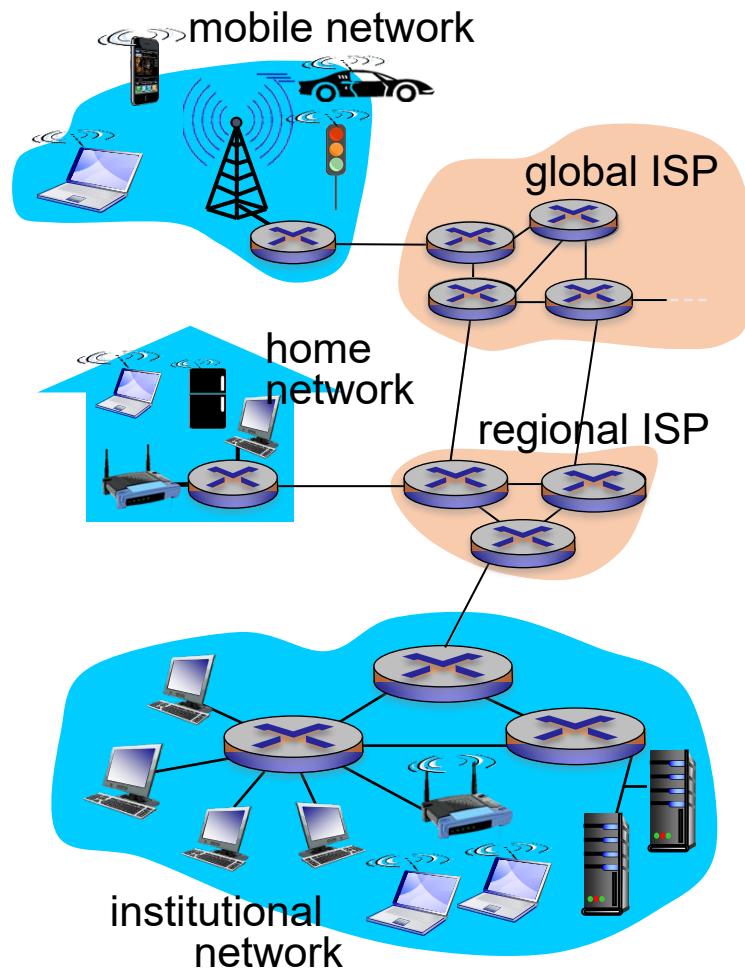
Lecturer: ZHANG Ying

Fall semester 2022

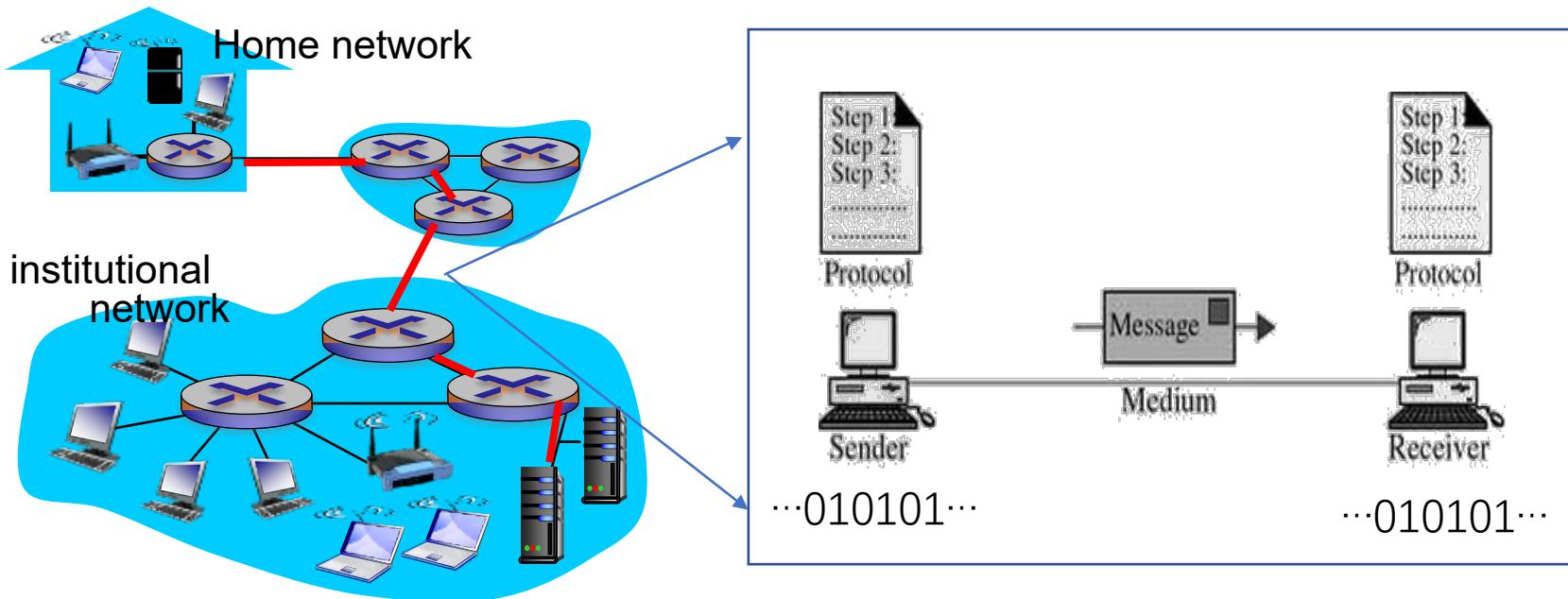
# Chapter 1: Introduction

# Network Edge

- Devices
  - end systems/hosts
- Access network:
  - end systems to edge router
- Physical media
  - Guided media
  - Unguided media



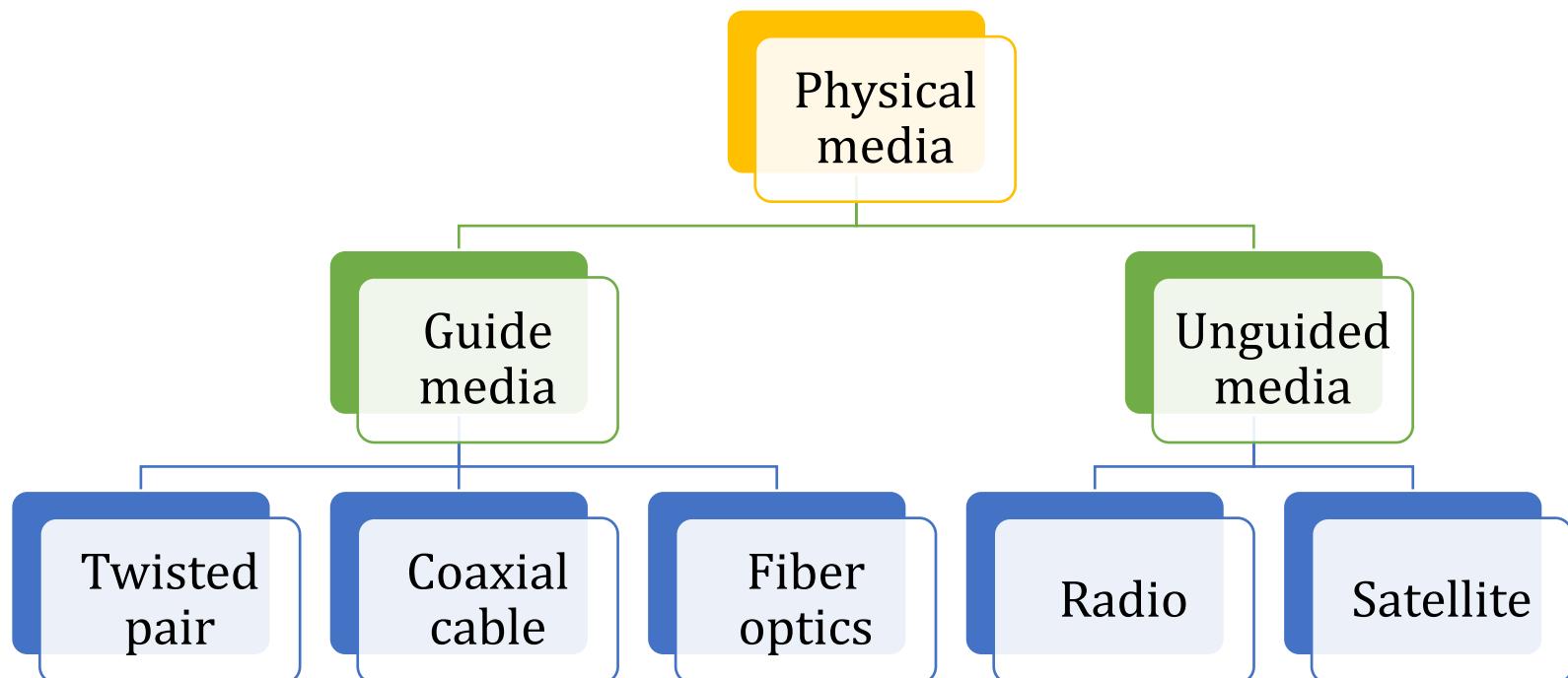
# Physical media – the journey of a “bit”



- Bits travel through **multiple sender-receiver pairs** from source to destination.
- Each pair via own physical media
- Various medias may exist.

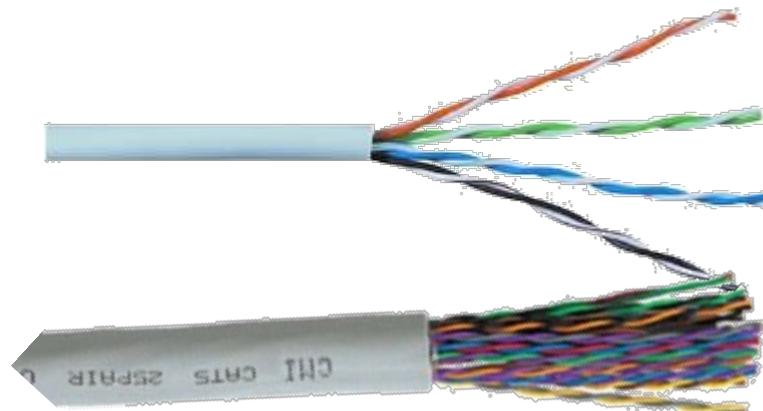
# Physical media

- **physical link:** lies between transmitter & receiver

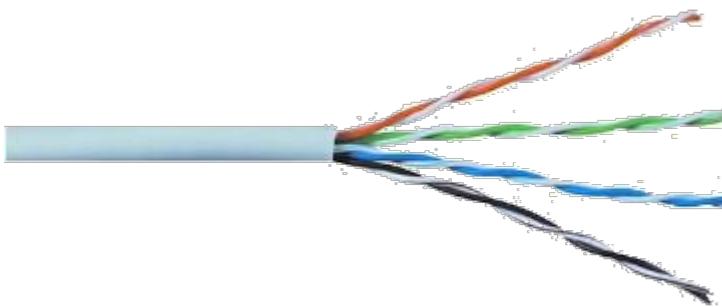


# Twisted-Pair Copper Wire

- most commonly-used physical media
- support both digital & analog signal



# Unshielded twisted pair cable/ UTP



**Outer jacket:**  
protect the copper  
wire from physical  
damage

**Twisted-pair:**  
protect the signal  
from interference

**Color-coded plastic  
insulation:**  
electrically isolated  
wires from each  
other and identifies  
each pair

# Category 3 UTP & Category 5 UTP



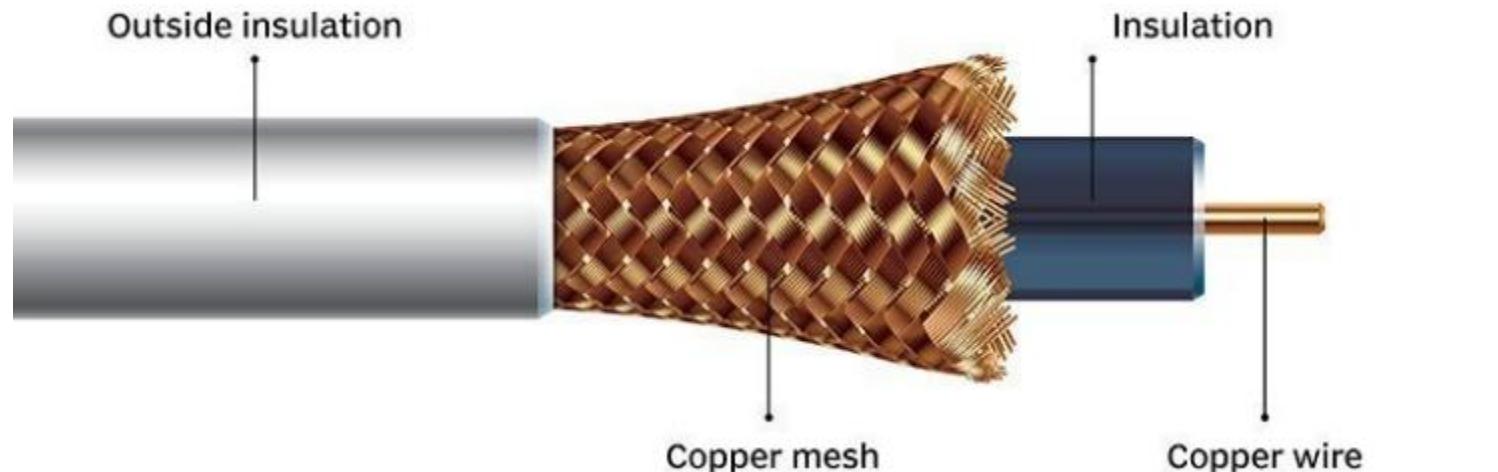
(a) Category 3 UTP.



(b) Category 5 UTP.

# Coaxial-Cable

- Special insulation and shielding to well block the signal interference.
- coaxial cable can have higher bit rates than twisted pair.



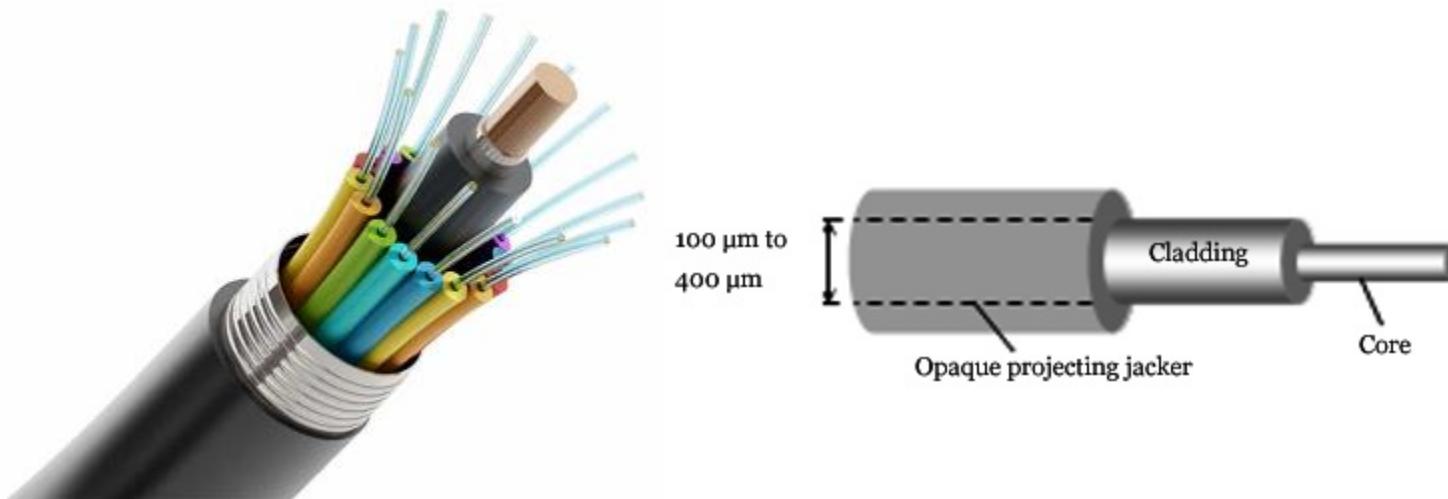
# baseband & broadband coaxial cable

Two common categories

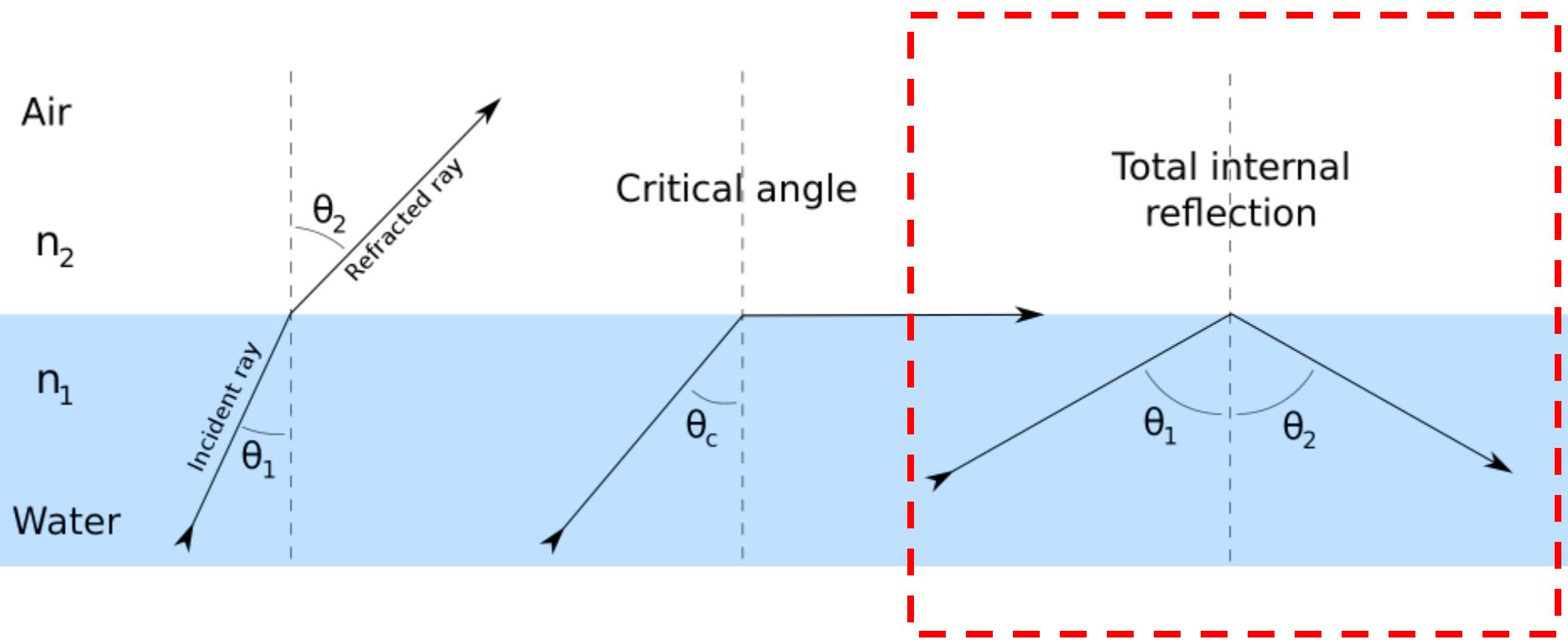
- 50-ohm cable/baseband
  - LAN
  - digital transmission
- 75-ohm cable/broadband
  - cable television systems
  - analog transmission

# Fiber Optics

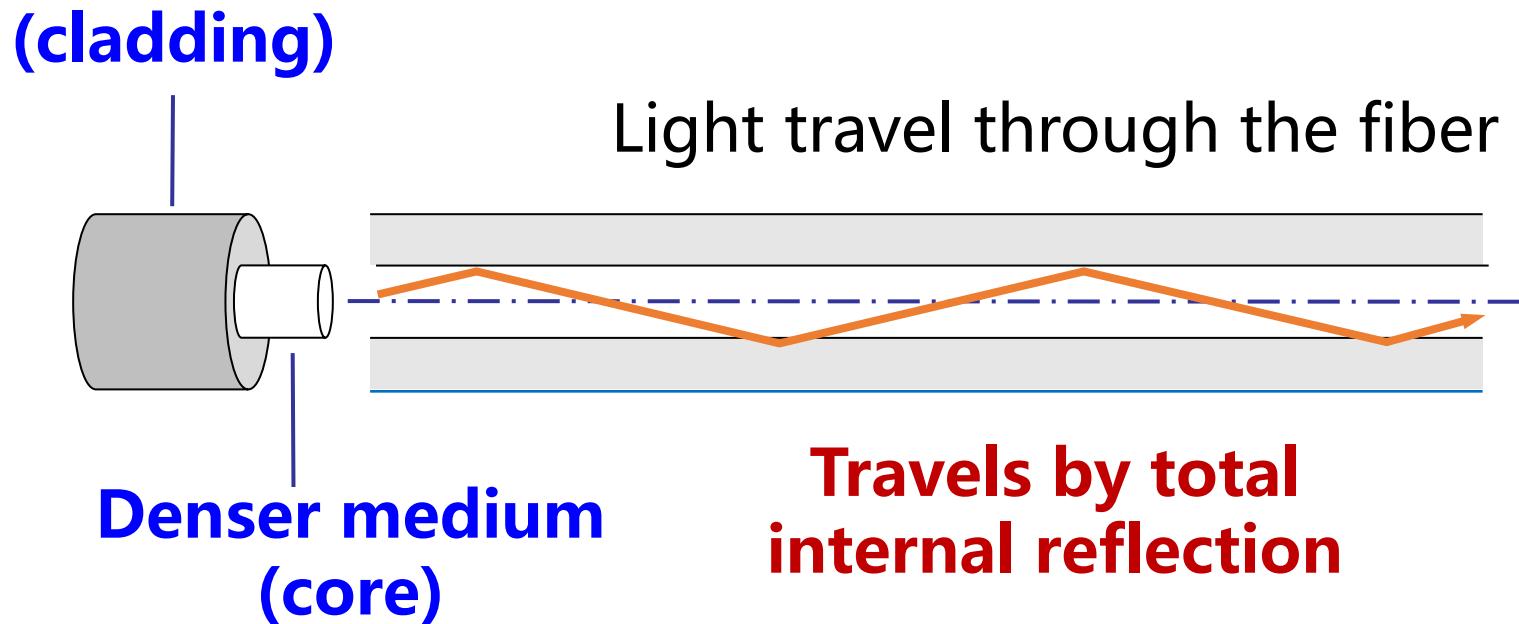
- optical fiber: light pulse represents a bit
- low signal attenuation up to 100 kilometers - Preferred long distance transmission media
- But with high cost



# Working of Optical Fiber



# Working of Optical Fiber



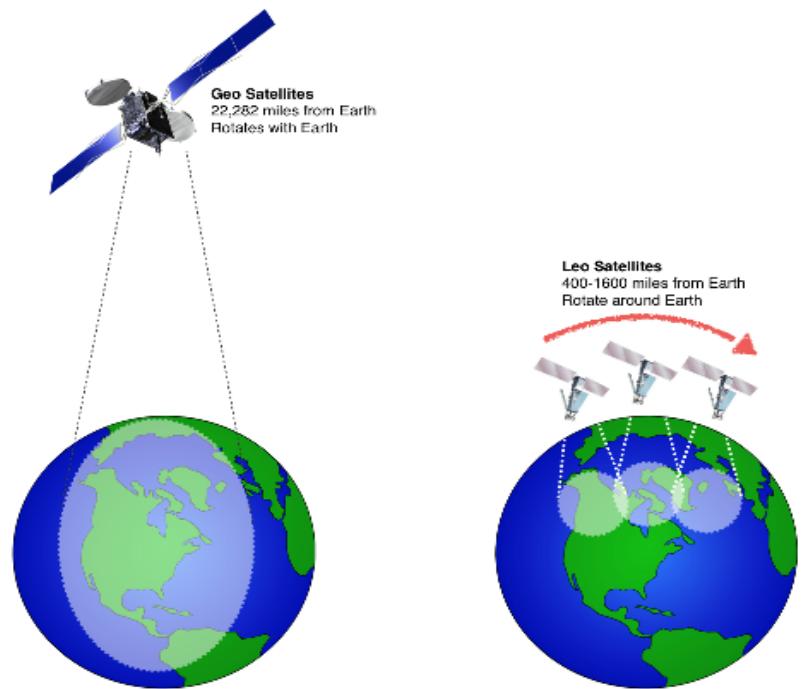
# Terrestrial Radio

- carry signal carried in electromagnetic spectrum
- no physical “wire”
- LAN (e.g., WiFi)
  - 54 Mbps
- wide-area (e.g., cellular)
  - 4G cellular: ~ 10 Mbps

# Satellite Radio Channels

## ■ satellite

- Bandwidth
- end-end delay
- Geostationary vs  
Low altitude satellite



# Outline

1

what is the Internet

2

network edge

3

network core:

packet switching, circuit switching, message switching

4

delay, loss, throughput in networks

5

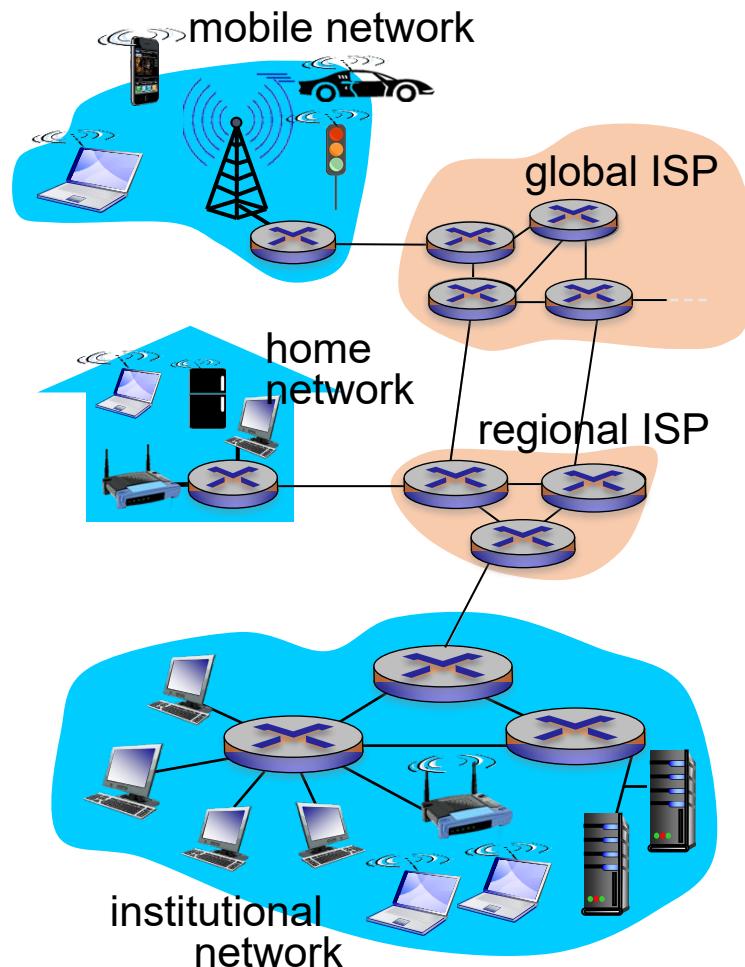
protocol layers, service models

6

history

# Network core

- **Network core / backbone network** is the mesh of routers that interconnect the Internet's end-systems
- Build via:
  - Circuit switching
  - Packet switching
  - (message switching)

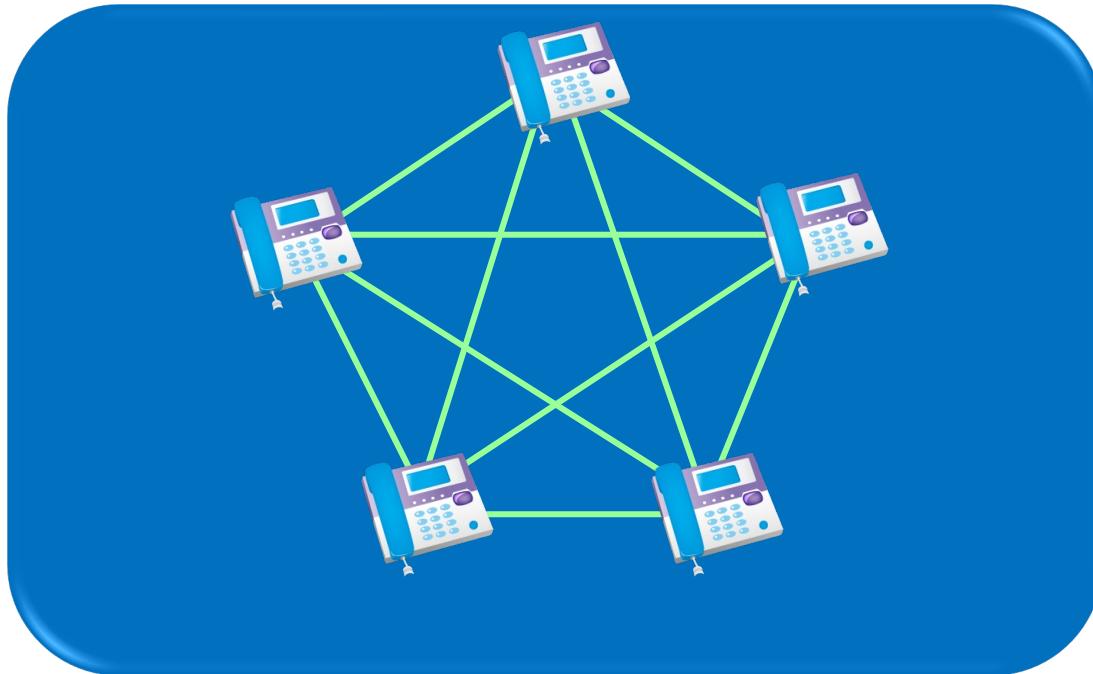


# How to connect two phones?



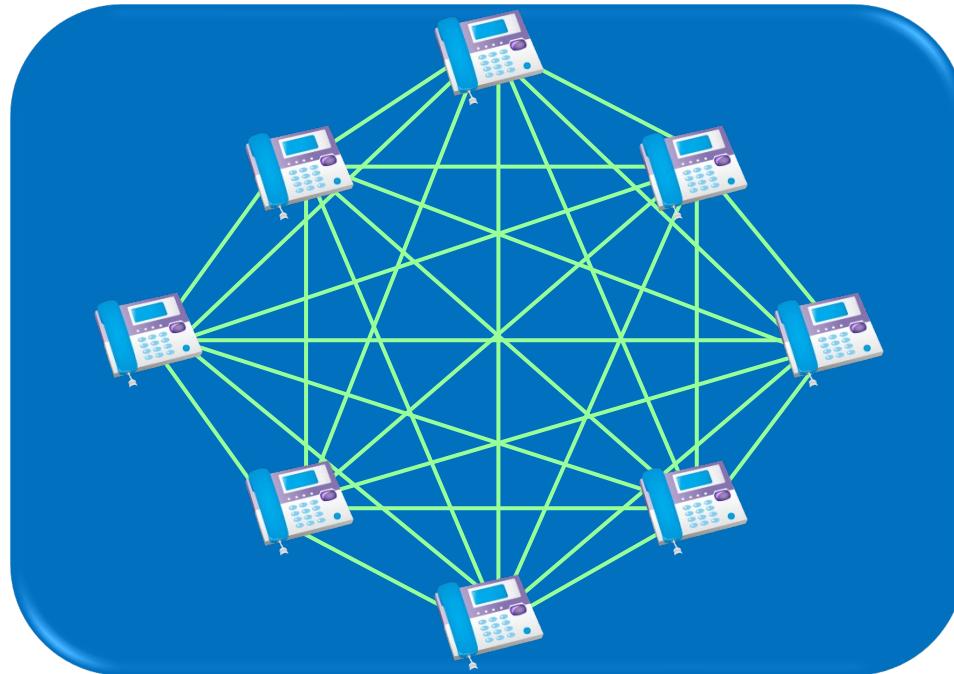
**Use a single telephone cable to connect 2 phones**

# How to connect 5 phones?



**10 cables to connect 5 phones**

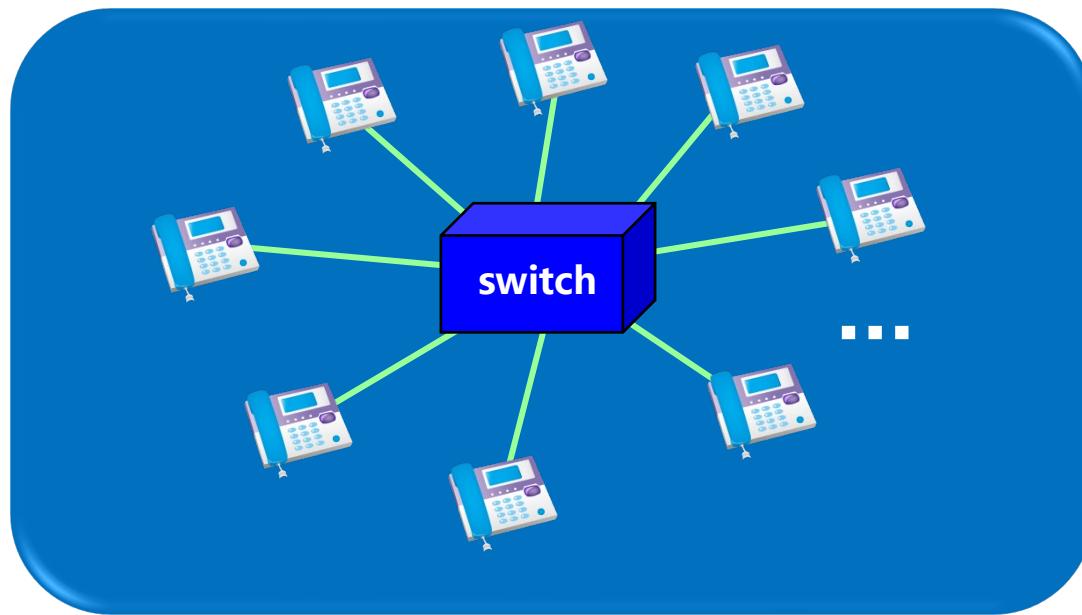
# How to connect N phones?



$N(N-1)/2$  cables to connect N phones

A huge number!

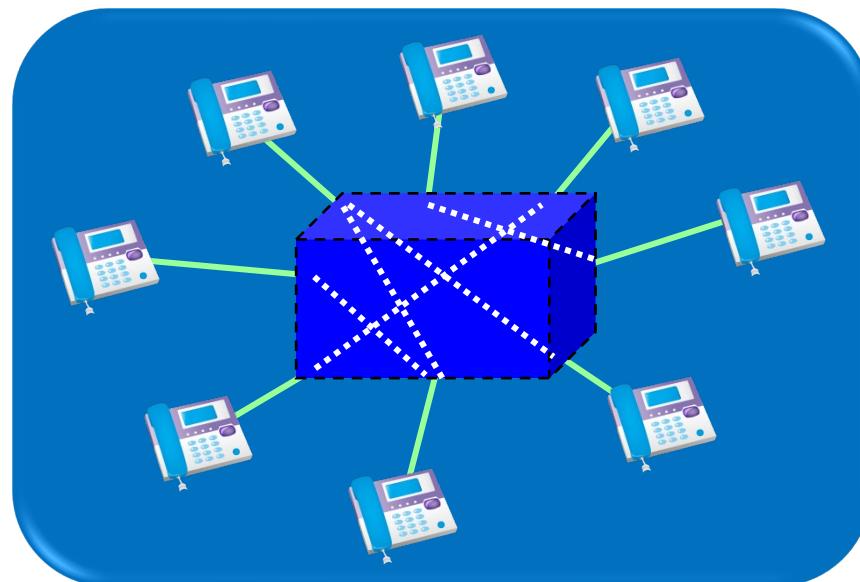
# How to connect N phones **efficiently**?



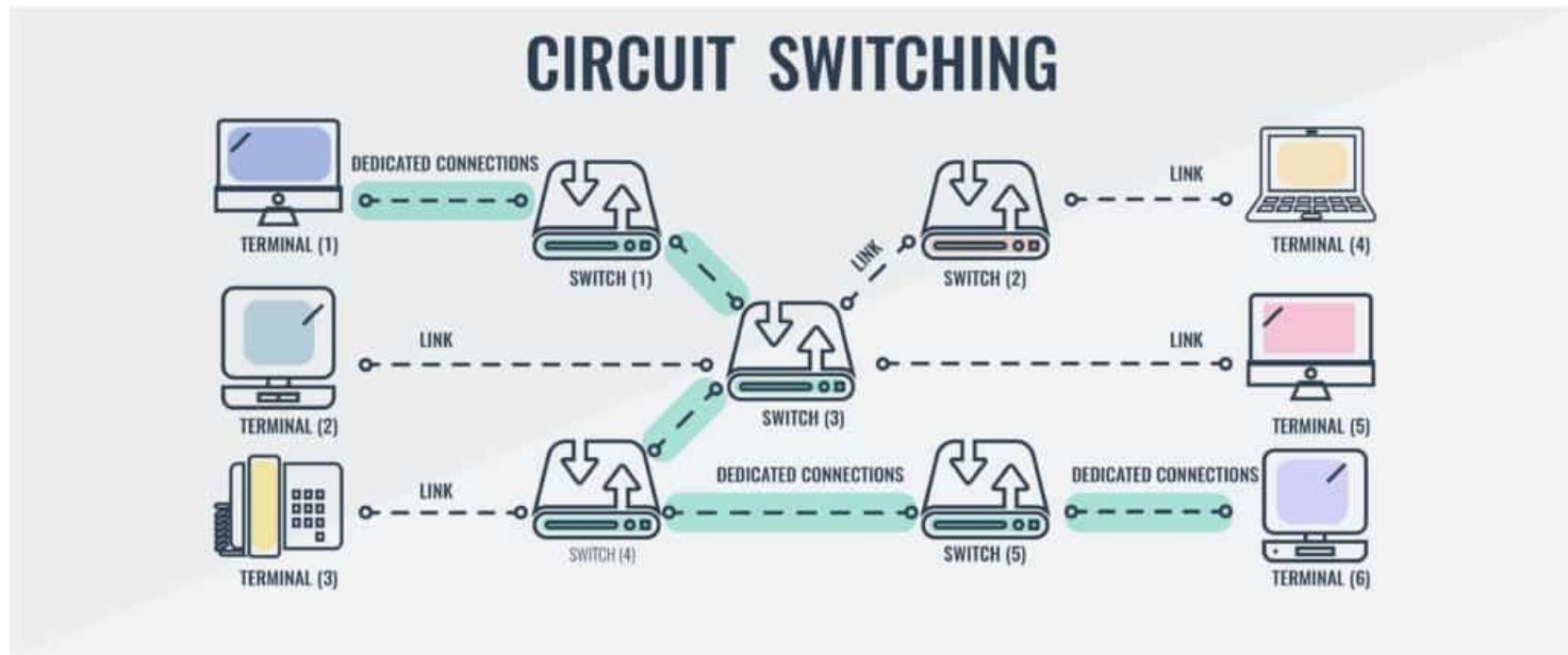
**Use a switch! – this is named as “circuit switching”**

# About switching

- Switching: **transfer** one phone line to another so that they are connected
- allocate the resources of the transmission line



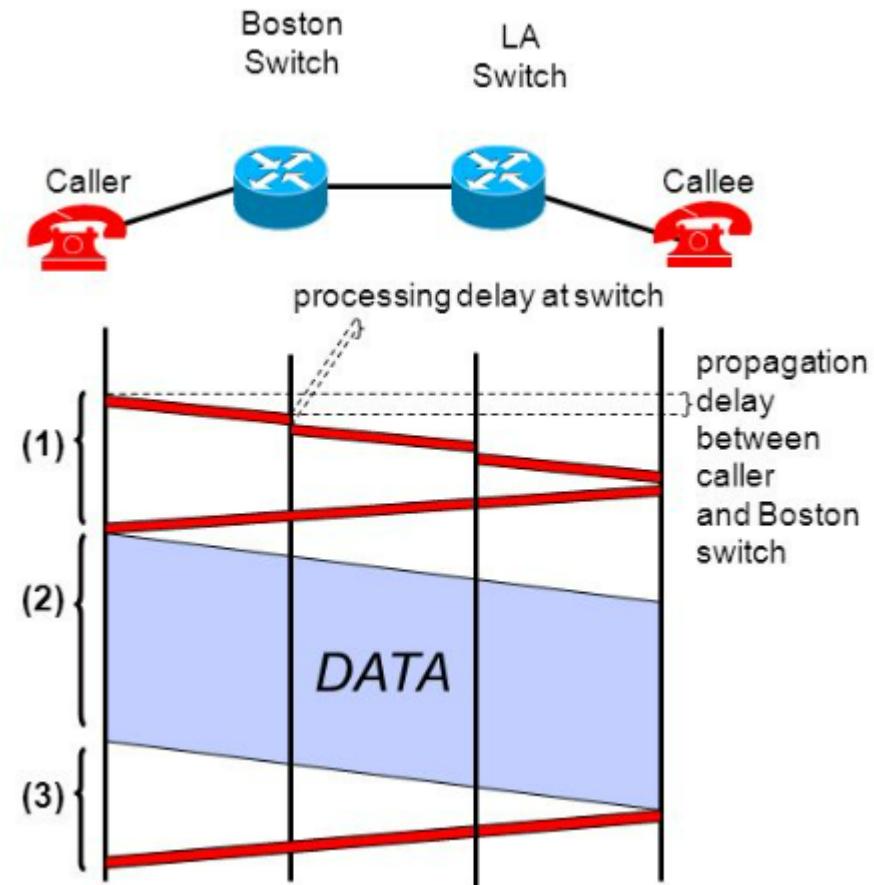
# Circuit switching – a quick view



- *Circuit switching establishes a dedicated channel or circuit before users can speak to each other on a call.*
- *A channel used in circuit switching is kept reserved at all times and is used once the two users communicate*

# Circuit switching: connection-oriented networks

- Circuit switching is **connection-oriented**
- Three phases
  - ① Establish circuit from end-to-end
  - ② Transfer/communicate
  - ③ Disconnect/close circuit

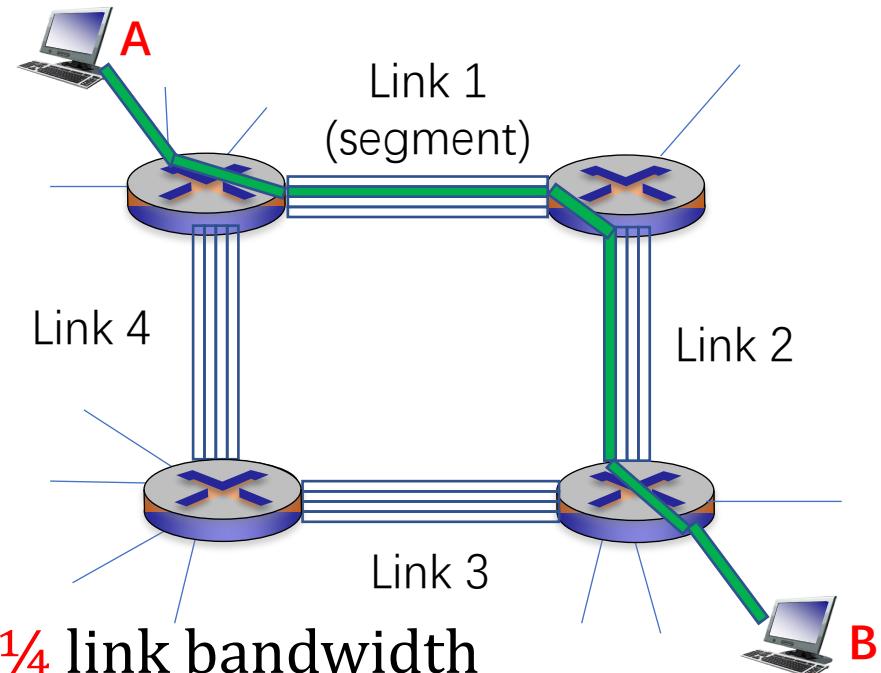


# Circuit switching

- **end-end resources allocated to, reserved for “call” between source & destination:**

Example: each link/segment has four circuits. A wants to send to B via 2<sup>nd</sup> circuit on link-1 & 4<sup>th</sup> circuit on link-2

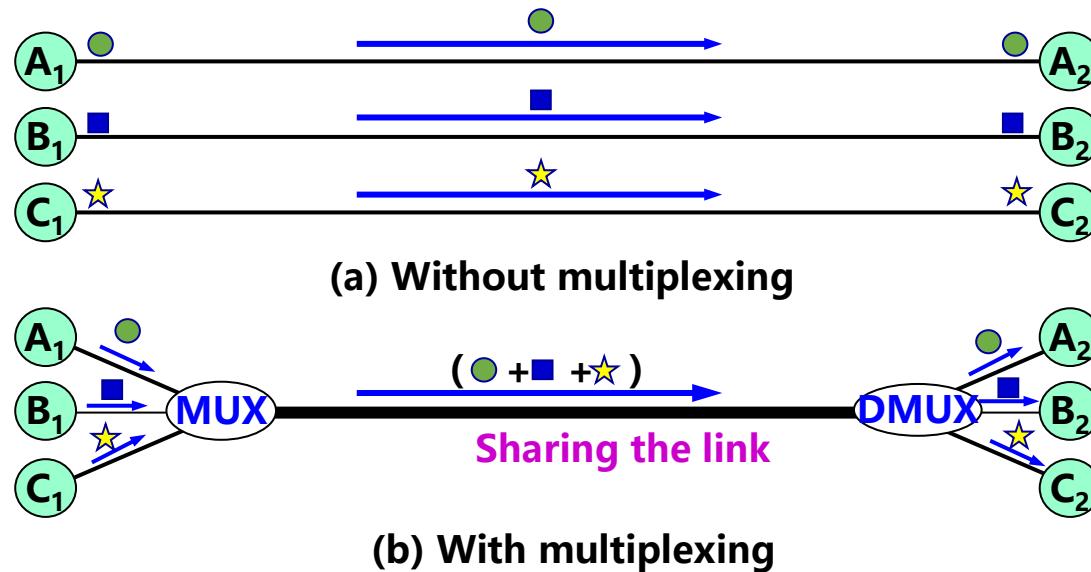
Link: 1Mbps = 1000 kbps  
1/4Mbps = 250kbps



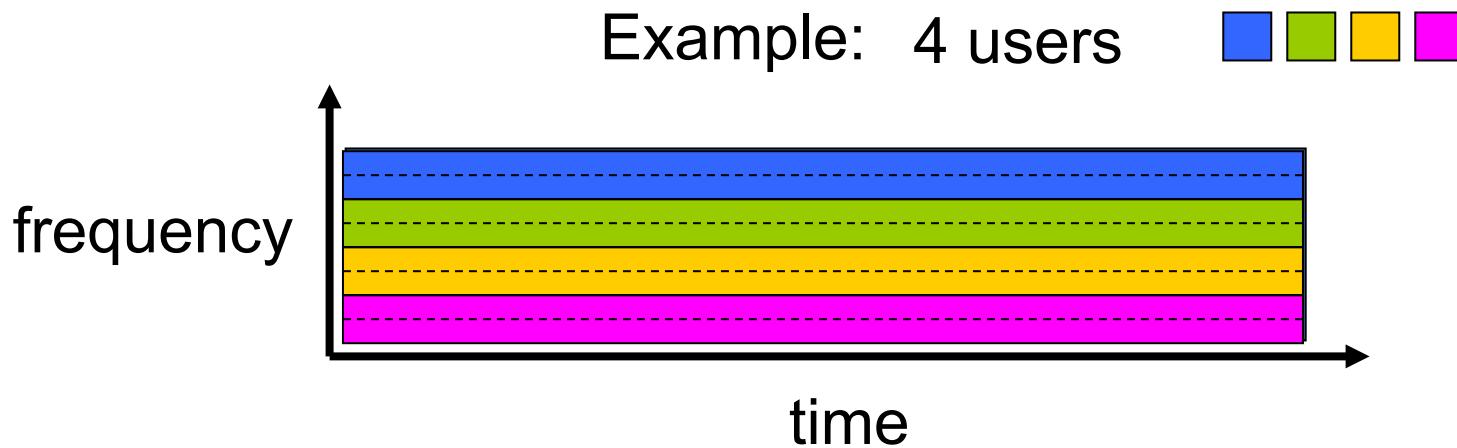
- Each circuit get **250kbps**,  $\frac{1}{4}$  link bandwidth
- circuit segment wasted if not used by call (no sharing)

# How to create multiple circuits on a link?

- **Multiplexing**: a method combines multiple analog or digital signals into one signal over a shared medium.



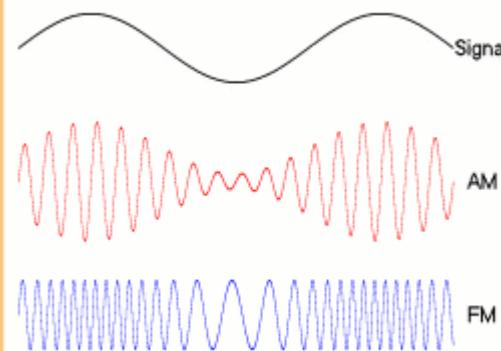
# Circuit switching: FDM



- Frequency Division multiplexing
- The frequency is divided into multi-bands
- Each user carry one frequency band

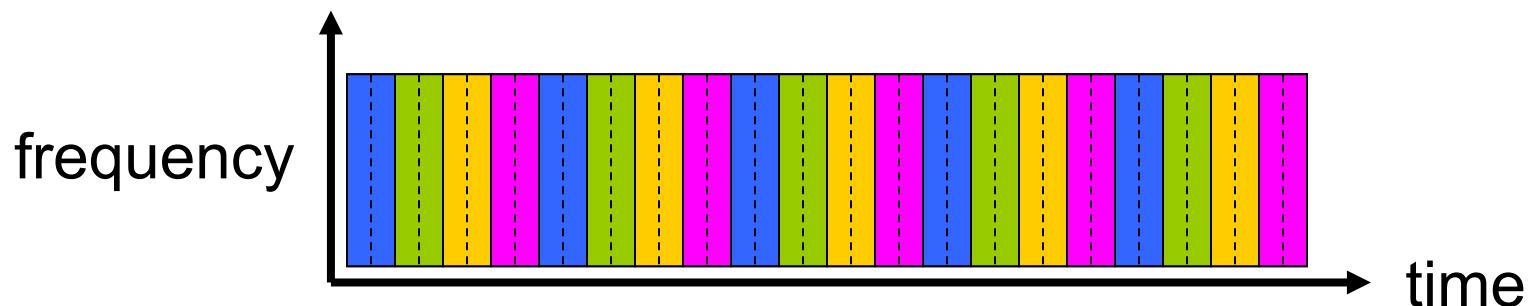
# Applications of FDM

- Television broadcasting
- FM and AM radio broadcasting
- First generation cellular phones



# Circuit switching: TDM

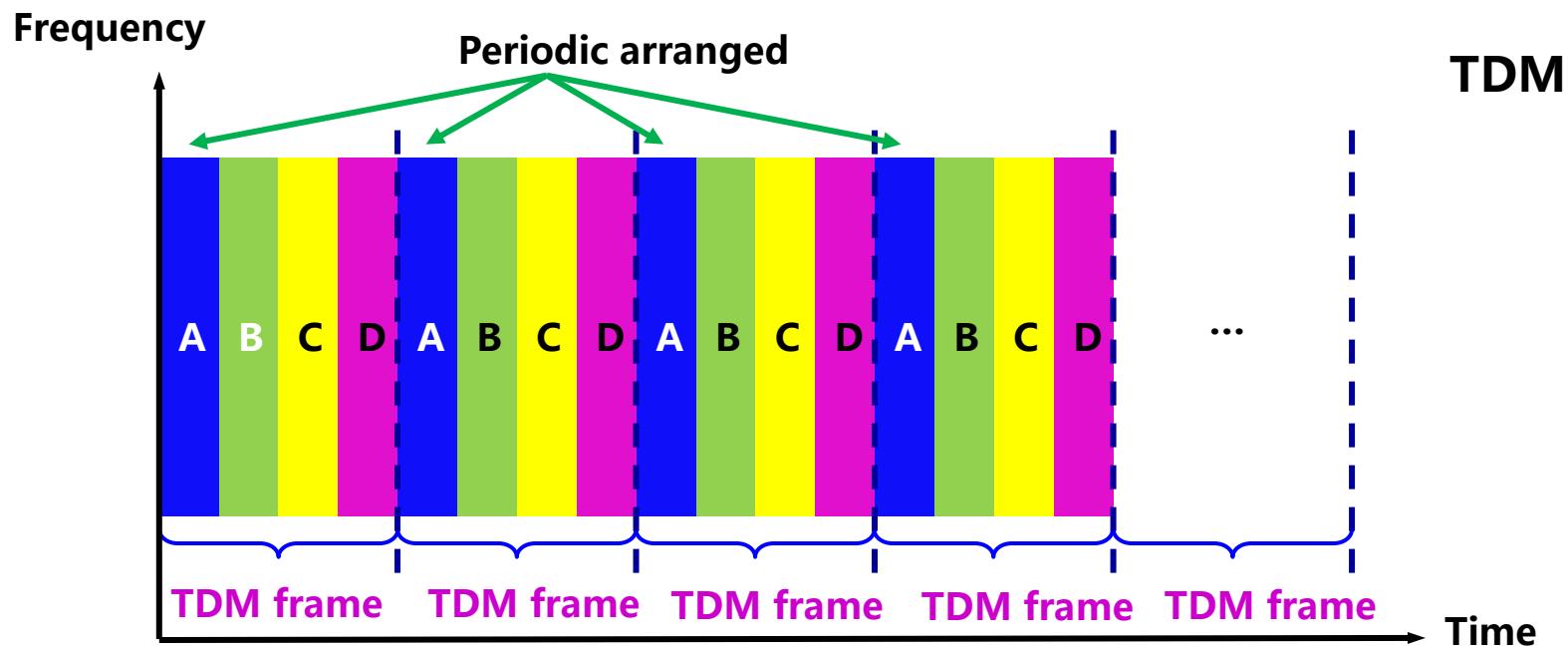
Example: 4 users



All same-color slots are allocated to a specific sender-receiver pair/user

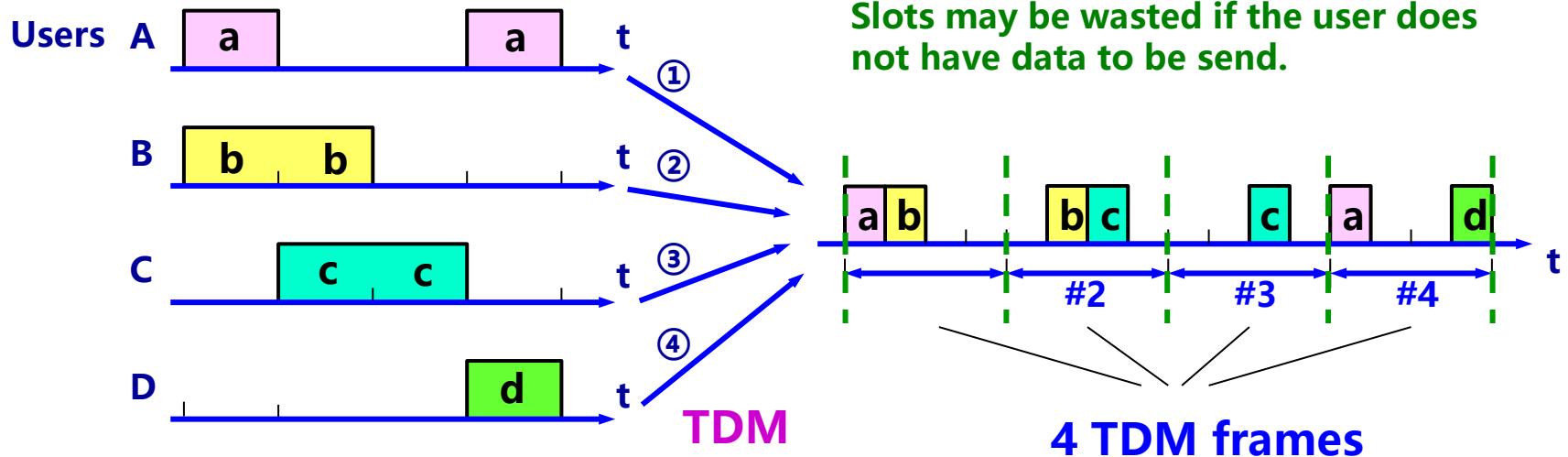
- Time Division Multiplexing
- The time is divided into multiple **frames**.
- Each frame is divided into multiple **time slots**

# TDM frame



- The slots are allocated **in a fixed order** to the different incoming channels.

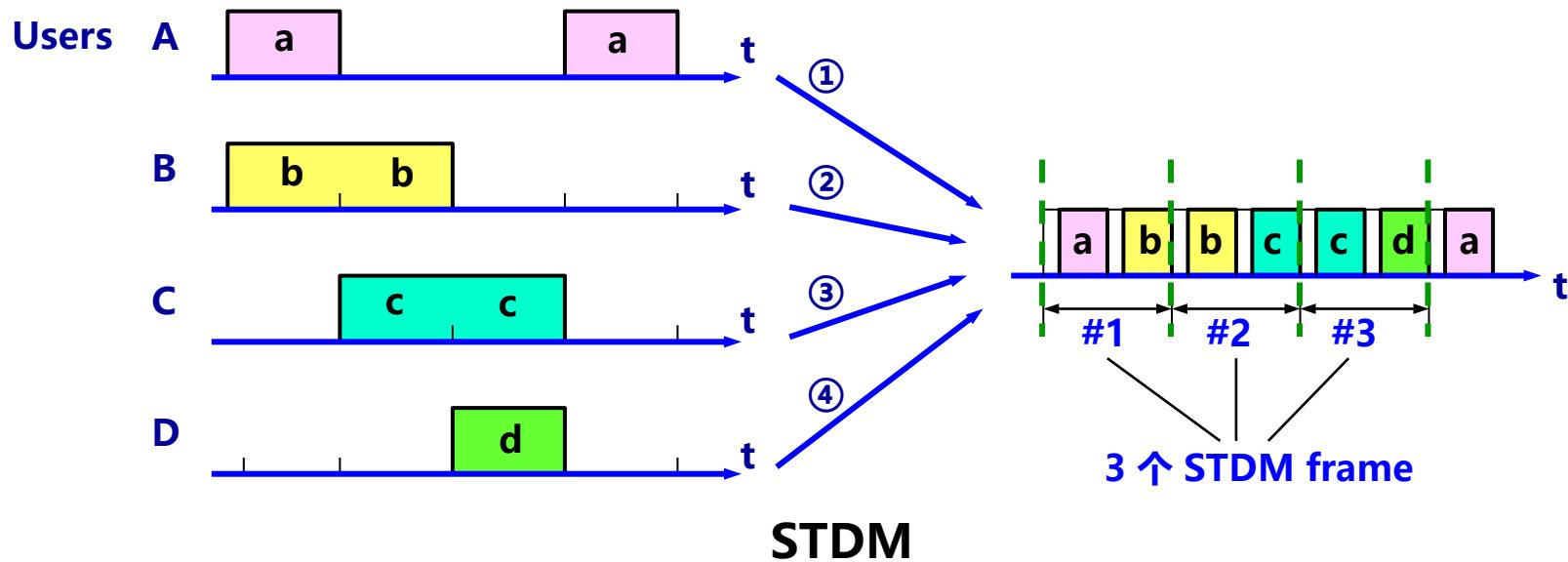
# Problems with TDM



bandwidth wastage in case of e.g.,

- slot reservation for station without any data to transmit
- uneven distribution of traffic

# Statistical TDM: An improvement from TDM



- Designed to make use of the free time slots
- Time slots used as needed
- But more complex to implement

# TAKEAWAYS

- Physical Media
  - Guided media
  - Unguided media
- Circuit Switching
  - Basic principles
  - Multiplexing



西北工業大學  
NORTHWESTERN POLYTECHNICAL UNIVERSITY



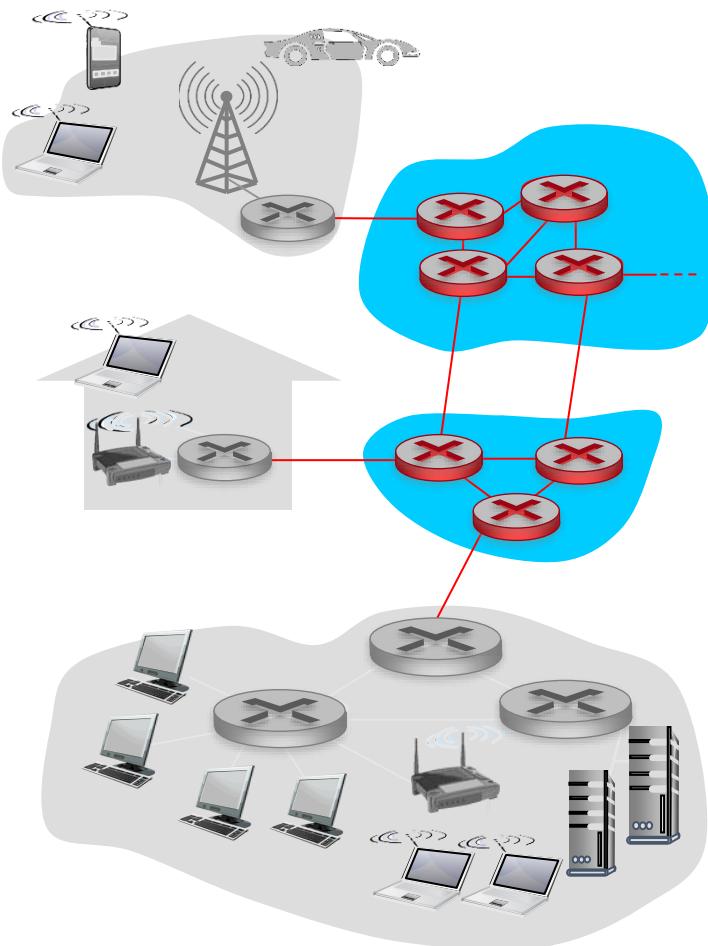
# Computer Networks

Lecturer: ZHANG Ying

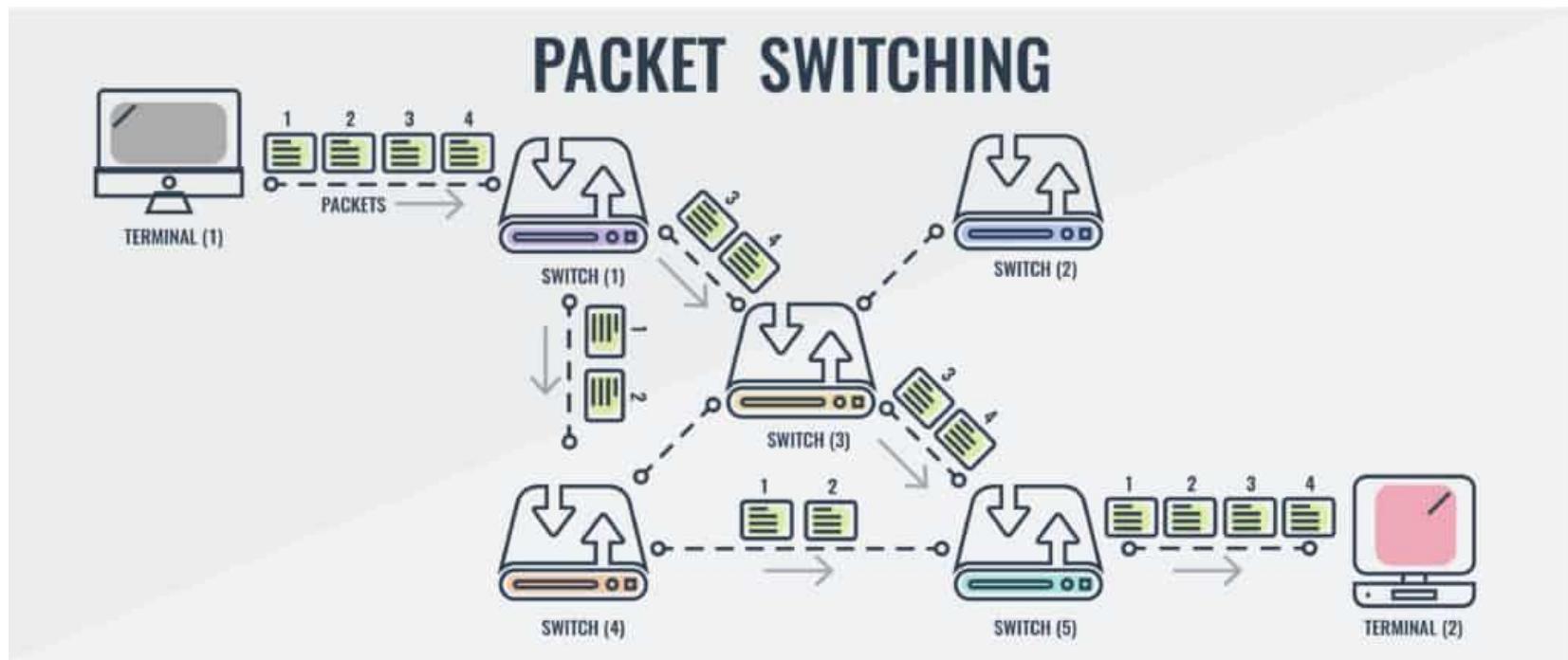
Fall semester 2022

# Network core

- Network core/ backbone network is the mesh of routers that interconnect the Internet's end-systems
- Build the core via:
  - Circuit switching
  - **Packet switching**
  - (message switching)



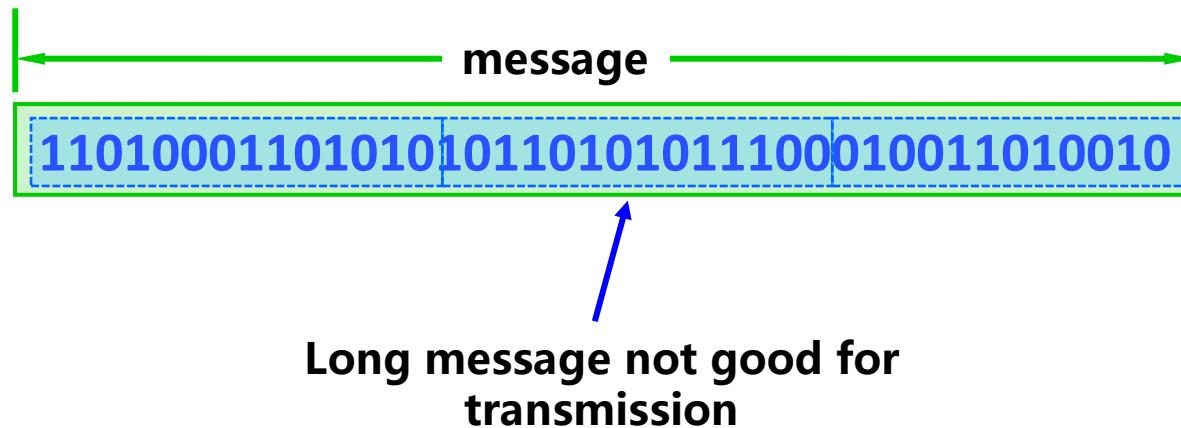
# Packet switching – a quick view



- *Packet switching not requires to establish a channel. The channel is available to users throughout the data network.*
- *Long messages are broken down into packets and sent individually to the network*

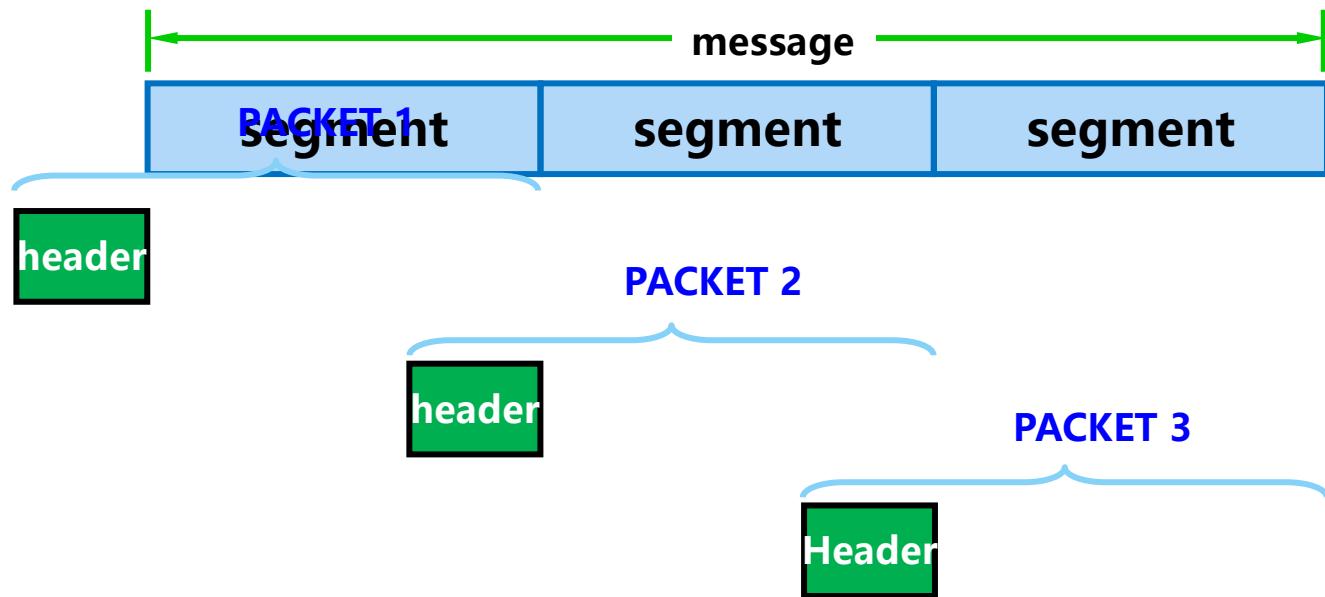
# Packet – the basic unit

- Sender: First divide the longer message into shorter, fixed-length data segments



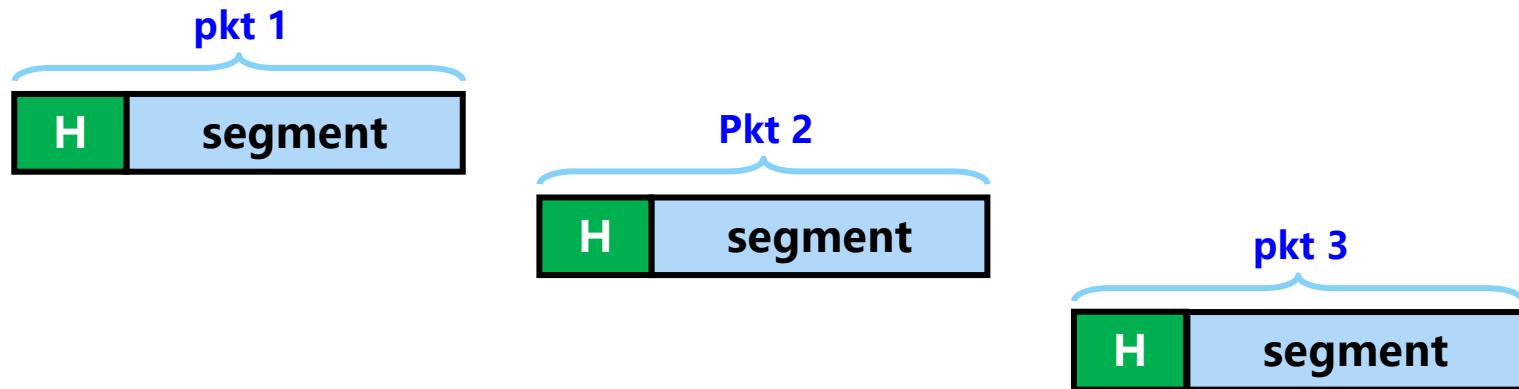
# Packet – sender side

- Add a header to the front of each data segment to form a packet.



# Packet – sender side

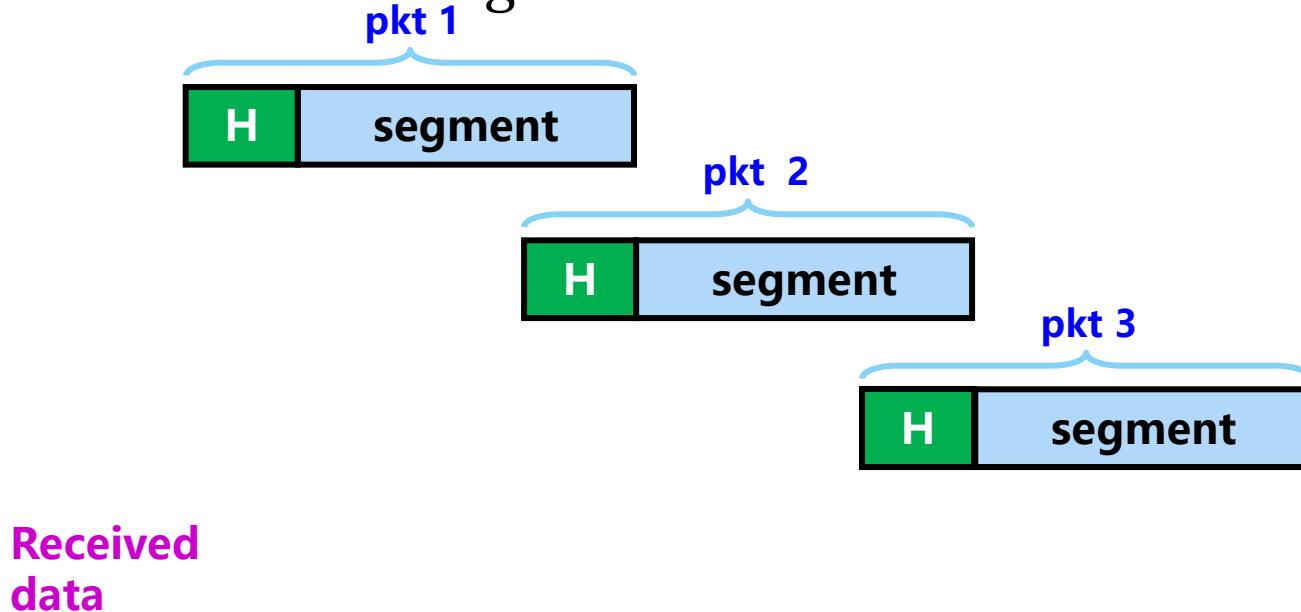
- "packets" : basic data transmission unit.
- Each packet is sent to the receiver in turn



(assuming the receiver is on the left).

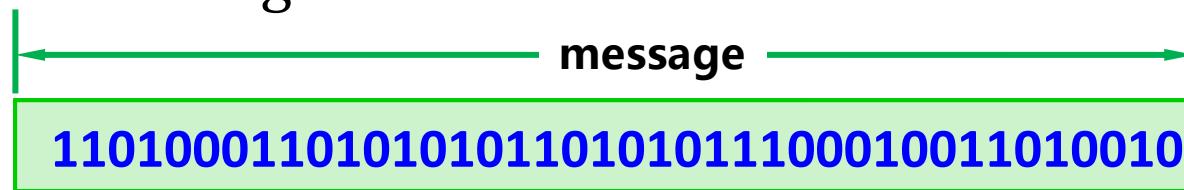
# Packet – receiver side

After receiving the packet, the receiving end strips the header and restores it to a message.



# Packet – the basic unit

- At the receiving end, the received data is restored to the original message



- Here we assume that the packets are transmitted without errors and are not discarded during forwarding.

# Circuit Switching

*resource reservation*



Restaurant A

*accepts reservation*

# Packet Switching

*no resource reservation*

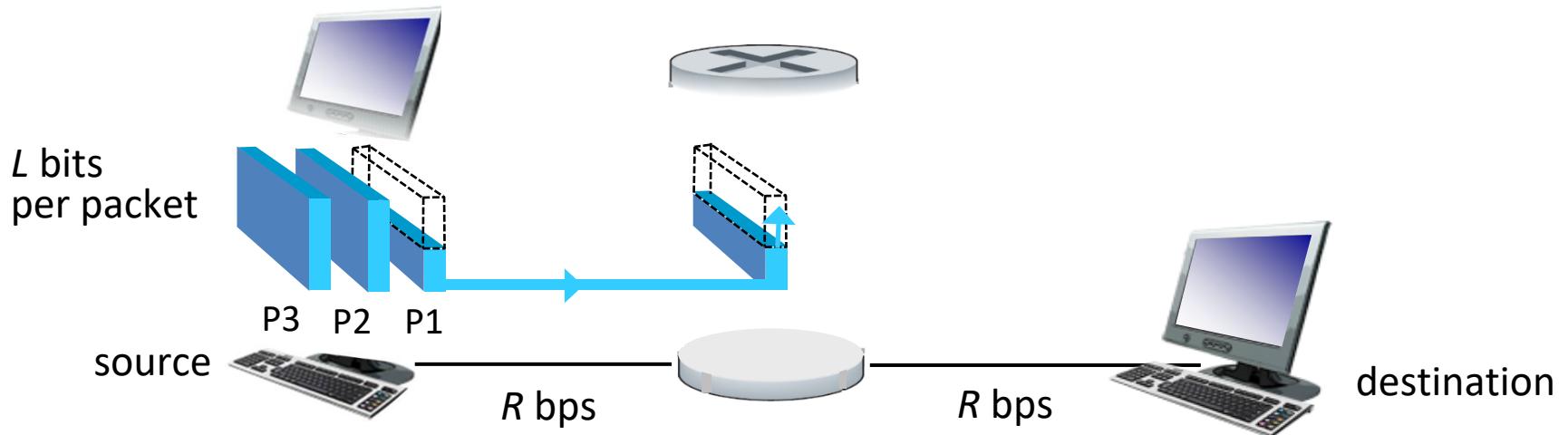


Restaurant B

*no reservation*

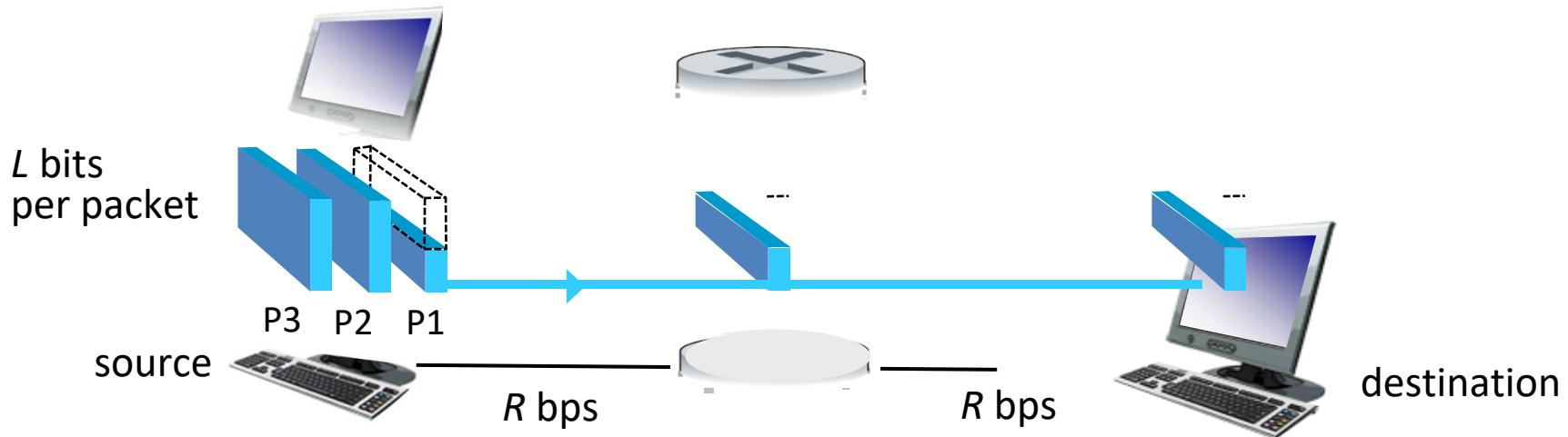


# Store and forward technique



- Each packet takes  $L/R$  seconds to transmit (push out)  $L$ -bit packet into link at  $R$  bps
- *store and forward*: entire packet must arrive at router before it can be transmitted on next link
- end-end delay =  $2L/R$  (assume zero prop-delay)

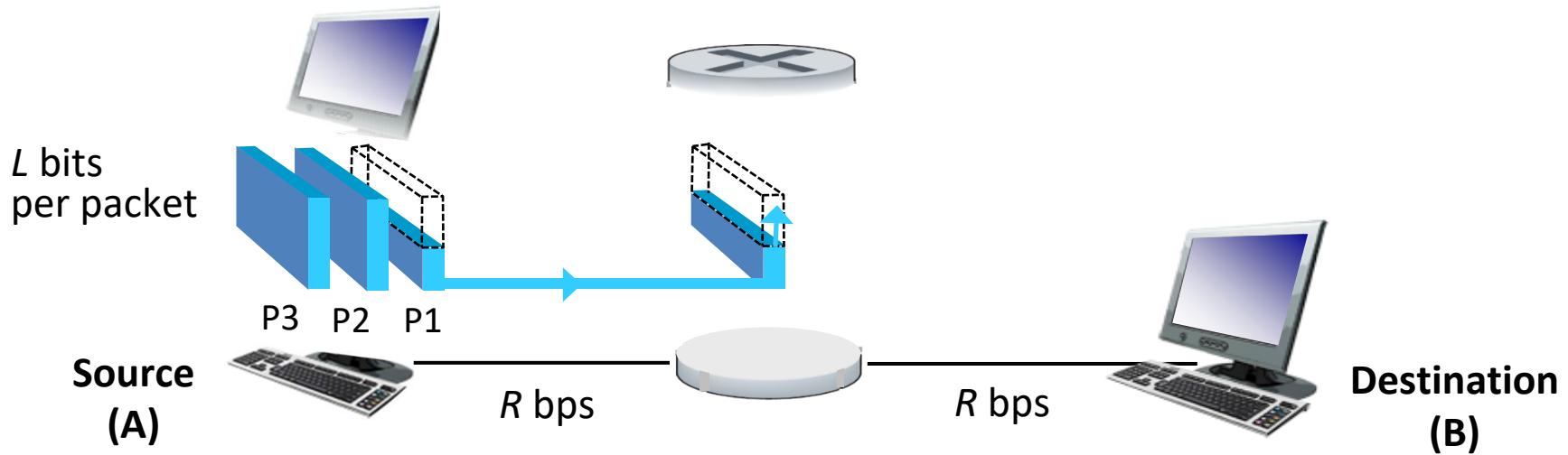
# Scenario 1: What if **not** using store and forward?



- *The router push out each bit **immediately after its arrival***
- end-end delay =  $L/R$
- as one hop

more on delay shortly ...

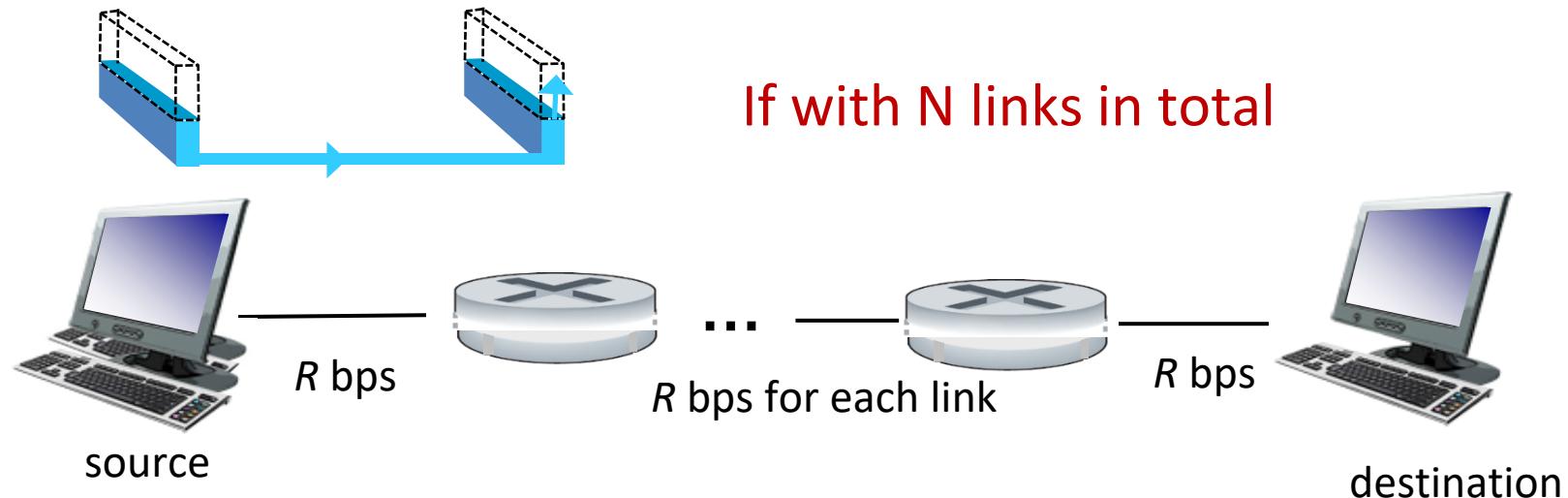
## Scenario 2: Store and forward technique with 3 pkts



- End-end delay per packet:  $2L/R$  seconds
- $2L/R$  seconds: 1<sup>st</sup> *pkt* @ B & 2<sup>nd</sup> *pkt* @ router
- $3L/R$  seconds: 2<sup>nd</sup> *pkt* @ B, 3<sup>rd</sup> *pkt* @ router
- $4L/R$  seconds: 3<sup>rd</sup> *pkt* @ B

# Scenario 3: A more general case with N links for 1pkt:

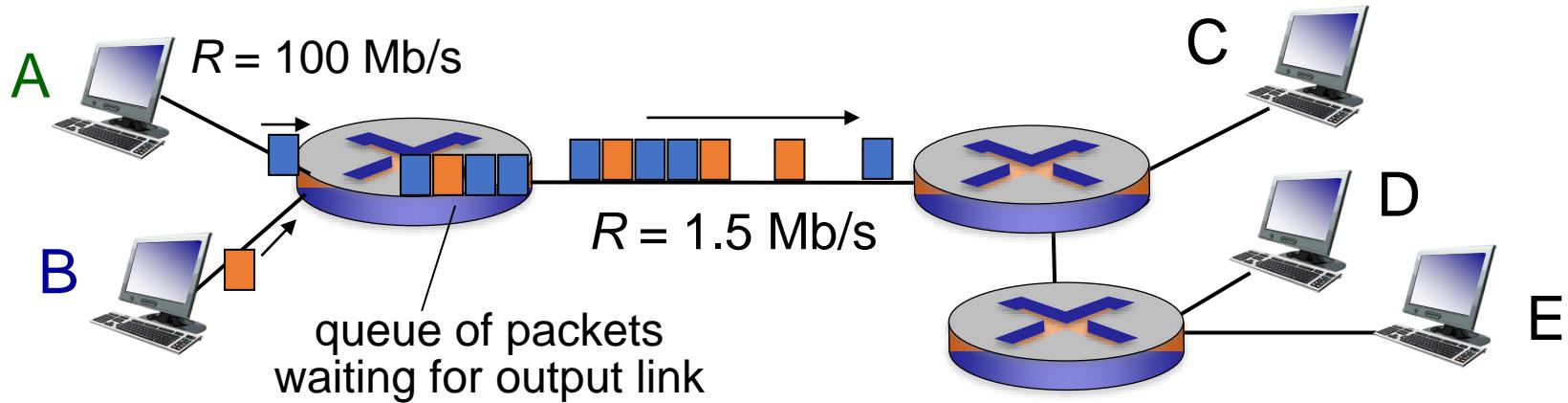
One  $L$  bits packet



- By store and forwarding technique:

$$\text{End-end delay} = N \times L / R \text{ (sec)}$$

# Packet loss



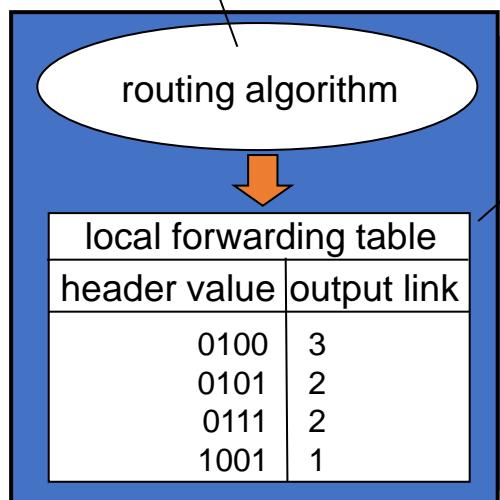
if arrival rate (in bits) to link **exceeds** transmission rate of link for a period of time:

- packets will **queue**, wait to be transmitted on link
- packets can be **dropped** (lost) if memory (buffer) fills up

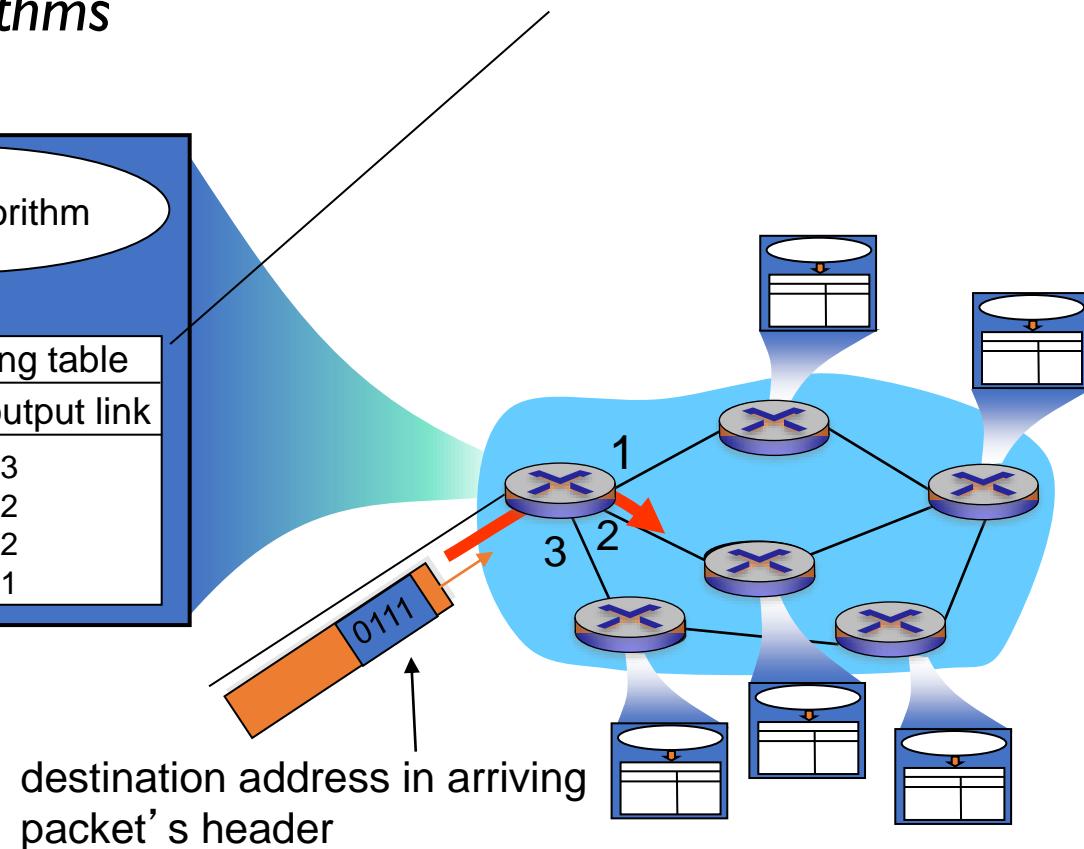
# Two key functions

***routing***: determines source-destination route taken by packets

- *routing algorithms*



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



# Router

Router processes the packet:

- Put the received packets into the cache first (temporarily store);
- Look up the forwarding table to find out which port should be forwarded to a destination address;
- Forward the packet to the appropriate port.

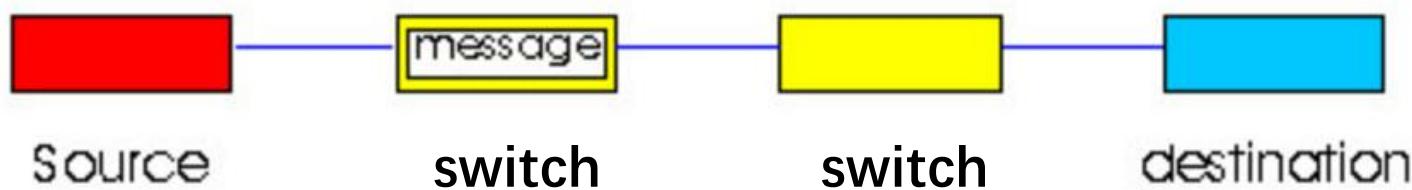
# Router is different from Host

- The **host** computer processes information for the user and sends and receives packets to and from the network.
- The **router** stores and forwards the packet, and finally delivers the packet to the destination host.

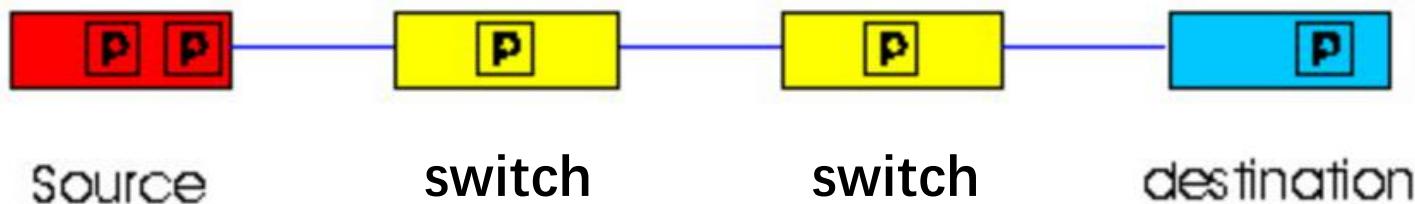
# A comparison btw Circuit & Packet switching

| Circuit switching                          | Packet switching                           |
|--|--|
| Dedicated transmission path                | No dedicated path                          |
| Continuous transmission of data            | Transmission of packets                    |
| Reserves the required bandwidth in advance | Acquires and releases bandwidth as needed. |

# message switching vs. packet switching

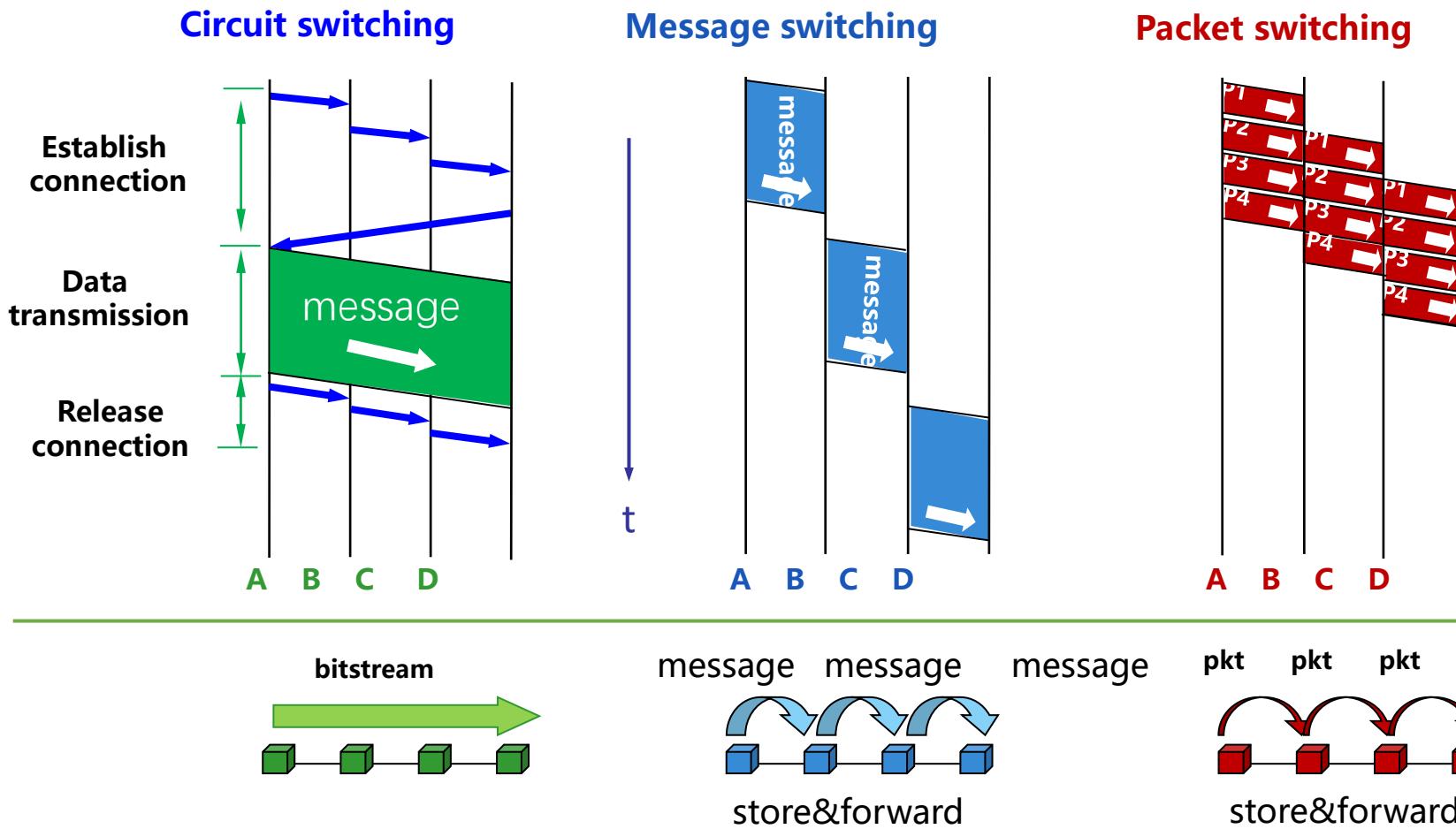


(a) message switching



(b) packet switching

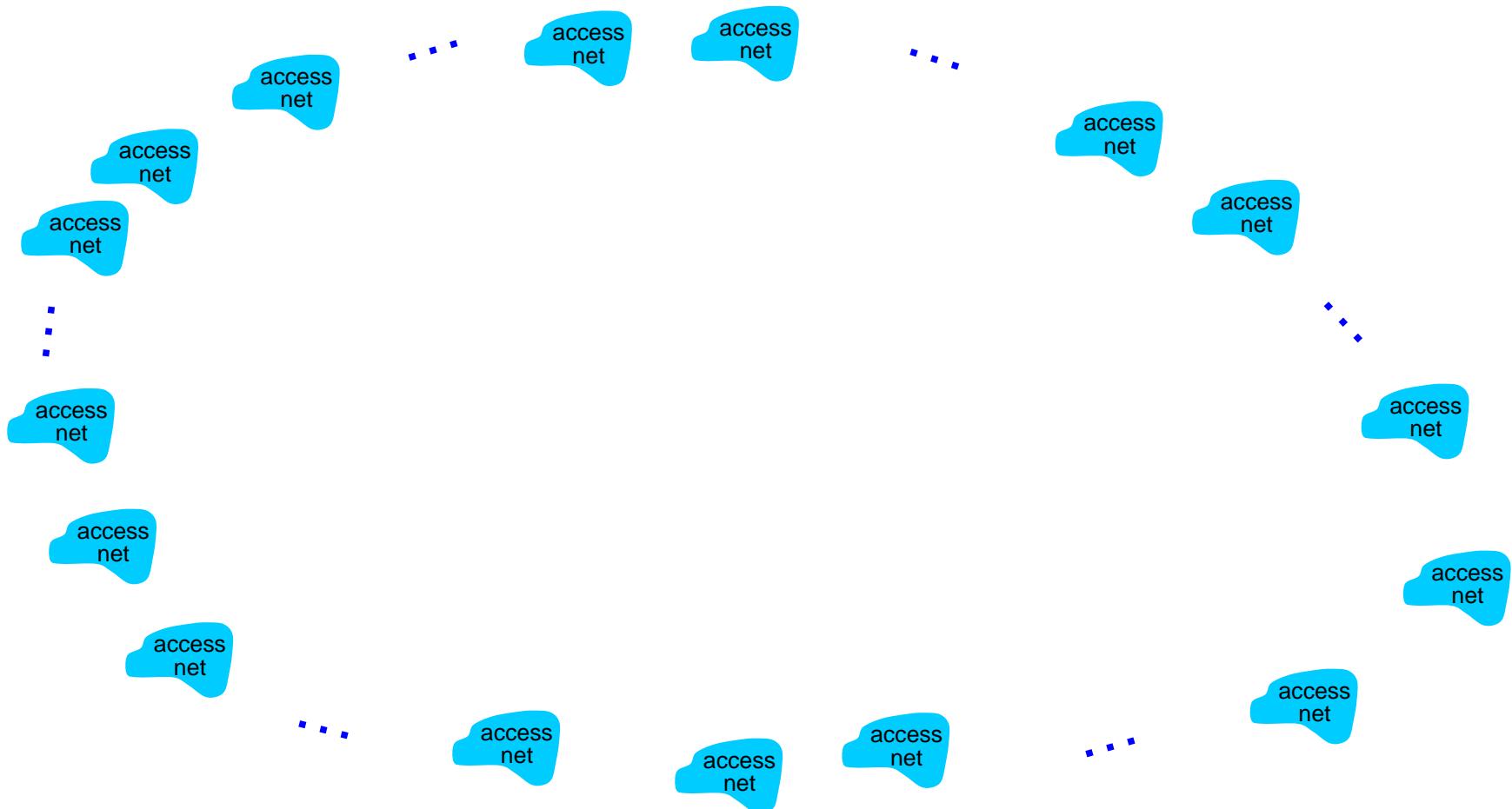
# Data transmission difference



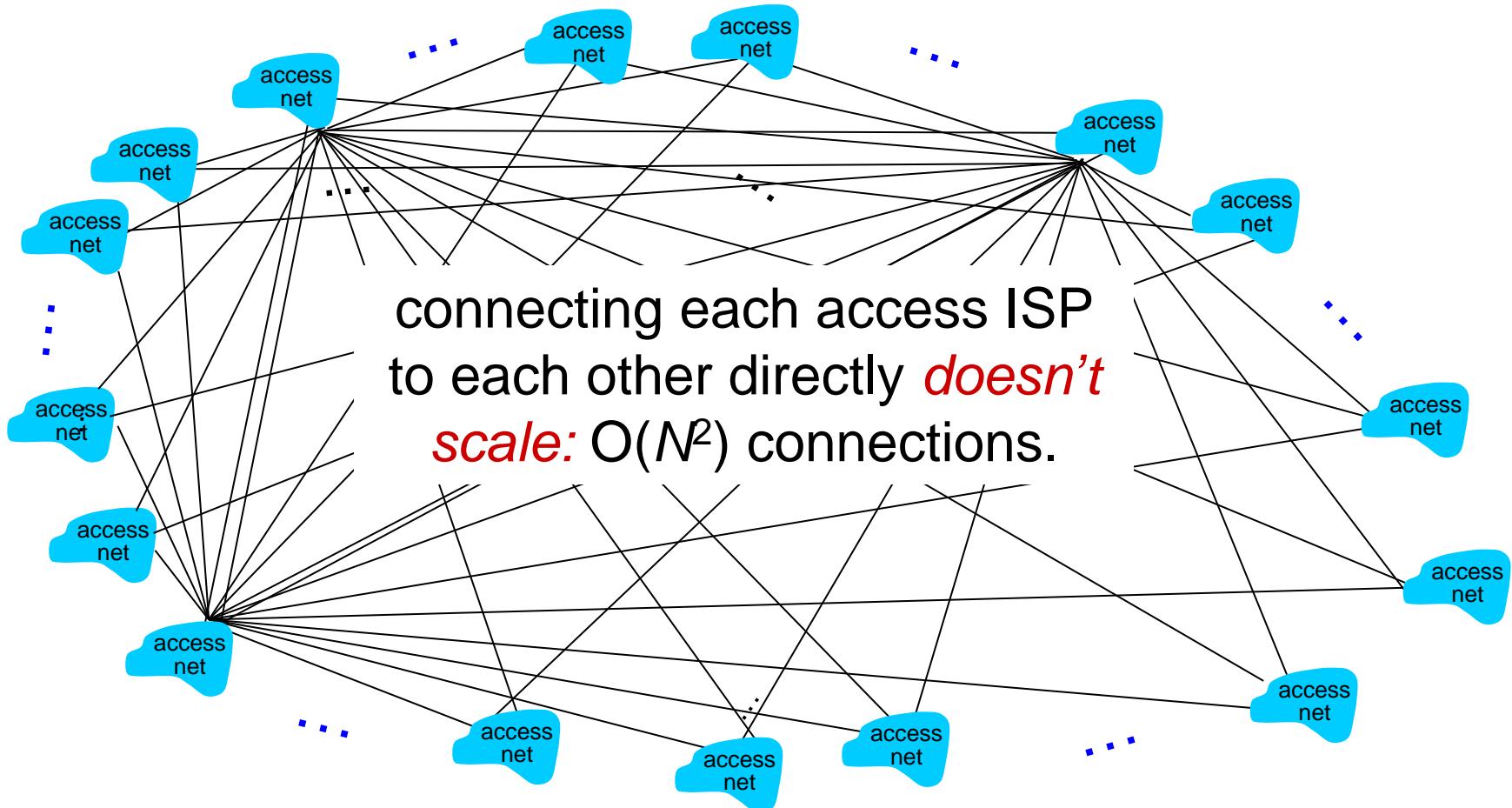
# Internet structure: network of networks

- 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**
- Let's take a stepwise approach to describe current Internet structure

*Question:* given *millions* of access ISPs, how to connect them together?

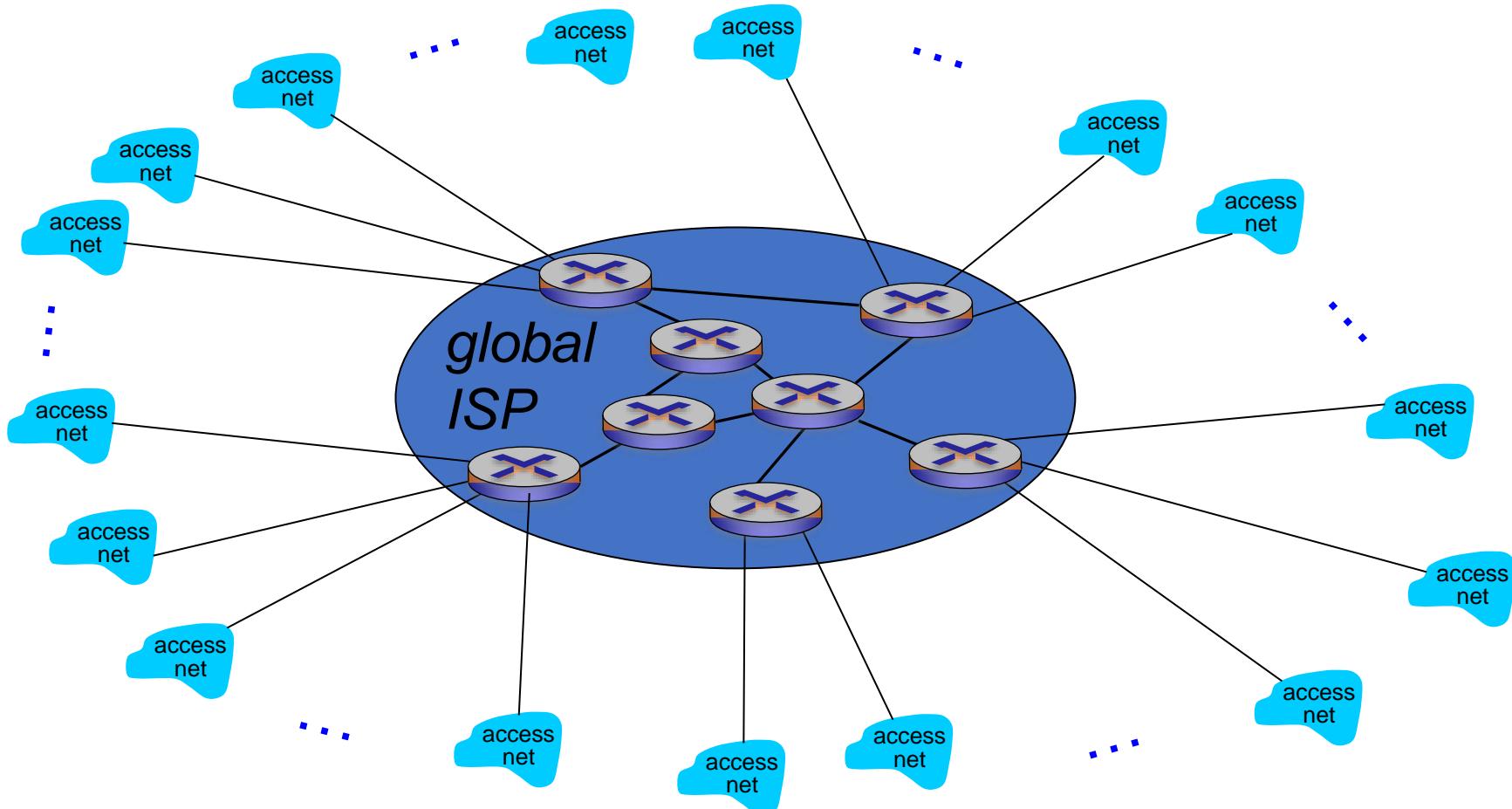


*Option1: connect each access ISP to every other access ISP?*

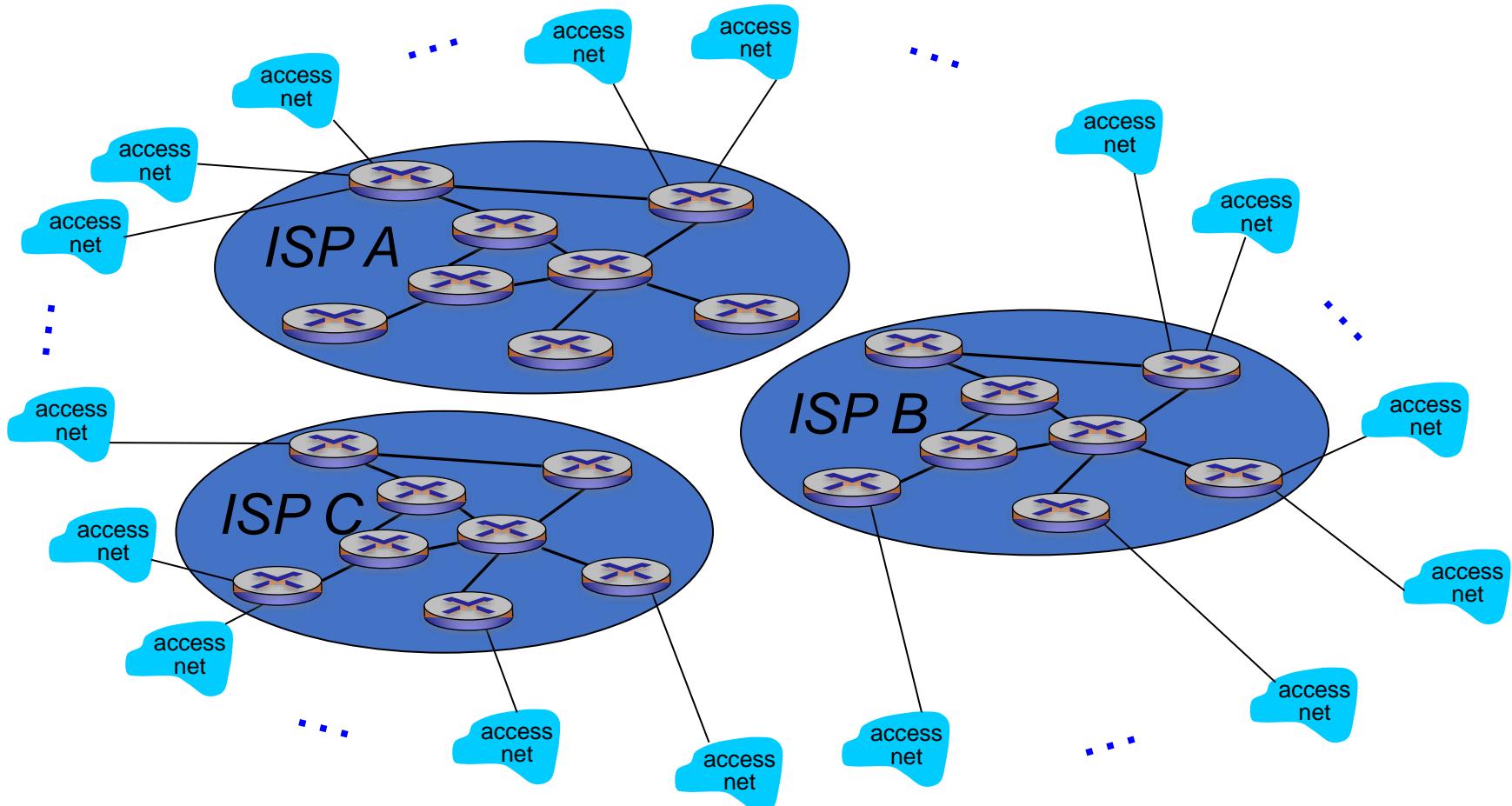


*Option2: connect each access ISP to one global transit ISP?*

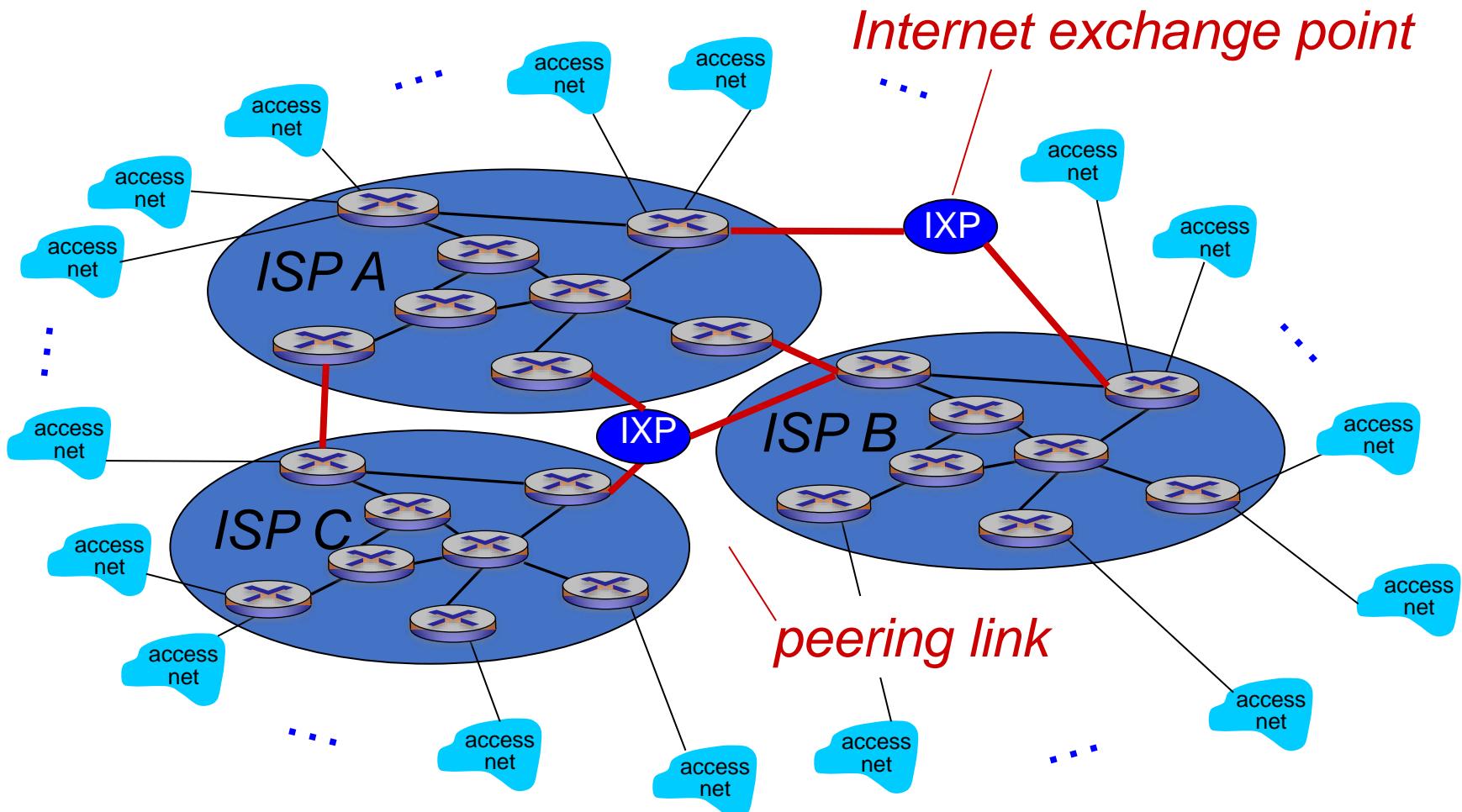
*Customer and provider ISPs have economic agreement.*



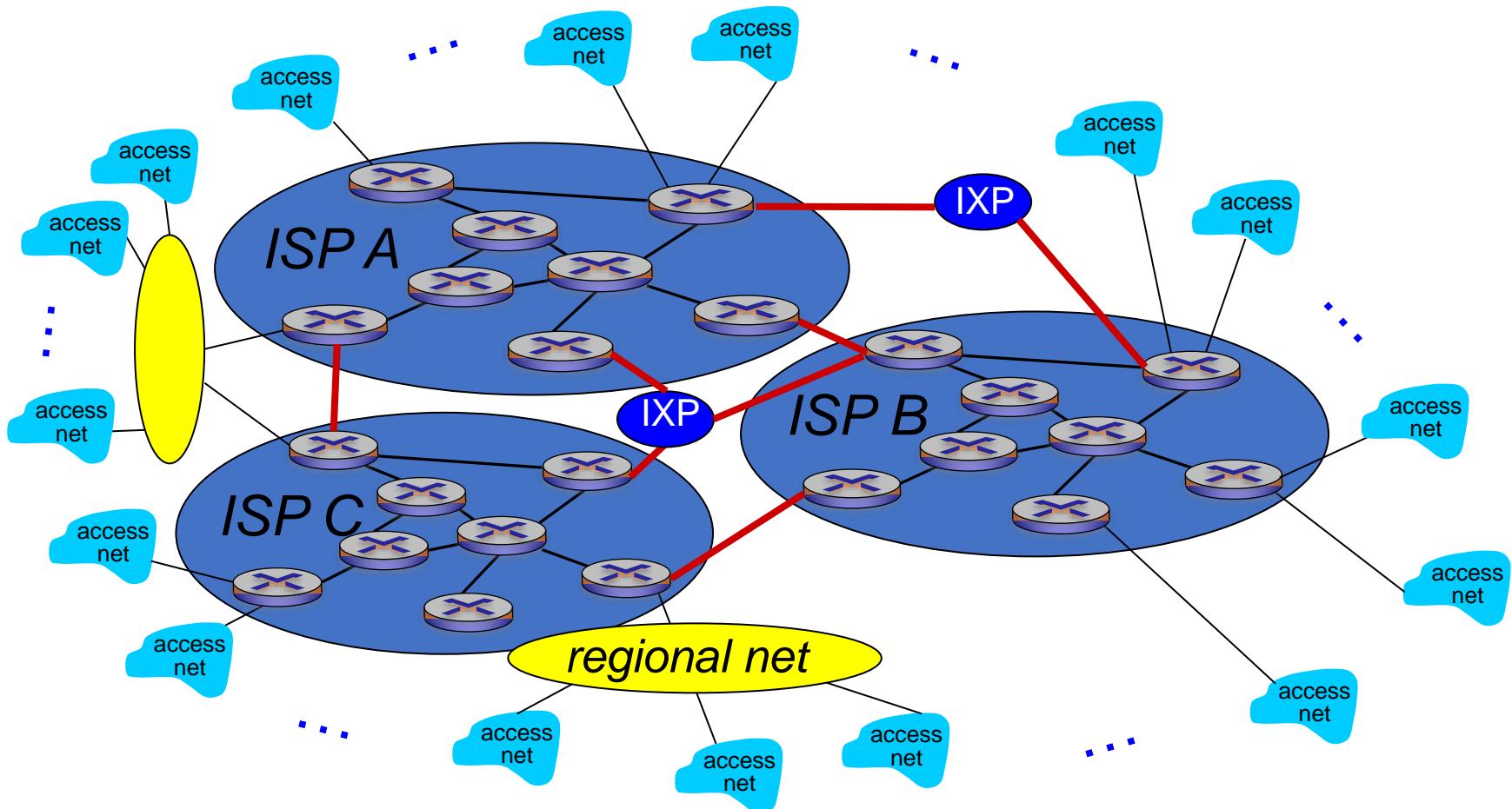
But if one global ISP is viable business, there will be competitors .... **Multiple ISPs**



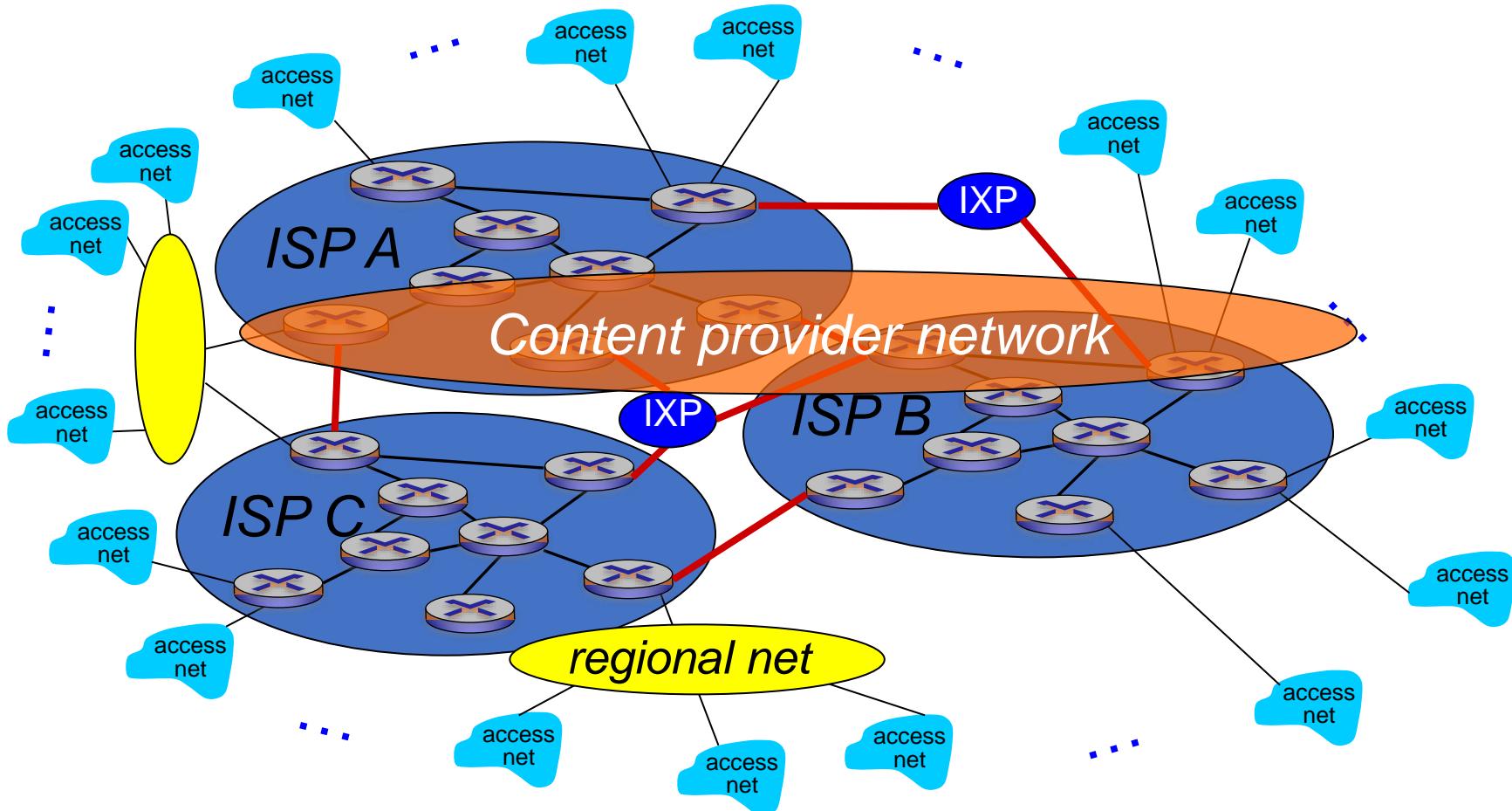
But if one global ISP is viable business, there will be competitors .... **which must be interconnected**



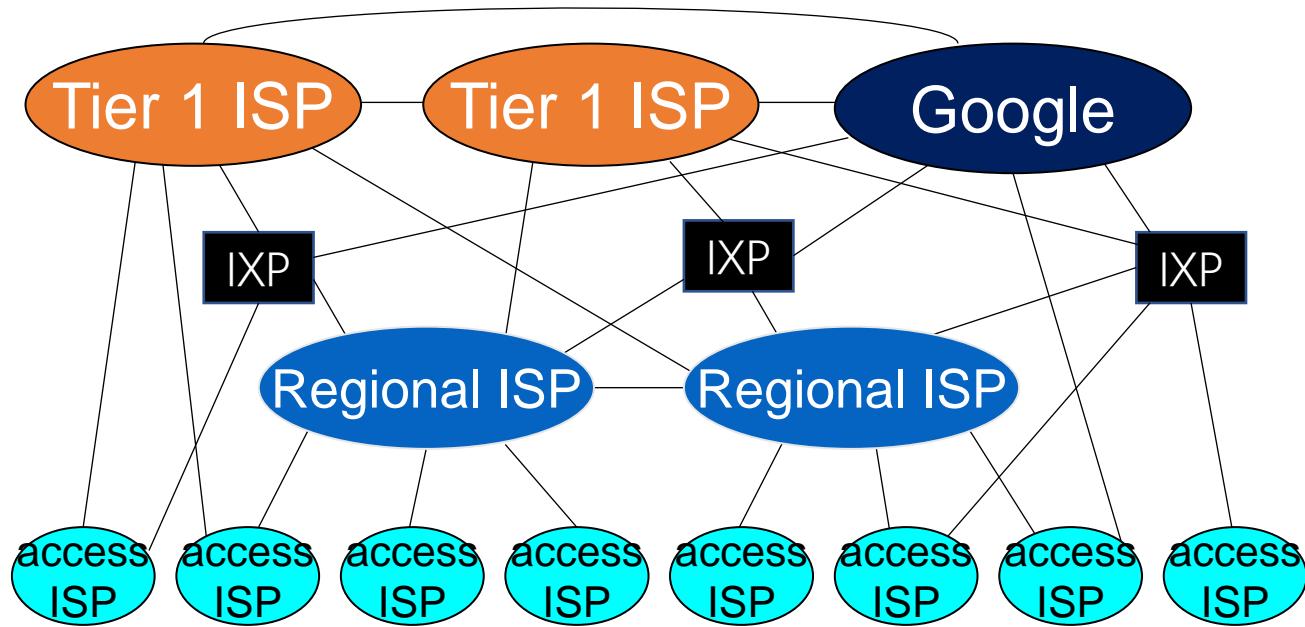
... and regional networks may arise to connect access nets to ISPs



... 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 its data centers to Internet, often bypassing tier-1, regional ISPs

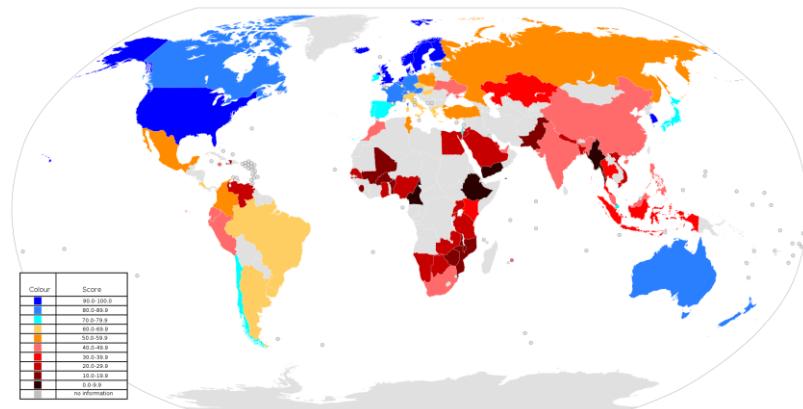
# Internet or internet?

| Internet                                   | internet                                |
|--|---|
| Same                                       |   |
| Network of networks                        | Network of networks                     |
| Difference                                 |   |
| Follow TCP/IP protocols<br>Global coverage | a network of multiple computer networks |
| TCP/IP                                     | TCP/IP or others                        |
| A dedicated term                           | A more general term                     |

Arbitrarily interconnecting several computer networks (no matter what protocol is used) and being able to communicate with each other, this constitutes an Internet, rather than the Internet.

# World Wide Web & Internet

- The Internet has become the largest and fastest-growing computer network in the world
- The World Wide Web (World Wide Web) has become the main driving force for this exponential growth of the Internet



A global map of the Web Index for countries in 2014



historic World  
Wide Web logo

# Outline

1

what is the Internet

2

network edge

3

network core

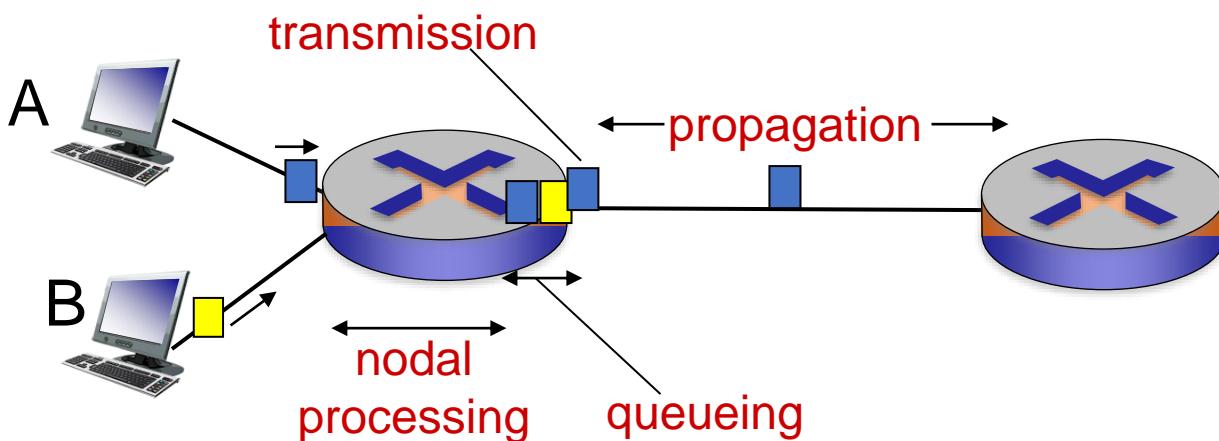
4

delay, loss, throughput in networks

5

protocol layers, service models

# Four sources of packet delay



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

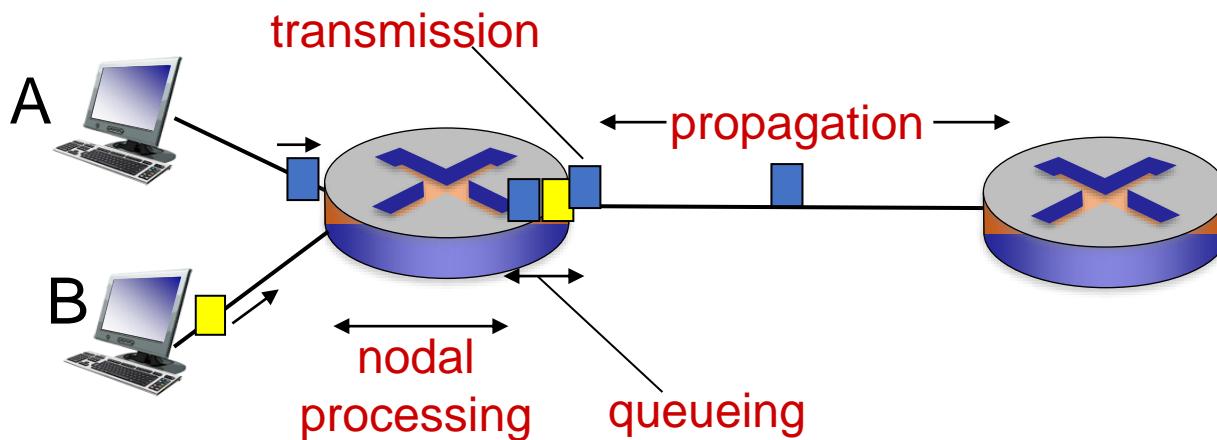
$d_{\text{proc}}$ : nodal processing

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

$d_{\text{queue}}$ : queueing delay

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

# Four sources of packet delay



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

$d_{\text{trans}}$ : transmission delay:

- $L$ : packet length (bits)
- $R$ : link bandwidth ( $b\text{ps}$ )
- $d_{\text{trans}} = L/R$

$d_{\text{trans}}$  and  $d_{\text{prop}}$   
very different

$d_{\text{prop}}$ : propagation delay:

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

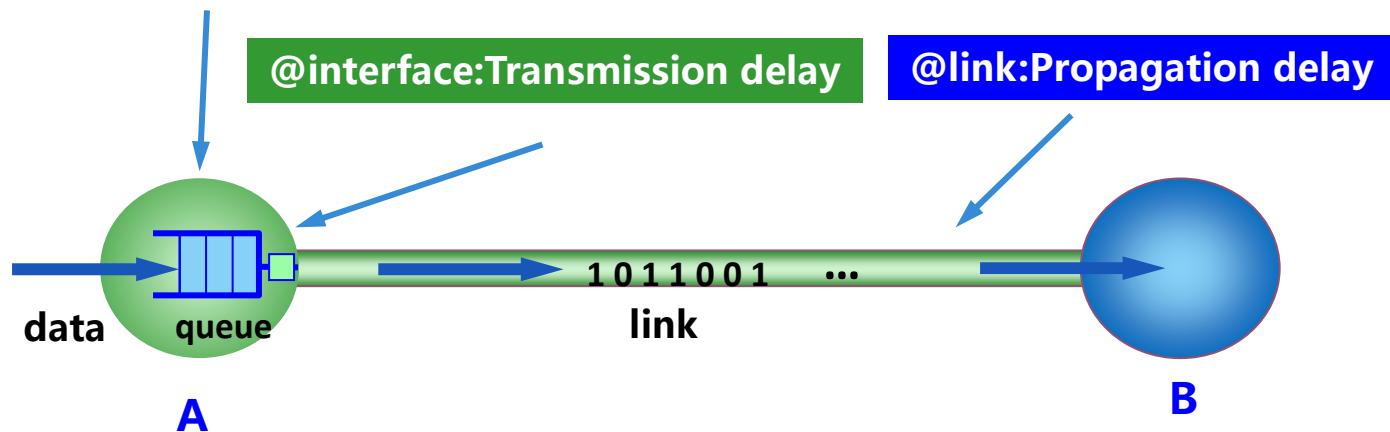
# *Transmission vs Propagation*

- Very different
- Transmission: time to push the pkts to next link
  - **NOT** Related to the physical distance
- Propagation: carry the bit via physical media to reach destination.
  - Related to the physical distance

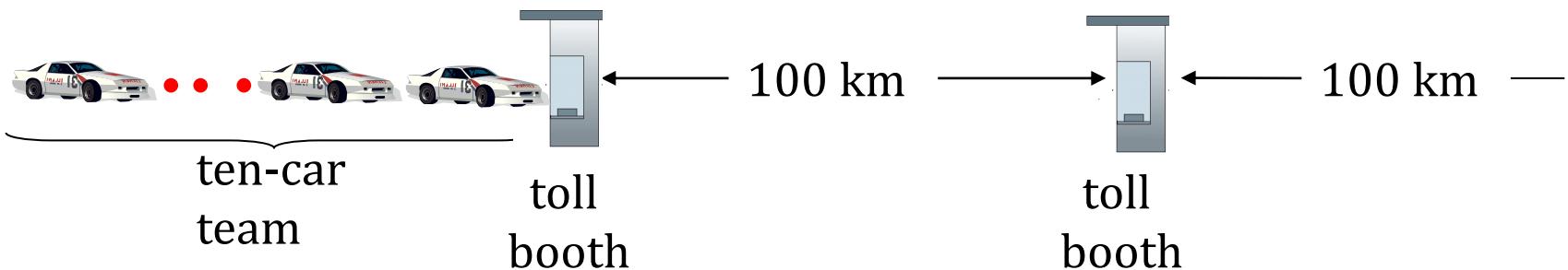
# Where the delays occurs?

**Node A sends data to Node B**

**@A: process delay & queue delay**

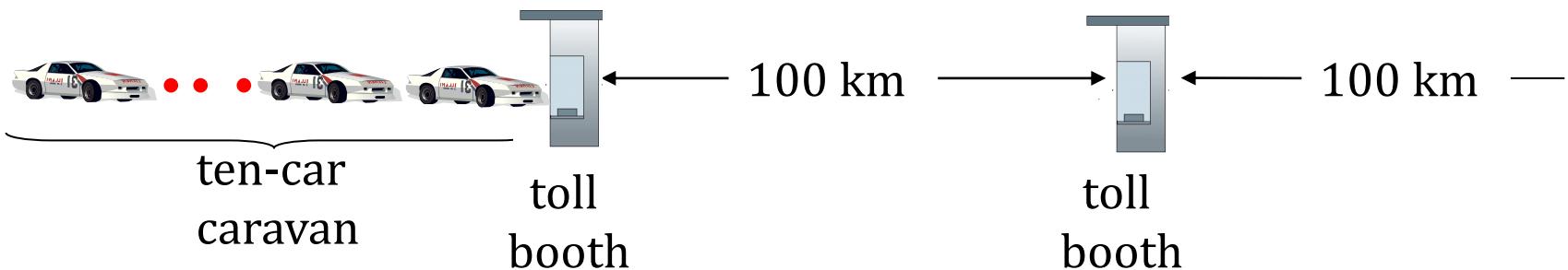


# An analogy - 1



- cars “propagate” at 100 km/hr
- toll booth takes 12 sec to service car (bit transmission time)
- car ~ bit; team ~ packet
- *Q: How long until team is lined up before 2nd toll booth?*
- time to “push” entire team through toll booth onto highway  
 $= 12 * 10 = 120 \text{ sec}$
- time for last car to propagate from 1st to 2nd toll both:  
 $100\text{km}/(100\text{km/hr}) = 1 \text{ hr}$
- *A: 62 minutes*

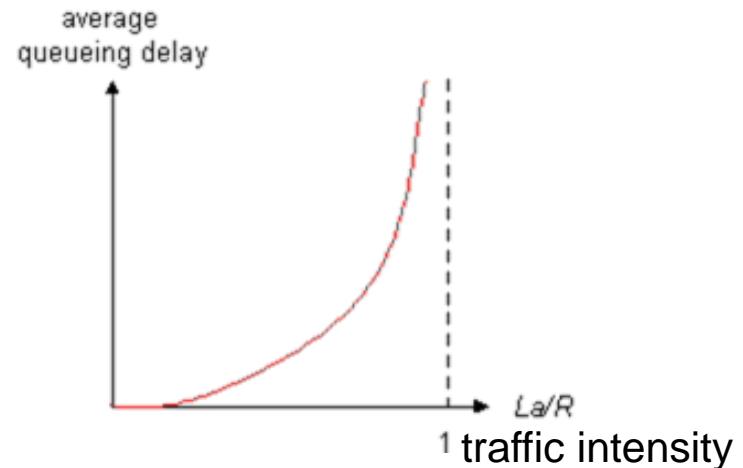
## An analogy - 2



- suppose cars now “propagate” at 1000 km/hr
- suppose toll booth takes 1 min to serve a car
- **Q:** Will cars arrive to 2nd booth before all cars serviced at first booth?
- **A:** Yes! after 7 min, first car arrives at second booth; three cars still at first booth

# Queueing delay and traffic intensity

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



- $La/R \sim 0$ : avg. queue delay small
- $La/R \sim 1$ : avg. queue delay large
- $La/R > 1$ : more “work” arriving than can be serviced
  - average delay infinite

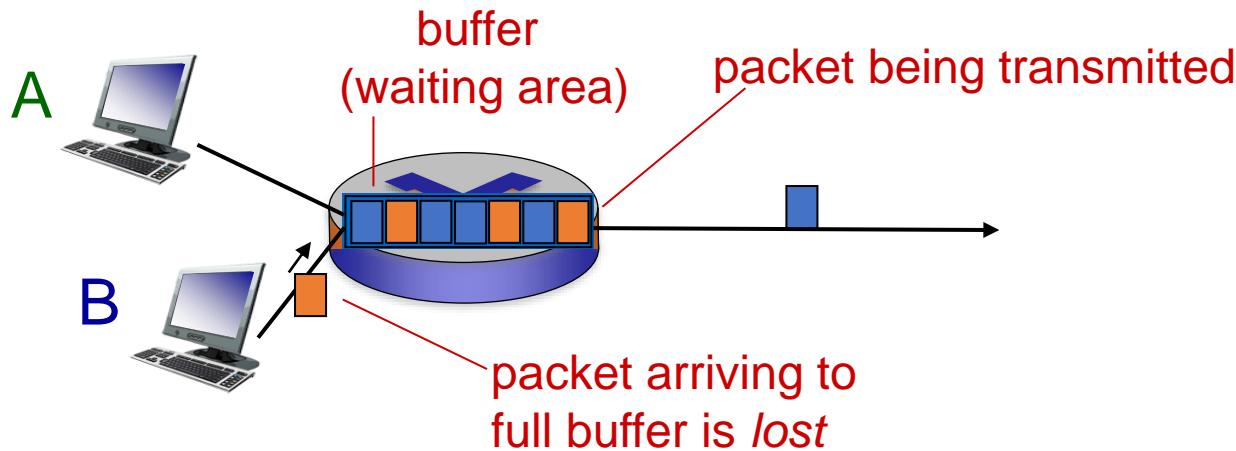


$La/R \sim 0$



$La/R \rightarrow 1$

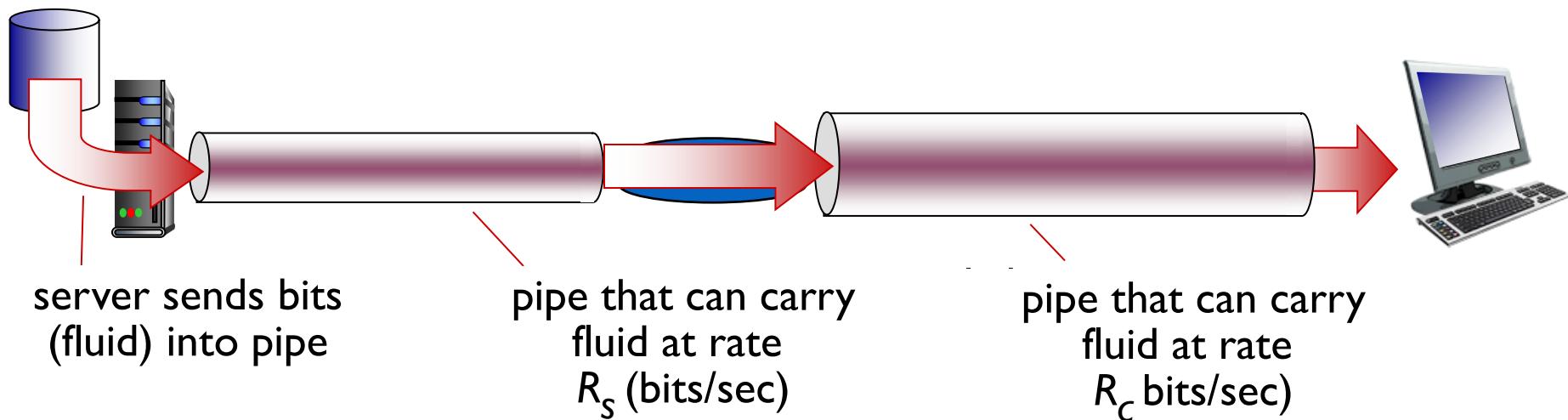
# Packet loss



- queue (aka buffer) preceding link in buffer has finite capacity
- packet arriving to full queue dropped (aka *lost*)
- lost packet may be retransmitted by previous node, by source end system, or not at all

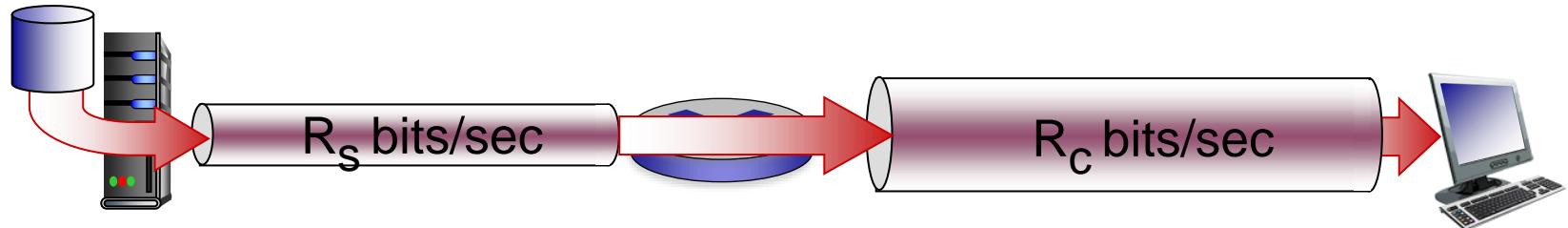
# Throughput

- *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



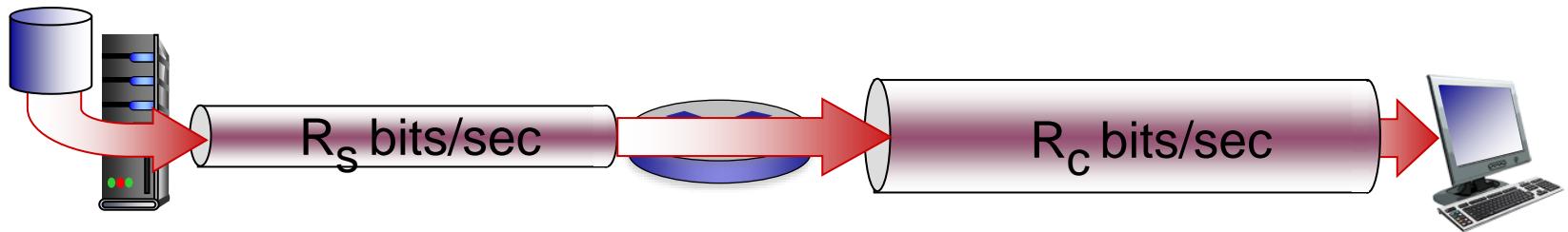
# Throughput - 1

- $R_s < R_c$  What is average end-end throughput?
- *Answer :  $R_s$*



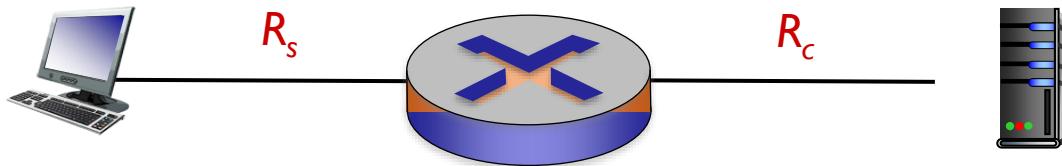
# Throughput - 2

- $R_s > R_c$  What is average end-end throughput?
- *Answer :  $R_c$*

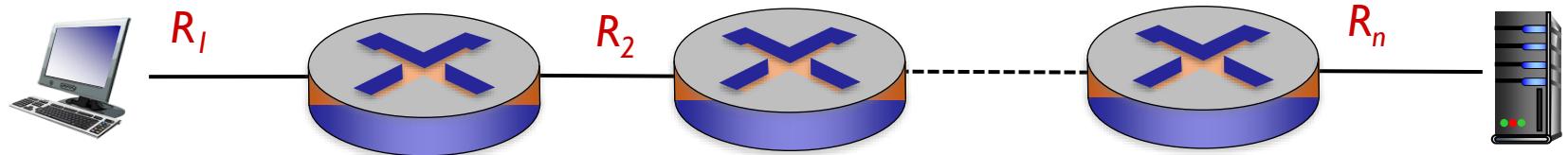


*bottleneck link*

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



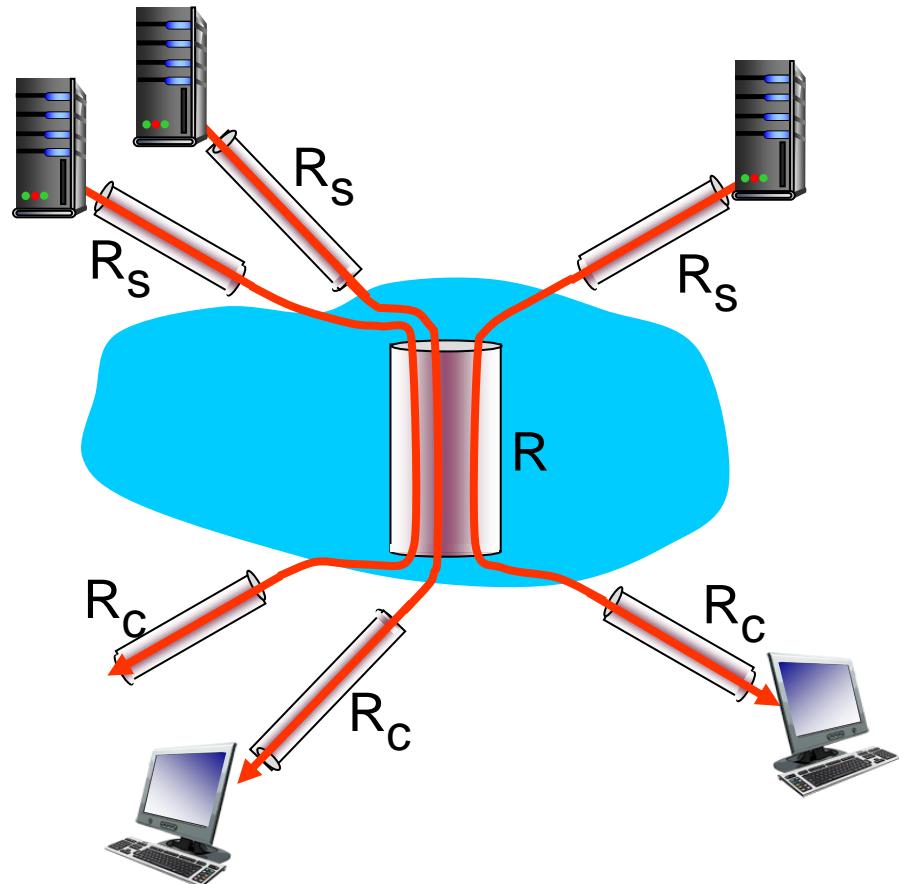
$$\text{Throughput} = \min \{R_s, R_c\}$$



$$\text{Throughput} = \min \{R_1, R_2, \dots, R_n\}$$

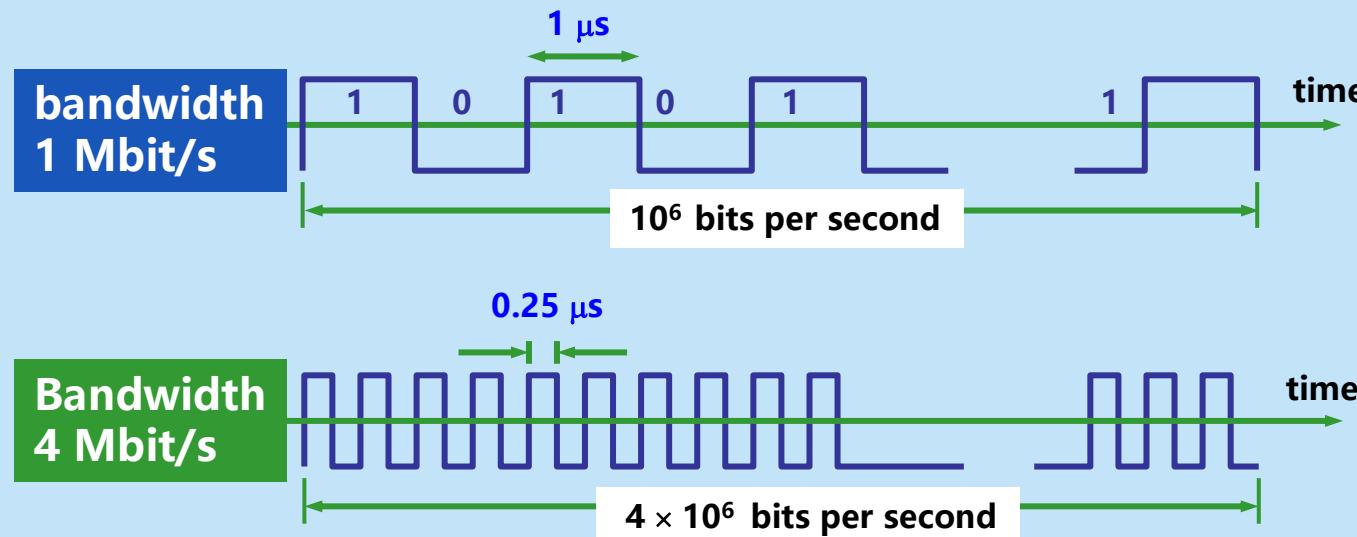
# A more common example in Internet

- per-connection end-end throughput:  
 $\min(R_c, R_s, R/10)$
- in practice:  $R_c$  or  $R_s$  is often bottleneck



10 connections (fairly) share  
backbone bottleneck link  $R$  bits/sec

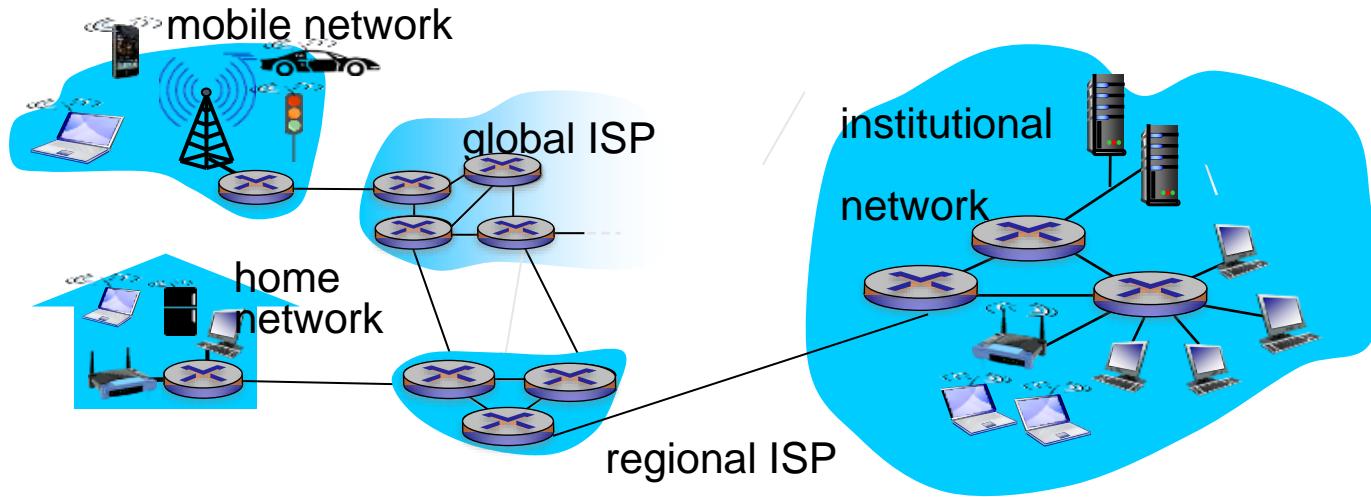
# bandwidth



The width of the signal on the time axis narrows as the bandwidth increases.

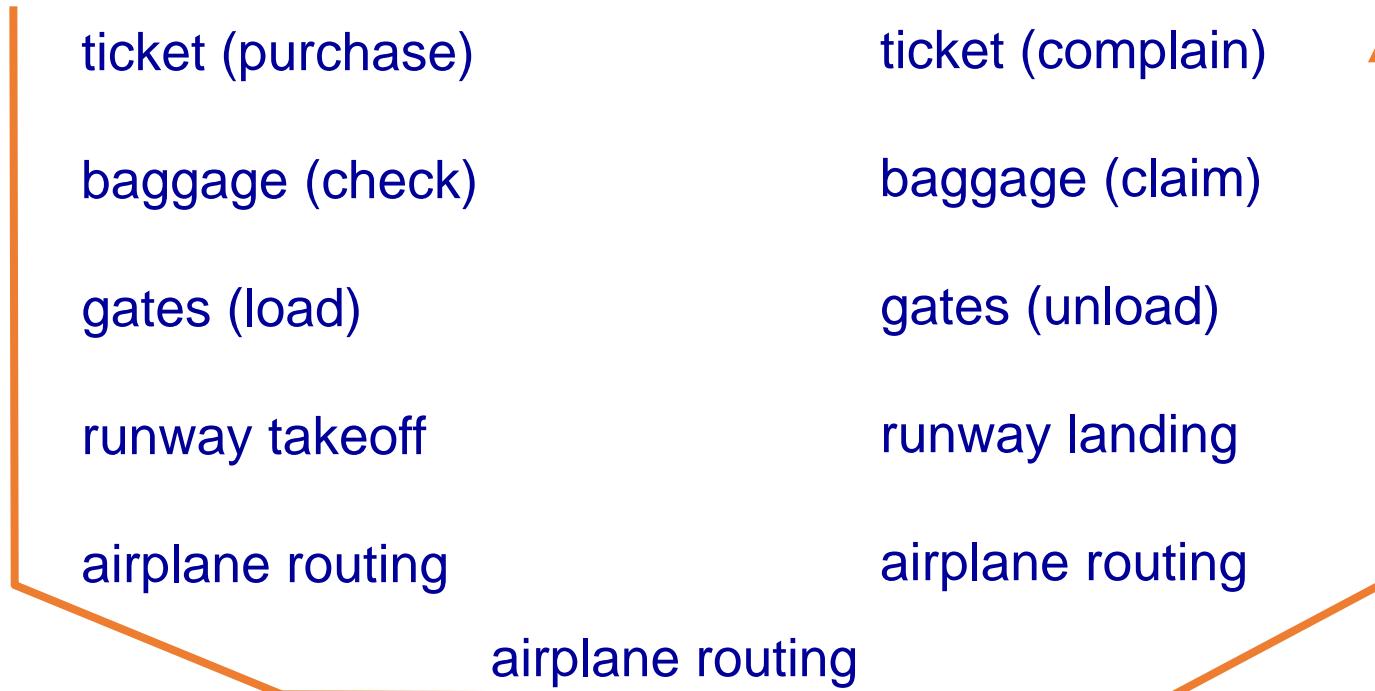
# Organizing the Internet

- Internet is a complex system:

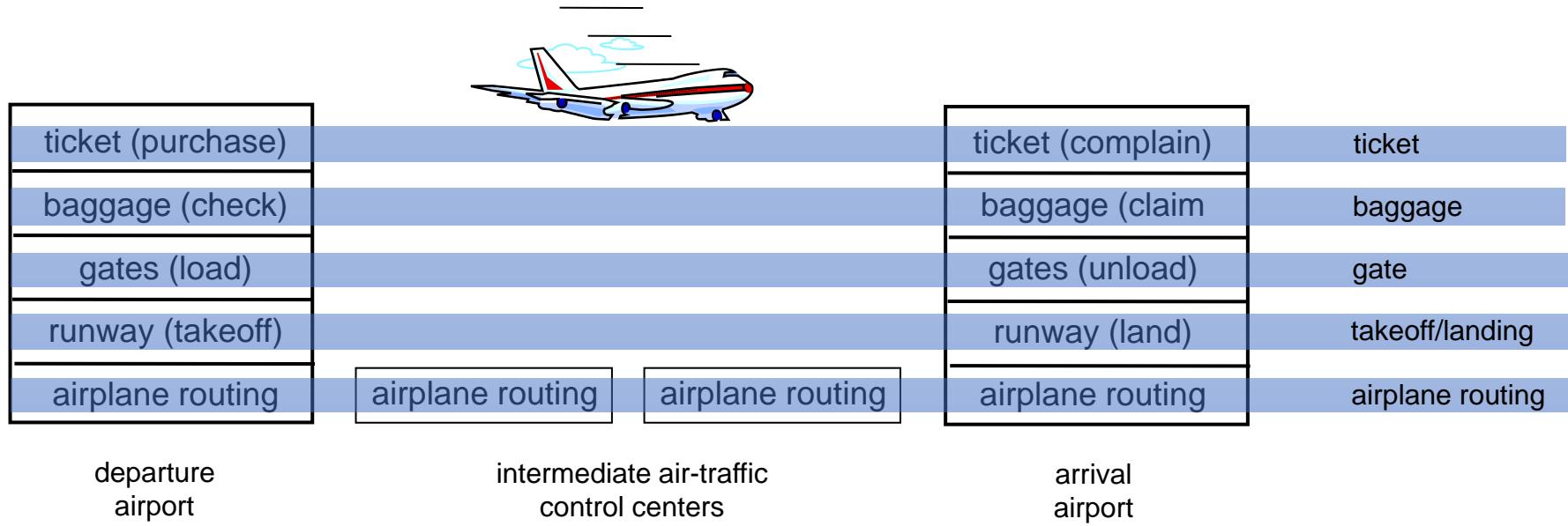


*Question :* is there any hope of organizing the structure of computer network?

# Analogy: Organization of air travel



# Layering of airline functionality



**layers**: each layer implements a **service**

- via its own internal-layer actions
  - relying on services provided by layer below

# Layered architectures

- Each layer is functionally independent
- Each layer has a defined interface to the previous & preceding layer
- Each layer builds on the previous layer
- Virtual communication takes place between layers at the same level
- Layered architectures are often called **protocol stacks**

# 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

# Pros & cons

## pros

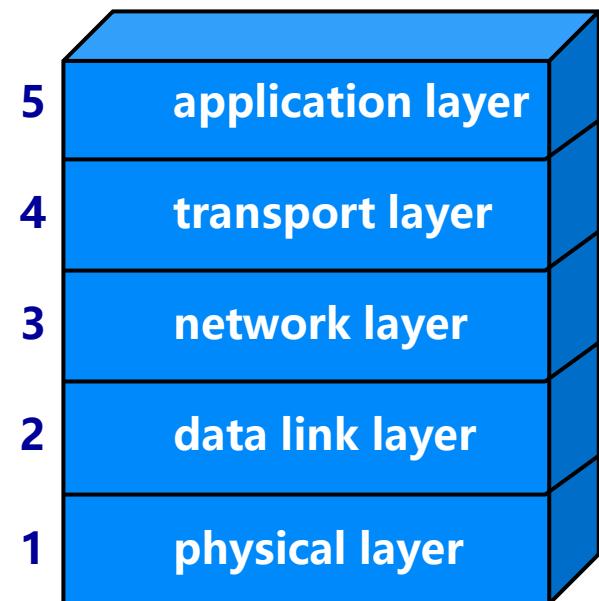
- independent
- flexible
- Separable structure
- eases maintenance
- Promote standardization

## cons

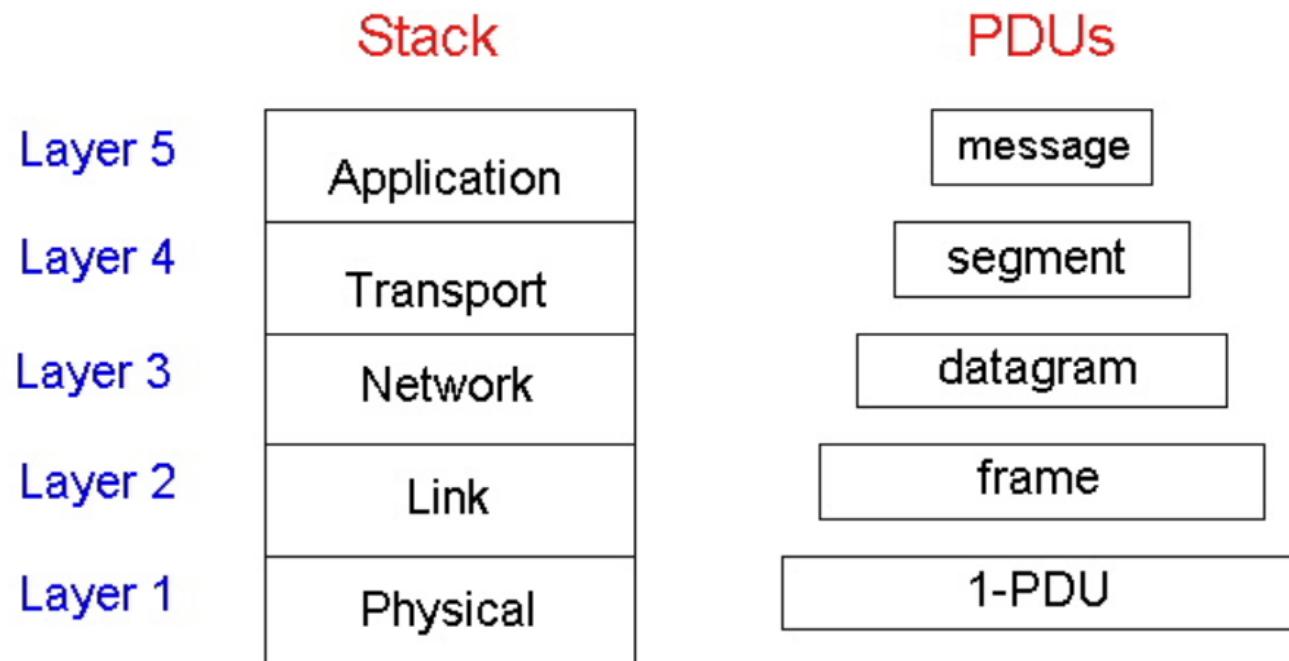
- not very efficient
- Similar function may appear in multi-layers with additional cost.

# Protocol stacks of Internet

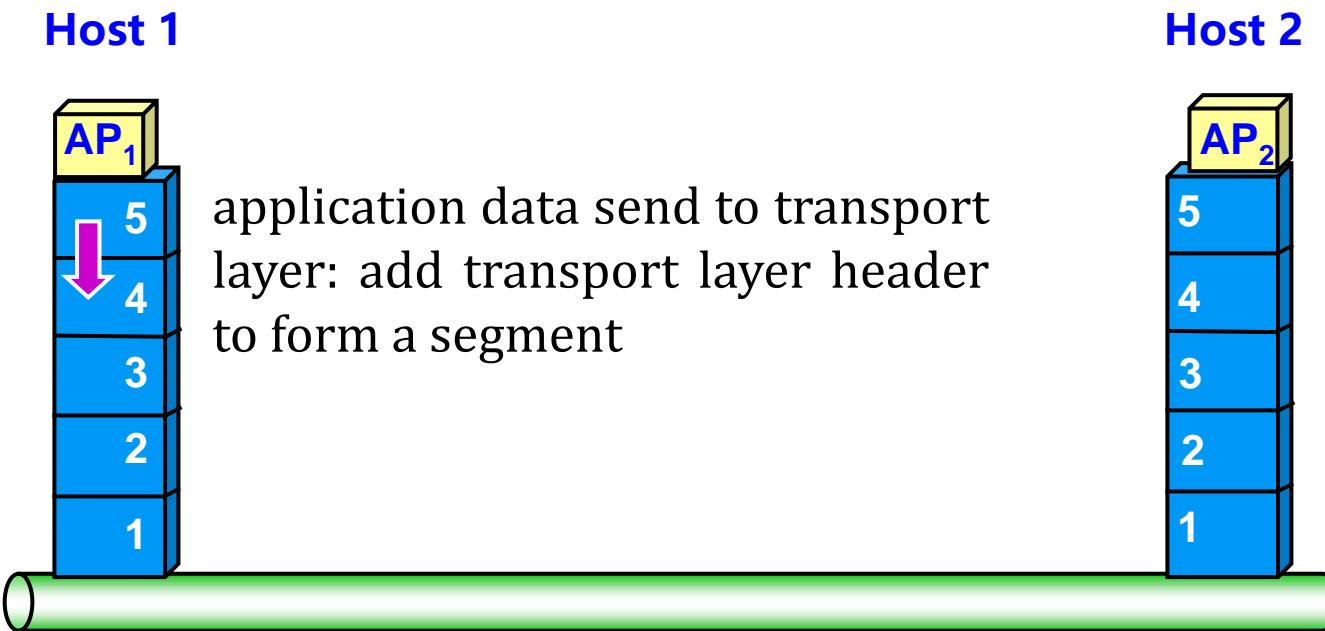
- *application*: supporting network applications
  - FTP, SMTP, HTTP
- *transport*: process-process data transfer
  - TCP, UDP
- *network*: routing of datagrams from source to destination
  - IP, routing protocols
- *link*: data transfer between neighboring network elements
  - Ethernet, 802.111 (WiFi), PPP
- *physical*: bits “on the wire”



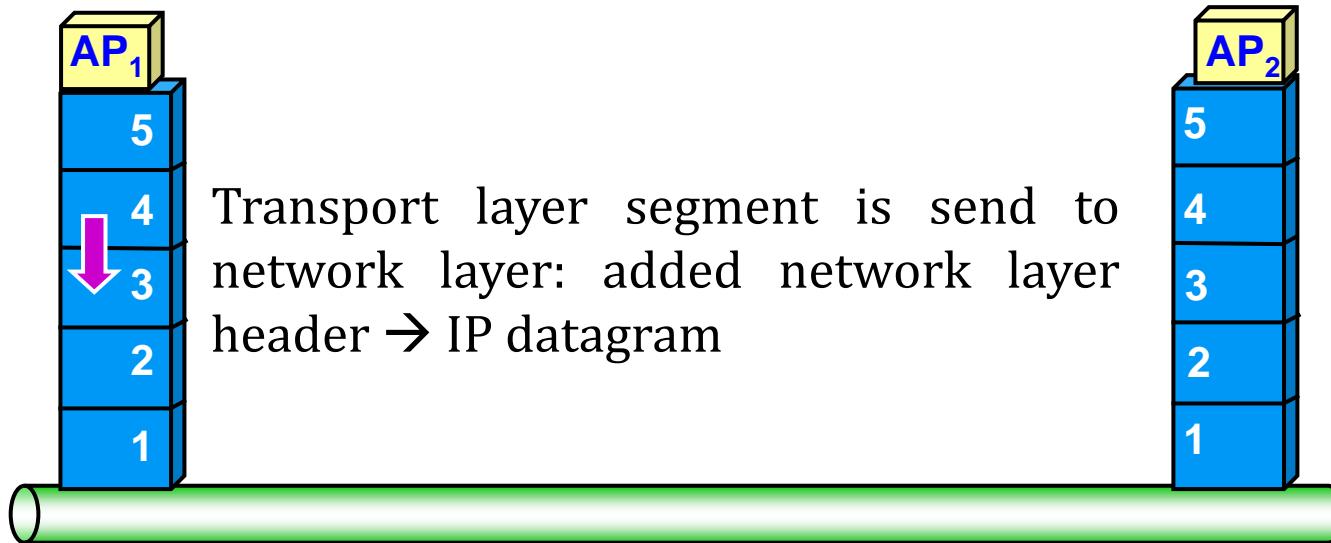
# The protocol stack, and protocol data units



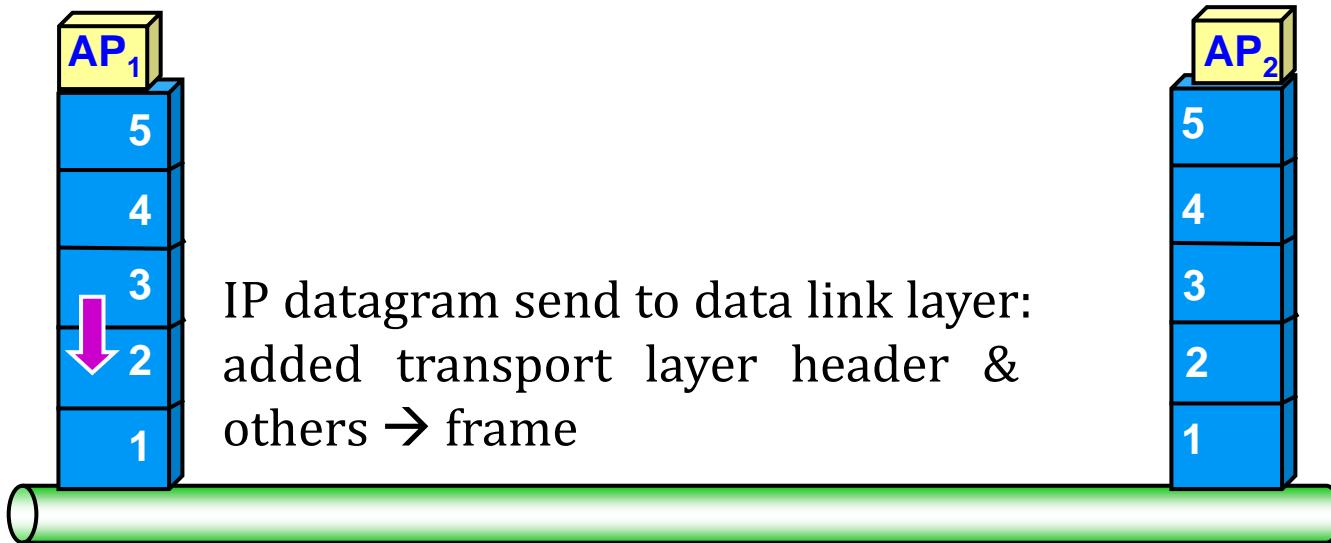
# Example



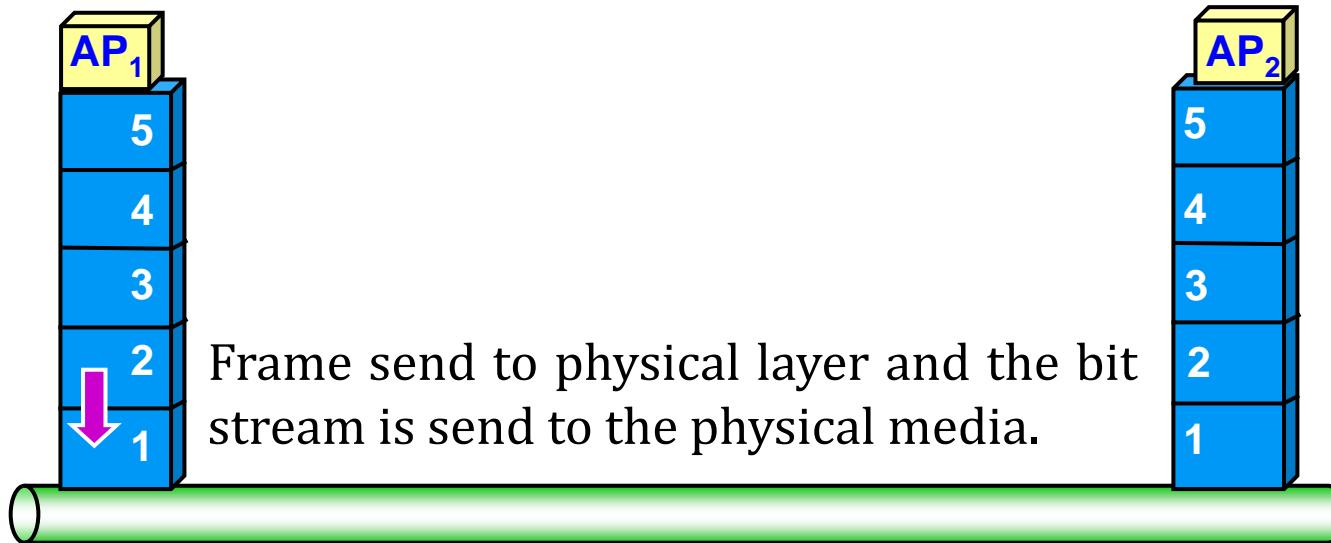
# Example



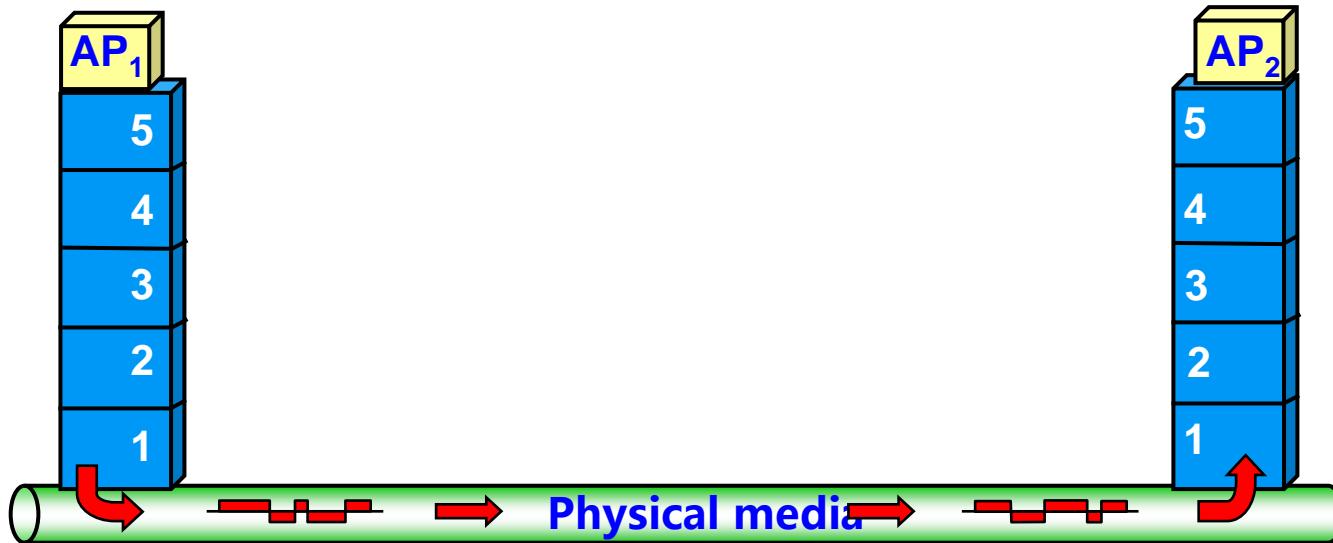
# Example



# Example

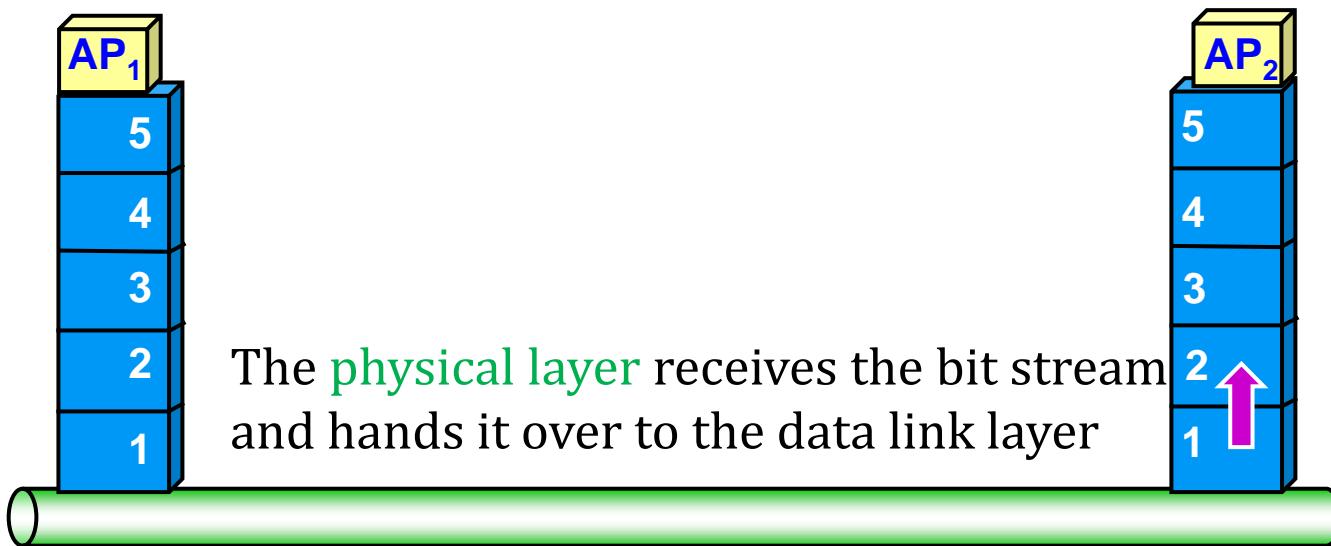


# Example

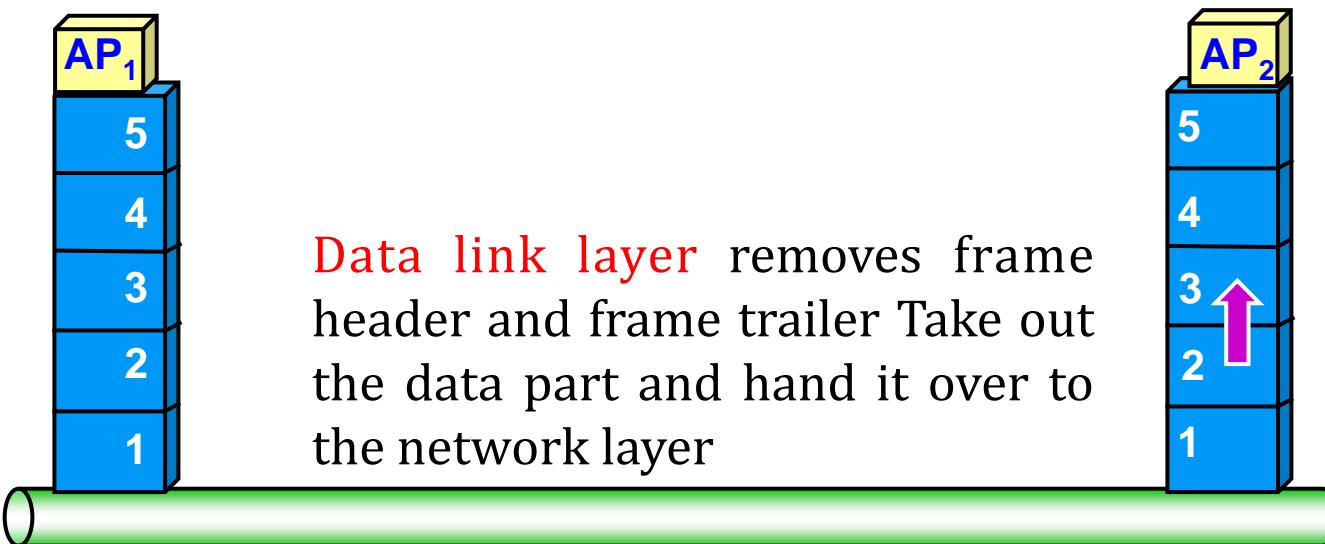


- Electrical signals (or optical signals) propagate in physical media
- From the physical layer of the sender to the physical layer of the receiver

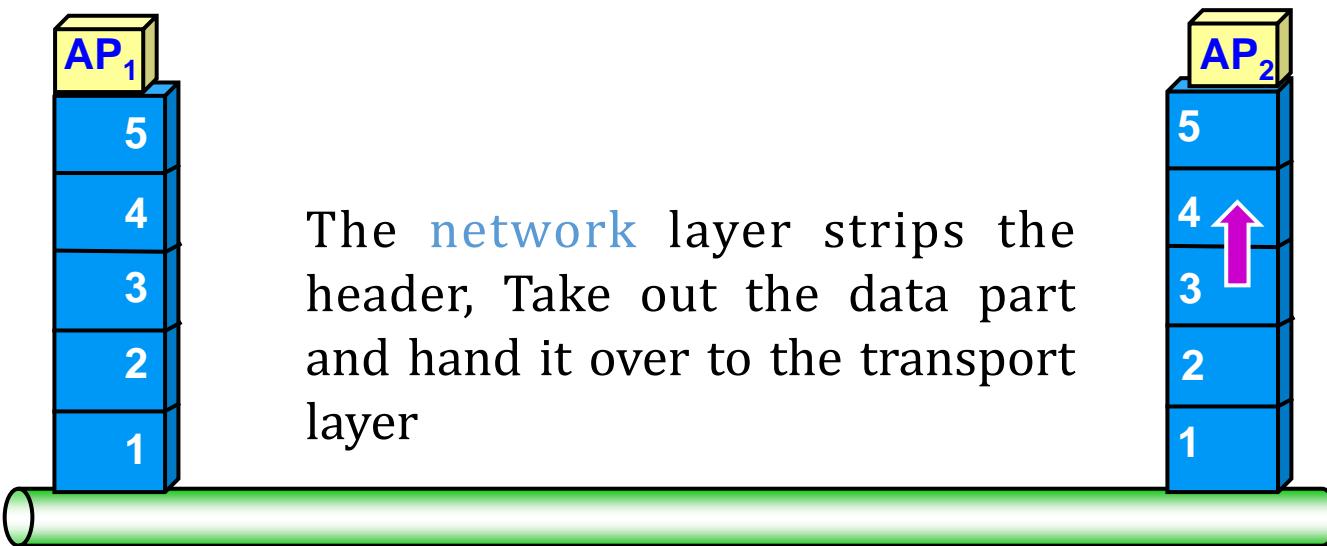
# Example



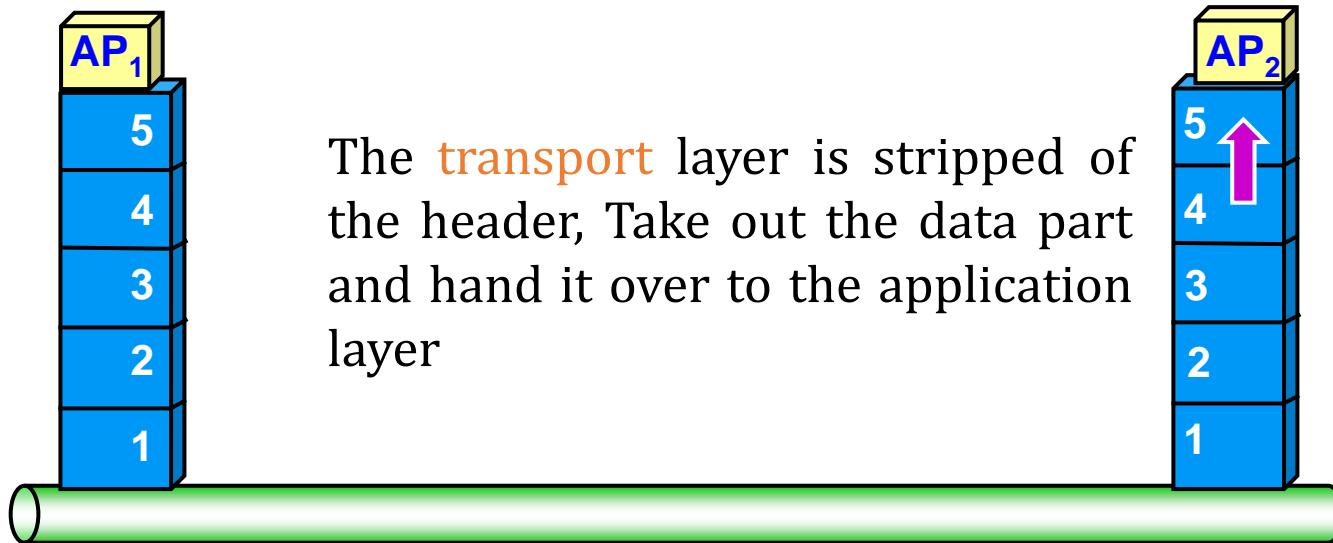
# Example



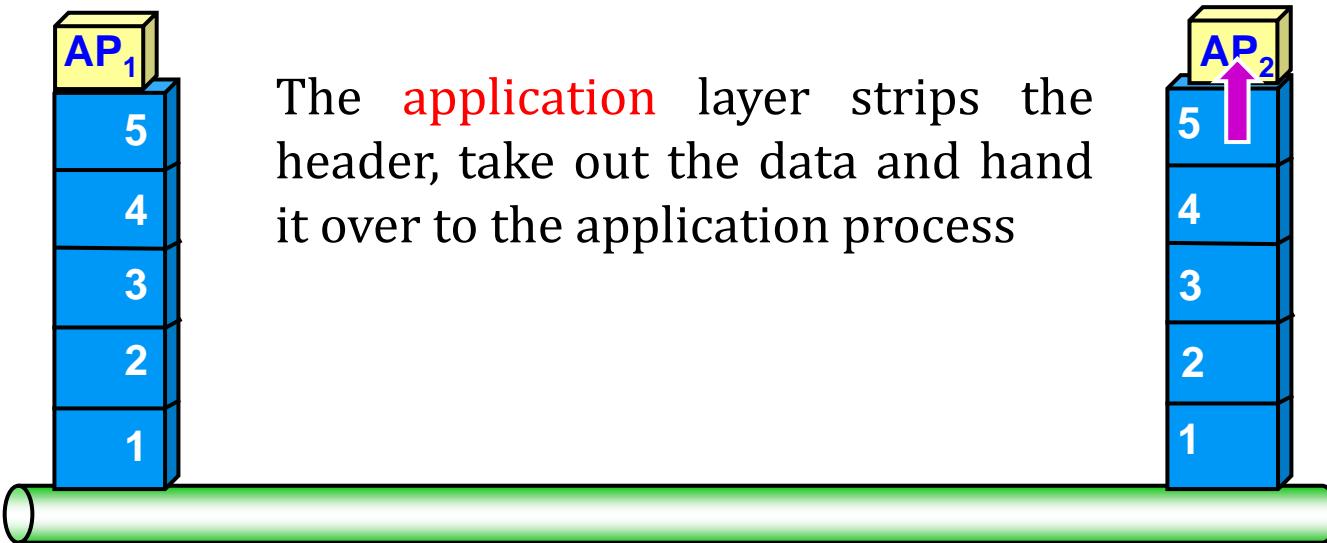
# Example



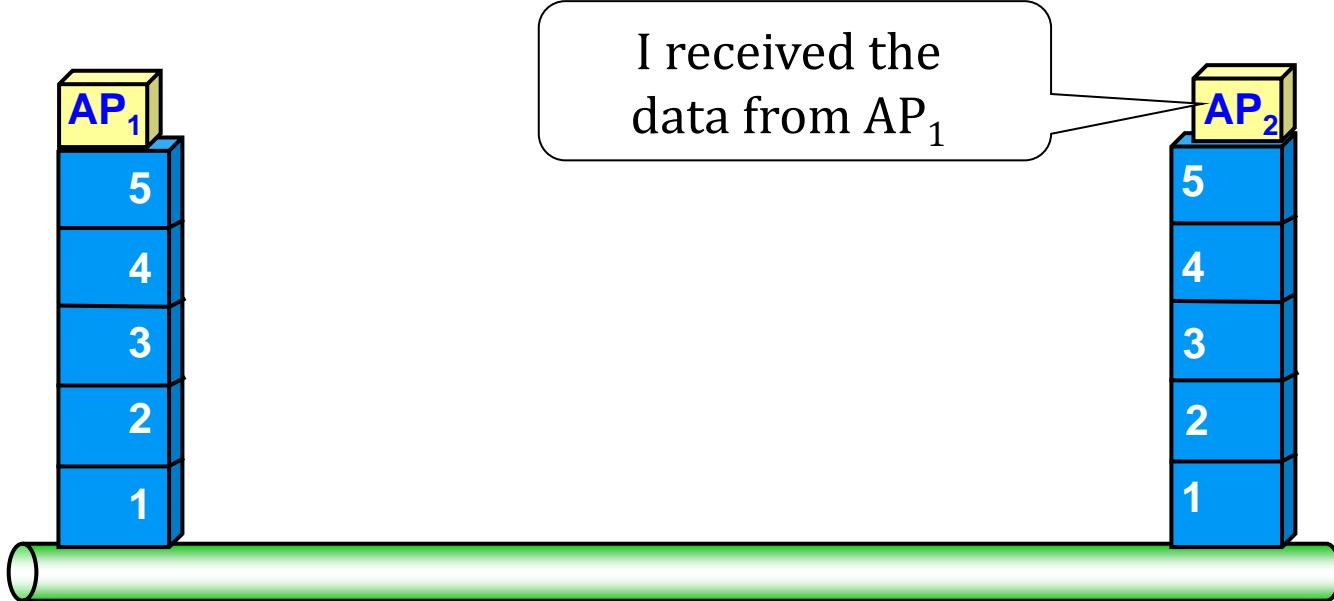
# Example



# Example

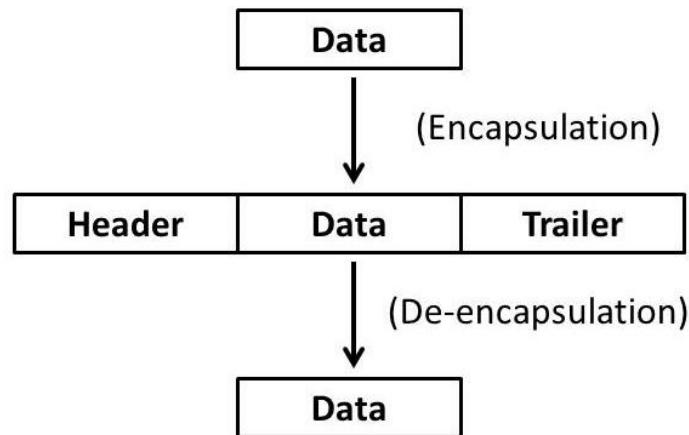


# Example

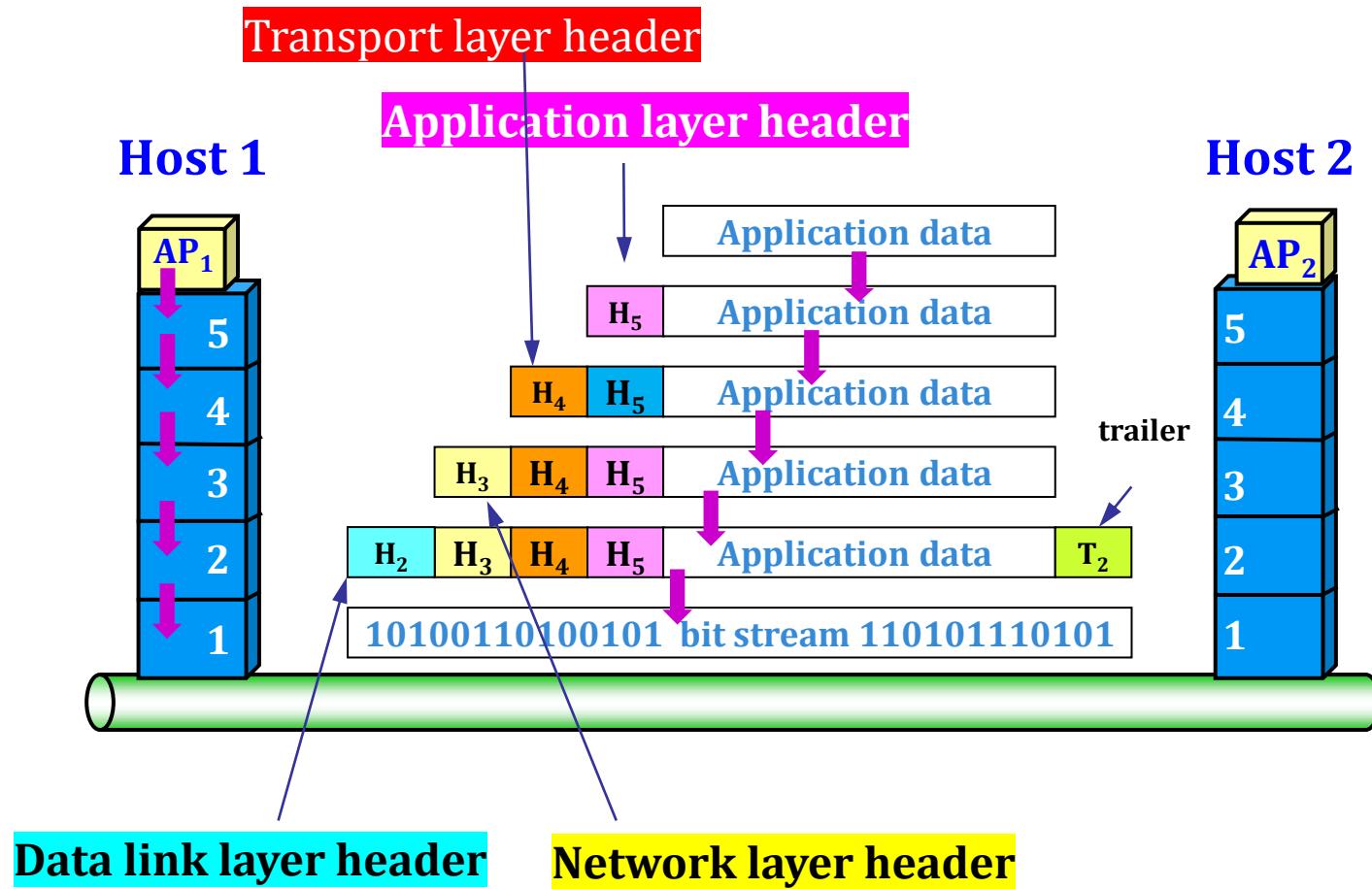


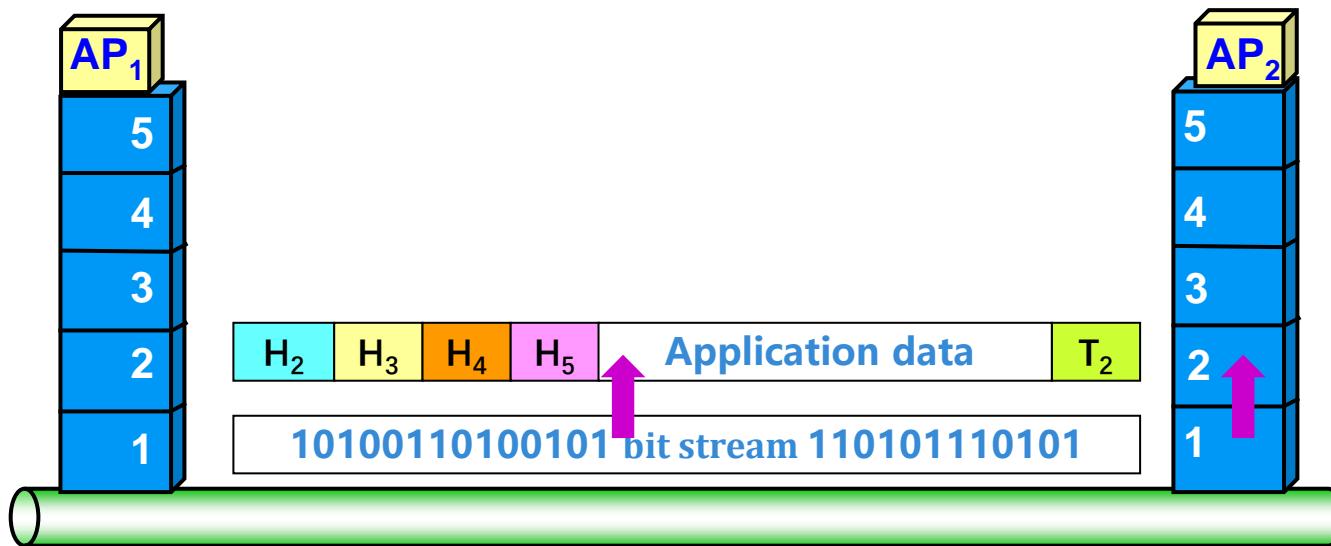
# Encapsulation

**Definition :** A process of adding control information as a message passes through the layered model



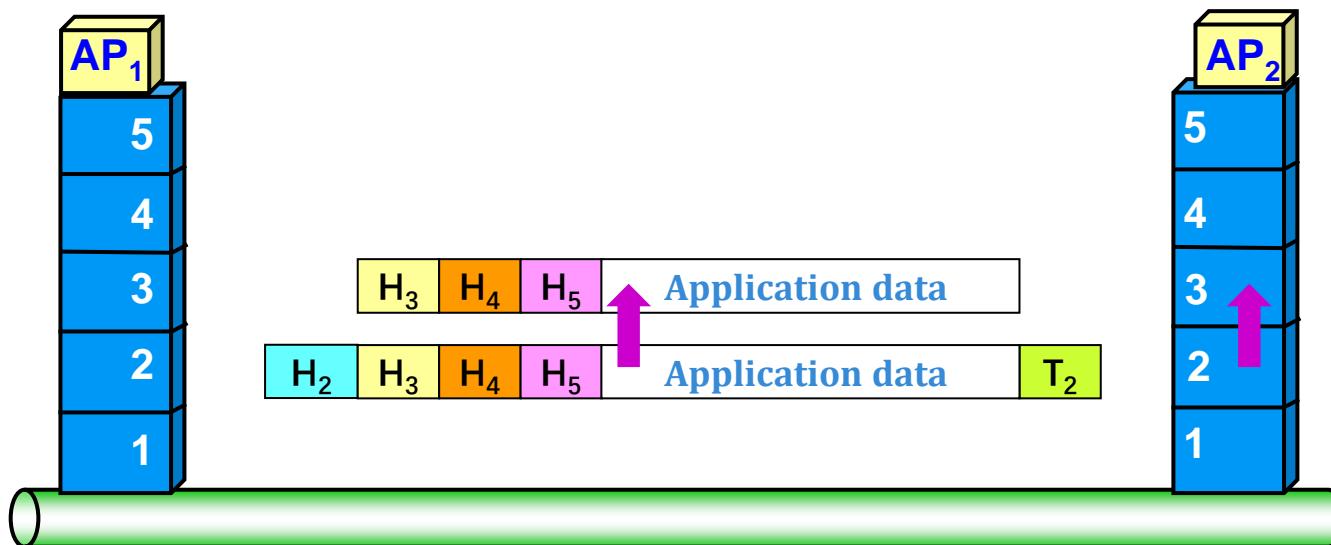
**Encapsulation and De-encapsulation**



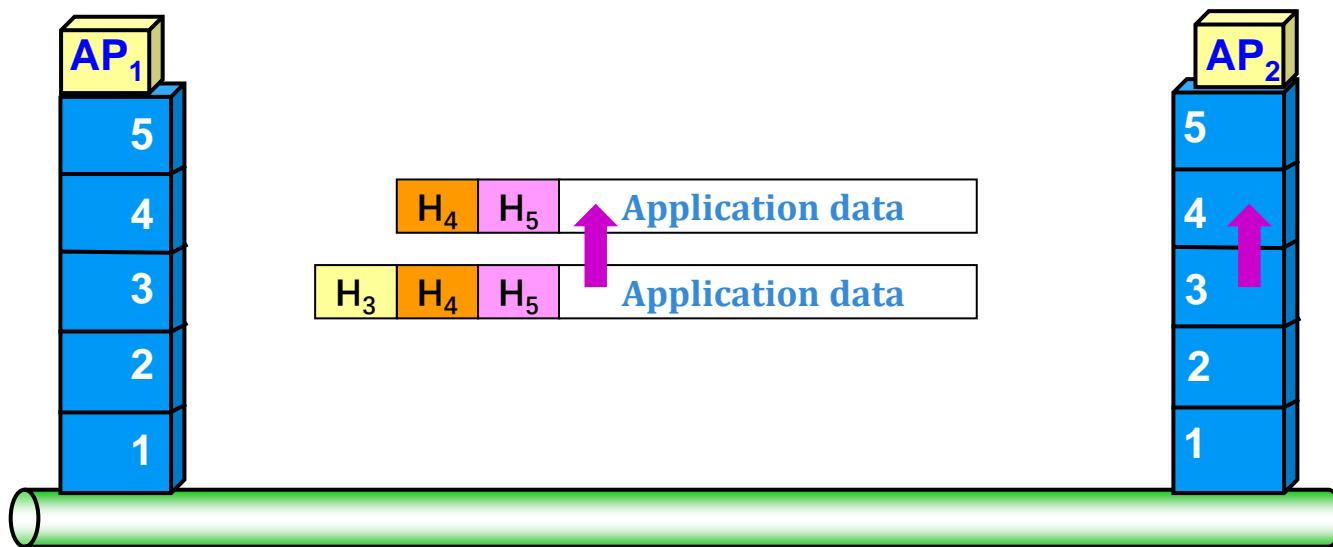


- 1. The **physical layer** receives the bit stream and hands it over to the data link layer

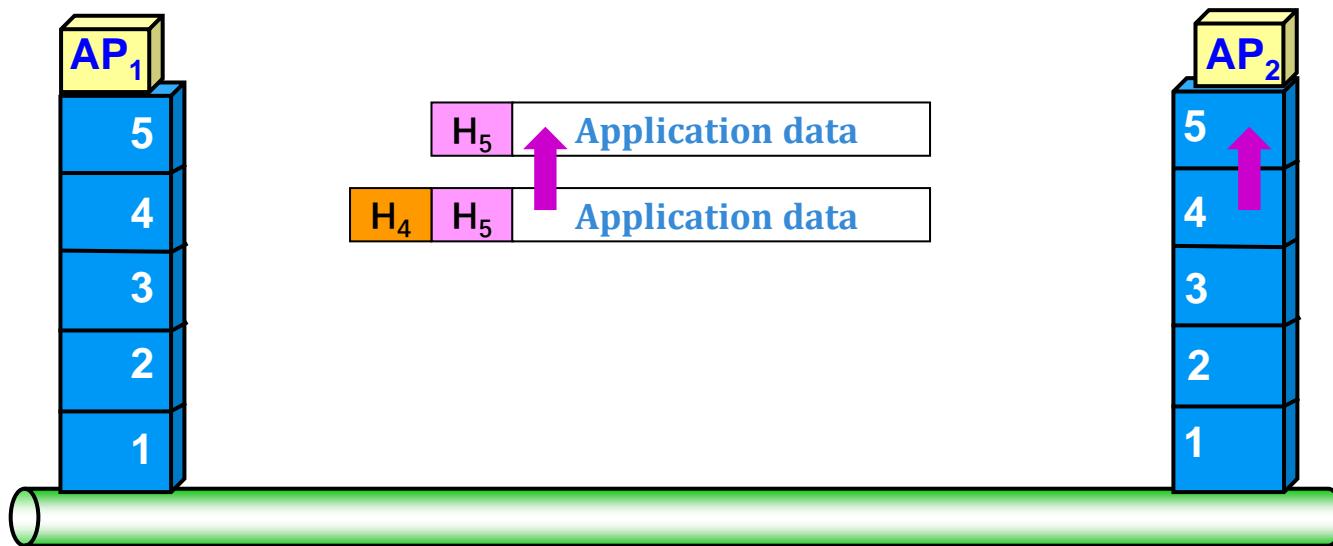
- 2, Data link layer
  - removes frame header & trailer,
  - take out the data part
  - hand it over to the network layer



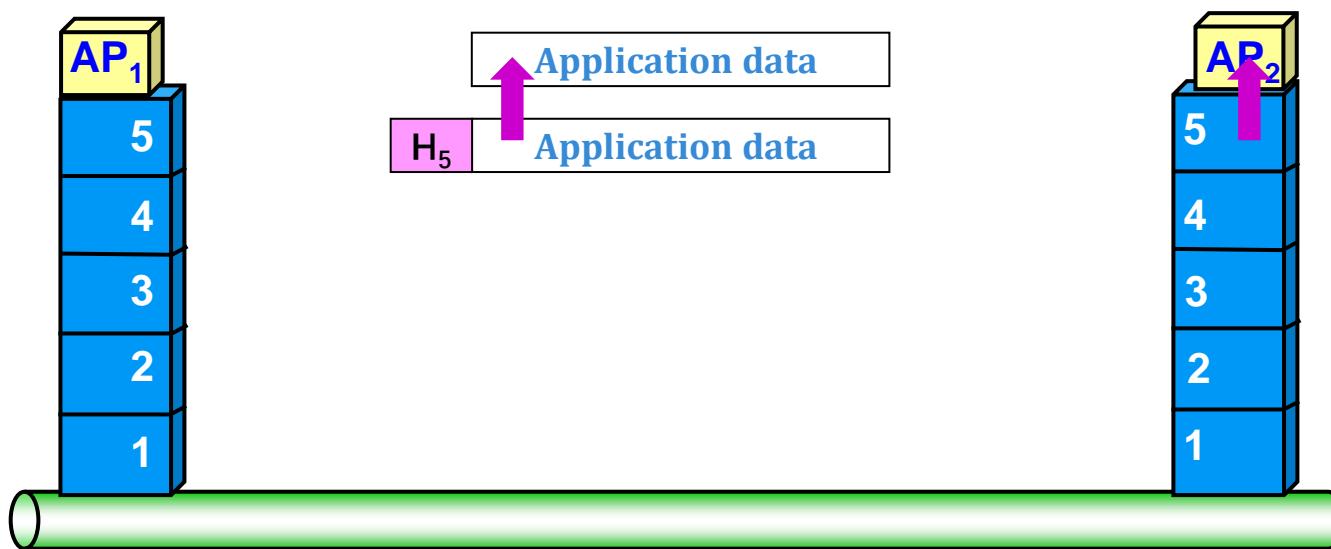
- 3, The **network** layer
  - strips the header,
  - take out the data part
  - hand it over to the transport layer

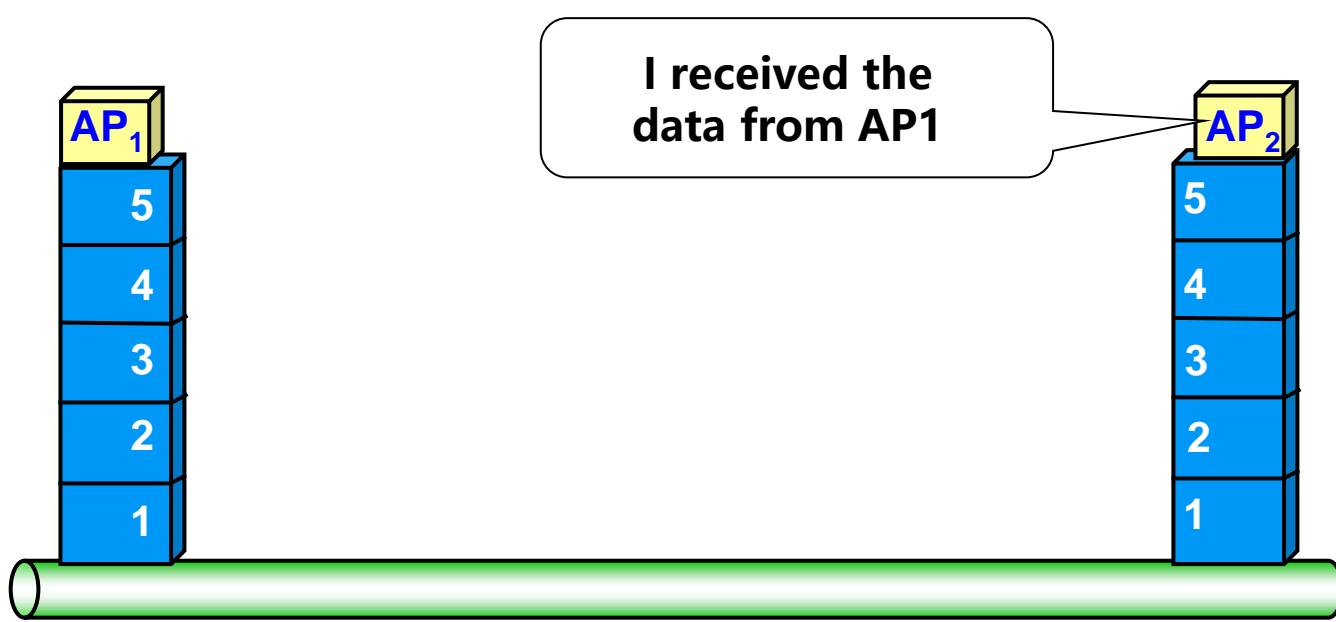


- 4, The transport layer
  - removes the header,
  - take out the data part
  - hand it over to the application layer

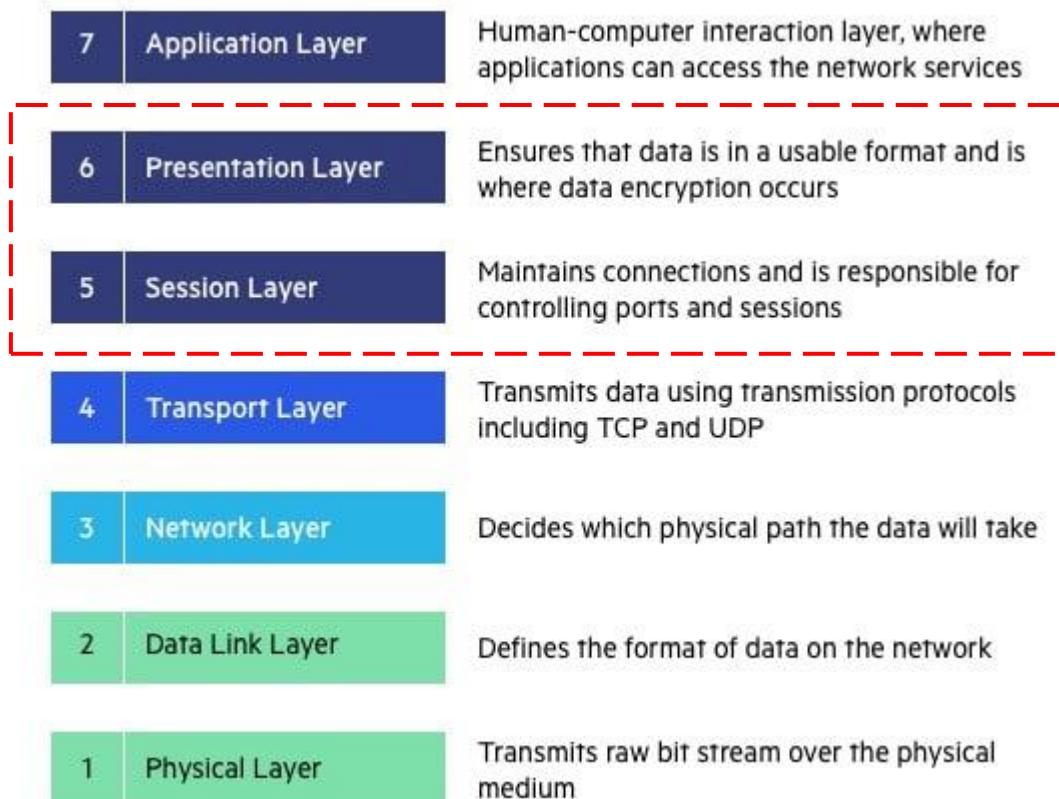


- 5, The application layer
  - strips the header
  - take out the data
  - hand it over to the application process

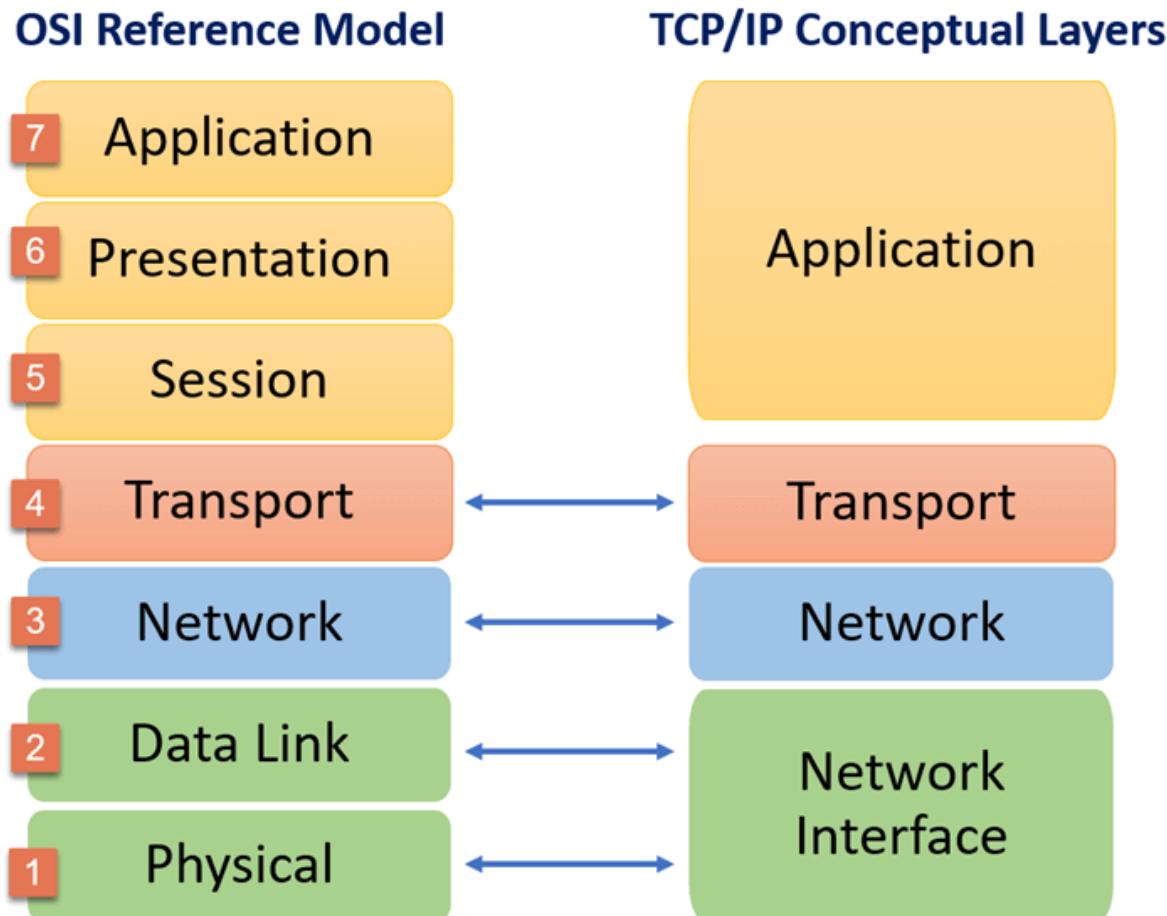




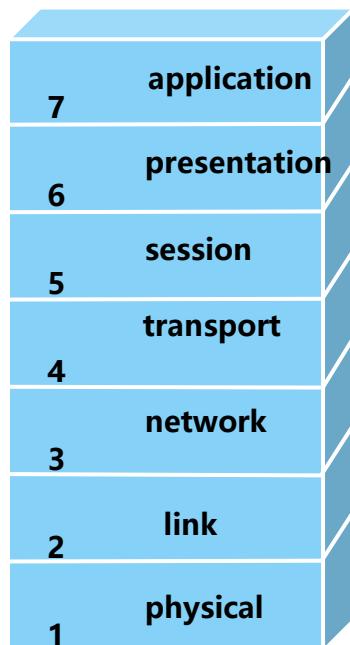
# Other protocol stack exists – OSI model



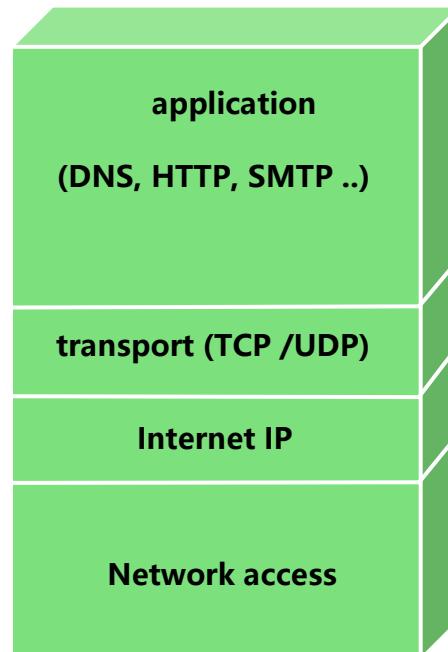
# TCP/IP model



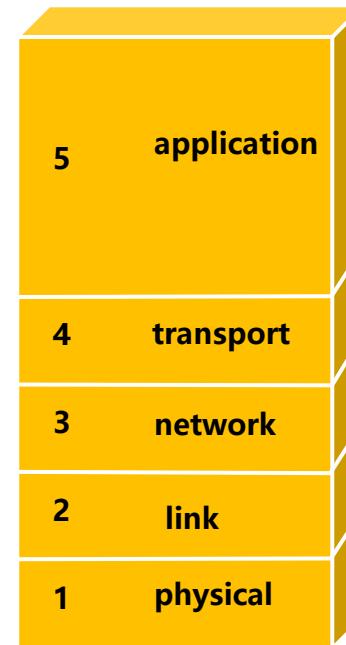
# Network protocol model(s)



(a) OSI



(b) TCP/IP



(c) Internet

# TAKEAWAYS

- Packet switching
- (Message switching)
- Network performance
  - Delay
  - Loss
  - Throughput
  - Bandwidth
- Layered protocol model
  - 5-layered model
  - Encapsulation & Decapsulation
  - OSI model & TCP/IP model

- End of Chapter 1 -



西北工业大学  
NORTHWESTERN POLYTECHNICAL UNIVERSITY



# Computer Networks

Lecturer: ZHANG Ying  
Fall semester 2022

# Chapter 2

# Application Layer

# Chapter outline

1

principles of network applications

2

Web & HTTP

3

Email

4

DNS

# Some network apps

Application Layer: Used by Network Applications



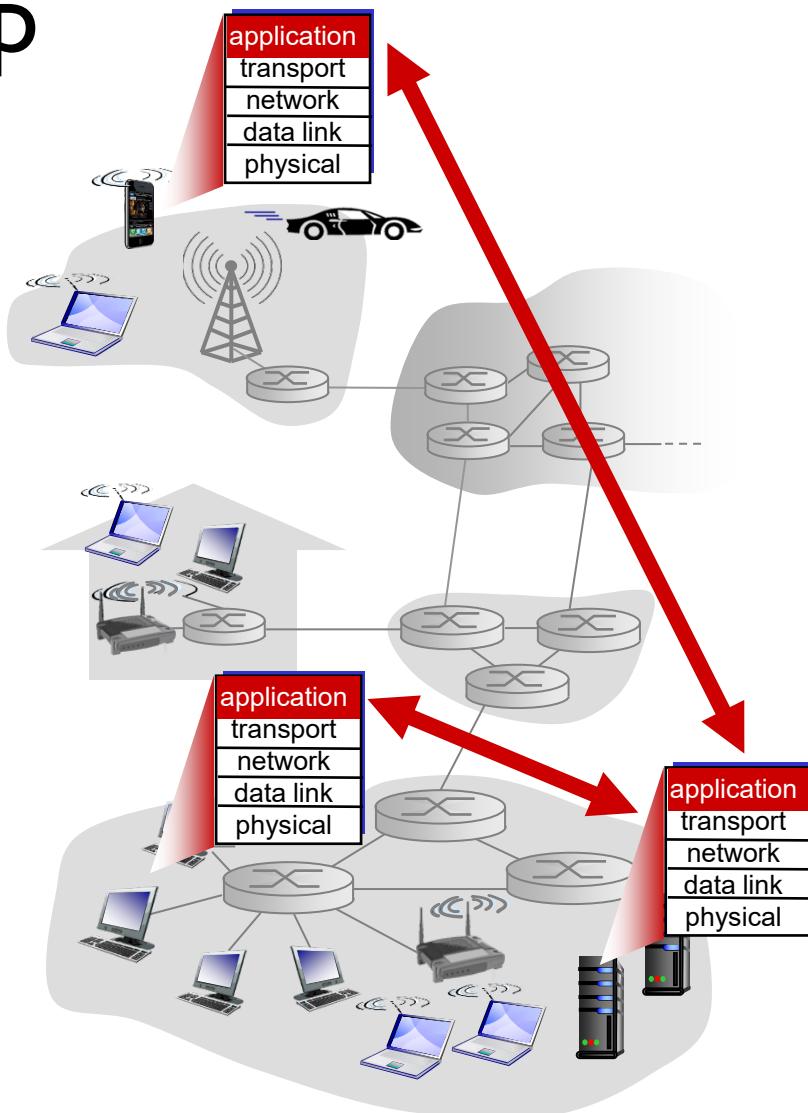
# Creating a network app

write programs that:

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

no need to write software for network-core devices

- network-core devices do not run user applications
- applications on end systems allows for rapid app development, propagation

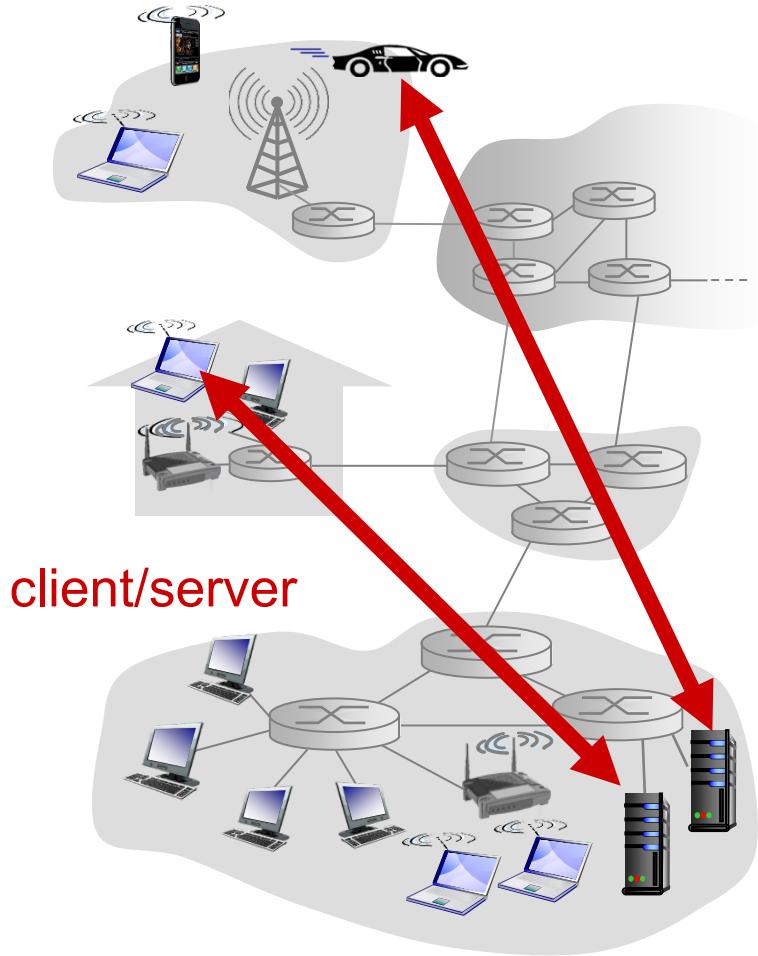


# Application architectures

possible structure of applications:

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

# Client-server architecture



## server:

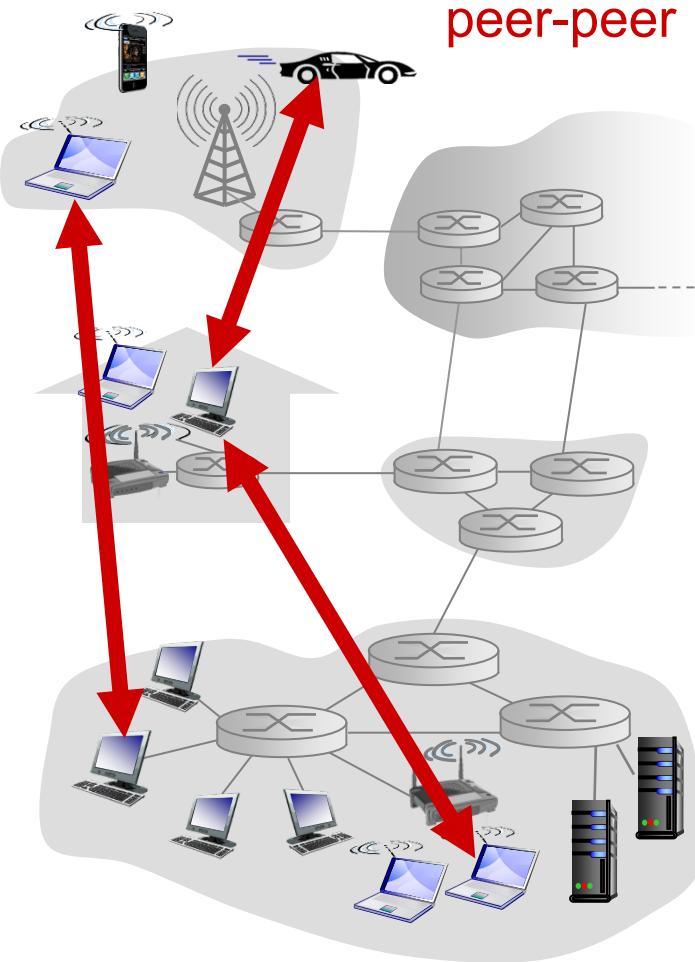
- always-on host
- permanent IP address
- data centers for scaling

## clients:

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

# P2P architecture

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



# Processes communicating

*process*: program running within a host

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

clients, servers

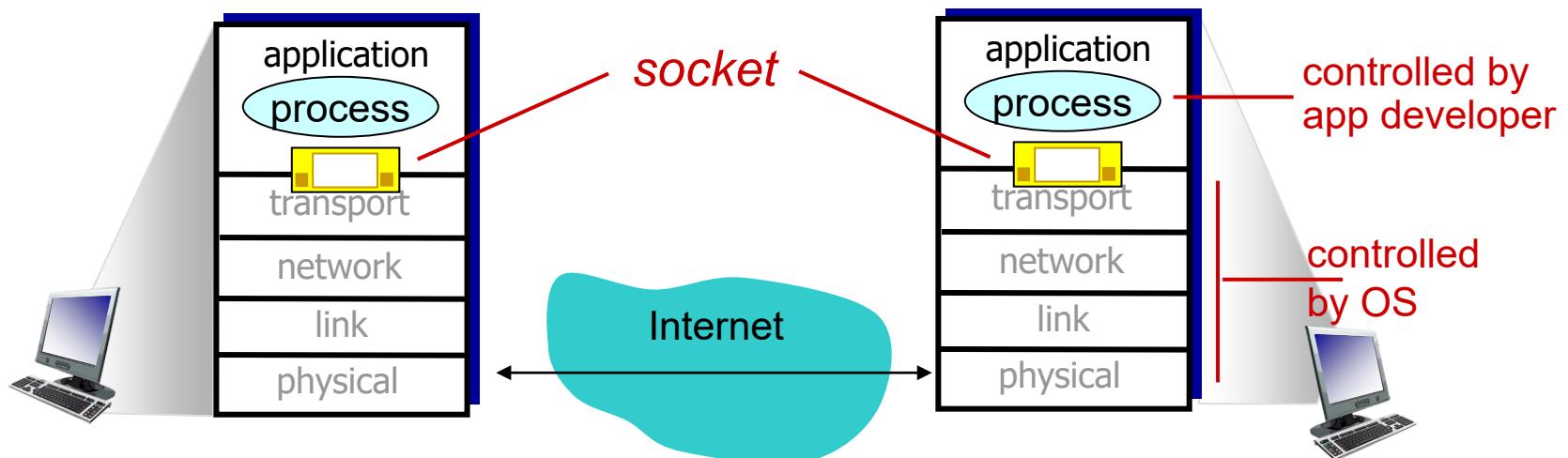
*client process*: process that initiates communication

*server process*: process that waits to be contacted

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

# Sockets

- process sends/receives messages to/from its **socket**
- socket analogous to door
  - sending process shoves message out door
  - sending process relies on transport infrastructure on other side of door to deliver message to socket at receiving process



# Addressing processes

- to receive messages, process must have *identifier*
- host device has unique 32-bit IP address
- Q: does IP address of host on which process runs suffice for identifying the process?
  - A: no, many processes can be running on same host
- *identifier* includes both IP address and port numbers associated with process on host.
- example port numbers:
  - HTTP server: 80
  - mail server: 25
- to send HTTP message to gaia.cs.umass.edu web server:
  - IP address: 128.119.245.12
  - port number: 80
- more shortly...

# IP address vs Port number

## IP ADDRESS

A numerical label assigned to each device connected to a computer network that uses the Internet Protocol for communication

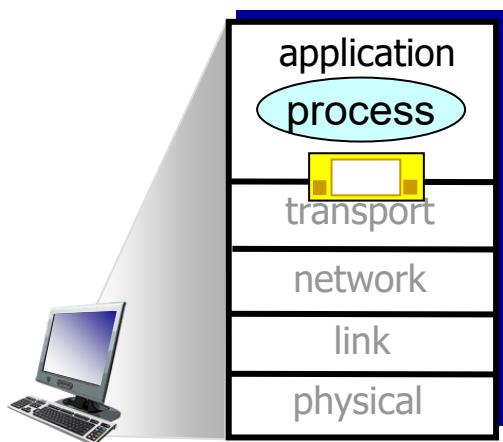
Used to identify a particular device in the network

## PORT NUMBER

A numerical value that is assigned to an application in an endpoint of communication

Used to identify a particular process executing in the device

# What transport service does an app need?



# What transport service does an app need?

## data integrity

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

## timing

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

## Throughput

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

## security

- encryption, data integrity, ...

# Transport service requirements: common apps

| <b>application</b>    | <b>data loss</b> | <b>throughput</b>                        | <b>time sensitive</b>             |
|-----------------------|------------------|--|-----------------------------------|
| 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<br>video:10kbps-5Mbps | yes, 100' s<br>msec               |
| stored audio/video    | loss-tolerant    | same as above                            |                                   |
| interactive games     | loss-tolerant    | few kbps up                              | yes, few secs                     |
| text messaging        | no loss          | elastic                                  | yes, 100' s<br>msec<br>yes and no |

# TCP Service

*TCP service:*

- *connection-oriented*: setup required between client and server processes
- *reliable transport* between sending and receiving process
- *congestion control*: throttle sender when network overloaded
- *flow control*: sender won't overwhelm receiver

*does not provide:*

- timing, minimum throughput guarantee, security

# UDP service

- *Connectionless: no handshaking before comm starts.*
- *unreliable data transfer* between sending and receiving process

*does not provide:* reliability, flow control, congestion control, timing, throughput guarantee, security, or connection setup,

# Internet apps: application, transport protocols

| application            | application<br>layer protocol           | underlying<br>transport protocol |
|------------------------|---|----------------------------------|
| e-mail                 | SMTP [RFC 2821]                         | TCP                              |
| remote terminal access | Telnet [RFC 854]                        | TCP                              |
| Web                    | HTTP [RFC 2616]                         | TCP                              |
| file transfer          | FTP [RFC 959]                           | TCP                              |
| streaming multimedia   | HTTP (e.g., YouTube),<br>RTP [RFC 1889] | TCP or UDP                       |
| Internet telephony     | SIP, RTP, proprietary<br>(e.g., Skype)  | TCP or UDP                       |

# Chapter outline

1 principles of network applications

2 Web & HTTP

3 Email

4 DNS

# World Wide Web (WWW)

- WWW: an information system enabling documents and other web resources to be accessed over the Internet
- Also known as **Web**



# Web page

*First, a review...*

- *web page* consists of *objects*
- object can be HTML file, JPEG image, audio file,...
- web page consists of *base HTML-file* which includes *several referenced objects*
- each object is addressable by a *URL*, e.g.,

www.someschool.edu/someDept/pic.gif

host name

path name

# Hypertext Transfer Protocol

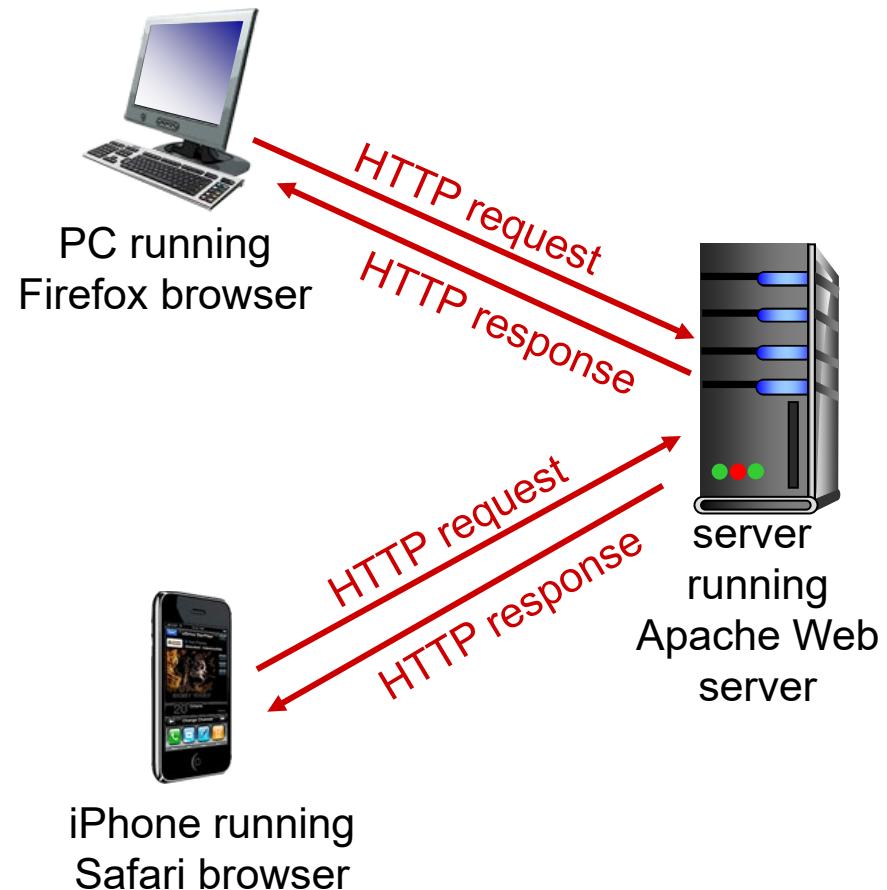
- The Hypertext Transfer Protocol (HTTP), the Web's application-layer protocol, is at the heart of the Web.



# 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

suppose user enters URL:

`www.someSchool.edu/someDepartment/home.index`

(contains text,  
references to 10  
jpeg images)

Ia. HTTP client initiates TCP connection to HTTP server (process) at `www.someSchool.edu` on port 80

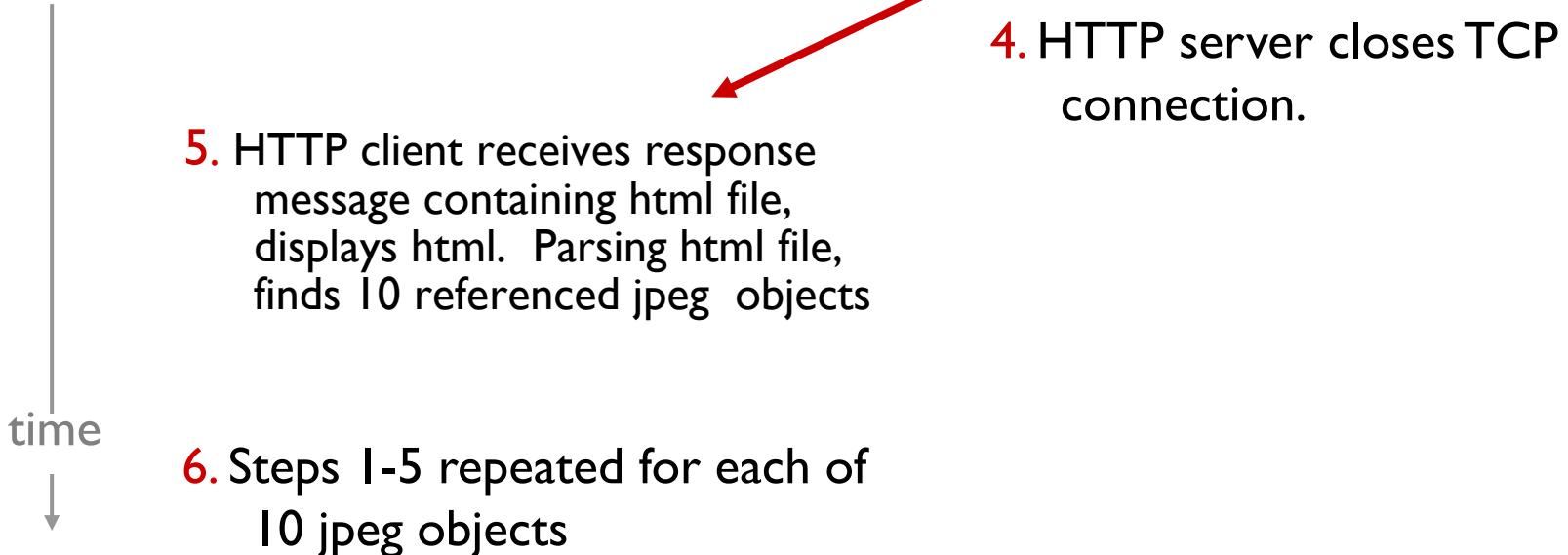
Ib. HTTP server at host `www.someSchool.edu` waiting for TCP connection at port 80. “accepts” connection, notifying client

2. HTTP client sends HTTP *request message* (containing URL) into TCP connection socket. Message indicates that client wants object `someDepartment/home.index`

3. HTTP server receives request message, forms *response message* containing requested object, and sends message into its socket

time ↓

# Non-persistent HTTP (cont.)



11 TCP connections are generated.



西北工业大学  
NORTHWESTERN POLYTECHNICAL UNIVERSITY



# Computer Networks

Lecturer: ZHANG Ying  
Fall semester 2022

# Chapter 2

# Application Layer

# Chapter Outline

1

principles of network applications

2

Web and HTTP

# HTTP connections

## *non-persistent HTTP*

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

## *persistent HTTP*

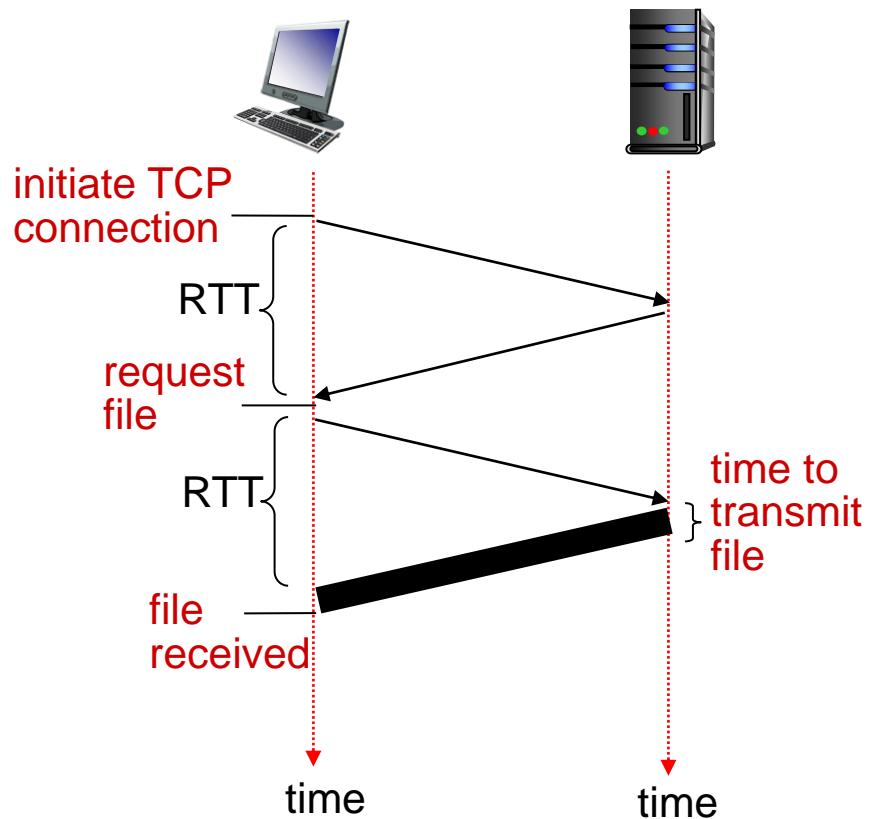
- multiple objects can be sent over single TCP connection between client, server

# 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 =  $2\text{RTT} + \text{file transmission time}$**



# non-persistent HTTP issues

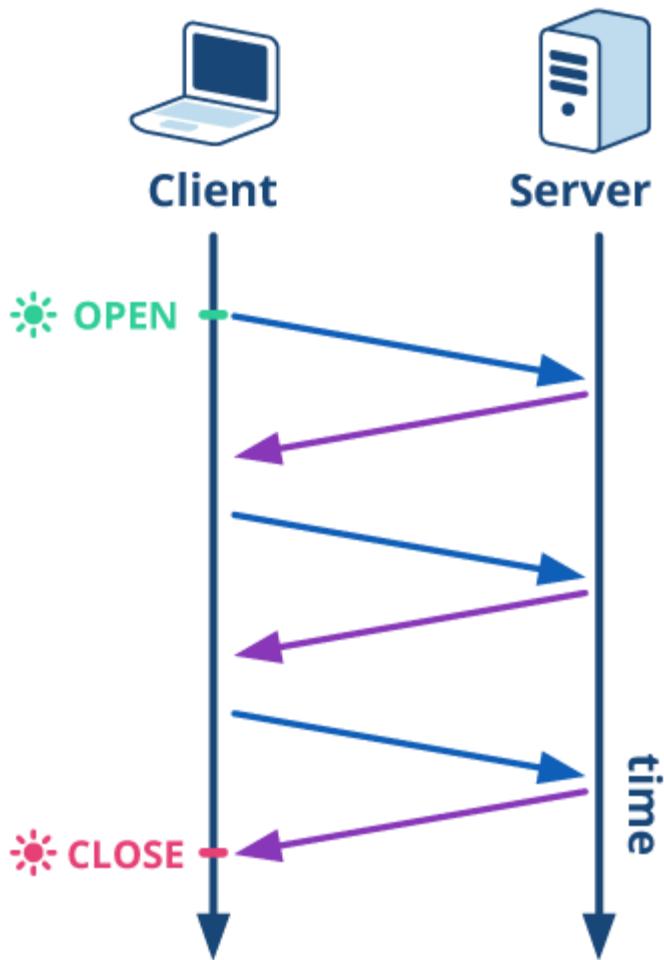
- OS overhead for *each* TCP connection
- requires 2 RTTs per object
- Slow start
- browsers often open parallel TCP connections to fetch referenced objects

# Persistent HTTP

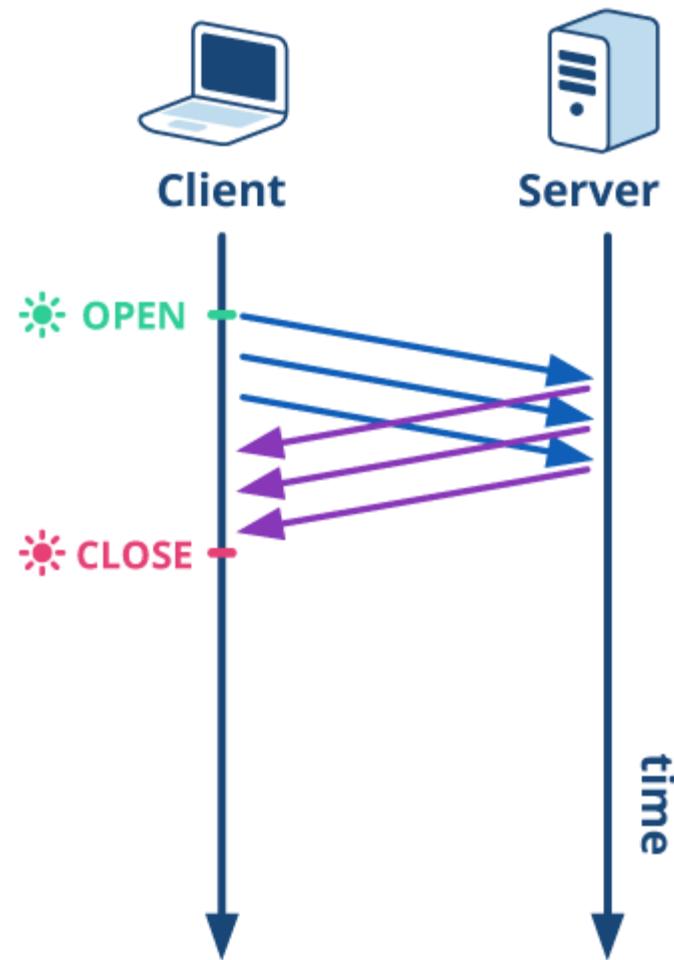
## *persistent HTTP:*

- 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

## NO PIPELINING



## PIPELINING



# HTTP request message

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

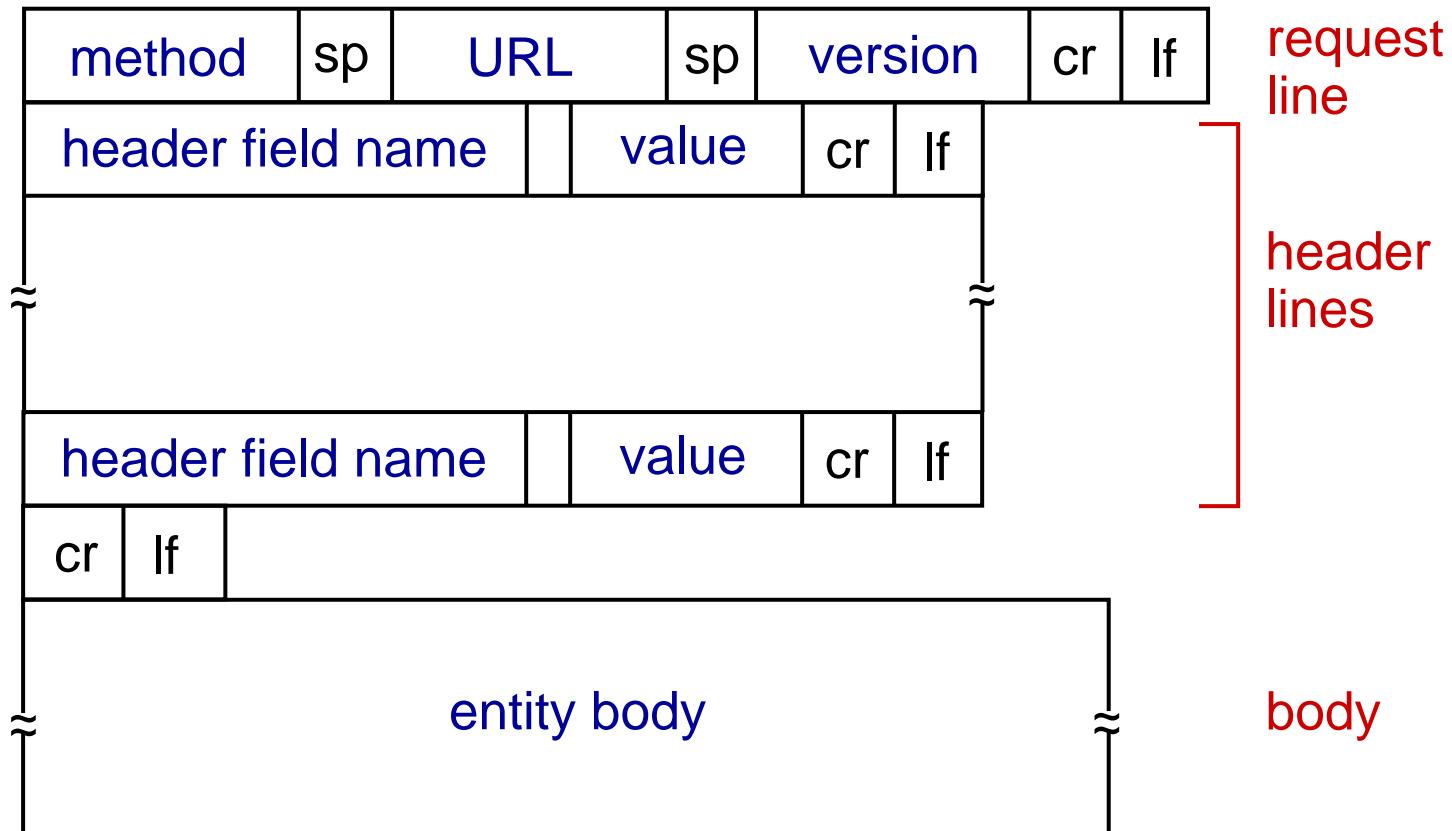
request line

(GET, POST, HEAD commands)

```
GET /somedir/page.html HTTP/1.1
Connection: close
User-agent: Mozilla/5.0
Accept: text/html, image/gif, image/jpeg
Accept-language:fr
```

header  
lines

# HTTP request message: general format



**HTTP Request Methods**  
GET, POST, PUT, DELETE

# Method types

## HTTP/1.0:

- GET
- POST
- HEAD
  - asks server to leave requested object out of response

## HTTP/1.1:

- GET, POST, HEAD
- PUT
  - uploads file in entity body to path specified in URL field
- DELETE
  - deletes file specified in the URL field

# HTTP response message Example

status line (protocol status code status phrase)

HTTP/1.1 200 OK

Connection: close

Date: Thu, 06 Aug 2018 12:00:15 GMT

Server: Apache/1.3.0 (Unix)

Last-Modified: Mon, 22 Jun 2018 09:23:24 GMT

Content-Length: 6821

Content-Type: text/html

*data data data data ...*

header  
lines

data, e.g., requested HTML file

# HTTP response status codes

- status code indicates whether a specific HTTP request has been successfully completed

1. [Informational responses](#) ( 100 – 199 )
2. [Successful responses](#) ( 200 – 299 )
3. [Redirection messages](#) ( 300 – 399 )
4. [Client error responses](#) ( 400 – 499 )
5. [Server error responses](#) ( 500 – 599 )

# HTTP response status codes

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

## **200 OK**

- request succeeded, requested object later in this msg

## **301 Moved Permanently**

- requested object moved, new location specified later in this msg  
(Location:)

## **400 Bad Request**

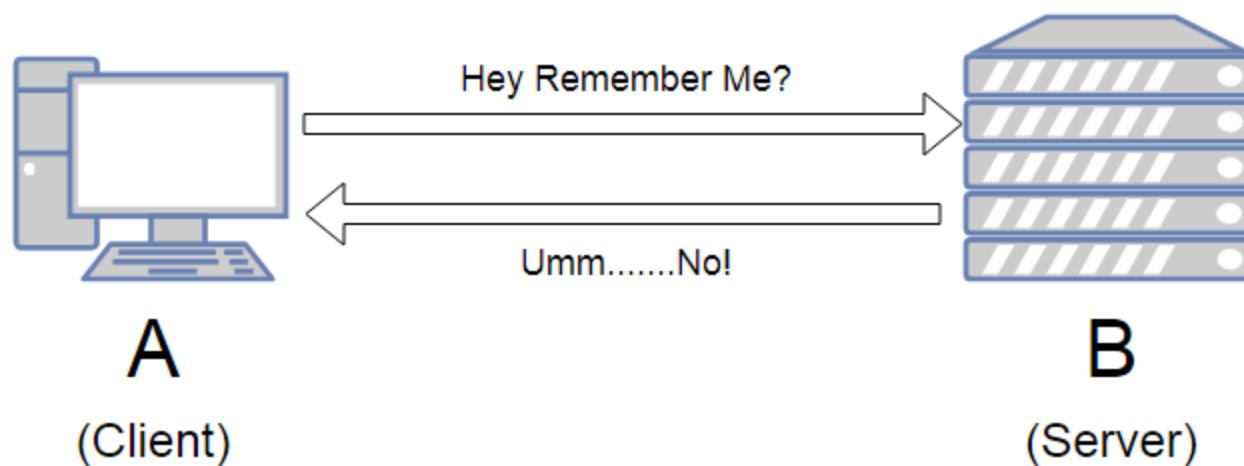
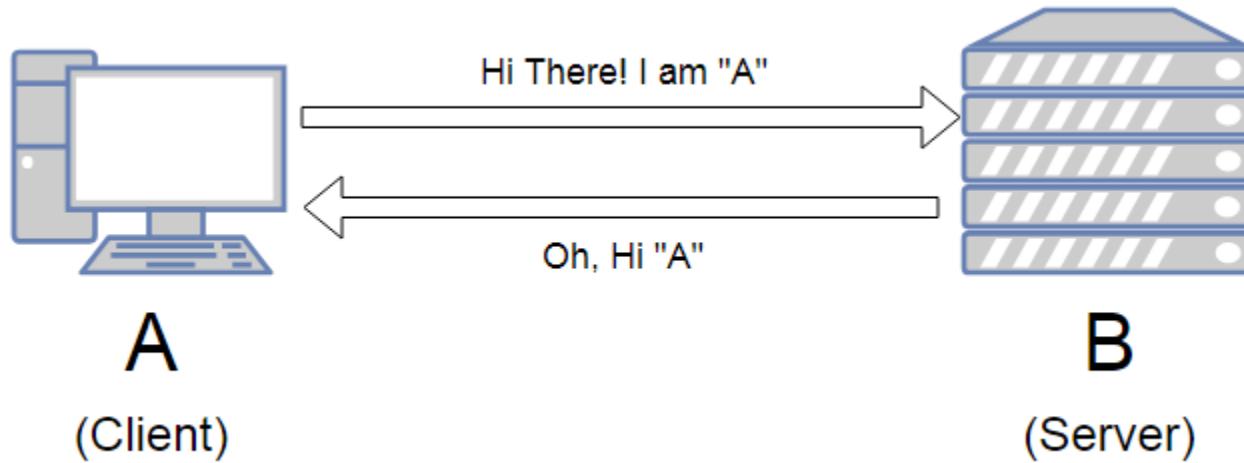
- request msg not understood by server

## **404 Not Found**

- requested document not found on this server

## **505 HTTP Version Not Supported**

# http: a stateless protocol



# User-server state: cookies

many Web sites use cookies

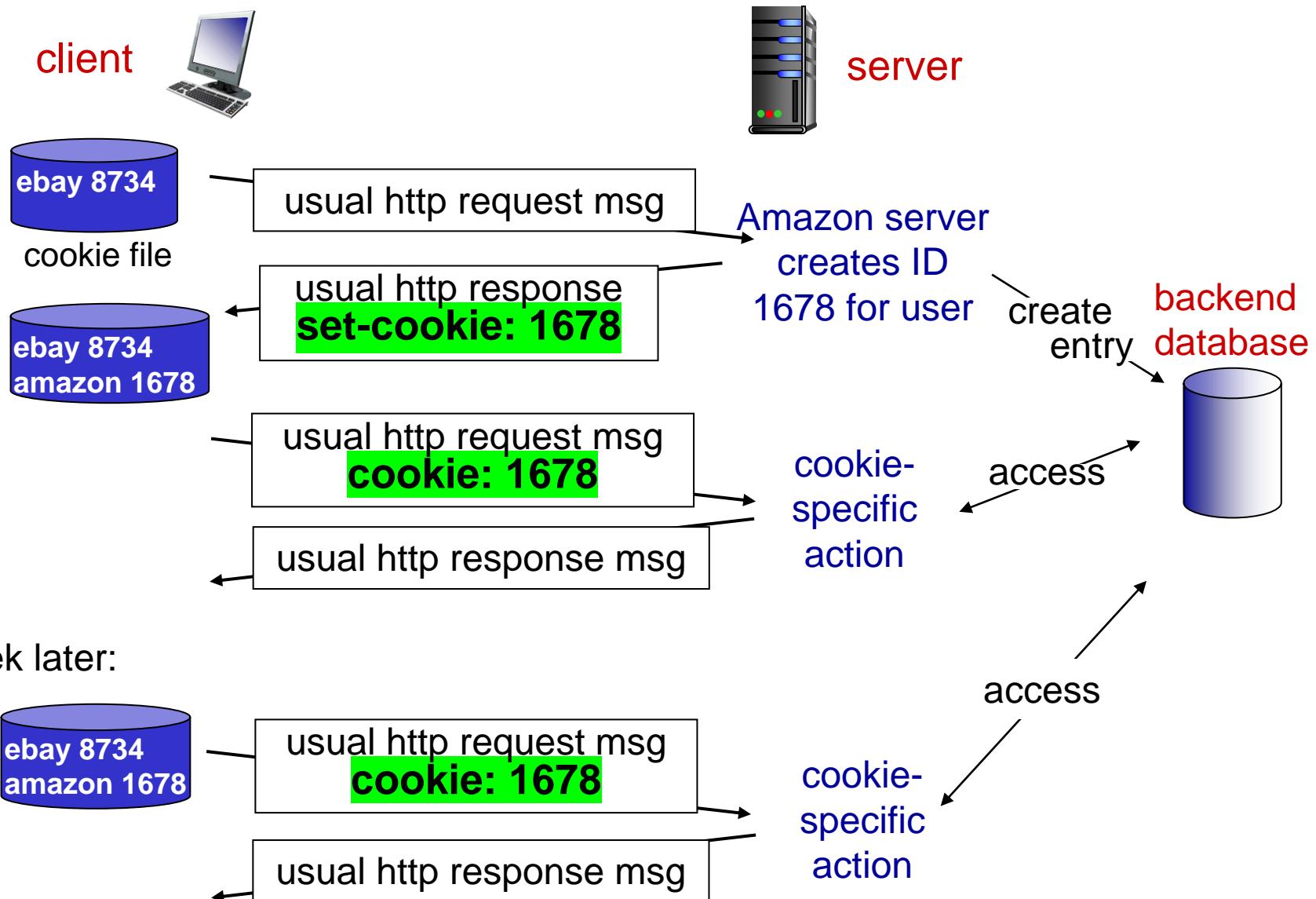
*four components:*

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

**example:**

- Susan always access Internet from PC
- visits specific e-commerce site for first time
- when initial HTTP requests arrives at site, site creates:
  - unique ID
  - entry in backend database for ID

# Cookies: keeping “state” (cont.)



# Cookies (continued)

*what cookies can be used for:*

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



aside

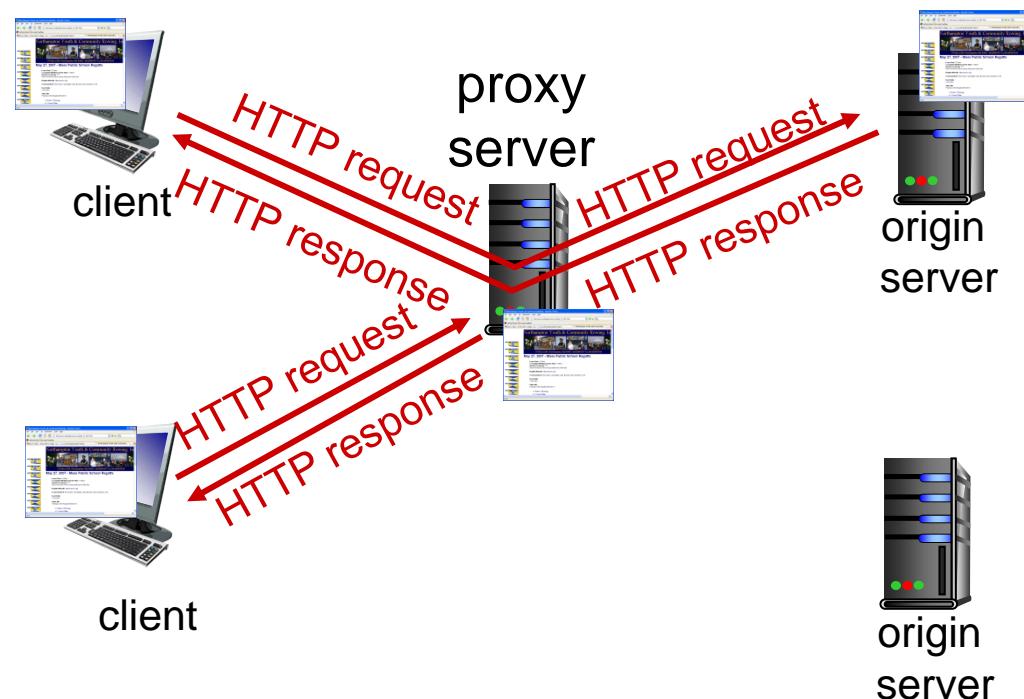
*cookies and privacy:*

- cookies permit sites to learn a lot about you
- you may supply name and e-mail to sites

# 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



# More about Web caching

- cache acts as both client and server
  - server for original requesting client
  - client to origin server
- typically cache is installed by ISP (university, company, residential ISP)

# More about Web caching

## *why Web caching?*

- reduce response time for client request
- reduce traffic on an institution's access link
- Internet dense with caches: enables “poor” content providers to effectively deliver content (so too does P2P file sharing)

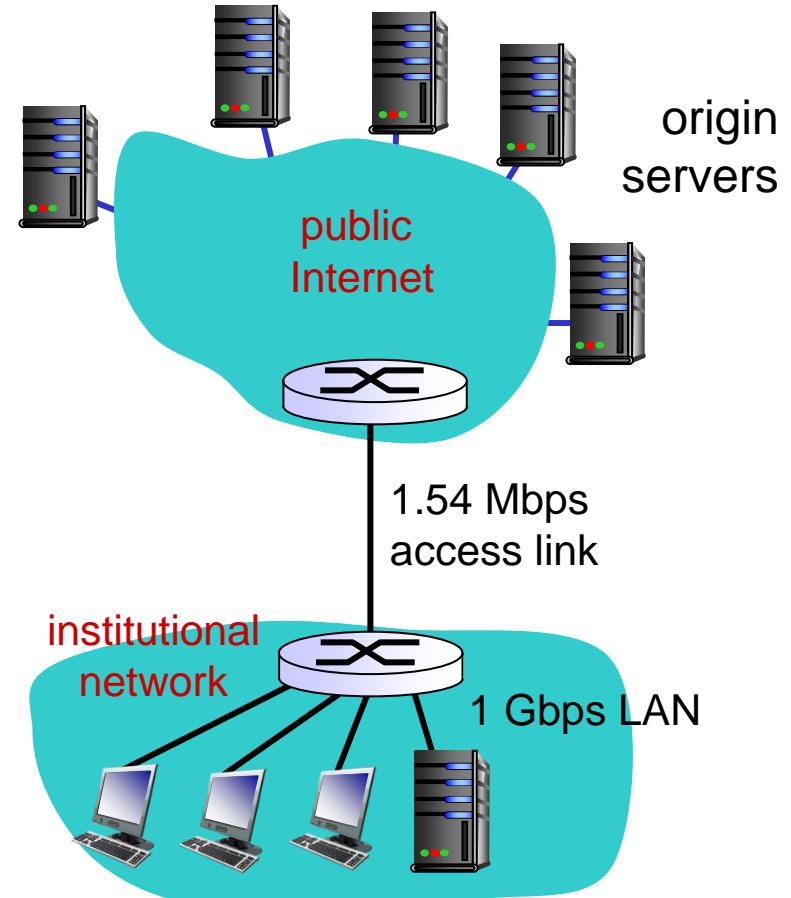
# Caching example:

## assumptions:

- avg object size: 100K bits
- avg request rate from browsers to origin servers: 15/sec
- avg data rate to browsers: 1.50 Mbps
- RTT from institutional router to any origin server: 2 sec
- access link rate: 1.54 Mbps

## consequences:

- LAN utilization: 15% *problem!*
- access link utilization = **99%**
- total delay = Internet delay + access delay + LAN delay  
= 2 sec + minutes + usecs



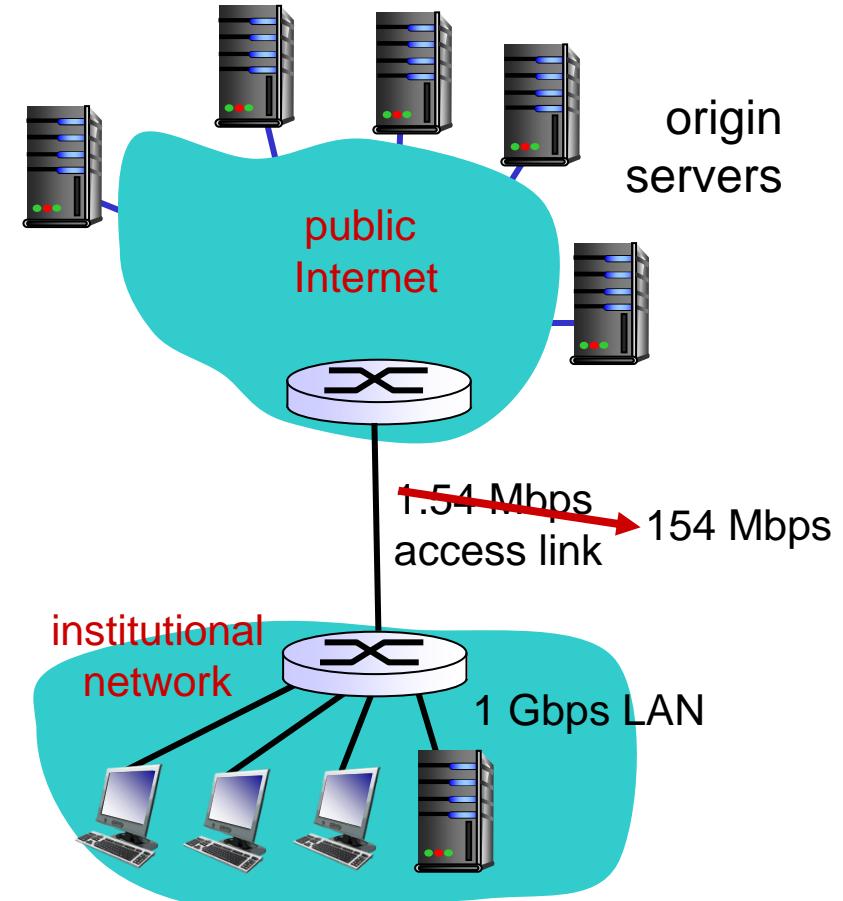
# Option I - fatter access link

## assumptions:

- avg object size: 100K bits
- avg request rate from browsers to origin servers: 15/sec
- avg data rate to browsers: 1.50 Mbps
- RTT from institutional router to any origin server: 2 sec
- access link rate: ~~1.54 Mbps~~  $\rightarrow$  154 Mbps

## consequences:

- LAN utilization: 15%
- access link utilization = ~~99%~~  $\rightarrow$  9.9%
- total delay = Internet delay + access delay + LAN delay  
 $= 2 \text{ sec} + \cancel{\text{minutes}} + \cancel{\text{usecs}} \rightarrow \text{msecs}$



**Cost:** increased access link speed (not cheap!)

# Option 2: install local cache

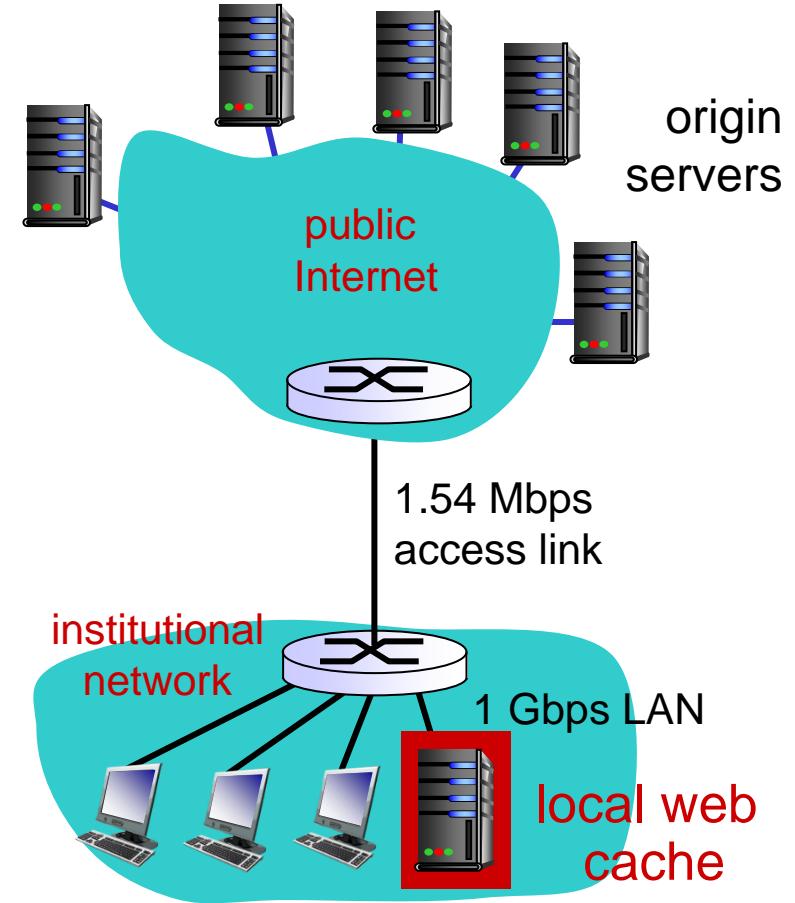
## assumptions:

- avg object size: 100K bits
- avg request rate from browsers to origin servers: 15/sec
- avg data rate to browsers: 1.50 Mbps
- RTT from institutional router to any origin server: 2 sec
- access link rate: 1.54 Mbps

## consequences:

- LAN utilization: 15%
- access link utilization = ?
- total delay = ?

*How to compute link utilization, delay?*

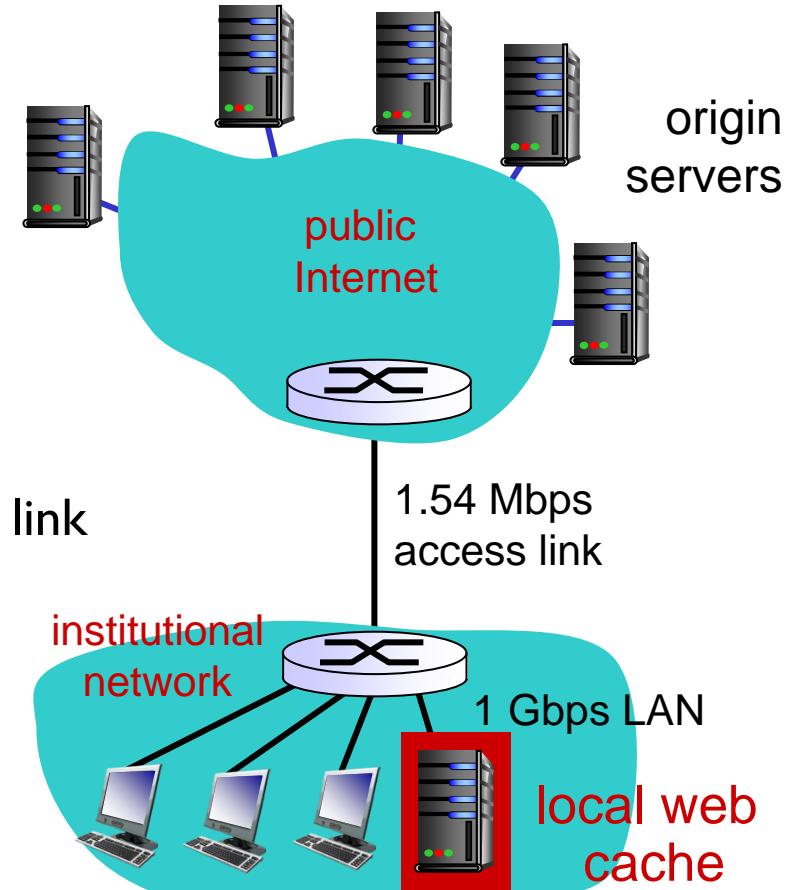


*Cost:* web cache (cheap!)

# Option 2: install local cache

*Calculating access link utilization, delay with cache:*

- suppose cache hit rate is 0.4
  - 40% requests satisfied at cache, 60% requests satisfied at origin
- access link utilization:
  - 60% of requests use access link
- data rate to browsers over access link  
 $= 0.6 * 1.50 \text{ Mbps} = .9 \text{ Mbps}$ 
  - utilization =  $0.9 / 1.54 = .58$
- total delay
  - $= 0.6 * (\text{delay from origin servers}) + 0.4 * (\text{delay when satisfied at cache})$
  - $= 0.6 (2.01) + 0.4 (\sim \text{msecs}) = \sim 1.2 \text{ secs}$
  - less than with 154 Mbps link (and cheaper too!)



# Chapter Outline

1 principles of network applications

2 Web and HTTP

3 DNS

# DNS: domain name system

*people*: many identifiers:

- SSN, name, passport #

*Internet hosts, routers*:

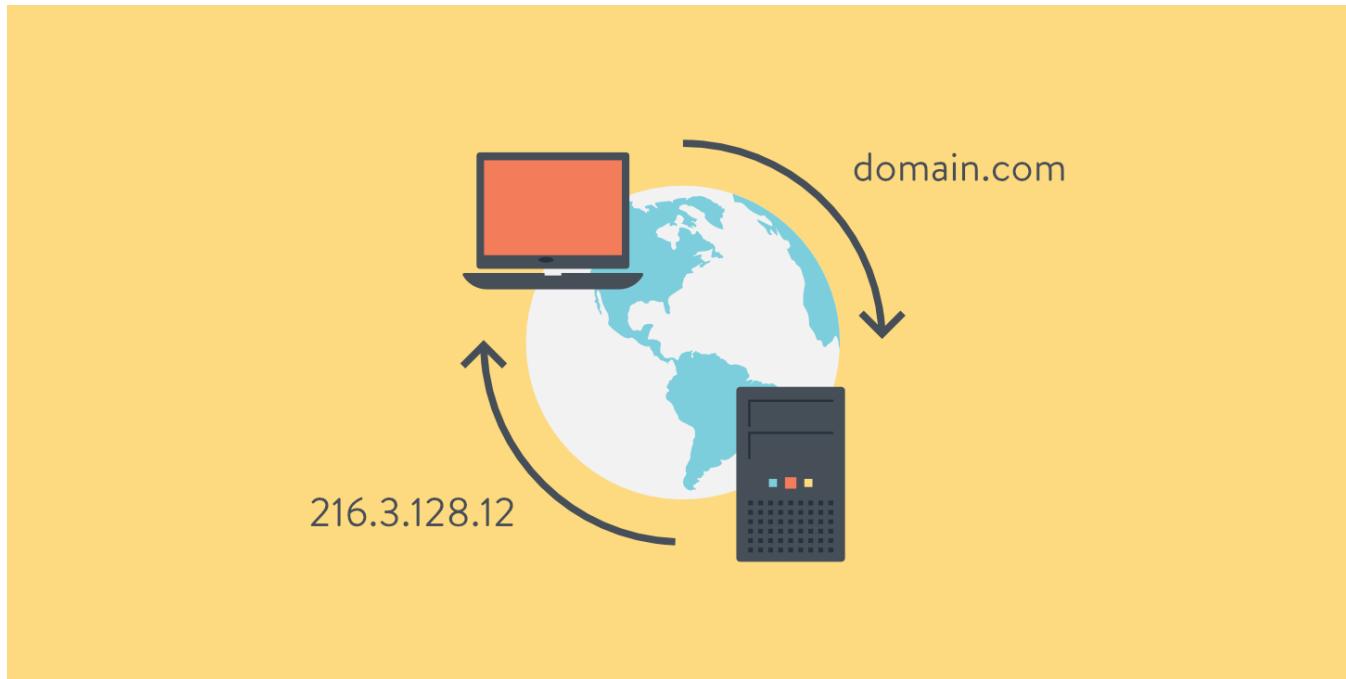
- “name”, e.g., `www.youtube.com` - used by humans
- IP address (32 bit) - used for addressing datagrams

e.g., `121.7.106.83`

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

# DNS: domain name system

DNS: map between IP address and name



# DNS: domain name system

## *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: services

## *DNS services*

- hostname to IP address translation
- host aliasing
  - canonical, alias names
- mail server aliasing
- load distribution
  - replicated Web servers: many IP addresses correspond to one name

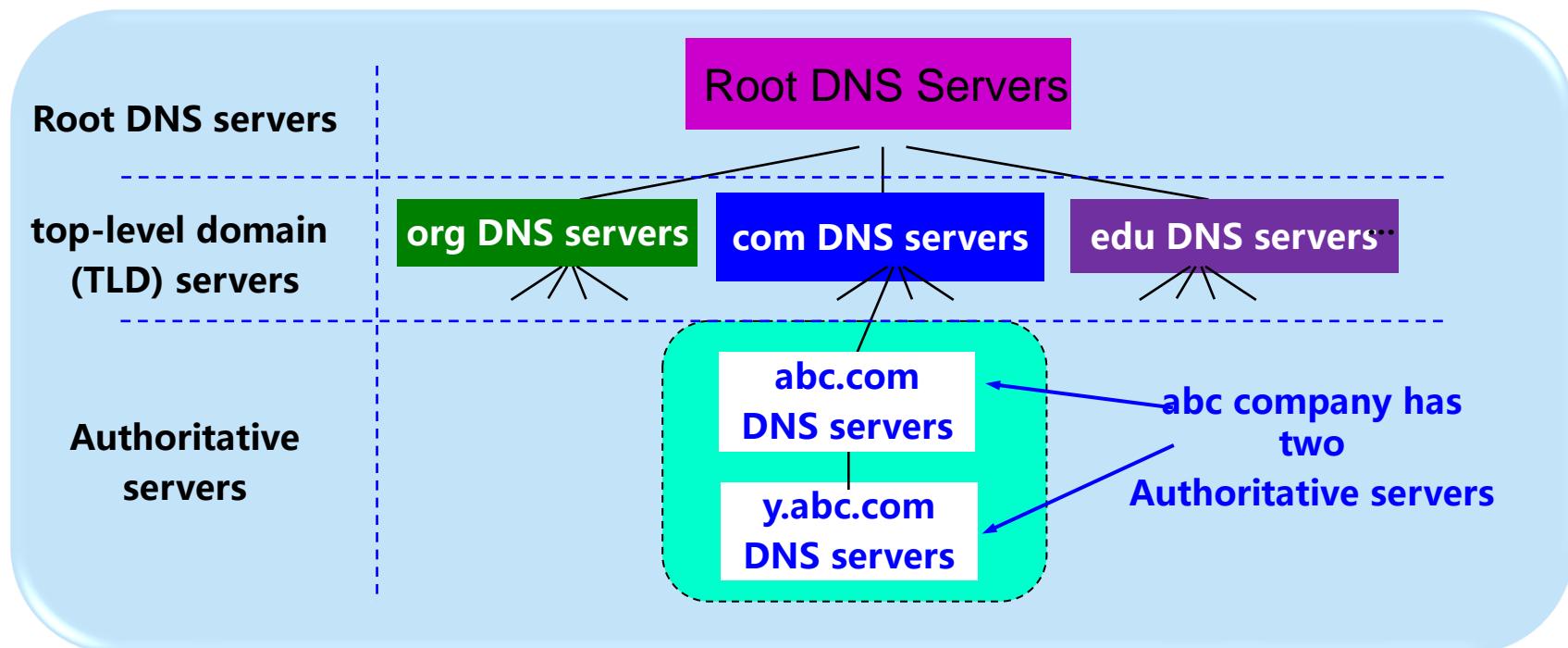
# DNS: structure

*why not centralize DNS?*

- single point of failure
- traffic volume
- distant centralized database
- maintenance

*A: doesn't scale!*

# DNS: a distributed, hierarchical database





# Computer Networks

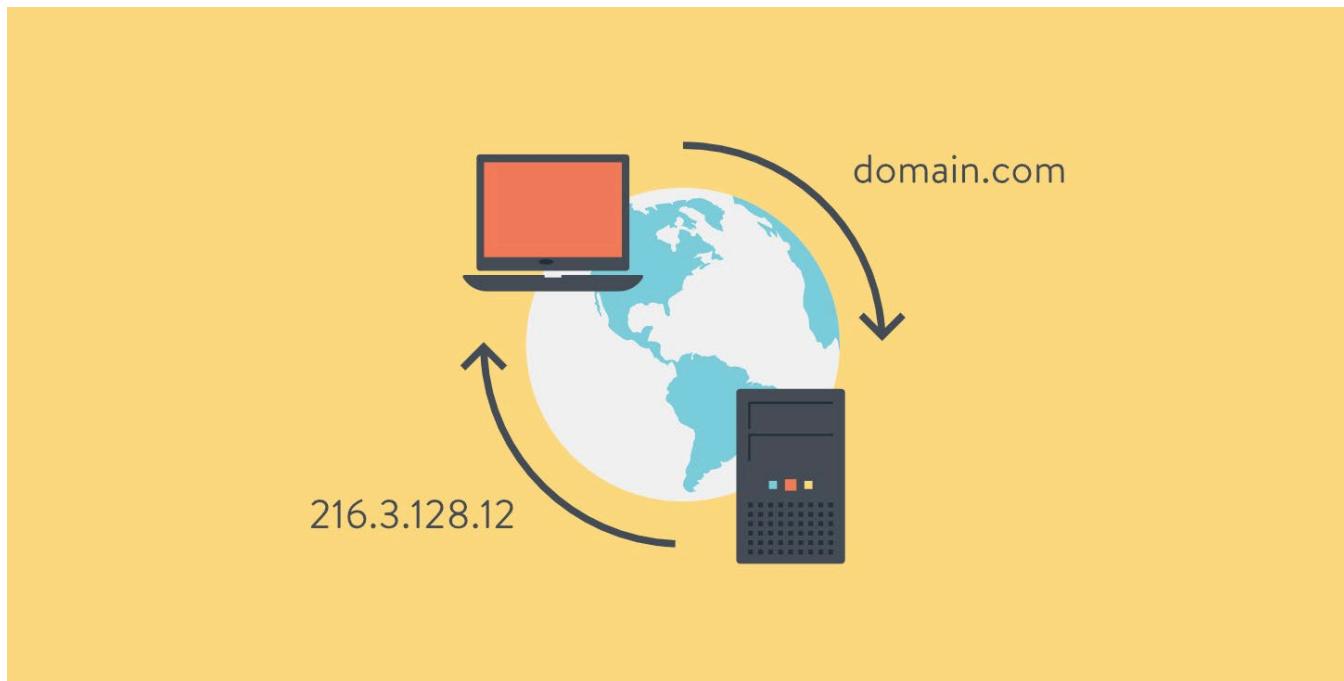
Lecturer: ZHANG Ying  
Fall semester 2022

# Chapter 2

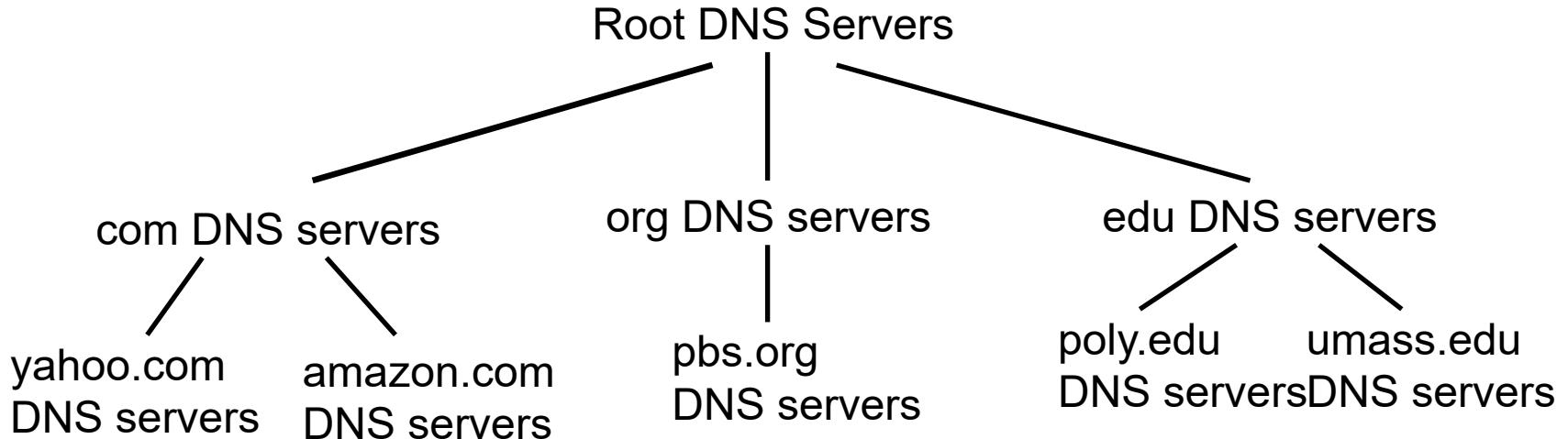
# Application Layer

# DNS: domain name system

DNS: map between IP address and name



# DNS: a distributed, hierarchical database



*client wants IP for **www.amazon.com**;*

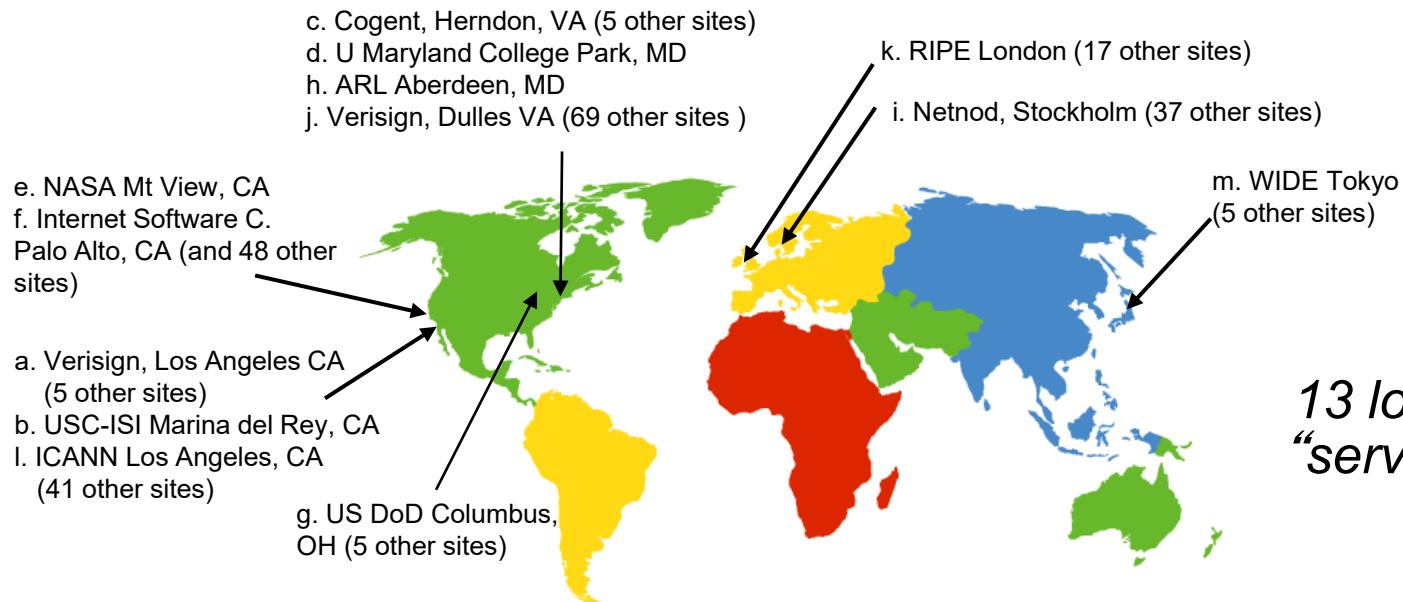
- client queries **root** server to find **.com** DNS server
- client queries **.com** DNS server to get **amazon.com** DNS server
- client queries **amazon.com** DNS server to get **IP address** for **www.amazon.com**

# 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

# root DNS name servers

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

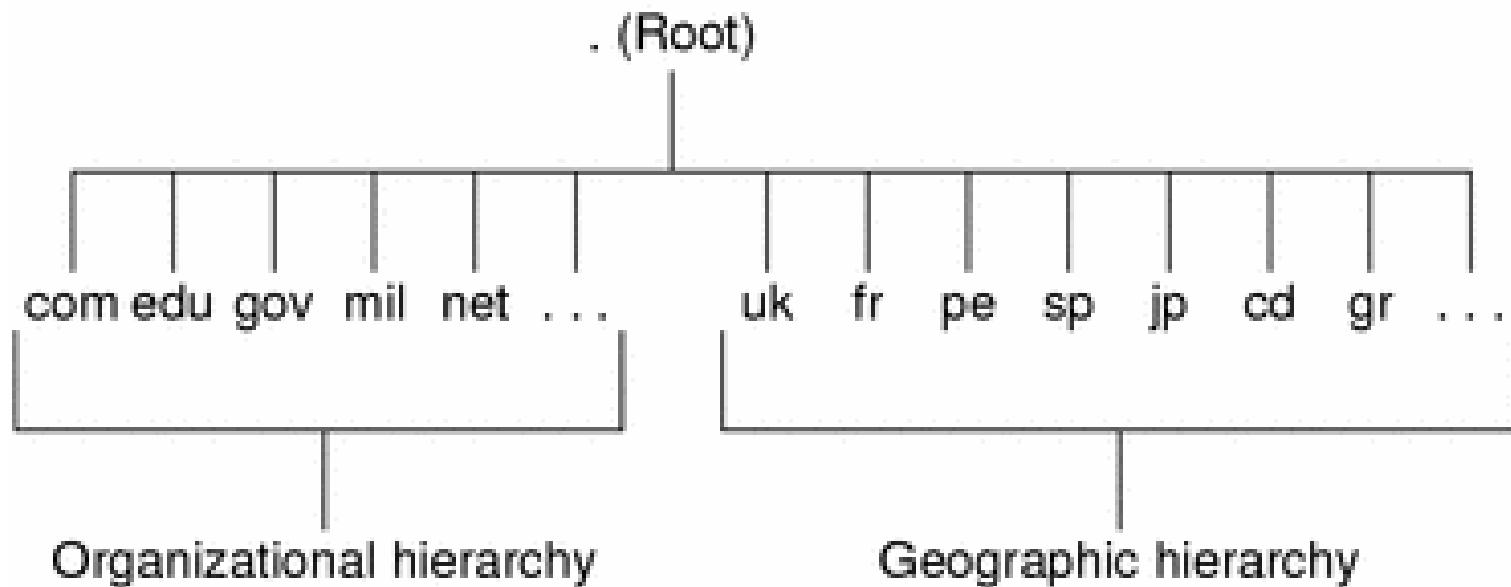


*13 logical root name  
“servers” worldwide*

# top-level domain (TLD) servers:

Divided into two distinct sub-categories:

- Organizational Hierarchy
- Geographical Hierarchy



# Organizational Hierarchy

## Domain Purpose

- .com commercial organizations
- .edu educational organizations
- .gov government institutions
- .mil military groups
- .net major network support centers
- .org Nonprofit organizations and others
- .int International organizations

# authoritative DNS servers

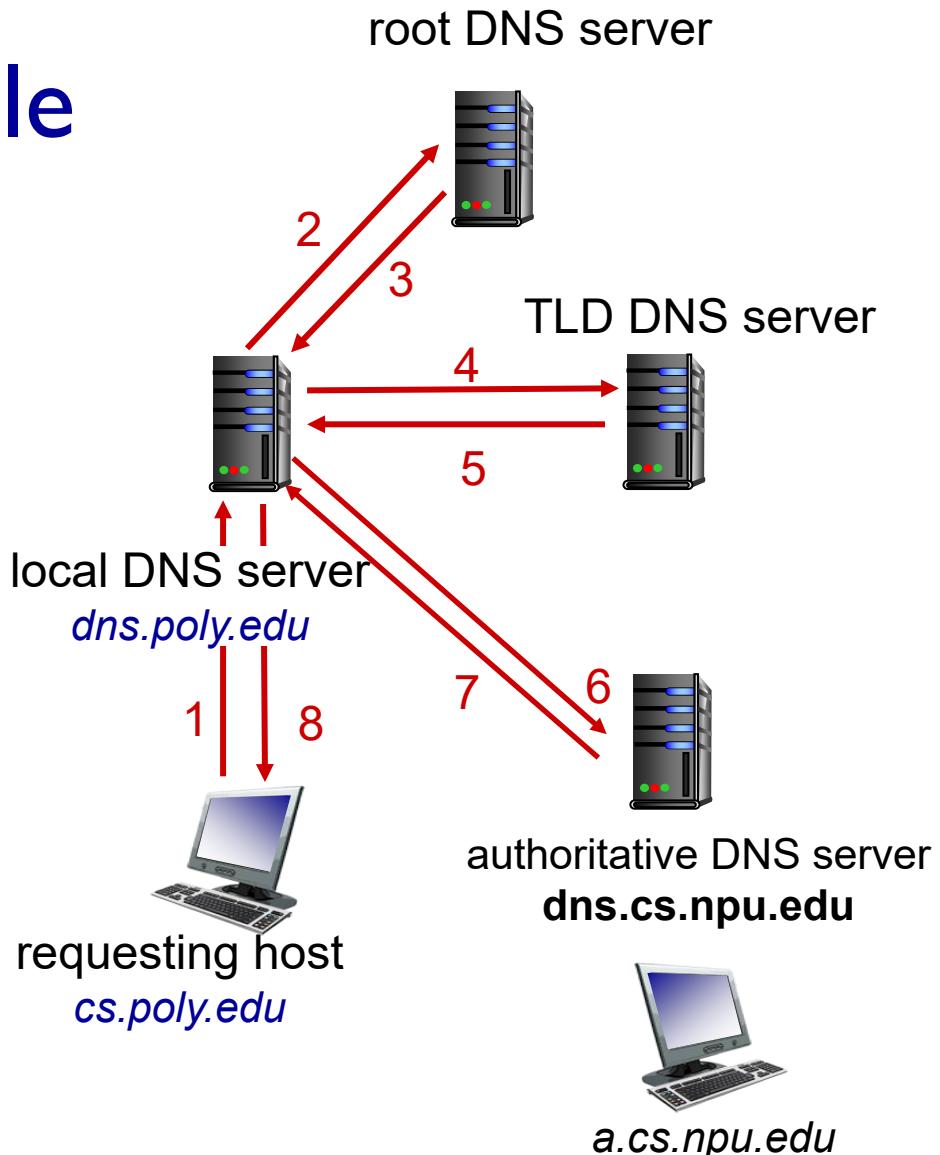
- organization's own DNS server(s), providing authoritative hostname to IP mappings for organization's named hosts
- can be maintained by organization or service provider

# DNS name resolution example

- host at `cs.poly.edu` wants IP address for `a.cs.npu.edu`

## *iterated query:*

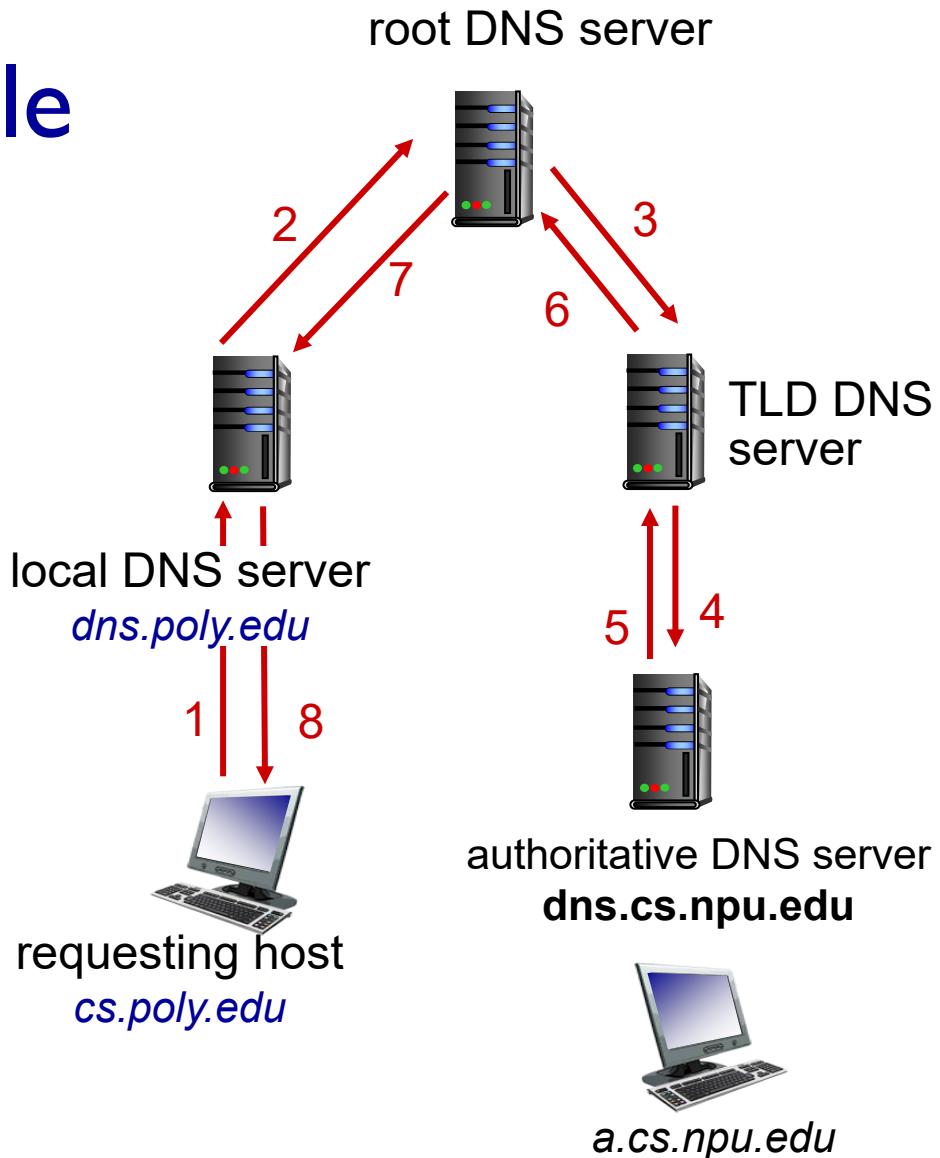
- contacted server replies with name of server to contact
- “I don’t know this name, but ask this server”



# DNS name resolution example

*recursive query:*

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



# DNS: caching, updating records

- once (any) name server learns mapping, it *caches* mapping
  - cache entries timeout (disappear) after some time (TTL)
  - TLD servers typically cached in local name servers
    - thus root name servers not often visited
- cached entries may be *out-of-date* (best effort name-to-address translation!)
  - if name host changes IP address, may not be known Internet-wide until all TTLs expire

# 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

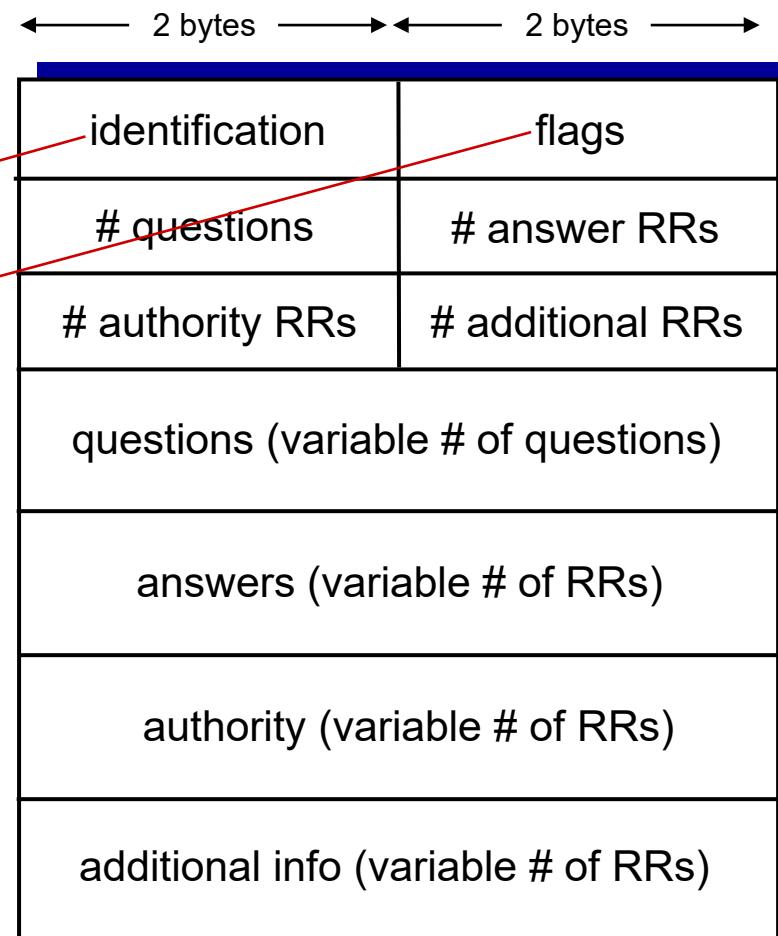
- **value** is name of mailserver  
associated with **name**

# DNS protocol, messages

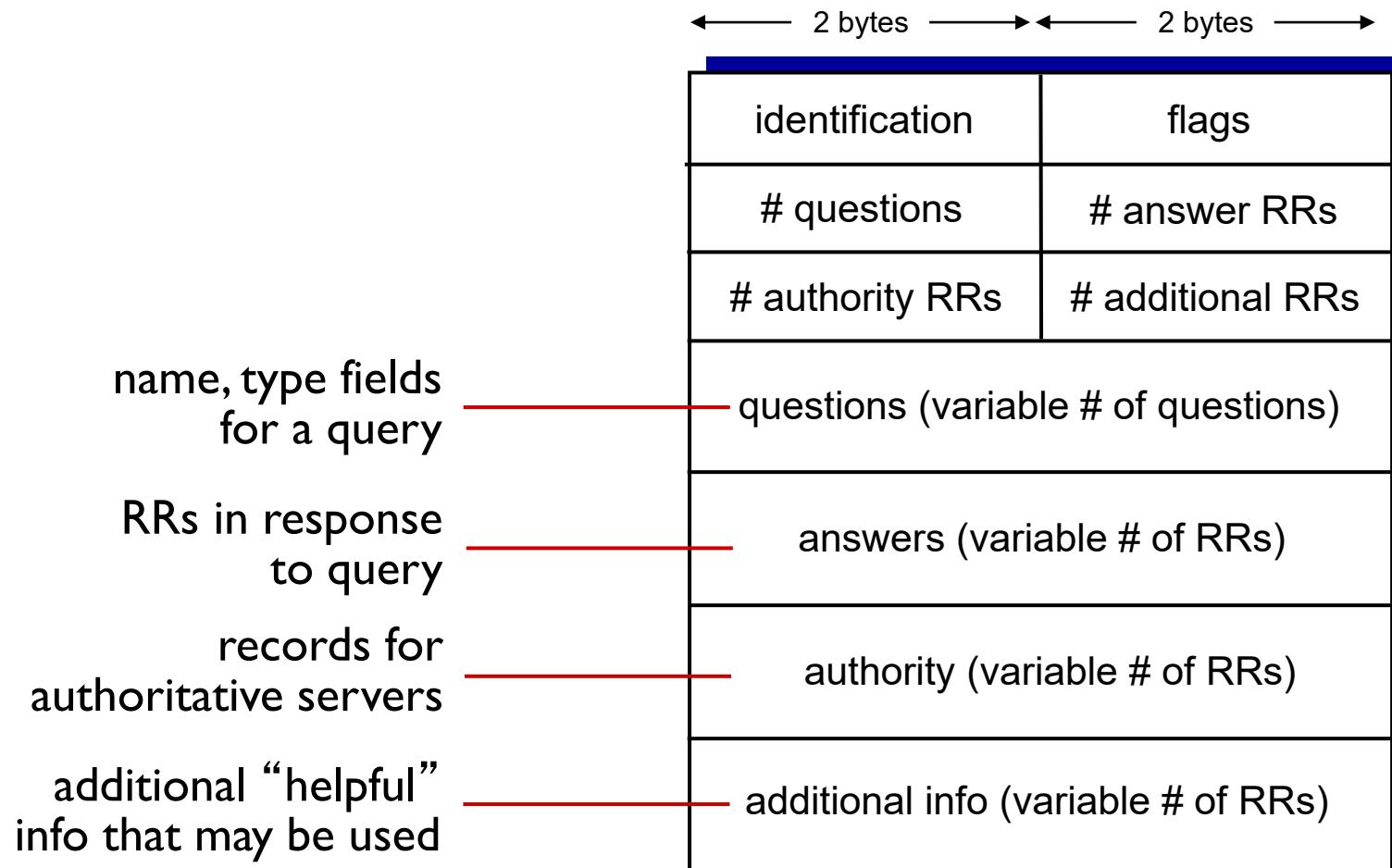
- *query* and *reply* messages, both with same *message format*

message header

- identification: 16 bit # for query, reply to query uses same #
- flags:
  - query or reply
  - recursion desired
  - recursion available
  - reply is authoritative



# DNS protocol, messages



# Chapter Outline

1

principles of network applications

2

Web and HTTP

3

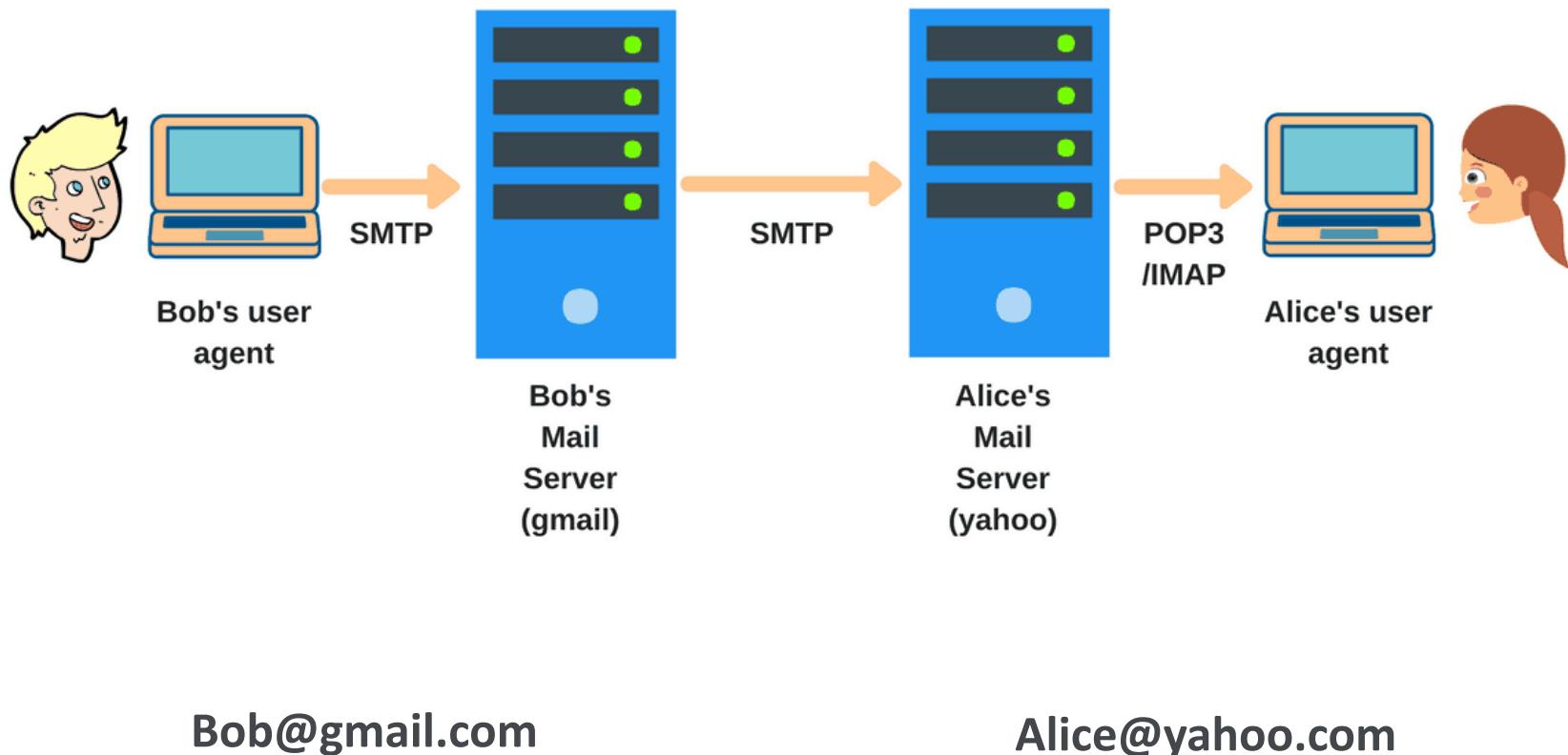
electronic mail

# Electronic mail

- **Electronic mail (email or e-mail)** is a method of exchanging messages ("mail") between people using electronic devices.
- A-synchronous
- Inexpensive
- Rich contents



# How email works – example



# Electronic mail

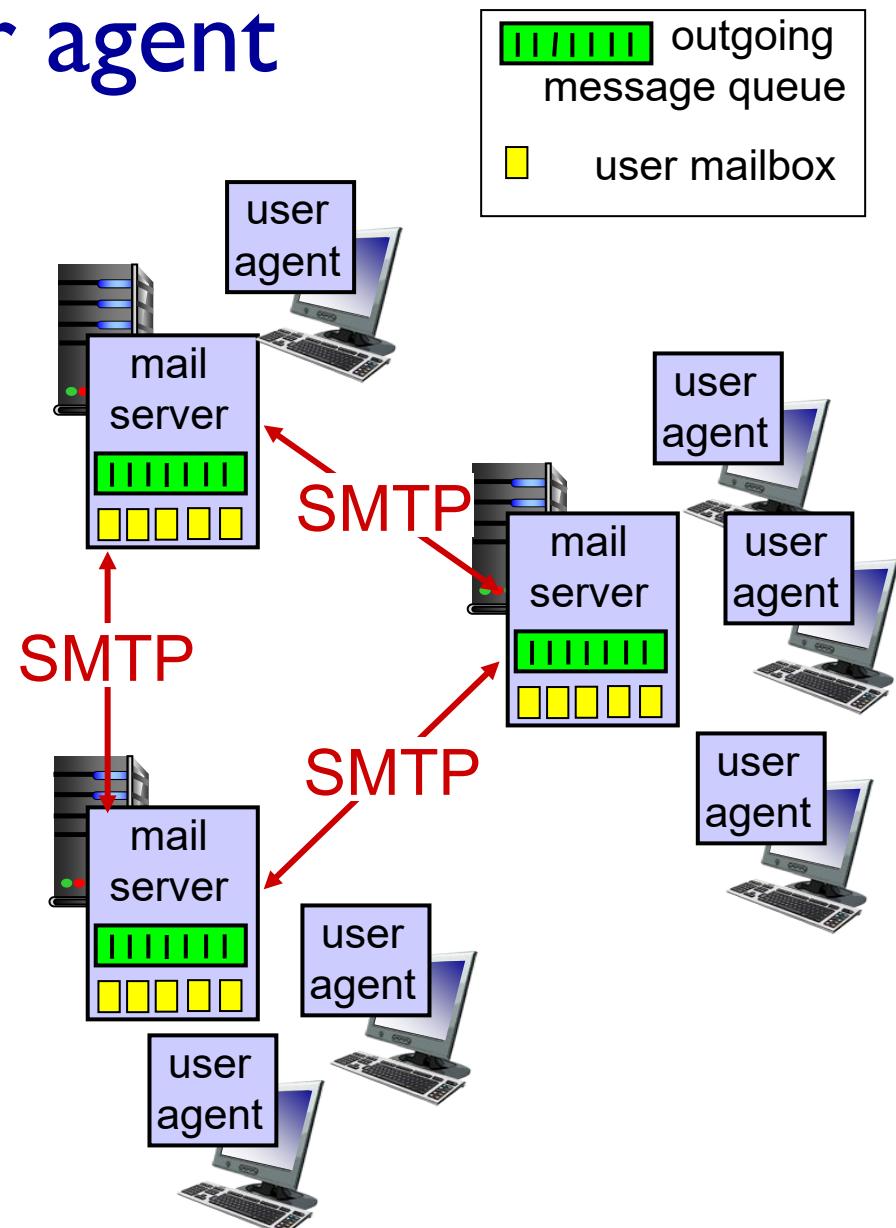
*Major components:*

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

# Electronic mail: user agent

## User Agent

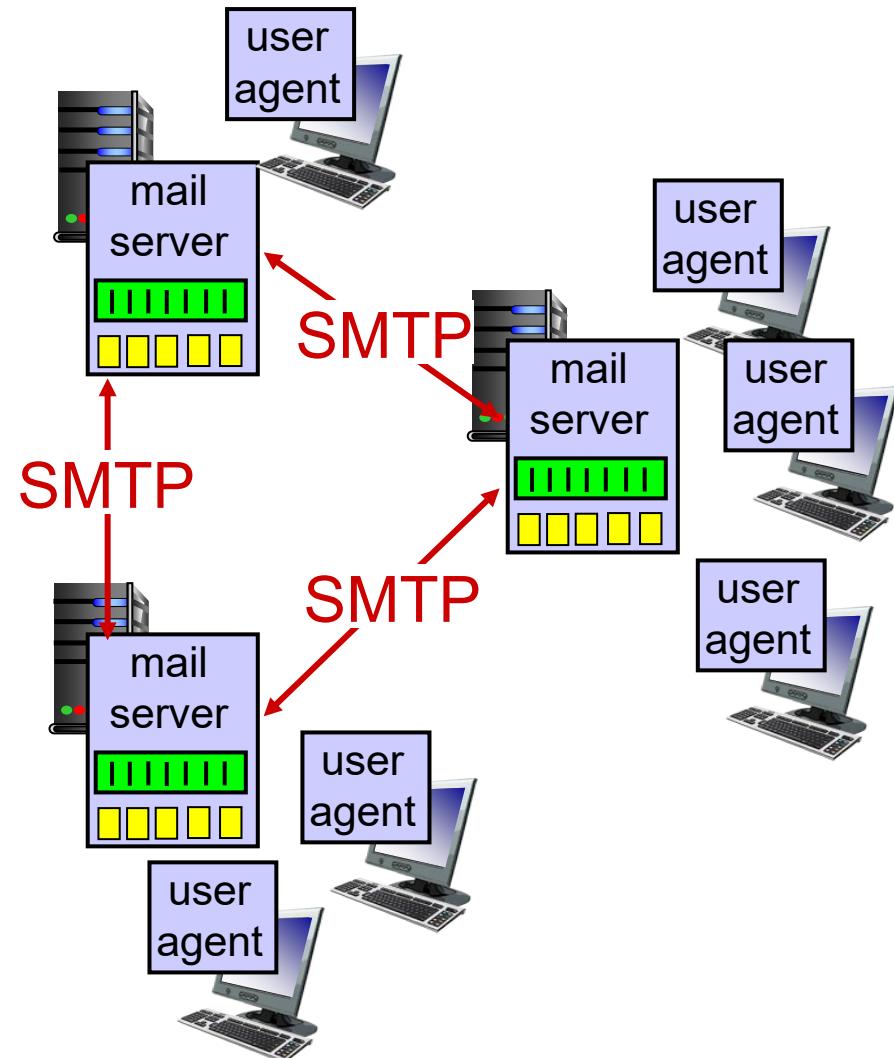
- a.k.a. “mail reader”
- composing, editing, reading mail messages
- e.g., Outlook, iPhone mail client
- outgoing, incoming messages stored on server



# Electronic mail: mail servers

## mail servers:

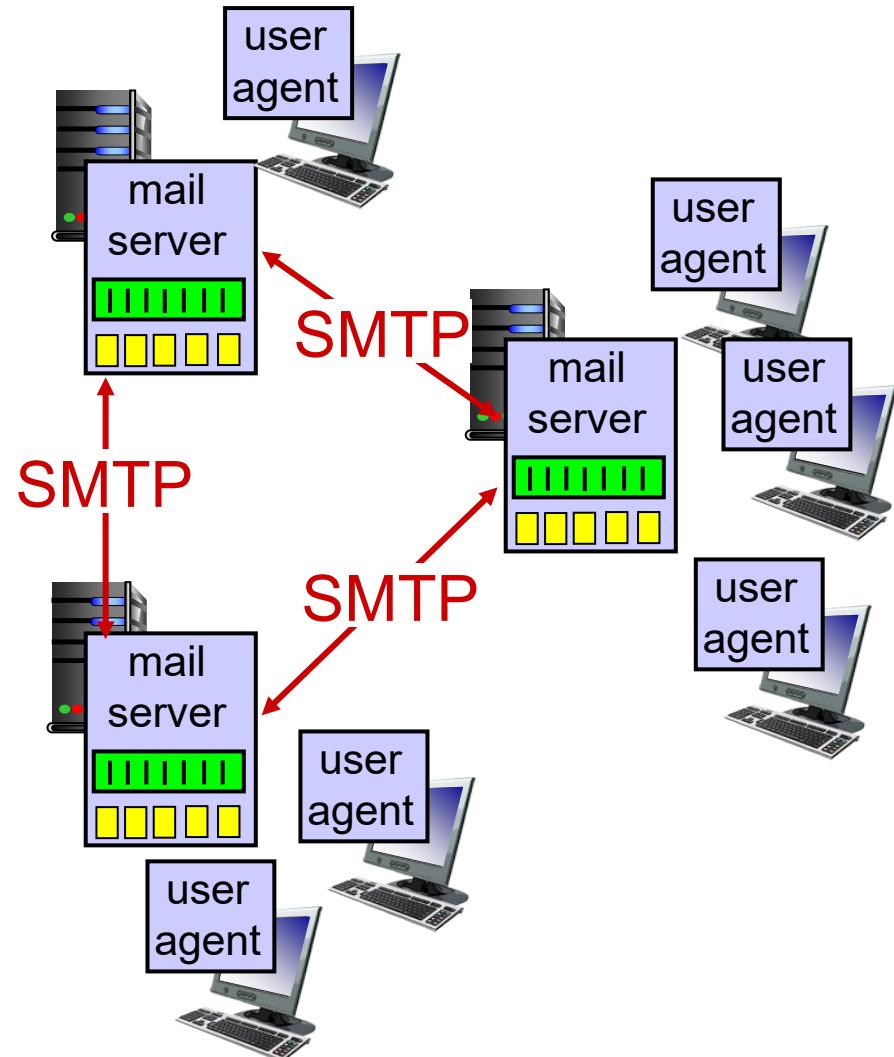
- *mailbox* contains incoming messages for user
- *message queue* of outgoing (to be sent) mail messages



# Electronic mail: SMTP protocol

*SMTP protocol* between mail servers to send email messages

- client: sending mail server
- “server”: receiving mail server



# Electronic Mail: SMTP [RFC 2821]

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

# Sample SMTP interaction

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

# SMTP vs HTTP

## SMTP

- SMTP: push
- SMTP: multiple objects sent in multipart message

## HTTP

- HTTP: pull
- HTTP: each object encapsulated in its own response message

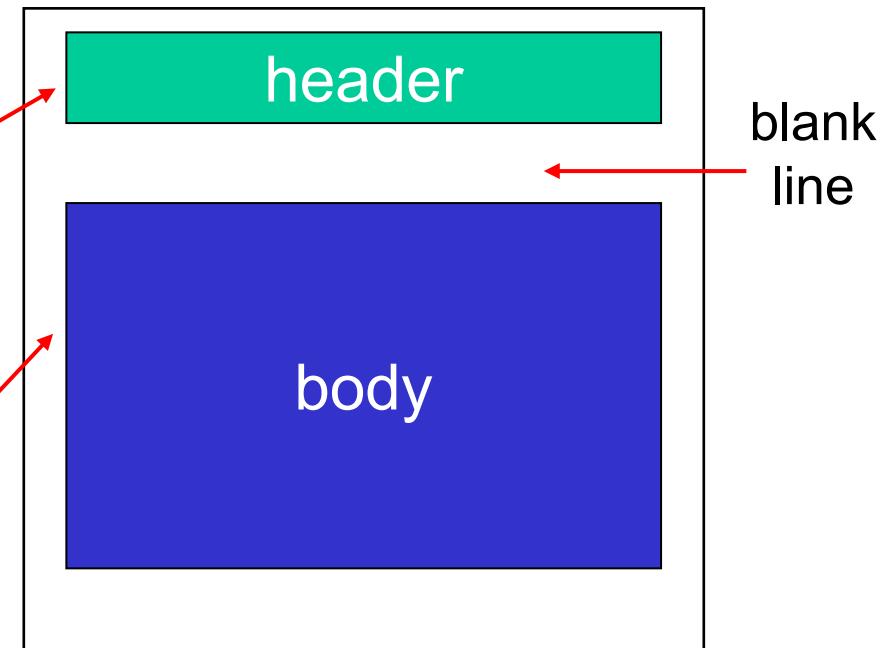
Common: both have ASCII command/response interaction, status codes (but may have different format requirement)

# Mail message format

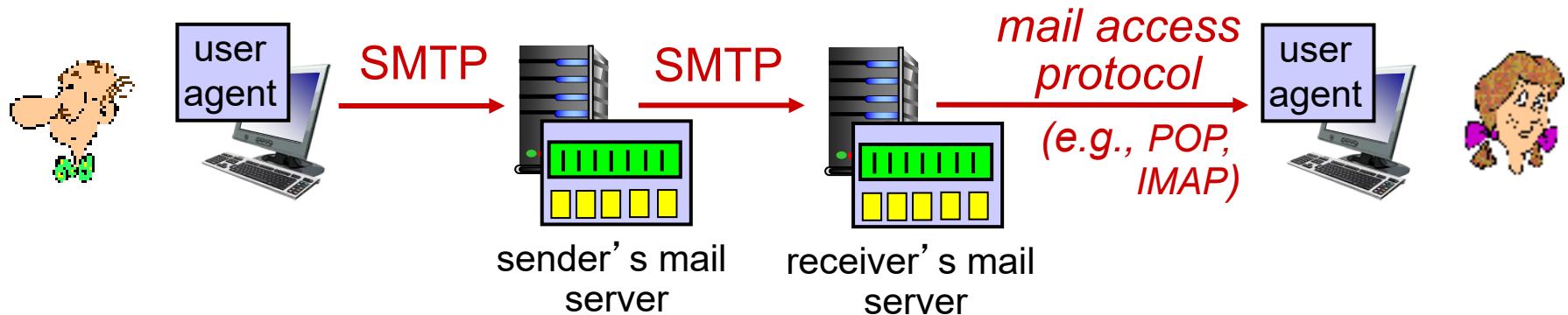
SMTP: protocol for  
exchanging email messages

RFC 822: standard for text  
message format:

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

# POP3 protocol

Three main phases: authorization; transaction; update

## *authorization phase*

- client commands:
  - **user**: declare username
  - **pass**: password
- server responses
  - +OK
  - -ERR

```
S: +OK POP3 server ready
C: user bob
S: +OK
C: pass hungry
S: +OK user successfully logged on
```

# POP3 protocol

## *transaction phase*, client:

- **list**: list message numbers
- **retr**: retrieve message by number
- **dele**: delete
- **Quit**

## Two modes:

- Download and delete
  - Bob cannot re-read email if he changes client
- Download and keep
  - copies of messages on different clients

```
C: list
S: 1 498
S: 2 912
S: .
C: retr 1
S: <message 1 contents>
S: .
C: dele 1
C: retr 2
S: <message 1 contents>
S: .
C: dele 2
C: quit
S: +OK POP3 server signing off
```

# IMAP

- Another mail access protocol
- IMAP keeps all messages in one place: at server
- allows user to organize messages in folders
- keeps user state across sessions:
  - names of folders and mappings between message IDs and folder name

*IMAP has many more features than POP3, but it is also significantly more complex*

# Chapter Outline

1

principles of network applications

2

Web and HTTP

3

DNS

4

electronic mail

5

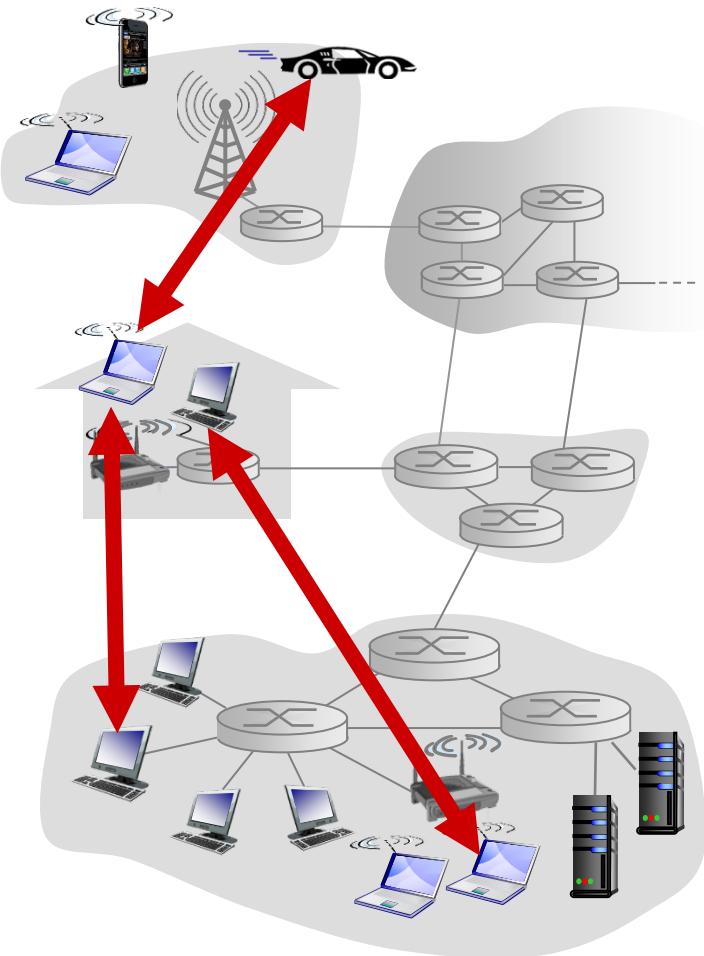
P2P applications

# Pure P2P architecture

- no always-on server
- arbitrary end systems directly communicate
- peers are intermittently connected and change IP addresses

## examples:

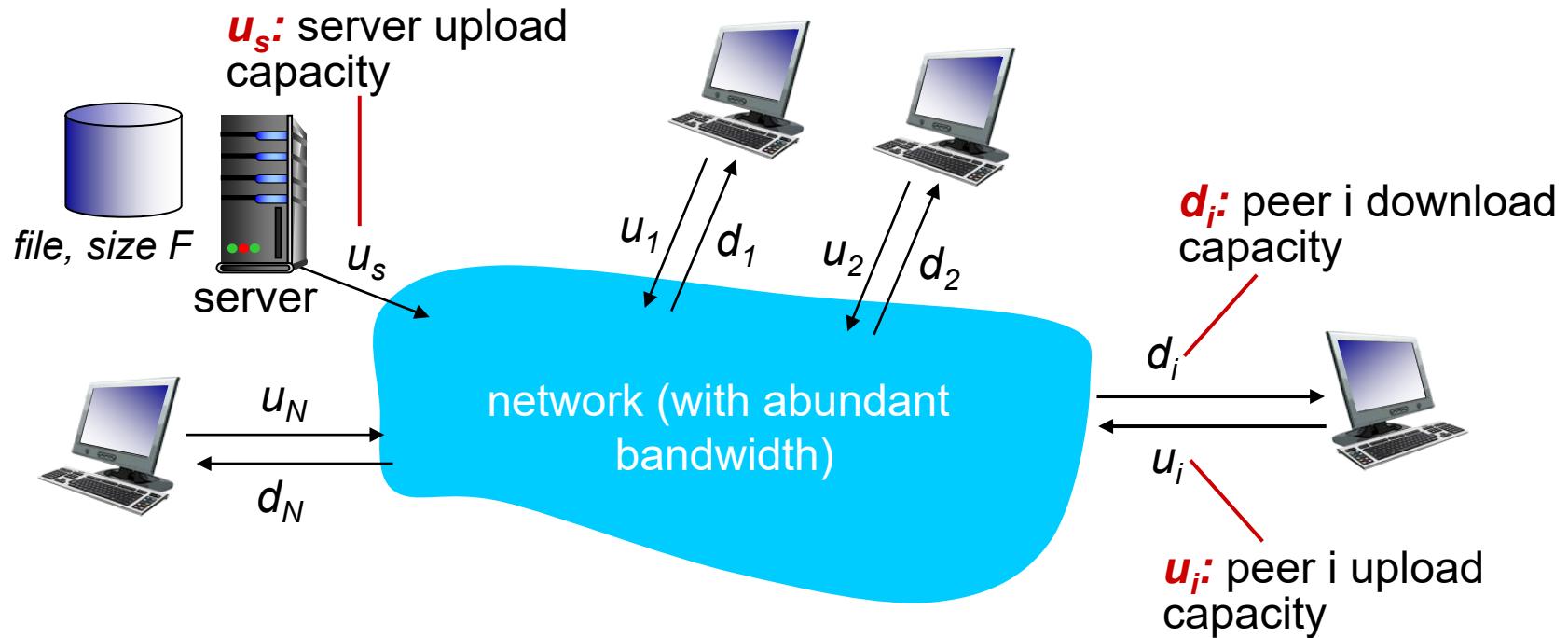
- file distribution (BitTorrent)
- Streaming (KanKan)
- VoIP (Skype)



# File distribution: client-server vs P2P

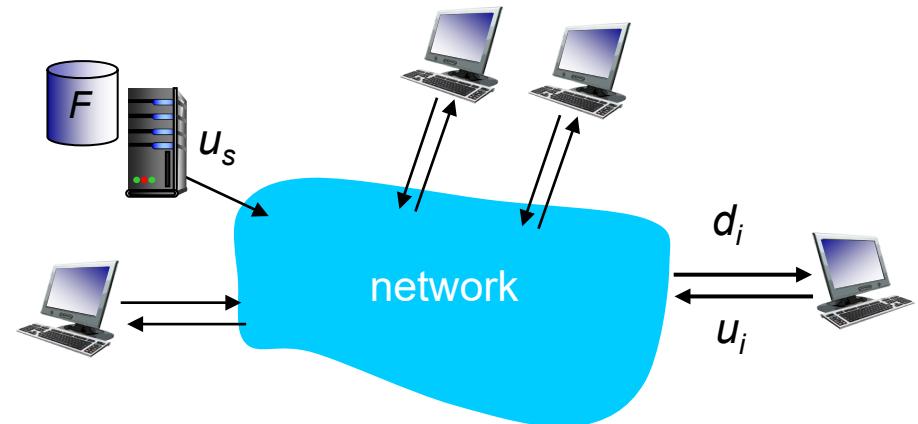
Question: how much time to distribute file (size  $F$ ) from one server to  $N$  peers?

- peer upload/download capacity is limited resource



# File distribution time: client-server

- **server transmission:** must sequentially send (upload)  $N$  file copies:
  - time to send one copy:  $F/u_s$
  - time to send  $N$  copies:  $NF/u_s$



- **client:** each client must download file copy
  - $d_{min}$  = min client download rate
  - min client download time:  $F/d_{min}$

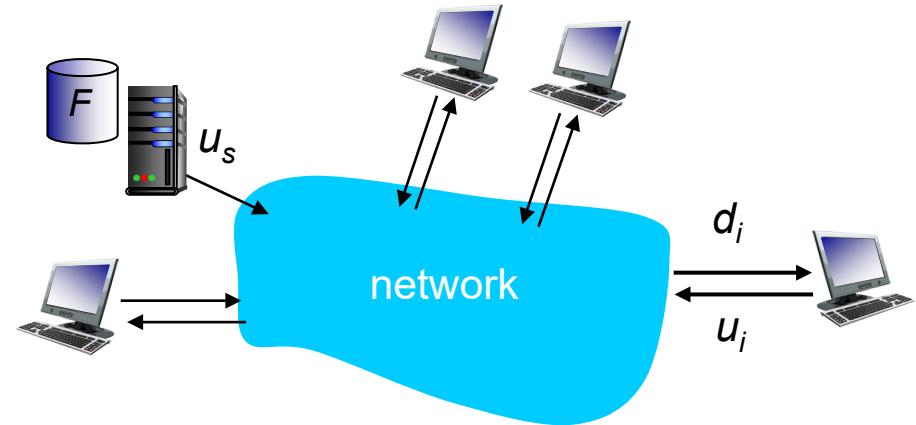
*time to distribute  $F$   
to  $N$  clients using  
client-server approach*

$$D_{c-s} \geq \max\{NF/u_s, F/d_{min}\}$$

increases linearly in  $N$

# File distribution time: P2P

- **server transmission:** must upload at least one copy
  - time to send one copy:  $F/u_s$



- **client:** each client must download file copy
  - min client download time:  $F/d_{\min}$

- **clients:** as aggregate must download  $NF$  bits
  - max upload rate (limiting max download rate) is  $u_s + \sum u_i$

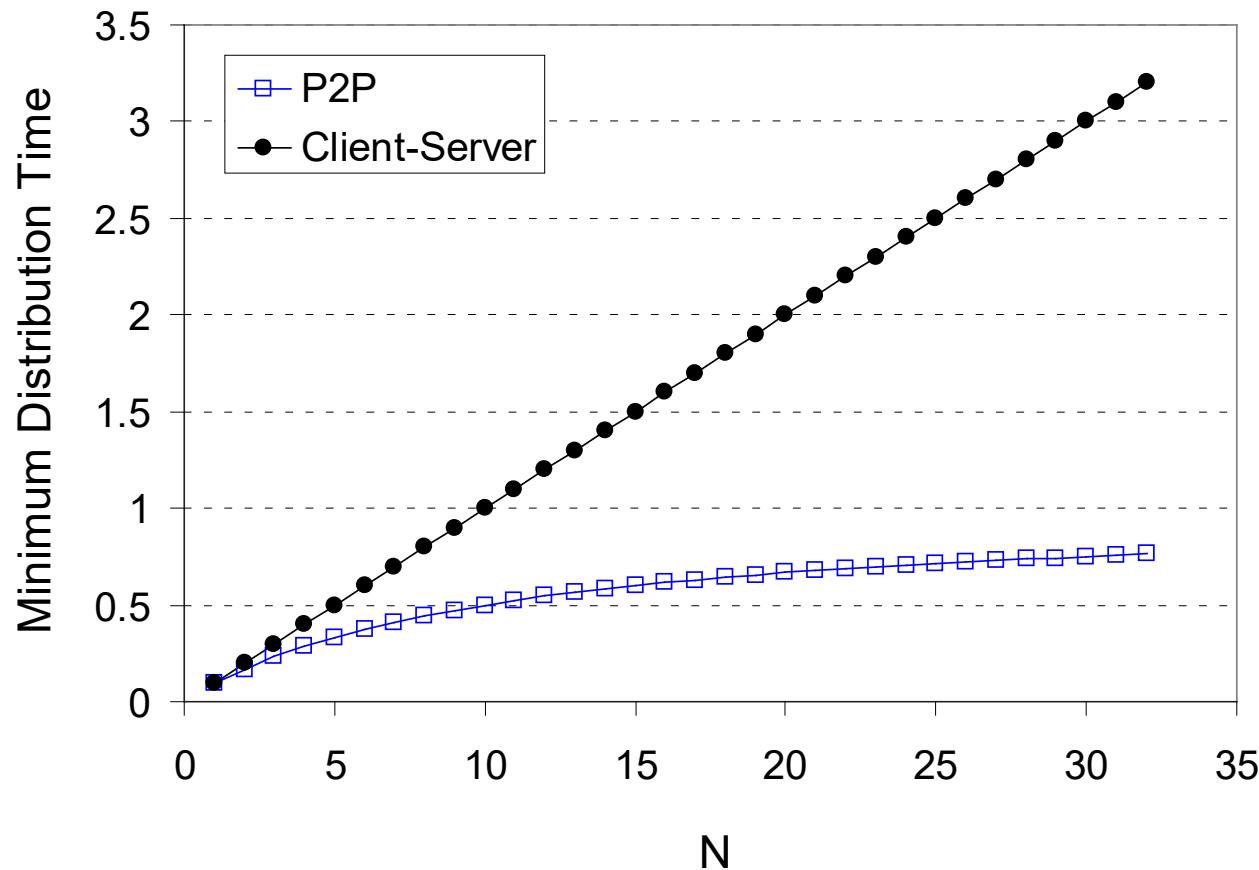
time to distribute  $F$   
to  $N$  clients using  
P2P approach

$$D_{P2P} \geq \max\{F/u_s, F/d_{\min}, NF/(u_s + \sum u_i)\}$$

increases linearly in  $N$  ...  
... but so does this, as each peer brings service capacity

# Client-server vs. P2P: example

client upload rate =  $u$ ,  $F/u = 1$  hour,  $u_s = 10u$ ,  $d_{min} \geq u_s$

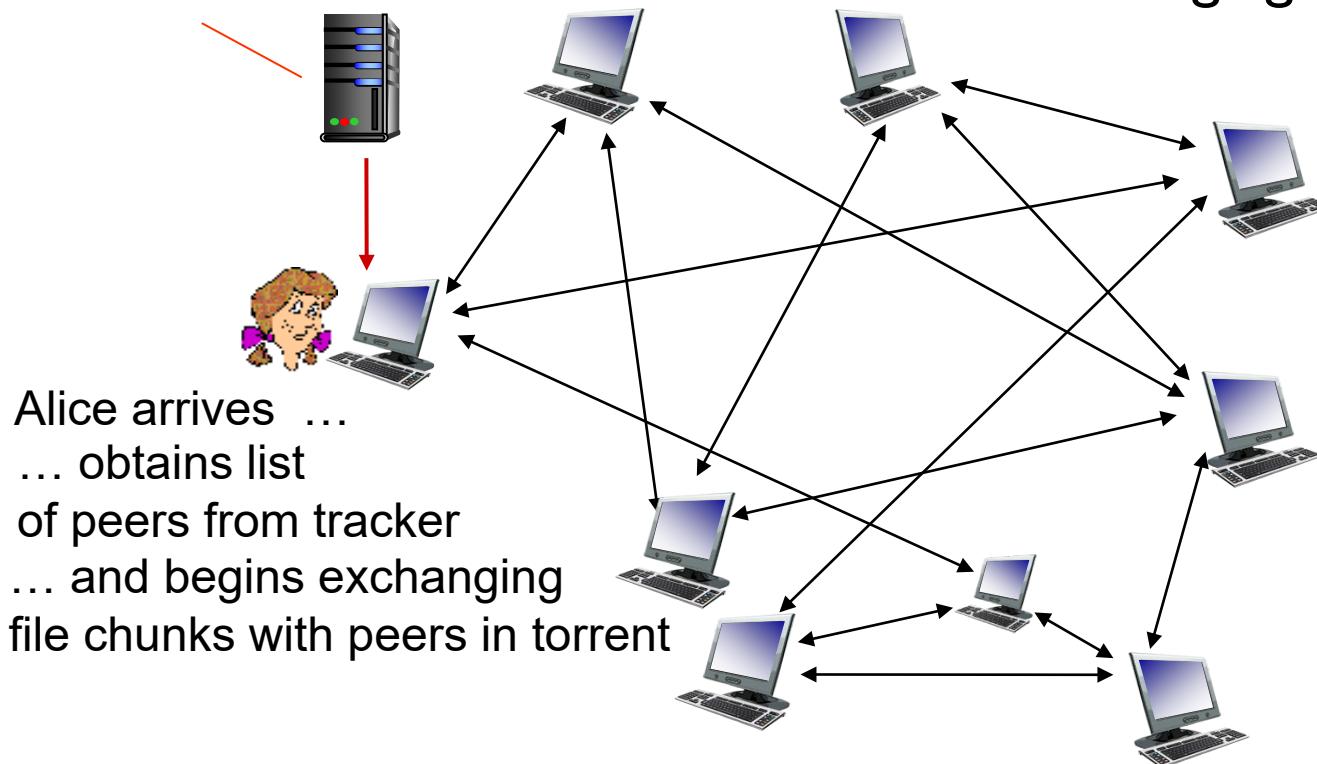


# P2P file distribution: BitTorrent

- file divided into 256Kb chunks
- peers in torrent send/receive file chunks

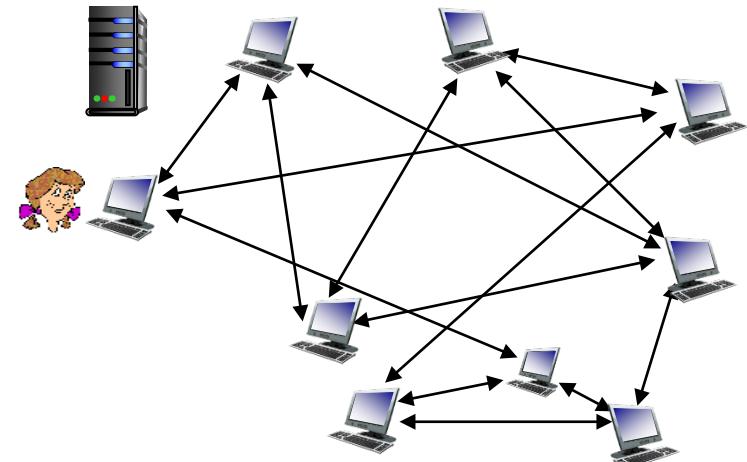
*tracker*: tracks peers  
participating in torrent

*torrent*: group of peers  
exchanging chunks of a file



# P2P file distribution: BitTorrent

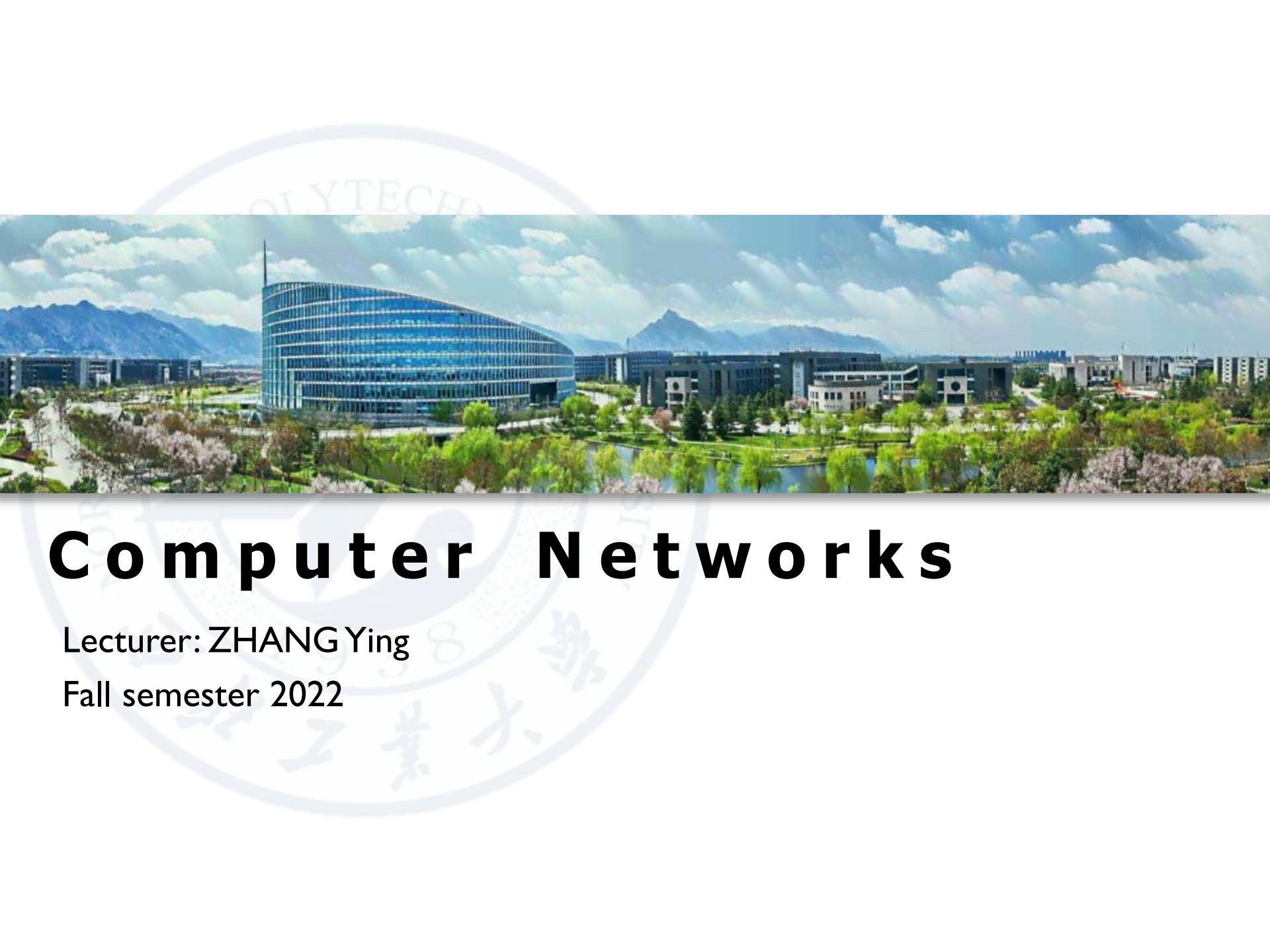
- peer joining torrent:
  - has no chunks, but will accumulate them over time from other peers
  - registers with tracker to get list of peers, connects to subset of peers (“neighbors”)
- while downloading, peer uploads chunks to other peers
- peer may change peers with whom it exchanges chunks
- *churn*: peers may come and go
- once peer has entire file, it may (selfishly) leave or (altruistically) remain in torrent



# BitTorrent: requesting, sending file chunks

## *requesting chunks:*

- at any given time, different peers have different subsets of file chunks
- periodically, Alice asks each peer for list of chunks that they have
- Alice requests missing chunks from peers, rarest first



# Computer Networks

Lecturer: ZHANG Ying

Fall semester 2022

# Chapter2 Outline

1

principles of network applications

2

Web and HTTP

3

DNS

4

electronic mail

5

P2P applications

6

Contents Distribution Network/CDN

# Video Streaming and CDNs: context

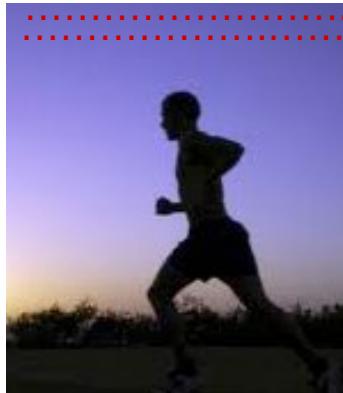
- video traffic: major consumer of Internet bandwidth
  - Netflix, YouTube, Amazon Prime: 80% of downstream residential ISP traffic (2020)
- challenge: scale - how to reach ~1B users?
  - single mega-video server won't work (why?)
- challenge: heterogeneity
  - different users have different capabilities (e.g., wired versus mobile; bandwidth rich versus bandwidth poor)
- *solution:* distributed, application-level infrastructure



# Multimedia: video

- video: sequence of images displayed at constant rate
  - e.g., 24 images/sec
- digital image: array of pixels
  - each pixel represented by bits
- coding: use redundancy *within* and *between* images to decrease # bits used to encode image
  - spatial (within image)
  - temporal (from one image to next)

*spatial coding example:* instead of sending  $N$  values of same color (all purple), send only two values: color value (*purple*) and *number of repeated values* ( $N$ )



frame  $i$

*temporal coding example:* instead of sending complete frame at  $i+1$ , send only differences from frame  $i$



frame  $i+1$

# Multimedia: video

- **CBR: (constant bit rate):** video encoding rate fixed
- **VBR: (variable bit rate):** video encoding rate changes as amount of spatial, temporal coding changes
- **examples:**
  - MPEG I (CD-ROM) 1.5 Mbps
  - MPEG2 (DVD) 3-6 Mbps
  - MPEG4 (often used in Internet, 12 Mbps)

*spatial coding example:* instead of sending  $N$  values of same color (all purple), send only two values: color value (*purple*) and *number of repeated values* ( $N$ )



frame  $i$

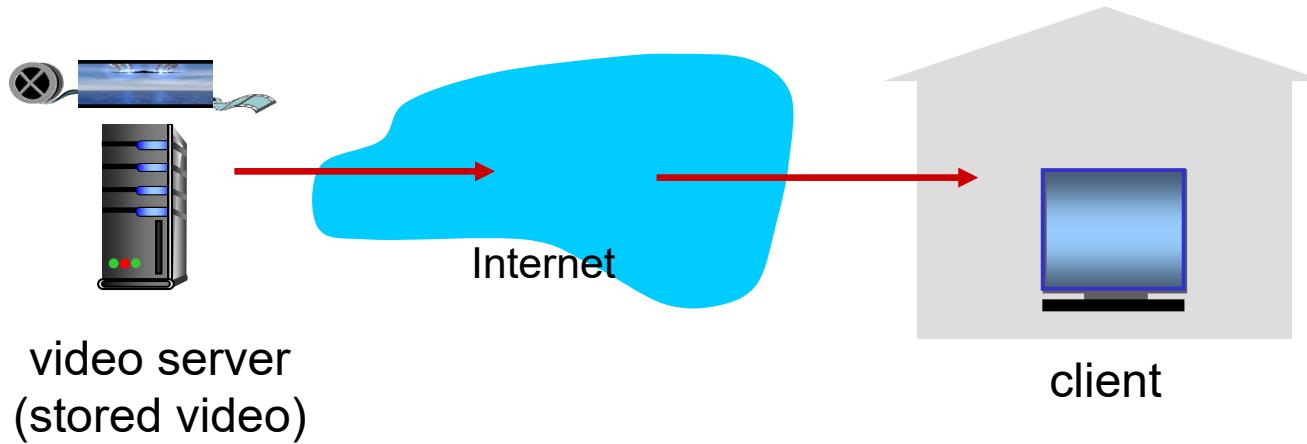
*temporal coding example:* instead of sending complete frame at  $i+1$ , send only differences from frame  $i$



frame  $i+1$

# Streaming stored video:

simple scenario:



Main challenges:

- Server to client bandwidth vary over time.
- Packet loss, delay due to congestion will delay playout, poor video quality.

# Streaming multimedia: DASH

- **DASH: Dynamic, Adaptive Streaming over HTTP**
- **server:**
  - divides video file into multiple chunks
  - each chunk stored, encoded at different rates
  - *manifest file*: provides URLs for different chunks
- **client:**
  - periodically measures server-to-client bandwidth
  - consulting manifest, requests one chunk at a time
    - chooses maximum coding rate sustainable given current bandwidth
    - can choose different coding rates at different points in time (depending on available bandwidth at time)

# Streaming multimedia: DASH

- *DASH: Dynamic, Adaptive Streaming over HTTP*
- “*intelligence*” at client: client determines
  - *when* to request chunk (so that buffer starvation, or overflow does not occur)
  - *what encoding rate* to request (higher quality when more bandwidth available)
  - *where* to request chunk (can request from URL server that is “close” to client or has high available bandwidth)

# Content distribution networks

- **challenge:** how to stream content (selected from millions of videos) to hundreds of thousands of *simultaneous* users?
- **option 1:** single, large “mega-server”
  - single point of failure
  - point of network congestion
  - long path to distant clients
  - multiple copies of video sent over outgoing link

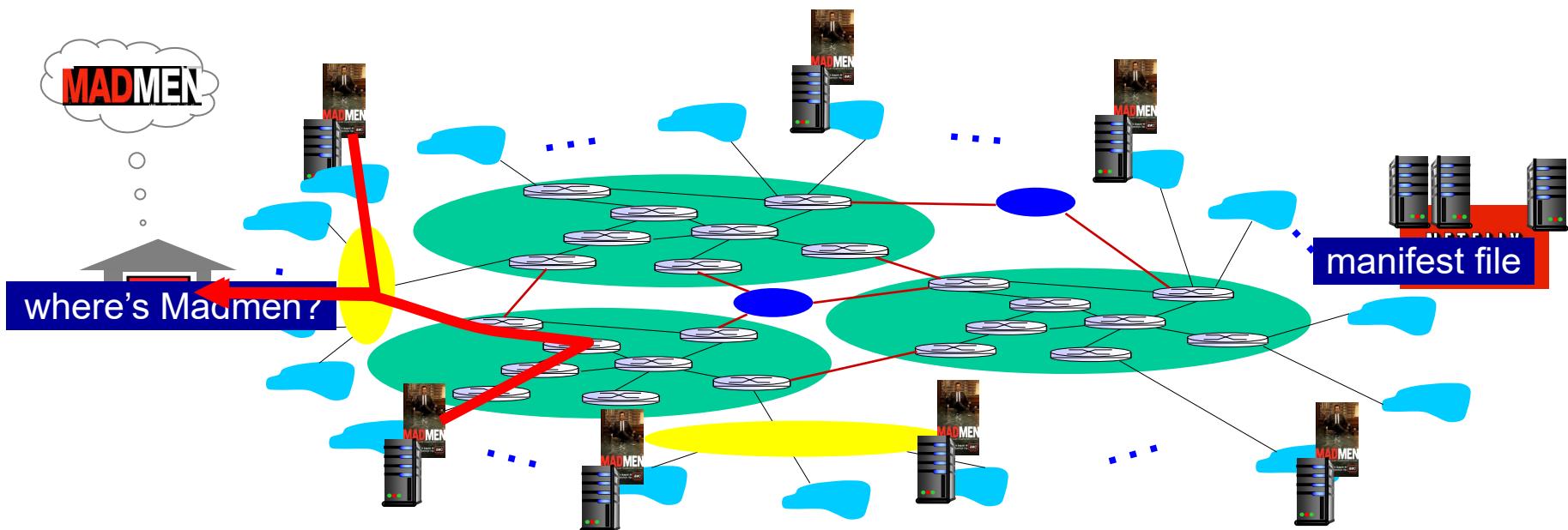
....quite simply: this solution *doesn't scale*

# Content distribution networks

- *challenge*: how to stream content (selected from millions of videos) to hundreds of thousands of simultaneous users?
- *option 2*: store/serve multiple copies of videos at multiple geographically distributed sites (*CDN*)
  - *enter deep*: push CDN servers deep into many access networks
    - close to users
    - used by Akamai, 1700 locations
  - *bring home*: smaller number (10's) of larger clusters in POPs near (but not within) access networks
    - used by Limelight

# Content Distribution Networks (CDNs)

- CDN: stores copies of content at CDN nodes
  - e.g. Netflix stores copies of MadMen
- subscriber requests content from CDN
  - directed to nearby copy, retrieves content
  - may choose different copy if network path congested



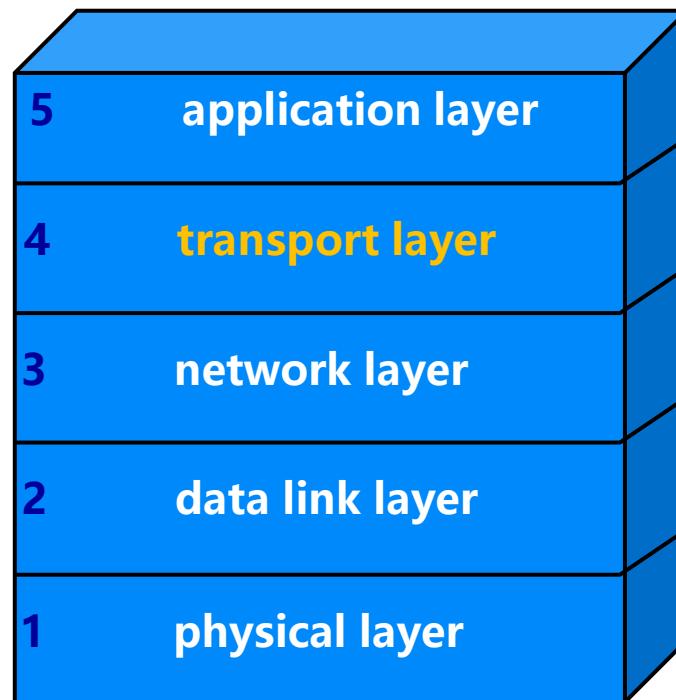
# Chapter 2: summary

- application architectures
  - client-server
  - P2P
- application service requirements:
  - reliability, bandwidth, delay
- Internet transport service model
  - connection-oriented, reliable: TCP
  - unreliable, datagrams: UDP
- specific protocols:
  - HTTP
  - SMTP, POP, IMAP
  - DNS
  - P2P
- video streaming, CDNs

# Chapter 3

# Transport Layer

# Five layered computer network architecture



# Chapter 3 outline

3.1 transport-layer services

3.2 multiplexing and demultiplexing

3.3 connectionless transport: UDP

3.4 principles of reliable data transfer

3.5 connection-oriented transport: TCP

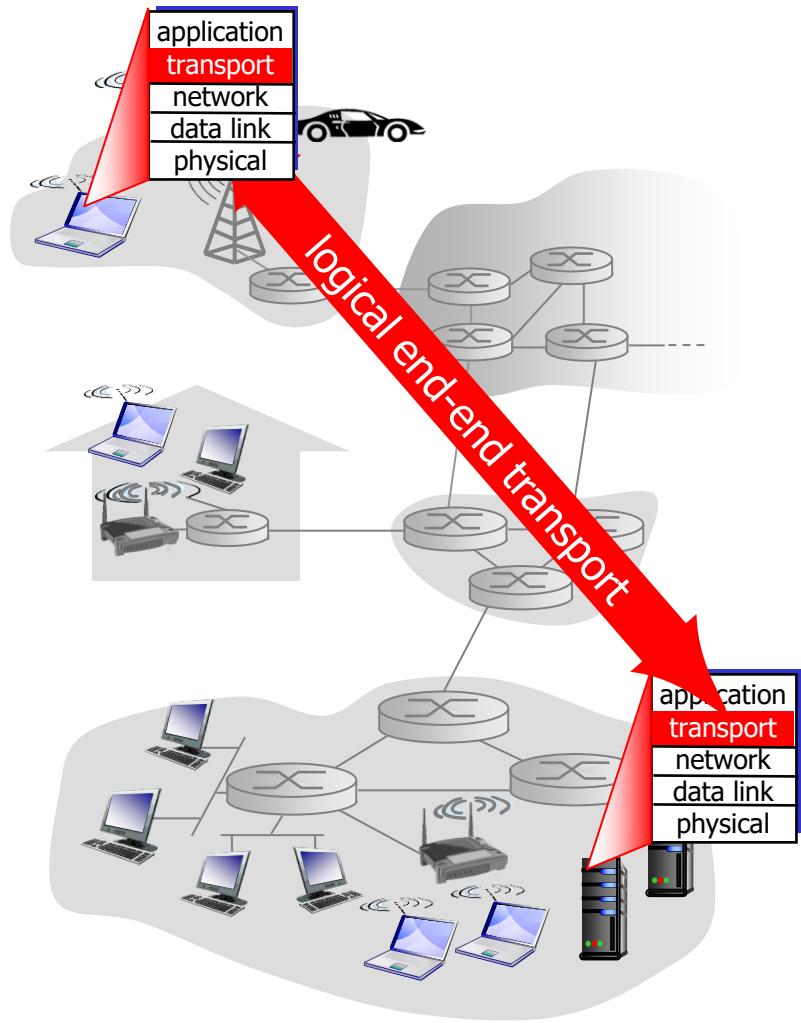
- segment structure
- reliable data transfer
- flow control
- connection management

3.6 principles of congestion control

3.7 TCP congestion control

# 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
  - TCP and UDP



# Transport vs. network layer services and protocols



*household analogy:*

*12 kids in Ann's house sending letters to 12 kids in Bill's house:*

- hosts = houses
- processes = kids
- app messages = letters in envelopes
- transport protocol = Ann and Bill who demux to in-house siblings
- network-layer protocol = postal service

# Transport vs. network layer

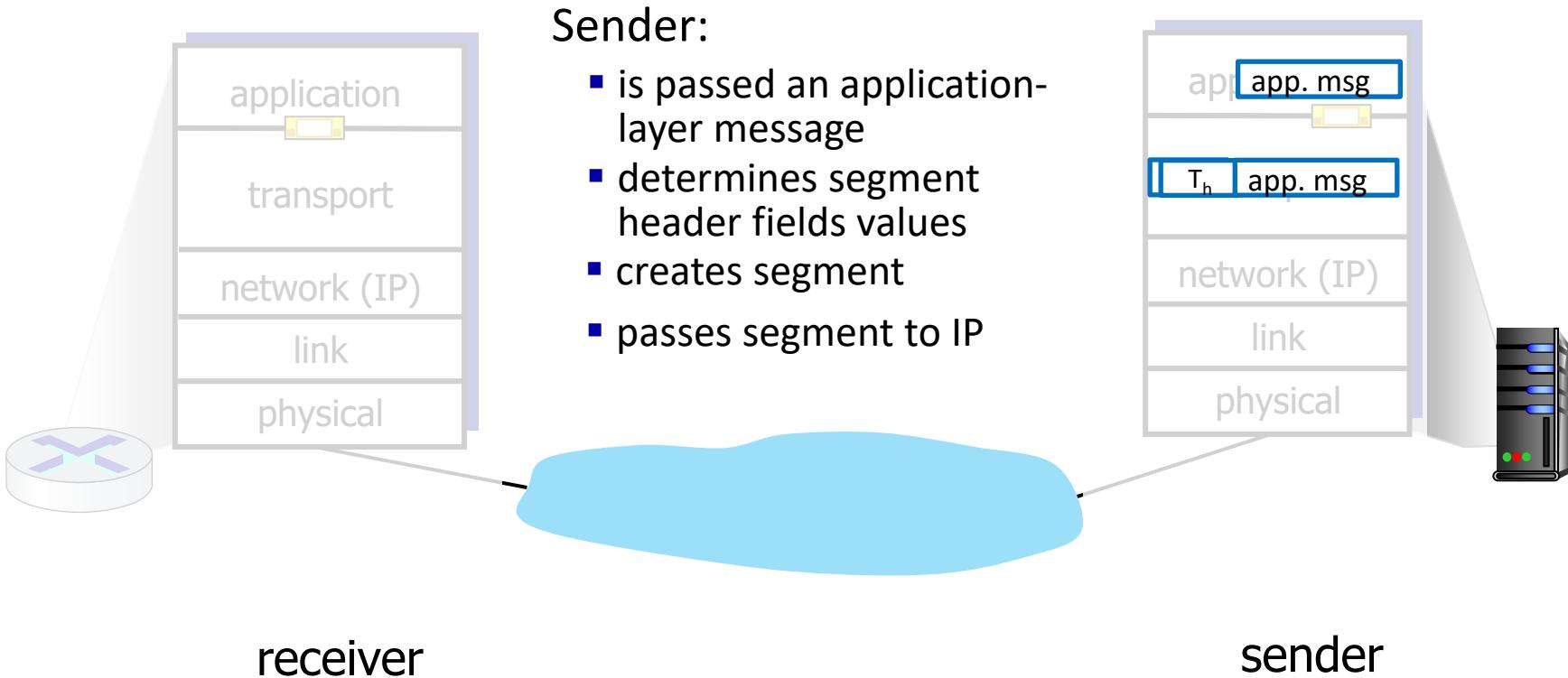
- *network layer*: logical communication between **hosts**
- *transport layer*: logical communication between **processes**
  - relies on, enhances, network layer services

*household analogy:*

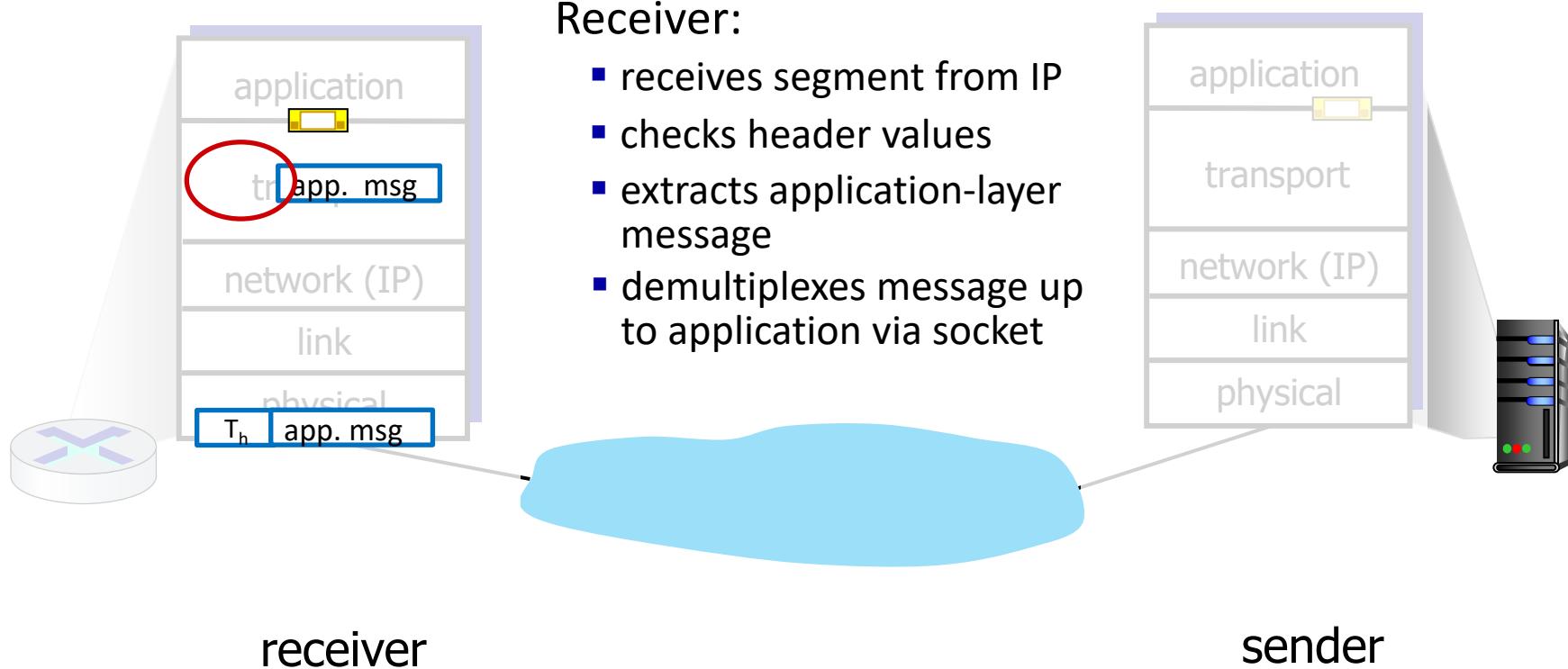
*12 kids in Ann's house sending letters to 12 kids in Bill's house:*

- hosts = houses
- processes = kids
- app messages = letters in envelopes
- transport protocol = Ann and Bill who demux to in-house siblings
- network-layer protocol = postal service

# Transport Layer Actions

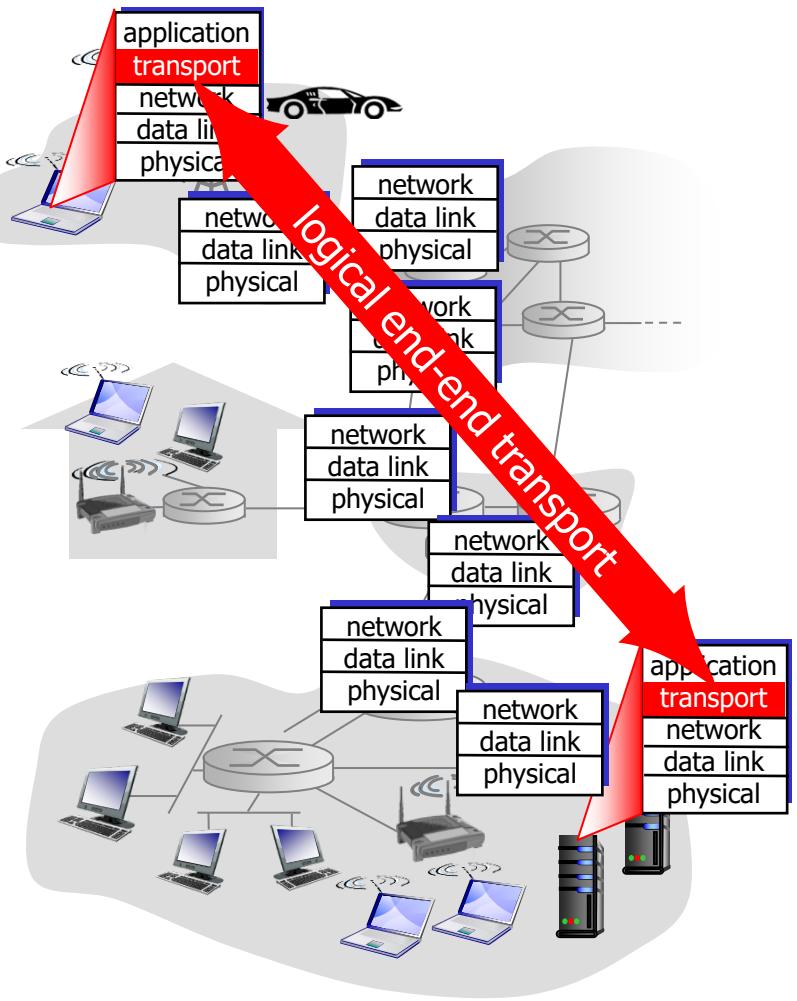


# Transport Layer Actions



# Internet transport-layer protocols

- TCP:
  - reliable, in-order delivery
  - congestion control
  - flow control
  - connection setup
- UDP
  - unreliable, unordered delivery
  - no-frills extension of “best-effort” IP
- services not available:
  - delay guarantees
  - bandwidth guarantees



# Chapter 3 outline

3.1 transport-layer services

3.2 multiplexing and demultiplexing

3.3 connectionless transport: UDP

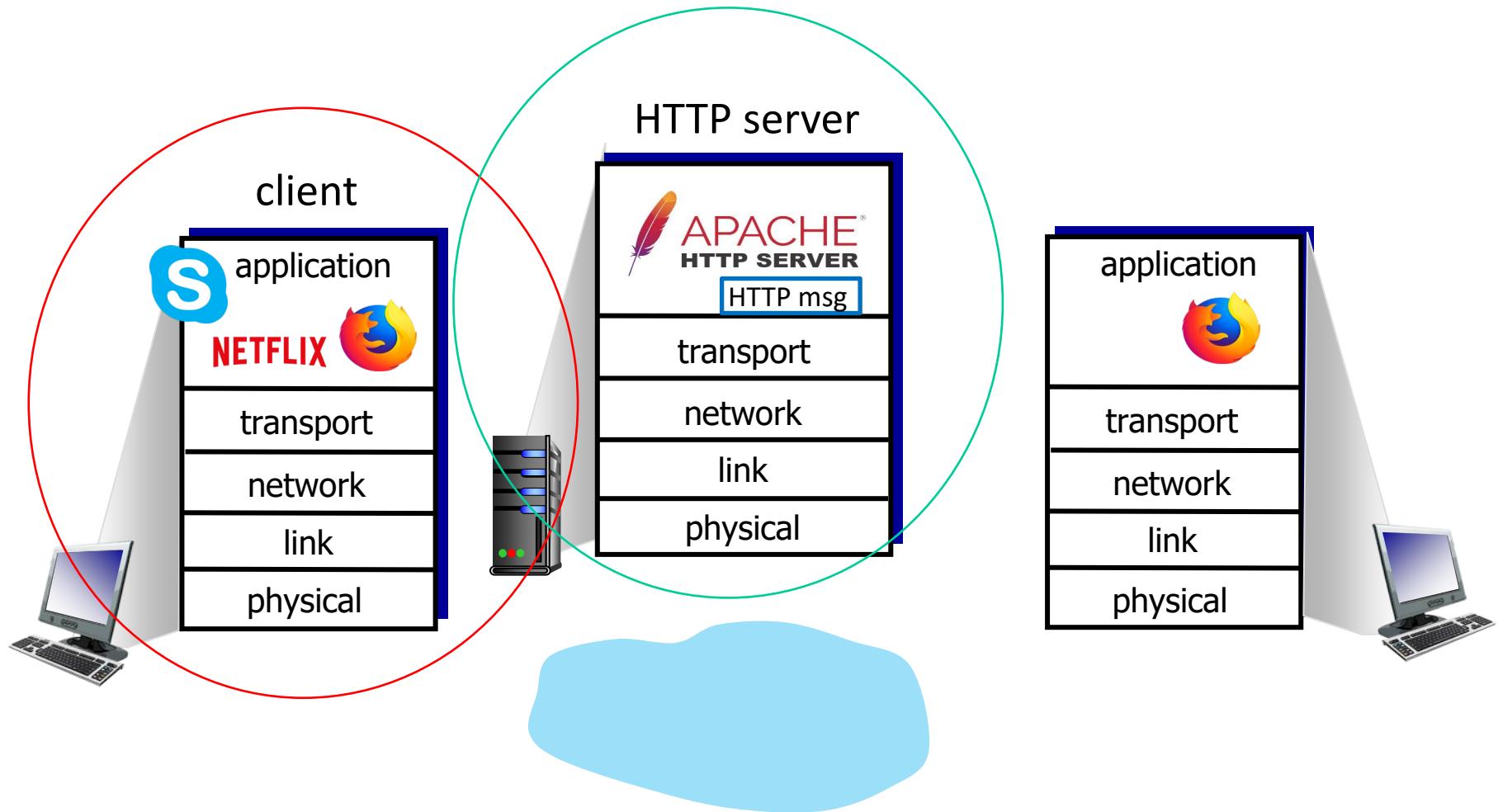
3.4 principles of reliable data transfer

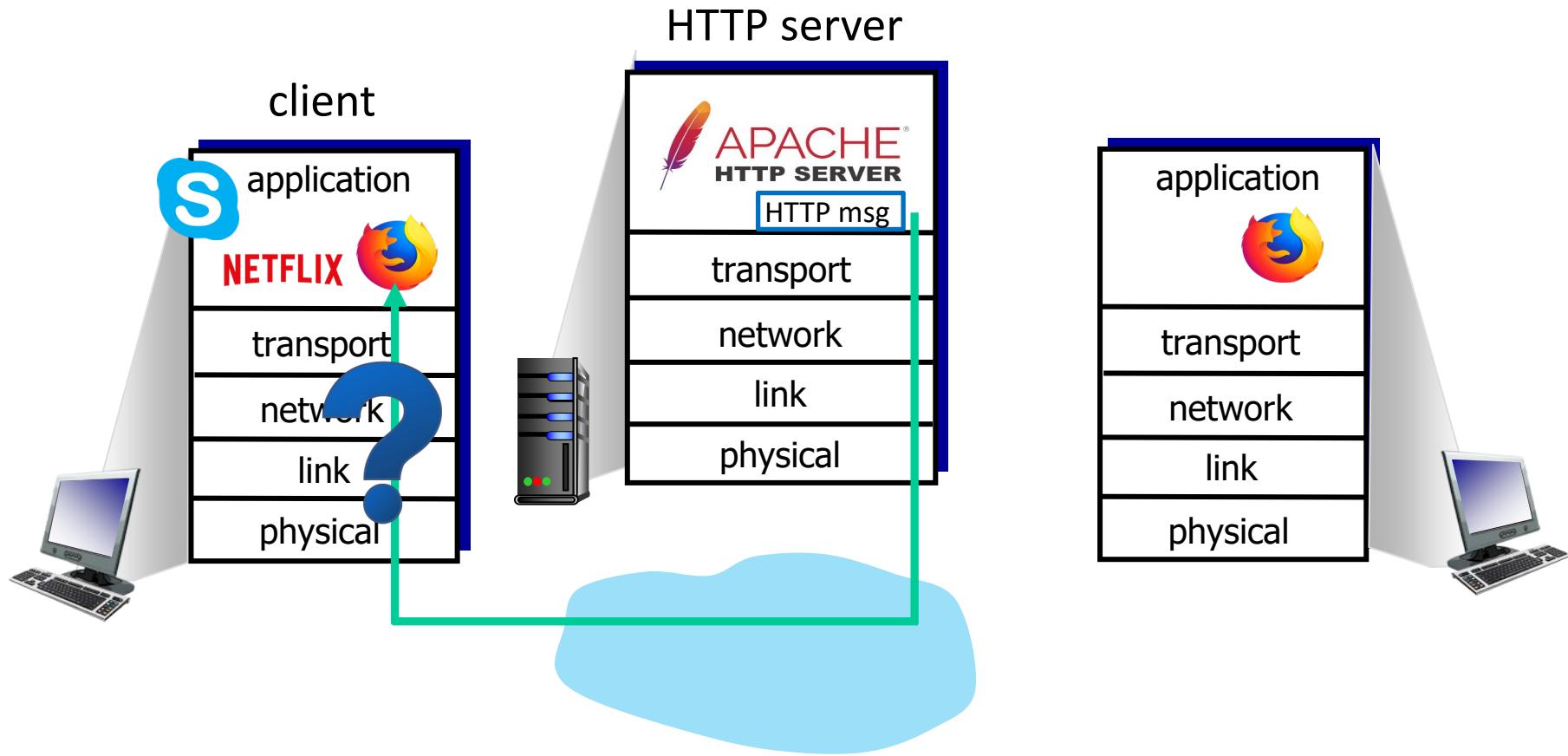
3.5 connection-oriented transport: TCP

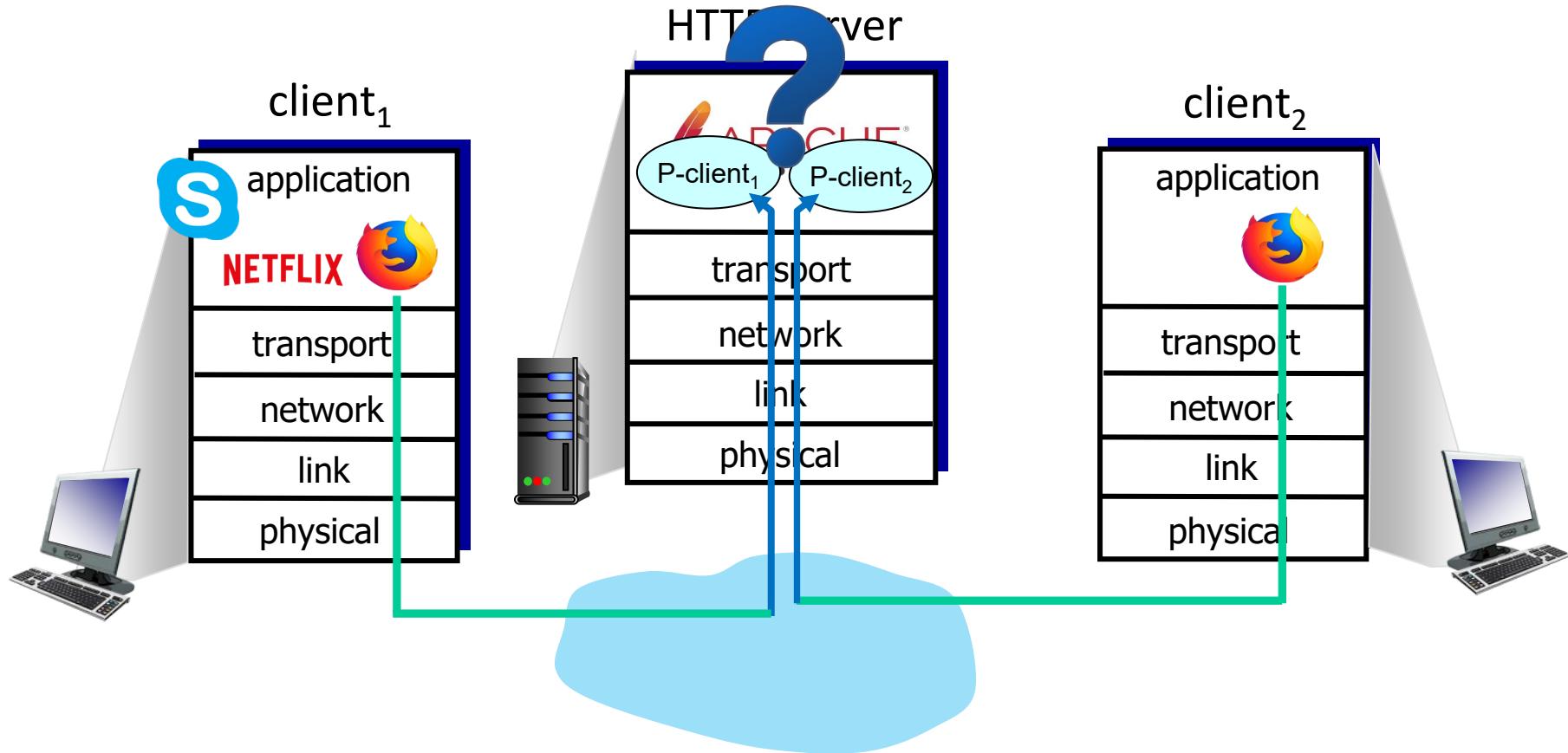
- segment structure
- reliable data transfer
- flow control
- connection management

3.6 principles of congestion control

3.7 TCP congestion control







# Daily-life example for demux



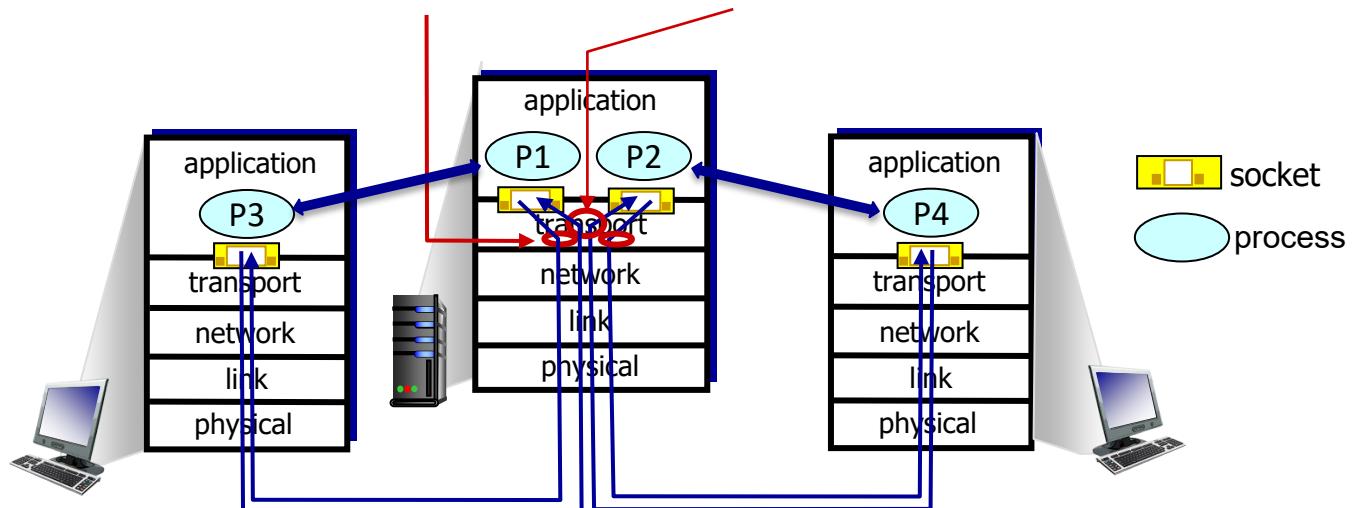
# Multiplexing/demultiplexing

*multiplexing at sender:*

handle data from multiple sockets, add transport header (later used for demultiplexing)

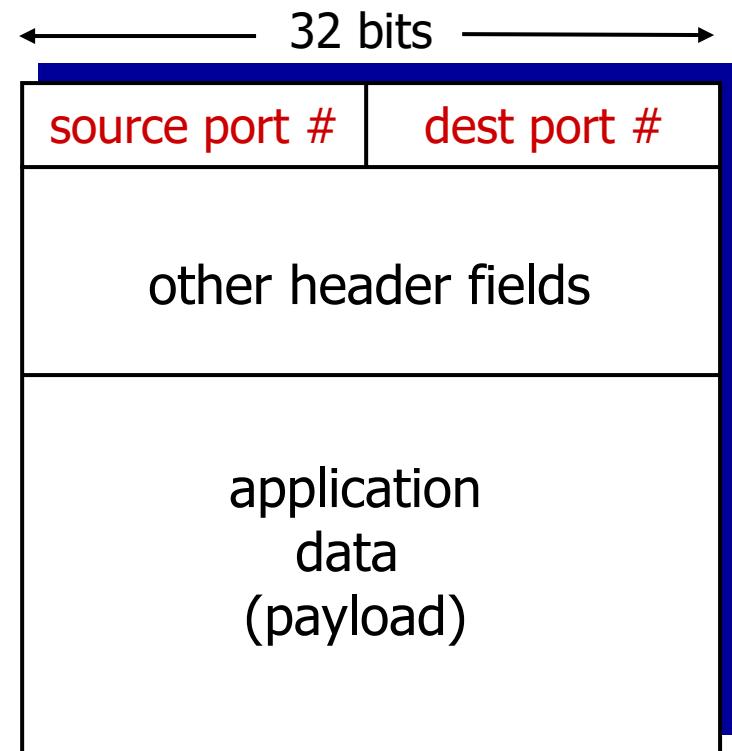
*demultiplexing at receiver:*

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 demultiplexing

- *recall:* created socket has host-local port #:

```
DatagramSocket mySocket1  
= new DatagramSocket(1234) ;
```

- *recall:* when creating datagram to send into UDP socket, must specify
  - destination IP address
  - destination port #

- 
- when host receives UDP segment:

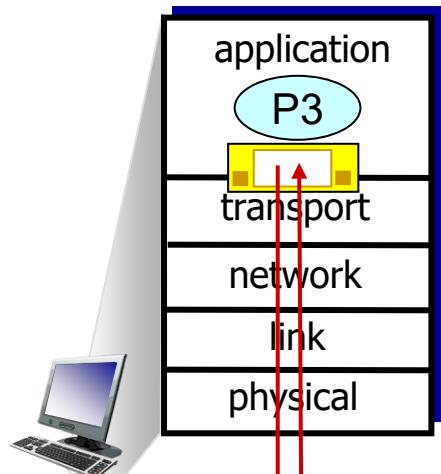
- checks destination port # in segment
- directs UDP segment to socket with that port #



IP datagrams with *same dest. port #*, but different source IP addresses and/or source port numbers will be directed to *same socket* at dest

# Connectionless demux: example

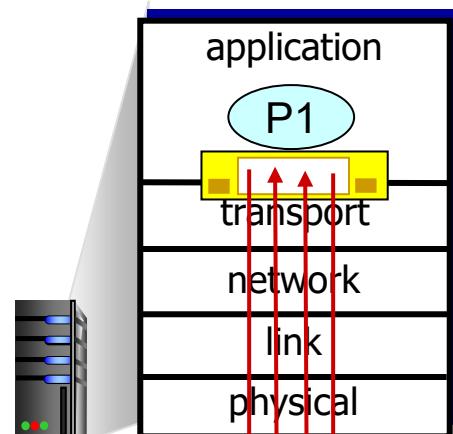
```
DatagramSocket  
mySocket2 = new  
DatagramSocket  
(9157);
```



source port: 9157  
dest port: 6428

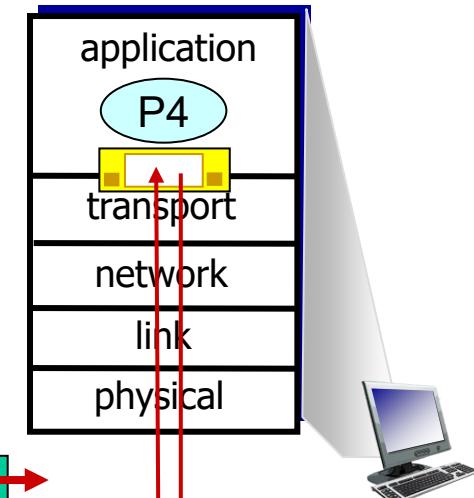
## DatagramSocket

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



source port: 6428  
dest port: 9157

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

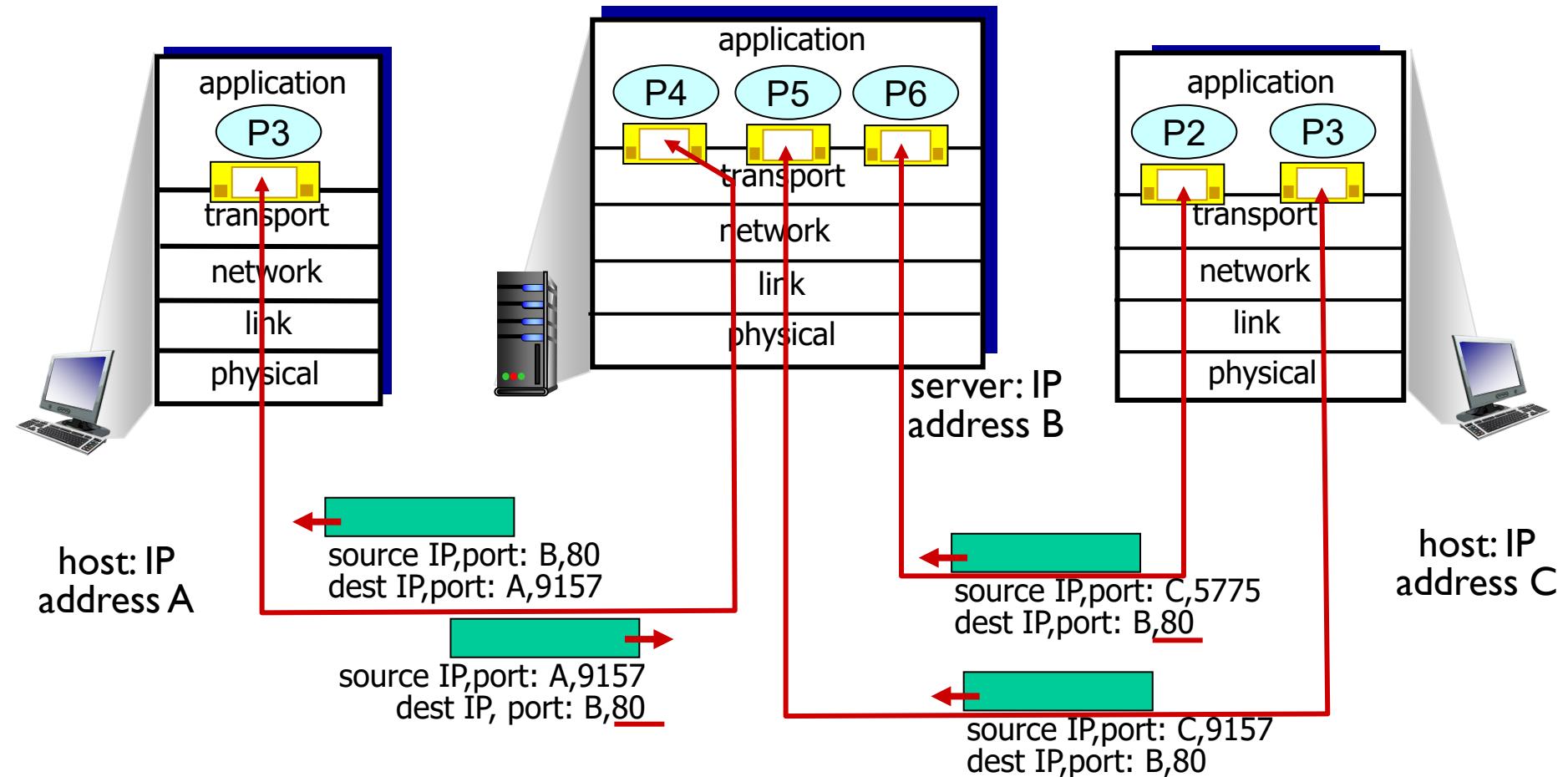


source port: ?  
dest port: ?

# 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

# Connection-oriented demux: example



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

# Summary

- Multiplexing, demultiplexing: based on segment, datagram header field values
- **UDP:** demultiplexing using destination port number (only)
- **TCP:** demultiplexing using 4-tuple: source and destination IP addresses, and port numbers
- Multiplexing/demultiplexing happen at *all* layers

# Chapter 3 outline

3.1 transport-layer services

3.2 multiplexing and demultiplexing

3.3 connectionless transport: UDP

3.4 principles of reliable data transfer

3.5 connection-oriented transport: TCP

- segment structure
- reliable data transfer
- flow control
- connection management

3.6 principles of congestion control

3.7 TCP congestion control

# UDP: User Datagram Protocol

- “no frills,” “bare bones” Internet transport protocol
- “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

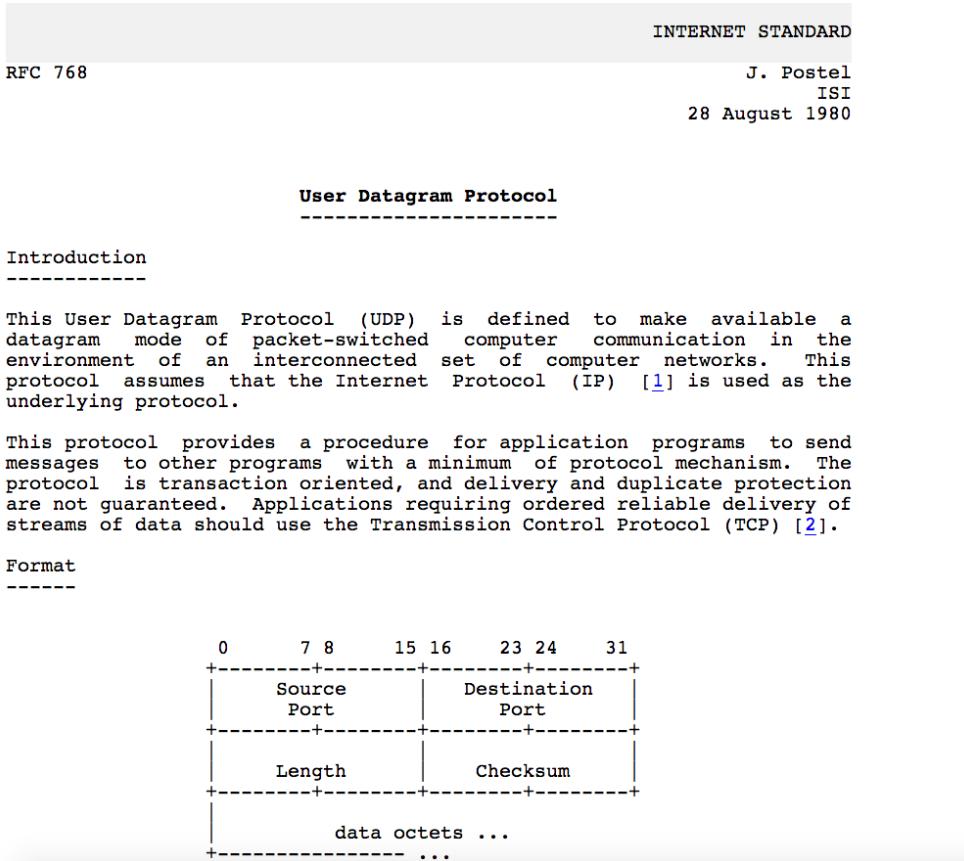
## Why is there a UDP?

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

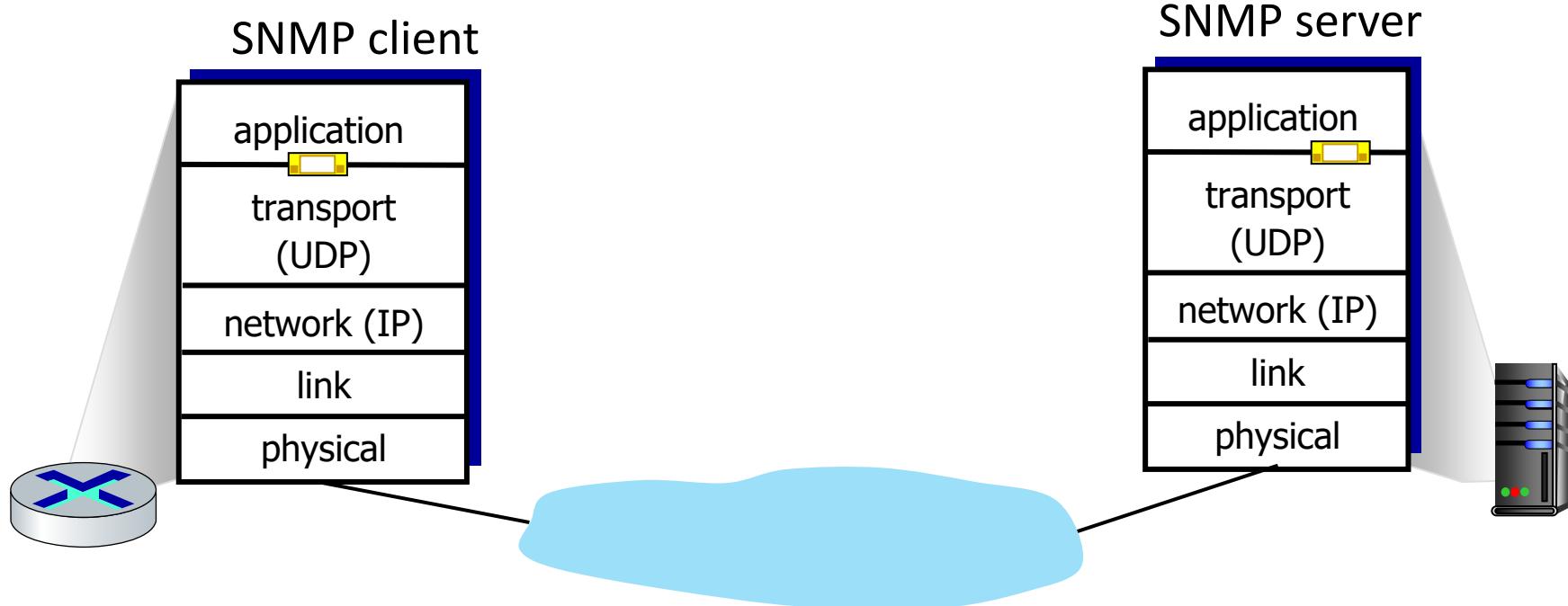
# UDP: User Datagram Protocol

- UDP use:
  - streaming multimedia apps (loss tolerant, rate sensitive)
  - DNS
  - SNMP – network management
  - HTTP/3

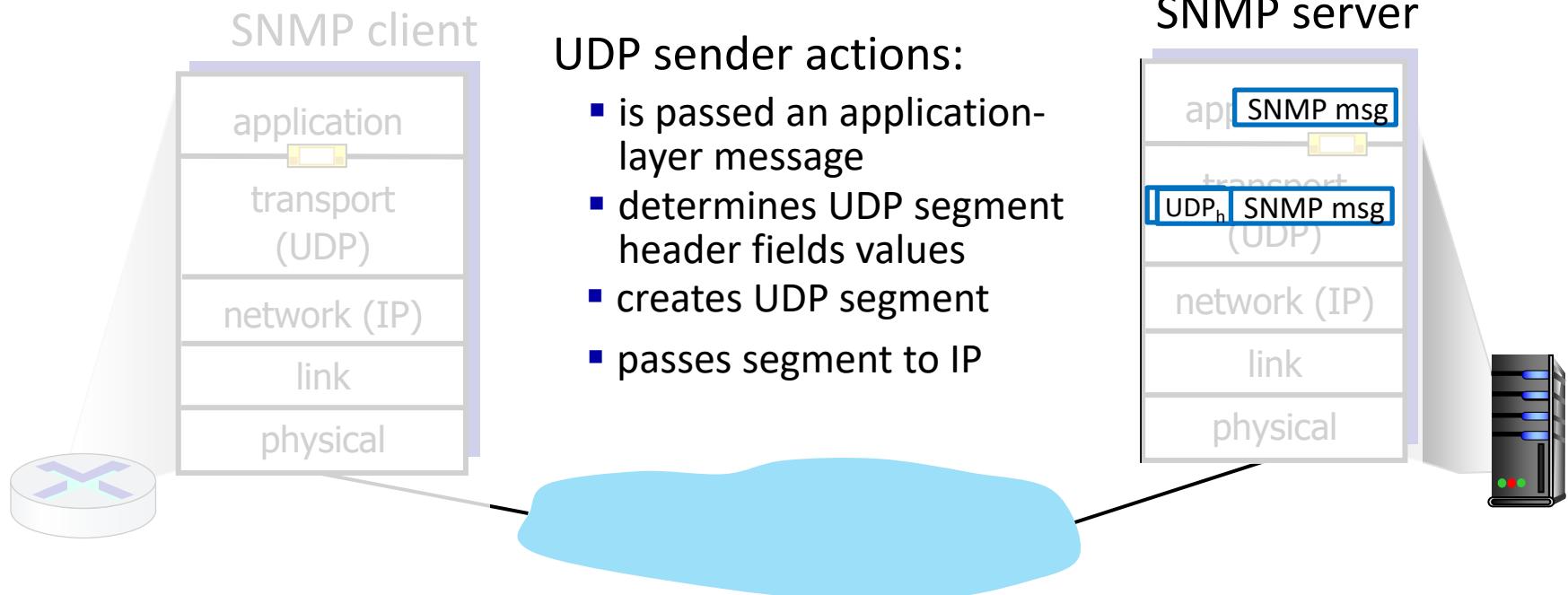
# UDP: User Datagram Protocol [RFC 768]



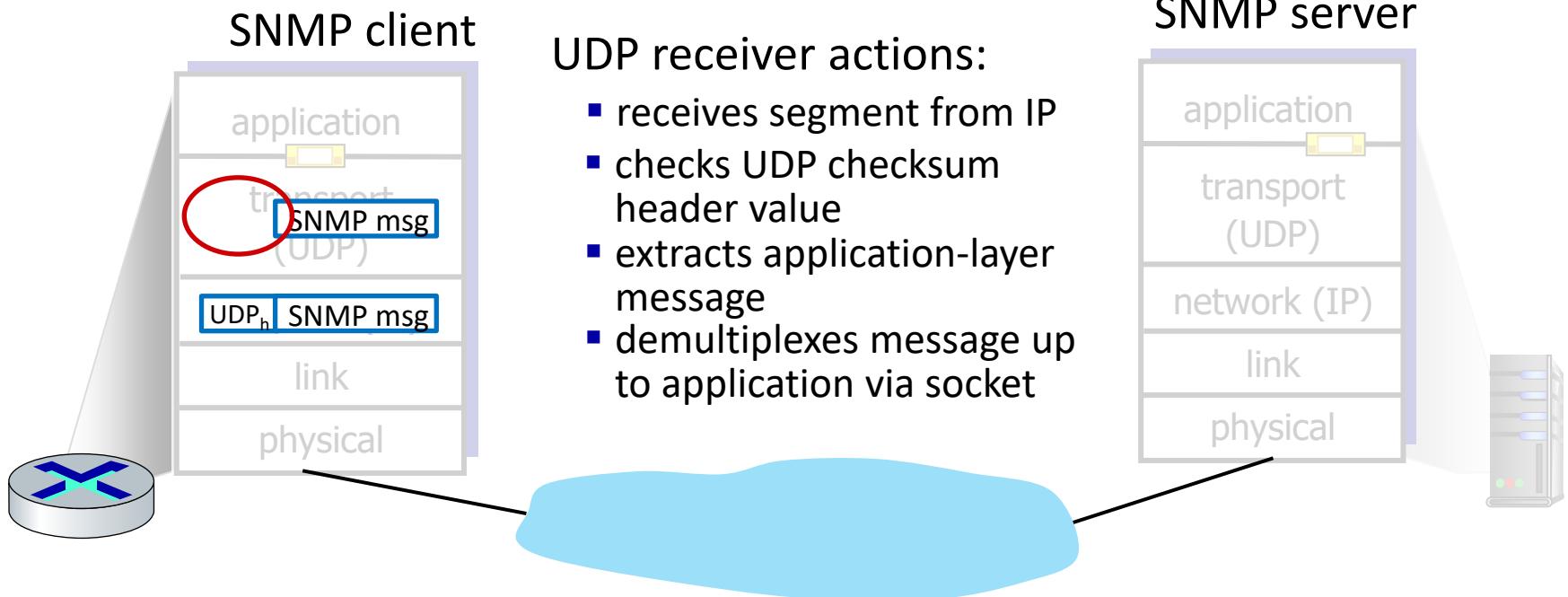
# UDP: Transport Layer Actions



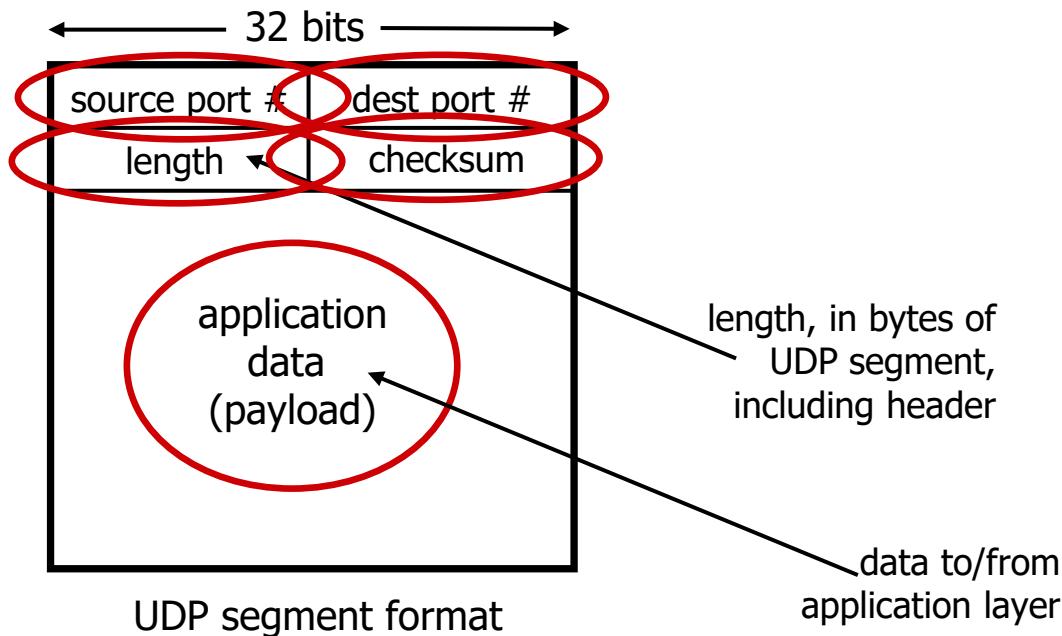
# UDP: Transport Layer Actions



# UDP: Transport Layer Actions



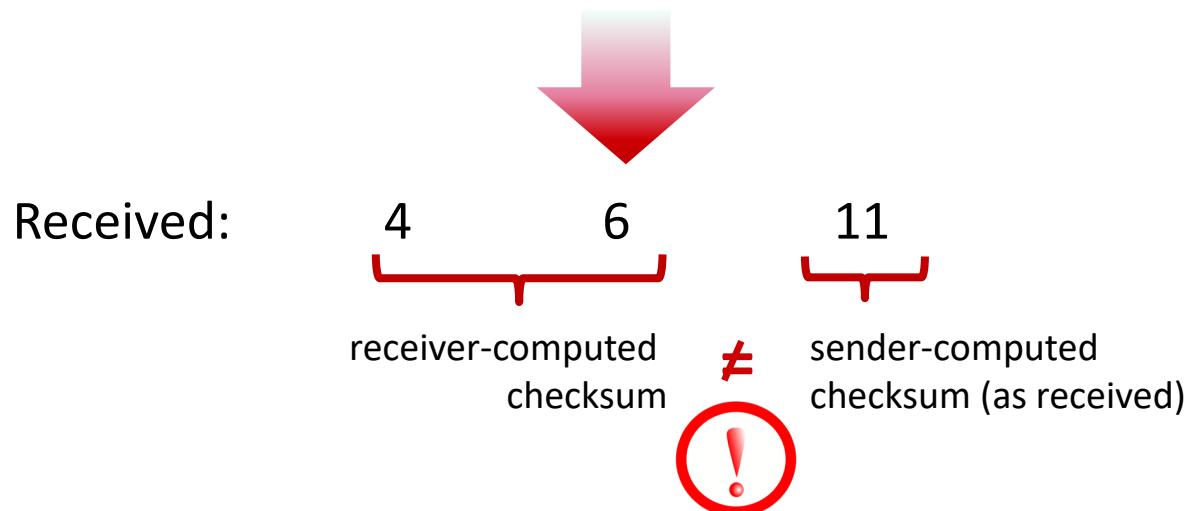
# UDP segment header



# UDP checksum

**Goal:** detect errors (i.e., flipped bits) in transmitted segment

|              | 1 <sup>st</sup> number | 2 <sup>nd</sup> number | sum |
|--------------|------------------------|------------------------|-----|
| Transmitted: | 5                      | 6                      | 11  |



# UDP checksum

**Goal:** detect errors (i.e., flipped bits) in transmitted segment

## sender:

- treat contents of UDP segment (including UDP header fields and IP addresses) as sequence of 16-bit integers
- **checksum:** addition (one's complement sum) of segment content
- checksum value put into UDP checksum field

## receiver:

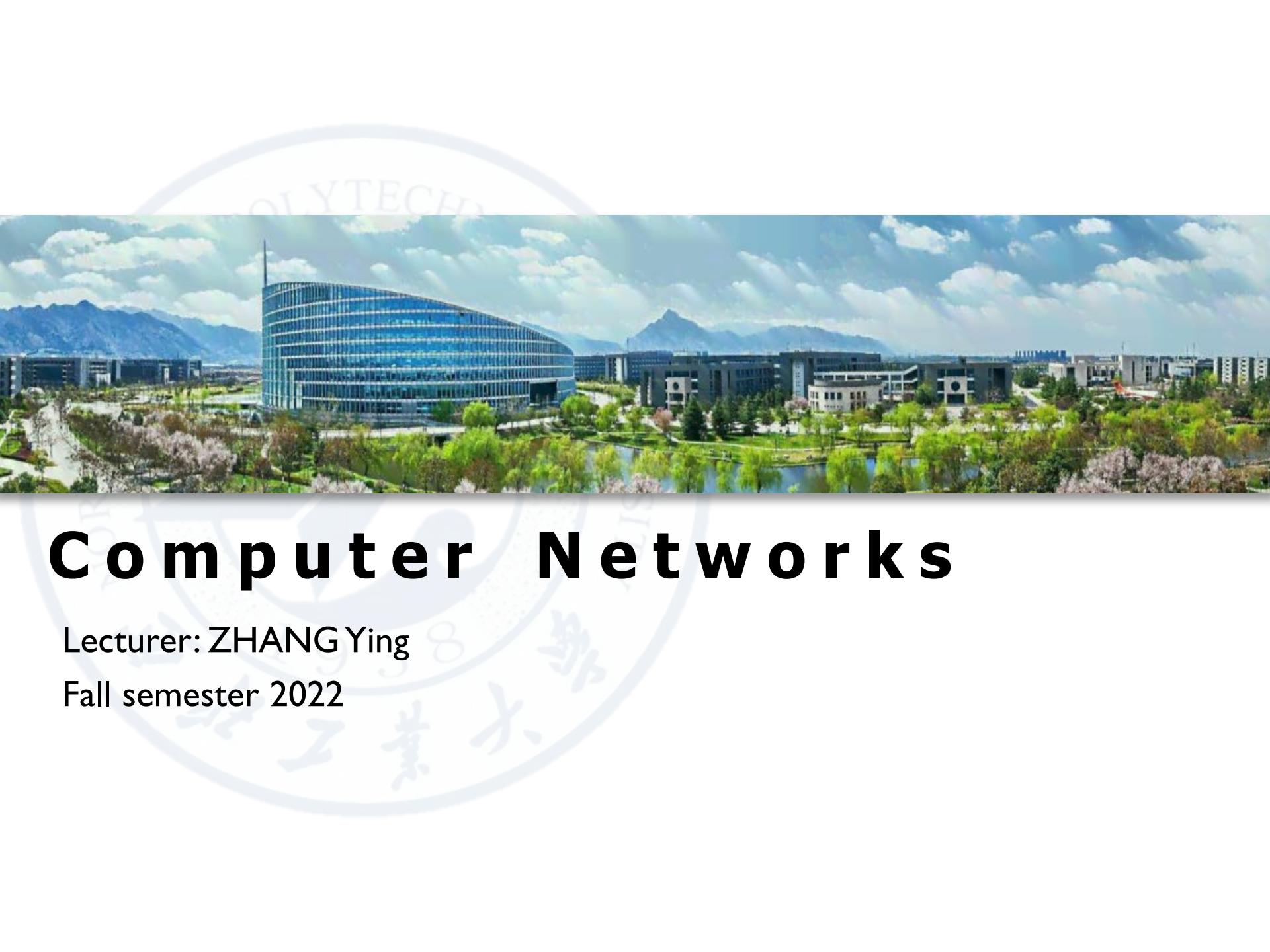
- compute checksum of received segment
- check if computed checksum equals checksum field value:
  - Not equal - error detected
  - Equal - no error detected. *But maybe errors nonetheless? More later ....*

# Internet checksum: an 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 |
| <hr/>      |  |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| wraparound | 1  | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 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



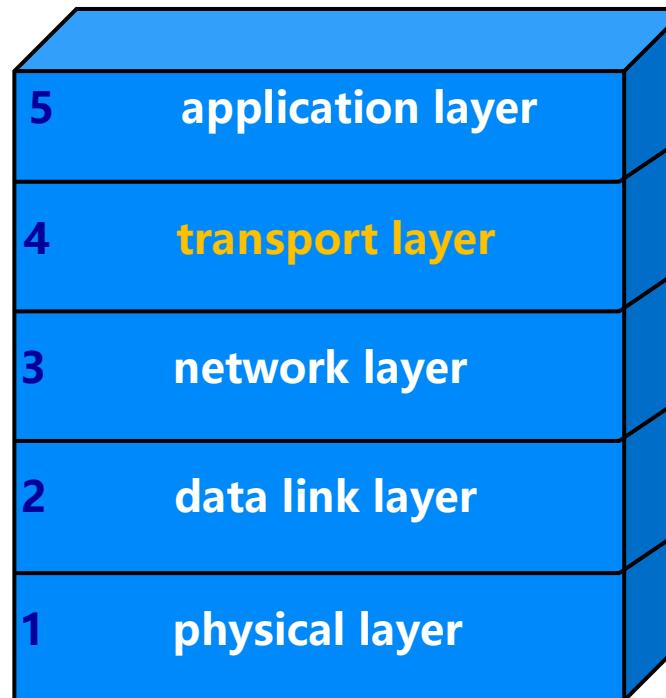
# Computer Networks

Lecturer: ZHANG Ying

Fall semester 2022

# Chapter 3

## Transport Layer



# Internet checksum: an 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 |
| <hr/>      |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| wraparound | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 |
|            | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 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/](http://gaia.cs.umass.edu/kurose_ross/interactive/)

# Internet checksum: weak protection!

example: add two 16-bit integers

|            |   |   |
|------------|---|---|
|            | $\begin{array}{r} 1\ 1\ 1\ 0\ 0\ 1\ 1\ 0\ 0\ 1\ 1\ 0\ 0\ 1\ 1\ 0 \\ 1\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1 \\ \hline \end{array}$ | $\begin{array}{r} 0\ 1 \\ 1\ 0 \end{array}$               |
| wraparound | $\begin{array}{r} 1\ 1\ 0\ 1\ 1\ 0\ 1\ 1\ 1\ 0\ 1\ 1\ 1\ 0\ 1\ 1 \\ \hline \end{array}$   | $\left. \begin{array}{r} 1 \\ \hline \end{array} \right]$ |
| sum        | $1\ 0\ 1\ 1\ 1\ 0\ 1\ 1\ 1\ 0\ 1\ 1\ 1\ 1\ 0\ 0$  |   |
| checksum   | $0\ 1\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 1\ 1$   |   |

Even though numbers have changed (bit flips), **no** change in checksum!

# Chapter 3 outline

3.1 transport-layer services

3.2 multiplexing and demultiplexing

3.3 connectionless transport: UDP

3.4 principles of reliable data transfer

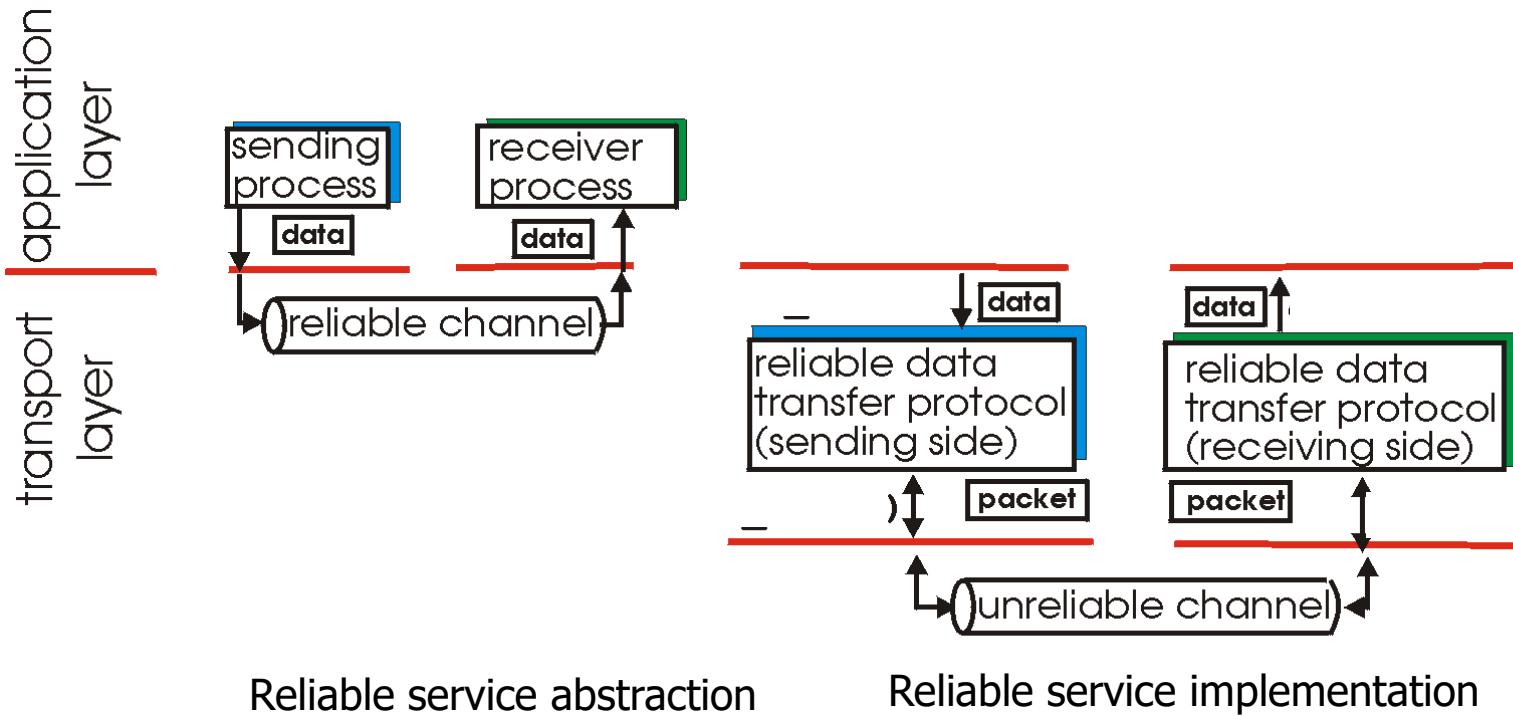
3.5 connection-oriented transport: TCP

- segment structure
- reliable data transfer
- flow control
- connection management

3.6 principles of congestion control

3.7 TCP congestion control

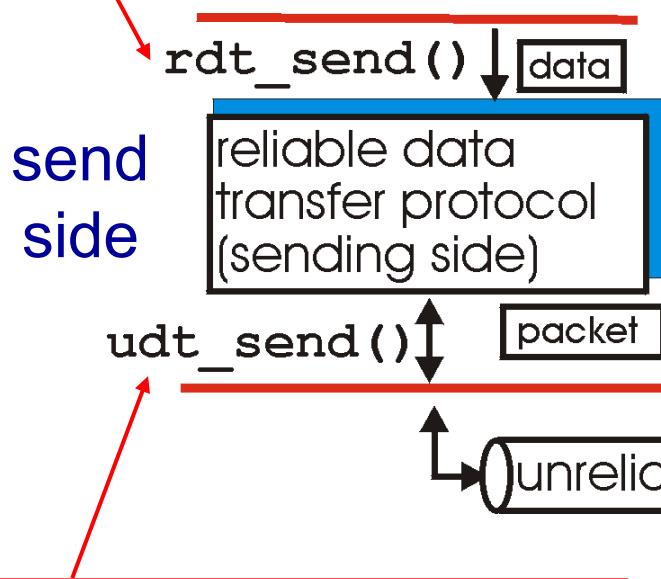
# Principles of reliable data transfer



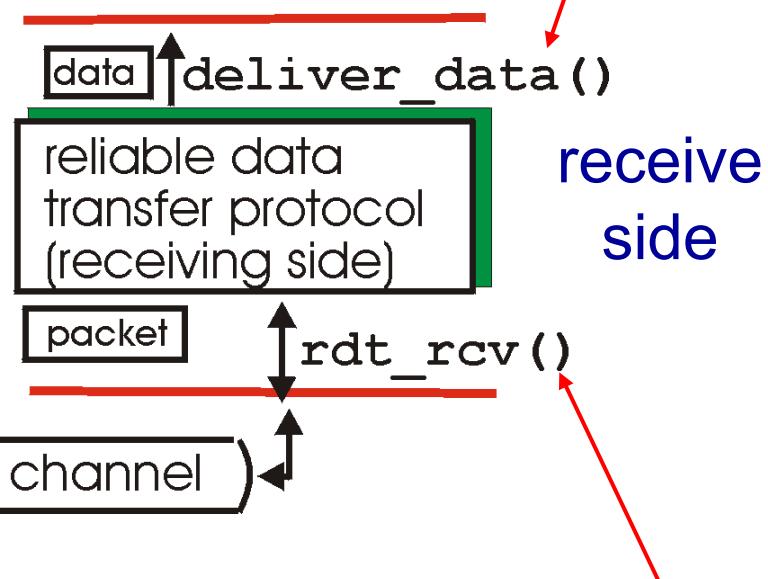
- characteristics of unreliable channel will determine complexity of reliable data transfer protocol (rdt)

# Reliable data transfer: getting started

**`rdt_send()`**: called from above,  
(e.g., by app.). Passed data to  
deliver to receiver upper layer



**`deliver_data()`**: called by  
**rdt** to deliver data to upper



**`udt_send()`**: called by rdt,  
to transfer packet over  
unreliable channel to receiver

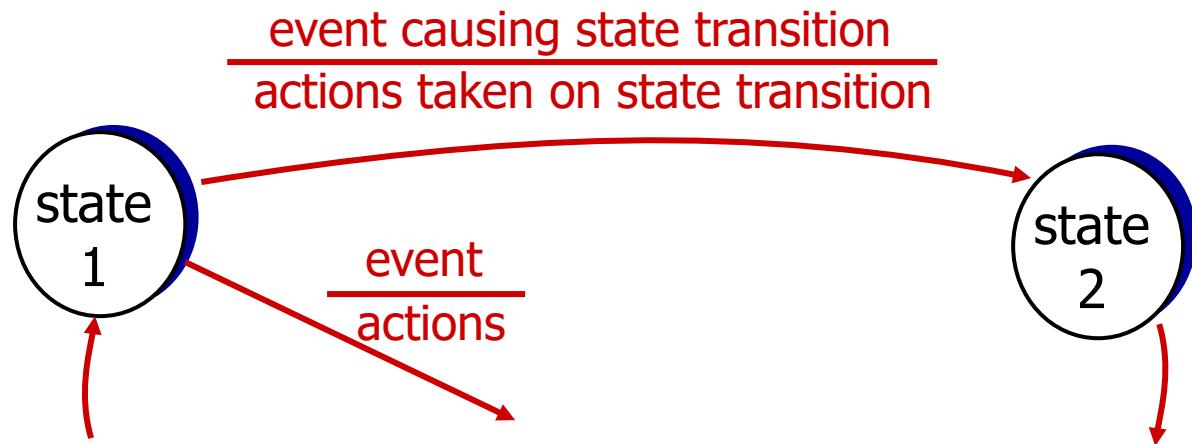
**`rdt_rcv()`**: called when packet  
arrives on rcv-side of channel

# Reliable data transfer: getting started

we'll:

- incrementally develop sender, receiver sides of reliable data transfer protocol (rdt)
- consider only unidirectional data transfer
  - but control info will flow on both directions!
- use finite state machines (FSM) to specify sender, receiver

**state:** when in this “state” next state uniquely determined by next event



# rdt2.0: channel with bit errors

- underlying channel may flip bits in packet
  - checksum to detect bit errors
- the question: how to recover from errors:

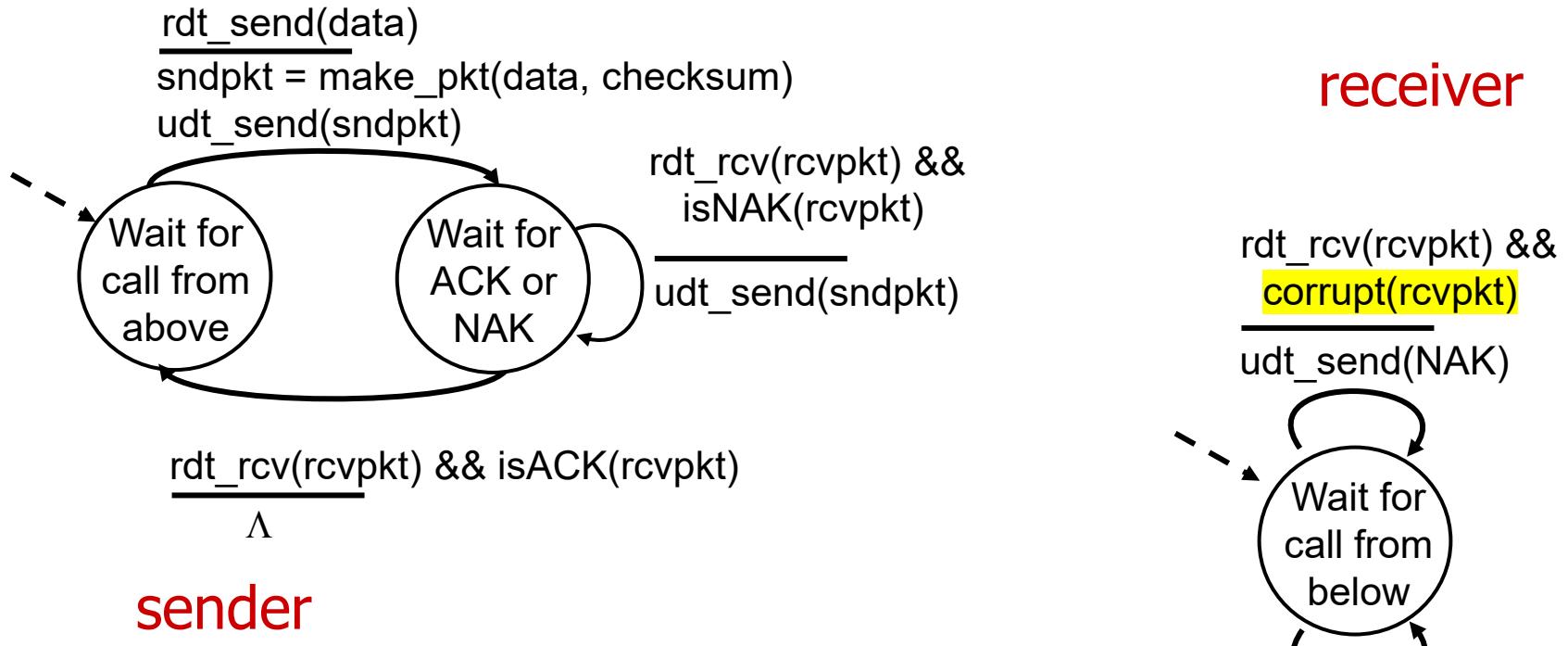
*How do humans recover from “errors”  
during conversation?*



# rdt2.0: channel with bit errors

- underlying channel may flip bits in packet
  - checksum to detect bit errors
- the question: how to recover from errors:
  - *acknowledgements (ACKs)*: receiver explicitly tells sender that pkt received OK
  - *negative acknowledgements (NAKs)*: receiver explicitly tells sender that pkt had errors
  - sender retransmits pkt on receipt of NAK
- new mechanisms in rdt2.0 (beyond rdt1.0):
  - **error detection**
  - **feedback**: control msgs (ACK,NAK) from receiver to sender

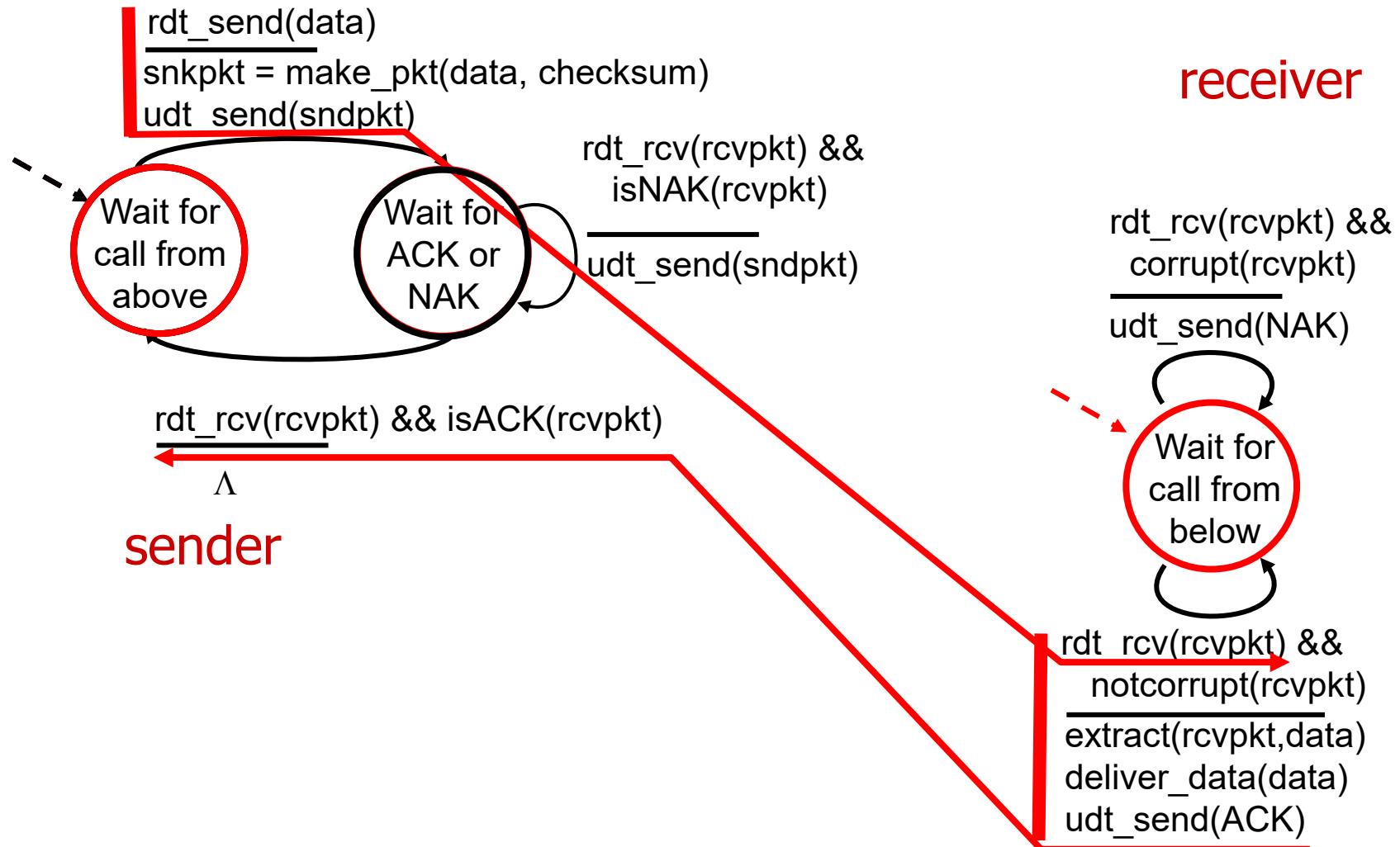
# rdt2.0: FSM specification



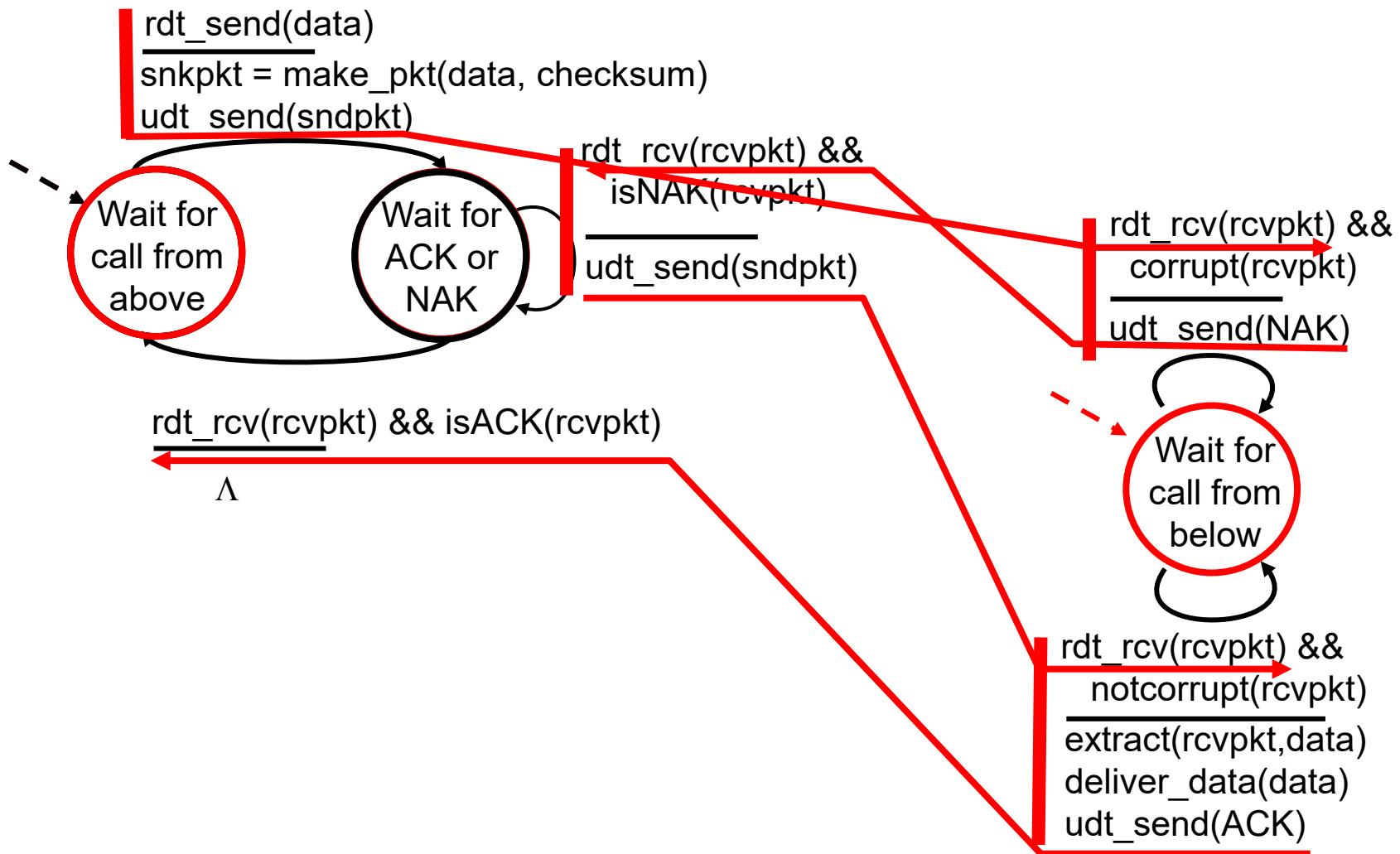
stop and wait

sender sends one packet, then waits for receiver response

# rdt2.0: operation with no errors



# rdt2.0: error scenario



# rdt2.0 has a fatal flaw!

## what happens if ACK/NAK corrupted?

- sender doesn't know what happened at receiver!
- can't just retransmit: possible duplicate

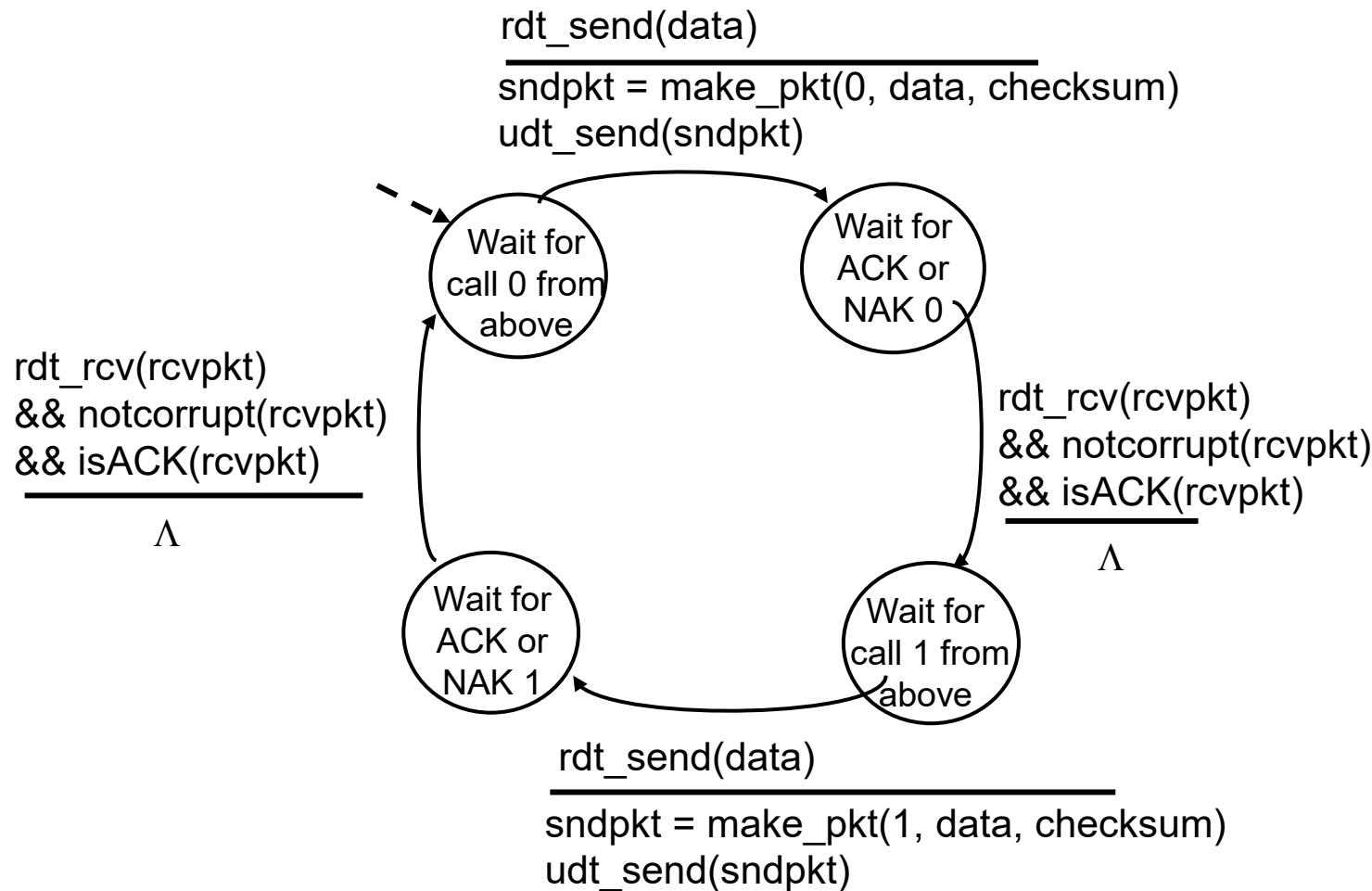
## handling duplicates:

- sender retransmits current pkt if ACK/NAK corrupted
- sender adds *sequence number* to each pkt
- receiver discards (doesn't deliver up) duplicate pkt

stop and wait  
sender sends one packet,  
then waits for receiver  
response

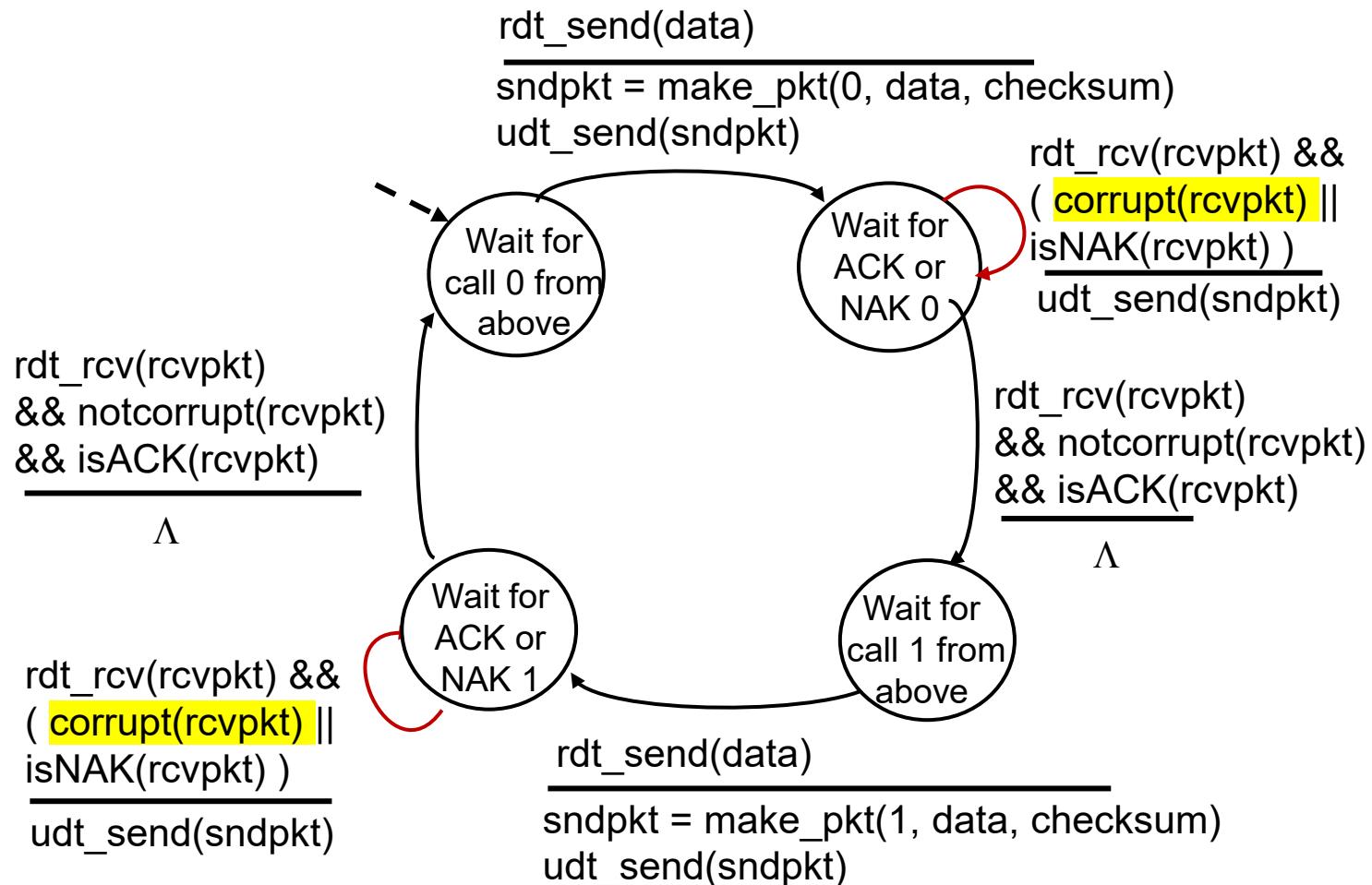
# Scenario I: no corruption

# rdt2.1: sender, handles garbled ACK/NAKs

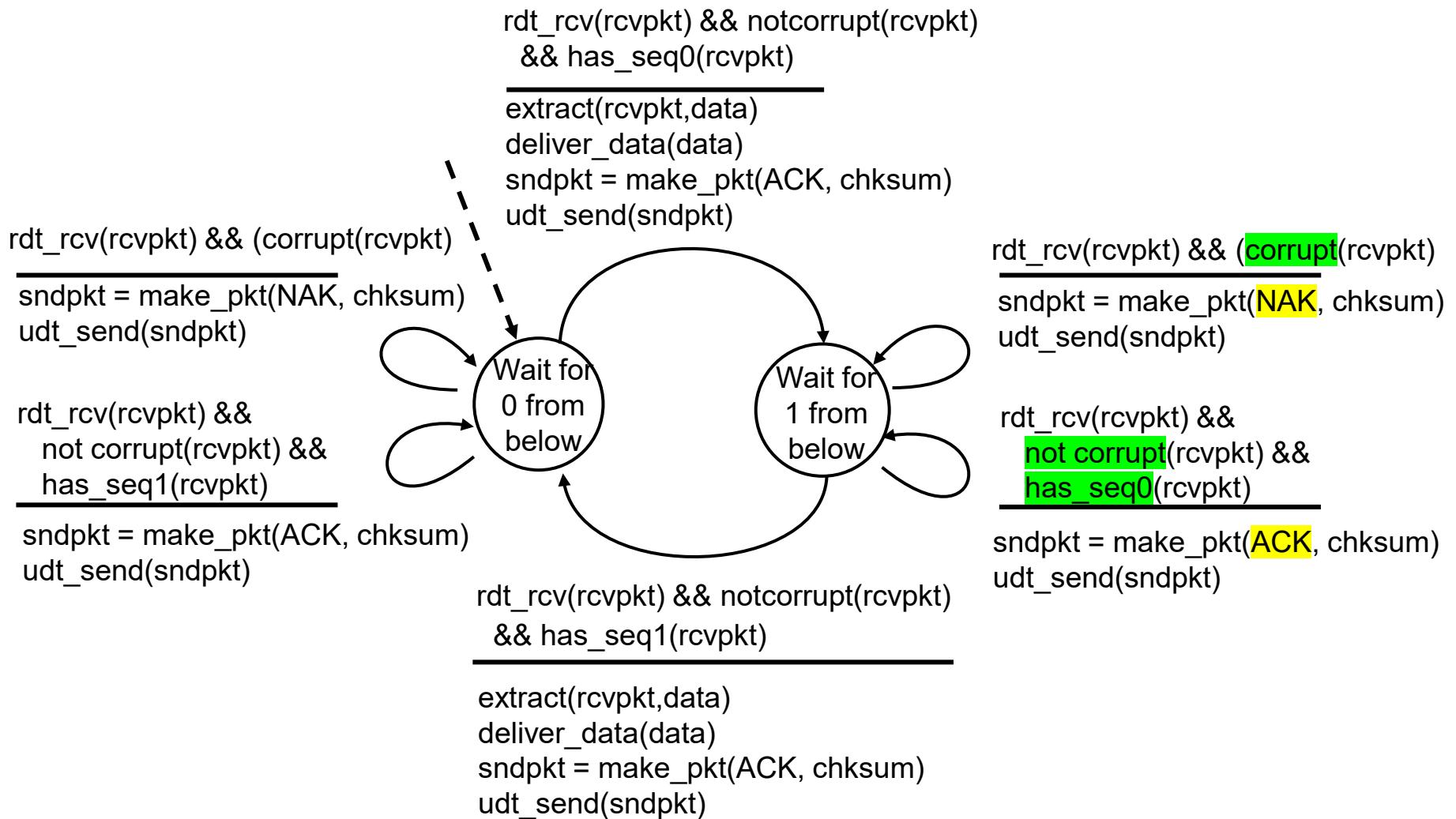


# Scenario II: bit errors occurs

# rdt2.1: sender, handles garbled ACK/NAKs



# rdt2.1: receiver, handles garbled ACK/NAKs



# rdt2.1: discussion

## sender:

- seq # added to pkt
- two seq. #'s (0,1) will suffice. Why?
- must check if received ACK/NAK corrupted
- twice as many states
  - state must “remember” whether “expected” pkt should have seq # of 0 or 1

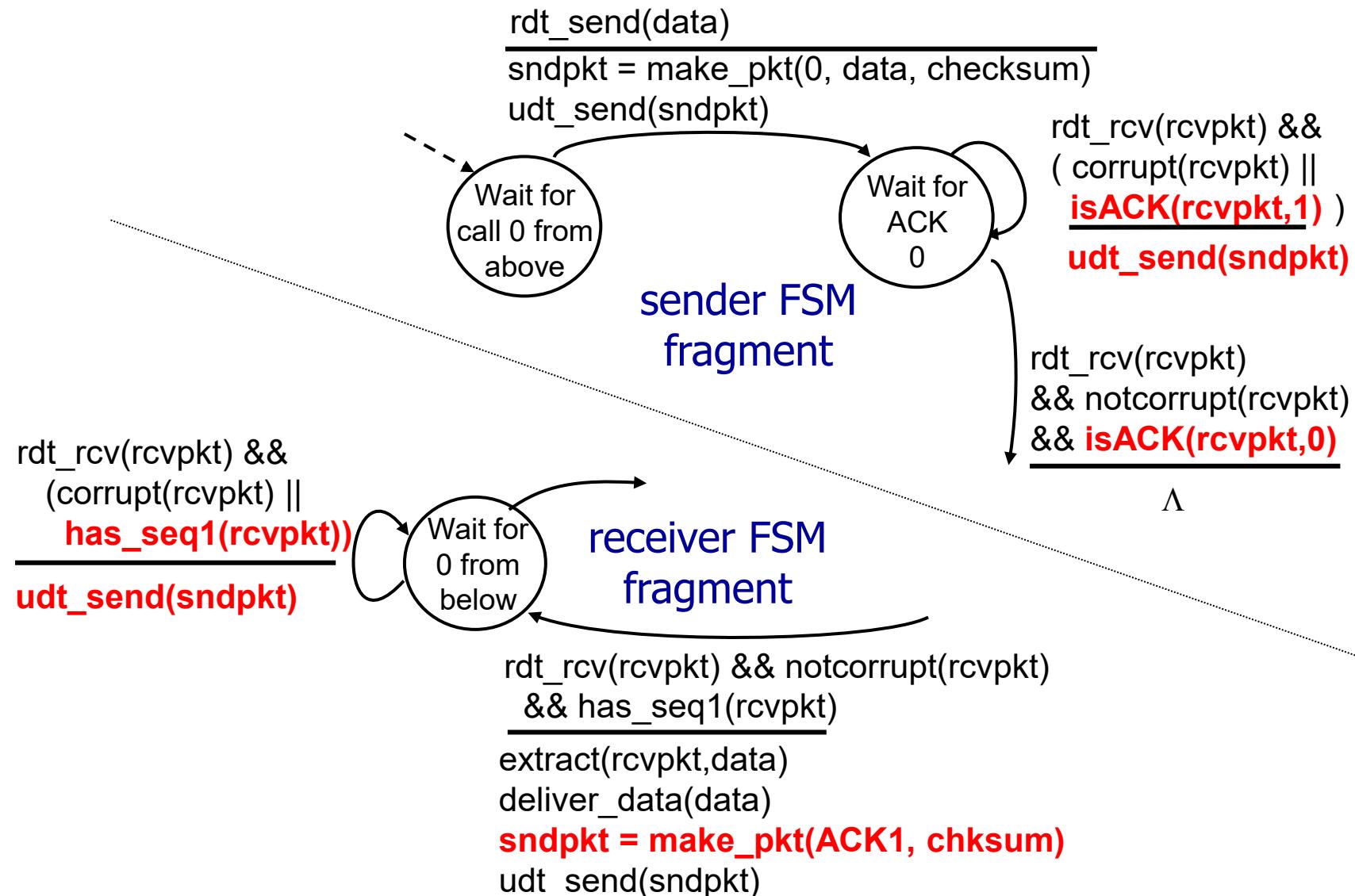
## receiver:

- must check if received packet is duplicate
  - state indicates whether 0 or 1 is expected pkt seq #
- note: receiver can *not* know if its last ACK/NAK received OK at sender

# rdt2.2: a NAK-free protocol

- same functionality as rdt2.1, using ACKs only
- instead of NAK, receiver sends ACK **for last pkt received OK**
  - receiver must *explicitly* include seq # of pkt being ACKed
- duplicate ACK at sender results in same action as NAK: *retransmit current pkt*

# rdt2.2: sender, receiver fragments



# rdt3.0: channels with errors and loss

New channel assumption: underlying channel can also **lose** packets (data, ACKs)

- checksum, seq. #, ACKs, retransmissions will be of help ... but not enough

*Q:* How do *humans* handle lost sender-to-receiver words in conversation?

# rdt3.0: channels with errors and loss

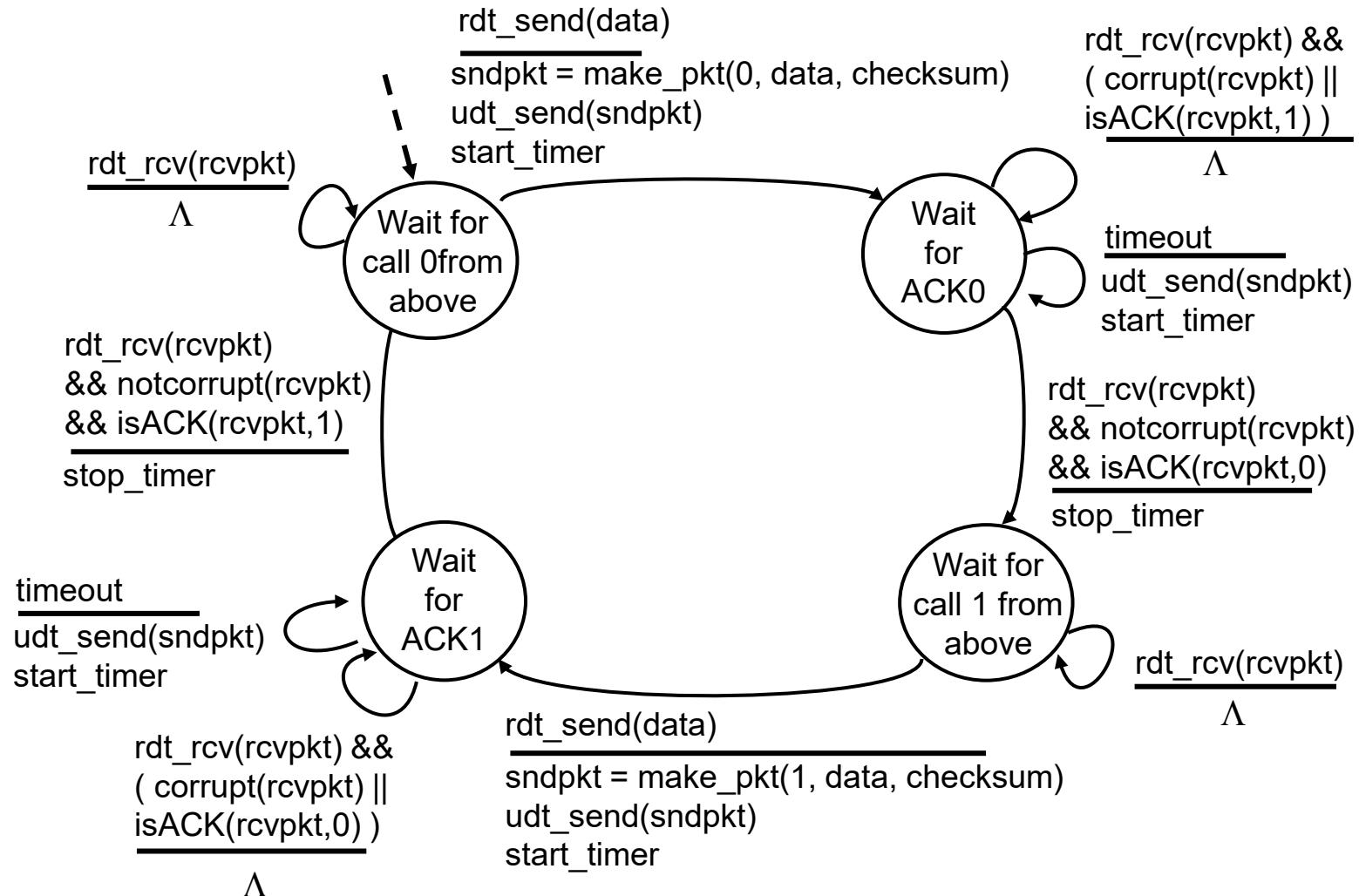
approach: sender waits “reasonable” amount of time for ACK

- retransmits if no ACK received in this time
- if pkt (or ACK) just delayed (not lost):
  - retransmission will be duplicate, but seq. #'s already handles this
  - receiver must specify seq # of pkt being ACKed
- requires countdown timer

*timeout*



# rdt3.0 sender





# Computer Networks

Lecturer: ZHANG Ying  
Fall semester 2022

# TCP 3-way handshake

## Client state

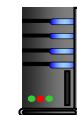
```
clientSocket = socket(AF_INET, SOCK_STREAM)
```

LISTEN

```
clientSocket.connect((serverName, serverPort))
```

SYNSENT

choose init seq num, x  
send TCP SYN msg



SYNbit=1, Seq=x

ESTAB

received SYNACK(x)  
indicates server is live;  
send ACK for SYNACK;  
this segment may contain  
client-to-server data

SYNbit=1, Seq=y  
ACKbit=1; ACKnum=x+1

ACKbit=1, ACKnum=y+1

## Server state

```
serverSocket = socket(AF_INET, SOCK_STREAM)  
serverSocket.bind(('', serverPort))  
serverSocket.listen(1)  
connectionSocket, addr = serverSocket.accept()
```

LISTEN

SYN RCVD

choose init seq num, y  
send TCP SYNACK  
msg, acking SYN

received ACK(y)  
indicates client is live

ESTAB

# Closing a TCP connection

- client, server each close their side of connection
  - send TCP segment with FIN bit = 1
- respond to received FIN with ACK
  - on receiving FIN, ACK can be combined with own FIN
- simultaneous FIN exchanges can be handled

# Chapter 3: roadmap

- Transport-layer services
- Multiplexing and demultiplexing
- Connectionless transport: UDP
- Principles of reliable data transfer
- Connection-oriented transport: TCP
- **Principles of congestion control**
- TCP congestion control



# Principles of congestion control

## Congestion:

- informally: “too many sources sending too much data too fast for *network* to handle”
- manifestations:
  - long delays (queueing in router buffers)
  - packet loss (buffer overflow at routers)

# Principles of congestion control

## Congestion:

- informally: “too many sources sending too much data too fast for *network* to handle”
- manifestations:
  - long delays (queueing in router buffers)
  - packet loss (buffer overflow at routers)
- different from flow control!
- a top-10 problem!



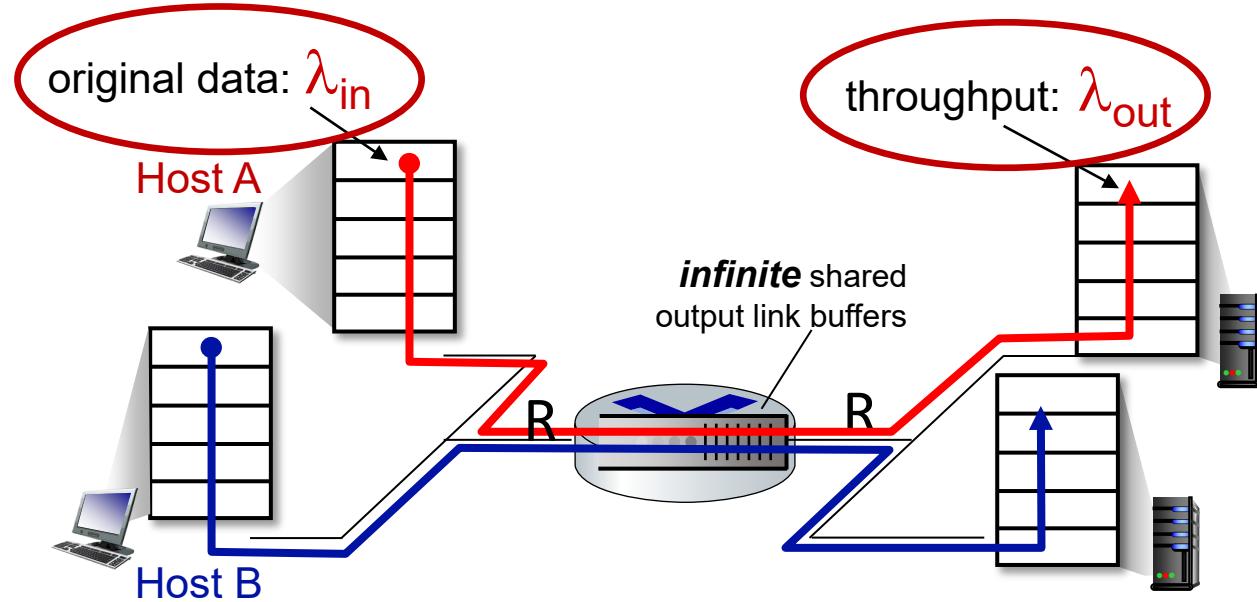
**congestion control:**  
too many senders,  
sending too fast

**flow control:** one sender  
too fast for one receiver

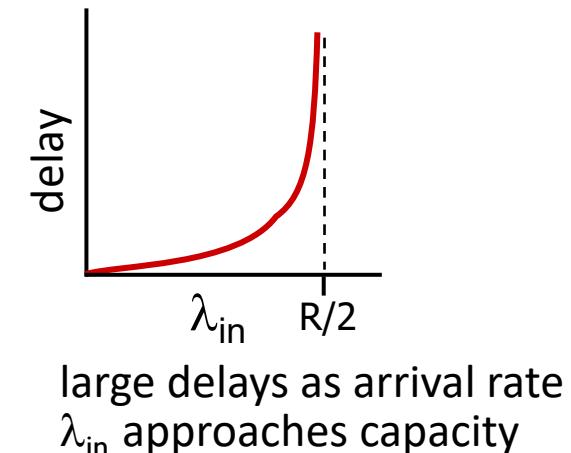
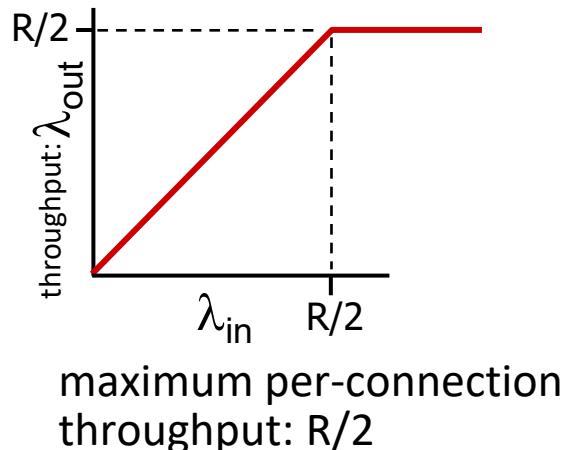
# Causes/costs of congestion: scenario 1

Simplest scenario:

- one router, infinite buffers
- input, output link capacity:  $R$
- two flows
- no retransmissions needed

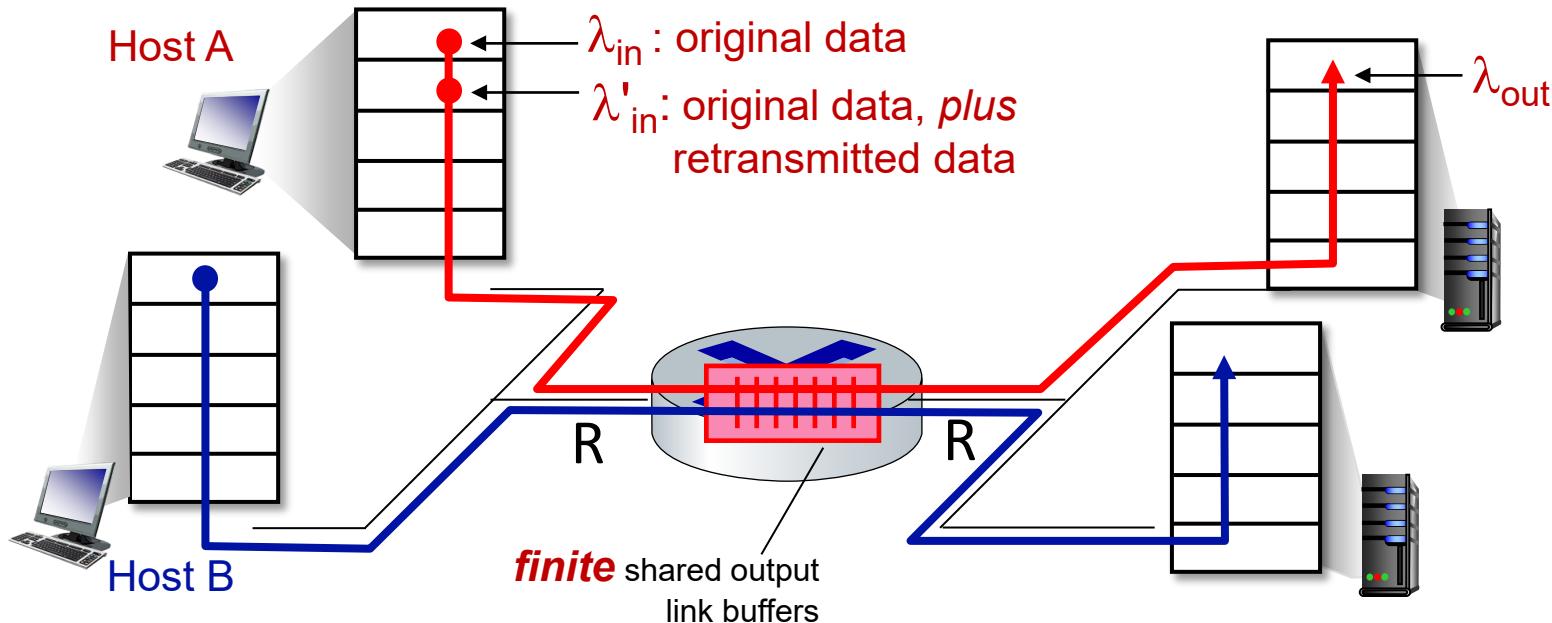


**Q:** What happens as arrival rate  $\lambda_{in}$  approaches  $R/2$ ?



# Causes/costs of congestion: scenario 2

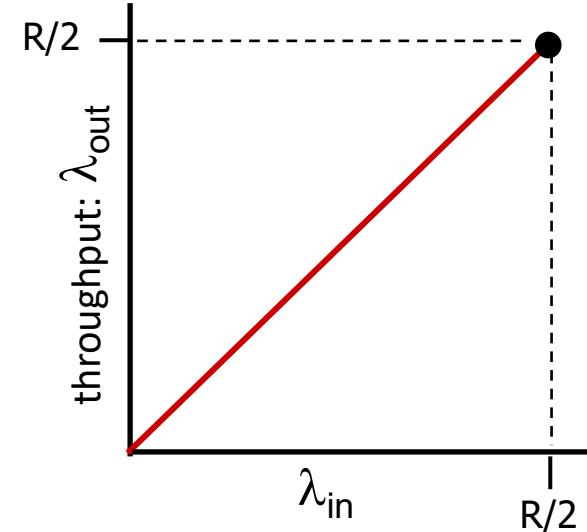
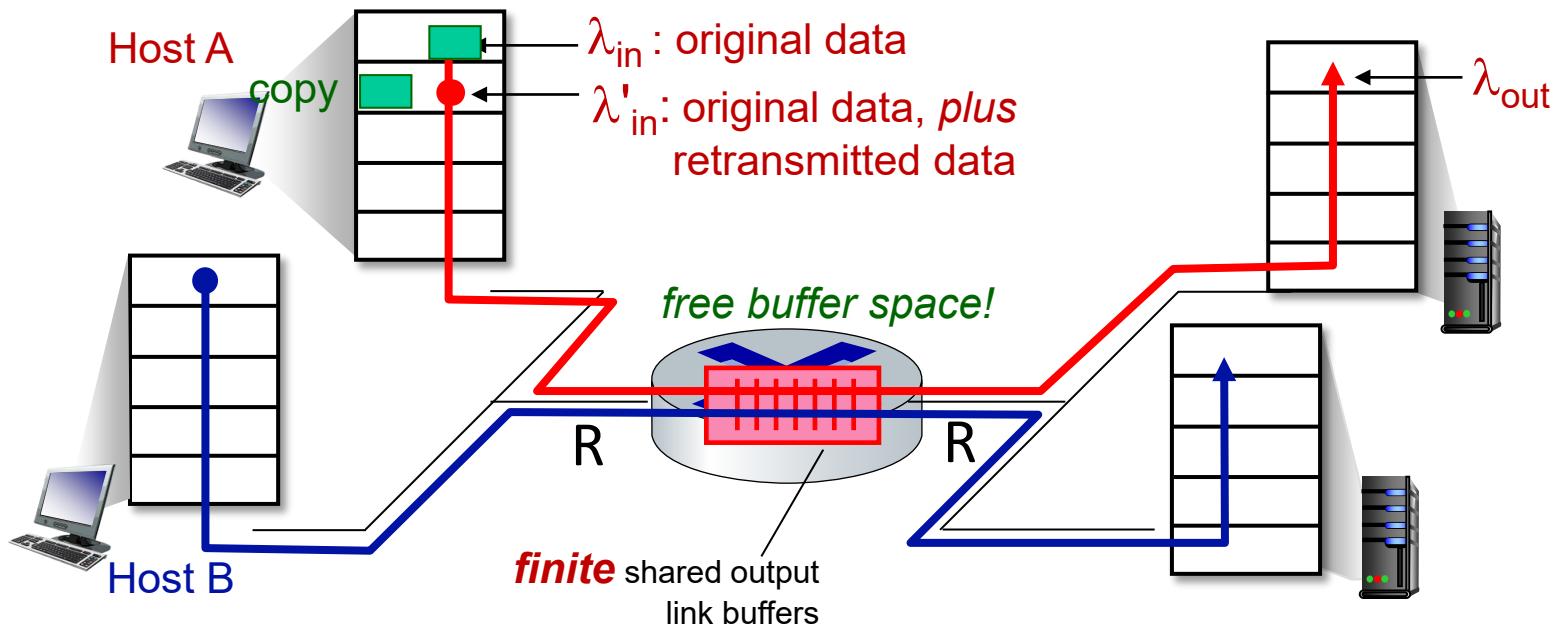
- one router, *finite* buffers
- sender retransmits lost, timed-out packet
  - application-layer input = application-layer output:  $\lambda_{in} = \lambda_{out}$
  - transport-layer input includes *retransmissions* :  $\lambda'_{in} \geq \lambda_{in}$



# Causes/costs of congestion: scenario 2-1

Idealization: perfect knowledge

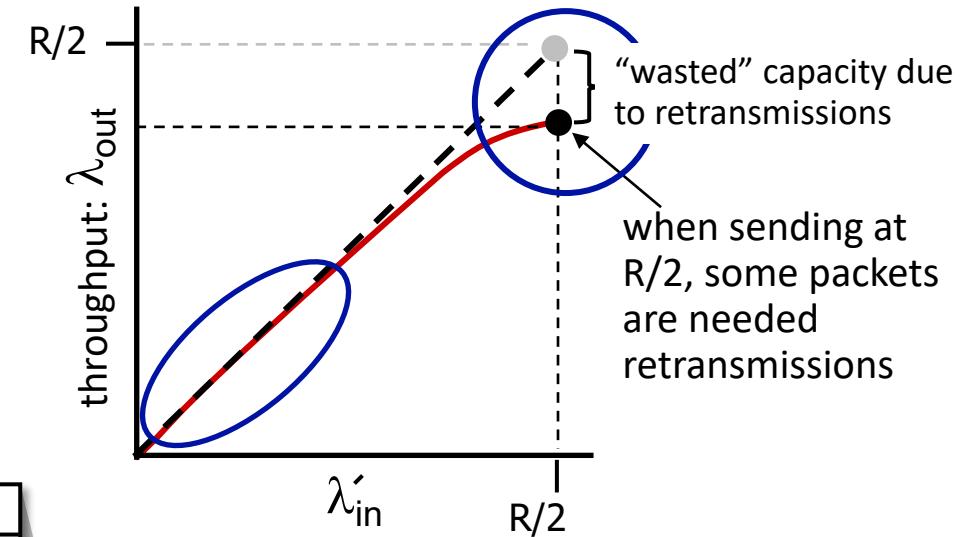
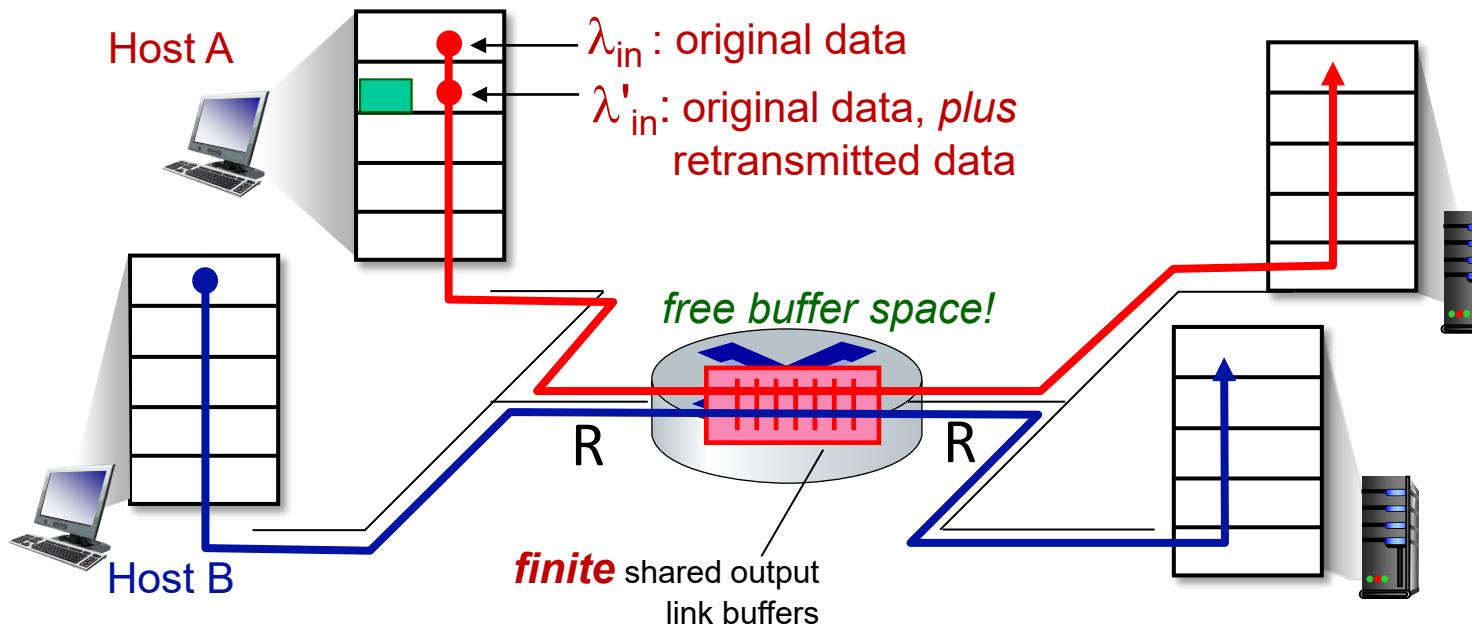
- sender sends only when router buffers available



# Causes/costs of congestion: scenario 2-2

Idealization: *some* perfect knowledge

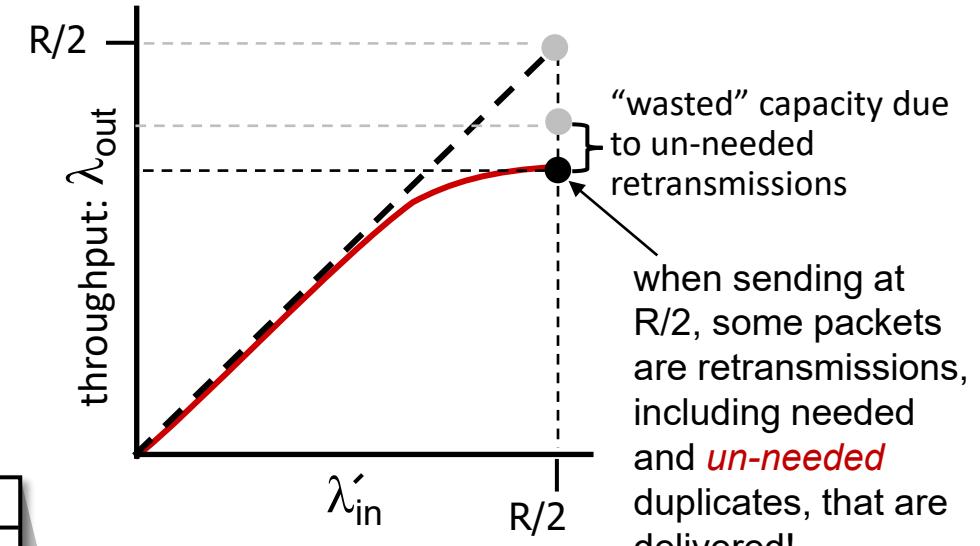
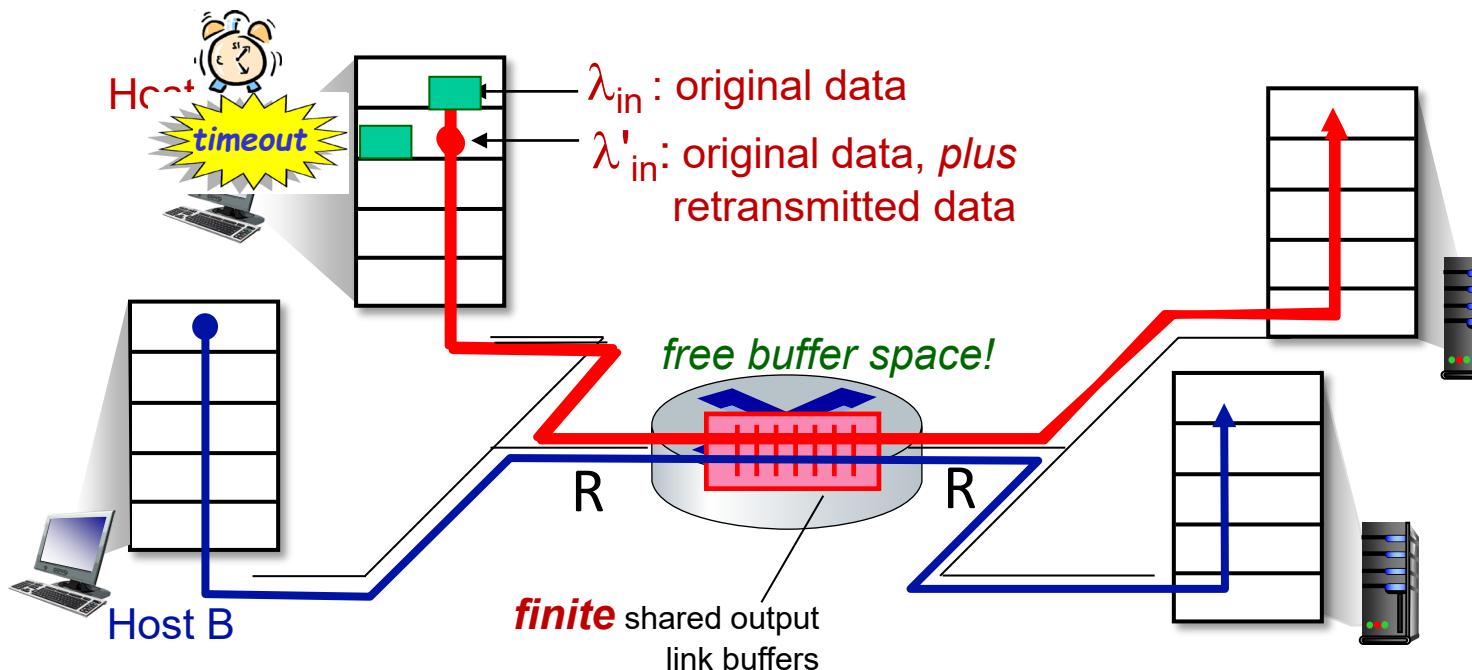
- packets can be lost (dropped at router) due to full buffers
- sender knows when packet has been dropped: only resends if packet *known* to be lost



# Causes/costs of congestion: scenario 2-3

## Realistic scenario: *un-needed duplicates*

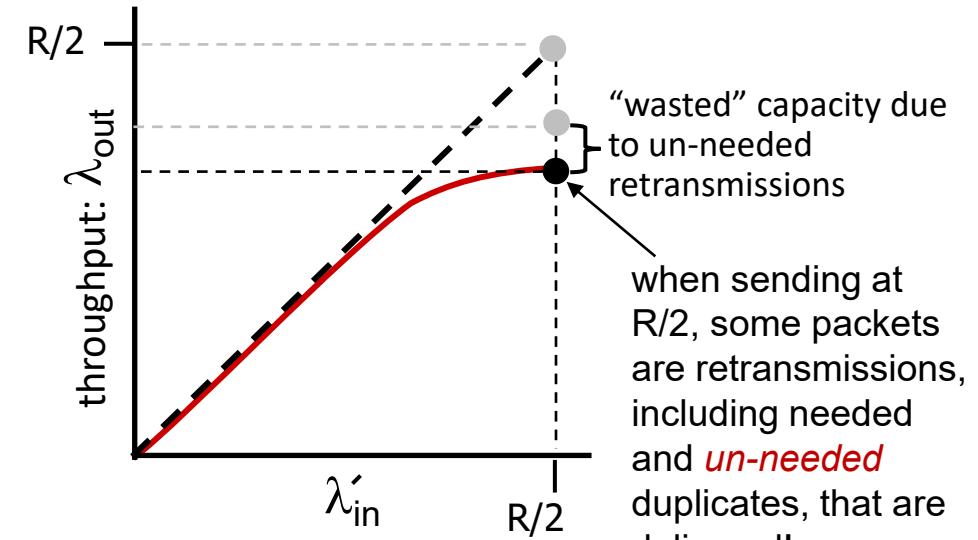
- packets can be lost, dropped at router due to full buffers – requiring retransmissions
- but sender times can time out prematurely, sending *two* copies, *both* of which are delivered



# Causes/costs of congestion: scenario 2-3

## Realistic scenario: *un-needed duplicates*

- packets can be lost, dropped at router due to full buffers – requiring retransmissions
- but sender times can time out prematurely, sending *two* copies, *both* of which are delivered



## "costs" of congestion:

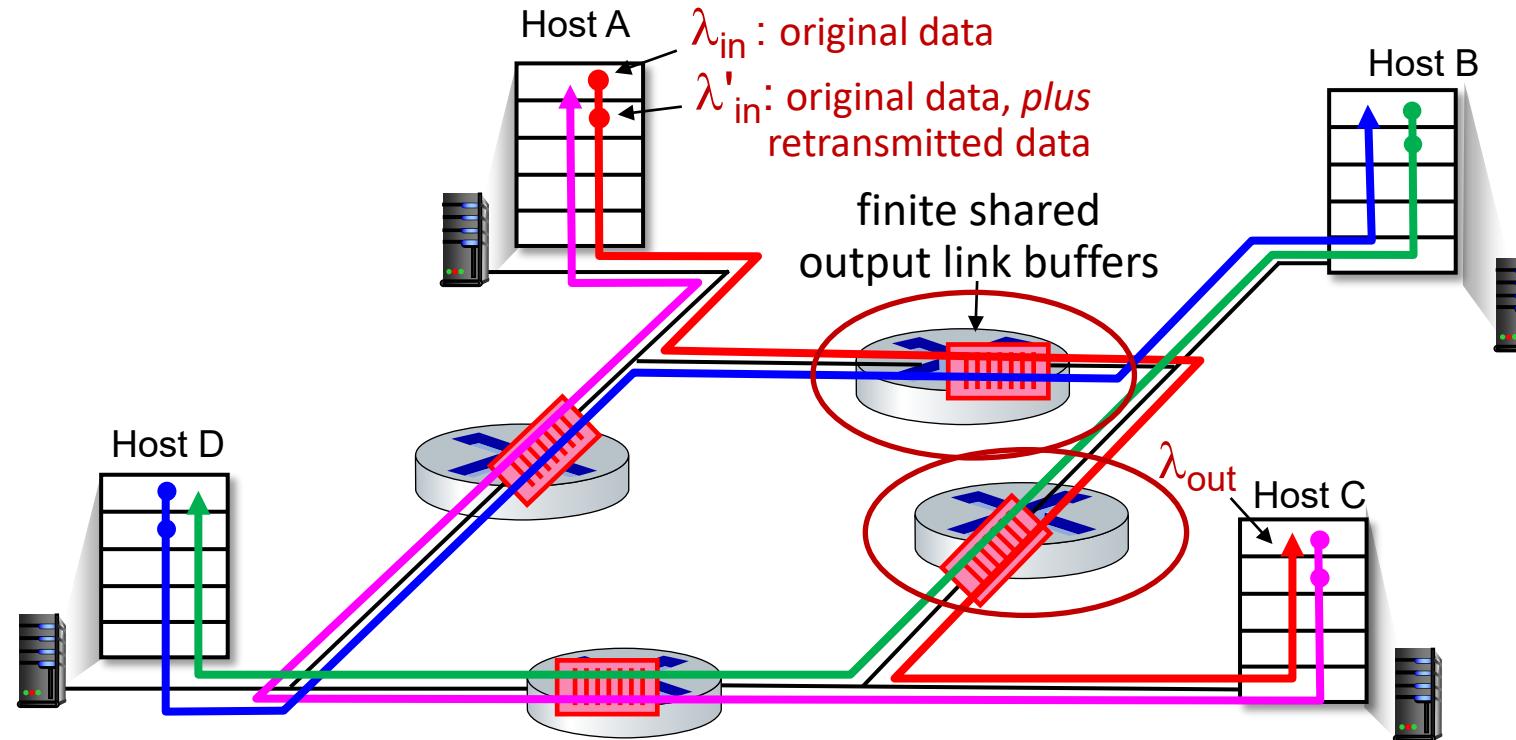
- more work (retransmission) for given receiver throughput
- unneeded retransmissions: link carries multiple copies of a packet
  - decreasing maximum achievable throughput

# Causes/costs of congestion: scenario 3

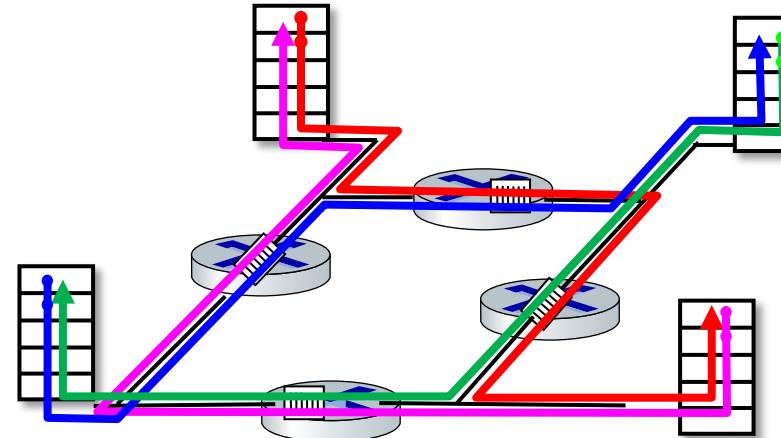
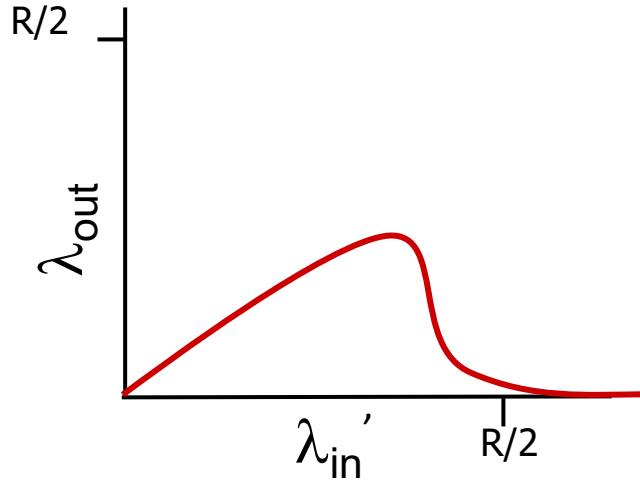
- four senders
- multi-hop paths
- timeout/retransmit

**Q:** what happens as  $\lambda_{in}$  and  $\lambda'_{in}$  increase ?

**A:** as red  $\lambda'_{in}$  increases, all arriving blue pkts at upper queue are dropped, blue throughput  $\rightarrow 0$



# Causes/costs of congestion: scenario 3

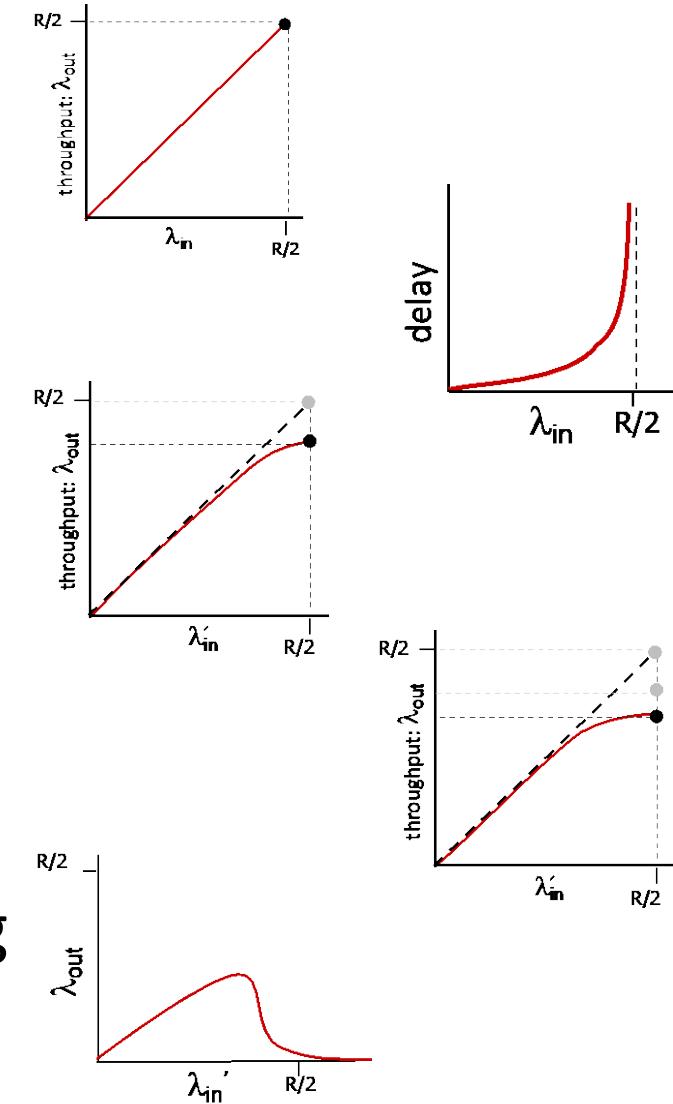


another “cost” of congestion:

- when packet dropped, any upstream transmission capacity and buffering used for that packet was wasted!

# Causes/costs of congestion: insights

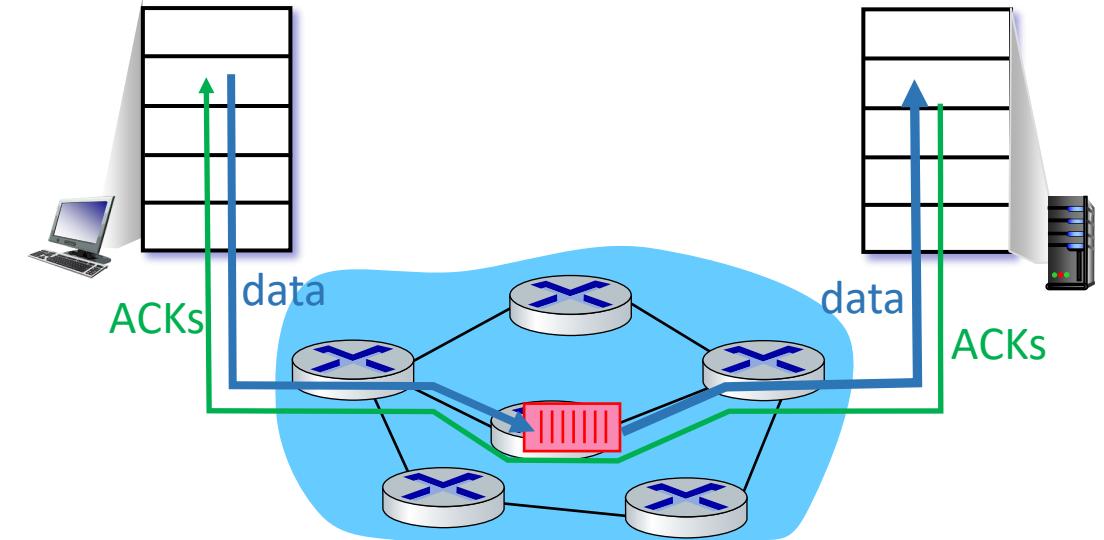
- throughput can never exceed capacity
- delay increases as capacity approached
- loss/retransmission decreases effective throughput
- un-needed duplicates further decreases effective throughput
- upstream transmission capacity / buffering wasted for packets lost downstream



# Approaches towards congestion control

## End-end congestion control:

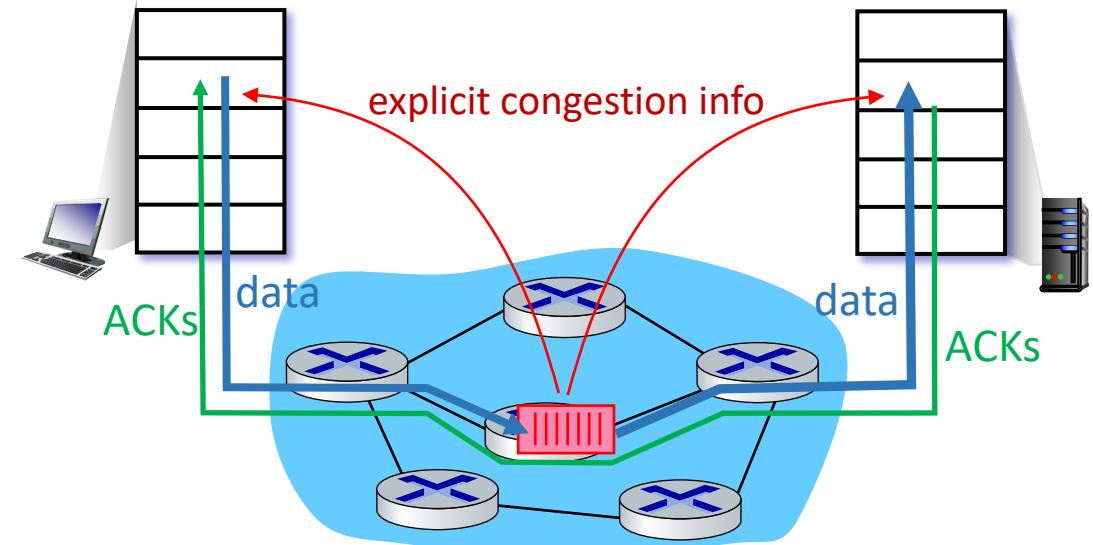
- no explicit feedback from network
- congestion *inferred* from observed loss, delay
- approach taken by TCP



# Approaches towards congestion control

## Network-assisted congestion control:

- routers provide *direct* feedback to sending/receiving hosts with flows passing through congested router
- may indicate congestion level or explicitly set sending rate
- TCP ECN, ATM, DECbit protocols



# Chapter 3: roadmap

- Transport-layer services
- Multiplexing and demultiplexing
- Connectionless transport: UDP
- Principles of reliable data transfer
- Connection-oriented transport: TCP
- Principles of congestion control
- **TCP congestion control**
- Evolution of transport-layer functionality



# TCP congestion control: AIMD

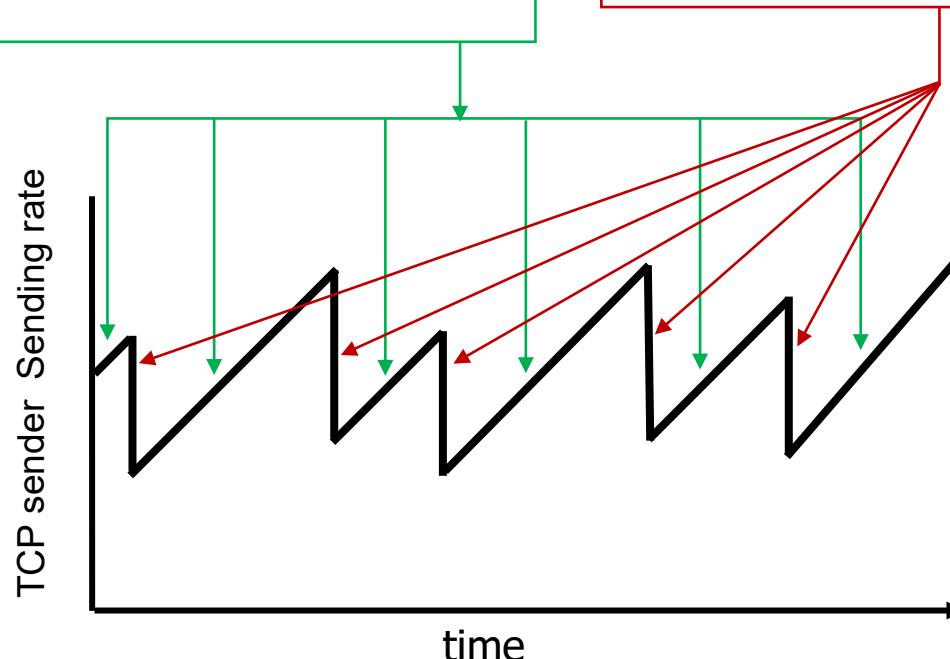
- *approach:* senders can increase sending rate until packet loss (congestion) occurs, then decrease sending rate on loss event

## Additive Increase

increase sending rate by 1 maximum segment size every RTT until loss detected

## Multiplicative Decrease

cut sending rate in half at each loss event



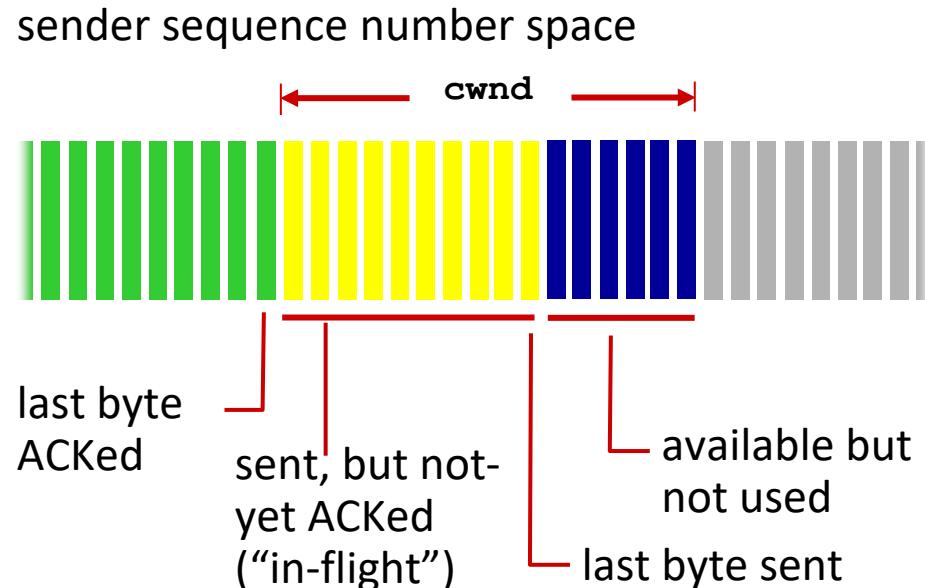
**AIMD** sawtooth behavior: *probing* for bandwidth

# TCP AIMD: more

*Multiplicative decrease* detail: sending rate is

- Cut in half on loss detected by triple duplicate ACK (TCP Reno)
- Cut to 1 MSS (maximum segment size) when loss detected by timeout (TCP Tahoe)

# TCP congestion control: details



TCP sending behavior:

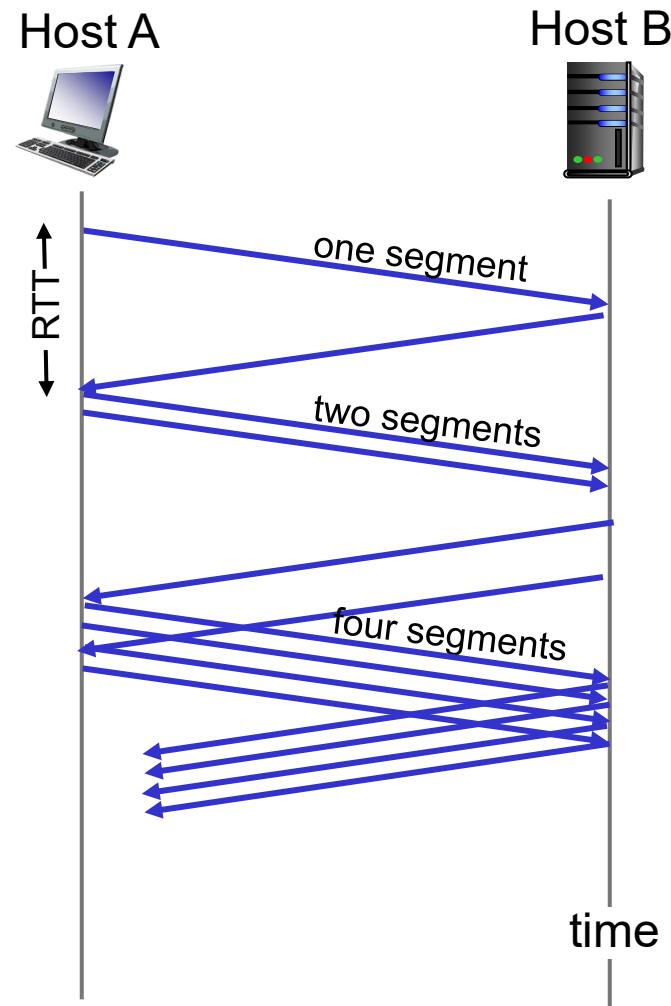
- *roughly*: send  $cwnd$  bytes, wait RTT for ACKS, then send more bytes

$$\text{TCP rate} \approx \frac{cwnd}{RTT} \text{ bytes/sec}$$

- TCP sender limits transmission:  $\text{LastByteSent} - \text{LastByteAcked} \leq cwnd$
- $cwnd$  is dynamically adjusted in response to observed network congestion (implementing TCP congestion control)

# TCP slow start

- when connection begins, increase rate exponentially until first loss event:
  - initially **cwnd** = 1 MSS
  - double **cwnd** every RTT
  - done by incrementing **cwnd** for every ACK received
- *summary:* initial rate is slow, but ramps up exponentially fast



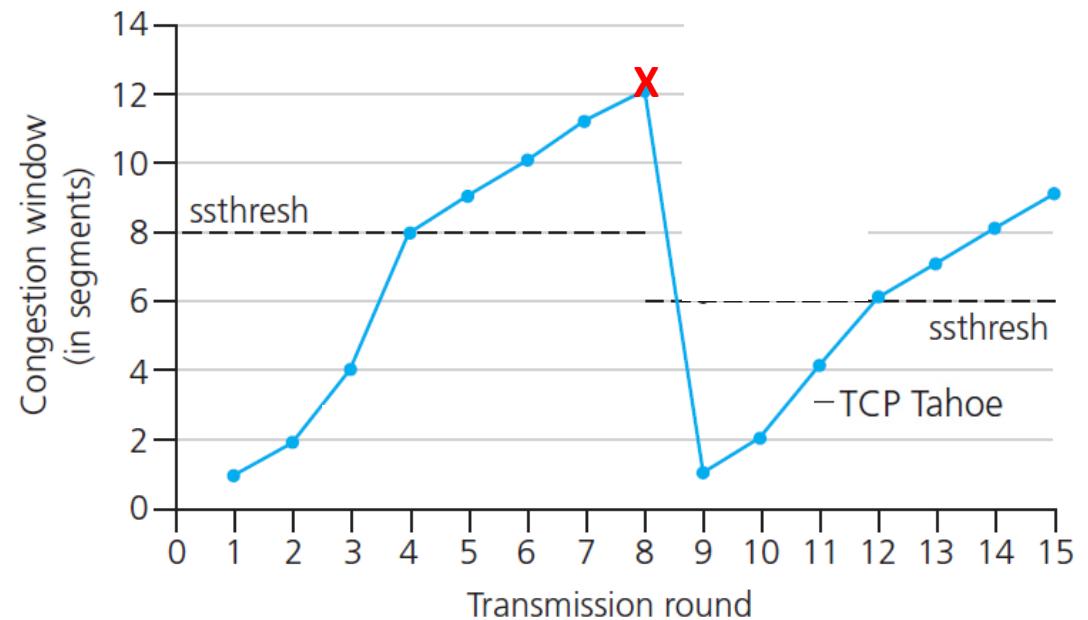
# TCP: from slow start to congestion avoidance

**Q:** when should the exponential increase switch to linear?

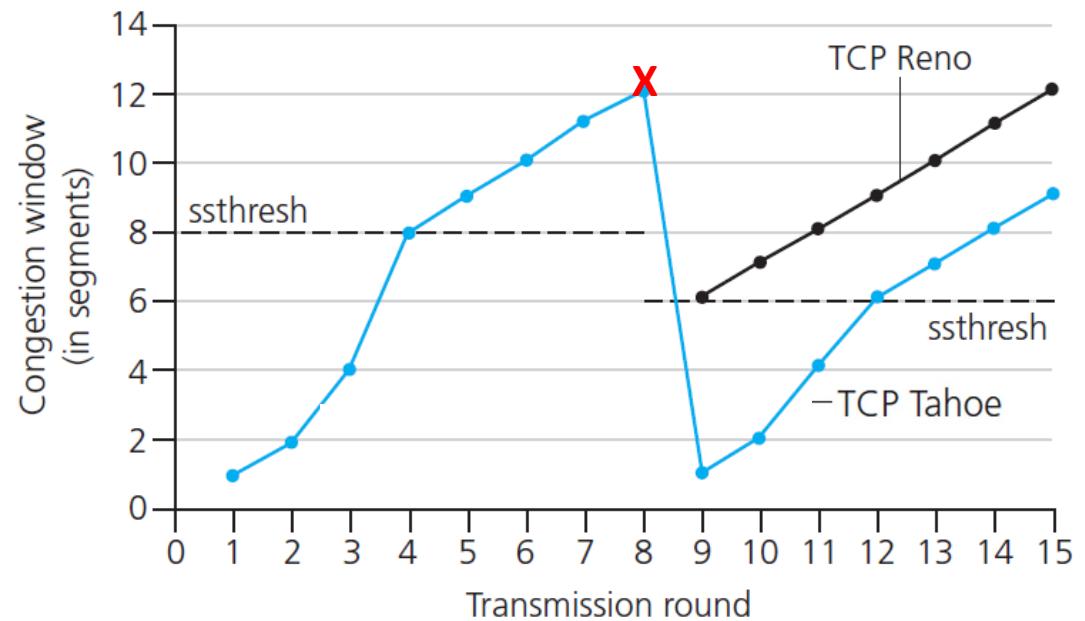
**A:** when **cwnd** gets to 1/2 of its value before timeout.

## Implementation:

- variable **ssthresh**
- on loss event, **ssthresh** is set to 1/2 of **cwnd** just before loss event



# TCP Tahoe & Reno





# Computer Networks

Lecturer: ZHANG Ying  
Fall semester 2022

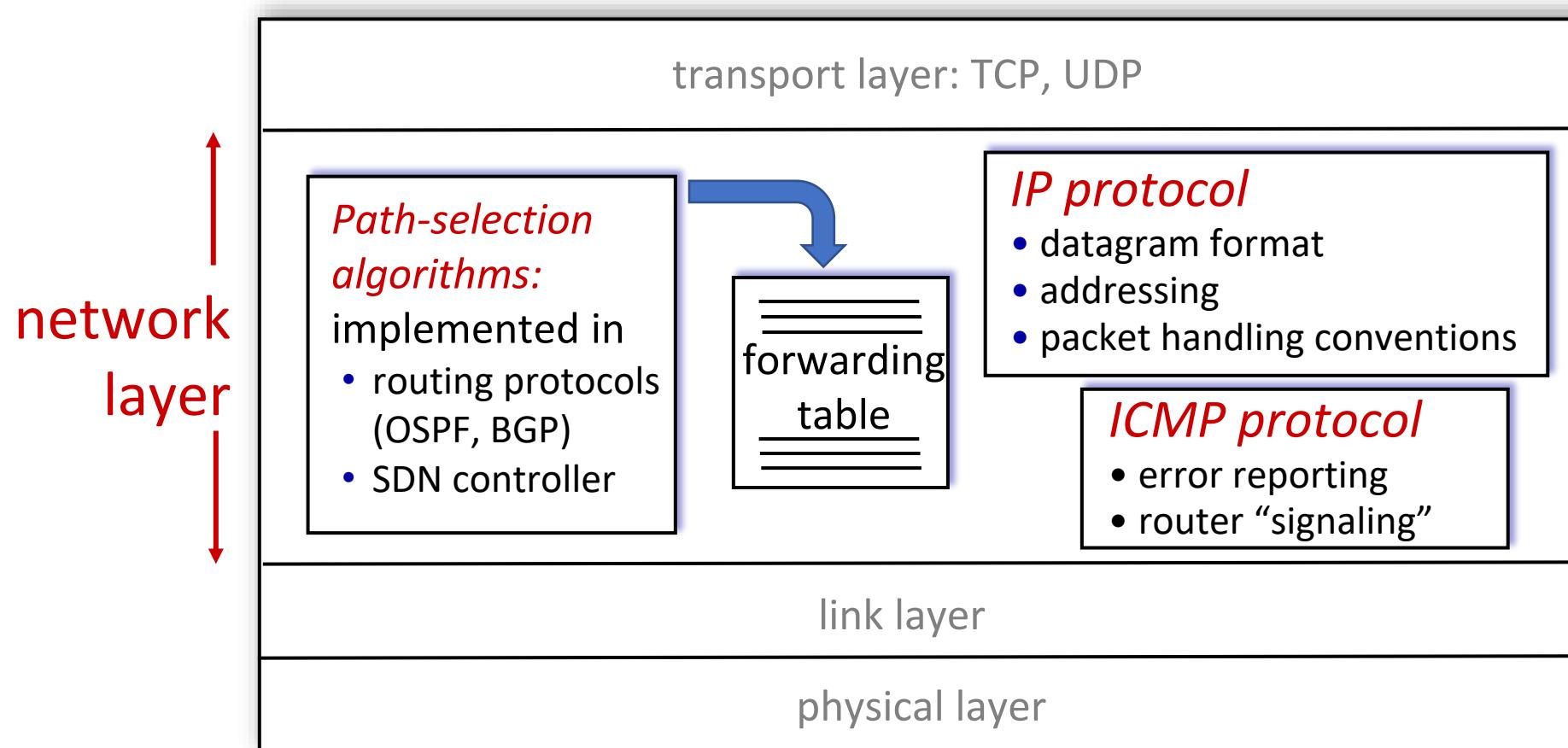
# Network layer: “data plane” roadmap

- Network layer: overview
  - data plane
  - control plane
- What's inside a router
  - input ports, switching, output ports
  - buffer management, scheduling
- IP: the Internet Protocol
  - datagram format
  - addressing
  - network address translation
  - IPv6

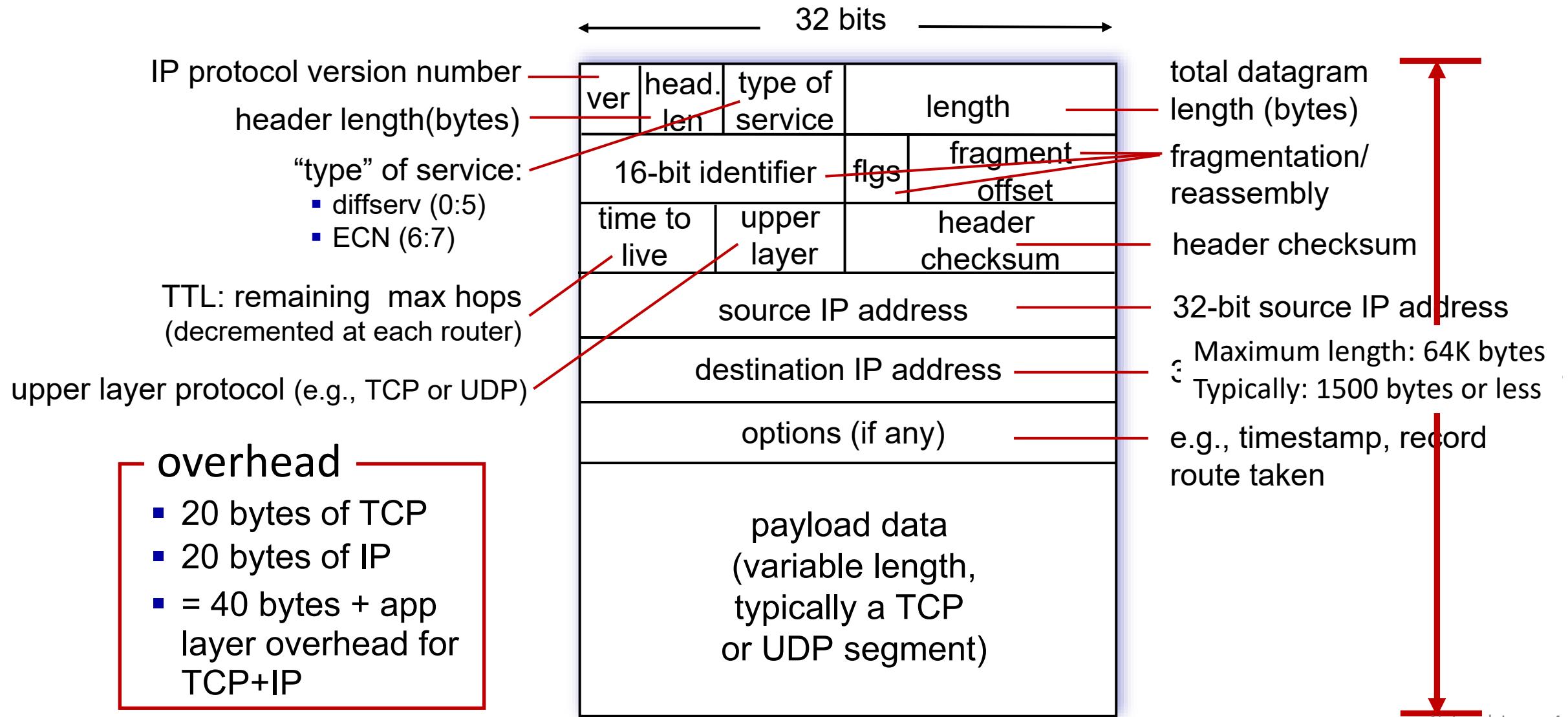


# Network Layer: Internet

host, router network layer functions:

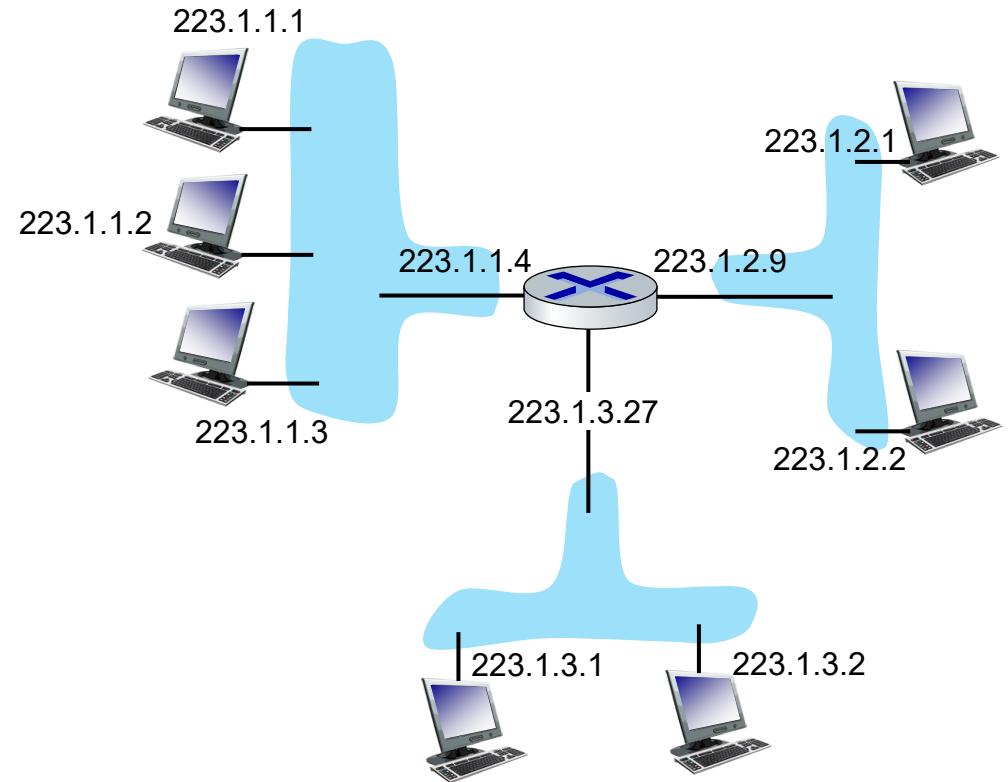


# IP Datagram format



# IP addressing: introduction

- **IP address:** 32-bit identifier associated with each host or 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)



dotted-decimal IP address notation:

223.1.1.1 =  $\begin{array}{cccc} 11011111 & 00000001 & 00000001 & 00000001 \end{array}$

223 1 1 1  
Network Layer: 4-5

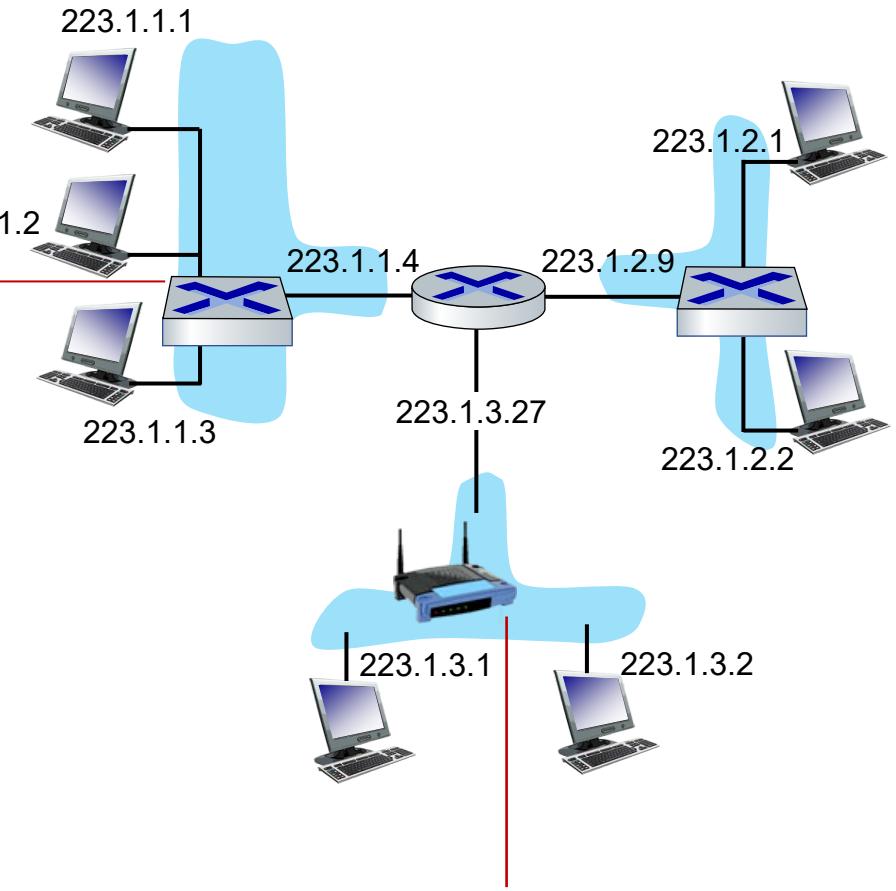
# IP addressing: introduction

**Q:** how are interfaces actually connected?

**A:** we'll learn about that in chapters 6, 7

*For now:* don't need to worry about how one interface is connected to another (with no intervening router)

**A:** wired Ethernet interfaces connected by Ethernet switches



**A:** wireless WiFi interfaces connected by WiFi base station

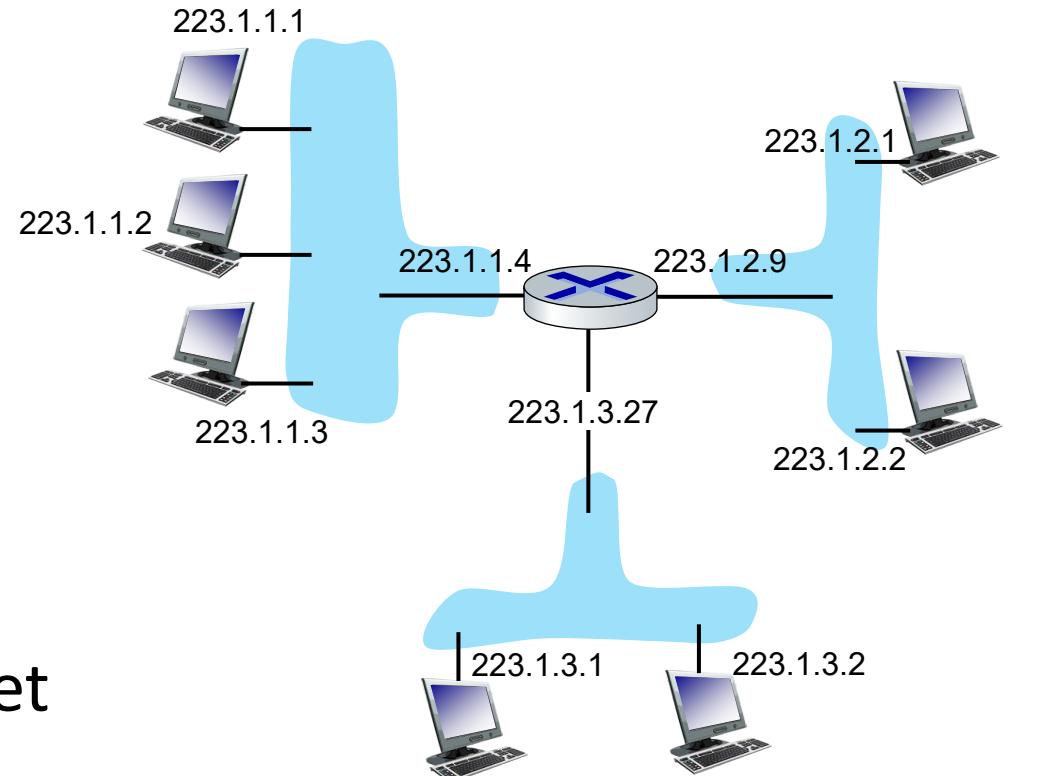
# Subnets-1

- *What's a subnet ?*

- device interfaces that can physically reach each other **without passing through an intervening router**

- IP addresses have structure:

- **subnet part:** devices in same subnet have common high order bits
- **host part:** remaining low order bits

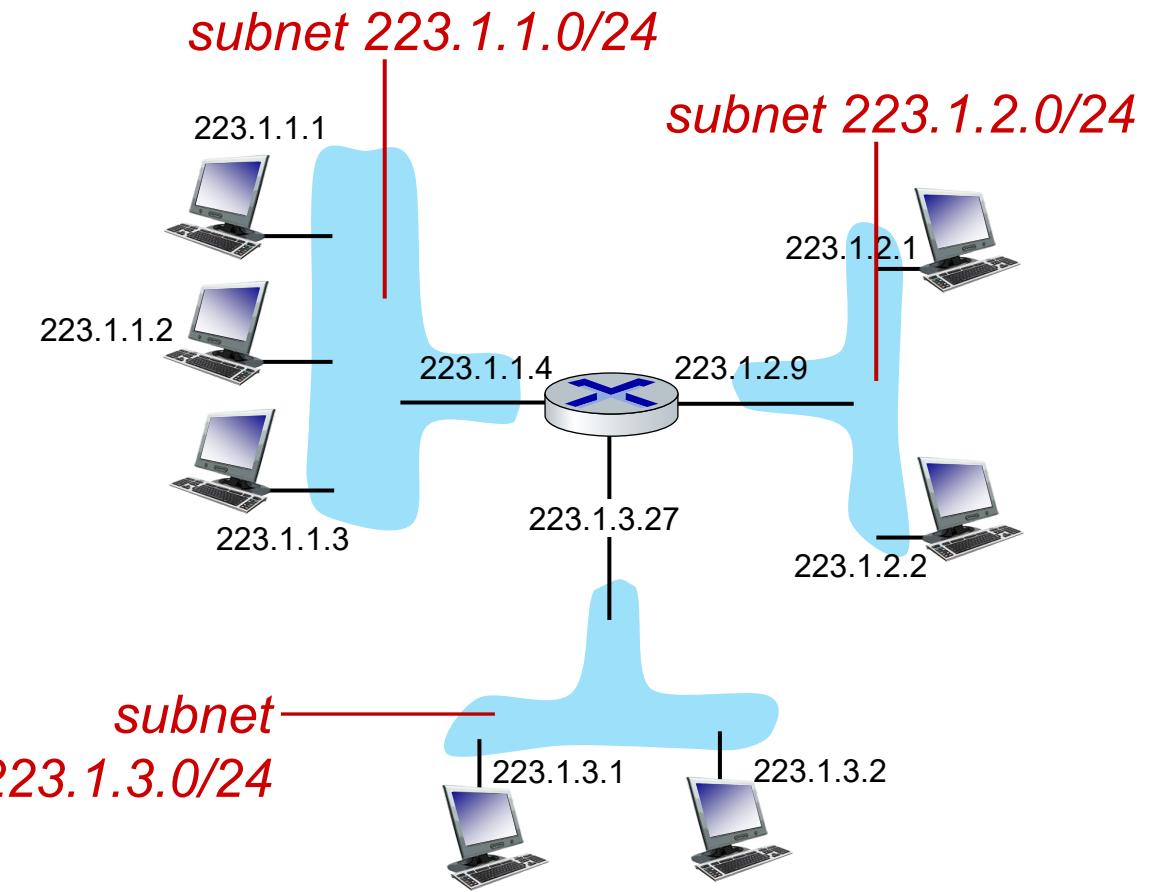


network consisting of 3 subnets

# Subnets-2

*Recipe for defining subnets:*

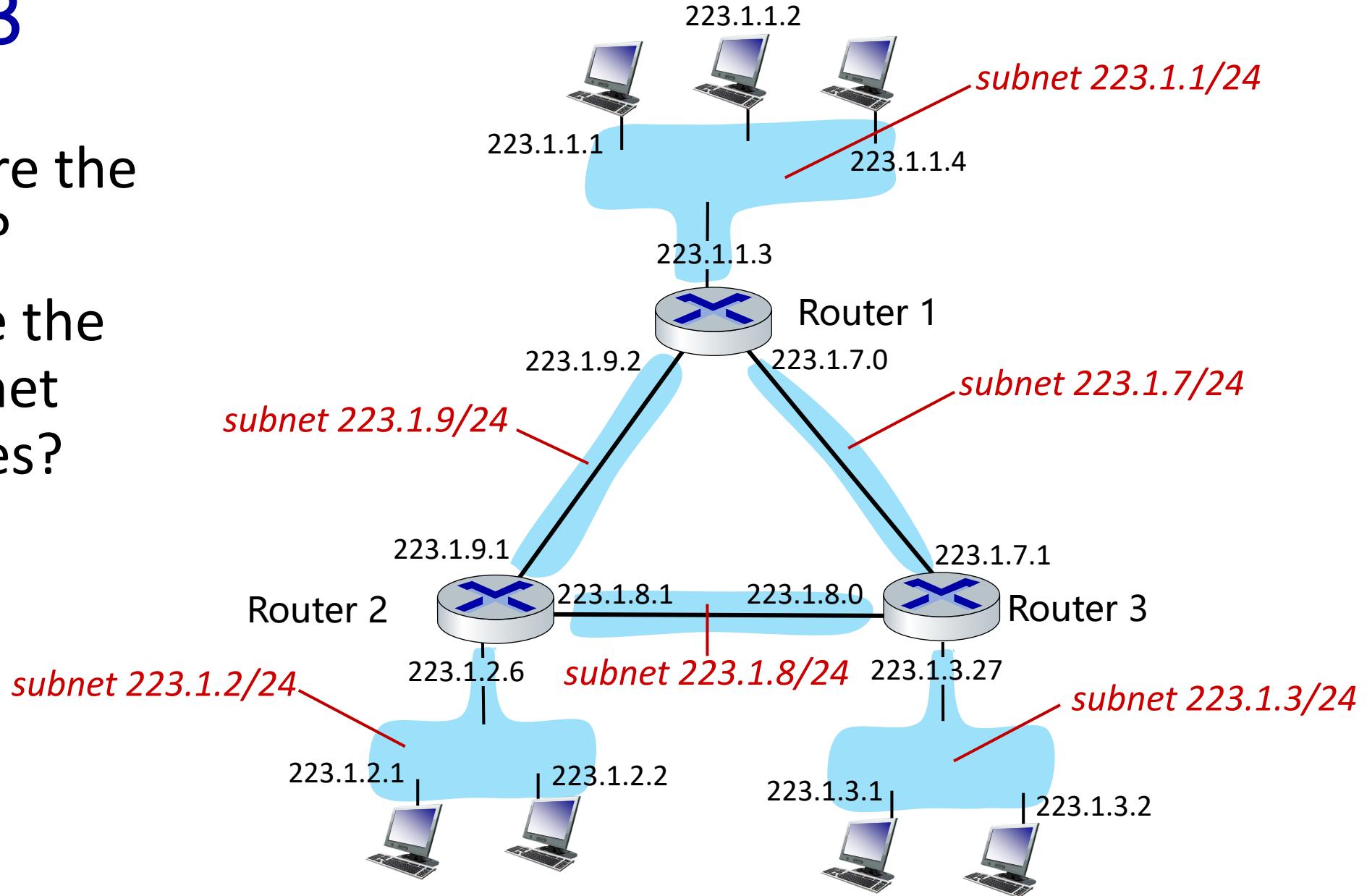
- detach each interface from its host or router, creating “islands” of isolated networks
- each isolated network is called a *subnet*



subnet mask: /24  
(high-order 24 bits: subnet part of IP address)

# Subnets-3

- where are the subnets?
- what are the /24 subnet addresses?



# IP addressing: CIDR

# CIDR: Classless InterDomain Routing (pronounced “cider”)

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



# IP addresses: how to get one?

That's actually **two** questions:

1. Q: How does a *host* get IP address within its network (host part of address)?
2. Q: How does a *network* get IP address for itself (network part of address)

How does *host* get IP address?

- hard-coded by sysadmin in config file (e.g., `/etc/rc.config` in UNIX)
- **DHCP: Dynamic Host Configuration Protocol:** dynamically get address from server
  - “plug-and-play”

# DHCP: Dynamic Host Configuration Protocol

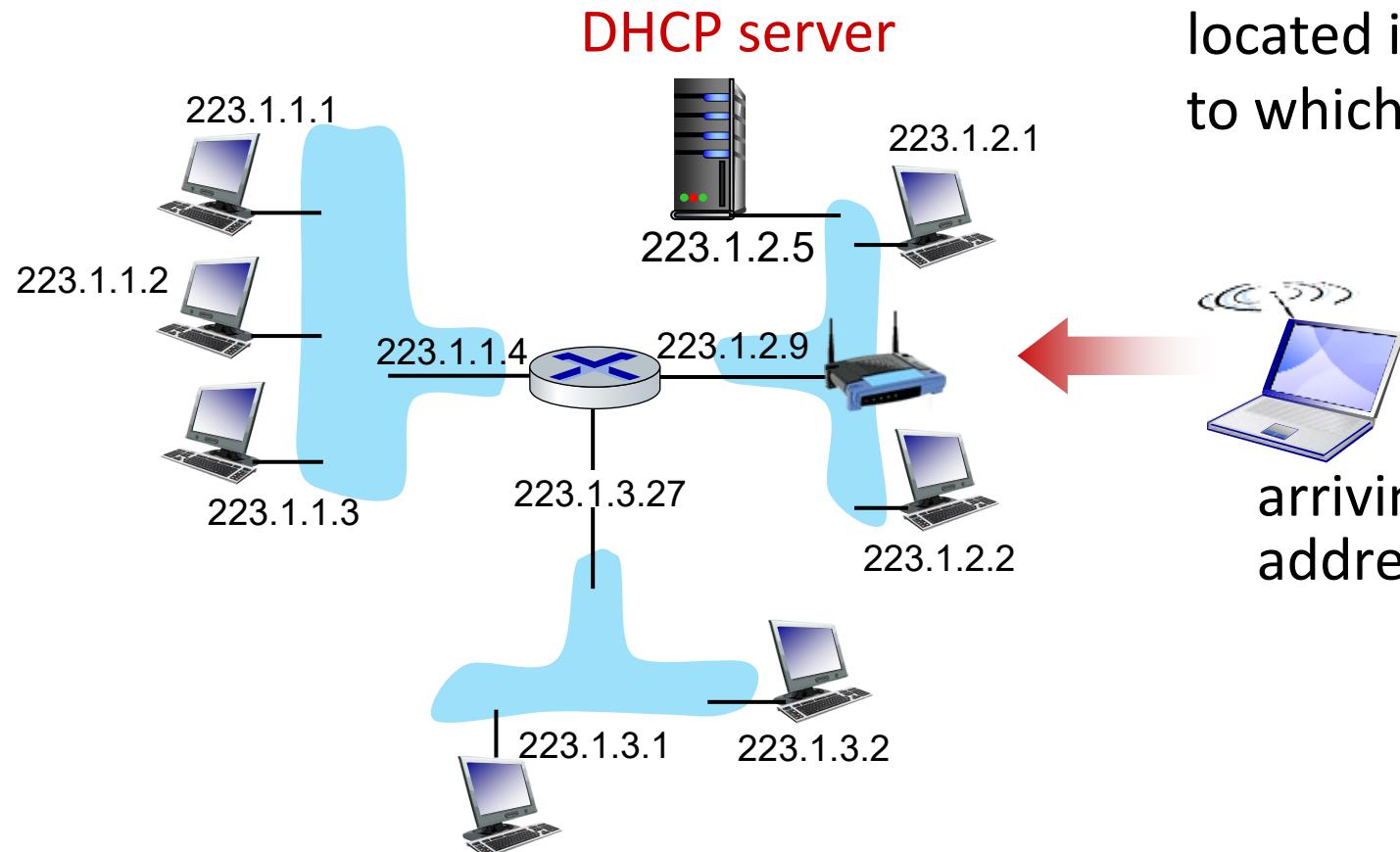
**goal:** host *dynamically* obtains IP address from network server when it “joins” network

- can renew its lease on address in use
- allows reuse of addresses (only hold address while connected/on)
- support for mobile users who join/leave network

## DHCP overview:

- host broadcasts **DHCP discover** msg [optional]
- DHCP server responds with **DHCP offer** msg [optional]
- host requests IP address: **DHCP request** msg
- DHCP server sends address: **DHCP ack** msg

# DHCP client-server scenario

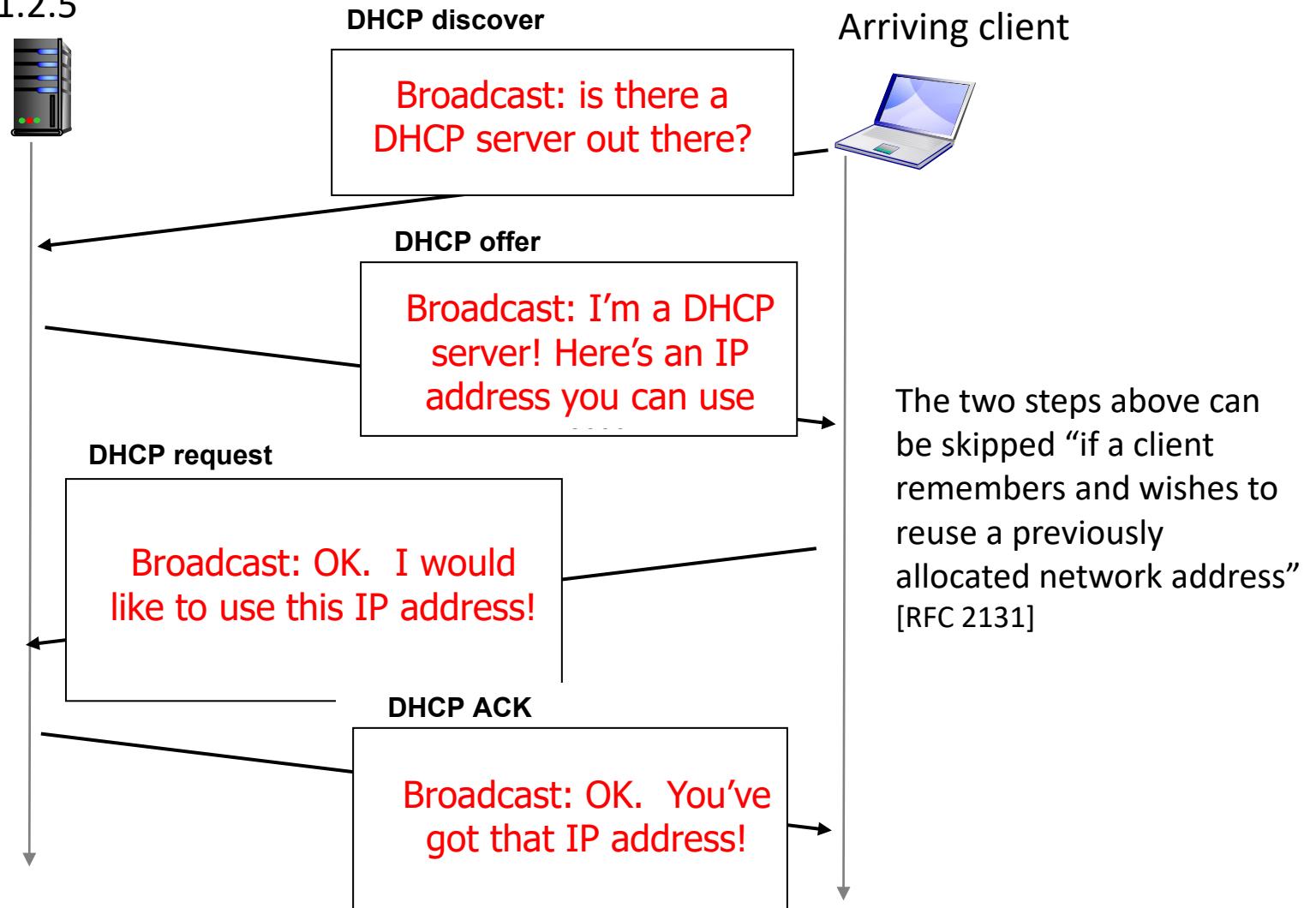


Typically, DHCP server will be co-located in router, serving all subnets to which router is attached

arriving **DHCP client** needs address in this network

# DHCP client-server scenario

DHCP server: 223.1.2.5



# DHCP: more than IP addresses

DHCP can return more than just allocated IP address on subnet:

- address of first-hop router for client
- name and IP address of DNS sever
- network mask (indicating network versus host portion of address)

# IP addresses: how to get one?

**Q:** how does *network* get subnet part of IP address?

**A:** gets allocated portion of its provider ISP's address space

|             |   |                |
|-------------|---|----------------|
| ISP's block | <u>11001000</u> <u>00010111</u> <u>00010000</u> <u>00000000</u> | 200.23.16.0/20 |
|-------------|---|----------------|

ISP can then allocate out its address space in 8 blocks:

|                |   |                |
|----------------|---|----------------|
| Organization 0 | <u>11001000</u> <u>00010111</u> <u>00010000</u> <u>00000000</u> | 200.23.16.0/23 |
|----------------|---|----------------|

|                |   |                |
|----------------|---|----------------|
| Organization 1 | <u>11001000</u> <u>00010111</u> <u>00010010</u> <u>00000000</u> | 200.23.18.0/23 |
|----------------|---|----------------|

|                |   |                |
|----------------|---|----------------|
| Organization 2 | <u>11001000</u> <u>00010111</u> <u>00010100</u> <u>00000000</u> | 200.23.20.0/23 |
|----------------|---|----------------|

...

.....

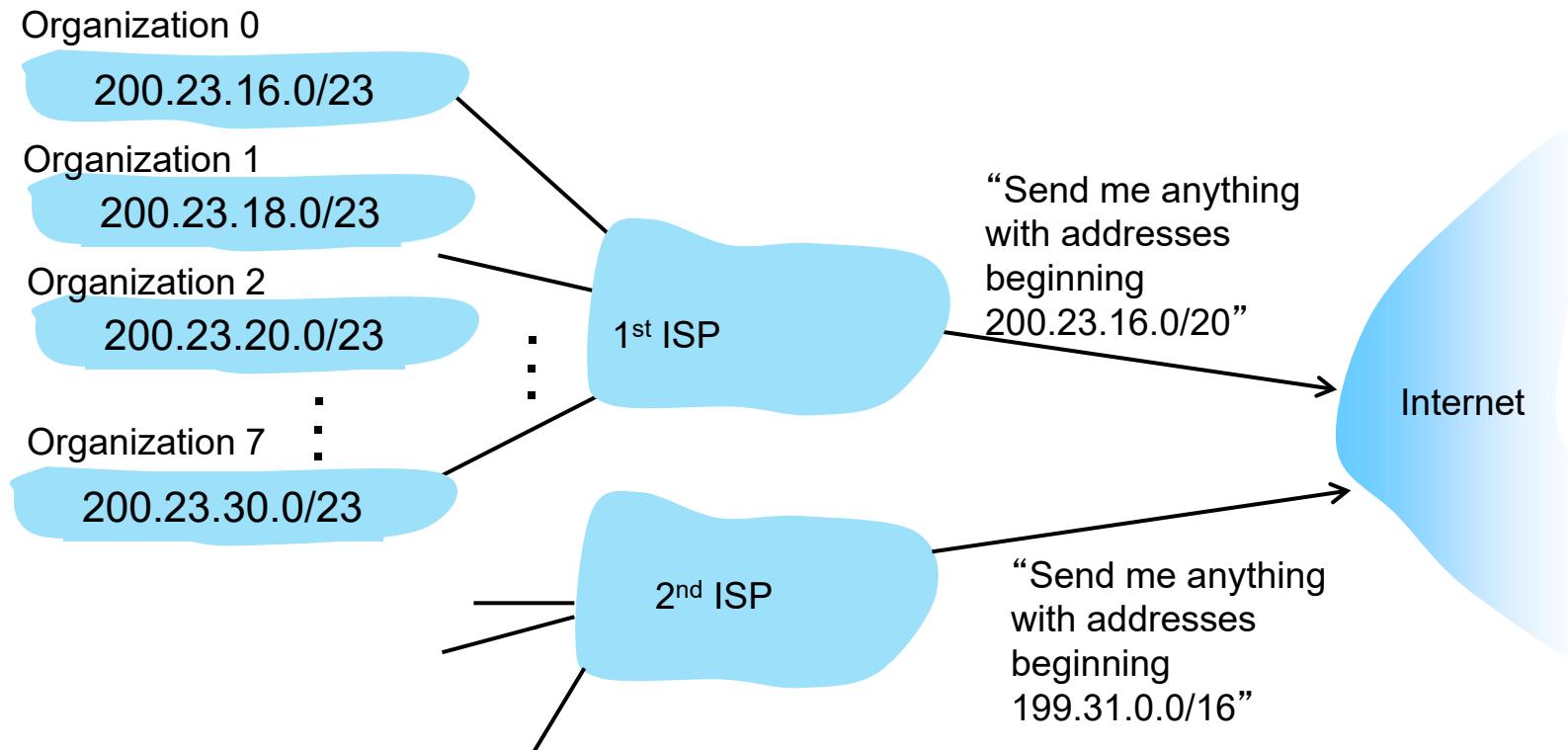
.....

....

|                |   |                |
|----------------|---|----------------|
| Organization 7 | <u>11001000</u> <u>00010111</u> <u>00011110</u> <u>00000000</u> | 200.23.30.0/23 |
|----------------|---|----------------|

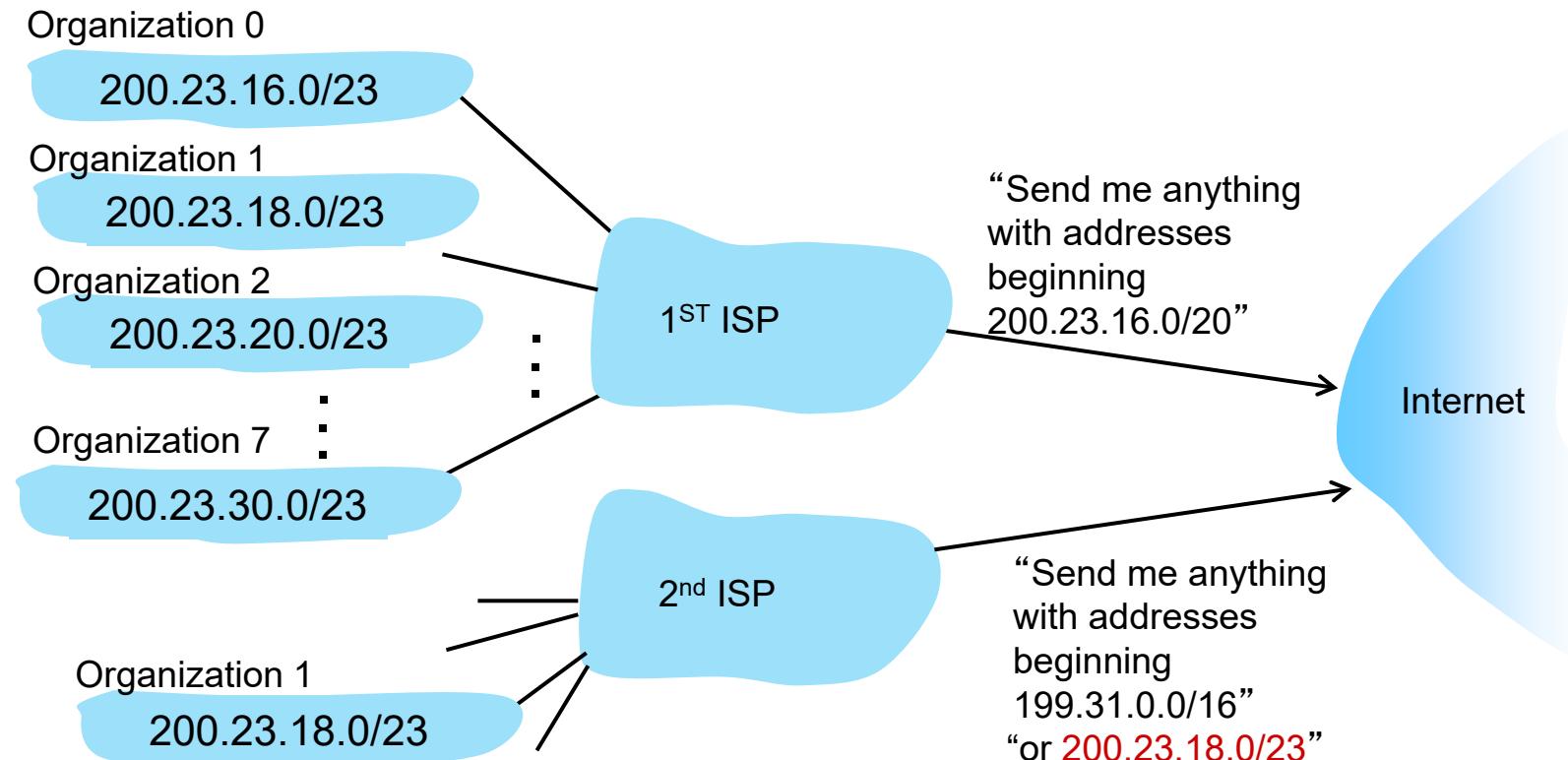
# Hierarchical addressing: route aggregation

hierarchical addressing allows efficient advertisement of routing information:



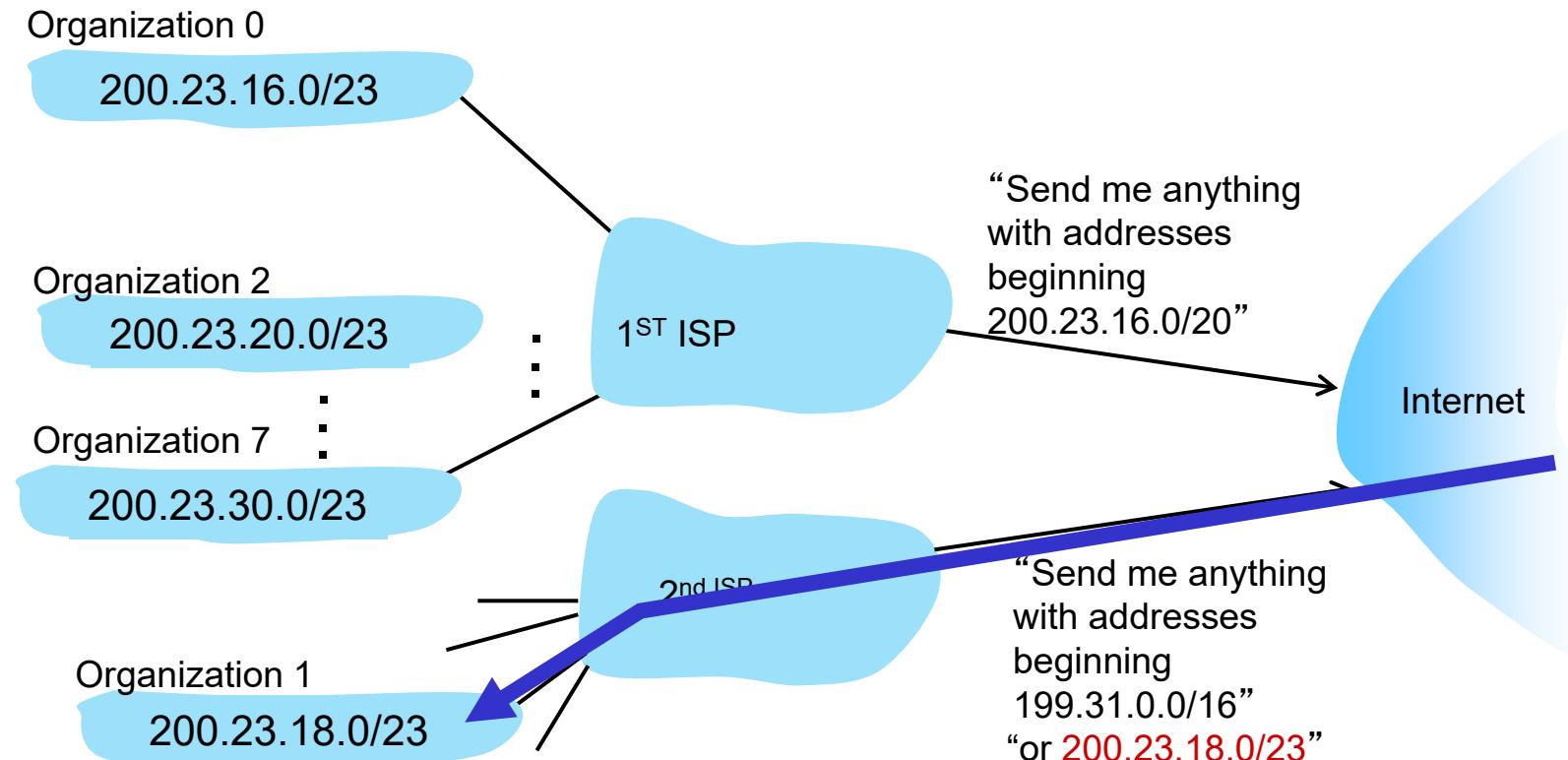
# Hierarchical addressing: more specific routes

- Organization 1 moves from 1<sup>st</sup> ISP to 2<sup>nd</sup> ISP
- 2<sup>nd</sup> ISP now advertises a more specific route to Organization 1



# Hierarchical addressing: more specific routes

- Organization 1 moves from 1<sup>st</sup> ISP to 2<sup>nd</sup> ISP
- 2<sup>nd</sup> ISP now advertises a more specific route to Organization 1



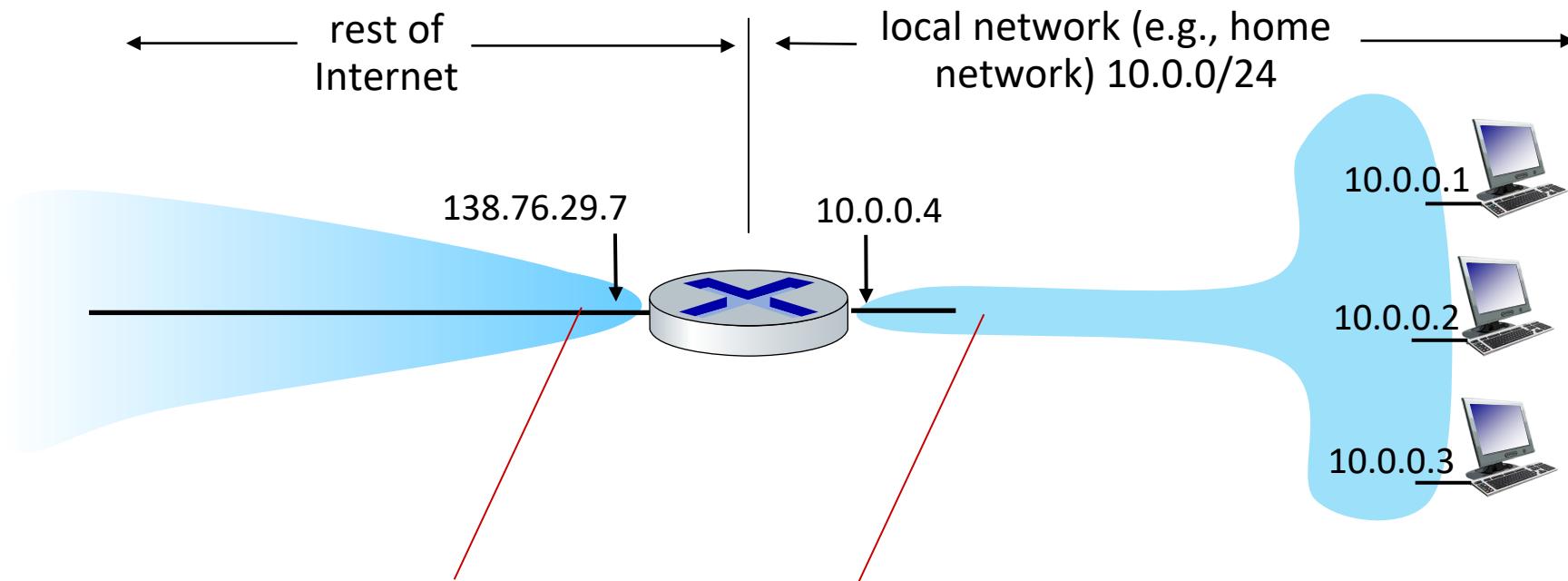
# Network layer: “data plane” roadmap

- Network layer: overview
  - data plane
  - control plane
- What's inside a router
  - input ports, switching, output ports
  - buffer management, scheduling
- IP: the Internet Protocol
  - datagram format
  - addressing
  - network address translation
  - IPv6



# NAT: network address translation

**NAT:** all devices in local network share just **one** IPv4 address as far as outside world is concerned



*all* datagrams *leaving* local network have *same* source NAT IP address: 138.76.29.7, but *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

- all devices in local network have 32-bit addresses in a “private” IP address space (10/8, 172.16/12, 192.168/16 prefixes) that can only be used in local network
- advantages:
  - just **one** IP address needed from provider ISP for ***all*** devices
  - can change addresses of host in local network without notifying outside world
  - can change ISP without changing addresses of devices in local network
  - security: devices inside local net not directly addressable, visible by outside world

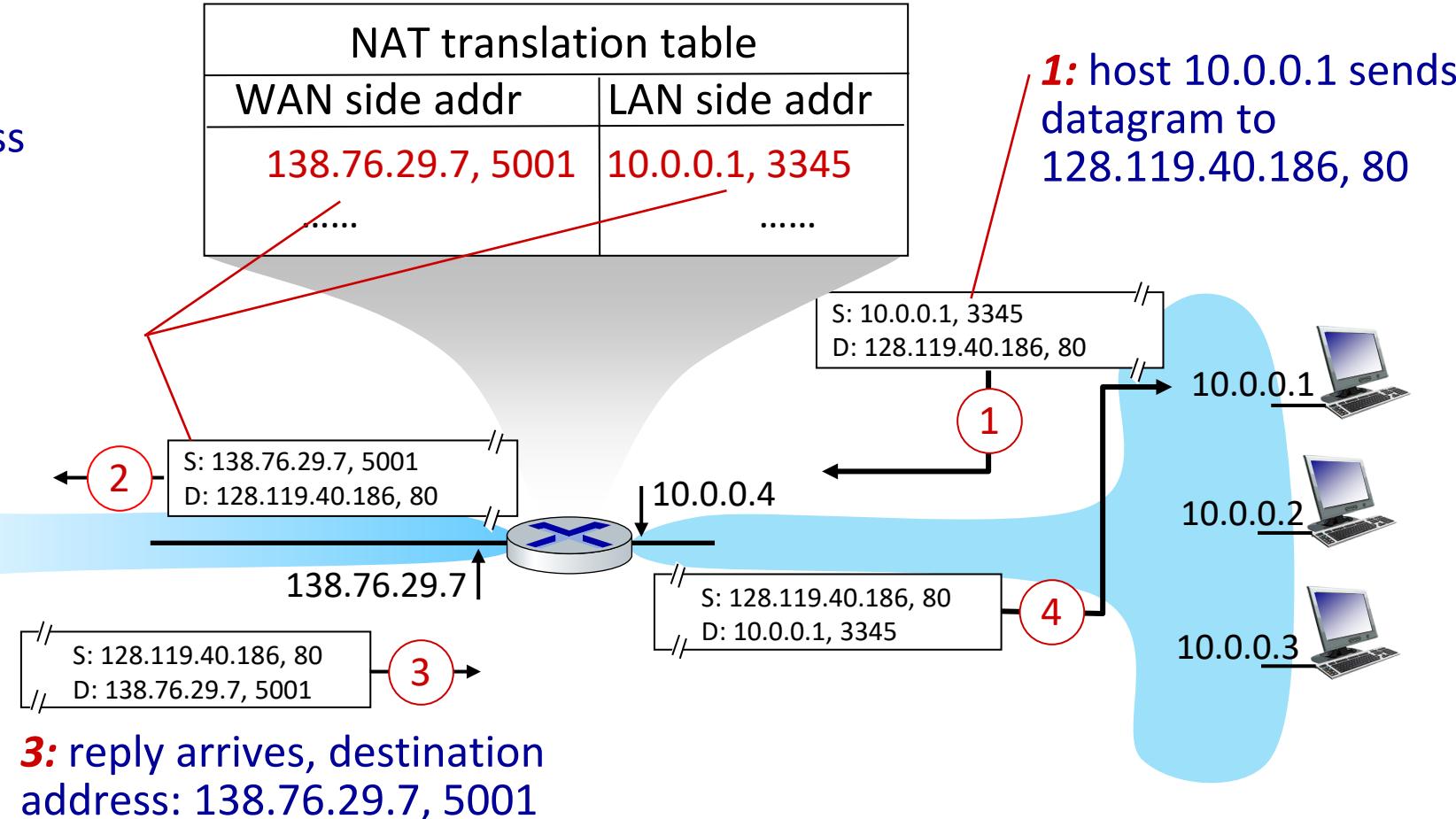
# NAT: network address translation

implementation: NAT router must (transparently):

- 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 address
- 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 destination 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 address from 10.0.0.1, 3345 to 138.76.29.7, 5001, updates table



# NAT: network address translation

- NAT has been controversial:
  - routers “should” only process up to layer 3
  - address “shortage” should be solved by IPv6
  - violates end-to-end argument (port # manipulation by network-layer device)
  - NAT traversal: what if client wants to connect to server behind NAT?
- but NAT is here to stay:
  - extensively used in home and institutional nets, 4G/5G cellular nets

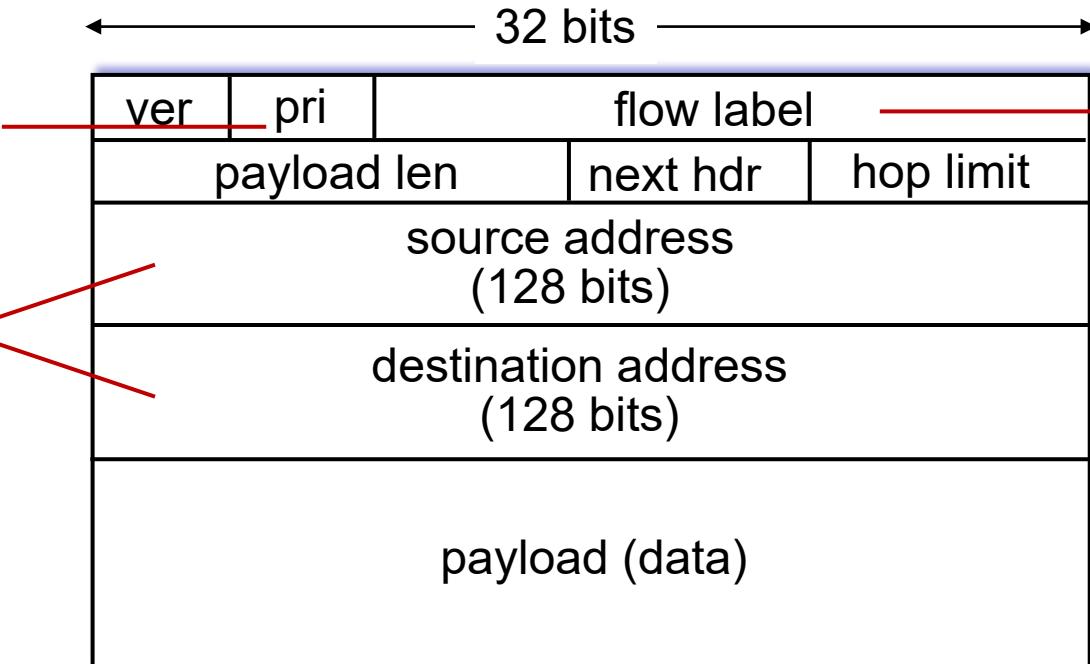
# IPv6: motivation

- **initial motivation:** 32-bit IPv4 address space would be completely allocated
- additional motivation:
  - speed processing/forwarding: 40-byte fixed length header
  - enable different network-layer treatment of “flows”

# IPv6 datagram format

**priority:** identify priority among datagrams in flow

**128-bit IPv6 addresses**



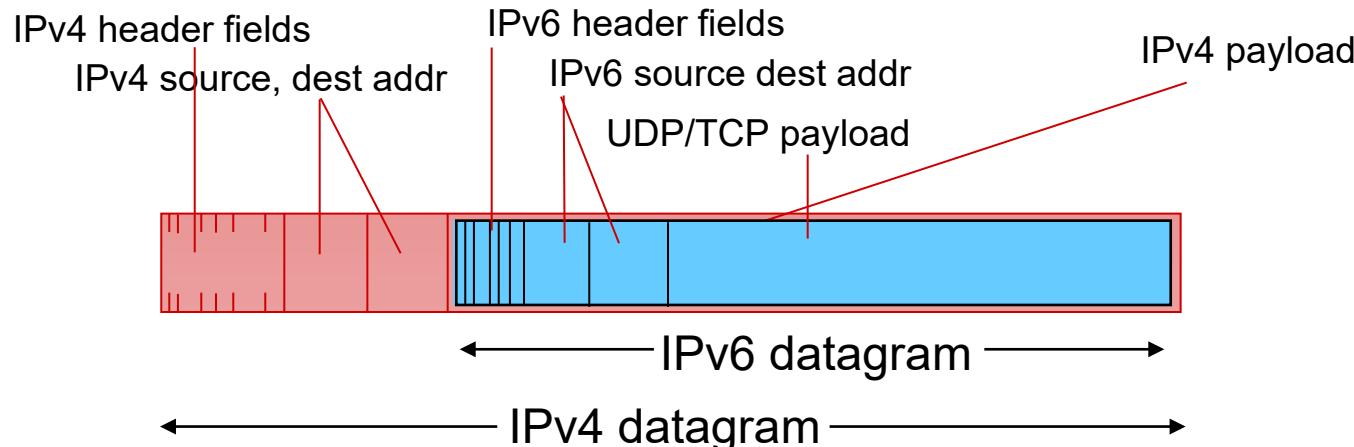
**flow label:** identify datagrams in same "flow." (concept of "flow" not well defined.).

What's missing (compared with IPv4):

- no checksum (to speed processing at routers)
- no fragmentation/reassembly
- no options (available as upper-layer, next-header protocol at router)

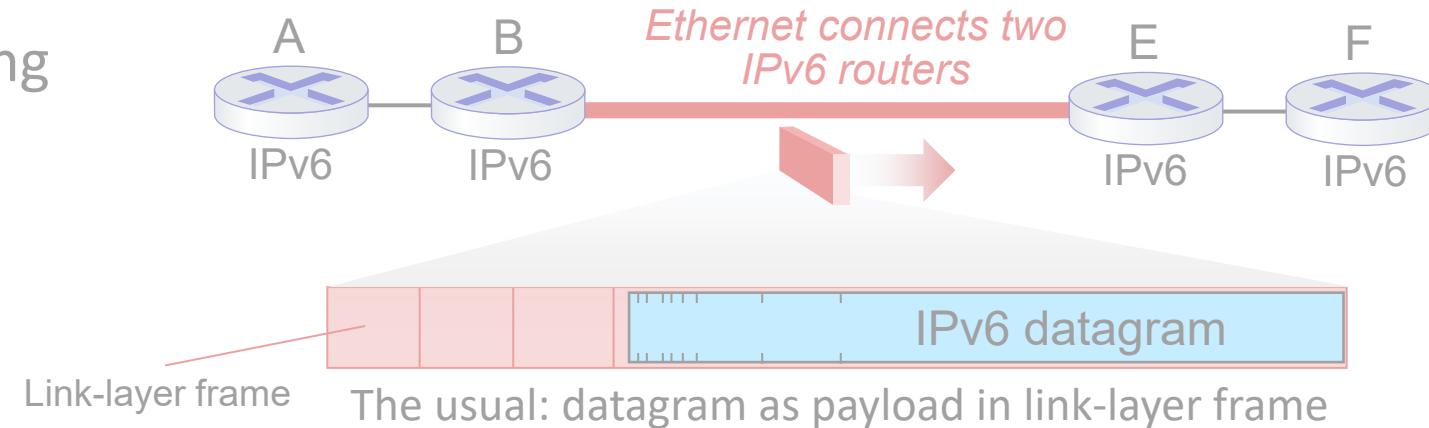
# Transition from IPv4 to IPv6

- not all routers can be upgraded simultaneously
  - no “flag days”
  - how will network operate with mixed IPv4 and IPv6 routers?
- **tunneling:** IPv6 datagram carried as *payload* in IPv4 datagram among IPv4 routers (“packet within a packet”)
  - tunneling used extensively in other contexts (4G/5G)

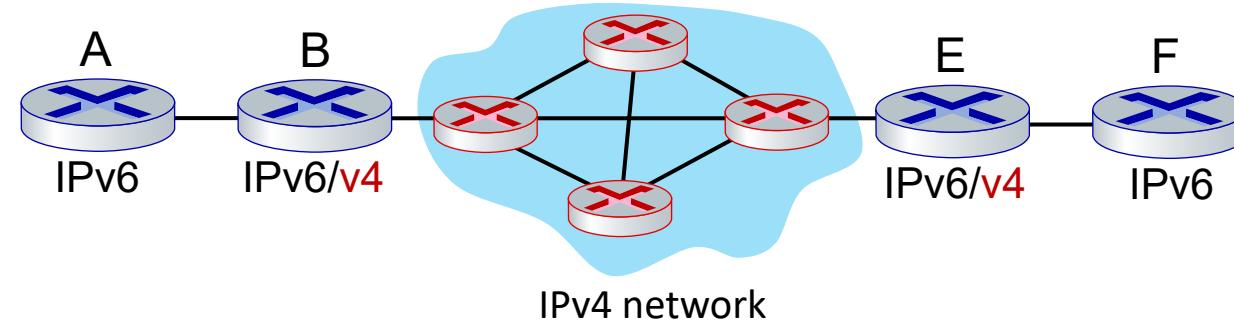


# Tunneling and encapsulation

Ethernet connecting  
two IPv6 routers:

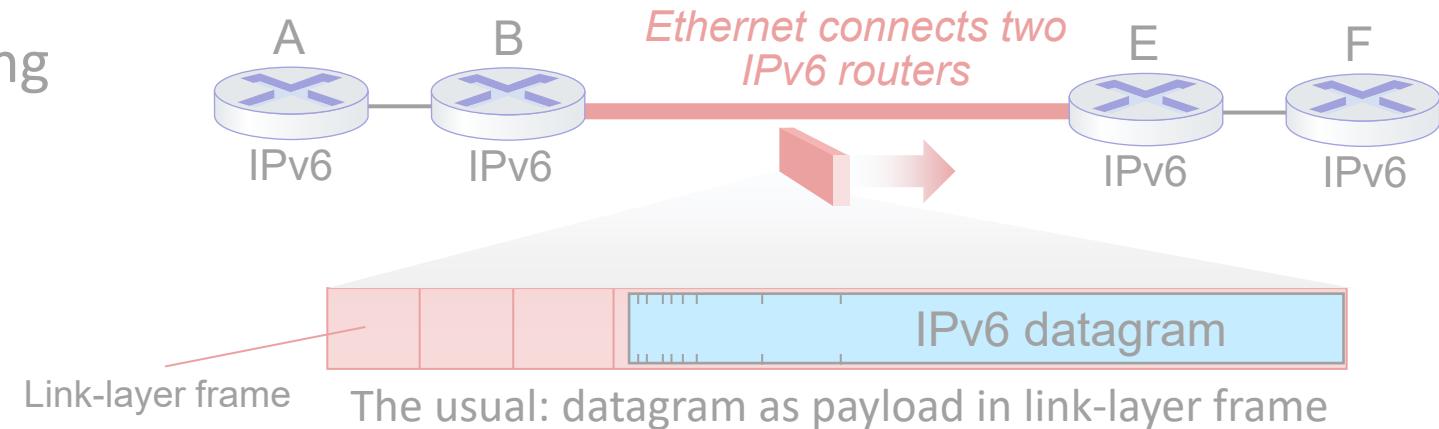


IPv4 network  
connecting two  
IPv6 routers

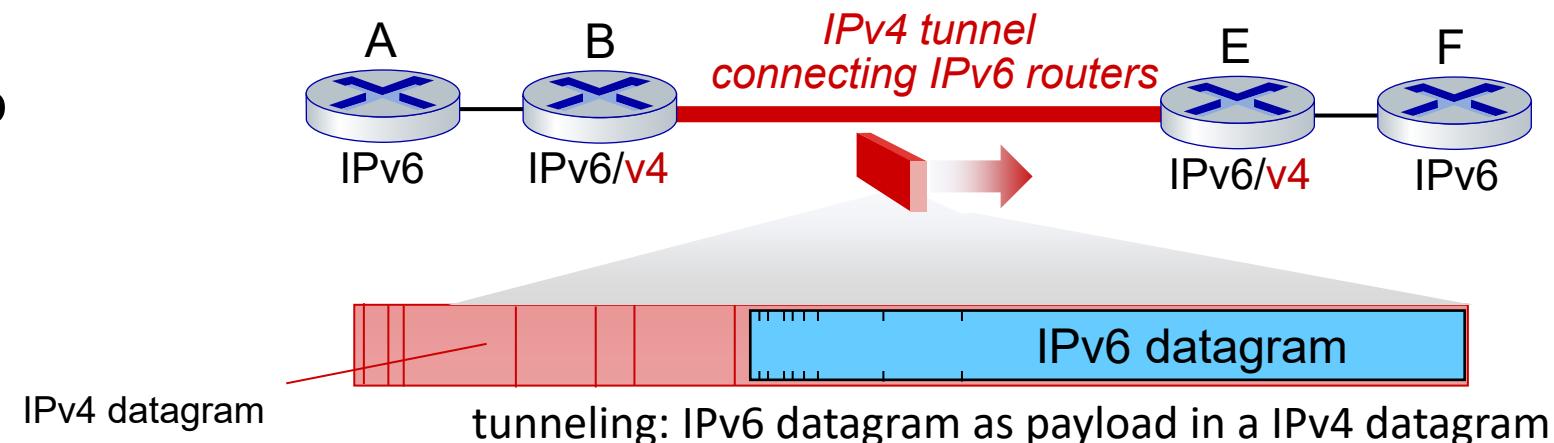


# Tunneling and encapsulation

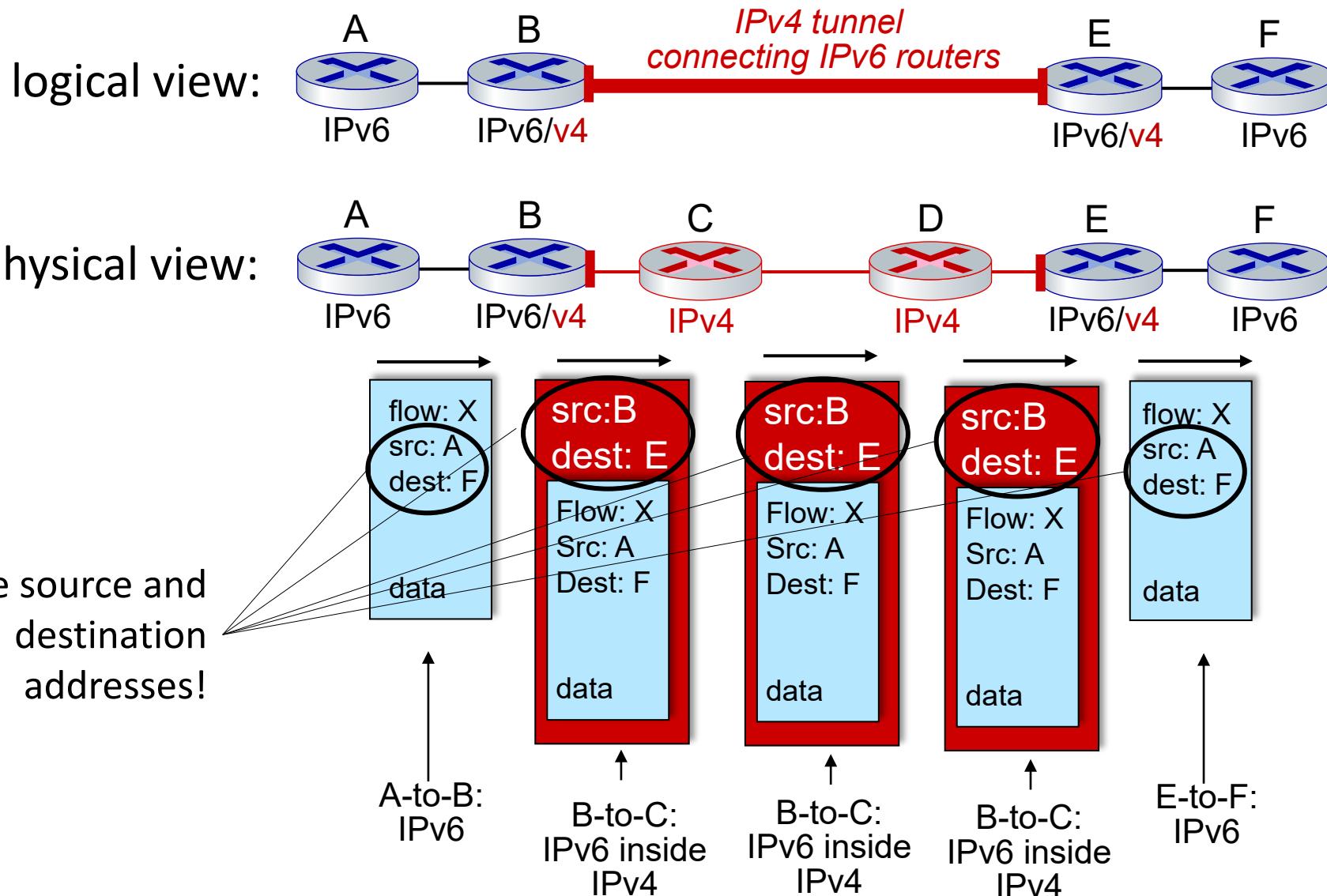
Ethernet connecting  
two IPv6 routers:



IPv4 tunnel  
connecting two  
IPv6 routers



# Tunneling

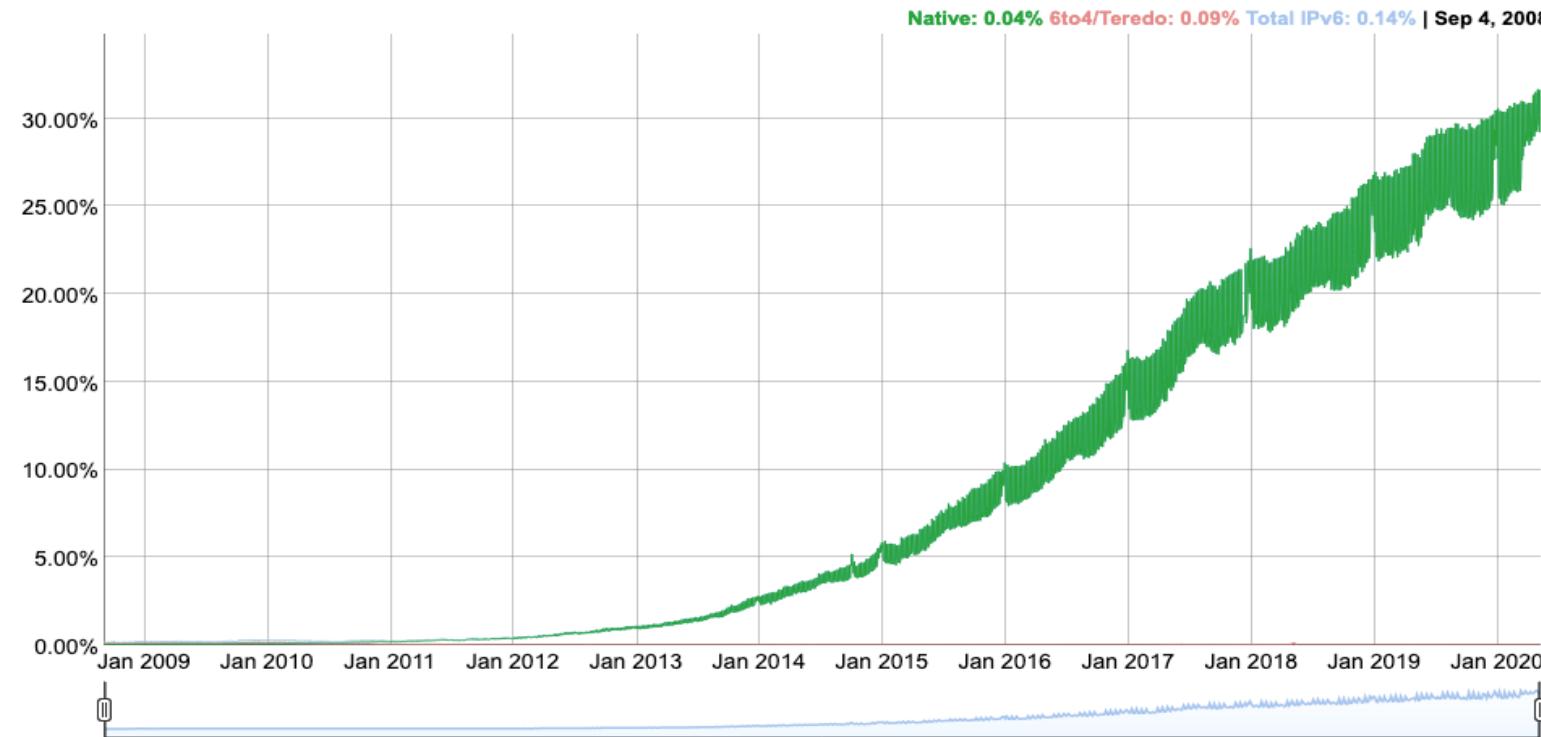


# IPv6: adoption

- Google<sup>1</sup>: ~ 30% of clients access services via IPv6
- NIST: 1/3 of all US government domains are IPv6 capable

## IPv6 Adoption

We are continuously measuring the availability of IPv6 connectivity among Google users. The graph shows the percentage of users that access Google over IPv6.



1

<https://www.google.com/intl/en/ipv6/statistics.html>

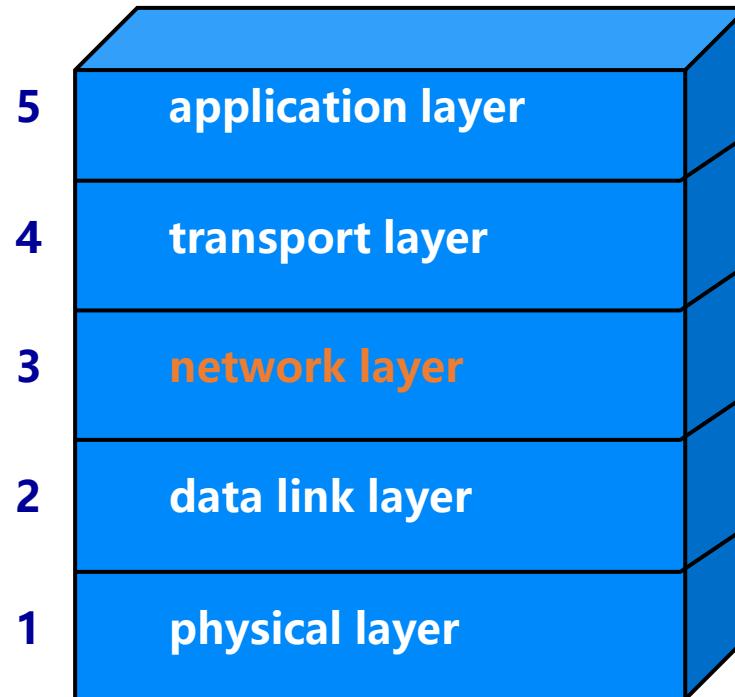


# Computer Networks

Lecturer: ZHANG Ying  
Fall semester 2022

# Chapter 5

# Network Layer – Control Plane



# Network-layer functions

- **forwarding:** move packets from router's input to appropriate router output
- **routing:** determine route taken by packets from source to destination

*data plane*

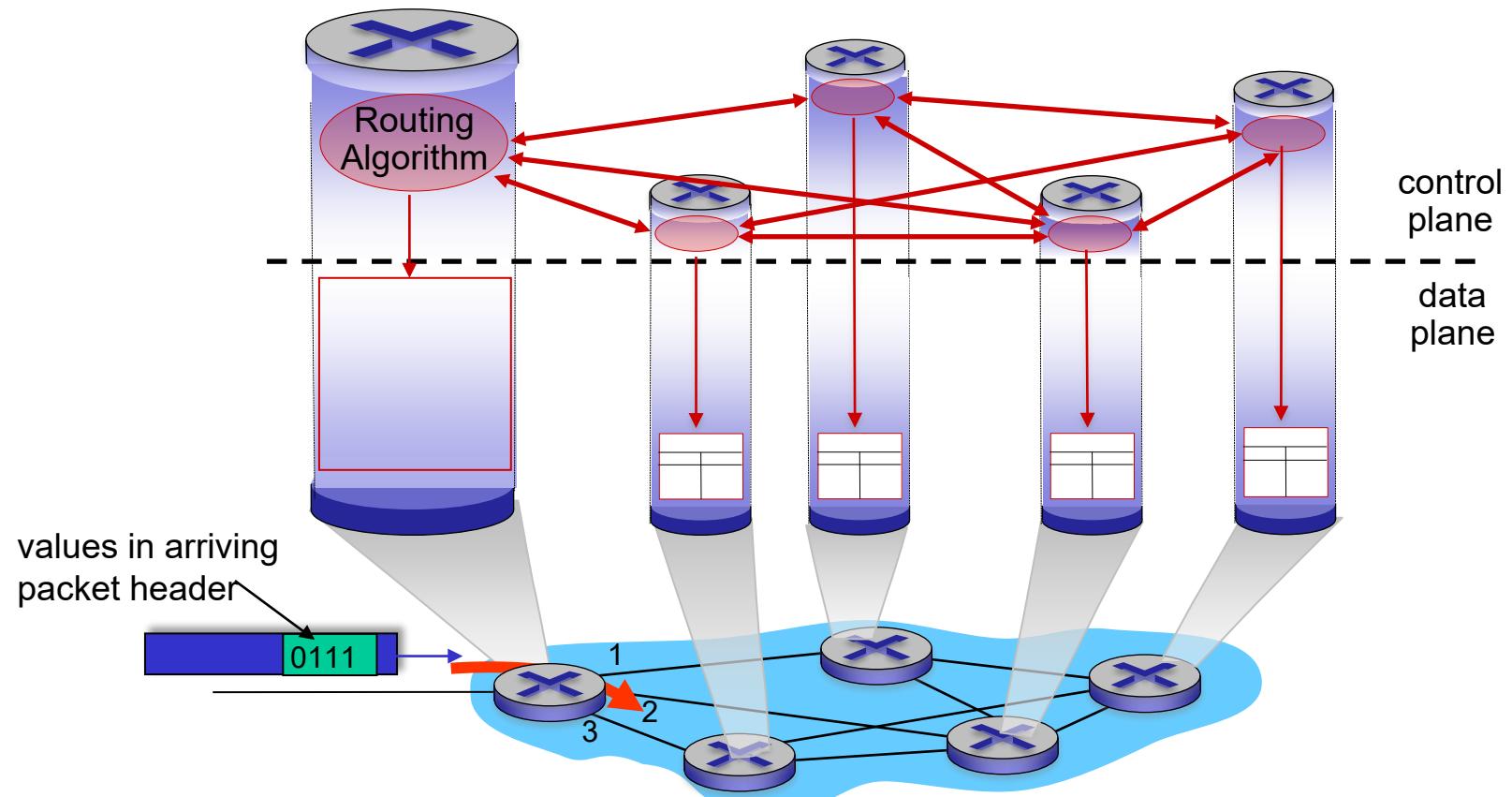
*control plane*

Two approaches to structuring network control plane:

- per-router control (traditional)
- logically centralized control (software defined networking)

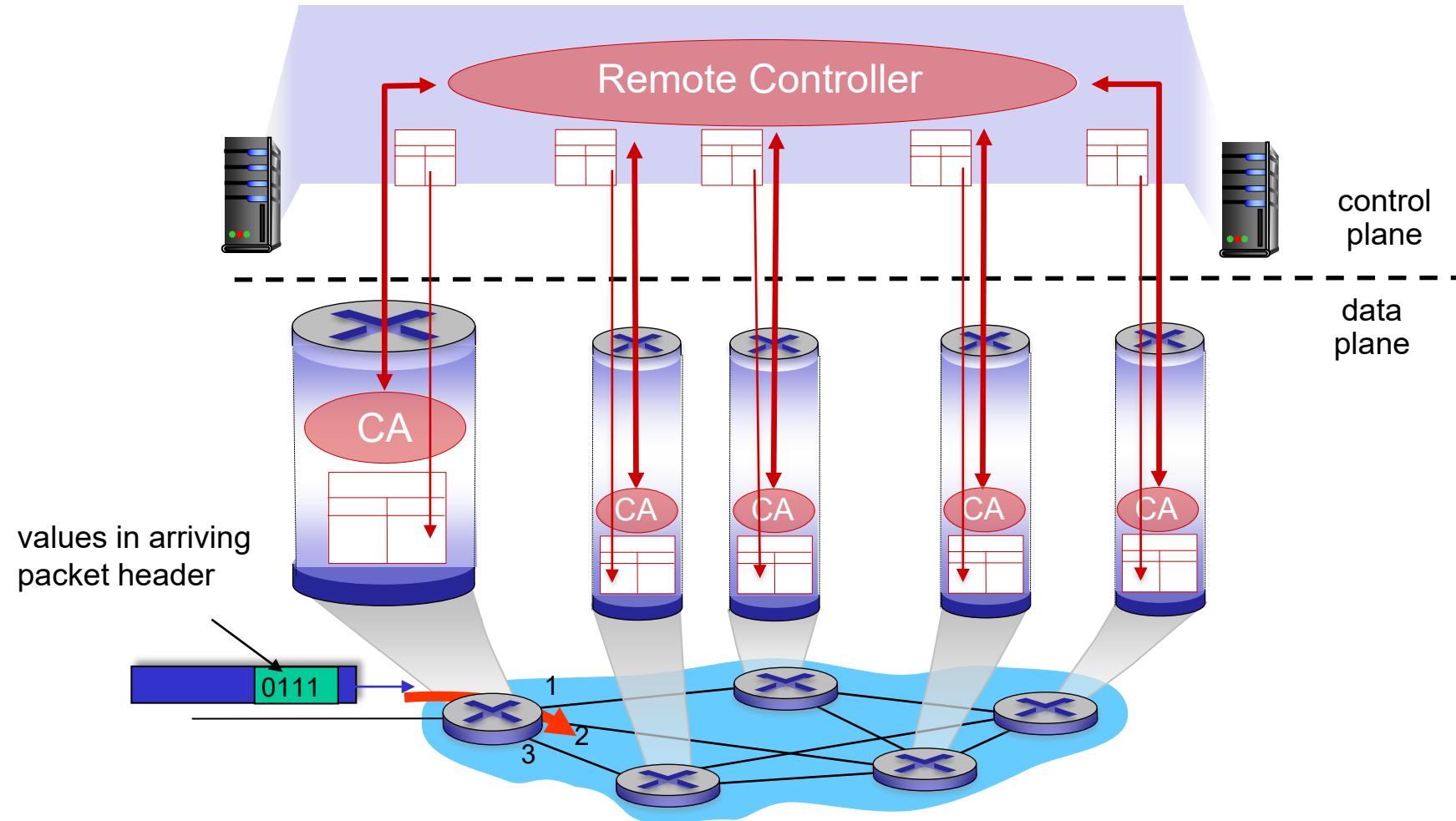
# Per-router control plane

Individual routing algorithm components *in each and every router* interact in the control plane



# Software-Defined Networking (SDN) control plane

Remote controller computes, installs forwarding tables in routers



# Network layer: “control plane” roadmap

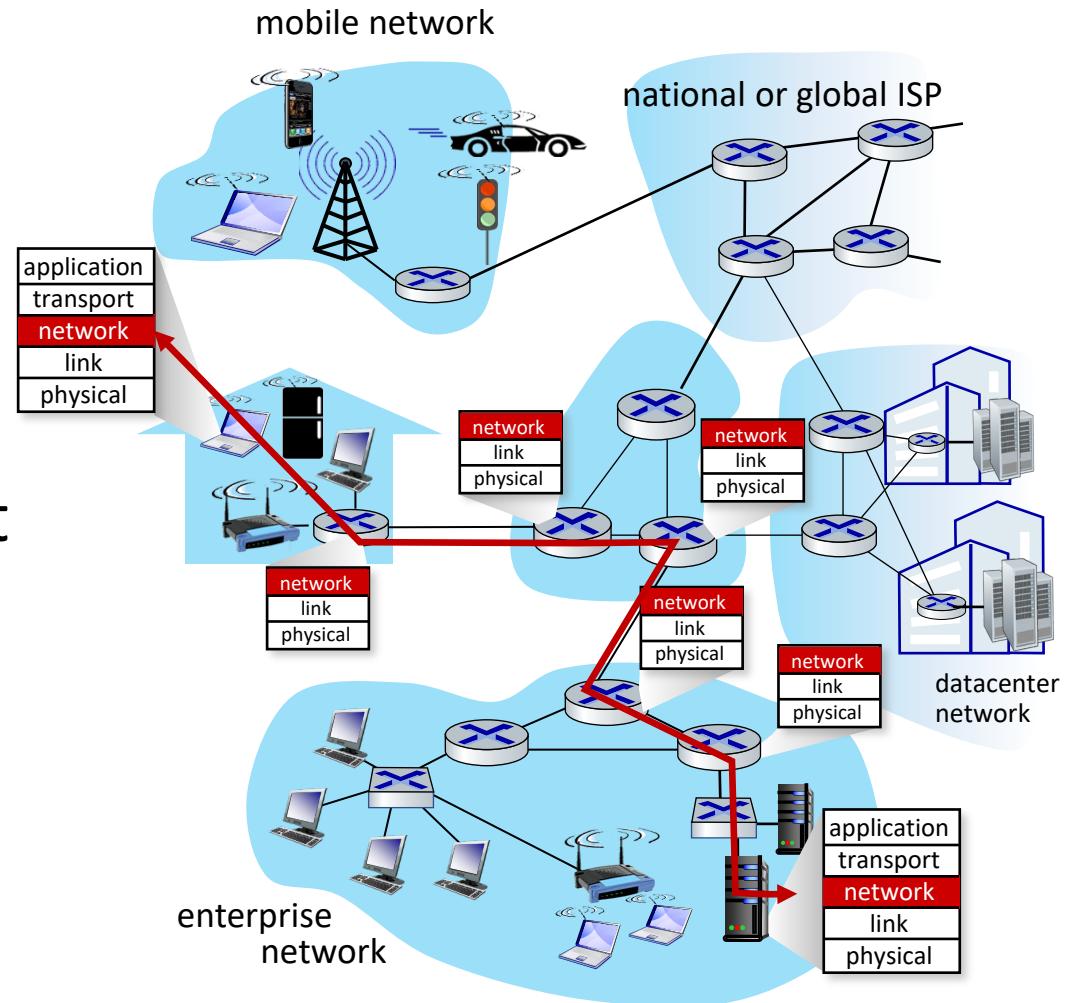
- introduction
- routing protocols
  - link state
  - distance vector



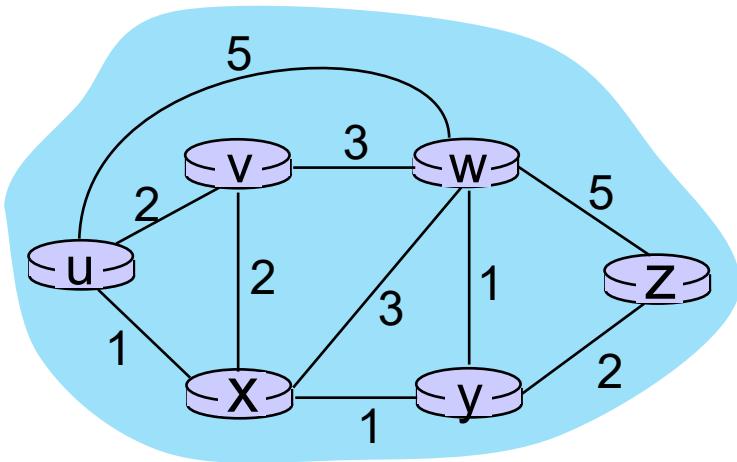
# Routing protocols

**Routing protocol goal:** determine “good” paths (equivalently, routes), from sending hosts to receiving host, through network of routers

- **path:** sequence of routers packets traverse from given initial source host to final destination host
- **“good”:** least “cost”, “fastest”, “least congested”
- **routing:** a “top-10” networking challenge!



# Graph abstraction: link costs



graph:  $G = (N, E)$

$N$ : set of routers = {  $u, v, w, x, y, z$  }

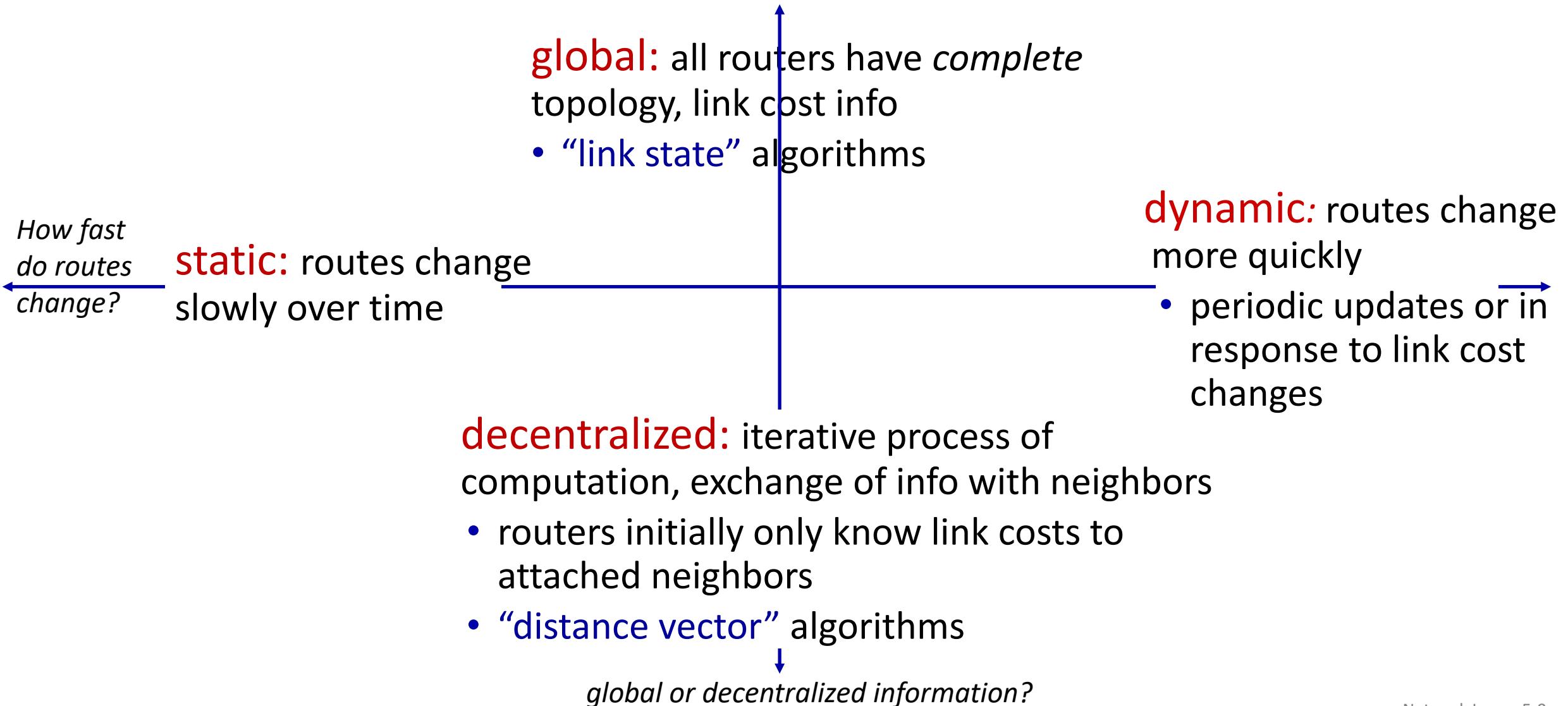
$E$ : set of links = {  $(u,v), (u,x), (v,x), (v,w), (x,w), (x,y), (w,y), (w,z), (y,z)$  }

$c_{a,b}$ : cost of *direct* link connecting  $a$  and  $b$

e.g.,  $c_{w,z} = 5, c_{u,z} = \infty$

cost defined by network operator:  
could always be 1, or inversely related  
to bandwidth, or inversely related to  
congestion

# Routing algorithm classification



# Network layer: “control plane” roadmap

- introduction
- routing protocols
  - link state
  - distance vector



# Dijkstra's link-state routing algorithm

- **centralized:** network topology, link costs known to *all* nodes
  - accomplished via “link state broadcast”
  - all nodes have same info
- computes least cost paths from one node (“source”) to all other nodes
  - gives *forwarding table* for that node
- **iterative:** after  $k$  iterations, know least cost path to  $k$  destinations

## notation

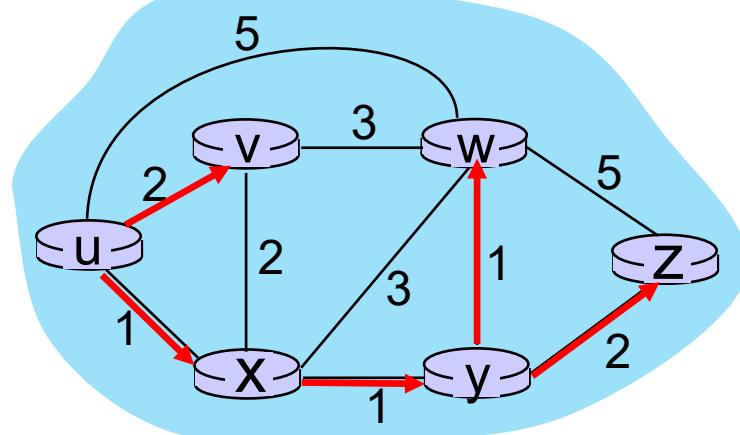
- $c_{x,y}$ : direct link cost from node  $x$  to  $y$ ;  $= \infty$  if not direct neighbors
- $D(v)$ : *current* estimate of cost of least-cost-path from source to destination  $v$
- $p(v)$ : predecessor node along path from source to  $v$
- $N'$ : set of nodes whose least-cost-path *definitively* known

# Dijkstra's link-state routing algorithm

```
1 Initialization:
2    $N' = \{u\}$                                 /* compute least cost path from u to all other nodes */
3   for all nodes  $v$ 
4     if  $v$  adjacent to  $u$                       /*  $u$  initially knows direct-path-cost only to direct neighbors */
5       then  $D(v) = c_{u,v}$                       /* but may not be minimum cost!
6     else  $D(v) = \infty$ 
7
8 Loop
9   find  $w$  not in  $N'$  such that  $D(w)$  is a minimum
10  add  $w$  to  $N'$ 
11  update  $D(v)$  for all  $v$  adjacent to  $w$  and not in  $N'$  :
12     $D(v) = \min(D(v), D(w) + c_{w,v})$ 
13  /* new least-path-cost to  $v$  is either old least-cost-path to  $v$  or known
14  least-cost-path to  $w$  plus direct-cost from  $w$  to  $v$  */
15 until all nodes in  $N'$ 
```

# Dijkstra's algorithm: an example

| Step | $N'$   | $D(v), p(v)$ | $D(w), p(w)$ | $D(x), p(x)$ | $D(y), p(y)$ | $D(z), p(z)$ |
|------|--------|--------------|--------------|--------------|--------------|--------------|
| 0    | u      | 2,u          | 5,u          | 1,u          | $\infty$     | $\infty$     |
| 1    | ux     | 2,u          | 4,x          |              | 2,x          | $\infty$     |
| 2    | uxy    | 2,u          | 3,y          |              |              | 4,y          |
| 3    | uxyyv  |              | 3,y          |              |              | 4,y          |
| 4    | uxyvw  |              |              |              |              | 4,y          |
| 5    | uxyvwz |              |              |              |              |              |

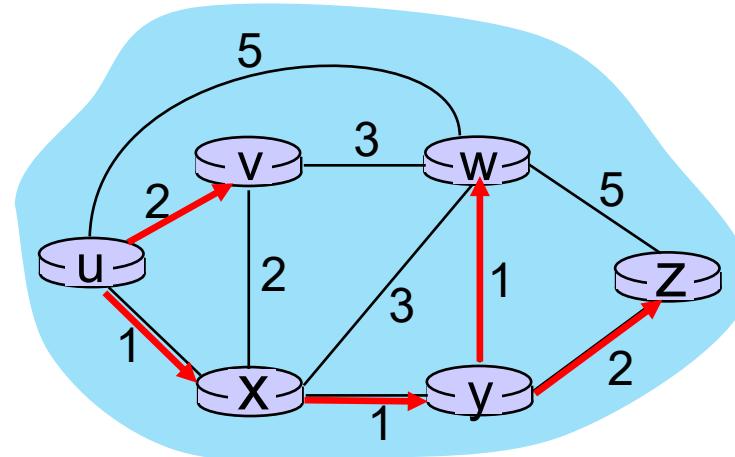


Initialization (step 0): For all  $a$ : if  $a$  adjacent to then  $D(a) = c_{u,a}$

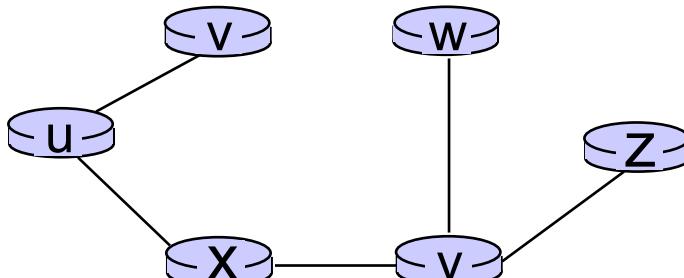
find  $a$  not in  $N'$  such that  $D(a)$  is a minimum  
 add  $a$  to  $N'$   
 update  $D(b)$  for all  $b$  adjacent to  $a$  and not in  $N'$ :  

$$D(b) = \min ( D(b), D(a) + c_{a,b} )$$

# Dijkstra's algorithm: an example



resulting least-cost-path tree from u:



resulting forwarding table in u:

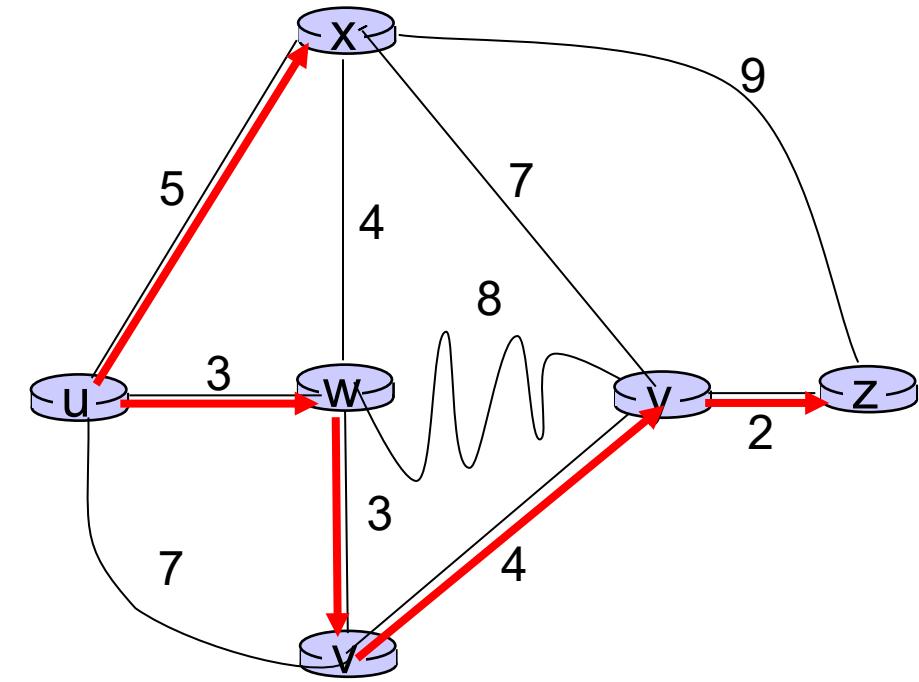
| destination | outgoing link |
|-------------|---------------|
| v           | (u,v)         |
| x           | (u,x)         |
| y           | (u,x)         |
| w           | (u,x)         |
| z           | (u,x)         |

route from  $u$  to  $v$  directly

route from  $u$  to all other destinations via  $x$

# Dijkstra's algorithm: another example

| Step | $N'$     | $v$          | $w$          | $x$          | $y$          | $z$          |
|------|----------|--------------|--------------|--------------|--------------|--------------|
| 0    | $u$      | $D(v), p(v)$ | $D(w), p(w)$ | $D(x), p(x)$ | $D(y), p(y)$ | $D(z), p(z)$ |
| 1    | $uw$     | $7, u$       | $3, u$       | $5, u$       | $\infty$     | $\infty$     |
| 2    | $uwx$    | $6, w$       | $5, u$       | $11, w$      | $\infty$     |              |
| 3    | $uwxv$   |              |              | $11, w$      | $14, x$      |              |
| 4    | $uwxvy$  |              |              | $10, v$      | $14, x$      |              |
| 5    | $uwxvzy$ |              |              |              | $12, y$      |              |



notes:

- construct least-cost-path tree by tracing predecessor nodes
- ties can exist (can be broken arbitrarily)

# Dijkstra's algorithm: discussion

## algorithm complexity: $n$ nodes

- each of  $n$  iteration: need to check all nodes,  $w$ , not in  $N$
- $n(n+1)/2$  comparisons:  $O(n^2)$  complexity
- more efficient implementations possible:  $O(n \log n)$

## message complexity:

- each router must *broadcast* its link state information to other  $n$  routers
- efficient (and interesting!) broadcast algorithms:  $O(n)$  link crossings to disseminate a broadcast message from one source
- each router's message crosses  $O(n)$  links: overall message complexity:  $O(n^2)$

# Network layer: “control plane” roadmap

- introduction
- routing protocols
  - link state
  - **distance vector**



# Distance vector algorithm

Based on *Bellman-Ford* (BF) equation (dynamic programming):

Bellman-Ford equation

Let  $D_x(y)$ : cost of least-cost path from  $x$  to  $y$ .

Then:

$$D_x(y) = \min_v \{ c_{x,v} + D_v(y) \}$$

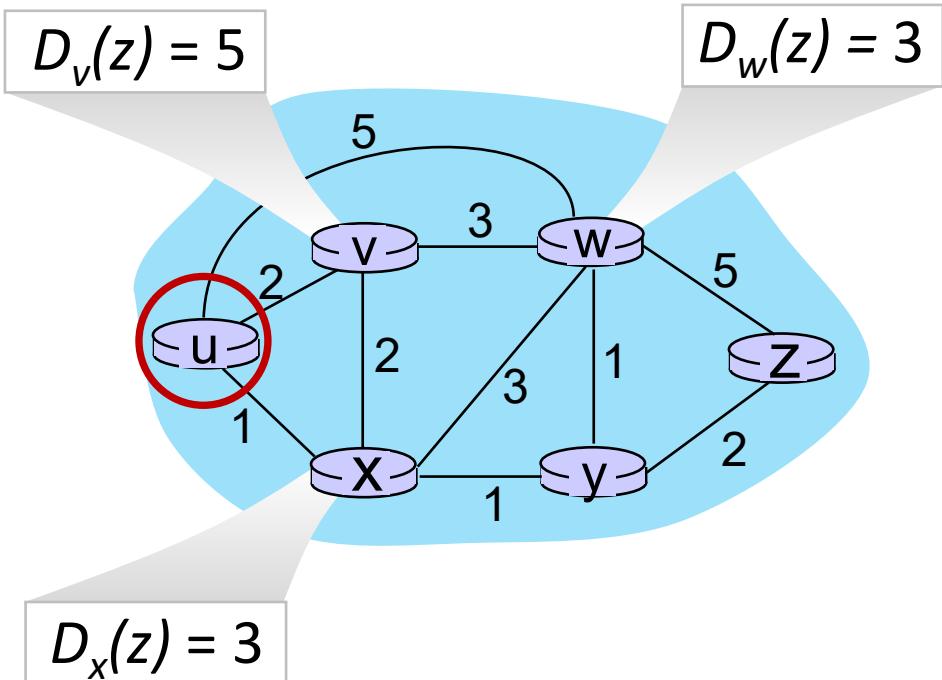
$\min$  taken over all neighbors  $v$  of  $x$

$v$ 's estimated least-cost-path cost to  $y$

direct cost of link from  $x$  to  $v$

# Bellman-Ford Example

Suppose that  $u$ 's neighboring nodes,  $x, v, w$ , know that for destination  $z$ :



Bellman-Ford equation says:

$$\begin{aligned} D_u(z) &= \min \{ c_{u,v} + D_v(z), \\ &\quad c_{u,x} + D_x(z), \\ &\quad c_{u,w} + D_w(z) \} \\ &= \min \{ 2 + 5, \\ &\quad 1 + 3, \\ &\quad 5 + 3 \} = 4 \end{aligned}$$

*node achieving minimum (x) is next hop on estimated least-cost path to destination (z)*

# Distance vector algorithm

key idea:

- from time-to-time, each node sends its own distance vector estimate to neighbors
- when  $x$  receives new DV estimate from any neighbor, it updates its own DV using B-F equation:

$$D_x(y) \leftarrow \min_v \{c_{x,v} + D_v(y)\} \text{ for each node } y \in N$$

- under minor, natural conditions, the estimate  $D_x(y)$  converge to the actual least cost  $d_x(y)$

# Distance vector algorithm:

each node:

- 
- ```
graph TD; A[each node:] --> B["wait for (change in local link cost or msg from neighbor)"]; B --> C["recompute DV estimates using DV received from neighbor"]; C --> D["if DV to any destination has changed, notify neighbors"];
```
- wait* for (change in local link cost or msg from neighbor)
  - recompute* DV estimates using DV received from neighbor
  - if DV to any destination has changed, *notify* neighbors

**iterative, asynchronous:** each local iteration caused by:

- local link cost change
- DV update message from neighbor

**distributed, self-stopping:** each node notifies neighbors *only* when its DV changes

- neighbors then notify their neighbors – *only if necessary*
- no notification received, no actions taken!

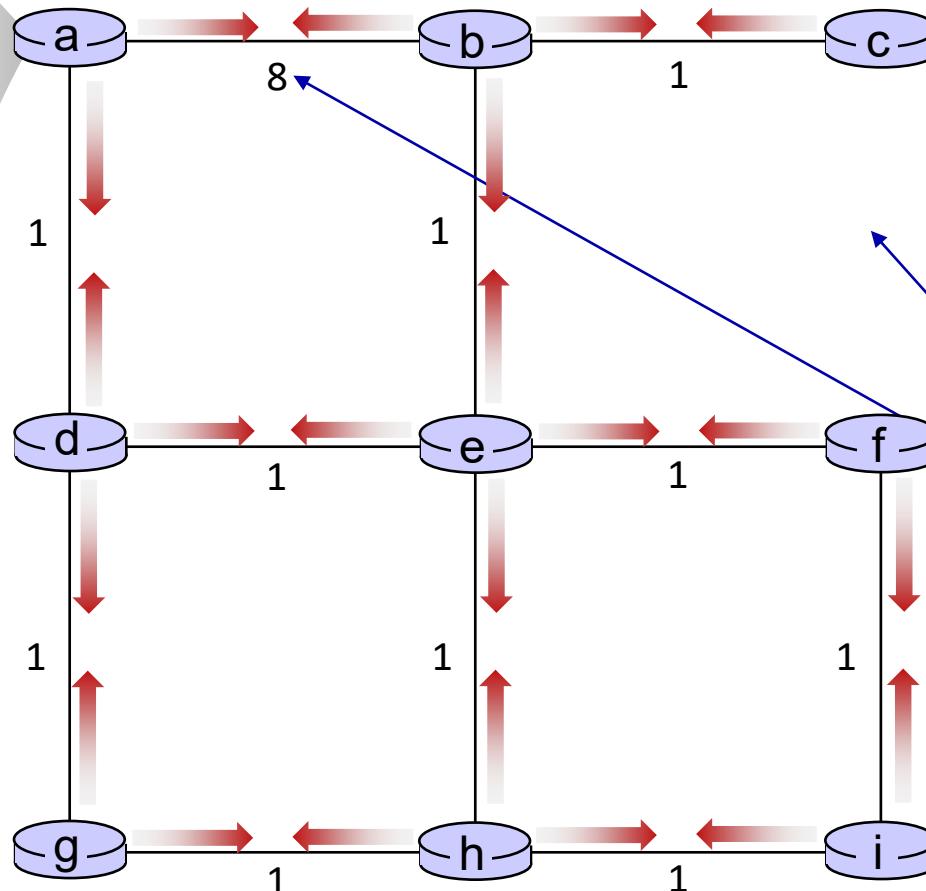
# Distance vector: example



$t=0$

- All nodes have distance estimates to nearest neighbors (only)
- All nodes send their local distance vector to their neighbors

| DV in a:          |
|-------------------|
| $D_a(a)=0$        |
| $D_a(b) = 8$      |
| $D_a(c) = \infty$ |
| $D_a(d) = 1$      |
| $D_a(e) = \infty$ |
| $D_a(f) = \infty$ |
| $D_a(g) = \infty$ |
| $D_a(h) = \infty$ |
| $D_a(i) = \infty$ |



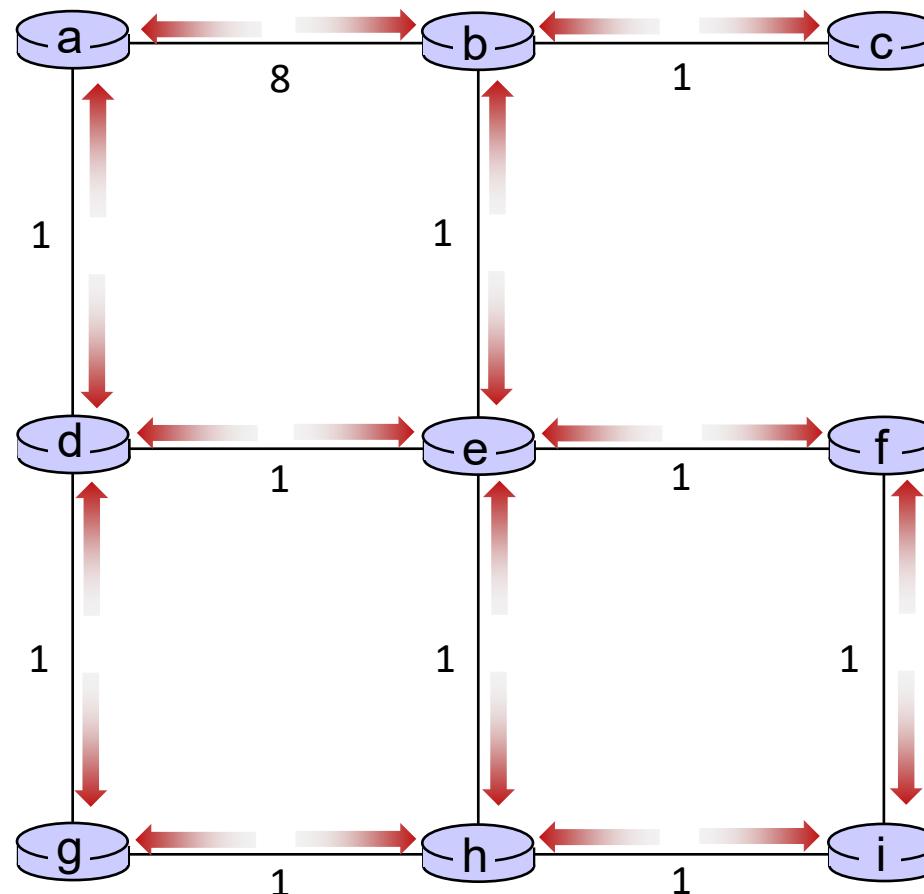
# Distance vector example: iteration



$t=1$

All nodes:

- receive distance vectors from neighbors
- compute their new local distance vector
- send their new local distance vector to neighbors



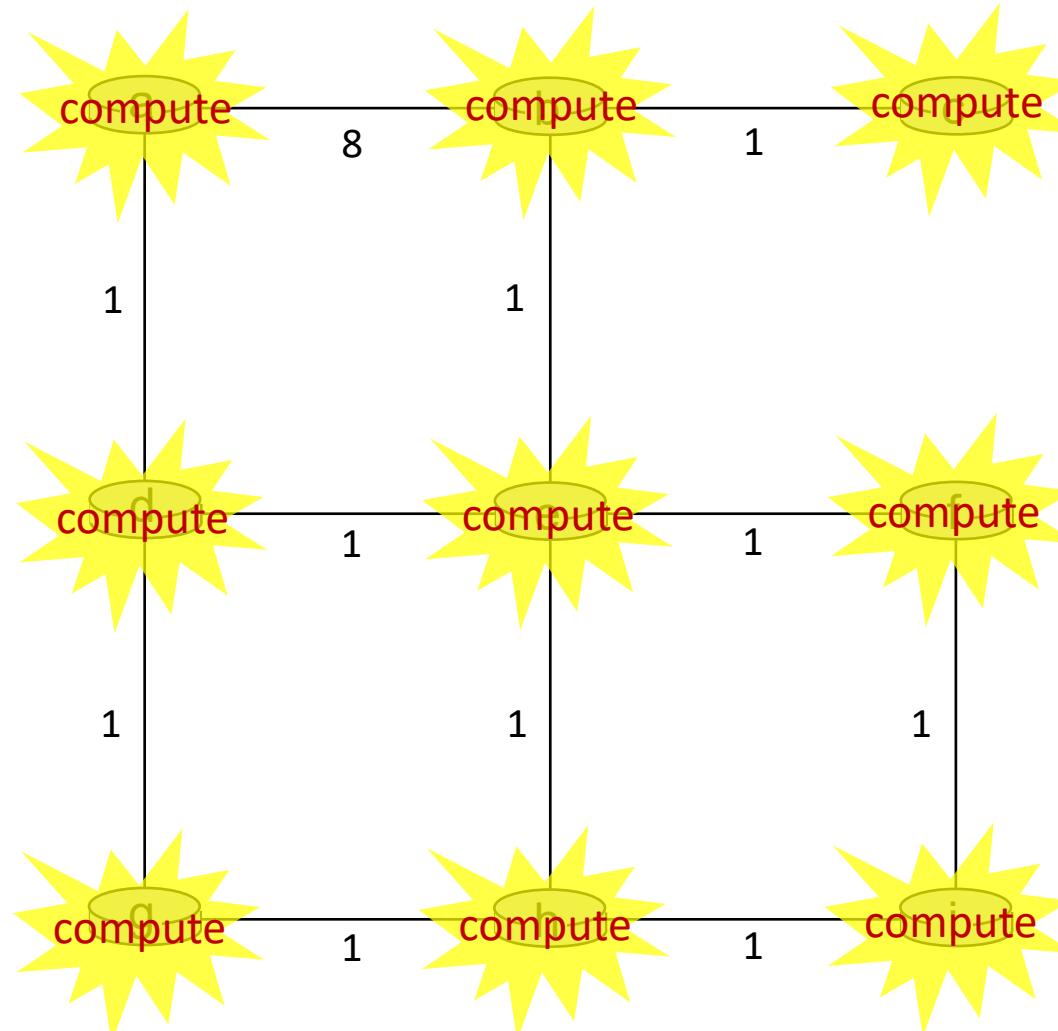
# Distance vector example: iteration



$t=1$

All nodes:

- receive distance vectors from neighbors
- compute their new local distance vector
- send their new local distance vector to neighbors



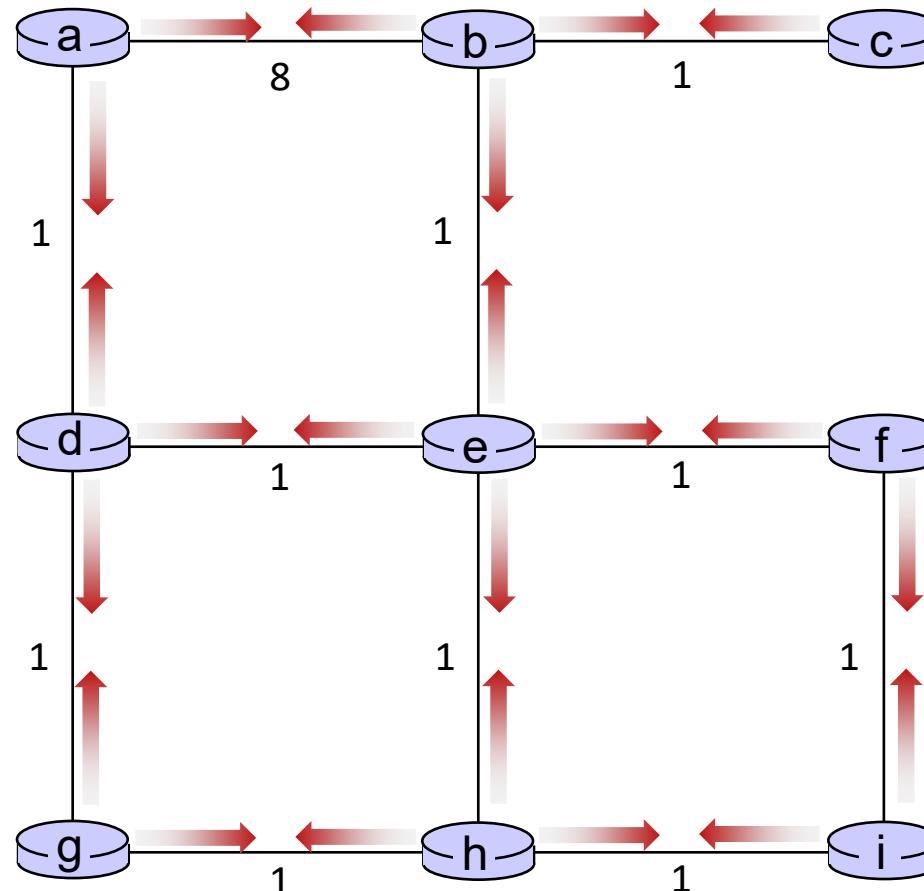
# Distance vector example: iteration



$t=1$

All nodes:

- receive distance vectors from neighbors
- compute their new local distance vector
- send their new local distance vector to neighbors



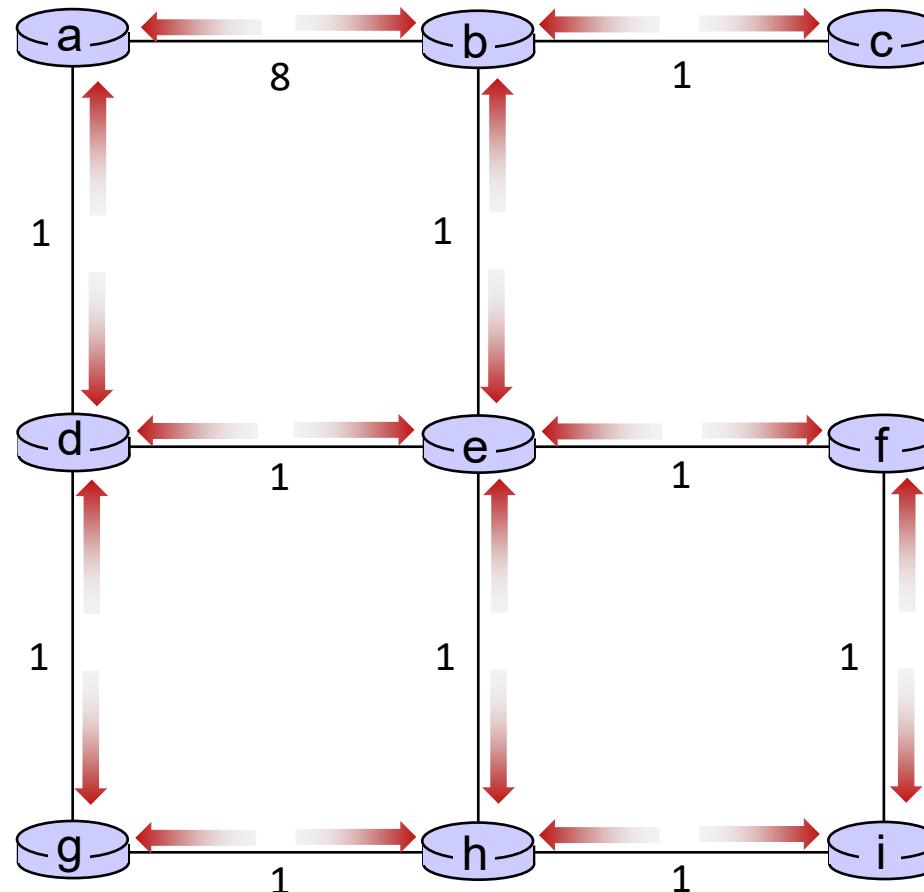
# Distance vector example: iteration



$t=2$

All nodes:

- receive distance vectors from neighbors
- compute their new local distance vector
- send their new local distance vector to neighbors



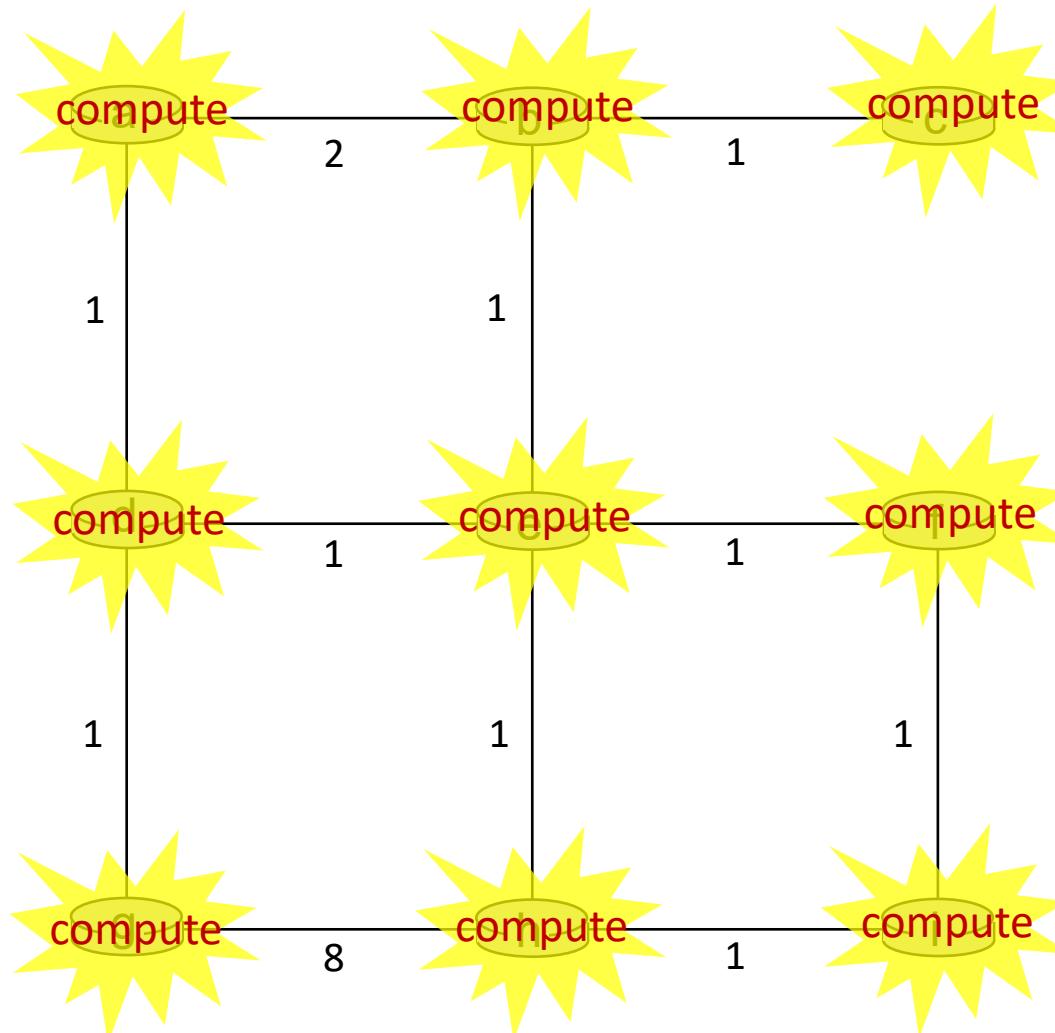
# Distance vector example: iteration



$t=2$

All nodes:

- receive distance vectors from neighbors
- compute their new local distance vector
- send their new local distance vector to neighbors



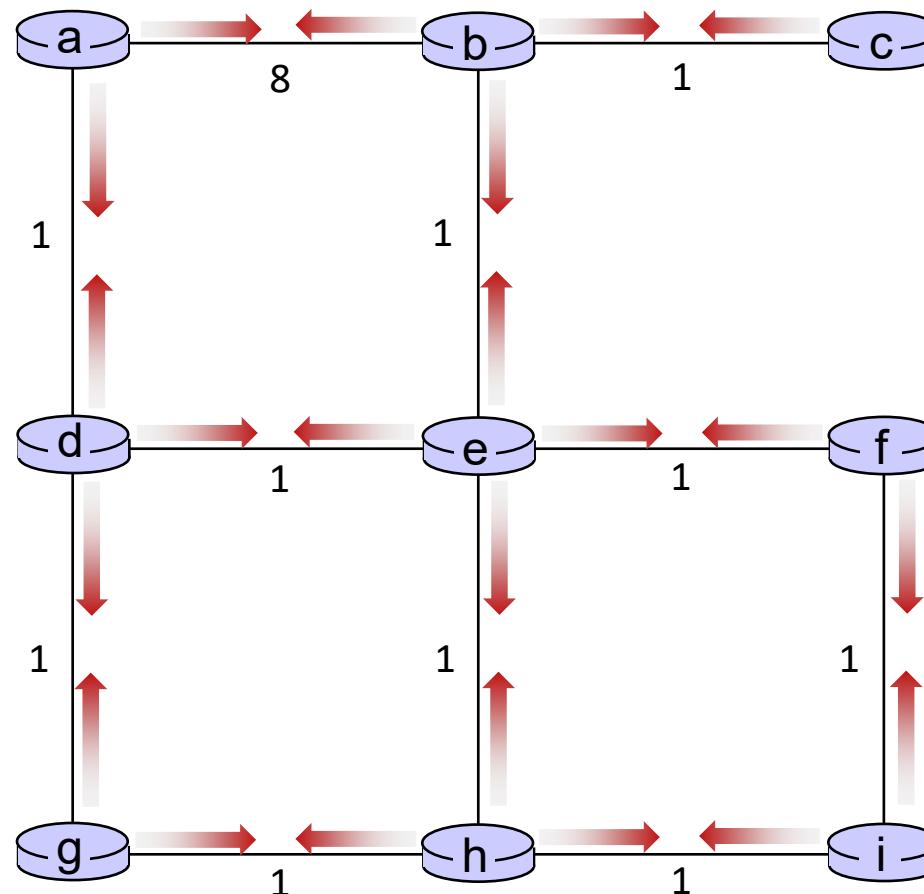
# Distance vector example: iteration



$t=2$

All nodes:

- receive distance vectors from neighbors
- compute their new local distance vector
- send their new local distance vector to neighbors



# Distance vector example: iteration

.... and so on

Let's next take a look at the iterative *computations* at nodes

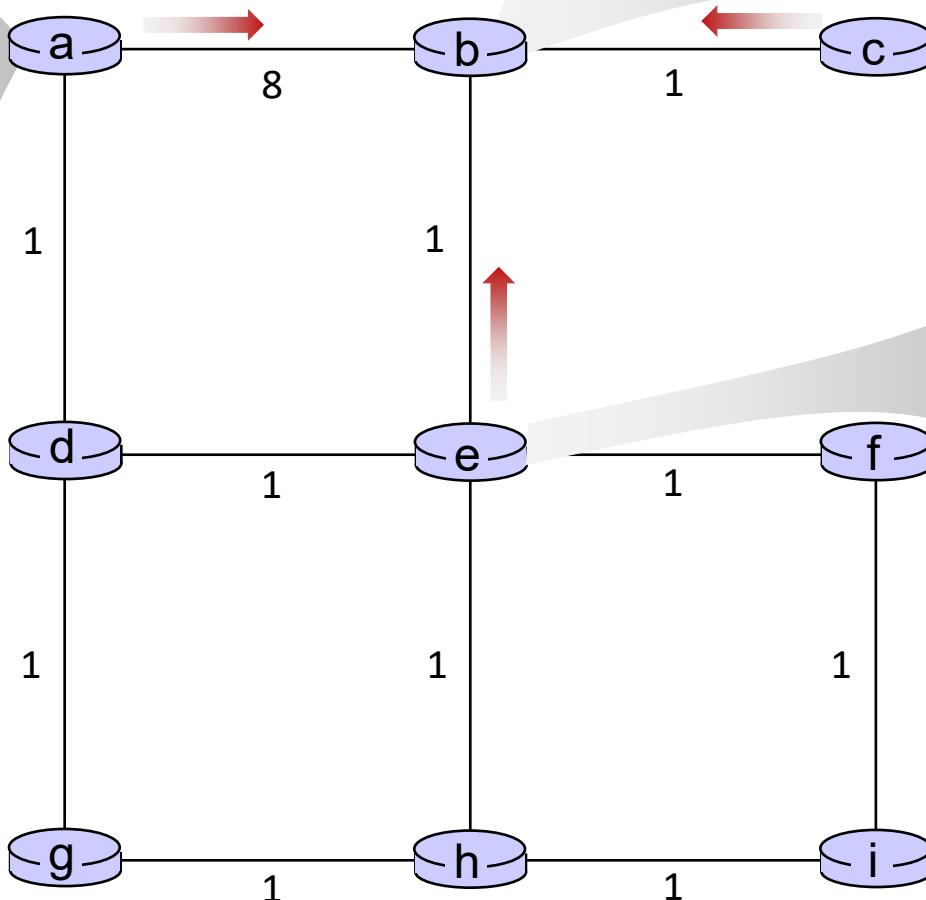
# Distance vector example: t=1



**t=1**

- b receives DVs from a, c, e

| DV in a:          |
|-------------------|
| $D_a(a) = 0$      |
| $D_a(b) = 8$      |
| $D_a(c) = \infty$ |
| $D_a(d) = 1$      |
| $D_a(e) = \infty$ |
| $D_a(f) = \infty$ |
| $D_a(g) = \infty$ |
| $D_a(h) = \infty$ |
| $D_a(i) = \infty$ |



| DV in b:          |
|-------------------|
| $D_b(a) = 8$      |
| $D_b(f) = \infty$ |
| $D_b(c) = 1$      |
| $D_b(g) = \infty$ |
| $D_b(d) = \infty$ |
| $D_b(h) = \infty$ |
| $D_b(e) = 1$      |
| $D_b(i) = \infty$ |

| DV in c:          |
|-------------------|
| $D_c(a) = \infty$ |
| $D_c(b) = 1$      |
| $D_c(c) = 0$      |
| $D_c(d) = \infty$ |
| $D_c(e) = \infty$ |
| $D_c(f) = \infty$ |
| $D_c(g) = \infty$ |
| $D_c(h) = \infty$ |
| $D_c(i) = \infty$ |

| DV in e:          |
|-------------------|
| $D_e(a) = \infty$ |
| $D_e(b) = 1$      |
| $D_e(c) = \infty$ |
| $D_e(d) = 1$      |
| $D_e(e) = 0$      |
| $D_e(f) = 1$      |
| $D_e(g) = \infty$ |
| $D_e(h) = 1$      |
| $D_e(i) = \infty$ |

# Distance vector example: t=1

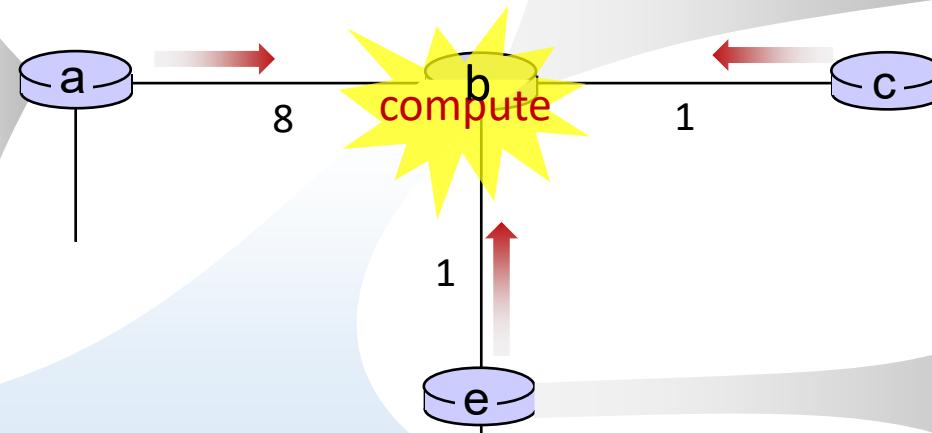


**t=1**

- b receives DVs from a, c, e, computes:

$$\begin{aligned}
 D_b(a) &= \min\{c_{b,a}+D_a(a), c_{b,c}+D_c(a), c_{b,e}+D_e(a)\} = \min\{8, \infty, \infty\} = 8 \\
 D_b(c) &= \min\{c_{b,a}+D_a(c), c_{b,c}+D_c(c), c_{b,e}+D_e(c)\} = \min\{\infty, 1, \infty\} = 1 \\
 D_b(d) &= \min\{c_{b,a}+D_a(d), c_{b,c}+D_c(d), c_{b,e}+D_e(d)\} = \min\{9, 2, \infty\} = 2 \\
 D_b(e) &= \min\{c_{b,a}+D_a(e), c_{b,c}+D_c(e), c_{b,e}+D_e(e)\} = \min\{\infty, \infty, 1\} = 1 \\
 D_b(f) &= \min\{c_{b,a}+D_a(f), c_{b,c}+D_c(f), c_{b,e}+D_e(f)\} = \min\{\infty, \infty, 2\} = 2 \\
 D_b(g) &= \min\{c_{b,a}+D_a(g), c_{b,c}+D_c(g), c_{b,e}+D_e(g)\} = \min\{\infty, \infty, \infty\} = \infty \\
 D_b(h) &= \min\{c_{b,a}+D_a(h), c_{b,c}+D_c(h), c_{b,e}+D_e(h)\} = \min\{\infty, \infty, 2\} = 2 \\
 D_b(i) &= \min\{c_{b,a}+D_a(i), c_{b,c}+D_c(i), c_{b,e}+D_e(i)\} = \min\{\infty, \infty, \infty\} = \infty
 \end{aligned}$$

| DV in a:          |
|-------------------|
| $D_a(a)=0$        |
| $D_a(b) = 8$      |
| $D_a(c) = \infty$ |
| $D_a(d) = 1$      |
| $D_a(e) = \infty$ |
| $D_a(f) = \infty$ |
| $D_a(g) = \infty$ |
| $D_a(h) = \infty$ |
| $D_a(i) = \infty$ |



| DV in b:          |                   |
|-------------------|-------------------|
| $D_b(a) = 8$      | $D_b(f) = \infty$ |
| $D_b(c) = 1$      | $D_b(g) = \infty$ |
| $D_b(d) = \infty$ | $D_b(h) = \infty$ |
| $D_b(e) = 1$      | $D_b(i) = \infty$ |

| DV in c:          |
|-------------------|
| $D_c(a) = \infty$ |
| $D_c(b) = 1$      |
| $D_c(c) = 0$      |
| $D_c(d) = \infty$ |
| $D_c(e) = \infty$ |
| $D_c(f) = \infty$ |
| $D_c(g) = \infty$ |
| $D_c(h) = \infty$ |
| $D_c(i) = \infty$ |

| DV in e:          |
|-------------------|
| $D_e(a) = \infty$ |
| $D_e(b) = 1$      |
| $D_e(c) = \infty$ |
| $D_e(d) = 1$      |
| $D_e(e) = 0$      |
| $D_e(f) = 1$      |
| $D_e(g) = \infty$ |
| $D_e(h) = 1$      |
| $D_e(i) = \infty$ |

| DV in b:     |                   |
|--------------|-------------------|
| $D_b(a) = 8$ | $D_b(f) = 2$      |
| $D_b(c) = 1$ | $D_b(g) = \infty$ |
| $D_b(d) = 2$ | $D_b(h) = 2$      |
| $D_b(e) = 1$ | $D_b(i) = \infty$ |

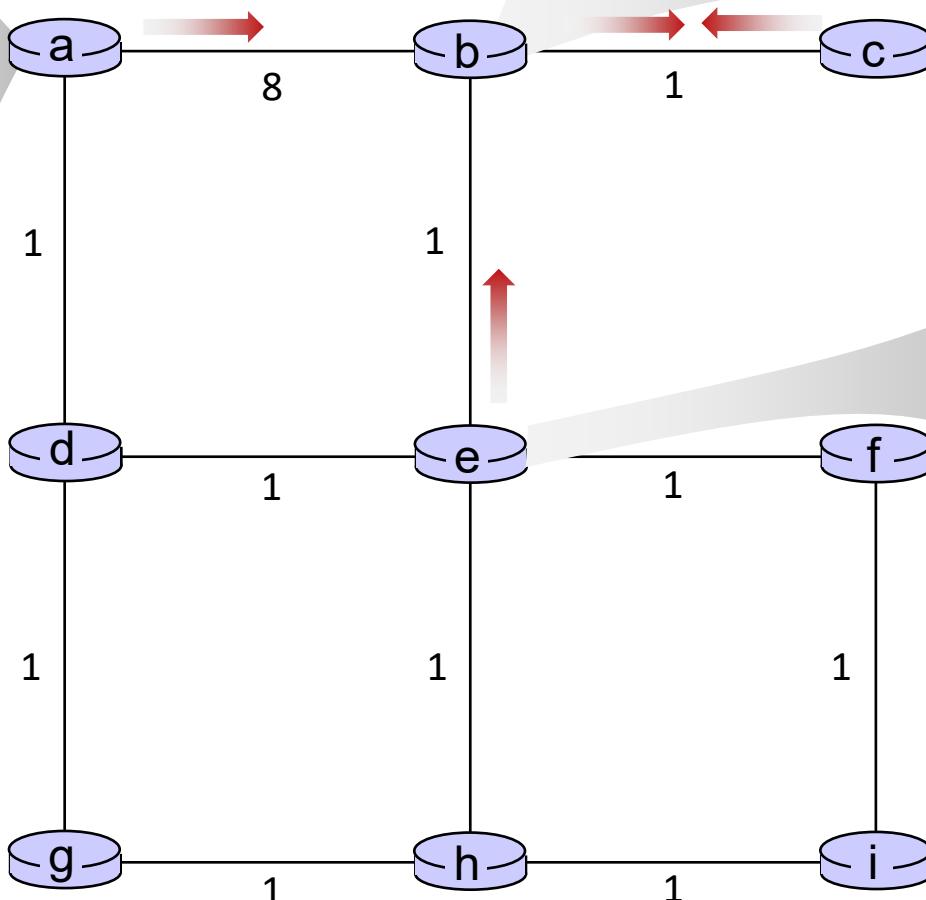
# Distance vector example: t=1



**t=1**

- c receives DVs from b

| DV in a:          |
|-------------------|
| $D_a(a) = 0$      |
| $D_a(b) = 8$      |
| $D_a(c) = \infty$ |
| $D_a(d) = 1$      |
| $D_a(e) = \infty$ |
| $D_a(f) = \infty$ |
| $D_a(g) = \infty$ |
| $D_a(h) = \infty$ |
| $D_a(i) = \infty$ |



| DV in b:          |
|-------------------|
| $D_b(a) = 8$      |
| $D_b(f) = \infty$ |
| $D_b(c) = 1$      |
| $D_b(g) = \infty$ |
| $D_b(d) = \infty$ |
| $D_b(h) = \infty$ |
| $D_b(e) = 1$      |
| $D_b(i) = \infty$ |

| DV in c:          |
|-------------------|
| $D_c(a) = \infty$ |
| $D_c(b) = 1$      |
| $D_c(c) = 0$      |
| $D_c(d) = \infty$ |
| $D_c(e) = \infty$ |
| $D_c(f) = \infty$ |
| $D_c(g) = \infty$ |
| $D_c(h) = \infty$ |
| $D_c(i) = \infty$ |

| DV in e:          |
|-------------------|
| $D_e(a) = \infty$ |
| $D_e(b) = 1$      |
| $D_e(c) = \infty$ |
| $D_e(d) = 1$      |
| $D_e(e) = 0$      |
| $D_e(f) = 1$      |
| $D_e(g) = \infty$ |
| $D_e(h) = 1$      |
| $D_e(i) = \infty$ |

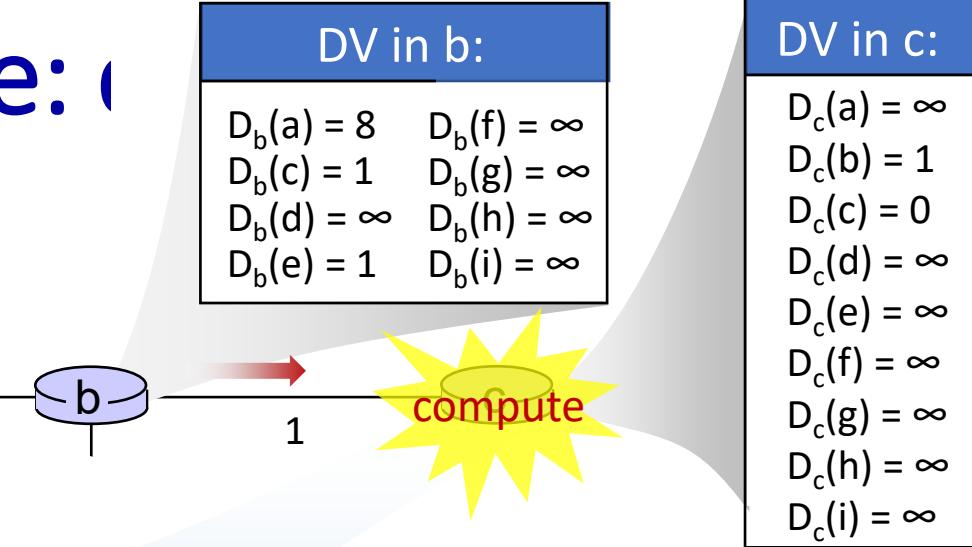
# Distance vector example: (t=1)



t=1

- c receives DVs from b computes:

$$\begin{aligned}D_c(a) &= \min\{c_{c,b} + D_b(a)\} = 1 + 8 = 9 \\D_c(b) &= \min\{c_{c,b} + D_b(b)\} = 1 + 0 = 1 \\D_c(d) &= \min\{c_{c,b} + D_b(d)\} = 1 + \infty = \infty \\D_c(e) &= \min\{c_{c,b} + D_b(e)\} = 1 + 1 = 2 \\D_c(f) &= \min\{c_{c,b} + D_b(f)\} = 1 + \infty = \infty \\D_c(g) &= \min\{c_{c,b} + D_b(g)\} = 1 + \infty = \infty \\D_c(h) &= \min\{c_{c,b} + D_b(h)\} = 1 + \infty = \infty \\D_c(i) &= \min\{c_{c,b} + D_b(i)\} = 1 + \infty = \infty\end{aligned}$$



DV in c:

|                   |
|-------------------|
| $D_c(a) = 9$      |
| $D_c(b) = 1$      |
| $D_c(c) = 0$      |
| $D_c(d) = 2$      |
| $D_c(e) = \infty$ |
| $D_c(f) = \infty$ |
| $D_c(g) = \infty$ |
| $D_c(h) = \infty$ |
| $D_c(i) = \infty$ |

\* Check out the online interactive exercises for more examples:  
[http://gaia.cs.umass.edu/kurose\\_ross/interactive/](http://gaia.cs.umass.edu/kurose_ross/interactive/)

# Distance vector example: t=1

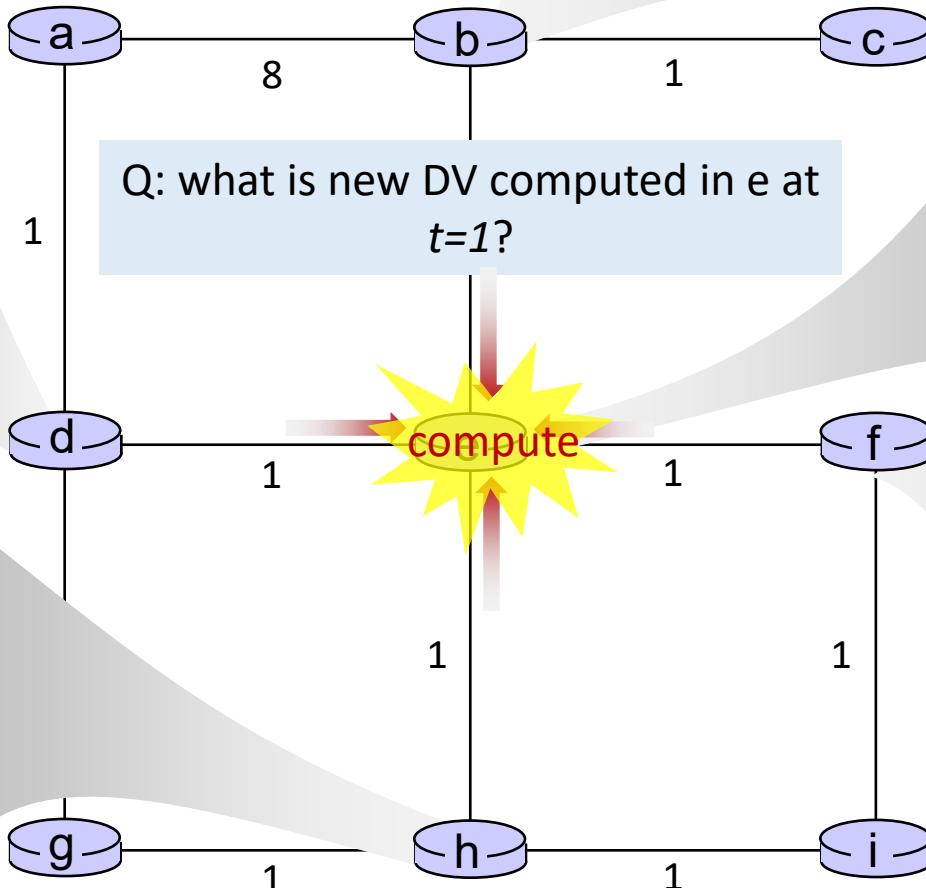


**t=1**

- e receives DVs from b, d, f, h

| DV in d:          |
|-------------------|
| $D_c(a) = 1$      |
| $D_c(b) = \infty$ |
| $D_c(c) = \infty$ |
| $D_c(d) = 0$      |
| $D_c(e) = 1$      |
| $D_c(f) = \infty$ |
| $D_c(g) = 1$      |
| $D_c(h) = \infty$ |
| $D_c(i) = \infty$ |

| DV in h:          |
|-------------------|
| $D_c(a) = \infty$ |
| $D_c(b) = \infty$ |
| $D_c(c) = \infty$ |
| $D_c(d) = \infty$ |
| $D_c(e) = 1$      |
| $D_c(f) = \infty$ |
| $D_c(g) = 1$      |
| $D_c(h) = 0$      |
| $D_c(i) = 1$      |



| DV in b:          |
|-------------------|
| $D_b(a) = 8$      |
| $D_b(f) = \infty$ |
| $D_b(c) = 1$      |
| $D_b(g) = \infty$ |
| $D_b(d) = \infty$ |
| $D_b(h) = \infty$ |
| $D_b(e) = 1$      |
| $D_b(i) = \infty$ |

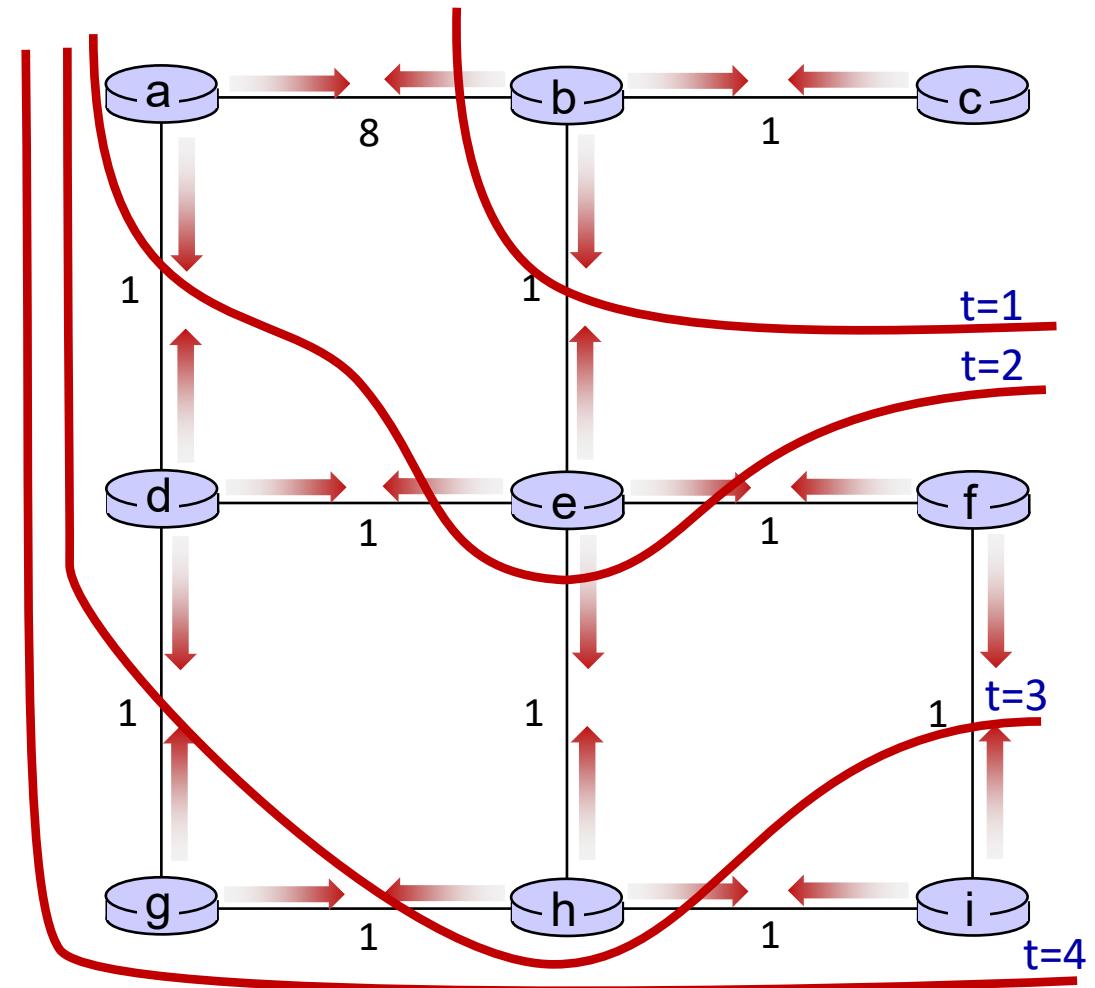
| DV in e:          |
|-------------------|
| $D_e(a) = \infty$ |
| $D_e(b) = 1$      |
| $D_e(c) = \infty$ |
| $D_e(d) = 1$      |
| $D_e(e) = 0$      |
| $D_e(f) = 1$      |
| $D_e(g) = \infty$ |
| $D_e(h) = 1$      |
| $D_e(i) = \infty$ |

| DV in f:          |
|-------------------|
| $D_c(a) = \infty$ |
| $D_c(b) = \infty$ |
| $D_c(c) = \infty$ |
| $D_c(d) = \infty$ |
| $D_c(e) = 1$      |
| $D_c(f) = 0$      |
| $D_c(g) = \infty$ |
| $D_c(h) = \infty$ |
| $D_c(i) = 1$      |

# Distance vector: state information diffusion

Iterative communication, computation steps diffuses information through network:

-  t=0 c's state at t=0 is at c only
-  t=1 c's state at t=0 has propagated to b, and may influence distance vector computations up to **1** hop away, i.e., at b
-  t=2 c's state at t=0 may now influence distance vector computations up to **2** hops away, i.e., at b and now at a, e as well
-  t=3 c's state at t=0 may influence distance vector computations up to **3** hops away, i.e., at b,a,e and now at c,f,h as well
-  t=4 c's state at t=0 may influence distance vector computations up to **4** hops away, i.e., at b,a,e, c, f, h and now at g,i as well



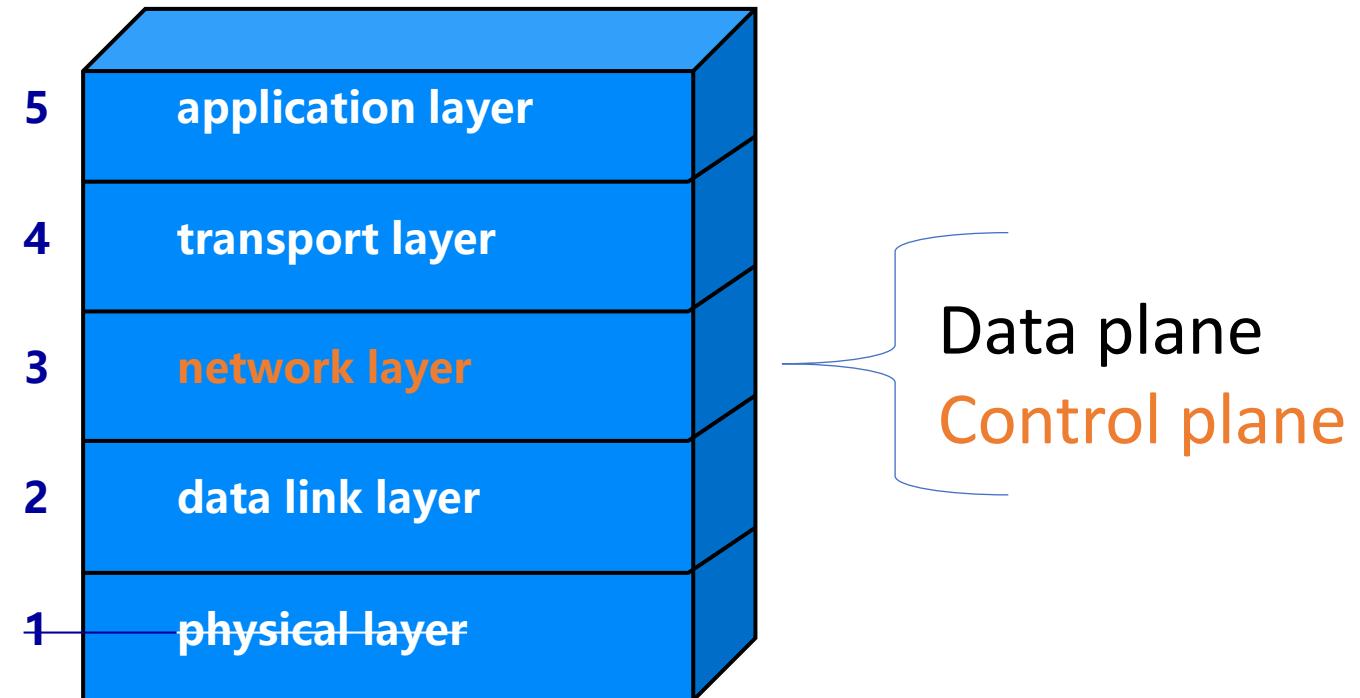


# Computer Networks

Lecturer: ZHANG Ying  
Fall semester 2022

# Chapter 5

## Network Layer – Control Plane



# Distance vector algorithm

Based on *Bellman-Ford* (BF) equation (dynamic programming):

Bellman-Ford equation

Let  $D_x(y)$ : cost of least-cost path from  $x$  to  $y$ .

Then:

$$D_x(y) = \min_v \{ c_{x,v} + D_v(y) \}$$

$\min$  taken over all neighbors  $v$  of  $x$

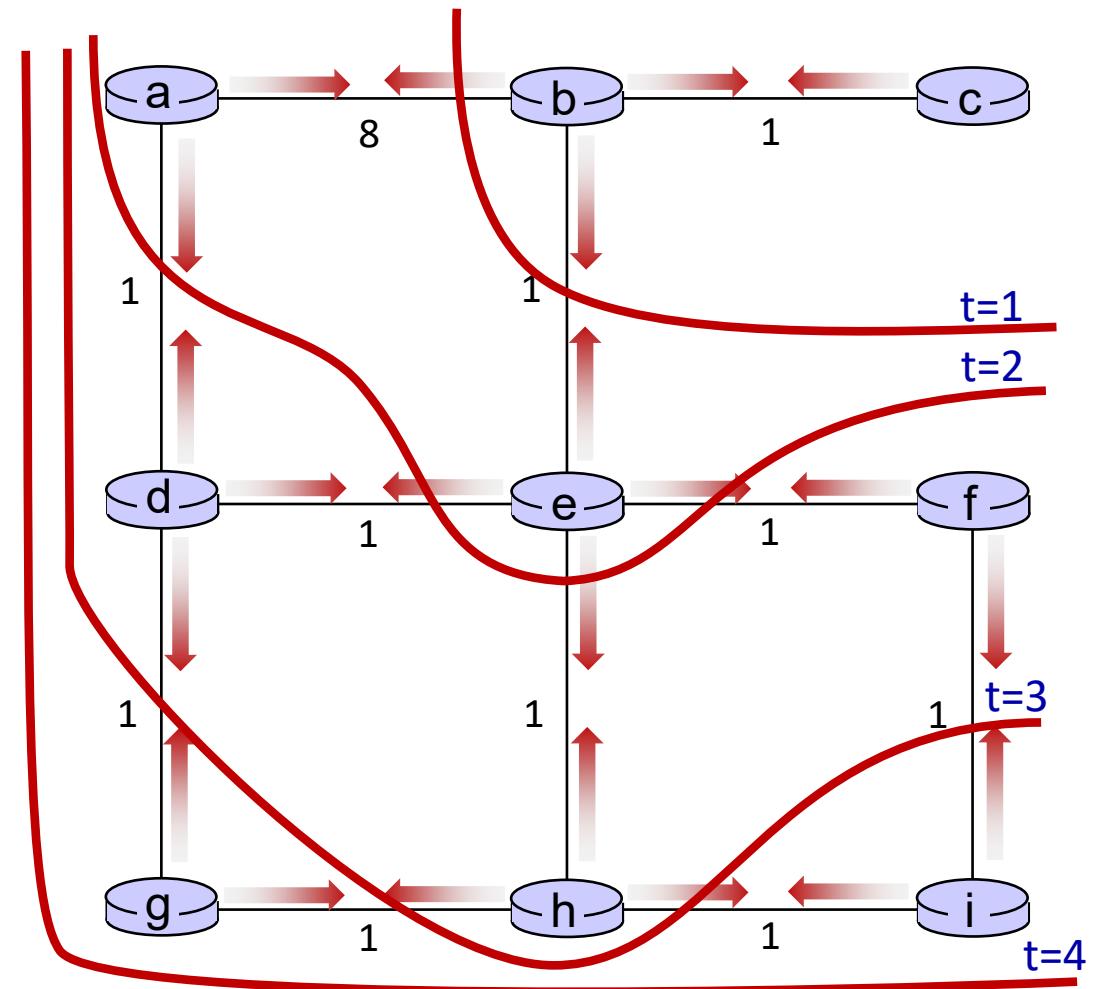
$v$ 's estimated least-cost-path cost to  $y$

direct cost of link from  $x$  to  $v$

# Distance vector: state information diffusion

Iterative communication, computation steps diffuses information through network:

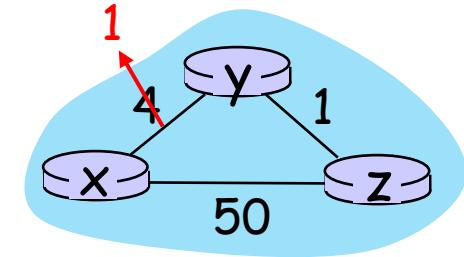
-  t=0 c's state at t=0 is at c only
-  t=1 c's state at t=0 has propagated to b, and may influence distance vector computations up to **1** hop away, i.e., at b
-  t=2 c's state at t=0 may now influence distance vector computations up to **2** hops away, i.e., at b and now at a, e as well
-  t=3 c's state at t=0 may influence distance vector computations up to **3** hops away, i.e., at b,a,e and now at c,f,h as well
-  t=4 c's state at t=0 may influence distance vector computations up to **4** hops away, i.e., at b,a,e, c, f, h and now at g,i as well



# Distance vector: link cost changes

## link cost changes:

- node detects local link cost change
- updates routing info, recalculates local DV
- if DV changes, notify neighbors



$t_0$ : y detects link-cost change, updates its DV, informs its neighbors.

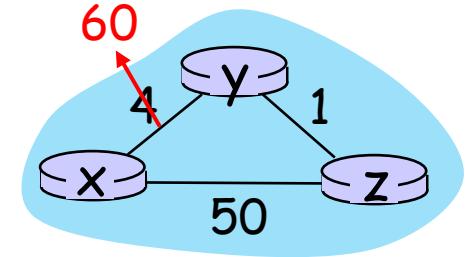
**“good news travels fast”**  $t_1$ : z receives update from y, updates its table, computes new least cost to x , sends its neighbors its DV.

$t_2$ : y receives z’s update, updates its distance table. y’s least costs do *not* change, so y does *not* send a message to z.

# Distance vector: link cost changes

## link cost changes:

- node detects local link cost change
- “**bad news travels slow**” – count-to-infinity problem:
  - y sees direct link to x has new cost 60, but z has said it has a path at cost of 5. So y computes “my new cost to x will be 6, via z); notifies z of new cost of 6 to x.
  - z learns that path to x via y has new cost 6, so z computes “my new cost to x will be 7 via y), notifies y of new cost of 7 to x.
  - y learns that path to x via z has new cost 7, so y computes “my new cost to x will be 8 via y), notifies z of new cost of 8 to x.
  - z learns that path to x via y has new cost 8, so z computes “my new cost to x will be 9 via y), notifies y of new cost of 9 to x.
  - ...



# Comparison of LS and DV algorithms

## message complexity

LS:  $n$  routers,  $O(n^2)$  messages sent

DV: exchange between neighbors;  
convergence time varies

## speed of convergence

LS:  $O(n^2)$  algorithm,  $O(n^2)$  messages

- may have oscillations

DV: convergence time varies

- may have routing loops
- count-to-infinity problem

**robustness:** what happens if router malfunctions, or is compromised?

LS:

- router can advertise incorrect *link* cost
- each router computes only its *own* table

DV:

- DV router can advertise incorrect *path* cost (“I have a *really* low cost path to everywhere”): black-holing
- each router’s table used by others: error propagate thru network

# Network layer: “control plane” roadmap

- introduction
- routing protocols
- intra-ISP routing: OSPF



# Making routing scalable

our routing study thus far - idealized

- all routers identical
- network “flat”

... not true in practice

**scale:** billions of destinations:

- can't store all destinations in routing tables!
- routing table exchange would swamp links!

**administrative autonomy:**

- Internet: a network of networks
- each network admin may want to control routing in its own network

# Internet approach to scalable routing

aggregate routers into regions known as “autonomous systems” (AS) (a.k.a. “domains”)

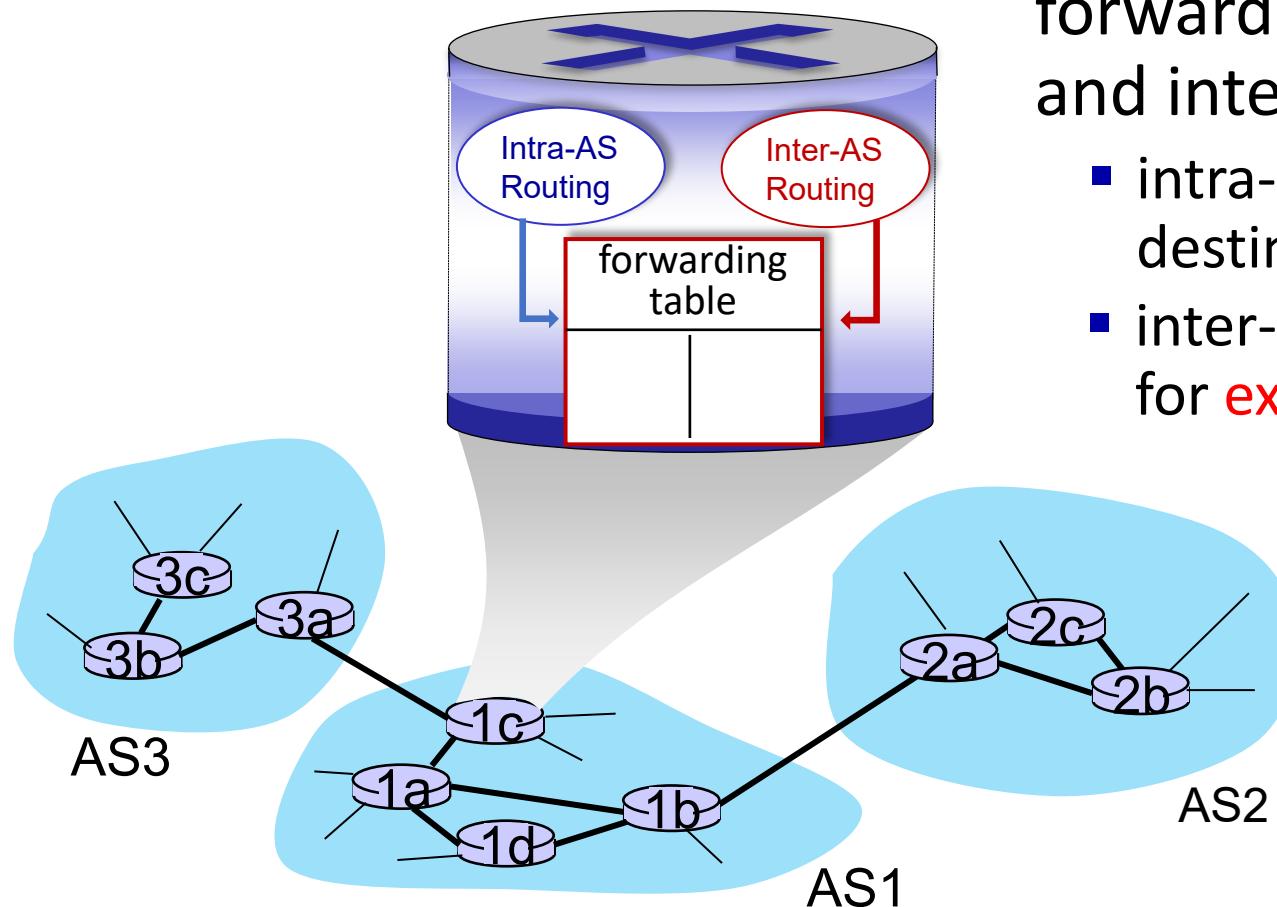
**intra-AS (aka “intra-domain”):**  
routing among *within same AS (“network”)*

- all routers in AS must run same intra-domain protocol
- routers in different AS can run different intra-domain routing protocols
- **gateway router:** at “edge” of its own AS, has link(s) to router(s) in other AS'es

**inter-AS (aka “inter-domain”):**  
routing *among* AS'es

- gateways perform inter-domain routing (as well as intra-domain routing)

# Interconnected ASes



forwarding table configured by intra-  
and inter-AS routing algorithms

- intra-AS routing determine entries for destinations **within** AS
- inter-AS & intra-AS determine entries for **external** destinations

# Inter-AS routing: routing within an AS

most common intra-AS routing protocols:

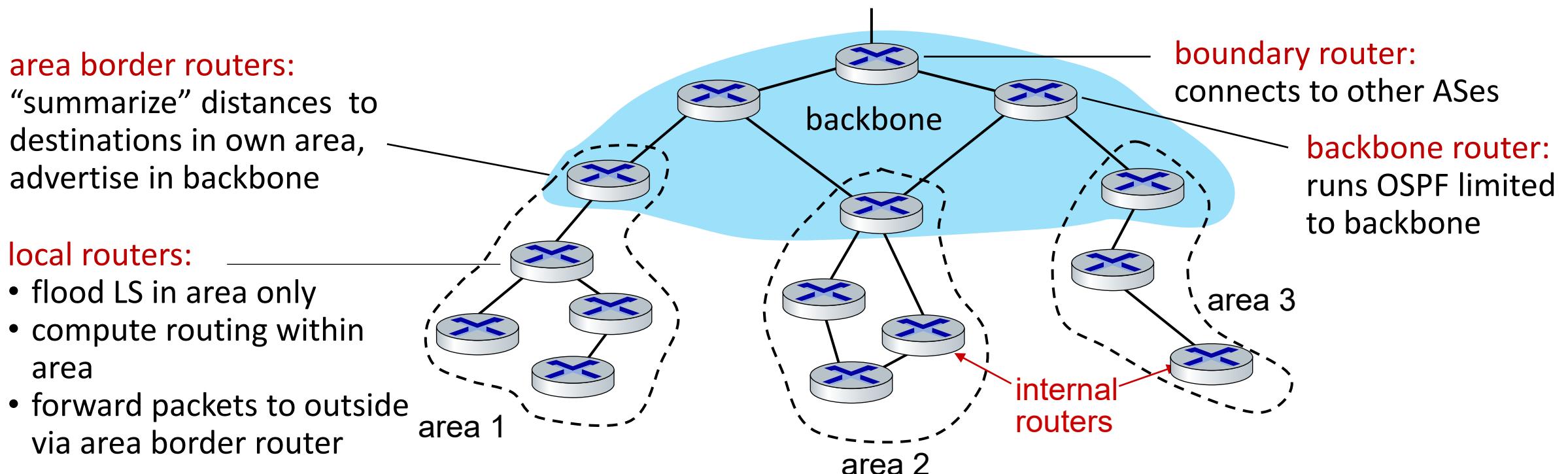
- **RIP: Routing Information Protocol [RFC 1723]**
  - classic DV: DVs exchanged every 30 secs
  - no longer widely used
- **EIGRP: Enhanced Interior Gateway Routing Protocol**
  - DV based
  - formerly Cisco-proprietary for decades (became open in 2013 [RFC 7868])
- **OSPF: Open Shortest Path First [RFC 2328]**
  - link-state routing
  - IS-IS protocol (ISO standard, not RFC standard) essentially same as OSPF

# OSPF (Open Shortest Path First) routing

- “open”: publicly available
- classic link-state
  - each router floods OSPF link-state advertisements (directly over IP rather than using TCP/UDP) to all other routers in entire AS
  - multiple link costs metrics possible: bandwidth, delay
  - each router has full topology, uses Dijkstra’s algorithm to compute forwarding table
- *security*: all OSPF messages authenticated (to prevent malicious intrusion)

# Hierarchical OSPF

- **two-level hierarchy:** local area, backbone.
  - link-state advertisements flooded only in area, or backbone
  - each node has detailed area topology; only knows direction to reach other destinations



# Network layer: “control plane” roadmap

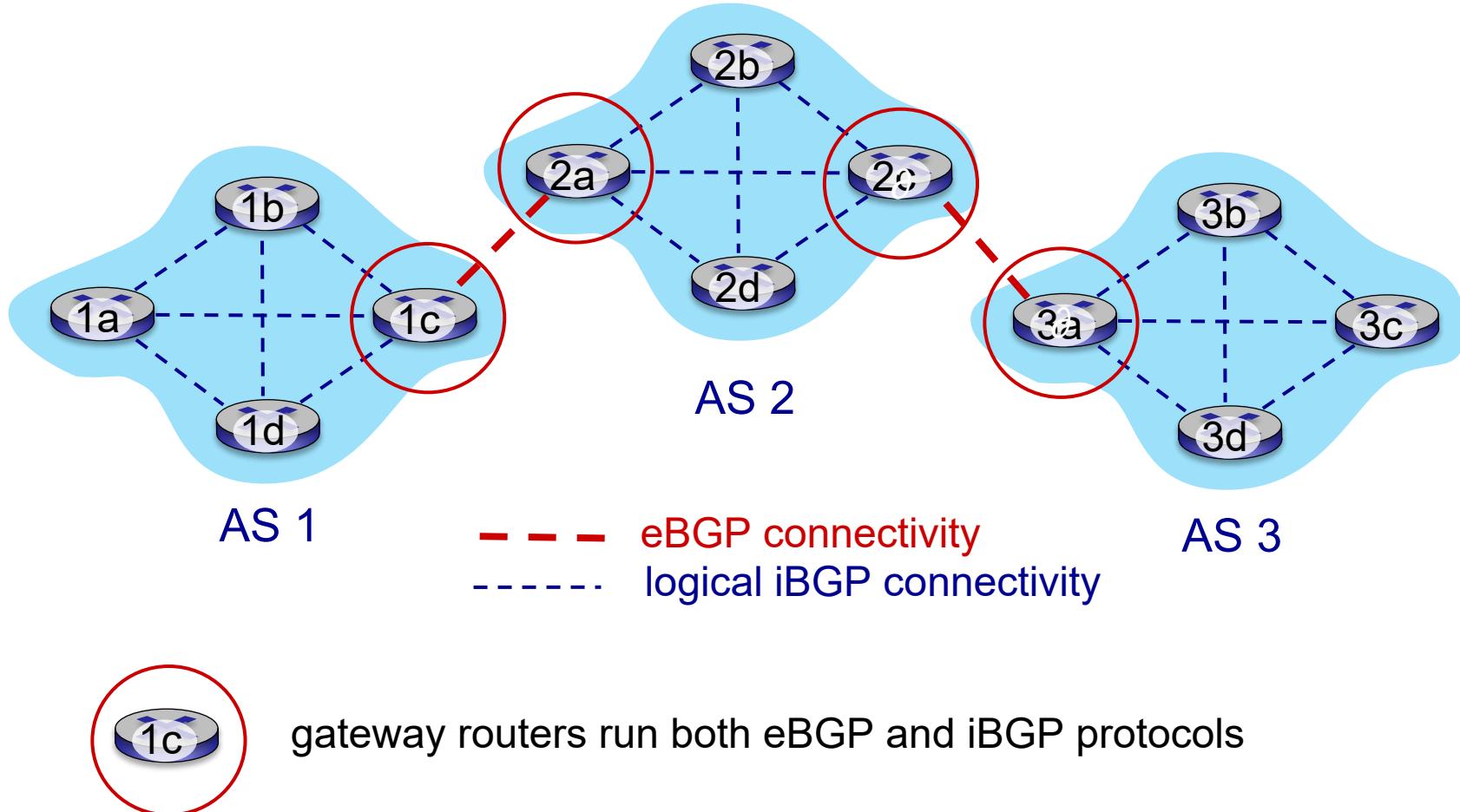
- introduction
- routing protocols
- intra-ISP routing: OSPF
- **routing among ISPs: BGP**



# Internet inter-AS routing: BGP

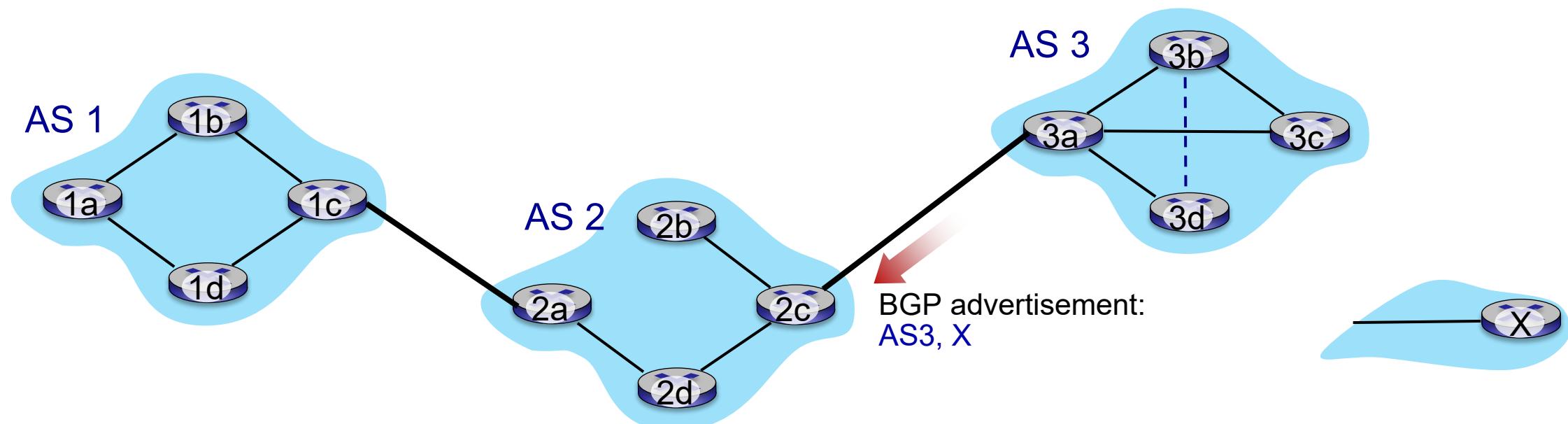
- BGP (Border Gateway Protocol): *the de facto* inter-domain routing protocol
  - “glue that holds the Internet together”
- allows subnet to advertise its existence, and the destinations it can reach, to rest of Internet: *“I am here, here is who I can reach, and how”*
- BGP provides each AS a means to:
  - eBGP: obtain subnet reachability information from neighboring ASes
  - iBGP: propagate reachability information to all AS-internal routers.
  - determine “good” routes to other networks based on reachability information and *policy*

# eBGP, iBGP connections



# BGP basics

- **BGP session:** two BGP routers (“peers”) exchange BGP messages over semi-permanent TCP connection:
  - advertising *paths* to different destination network prefixes (BGP is a “path vector” protocol)
- when AS3 gateway 3a advertises **path AS3,X** to AS2 gateway 2c:
  - AS3 *promises* to AS2 it will forward datagrams towards X



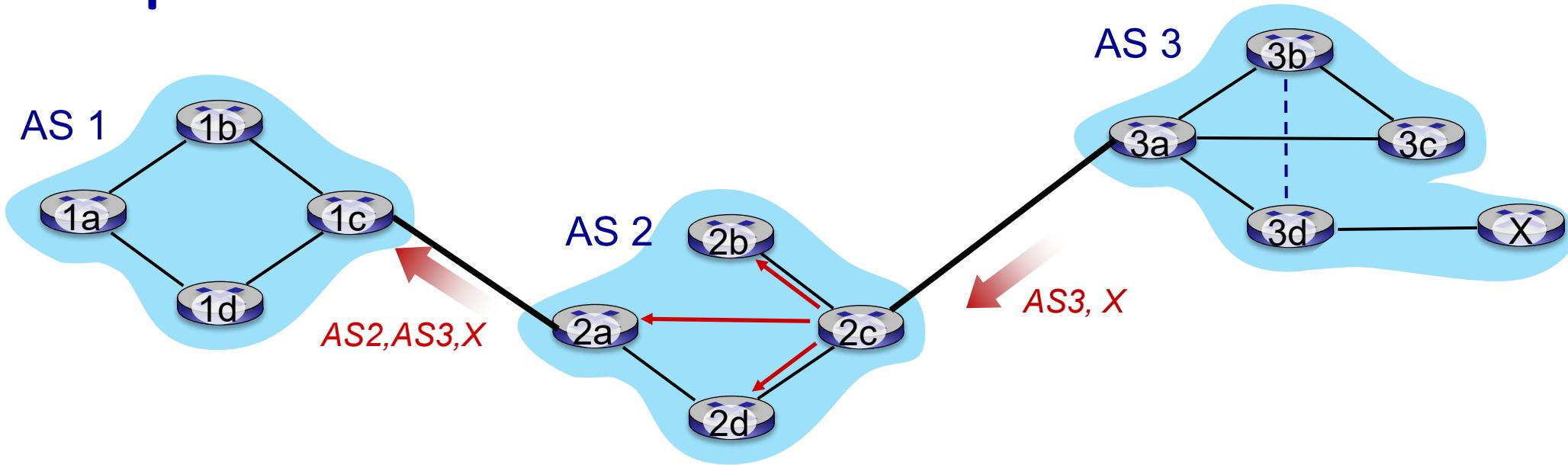
# BGP messages

- BGP messages exchanged between peers over TCP connection
- BGP messages:
  - **OPEN**: opens TCP connection to remote BGP peer and authenticates sending BGP peer
  - **UPDATE**: advertises new path (or withdraws old)
  - **KEEPALIVE**: keeps connection alive in absence of UPDATES; also ACKs OPEN request
  - **NOTIFICATION**: reports errors in previous msg; also used to close connection

# Path attributes and BGP routes

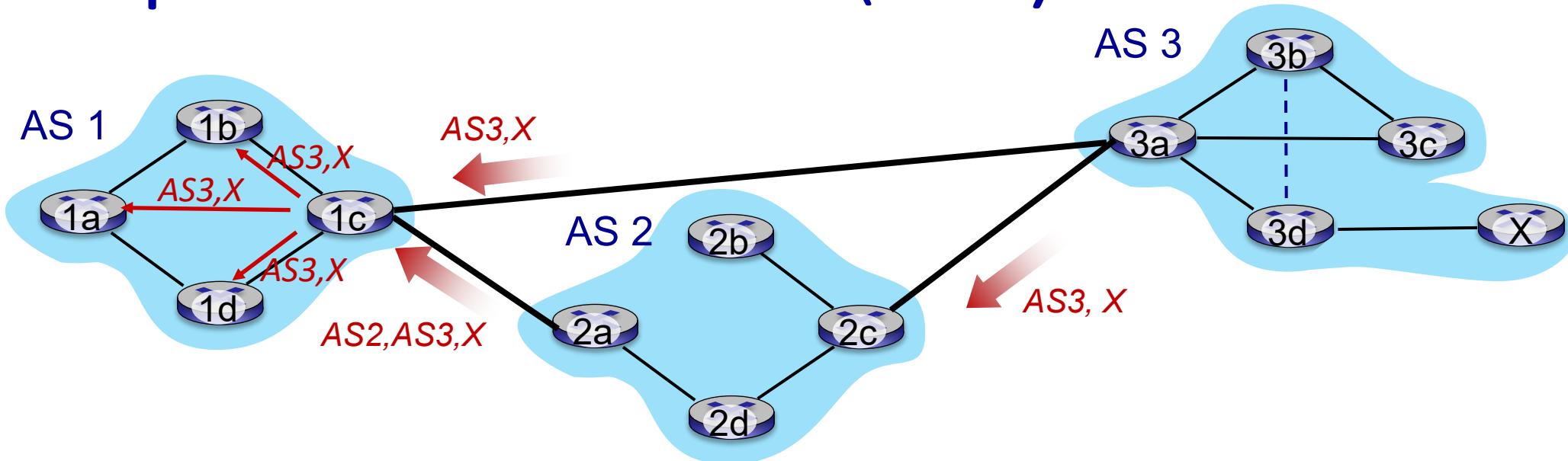
- BGP advertised route: prefix + attributes
  - prefix: destination being advertised
  - two important attributes:
    - AS-PATH: list of ASes through which prefix advertisement has passed
    - NEXT-HOP: indicates specific internal-AS router to next-hop AS
- policy-based routing:
  - gateway receiving route advertisement uses *import policy* to accept/decline path (e.g., never route through AS Y).
  - AS policy also determines whether to *advertise* path to other other neighboring ASes

# BGP path advertisement



- AS2 router 2c receives path advertisement **AS3,X** (via eBGP) from AS3 router 3a
- based on AS2 policy, AS2 router 2c accepts path AS3,X, propagates (via iBGP) to all AS2 routers
- based on AS2 policy, AS2 router 2a advertises (via eBGP) path **AS2, AS3, X** to AS1 router 1c

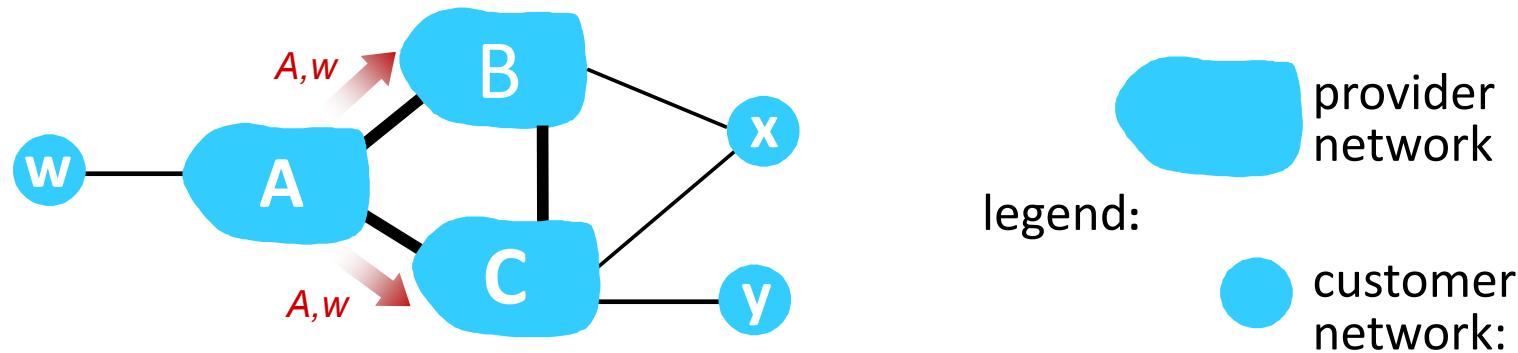
# BGP path advertisement (more)



gateway router may learn about multiple paths to destination:

- AS1 gateway router 1c learns path **AS2,AS3,X** from 2a
- AS1 gateway router 1c learns path **AS3,X** from 3a
- based on *policy*, AS1 gateway router 1c chooses path **AS3,X** and advertises path within AS1 via iBGP

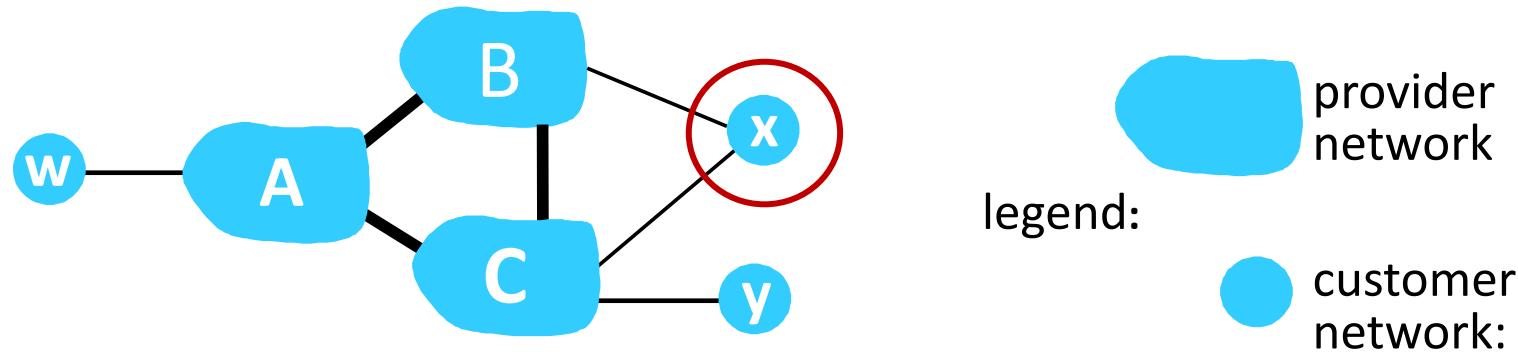
# BGP: achieving policy via advertisements



ISP only wants to route traffic to/from its customer networks (does not want to carry transit traffic between other ISPs – a typical “real world” policy)

- A advertises path  $Aw$  to B and to C
- B *chooses not to advertise*  $BAw$  to C!
  - B gets no “revenue” for routing  $BAw$ , since none of C, A, w are B’s customers
  - C does *not* learn about  $BAw$  path
- C will route  $CAw$  (not using B) to get to w

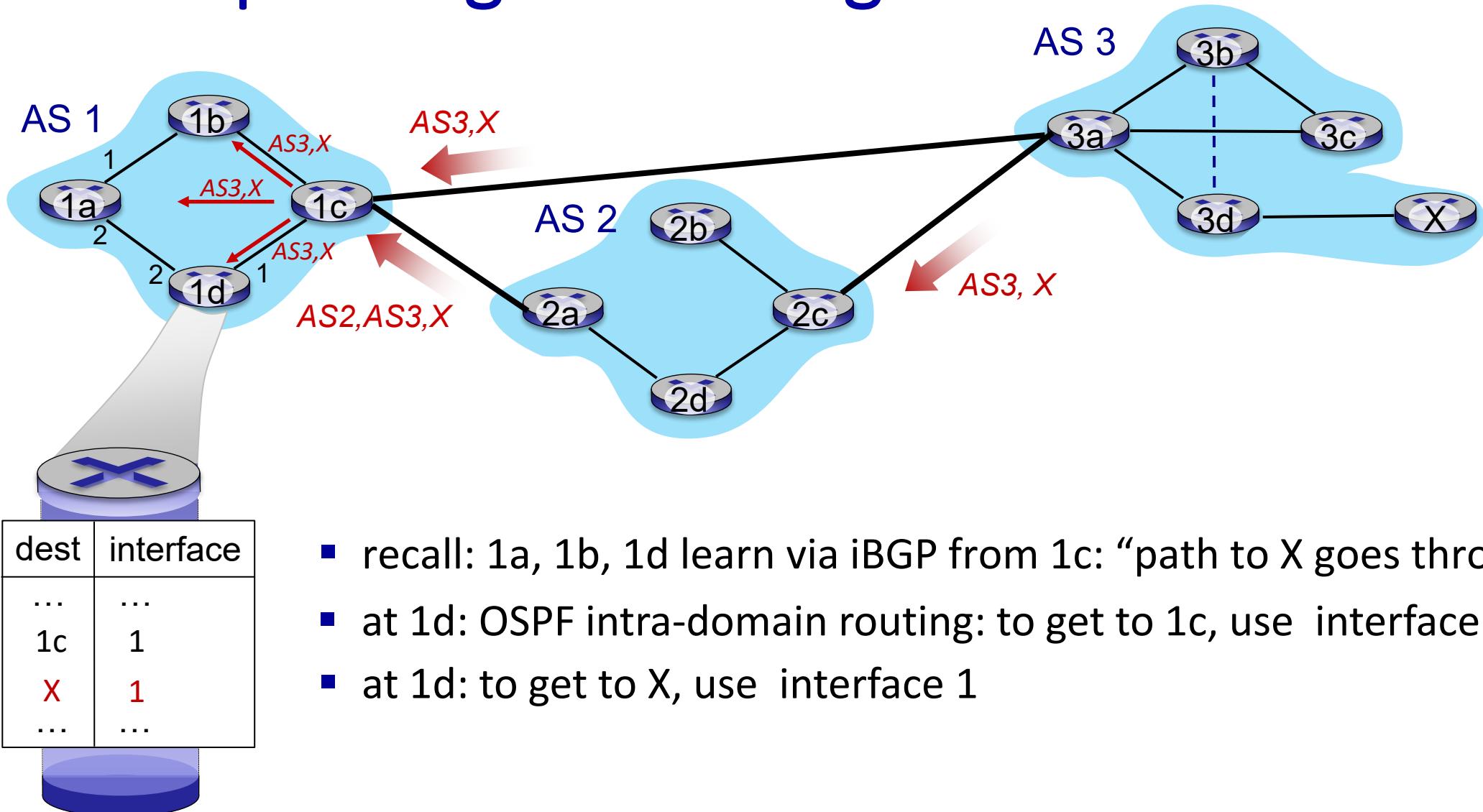
# BGP: achieving policy via advertisements (more)



ISP only wants to route traffic to/from its customer networks (does not want to carry transit traffic between other ISPs – a typical “real world” policy)

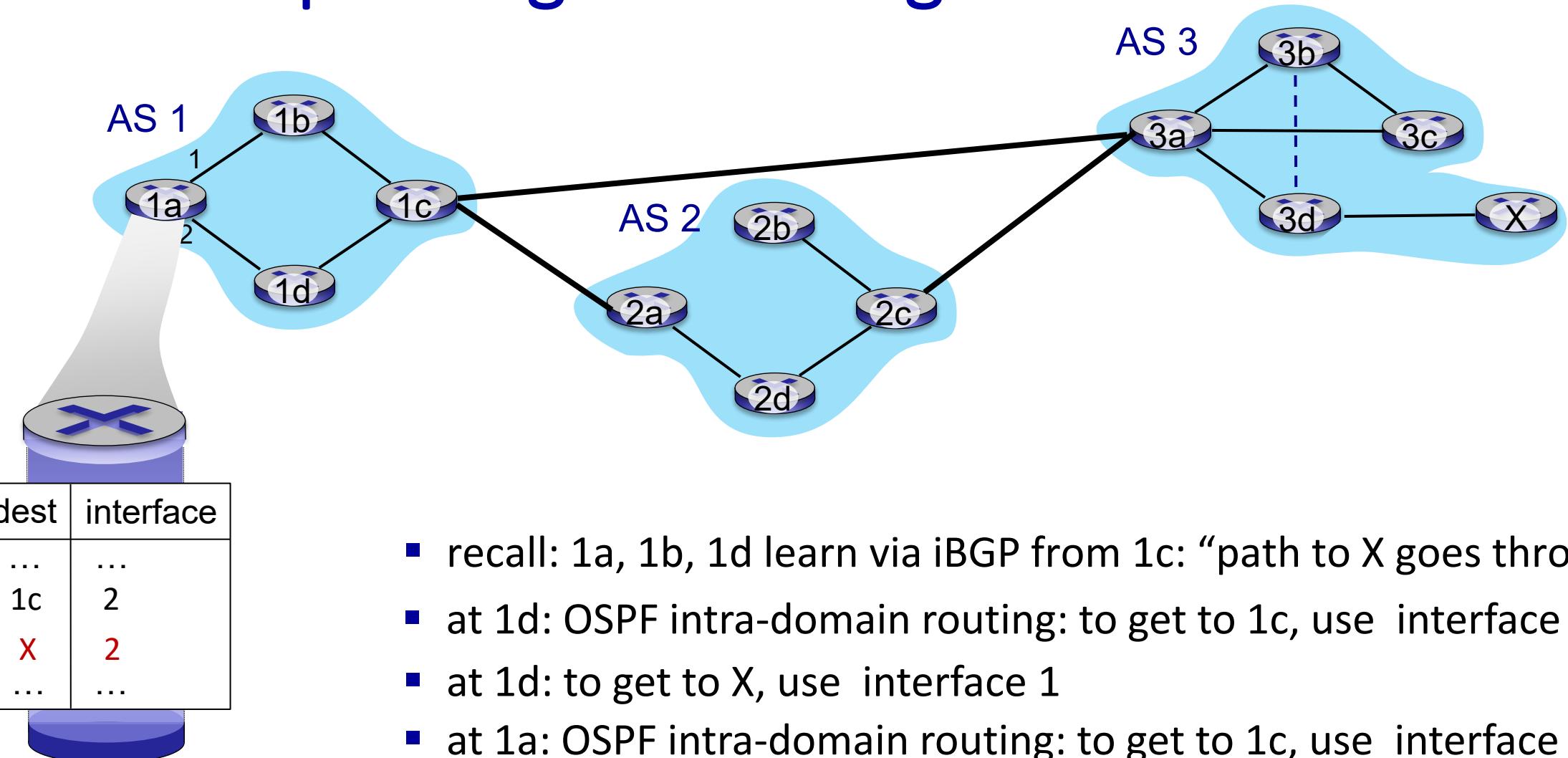
- A,B,C are **provider networks**
- x,w,y are **customer** (of provider networks)
- x is **dual-homed**: attached to two networks
- **policy to enforce**: x does not want to route from B to C via x
  - .. so x will not advertise to B a route to C

# BGP: Populating forwarding tables



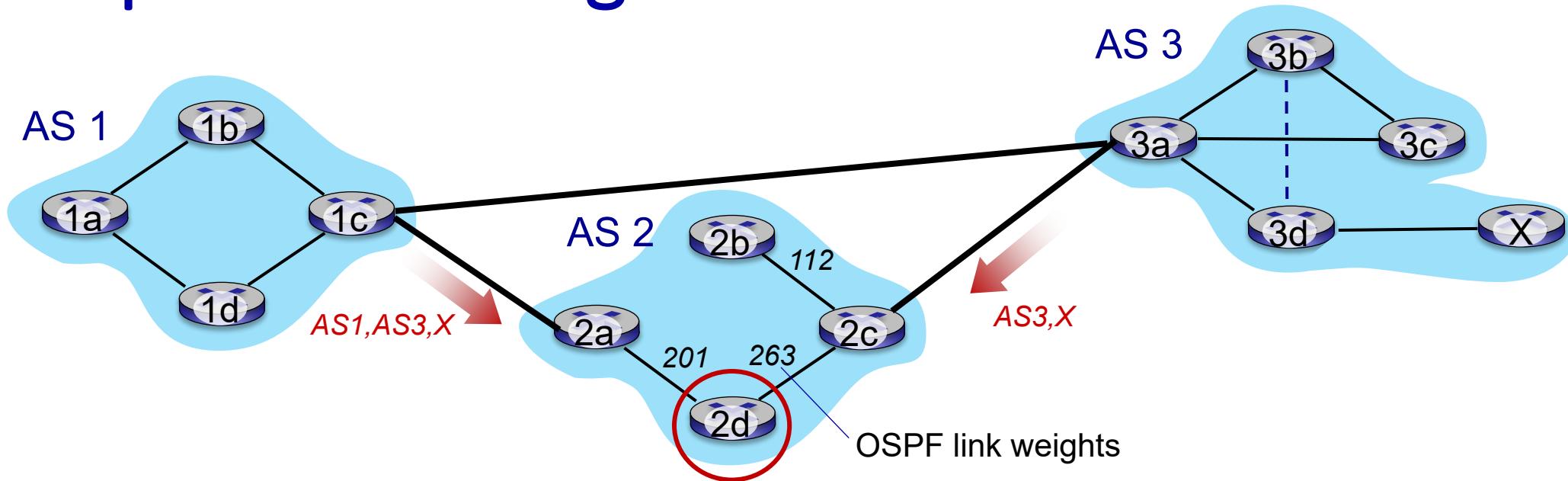
- recall: 1a, 1b, 1d learn via iBGP from 1c: “path to X goes through 1c”
- at 1d: OSPF intra-domain routing: to get to 1c, use interface 1
- at 1d: to get to X, use interface 1

# BGP Populating forwarding tables



- recall: 1a, 1b, 1d learn via iBGP from 1c: “path to X goes through 1c”
- at 1d: OSPF intra-domain routing: to get to 1c, use interface 1
- at 1d: to get to X, use interface 1
- at 1a: OSPF intra-domain routing: to get to 1c, use interface 2
- at 1a: to get to X, use interface 2

# Hot potato routing



- 2d learns (via iBGP) it can route to X via 2a or 2c
- **hot potato routing:** choose local gateway that has least *intra-domain* cost (e.g., 2d chooses 2a, even though more AS hops to X): don't worry about inter-domain cost!

# BGP route selection

- router may learn about more than one route to destination AS, selects route based on:
  1. local preference value attribute: policy decision
  2. shortest AS-PATH
  3. closest NEXT-HOP router: hot potato routing
  4. additional criteria

# Why different Intra-, Inter-AS routing ?

policy:

- inter-AS: admin wants control over how its traffic routed, who routes through its network
- intra-AS: single admin, so policy less of an issue

scale:

- hierarchical routing saves table size, reduced update traffic

performance:

- intra-AS: can focus on performance
- inter-AS: policy dominates over performance

# Network layer: “control plane” roadmap

- introduction
- routing protocols
- intra-ISP routing: OSPF
- routing among ISPs: BGP
- **Internet Control Message Protocol**

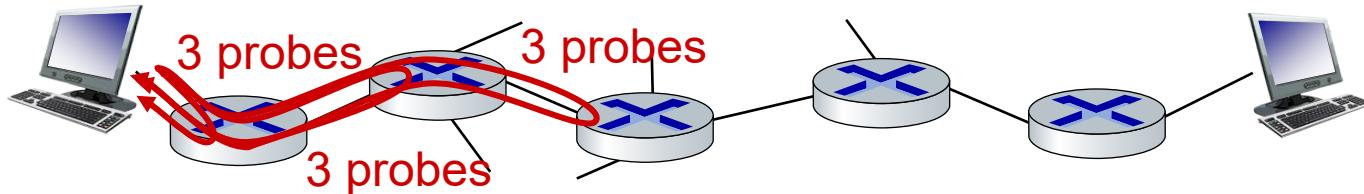


# ICMP: internet control message protocol

- used by hosts and routers to communicate network-level information
  - error reporting: unreachable host, network, port, protocol
  - echo request/reply (used by ping)
- network-layer “above” IP:
  - ICMP messages carried in IP datagrams
- *ICMP message*: type, code plus first 8 bytes of IP datagram causing error

| Type | Code | description                                   |
|------|------|-----------------------------------------------|
| 0    | 0    | echo reply (ping)                             |
| 3    | 0    | dest. network unreachable                     |
| 3    | 1    | dest host unreachable                         |
| 3    | 2    | dest protocol unreachable                     |
| 3    | 3    | dest port unreachable                         |
| 3    | 6    | dest network unknown                          |
| 3    | 7    | dest host unknown                             |
| 4    | 0    | source quench (congestion control - not used) |
| 8    | 0    | echo request (ping)                           |
| 9    | 0    | route advertisement                           |
| 10   | 0    | router discovery                              |
| 11   | 0    | TTL expired                                   |
| 12   | 0    | bad IP header                                 |

# Traceroute and ICMP



- source sends sets of UDP segments to destination
  - 1<sup>st</sup> set has TTL =1, 2<sup>nd</sup> set has TTL=2, etc.
- datagram in *n*th set arrives to *n*th router:
  - router discards datagram and sends source ICMP message (type 11, code 0)
  - ICMP message possibly includes name of router & IP address
- when ICMP message arrives at source: record RTTs

## stopping criteria:

- UDP segment eventually arrives at destination host
- destination returns ICMP “port unreachable” message (type 3, code 3)
- source stops



# Computer Networks

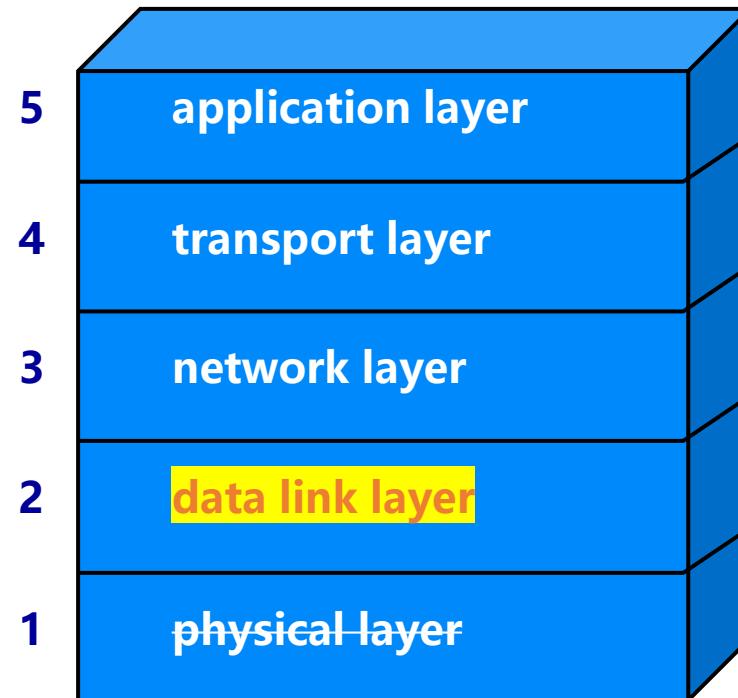
Lecturer: ZHANG Ying  
Fall semester 2022

# FINAL EXAM

- Time: November 17<sup>th</sup>, Exam time will be 16:00 – 18:00 (Next Thursday), all students should enter platform at 15:30 for identity check with your student card and passport.
- Platform: Tencent Meeting (Meeting code will be provided 35 min before)
- Closed-book paper-based exam.
- Setting requirements:
  - Camera-based live streaming. The camera should cover NOT ONLY the face, BUT ALSO the desk and the computer screen in a light- sufficient environment.
  - Mic should be OPEN all the time.
  - Ask question ONLY with mic
  - Strictly no other communication
- The detailed policy will be shared for everyone soon and we may have an environment check before exam next week. (Tentative time: 15<sup>th</sup> Nov, 16:00)

# Chapter 6

# Data Link Layer



# Outline

- introduction
- error detection, correction
- multiple access protocols
- LANs

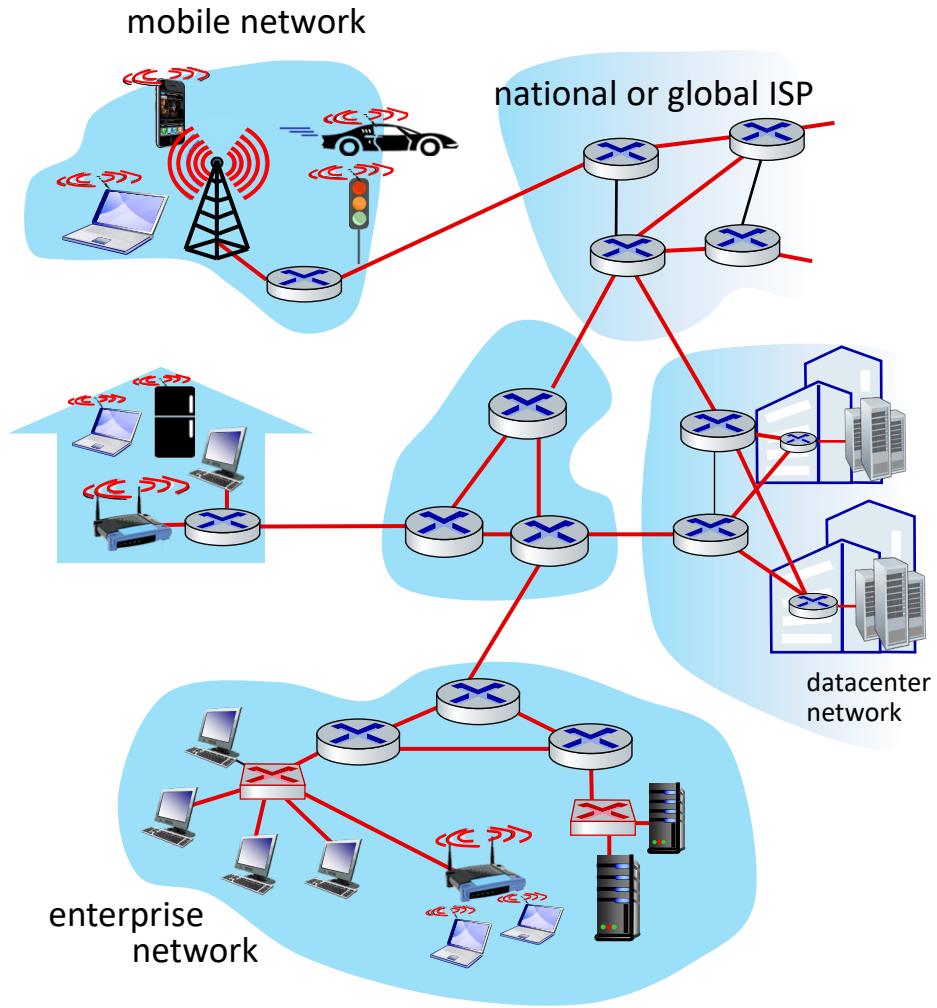


# Link layer: introduction

terminology:

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

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



# Link layer: context

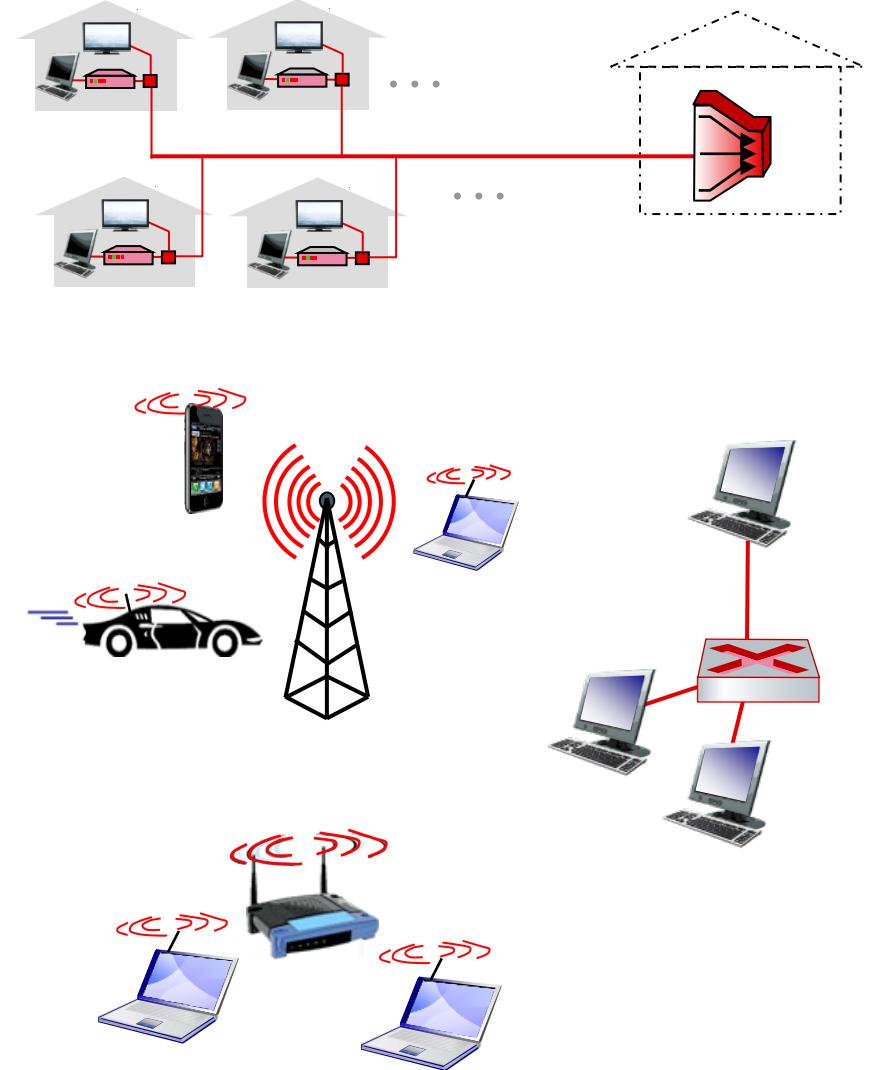
- datagram transferred by different link protocols over different links:
  - e.g., WiFi on first link, Ethernet on next link
- each link protocol provides different services
  - e.g., may or may not provide reliable data transfer over link

## transportation analogy:

- trip from Princeton to Lausanne
  - limo: Princeton to JFK
  - plane: JFK to Geneva
  - train: Geneva to Lausanne
- tourist = **datagram**
- transport segment = **communication link**
- transportation mode = **link-layer protocol**
- travel agent = **routing algorithm**

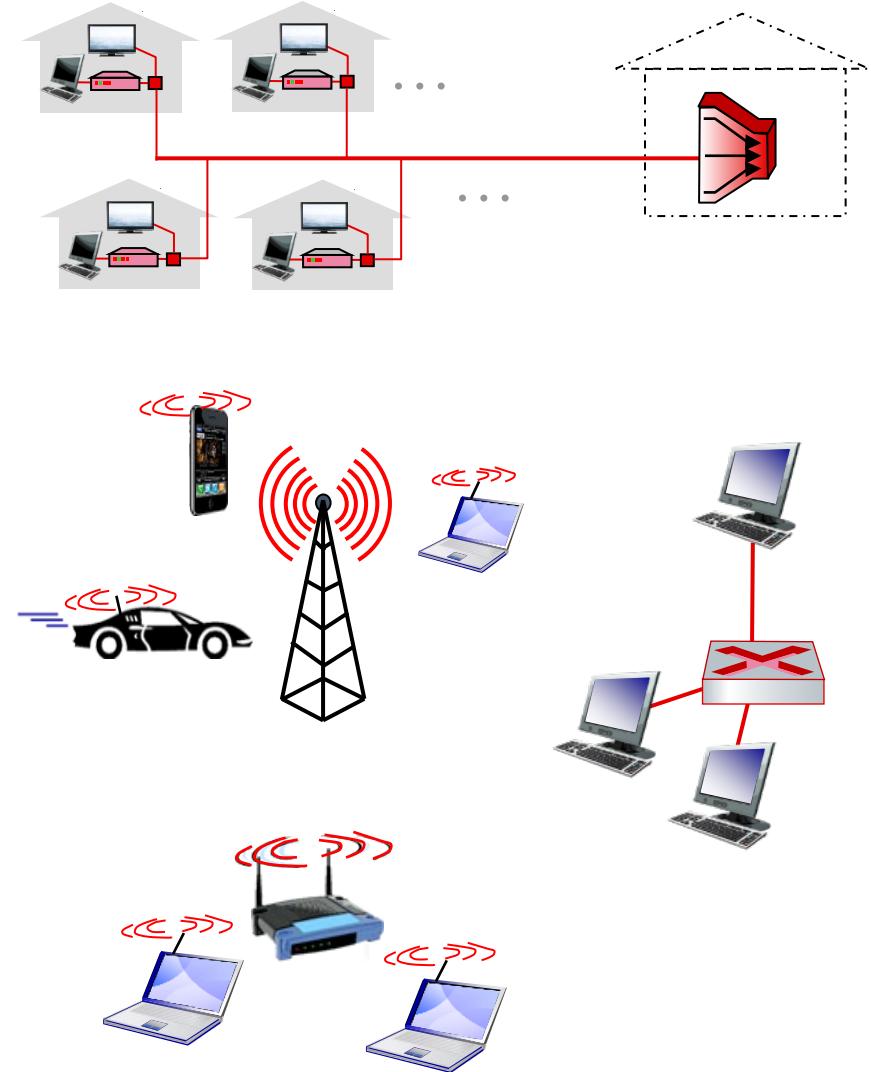
# Link layer: services

- **framing, link access:**
  - encapsulate datagram into frame, adding header, trailer
  - channel access if shared medium
  - “MAC” addresses in frame headers identify source, destination (different from IP address!)
- **reliable delivery between adjacent nodes**
  - we already know how to do this!
  - seldom used on low bit-error links
  - wireless links: high error rates
    - Q: why both link-level and end-end reliability?



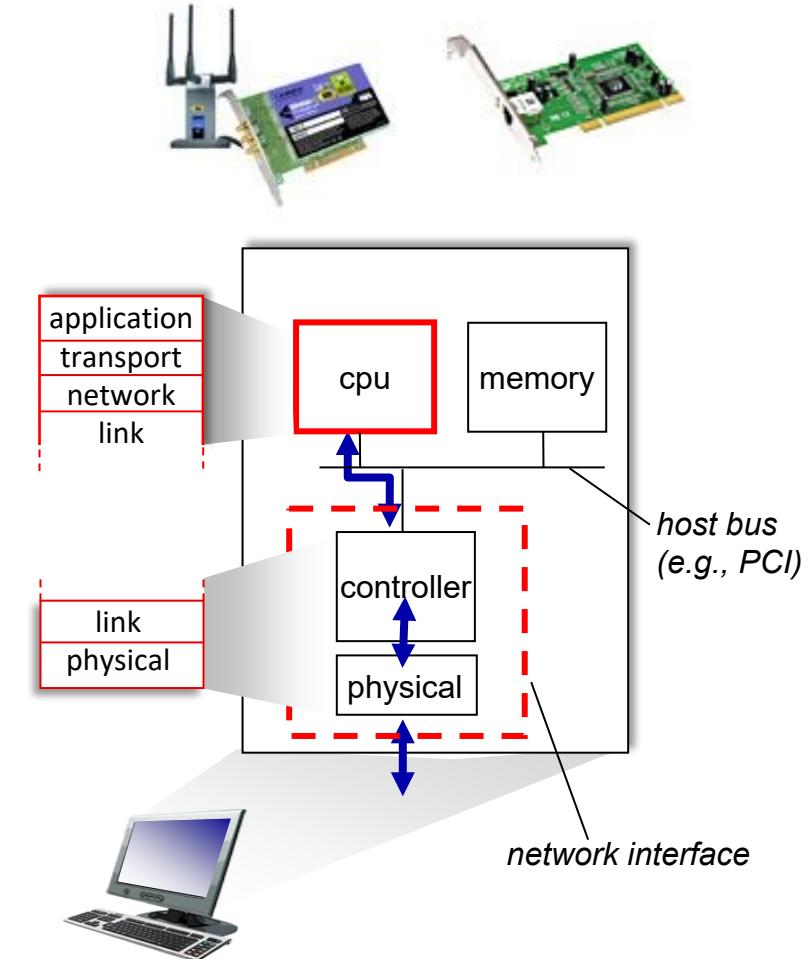
# Link layer: services (more)

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

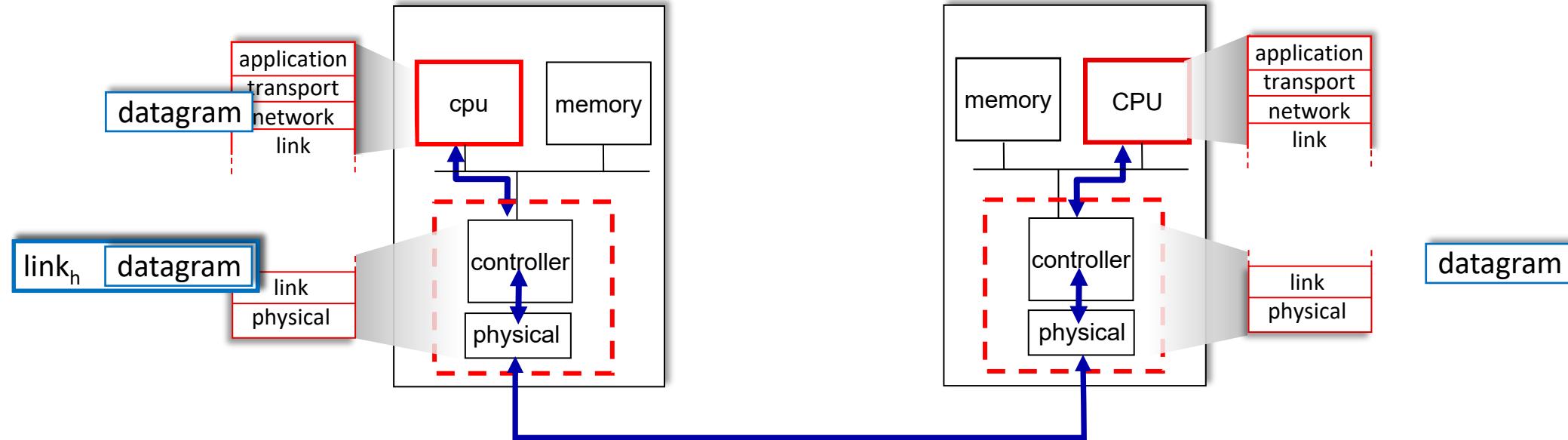


# Where is the link layer implemented?

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



# Interfaces communicating



sending side:

- encapsulates datagram in frame
- adds error checking bits, reliable data transfer, flow control, etc.

receiving side:

- looks for errors, reliable data transfer, flow control, etc.
- extracts datagram, passes to upper layer at receiving side

# outline

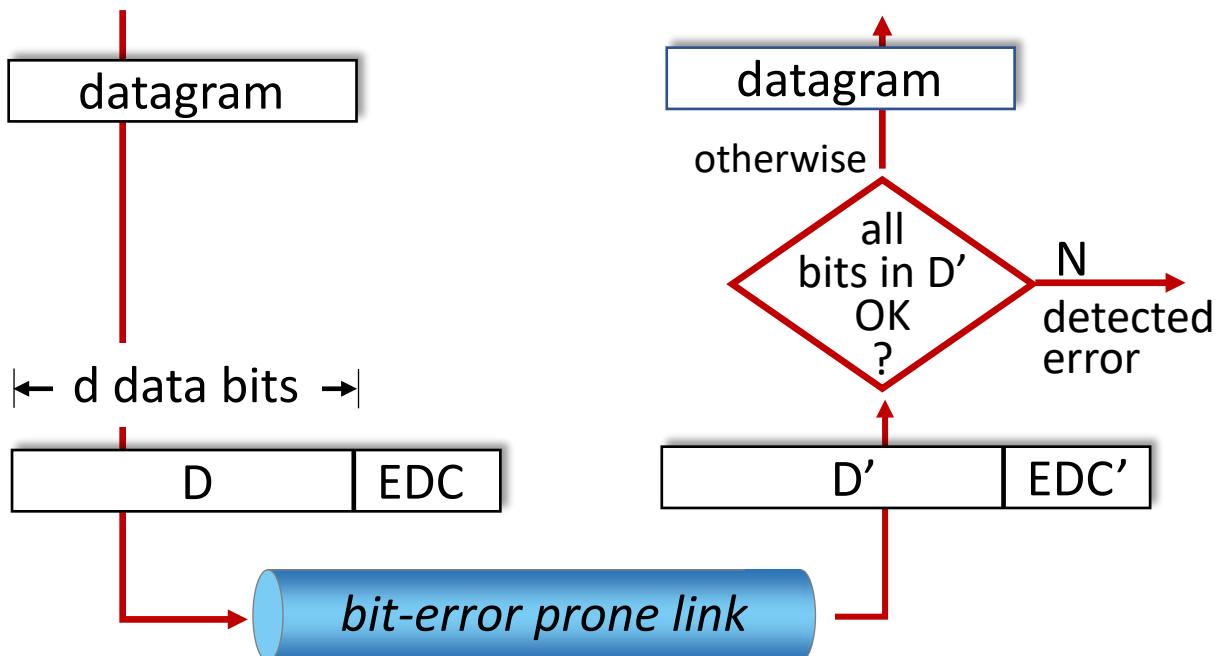
- introduction
- **error detection, correction**
- multiple access protocols
- LANs



# Error detection

EDC: error detection and correction bits (e.g., redundancy)

D: data protected by error checking, may include header fields



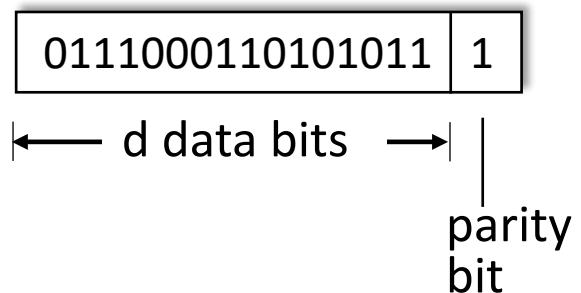
Error detection not 100% reliable!

- protocol may miss some errors, but rarely
- larger EDC field yields better detection and correction

# Parity checking

single bit parity:

- detect single bit errors

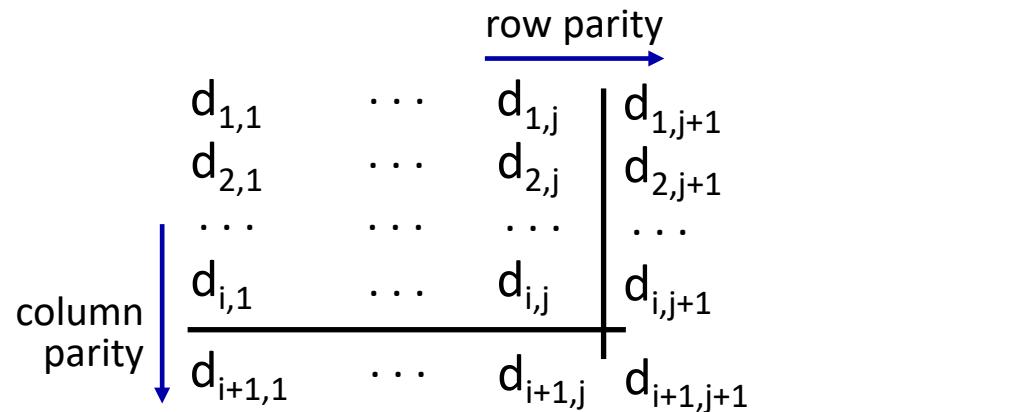


Even parity: set parity bit so there is an even number of 1's

# Parity checking

two-dimensional bit parity:

- detect *and correct* single bit errors



no errors: 1 0 1 0 1 | 1  
1 1 1 1 0 | 0  
0 1 1 1 0 | 1  
----- |  
1 0 1 0 1 | 0

detected  
and  
correctable  
single-bit  
error:

1 0 1 0 1 | 1  
1 0 1 1 0 | 0 → parity error  
0 1 1 1 0 | 1  
----- |  
1 0 1 0 1 | 0  
↓ parity error

# Internet checksum (review)

**Goal:** detect errors (*i.e.*, flipped bits) in transmitted segment

sender:

- treat contents of UDP segment (including UDP header fields and IP addresses) as sequence of 16-bit integers
- **checksum:** addition (one's complement sum) of segment content
- checksum value put into UDP checksum field

receiver:

- compute checksum of received segment
- check if computed checksum equals checksum field value:
  - not equal - error detected
  - equal - no error detected. *But maybe errors nonetheless?* More later ....

# Cyclic Redundancy Check (CRC)

- more powerful error-detection coding
- **D**: data bits (given, think of these as a binary number)
- **G**: bit pattern (generator), of  $r+1$  bits (given)



goal: choose  $r$  CRC bits, **R**, such that  $\langle D, R \rangle$  exactly divisible by **G** ( $\text{mod } 2$ )

- receiver knows **G**, divides  $\langle D, R \rangle$  by **G**. If non-zero remainder: error detected!
- can detect all burst errors less than  $r+1$  bits
- widely used in practice (Ethernet, 802.11 WiFi)

# Cyclic Redundancy Check (CRC): example

## Steps:

1. find the length of the divisor
  2. Append L-1 bits to the original message
  3. Perform binary division operation.
  4. Reminder of the division = CRC

## We want:

$D \cdot 2^r \text{ } XOR \text{ } R = nG$

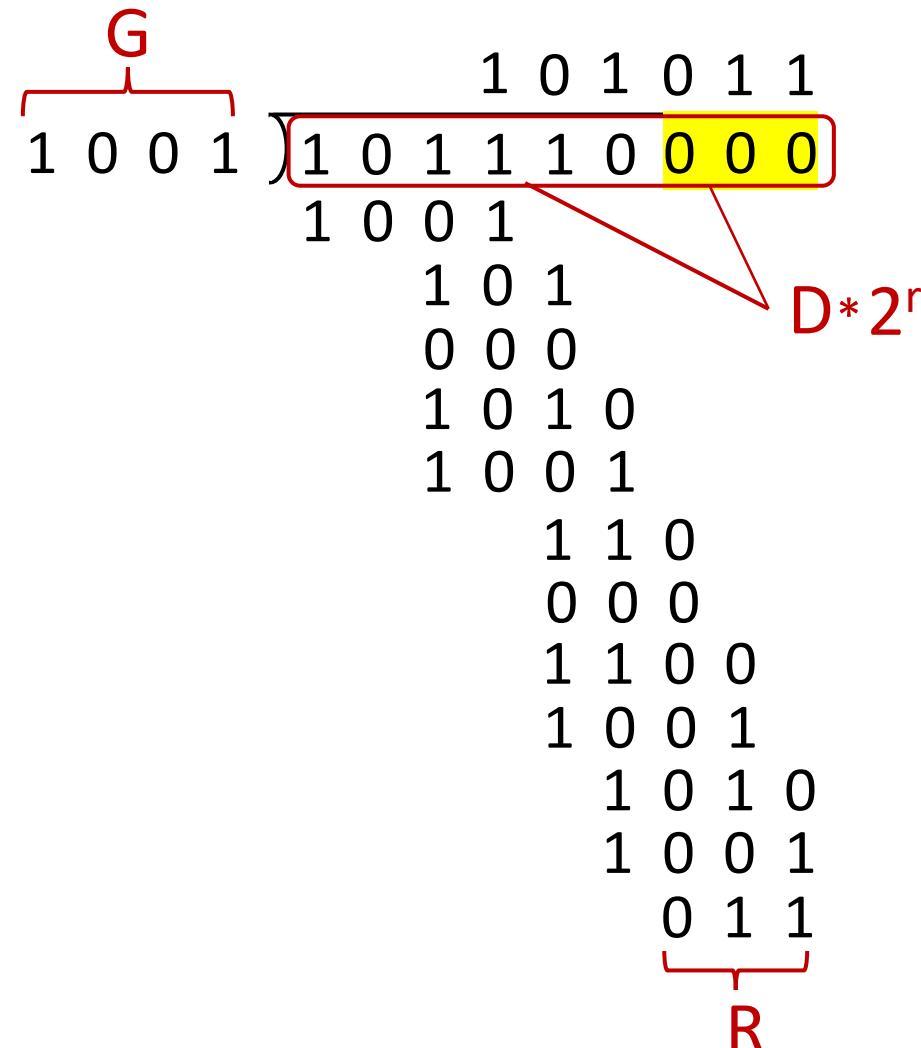
or equivalently:

$$D \cdot 2^r = nG \text{ } XOR \text{ } R$$

or equivalently:

if we divide  $D \cdot 2^r$  by  $G$ , want remainder  $R$  to satisfy:

$$R = \text{remainder} \left[ \frac{D \cdot 2^r}{G} \right]$$



# outline

- introduction
- error detection, correction
- **multiple access protocols**
- LANs



# Multiple access links, protocols

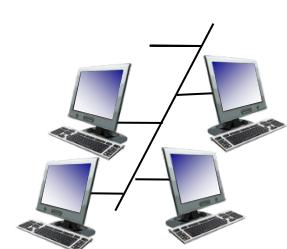
two types of “links”:

- point-to-point
  - point-to-point link between Ethernet switch, host
  - PPP for dial-up access
- broadcast (shared wire or medium)
  - old-fashioned Ethernet
  - upstream HFC in cable-based access network
  - 802.11 wireless LAN, 4G/4G satellite



humans at a cocktail party  
(shared air, acoustical)

# Multiple access links and protocols



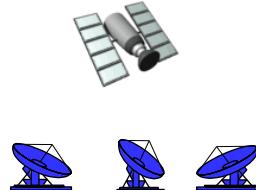
shared wire (e.g., cabled Ethernet)



shared radio: 4G/5G

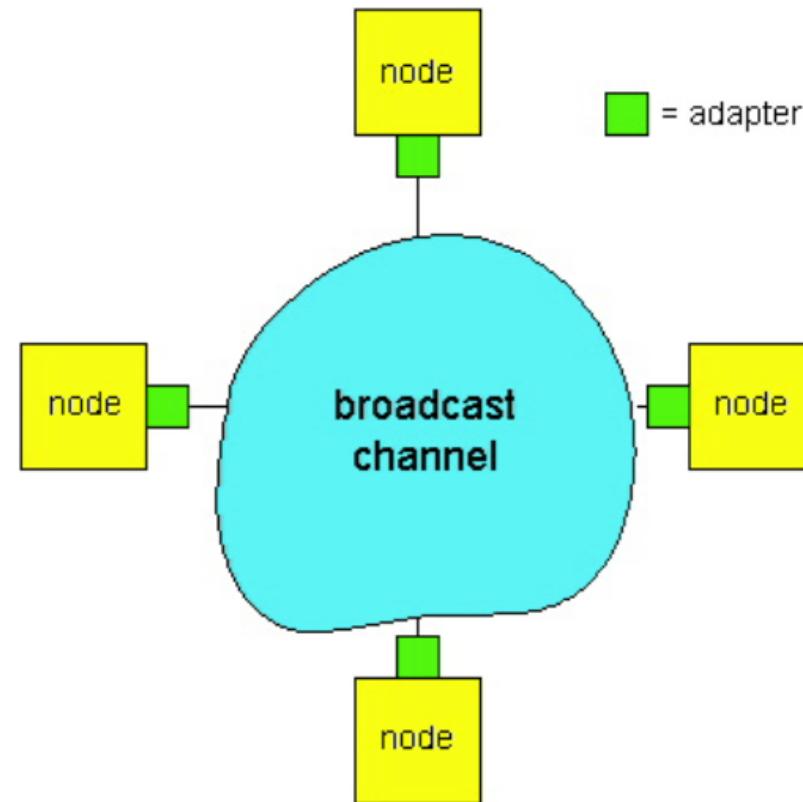


shared radio: WiFi



shared radio: satellite

■ = adapter



A broadcast channel interconnecting four nodes.

# Multiple access protocols

- single shared broadcast channel
- two or more simultaneous transmissions by nodes: interference
  - *collision* if node receives two or more signals at the same time

## multiple access protocol

- distributed algorithm that determines how nodes share channel, i.e., determine when node can transmit
- communication about channel sharing must use channel itself!
  - no out-of-band channel for coordination

# An ideal multiple access protocol

*given:* multiple access channel (MAC) of rate  $R$  bps

*desiderata:*

1. when one node wants to transmit, it can send at rate  $R$ .
2. when  $M$  nodes want to transmit, each can send at average rate  $R/M$
3. fully decentralized:
  - no special node to coordinate transmissions
  - no synchronization of clocks, slots
4. simple

# MAC protocols: taxonomy

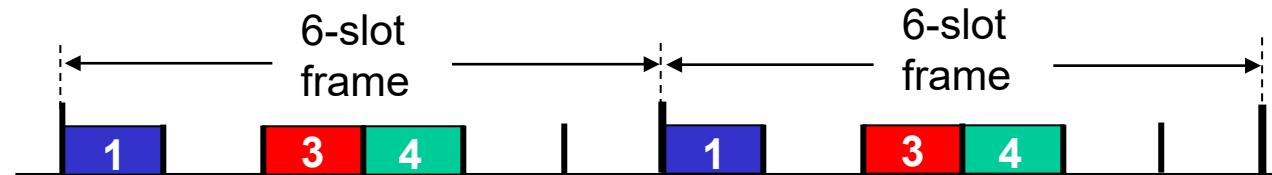
three broad classes:

- **channel partitioning**
  - divide channel into smaller “pieces” (time slots, frequency, code)
  - allocate piece to node for exclusive use
- ***random access***
  - channel not divided, allow collisions
  - “recover” from collisions
- **“taking turns”**
  - nodes take turns, but nodes with more to send can take longer turns

# Channel partitioning MAC protocols: TDMA

## TDMA: time division multiple access

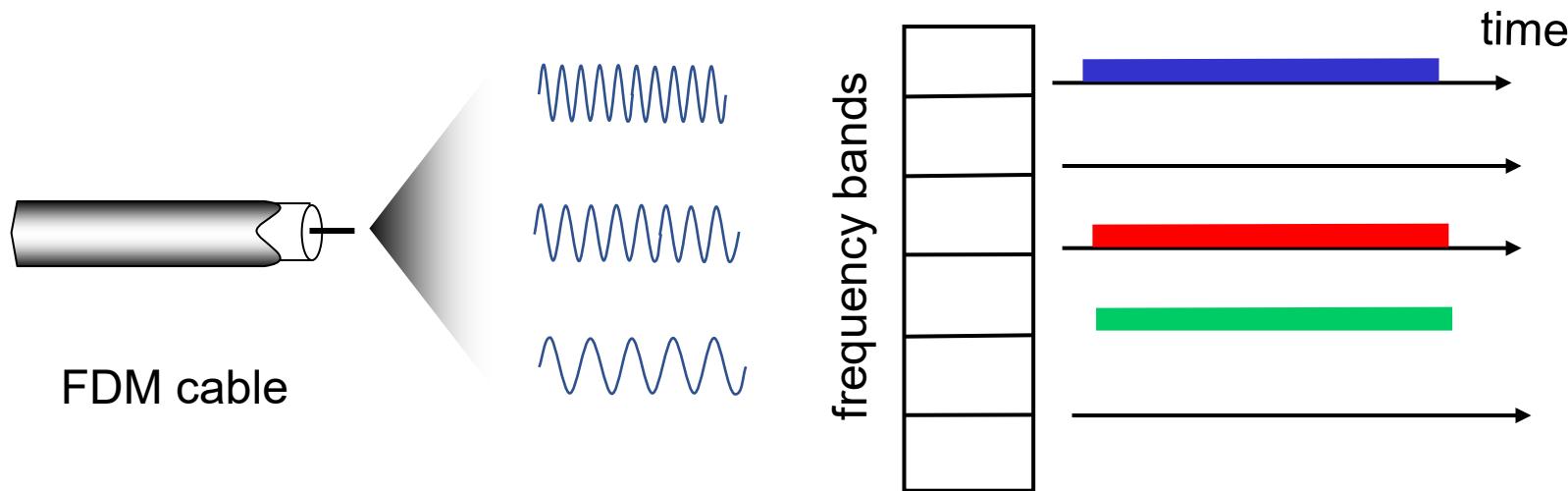
- access to channel in “rounds”
- each station gets fixed length slot (length = packet transmission time) in each round
- unused slots go idle
- example: 6-station LAN, 1,3,4 have packets to send, slots 2,5,6 idle



# Channel partitioning MAC protocols: FDMA

## FDMA: frequency division multiple access

- channel spectrum divided into frequency bands
- each station assigned fixed frequency band
- unused transmission time in frequency bands go idle
- example: 6-station LAN, 1,3,4 have packet to send, frequency bands 2,5,6 idle



# Random access protocols

- when node has packet to send
  - transmit at full channel data rate  $R$ .
  - no *a priori* coordination among nodes
- two or more transmitting nodes: “collision”
- **random access MAC protocol** specifies:
  - how to detect collisions
  - how to recover from collisions (e.g., via delayed retransmissions)
- examples of random access MAC protocols:
  - ALOHA, slotted ALOHA
  - CSMA, CSMA/CD, CSMA/CA

# Slotted ALOHA

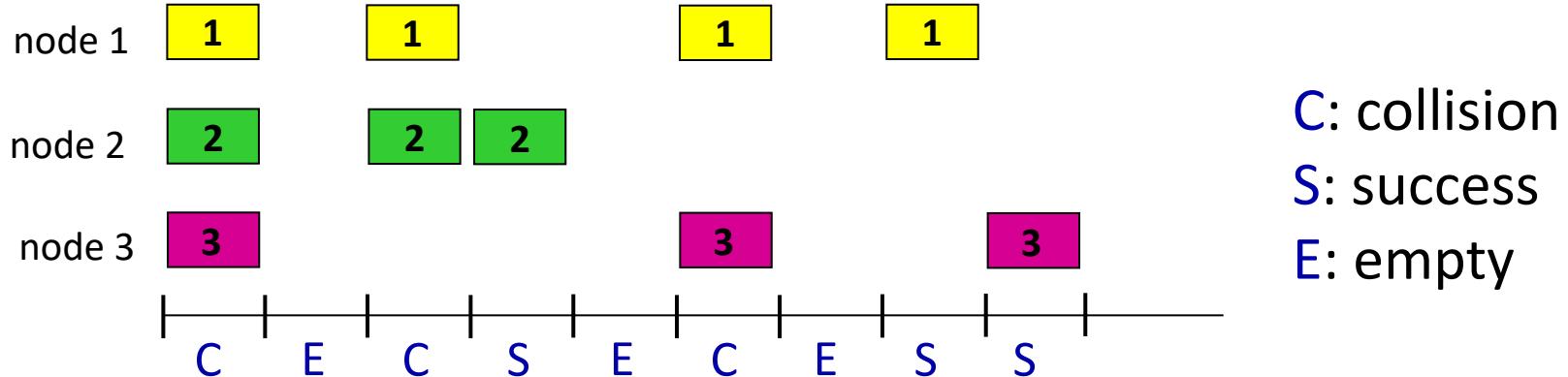
## assumptions:

- all frames same size
- time divided into equal size slots (time to transmit 1 frame)
- nodes start to transmit only slot beginning
- nodes are synchronized
- if 2 or more nodes transmit in slot, all nodes detect collision

## operation:

- when node obtains fresh frame, transmits in next slot
  - *if no collision*: node can send new frame in next slot
  - *if collision*: node retransmits frame in each subsequent slot with probability  $p$  until success

# Slotted ALOHA



## Pros:

- single active node can continuously transmit at full rate of channel
- highly decentralized: only slots in nodes need to be in sync
- simple

## Cons:

- collisions, wasting slots
- idle slots
- nodes may be able to detect collision in less than time to transmit packet
- clock synchronization

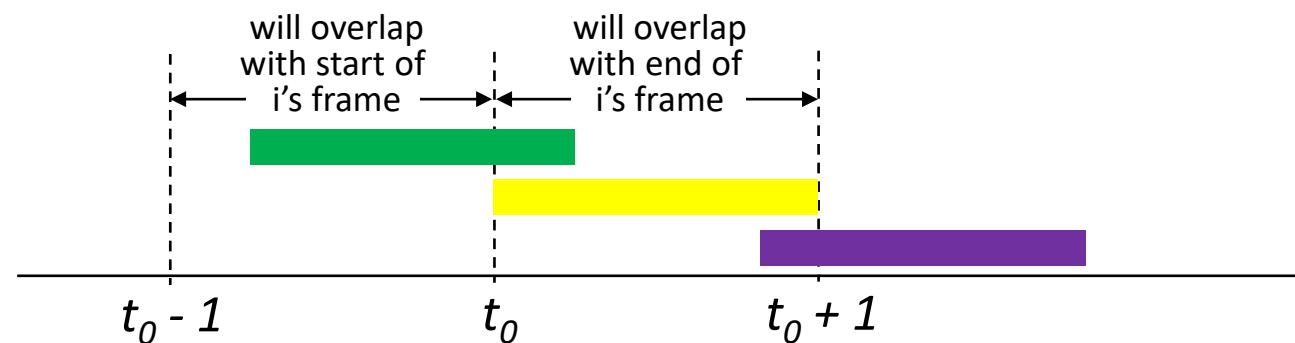
# Slotted ALOHA: efficiency

**efficiency:** long-run fraction of successful slots (many nodes, all with many frames to send)

- *suppose:*  $N$  nodes with many frames to send, each transmits in slot with probability  $p$ 
  - prob that given node has success in a slot =  $p(1-p)^{N-1}$
  - prob that *any* node has a success =  $Np(1-p)^{N-1}$
  - max efficiency: find  $p^*$  that maximizes  $Np(1-p)^{N-1}$
  - for many nodes, take limit of  $Np^*(1-p^*)^{N-1}$  as  $N$  goes to infinity, gives:  
*max efficiency =  $1/e = .37$*
- *at best:* channel used for useful transmissions 37% of time!

# Pure ALOHA

- unslotted Aloha: simpler, no synchronization
  - when frame first arrives: transmit immediately
- collision probability increases with no synchronization:
  - frame sent at  $t_0$  collides with other frames sent in  $[t_0-1, t_0+1]$



- pure Aloha efficiency: 18% !



# Computer Networks

Lecturer: ZHANG Ying  
Fall semester 2022

# FINAL EXAM

- Time: November 17<sup>th</sup>, Exam time will be 16:00 – 18:00 (Next Thursday), **all students should enter platform at 15:30 for identity check with your student card and passport.**
- Platform: Tencent Meeting (Meeting code will be provided 35 min before)
- Closed-book paper-based exam.
- Setting requirements:
  - Camera-based live streaming. The camera should cover NOT ONLY the face, BUT ALSO the desk and the computer screen in a light- sufficient environment.
  - Mic should be OPEN all the time.
  - Ask question ONLY with mic
  - Strictly no other communication
- The detailed policy will be shared for everyone soon and we will have an environment check **before exam next week.** (Tentative time: 15<sup>th</sup> Nov, 16:00)

# CSMA (carrier sense multiple access)



humans at a cocktail party  
(shared air, acoustical)

- *Listen before speaking* - **carrier sensing**
- *Stop talking if someone else begins talking* - **collision detection**

# CSMA (carrier sense multiple access)

simple **CSMA**: listen before transmit:

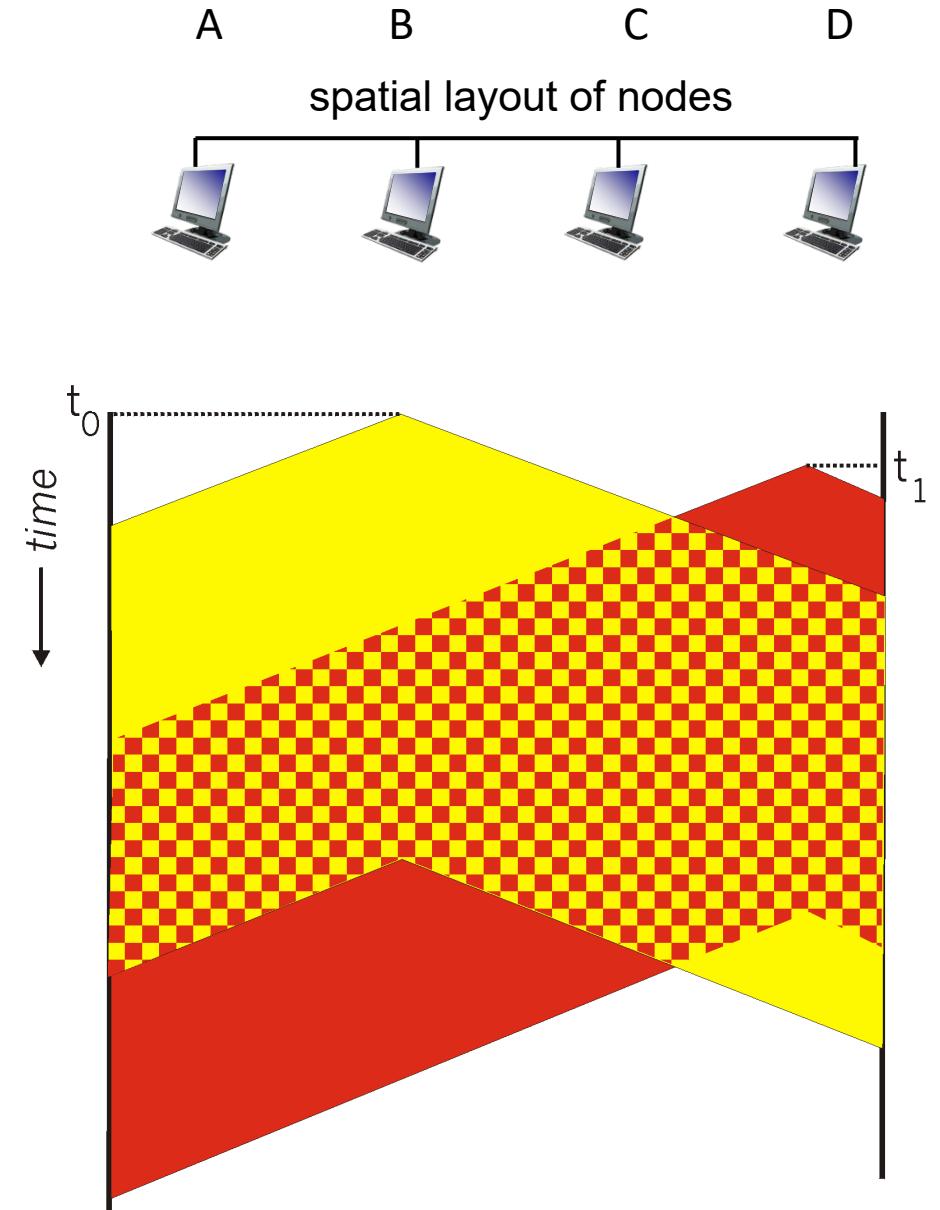
- if channel sensed idle: transmit entire frame
- if channel sensed busy: defer transmission
- human analogy: don't interrupt others!

**CSMA/CD**: CSMA with *collision detection*

- collisions *detected* within short time
- colliding transmissions aborted, reducing channel wastage
- collision detection easy in wired, difficult with wireless
- human analogy: the polite conversationalist

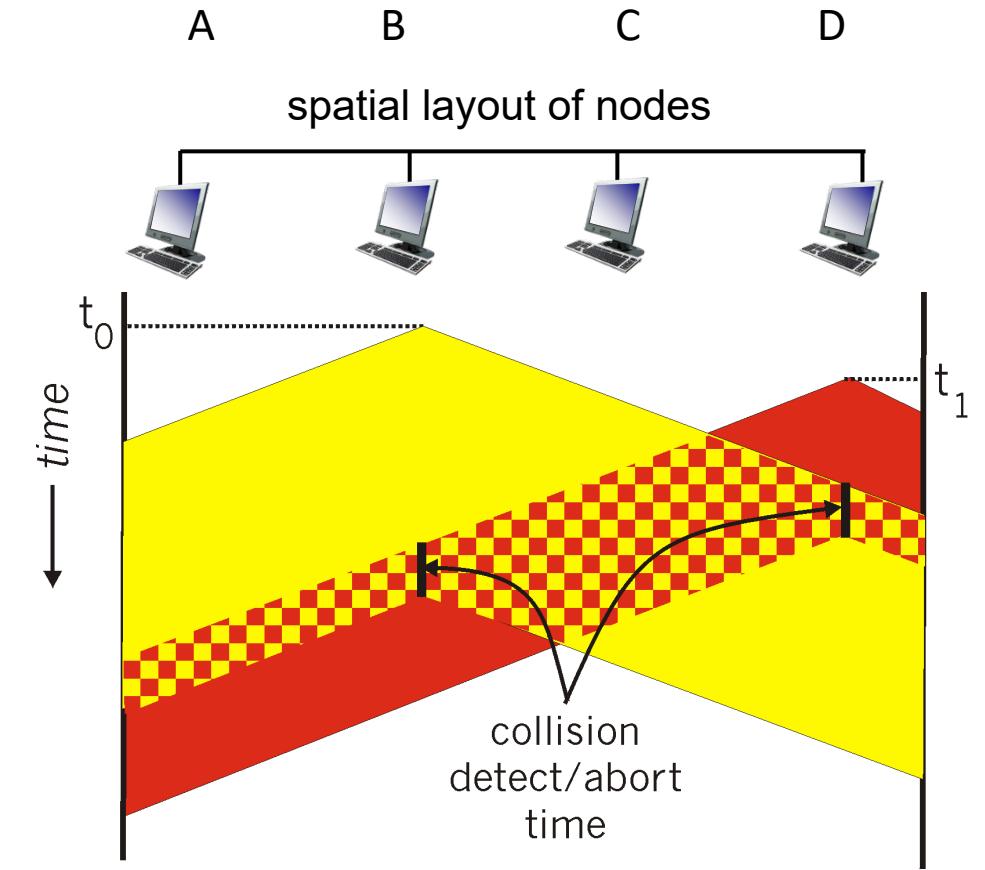
# CSMA: collisions

- collisions *can* still occur with carrier sensing:
  - propagation delay means two nodes may not hear each other's just-started transmission
- collision: entire packet transmission time wasted
  - distance & propagation delay play role in determining collision probability



# CSMA/CD:

- CSMA/CS reduces the amount of time wasted in collisions
  - transmission aborted on collision detection



# CSMA/CD efficiency

- $T_{prop}$  = max prop delay between 2 nodes in LAN
- $t_{trans}$  = time to transmit max-size frame

$$efficiency = \frac{1}{1 + 5t_{prop}/t_{trans}}$$

- efficiency goes to 1
  - as  $t_{prop}$  goes to 0
  - as  $t_{trans}$  goes to infinity
- better performance than ALOHA: and simple, cheap, decentralized!

# “Taking turns” MAC protocols

## channel partitioning MAC protocols:

- share channel *efficiently* and *fairly* at high load
- inefficient at low load: delay in channel access,  $1/N$  bandwidth allocated even if only 1 active node!

## random access MAC protocols

- efficient at low load: single node can fully utilize channel
- high load: collision overhead

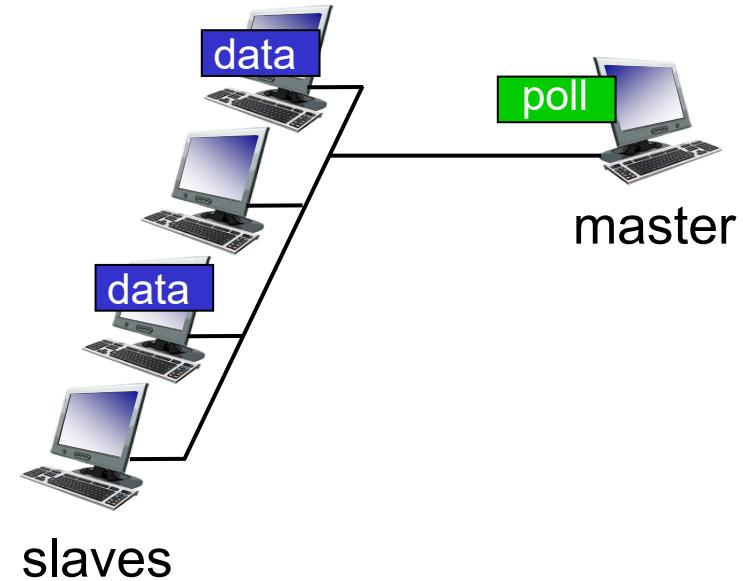
## “taking turns” protocols

- look for best of both worlds!

# “Taking turns” MAC protocols

## polling:

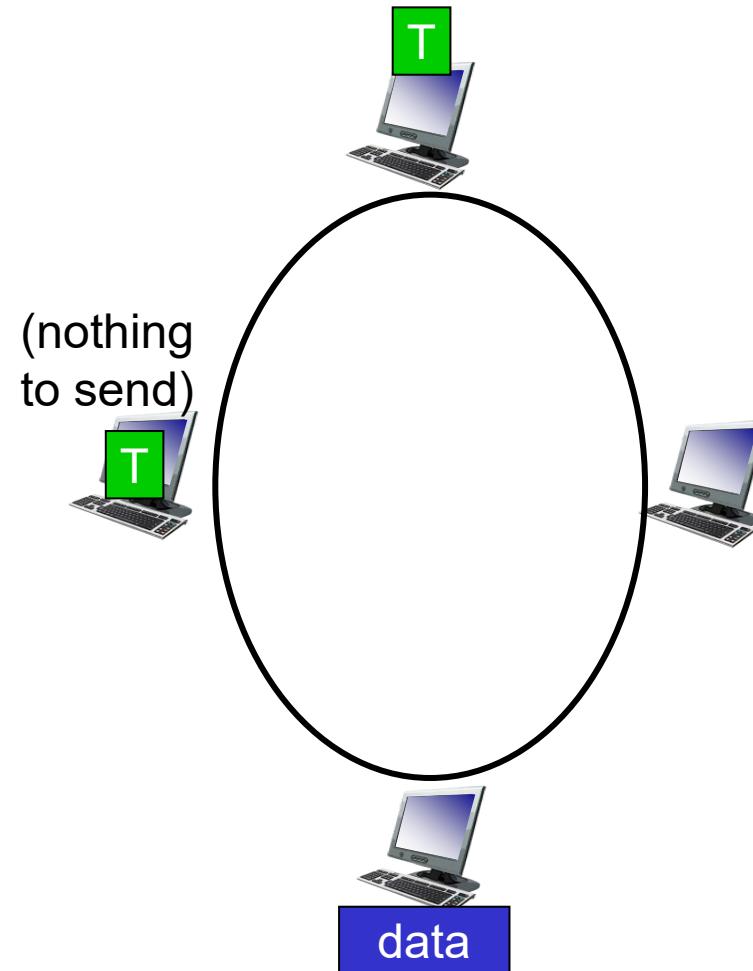
- master node “invites” other nodes to transmit in turn
- typically used with “dumb” devices
- concerns:
  - polling overhead
  - latency
  - single point of failure (master)



# “Taking turns” MAC protocols

## token passing:

- control *token* passed from one node to next sequentially.
- token message
- concerns:
  - token overhead
  - latency
  - single point of failure (token)



# Summary of MAC protocols

- **channel partitioning**, by time, frequency or code
  - Time Division, Frequency Division
- **random access (dynamic)**,
  - ALOHA, S-ALOHA, CSMA, CSMA/CD
  - carrier sensing: easy in some technologies (wire), hard in others (wireless)
  - CSMA/CD used in Ethernet
  - CSMA/CA used in 802.11
- **taking turns**
  - polling from central site, token passing
  - Bluetooth, FDDI, token ring

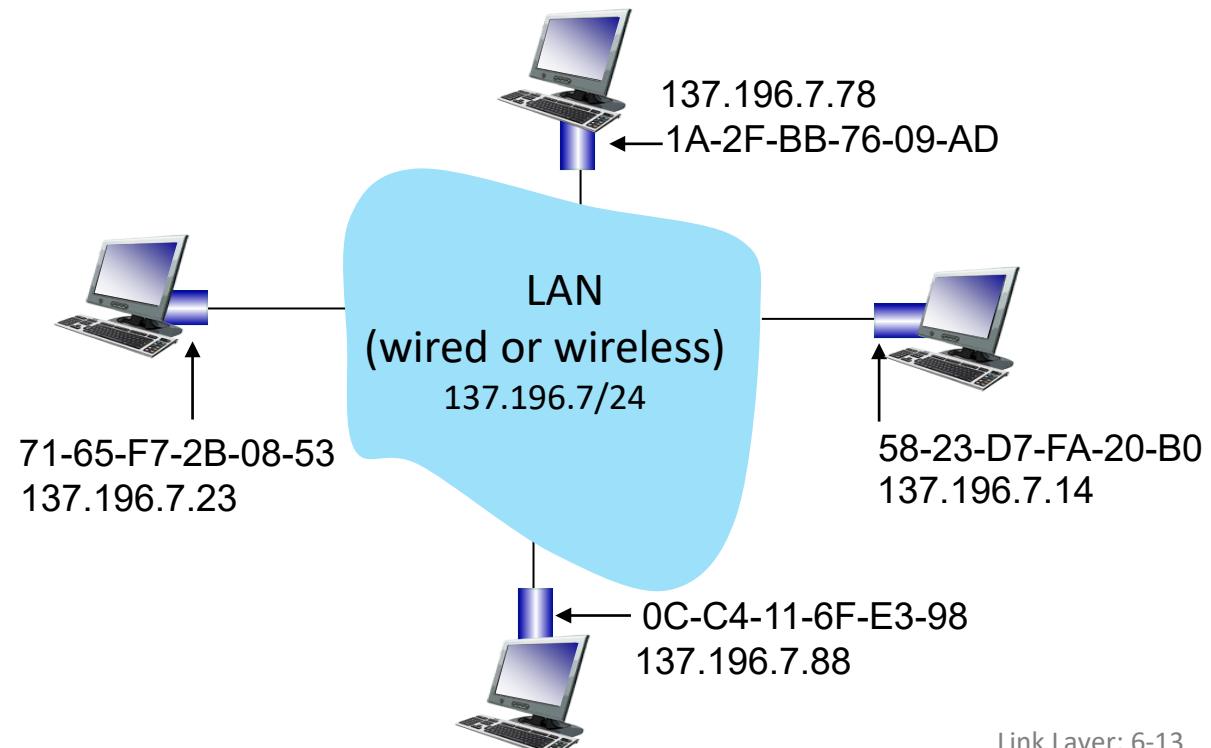
# Link layer, LANs: roadmap

- introduction
- error detection, correction
- multiple access protocols
- **LANs**
  - addressing, ARP



# MAC addresses

- MAC (or LAN or physical or Ethernet) address:
  - function: used “locally” to get frame from one interface to another physically-connected interface (same subnet, in IP-addressing sense)
  - 48-bit MAC address (for most LANs) burned in NIC ROM, also sometimes software settable
    - e.g.: 1A-2F-BB-76-09-AD
- each interface on LAN
  - has unique 48-bit **MAC** address
  - has a locally unique 32-bit IP address (as we've seen)

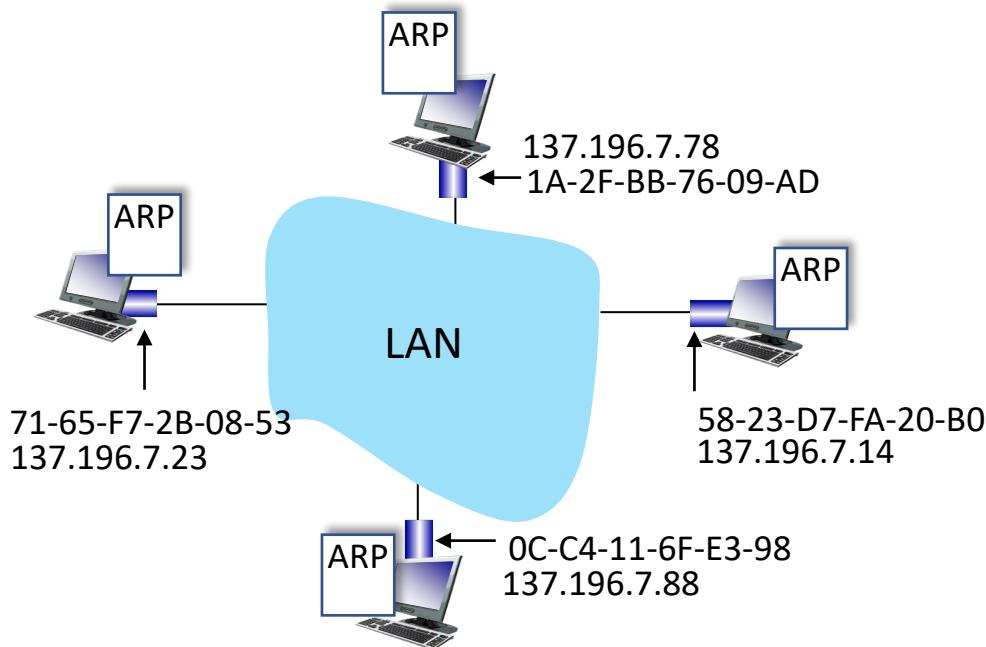


# MAC addresses

- MAC address allocation administered by IEEE
- manufacturer buys portion of MAC address space (to assure uniqueness)
- analogy:
  - MAC address: like Social Security Number
  - IP address: like postal address
- MAC flat address: portability
  - can move interface from one LAN to another
  - recall IP address *not* portable: depends on IP subnet to which node is attached

# ARP: address resolution protocol

*Question:* how to determine interface's MAC address, knowing its IP address?



**ARP table:** each IP node (host, router) on LAN has table

- IP/MAC address mappings for some LAN nodes:  
<IP address; MAC address; TTL>
- TTL (Time To Live): time after which address mapping will be forgotten (typically 20 min)

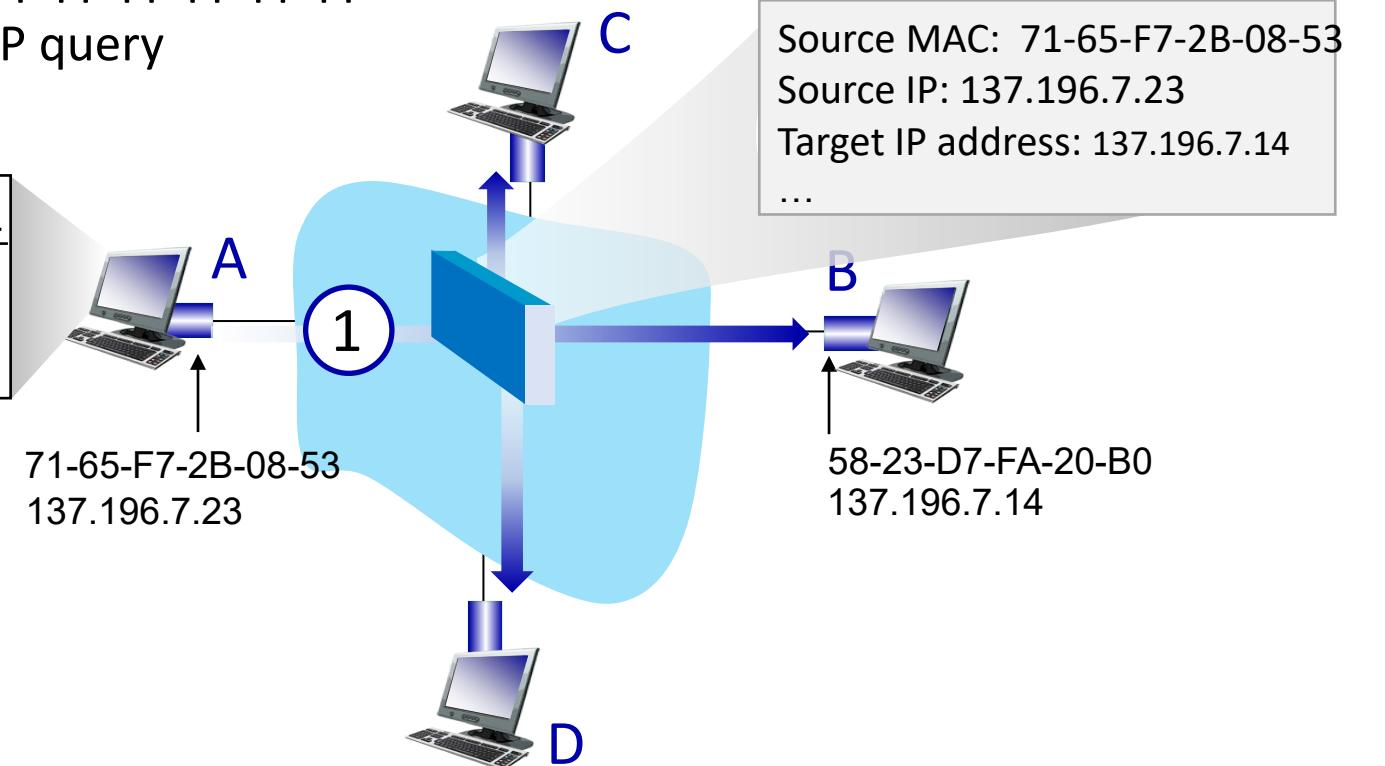
# ARP protocol in action

example: A wants to send datagram to B

- B's MAC address not in A's ARP table, so A uses ARP to find B's MAC address

- 1 A broadcasts ARP query, containing B's IP addr
- destination MAC address = FF-FF-FF-FF-FF-FF
  - all nodes on LAN receive ARP query

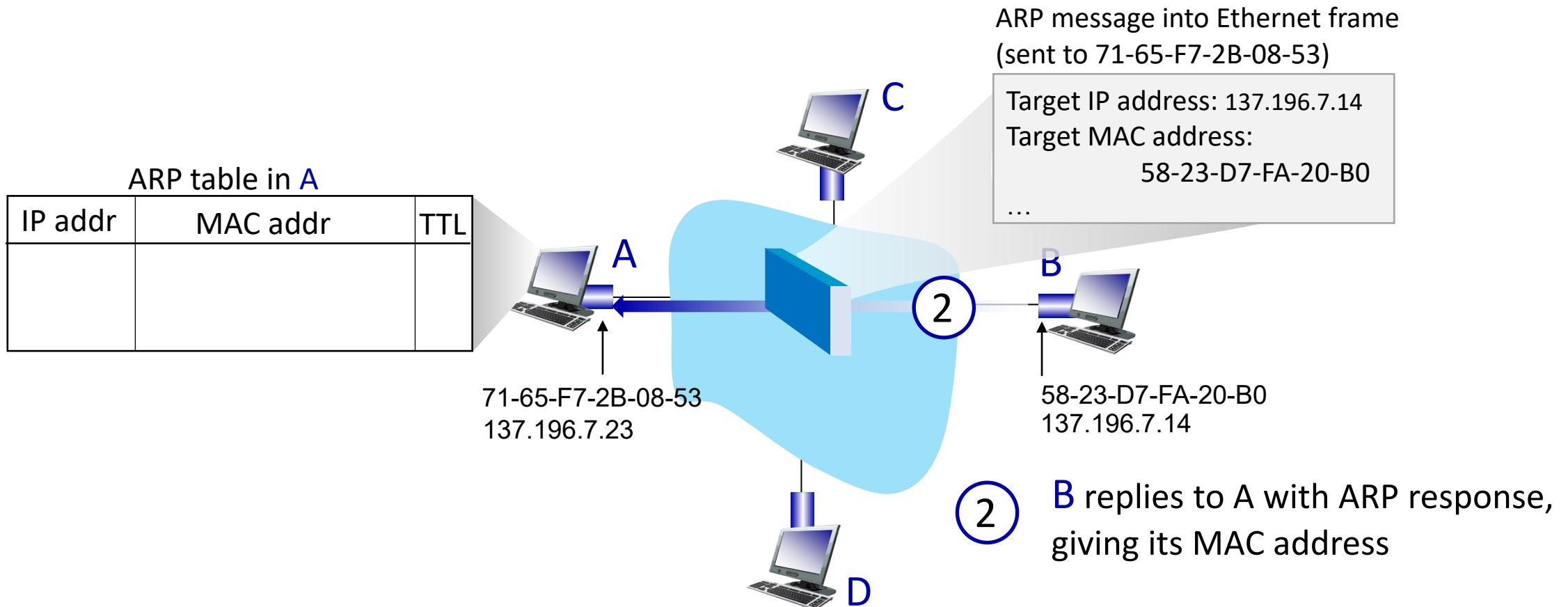
| ARP table in A |          |     |
|----------------|----------|-----|
| IP addr        | MAC addr | TTL |
|                |          |     |



# ARP protocol in action

example: A wants to send datagram to B

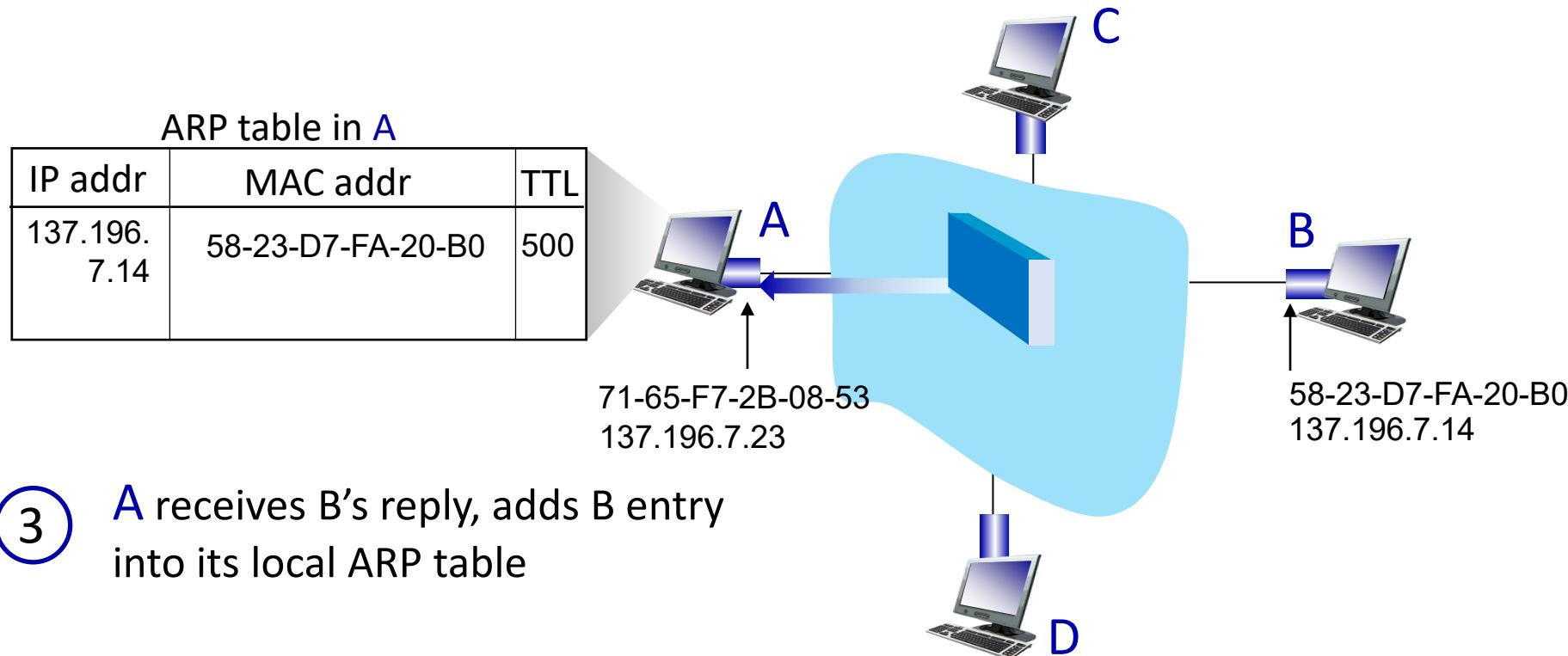
- B's MAC address not in A's ARP table, so A uses ARP to find B's MAC address



# ARP protocol in action

example: A wants to send datagram to B

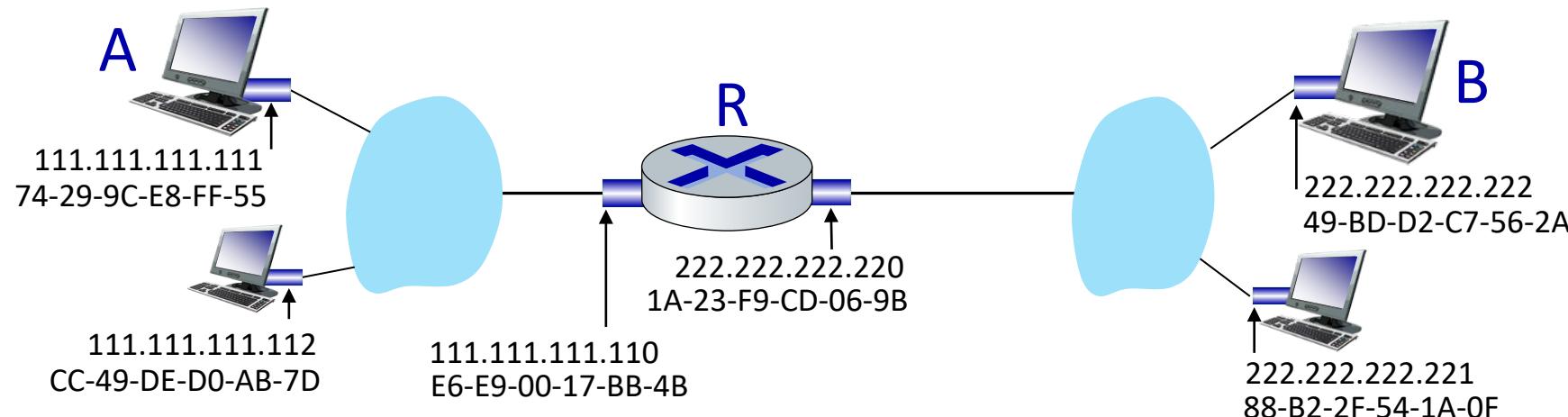
- B's MAC address not in A's ARP table, so A uses ARP to find B's MAC address



# Routing to another subnet: addressing

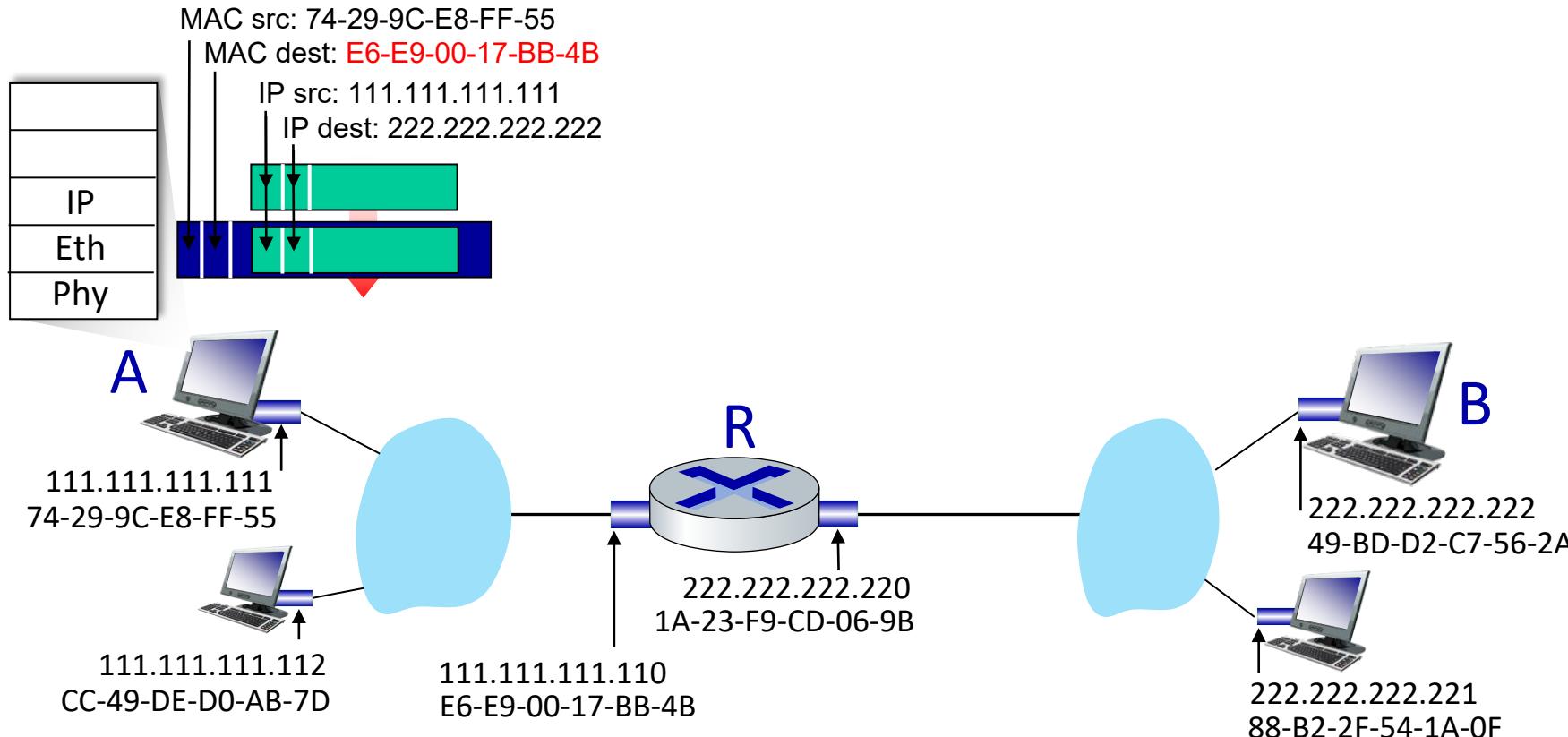
walkthrough: sending a datagram from *A* to *B* via *R*

- focus on addressing – at IP (datagram) and MAC layer (frame) levels
- assume that:
  - A knows B's IP address
  - A knows IP address of first hop router, R
  - A knows R's MAC address



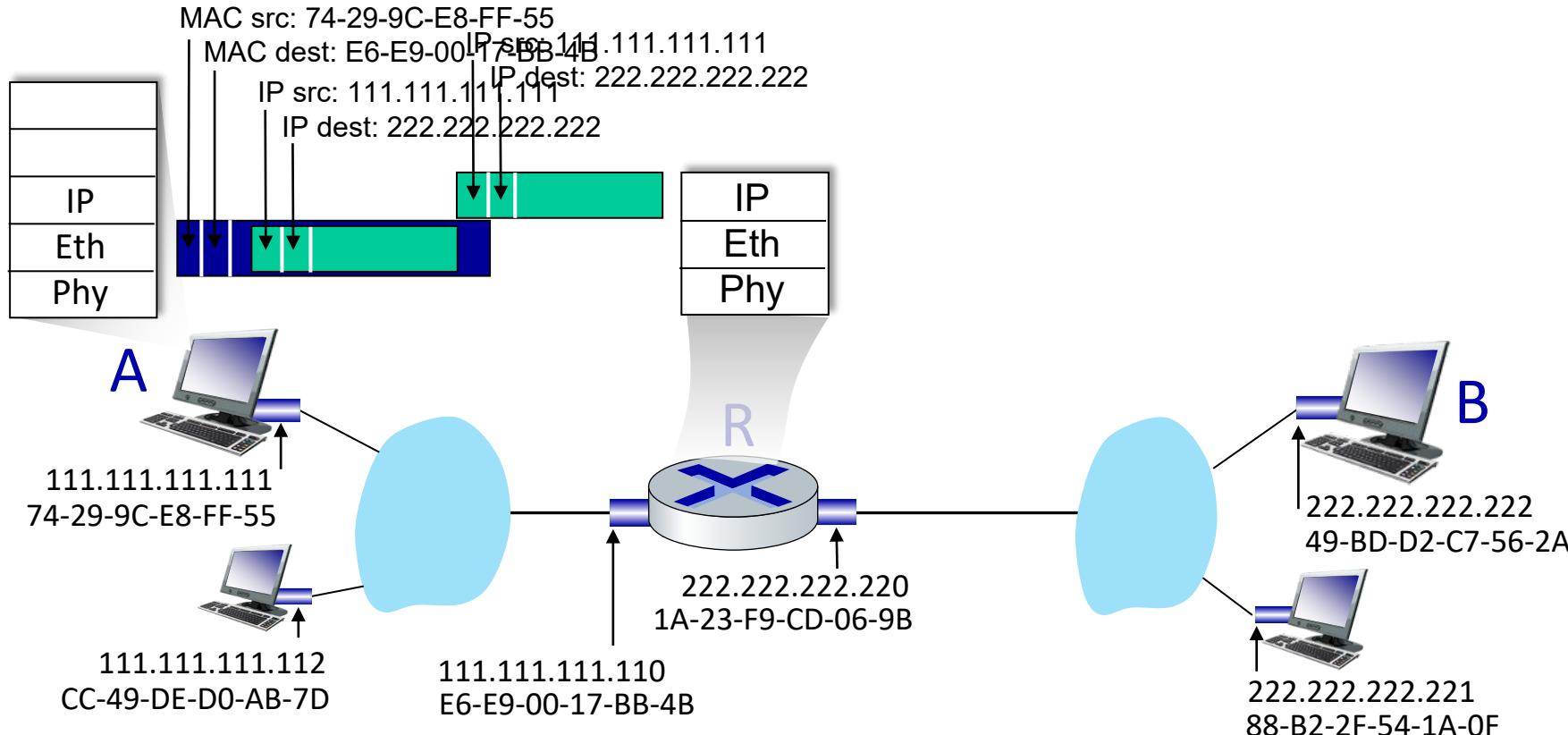
# Routing to another subnet: addressing

- A creates IP datagram with IP source A, destination B
- A creates link-layer frame containing A-to-B IP datagram
  - R's MAC address is frame's destination



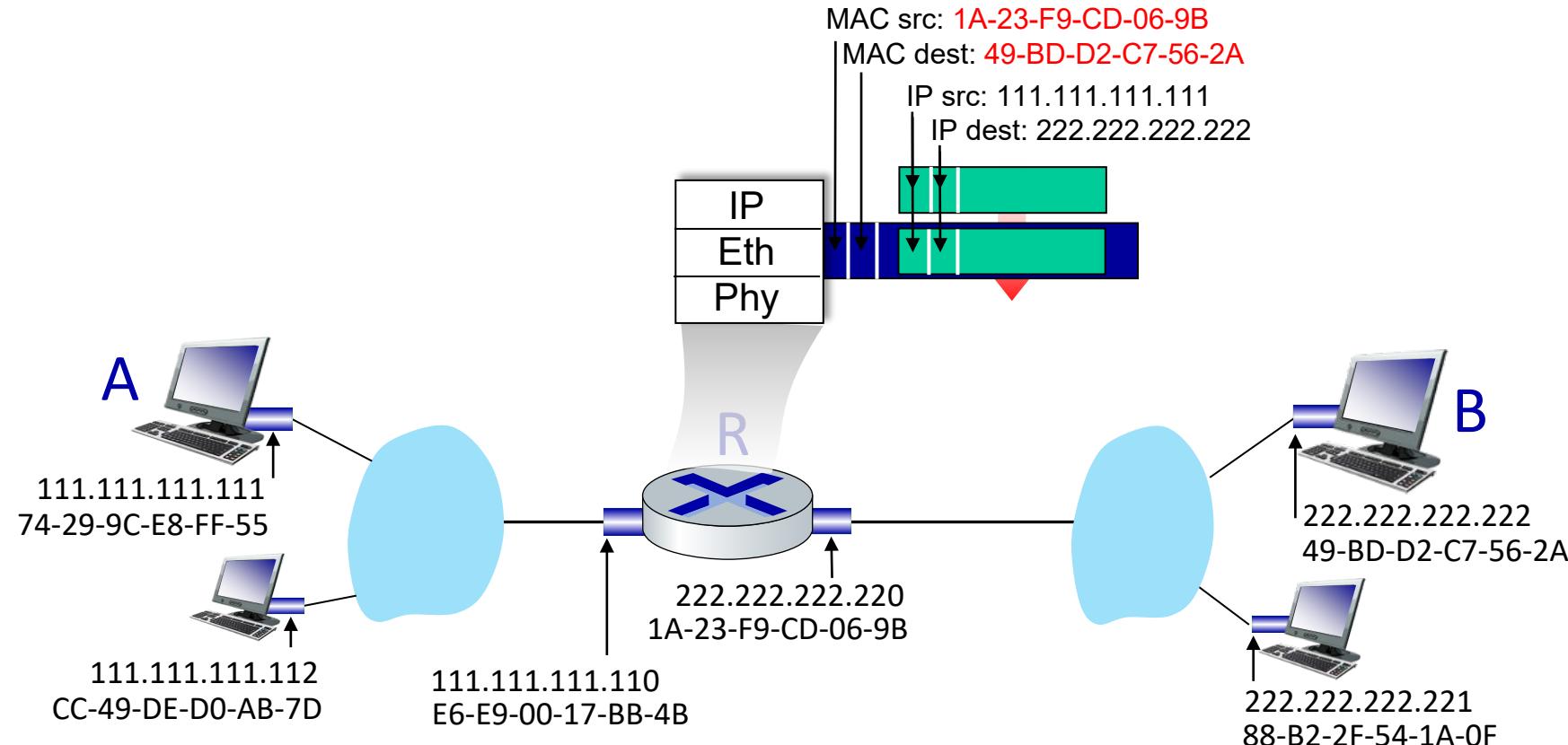
# Routing to another subnet: addressing

- frame sent from A to R
- frame received at R, datagram removed, passed up to IP



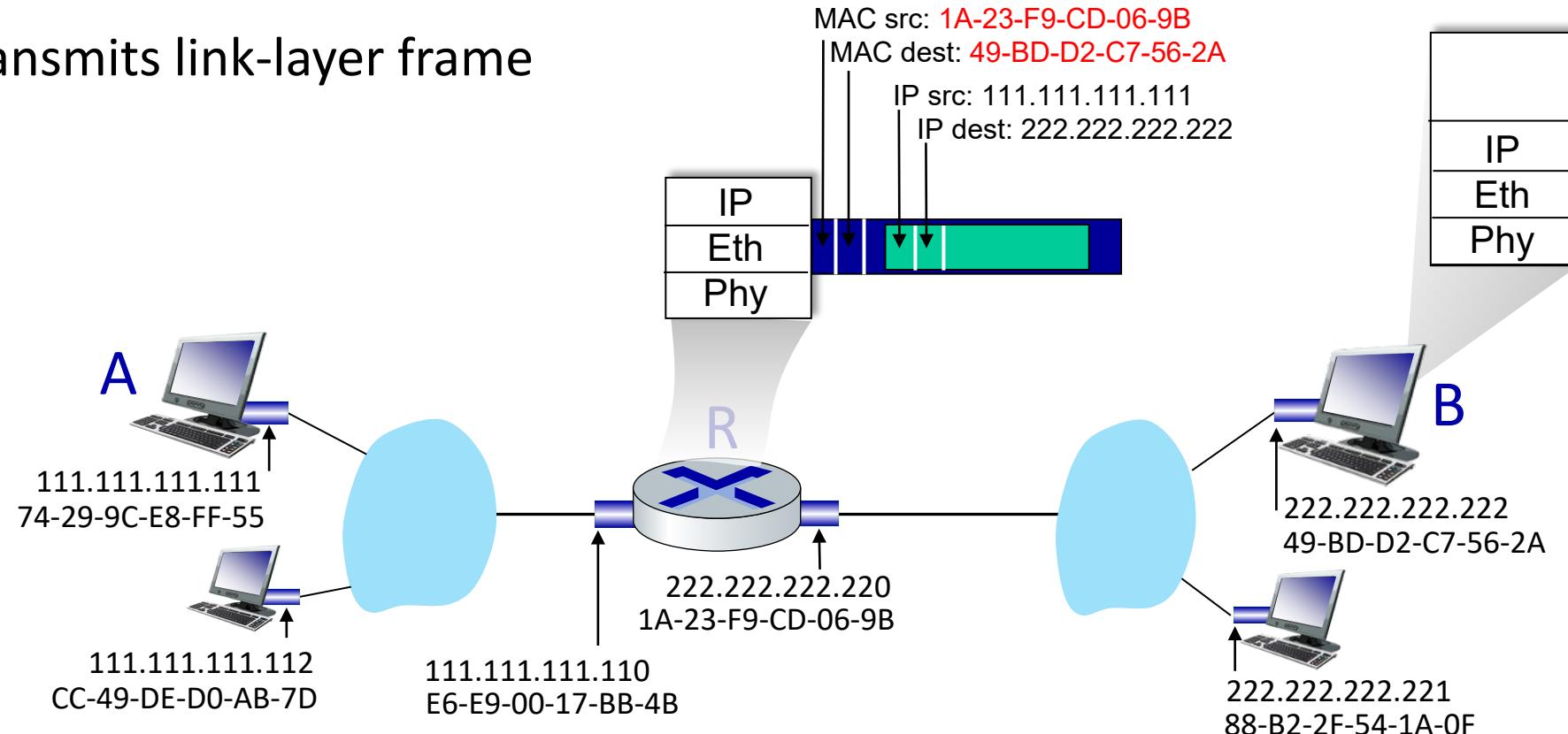
# Routing to another subnet: addressing

- R determines outgoing interface, passes datagram with IP source A, destination B to link layer
- R creates link-layer frame containing A-to-B IP datagram. Frame destination address: B's MAC address



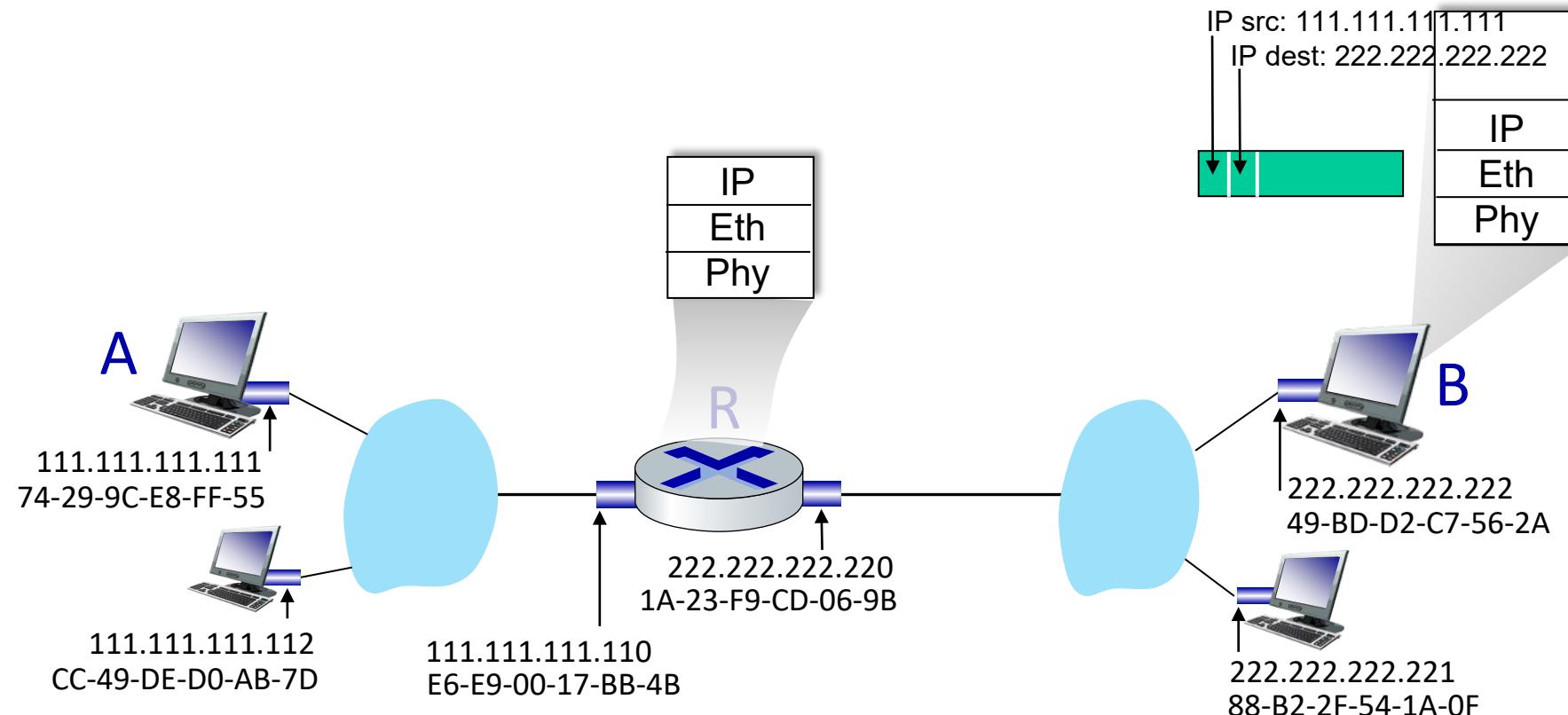
# Routing to another subnet: addressing

- R determines outgoing interface, passes datagram with IP source A, destination B to link layer
- R creates link-layer frame containing A-to-B IP datagram. Frame destination address: B's MAC address
- transmits link-layer frame



# Routing to another subnet: addressing

- B receives frame, extracts IP datagram destination B
- B passes datagram up protocol stack to IP



# Link layer, LANs: roadmap

- introduction
- error detection, correction
- multiple access protocols
- **LANs**
  - addressing, ARP
  - **Ethernet**

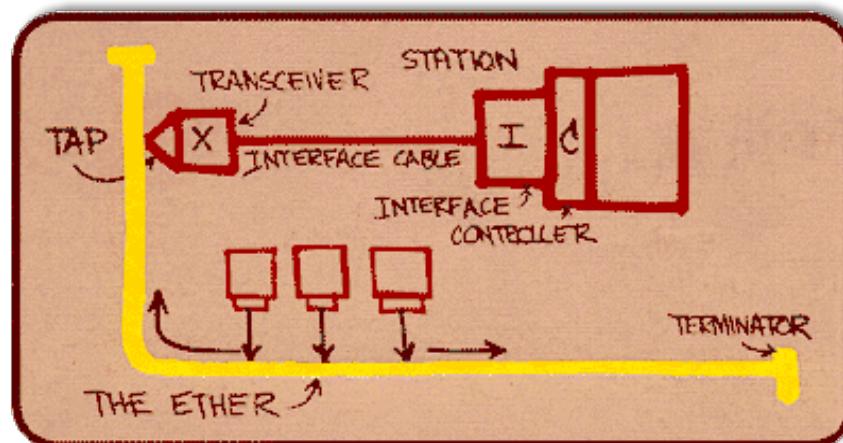


- a day in the life of a web request

# Ethernet

“dominant” wired LAN technology:

- first widely used LAN technology
- simpler, cheap
- kept up with speed race: 10 Mbps – 400 Gbps
- single chip, multiple speeds (e.g., Broadcom BCM5761)

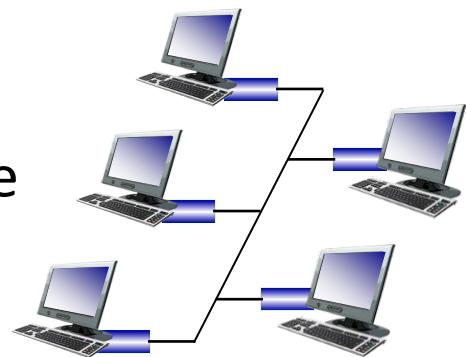


*Metcalfe's Ethernet sketch*

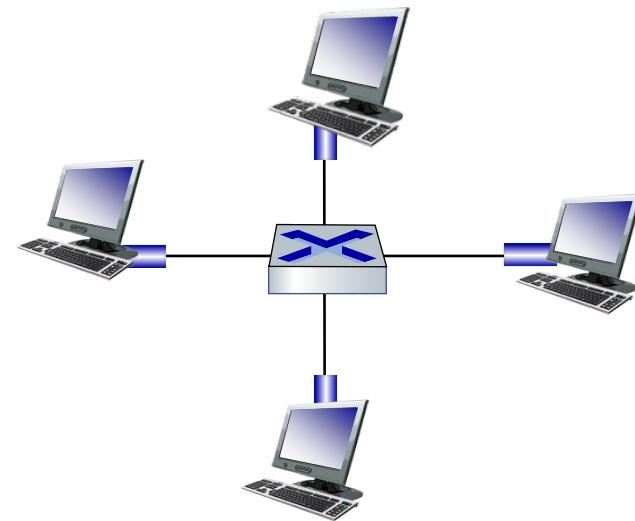
# Ethernet: physical topology

- **bus:** popular through mid 90s
  - all nodes in same collision domain (can collide with each other)
- **switched:** prevails today
  - active link-layer 2 *switch* in center
  - each “spoke” runs a (separate) Ethernet protocol (nodes do not collide with each other)

bus: coaxial cable



switched



# Ethernet frame structure

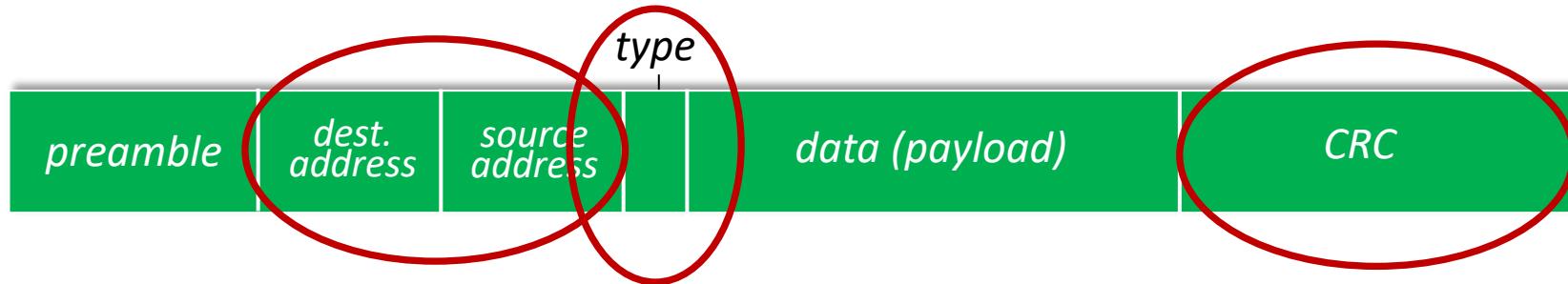
sending interface encapsulates IP datagram (or other network layer protocol packet) in **Ethernet frame**



*preamble*:

- used to synchronize receiver, sender clock rates
- 7 bytes of 10101010 followed by one byte of 10101011

# Ethernet frame structure (more)



- **addresses**: 6 byte source, destination MAC addresses
  - if adapter receives frame with matching destination address, or with broadcast address (e.g., ARP packet), it passes data in frame to network layer protocol
  - otherwise, adapter discards frame
- **type**: indicates higher layer protocol
  - mostly IP but others possible, e.g., Novell IPX, AppleTalk
  - used to demultiplex up at receiver
- **CRC**: cyclic redundancy check at receiver
  - error detected: frame is dropped

# Ethernet: unreliable, connectionless

- **connectionless:** no handshaking between sending and receiving NICs
- **unreliable:** receiving NIC doesn't send ACKs or NAKs to sending NIC
  - data in dropped frames recovered only if initial sender uses higher layer rdt (e.g., TCP), otherwise dropped data lost
- Ethernet's MAC protocol: unslotted **CSMA/CD with binary backoff**

# Ethernet CSMA/CD algorithm

1. NIC receives datagram from network layer, creates frame
2. If NIC senses channel:
  - if **idle**: start frame transmission.
  - if **busy**: wait until channel idle, then transmit
3. If NIC transmits entire frame without collision, NIC is done with frame !
4. If NIC detects another transmission while sending: abort, send jam signal
5. After aborting, NIC enters *binary (exponential) backoff*:
  - after  $m$ th collision, NIC chooses  $K$  at random from  $\{0,1,2, \dots, 2^m-1\}$ . NIC waits  $K \cdot 512$  bit times, returns to Step 2
  - more collisions: longer backoff interval

# Ethernet CSMA/CD algorithm

*binary (exponential) backoff:*

- after  $m$ th collision, NIC chooses  $K$  at random from  $\{0,1,2, \dots, 2^m-1\}$ . NIC waits  $K \cdot 512$  bit times, returns to Step 2
- more collisions: longer backoff interval

Example:

- 1<sup>st</sup> collision,  $m = 1$ , range =  $\{0,1\}$
- 2<sup>nd</sup> collision,  $m = 2$ , range =  $\{0,1,2,3\}$
- 3<sup>rd</sup> collision,  $m = 3$ , range =  $\{0,1,2,3,4,5,6,7\}$
- ..... Usually with a upper boundary
- E.g., range =  $\{0,1,2,3,\dots, 1023\}$ , exponential backoff

# 802.3 Ethernet standards: link & physical layers

- *many* different Ethernet standards
  - common MAC protocol and frame format
  - different speeds: 2 Mbps, 10 Mbps, 100 Mbps, 1Gbps, 10 Gbps, 40 Gbps
  - different physical layer media: fiber, cable

