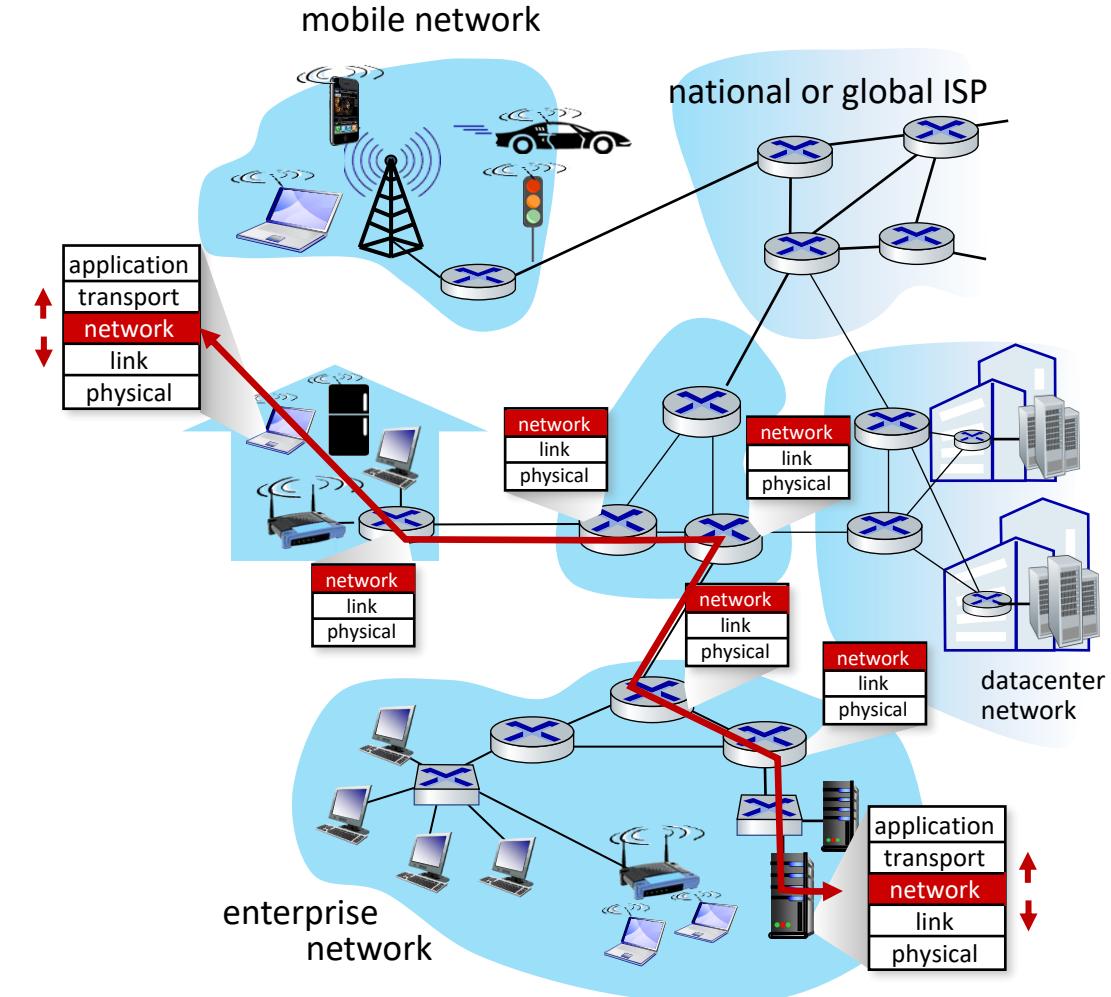


Network Layer: Part 1

Slides are adapted from the companion website of the book *Computer Networking: A Top-Down Approach* by Jim Kurose, Keith Ross

Network-layer services and protocols

- transport segment from sending to receiving host
 - sender: encapsulates segments into datagrams, passes to link layer
 - receiver: delivers segments to **transport layer protocol**
- network layer protocols in *every Internet device*: hosts, routers
- routers:
 - examines header fields in all IP datagrams passing through it
 - moves datagrams from input ports to output ports to transfer datagrams along end-end path



Two key network-layer functions

network-layer functions:

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

analogy: taking a trip

- *forwarding*: process of getting through single interchange
- *routing*: process of planning trip from source to destination



forwarding

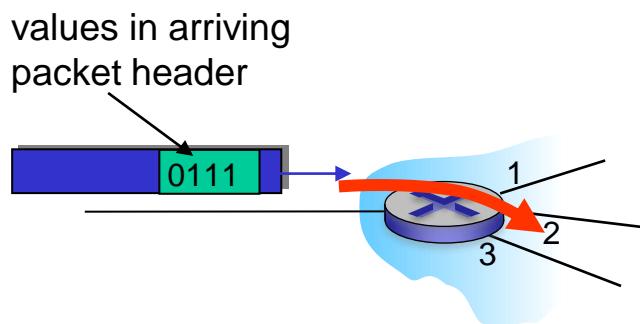


routing

Network layer: data plane, control plane

Data plane:

- *local*, per-router function
- determines how datagram arriving on router input port is forwarded to router output port

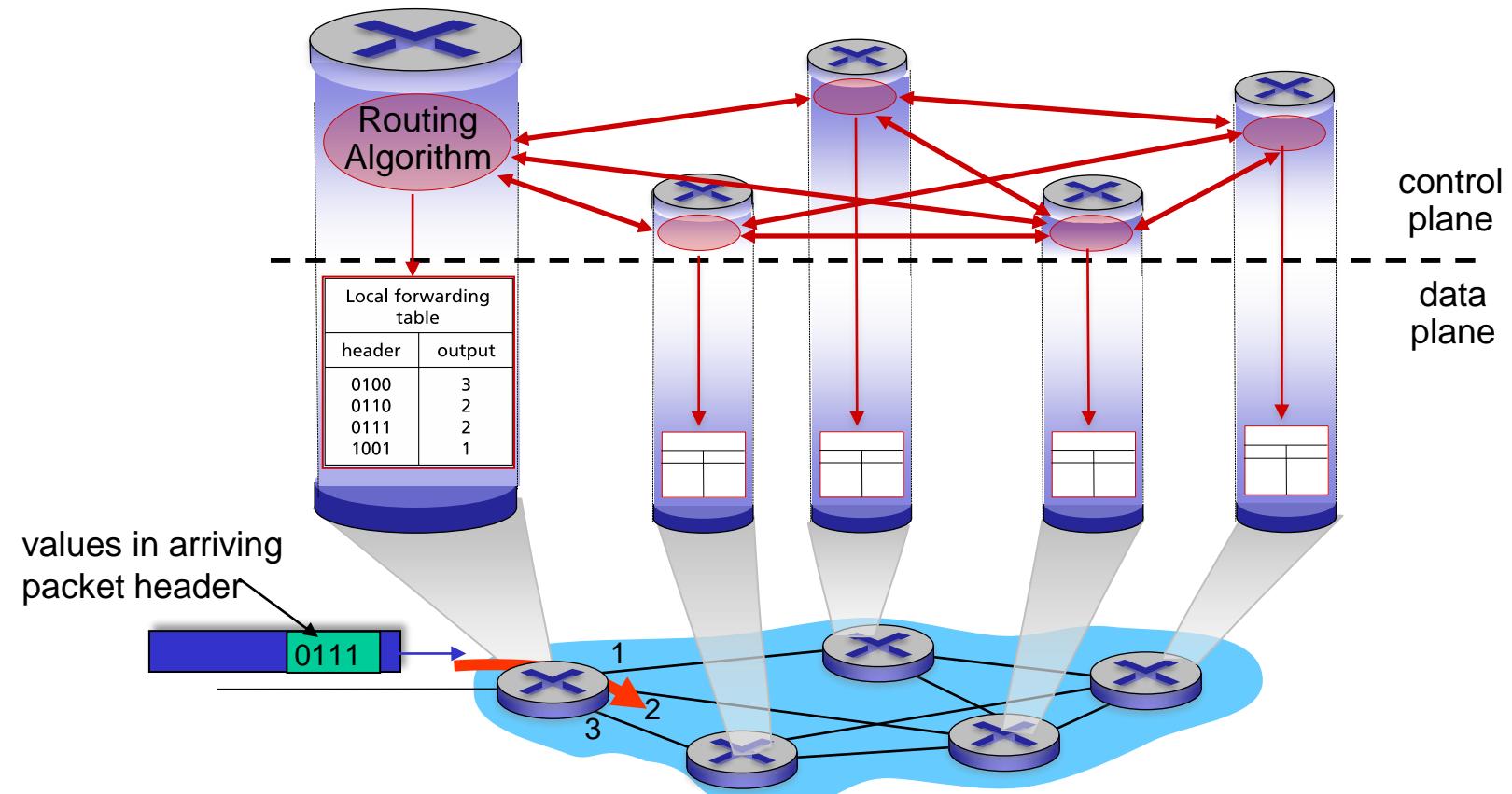


Control plane

- *network-wide* logic
- determines how datagram is routed among routers along end-end path from source host to destination host
- two control-plane approaches:
 - *traditional routing algorithms*: implemented in routers
 - *software-defined networking (SDN)*: implemented in (remote) servers

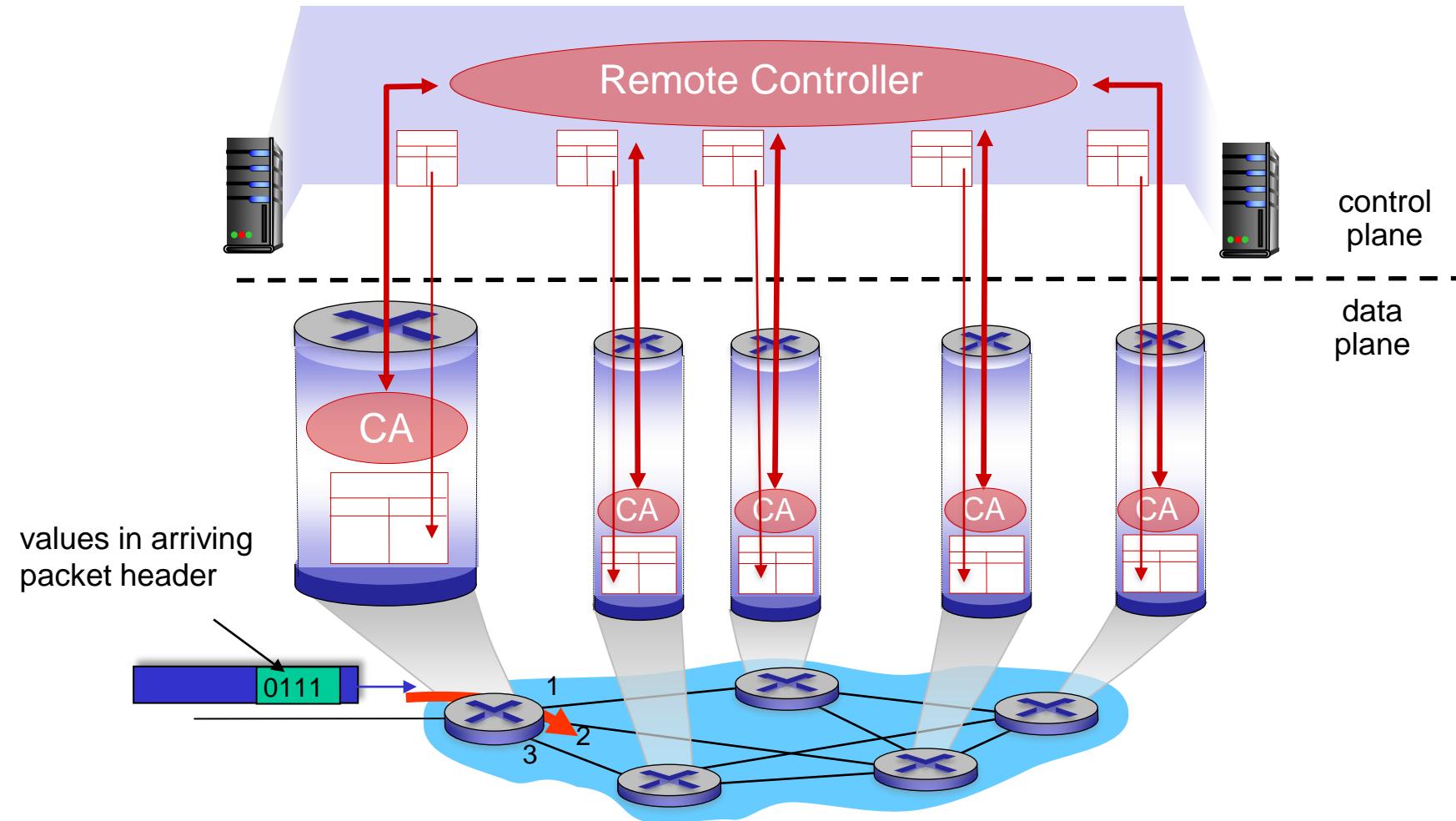
Per-router control plane

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

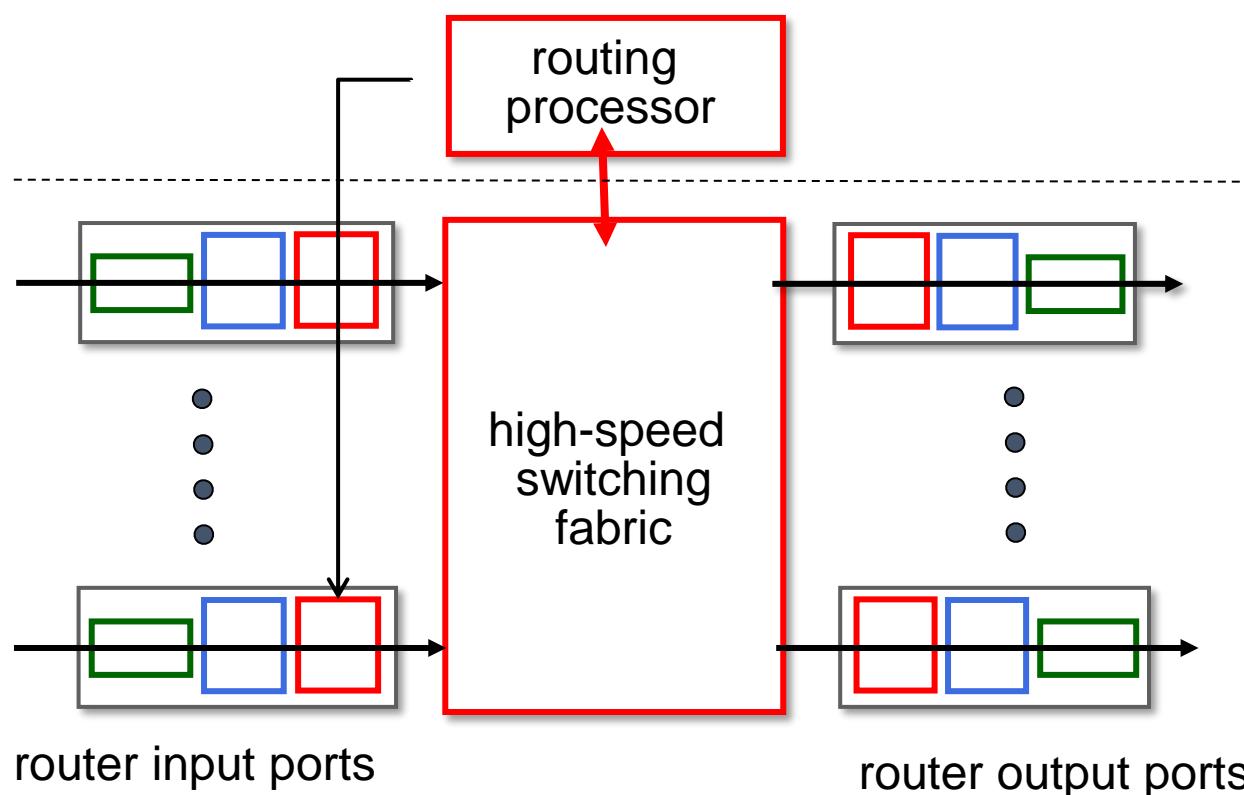


Network layer: roadmap

- 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
- Generalized Forwarding, SDN
 - Match+action
 - OpenFlow: match+action in action
- Middleboxes

Router architecture overview

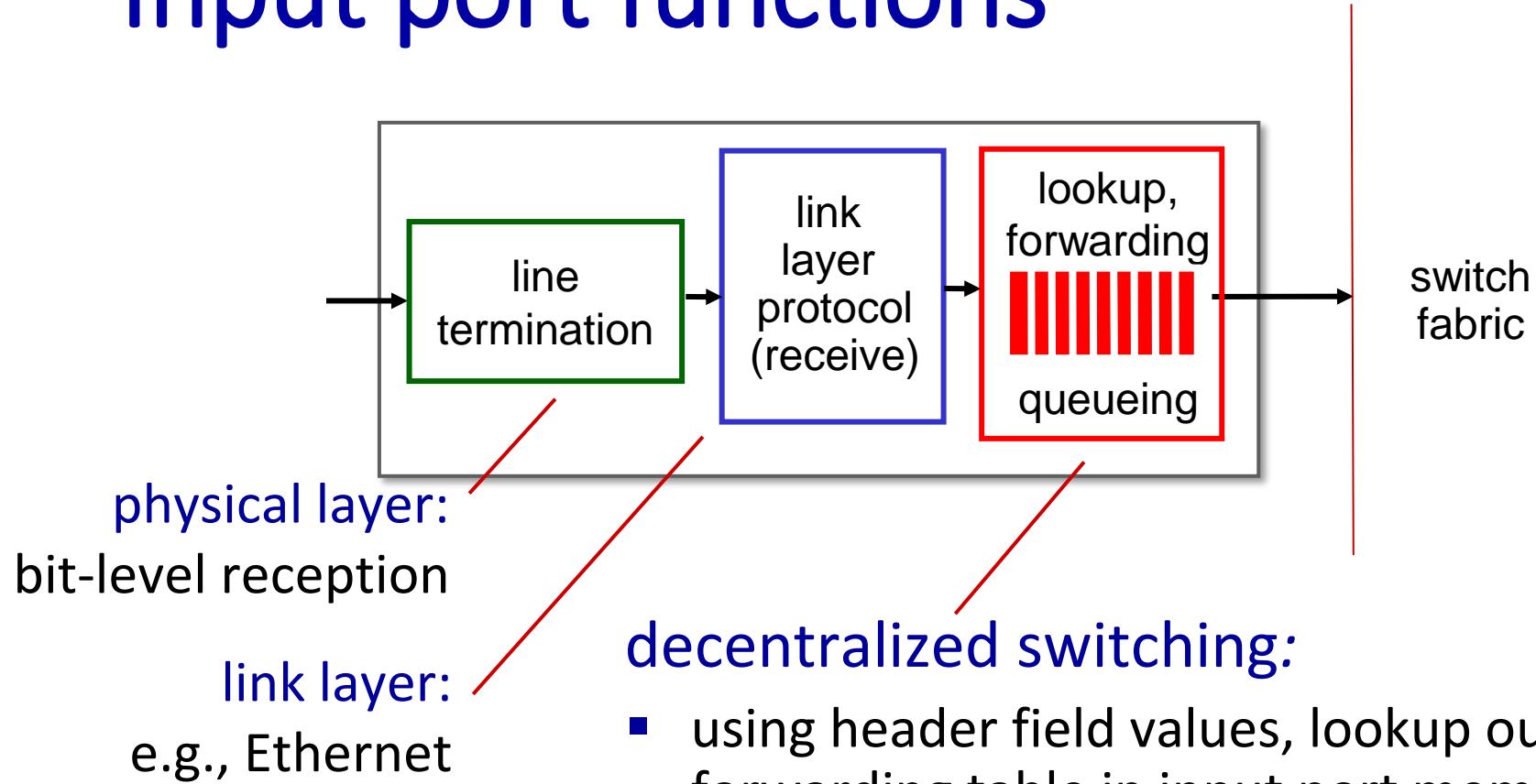
high-level view of generic router architecture:



routing, management control plane (software)
operates in millisecond time frame

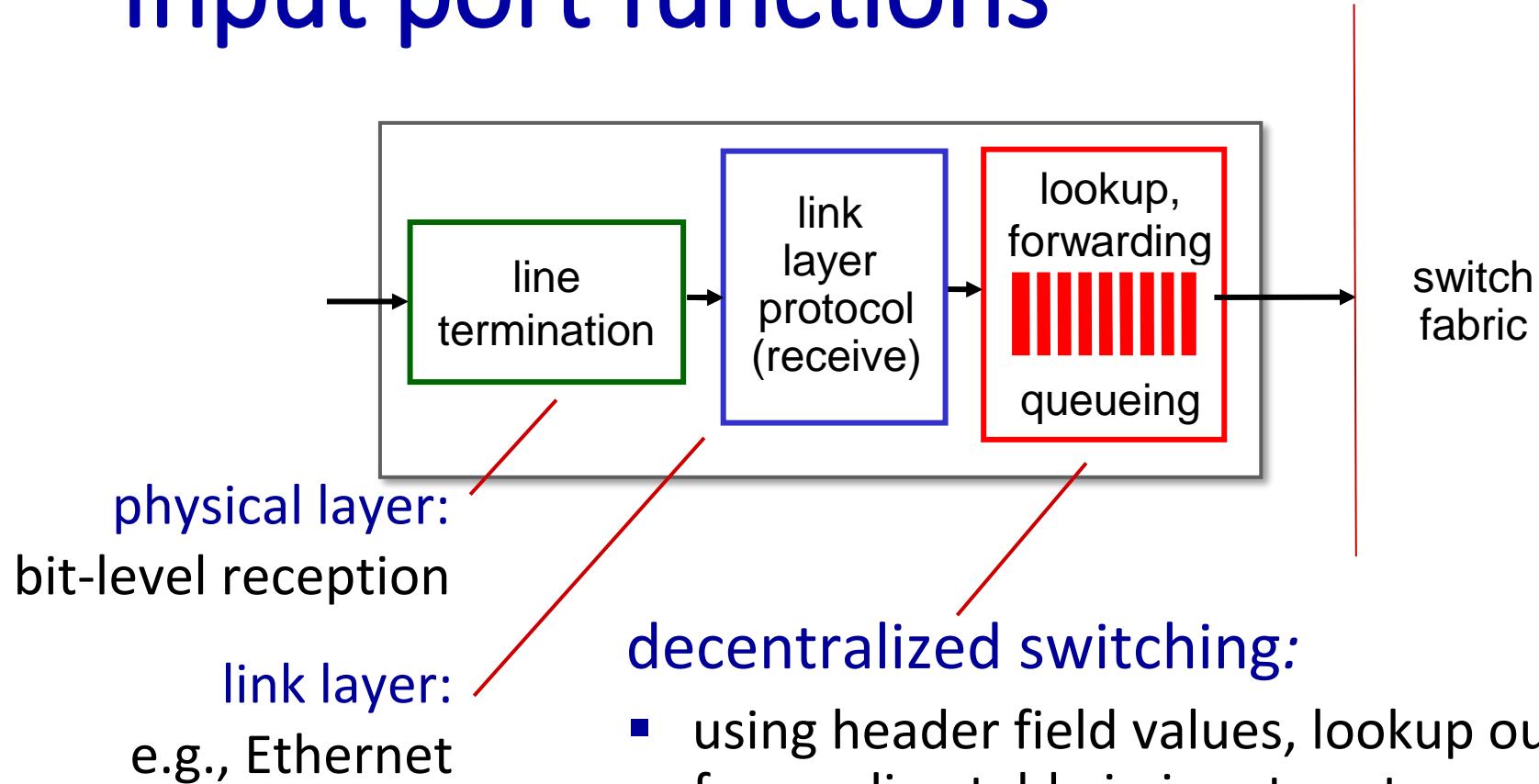
forwarding data plane (hardware)
operates in nanosecond timeframe

Input port functions



- using header field values, lookup output port using forwarding table in input port memory ("*match plus action*")
- goal: complete input port processing at 'line speed'
- **input port queuing:** if datagrams arrive faster than forwarding rate into switch fabric

Input port functions



decentralized switching:

- using header field values, lookup output port using forwarding table in input port memory ("*match plus action*")
- **destination-based forwarding**: forward based only on destination IP address (traditional)
- **generalized forwarding**: forward based on any set of header field values

Destination-based forwarding

<i>forwarding table</i>	
Destination Address Range	Link Interface
11001000 00010111 00010000 00000000 through	n
11001000 00010111 00010000 00000100 through	3
11001000 00010111 00010000 00000111	
11001000 00010111 00011000 11111111	
11001000 00010111 00011001 00000000 through	2
11001000 00010111 00011111 11111111	
otherwise	3

Q: but what happens if ranges don't divide up so nicely?

Longest prefix matching

longest prefix match

when looking for forwarding table entry for given destination address, use *longest* address prefix that matches destination address.

Destination Address Range	Link interface
11001000 00010111 00010*** *****	0
11001000 00010111 00011000 *****	1
11001000 00010111 00011*** *****	2
otherwise	3

examples:

- | | |
|-------------------------------------|------------------|
| 11001000 00010111 00010110 10100001 | which interface? |
| 11001000 00010111 00011000 10101010 | which interface? |

Longest prefix matching

longest prefix match

when looking for forwarding table entry for given destination address, use *longest* address prefix that matches destination address.

Destination Address Range	Link interface
11001000 00010111 00010*****	0
11001000 00010111 00011000 *****	1
11001000 1 00011*** *****	2
otherwise	3

examples:

11001000 00010111 00010110 10100001 which interface?

11001000 00010111 00011000 10101010 which interface?

Longest prefix matching

longest prefix match

when looking for forwarding table entry for given destination address, use *longest* address prefix that matches destination address.

Destination Address Range					Link interface
11001000	00010111	00010***	*****	*	0
11001000	00010111	00011000	*****	*	1
11001000	00010111	00011***	*****	*	2
otherwise					3

match!

examples:

11001000	00010111	00010110	10100001	which interface?
11001000	00010111	00011000	10101010	which interface?

Longest prefix matching

longest prefix match

when looking for forwarding table entry for given destination address, use *longest* address prefix that matches destination address.

Destination Address Range					Link interface
11001000	00010111	00010***	*****	*	0
11001000	00010111	00011000	*****	*	1
11001000	00010111	00011***	*****	*	2
otherwise					3

match!

examples:

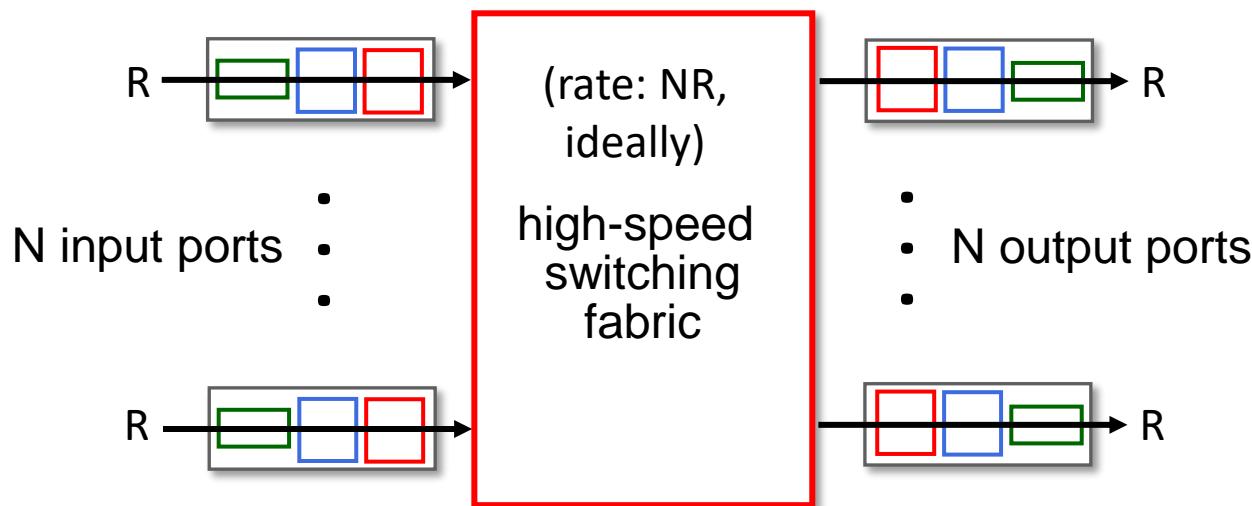
11001000	00010111	00010110	10100001	which interface?
11001000	00010111	00011000	10101010	which interface?

Longest prefix matching

- we'll see *why* longest prefix matching is used shortly, when we study addressing
- longest prefix matching: often performed using ternary content addressable memories (TCAMs)
 - *content addressable*: present address to TCAM: retrieve address in one clock cycle, regardless of table size
 - Cisco Catalyst: ~1M routing table entries in TCAM

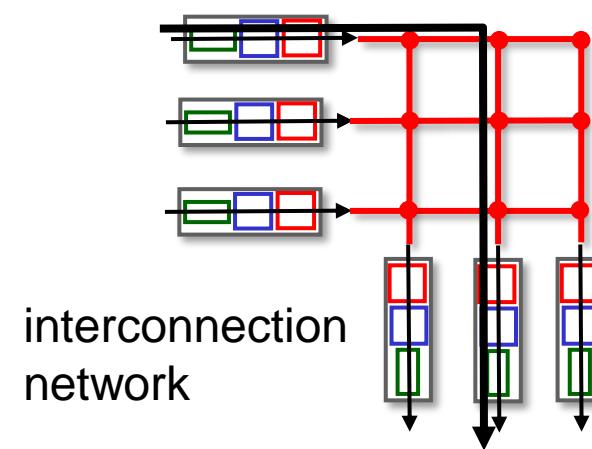
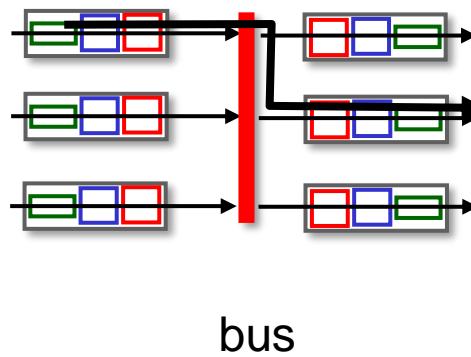
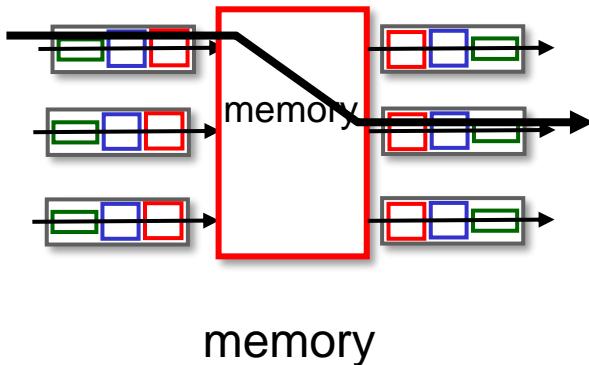
Switching fabrics

- transfer packet from input link to appropriate output link
- **switching rate:** rate at which packets can be transferred from inputs to outputs
 - often measured as multiple of input/output line rate
 - N inputs: switching rate N times line rate desirable



Switching fabrics

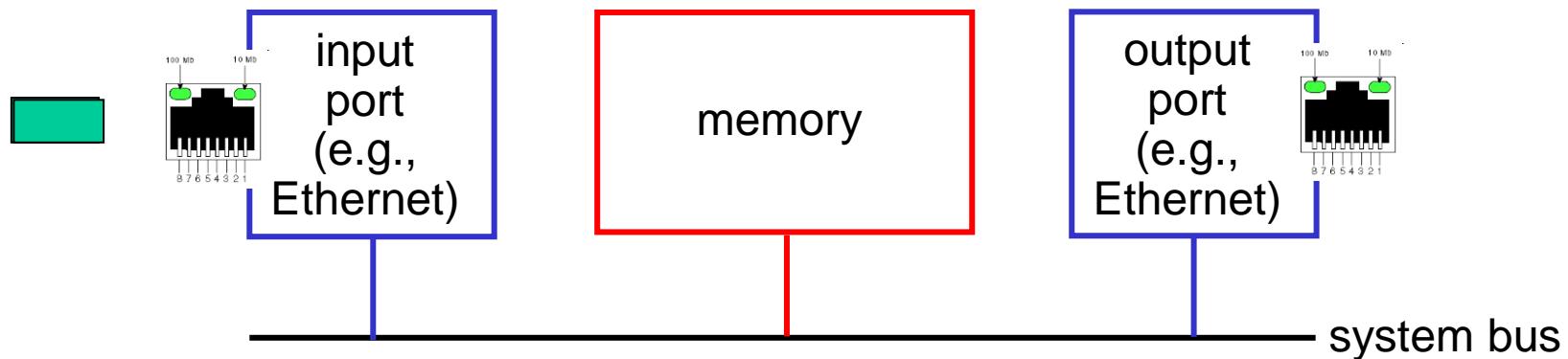
- transfer packet from input link to appropriate output link
- **switching rate:** rate at which packets can be transferred from inputs to outputs
 - often measured as multiple of input/output line rate
 - N inputs: switching rate N times line rate desirable
- three major types of switching fabrics:



Switching via memory

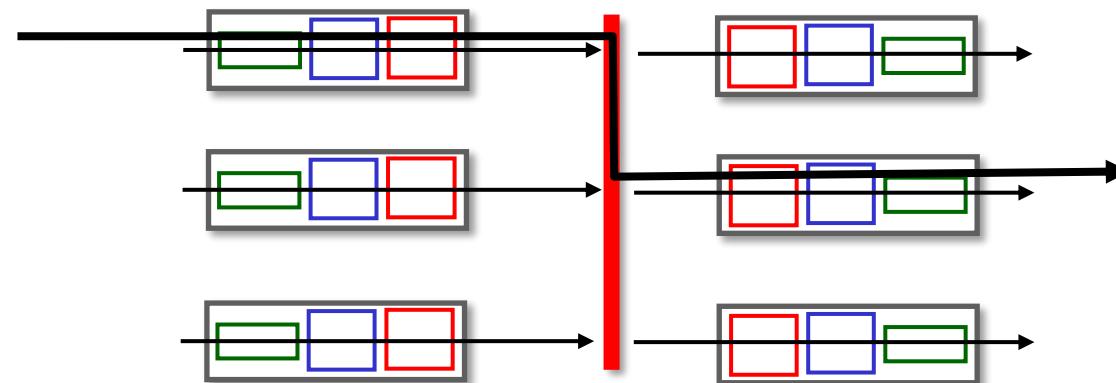
first generation routers:

- traditional computers with switching under direct control of CPU
- packet copied to system's memory
- speed limited by memory bandwidth (2 bus crossings per datagram)



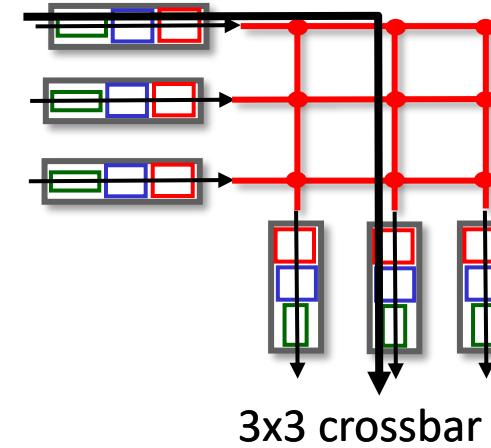
Switching via a bus

- datagram from input port memory to output port memory via a shared bus
- *bus contention*: switching speed limited by bus bandwidth
- 32 Gbps bus, Cisco 5600: sufficient speed for access routers

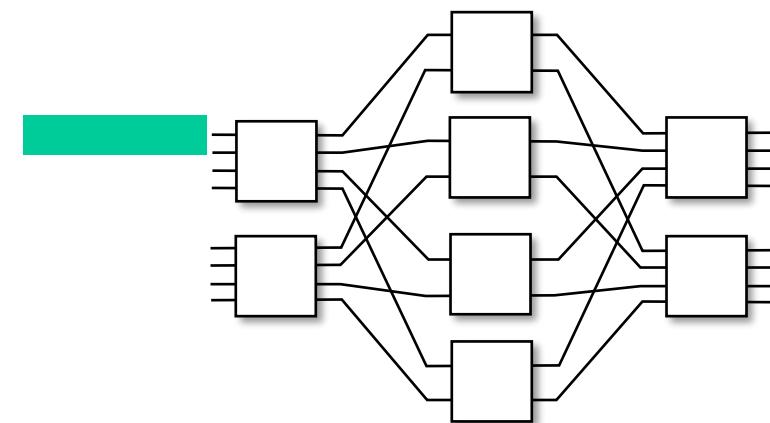


Switching via interconnection network

- Crossbar, Clos networks, other interconnection nets initially developed to connect processors in multiprocessor
- **multistage switch:** $n \times n$ switch from multiple stages of smaller switches
- **exploiting parallelism:**
 - fragment datagram into fixed length cells on entry
 - switch cells through the fabric, reassemble datagram at exit



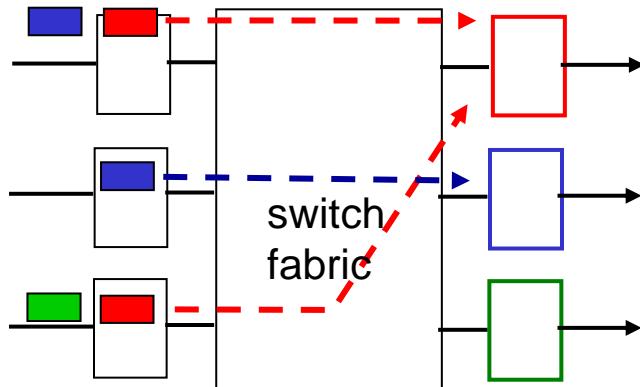
3x3 crossbar



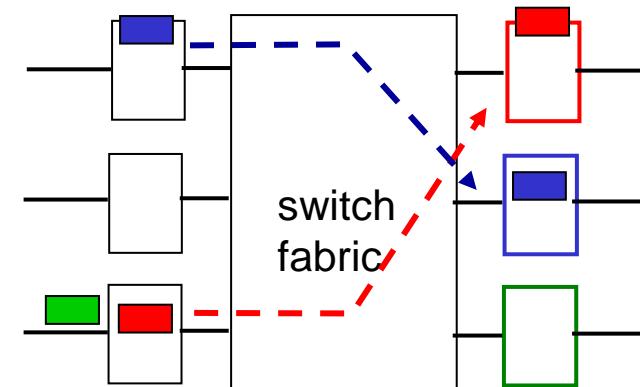
8x8 multistage switch
built from smaller-sized switches

Input port queuing

- If switch fabric slower than input ports combined -> queueing may occur at input queues
 - queueing delay and loss due to input buffer overflow!
- **Head-of-the-Line (HOL) blocking:** queued datagram at front of queue prevents others in queue from moving forward

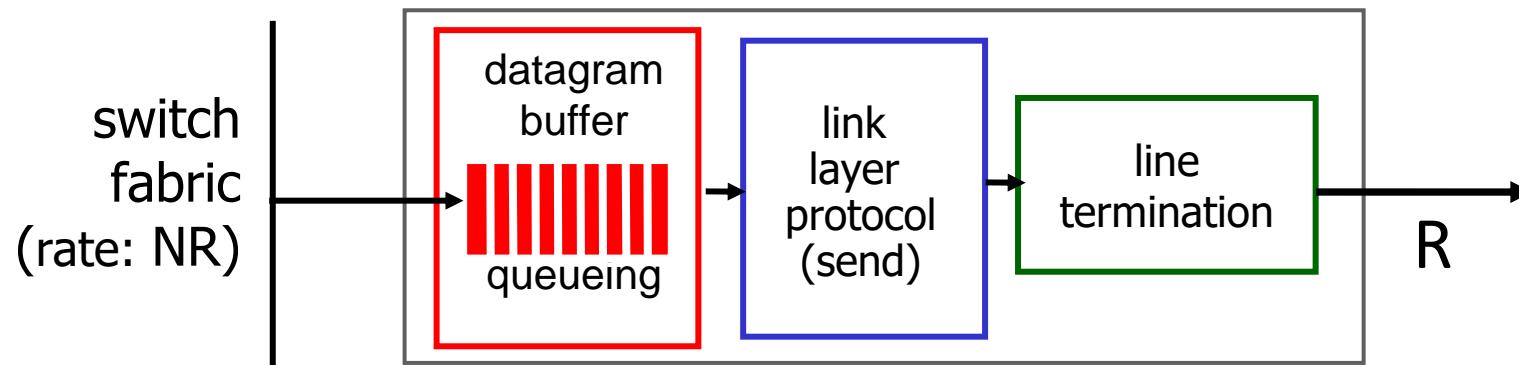


output port contention: only one red datagram can be transferred. lower red packet is *blocked*



one packet time later: green packet experiences HOL blocking

Output port queuing



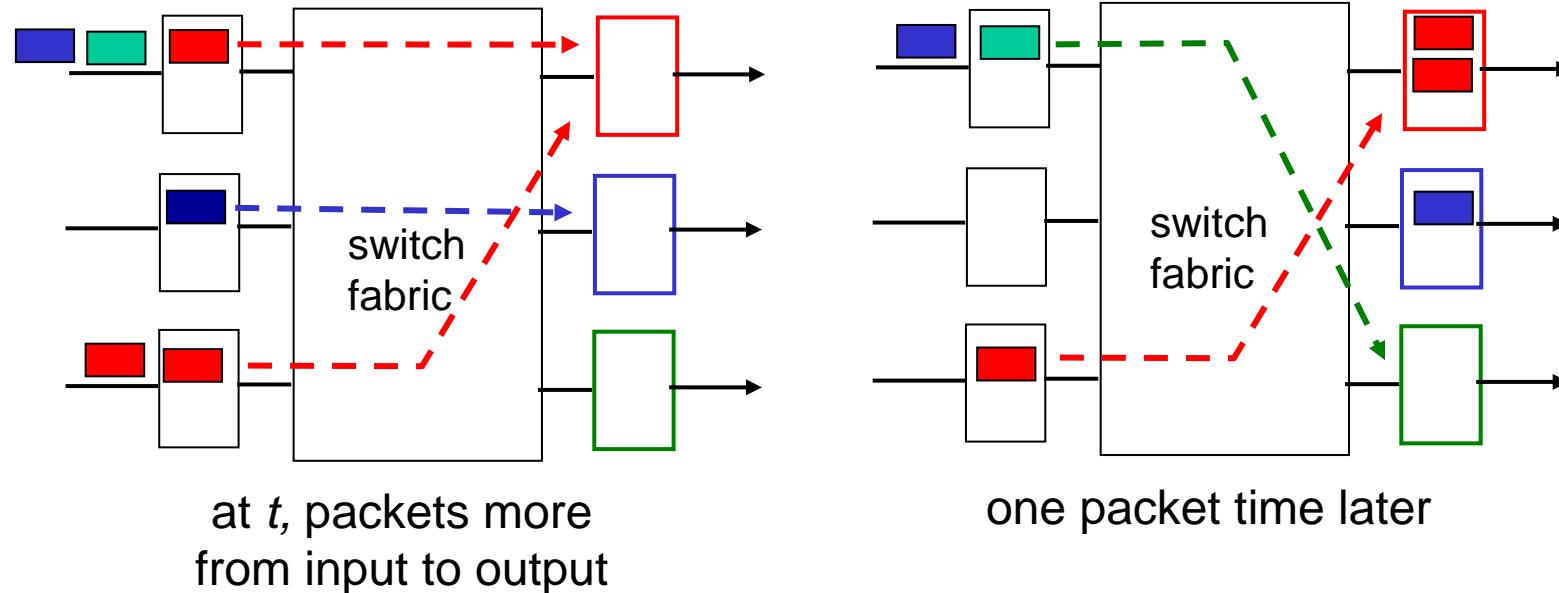
- **Buffering** required when datagrams arrive from fabric faster than link transmission rate. **Drop policy:** which datagrams to drop if no free buffers?

Datagrams can be lost due to congestion, lack of buffers

- **Scheduling discipline** chooses among queued datagrams for transmission

Priority scheduling – who gets best performance, network neutrality

Output port queuing



- buffering when arrival rate via switch exceeds output line speed
- *queueing (delay) and loss due to output port buffer overflow!*

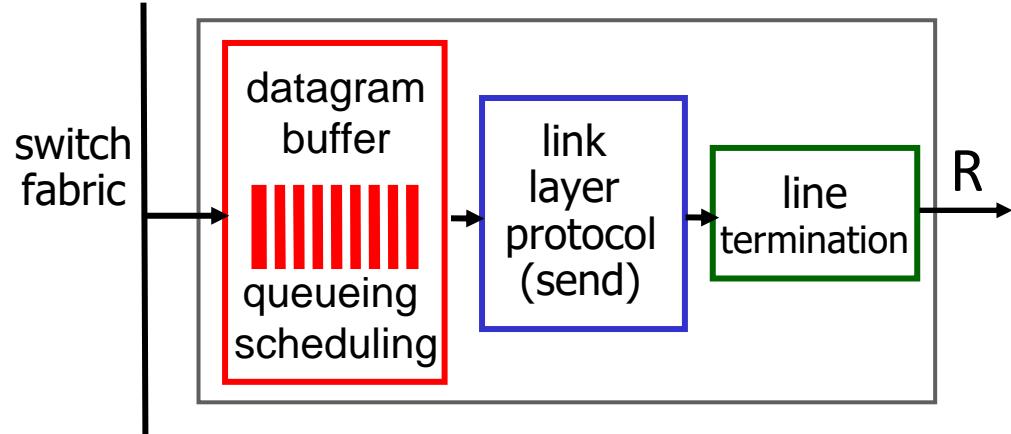
How much buffering?

- RFC 3439 rule of thumb: average buffering equal to “typical” RTT (say 250 msec) times link capacity C
 - e.g., $C = 10 \text{ Gbps}$ link: 2.5 Gbit buffer
- more recent recommendation: with N flows, buffering equal to

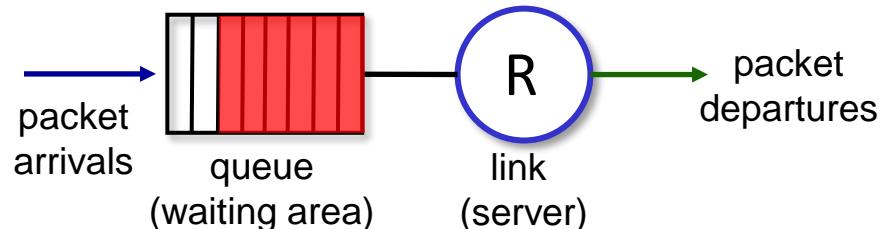
$$\frac{\text{RTT} \cdot C}{\sqrt{N}}$$

- but *too* much buffering can increase delays (particularly in home routers)
 - long RTTs: poor performance for realtime apps, sluggish TCP response
 - recall delay-based congestion control: “keep bottleneck link just full enough (busy) but no fuller”

Buffer Management



Abstraction: queue



buffer management:

- **drop**: which packet to add, drop when buffers are full
 - tail drop: drop arriving packet
 - priority: drop/remove on priority basis

- **marking**: which packets to mark to signal congestion (ECN, RED)

Packet Scheduling: FCFS

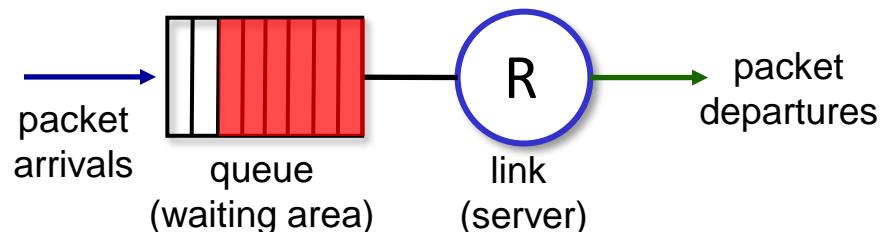
packet scheduling: deciding which packet to send next on link

- first come, first served
- priority
- round robin
- weighted fair queueing

FCFS: packets transmitted in order of arrival to output port

- also known as: First-in-first-out (FIFO)
- real world examples?

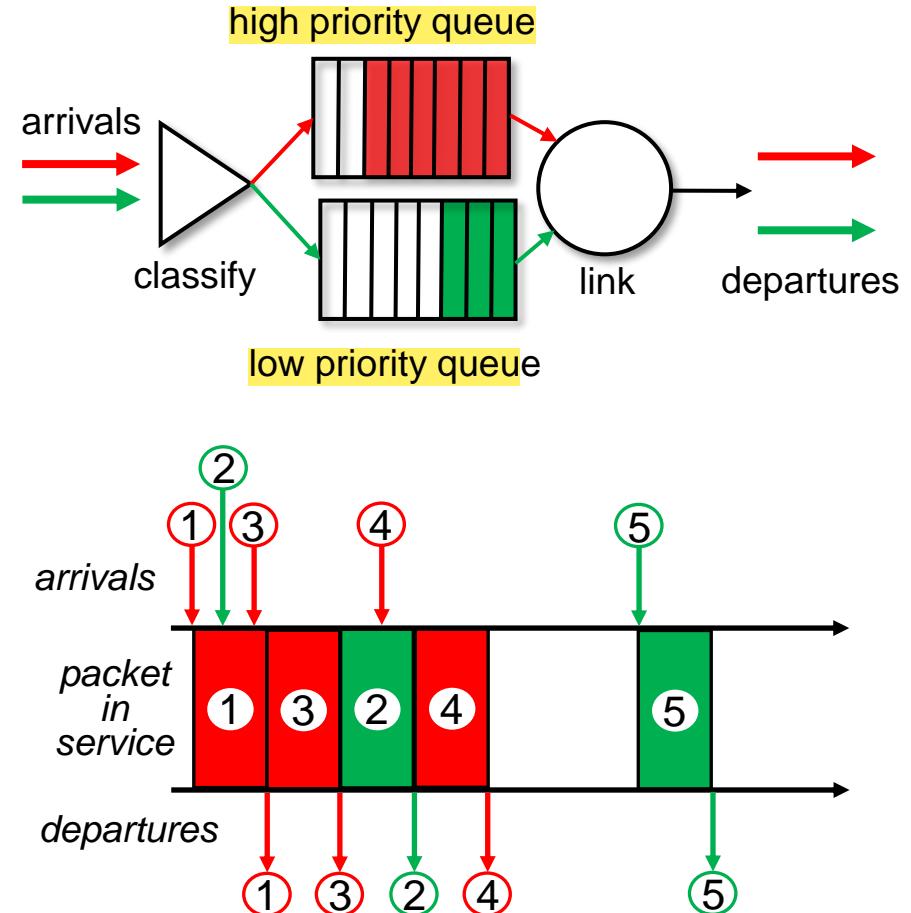
Abstraction: queue



Scheduling policies: priority

Priority scheduling:

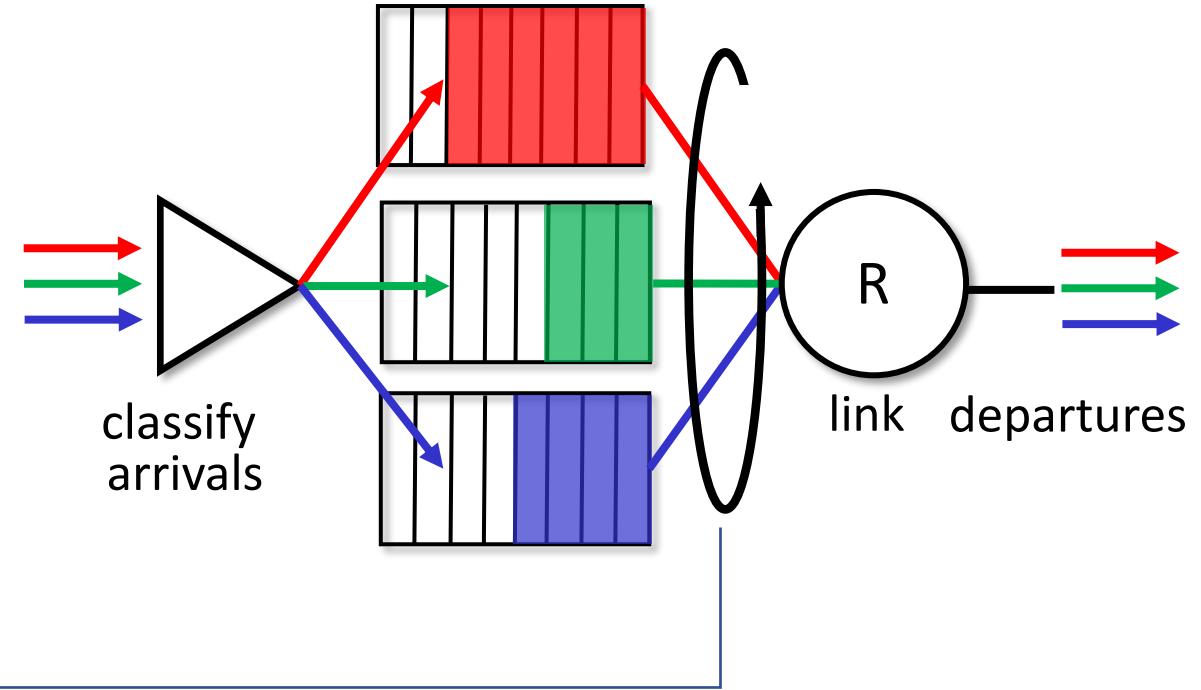
- arriving traffic classified, queued by class
 - any header fields can be used for classification
- send packet from highest priority queue that has buffered packets
 - FCFS within priority class



Scheduling policies: round robin

Round Robin (RR) scheduling:

- arriving traffic classified, queued by class
 - any header fields can be used for classification
- server cyclically, repeatedly scans class queues, sending one complete packet from each class (if available) in turn



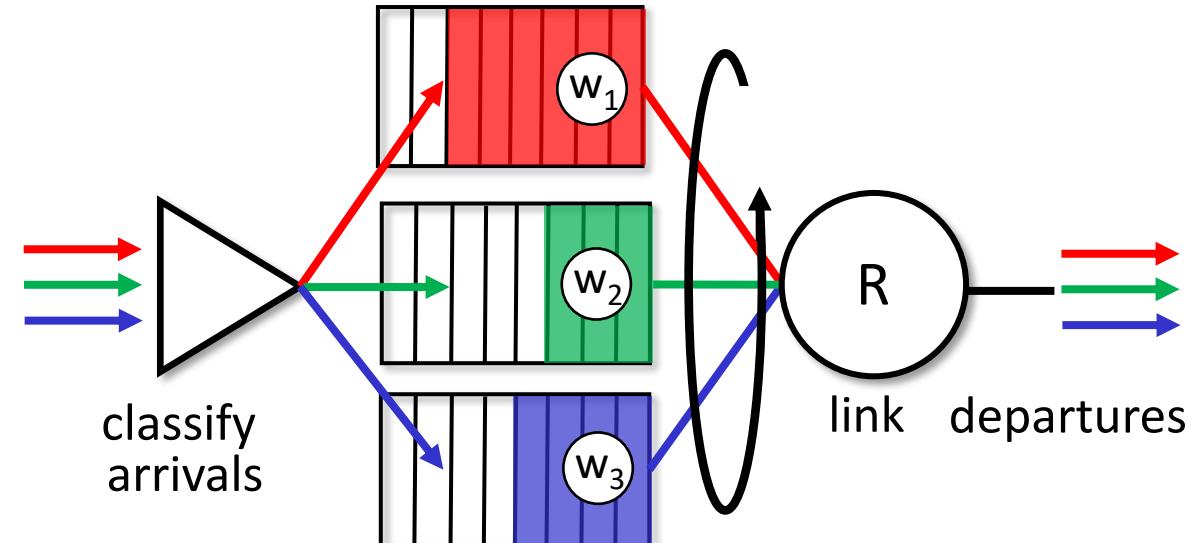
Scheduling policies: weighted fair queueing

Weighted Fair Queueing (WFQ):

- generalized Round Robin
- each class, i , has weight, w_i , and gets weighted amount of service in each cycle:

$$\frac{w_i}{\sum_j w_j}$$

- minimum bandwidth guarantee (per-traffic-class)



Sidebar: Network Neutrality

What is network neutrality?

- *technical*: how an ISP should share/allocation its resources
 - packet scheduling, buffer management are the *mechanisms*
- *social, economic* principles
 - protecting free speech
 - encouraging innovation, competition
- enforced *legal* rules and policies

Different countries have different “takes” on network neutrality

Sidebar: Network Neutrality

2015 US FCC *Order on Protecting and Promoting an Open Internet*: three “clear, bright line” rules:

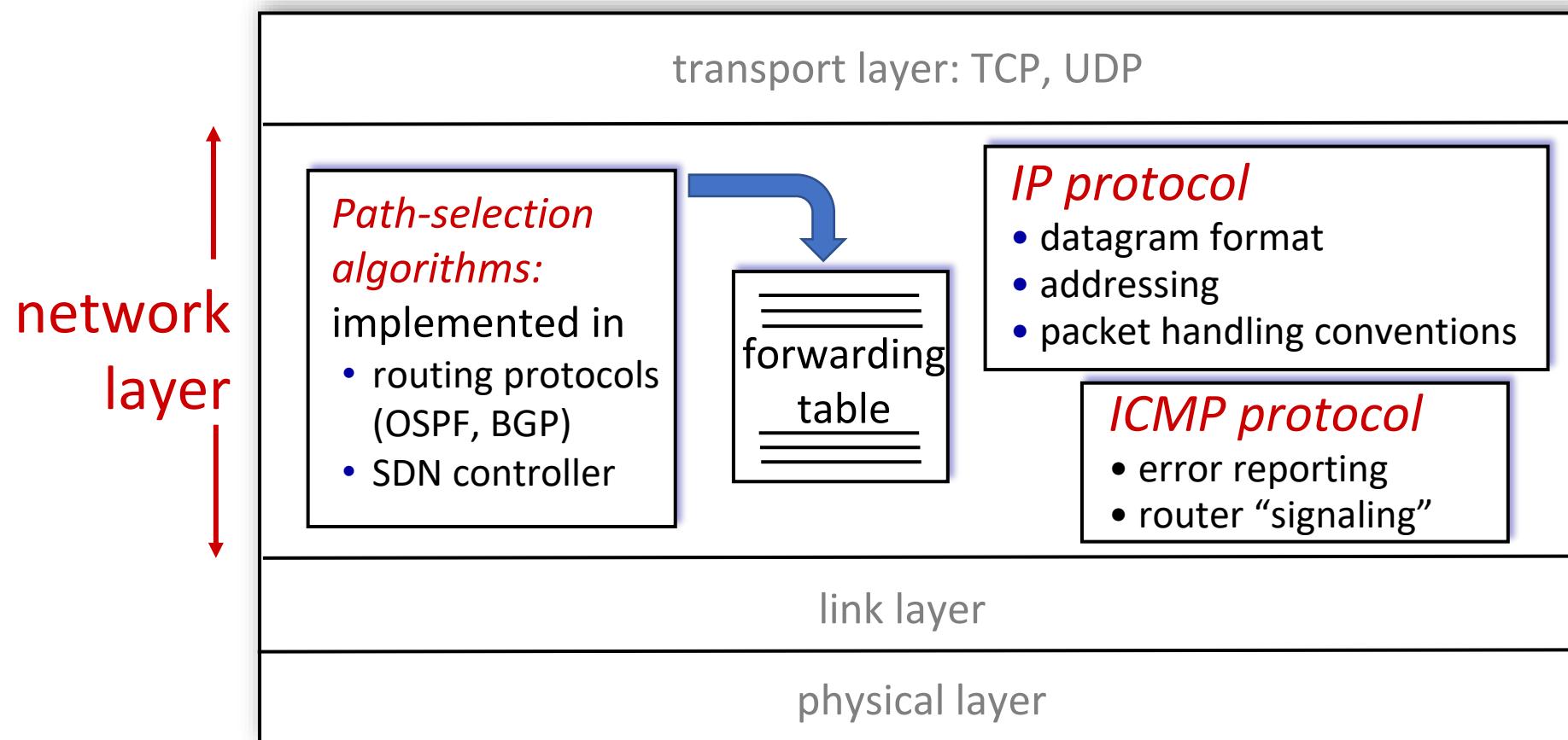
- **no blocking** ... “shall not block lawful content, applications, services, or non-harmful devices, subject to reasonable network management.”
- **no throttling** ... “shall not impair or degrade lawful Internet traffic on the basis of Internet content, application, or service, or use of a non-harmful device, subject to reasonable network management.”
- **no paid prioritization.** ... “shall not engage in paid prioritization”

Network layer roadmap

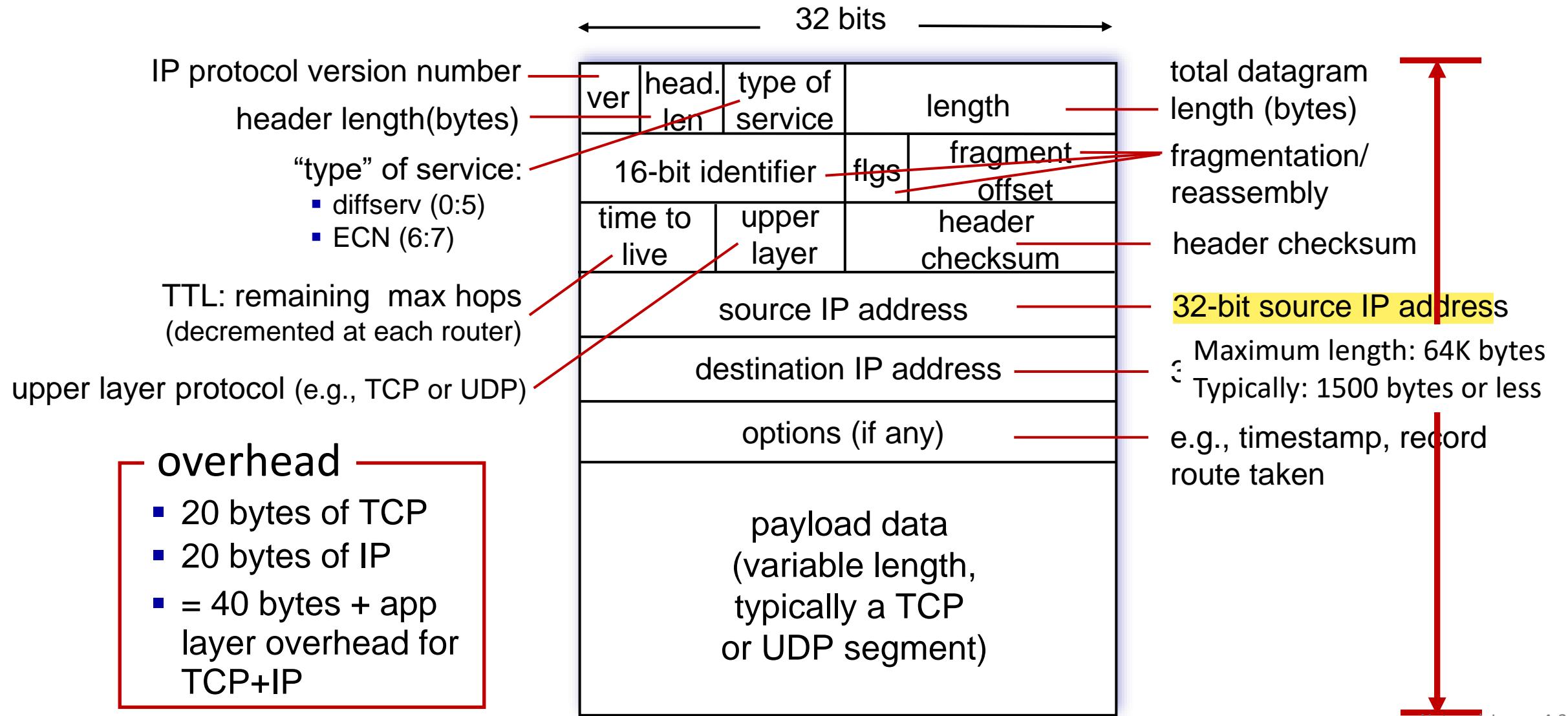
- Network layer: overview
 - data plane
 - control plane
- What's inside a router
 - input ports, switching, output ports
 - buffer management, scheduling
- IP: the Internet Protocol
 - datagram format
 - addressing
 - network address translation
 - IPv6
- Generalized Forwarding, SDN
 - match+action
 - OpenFlow: match+action in action
- Middleboxes

Network Layer: Internet

host, router network layer functions:

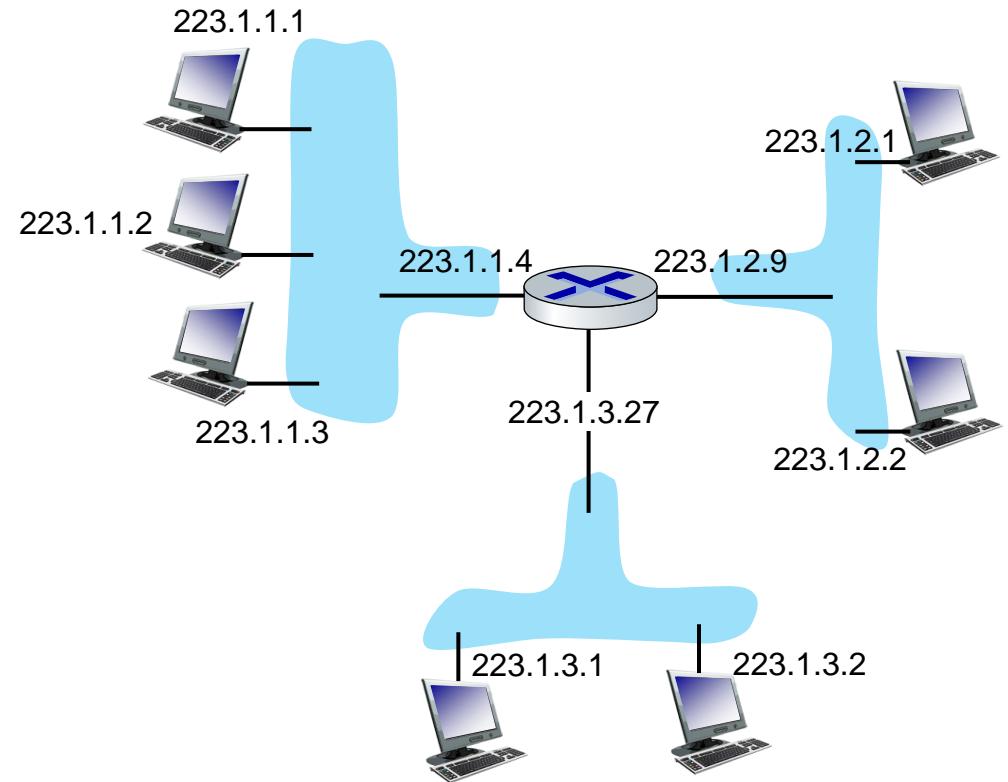


IP Datagram format



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)

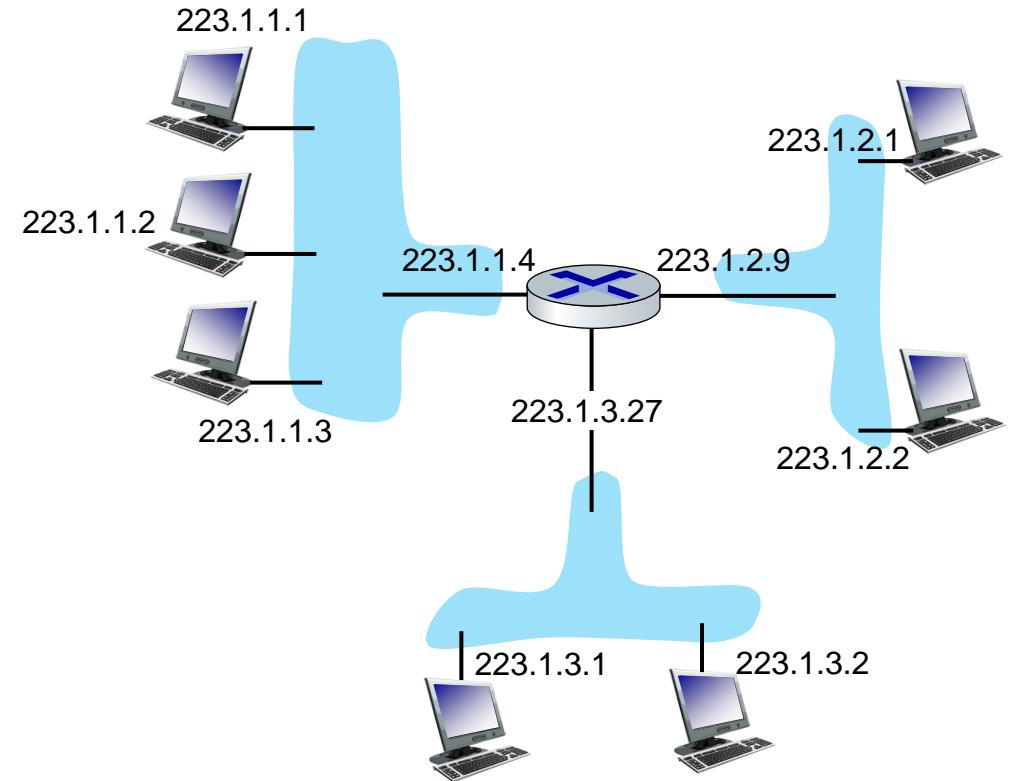


dotted-decimal IP address notation:

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

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:

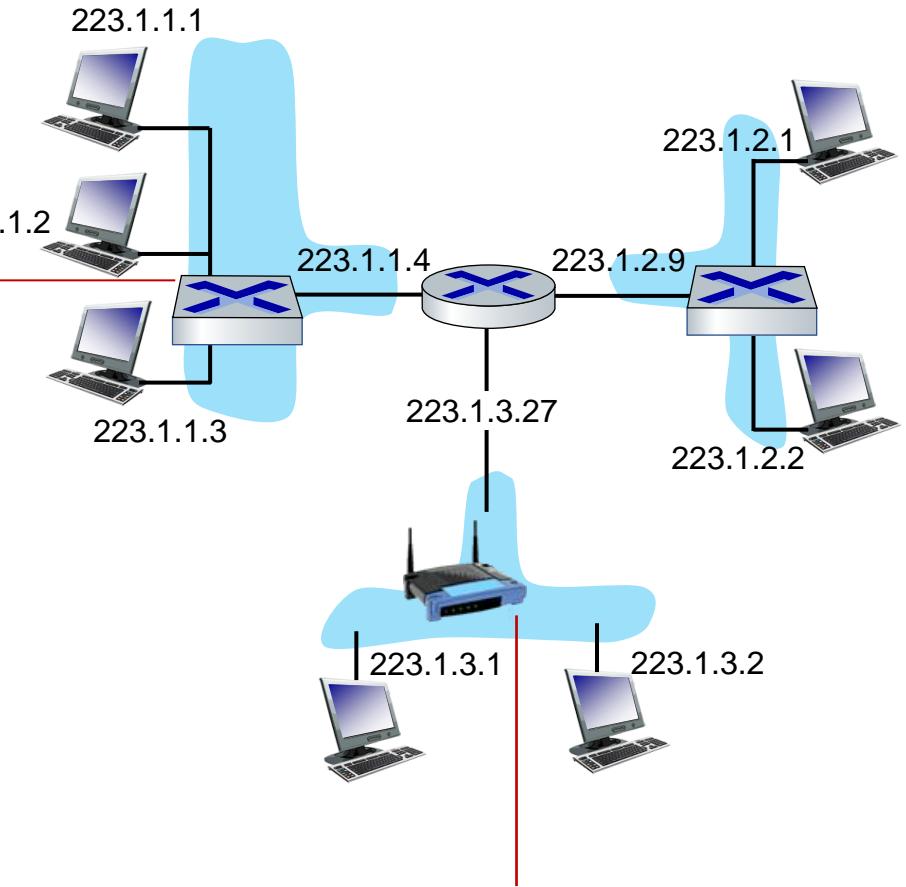
223.1.1.1 = $\begin{array}{cccc} 11011111 & 00000001 & 00000001 & 00000001 \end{array}$

223 1 1 1

IP addressing: introduction

Q: how are interfaces actually connected?

A: wired
Ethernet interfaces
connected by
Ethernet switches



A: wireless WiFi interfaces
connected by WiFi base station

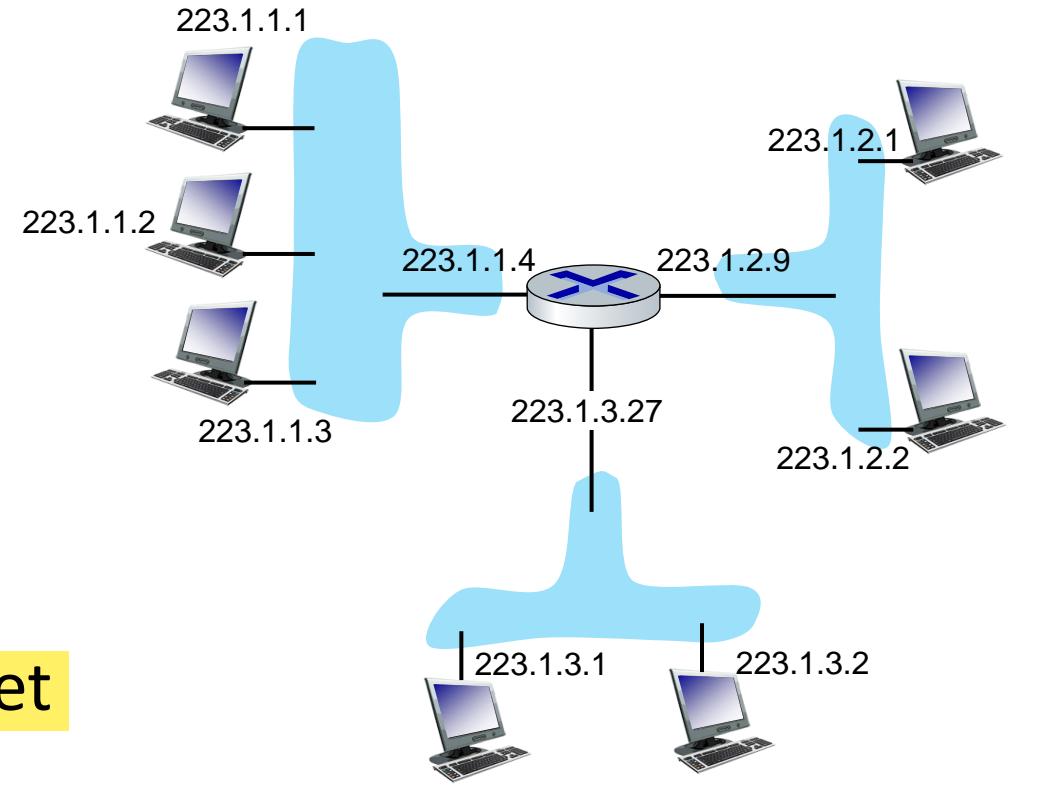
Subnets

■ 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

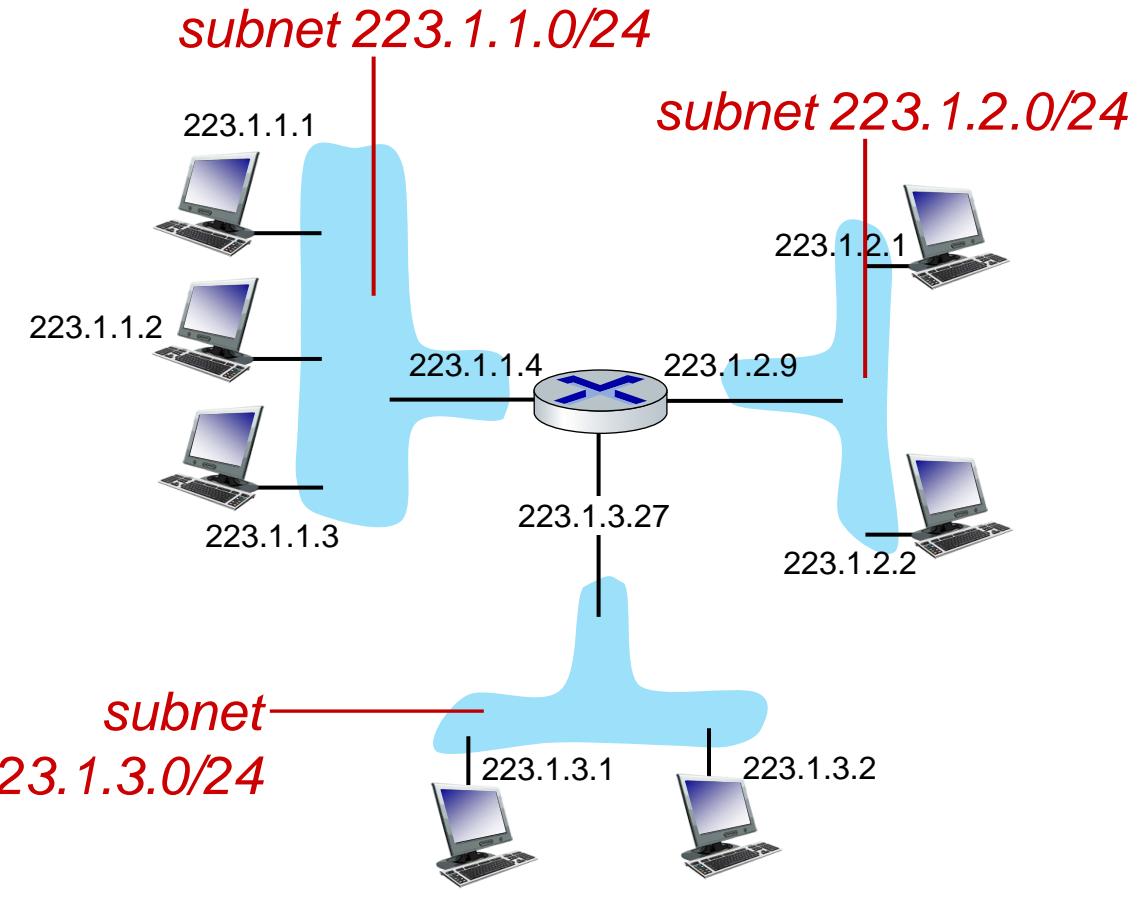


network consisting of 3 subnets

Subnets

Recipe for defining subnets:

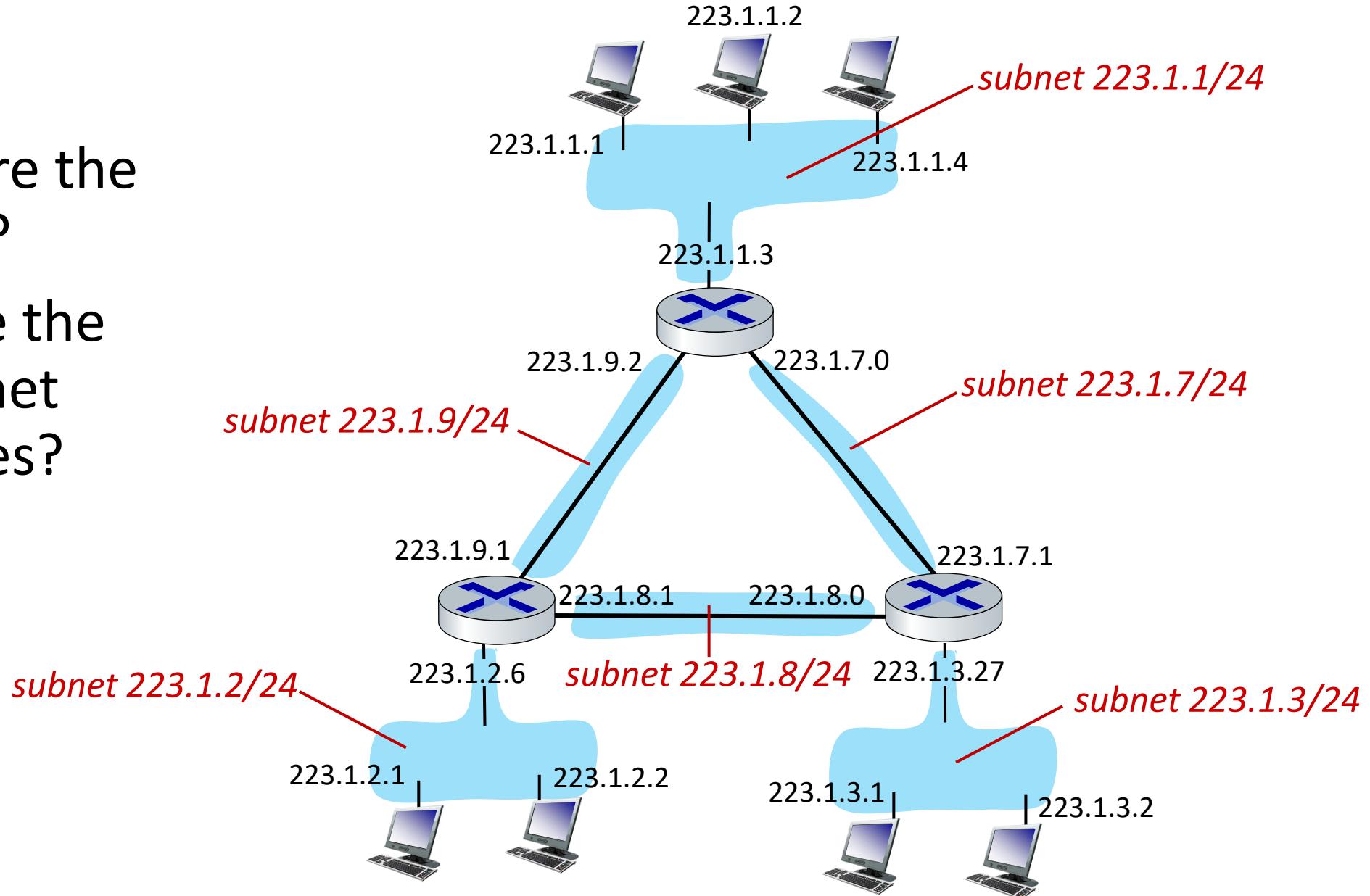
- detach each interface from its host or router, creating “islands” of isolated networks
- each isolated network is called a *subnet*



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

Subnets

- 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. How does a *host* get IP address within its network (host part of address)?
2. 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

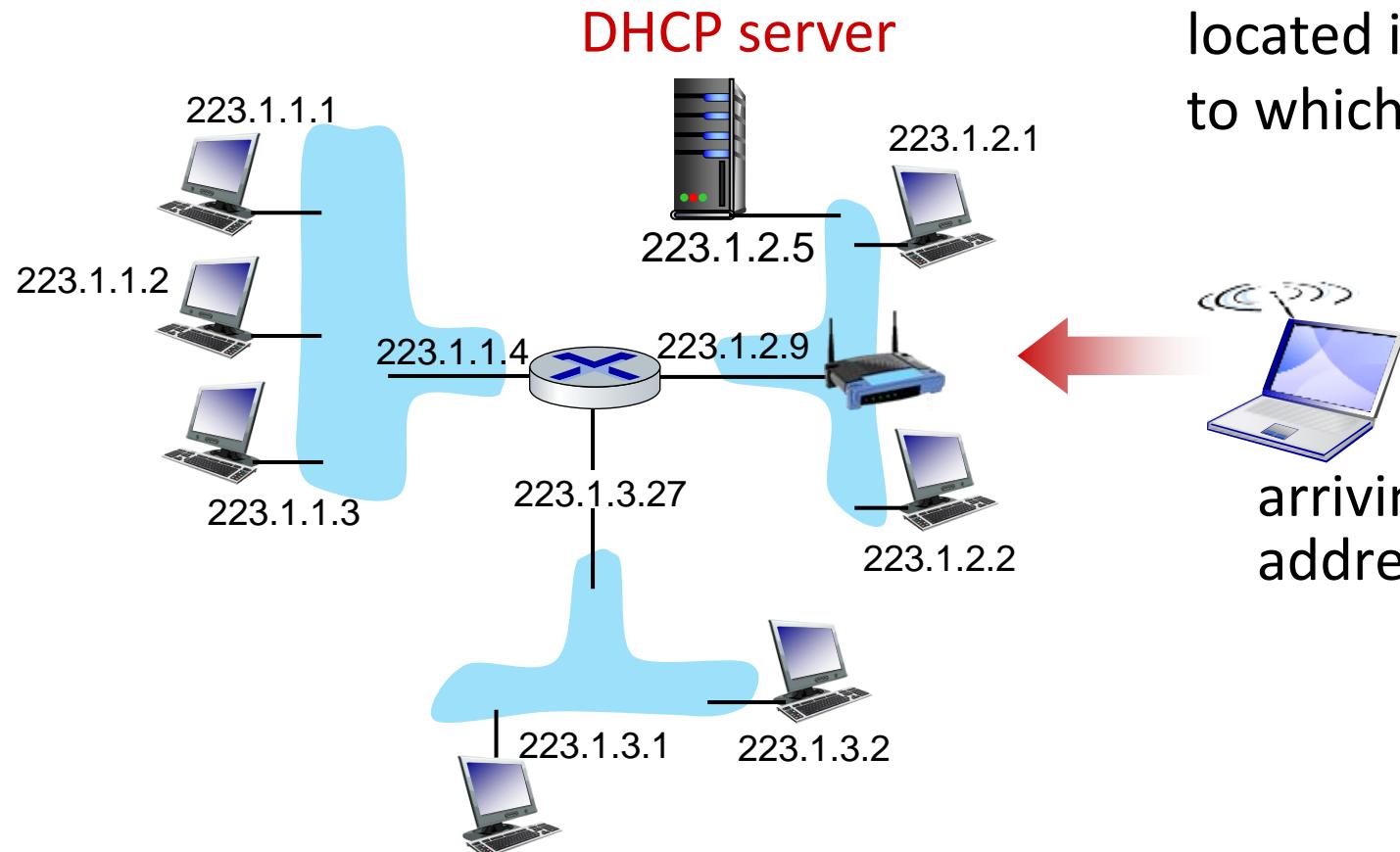
goal: host *dynamically* obtains IP address from network server when it “joins” network

- 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

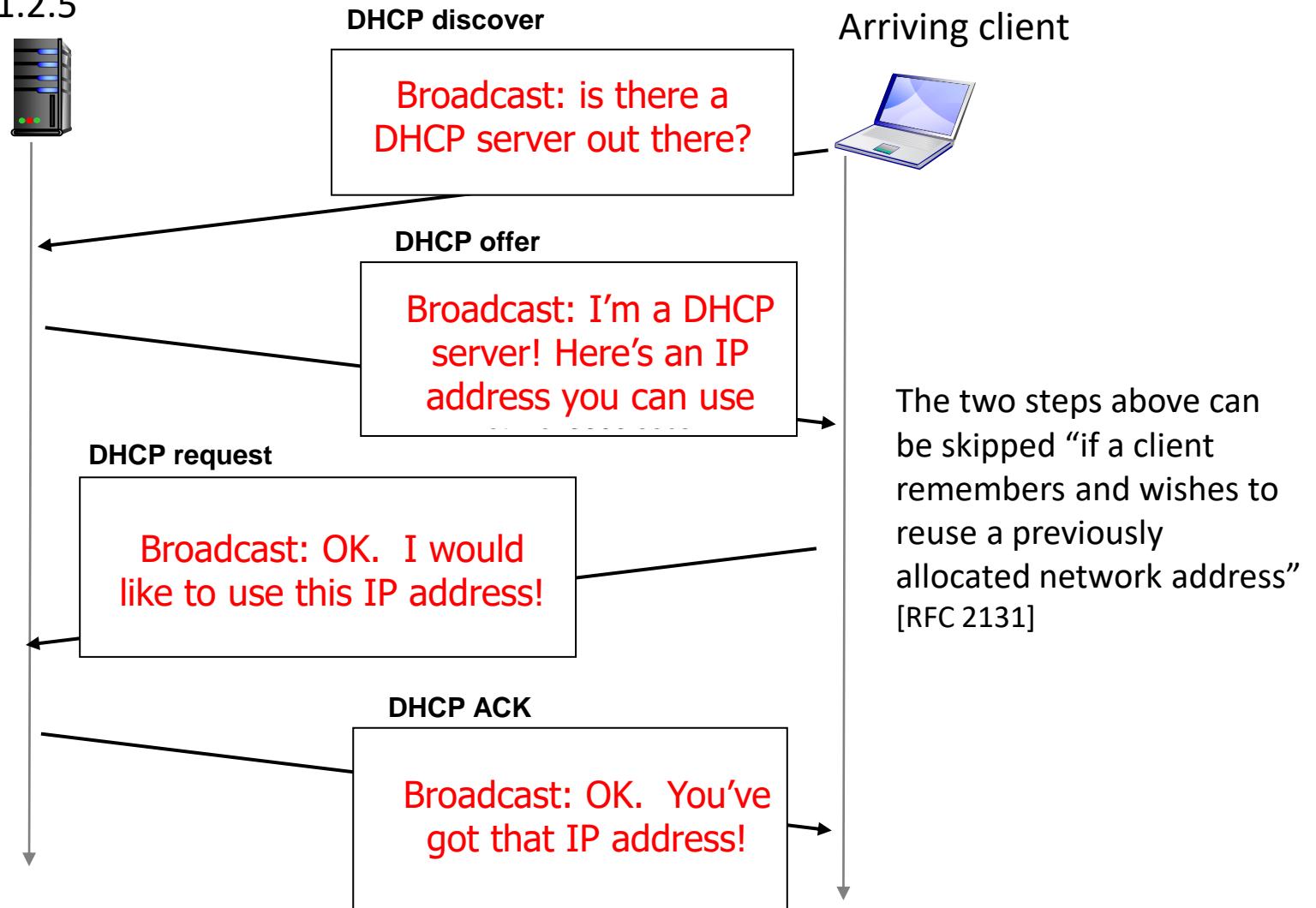


Typically, DHCP server will be co-located in router, serving all subnets to which router is attached

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

DHCP client-server scenario

DHCP server: 223.1.2.5

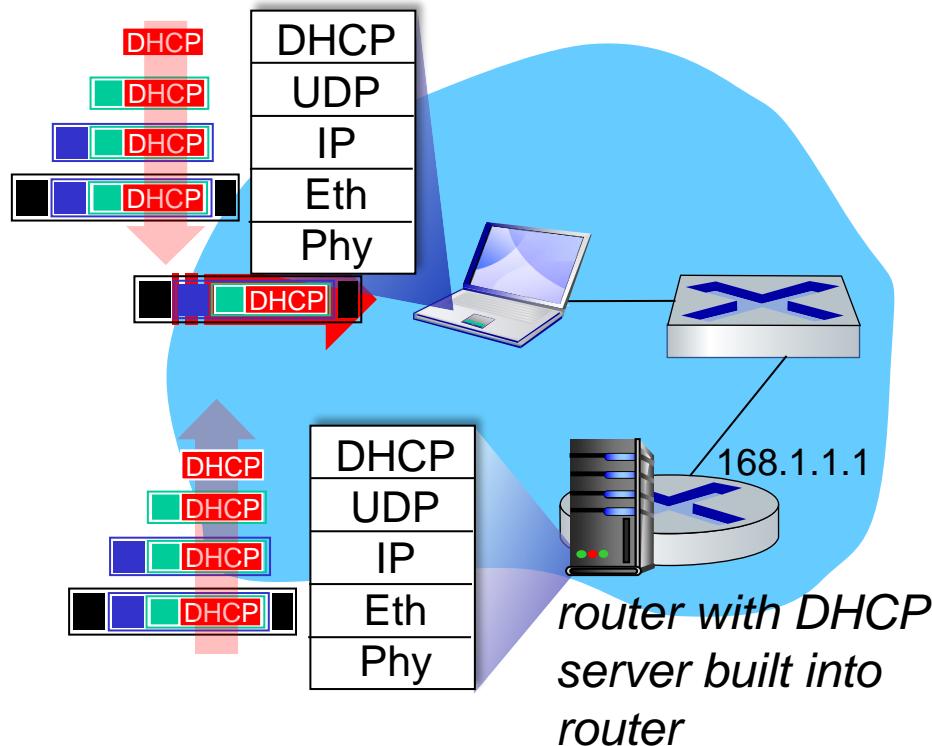


DHCP: more than IP addresses

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

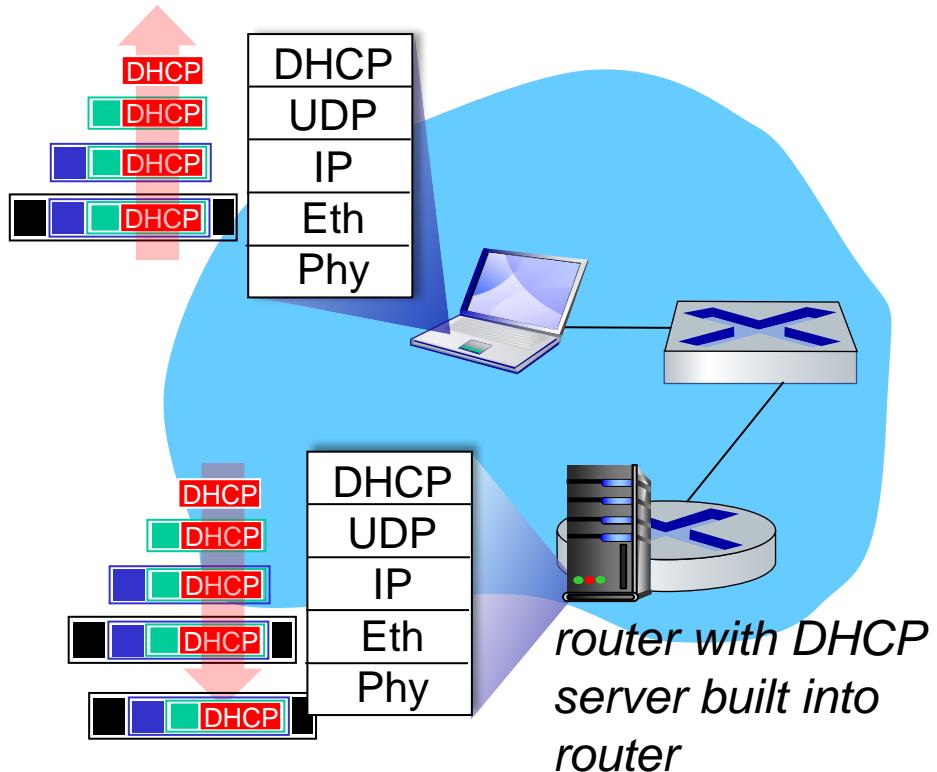
- address of first-hop router for client
- name and IP address of DNS sever
- network mask (indicating network versus host portion of address)

DHCP: example



- Connecting laptop will use DHCP to get IP address, address of first-hop router, address of DNS server.
- DHCP REQUEST message encapsulated in UDP, encapsulated in IP, encapsulated in Ethernet
- Ethernet frame broadcast (dest: FFFFFFFFFFFF) on LAN, received at router running DHCP server
- Ethernet demux'ed to IP demux'ed, UDP demux'ed to DHCP

DHCP: example



- DCP server formulates DHCP ACK containing client's IP address, IP address of first-hop router for client, name & IP address of DNS server
- encapsulated DHCP server reply forwarded to client, demuxing up to DHCP at client
- client now knows its IP address, name and IP address of DNS server, IP address of its first-hop router

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
----------------	---	----------------

...

.....

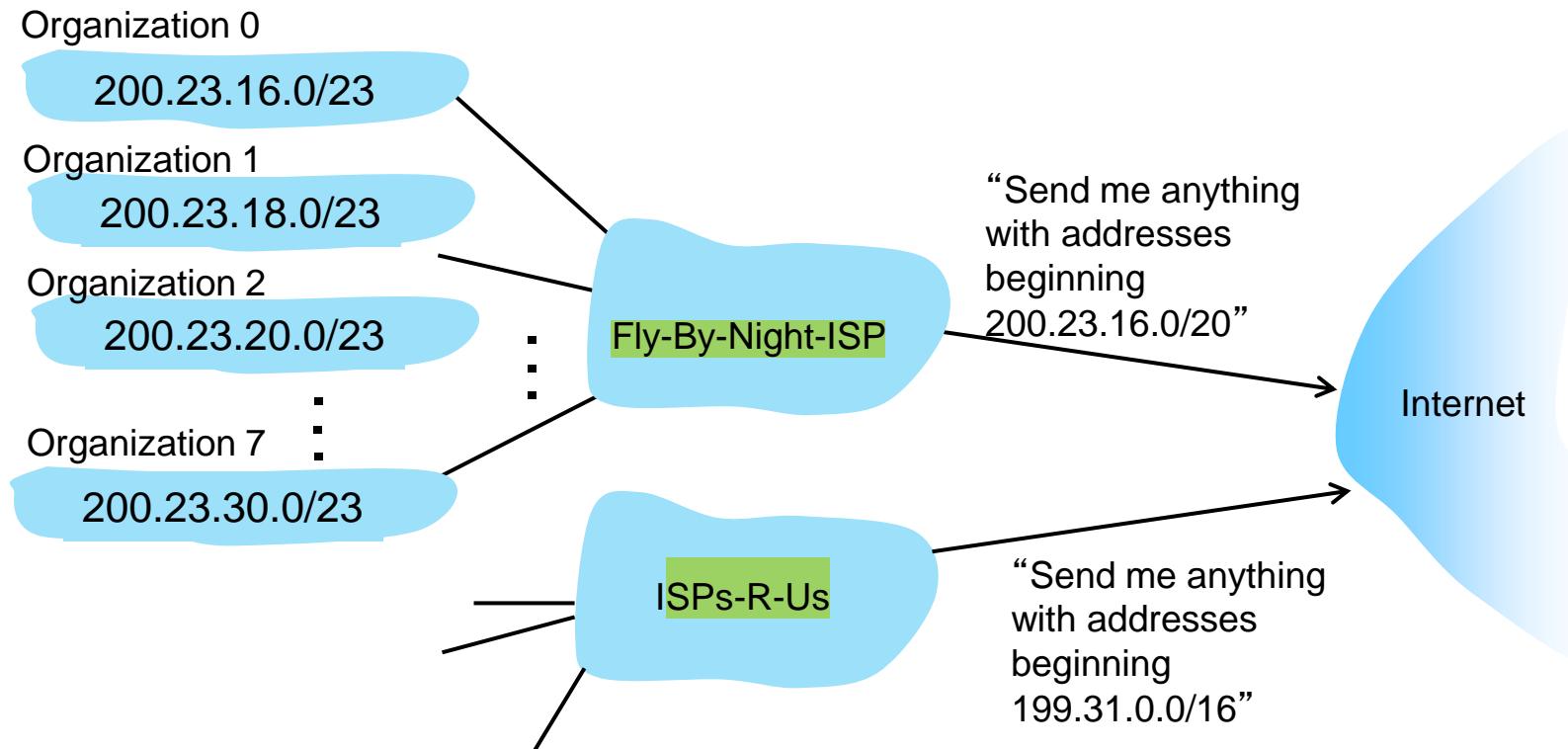
.....

.....

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

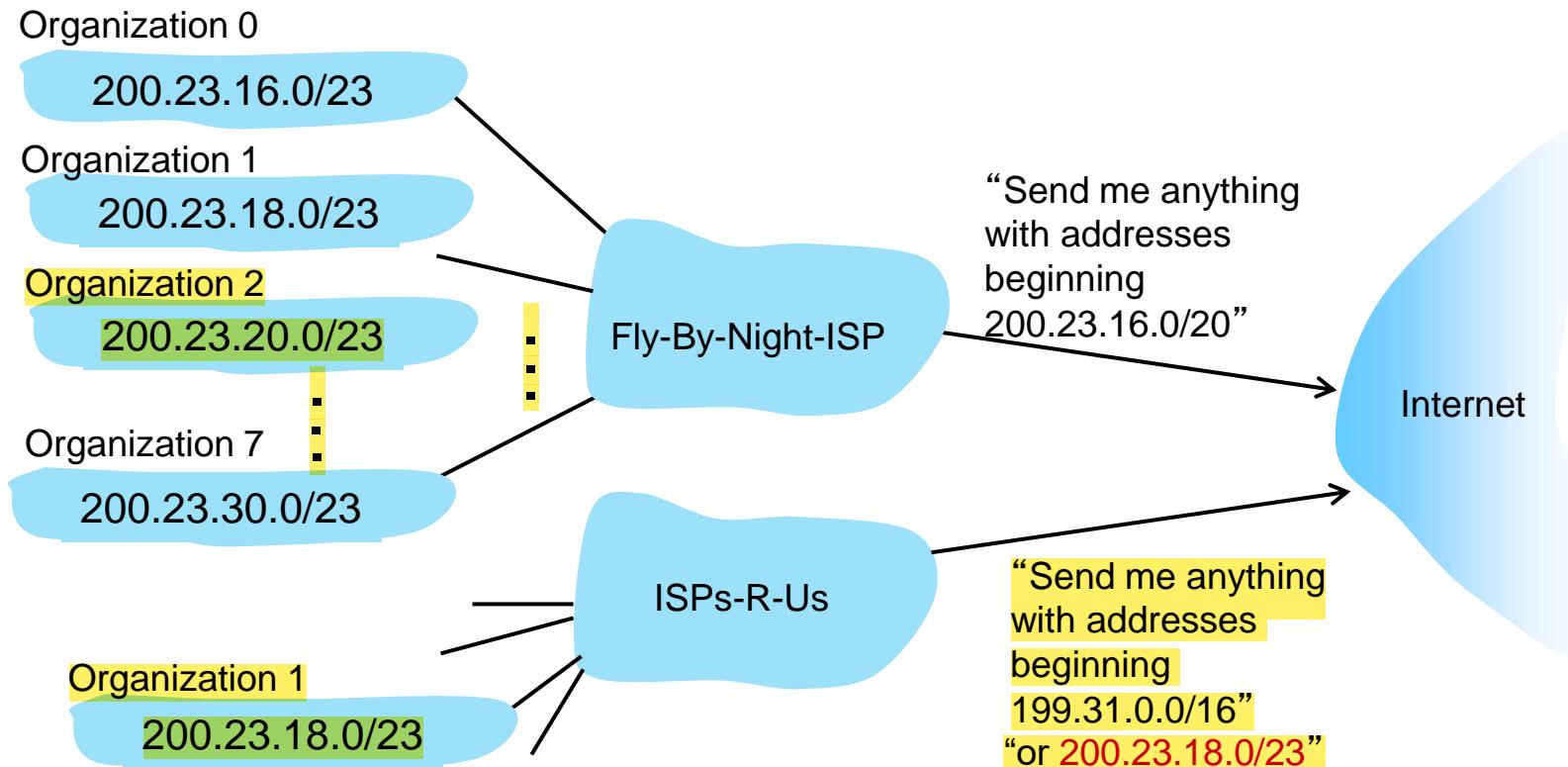
Hierarchical addressing: route aggregation

hierarchical addressing allows efficient advertisement of routing information:



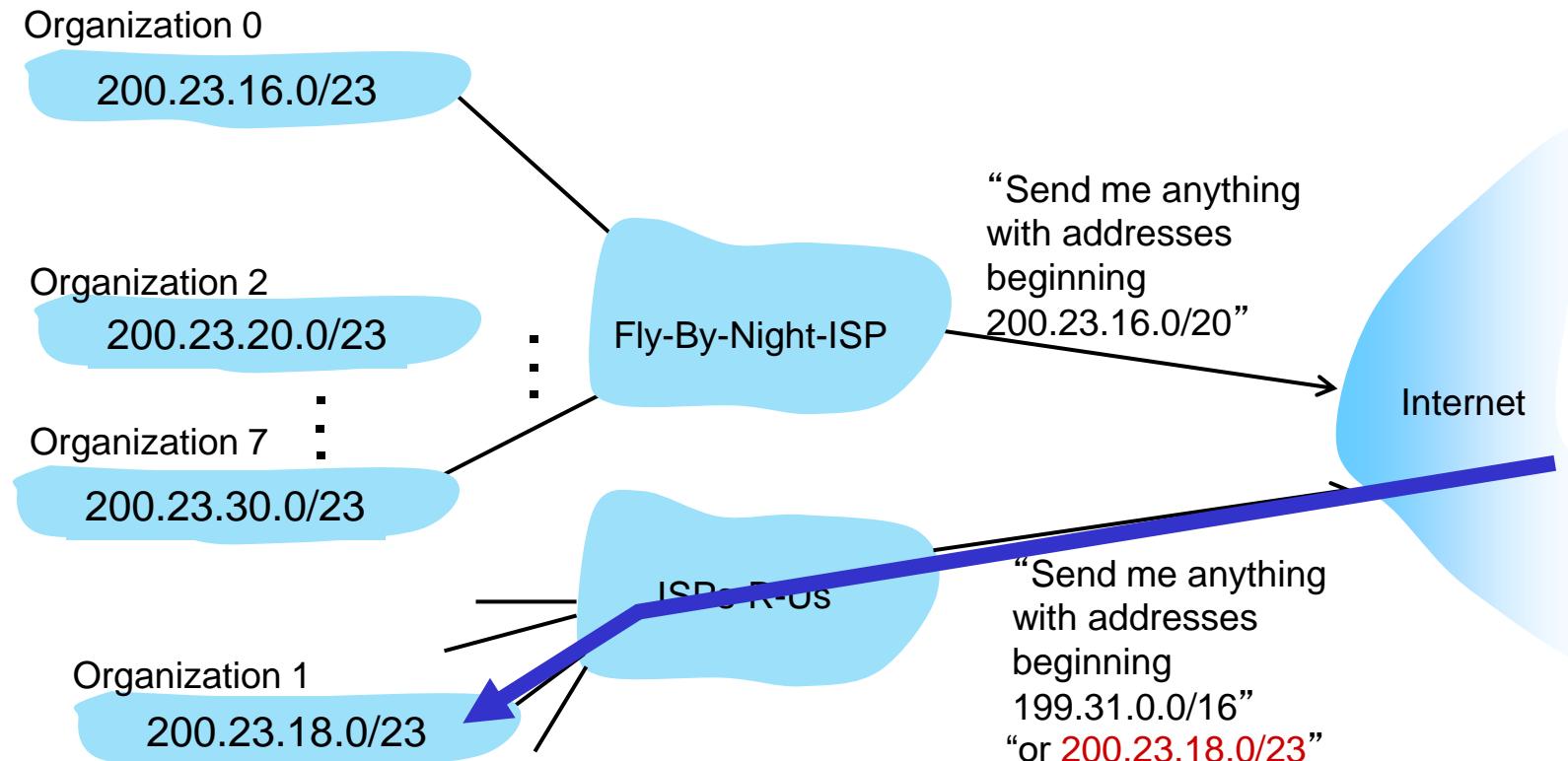
Hierarchical addressing: more specific routes

- Organization 1 moves from Fly-By-Night-ISP to ISPs-R-Us
- ISPs-R-Us now advertises a more specific route to Organization 1



Hierarchical addressing: more specific routes

- Organization 1 moves from Fly-By-Night-ISP to ISPs-R-Us
- ISPs-R-Us now advertises a more specific route to Organization 1



IP addressing: last words ...

Q: how does an ISP get block of addresses?

A: ICANN: Internet Corporation for Assigned Names and Numbers

<http://www.icann.org/>

- allocates IP addresses, through 5 regional registries (RRs) (who may then allocate to local registries)
- manages DNS root zone, including delegation of individual TLD (.com, .edu , ...) management

Q: are there enough 32-bit IP addresses?

- ICANN allocated last chunk of IPv4 addresses to RRs in 2011
- NAT (next) helps IPv4 address space exhaustion
- **IPv6 has 128-bit address space**

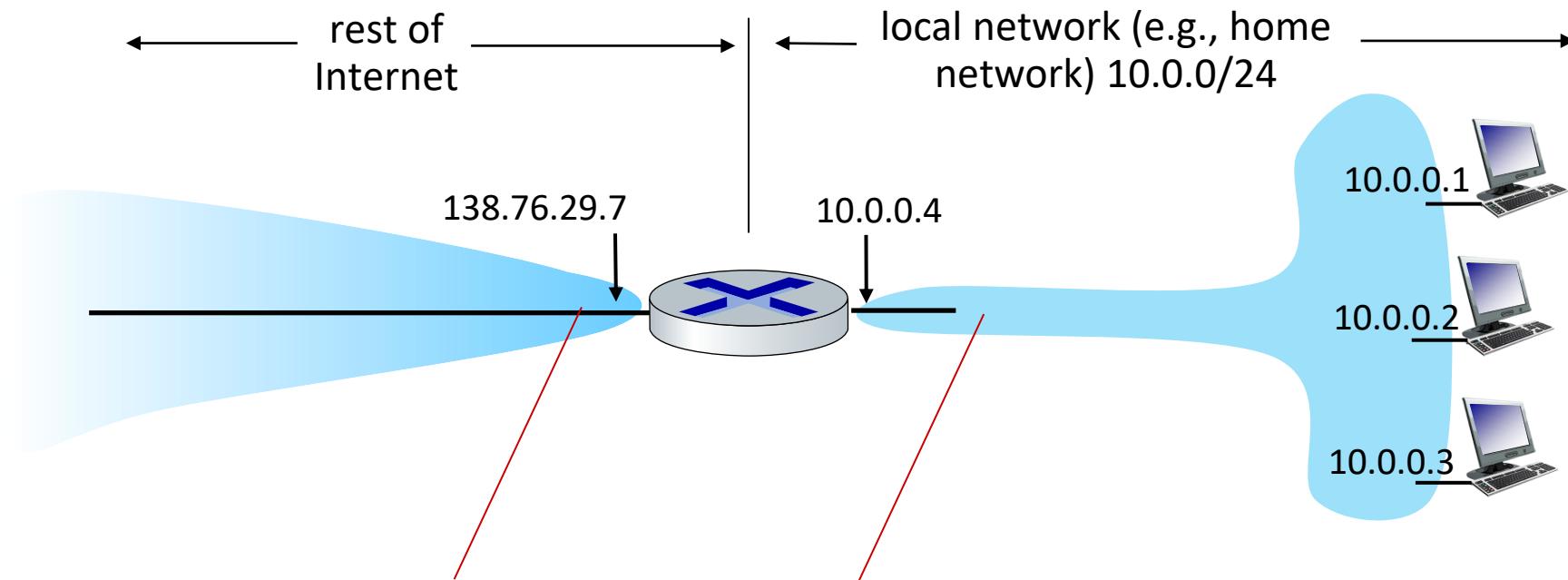
"Who the hell knew how much address space we needed?" Vint Cerf (reflecting on decision to make IPv4 address 32 bits long)

Network layer 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
- Generalized Forwarding, SDN
 - match+action
 - OpenFlow: match+action in action
- Middleboxes

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

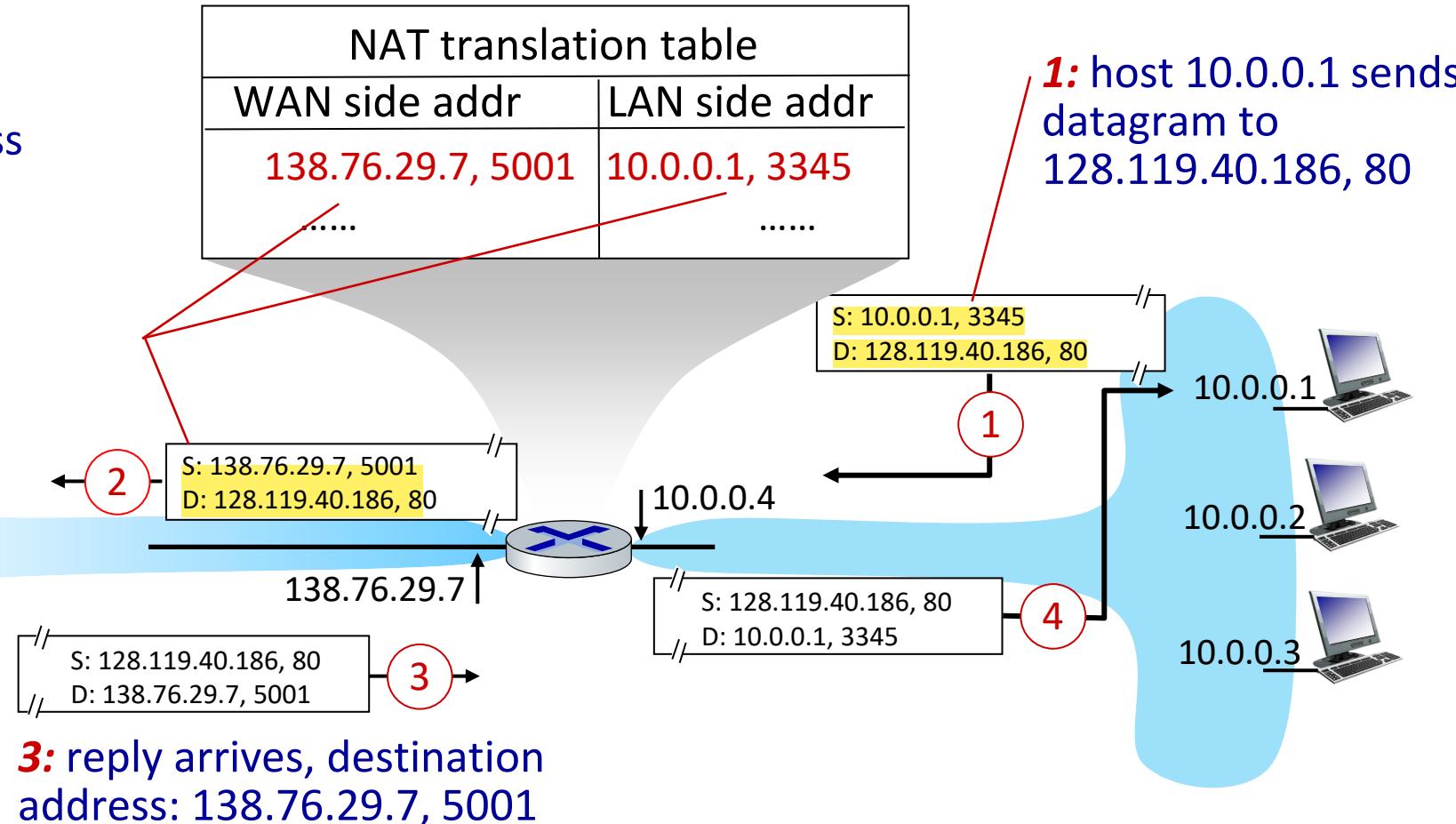
NAT: network address translation

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

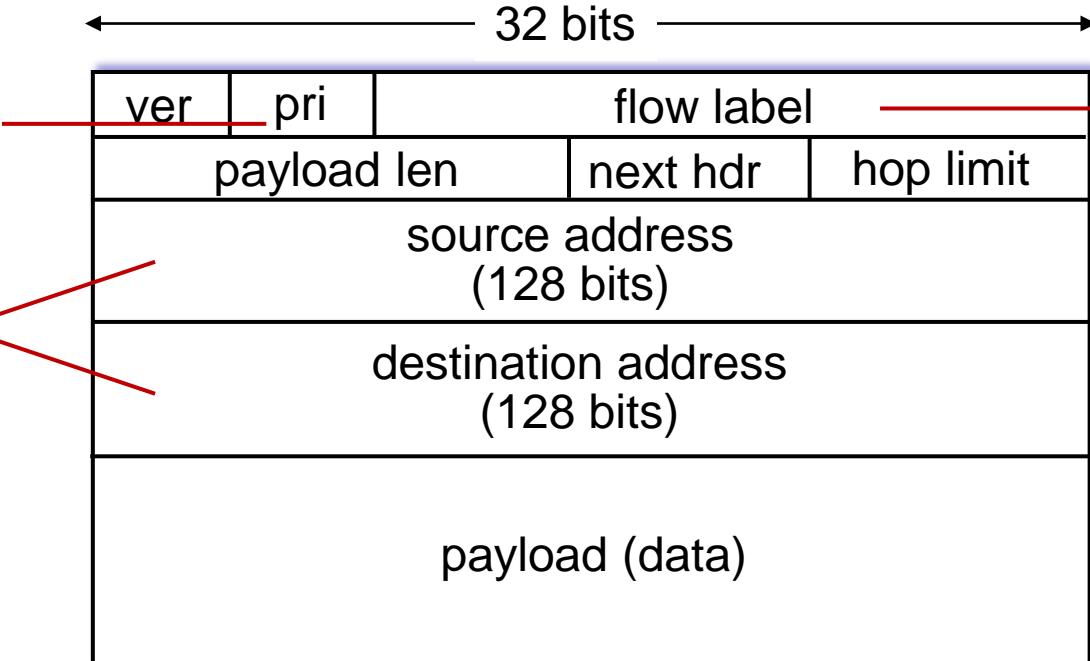
IPv6: motivation

- **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



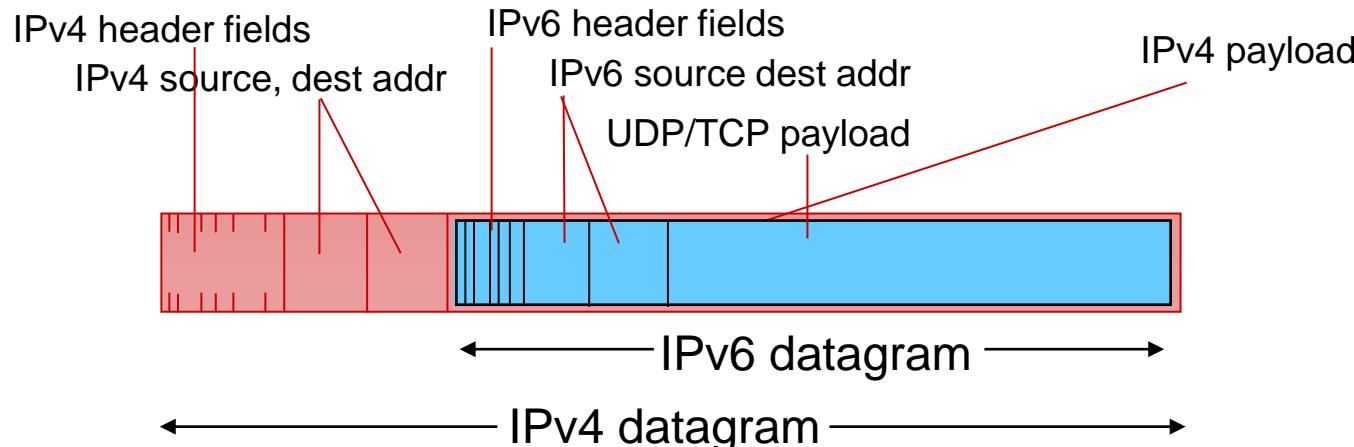
flow label: identify datagrams in same "flow." (concept of "flow" not well defined).

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)

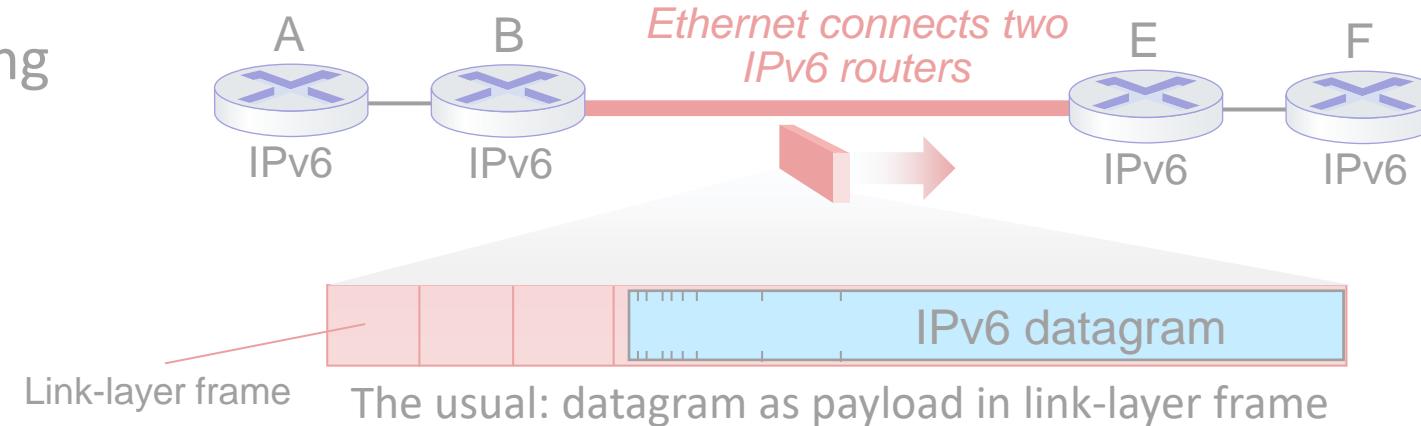
Transition from IPv4 to IPv6

- 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)

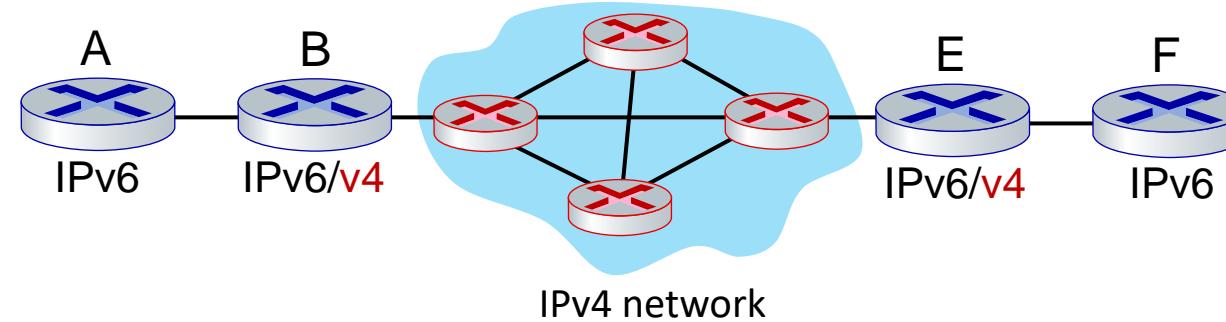


Tunneling and encapsulation

Ethernet connecting
two IPv6 routers:

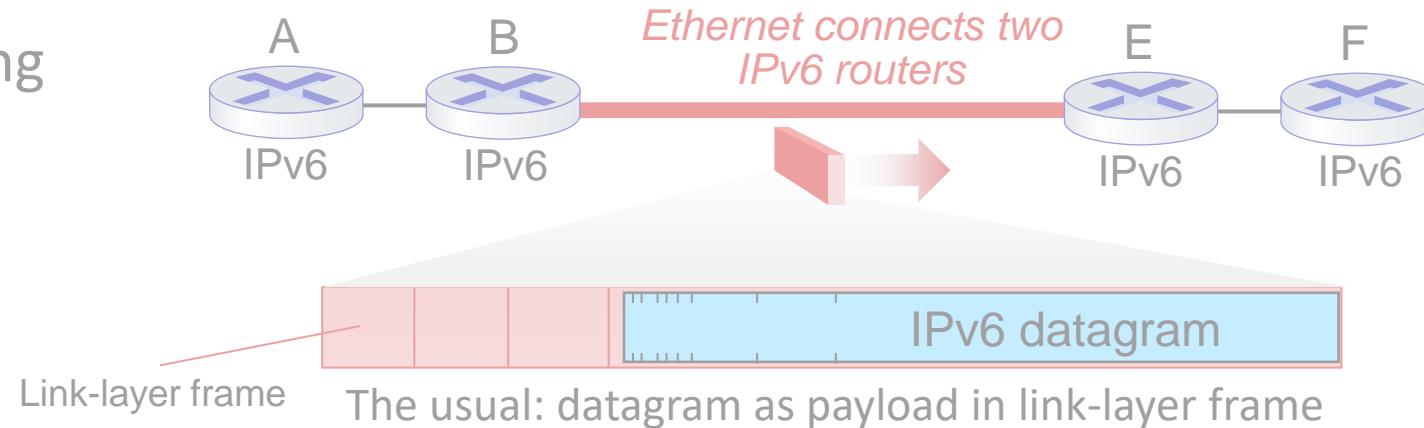


IPv4 network
connecting two
IPv6 routers

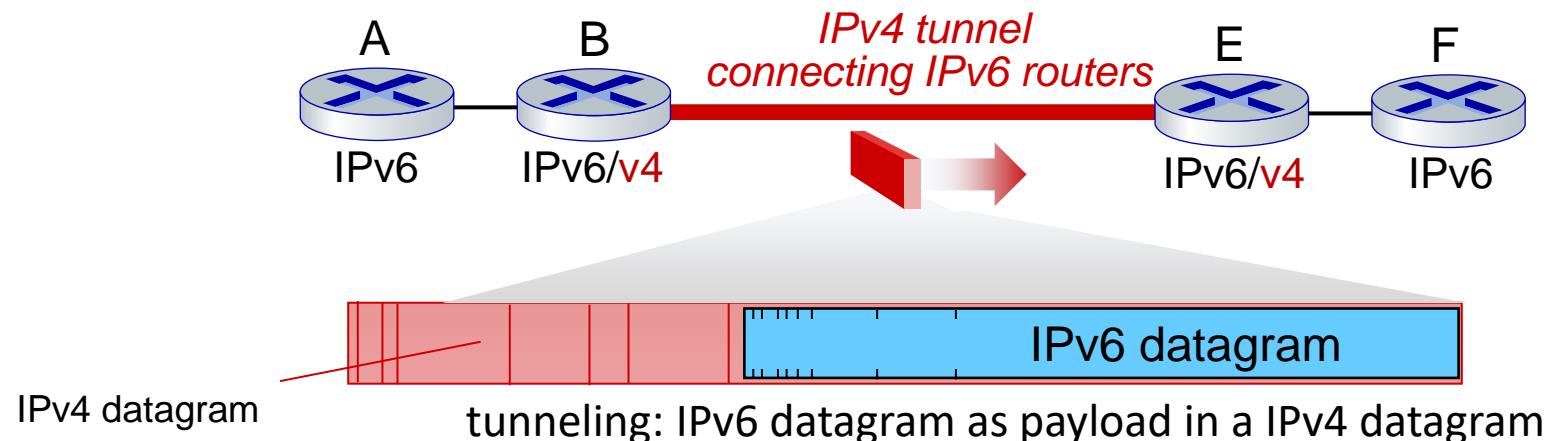


Tunneling and encapsulation

Ethernet connecting
two IPv6 routers:

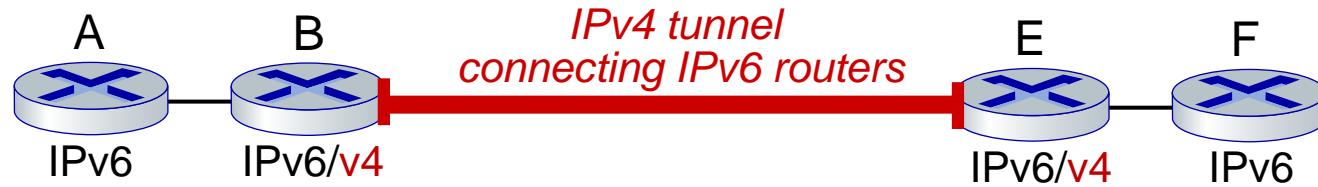


IPv4 tunnel
connecting two
IPv6 routers

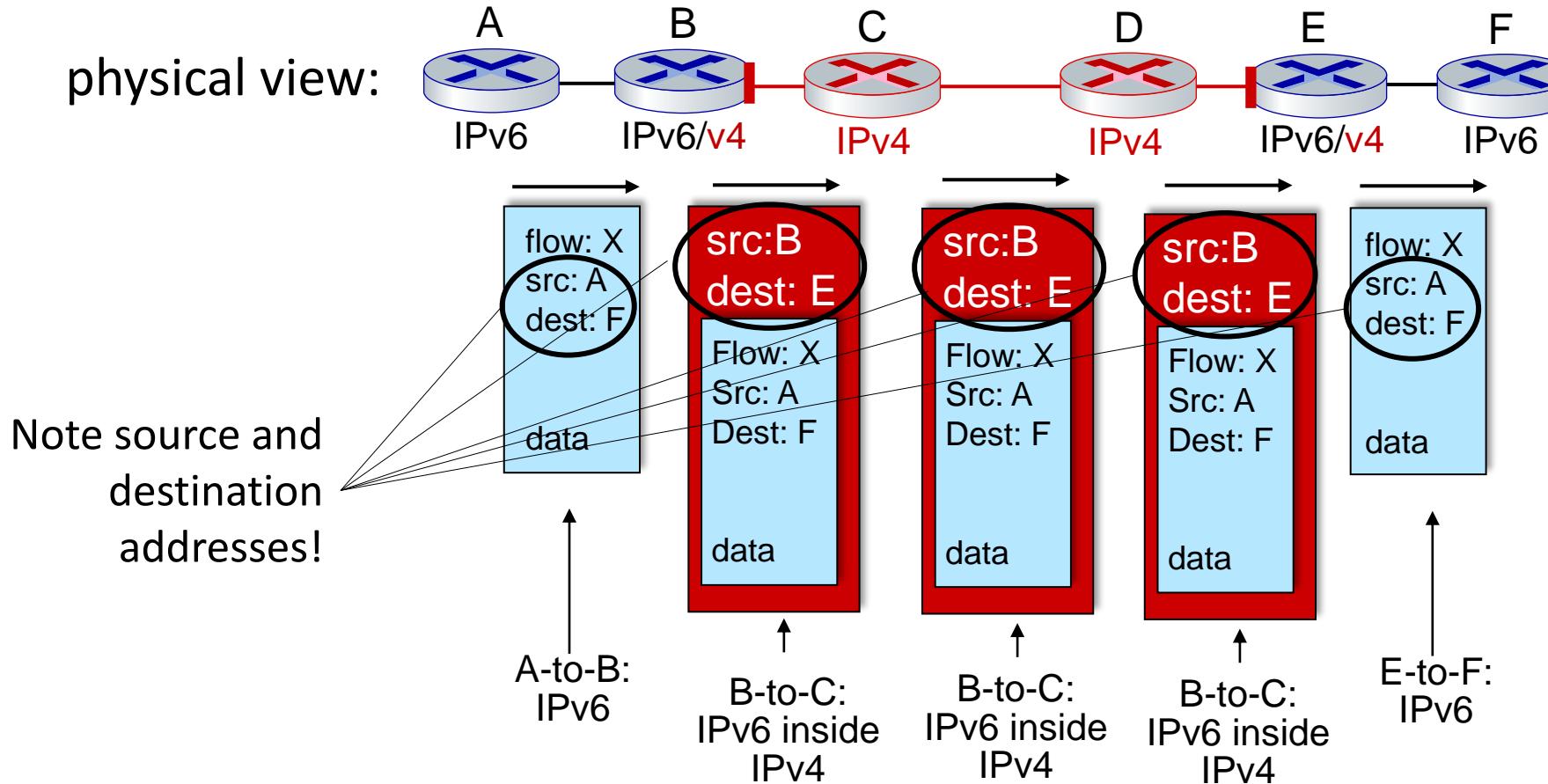


Tunneling

logical view:



physical view:

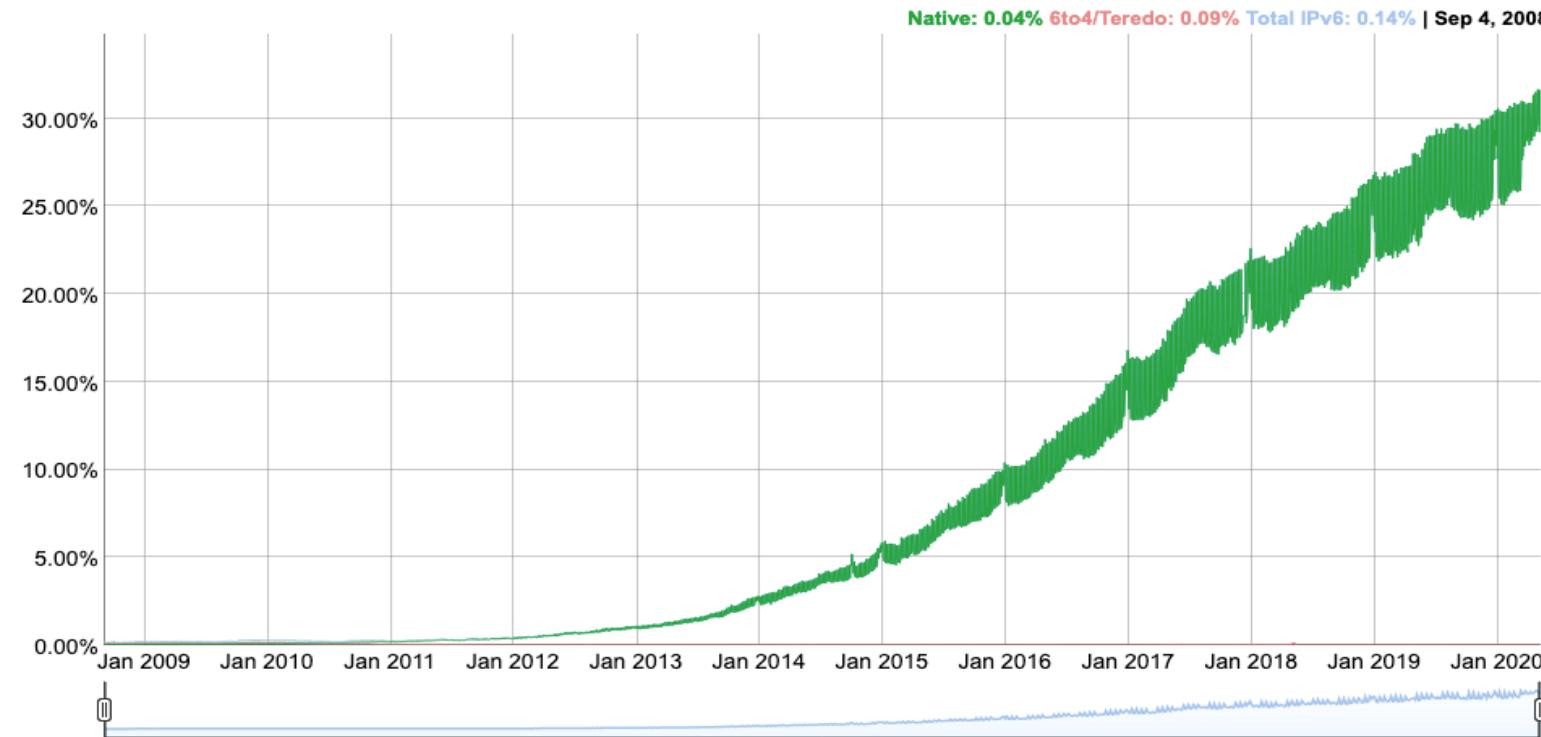


IPv6: adoption

- Google¹: ~ 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>

IPv6: adoption

- Google¹: ~ 30% of clients access services via IPv6
- NIST: 1/3 of all US government domains are IPv6 capable
- Long (long!) time for deployment, use
 - 25 years and counting!
 - think of application-level changes in last 25 years: WWW, social media, streaming media, gaming, telepresence, ...
 - *Why?*

¹ <https://www.google.com/intl/en/ipv6/statistics.html>

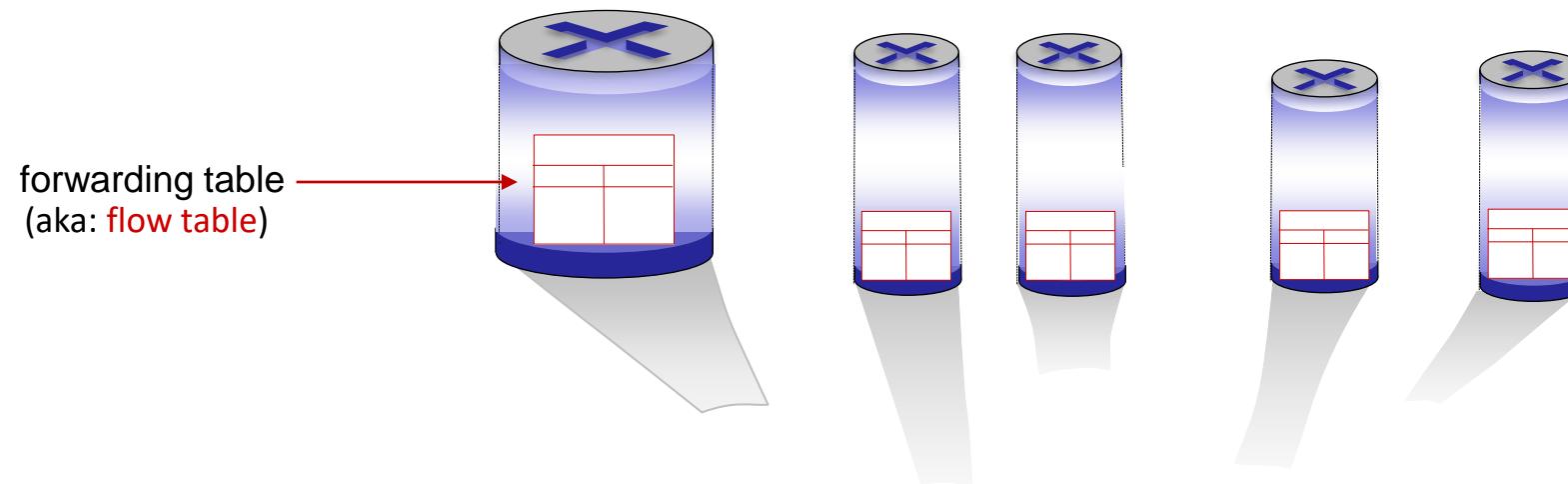
Network layer roadmap

- 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
- Generalized Forwarding, SDN
 - Match+action
 - OpenFlow: match+action in action
- Middleboxes

Generalized forwarding: match plus action

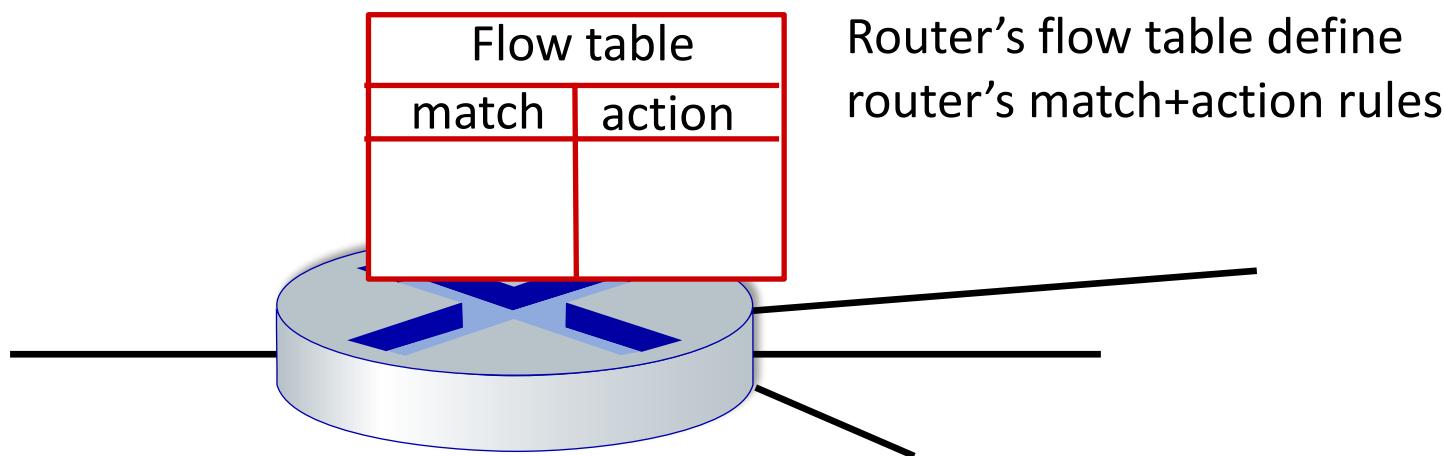
Review: each router contains a **forwarding table** (aka: **flow table**)

- “**match plus action**” abstraction: match bits in arriving packet, take action
 - *destination-based forwarding*: forward based on dest. IP address
 - *generalized forwarding*:
 - many header fields can determine action
 - many action possible: drop/copy/modify/log packet



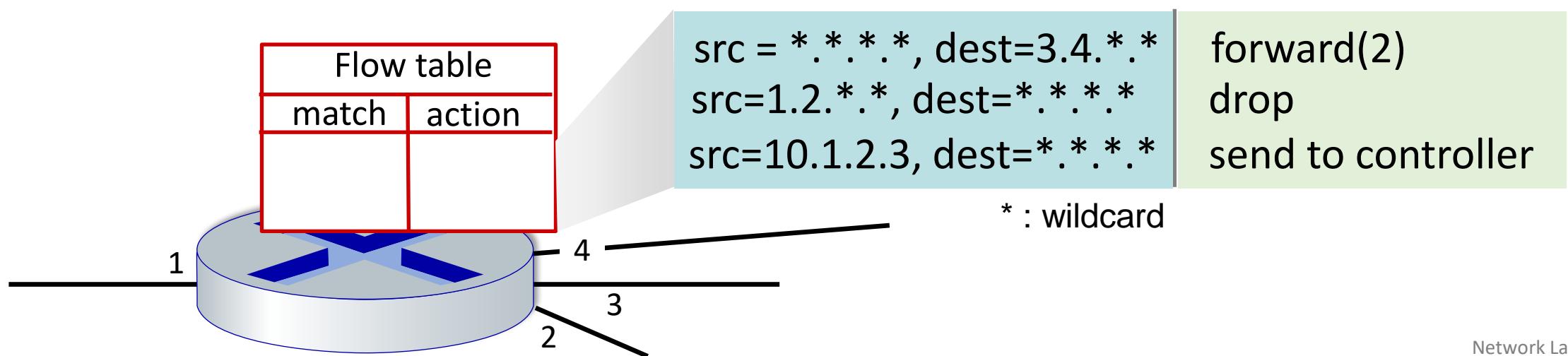
Flow table abstraction

- **flow**: defined by header field values (in link-, network-, transport-layer fields)
- **generalized forwarding**: simple packet-handling rules
 - **match**: pattern values in packet header fields
 - **actions**: for matched packet: drop, forward, modify, matched packet or send matched packet to controller
 - **priority**: disambiguate overlapping patterns
 - **counters**: #bytes and #packets

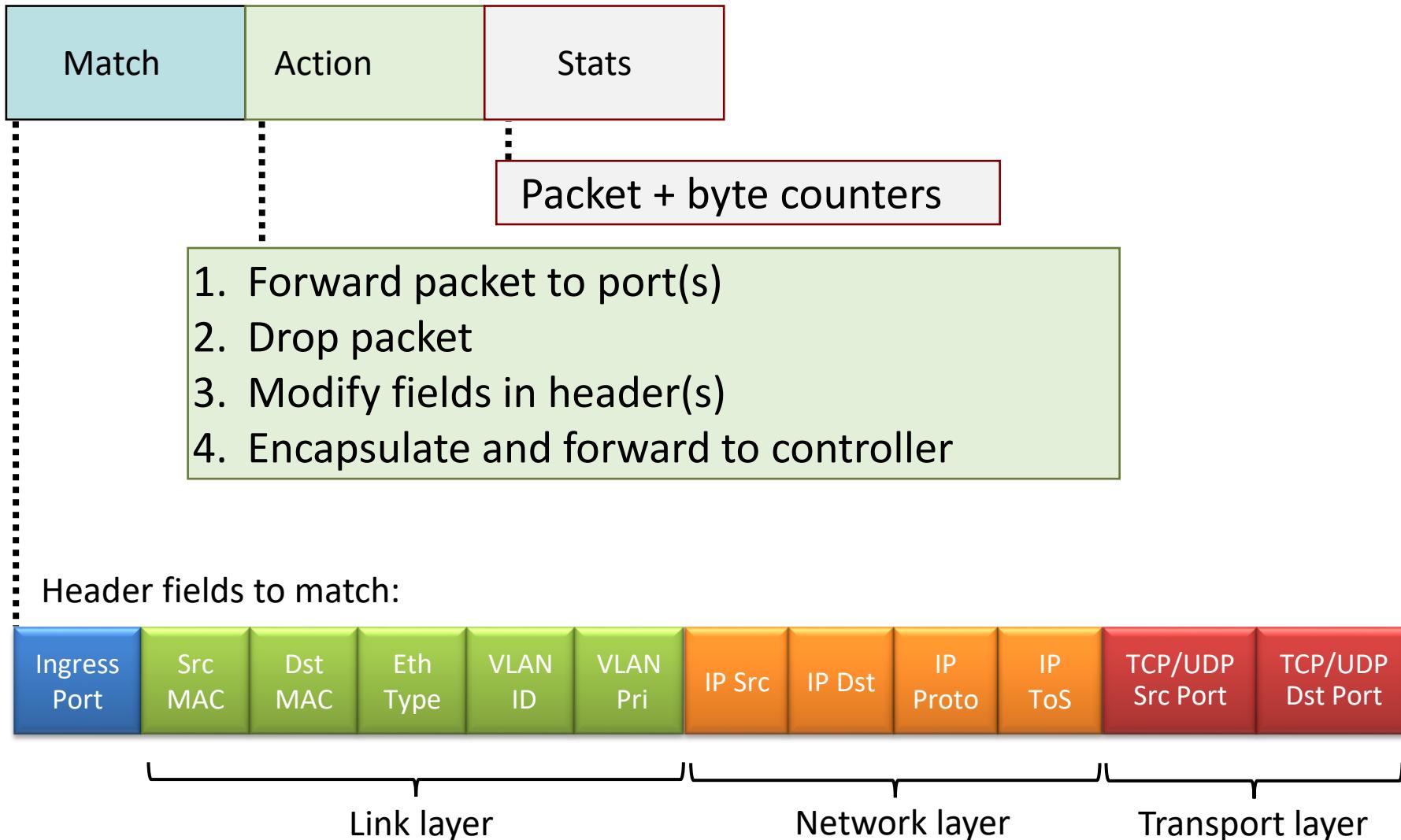


Flow table abstraction

- **flow:** defined by header fields
- **generalized forwarding:** simple packet-handling rules
 - **match:** pattern values in packet header fields
 - **actions:** for matched packet: drop, forward, modify, matched packet or send matched packet to controller
 - **priority:** disambiguate overlapping patterns
 - **counters:** #bytes and #packets



OpenFlow: flow table entries



OpenFlow: examples

Destination-based forwarding:

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	VLAN Pri	IP Src	IP Dst	IP Prot	IP ToS	TCP s-port	TCP d-port	Action
*	*	*	*	*	*	*	51.6.0.8	*	*	*	*	port6

IP datagrams destined to IP address 51.6.0.8 should be forwarded to router output port 6

Firewall:

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	VLAN Pri	IP Src	IP Dst	IP Prot	IP ToS	TCP s-port	TCP d-port	Action
*	*	*	*	*	*	*	*	*	*	*	22	drop

Block (do not forward) all datagrams destined to TCP port 22 (ssh port #)

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	VLAN Pri	IP Src	IP Dst	IP Prot	IP ToS	TCP s-port	TCP d-port	Action
*	*	*	*	*	*	*	128.119.1.1	*	*	*	*	drop

Block (do not forward) all datagrams sent by host 128.119.1.1

OpenFlow: examples

Layer 2 destination-based forwarding:

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	VLAN Pri	IP Src	IP Dst	IP Prot	IP ToS	TCP s-port	TCP d-port	Action
*	*	22:A7:23: 11:E1:02	*	*	*	*	*	*	*	*	*	port3

layer 2 frames with destination MAC address 22:A7:23:11:E1:02 should be forwarded to output port 3

OpenFlow abstraction

- **match+action:** abstraction unifies different kinds of devices

Router

- *match:* longest destination IP prefix
- *action:* forward out a link

Switch

- *match:* destination MAC address
- *action:* forward or flood

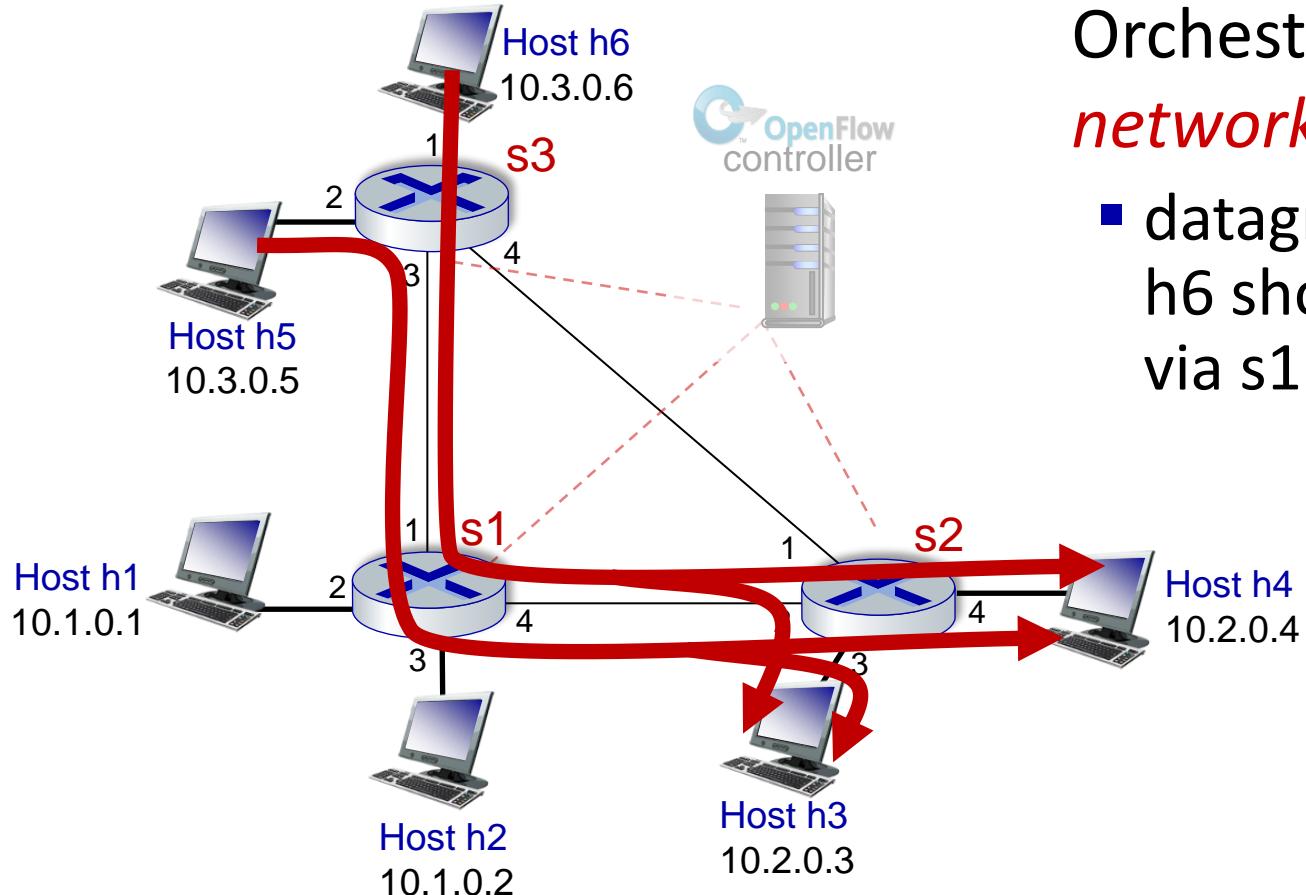
Firewall

- *match:* IP addresses and TCP/UDP port numbers
- *action:* permit or deny

NAT

- *match:* IP address and port
- *action:* rewrite address and port

OpenFlow example

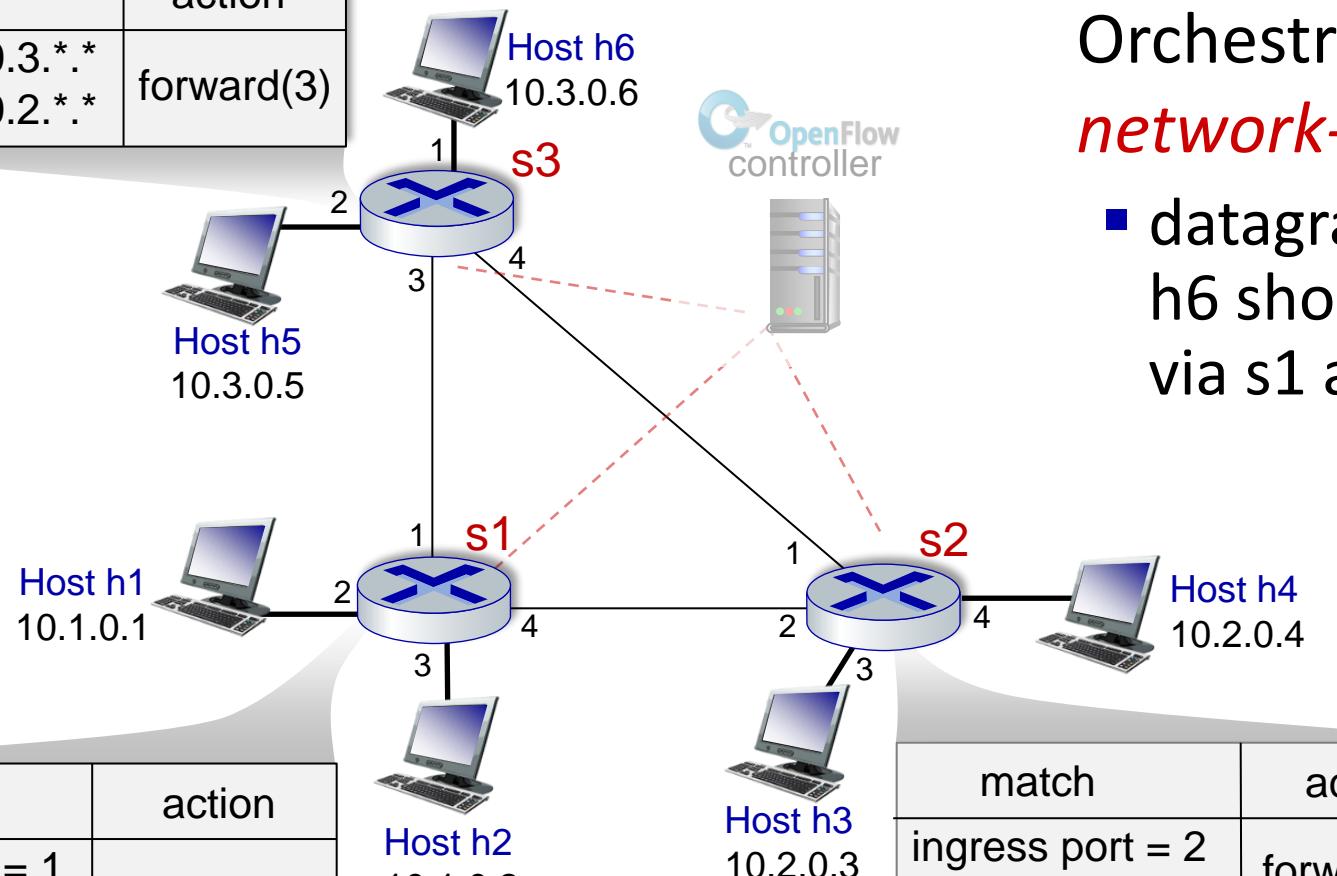


Orchestrated tables can create *network-wide* behavior, e.g.,:

- datagrams from hosts h5 and h6 should be sent to h3 or h4, via s1 and from there to s2

OpenFlow example

match	action
IP Src = 10.3.*.*	
IP Dst = 10.2.*.*	forward(3)



Orchestrated tables can create *network-wide* behavior, e.g.,:

- datagrams from hosts h5 and h6 should be sent to h3 or h4, via s1 and from there to s2

Generalized forwarding: summary

- “**match plus action**” abstraction: match bits in arriving packet header(s) in any layers, take action
 - matching over many fields (link-, network-, transport-layer)
 - local actions: drop, forward, modify, or send matched packet to controller
 - “program” *network-wide* behaviors
- simple form of “network programmability”
 - programmable, per-packet “processing”
 - *historical roots*: active networking
 - *today*: more generalized programming:
P4 (see p4.org).

Network layer 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
- Generalized Forwarding, SDN
 - Match+action
 - OpenFlow: match+action in action
- **Middleboxes**
 - middlebox functions
 - evolution, architectural principles of the Internet

Middleboxes

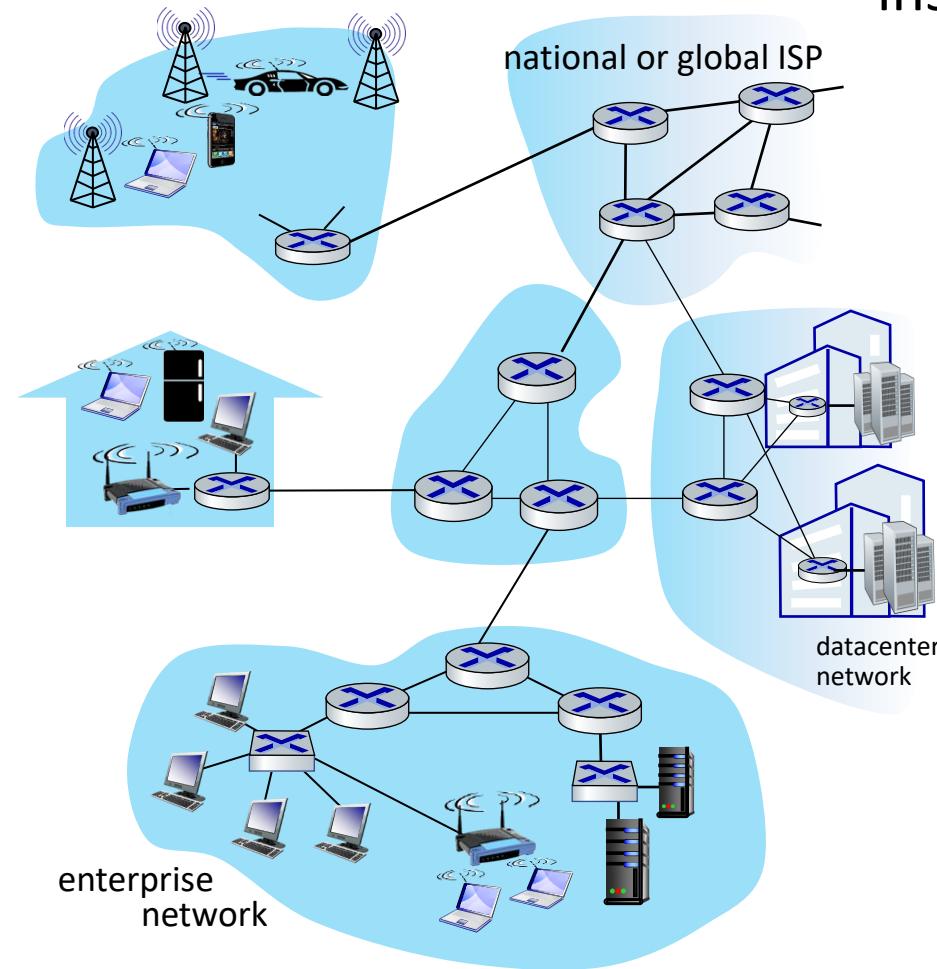
Middlebox (RFC 3234)

“any intermediary box performing functions apart from normal, standard functions of an IP router on the data path between a source host and destination host”

Middleboxes everywhere!

NAT: home,
cellular,
institutional

Application-specific: service providers, institutional, CDN



Firewalls, IDS: corporate, institutional, service providers, ISPs

Load balancers: corporate, service provider, data center, mobile nets

Caches: service provider, mobile, CDNs

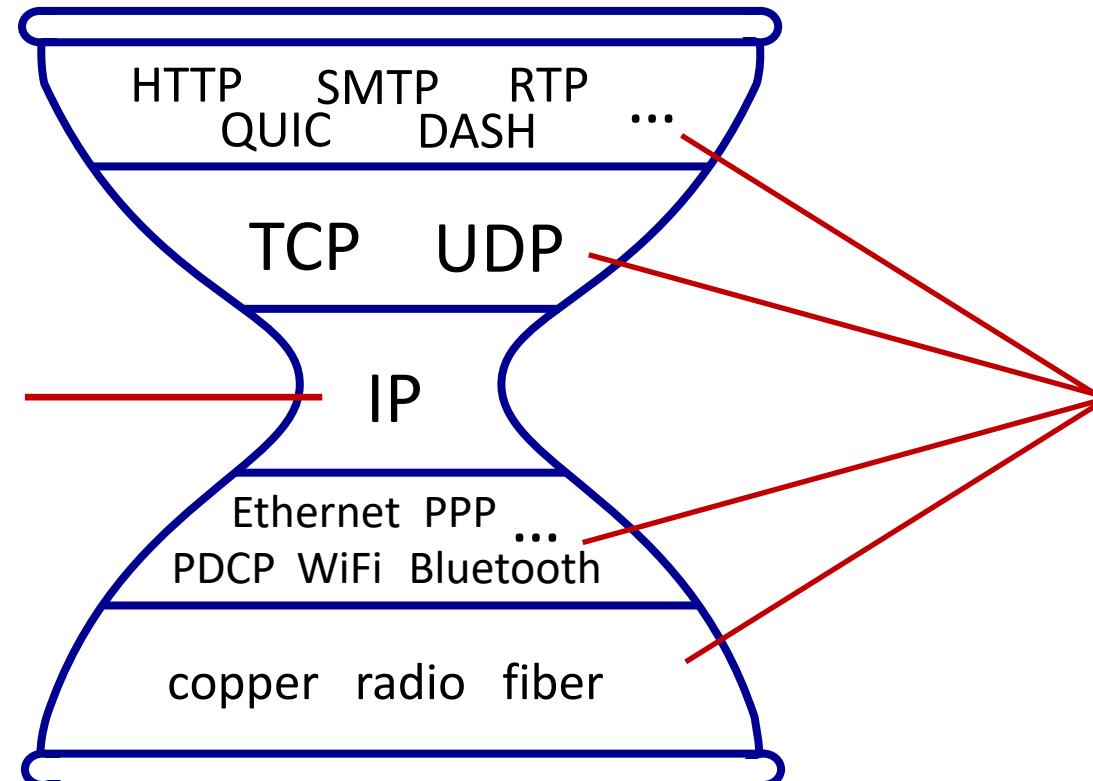
Middleboxes

- initially: proprietary (closed) hardware solutions
- move towards “whitebox” hardware implementing open API
 - move away from proprietary hardware solutions
 - programmable local actions via match+action
 - move towards innovation/differentiation in software
- SDN: (logically) centralized control and configuration management often in private/public cloud
- network functions virtualization (NFV): programmable services over white box networking, computation, storage

The IP hourglass

Internet's "thin waist":

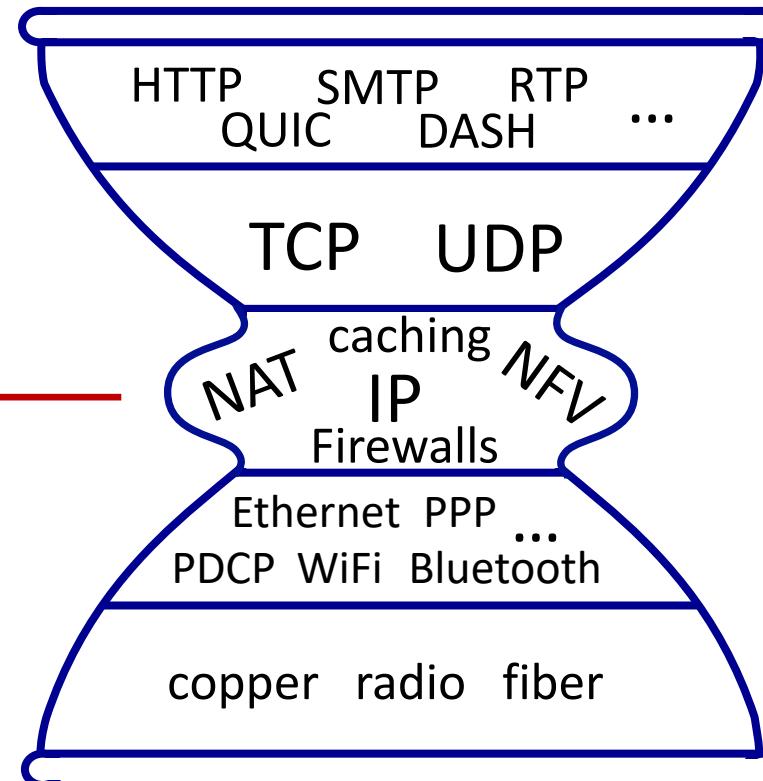
- *one* network layer protocol: IP
- *must* be implemented by every (billions) of Internet-connected devices



many protocols in physical, link, transport, and application layers

The IP hourglass, at middle age

Internet's middle age
“love handles”?
■ middleboxes,
operating inside the
network



Architectural Principles of the Internet

RFC 1958

“Many members of the Internet community would argue that there is no architecture, but only a tradition, which was not written down for the first 25 years (or at least not by the IAB). However, in very general terms, the community believes that

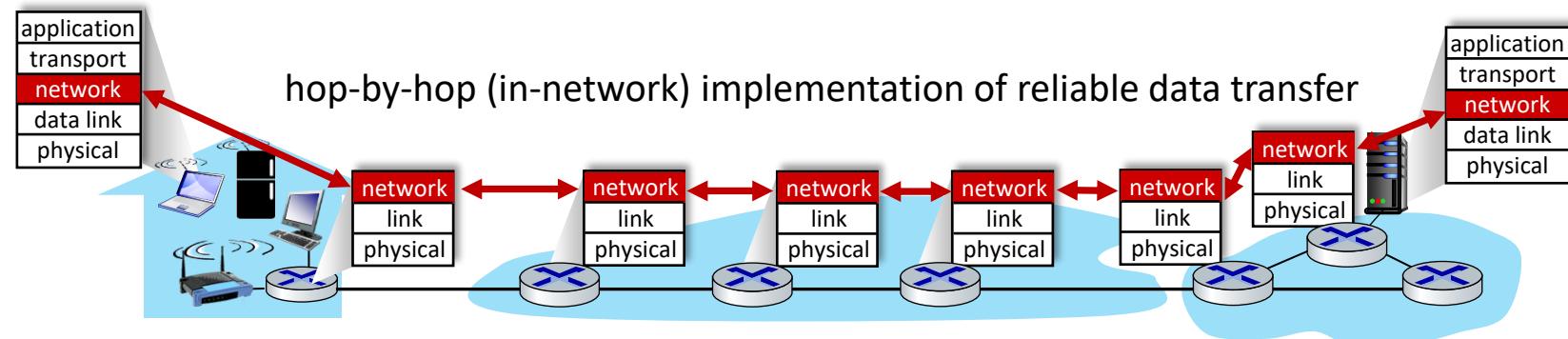
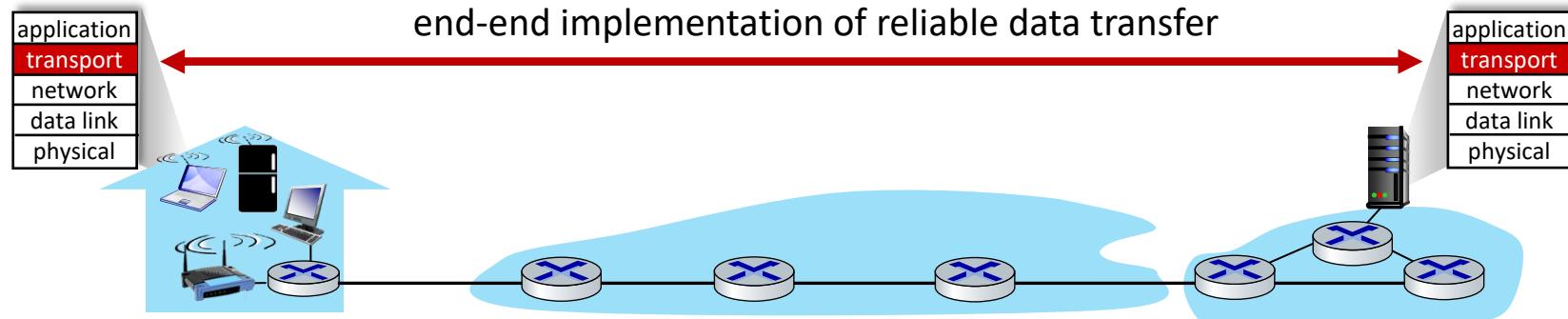
the goal is connectivity, the tool is the Internet Protocol, and the intelligence is end to end rather than hidden in the network.”

Three cornerstone beliefs:

- simple connectivity
- IP protocol: that narrow waist
- intelligence, complexity at network edge

The end-end argument

- some network functionality (e.g., reliable data transfer, congestion) can be implemented in **network**, or at **network edge**



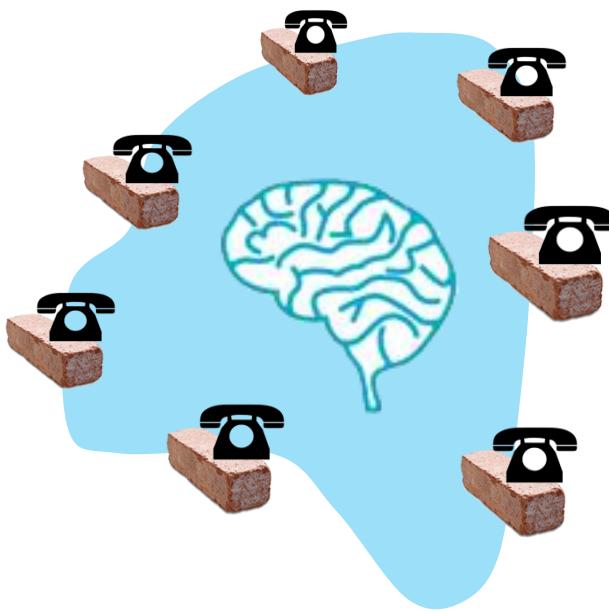
The end-end argument

- some network functionality (e.g., reliable data transfer, congestion) can be implemented in network, or at network edge

“The function in question can completely and correctly be implemented only with the knowledge and help of the application standing at the end points of the communication system. Therefore, providing that questioned function as a feature of the communication system itself is not possible. (Sometimes an incomplete version of the function provided by the communication system may be useful as a performance enhancement.)

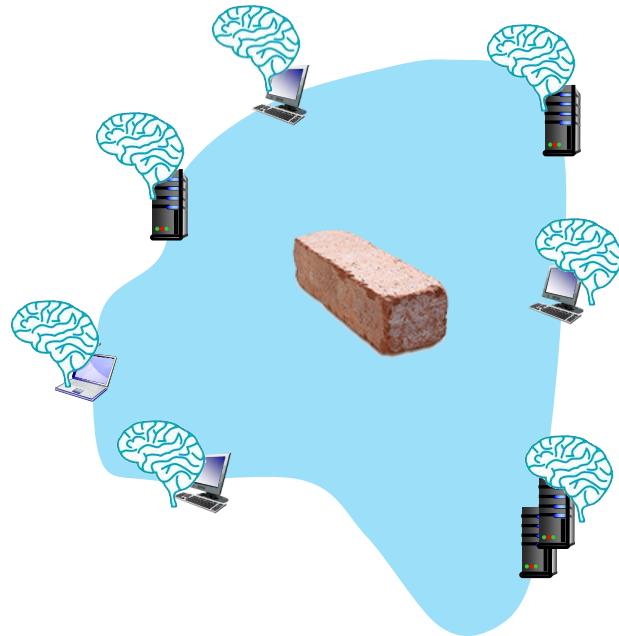
We call this line of reasoning against low-level function implementation the “end-to-end argument.”

Where's the intelligence?



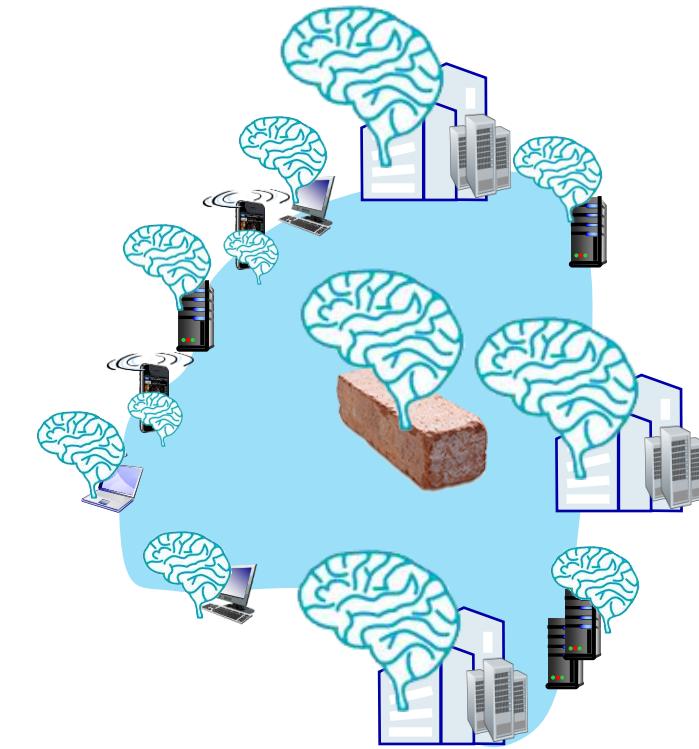
20th century phone net:

- intelligence/computing at network switches



Internet (pre-2005)

- intelligence, computing at edge



Internet (post-2005)

- programmable network devices
- intelligence, computing, massive application-level infrastructure at edge

Network Layer: Part 2

Slides are adapted from the companion website of the book *Computer Networking: A Top-Down Approach* by Jim Kurose, Keith Ross

Network layer: roadmap

- introduction
- routing protocols
 - link state
 - distance vector
- intra-ISP routing: OSPF
- routing among ISPs: BGP
- SDN control plane
- Internet Control Message Protocol

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

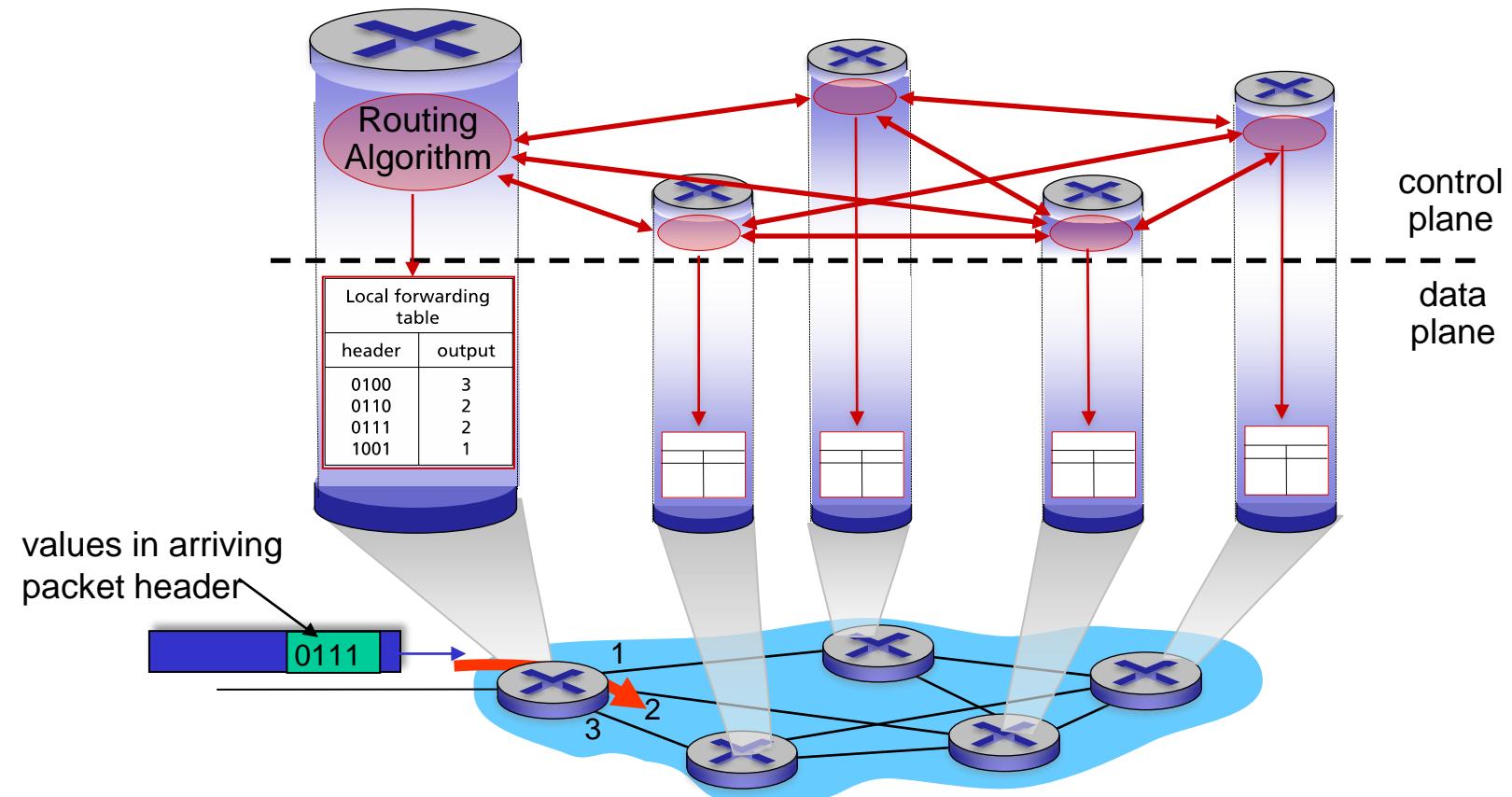
control plane

Two approaches to structuring network control plane:

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

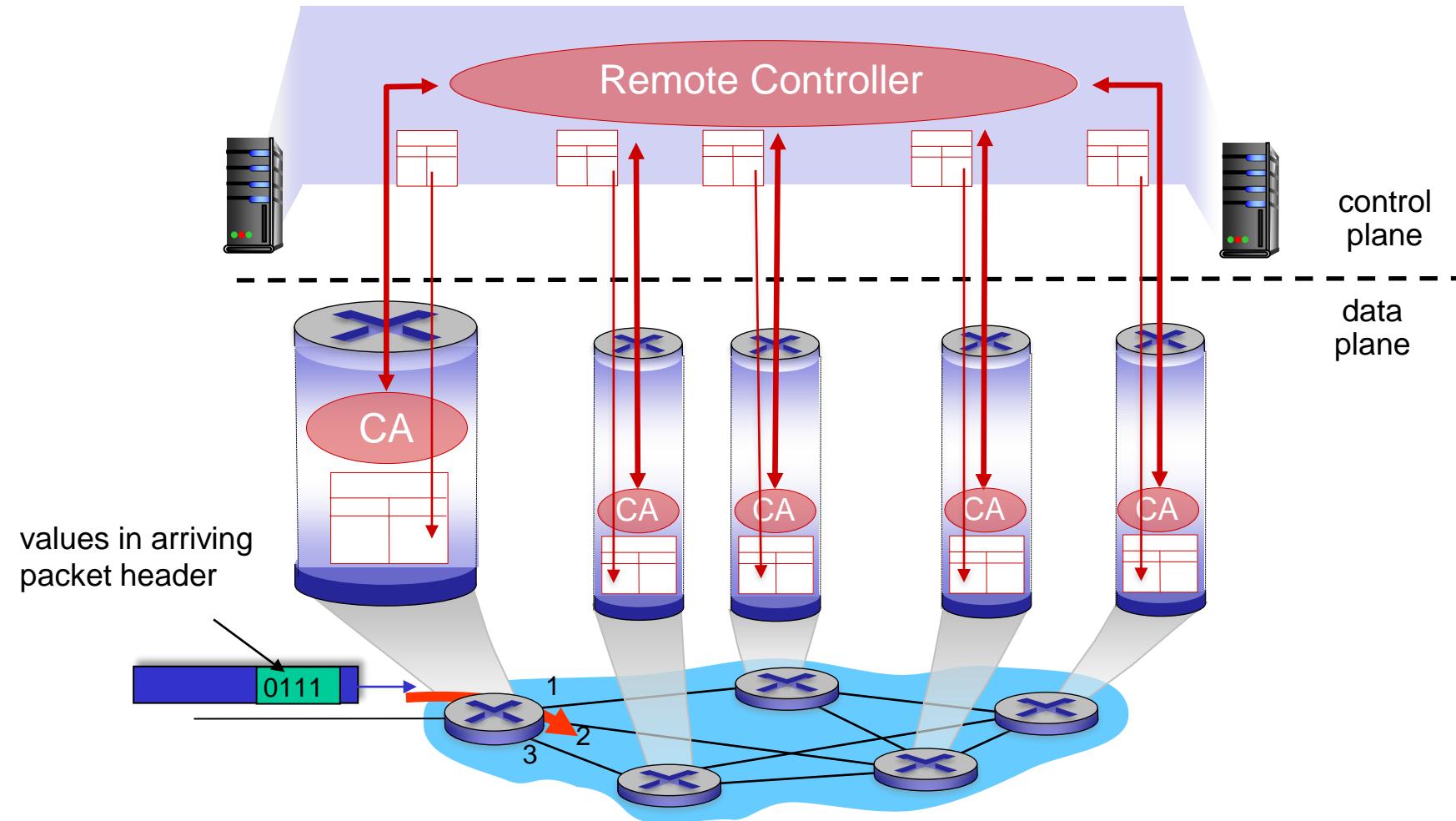
Per-router control plane

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



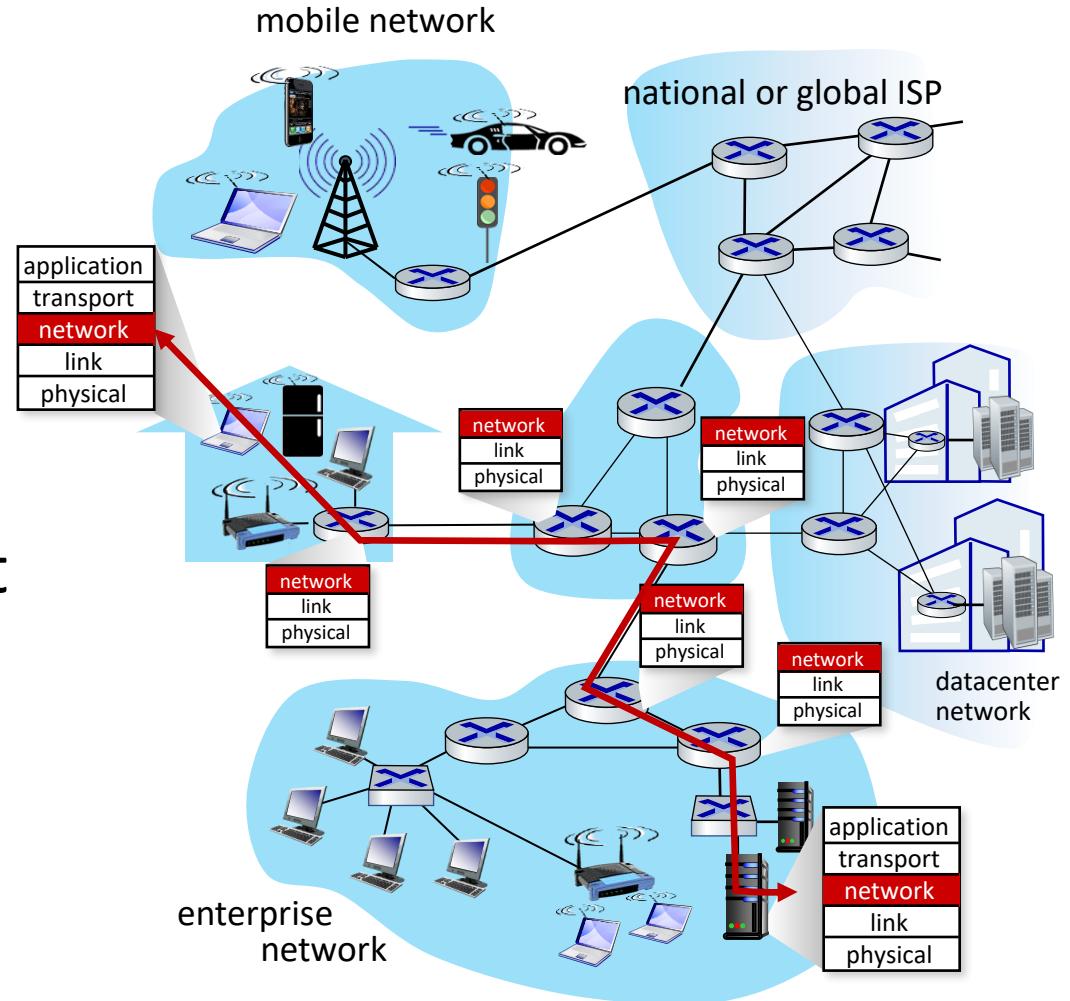
Network layer: roadmap

- introduction
- **routing protocols**
 - link state
 - distance vector
- intra-ISP routing: OSPF
- routing among ISPs: BGP
- SDN control plane
- Internet Control Message Protocol

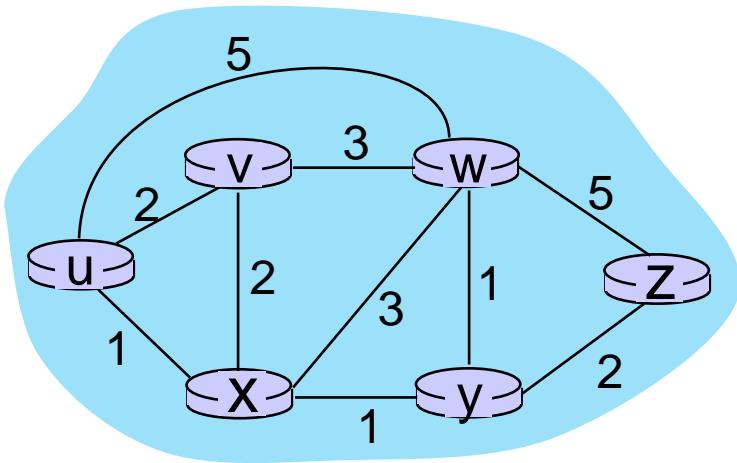
Routing protocols

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”



Graph abstraction: link costs



graph: $G = (N, E)$

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

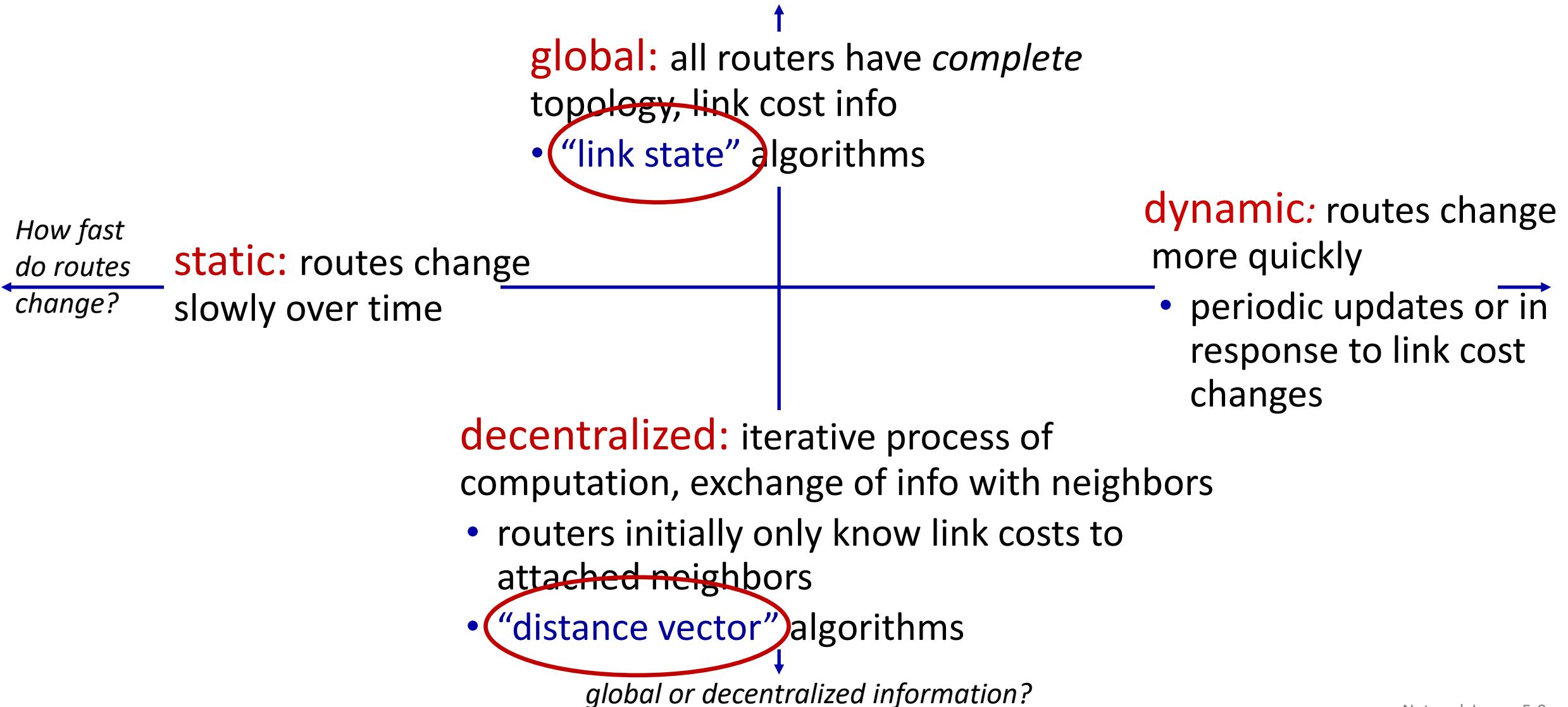
E : set of links = { $(u,v), (u,x), (v,x), (v,w), (x,w), (x,y), (w,y), (w,z), (y,z)$ }

$c_{a,b}$: 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: roadmap

- introduction
- routing protocols
 - link state
 - distance vector
- intra-ISP routing: OSPF
- routing among ISPs: BGP
- SDN control plane
- Internet Control Message Protocol

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

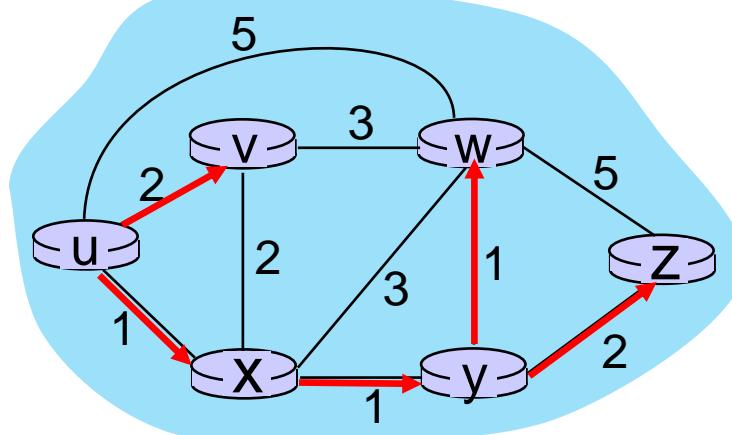
- $c_{x,y}$: direct link cost from node x to y ; $= \infty$ if not direct neighbors
- $D(v)$: *current* estimate of cost of least-cost-path from source to destination v
- $p(v)$: predecessor node along path from source to v
- N' : set of nodes whose least-cost-path *definitively* known

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

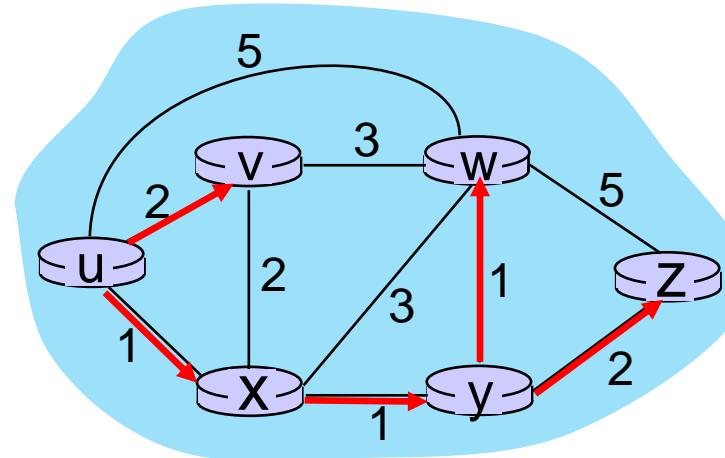
Step	N'	$D(v), p(v)$	$D(w), p(w)$	$D(x), p(x)$	$D(y), p(y)$	$D(z), p(z)$
0	u	2, u	5, u	1, u	∞	∞
1	u, x	2, u	4, x	2, x	∞	∞
2	u, x, y	2, u	3, y	∞	4, y	∞
3	u, x, y, v	∞	3, y	∞	4, y	∞
4	u, x, y, v, w	∞	∞	∞	4, y	∞
5	u, x, y, v, w, z	∞	∞	∞	∞	∞



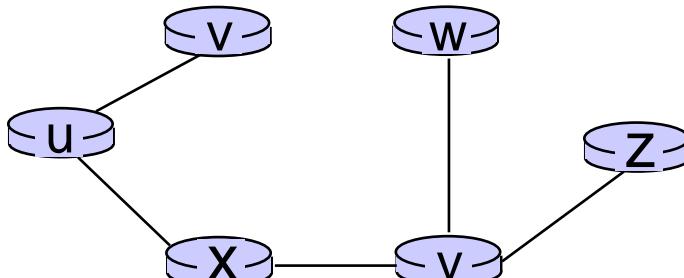
Initialization (step 0): For all a : if a adjacent to u then $D(a) = c_{u,a}$

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)
x	(u,x)

route from u to v directly

route from u to all other destinations via x

Dijkstra's algorithm: discussion

algorithm complexity: n nodes

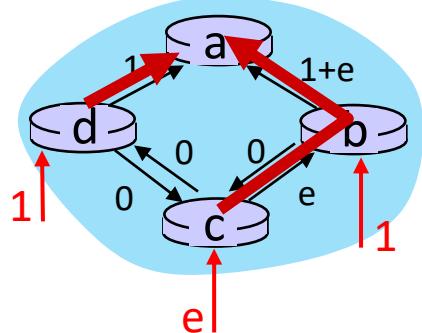
- each of n iteration: need to check all nodes, w , not in N
- $n(n+1)/2$ comparisons: $O(n^2)$ complexity
- more efficient implementations possible: $O(n \log n)$

message complexity:

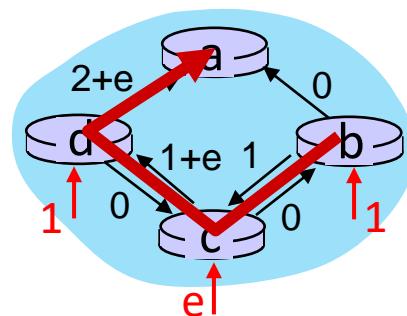
- each router must *broadcast* its link state information to other n routers
- efficient broadcast algorithms: $O(n)$ link crossings to disseminate a broadcast message from one source
- each router's message crosses $O(n)$ links: overall message complexity: $O(n^2)$

Dijkstra's algorithm: oscillations possible

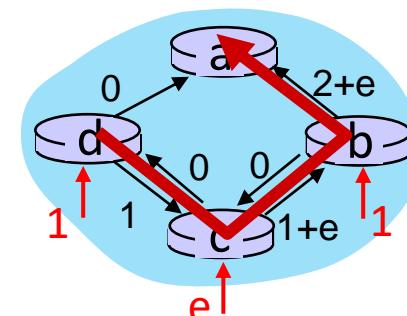
- when link costs depend on traffic volume, **route oscillations** possible
- sample scenario:
 - routing to destination a, traffic entering at d, c, e with rates 1, e (<1), 1
 - link costs are directional, and volume-dependent



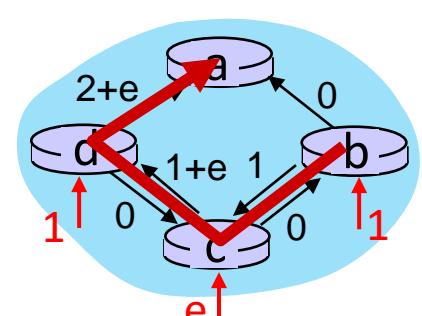
initially



given these costs,
find new routing....
resulting in new costs



given these costs,
find new routing....
resulting in new costs



given these costs,
find new routing....
resulting in new costs

Network layer: roadmap

- introduction
- routing protocols
 - link state
 - **distance vector**
- intra-ISP routing: OSPF
- routing among ISPs: BGP
- SDN control plane
- Internet Control Message Protocol

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:

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

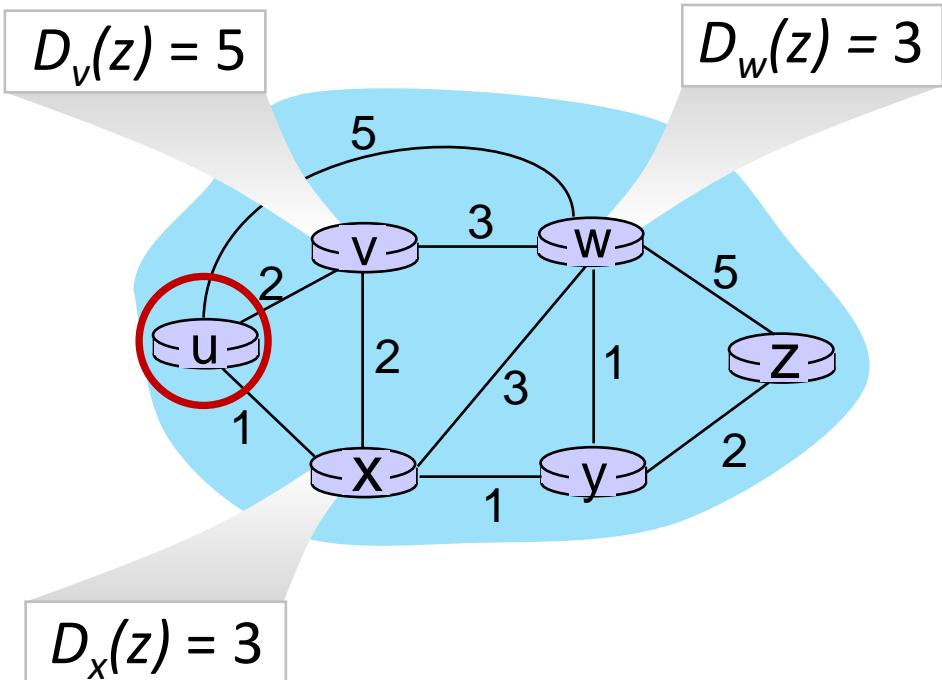
\min taken over all neighbors v of x

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["wait for (change in local link cost or msg from neighbor)"] --> B["recompute DV estimates using DV received from neighbor"]; B --> C["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!

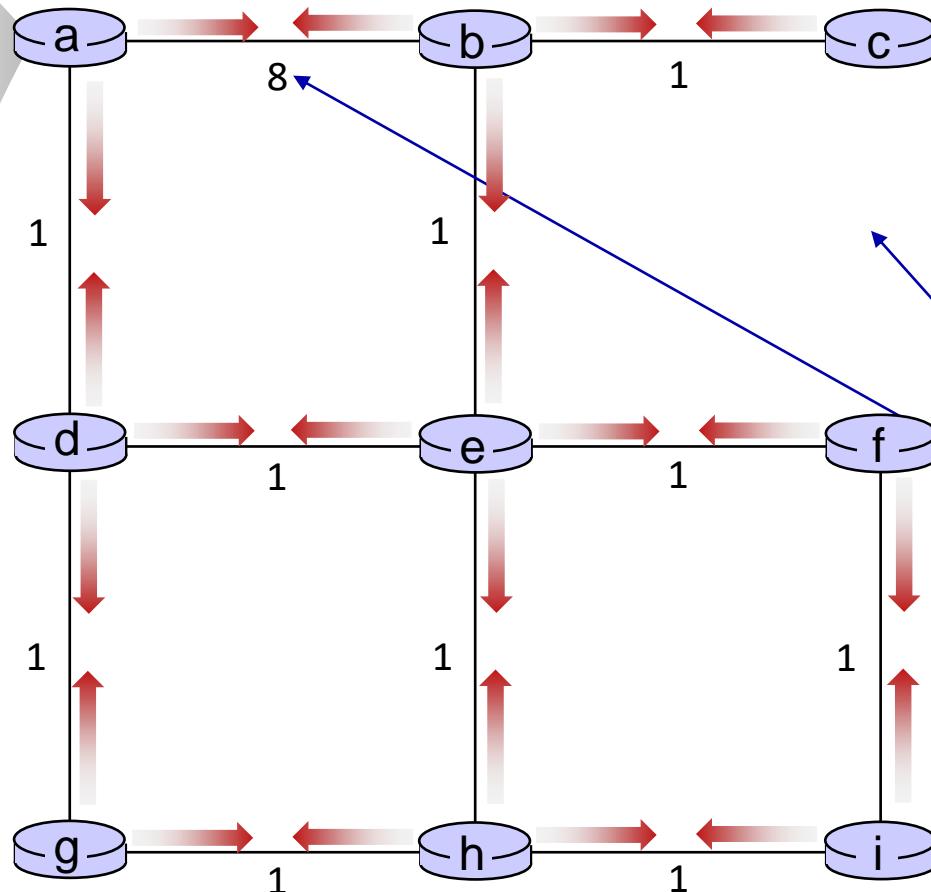
# Distance vector: example



$t=0$

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

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



- A few asymmetries:
  - missing link
  - larger cost

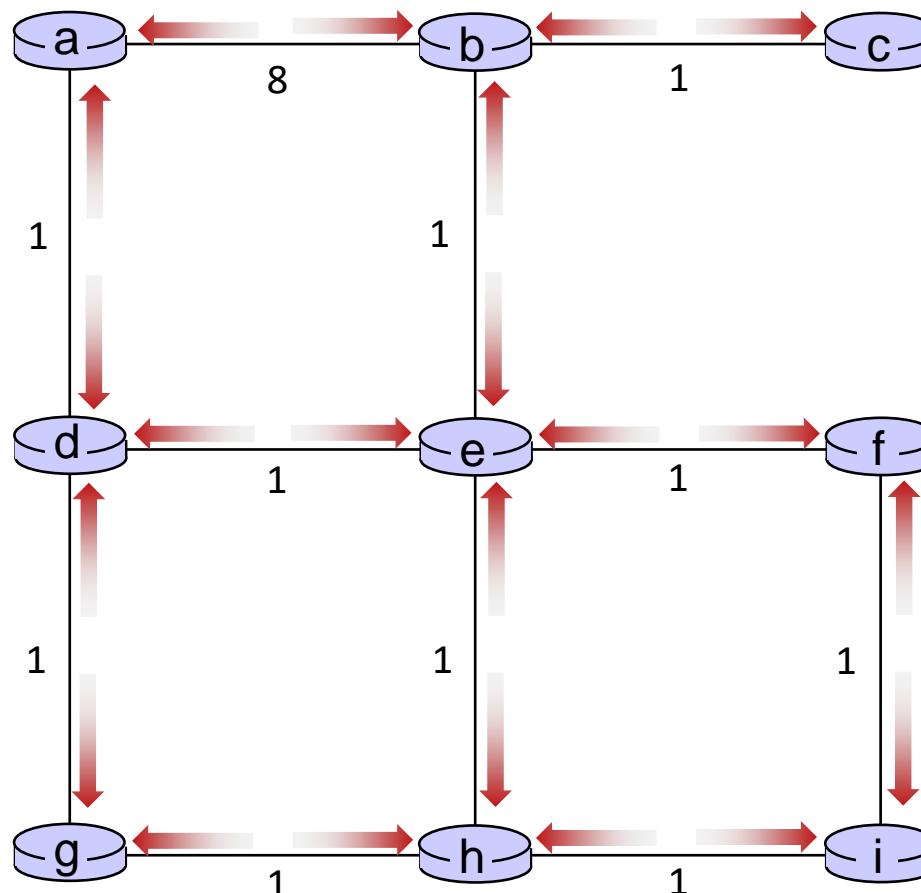
# Distance vector example: iteration



$t=1$

All nodes:

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



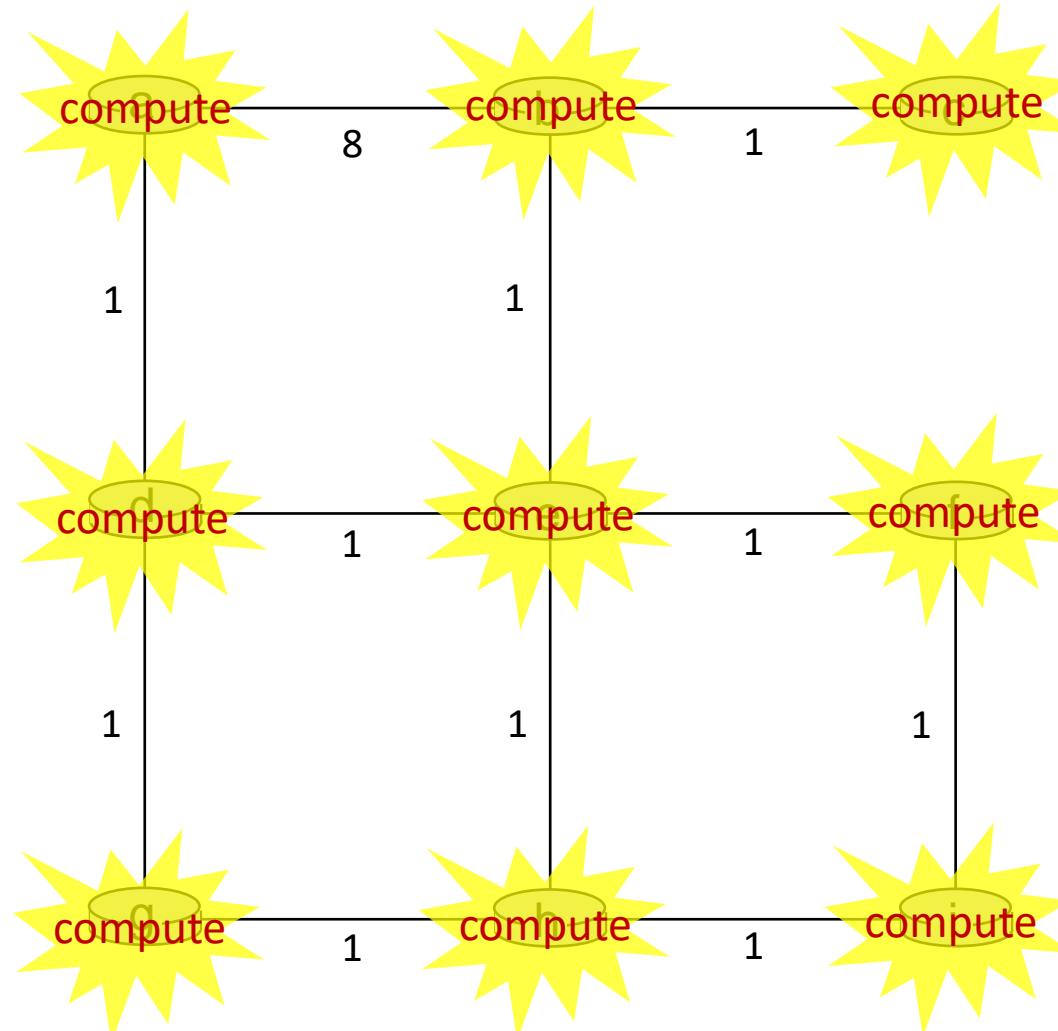
# Distance vector example: iteration



$t=1$

All nodes:

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



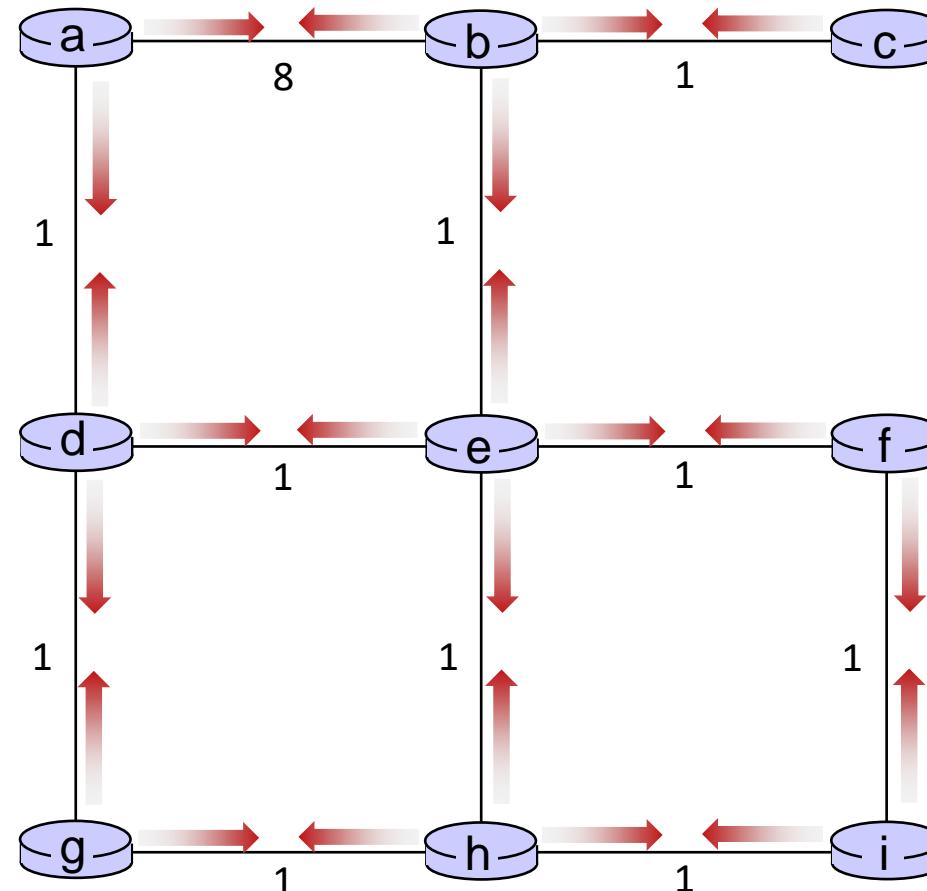
# Distance vector example: iteration



$t=1$

All nodes:

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



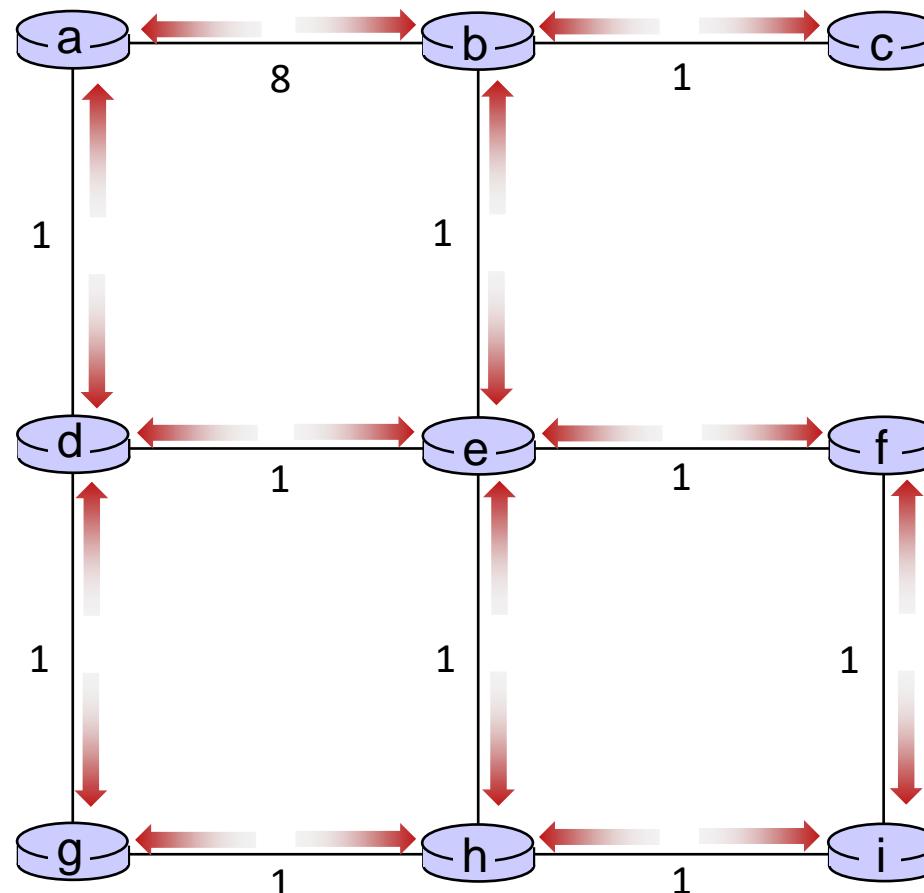
# Distance vector example: iteration



$t=2$

All nodes:

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



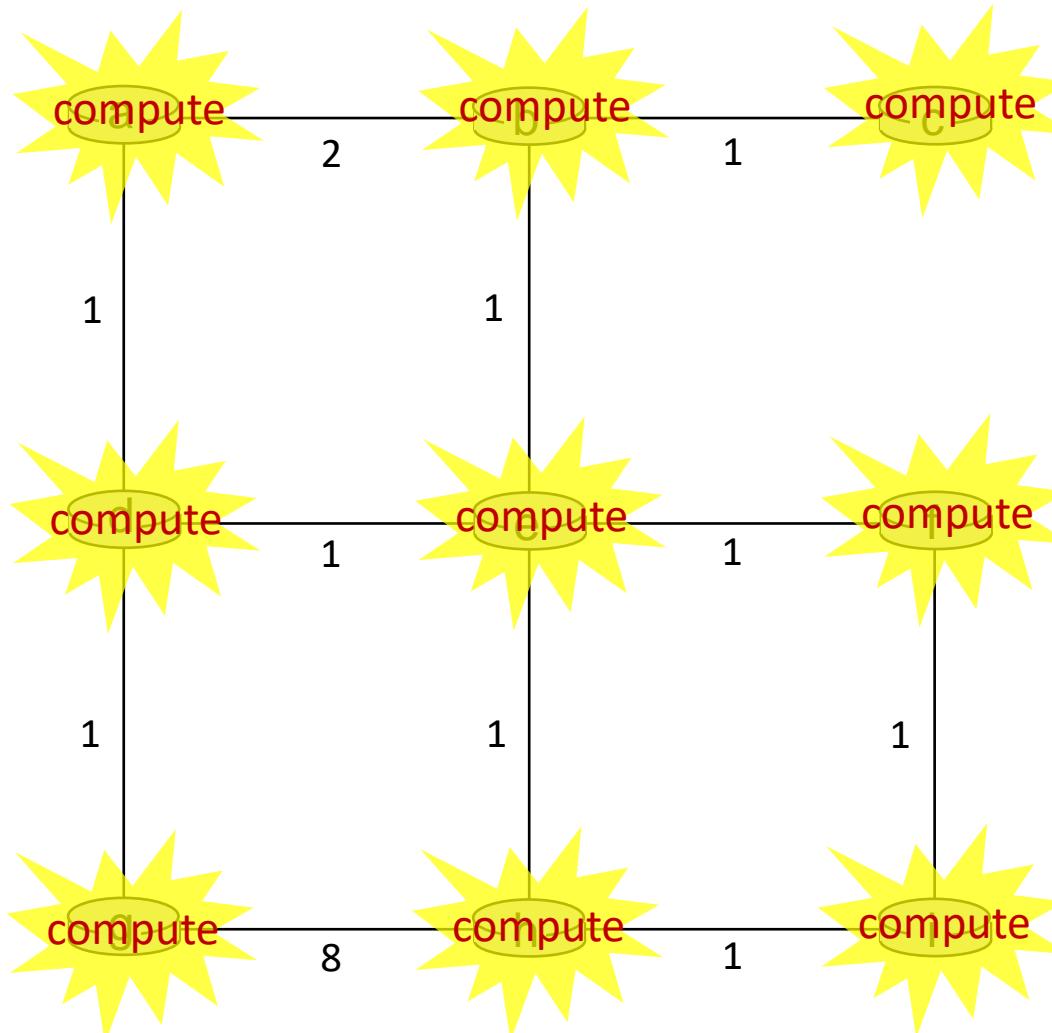
# Distance vector example: iteration



$t=2$

All nodes:

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



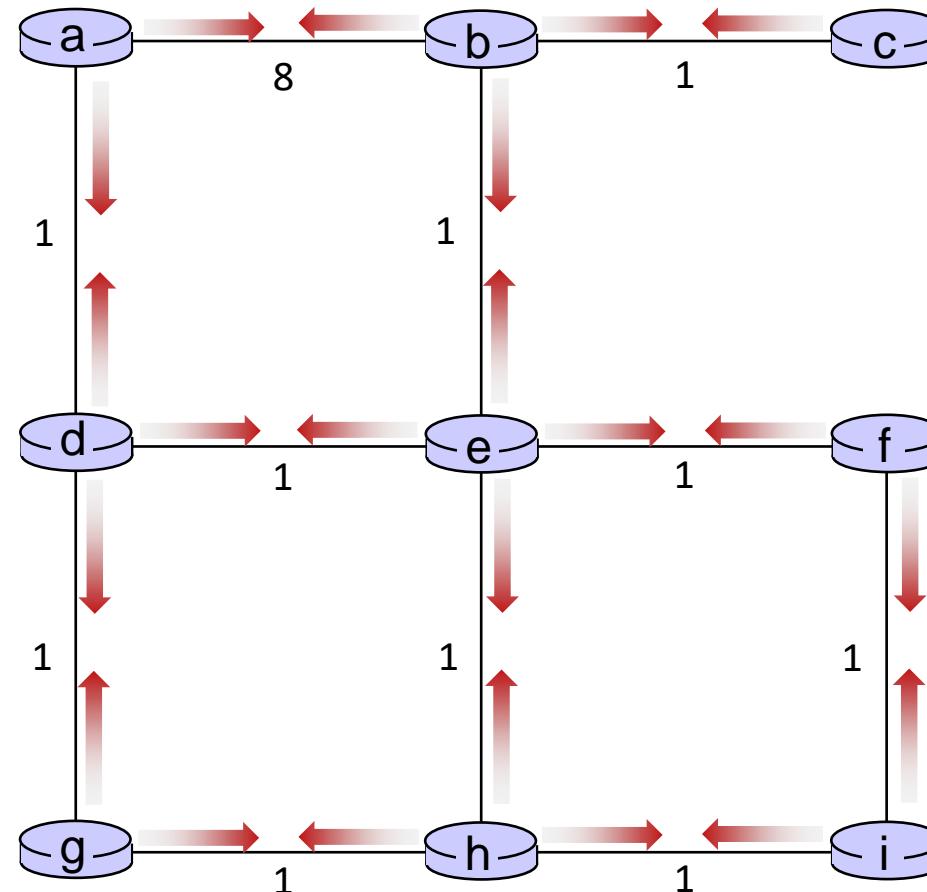
# Distance vector example: iteration



$t=2$

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

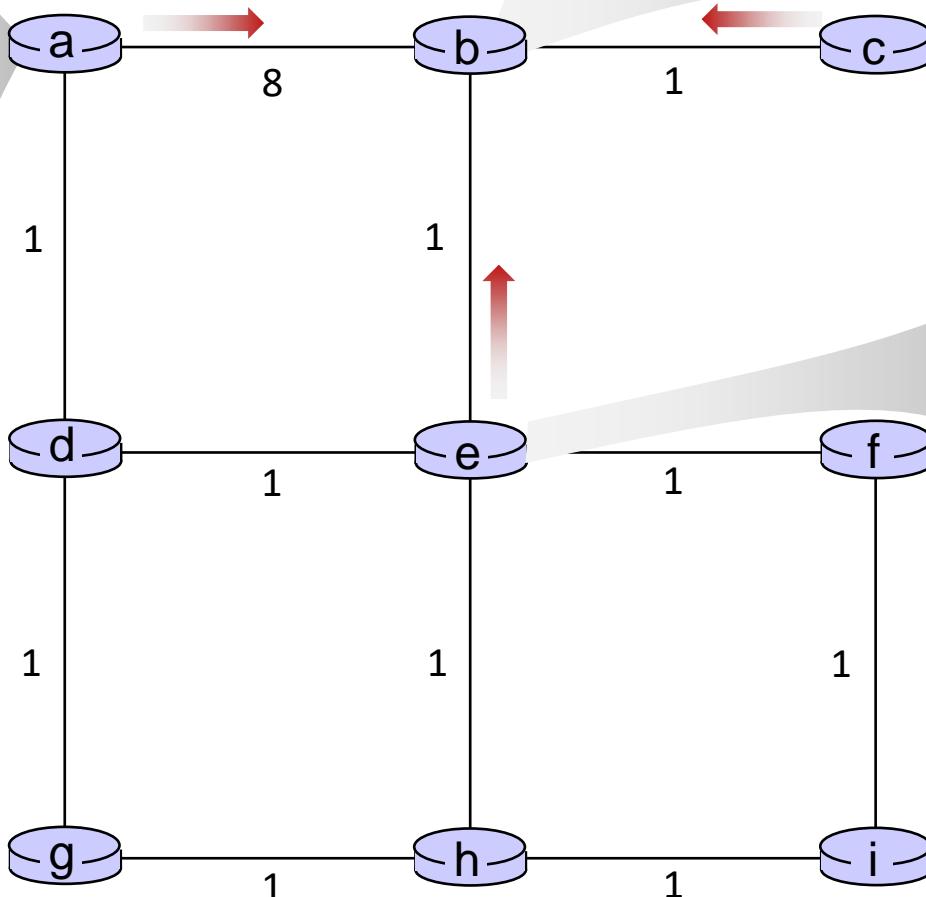
# Distance vector example: t=1



**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

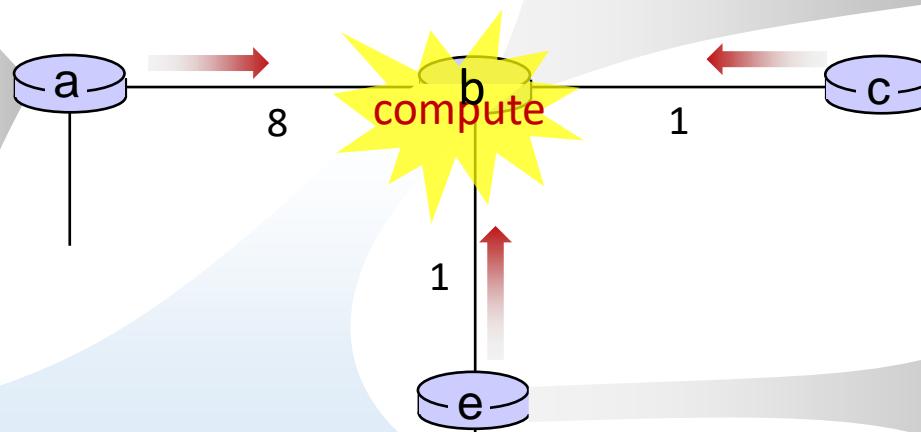


**t=1**

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

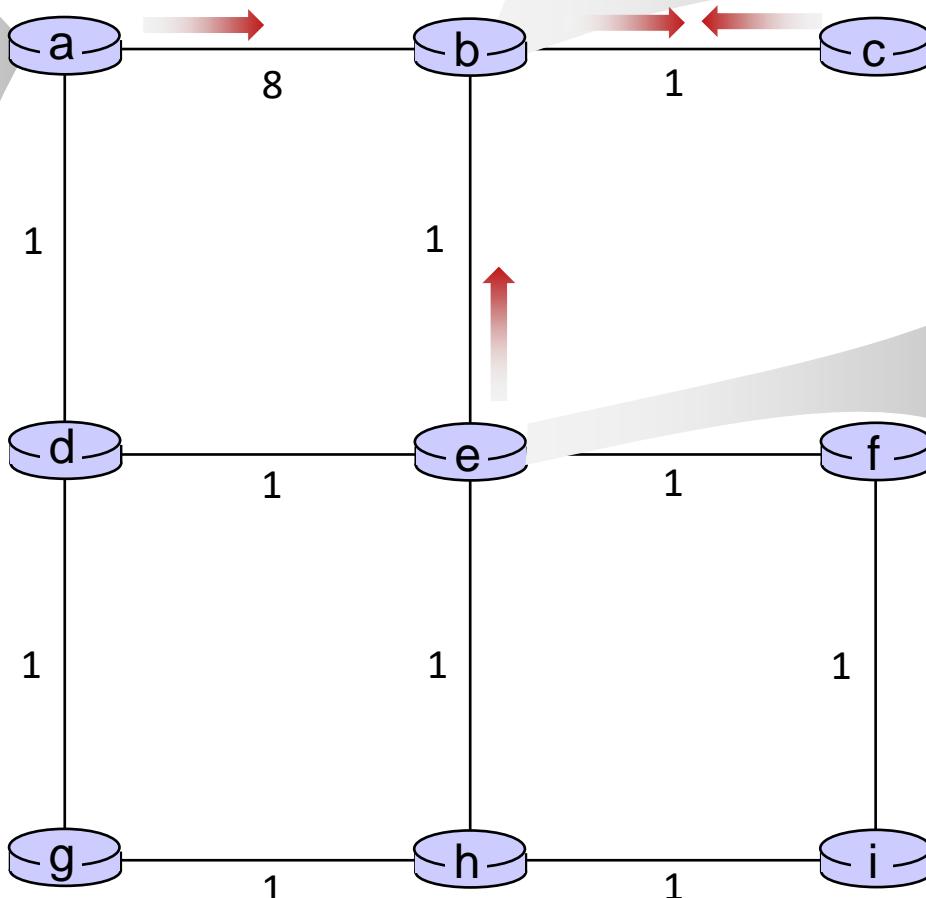
# Distance vector example: t=1



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

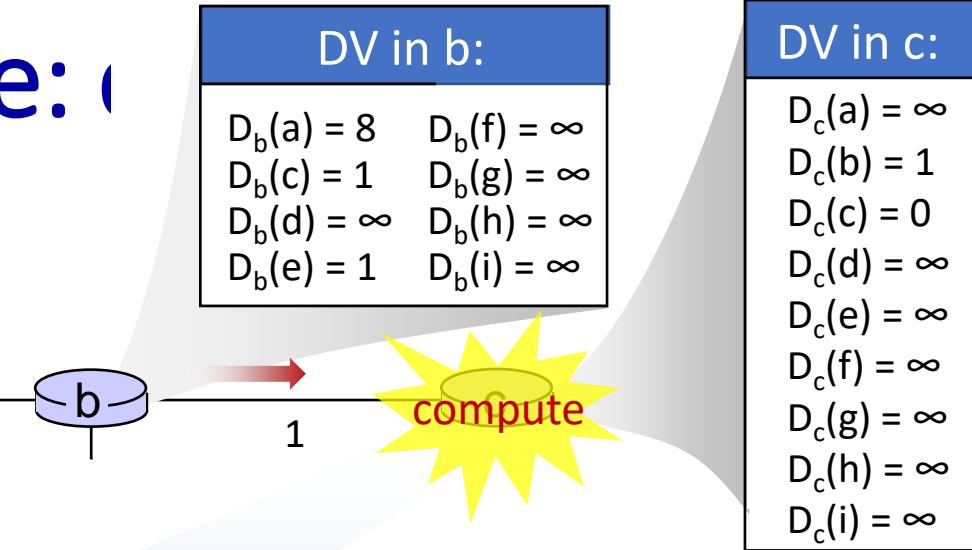
# Distance vector example: (t=1)



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:

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

# Distance vector example: t=1

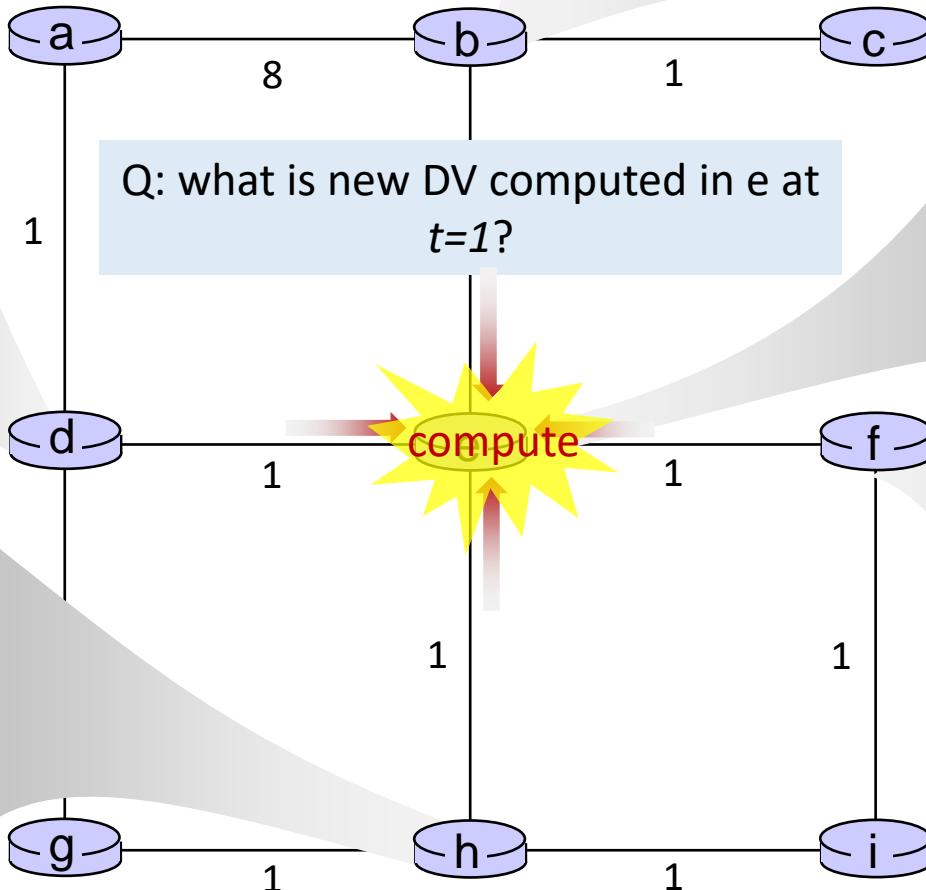


**t=1**

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

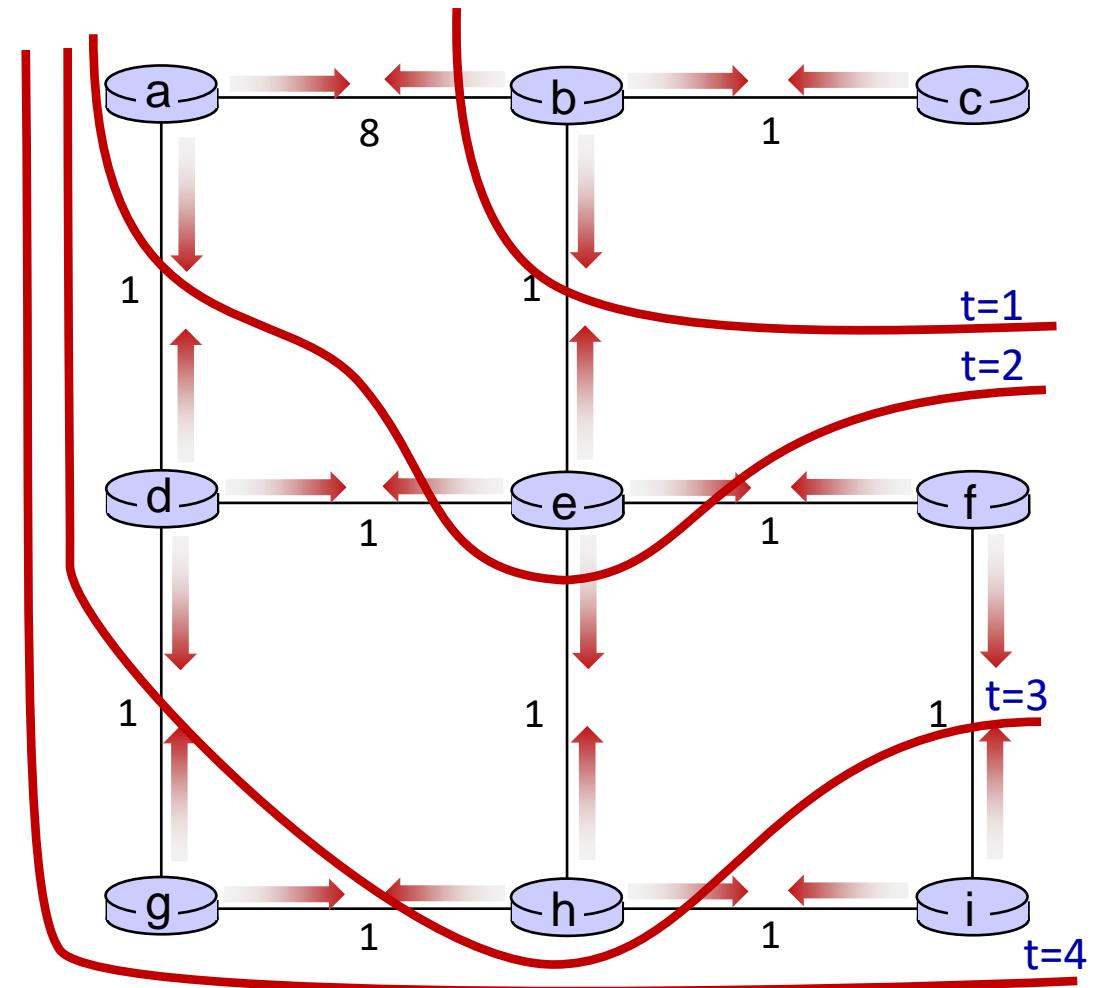
| DV in e:          |
|-------------------|
| $D_e(a) = \infty$ |
| $D_e(b) = 1$      |
| $D_e(c) = \infty$ |
| $D_e(d) = 1$      |
| $D_e(e) = 0$      |
| $D_e(f) = 1$      |
| $D_e(g) = \infty$ |
| $D_e(h) = 1$      |
| $D_e(i) = \infty$ |

| 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:

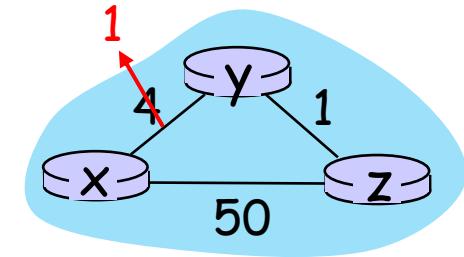
-  t=0 c's state at t=0 is at c only
-  t=1 c's state at t=0 has propagated to b, and may influence distance vector computations up to **1** hop away, i.e., at b
-  t=2 c's state at t=0 may now influence distance vector computations up to **2** hops away, i.e., at b and now at a, e as well
-  t=3 c's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



# 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.

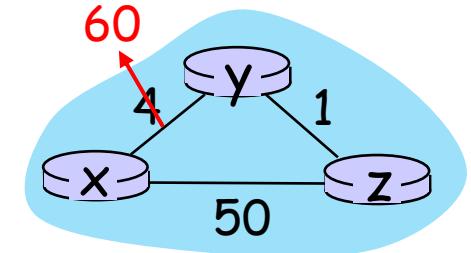
**“good news travels fast”**     $t_1$ :  $z$  receives update from  $y$ , updates its table, computes new least cost to  $x$ , sends its neighbors its DV.

$t_2$ :  $y$  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: roadmap

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

# 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”)

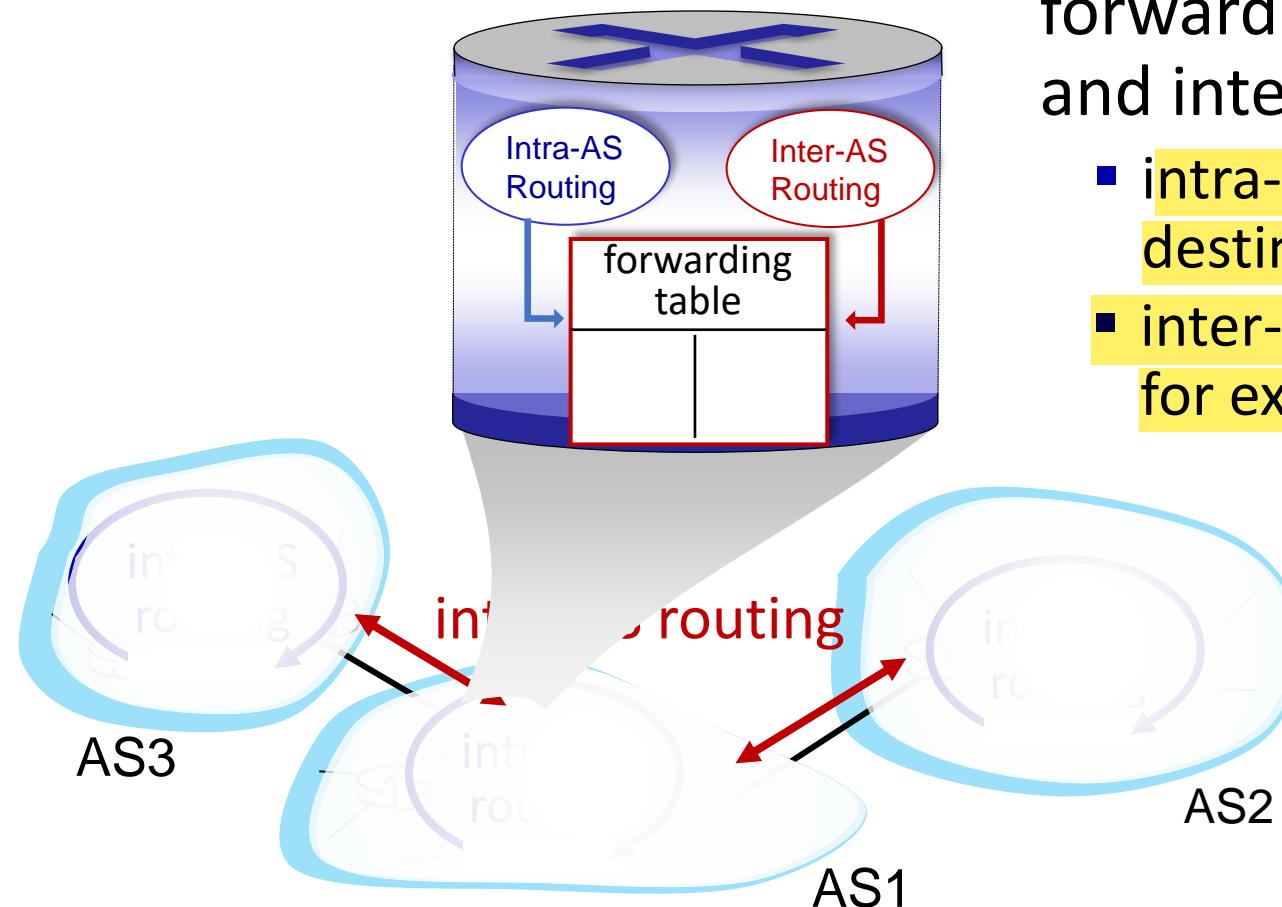
**intra-AS (aka “intra-domain”):**  
routing among *within same AS (“network”)*

- 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

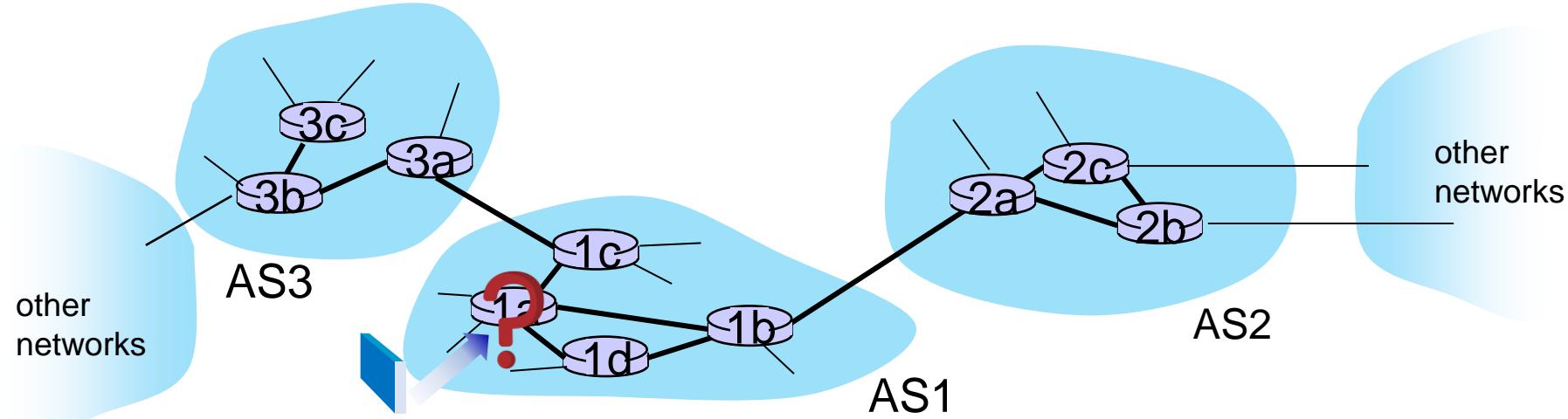
- intra-AS routing determine entries for destinations within AS
- inter-AS & intra-AS determine entries for external destinations

# Inter-AS routing: a role in intradomain forwarding

- suppose router in AS1 receives datagram destined outside of AS1:
  - router should forward packet to gateway router in AS1, but which one?

**AS1 inter-domain routing must:**

1. learn which destinations reachable through AS2, which through AS3
2. propagate this reachability info to all routers in AS1



# Intra-AS routing: routing within an AS

most common intra-AS routing protocols:

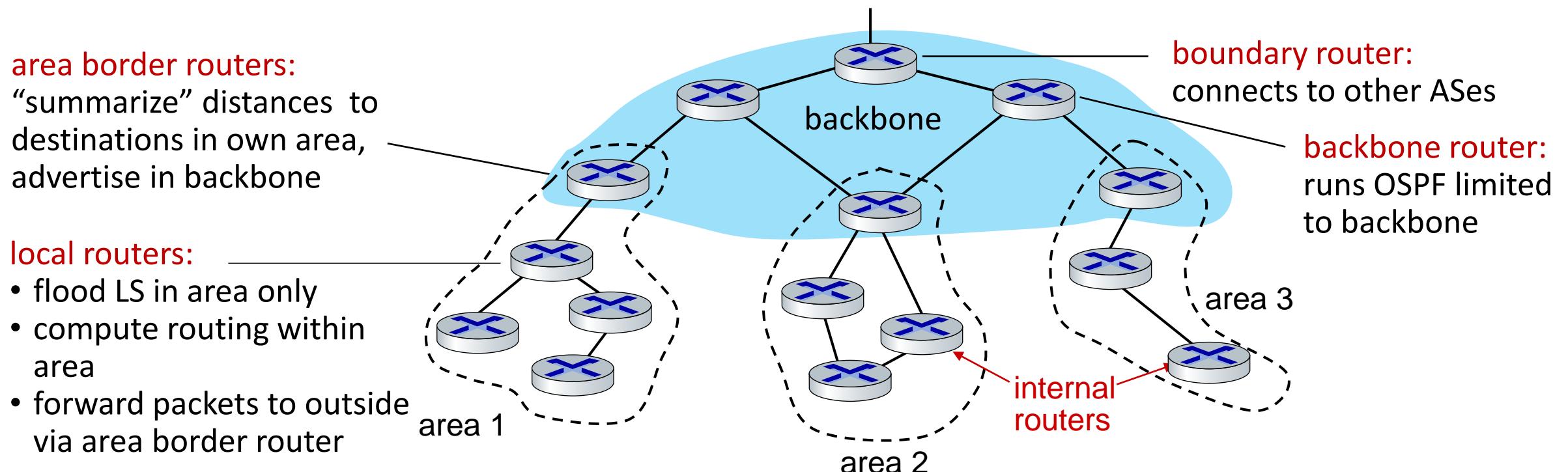
- **RIP: Routing Information Protocol [RFC 1723]**
  - classic DV: DVs exchanged every 30 secs
  - no longer widely used
- **EIGRP: Enhanced Interior Gateway Routing Protocol**
  - DV based
  - formerly Cisco-proprietary for decades (became open in 2013 [RFC 7868])
- **OSPF: Open Shortest Path First [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



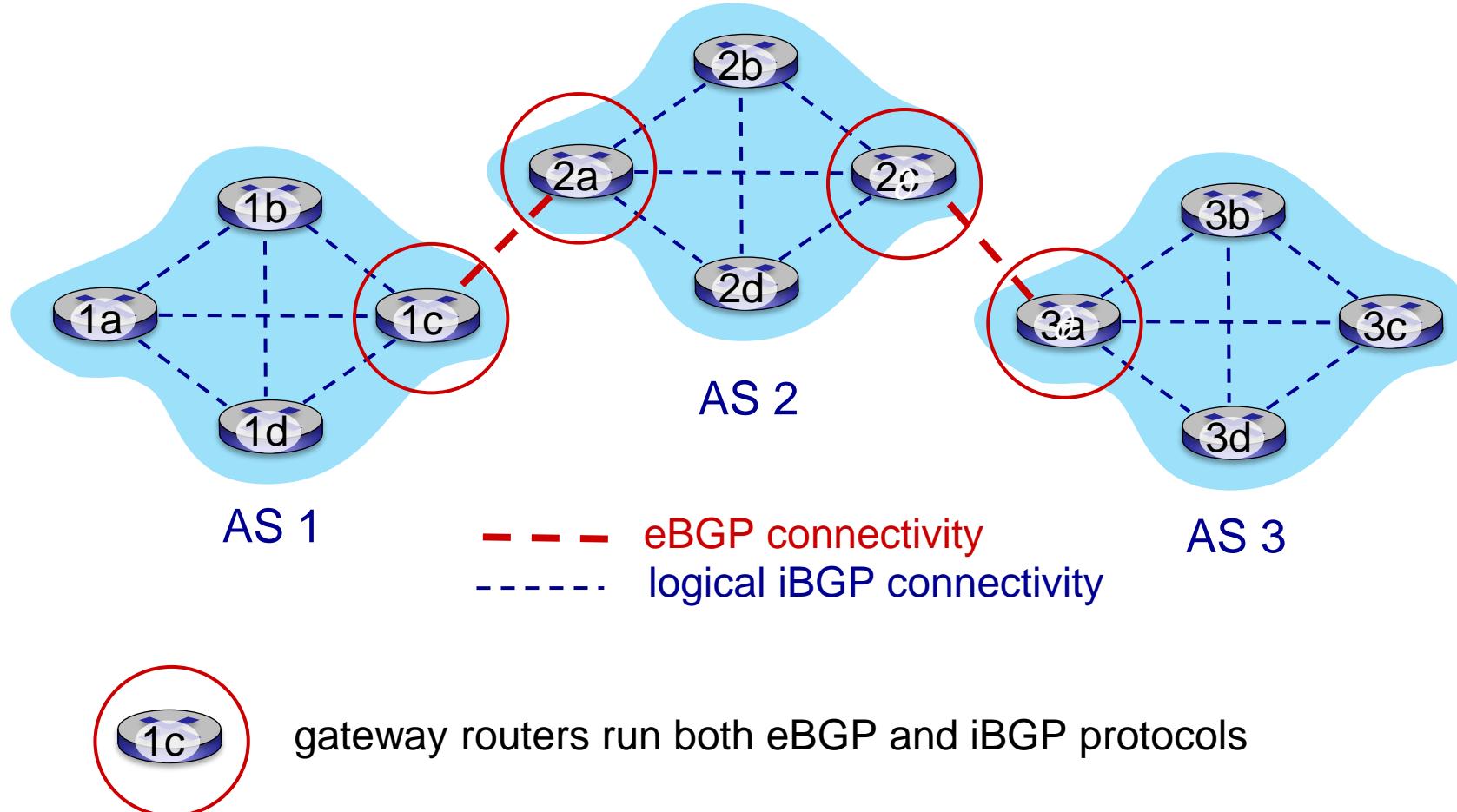
# Network layer: roadmap

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

# Internet inter-AS routing: BGP

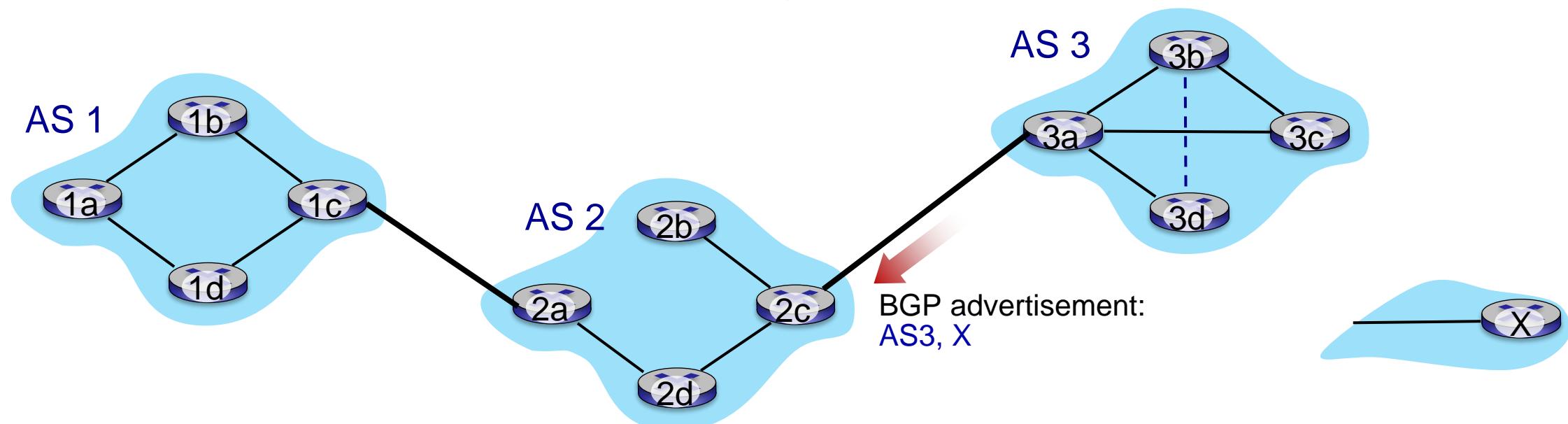
- BGP (Border Gateway Protocol): *the de facto* inter-domain routing protocol
  - “glue that holds the Internet together”
- allows subnet to advertise its existence, and the destinations it can reach, to rest of Internet: *“I am here, here is who I can reach, and how”*
- BGP provides each AS a means 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



# Path attributes and BGP routes

✓ BGP advertised route: prefix + attributes

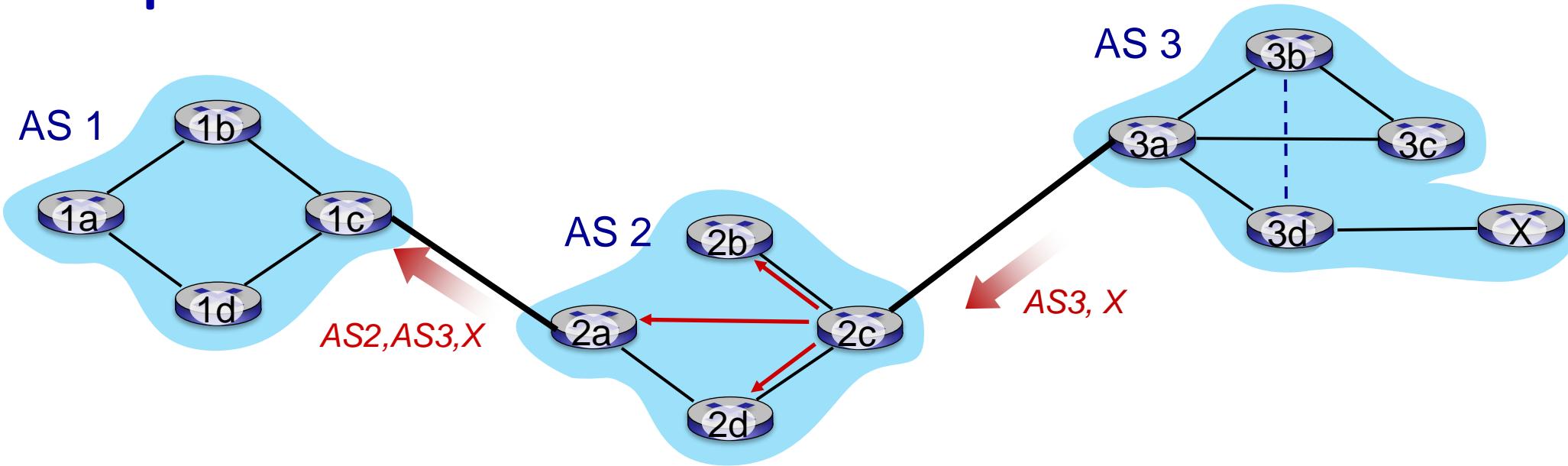
- prefix: destination being advertised ✓
- two important attributes:

- 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:

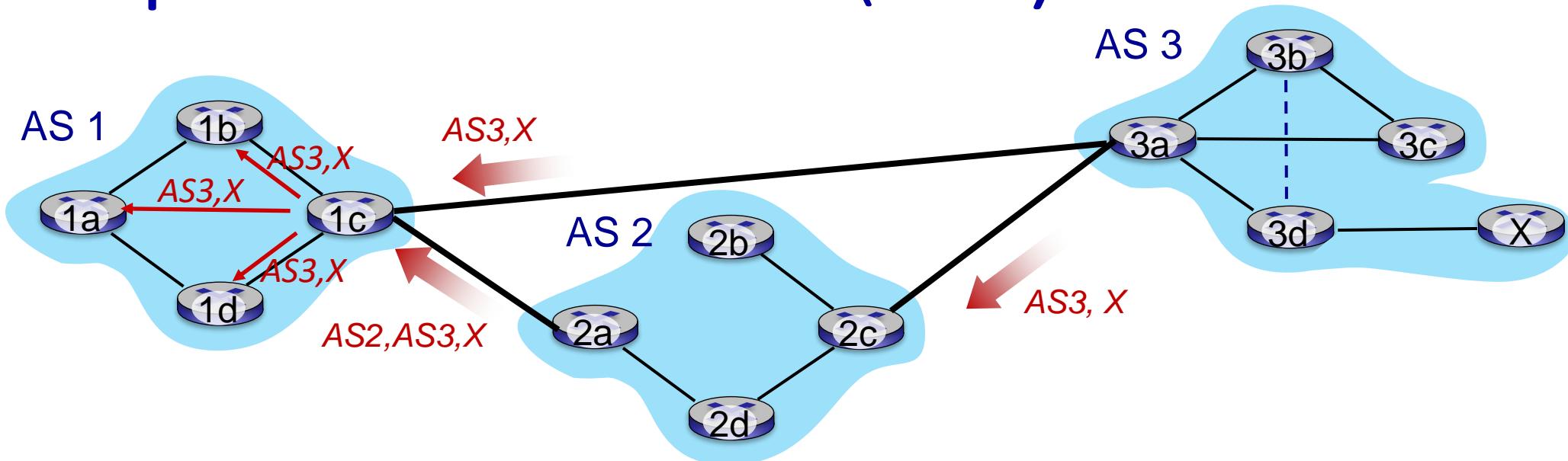
- gateway receiving route advertisement uses *import policy* to accept/decline path (e.g., never route through AS Y). ✓
- AS policy also determines whether to *advertise* path to other other neighboring ASes

# 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

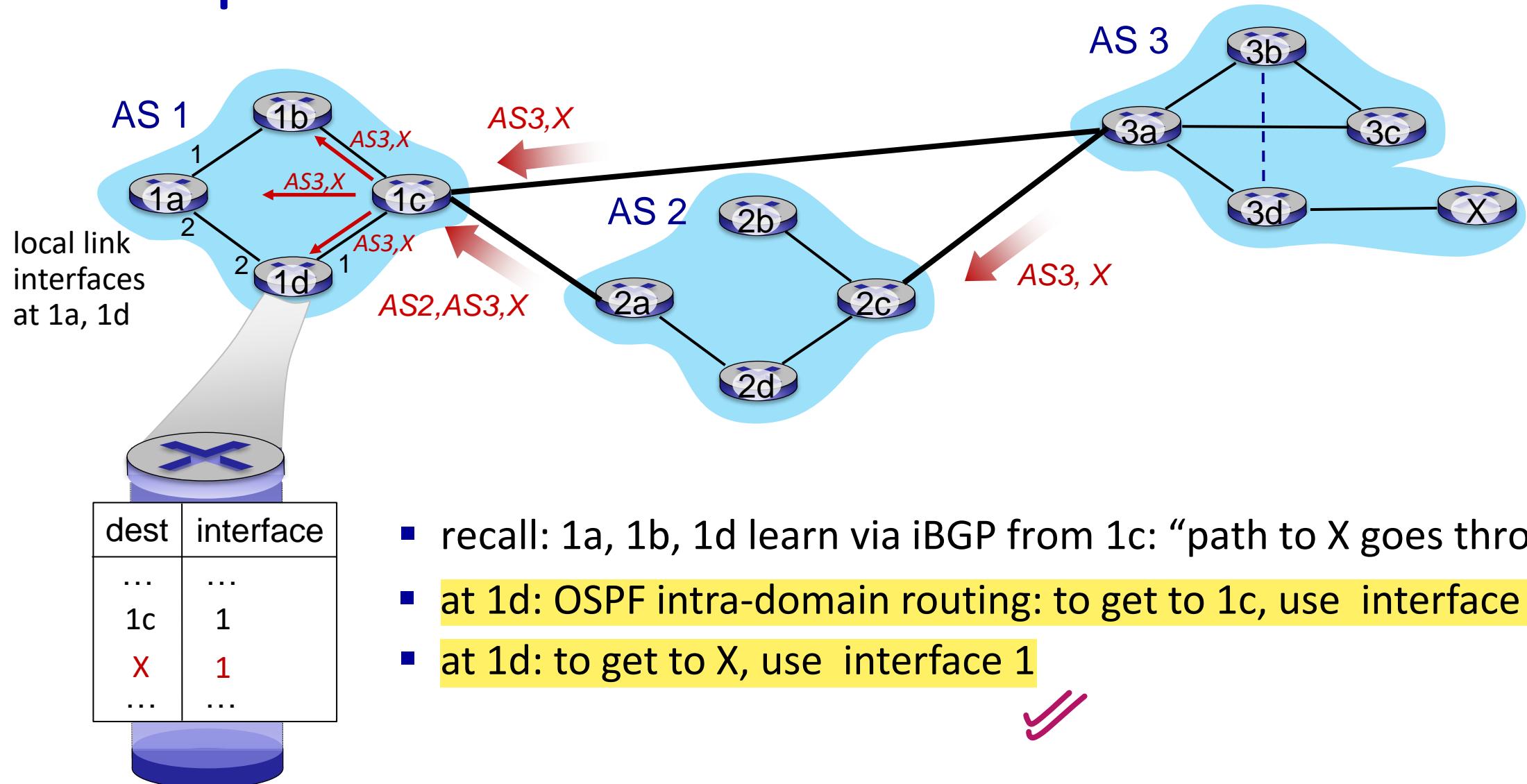
# BGP path advertisement (more)



gateway router may learn about multiple paths to destination:

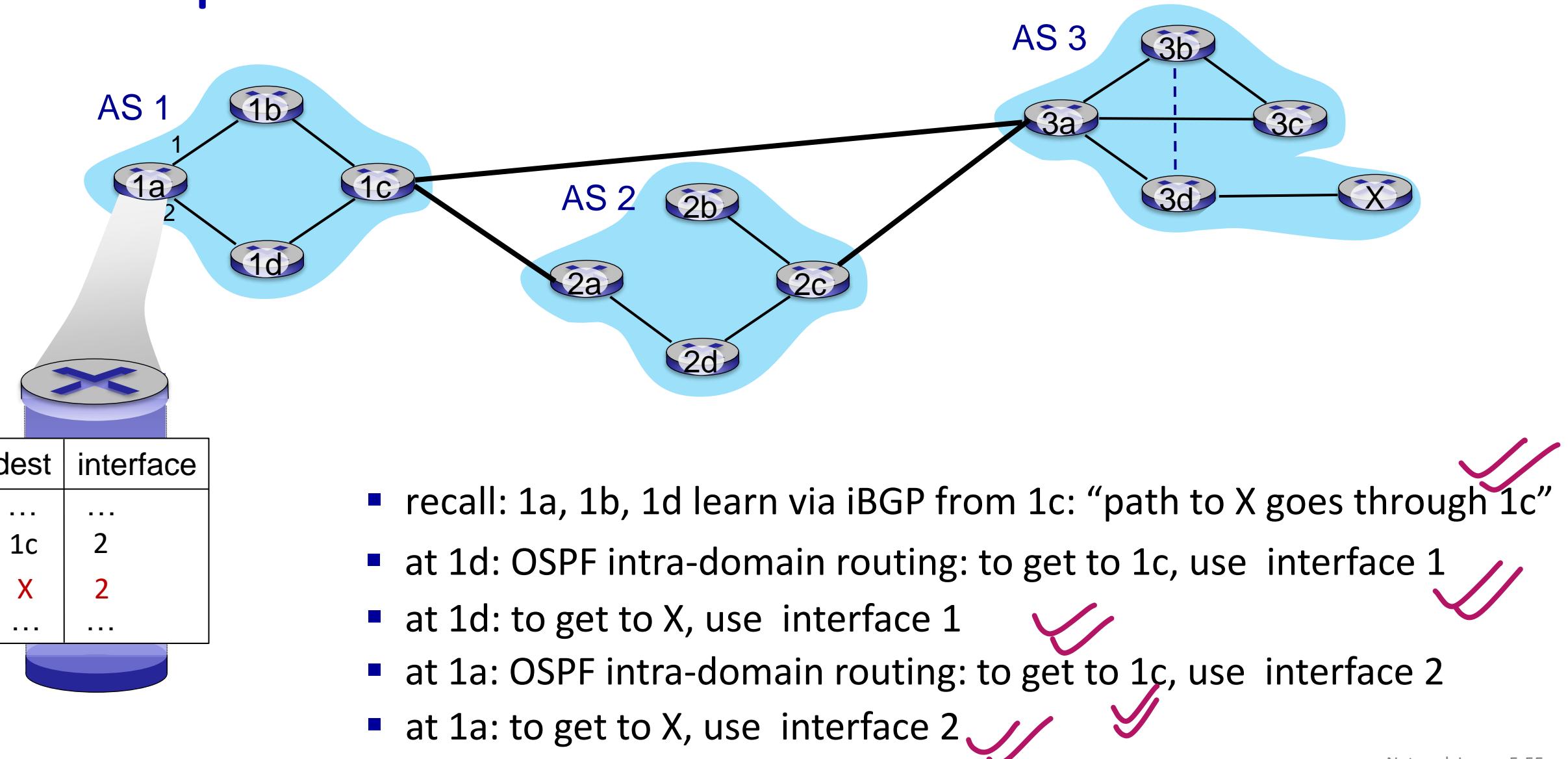
- AS1 gateway router 1c learns path **AS2,AS3,X** from 2a ✓
- AS1 gateway router 1c learns path **AS3,X** from 3a ✓
- based on *policy*, AS1 gateway router 1c chooses path **AS3,X** and advertises path within AS1 via iBGP ✓

# BGP path advertisement



- 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 path advertisement



# 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

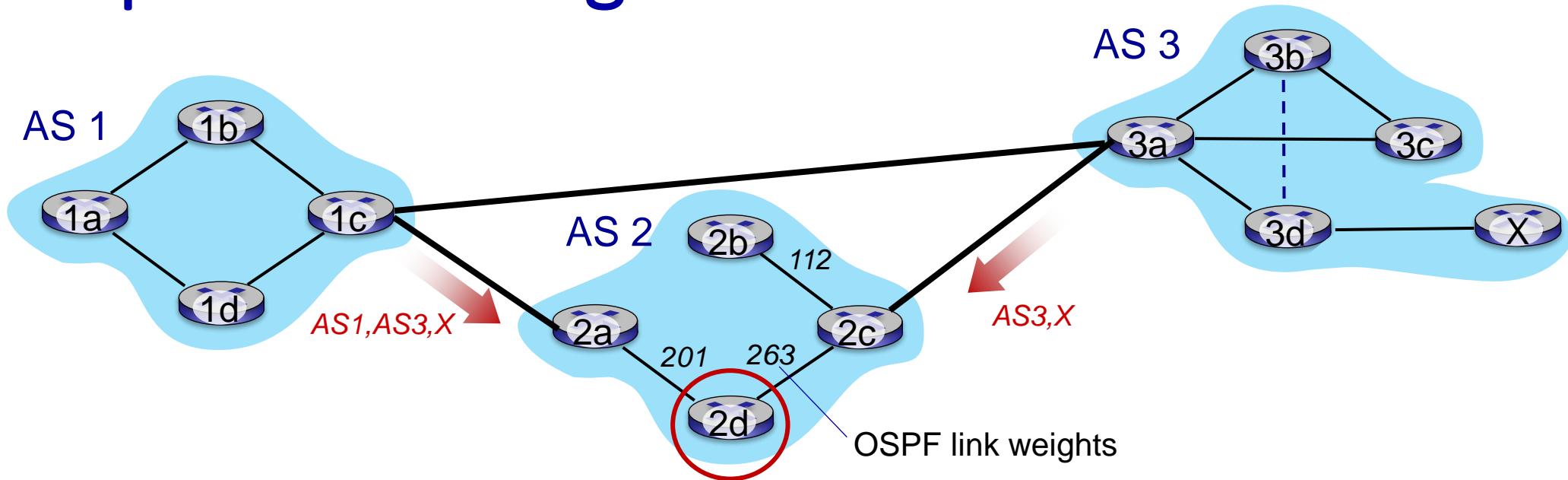
scale:

- hierarchical routing saves table size, reduced update traffic

performance:

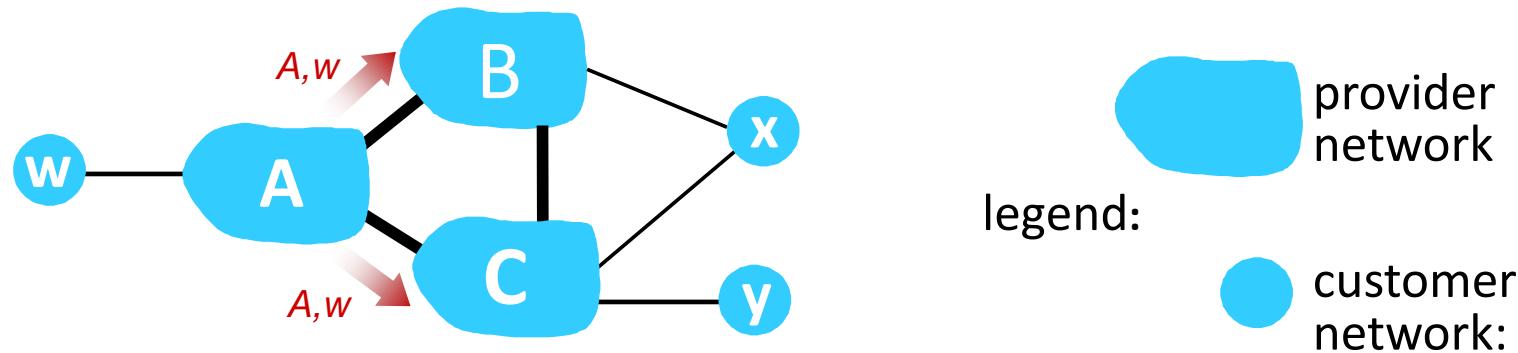
- intra-AS: can focus on performance
- inter-AS: policy dominates over performance

# 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!

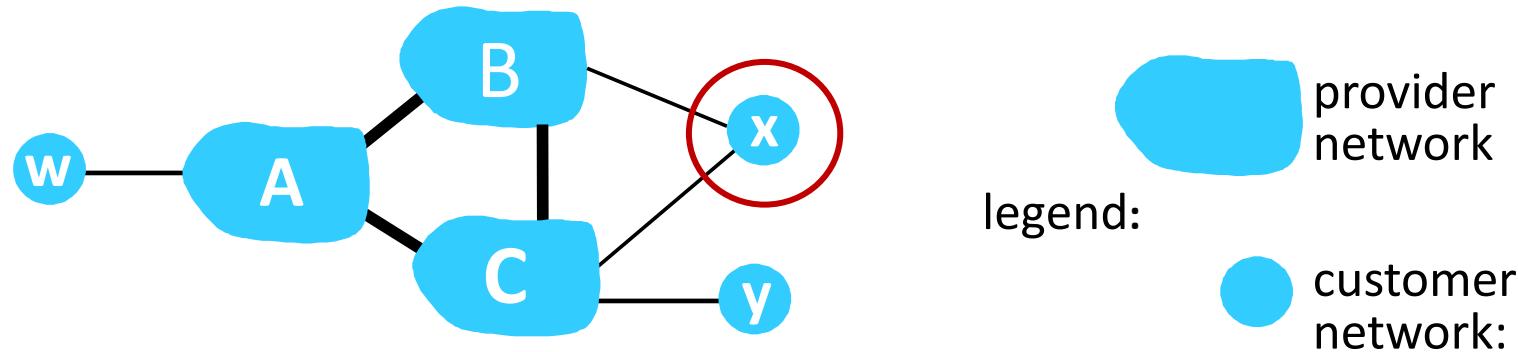
# BGP: achieving policy via advertisements



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 CBAw, since none of C, A, w are B’s customers
  - C does *not* learn about CBAw 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 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

# 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

# Network layer: roadmap

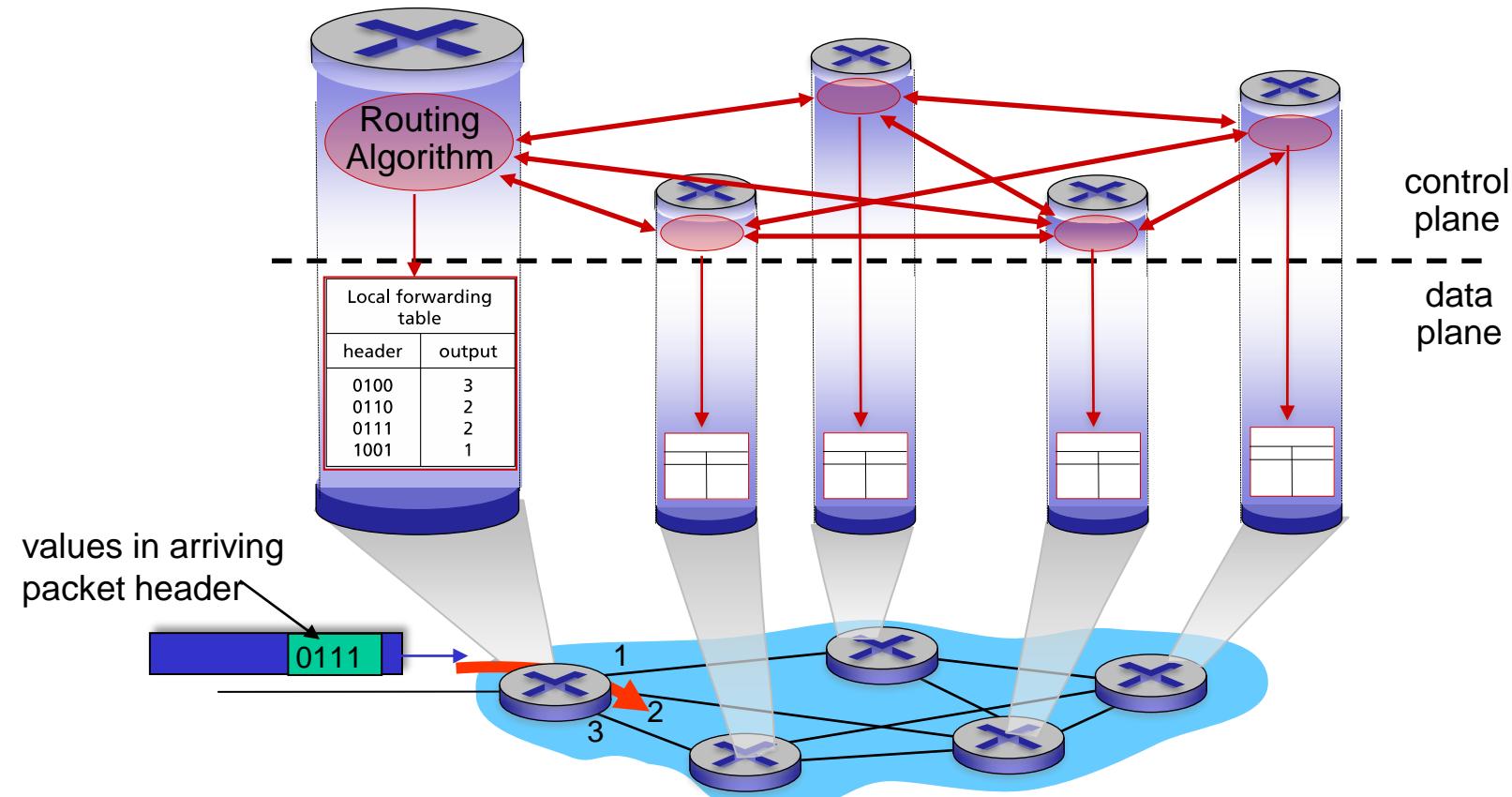
- introduction
- routing protocols
- intra-ISP routing: OSPF
- routing among ISPs: BGP
- **SDN control plane**
- Internet Control Message Protocol

# Software defined networking (SDN)

- Internet network layer: historically implemented via distributed, per-router control approach:
  - *monolithic* router contains switching hardware, runs proprietary implementation of Internet standard protocols (IP, RIP, IS-IS, OSPF, BGP) in proprietary router OS (e.g., Cisco IOS)
  - different “middleboxes” for different network layer functions: firewalls, load balancers, NAT boxes, ..
- ~2005: renewed interest in rethinking network control plane

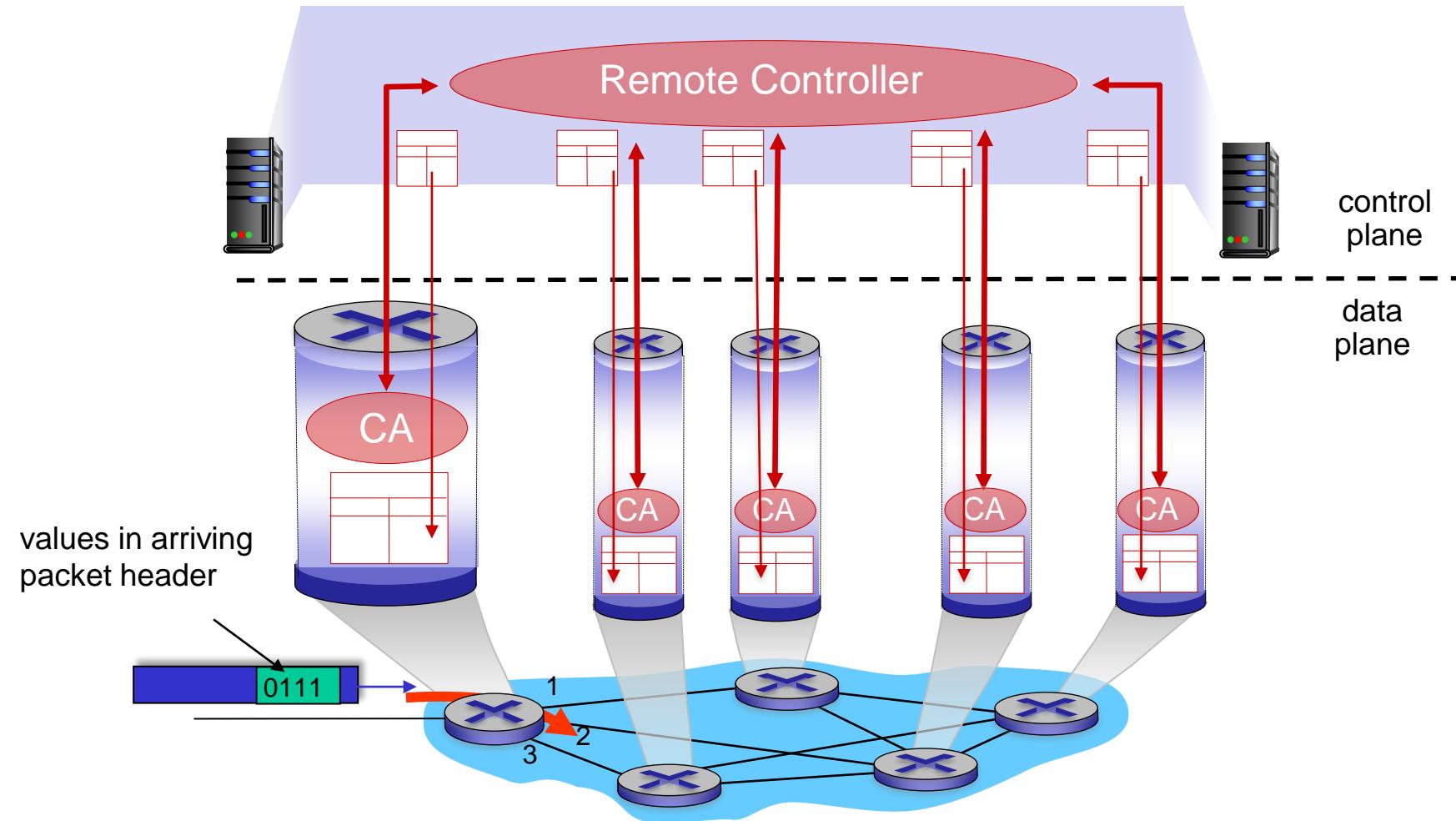
# Per-router control plane

Individual routing algorithm components *in each and every router* interact in the control plane to compute forwarding tables



# Software-Defined Networking (SDN) control plane

Remote controller computes, installs forwarding tables in routers



# Software defined networking (SDN)

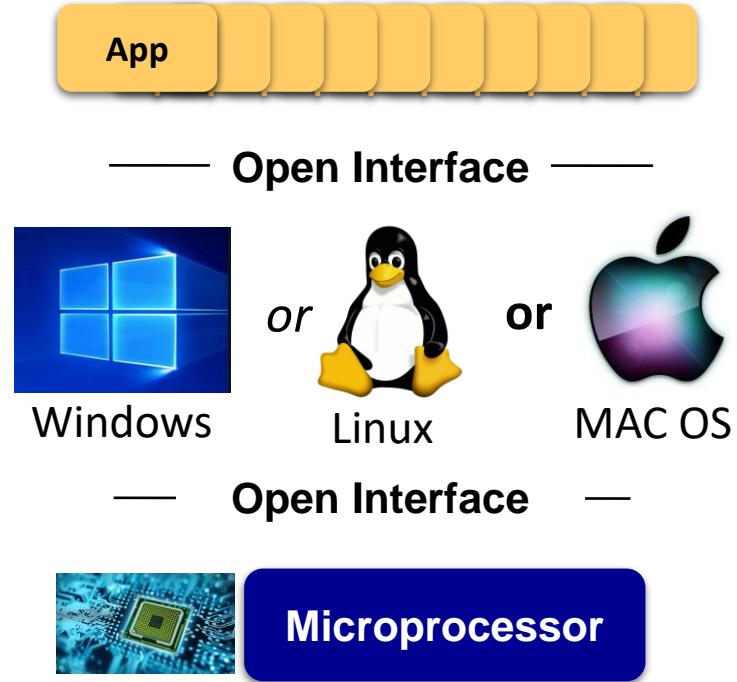
*Why* a *logically centralized* control plane?

- easier network management: avoid router misconfigurations,  
greater flexibility of traffic flows
- table-based forwarding allows “programming” routers
  - centralized “programming” easier: compute tables centrally and distribute
  - distributed “programming” more difficult: compute tables as result of  
distributed algorithm (protocol) implemented in each-and-every router
- open (non-proprietary) implementation of control plane
  - foster innovation: let 1000 flowers bloom

# SDN analogy: mainframe to PC revolution

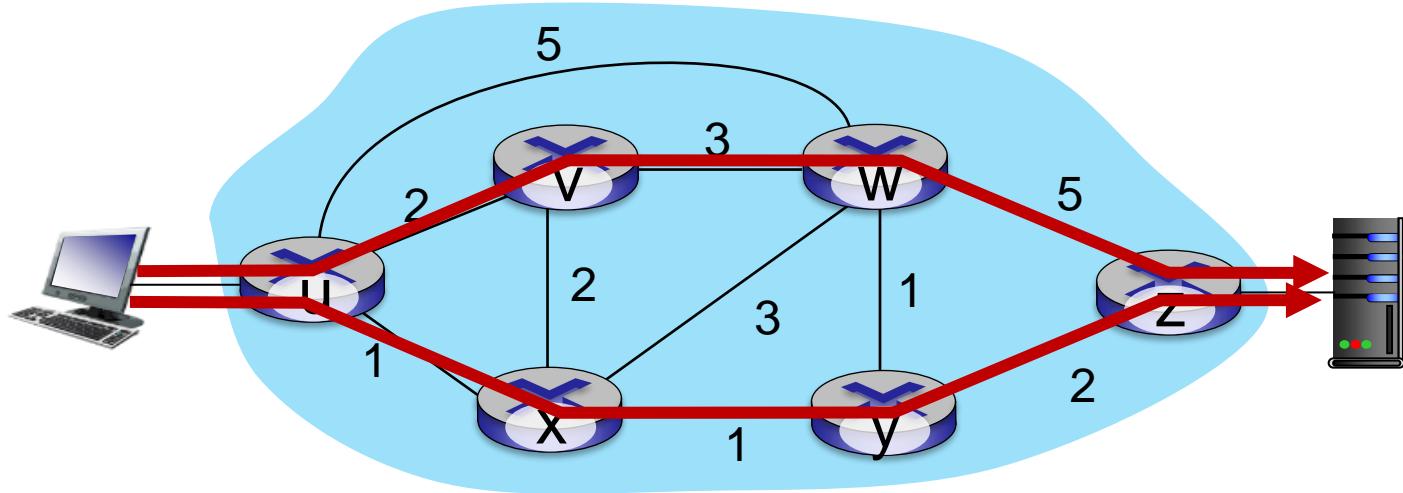


Vertically integrated  
Closed, proprietary  
Slow innovation  
Small industry



Horizontal  
Open interfaces  
Rapid innovation  
Huge industry

# Traffic engineering: difficult with traditional routing

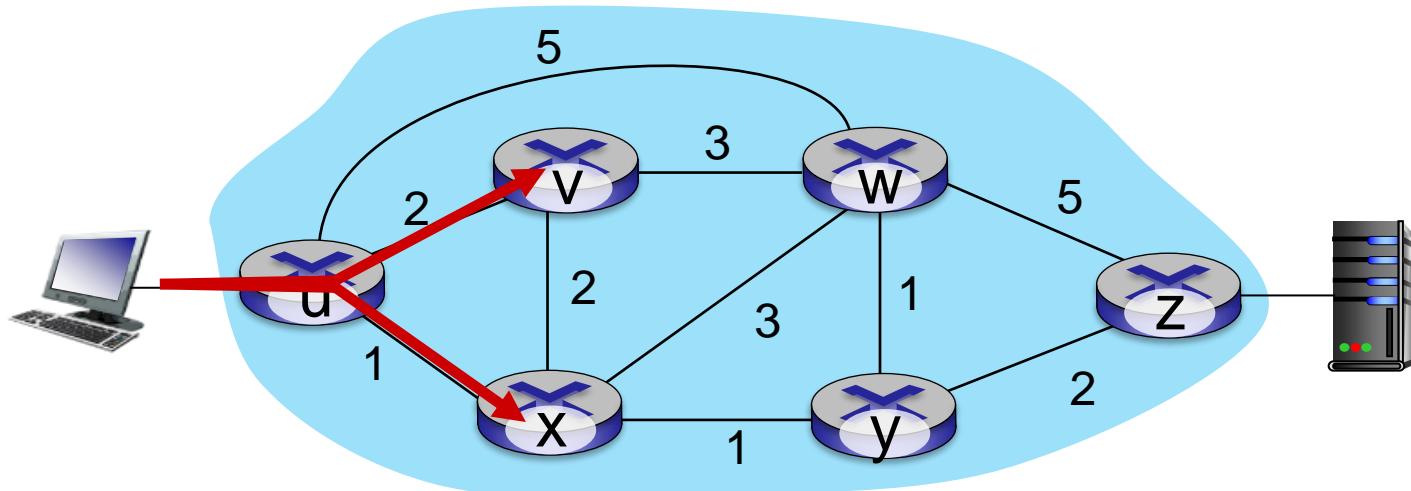


Q: what if network operator wants u-to-z traffic to flow along  $uvwz$ , rather than  $uxyz$ ?

A: need to re-define link weights so traffic routing algorithm computes routes accordingly (or need a new routing algorithm)!

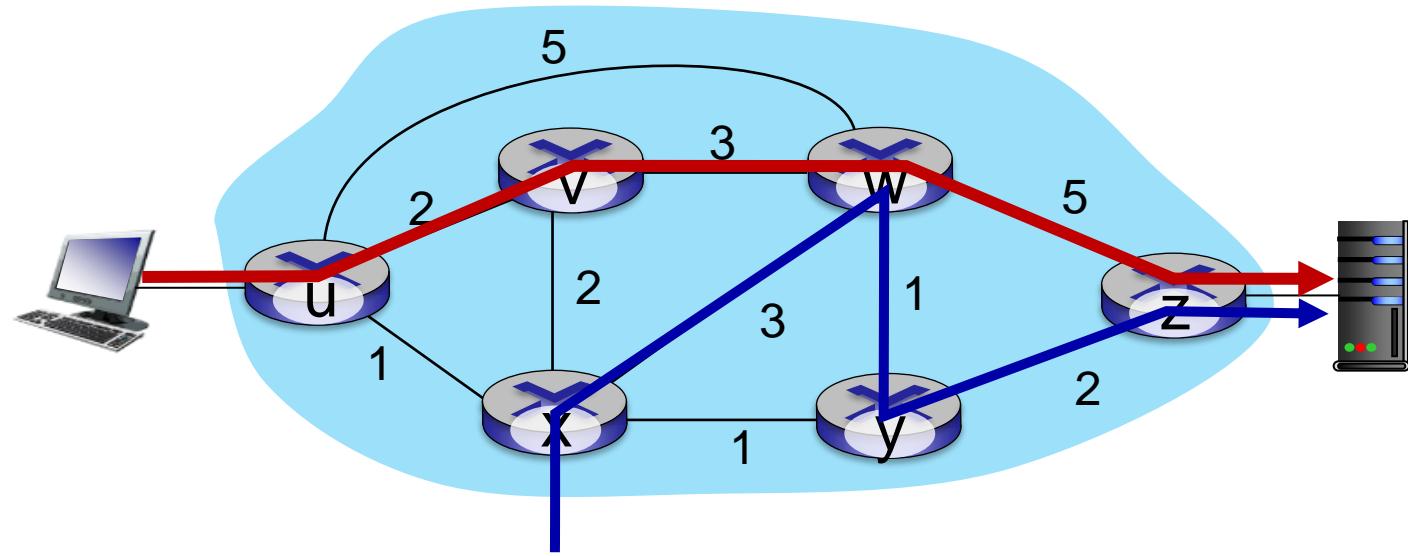
*link weights are only control “knobs”: not much control!*

# Traffic engineering: difficult with traditional routing



Q: what if network operator wants to split u-to-z traffic along uvwz *and* uxyz (load balancing)?  
A: can't do it (or need a new routing algorithm)

## Traffic engineering: difficult with traditional routing



Q: what if w wants to route blue and red traffic differently from w to z?

A: can't do it (with destination-based forwarding, and LS, DV routing)

# Software defined networking (SDN)

4. programmable  
control  
applications

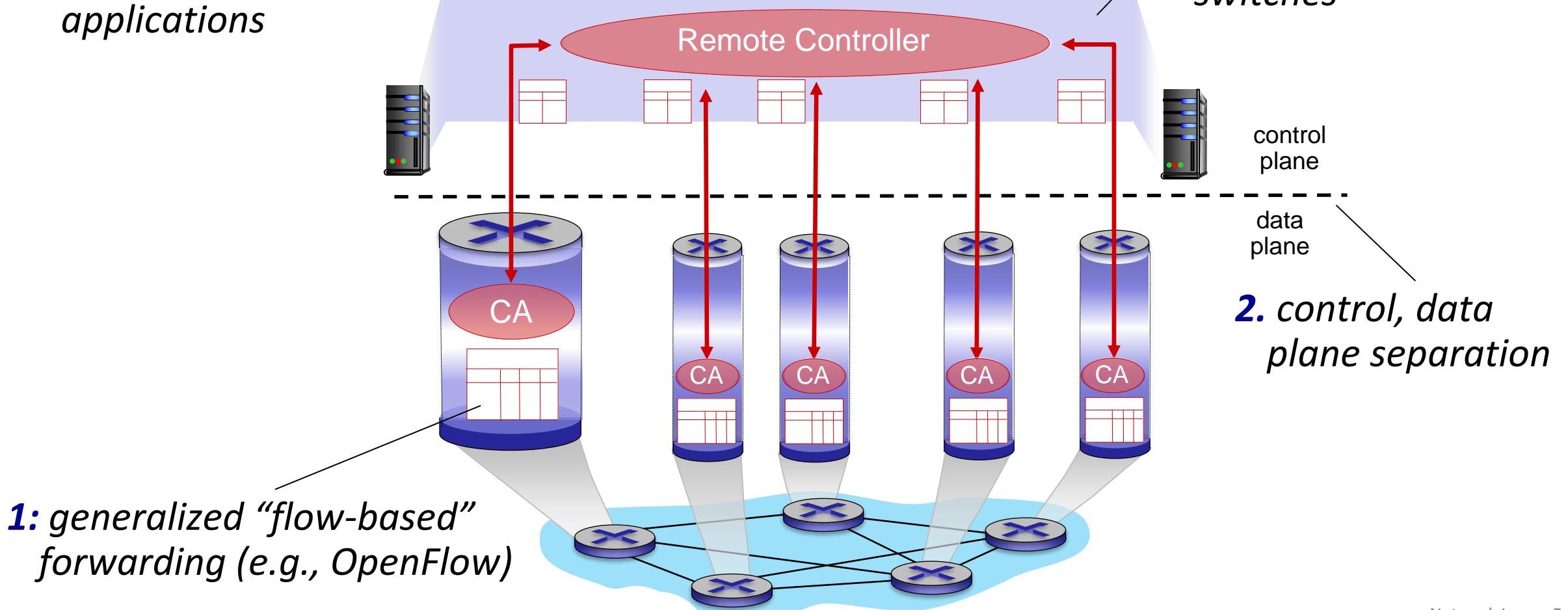
routing

access  
control

...

load  
balance

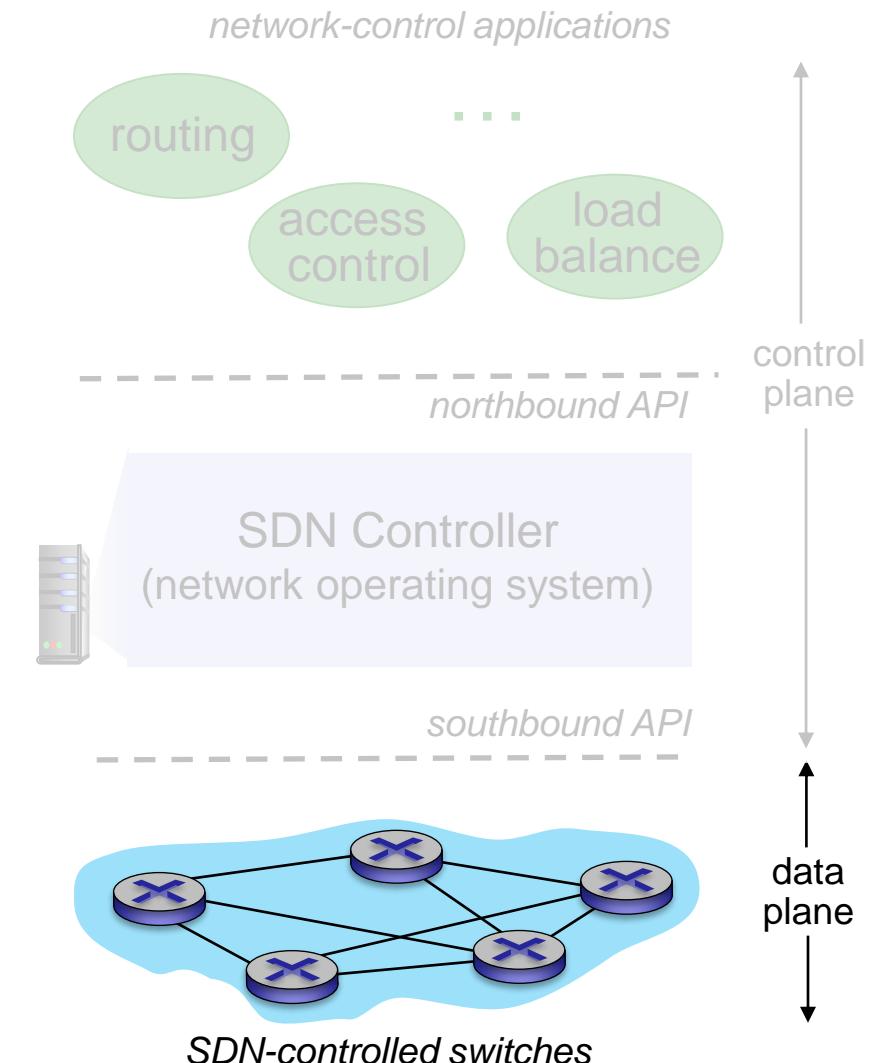
3. control plane functions  
external to data-plane  
switches



# Software defined networking (SDN)

## Data-plane switches:

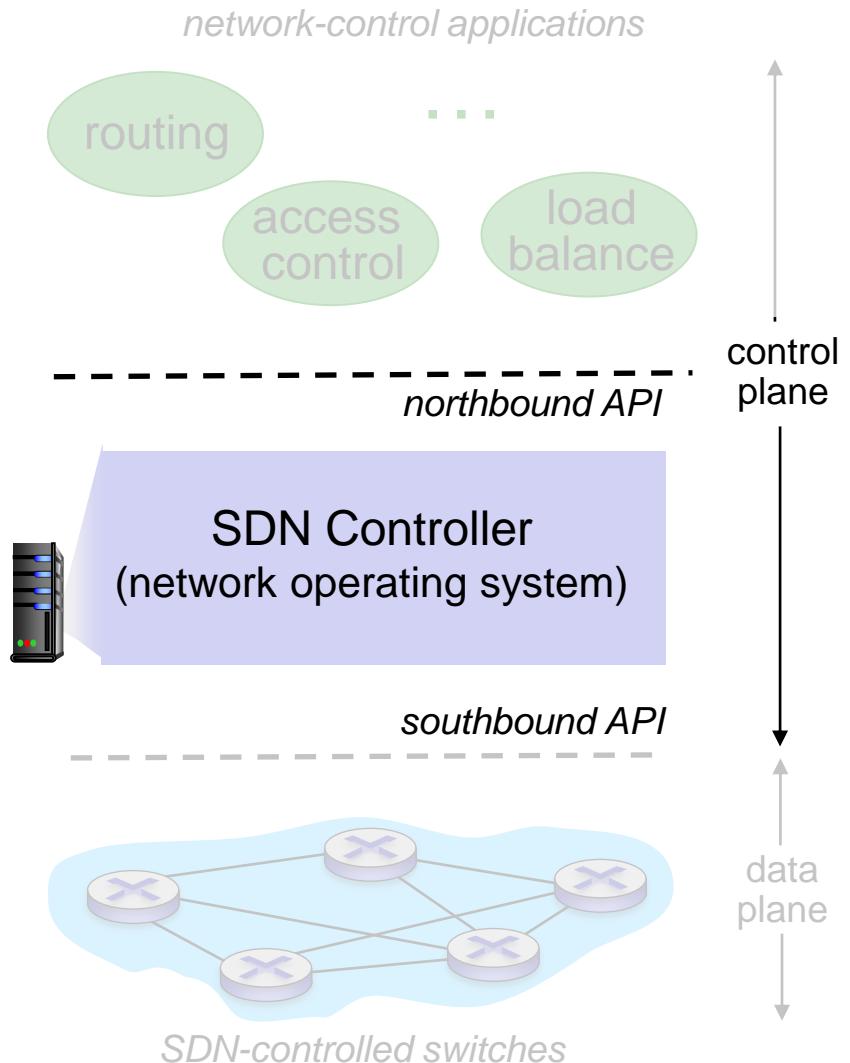
- fast, simple, commodity switches  
implementing generalized data-plane  
forwarding in hardware
- flow (forwarding) table computed,  
installed under controller supervision
- API for table-based switch control  
(e.g., OpenFlow)
  - defines what is controllable, what is not
- protocol for communicating with  
controller (e.g., OpenFlow)



# Software defined networking (SDN)

## SDN controller (network OS):

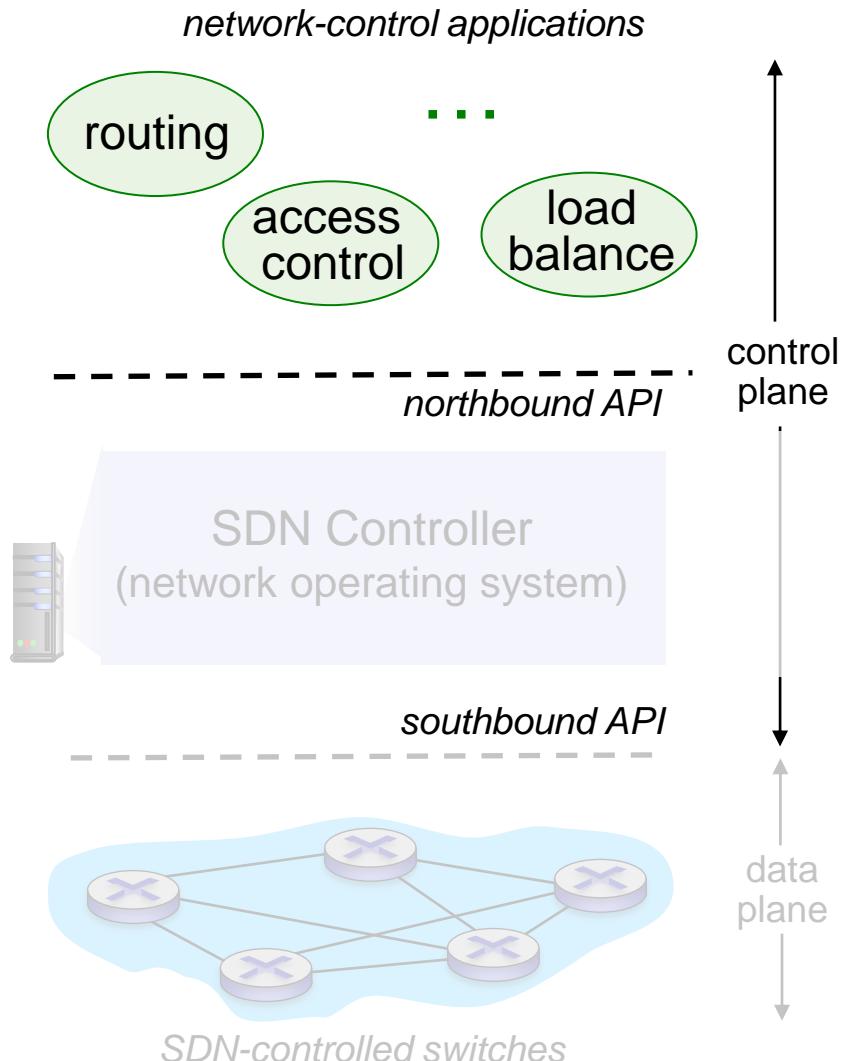
- maintain network state information
- interacts with network control applications “above” via northbound API
- interacts with network switches “below” via southbound API
- implemented as distributed system for performance, scalability, fault-tolerance, robustness



# Software defined networking (SDN)

## network-control apps:

- “brains” of control:  
implement control functions  
using lower-level services, API  
provided by SDN controller
- *unbundled*: can be provided by  
3<sup>rd</sup> party: distinct from routing  
vendor, or SDN controller

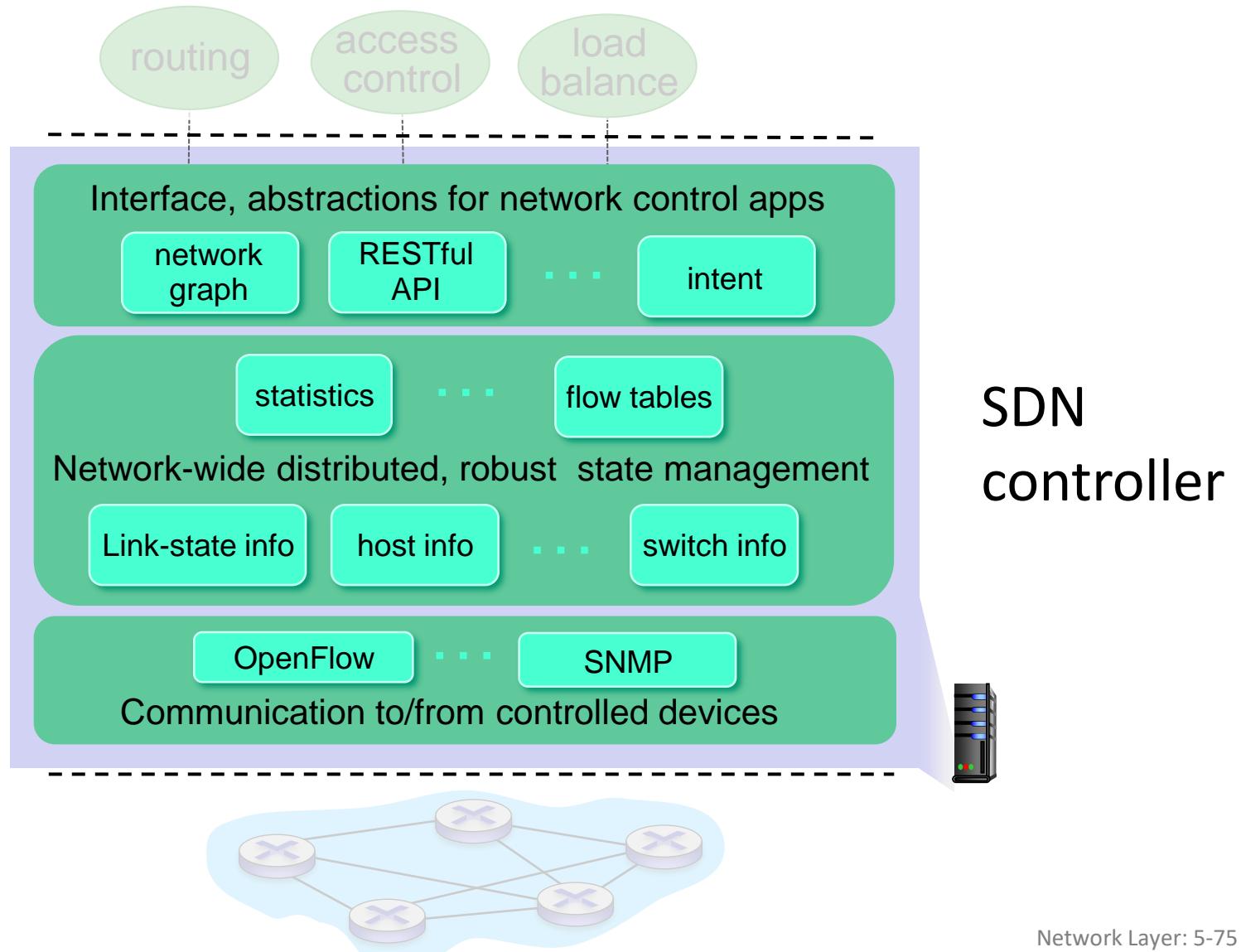


# Components of SDN controller

interface layer to network control apps: abstractions API

network-wide state management : state of networks links, switches, services: a *distributed database*

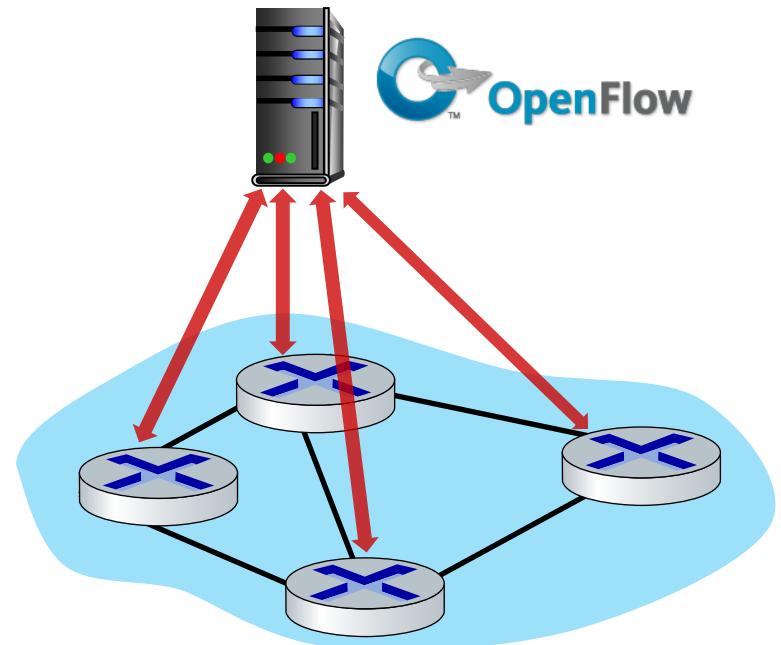
**communication**: communicate between SDN controller and controlled switches



# OpenFlow protocol

- operates between controller, switch
- TCP used to exchange messages
  - optional encryption
- three classes of OpenFlow messages:
  - controller-to-switch
  - asynchronous (switch to controller)
  - symmetric (misc.)
- distinct from OpenFlow API
  - API used to specify generalized forwarding actions

OpenFlow Controller

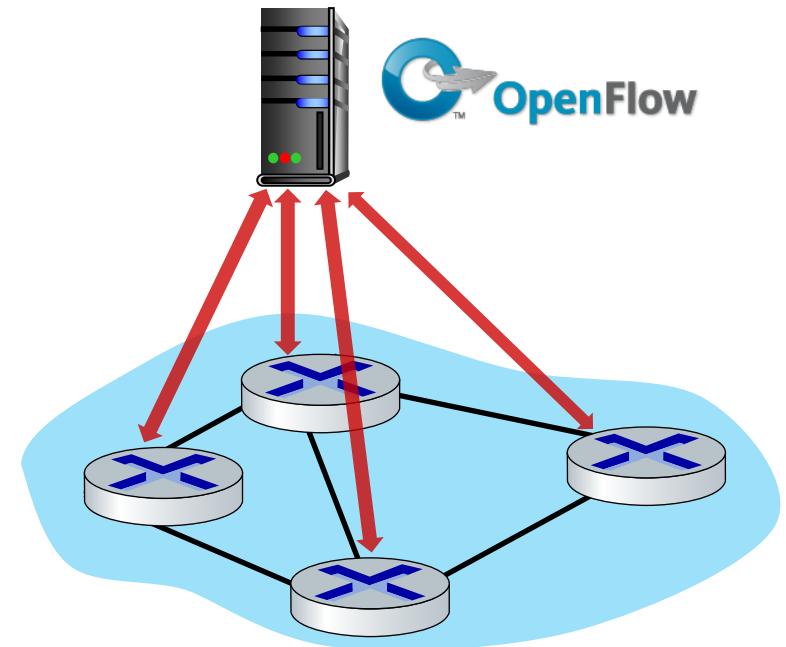


# OpenFlow: controller-to-switch messages

## Key controller-to-switch messages

- *features*: controller queries switch features, switch replies
- *configure*: controller queries/sets switch configuration parameters
- *modify-state*: add, delete, modify flow entries in the OpenFlow tables
- *packet-out*: controller can send this packet out of specific switch port

## OpenFlow Controller

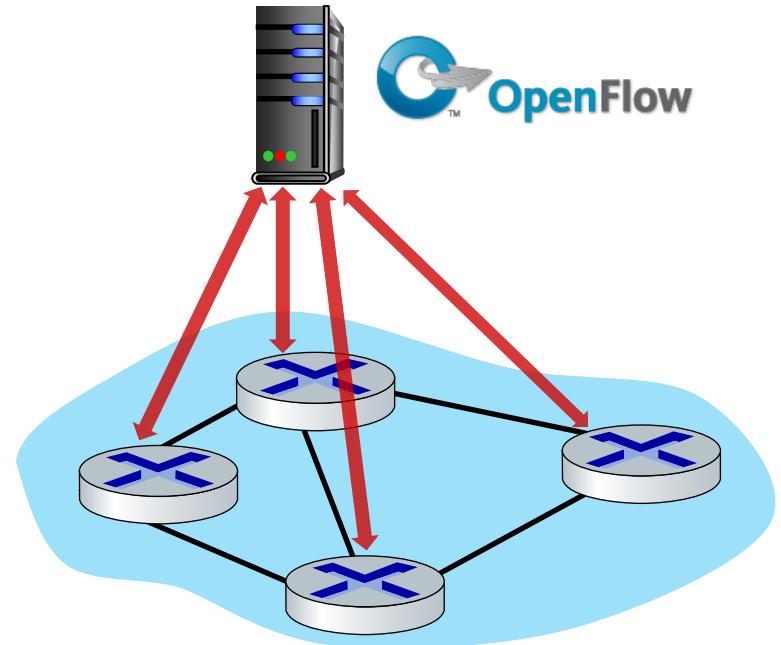


# OpenFlow: switch-to-controller messages

## Key switch-to-controller messages

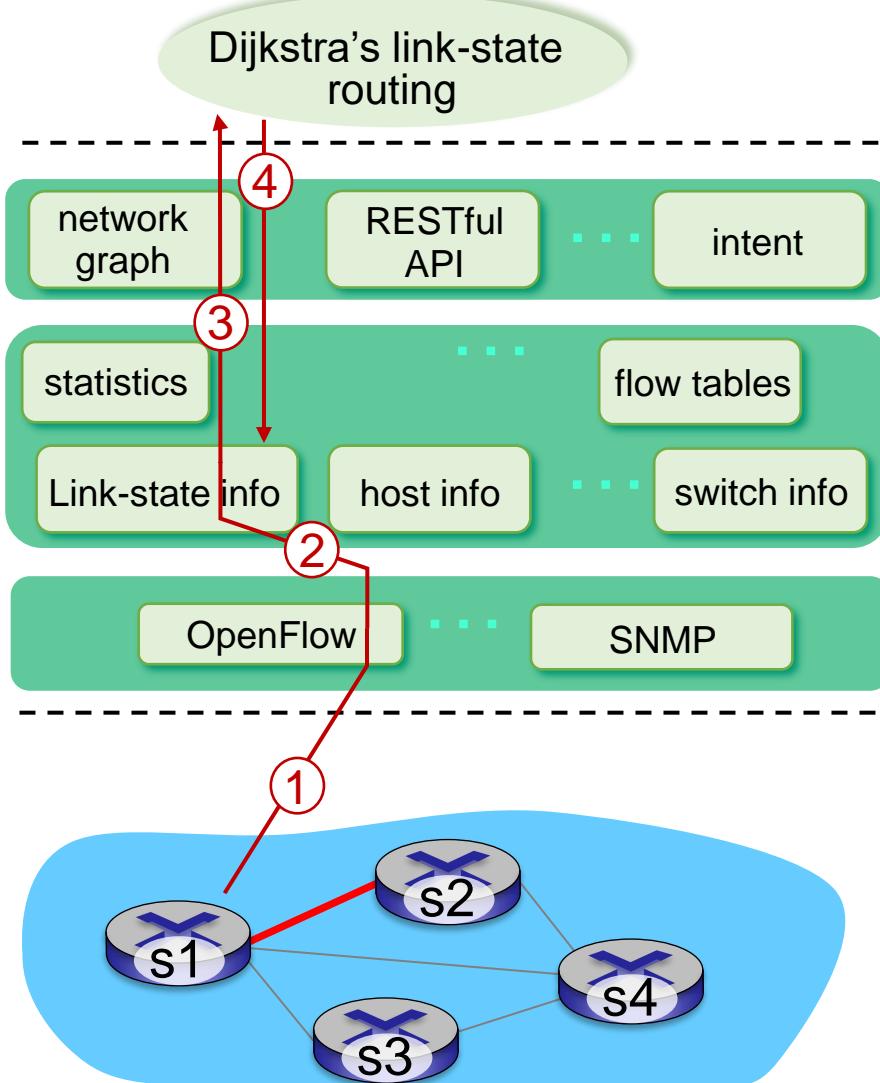
- *packet-in*: transfer packet (and its control) to controller. See packet-out message from controller
- *flow-removed*: flow table entry deleted at switch
- *port status*: inform controller of a change on a port.

## OpenFlow Controller



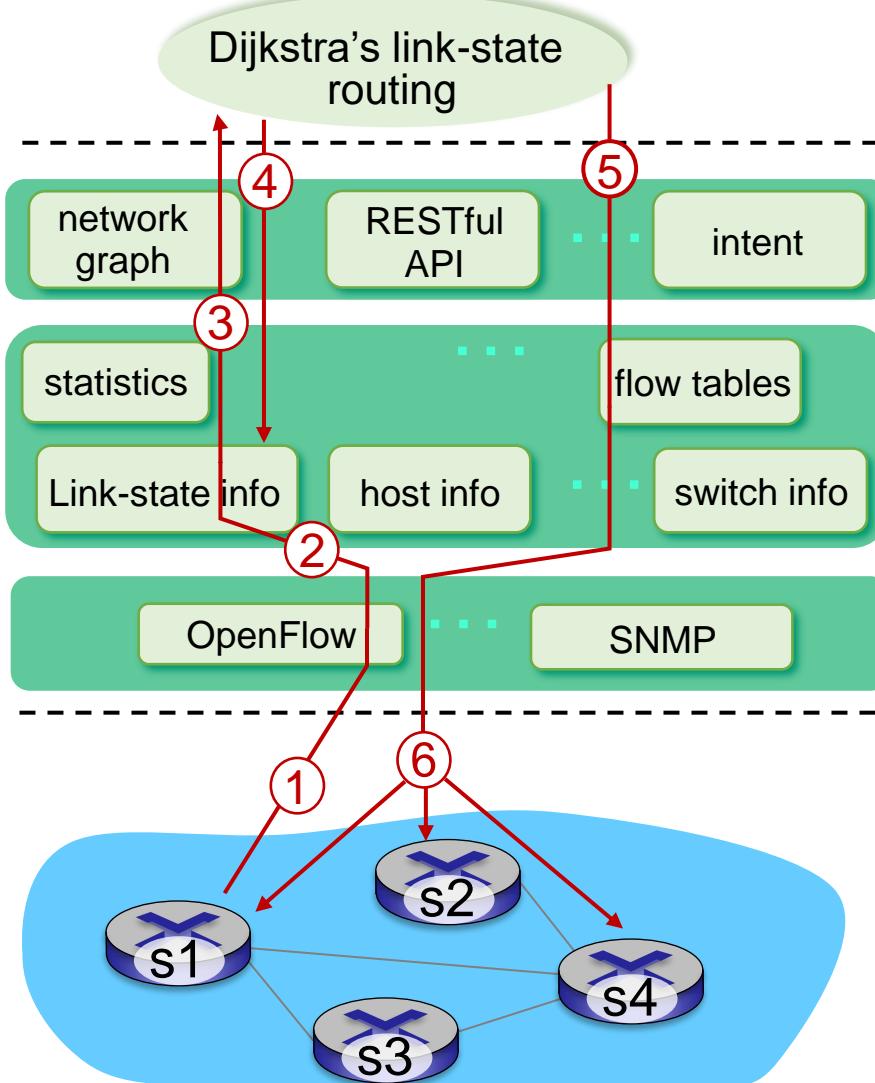
Fortunately, network operators don't "program" switches by creating/sending OpenFlow messages directly. Instead use higher-level abstraction at controller

# SDN: control/data plane interaction example



- ① S1, experiencing link failure uses OpenFlow port status message to notify controller
- ② SDN controller receives OpenFlow message, updates link status info
- ③ Dijkstra's routing algorithm application has previously registered to be called whenever link status changes. It is called.
- ④ Dijkstra's routing algorithm access network graph info, link state info in controller, computes new routes

# SDN: control/data plane interaction example



- ⑤ link state routing app interacts with flow-table-computation component in SDN controller, which computes new flow tables needed
- ⑥ controller uses OpenFlow to install new tables in switches that need updating

# Network layer: roadmap

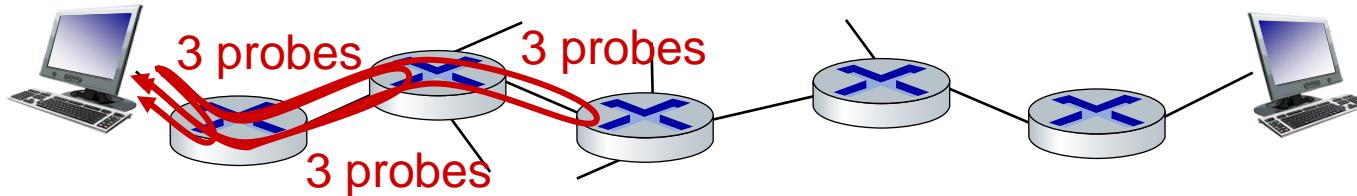
- introduction
- routing protocols
- intra-ISP routing: OSPF
- routing among ISPs: BGP
- SDN control plane
- **Internet Control Message Protocol (ICMP)**

# 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

# Link Layer

Slides are adapted from the companion website of the book *Computer Networking: A Top-Down Approach* by Jim Kurose, Keith Ross

# Link layer: roadmap

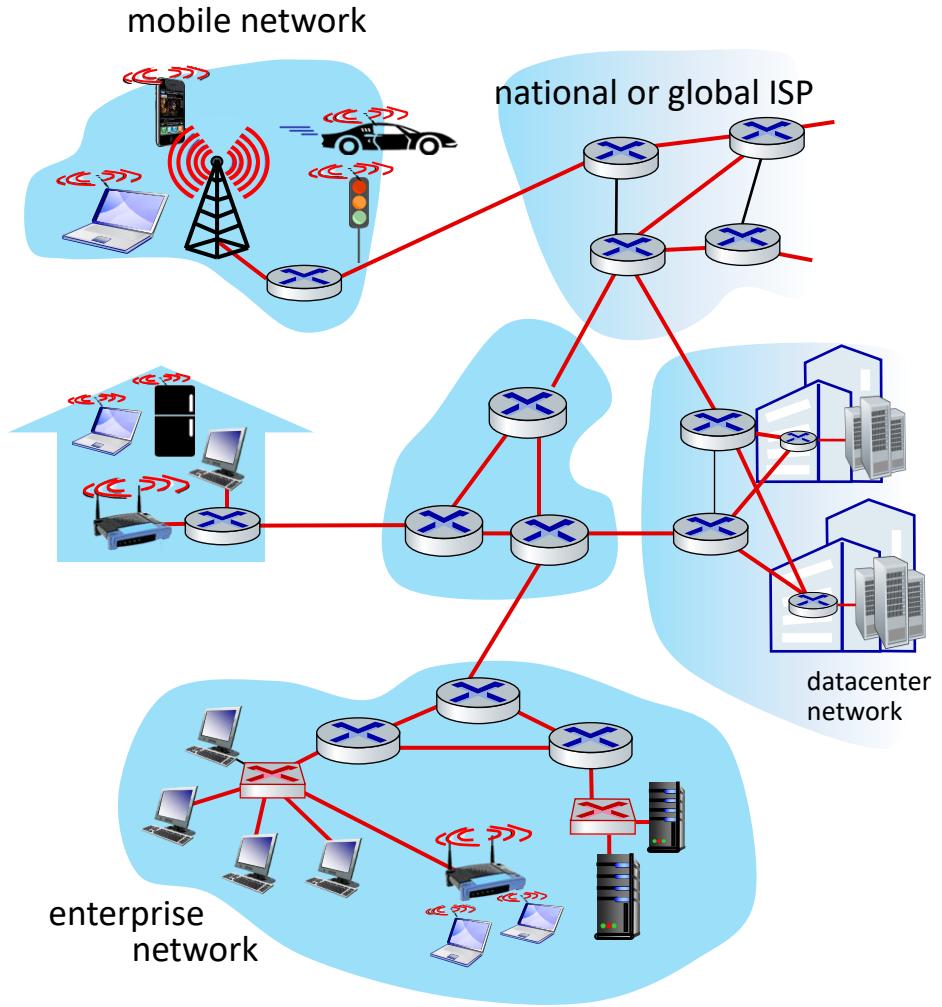
- introduction
- error detection, correction
- multiple access protocols
- LANs
  - addressing, ARP
  - Ethernet
  - switches
- data center networking
- putting it all together

# 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 a *physically adjacent* node over a link



# Link layer: context

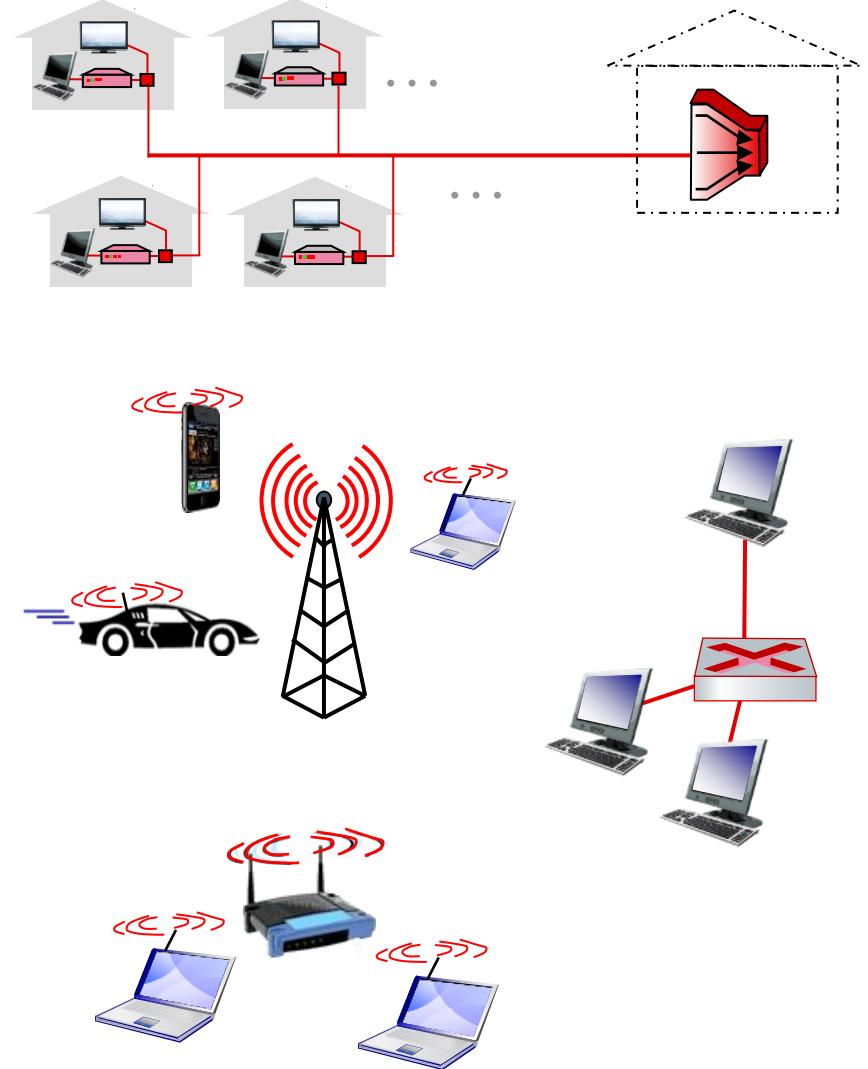
- datagram transferred by different link protocols over different links:
  - e.g., WiFi on first link, Ethernet on next link
- each link protocol provides different services
  - e.g., may or may not provide reliable data transfer over link

## transportation analogy:

- trip from Hauz Khas to Saarbrucken
  - cab: Hauz Khas to DEL
  - plane: DEL to FRA
  - train: FRA to Saarbrucken
- tourist = **datagram**
- transport segment = **communication link**
- transportation mode = **link-layer protocol**
- travel agent = **routing algorithm**

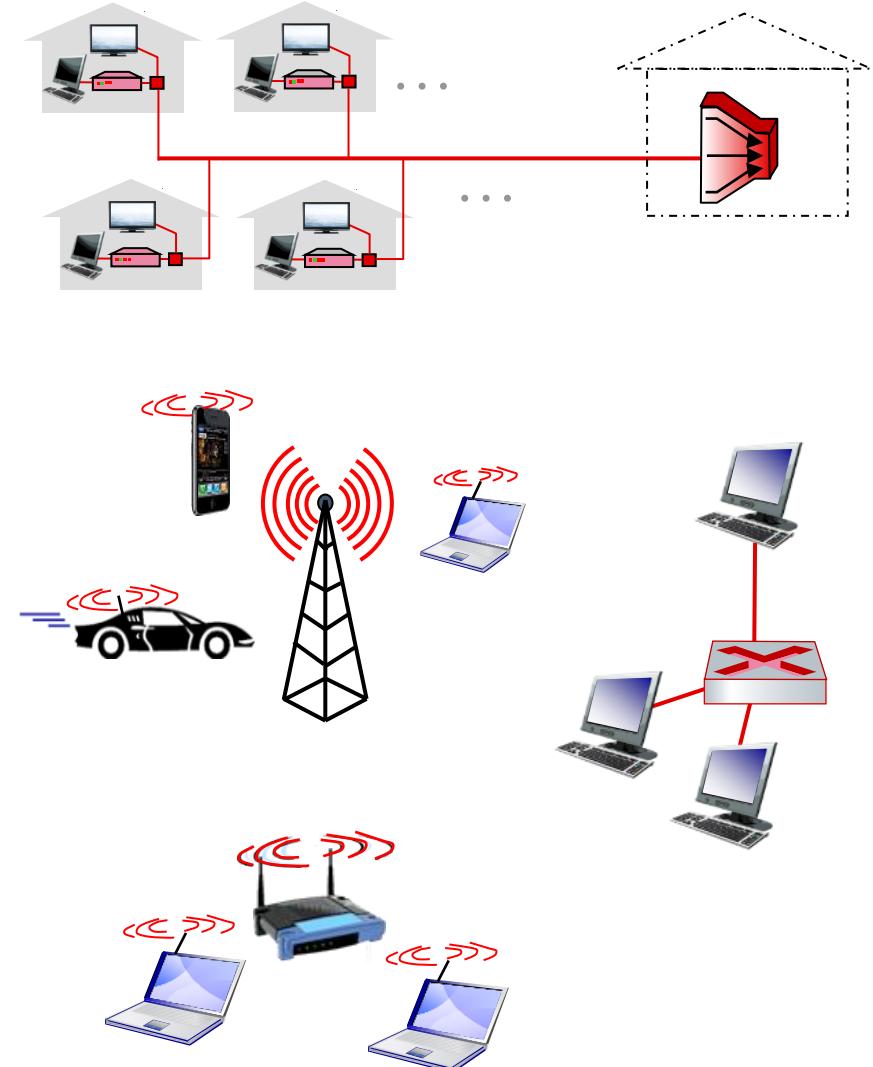
# Link layer: services

- **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?



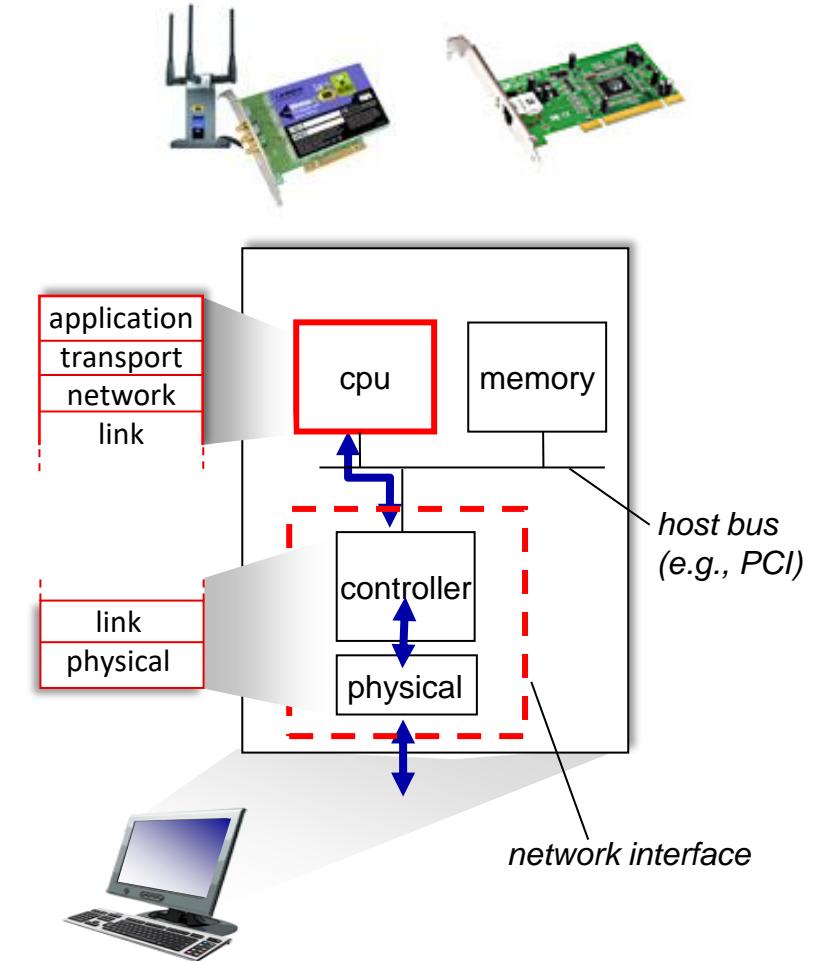
# Link layer: services (more)

- **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

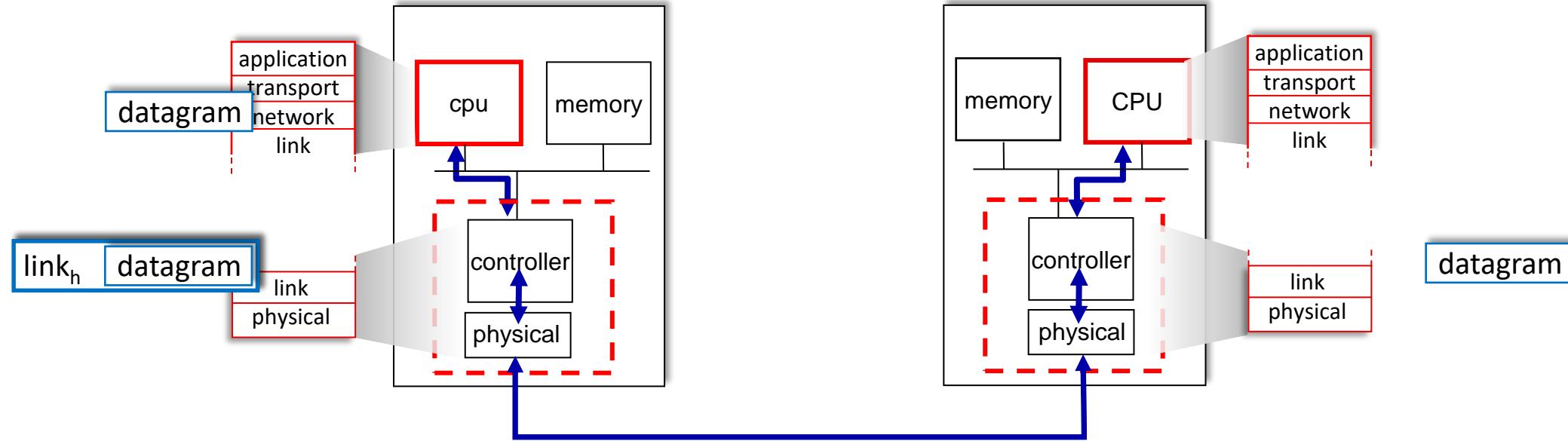


# Where is the link layer implemented?

- 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

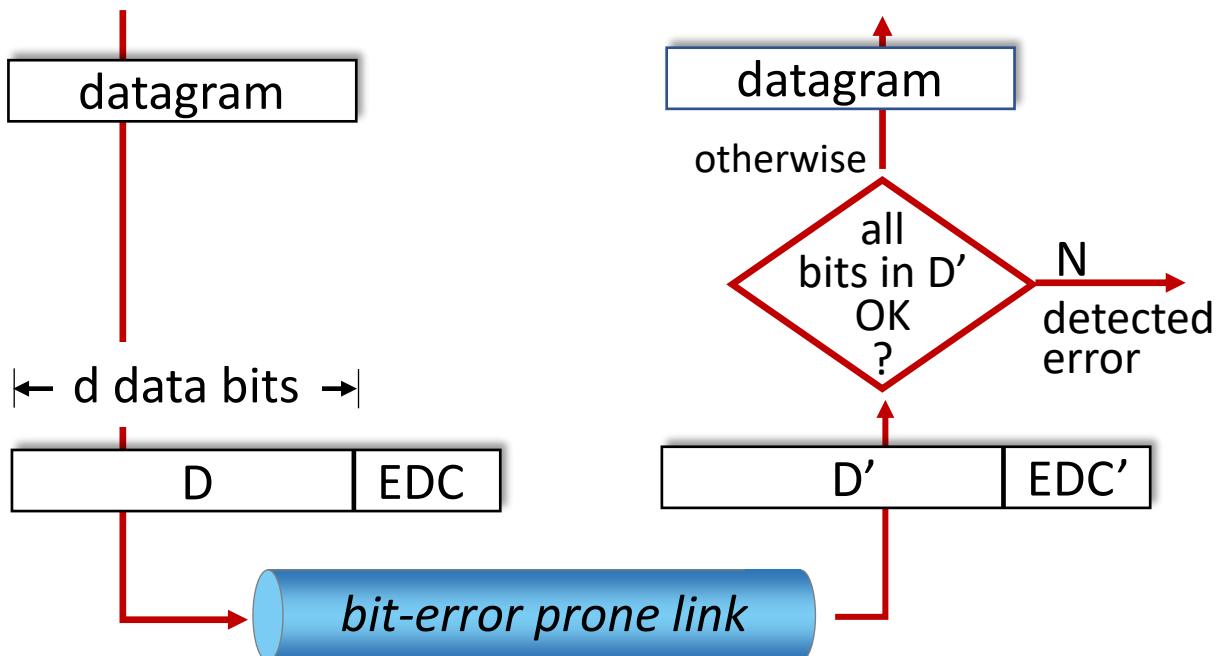
# Link layer: roadmap

- introduction
- error detection, correction
- multiple access protocols
- LANs
  - addressing, ARP
  - Ethernet
  - switches
- data center networking
- putting it all together

# Error detection

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

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



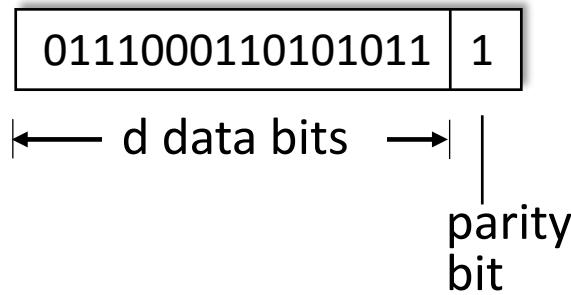
Error detection not 100% reliable!

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

# Parity checking

## single bit parity:

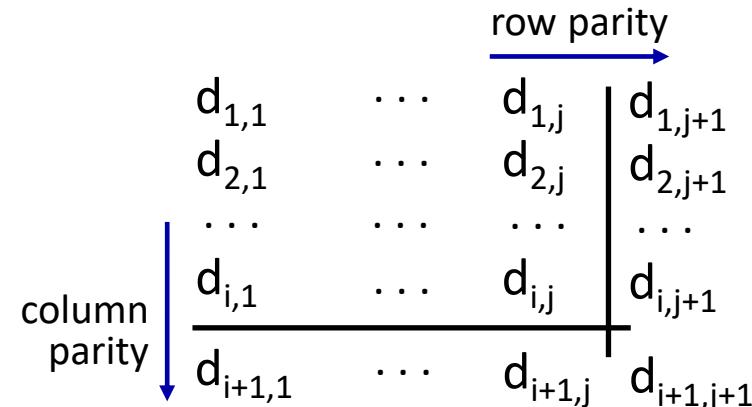
- detect single bit errors



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

## 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 |
| 0 1 1 1 0   1 |
| 1 0 1 0 1   0 |

parity error

parity error

# Internet 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?*

# 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

We want:

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

or equivalently:

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

or equivalently:

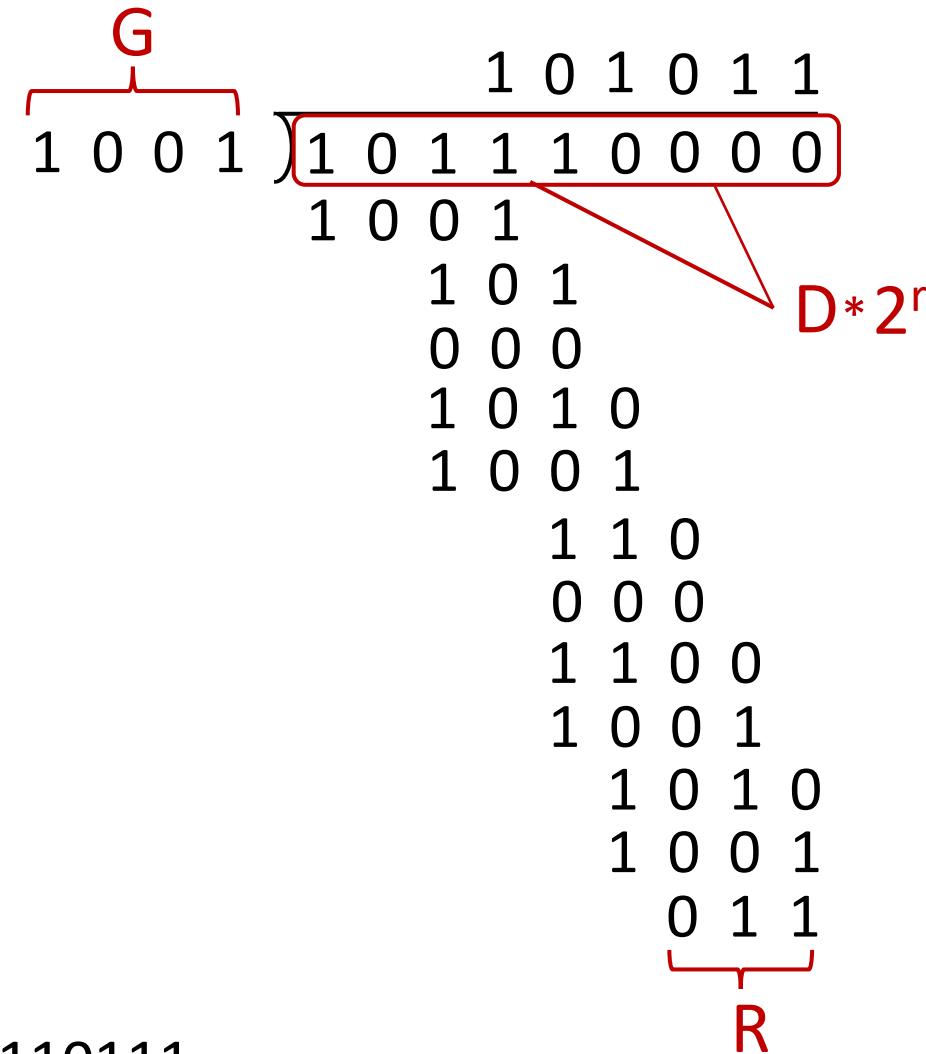
if we divide  $D \cdot 2^r$  by  $G$ , want remainder  $R$  to satisfy:

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

How to select  $G$ ?

Often some standard value is used

CRC32 = 100000100110000010001110110110111



# Link layer: roadmap

- introduction
- error detection, correction
- multiple access protocols
- LANs
  - addressing, ARP
  - Ethernet
  - switches
- data center networking
- putting it all together

# 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



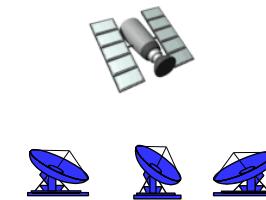
shared wire (e.g.,  
cabled Ethernet)



shared radio: 4G/5G



shared radio: WiFi



shared radio: satellite



humans at a cocktail party  
(shared air, acoustical)

# 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

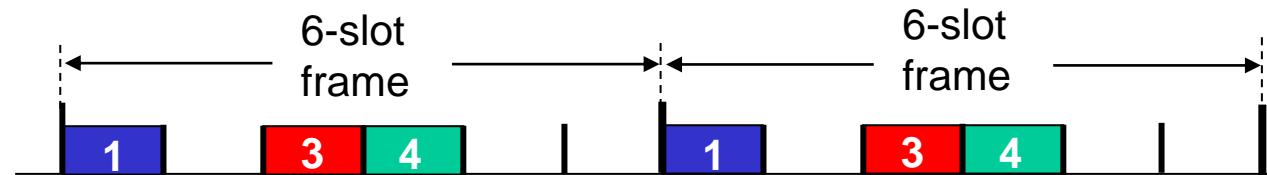
three broad classes:

- **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

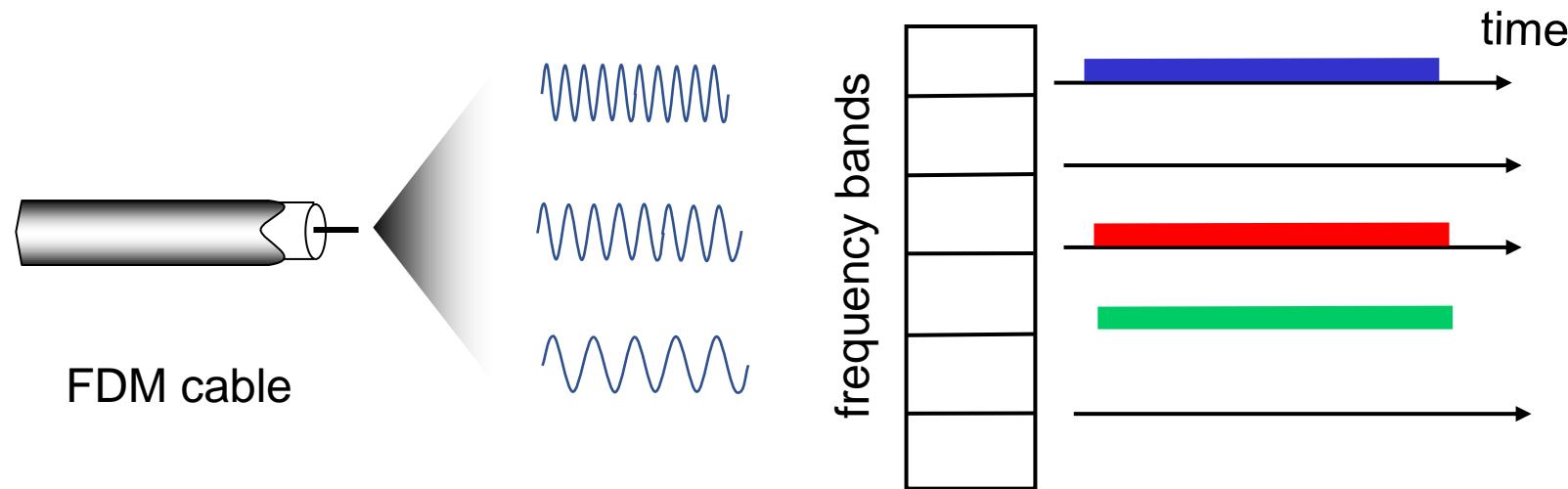
- access to channel in “rounds”
- each station gets fixed length slot (length = packet transmission time) in each round
- unused slots go idle
- example: 6-station LAN, 1,3,4 have packets to send, slots 2,5,6 idle



# 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



# Code Division Multiple Access (CDMA)

- unique “code” assigned to each user; i.e., code set partitioning
  - all users share same frequency, but each user has own “chipping” sequence (i.e., code) to encode data
  - allows multiple users to “coexist” and transmit simultaneously with minimal interference (if codes are “orthogonal”)
- **encoding:** original data XOR chipping sequence
- **decoding:** encoded data XOR chipping sequence
- Above is a simplified scheme. Actual CDMA used in wireless communication uses spread spectrum techniques.
- Code assignment is complex. See for example  
<https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1453>

# 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:

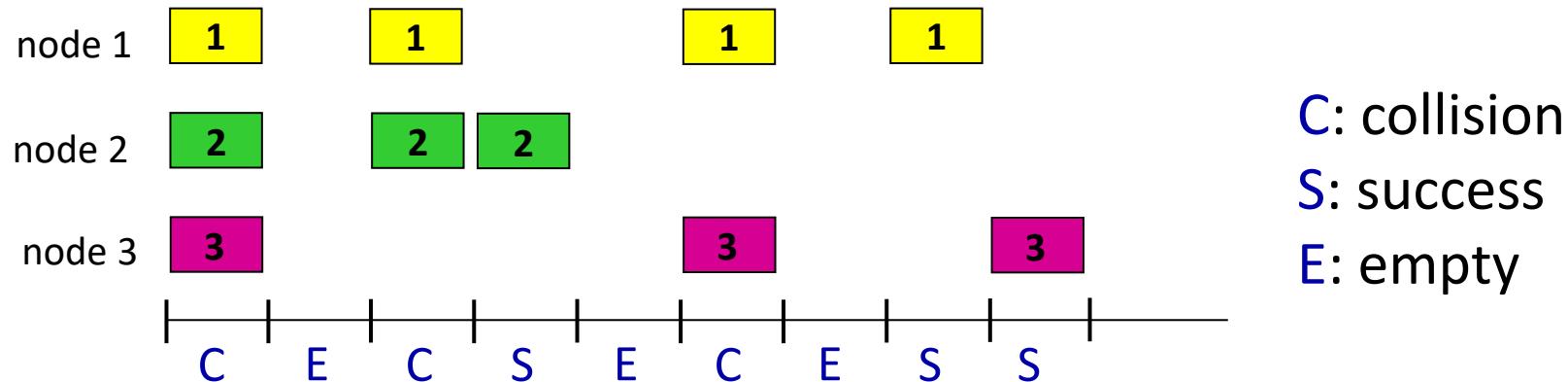
- all frames are of same size
- time divided into equal size slots (time to transmit 1 frame)
- nodes start to transmit only slot beginning
- nodes are synchronized
- if 2 or more nodes transmit in slot, all nodes detect collision

## 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

randomization – *why?*

# 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: Throughput

Let  $T$  be the frame time, i.e., the time required for 1 frame to be transmitted.

Let  $G$  be the number of transmission attempts per frame time.

The probability that  $k$  frames are generated during the frame time is given by the Poisson distribution:  $P(k) = G^k e^{-G} / k!$

The probability that 0 frames are generated ( $k = 0$ ) during the frame time is  $e^{-G}$

For slotted ALOHA, the vulnerable time period for collision between two frames is the time duration of 1 slot = 1 frame time  $T$ .

In  $T$  time, average number of transmission attempts is  $G$ .

The probability that 0 frames are initiated in the vulnerable time period will be  $P(0) = e^{-G}$

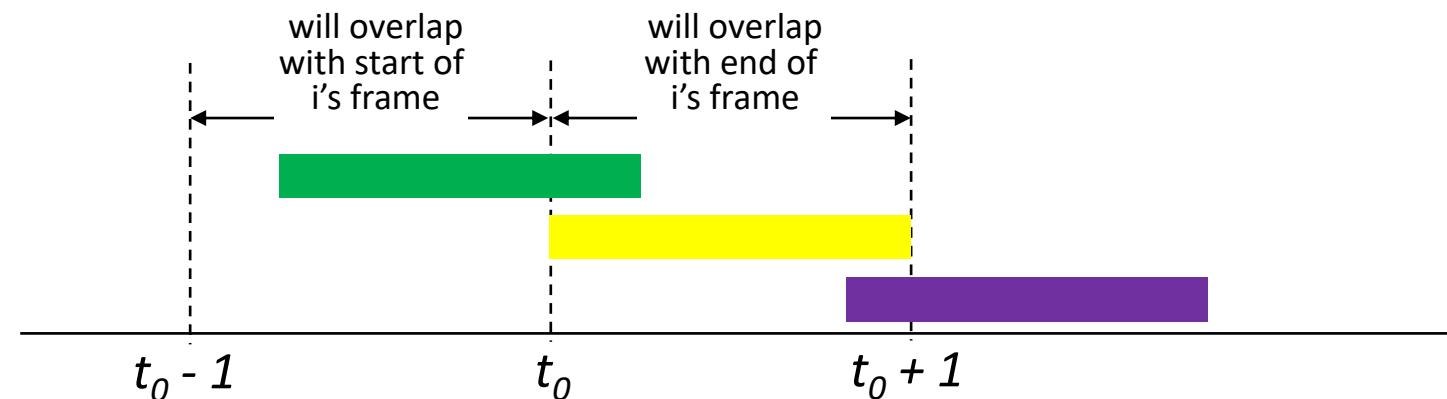
The throughput,  $S$ , is calculated as the number of transmission attempts per frame time,  $G$ , multiplied by the probability of success,  $P(0)$ :  $S = G.P(0)$  or  $S = Ge^{-G}$

Maximum throughput of Slotted ALOHA occurs when  $G = 1$ .

The maximum throughput is thus  $S_{\max} = 1 \times e^{-1} = 1/e = 0.368$

# 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% !

# CSMA (carrier sense multiple access)

simple **CSMA**: listen before transmit:

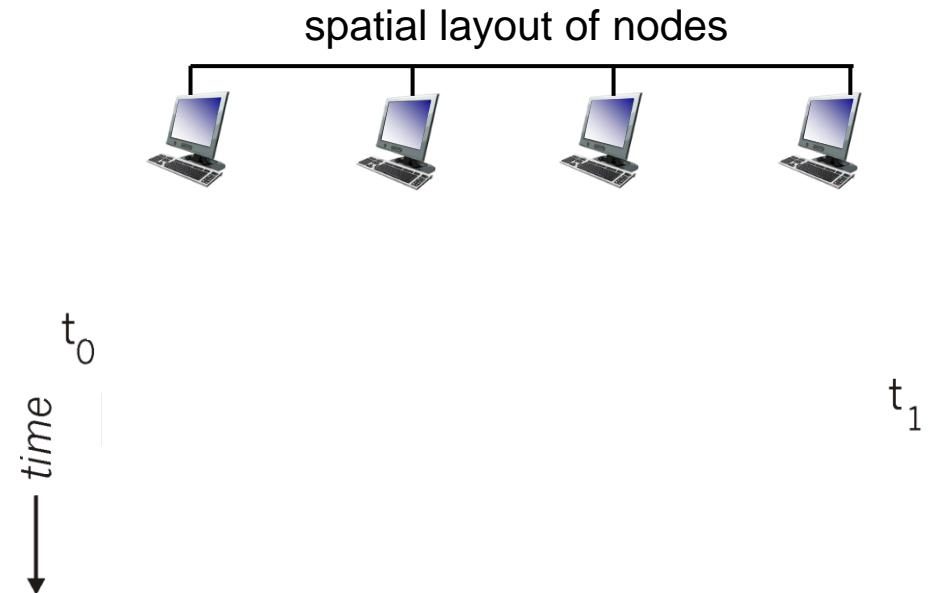
- if channel sensed idle: transmit entire frame
- if channel sensed busy: defer transmission
- human analogy: don't interrupt others!

**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

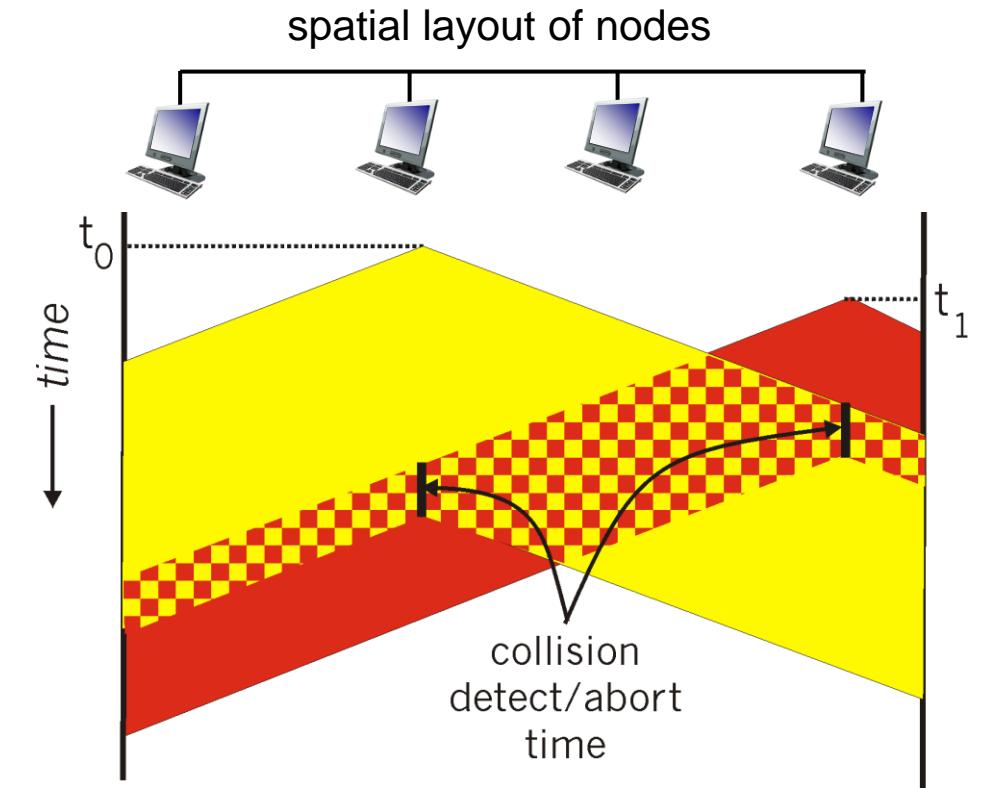
# CSMA: collisions

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



# 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

# CSMA/CD efficiency

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

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

- See [https://www.sce.carleton.ca/faculty/lambadaris/courses/462/csma\\_cd.pdf](https://www.sce.carleton.ca/faculty/lambadaris/courses/462/csma_cd.pdf) for proof
- 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

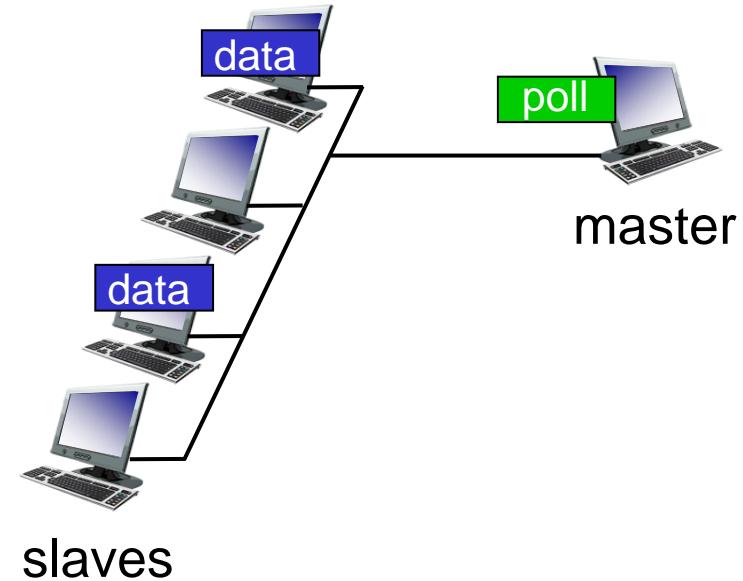
## “taking turns” protocols

- look for best of both worlds!

# “Taking turns” MAC protocols

## polling:

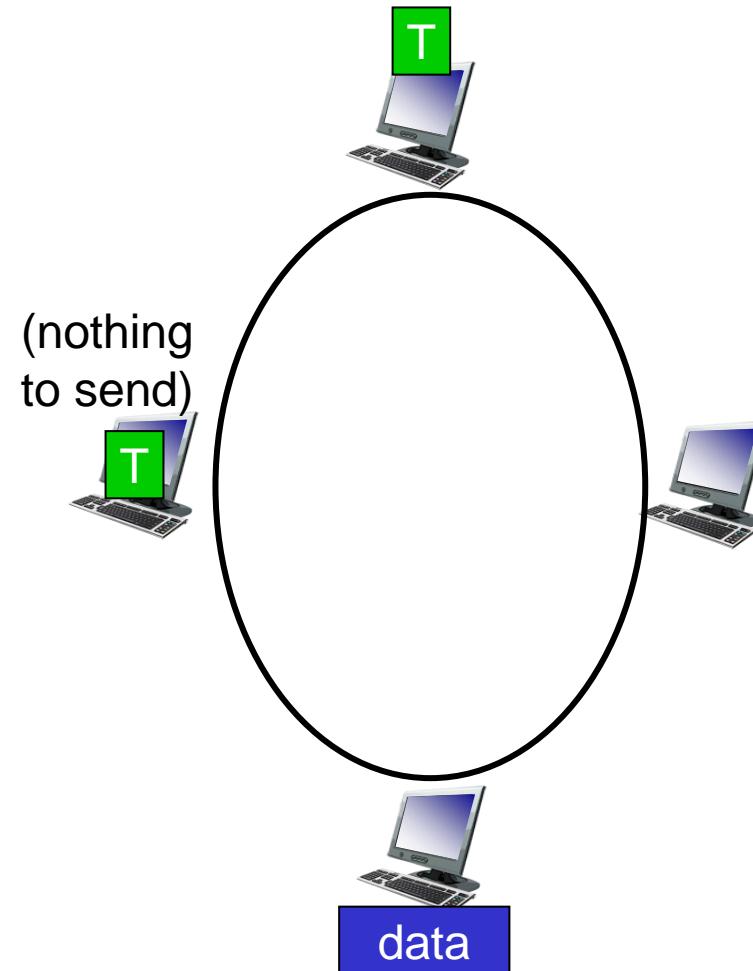
- master node “invites” other nodes to transmit in turn
- transmits if there is data to send
- concerns:
  - polling overhead
  - latency
  - single point of failure (master)



# “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)

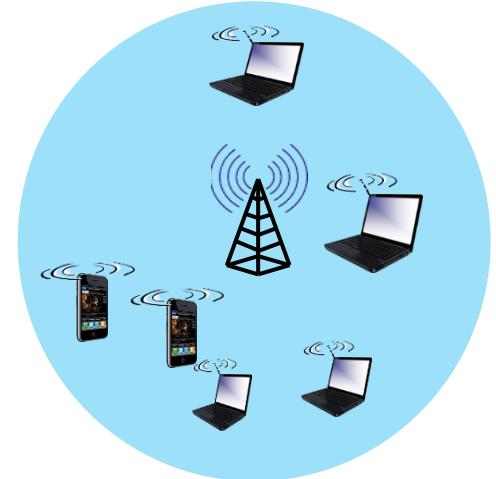


# Wireless link characteristics

*important* differences from wired link ....

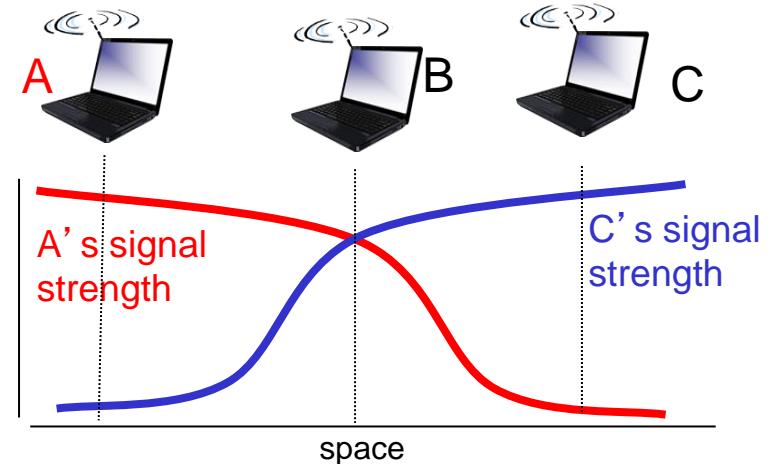
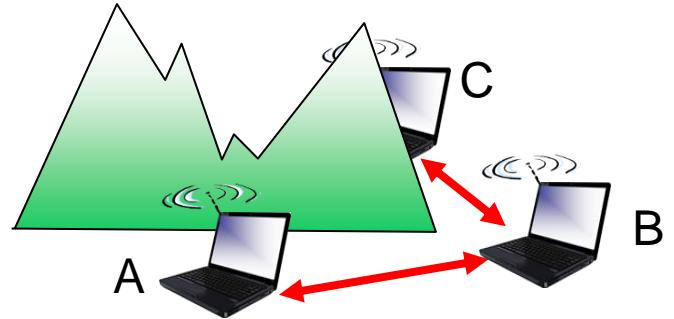
- **decreased signal strength:** radio signal attenuates as it propagates through matter (path loss)
- **interference from other sources:** wireless network frequencies (e.g., 2.4 GHz) shared by many devices (e.g., WiFi, cellular, motors): interference
- **multipath propagation:** radio signal reflects off objects ground, arriving at destination at slightly different times

.... make communication across (even a point to point) wireless link much more “difficult”



# Wireless link characteristics

Multiple wireless senders, receivers create additional problems (beyond multiple access):



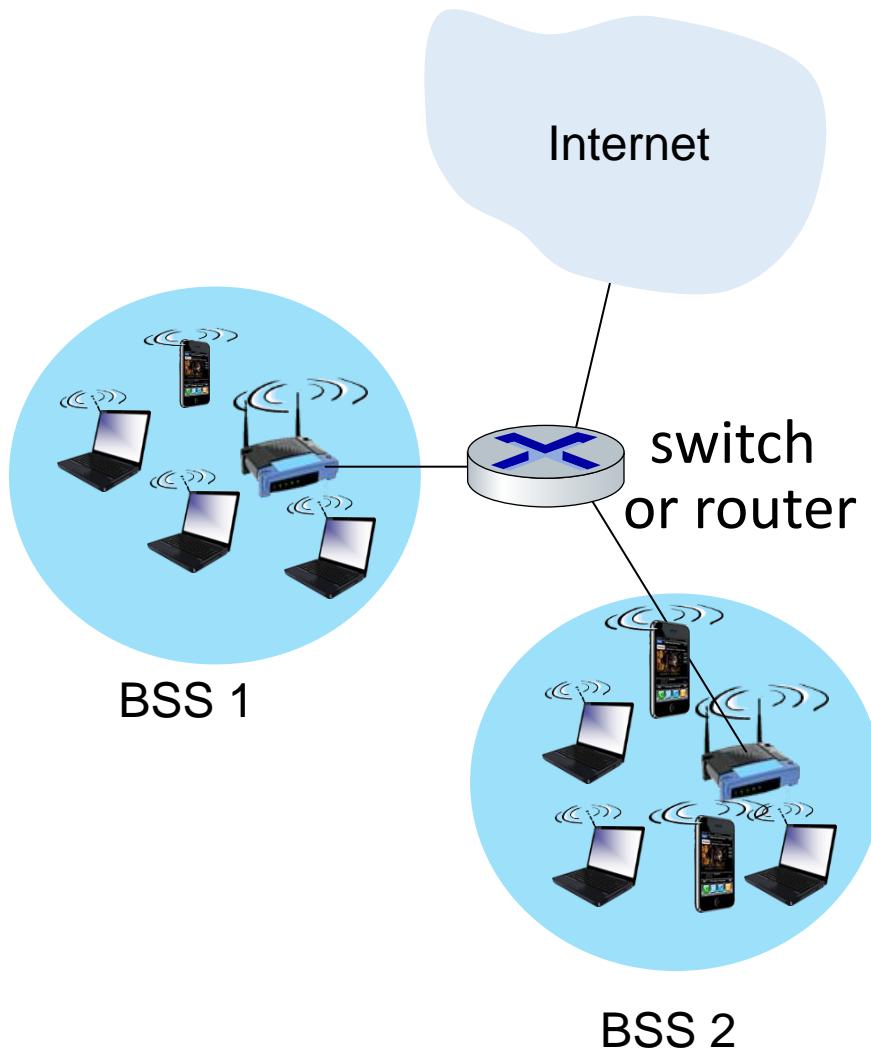
## Hidden terminal problem

- B, A hear each other
- B, C hear each other
- A, C can not hear each other means A, C unaware of their interference at B

## Signal attenuation:

- B, A hear each other
- B, C hear each other
- A, C can not hear each other interfering at B

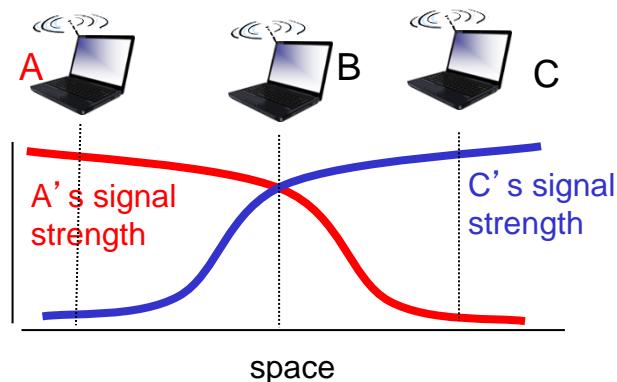
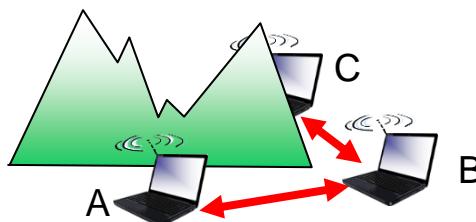
# 802.11 LAN/ WiFi architecture



- wireless host communicates with base station
  - base station = access point (AP)
- **Basic Service Set (BSS) (aka “cell”)** in infrastructure mode contains:
  - wireless hosts
  - access point (AP): base station
  - ad hoc mode: hosts only

# IEEE 802.11: multiple access

- avoid collisions:  $2^+$  nodes transmitting at same time
- 802.11: CSMA - sense before transmitting
  - don't collide with detected ongoing transmission by another node
- 802.11: *no collision detection!*
  - difficult to sense collisions: high transmitting signal, weak received signal due to fading
  - can't sense all collisions in any case: hidden terminal, fading
  - goal: *avoid collisions:* CSMA/CollisionAvoidance



# IEEE 802.11 MAC Protocol: CSMA/CA

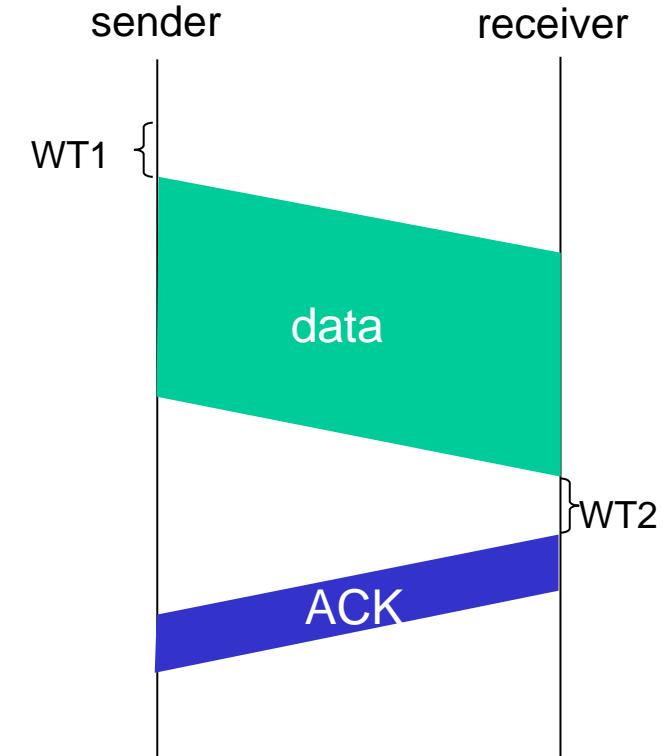
## 802.11 sender

1 if sense channel idle for time WT1 (called DIFS)  
then transmit entire frame (no CD)

2 if sense channel busy then  
start random backoff time  
timer counts down while channel idle  
transmit when timer expires  
if no ACK, increase random backoff interval,  
repeat 2

## 802.11 receiver

if frame received OK  
return ACK after WT2 (called SIFS)  
(ACK needed due to hidden terminal problem)

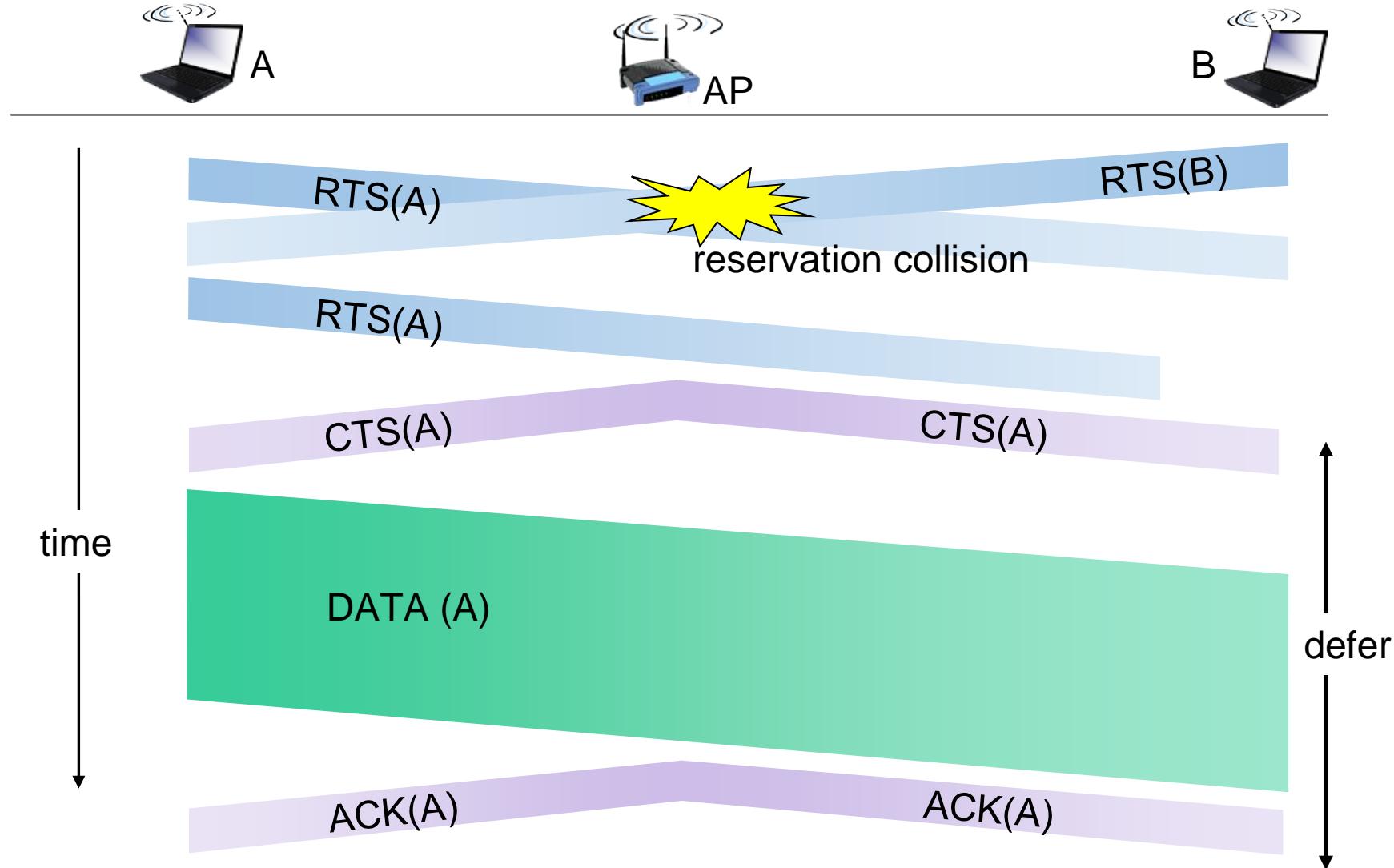


# Avoiding collisions

**idea:** sender “reserves” channel use for data frames using small reservation packets

- sender first transmits *small* request-to-send (RTS) packet to BS using CSMA
  - RTSs may still collide with each other (but they’re short)
- BS broadcasts clear-to-send CTS in response to RTS
- CTS heard by all nodes
  - sender transmits data frame
  - other stations defer transmissions

# Collision Avoidance: RTS-CTS exchange



# Link layer: roadmap

- introduction
- error detection, correction
- multiple access protocols
- LANs
  - addressing, ARP
  - Ethernet
  - switches
- data center networking
- putting it all together

# MAC addresses

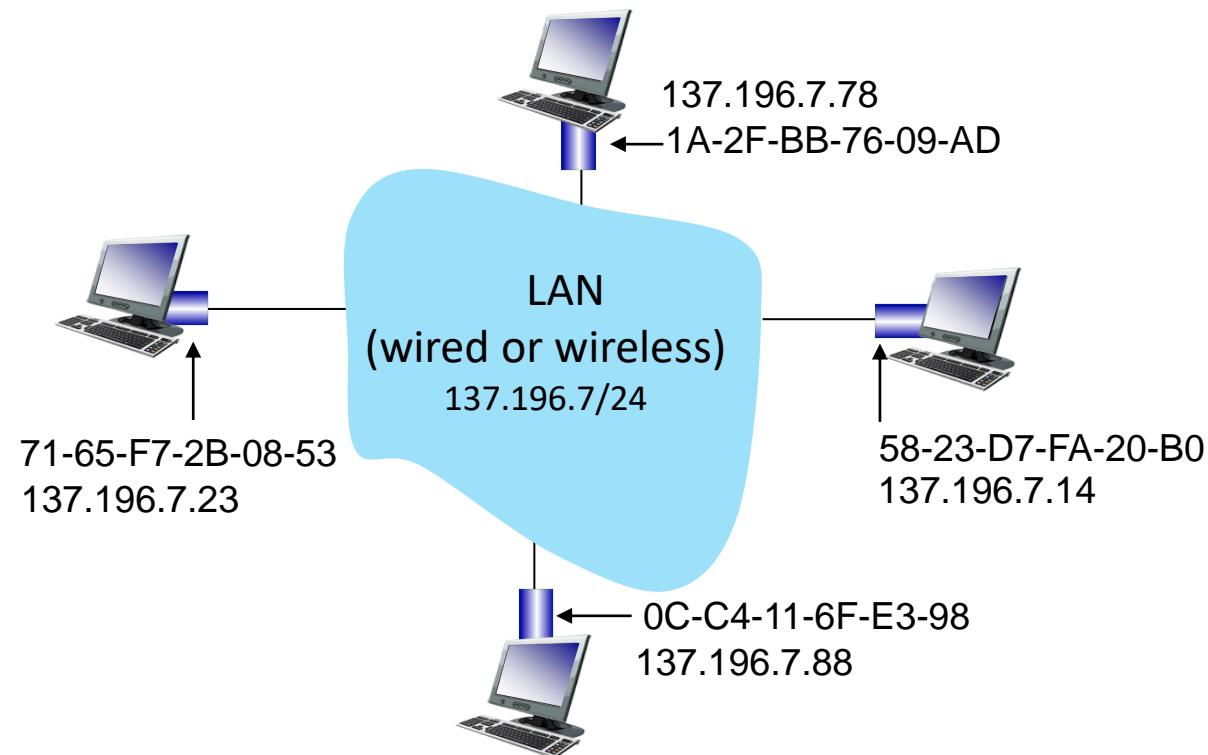
- 32-bit IP address:
  - *network-layer* address for interface
  - used for layer 3 (network layer) forwarding
  - e.g.: 128.119.40.136
- MAC (or LAN or physical or Ethernet) address:
  - function: used “locally” to get frame from one interface to another physically-connected interface (same subnet, 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

*hexadecimal (base 16) notation  
(each “numeral” represents 4 bits)*

# MAC addresses

each interface on LAN

- has a unique 48-bit **MAC address**
- has a locally unique 32-bit IP address (as we've seen)

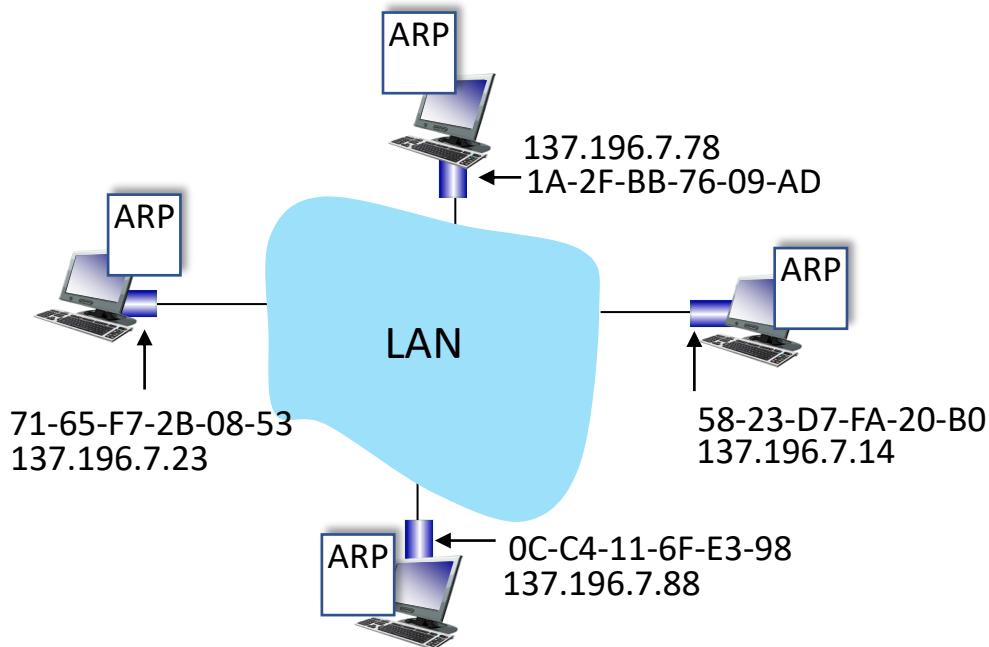


# MAC addresses

- MAC address allocation administered by IEEE
- manufacturer buys portion of MAC address space (to assure uniqueness)
- analogy:
  - MAC address: like Social Security Number/Aadhar
  - IP address: like postal address
- MAC flat address: portability
  - can move interface from one LAN to another
  - IP address *not* portable: depends on IP subnet to which node is attached
- Random Mac address
  - See <https://www.mathyvanhoef.com/2016/03/how-mac-address-randomization-works-on.html> for details

# 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)

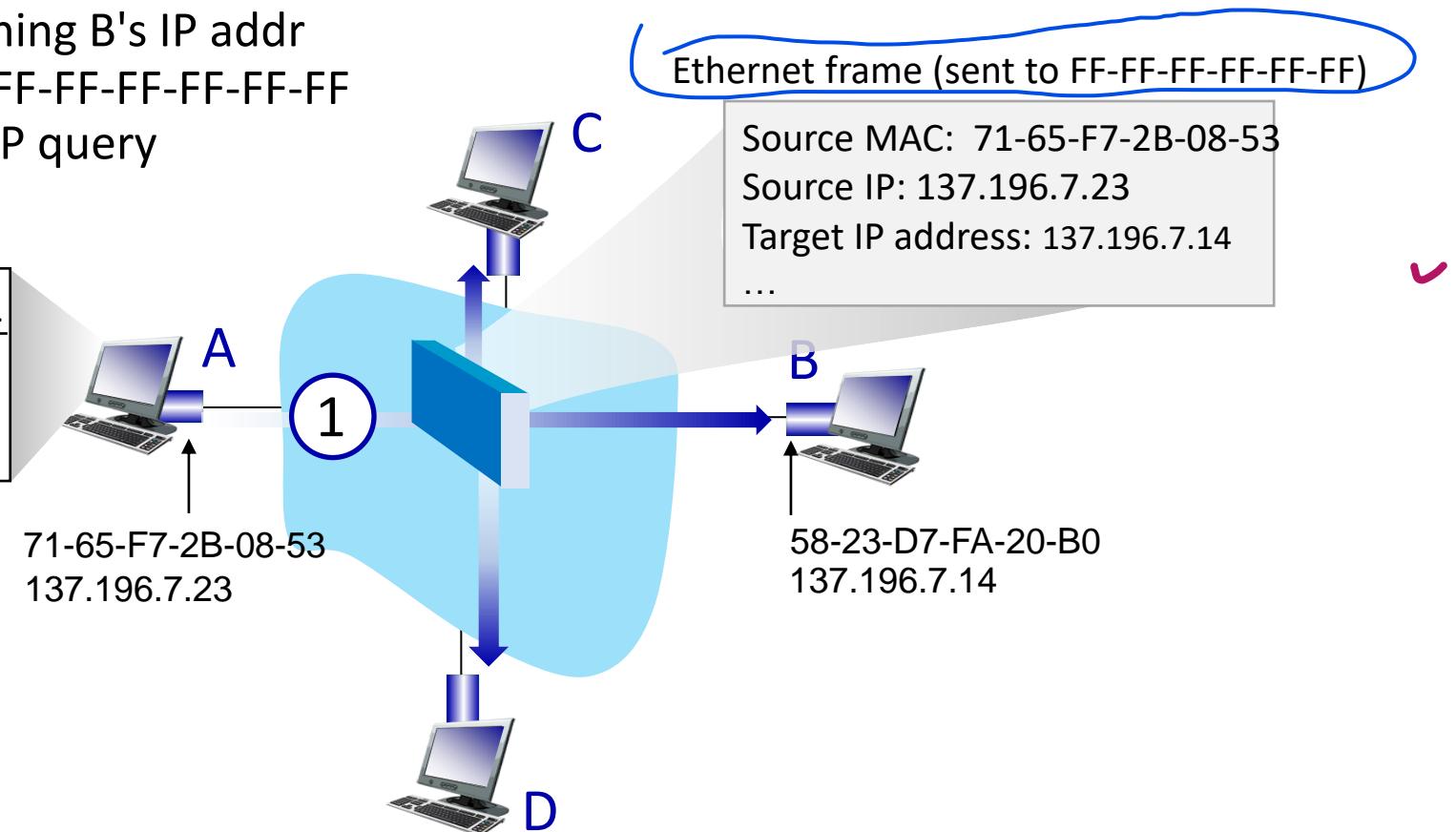
# ARP protocol in action

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

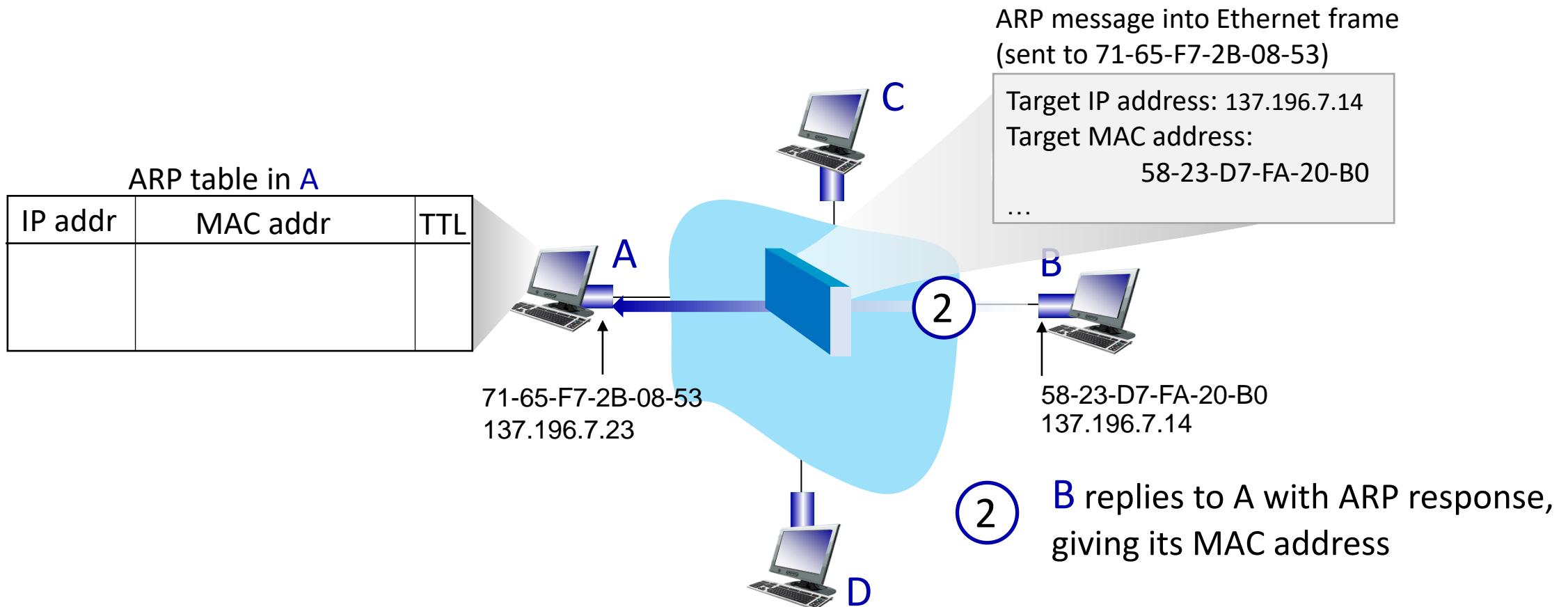
| ARP table in A |          |     |
|----------------|----------|-----|
| IP addr        | MAC addr | TTL |
|                |          |     |



# ARP protocol in action

example: A wants to send datagram to B

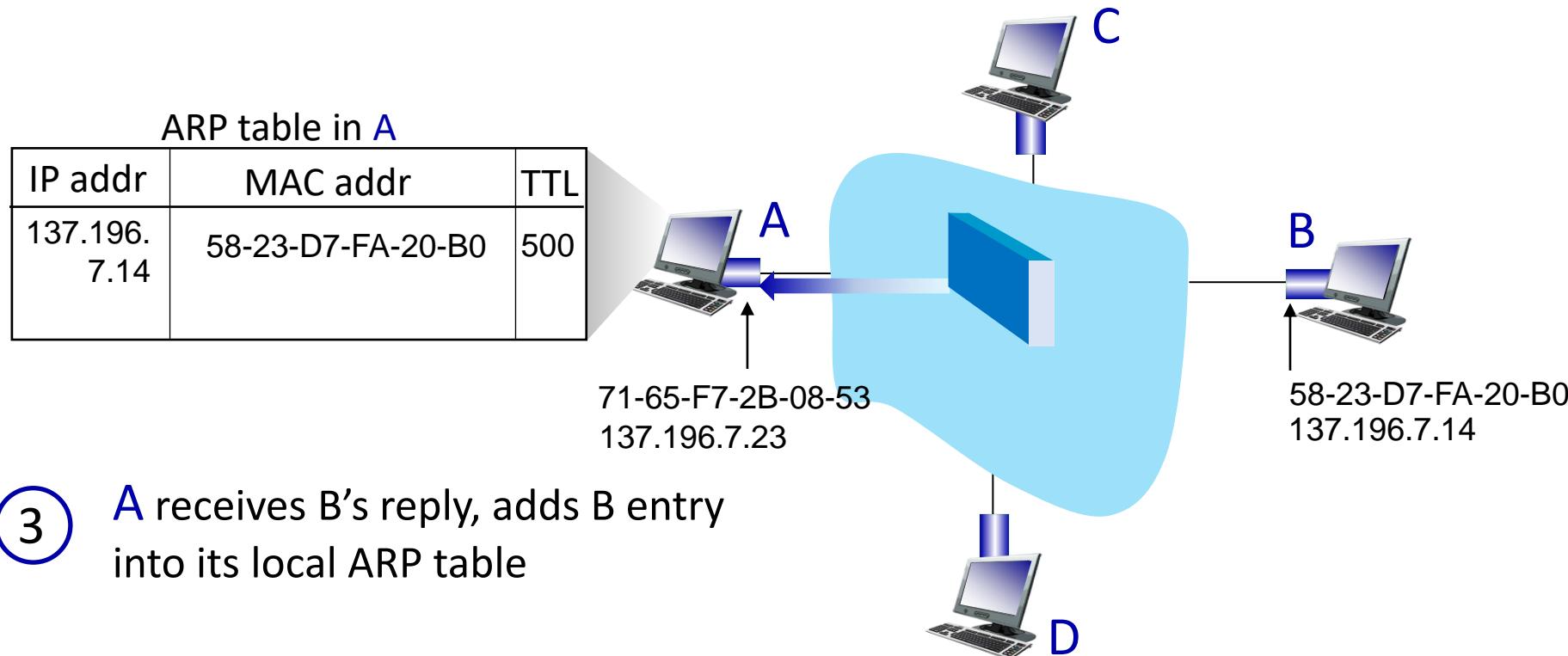
- B's MAC address not in A's ARP table, so A uses ARP to find B's MAC address



# ARP protocol in action

example: A wants to send datagram to B

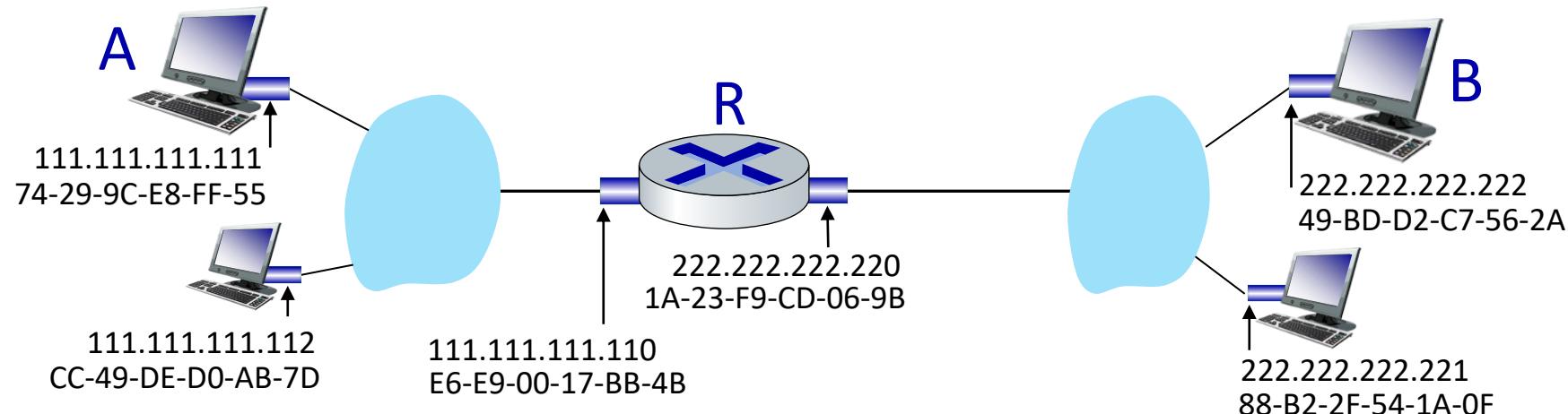
- B's MAC address not in A's ARP table, so A uses ARP to find B's MAC address



# Routing to another subnet: addressing

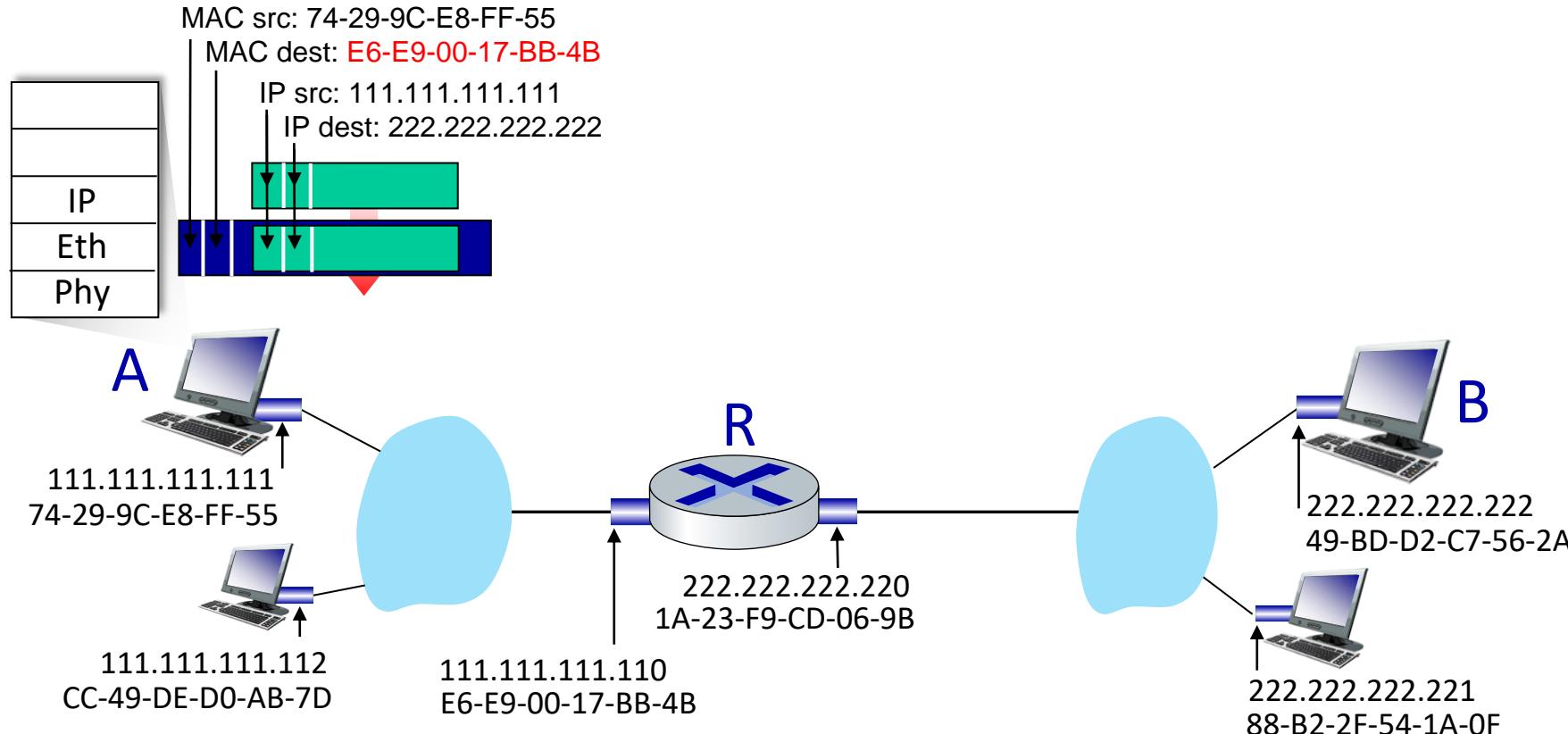
walkthrough: sending a datagram from A to B via R

- 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 (how?) ✓ DHCP
  - A knows R's MAC address (how?). Address resolution protocol



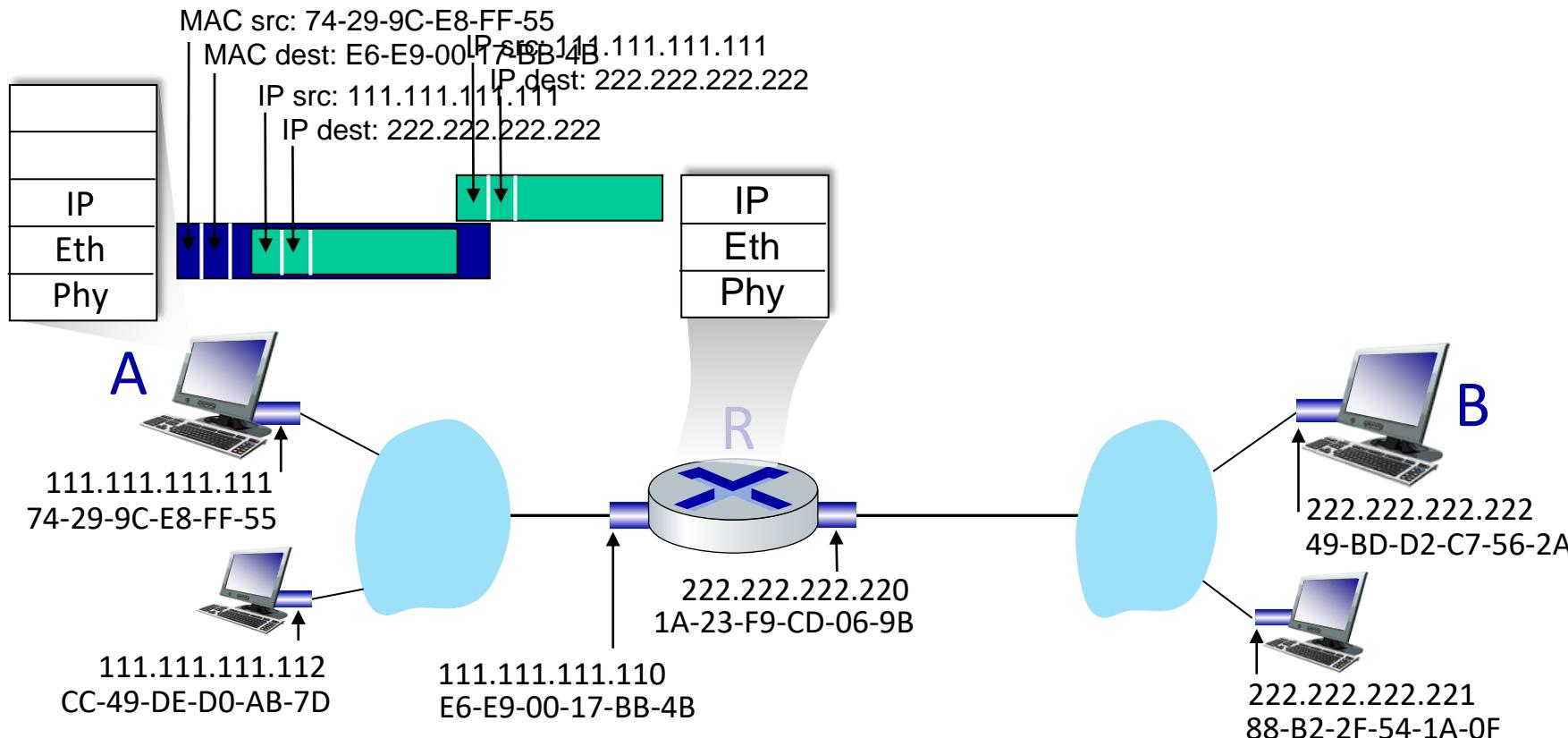
# Routing to another subnet: addressing

- 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



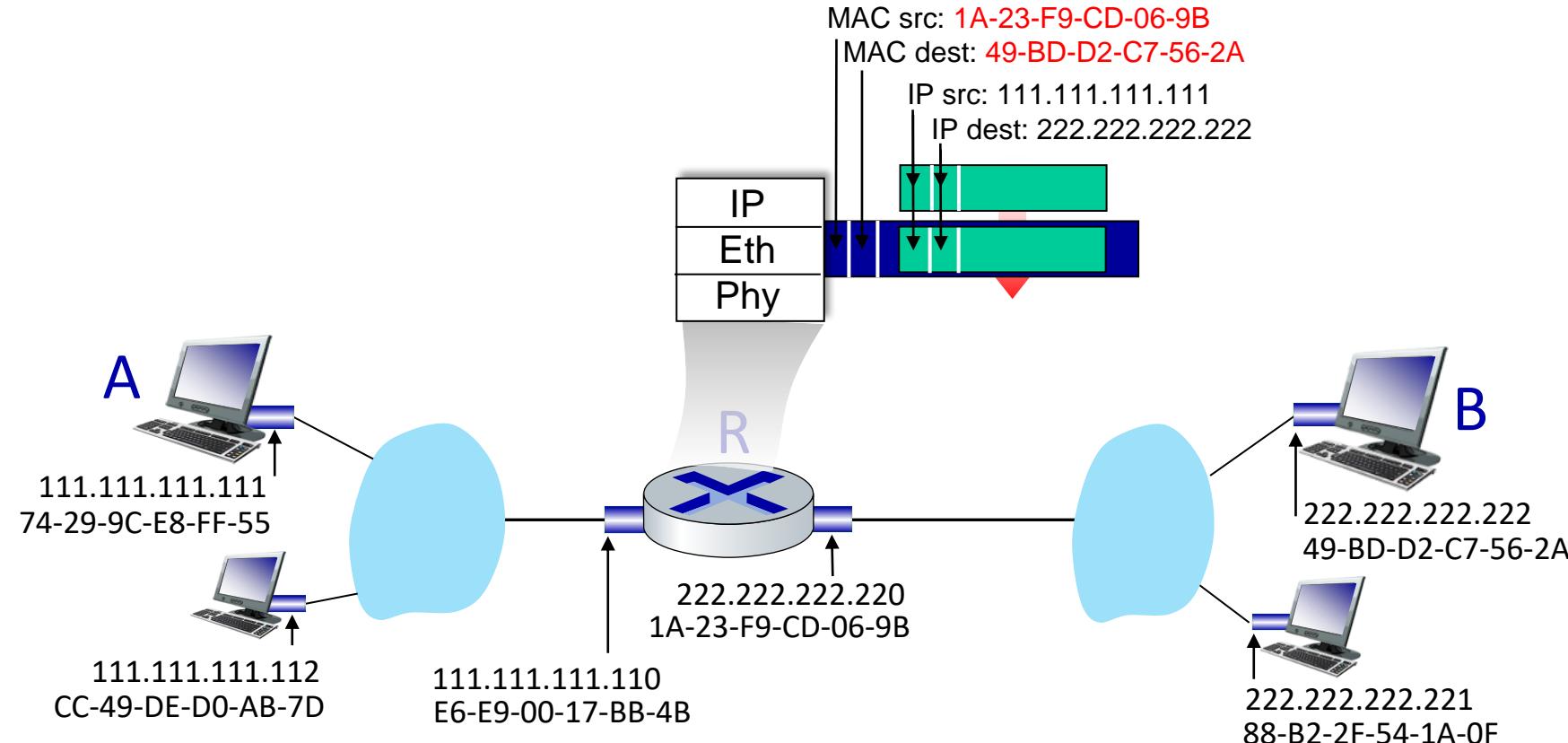
# Routing to another subnet: addressing

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



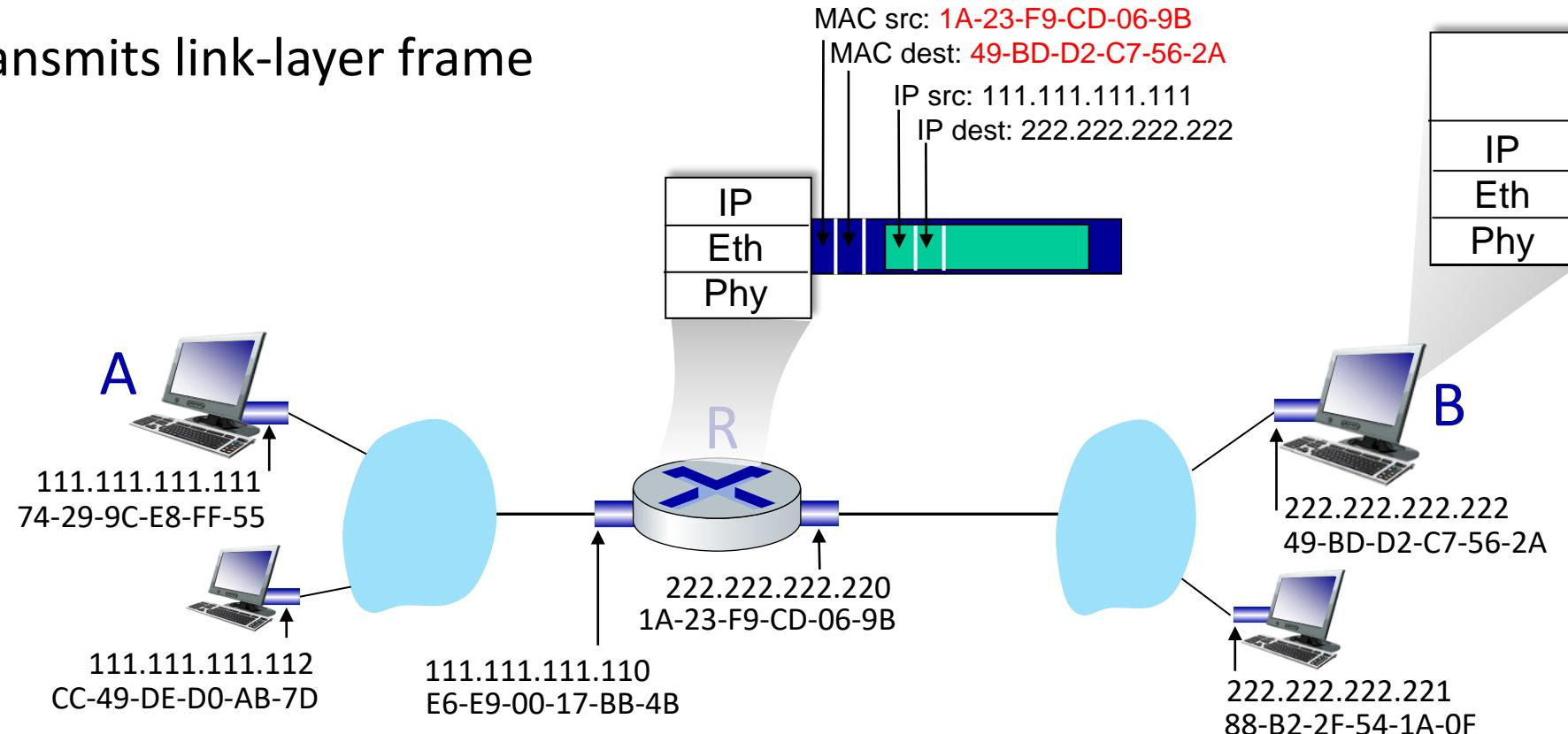
# Routing to another subnet: addressing

- 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



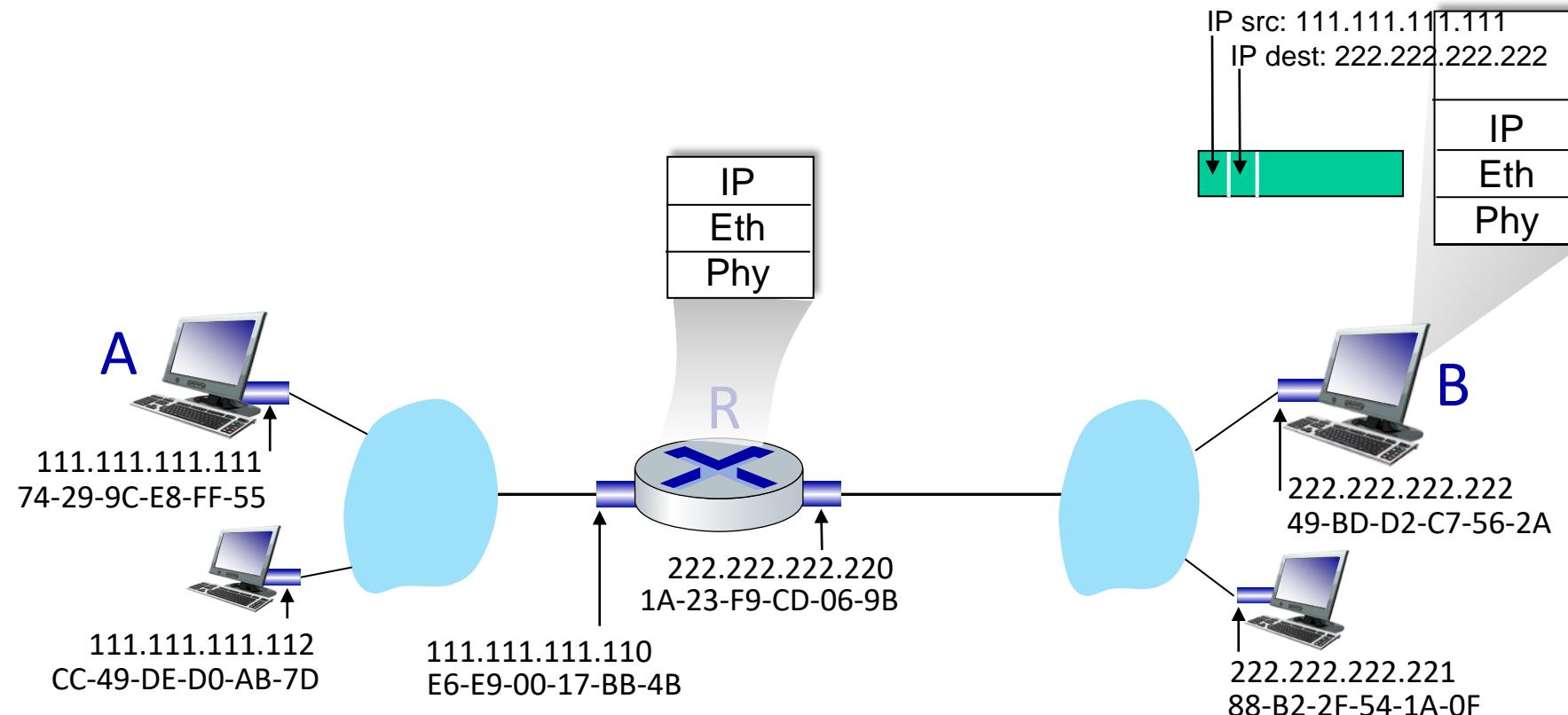
# Routing to another subnet: addressing

- 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**
  - switches
- data center networking
- putting it all together

# 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)

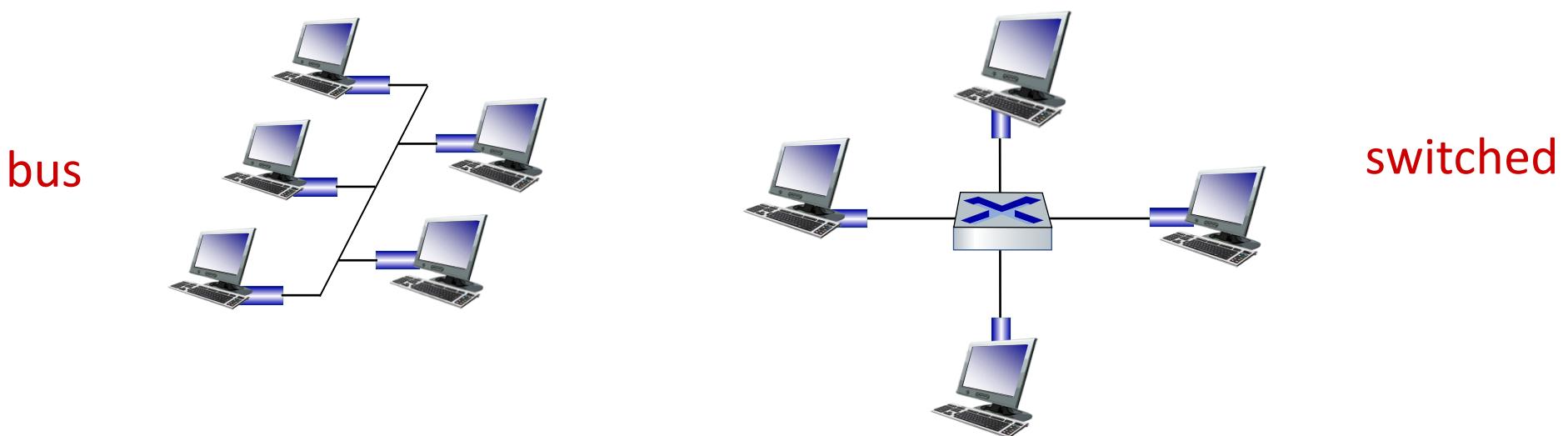
# Ethernet: physical topology

✓ bus: popular through mid 90s

- all nodes in same collision domain (can collide with each other)

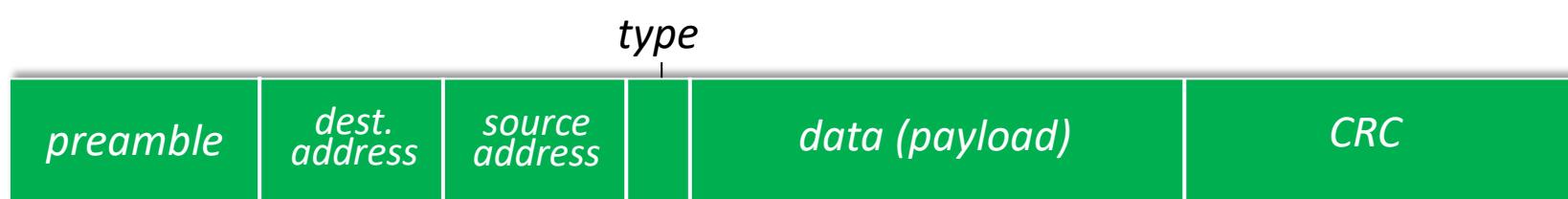
✓ switched: prevails today

- active link-layer 2 *switch* in center
- each “spoke” runs a (separate) Ethernet protocol (nodes do not collide with each other)



# Ethernet frame structure

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

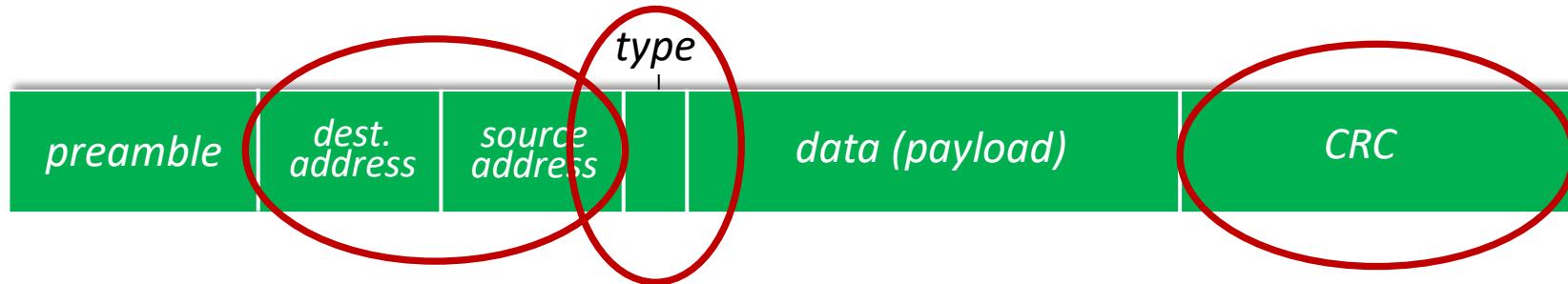


*preamble:*



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

# Ethernet frame structure (more)



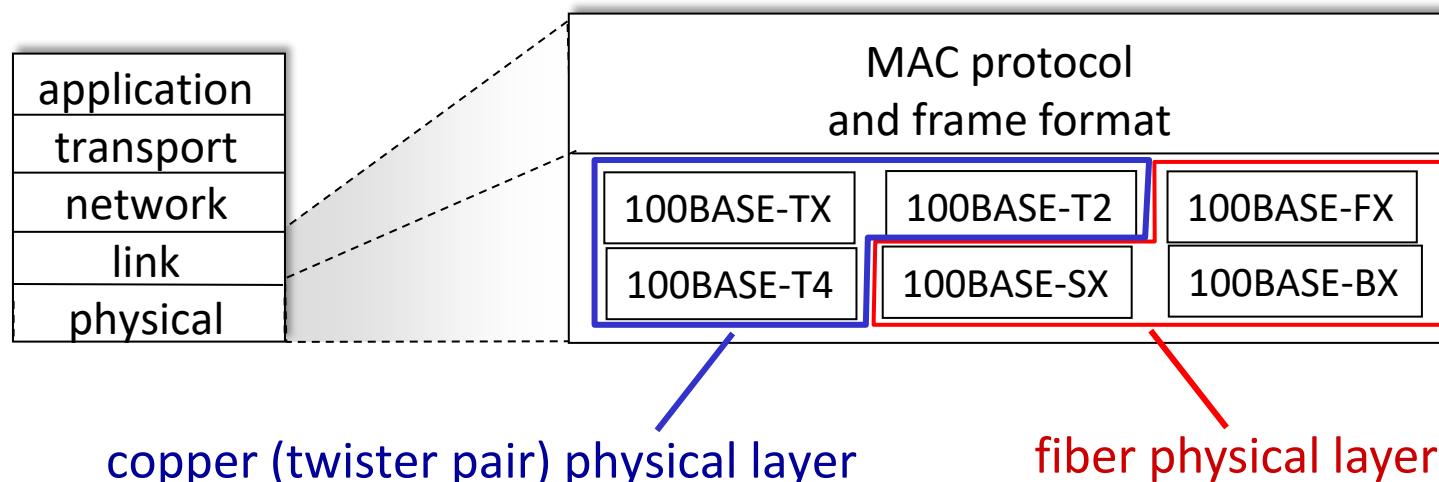
- **addresses**: 6 byte source, destination MAC addresses
  - if adapter receives frame with matching destination address, or with broadcast address (e.g., ARP packet), it passes data in frame to network layer protocol
  - otherwise, adapter discards frame
- **type**: indicates higher layer protocol
  - 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

# 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



# Link layer, LANs: roadmap

- introduction
- error detection, correction
- multiple access protocols
- **LANs**
  - addressing, ARP
  - Ethernet
  - **switches**
- data center networking
- putting it all together

# Ethernet switch

- Switch is a **link-layer** device: takes an *active* role
  - store, forward Ethernet frames
  - examine incoming frame's MAC address, *selectively* forward frame to one-or-more outgoing links when frame is to be forwarded on segment, uses CSMA/CD to access segment

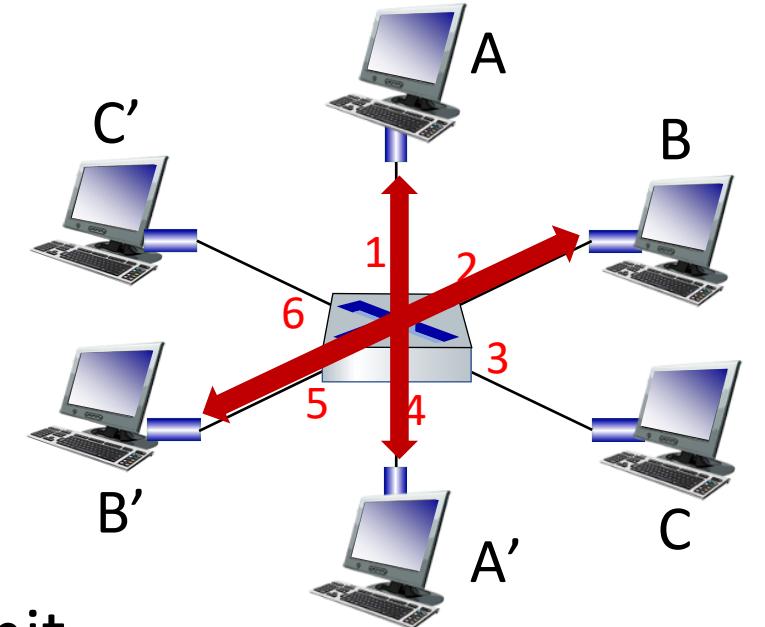
✓ **transparent:** hosts *unaware* of presence of switches

✓ **plug-and-play, self-learning**

- switches do not need to be configured

# Switch: multiple simultaneous transmissions

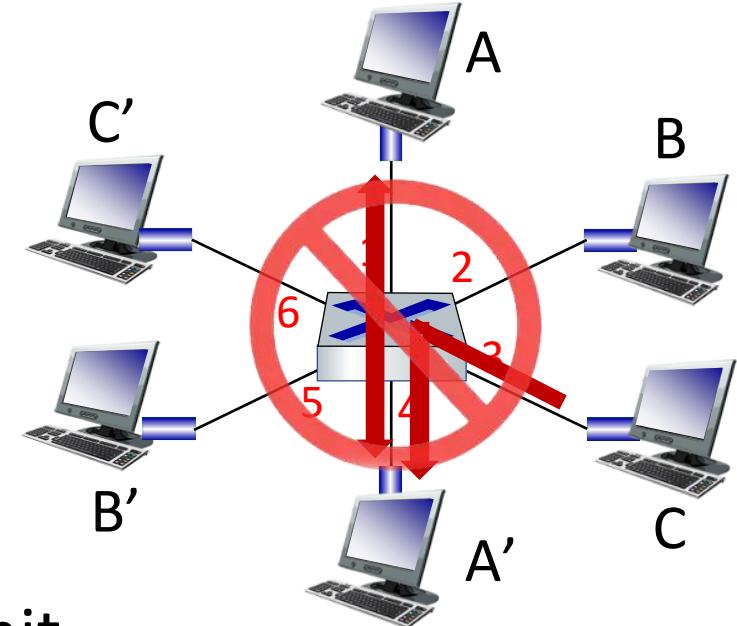
- hosts have dedicated, direct connection to switch
- switches buffer packets
- Ethernet protocol used on *each* incoming link, so:
  - no collisions; full duplex ✓
  - each link is its own collision domain
- **switching:** A-to-A' and B-to-B' can transmit simultaneously, without collisions



switch with six  
interfaces (1,2,3,4,5,6)

# Switch: multiple simultaneous transmissions

- hosts have dedicated, direct connection to switch
- switches buffer packets
- Ethernet protocol used on *each* incoming link, so:
  - no collisions; full duplex
  - each link is its own collision domain
- **switching:** A-to-A' and B-to-B' can transmit simultaneously, without collisions
  - but A-to-A' and C to A' can *not* happen simultaneously



switch with six  
interfaces (1,2,3,4,5,6)

# Switch forwarding table

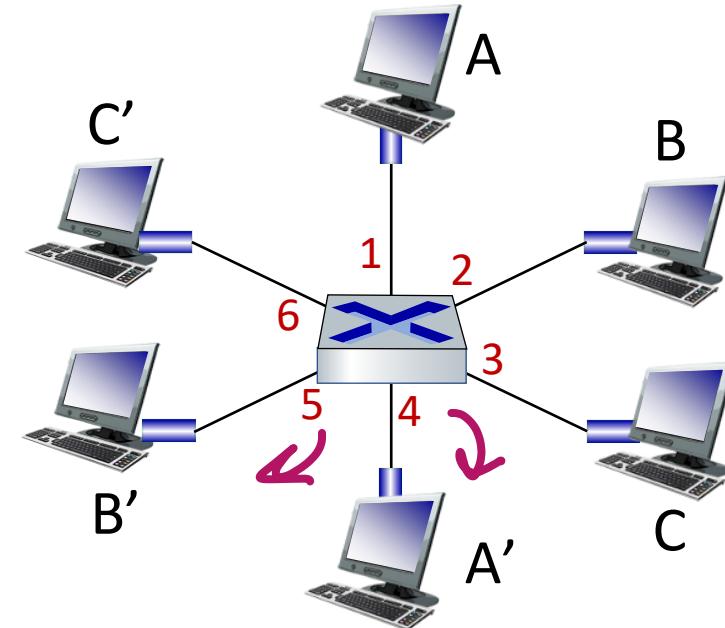
**Q:** how does switch know A' reachable via interface 4, B' reachable via interface 5?

**A:** each switch has a **switch table**, each entry:

- ✓ (MAC address of host, interface to reach host, time stamp)
- ✓ looks like a routing table!

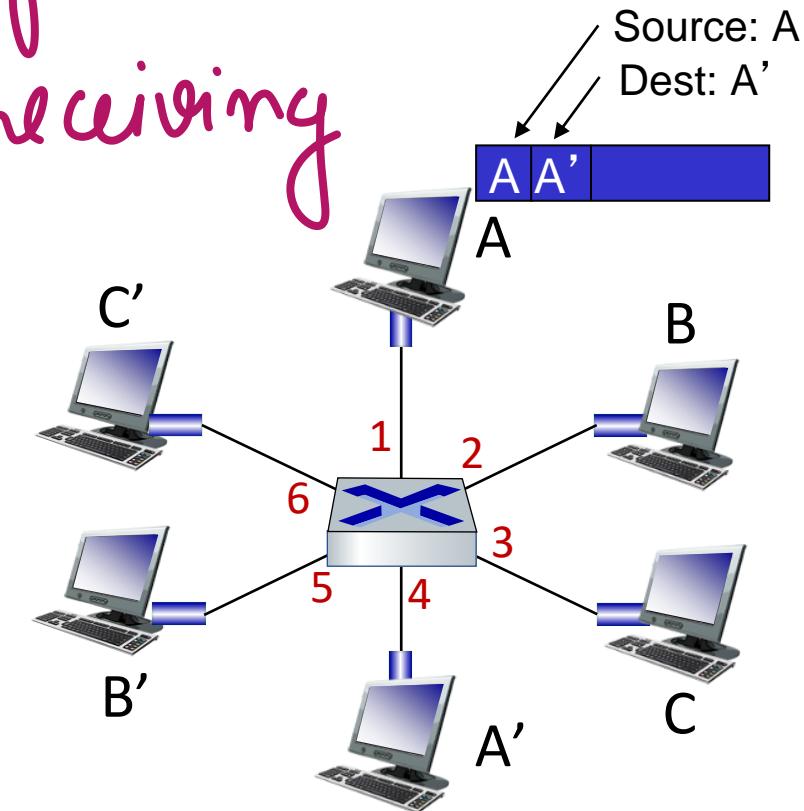
**Q:** how are entries created, maintained in switch table?

- something like a routing protocol?



# Switch: self-learning → only on receiving

- switch *learns* which hosts can be reached through which interfaces
- when frame received, switch “learns” location of sender: incoming LAN segment
- records sender/location pair in switch table



| MAC addr | interface | TTL |
|----------|-----------|-----|
| A        | 1         | 60  |

Switch table  
(initially empty)

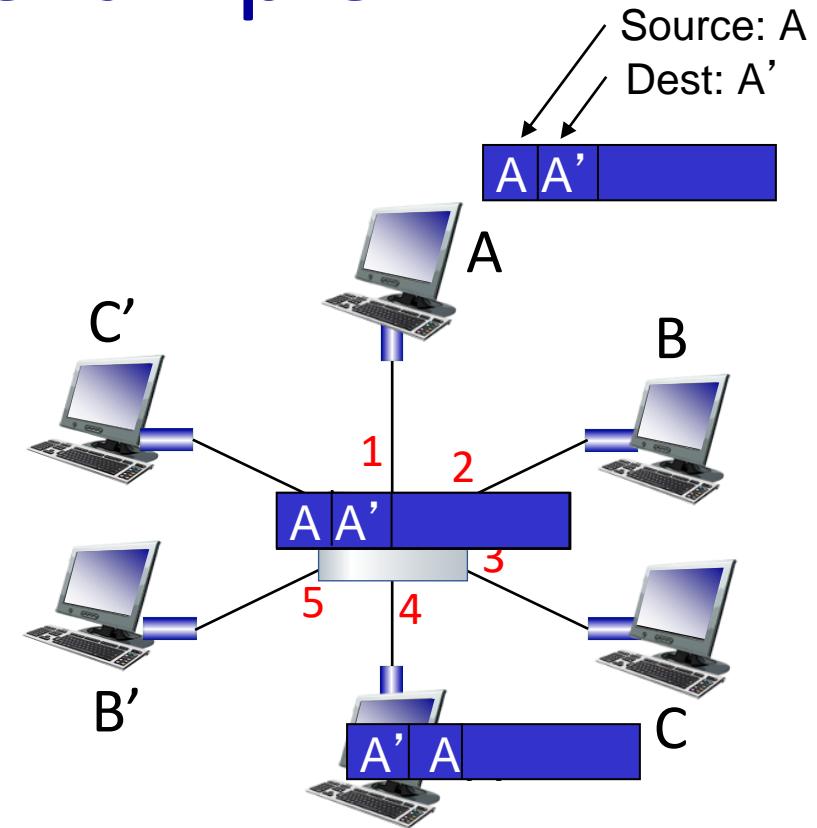
# Switch: frame filtering/forwarding

when frame received at switch:

- ✓ 1. record incoming link, MAC address of sending host
- ✓ 2. index switch table using MAC destination address
- ✓ 3. if entry found for destination **then**
  - {
    - if destination on segment from which frame arrived
      - ✓ then drop frame
      - ✓ else forward frame on interface indicated by entry
  - }
  - ✓ else flood /\* forward on all interfaces except arriving interface \*/

# Self-learning, forwarding: example

- frame destination, A', location unknown: **flood**
- destination A location known: **selectively send on just one link**

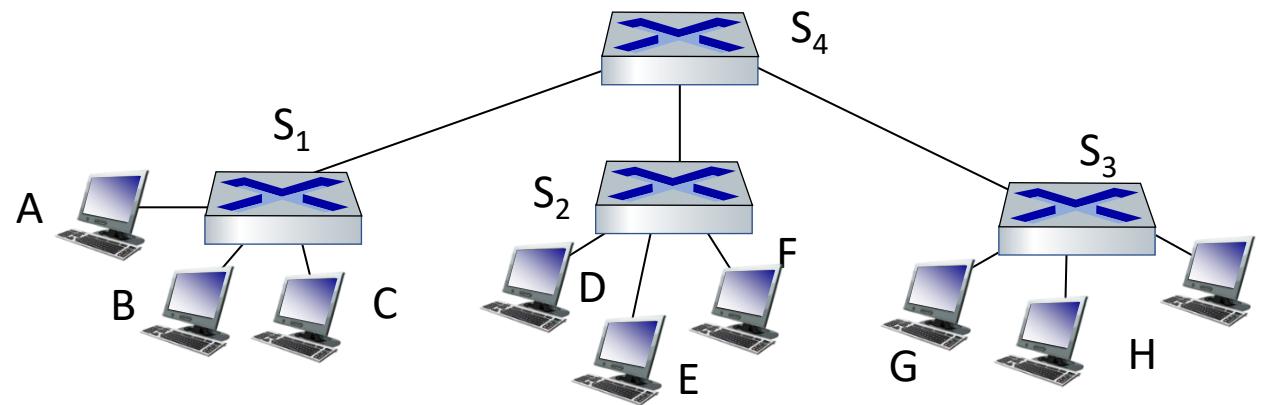


| MAC addr | interface | TTL |
|----------|-----------|-----|
| A        | 1         | 60  |
| A'       | 4         | 60  |

*switch table  
(initially empty)*

# Interconnecting switches

self-learning switches can be connected together:

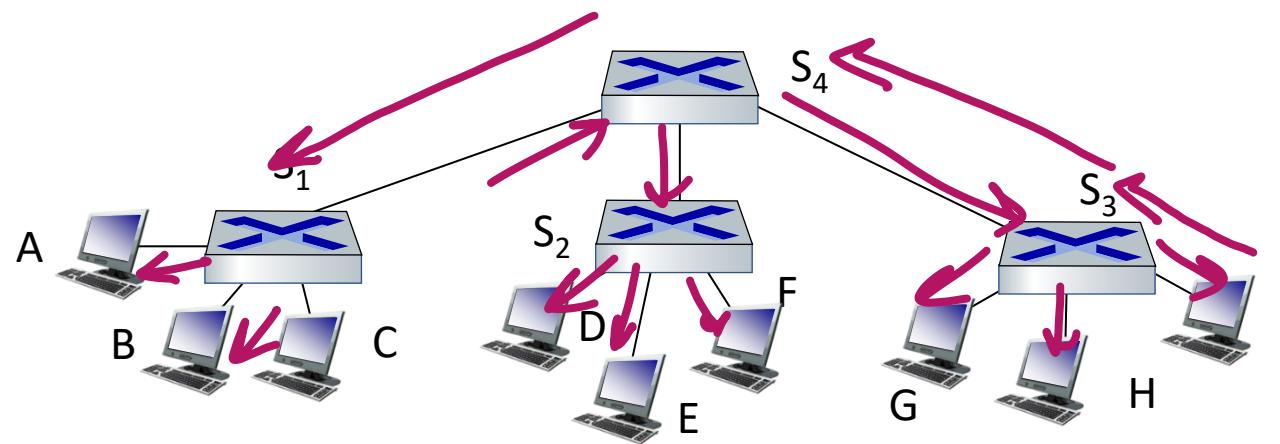


Q: sending from A to G - how does  $S_1$  know to forward frame destined to G via  $S_4$  and  $S_3$ ?

- A: self learning! (works exactly the same as in single-switch case!)

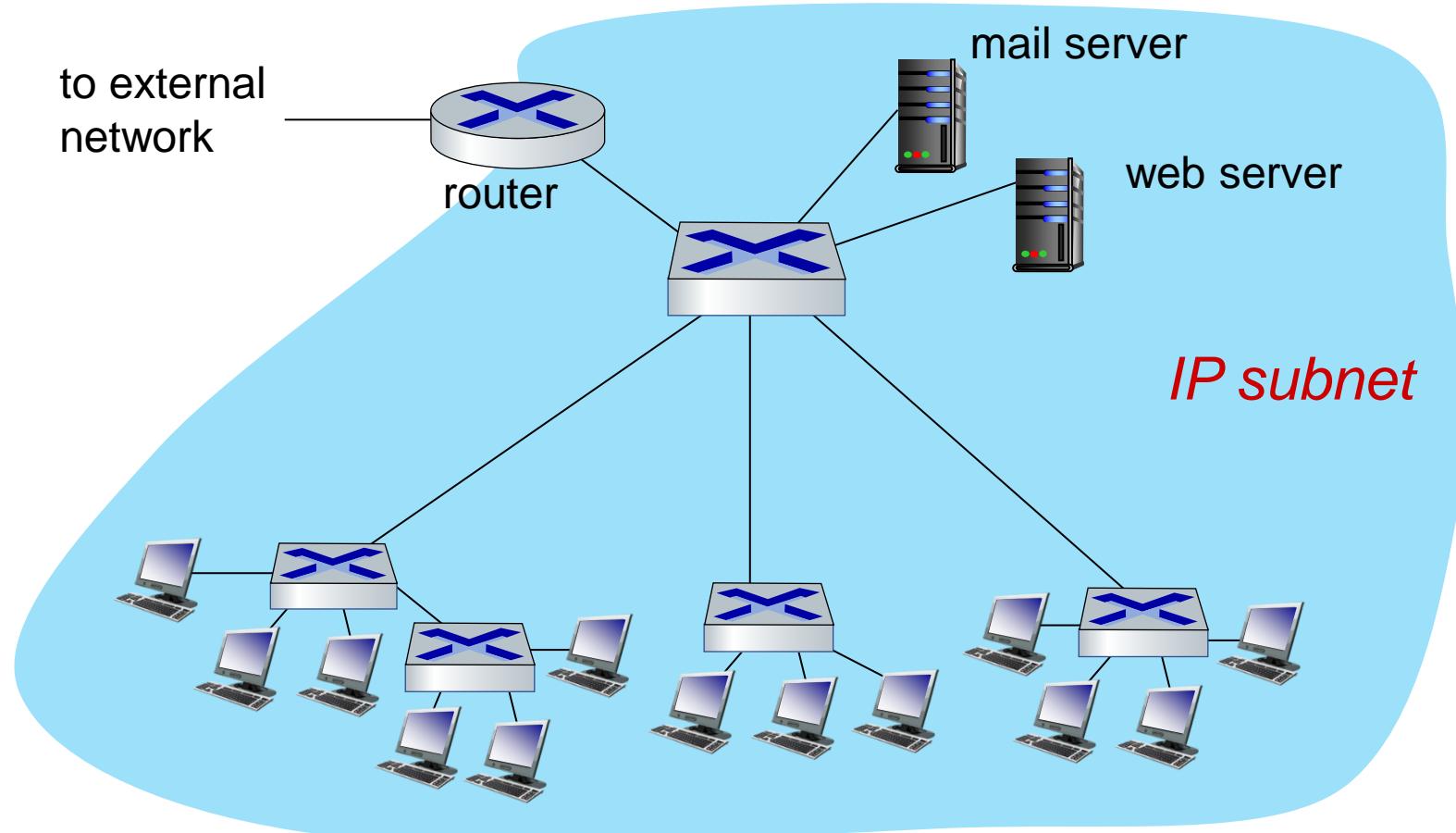
# Self-learning multi-switch example

Suppose C sends frame to I, I responds to C



Q: show switch tables and packet forwarding in  $S_1, S_2, S_3, S_4$

# Small institutional network



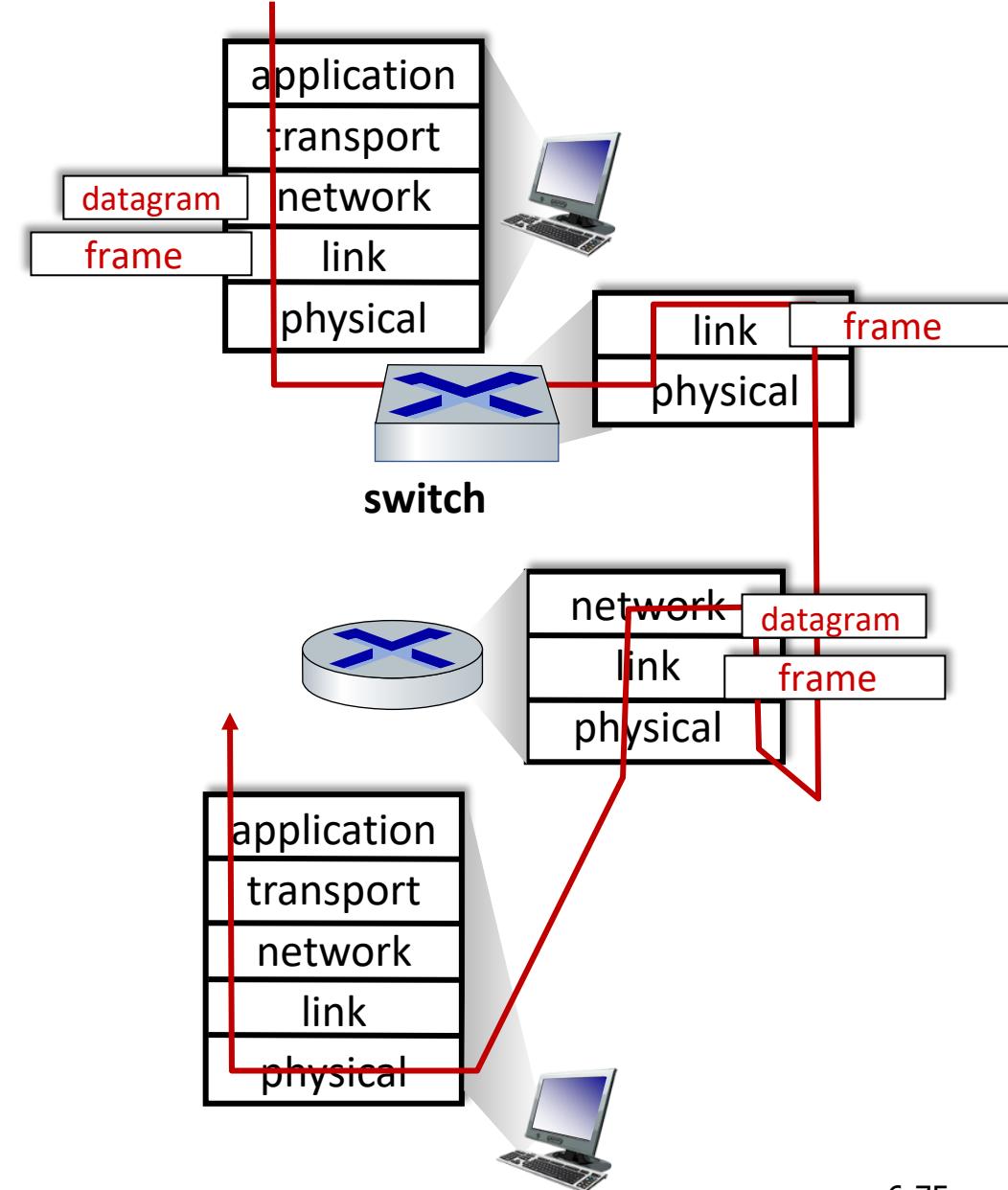
# Switches vs. routers

✓ both are store-and-forward:

- *routers*: network-layer devices (examine network-layer headers)
- *switches*: link-layer devices (examine link-layer headers)

both have forwarding tables:

- *routers*: compute tables using routing algorithms, IP addresses
- *switches*: Learn forwarding table using flooding, learning, MAC addresses



# Link layer, LANs: roadmap

- introduction
- error detection, correction
- multiple access protocols
- LANs
  - addressing, ARP
  - Ethernet
  - switches
- data center networking
- putting it all together

# Datacenter networks

10's to 100's of thousands of hosts, often closely coupled, in close proximity:

- e-business (e.g. Amazon)
- content-servers (e.g., YouTube, Akamai, Apple, Microsoft)
- search engines, data mining (e.g., Google)

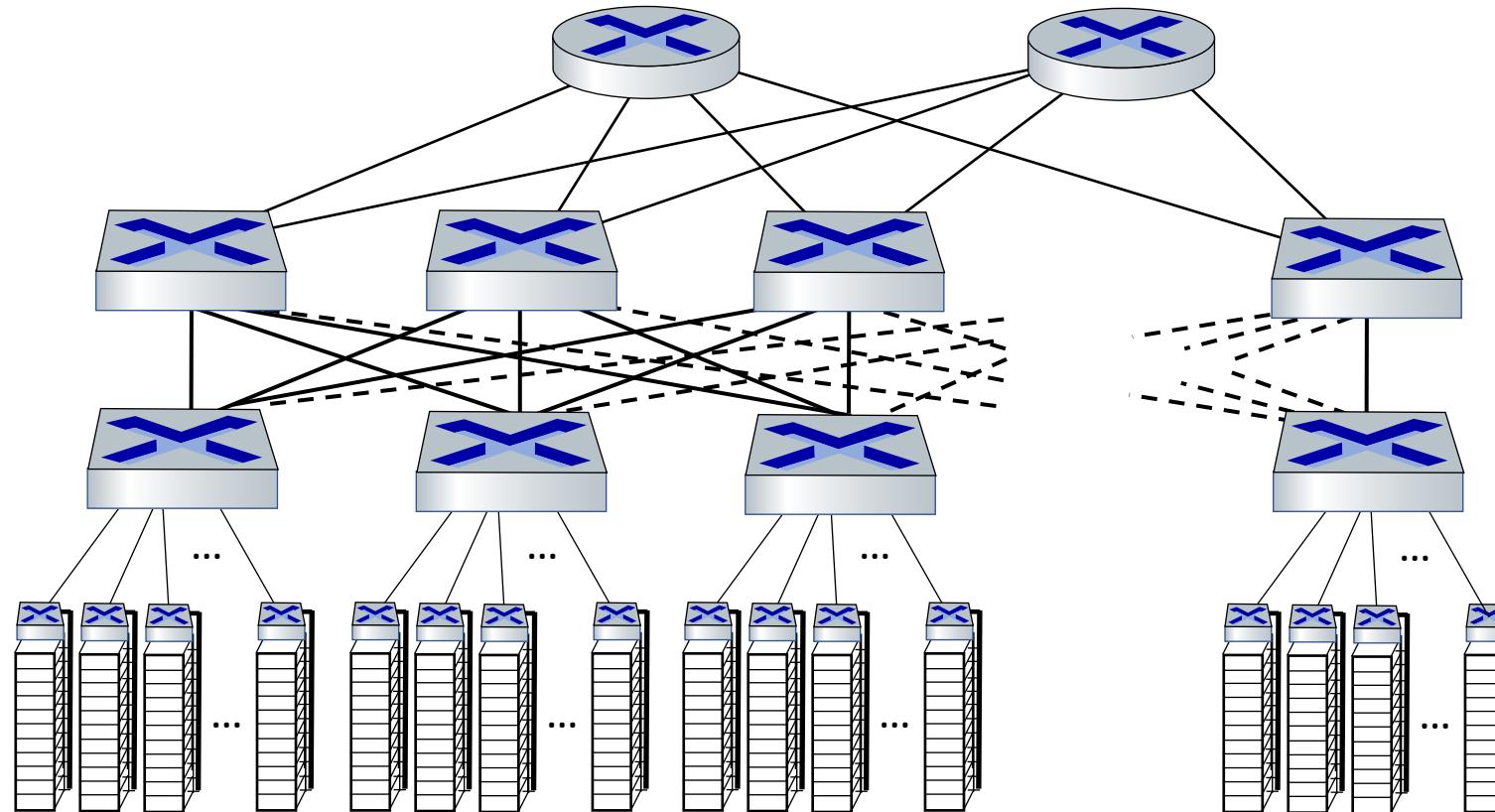
challenges:

- multiple applications, each serving massive numbers of clients
- reliability
- managing/balancing load, avoiding processing, networking, data bottlenecks



Inside a 40-ft Microsoft container, Chicago data center

# Datacenter networks: network elements



## Border routers

- connections outside datacenter

## Tier-1 switches

- connecting to ~16 T-2s below

## Tier-2 switches

- connecting to ~16 TORs below

## Top of Rack (TOR) switch

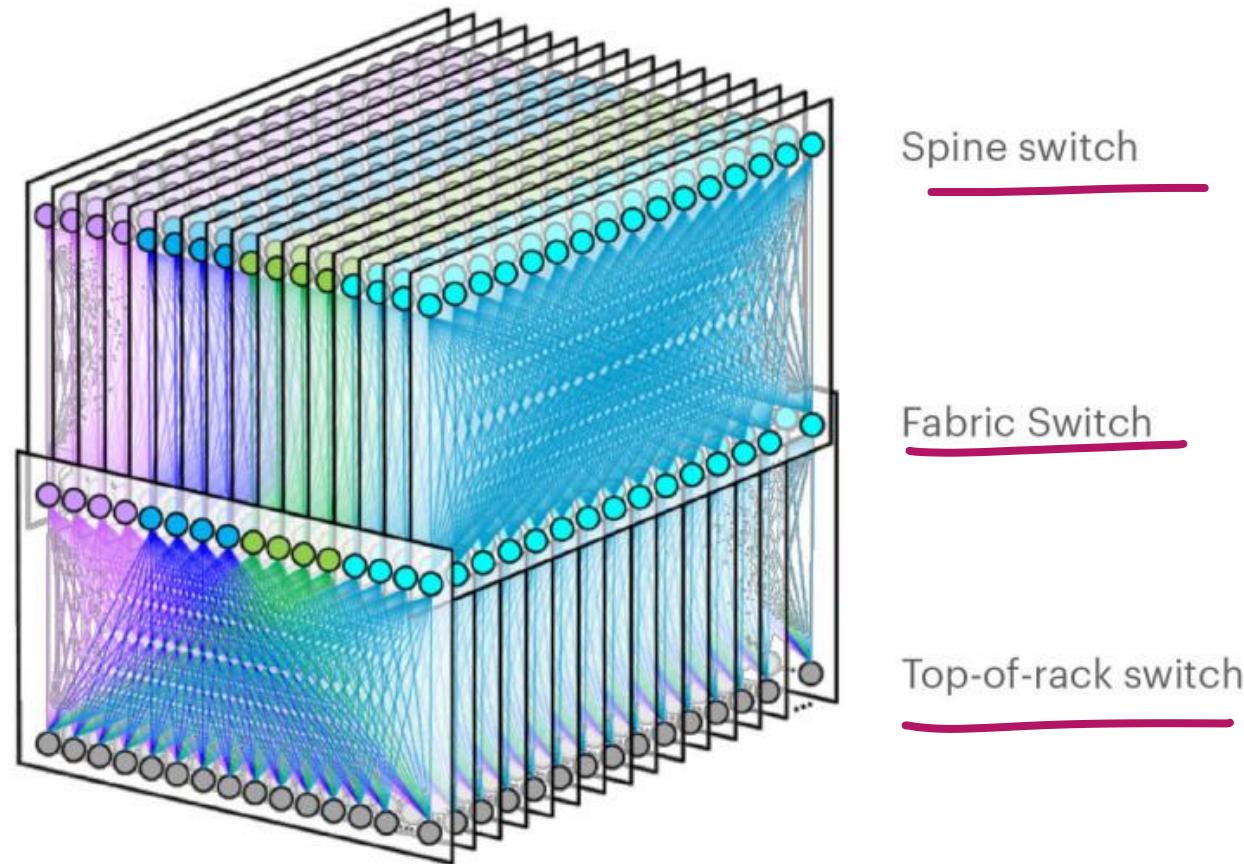
- one per rack
- 40-100Gbps Ethernet to blades

## Server racks

- 20- 40 server blades; hosts

# Datacenter networks: network elements

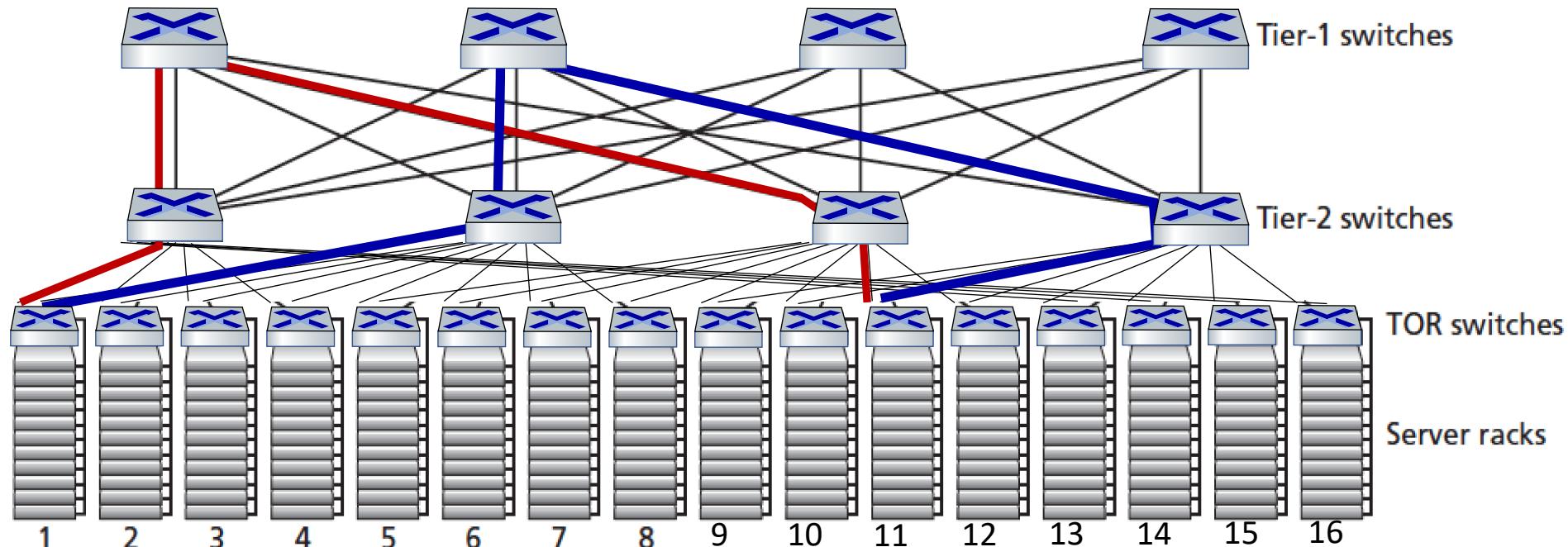
Facebook F16 data center network topology:



<https://engineering.fb.com/data-center-engineering/f16-minipack/> (posted 3/2019)

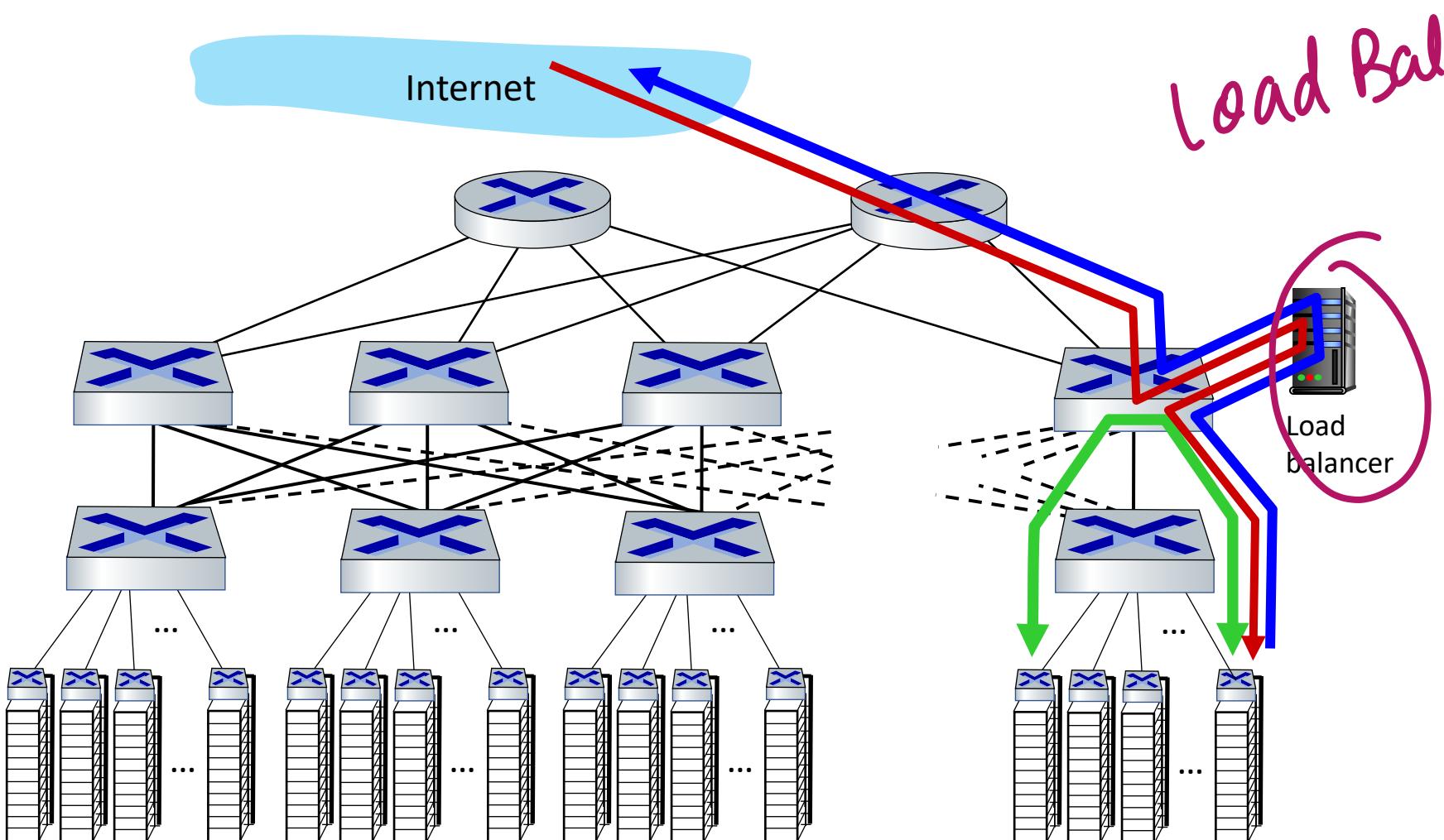
# Datacenter networks: multipath

- rich interconnection among switches, racks:
  - increased throughput between racks (multiple routing paths possible)
  - increased reliability via redundancy



two **disjoint** paths highlighted between racks 1 and 11

# Datacenter networks: application-layer routing



load balancer:  
application-layer  
routing

- receives external client requests
- directs workload within data center
- returns results to external client (hiding data center internals from client)

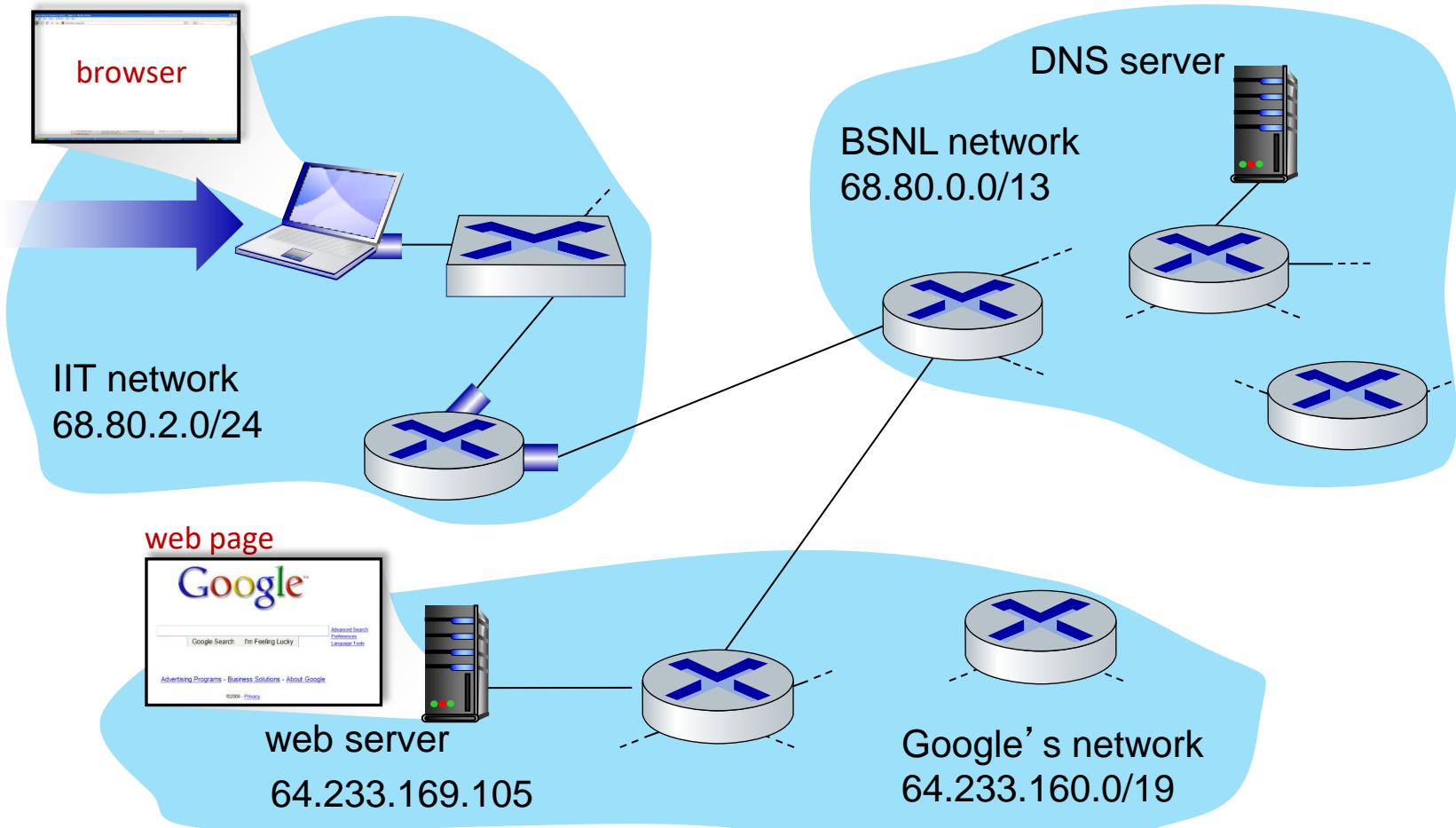
# Link layer, LANs: roadmap

- introduction
- error detection, correction
- multiple access protocols
- LANs
  - addressing, ARP
  - Ethernet
  - switches
- data center networking
- putting it all together

# Synthesis: a day in the life of a web request

- our journey down the protocol stack is now complete!
  - application, transport, network, link
- putting-it-all-together: synthesis!
  - *goal:* identify, review, understand protocols (at all layers) involved in seemingly simple scenario: requesting www page
  - *scenario:* student attaches laptop to campus network, requests/receives [www.google.com](http://www.google.com)

# A day in the life: scenario

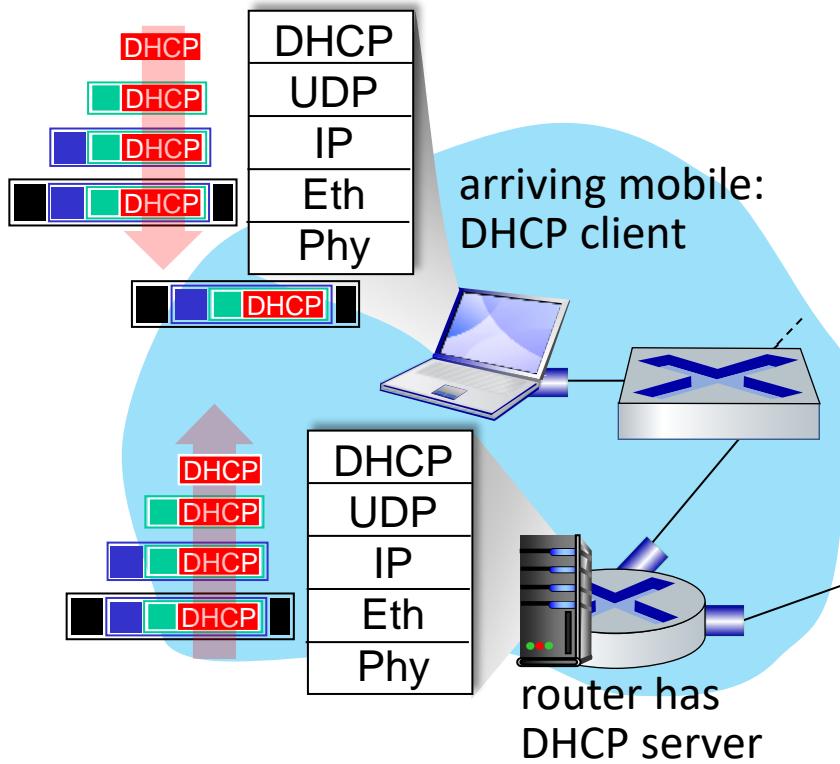


scenario:

- arriving mobile client attaches to network ...
- requests web page:  
[www.google.com](http://www.google.com)

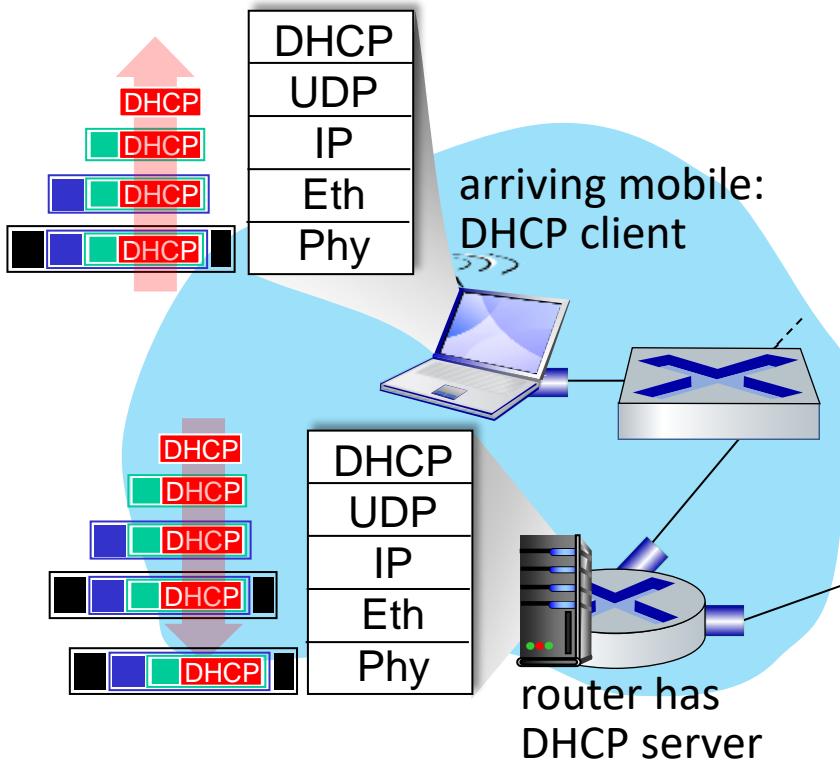
*Sounds simple!* !

# A day in the life: connecting to the Internet



- ✓ connecting laptop needs to get its own IP address, addr of first-hop router, addr of DNS server: use DHCP
- ✓ DHCP request encapsulated in UDP, encapsulated in IP, encapsulated in 802.3 Ethernet
- ✓ Ethernet frame broadcast (dest: FFFFFFFFFFFF) on LAN, received at router running DHCP server
- ✓ Ethernet demuxed to IP demuxed, UDP demuxed to DHCP

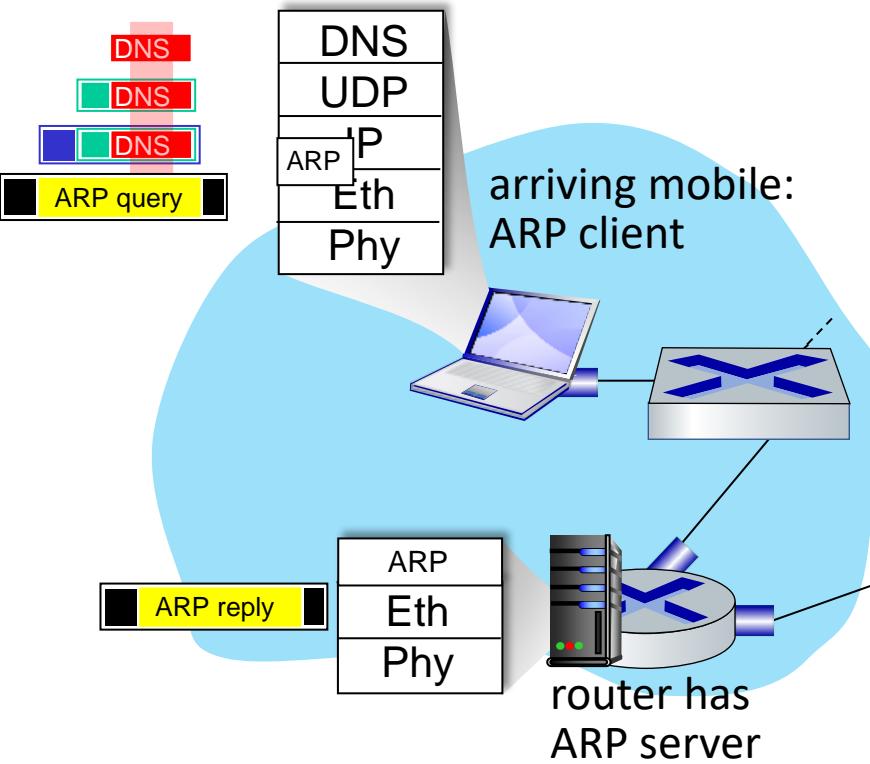
# A day in the life: connecting to the Internet



- DHCP server formulates **DHCP ACK** containing client's IP address, IP address of first-hop router for client, name & IP address of DNS server
- encapsulation at DHCP server, frame forwarded (**switch learning**) through LAN, demultiplexing at client
- DHCP client receives DHCP ACK reply

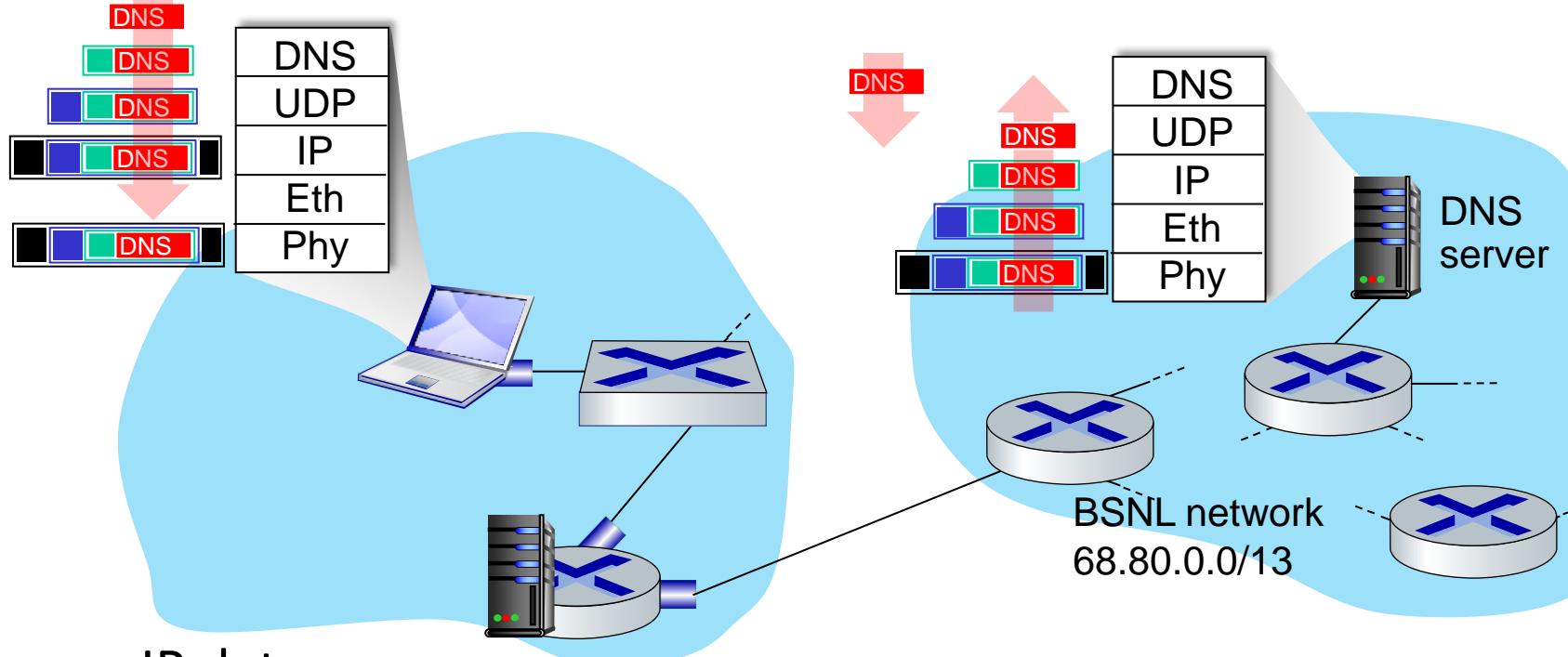
*Client now has IP address, knows name & addr of DNS server, IP address of its first-hop router*

# A day in the life... ARP (before DNS, before HTTP)



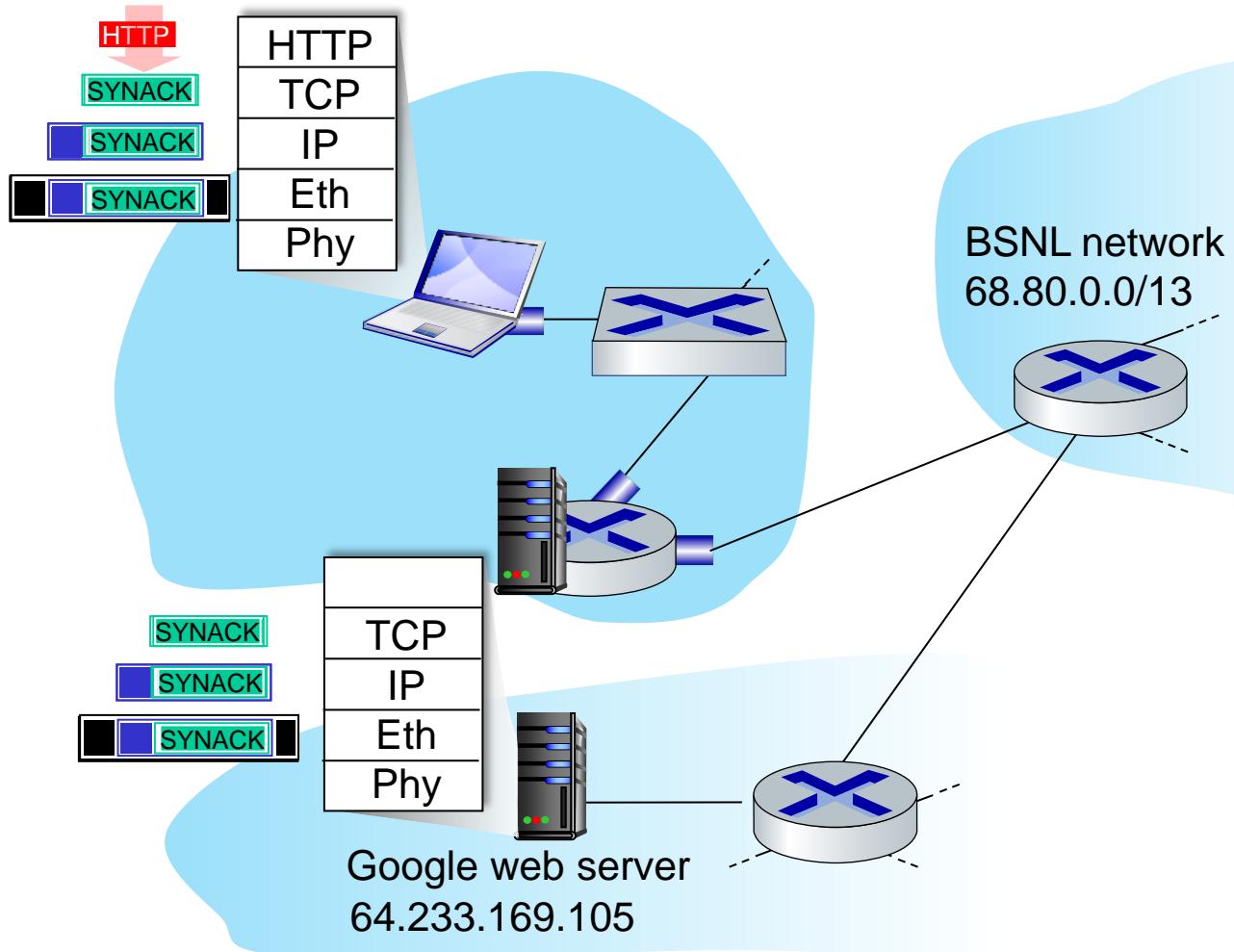
- before sending **HTTP** request, need IP address of www.google.com: **DNS**
  - DNS query created, encapsulated in UDP, encapsulated in IP, encapsulated in Eth. To send frame to router, need MAC address of router interface: ARP
  - **ARP query broadcast**, received by router, which replies with **ARP reply** giving MAC address of router interface
- client now knows MAC address of first hop router, so can now send frame containing DNS query

# A day in the life... using DNS



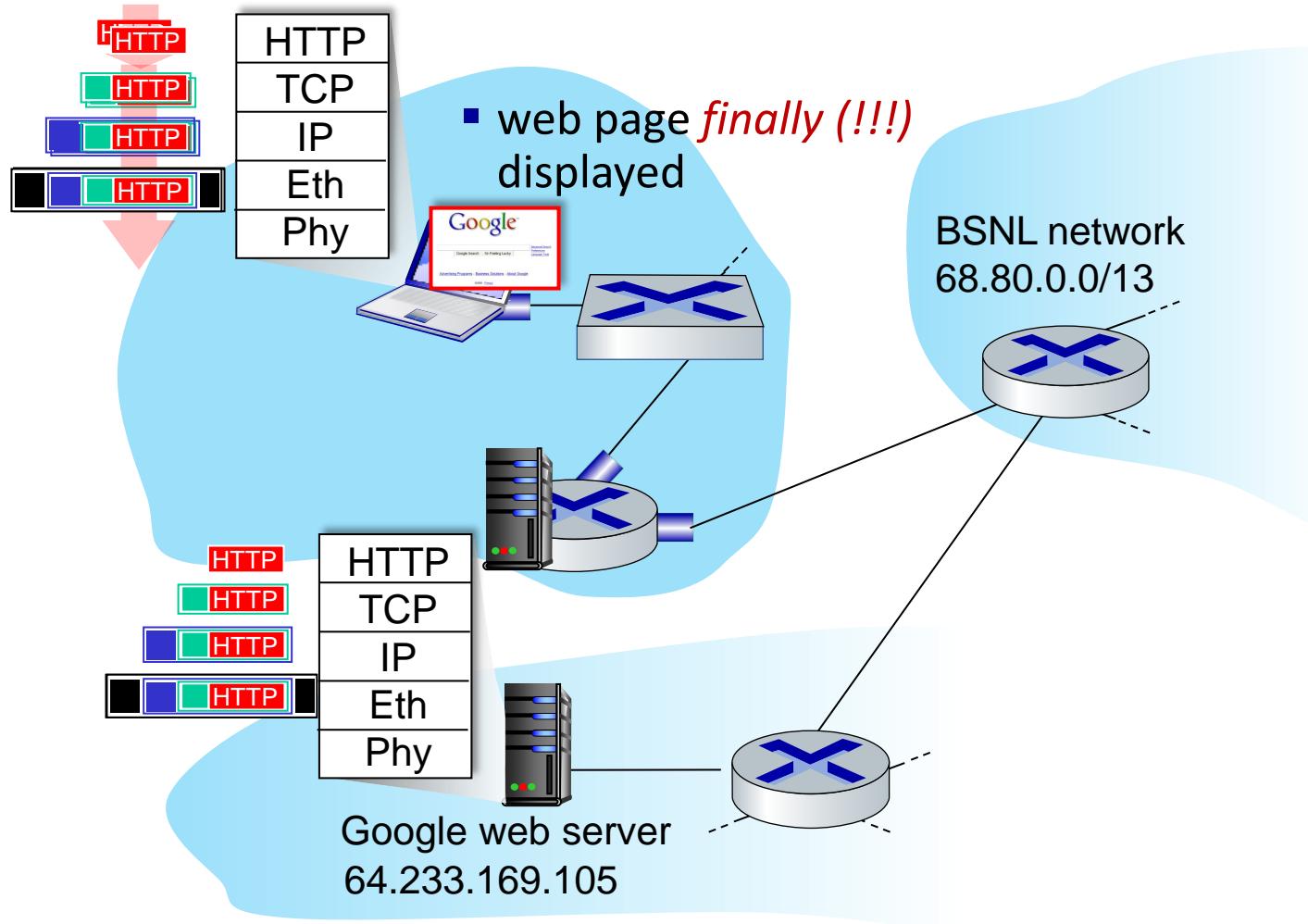
- IP datagram containing DNS query forwarded via LAN switch from client to 1<sup>st</sup> hop router
- IP datagram forwarded from campus network into BSNL network, routed (tables created by **RIP, OSPF, IS-IS** and/or **BGP** routing protocols) to DNS server
- demuxed to DNS
- DNS replies to client with IP address of [www.google.com](http://www.google.com)

# A day in the life...TCP connection carrying HTTP



- to send HTTP request, client first opens **TCP socket** to web server
- **TCP SYN segment** (step 1 in TCP 3-way handshake) inter-domain routed to web server
- web server responds with **TCP SYNACK** (step 2 in TCP 3-way handshake)
- **TCP connection established!**

# A day in the life... HTTP request/reply



- **HTTP request** sent into TCP socket
- IP datagram containing HTTP request routed to [www.google.com](http://www.google.com)
- web server responds with **HTTP reply** (containing web page)
- IP datagram containing HTTP reply routed back to client

# Wireless and Mobile Networks

Slides are adapted from the companion website of the book *Computer Networking: A Top-Down Approach* by Jim Kurose, Keith Ross

# Wireless and Mobile Networks: context

- more wireless (mobile) phone subscribers than fixed (wired) phone subscribers (10-to-1 in 2019)!
- more mobile-broadband-connected devices than fixed-broadband-connected devices (5-1 in 2019)!
  - 4G/5G cellular networks now embracing Internet protocol stack, including SDN
- two important (but different) challenges
  - **wireless**: communication over wireless link
  - **mobility**: handling the mobile user who changes point of attachment to network

# Outline

- **Introduction**

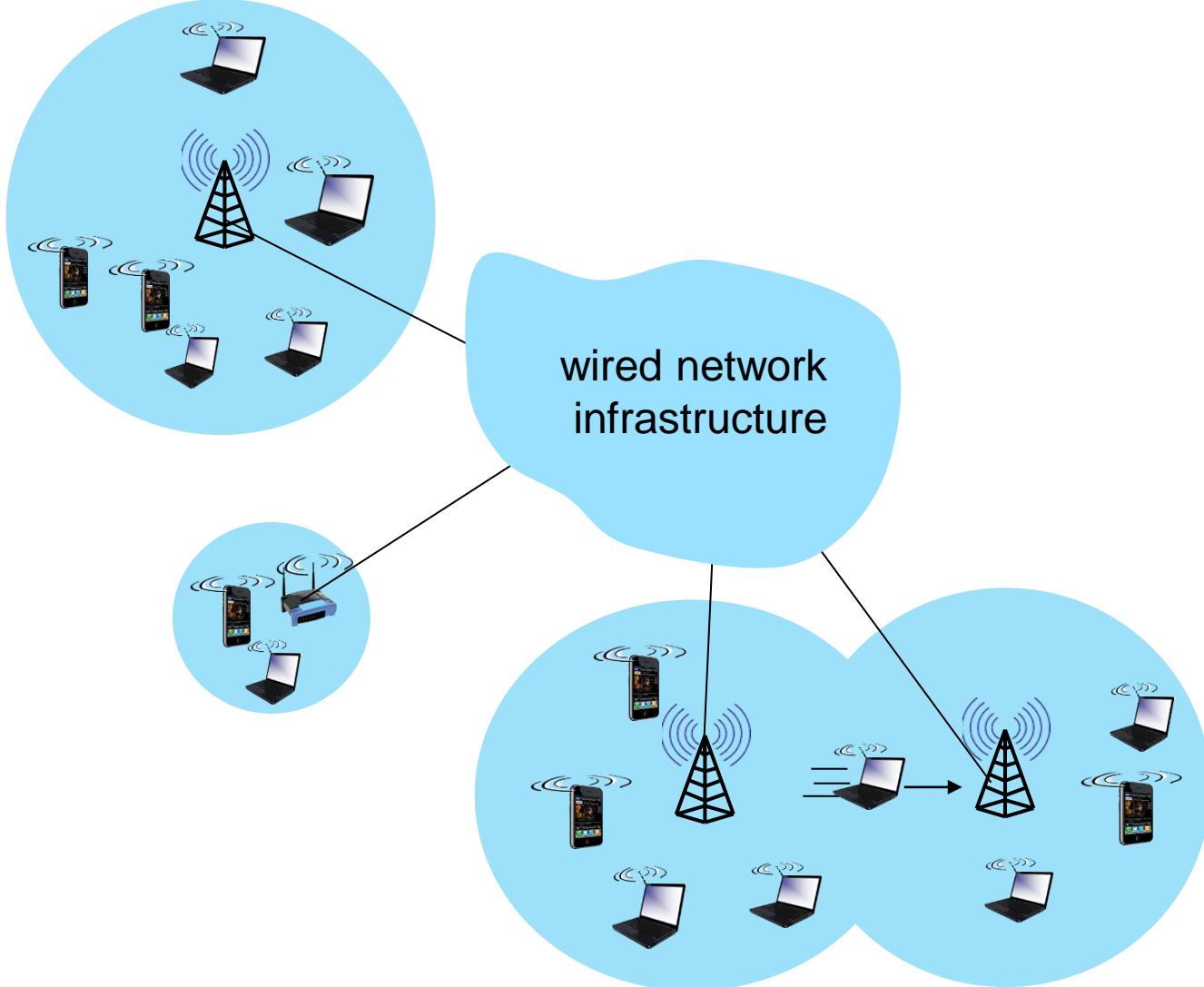
## Wireless

- Wireless Links and network characteristics
- WiFi: 802.11 wireless LANs
- Cellular networks: 4G and 5G

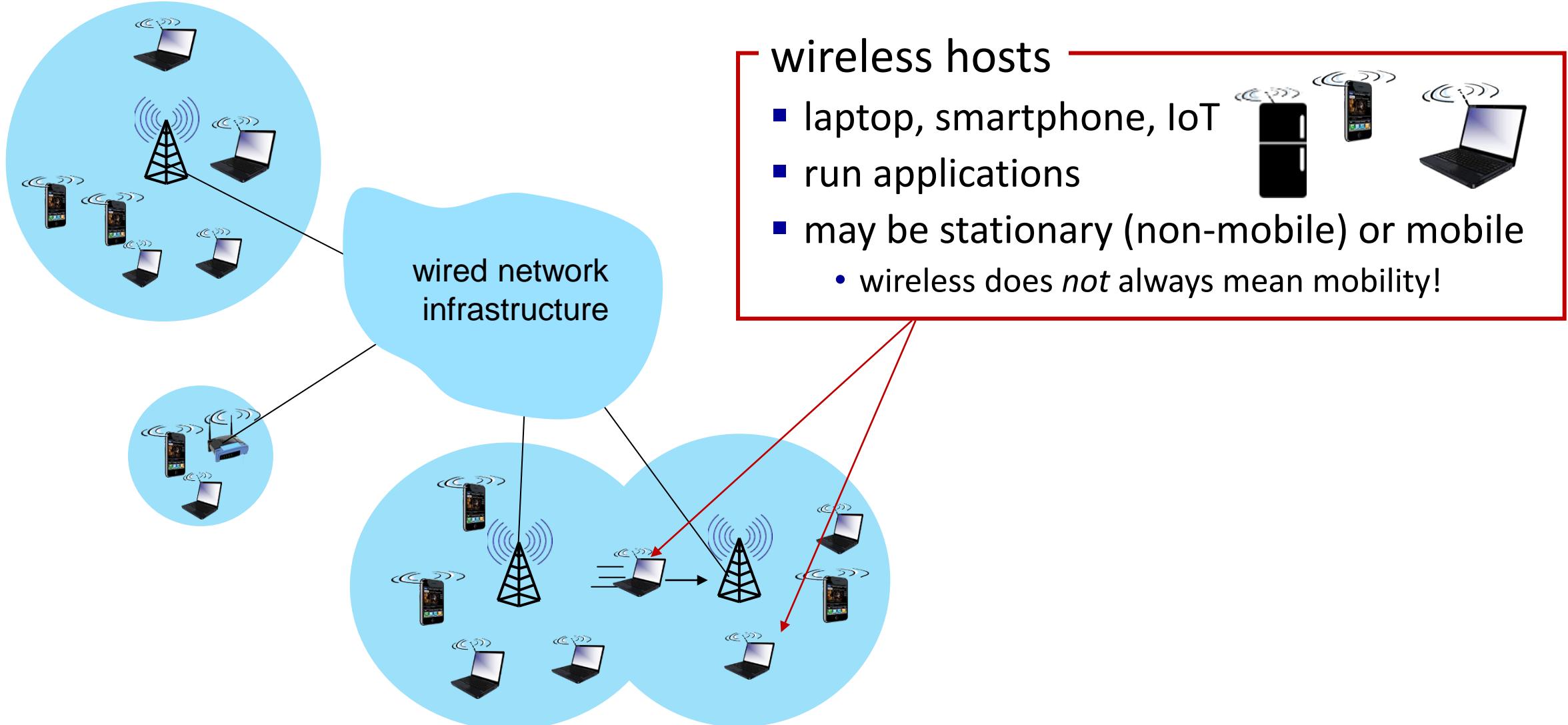
## Mobility

- Mobility management: principles
- Mobility management: practice
  - 4G/5G networks
  - Mobile IP
- Mobility: impact on higher-layer protocols

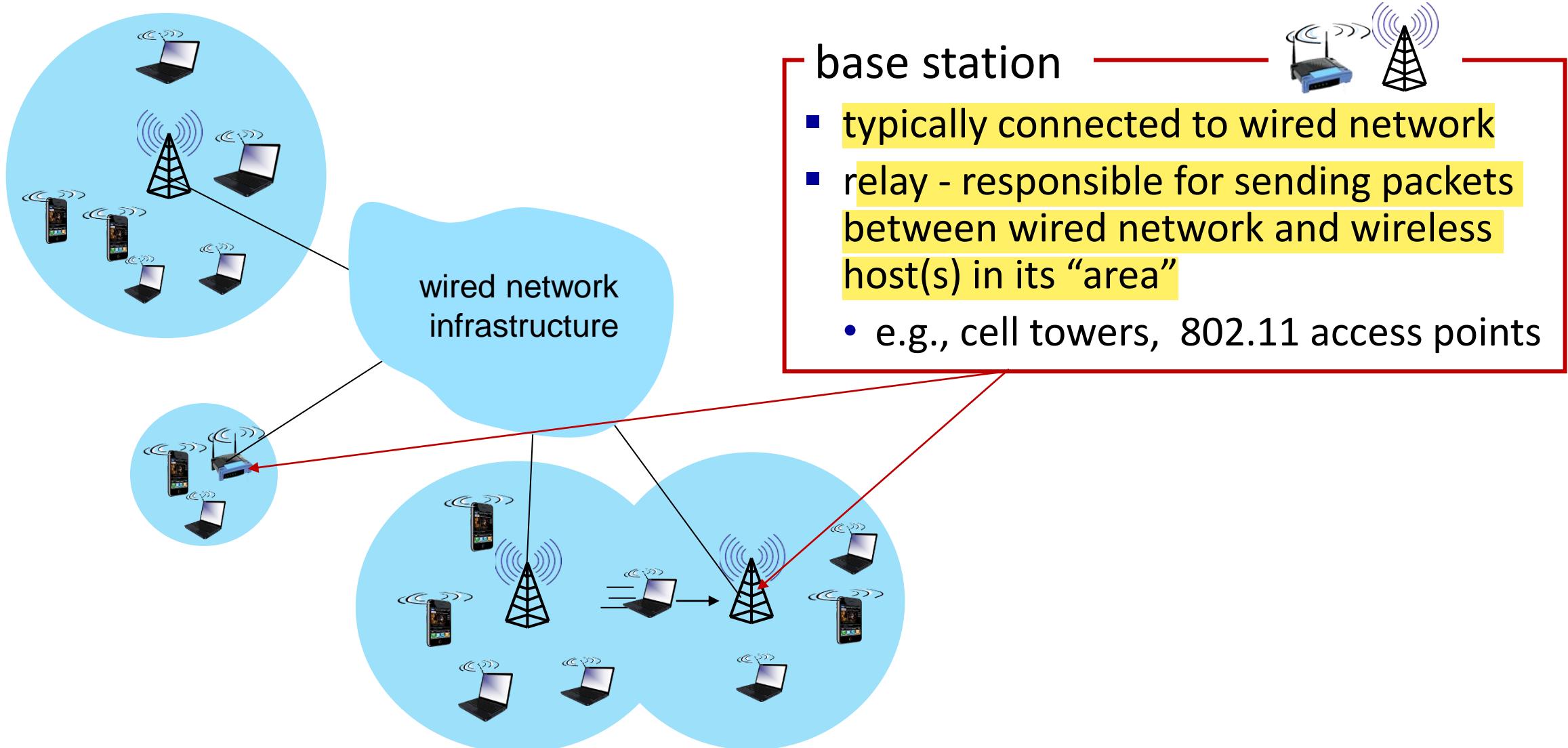
# Elements of a wireless network



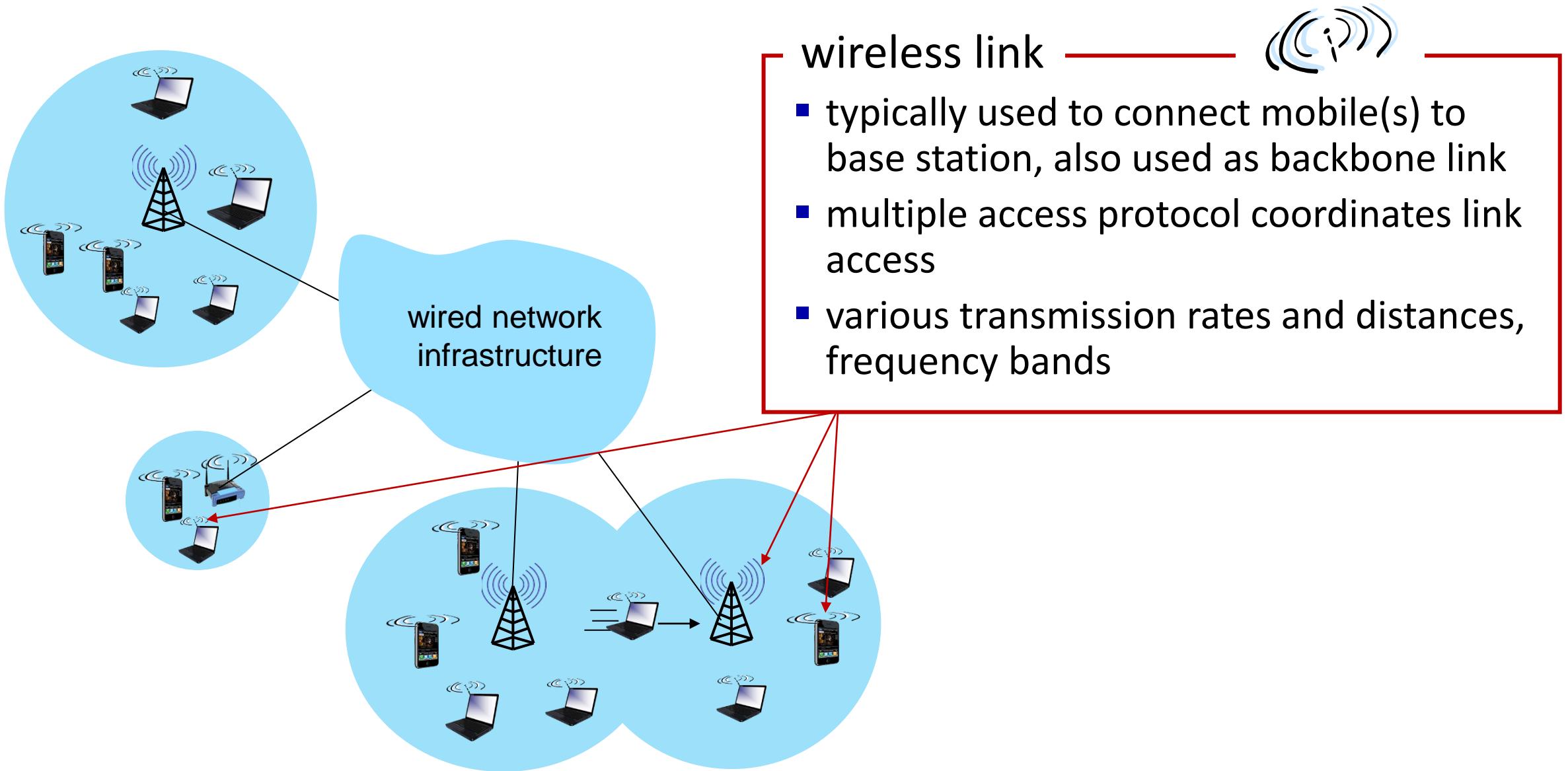
# Elements of a wireless network



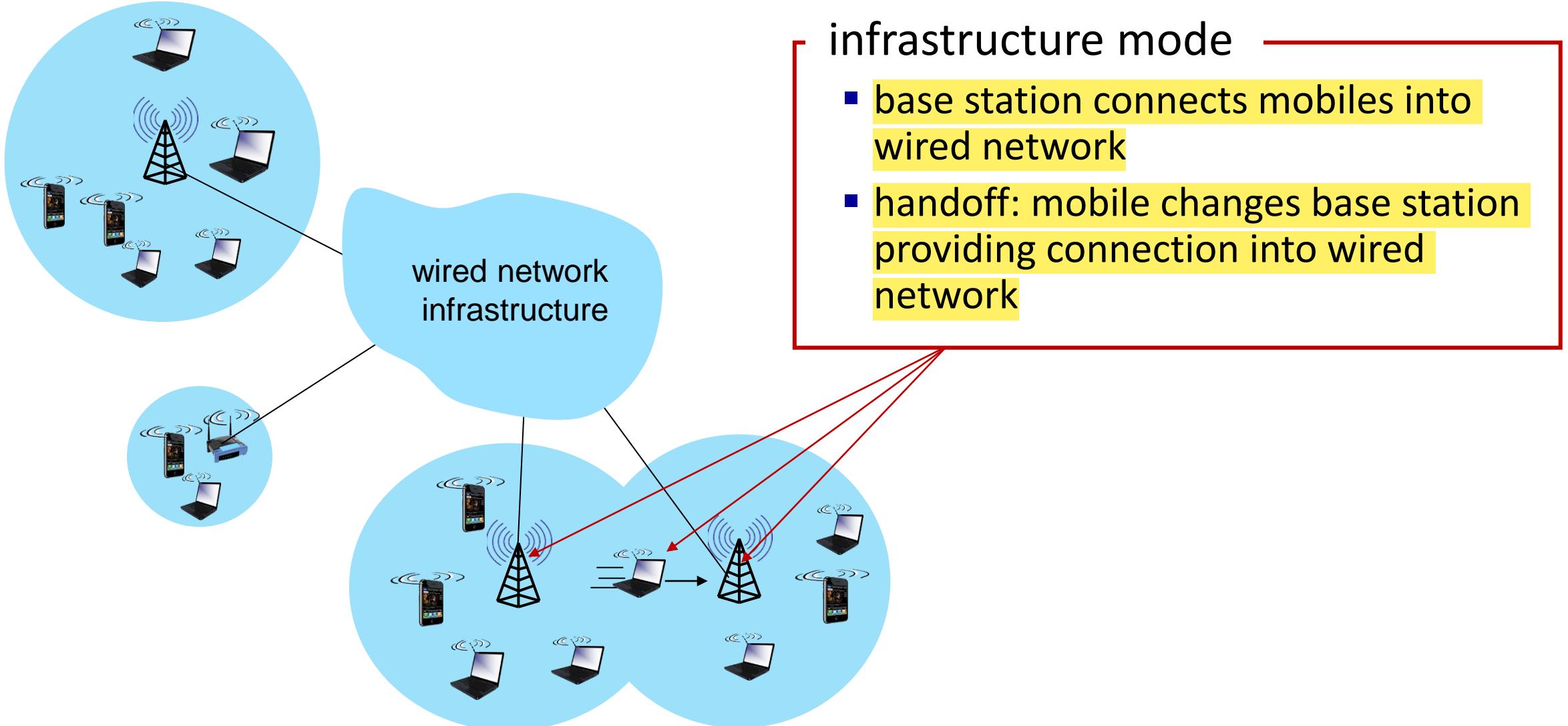
# Elements of a wireless network



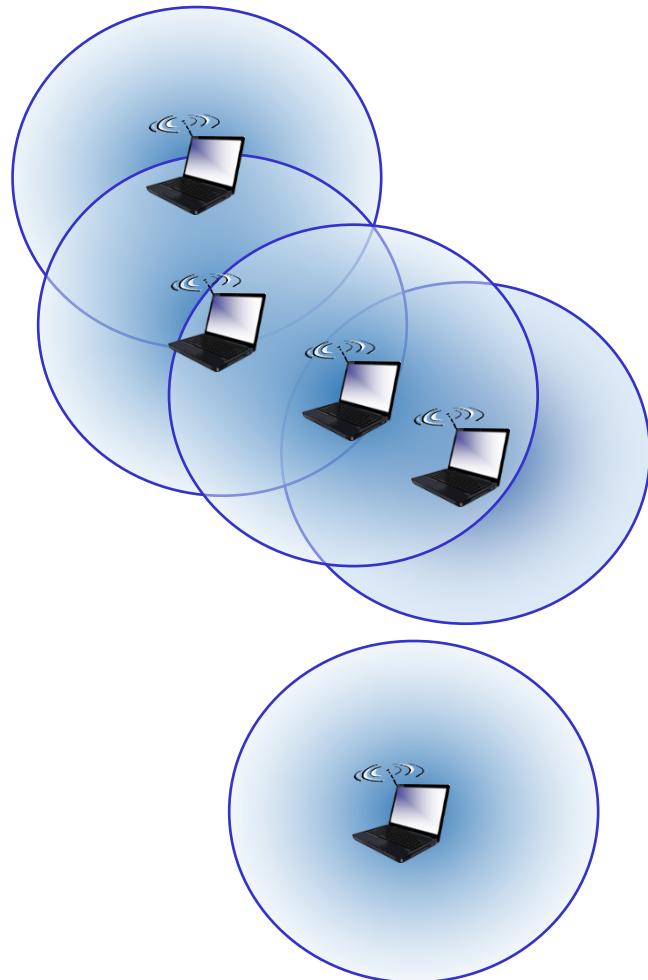
# Elements of a wireless network



# Elements of a wireless network



# Elements of a wireless network



ad hoc mode

- no base stations
- nodes can only transmit to other nodes within link coverage
- nodes organize themselves into a network: route among themselves

# Wireless network taxonomy

|                               | single hop                                                                       | multiple hops                                                                                                          |
|-------------------------------|----------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------|
| infrastructure<br>(e.g., APs) | host connects to base station (WiFi, cellular) which connects to larger Internet | host may have to relay through several wireless nodes to connect to larger Internet: <i>mesh net</i>                   |
| <i>no infrastructure</i>      | no base station, no connection to larger Internet (Bluetooth, ad hoc nets)       | no base station, no connection to larger Internet. May have to relay to reach other a given wireless node MANET, VANET |

# Outline

- Introduction

## Wireless

- Wireless links and network characteristics
- WiFi: 802.11 wireless LANs
- Cellular networks: 4G and 5G

## Mobility

- Mobility management: principles
- Mobility management: practice
  - 4G/5G networks
  - Mobile IP
- Mobility: impact on higher-layer protocols

# Wireless link characteristics (1)

*important* differences from wired link ....

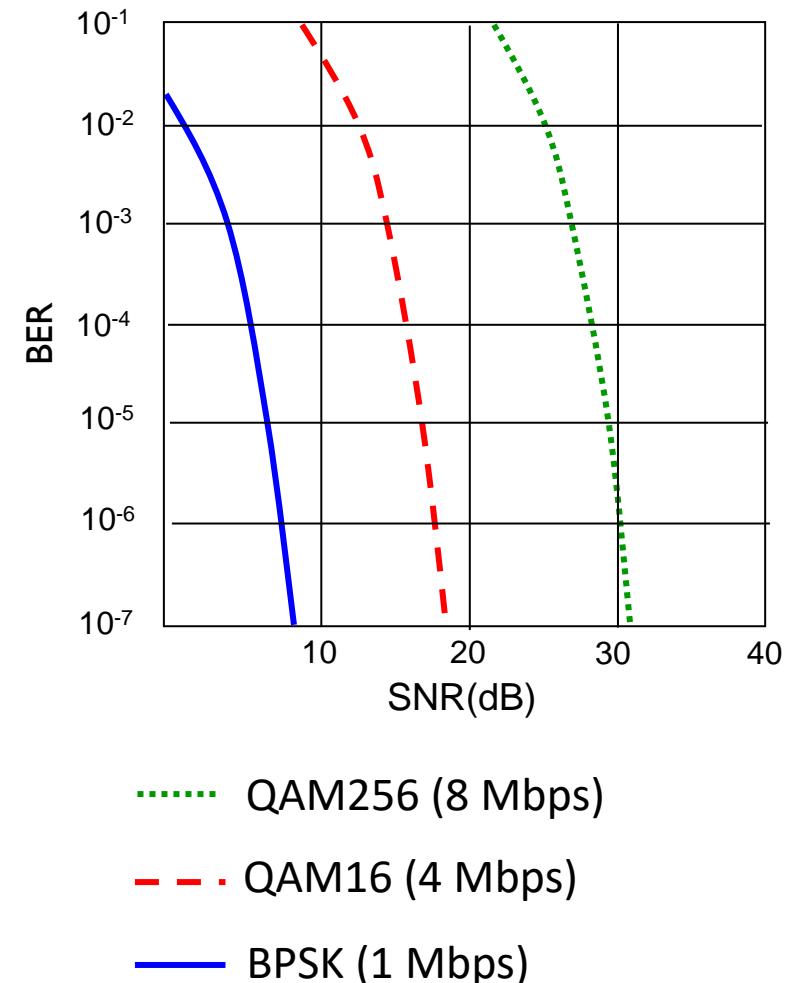
- **decreased signal strength:** radio signal attenuates as it propagates through matter (path loss)
- **interference from other sources:** wireless network frequencies (e.g., 2.4 GHz) shared by many devices (e.g., WiFi, cellular, motors): interference
- **multipath propagation:** radio signal reflects off objects ground, arriving at destination at slightly different times

.... make communication across (even a point to point) wireless link much more “difficult”



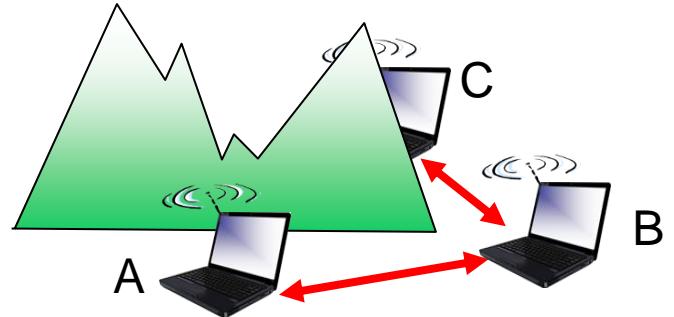
# Wireless link characteristics (2)

- SNR: signal-to-noise ratio
  - larger SNR – easier to extract signal from noise (a “good thing”)
- SNR versus BER tradeoffs
  - *given physical layer*: increase power -> increase SNR->decrease Bit Error Rate
  - *given SNR*: choose physical layer that meets BER requirement, giving highest throughput
    - SNR may change with mobility: dynamically adapt physical layer (modulation technique, rate)



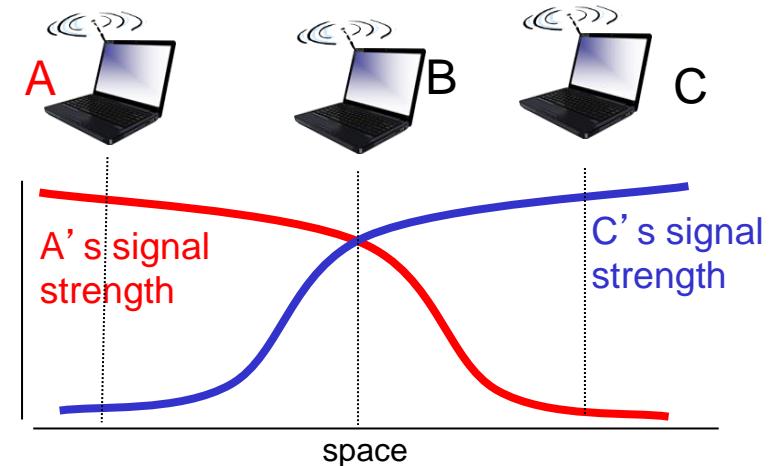
# Wireless link characteristics (3)

Multiple wireless senders, receivers create additional problems (beyond multiple access):



## Hidden terminal problem

- B, A hear each other
- B, C hear each other
- A, C can not hear each other means A, C unaware of their interference at B



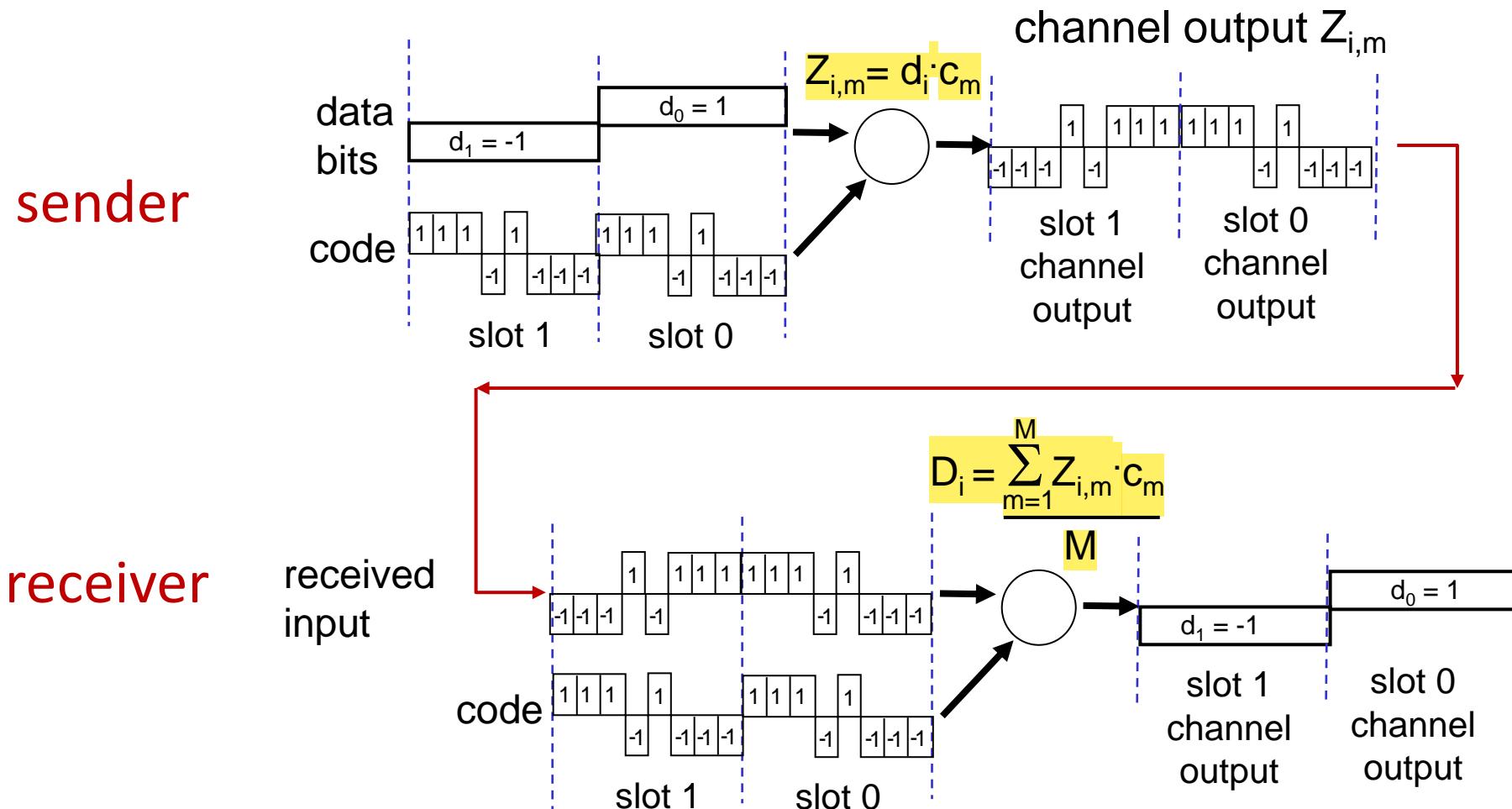
## Signal attenuation:

- B, A hear each other
- B, C hear each other
- A, C can not hear each other interfering at B

# Code Division Multiple Access (CDMA)

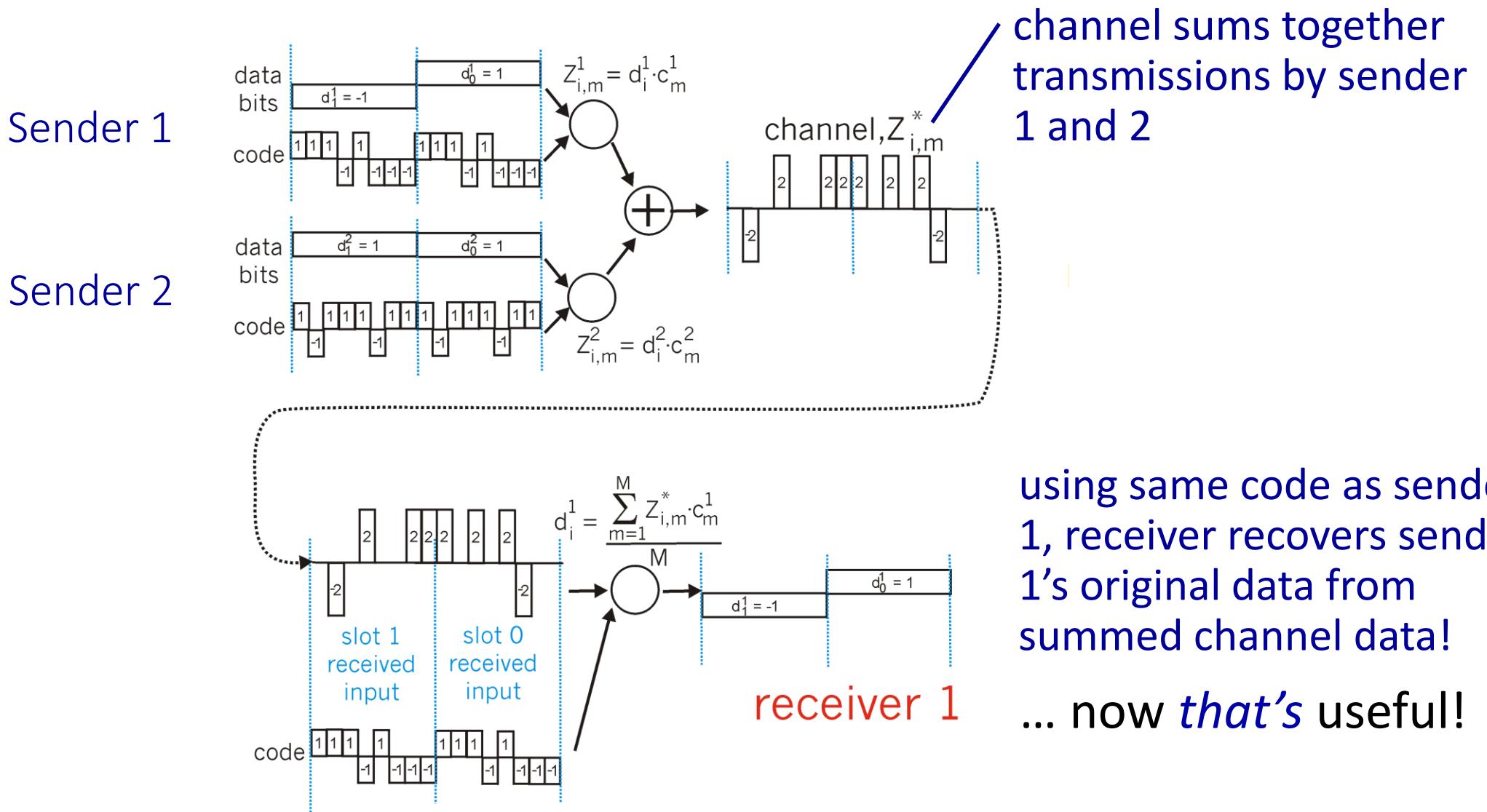
- unique “code” assigned to each user; i.e., code set partitioning
  - all users share same frequency, but each user has own “chipping” sequence (i.e., code) to encode data
  - allows multiple users to “coexist” and transmit simultaneously with minimal interference (if codes are “orthogonal”)
- **encoding:** inner product: (original data)  $\times$  (chipping sequence)
- **decoding:** summed inner-product: (encoded data)  $\times$  (chipping sequence)

# CDMA encode/decode



... but this isn't really useful yet!

# CDMA: two-sender interference



# Outline

- Introduction

## Wireless

- Wireless links and network characteristics
- WiFi: 802.11 wireless LANs
- Cellular networks: 4G and 5G

## Mobility

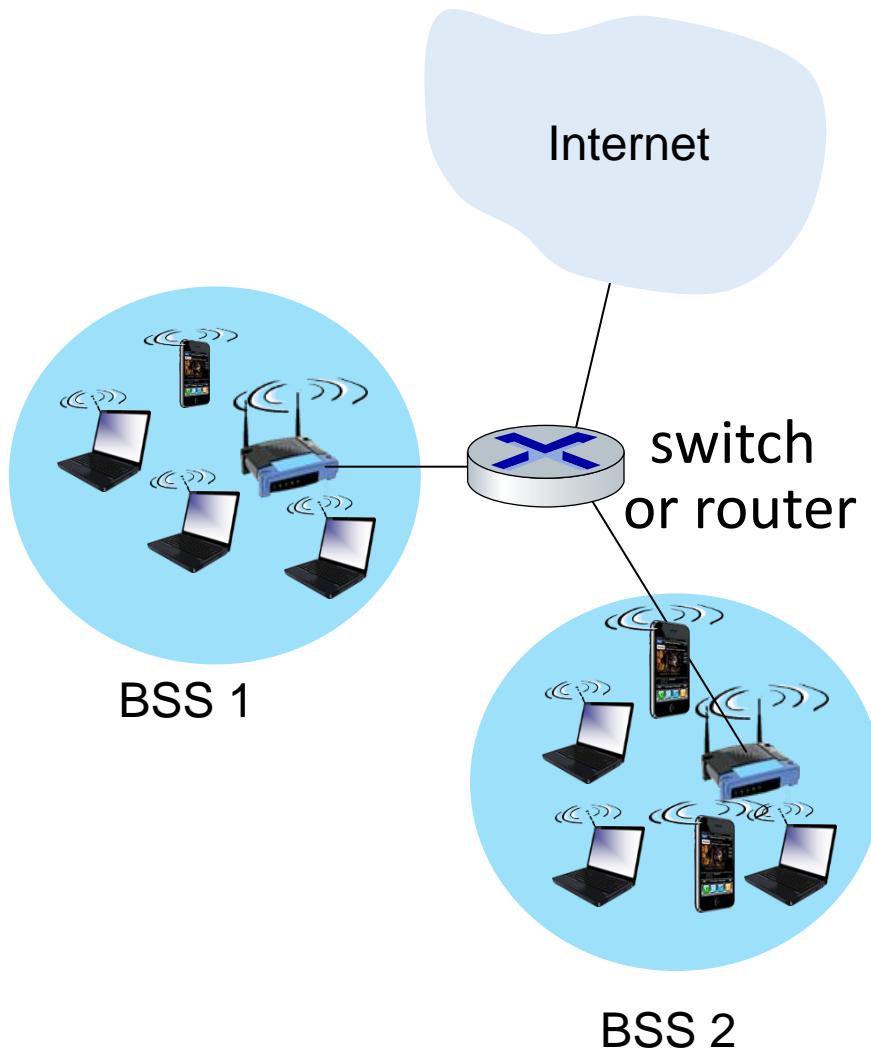
- Mobility management: principles
- Mobility management: practice
  - 4G/5G networks
  - Mobile IP
- Mobility: impact on higher-layer protocols

# IEEE 802.11 Wireless LAN

| IEEE 802.11 standard | Year        | Max data rate | Range | Frequency                    |
|----------------------|-------------|---------------|-------|------------------------------|
| 802.11b              | 1999        | 11 Mbps       | 30 m  | 2.4 Ghz                      |
| 802.11g              | 2003        | 54 Mbps       | 30m   | 2.4 Ghz                      |
| 802.11n (WiFi 4)     | 2009        | 600           | 70m   | 2.4, 5 Ghz                   |
| 802.11ac (WiFi 5)    | 2013        | 3.47Gpbs      | 70m   | 5 Ghz                        |
| 802.11ax (WiFi 6)    | 2020 (exp.) | 14 Gbps       | 70m   | 2.4, 5 Ghz                   |
| 802.11af             | 2014        | 35 – 560 Mbps | 1 Km  | unused TV bands (54-790 MHz) |
| 802.11ah             | 2017        | 347Mbps       | 1 Km  | 900 Mhz                      |

- all use CSMA/CA for multiple access, and have base-station and ad-hoc network versions

# 802.11 LAN architecture



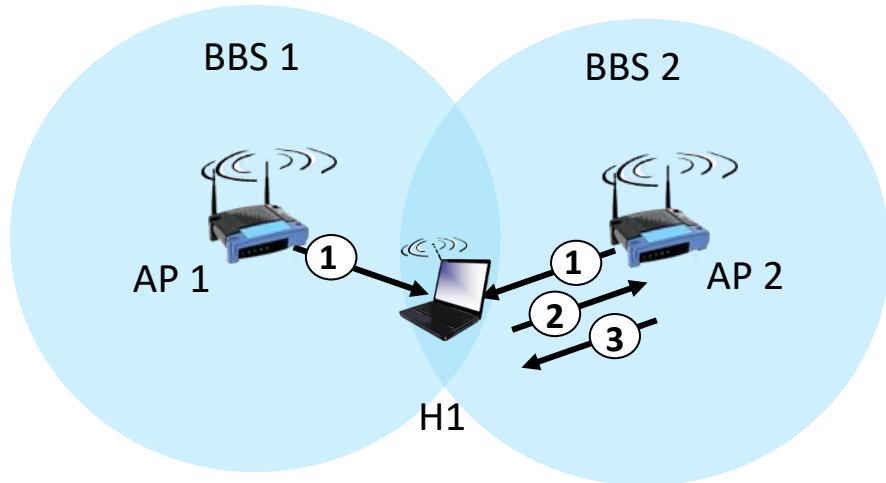
- wireless host communicates with base station
  - **base station = access point (AP)**
- **Basic Service Set (BSS) (aka “cell”)** in infrastructure mode contains:
  - wireless hosts
  - access point (AP): base station
  - ad hoc mode: hosts only

# 802.11: Channels, association

- spectrum divided into channels at different frequencies
  - AP admin chooses frequency for AP
  - interference possible: channel can be same as that chosen by neighboring AP!
- arriving host: must **associate** with an AP
  - scans channels, listening for *beacon frames* containing AP's name (SSID) and MAC address
  - selects AP to associate with
  - then may perform authentication
  - then typically run DHCP to get IP address in AP's subnet

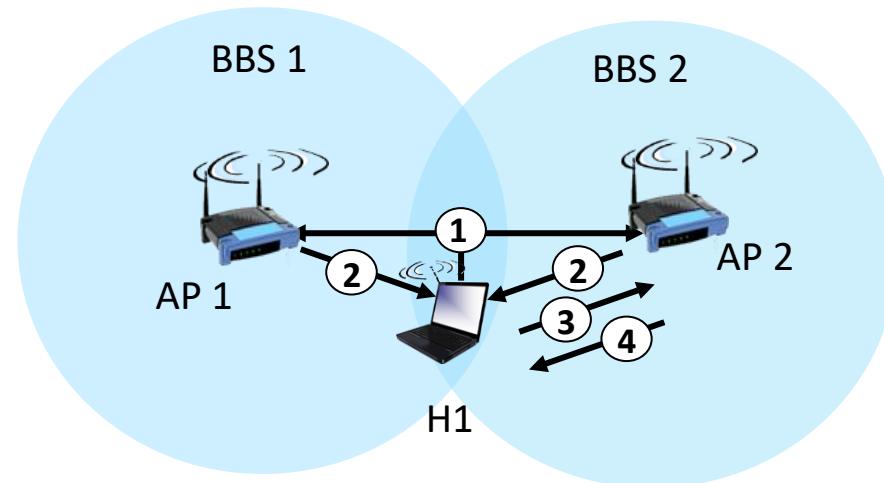


# 802.11: passive/active scanning



## passive scanning:

- (1) beacon frames sent from APs
- (2) association Request frame sent: H1 to selected AP
- (3) association Response frame sent from selected AP to H1

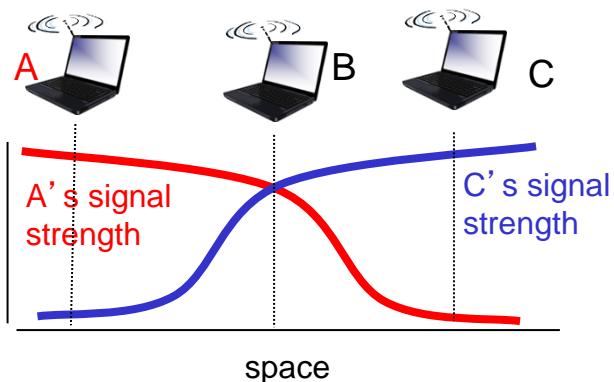
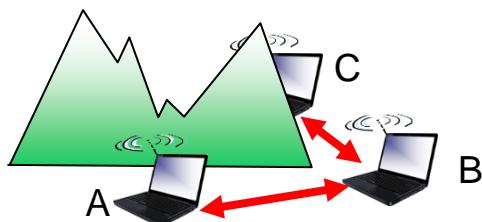


## active scanning:

- (1) Probe Request frame broadcast from H1
- (2) Probe Response frames sent from APs
- (3) Association Request frame sent: H1 to selected AP
- (4) Association Response frame sent from selected AP to H1

# IEEE 802.11: multiple access

- avoid collisions:  $2^+$  nodes transmitting at same time
- 802.11: CSMA - sense before transmitting
  - don't collide with detected ongoing transmission by another node
- 802.11: *no collision detection!*
  - difficult to sense collisions: high transmitting signal, weak received signal due to fading
  - can't sense all collisions in any case: hidden terminal, fading
  - goal: *avoid collisions:* CSMA/CollisionAvoidance



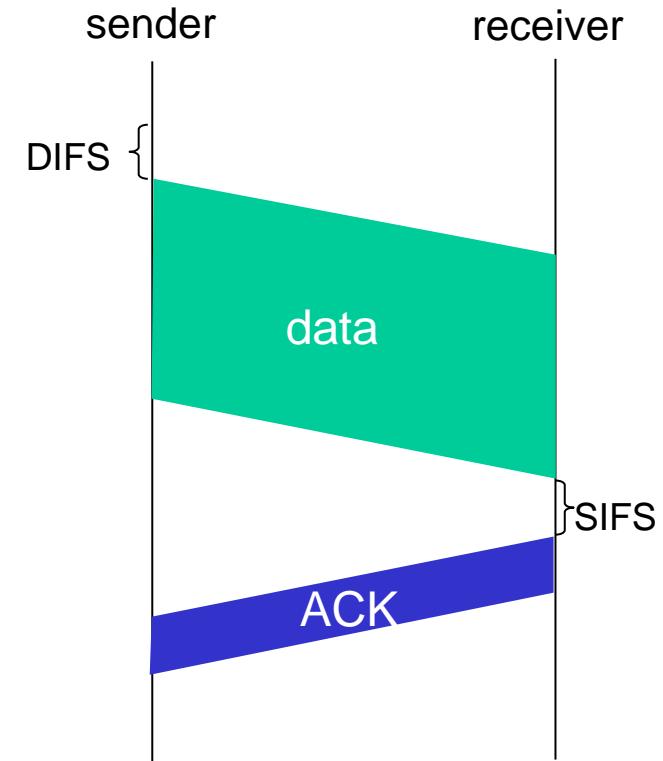
# IEEE 802.11 MAC Protocol: CSMA/CA

## 802.11 sender

- 1 if sense channel idle for **DIFS** then  
    transmit entire frame (no CD)
- 2 if sense channel busy then  
    start random backoff time  
    timer counts down while channel idle  
    transmit when timer expires  
    if no ACK, increase random backoff interval, repeat 2

## 802.11 receiver

if frame received OK  
    return ACK after **SIFS** (ACK needed due to hidden  
    terminal problem)

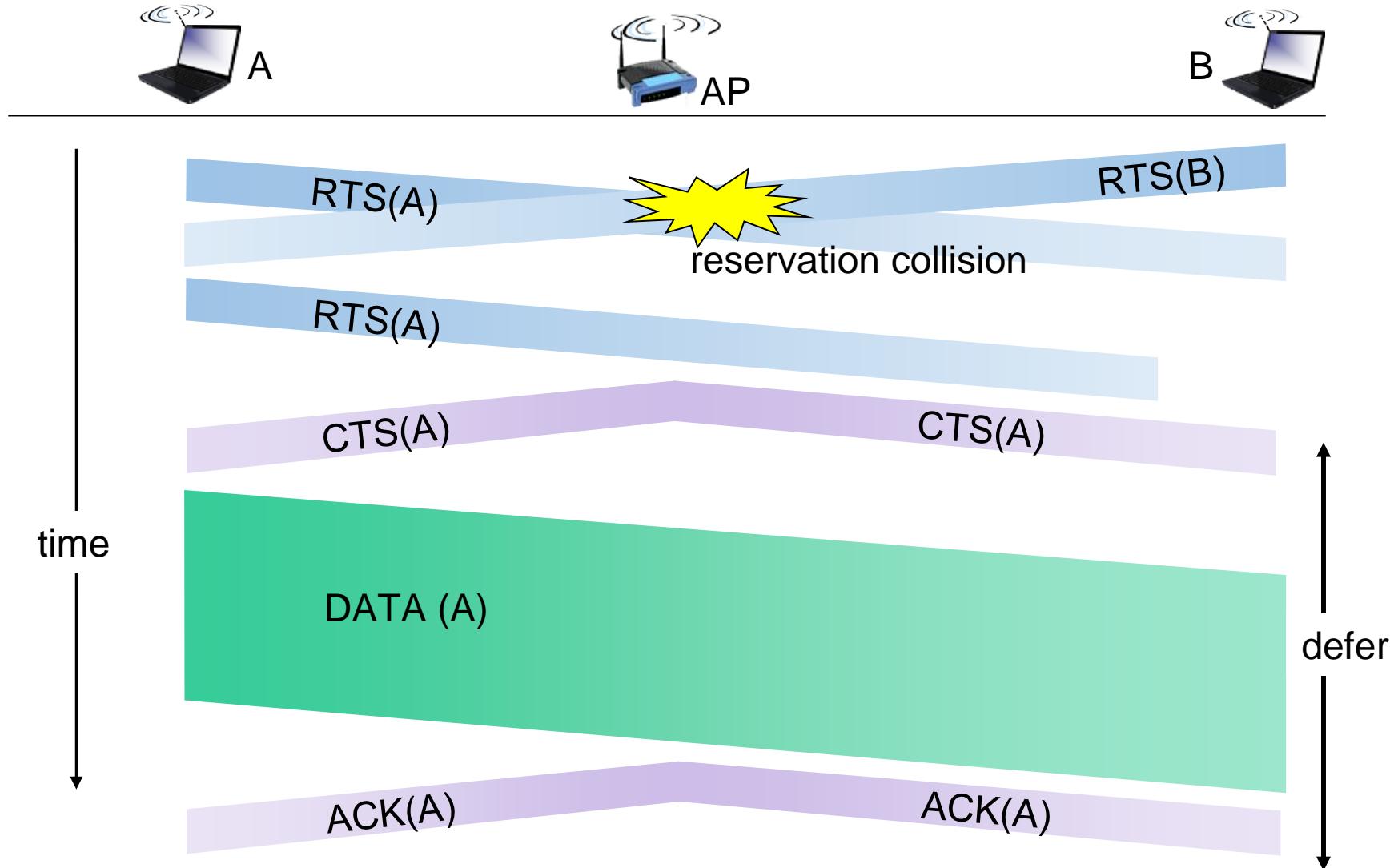


# Avoiding collisions (more)

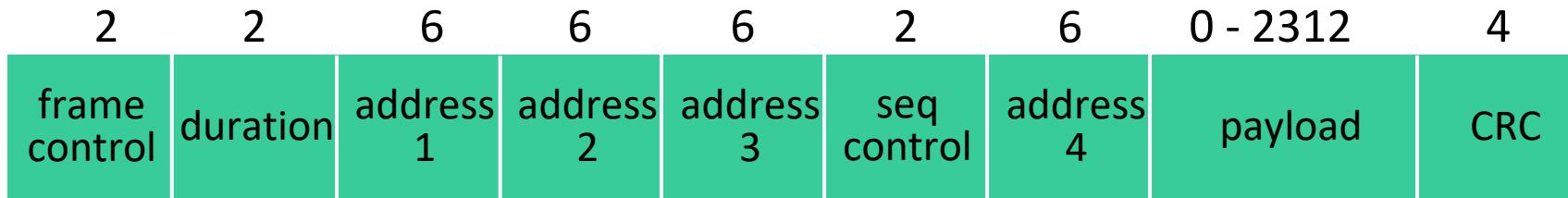
**idea:** sender “reserves” channel use for data frames using small reservation packets

- sender first transmits *small* request-to-send (RTS) packet to BS using CSMA
  - RTSs may still collide with each other (but they’re short)
- BS broadcasts clear-to-send CTS in response to RTS
- CTS heard by all nodes
  - sender transmits data frame
  - other stations defer transmissions

# Collision Avoidance: RTS-CTS exchange



# 802.11 frame: addressing



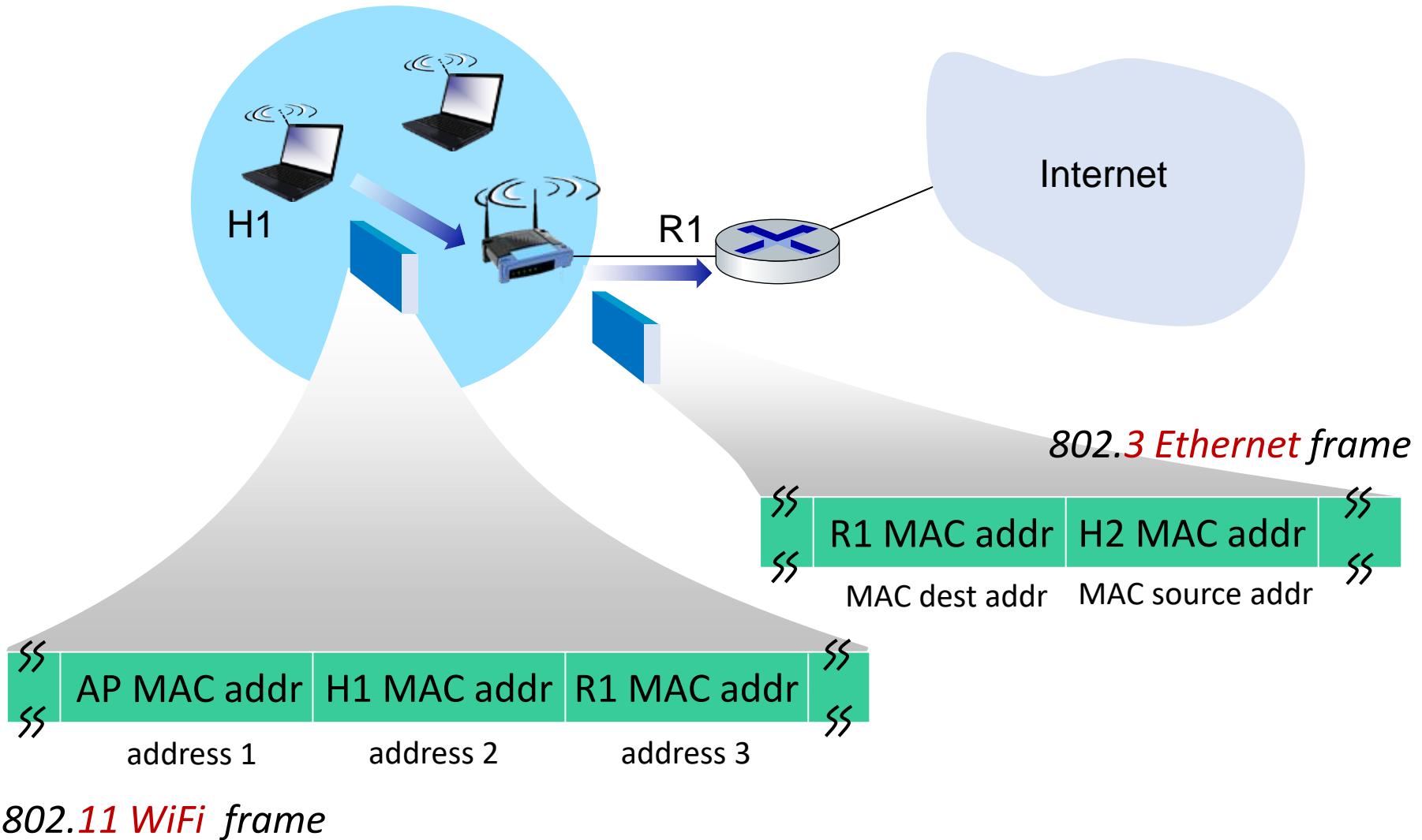
**Address 1:** MAC address of wireless host or AP to receive this frame

**Address 2:** MAC address of wireless host or AP transmitting this frame

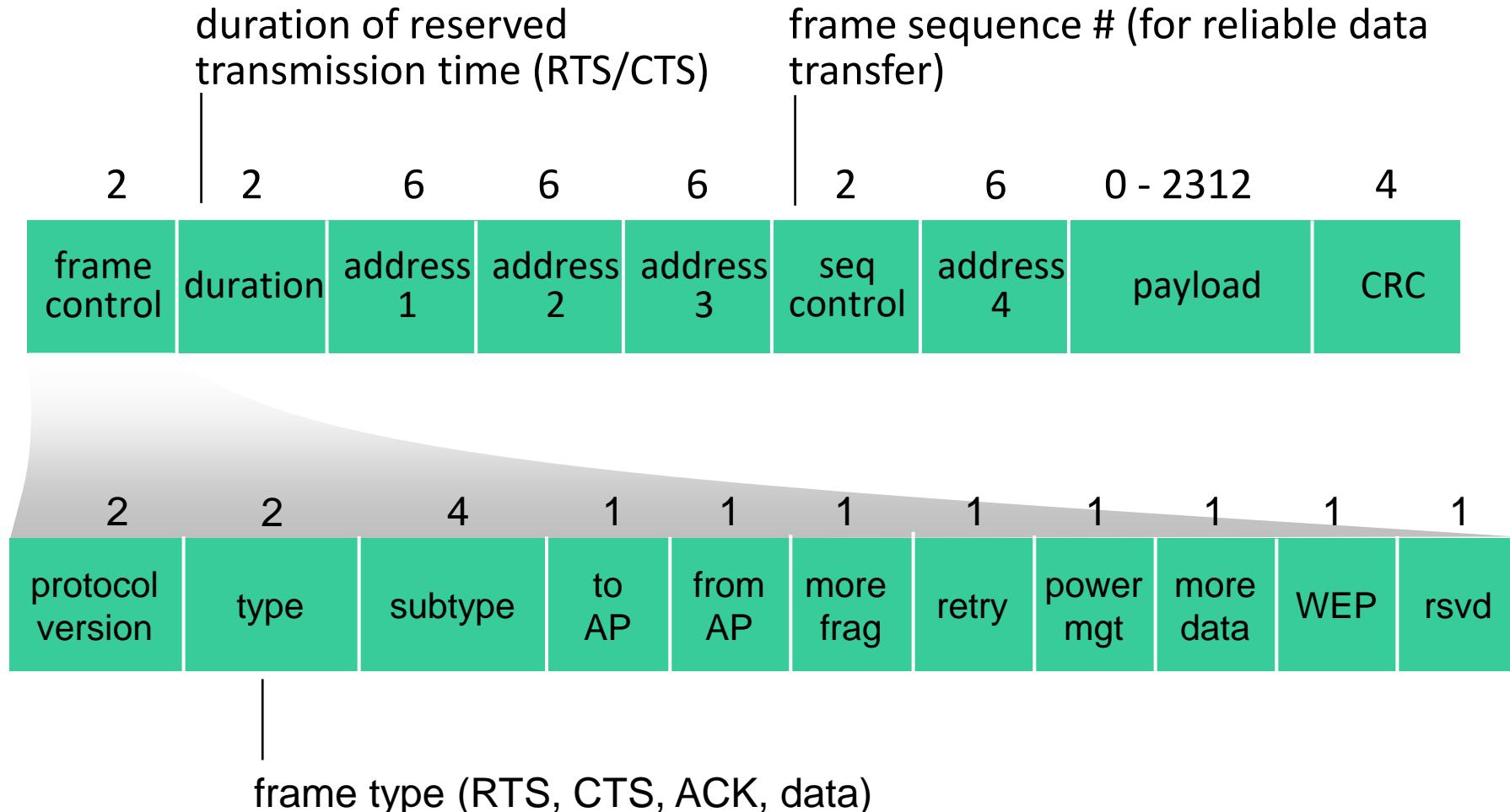
**Address 4:** used only in ad hoc mode

**Address 3:** MAC address of router interface to which AP is attached

# 802.11 frame: addressing

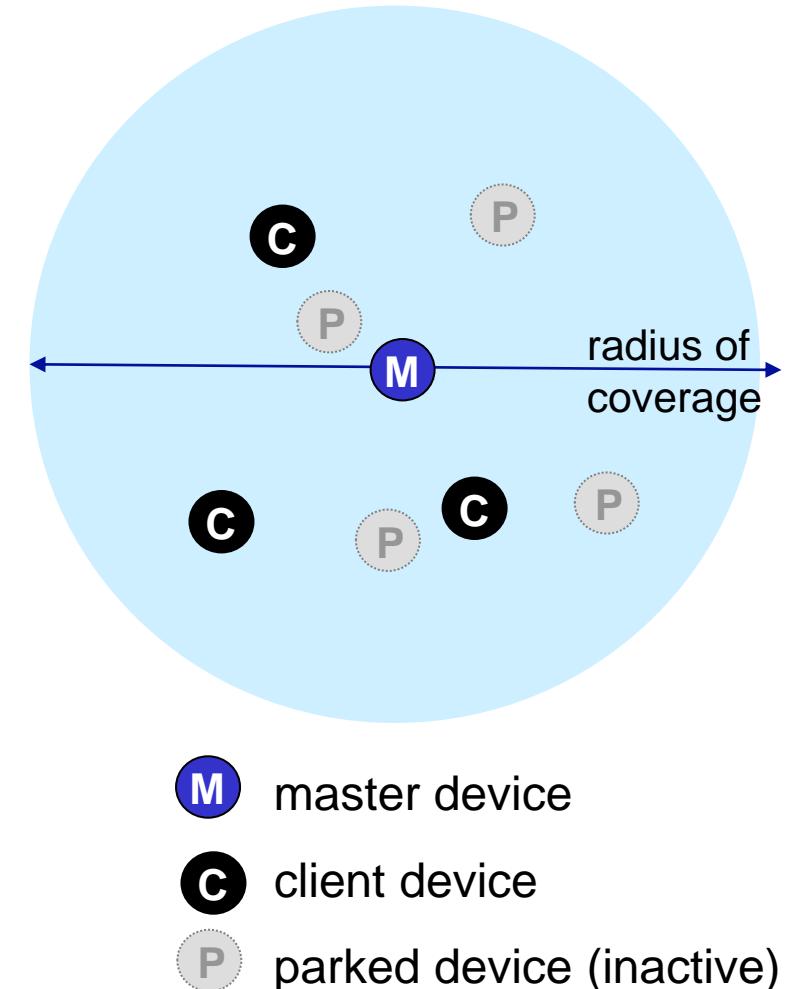


# 802.11 frame: addressing



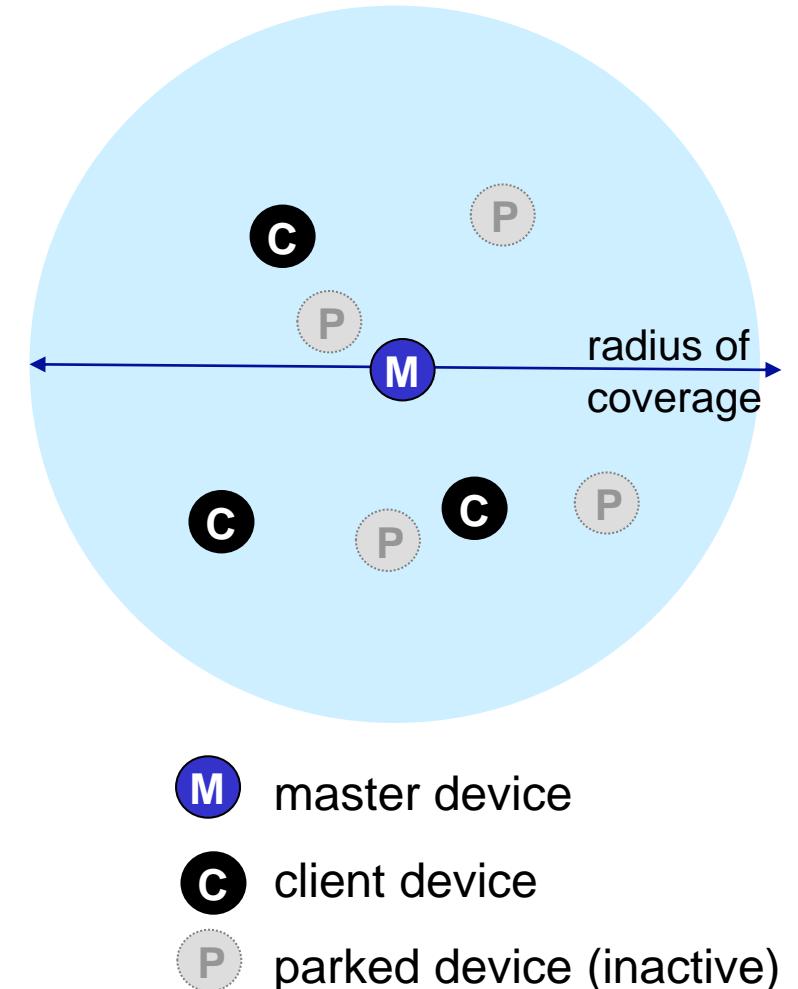
# Personal area networks: Bluetooth

- less than 10 m diameter
- replacement for cables (mouse, keyboard, headphones)
- ad hoc: no infrastructure
- 2.4-2.5 GHz ISM radio band, up to 3 Mbps
- master controller / client devices:
  - master polls clients, grants requests for client transmissions



# Personal area networks: Bluetooth

- TDM, 625  $\mu$ sec sec. slot
- FDM: sender uses 79 frequency channels in known, pseudo-random order slot-to-slot (spread spectrum)
  - other devices/equipment not in piconet only interfere in some slots
- **parked mode:** clients can “go to sleep” (park) and later wakeup (to preserve battery)
- **bootstrapping:** nodes self-assemble (plug and play) into piconet



# Outline

- Introduction

## Wireless

- Wireless links and network characteristics
- WiFi: 802.11 wireless LANs
- Cellular networks: 4G and 5G

## Mobility

- Mobility management: principles
- Mobility management: practice
  - 4G/5G networks
  - Mobile IP
- Mobility: impact on higher-layer protocols

# cellular networks

- *the solution for wide-area mobile Internet*
- widespread deployment/use:
  - more mobile-broadband-connected devices than fixed-broadband-connected devices (5-1 in 2019)!
  - 4G availability: 97% of time in Korea (90% in US)
- transmission rates up to 100's Mbps
- technical standards: 3rd Generation Partnership Project (3GPP)
  - [www.3gpp.org](http://www.3gpp.org)
  - 4G: Long-Term Evolution (LTE)standard

# cellular networks

## *similarities to wired Internet*

- edge/core distinction, but both belong to same carrier
- global cellular network: a network of networks
- widespread use of protocols we've studied: HTTP, DNS, TCP, UDP, IP, NAT, separation of data/control planes, SDN, Ethernet, tunneling
- interconnected to wired Internet

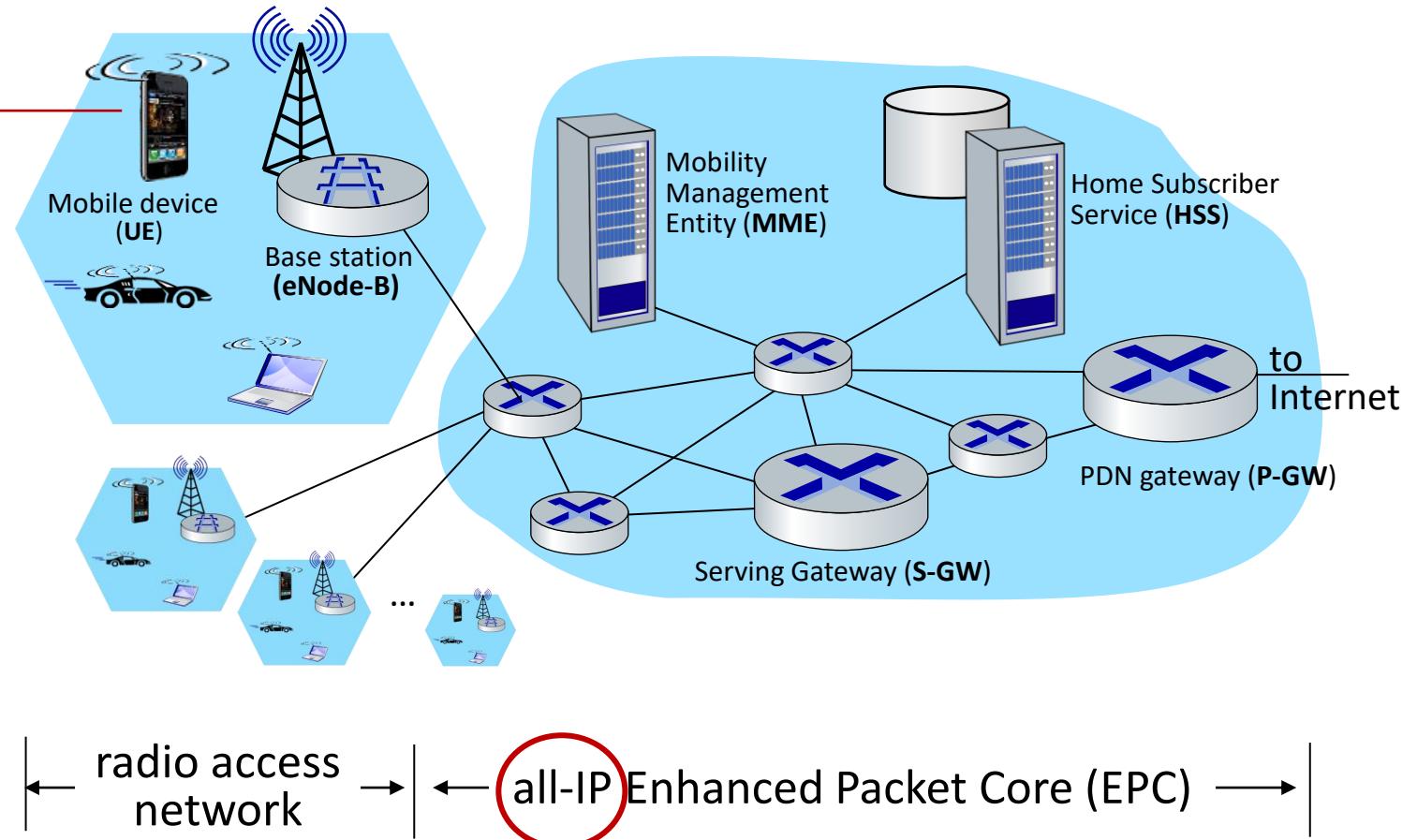
## *differences from wired Internet*

- different wireless link layer
- mobility as a 1<sup>st</sup> class service
- user "identity" (via SIM card)
- business model: users subscribe to a cellular provider
  - strong notion of "home network" versus roaming on visited nets
  - global access, with authentication infrastructure, and inter-carrier settlements

# Elements of 4G LTE architecture

## Mobile device:

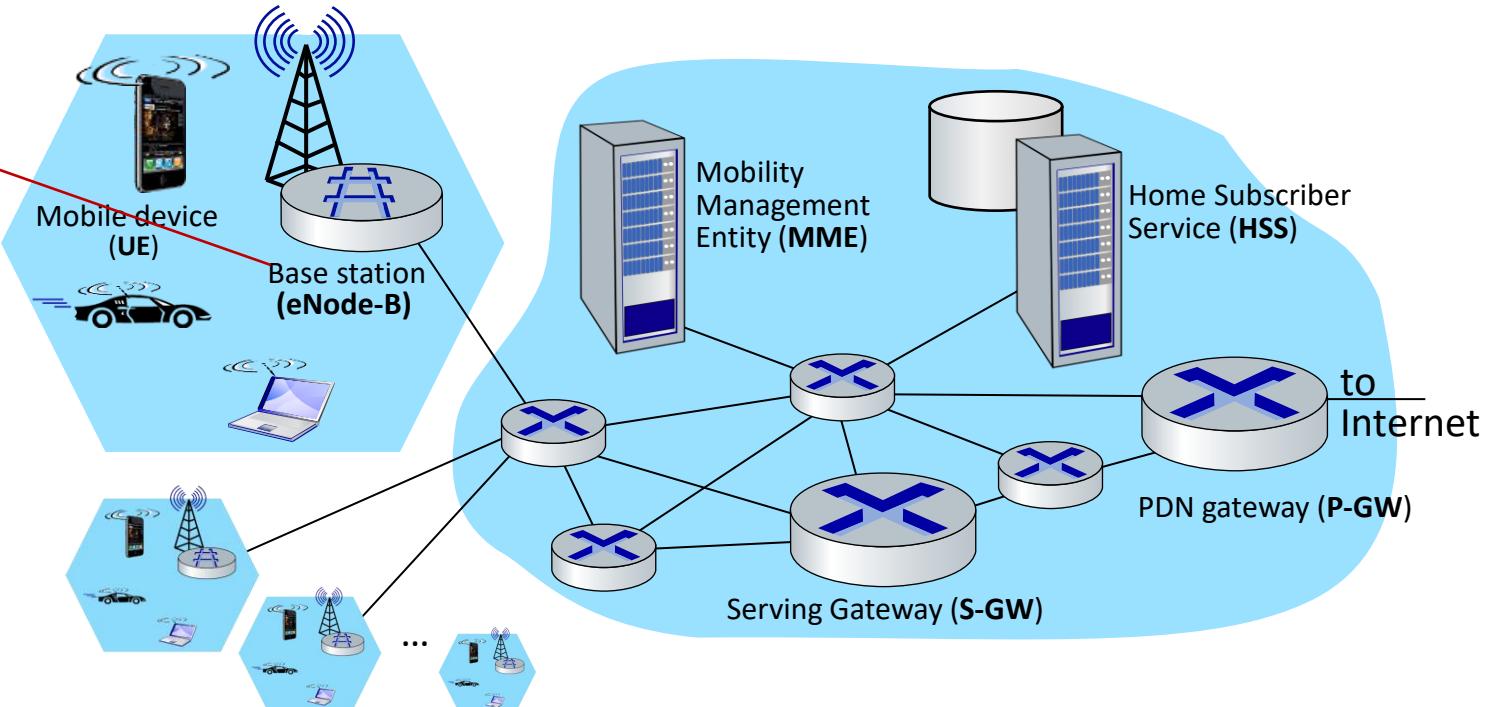
- smartphone, tablet, laptop, IoT, ... with 4G LTE radio
- 64-bit International Mobile Subscriber Identity (IMSI), stored on SIM (Subscriber Identity Module) card
- LTE jargon: User Equipment (UE)



# Elements of 4G LTE architecture

## Base station:

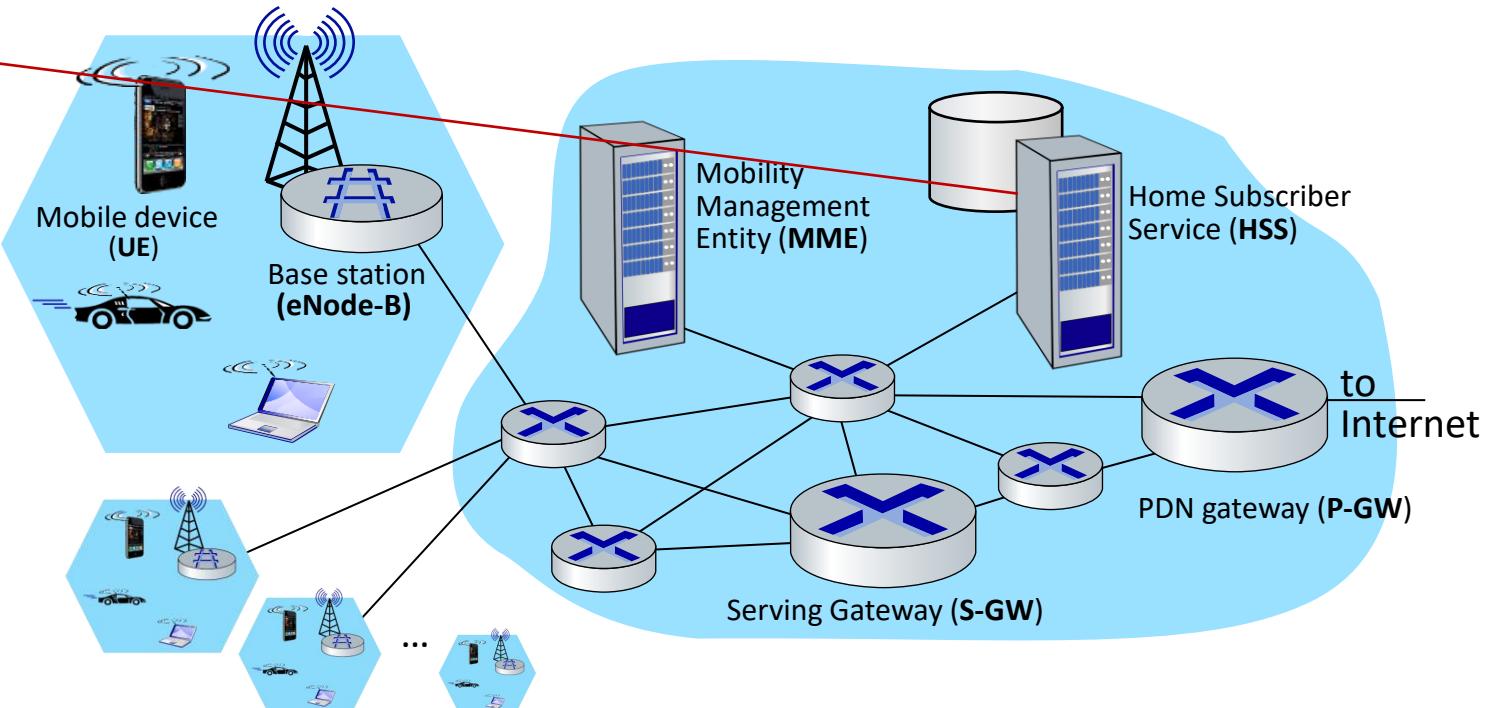
- at “edge” of carrier’s network
- manages wireless radio resources, mobile devices in its coverage area (“cell”)
- coordinates device authentication with other elements
- similar to WiFi AP but:
  - active role in user mobility
  - coordinates with nearby base stations to optimize radio use
- LTE jargon: eNode-B



# Elements of 4G LTE architecture

## Home Subscriber Service

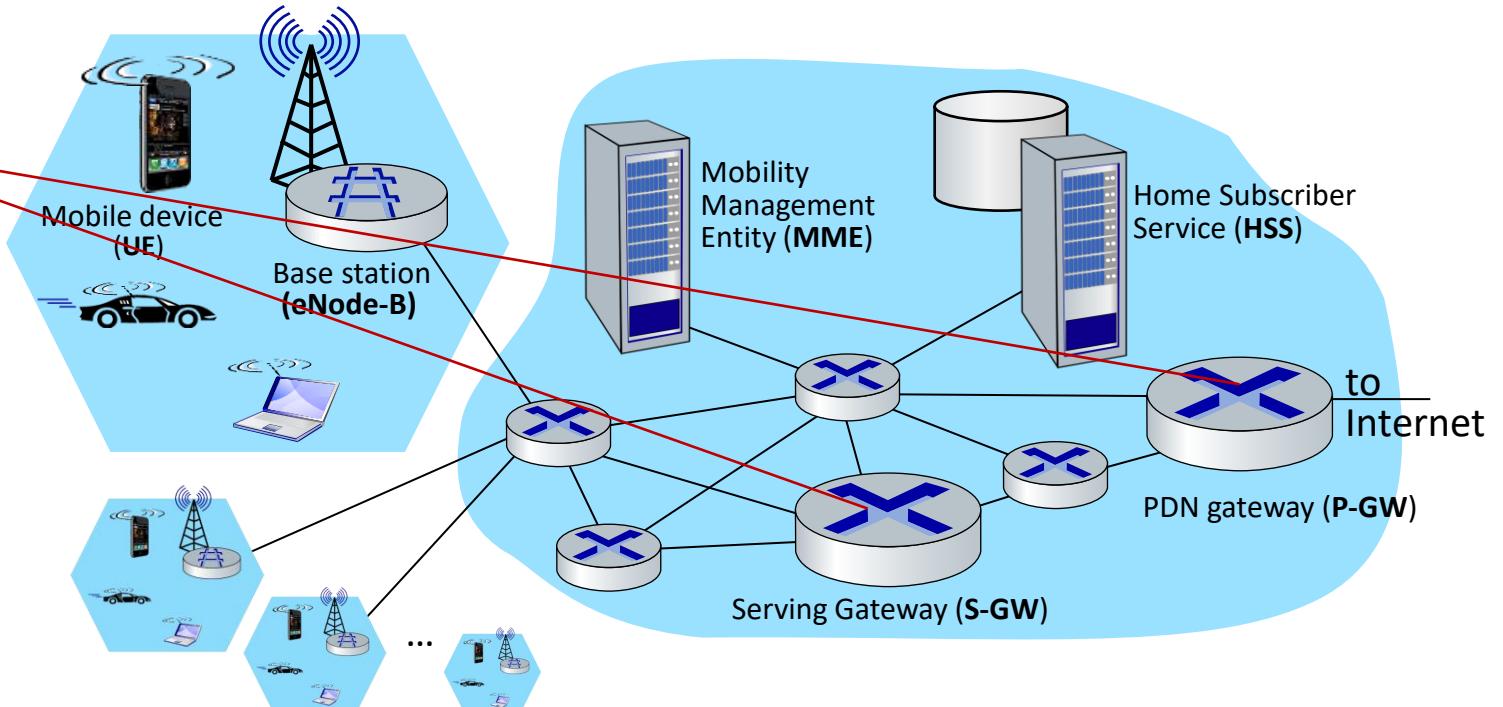
- stores info about mobile devices for which the HSS's network is their "home network"
- works with MME in device authentication



# Elements of 4G LTE architecture

## Serving Gateway (S-GW), PDN Gateway (P-GW)

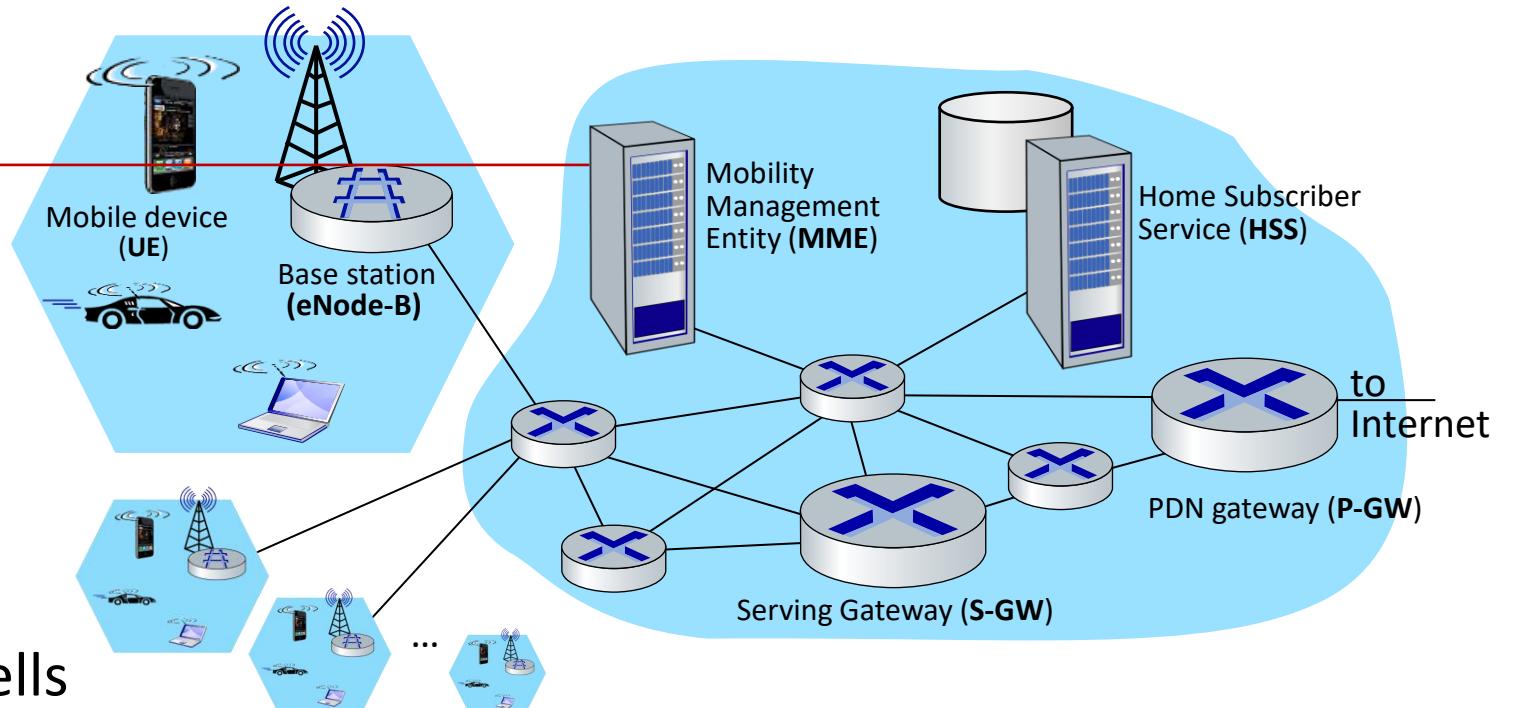
- lie on data path from mobile to/from Internet
- P-GW
  - gateway to mobile cellular network
  - Looks like any other internet gateway router
  - provides NAT services
- other routers:
  - extensive use of tunneling



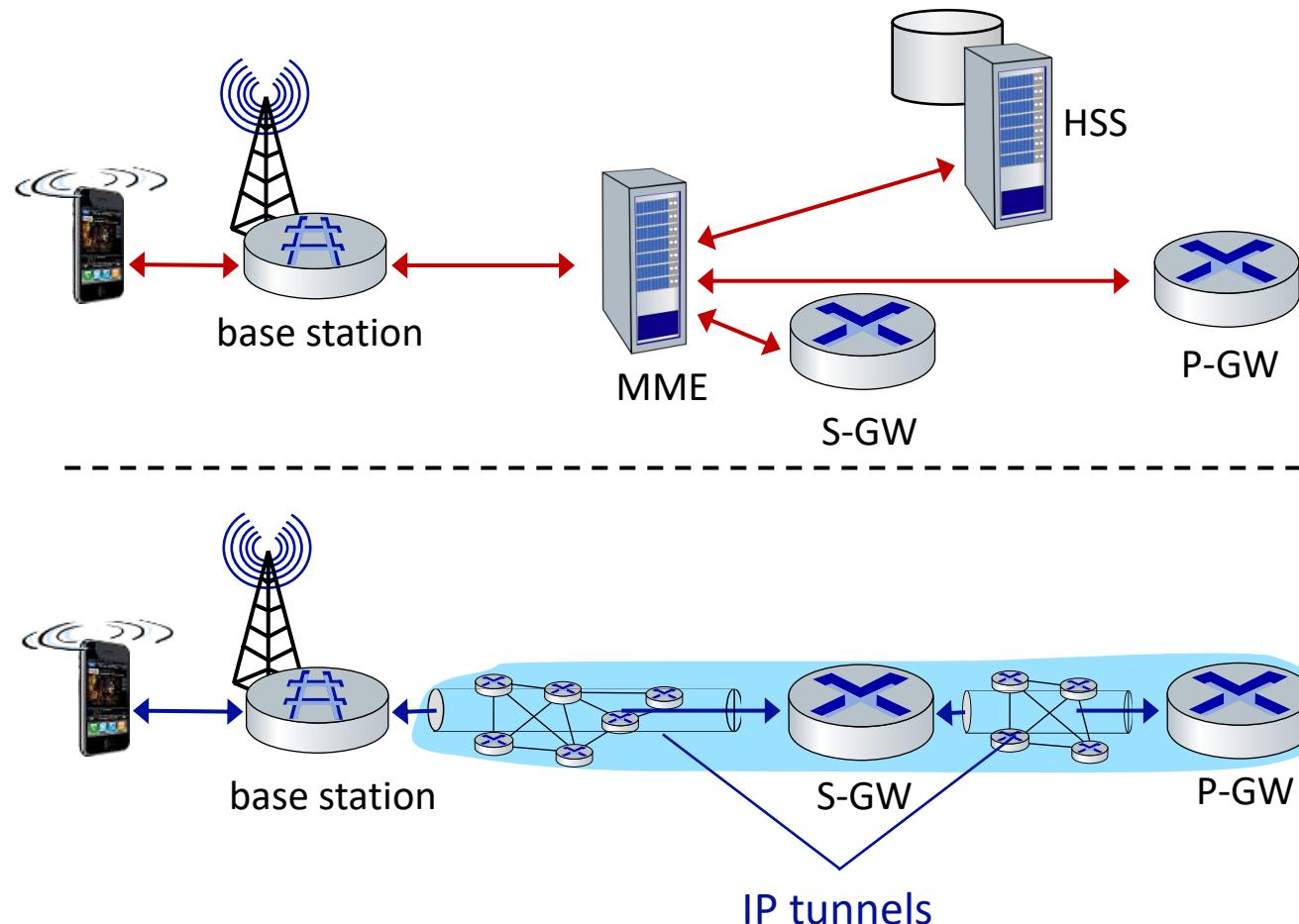
# Elements of 4G LTE architecture

## Mobility Management Entity

- device authentication (device-to-network, network-to-device) coordinated with mobile home network HSS
- mobile device management:
  - device handover between cells
  - tracking/paging device location
- path (tunneling) setup from mobile device to P-GW



# LTE: data plane control plane separation



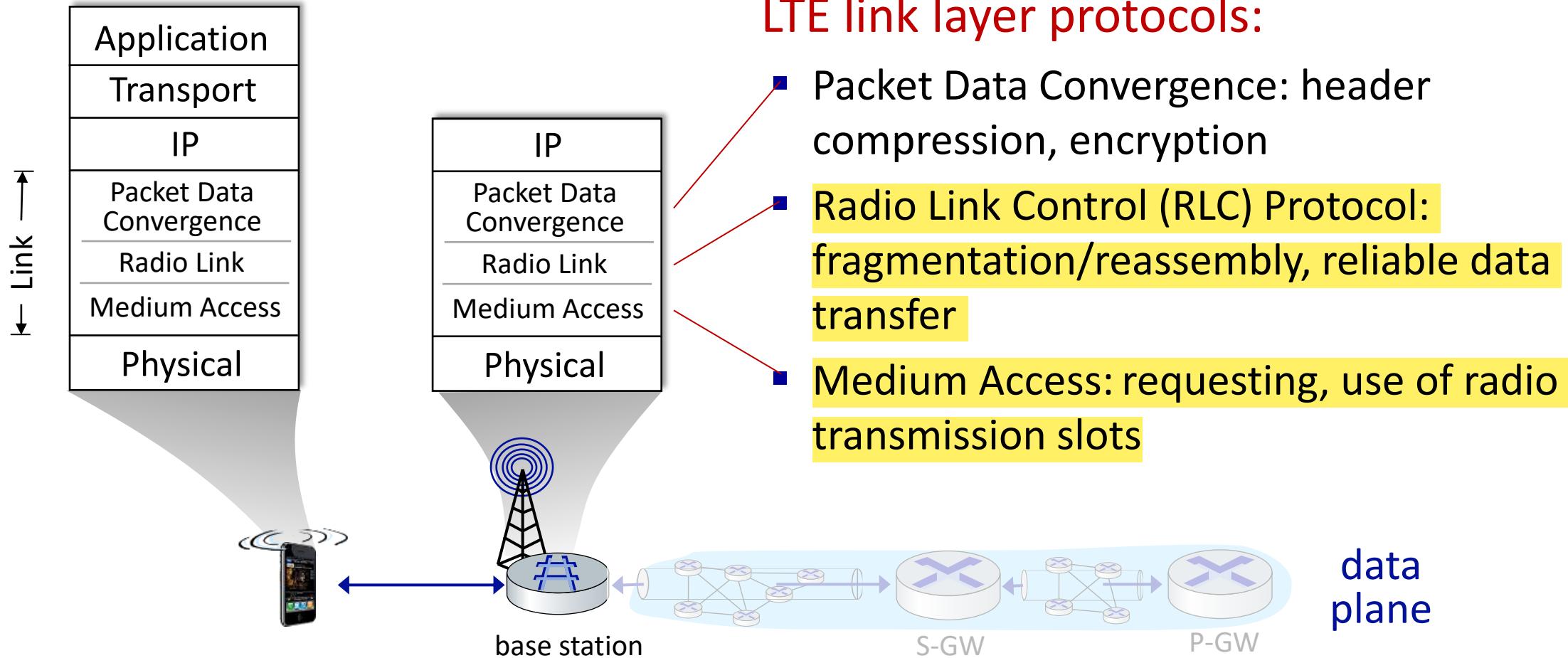
## control plane

- new protocols for mobility management , security, authentication (later)

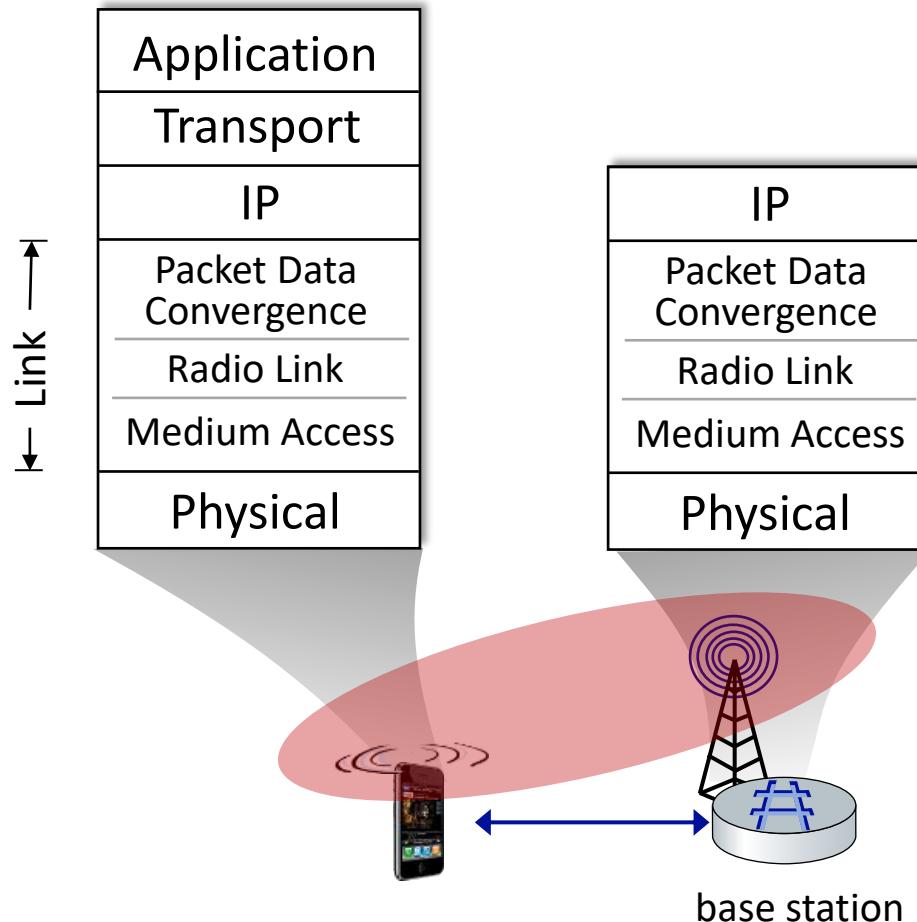
## data plane

- new protocols at link, physical layers
- extensive use of tunneling to facilitate mobility

# LTE data plane protocol stack: first hop



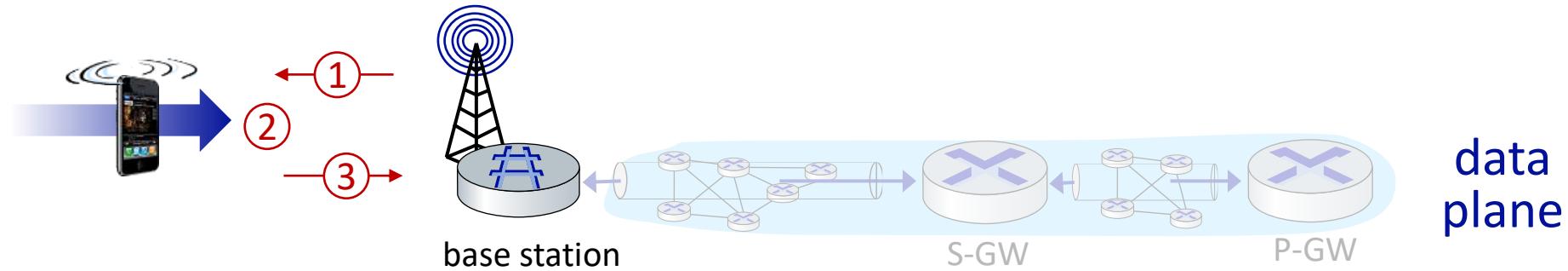
# LTE data plane protocol stack: first hop



## LTE radio access network:

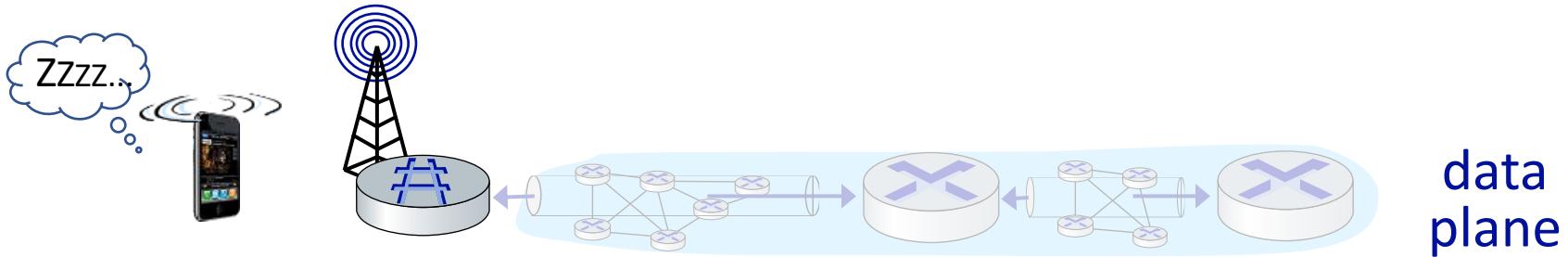
- downstream channel: FDM, TDM within frequency channel (OFDM - orthogonal frequency division multiplexing)
  - “orthogonal”: minimal interference between channels
- upstream: FDM, TDM similar to OFDM
- each active mobile device allocated two or more 0.5 ms time slots over 12 frequencies
  - scheduling algorithm not standardized – up to operator
  - 100's Mbps per device possible

# LTE data plane: associating with a BS



- ① BS broadcasts primary sync signal every 5 ms on all frequencies
  - BSs from multiple carriers may be broadcasting sync signals
- ② mobile finds a primary sync signal, then locates 2<sup>nd</sup> sync signal on this freq.
  - mobile then finds info broadcast by BS: channel bandwidth, configurations; BS's cellular carrier info
  - mobile may get info from multiple base stations, multiple cellular networks
- ③ mobile selects which BS to associate with (*e.g.*, preference for home carrier)
- ④ more steps still needed to authenticate, establish state, set up data plane

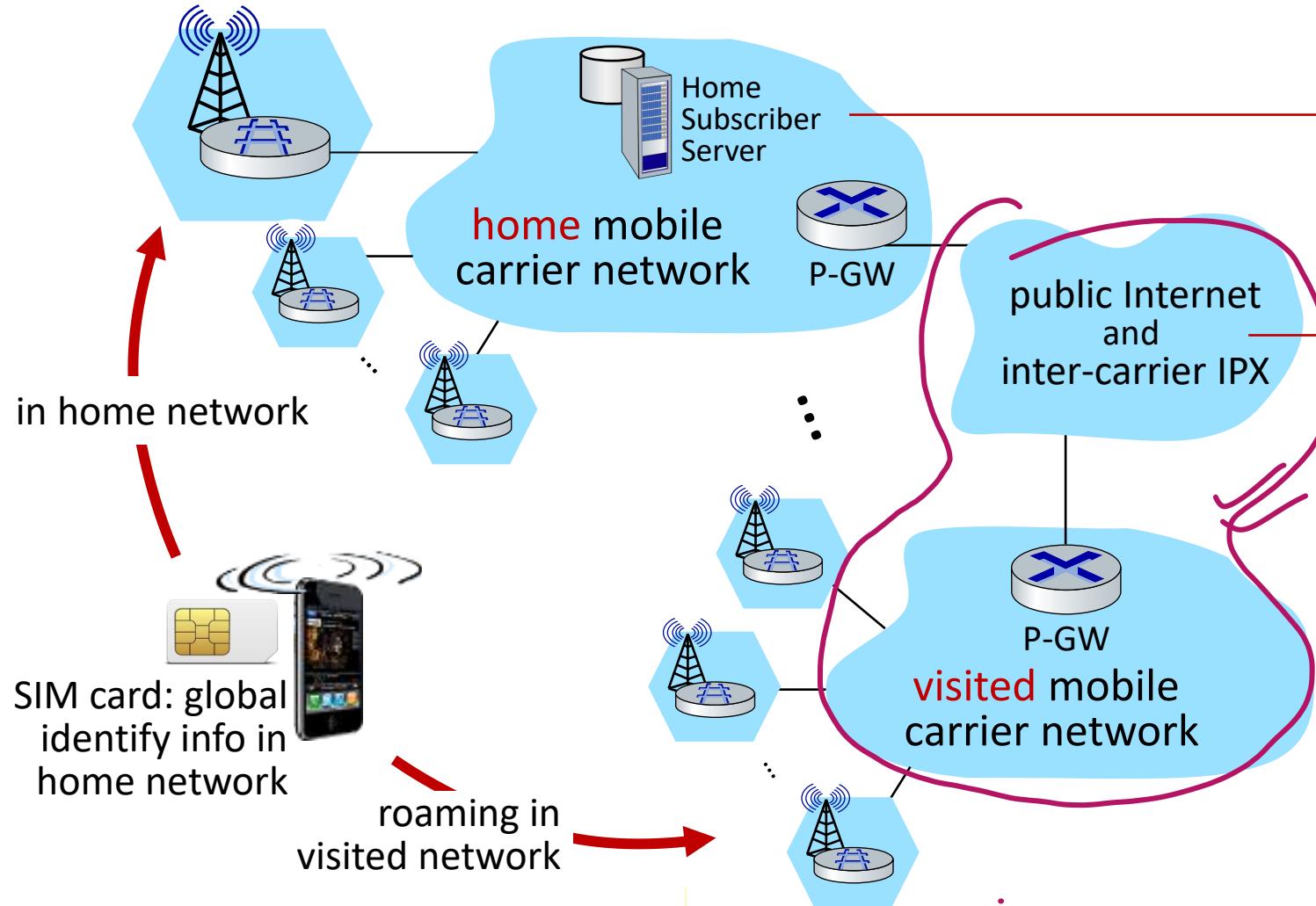
# LTE mobiles: sleep modes



as in WiFi, Bluetooth: LTE mobile may put radio to “sleep” to conserve battery:

- **light sleep:** after 100's msec of inactivity
  - wake up periodically (100's msec) to check for downstream transmissions
- **deep sleep:** after 5-10 secs of inactivity
  - mobile may change cells while deep sleeping – need to re-establish association

# Global cellular network: a network of IP networks



## home network HSS:

- identify & services info, while in home network and roaming

## all IP:

- carriers interconnect with each other, and public internet at exchange points
- legacy 2G, 3G: not all IP, handled otherwise

# Outline

- Introduction

## Wireless

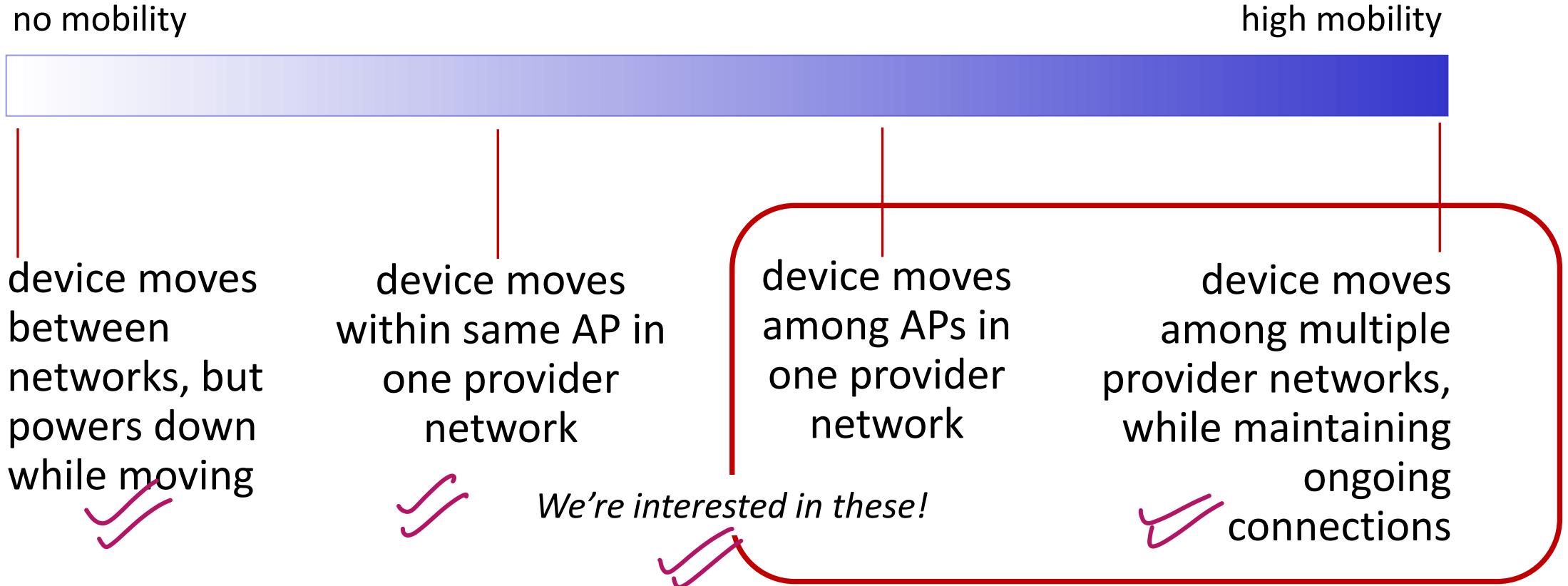
- Wireless links and network characteristics
- WiFi: 802.11 wireless LANs
- Cellular networks: 4G and 5G

## Mobility

- Mobility management: principles
- Mobility management: practice
  - 4G/5G networks
  - Mobile IP
- Mobility: impact on higher-layer protocols

# What is mobility?

- spectrum of mobility, from the **network** perspective:



# Mobility approaches

- let network (routers) handle it:
  - routers advertise well-known name, address (e.g., permanent 32-bit IP address), or number (e.g., cell #) of visiting mobile node via usual routing table exchange
  - Internet routing could do this already *with no* changes! Routing tables indicate where each mobile located via longest prefix match!

# Mobility approaches

- let network (routers) handle it:
  - routers advertise well-known address (e.g., permanent 32-bit IP address), or number of visiting mobile node via usual routing table exchange
    - not scalable to billions of mobiles
  - Internet routing could do the same *with no changes!* Routing tables indicate where each mobile located via longest prefix match!
- let end-systems handle it: functionality at the “edge”
  - ✓ *indirect routing:* communication from correspondent to mobile goes through home network, then forwarded to remote mobile
  - ✓ *direct routing:* correspondent gets foreign address of mobile, send directly to mobile

# Contacting a mobile friend:

Consider friend frequently changing locations, how do you find him/her?

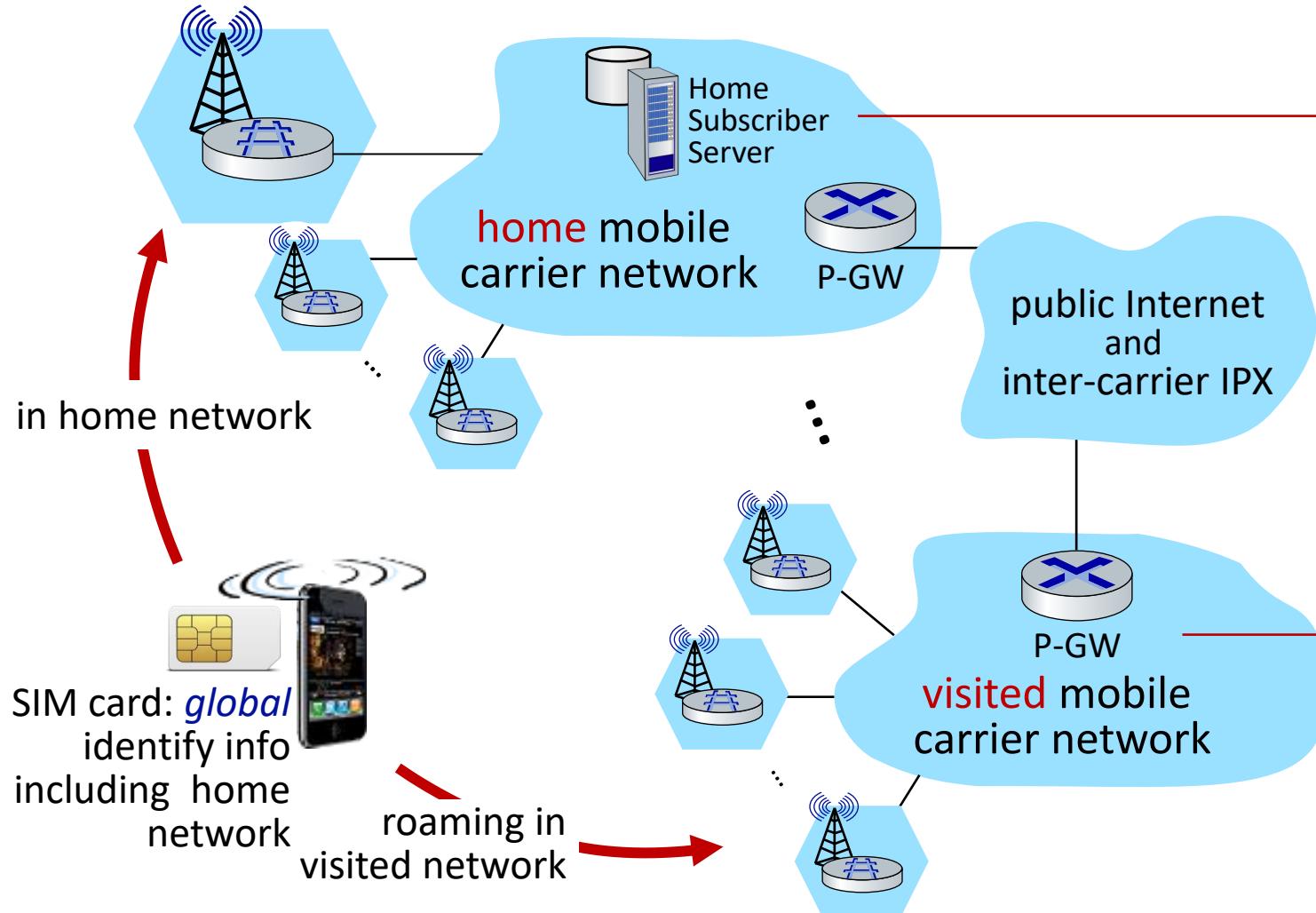
- search all phone books?
- expect her to let you know where he/she is?
- call his/her parents?
- Facebook!



The importance of having a “home”:

- a definitive source of information about you
- a place where people can find out where you are

# Home network, visited network: 4G/5G



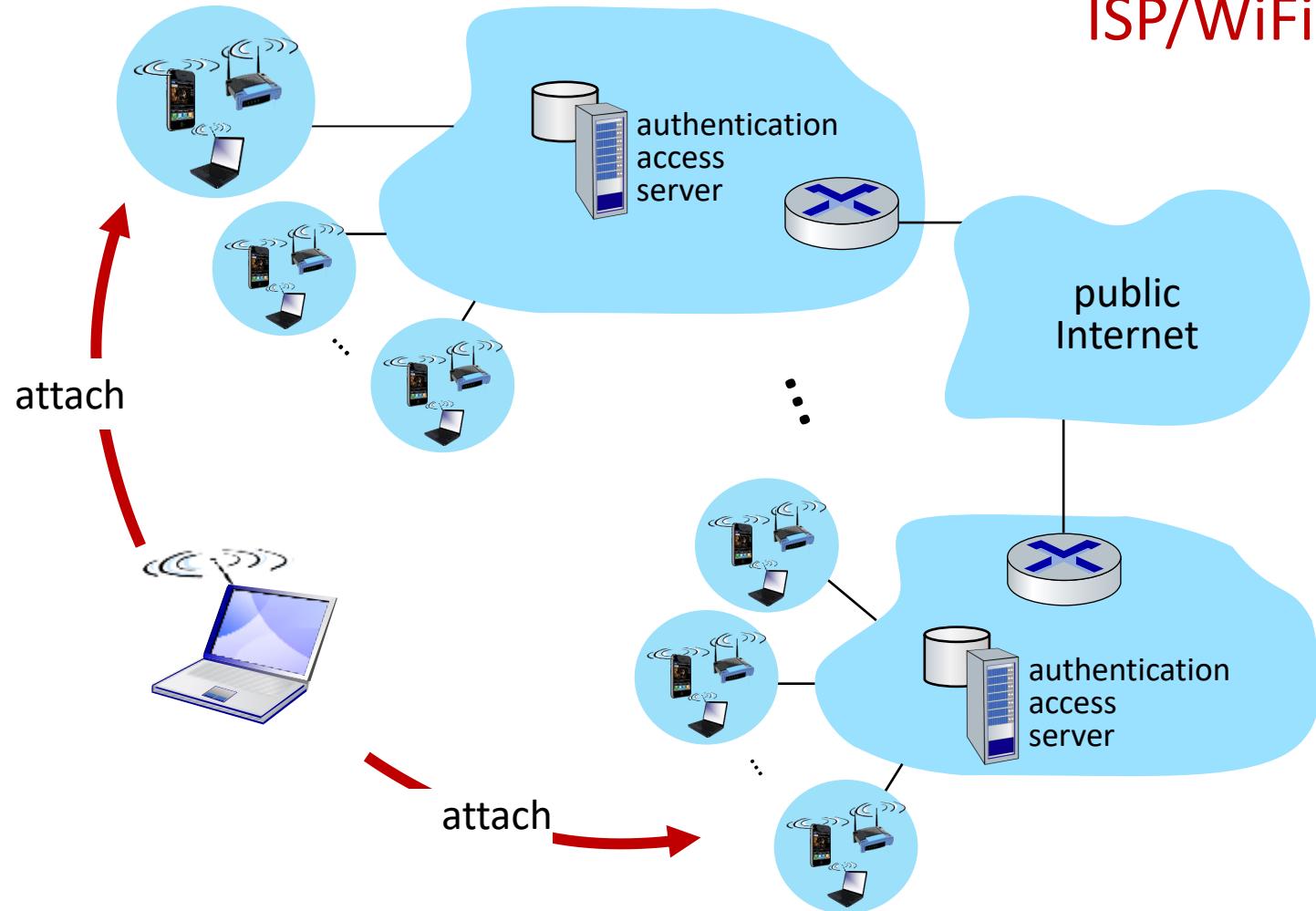
## home network:

- (paid) service plan with cellular provider, e.g., Verizon, Orange
- home network HSS stores identify & services info

## visited network:

- any network other than your home network
- service agreement with other networks: to provide access to visiting mobile

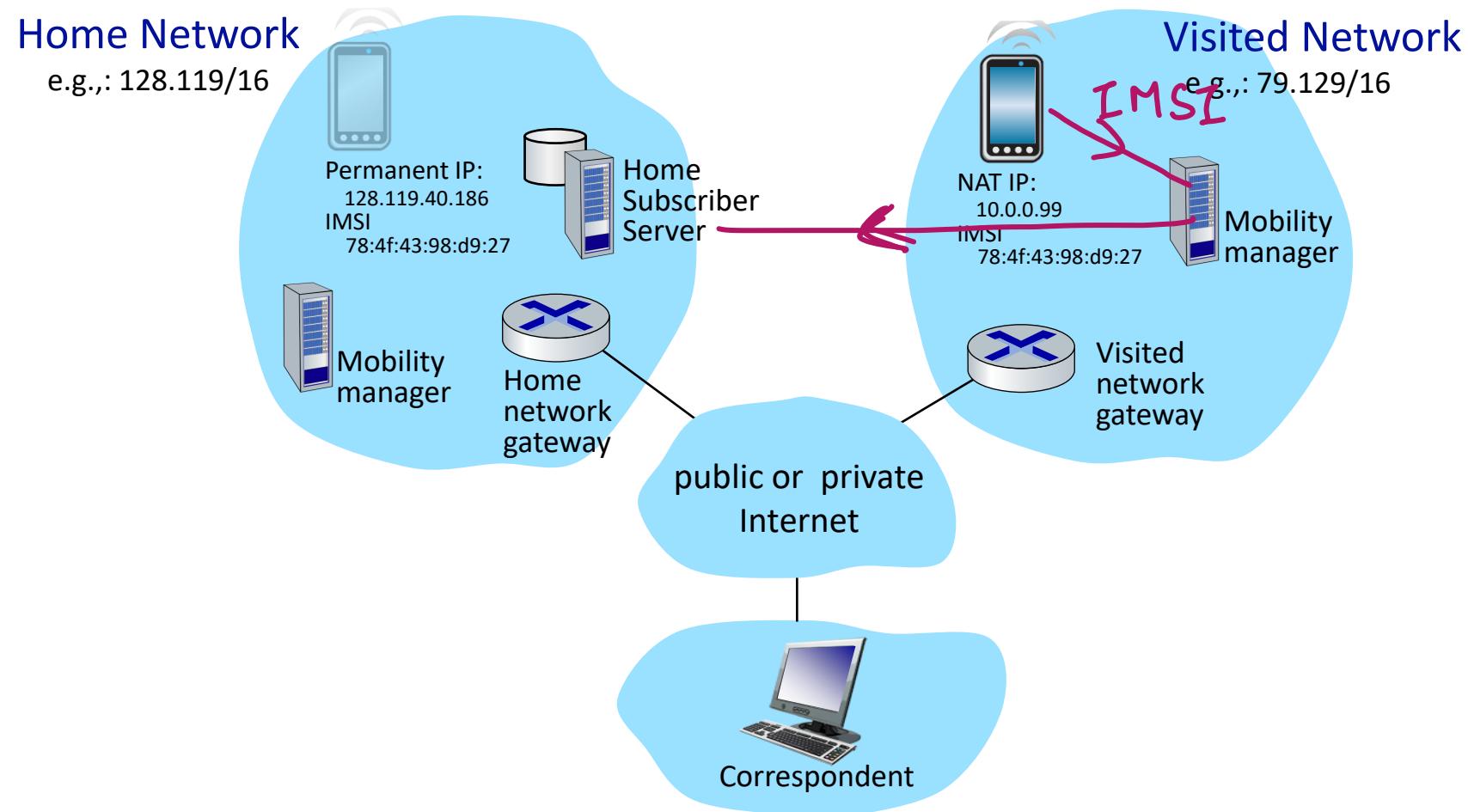
# Home network, visited network: ISP/WiFi



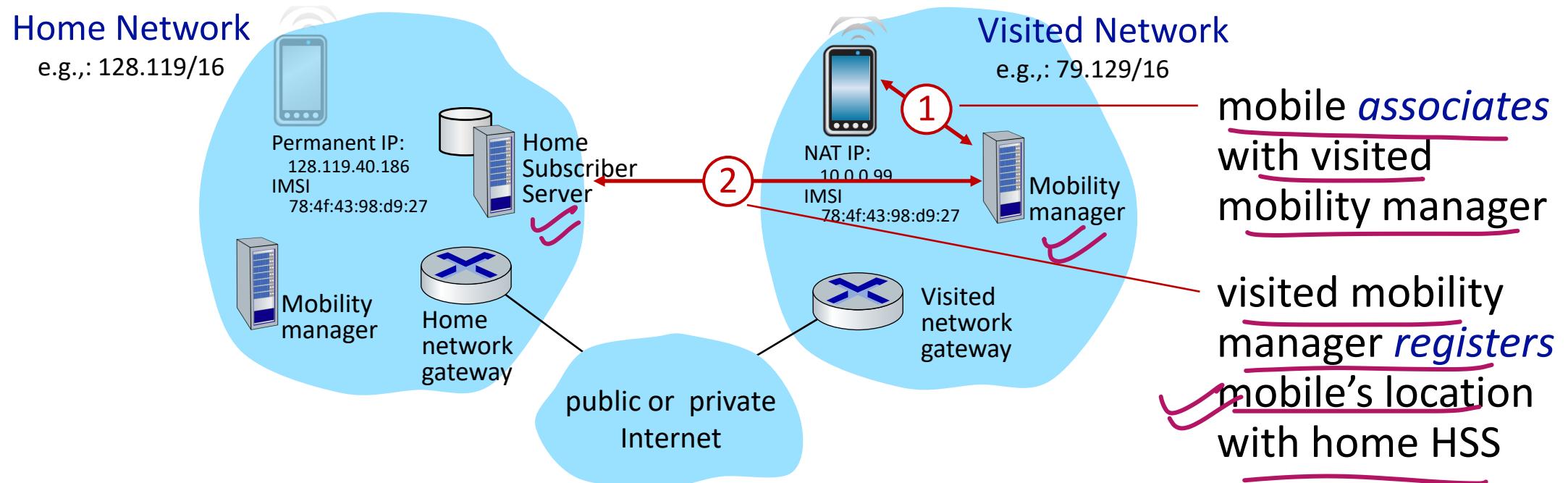
ISP/WiFi: no notion of global “home”

- credentials from ISP (e.g., username, password) stored on device or with user
- ISPs may have national, international presence
- different networks: different credentials
  - some exceptions (e.g., eduroam)
  - architectures exist (mobile IP) for 4G-like mobility, but not used

# Home network, visited network: generic



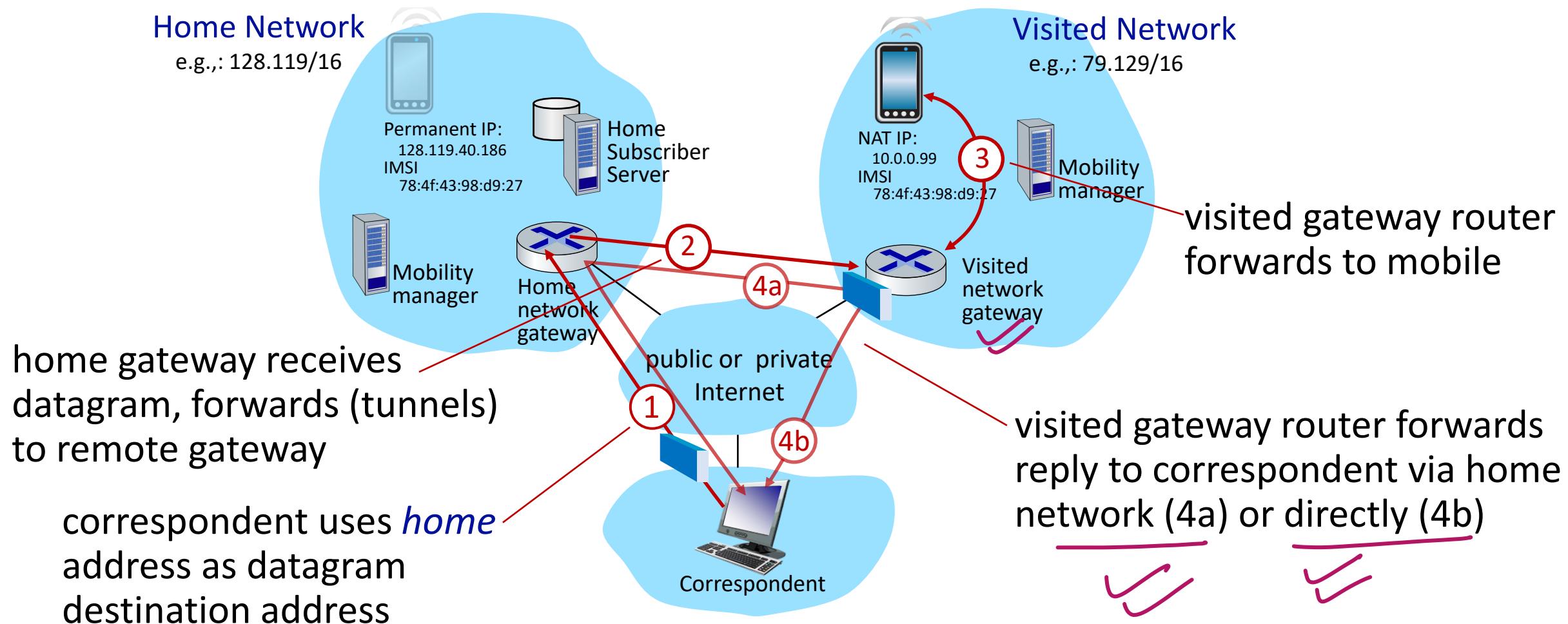
# Registration: home needs to know where you are!



end result:

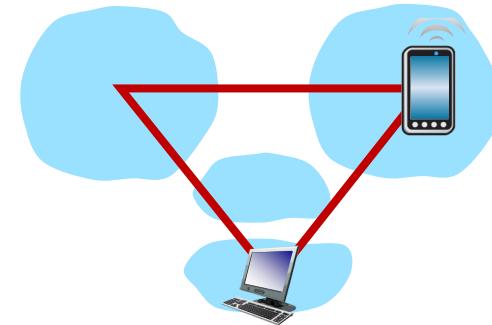
- ✓ visited mobility manager knows about mobile
- ✓ home HSS knows location of mobile

# Mobility with indirect routing

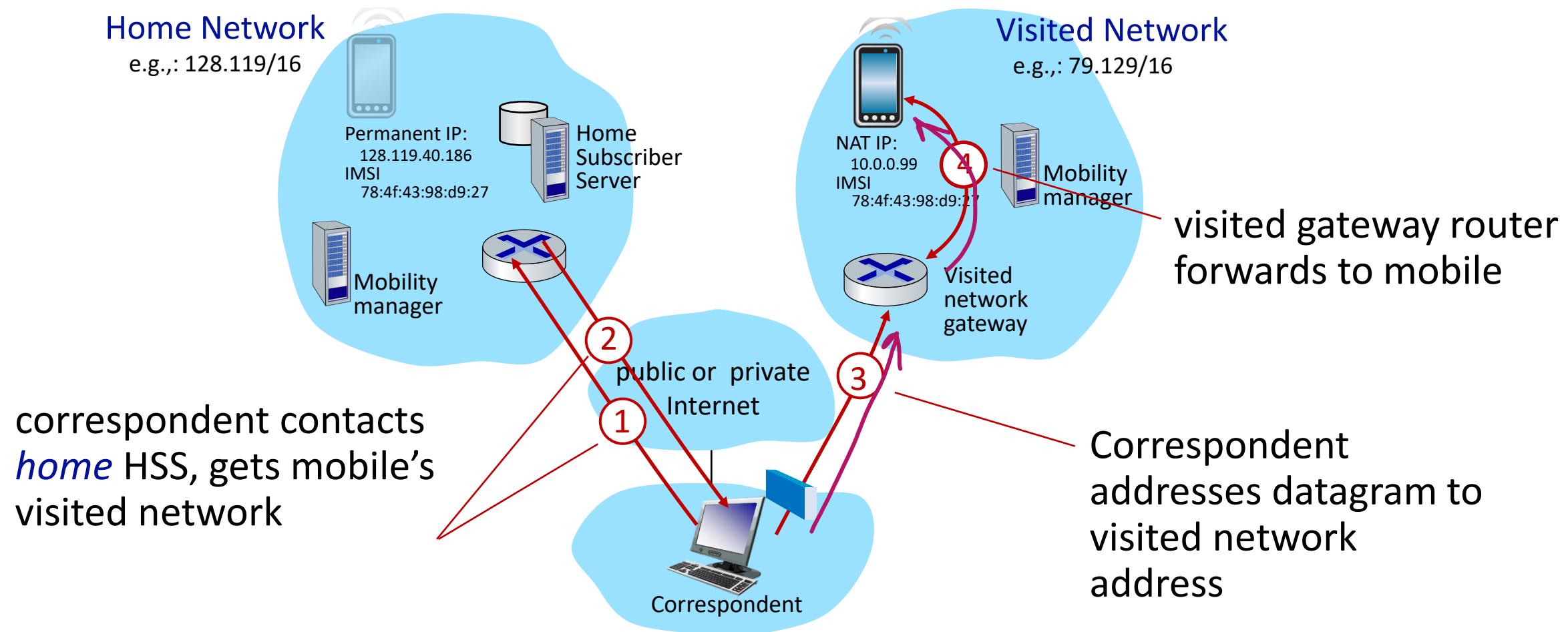


# Mobility with indirect routing: comments

- triangle routing:
  - ✓ inefficient when correspondent and mobile are in same network
- mobile moves among visited networks: transparent to correspondent!
  - ✓ registers in new visited network
  - ✓ new visited network registers with home HSS
  - ✓ datagrams continue to be forwarded from home network to mobile in new network
  - ✓ *on-going (e.g., TCP) connections between correspondent and mobile can be maintained!*



# Mobility with direct routing



# Mobility with direct routing: comments

- overcomes triangle routing inefficiencies
- *non-transparent to correspondent*: correspondent must get care-of-address from home agent
- what if mobile changes visited network?  
✓ can be handled, but with additional complexity

# Outline

- Introduction

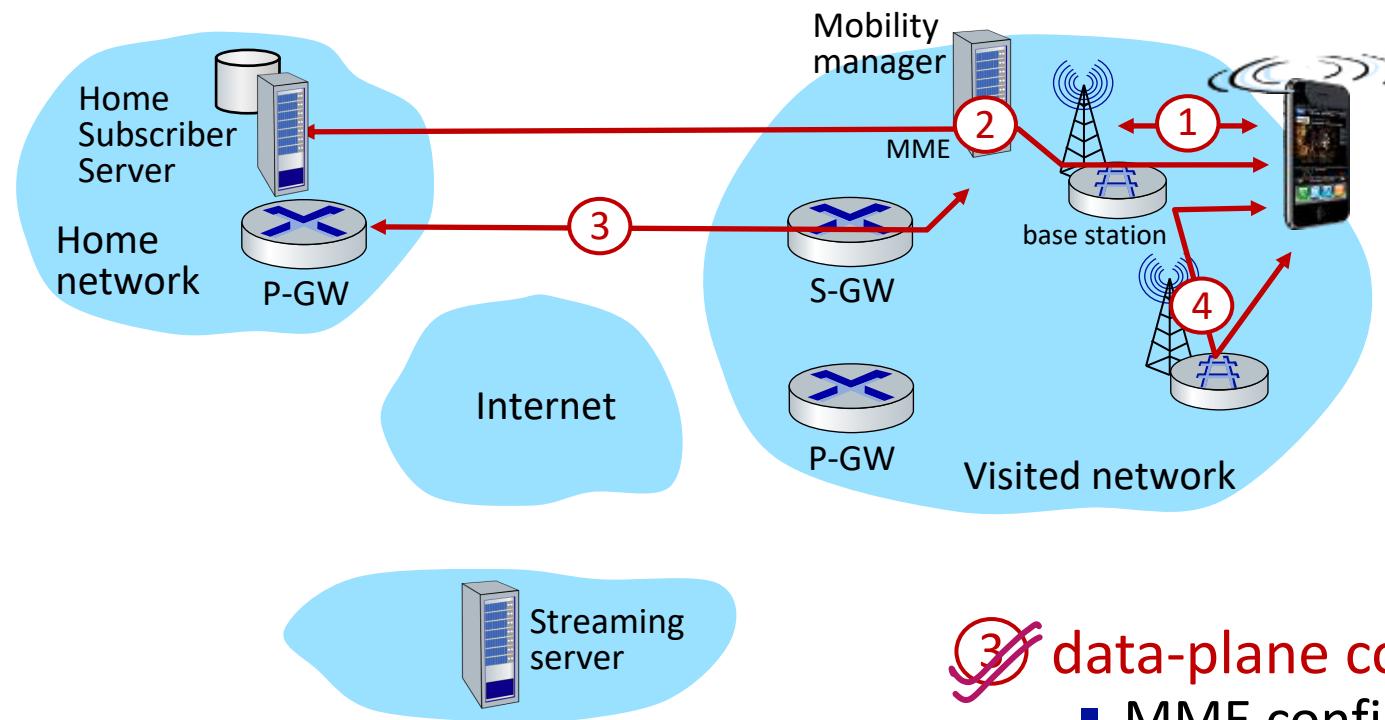
## Wireless

- Wireless links and network characteristics
- WiFi: 802.11 wireless LANs
- Cellular networks: 4G and 5G

## Mobility

- Mobility management: principles
- **Mobility management: practice**
  - 4G/5G networks
  - Mobile IP
- Mobility: impact on higher-layer protocols

# Mobility in 4G networks: major mobility tasks



✓  
① **base station association:**

- covered earlier
- mobile provides IMSI – identifying itself, home network

✗  
② **control-plane configuration:**

- MME, home HSS establish control-plane state - mobile is in visited network

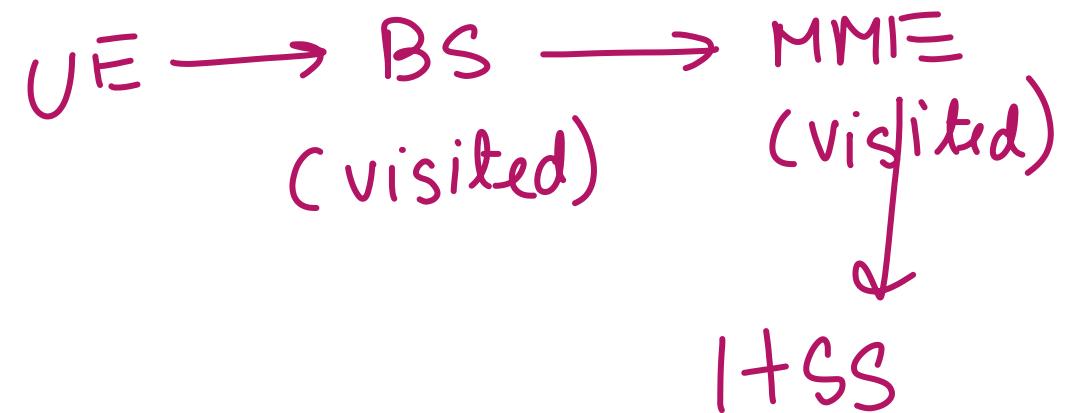
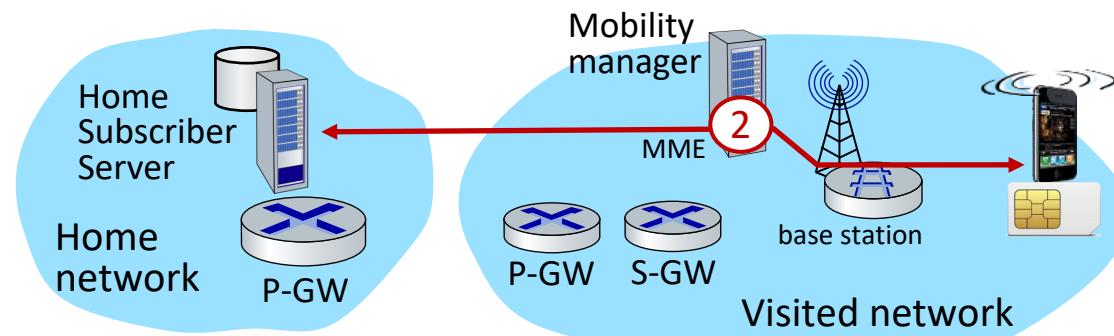
✗  
③ **data-plane configuration:**

- MME configures forwarding tunnels for mobile
- visited, home network establish tunnels from home P-GW to mobile

✗  
④ **mobile handover:**

- mobile device changes its point of attachment to visited network

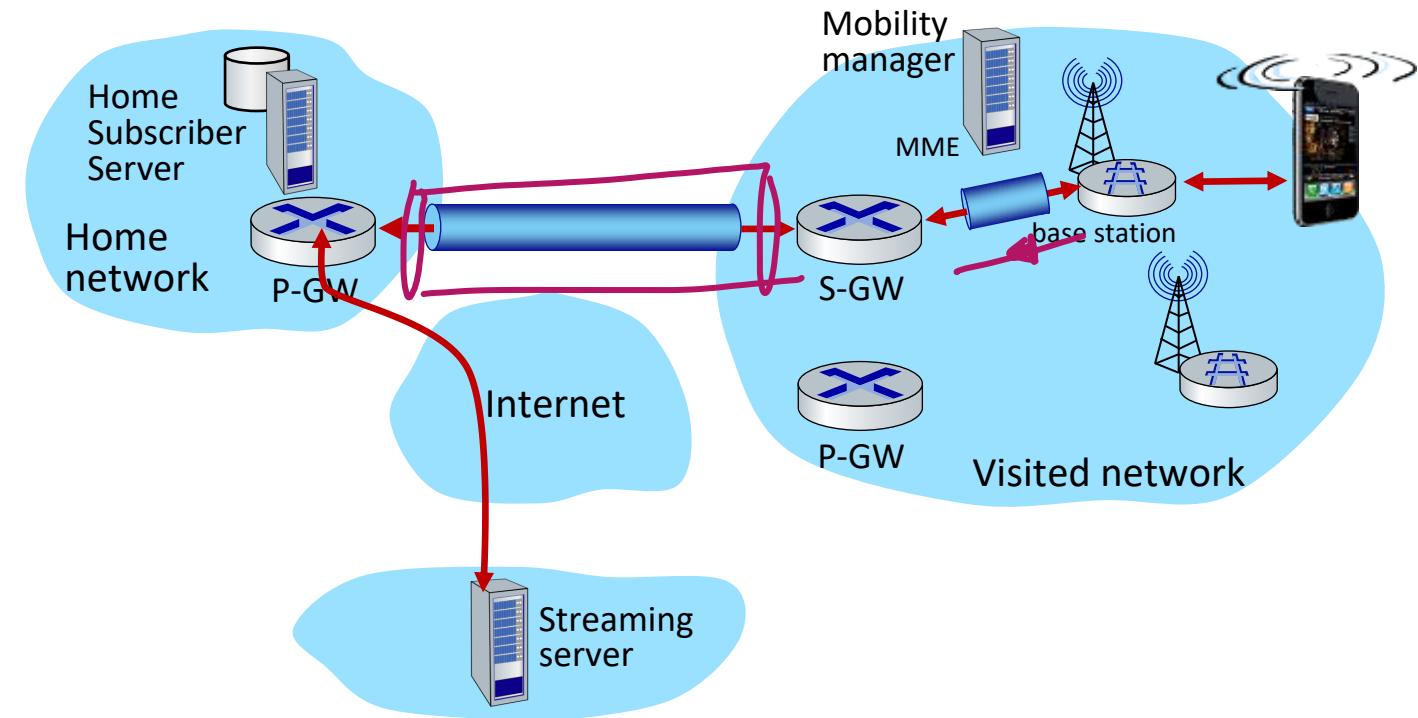
# Configuring LTE control-plane elements



- ✓ Mobile communicates with local MME via BS control-plane channel
- ✓ MME uses mobile's IMSI info to contact mobile's home HSS
  - retrieve authentication, encryption, network service information
  - home HSS knows mobile now resident in visited network
- BS, mobile select parameters for BS-mobile data-plane radio channel

# Configuring data-plane tunnels for mobile

- S-GW to BS tunnel: when mobile changes base stations, simply change endpoint IP address of tunnel
- S-GW to home P-GW tunnel: implementation of indirect routing
- tunneling via GTP (GPRS tunneling protocol): mobile's datagram to streaming server encapsulated using GTP inside UDP, inside datagram

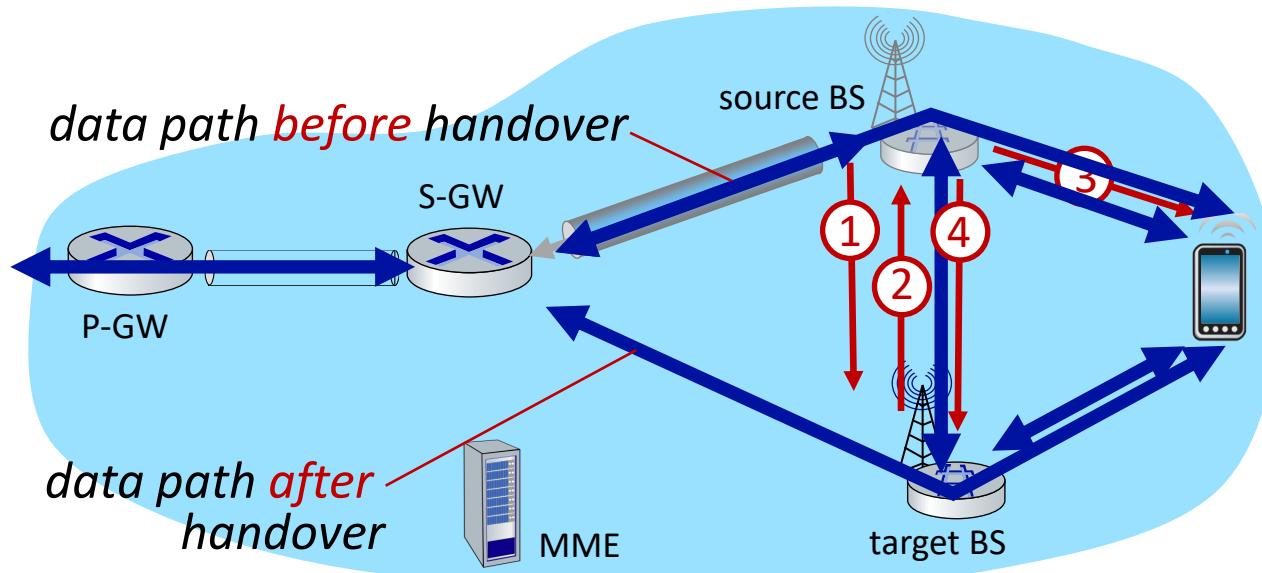


UE — BS  
c visited

S - Gw  
(visited)

P - Gw

# Handover between BSs in same cellular network



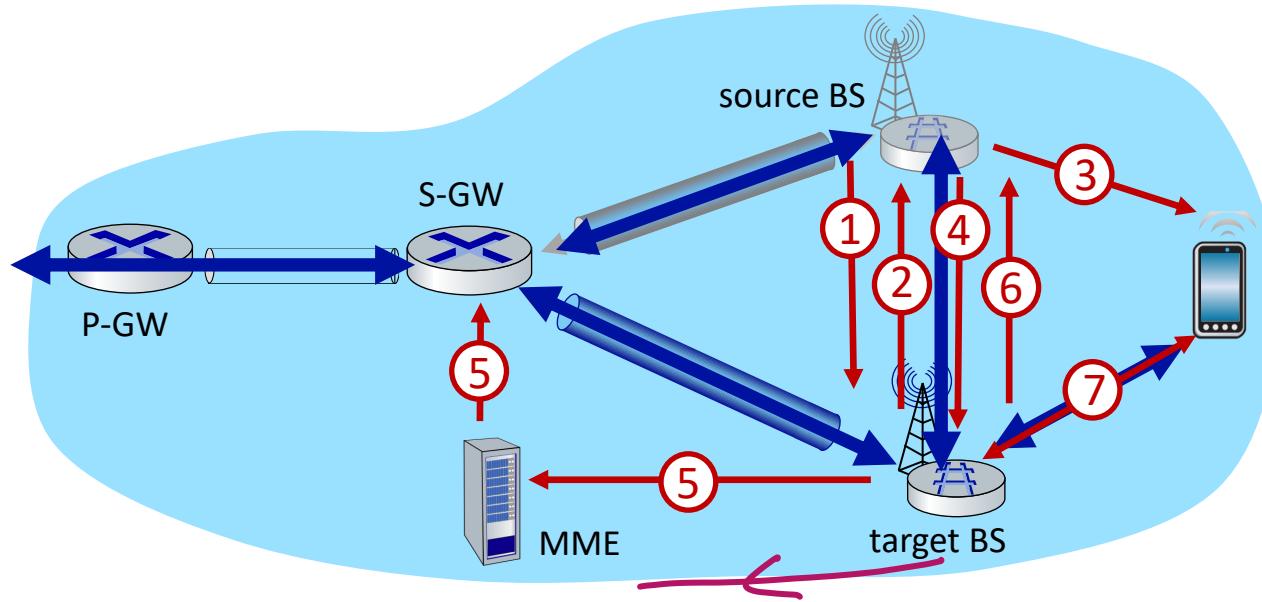
① current (source) BS selects target BS, sends Handover Request message to target BS

② target BS pre-allocates radio time slots, responds with HR ACK with info for mobile

③ source BS informs mobile of new BS  
▪ mobile can now send via new BS - handover looks complete to mobile

④ source BS stops sending datagrams to mobile, instead forwards to new BS (who forwards to mobile over radio channel)

# Handover between BSs in same cellular network



⑤ target BS informs MME that it is new BS for mobile

- MME instructs S-GW to change tunnel endpoint to be (new) target BS

⑥ target BS ACKs back to source BS: handover complete, source BS can release resources

⑦ mobile's datagrams now flow through new tunnel from target BS to S-GW

# Wireless, mobility: impact on higher layer protocols

- logically, impact *should* be minimal ...
  - best effort service model remains unchanged
  - TCP and UDP can (and do) run over wireless, mobile
- ... but performance-wise:
  - packet loss/delay due to bit-errors (discarded packets, delays for link-layer retransmissions), and handover loss
  - TCP interprets loss as congestion, will decrease congestion window unnecessarily
  - delay impairments for real-time traffic
  - bandwidth a scarce resource for wireless links

# Network Security

Slides are adapted from the companion website of the book *Computer Networking: A Top-Down Approach* by Jim Kurose, Keith Ross

# What is network security?

**confidentiality:** only sender, intended receiver should “understand” message contents

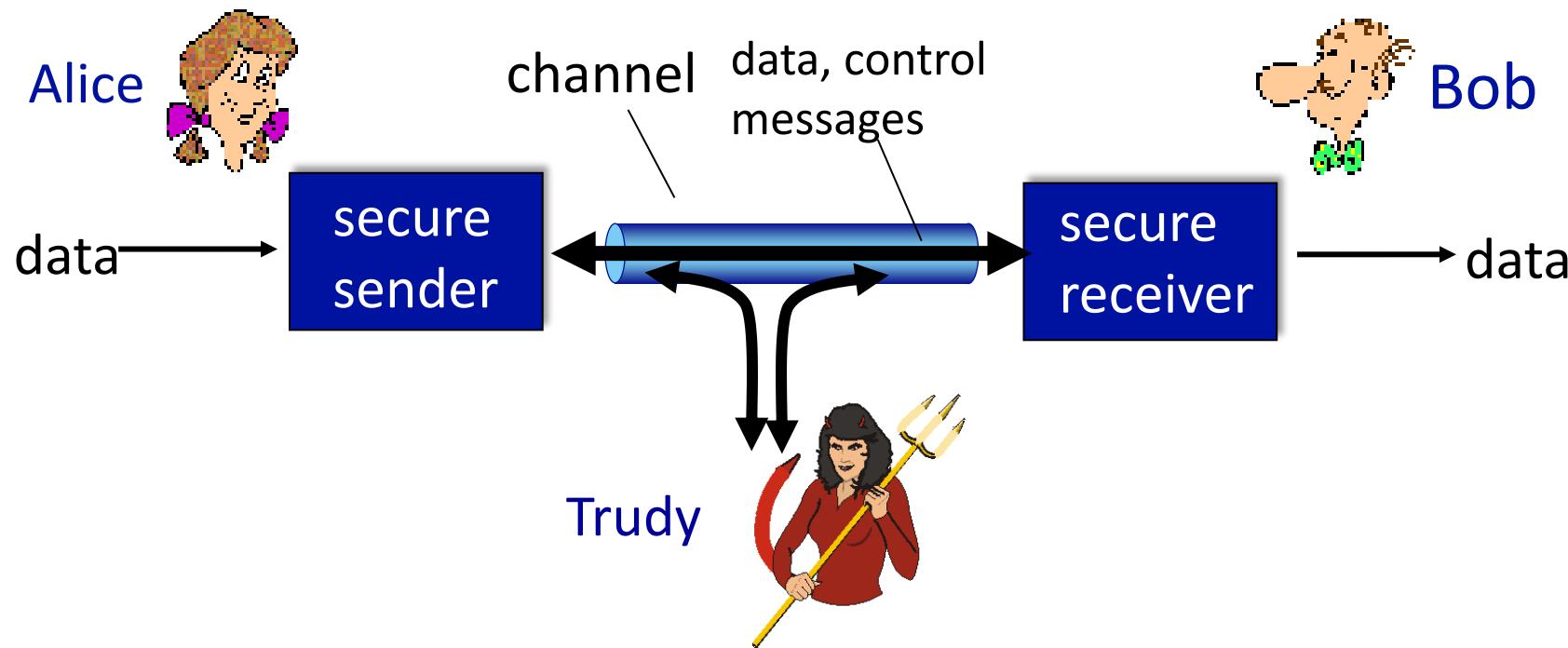
- sender encrypts message
- receiver decrypts message

**authentication:** sender, receiver want to confirm identity of each other

**message integrity:** sender, receiver want to ensure message not altered (in transit, or afterwards) without detection

# Friends and enemies: Alice, Bob, Trudy

- well-known in network security world
- Bob, Alice want to communicate “securely”
- Trudy (intruder) may intercept, delete, add messages



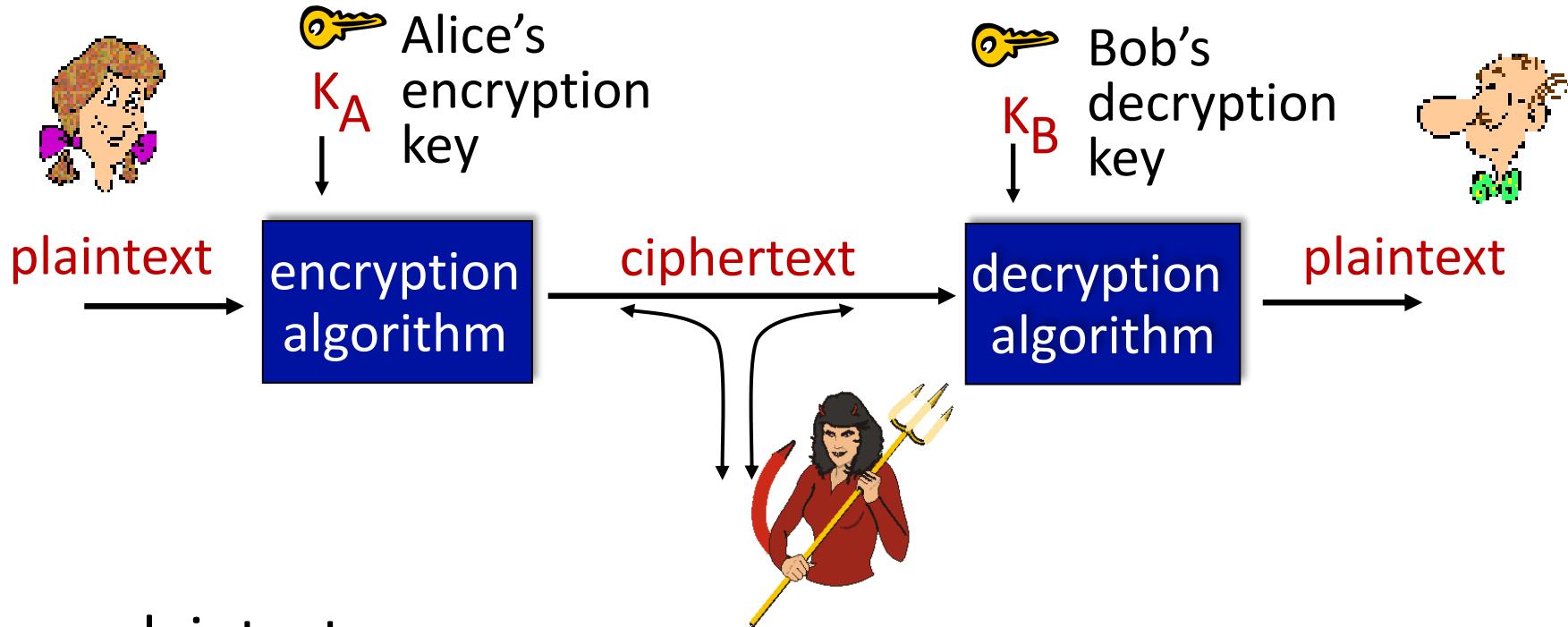
# Bad activities

- **eavesdrop:** intercept messages
- actively **insert** messages into connection
- **impersonation:** can fake (spoof) source address in packet (or any field in packet)
- **hijacking:** “take over” ongoing connection by removing sender or receiver, inserting himself in place
- **denial of service:** prevent service from being used by others (e.g., by overloading resources)

# Outline

- What is network security?
- **Principles of cryptography**
- Message integrity, authentication
- Securing e-mail
- Securing TCP connections: TLS
- Operational security: firewalls

# The language of cryptography



$m$ : plaintext message

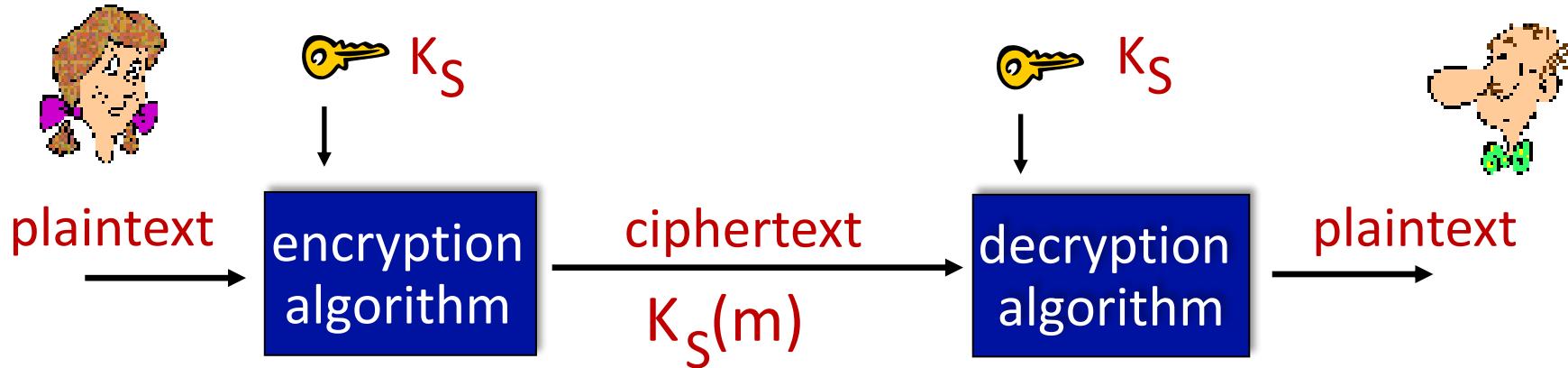
$K_A(m)$ : ciphertext, encrypted with key  $K_A$

$m = K_B(K_A(m))$

# Breaking an encryption scheme

- **cipher-text only attack:**  
Trudy has ciphertext she can analyze
- **two approaches:**
  - brute force: search through all keys
  - statistical analysis
- **known-plaintext attack:**  
Trudy has plaintext corresponding to ciphertext
  - e.g., in monoalphabetic cipher, Trudy determines pairings for a,l,i,c,e,b,o,
- **chosen-plaintext attack:**  
Trudy can get ciphertext for chosen plaintext

# Symmetric key cryptography



**symmetric key crypto:** Bob and Alice share same (symmetric) key:  $K$

- e.g., key is knowing substitution pattern in mono alphabetic substitution cipher

Q: how do Bob and Alice agree on key value?

# Simple encryption scheme

*substitution cipher:* substituting one thing for another

- monoalphabetic cipher: substitute one letter for another

plaintext: abcdefghijklmnopqrstuvwxyz

ciphertext: mnbvctxzasdfghjklpoiuytrewq

e.g.: Plaintext: bob. i love you. alice

ciphertext: nkn. s gktc wky. mgsbc



*Encryption key:* mapping from set of 26 letters

to set of 26 letters

# A more sophisticated encryption approach

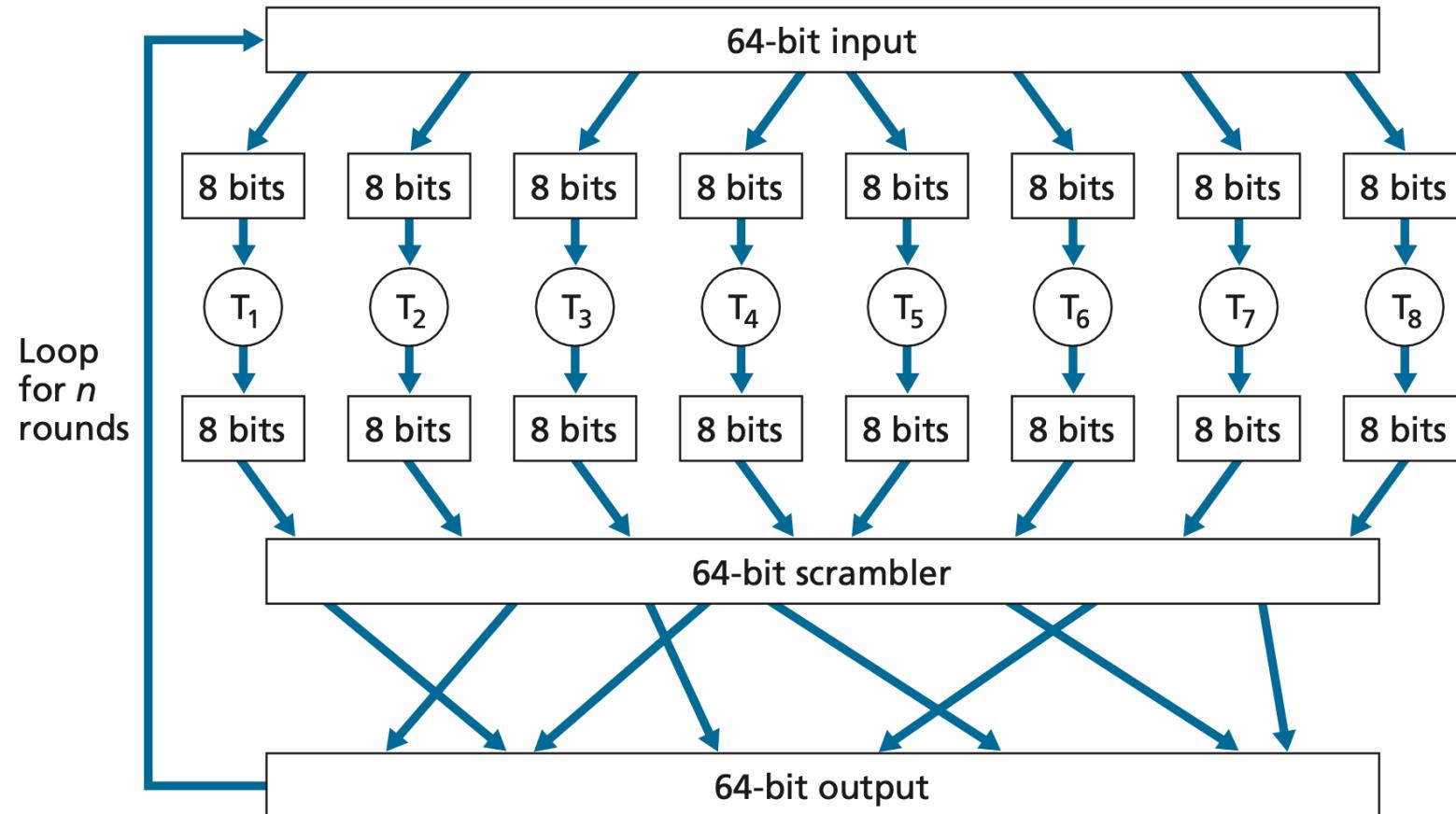
- n substitution ciphers,  $M_1, M_2, \dots, M_n$
  - cycling pattern:
    - e.g., n=4:  $M_1, M_3, M_4, M_3, M_2; M_1, M_3, M_4, M_3, M_2; \dots$
  - for each new plaintext symbol, use subsequent substitution pattern in cyclic pattern
    - dog: d from  $M_1$ , o from  $M_3$ , g from  $M_4$
-  *Encryption key:* n substitution ciphers, and cyclic pattern

# A simple block cipher

| input | output | input | output |
|-------|--------|-------|--------|
| 000   | 110    | 100   | 011    |
| 001   | 111    | 101   | 010    |
| 010   | 101    | 110   | 000    |
| 011   | 100    | 111   | 001    |

- The message to be encrypted is processed in blocks of k bits and each block is encrypted independently.
- To encode a block, the cipher uses a one-to-one mapping to map the k-bit block of cleartext to a k-bit block of ciphertext.

# Practical block ciphers



In practice, functions are used rather than predetermined tables

# Symmetric key crypto: DES

## DES: Data Encryption Standard

- US encryption standard [NIST 1993]
- 56-bit symmetric key, 64-bit plaintext input
- block cipher with cipher block chaining
- how secure is DES?
  - DES Challenge: 56-bit-key-encrypted phrase decrypted (brute force) in less than a day
  - no known good analytic attack
- making DES more secure:
  - 3DES: encrypt 3 times with 3 different keys

# AES: Advanced Encryption Standard

- symmetric-key NIST standard, replaced DES (Nov 2001)
- processes data in 128 bit blocks
- 128, 192, or 256 bit keys
- brute force decryption (try each key) taking 1 sec on DES, takes 149 trillion years for AES

# Public Key Cryptography

## symmetric key crypto:

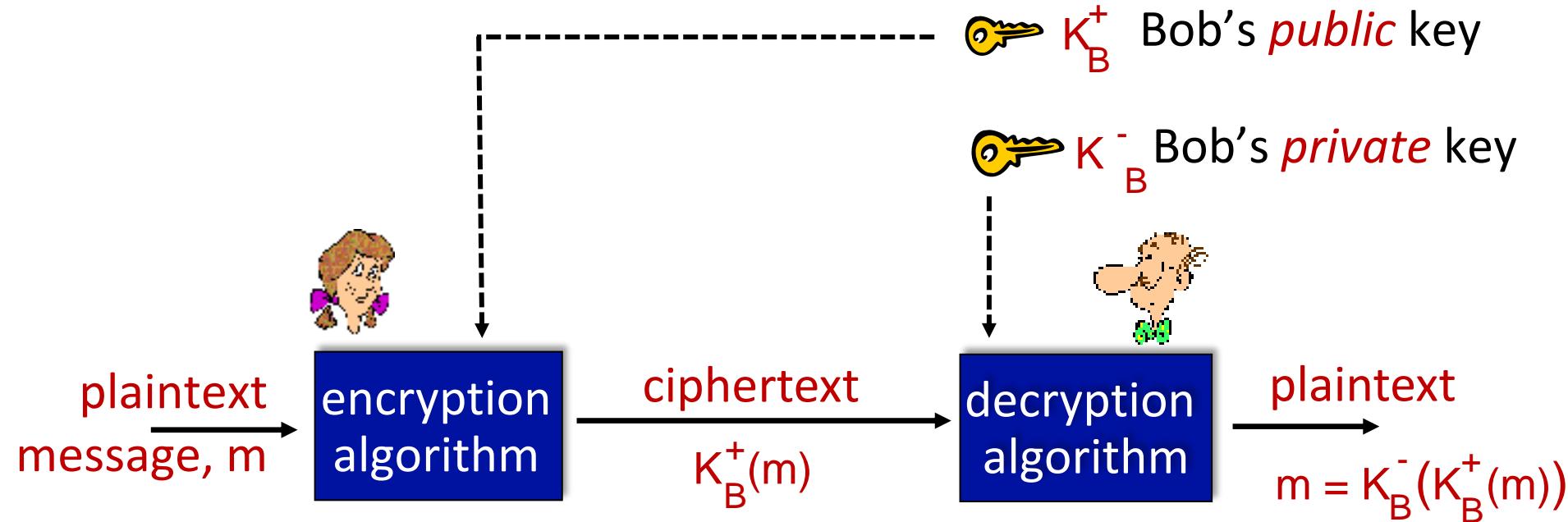
- requires sender, receiver know shared secret key
- Q: how to agree on key in first place (particularly if never “met”)?

## public key crypto

- *radically* different approach [Diffie-Hellman76, RSA78]
- sender, receiver do *not* share secret key
- **public** encryption key known to *all*
- **private** decryption key known only to receiver



# Public Key Cryptography



**Wow** - public key cryptography revolutionized 2000-year-old (previously only symmetric key) cryptography!

- similar ideas emerged at roughly same time, independently in US and UK (classified)

# Public key encryption algorithms

requirements:

- ① need  $K_B^+(\cdot)$  and  $K_B^-(\cdot)$  such that

$$K_B^-(K_B^+(m)) = m$$

- ② given public key  $K_B^+$ , it should be impossible to compute private key  $K_B^-$

**RSA:** Rivest, Shamir, Adelson algorithm

# Prerequisite: modular arithmetic

- $x \bmod n$  = remainder of  $x$  when divide by  $n$

- facts:

$$[(a \bmod n) + (b \bmod n)] \bmod n = (a+b) \bmod n$$

$$[(a \bmod n) - (b \bmod n)] \bmod n = (a-b) \bmod n$$

$$[(a \bmod n) * (b \bmod n)] \bmod n = (a*b) \bmod n$$

- thus

$$(a \bmod n)^d \bmod n = a^d \bmod n$$

- example:  $x=14$ ,  $n=10$ ,  $d=2$ :

$$(x \bmod n)^d \bmod n = 4^2 \bmod 10 = 6$$

$$x^d = 14^2 = 196 \quad x^d \bmod 10 = 6$$

# RSA: getting ready

- message: just a bit pattern
- bit pattern can be uniquely represented by an integer number
- thus, encrypting a message is equivalent to encrypting a number

example:

- $m = 10010001$ . This message is uniquely represented by the decimal number 145.
- to encrypt  $m$ , we encrypt the corresponding number, which gives a new number (the ciphertext).

# RSA: Creating public/private key pair

1. choose two large prime numbers  $p, q$ . (e.g., 1024 bits each)
2. compute  $n = pq, z = (p-1)(q-1)$
3. choose  $e$  (with  $e < n$ ) that has no common factors with  $z$   
( $e, z$  are “relatively prime”).
4. choose  $d$  such that  $ed-1$  is exactly divisible by  $z$ .  
(in other words:  $ed \bmod z = 1$  ).
5. *public* key is  $\underbrace{(n,e)}_{K_B^+}$ . *private* key is  $\underbrace{(n,d)}_{K_B^-}$ .

# RSA: encryption, decryption

0. given  $(n, e)$  and  $(n, d)$  as computed above
1. to encrypt message  $m (< n)$ , compute

$$c = m^e \text{ mod } n$$

2. to decrypt received bit pattern,  $c$ , compute

$$m = c^d \text{ mod } n$$

magic happens!

$$m = (\underbrace{m^e \text{ mod } n}_c)^d \text{ mod } n$$

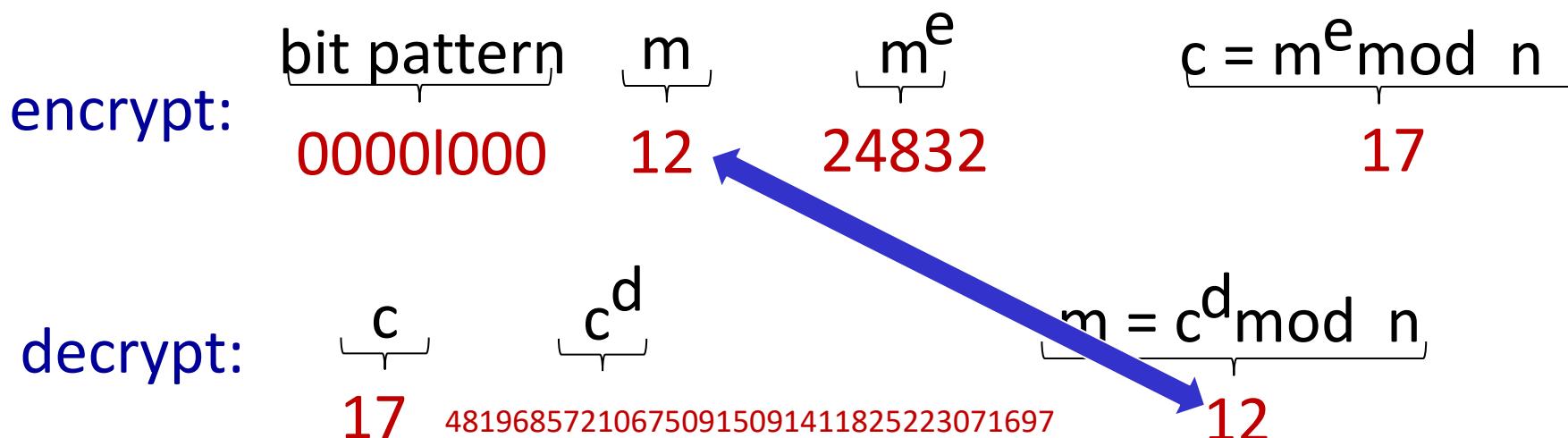
# RSA example:

Bob chooses  $p=5$ ,  $q=7$ . Then  $n=35$ ,  $z=24$ .

$e=5$  (so  $e, z$  relatively prime).

$d=29$  (so  $ed-1$  exactly divisible by  $z$ ).

encrypting 8-bit messages.



# Why does RSA work?

- must show that  $c^d \bmod n = m$ , where  $c = m^e \bmod n$
- A remarkable result from Number Theory:  
for any  $x$  and  $y$ :  $x^y \bmod n = x^{(y \bmod z)} \bmod n$ 
  - where  $n = pq$  and  $z = (p-1)(q-1)$

- thus,  
$$\begin{aligned} c^d \bmod n &= (m^e \bmod n)^d \bmod n \\ &= m^{ed} \bmod n \\ &= m^{(ed \bmod z)} \bmod n \\ &= m^1 \bmod n \\ &= m \end{aligned}$$

# RSA: another important property

The following property is *very useful* :

$$\underbrace{K_B^-(K_B^+(m))}_{= m} = K_B^+(K_B^-(m))$$

use public key  
first, followed  
by private key

use private key  
first, followed  
by public key

*result is the same!*

Why  $K_B^-(K_B^+(m)) = m = K_B^+(K_B^-(m))$  ?

follows directly from modular arithmetic:

$$\begin{aligned}(m^e \bmod n)^d \bmod n &= m^{ed} \bmod n \\&= m^{de} \bmod n \\&= (m^d \bmod n)^e \bmod n\end{aligned}$$

# Why is RSA secure?

- suppose you know Bob's public key  $(n, e)$ . How hard is it to determine  $d$ ?
- essentially need to find factors of  $n$  without knowing the two factors  $p$  and  $q$ 
  - fact: factoring a big number is hard

# RSA in practice: session keys

- exponentiation in RSA is computationally intensive
- DES is at least 100 times faster than RSA
- use public key crypto to establish secure connection, then establish second key – symmetric session key – for encrypting data

## session key, $K_s$

- Bob and Alice use RSA to exchange a symmetric session key  $K_s$
- once both have  $K_s$ , they use symmetric key cryptography

# Outline

- What is network security?
- Principles of cryptography
- **Authentication**, message integrity
- Securing e-mail
- Securing TCP connections: TLS
- Operational security: firewalls

# Authentication

**Goal:** Bob wants Alice to “prove” her identity to him

**Protocol ap1.0:** Alice says “I am Alice”



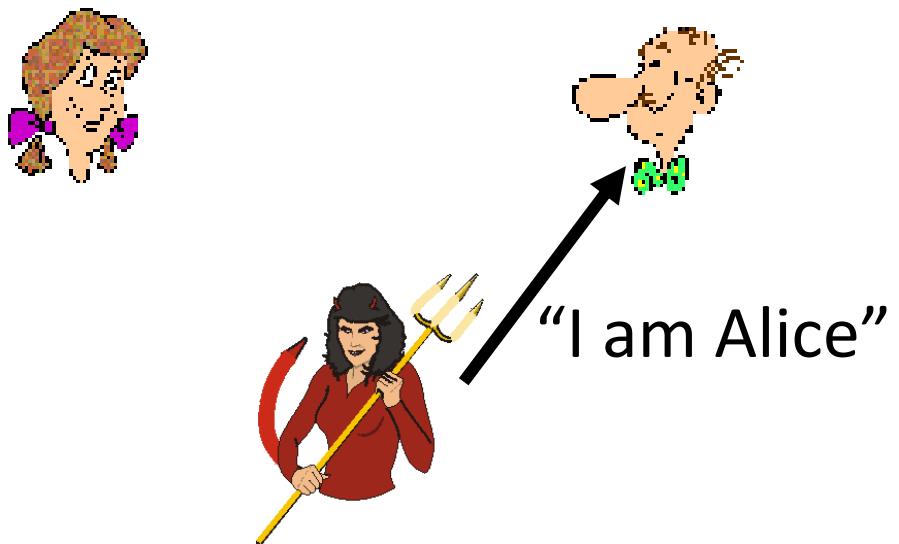
*failure scenario??*



# Authentication

**Goal:** Bob wants Alice to “prove” her identity to him

**Protocol ap1.0:** Alice says “I am Alice”



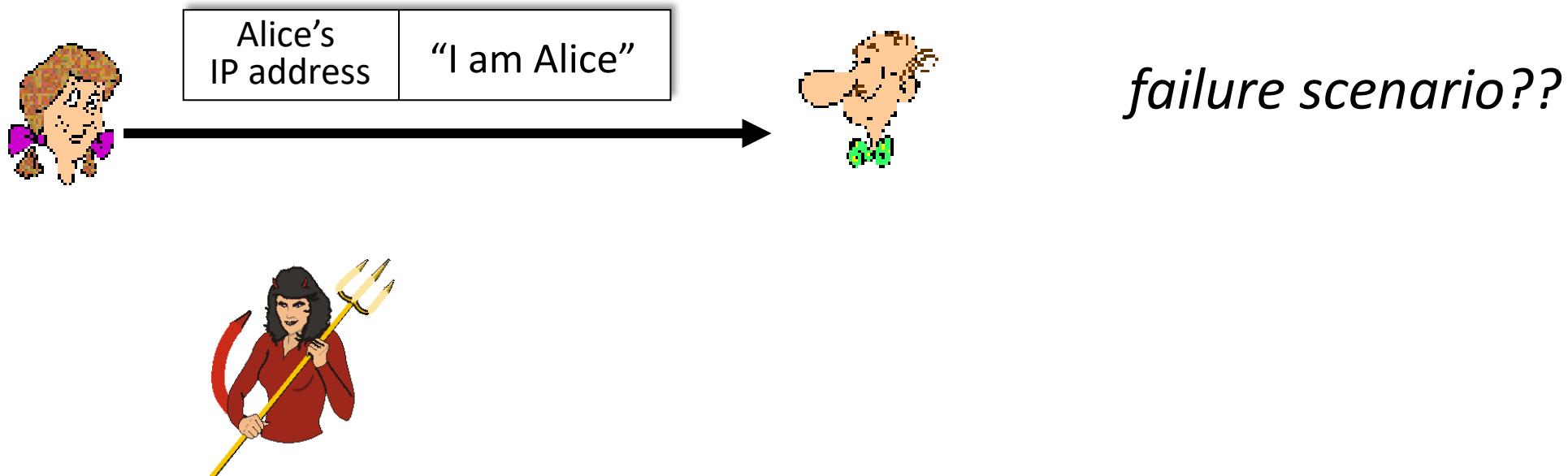
*in a network, Bob can not “see” Alice, so Trudy simply declares herself to be Alice*



# Authentication: another try

**Goal:** Bob wants Alice to “prove” her identity to him

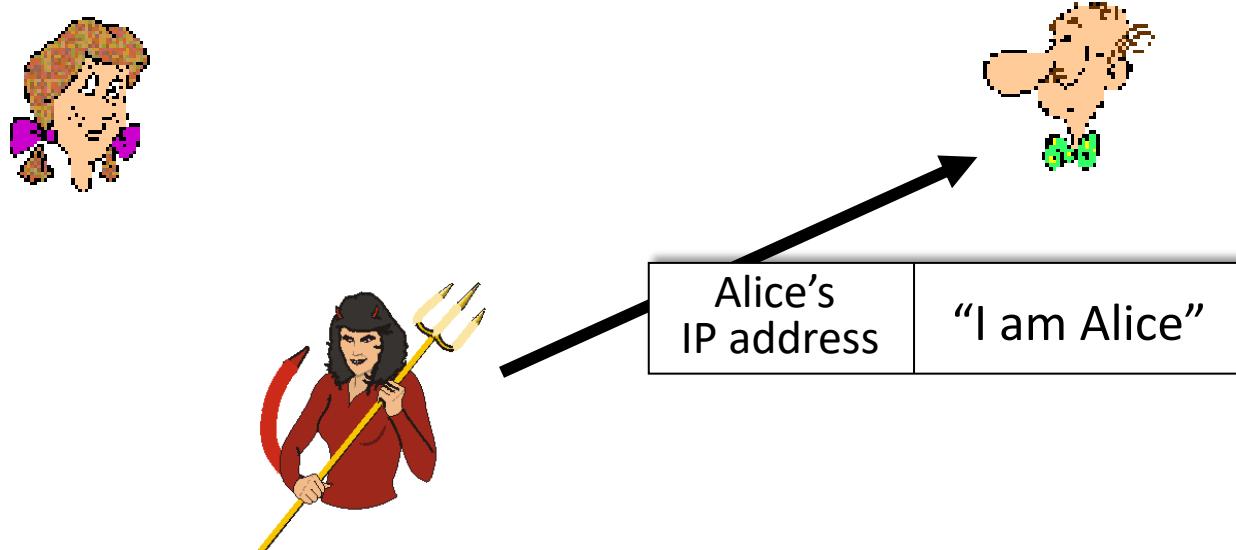
**Protocol ap2.0:** Alice says “I am Alice” in an IP packet containing her source IP address



# Authentication: another try

**Goal:** Bob wants Alice to “prove” her identity to him

**Protocol ap2.0:** Alice says “I am Alice” in an IP packet containing her source IP address

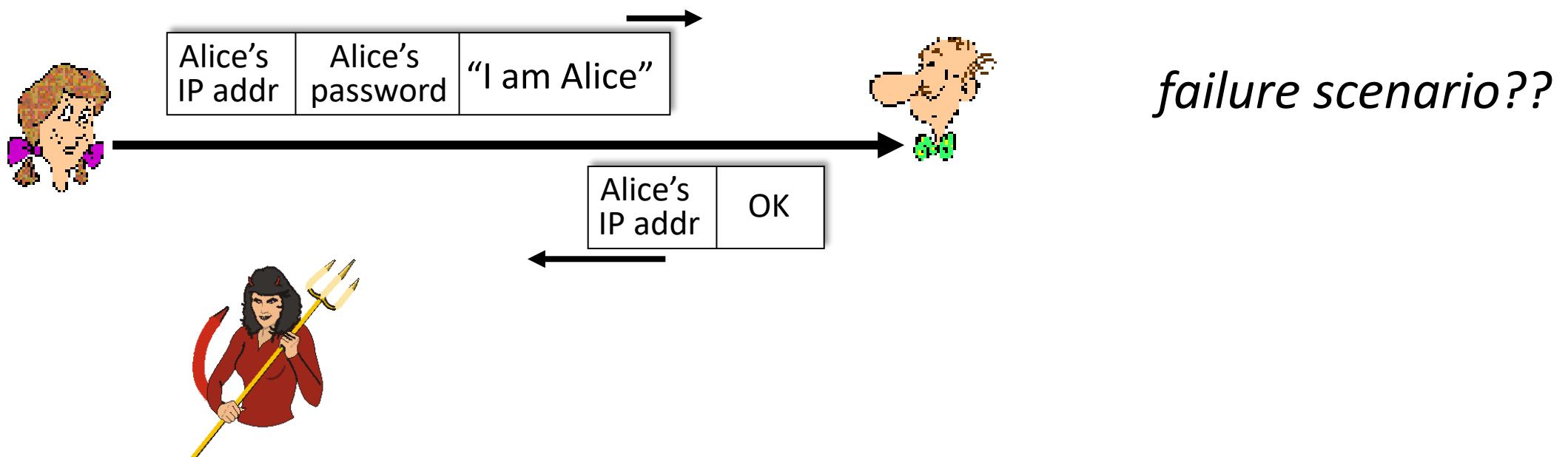


*Trudy can create  
a packet “spoofing”  
Alice’s address*

# Authentication: a third try

**Goal:** Bob wants Alice to “prove” her identity to him

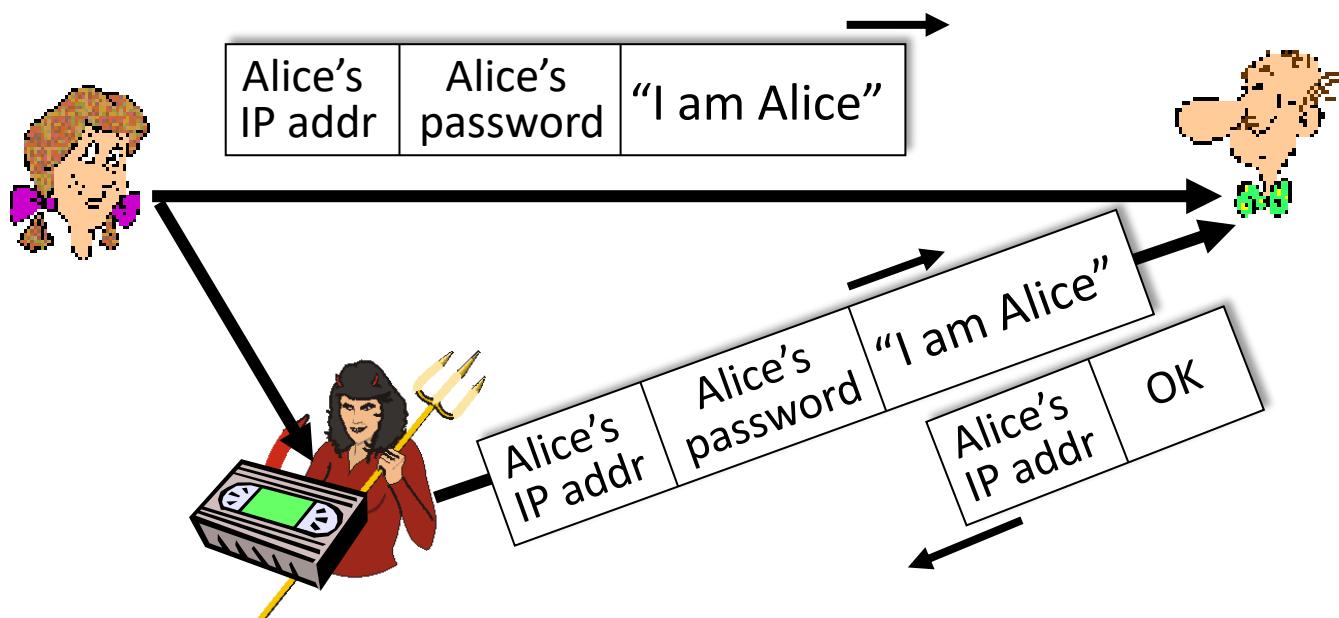
**Protocol ap3.0:** Alice says “I am Alice” and sends her secret password to “prove” it.



# Authentication: a third try

**Goal:** Bob wants Alice to “prove” her identity to him

**Protocol ap3.0:** Alice says “I am Alice” and sends her secret password to “prove” it.

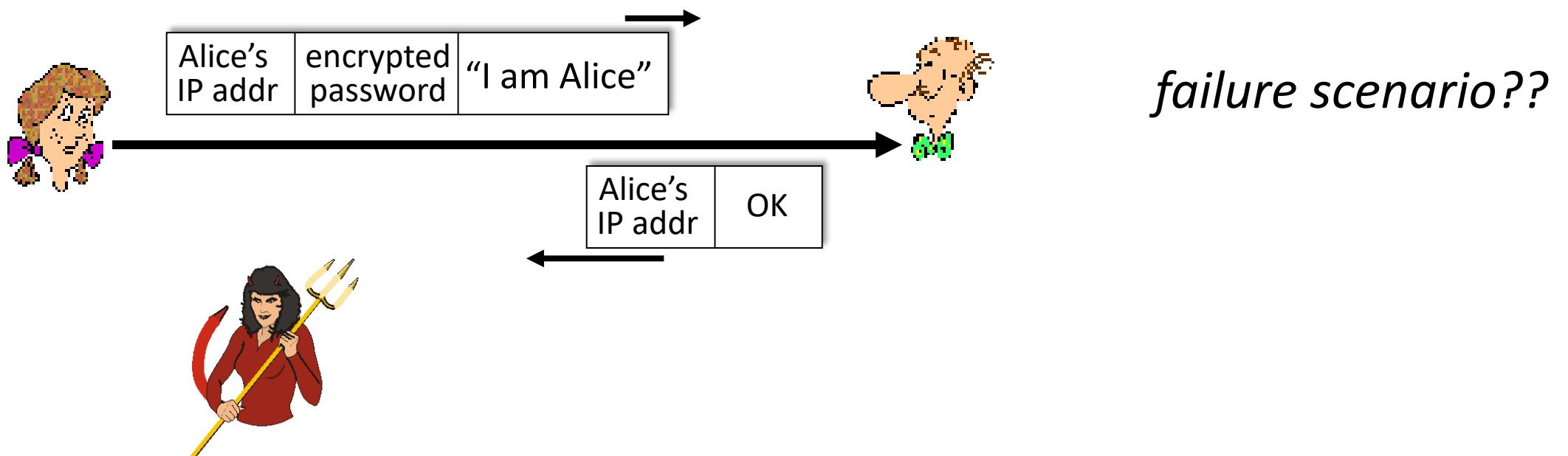


*playback attack:  
Trudy records  
Alice's packet  
and later  
plays it back to Bob*

# Authentication: a modified third try

**Goal:** Bob wants Alice to “prove” her identity to him

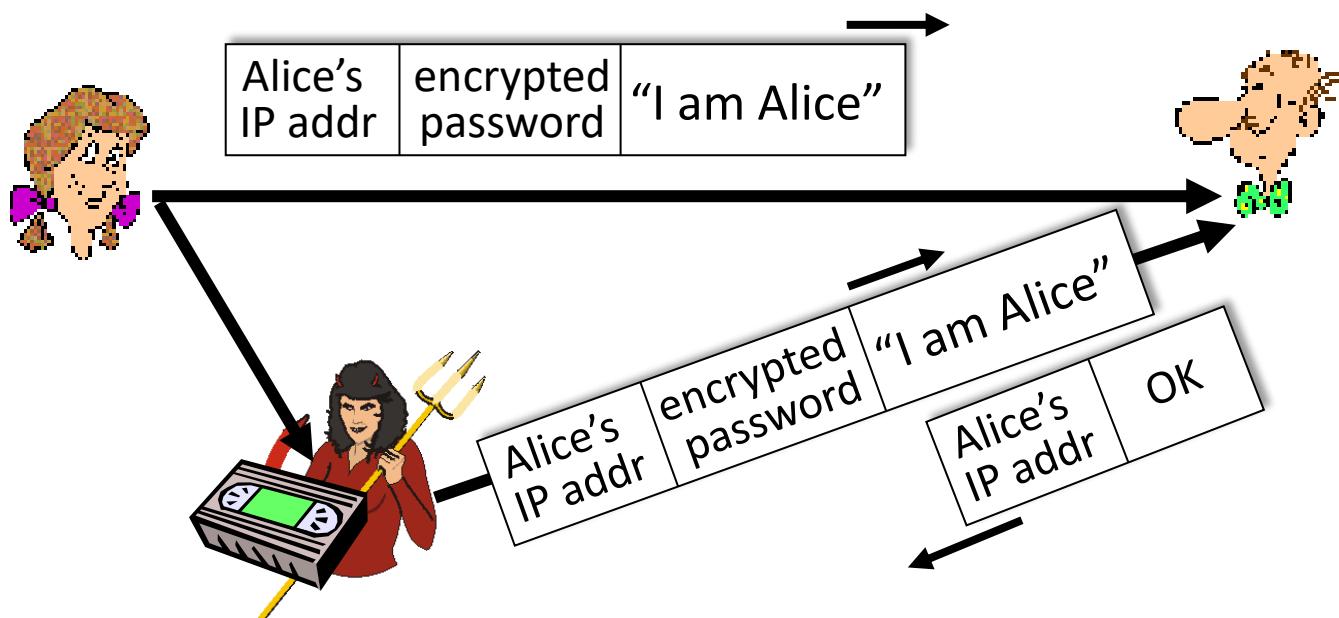
**Protocol ap3.0:** Alice says “I am Alice” and sends her encrypted secret password to “prove” it.



# Authentication: a modified third try

**Goal:** Bob wants Alice to “prove” her identity to him

**Protocol ap3.0:** Alice says “I am Alice” and sends her encrypted secret password to “prove” it.



*playback attack still works: Trudy records Alice's packet and later plays it back to Bob*

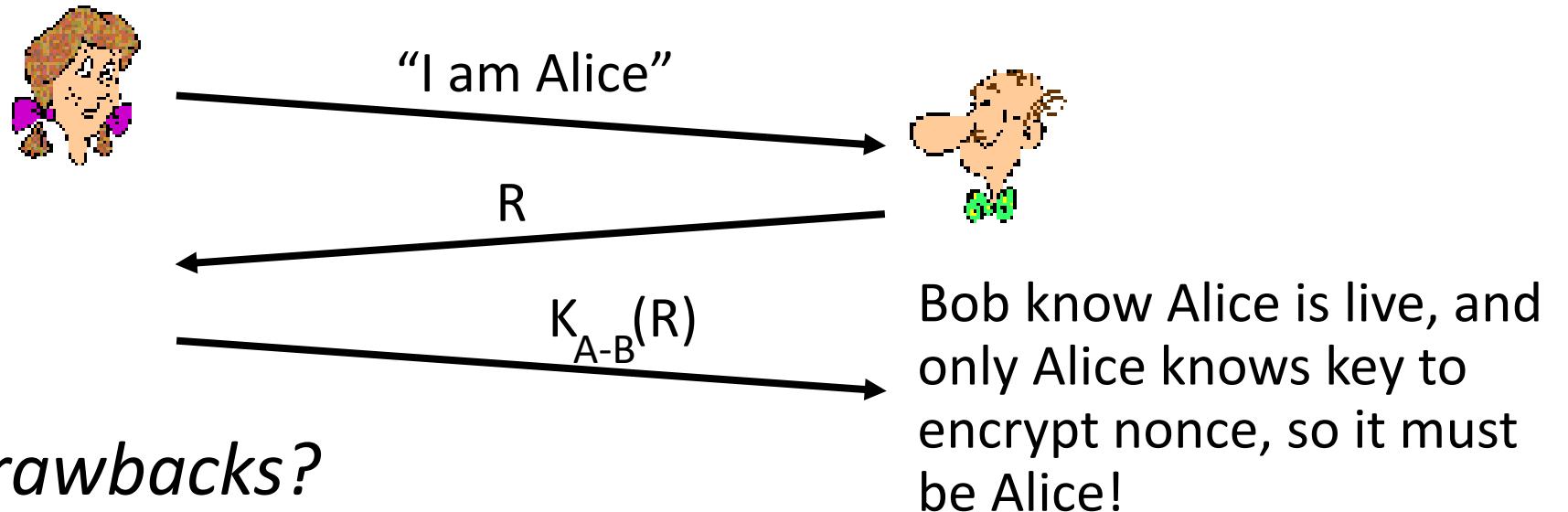
# Authentication: a fourth try

**Goal:** avoid playback attack

**nonce:** number (R) used only **once-in-a-lifetime**

**protocol ap4.0:** to prove Alice “live”, Bob sends Alice nonce, R

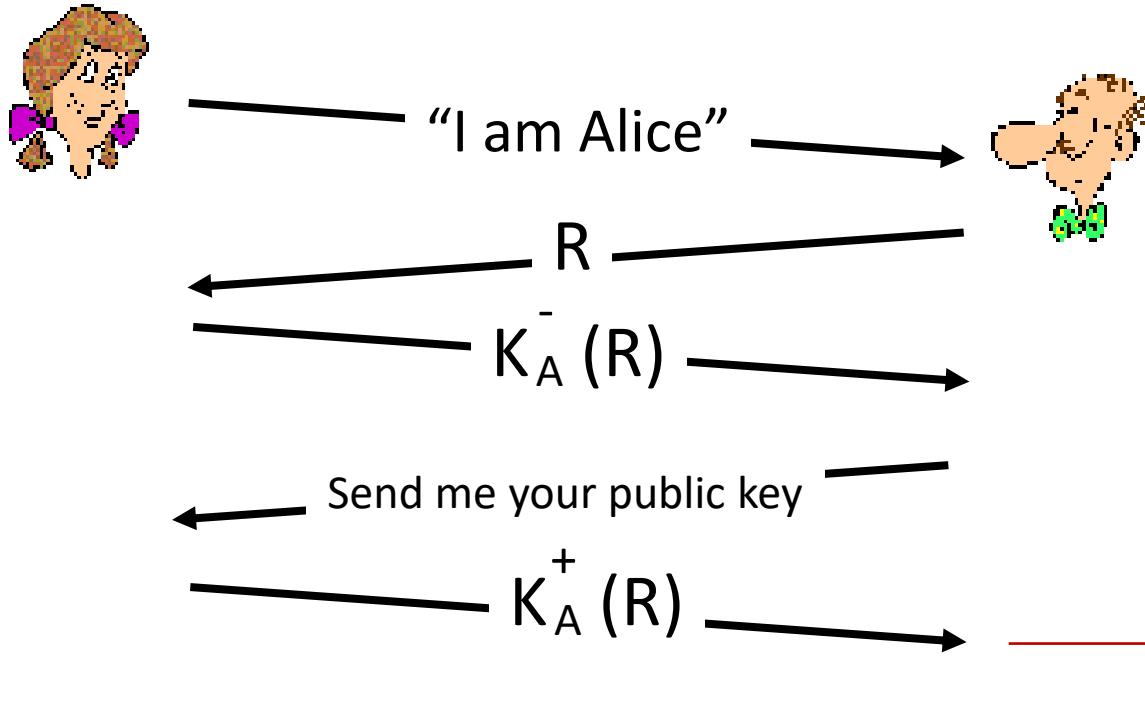
- Alice must return R, encrypted with shared secret key



# Authentication: ap5.0

ap4.0 requires shared symmetric key - can we authenticate using public key techniques?

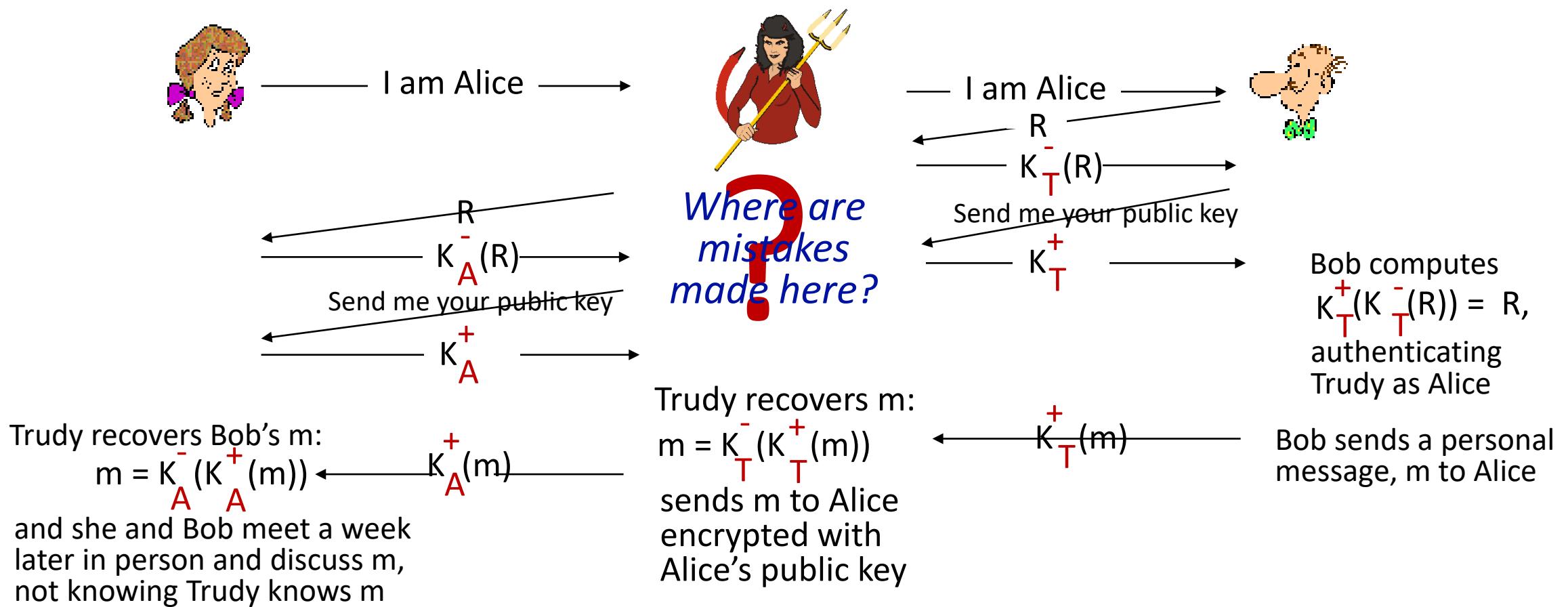
ap5.0: use nonce, public key cryptography



Bob computes  
 $K_A^+ (K_A^-(R)) = R$   
and knows only Alice could have the private key, that encrypted  $R$  such that  
 $K_A^+ (K_A^-(R)) = R$

# Authentication: ap5.0 – there's still a flaw!

man (or woman) in the middle attack: Trudy poses as Alice (to Bob) and as Bob (to Alice)



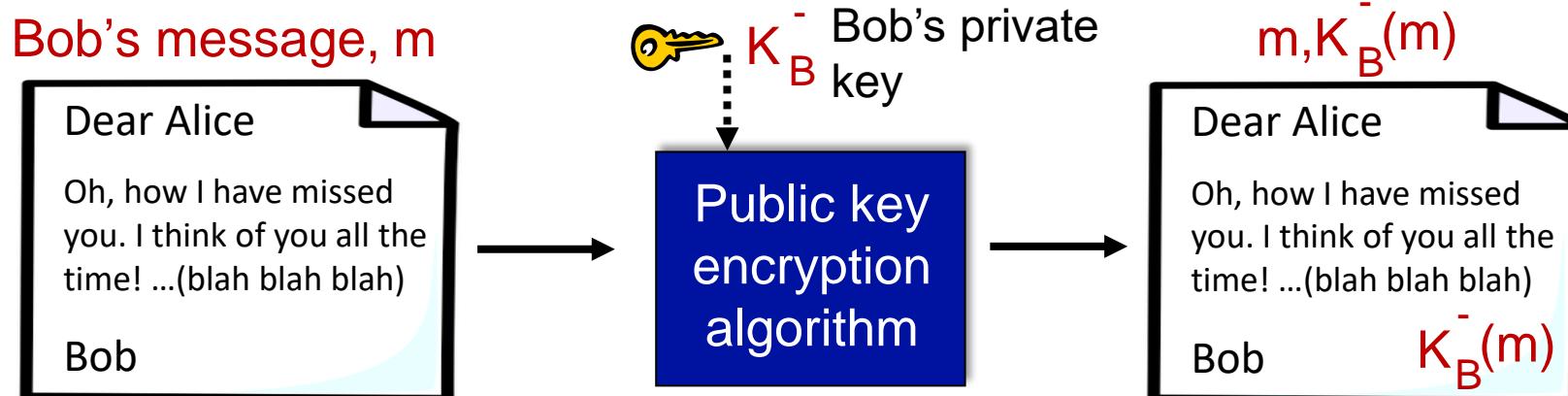
# Outline

- What is network security?
- Principles of cryptography
- Authentication, **message integrity**
- Securing e-mail
- Securing TCP connections: TLS
- Operational security: firewalls

# Digital signatures

cryptographic technique analogous to hand-written signatures:

- sender (Bob) digitally signs document: he is document owner/creator.
- *verifiable, nonforgeable*: recipient (Alice) can prove to someone that Bob, and no one else (including Alice), must have signed document
- simple digital signature for message  $m$ :
  - Bob signs  $m$  by encrypting with his private key  $K_B^-$ , creating “signed” message,  $K_B^-(m)$



# Digital signatures

- suppose Alice receives msg  $m$ , with signature:  $m, \bar{K}_B(m)$
- Alice verifies  $m$  signed by Bob by applying Bob's public key  $\bar{K}_B$  to  $\bar{K}_B(m)$  then checks  $\bar{K}_B(\bar{K}_B(m)) = m$ .
- If  $\bar{K}_B(\bar{K}_B(m)) = m$ , whoever signed  $m$  must have used Bob's private key

Alice thus verifies that:

- Bob signed  $m$
- no one else signed  $m$
- Bob signed  $m$  and not  $m'$

non-repudiation:

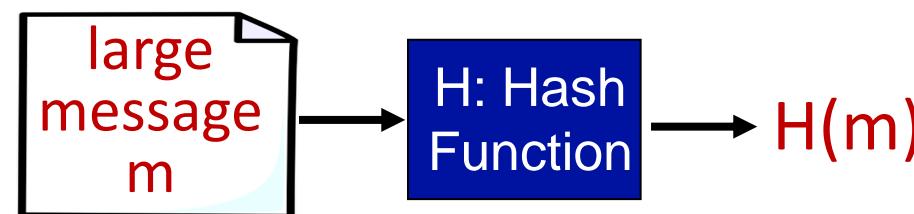
- ✓ Alice can take  $m$ , and signature  $\bar{K}_B(m)$  to court and prove that Bob signed  $m$

# Message digests

computationally expensive to public-key-encrypt long messages

**goal:** fixed-length, easy- to-compute digital “fingerprint”

- apply hash function  $H$  to  $m$ , get fixed size message digest,  $H(m)$



**Hash function properties:**

- many-to-1
- produces fixed-size msg digest (fingerprint)
- given message digest  $x$ , computationally infeasible to find  $m$  such that  $x = H(m)$

# Internet checksum: poor crypto hash function

Internet checksum has some properties of hash function:

- produces fixed length digest (16-bit sum) of message
- is many-to-one

but given message with given hash value, it is easy to find another message with same hash value:

| <u>message</u> | <u>ASCII format</u> |
|----------------|---------------------|
| I O U 1        | 49 4F 55 31         |
| 0 0 . 9        | 30 30 2E 39         |
| 9 B O B        | 39 42 D2 42         |

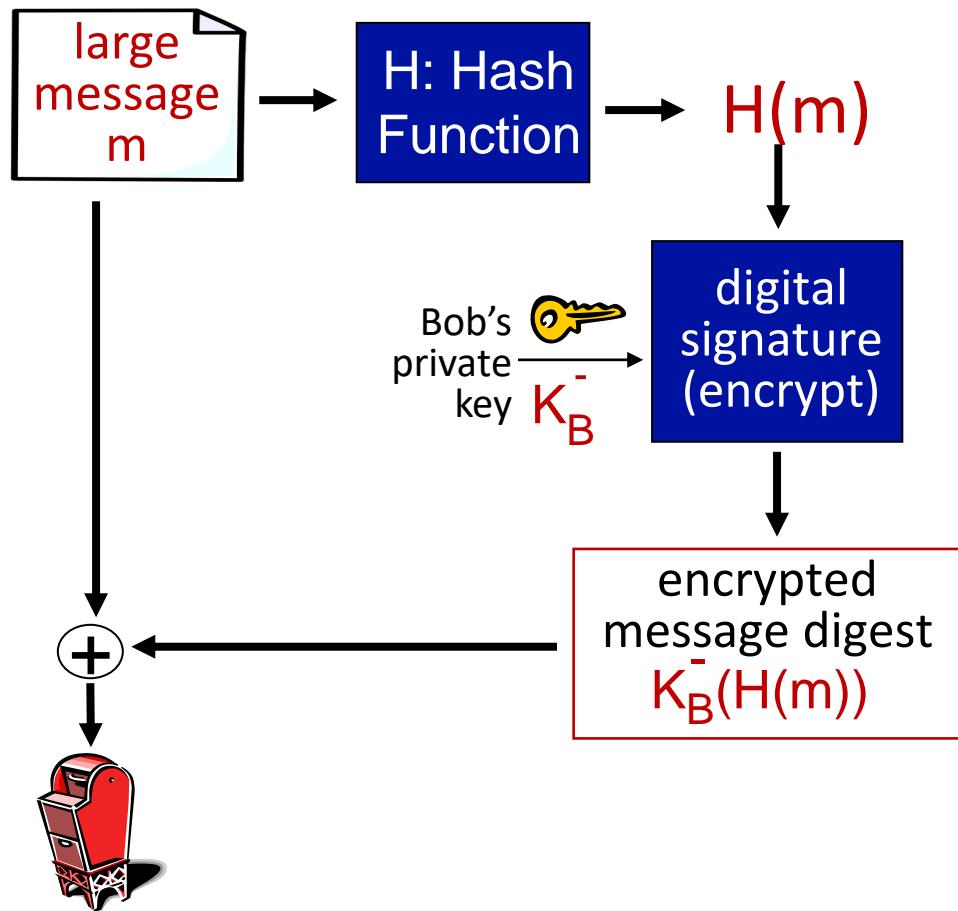
| <u>message</u> | <u>ASCII format</u> |
|----------------|---------------------|
| I O U 9        | 49 4F 55 <u>39</u>  |
| 0 0 . 1        | 30 30 2E <u>31</u>  |
| 9 B O B        | 39 42 D2 42         |

*different messages*

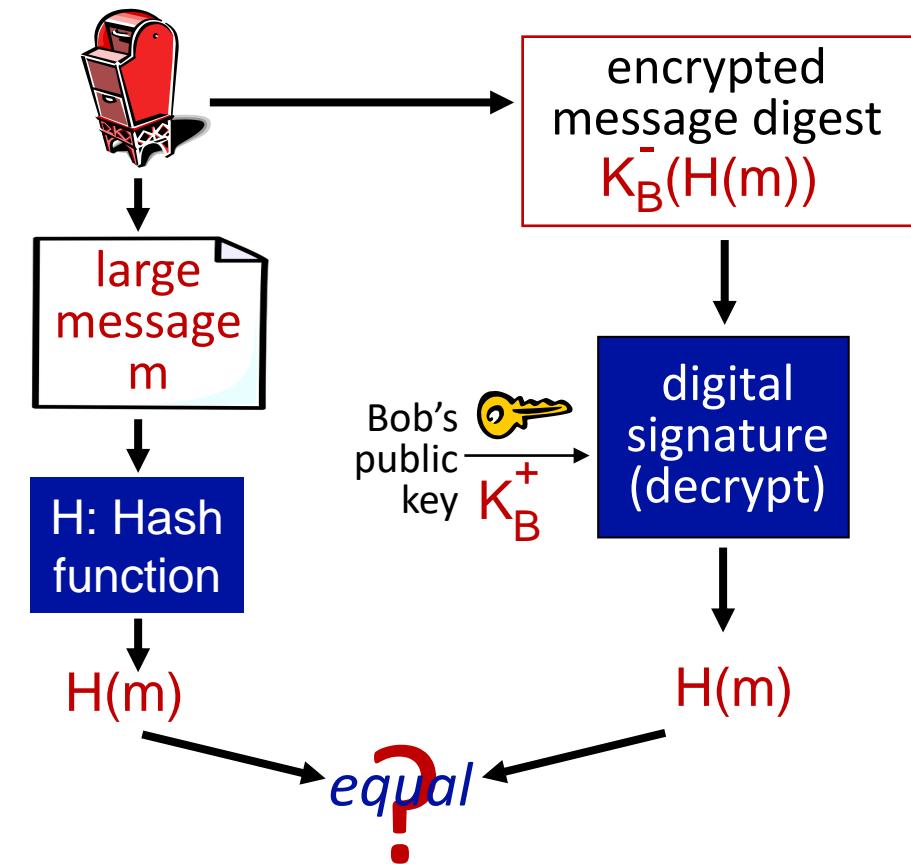
*but identical checksums!*

Digital signature = signed message digest

Bob sends digitally signed message:



Alice verifies signature, integrity  
of digitally signed message:

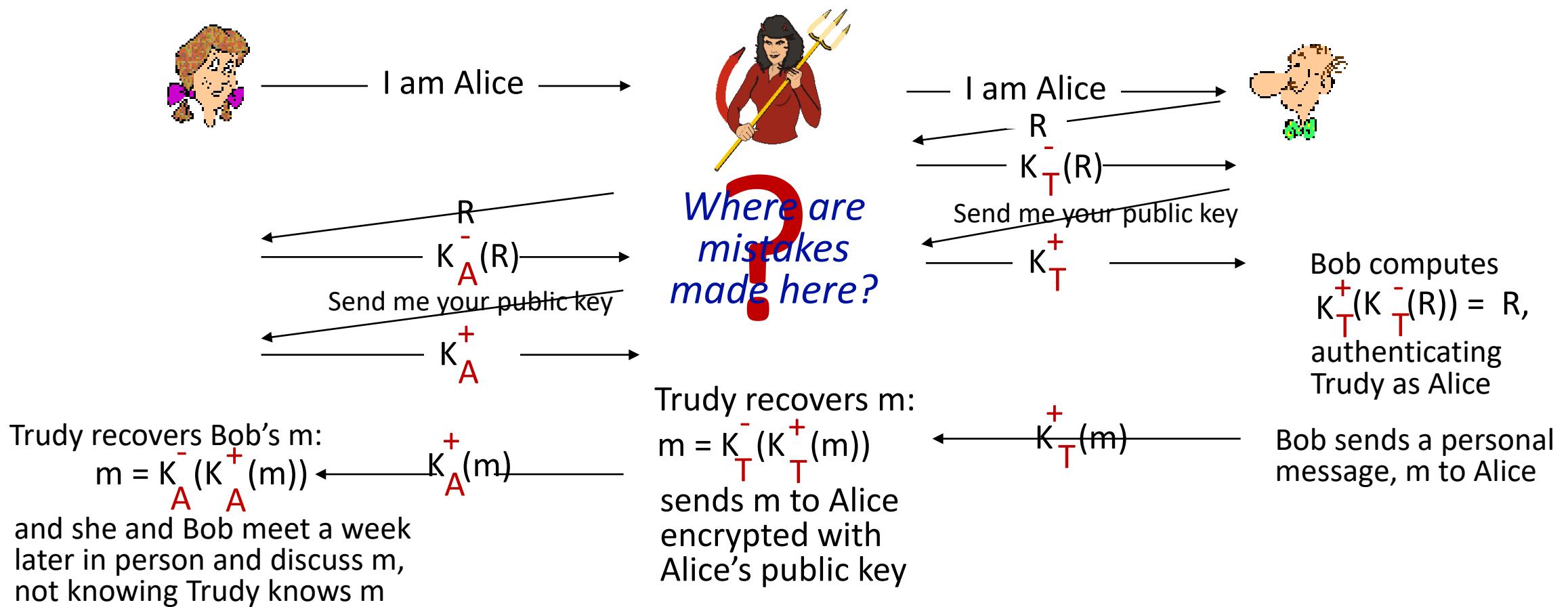


# Hash function algorithms

- MD5 hash function widely used (RFC 1321)
  - computes 128-bit message digest in 4-step process.
  - arbitrary 128-bit string  $x$ , appears difficult to construct msg  $m$  whose MD5 hash is equal to  $x$
- SHA-1 is also used
  - US standard [NIST, FIPS PUB 180-1]
  - 160-bit message digest

# Authentication: ap5.0 – let's fix it!!

Recall the problem: Trudy poses as Alice (to Bob) and as Bob (to Alice)



# Need for certified public keys

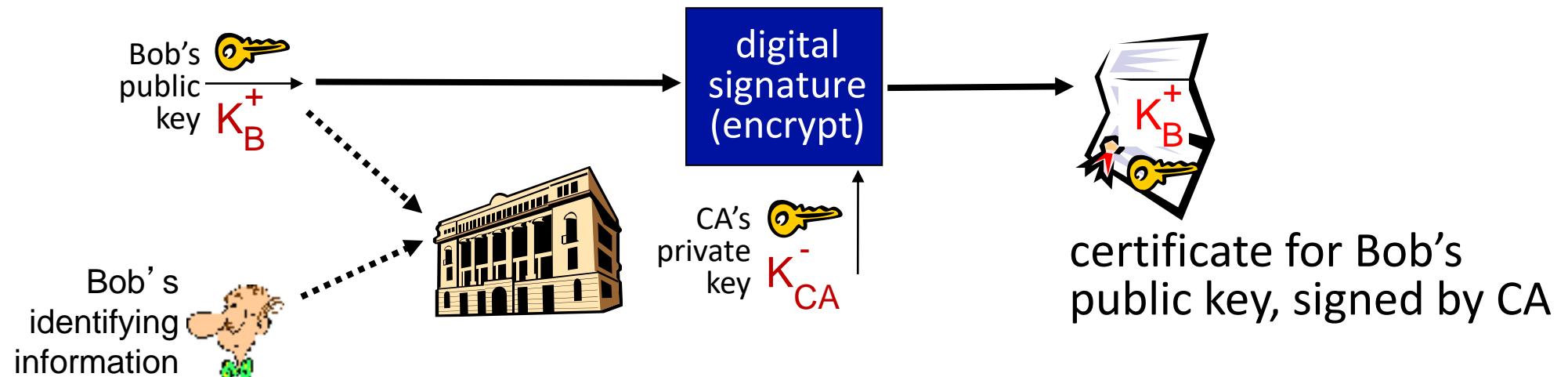
- motivation: Trudy plays pizza prank on Bob

- Trudy creates e-mail order:  
*Dear Pizza Store, Please deliver to me four pepperoni pizzas. Thank you, Bob*
- Trudy signs order with her private key
- Trudy sends order to Pizza Store
- Trudy sends to Pizza Store her public key, but says it's Bob's public key
- Pizza Store verifies signature; then delivers four pepperoni pizzas to Bob
- Bob doesn't even like pepperoni



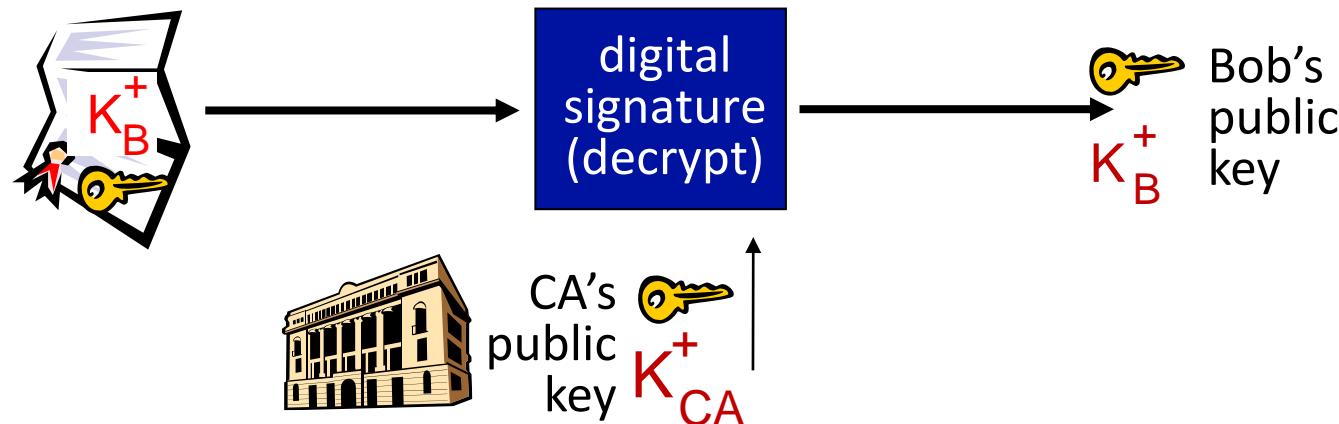
# Public key Certification Authorities (CA)

- certification authority (CA): binds public key to particular entity, E
- entity (person, website, router) registers its public key with CE provides “proof of identity” to CA
  - CA creates certificate binding identity E to E’s public key
  - certificate containing E’s public key digitally signed by CA: CA says “this is E’s public key”



# Public key Certification Authorities (CA)

- when Alice wants Bob's public key:
  - gets Bob's certificate (Bob or elsewhere)
  - apply CA's public key to Bob's certificate, get Bob's public key

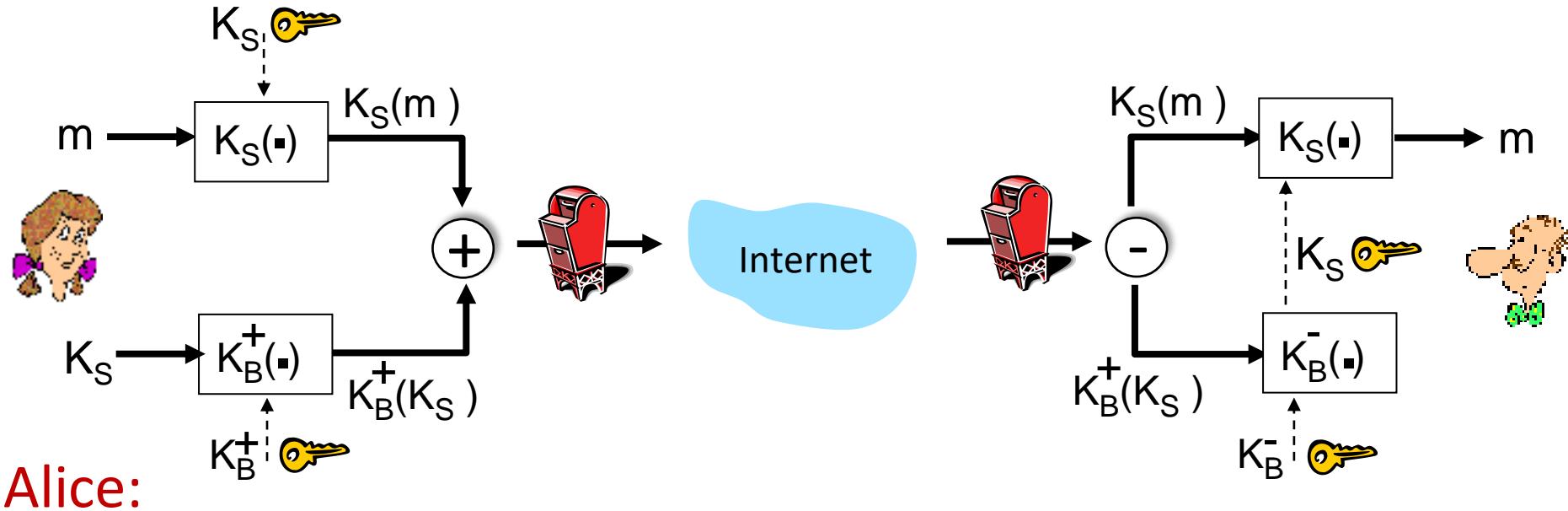


# Outline

- What is network security?
- Principles of cryptography
- Authentication, message integrity
- **Securing e-mail**
- Securing TCP connections: TLS
- Operational security: firewalls

# Secure e-mail: confidentiality

Alice wants to send *confidential* e-mail,  $m$ , to Bob.

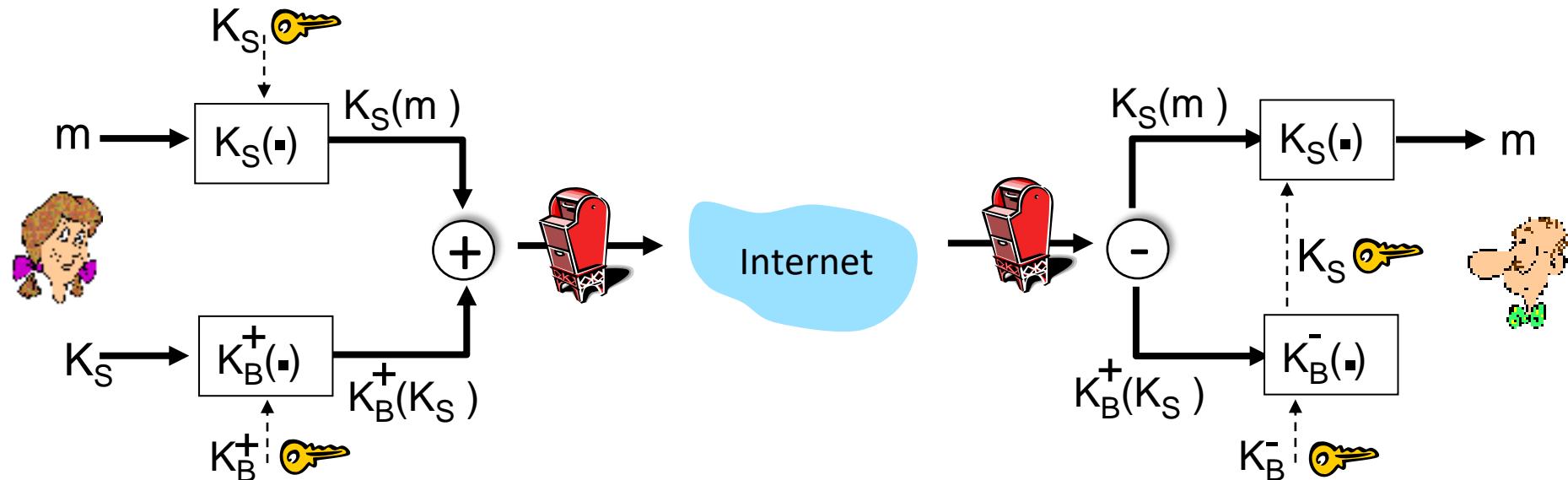


Alice:

- generates random *symmetric* private key,  $K_S$
- encrypts message with  $K_S$  (for efficiency)
- also encrypts  $K_S$  with Bob's public key
- sends both  $K_S(m)$  and  $K_B^+(K_S)$  to Bob

# Secure e-mail: confidentiality (more)

Alice wants to send *confidential* e-mail,  $m$ , to Bob.

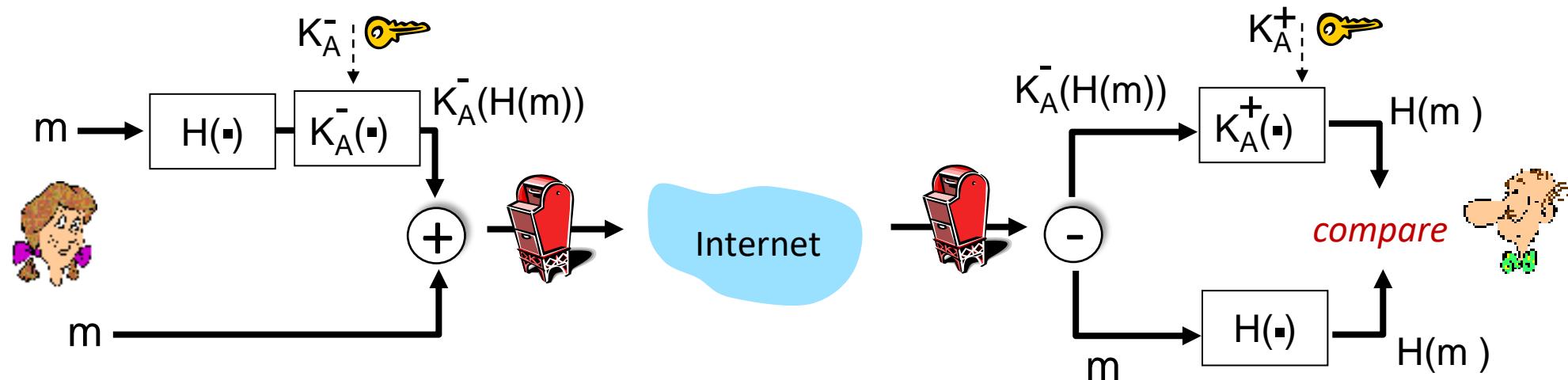


**Bob:**

- uses his private key to decrypt and recover  $K_S$
- uses  $K_S$  to decrypt  $K_S(m)$  to recover  $m$

# Secure e-mail: integrity, authentication

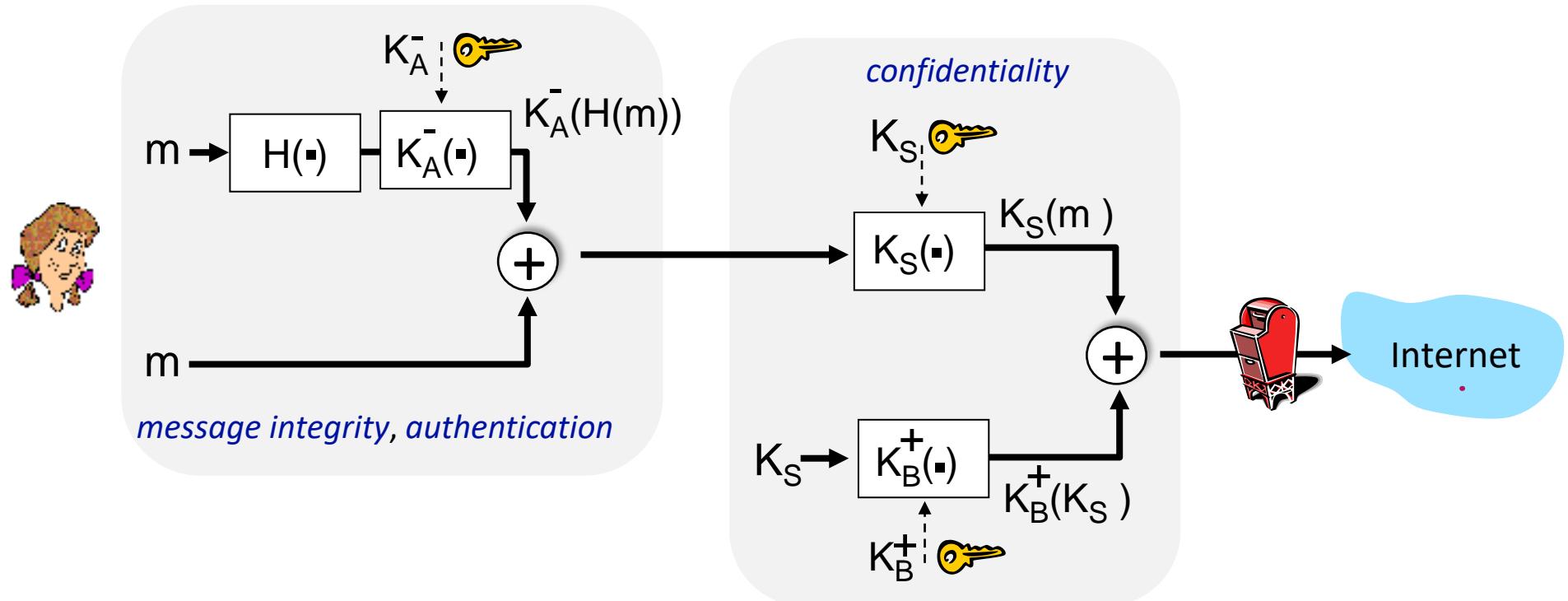
Alice wants to send  $m$  to Bob, with *message integrity, authentication*



- Alice digitally signs hash of her message with her private key, providing integrity and authentication
- sends both message (in the clear) and digital signature

# Secure e-mail: integrity, authentication

Alice sends  $m$  to Bob, with *confidentiality, message integrity, authentication*



Alice uses three keys: her private key, Bob's public key, new symmetric key

*What are Bob's complementary actions?*

# Outline

- What is network security?
- Principles of cryptography
- Authentication, message integrity
- Securing e-mail
- Securing TCP connections: TLS**
- Operational security: firewalls

# Transport-layer security (TLS)

- widely deployed security protocol above the transport layer
    - supported by almost all browsers, web servers: https (port 443)
  - provides:
    - **confidentiality**: via *symmetric encryption*
    - **integrity**: via *cryptographic hashing*
    - **authentication**: via *public key cryptography*
  - history:
    - early research, implementation: secure network programming, secure sockets
    - secure socket layer (SSL) deprecated [2015]
    - TLS 1.3: RFC 8846 [2018]
- 
- all techniques we have studied!*

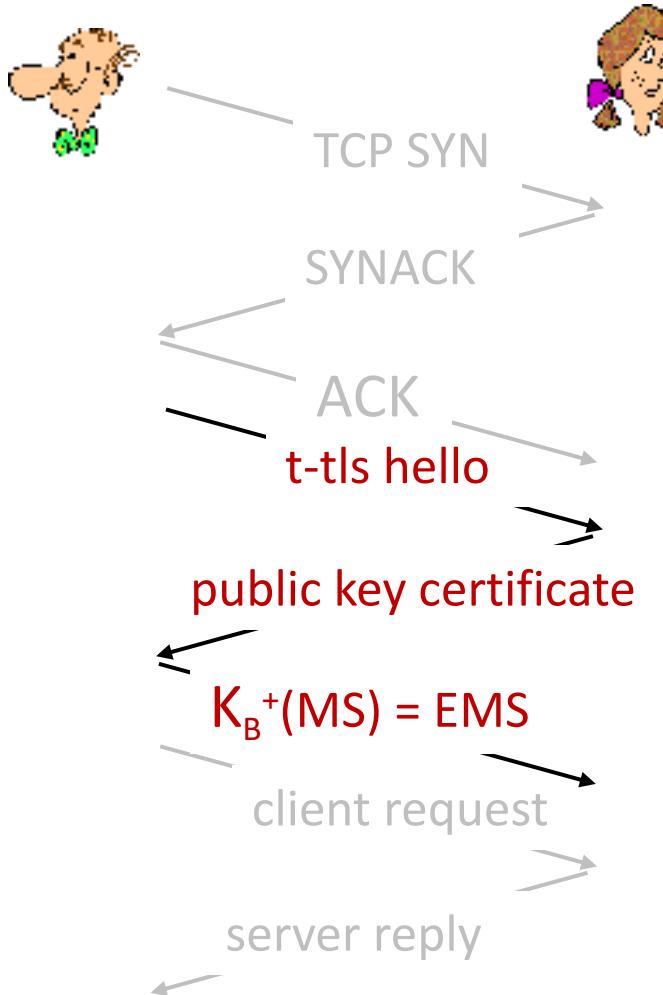
# Transport-layer security (TLS)

- widely deployed security protocol above the transport layer
    - supported by almost all browsers, web servers: https (port 443)
  - provides:
    - **confidentiality**: via *symmetric encryption*
    - **integrity**: via *cryptographic hashing*
    - **authentication**: via *public key cryptography*
  - history:
    - early research, implementation: secure network programming, secure sockets
    - secure socket layer (SSL) deprecated [2015]
    - TLS 1.3: RFC 8846 [2018]
- 
- all techniques we have studied!*

# Transport-layer security: what's needed?

- let's *build* a toy TLS protocol, *t-tls*, to see what's needed!
- we've seen the “pieces” already:
  - **handshake**: Alice, Bob use their certificates, private keys to authenticate each other, exchange or create shared secret
  - **key derivation**: Alice, Bob use shared secret to derive set of keys
  - **data transfer**: stream data transfer: data as a series of records
    - not just one-time transactions
  - **connection closure**: special messages to securely close connection

# t-tls: initial handshake



## t-tls handshake phase:

- Bob establishes TCP connection with Alice
- Bob verifies that Alice is really Alice
- Bob sends Alice a master secret key (MS), used to generate all other keys for TLS session
- potential issues:
  - 3 RTT before client can start receiving data (including TCP handshake)

# t-tls: cryptographic keys

- considered bad to use same key for more than one cryptographic function
  - different keys for message authentication code (MAC) and encryption
- four keys:
  - 🔑  $K_c$  : encryption key for data sent from client to server
  - 🔑  $M_c$  : MAC key for data sent from client to server
  - 🔑  $K_s$  : encryption key for data sent from server to client
  - 🔑  $M_s$  : MAC key for data sent from server to client
- keys derived from key derivation function (KDF)
  - takes master secret and (possibly) some additional random data to create new keys

# t-tls: encrypting data

- recall: TCP provides data *byte stream* abstraction
- Q: can we encrypt data in-stream as written into TCP socket?
  - A: where would MAC go? If at end, no message integrity until all data received and connection closed!
  - solution: break stream in series of “records”
    - each client-to-server record carries a MAC, created using  $M_c$
    - receiver can act on each record as it arrives
- t-tls record encrypted using symmetric key,  $K_c$ , passed to TCP:

$K_c($    $)$

# t-tls: encrypting data (more)

- possible attacks on data stream?
  - *re-ordering*: man-in middle intercepts TCP segments and reorders (manipulating sequence #s in unencrypted TCP header)
  - *replay*
- solutions:
  - use TLS sequence numbers (data, TLS-seq-# incorporated into MAC)
  - use nonce

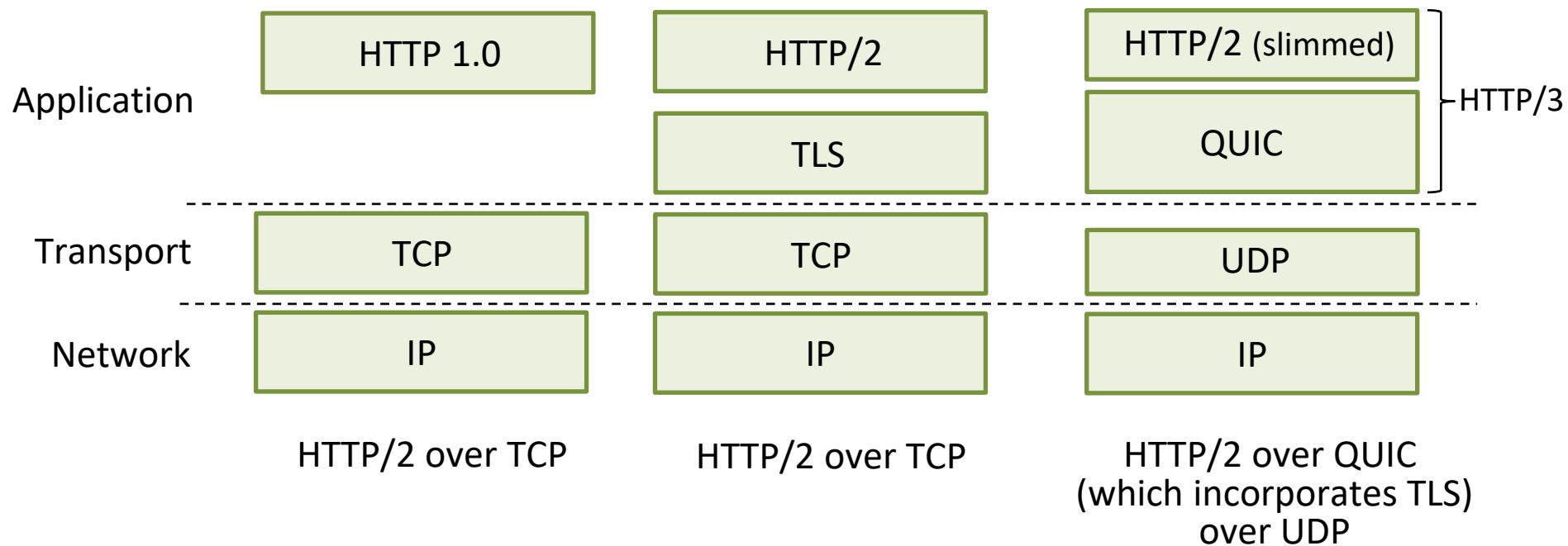
# t-tls: connection close

- truncation attack:
  - attacker forges TCP connection close segment
  - one or both sides thinks there is less data than there actually is
- solution: record types, with one type for closure
  - type 0 for data; type 1 for close
- MAC now computed using data, type, sequence #

$$K_C( \begin{array}{|c|c|c|c|} \hline length & type & data & MAC \\ \hline \end{array} )$$

# Transport-layer security (TLS)

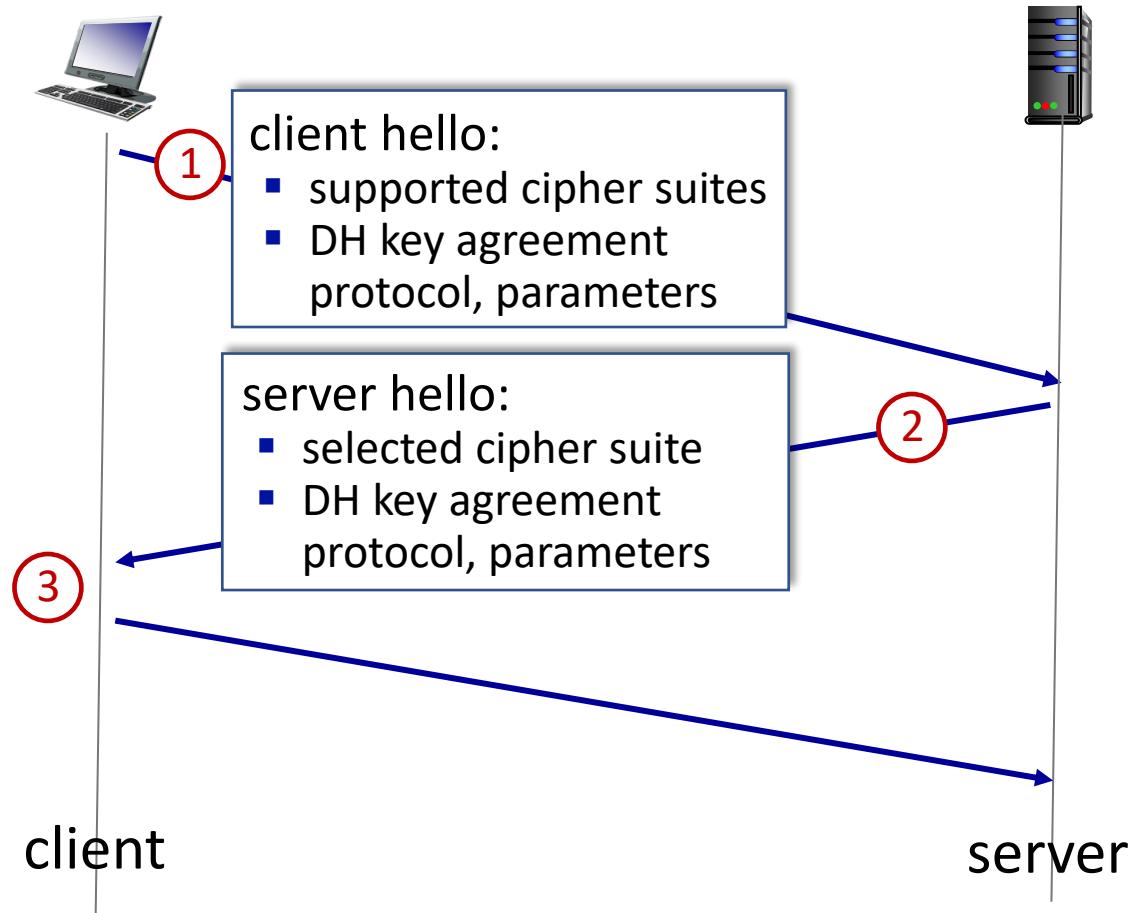
- TLS provides an API that *any* application can use
- an HTTP view of TLS:



# TLS: 1.3 cipher suite

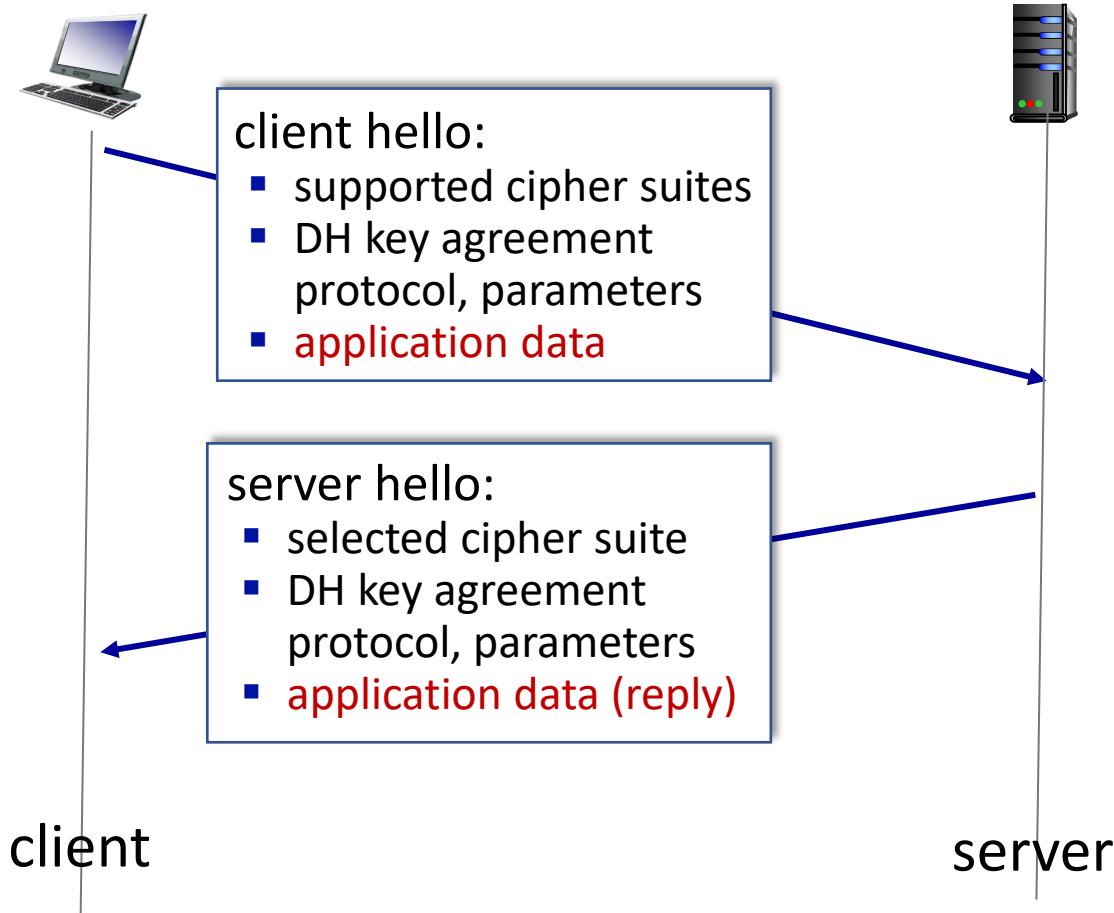
- “cipher suite”: algorithms that can be used for key generation, encryption, MAC, digital signature
- TLS: 1.3 (2018): more limited cipher suite choice than TLS 1.2 (2008)
  - only 5 choices, rather than 37 choices
  - *requires* Diffie-Hellman (DH) for key exchange, rather than DH or RSA
  - combined encryption and authentication algorithm (“authenticated encryption”) for data rather than serial encryption, authentication
    - 4 based on AES
  - HMAC uses SHA (256 or 284) cryptographic hash function

# TLS 1.3 handshake: 1 RTT



- ① client TLS hello msg:
  - guesses key agreement protocol, parameters
  - indicates cipher suites it supports
- ② server TLS hello msg chooses
  - key agreement protocol, parameters
  - cipher suite
  - server-signed certificate
- ③ client:
  - checks server certificate
  - generates key
  - can now make application request (e.g., HTTPS GET)

# TLS 1.3 handshake: 0 RTT



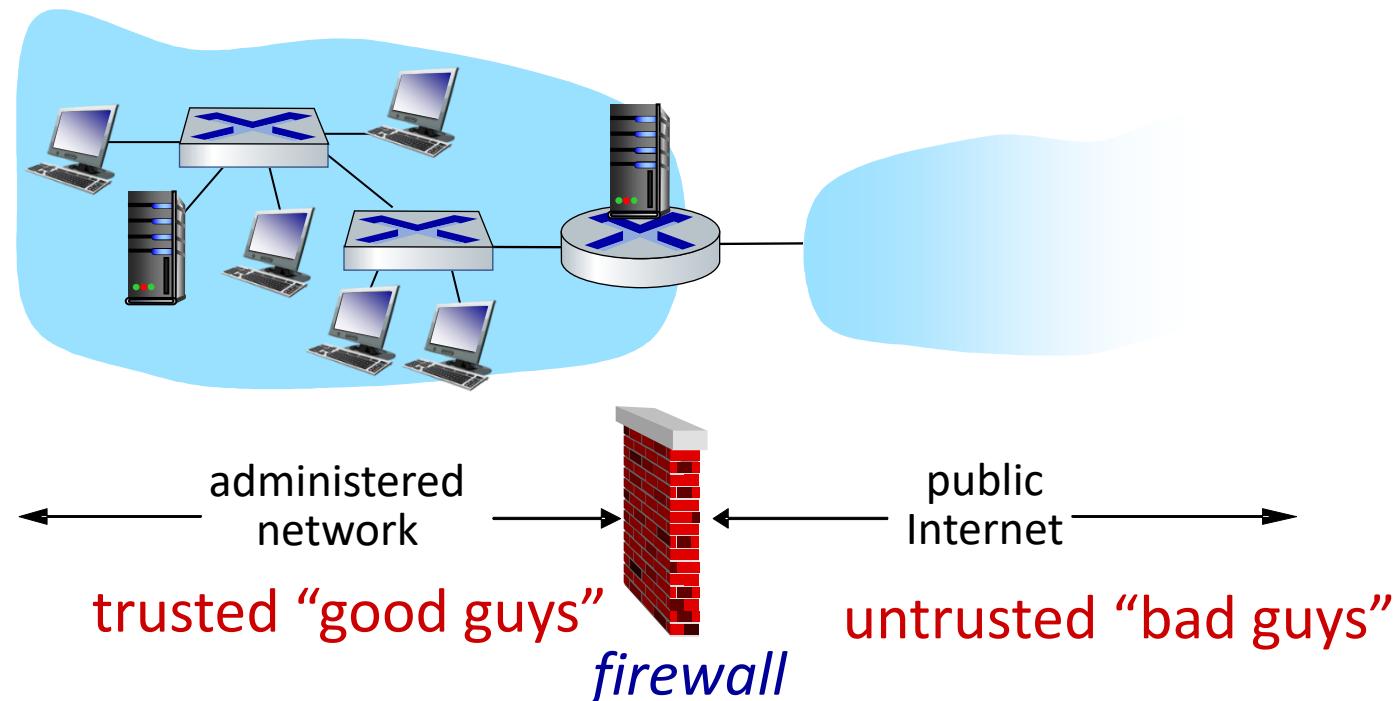
- initial hello message contains encrypted application data!
  - “resuming” earlier connection between client and server
  - application data encrypted using “resumption master secret” from earlier connection
- vulnerable to replay attacks!
  - maybe OK for get HTTP GET or client requests not modifying server state

# Outline

- What is network security?
- Principles of cryptography
- Authentication, message integrity
- Securing e-mail
- Securing TCP connections: TLS
- **Operational security: firewalls**

# Firewalls

**firewall**  
✓ isolates organization's internal network from larger Internet, allowing some packets to pass, blocking others



# Firewalls: why

✓ prevent denial of service attacks:

- SYN flooding: attacker establishes many bogus TCP connections, no resources left for “real” connections

✓ prevent illegal modification/access of internal data

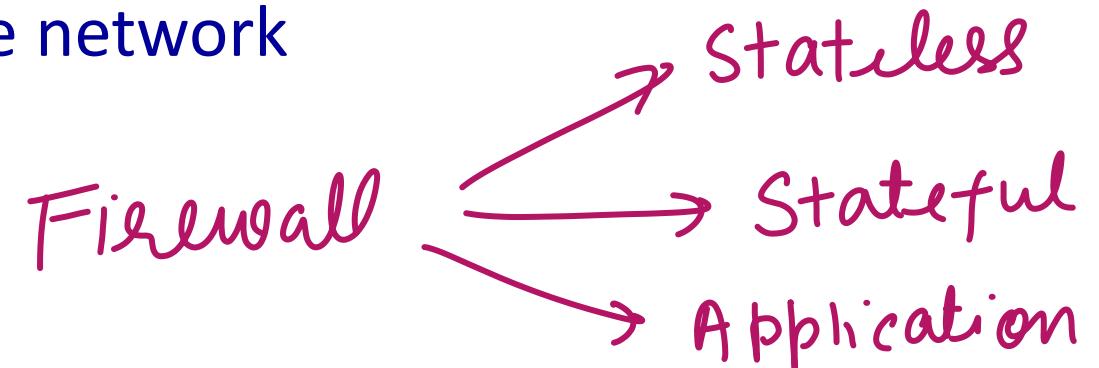
- e.g., attacker replaces CIA’s homepage with something else

✓ allow only authorized access to inside network

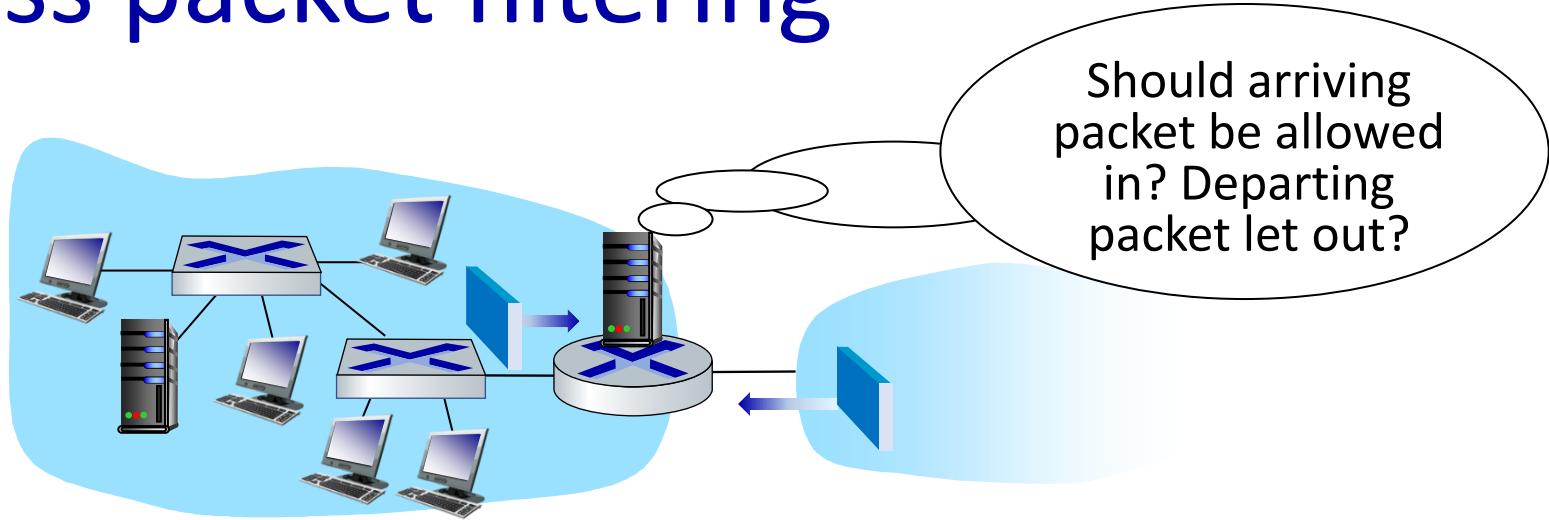
- set of authenticated users/hosts

three types of firewalls:

- stateless packet filters
- stateful packet filters
- application gateways

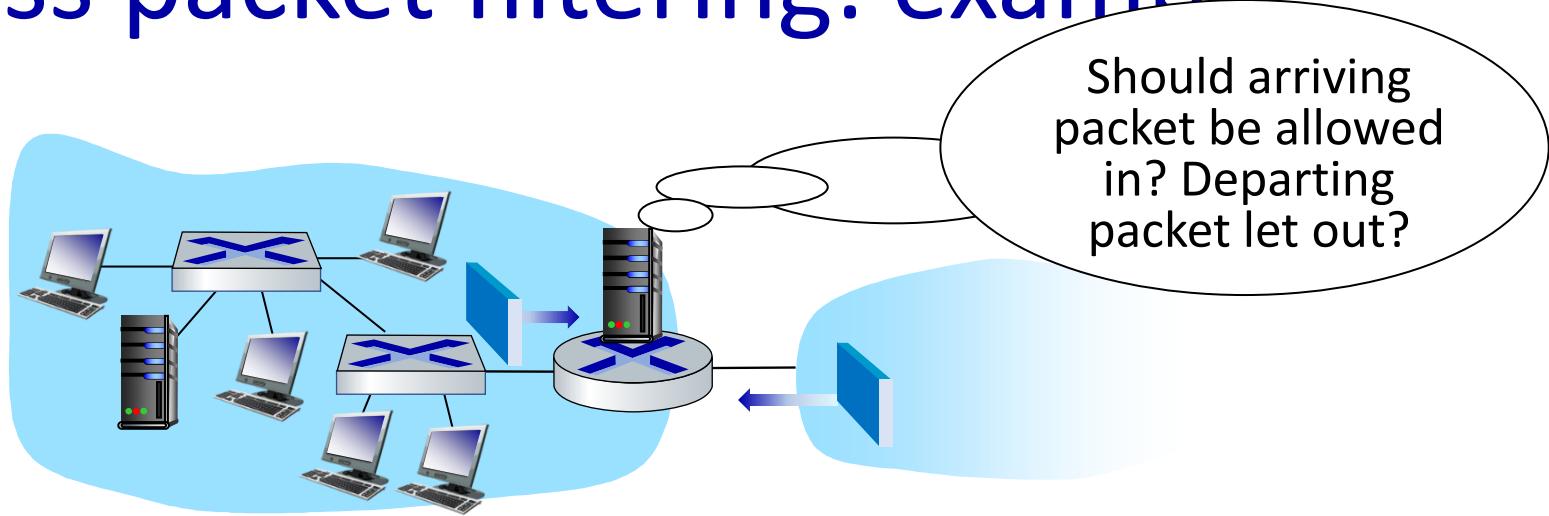


# Stateless packet filtering



- internal network connected to Internet via router **firewall**
- filters **packet-by-packet**, decision to forward/drop packet based on:
  - ✓ source IP address, destination IP address
  - ✓ TCP/UDP source, destination port numbers
  - ✓ ICMP message type
  - ✓ TCP SYN, ACK bits

# Stateless packet filtering: example



- **example 1:** block incoming and outgoing datagrams with IP protocol field = 17 and with either source or dest port = 23
  - **result:** all incoming, outgoing UDP flows and telnet connections are blocked
- **example 2:** block inbound TCP segments with ACK=0
  - **result:** prevents external clients from making TCP connections with internal clients, but allows internal clients to connect to outside

# Stateless packet filtering: more examples

| Policy                                                                              | Firewall Setting                                                            |
|-------------------------------------------------------------------------------------|-----------------------------------------------------------------------------|
| no outside Web access                                                               | drop all outgoing packets to any IP address, port 80                        |
| no incoming TCP connections, except those for institution's public Web server only. | drop all incoming TCP SYN packets to any IP except 130.207.244.203, port 80 |
| prevent Web-radios from eating up the available bandwidth.                          | drop all incoming UDP packets - except DNS and router broadcasts.           |
| prevent your network from being used for a smurf DoS attack.                        | drop all ICMP packets going to a "broadcast" address (e.g. 130.207.255.255) |
| prevent your network from being tracerouted                                         | drop all outgoing ICMP TTL expired traffic                                  |

# Access Control Lists

**ACL:** table of rules, applied top to bottom to incoming packets: (action, condition) pairs: looks like OpenFlow forwarding (Ch. 4)!

| action | source address          | dest address            | protocol | source port | dest port | flag bit |
|--------|-------------------------|-------------------------|----------|-------------|-----------|----------|
| allow  | 222.22/16               | outside of<br>222.22/16 | TCP      | > 1023      | 80        | any      |
| allow  | outside of<br>222.22/16 | 222.22/16               | TCP      | 80          | > 1023    | ACK      |
| allow  | 222.22/16               | outside of<br>222.22/16 | UDP      | > 1023      | 53        | ---      |
| allow  | outside of<br>222.22/16 | 222.22/16               | UDP      | 53          | > 1023    | ----     |
| deny   | all                     | all                     | all      | all         | all       | all      |

# Stateful packet filtering

- *stateless packet filter*: heavy handed tool

- admits packets that “make no sense,” e.g., dest port = 80, ACK bit set, even though no TCP connection established:

| action | source address          | dest address | protocol | source port | dest port | flag bit |
|--------|-------------------------|--------------|----------|-------------|-----------|----------|
| allow  | outside of<br>222.22/16 | 222.22/16    | TCP      | 80          | > 1023    | ACK      |

- *stateful packet filter*: track status of every TCP connection

- track connection setup (SYN), teardown (FIN): determine whether incoming, outgoing packets “makes sense”
  - timeout inactive connections at firewall: no longer admit packets

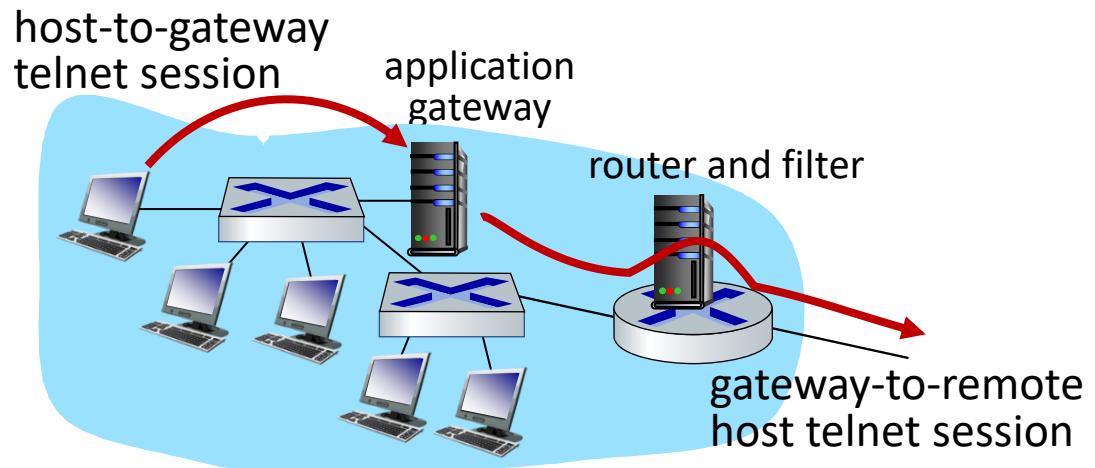
# Stateful packet filtering

ACL augmented to indicate need to check connection state table before admitting packet

| action | source address       | dest address         | proto | source port | dest port | flag bit | check connection |
|--------|----------------------|----------------------|-------|-------------|-----------|----------|------------------|
| allow  | 222.22/16            | outside of 222.22/16 | TCP   | > 1023      | 80        | any      |                  |
| allow  | outside of 222.22/16 | 222.22/16            | TCP   | 80          | > 1023    | ACK      | X                |
| allow  | 222.22/16            | outside of 222.22/16 | UDP   | > 1023      | 53        | ---      |                  |
| allow  | outside of 222.22/16 | 222.22/16            | UDP   | 53          | > 1023    | ----     | X                |
| deny   | all                  | all                  | all   | all         | all       | all      |                  |

# Application gateways

- filter packets on application data as well as on IP/TCP/UDP fields.
- *example:* allow select internal users to telnet outside



1. require all telnet users to telnet through gateway.
2. for authorized users, gateway sets up telnet connection to dest host
  - gateway relays data between 2 connections
3. router filter blocks all telnet connections not originating from gateway

# Limitations of firewalls, gateways

- **IP spoofing:** router can't know if data "really" comes from claimed source
- if multiple apps need special treatment, each has own app. gateway
- client software must know how to contact gateway
  - e.g., must set IP address of proxy in Web browser
- filters often use all or nothing policy for UDP
- ***tradeoff:*** degree of communication with outside world, level of security
- many highly protected sites still suffer from attacks