


# Computer and Communication Networks (2026)

Iordanis Koutsopoulos, Vasilios Siris, George Xylomenos

Professors, Department of Informatics, AUEB

# Acknowledgment

These slides have been adapted from the slideset accompanying the book Computer Networking: A Top-Down Approach, 9th edition, by J.F Kurose and K.W. Ross, Pearson 2025.

Original slides copyright notice: 1996-2025, J.F Kurose and K.W. Ross, All Rights Reserved

# Network layer: our goals

- Understand principles behind **network layer services**, focusing on data plane:
  - Network layer service models
  - Forwarding versus routing
  - How a router works
  - Addressing
  - Generalized forwarding
  - Internet architecture

- Implementation in the Internet
  - IP protocol
  - NAT



# Network layer: “data plane” roadmap

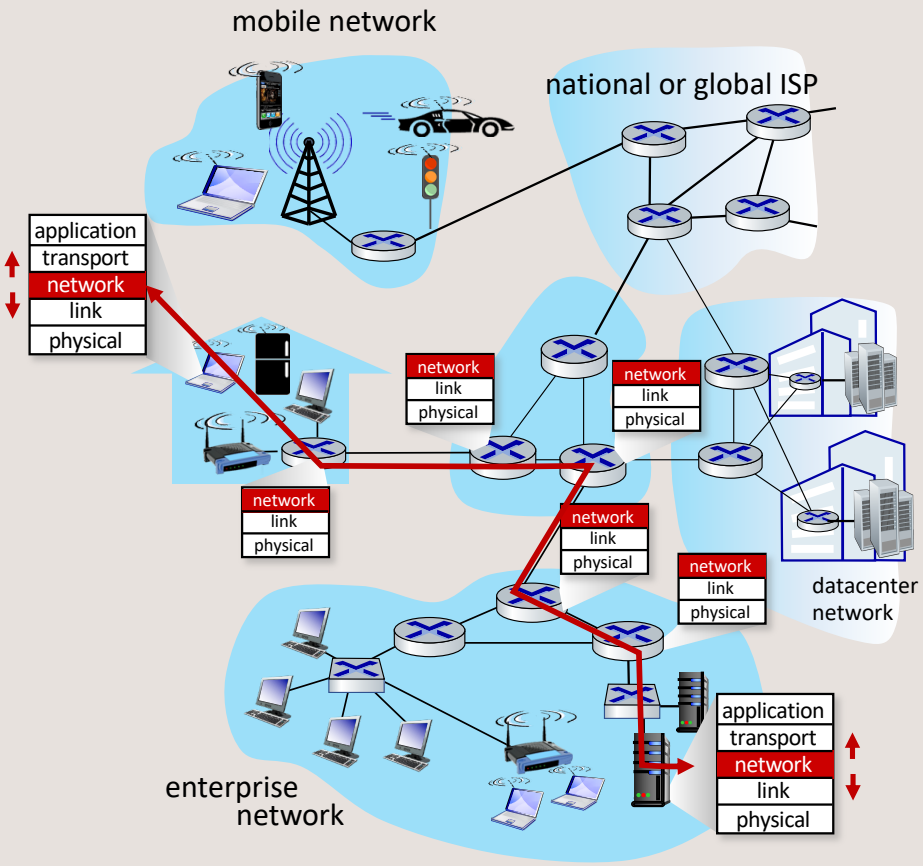
- Network layer: overview
  - data plane
  - control plane
- What’s inside a router
  - input/output ports, switching
  - buffer management
  - scheduling
- IP: the Internet Protocol
  - datagram format, addressing
  - network address translation (NAT)
  - IPv6



- Generalized Forwarding
  - Match+action
  - OpenFlow
- Internet architecture

# 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



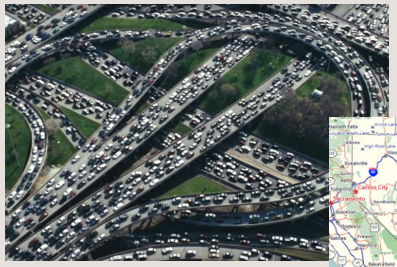
# Two key network-layer functions

## 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

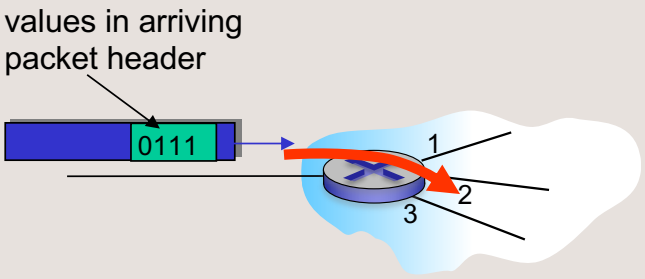


routing

# 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

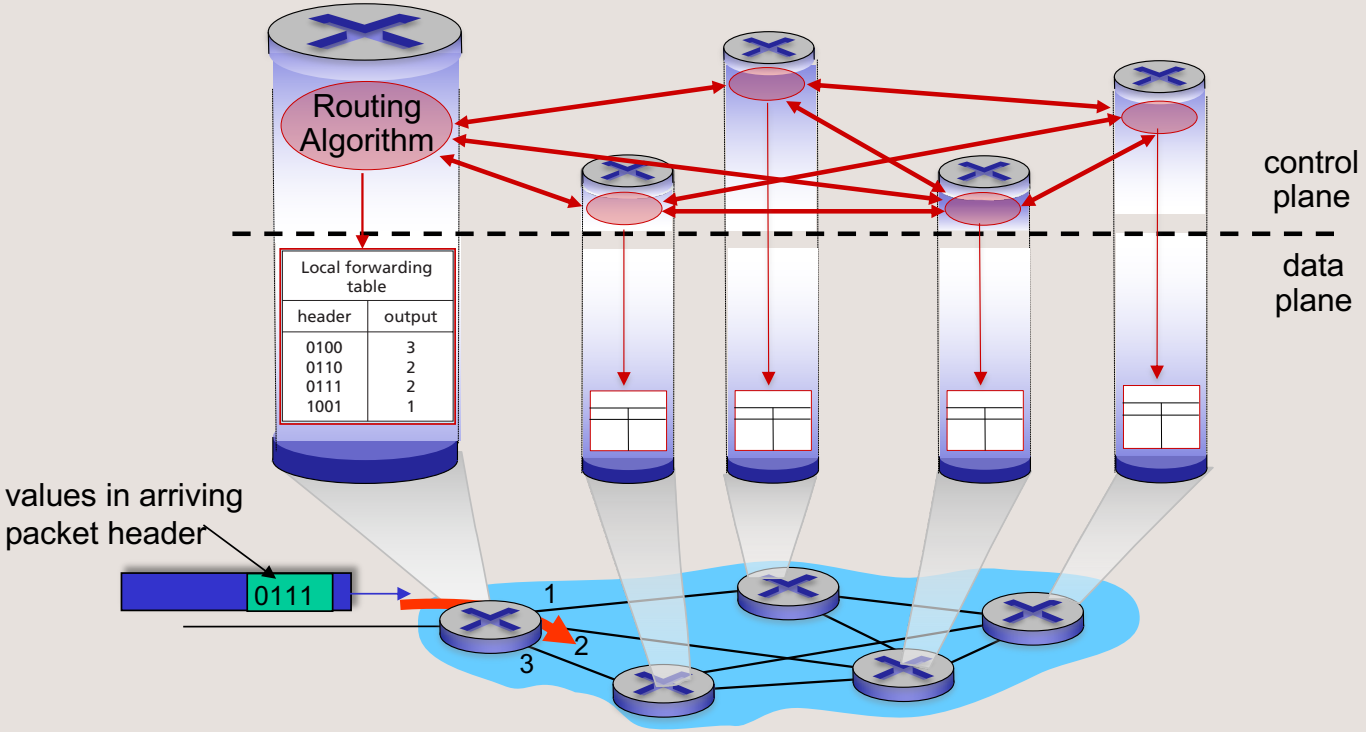



## Control plane

- *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

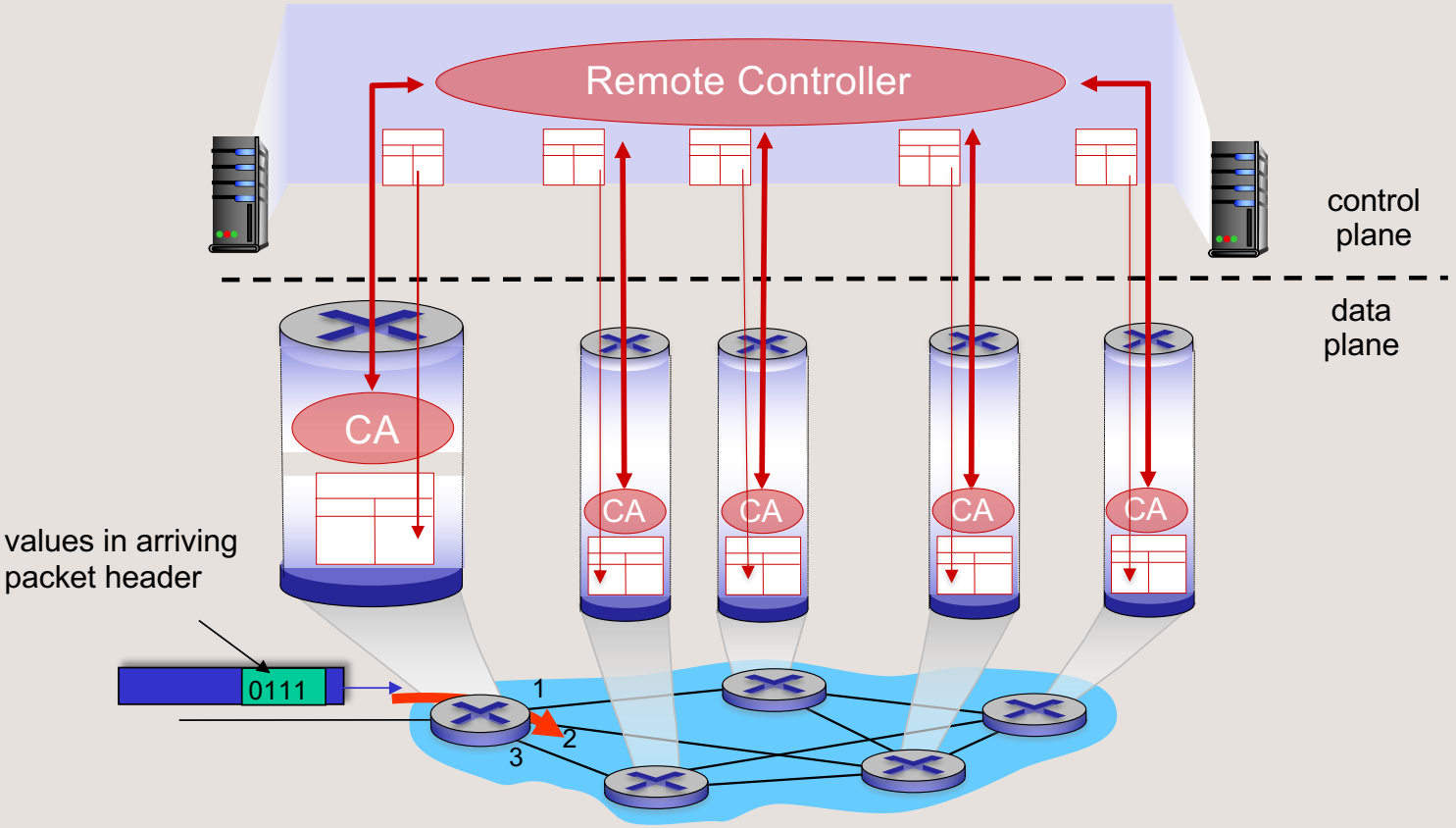
# Per-router control plane

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



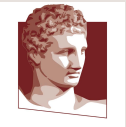
# Software-Defined Networking (SDN) control plane

## Remote controller computes, installs forwarding tables in routers



# Software Defined Networking

- Basic idea: Centralization of network operation
- Controller: Software that manages the network
- Placement of Controller at a network node
- Distribution of new code, state information, and configuration settings from the Controller to Routers/Switches; updating forwarding tables
- Controller: Maintains a global view of the network based on information collected periodically
  - Potentially high computational workload
- Switch/Router: Performs local computations using the forwarding table, enabling fast execution
- Does SDN help simplify network design?
  - YES: It shifts complexity to the Controller and standardizes and simplifies the installation of rules on Switches.



# Network service model

*Q:* What *service model* for "channel" transporting datagrams from sender to receiver?

example services for *individual* datagrams:

- guaranteed delivery
- guaranteed delivery with less than 40 msec delay

example services for a *flow* of datagrams:

- in-order datagram delivery
- guaranteed minimum bandwidth to flow
- restrictions on changes in inter-packet spacing

# Network-layer service model

| Network Architecture | Service Model | Quality of Service (QoS) Guarantees ? | | | |
|---|---|---|---|---|---|
| | | Bandwidth | Loss | Order | Timing |
| Internet | best effort | none | no | no | no |

Internet "best effort" service model

*No* guarantees on:

i. successful datagram delivery to destination
ii. timing or order of delivery
iii. bandwidth available to end-end flow

# Network-layer service model

| Network Architecture | Service Model | Quality of Service (QoS) Guarantees ? | | | |
|---|---|---|---|---|---|
| | | Bandwidth | Loss | Order | Timing |
| Internet | best effort | none | no | no | no |
| ATM | Constant Bit Rate | Constant rate | yes | yes | yes |
| ATM | Available Bit Rate | Guaranteed min | no | yes | no |
| Internet | Intserv Guaranteed (RFC 1633) | yes | yes | yes | yes |
| Internet | Diffserv (RFC 2475) | possible | possibly | possibly | no |

# Reflections on best-effort service:

- simplicity of mechanism has allowed Internet to be widely deployed adopted
- sufficient provisioning of bandwidth allows performance of real-time applications (e.g., interactive voice, video) to be "good enough" for "most of the time"
- replicated, application-layer distributed services (datacenters, content distribution networks) connecting close to clients' networks, allow services to be provided from multiple locations
- congestion control of "elastic" services helps

*It's hard to argue with success of best-effort service model*

# Network layer: “data plane” roadmap

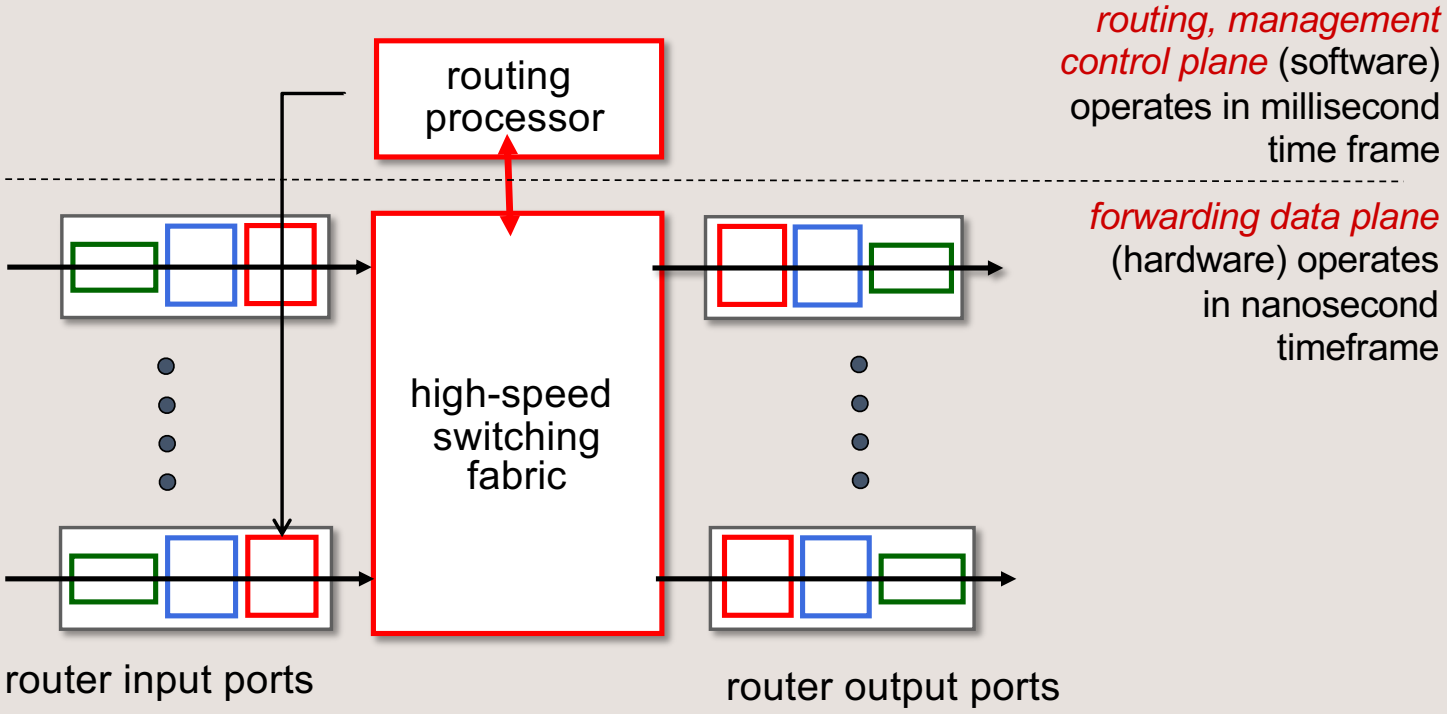
- Network layer: overview
  - data plane
  - control plane
- **What’s inside a router**
  - input/output ports, switching
  - buffer management
  - scheduling
- IP: the Internet Protocol
  - datagram format, addressing
  - network address translation (NAT)
  - IPv6



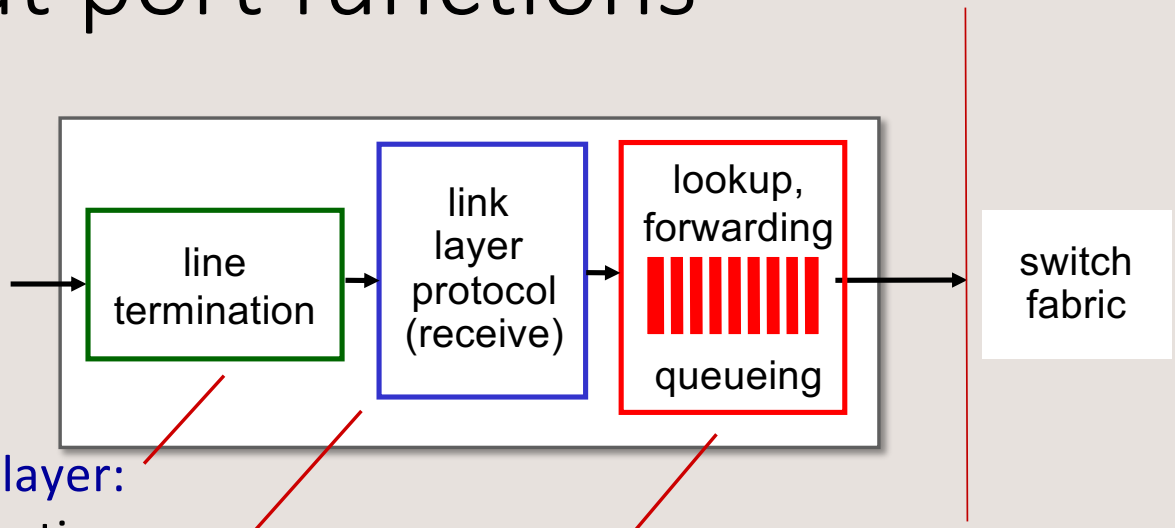
- Generalized Forwarding
  - Match+action
  - OpenFlow
- Internet architecture

# Router architecture overview

high-level view of generic router architecture:



# Input port functions



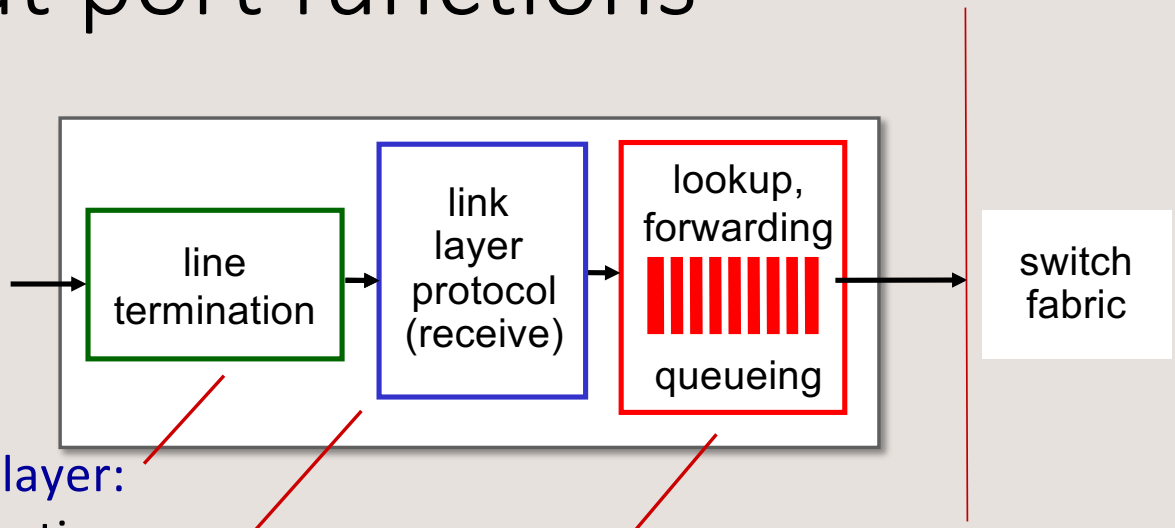

physical layer:
bit-level reception

link layer:
e.g., Ethernet (chapter 6)

## decentralized switching:

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

# Input port functions




physical layer:
bit-level reception

link layer:
e.g., Ethernet
(chapter 6)

## 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)

# Destination-based forwarding

*forwarding table*

| **Destination Address Range** | Link Interface |
|---|---|
| 11001000 00010111 00010000 00000000<br>through<br>11001000 00010111 00010111 11111111 | 0 |
| 11001000 00010111 00011000 00000000<br>through<br>11001000 00010111 00011000 11111111 | 1 |
| 11001000 00010111 00011001 00000000<br>through<br>11001000 00010111 00011111 11111111 | 2 |
| otherwise | 3 |

# 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:

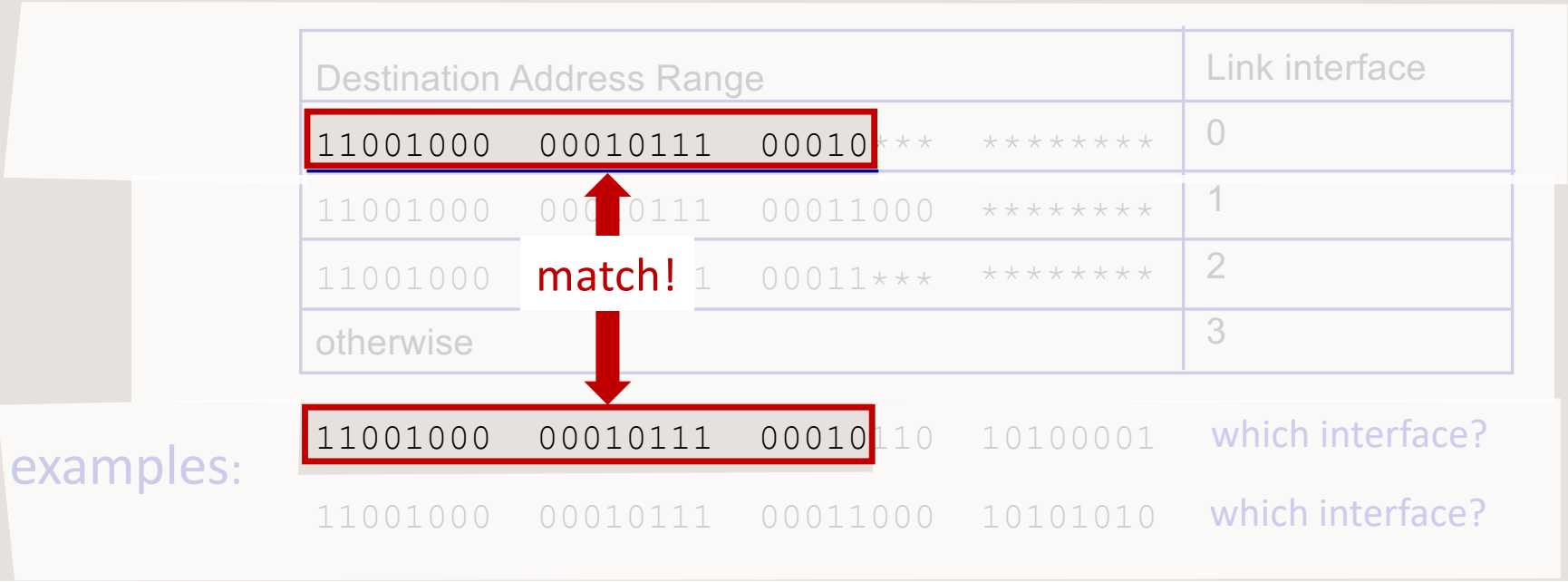
11001000 00010111 00010110 10100001 which interface?

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.



# 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!

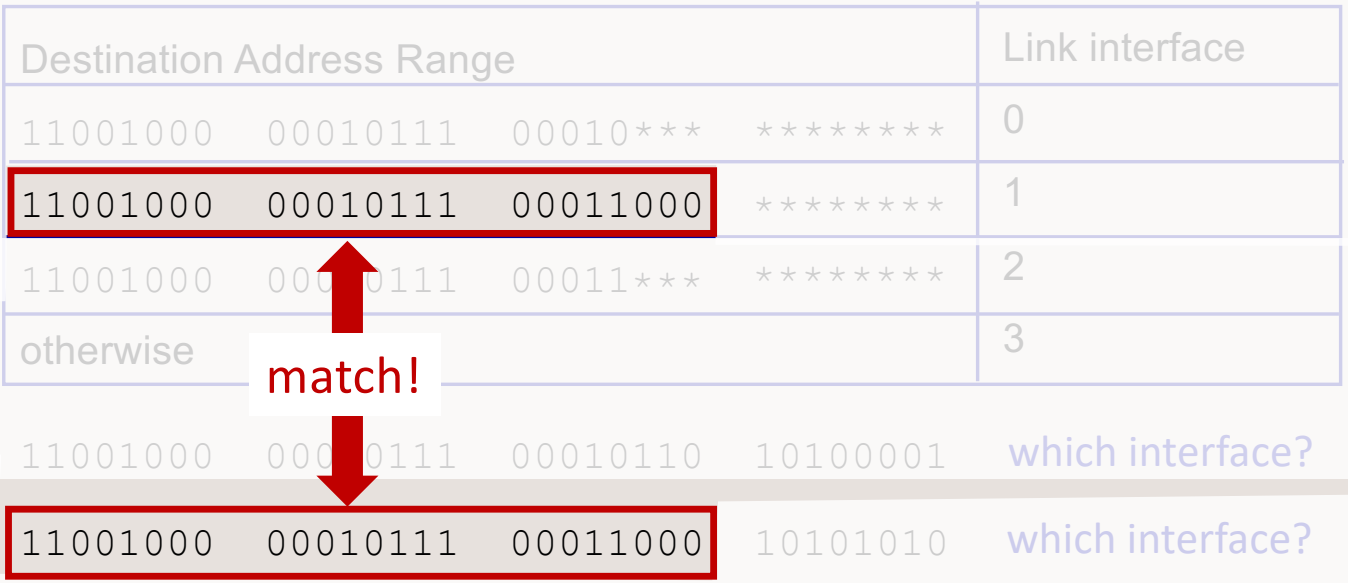
examples:

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

# Longest prefix matching

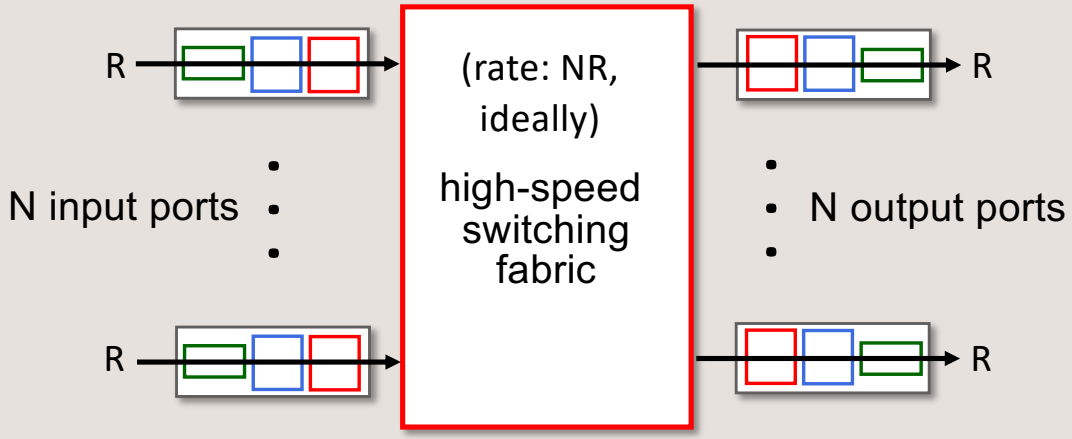
**longest prefix match**

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



# Switching fabrics

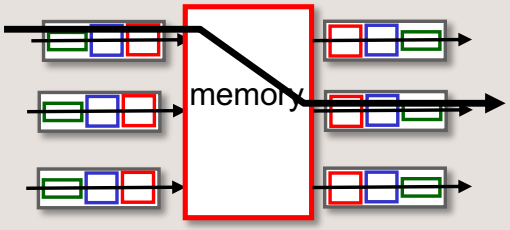
- transfer packet from input link to appropriate output link
- switching rate: rate at which packets can be transferred from inputs to outputs
  - often measured as multiple of input/output line rate
  - N inputs: switching rate N times line rate is desirable



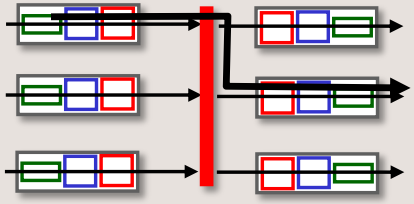



# Switching fabrics

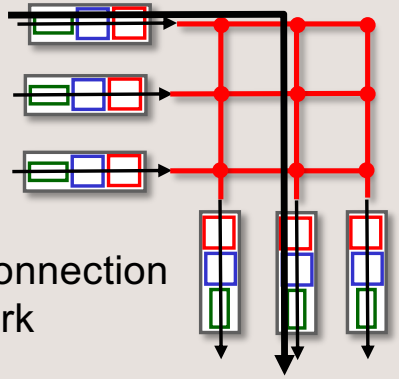
- transfer packet from input link to appropriate output link
- switching rate: rate at which packets can be transferred from inputs to outputs
  - often measured as multiple of input/output line rate
  - N inputs: switching rate N times line rate is desirable
- three major types of switching fabrics:




memory



bus



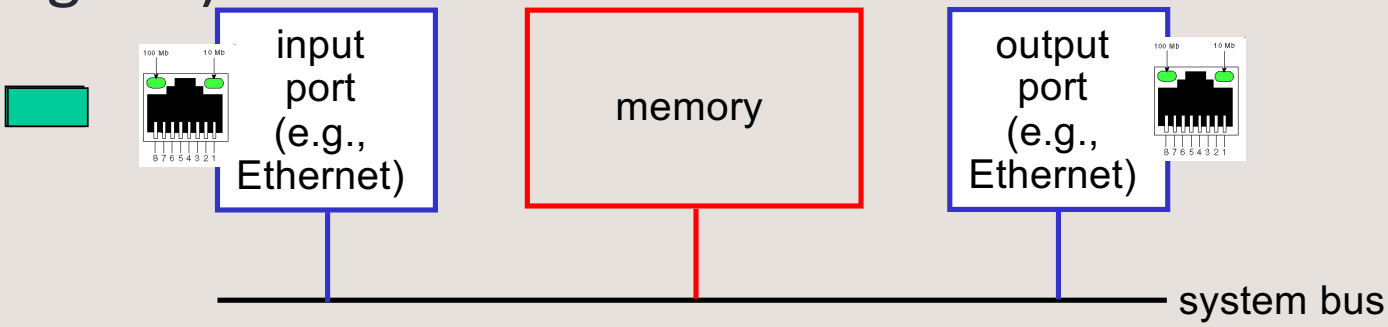
interconnection network



# Switching via memory

first generation routers:

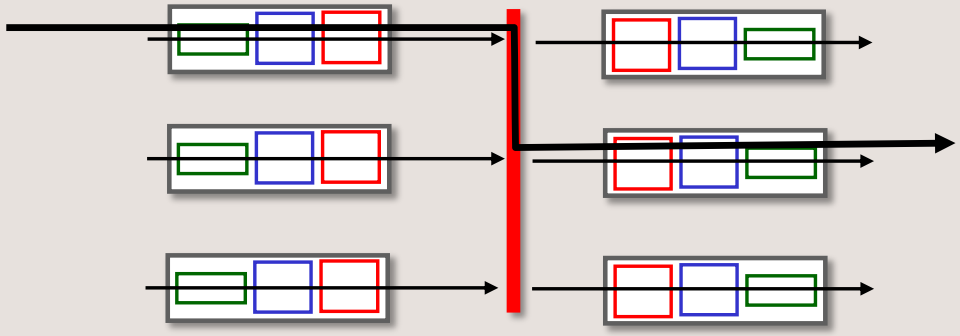
- traditional computers with switching under direct control of CPU
- packet copied to system's memory
- speed limited by memory bandwidth (2 bus crossings per datagram)






# Switching via a bus

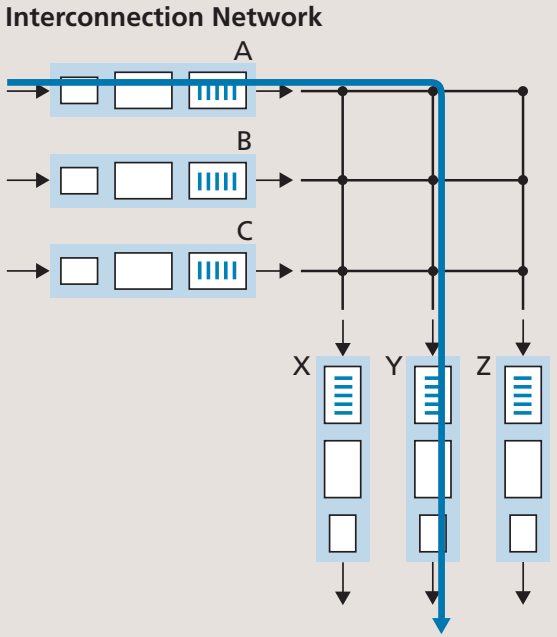
- datagram from input port memory to output port memory via a shared bus
- *bus contention:* switching speed limited by bus bandwidth
- 32 Gbps bus, Cisco 5600: sufficient speed for access routers





# Switching via a Cross-bar

- Overcomes bandwidth limitations of the bus
- Crossbar and other interconnection networks were developed to interconnect processors in multiprocessor systems.
- 2N intersecting buses for N input ports and N output ports.
- Packets can be transmitted simultaneously from A to Y and from B to X (because they use different buses)
- However: If two packets arriving at two different input ports are destined for the same output port, one must wait, because only one packet can be forwarded to an output port at any given time.
- Cisco 12000: switches 60 Gbps through its interconnection network.

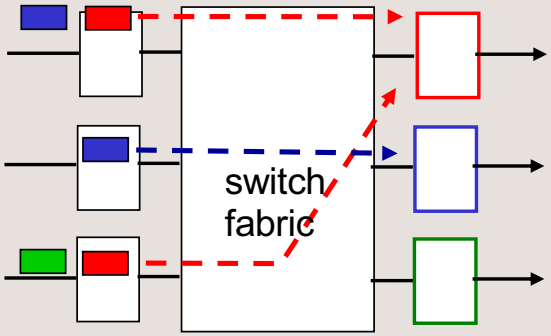



Example: What is the maximum number of packets that can be forwarded from input to output if each packet consists of 1500 bytes? (assume a Cisco 12000 router)

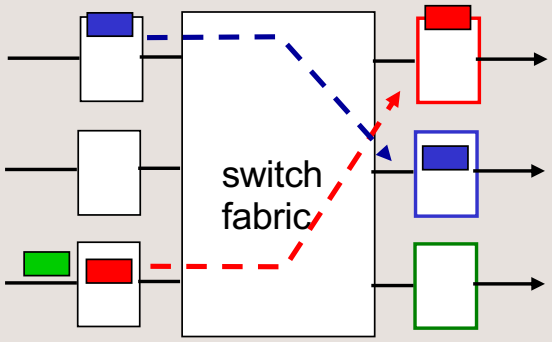


# Input port queuing

- 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



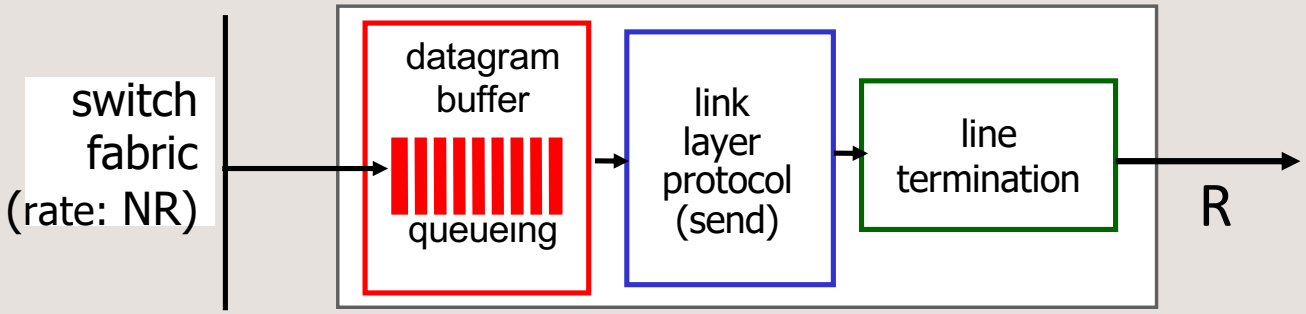

output port contention: only one red datagram can be transferred. lower red packet is *blocked*




one packet time later: green packet experiences HOL blocking



# Output port queuing






This is a really important slide

- *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

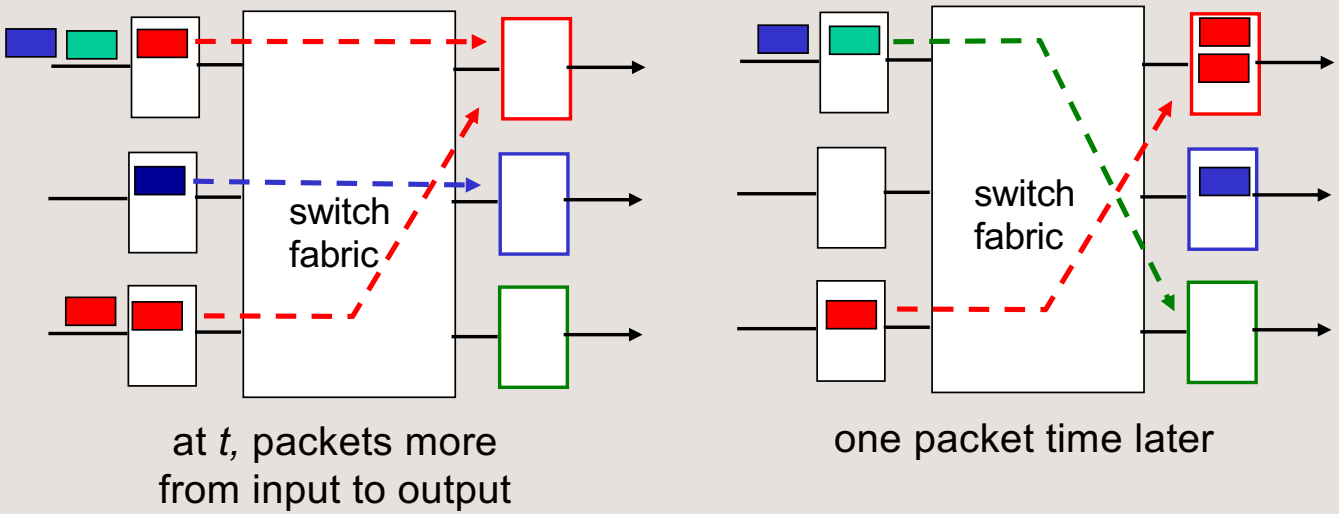
- *Scheduling discipline* chooses among queued datagrams for transmission



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?

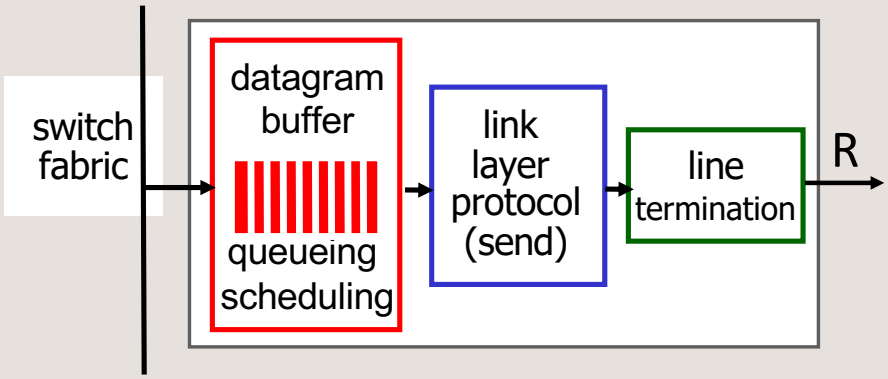
- RFC 3439 rule of thumb: average buffering equal to "typical" RTT (say 250 msec) times link capacity C
  - e.g., C = 10 Gbps link: 2.5 Gbit buffer
- more recent recommendation: with *N* flows, buffering equal to

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

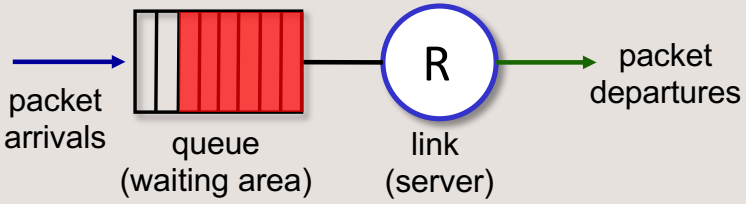
- but *too* much buffering can increase delays (particularly in home routers)
  - long RTTs: poor performance for real-time apps, sluggish TCP response



# Buffer Management




Abstraction: queue




buffer management:

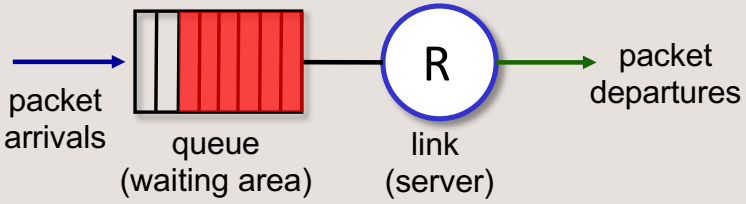
- drop: which packet to drop when buffers are full
  - tail drop: drop arriving packet
  - priority: drop/remove on priority basis
- marking: which packets to mark to signal congestion before buffer gets full (ECN, RED)
  - Early Congestion Notification
  - Random Early Detection

# Packet Scheduling: FCFS

packet scheduling: deciding which packet to send next on link

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

Abstraction: queue




FCFS: packets transmitted in order of arrival to output port

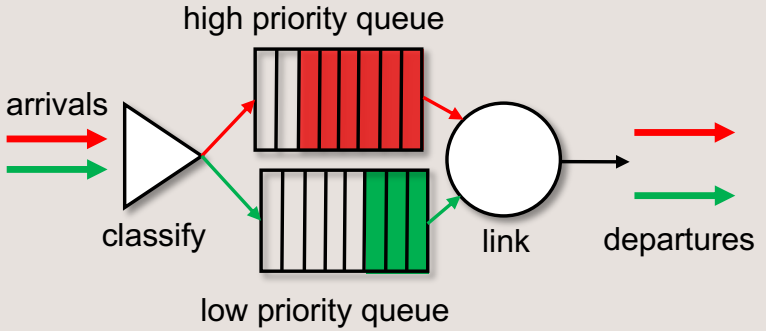
- also known as: First-in-first-out (FIFO)
- real world examples?

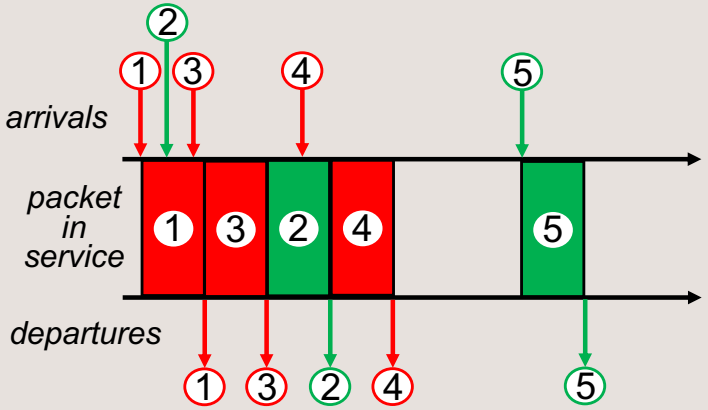


# Scheduling policies: priority

*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
- As long as there are packets in the highest-priority queue, those packets are transmitted first
- Example: Real-time voice-over-IP (VoIP) packets have higher priority than packets associated with email



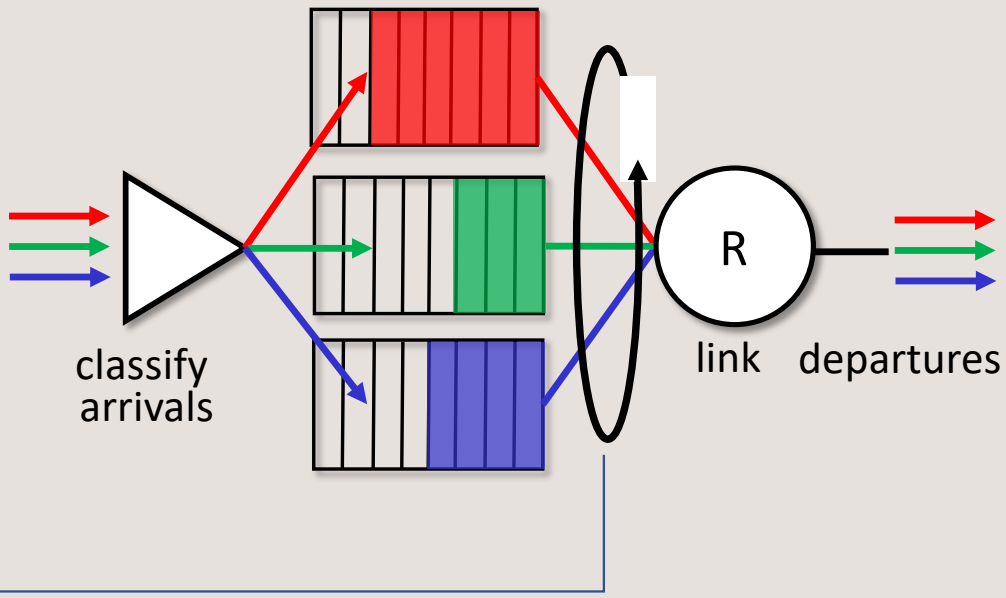






# Scheduling policies: round robin

*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





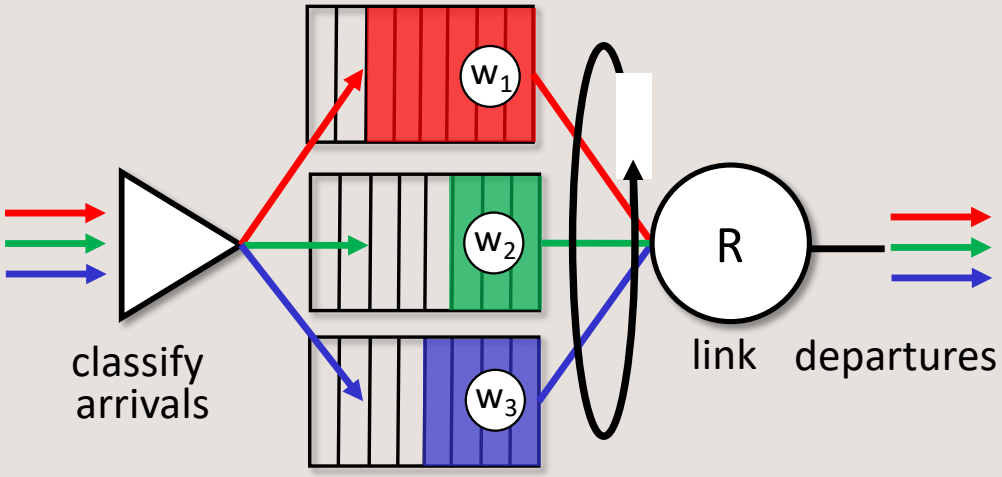

# Scheduling policies: weighted fair queueing

*Weighted Fair Queuing (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*



# On network neutrality

- Net neutrality: The principle that ISPs (Internet Service Providers) must treat all Internet traffic equally, without discrimination.
- ISPs must not block, throttle (deliberately slow down), or charge more for Internet traffic originating from certain websites or applications.
- Examples of net neutrality violations:
  - Comcast deliberately slowed down traffic from P2P file-sharing services.
  - AT&T blocked the FaceTime application for customers who did not have a specific data plan.
  - Verizon deliberately slowed down video streaming from Netflix and YouTube.
  - In many Arab countries, Skype was blocked in the past.



# 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"

2017: rollback of 2015 order

2024: rollback of 2017 rollback



# Network layer: “data plane” roadmap

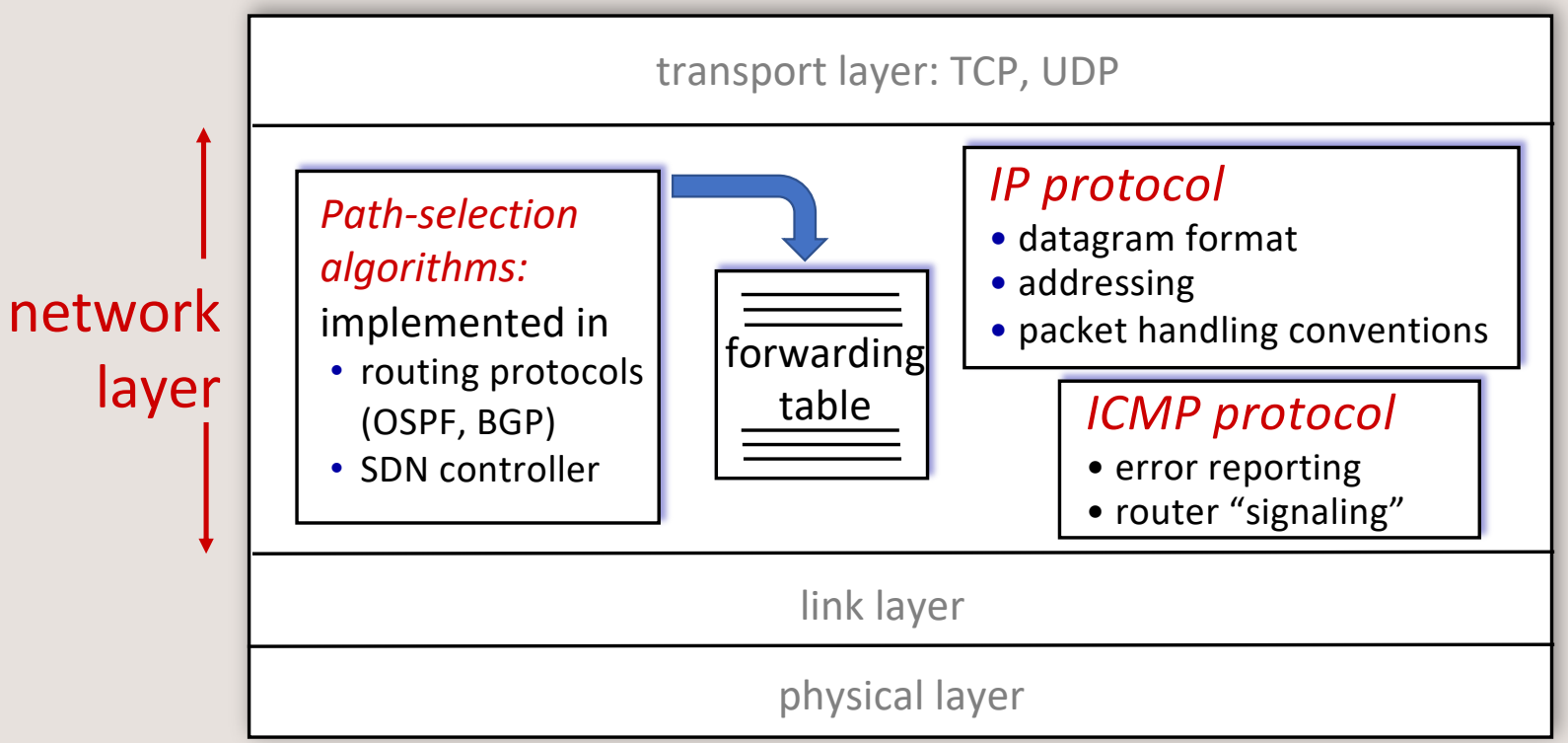
- Network layer: overview
  - data plane
  - control plane
- What’s inside a router
  - input/output ports, switching
  - buffer management
  - scheduling
- IP: the Internet Protocol
  - datagram format, addressing
  - network address translation (NAT)
  - IPv6



- Generalized Forwarding
  - Match+action
  - OpenFlow
- Internet architecture

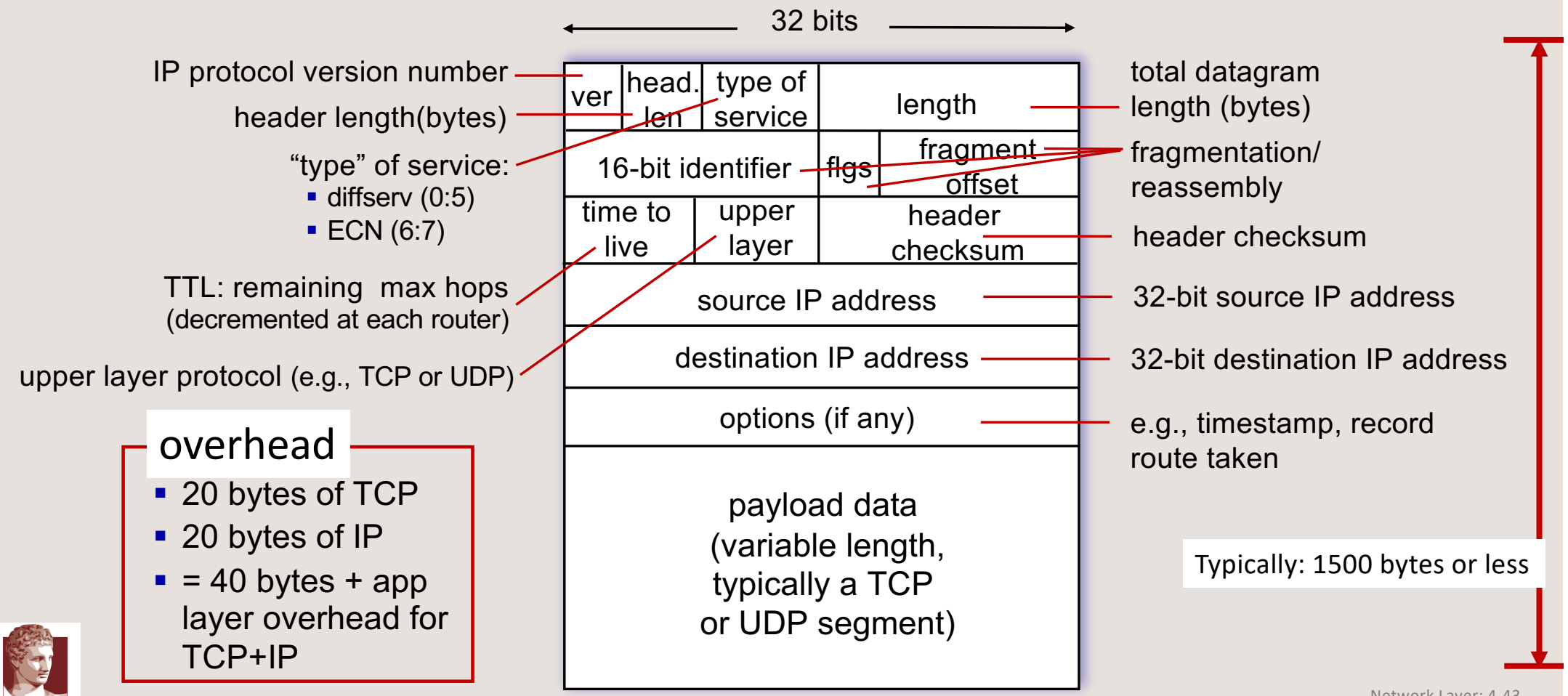
# Network Layer: Internet

host, router network layer functions:






# IP Datagram format

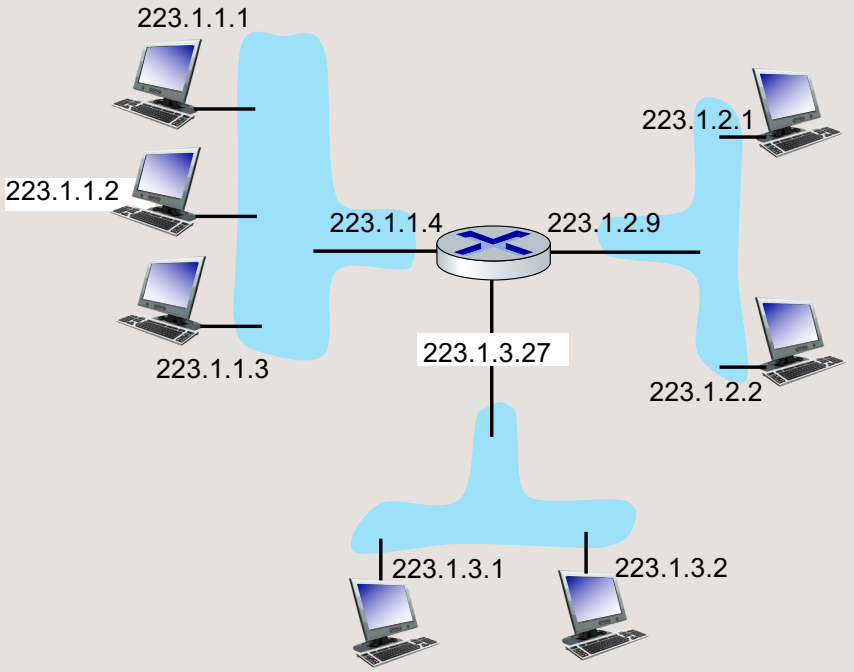



## overhead

- 20 bytes of TCP
- 20 bytes of IP
- = 40 bytes + app layer overhead for TCP+IP

# 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:

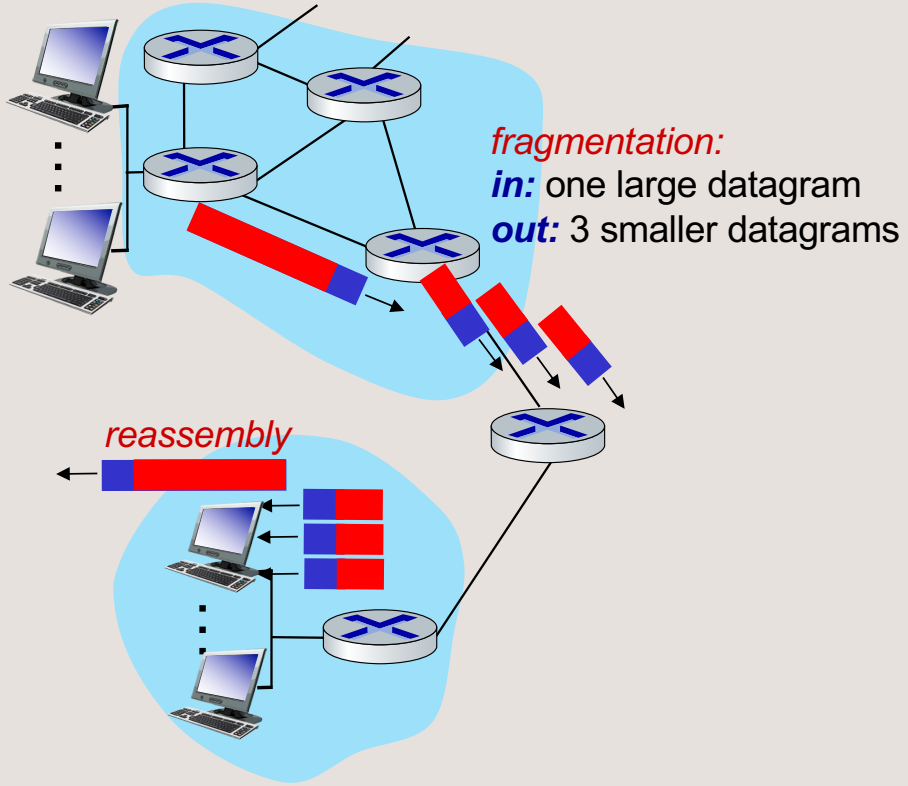
223.1.1.1 = 11011111 00000001 00000001 00000001

223 1 1 1



# IP fragmentation/reassembly

- network links have MTU (max. transfer size) - largest possible link-level frame
  - different link types, different MTUs
- large IP datagram divided ("fragmented") within net
  - one datagram becomes several datagrams
  - "reassembled" only at *destination*
  - IP header bits used to identify, order related fragments

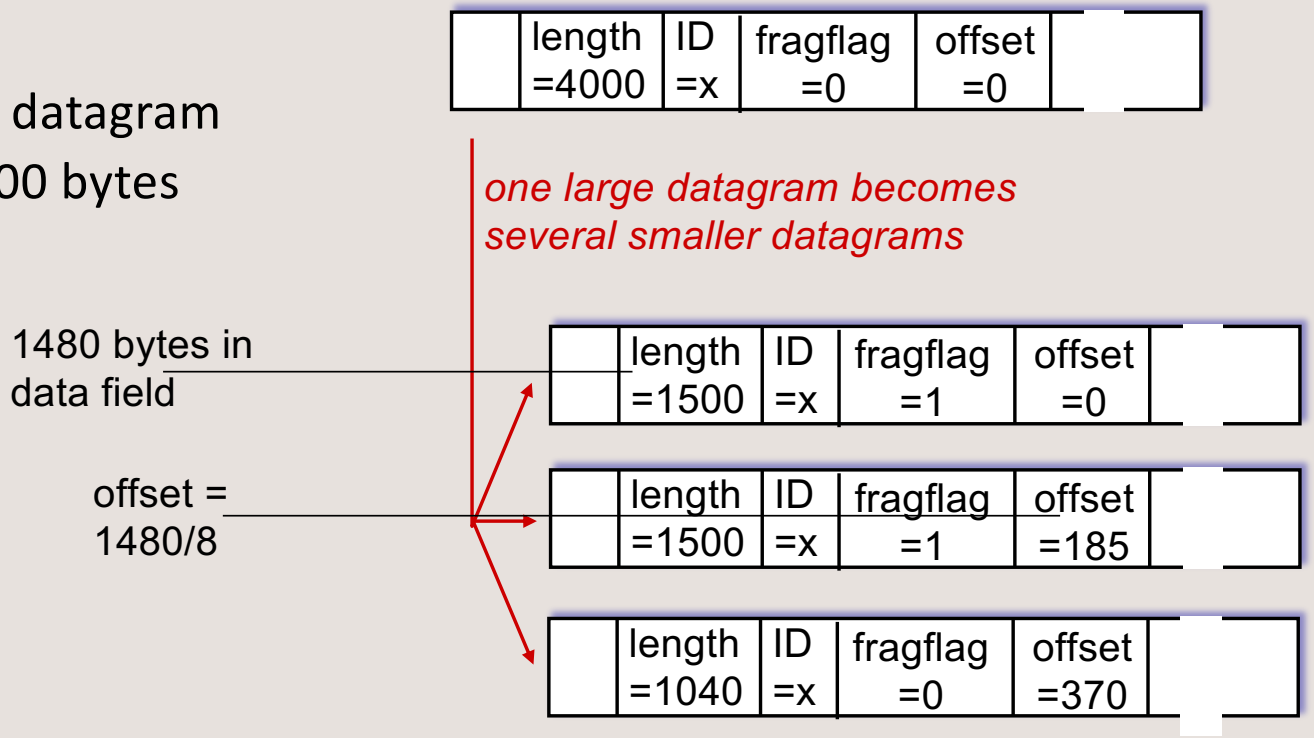





# IP fragmentation/reassembly

example:

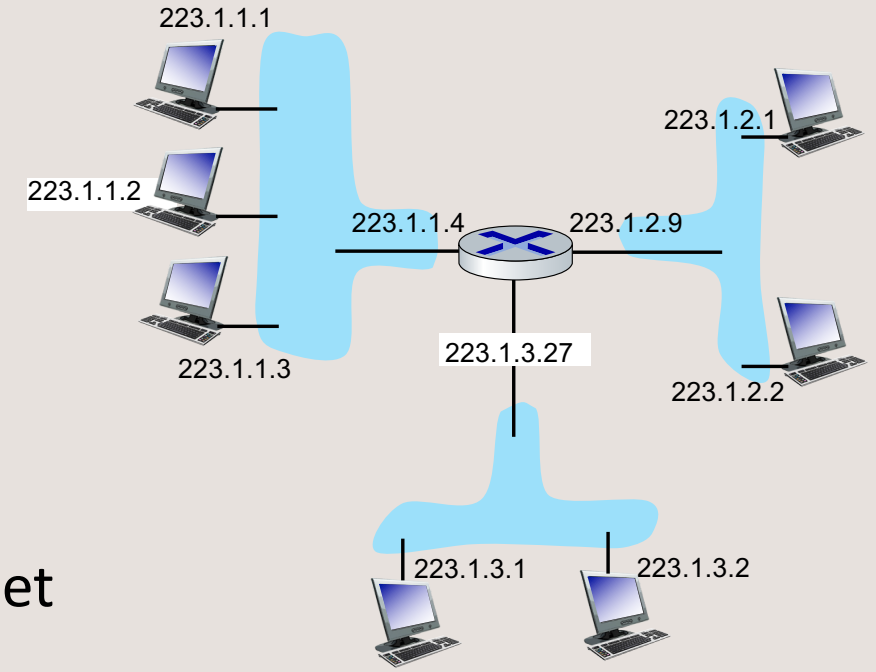
- 4000 byte datagram
- MTU = 1500 bytes






# 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



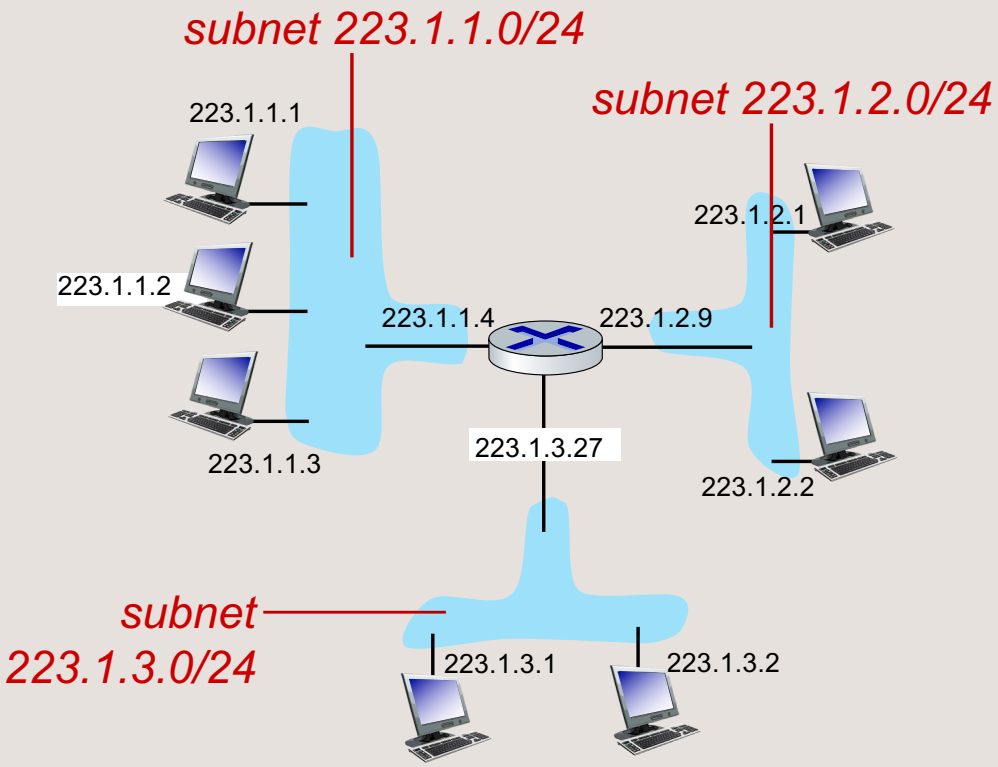

network consisting of 3 subnets



# 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*

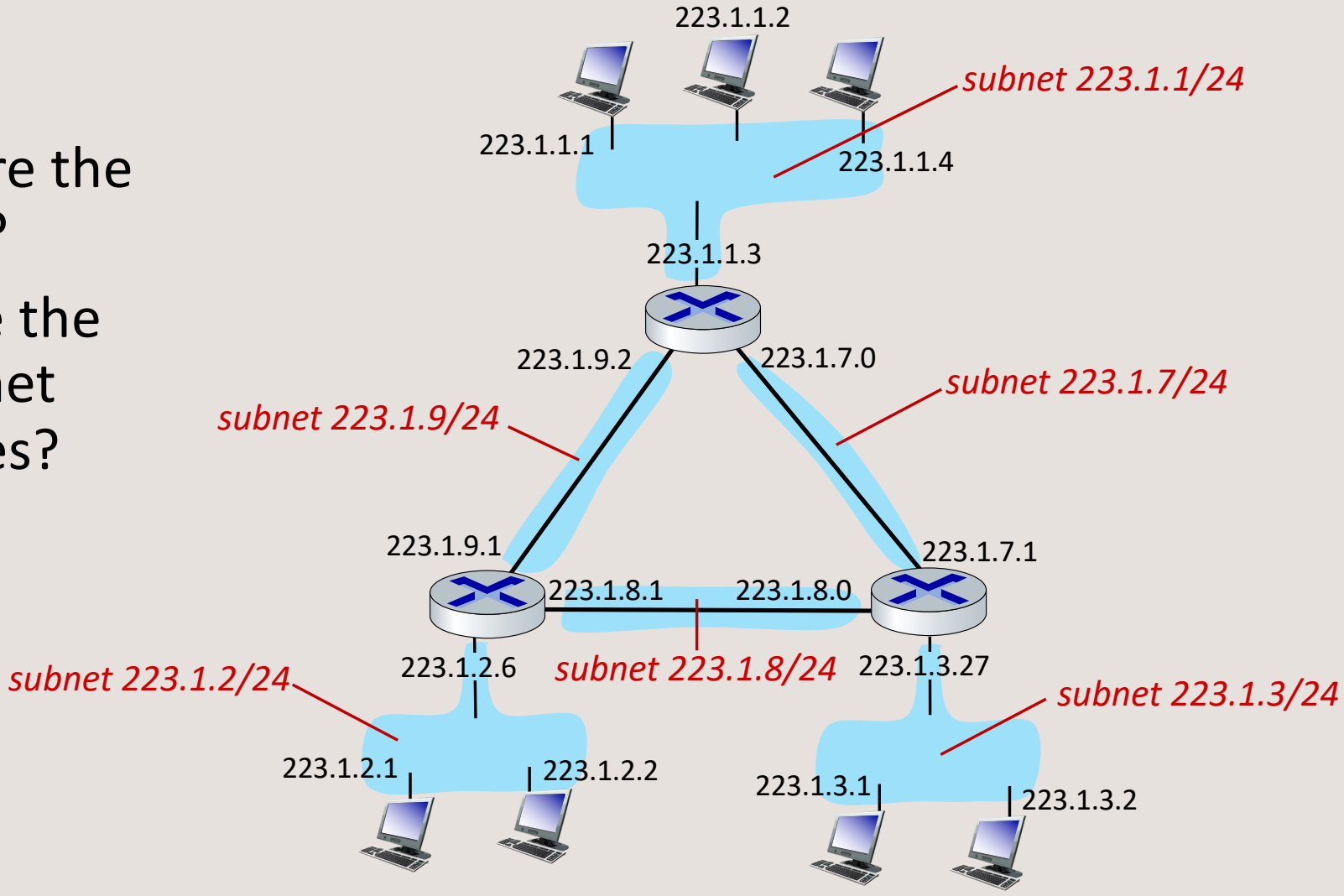



subnet mask: /24
(high-order 24 bits: subnet part of IP address)



# 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




200.23.16.0/23



# IP addresses: how to get one?

That's actually two questions:

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

How does *host* get IP address?

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



# DHCP: Dynamic Host Configuration Protocol

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

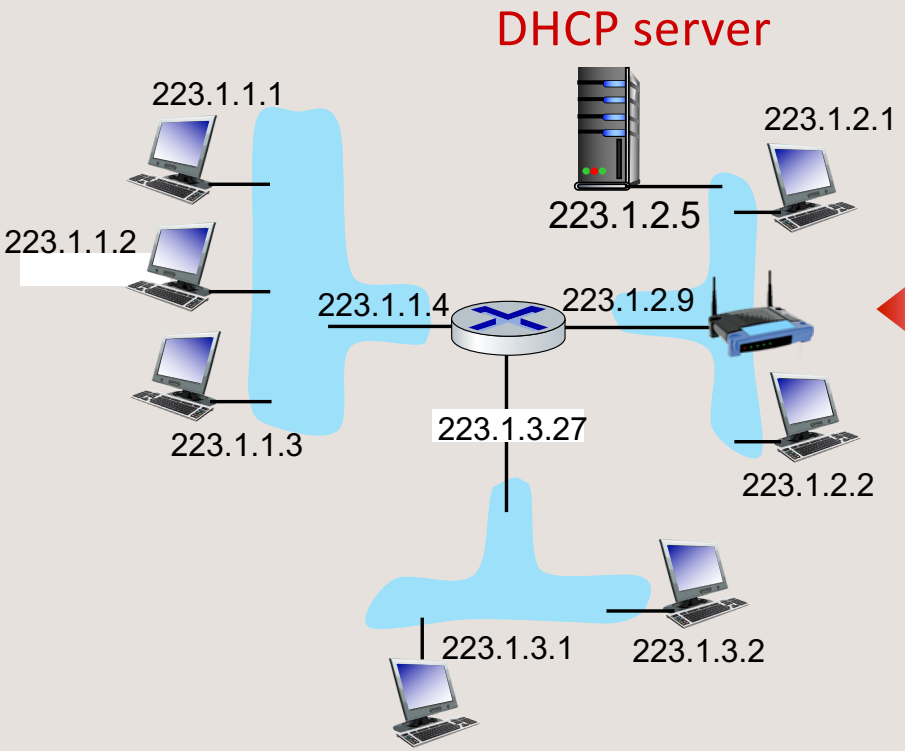
- can renew its lease on address in use
- allows reuse of addresses (only hold address while connected/on)
- support for mobile users who join/leave network

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



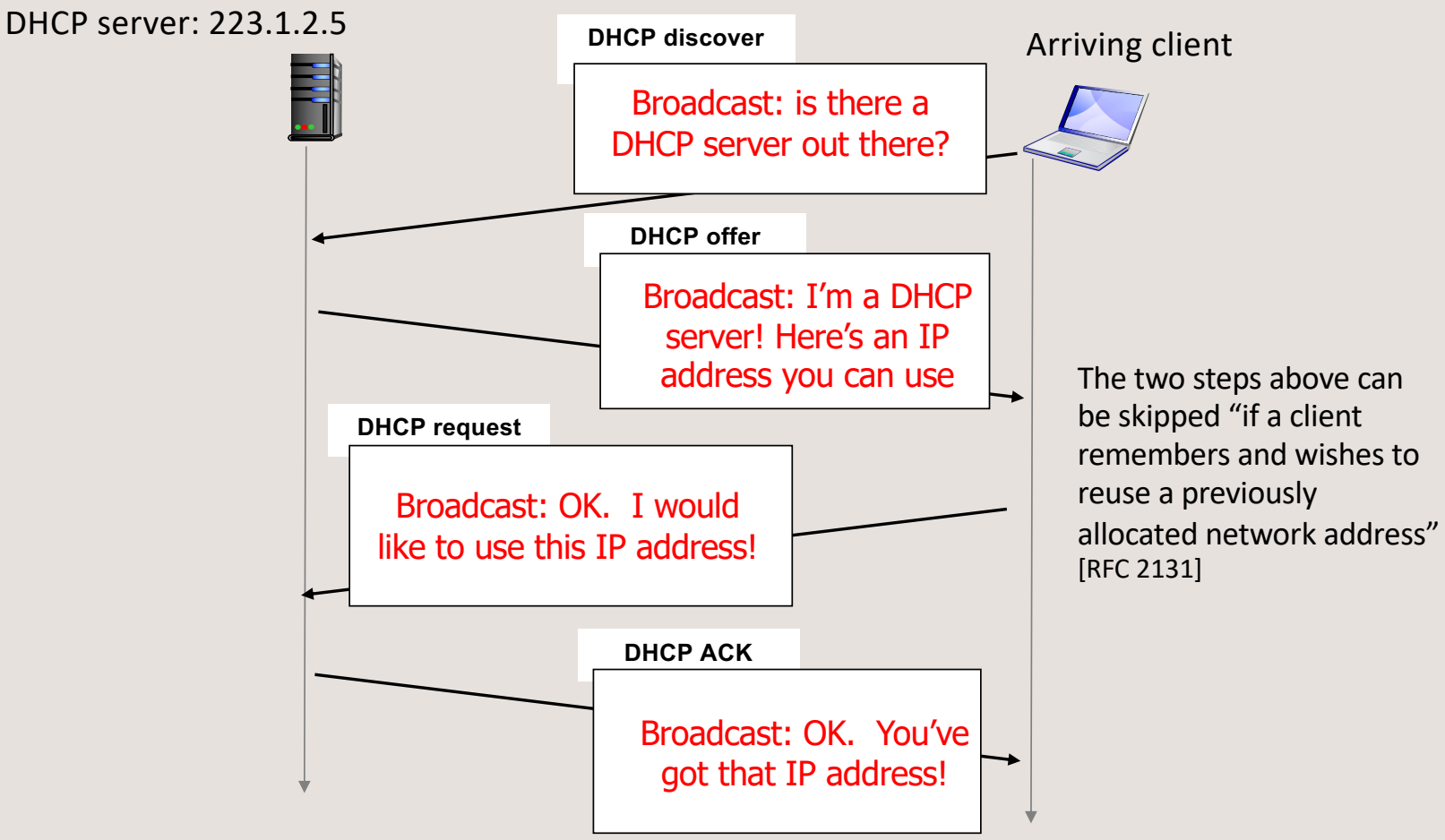

Typically, DHCP server will be co-located in router, serving all subnets to which router is attached



arriving DHCP client needs address in this network



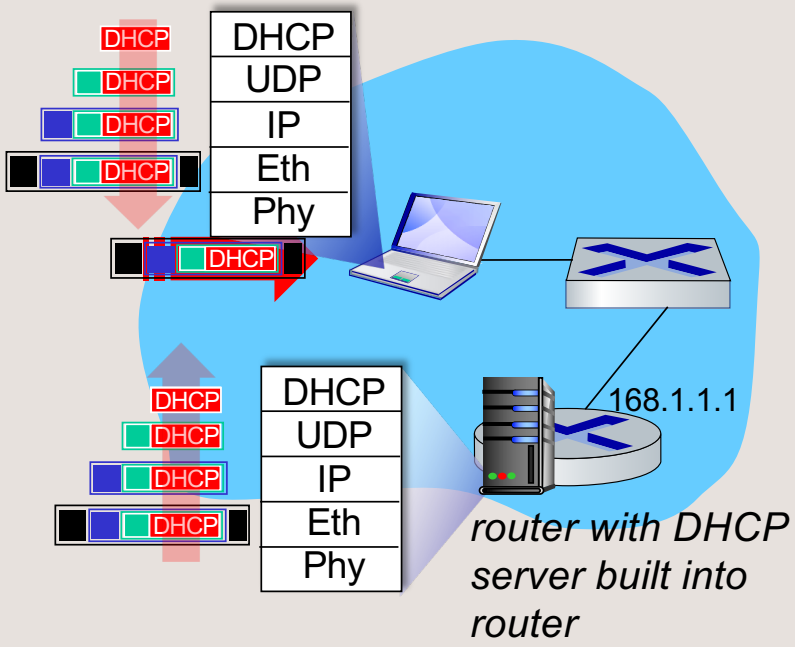
# DHCP client-server scenario




The two steps above can be skipped "if a client remembers and wishes to reuse a previously allocated network address" [RFC 2131]



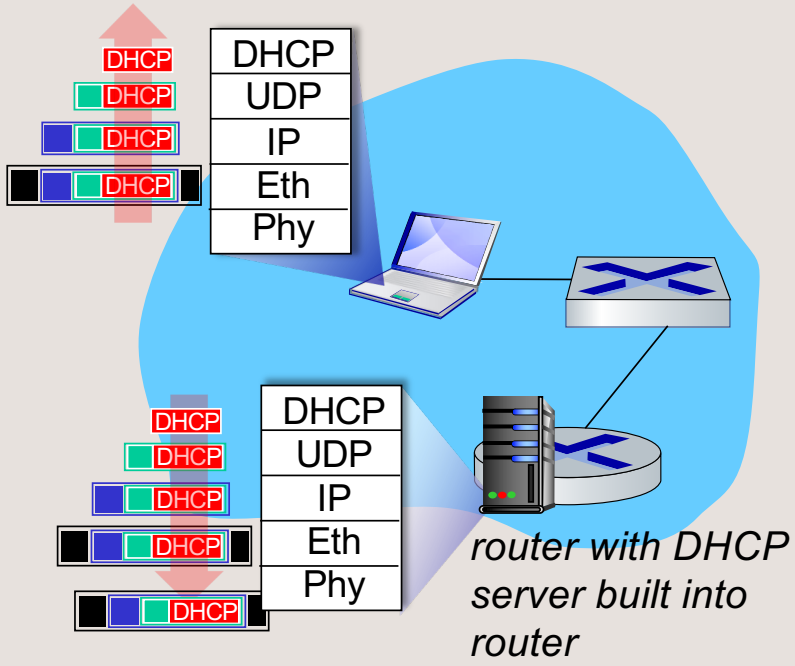
# DHCP: example




- 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 de-mux'ed to IP de-mux'ed, UDP de-mux'ed to DHCP



# DHCP: example (2)




- 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, de-muxing 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 | 11001000 00010111 00010000 | 00000000 | 200.23.16.0/20 |

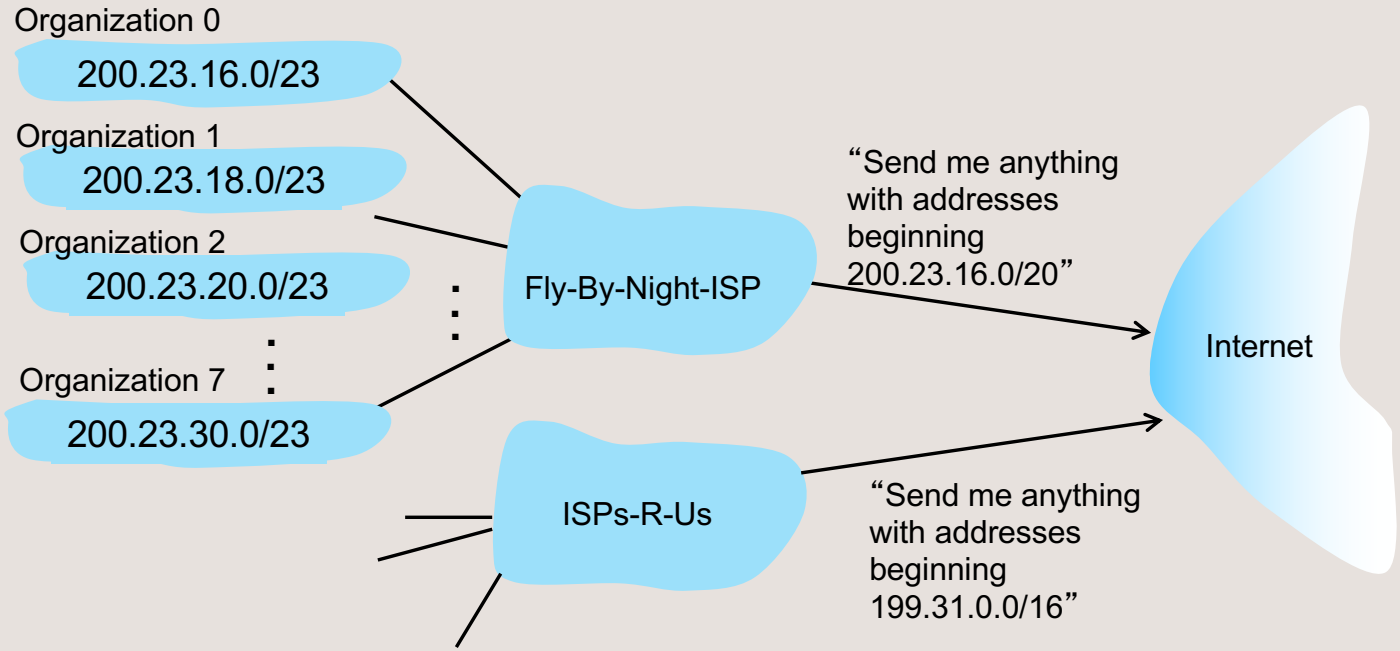
ISP can then allocate out its address space in 8 blocks:

| | | | |
|---|---|---|---|
| Organization 0 | 11001000 00010111 00010000 | 00000000 | 200.23.16.0/23 |
| Organization 1 | 11001000 00010111 00010010 | 00000000 | 200.23.18.0/23 |
| Organization 2 | 11001000 00010111 00010100 | 00000000 | 200.23.20.0/23 |
| ... | ..... | .... | .... |
| Organization 7 | 11001000 00010111 00011110 | 00000000 | 200.23.30.0/23 |



# Hierarchical addressing: route aggregation

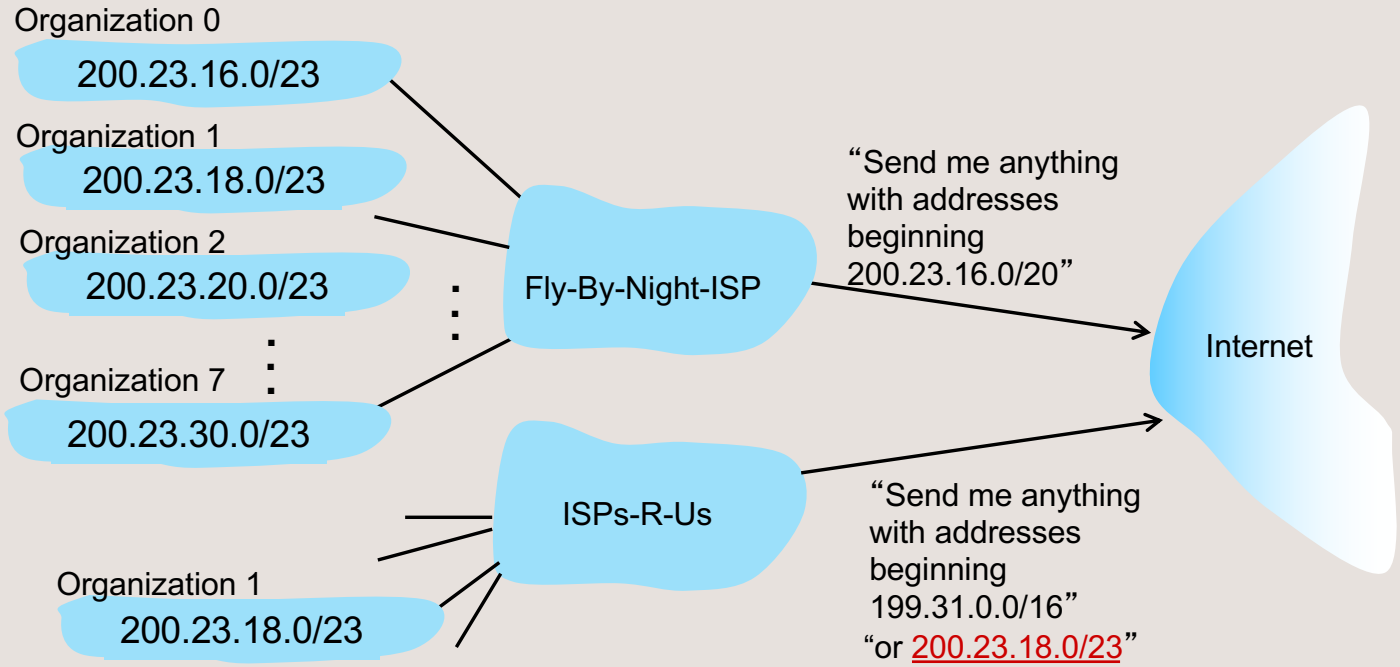
hierarchical addressing allows efficient advertisement of routing information:






# Hierarchical addressing: more specific routes

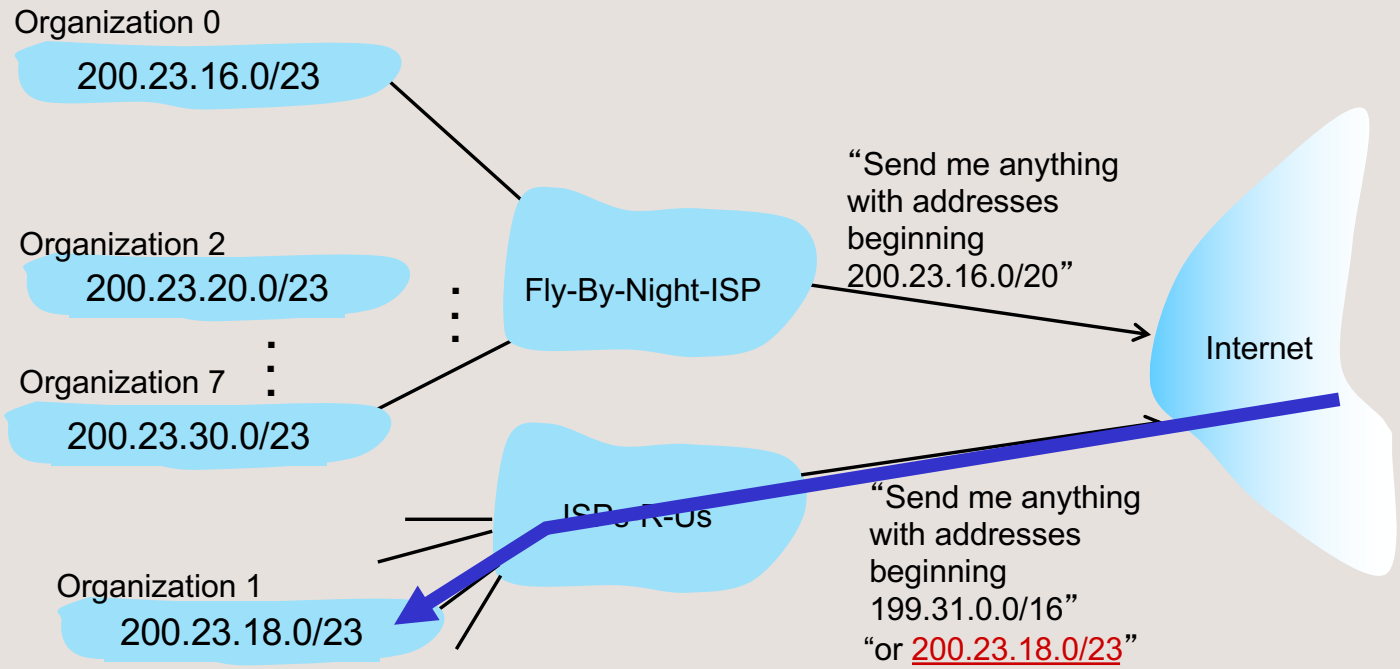
- Organization 1 moves from Fly-By-Night-ISP to ISPs-R-Us
- ISPs-R-Us now advertises a more specific route to Organization 1






# 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



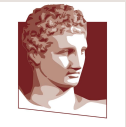



# IP addressing: last words ...

*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 top-level domain (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: “data plane” roadmap

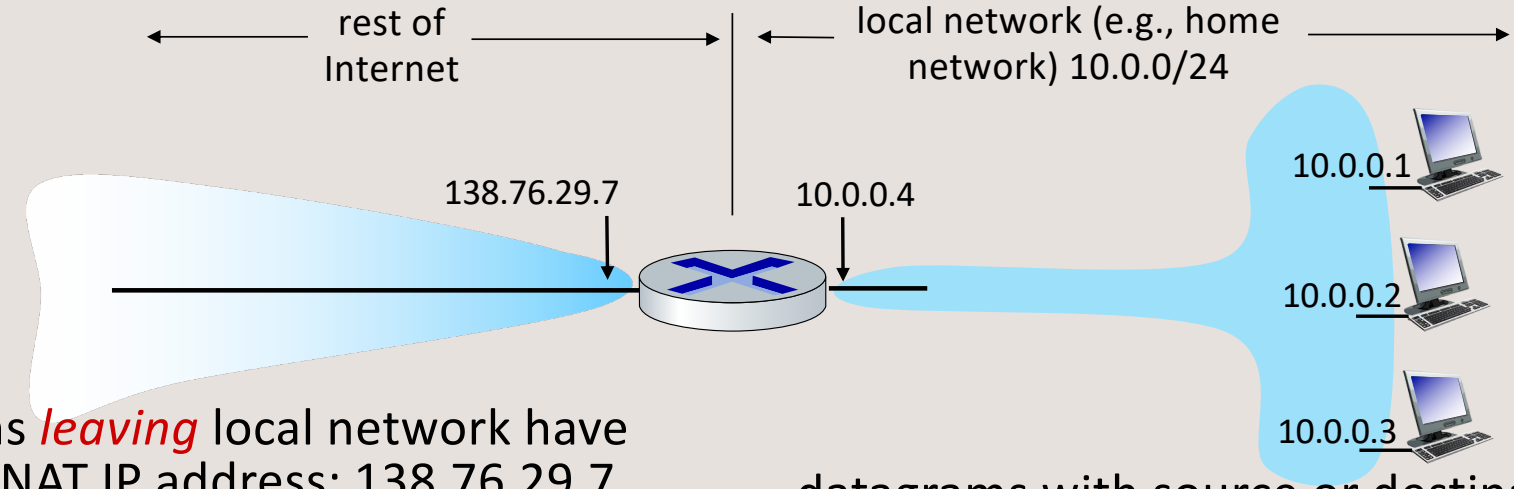
- Network layer: overview
  - data plane
  - control plane
- What’s inside a router
  - input/output ports, switching
  - buffer management
  - scheduling
- IP: the Internet Protocol
  - datagram format, addressing
  - network address translation (NAT)
  - IPv6



- Generalized Forwarding
  - Match+action
  - OpenFlow
- Internet architecture

# 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)



Another local area network (LAN) in a different location can use the same IP addresses 10.0.0.0/24, internally.

# NAT: network address translation

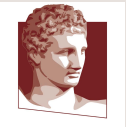
- all devices in local network have 32-bit addresses in a "private" IP address space (10/8, 172.16/12, 192.168/16 prefixes) that can only be used in local network
- advantages:
  - just one IP address needed from provider ISP for *all* devices
  - can change addresses of host in local network without 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



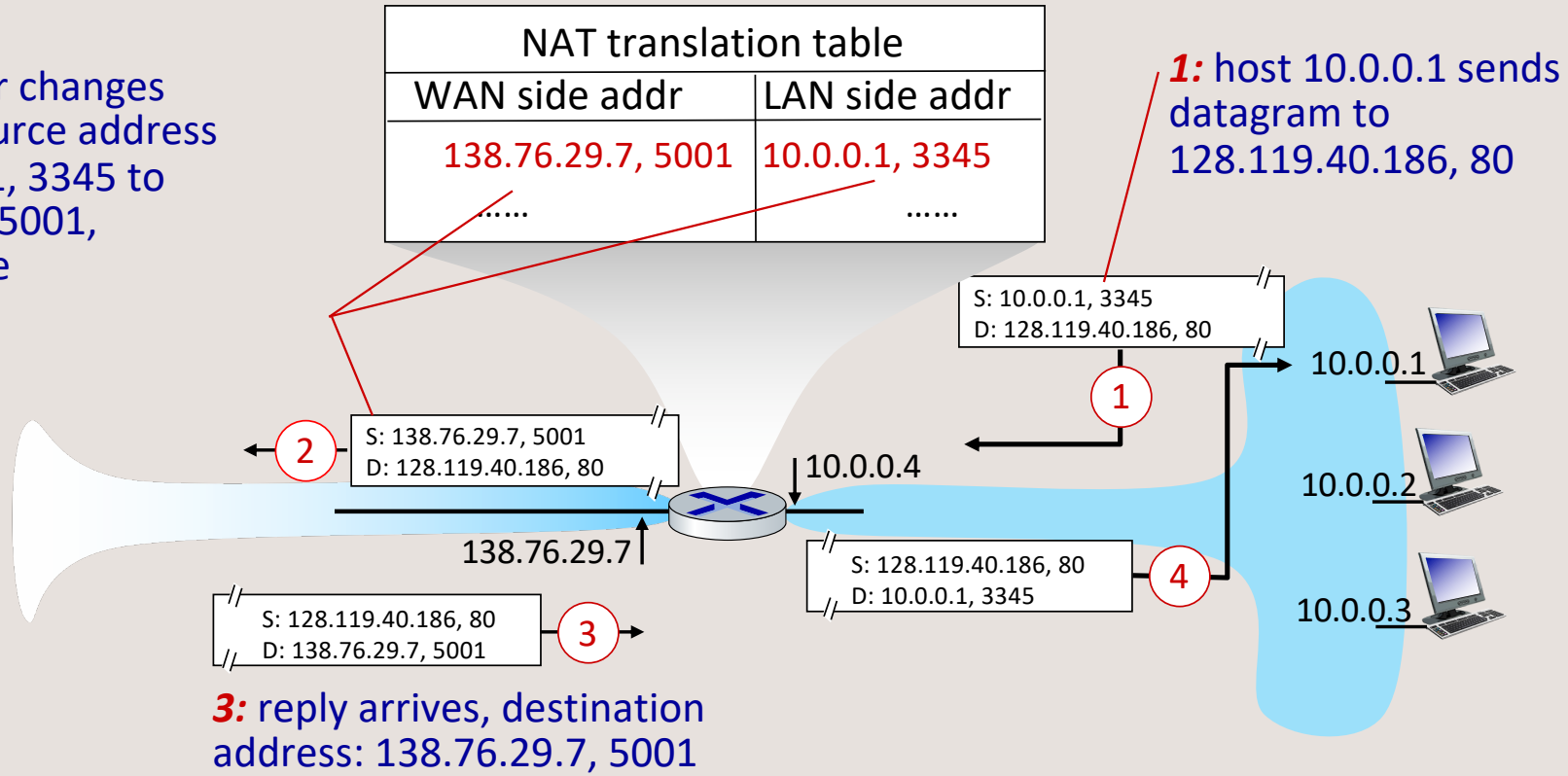
# NAT: network address translation

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






# 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)
    - *"ports are for processes"*
  - 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

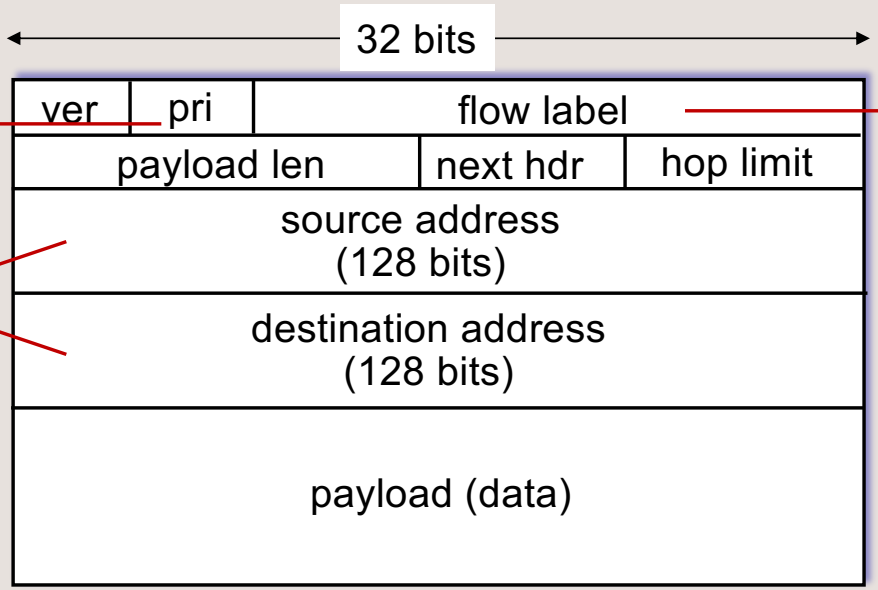
- initial motivation: 32-bit IPv4 address space would be completely allocated
- additional motivation:
  - speed processing/forwarding: 40-byte fixed length header
  - enable different network-layer treatment of “flows”



# 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).

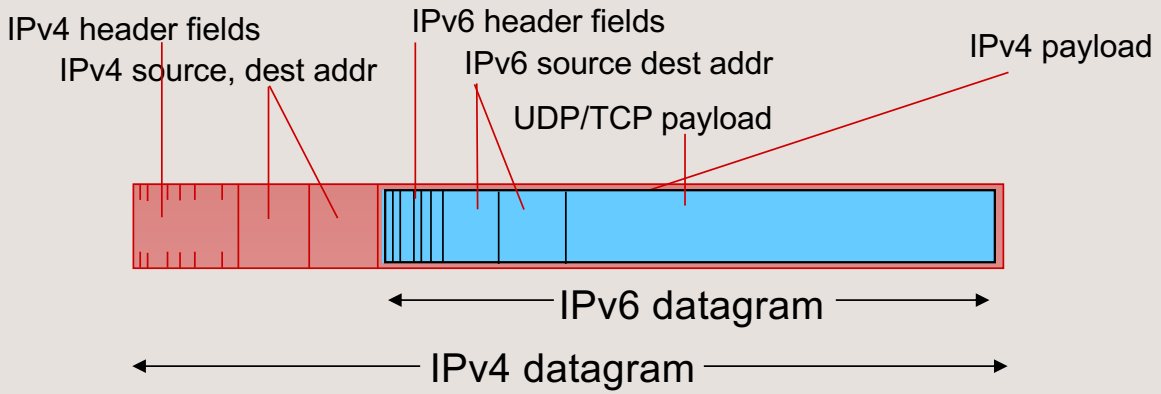
What's missing (compared with IPv4):

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



# Transition from IPv4 to IPv6

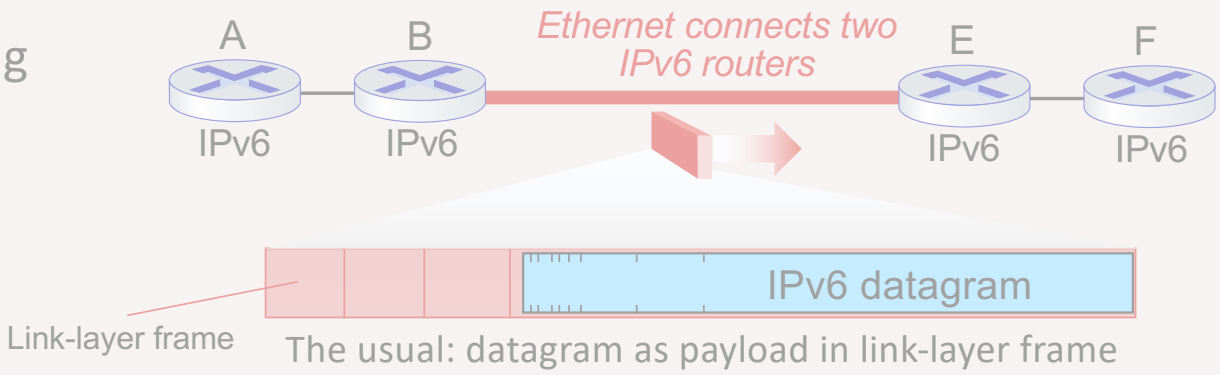
- not all routers can be upgraded simultaneously
  - no “flag days”
  - how will network operate with mixed IPv4 and IPv6 routers?
- tunneling: IPv6 datagram carried as *payload* in IPv4 datagram among IPv4 routers (“packet within a packet”)
  - tunneling used extensively in other contexts (4G/5G)



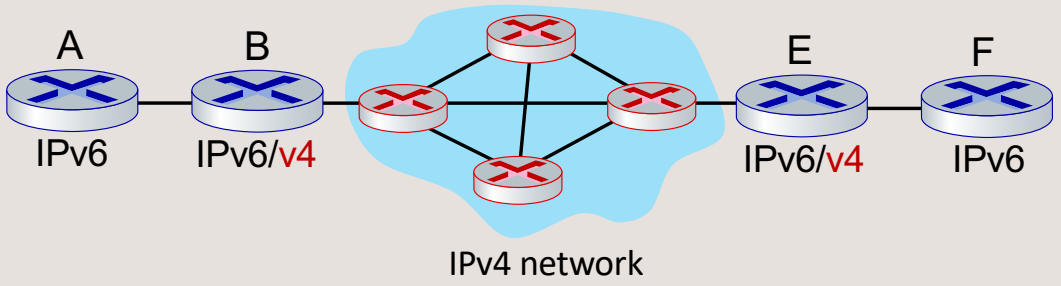



# Tunneling and encapsulation

Ethernet connecting two IPv6 routers:



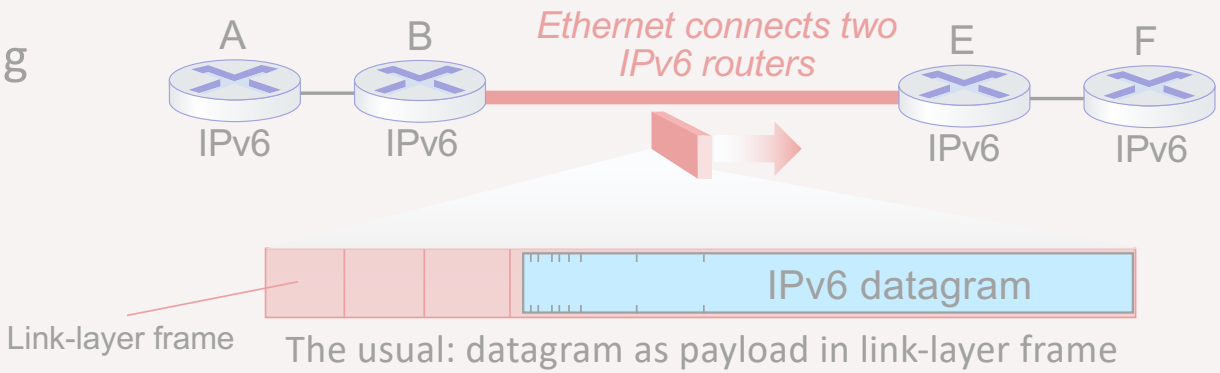

IPv4 network connecting two IPv6 routers



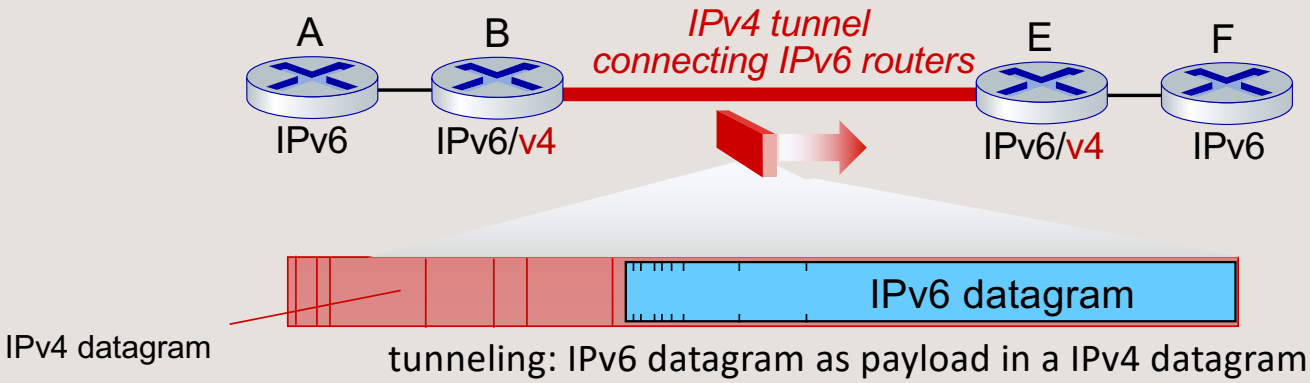



# Tunneling and encapsulation

Ethernet connecting two IPv6 routers:

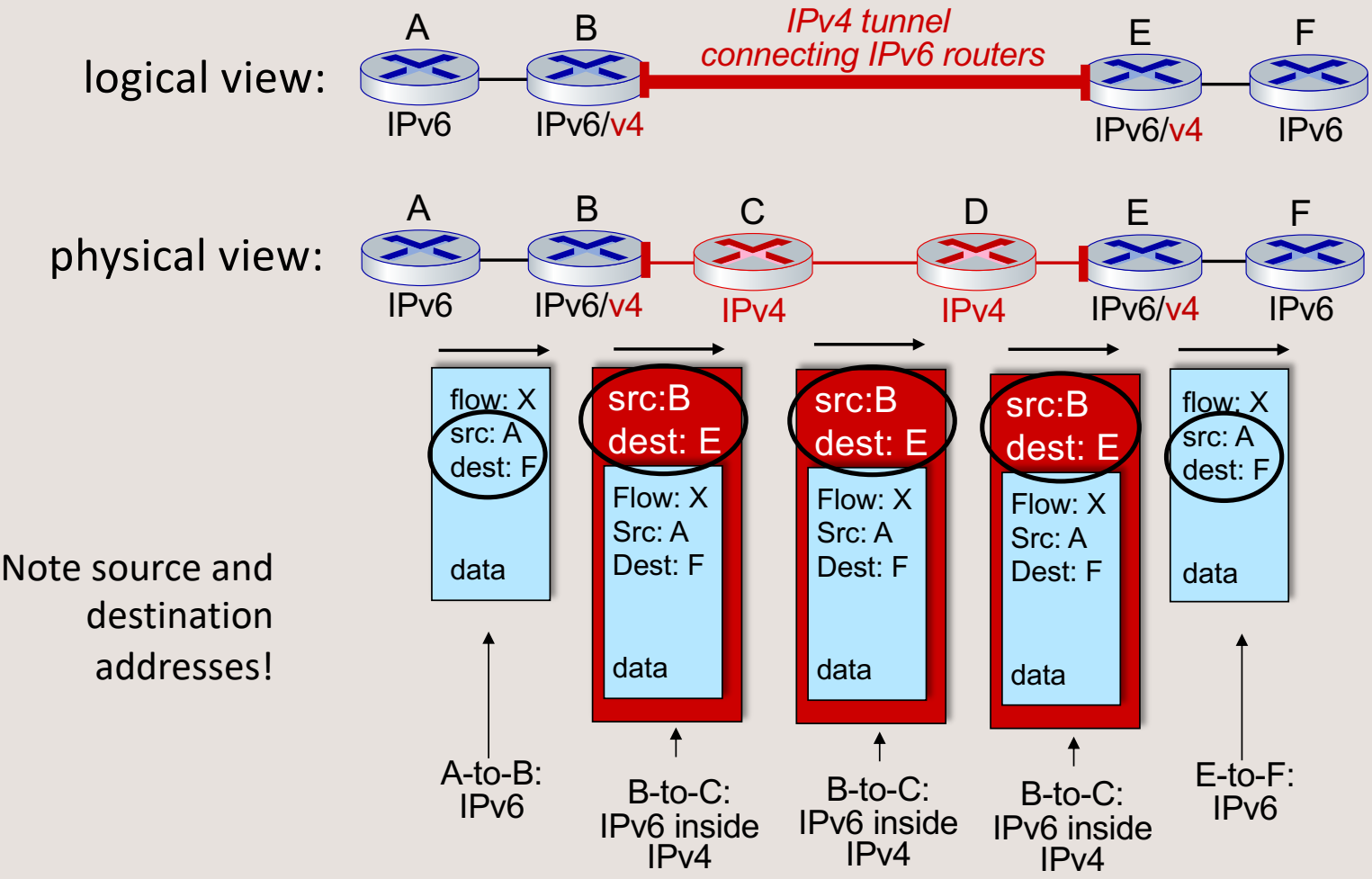



IPv4 tunnel connecting two IPv6 routers





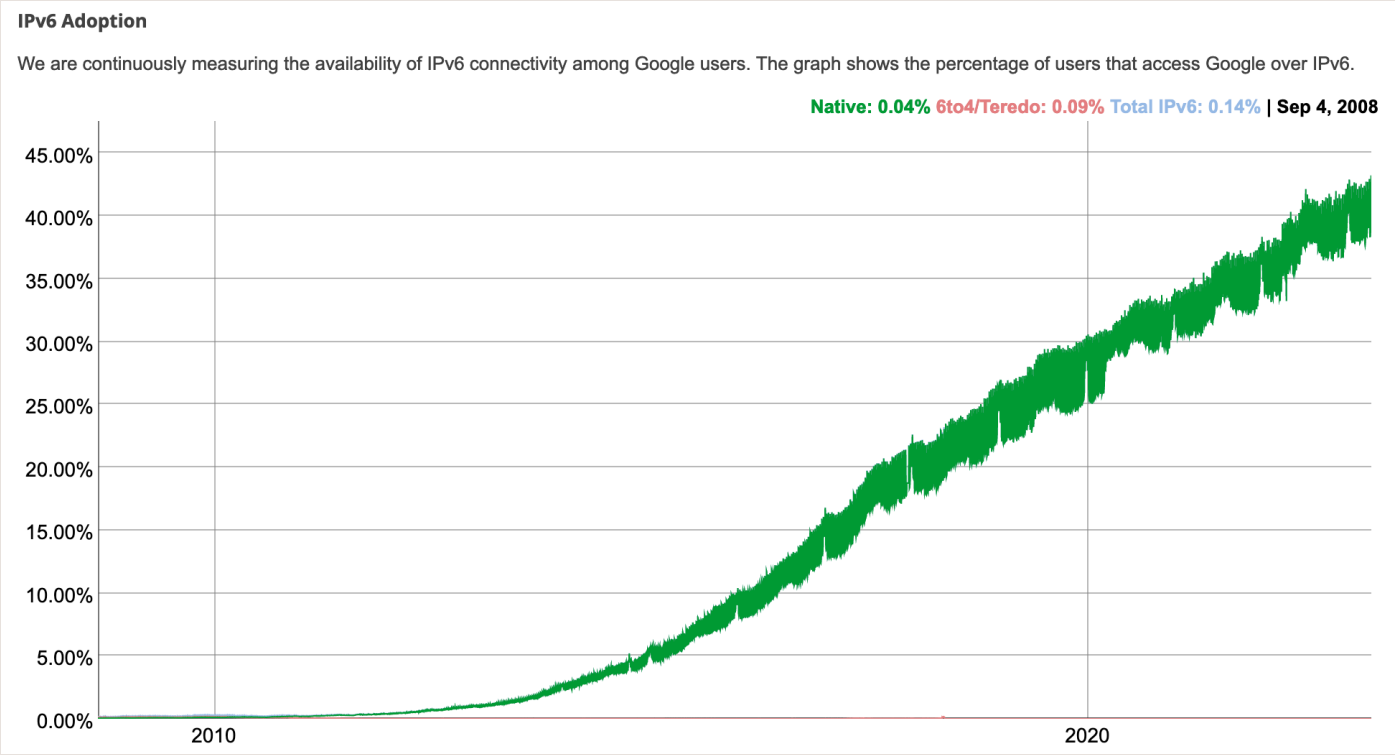

# Tunneling


logical view:
A
B
IPv4 tunnel
connecting IPv6 routers
E
F
IPv6
IPv6/v4
IPv6/v4
IPv6
physical view:
A
B
C
D
E
F
IPv6
IPv6/v4
IPv4
IPv4
IPv6/v4
IPv6
flow: X
src: A
dest: F
data
src:B
dest: E
Flow: X
Src: A
Dest: F
data
src:B
dest: E
Flow: X
Src: A
Dest: F
data
src:B
dest: E
Flow: X
Src: A
Dest: F
data
flow: X
src: A
dest: F
data
Note source and
destination
addresses!
A-to-B:
IPv6
B-to-C:
IPv6 inside
IPv4
B-to-C:
IPv6 inside
IPv4
B-to-C:
IPv6 inside
IPv4
E-to-F:
IPv6



# IPv6: adoption

- Google[1]: ~ 40% of clients access services via IPv6 (2023)
- NIST: 1/3 of all US government domains are IPv6 capable






# IPv6: adoption

- Google[1]: ~ 40% of clients access services via IPv6 (2023)
- 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, …



[1] https://www.google.com/intl/en/ipv6/statistics.html

# Network layer: “data plane” roadmap

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

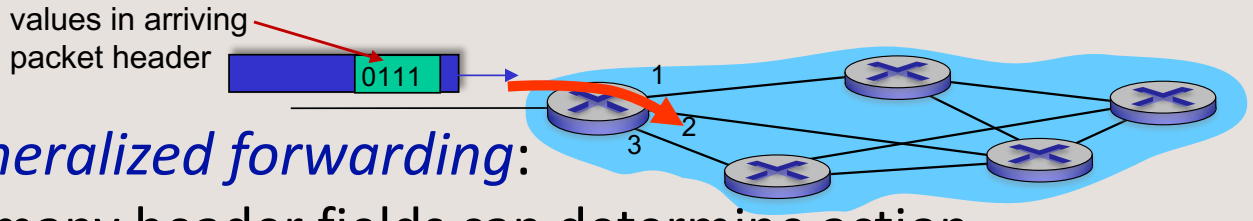


- Generalized Forwarding
  - Match+action
  - OpenFlow
- Internet architecture

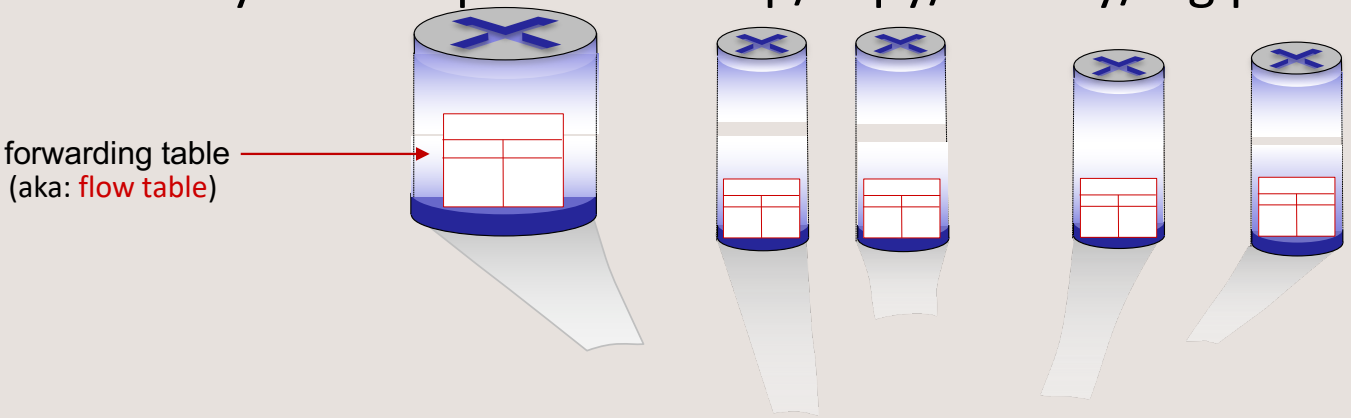
# Generalized forwarding: match plus action

*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

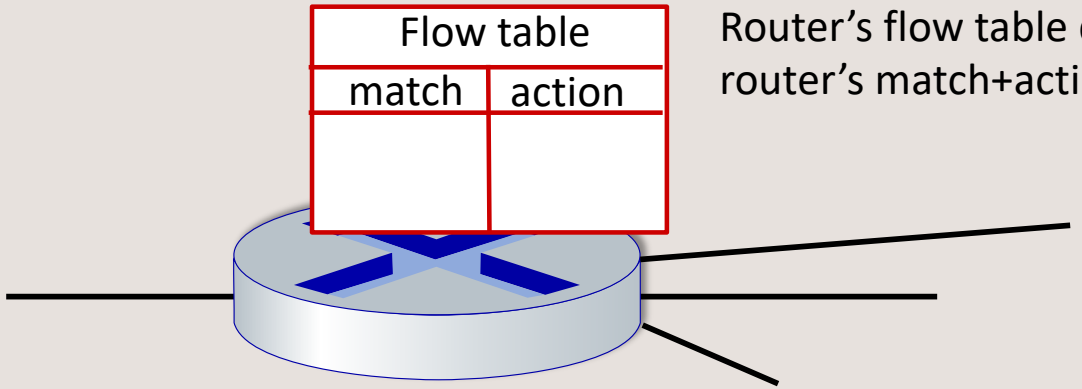



  - *generalized forwarding*:
    - many header fields can determine action
    - many actions possible: drop/copy/modify/log packet



# 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

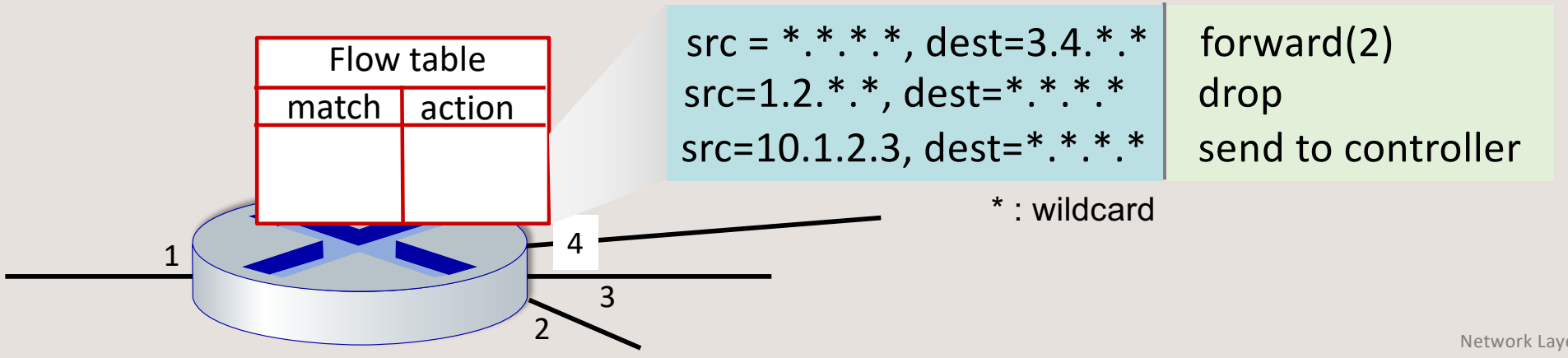



Router's flow table define router's match+action rules



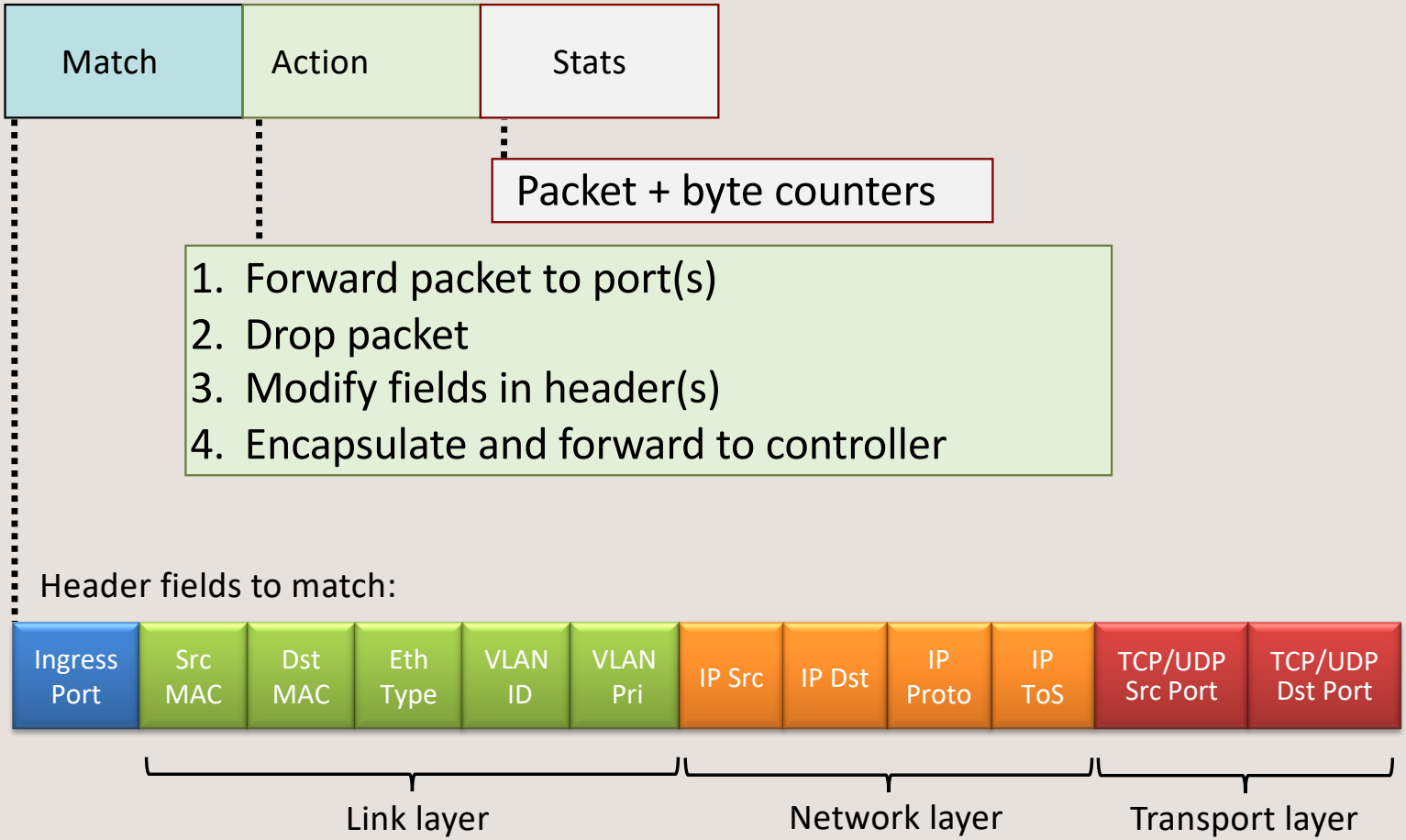
# Flow table abstraction

- 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 |

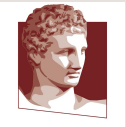
IP datagrams destined to IP address 51.6.0.8 should be forwarded to router output port 6

## 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

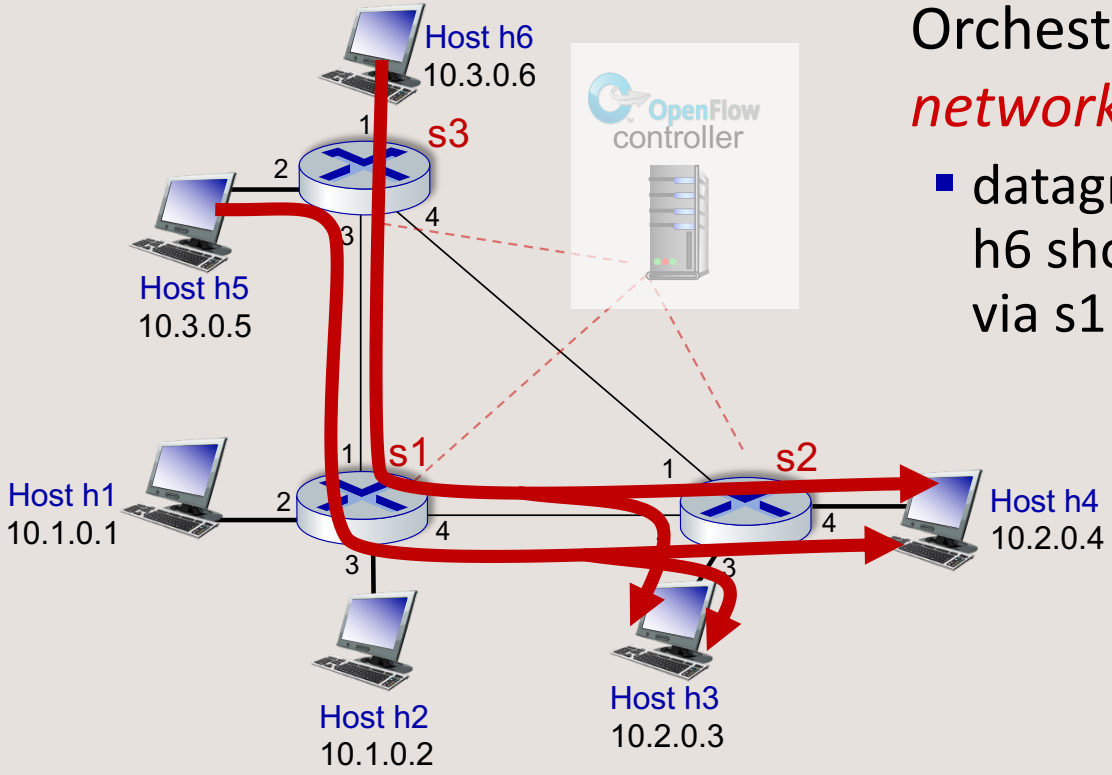
## Layer 2 destination-based forwarding:

| 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 example

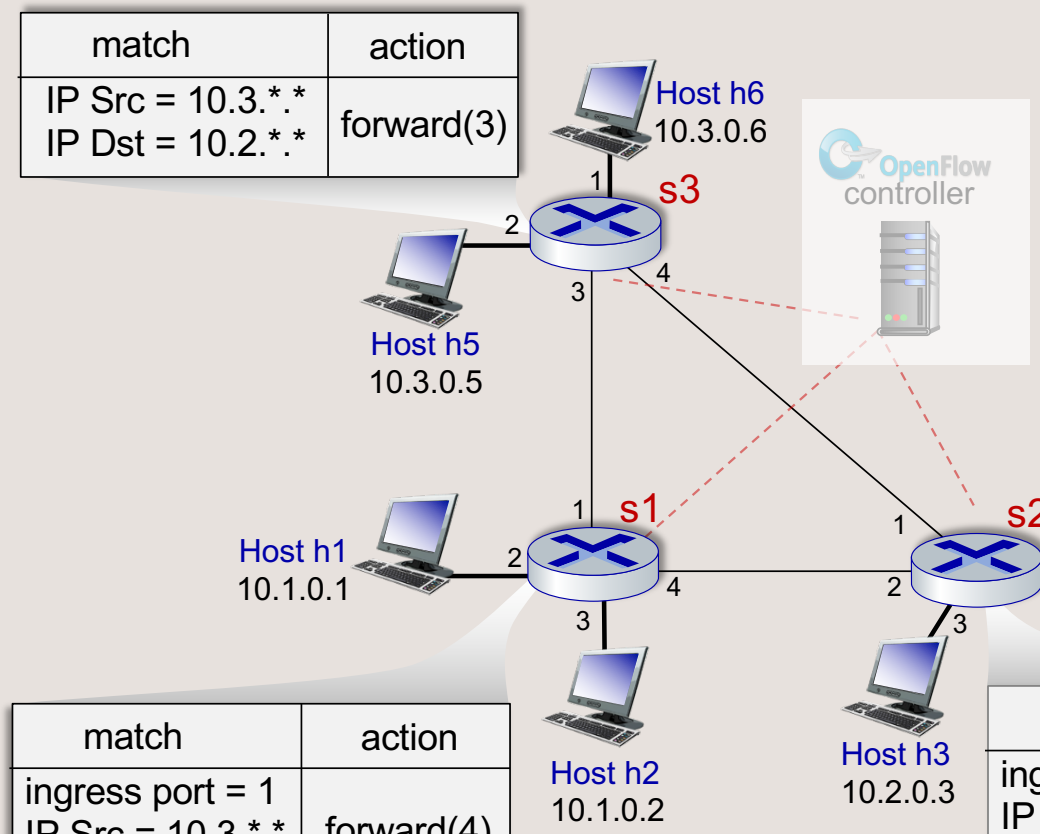



Orchestrated tables can create *network-wide* behavior, e.g.,:

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



# OpenFlow example




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" n*etwork-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: “data plane” roadmap

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

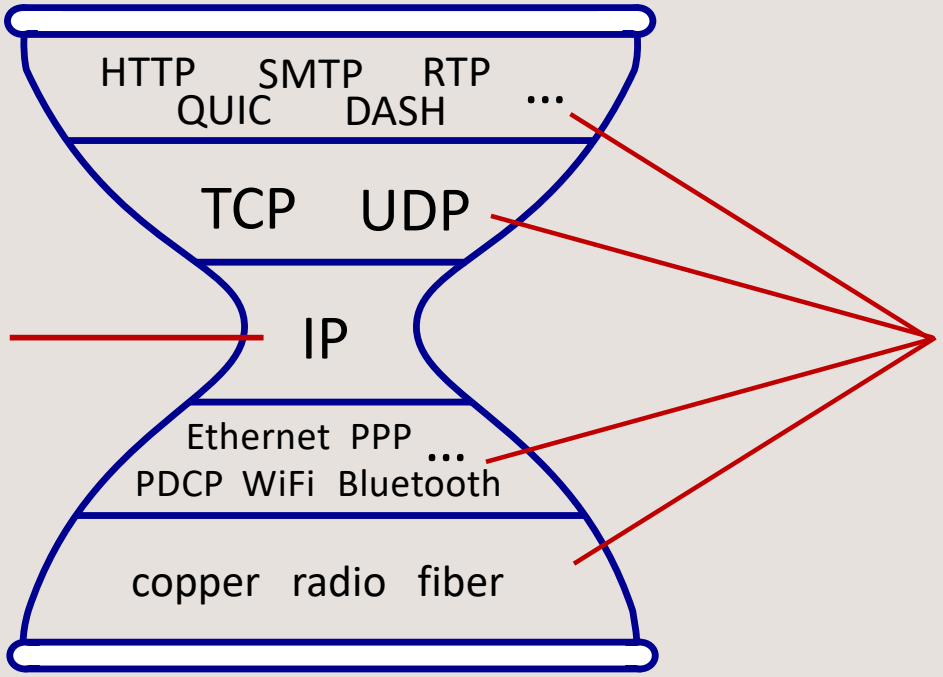


- Generalized Forwarding
  - Match+action
  - OpenFlow
- Internet architecture

# 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

# 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



# Chapter 4: done!

- Network layer: overview
- What's inside a router
- IP: the Internet Protocol
- Generalized Forwarding, SDN



*Question:* how are forwarding tables (destination-based forwarding) or flow tables (generalized forwarding) computed?

*Answer:* by the control plane (next chapter)