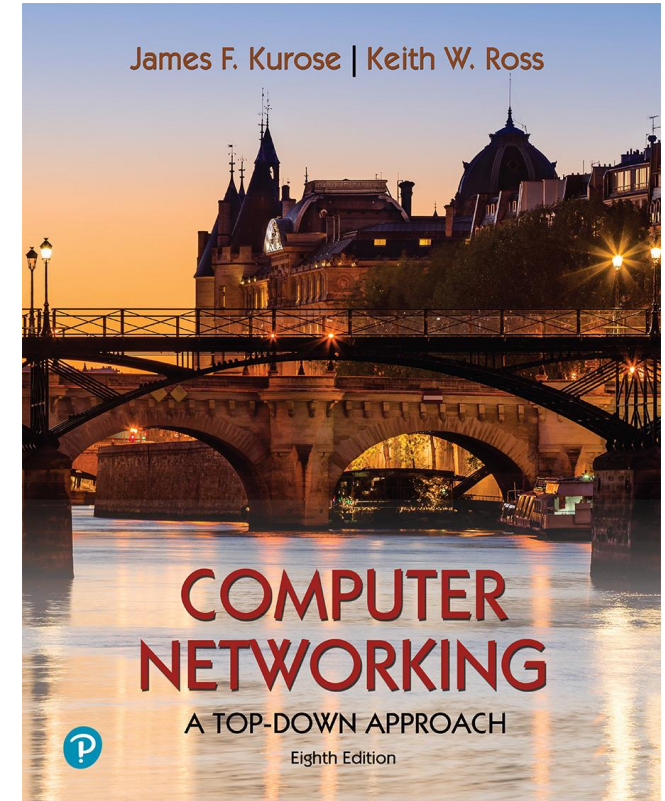


# Chapter 4

## Network Layer: Data Plane



### *Computer Networking: A Top-Down Approach*

8<sup>th</sup> edition

Jim Kurose, Keith Ross  
Pearson, 2020

# 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
- instantiation, implementation in the Internet
  - IP protocol
  - NAT, middleboxes

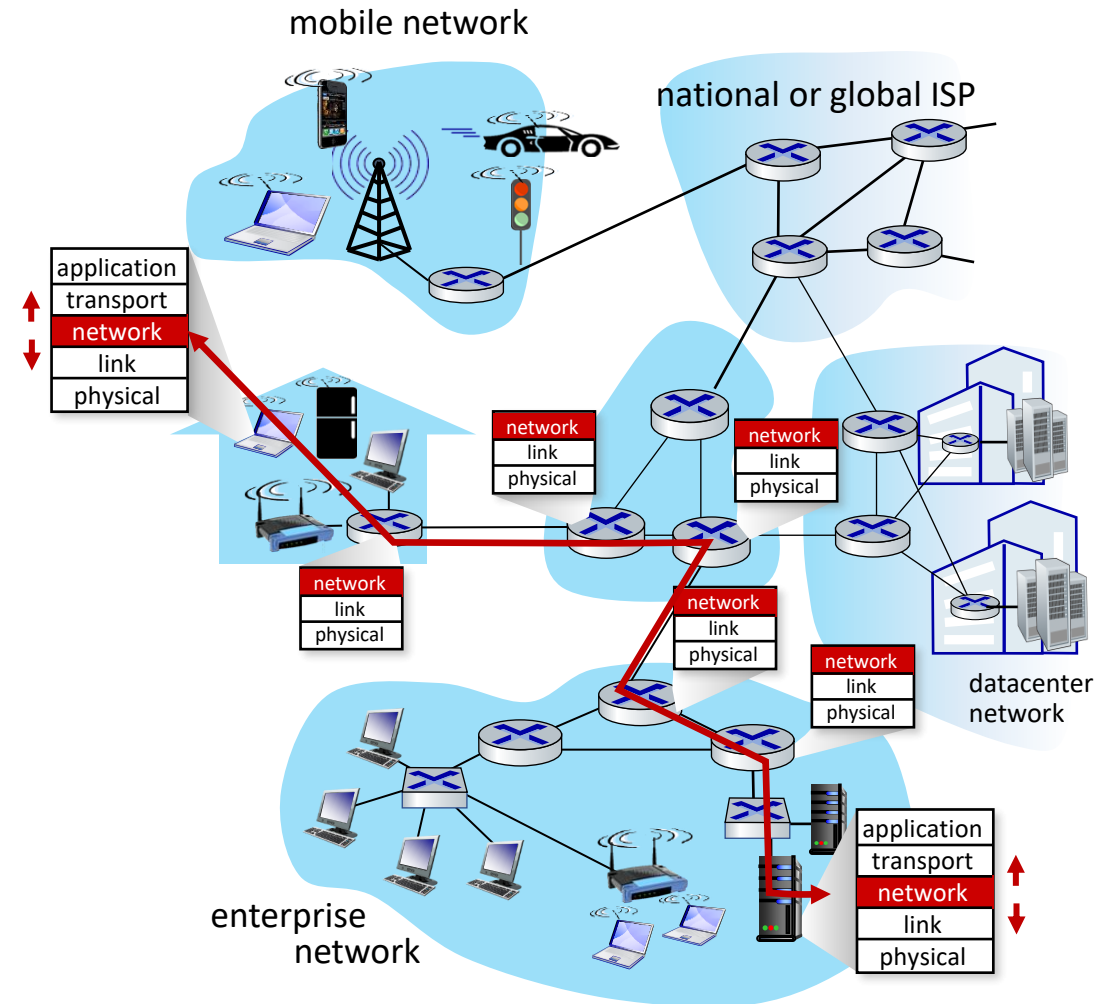
# Network layer: “data plane” roadmap

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



# 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



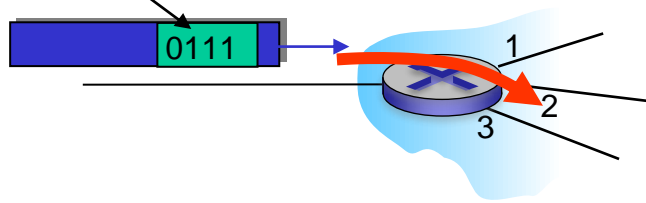
routing

# 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

values in arriving  
packet header



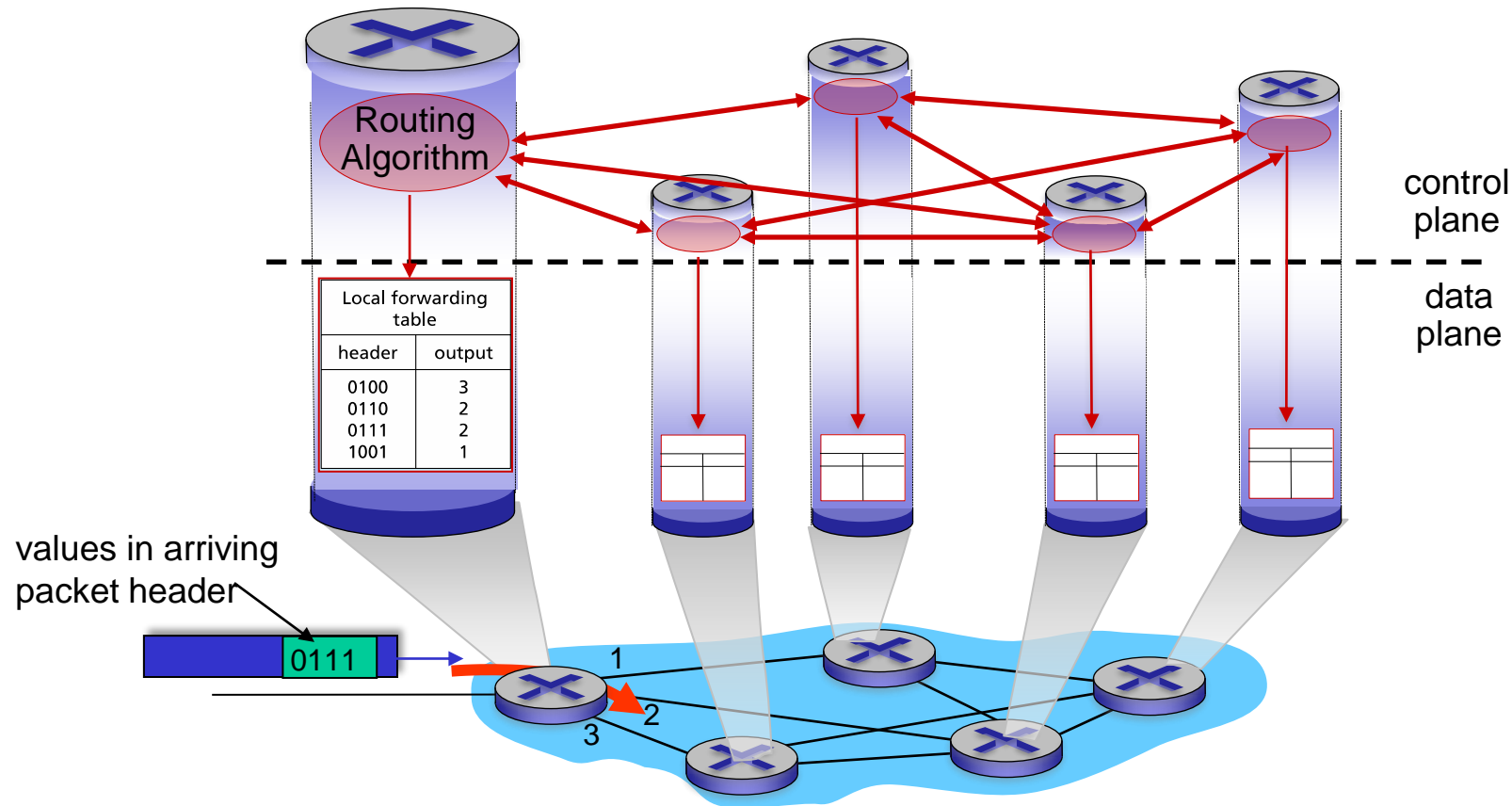
## Control plane

- *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



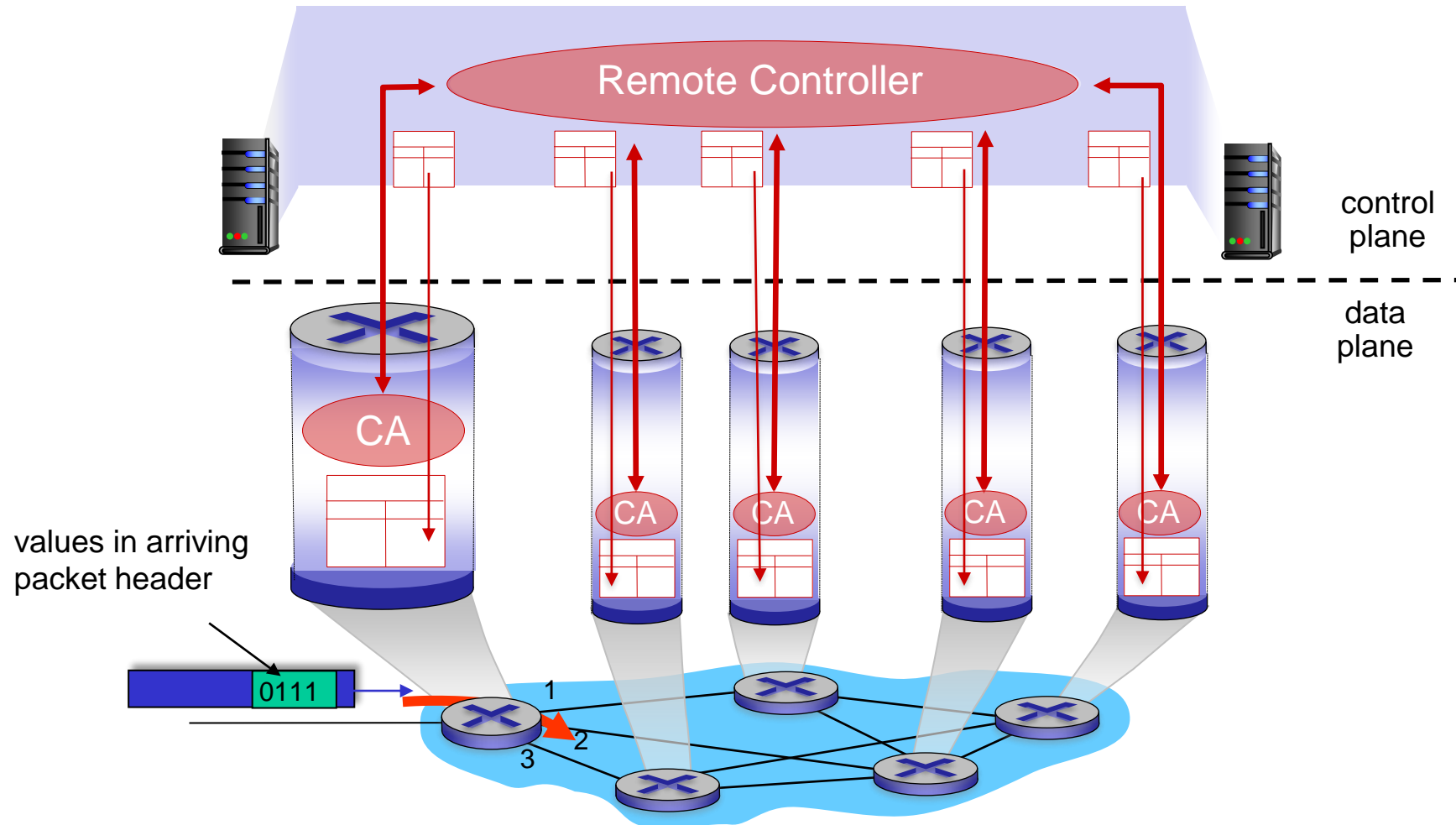
# Per-router control plane

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 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*

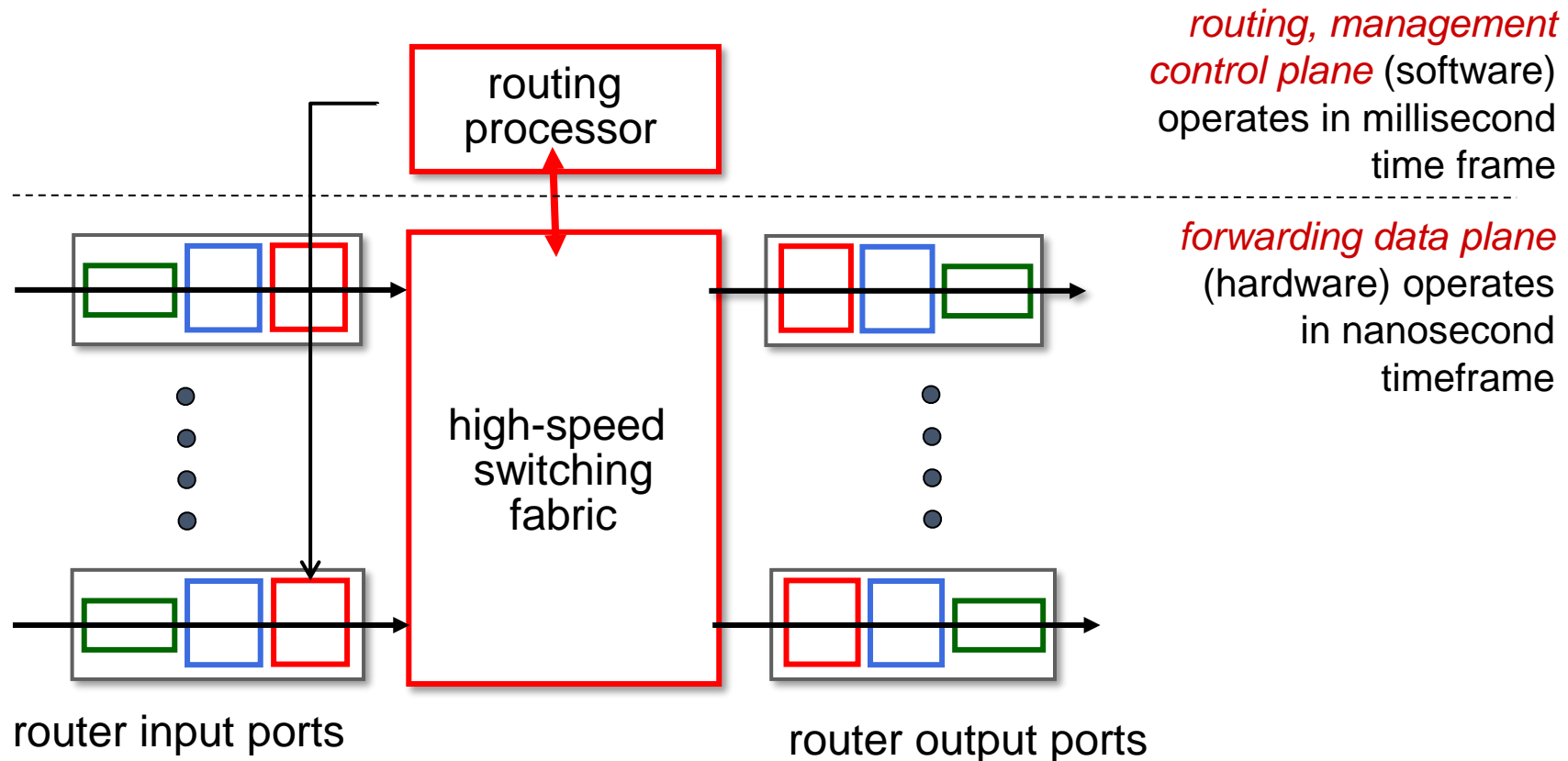
# Network layer: “data plane” roadmap

- Network layer: overview
  - data plane
  - control plane
- What’s inside a router
  - input 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



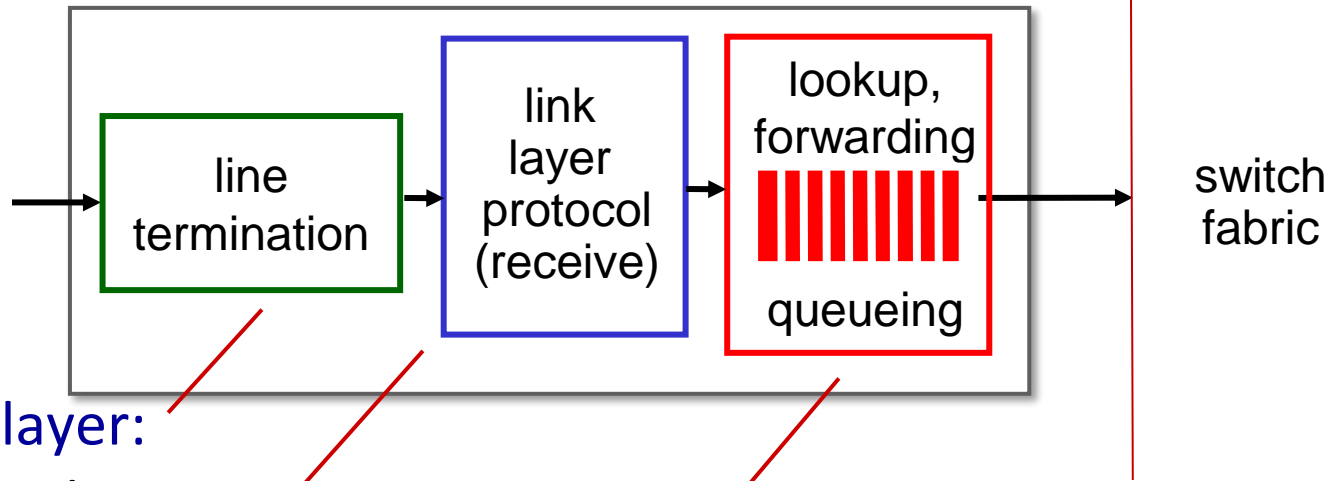
# Router architecture overview

high-level view of generic router architecture:





# Input port functions



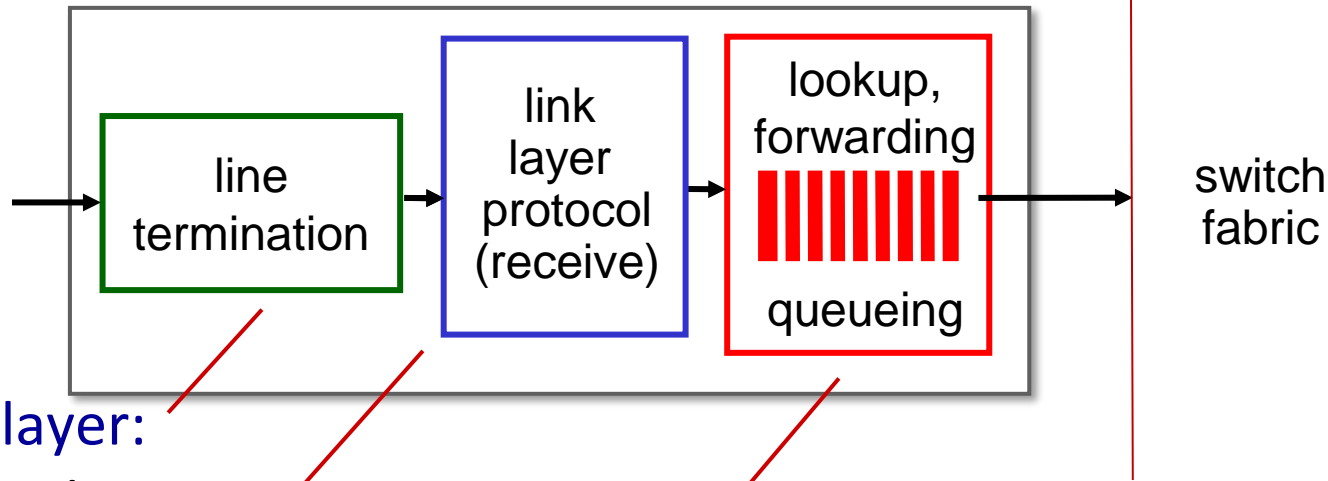
physical layer:  
bit-level reception

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”*)
- goal: complete input port processing at ‘line speed’
- **input port queueing:** 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)
- **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<br>through<br>11001000 00010111 00010000 00000100<br>through<br>11001000 00010111 00010000 00000111 | n<br>3         |
| 11001000 00010111 00011000 11111111<br>11001000 00010111 00011001 00000000<br>through<br>11001000 00010111 00011111 11111111            | 2              |
| 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 match! 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:

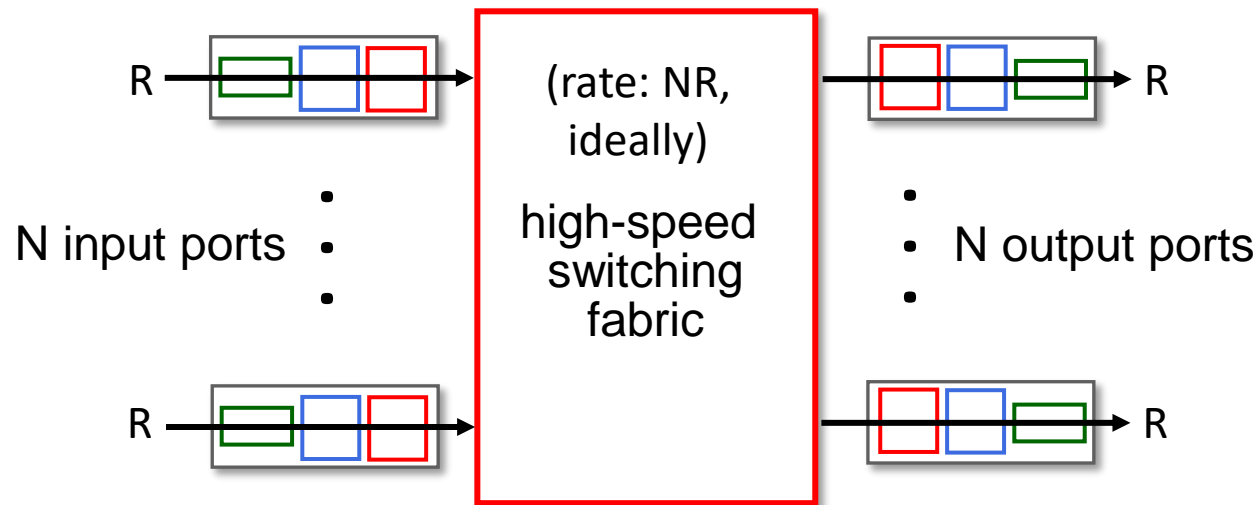
|                                     |                  |
|-------------------------------------|------------------|
| 11001000 00010111 00010110 10100001 | which interface? |
| 11001000 00010111 00011000 10101010 | which interface? |

# 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

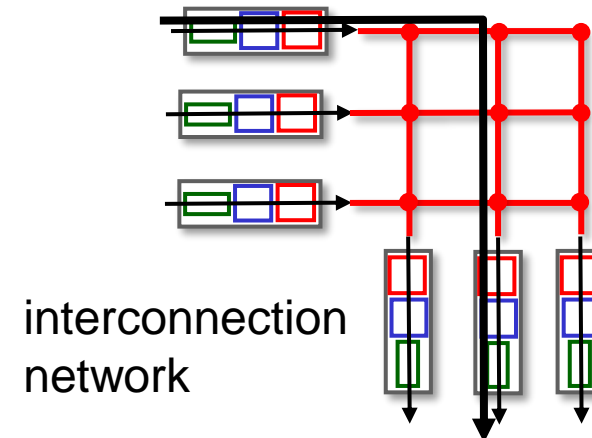
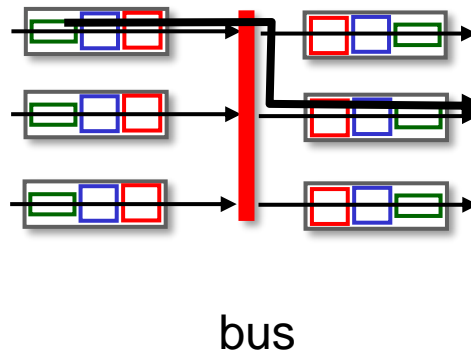
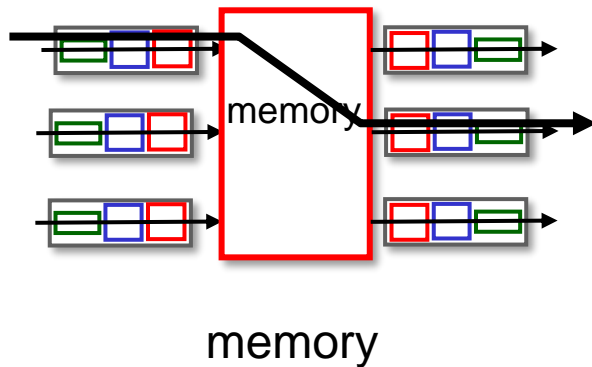
# Switching fabrics

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



# Switching fabrics

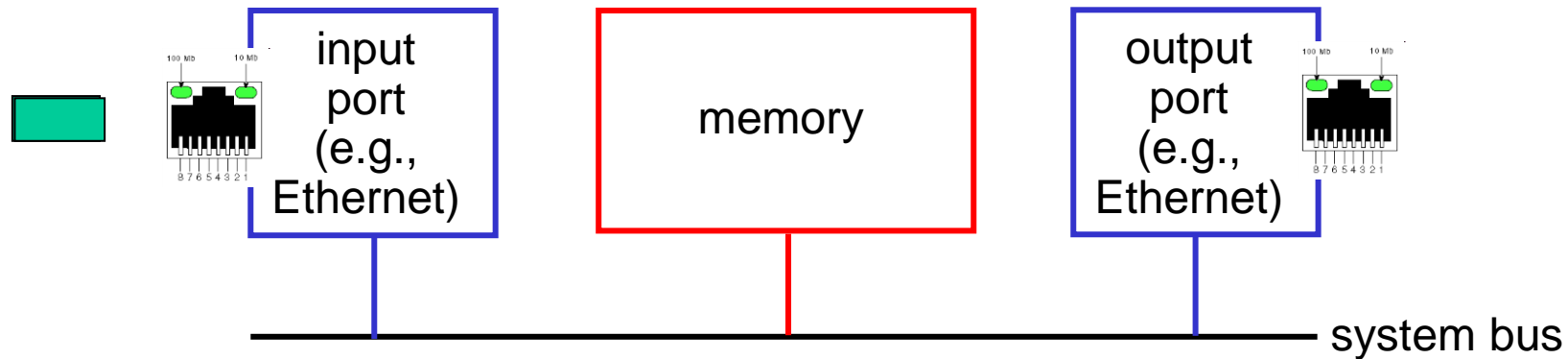
- transfer packet from input link to appropriate output link
- **switching rate**: rate at which packets can be transfer from inputs to outputs
  - often measured as multiple of input/output line rate
  - N inputs: switching rate N times line rate desirable
- three major types of switching fabrics:



# Switching via memory

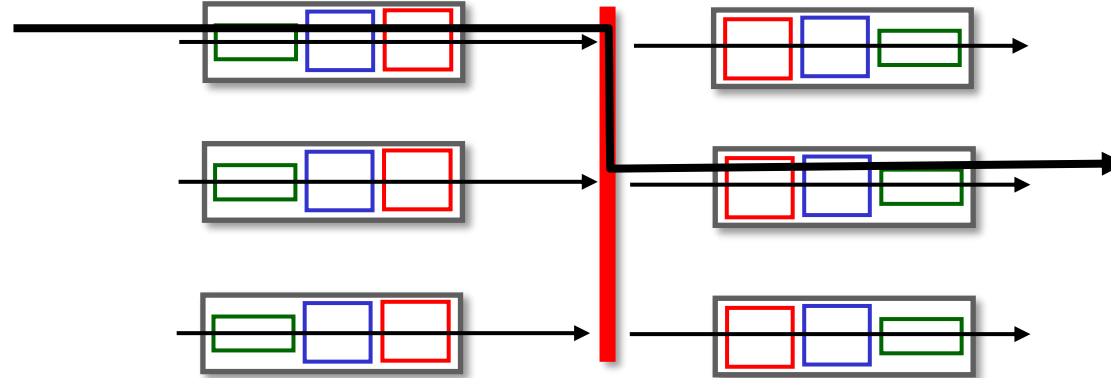
## first generation routers:

- 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

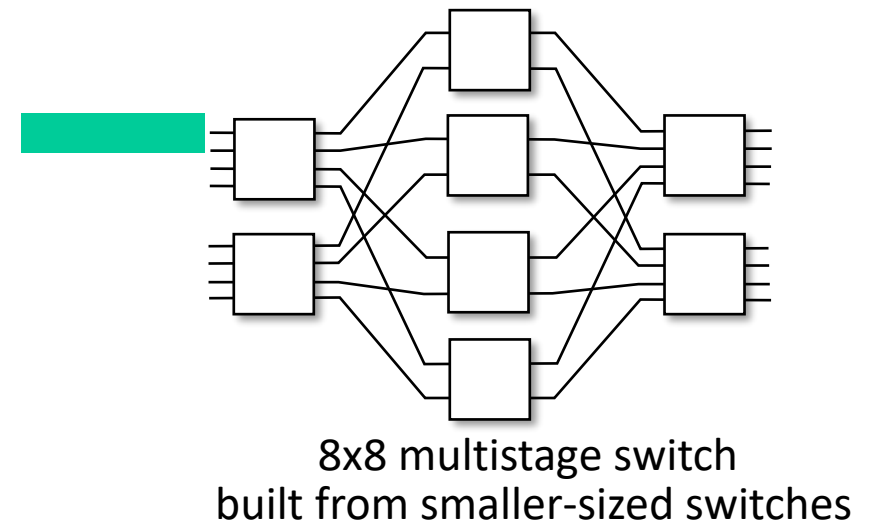
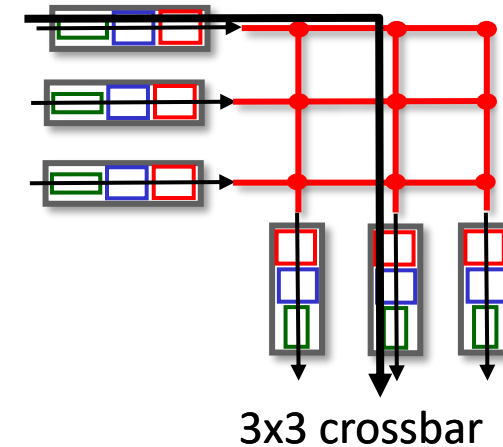
- datagram from input port memory to output port memory via a shared bus
- *bus contention*: switching speed limited by bus bandwidth
- 32 Gbps bus, Cisco 5600: sufficient speed for access routers





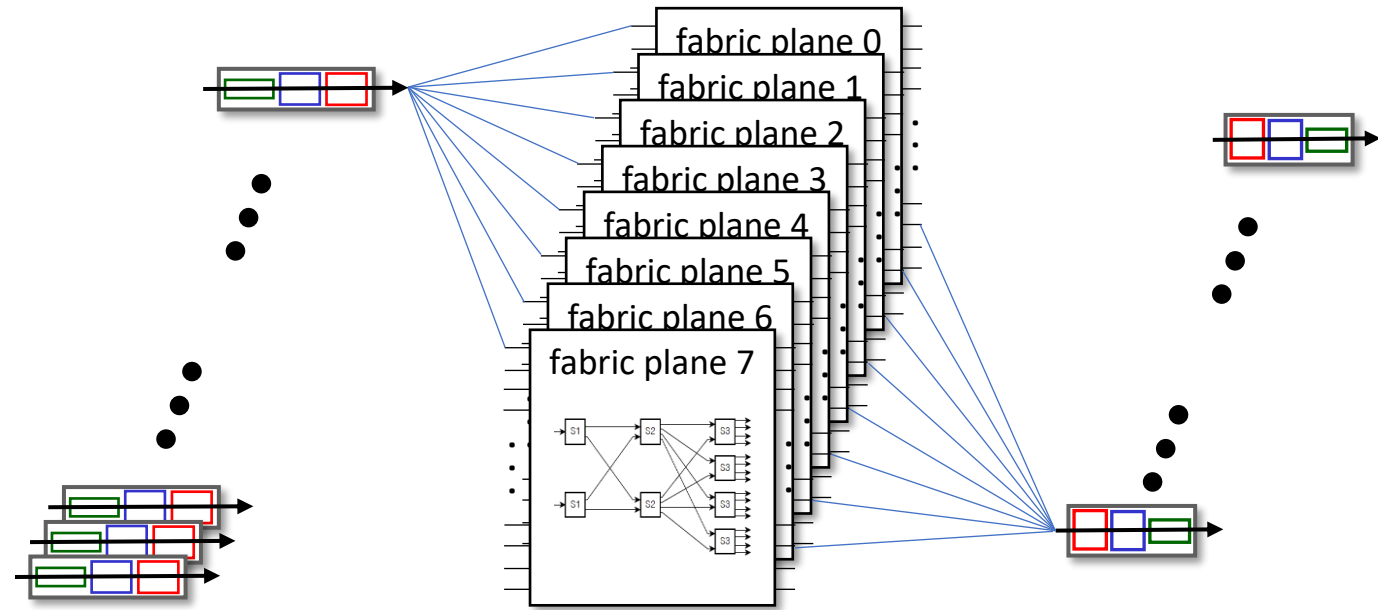
# Switching via interconnection network

- 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



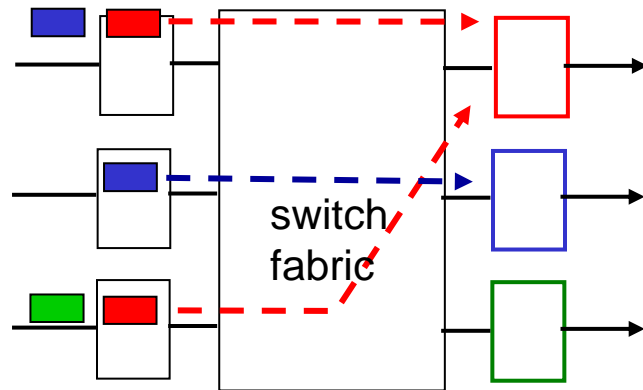
# Switching via interconnection network

- scaling, using multiple switching “planes” in parallel:
  - speedup, scaleup via parallelism
- Cisco CRS router:
  - basic unit: 8 switching planes
  - each plane: 3-stage interconnection network
  - up to 100's Tbps switching capacity

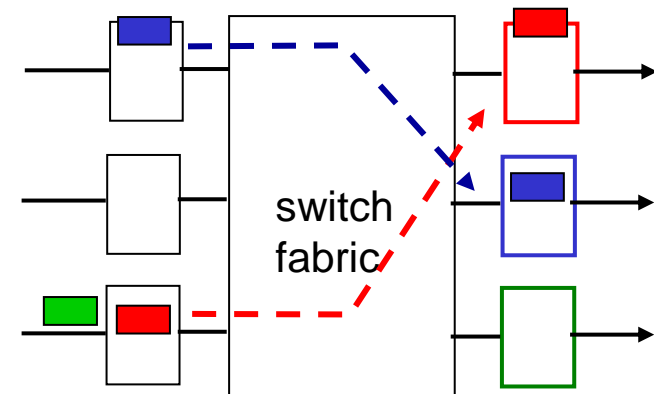


# Input port queuing

- 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

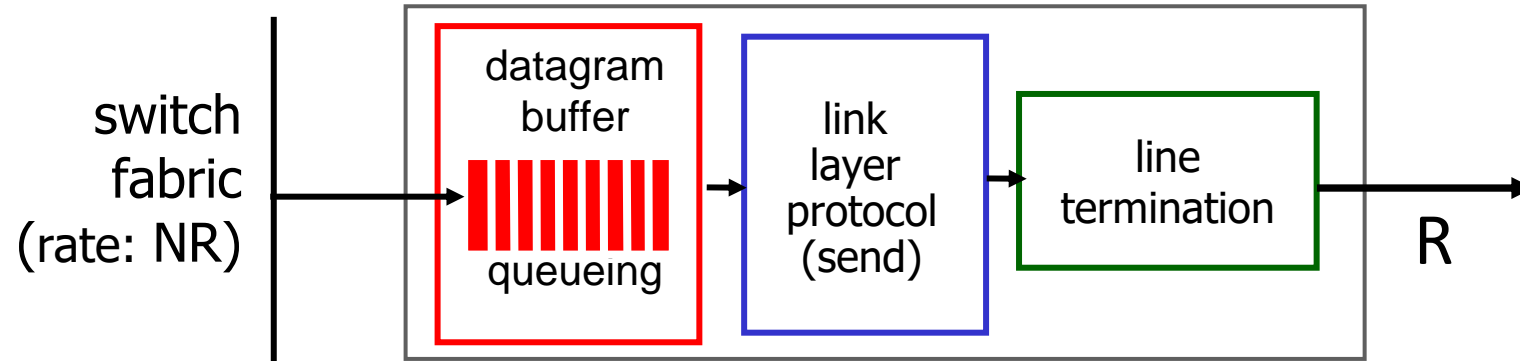


output port contention: only one red datagram can be transferred. lower red packet is *blocked*



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

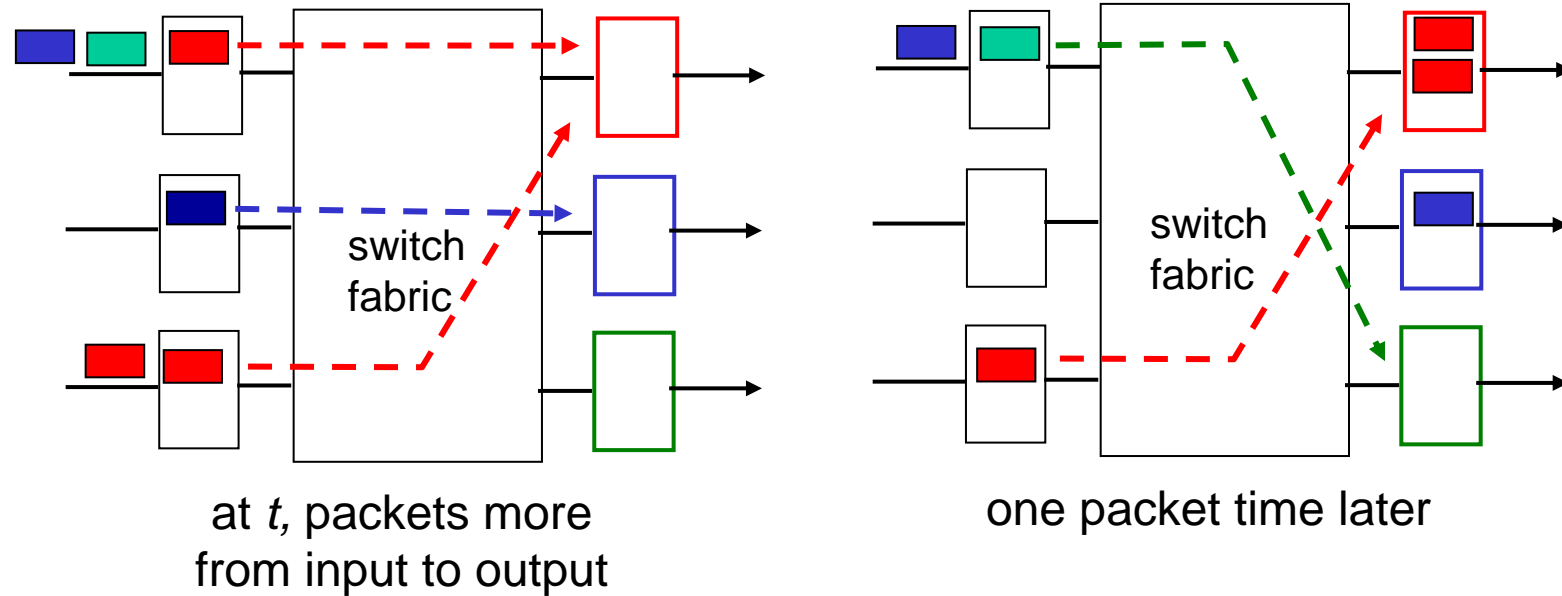


Datagrams can be lost due to congestion, lack of buffers



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

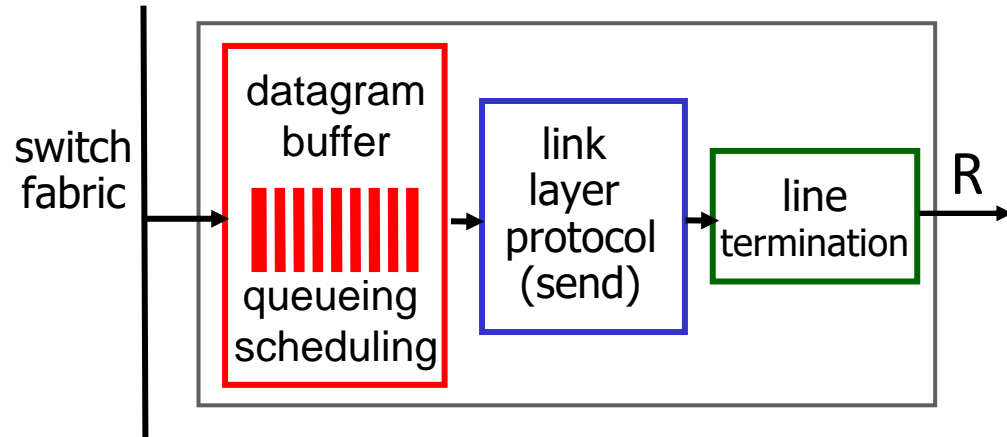
- more recent recommendation: with  $N$  flows, buffering equal to

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

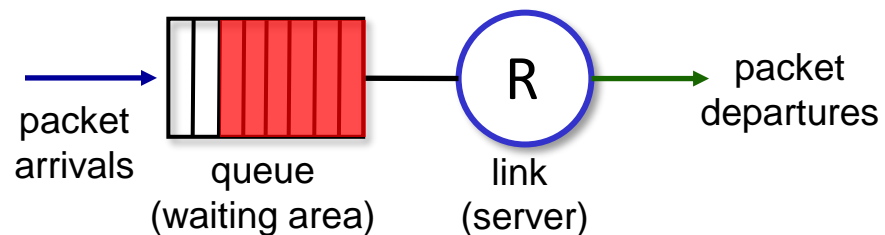
- but *too* much buffering can increase delays (particularly in home routers)
  - long RTTs: poor performance for realtime apps, sluggish TCP response
  - recall delay-based congestion control: “keep bottleneck link just full enough (busy) but no fuller”



# 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

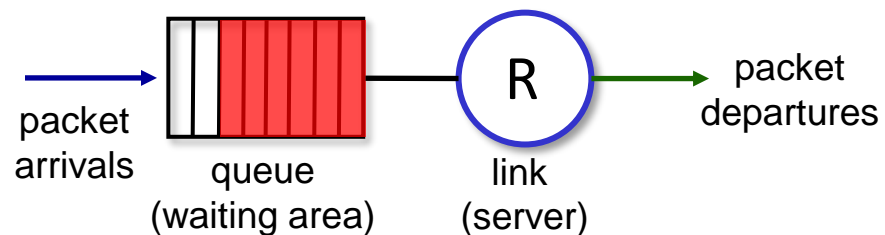
**packet scheduling:** deciding which packet to send next on link

- 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?

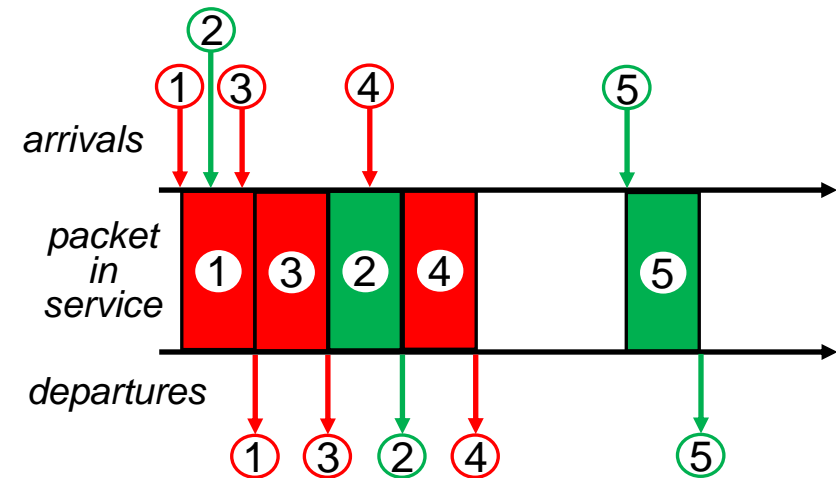
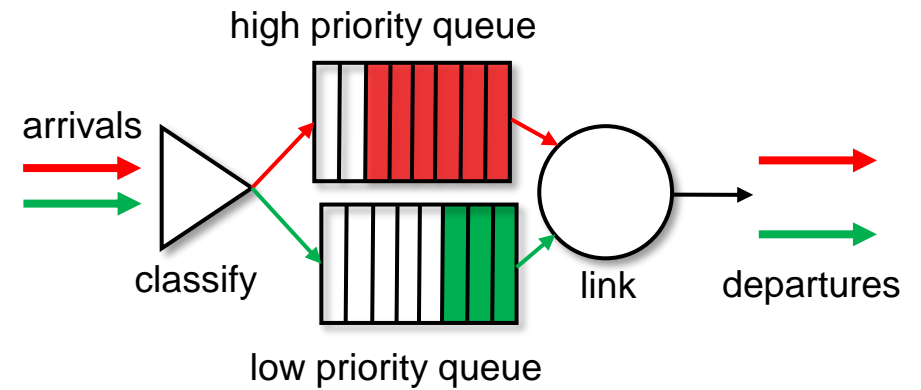
Abstraction: queue



# Scheduling policies: priority

## *Priority scheduling:*

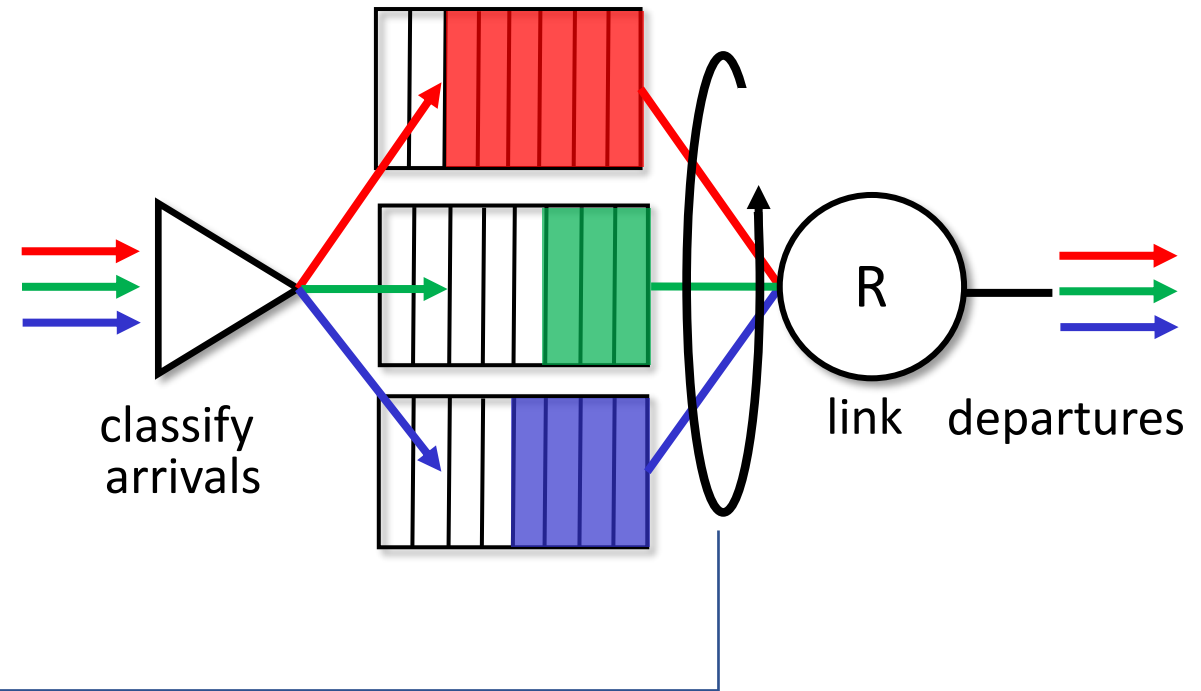
- arriving traffic classified, queued by class
  - any header fields can be used for classification
- send packet from highest priority queue that has buffered packets
  - FCFS within priority class



# 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



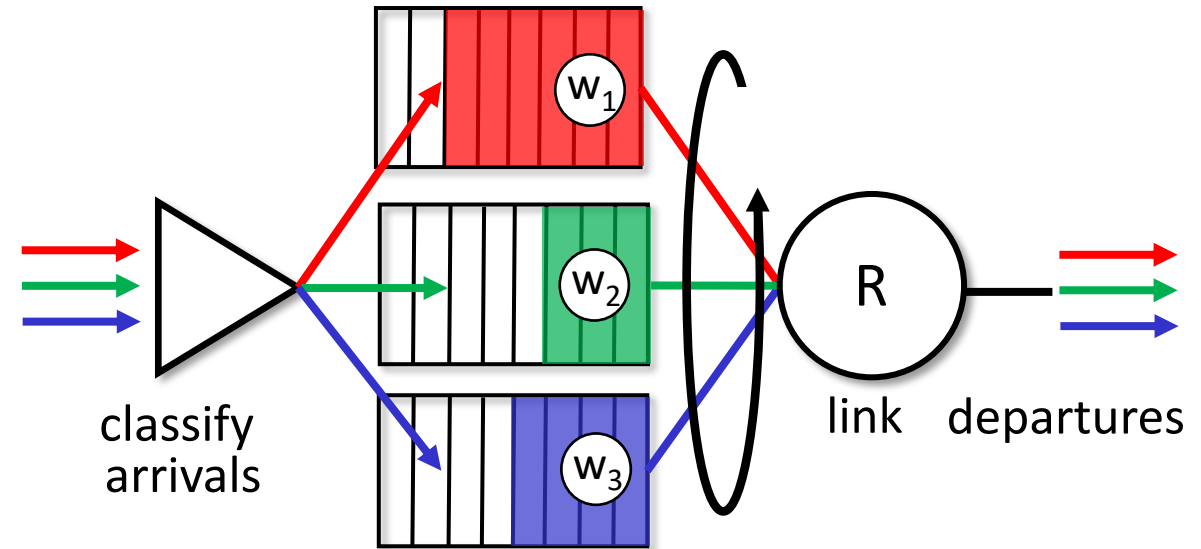
# Scheduling policies: weighted fair queueing

## *Weighted Fair 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*

# 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”

# ISP: telecommunications or information service?

Is an ISP a “telecommunications service” or an “information service” provider?

- the answer *really* matters from a regulatory standpoint!

US Telecommunication Act of 1934 and 1996:

- *Title II*: imposes “common carrier duties” on *telecommunications services*: reasonable rates, non-discrimination and *requires regulation*
- *Title I*: applies to *information services*:
  - no common carrier duties (*not regulated*)
  - but grants FCC authority “... as may be necessary in the execution of its functions”<sup>4</sup>



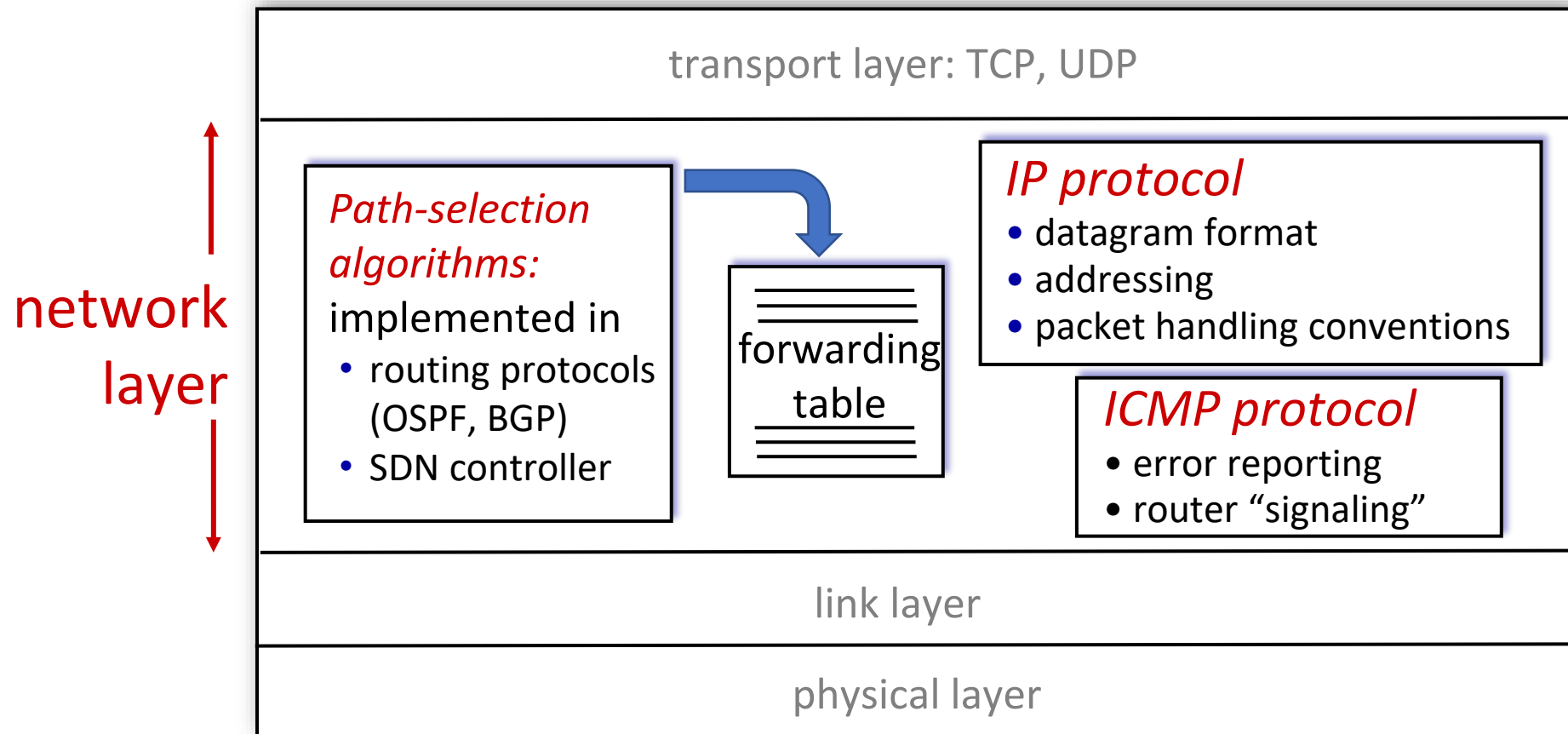
# Network layer: “data plane” roadmap

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

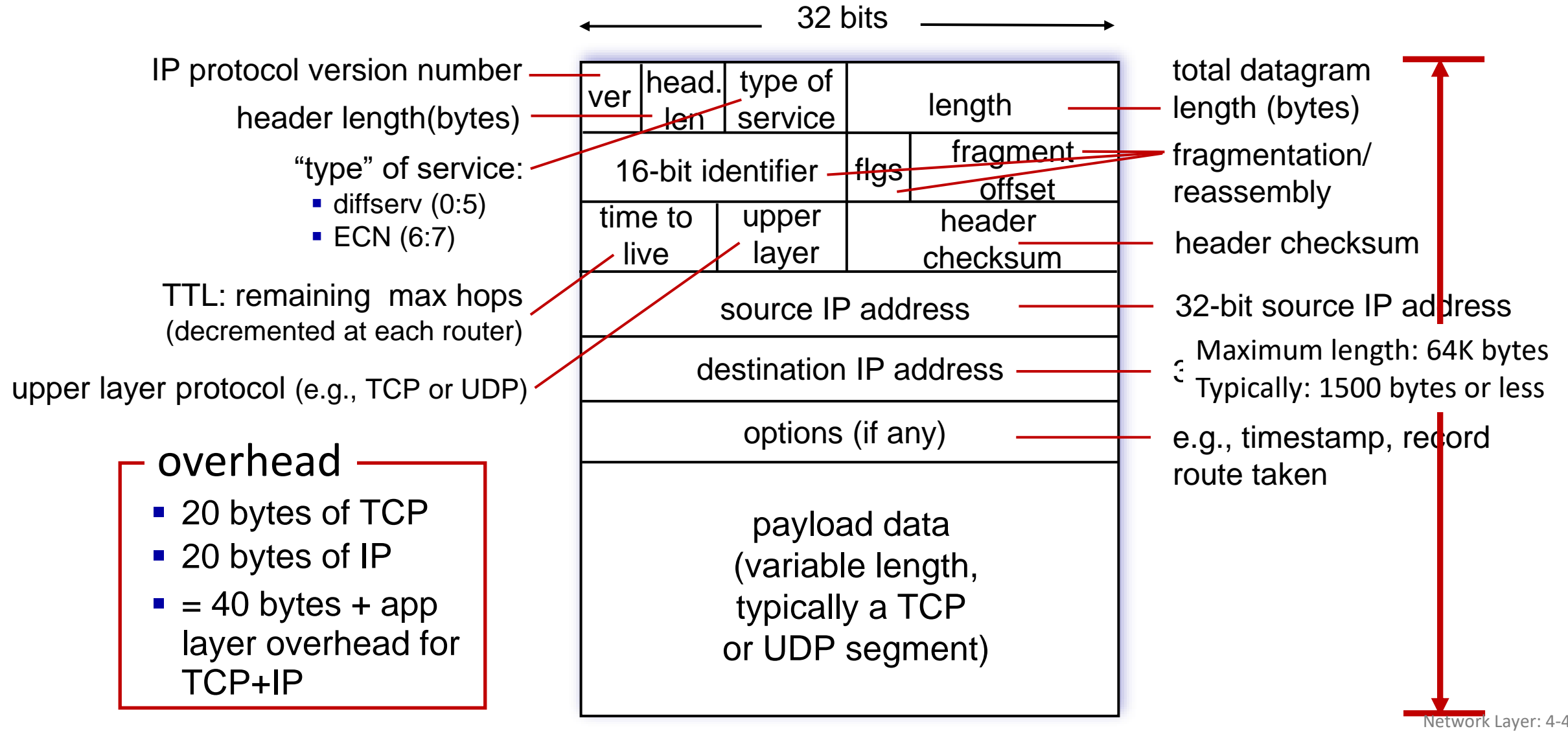


# Network Layer: Internet

host, router network layer functions:

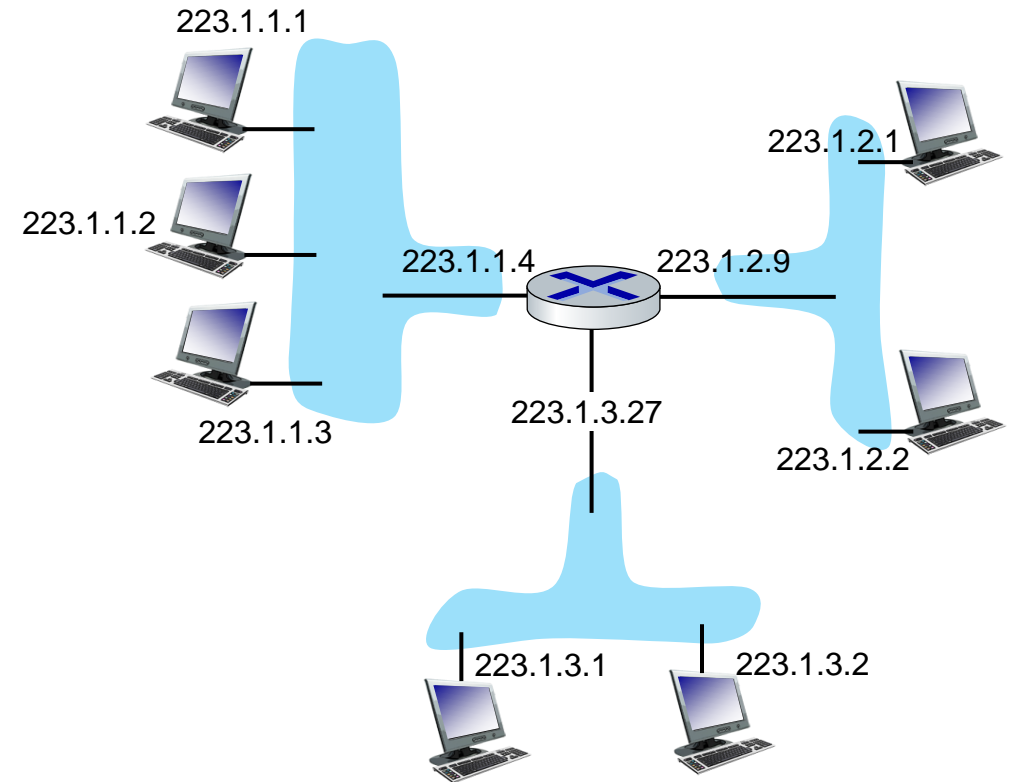


# IP Datagram format



# 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)



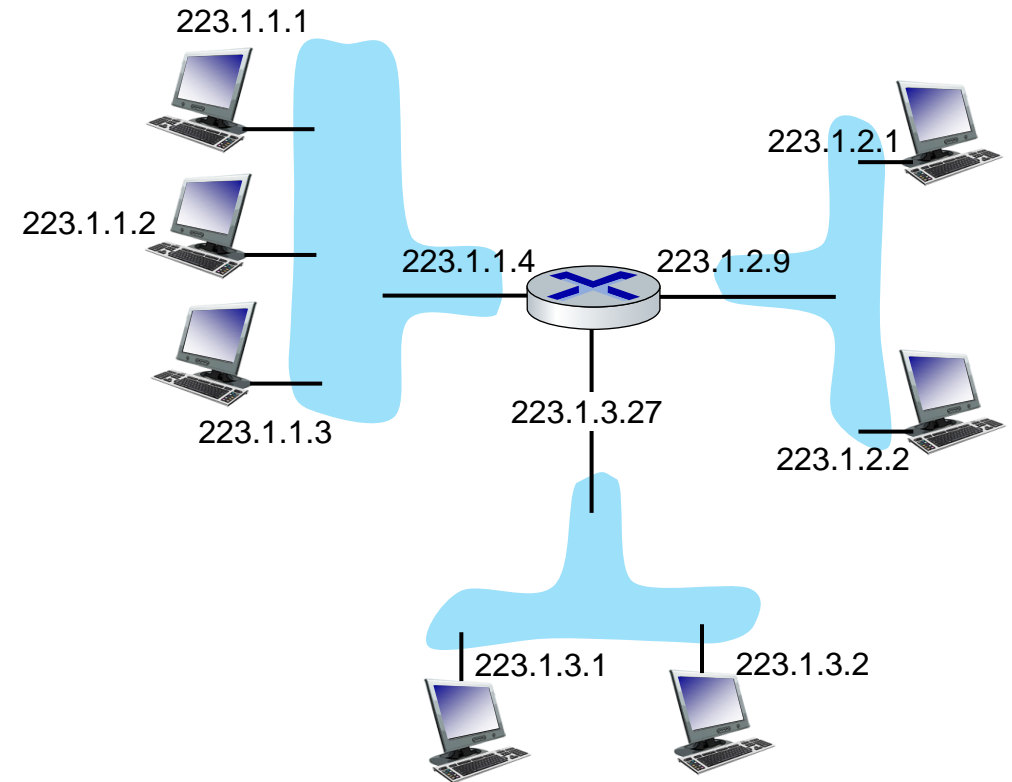
dotted-decimal IP address notation:

223.1.1.1 = 11011111 00000001 00000001 00000001

223                      1                      1                      1

# 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

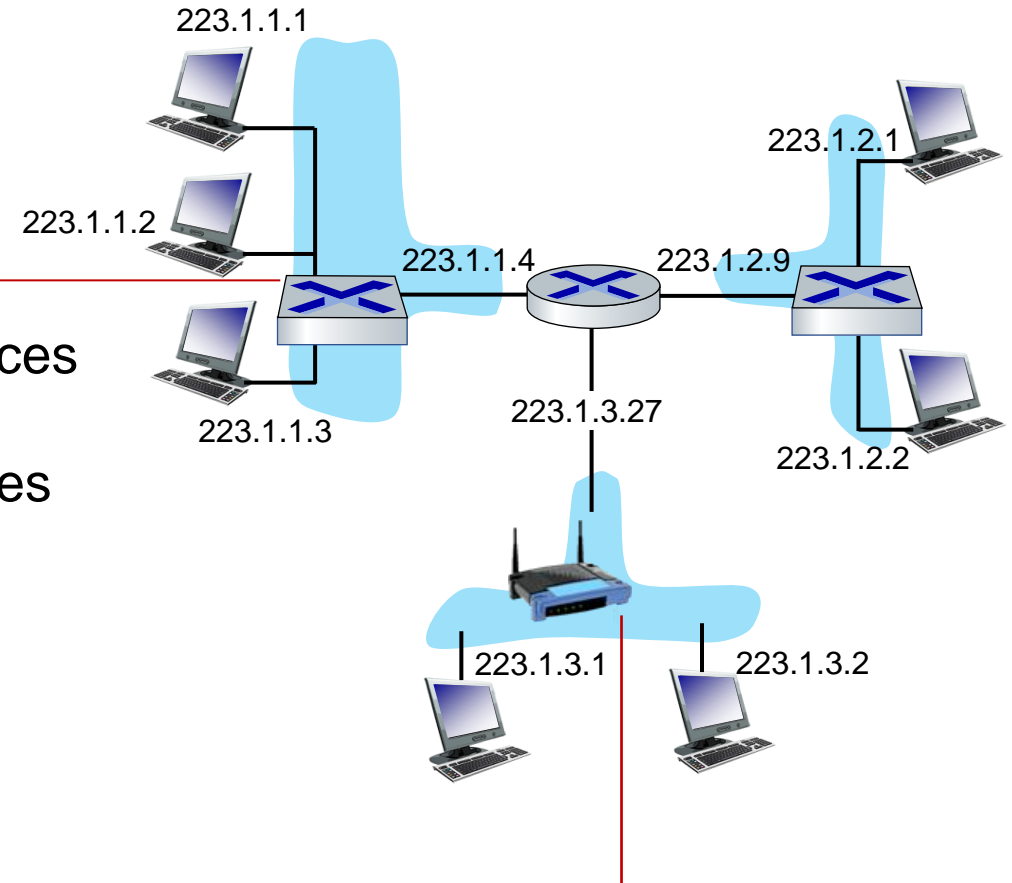
# IP addressing: introduction

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

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

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

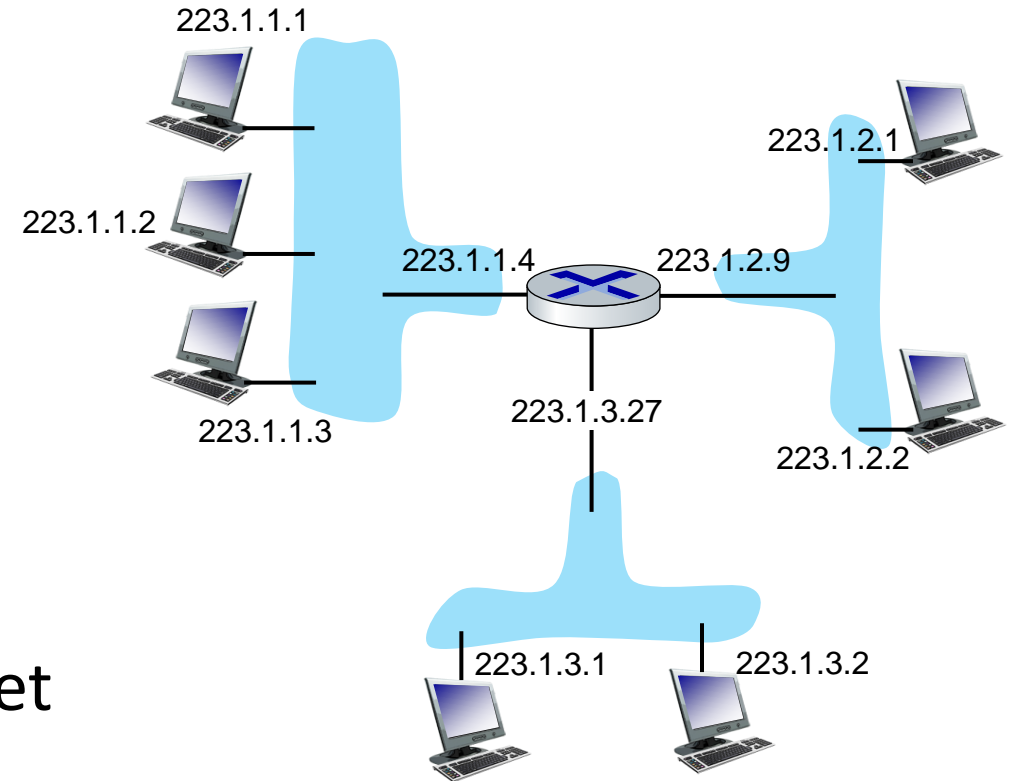
**A:** wired Ethernet interfaces connected by Ethernet switches



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

# Subnets

- *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

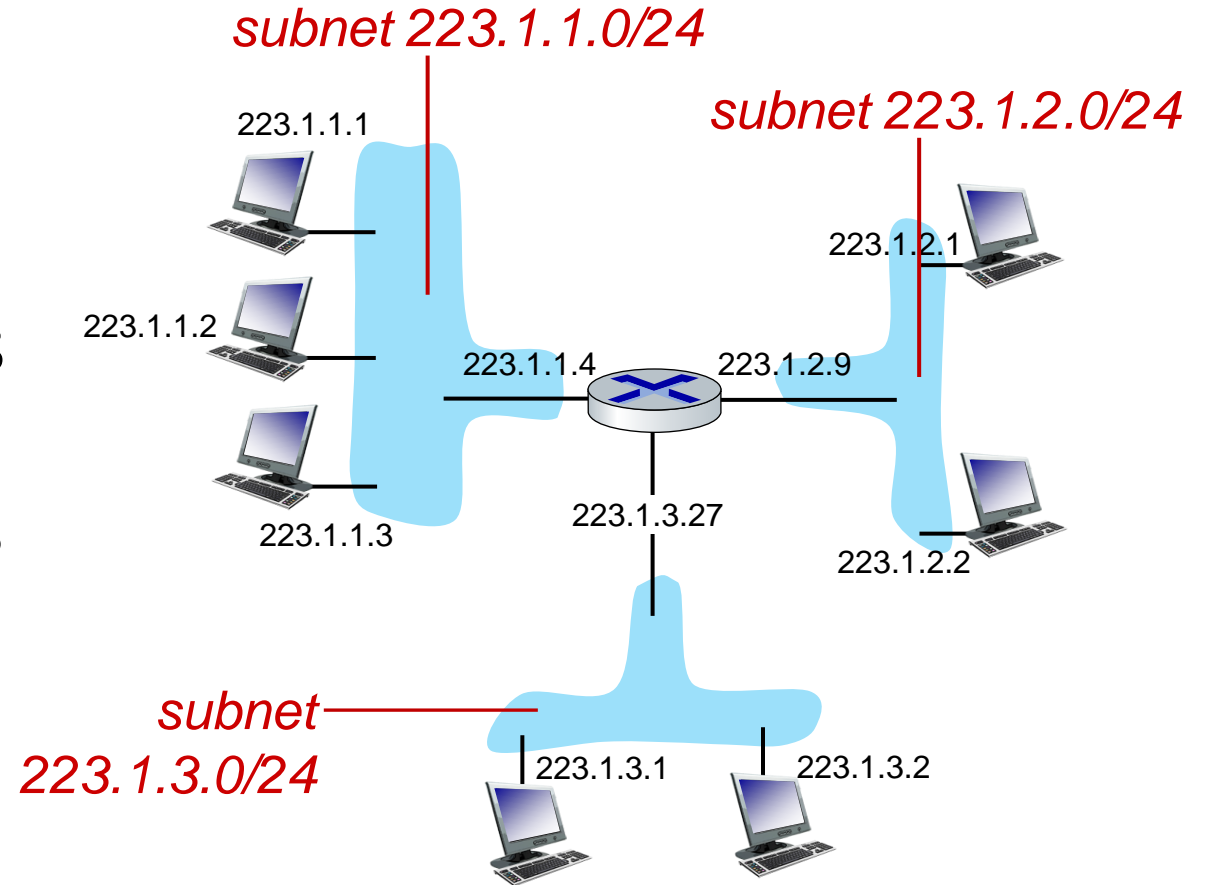


network consisting of 3 subnets

# 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*



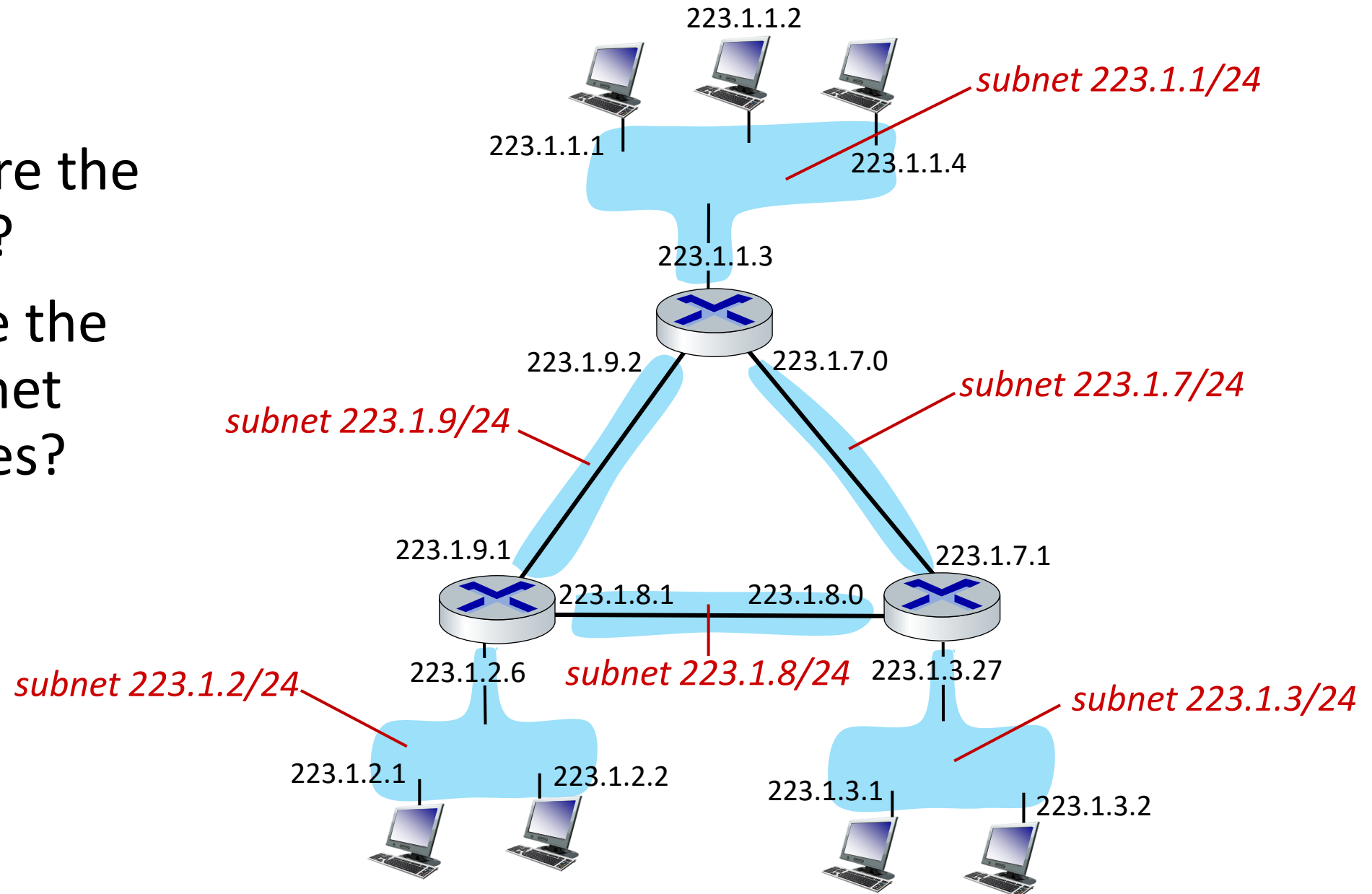
subnet mask: /24

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



# Subnets

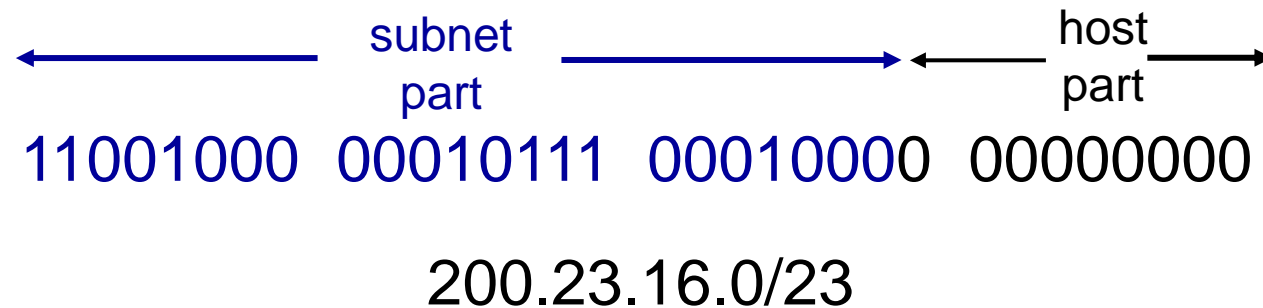
- where are the subnets?
- what are the /24 subnet addresses?



# 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. 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

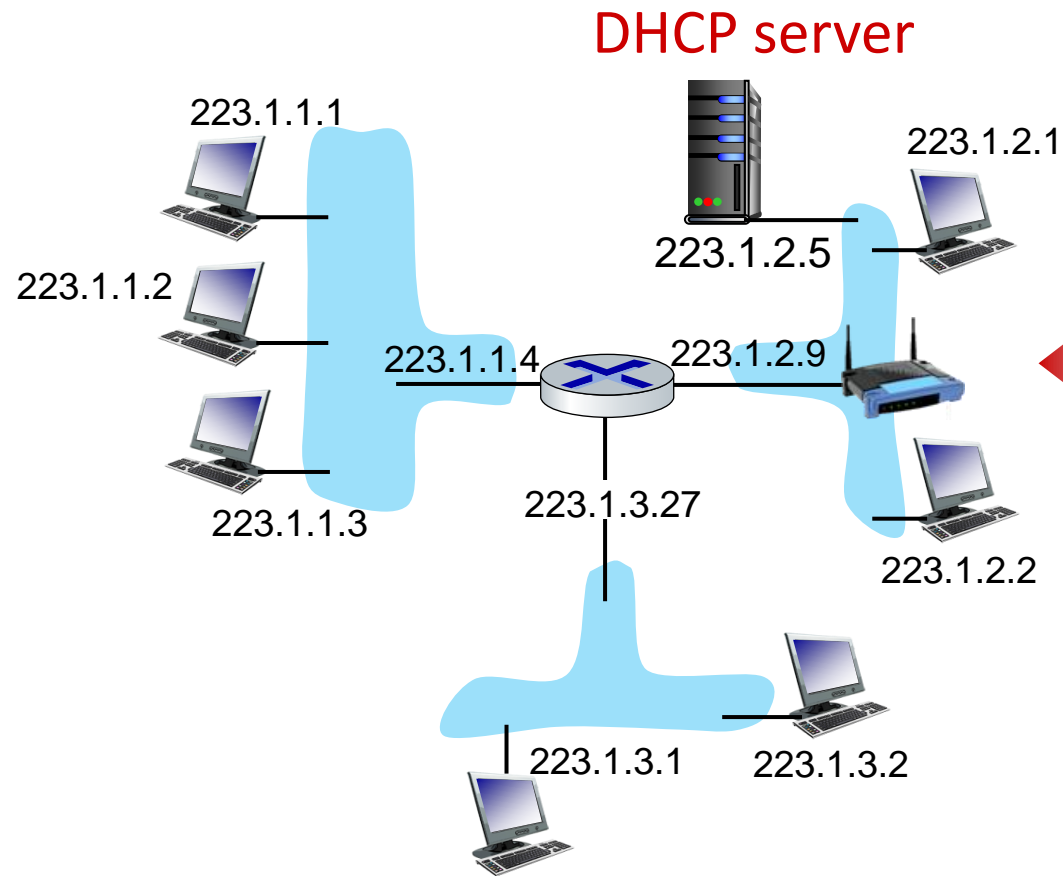
**goal:** host *dynamically* obtains IP address from network server when it “joins” network

- 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

DHCP server: 223.1.2.5



**DHCP discover**

Broadcast: is there a  
DHCP server out there?

Arriving client



**DHCP offer**

Broadcast: I'm a DHCP  
server! Here's an IP  
address you can use

**DHCP request**

Broadcast: OK. I would  
like to use this IP address!

**DHCP ACK**

Broadcast: OK. You've  
got that IP address!

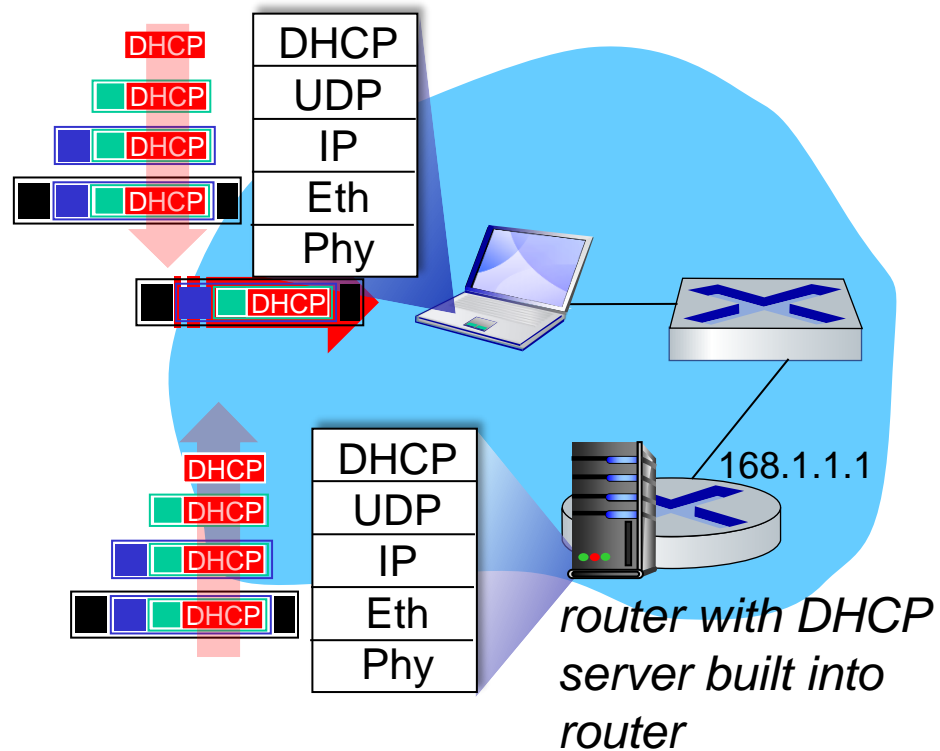
The two steps above can  
be skipped "if a client  
remembers and wishes to  
reuse a previously  
allocated network address"  
[RFC 2131]

# 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)

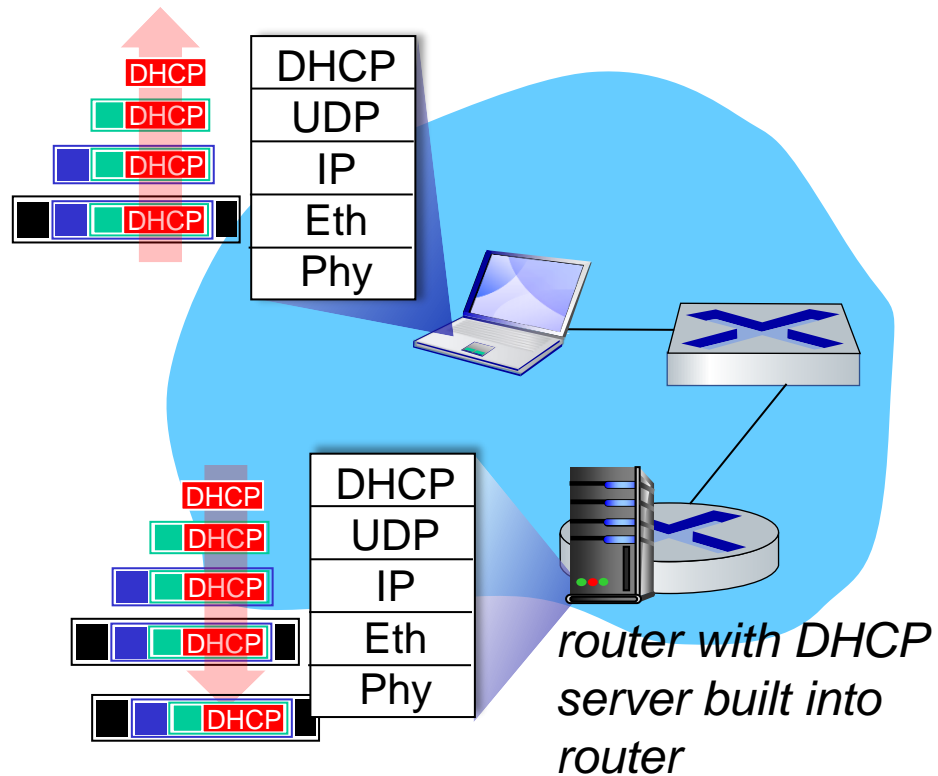
# DHCP: example



- 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: FFFFFFFF) on LAN, received at router running DHCP server
- Ethernet demux'ed to IP demux'ed, UDP demux'ed to DHCP



# DHCP: example



- DHCP 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

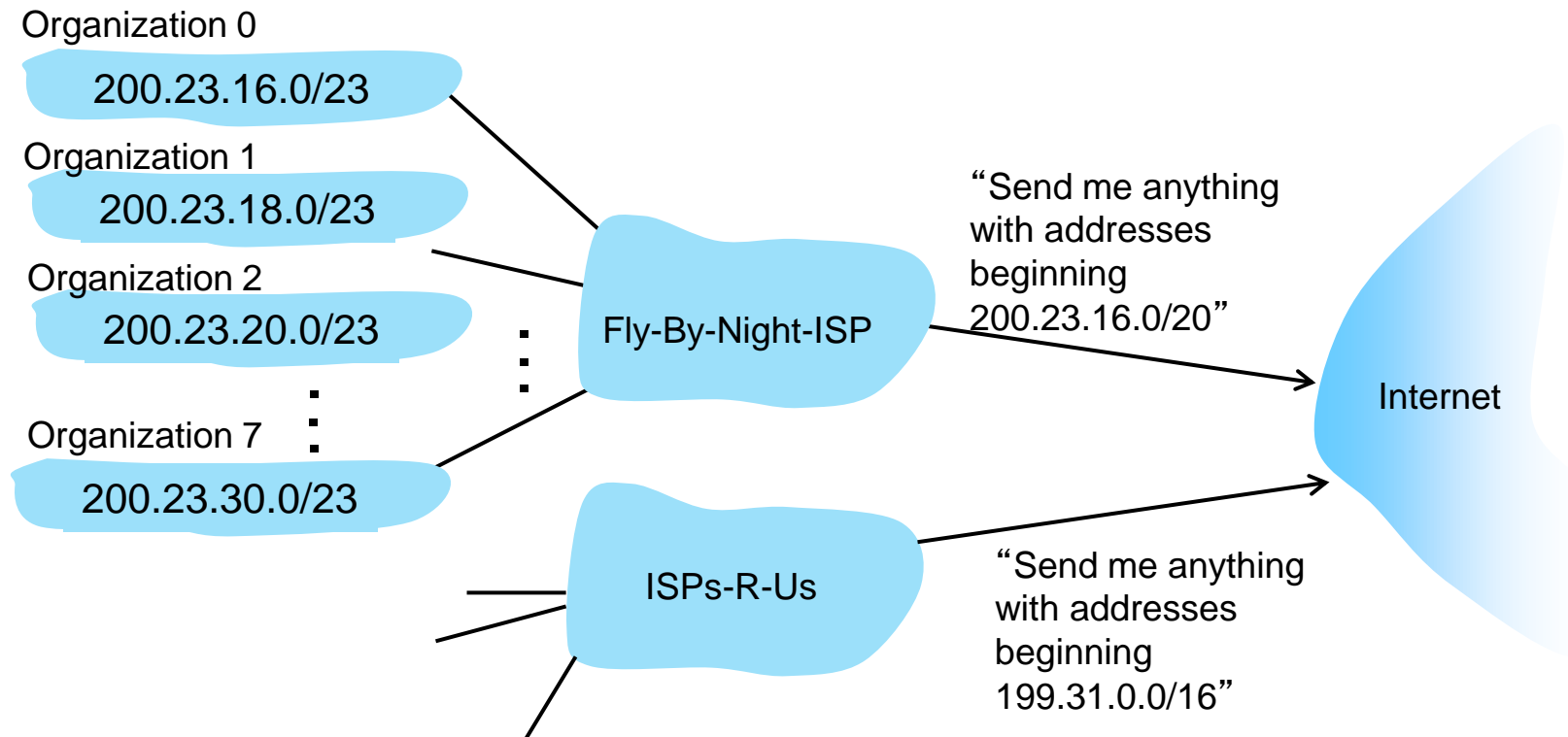
ISP's block      11001000 00010111 00010000 00000000    200.23.16.0/20

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

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

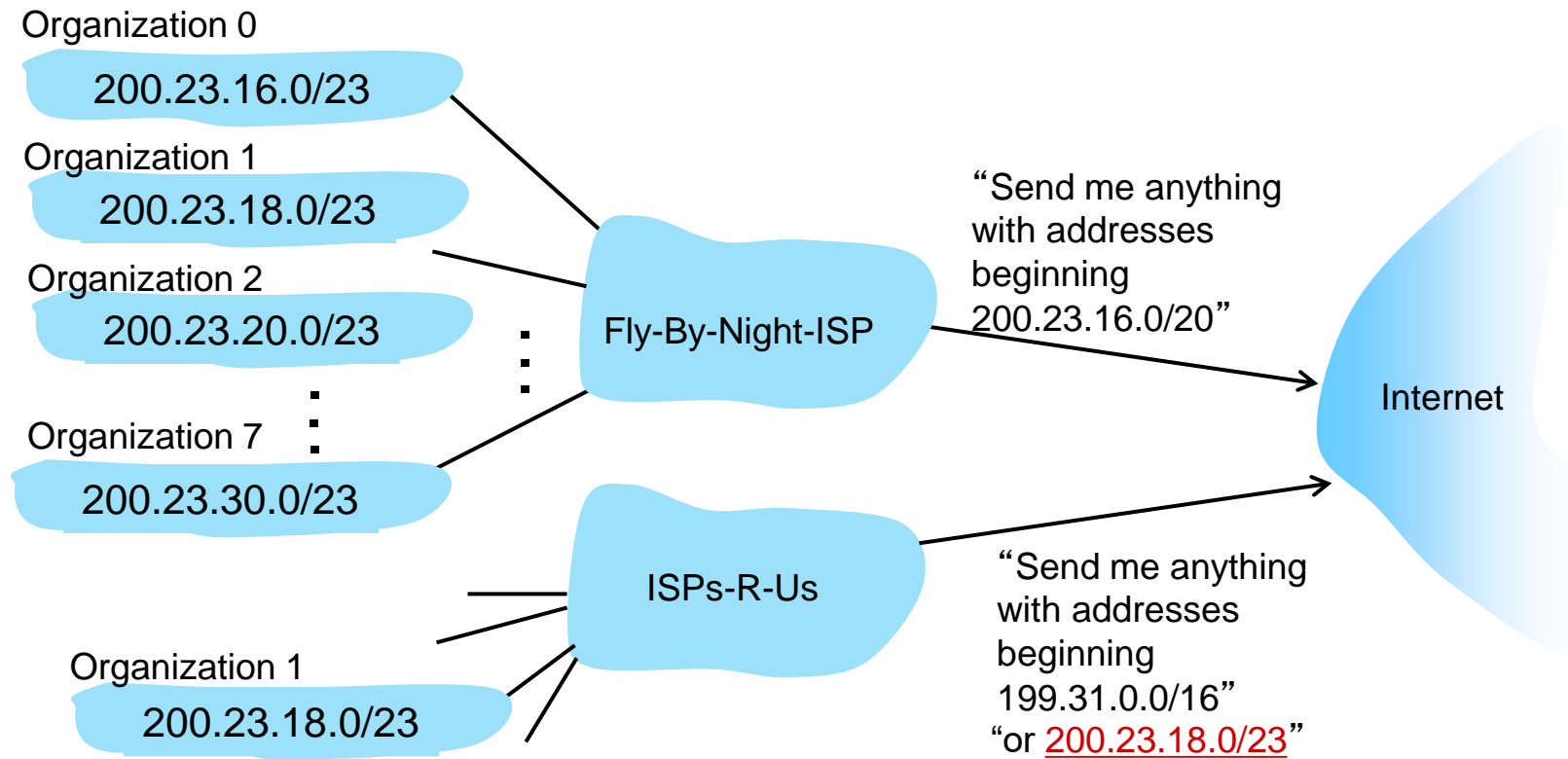
# Hierarchical addressing: route aggregation

hierarchical addressing allows efficient advertisement of routing information:



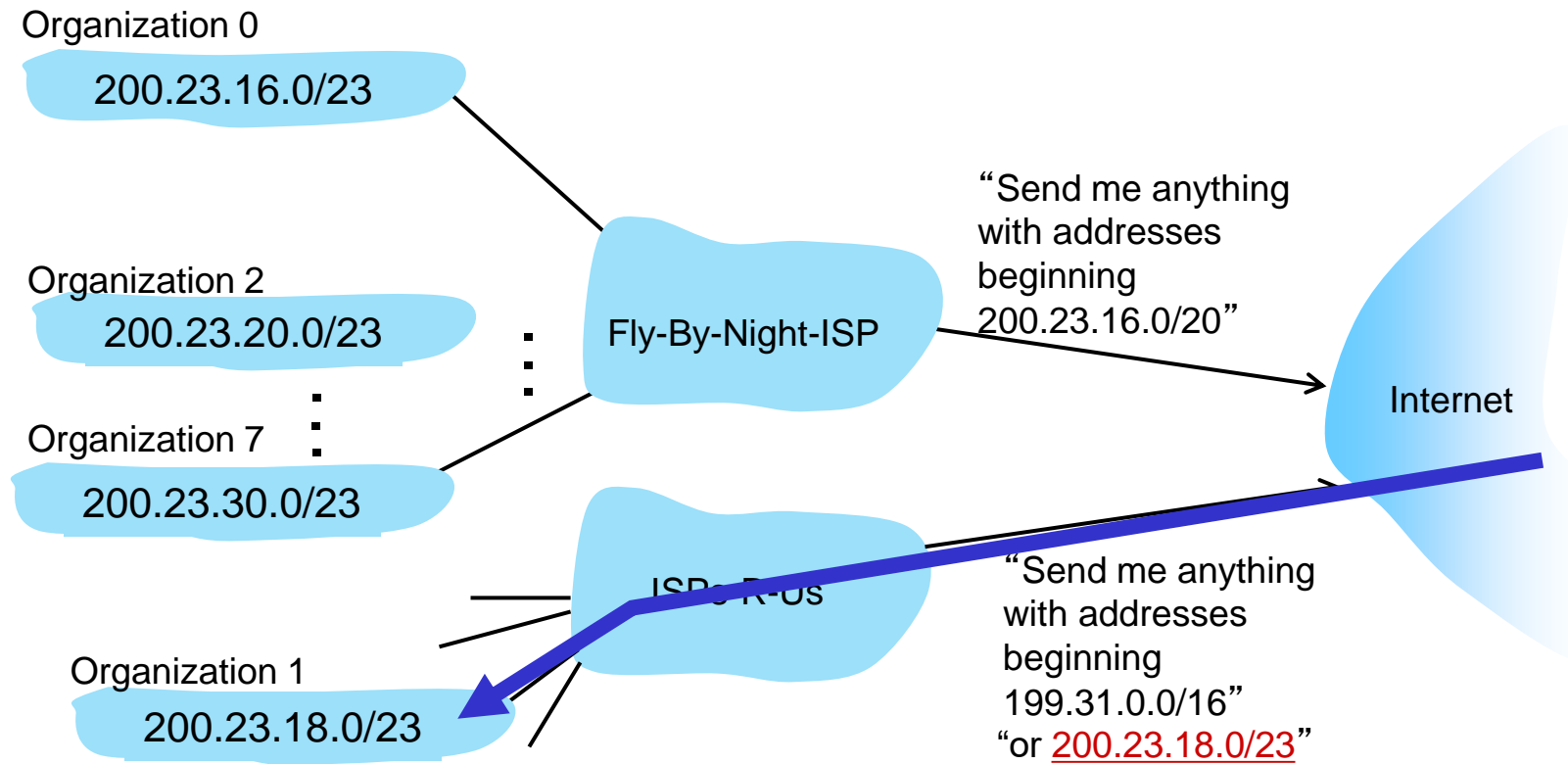
# Hierarchical addressing: more specific routes

- 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 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)

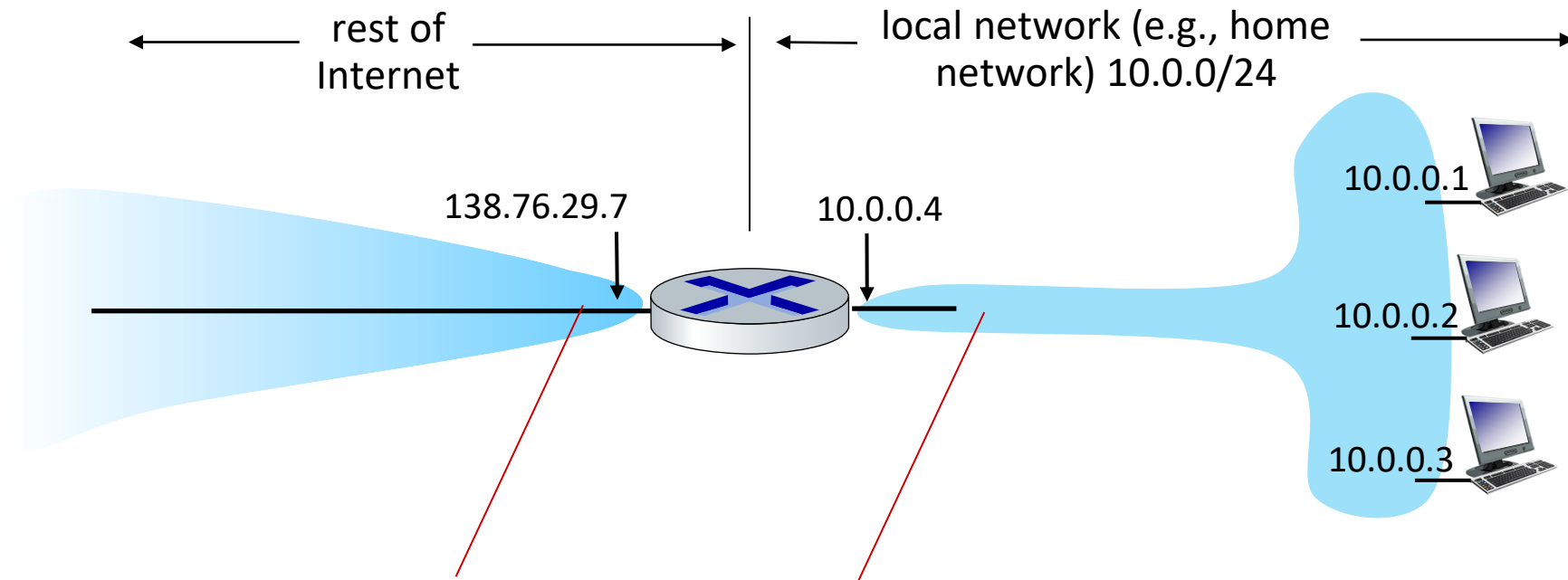
# Network layer: “data plane” roadmap

- Network layer: overview
  - data plane
  - control plane
- What’s inside a router
  - input 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

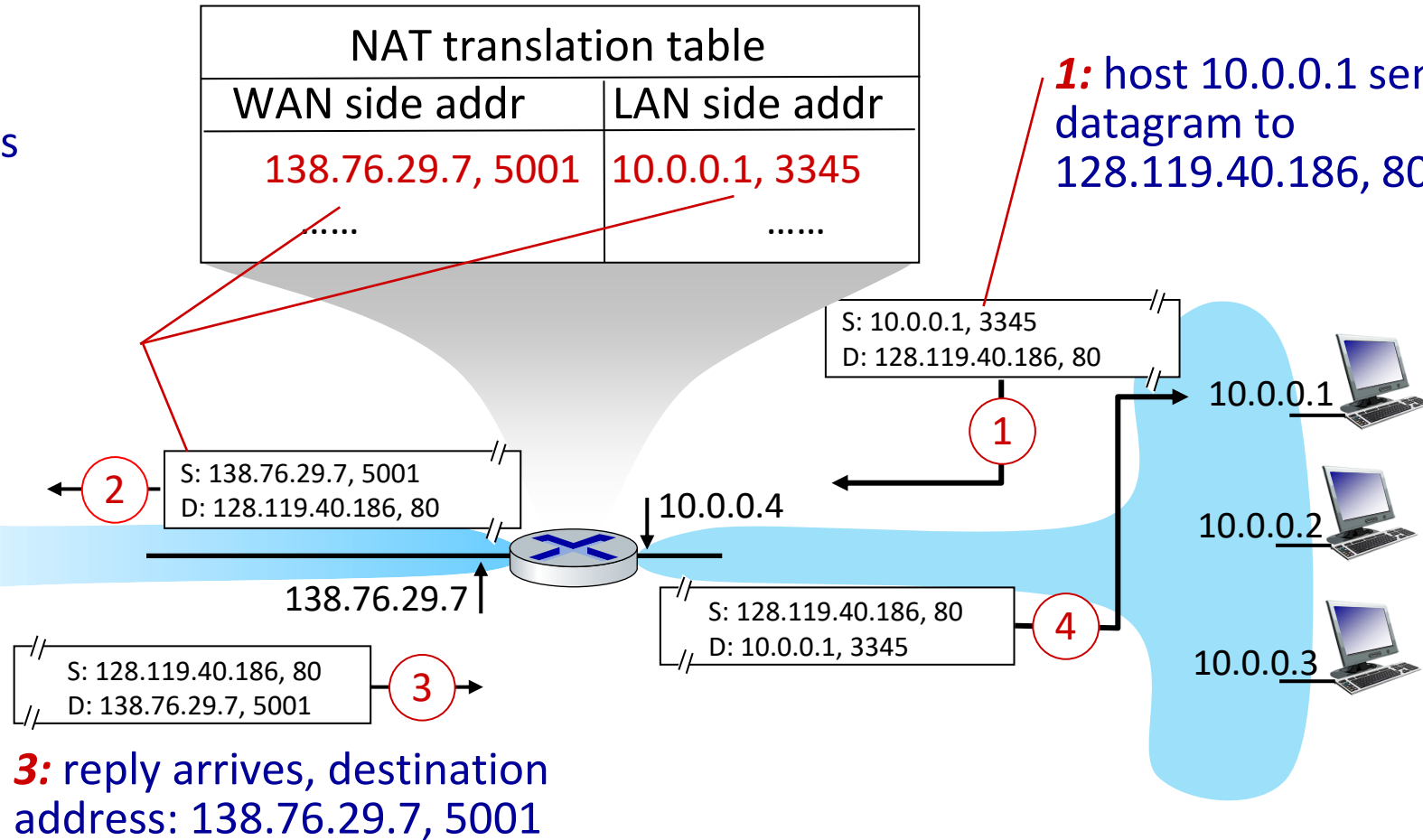
**implementation:** NAT router must (transparently):

- **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

**1:** host 10.0.0.1 sends datagram to 128.119.40.186, 80



**3:** reply arrives, destination address: 138.76.29.7, 5001

# NAT: network address translation

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

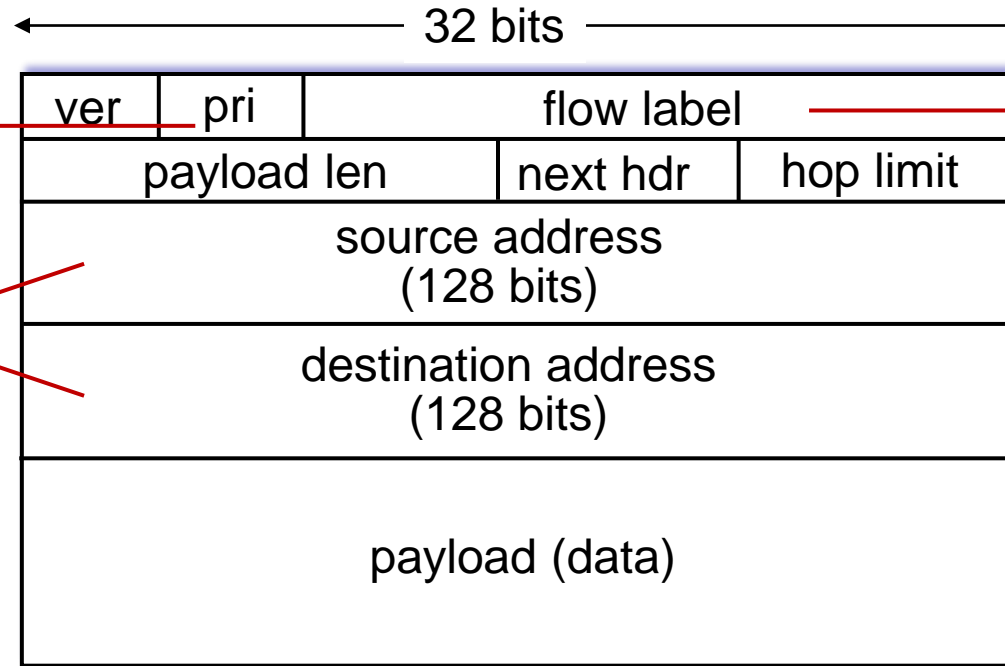
# IPv6: motivation

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

# IPv6 datagram format

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

**128-bit**  
IPv6 addresses



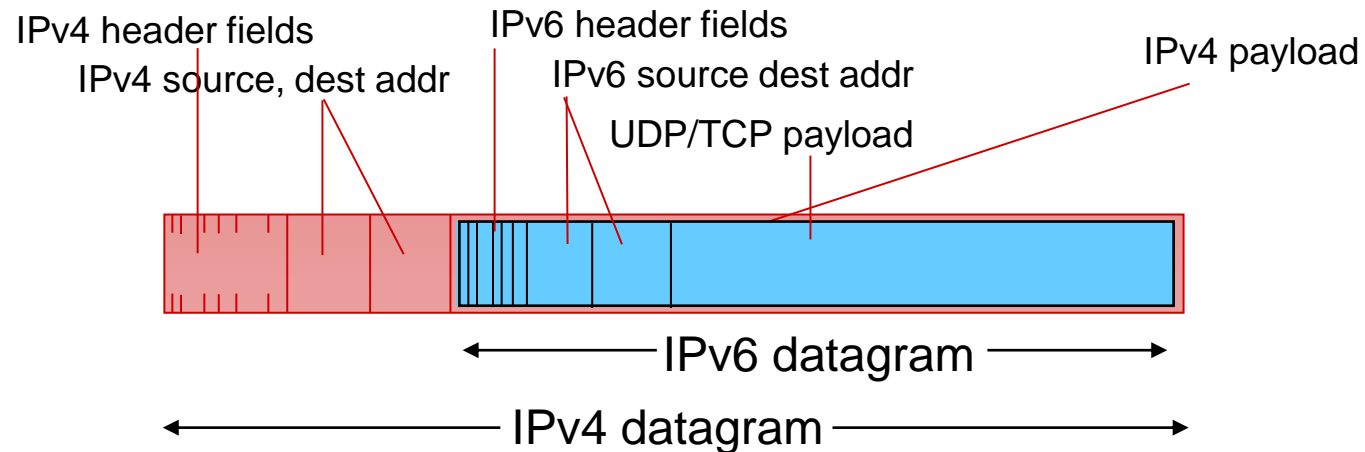
**flow label:** identify  
datagrams in same  
"flow." (concept of  
"flow" not well defined).

What's missing (compared with IPv4):

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

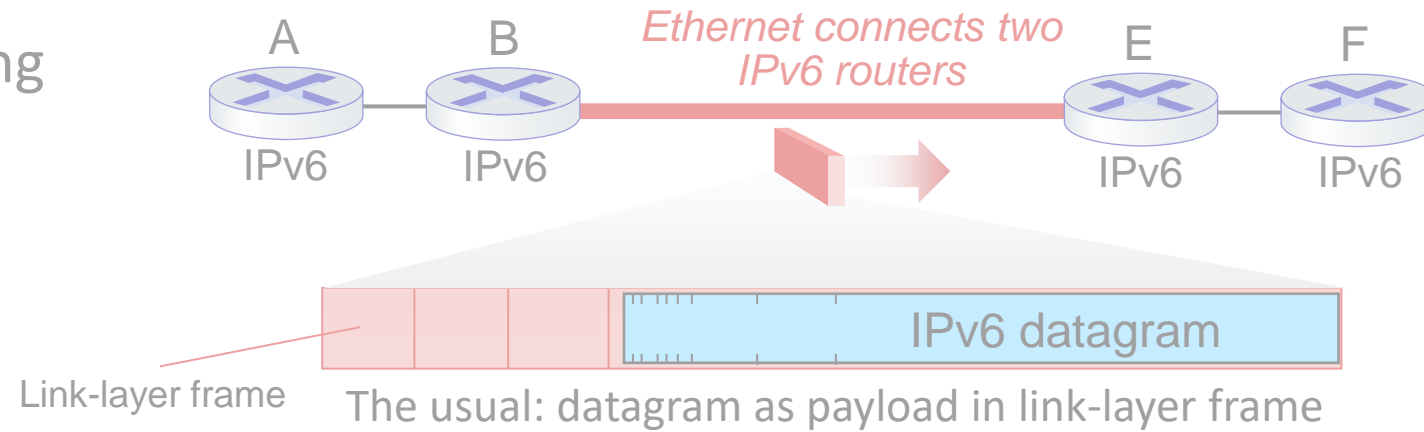
# Transition from IPv4 to IPv6

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

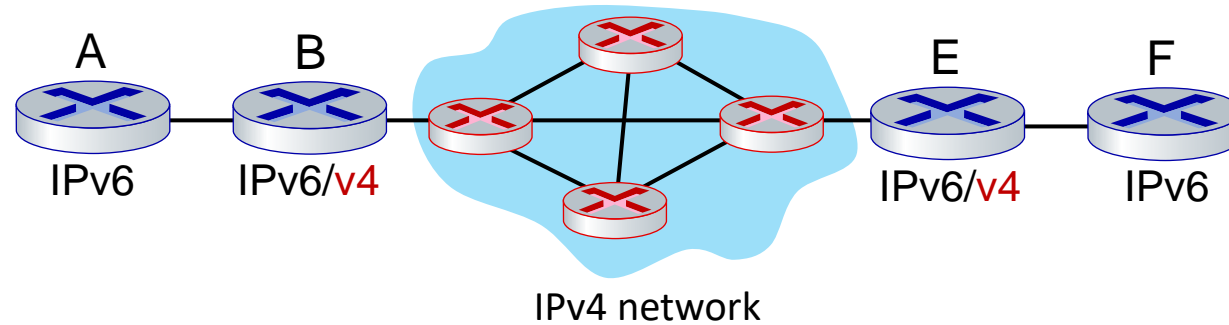


# Tunneling and encapsulation

Ethernet connecting two IPv6 routers:



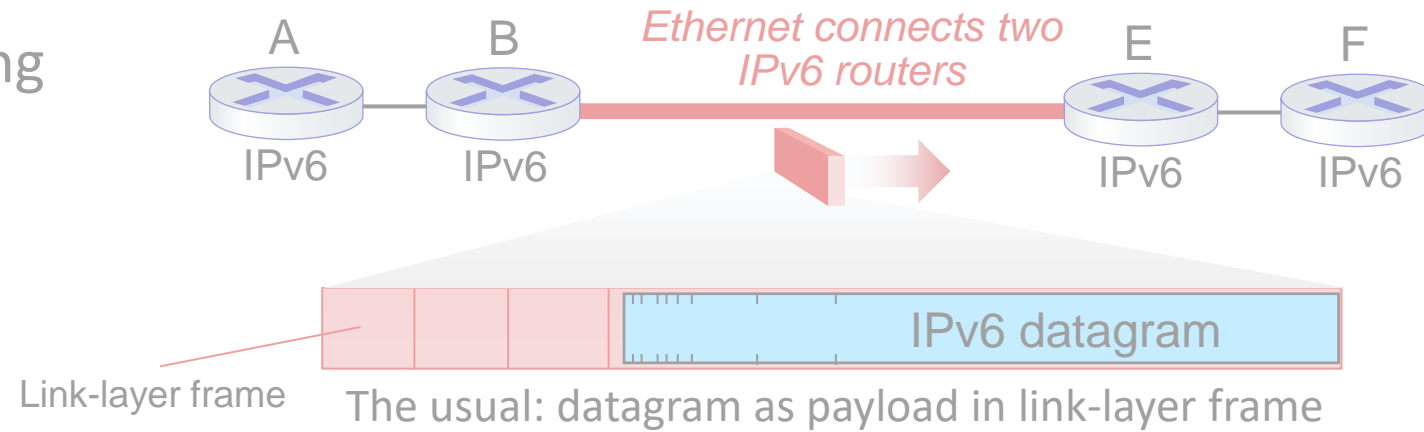
IPv4 network connecting two IPv6 routers



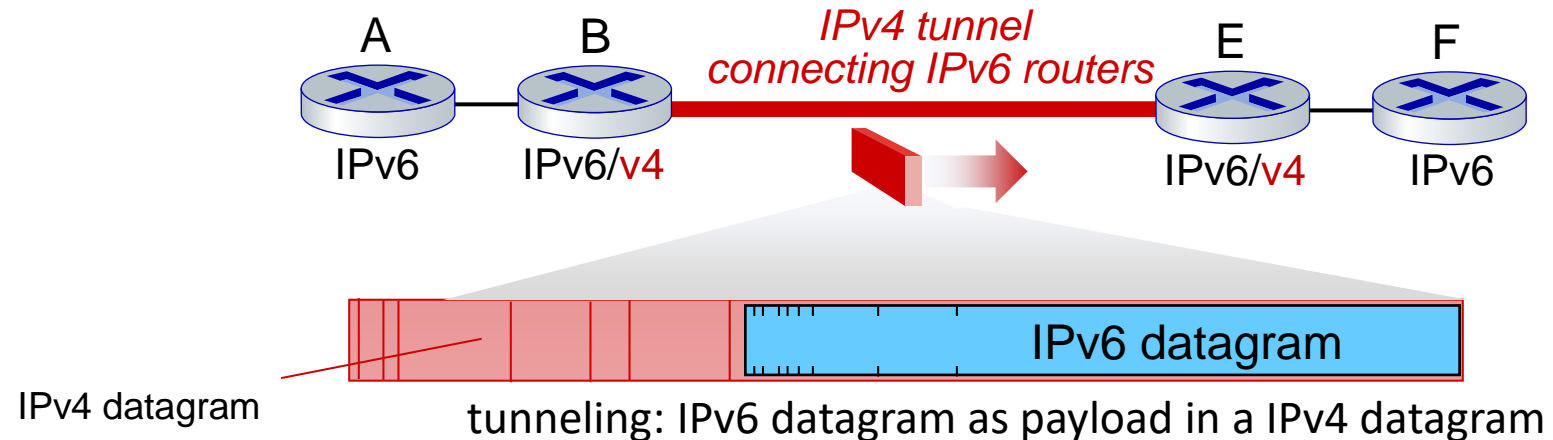


# Tunneling and encapsulation

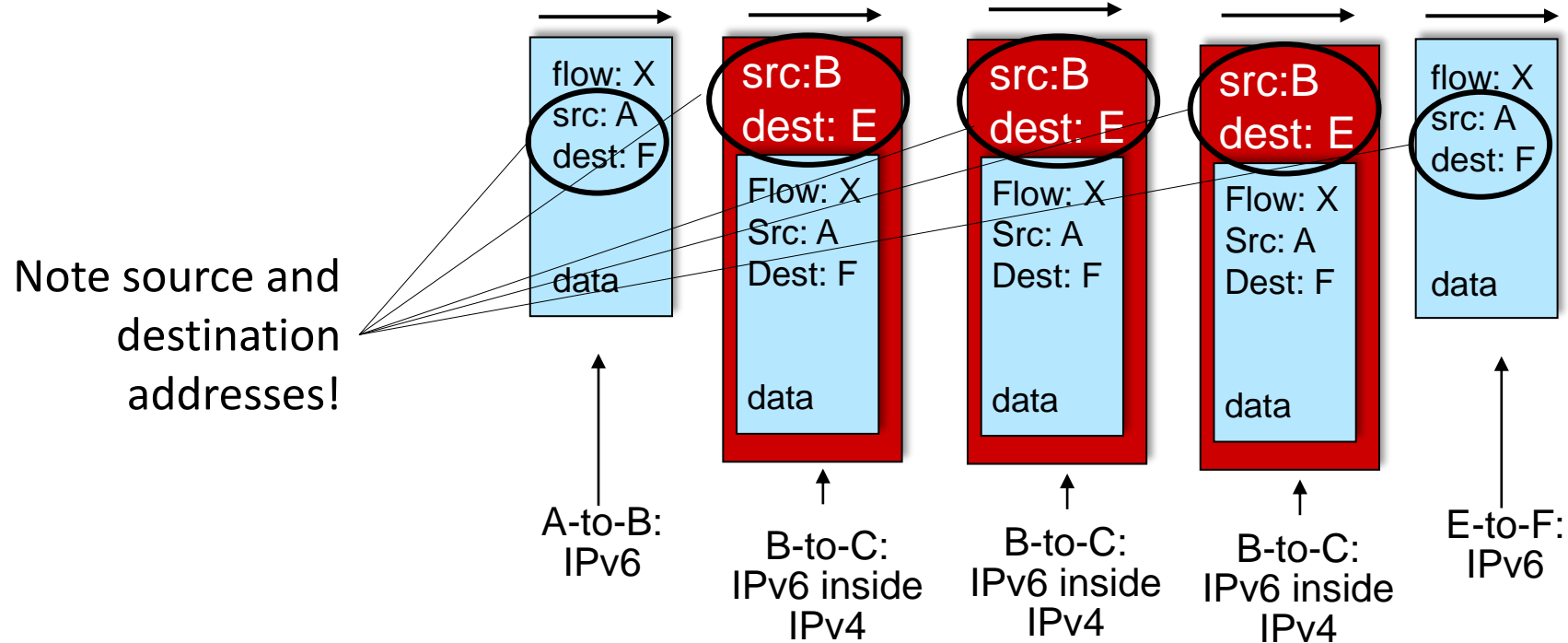
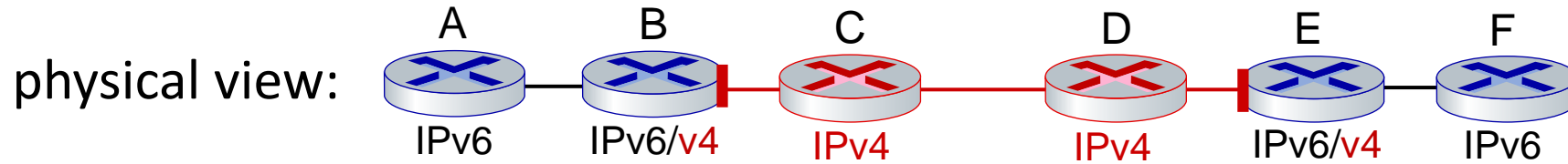
Ethernet connecting two IPv6 routers:



IPv4 tunnel connecting two IPv6 routers



# Tunneling

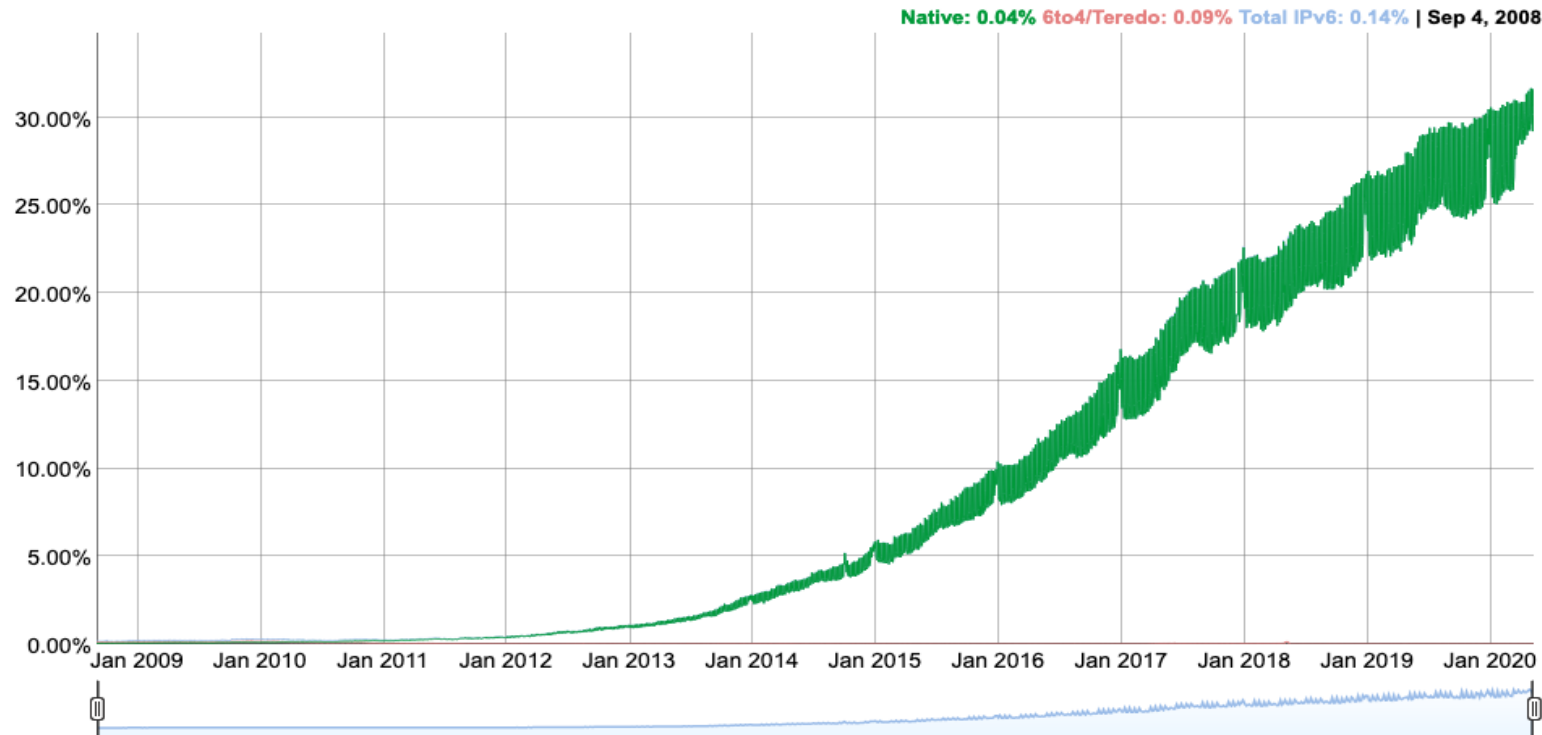


# IPv6: adoption

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

## IPv6 Adoption

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



1

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

# IPv6: adoption

- Google<sup>1</sup>: ~ 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?*

<sup>1</sup> <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 ports, switching, output ports
  - buffer management, scheduling
- IP: the Internet Protocol
  - datagram format
  - addressing
  - network address translation
  - IPv6

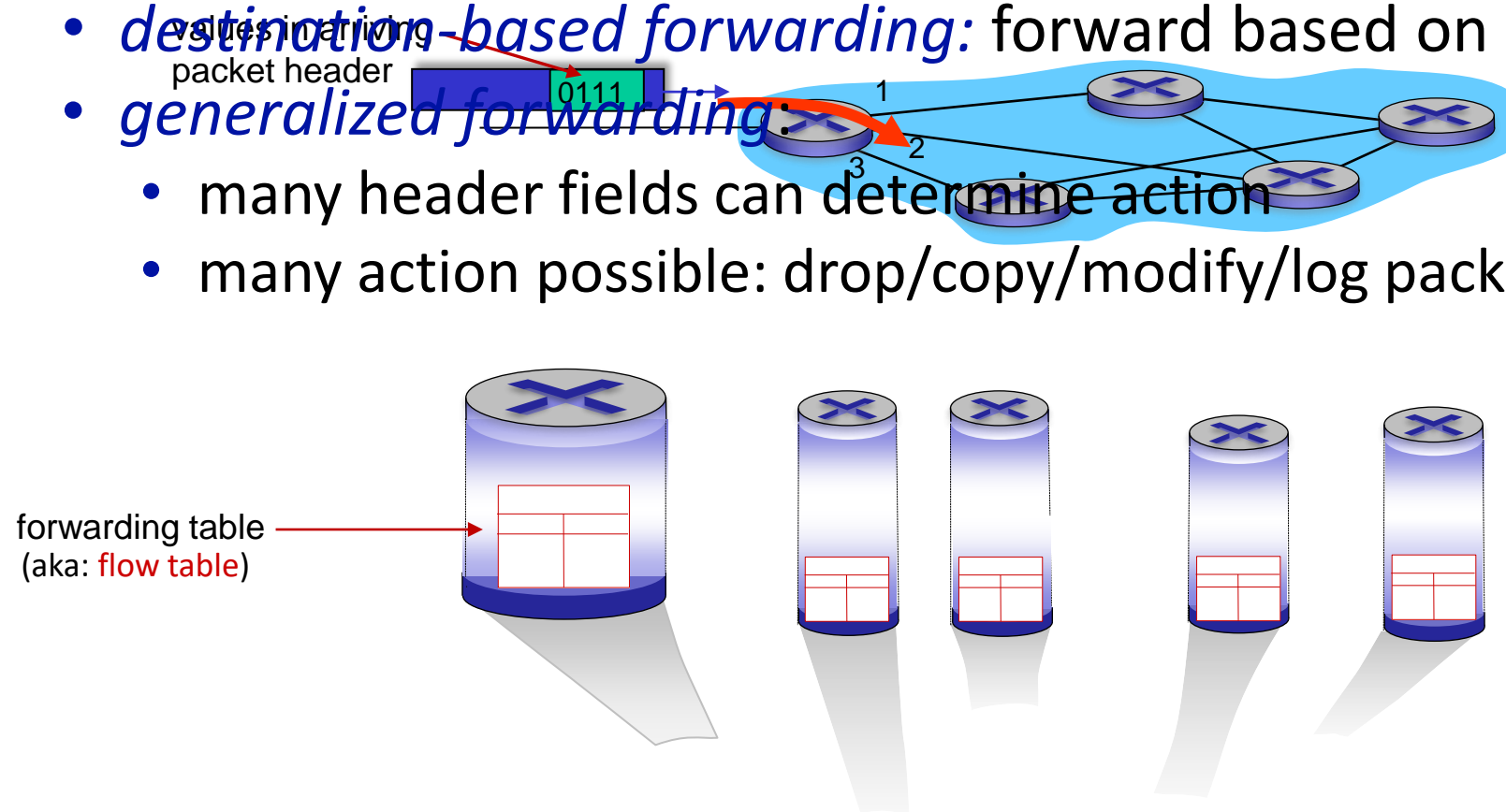


- Generalized Forwarding, SDN
  - Match+action
  - OpenFlow: match+action in action
- Middleboxes

# Generalized forwarding: match plus action

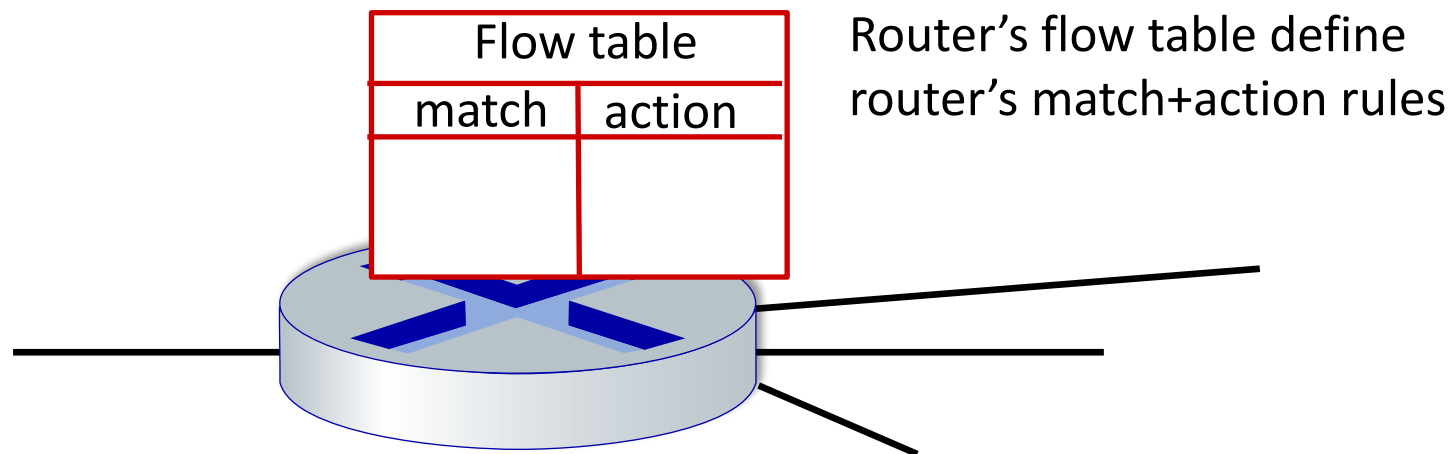
*Review:* each router contains a **forwarding table** (aka: **flow table**)

- “**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



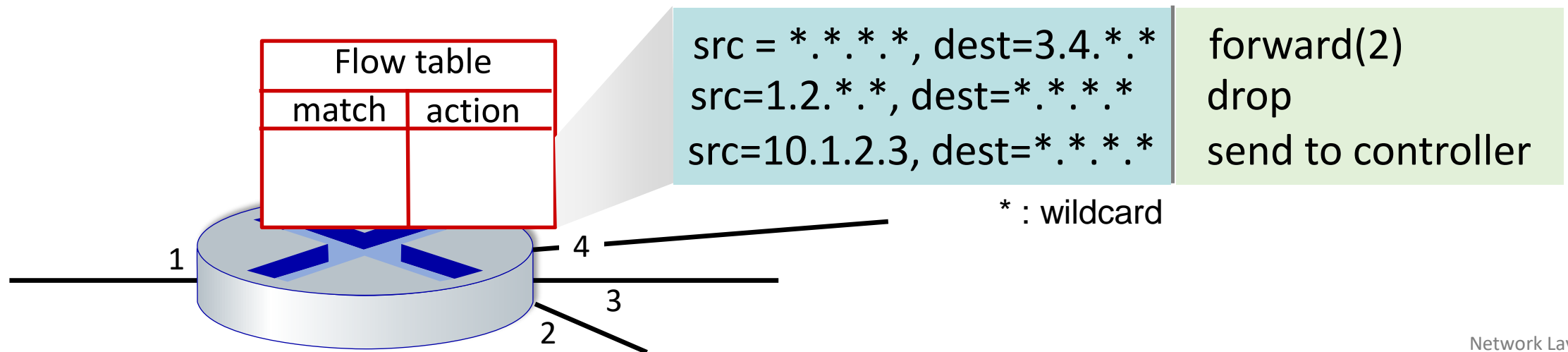
# Flow table abstraction

- **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



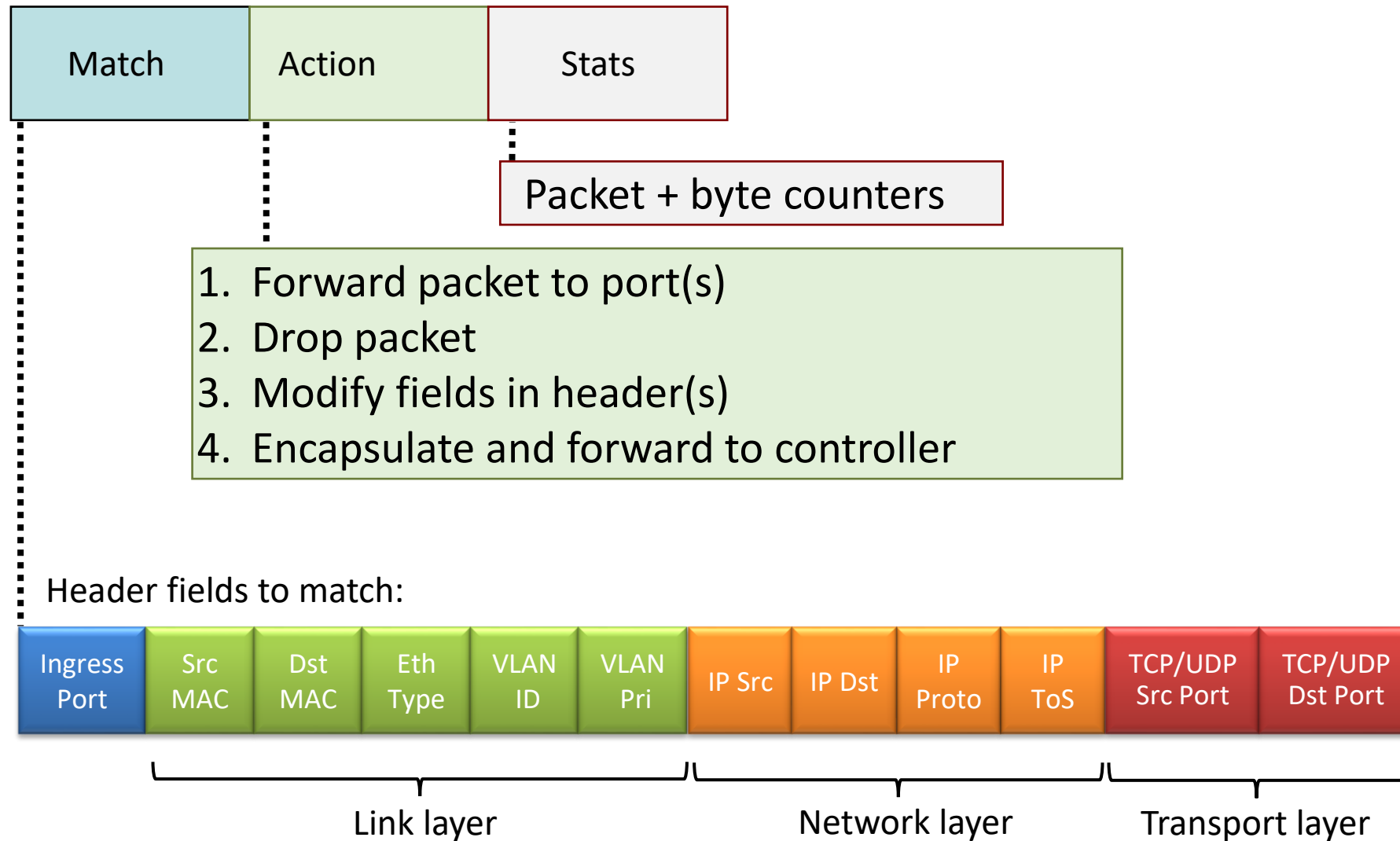
# Flow table abstraction

- **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:<br>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

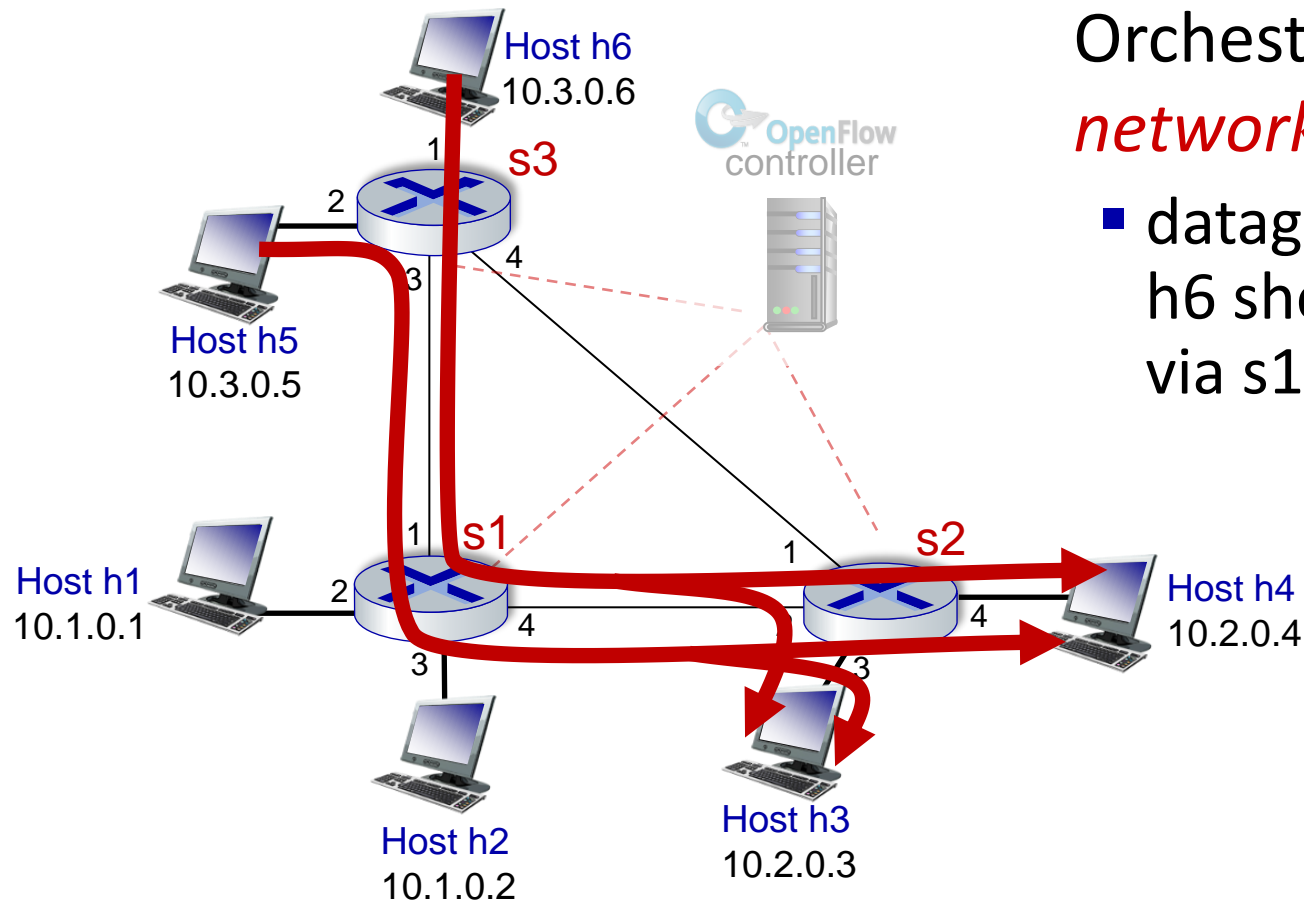
## Firewall

- *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

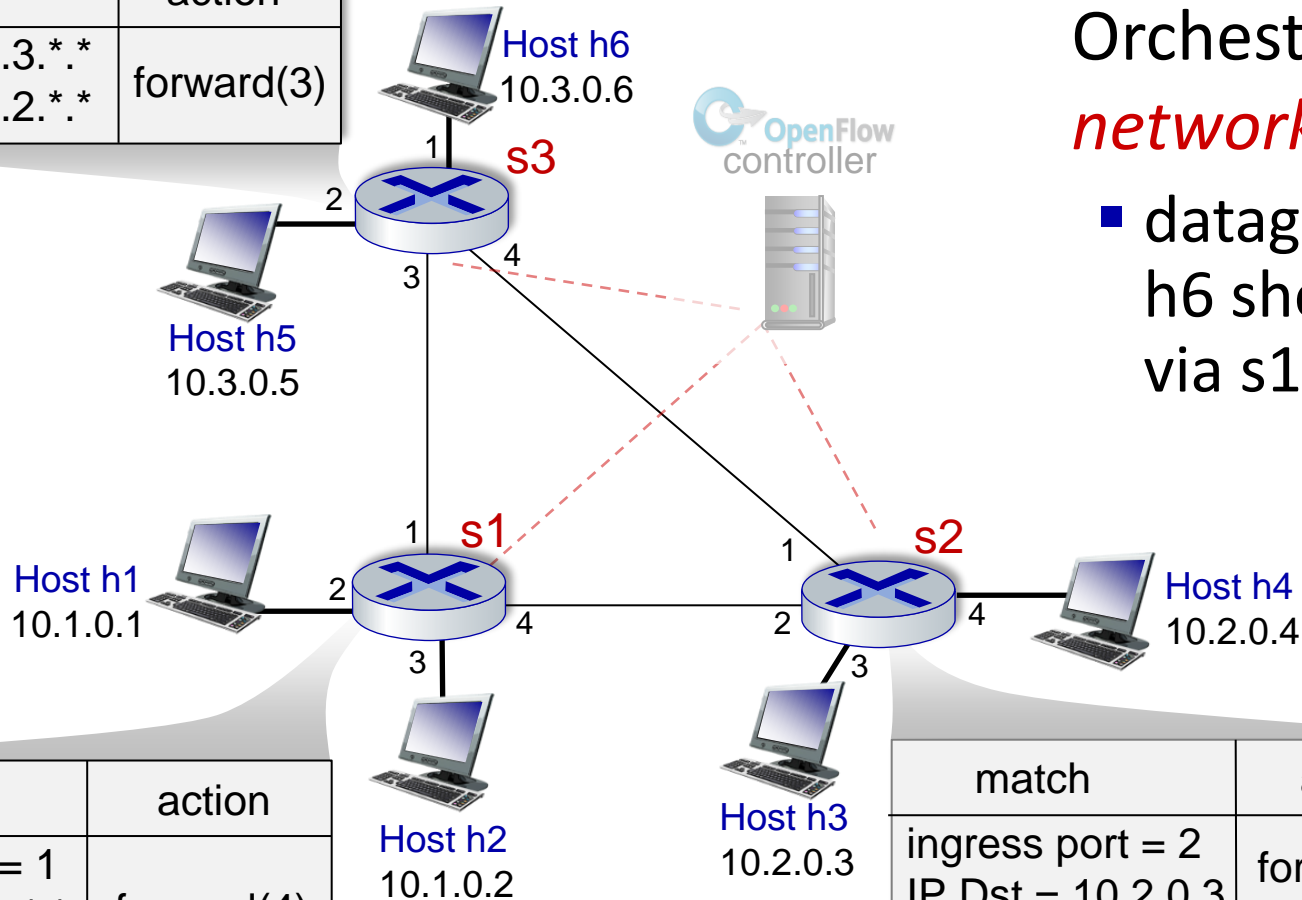


Orchestrated tables can create *network-wide* behavior, e.g.,:

- 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.*.*<br>IP Dst = 10.2.*.* | forward(3) |



| match  | action     |
|--|------------|
| ingress port = 1<br>IP Src = 10.3.*.*<br>IP Dst = 10.2.*.* | forward(4) |

| match                                 | action     |
|---------------------------------------|------------|
| ingress port = 2<br>IP Dst = 10.2.0.3 | forward(3) |
| ingress port = 2<br>IP Dst = 10.2.0.4 | forward(4) |

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](http://p4.org)).

# Network layer: “data plane” roadmap

- Network layer: overview
- What’s inside a router
- IP: the Internet Protocol
- Generalized Forwarding
- **Middleboxes**
  - middlebox functions
  - evolution, architectural principles of the Internet





# Middleboxes

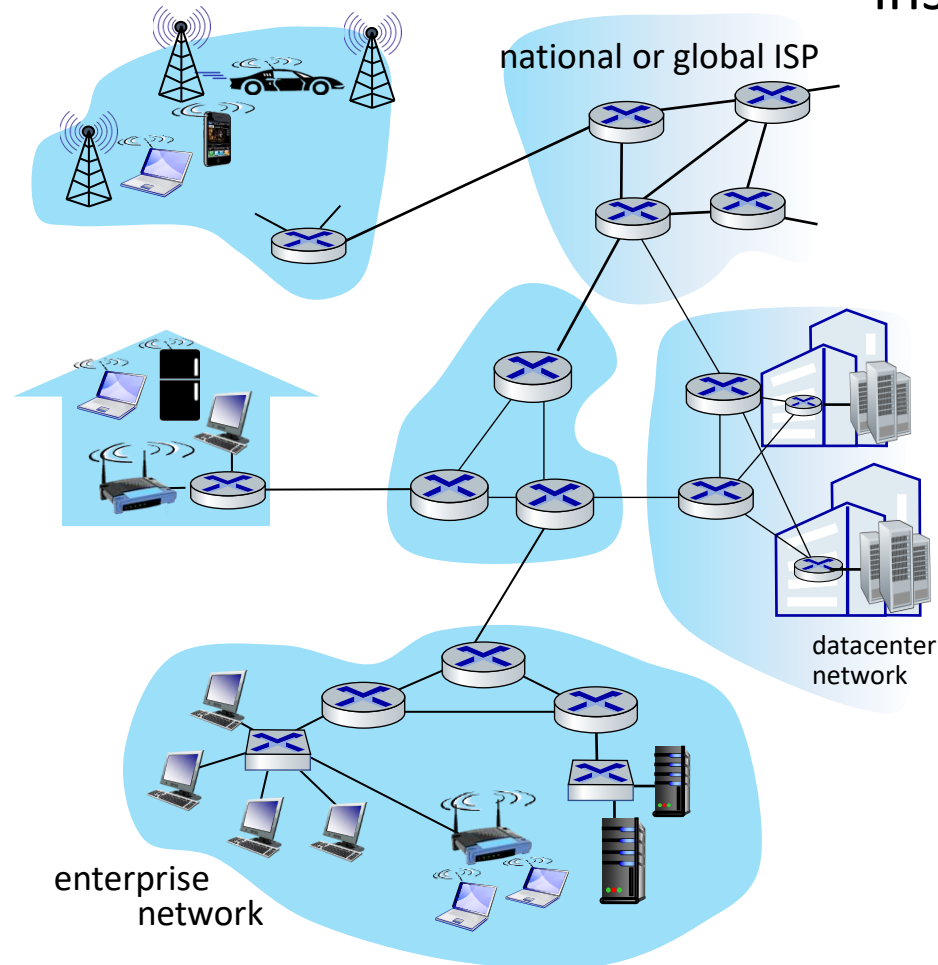
Middlebox (RFC 3234)

“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

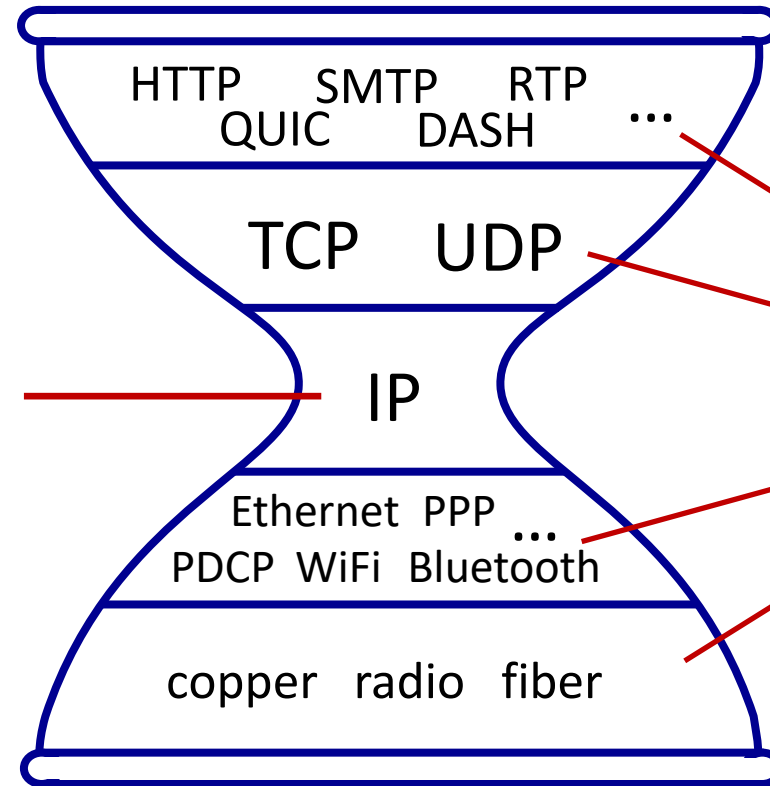
# Middleboxes

- 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

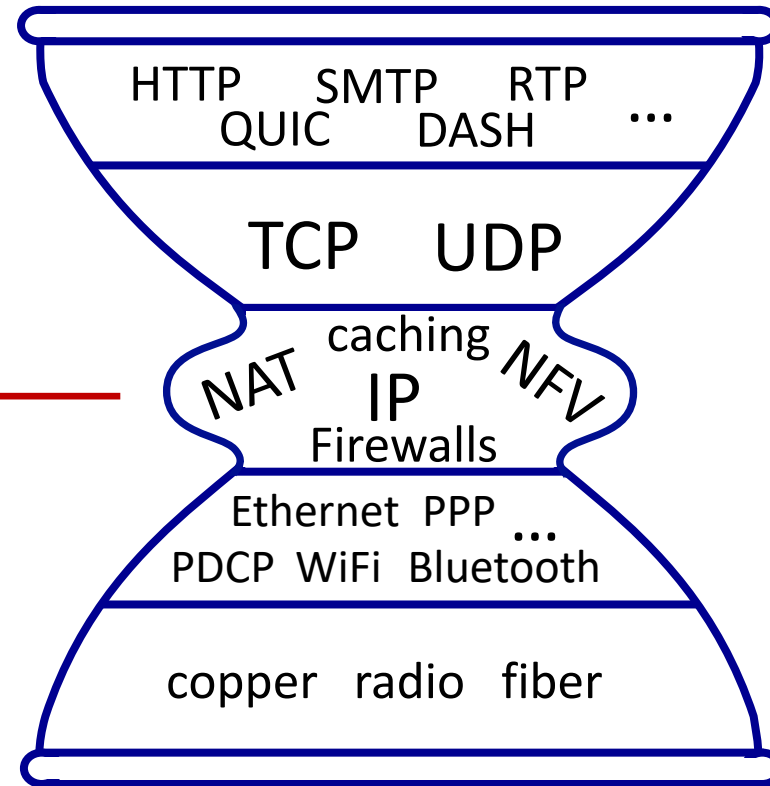


*many* protocols in physical, link, transport, and application layers

# The IP hourglass, at middle age

Internet's middle age  
"love handles"?

- middleboxes, — operating inside the network



# Architectural Principles of the Internet

## RFC 1958

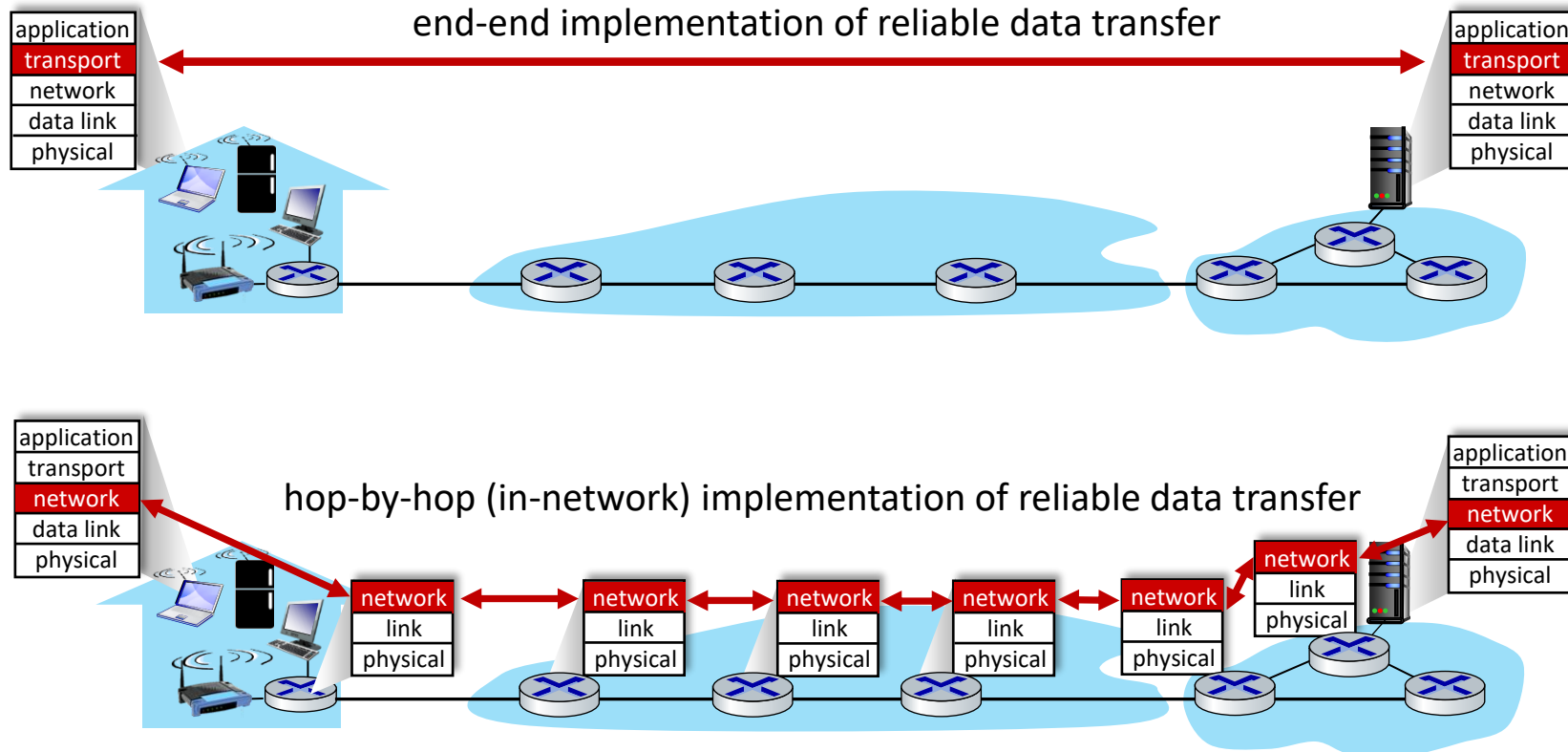
“Many members of the Internet community would argue that there is no architecture, but only a tradition, which was not written down for the first 25 years (or at least not by the IAB). However, in very general terms, the community believes that **the goal is connectivity, the tool is the Internet Protocol,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

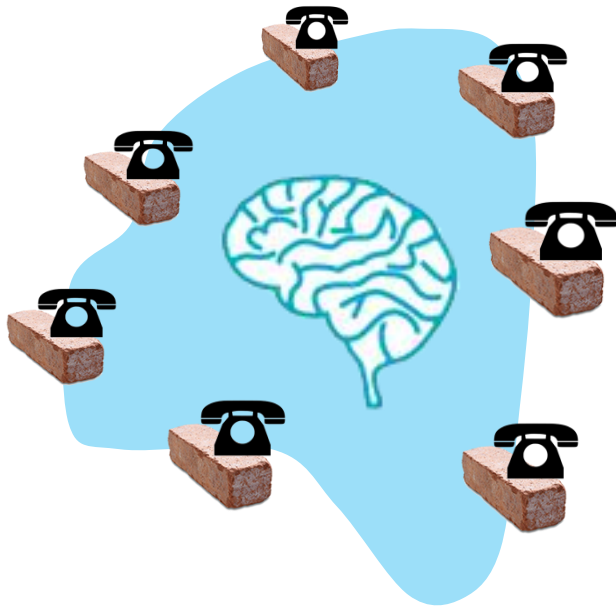
“The function in question can completely and correctly be implemented only with the knowledge and help of the application standing at the end points of the communication system. Therefore, providing that questioned function as a feature of the communication system itself is not possible. (Sometimes an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

Saltzer, Reed, Clark 1981

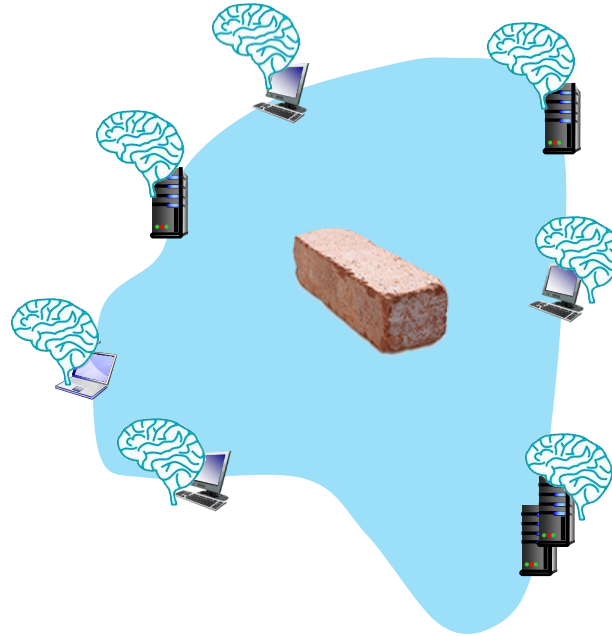


# Where's the intelligence?



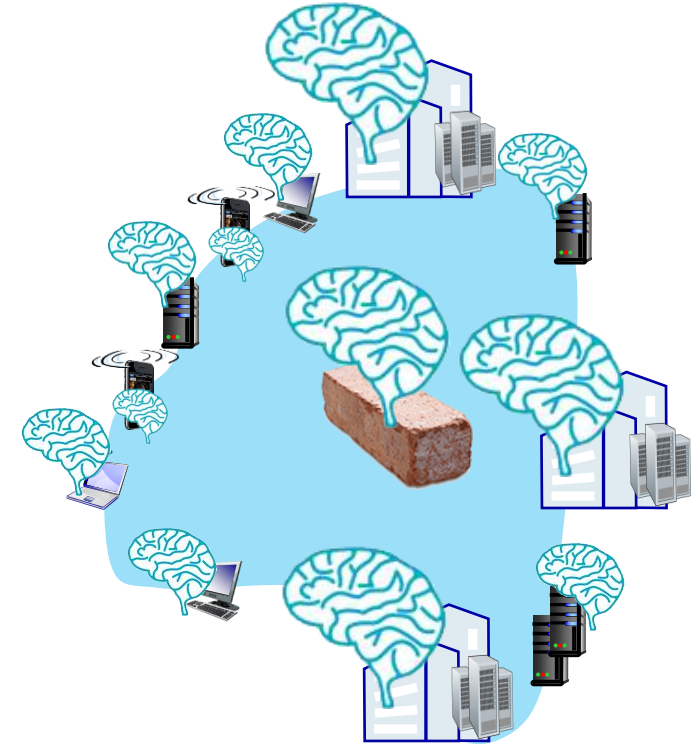
## 20<sup>th</sup> century phone net:

- 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

# Chapter 4: done!

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



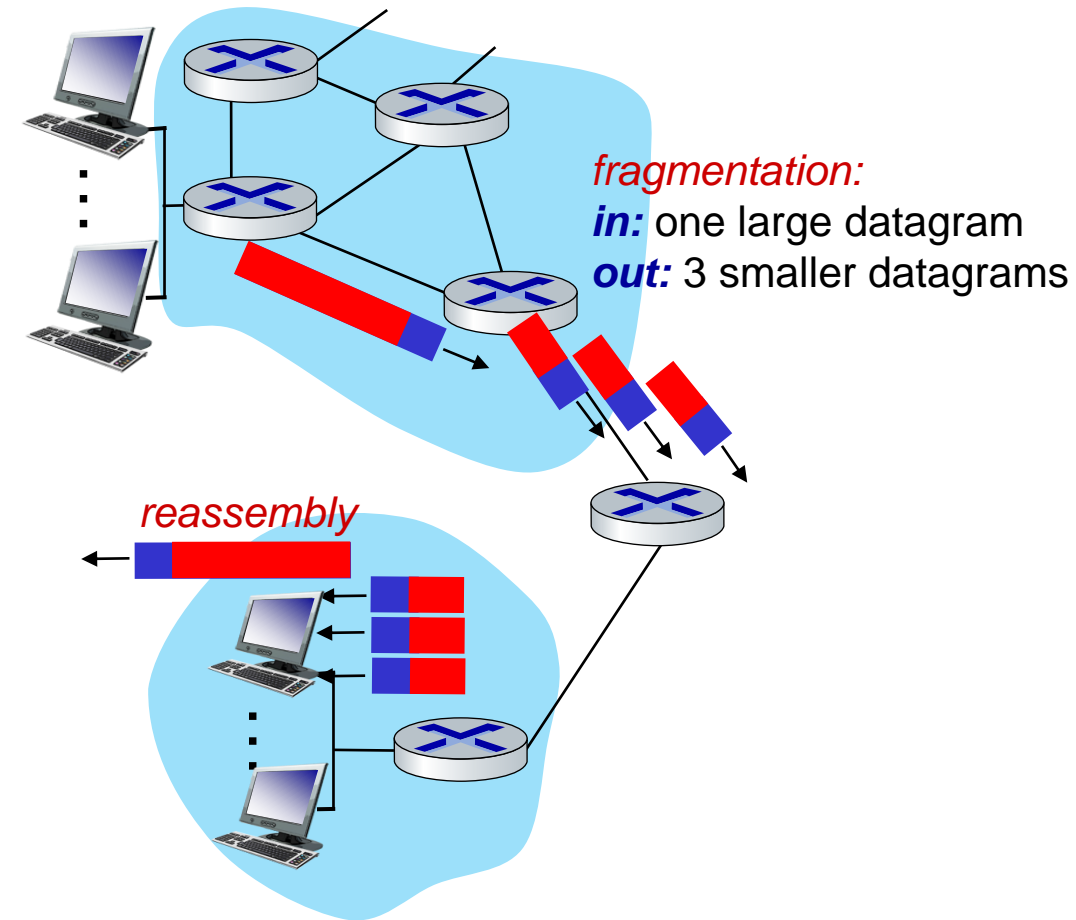
*Question:* how are forwarding tables (destination-based forwarding) or flow tables (generalized forwarding) computed?

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

# Additional Chapter 4 slides

# 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

|  |        |    |          |        |  |
|--|--------|----|----------|--------|--|
|  | length | ID | fragflag | offset |  |
|  | =4000  | =x | =0       | =0     |  |

*one large datagram becomes  
several smaller datagrams*

1480 bytes in  
data field

offset =  
 $1480/8$

|  |        |    |          |        |  |
|--|--------|----|----------|--------|--|
|  | length | ID | fragflag | offset |  |
|  | =1500  | =x | =1       | =0     |  |

|  |        |    |          |        |  |
|--|--------|----|----------|--------|--|
|  | length | ID | fragflag | offset |  |
|  | =1500  | =x | =1       | =185   |  |

|  |        |    |          |        |  |
|--|--------|----|----------|--------|--|
|  | length | ID | fragflag | offset |  |
|  | =1040  | =x | =0       | =370   |  |

# DHCP: Wireshark output (home LAN)

Message type: **Boot Request (1)**

Hardware type: Ethernet

Hardware address length: 6

Hops: 0

request

**Transaction ID: 0x6b3a11b7**

Seconds elapsed: 0

Bootp flags: 0x0000 (Unicast)

Client IP address: 0.0.0.0 (0.0.0.0)

Your (client) IP address: 0.0.0.0 (0.0.0.0)

Next server IP address: 0.0.0.0 (0.0.0.0)

Relay agent IP address: 0.0.0.0 (0.0.0.0)

**Client MAC address: Wistron\_23:68:8a (00:16:d3:23:68:8a)**

Server host name not given

Boot file name not given

Magic cookie: (OK)

Option: (t=53,l=1) **DHCP Message Type = DHCP Request**

Option: (61) Client identifier

Length: 7; Value: 010016D323688A;

Hardware type: Ethernet

Client MAC address: Wistron\_23:68:8a (00:16:d3:23:68:8a)

Option: (t=50,l=4) Requested IP Address = 192.168.1.101

Option: (t=12,l=5) Host Name = "nomad"

**Option: (55) Parameter Request List**

Length: 11; Value: 010F03062C2E2F1F21F92B

**1 = Subnet Mask; 15 = Domain Name**

**3 = Router; 6 = Domain Name Server**

**44 = NetBIOS over TCP/IP Name Server**

.....

Message type: **Boot Reply (2)**

Hardware type: Ethernet

Hardware address length: 6

Hops: 0

reply

**Transaction ID: 0x6b3a11b7**

Seconds elapsed: 0

Bootp flags: 0x0000 (Unicast)

**Client IP address: 192.168.1.101 (192.168.1.101)**

Your (client) IP address: 0.0.0.0 (0.0.0.0)

**Next server IP address: 192.168.1.1 (192.168.1.1)**

Relay agent IP address: 0.0.0.0 (0.0.0.0)

Client MAC address: Wistron\_23:68:8a (00:16:d3:23:68:8a)

Server host name not given

Boot file name not given

Magic cookie: (OK)

**Option: (t=53,l=1) DHCP Message Type = DHCP ACK**

**Option: (t=54,l=4) Server Identifier = 192.168.1.1**

**Option: (t=1,l=4) Subnet Mask = 255.255.255.0**

**Option: (t=3,l=4) Router = 192.168.1.1**

**Option: (6) Domain Name Server**

**Length: 12; Value: 445747E2445749F244574092;**

**IP Address: 68.87.71.226;**

**IP Address: 68.87.73.242;**

**IP Address: 68.87.64.146**

**Option: (t=15,l=20) Domain Name = "hsd1.ma.comcast.net."**