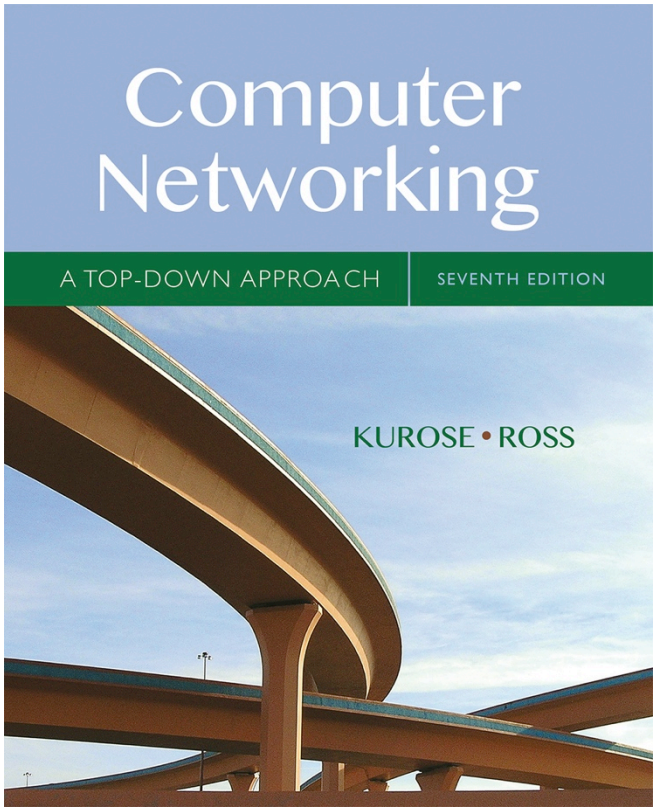
# Chapter 4
# Network Layer: The Data Plane




*Computer Networking: A Top Down Approach*

7th edition
Jim Kurose, Keith Ross
Pearson/Addison Wesley
April 2016

A note on the use of these Powerpoint slides:

We're making these slides freely available to all (faculty, students, readers). They're in PowerPoint form so you see the animations; and can add, modify, and delete slides (including this one) and slide content to suit your needs. They obviously represent a *lot* of work on our part. In return for use, we only ask the following:

- If you use these slides (e.g., in a class) that you mention their source (after all, we'd like people to use our book!)
- If you post any slides on a www site, that you note that they are adapted from (or perhaps identical to) our slides, and note our copyright of this material.

Thanks and enjoy! JFK/KWR

# Chapter 4: outline

4.1 Overview of Network layer
- data plane
- control plane

4.2 What's inside a router

4.3 IP: Internet Protocol
- datagram format
- fragmentation
- IPv4 addressing
- network address translation
- IPv6

4.4 Generalized Forward and SDN
- match
- action
- OpenFlow examples of match-plus-action in action

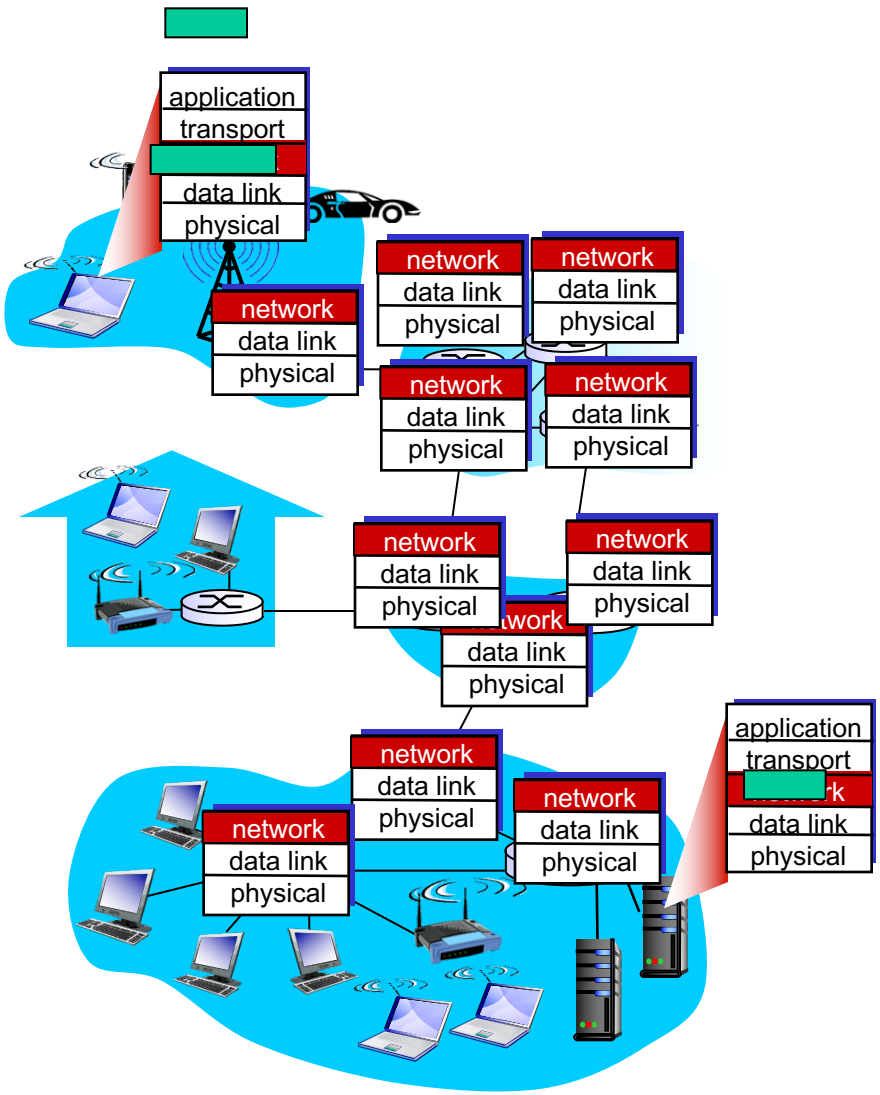
# Chapter 4: network layer

*chapter goals:*

- understand principles behind network layer services, focusing on data plane:
  - network layer service models
  - forwarding versus routing
  - how a router works
  - generalized forwarding
- instantiation, implementation in the Internet

# Network layer

- transport segment from sending to receiving host
- on sending side encapsulates segments into datagrams
- on receiving side, delivers segments to transport layer
- network layer protocols in *every* host, router
- router examines header fields in all IP datagrams passing through it



# Two key network-layer functions

*network-layer functions:*

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

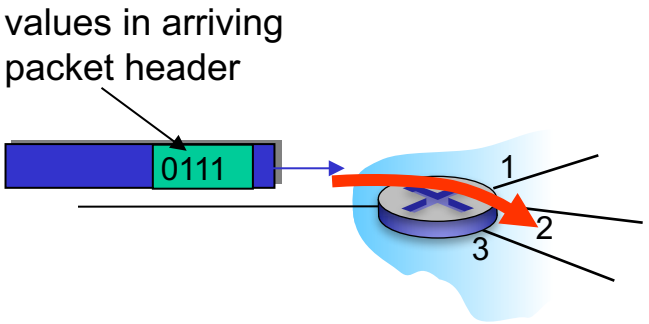
*analogy: taking a trip*

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

# 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
- forwarding function

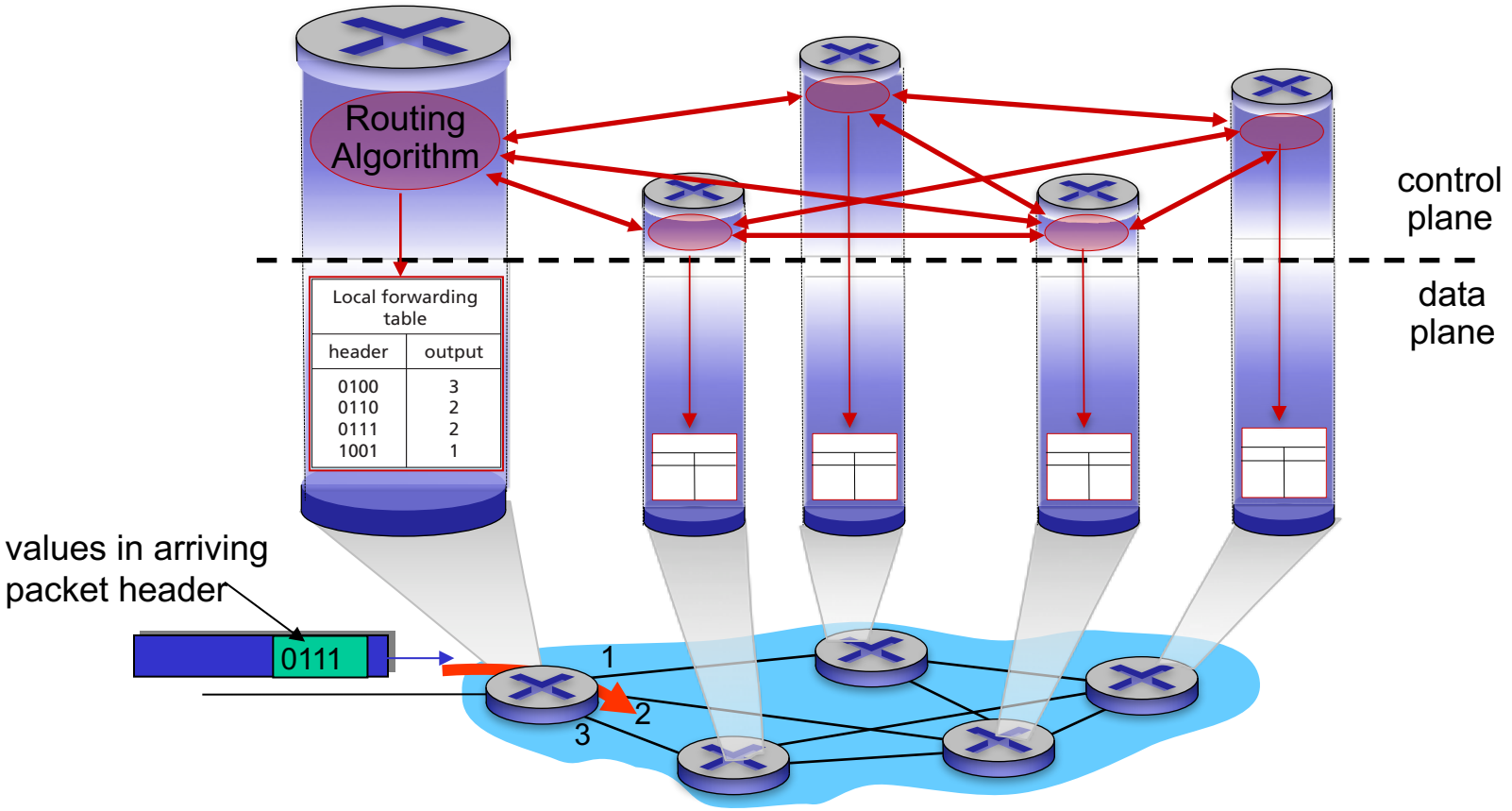



## *Control plane*

- 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

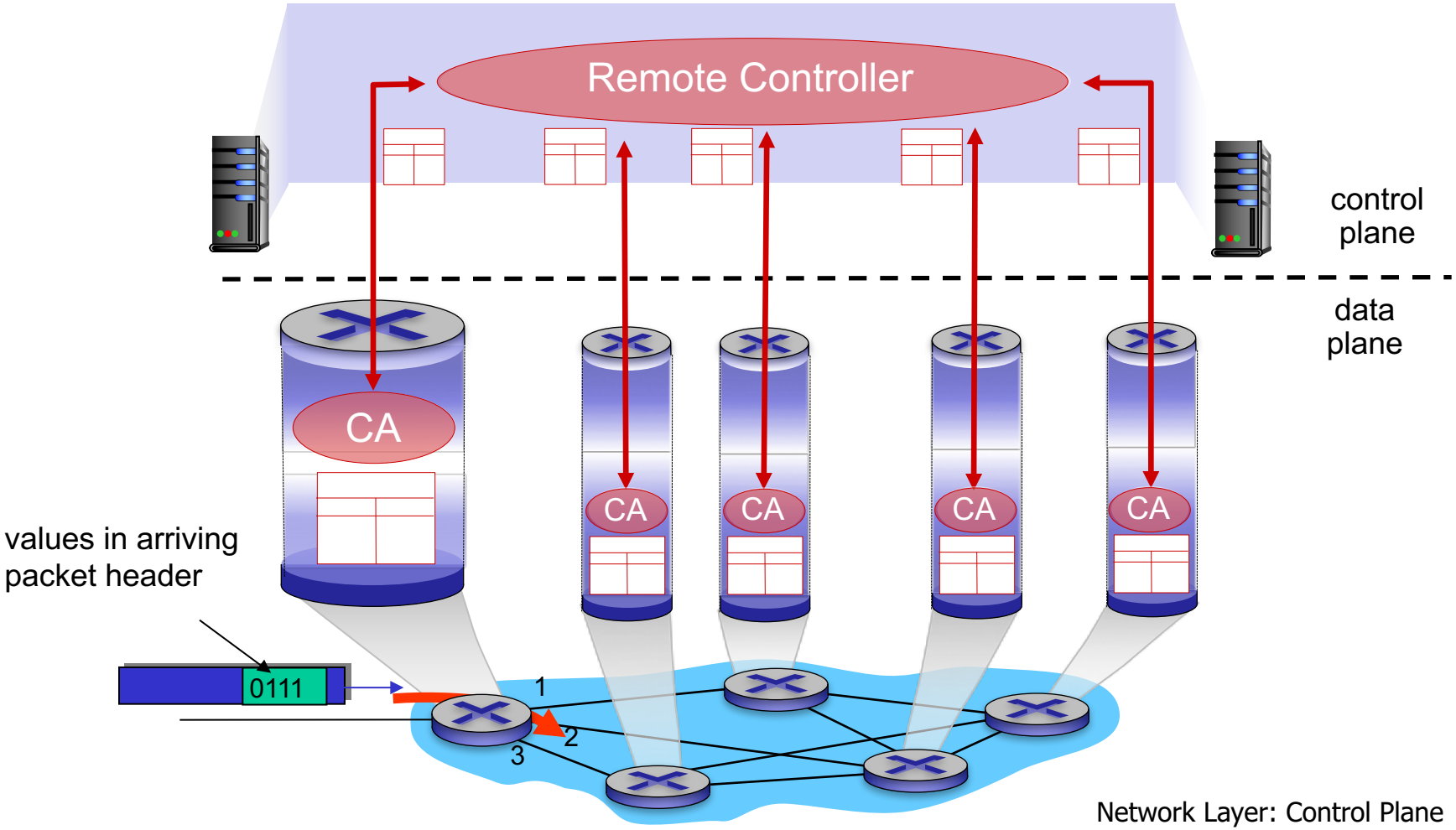
# Per-router control plane

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



# Logically centralized control plane

A distinct (typically remote) controller interacts with local control agents (CAs)



# 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 models:

| Network Architecture | Service Model | Guarantees ? | | | | Congestion feedback |
|---|---|---|---|---|---|---|
| | | Bandwidth | Loss | Order | Timing | |
| Internet | best effort | none | no | no | no | no (inferred via loss) |
| ATM | CBR | constant rate | yes | yes | yes | no congestion |
| ATM | VBR | guaranteed rate | yes | yes | yes | no congestion |
| ATM | ABR | guaranteed minimum | no | yes | no | yes |
| ATM | UBR | none | no | yes | no | no |

# Chapter 4: outline

4.1 Overview of Network layer
- data plane
- control plane

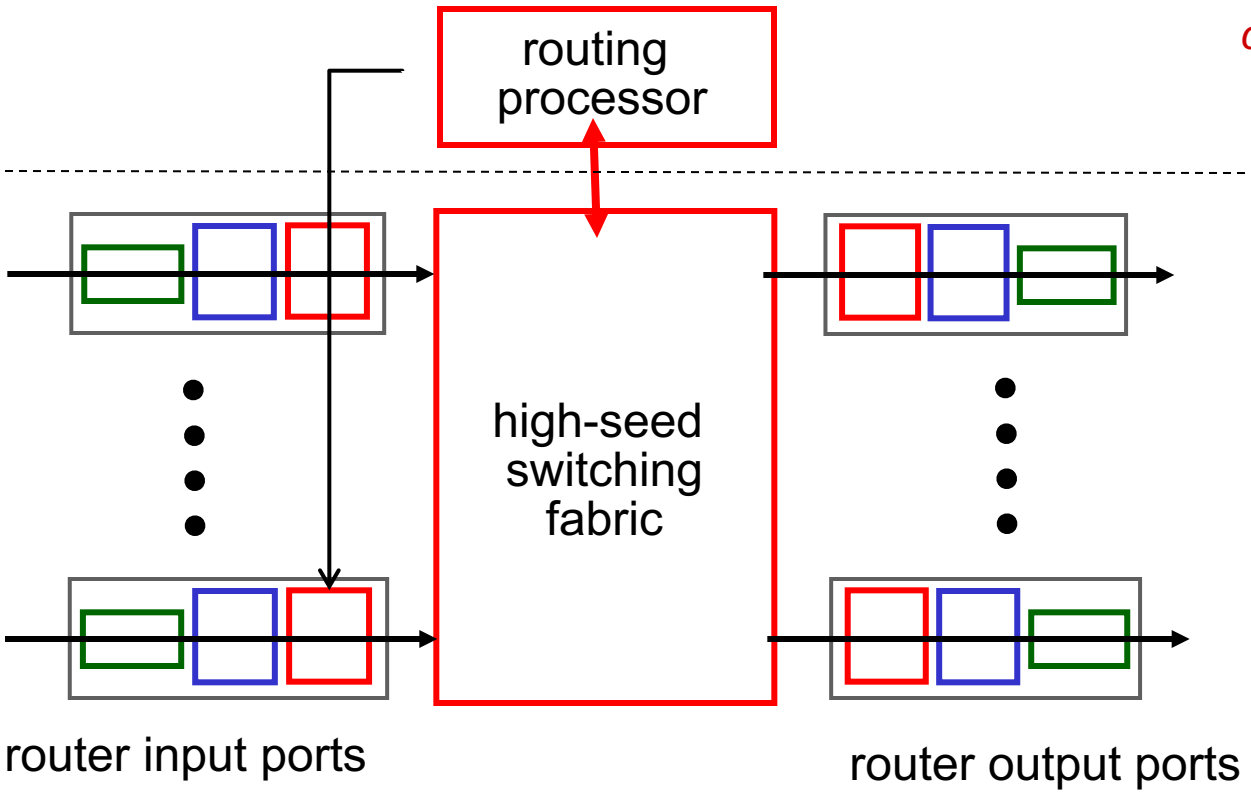
4.2 What's inside a router

4.3 IP: Internet Protocol
- datagram format
- fragmentation
- IPv4 addressing
- network address translation
- IPv6

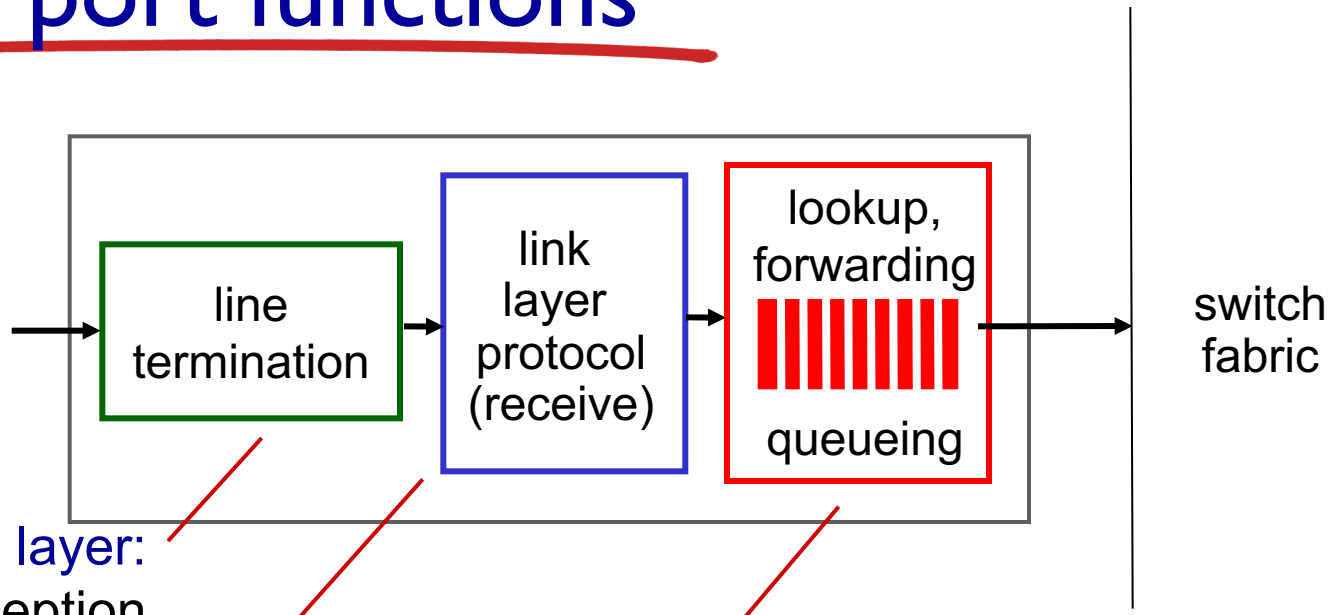
4.4 Generalized Forward and SDN
- match
- action
- OpenFlow examples of match-plus-action in action

# Router architecture overview

- high-level view of generic router architecture:



# Input port functions



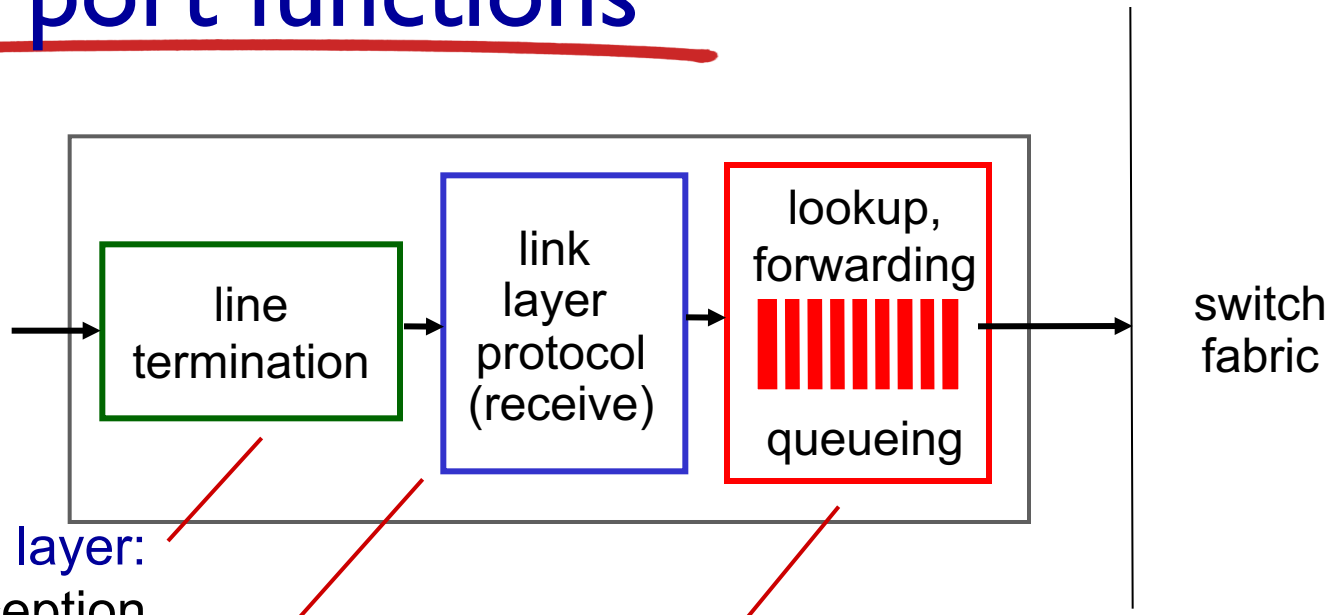

physical layer:
bit-level reception

data link layer:
e.g., Ethernet
see chapter 5

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'
- queuing: if datagrams arrive faster than forwarding rate into switch fabric

# Input port functions




physical layer:
bit-level reception

data link layer:
e.g., Ethernet
see chapter 5

**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

*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 |

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

# Longest prefix matching

*longest prefix matching*

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:

DA: 11001000 00010111 00010110 10100001 which interface?

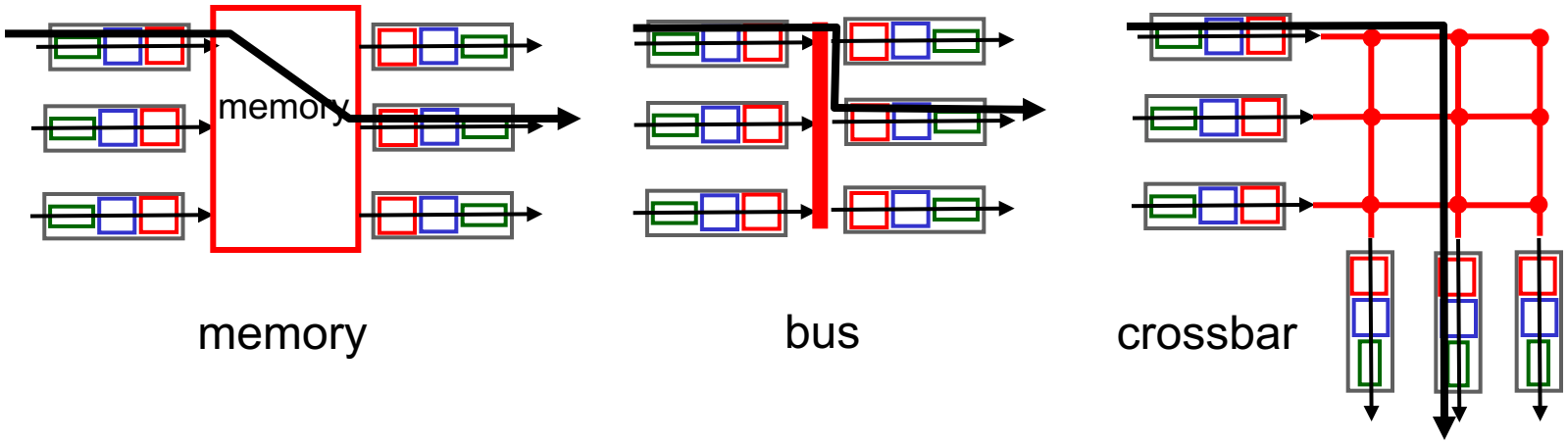
DA: 11001000 00010111 00011000 10101010 which interface?

# 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: can up ~1M routing table entries in TCAM

# Switching fabrics

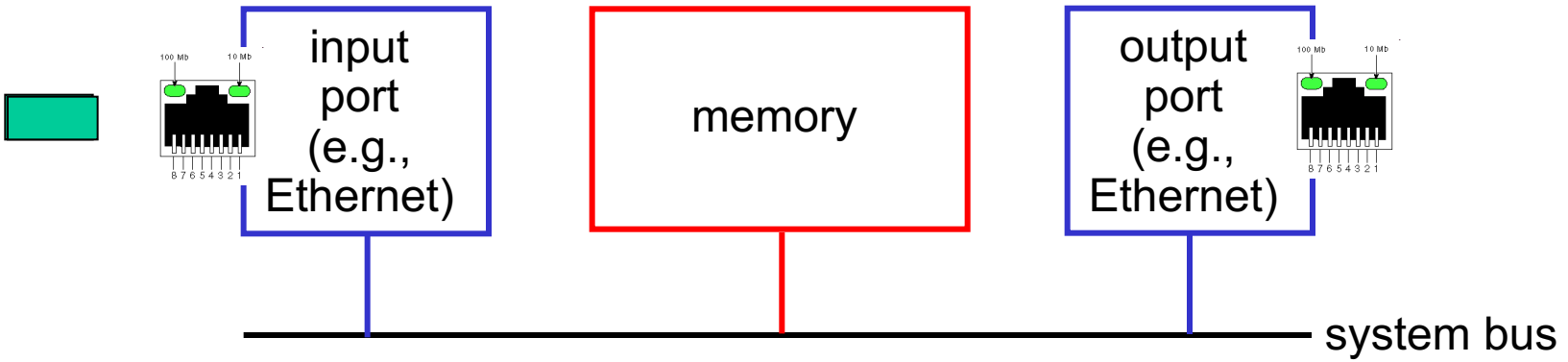
- transfer packet from input buffer to appropriate output buffer
- switching rate: rate at which packets can be transfer from inputs to outputs
  - often measured as multiple of input/output line rate
  - N inputs: switching rate N times line rate desirable
- three types of switching fabrics



# Switching via memory

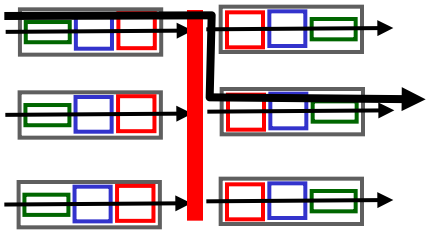
*first generation routers:*

- 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

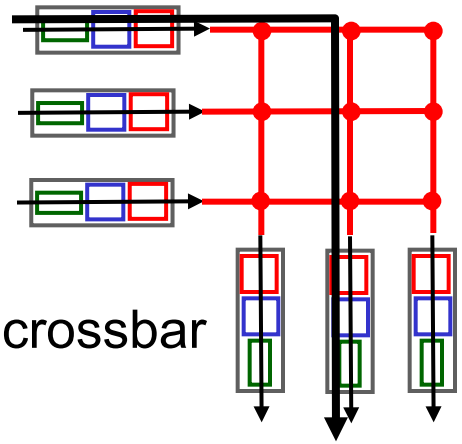
- datagram from input port memory to output port memory via a shared bus
- *bus contention:* switching speed limited by bus bandwidth
- 32 Gbps bus, Cisco 5600: sufficient speed for access and enterprise routers



bus

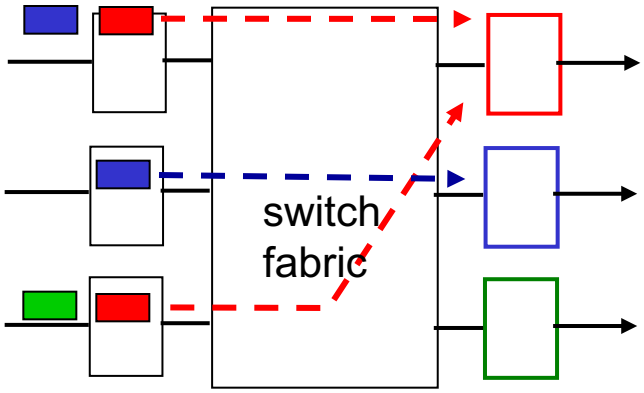
# Switching via interconnection network

- overcome  bus bandwidth limitations
- banyan networks, crossbar, other interconnection nets initially developed to connect processors in multiprocessor
- advanced design: fragmenting datagram into fixed length cells, switch cells through the fabric.
- Cisco 12000: switches 60 Gbps through the interconnection network

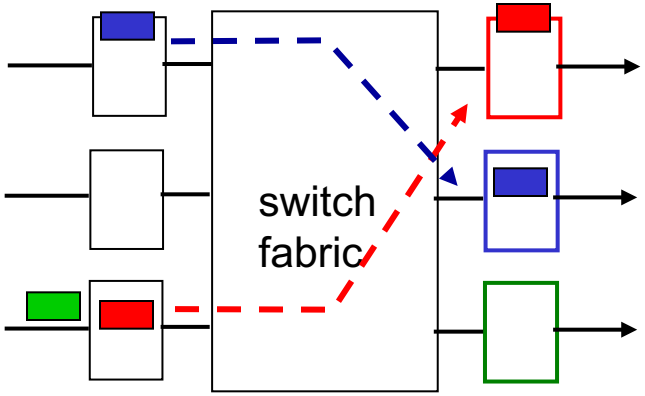


# Input port queuing

- 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



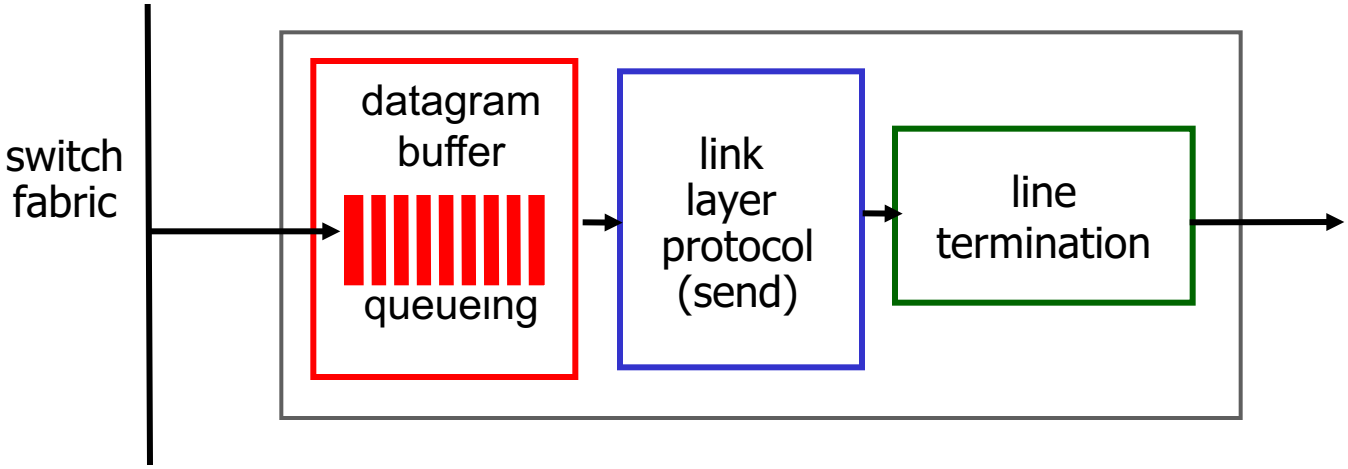

output port contention:
only one red datagram can be transferred.
*lower red packet is blocked*




one packet time later:
green packet
experiences HOL
blocking

# Output ports

*This slide in HUGELY important!*



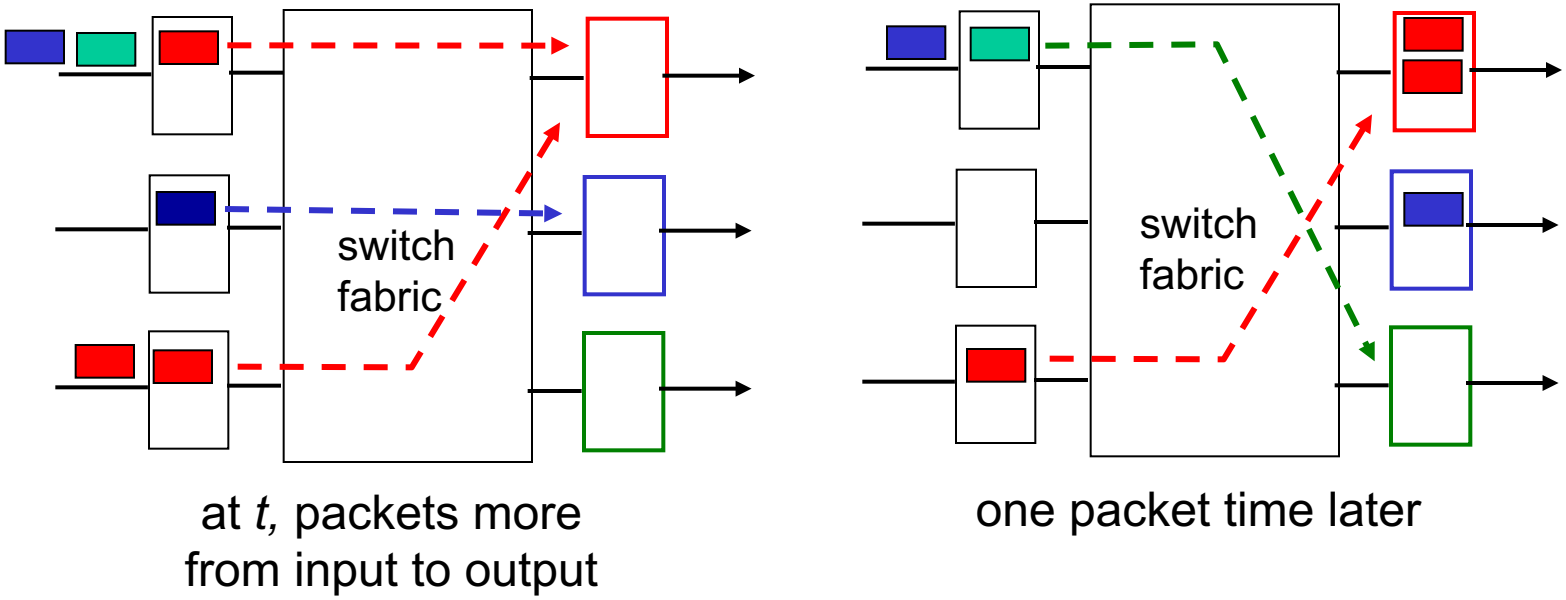

- *buffering* required from fabric faster rate

Datagram (packets) can be lost due to congestion, lack of buffers

- *scheduling* datagrams

Priority scheduling – who gets best performance, network neutrality

# Output port queueing




at *t*, packets more from input to output

one packet time later

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

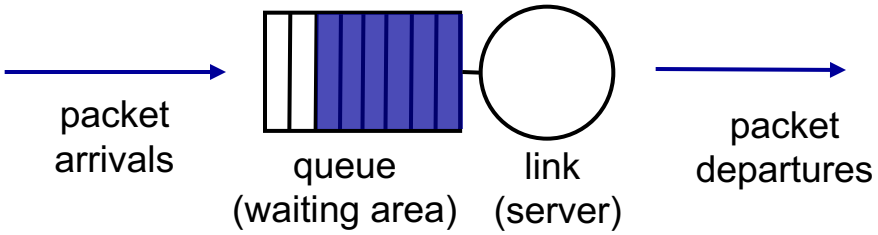
# How much buffering?

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

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

# Scheduling mechanisms

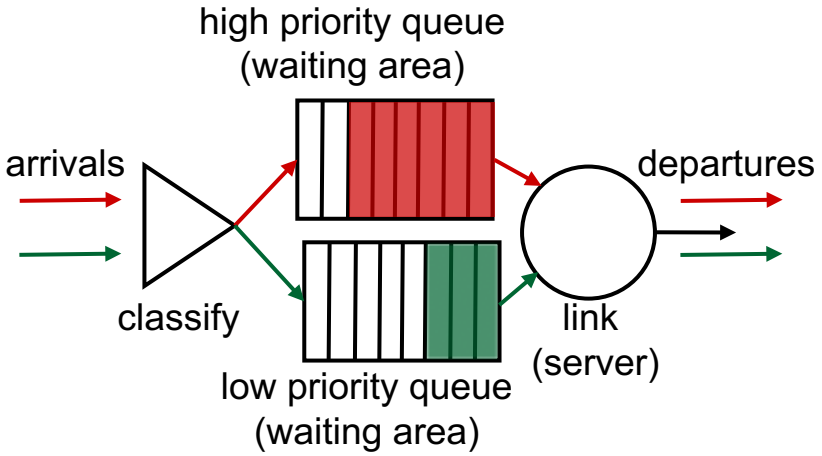
- *scheduling:* choose next packet to send on link
- *FIFO (first in first out) scheduling:* send in order of arrival to queue
  - real-world example?
  - *discard policy:* if packet arrives to full queue: who to discard?
    - *tail drop:* drop arriving packet
    - *priority:* drop/remove on priority basis
    - *random:* drop/remove randomly

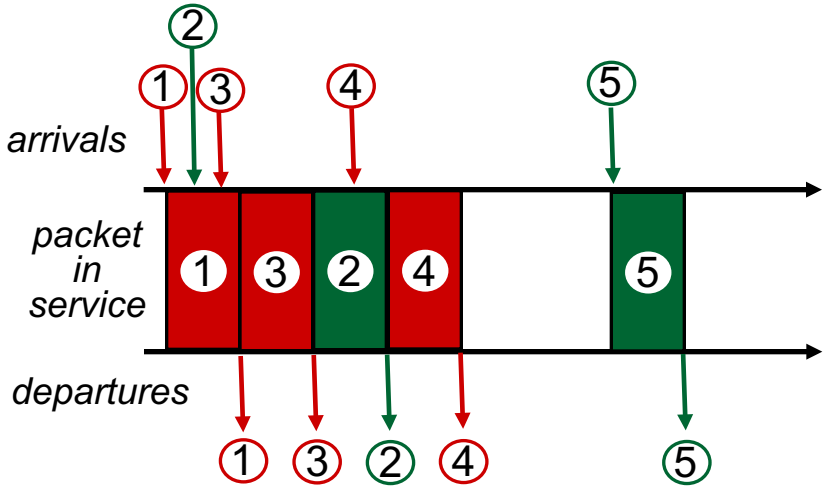


# Scheduling policies: priority

*priority scheduling:* send highest priority queued packet

- multiple *classes*, with different priorities
  - class may depend on marking or other header info, e.g. IP source/dest, port numbers, etc.
  - real world example?

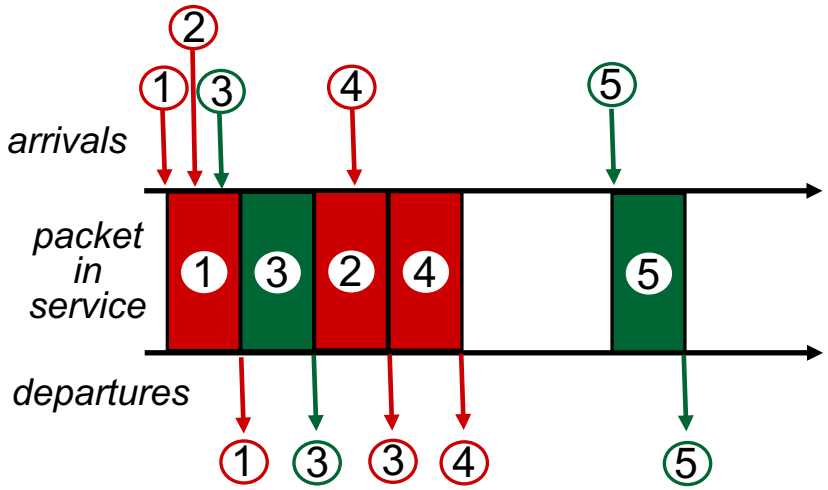


# Scheduling policies: still more

*Round Robin (RR) scheduling:*

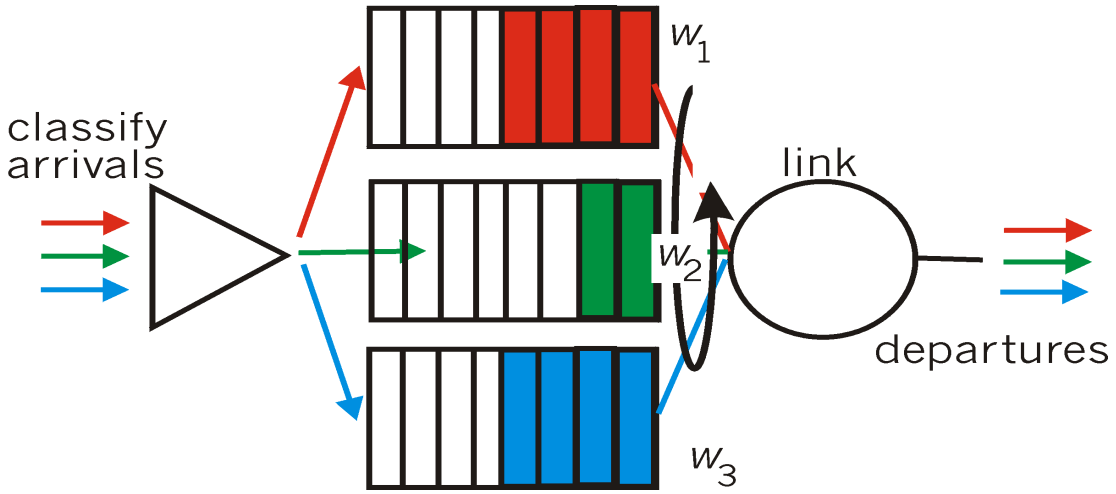
- multiple classes
- cyclically scan class queues, sending one complete packet from each class (if available)
- real world example?



# Scheduling policies: still more

*Weighted Fair Queuing (WFQ):*

- generalized Round Robin
- each class gets weighted amount of service in each cycle
- real-world example?



# Chapter 4: outline

4.1 Overview of Network layer
- data plane
- control plane

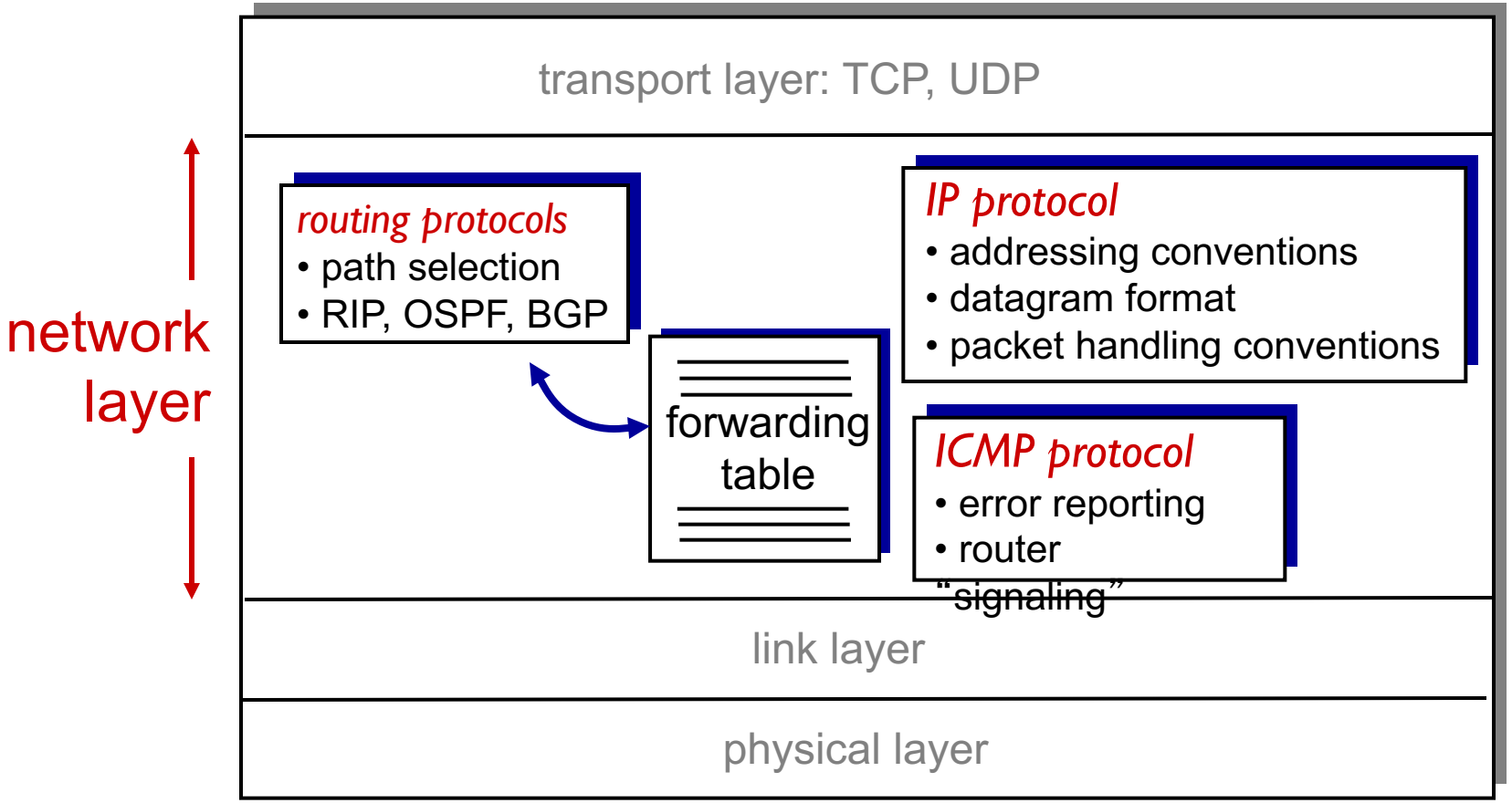
4.2 What's inside a router

4.3 IP: Internet Protocol
- datagram format
- fragmentation
- IPv4 addressing
- network address translation
- IPv6

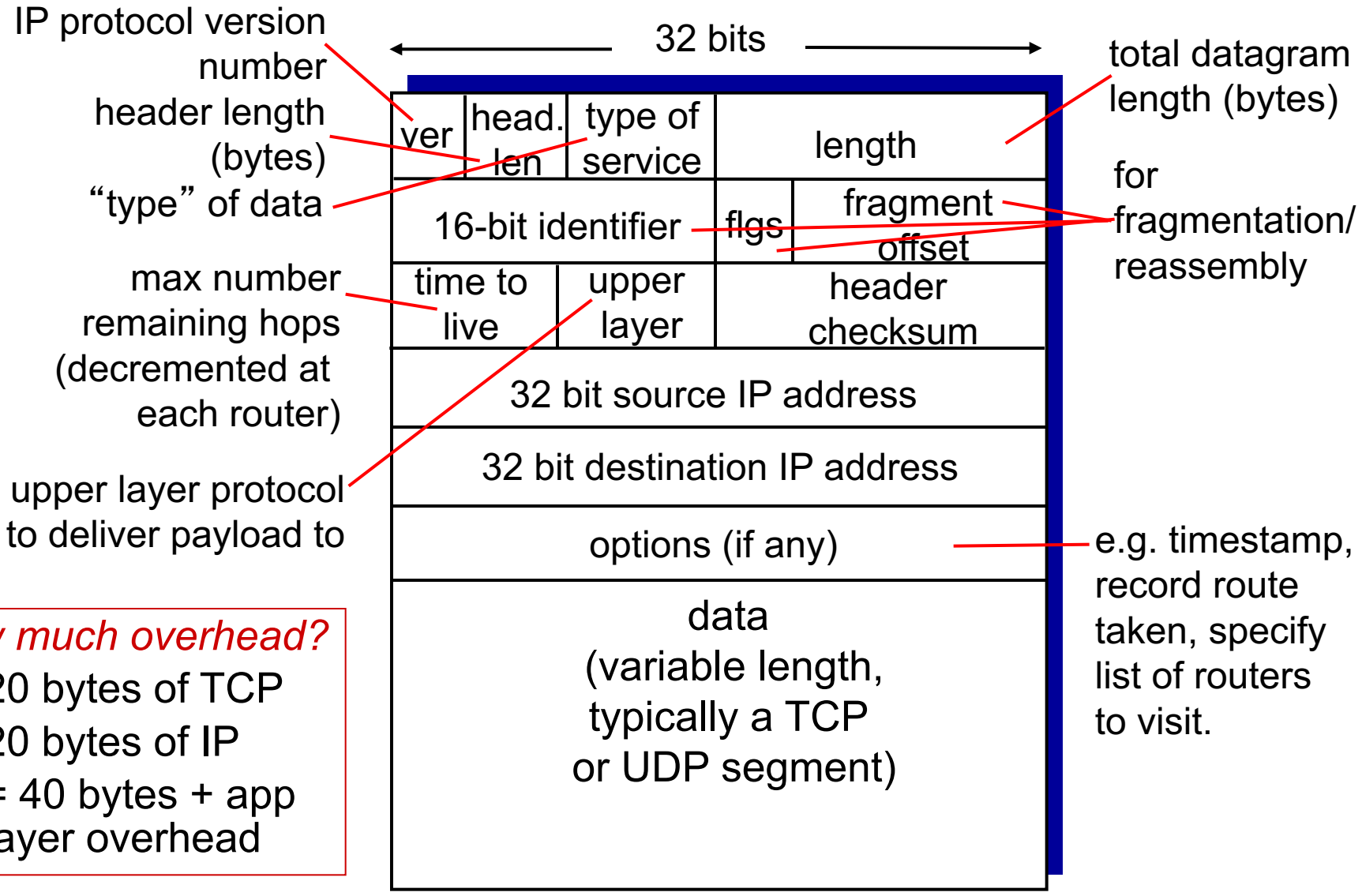
4.4 Generalized Forward and SDN
- match
- action
- OpenFlow examples of match-plus-action in action

# The Internet network layer

host, router network layer functions:



# IP datagram format

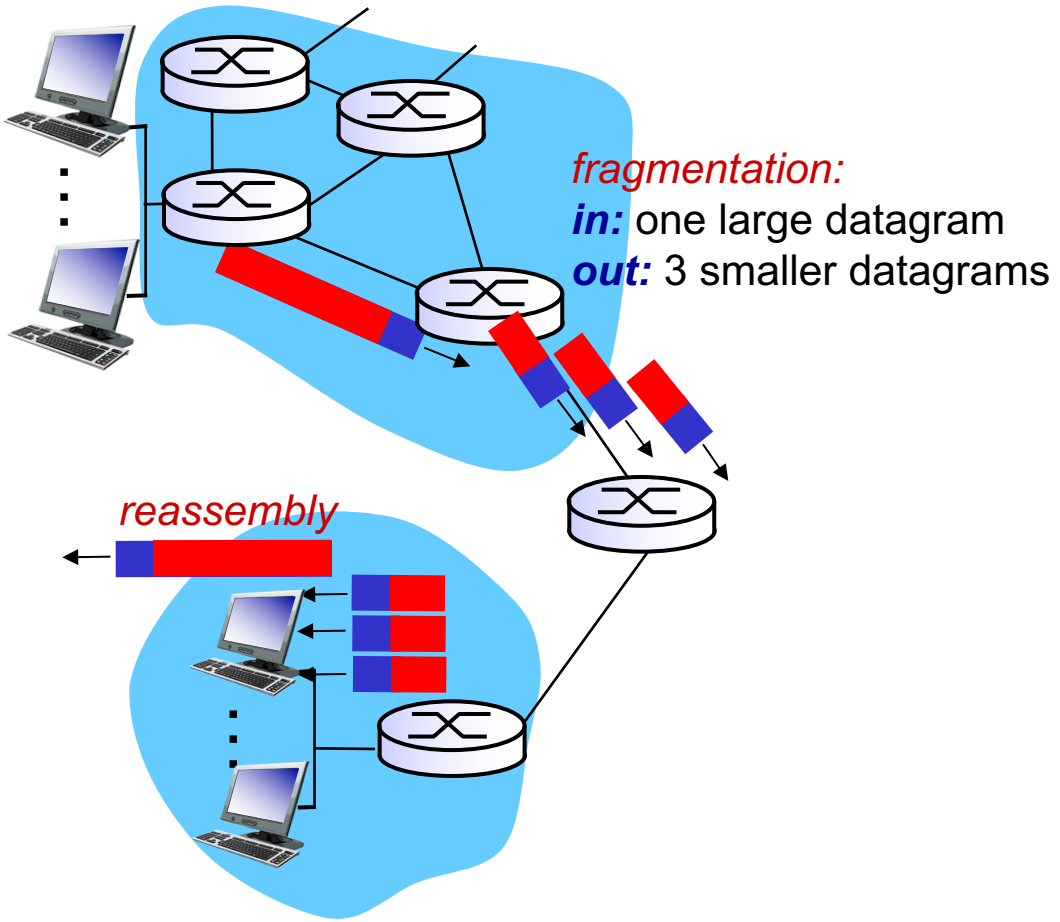



*how much overhead?*

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

# IP fragmentation, reassembly

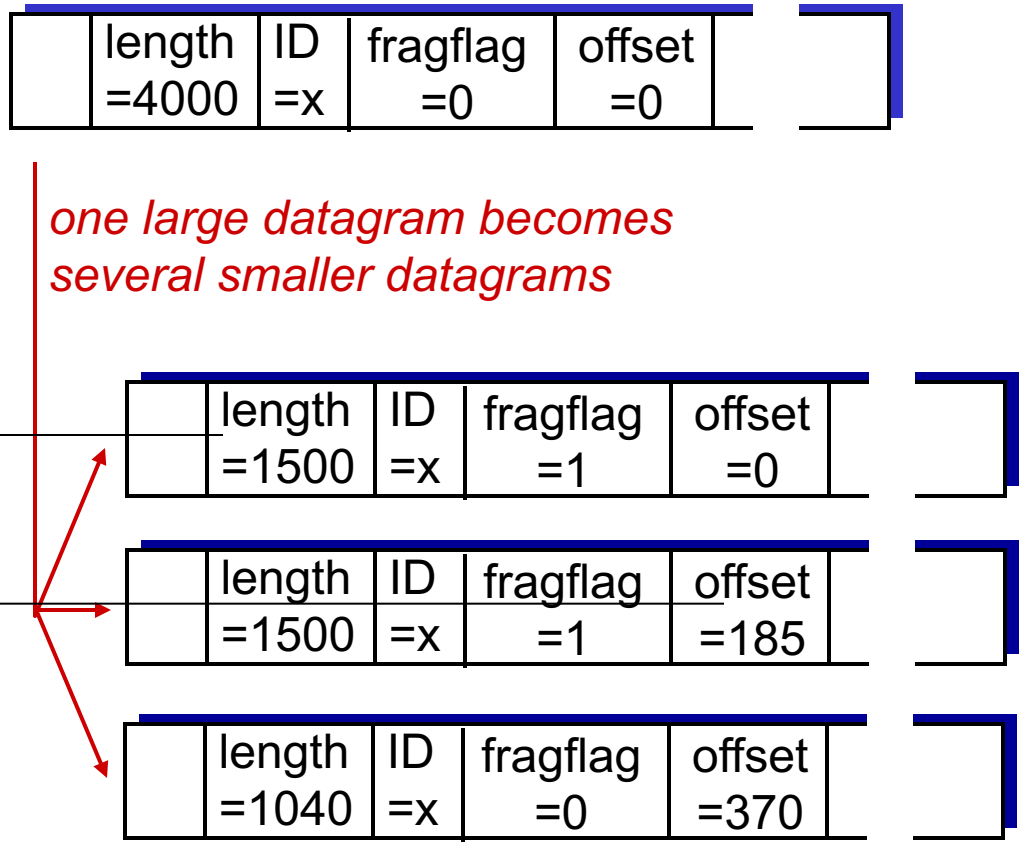
- network links have MTU (max.transfer size) - largest possible link-level frame
  - different link types, different MTUs
- large IP datagram divided ("fragmented") within net
  - one datagram becomes several datagrams
  - "reassembled" only at final destination
  - IP header bits used to identify, order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



# IP fragmentation, reassembly

*example:*

- 4000 byte datagram
- MTU = 1500 bytes



# Chapter 4: outline

4.1 Overview of Network layer
- data plane
- control plane

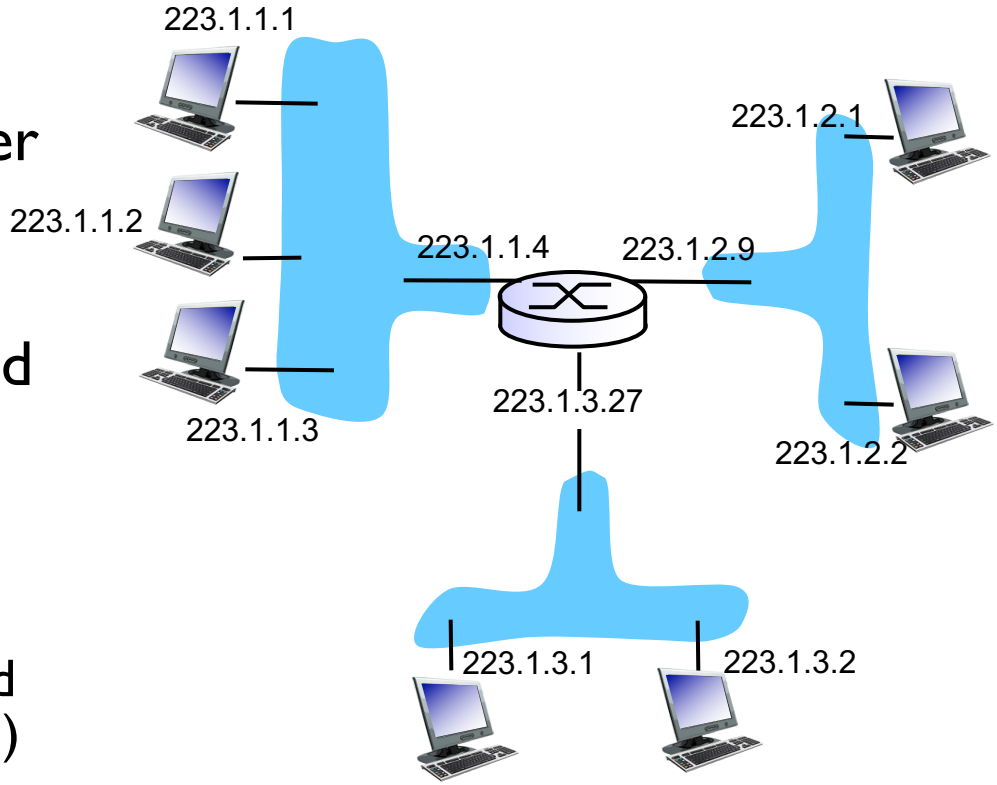
4.2 What's inside a router

4.3 IP: Internet Protocol
- datagram format
- fragmentation
- IPv4 addressing
- network address translation
- IPv6

4.4 Generalized Forward and SDN
- match
- action
- OpenFlow examples of match-plus-action in action

# IP addressing: introduction

- *IP address:* 32-bit identifier for host, router *interface*
- *interface:* connection between host/router and physical link
  - router's typically have multiple interfaces
  - host typically has one or two interfaces (e.g., wired Ethernet, wireless 802.11)
- *IP addresses associated with each interface*



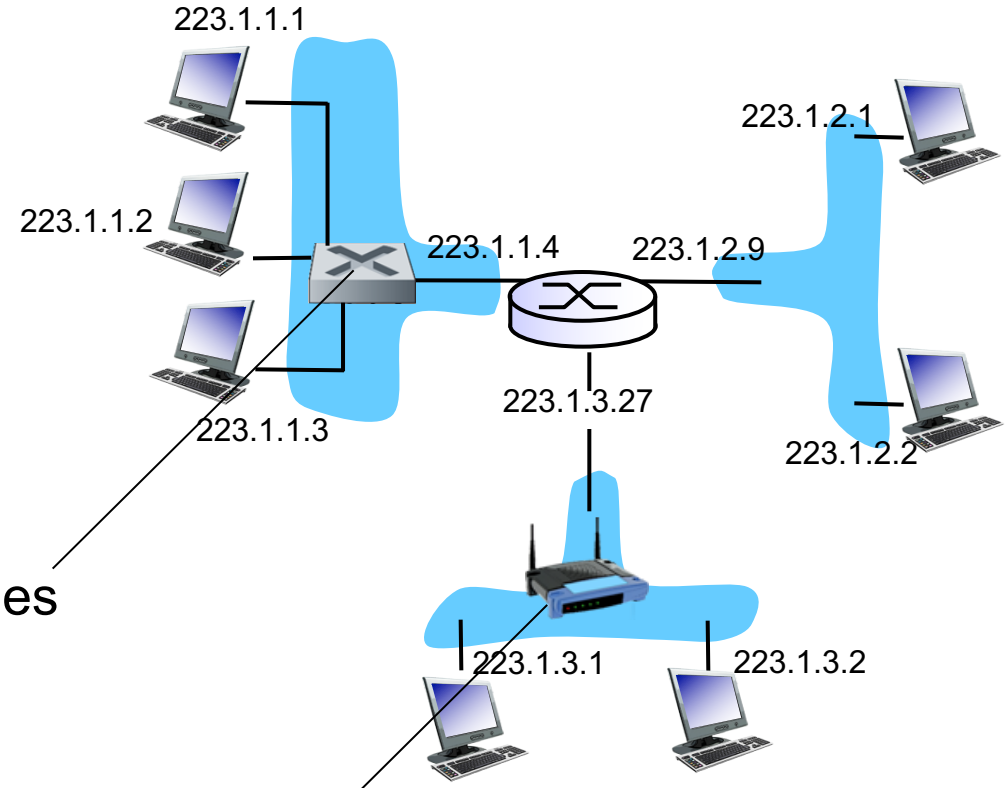

223.1.1.1 = 11011111 00000001 00000001 00000001

223 1 1 1

# IP addressing: introduction

*Q: how are interfaces actually connected?*

*A: we'll learn about that in chapter 5, 6.*



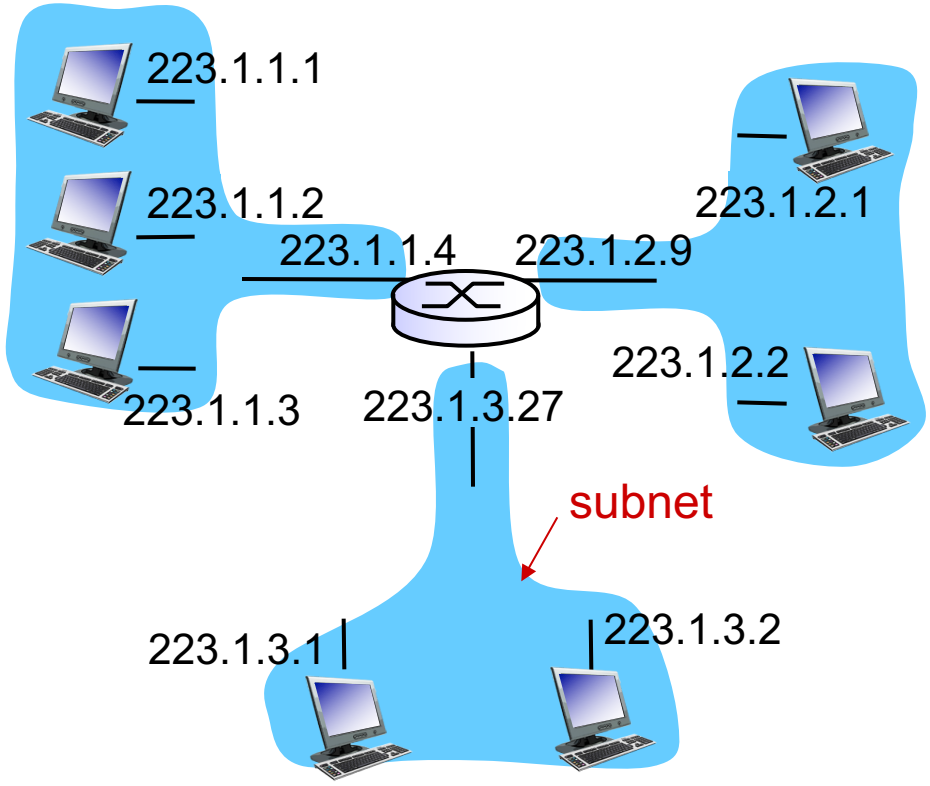

*A:* wired Ethernet interfaces connected by Ethernet switches

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

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

# Subnets

- IP address:
  - subnet part - high order bits
  - host part - low order bits
- *what's a subnet ?*
  - device interfaces with same subnet part of IP address
  - can physically reach each other *without intervening router*

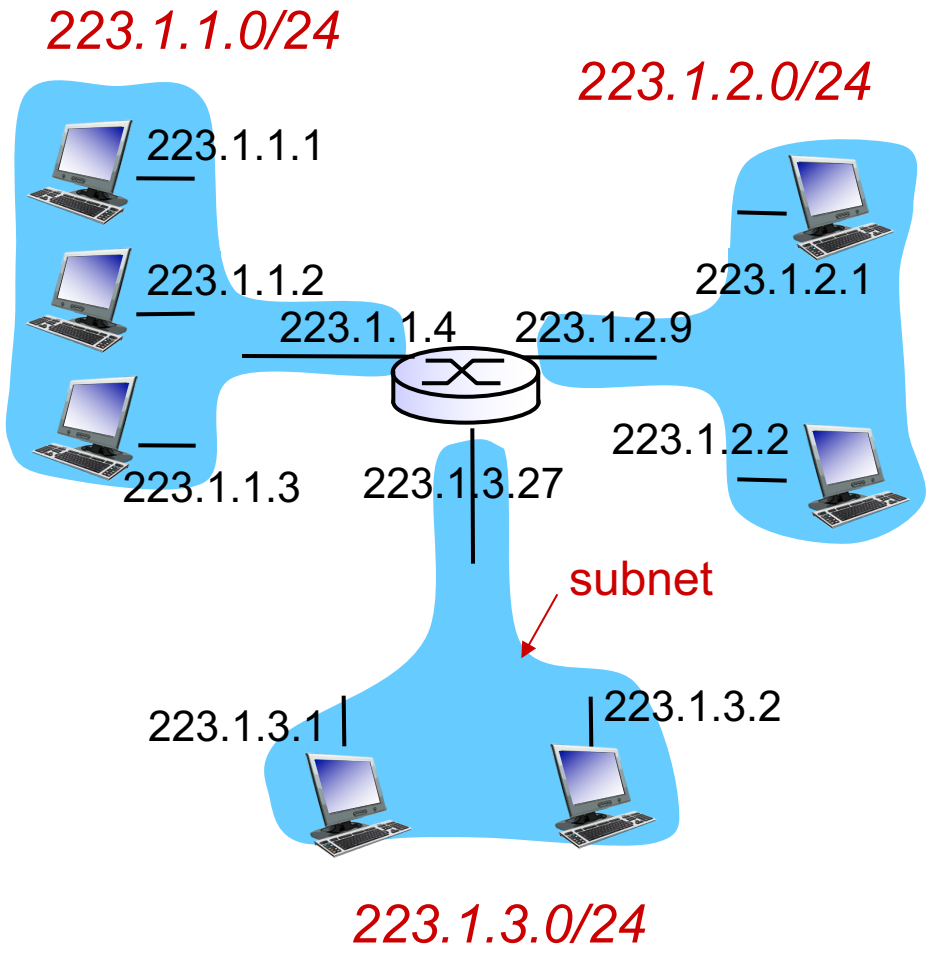



network consisting of 3 subnets

# Subnets

*recipe*

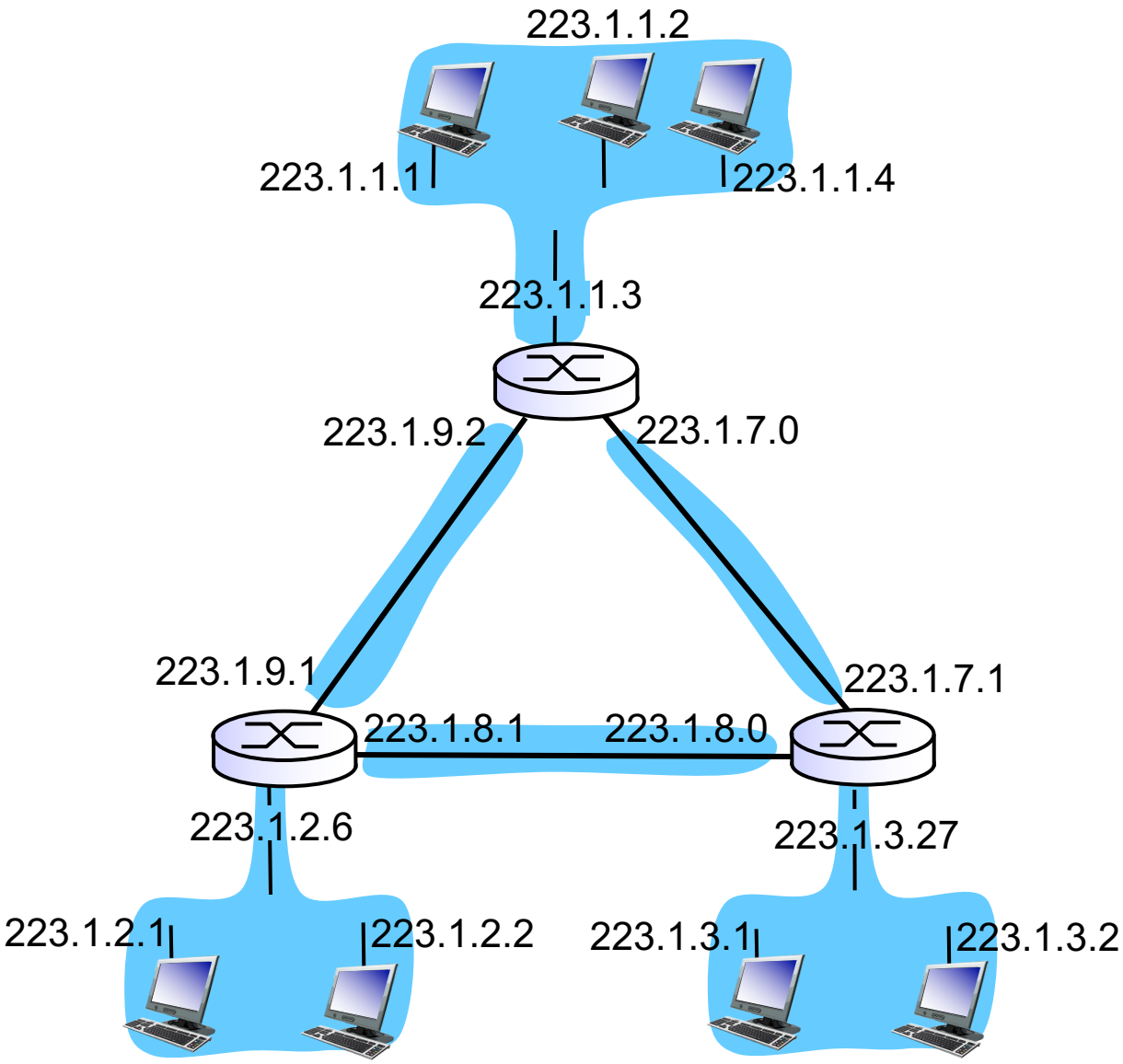
- to determine the subnets, detach each interface from its host or router, creating islands of isolated networks
- each isolated network is called a *subnet*




subnet mask: /24

# Subnets

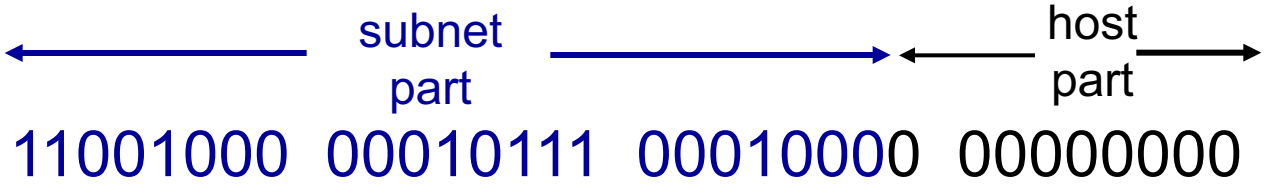
how many?



# IP addressing: CIDR

CIDR: Classless InterDomain Routing

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




200.23.16.0/23

# IP addresses: how to get one?

Q: How does a *host* get IP address?

- hard-coded by system admin in a file
  - Windows: control-panel->network->configuration->tcp/ip->properties
  - UNIX: /etc/rc.config
- DHCP: Dynamic Host Configuration Protocol: dynamically get address from as server
  - "plug-and-play"

# DHCP: Dynamic Host Configuration Protocol

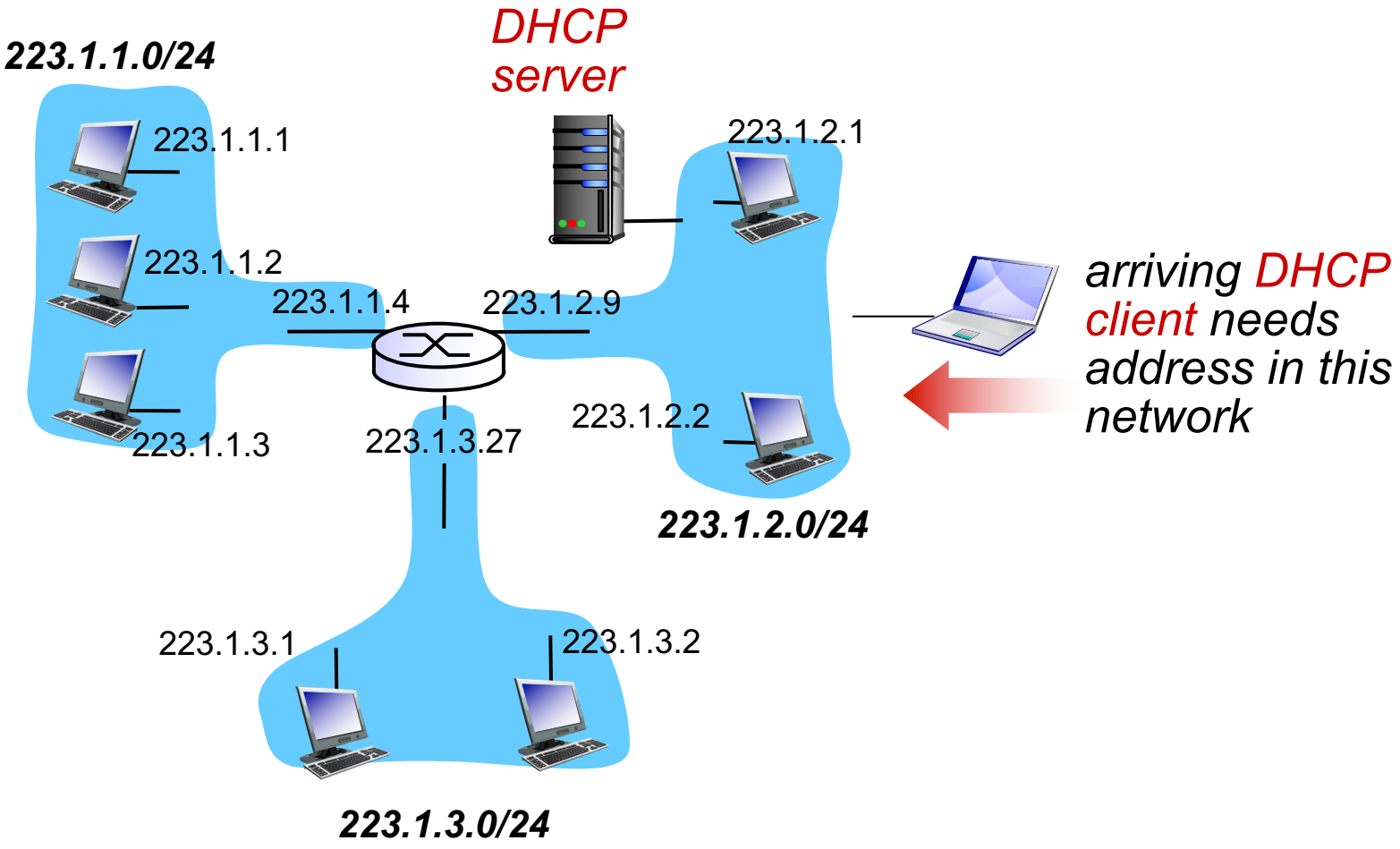
*goal:* allow host to *dynamically* obtain its IP address from network server when it joins network

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

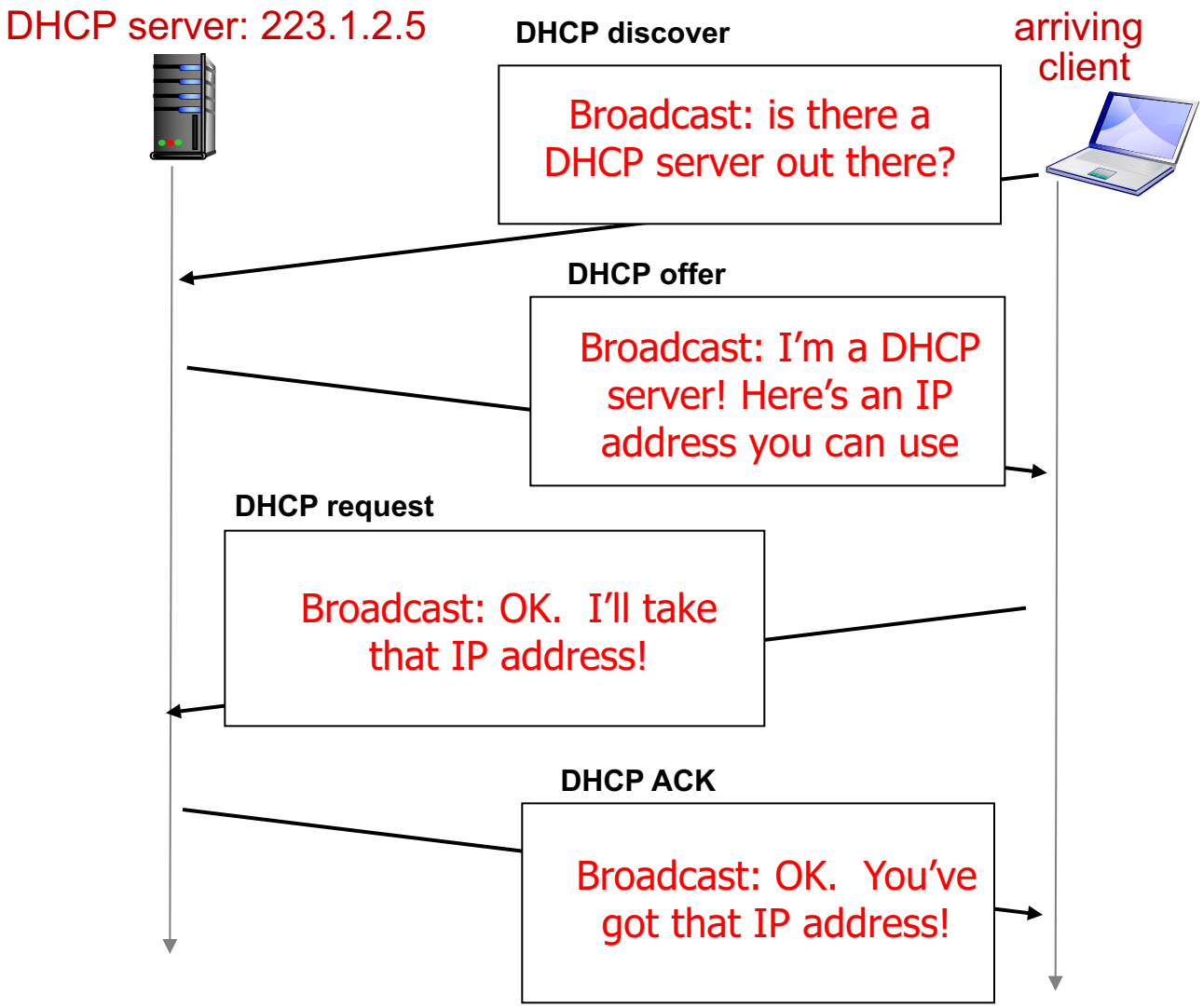
*DHCP overview:*

- 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



# DHCP client-server scenario

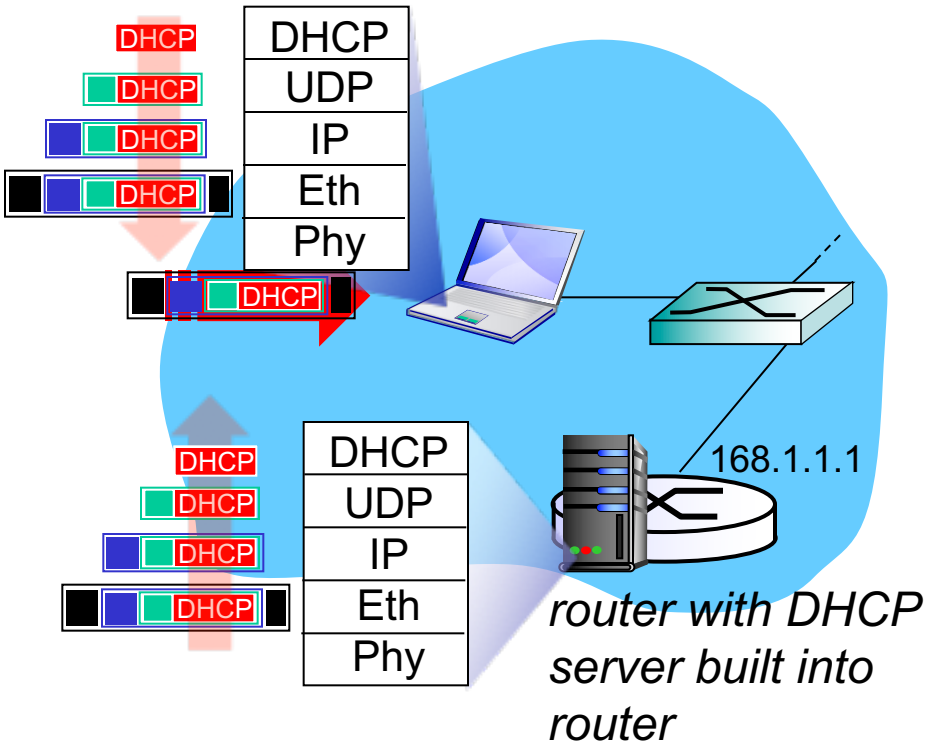


# DHCP: more than IP addresses

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

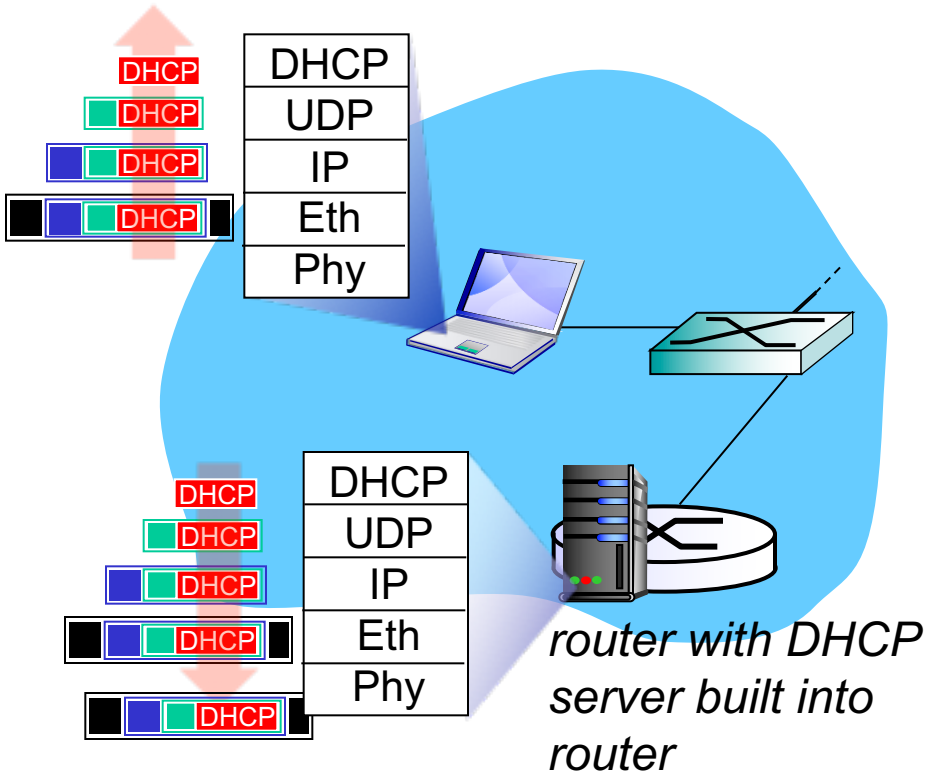
- address of first-hop router for client
- name and IP address of DNS sever
- network mask (indicating network versus host portion of address)

# DHCP: example




- connecting laptop needs its IP address, addr of first-hop router, addr of DNS server: use DHCP
- DHCP request encapsulated in UDP, encapsulated in IP, encapsulated in 802.1 Ethernet
- Ethernet frame broadcast (dest: FFFFFFFFFFFF) on LAN, received at router running DHCP server
- Ethernet demuxed to IP demuxed, UDP demuxed to DHCP

# DHCP: example




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

# DHCP: Wireshark output (home LAN)

request

Message type: **Boot Request (1)**
Hardware type: Ethernet
Hardware address length: 6
Hops: 0
**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
    ……

reply

Message type: **Boot Reply (2)**
Hardware type: Ethernet
Hardware address length: 6
Hops: 0
**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."**

# IP addresses: how to get one?

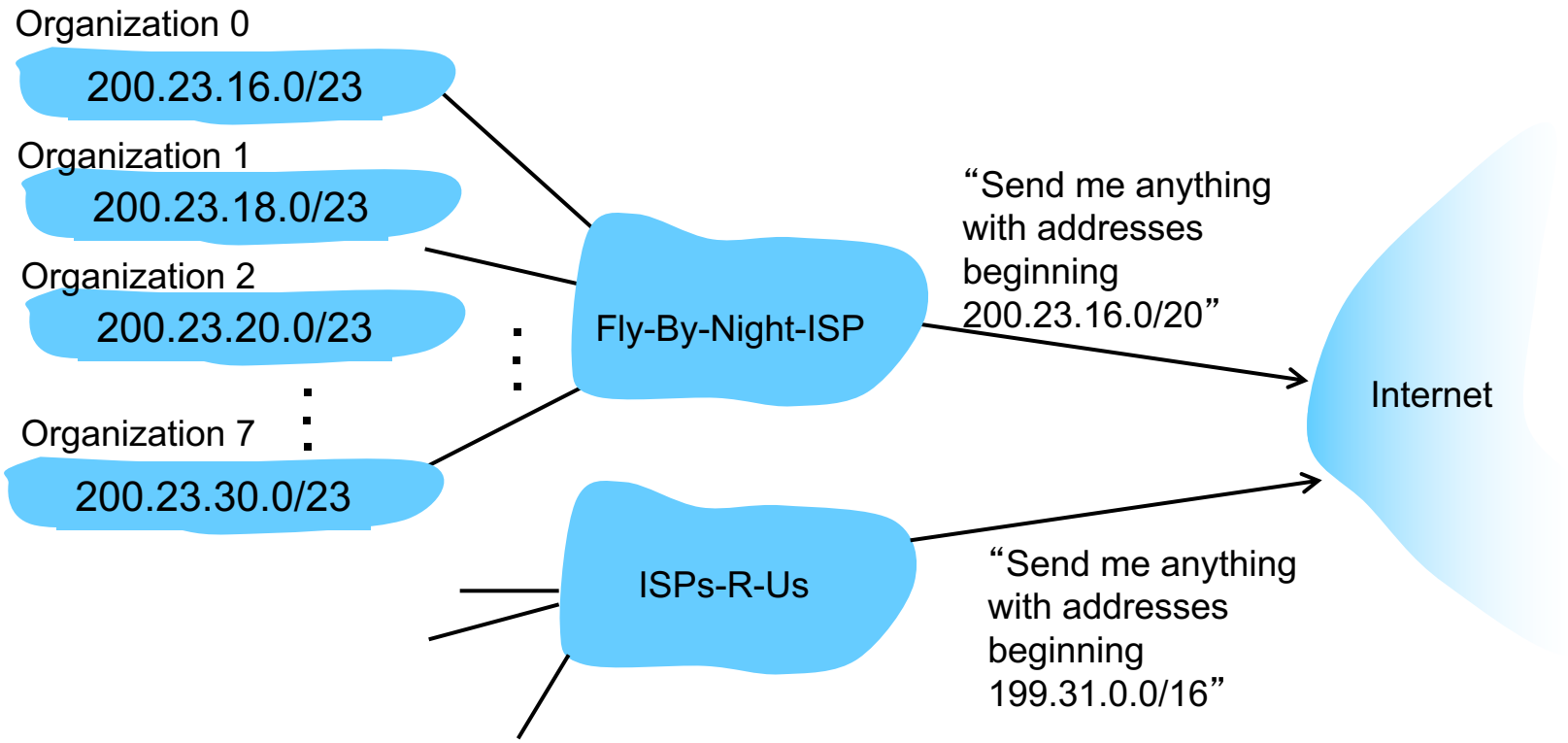
*Q:* how does *network* get subnet part of IP addr?

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

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

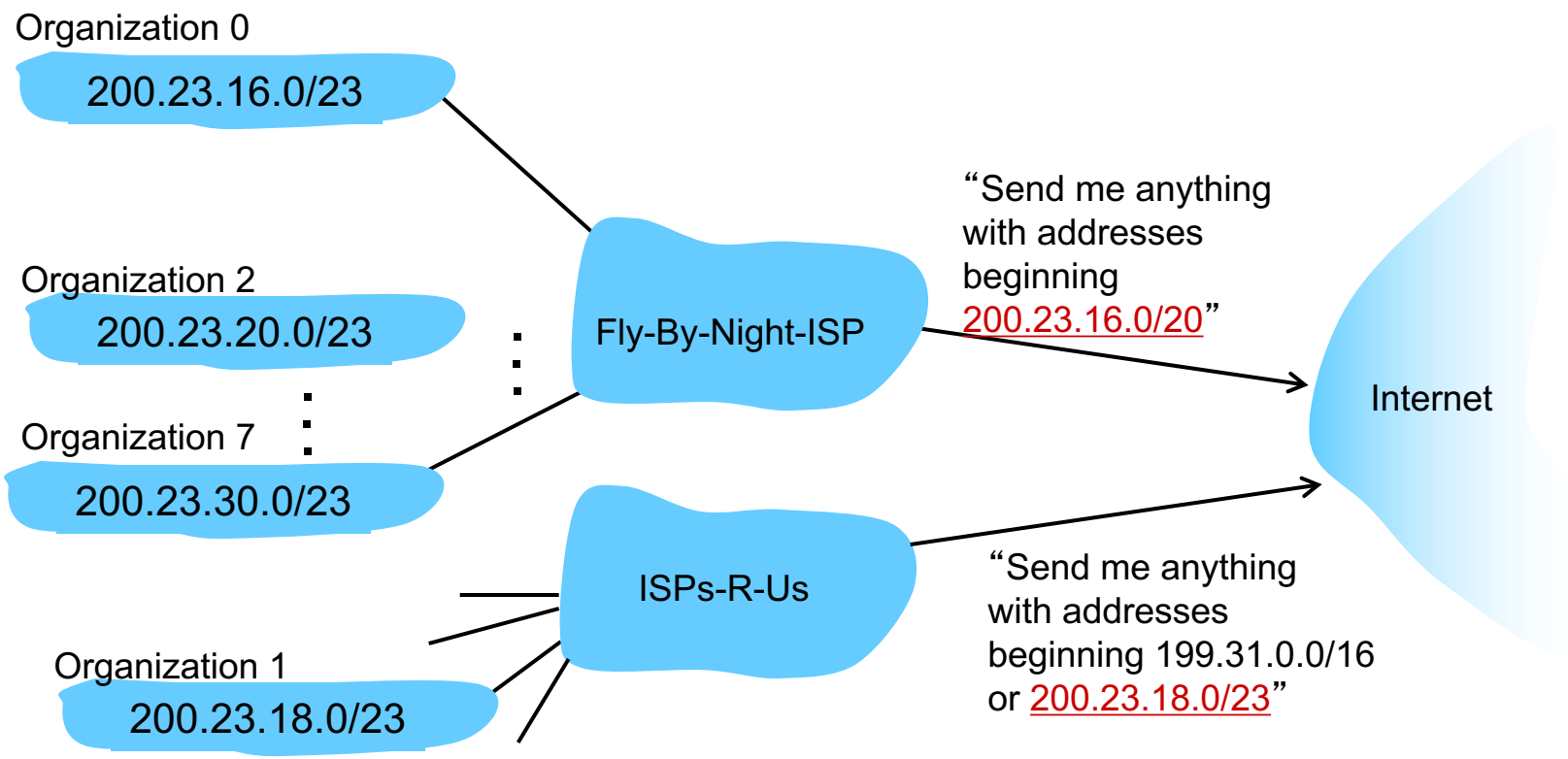
# Hierarchical addressing: route aggregation

hierarchical addressing allows efficient advertisement of routing information:



# Hierarchical addressing: more specific routes

ISPs-R-Us has a more specific route to Organization 1



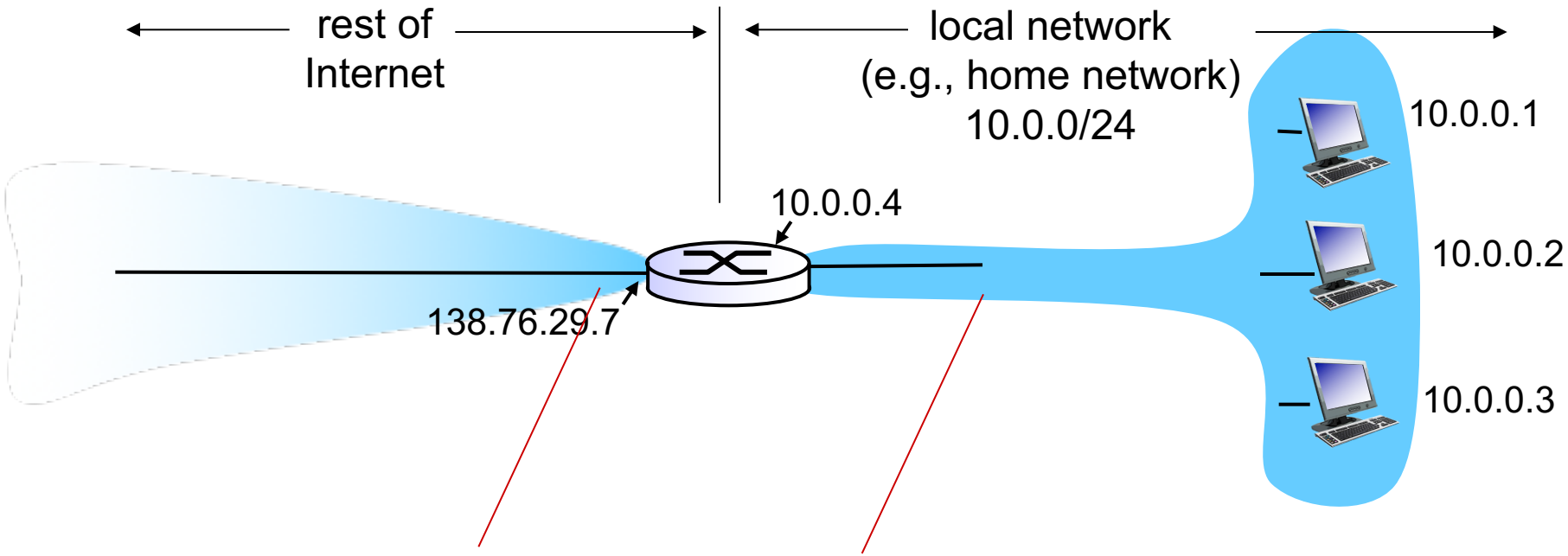
# IP addressing: the last word...

*Q:* how does an ISP get block of addresses?

*A:* ICANN: Internet Corporation for Assigned Names and Numbers http://www.icann.org/

- allocates addresses
- manages DNS
- assigns domain names, resolves disputes

# NAT: network address translation




*all* datagrams *leaving* local network have *same* single source NAT IP address: 138.76.29.7,different source port numbers

datagrams with source or destination in this network have 10.0.0/24 address for source, destination (as usual)

# NAT: network address translation

*motivation:* local network uses just one IP address as far as outside world is concerned:

- range of addresses not needed from ISP: just one IP address for all devices
- can change addresses of devices in local network without notifying outside world
- can change ISP without changing addresses of devices in local network
- devices inside local net not explicitly addressable, visible by outside world (a security plus)

# NAT: network address translation

*implementation:* NAT router must:

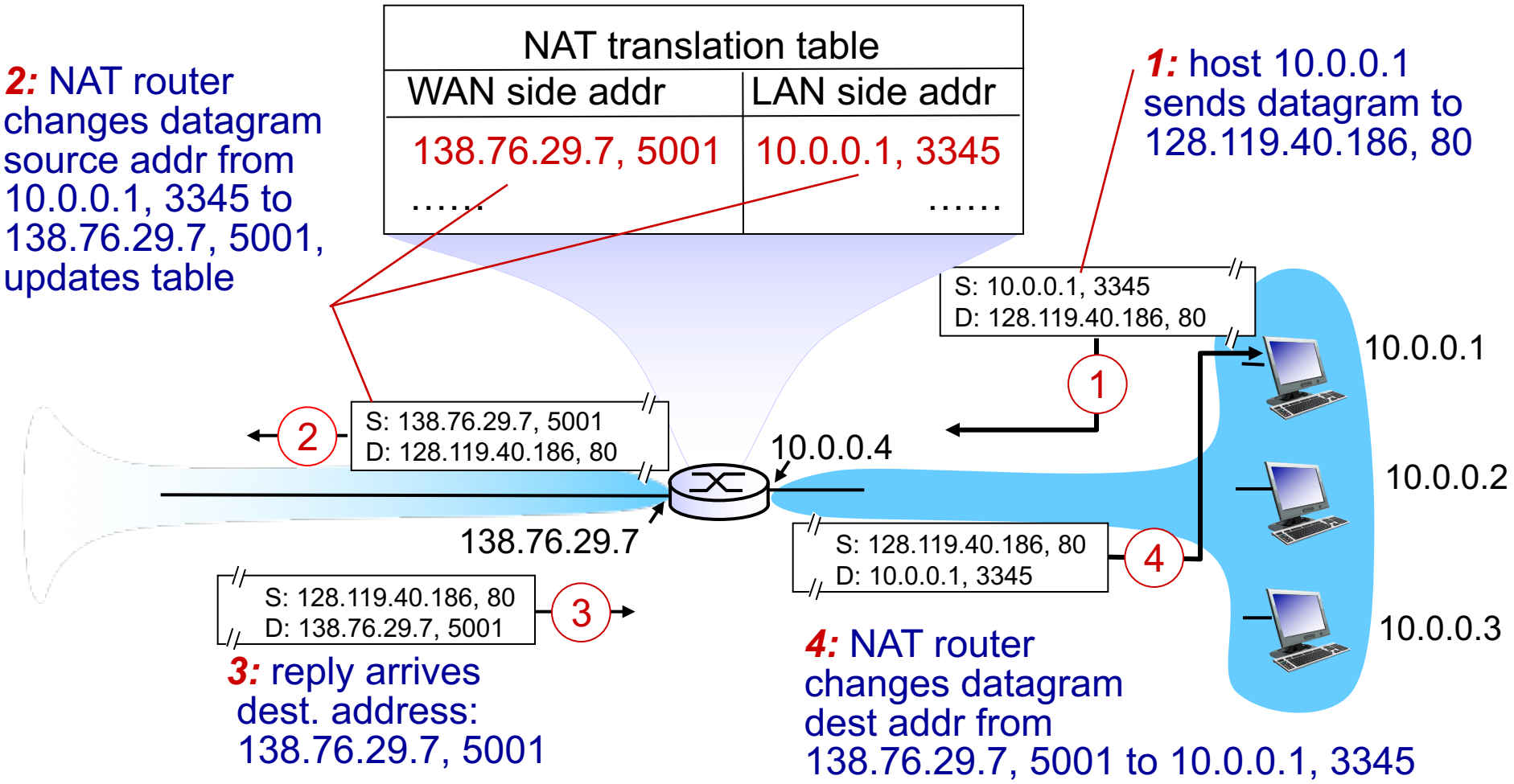
- *outgoing datagrams: replace* (source IP address, port #) of every outgoing datagram to (NAT IP address, new port #)

  . . . remote clients/servers will respond using (NAT IP address, new port #) as destination addr

- *remember (in NAT translation table)* every (source IP address, port #) to (NAT IP address, new port #) translation pair

- *incoming datagrams: replace* (NAT IP address, new port #) in dest fields of every incoming datagram with corresponding (source IP address, port #) stored in NAT table

# NAT: network address translation




* Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu/kurose_ross/interactive/

# NAT: network address translation

- 16-bit port-number field:
  - 60,000 simultaneous connections with a single LAN-side address!
- NAT is controversial:
  - routers should only process up to layer 3
  - address shortage should be solved by IPv6
  - violates end-to-end argument
    - NAT possibility must be taken into account by app designers, e.g., P2P applications
  - NAT traversal: what if client wants to connect to server behind NAT?

# Chapter 4: outline

4.1 Overview of Network layer
- data plane
- control plane

4.2 What's inside a router

4.3 IP: Internet Protocol
- datagram format
- fragmentation
- IPv4 addressing
- network address translation
- IPv6

4.4 Generalized Forward and SDN
- match
- action
- OpenFlow examples of match-plus-action in action

# IPv6: motivation

- *initial motivation:* 32-bit address space soon to be completely allocated.
- additional motivation:
  - header format helps speed processing/forwarding
  - header changes to facilitate QoS

*IPv6 datagram format:*

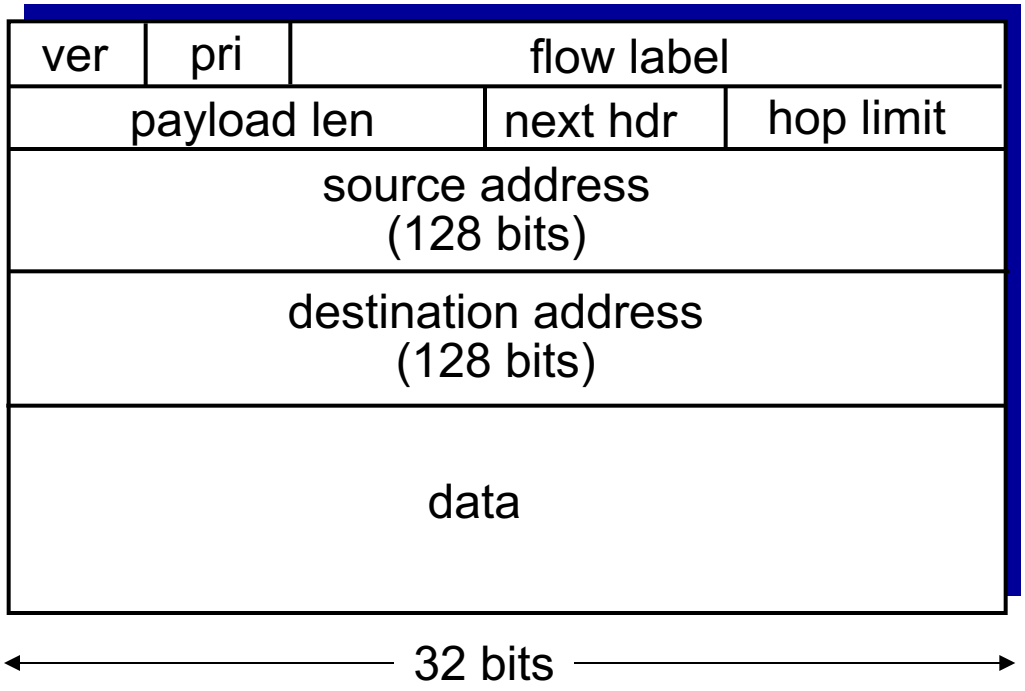
- fixed-length 40 byte header
- no fragmentation allowed

# IPv6 datagram format

*priority:* identify priority among datagrams in flow

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

*next header:* identify upper layer protocol for data

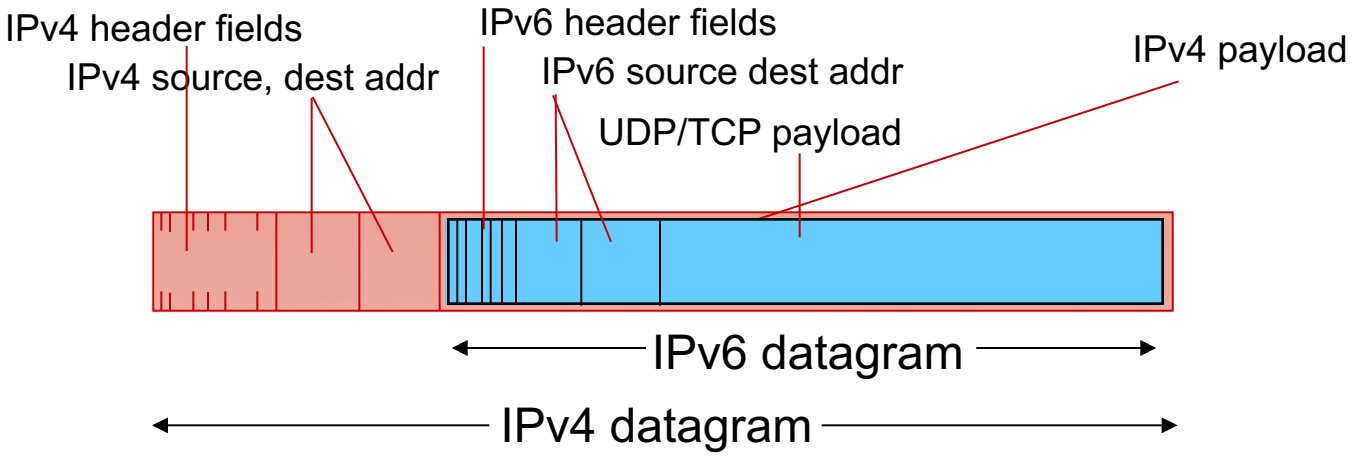


# Other changes from IPv4

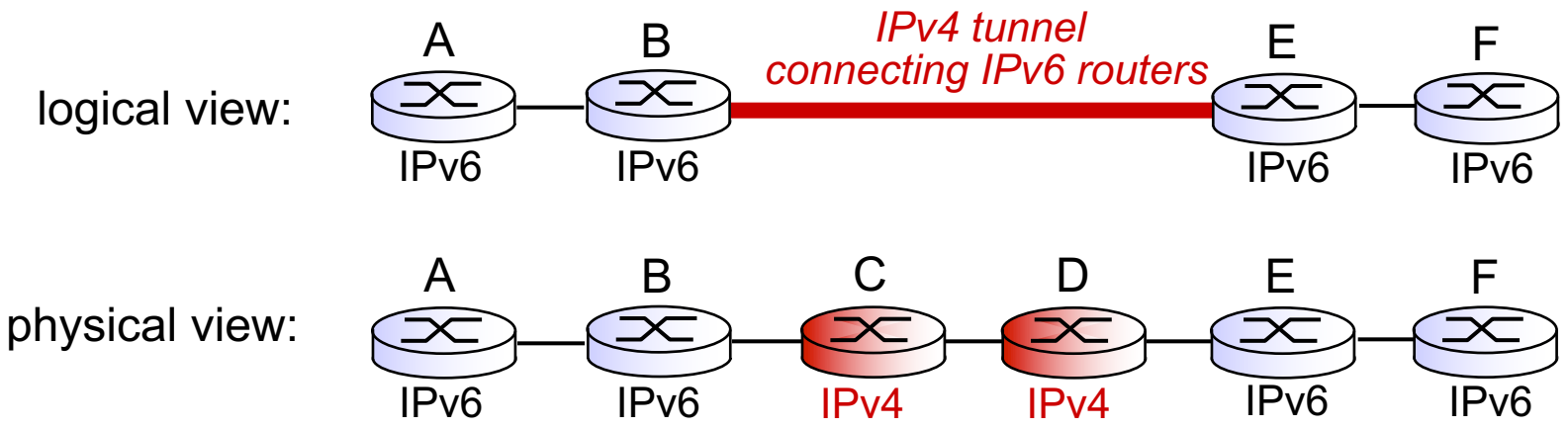
- *checksum:* removed entirely to reduce processing time at each hop
- *options:* allowed, but outside of header, indicated by "Next Header" field
- *ICMPv6:* new version of ICMP
  - additional message types, e.g. "Packet Too Big"
  - multicast group management functions

# Transition from IPv4 to IPv6

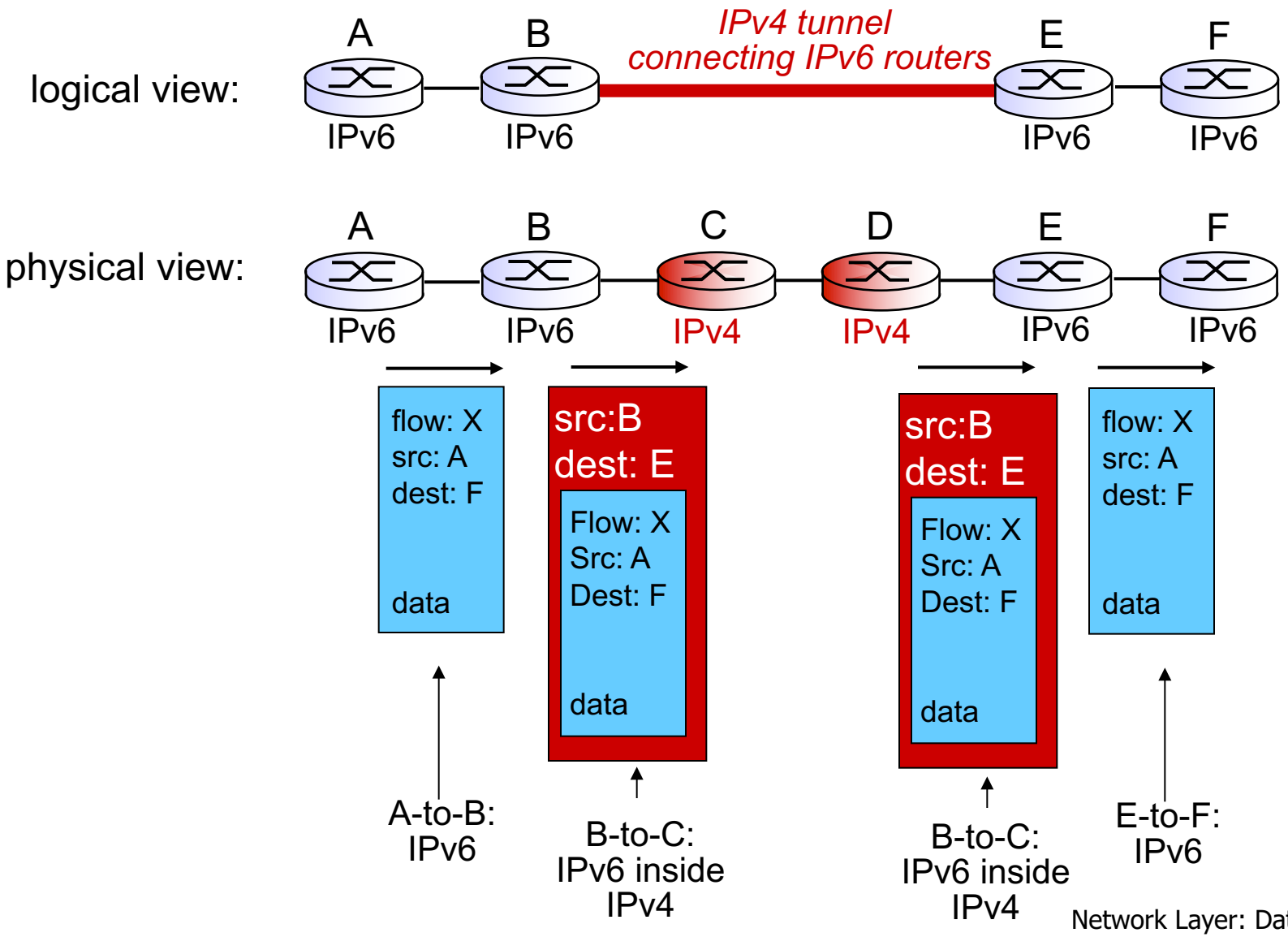
- not all routers can be upgraded simultaneously
  - no "flag days"
  - how will network operate with mixed IPv4 and IPv6 routers?
- *tunneling:* IPv6 datagram carried as *payload* in IPv4 datagram among IPv4 routers



# Tunneling


logical view:
A
B
IPv4 tunnel
connecting IPv6 routers
E
F
IPv6
IPv6
IPv6
IPv6
physical view:
A
B
C
D
E
F
IPv6
IPv6
IPv4
IPv4
IPv6
IPv6

# Tunneling


logical view:
A
B
IPv4 tunnel
connecting IPv6 routers
E
F
IPv6
IPv6
IPv6
IPv6
physical view:
A
B
C
D
E
F
IPv6
IPv6
IPv4
IPv4
IPv6
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
flow: X
src: A
dest: F
data
A-to-B:
IPv6
B-to-C:
IPv6 inside
IPv4
B-to-C:
IPv6 inside
IPv4
E-to-F:
IPv6

# IPv6: adoption

- Google: 8% of clients access services via IPv6
- NIST: 1/3 of all US government domains are IPv6 capable

- *Long (long!) time for deployment, use*
  - 20 years and counting!
  - think of application-level changes in last 20 years: WWW, Facebook, streaming media, Skype, …
  - *Why?*

# Chapter 4: outline

4.1 Overview of Network layer
- data plane
- control plane

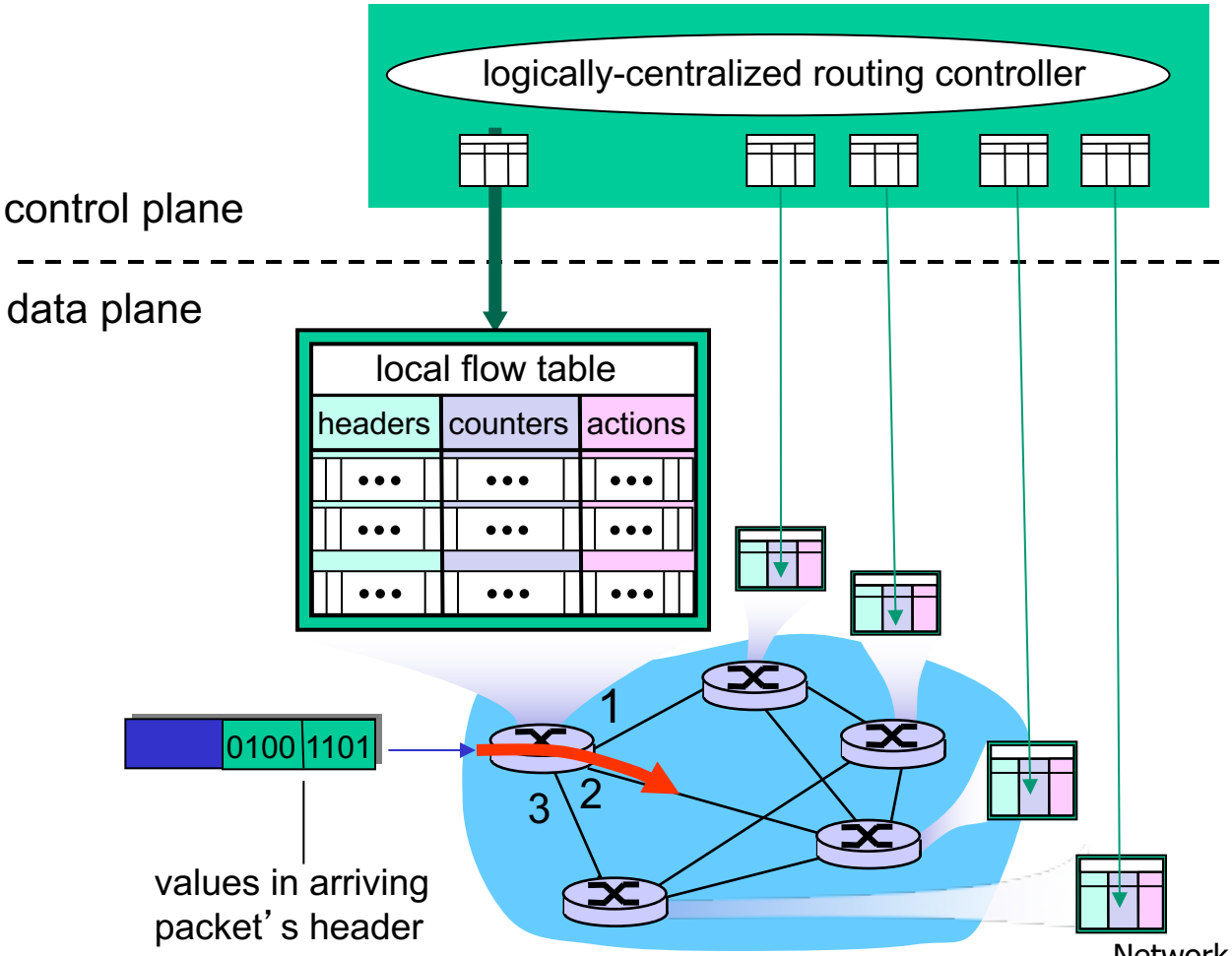
4.2 What's inside a router

4.3 IP: Internet Protocol
- datagram format
- fragmentation
- IPv4 addressing
- network address translation
- IPv6

4.4 Generalized Forward and SDN
- match
- action
- OpenFlow examples of match-plus-action in action

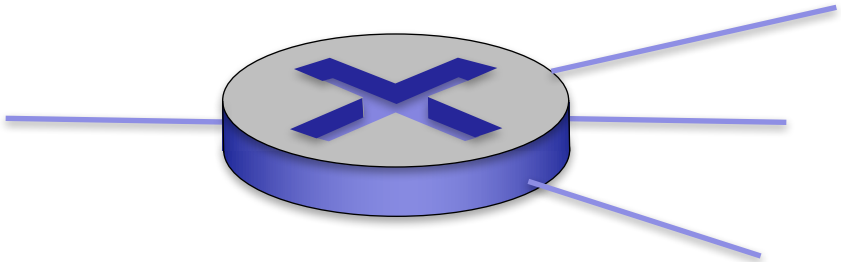
# Generalized Forwarding and SDN

Each router contains a *flow table* that is computed and distributed by a *logically centralized* routing controller



# OpenFlow data plane abstraction

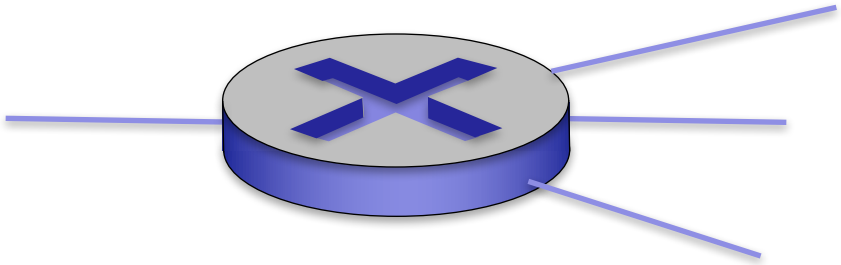
- *flow*: defined by header fields
- generalized forwarding: simple packet-handling rules
  - *Pattern:* match values in packet header fields
  - *Actions: for matched packet:* drop, forward, modify, matched packet or send matched packet to controller
  - *Priority*: disambiguate overlapping patterns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



*Flow table in a router (computed and distributed by controller) define router's match+action rules*

# OpenFlow data plane abstraction

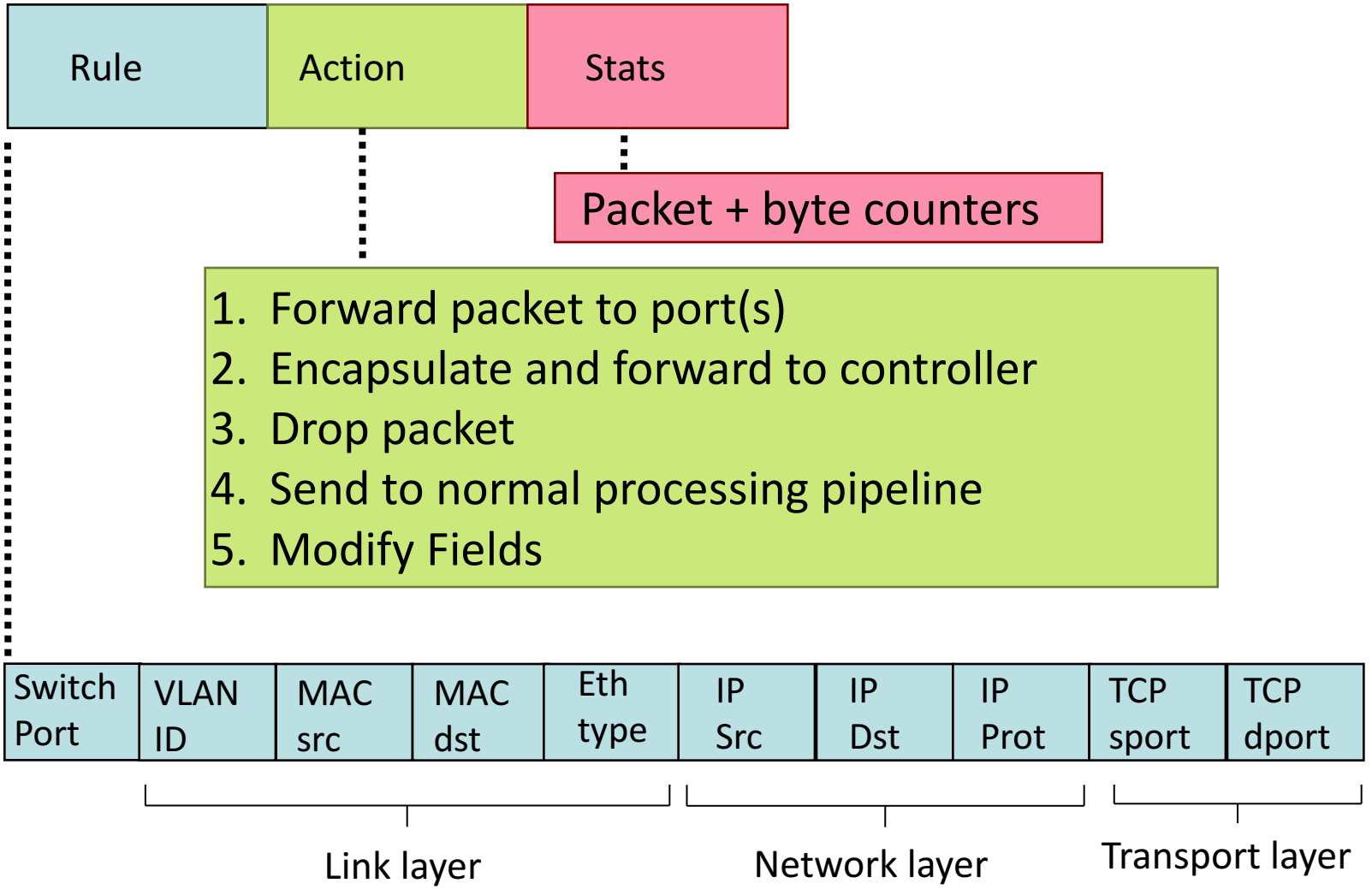
- *flow*: defined by header fields
- generalized forwarding: simple packet-handling rules
  - *Pattern:* match values in packet header fields
  - *Actions: for matched packet:* drop, forward, modify, matched packet or send matched packet to controller
  - *Priority*: disambiguate overlapping patterns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



* : wildcard

1. src=1.2.*.*, dest=3.4.5.* → drop
2. src = *.*.*.*, dest=3.4.*.* → forward(2)
3. src=10.1.2.3, dest=*.*.*.* → send to controller

# OpenFlow: Flow Table Entries



# Examples

## Destination-based forwarding:

| Switch Port | MAC src | MAC dst | Eth type | VLAN ID | IP Src | IP Dst | IP Prot | TCP sport | TCP dport | 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 | IP Src | IP Dst | IP Prot | TCP sport | TCP dport | Forward |
|---|---|---|---|---|---|---|---|---|---|---|
| * | * | * | * | * | * | * | * | * | 22 | drop |

*do not forward (block) all datagrams destined to TCP port 22*

| Switch Port | MAC src | MAC dst | Eth type | VLAN ID | IP Src | IP Dst | IP Prot | TCP sport | TCP dport | Forward |
|---|---|---|---|---|---|---|---|---|---|---|
| * | * | * | * | * | 128.119.1.1 | * | * | * | * | drop |

*do not forward (block) all datagrams sent by host 128.119.1.1*

# Examples

## Destination-based layer 2 (switch) forwarding:

| Switch Port | MAC src | MAC dst | Eth type | VLAN ID | IP Src | IP Dst | IP Prot | TCP sport | TCP dport | Action |
|---|---|---|---|---|---|---|---|---|---|---|
| * | 22:A7:23: 11:E1:02 | * | * | * | * | * | * | * | * | port3 |

*layer 2 frames from MAC address 22:A7:23:11:E1:02 should be forwarded to output port 6*

# OpenFlow abstraction

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

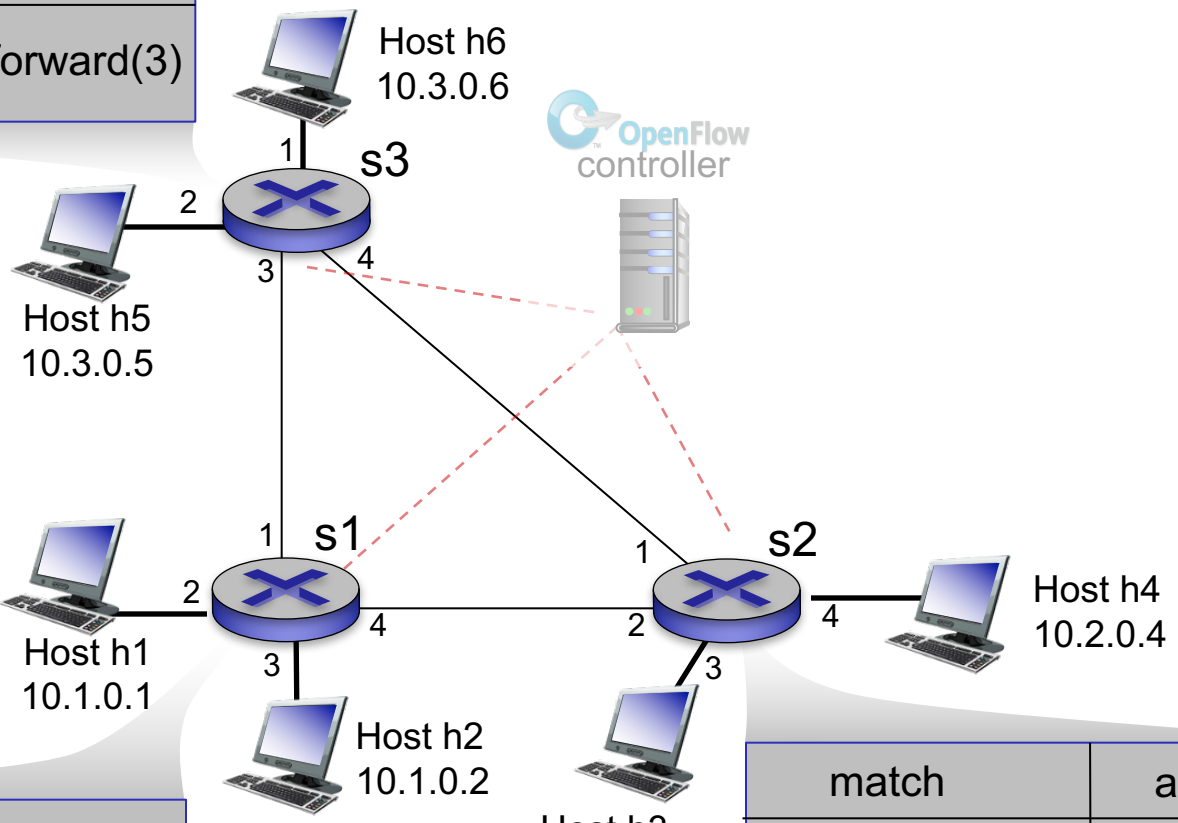
- Router
  - *match:* longest destination IP prefix
  - *action:* forward out a link
- Switch
  - *match:* destination MAC address
  - *action:* forward or flood

- 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

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

| 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) |

# Chapter 4: *done!*

4.1 Overview of Network layer: data plane and control plane

4.2 What's inside a router

4.3 IP: Internet Protocol

- datagram format
- fragmentation
- IPv4 addressing
- NAT
- IPv6

4.4 Generalized Forward and SDN

- match plus action
- OpenFlow example

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

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