

Chapter 4

Network Layer:

The Data Plane

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

© All material copyright 1996-2016
J.F Kurose and K.W. Ross, All Rights Reserved



Computer Networking: A Top Down Approach

7th edition

Jim Kurose, Keith Ross

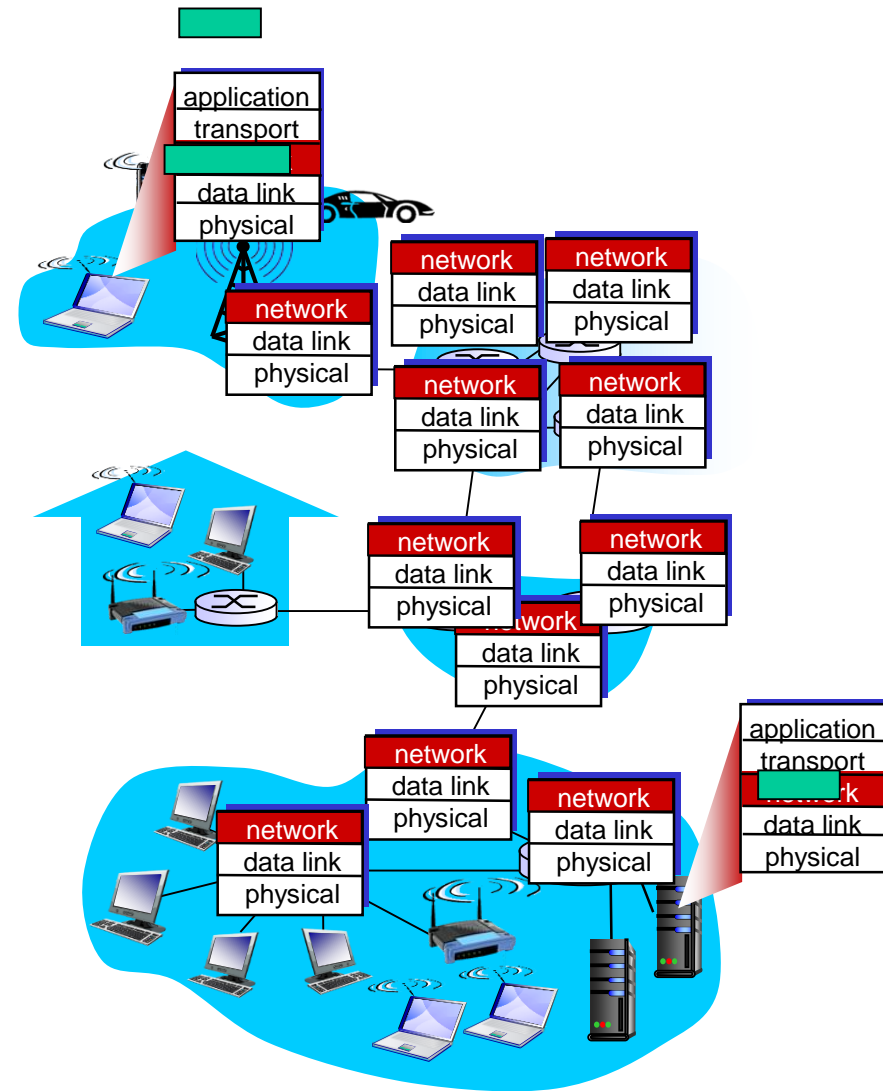
Pearson/Addison Wesley

April 2016

Network Layer

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

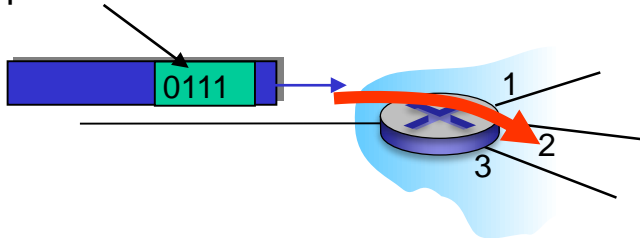
- *forwarding*: process of getting through single interchange
- *routing*: process of planning trip from source to destination

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

values in arriving packet header



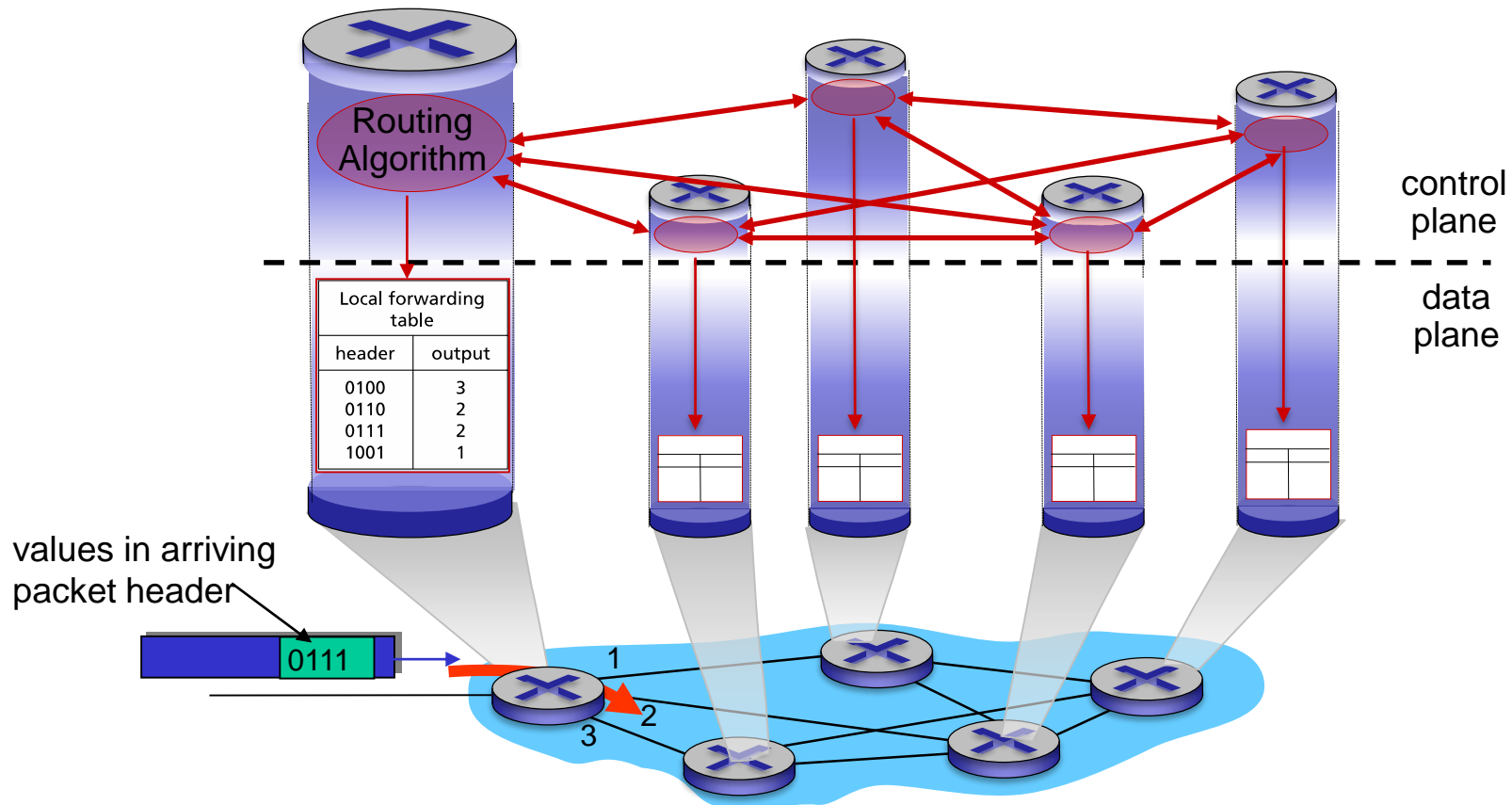
Control plane

- 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

Forwarding vs Routing

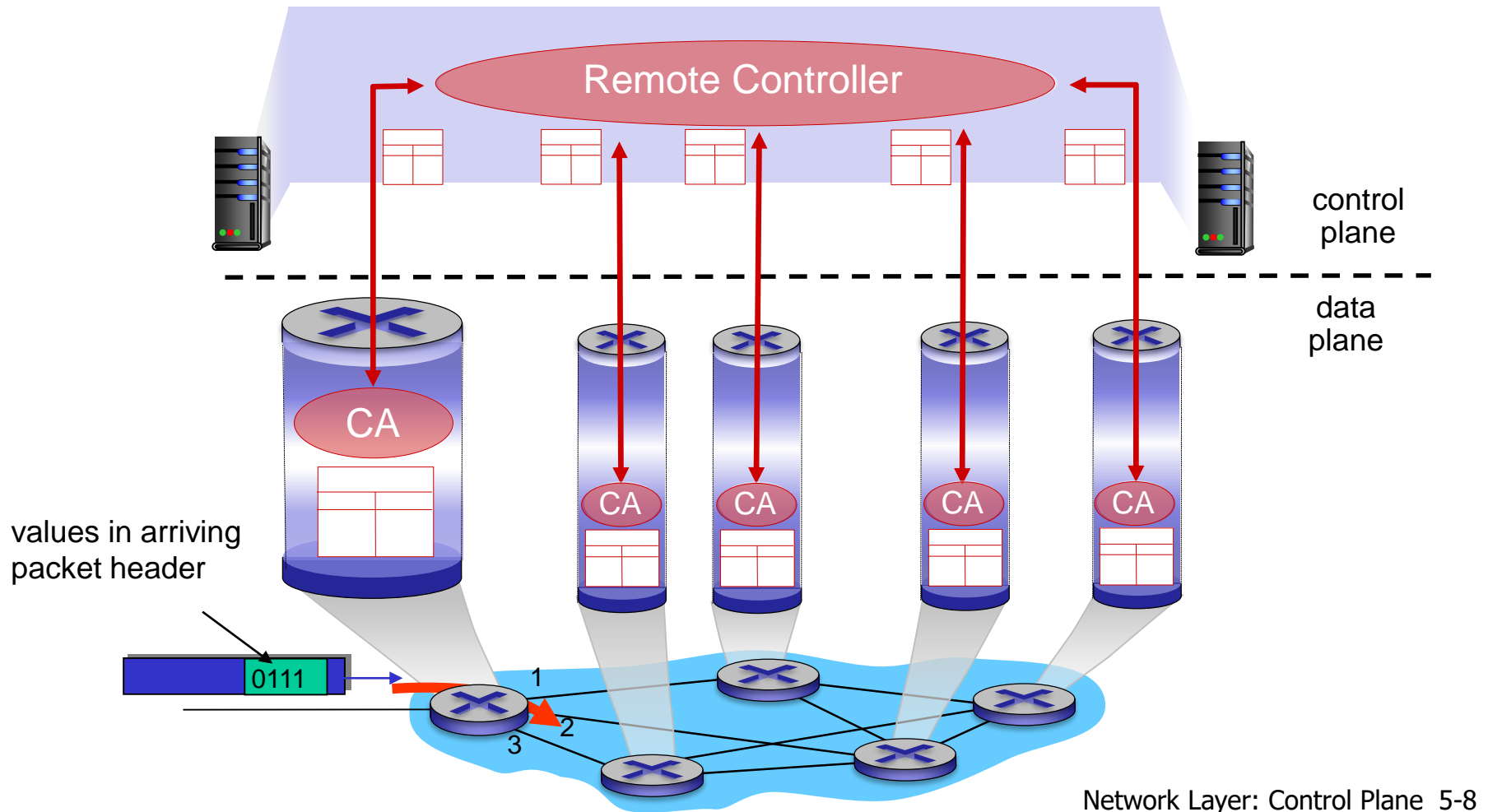
Per-router control plane

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:

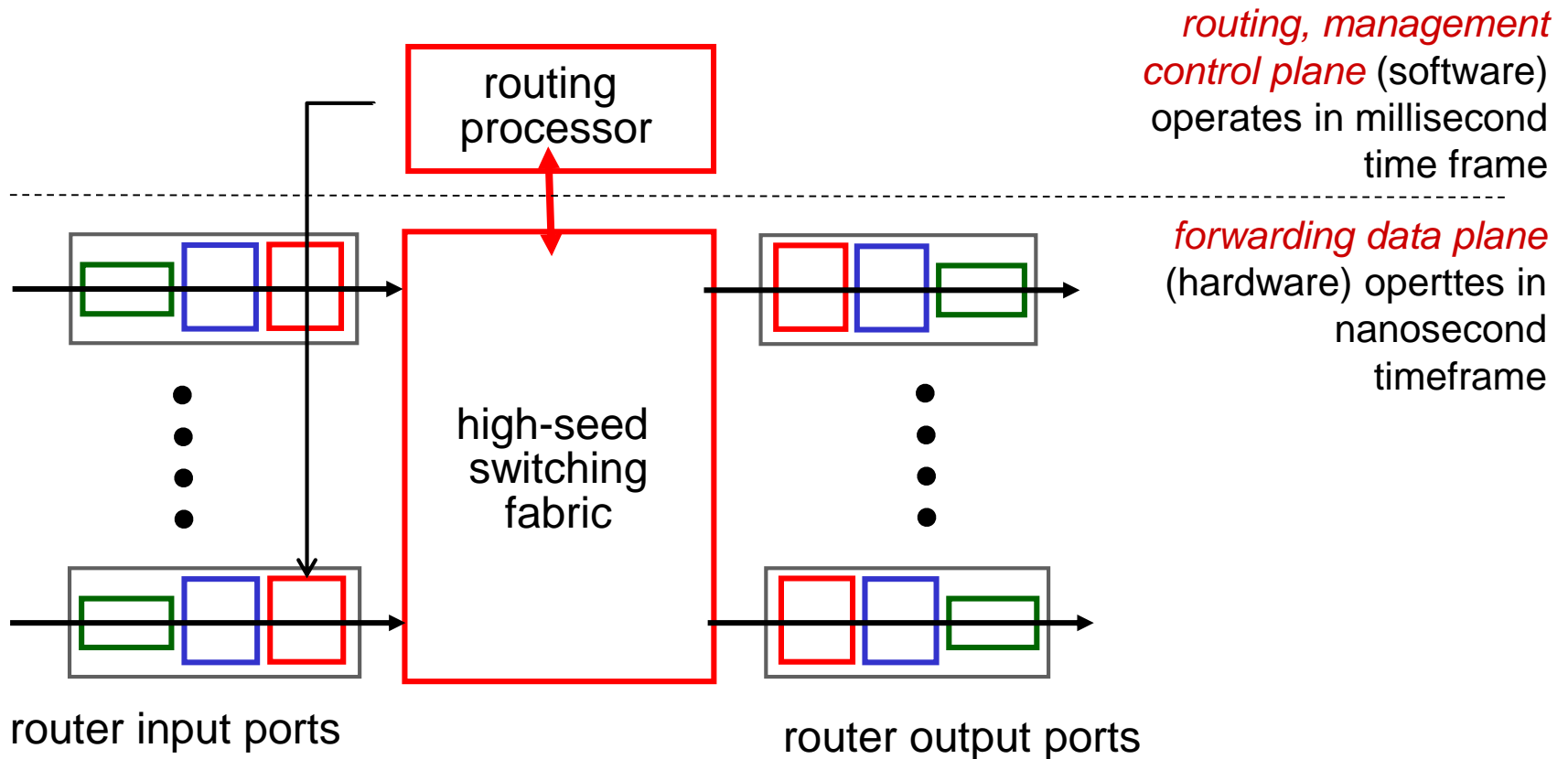
- guaranteed **delivery**
- guaranteed delivery with less than 40 msec **delay**

example services for a flow of datagrams:

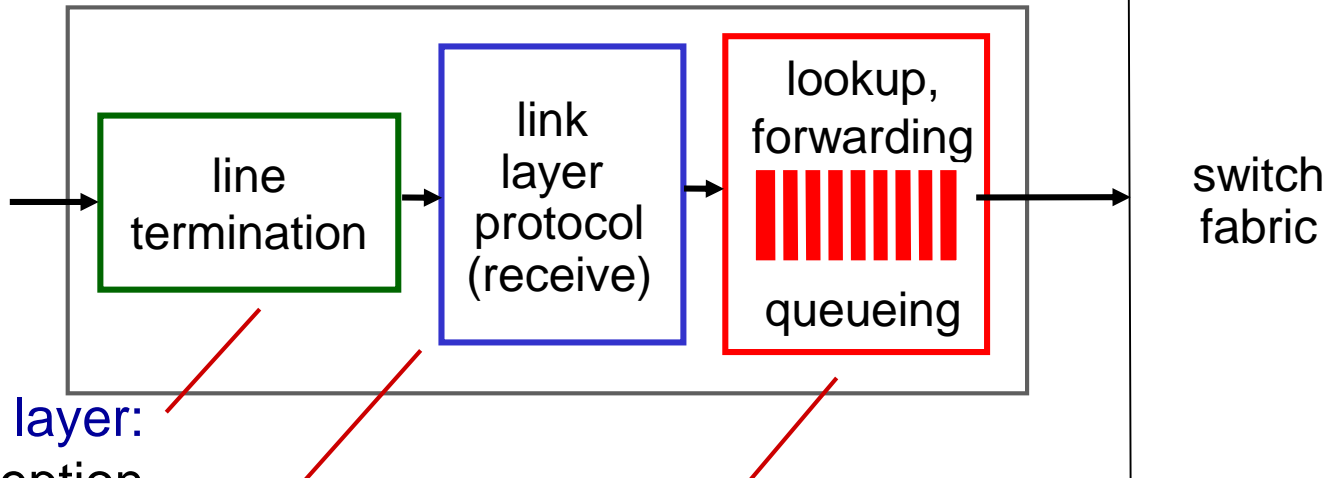
- in-**order** datagram delivery
- guaranteed minimum **bandwidth** to flow
- restrictions on changes in inter-packet spacing

Router architecture overview

- high-level view of generic router architecture:



Input port functions



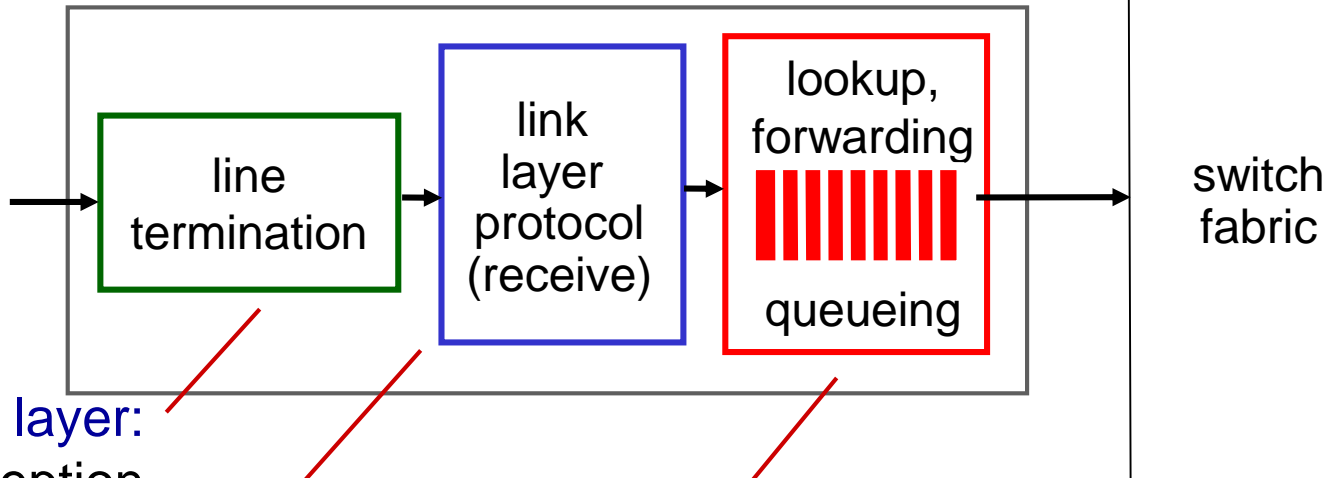
physical layer:
bit-level reception

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 through 11001000 00010111 00010111 11111111	0
11001000 00010111 00011000 00000000 through 11001000 00010111 00011000 11111111	1
11001000 00010111 00011001 00000000 through 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

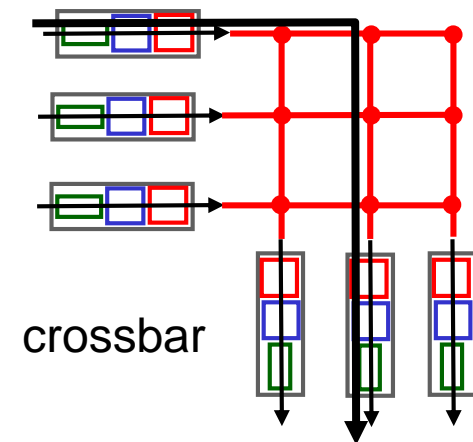
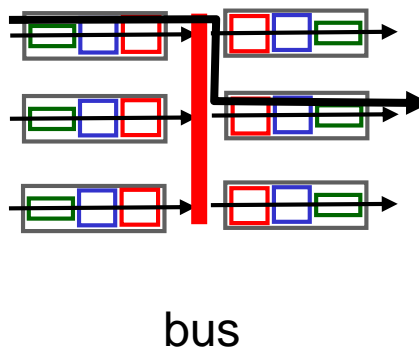
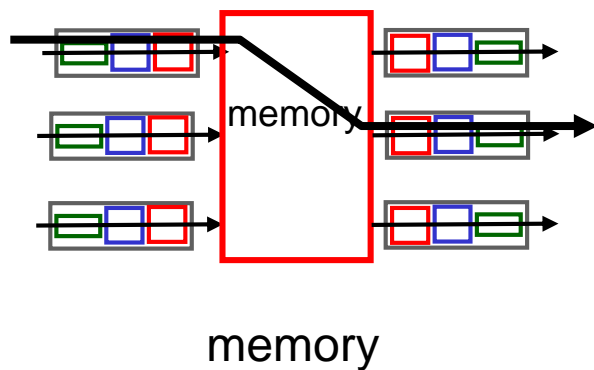
which interface?

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

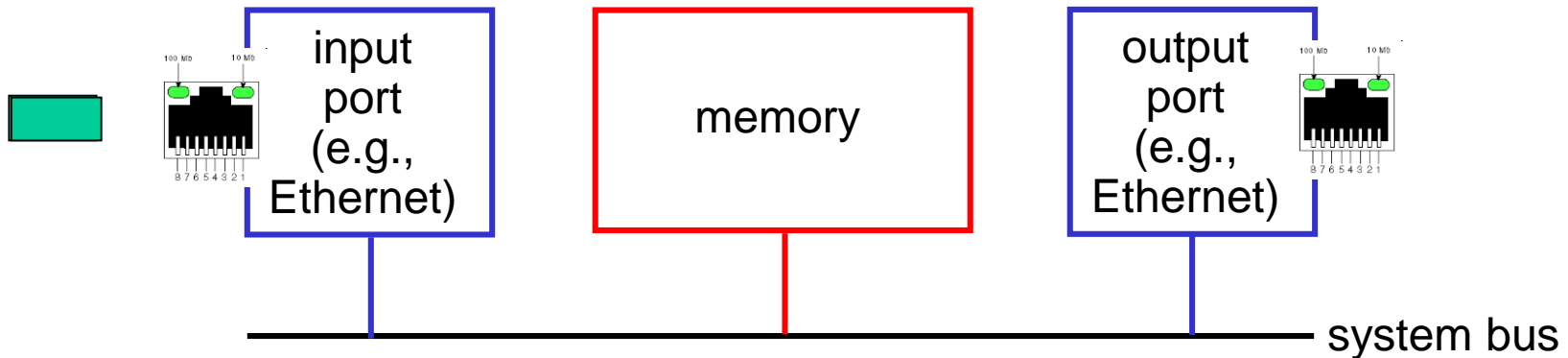
- transfer packet from input buffer to appropriate output buffer
- switching rate: rate at which packets can be transfer from inputs to outputs
 - often measured as multiple of input/output line rate
 - N inputs: switching rate N times line rate desirable
- three types of switching fabrics



Switching via memory

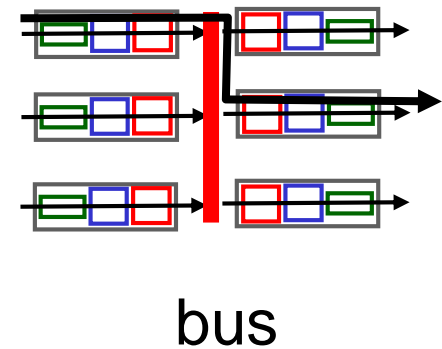
first generation routers:

- 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)



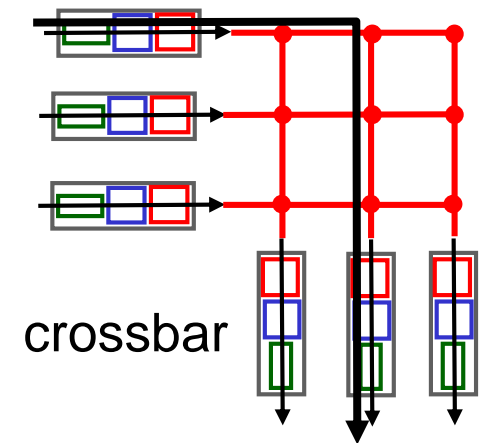
Switching via a bus

- 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



Switching via interconnection network

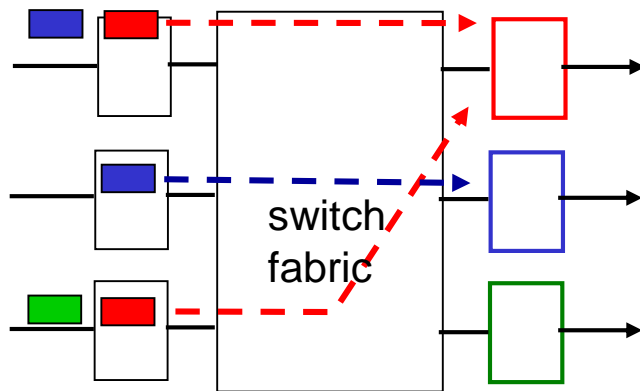
- overcome bus bandwidth limitations
- banyan networks, crossbar, other interconnection nets initially developed to connect processors in multiprocessor
- advanced design: fragmenting datagram into fixed length cells, switch cells through the fabric.
- Cisco I2000: switches 60 Gbps through the interconnection network



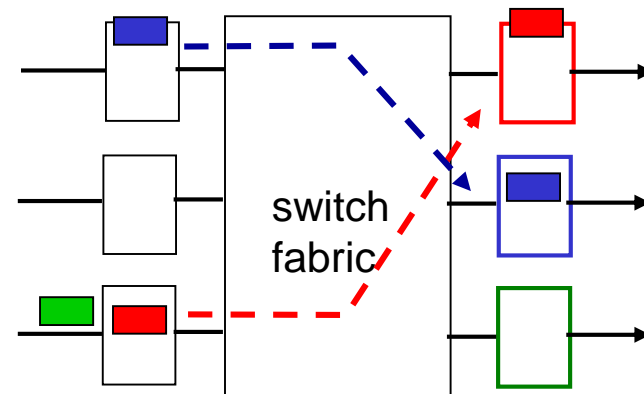
Queueing and Buffering

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



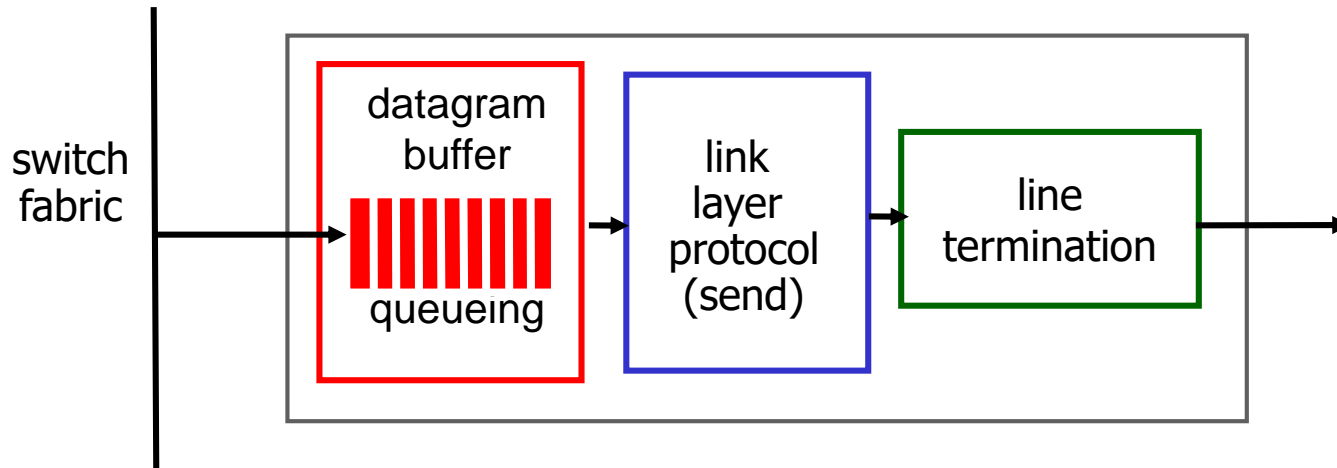
output port contention:
only one red datagram can be
transferred.
lower red packet is blocked



one packet time later:
green packet
experiences HOL
blocking

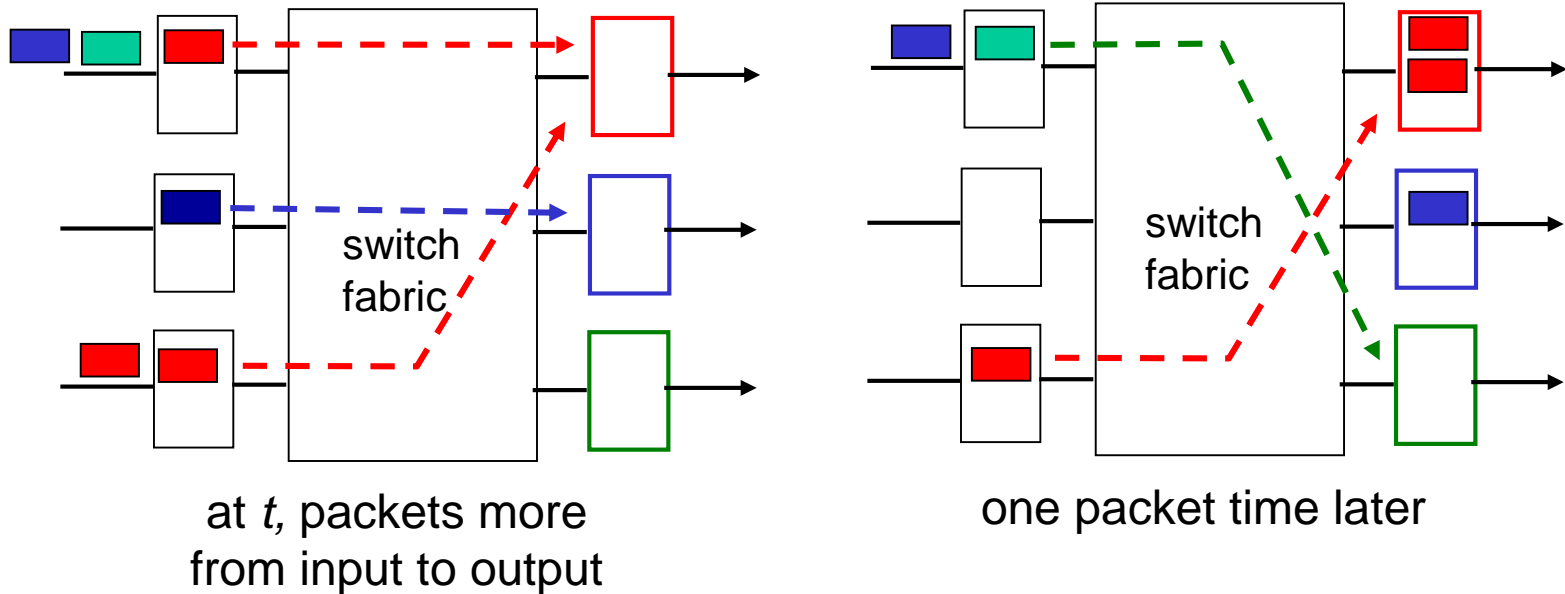
Output ports

This slide is HUGELY important!



- **buffering** required from faster fabric 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



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

How much buffering?

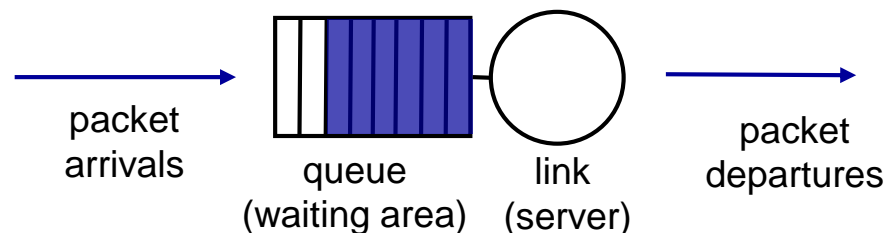
- RFC 3439 rule of thumb: average buffering equal to “typical” RTT (say 250 msec) times link speed (C)
 - e.g., 10 Gbps x 0.250 sec = 2.5 Gb buffer
- recent recommendation: with N flows, buffering equal to

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

Packet Scheduling

Scheduling mechanisms

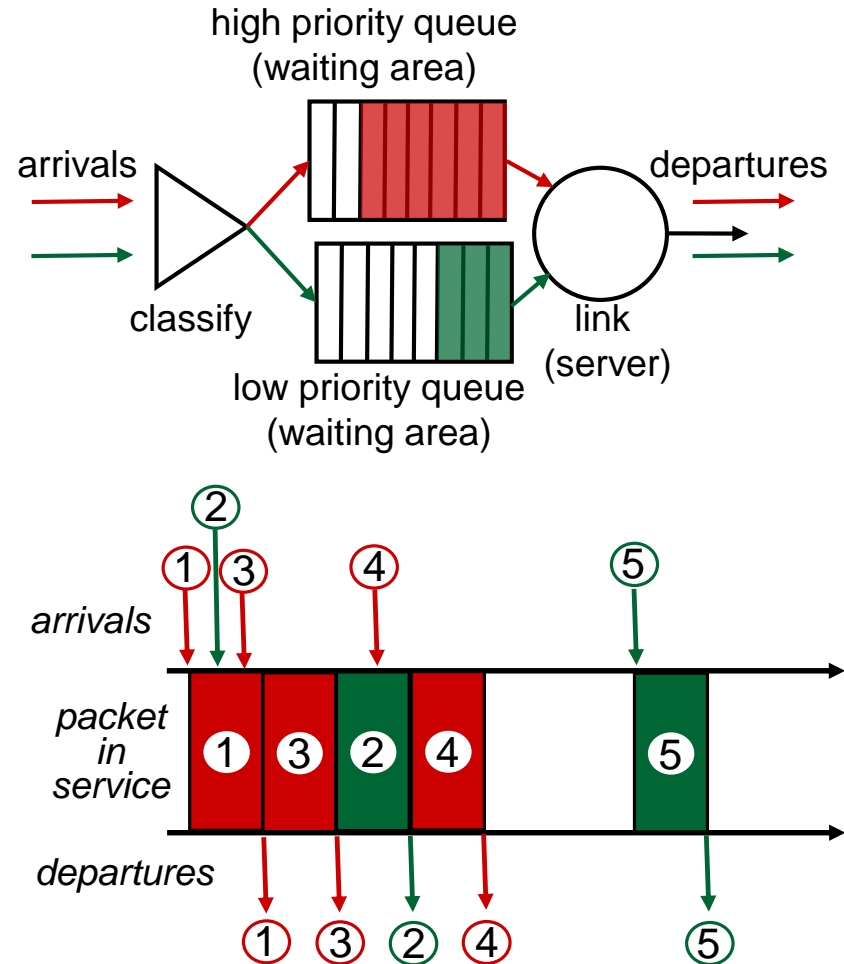
- *scheduling*: choose next packet to send on link
- *FIFO (first in first out) scheduling*: send in order of arrival to queue
 - real-world example?
 - *discard policy*: if packet arrives to full queue: who to discard?
 - *tail drop*: drop arriving packet
 - *priority*: drop/remove on priority basis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



Scheduling policies: priority

priority scheduling: send highest priority queued packet

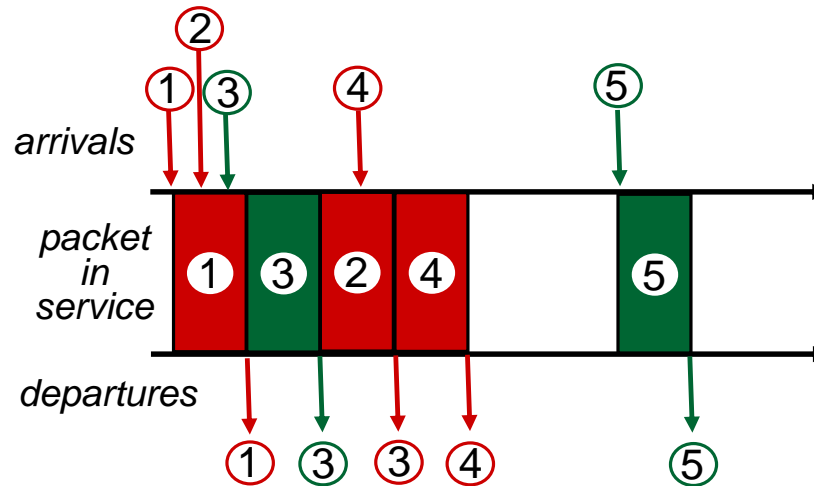
- multiple *classes*, with different priorities
 - class may depend on marking or other header info, e.g. IP source/dest, port numbers, etc.
 - real world example?



Scheduling policies: round-robin

Round Robin (RR) scheduling:

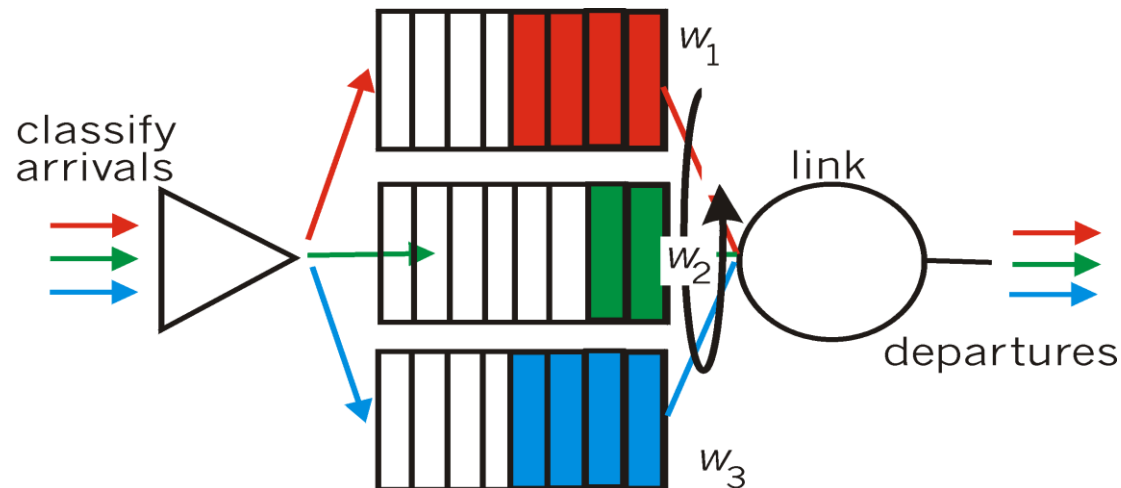
- multiple classes
- cyclically scan class queues, sending one complete packet from each class (if available)
- real world example?



Scheduling policies: WFQ

Weighted Fair Queuing (WFQ):

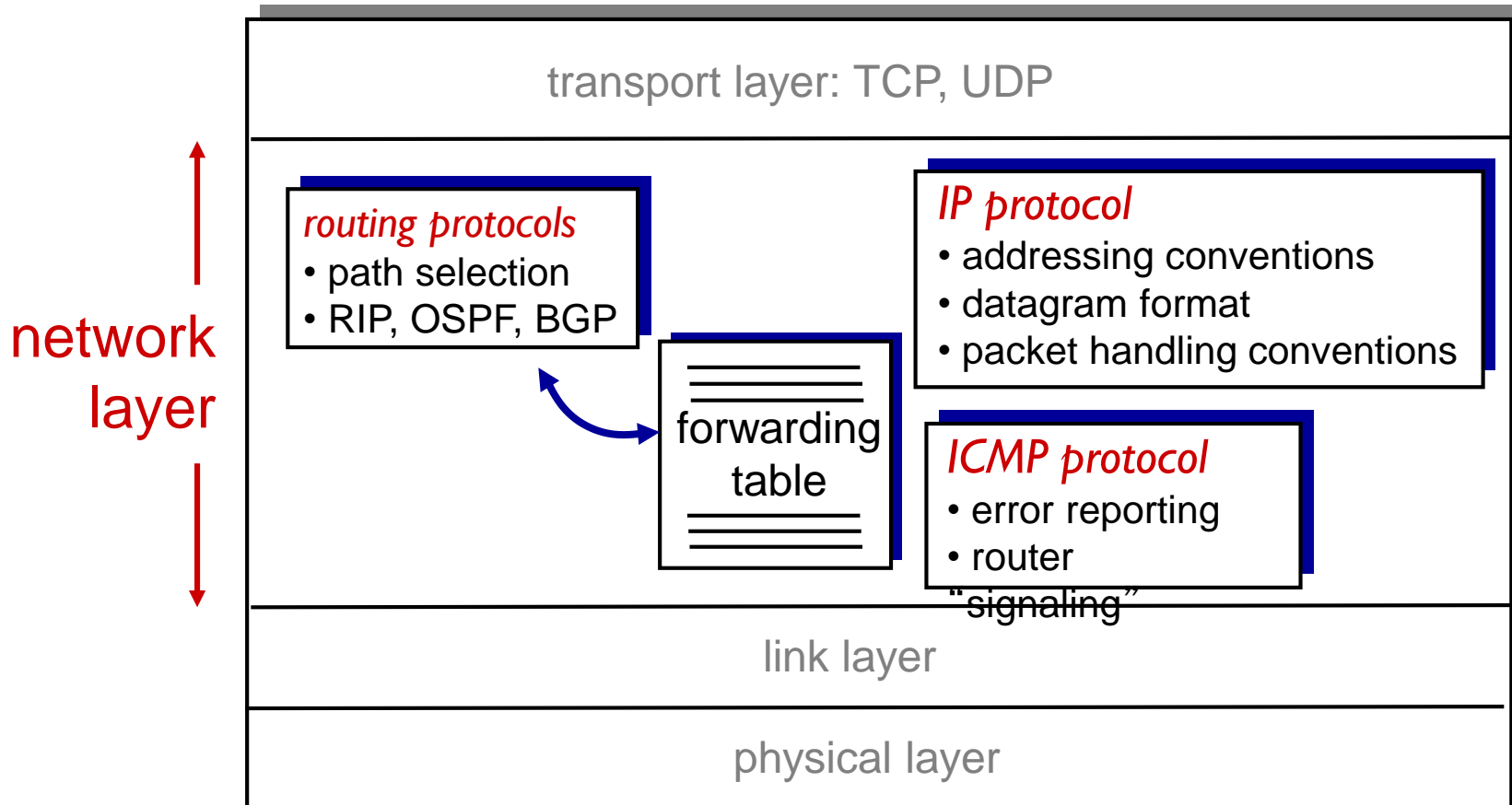
- generalized Round Robin
- each class gets weighted amount of service in each cycle
- real-world example?



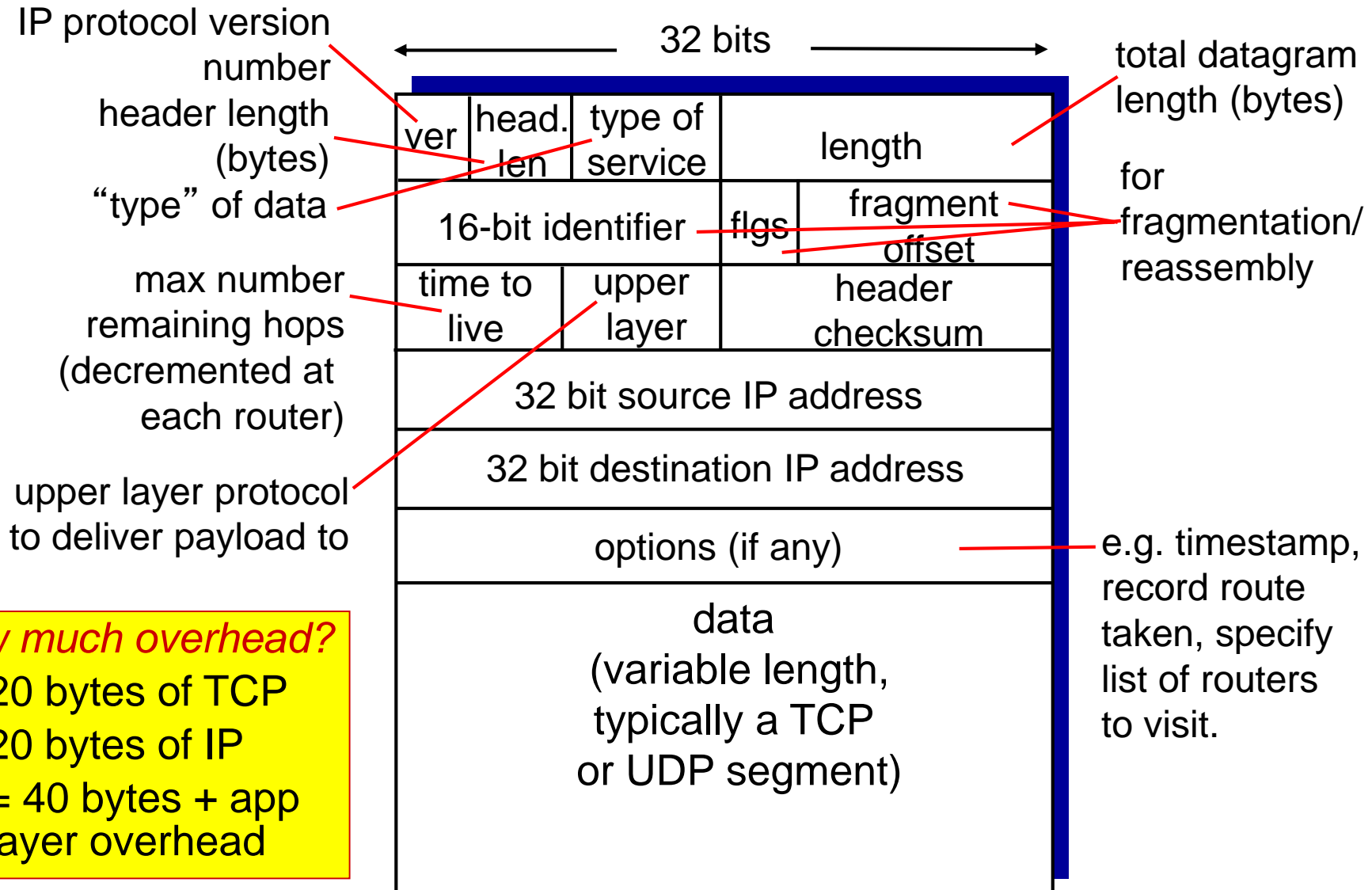
IP

The Internet network layer

host, router network layer functions:



IP datagram format

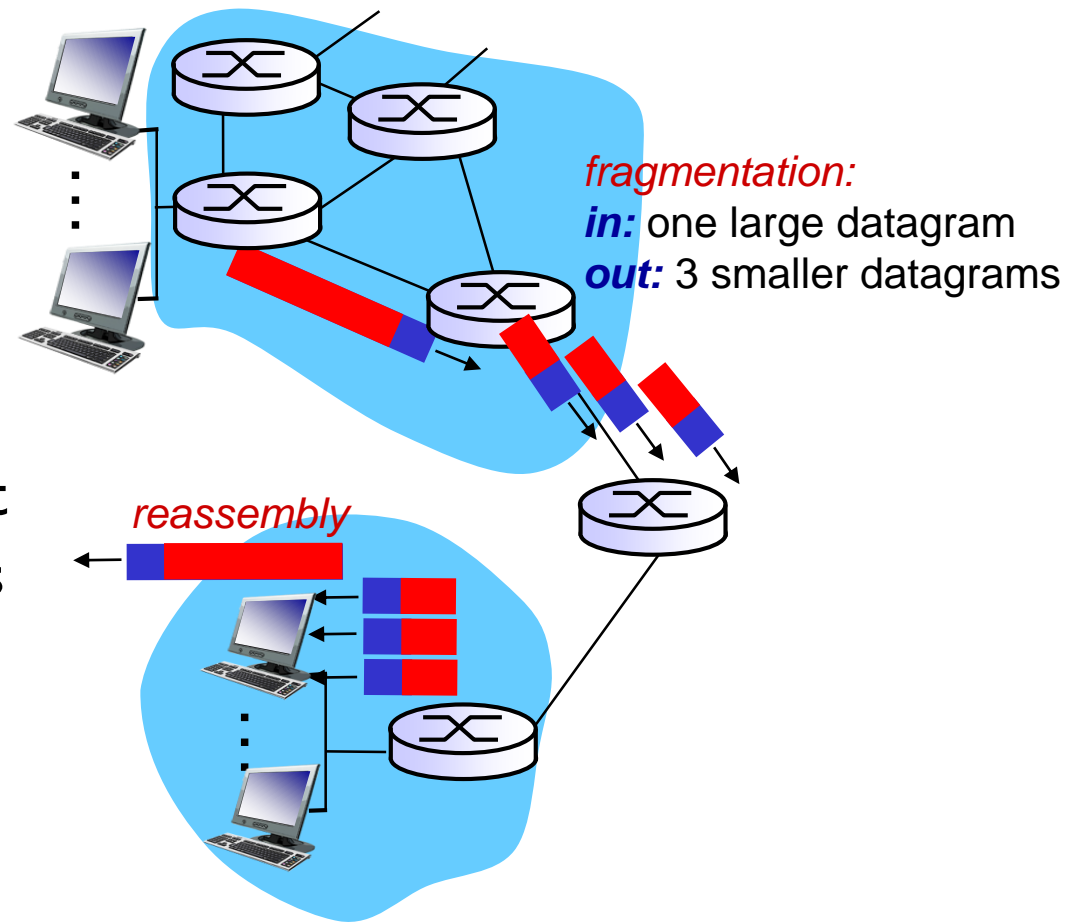


how much overhead?

- ❖ 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 (e.g., ethernet)
 - 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

	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	

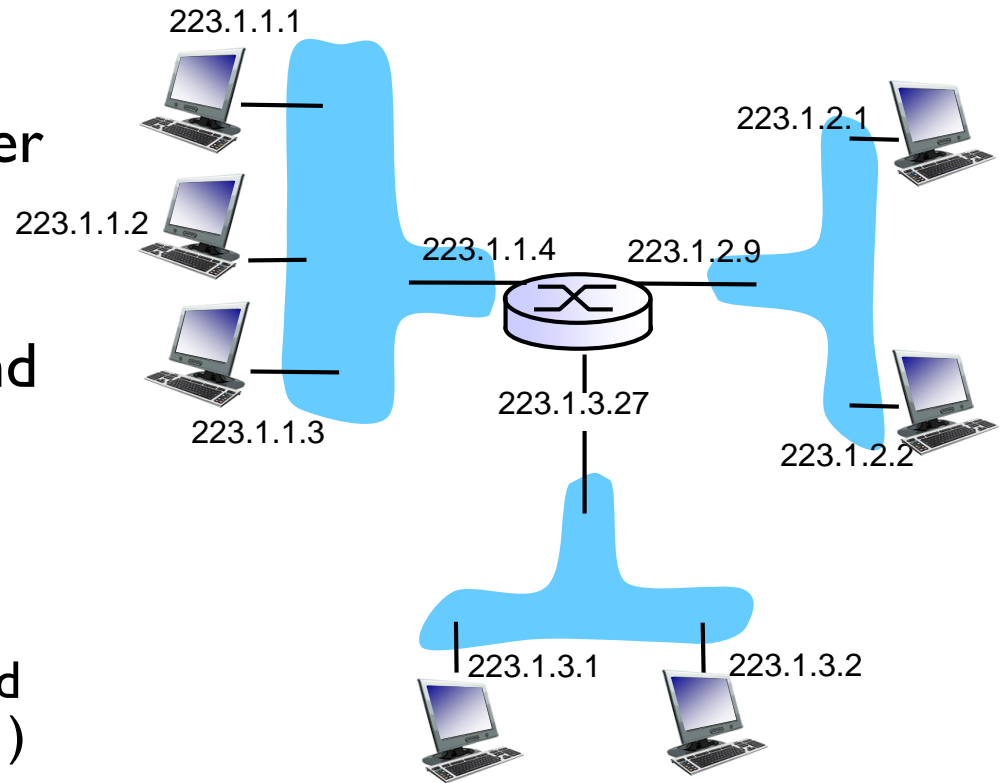
	length	ID	fragflag	offset	
	=1500	=x	=1	=185	

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

IP Address (v4 only)

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**



$$223.1.1.1 = \underbrace{11011111}_{223} \underbrace{00000001}_1 \underbrace{00000001}_1 \underbrace{00000001}_1$$

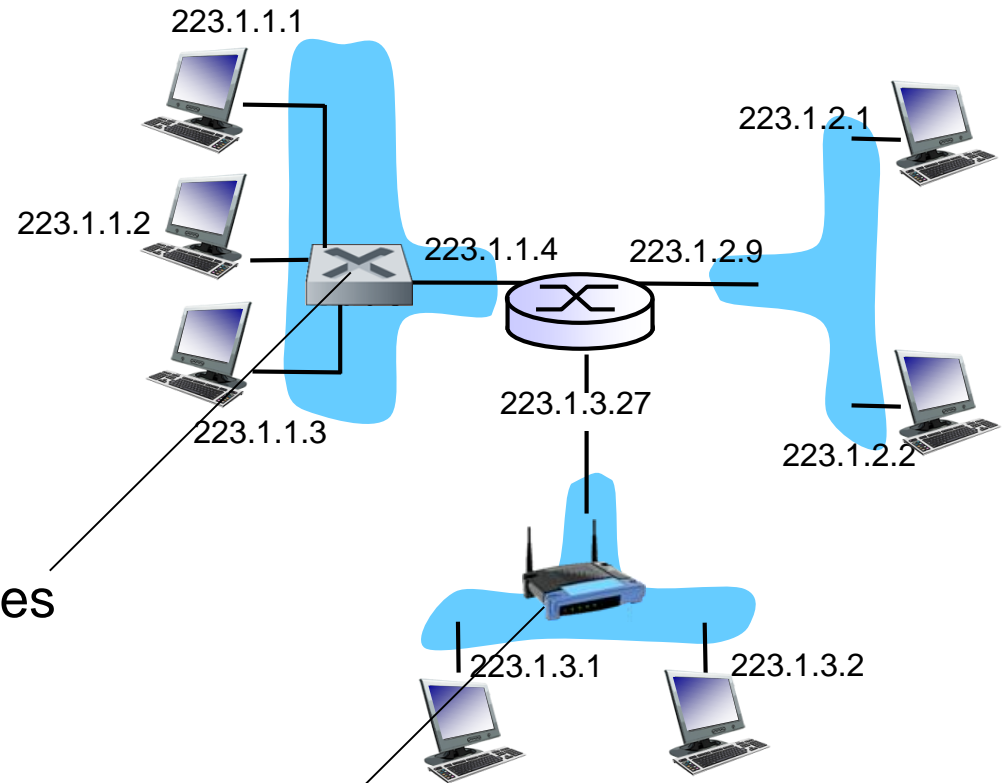
IP addressing: introduction

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

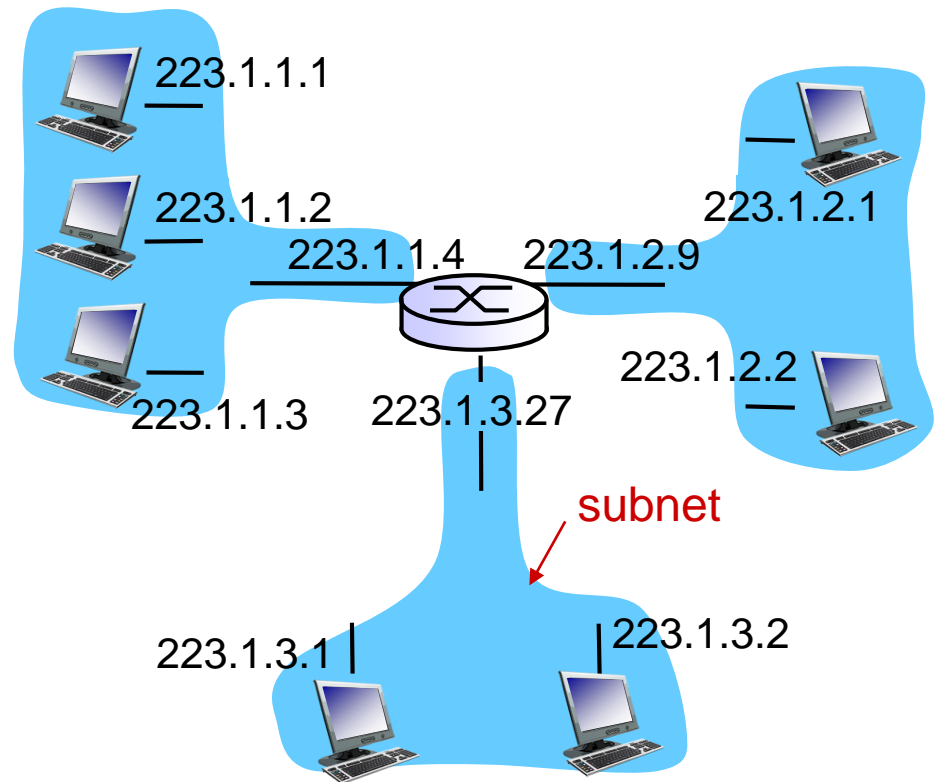
Network addresses

■ IP address:

- **network** part - high order bits
- **host** part - low order bits

■ *what's a subnet ?*

- device interfaces with same network part of IP address
- can physically reach each other *without intervening router*

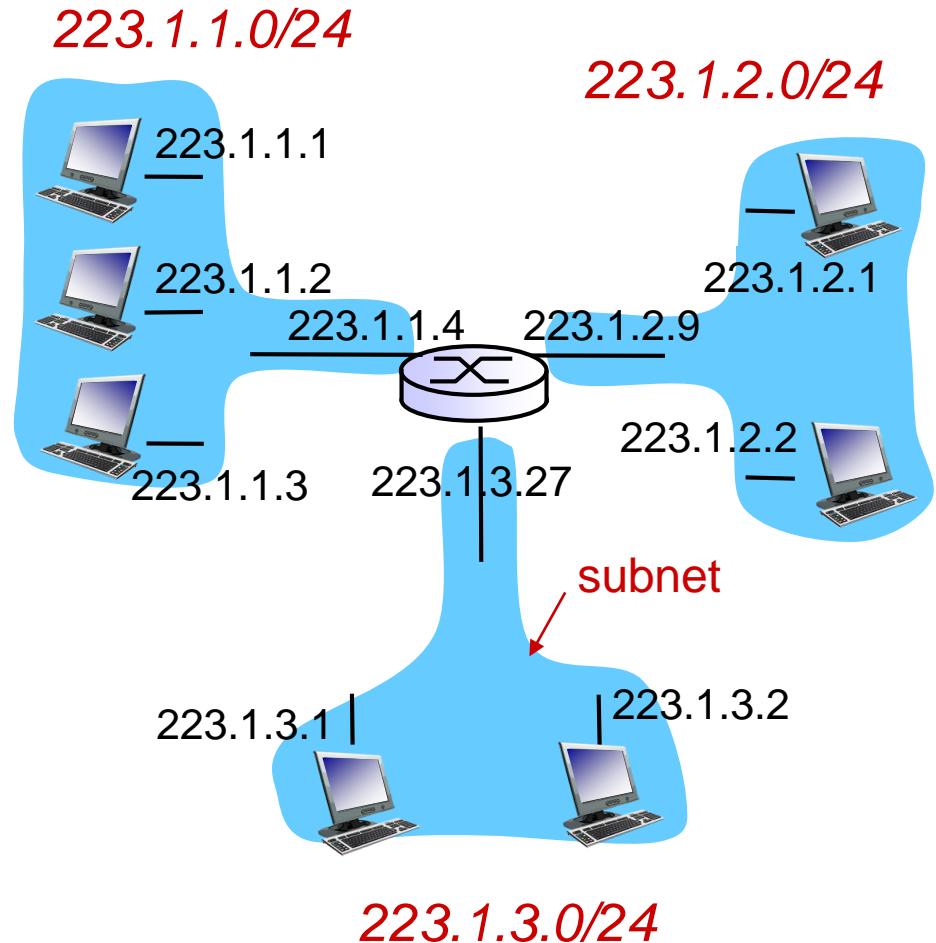


network consisting of 3 subnets

Subnets

recipe

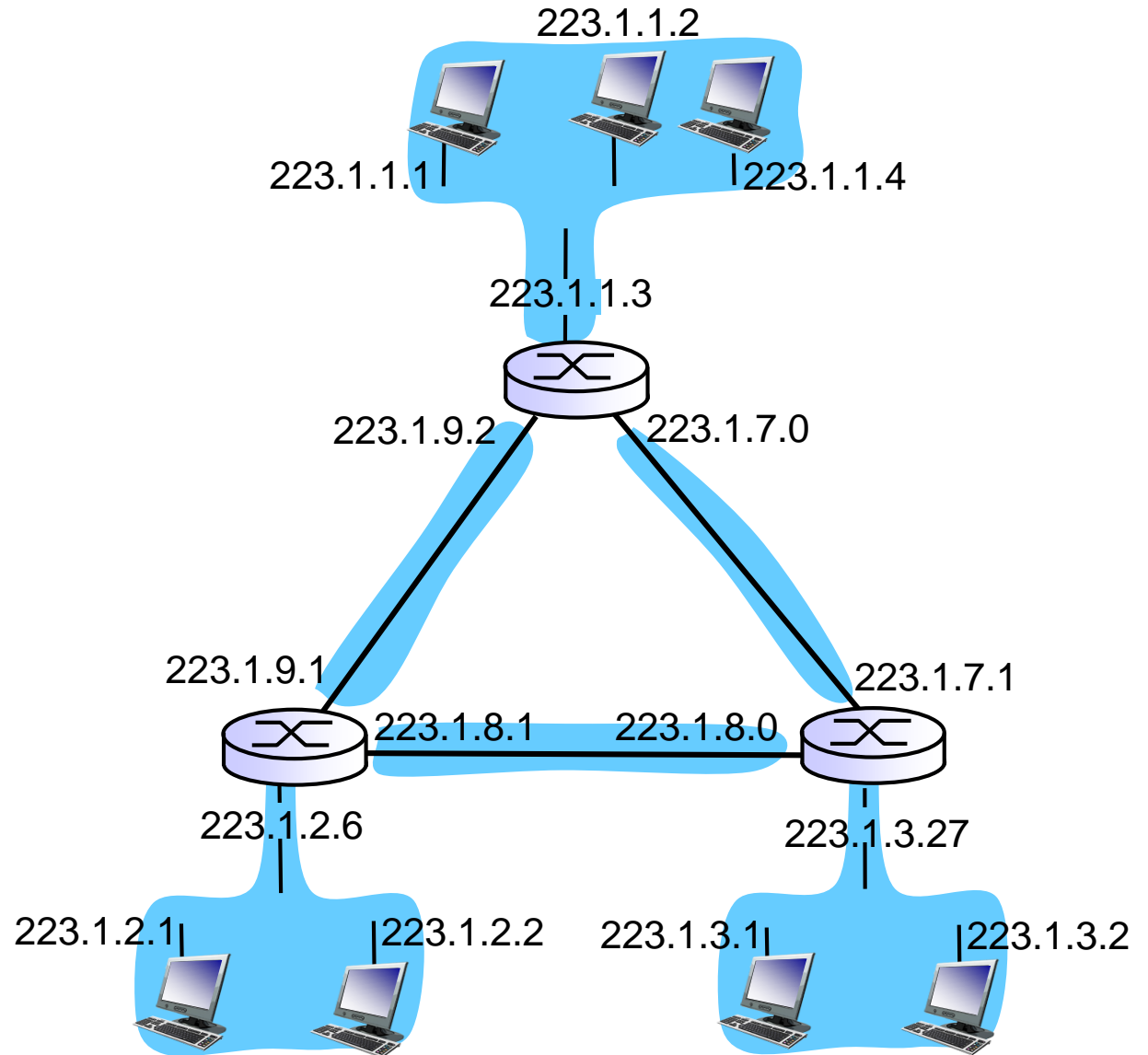
- to determine the subnets, detach each interface from its host or router, creating islands of isolated networks
- each isolated network is called a *subnet*



subnet mask: /24

Subnets

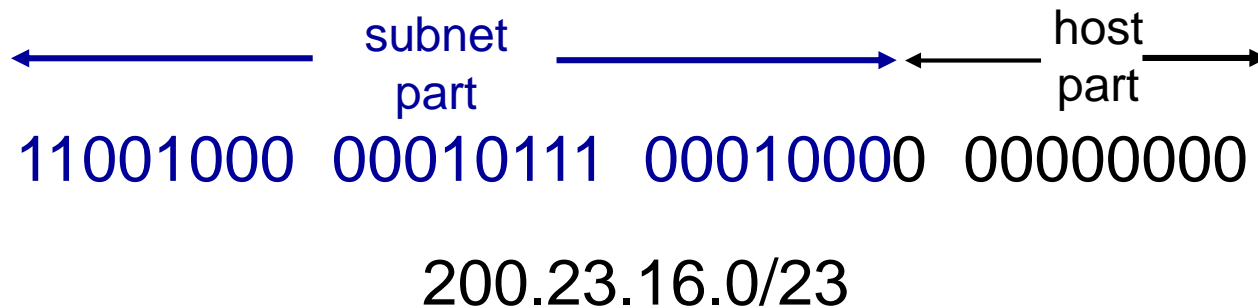
how many?



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



DHCP

IP addresses: how to get one?

Q: How does a *host* get an 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

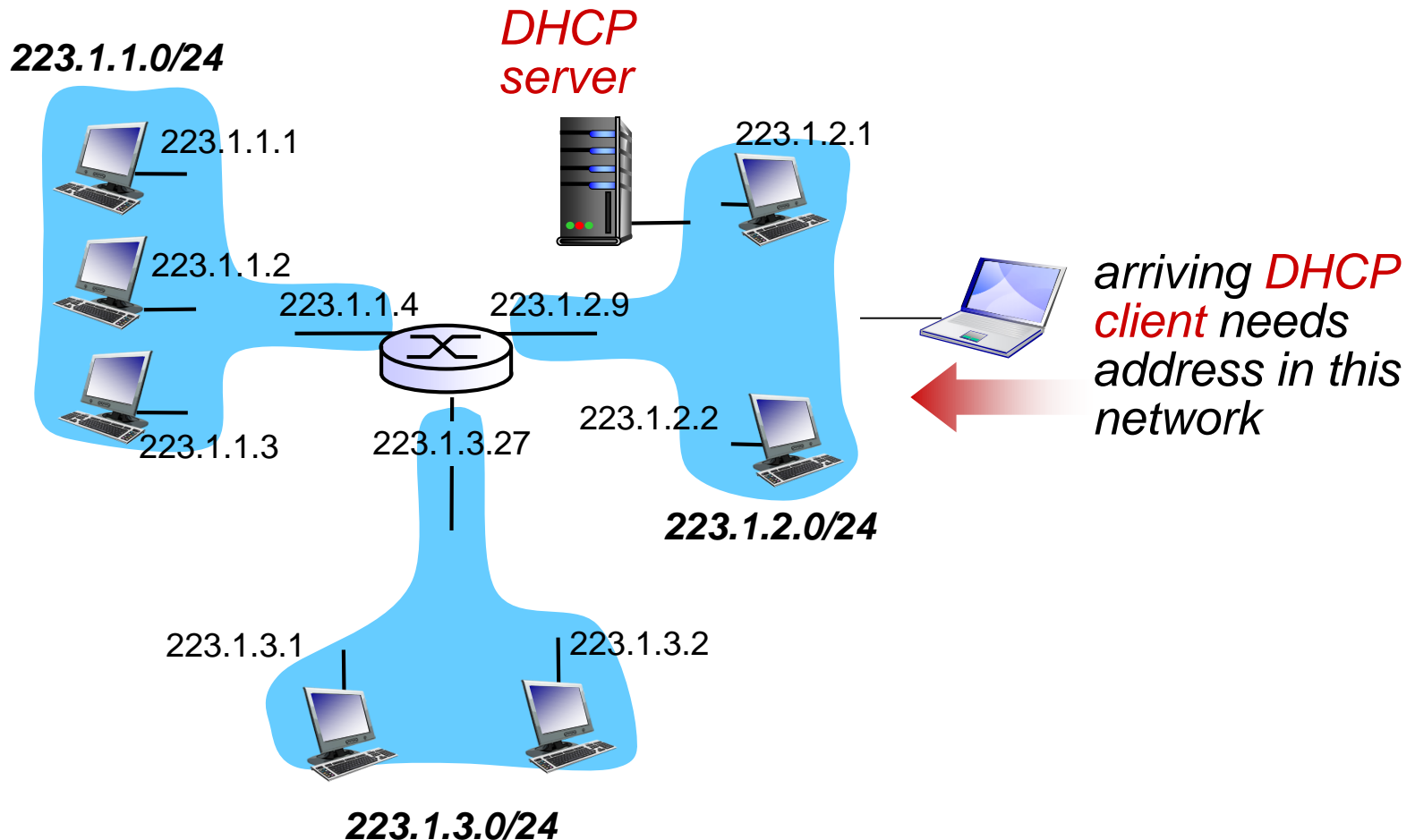
goal: allow host to *dynamically* obtain its IP address from network server when it joins network

- can renew its lease on address in use
- allows reuse of addresses (only hold address while connected or “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 server: 223.1.2.5

DHCP discover

arriving
client



Broadcast: is there a
DHCP server out there?

DHCP offer

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

DHCP request

Broadcast: OK. I'll take
that IP address!

DHCP ACK

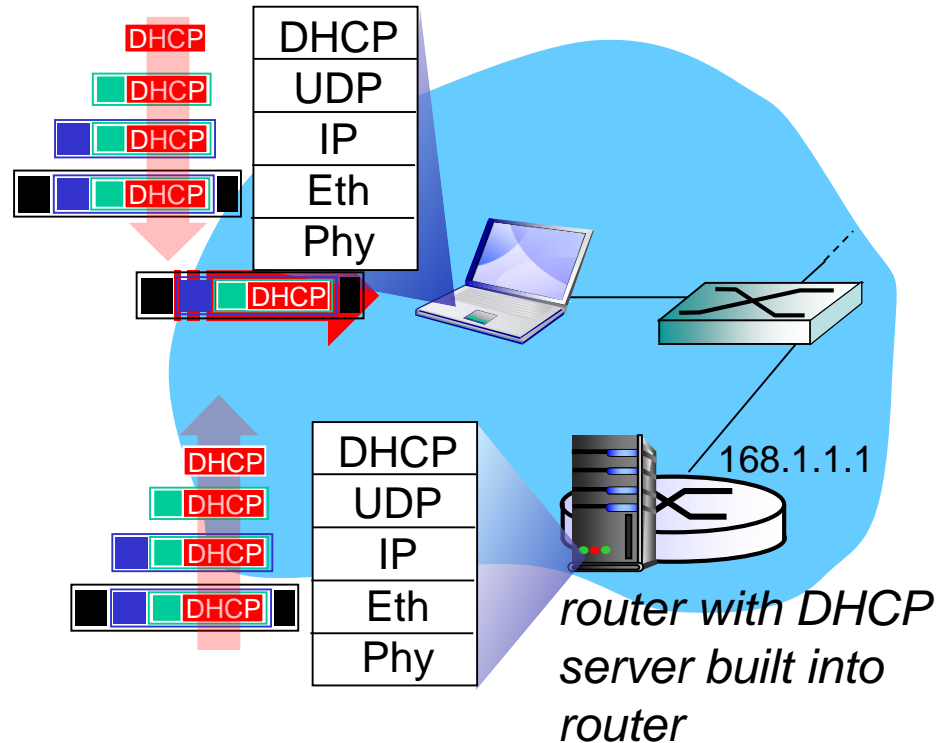
Broadcast: OK. You've
got that IP address!

DHCP: more than IP addresses

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

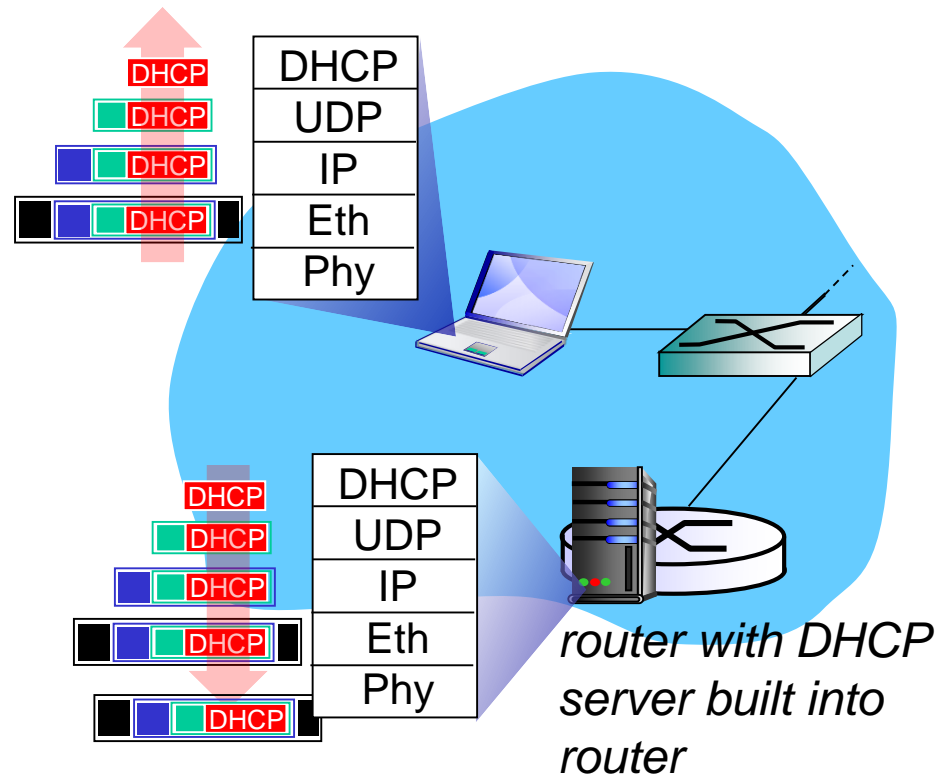
- address of first-hop **router** for client
- name and IP address of **DNS** sever
- network **mask** (indicating network versus host portion of address)

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 802.11 Ethernet
- Ethernet frame broadcast (dest: FFFFFFFF) 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)

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

.....

request

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."

reply

IP addresses: how to get one?

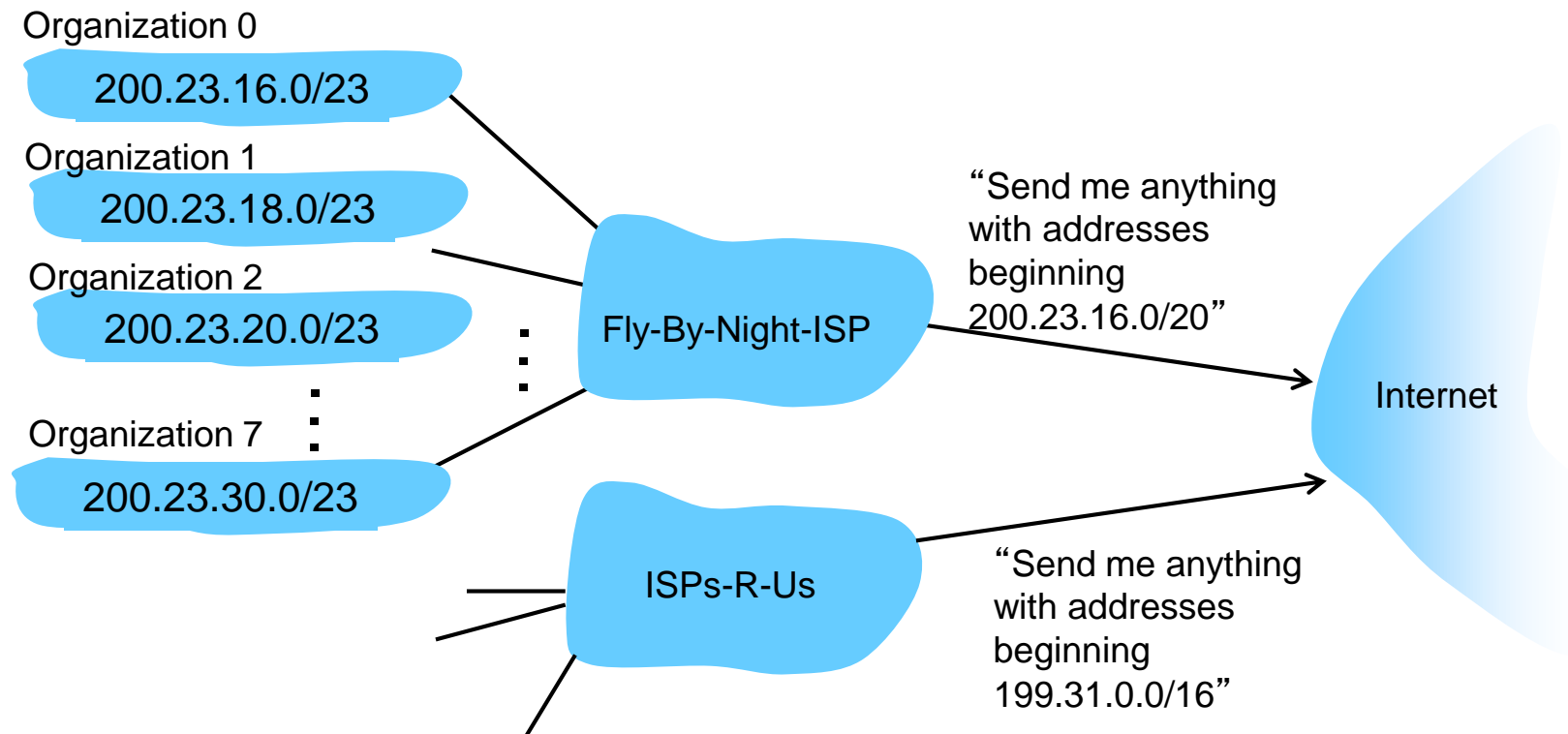
Q: how does *network* get subnet part of IP addr?

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

ISP's block	<u>11001000</u>	<u>00010111</u>	<u>00010000</u>	00000000	200.23.16.0/20
Organization 0	<u>11001000</u>	<u>00010111</u>	<u>0001000</u>	00000000	200.23.16.0/23
Organization 1	<u>11001000</u>	<u>00010111</u>	<u>0001001</u>	00000000	200.23.18.0/23
Organization 2	<u>11001000</u>	<u>00010111</u>	<u>0001010</u>	00000000	200.23.20.0/23
...
Organization 7	<u>11001000</u>	<u>00010111</u>	<u>0001111</u>	00000000	200.23.30.0/23

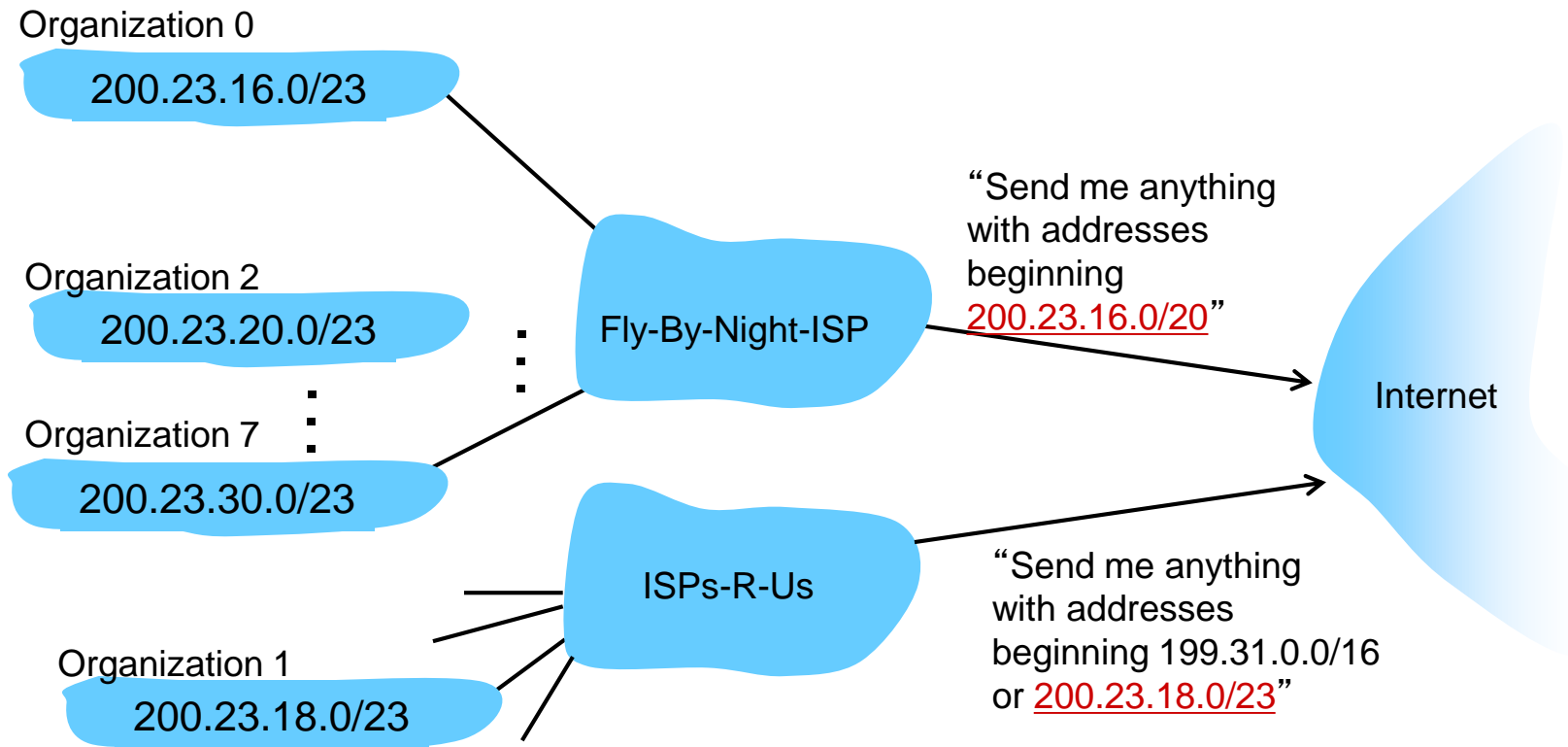
Hierarchical addressing: route aggregation

hierarchical addressing allows efficient advertisement of routing information:



Hierarchical addressing: more specific routes

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



IP addressing: the last word...

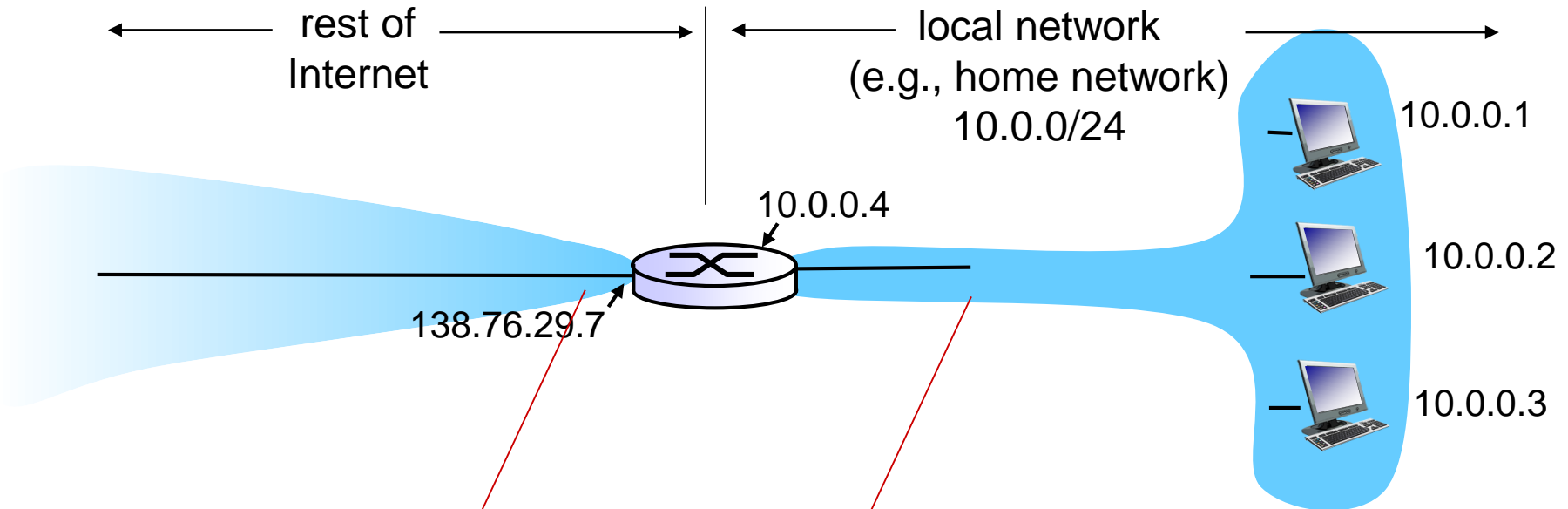
Q: how does an ISP get block of addresses?

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

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

NAT

NAT: network address translation



all datagrams *leaving* local network have *same* single 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

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

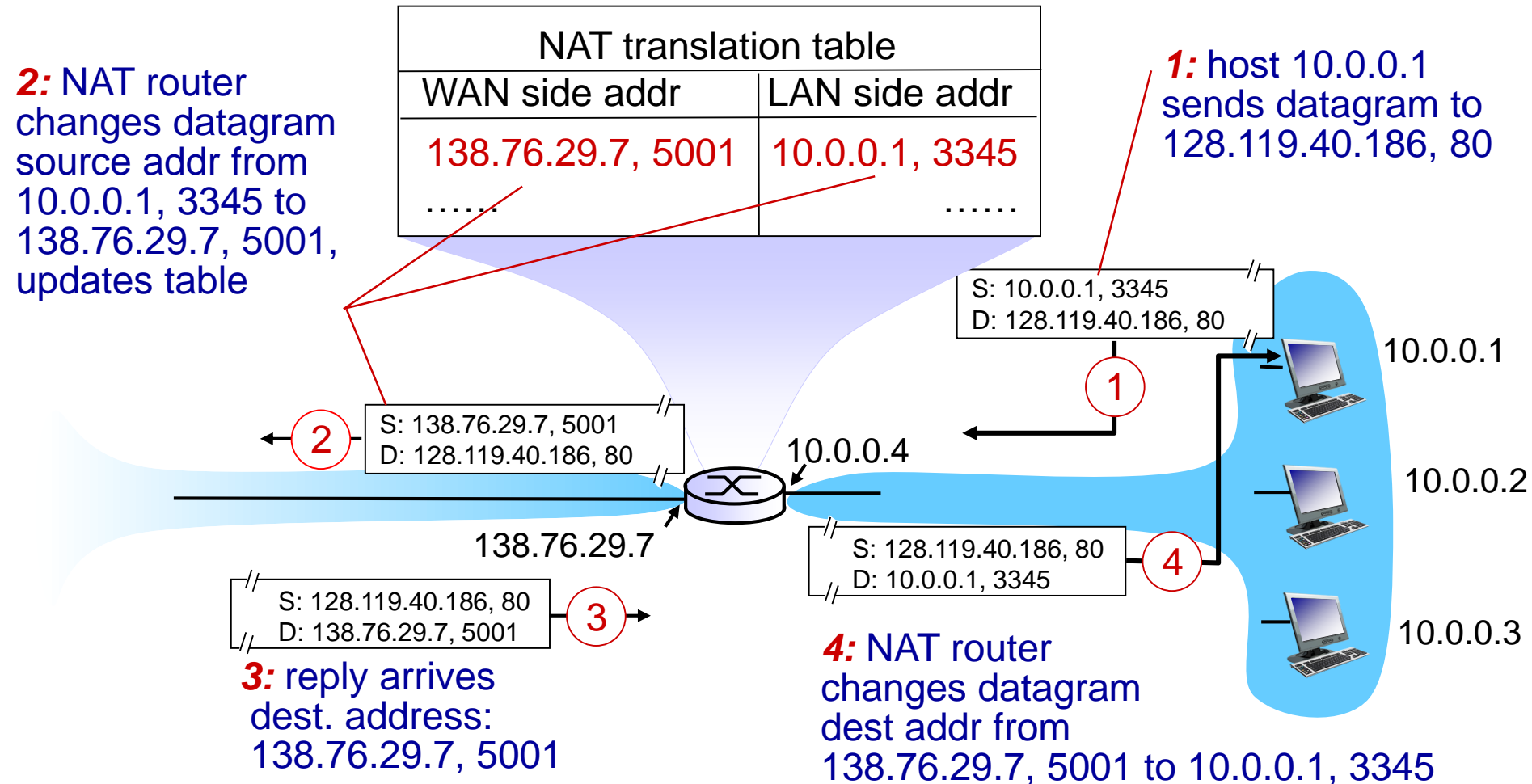
- range of addresses not needed from ISP: just one IP address for all devices
- can change addresses of devices in local network without notifying outside world
- can change ISP without changing addresses of devices in local network
- devices inside local net not explicitly addressable, visible by outside 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 **WAN-side** address!
- NAT is **controversial**: (**Why?**)
 - routers should only process up to IP layer (no port #!)
 - address shortage should be solved by IPv6
 - violates end-to-end argument
 - P2P applications are more difficult behind NAT
 - NAT traversal: *what if client wants to connect to server behind NAT?*