

# CS 305: Computer Networks

## Fall 2022

### **Lecture 9: Network Layer – The Data Plane**

**Ming Tang**

Department of Computer Science and Engineering  
Southern University of Science and Technology (SUSTech)

# Chapter 3 outline

3.1 transport-layer services

3.2 multiplexing and demultiplexing

3.3 connectionless transport: UDP

3.4 principles of reliable data transfer

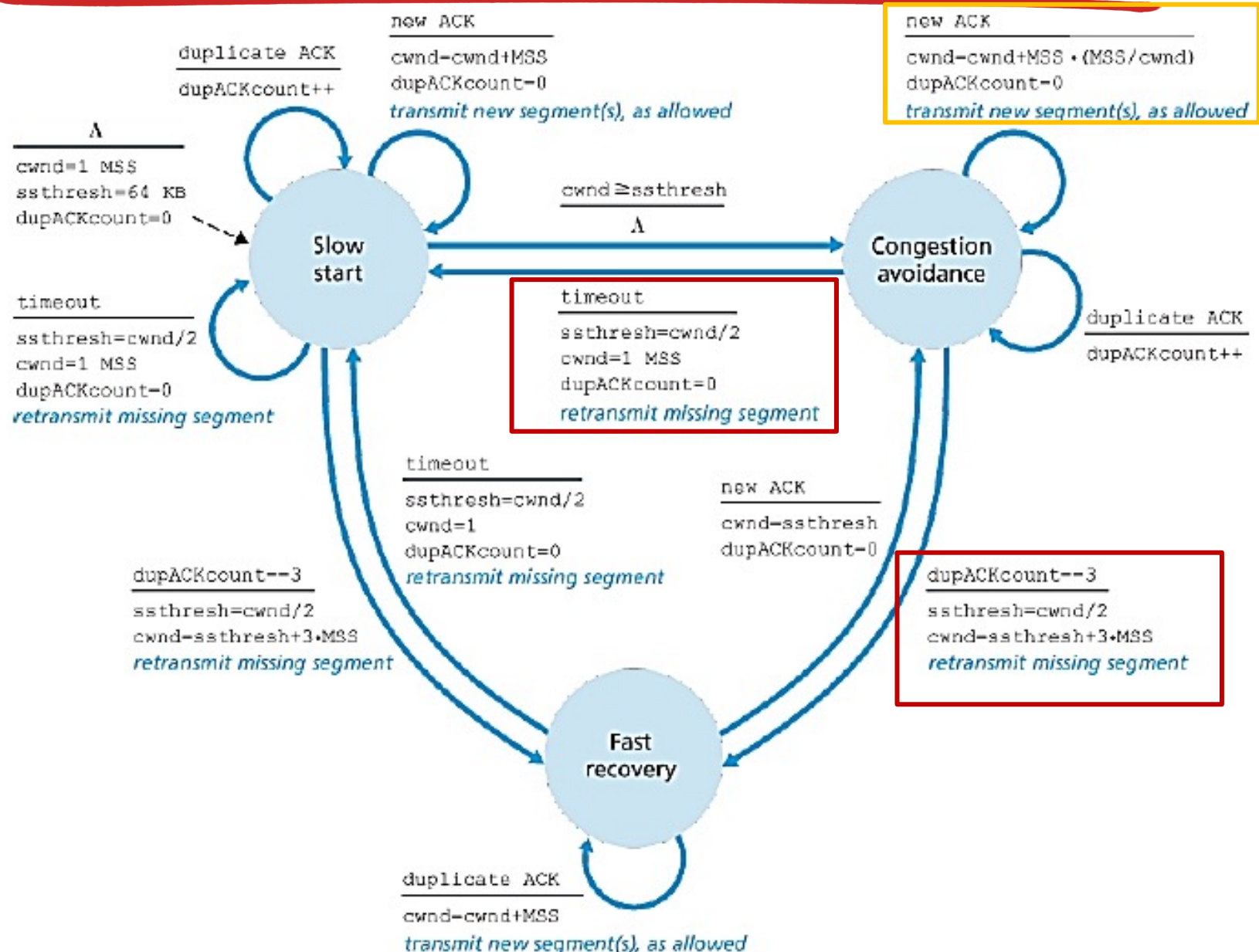
3.5 connection-oriented transport: TCP

- segment structure
- reliable data transfer
- flow control
- connection management

3.6 principles of congestion control

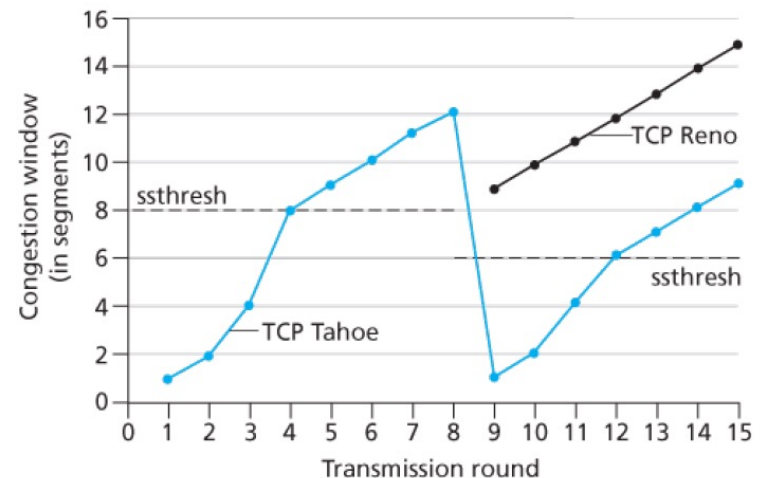
3.7 TCP congestion control

# TCP Congestion Control: FSM

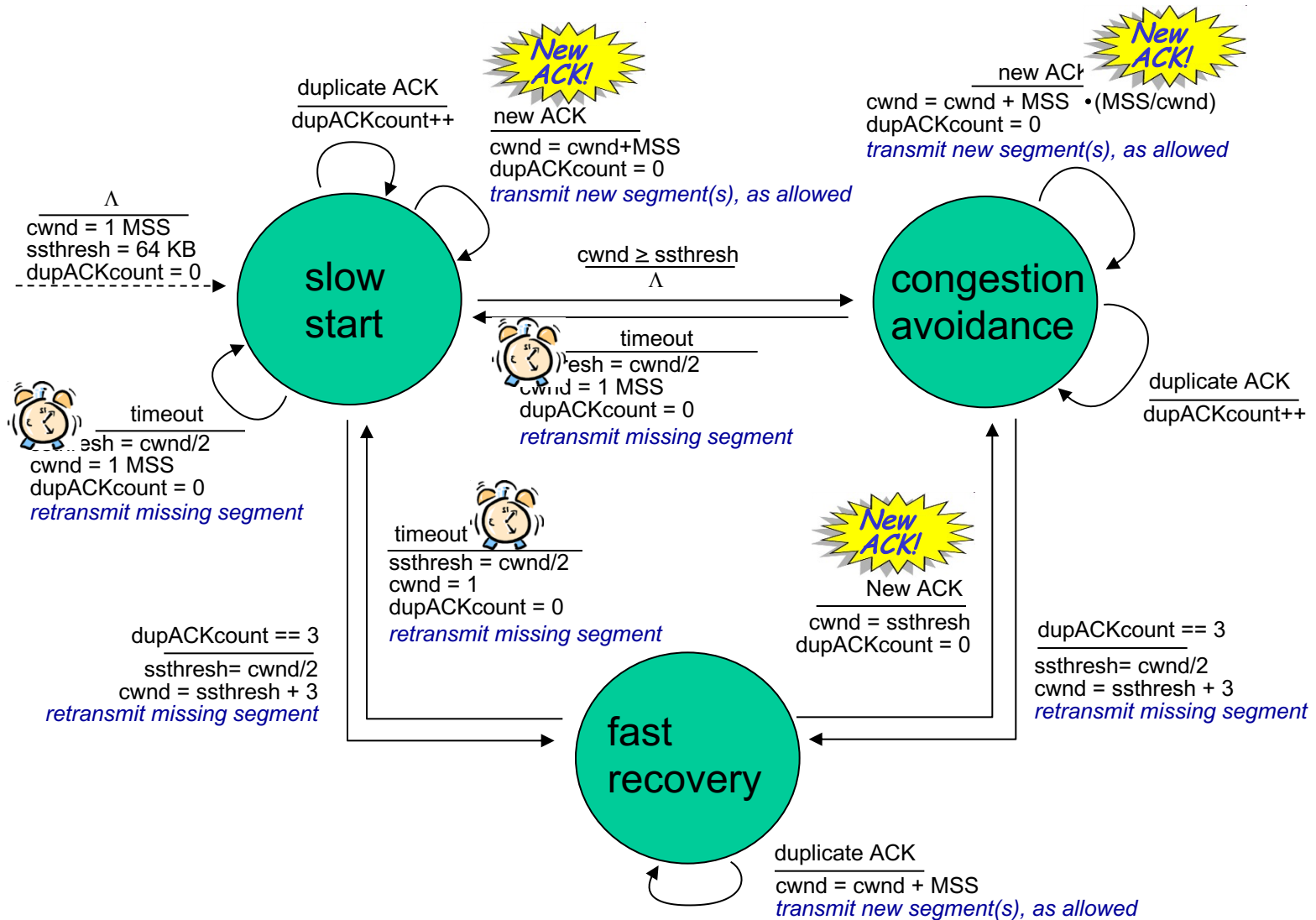


# TCP Fast Recovery

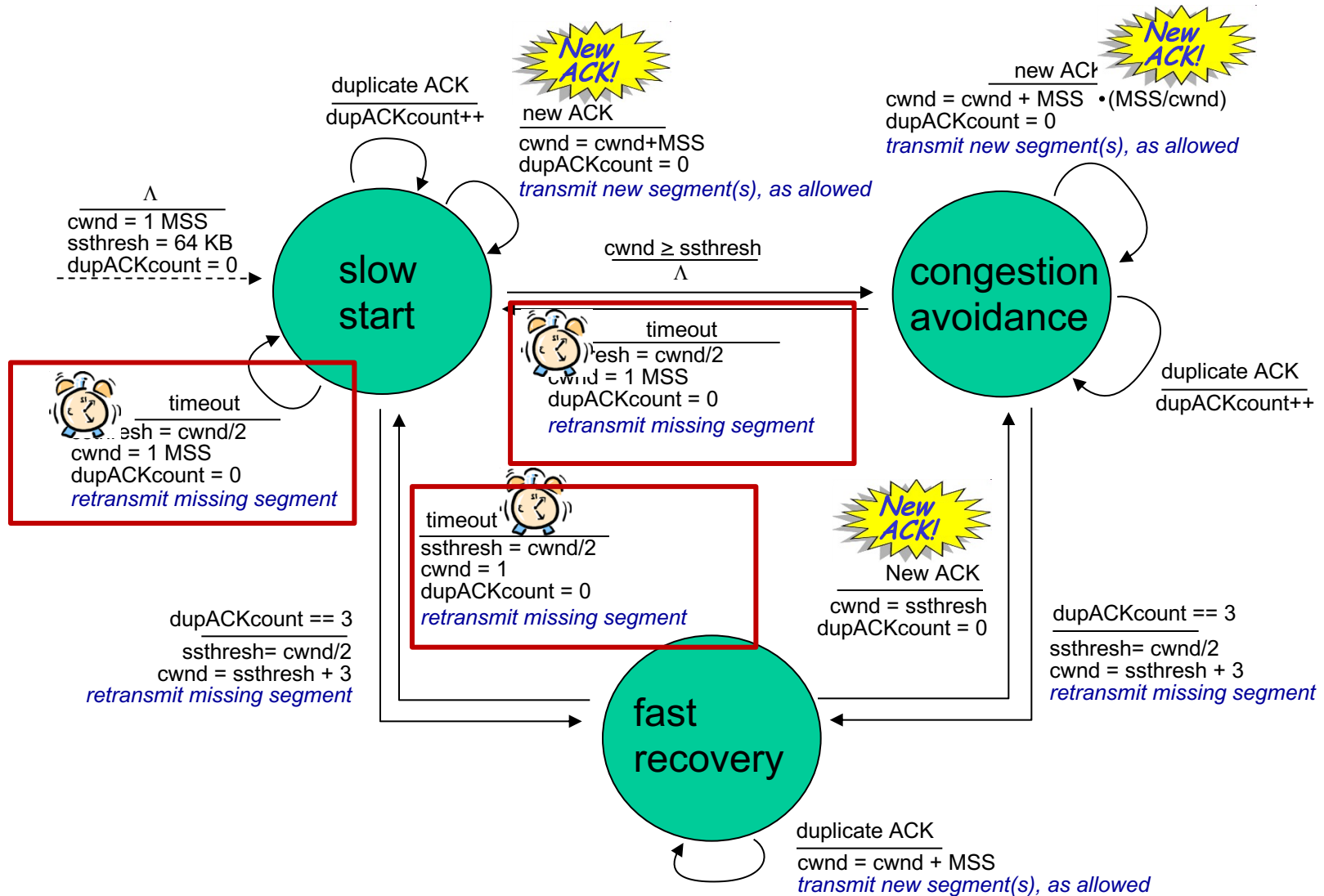
- ❖ Trigger (RENO) : triple duplicate ACKs
- ❖  $ssthresh = cwnd / 2; cwnd = ssthresh + 3MSS$
- ❖ The value of  $cwnd$  is **increased by 1 MSS** for every duplicate ACK received for the **missing segment** that caused TCP to enter the fast-recovery state
- ❖ when an ACK arrives for the missing segment, enter congestion avoidance



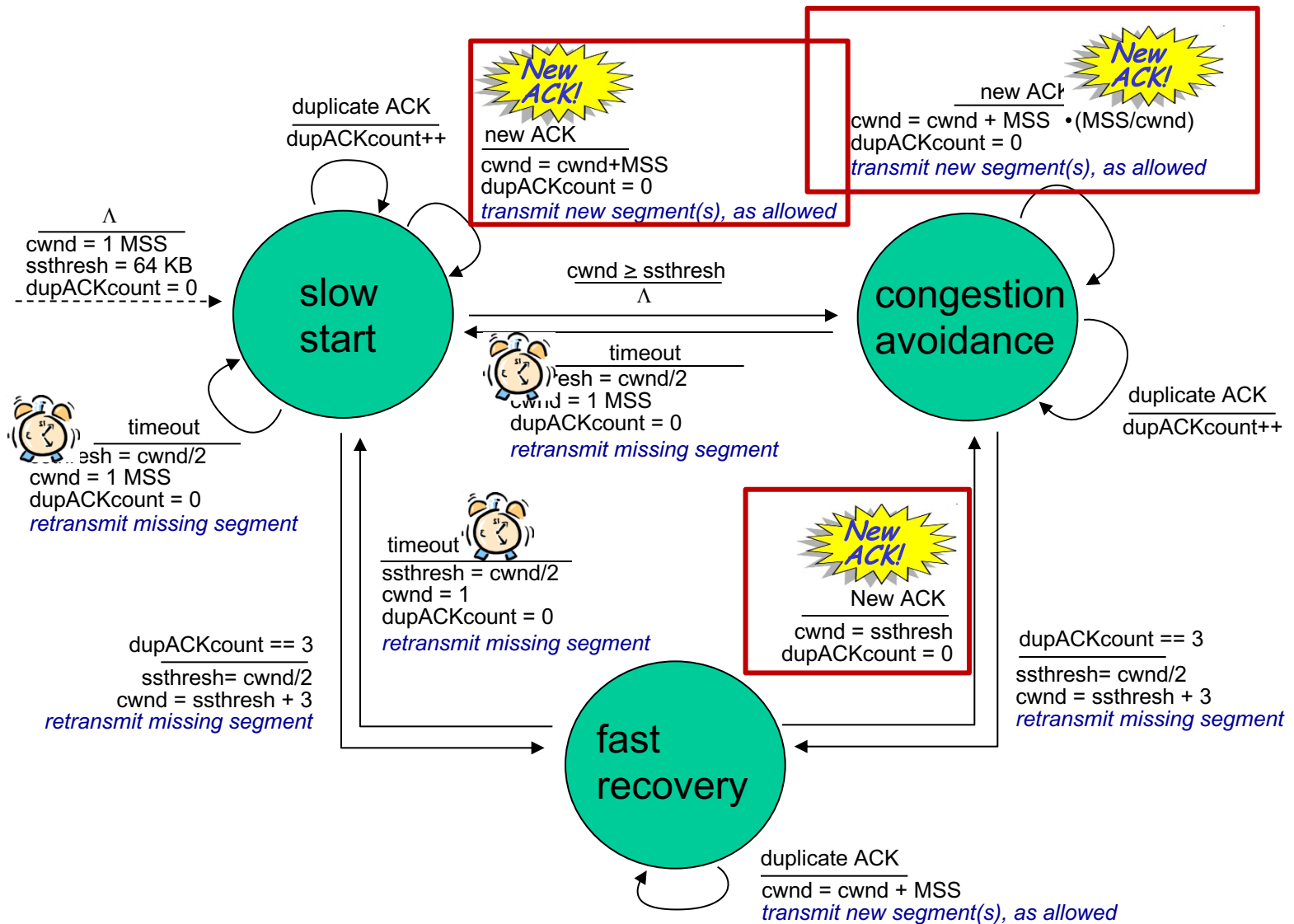
# Summary: TCP Congestion Control



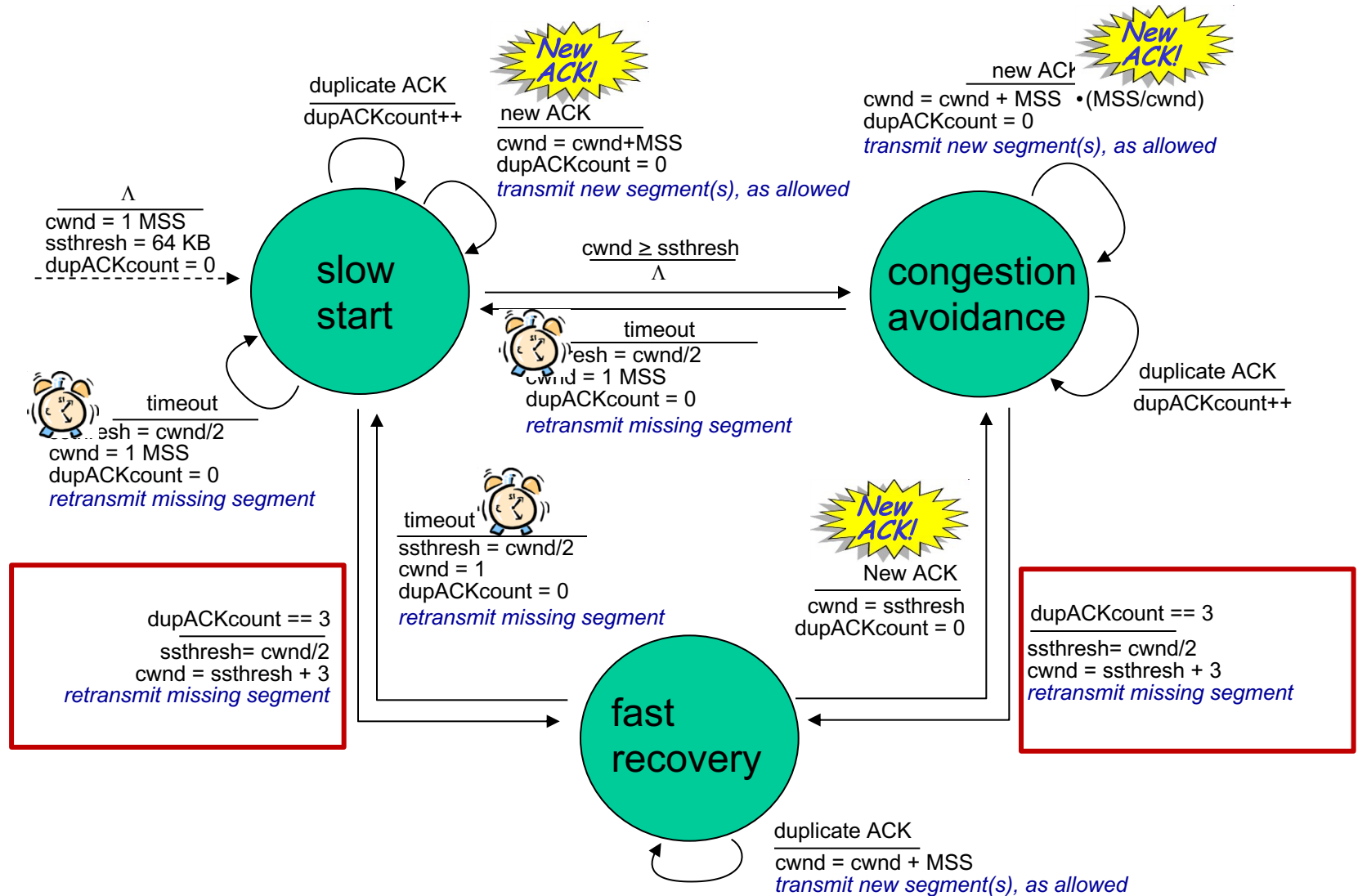
# Summary: TCP Congestion Control



# Summary: TCP Congestion Control



# Summary: TCP Congestion Control



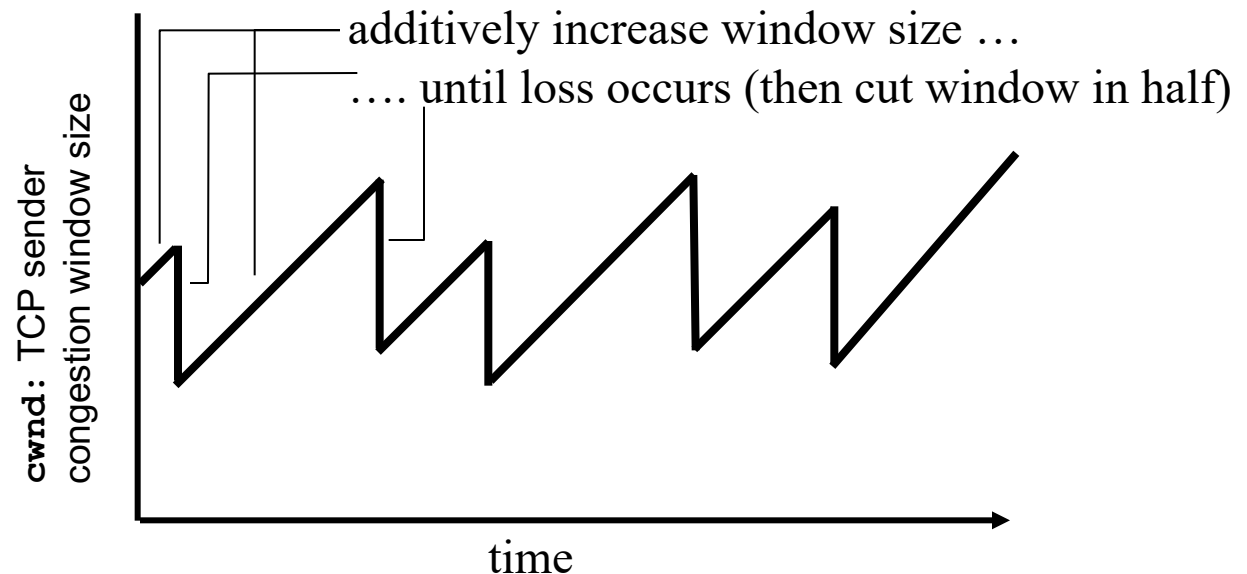


# TCP congestion control: additive increase multiplicative decrease

**Approach:** sender increases transmission rate (window size), **probing** for usable bandwidth, until loss occurs

- *additive increase*: increase **cwnd** by 1 MSS every RTT until loss detected
- *multiplicative decrease*: cut **cwnd** in half after loss

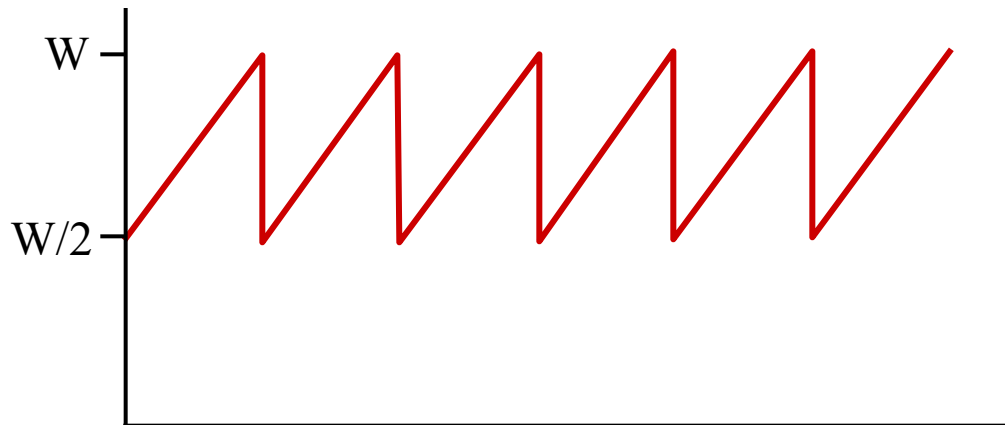
AIMD: probing for bandwidth



# TCP throughput

- ❖ Avg. TCP throughput as function of window size, RTT?
  - ignore slow start, assume always data to send
- ❖  $W$ : window size (measured in bytes) where loss occurs
  - avg. window size (# in-flight bytes) is  $\frac{3}{4} W$
  - avg. throughput is  $\frac{3}{4}W$  per RTT

$$\text{avg TCP thruput} = \frac{3}{4} \frac{W}{\text{RTT}} \text{ bytes/sec}$$



# TCP Futures: TCP over “long, fat pipes”

$$\text{avg TCP thruput} = \frac{3}{4} \frac{W}{\text{RTT}} \text{ bytes/sec}$$

- ❖ example: 1500 byte segments, 100ms RTT, want 10 Gbps throughput
- ❖ requires  $W = 83,333$  in-flight segments
- ❖ throughput in terms of segment loss probability,  $L$  [Mathis 1997]:

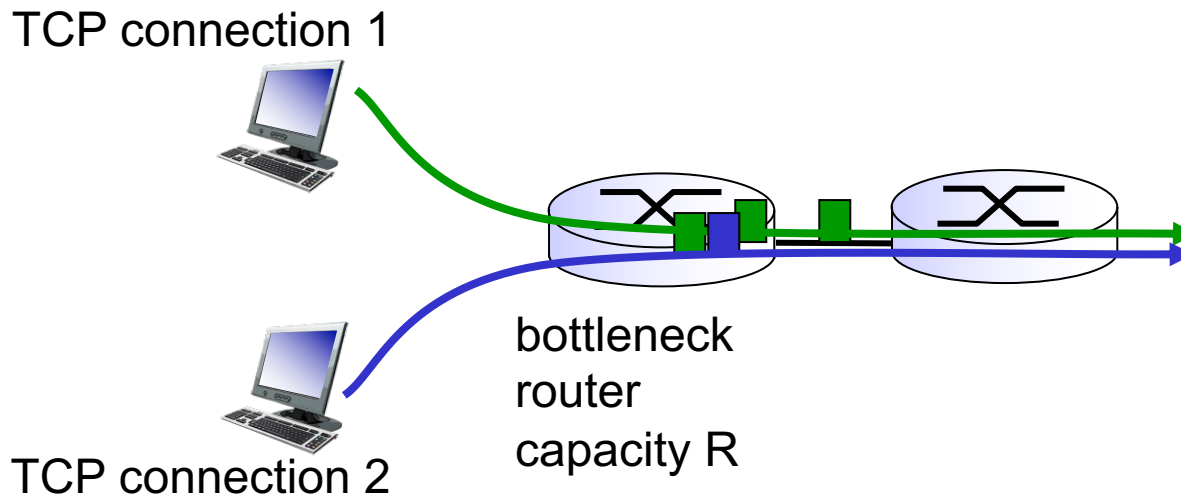
$$\text{TCP throughput} = \frac{1.22 \cdot \text{MSS}}{\text{RTT} \sqrt{L}}$$

→ to achieve 10 Gbps throughput, need a loss rate of  $L = 2 \cdot 10^{-10}$  – *a very small loss rate!*

- ❖ new versions of TCP for high-speed

# TCP Fairness

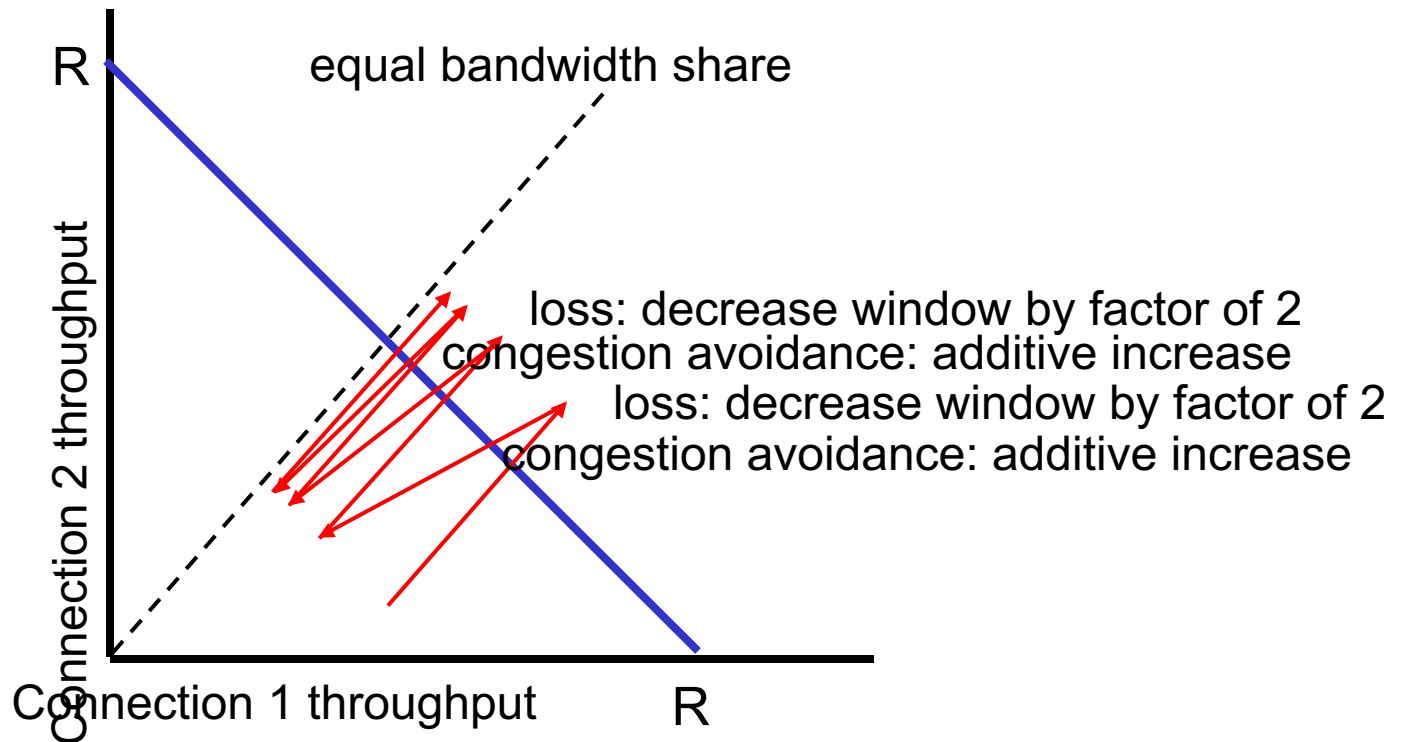
**Fairness goal:** if  $K$  TCP sessions share same bottleneck link of bandwidth  $R$ , each should have average rate of  $R/K$



# Why is TCP fair?

two competing sessions:

- ❖ additive increase gives slope of 1, as throughput increases
- ❖ multiplicative decrease decreases throughput proportionally



# Fairness (more)

## *Fairness and UDP*

- ❖ multimedia apps often do not use TCP
  - do not want rate throttled by congestion control
- ❖ use UDP:
  - send audio/video at constant rate, tolerate packet loss
- ❖ UDP sources to crowd out TCP traffic

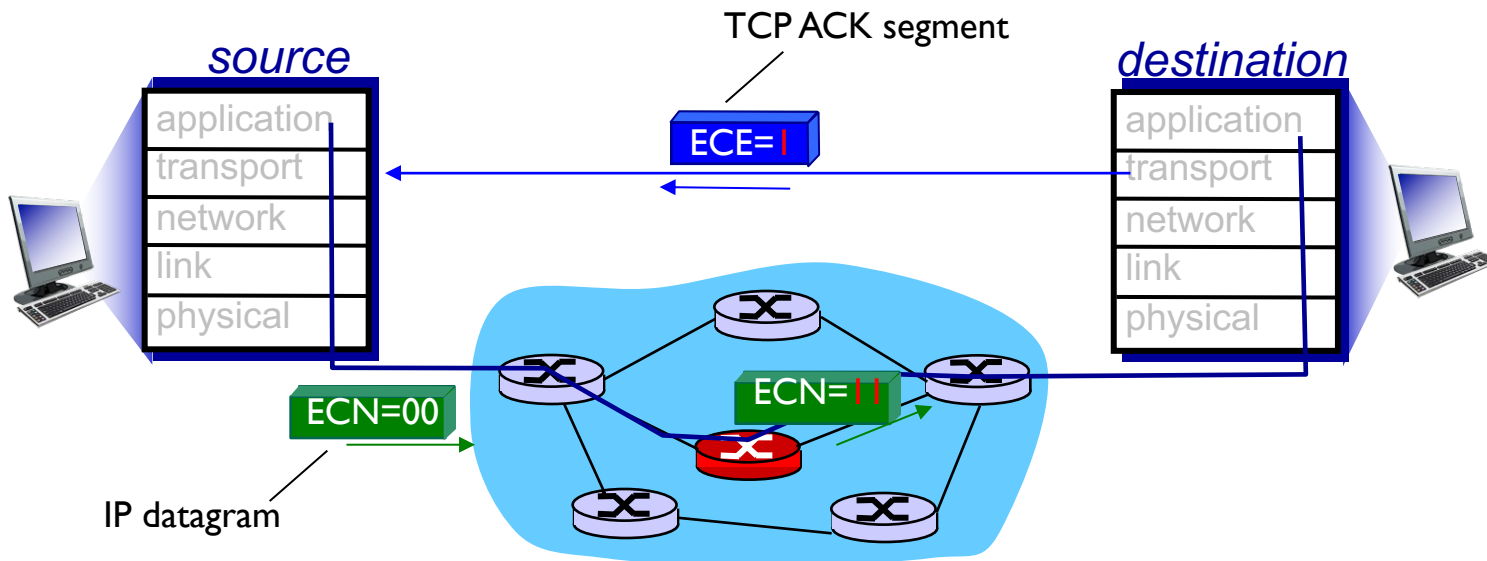
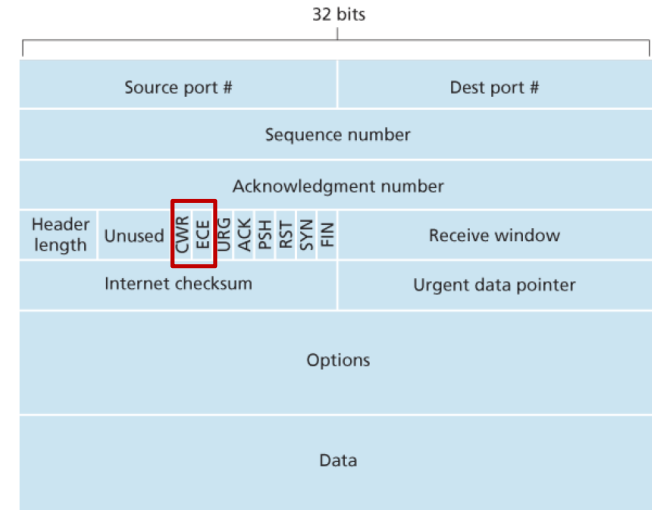
## *Fairness, parallel TCP connections*

- ❖ application can open multiple parallel connections between two hosts
- ❖ web browsers do this
- ❖ e.g., link of rate  $R$  with 9 existing connections:
  - new app asks for 1 TCP, gets rate  $R/10$
  - new app asks for 11 TCPs, gets more than  $R/2$

# Explicit Congestion Notification (ECN)

*network-assisted congestion control:*

- two bits in IP header (ToS field) marked *by network router* to indicate congestion
- congestion indication carried to receiving host
- receiver sets ECE bit on receiver-to-sender ACK segment to notify sender of congestion



# Chapter 3: summary

- ❖ principles behind transport layer services:
    - multiplexing, demultiplexing
    - reliable data transfer
    - flow control
    - congestion control
  - ❖ instantiation, implementation in the Internet
    - UDP
    - TCP
- next:
- ❖ leaving the network “edge” (application, transport layers)
  - ❖ into the network “core”



# 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

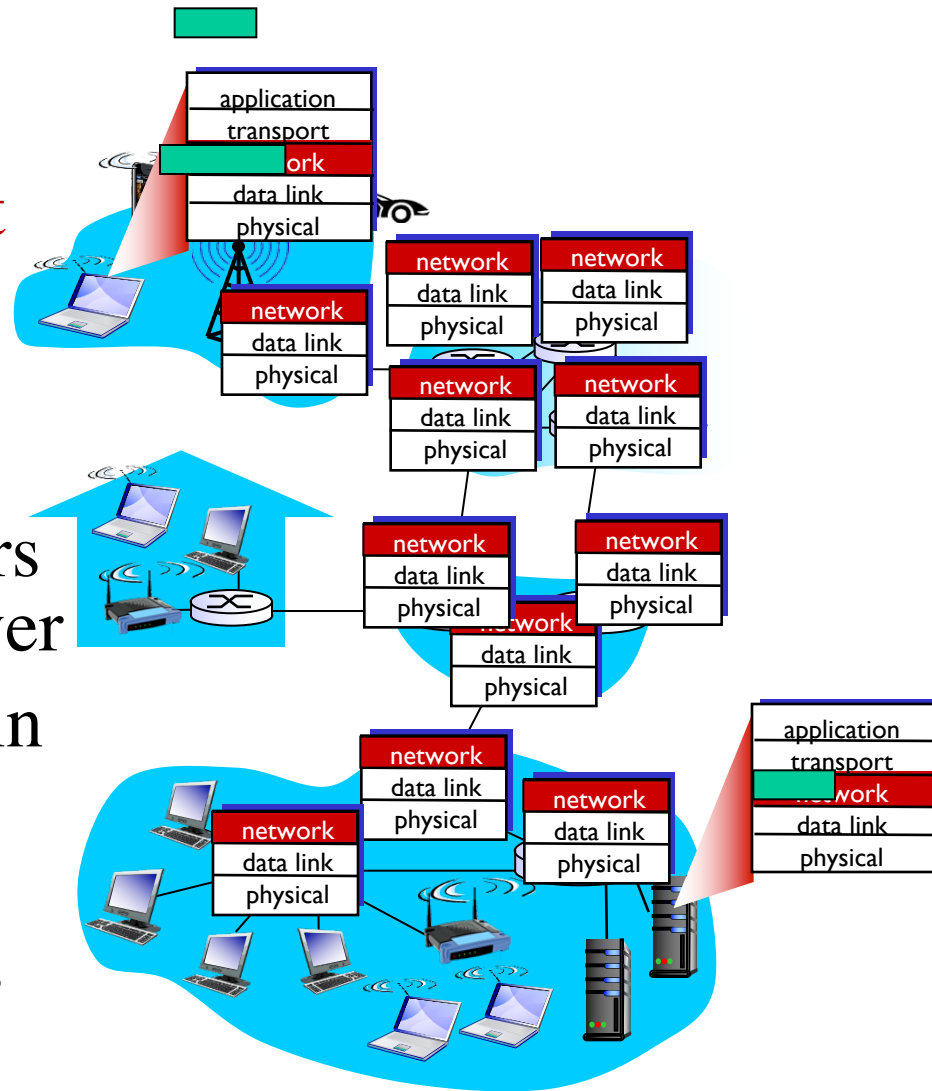
# Chapter 4: network layer

## 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

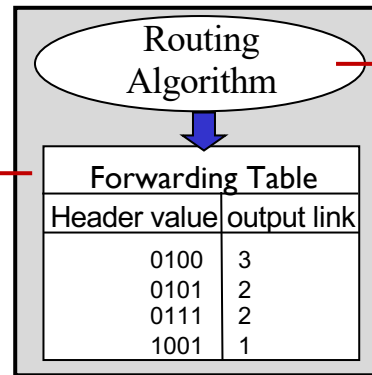
- transport segment from sending to receiving **host**
- on sending side encapsulates segments into datagrams
- on receiving side, delivers segments to transport layer
- network layer protocols in *every* host, router
- router examines header fields in all IP datagrams passing through it



# Two key network-core functions

## Forwarding:

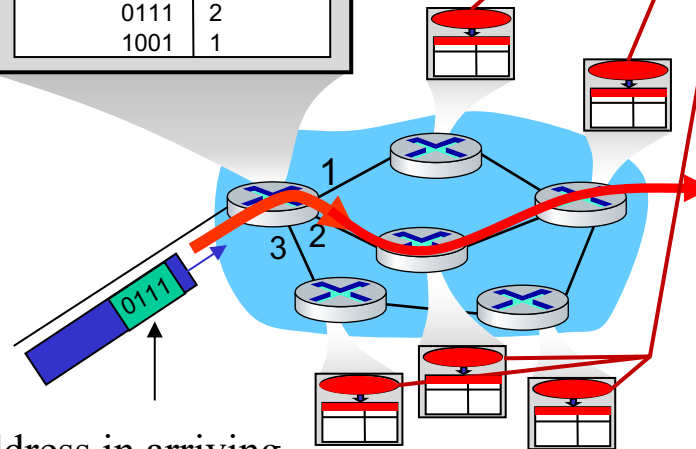
**local** action: move arriving packets from router's input link to appropriate router output link



destination address in arriving packet's header

## Routing:

- **global** action: determine source-destination paths taken by packets
- routing algorithms



# Two key network-layer functions

## Network-layer functions:

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

## Analogy: taking a trip

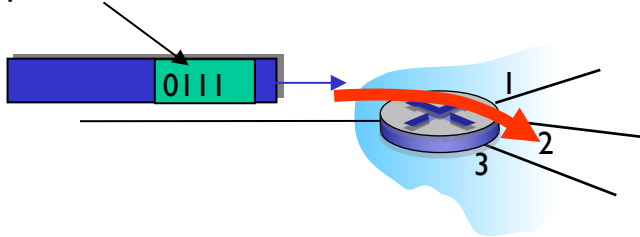
- *forwarding*: process of getting through single interchange
- *routing*: process of planning trip from source to destination

# Network layer: data plane, control plane

## *Data plane*

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

values in arriving  
packet header



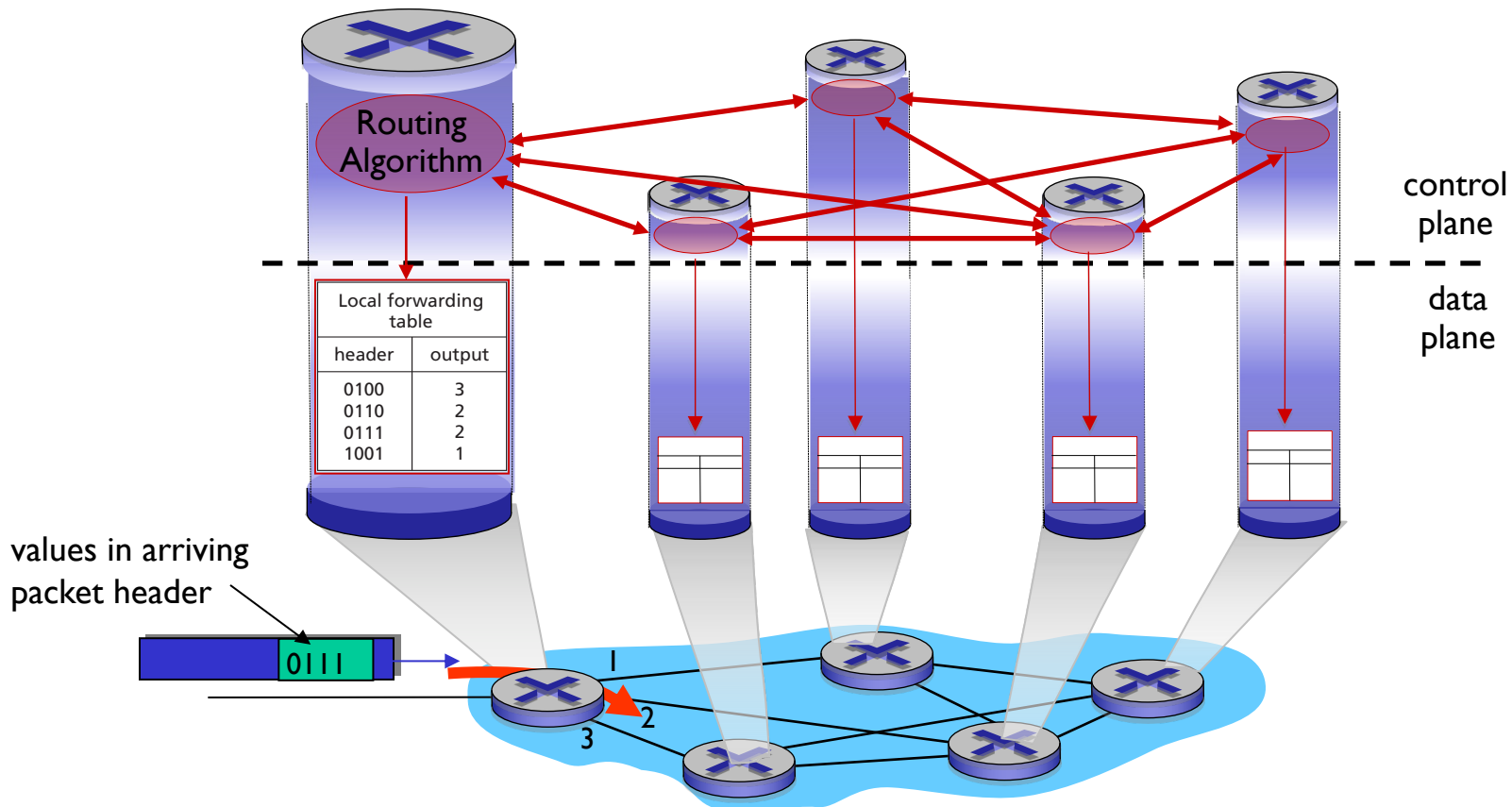
## *Control plane*

- network-wide logic, software
- 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

# Control plane: Traditional Approach

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

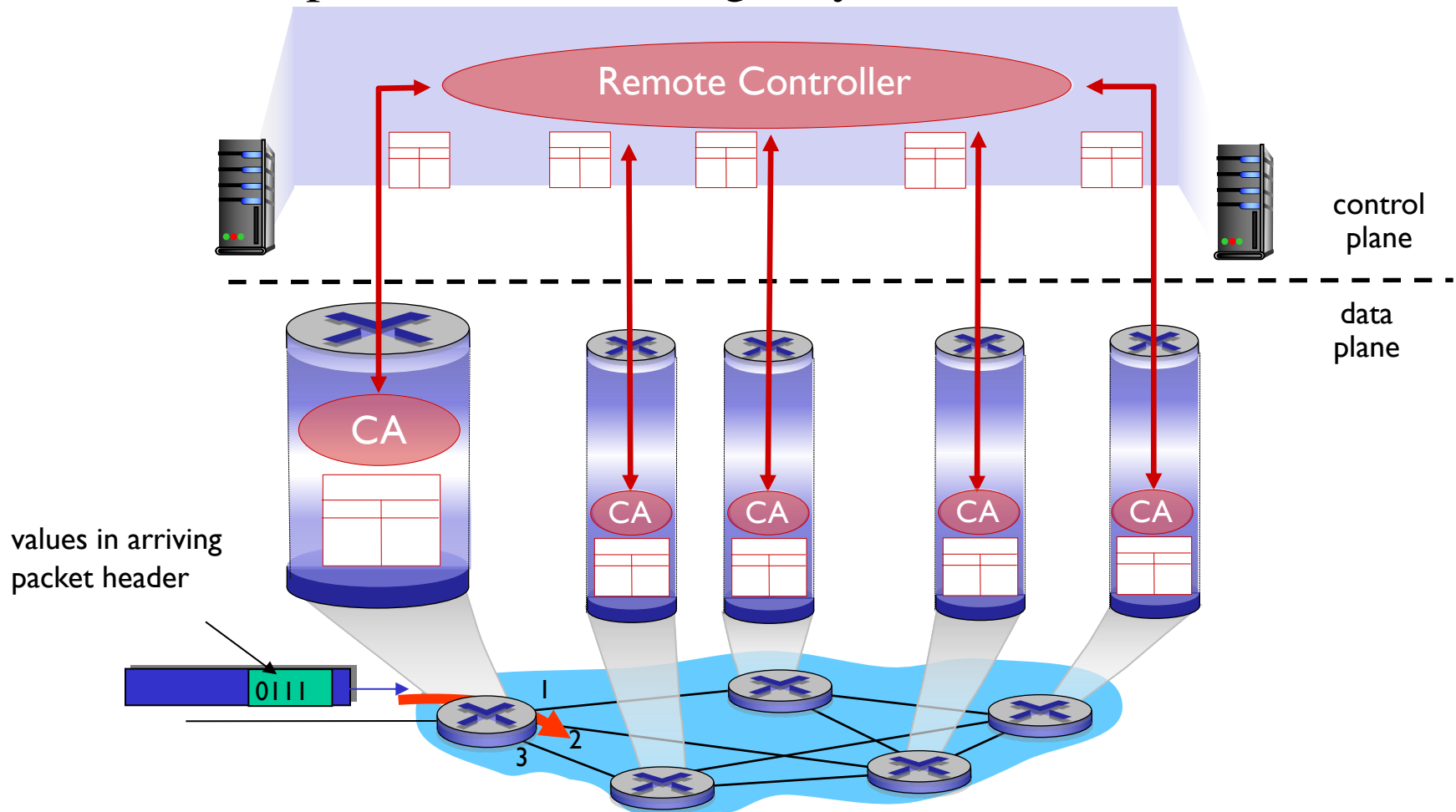
- Forwarding and routing functions are contained within a router



# Control Plane: The SDN Approach

**Logically centralized control plane:** A distinct (typically remote) controller interacts with local control agents (CAs)

- Router performs forwarding only





# 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

*other example services:* security

*Internet service model provide “best effort” service, no guarantee on bandwidth, loss, order or timing.*

# 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

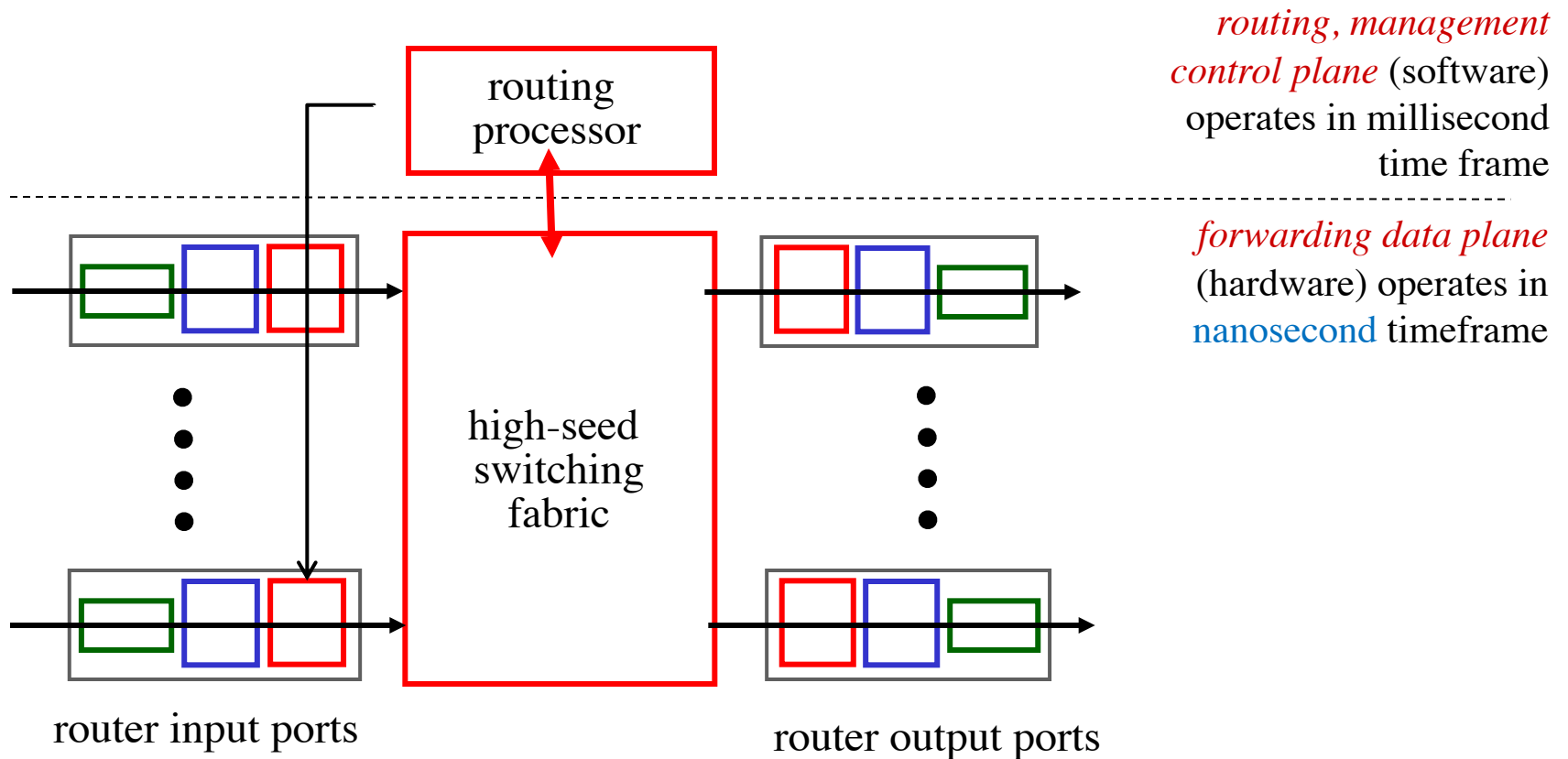
- datagram format
- fragmentation
- IPv4 addressing
- network address translation
- IPv6

## 4.4 Generalized Forward and SDN

- match
- action
- OpenFlow examples of match-plus-action in action

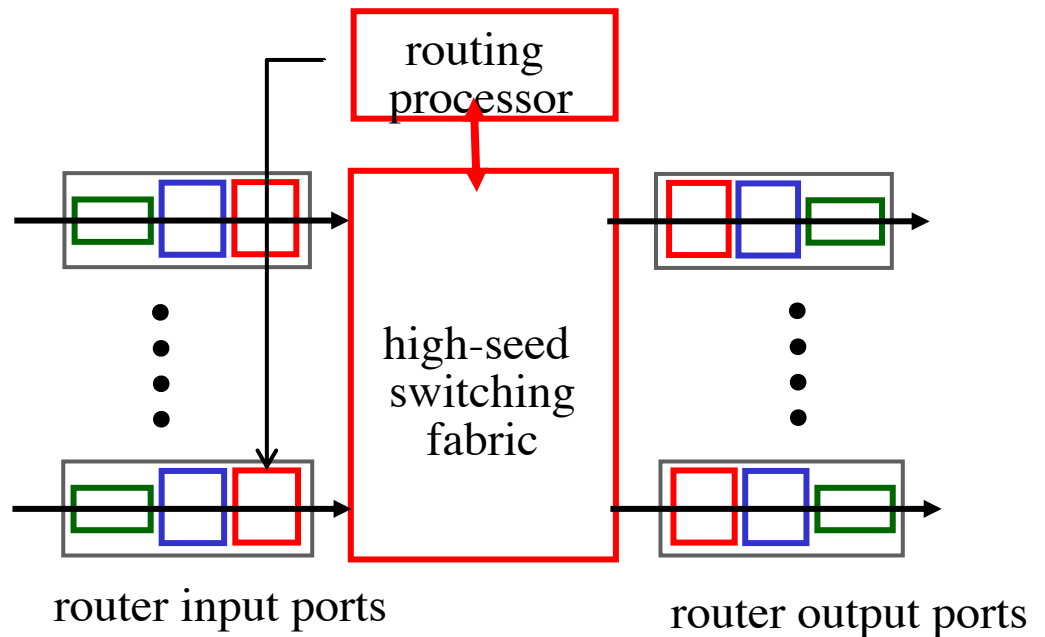
# Router architecture overview

High-level view of generic router architecture:

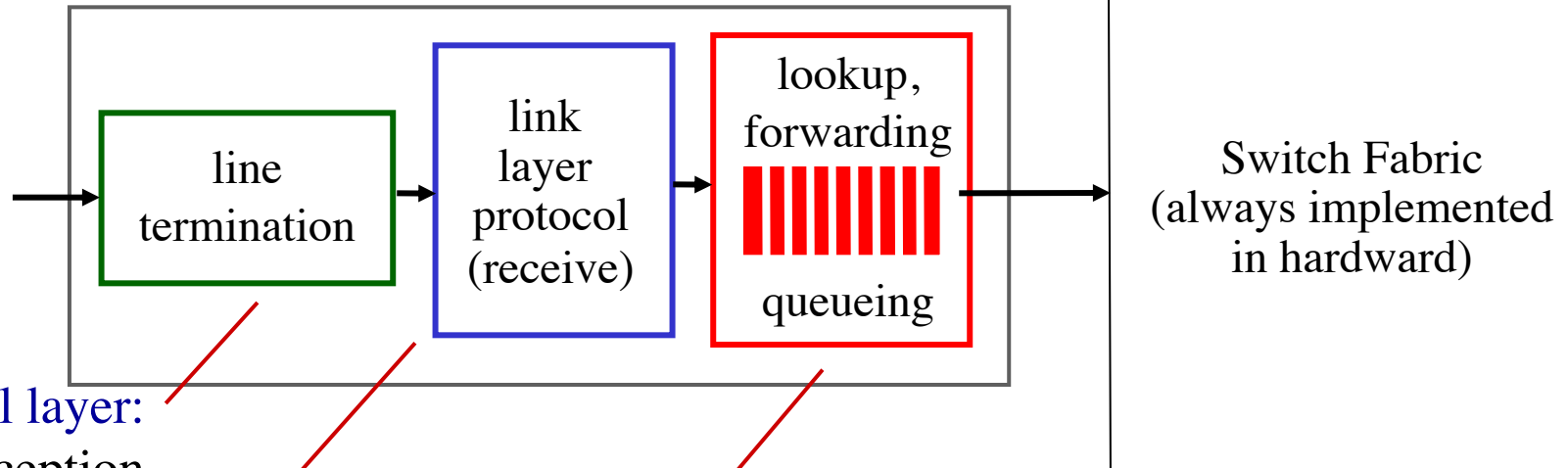


# Router Overview

- **Input port**
- Switch fabrics
  - Output port
- Queuing
  - Input port queue
  - Output port queue
  - Scheduling



# 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

# 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 (i.e., overlap between entities)?

# 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

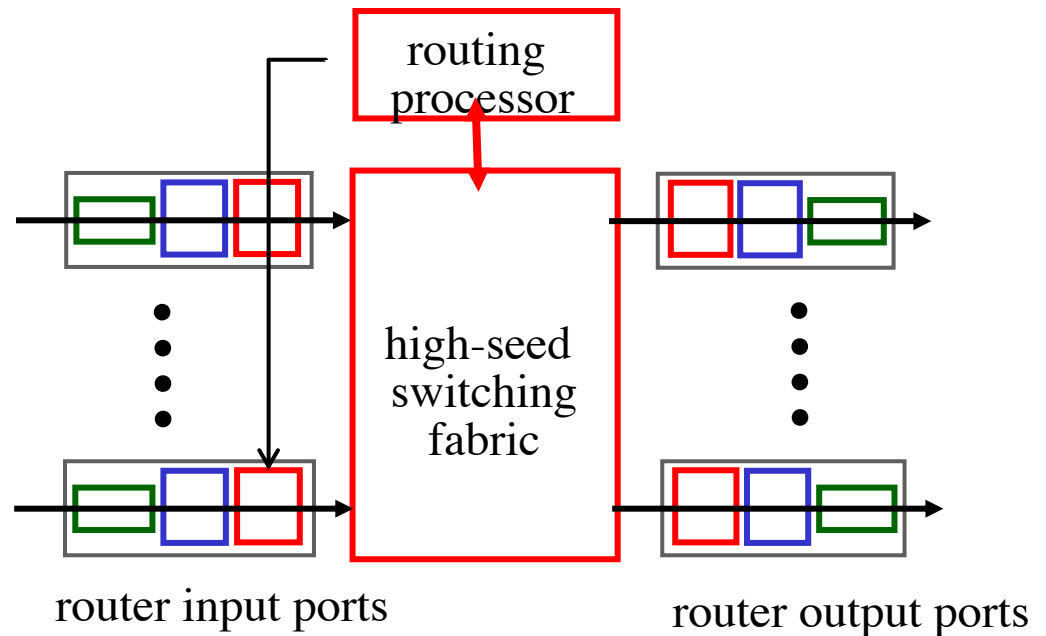
which interface?

DA: 11001000 00010111 00011000 10101010

which interface?

# Router Overview

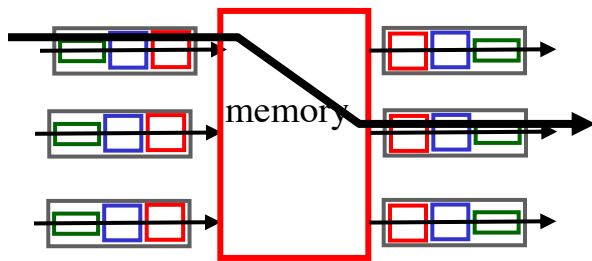
- Input port
- **Switching fabrics**
- Output port
- Queuing
  - Input port queue
  - Output port queue
  - Scheduling





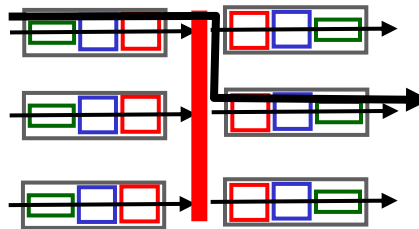
# Switching fabrics

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



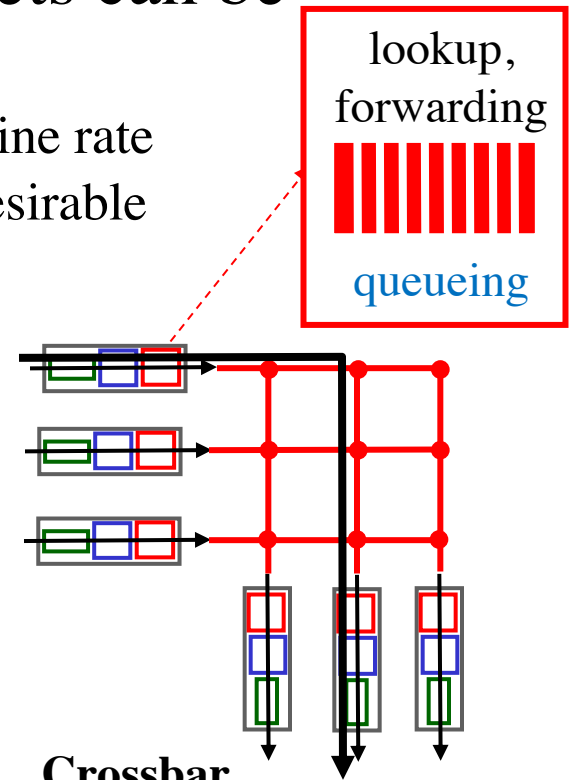
## Switch via memory

- Interrupt; write and read
- Two packets cannot be forwarded at the same time



## Switch via bus

- Broadcast; label
- One packet can cross at a time

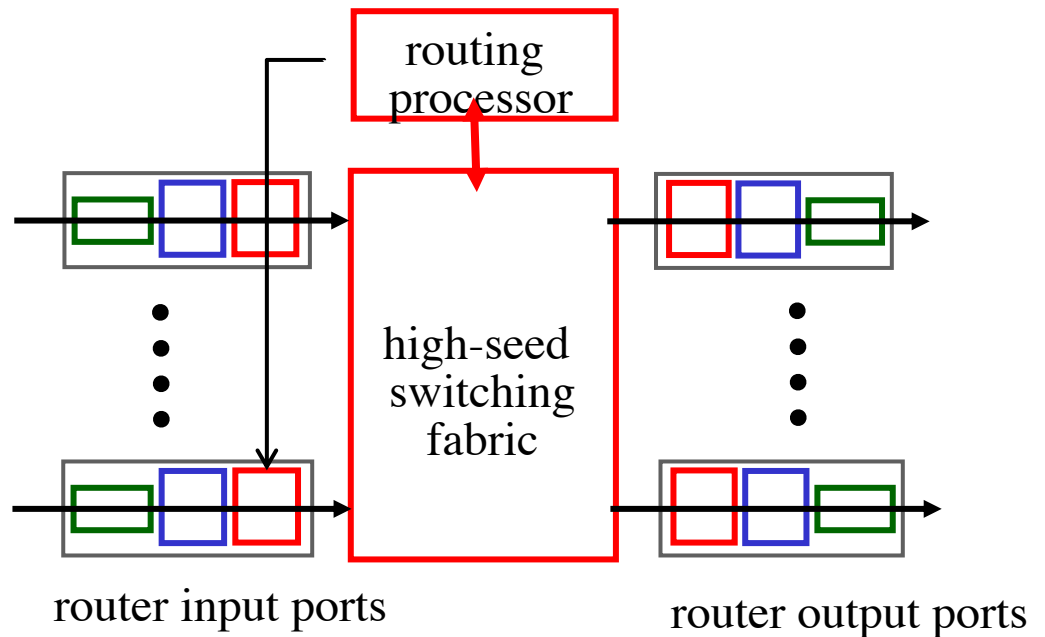


## Crossbar

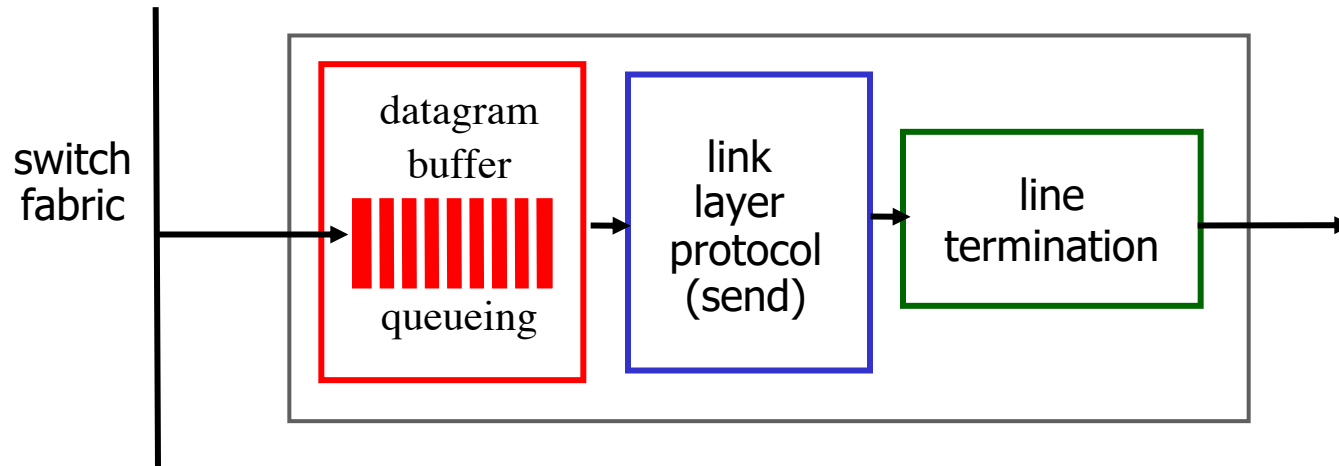
- Multiple packets in parallel
- Non-blocking

# Router Overview

- Input port
- Switch fabrics
- **Output port**
- Queuing
  - Input port queue
  - Output port queue
  - Scheduling



# Output ports



- *buffering* required when datagrams arrive from fabric faster than the transmission rate

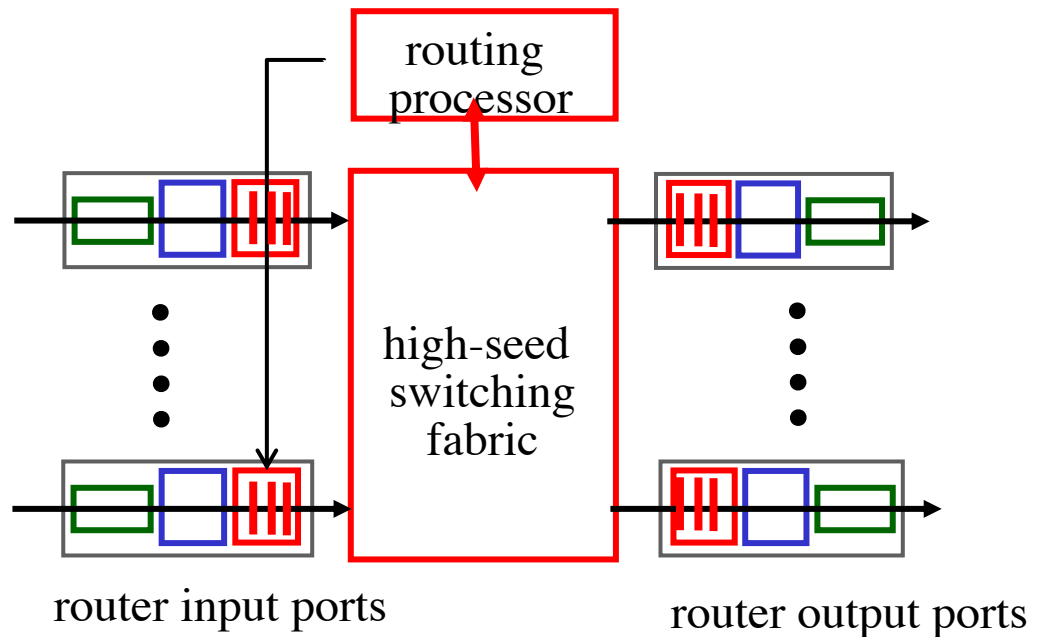
Datagram (packets) can be lost due to congestion, lack of buffers

- *scheduling discipline* chooses among queued datagrams for transmission

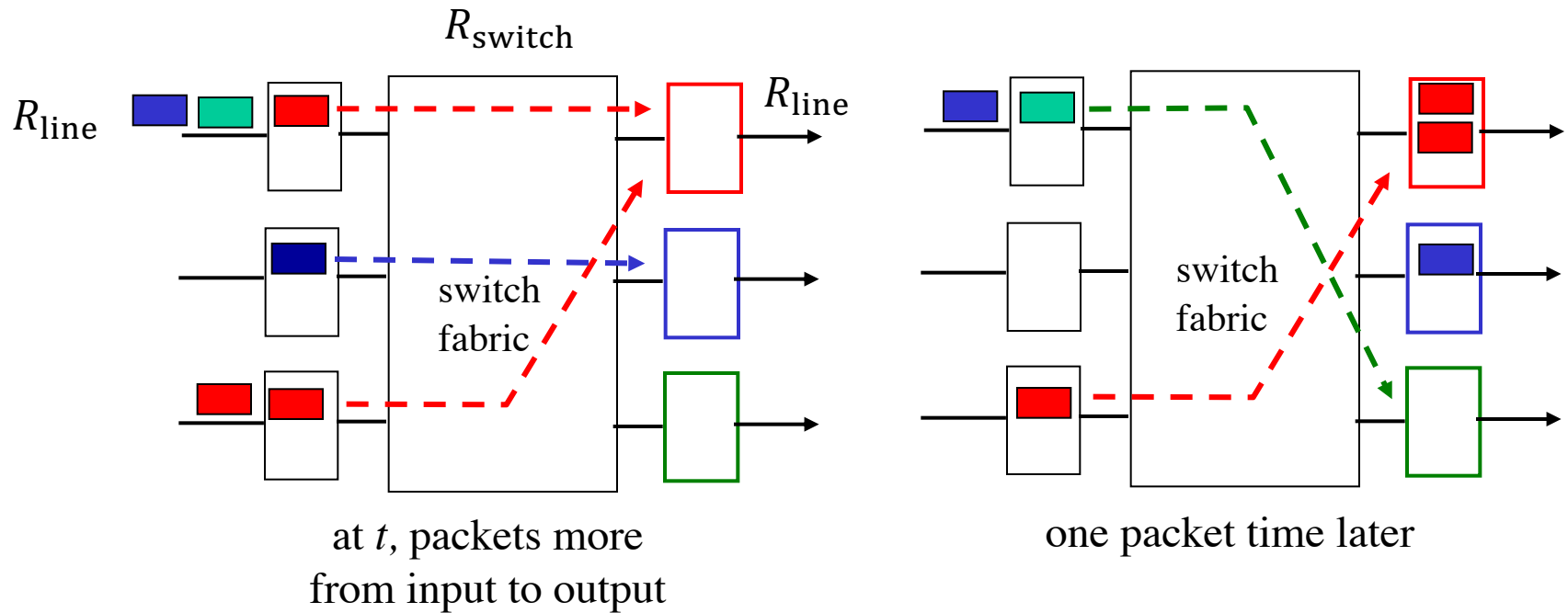
Priority scheduling – who gets best performance, network neutrality

# Router Overview

- ❖ Input port
- ❖ Switch fabrics
- ❖ Output port
- ❖ **Queuing**
  - Input port queue
  - Output port queue
  - Queue scheduling



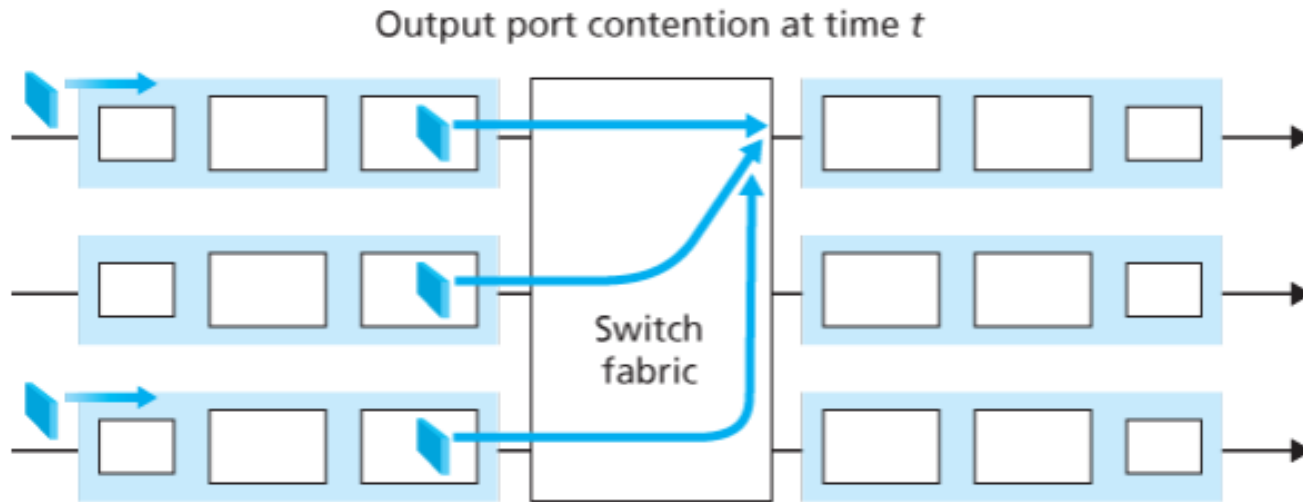
# Input port queueing



Switch fabric is not fast enough (e.g., suppose  $R_{\text{switch}} = R_{\text{line}}$ )

- Packet queuing occur at **input port**
- Crossbar, and **multiple packets** must be transferred to the **same port**

# Output port queueing



- If  $R_{\text{switch}}$  is  $N$  times faster than  $R_{\text{line}}$ , then negligible queuing at the input ports.
- buffering when arrival rate via switch exceeds output line speed
- *queueing (delay) and loss due to output port buffer overflow!*

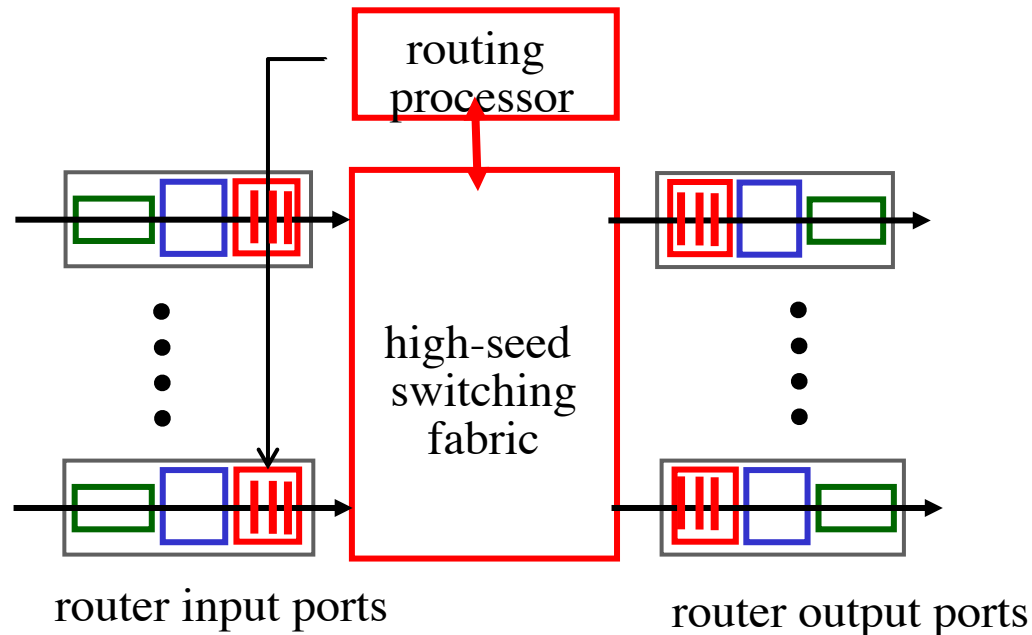
# How much buffering?

- 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

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

# Scheduling mechanisms

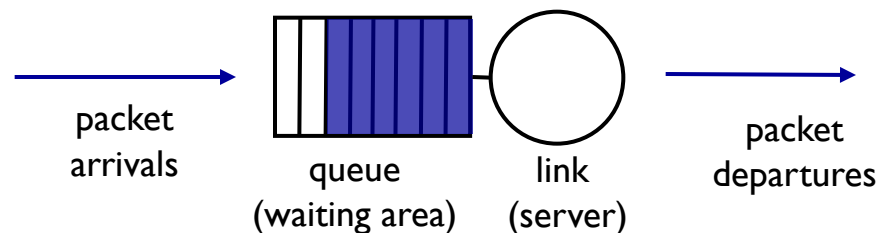
- *scheduling*: choose next packet to send on link





# Scheduling policies: FIFO

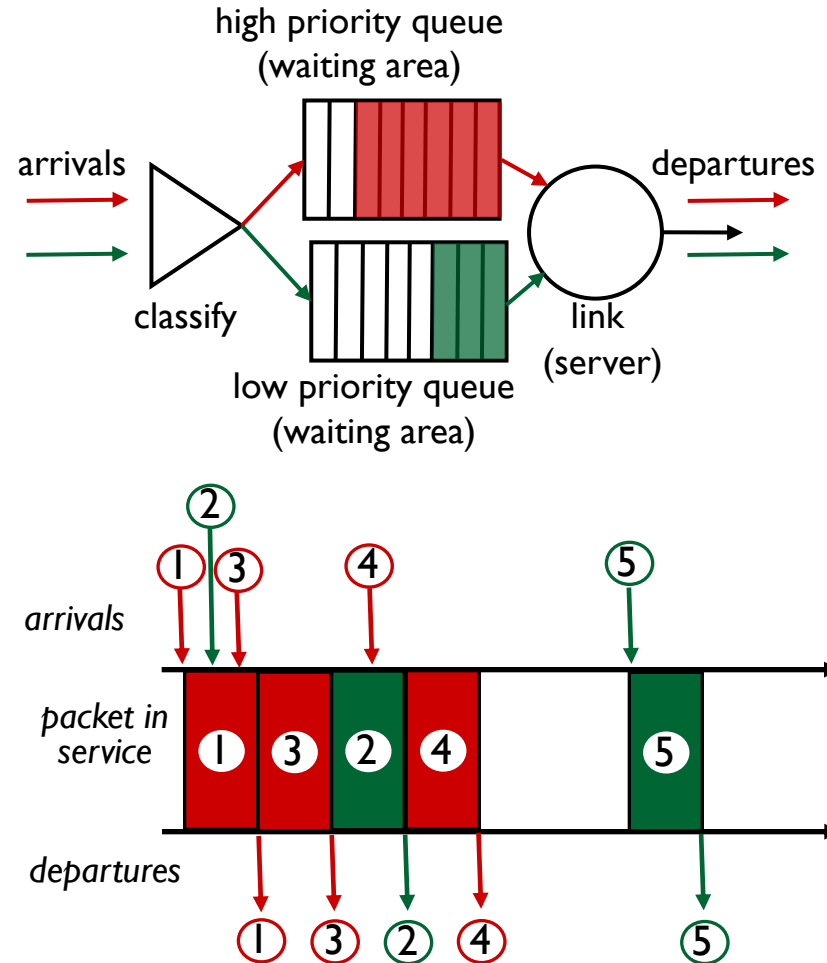
- *FIFO (first in first out) queuing*: send in order of arrival to queue
- *discard policy*: if packet arrives to full queue: who to discard?
  - *tail drop*: drop arriving packet
  - *priority*: drop/remove on priority basis
  - *random*: drop/remove randomly



# Scheduling policies: priority

*priority queuing*: send highest priority queued packet

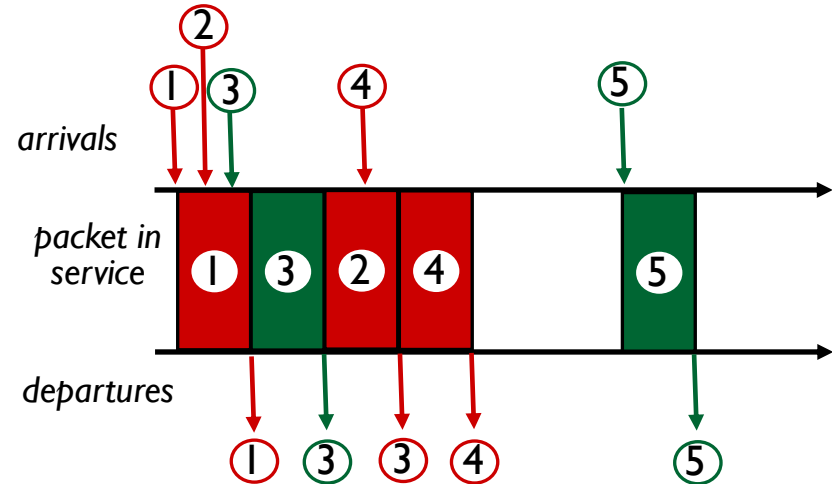
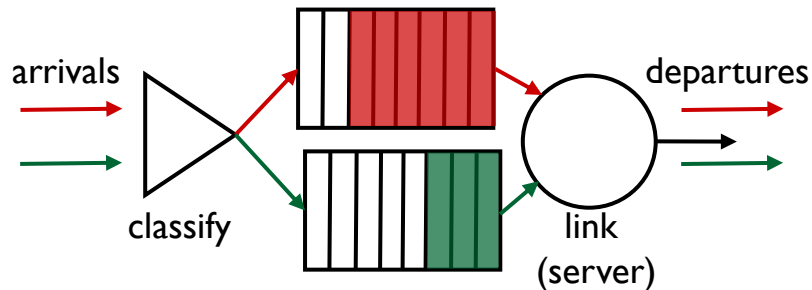
- multiple *classes*, with different priorities
  - class may depend on marking or other header info, e.g. IP source/dest, port numbers, etc.
- Transmit a packet from the highest priority class
- **Non-preemptive** priority queuing



# Scheduling policies: still more

## *Round Robin (RR) scheduling:*

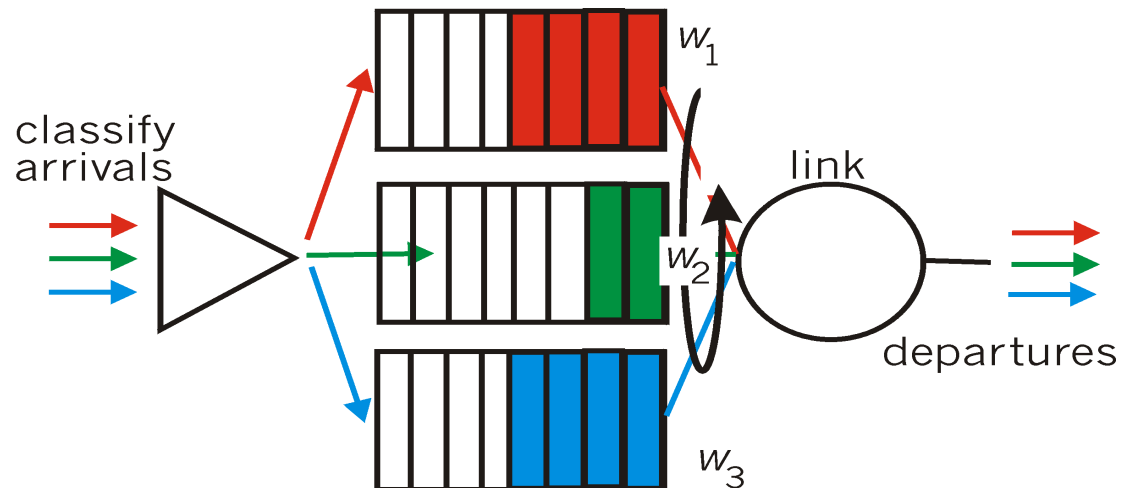
- multiple classes
- cyclically scan class queues, sending one complete packet from each class (if available)



# Scheduling policies

## *Weighted Fair Queuing (WFQ):*

- generalized Round Robin
- each class gets weighted amount of service in each cycle
  - $w_i / \sum_{j \in Q^{busy}} w_j$  of the bandwidth (throughput)
  - $Q^{busy}$ : all classes that have queued packets
  - Worst case: all queues have packets;  $w_i / \sum_{j \in Q} w_j$



# 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

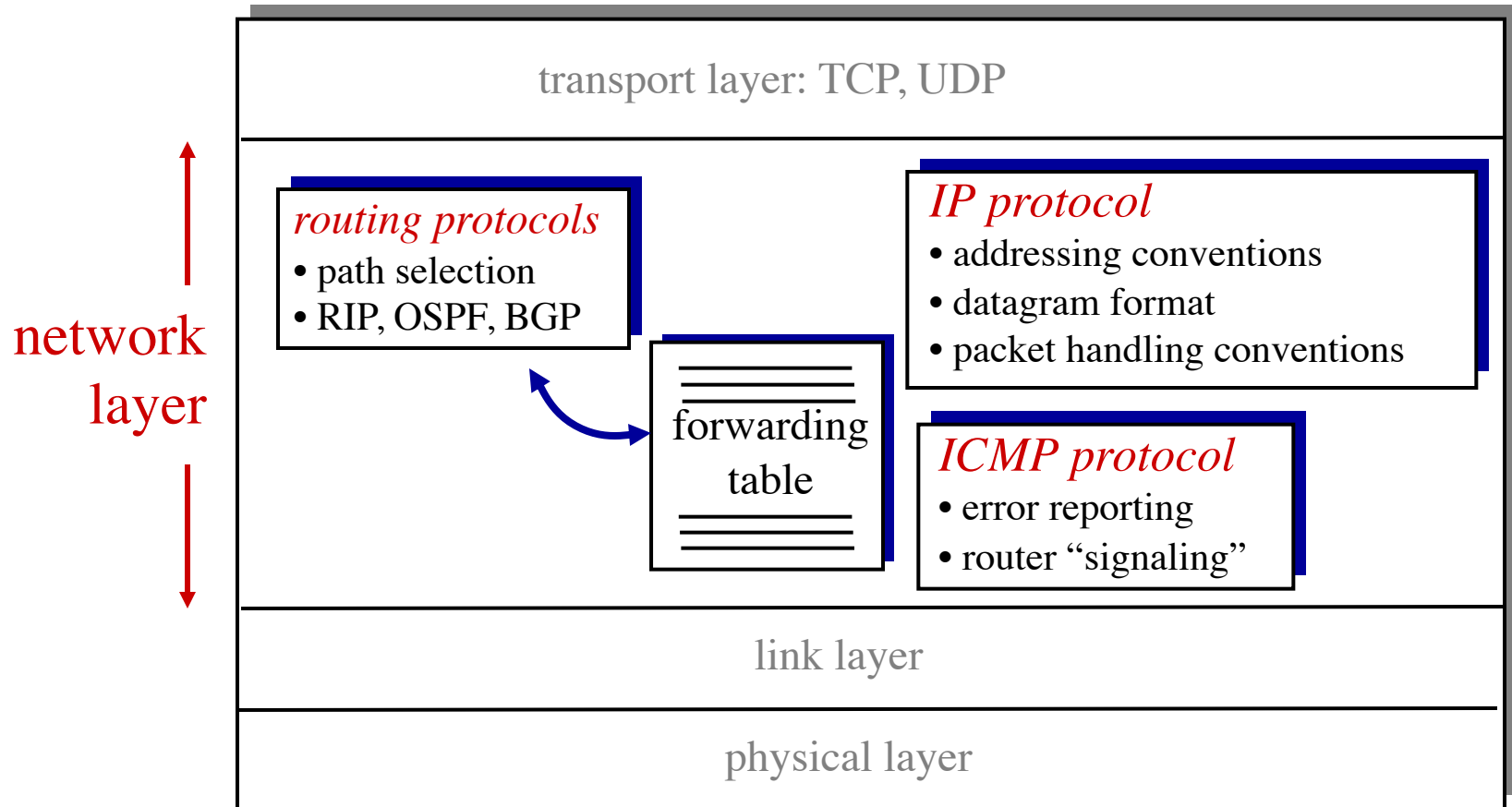
- datagram format
- fragmentation
- IPv4 addressing
- network address translation
- IPv6

## 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

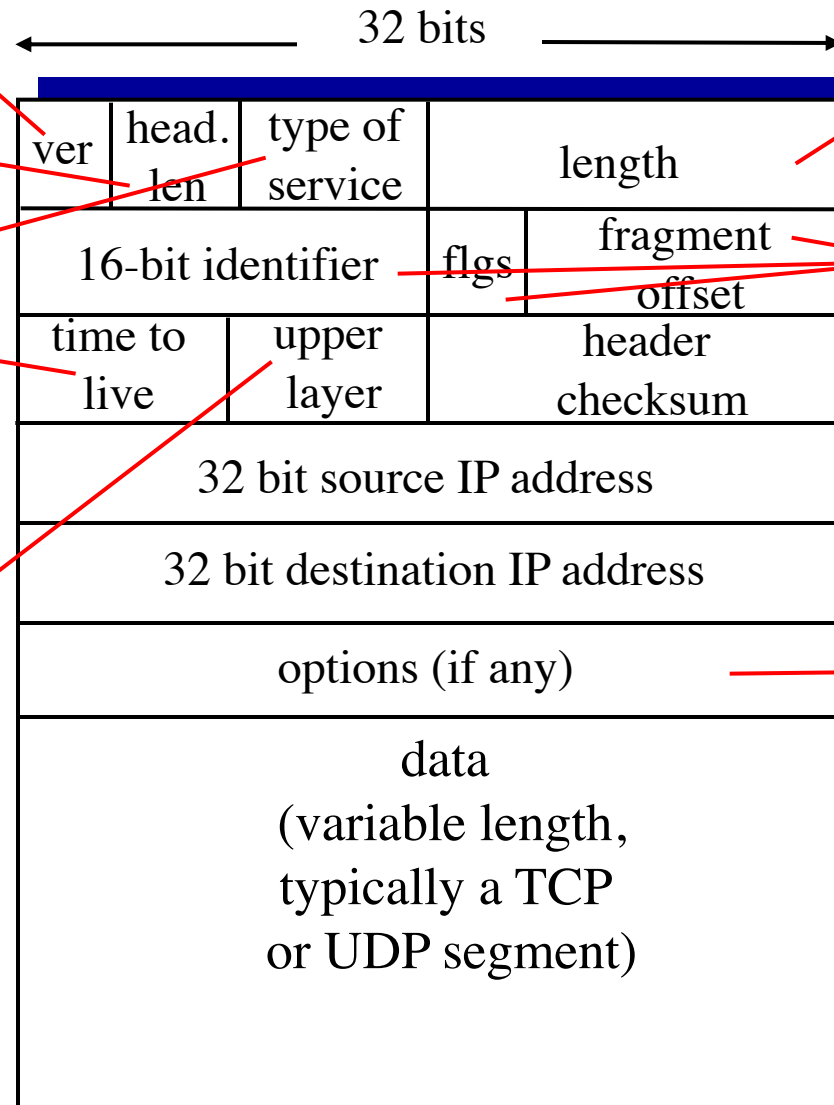
IP protocol version  
number

header length  
(bytes)

“type” of data

max number  
remaining hops  
(decremented at  
each router)

upper layer protocol  
to deliver payload to



total datagram  
length (bytes)

for  
fragmentation/  
reassembly

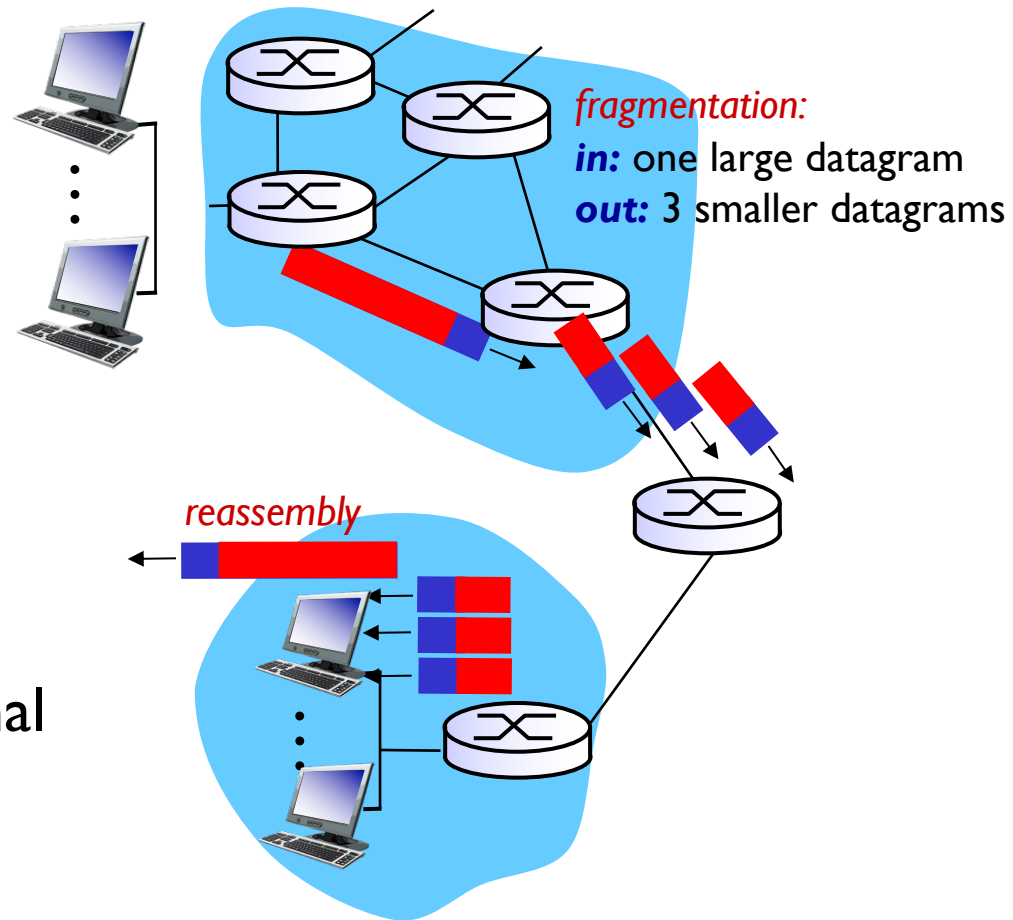
e.g. timestamp,  
record route  
taken, specify  
list of routers  
to visit.

*how much overhead?*

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

# IP fragmentation, reassembly

- network links have max transmission unit (MTU) - 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

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

*one large datagram becomes  
several smaller datagrams*

1480 bytes in  
data field

offset =  
1480/8

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

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

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

# 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

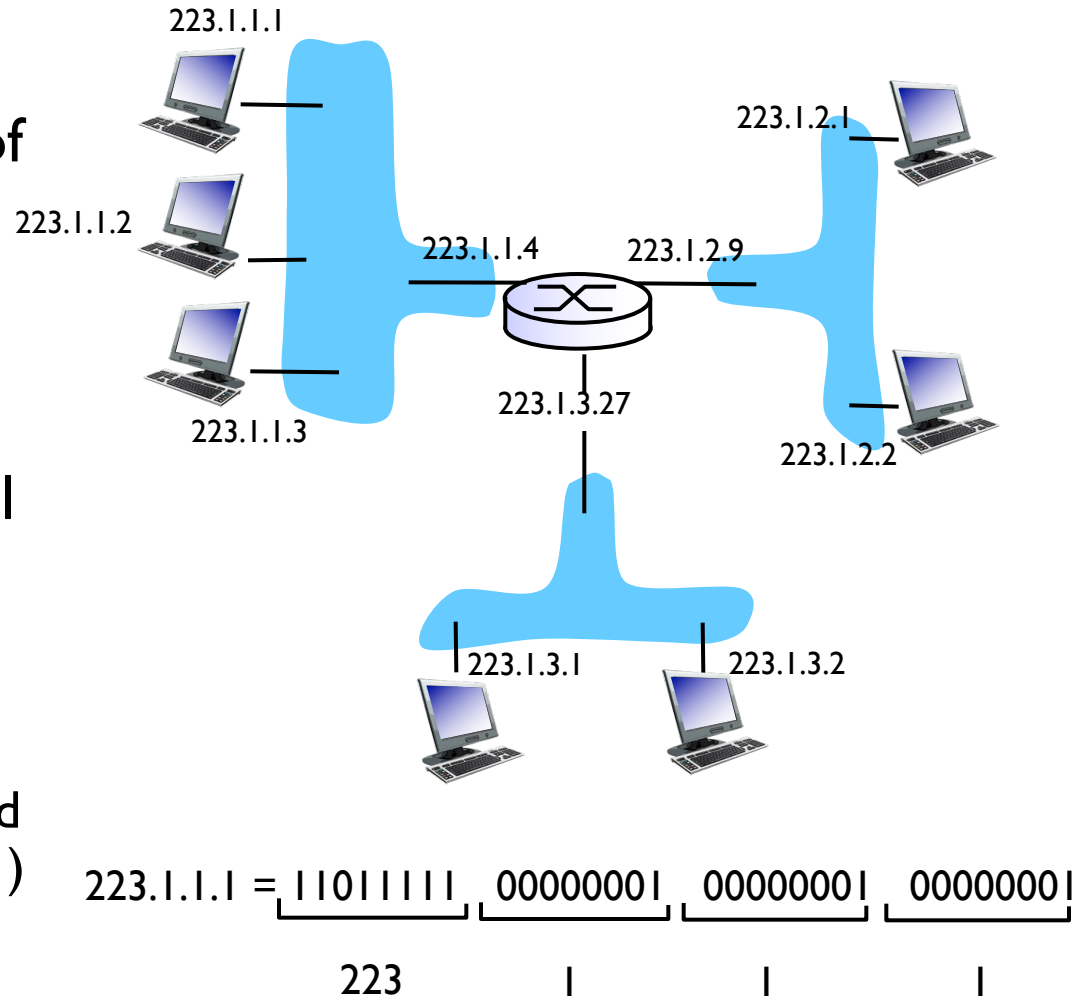
- match
- action
- OpenFlow examples of match-plus-action in action

# Overview

- IP addressing
- Subnet
- How to assign/obtain IP address?

# IP addressing: introduction

- **IP address:** 32-bit identifier for interface of hosts and routers
- **interface:** (network interface card) 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**



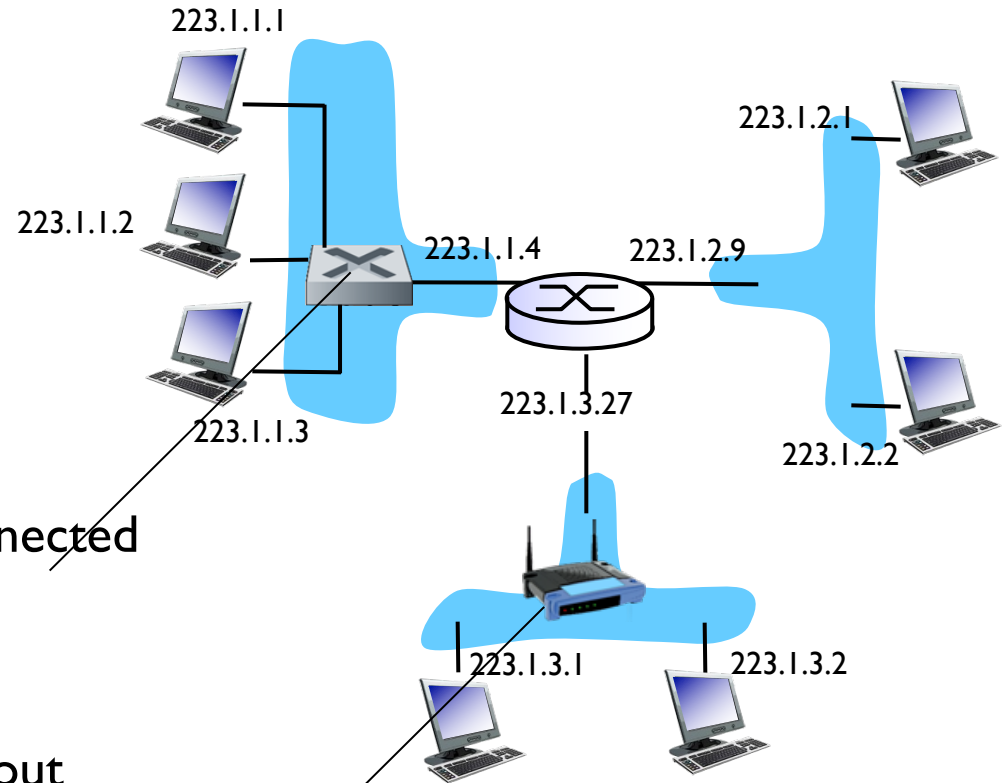
# IP addressing: introduction

**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

# Overview

- IP addressing
- Subnet
- How to assign/obtain IP address?

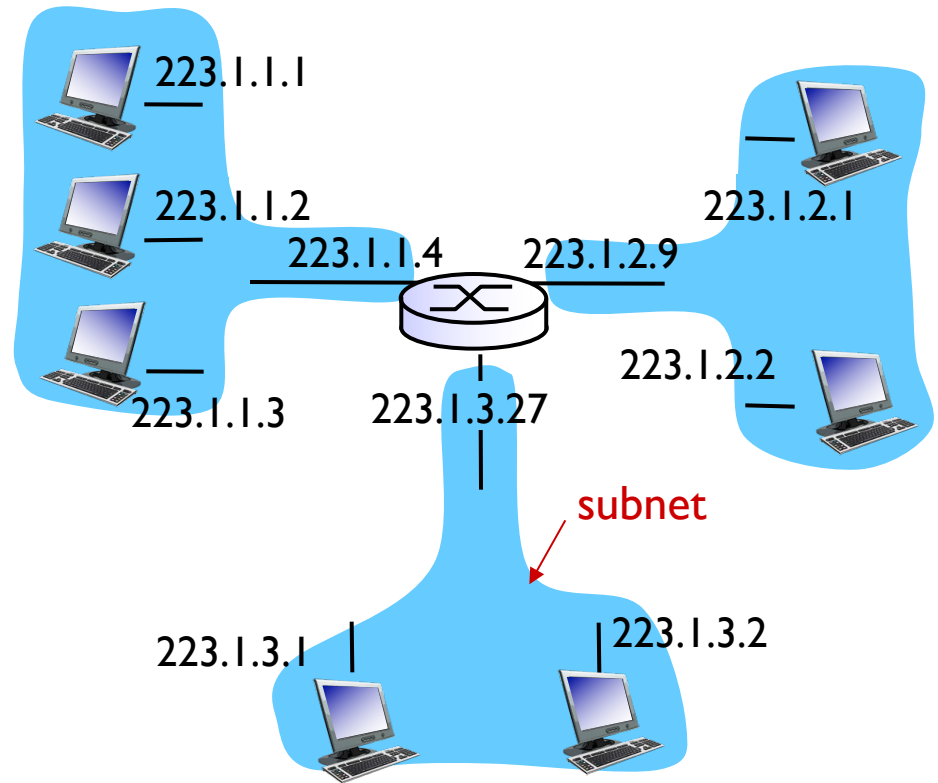
# Subnets

## ■ 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*

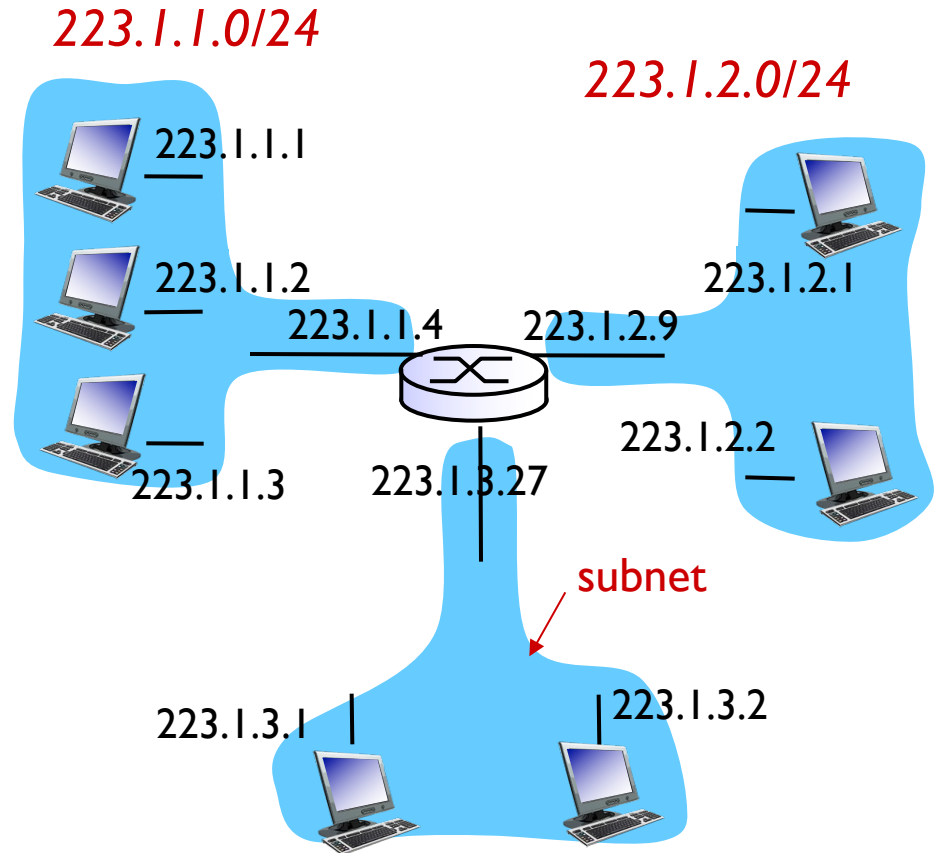


network consisting of 3 subnets

# 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*

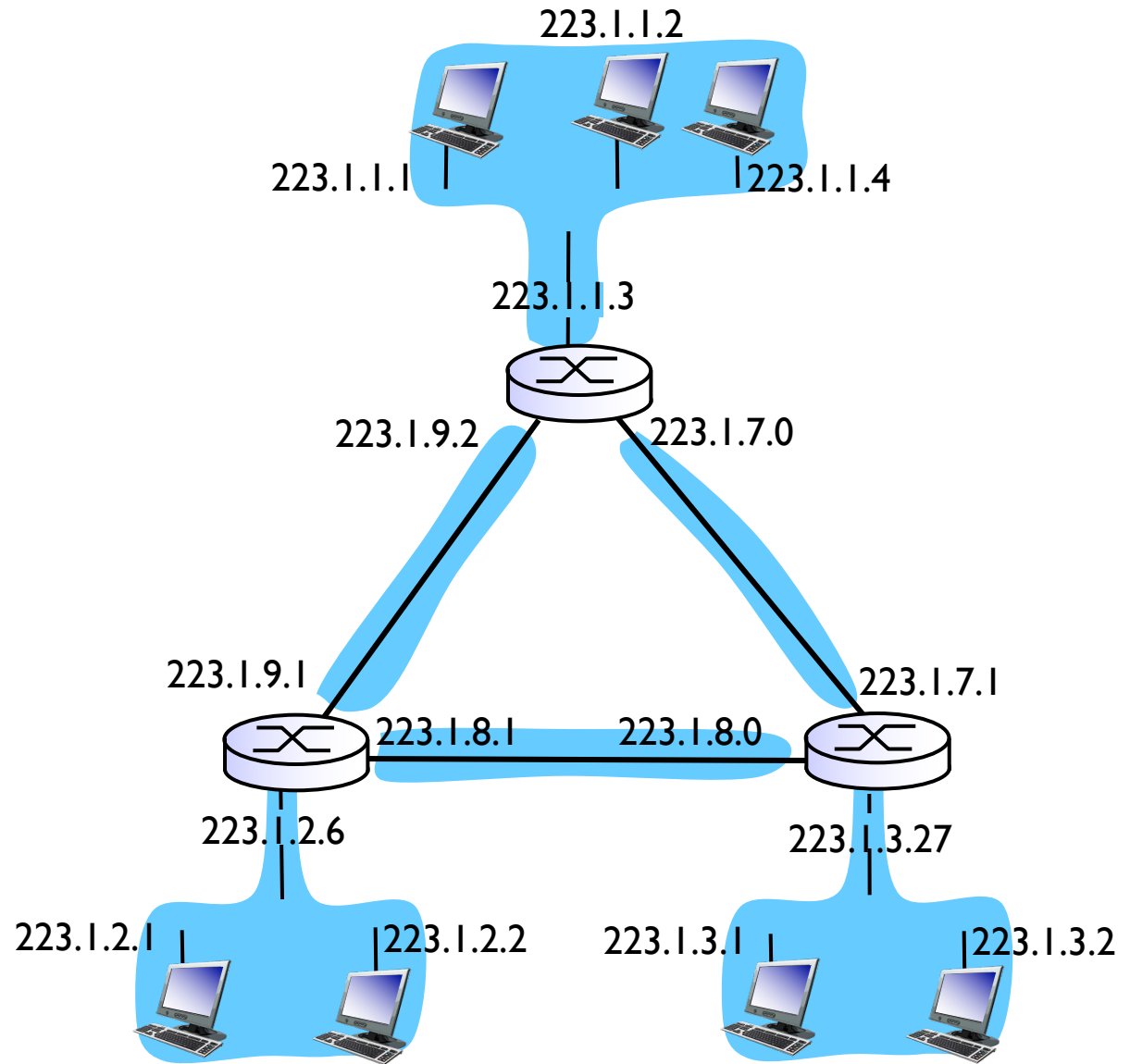


subnet mask: /24  
255.255.255.0



# Subnets

how many?



# Overview

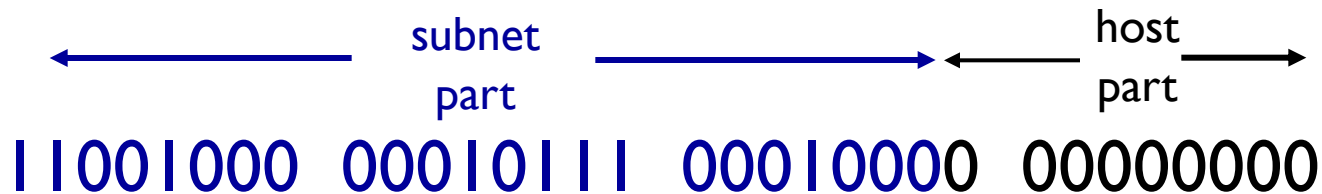
---

- IP addressing
- Subnet
- How to assign/obtain IP address?

# IP addressing: CIDR

## CIDR: Classless InterDomain Routing

- A method to assign blocks of IP address
- 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

Subnet mask: 255.255.254.0

# How does an ISP get block of addresses?

**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

# How does a subnet get block of addresses?

**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 00010111 00010000</u> 00000000	200.23.16.0/20
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

# How does a subnet get block of addresses?

Suppose all of the interfaces in each of these three subnets are required to have the prefix 166.4.20.128/25.

- Subnet 1 is required to support at least 62 interfaces
- Subnet 2 is required to support at least 30 interfaces
- Subnet 3 is required to support at least 28 interfaces

Provide three network addresses (of the form a.b.c.d/x) that satisfy these constraints.

# How does a subnet get block of addresses?

Block of addresses: 166.4.20.128/25. Subnet 1: at least 62 interfaces; Subnet 2: at least 30 interfaces; Subnet 3: at least 28 interfaces

This block of IP addresses can be written as

10100110 00000100 00010100 10000000

Subnet 1: since  $2^6 = 64 > 62$ , we can assign the following block

10100110 00000100 00010100 10000000

which can be represented as 166.4.20.128/26.

Subnet 2: since  $2^5 = 32 > 30$ , we can assign the following block

10100110 00000100 00010100 11000000

which can be represented as 166.4.20.192/27.

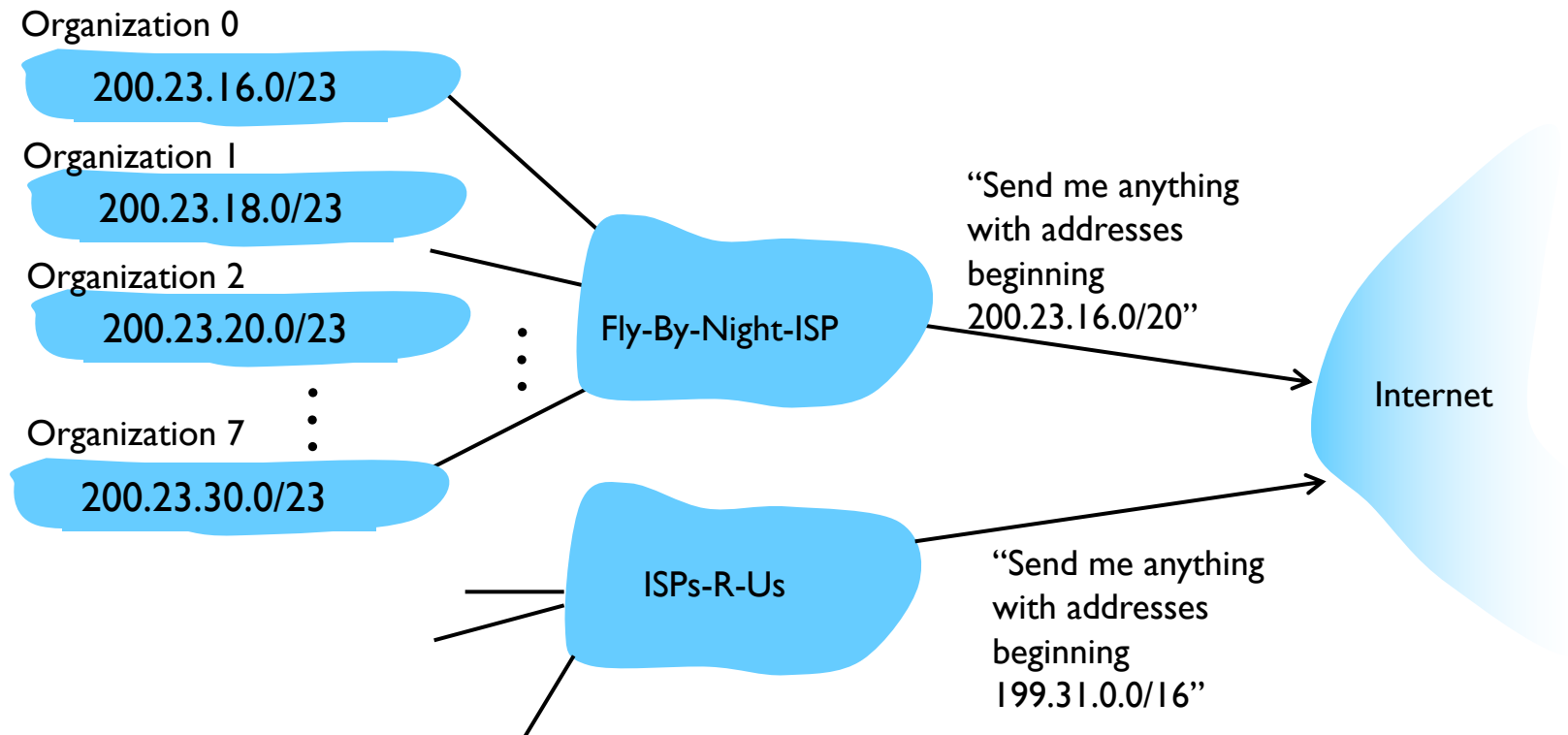
Subnet 3: since  $2^5 = 32 > 28$ , we can assign the following block

10100110 00000100 00010100 11100000

which can be represented as 166.4.20.224/27.

# Hierarchical addressing: route aggregation

hierarchical addressing allows efficient advertisement of routing information:



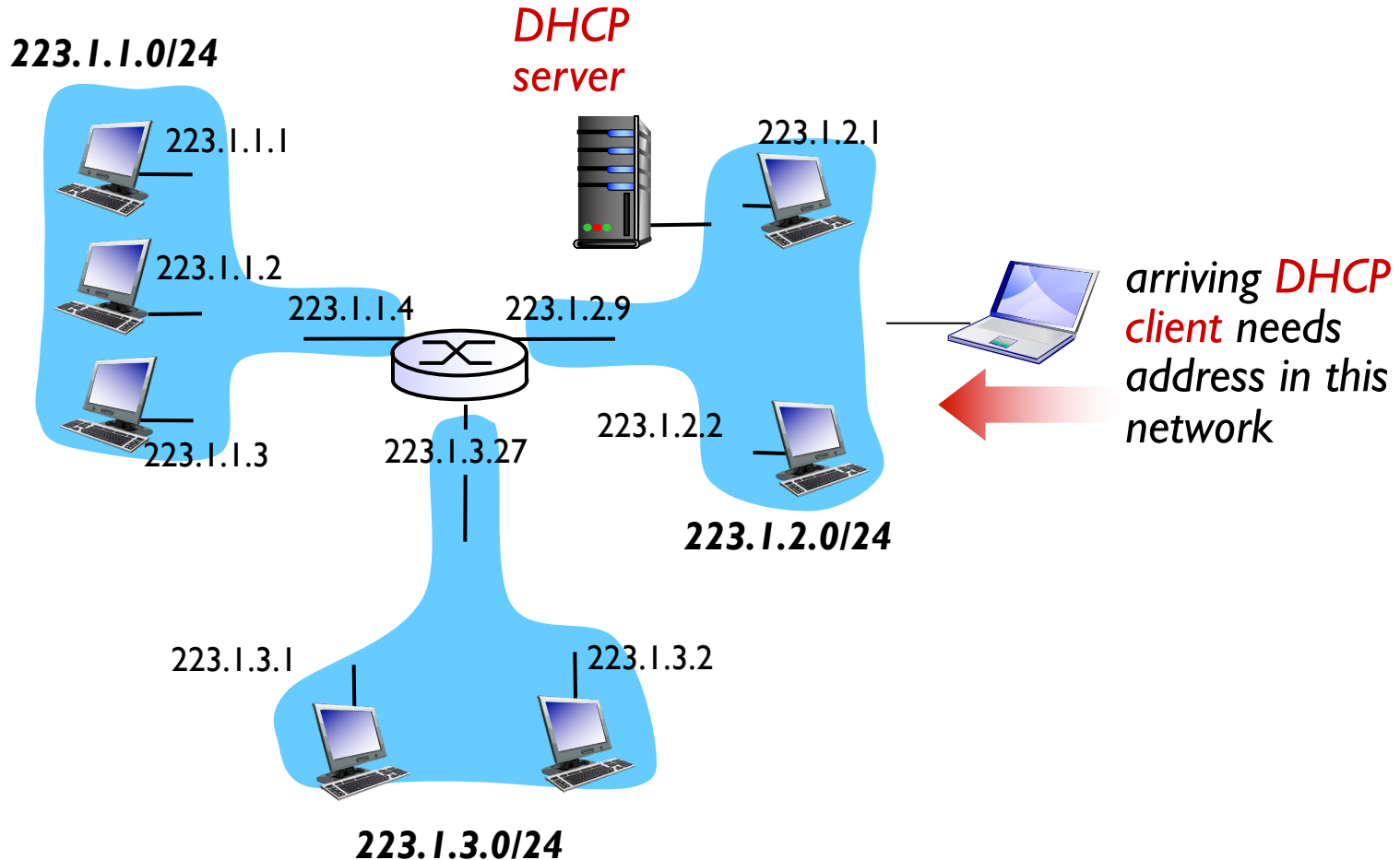


# How does a host get an IP?

**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 client-server scenario



# 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
- DHCP server responds with “DHCP offer” msg
- host requests IP address: “DHCP request” msg
- DHCP server sends address: “DHCP ack” msg

# 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'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)