# Networks (2IRR20)

## Network Layer – Data Plane (04)


Dr. Tanir Ozcelebi

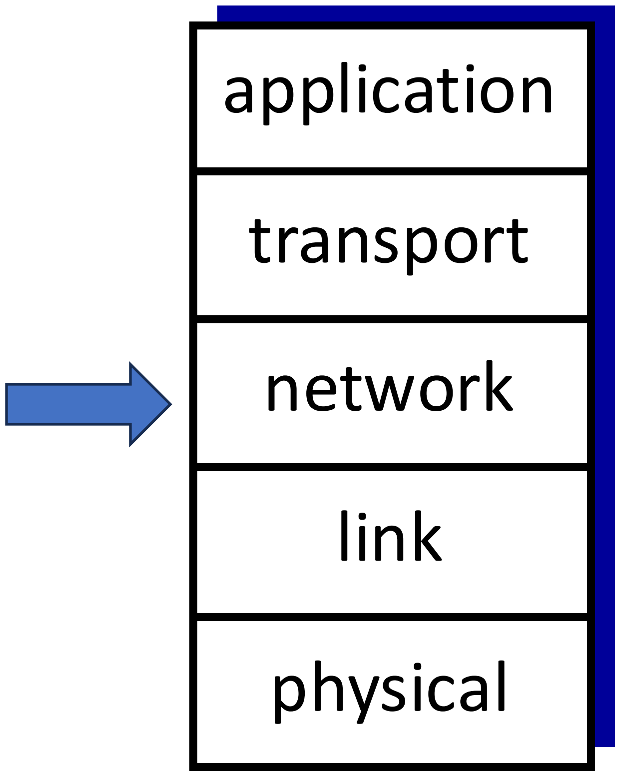
**TU/e Computer Science**

**Interconnected Resource-aware Intelligent Systems (IRIS)**


*Slides adapted from the textbook:*
***Computer Networking: A Top-Down Approach***
8th edition, Global Edition by Jim Kurose, Keith Ross

# This slide set

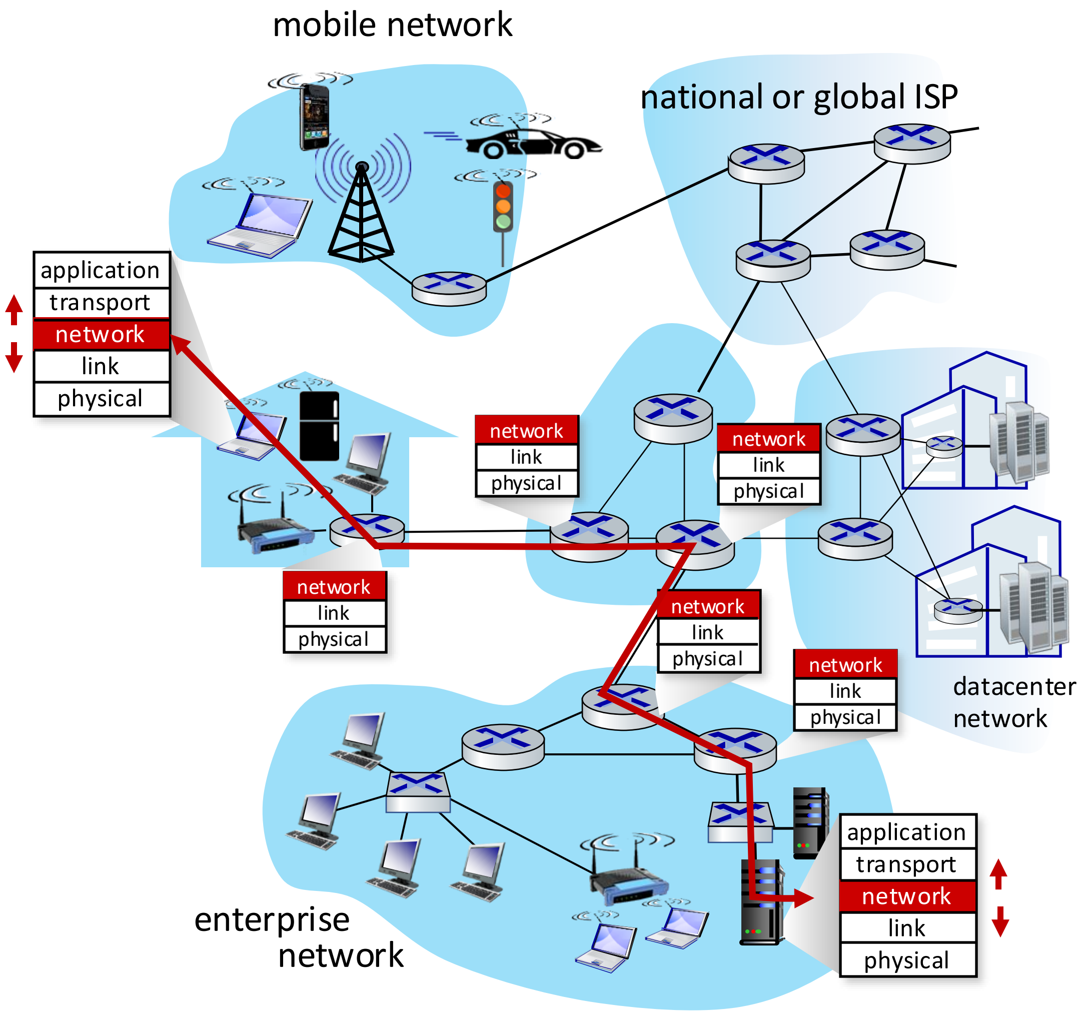
- Network layer key functions
- Data plane: Inside a router (forwarding)
- Internet Protocol (IP)



# Network layer key functions

# Router

- examines header fields in all IP datagrams passing through it
- moves each datagram in direction of destinations (to the corresponding output port)



# Two key network-layer functions

## network-layer functions:

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

## analogy: taking a trip

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

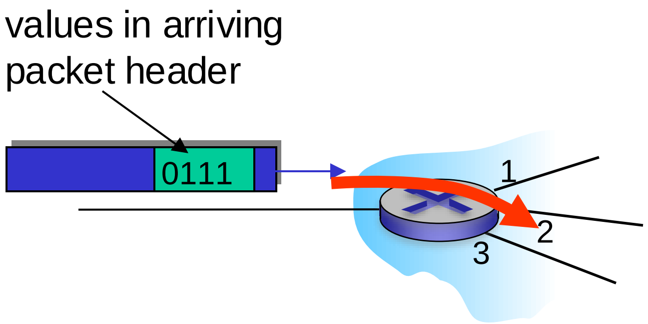
forwarding

routing

# Network layer: data plane, control plane

## Data plane:

- *local* logic
- determines per-router forwarding behavior



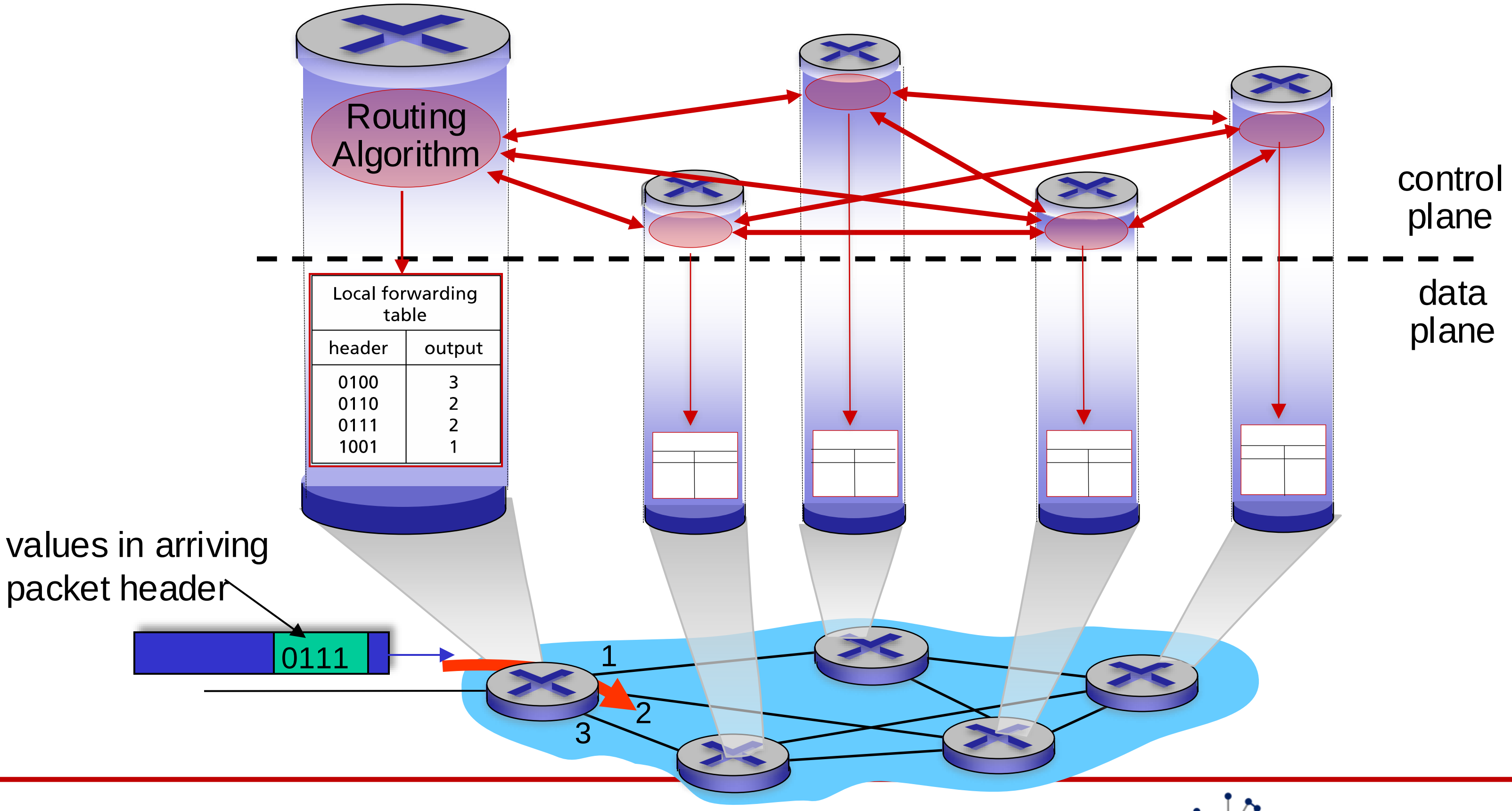

## Control plane

- *network-wide* logic
- determines how datagram is routed end-to-end from source host to destination host

- 2 control-plane approaches:
  - *traditional routing algorithms:* implemented in routers
  - *software-defined networking (SDN)*: implemented in (remote) servers

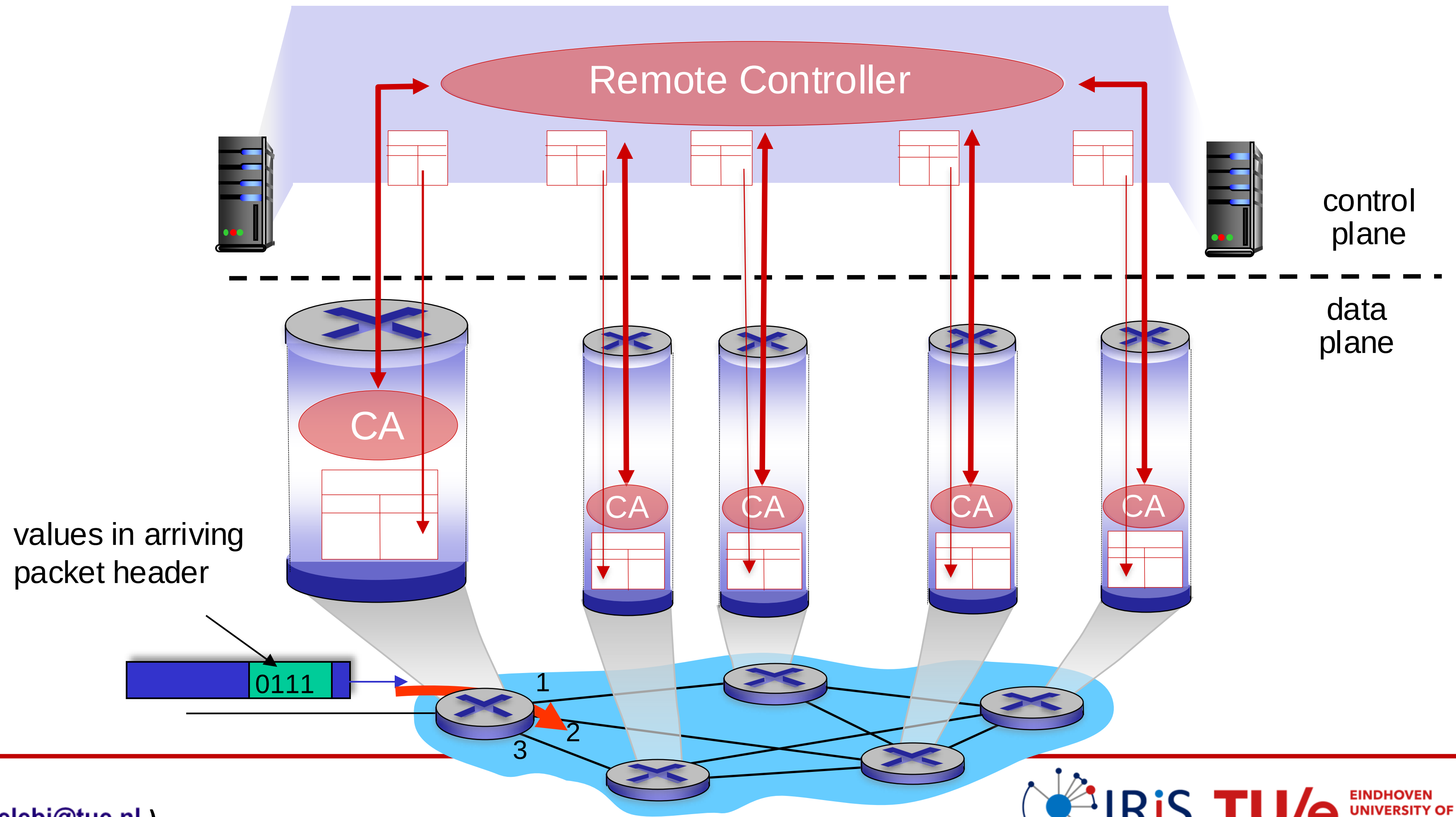
# Per-router control plane

Components of a distributed routing algorithm (*in each and every router*) interact/coordinate in the control plane.



# Software-Defined Networking (SDN) control plane

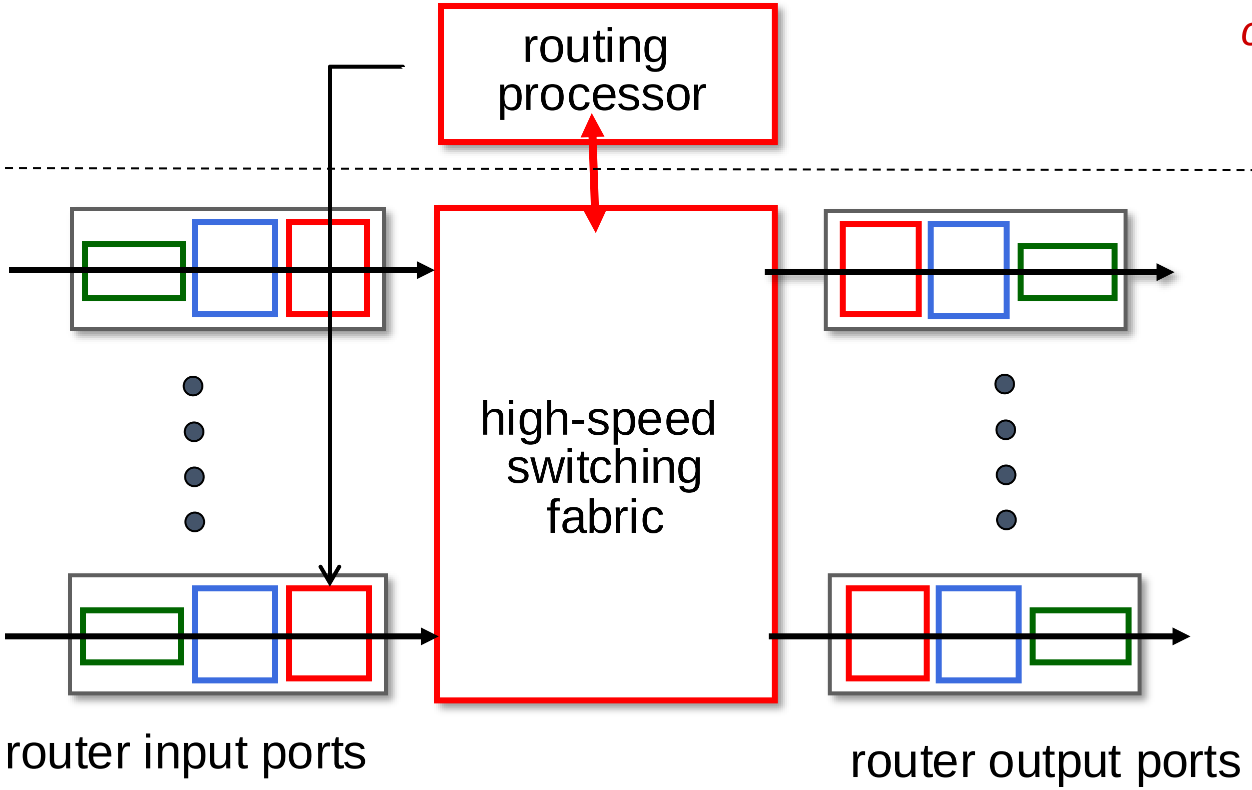
Remote controller computes, installs forwarding tables in routers.



# Data plane: Inside a router (forwarding)

# Router architecture overview

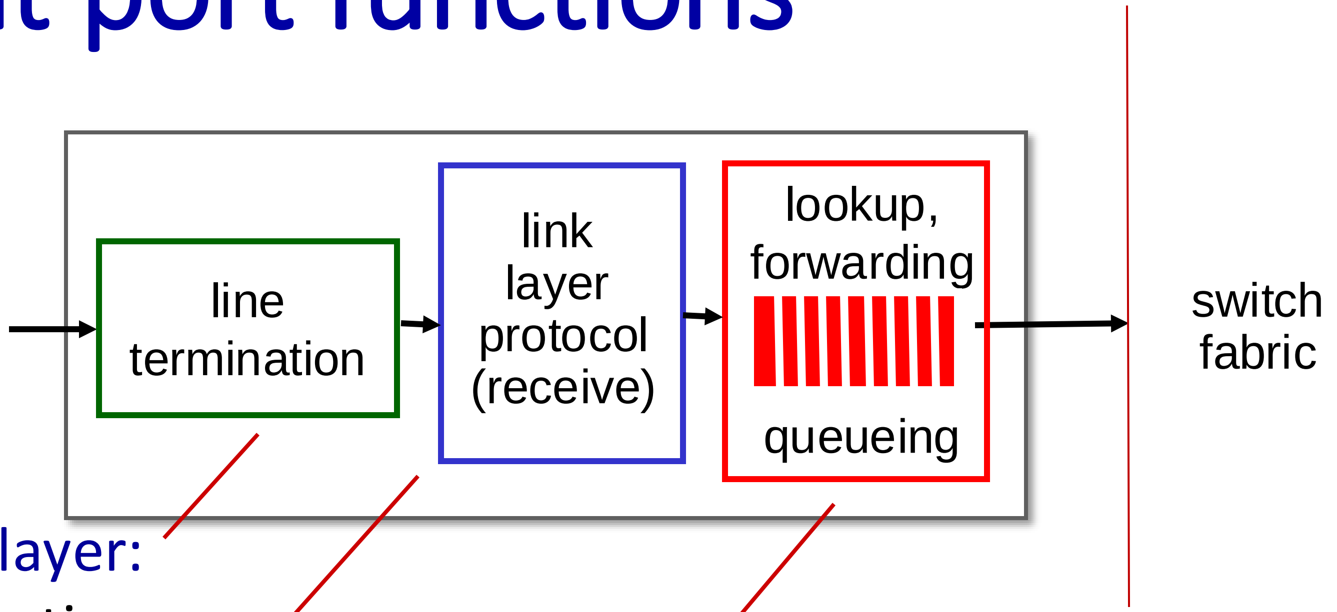
high-level view of generic router architecture:




*routing, management*
*control plane* (software)
operates in millisecond
time frame

*forwarding data plane*
(hardware) operates
in nanosecond
timeframe

# Input port functions



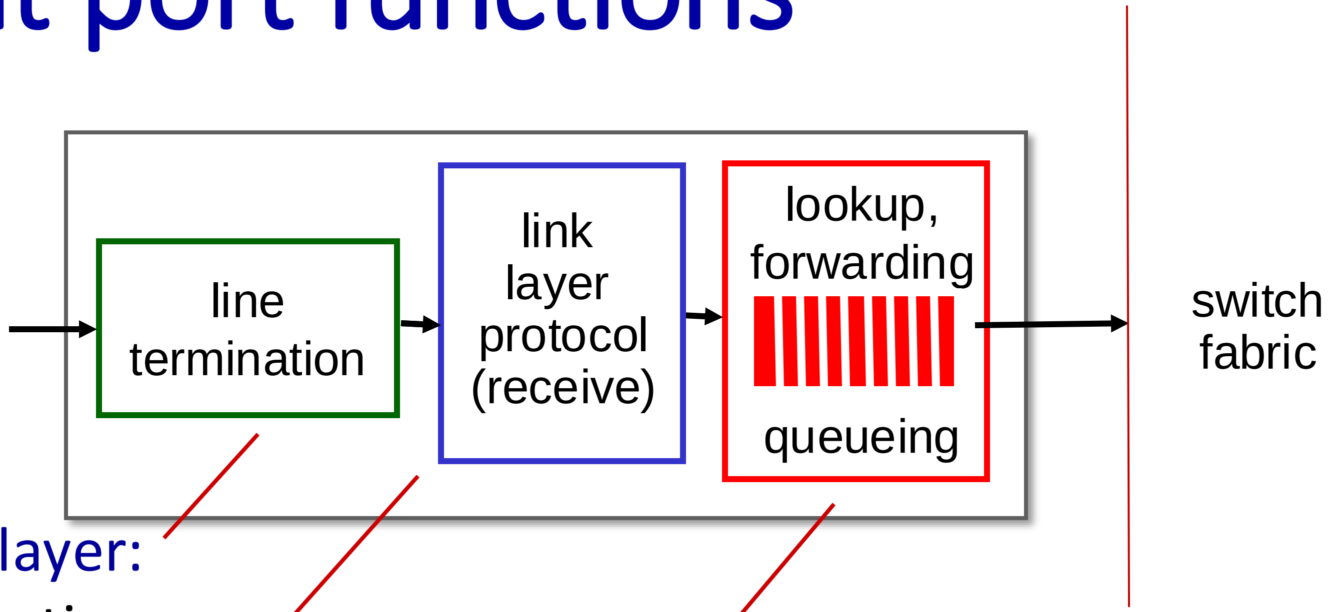

physical layer:
bit-level reception

link layer:
e.g., Ethernet

decentralized switching:

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

# Input port functions




physical layer:
bit-level reception

link layer:
e.g., Ethernet

decentralized switching*:*

- using header field values, lookup output port using forwarding table in input port memory *("match plus action")*
- destination-based forw.: based on destination IP (traditional)
- generalized forw.: forward based on any set of header fields

# Destination-based forwarding

*forwarding table*

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

# Longest prefix matching

**longest prefix match**

choose *longest* address prefix that matches destination address.

| Destination Address Range | | | | Link interface |
|---|---|---|---|---|
| 11001000 | 00010111 | 00010*** | ******** | 0 |
| 11001000 | 00010111 | 00011000 | ******** | 1 |
| 11001000 | 00010111 | 00011*** | ******** | 2 |
| otherwise | | | | 3 |

examples:

11001000 00010111 00010110 10100001 which interface?

11001000 00010111 00011000 10101010 which interface?

# Longest prefix matching

**longest prefix match**

choose *longest* address prefix that matches destination address.

| Destination Address Range | Link interface |
|---|---|
| 11001000 00010111 00010*** ******** | 0 |
| 11001000 00010111 00011000 ******** | 1 |
| 11001000 00010111 00011*** ******** | 2 |
| otherwise | 3 |

match!

examples:

11001000 00010111 00010110 10100001 which interface?

11001000 00010111 00011000 10101010 which interface?

# Longest prefix matching

**longest prefix match**

choose *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 |

(shorter) match!

examples:

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

# Longest prefix matching

**longest prefix match**

choose *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 |

longest prefix match!

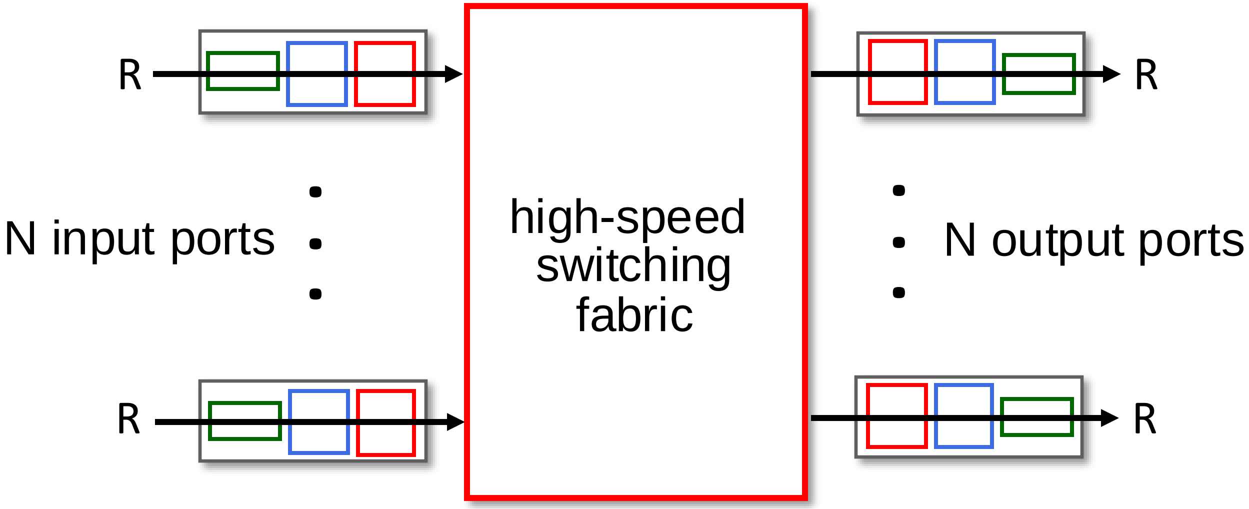
examples:

11001000 00010111 00010110 10100001 which interface?

11001000 00010111 00011000 10101010 which interface?

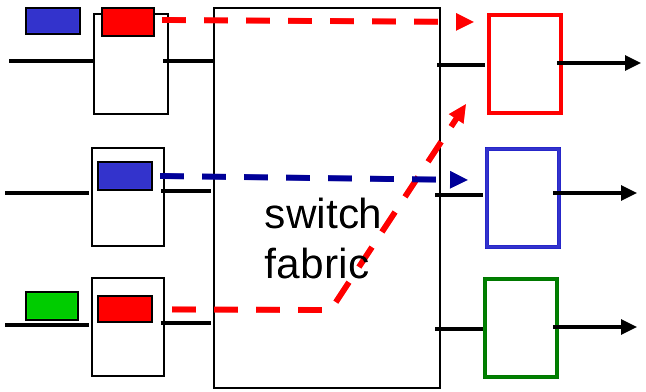
# Switching fabrics

- transfer packet from input link to appropriate output link

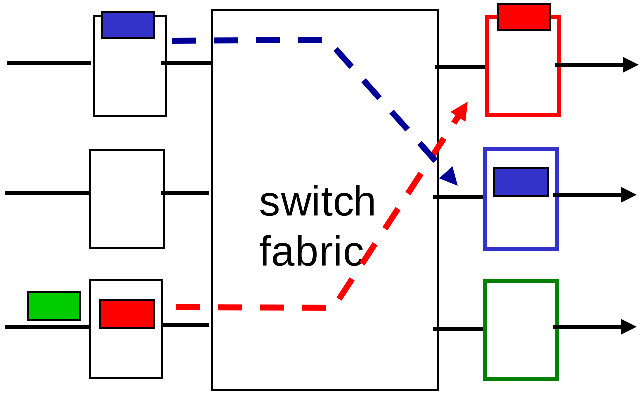


# Input port queuing

- If switch fabric slower than input ports combined -> queueing may occur at input queues
  - *queueing delay and loss due to input buffer overflow!*
- Head-of-the-Line (HOL) blocking: queued datagram at front of queue prevents others in queue from moving forward

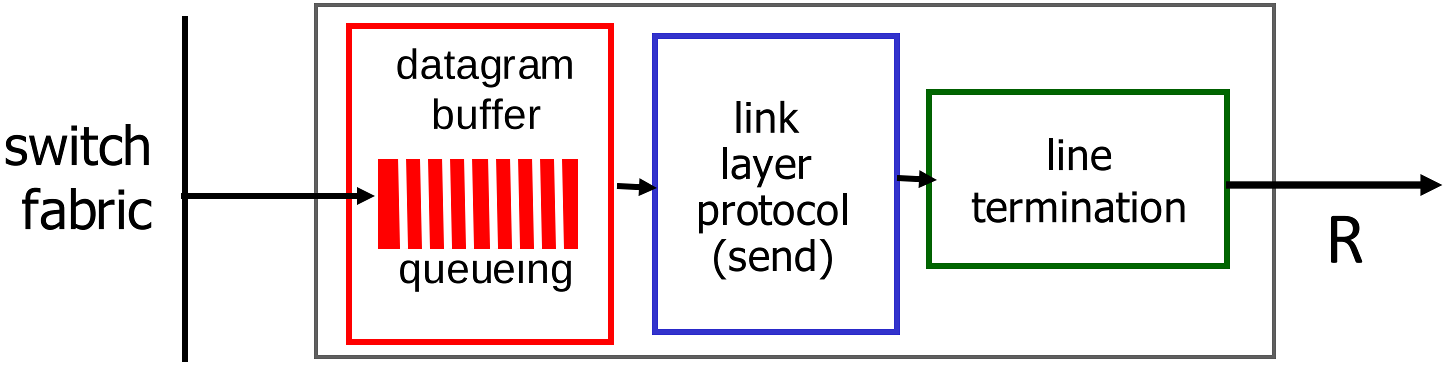



output port contention: only one red datagram can be transferred -- lower red packet is *blocked*




one packet time later: green packet experiences HOL blocking

# Output port queuing



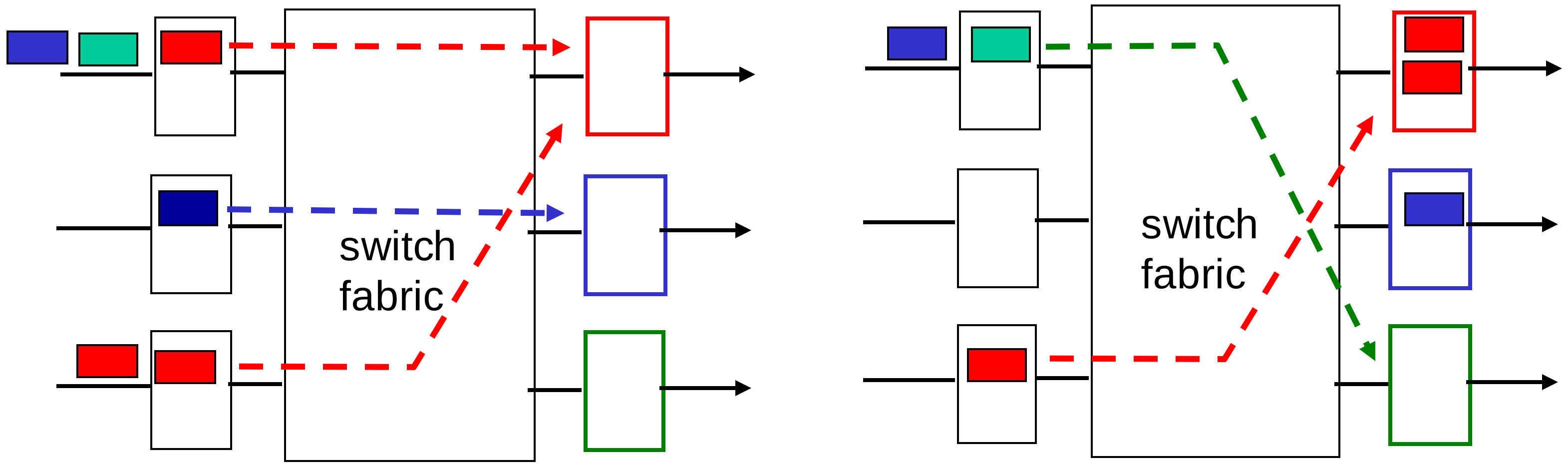

- *Buffering* required when datagrams arrive from fabric faster than link transmission rate.

Datagrams can be lost due to congestion, lack of buffers.

- *Scheduling* chooses among queued datagrams for transmission.

Priority scheduling – who gets best performance, network neutrality.

# Output port queuing

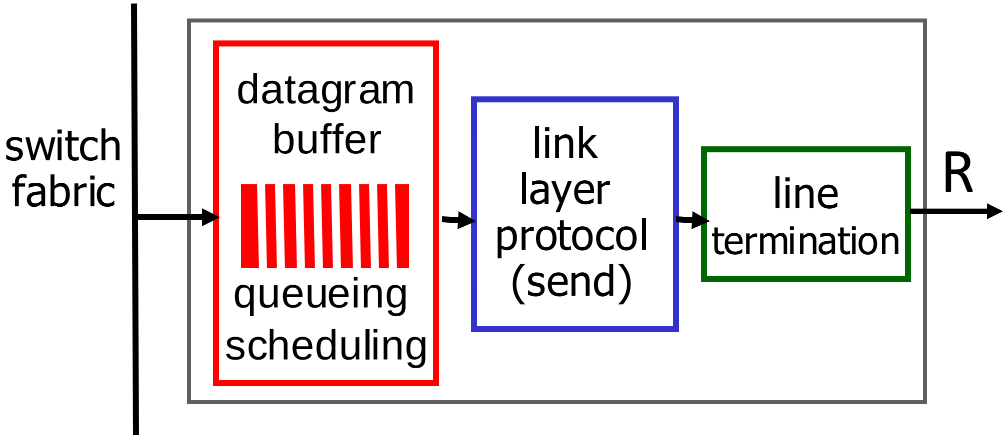



at *t*, packets move from input to output

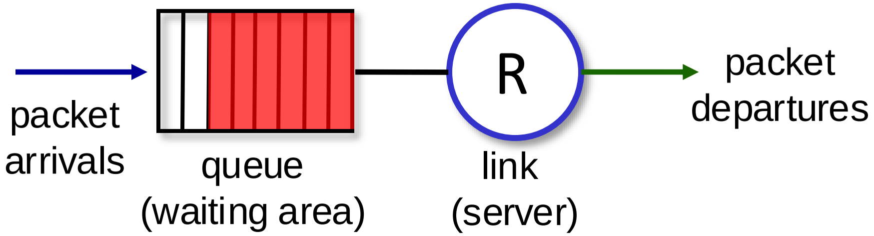
one packet time later

- buffering when arrival rate via switch exceeds output line speed
  - *queueing (delay) and loss due to output port buffer overflow!*
  - *drop policy:* which datagrams to drop if no free buffers?

# (Output) buffer management




Abstraction: queue




- drop policy: decides which packet to drop when packet arrives to a full buffer.
  - tail drop: drop newly arriving packet
  - priority: drop/remove on priority basis

# Packet scheduling

Router also needs a *scheduling policy* for deciding which packet to send next on link.

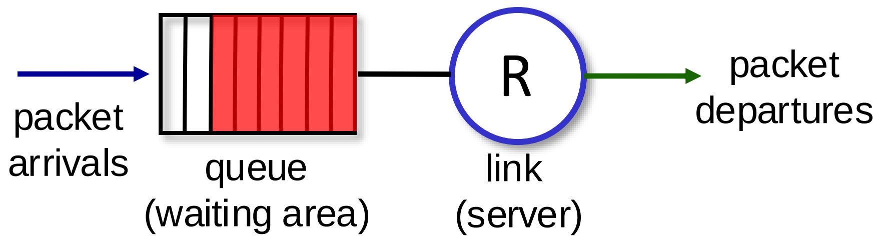
- first come, first served (FCFS)
- priority based
- round robin
- weighted fair queueing

# Scheduling policies: FCFS

*FCFS scheduling:*

- packets transmitted in order of arrival to output port
- a.k.a. first in, first out (FIFO)

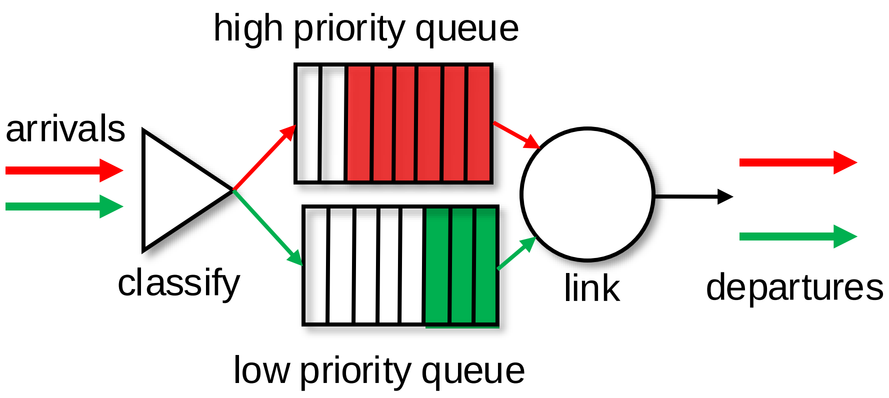
Abstraction: queue

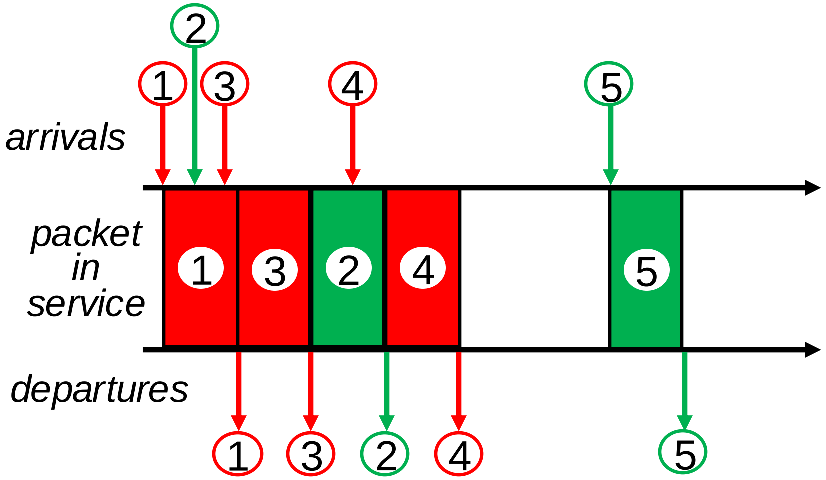


# Scheduling policies: priority

*Priority based scheduling:*

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

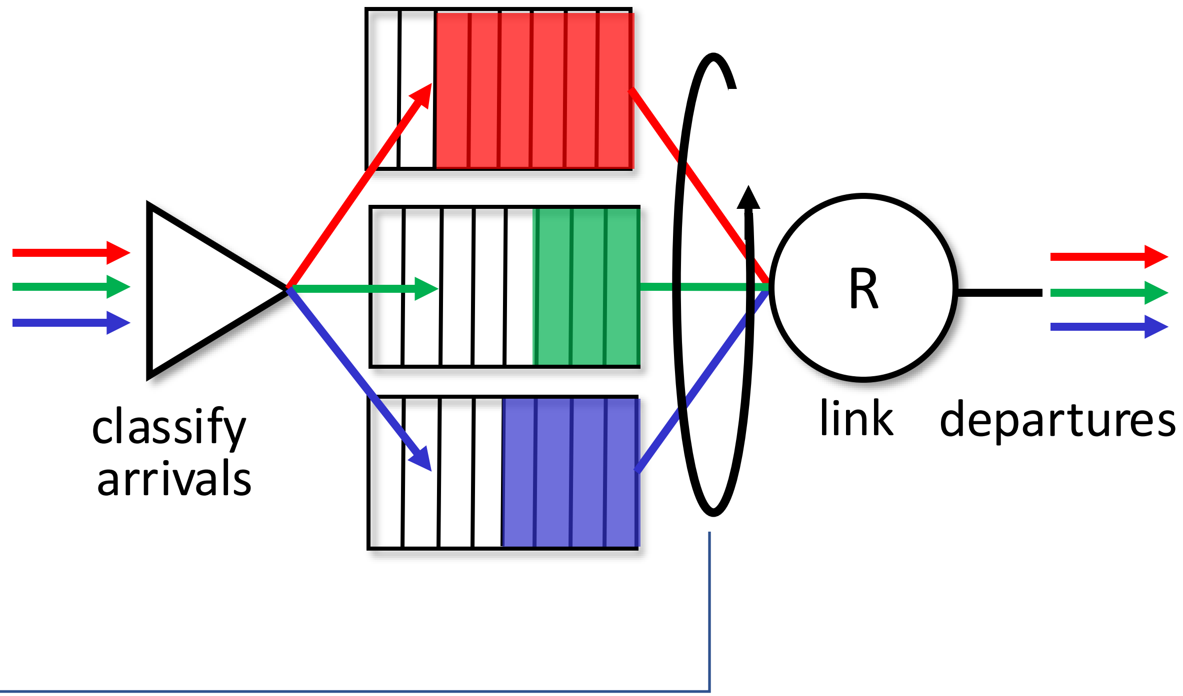


# Scheduling policies: round robin

*Round Robin (RR) scheduling:*

- arriving traffic classified, queued by class
  - any header fields can be used for classification
- server cyclically, repeatedly scans class queues, sending one complete packet from each class (if available) in turn



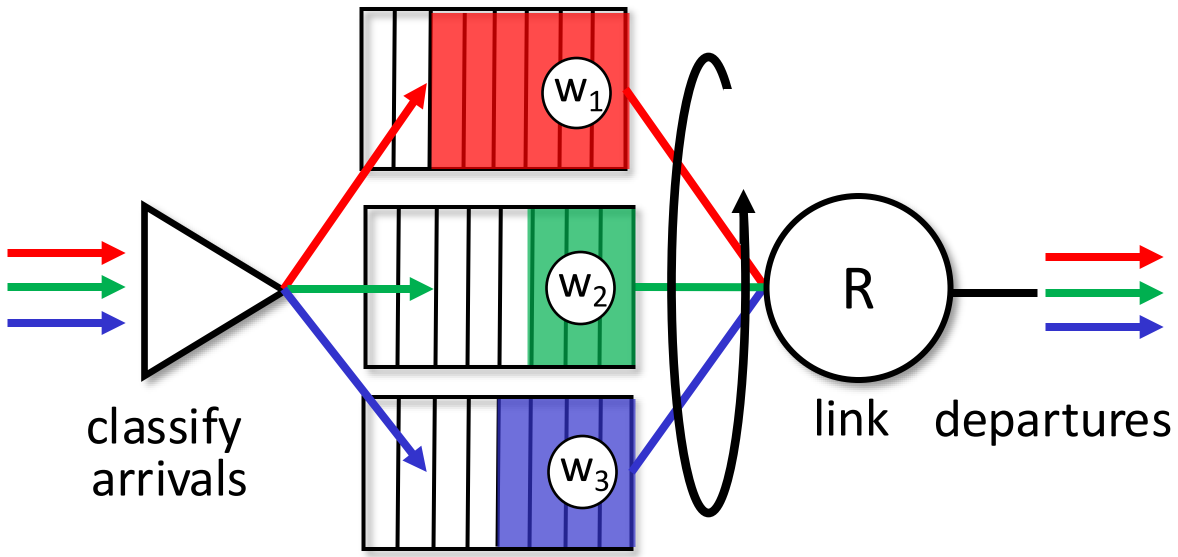
# Scheduling policies: weighted fair queueing

*Weighted Fair Queuing (WFQ):*

- generalized Round Robin
- each class, *i*, has weight, $w_i$, and gets weighted amount of service in each cycle:

$$\frac{w_i}{\Sigma_j w_j}$$

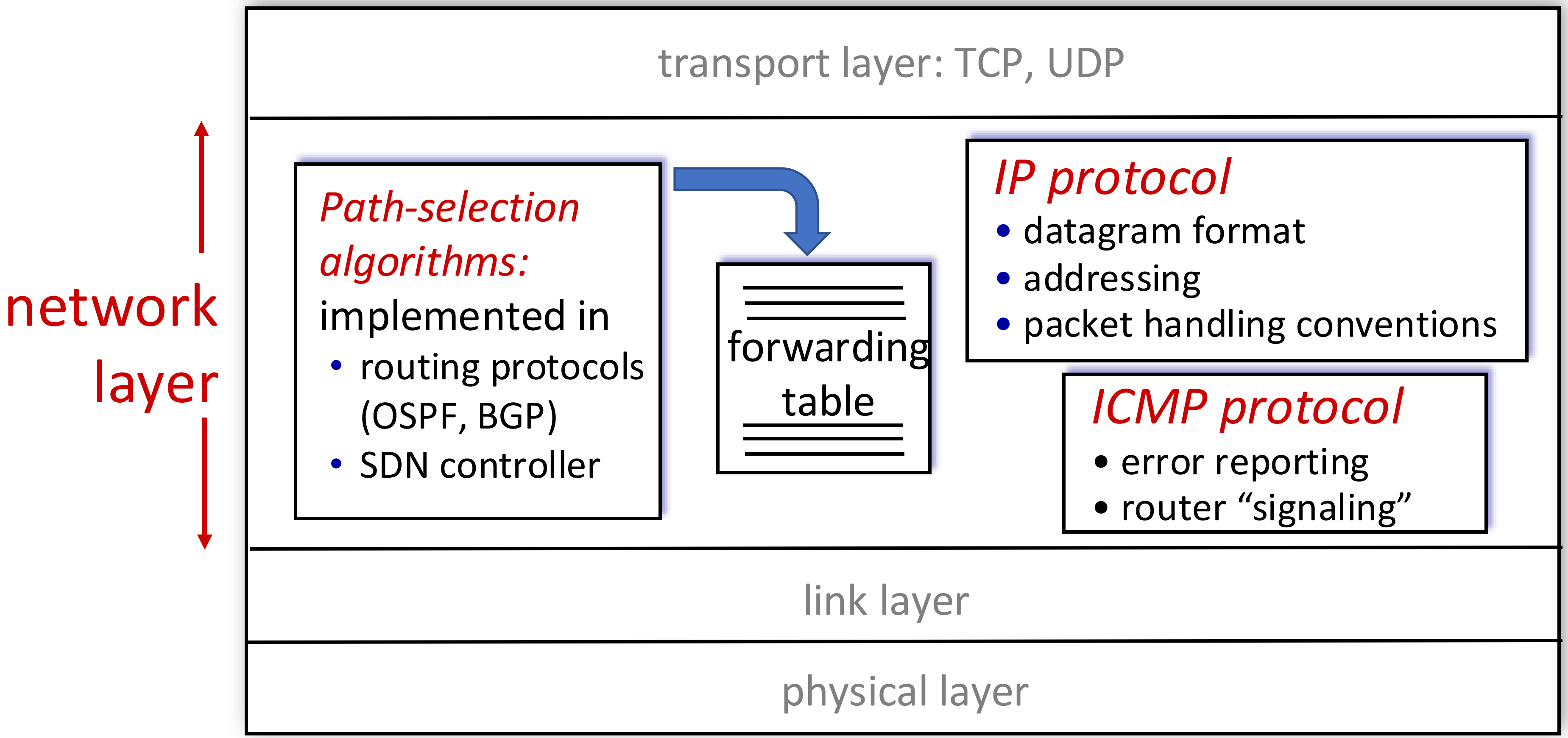
- minimum bandwidth guarantee (per-traffic-class)



# Internet Protocol (IP)

# Network Layer: Internet

host, router network layer functions:



# IP Datagram format

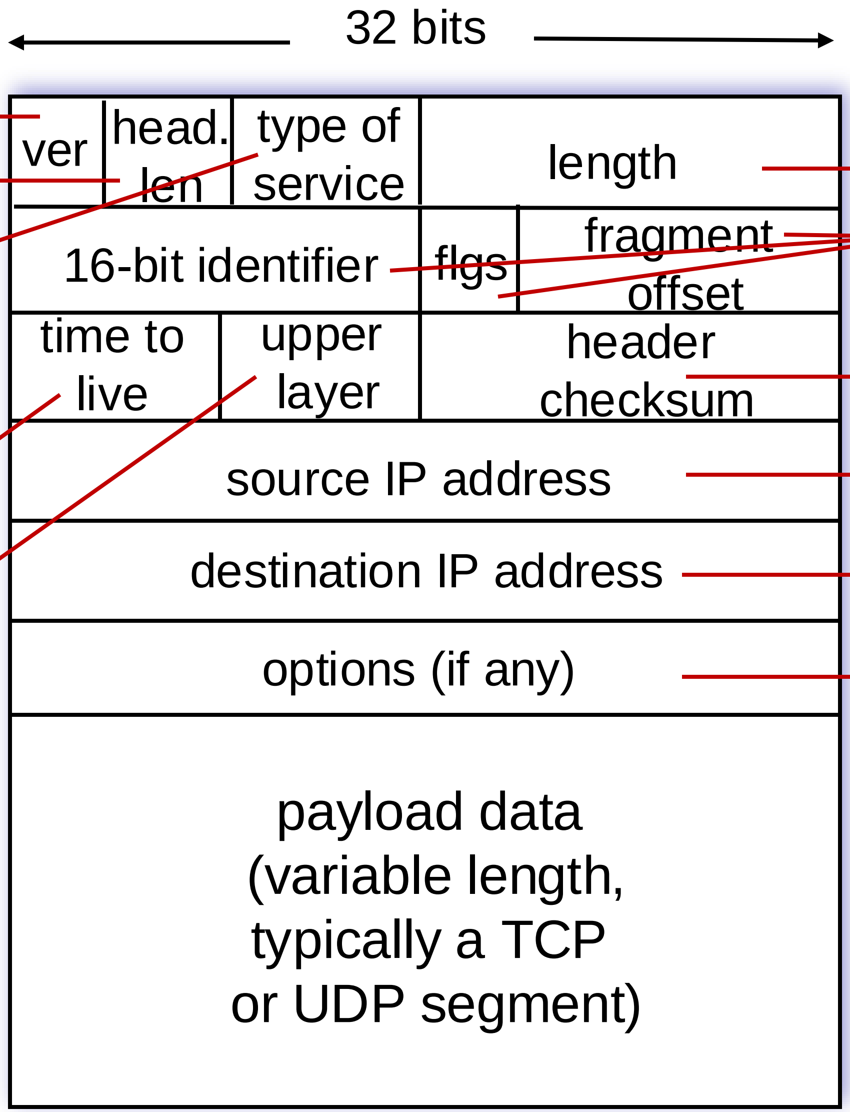
IP protocol version number

header length(bytes)

"type" of service:
- diffserv (0:5)
- ECN (6:7)

TTL: remaining max hops (decremented at each router)

upper layer protocol (e.g., TCP or UDP)




total datagram length (bytes)

fragmentation/ reassembly

header checksum

32-bit source IP address

32-bit destination IP address

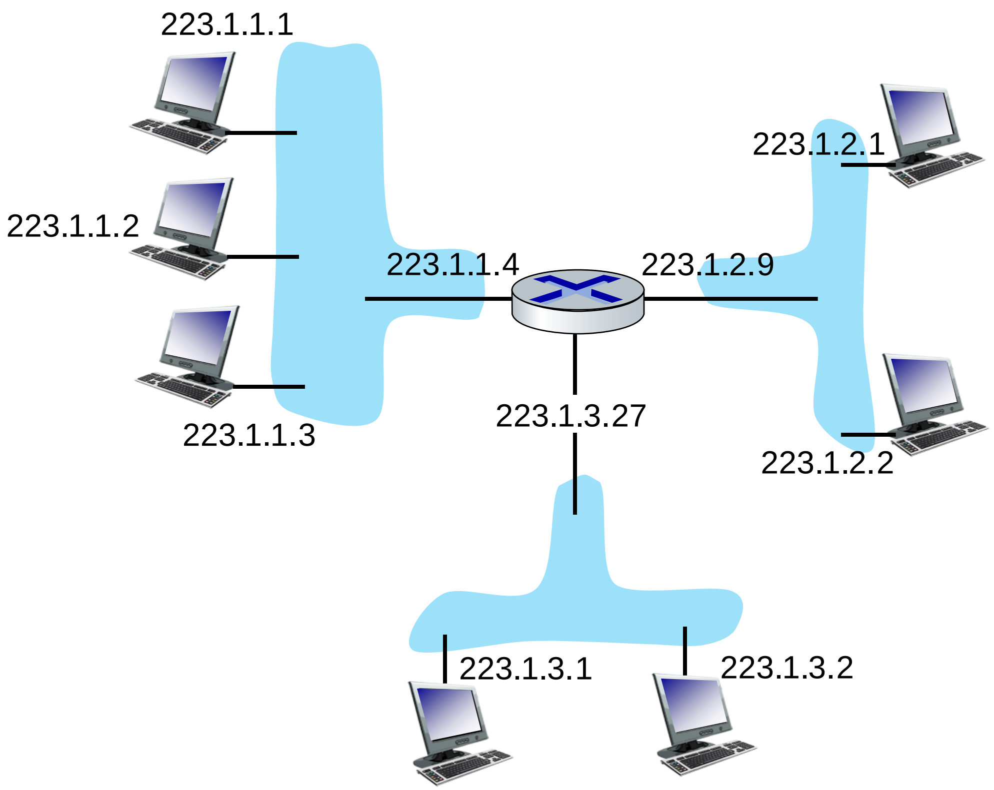
e.g., timestamp, record route taken

Maximum length: 64K bytes
Typically: 1500 bytes or less

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

# IP addressing: introduction

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



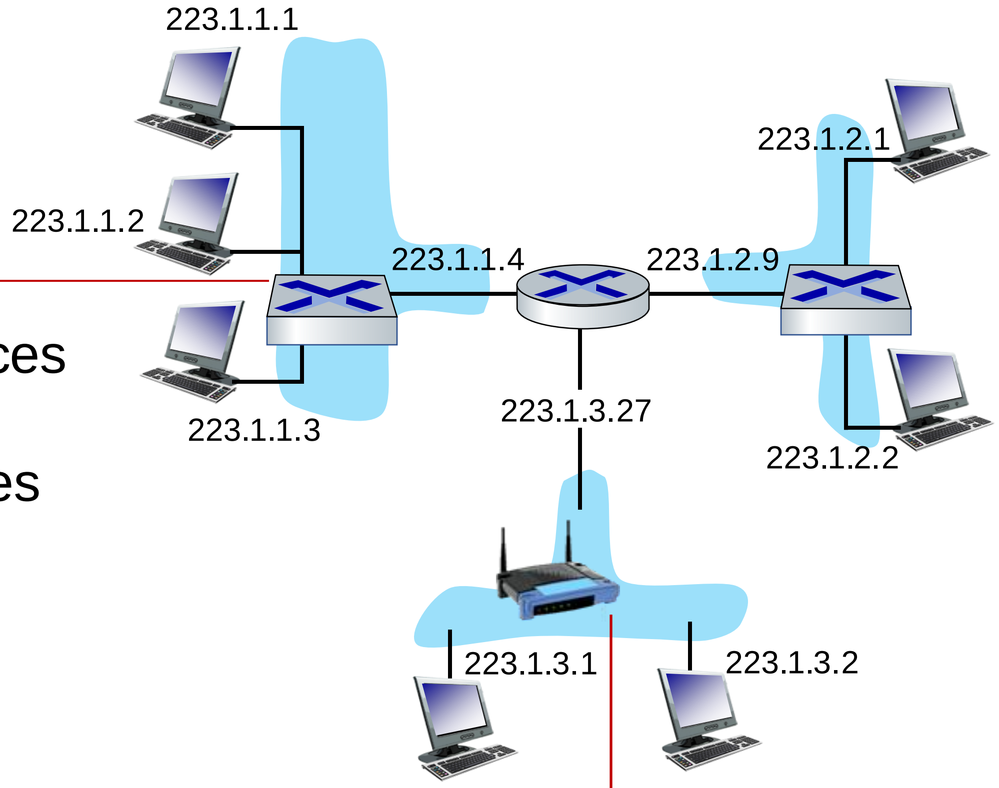

dotted-decimal IP address notation:

$$223.1.1.1 = \underbrace{11011111}_{223}\ \underbrace{00000001}_{1}\ \underbrace{00000001}_{1}\ \underbrace{00000001}_{1}$$

# IP addressing: introduction

Q: how are interfaces actually connected?

*A:* wired Ethernet interfaces connected by Ethernet switches

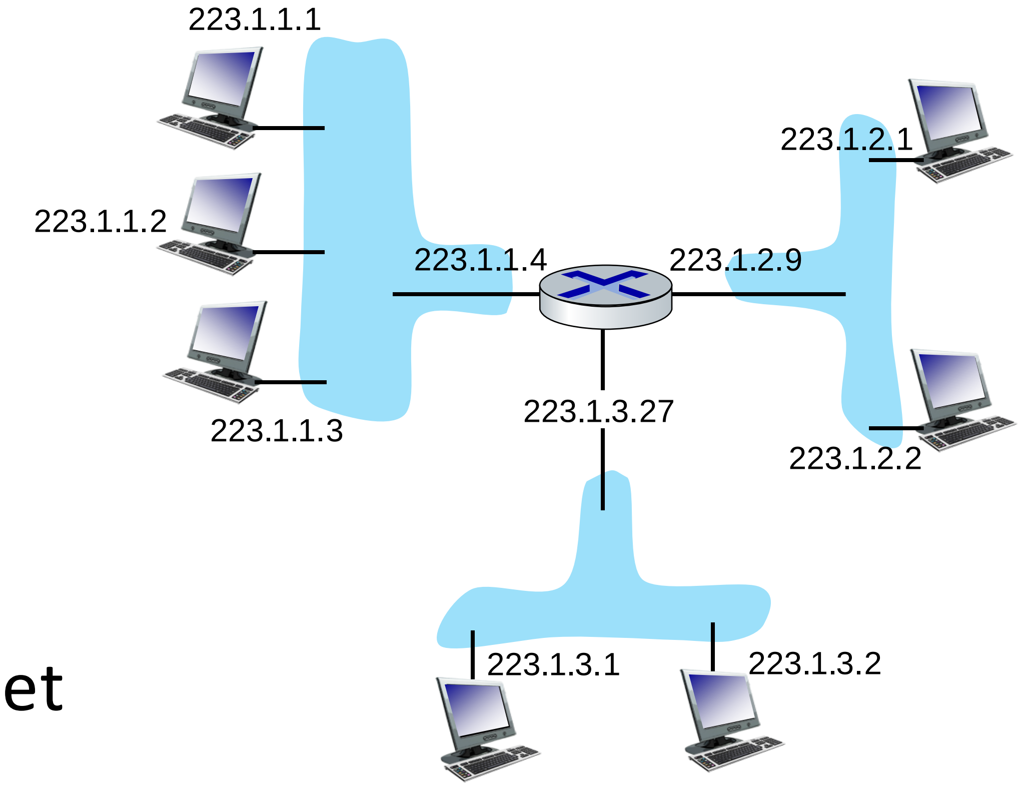



*For now:* don't need to worry about how one interface is connected to another (with no intervening router) – *a topic for our Link Layer lectures*.

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

# Subnets

- *What's a subnet ?*
  - device interfaces that can physically reach each other without passing through an intervening router
- IP addresses have structure:
  - subnet part: devices in same subnet have common high order bits
  - host part: remaining low order bits

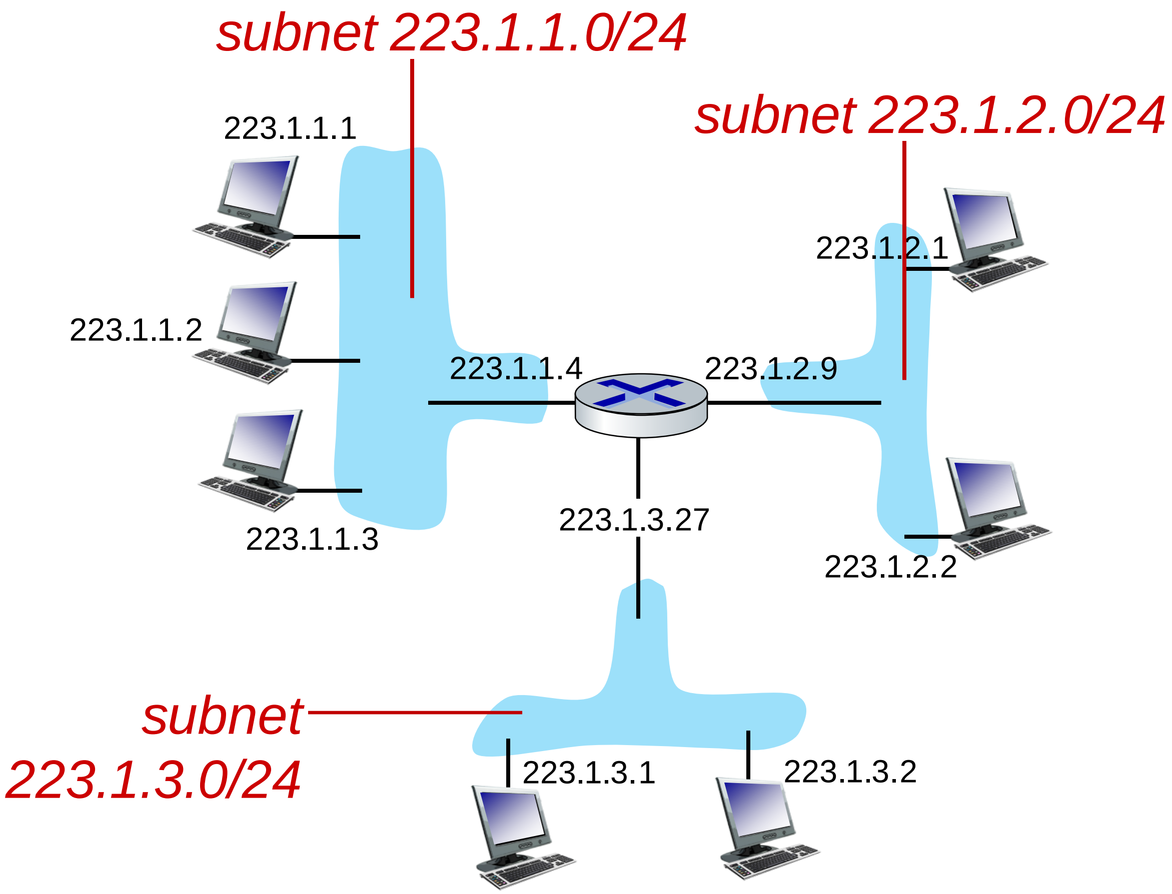



network consisting of 3 subnets

# Subnets

*Recipe for defining subnets:*

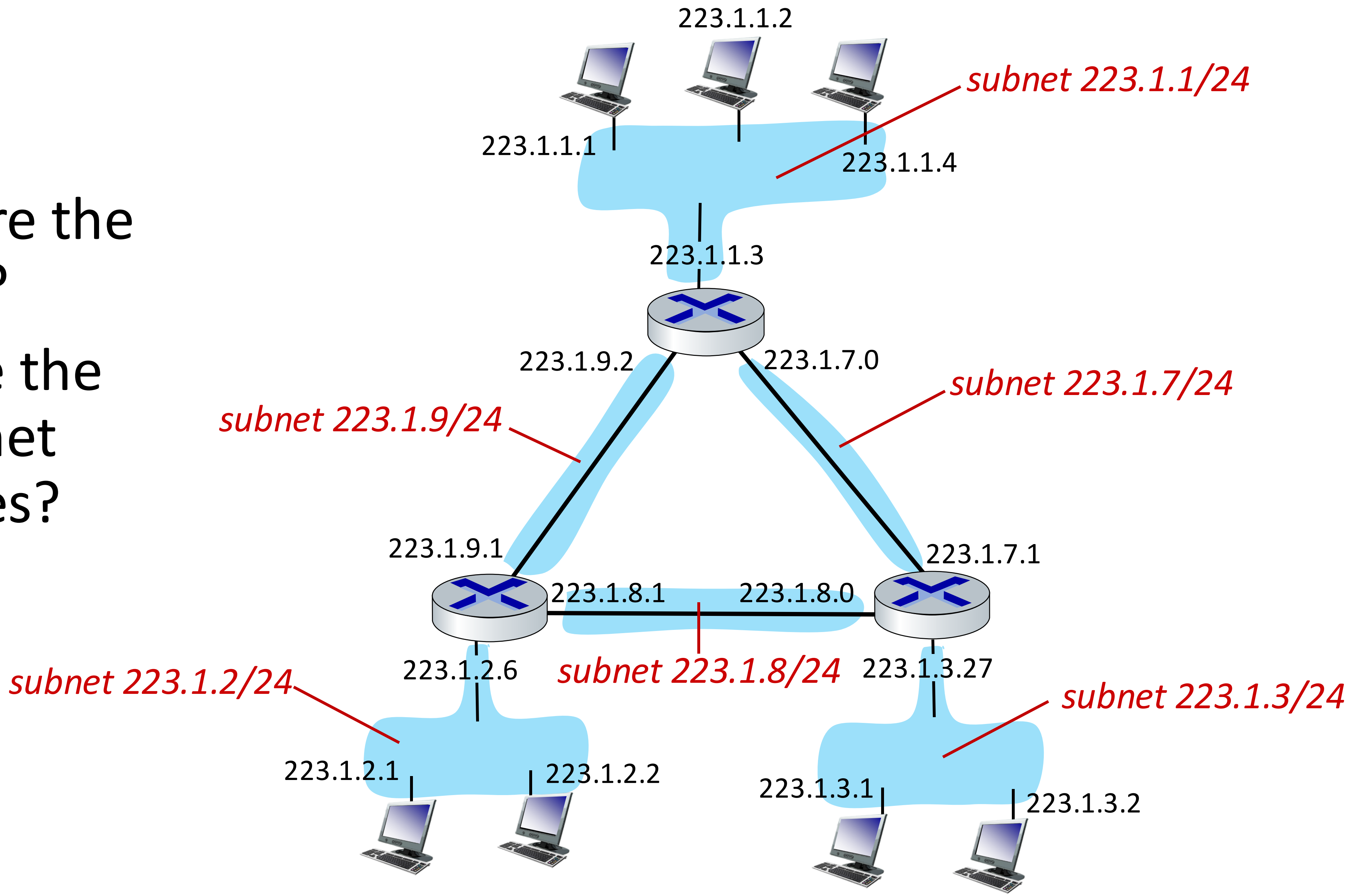
- detach each interface from its host or router, creating "islands" of isolated networks
- each isolated network is called a *subnet*




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

# Subnets

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



# IP addressing: CIDR

CIDR: Classless Inter-Domain Routing (pronounced “cider”)

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

subnet part | host part

11001000 00010111 00010000 00000000

200.23.16.0/23

# IP addresses: how to get one?

- can be hard-coded in config file (e.g., /etc/rc.config in UNIX)

OR

- DHCP: Dynamic Host Configuration Protocol
  - dynamically get address from as server ("plug-and-play")

# DHCP: Dynamic Host Configuration Protocol

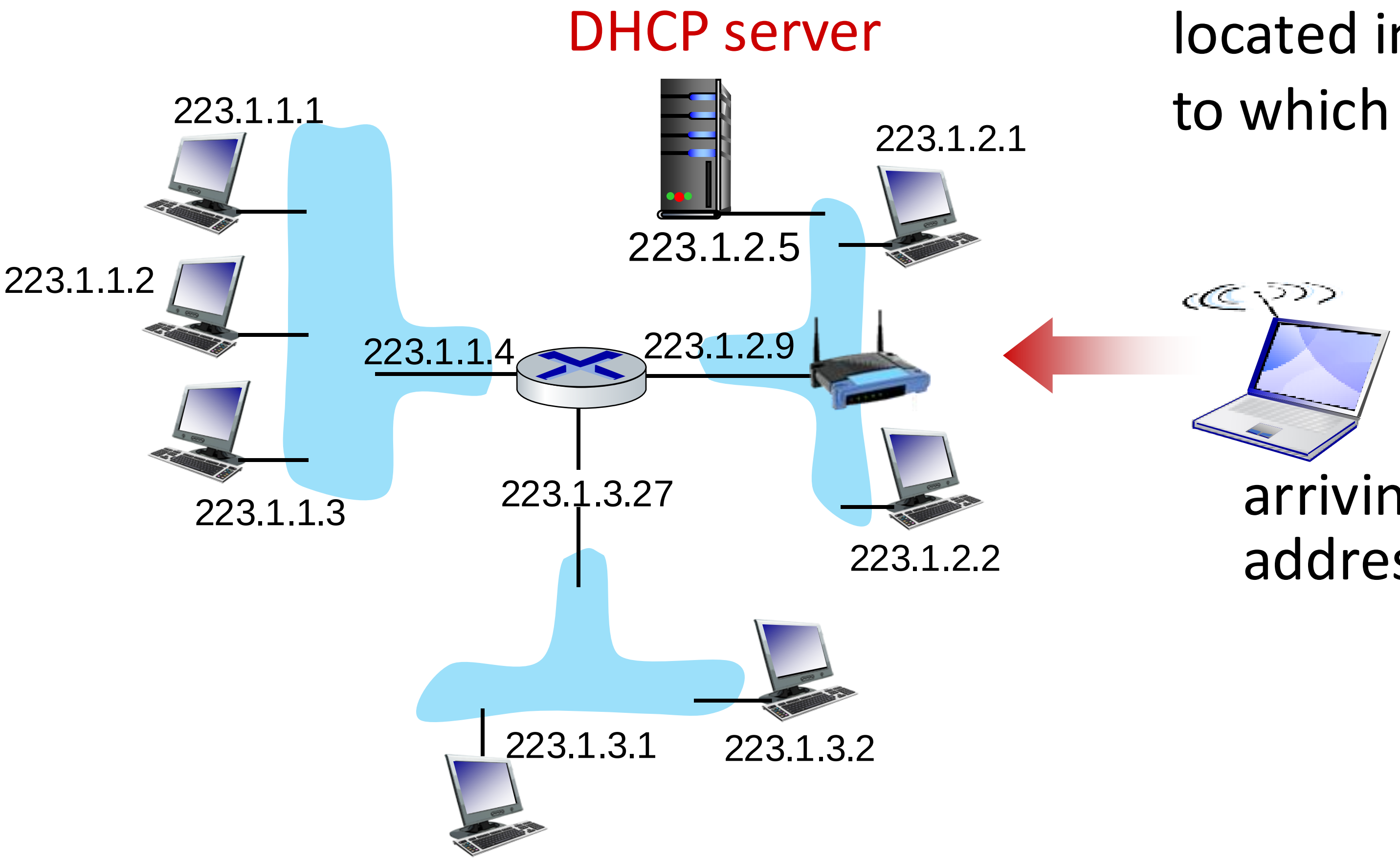
goal: host *dynamically* obtains IP address from network server when it "joins" network

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

## DHCP overview:

- host broadcasts DHCP discover msg [optional]
- DHCP server responds with DHCP offer msg [optional]
- host requests IP address: DHCP request msg
- DHCP server sends address: DHCP ack msg

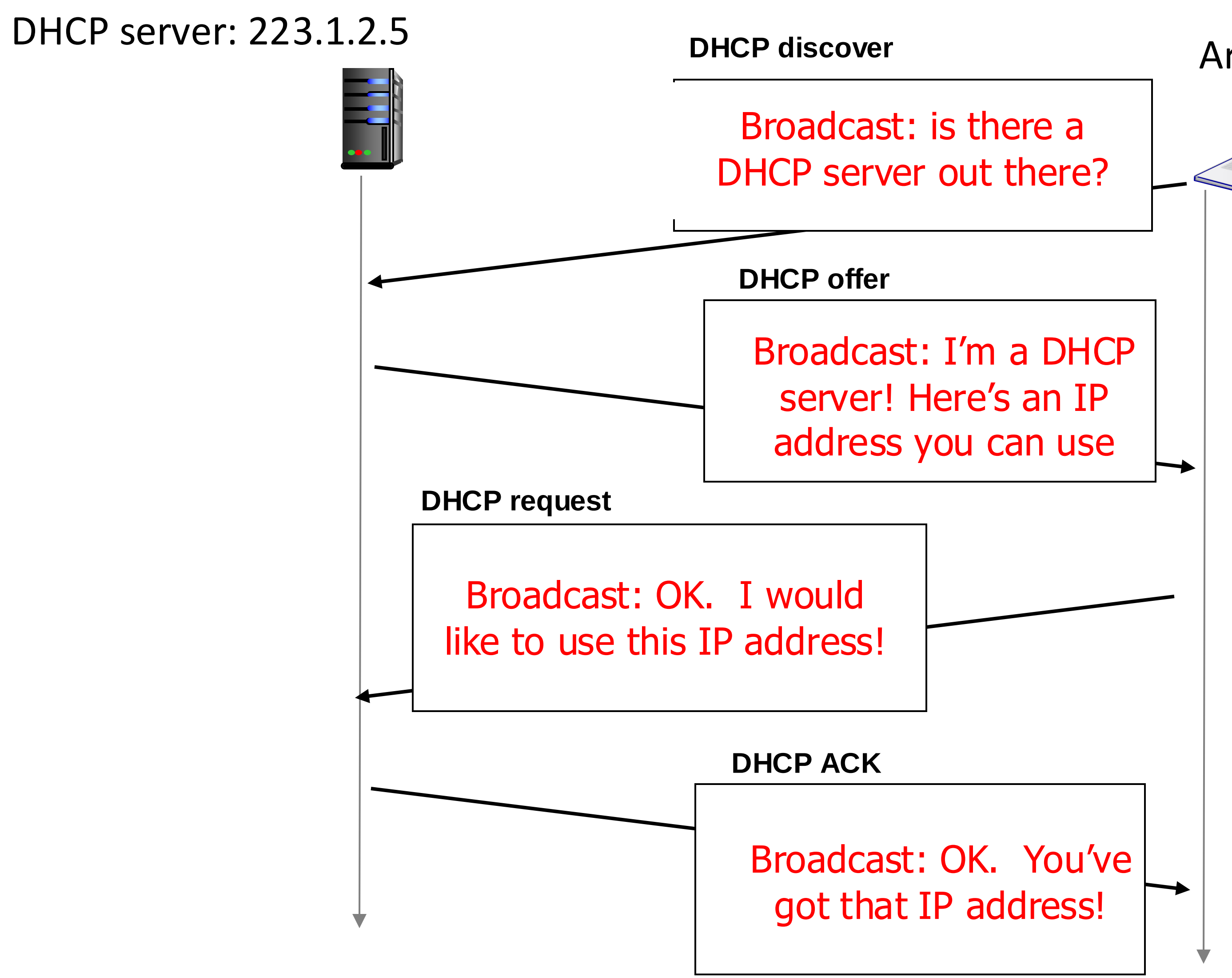
# DHCP client-server scenario




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

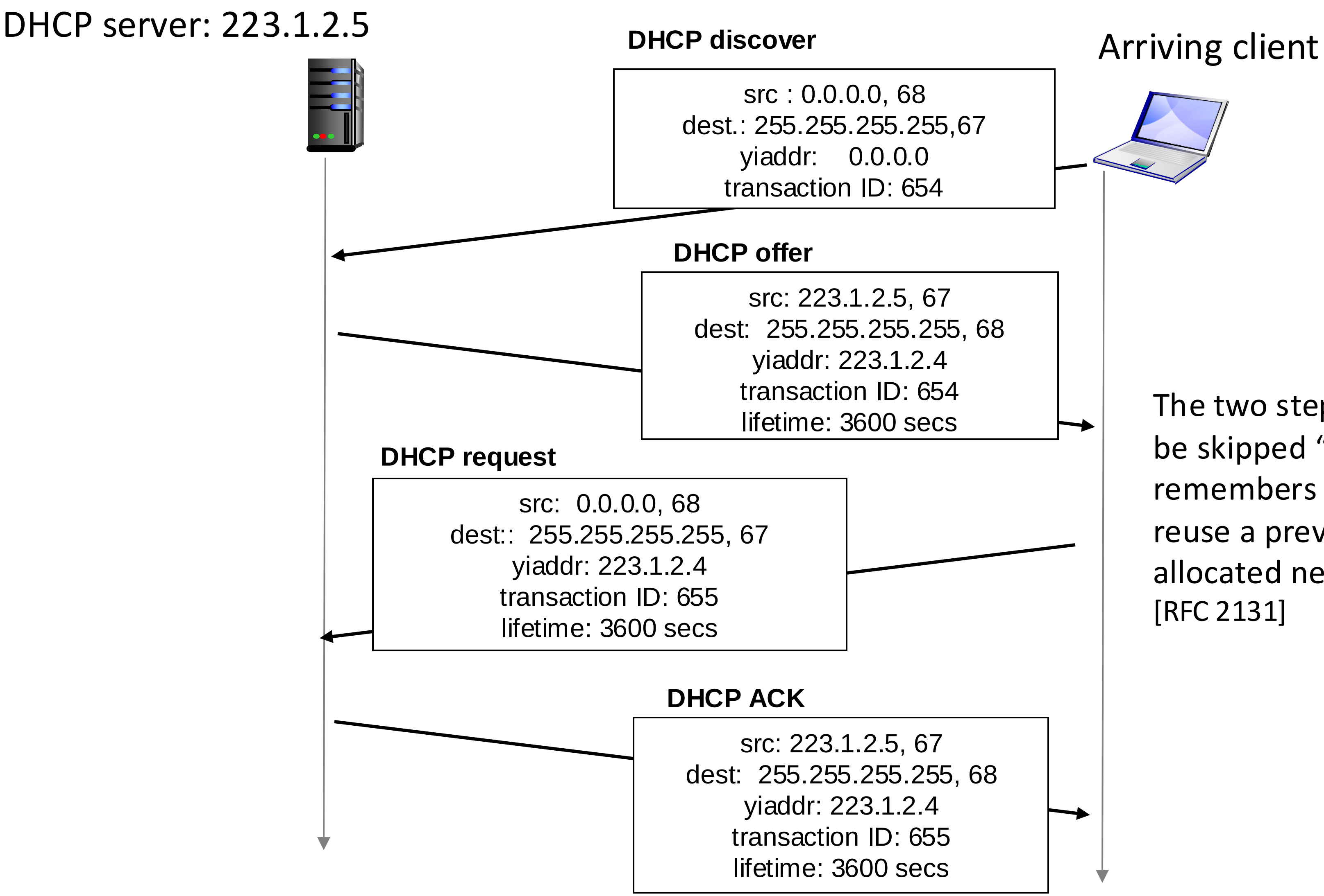
arriving DHCP client needs address in this network

# DHCP client-server scenario




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

# DHCP client-server scenario



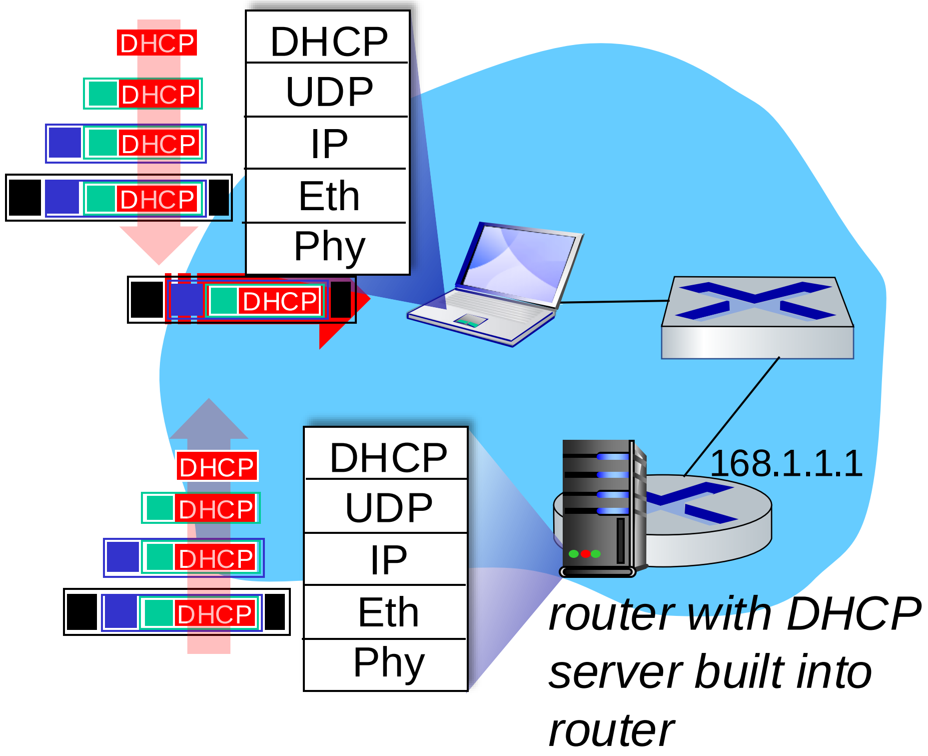

The two steps above can be skipped “if a client remembers and wishes to reuse a previously allocated network address” [RFC 2131]

# DHCP: more than IP addresses

DHCP can return more than just allocated IP address:

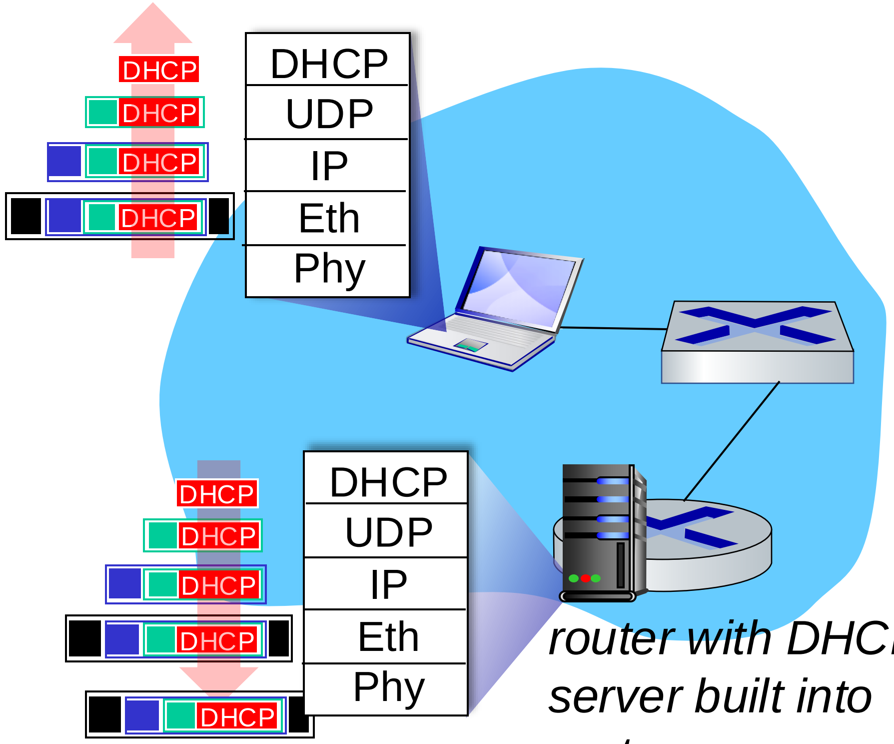
- address of first-hop router for client
- name and IP address of DNS sever
- network mask (indicating network versus host portion of address)

# DHCP: example




- Connecting laptop will use DHCP to get IP address, address of first-hop router, address of DNS server.
- DHCP REQUEST message encapsulated in UDP, encapsulated in IP, encapsulated in Ethernet
- Ethernet frame broadcast (dest: FFFFFFFFFFFF) on LAN, received at router running DHCP server
- Ethernet demux'ed to IP demux'ed, UDP demux'ed to DHCP

# DHCP: example




*router with DHCP server built into router*

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

# IP addresses: how to get one?

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

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

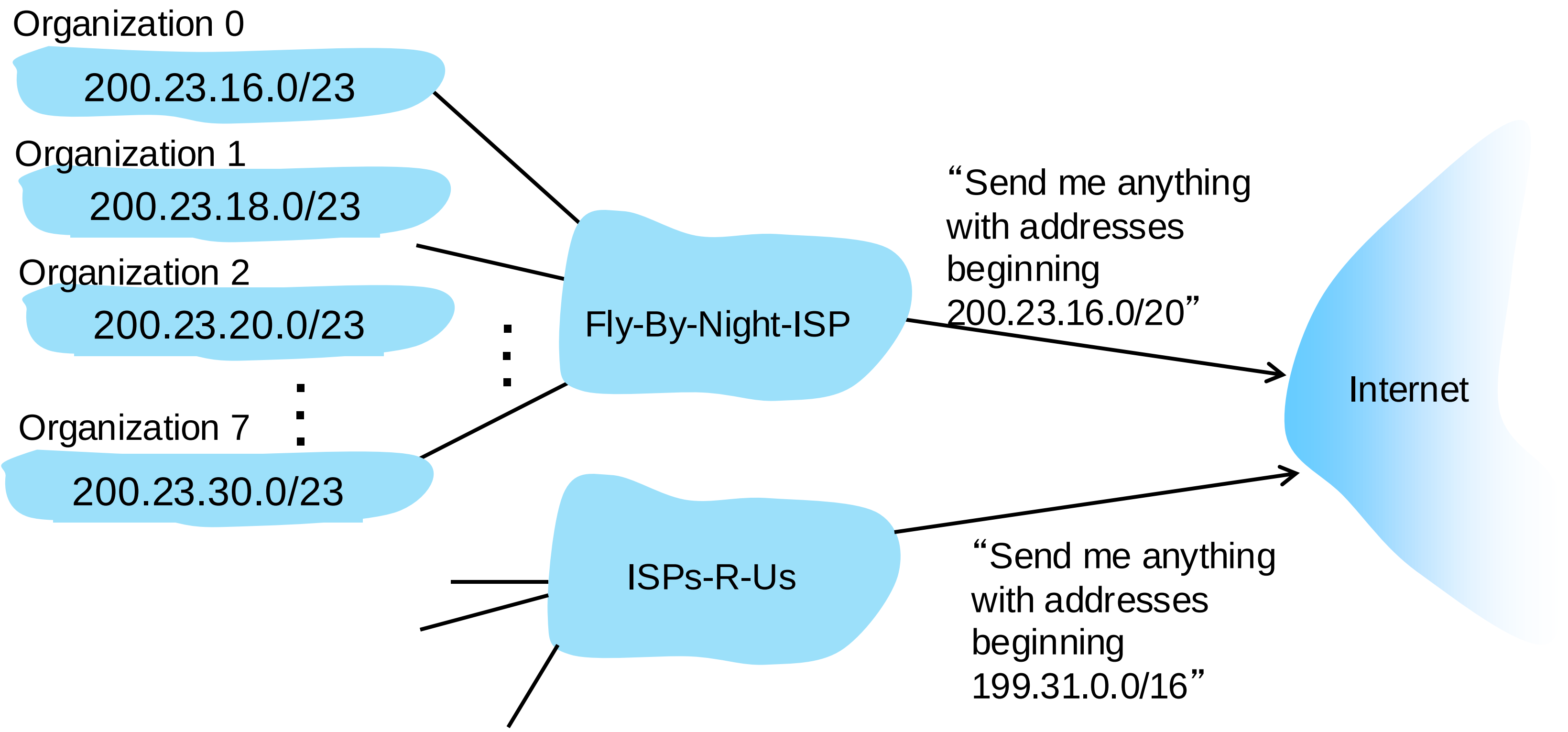
| | | | | | |
|---|---|---|---|---|---|
| ISP's block | 11001000 | 00010111 | 00010000 | 00000000 | 200.23.16.0/20 |

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

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

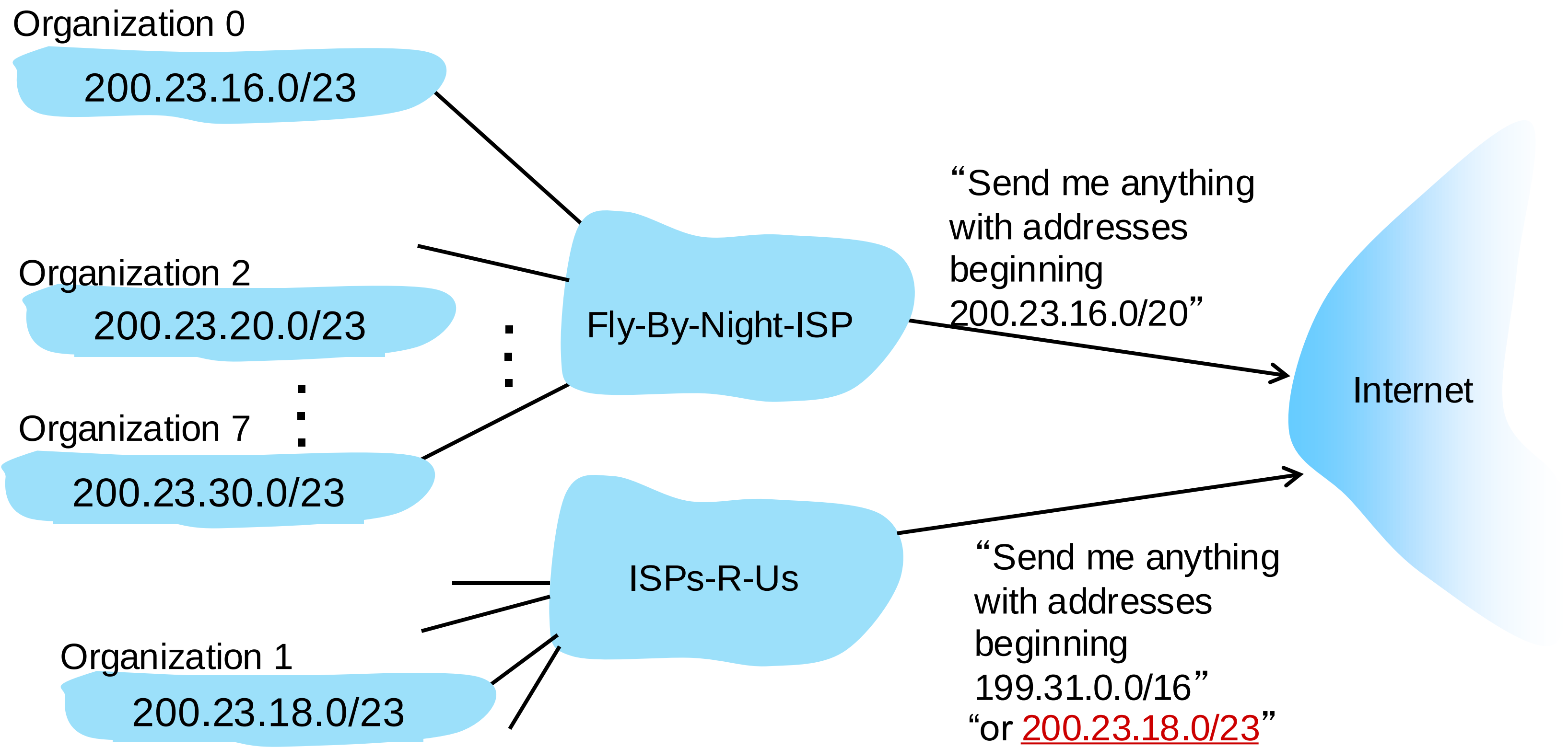
# Hierarchical addressing: route aggregation

hierarchical addressing allows efficient advertisement of routing information:



# Hierarchical addressing: more specific routes

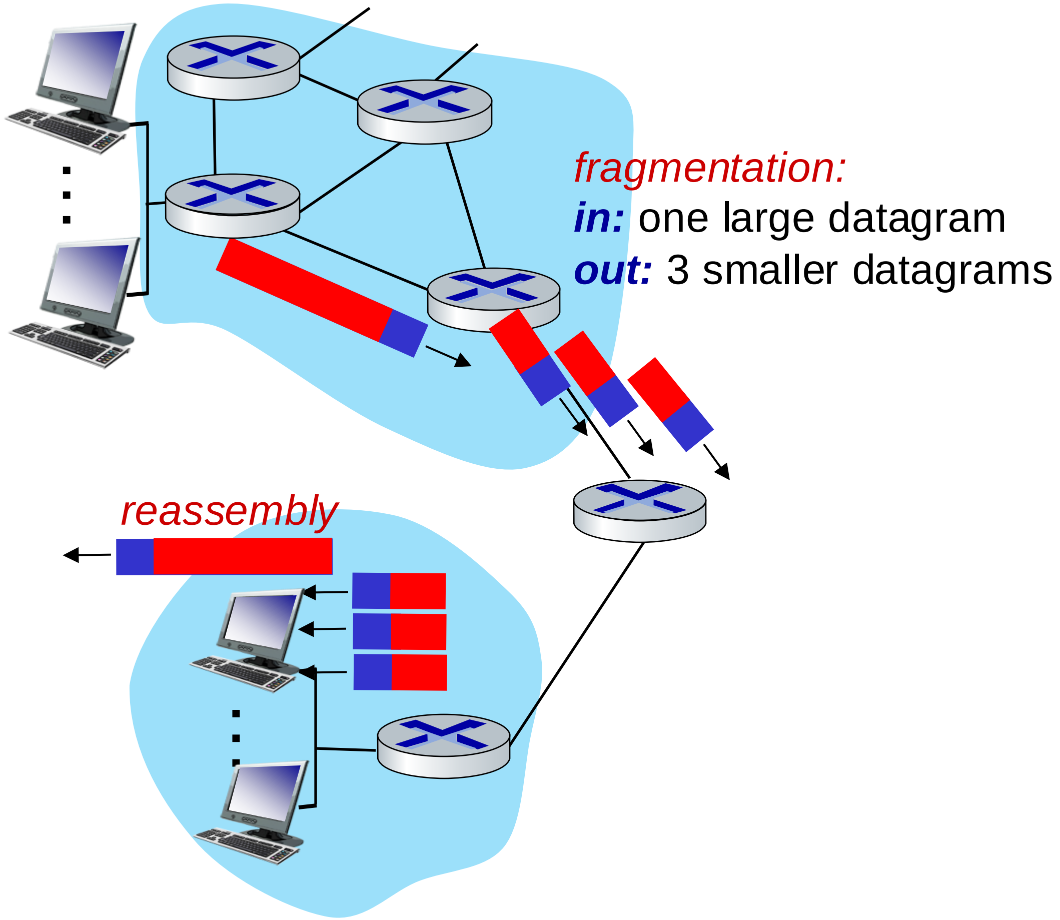
- Organization 1 moves from Fly-By-Night-ISP to ISPs-R-Us
- ISPs-R-Us now advertises a more specific route to Organization 1




Dr. Tanir Ozcelebi ( t.ozcelebi@tue.nl )

# IP fragmentation/reassembly

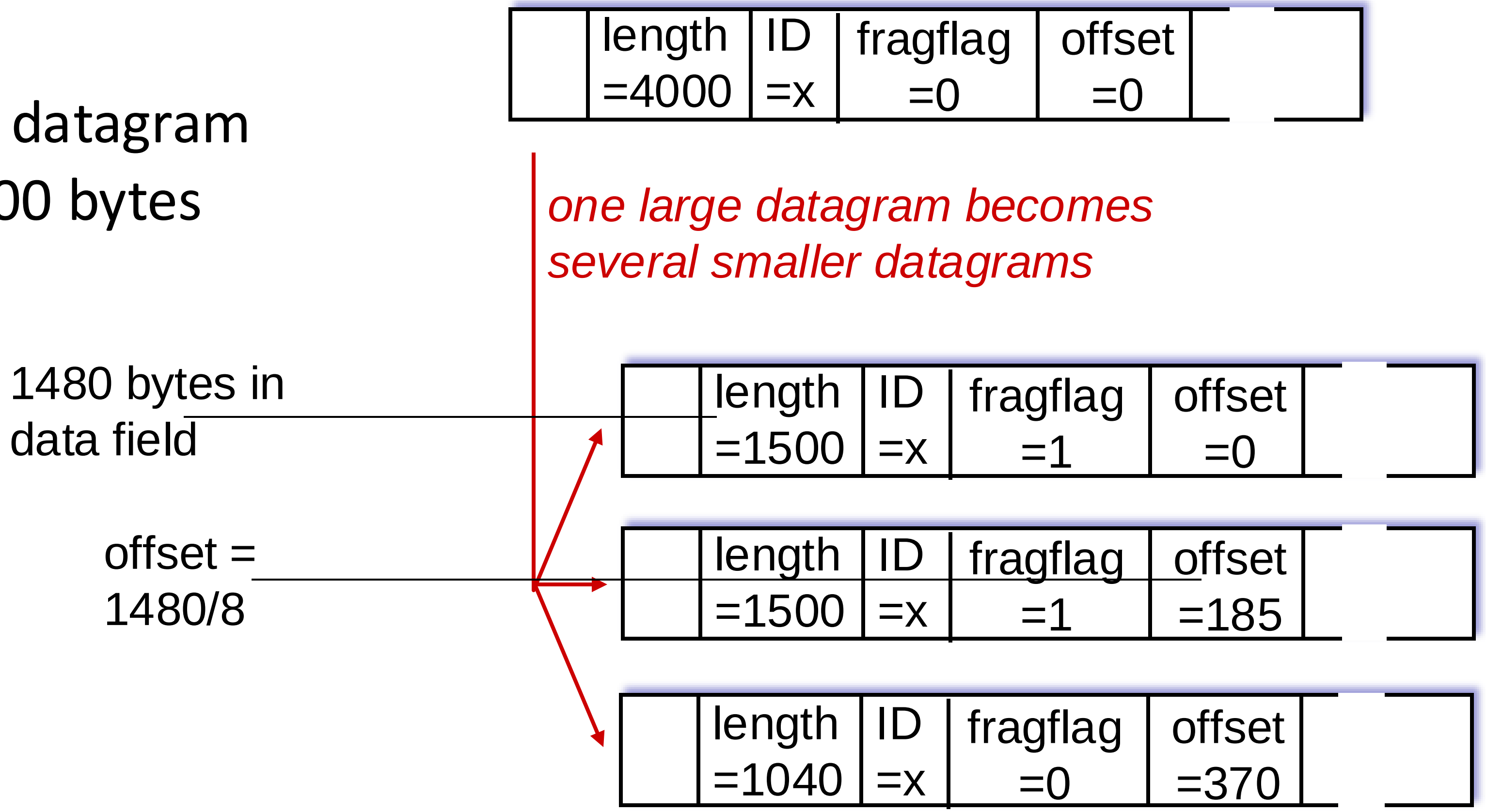
- network links have MTU (max. transfer size) - largest possible link-level frame
  - different link types, different MTUs
- large IP datagram divided ("fragmented") within net
  - one datagram becomes several datagrams
  - "reassembled" only at *destination*
  - IP header bits used to identify, order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



# IP fragmentation/reassembly

## example:

- 4000 byte datagram
- MTU = 1500 bytes



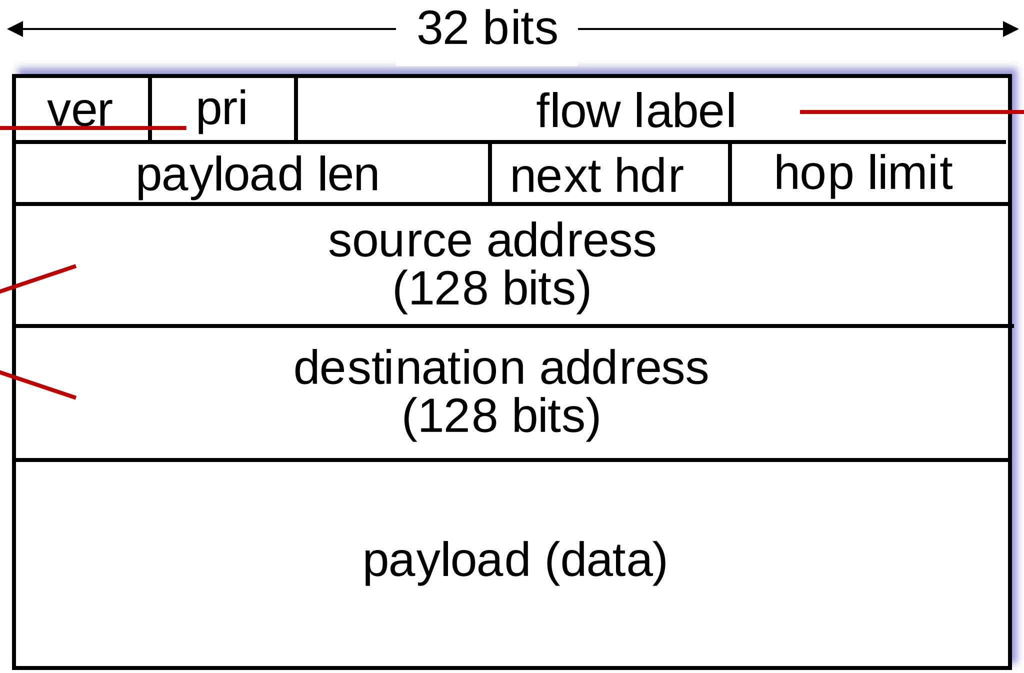
# IPv6: motivation

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

# IPv6 datagram format

priority: identify priority among datagrams in flow

128-bit IPv6 addresses



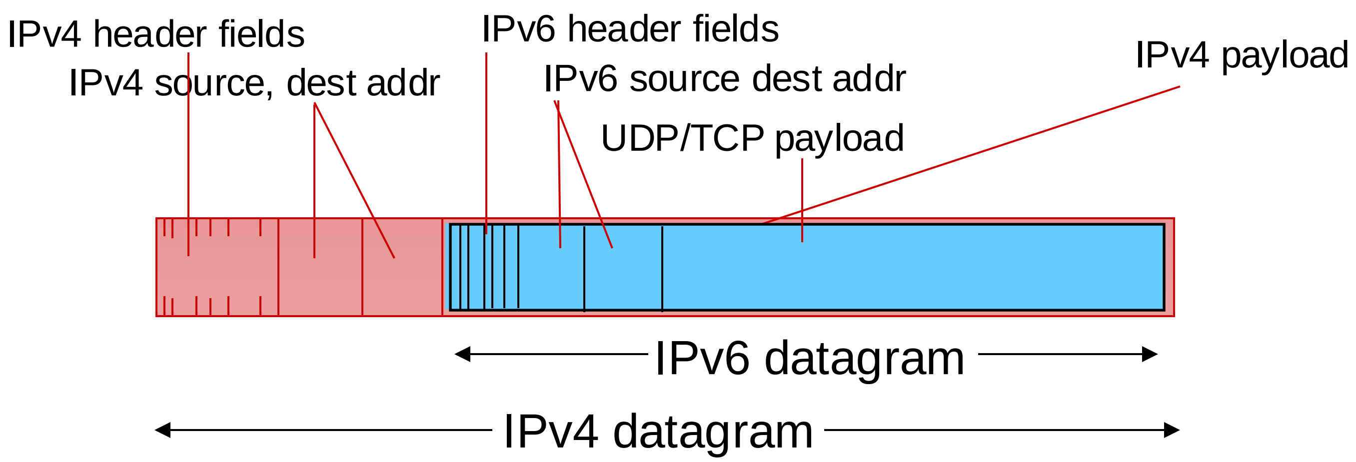

flow label: identify datagrams in same "flow." (concept of "flow" not well defined).

What's missing (compared with IPv4):

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

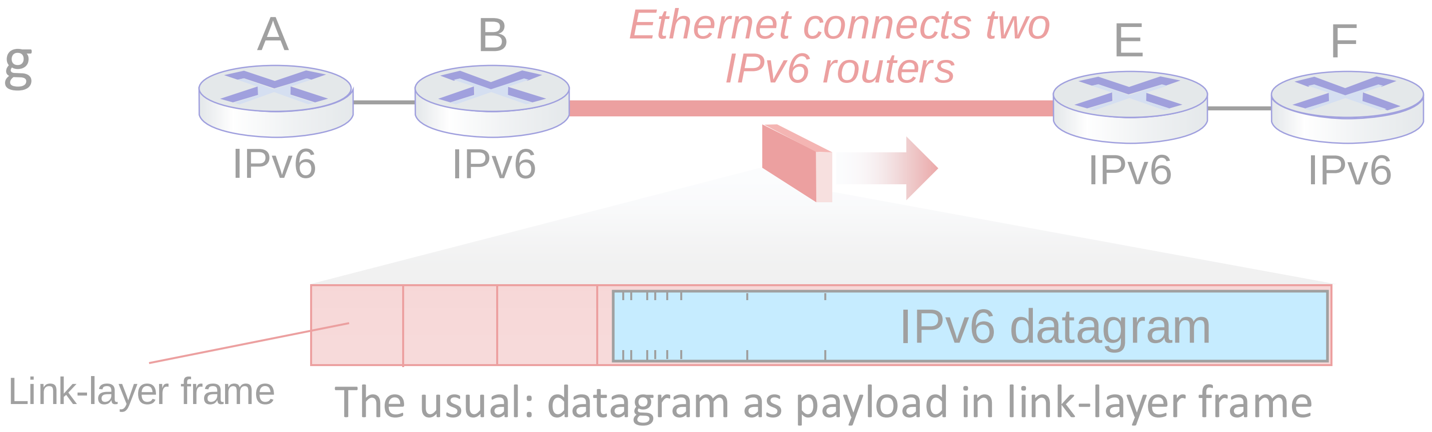
# Transition from IPv4 to IPv6

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

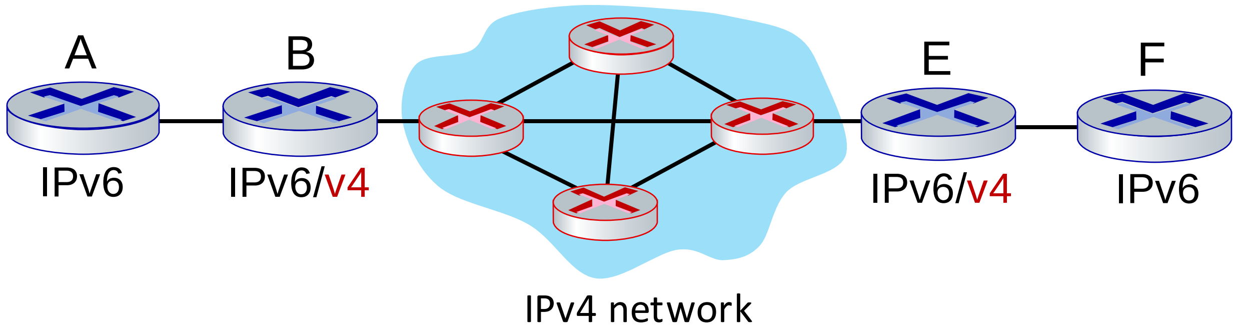


# Tunneling and encapsulation

Ethernet connecting two IPv6 routers:

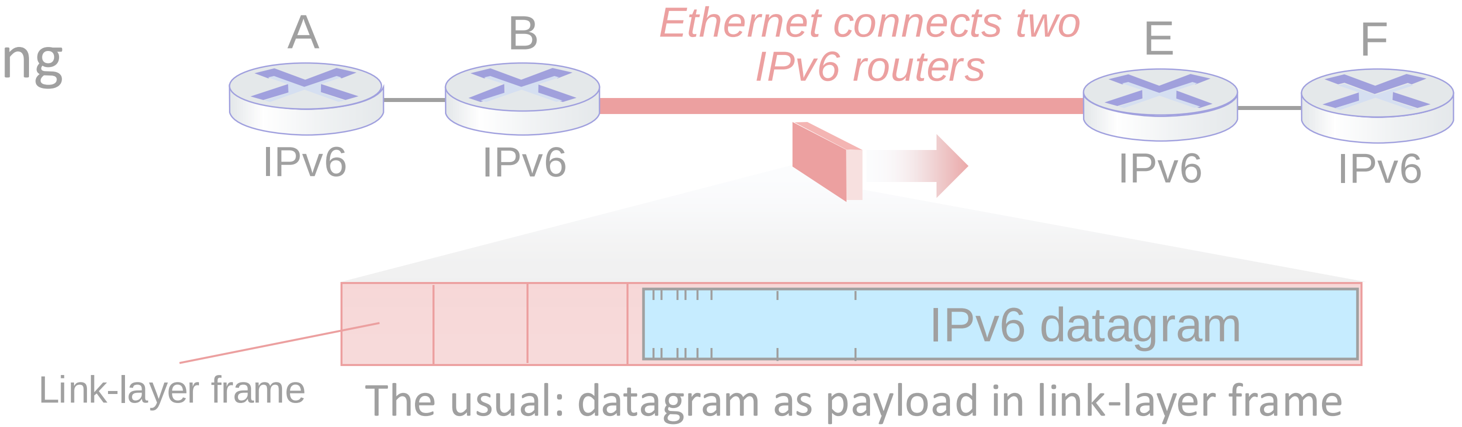



IPv4 network connecting two IPv6 routers



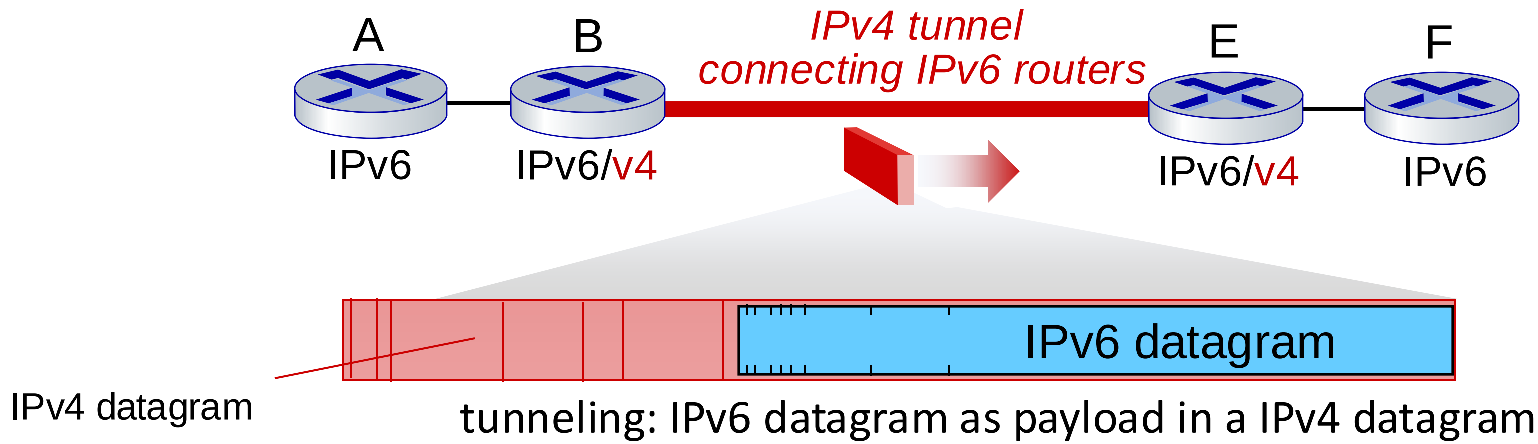
# Tunneling and encapsulation

Ethernet connecting two IPv6 routers:



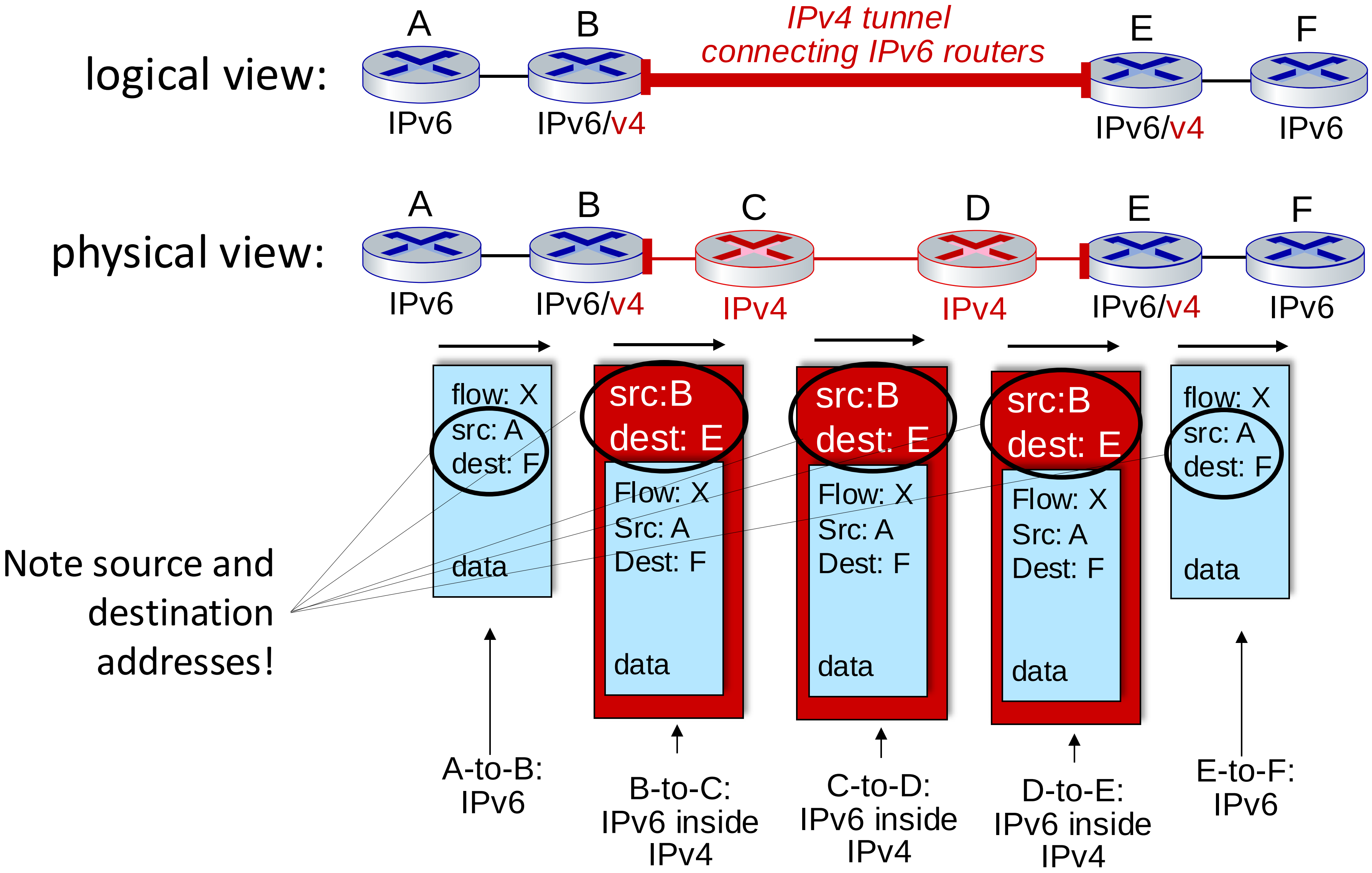

The usual: datagram as payload in link-layer frame

IPv4 tunnel connecting two IPv6 routers




tunneling: IPv6 datagram as payload in a IPv4 datagram

# Tunneling



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