# Network layer

- most complex layer
- host-to-host communication service

    (TL: process-to-process)
- data and control planes:

    intra (per router) functions,

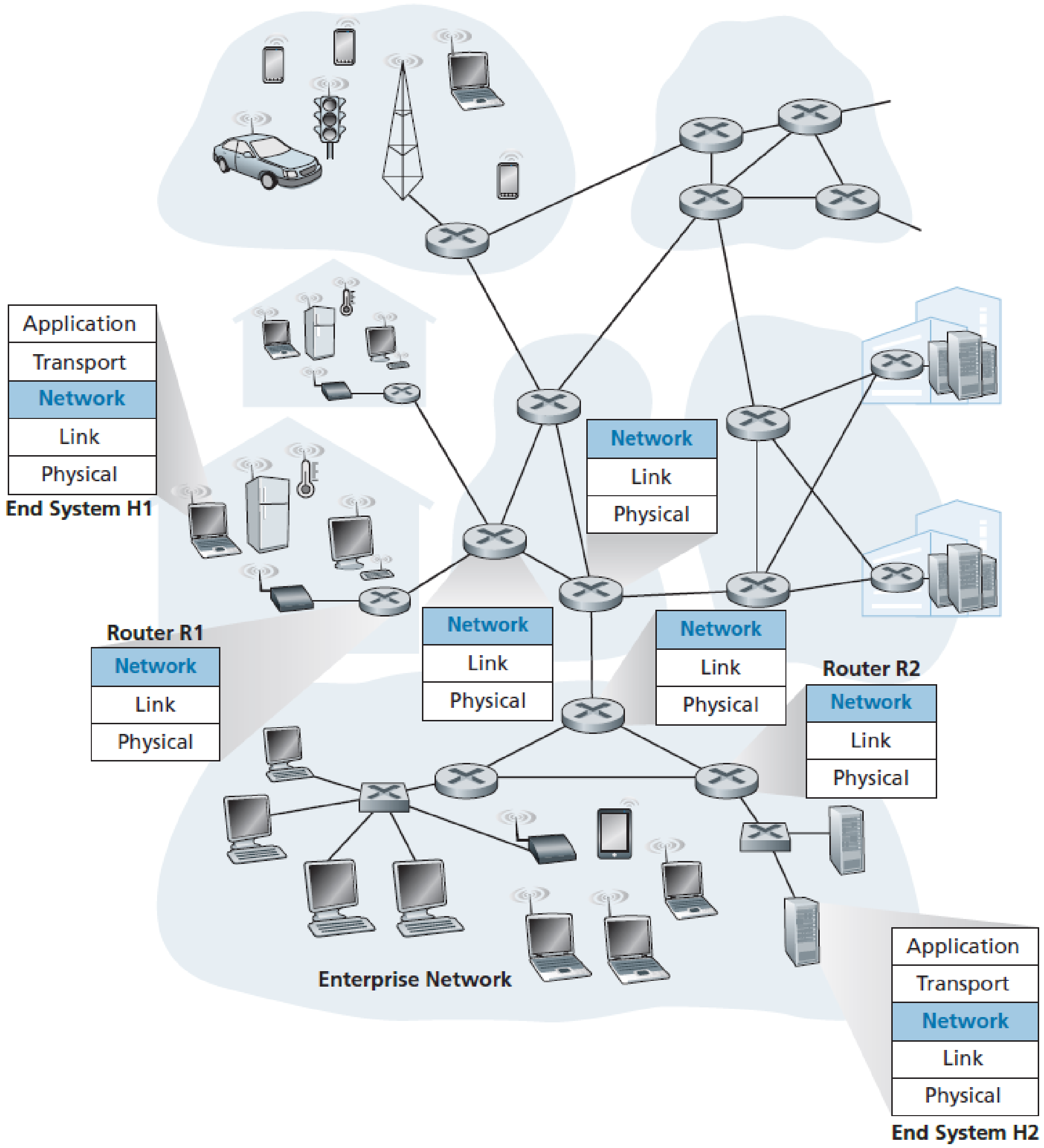
    inter (network wide) logic

The slides are based on the slides by Computer Networking: A Top-Down Approach 8th edition, Jim Kurose, Keith Ross, Pearson, 2020

# Network layer: "data plane" roadmap

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

# Network layer services and protocols




- sender: H1, receiver: H2
  - NL of H1: encapsulates segments into datagrams, passes to link layer
  - NL of H2: delivers segments to transport layer protocol
- network layer protocols run in *every Internet device*: hosts, routers
- routers:
  - examines header fields in all IP datagrams passing through it
  - moves datagrams from input ports to output ports (data plane role)
  - coordinate, per-router forwarding actions to provide transfers along paths of routers, between source and destination hosts (control plane role)

# Two key network layer functions

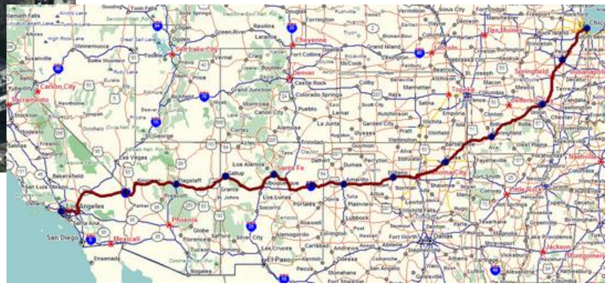
network-layer functions:

- *forwarding (data plane):* move packets from a router's input link to appropriate router output link
- *routing (control plane):* determine route taken by packets from source to destination
  - *routing algorithms: calculate the paths taken by packets to flow from a sender to a receiver*

analogy: taking a trip

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

forwarding

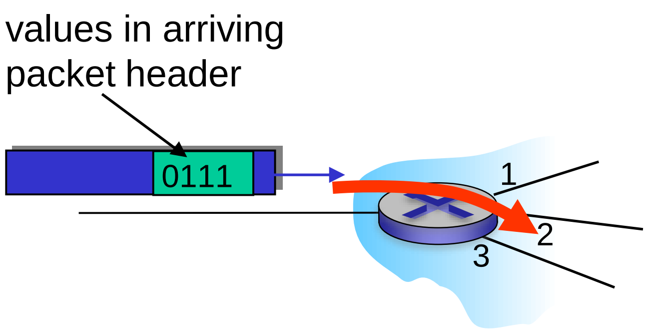


routing

# Network layer: data plane, control plane

## Data plane:

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

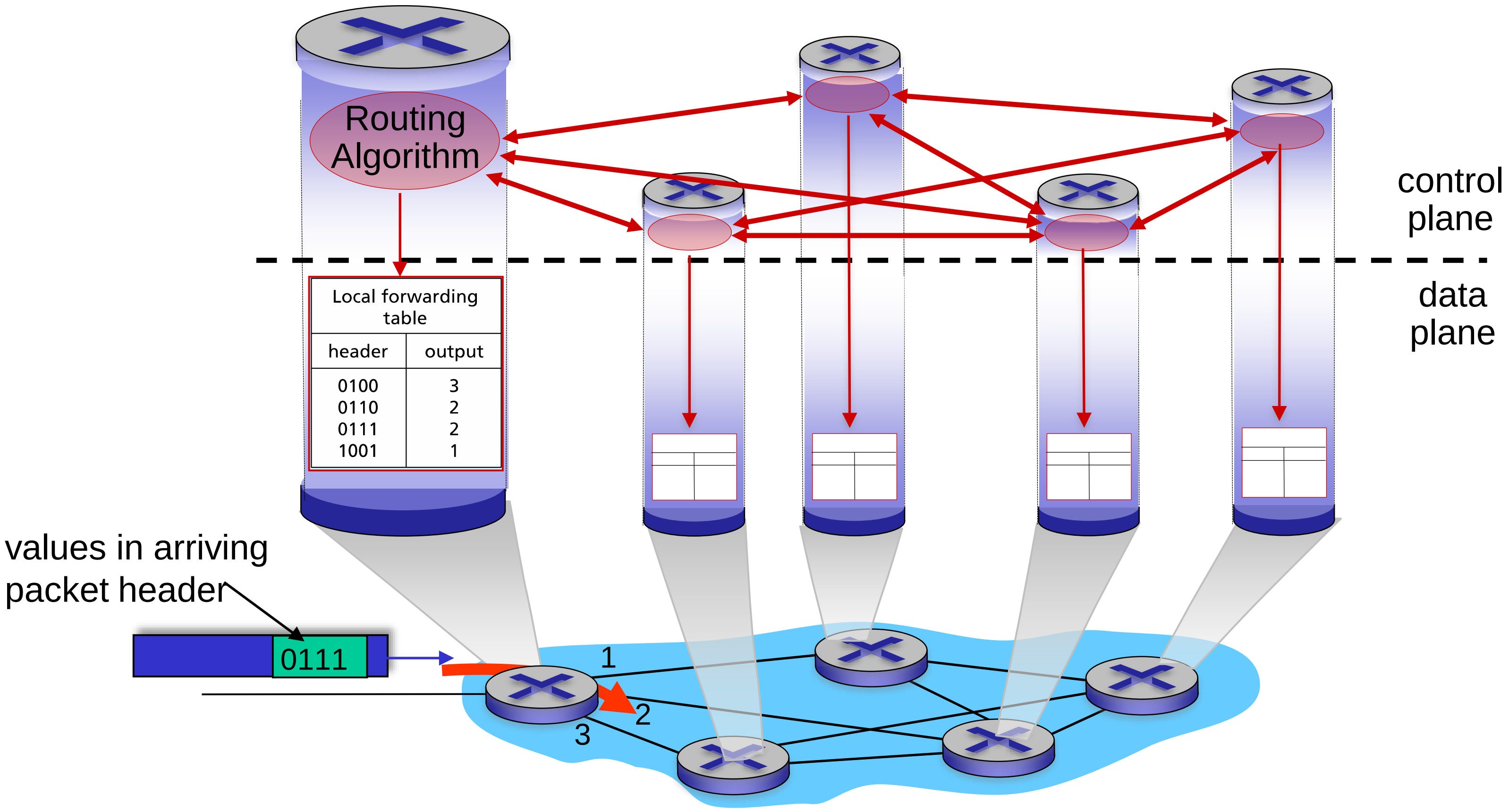



## Control plane

- *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

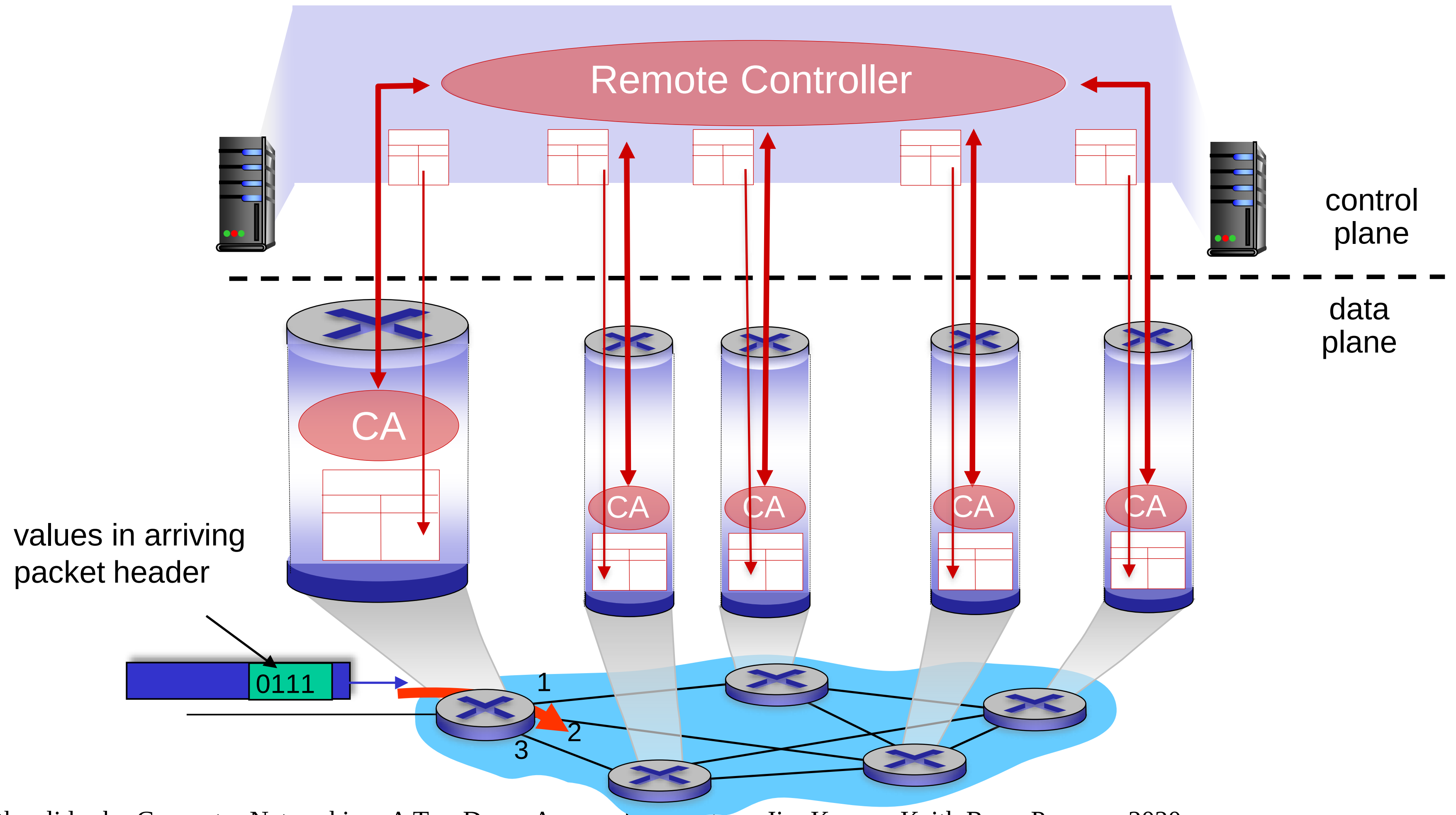
# Per-router control plane

Individual routing algorithm components *in each and every router* interact in the control plane (via the routing protocols)



# Software-Defined Networking (SDN) control plane

Routers performs forwarding only, remote controller computes and installs forwarding tables in routers (may be in a remote data center, may be managed by  ISP or 3 rd party)



# Network service model

Possible services that the network layer could provide:

example services for *individual* datagrams:

- guaranteed delivery: a packet sent by a source host will eventually arrive at the destination host
- guaranteed delivery with bounded delay: delivery within a specified host-to-host delay bound

example services for a *flow* of datagrams:

- in-order datagram delivery: arriving in the order that they were sent
- guaranteed minimum bandwidth to flow: emulates the behavior of a transmission link of a specified bit rate sending and receiving hosts
- security: encryption on network layer to provide confidentiality to all transport-layer segments

# Network service model

| Network Architecture | Service Model | Quality of Service (QoS) Guarantees ? | | | |
|---|---|---|---|---|---|
| | | Bandwidth | Loss | Order | Timing |
| Internet | best effort | none | no | no | no |
| ATM | Constant Bit Rate | Constant rate | yes | yes | yes |
| ATM | Available Bit Rate | Guaranteed min | no | yes | no |

## Internet "best effort" service model

*No* guarantees on:

i. successful datagram delivery to destination
ii. timing or order of delivery
iii. bandwidth available to end-end flow

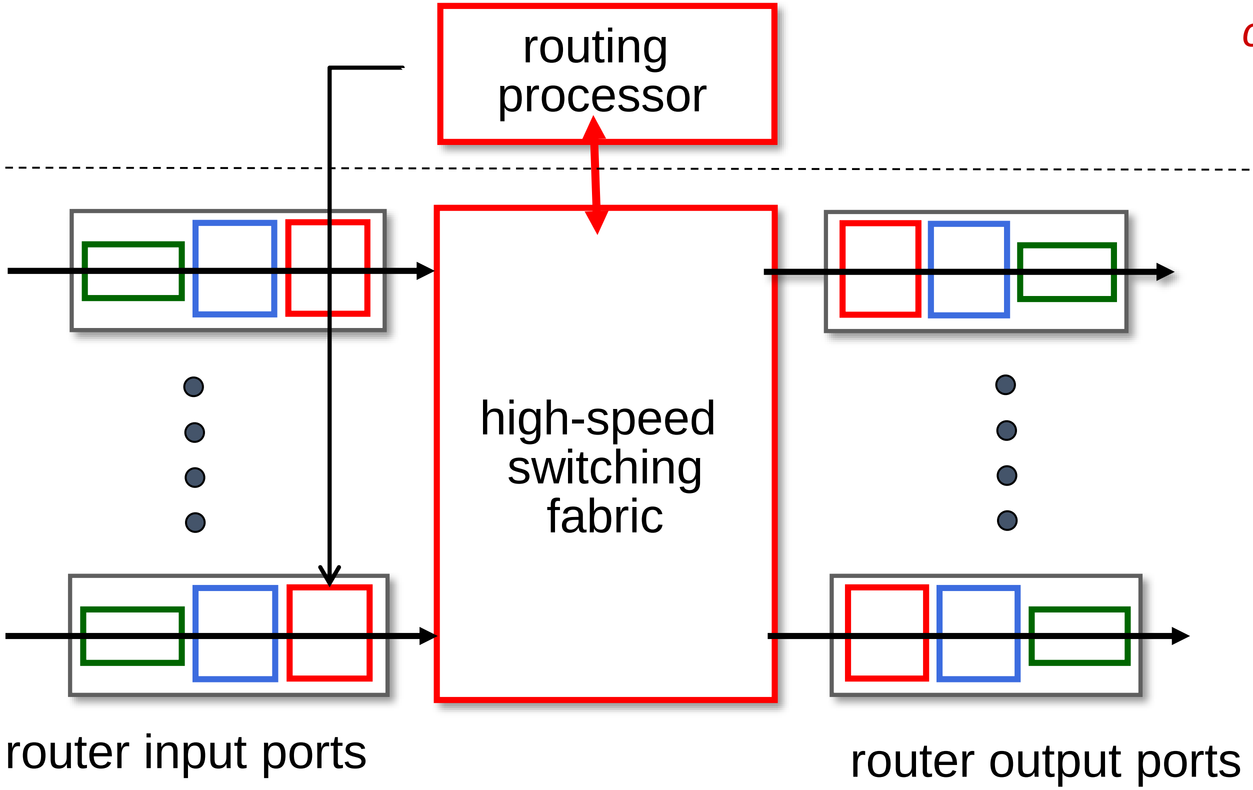
no service at all—a network that delivered no packets to the destination would satisfy the definition of best-effort delivery service

# Network layer: “data plane” roadmap

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

# Router architecture overview

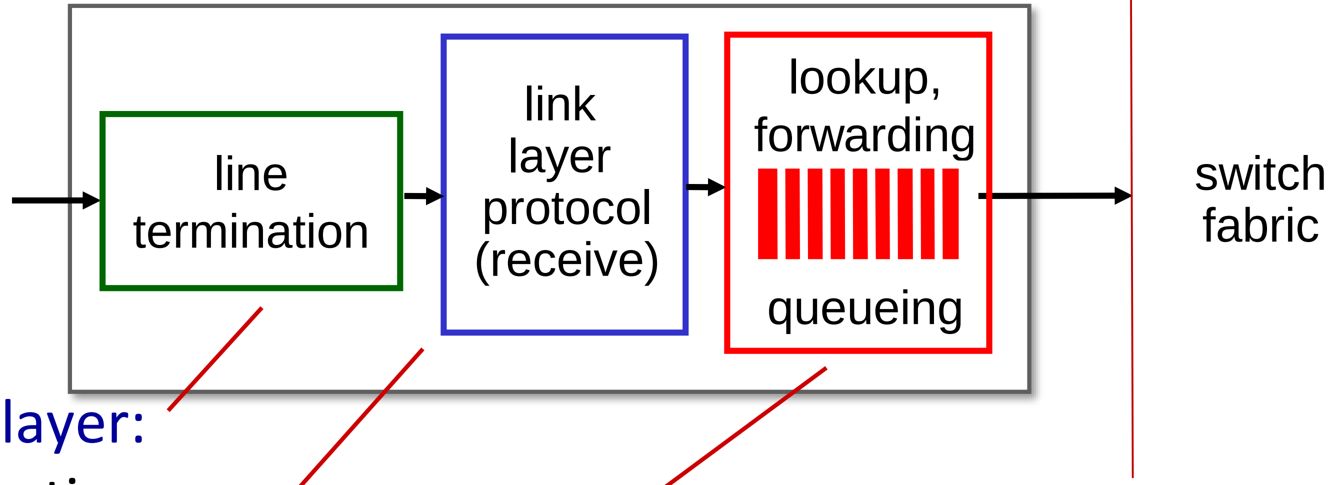
high-level view of generic router architecture:
4 components: input & output ports, routing processor, switching fabric




*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")*
- goal: complete input port processing at 'line speed'
- input port queuing: if datagrams arrive faster than forwarding rate into switch fabric

* physical layer function: terminating an incoming physical link at a router

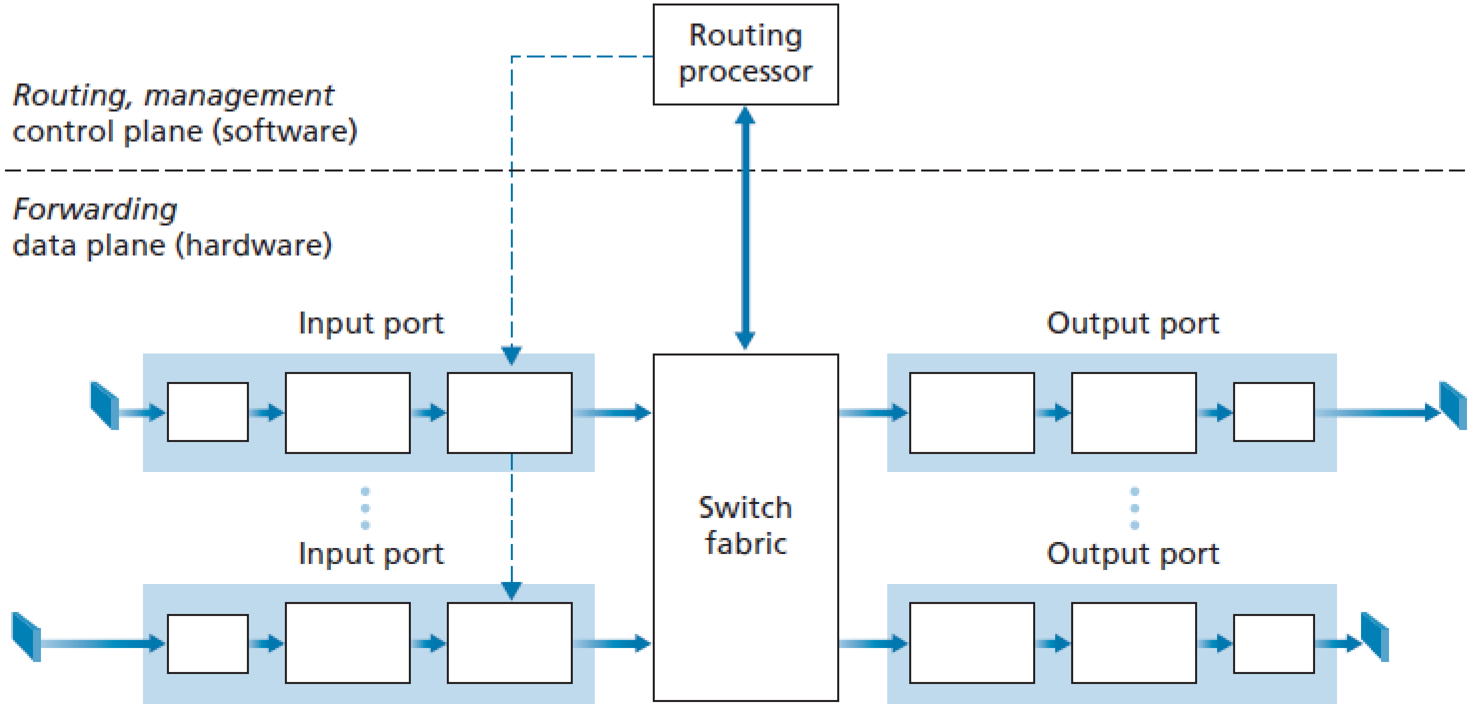
* link-layer functions: needed to interoperate with the link layer

* lookup function: forwarding table is consulted to determine output port (to which an arriving packet will be forwarded via the switching fabric).

Control packets (e.g. packets carrying routing protocol information) are forwarded from an input port to the routing processor.

*** port: physical, router interfaces, different from the software ports (network apps, sockets)

# Input port functions




* A router uses the forwarding table to look up the output port (to which an arriving packet will be forwarded via the switching fabric).

* The forwarding table can be computed and updated by the routing processor or can be received from a remote SDN controller.

* The forwarding table is copied from the routing processor to the line cards over a separate bus (dashed line from the routing processor to the input line cards).

*So, forwarding decisions can be made locally, at each input port.*

Destination based forwarding: most common type, all forwarding decisions are made based on dest IP address (independent of the original sender).

Generalized forwarding: a router forwards based on header field values.

# Longest prefix matching

## longest prefix match

when looking for forwarding table entry for given destination address, use *longest* address prefix that matches destination address.

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

examples:

11001000 00010111 00010110 10100001 which interface?

11001000 00010111 00011000 10101010 which interface?

# Longest prefix matching

## longest prefix match

when looking for forwarding table entry for given destination address, use *longest* address prefix that matches destination address.

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

match!

examples:

11001000 00010111 00010110 10100001 which interface?

11001000 00010111 00011000 10101010 which interface?

# Longest prefix matching

## longest prefix match

when looking for forwarding table entry for given destination address, use *longest* address prefix that matches destination address.

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

match!

examples:

11001000 00010111 00010110 10100001 which interface?

11001000 00010111 00011000 10101010 which interface?

# Longest prefix matching

**longest prefix match**

When there are multiple matches, the router uses the longest prefix matching rule: finds the longest matching entry in the table and forwards the packet to the link interface associated with the longest prefix match.

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

match!

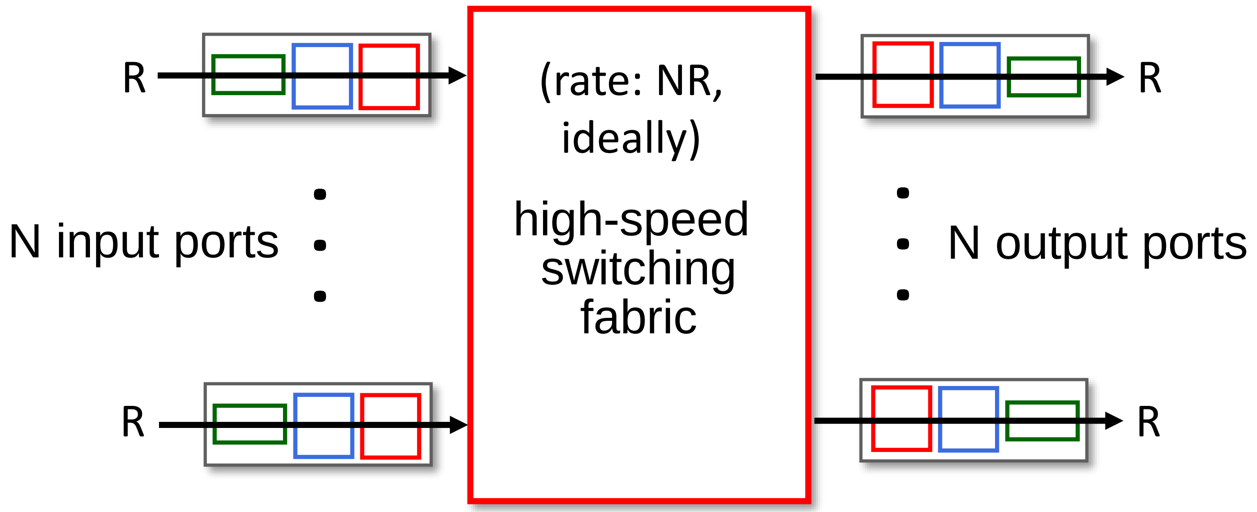
examples:

11001000 00010111 00010110 10100001 which interface?

11001000 00010111 00011000 10101010 which interface?

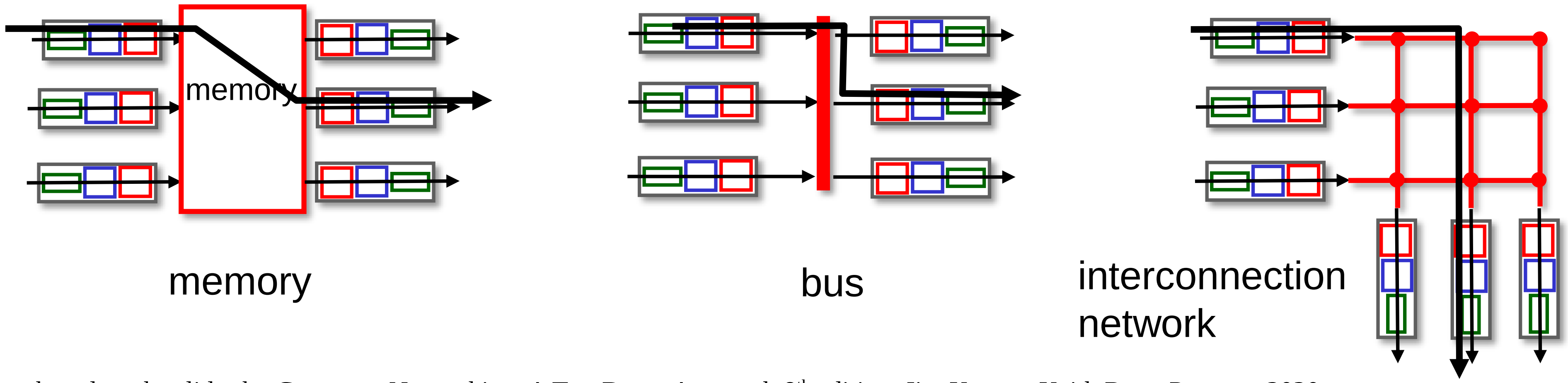
# Switching fabrics

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



# Switching fabrics

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



# Switching via memory

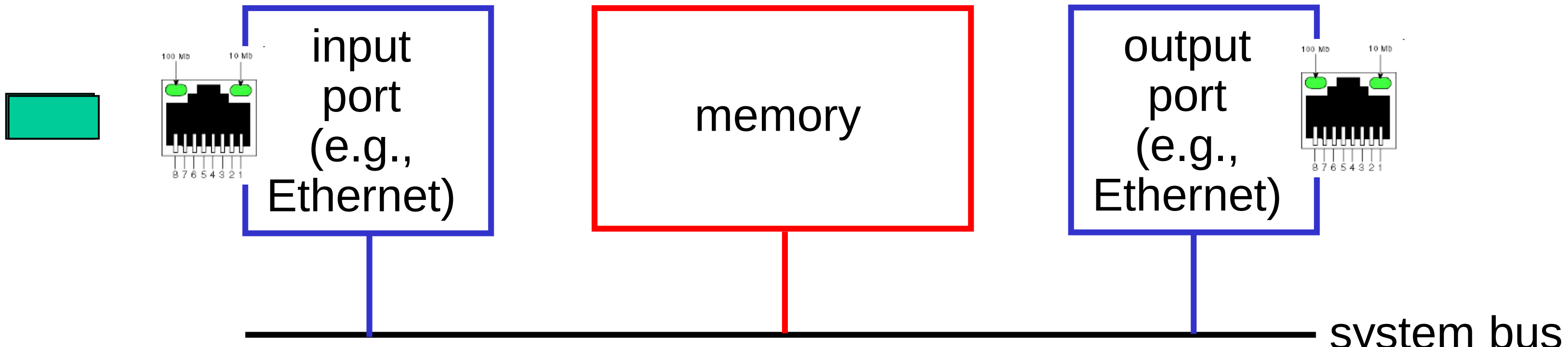
first generation routers:

- switching between input and output ports being done under direct control of the CPU (routing processor)
- process:

    an input port with an arriving packet first signals the routing processor via an interrupt

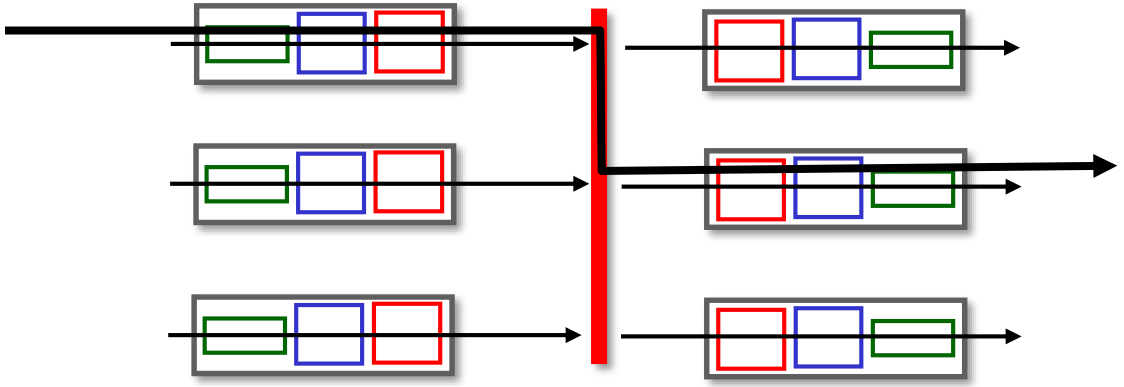
    the packet is copied from the input port into processor memory

    routing processor extracts the destination address from the header, finds the appropriate output port in the forwarding table, and copies the packet to the output port's buffers.



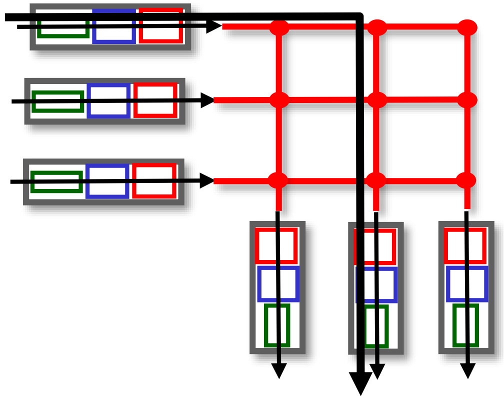
# Switching via a bus

- an input port transfers a packet directly to the output port over a shared bus (without intervention by the routing processor)
- *bus contention:* switching speed limited by bus bandwidth

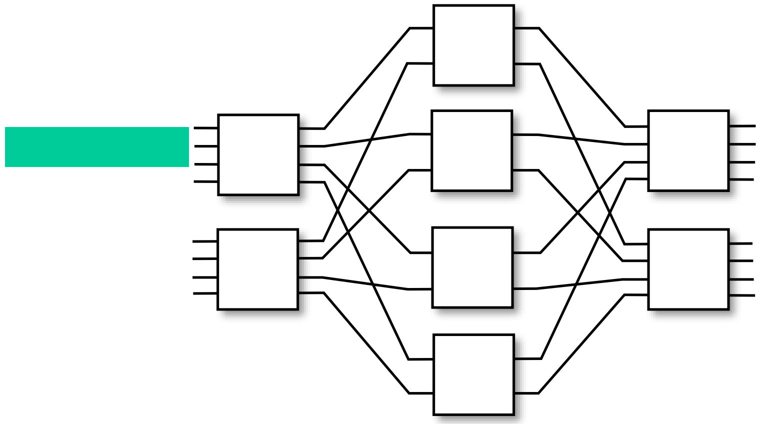


# Switching via interconnection network

- to overcome the bandwidth limitation of a single, shared bus
- uses interconnection network,
- consists of 2N buses that connect N input ports to N output ports



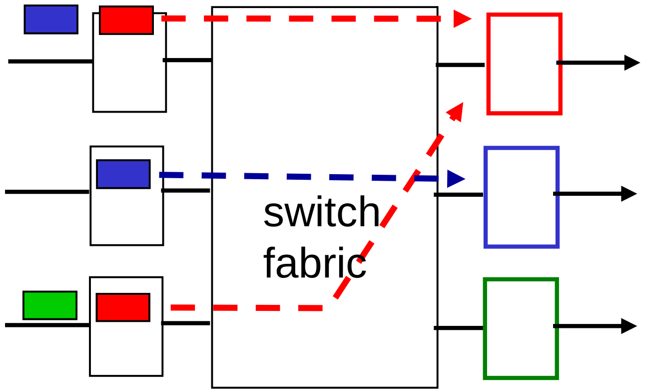
3x3 crossbar



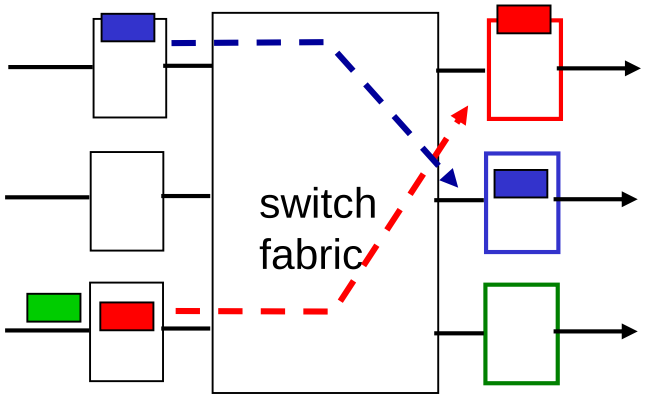
8x8 multistage switch
built from smaller-sized switches

# Input port queuing

- switch fabric slower than input ports combined -> queueing 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

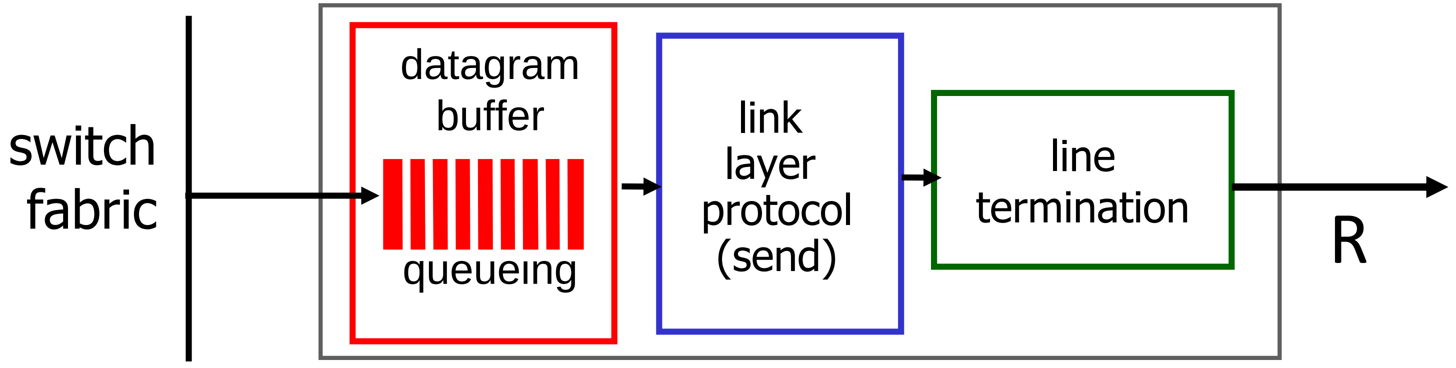



output port contention: only one red datagram can be transferred. lower red packet is *blocked*



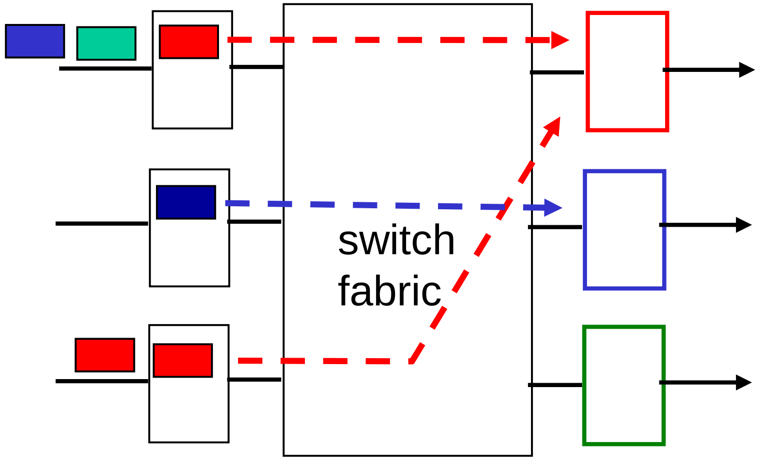

one packet time later: green packet experiences HOL blocking

# Output port queuing

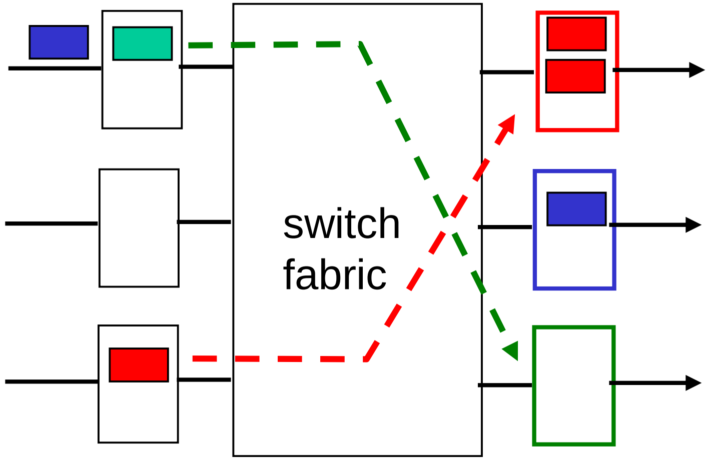



- *Buffering* required when datagrams arrive from fabric faster than outgoing link transmission rate.
- *Drop policy:* which datagrams to drop if no free buffers?

# Output port queuing




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*
- a packet scheduler at the output port must choose one packet, among those queued, for transmission (one of the three red packets)

# Packet scheduling: FCFS

packet scheduling: deciding which packet to send next on link

- first come, first served
- priority
- round robin
- weighted fair queueing

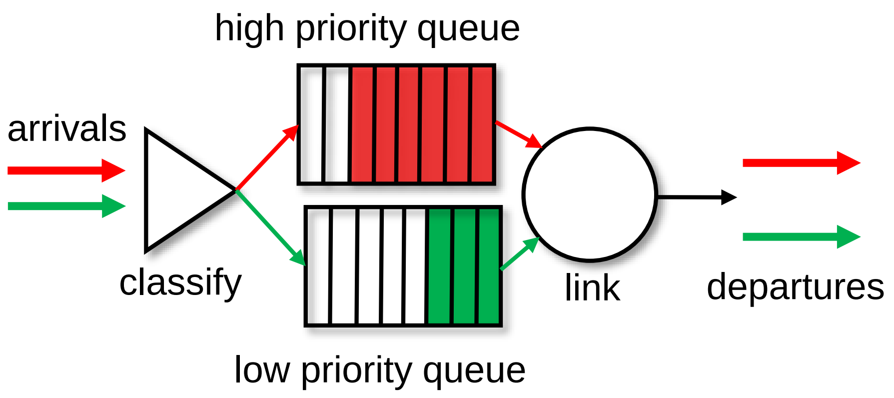
FCFS: packets transmitted in order of arrival to output port

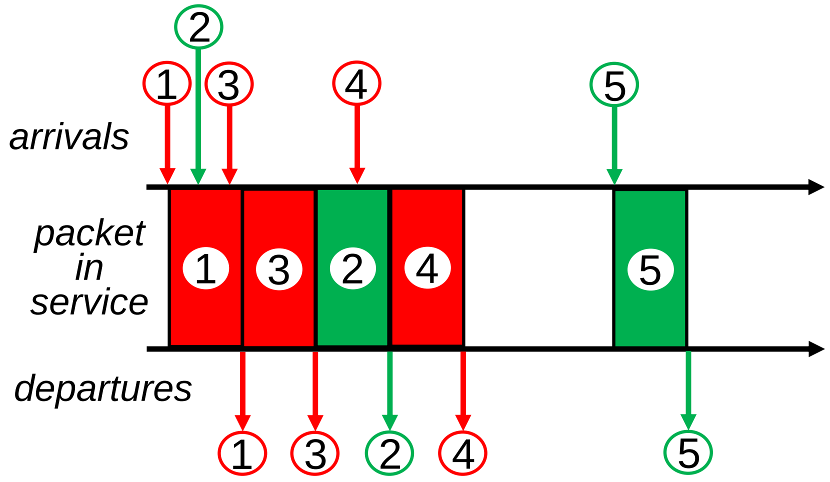
- also known as: First-in-first-out (FIFO)

# Scheduling policies: priority

*Priority scheduling:*

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

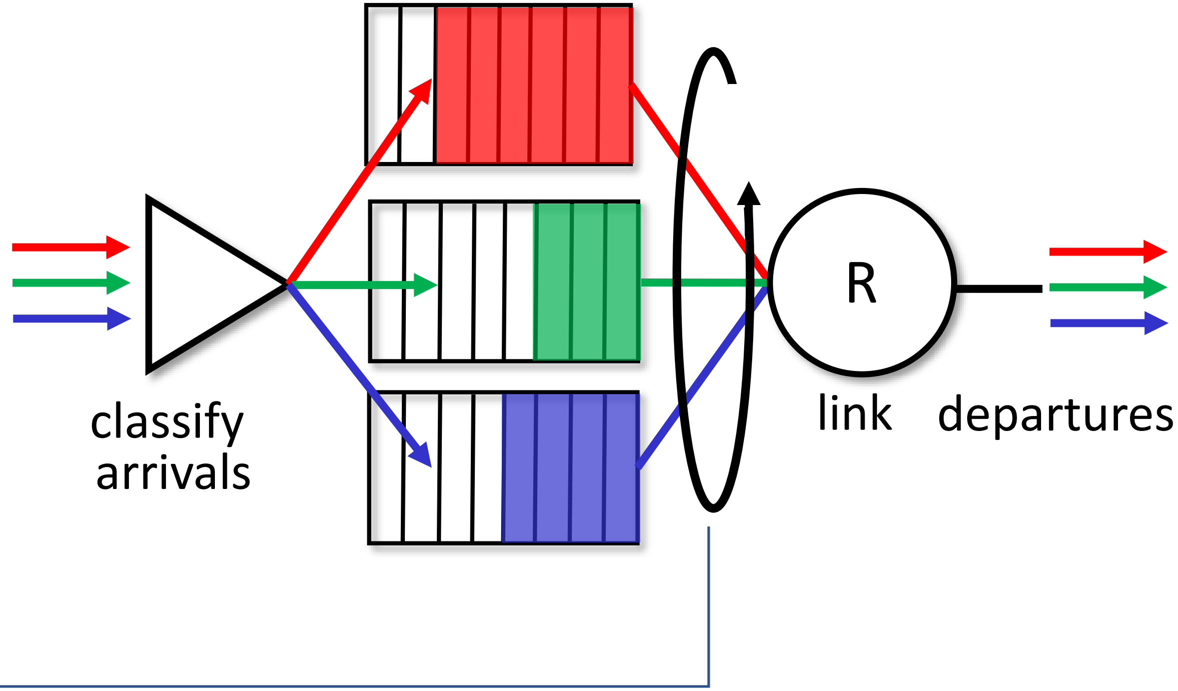


# Scheduling policies: round robin

*Round Robin (RR) scheduling:*

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



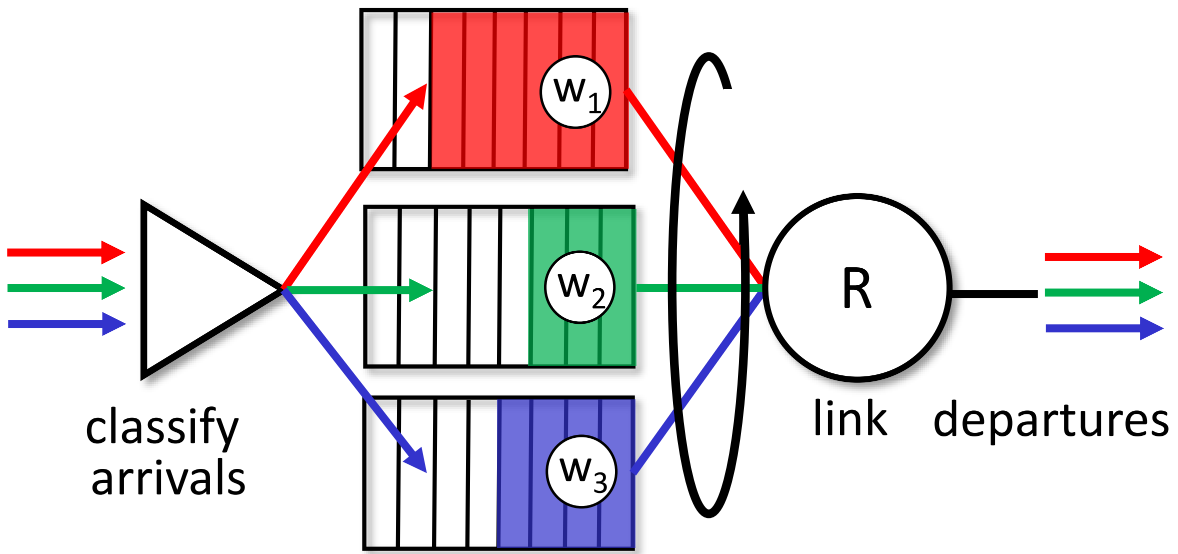
# Scheduling policies: weighted fair queueing

*Weighted Fair Queuing (WFQ):*

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

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

- minimum bandwidth guarantee, a throughput of at least R $\frac{w_i}{\Sigma_j w_j}$



# Router architecture summary

4 components: input & output ports, routing processor, switching fabric

* Input ports: physical layer function (terminating an incoming physical link), link-layer functions (needed to interoperate with the link layer) lookup function (determining output port)
* Switching fabric: connects the router’s input ports to its output ports
* Output ports: store packets received from the switching fabric and transmit these packets on the outgoing link by performing the link-layer and physical-layer functions
* Routing processor: performs control-plane functions.

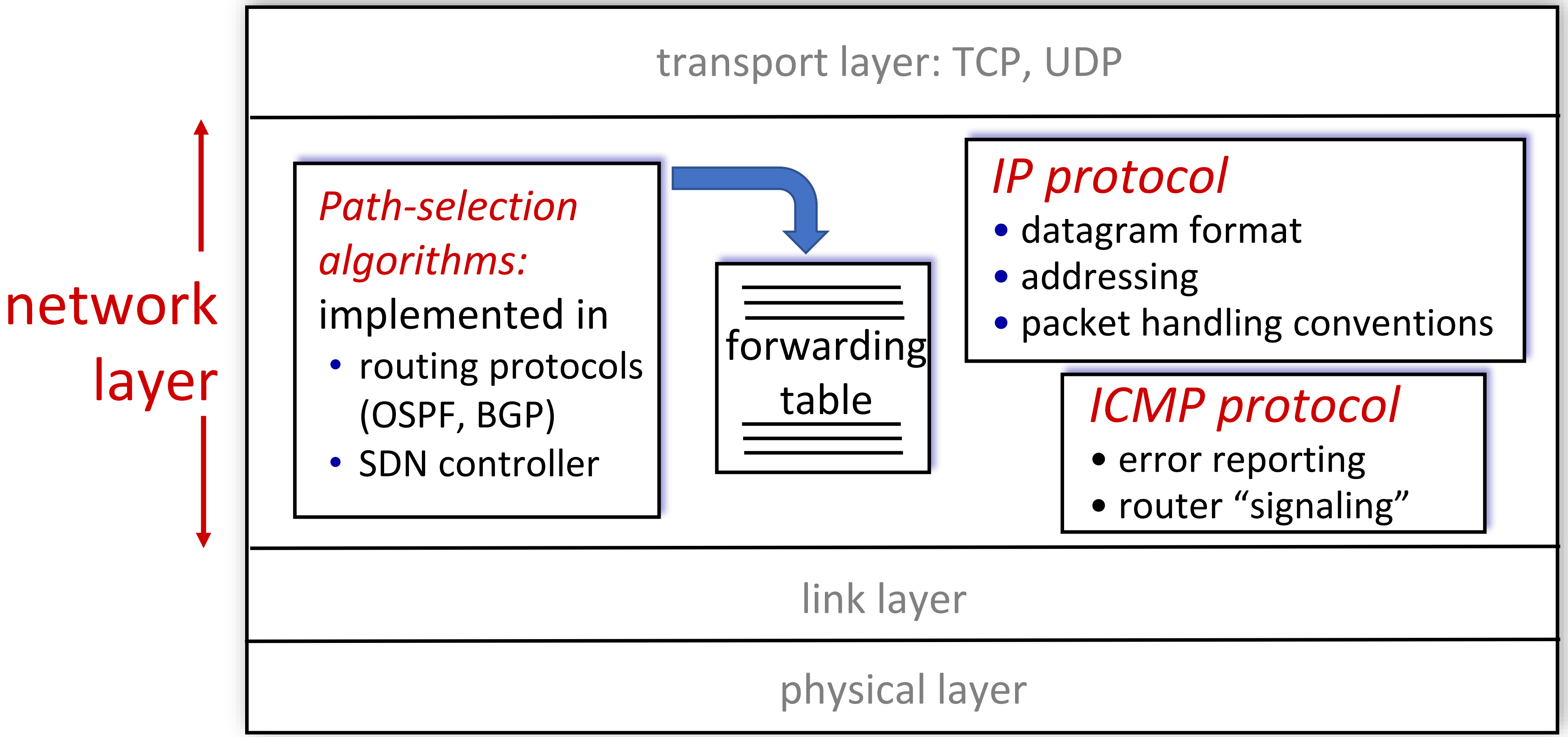
<u>In traditional routers</u>, it executes the routing protocols, maintains routing tables and attached link state information, and computes the forwarding table for the router.
<u>In SDN routers</u>, it is responsible for communicating with the remote controller in order to receive forwarding table entries and install these entries in the router’s input ports.

# Network layer: “data plane” roadmap

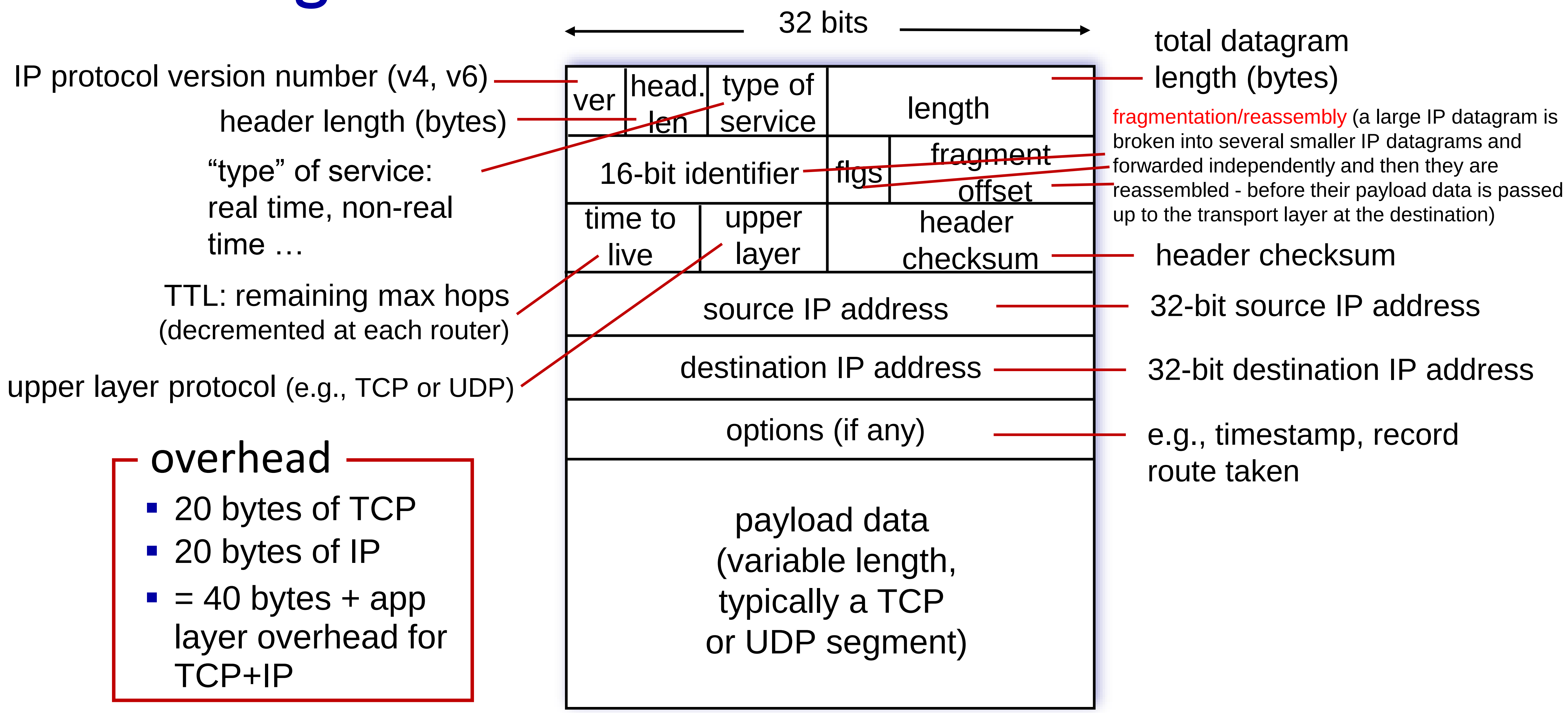
- Network layer: overview
  - data plane
  - control plane
- What’s inside a router
  - input ports, switching, output ports
  - buffer management, scheduling
- IP: the Internet Protocol
  - datagram format
  - addressing
  - network address translation
  - IPv6
- Generalized Forwarding, SDN
  - Match+action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

# Network layer: Internet

host, router network layer functions:



# IP Datagram format

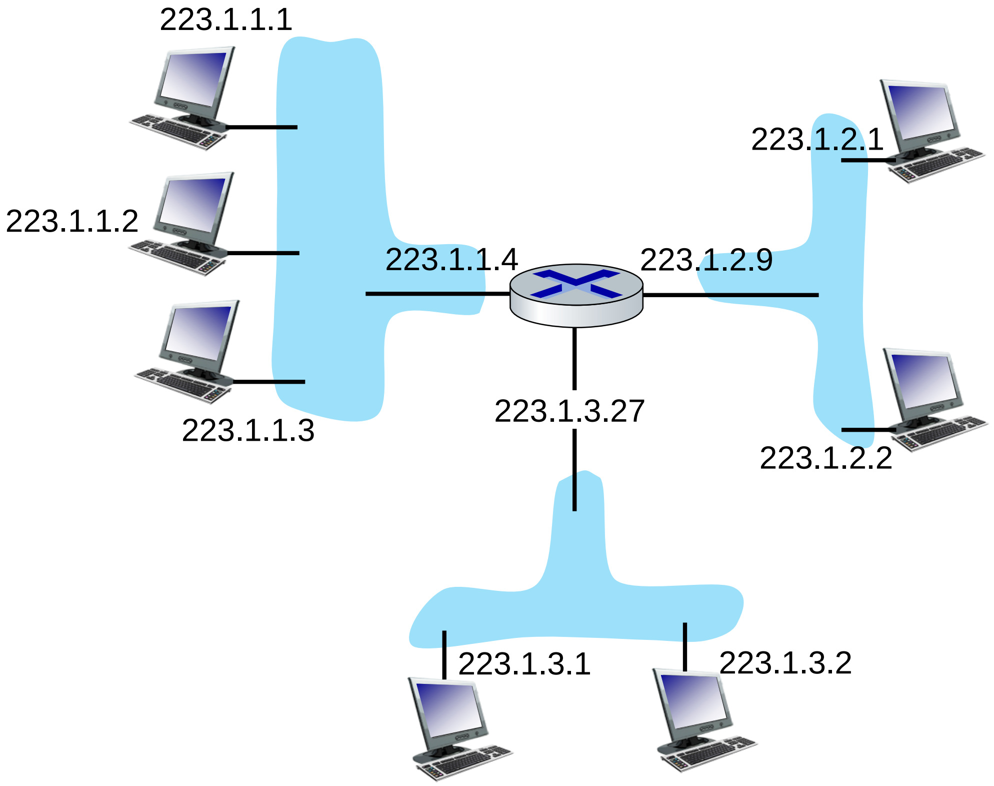



overhead

- 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* (rather than with the host or router containing that interface)
- interface: connection between host/router and physical link
  - routers typically have multiple interfaces
  - hosts typically have one or two interfaces (e.g., wired Ethernet, wireless 802.11)




dotted-decimal IP address notation:

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

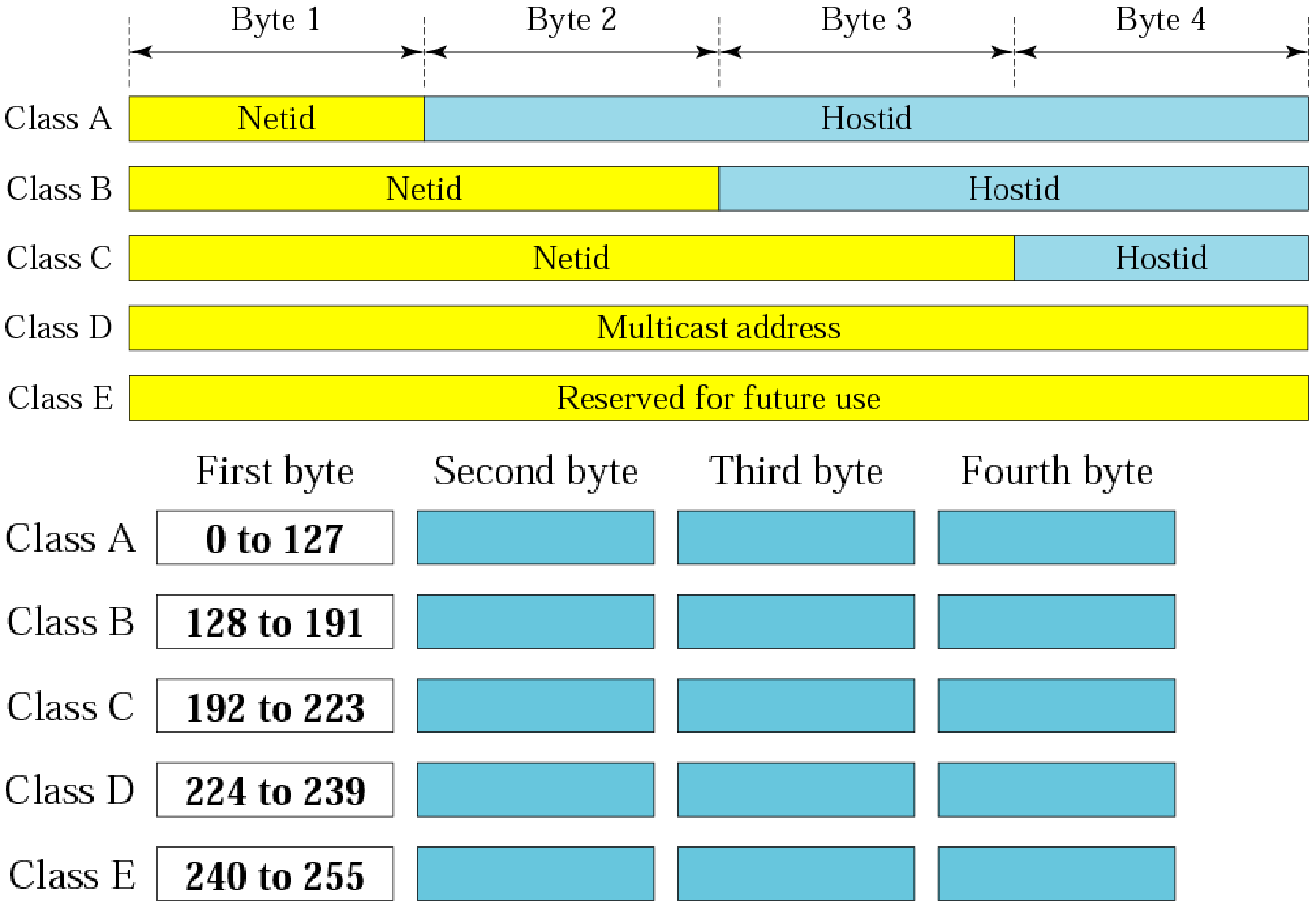
# IP addressing: introduction

| | First byte | Second byte | Third byte | Fourth byte |
|---|---|---|---|---|
| Class A | 0 | | | |
| Class B | 10 | | | |
| Class C | 110 | | | |
| Class D | 1110 | | | |
| Class E | 1111 | | | |

| | First byte | Second byte | Third byte | Fourth byte |
|---|---|---|---|---|
| Class A | 0 to 127 | | | |
| Class B | 128 to 191 | | | |
| Class C | 192 to 223 | | | |
| Class D | 224 to 239 | | | |
| Class E | 240 to 255 | | | |

- IPv4 address space is divided into 5 classes: A, B, C, and E
- Binary & decimal forms

# IP addressing: introduction




- the first address: network address
- defines the network to the rest of the Internet
- Given the network address, class of the address, the block, and the range of the addresses in the block can be found

# IP addressing: introduction

| Address Start Range | Address End Range | What Range is Used for |
|---|---|---|
| 0.0.0.0 | 0.255.255.255 | Reserved |
| 10.0.0.0 | 10.255.255.255 | Class A Private Address Block |
| 127.0.0.0 | 127.255.255.255 | Loopback Address Block |
| 128.0.0.0 | 128.0.255.255 | Reserved |
| 169.254.0.0 | 169.254.255.255 | Private Address Block Reserved for APIPA |
| 172.16.0.0 | 173.31.255.255 | Class B Private Address Block |
| 191.255.0.0 | 191.255.255.255 | Reserved |
| 192.0.0.0 | 192.0.0.255 | Reserved |
| 192.168.0.0 | 192.168.255.255 | Class C Private Address Block |
| 223.255.255.0 | 223.255.255.255 | Reserved |

- Special purpose, reserved IP addresses
- APIPA - Automatic Private IP Addressing
- With APIPA, DHCP clients can automatically self-configure an IP address and subnet mask when a DHCP server isn't available
- When a DHCP client boots up, it first looks for a DHCP server in order to obtain an IP address and subnet mask
- If the client is unable to find the information, it uses APIPA to automatically configure itself with an IP address

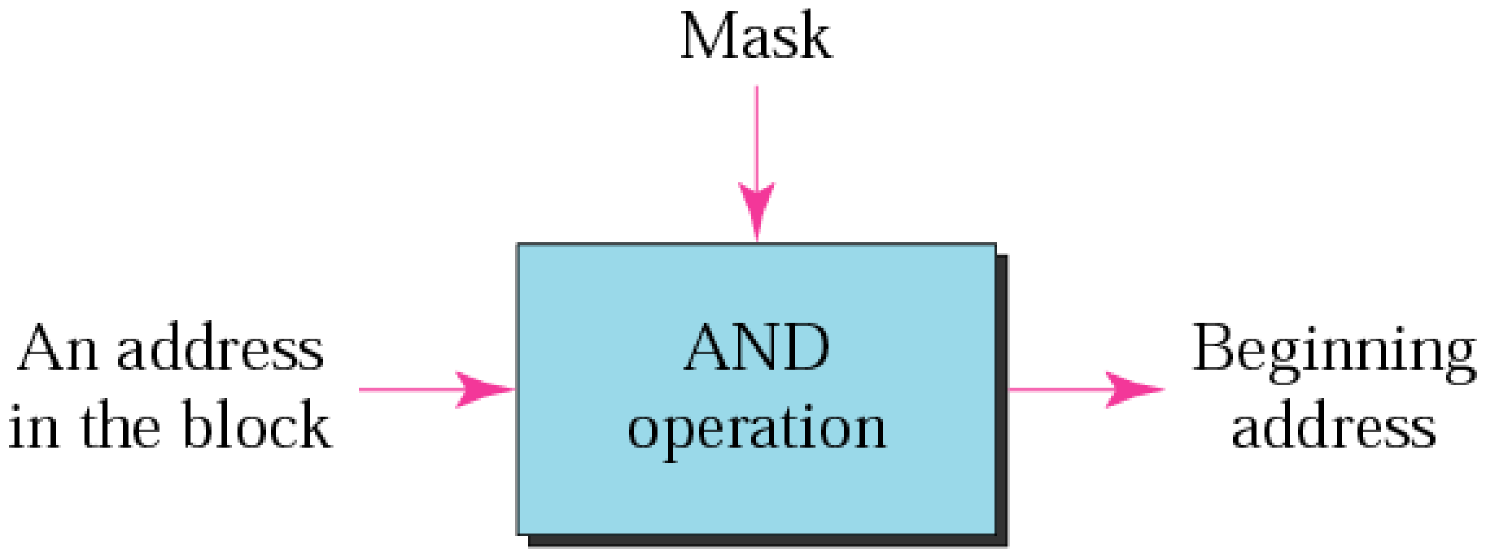
# IP addressing: introduction

**Given the network address 132.21.0.0, find the class, the block, and the range of the addresses.**

- The 1st byte is between 128 and 191, Hence Class B
- The block has a netid of 132.21.
- The addresses range

from 132.21.0.0 to 132.21.255.255

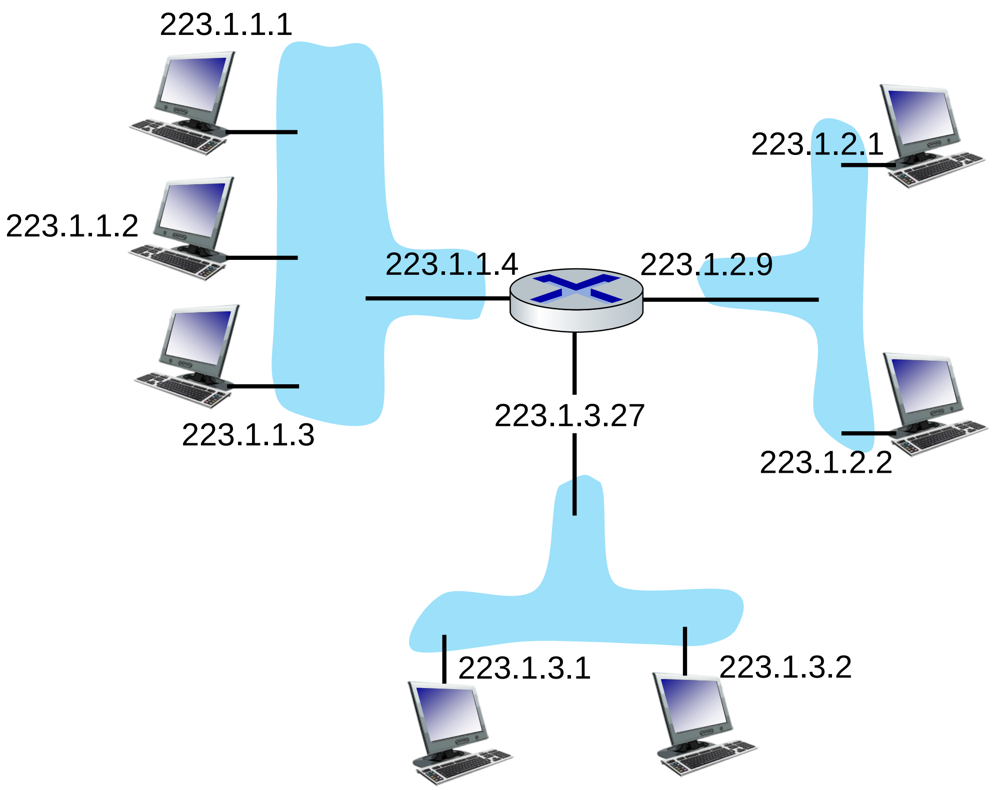
# IP addressing: introduction




- **A mask is a 32-bit binary number**
- **The mask is ANDed with IP address to get the block address (network address)**
- **Class A default mask is 255.0.0.0**
- **Class B default mask is 255.255.0.0**
- **Class C Default mask 255.255.255.0**

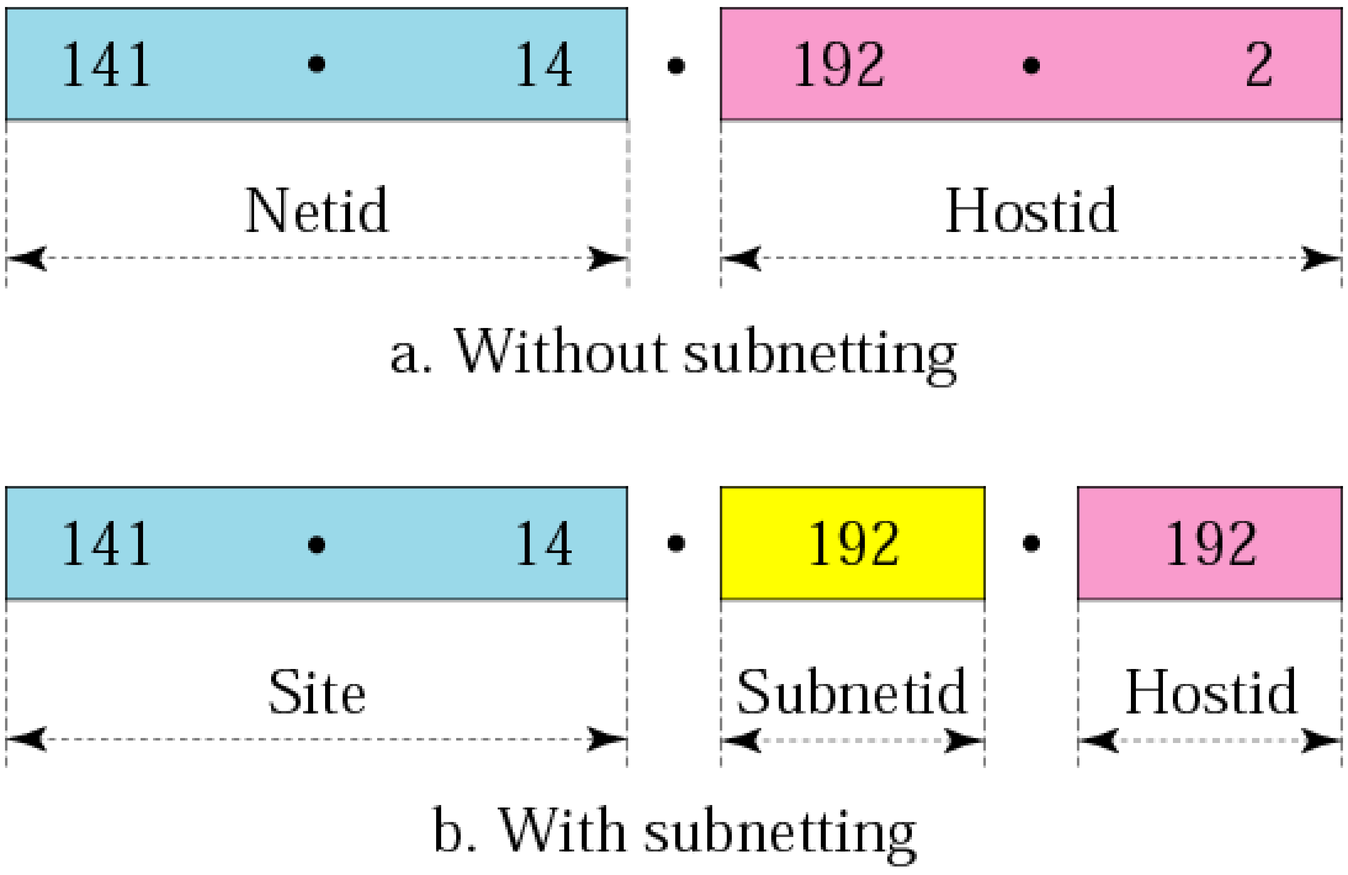
# Subnets

- *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
  - Subnetting is done by borrowing bits from the host part and add them the network part

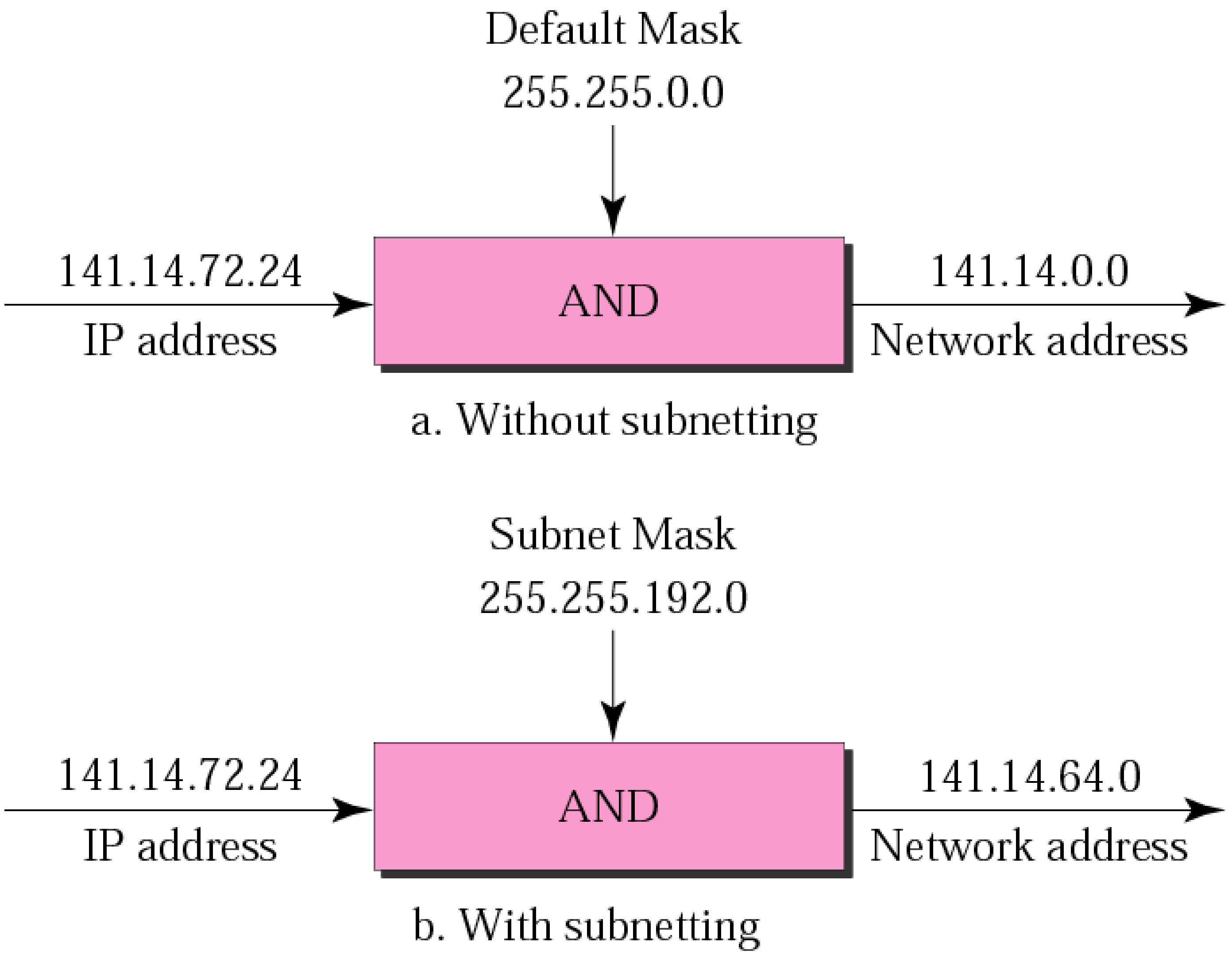



network consisting of 3 subnets

# Subnets


141 • 14 • 192 • 2
Netid
Hostid
a. Without subnetting
141 • 14 • 192 • 192
Site
Subnetid
Hostid
b. With subnetting

# Subnets

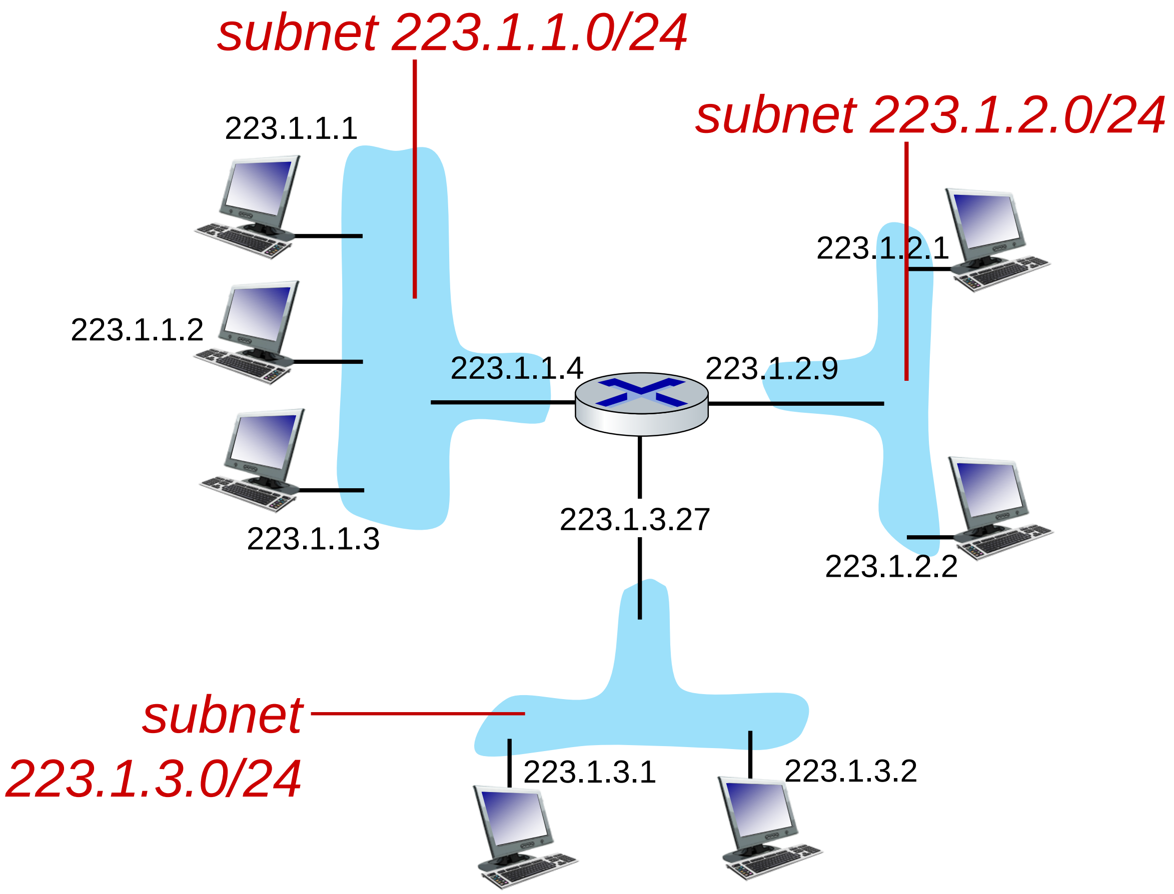


Default Mask
255.255.0.0
141.14.72.24
IP address
AND
141.14.0.0
Network address
a. Without subnetting
Subnet Mask
255.255.192.0
141.14.72.24
IP address
AND
141.14.64.0
Network address
b. With subnetting

# Subnets

*Recipe for defining subnets:*

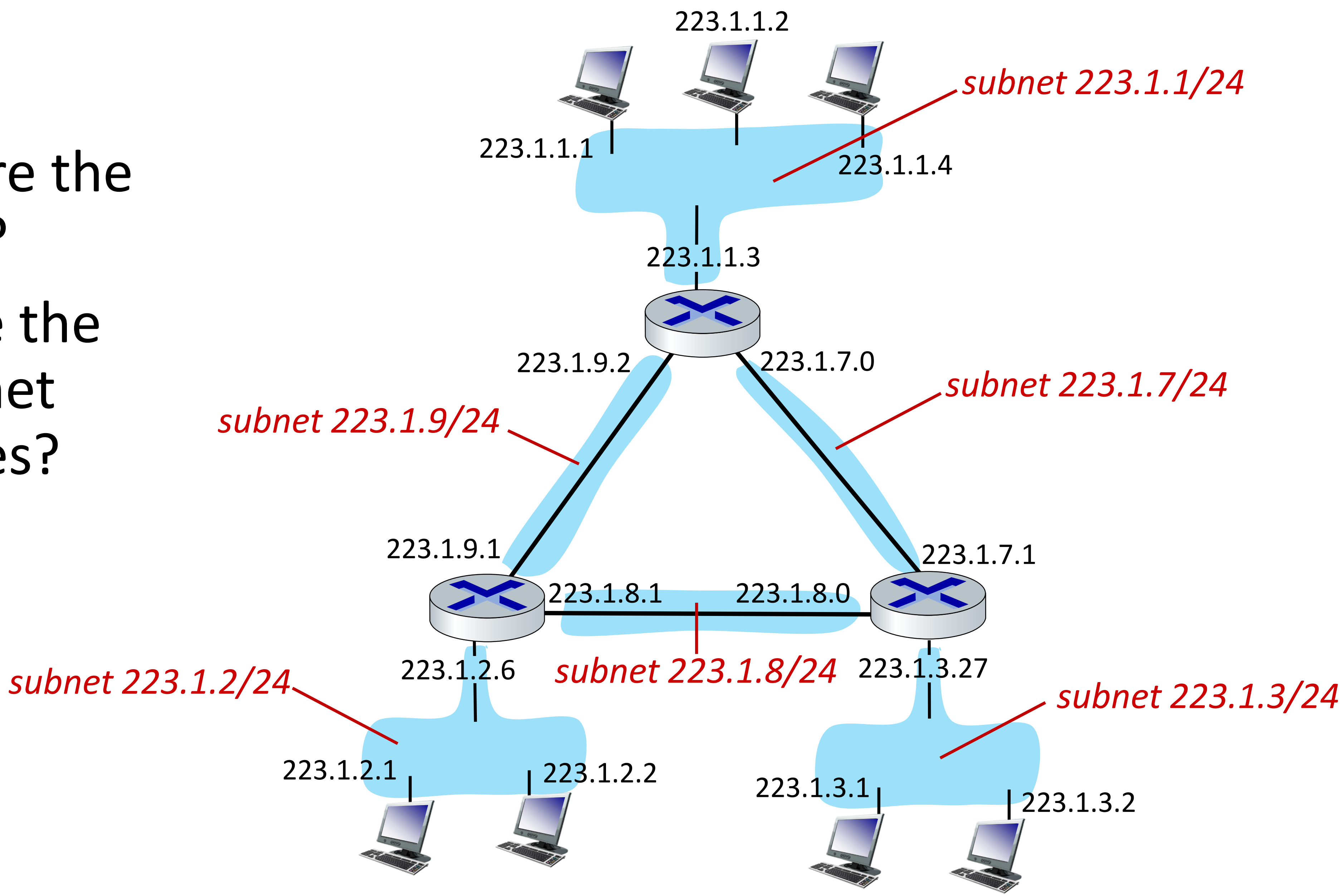
- detach each interface from its host or router, creating "islands" of isolated networks
- each isolated network is called a *subnet*




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

# Subnets

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



# IP addressing: CIDR

CIDR: Classless InterDomain Routing (pronounced "cider"), slash notation

- 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?

Two questions:

1. Q: How does a *host* get IP address within its network (host part of address)?
2. Q: How does a *network* get IP address for itself (network part of address)

How does *host* get IP address?

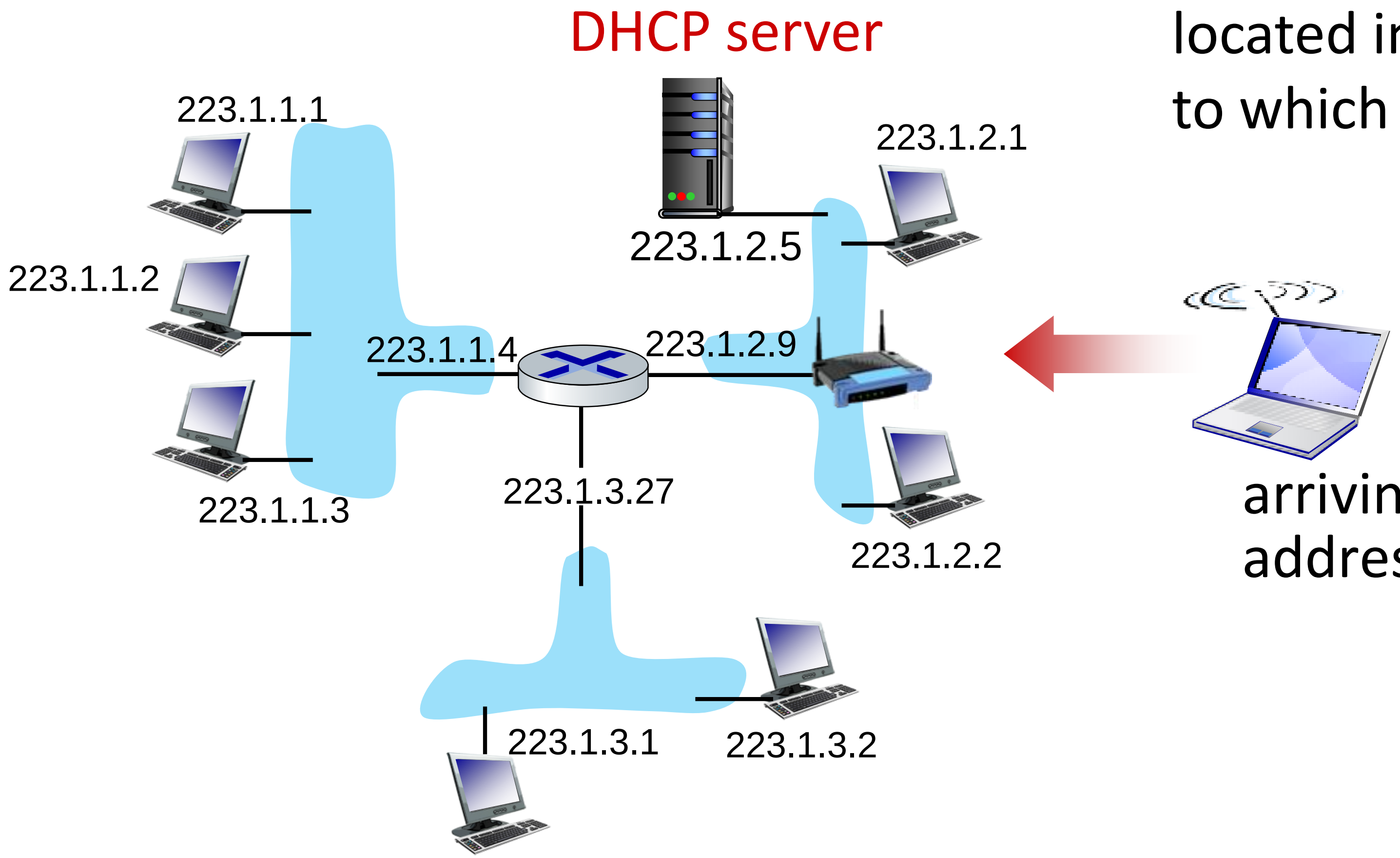
- hard-coded by sysadmin in config file (e.g., /etc/rc.config in UNIX)
- DHCP: Dynamic Host Configuration Protocol: dynamically get address from as server
  - "plug-and-play"

# DHCP: Dynamic Host Configuration Protocol

goal: host *dynamically* obtains IP address from network server when it "joins" network
- can renew its lease on address in use
- allows reuse of addresses (only hold address while connected/on)
- support for mobile users who join/leave network

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

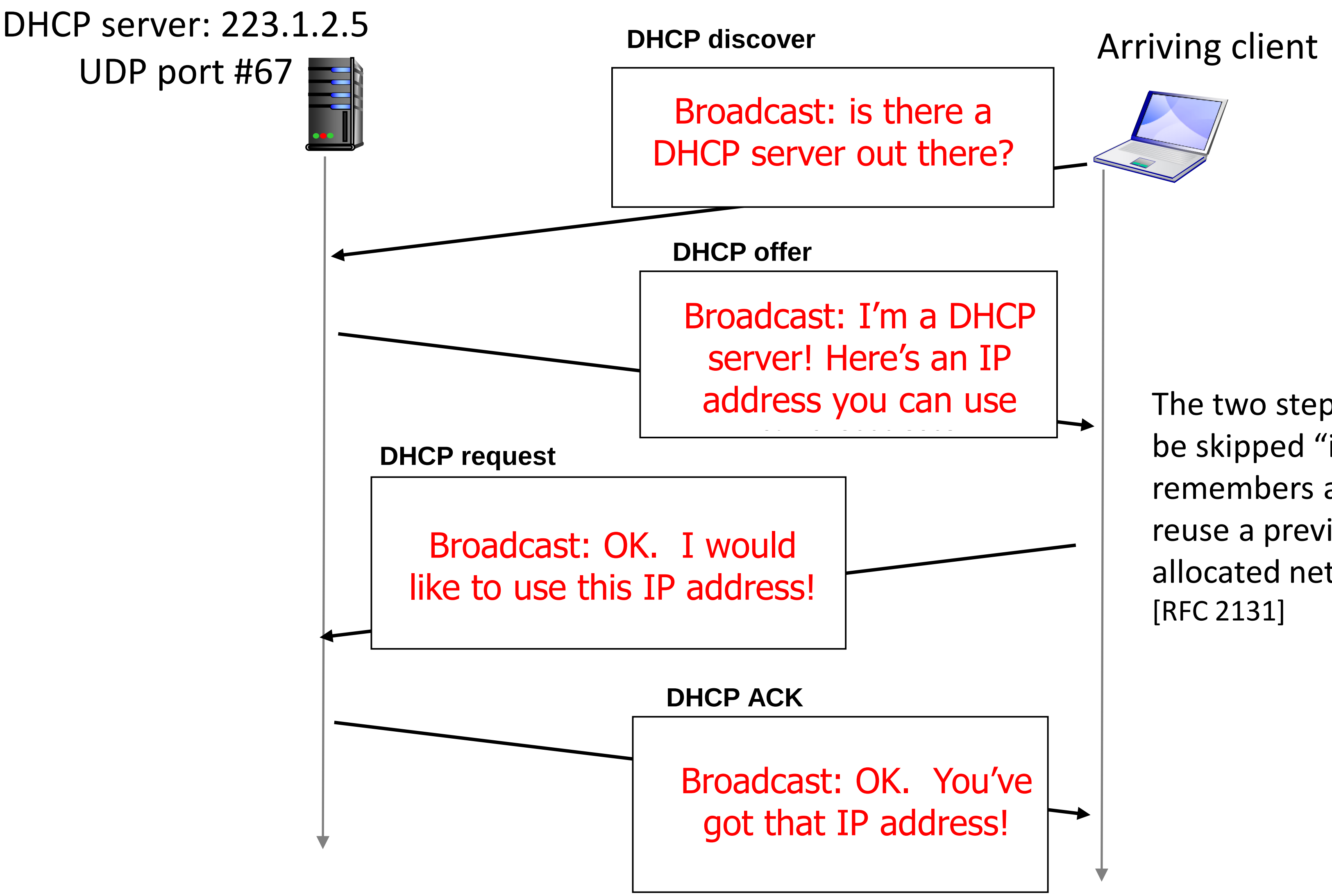
# DHCP client-server scenario




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

arriving DHCP client needs address in this network

# DHCP client-server scenario




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

# DHCP: more than IP addresses

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

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

# IP addresses: how to get one?

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

*A:* to obtain a block of IP addresses for use within an organization's subnet, a network administrator might first contact its ISP, which would provide addresses from a larger block of addresses that had already been allocated to the ISP.

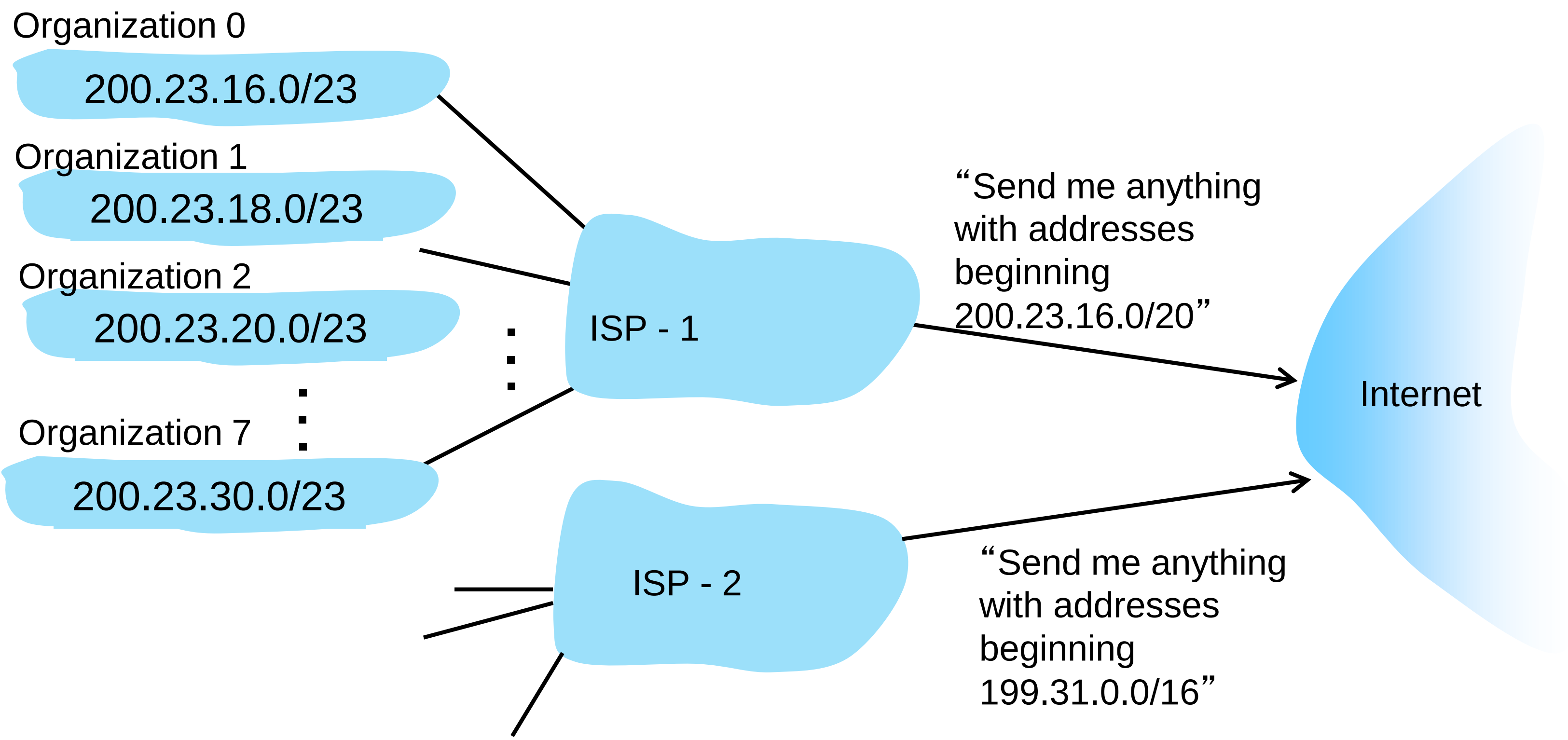
| | | |
|---|---|---|
| ISP's block | 11001000 00010111 0001 0000 00000000 | 200.23.16.0/20 |

for example: ISP can allocate out its address space in 8 blocks:

| | | |
|---|---|---|
| Organization 0 | 11001000 00010111 0001000 0 00000000 | 200.23.16.0/23 |
| Organization 1 | 11001000 00010111 0001001 0 00000000 | 200.23.18.0/23 |
| Organization 2 | 11001000 00010111 0001010 0 00000000 | 200.23.20.0/23 |
| ... | ..... .... | .... |
| Organization 7 | 11001000 00010111 0001111 0 00000000 | 200.23.30.0/23 |

# Hierarchical addressing: route aggregation

hierarchical addressing allows efficient advertisement of routing information:



# IP addressing: last words ...

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

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

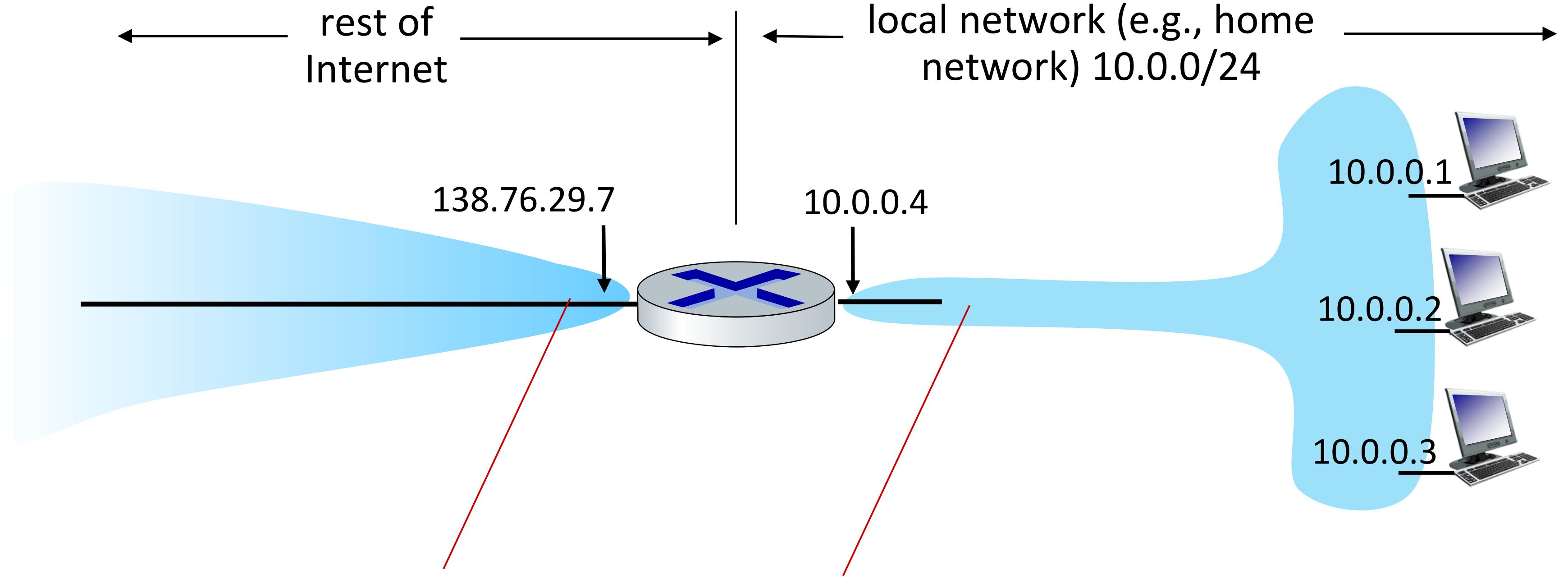
- allocates IP addresses, through 5 regional registries (RRs) (who may then allocate to local registries)
- manages DNS root zone, including delegation of individual TLD (.com, .edu , …) management

*Q:* are there enough 32-bit IP addresses?

- NAT (next) helps IPv4 address space exhaustion
- IPv6 has 128-bit address space

# NAT: network address translation

NAT: all devices in local network share just one IPv4 address as far as outside world is concerned




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

datagrams with source or destination in this network have 10.0.0/24 address for source, destination

# NAT: network address translation

- all devices in local network have 32-bit addresses in a "private" IP address space (10/8, 172.16/12, 192.168/16 prefixes) that can only be used in local network
- advantages:
  - just one IP address needed from provider ISP for *all* devices
  - can change addresses of host in local network without notifying outside world
  - can change ISP without changing addresses of devices in local network
  - security: devices inside local net not directly addressable, visible by outside world

# NAT: network address translation

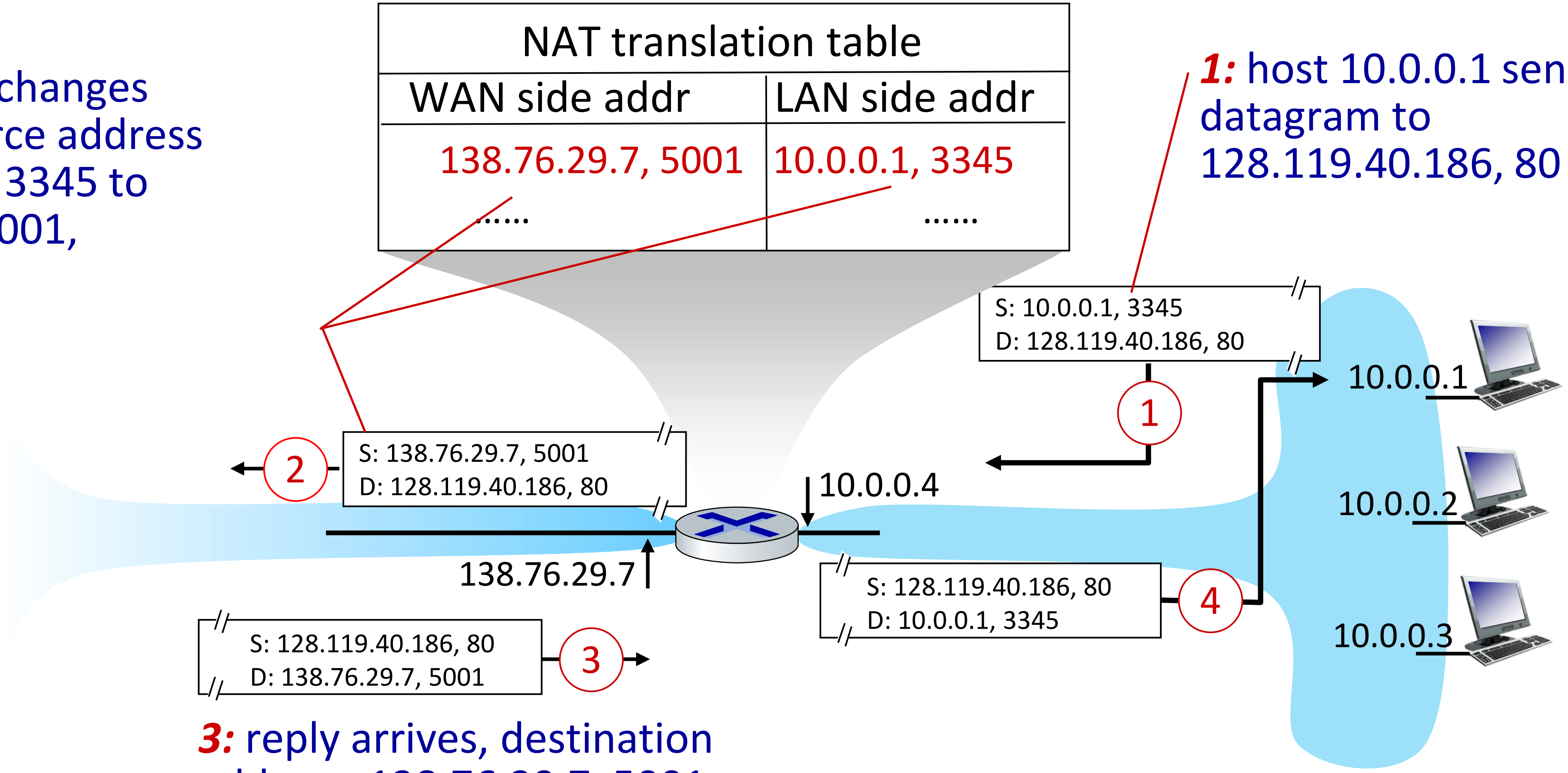
implementation: NAT router must (transparently):

- outgoing datagrams: replace (source IP address, port #) of every outgoing datagram to (NAT IP address, new port #)
  - remote clients/servers will respond using (NAT IP address, new port #) as destination address
- remember (in NAT translation table) every (source IP address, port #) to (NAT IP address, new port #) translation pair
- incoming datagrams: replace (NAT IP address, new port #) in destination fields of every incoming datagram with corresponding (source IP address, port #) stored in NAT table

# NAT: network address translation

**2:** NAT router changes datagram source address from 10.0.0.1, 3345 to 138.76.29.7, 5001, updates table

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




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

# IPv6: motivation

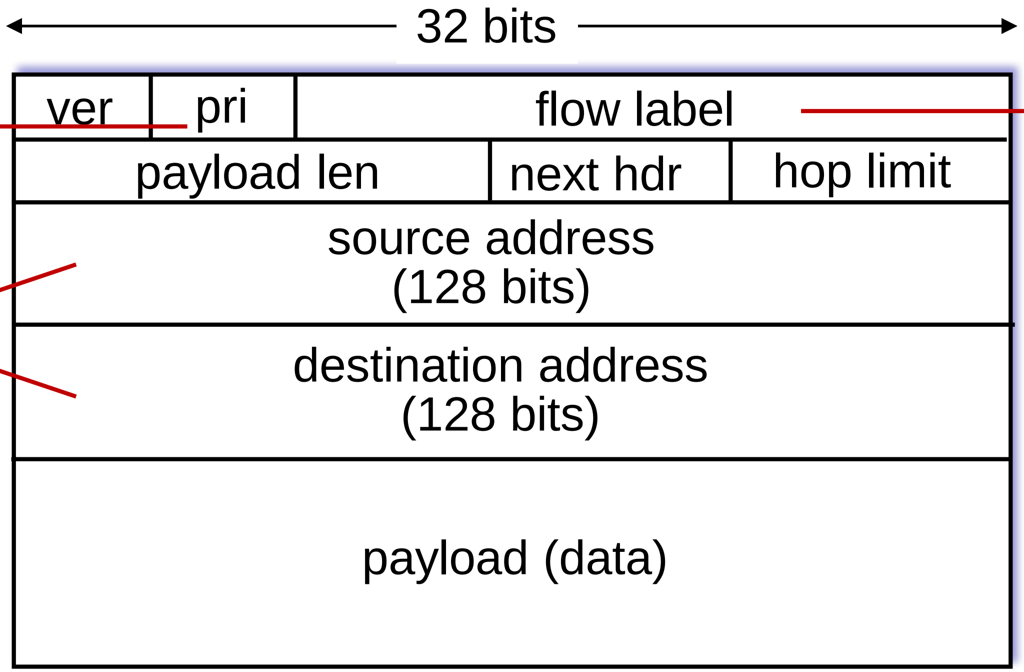
- initial motivation: 32-bit IPv4 address space would be completely allocated
- additional motivation:
  - speed processing/forwarding: 40-byte fixed length header
  - enable different network-layer treatment of “flows”

# IPv6 datagram format

(to respond the need for a large IP address space)

priority: identify priority among datagrams in flow

128-bit IPv6 addresses




flow label: identify datagrams in same "flow.” For example, audio and video transmission might likely be treated as a flow. The more traditional applications, such as file transfer and e-mail, might not be treated as flows. Concept of “flow” not well defined.

What’s missing (compared with IPv4):

- no checksum (to speed processing at routers)
- no fragmentation/reassembly
- no options

# Network layer: “data plane” roadmap

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

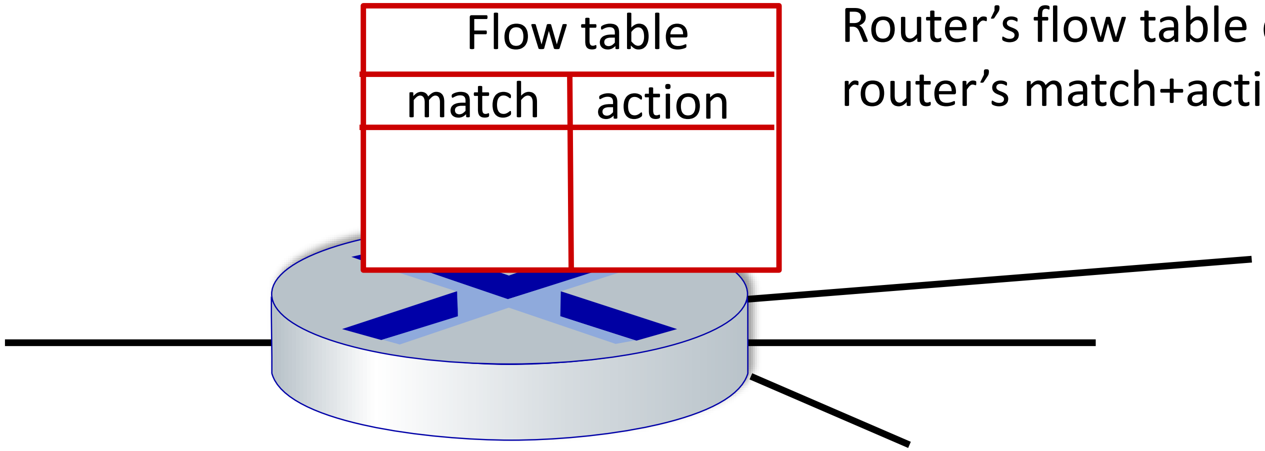
# Generalized forwarding: match plus action

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

- "match plus action" abstraction: match bits in arriving packet, take action
  - *destination-based forwarding:* forward based on dest. IP address
  - *generalized forwarding*: generalizes matches and actions
    - many header fields can determine action
    - many action possible: drop/copy/modify/log packet

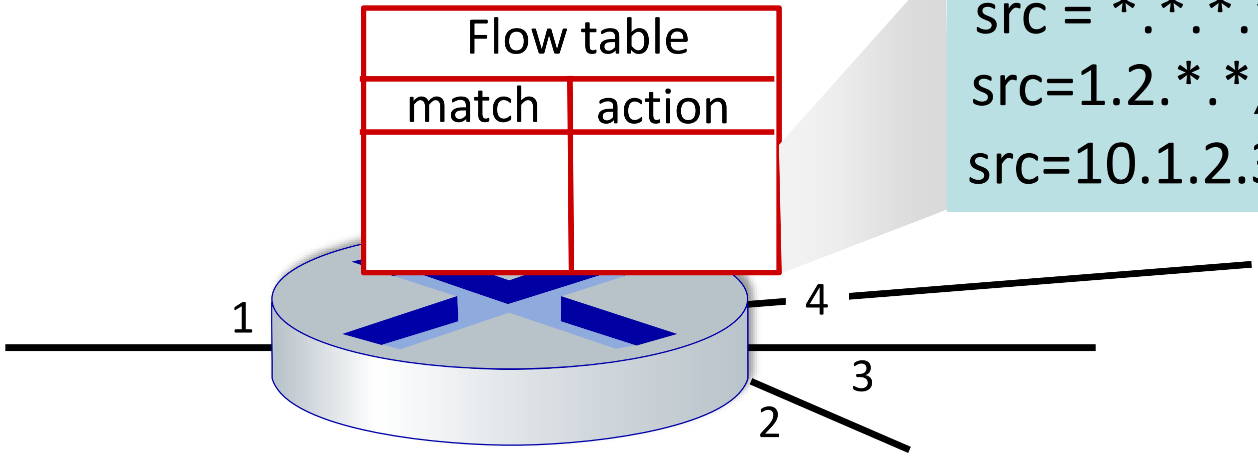
# Flow table abstraction

- flow: defined by header field values (in link-, network-, transport-layer fields)
- generalized forwarding: simple packet-handling rules
  - match: pattern values in packet header fields
  - actions: for matched packet: drop, forward, modify, matched packet or send matched packet to controller




Router's flow table define router's match+action rules

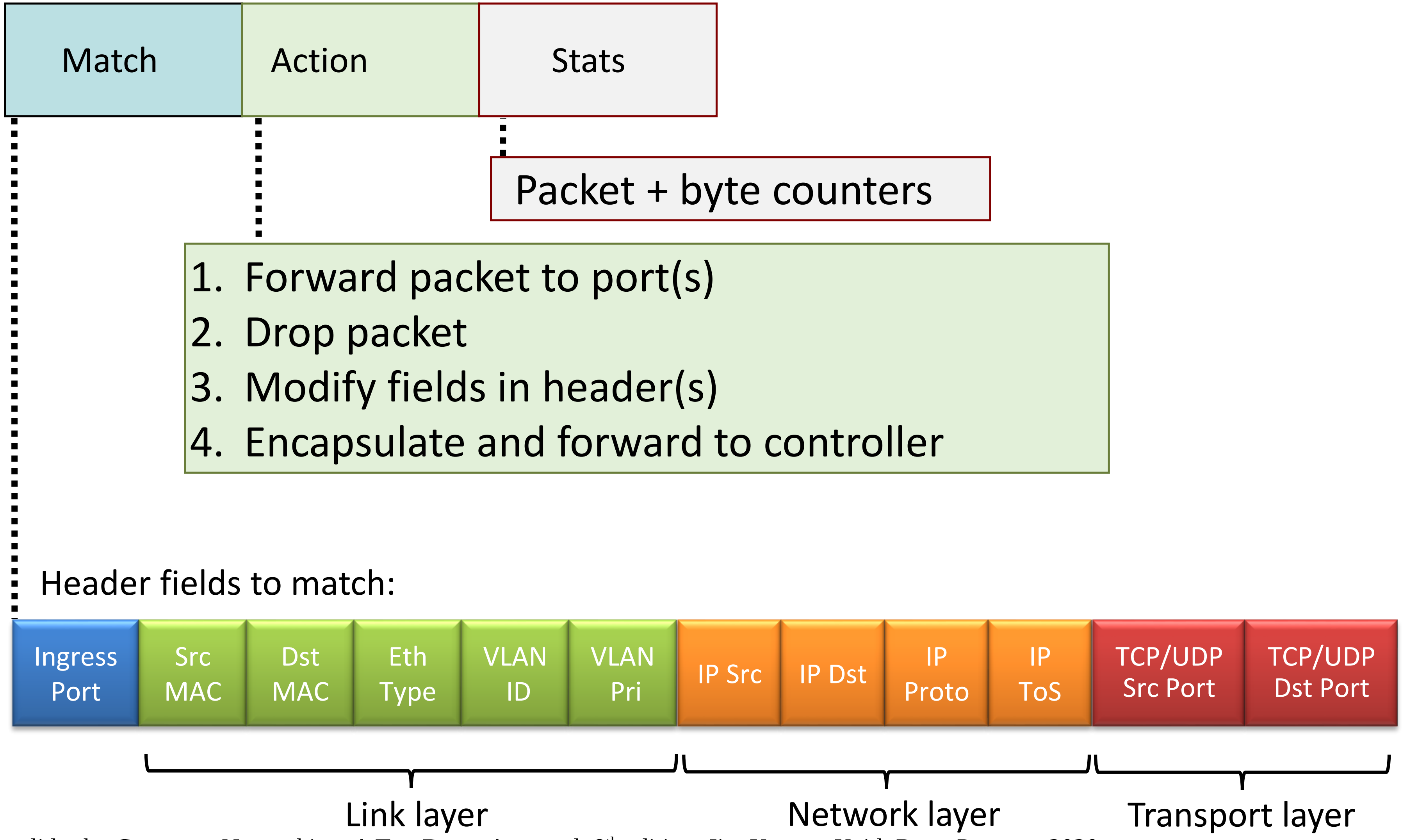
# Flow table abstraction




| match | action |
|---|---|
| src = *.*.*.*, dest=3.4.*.* | forward(2) |
| src=1.2.*.*, dest=*.*.*.* | drop |
| src=10.1.2.3, dest=*.*.*.* | send to controller |

* : wildcard

# OpenFlow: flow table entries



# OpenFlow: examples

## Destination-based forwarding:

| Switch Port | MAC src | MAC dst | Eth type | VLAN ID | VLAN Pri | IP Src | IP Dst | IP Prot | IP ToS | TCP s-port | TCP d-port | Action |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| * | * | * | * | * | * | * | 51.6.0.8 | * | * | * | * | port6 |

IP datagrams destined to IP address 51.6.0.8 should be forwarded to router output port 6

## Firewall:

| Switch Port | MAC src | MAC dst | Eth type | VLAN ID | VLAN Pri | IP Src | IP Dst | IP Prot | IP ToS | TCP s-port | TCP d-port | Action |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| * | * | * | * | * | * | * | * | * | * | * | 22 | drop |

Block (do not forward) all datagrams destined to TCP port 22 (ssh port #)

| Switch Port | MAC src | MAC dst | Eth type | VLAN ID | VLAN Pri | IP Src | IP Dst | IP Prot | IP ToS | TCP s-port | TCP d-port | Action |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| * | * | * | * | * | * | 128.119.1.1 | * | * | * | * | * | drop |

Block (do not forward) all datagrams sent by host 128.119.1.1

# OpenFlow: examples

## Layer 2 destination-based forwarding:

| Switch Port | MAC src | MAC dst | Eth type | VLAN ID | VLAN Pri | IP Src | IP Dst | IP Prot | IP ToS | TCP s-port | TCP d-port | Action |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| * | * | 22:A7:23:<br>11:E1:02 | * | * | * | * | * | * | * | * | * | port3 |

layer 2 frames with destination MAC address 22:A7:23:11:E1:02 should be forwarded to output port 3

# OpenFlow abstraction

- match+action: abstraction unifies different kinds of devices

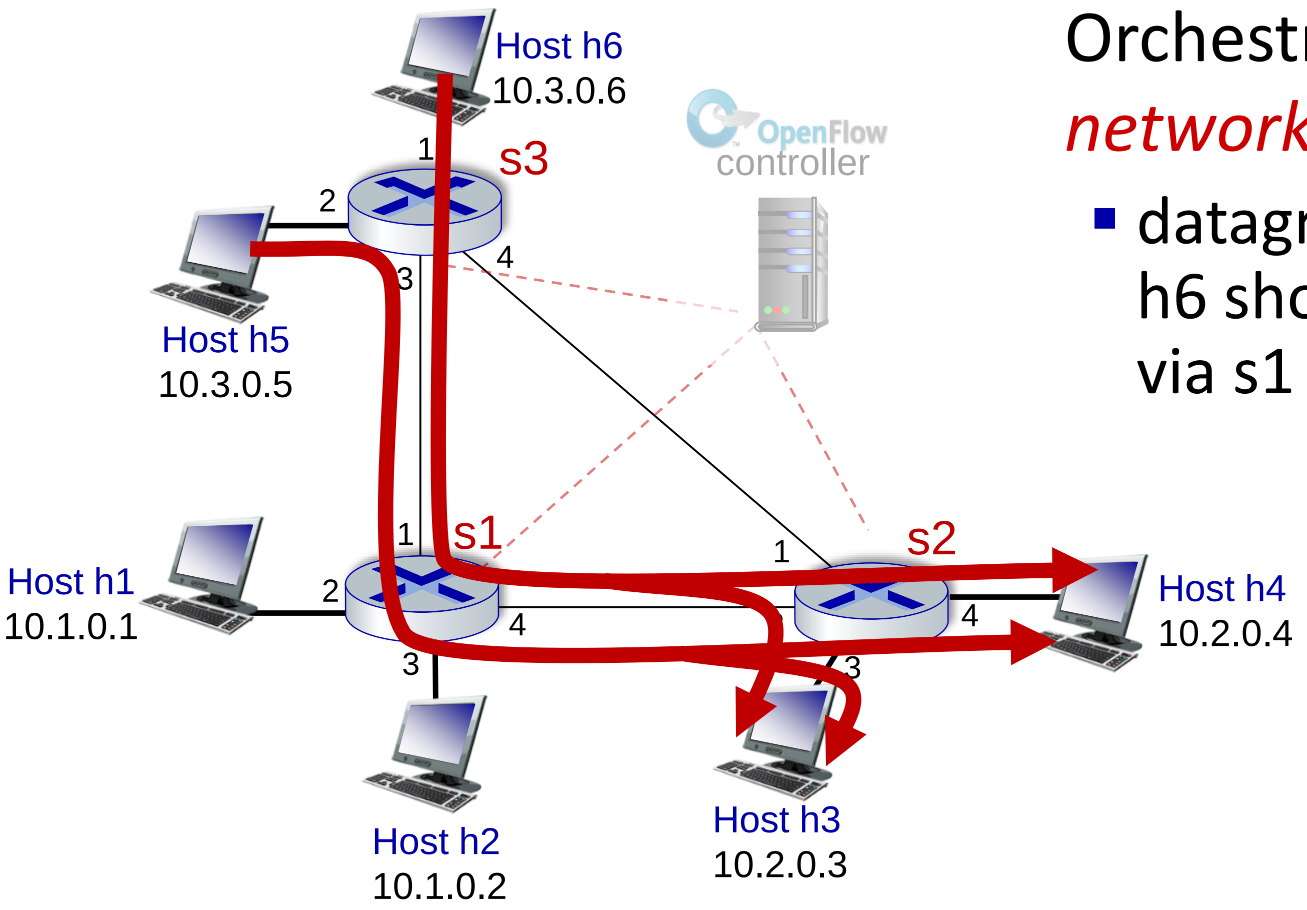
**Router**
- *match:* longest destination IP prefix
- *action:* forward out a link

**Switch**
- *match:* destination MAC address
- *action:* forward or flood

**Firewall**
- *match*: IP addresses and TCP/UDP port numbers
- *action:* permit or deny

**NAT**
- *match:* IP address and port
- *action:* rewrite address and port

# OpenFlow example

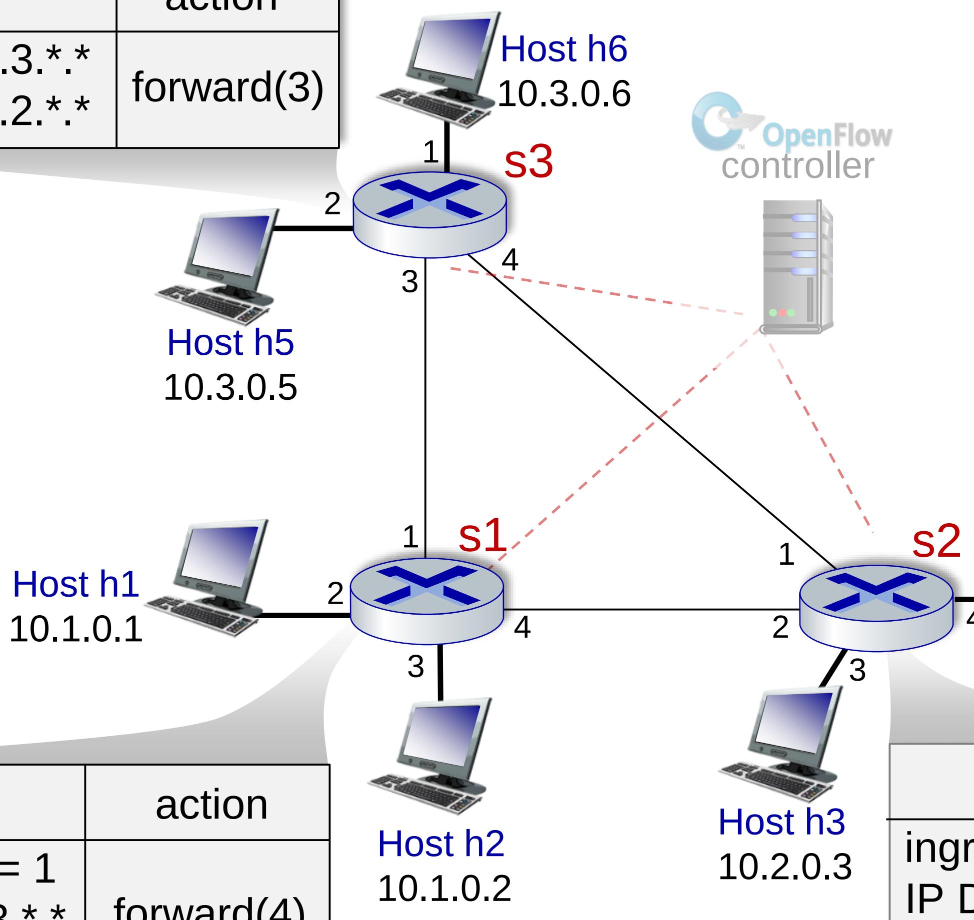



Orchestrated tables can create *network-wide* behavior, e.g.,:

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

# OpenFlow example

| match | action |
| --- | --- |
| IP Src = 10.3.*.* IP Dst = 10.2.*.* | forward(3) |




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

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

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

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

# Network layer: "data plane" roadmap

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

# Middleboxes

Middlebox (RFC 3234)

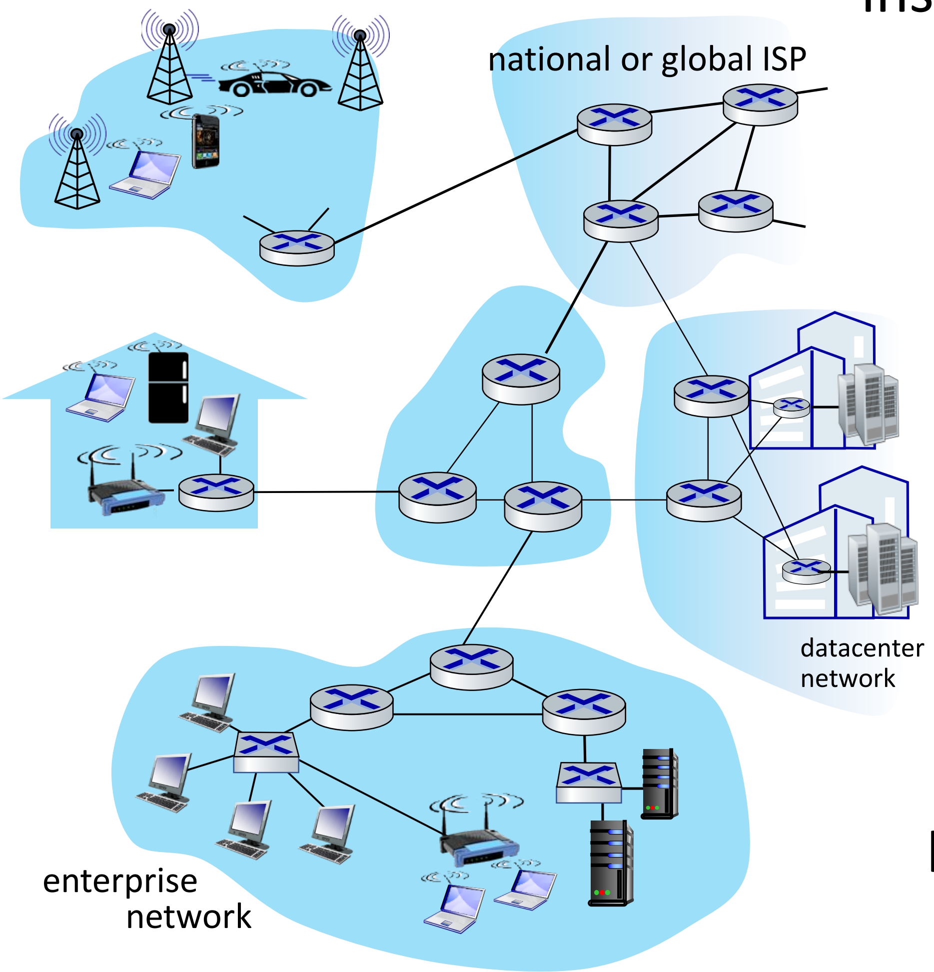
"any intermediary box performing functions apart from normal, standard functions of an IP router on the data path between a source host and destination host"

*perform functions other than forwarding*

# Middleboxes everywhere!

NAT: home, cellular, institutional

Application-specific: service providers, institutional, CDN




Firewalls, IDS: corporate, institutional, service providers, ISPs

Load balancers: corporate, service provider, data center, mobile nets

Caches: service provider, mobile, CDNs