Network Layer – Data Plane

TDTS04 – Computer Networks and Distributed Systems

Ulf Kargén

Division for Database and Information Techniques (ADIT)
Department of Computer and Information Science (IDA)

LINKÖPING UNIVERSITY

Network layer: our goals

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

- instantiation, implementation in the Internet
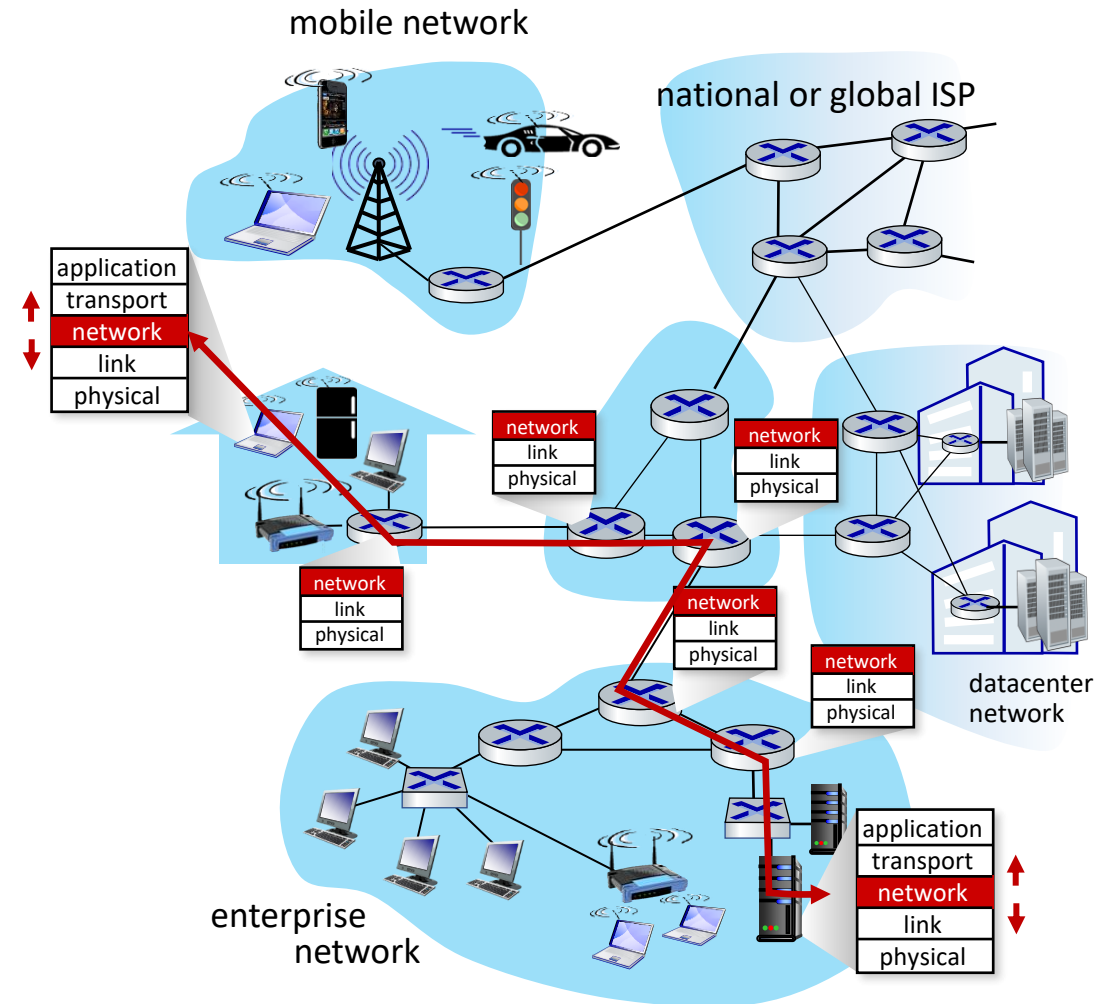 - IP protocol
  - NAT, middleboxes

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

■ transport segment from sending to receiving host
  - sender: encapsulates segments into datagrams, passes to link layer
  - receiver: delivers segments to transport layer protocol

■ network layer protocols 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 to transfer datagrams along end-end path

mobile network

national or global ISP

application
transport
network
link
physical

network
link
physical

network
link
physical

network
link
physical

network
link
physical

network
link
physical

network
link
physical

datacenter network

enterprise network

application
transport
network
link
physical

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
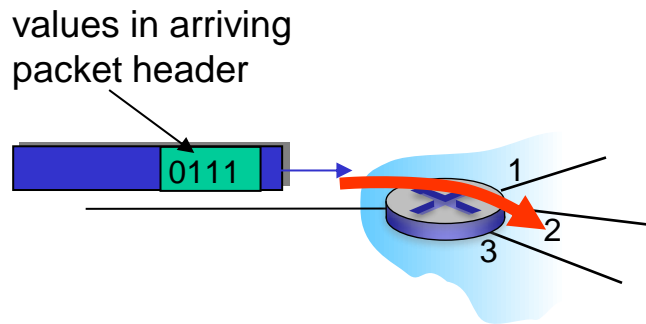

forwarding

routing

Network layer: data plane, control plane

Data plane:

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


values in arriving packet header

Control plane

- *network-wide* logic
- determines how datagram is routed among routers along end-end path from source host to destination host
- two control-plane approaches:
  - *traditional routing algorithms:* implemented in routers
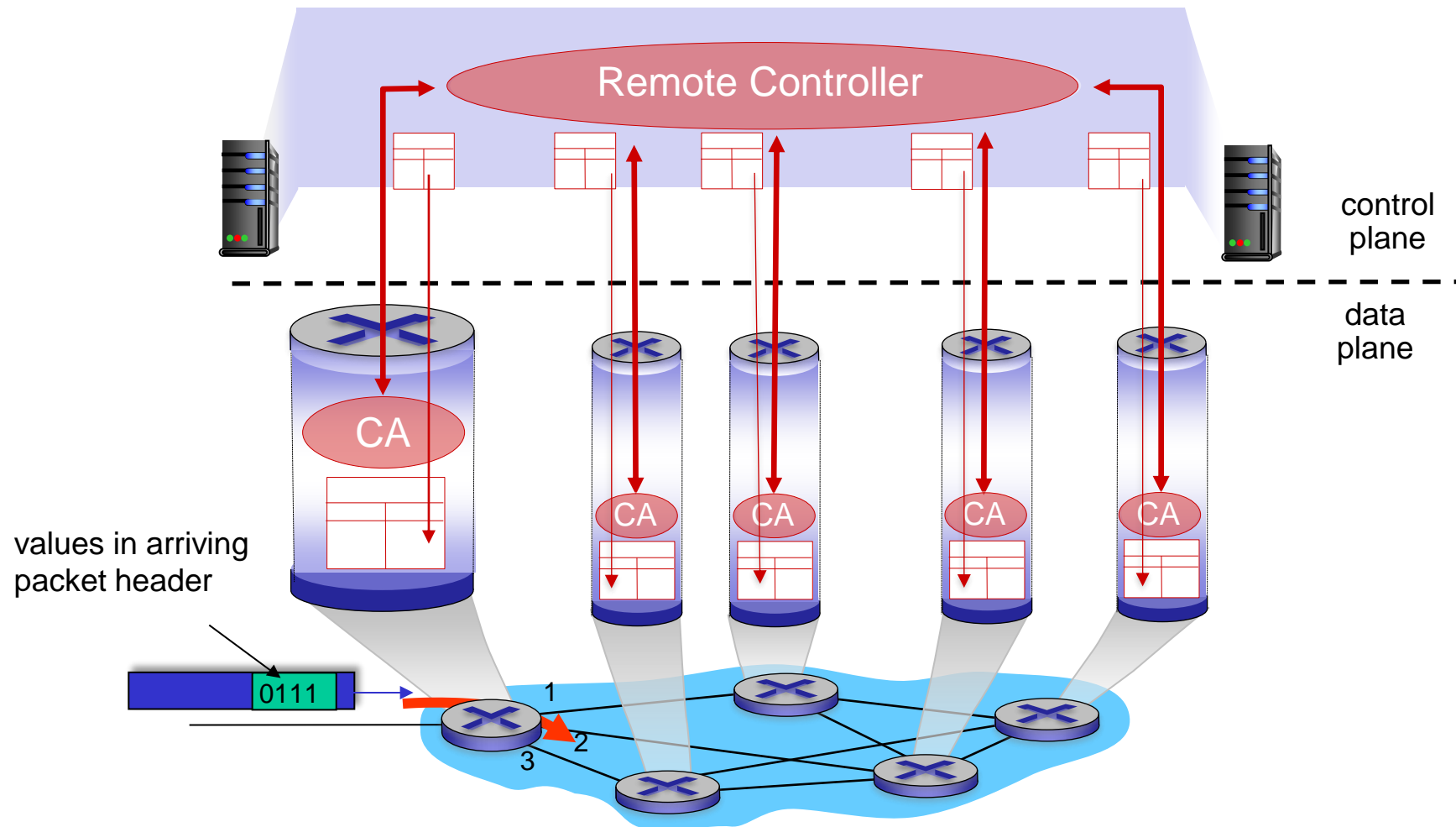 - *software-defined networking (SDN)*: implemented in (remote) servers

Per-router control plane

Individual routing algorithm components *in each and every router* interact in the control plane
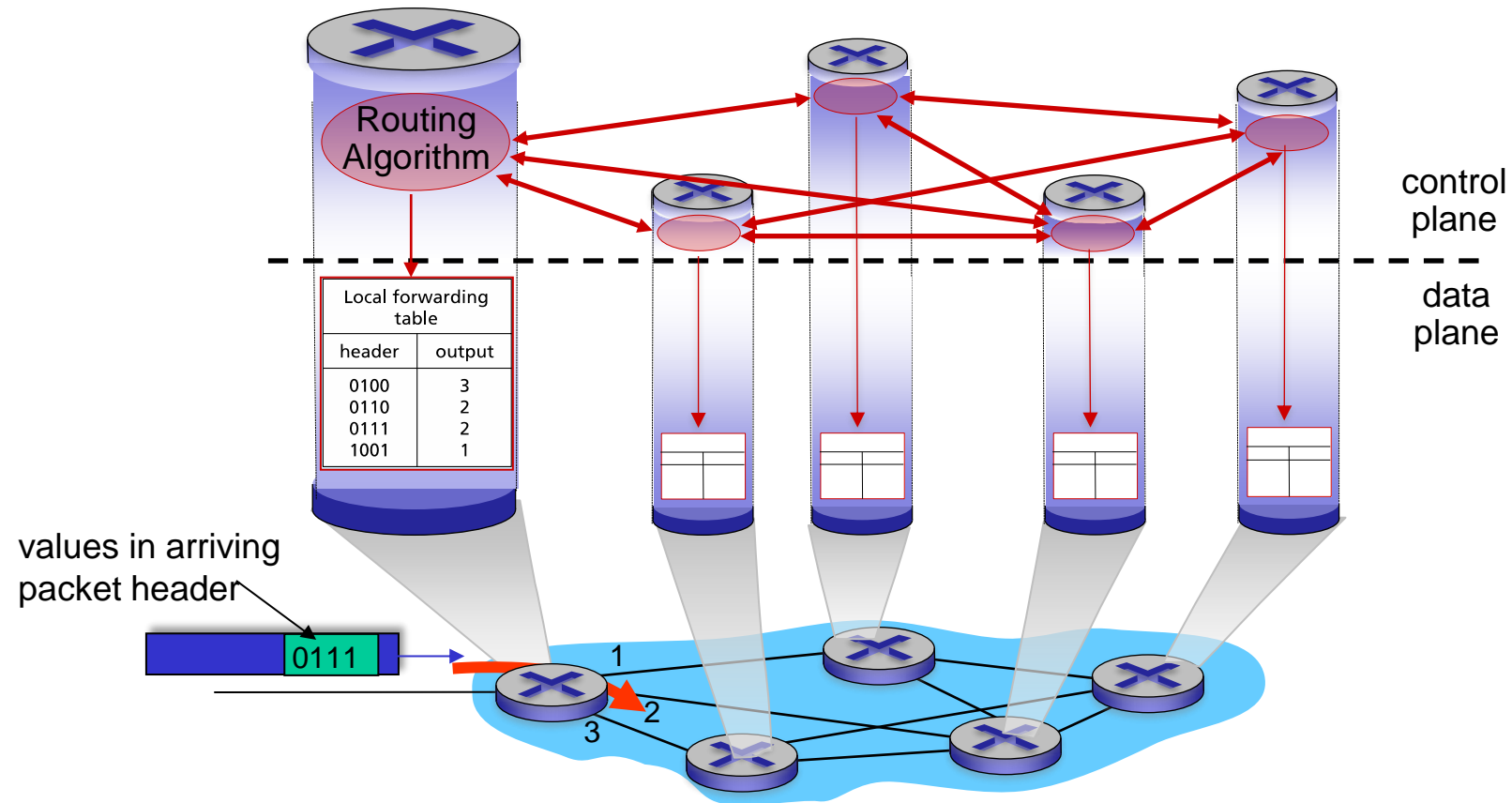
Software-Defined Networking (SDN) control plane

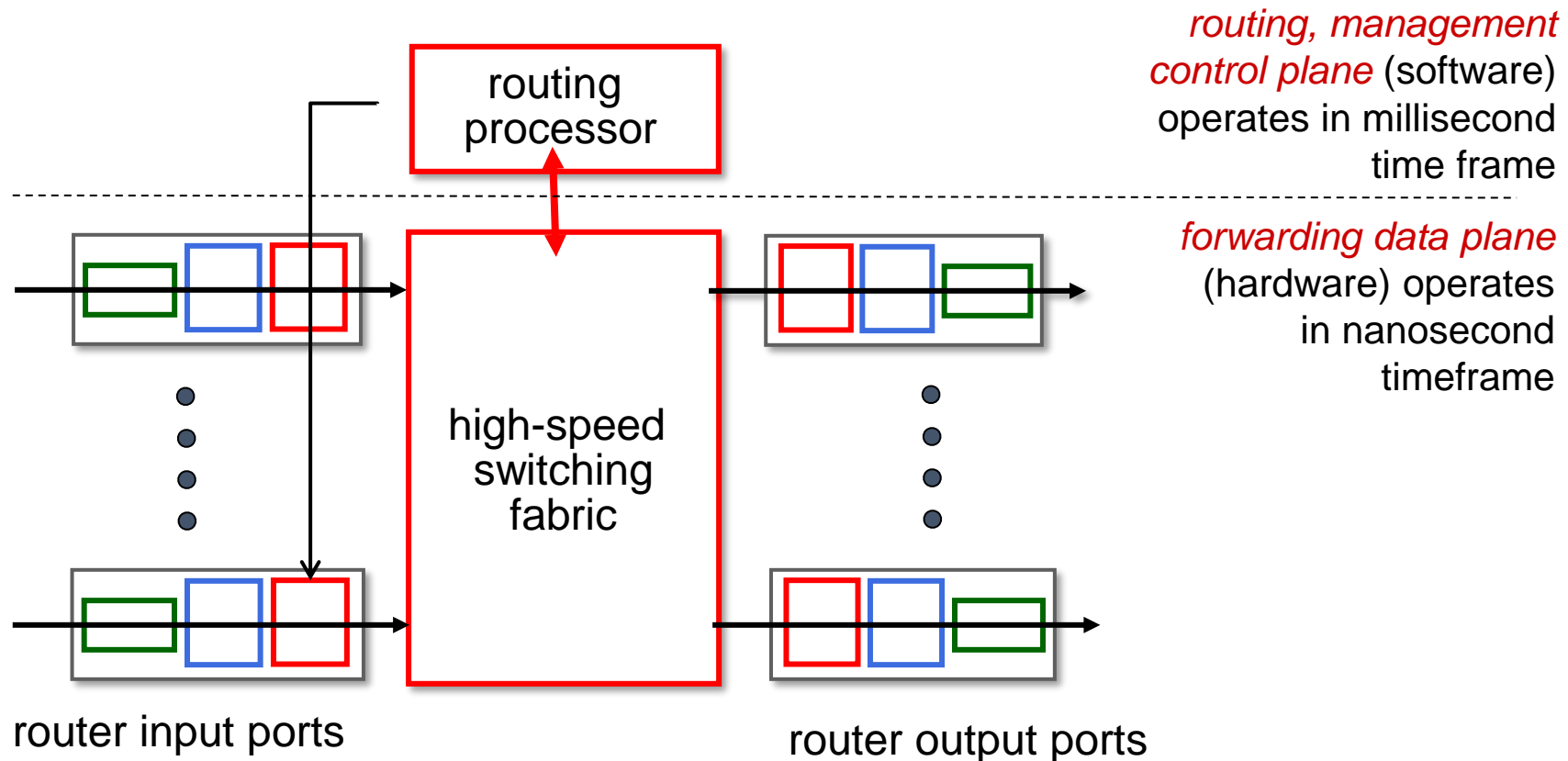Remote controller computes, installs forwarding tables in routers

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

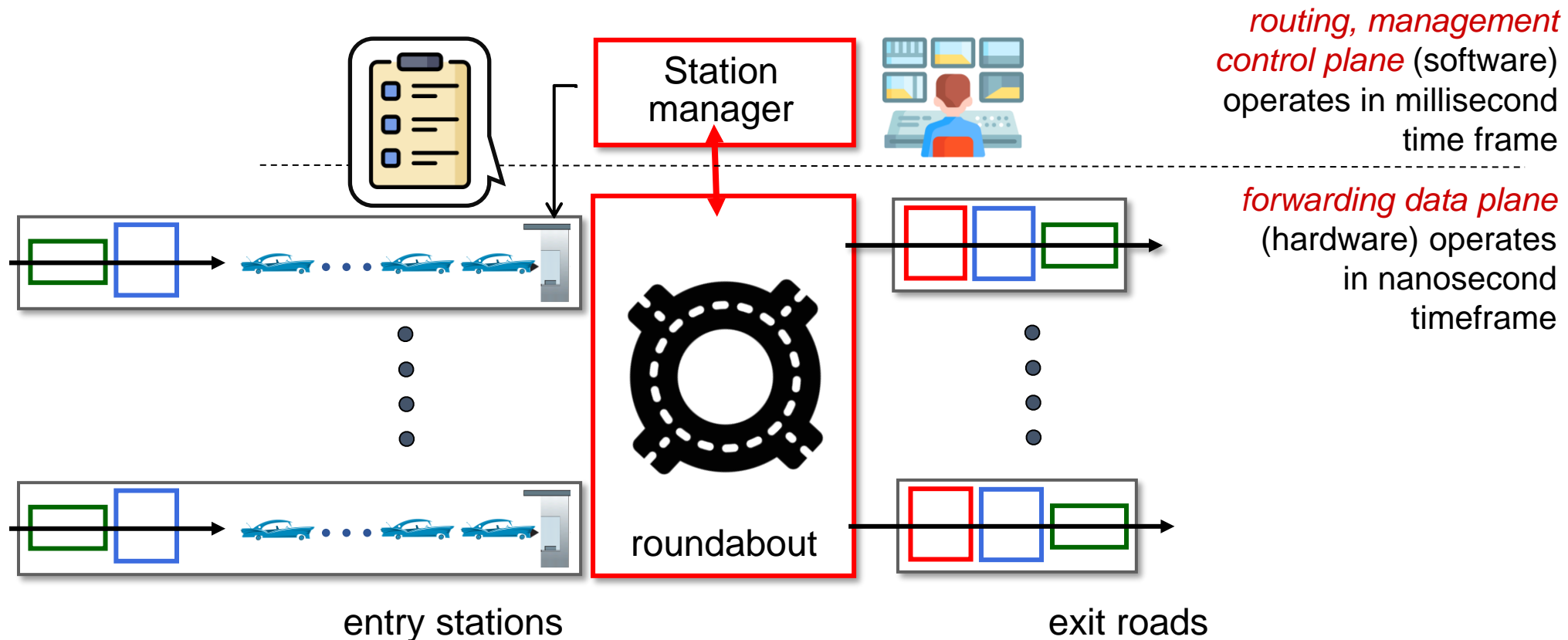- Middleboxes

Router architecture overview

high-level view of generic router architecture:

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

forwarding data plane (hardware) operates in nanosecond timeframe

Router architecture overview

analogy view of generic router architecture:

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

forwarding data plane (hardware) operates in nanosecond timeframe

Station manager

roundabout

entry stations

exit roads

Input port functions
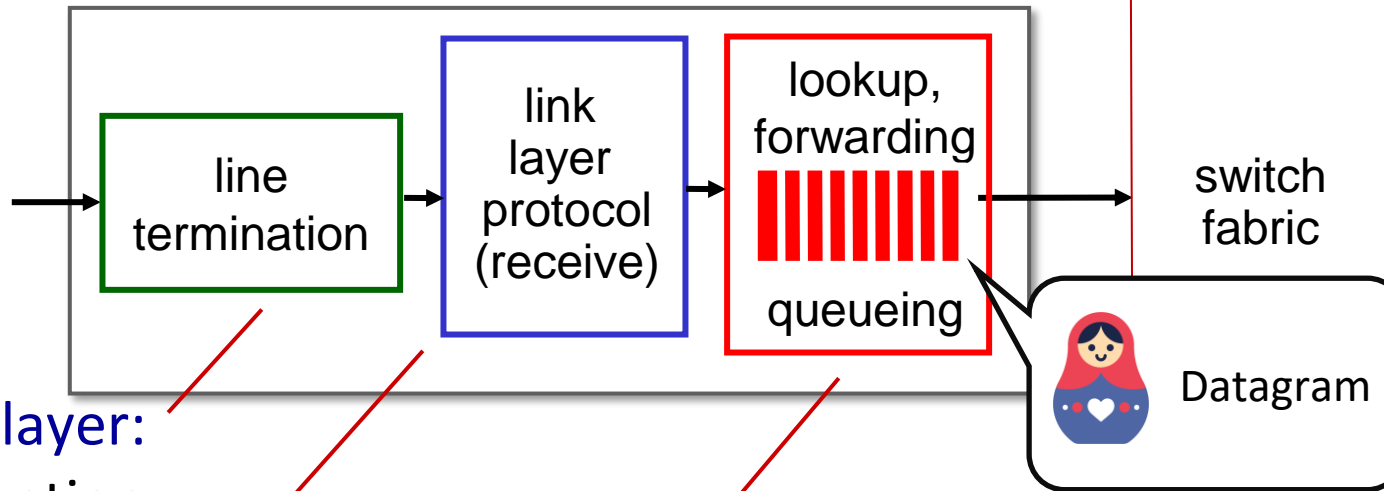
physical layer:
bit-level reception

link layer:
e.g., Ethernet
(chapter 6)

Frame

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'
  - otherwise, throughput limited by switching fabric
- input port queuing: if datagrams arrive faster than forwarding rate into switch fabric

Input port functions

line termination

link layer protocol (receive)

lookup, forwarding

queueing

switch fabric

physical layer:
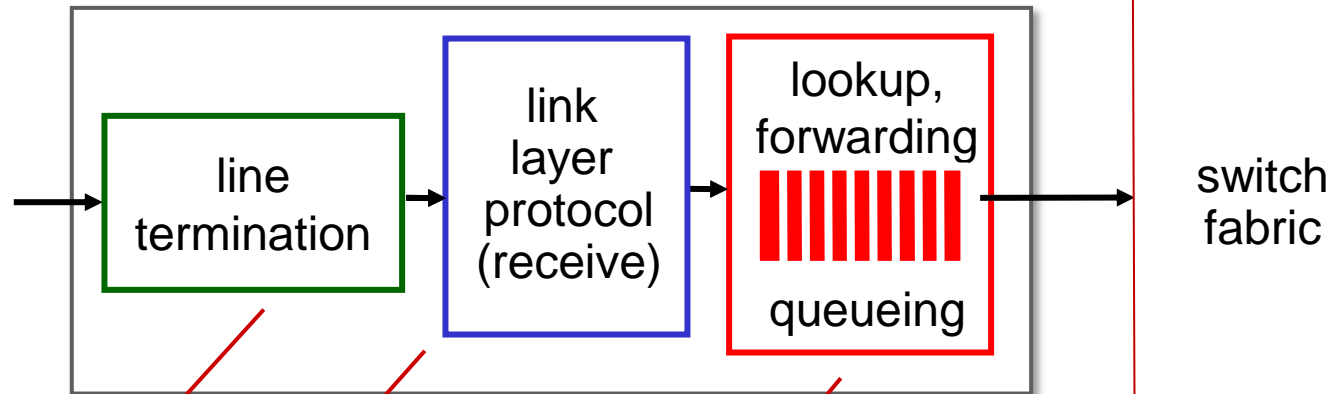bit-level reception

link layer:
e.g., Ethernet (chapter 6)

decentralized switching:

- using header field values, lookup output port using forwarding table in input port memory *("match plus action")*
- destination-based forwarding: forward based only on destination IP address (traditional)
- generalized forwarding: forward based on any set of header field values
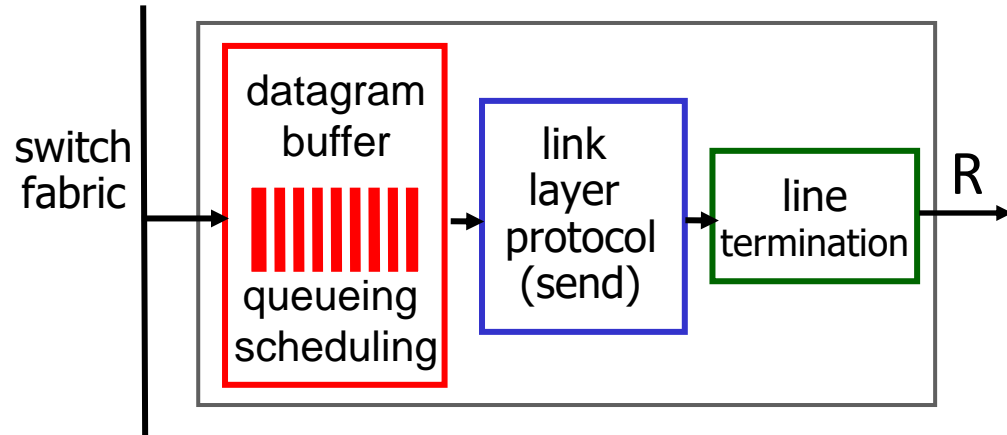
How much buffering?

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

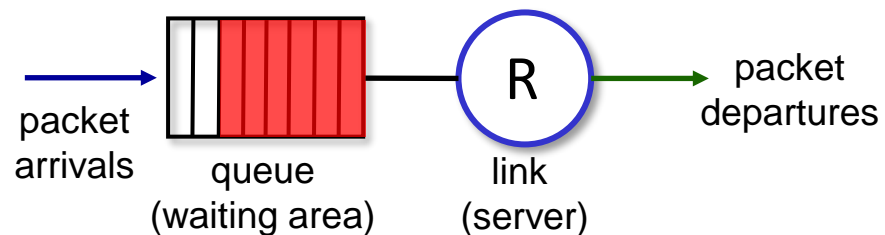- more recent recommendation: with *N* flows, buffering equal to

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

- but *too* much buffering can increase delays (particularly in home routers)
  - long RTTs: poor performance for real-time apps, sluggish TCP response
  - recall delay-based congestion control: "keep bottleneck link just full enough (busy) but no fuller"

Buffer Management

switch
fabric

datagram
buffer
queueing
scheduling

link
layer
protocol
(send)

line
termination

R

Abstraction: queue

packet
arrivals

queue
(waiting area)

link
(server)

R

packet
departures

buffer management:

- **drop:** which packet to add, drop when buffers are full
  - **tail drop:** drop arriving packet
  - **priority:** drop/remove on priority basis

- **marking:** which packets to mark to signal congestion (ECN, RED)
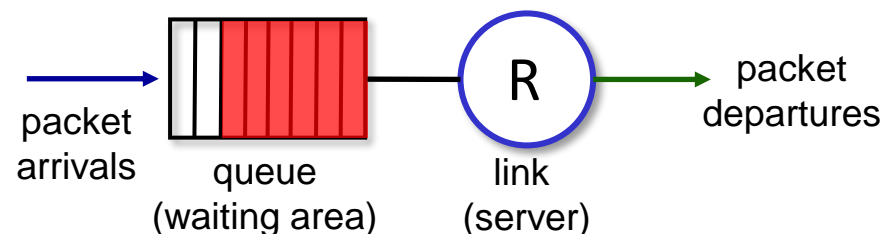
Packet Scheduling: FCFS

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)

Abstraction: queue

Destination-based forwarding

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?

Destination-based forwarding

forwarding table

Destination Address Range	Link Interface
11001000 00010111 00010000 00000000 through 11001000 00010111 00010000 00000100 through 11001000 00010111 00010000 00000111	0 3
11001000 00010111 00011000 11111111 11001000 00010111 00011001 00000000	2
11111 00011111 11111111	3

Let's say that a subset of addresses in first range should go to interface 3

Could split the first address range into multiple pieces as shown here, but there's a simpler solution…

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 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

- we'll see *why* longest prefix matching is used shortly, when we study addressing

- longest prefix matching: often performed using ternary content addressable memories (TCAMs)
  - *content addressable:* present address to TCAM: retrieve address in one clock cycle, regardless of table size
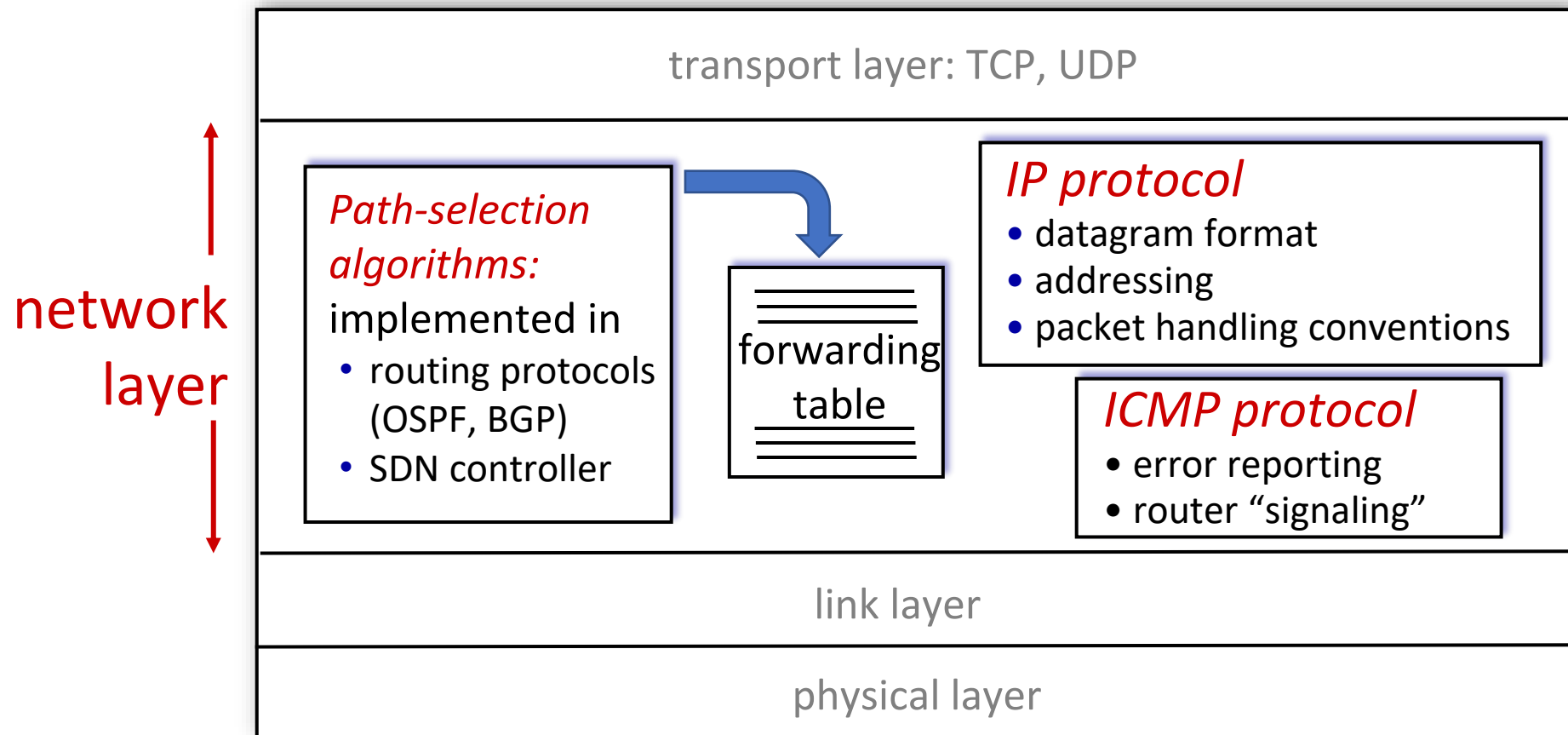 - Cisco Catalyst: ~1M routing table entries in TCAM

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
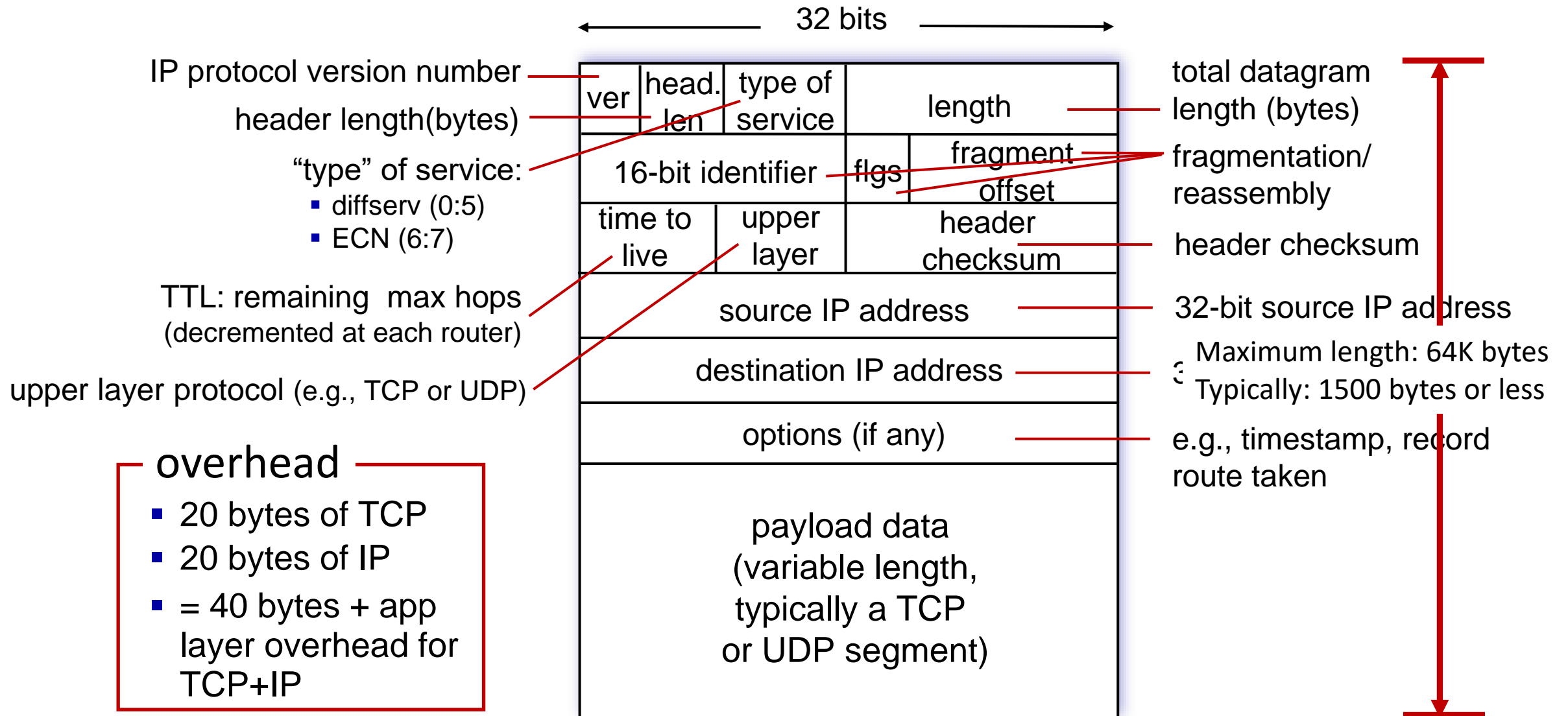- Middleboxes

Network Layer: Internet

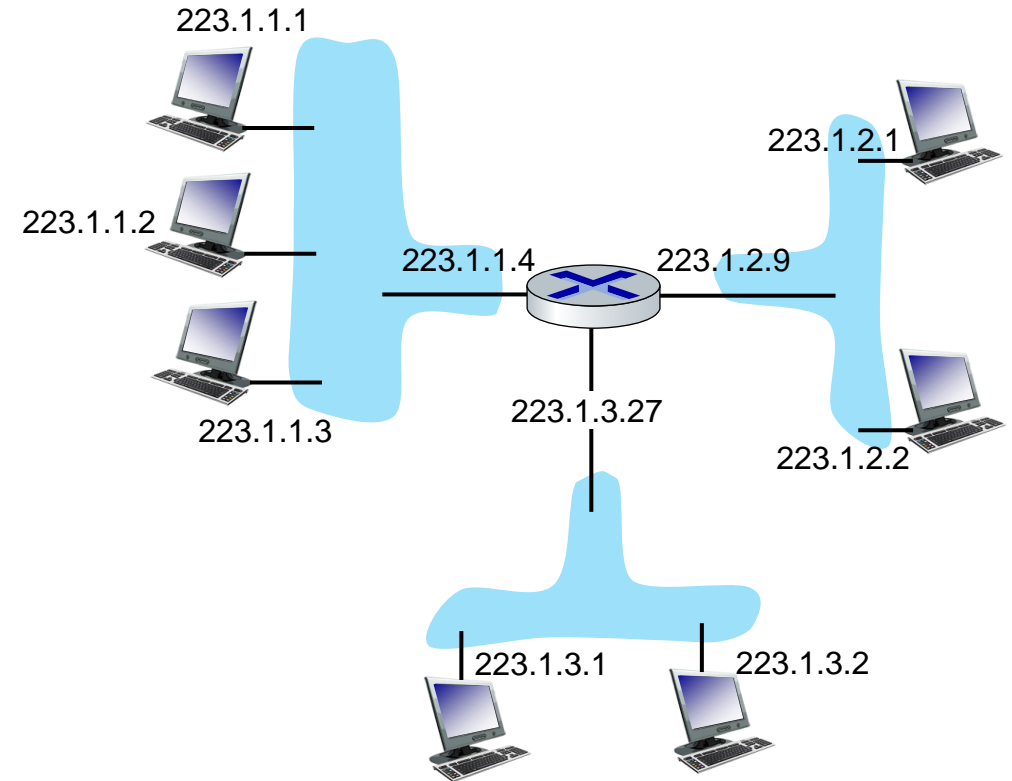host, router network layer functions:

IP Datagram format

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)

32 bits

ver	head. len	type of service	length
16-bit identifier		flgs	fragment offset
time to live	upper layer	header checksum
source IP address			
destination IP address			
options (if any)			
payload data (variable length, typically a TCP or UDP segment)			

total datagram length (bytes)

fragmentation/ reassembly

header checksum

32-bit source IP address

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

e.g., timestamp, record route taken

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
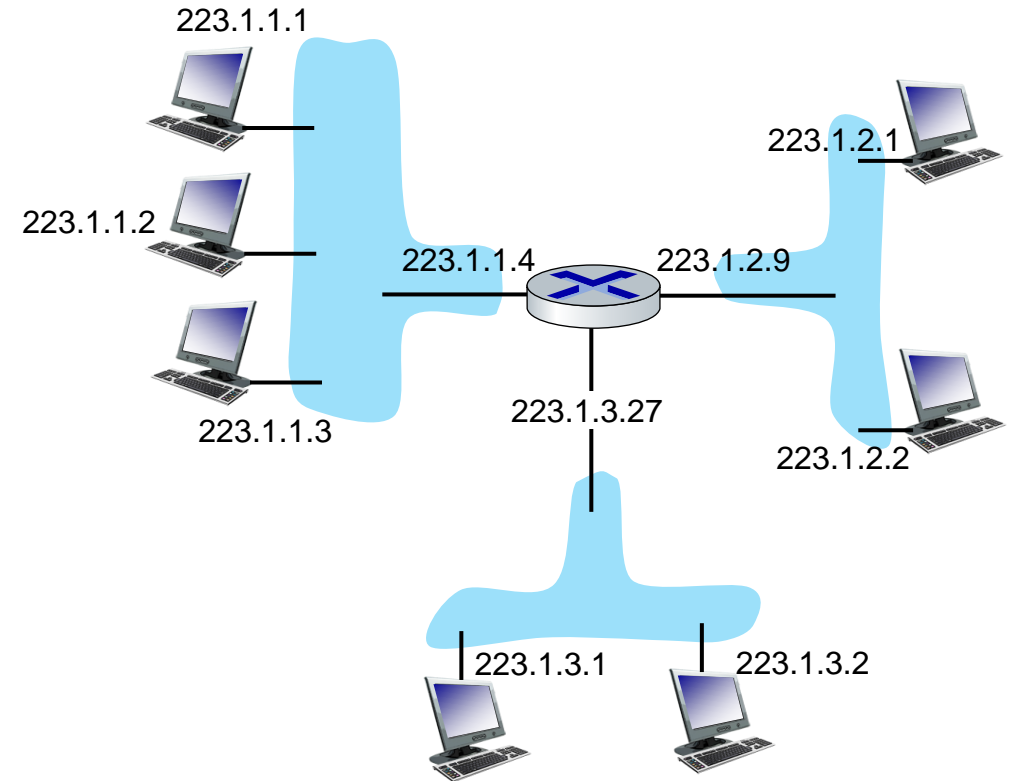 - host typically has one or two interfaces (e.g., wired Ethernet, wireless 802.11)

223.1.1.1

223.1.1.2

223.1.1.3

223.1.1.4

223.1.2.9

223.1.3.27

223.1.2.1

223.1.2.2

223.1.3.1

223.1.3.2

dotted-decimal IP address notation:

223.1.1.1 = 11011111 00000001 00000001 00000001

                 223   1   1   1

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)

223.1.1.1

223.1.1.2

223.1.1.4

223.1.1.3

223.1.2.1

223.1.2.9

223.1.2.2

223.1.3.27

223.1.3.1   223.1.3.2

dotted-decimal IP address notation:

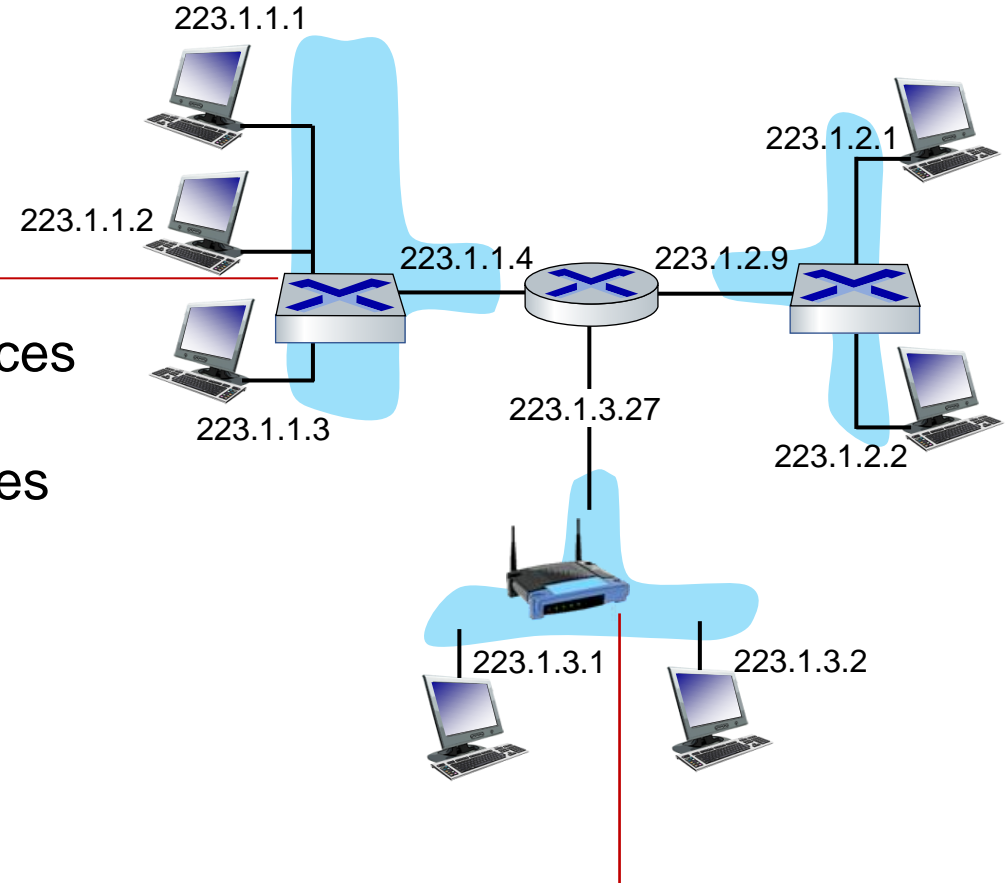223.1.1.1 = 11011111 00000001 00000001 00000001

223   1   1   1

IP addressing: introduction

Q: how are interfaces actually connected?

A: we'll learn about that in chapters 6, 7

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

223.1.1.1
223.1.1.2
223.1.1.3
223.1.1.4
223.1.2.1
223.1.2.9
223.1.2.2
223.1.3.27
223.1.3.1
223.1.3.2

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
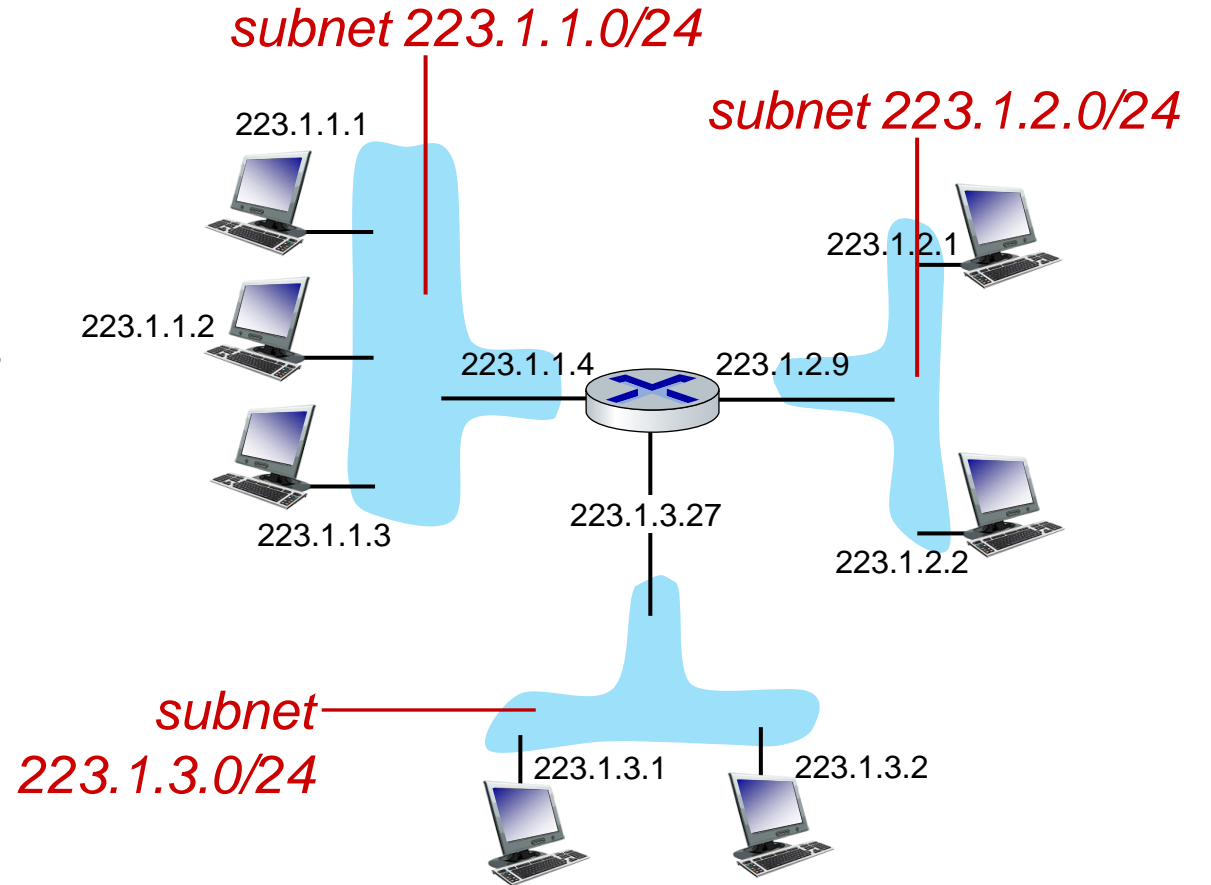 - host part: remaining low order bits
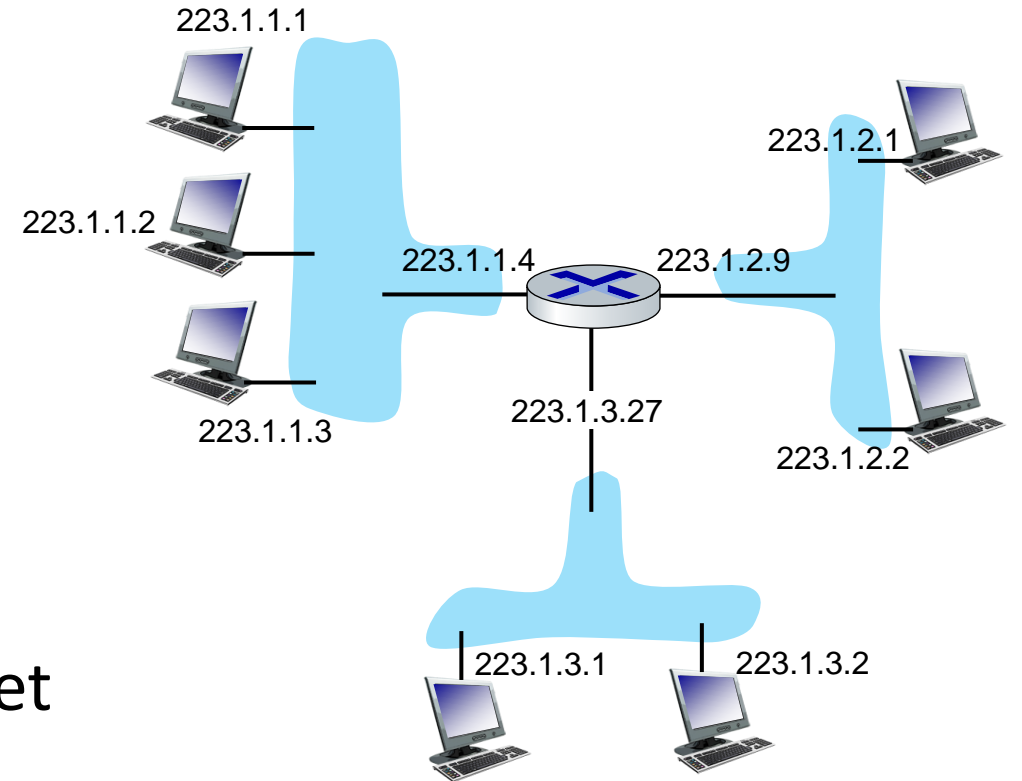
network consisting of 3 subnets

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 223.1.1.0/24

223.1.1.1

223.1.1.2

223.1.1.4

223.1.1.3

subnet 223.1.2.0/24

223.1.2.1

223.1.2.9

223.1.2.2

223.1.3.27

subnet 223.1.3.0/24

223.1.3.1

223.1.3.2

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

223.1.1.1 = 11011111 00000001 00000001 00000001

223   1   1   1

Subnets

- where are the subnets?

- what are the /24 subnet addresses?

223.1.1.2

subnet 223.1.1/24

223.1.1.1

223.1.1.4

223.1.1.3

223.1.9.2   223.1.7.0

subnet 223.1.9/24   *subnet 223.1.7/24*

223.1.9.1   223.1.7.1

223.1.8.1   223.1.8.0

223.1.2.6   *subnet 223.1.8/24*   223.1.3.27

subnet 223.1.2/24   *subnet 223.1.3/24*

223.1.2.1   223.1.2.2   223.1.3.1   223.1.3.2
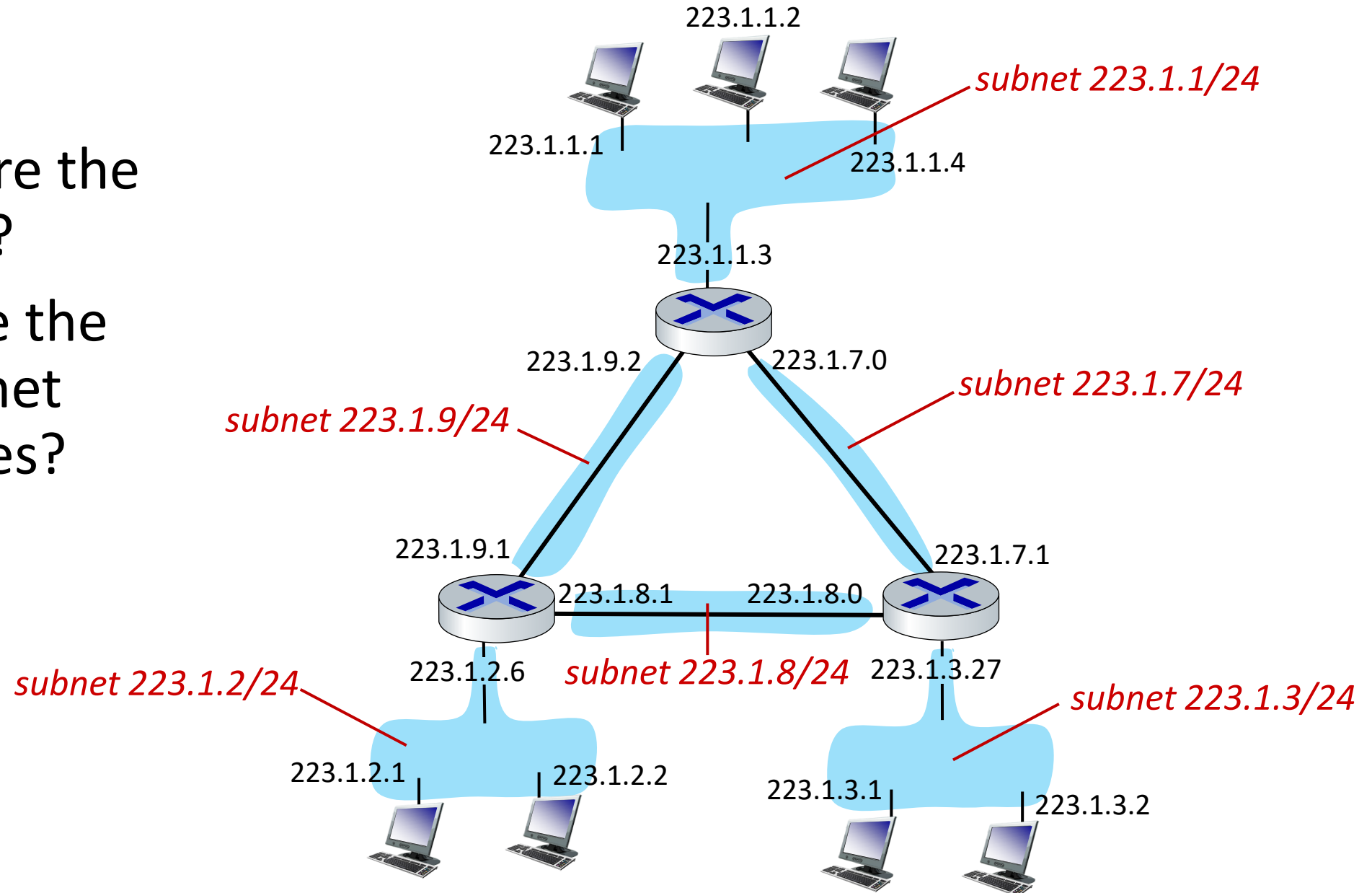
IP addressing: CIDR

CIDR: Classless InterDomain 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 ——————————→  ←— host —→
              part                    part
  11001000  00010111  00010000  00000000
```

200.23.16.0/23
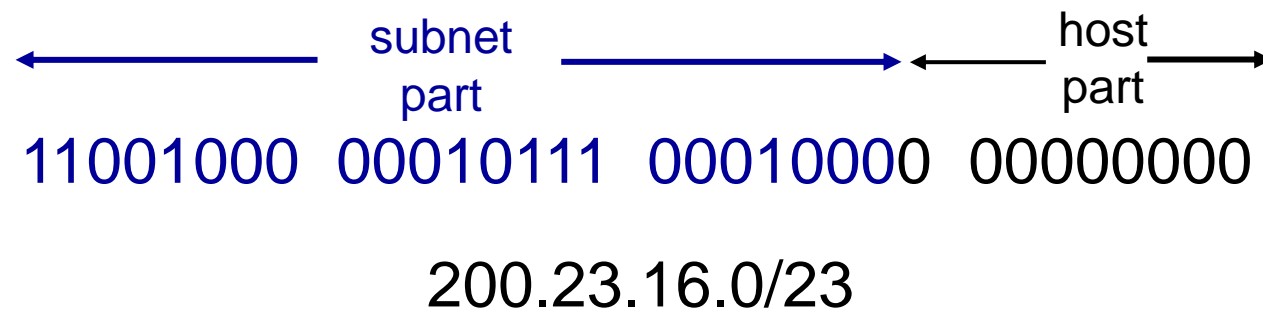
IP addresses: how to get one?

That's actually 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"

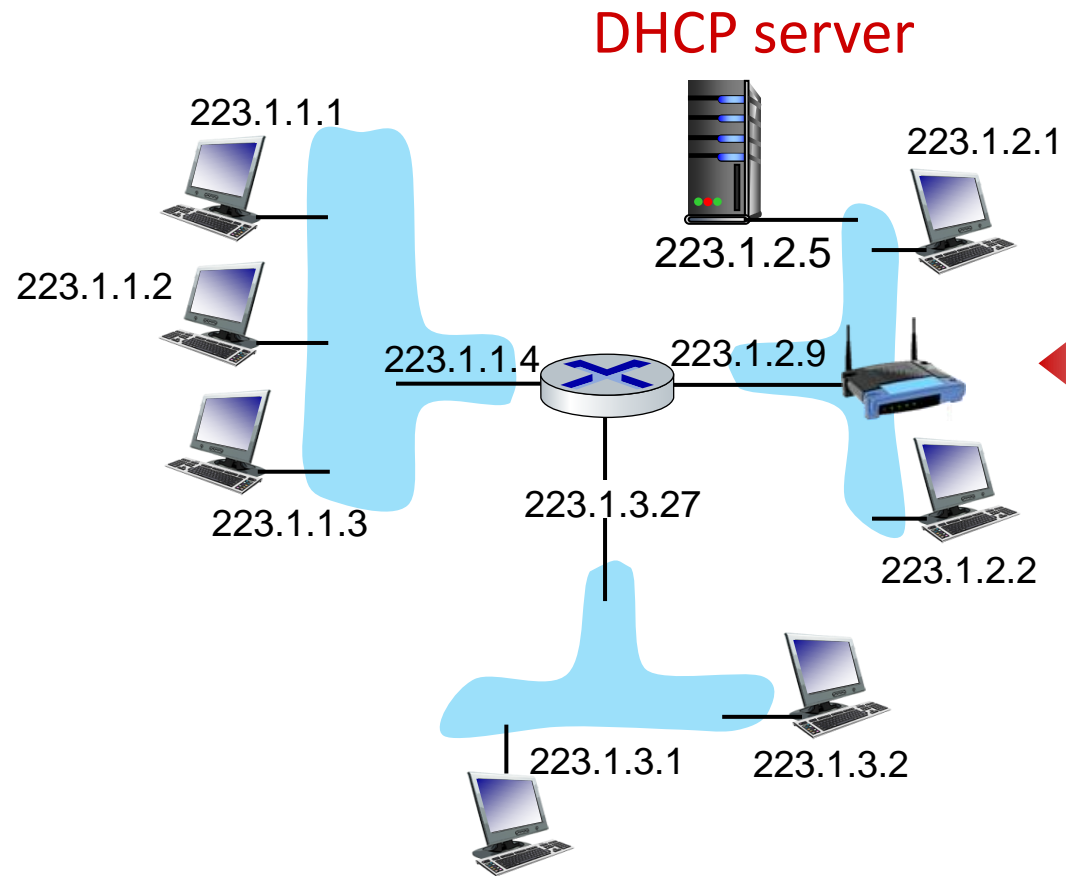# DHCP: Dynamic Host Configuration Protocol

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:

- Uses UDP on ports 67, 68
- host broadcasts DHCP discover msg [optional]
- DHCP server responds with DHCP offer msg [optional]
- host requests IP address: DHCP request msg
- DHCP server sends address: DHCP ack msg

DHCP client-server scenario

DHCP server

223.1.1.1

223.1.1.2

223.1.1.4   223.1.2.9

223.1.3.27

223.1.1.3

223.1.2.5

223.1.2.1

223.1.2.2

223.1.3.1   223.1.3.2

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

DHCP server: 223.1.2.5

DHCP discover

src : 0.0.0.0, 68
dest.: 255.255.255.255,67
yiaddr: 0.0.0.0
transaction ID: 654

Arriving client

DHCP offer

src: 223.1.2.5, 67
dest: 255.255.255.255, 68
yiaddr: 223.1.2.4
transaction ID: 654
lifetime: 3600 secs

DHCP request

src: 0.0.0.0, 68
dest:: 255.255.255.255, 67
yiaddr: 223.1.2.4
transaction ID: 655
lifetime: 3600 secs

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

DHCP ACK

src: 223.1.2.5, 67
dest: 255.255.255.255, 68
yiaddr: 223.1.2.4
transaction ID: 655
lifetime: 3600 secs

DHCP: more than IP addresses

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

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

DHCP: example

router with DHCP server built into router
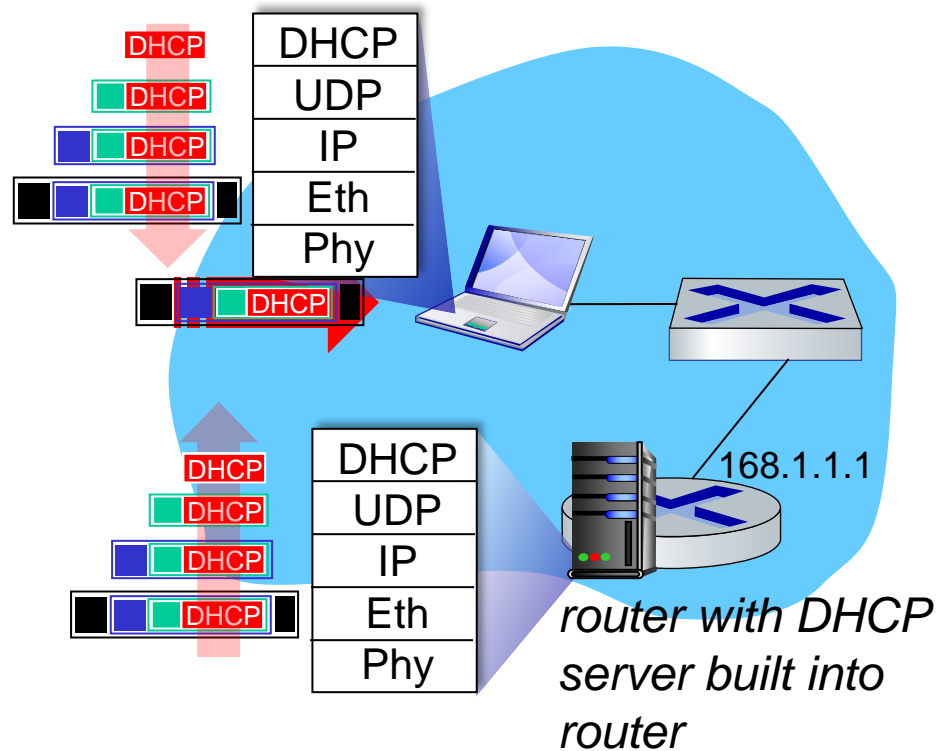
- Connecting laptop will use DHCP to get IP address, address of first-hop router, address of DNS server.

- 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 de-mux'ed to IP de-mux'ed, UDP de-mux'ed to DHCP

DHCP: example

router with DHCP server built into router
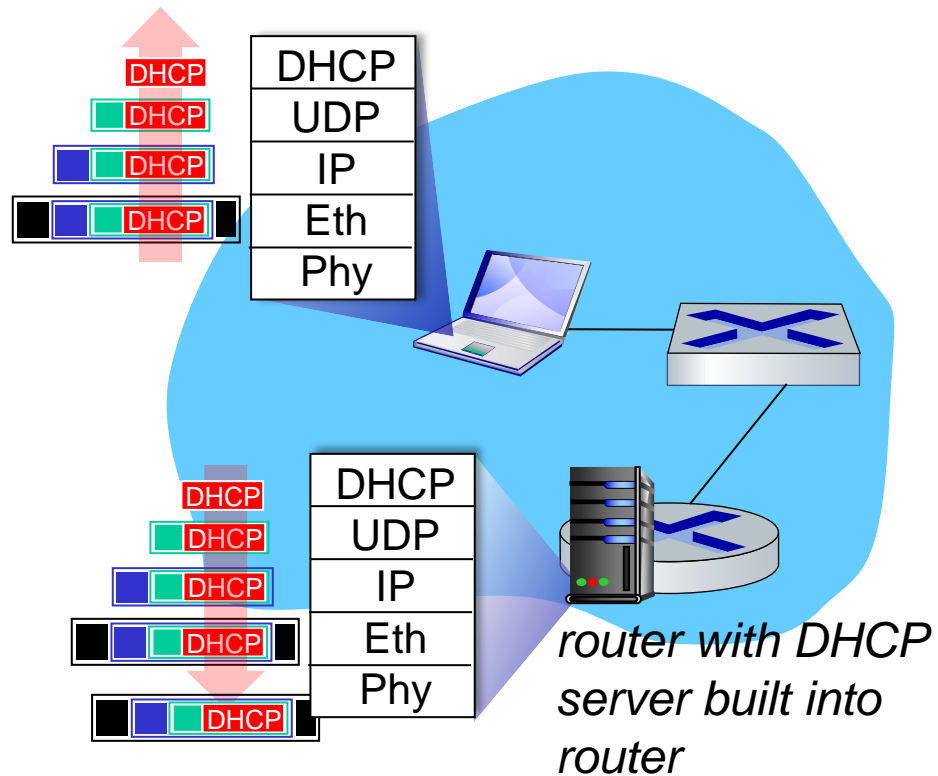
- DCP server formulates DHCP ACK containing client's IP address, IP address of first-hop router for client, name & IP address of DNS server

- encapsulated DHCP server reply forwarded to client, de-muxing 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?

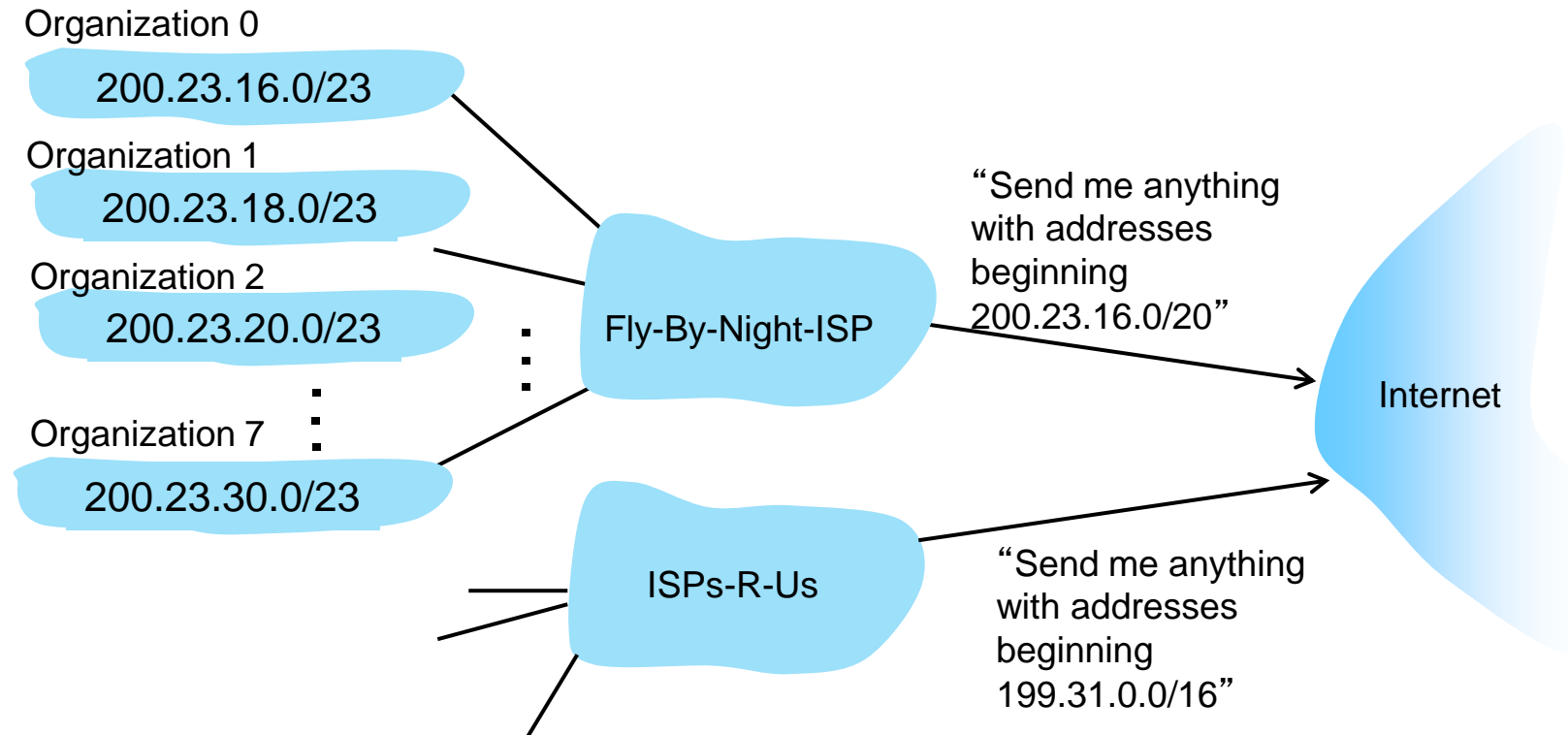*A:* gets allocated portion of its provider ISP's address space

ISP's block   <u>11001000 00010111 0001</u>0000 00000000   200.23.16.0/20

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

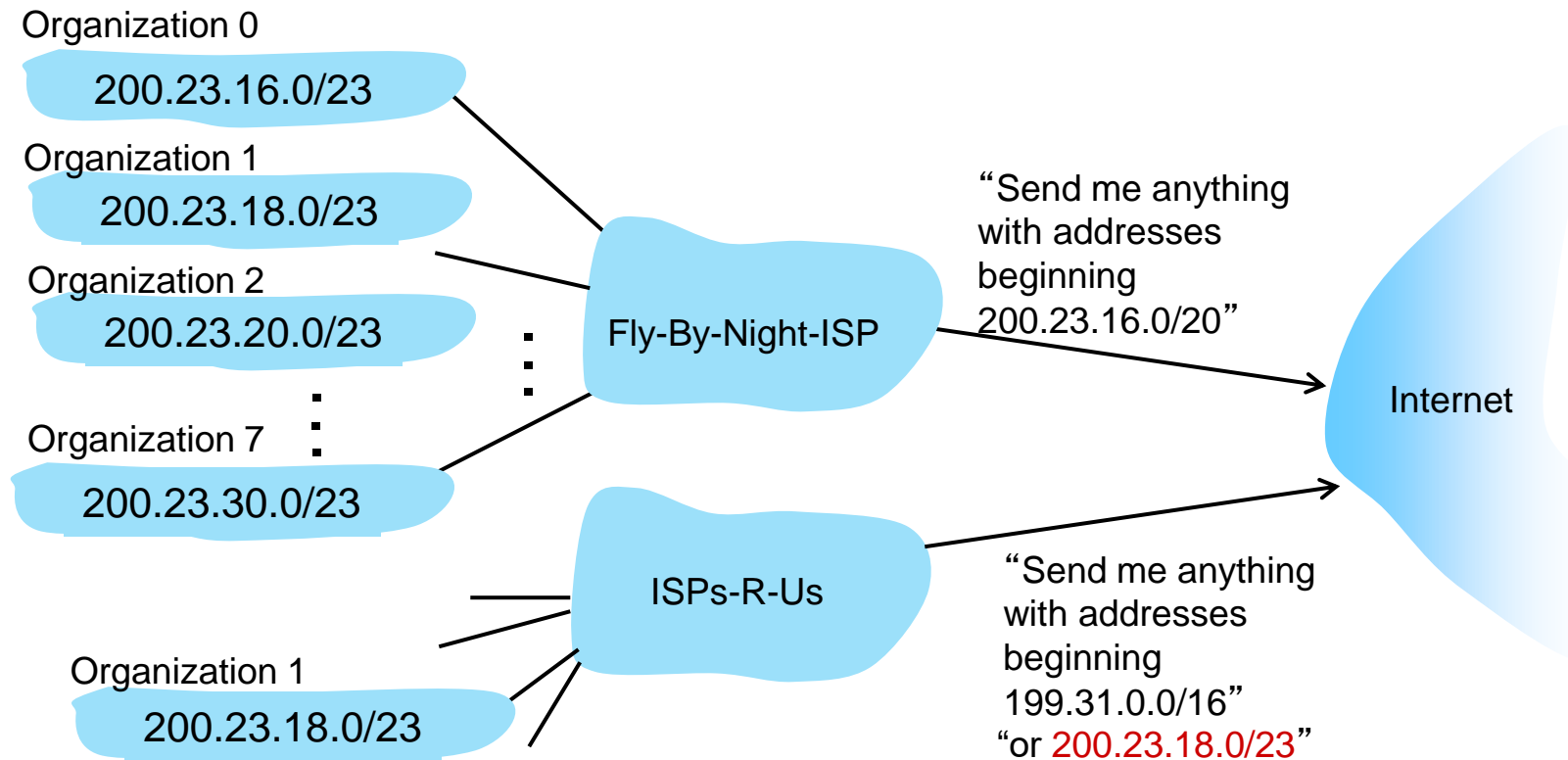|			
Organization 0	<u>11001000 00010111 0001000</u>0 00000000	200.23.16.0/23
Organization 1	<u>11001000 00010111 0001001</u>0 00000000	200.23.18.0/23
Organization 2	<u>11001000 00010111 0001010</u>0 00000000	200.23.20.0/23
...	…..	….	….
Organization 7	<u>11001000 00010111 0001111</u>0 00000000	200.23.30.0/23

Hierarchical addressing: route aggregation

hierarchical addressing allows efficient advertisement of routing information:

Organization 0
200.23.16.0/23

Organization 1
200.23.18.0/23

Organization 2
200.23.20.0/23

Organization 7
200.23.30.0/23

Fly-By-Night-ISP

"Send me anything with addresses beginning 200.23.16.0/20"

ISPs-R-Us

"Send me anything with addresses beginning 199.31.0.0/16"
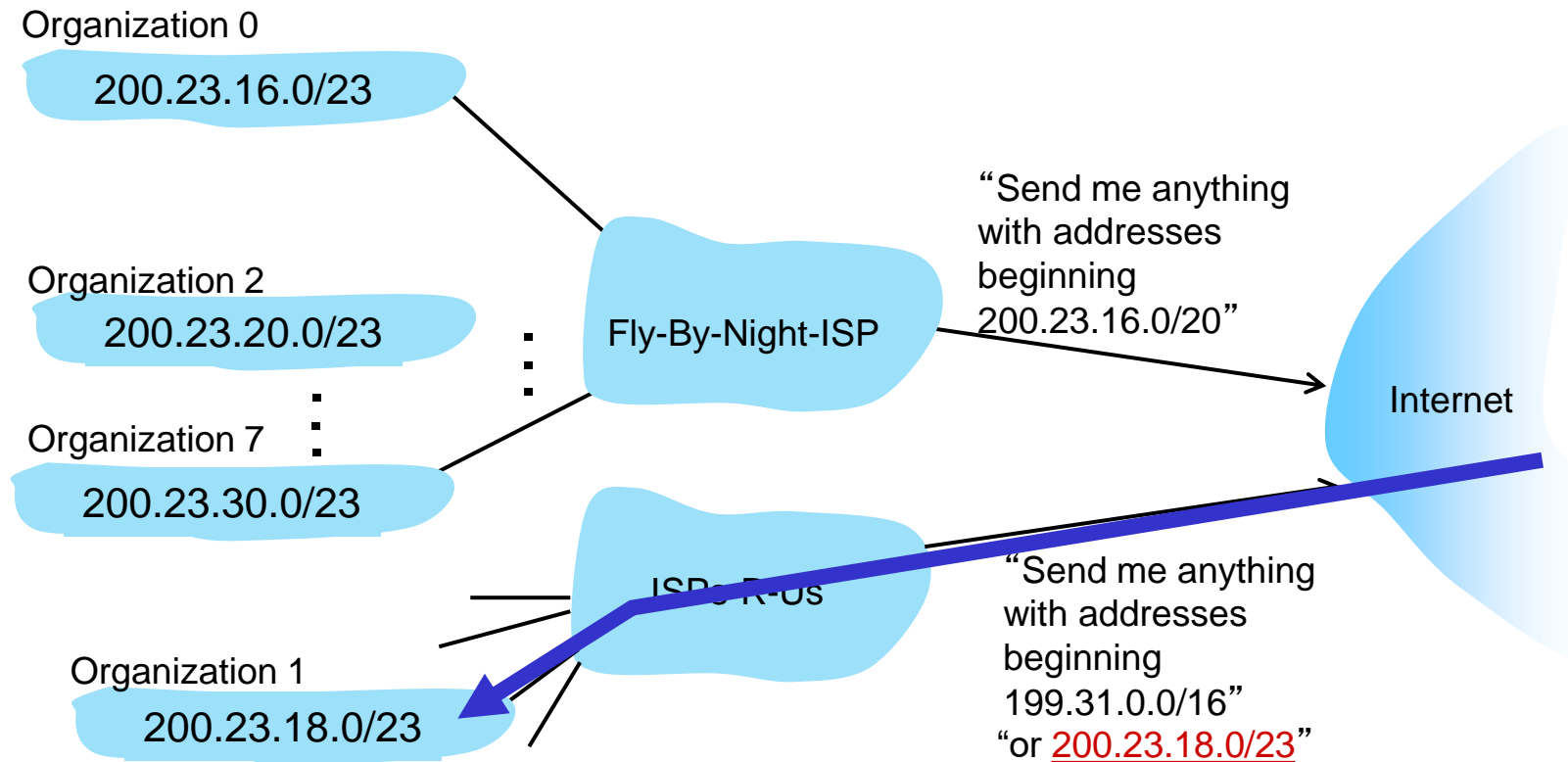
Internet

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

Organization 0
200.23.16.0/23

Organization 1
200.23.18.0/23

Organization 2
200.23.20.0/23

Organization 7
200.23.30.0/23

Fly-By-Night-ISP

"Send me anything with addresses beginning 200.23.16.0/20"

Internet

ISPs-R-Us

Organization 1
200.23.18.0/23

"Send me anything with addresses beginning 199.31.0.0/16"
"or 200.23.18.0/23"

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

Organization 0
200.23.16.0/23

Organization 2
200.23.20.0/23

Organization 7
200.23.30.0/23

Fly-By-Night-ISP

"Send me anything with addresses beginning 200.23.16.0/20"

Internet

ISPs-R-Us

Organization 1
200.23.18.0/23

"Send me anything with addresses beginning 199.31.0.0/16"
"or 200.23.18.0/23"

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?

- ICANN allocated last chunk of IPv4 addresses to RRs in 2011

- NAT (next) helps IPv4 address space exhaustion

- IPv6 has 128-bit address space

"Who the hell knew how much address space we needed?" Vint Cerf (reflecting on decision to make IPv4 address 32 bits long)
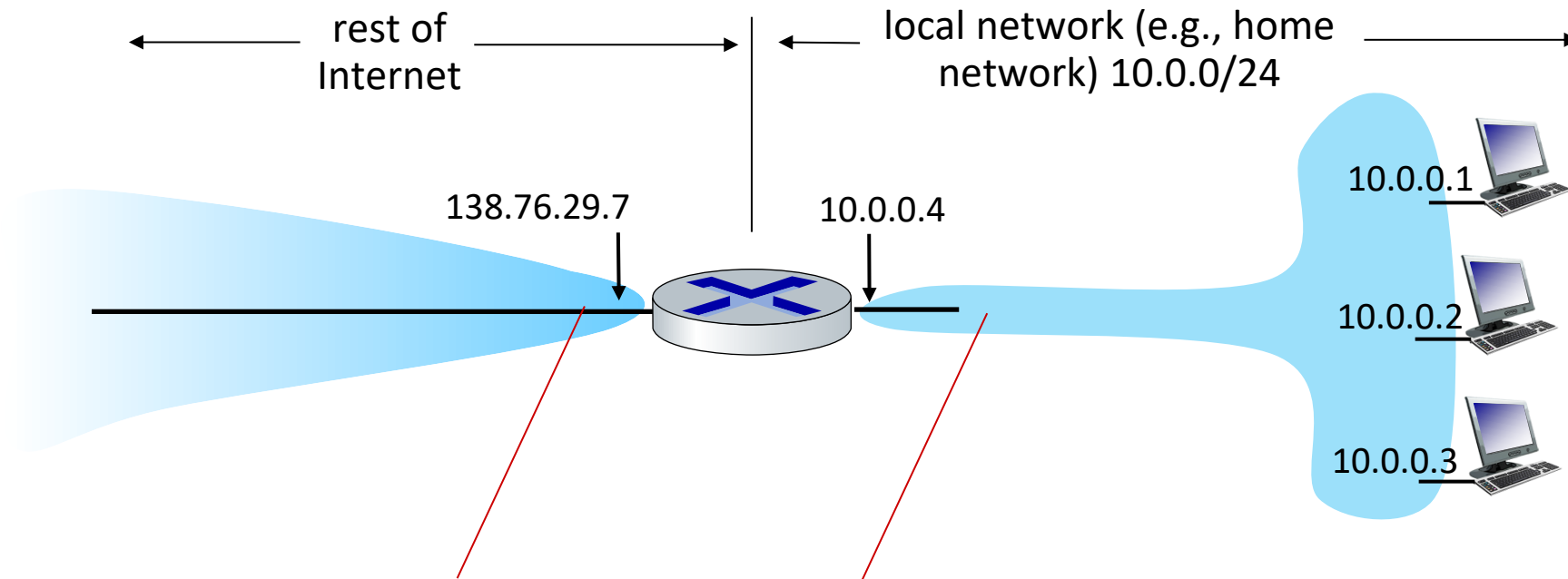
Network layer: "data plane" roadmap

- 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

NAT: network address translation

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

rest of Internet

local network (e.g., home network) 10.0.0/24

138.76.29.7

10.0.0.4

10.0.0.1

10.0.0.2

10.0.0.3

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 (as usual)

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
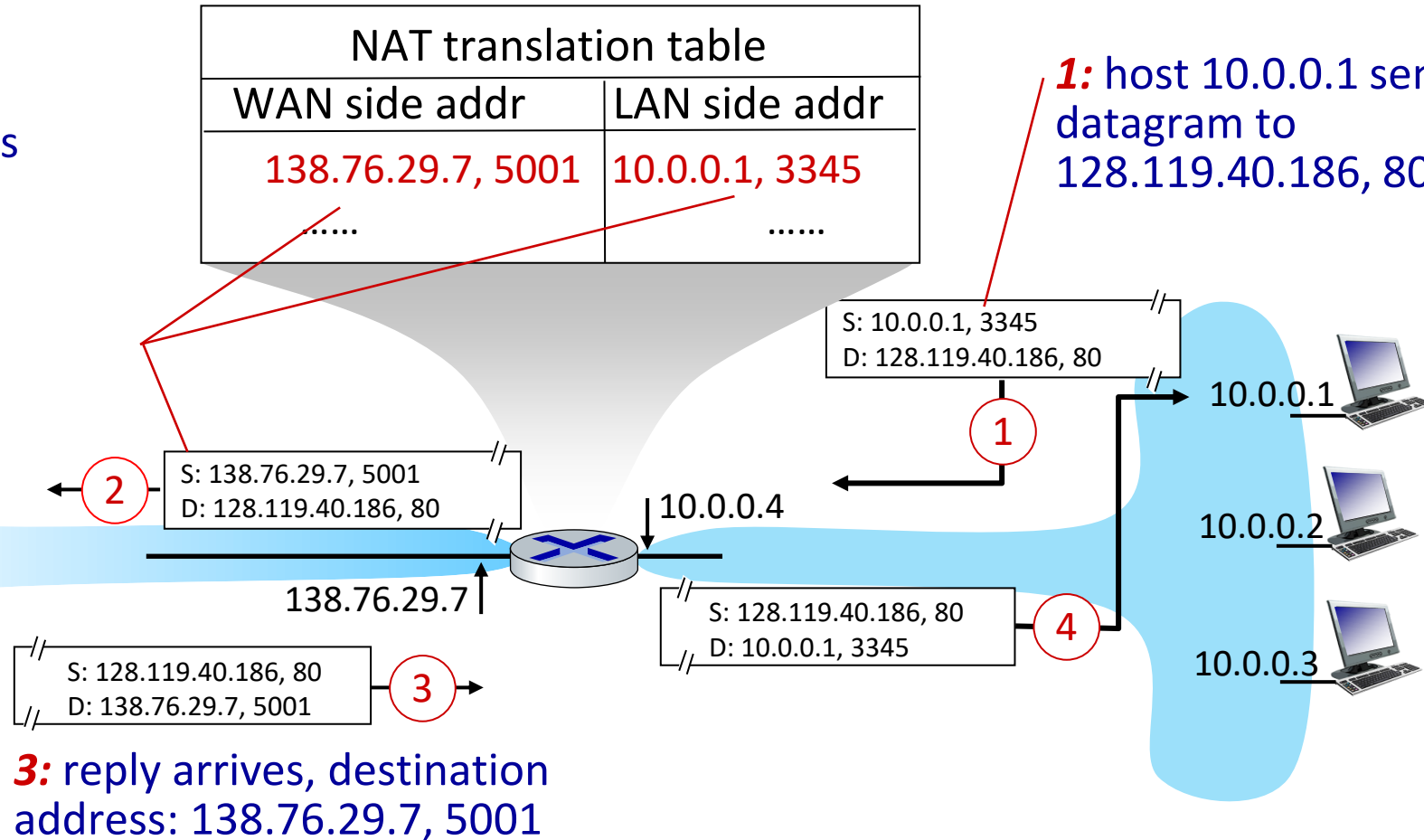
NAT: network address translation

■ NAT has been controversial:

- routers "should" only process up to layer 3

- address "shortage" should be solved by IPv6

- violates end-to-end argument (port # manipulation by network-layer device)

- NAT traversal: what if client wants to connect to server behind NAT?

■ but NAT is here to stay:

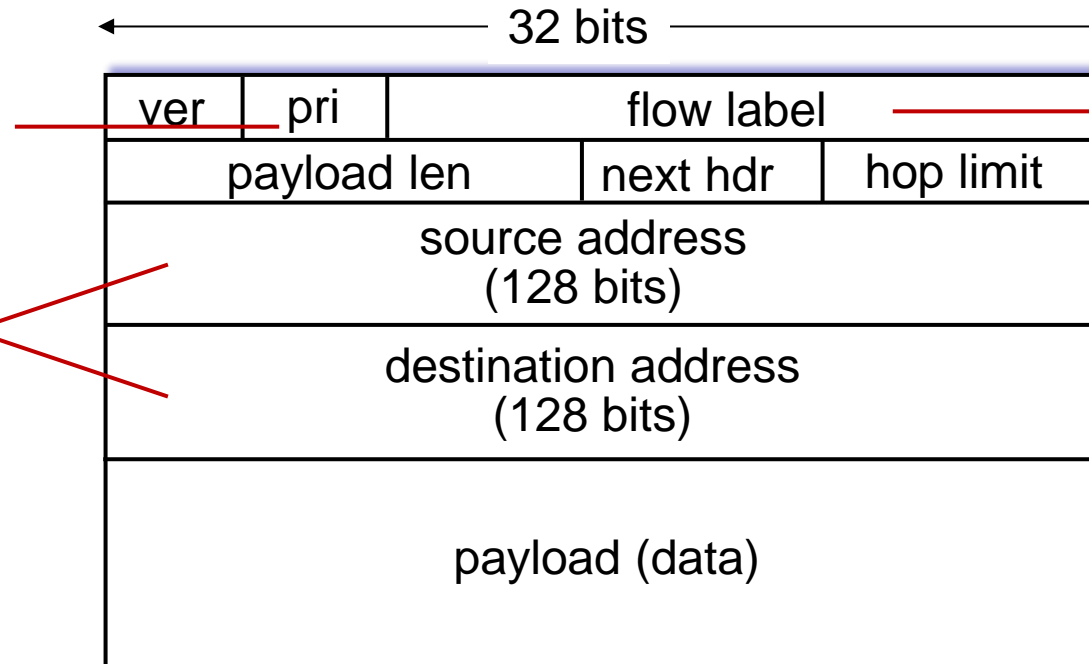- extensively used in home and institutional nets, 4G/5G cellular nets

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).

32 bits

ver	pri	flow label		
payload len			next hdr	hop limit
source address (128 bits)				
destination address (128 bits)				
payload (data)				

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)

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*:
    - many header fields can determine action
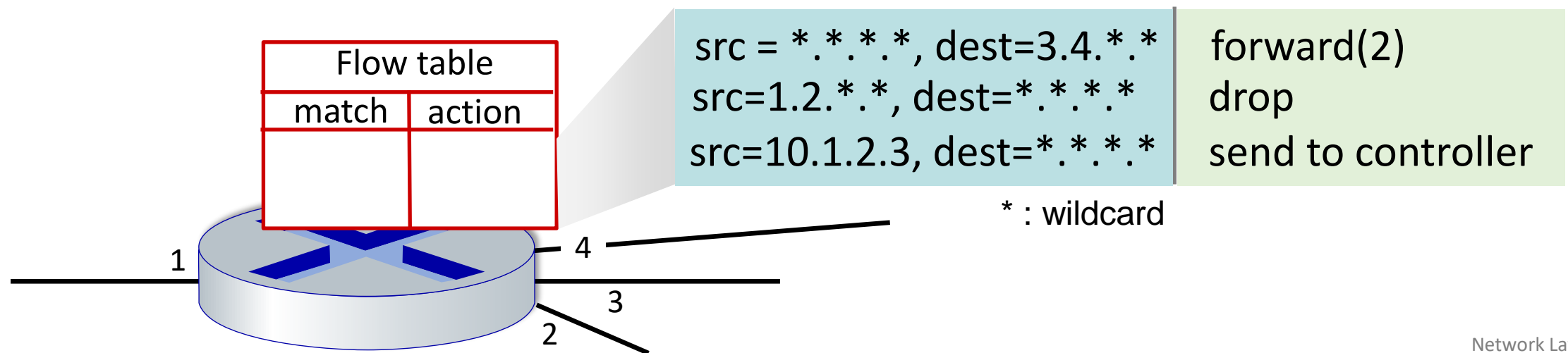   - many action possible: drop/copy/modify/log packet

forwarding table
(aka: flow table)

values in arriving
packet header

0111

1
2
3
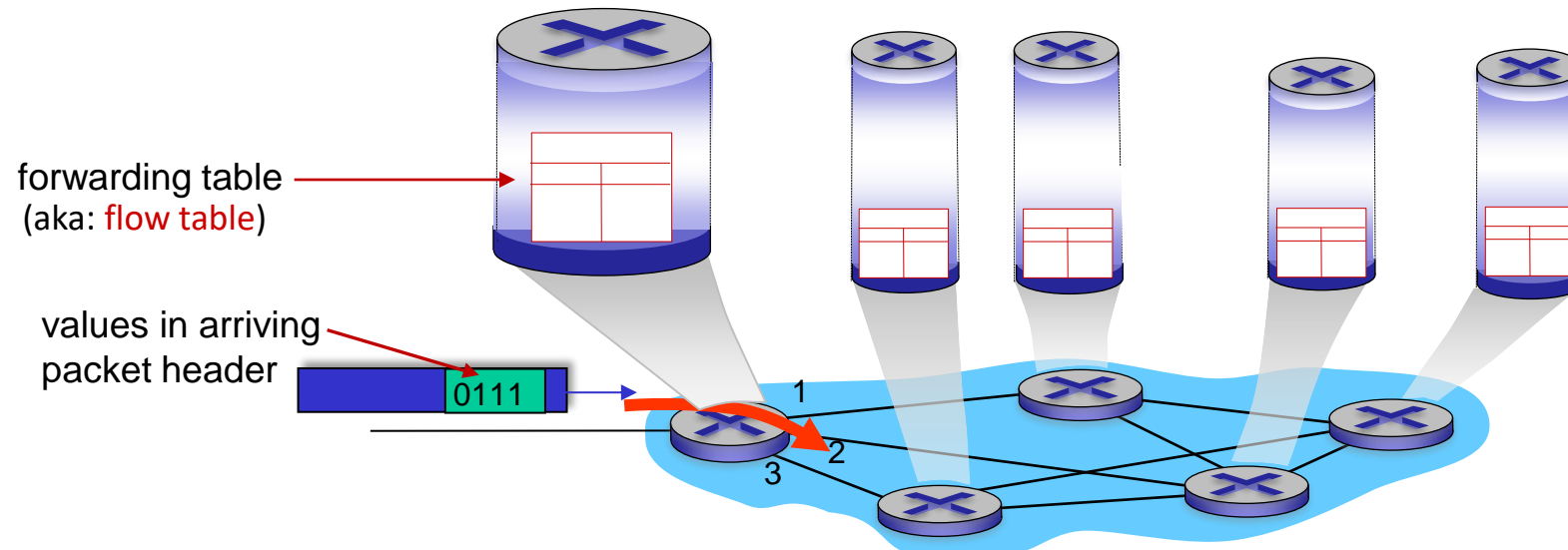
Flow table abstraction

- **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
  - priority: disambiguate overlapping patterns
  - counters: #bytes and #packets

Flow table	
match	action

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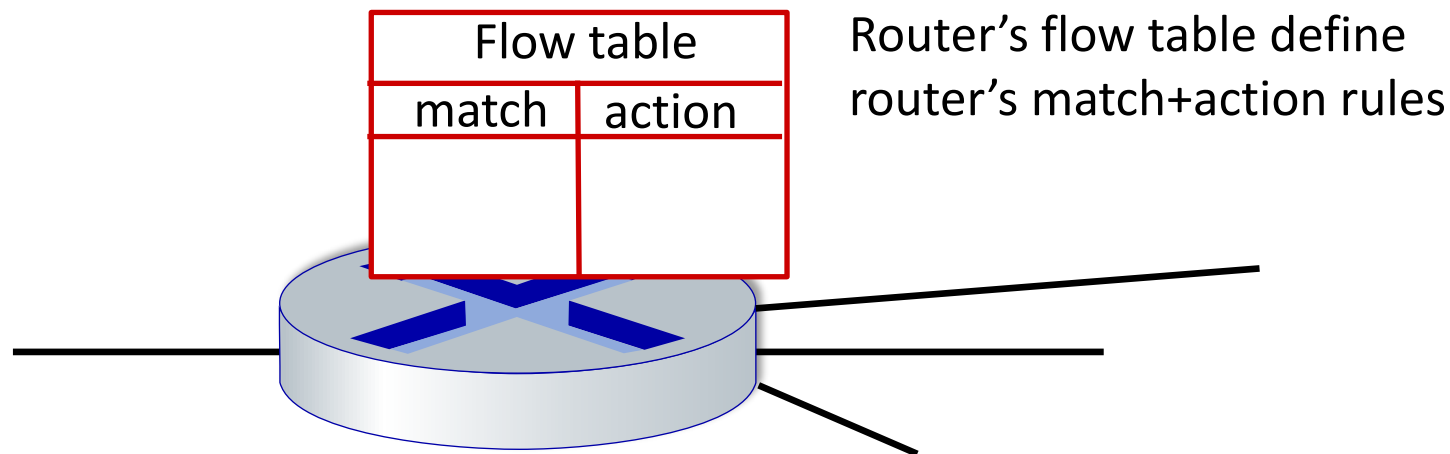
Flow table abstraction

- **flow:** defined by header 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
  - **priority:** disambiguate overlapping patterns
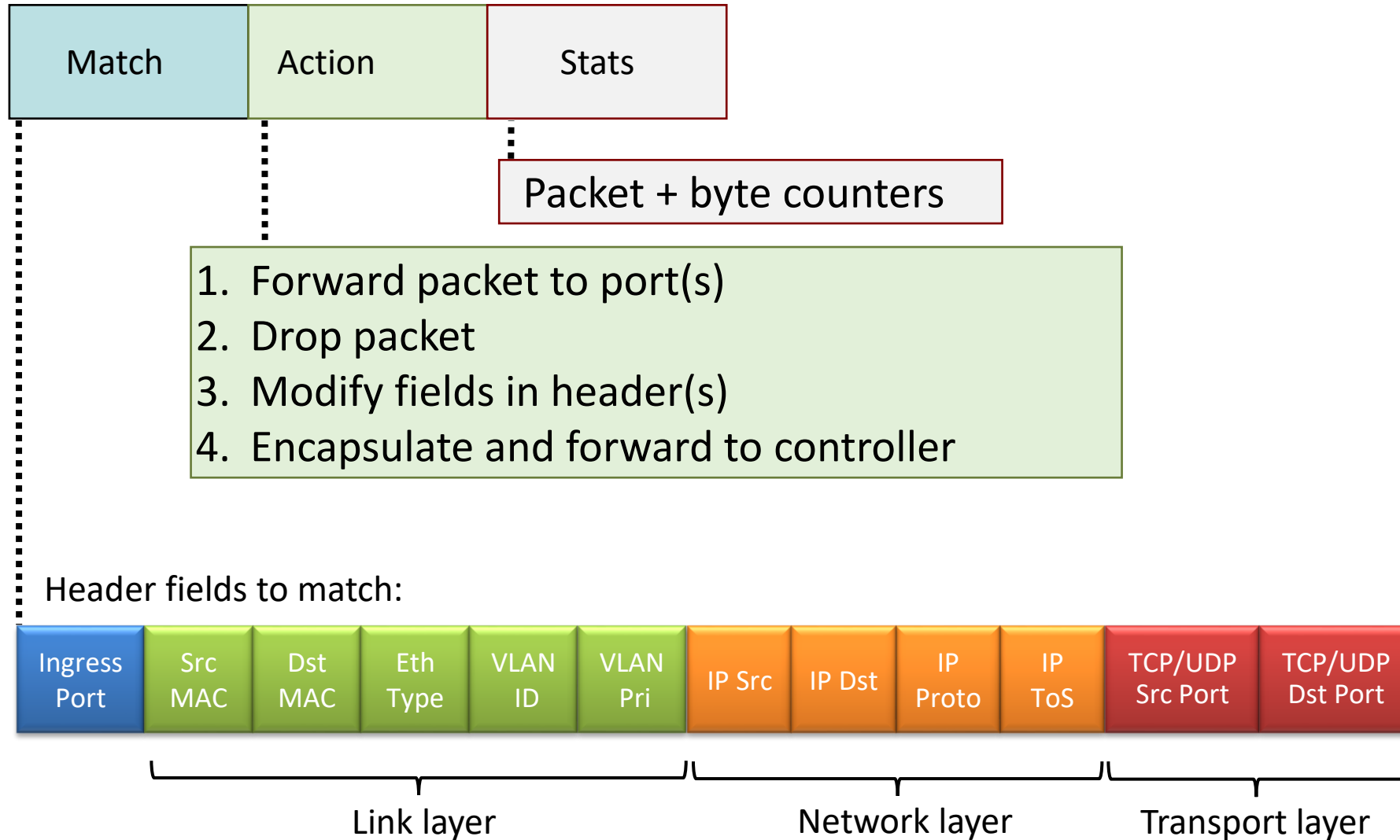 - **counters:** #bytes and #packets

Flow table

match	action

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

* : wildcard

1
4
3
2

OpenFlow: flow table entries

Match	Action	Stats

Packet + byte counters

1. Forward packet to port(s)
2. Drop packet
3. Modify fields in header(s)
4. Encapsulate and forward to controller

Header fields to match:

Ingress Port	Src MAC	Dst MAC	Eth Type	VLAN ID	VLAN Pri	IP Src	IP Dst	IP Proto	IP ToS	TCP/UDP Src Port	TCP/UDP Dst Port

Link layer | Network layer | Transport layer

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: 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
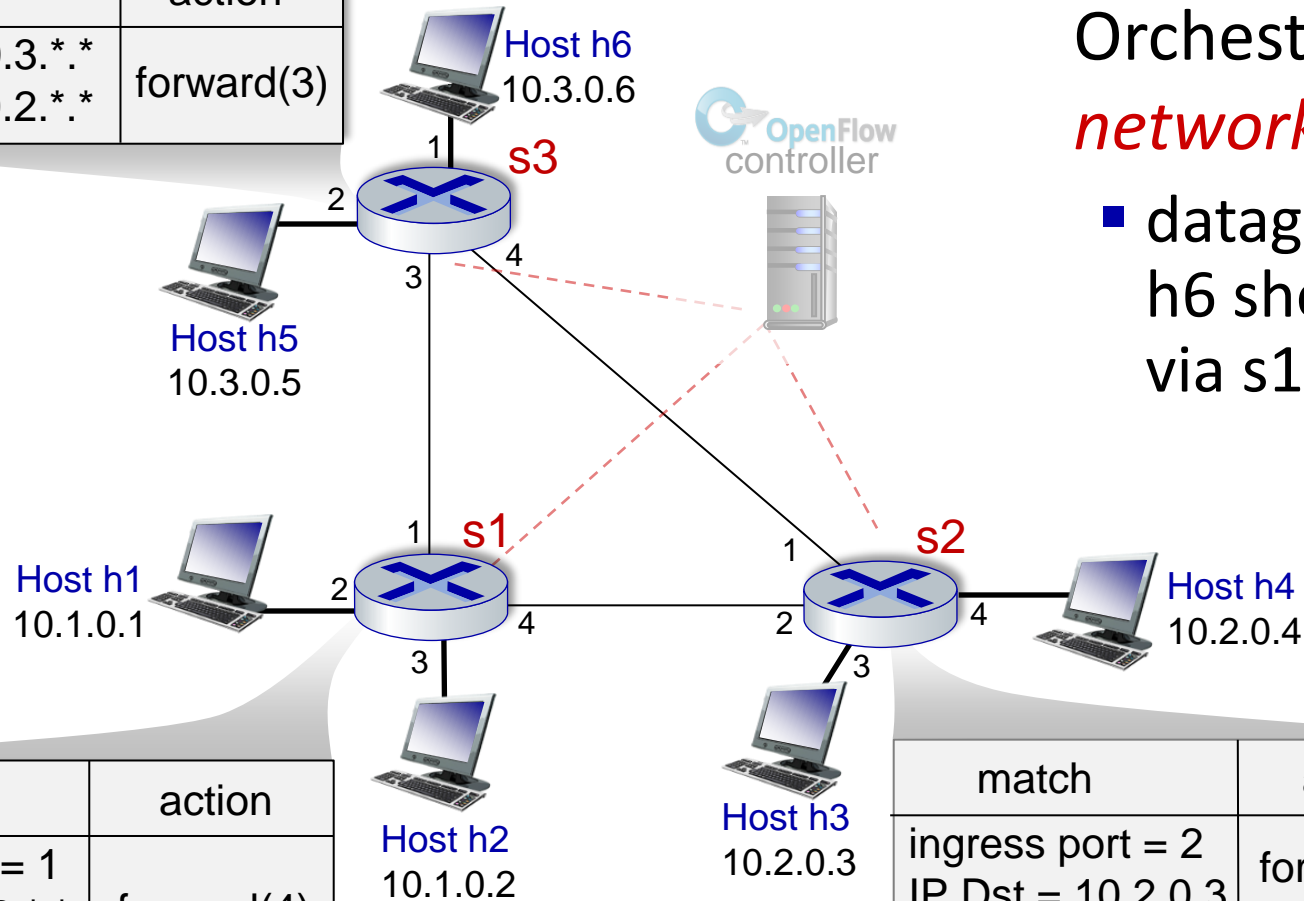- *action:* rewrite address and port

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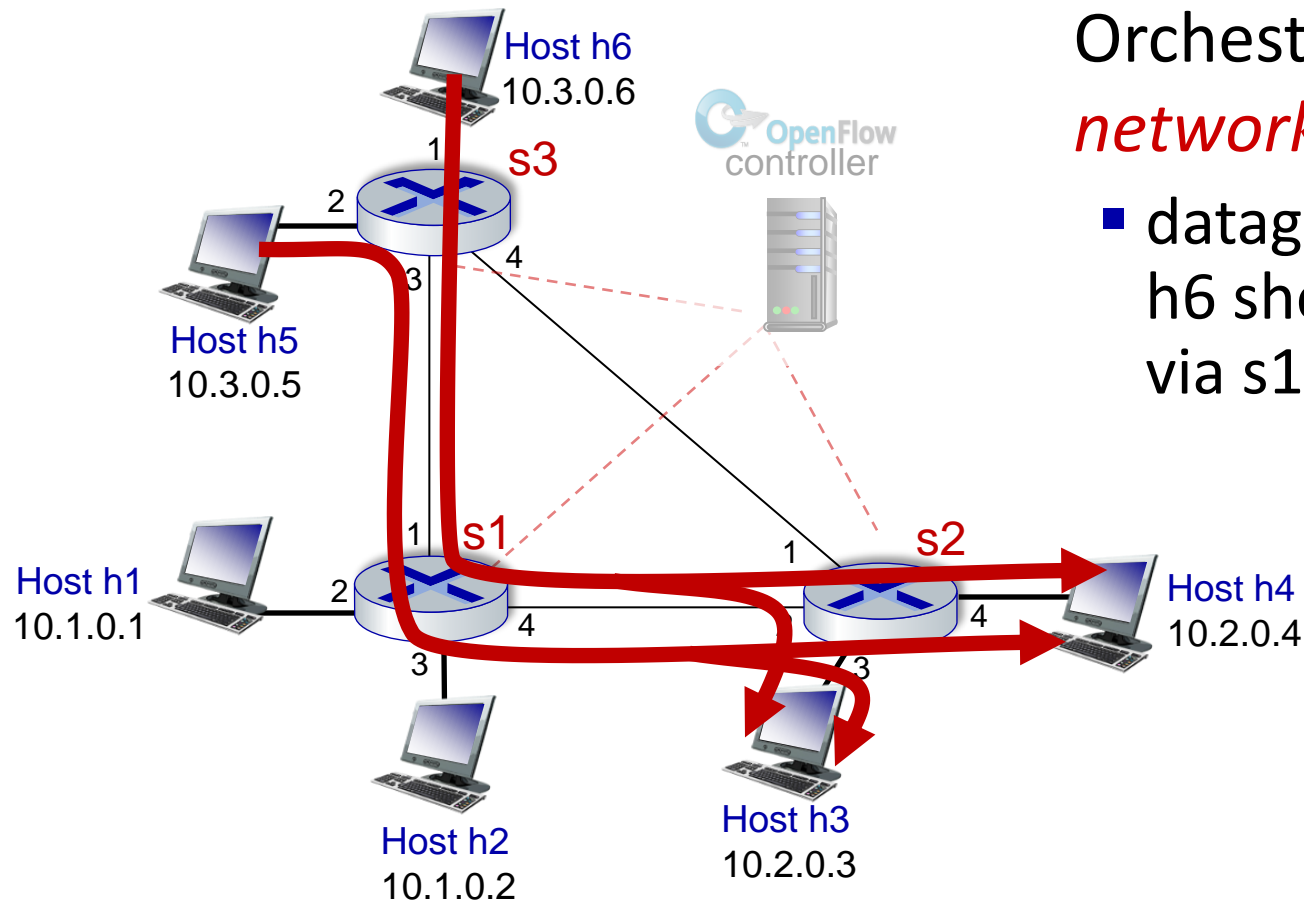
Orchestrated tables can create *network-wide* behavior, e.g.,:

- 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)

Host h6
10.3.0.6

OpenFlow
controller

s3

Host h5
10.3.0.5

s1

s2

Host h1
10.1.0.1

Host h4
10.2.0.4

Host h2
10.1.0.2

Host h3
10.2.0.3

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

Generalized forwarding: summary

- "match plus action" abstraction: match bits in arriving packet header(s) in any layers, take action
    - matching over many fields (link-, network-, transport-layer)
    - local actions: drop, forward, modify, or send matched packet to controller
    - "program" *network-wide* behaviors
- simple form of "network programmability"
    - programmable, per-packet "processing"
    - *historical roots:* active networking
    - *today:* more generalized programming: P4 (see p4.org).

Network layer: "data plane" roadmap

- Network layer: overview
- What's inside a router
- IP: the Internet Protocol
- Generalized Forwarding
- **Middleboxes**
  - middlebox functions
  - evolution, architectural principles of the Internet

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"

Middleboxes everywhere!

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

NAT: home, cellular, institutional

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

Application-specific: service providers, institutional, CDN

Caches: service provider, mobile, CDNs

national or global ISP

datacenter network

enterprise network

Middleboxes

- initially: proprietary (closed) hardware solutions
- move towards "whitebox" hardware implementing open API
    - move away from proprietary hardware solutions
    - programmable local actions via match+action
    - move towards innovation/differentiation in software
- SDN: (logically) centralized control and configuration management often in private/public cloud
- network functions virtualization (NFV): programmable services over white box networking, computation, storage

The IP hourglass

Internet's "thin waist":
- *one* network layer protocol: IP
- *must* be implemented by every (billions) of Internet-connected devices

HTTP   SMTP   RTP
QUIC   DASH   ...

TCP   UDP

IP

Ethernet PPP
PDCP WiFi Bluetooth ...

copper   radio   fiber
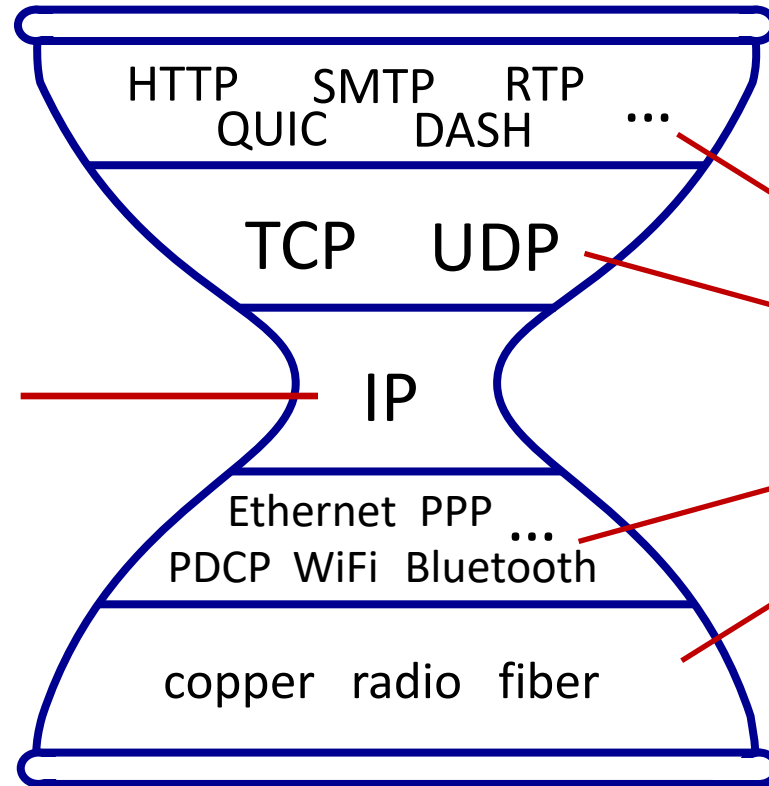
many protocols in physical, link, transport, and application layers

The IP hourglass, at middle age

Internet's middle age "love handles"?

- middleboxes, operating inside the network

HTTP   SMTP   RTP
QUIC   DASH   ...

TCP   UDP

caching
NAT   IP   NFV
Firewalls

Ethernet PPP
...
PDCP WiFi Bluetooth

copper   radio   fiber
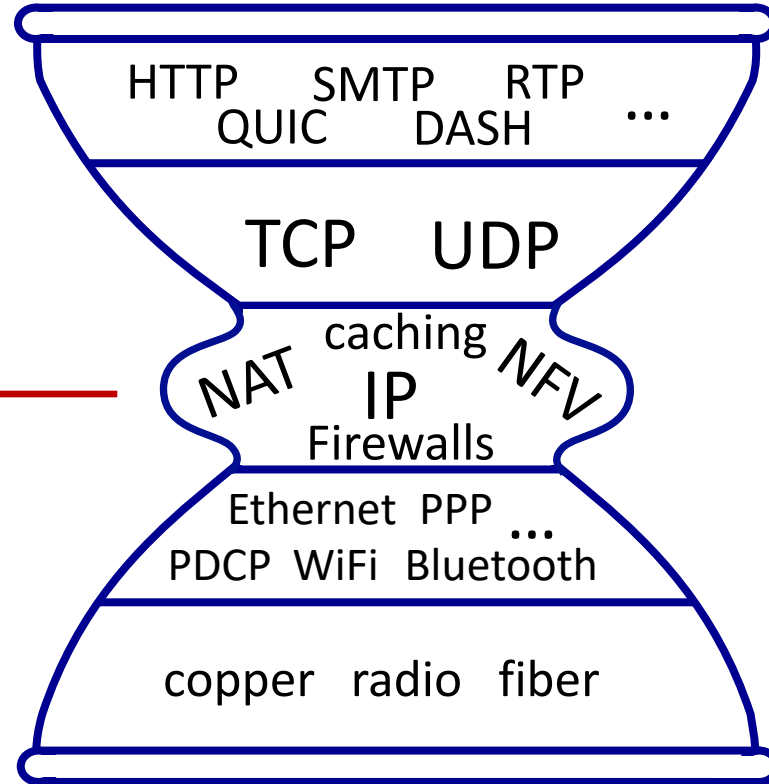
Architectural Principles of the Internet

RFC 1958

"Many members of the Internet community would argue that there is no architecture, but only a tradition, which was not written down for the first 25 years (or at least not by the IAB). However, in very general terms, the community believes that the goal is connectivity, the tool is the Internet Protocol, and the intelligence is end to end rather than hidden in the network."

Three cornerstone beliefs:
- simple connectivity
- IP protocol: that narrow waist
- intelligence, complexity at network edge

# The end-end argument

- some network functionality (e.g., reliable data transfer, congestion) can be implemented in network, or at network edge
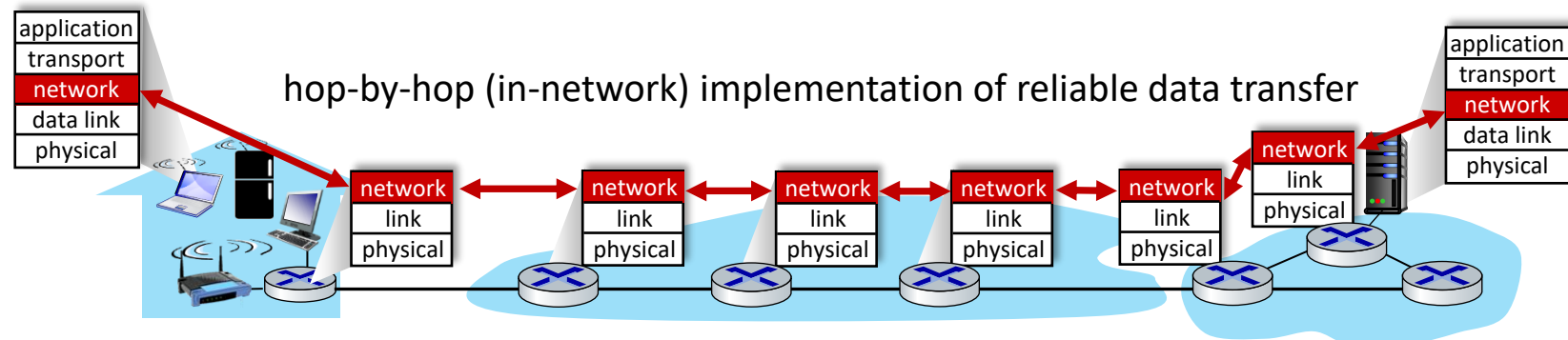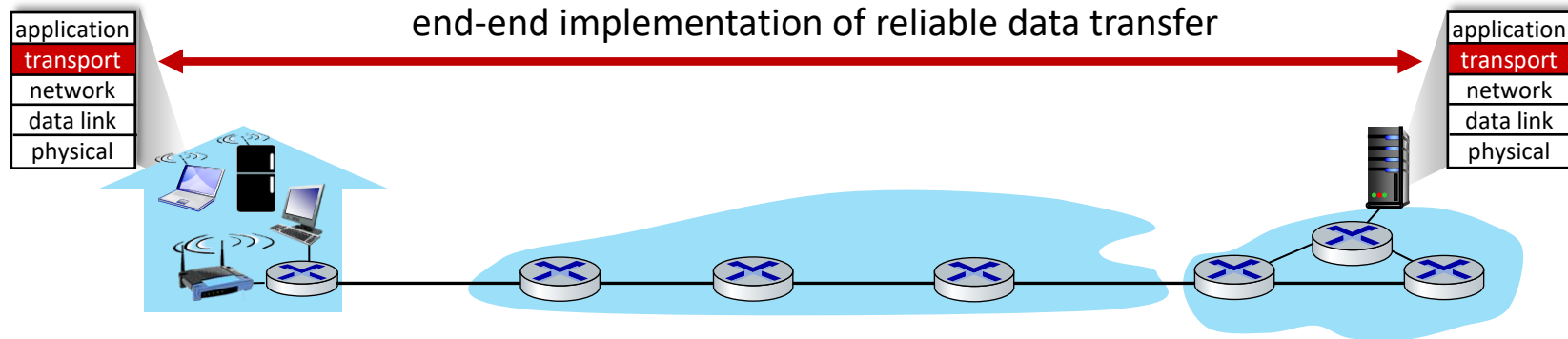
The end-end argument

- some network functionality (e.g., reliable data transfer, congestion) can be implemented in network, or at network edge

> "The function in question can completely and correctly be implemented only with the knowledge and help of the application standing at the end points of the communication system. Therefore, providing that questioned function as a feature of the communication system itself is not possible. (Sometimes an incomplete version of the function provided by the communication system may be useful as a performance enhancement.)
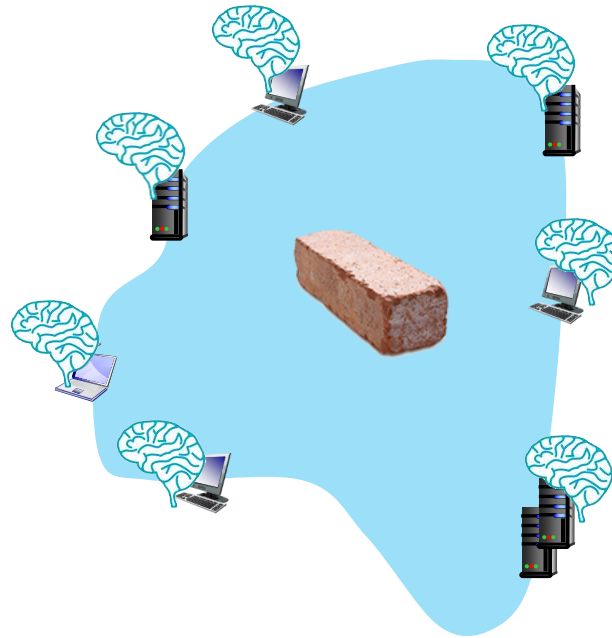>
> We call this line of reasoning against low-level function implementation the "end-to-end argument."
>
> — Saltzer, Reed, Clark 1981

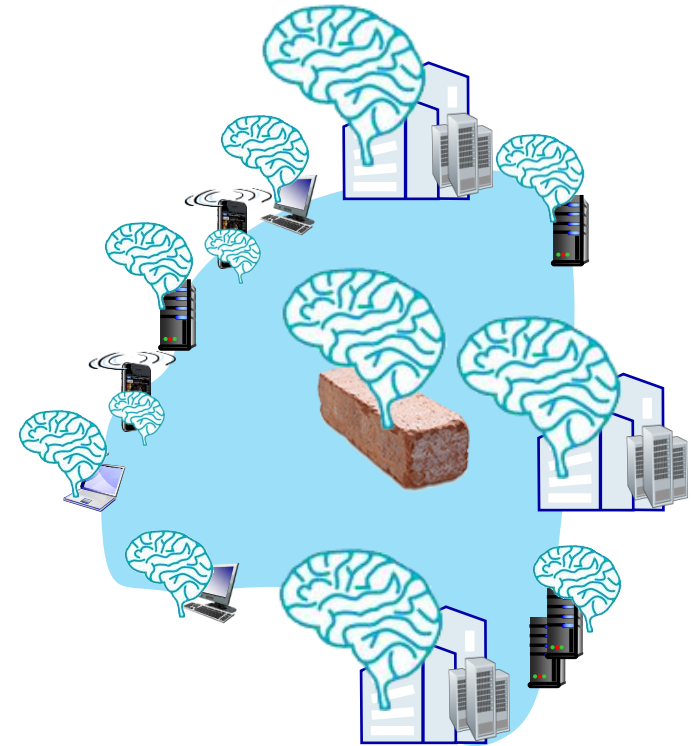# Where's the intelligence?
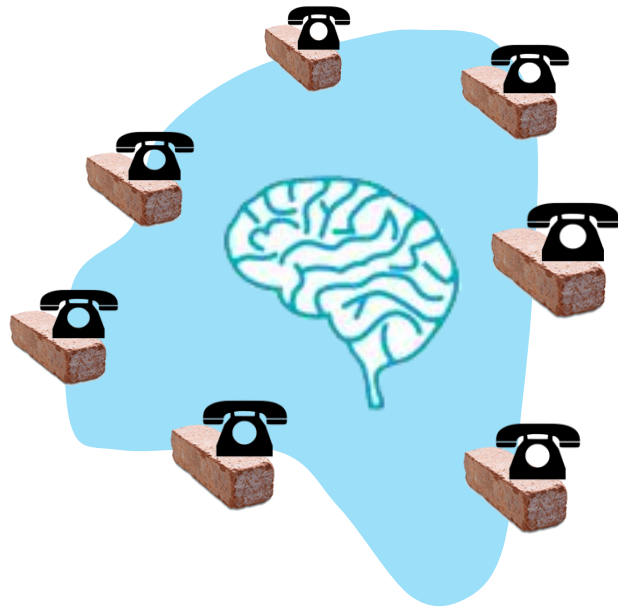
20th century phone net:
- intelligence/computing at network switches

Internet (pre-2005)
- intelligence, computing at edge

Internet (post-2005)
- programmable network devices
- intelligence, computing, massive application-level infrastructure at edge

Chapter 4: done!

- Network layer: overview
- What's inside a router
- IP: the Internet Protocol
- Generalized Forwarding, SDN
- Middleboxes

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

Answer: by the control plane (next chapter)