Chapter 4
Network Layer: Data Plane

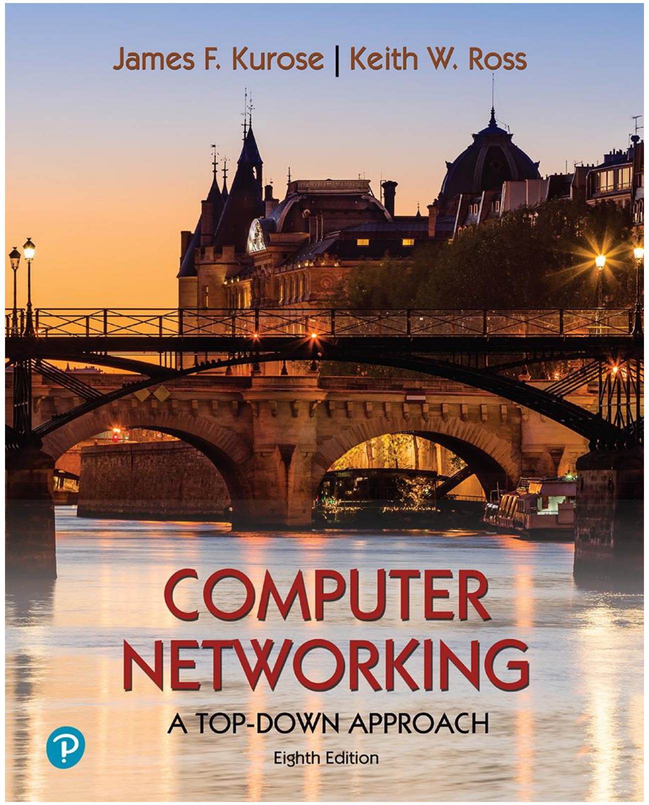
A note on the use of these PowerPoint slides:

We're making these slides freely available to all (faculty, students, readers). They're in PowerPoint form so you see the animations; and can add, modify, and delete slides (including this one) and slide content to suit your needs. They obviously represent a *lot* of work on our part. In return for use, we only ask the following:

- If you use these slides (e.g., in a class) that you mention their source (after all, we'd like people to use our book!)
- If you post any slides on a www site, that you note that they are adapted from (or perhaps identical to) our slides, and note our copyright of this material.

For a revision history, see the slide note for this page.

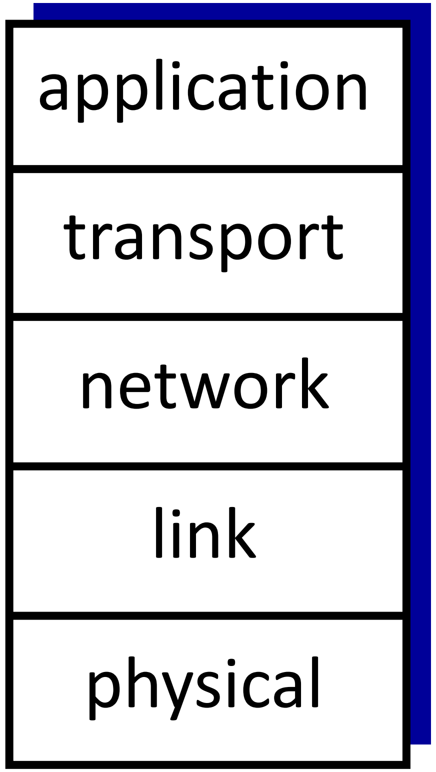
Thanks and enjoy! JFK/KWR



Computer Networking: A Top-Down Approach
8th edition
Jim Kurose, Keith Ross
Pearson, 2020

Internet protocol stack

- *application:* supporting network applications
  - IMAP, SMTP, HTTP
- *transport:* process-process data transfer
  - TCP, UDP
- *network:* routing of datagrams from source to destination
  - IP, routing protocols
- *link:* data transfer between neighboring network elements
  - Ethernet, 802.11 (WiFi), PPP
- *physical:* bits "on the wire"



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

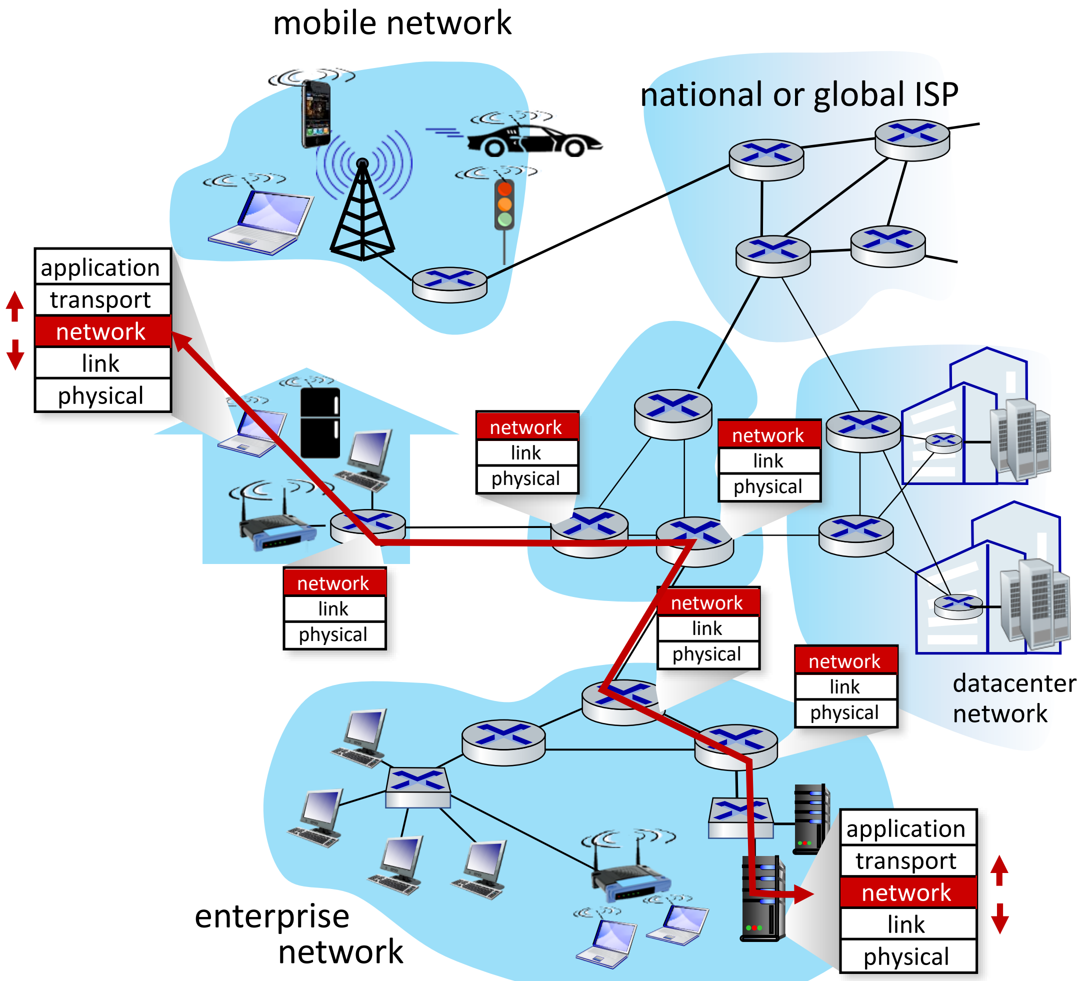
Network layer: "data plane" roadmap

- 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

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



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

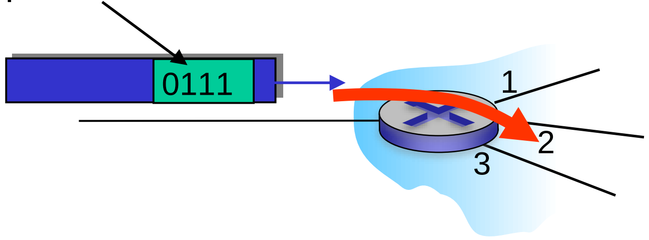
forwarding

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



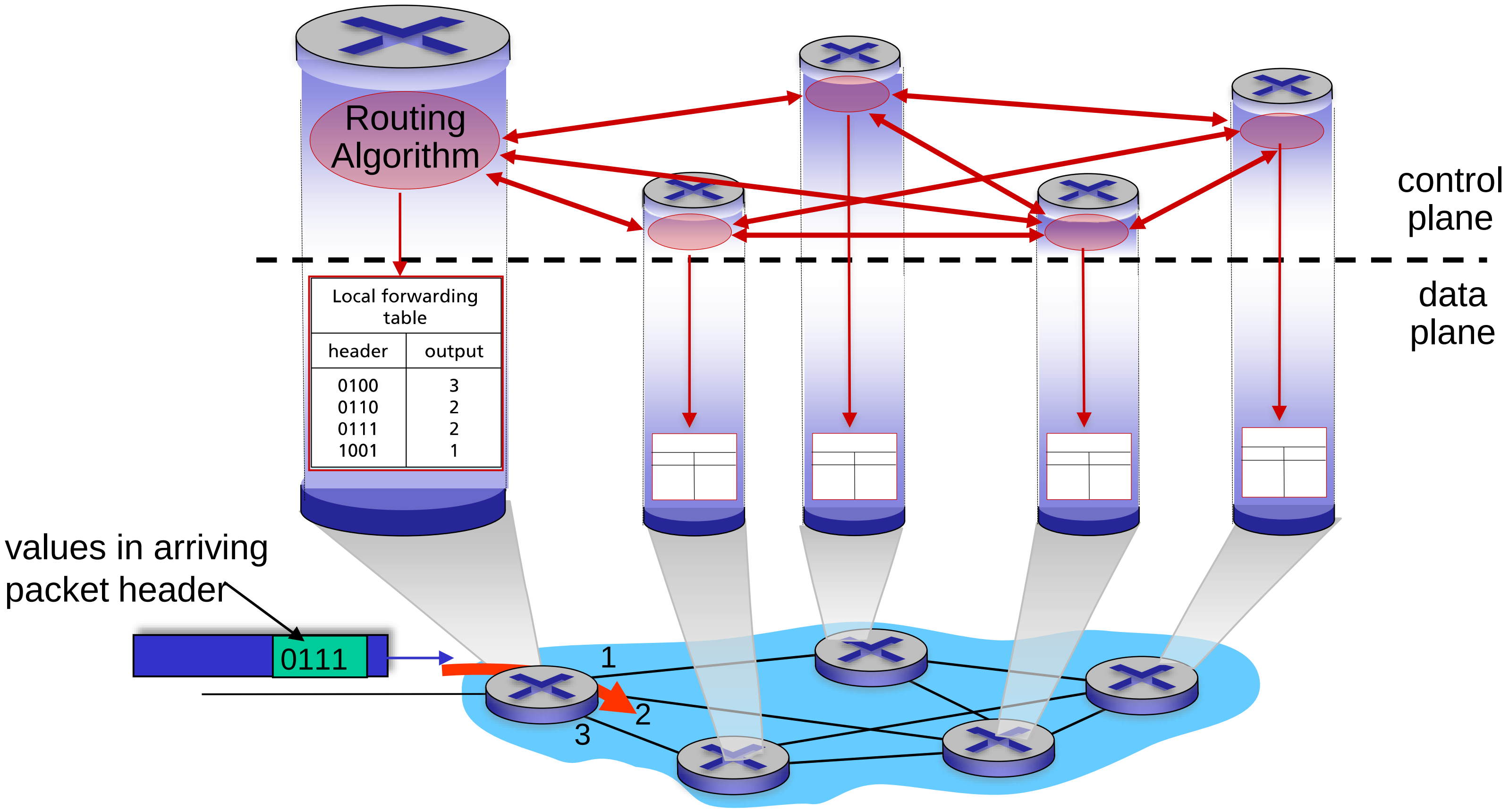
values in arriving packet header

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

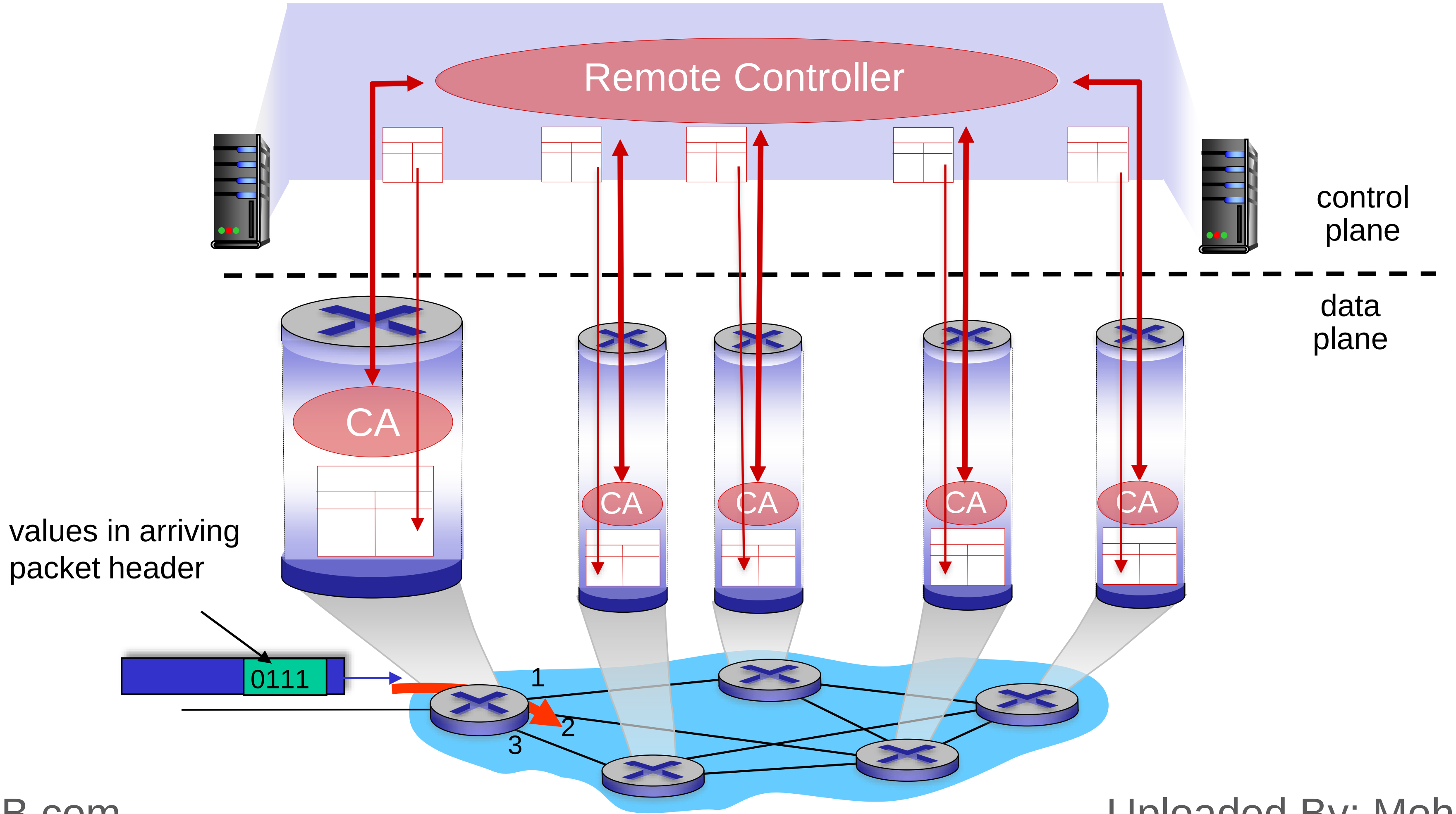


Local forwarding table

header	output
0100	3
0110	2
0111	2
1001	1

Software-Defined Networking (SDN) control plane

Remote controller computes, installs forwarding tables in routers



Network service model

Q: What *service model* for “channel” transporting datagrams from sender to receiver?

example services for *individual* datagrams:

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

example services for a *flow* of datagrams:

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

Network-layer service model

Network Architecture	Service Model	Quality of Service (QoS) Guarantees ?			
		Bandwidth	Loss	Order	Timing
Internet	best effort	none	no	no	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

Network-layer 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	Intserv Guaranteed (RFC 1633)	yes	yes	yes	yes
Internet	Diffserv (RFC 2475)	possible	possibly	possibly	no

Reflections on best-effort service:

- simplicity of mechanism has allowed Internet to be widely deployed adopted
- sufficient provisioning of bandwidth allows performance of real-time applications (e.g., interactive voice, video) to be “good enough” for “most of the time”
- replicated, application-layer distributed services (datacenters, content distribution networks) connecting close to clients’ networks, allow services to be provided from multiple locations
- congestion control of “elastic” services helps

It’s hard to argue with success of best-effort service model

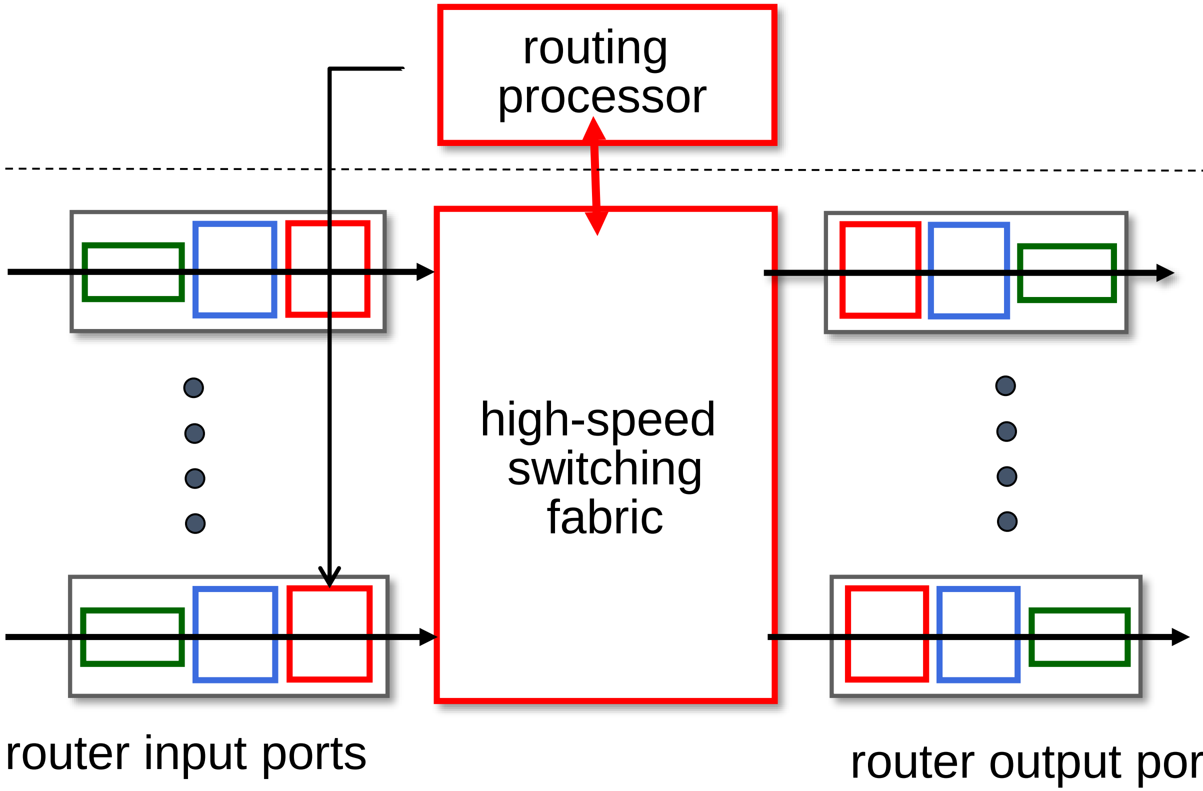
Network layer: "data plane" roadmap

- 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

Router architecture overview

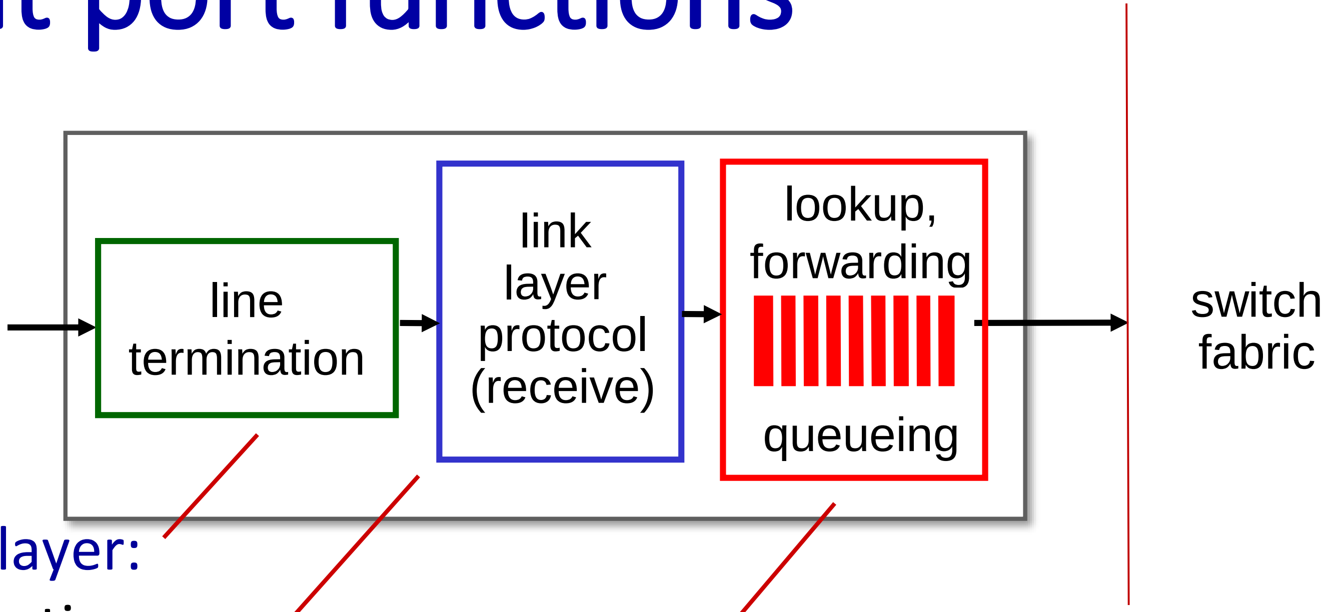
high-level view of generic router architecture:



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

forwarding data plane
(hardware) operates
in nanosecond
timeframe

Input port functions



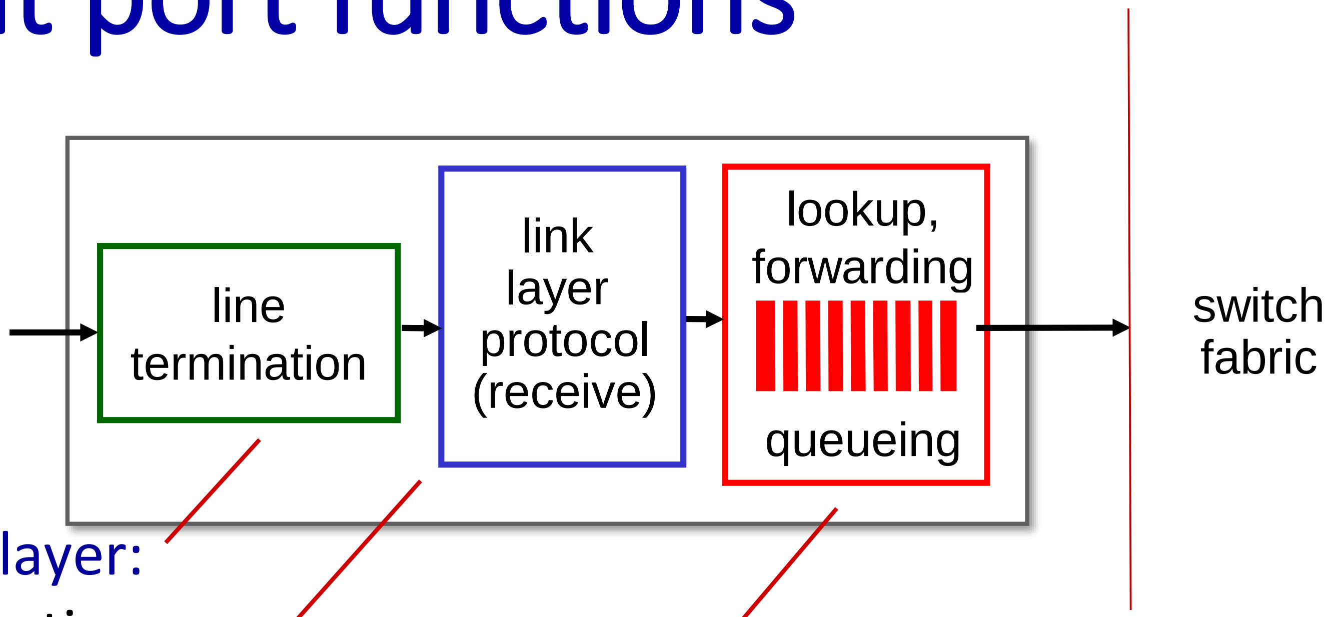
physical layer: bit-level reception

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

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

Input port functions



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

Destination-based forwarding

forwarding table

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

Q: but what happens if ranges don't divide up so nicely?

Longest prefix matching

longest prefix 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?

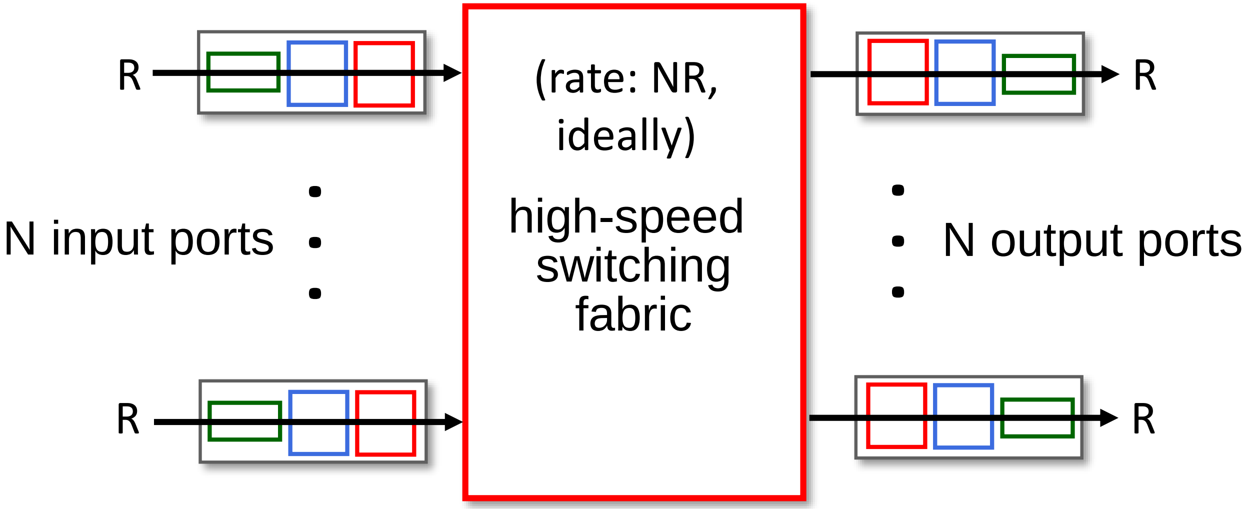
11001000 00010111 00011000 10101010 which interface?

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

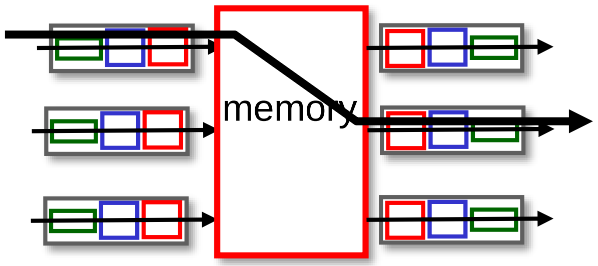
Switching fabrics

- 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

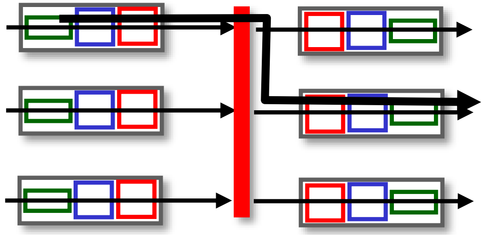


Switching fabrics

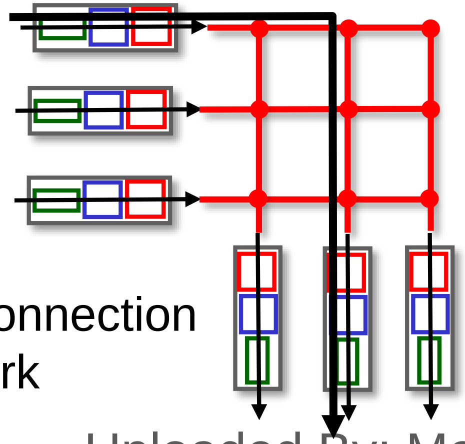
- transfer packet from input link to appropriate output link
- switching rate: rate at which packets can be transfer from inputs to outputs
  - often measured as multiple of input/output line rate
  - N inputs: switching rate N times line rate desirable
- three major types of switching fabrics:



memory



bus

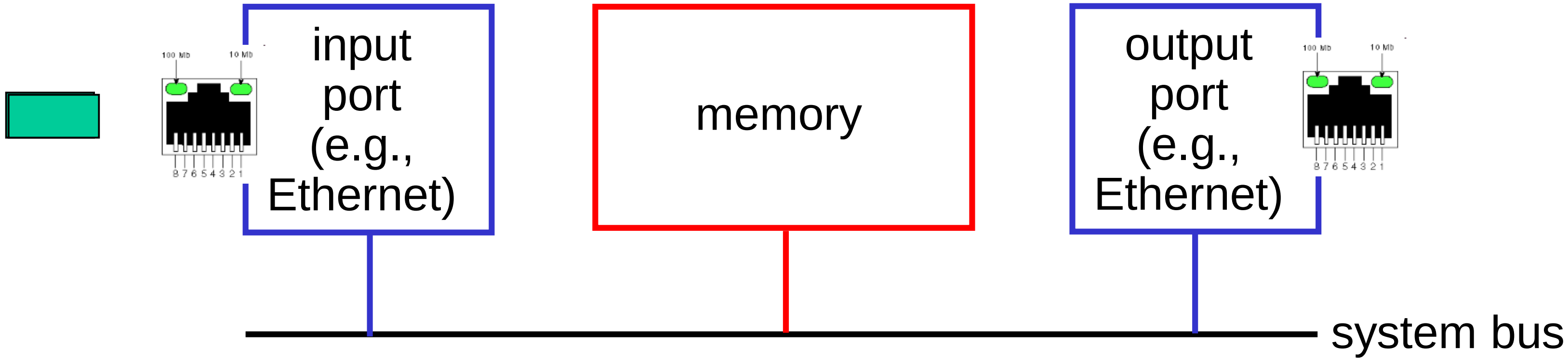


interconnection
network

Switching via memory

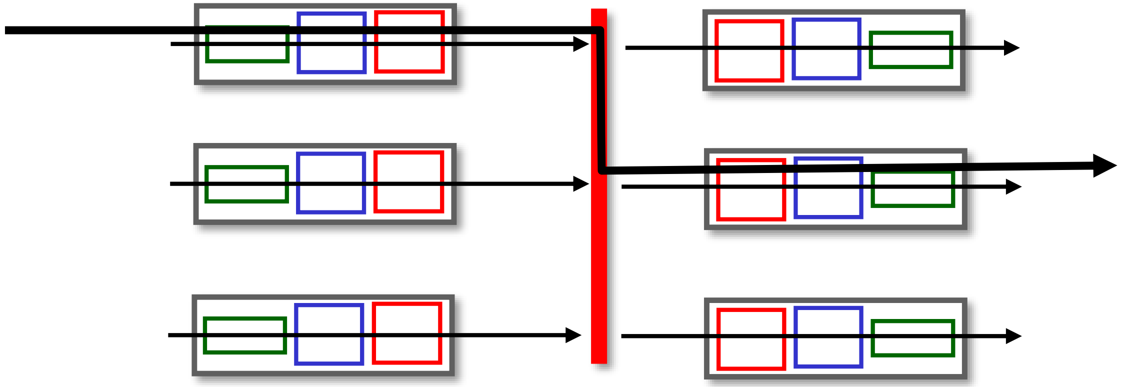
first generation routers:

- traditional computers with switching under direct control of CPU
- packet copied to system's memory
- speed limited by memory bandwidth (2 bus crossings per datagram)



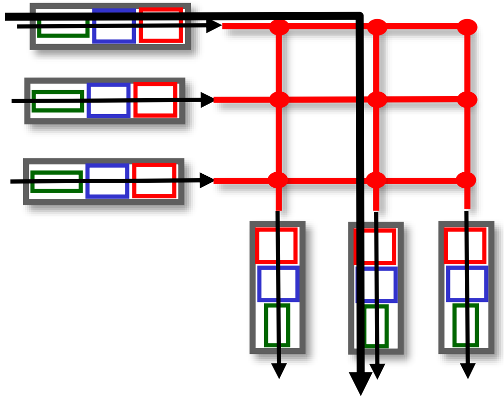
Switching via a bus

- datagram from input port memory to output port memory via a shared bus
- *bus contention:* switching speed limited by bus bandwidth
- 32 Gbps bus, Cisco 5600: sufficient speed for access routers

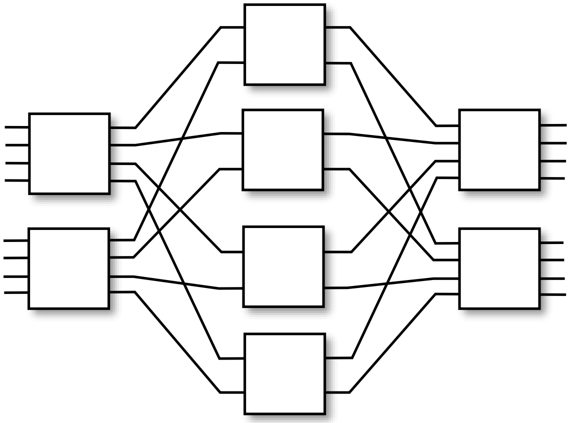


Switching via interconnection network

- Crossbar, Clos networks, other interconnection nets initially developed to connect processors in multiprocessor
- multistage switch: *nxn* switch from multiple stages of smaller switches
- exploiting parallelism:
  - fragment datagram into fixed length cells on entry
  - switch cells through the fabric, reassemble datagram at exit



3x3 crossbar

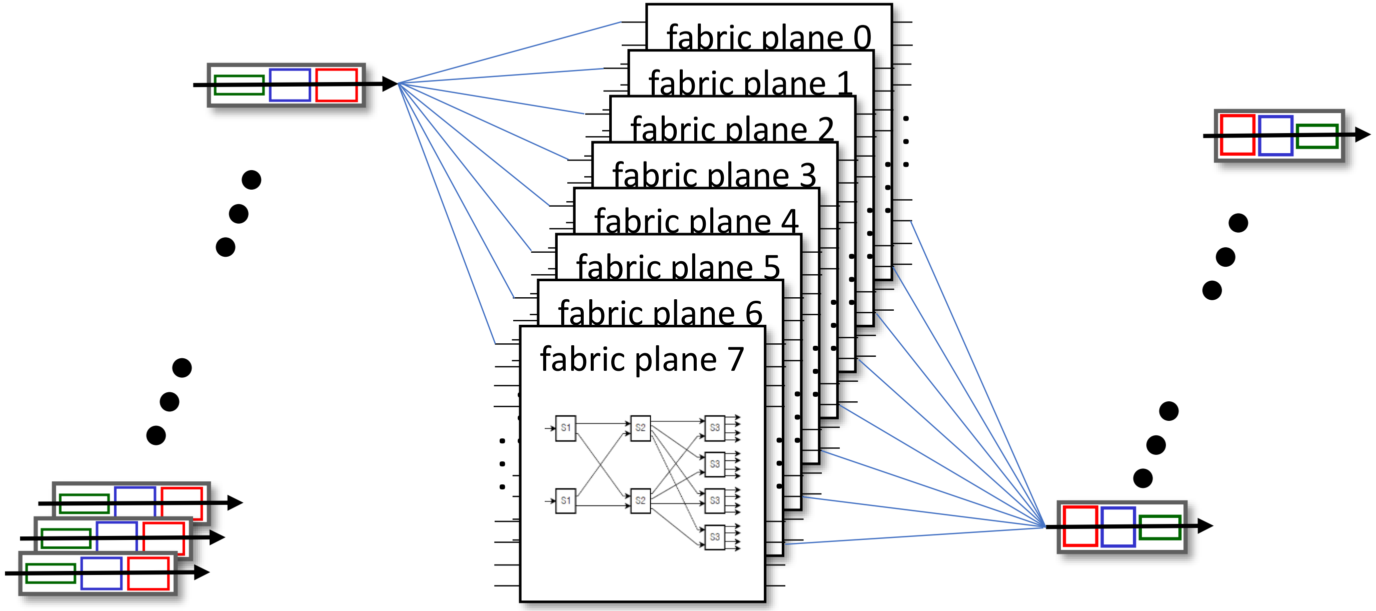


8x8 multistage switch
built from smaller-sized switches

Switching via interconnection network

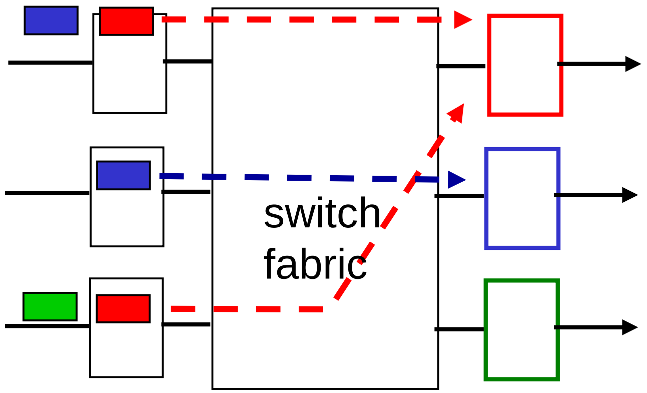
- scaling, using multiple switching "planes" in parallel:
  - speedup, scaleup via parallelism

- Cisco CRS router:
  - basic unit: 8 switching planes
  - each plane: 3-stage interconnection network
  - up to 100's Tbps switching capacity

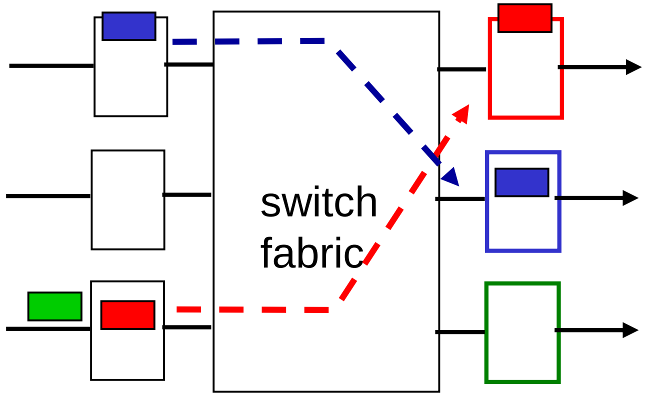


Input port queuing

- 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



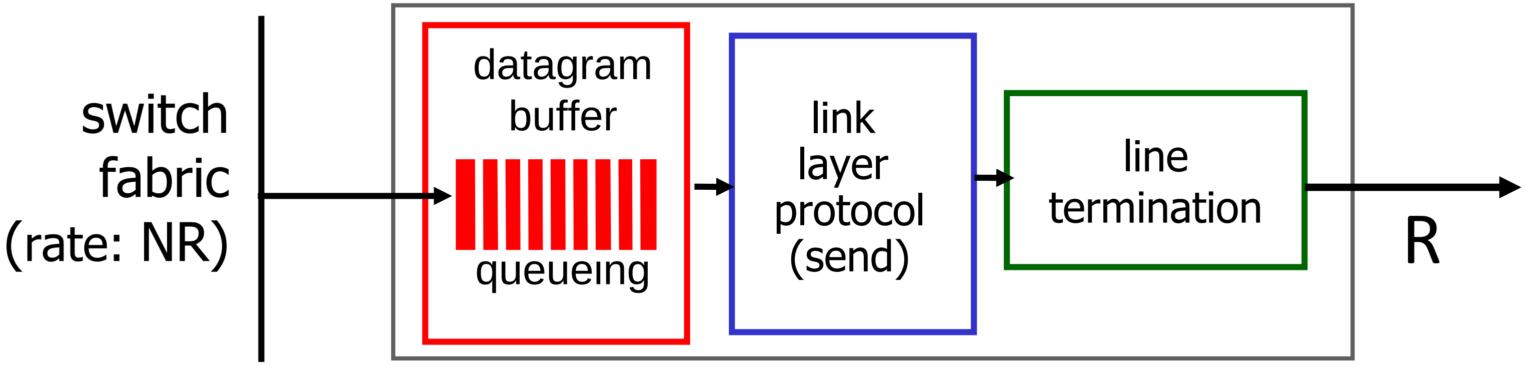
output port contention: only one red datagram can be transferred. lower red packet is *blocked*



one packet time later: green packet experiences HOL blocking

Output port queuing

This is a really important slide



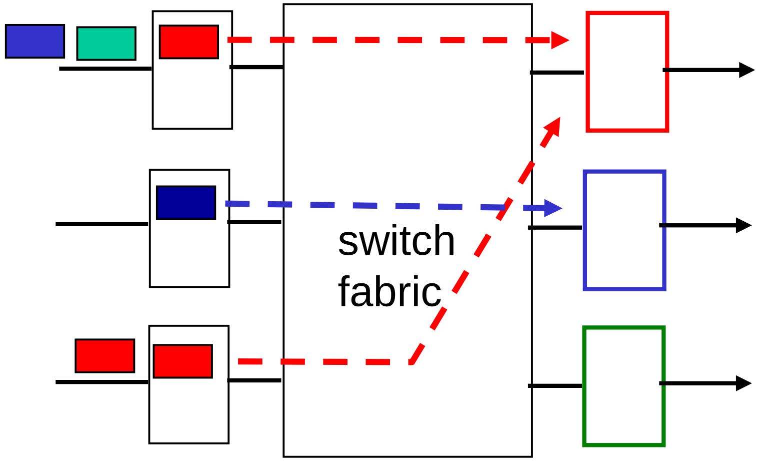
- *Buffering* required when datagrams arrive from fabric faster than link transmission rate. *Drop policy:* which datagrams to drop if no free buffers?

Datagrams can be lost due to congestion, lack of buffers

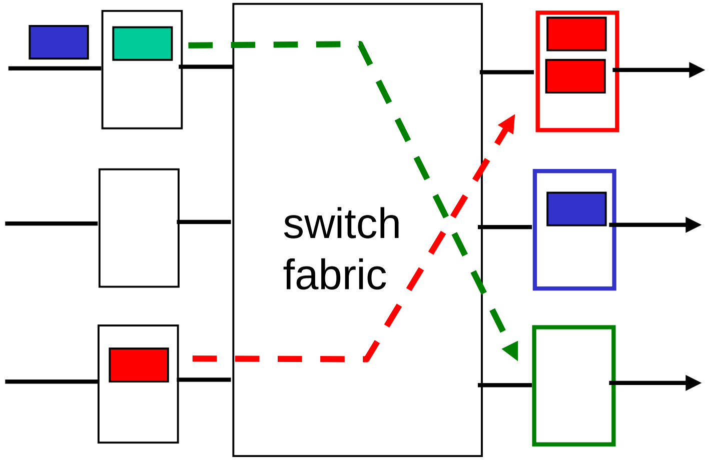
- *Scheduling discipline* chooses among queued datagrams for transmission

Priority scheduling – who gets best performance, network neutrality

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!*

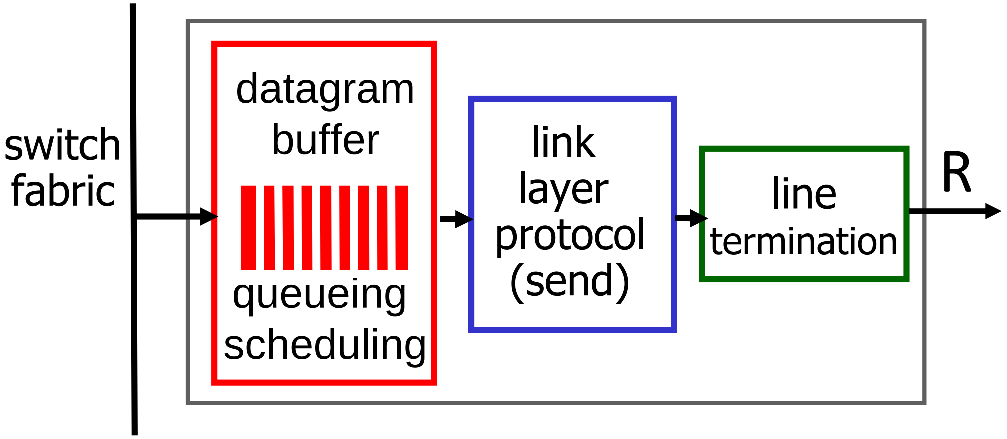
How much buffering?

- 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

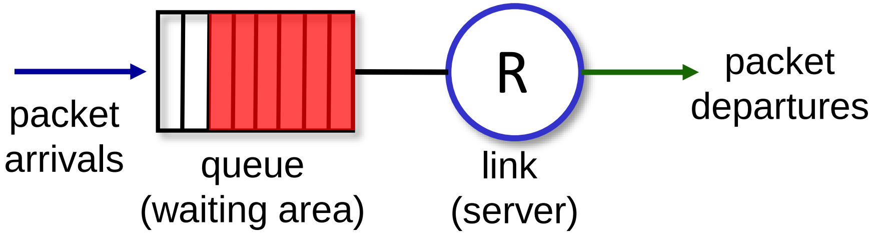
$$\frac{\mathrm{RTT} \cdot C}{\sqrt{N}}$$

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

Buffer Management



Abstraction: queue



buffer management:

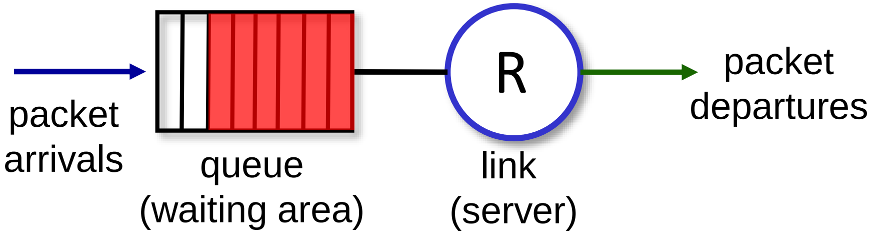
- drop: which packet to add, drop when buffers are full
  - tail drop: drop arriving packet
  - priority: drop/remove on priority basis
- marking: which packets to mark to signal congestion (ECN, RED)

Packet Scheduling: FCFS

packet scheduling: deciding which packet to send next on link

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

Abstraction: queue



packet arrivals

queue (waiting area)

R

link (server)

packet departures

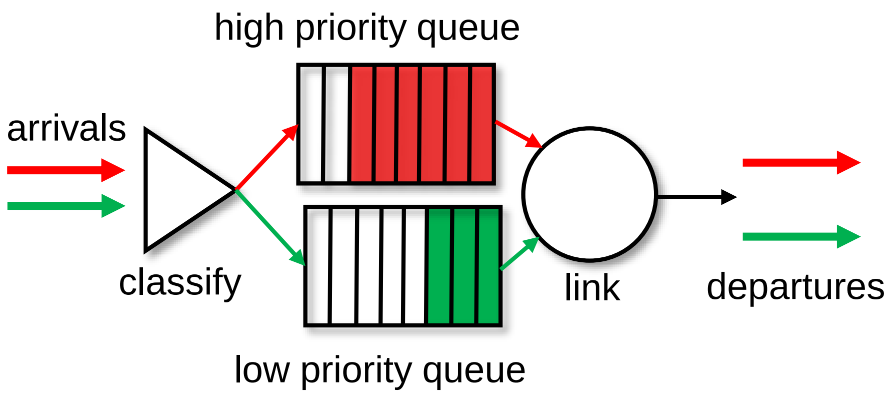
FCFS: packets transmitted in order of arrival to output port

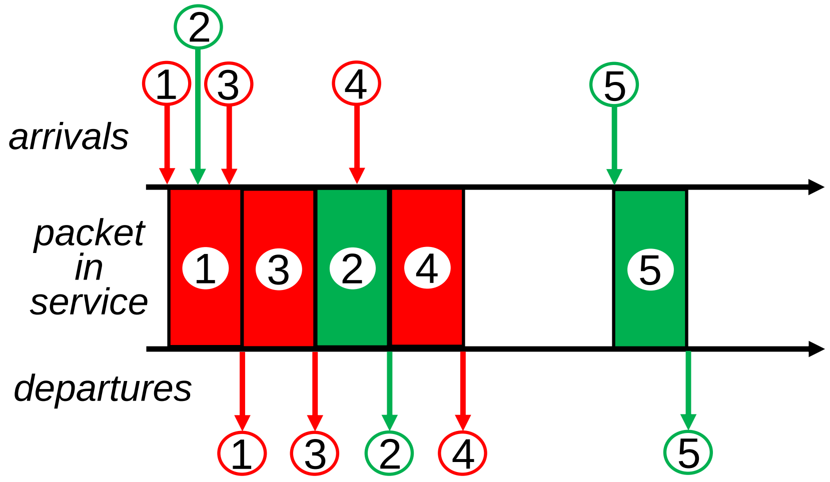
- also known as: First-in-first-out (FIFO)
- real world examples?

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

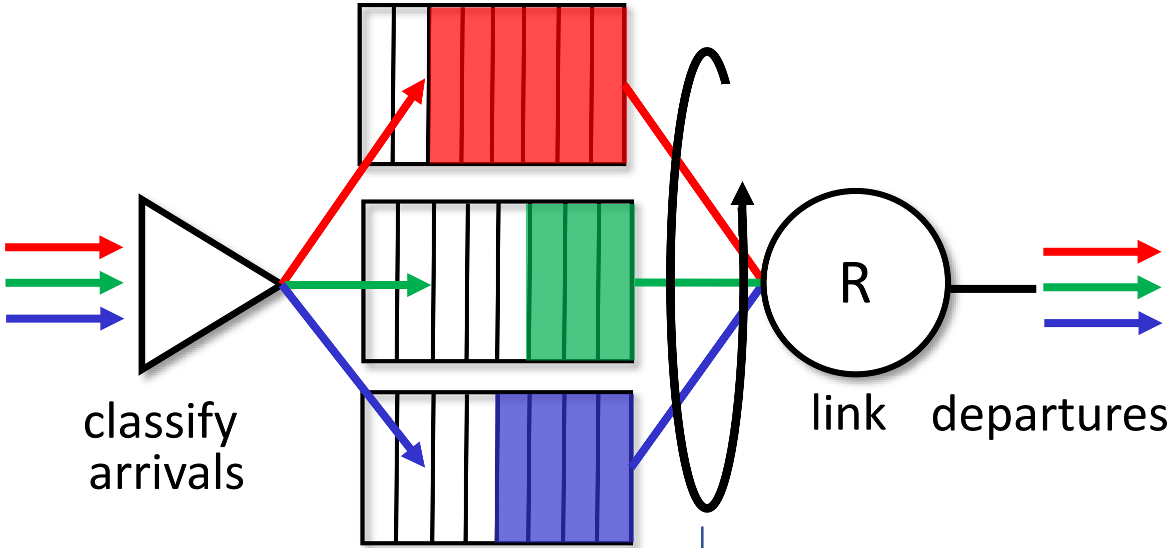




Scheduling policies: round robin

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



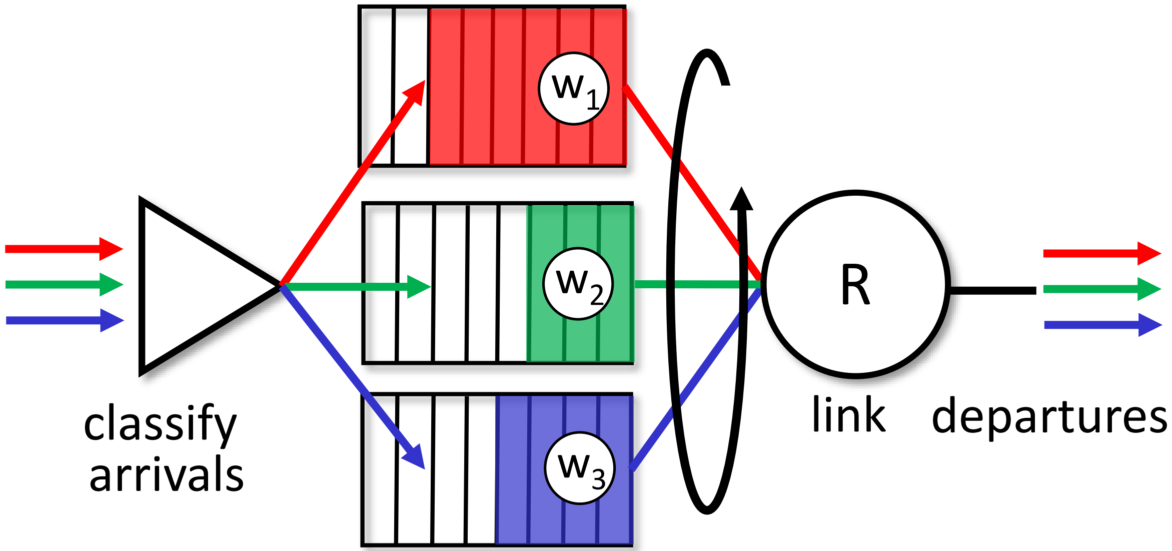
Scheduling policies: weighted fair queueing

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 (per-traffic-class)



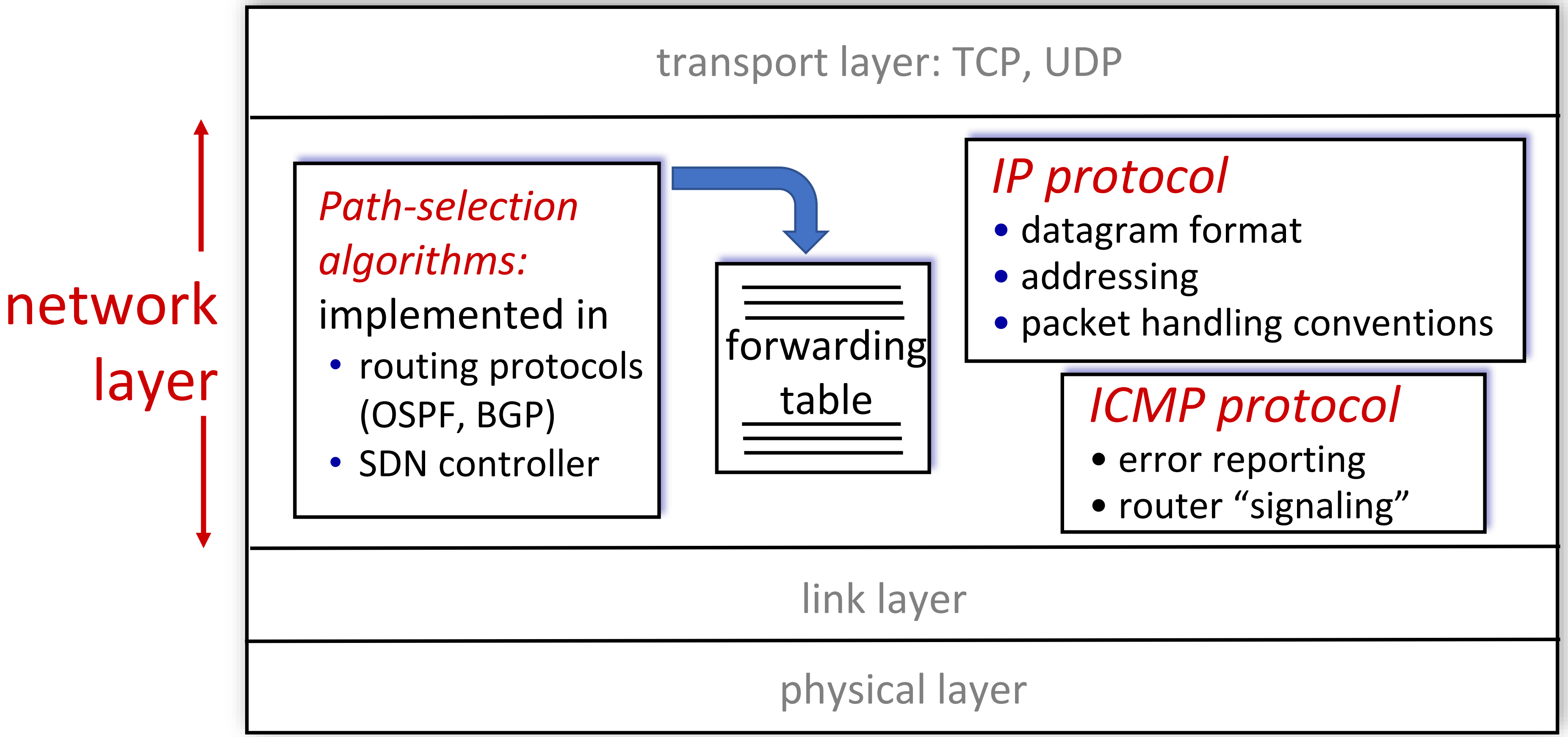
Network layer: “data plane” roadmap

- 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

Network Layer: Internet

host, router network layer functions:



transport layer: TCP, UDP

network layer

Path-selection algorithms: implemented in
- routing protocols (OSPF, BGP)
- SDN controller

forwarding table

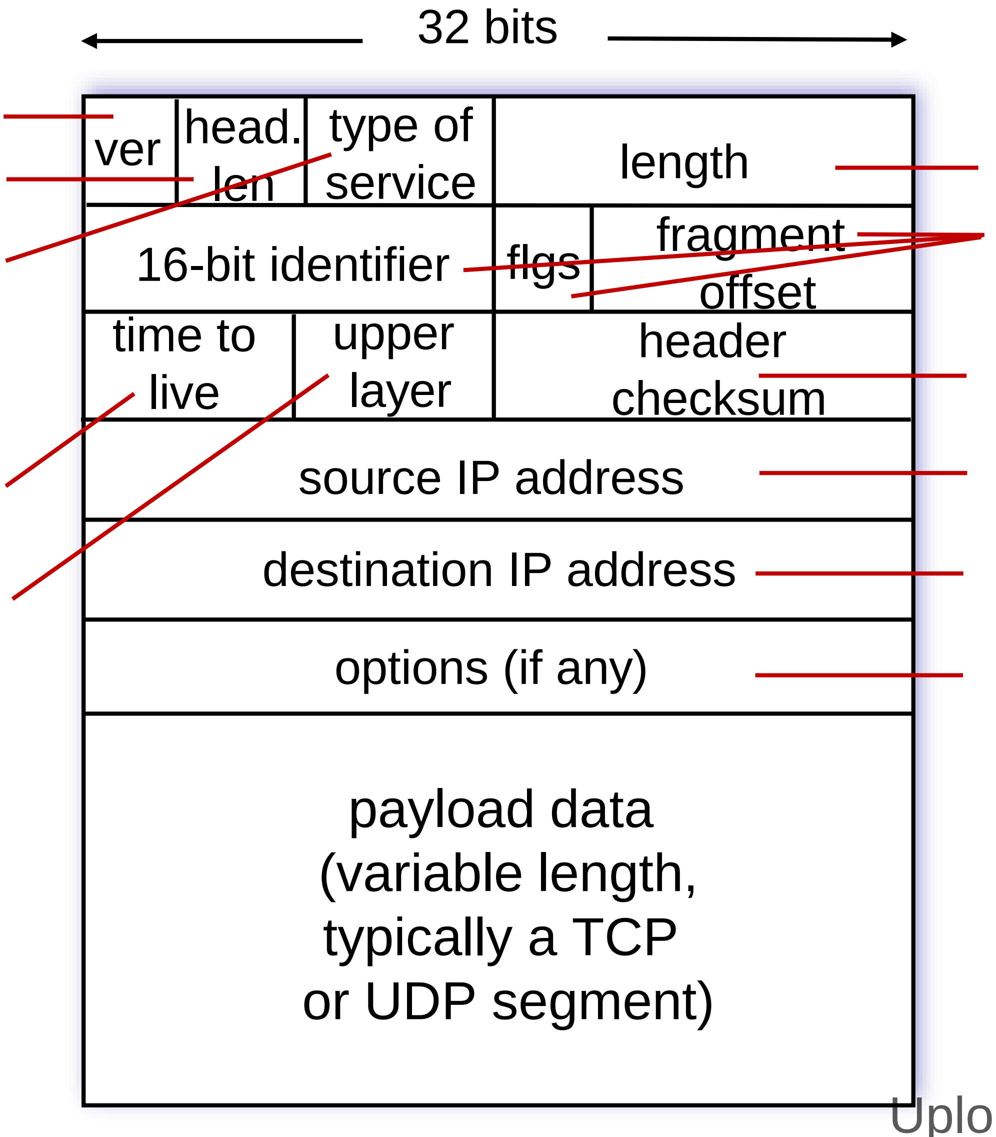
IP protocol
- datagram format
- addressing
- packet handling conventions

ICMP protocol
- error reporting
- router "signaling"

link layer

physical layer

IP Datagram format



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)			

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

3

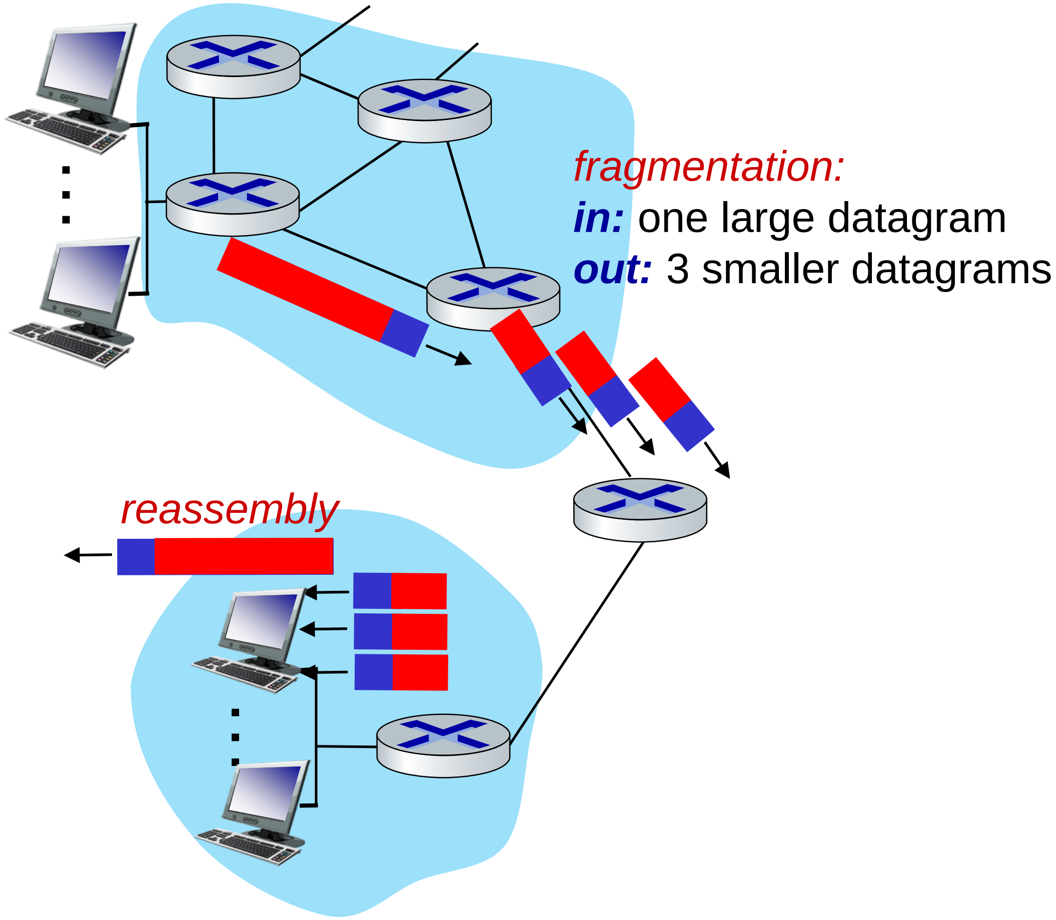
Maximum length: 64K bytes
Typically: 1500 bytes or less

e.g., timestamp, record route taken

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

IP 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



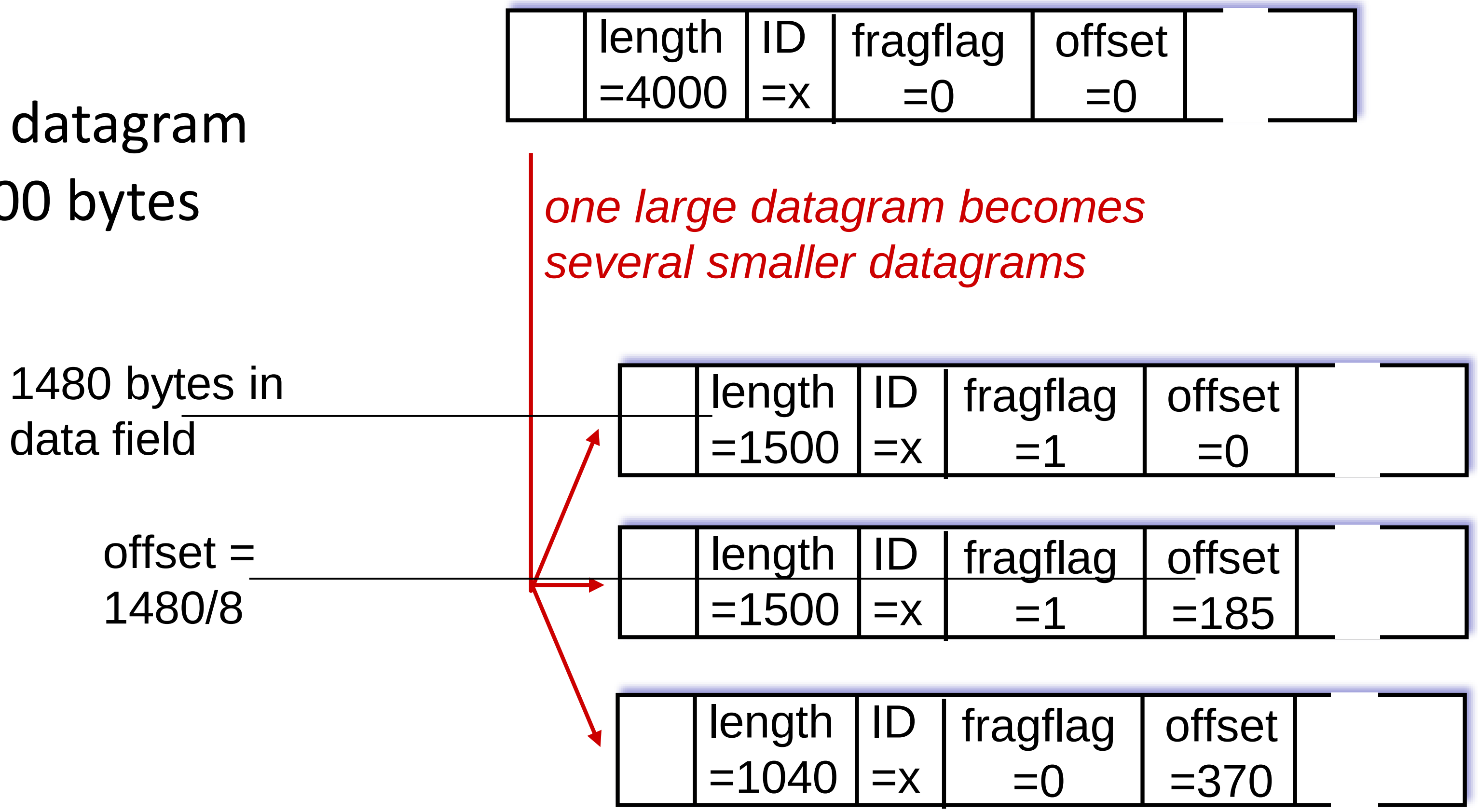
fragmentation:
in: one large datagram
out: 3 smaller datagrams

reassembly

IP fragmentation/reassembly

example:

- 4000 byte datagram
- MTU = 1500 bytes



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

one large datagram becomes several smaller datagrams

1480 bytes in data field

offset = 1480/8

length	ID	fragflag	offset
=1500	=x	=1	=0
=1500	=x	=1	=185
=1040	=x	=0	=370

IP addressing

Q: How does an ISP get block of addresses?

A: ICANN: Internet Corporation for Assigned Names and Numbers

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

IPv4

- IPv4 uses four octets in a group to create an IP address and each octet is made up of eight bits or 1 byte.
- Therefore every IP address is 32 binary bits (4 x 8 = 32) or 4 bytes.
- Designed so that there would be enough IP addresses for the foreseeable future.
- No one predicted the huge growth in IT
- An example of how an IPv4 address appears in binary:

11000011.	**11110000.**	**11001011.**	**11111100**
1st octet	2nd octet	3rd octet	4th octet

- Each grouping of eight numbers is an octet and the four octets gives us a 32 bit IP address.

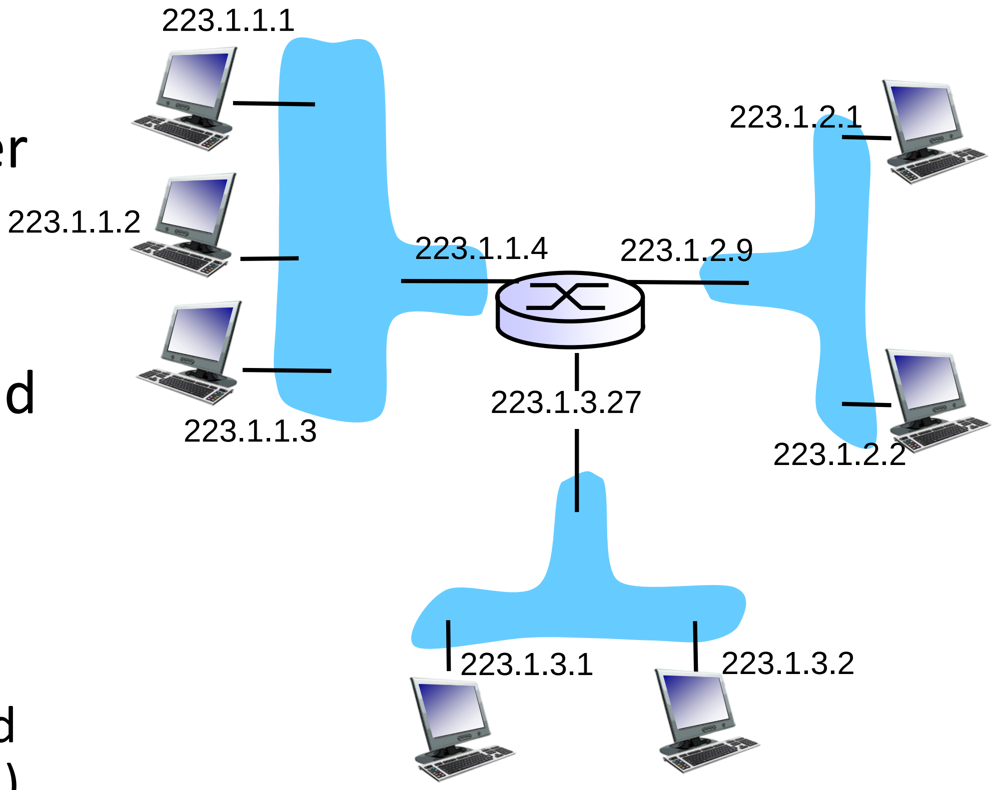
IPv4 Cont…

- The reason for having a 32-bit address is because it was determined that this amount would be more than enough for many years to come.
- Unfortunately, the huge growth of home and business computing was never anticipated.
- IPv6 has several trillion available addresses that should last a few years into the future.
- IP (version 4) addresses are broken into classes.
- Depending upon how large your organization was, dictated which class of IP address you were given.

IP addresses are assigned by a group called the IANA (Internet Assigned Number Authority). You can also buy one from an ISP who has in turn bought a block form the IANA

IP addressing: introduction

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

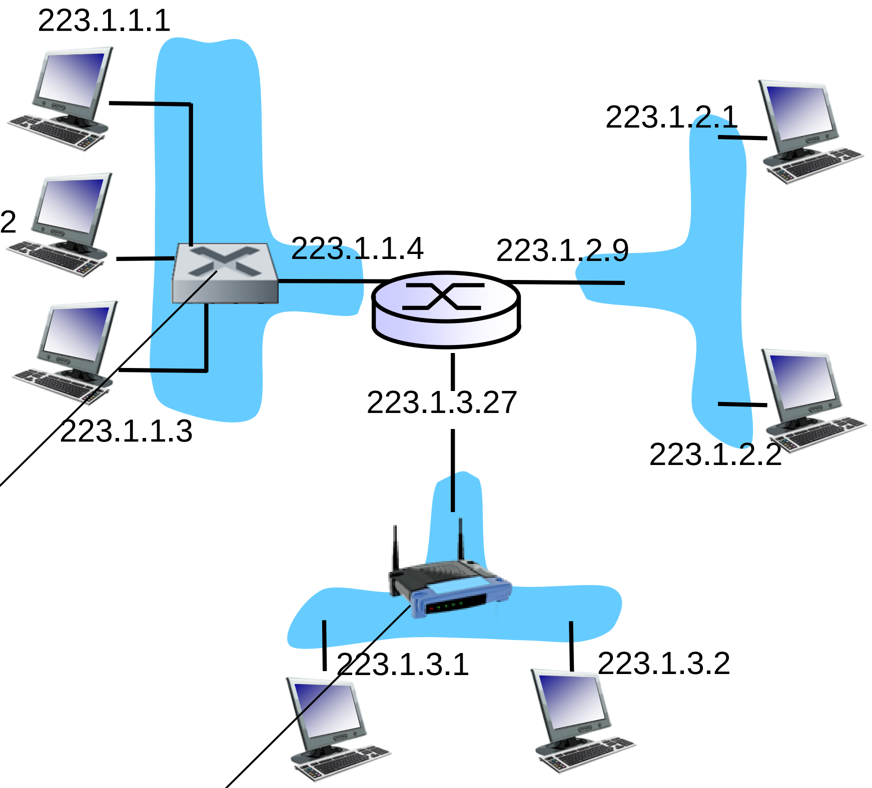


$$223.1.1.1 = \underbrace{11011111}_{223}\ \underbrace{00000001}_{1}\ \underbrace{00000001}_{1}\ \underbrace{00000001}_{1}$$

IP addressing: introduction

Q: how are interfaces actually connected?

A: we'll learn about that in chapter 5.



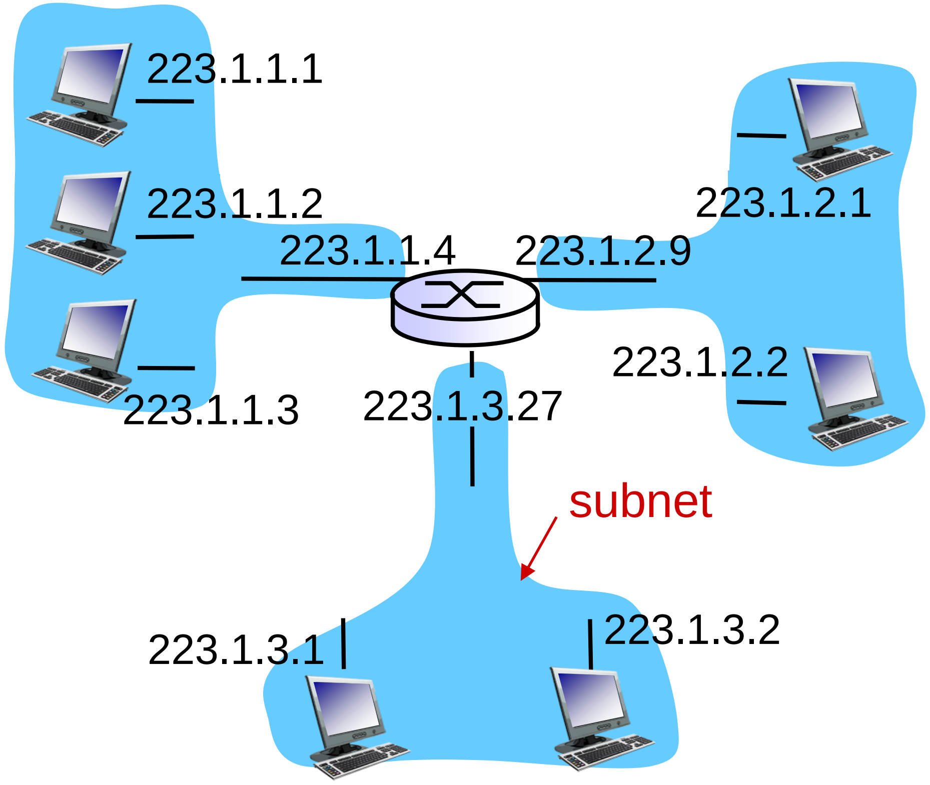
A: wired Ethernet interfaces connected by Ethernet switches

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

Subnets

- IP address:
  - subnet part - high order bits
  - host part - low order bits
- *what's a subnet ?*
  - device interfaces with same subnet part of IP address
  - can physically reach each other *without intervening router*



network consisting of 3 subnets

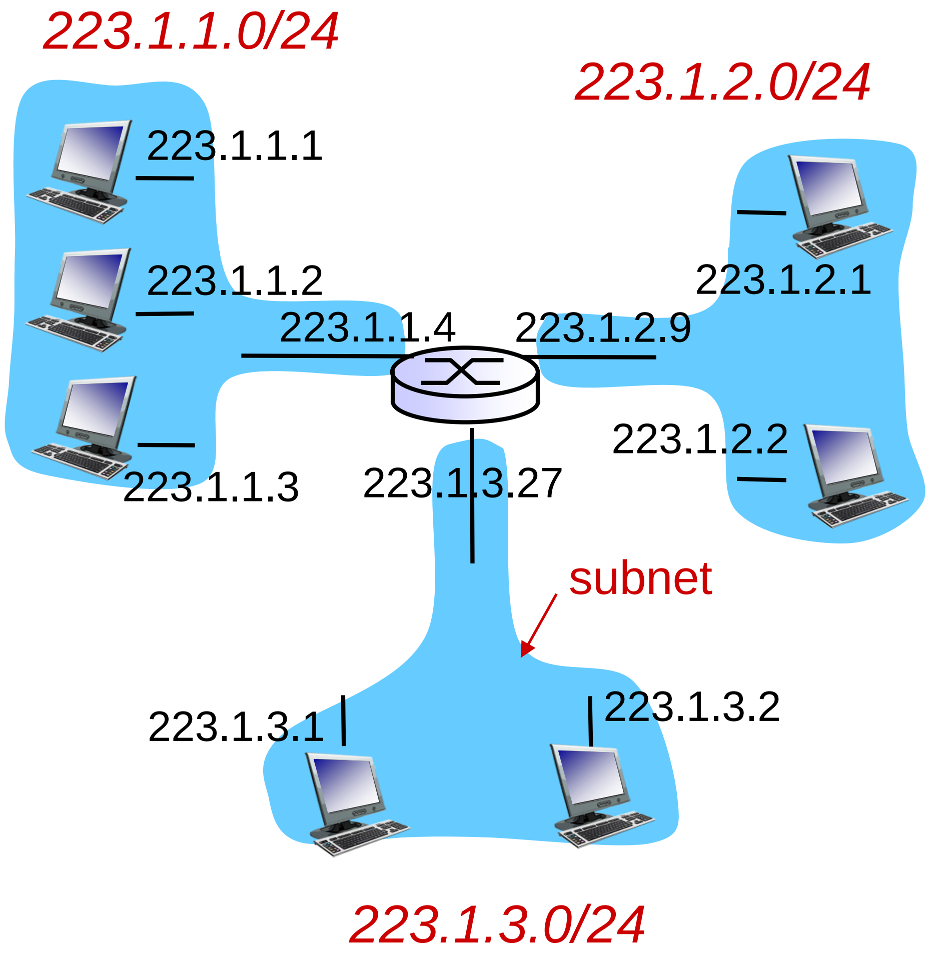
- IP address: 192.168.1.7
- Subnet mask 255.255.255.0
- Network IP 192.168.1.0
- We can write it as 192.168.1.0/24
- Usable IP addresses 2^8-2=254
- 192.168.1.255 is the broadcast address

- IP address: 172.180.1.7
- Subnet mask 255.255.0.0
- Network IP 172.180.0.0
- We can write it as 172.180.0.0/16
- Usable IP addresses 2^16-2=65534
- 172.180.255.255 is the broadcast address

Subnets

recipe

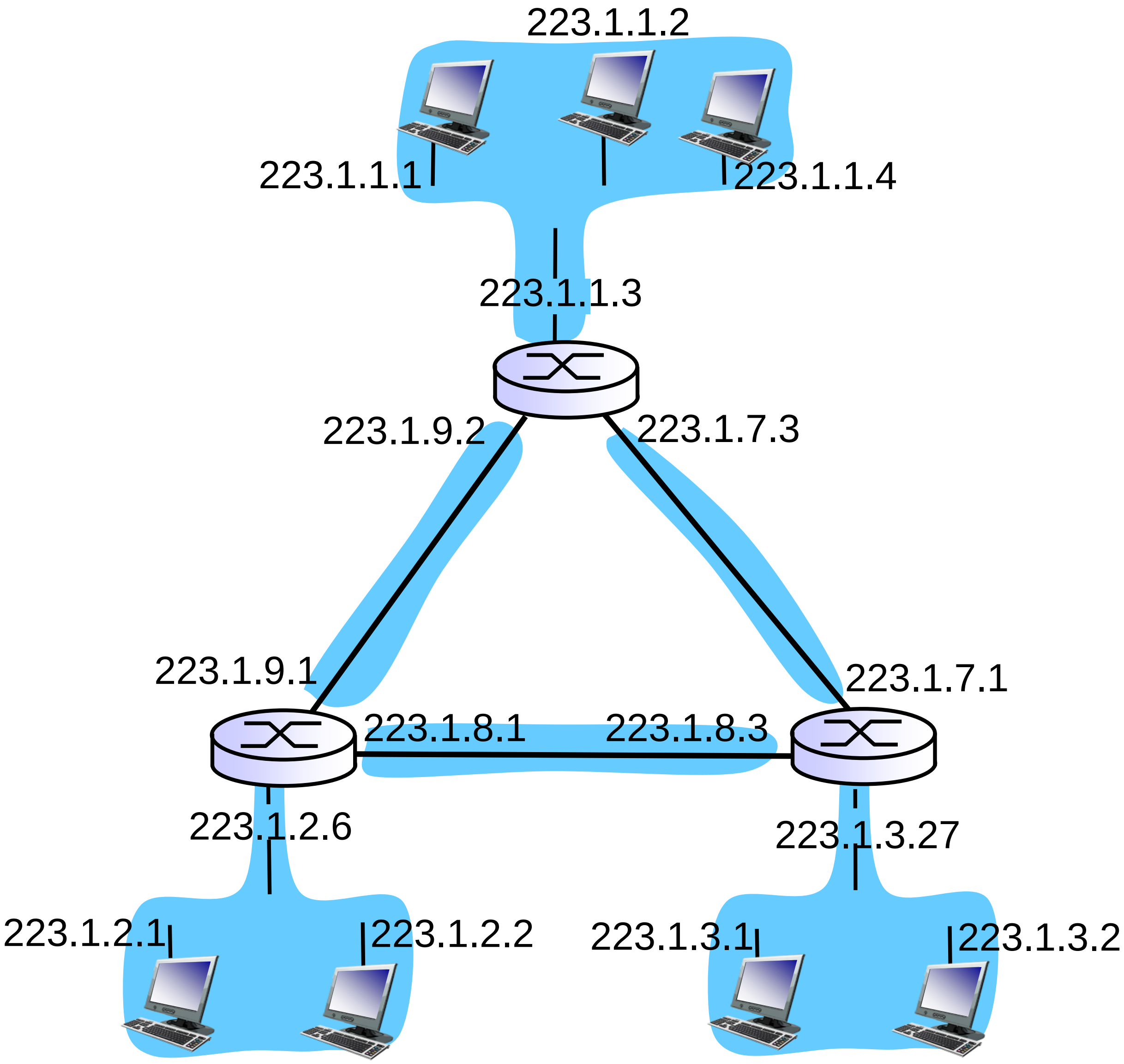
- to determine the subnets, detach each interface from its host or router, creating islands of isolated networks
- each isolated network is called a *subnet*



subnet mask: /24

Subnets

how many?



Classful Addressing

- In the olden days…
  - Class A: 0* (0xxxxxxx.*.*.*)
    - Very large /8 blocks (e.g., MIT has 18.0.0.0/8)
  - Class B: 10*
    - Large /16 blocks (e.g,. Princeton has 128.112.0.0/16)
  - Class C: 110*
    - Small /24 blocks (e.g., AT&T Labs has 192.20.225.0/24)
  - Class D: 1110*
    - Multicast groups
  - Class E: 11110*
    - Reserved for future use (sounds a bit scary…)
- And then, address space became scarce…

Class A Addresses

- These were given to the very largest organizations
  - tremendous number of IP addresses since they owned more computers than everyone else.
- Only use the first octet to identify the network number.
- The remaining three octets are left for identifying the hosts on the network.

Network.Host.Host.Host

10.2.5.4

- So the **network is 10** and **2.5.4 is a host** on that network.
- In binary it would look like:
  - nnnnnnnn.hhhhhhhh.hhhhhhhh.hhhhhhhh

Cont …

You would pronounce the above IP address as ten dot two dot five dot four.

- Class A addresses are numbered from 1 to 126 in the first octet.
- Network equipment identifies a class A address because the very first bit on the first octet has to be a 0.
  - It cannot have a 1 in this bit position.
- So the first network number is 1.

128	64	32	16	8	4	2	1
0	0	0	0	0	0	0	1

Cont …

- The last possible network number is 127.

128	64	32	16	8	4	2	1
0	1	1	1	1	1	1	1

- Use the powers of two rule:
  - The first octet can have a possible 256 (2^8 = 256) networks.
  - However, not allowed to use the first bit of the first octet, it is reserved for showing the 0 (binary) value.
  - So this leaves us with 7 digits. 2^7–(1)gives us 127 networks.
  - The full three octets to use for hosts so 8+8+8 bits gives us 2^24-(2) = 16,777,214 hosts per class A network.

Cont …

- Network number 127 cannot actually be used because the value 127.0.0.1 is reserved for troubleshooting.
  - 127.0.0.1 is known as a loopback address
  - You can ping the loopback address to check if TCP/IP is working on your host.
- We are not permitted to use 0 as a network number or the 127 which leaves us 126 available networks for class A addresses.
- For the hosts we can start at number one until every single possible value is used up.

Cont …

- Example:
  - 10.0.0.1 is the first host, or in binary

00001010.	00000000.	00000000.	00000001
10.	0.	0.	1

  - 10.0.0.2 is the second host, or in binary:

00001010.	00000000.	00000000.	00000010
10.	0.	0.	2

Cont …

- 10.255.255.254 is the last host, or in binary:

00001010.	11111111.	11111111.	11111110
10.	255.	255.	254

- Decimal notations are used so that it can be easy to write out the IP addresses and easy to remember.
- Why can’t we have 10.255.255.255 as a host?
  - Because when all the binary values have a 1 on the host part of the address this tells the network that it is a broadcast packet.

Class B Addresses

- They were reserved for large organizations that needed a lot of host numbers but not as many as the largest ones.
- When a class B address was assigned to an organization it resulted in thousands of wasted host numbers.
- They have to have the first two binary values on the first octet reserved with a 1 and a 0 next to it.
- So the first network number is 128
  - all the available network bits on the first octet turned off.

128	64	32	16	8	4	2	1
1	0	0	0	0	0	0	0

Cont …

- The last available class B network number is 191

128	64	32	16	8	4	2	1
1	0	1	1	1	1	1	1

  - network bits have been turned on (on the first octet).

- The first two octets for the network address the other two identify the hosts on the network.

- For example, the address 130.24.5.2
  - 130.24 is the network number
  - 5.2 is a host on that network

- The range of class B IP addresses is between 128 and 191.

Cont …

- Use the powers of two rule:
  - The first two octets can have a possible 65536 (2^16 = 65536) networks.
  - however, not allowed to use the first two bits of the first octet, they are reserved for showing the 10 (binary) value.
  - So this leaves us with 6+8 digits. 2^14 gives us 16384 networks.
  - The full two octets to use for hosts so 8+8 bits gives us 2^16 –(2) = 65534 hosts per class B network.

Class C Addresses

- Reserved for any other organization that was not large enough to warrant having a class A or B address.
- It has the first three bits reserved so the network device can recognize it as such.
  - The first three bits must show as 110.
- The first network number is 192. All the other network bits are off (0).

128	64	32	16	8	4	2	1
1	**1**	**0**	0	0	0	0	0

Cont …

- And the last is 223. This time all the network bits are on (on the first octet).

128	64	32	16	8	4	2	1
1	**1**	**0**	1	1	1	1	1

- An example of a class C address is 200.2.1.4
  - 200.2.1 is the network address
  - .4 is a host on that network
- There are lots of available network numbers to assign to companies
- Limited amount of numbers free to use for the hosts on our networks.

Cont …

- For networks we have to take the first three bits (011) from the first octet giving us 5+8+8= 21 (network bits).
  - 2^21 = 2097152
- For the hosts we have 2^8 giving us 256 (only 254 are usable though).

Class D and E Addresses

- Class D addresses are reserved for multicast traffic and cannot be used on your network.
  - Multicast traffic is traffic sent to multiple hosts using one IP
    - A live web cast of a rock concert would be an example of multicasting.
- Class E addresses are reserved for experimental use only.

Addresses Reserved for Private Use

- InterNIC has set aside certain addresses and have been reserved for private use only.
  - For example, 127.0.0.1 is reserved for testing purposes only
- Other include a list of addresses that are used only on *private networks*, not the Internet

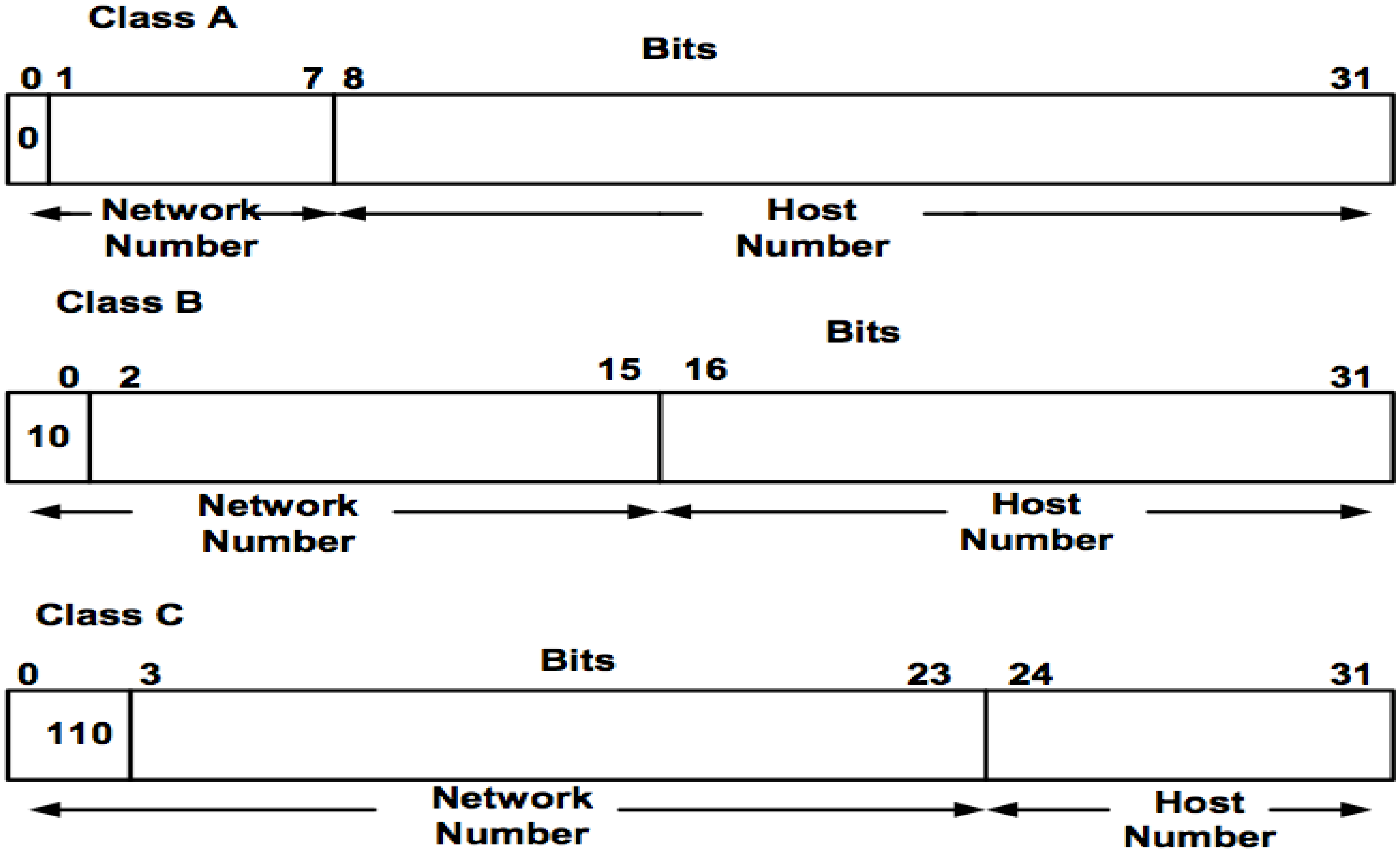
Cont …

- If you would like to use TCP/IP on your internal network (intranet) and not use the Internet, the following addresses are suggested:
  - **Class A** 10.0.0.0 through 10.255.255.255
  - **Class B** 172.16.0.0 through 172.31.255.255
  - **Class C** 192.168.0.0 through 192.168.255.255
- Routers on the Internet will not route data from or to these addresses; they are for internal, private use only.
- To use these addresses on an intranet and have access to the Internet, you must use a *proxy server* or *Network Address Translation (NAT).*

Summary

- **Class A** – first bit set to 0. Address range 1-126 (127 is reserved for testing) Network.Host.Host.Host
- **Class B** – first bits set to 10. Address range 128-191 Network.Network.Host.Host
- **Class C** – first bits set to 110. Address range 192-223 Network.Network.Network.Host
- **Class D** – first bits set to 1110. Address range from 224-239
- **Class E** – first bits set to 11110. Address range from 240-255
- To recognize the address class of an IP, look at the first octet.
  - 10.1.2.1 = **Class A**, 190.2.3.4 = **Class B**, 220.3.4.2 = **Class C**

IP Address Classes



- 12.0.0.0/8 Class A
- 255.0.0.0
- # of hosts=2^(24)-2=16M

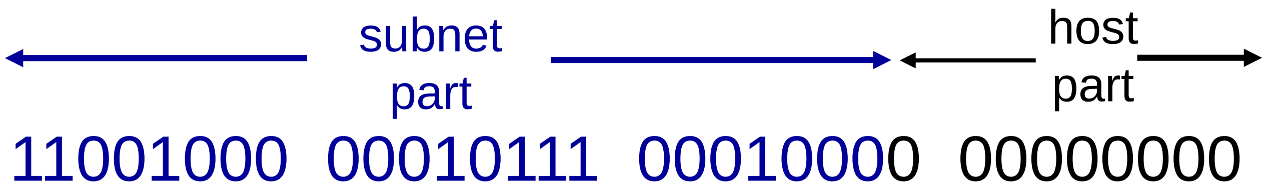
- 130.16.0.0/16 Class B
- 255.255.0.0
- # of hosts=2^(16)-2=65K

- 200.20.20.0/24 Class C
- # of hosts=2^8-2 =254

IP addressing: CIDR

CIDR: Classless InterDomain Routing

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



subnet part

host part

11001000 00010111 00010000 00000000

200.23.16.0/23
Subnet mask 255.255.254.0
#of hosts=2^9-2=510
Network Address= 200.23.16.0
First IP 200.23.16.1
Last IP 200.23.17.254
Broadcast address=200.23.17.255

200.16.16.0/24

- # of hosts 2^8-2=254
- We need 4 subnets
- 200.16.16.**00**000000
- 00
- 01
- 10
- 11

First Subnet

- 200.16.16.00000000
- 200.16.16.0/26
- Network Address 200.16.16.0
- First IP 200.16.16.1
- Last IP 200.16.16.62
- Broadcast Address 200.16.16.63
- Subnet mask=
- 255.255.255.11000000
- 255.255.255.192

Second Subnet

- 200.16.16.**01000000**
- 200.16.16.64/26
- Network address=200.16.16.64
- First IP address=200.16.16.65
- Last IP address=200.16.16.126
- Broadcast Address=200.16.16.127
- Subnet mask
- 255.255.255.192

Third Subnet

- 200.16.16.10000000
- 200.16.16.128/26
- Network address =200.16.16.128
- First IP 200.16.16.129
- Last IP 200.16.16.190
- Broadcast 200.16.16.191
- Subnet mask 255.255.255.192

Fourth IP

- 200.16.16.11000000
- 200.16.16.192/26
- Network address 200.16.16.192
- First IP 200.16.16.193
- Last IP 200.16.16.254
- Broadcast 200.16.16.255
- Subnet mask 255.255.255.192

- We need three subnets
- Net1 100 host → 7 bits in the host part
- Net2 55 host
- Net3 50

200.16.16.0/24

- Divide the network into 2 subnets
- 200.16.16.**00**000000
- 0
- 1
- Subnet 1 200.16.16.0/25 ➔ net1

- Subnet 2 200.16.16.128/25

Subnet 2 200.16.16.128/25
divide this network into 2 subnets

- 200.16.16.10000000
- 0
- 1
- 200.16.16.128/26 net 2
- 200.16.16.192/26 net 3

Subnet 1 200.16.16.0/25 → net1

- 200.16.16.00000000
- Network address 200.16.16.0
- First IP 200.16.16.1
- Last IP 200.16.16.126
- Broadcast 200.16.16.127
- Subnet mask
- 255.255.255.128

200.16.16.128/26 net 2

- 200.16.16.10000000

- Network address 200.16.16.128
- First IP address 200.16.16.129
- Last IP address 200.16.16.190
- Broadcast address 200.16.16.191
- Subnet mask 11111111.11111111.11111111.11000000
- Subnet mask 255.255.255.192

200.16.16.192/26 net 3

- Network address 200.16.16.192
- First IP address 200.16.16.193
- Last IP address 200.16.16.254
- Broadcast address 200.16.16.255
- Subnet mask 255.255.255.192

Subnetting Example

- An ISP has 134.16.0.0/16 and we need to have 4 subnets, find for each subnet:
- The IP address range
- Subnet mask
- Broadcast address
- Network address

- 134.16.0.0/16
- We need 4 subnets
- The first subnet
- 134.16.00000000.00000000 → 134.16.0.0/18
- Second subnet
- 134.16.01000000.00000000 → 134.16.64.0 /18
- Third subnet
- 134.16.10000000.00000000 134.16.128.0/18
- Fourth subnet
- 134.16.11000000.00000000 134.16.192.0/18

- The first subnet
- 134.16.00000000.00000000 →134.16.0.0/18
- First IP 134.16.00000000.00000001 → 134.16.0.1
- Last IP address 134.16.00111111.11111110→ 134.16.63.254
- Broadcast address → 134.16.63.255

- Second subnet
- 134.16.0100000.00000000 ➔ 134.16.64.0 /18
- Network address 134.16.64.0
- First IP address: 134.16.64.1
- Last IP address 134.16.127.254
- Broadcast IP address 134.16.127.255

subnet	Network Address	Broadcast Address	First IP	Last IP
1	134.16.0.0 /18	134.16.63.255	134.16.0.1	134.16.63.254
2	134.16.64.0/18	134.16.127.255	134.16.64.0	134.16.127.254
3	134.16.128.0/18	134.16.191.255	134.16.128.1	134.16.191.254
4	134.16.192.0/18	134.16.255.255	134.16.192.1	134.16.255.254

Subnetting Example

- An ISP has 134.16.0.0/16 and we need to have 6 subnets, find for each subnet:
- The IP address range
- Subnet mask
- Broadcast address
- Network address

- First subnet
- 134.16.00000000.00000000
- 134.16.0.0/19
- The new Subnet mask 255.255.224.0
- First IP address 134.16.0.1
- Last IP Address 134.16.31.254
- Broadcast Address 134.16.31.255

- Second subnet
- 134.16.00100000.00000000
- 134.16.32.0/19
- First IP address
- 134.16.32.1
- Last IP address 134.16.63.254
- Broadcast Address 134.16.63.255

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

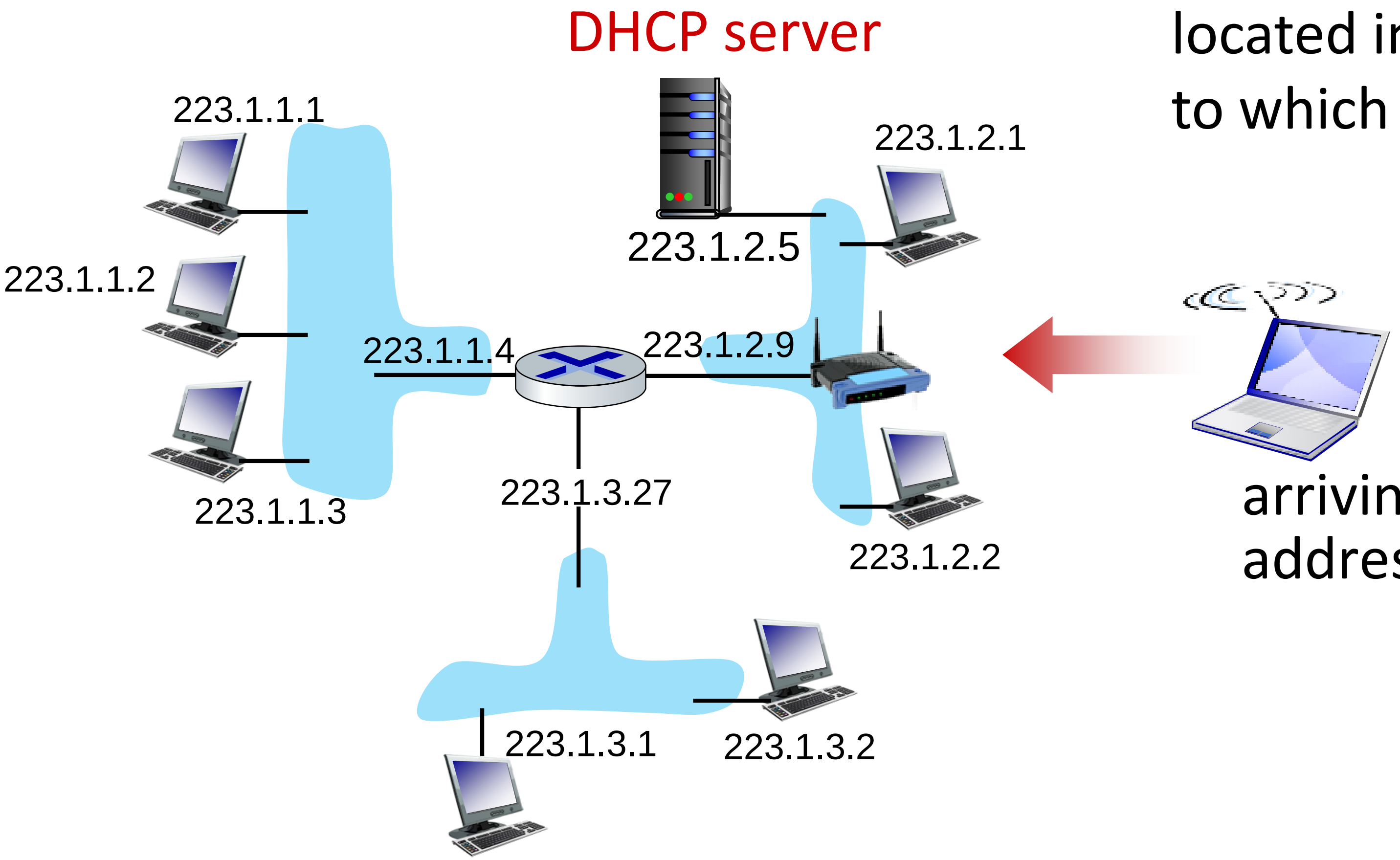
- can renew its lease on address in use
- allows reuse of addresses (only hold address while connected/on)
- support for mobile users who join/leave network

DHCP overview:

- 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

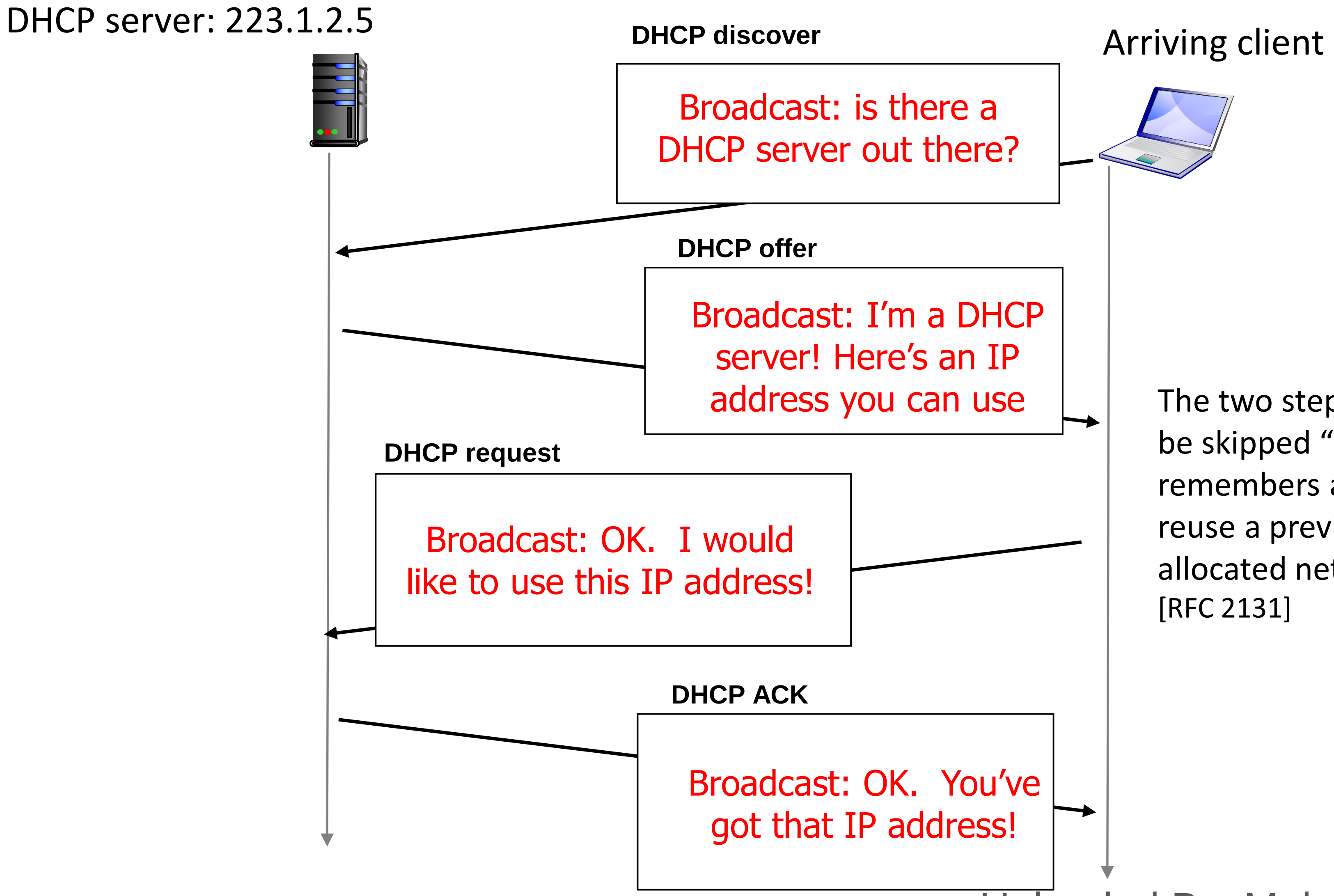
DHCP client-server scenario

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



DHCP server: 223.1.2.5

Arriving client

DHCP discover

Broadcast: is there a DHCP server out there?

DHCP offer

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

DHCP request

Broadcast: OK. I would like to use this IP address!

DHCP ACK

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

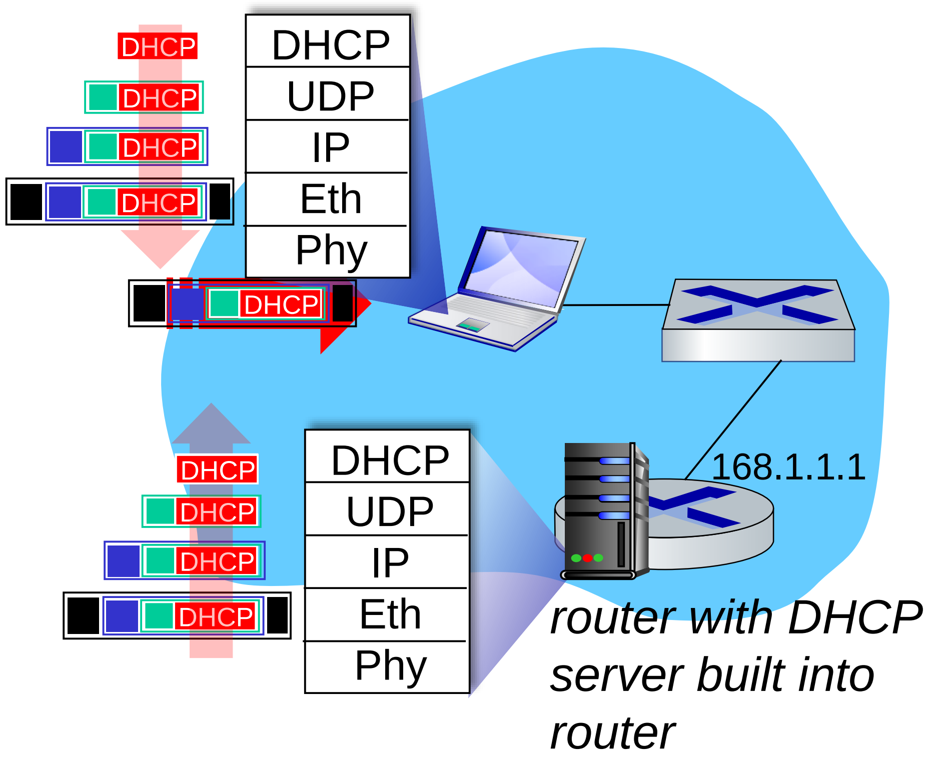
The two steps above can be skipped "if a client remembers and wishes to reuse a previously allocated network address" [RFC 2131]

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)

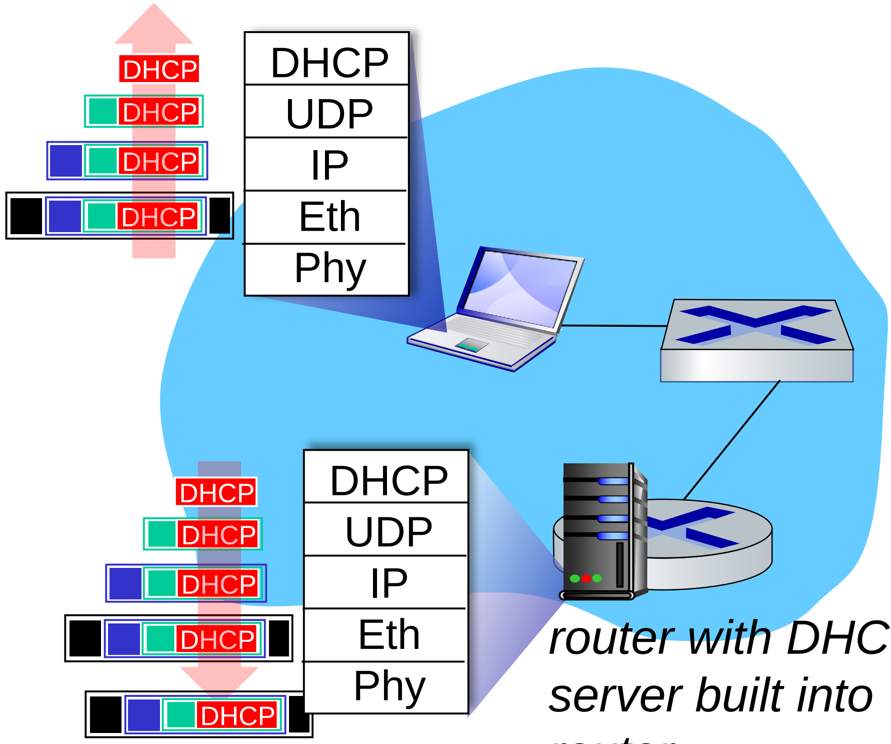
DHCP: example



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 demux'ed to IP demux'ed, UDP demux'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, 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
Subnet mask 255.255.240.0

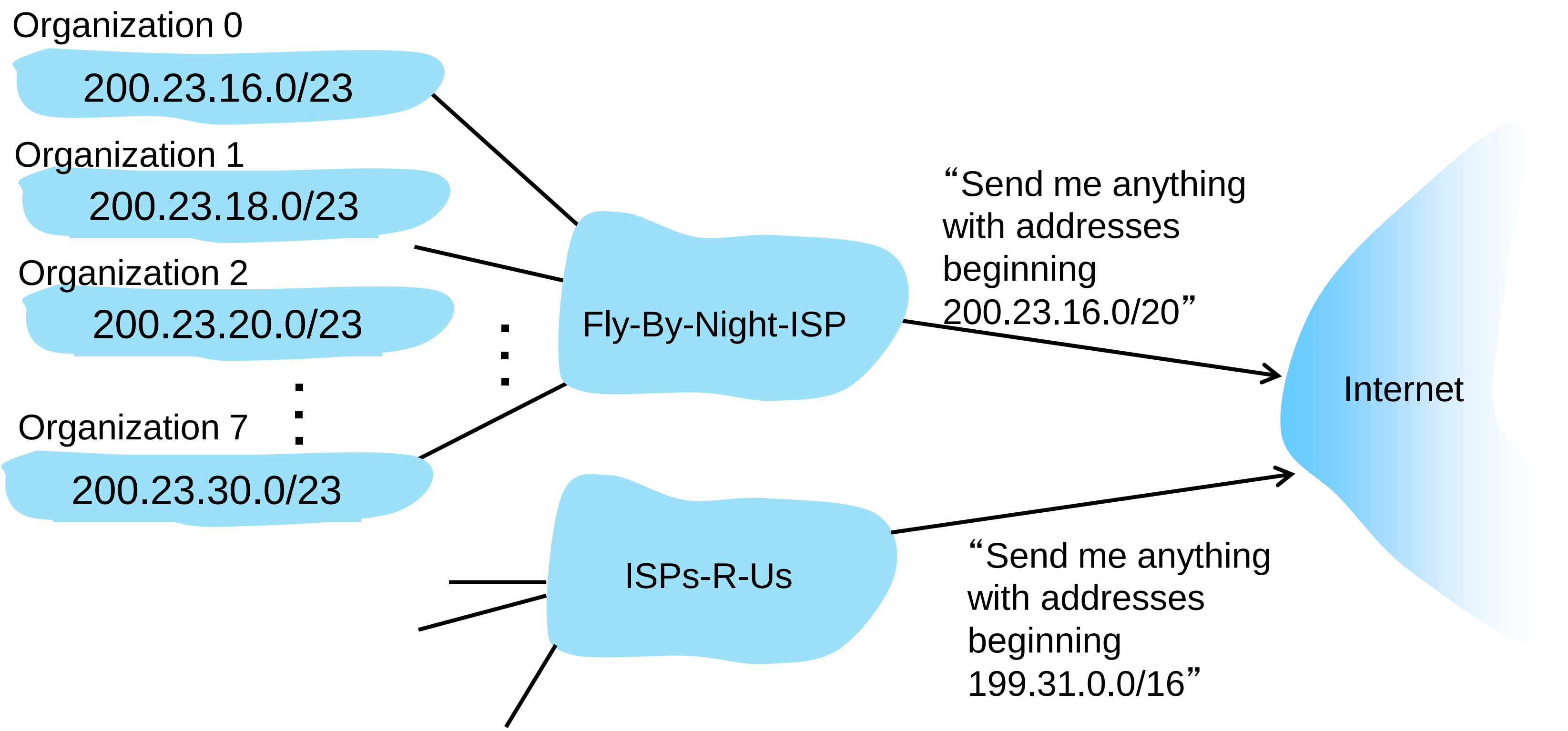
ISP can then allocate out its address space in 8 blocks:

Organization 0 11001000 00010111 00010000 00000000 200.23.16.0/23
First ip 200.23.16.1 last ip address 200.23.17.254
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

Subnet mask 255.255.254.0

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

ISPs-R-Us

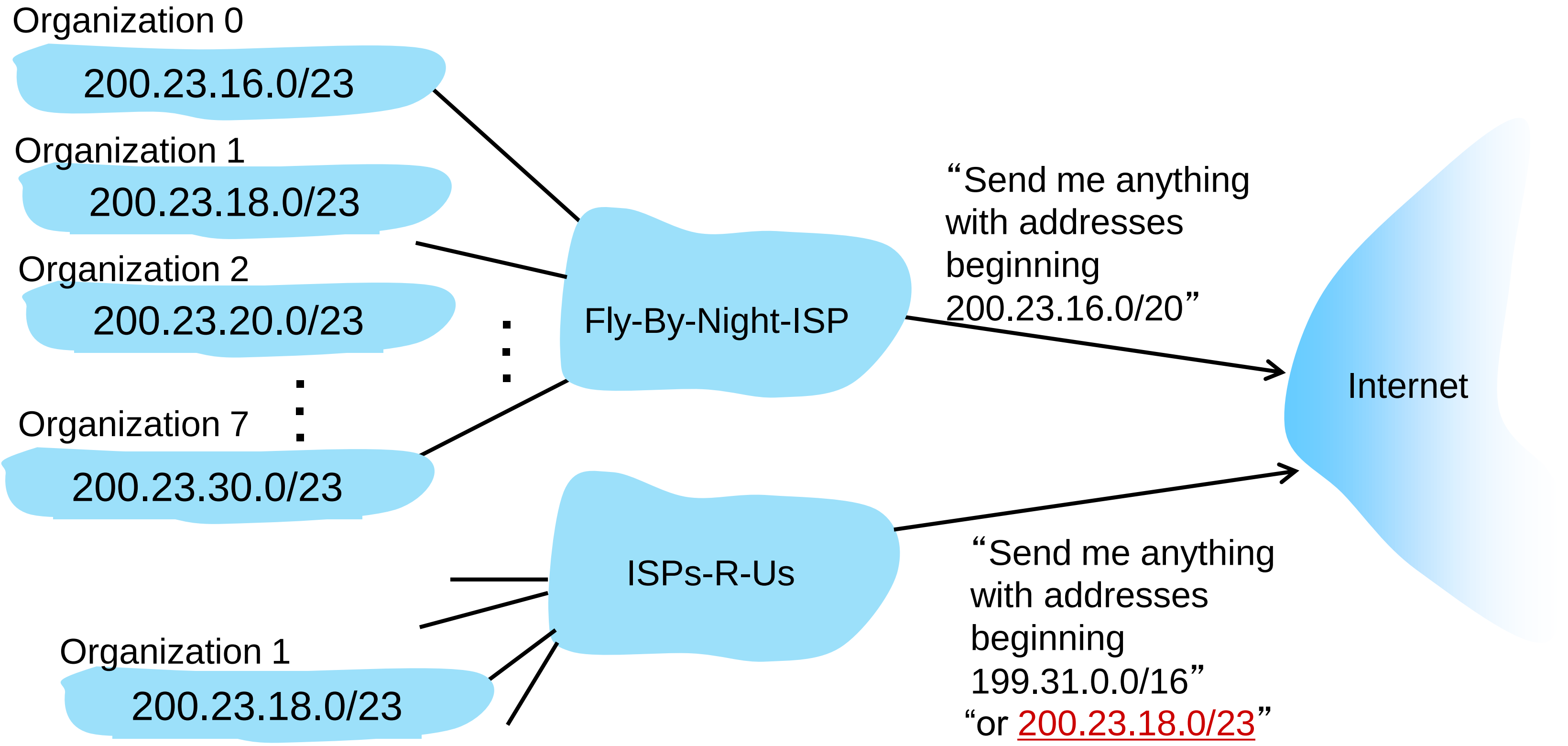
Internet

“Send me anything with addresses beginning 200.23.16.0/20”

“Send me anything with addresses beginning 199.31.0.0/16”

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

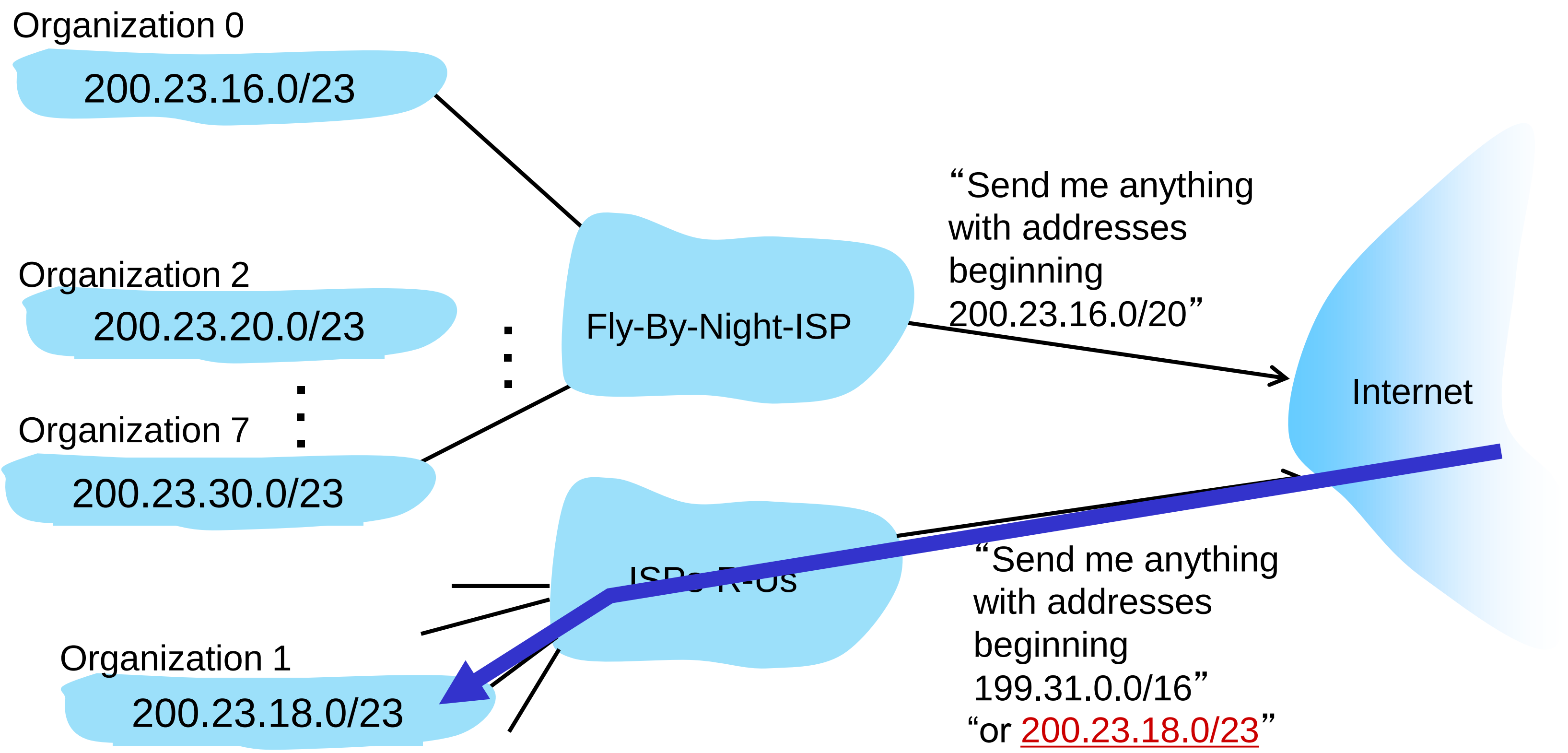
"Send me anything with addresses beginning 199.31.0.0/16" "or 200.23.18.0/23"

Organization 1

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



"Send me anything with addresses beginning 200.23.16.0/20"

"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

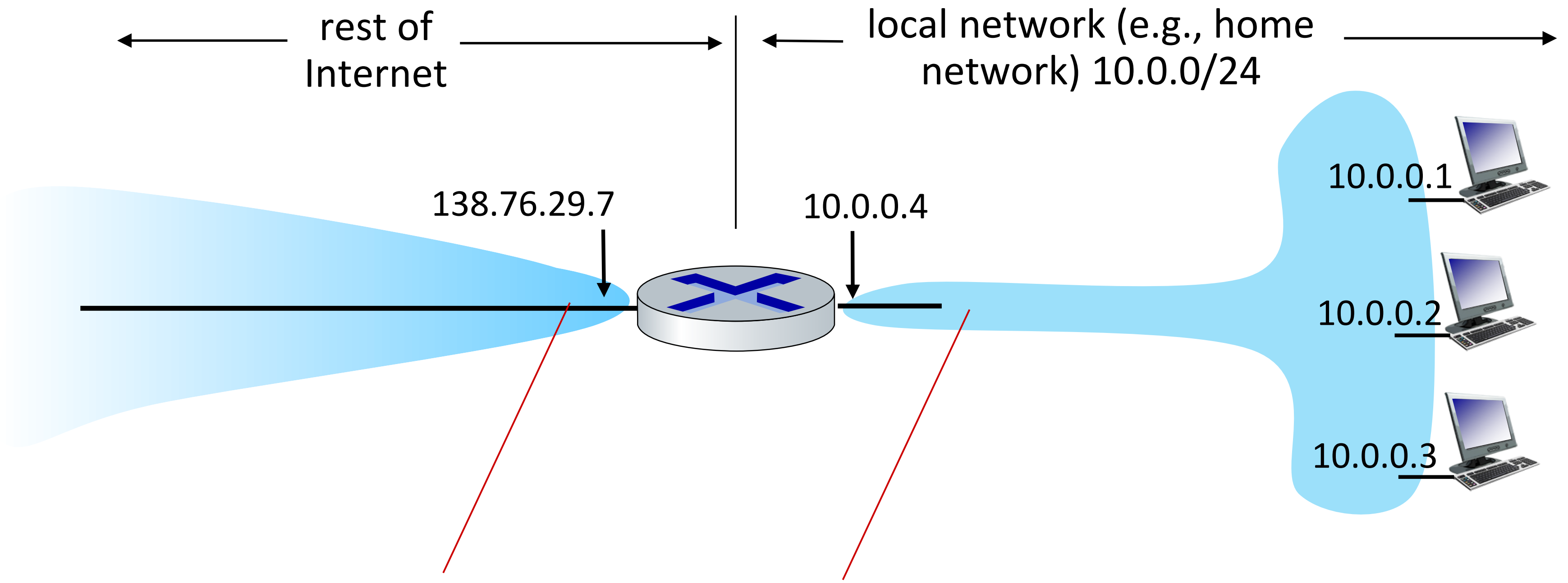
"Who the hell knew how much address space we needed?" Vint Cerf (reflecting on decision to make IPv4 address 32 bits long)

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

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 (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

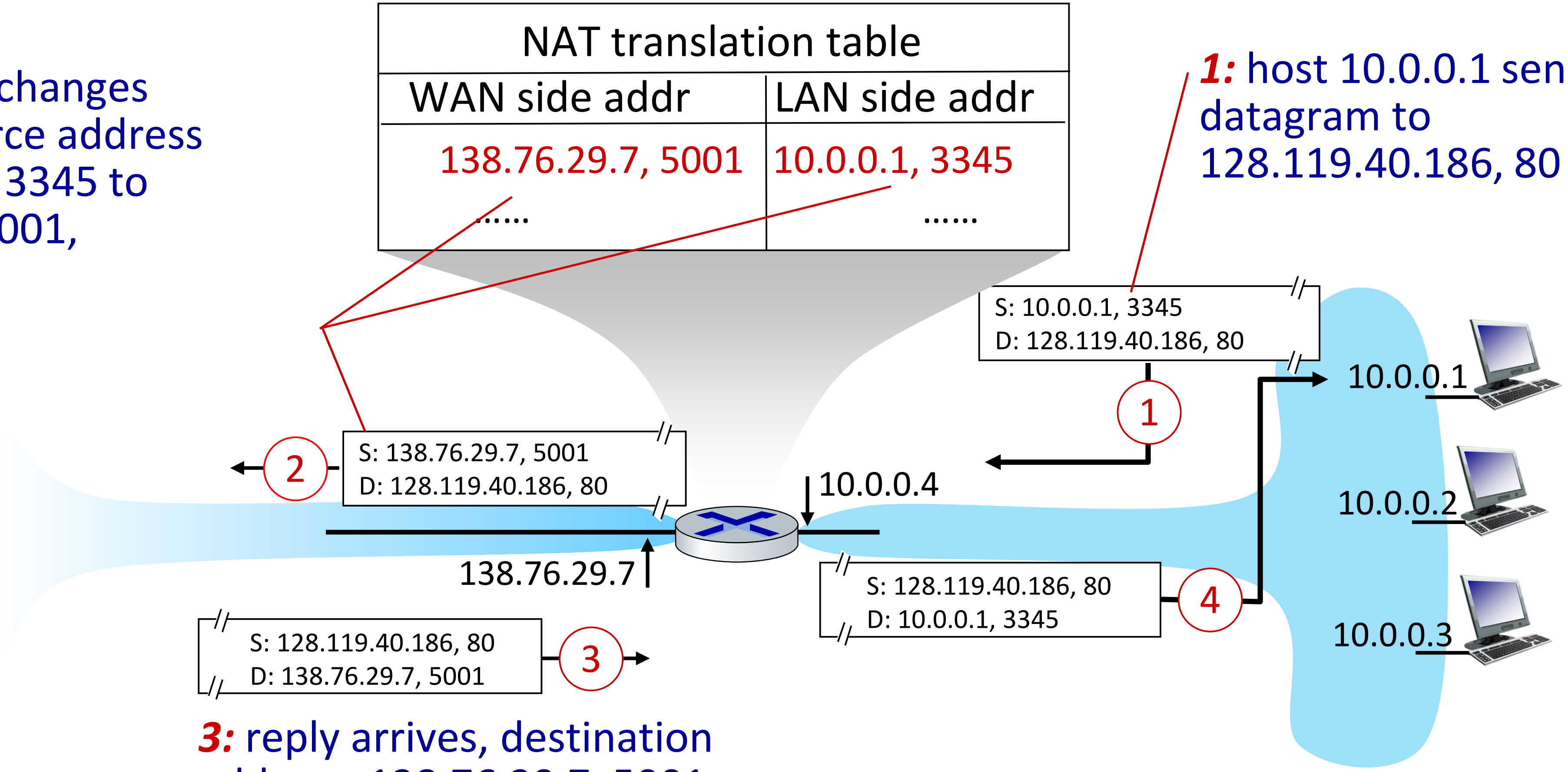
NAT: network address translation

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

NAT translation table

WAN side addr	LAN side addr
138.76.29.7, 5001	10.0.0.1, 3345
……	……

1: host 10.0.0.1 sends datagram to 128.119.40.186, 80

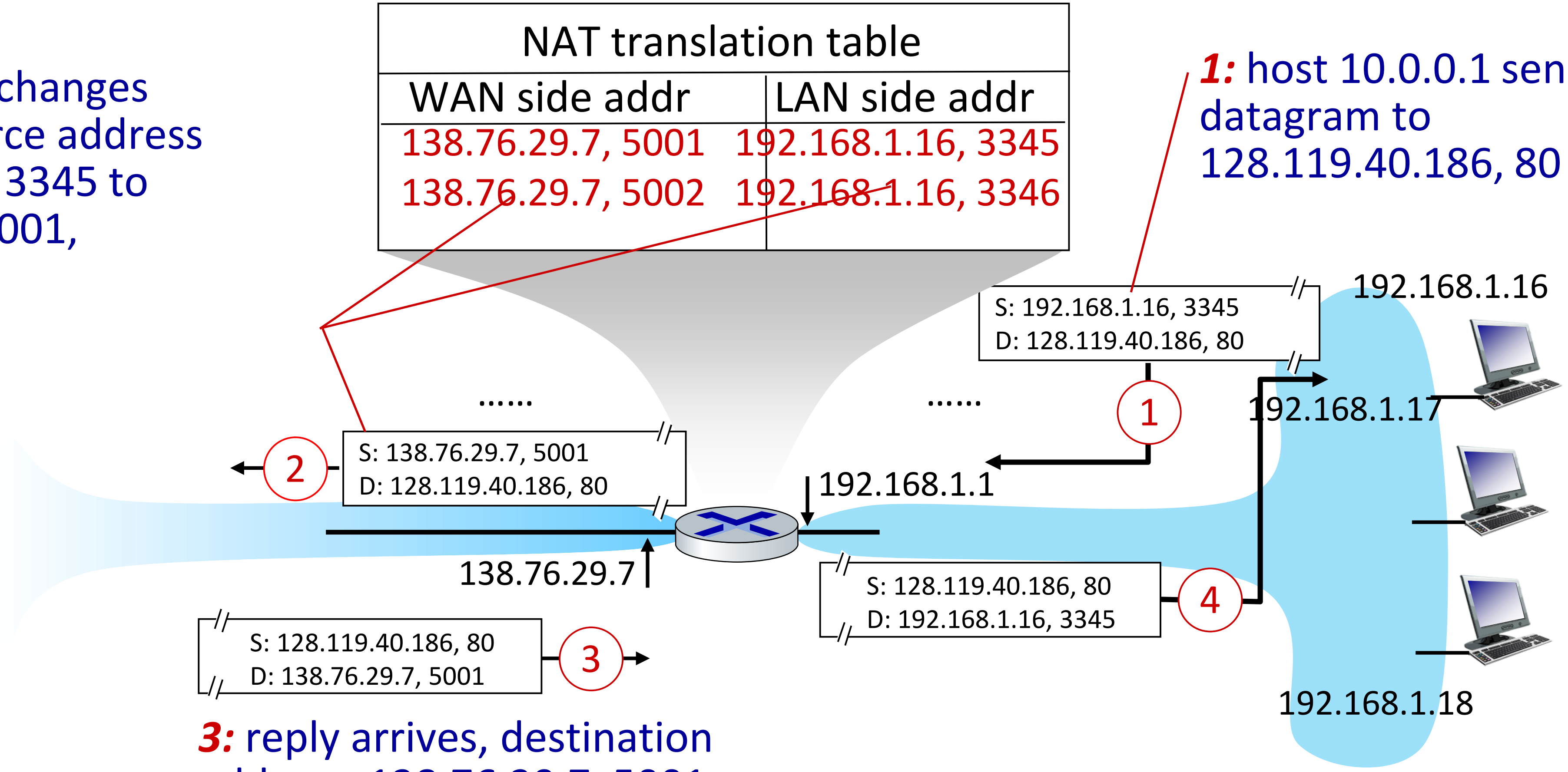


3: reply arrives, destination address: 138.76.29.7, 5001

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



NAT translation table

WAN side addr	LAN side addr
138.76.29.7, 5001	192.168.1.16, 3345
138.76.29.7, 5002	192.168.1.16, 3346

3: reply arrives, destination address: 138.76.29.7, 5001

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

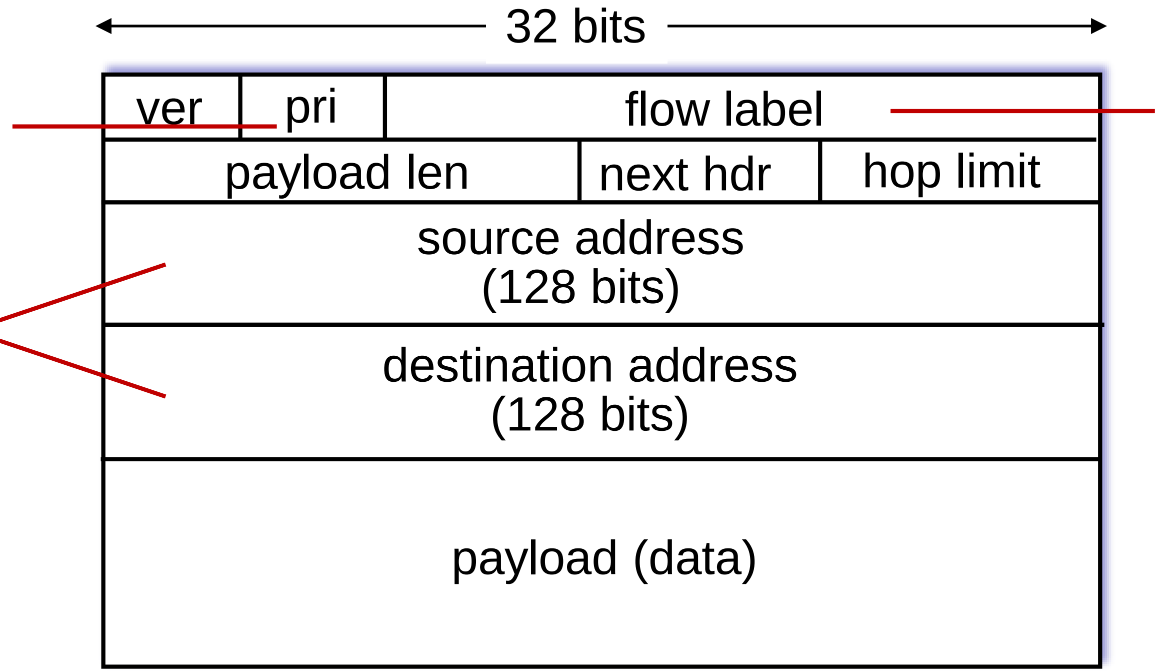
IPv6: motivation

- 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

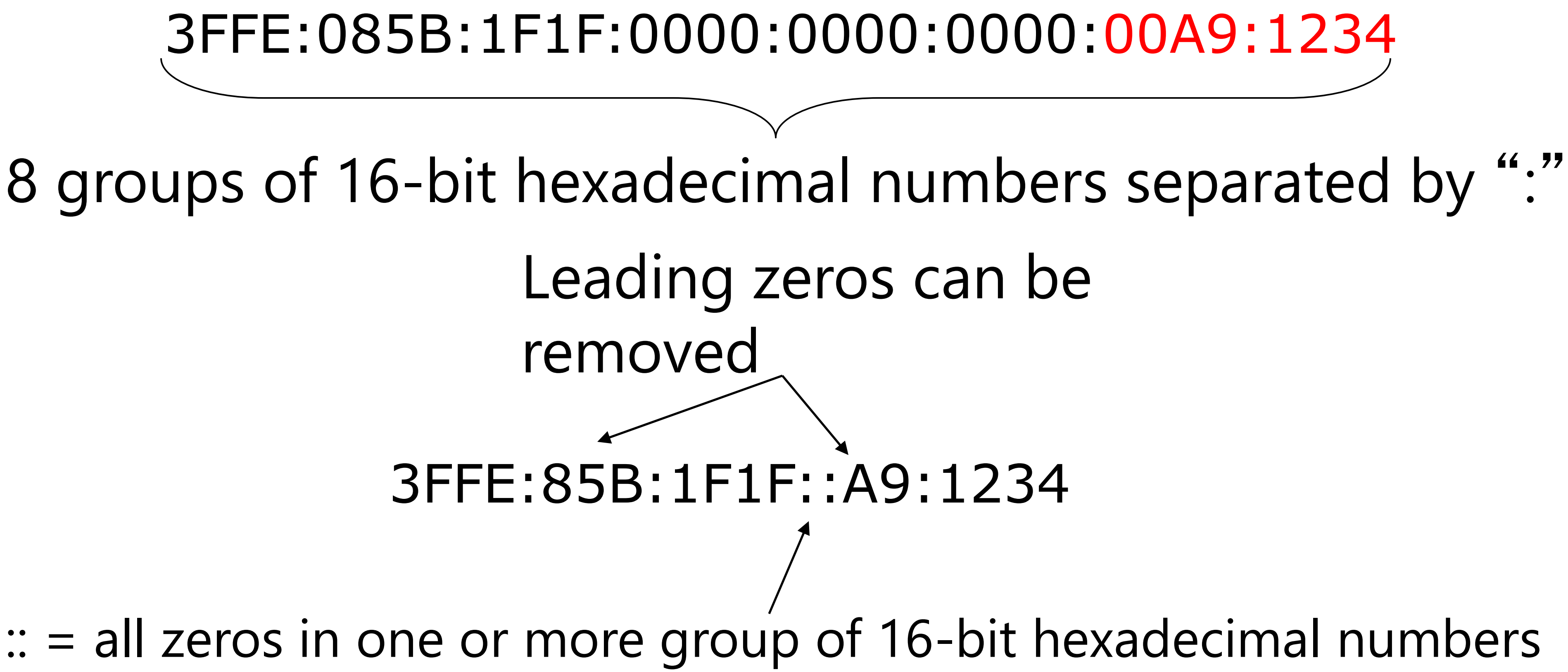


flow label: identify datagrams in same "flow." (concept of "flow" not well defined).

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)

128-bit IPv6 Address



3FFE:085B:1F1F:0000:0000:0000:00A9:1234

8 groups of 16-bit hexadecimal numbers separated by “:”

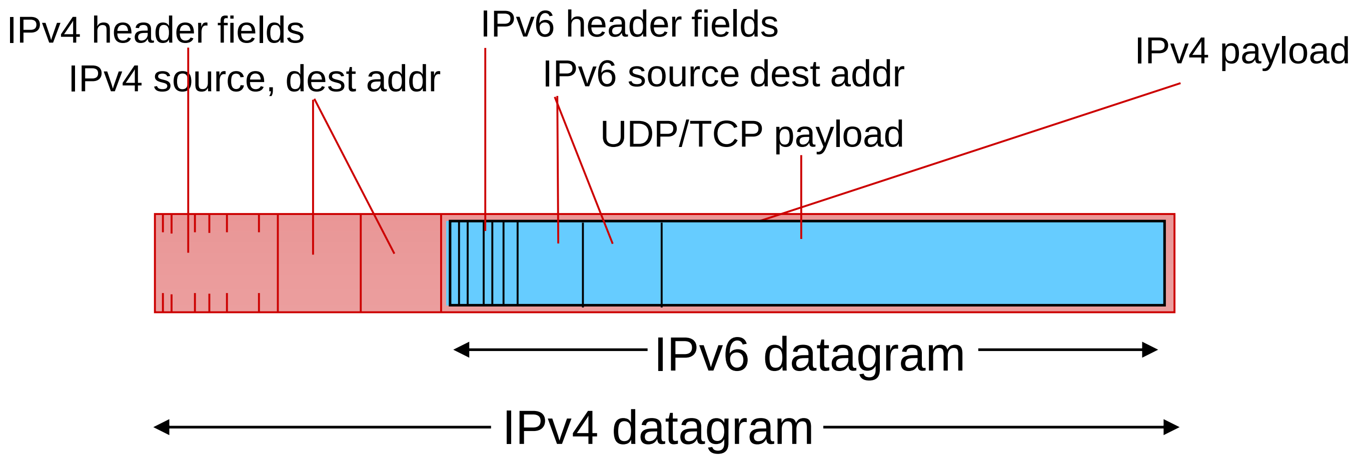
Leading zeros can be removed

3FFE:85B:1F1F::A9:1234

:: = all zeros in one or more group of 16-bit hexadecimal numbers

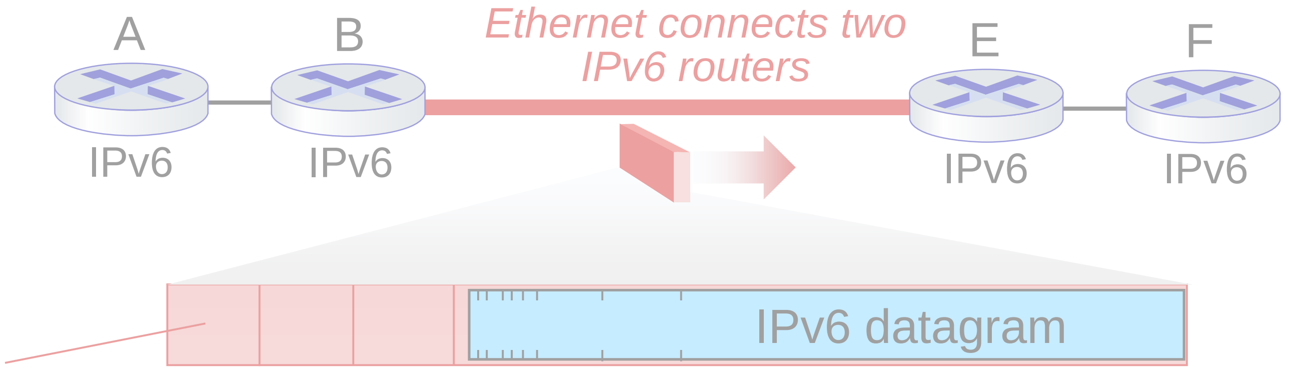
Transition from IPv4 to IPv6

- 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 used extensively in other contexts (4G/5G)



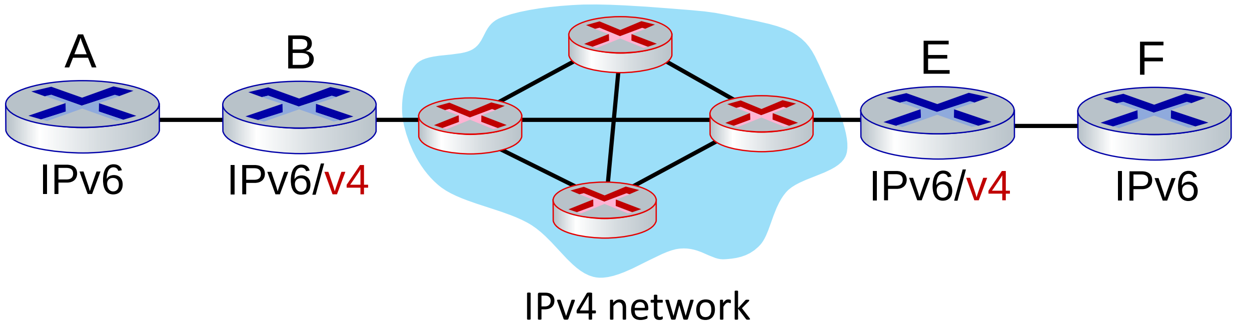
Tunneling and encapsulation

Ethernet connecting two IPv6 routers:



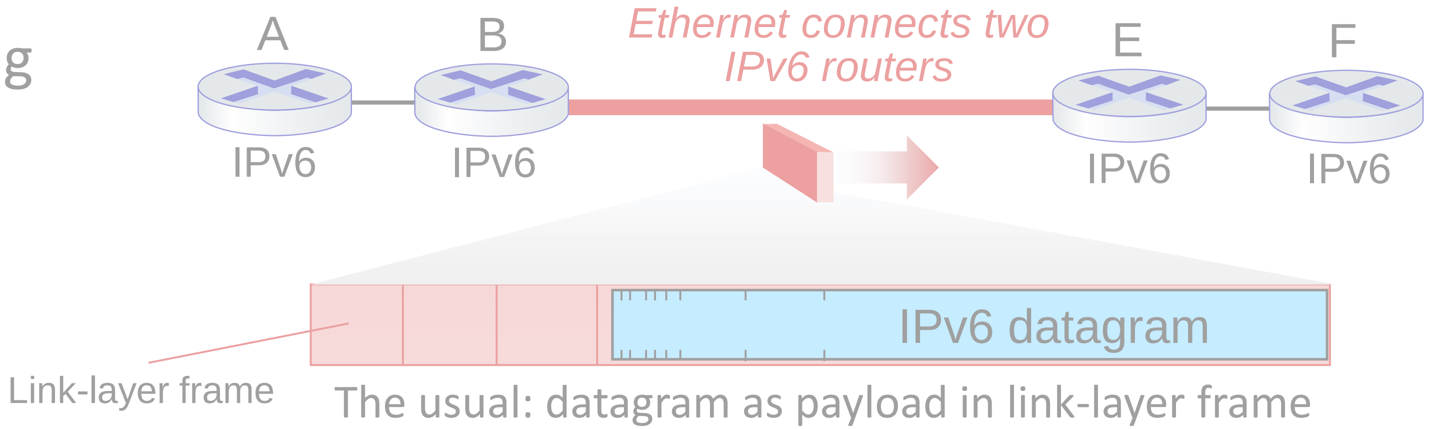
The usual: datagram as payload in link-layer frame

IPv4 network connecting two IPv6 routers



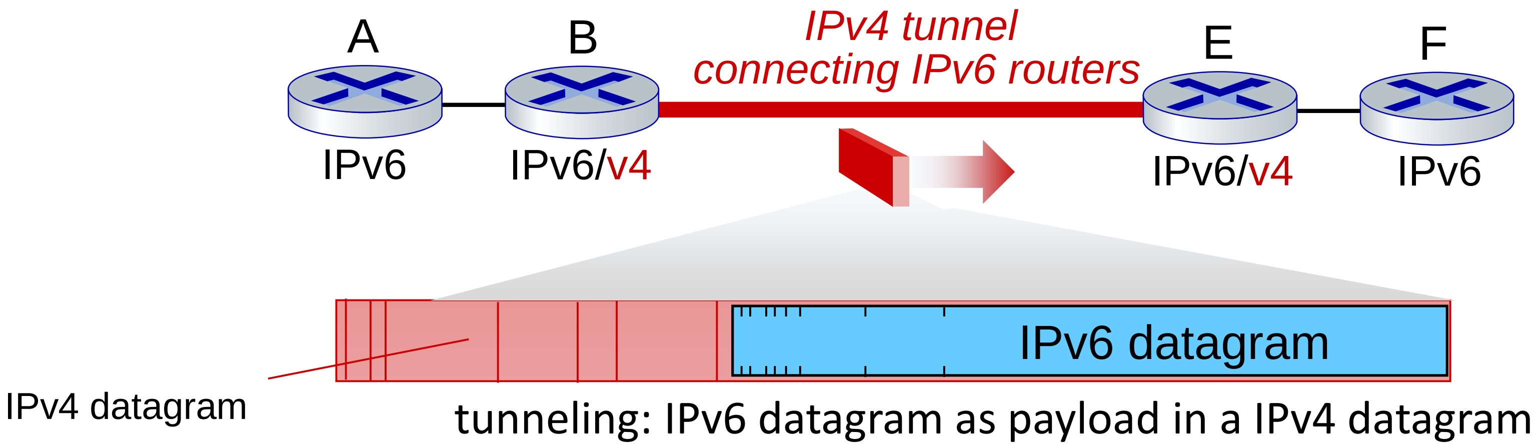
Tunneling and encapsulation

Ethernet connecting two IPv6 routers:



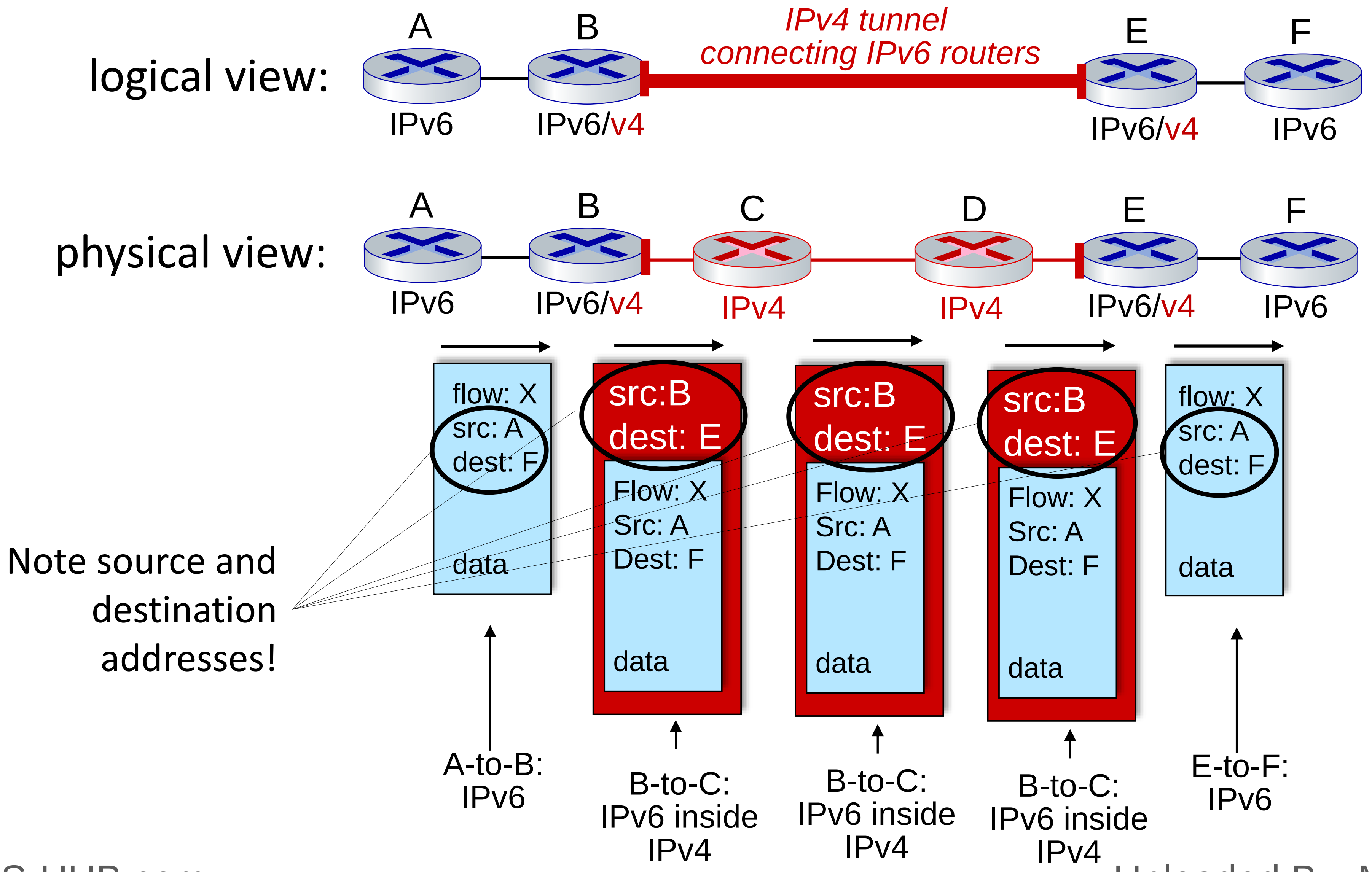
The usual: datagram as payload in link-layer frame

IPv4 tunnel connecting two IPv6 routers



tunneling: IPv6 datagram as payload in a IPv4 datagram

Tunneling


logical view:
A
IPv6
B
IPv6/v4
IPv4 tunnel
connecting IPv6 routers
E
IPv6/v4
F
IPv6
physical view:
A
IPv6
B
IPv6/v4
C
IPv4
D
IPv4
E
IPv6/v4
F
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
B-to-C: IPv6 inside IPv4
B-to-C: IPv6 inside IPv4
E-to-F: IPv6

Network layer: “data plane” roadmap

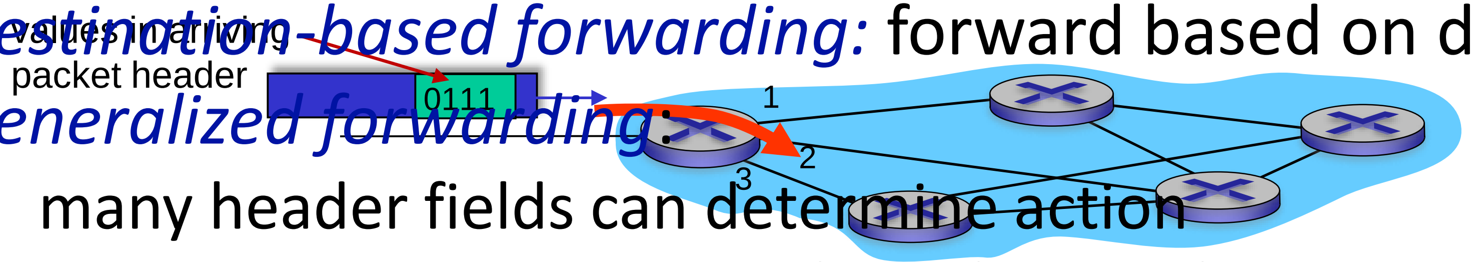
- Network layer: overview
  - data plane
  - control plane
- What’s inside a router
  - input ports, switching, output ports
  - buffer management, scheduling
- IP: the Internet Protocol
  - datagram format
  - addressing
  - network address translation
  - IPv6

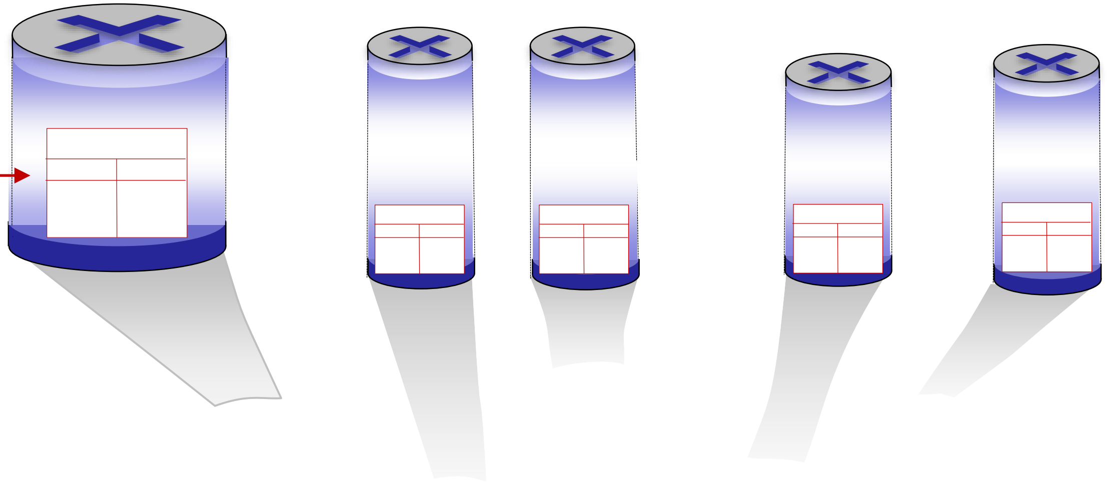
- Generalized Forwarding, SDN
  - Match+action
  - OpenFlow: match+action in action

Generalized forwarding: match plus action

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

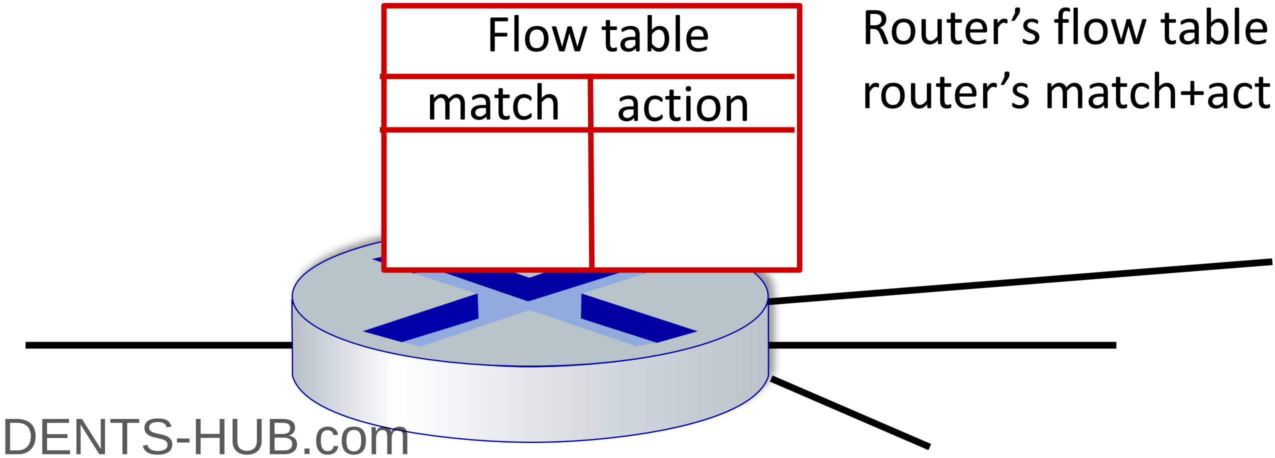
- "match plus action" abstraction: match bits in arriving packet, take action
  - *destination-based forwarding:* forward based on dest. IP address
  - *generalized forwarding:*
    - many header fields can determine action
    - many action possible: drop/copy/modify/log packet





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

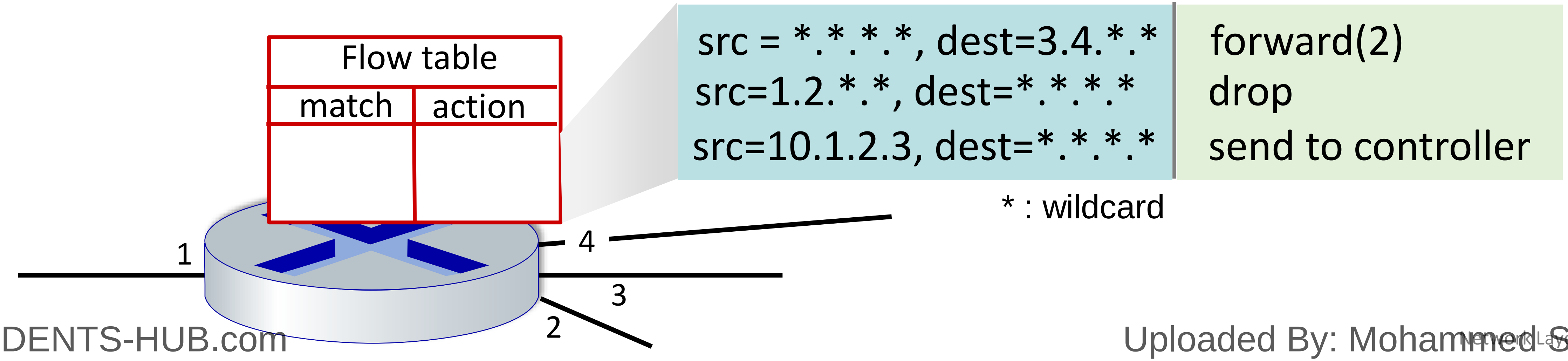


Flow table	
match	action

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

Flow table abstraction

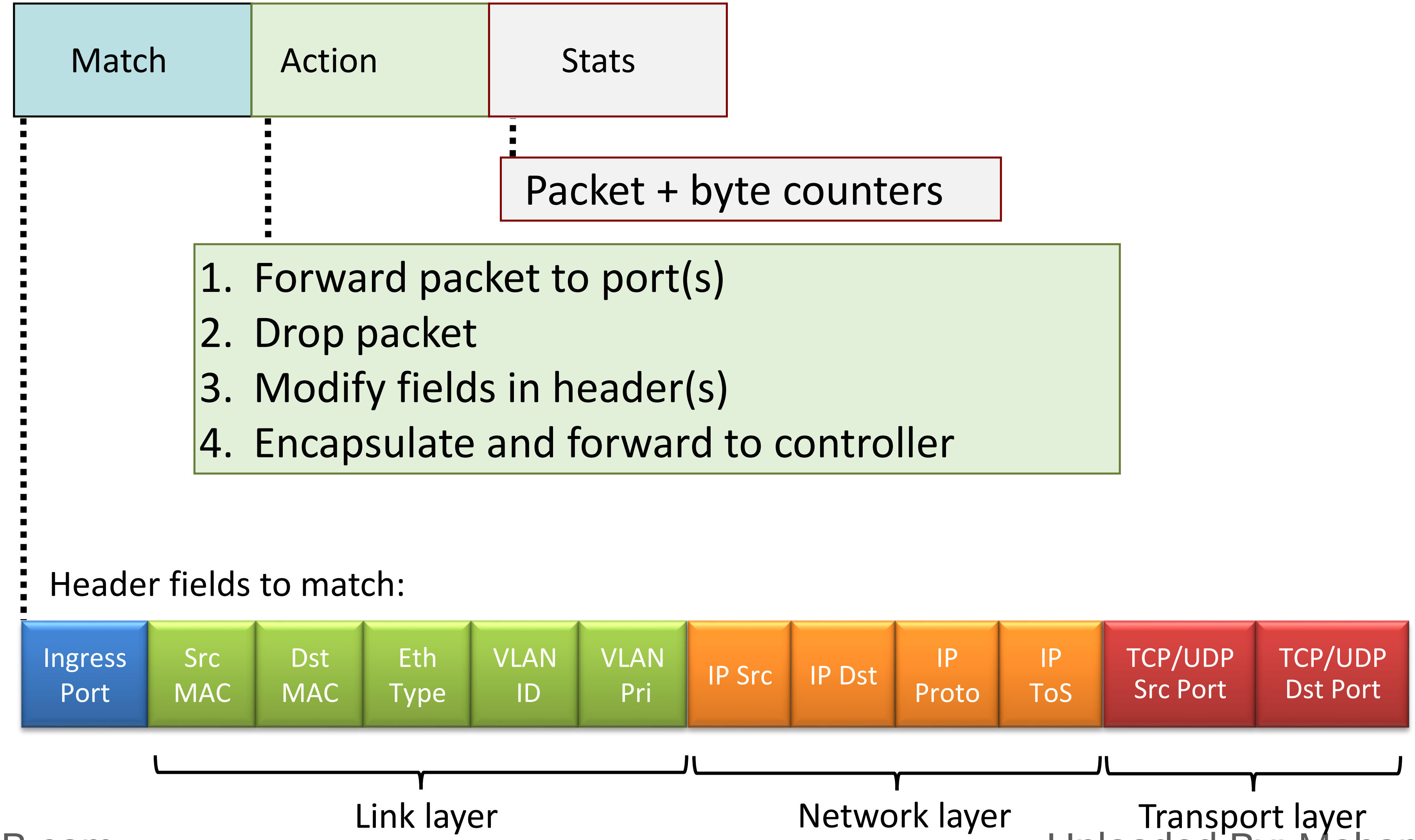
- flow: defined by header fields
- generalized forwarding: simple packet-handling rules
  - match: pattern values in packet header fields
  - actions: for matched packet: drop, forward, modify, matched packet or send matched packet to controller
  - priority: disambiguate overlapping patterns
  - counters: #bytes and #packets



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



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

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

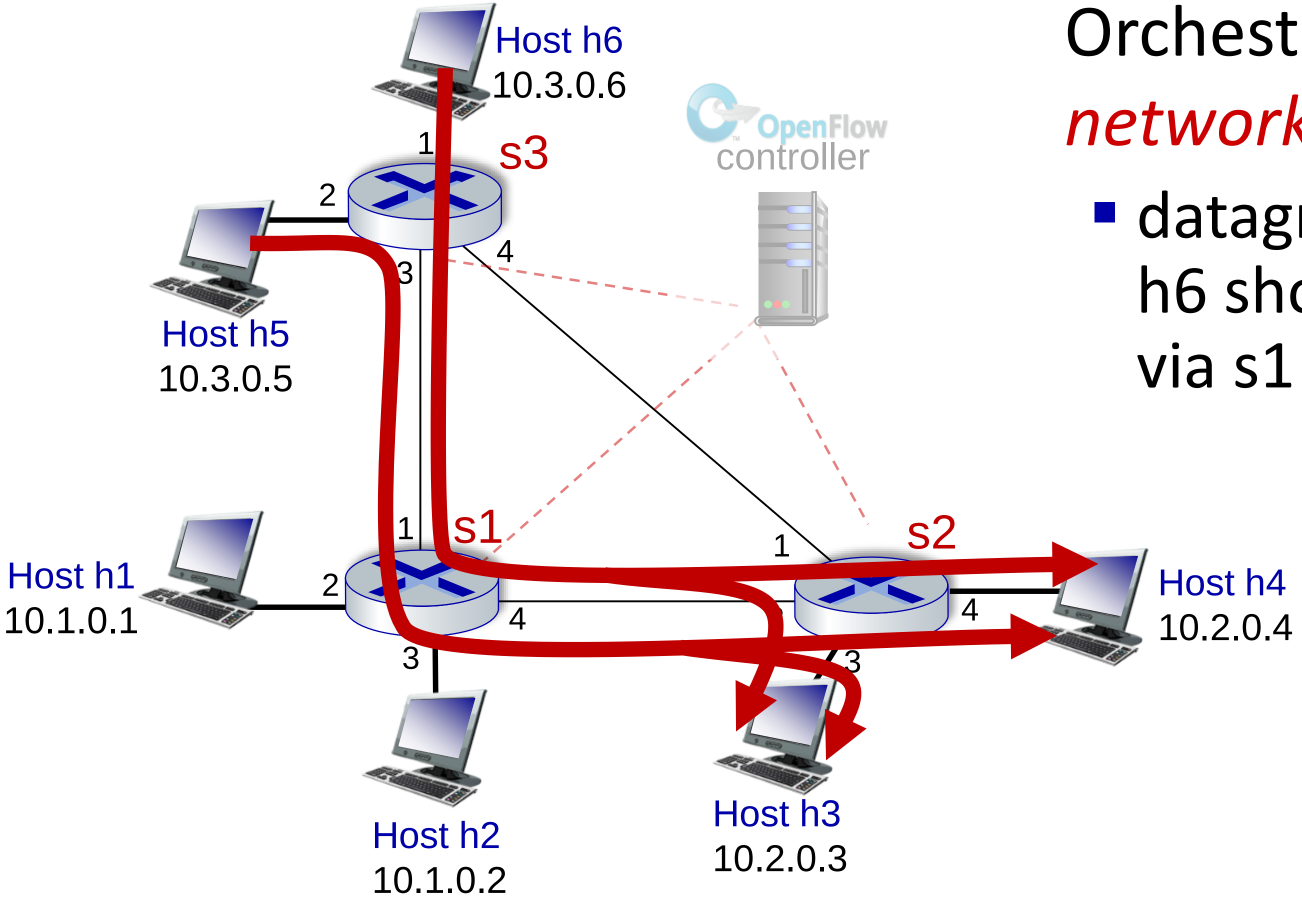
Firewall

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

NAT

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

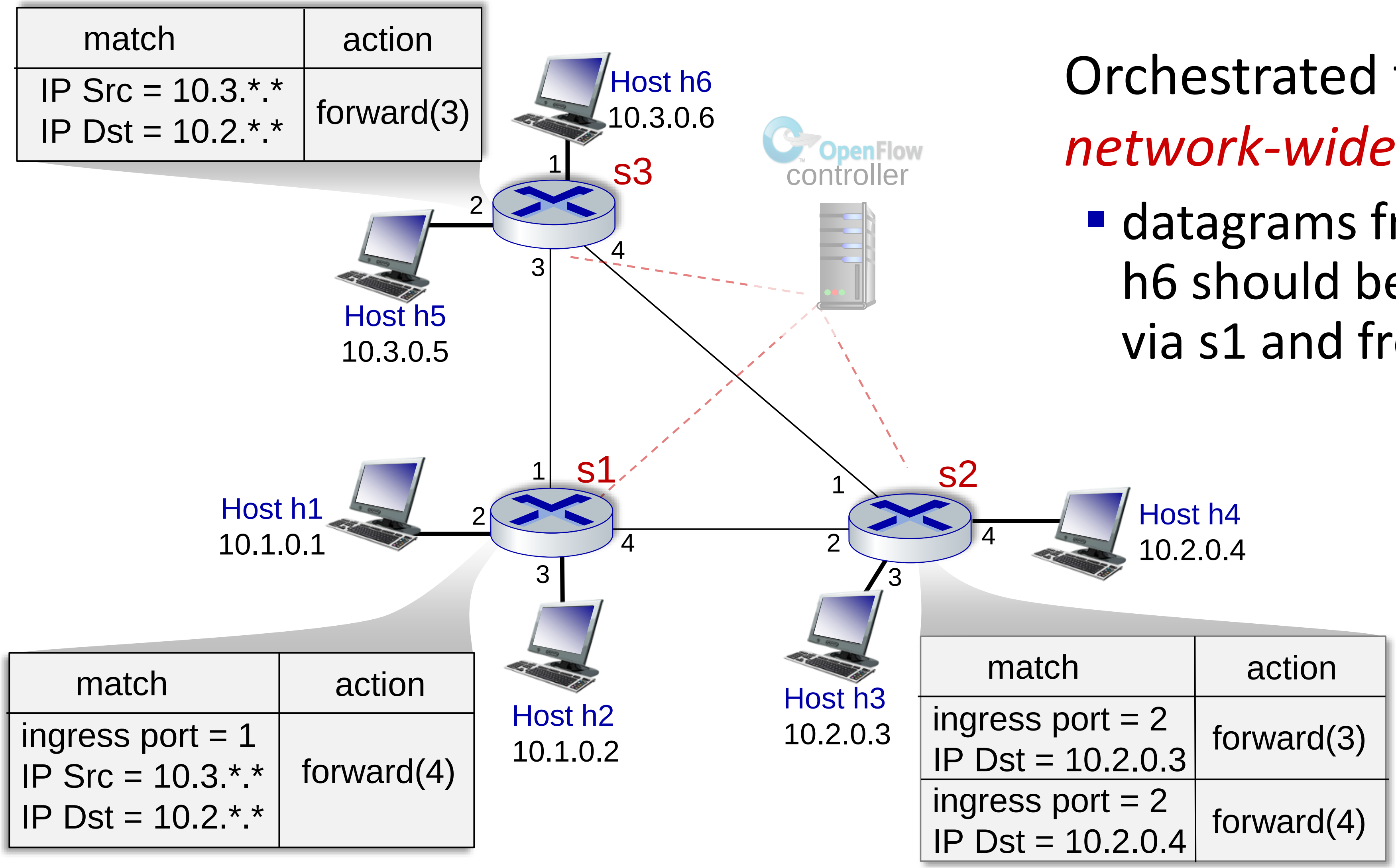
OpenFlow example



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)

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).

Chapter 4: done!

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

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

Answer: by the control plane (next chapter)

Additional Chapter 4 slides

DHCP: Wireshark output (home LAN)

request

```
Message type: Boot Request (1)
Hardware type: Ethernet
Hardware address length: 6
Hops: 0
Transaction ID: 0x6b3a11b7
Seconds elapsed: 0
Bootp flags: 0x0000 (Unicast)
Client IP address: 0.0.0.0 (0.0.0.0)
Your (client) IP address: 0.0.0.0 (0.0.0.0)
Next server IP address: 0.0.0.0 (0.0.0.0)
Relay agent IP address: 0.0.0.0 (0.0.0.0)
Client MAC address: Wistron_23:68:8a (00:16:d3:23:68:8a)
Server host name not given
Boot file name not given
Magic cookie: (OK)
Option: (t=53,l=1) DHCP Message Type = DHCP Request
Option: (61) Client identifier
    Length: 7; Value: 010016D323688A;
    Hardware type: Ethernet
    Client MAC address: Wistron_23:68:8a (00:16:d3:23:68:8a)
Option: (t=50,l=4) Requested IP Address = 192.168.1.101
Option: (t=12,l=5) Host Name = "nomad"
Option: (55) Parameter Request List
    Length: 11; Value: 010F03062C2E2F1F21F92B
    1 = Subnet Mask; 15 = Domain Name
    3 = Router; 6 = Domain Name Server
    44 = NetBIOS over TCP/IP Name Server
    ……
```

reply

```
Message type: Boot Reply (2)
Hardware type: Ethernet
Hardware address length: 6
Hops: 0
Transaction ID: 0x6b3a11b7
Seconds elapsed: 0
Bootp flags: 0x0000 (Unicast)
Client IP address: 192.168.1.101 (192.168.1.101)
Your (client) IP address: 0.0.0.0 (0.0.0.0)
Next server IP address: 192.168.1.1 (192.168.1.1)
Relay agent IP address: 0.0.0.0 (0.0.0.0)
Client MAC address: Wistron_23:68:8a (00:16:d3:23:68:8a)
Server host name not given
Boot file name not given
Magic cookie: (OK)
Option: (t=53,l=1) DHCP Message Type = DHCP ACK
Option: (t=54,l=4) Server Identifier = 192.168.1.1
Option: (t=1,l=4) Subnet Mask = 255.255.255.0
Option: (t=3,l=4) Router = 192.168.1.1
Option: (6) Domain Name Server
    Length: 12; Value: 445747E2445749F244574092;
     IP Address: 68.87.71.226;
     IP Address: 68.87.73.242;
     IP Address: 68.87.64.146
Option: (t=15,l=20) Domain Name = "hsd1.ma.comcast.net."
```