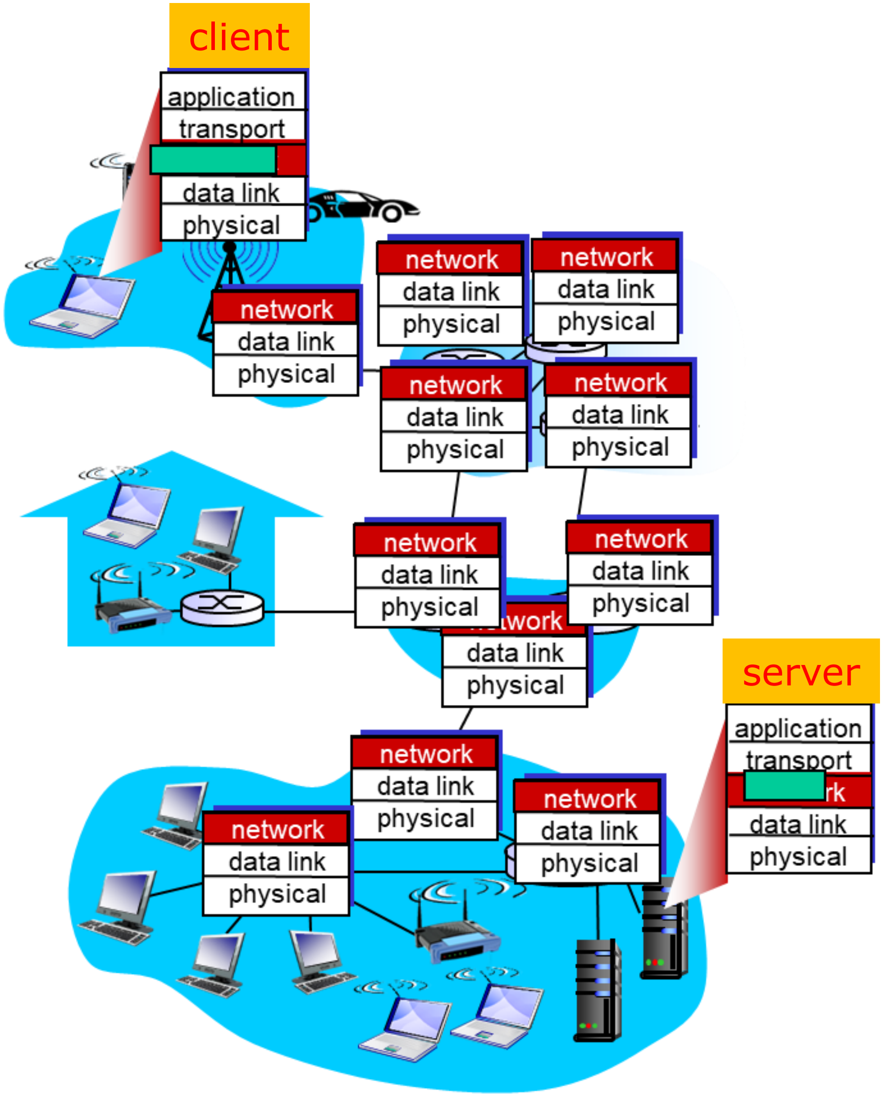
Chap 4. Network Layer: Data Plane

- Overview of Network Layer
- What’s inside a router?
- IP: Internet Protocol
- Generalized Forward and SDN

Network layer

- **transport layer protocols** on sending host and receiving host
- **network layer protocols** in *every* host and router
- router examines header fields in all IP datagrams passing through it



Two key network-layer functions

network-layer functions:

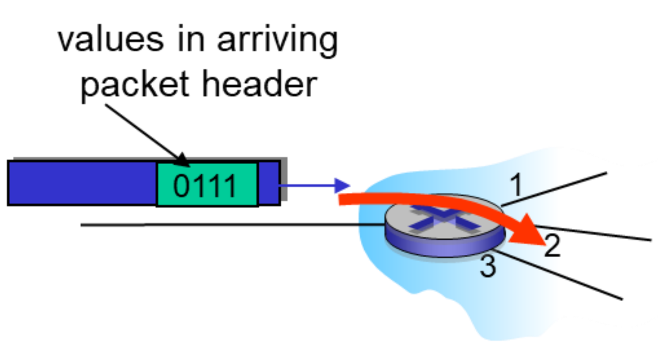
- forwarding: move packets from router's input to appropriate router output
- routing: determine route taken by packets from source to destination
  - routing algorithms

- Routing
  - pro-active routing: IP routing
  - re-active routing: mobile wireless networks

Network layer: data plane, control plane

Data plane

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

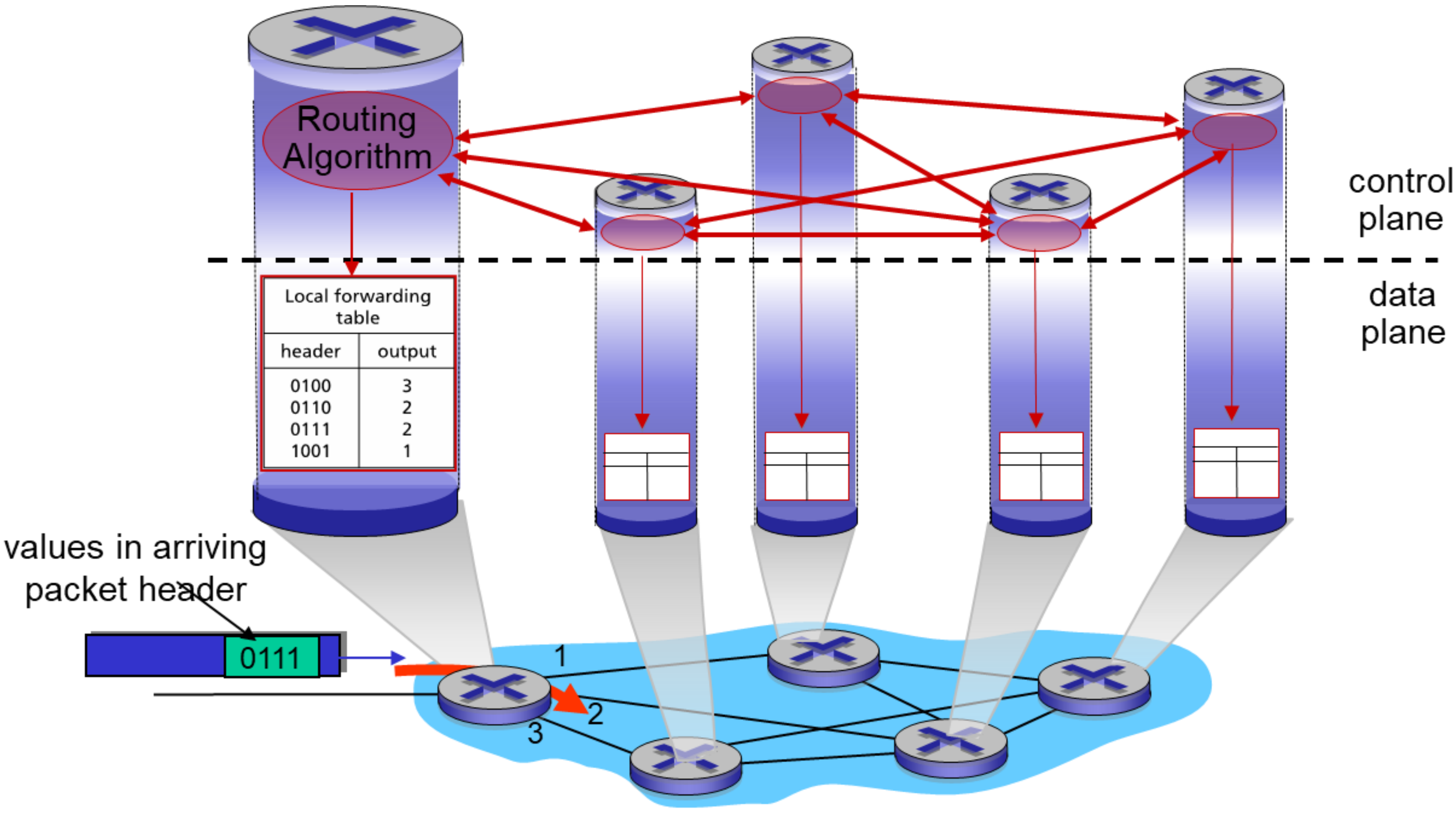


Control plane

- routing function
- determines e2e-path from source host to destination host
- two control-plane approaches:
  - traditional routing algorithms: implemented in distributed routers
  - SDN: implemented in centralized servers

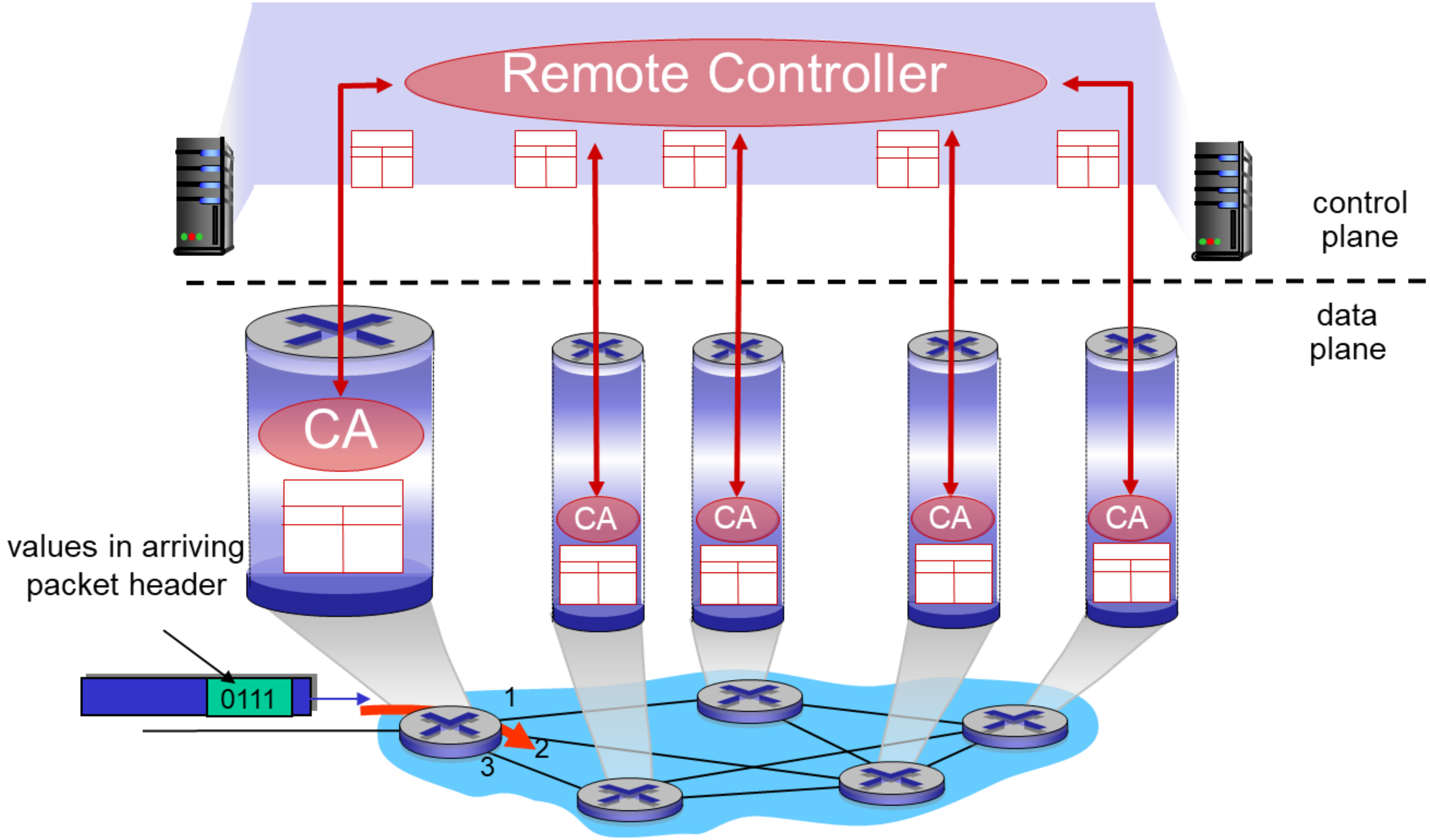
Traditional routing: Per-router control plane

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

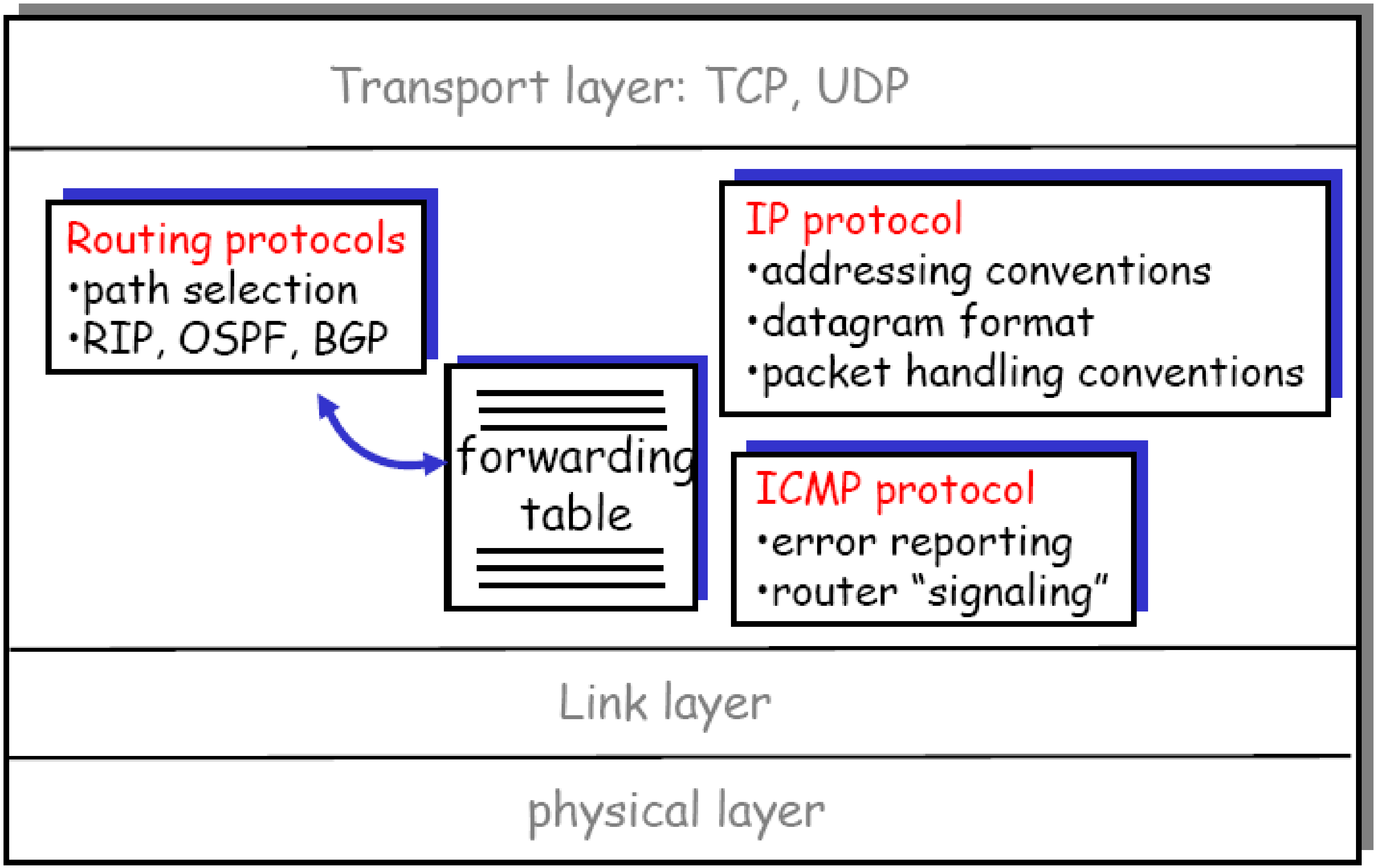


SDN: Logically centralized control plane

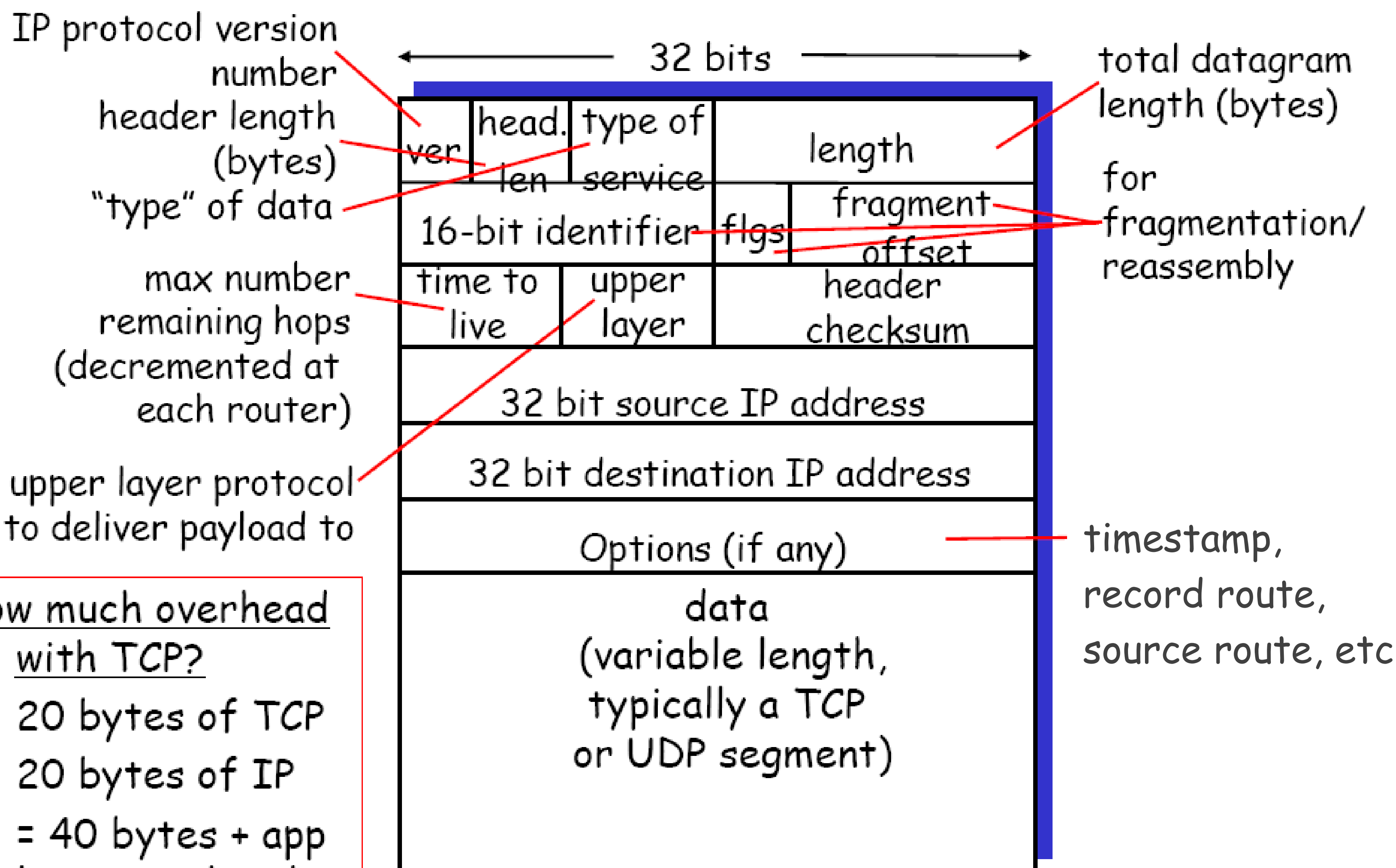
- A distinct (typically remote) controller interacts with local control agents (CAs)



Internet Network layer



IP datagram format

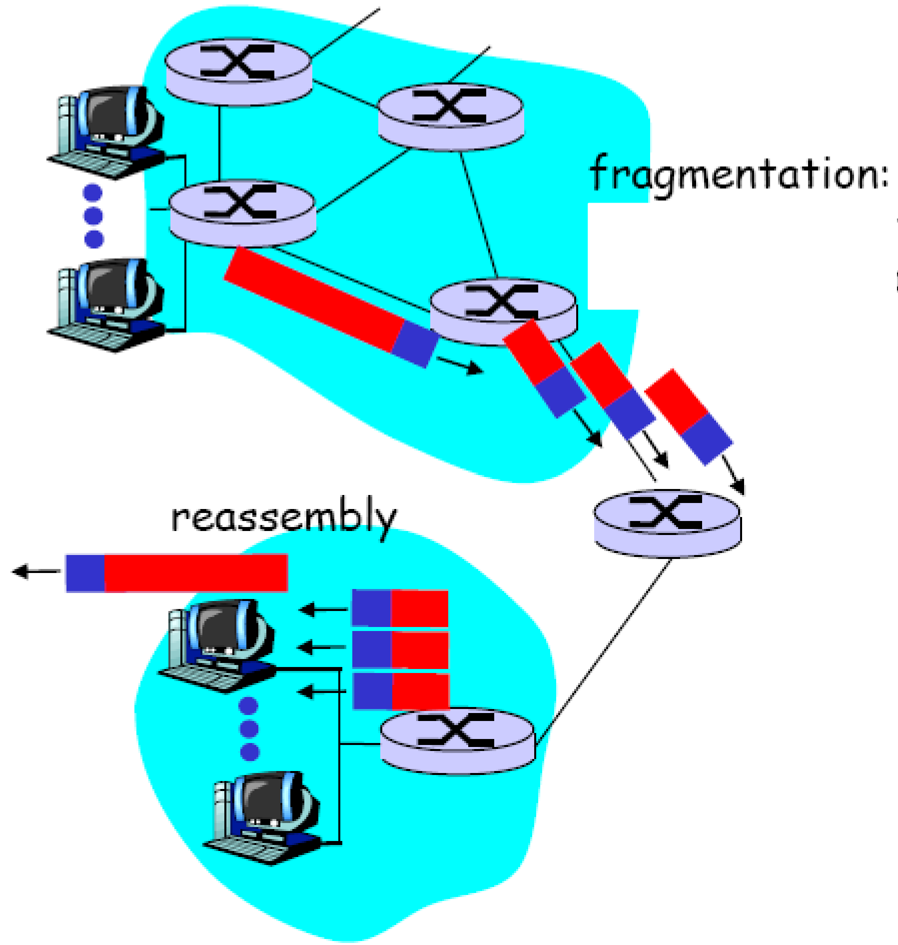


how much overhead with TCP?

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

IP Fragmentation & Reassembly

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

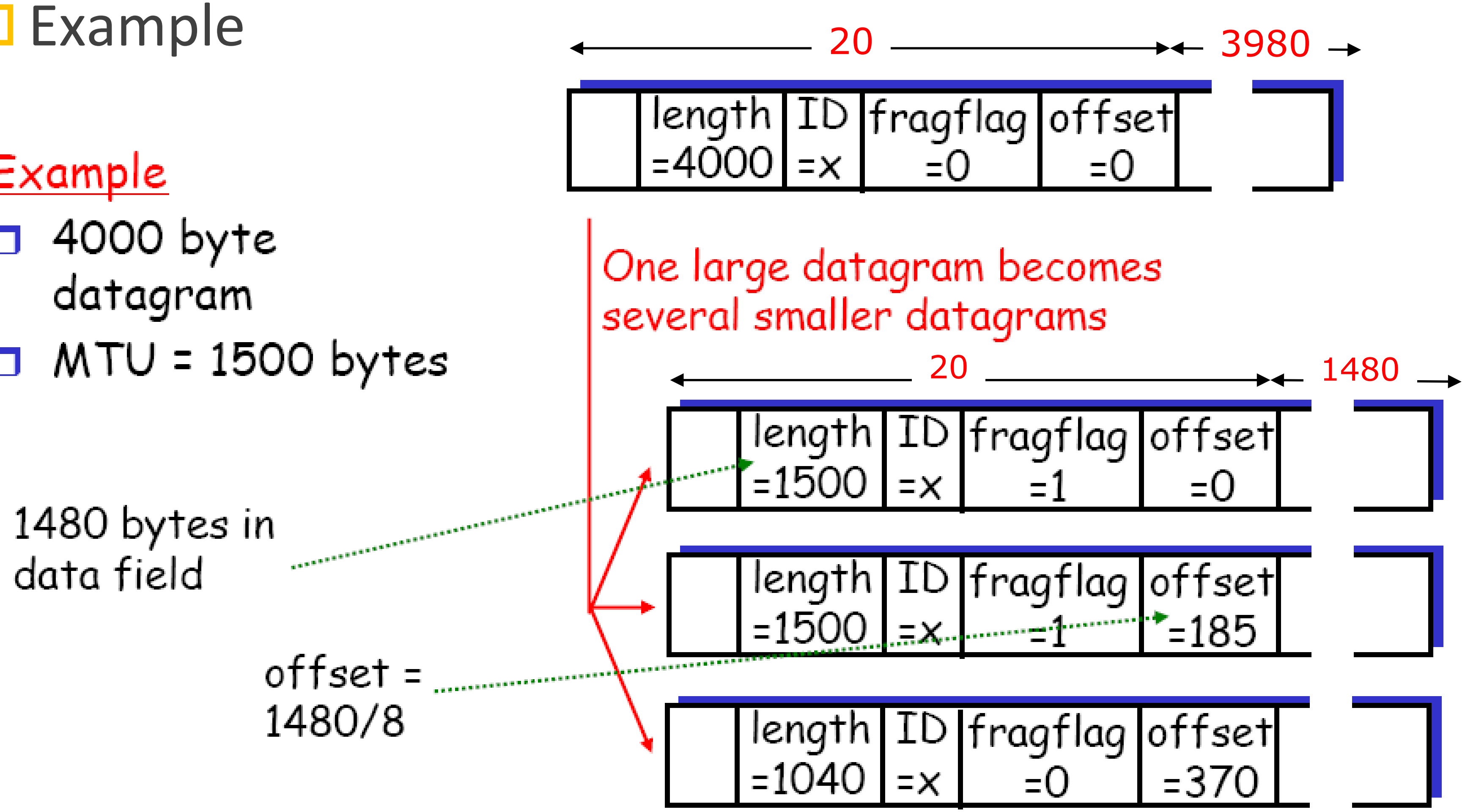


IP Fragmentation & Reassembly

- Example

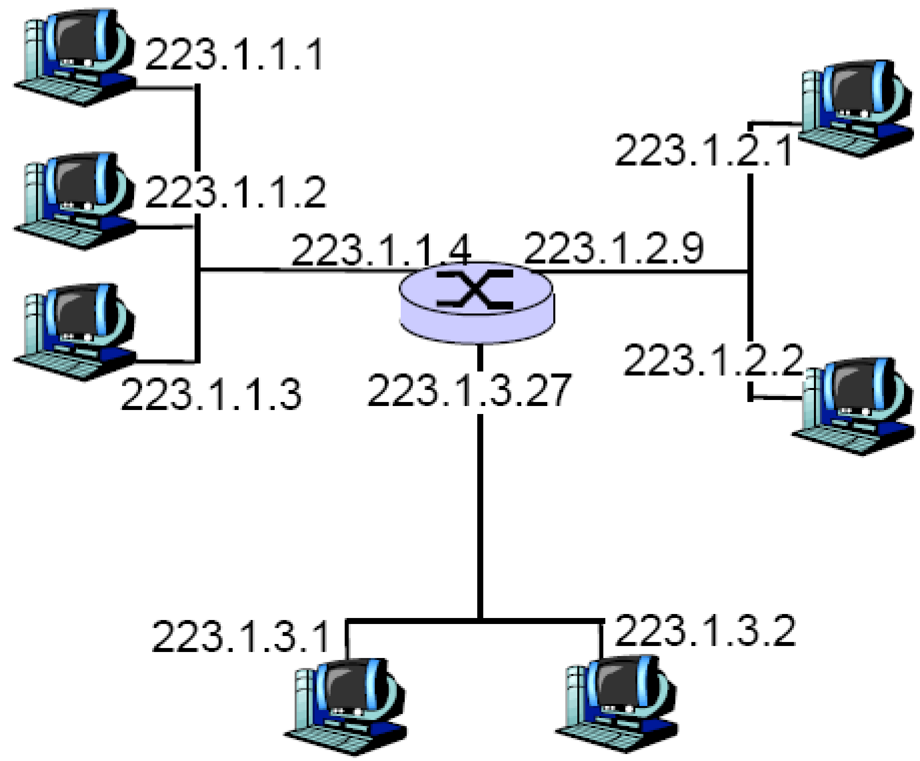
<u>Example</u>

- 4000 byte datagram
- MTU = 1500 bytes



IP Addressing

- IP address: 32-bit ID for network interface
- *interface:* connection b/w host/router and physical link
  - router's typically have multiple interfaces
  - host typically has one interface
  - IP addresses associated with each interface

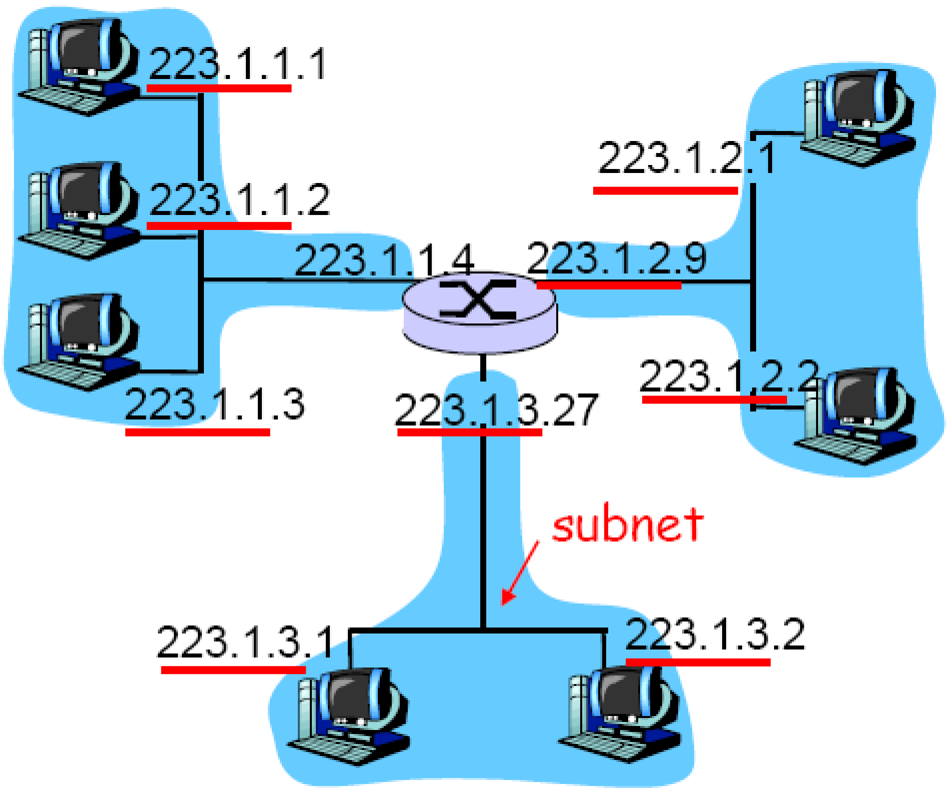


223.1.1.1 = 11011111 00000001 00000001 00000001

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

Subnets

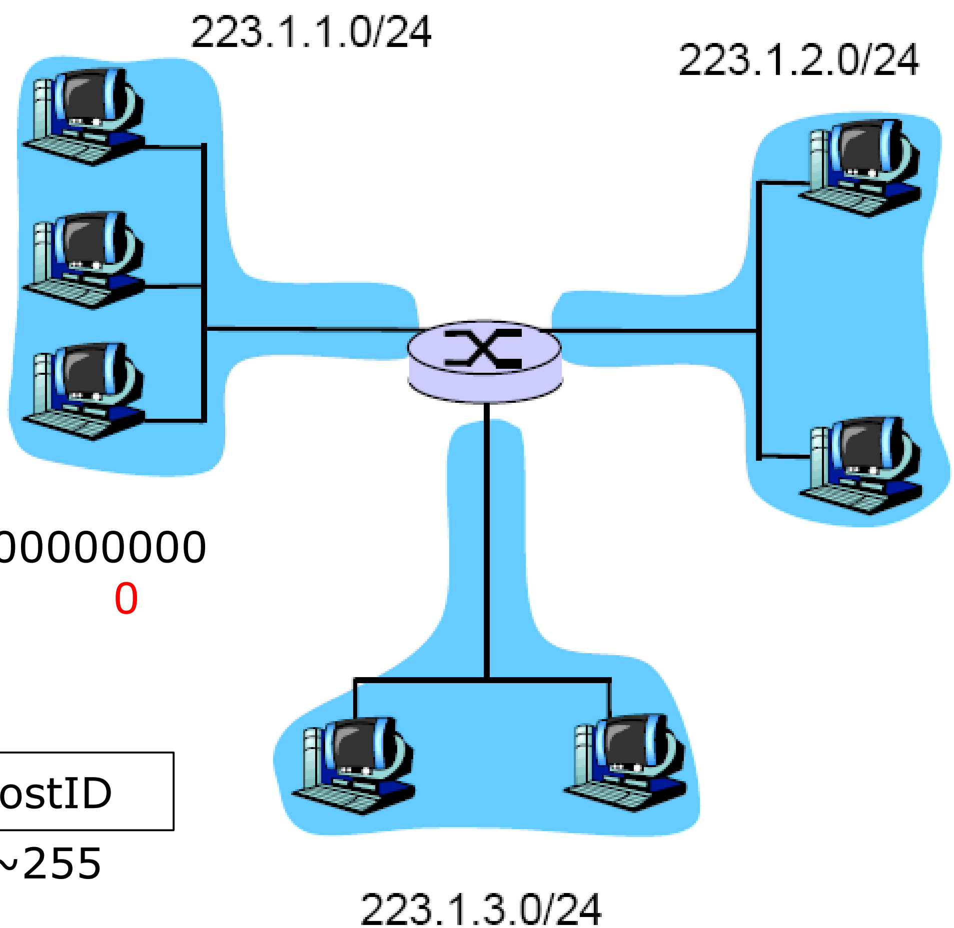
- IP address:
  - subnet part (high order bits)
  - host part (low order bits)
- *What's a subnet ?*
  - device interfaces with same subnet part of IP address
  - can reach each other without intervening router



network consisting of 3 subnets

Subnets

- Subnet mask
  - The number of bits that denote network id in the IP address
  - 223.1.1.0/24

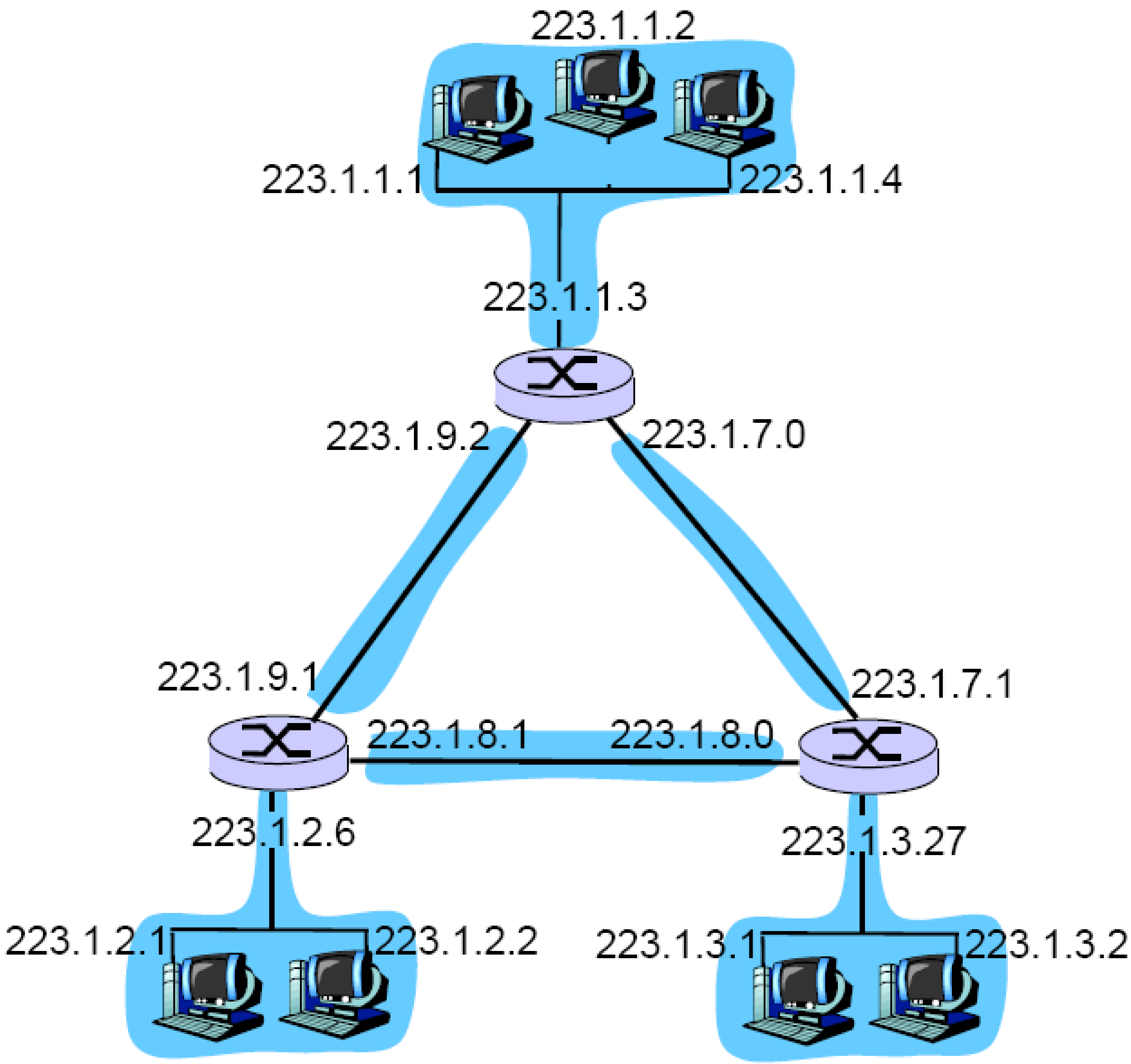


11111111	11111111	11111111	00000000
255 .	255 .	255 .	0

network addr.	223	1	1	hostID
				0~255
broadcast addr.	223	1	1	11111111

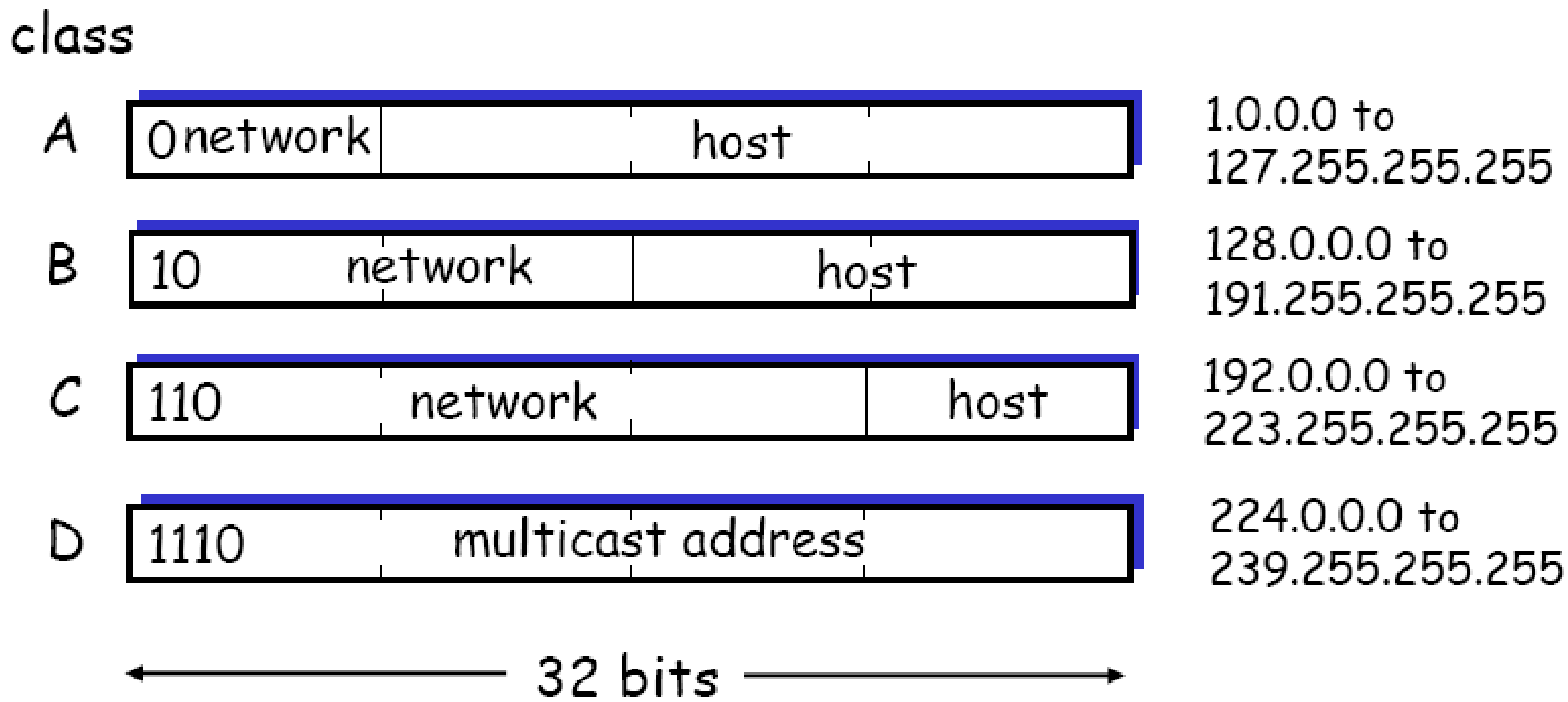
Subnets

- How many subnets?



IP addressing: Classful Addressing

- Classful addressing



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

11111111 11111111 11111110 00000000

255 . 255 . 254 . 0

IP addresses: how to get one?

Q: How does a *host* get IP address?

- static address: hard-coded by system admin in a file
  - Windows: control-panel → network → configuration → tcp/ip → properties
  - UNIX: /etc/rc.config
- dynamic address: DHCP(Dynamic Host Configuration Protocol)
  - dynamically get address from as server
  - “plug-and-play”

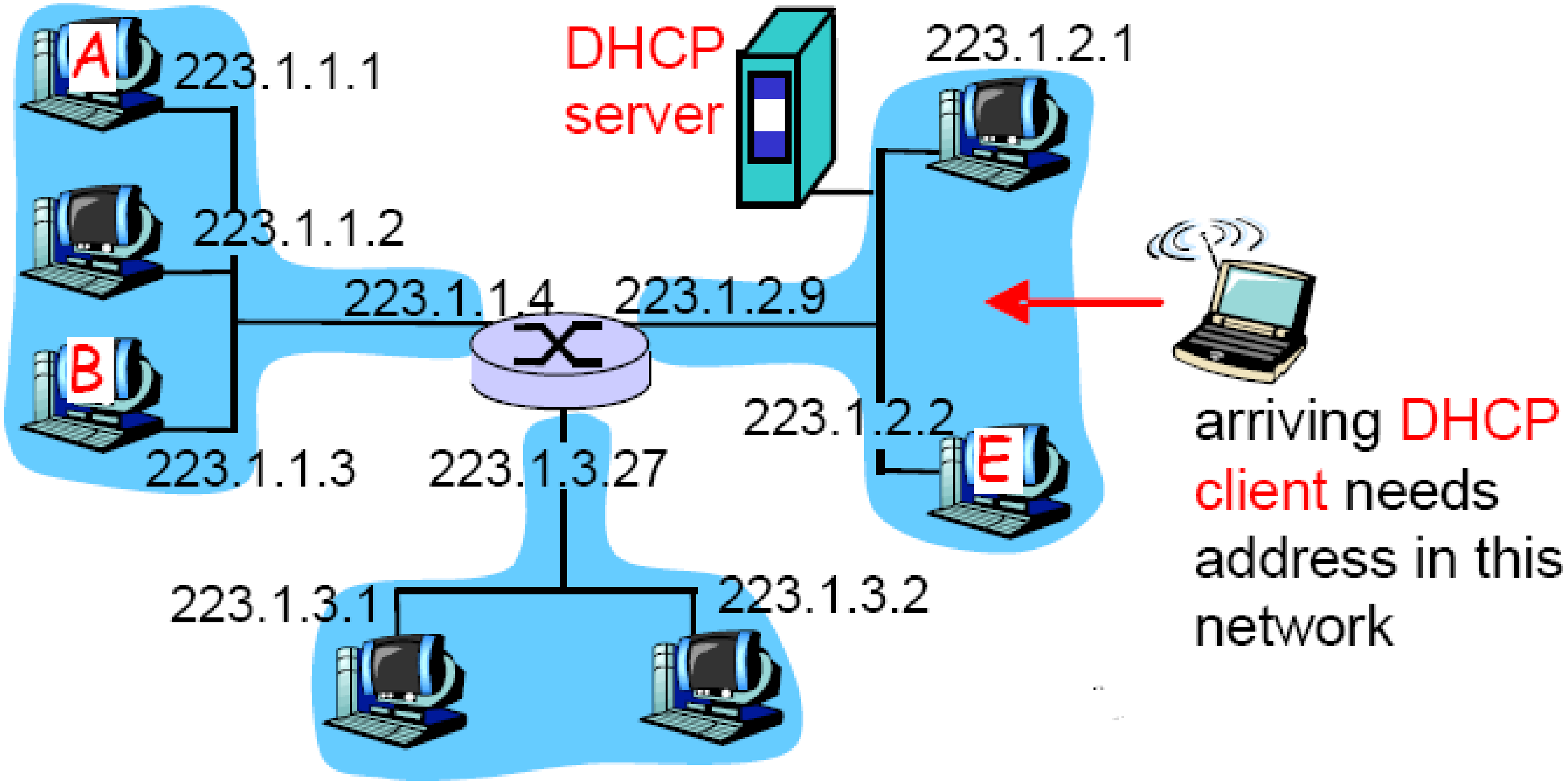
DHCP: Dynamic Host Configuration Protocol

Goal: allow host to *dynamically* obtain its IP address from network server when it joins network

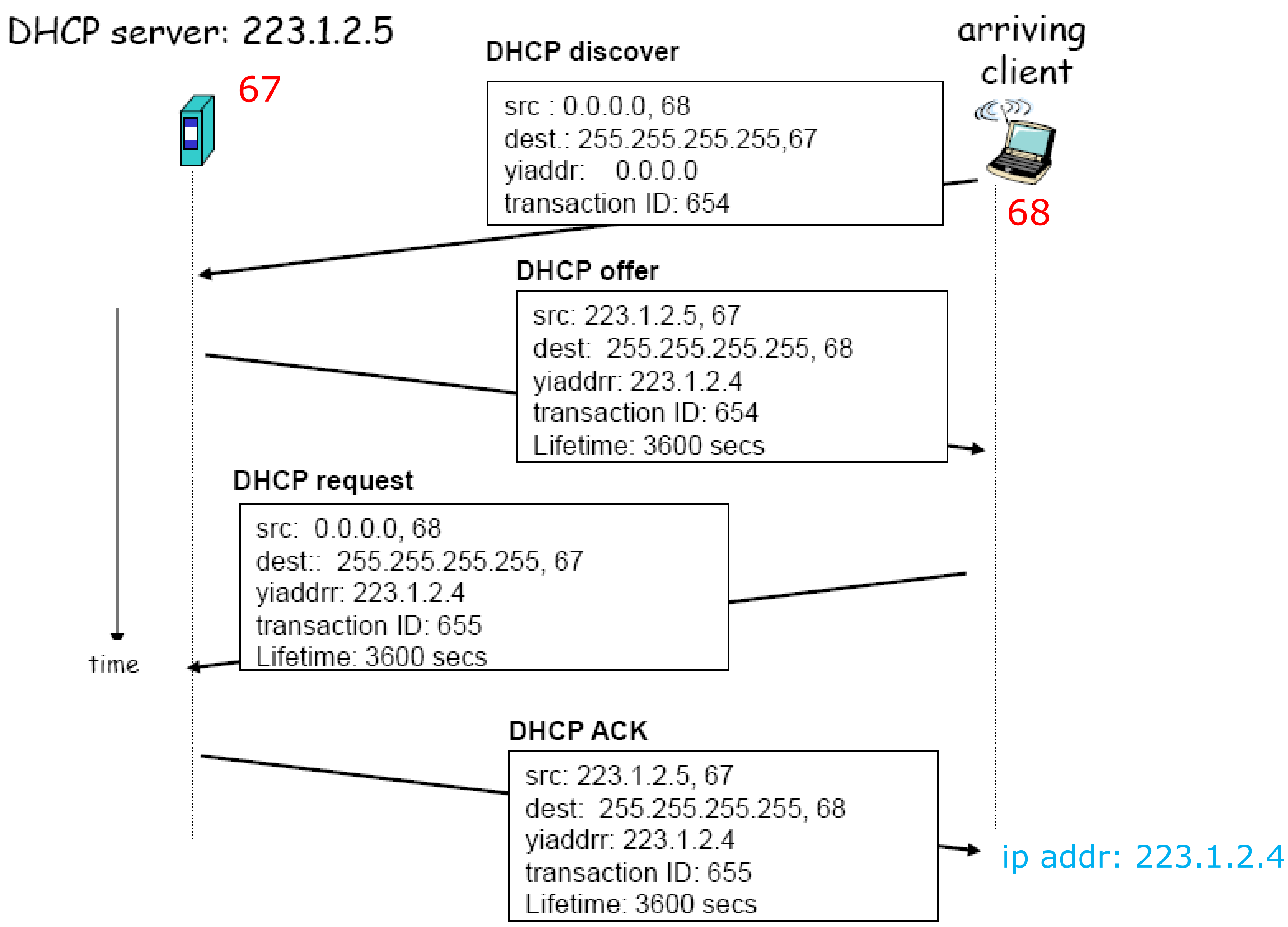
- DHCP overview:
  - host broadcasts “DHCP discover” msg
  - DHCP server responds with “DHCP offer” msg
  - host requests IP address: “DHCP request” msg
  - DHCP server sends address: “DHCP ack” msg

  - DHCP uses UDP ports 67(server) and 68(client)

DHCP client-server scenario



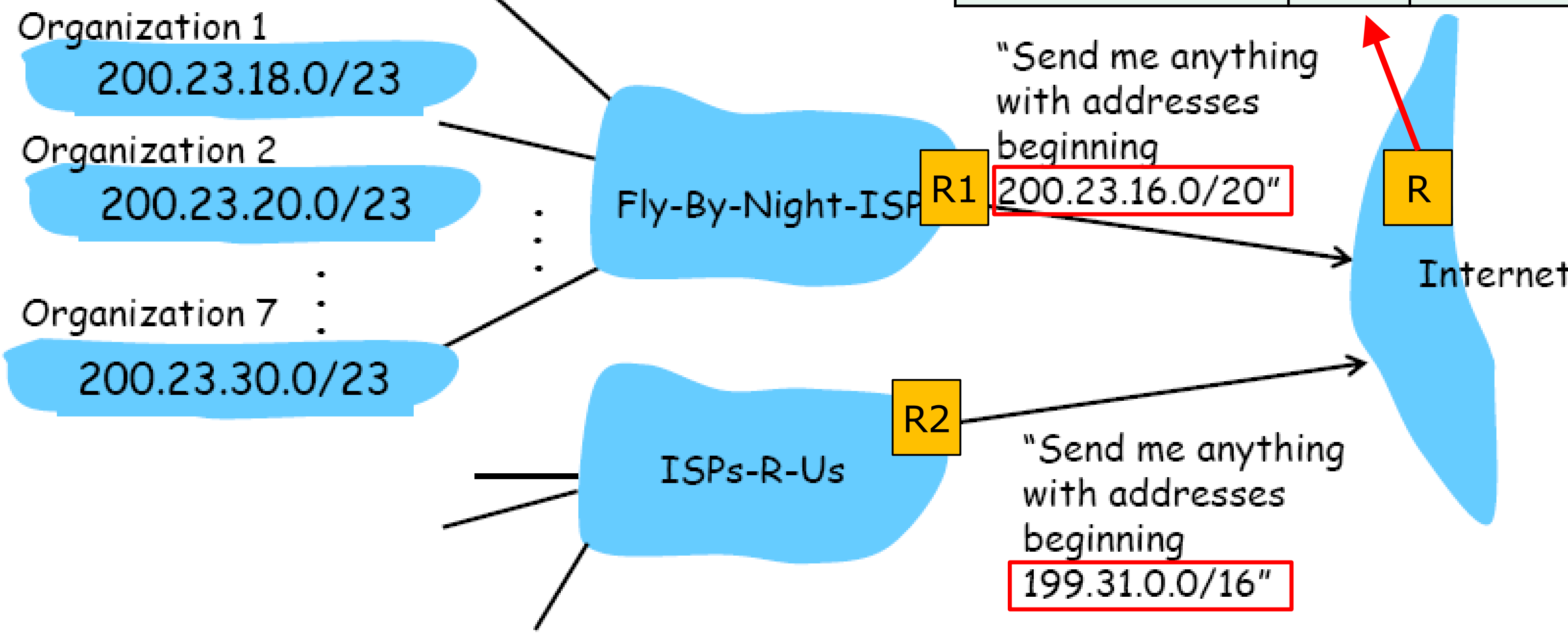
DHCP client-server scenario



Hierarchical addressing

- Route aggregation: hierarchical addressing allows efficient advertisement of routing information

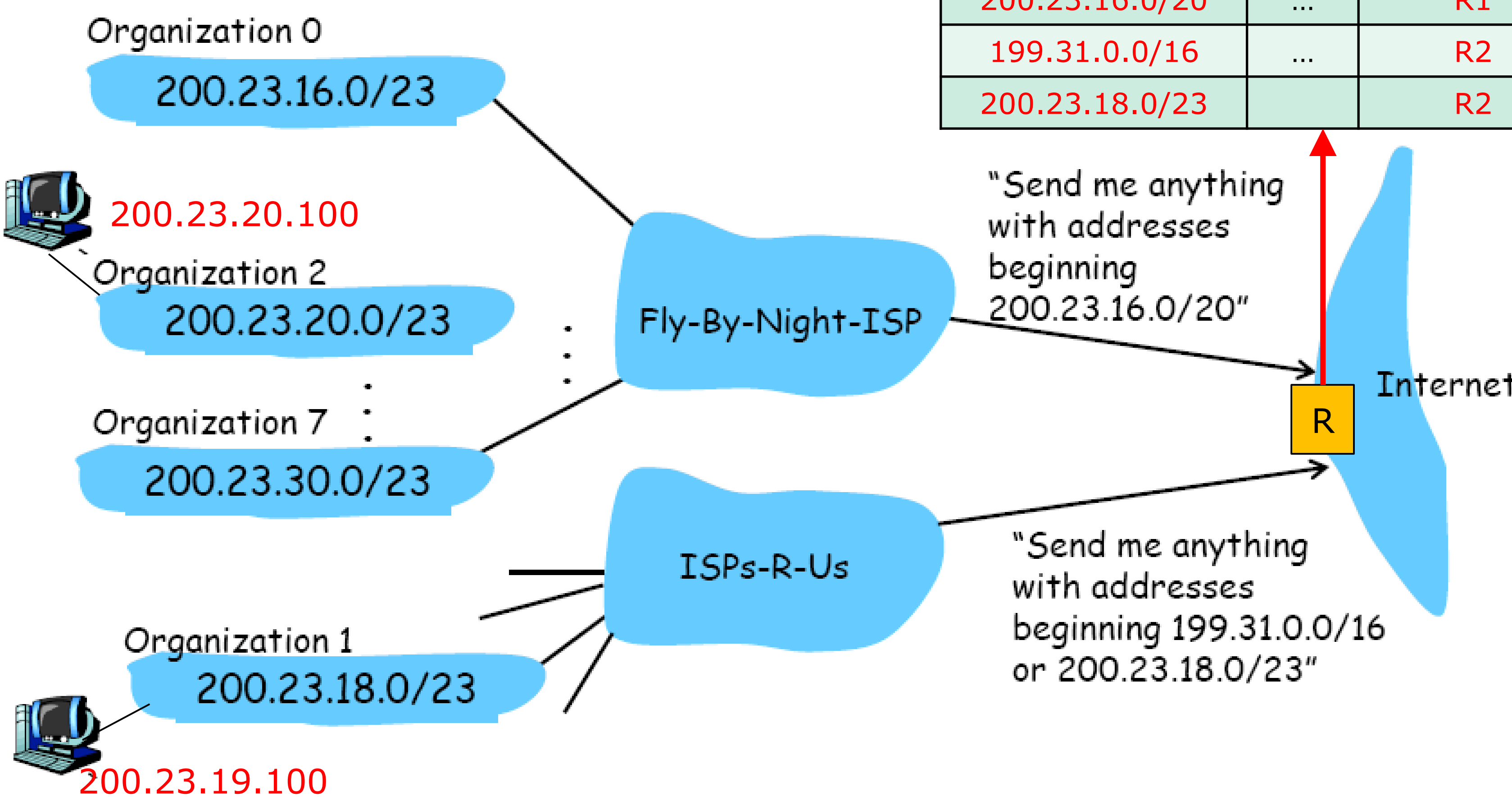
Dest	...	NextHop
200.23.16.0/20	...	R1
199.31.0.0/16	...	R2



Hierarchical addressing

- More specific routes: chooses the longest match

Dest	...	NextHop
200.23.16.0/20	...	R1
199.31.0.0/16	...	R2
200.23.18.0/23		R2



IP addressing

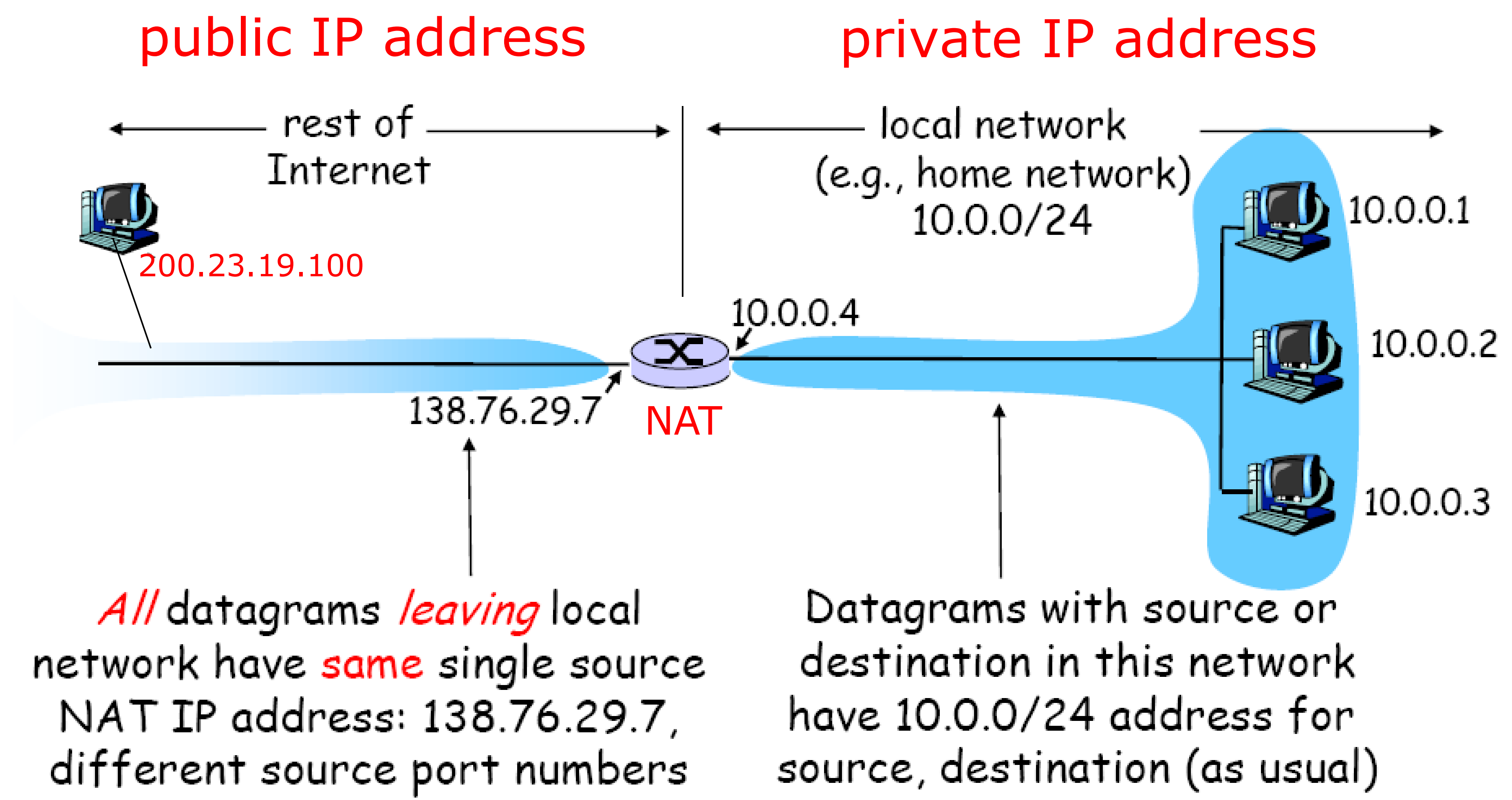
Q: How does an ISP get block of addresses?

A: ICANN: (Internet Corporation for Assigned Names and Numbers)

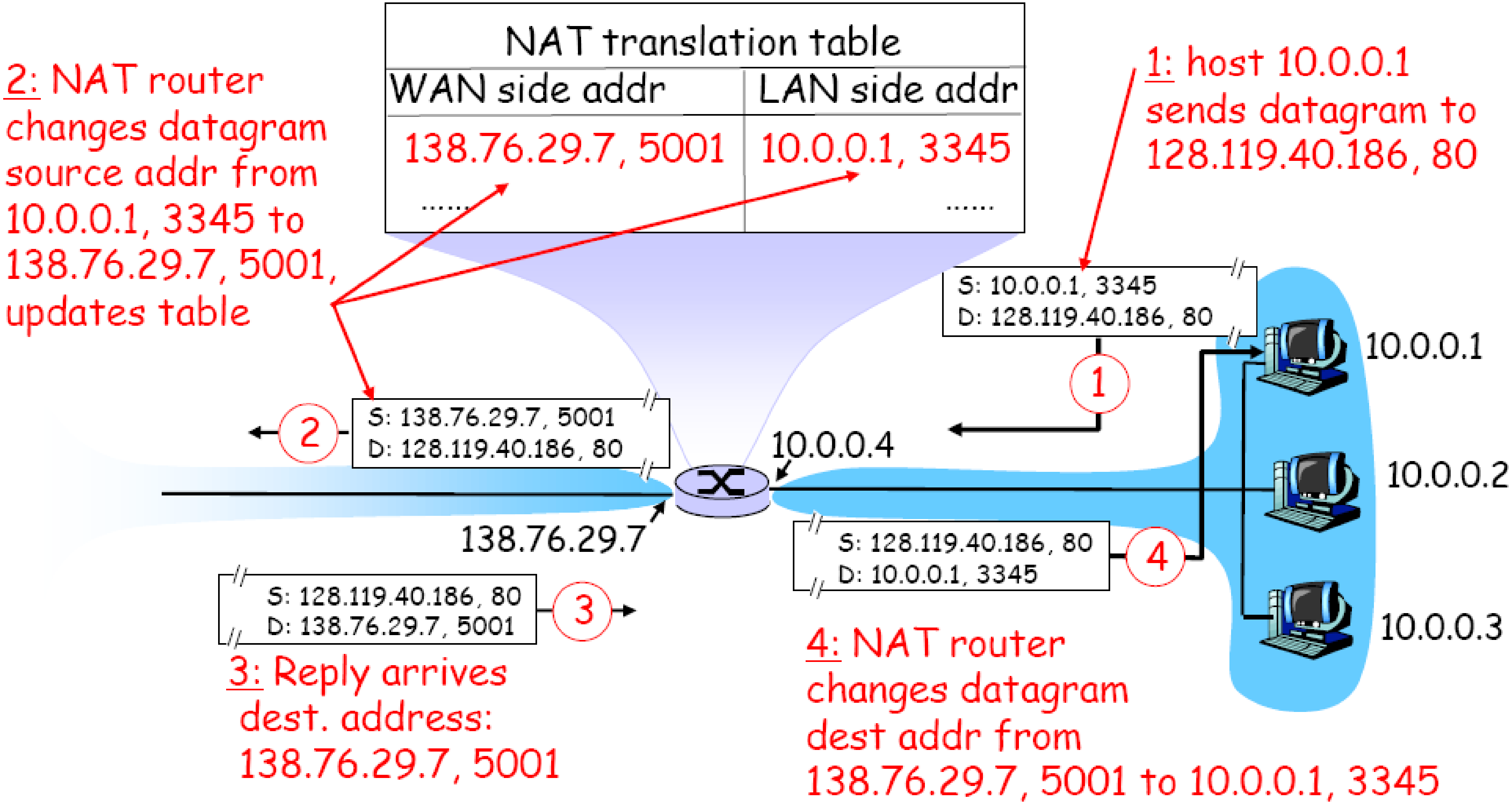
- allocates addresses
- manages DNS
- assigns domain names, resolves disputes

Class	Format	Range	private IP addr.
A	0network / host	1.0.0.0 to 127.255.255.255	10.x.x.x
B	10 network / host	128.0.0.0 to 191.255.255.255	172.16.x.x
C	110 network / host	192.0.0.0 to 223.255.255.255	192.168.x.x

NAT: Network Address Translation



NAT: Network Address Translation



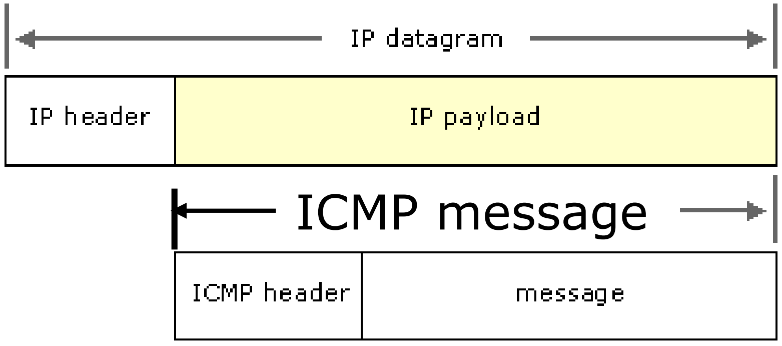
NAT: Network Address Translation

- Implementation
  - *outgoing datagrams: replace* (source IP address, port #) of every outgoing datagram to (NAT IP addr, new port #)
  - *remember (in NAT translation table)* every (source IP address, port #) to (NAT IP addr, new port #) translation mapping
  - *incoming datagrams: replace* (NAT IP address, new port #) in dest fields of every incoming datagram with corresponding (source IP addr, port #) stored in NAT table

ICMP: Internet Control Message Protocol

- used by hosts & routers to communicate network-level information
  - error reporting: unreachable host, network, port, protocol
  - echo request/reply (used by ping)
- network-layer "above" IP:
  - ICMP msgs carried in IP datagrams
- ICMP message: type, code plus first 8 bytes of IP datagram causing error

Type	Code	description
0	0	echo reply (ping)
3	0	dest. network unreachable
3	1	dest host unreachable
3	2	dest protocol unreachable
3	3	dest port unreachable
3	6	dest network unknown
3	7	dest host unknown
4	0	source quench (congestion control - not used)
8	0	echo request (ping)
9	0	route advertisement
10	0	router discovery
11	0	TTL expired
12	0	bad IP header



Traceroute

- Source sends series of UDP segments to dest.
  - use unused port number
  - First has TTL =1
  - Second has TTL=2, etc.
- When n-th datagram arrives to n-th router:
  - Router discards datagram
  - And sends to source an ICMP message (type 11, code 0)
  - Message includes name of router& IP address
- When ICMP message arrives, source calculates RTT
- Traceroute does this 3 times

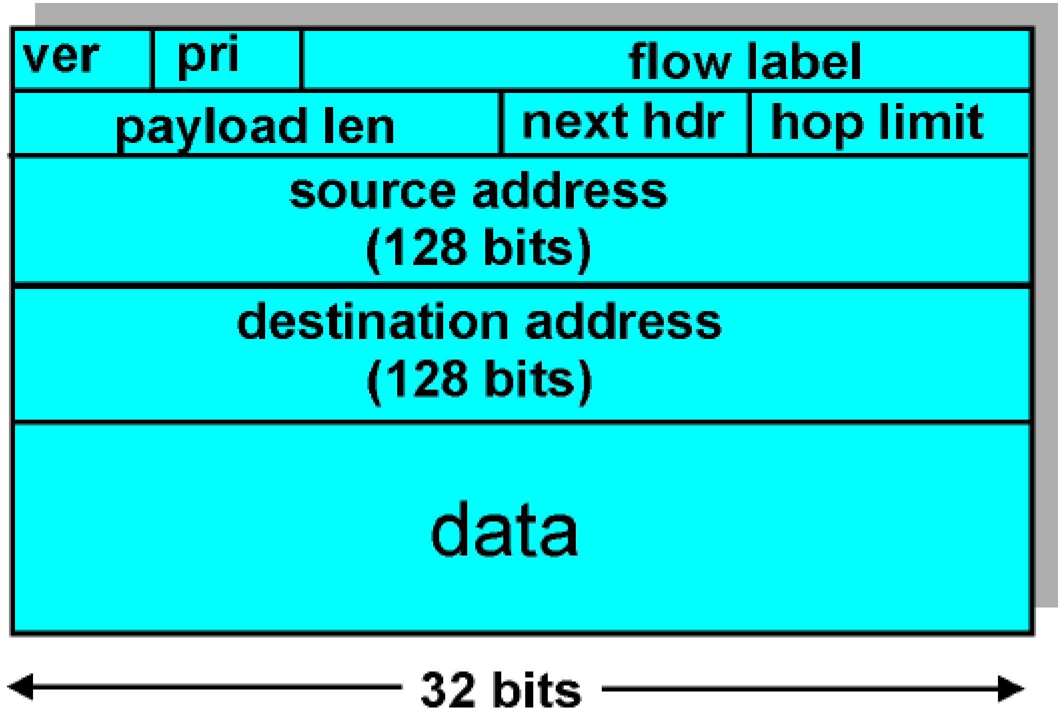
<u>Stopping criterion</u>

- UDP segment eventually arrives at destination host
- Destination returns ICMP "host unreachable" packet (type 3, code 3)
- When source gets this ICMP, stops

IPv6

- Initial motivation: 32-bit address space soon to be completely allocated.
- Additional motivation:
  - header format helps speed processing/forwarding
  - header changes to facilitate QoS
- IPv6 datagram format:
  - fixed-length 40 byte header
  - no fragmentation allowed

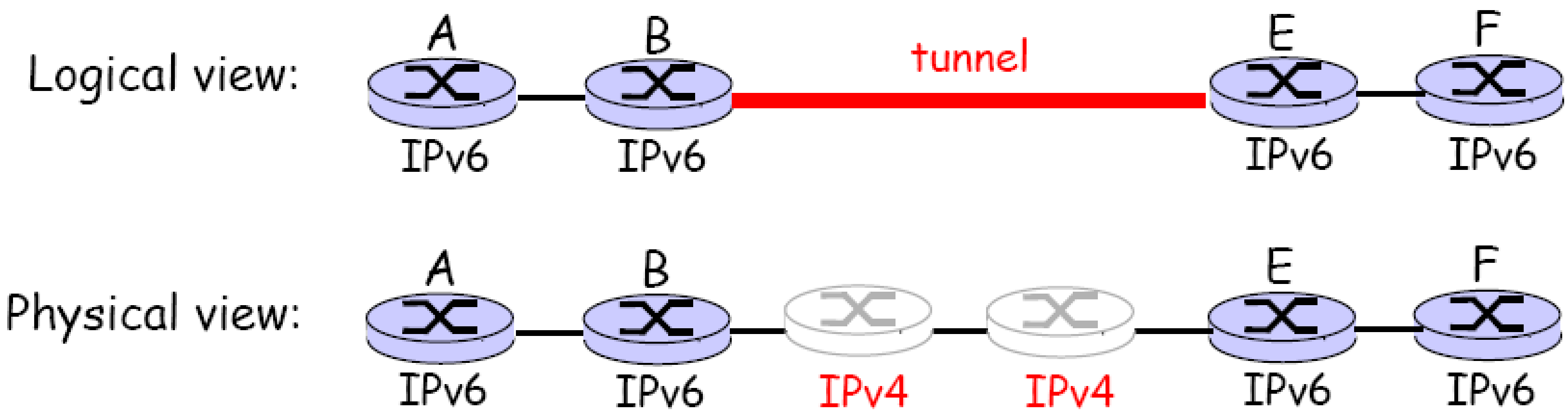
IPv6 Datagram Format



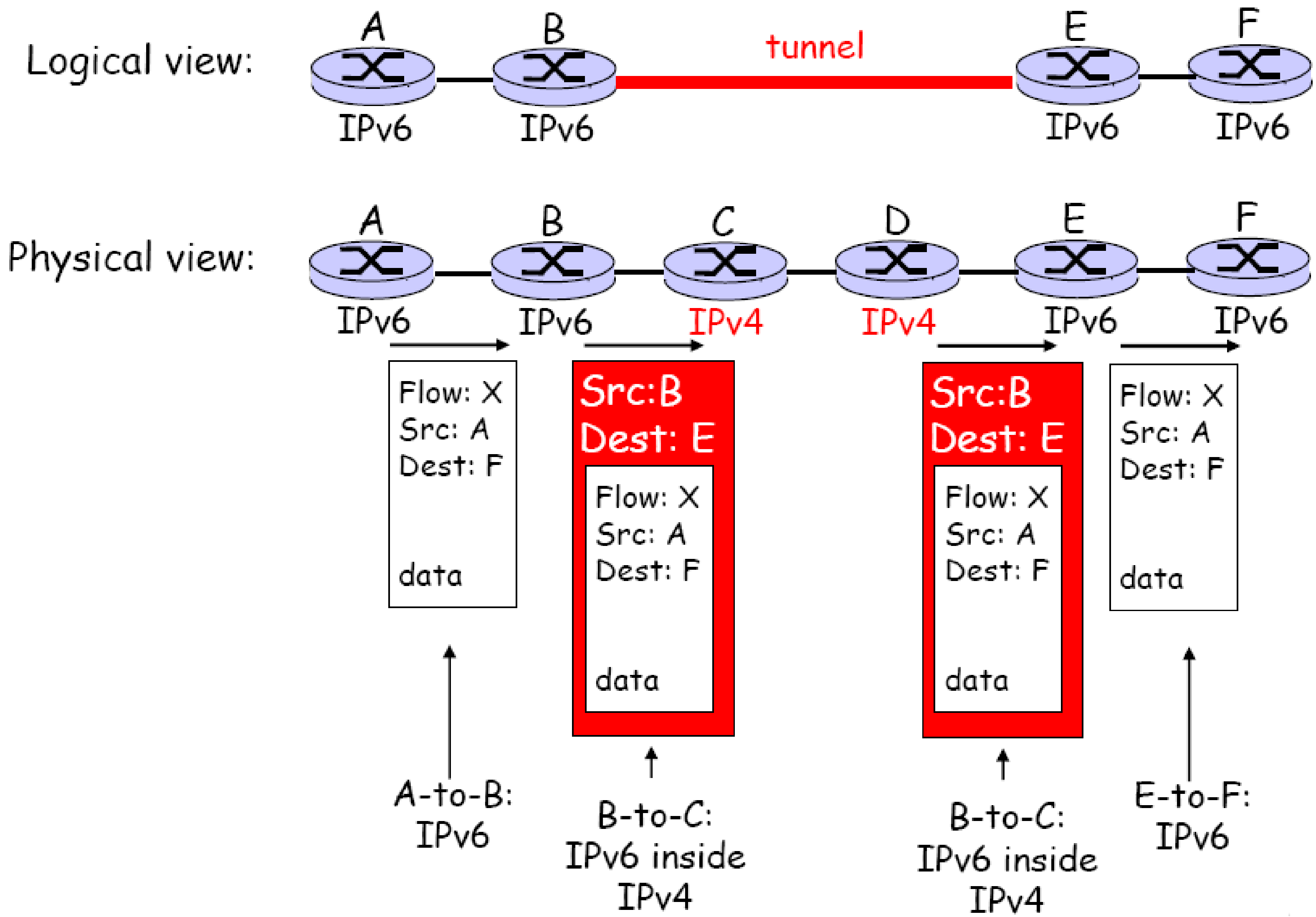
- *Priority:* identify priority among datagrams in flow
- *Flow Label:* identify datagrams in same "flow"
- *Next header:* identify upper layer protocol for data

Transition From IPv4 To IPv6

- Not all routers can be upgraded simultaneous
  - no “flag days”
  - How will the network operate with mixed IPv4 and IPv6 routers?
- *Tunneling:* IPv6 carried as payload in IPv4 datagram



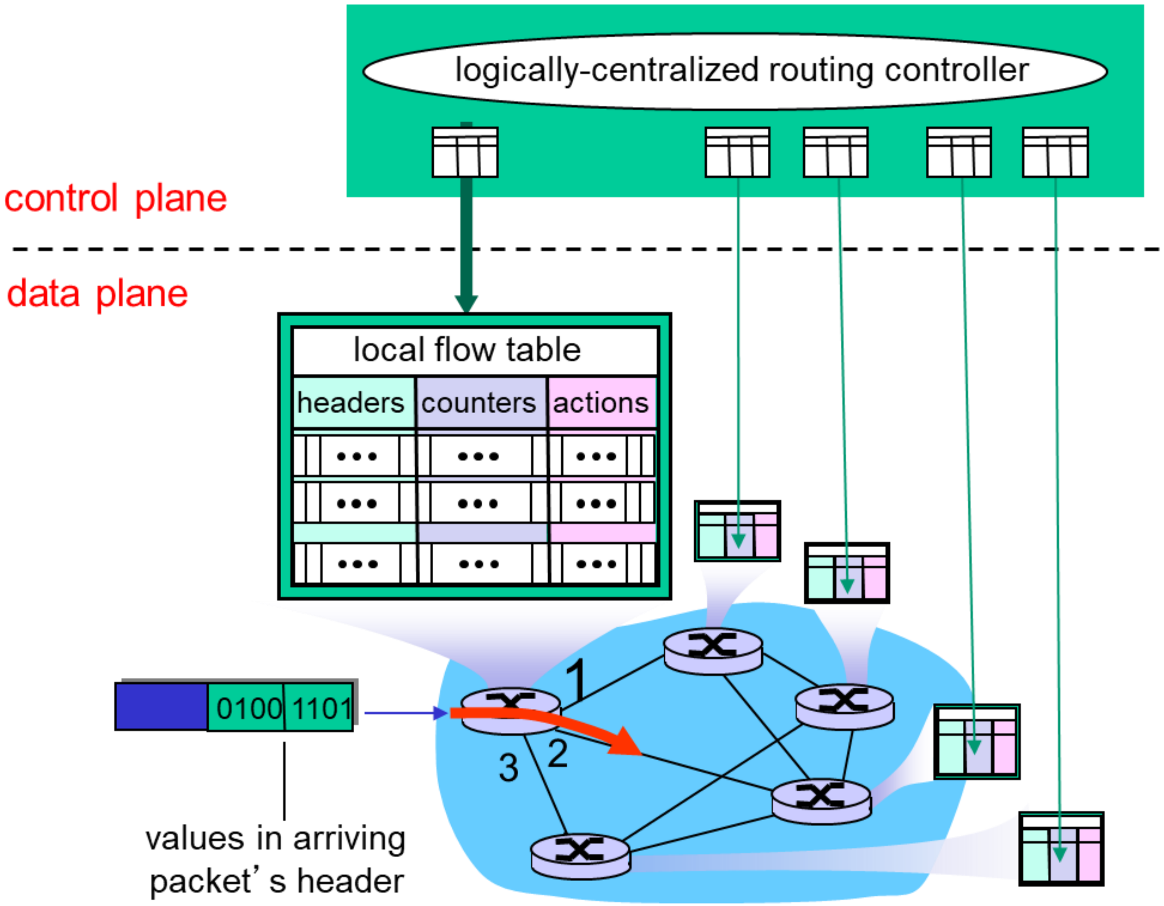
Tunneling



Logical view:
A
B
tunnel
E
F
IPv6
IPv6
IPv6
IPv6
Physical view:
A
B
C
D
E
F
IPv6
IPv6
IPv4
IPv4
IPv6
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
Flow: X
Src: A
Dest: F
data
A-to-B:
IPv6
B-to-C:
IPv6 inside
IPv4
B-to-C:
IPv6 inside
IPv4
E-to-F:
IPv6

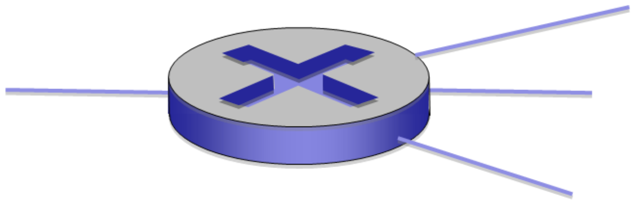
Generalized Forwarding and SDN

- Each router contains a *flow table* that is computed and distributed by a *logically centralized* routing controller



OpenFlow data plane abstraction

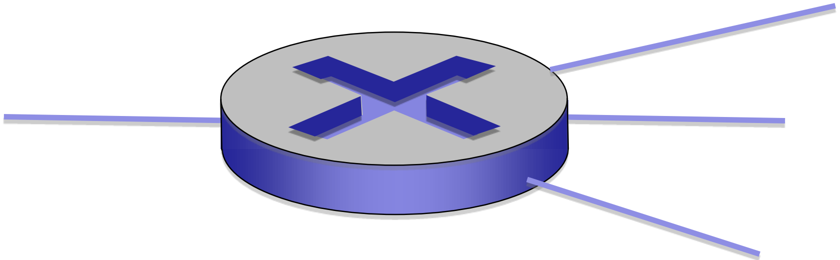
- *flow*: defined by values in header fields
- generalized forwarding: simple packet-handling rules
  - *Pattern:* match values in packet header fields
  - *Actions: for matched packet:* drop, forward, or modify matched packet, or send matched packet to controller
  - *Priority*: disambiguate overlapping patterns
  - *Counters:* #bytes and #packets since last match



flow table in a router (computed and distributed by controller) define router's match+action rules

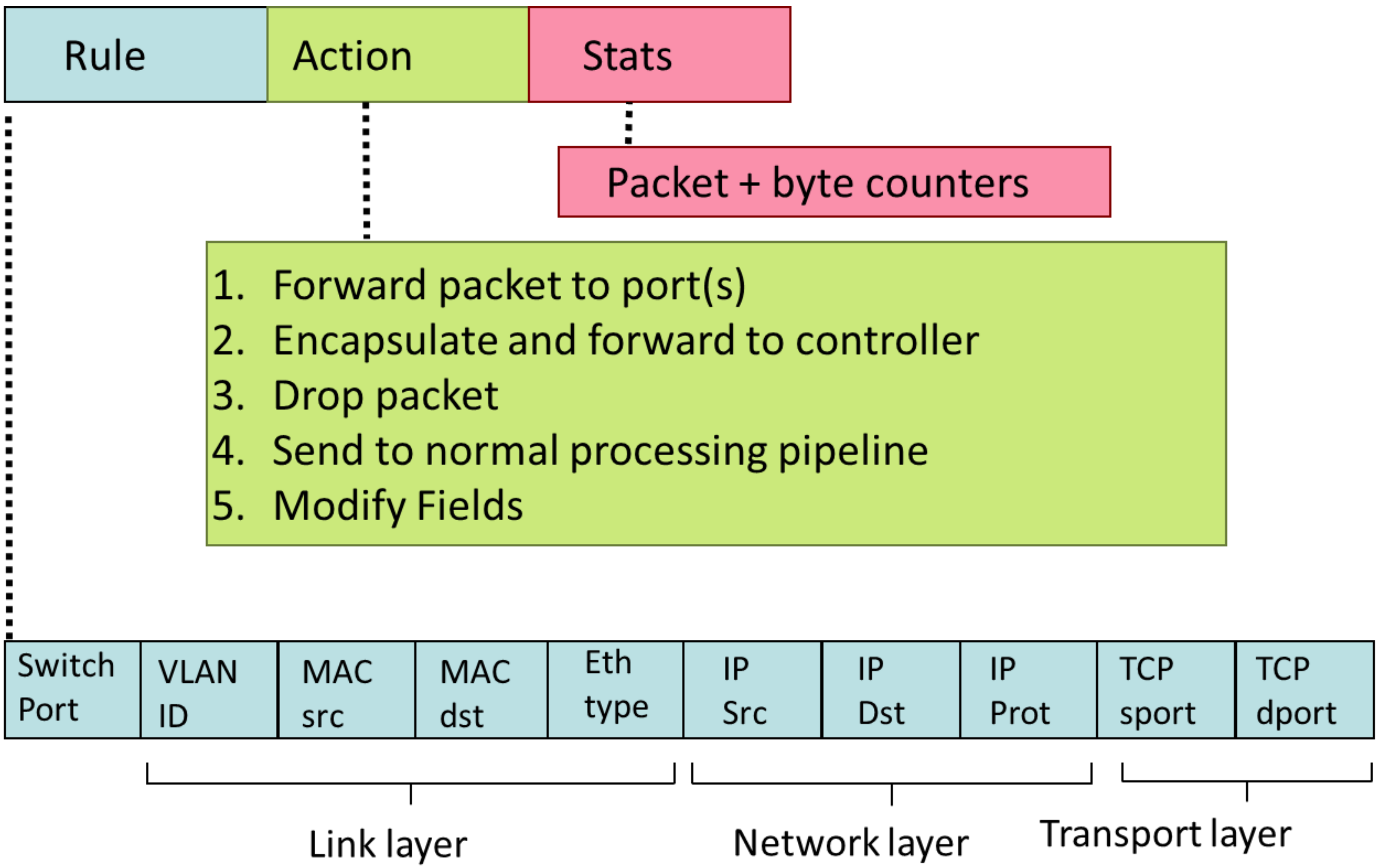
OpenFlow data plane abstraction

- flow table example



1. src=1.2.*.*, dest=3.4.5.* → drop
2. src = *.*.*.*, dest=3.4.*.* → forward to interface 2
3. src=10.1.2.3, dest=*.*.*.* → send to controller

OpenFlow: Flow Table Entries



OpenFlow: Flow Table Examples

Destination-based forwarding:

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
*	*	*	*	*	*	51.6.0.8	*	*	*	port6

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

Destination-based layer 2 (switch) forwarding:

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
*	22:A7:23: 11:E1:02	*	*	*	*	*	*	*	*	port3

layer 2 frames from MAC address 22:A7:23:11:E1:02 should be forwarded to output port 6

OpenFlow: Flow Table Examples

Firewall:

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Forward
*	*	*	*	*	*	*	*	*	22	drop

do not forward (block) all datagrams destined to TCP port 22

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Forward
*	*	*	*	*	128.119.1.1	*	*	*	*	drop

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

OpenFlow abstraction

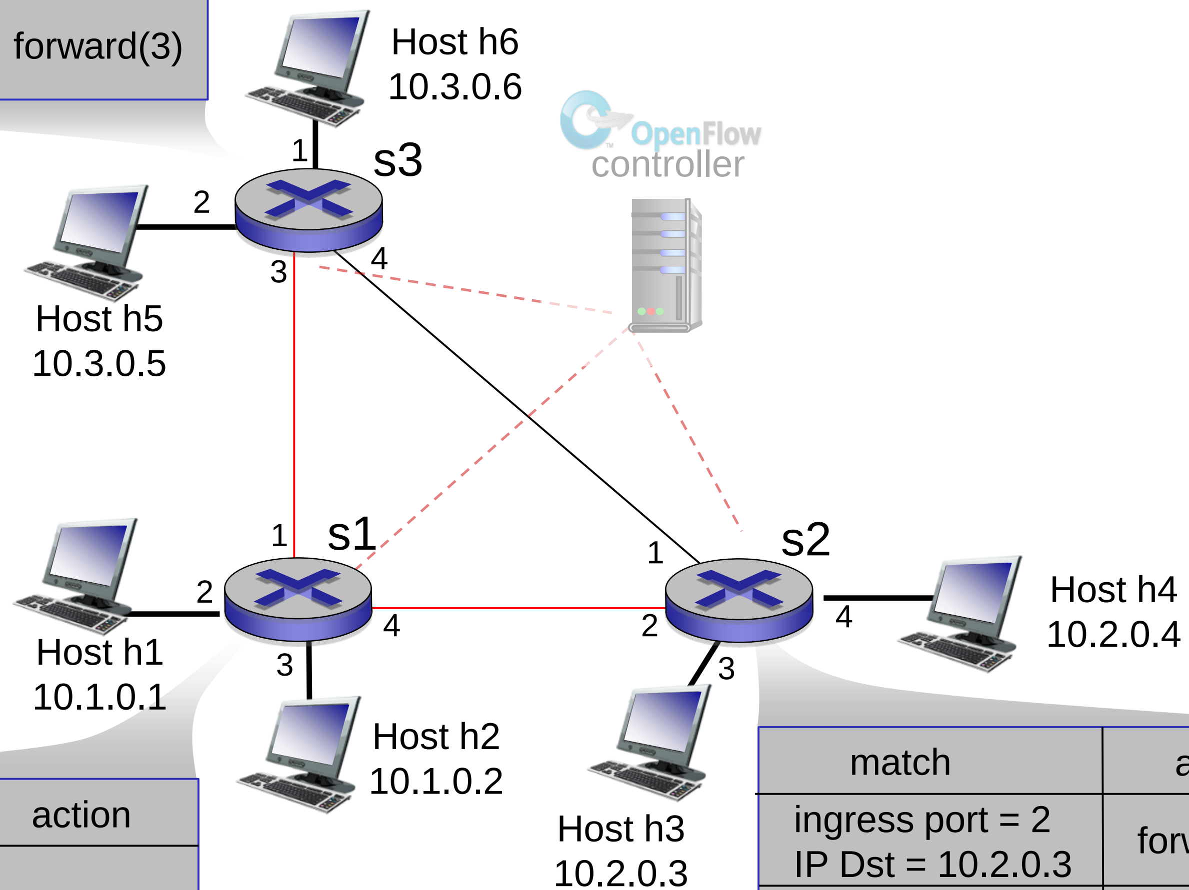
- *match+action:* unifies different kinds of devices

- 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

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

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)

End of NL – Data Plane