N/w Layer

IP std Header

local

global

Recap.

Routing

IPv4 addressing

Switching.

packet forward

Generation of architectures

- Software Switch
- Network Switch. Cross bar

IPv4. · 32 bit addr

$2^{32}$ addr ⇝ 4 billion addr

↳ more devices.

mobiles, IoT

↳ Internet of Things.

↳ → Sensor based embedded devices

Air pollution sensors → Server

{set of addr-} smaller than set of devices.

→ Devices not conn to Internet all the time-

eg. 100 devices.

50

at any given time - only 40 active.

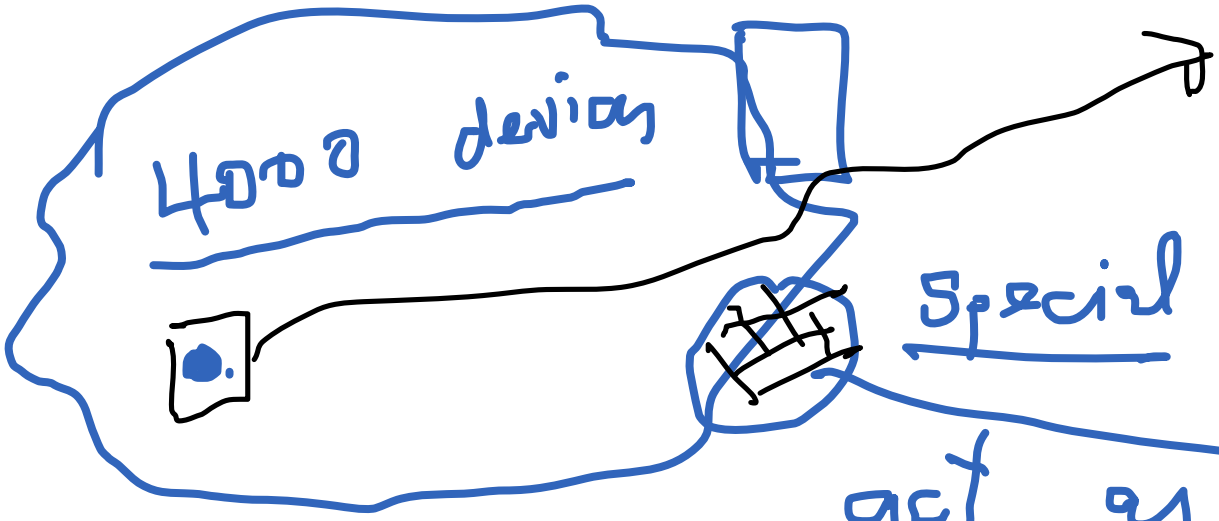
dynamic mapping

don't give permanent add to all 100 nodes

time multiplexing

DHCT

100 addresses

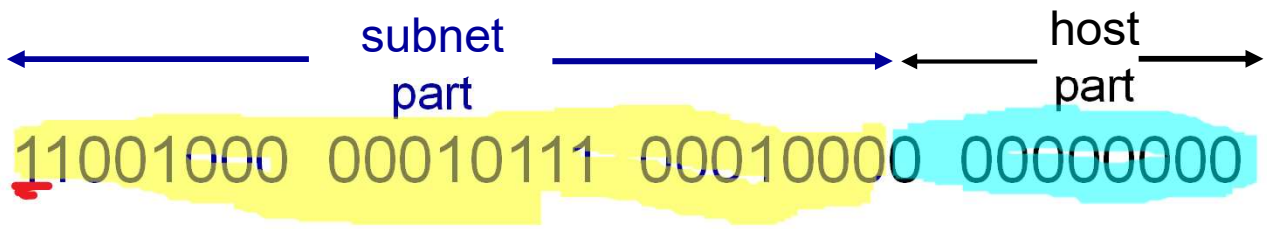



Spatial Multiplexing

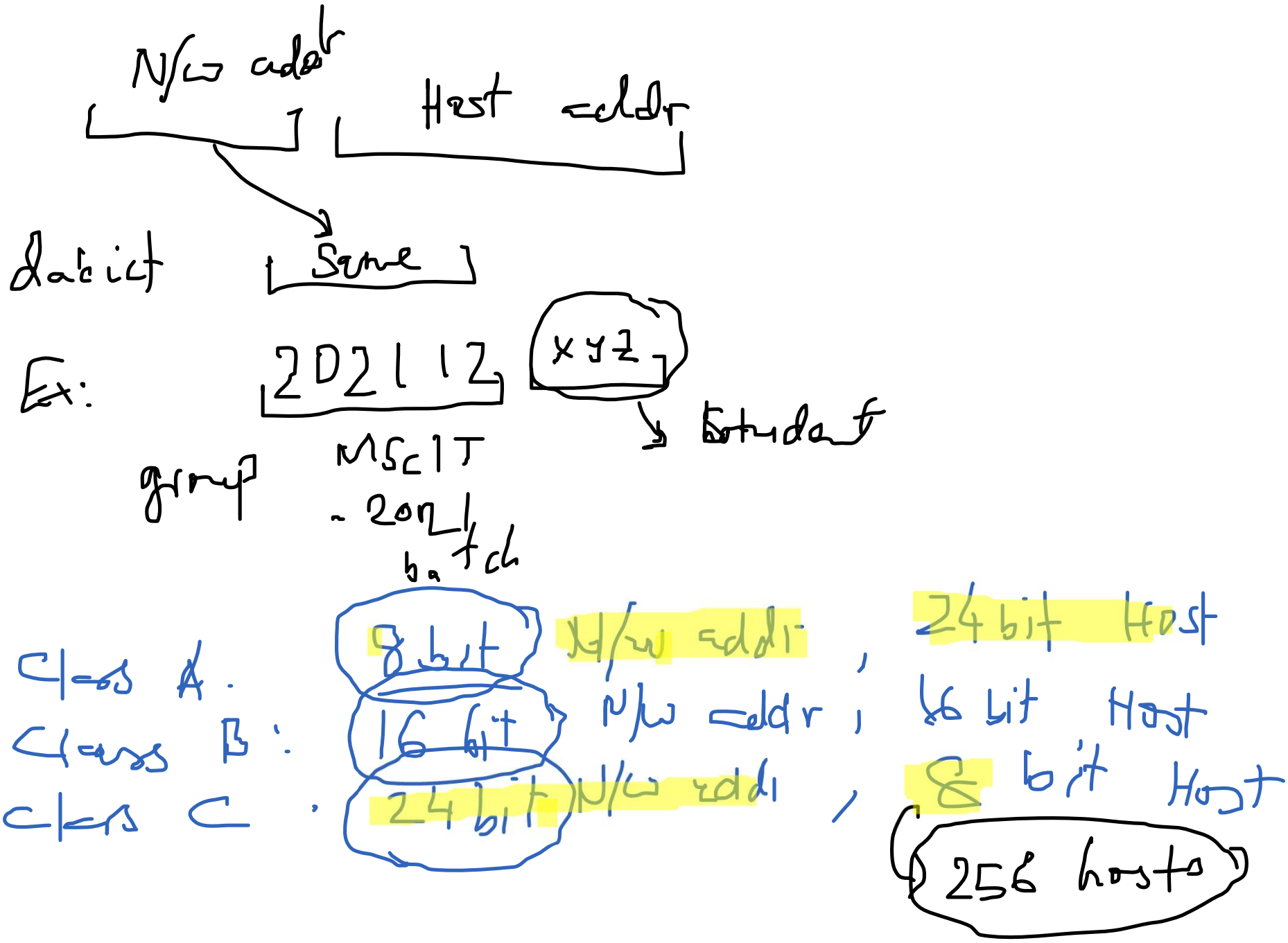
# IP addressing: CIDR

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




200.23.16.0/23


N/w addr
Host addr
Same
2021 12
xyz
MSCIT
- 2021
batch
8 bit N/w addr, 24 bit Host
16 bit N/w addr, 16 bit Host
24 bit N/w addr, 8 bit Host
256 hosts

# IP addresses: how to get one?

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

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

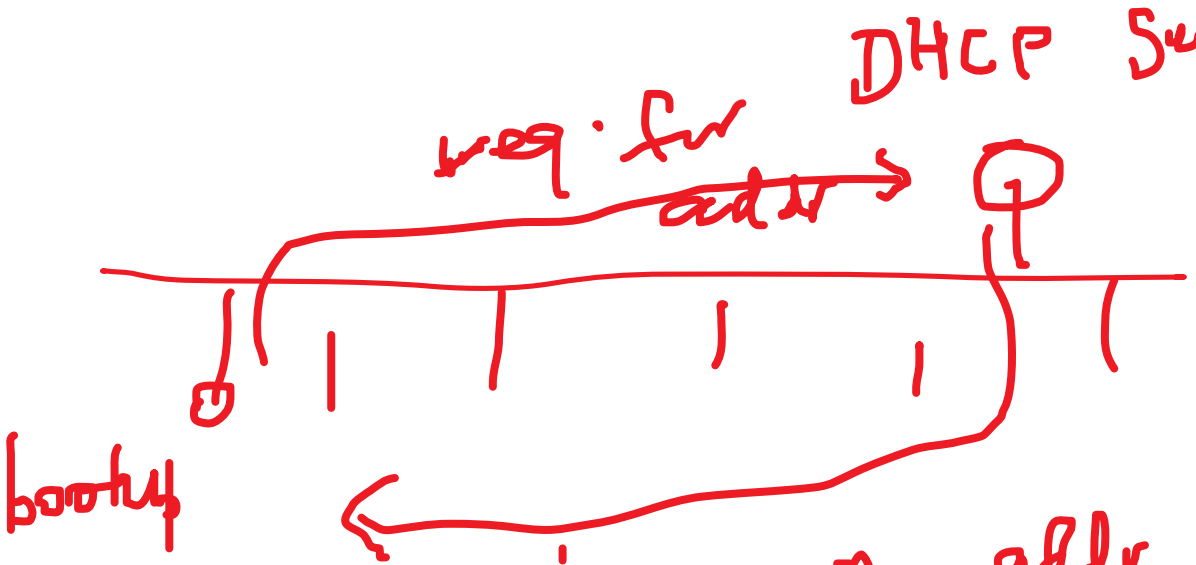
# DHCP: Dynamic Host Configuration Protocol

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

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

*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



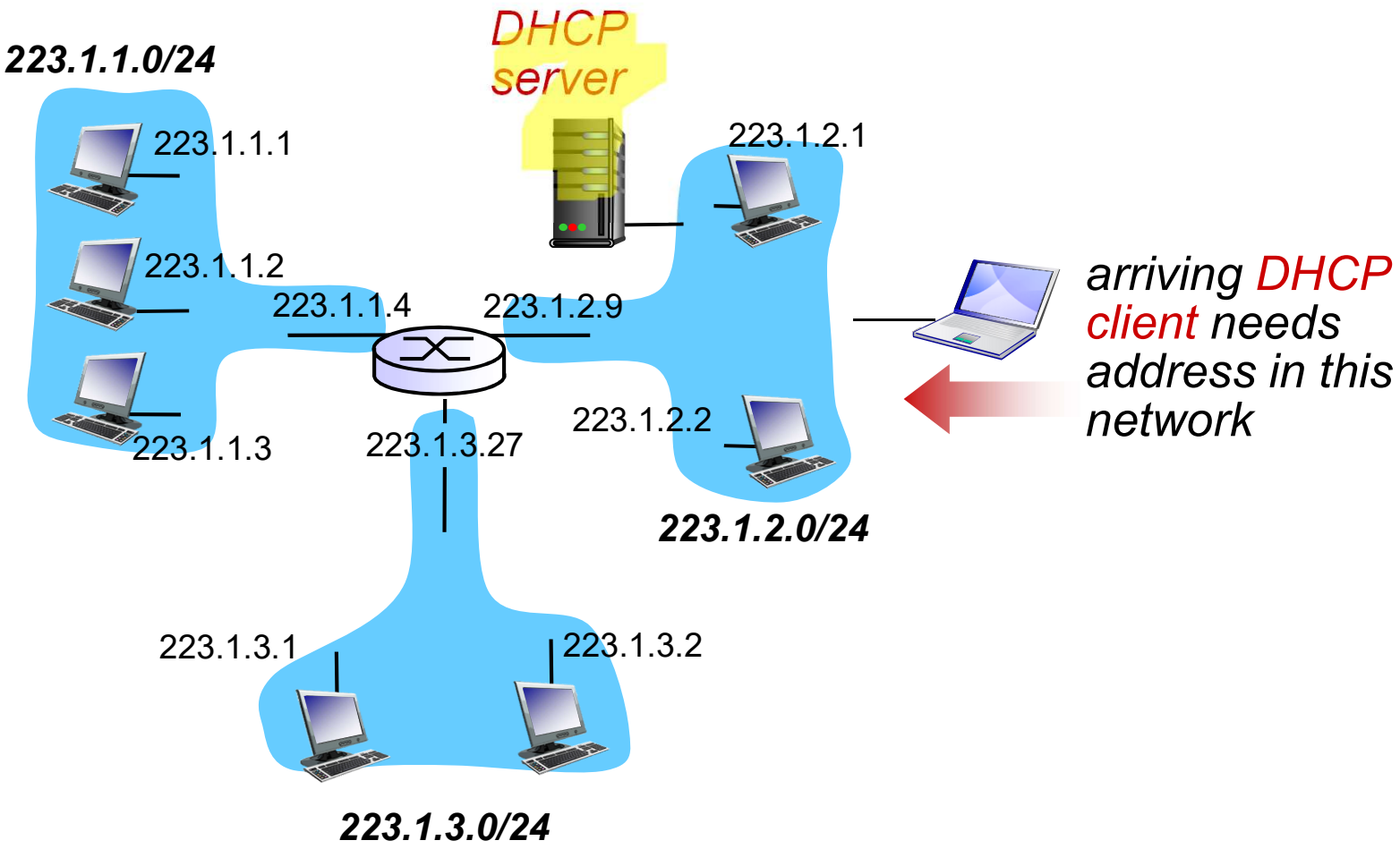

assign an addr (dynamic)

Maintains a list of used addr.

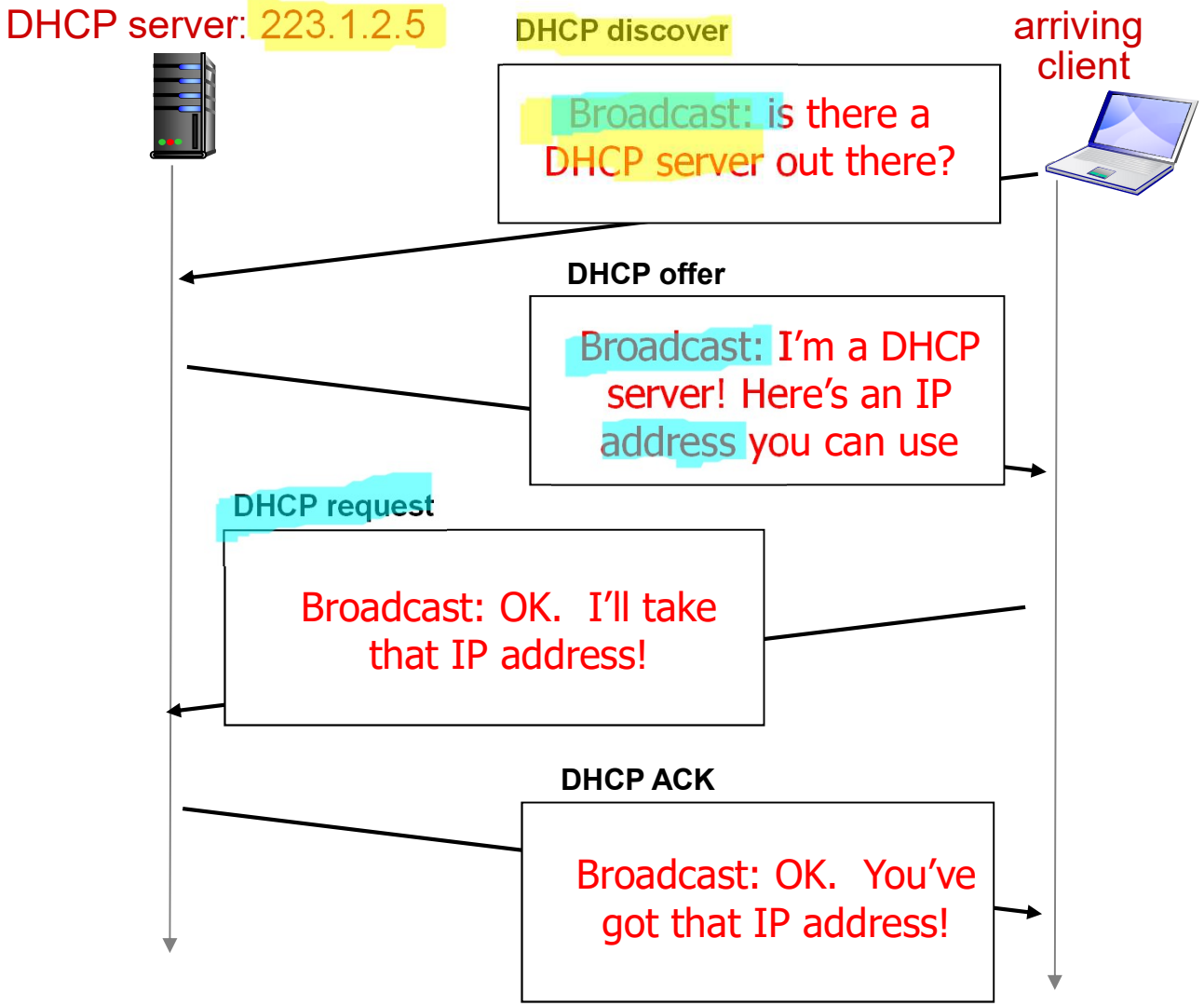
assign → expiration time.

client. renew the assignment

# DHCP client-server scenario



# DHCP client-server scenario

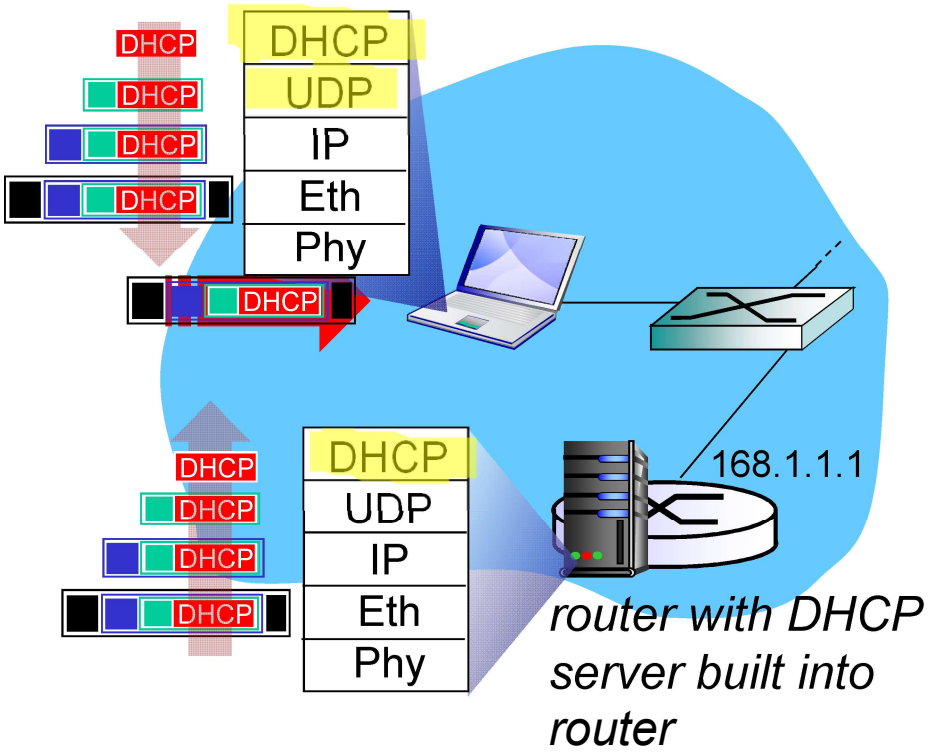


# DHCP: more than IP addresses

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

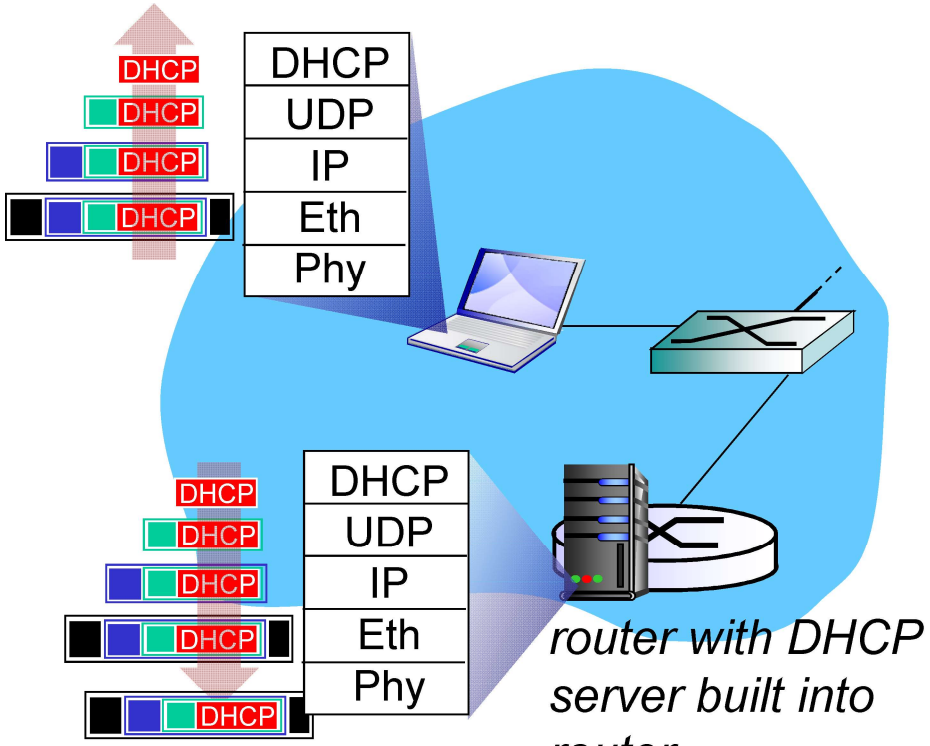
- address of first-hop router for client
- name and IP address of DNS sever
- network mask (indicating network versus host portion of address)

# DHCP: example




- connecting laptop needs its IP address, addr of first-hop router, addr of DNS server: use DHCP
- DHCP request encapsulated in UDP, encapsulated in IP, encapsulated in 802.1 Ethernet
- Ethernet frame broadcast (dest: FFFFFFFFFFFF) on LAN, received at router running DHCP server
- Ethernet demuxed to IP demuxed, UDP demuxed to DHCP

# DHCP: example




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

# IP addresses: how to get one?

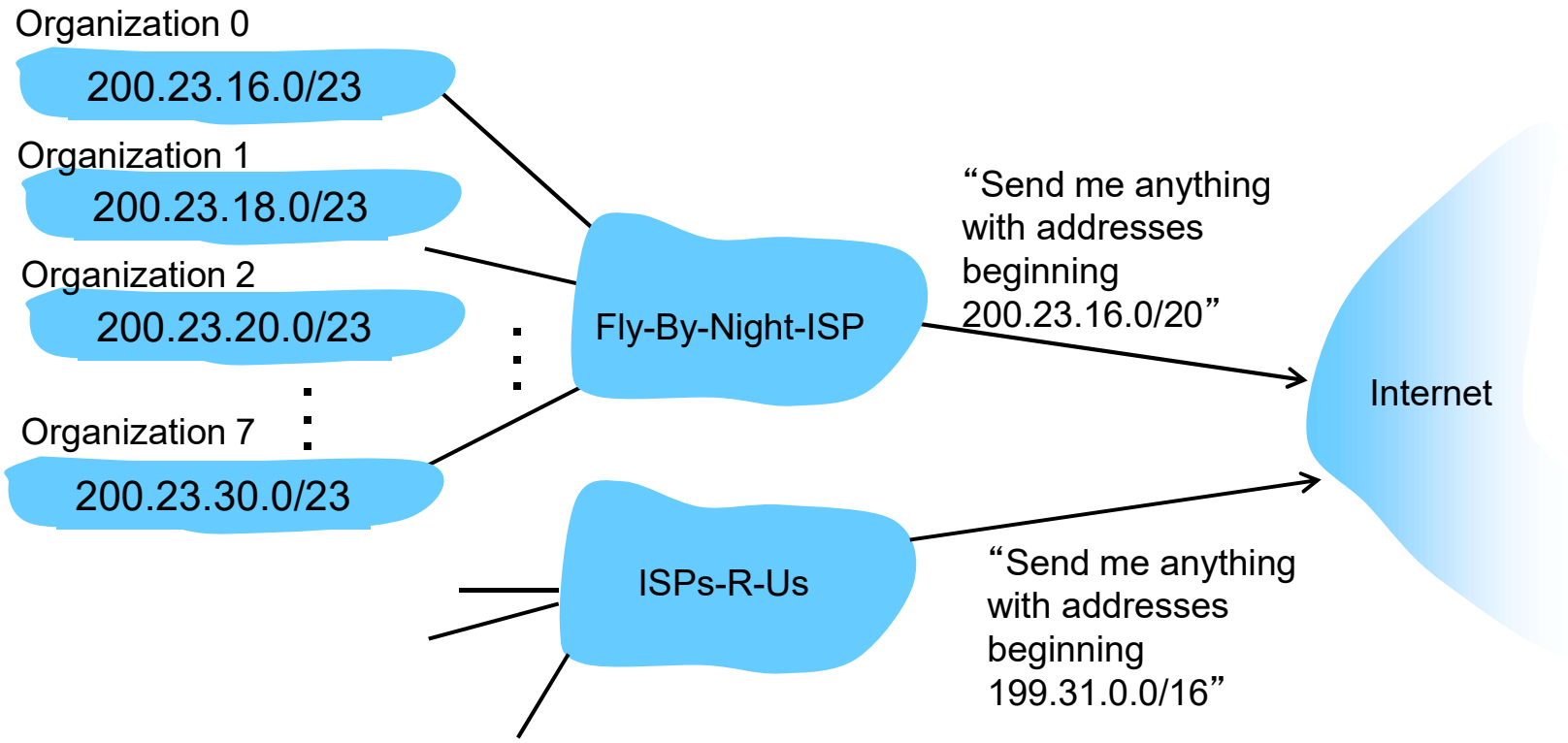
*Q:* how does *network* get subnet part of IP addr?

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

| | | | |
|---|---|---|---|
| ISP's block | 11001000 00010111 00010000 | 00000000 | 200.23.16.0/20 |
| Organization 0 | 11001000 00010111 00010000 | 00000000 | 200.23.16.0/23 |
| Organization 1 | 11001000 00010111 00010010 | 00000000 | 200.23.18.0/23 |
| Organization 2 | 11001000 00010111 00010100 | 00000000 | 200.23.20.0/23 |
| ... | ..... | .... | .... |
| Organization 7 | 11001000 00010111 00011110 | 00000000 | 200.23.30.0/23 |

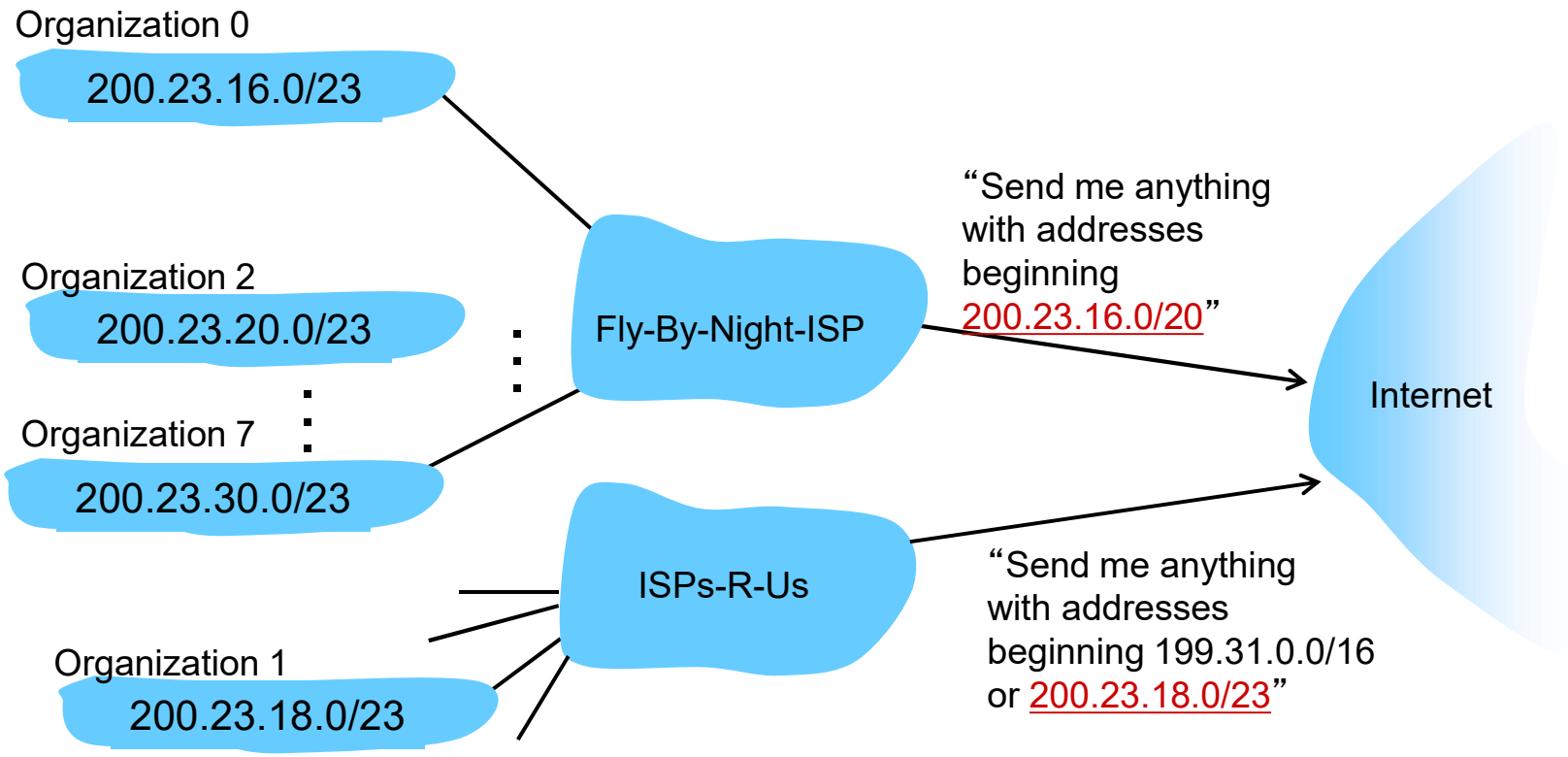
# Hierarchical addressing: route aggregation

hierarchical addressing allows efficient advertisement of routing information:



# Hierarchical addressing: more specific routes

ISPs-R-Us has a more specific route to Organization 1



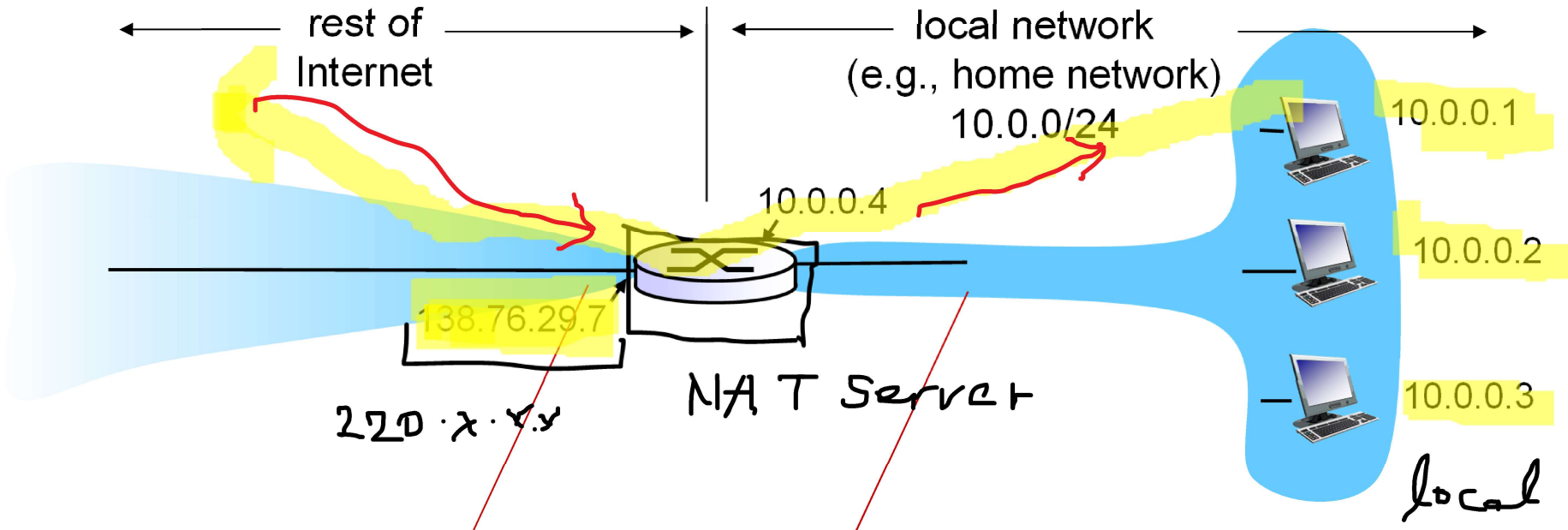
# IP addressing: the last word...

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

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

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

# NAT: network address translation




*all* datagrams *leaving* local network have *same* single source NAT IP address: 138.76.29.7,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

*motivation:* local network uses just one IP address as far as outside world is concerned:

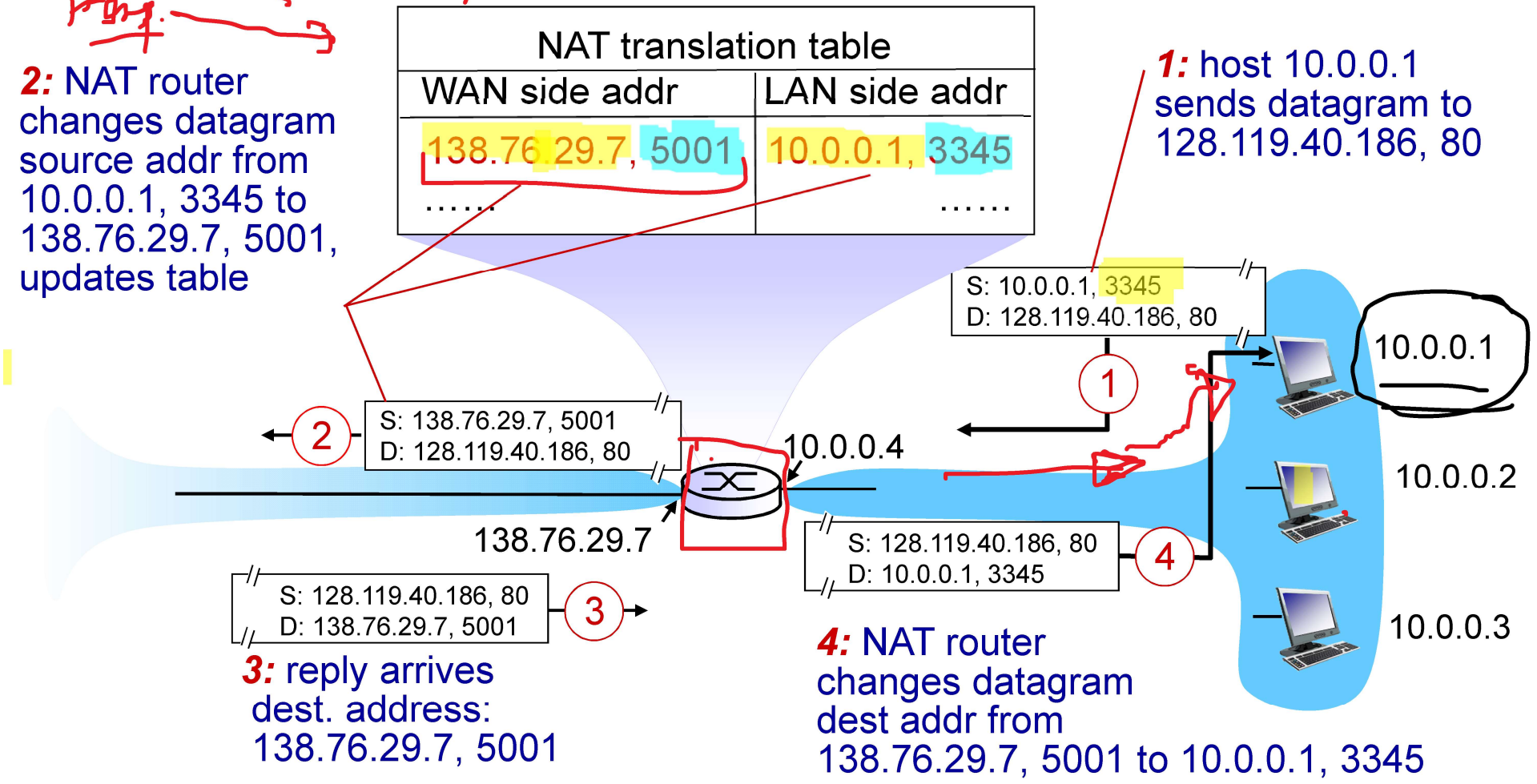
- range of addresses not needed from ISP: just one IP address for all devices
- can change addresses of devices in local network without notifying outside world
- can change ISP without changing addresses of devices in local network
- devices inside local net not explicitly addressable, visible by outside world (a security plus)

# NAT: network address translation

*implementation:* NAT router must:

- *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 addr
- *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 dest fields of every incoming datagram with corresponding (source IP address, port #)

# NAT: network address translation




* Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu/kurose_ross/interactive/

# NAT: network address translation

- 16-bit port-number field:
  - 60,000 simultaneous connections with a single LAN-side address!
- NAT is controversial:
  - routers should only process up to layer 3
  - address shortage should be solved by IPv6
  - violates end-to-end argument
    - NAT possibility must be taken into account by app designers, e.g., P2P applications
  - NAT traversal: what if client wants to connect to server behind NAT?

# IPv6: motivation

- *initial motivation:* 32-bit address space soon to be completely allocated.
- additional motivation:
  - header format helps speed processing/forwarding
  - header changes to facilitate QoS

Quality of Service.
→ guarantees
delay, bandwidth rate

*IPv6 datagram format:*

- fixed-length 40 byte header
- no fragmentation allowed

→ TCP/IP:
- Store & forward
- datagram service — Best effort
- + Reliability (TCP)

Appl: Conventional Appl. (Non-Real-time)
email, remote login, FTP, ...

→ video streaming, VoIP, video conf.
↳ Real-time Appl → QoS.

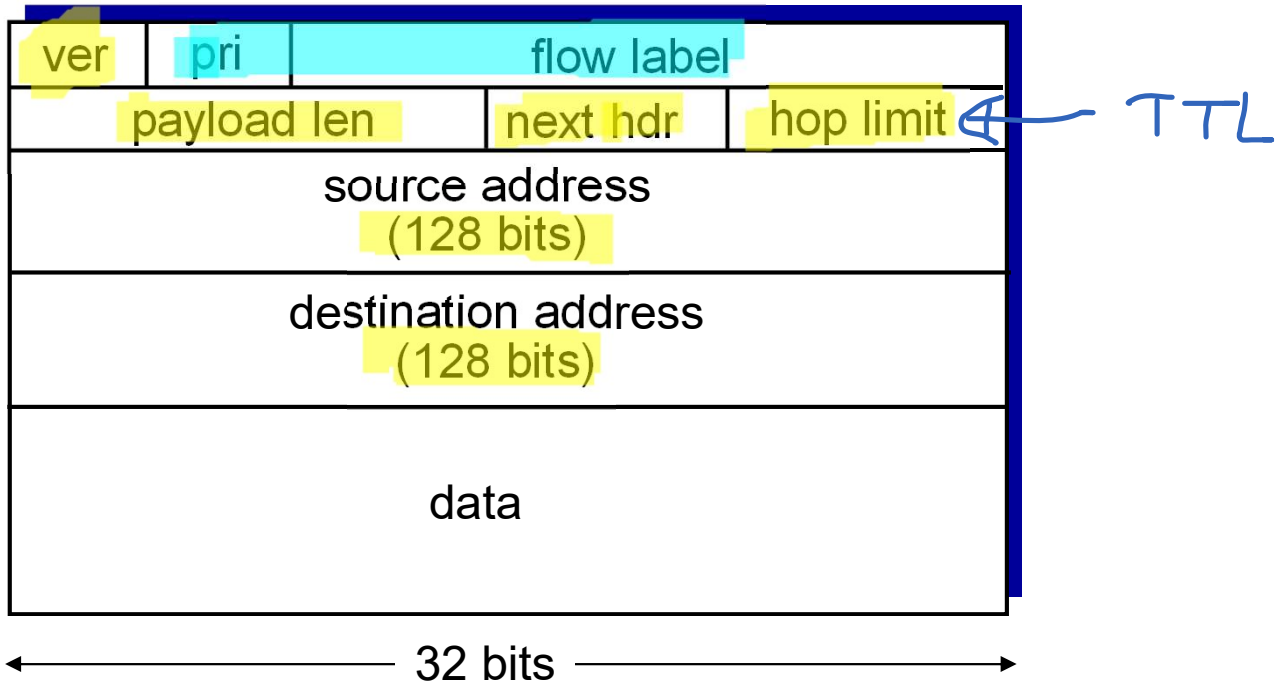
# IPv6 datagram format

$2^{128} \approx 10^{39}$

*priority:* identify priority among datagrams in flow

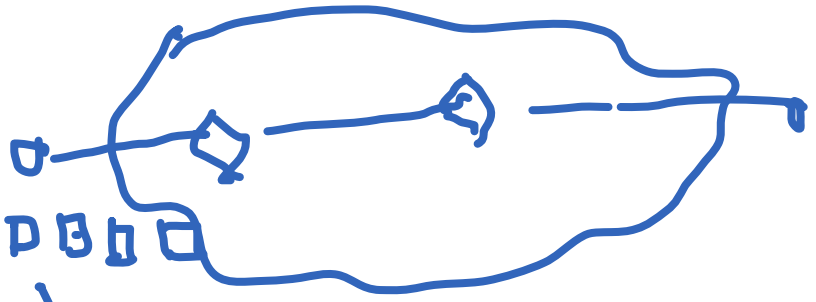
*flow Label:* identify datagrams in same "flow."
(concept of"flow" not well defined).

*next header:* identify upper layer protocol for data



QoS

flow ID.



↓ inter packet delay < $\Delta T$

video : 30 f/s :

$\Delta T \leq \frac{1}{30}$ s

IPv4. Best effort datagram.

All packets of a flow (socket) have same "flow id."

Router can recognize pkts of same flow and it can control delay/forwarding

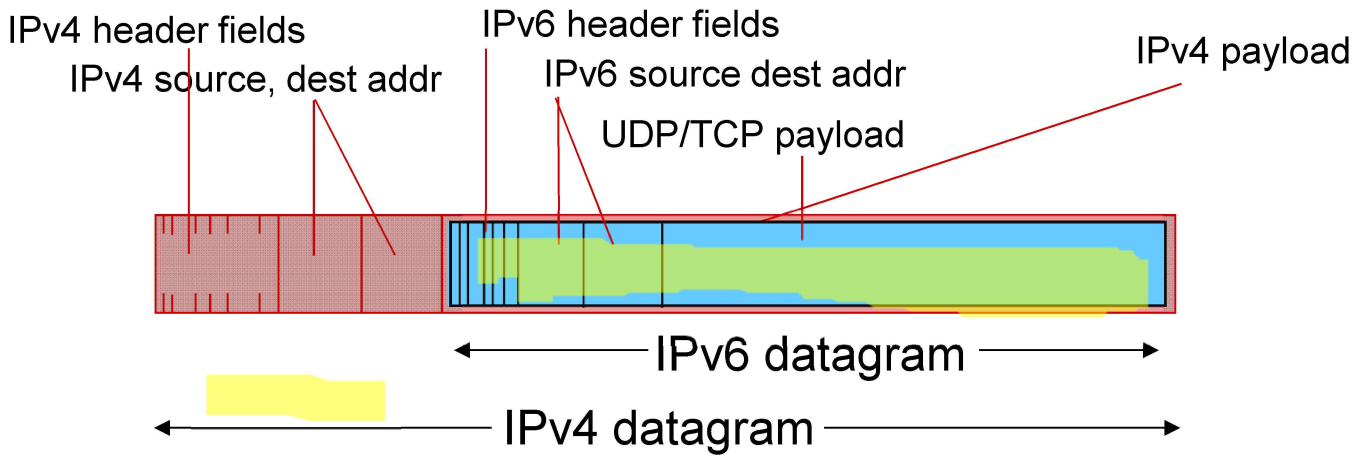
# Other changes from IPv4

- *checksum:* removed entirely to reduce processing time at each hop
- *options:* allowed, but outside of header, indicated by "Next Header" field
- *ICMPv6:* new version of ICMP
  - additional message types, e.g. "Packet Too Big"
  - multicast group management functions

fragmentation removed

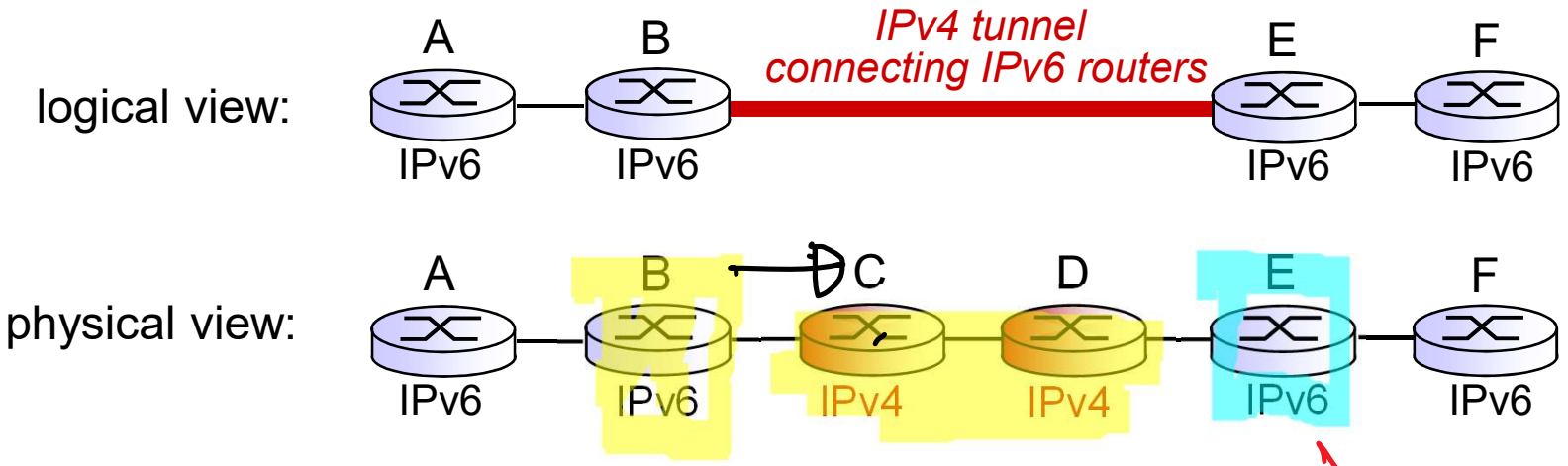
# Transition from IPv4 to IPv6

- 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

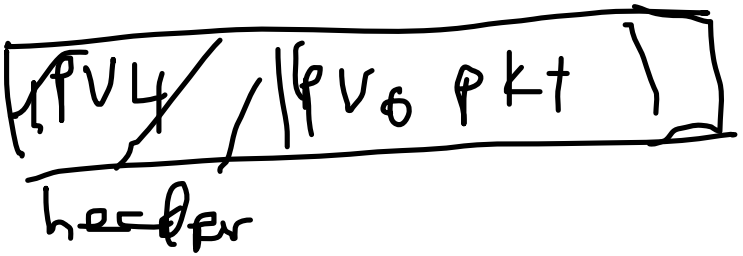


# Tunneling

Tunneling.

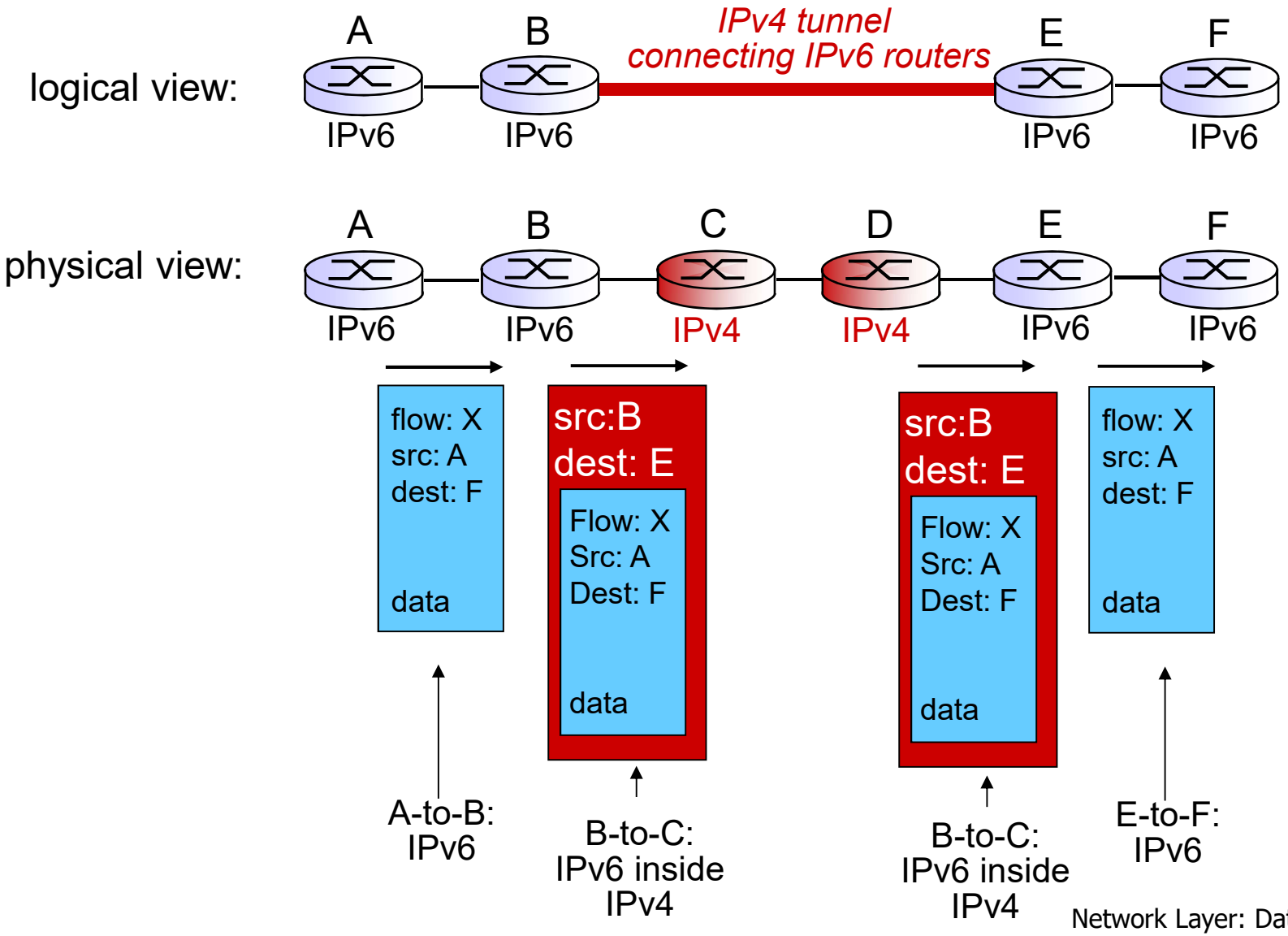



IPv6 packet →




extract IPv6 payload
↳ forwards

# Tunneling


logical view:
A
B
IPv4 tunnel
connecting IPv6 routers
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