Networking

based on slides by Prof. Sirer, Bracy, Van Renesse, Ross, Kurose

Basic Network Abstraction

- A process can create “endpoints”
- Each endpoint has a unique address
- Processes can receive messages on endpoints
- Processes can send messages to endpoints
- A message is a byte array

Some issues…

- How are addresses assigned?
- How does a message to some address find its way to the corresponding endpoint?
- Can one broadcast messages?
  - Can multiple endpoints share the same address?
- Can messages
  - be arbitrarily large?
  - be lost or garbled?
  - be re-ordered?
- What do processes "stick" in these messages?

Network “protocol”

- An agreement between processes about the content of messages
  - Syntax: Layout of bits, bytes, fields, etc.
    - message format
  - Semantics: What they mean
- Examples:
  - HTTP “get” requests and responses
    - HTML is part of the format
  - Excuse me, please, thank you, etc. in real life

Network Layering

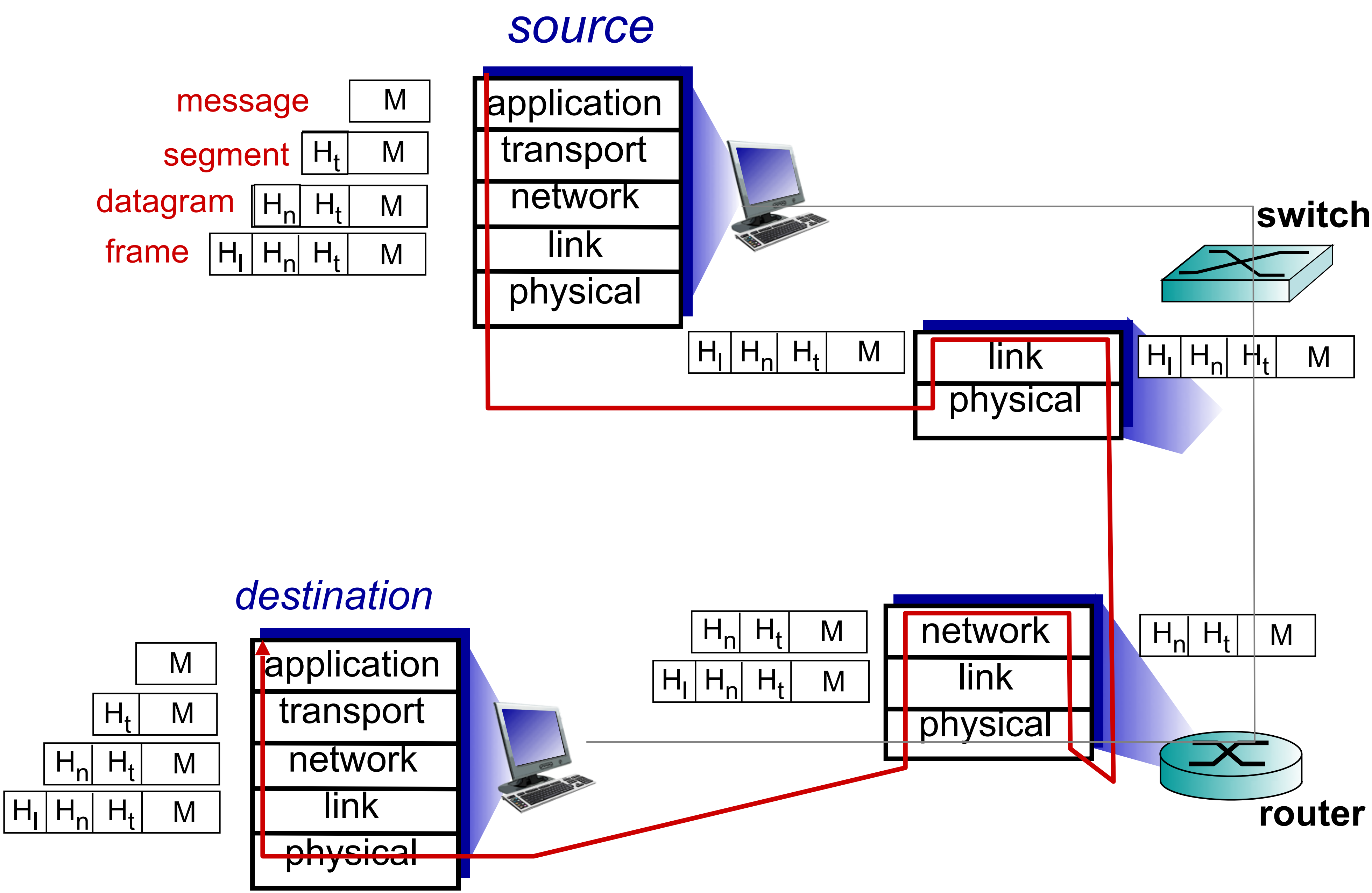
- The network abstraction is usually layered
  - Each layer provides a service to layers above; relies on services from layers below
- Example:

Application Layer	HTTP/FTP/DNS; exchanges messages
Transport Layer	Transports messages; TCP (connection oriented)/ UDP; exchanges segments
Network Layer	Transports segments; IP; exchanges datagrams
Link Layer	Transports datagrams; Ethernet/WiFi; exchanges frames
Physical Layer	Trasports frames;wires, signal encoding, wireless; exchanges bits

Why Layering?

- Modularity
  - Allows to identify relationship between distinct pieces of complex system
  - Eases maintenance and updating of system
    - change of implementation of layer's service transparent to rest of system
- Are there costs to modularity?

Encapsulation

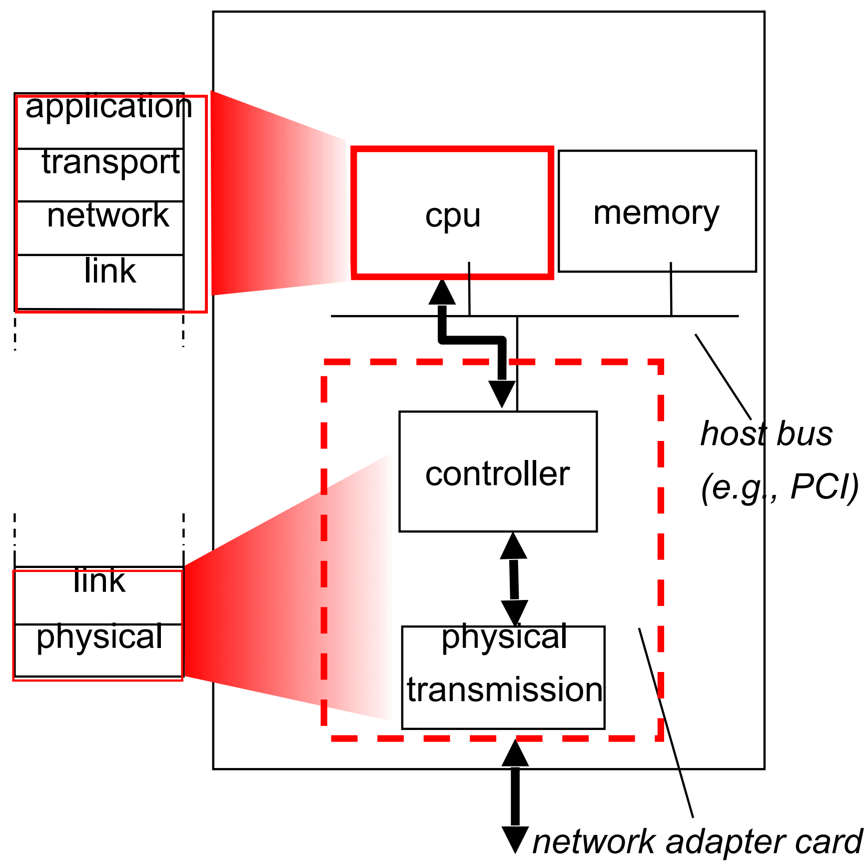


Application Layer
Transport Layer
Network Layer
Link Layer
Physical Layer

Link Layer:
Local Area Networking (LAN) and Ethernet

Where is the Link Layer implemented?

- In each host, through one or more *NICs*
  - *Network Interface Cards*
    - Ethernet, 802.11, etc.
- Attaches into host's system buses
- *Combination of hardware, software, firmware*



Addressing

- Each NIC has a *MAC address*
  - *Media Access Control* address
  - Unique!
  - 6 bytes long
  - Ethernet example: b8:e3:56:15:6a:72
  - Address space managed by IEEE; first 24 bits identify manufacturer
  - Does not change if the NIC moves
    - Not true of IP address!

Multiple access protocols

- single shared broadcast channel
- two or more simultaneous transmissions by nodes: interference
  - *collision* if node receives two or more signals at the same time

multiple access protocol

- distributed algorithm that determines how nodes share channel, i.e., determine when node can transmit
- communication about channel sharing must use channel itself!
  - no out-of-band channel for coordination

An ideal multiple access protocol

given: broadcast channel of rate R bps

desiderata:

1. node that wants to transmit, can send at rate R.
2. when M nodes want to transmit, each can send at average rate R/M
3. fully decentralized:
   - no special node to coordinate transmissions
   - no synchronization of clocks, slots
4. simple

MAC protocols: taxonomy

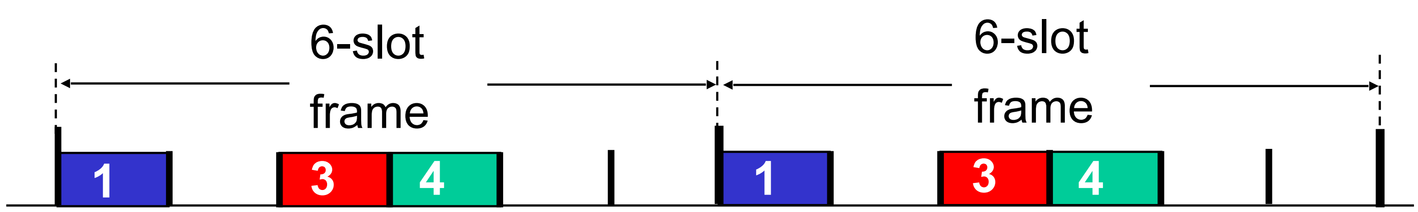
three broad classes:

- *channel partitioning*
  - divide channel into smaller "pieces" (time slots, frequency, code)
  - allocate piece to node for exclusive use
- *random access*
  - channel not divided, allow collisions
  - "recover" from collisions
- *"taking turns"*
  - nodes take turns, but nodes with more to send can take longer turns

Channel partitioning MAC protocols: TDMA

TDMA: time division multiple access

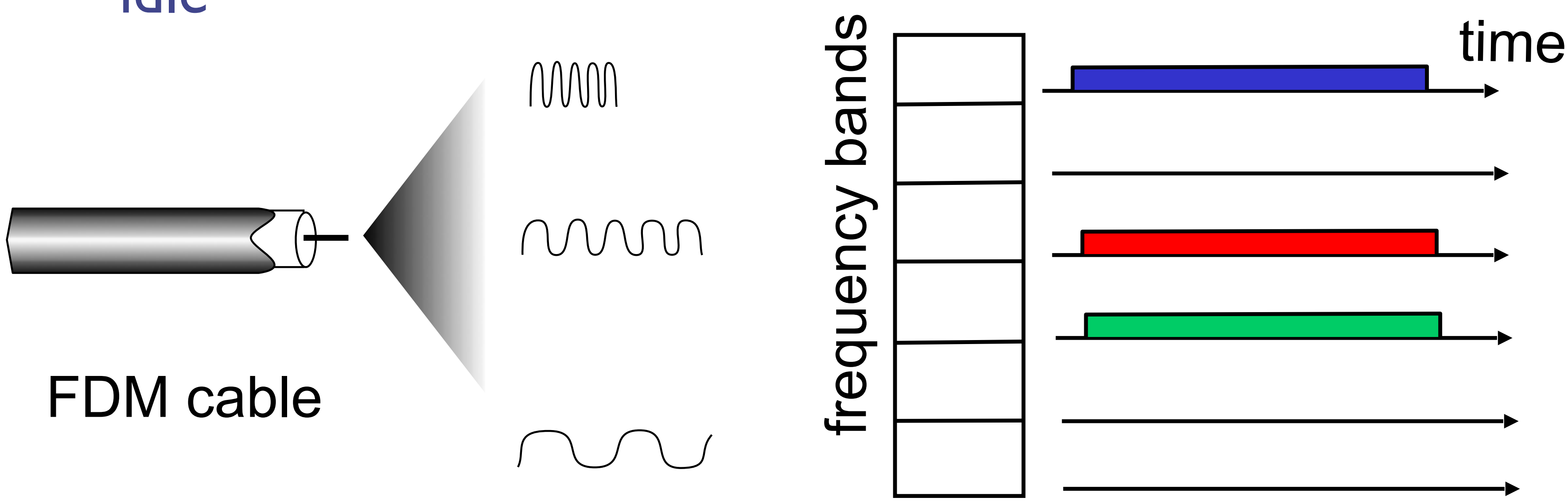
- access to channel in "rounds"
- each station gets fixed length slot (length = pkt trans time) in each round
- unused slots go idle
- example: 6-station LAN, 1,3,4 have frames, slots 2,5,6 idle



Channel partitioning MAC protocols: FDMA

FDMA: frequency division multiple access

- channel spectrum divided into frequency bands
- each station assigned fixed frequency band
- unused transmission time in frequency bands go idle
- example: 6-station LAN, 1,3,4 have pkt, frequency bands 2,5,6 idle



"Taking turns" MAC protocols

channel partitioning MAC protocols:

- share channel *efficiently* and *fairly* at high load
- inefficient at low load: delay in channel access, 1/N bandwidth allocated even if only 1 active node!

random access MAC protocols

- low load: single node can fully utilize channel
- high load: collision overhead

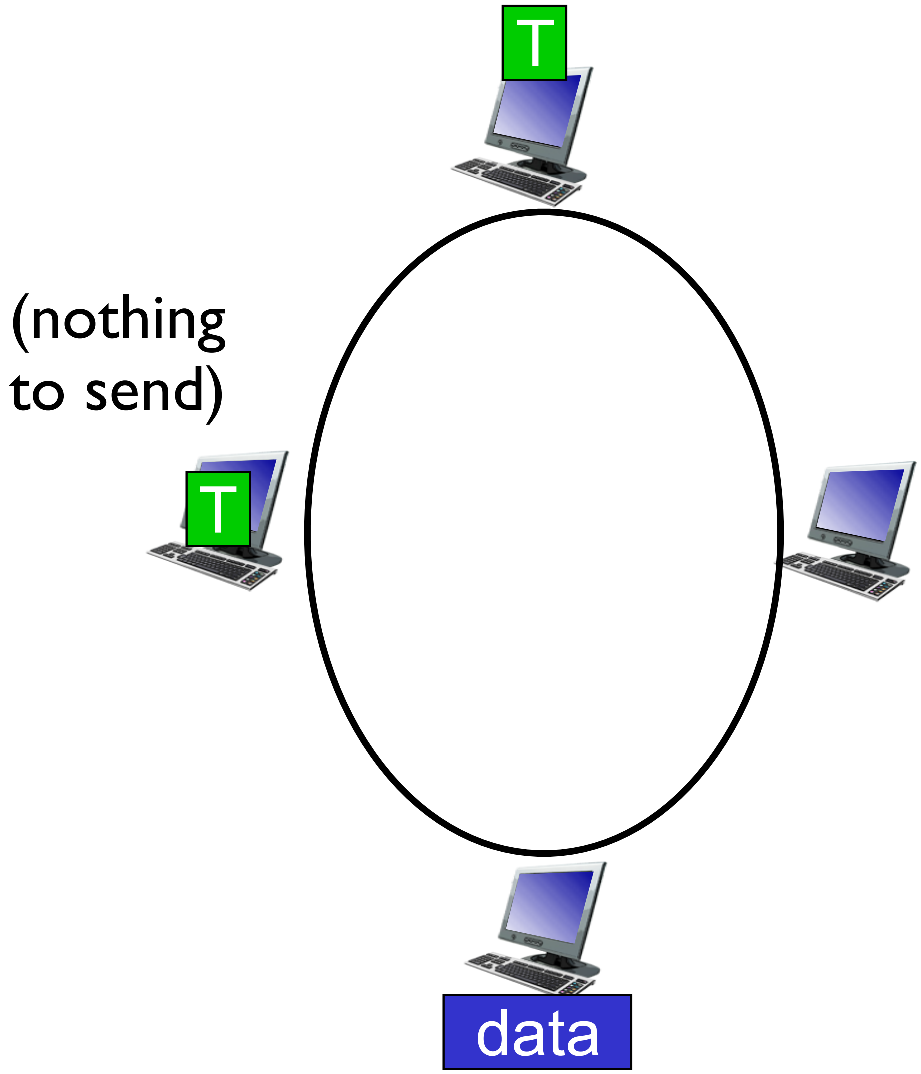
"taking turns" protocols

look for best of both worlds!

"Taking turns" MAC protocols

token passing:

- control token passed from one node to next sequentially.
- token message
- concerns:
  - token overhead
  - latency
  - single point of failure (token)



Random access protocols

- when node has packet to send
  - transmit at full channel data rate R.
  - no *a priori* coordination among nodes
- multiple transmitting nodes ➜ "collision",
- random access MAC protocol specifies:
  - how to detect collisions
  - how to recover from collisions (e.g., via delayed retransmissions)
- examples of random access MAC protocols:
  - slotted ALOHA, ALOHA
  - CSMA, CSMA/CD, CSMA/CA

Example: Ethernet

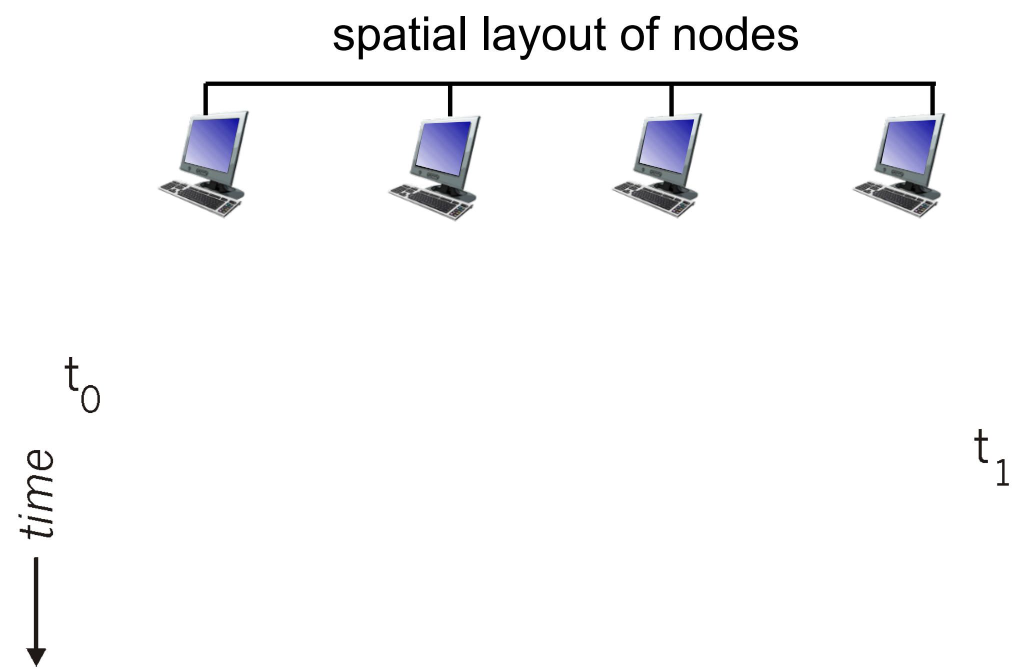
- 1976, Metcalfe & Boggs at Xerox
    - Later at 3COM
- Based on the Aloha network in Hawaii
- Named after the "*luminiferous ether*"
- Centered around a broadcast bus
- Simple link-level protocol, scales pretty well
- Tremendously successful
- Still in widespread use
    - many orders of magnitude increase in bandwidth since early versions

"CSMA/CD"

- Carrier sense
  - Listen before you speak
- Multiple access
  - Multiple hosts can access the network
- Collision detect
  - Detect and respond to cases where two hosts collide

CSMA collisions

- collisions can still occur: propagation delay means two nodes may not hear each other's transmission
- collision: entire packet transmission time wasted
  - distance & propagation delay play role in in determining collision probability

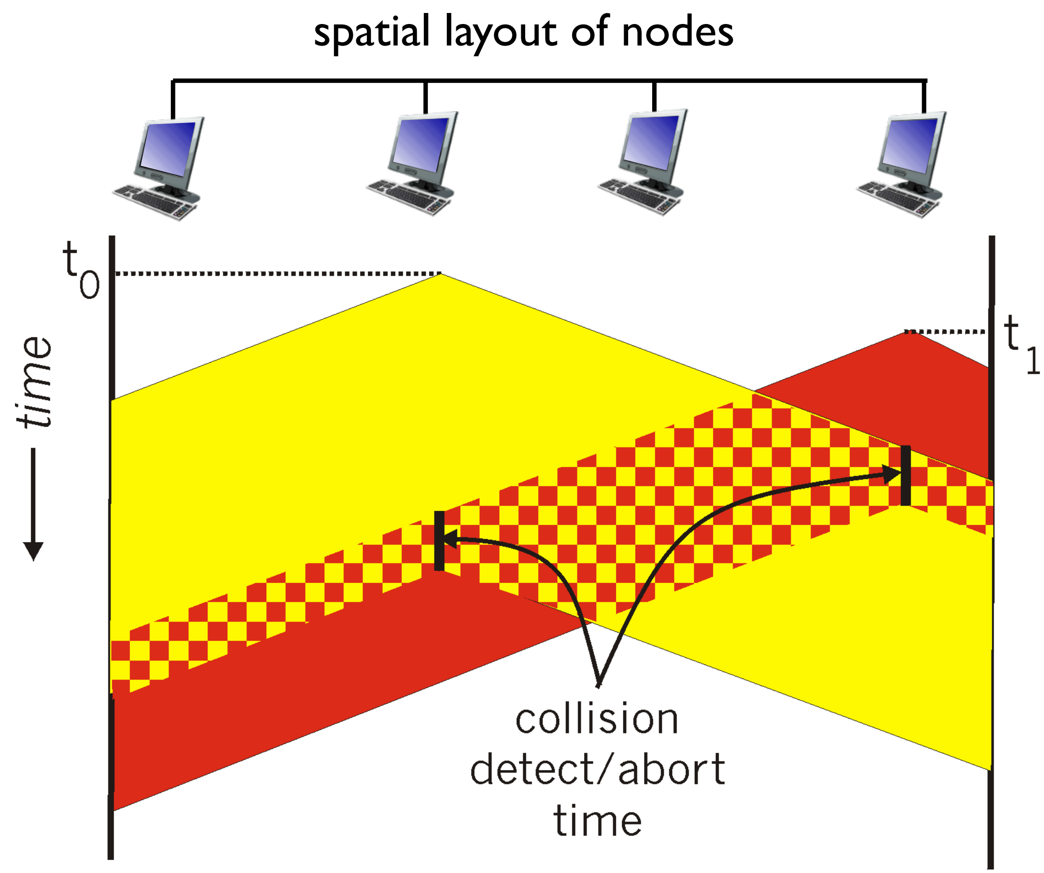


CSMA/CD (collision detection)

CSMA/CD: carrier sensing, deferral as in CSMA
- collisions *detected* within short time
- colliding transmissions aborted, reducing channel wastage

❖ collision detection:
- easy in wired LANs: measure signal strengths, compare transmitted, received signals
- difficult in wireless LANs: received signal strength overwhelmed by local transmission strength

CSMA/CD (collision detection)



Ethernet CSMA/CD algorithm

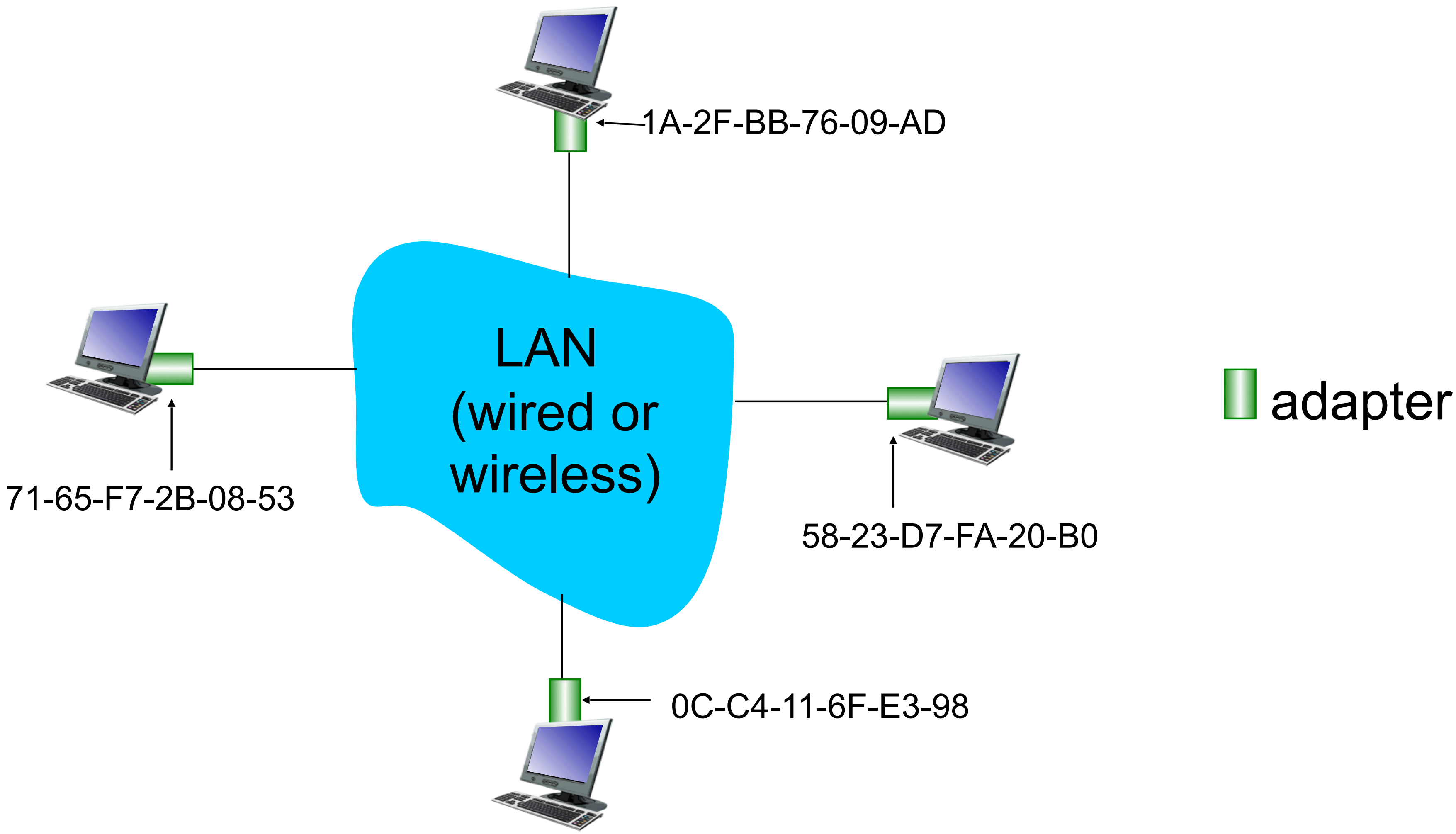
1. NIC receives datagram from network layer, creates frame
2. If channel idle, starts frame transmission. If channel busy, wait until channel idle, then transmit.
3. If entire frame transmitted without detecting another transmission, done!
4. If another transmission detected, abort and send jam signal
5. After aborting, NIC enters *binary (exponential) backoff:*
   - after *m*th collision, choose *K* at random from {$0,1,2, \ldots, 2^m-1$}. Wait K·512 bit times, return to Step 2
   - longer backoff interval with more collisions

MAC addresses

- 32-bit IP address:
  - *network-layer* address for interface
  - used for layer 3 (network layer) forwarding (coming up)
- MAC (or LAN or physical or Ethernet) address:
  - function: *used 'locally" to get frame from one interface to another physically-connected interface (same network, in IP-addressing sense)*
  - 48 bit MAC address (for most LANs) burned in NIC ROM, also sometimes software settable
  - e.g.: 1A-2F-BB-76-09-AD

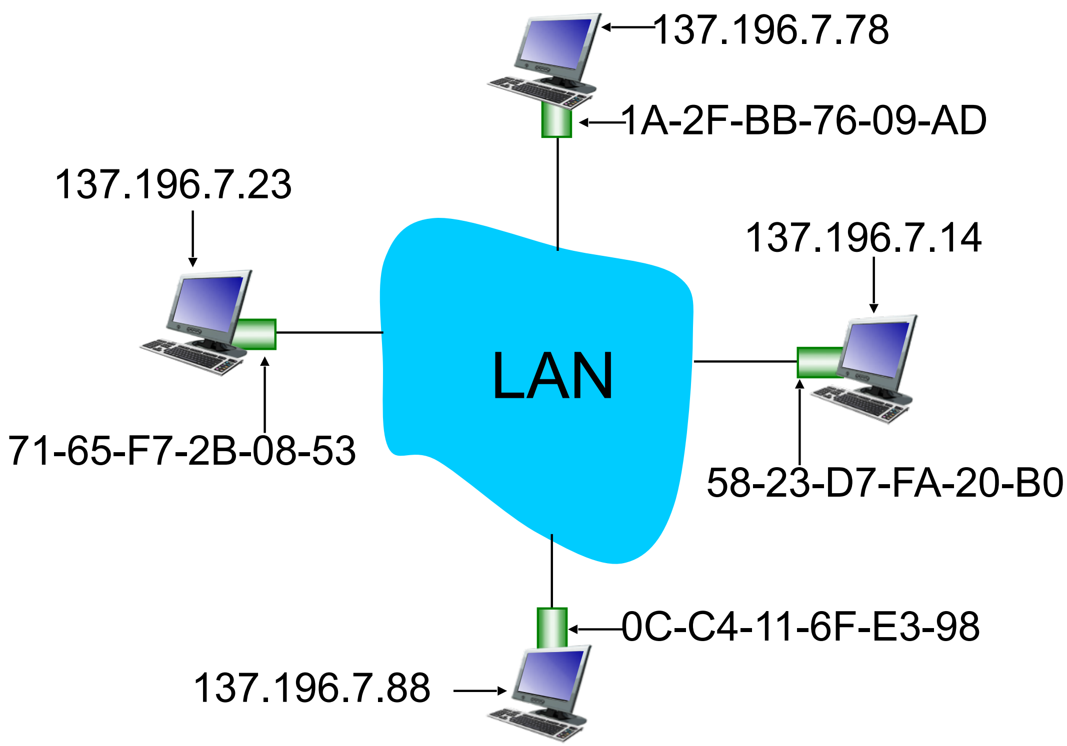
MAC addresses on a LAN

each adapter on LAN has unique *MAC* address



ARP: address resolution protocol

Question: how to determine interface's MAC address, knowing its IP address?



ARP table: each IP node (host, router) on LAN has table

- IP/MAC address mappings for some LAN nodes:

  < IP address; MAC address; TTL>

- TTL (Time To Live): time after which address mapping will be forgotten (typically 20 min)

ARP protocol: same LAN

- A wants to send datagram to B
  - B's MAC address not in A's ARP table.

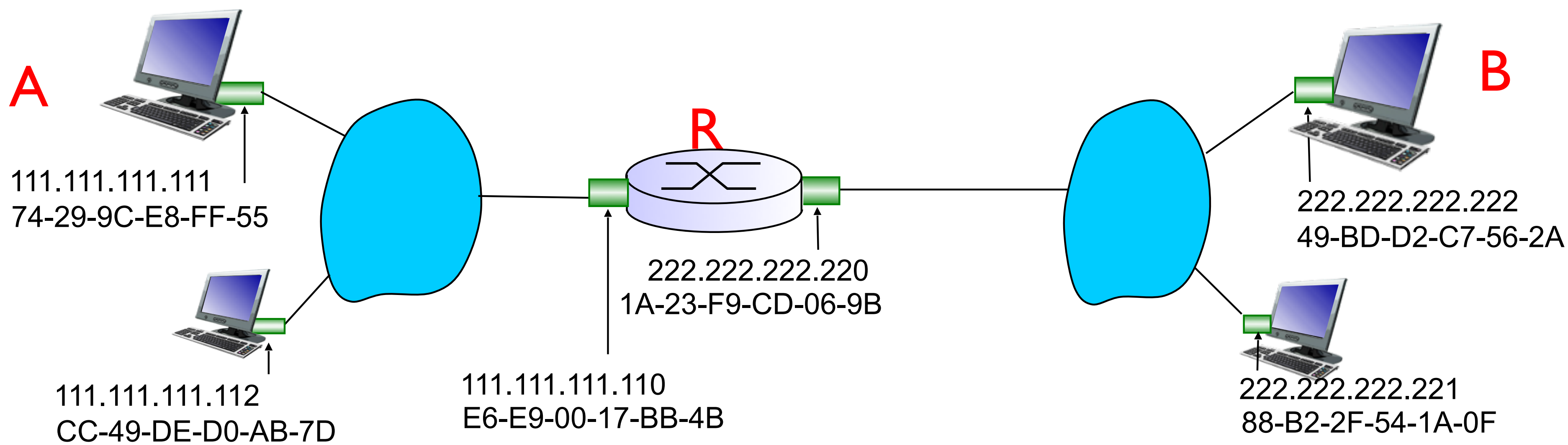
ARP protocol: same LAN

- A wants to send datagram to B
  - B's MAC address not in A's ARP table.
- A broadcasts ARP query packet, containing B's IP address
  - dest MAC address = FF-FF-FF-FF-FF-FF
  - all nodes on LAN receive ARP query
- B receives ARP packet, replies to A with its (B's) MAC address
  - frame sent to A's MAC address (unicast)
- A caches (saves) IP-to-MAC address pair in its ARP table until information becomes old (times out)
  - soft state: goes away unless refreshed
- ARP is "plug-and-play":
  - nodes create their ARP tables *without intervention from net administrator*

Addressing: routing to another LAN

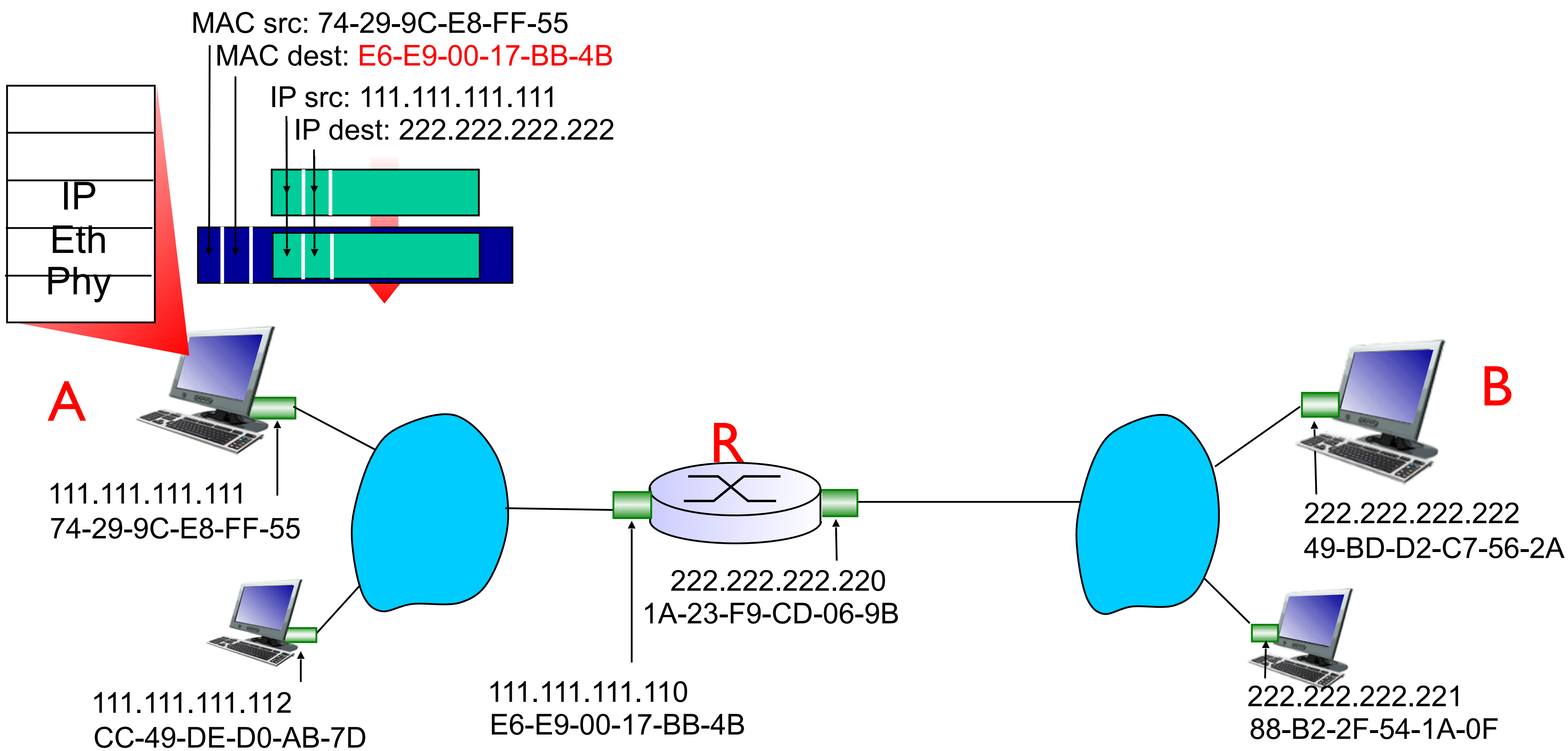
walkthrough: send datagram from A to B via R

- focus on addressing – at IP (datagram) and MAC layer (frame)
- assume A knows B's IP address
- assume A knows IP address of first hop router, R
- assume A knows R's MAC address (how?)



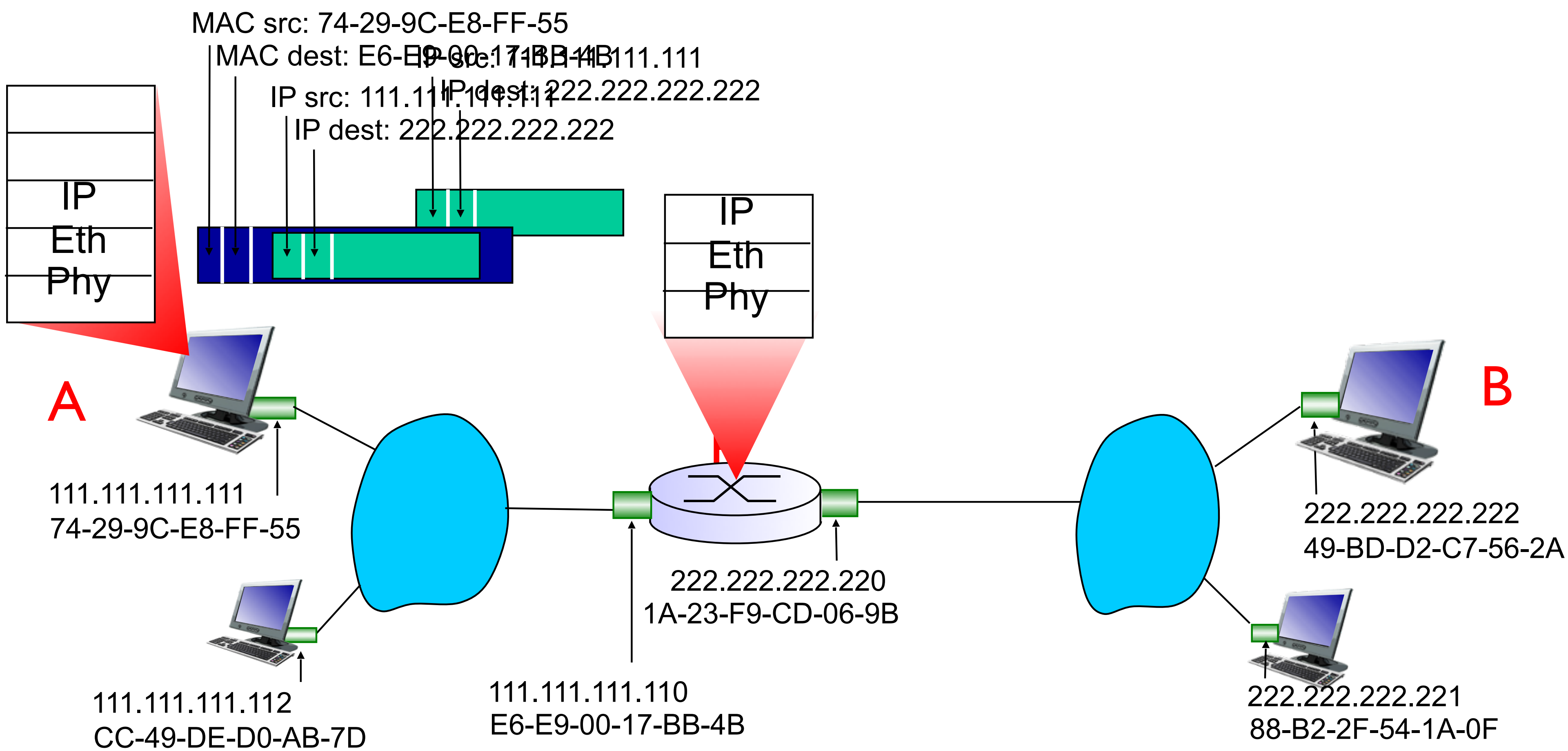
Addressing: routing to another LAN

- A creates IP datagram with IP source A, destination B
- A creates link-layer frame with R's MAC address as dest, frame contains A-to-B IP datagram



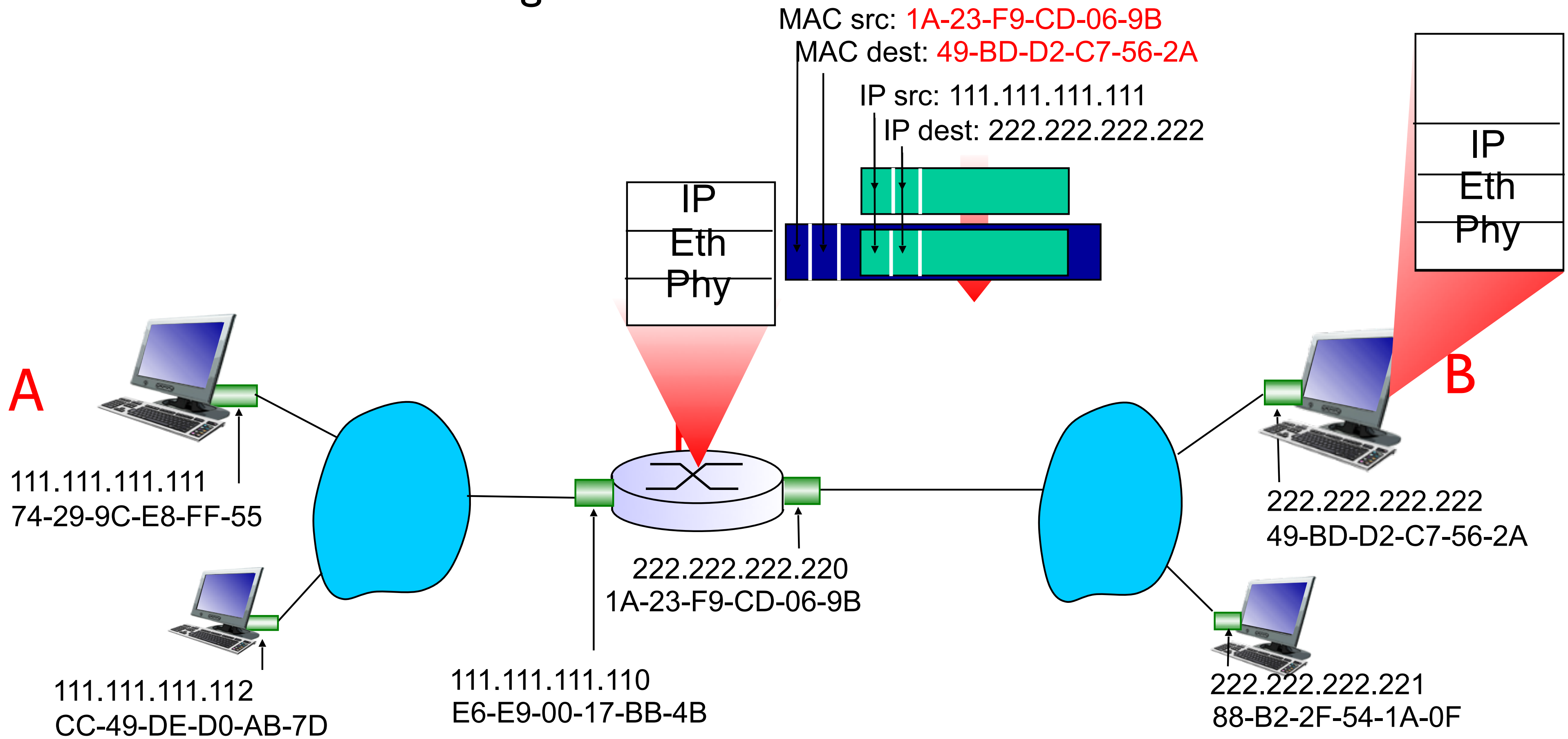
Addressing: routing to another LAN

- frame sent from A to R
- frame received at R, datagram removed, passed up to IP



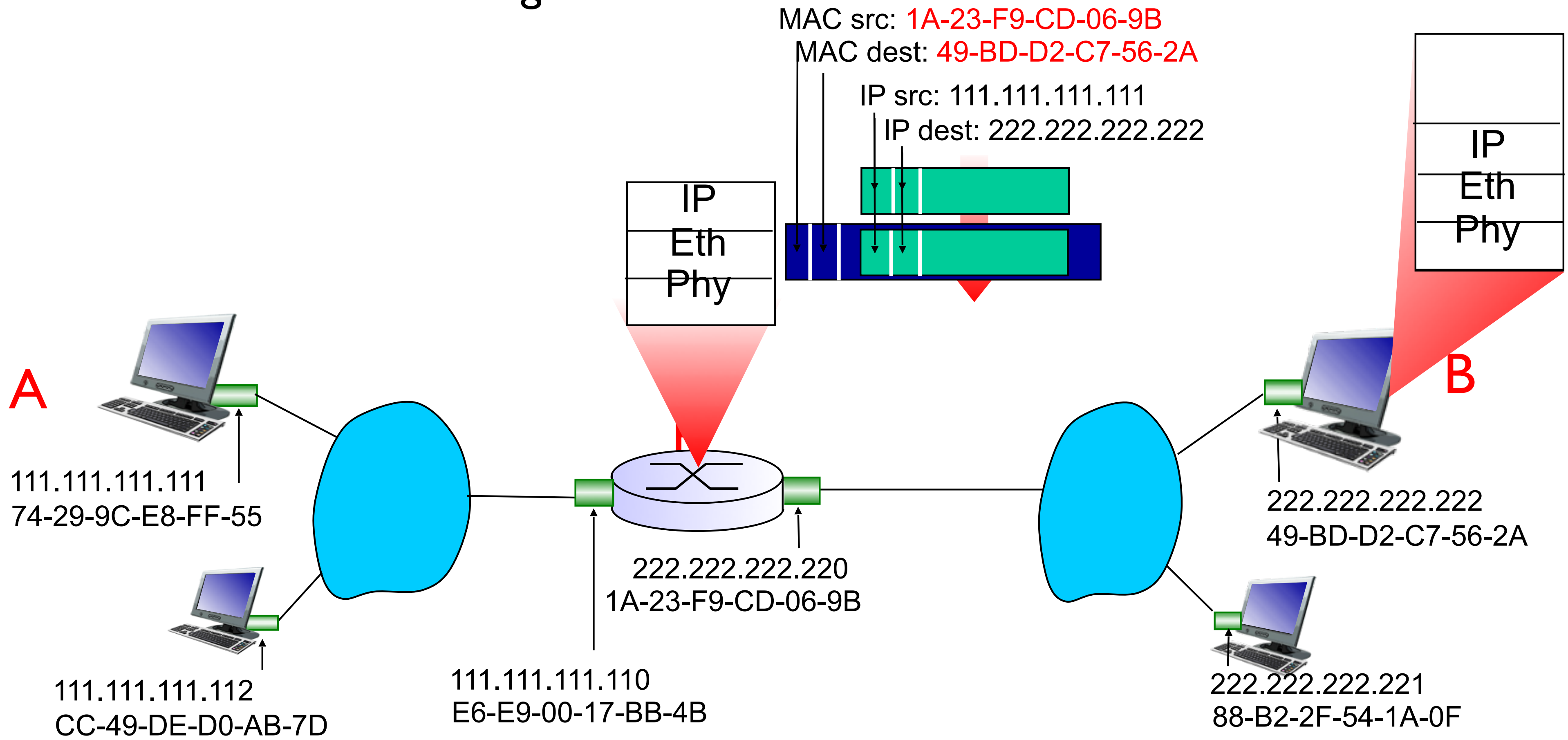
Addressing: routing to another LAN

- R forwards datagram with IP source A, destination B
- R creates link-layer frame with B's MAC address as dest, frame contains A-to-B IP datagram



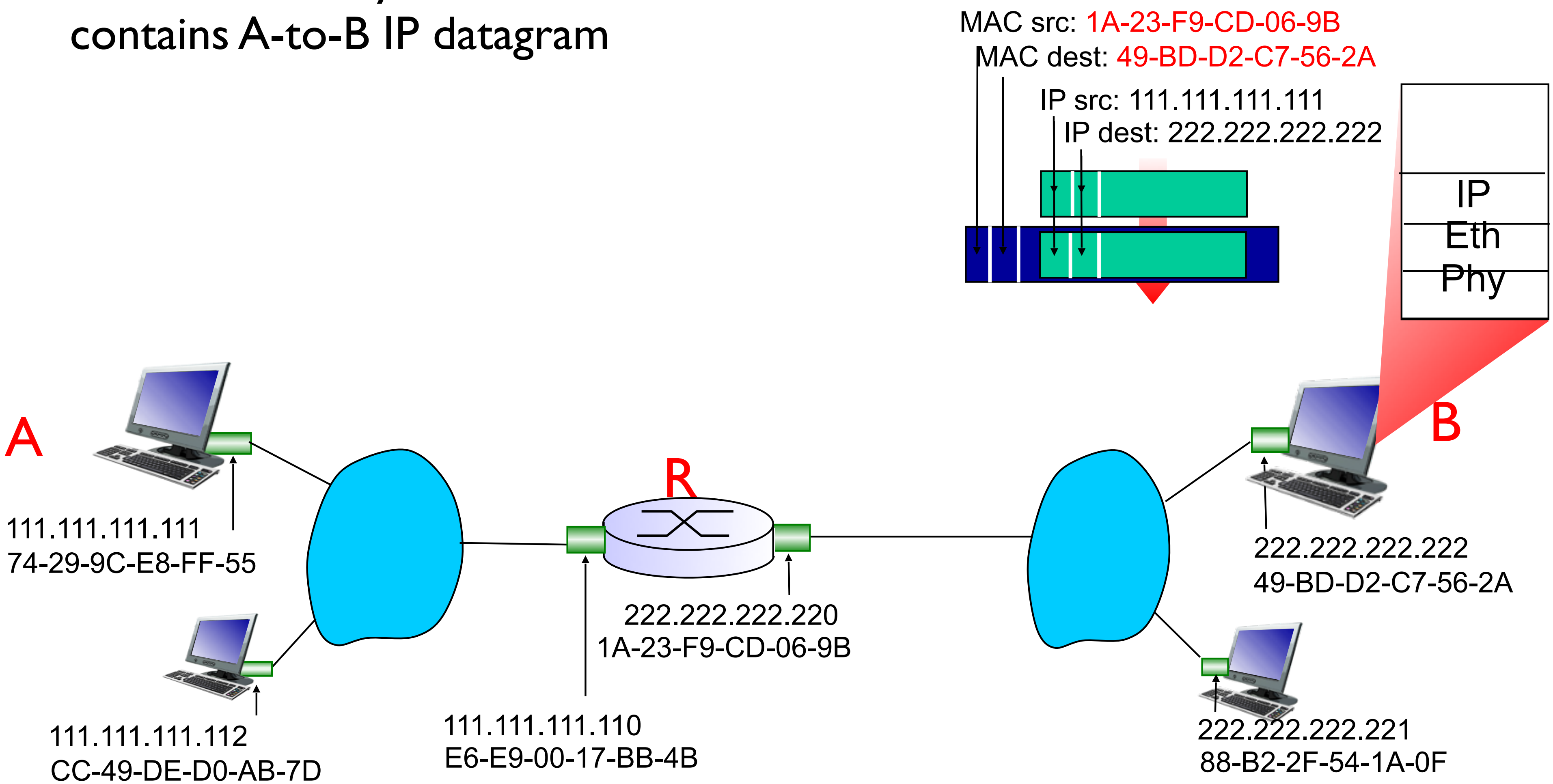
Addressing: routing to another LAN

- R forwards datagram with IP source A, destination B
- R creates link-layer frame with B's MAC address as dest, frame contains A-to-B IP datagram



Addressing: routing to another LAN

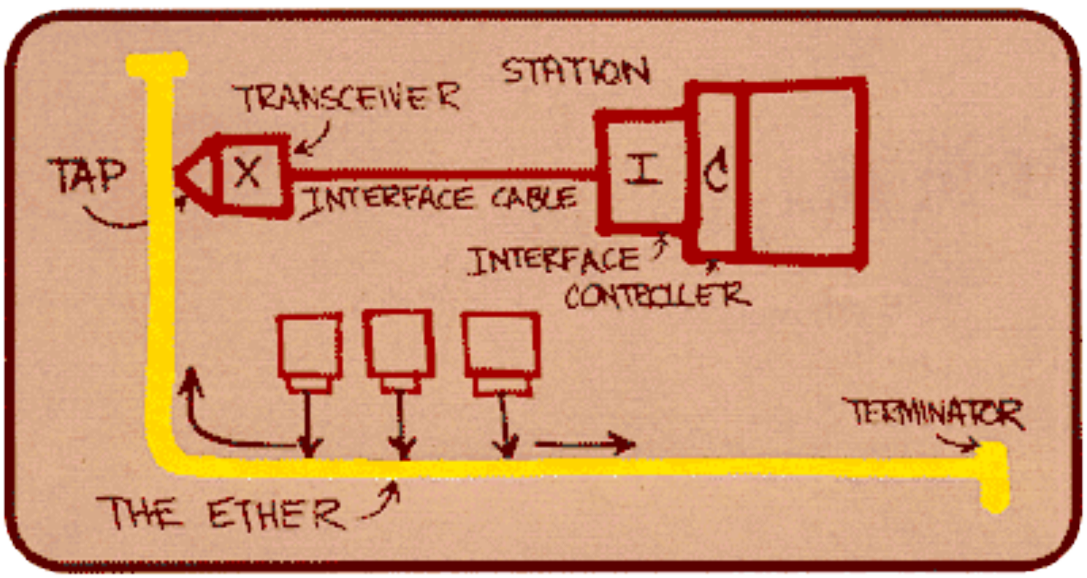
- R forwards datagram with IP source A, destination B
- R creates link-layer frame with B's MAC address as dest, frame contains A-to-B IP datagram



Ethernet

"dominant" wired LAN technology:

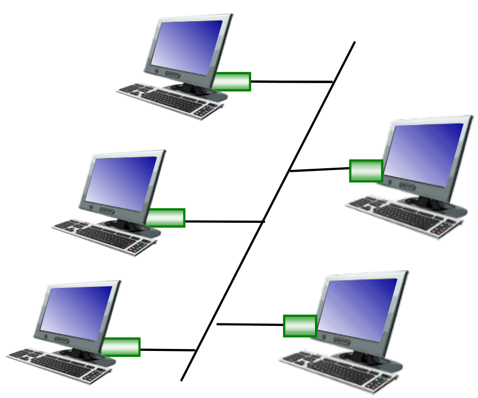
- cheap $20 for NIC
- first widely used LAN technology
- simpler, cheaper than token LANs and ATM
- kept up with speed race: 10 Mbps – 10 Gbps



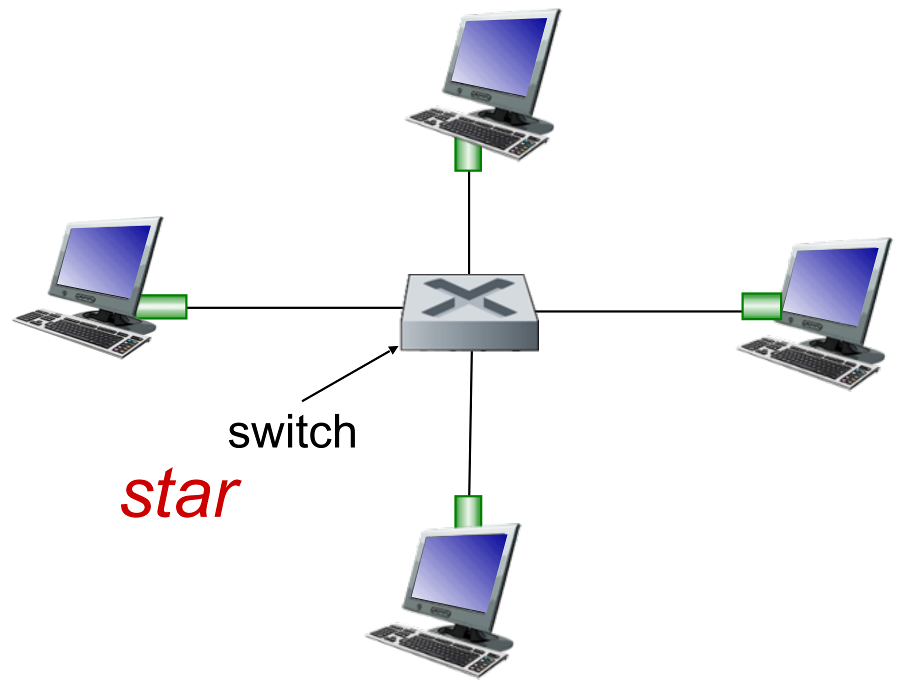
Metcalfe's Ethernet sketch

Ethernet: physical topology

- *bus:* popular through mid 90s
  - all nodes in same collision domain
- *star:* prevails today
  - active *switch* in center
  - each "spoke" runs a (separate) Ethernet protocol (nodes do not collide with each other)



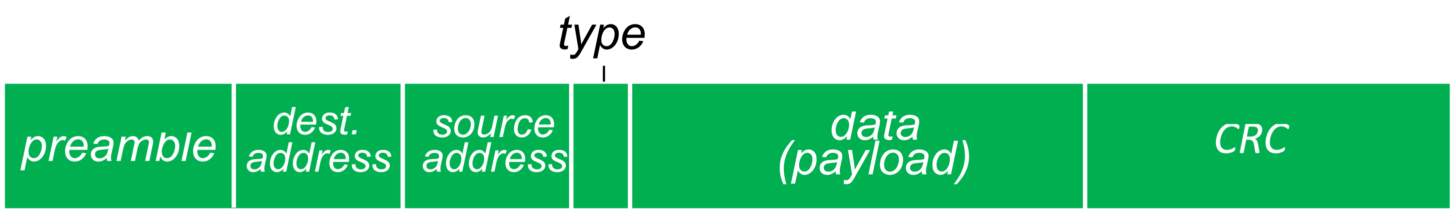
bus: coaxial cable



star

Ethernet frame structure

sending adapter encapsulates IP datagram (or other network layer protocol packet) in Ethernet frame

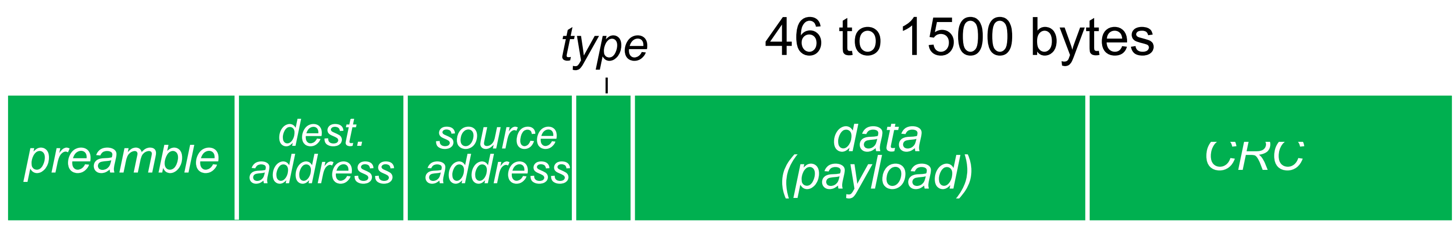


preamble:

- 7 bytes with pattern 10101010 followed by one byte with pattern 10101011
- used to synchronize receiver, sender clock rates

Ethernet frame structure (more)

- addresses: 6 byte source, destination MAC addresses
  - if adapter receives frame with matching destination address, or with broadcast address), it passes data in frame to network layer protocol
  - otherwise, adapter discards frame
- type: indicates higher layer protocol (mostly IP but others possible, e.g., Novell IPX, AppleTalk)
- CRC: cyclic redundancy check at receiver (basically, a hash of the frame)
  - error detected: frame is dropped



Ethernet: unreliable, connectionless

- *connectionless:* no handshaking between sending and receiving NICs
- *unreliable:* receiving NIC doesnt send acks or nacks to sending NIC
  - data in dropped frames recovered only if higher network layer ensures reliability (e.g., TCP), otherwise dropped data lost
- Ethernet's MAC protocol: *CSMA/CD wth binary backoff*

Ethernet Problems

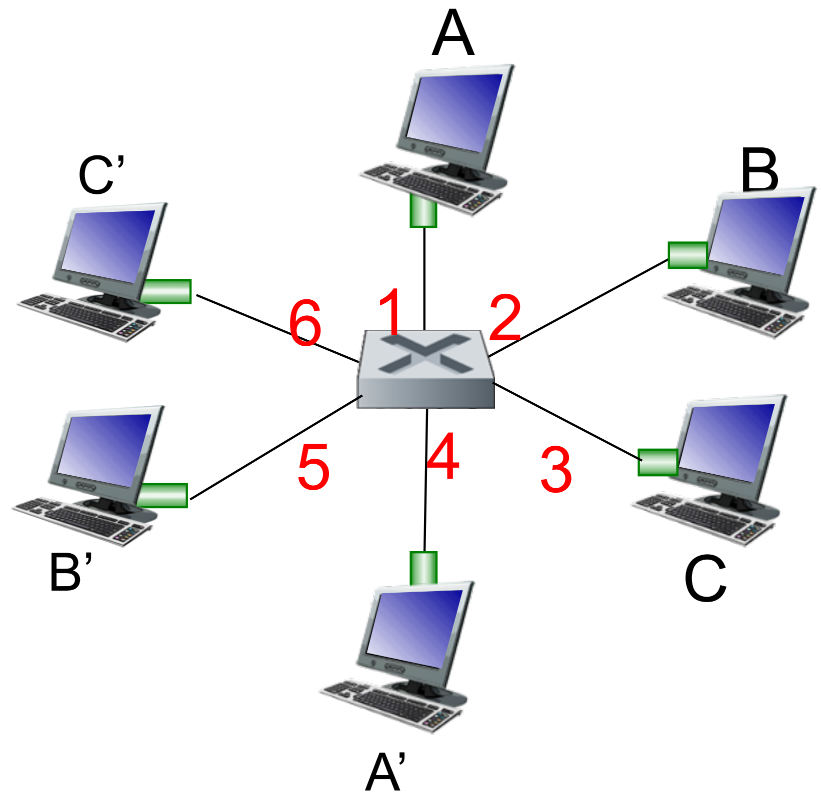
- The endpoints are trusted to follow the collision-detect and retransmit protocol
  - Certification process tries to assure compliance
  - Not everyone always backs off exponentially
- Hosts are trusted to only listen to packets destined for them
  - But the data is available for all to see
    - All packets are broadcast on the wire
    - Can place Ethernet card in promiscuous mode and liste

Ethernet switch

- link-layer device: takes an *active* role
  - store, forward Ethernet frames
  - examine incoming frame's MAC address, selectively forward frame to one-or-more outgoing links when frame is to be forwarded on segment, uses CSMA/CD to access segment
- *transparent*
  - hosts are unaware of presence of switches
- *plug-and-play, self-learning*
  - switches do not need to be configured

Switch: *multiple* simultaneous transmissions

- hosts have dedicated, direct connection to switch
- switches buffer packets
- Ethernet protocol used on *each* incoming link, but no collisions; full duplex
  - each link is its own collision domain
- *switching:* A-to-A' and B-to-B' can transmit simultaneously, without collisions



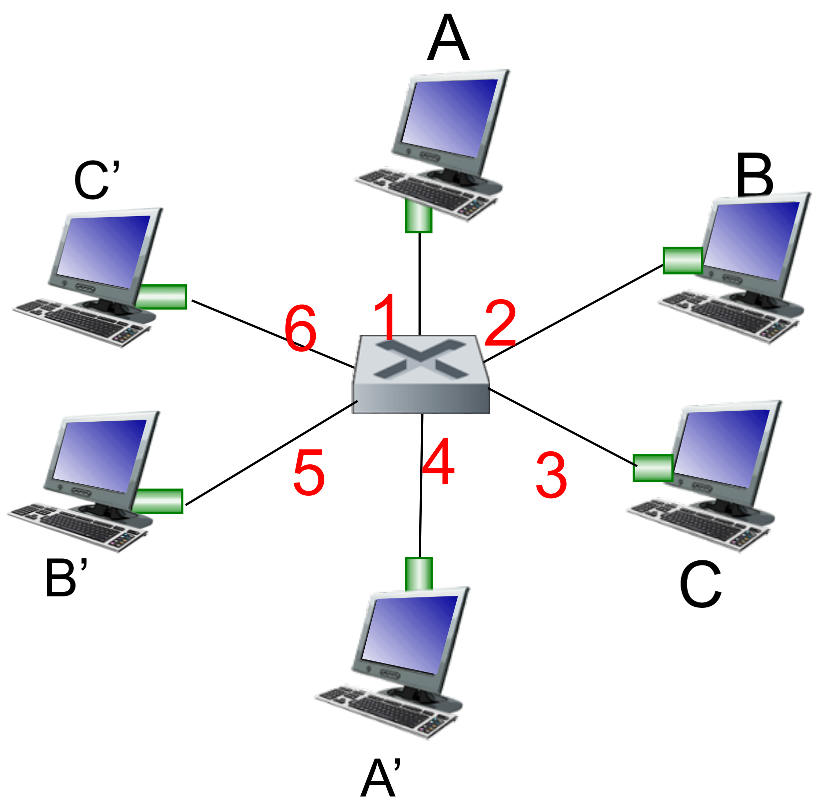
switch with six interfaces (1,2,3,4,5,6)

Switch forwarding table

Q: how does switch know A' reachable via interface 4, B' reachable via interface 5?

- A: each switch has a switch table, each entry:
  - (MAC address of host, interface to reach host, time stamp)
  - a routing table!

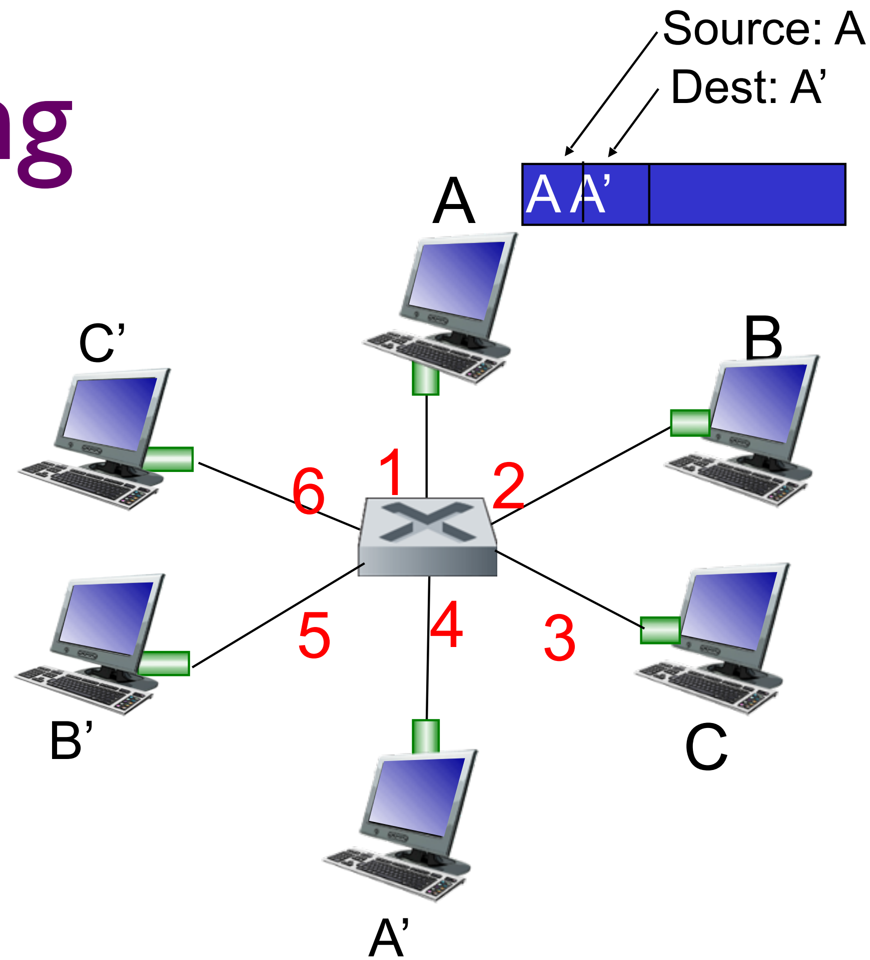
Q: how are entries created, maintained in switch table?



switch with six interfaces (1,2,3,4,5,6)

Switch: self-learning

- switch *learns* which hosts can be reached through which interfaces
  - when frame received, switch "learns" location of sender: incoming LAN segment
  - records sender/location pair in switch table

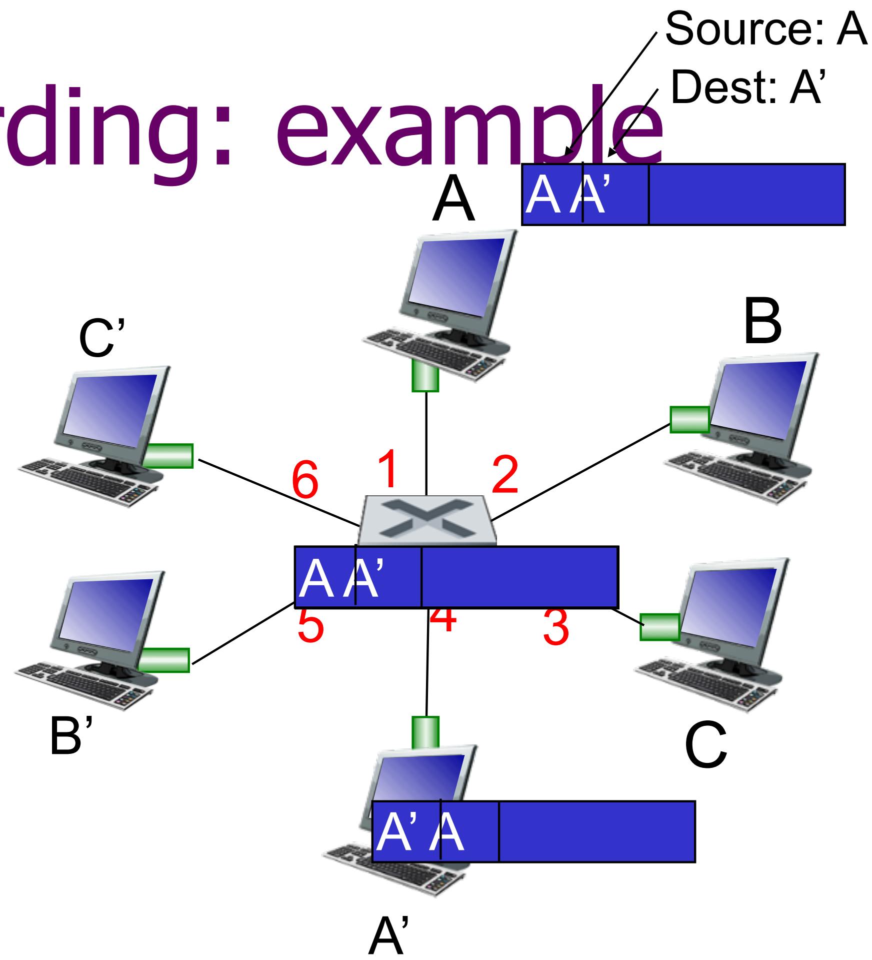


MAC addr	interface	TTL
A	*1*	*60*

Switch table (initially empty)

Self-learning, forwarding: example

- frame destination, A', locaton unknown: *flood*
- destination A location known: selectively send on just one link



MAC addr	interface	TTL
A	*1*	*60*
A'	*4*	*60*

switch table (initially empty)

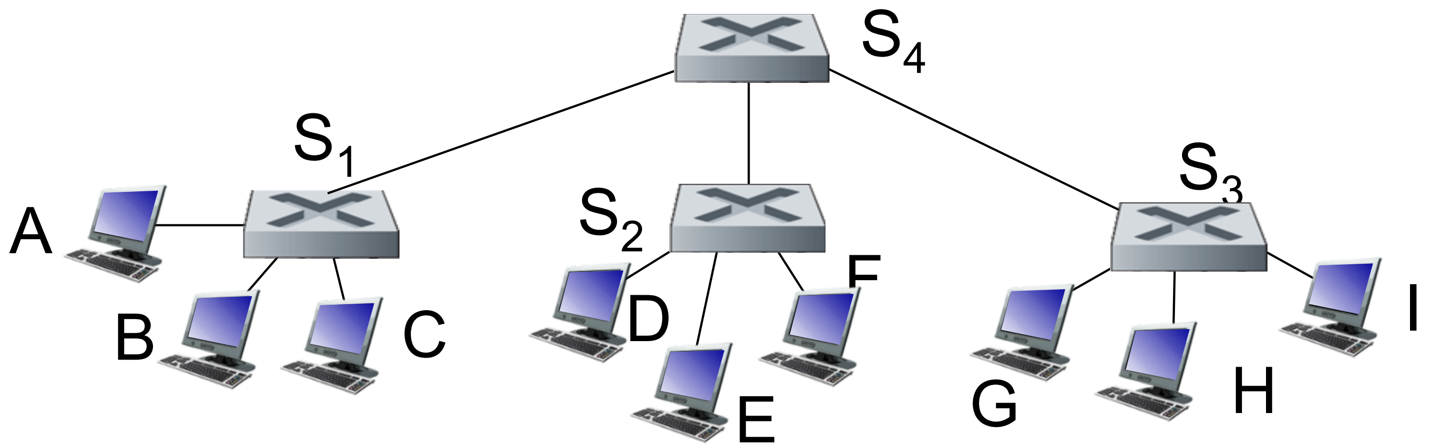
Switch: frame filtering/forwarding

when frame received at switch:

1. record incoming link, MAC address of sending host
2. index switch table using MAC destination address
3. if entry found for destination
   then {
   if destination on LAN segment from which frame arrived
   then drop frame
   else forward frame on interface indicated by entry
   }
   else flood /* forward on all interfaces except arriving interface */

Interconnecting switches

- switches can be connected together



Q: sending from A to G - how does S_1 know to forward frame destined to F via S_4 and S_3?

- A: self learning! (works exactly the same as in single-switch case!)

Lessons for LAN design

- Best-effort delivery simplifies network design
- A simple, distributed protocol can tolerate failures and be easy to administer

Application Layer
Transport Layer
Network Layer
Link Layer
Physical Layer

Network Layer

Network Layer

- There are lots of Local Area Networks
  - each with their own
    - address format and allocation scheme
    - packet format
    - LAN-level protocols, reliability guarantees
- Wouldn’t it be nice to tie them all together?
  - Nodes with multiple NICs can provide the glue!
  - Standardize address and packet formats
- This gives rise to an “Internetwork”
  - aka WAN (wide-area network)

Internetworking Origins

- Expensive supercomputers scattered throughout US
- Researchers scattered differently throughout the US
- Needed a way to connect researchers to expensive machinery

Internetworking Origins

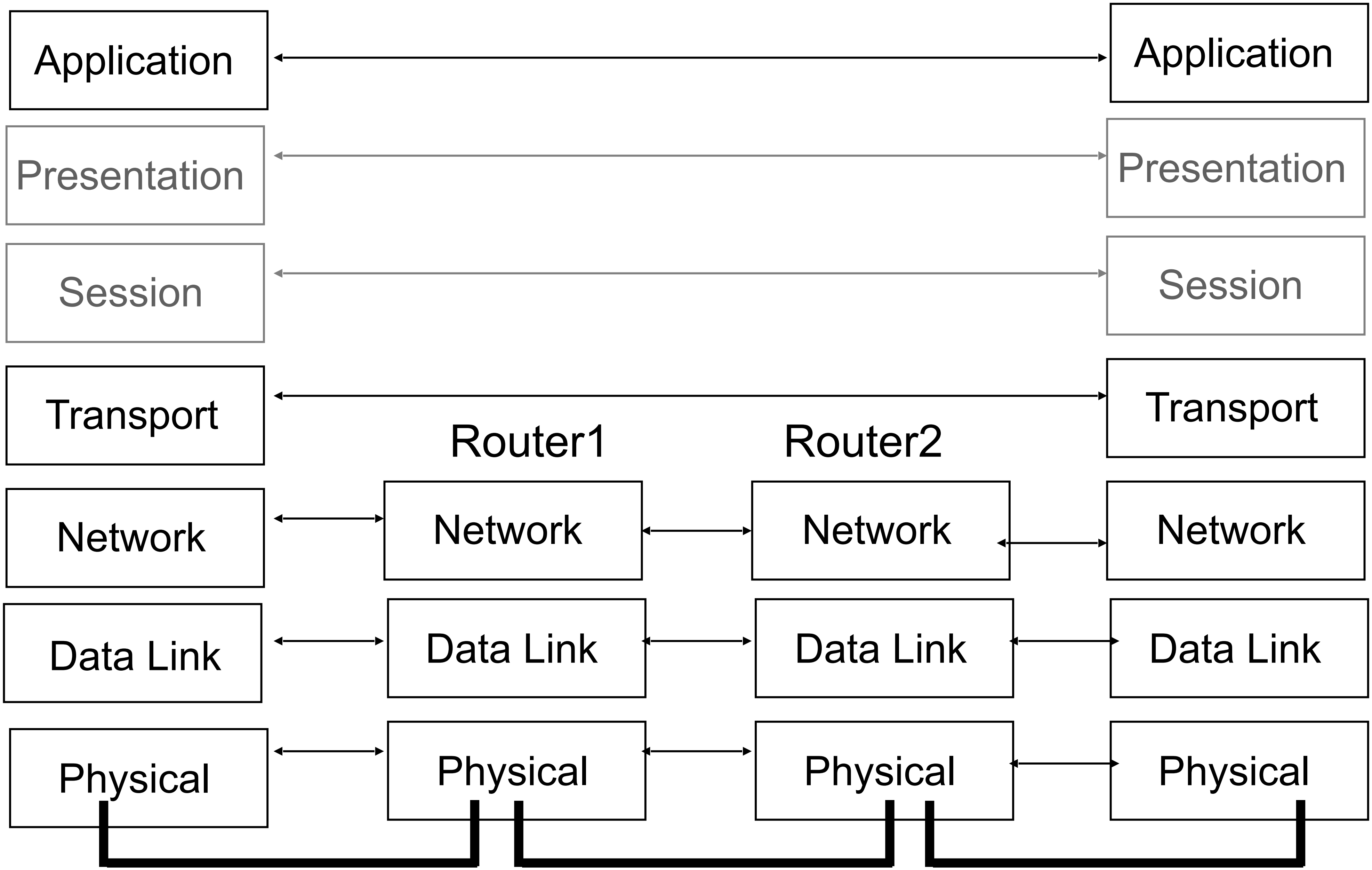
- Department of Defense initiated studies on how to build a resilient global network
  - How do you coordinate a nuclear attack ?
- Interoperability and dynamic routing are a must
  - Along with a lot of other properties
- Result: *Internet (orig. ARPAnet)*
- A **complex** system with **simple** components

Internet Overview

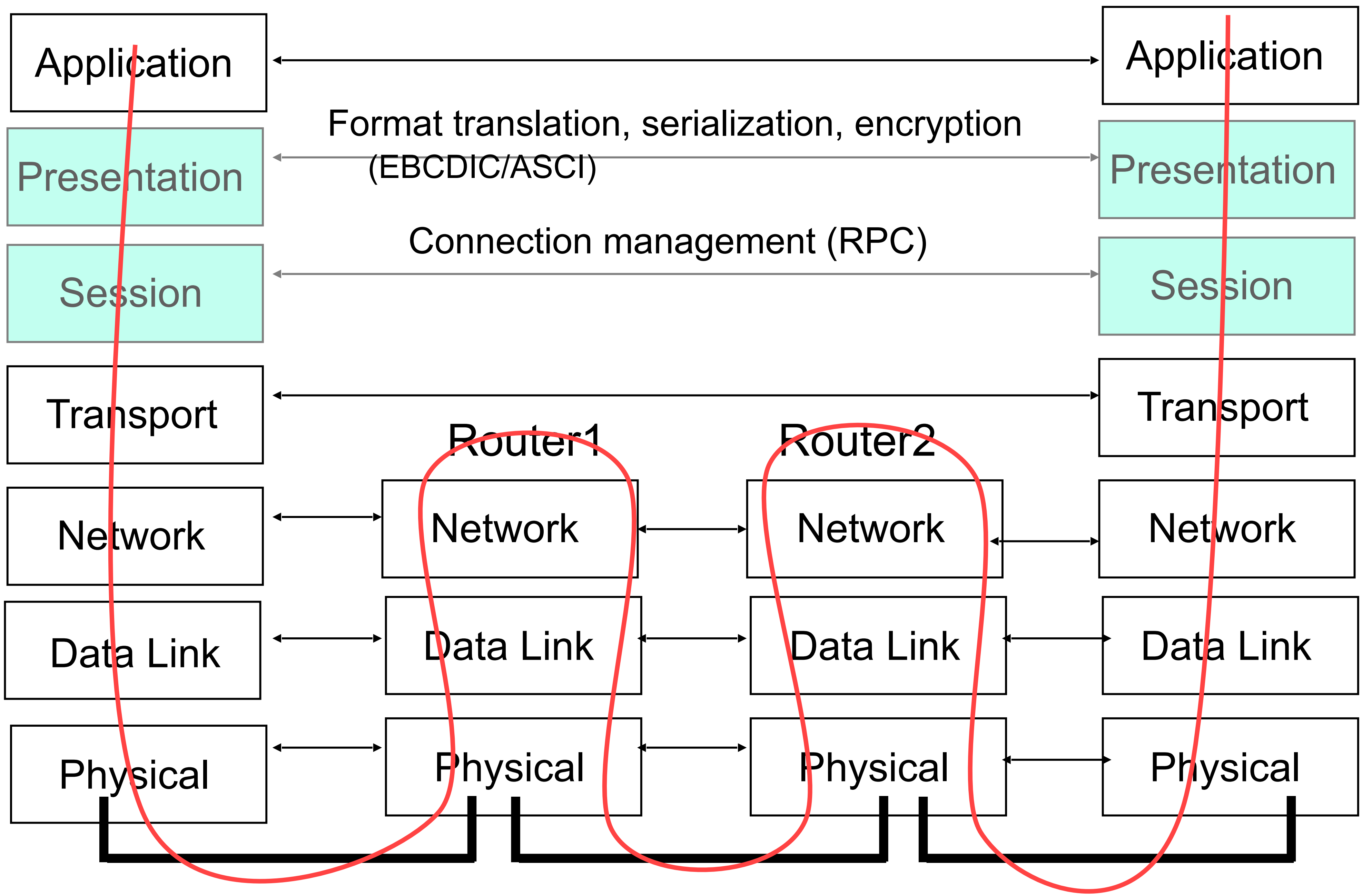
- Every host is assigned, and identified by, an <u>IP address</u>
- Messages are called <u>datagrams</u>
  - the term *packet* is probably more common though…
- Each datagram contains a <u>header</u> that specifies the destination address
- The network <u>routes</u> datagrams from the source to the destination

- Design Decision: What kinds of properties should the network provide?

The Big Picture



The Big Picture



Network Stack – quite literally

- Each layer has its own header
- You can think of packet as a stack
- On send, each layer pushes a header onto the stack
- On receipt, each layer pops a header
  - Headers often contain a "demultiplexer" like a port or protocol number to decide where to transfer control on the way up the stack.

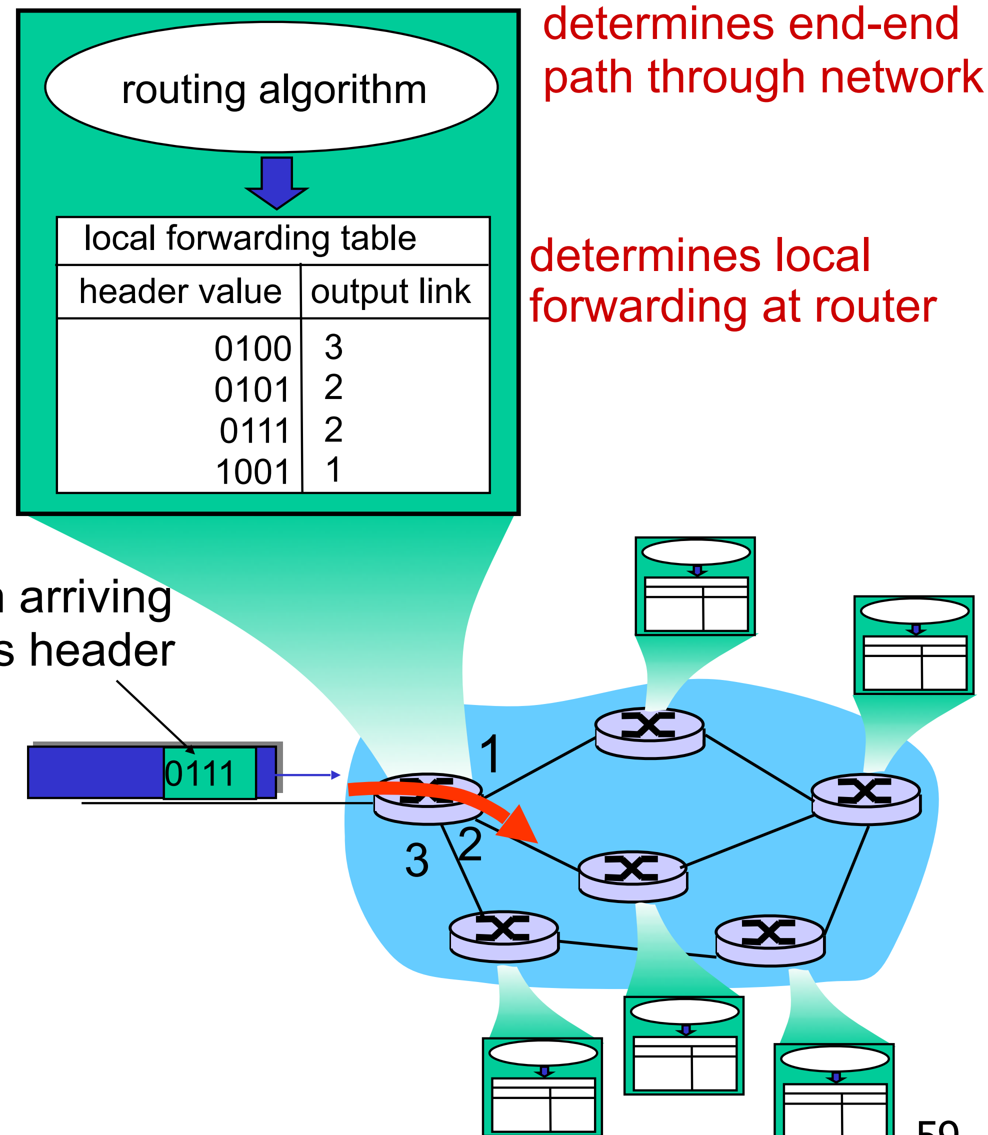
End-to-End Argument

- A kind of Occam's Razor for Internet architecture
- Application-specific properties are best provided by the applications, not the network
  - Guaranteed, or ordered, packet delivery, duplicate suppression, security, etc.
- The Internet performs the simplest packet routing and delivery service it can
  - Packets are sent on a best-effort basis
  - Higher-level applications do the rest

Two key network-layer functions

- *forwarding:* move packets from router's input to appropriate router output

- *routing:* determine route taken by packets from source to dest.
  - *routing algorithms*



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 models

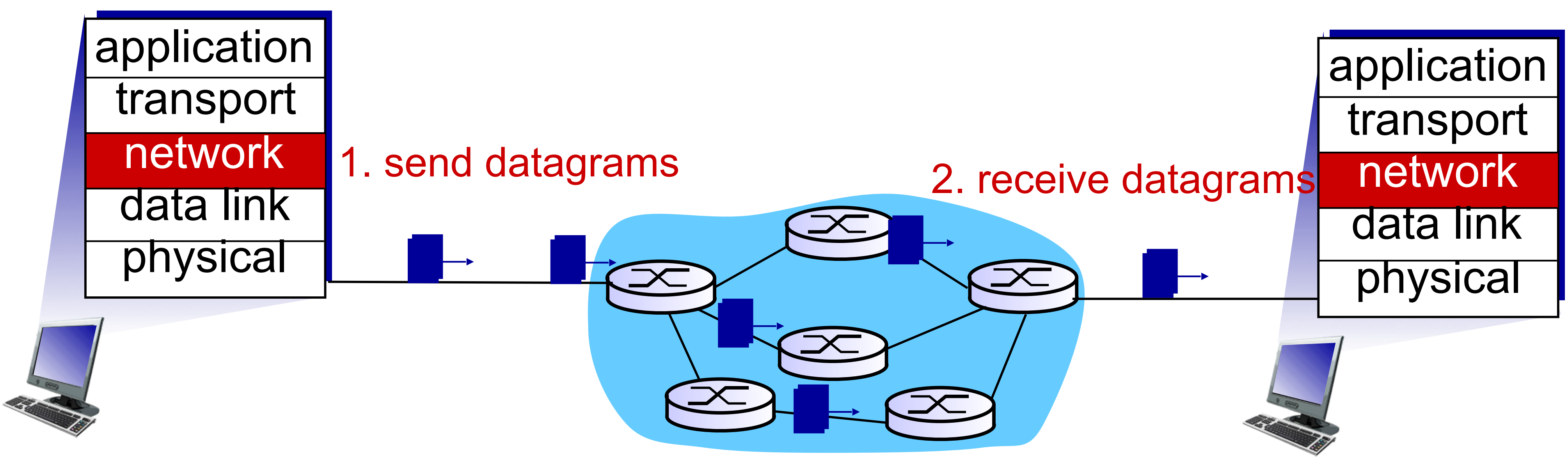
Network Architecture	Service Model	Guarantees ? Bandwidth	Loss	Order	Timing	Congestion feedback
Internet	best effort	none	no	no	no	no (inferred via loss)
ATM	CBR	constant rate	yes	yes	yes	no congestion
ATM	VBR	guaranteed rate	yes	yes	yes	no congestion
ATM	ABR	guaranteed minimum	no	yes	no	yes
ATM	UBR	none	no	yes	no	no

Connection, connection-less service

- *datagram* network provides network-layer *connectionless* service (IP)
- *virtual-circuit* network provides network-layer *connection* service (ATM)
  - Much like a phone network
- analogous to TCP/UDP connection-oriented / connectionless transport-layer services
  - Coming up…

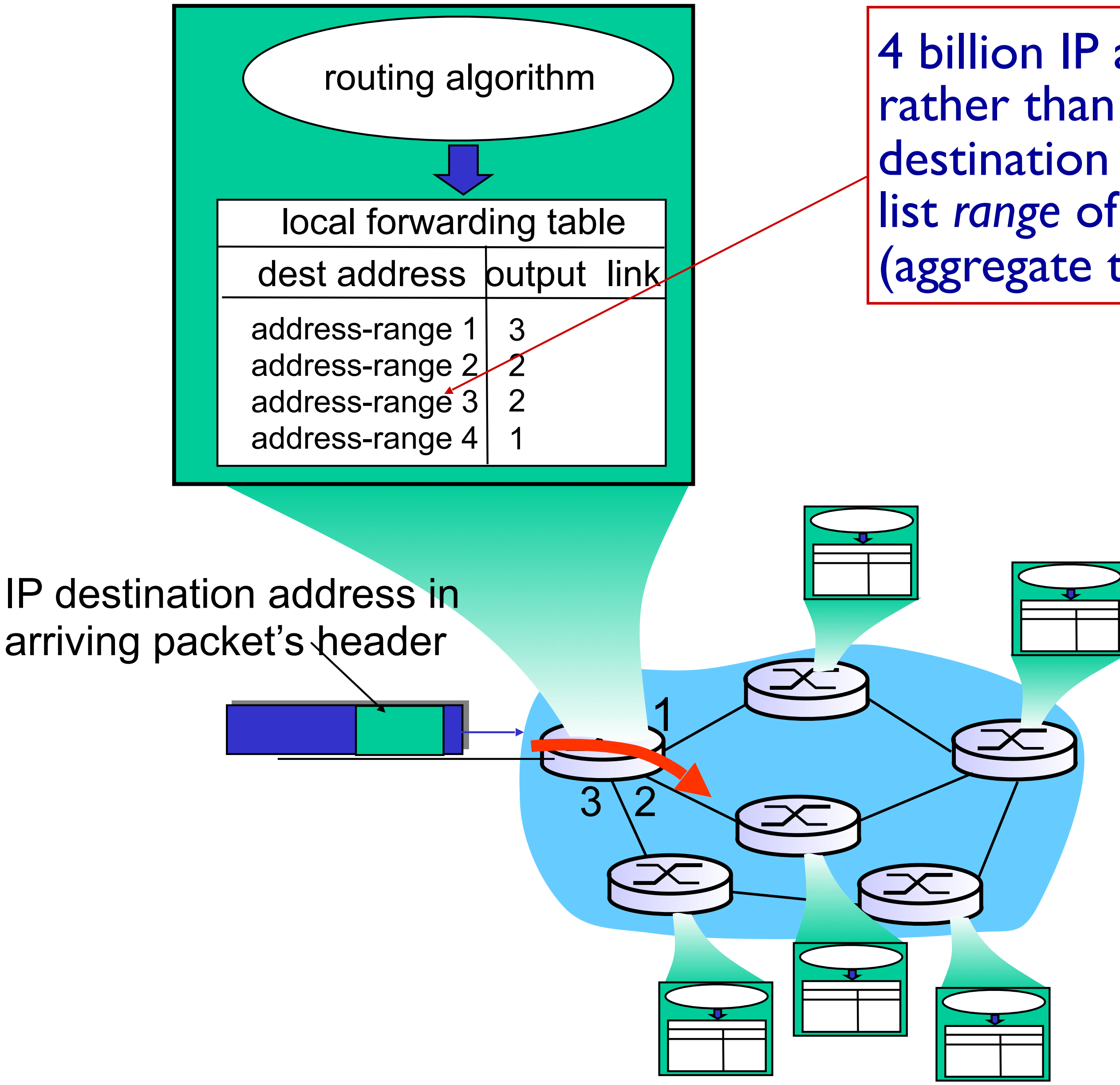
Datagram networks

- no call setup at network layer
- routers: no state about end-to-end connections
  - no network-level concept of "connection"
- packets forwarded using destination host address



Datagram forwarding table

4 billion IP addresses, so rather than list individual destination address list *range* of addresses (aggregate table entries)



Datagram forwarding table

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?

Longest prefix matching

longest prefix matching

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

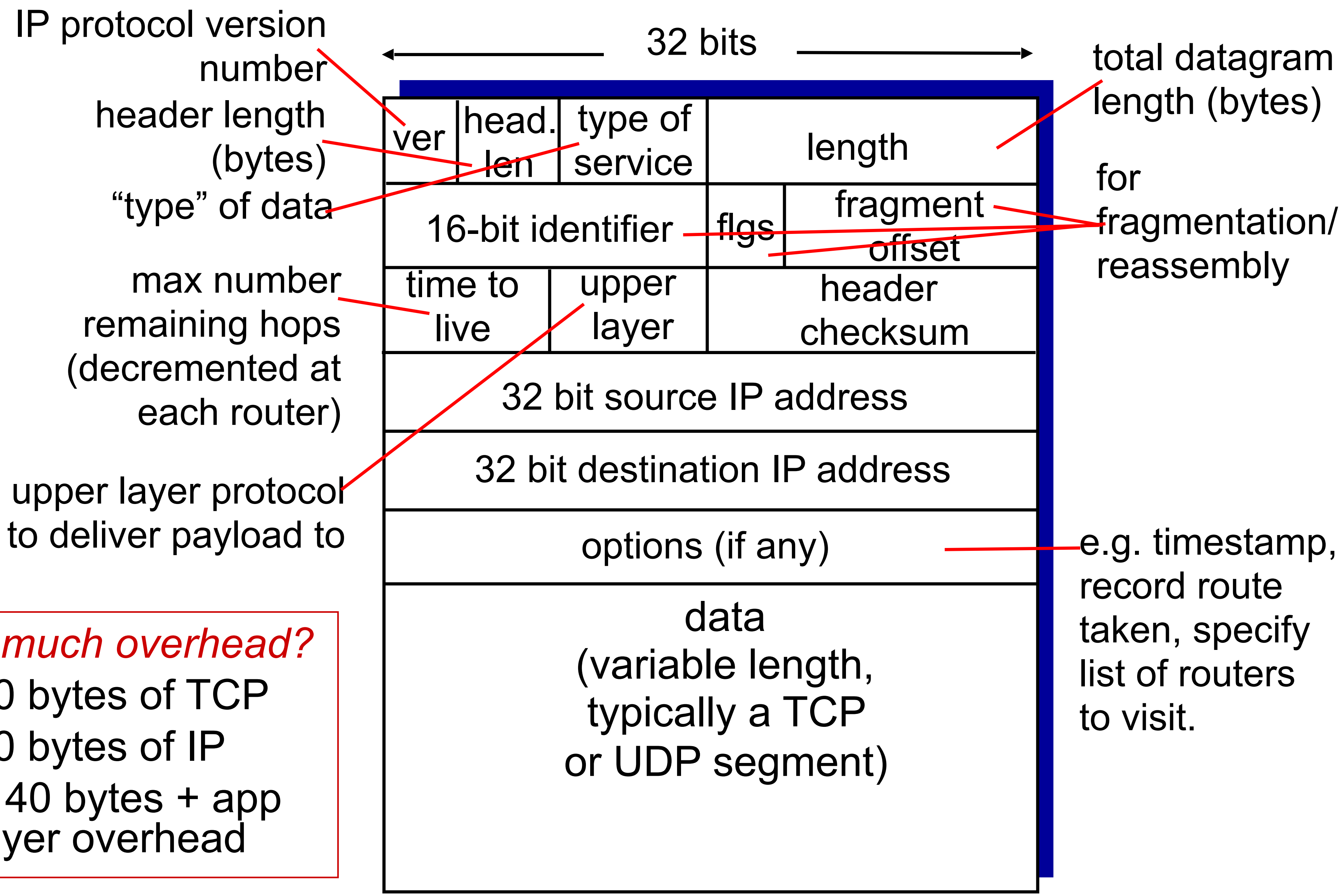
Destination Address Range	Link interface
11001000 00010111 00010*** *********	0
11001000 00010111 00011000 *********	1
11001000 00010111 00011*** *********	2
otherwise	3

examples:

DA: 11001000 00010111 00010110 10100001 which interface?

DA: 11001000 00010111 00011000 10101010 which interface?

IP datagram format

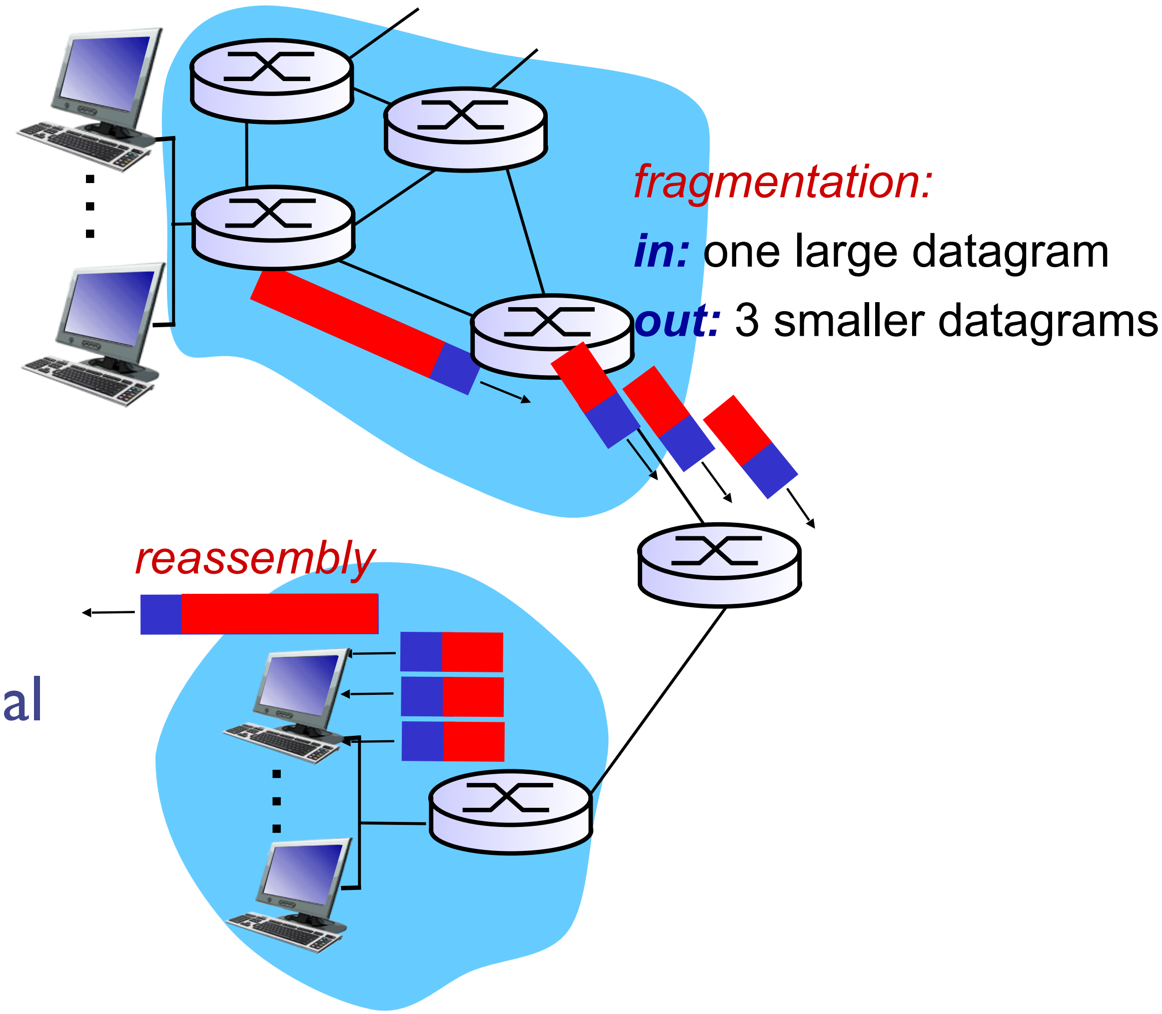


how much overhead?

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

IP fragmentation, reassembly

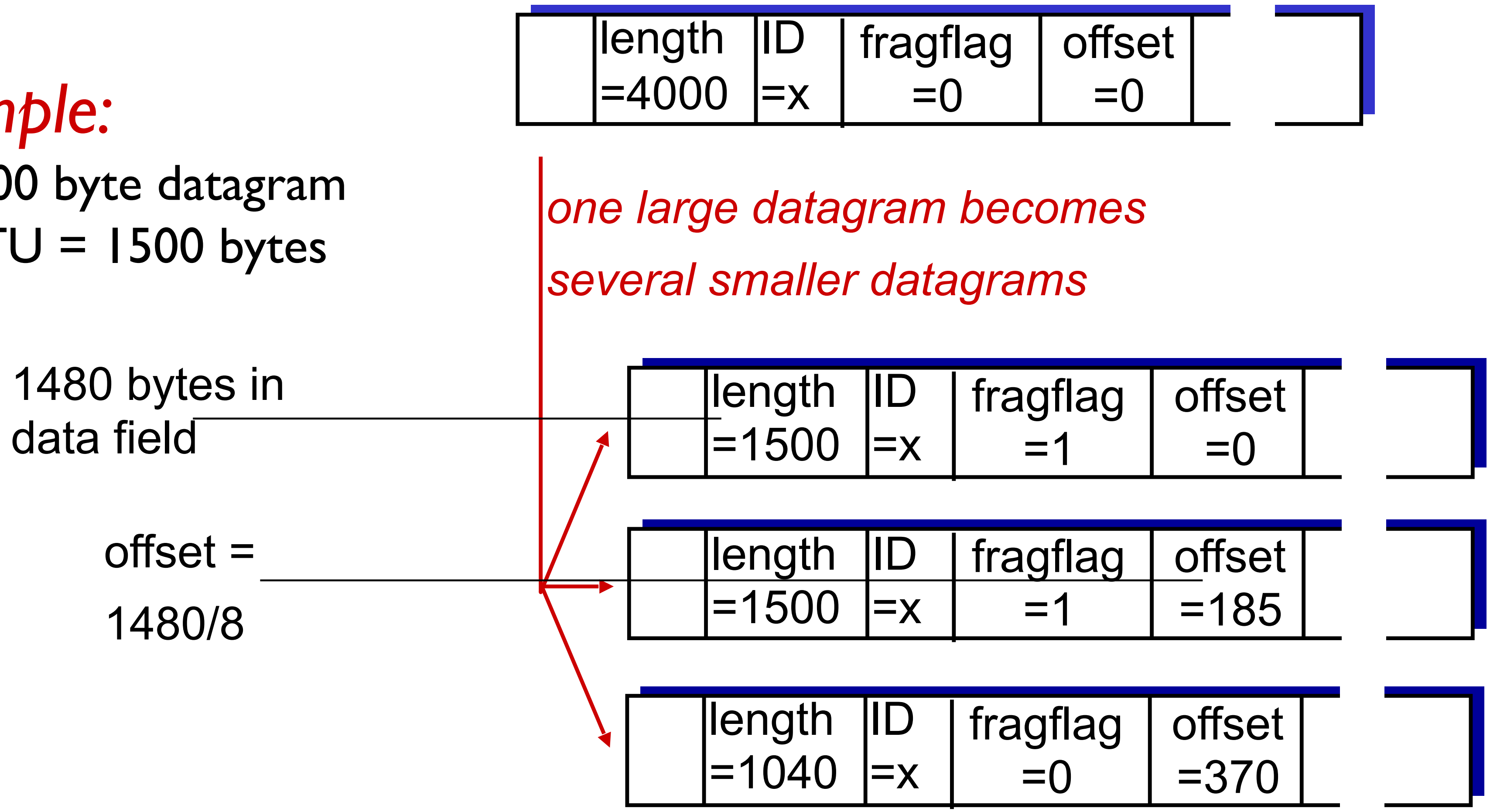
- network links have MTU (max.transfer size) - largest possible link-level frame
  - different link types, different MTUs
- large IP datagram divided ("fragmented") within net
  - one datagram becomes several datagrams
  - "reassembled" only at final destination
  - IP header bits used to identify, order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



IP fragmentation, reassembly

example:

- 4000 byte datagram
- MTU = 1500 bytes

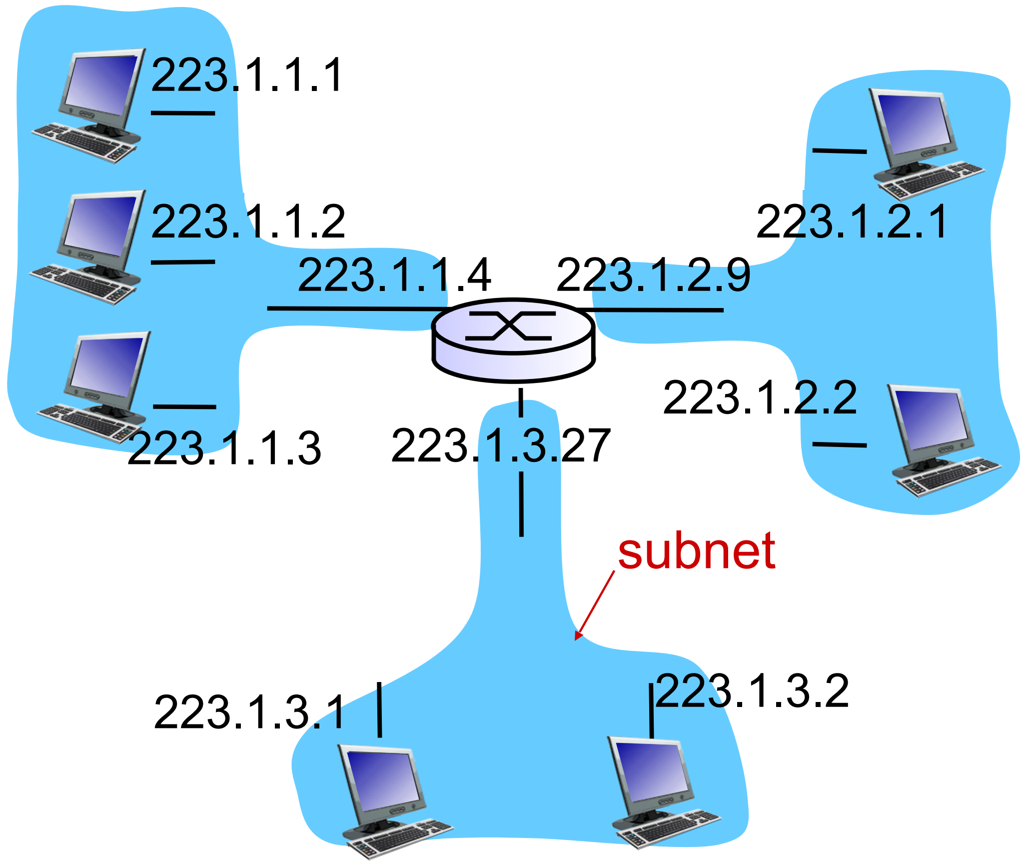


IP Addressing

- Every (active) NIC has an IP address
  - IPv4: 32-bit descriptor, e.g. 128.84.12.43
  - IPv6: 128-bit descriptor (but only 64 bits "functional")
  - Will use IPv4 unless specified otherwise…
- Each Internet Service Provider (ISP) owns a set of IP addresses
- ISPs assign IP addresses to NICs
- An IP address is not an identifier:
  - IP addresses can be re-used
  - Same NIC may have different IP addresses over time

Subnets

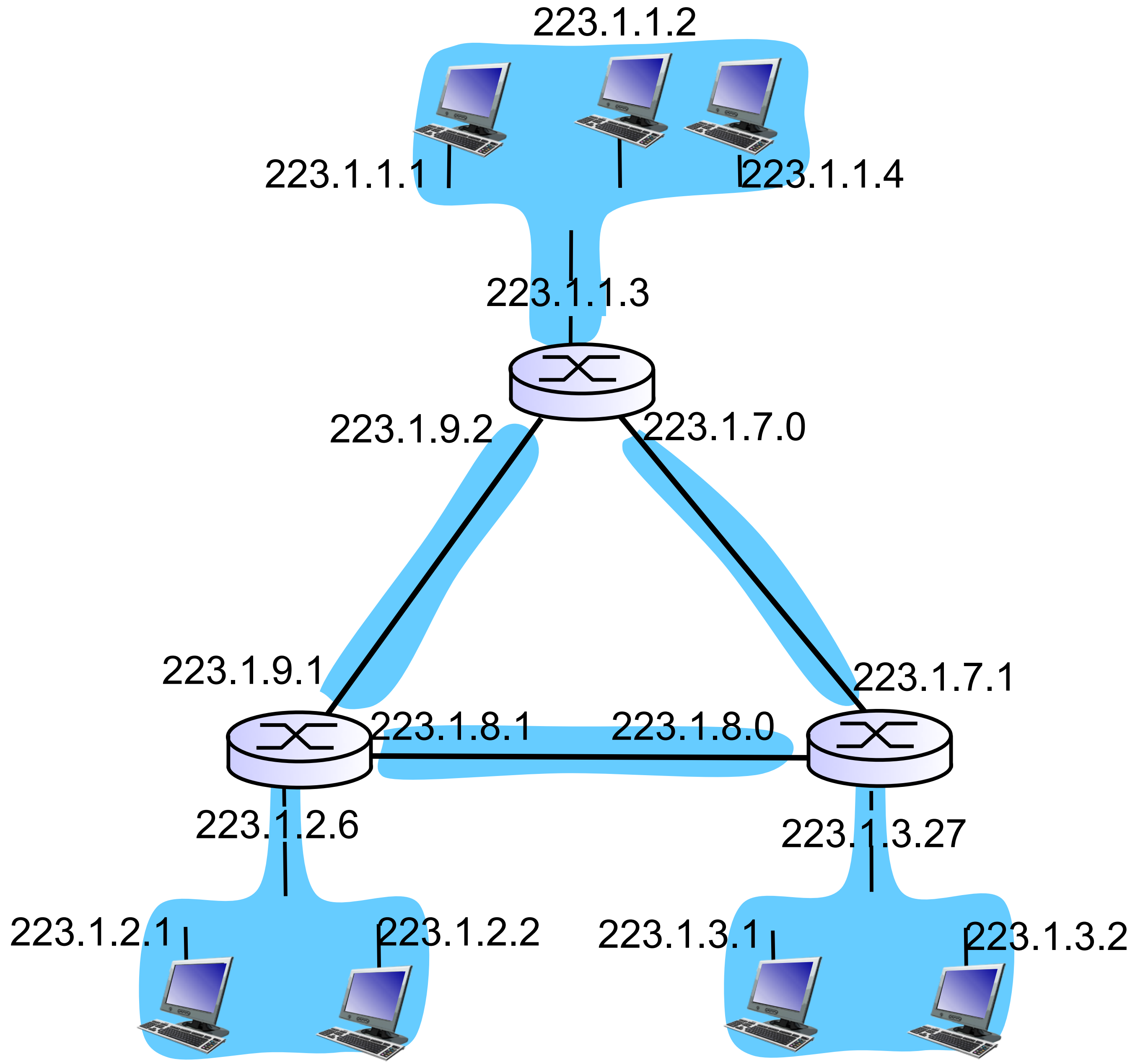
- IP address:
  - subnet part - high order bits
  - host part - low order bits
- *what's a subnet ?*
  - device interfaces with same "subnet part" of IP address
  - can physically reach each other *without intervening router*



network consisting of 3 subnets

Subnets

how many?



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

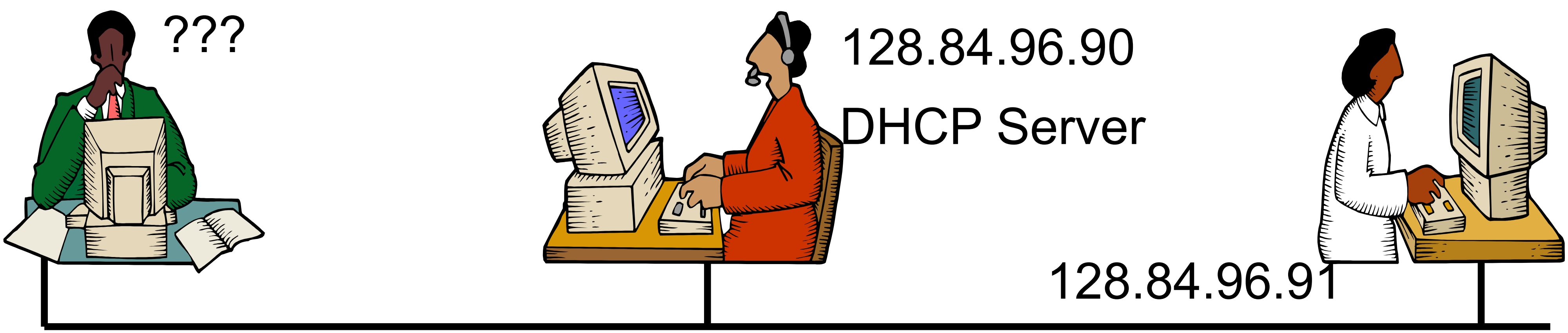
200.23.16.0/23

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"

Addressing & DHCP



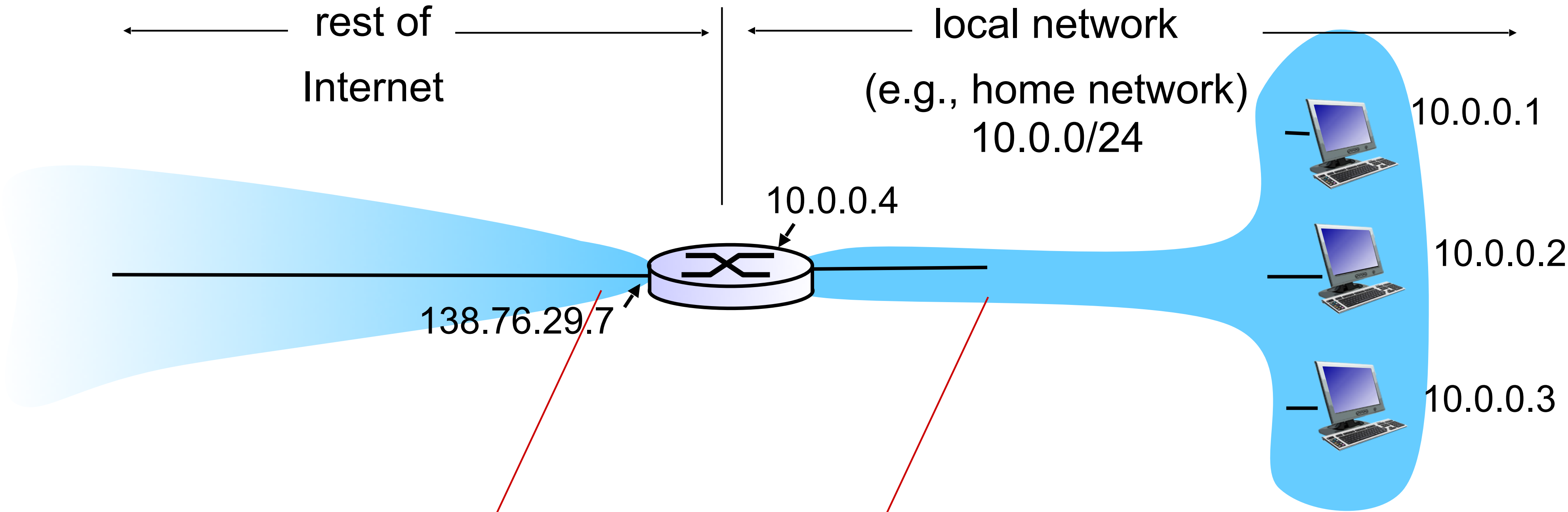
"I just got here. My physical address is 1a:34:2c:9a:de:cc. What's my IP?"

"Your IP is 128.84.96.89 for the next 24 hours"

DHCP is used to discover IP addresses (and more)

DHCP = Dynamic Host Configuration Protocol

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:

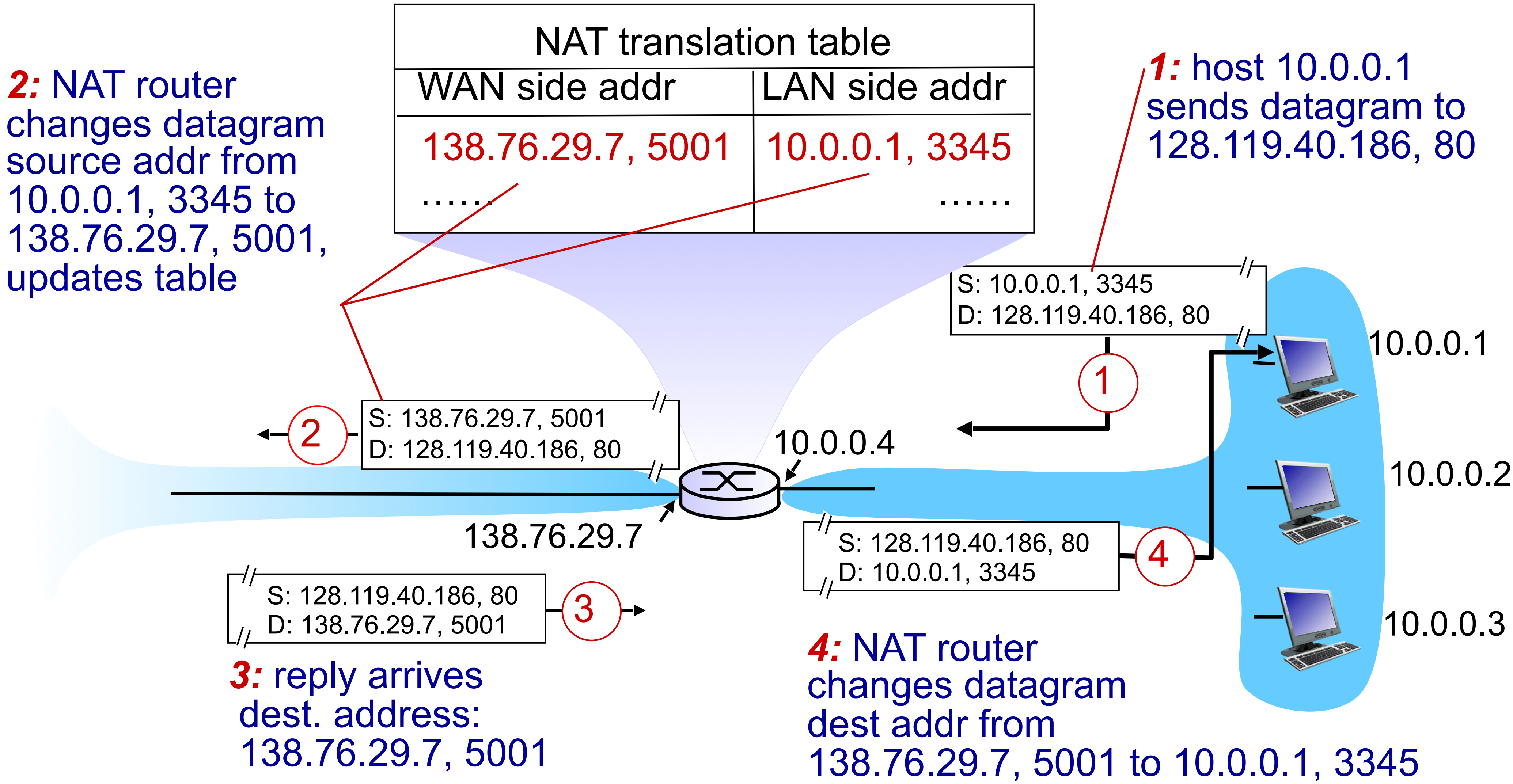
- range of addresses not needed from ISP: just one IP address for all devices
- can change addresses of devices in local network without notifying outside world
- can change ISP without changing addresses of devices in local network
- devices inside local net not explicitly addressable, visible by outside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

NAT: network address translation

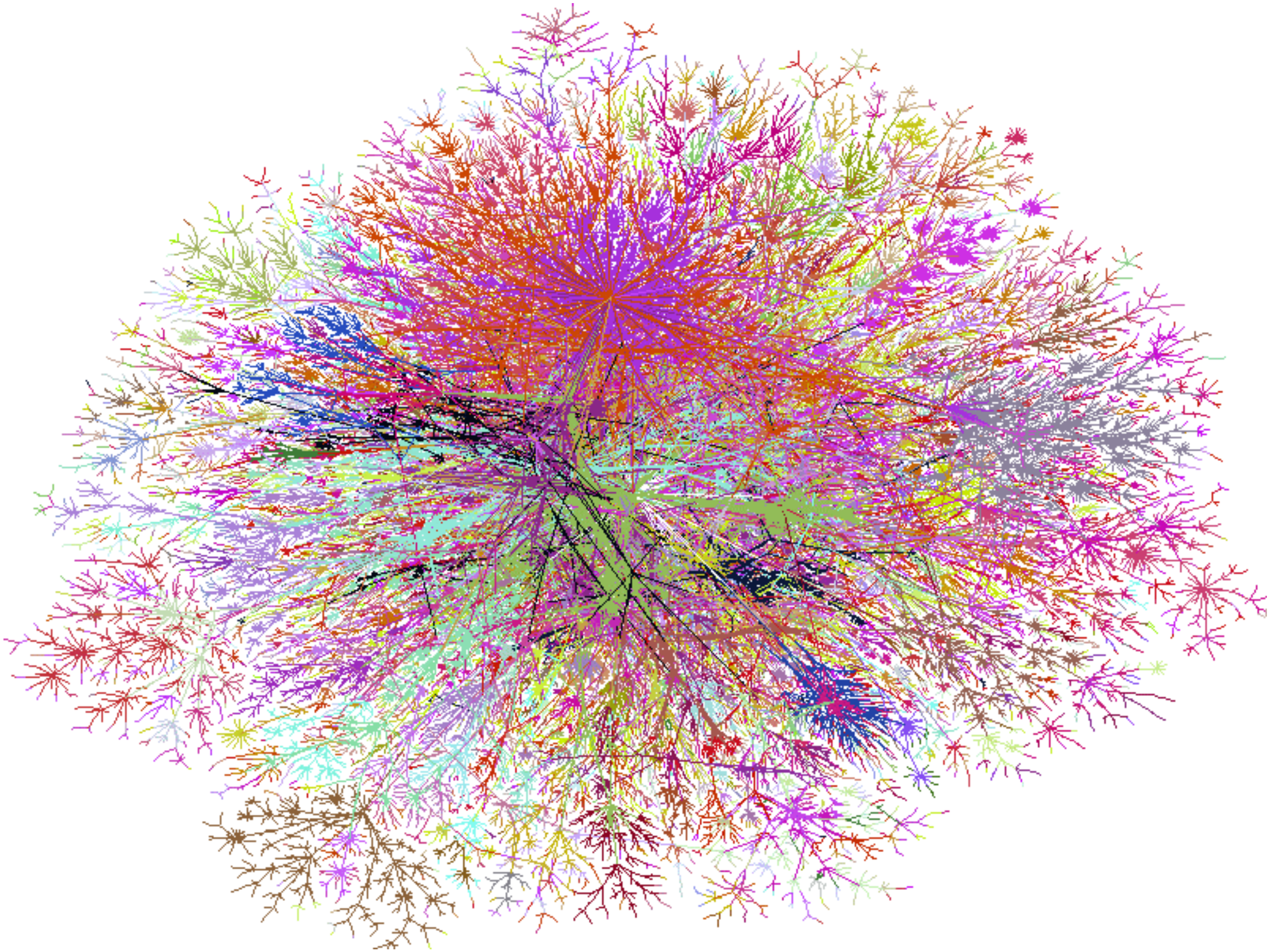


The NAT controversy

- 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
  - violates end-to-end argument
    - NAT possibility must be taken into account by app designers, e.g., P2P applications
  - address shortage should instead be solved by IPv6

Routing

The Internet is Big…



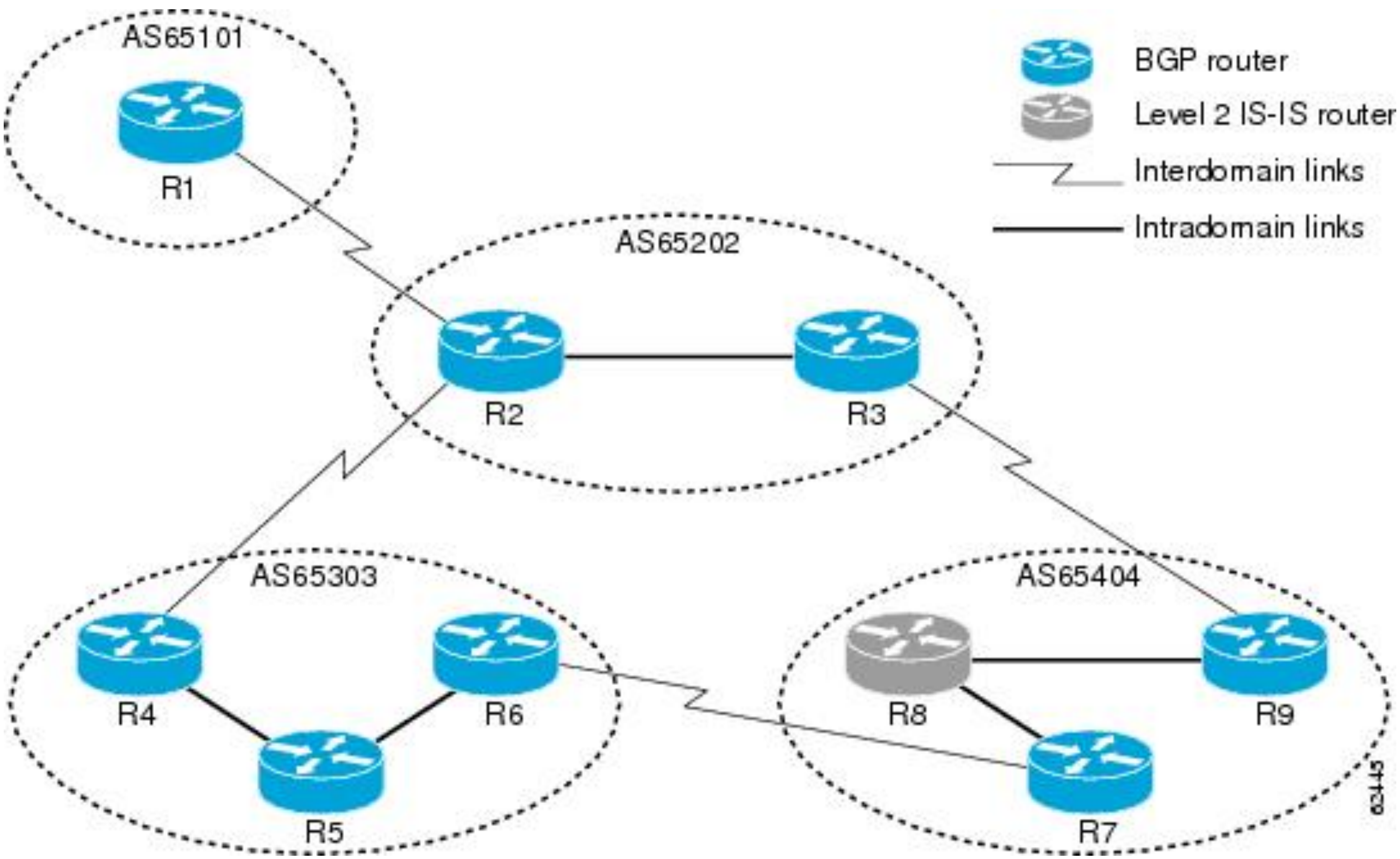
Routing

- How do we route messages from one machine to another?
- Subject to
  - churn
  - efficiency
  - reliability
  - economical considerations
  - political considerations

Internet Protocol (IP)

- The Internet is subdivided into disjoint Autonomous Systems (AS)

Graph of subgraphs



Autonomous Systems

- ASs are organized in a graph
- routing between ASs using BGP (Border Gateway Protocol) Each AS is a routing domain in its own right
  - has a private IP network
  - runs its own routing protocols
  - may have multiple IP subnets
    - each with their own IP prefix
  - has a unique "AS number"

Thus routing is hierarchical!

Three steps:

1. A packet is first routed to an "edge router" (often called "gateway") at the source AS---using the internal routing protocol used by the source AS
2. Next the packet is routed to an edge router at the destination AS---determined by the destination address prefix---using BGP
3. The AS's edge router then forwards the packet to its ultimate destination---determined by the address suffix---using the internal routing protocol used by the destination AS

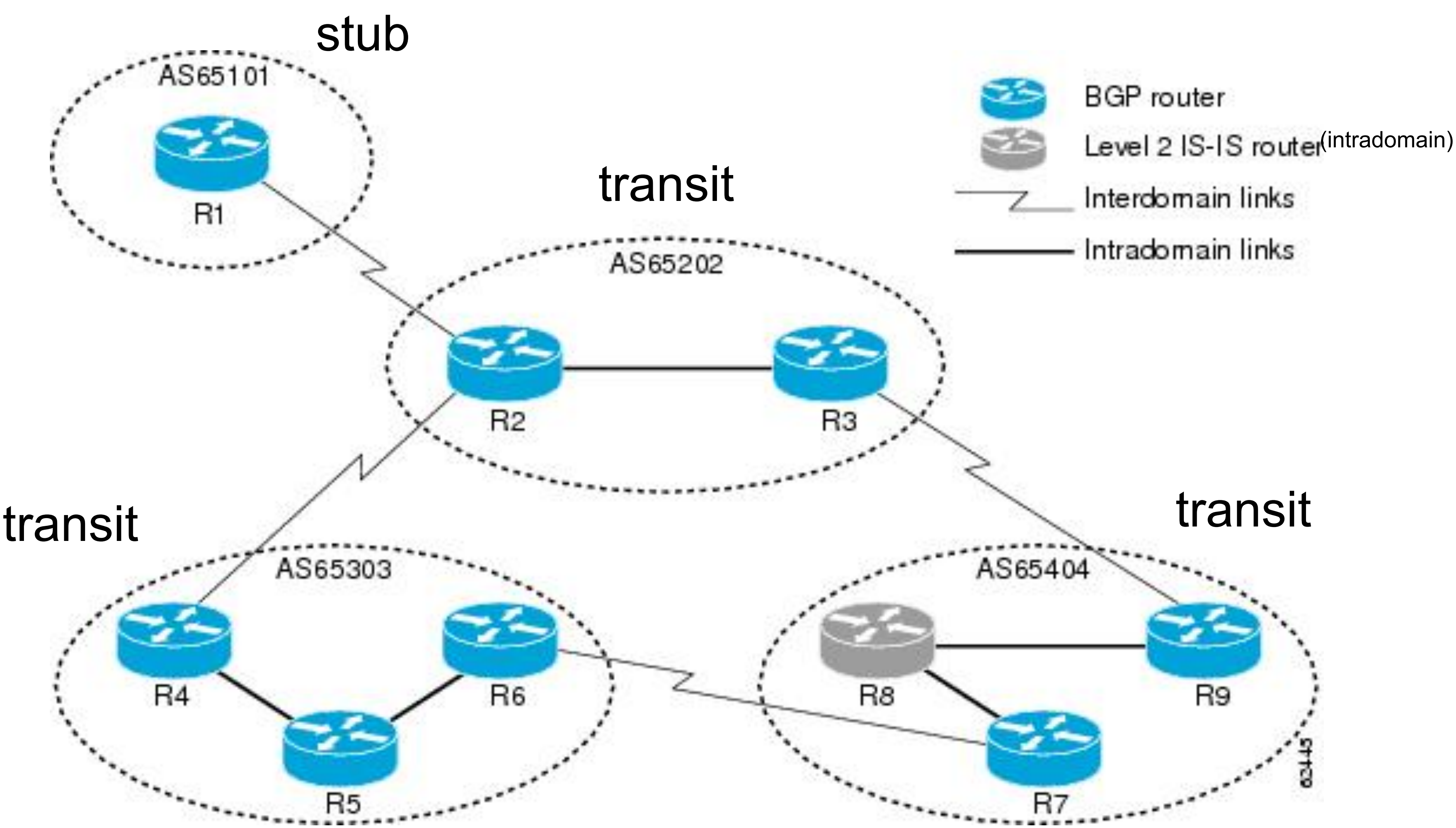
Routers (Layer-3 Switches)

- Connects multiple LANs (subnets)
- Two classes:
  - Edge or Border router: Resides at the edge of an AS, and has two faces
    - one faces outside to connect to one or more per edge router in other ASs
    - one faces inside, connecting to zero or more other routers within the same AS
  - Interior router:
    - has no connections to routers in other ASs

Internet Routing, observations

- There are no special “government” routers that route between ASs. Instead, each AS has one or more “edge routers” that are connected by interdomain links.
- Two types:
  - **Transit AS**: forwards packets coming from one AS to another AS
  - **Stub AS**: has only links to ASs higher in the hierarchy and does not do any forwarding

Transit ASs


stub
AS65101
R1
transit
AS65202
R2
R3
transit
AS65303
R4
R5
R6
transit
AS65404
R8
R9
R7
62445
BGP router
Level 2 IS-IS router(intradomain)
Interdomain links
Intradomain links

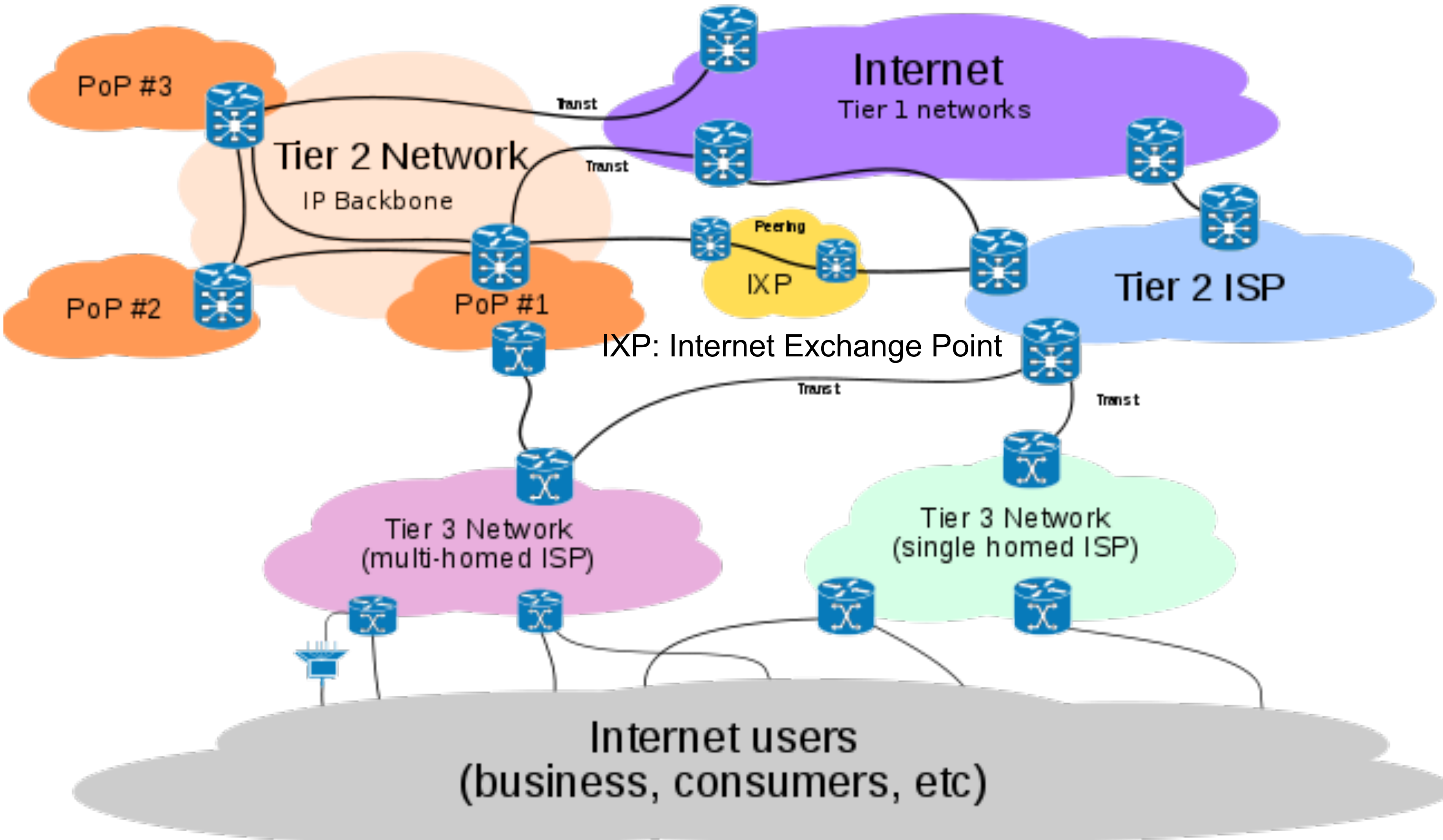
What's an ISP?

- An ISP (Internet Service Provider) is simply an AS (or collection of ASs) that provides, to its customers (which may be people or other ASs), access to the "The Internet"
- Provides one or more PoPs (Points of Presence) where its customers can connect.

AS Tiers

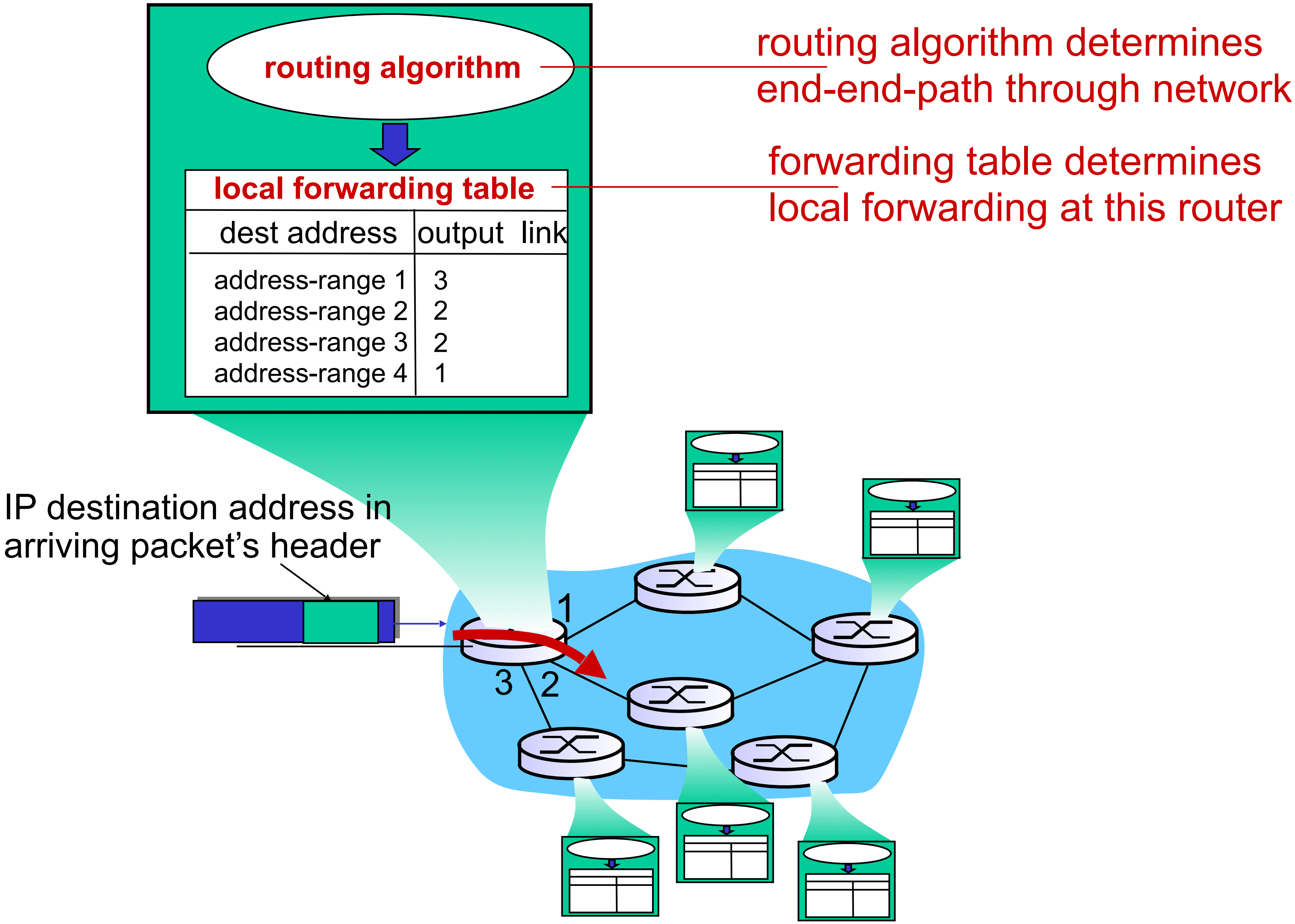
- Tier-1
  - no "upstream peers"
  - instead, peers with every other Tier-1 AS
  - "default-free" routing
  - "settlement-free connections"
- Tier-3
  - a stub, connecting to one or more upstream ISPs
  - connects consumers to the Internet
- Tier-2
  - everything in between, i.e., transit ASs that have upstream ASs, default routes, etc.

Tiers



PoP #3
Tier 2 Network
IP Backbone
Transit
Internet
Tier 1 networks
Transit
Peering
PoP #2
PoP #1
IXP
Tier 2 ISP
IXP: Internet Exchange Point
Transit
Transit
Tier 3 Network
(multi-homed ISP)
Tier 3 Network
(single homed ISP)
Internet users
(business, consumers, etc)

Interplay between routing, forwarding



Model for Routing

- A graph G(V,E), where vertices represent routers, edges represent available links
  - For now, assume a unity weight associated with each link
- Centralized "link state" algorithms for finding suitable routes are straightforward
  - e.g., Dijkstra's shortest path algorithm
- Need distributed algorithms
  - Distance vector algorithm

Distance vector algorithm

- $D_x(y)$ = estimate of least cost from x to y
  - x maintains distance vector $\vec{D}_x = [D_x(y) : y \in N]$
- node x:
  - knows cost to each neighbor v: $c(x,v)$
  - maintains its neighbors' distance vectors. For each neighbor v, x maintains

    $\vec{D}_v = [D_v(y) : y \in N]$

Distance vector algorithm

key idea:

- from time-to-time, each node sends its own distance vector estimate to neighbors
- when x receives new DV estimate from neighbor, it updates its own DV using B-F equation:

$$D_x(y) \leftarrow min_v\{c(x,v) + D_v(y)\} \textit{ for each } y \in N$$

- under minor, natural conditions, the estimate $D_x(y)$ *converge to the actual least cost* $d_x(y)$

Distance vector algorithm

iterative, asynchronous: each local iteration caused by:

- local link cost change
- DV update message from neighbor

distributed:

- each node notifies neighbors *only* when its DV changes
  - neighbors then notify their neighbors if necessary

each node:

wait for (change in local link cost or msg from neighbor)

recompute estimates

if DV to any dest has changed, *notify* neighbors

Routing Loops?

- In steady state, there should be no routing loops
- But steady state is rare. If routing tables are not in sync, routing loops can occur.
- To avoid problems, IP packets maintain a maximum hop count (TTL) that is decreased on every hop until 0 is reached, at which point a packet is dropped.

Most Common Example

- BGP (Border Gateway Protocol)
  - but instead of shortest path, uses various other considerations to select which route is best!
- Used as the most common interdomain routing protocol or "Exterior Gateway Protocol", but is also used in ASs for intradomain or "Interior Gateway" routing.

Why BGP?

- Shortest path algorithms insufficient to handle myriad of operational (e.g., loop handling), economic, and political considerations
- Policy categories (Caesar and Rexford):
  - business relationships
  - traffic engineering
  - scalability (improving stability, aggregation, etc.)
  - security

BGP Policy Implementation

- policies at a router control
  - import policy: which routes (advertised by peers) are accepted
  - decision process: which routes are used
  - export policy: which routes are advertised to peers
- policies sometimes need to be negotiated and implemented across multiple ISPs
  - BGP allows advertised routes to be tagged with policies using the "community" attribute

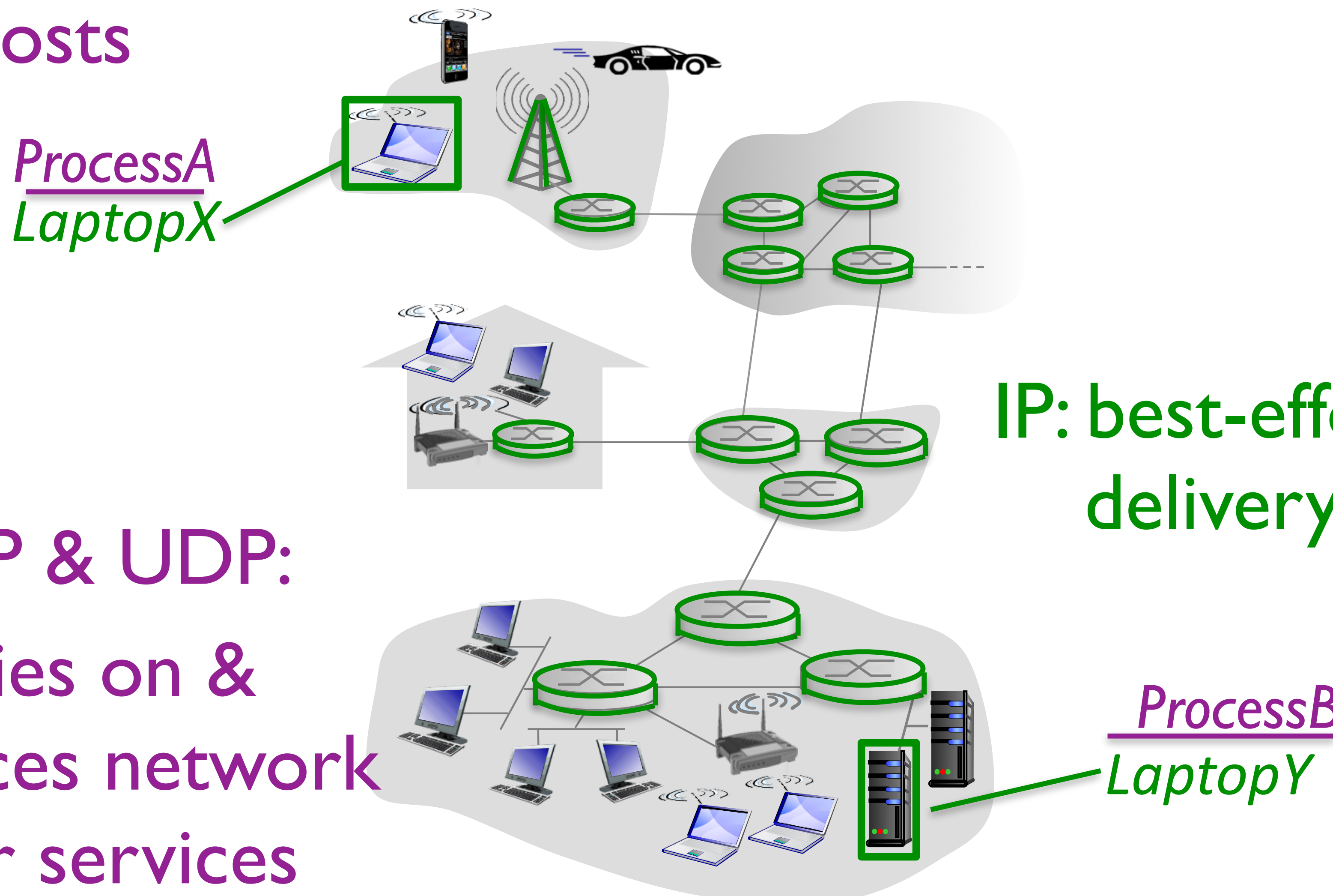
Application Layer
Transport Layer
Network Layer
Link Layer
Physical Layer

Transport Layer

Transport Layer vs. Network Layer

Logical communication between *processes* on hosts

Logical communication between *hosts*

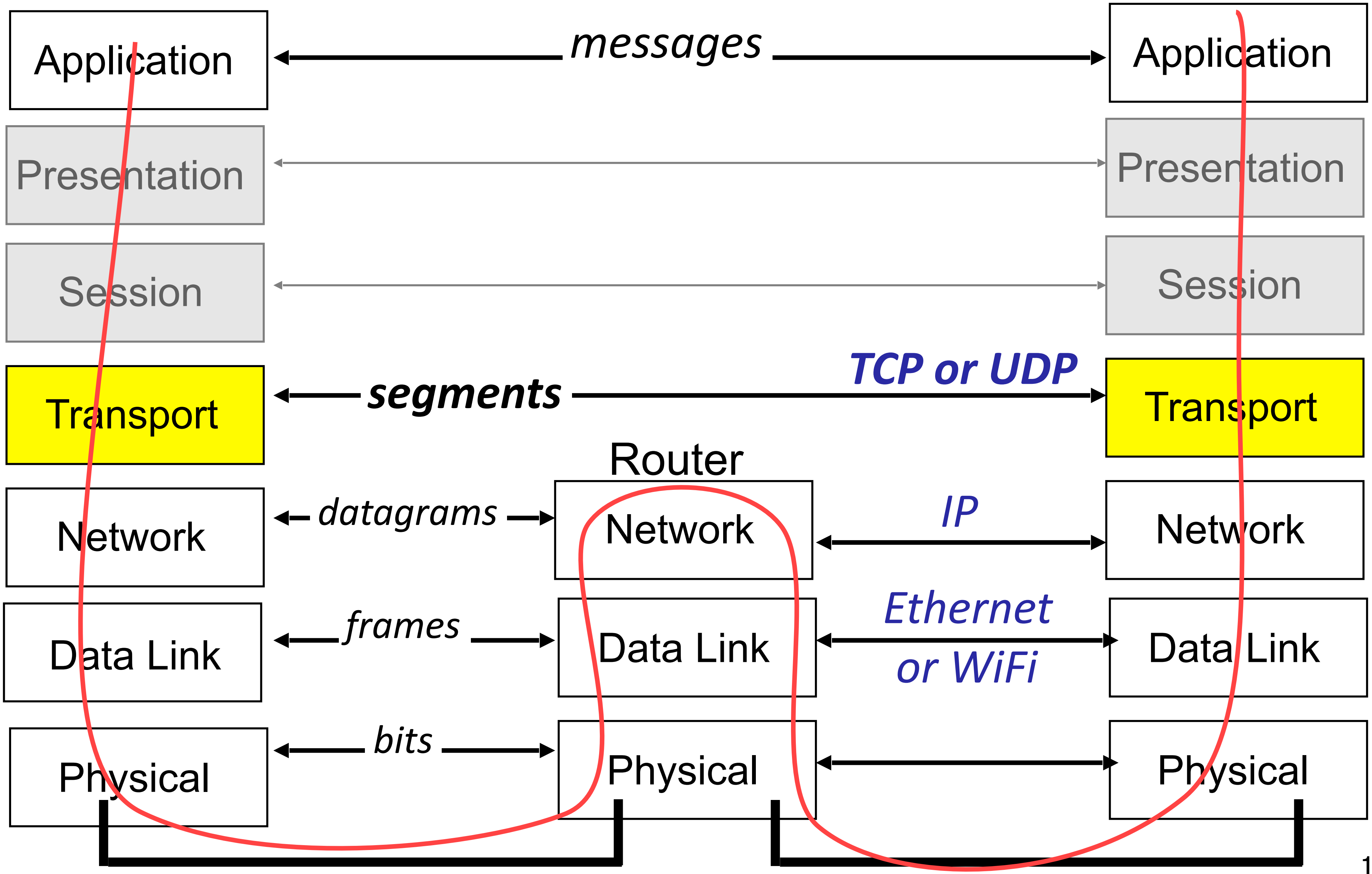


IP: best-effort delivery

TCP & UDP:

relies on & enhances network layer services

The Big Picture



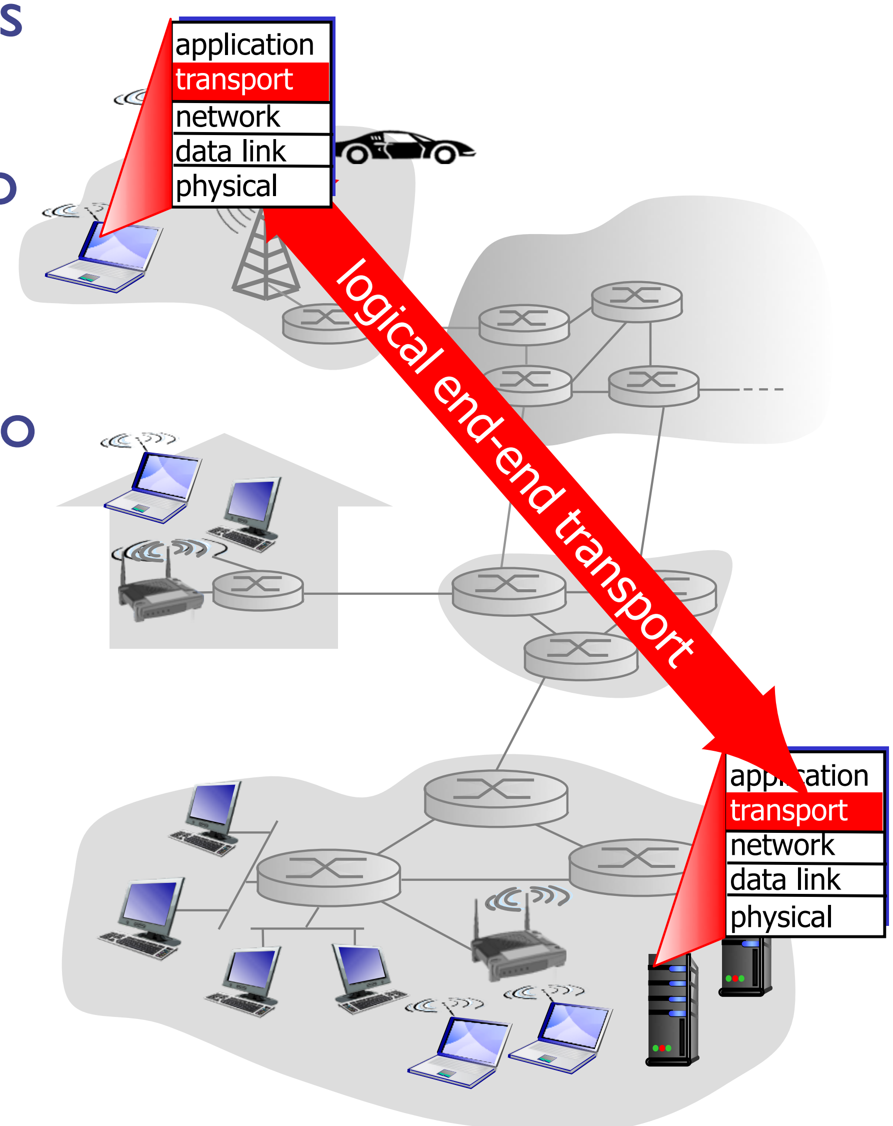
Transport services and protocols

Transport protocols run in end systems

- **sender side:** breaks app messages into *segments*, passes to network layer
- **receiver side:** reassembles segments into messages, passes to app layer

More than one transport protocol available to apps

- Internet: TCP and UDP



Transport Layer Analogy

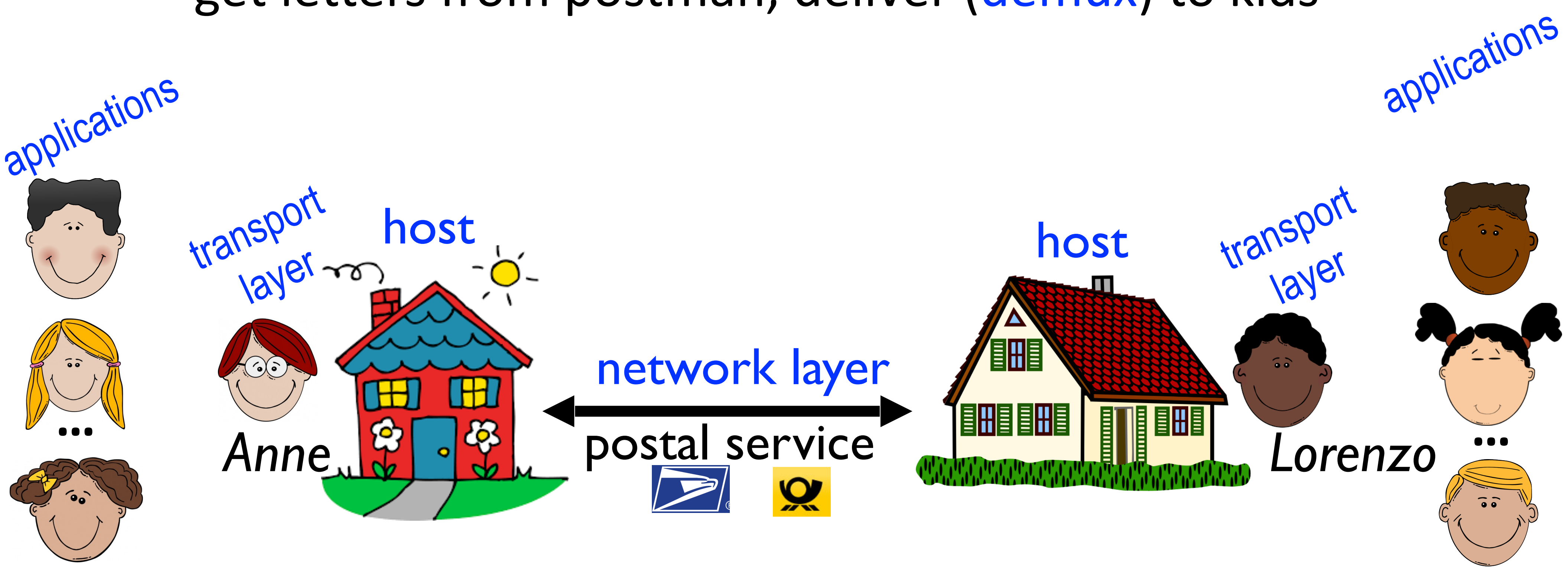
2 houses (hosts), each has 12 kid siblings

Kids: (applications)

- write letters (messages) to cousins

Parents: (transport layer protocol)

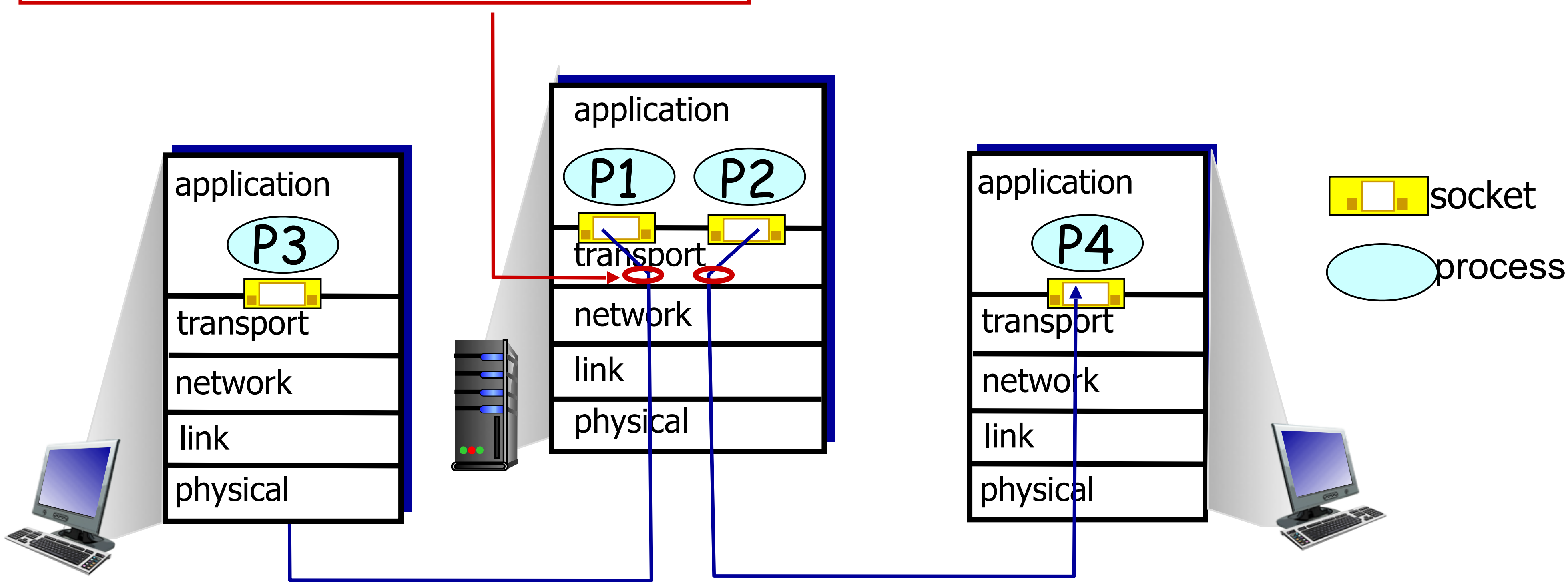
- gather the letters (multiplexing)
- put them in addressed envelopes (segments)
- give them to the postman (network layer)
- get letters from postman, deliver (demux) to kids



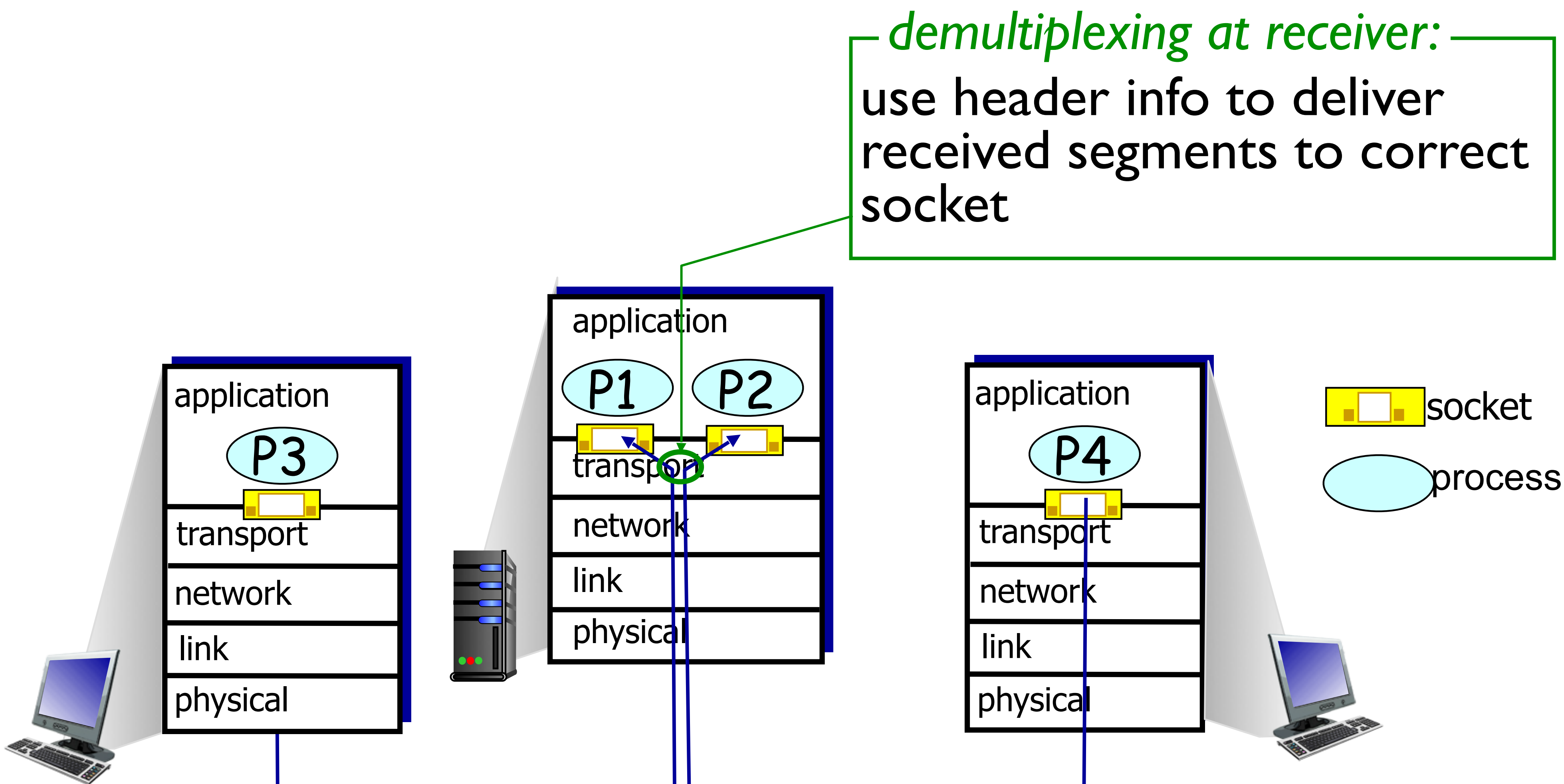
Multiplexing

multiplexing at sender:

handle data from multiple sockets, add transport header (later used for demultiplexing)



Demultiplexing

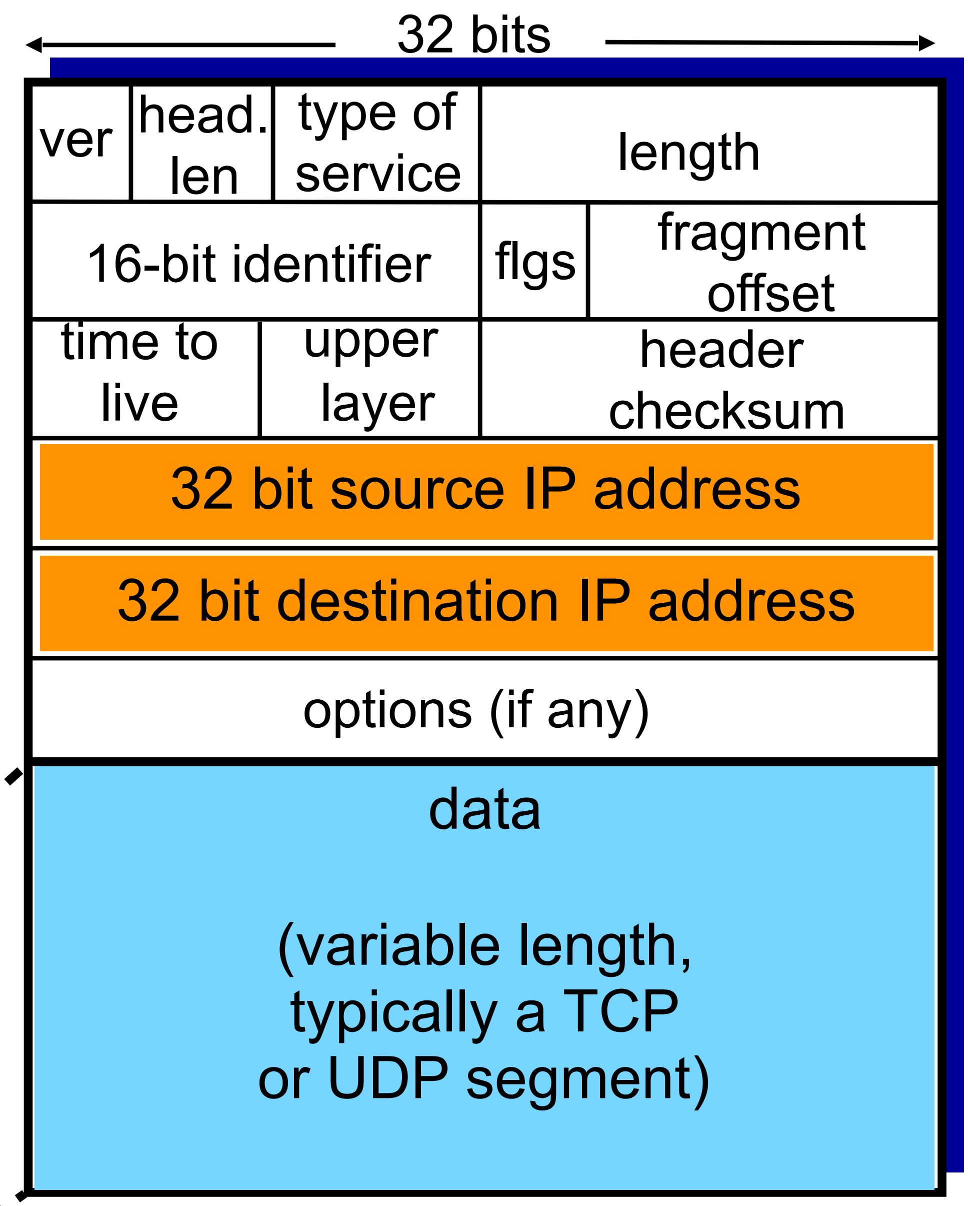


(Datagram(Segment(Message)))

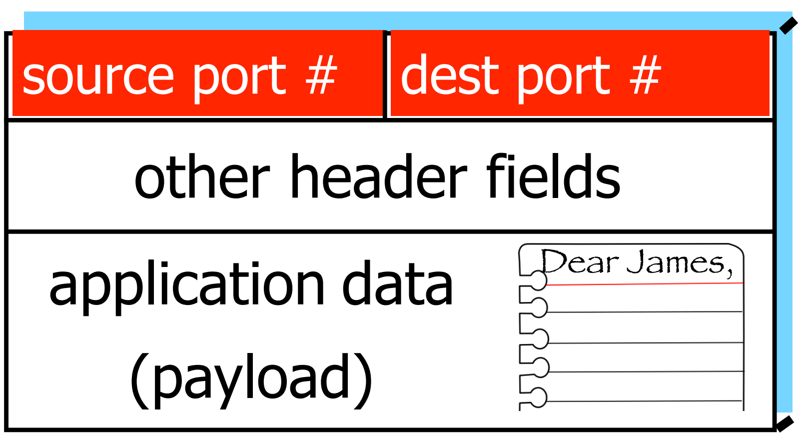
Host gets IP Datagram with:

- source & dest IP addresses
- one transport-layer segment
  - inside: source & dest port #

Host uses *IP addr* & *port #s* to direct segment to appropriate socket



IP Datagram



TCP/UDP segment format

Internet transport-layer protocols

Transport Control Protocol (TCP)

“Trusty Connection Protocol” ??

- reliable, in-order delivery
- congestion control
- flow control
- connection setup

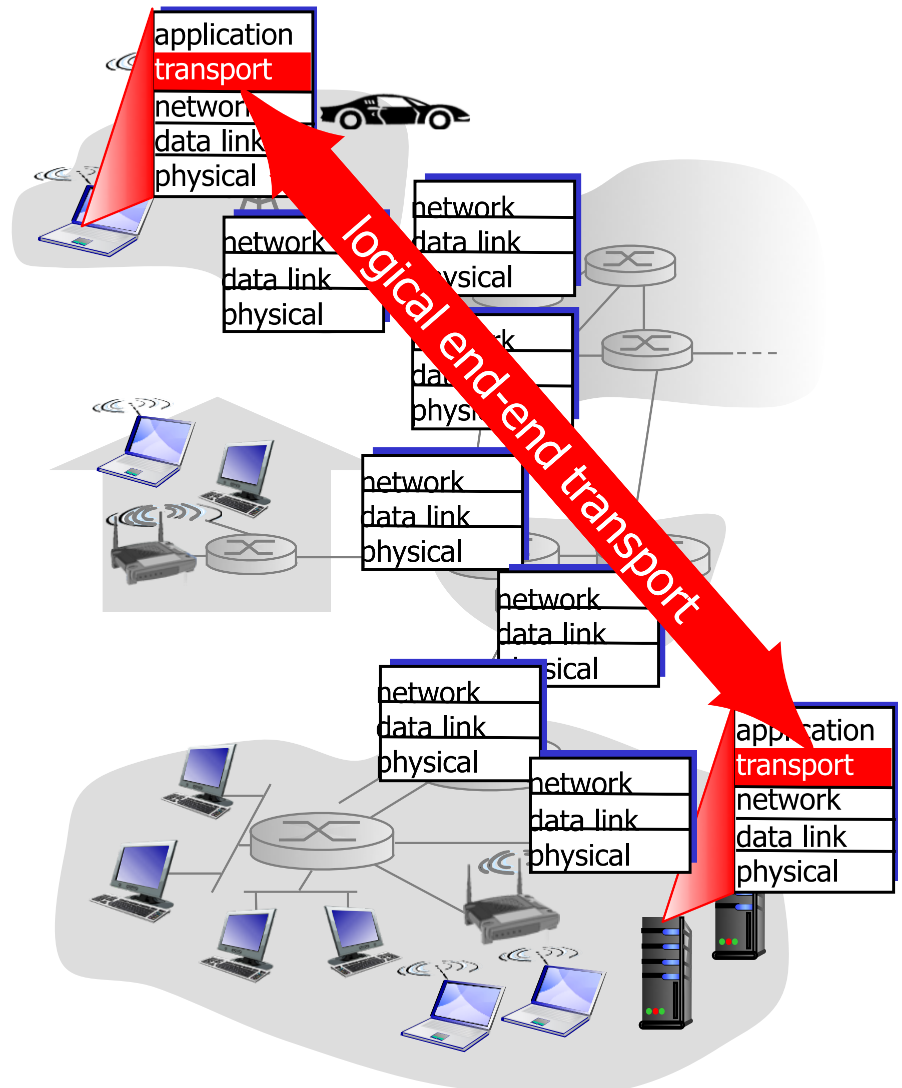
User Datagram Protocol (UDP)

“Unreliable Datagram Protocol”

- unreliable, unordered delivery
- no-frills extension of “best-effort” IP

Services not available:

- delay guarantees
- bandwidth guarantees



UDP: User Datagram Protocol [RFC 768]

- "no frills," "bare bones" Internet transport protocol
- "best effort" service, UDP segments may be:
  - lost
  - delivered out-of-order, duplicated to app
- connectionless:
  - no handshaking between UDP sender, receiver
  - each UDP segment handled independently of others
- reliable transfer still possible:
  - add reliability at application layer
  - application-specific error recovery!

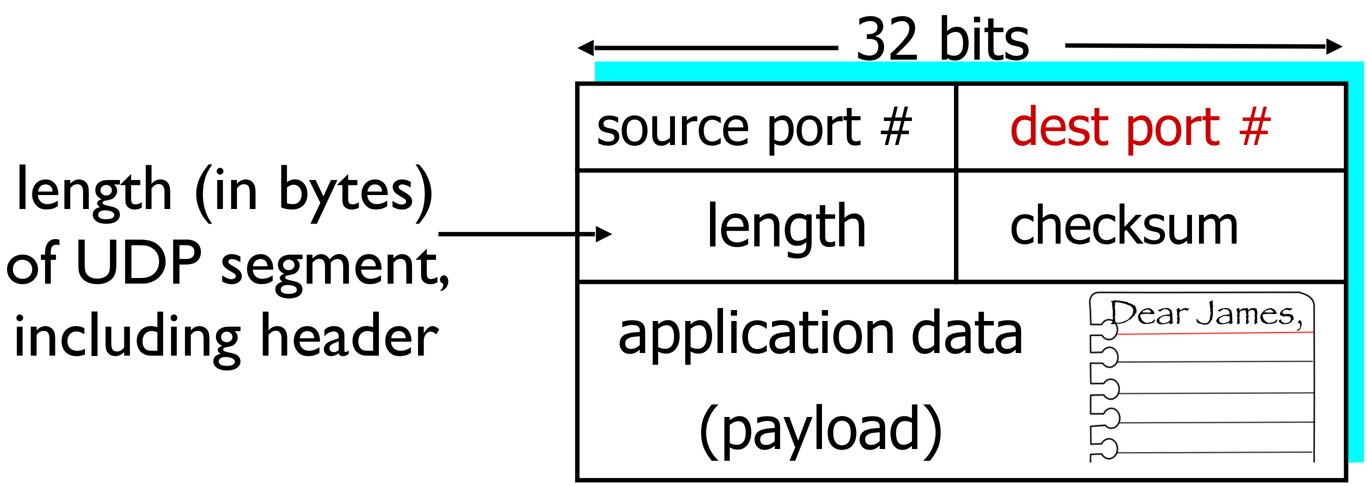
I was gonna tell you guys a joke about UDP…
But you might not get it

I was you guys about UDP might not

Connectionless demultiplexing

Host receives UDP segment:

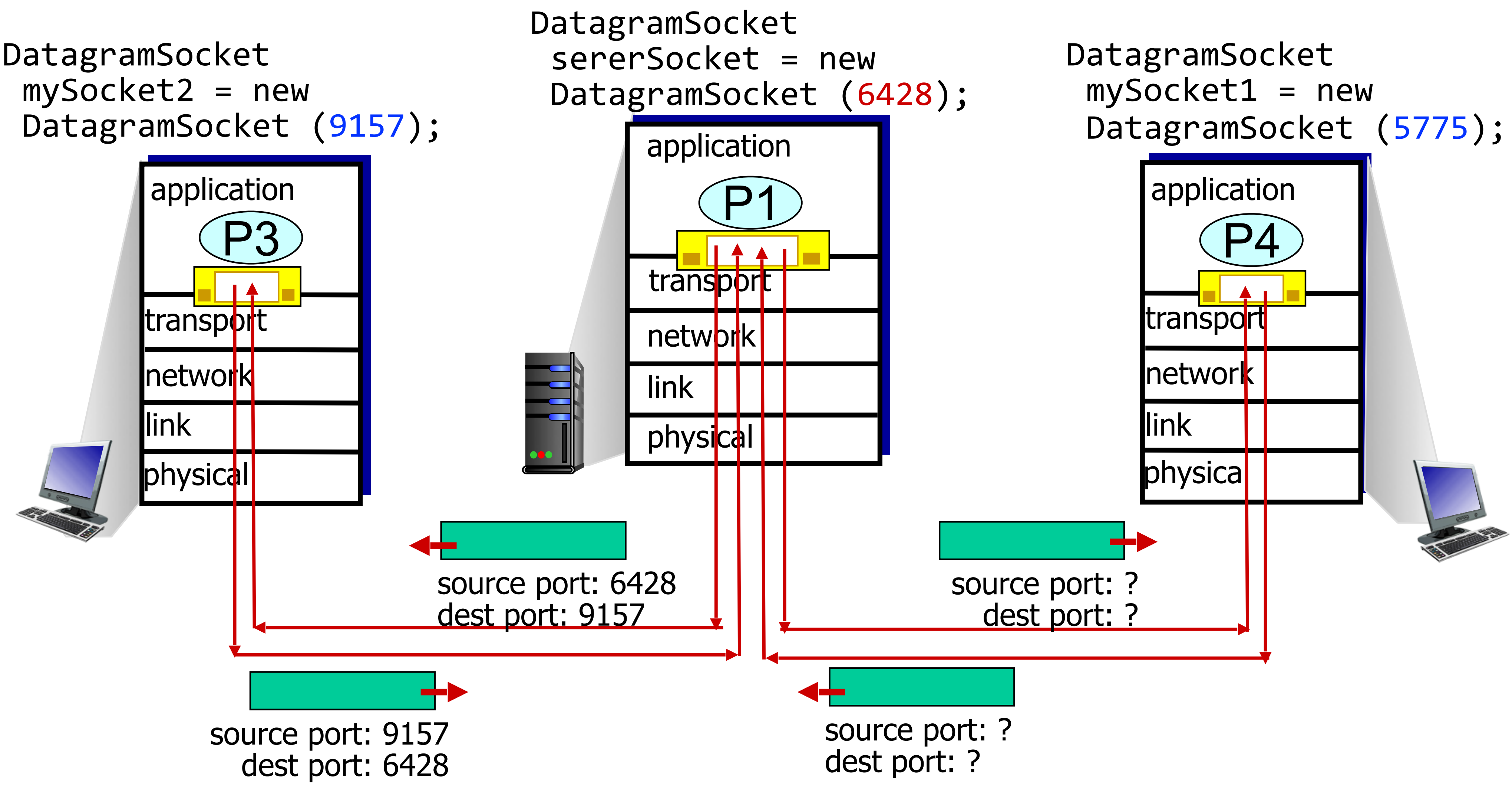
- checks destination port # in segment
- directs UDP segment to socket with that port #



UDP segment format

Connectionless demux: example

IP datagrams w/ *same dest port #,* but *different source IP addr or port #s*
→ directed to *same socket* at dest



Is there anything good about UDP?

Speed:

- no connection establishment (which can add delay)
- no congestion control: UDP can blast away as fast as desired

Simplicity:

- no connection state at sender, receiver
- small header size

Target Users:

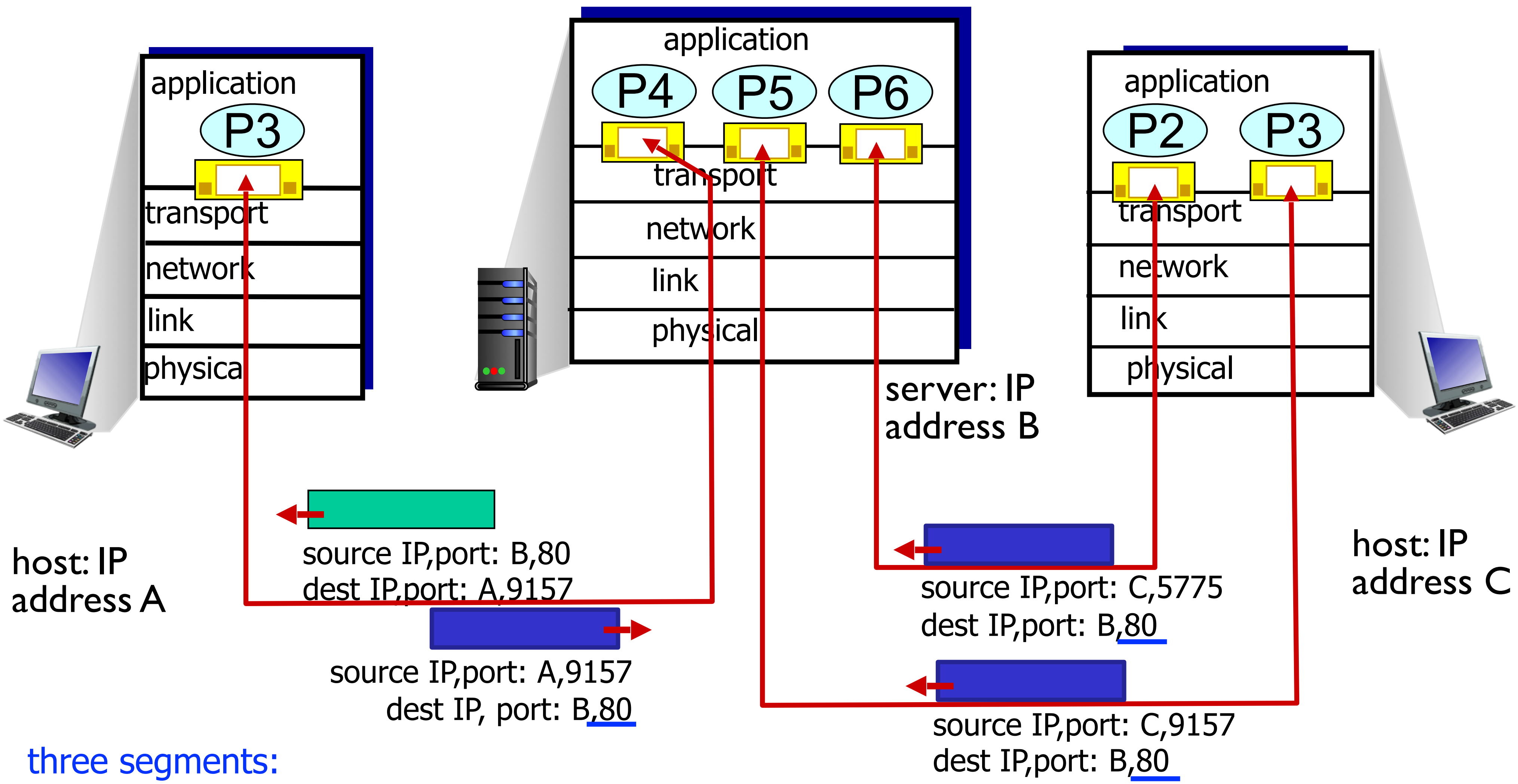
- streaming multimedia apps (loss tolerant, rate sensitive)
- DNS

Connection-oriented demux

- TCP socket identified by 4-tuple:
  - source IP address
  - source port number
  - dest IP address
  - dest port number
- demux: receiver uses all four values to direct segment to appropriate socket

- server host may support many simultaneous TCP sockets:
  - each socket identified by its own 4-tuple
- web servers have different sockets for each connecting client
  - non-persistent HTTP will have different socket for each request

Connection-oriented demux: example

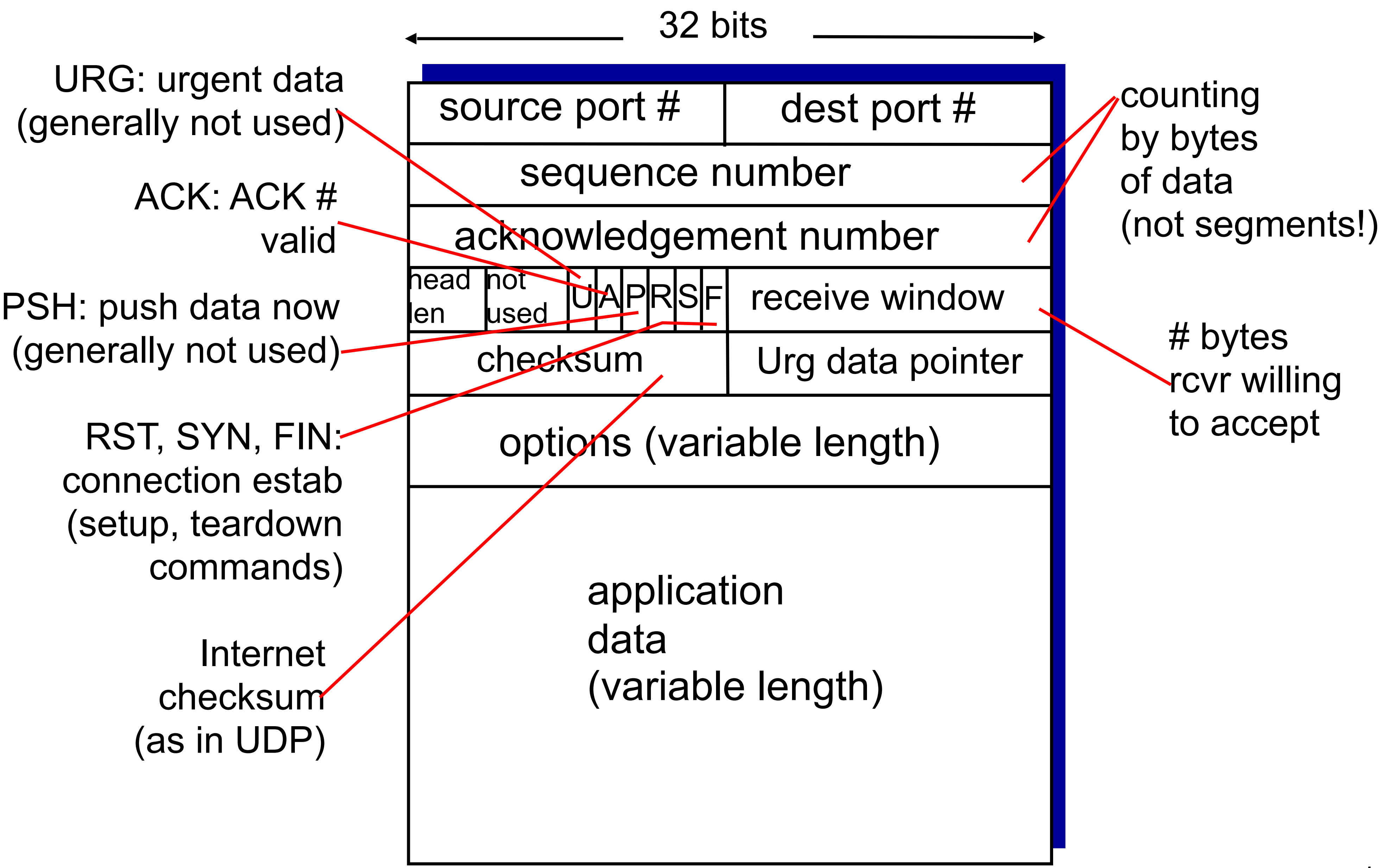


three segments:
- all destined to IP address: B, dest port: 80
- demultiplexed to different sockets

TCP: Transmission Control Protocol

- Reliable, ordered, 2-way byte-stream communication
- Many applications demand reliable, ordered delivery. They should not have to implement their own protocol.
- A standard, adaptive protocol that delivers good-enough performance and deals well with congestion
  - E.g., all web traffic travels over TCP/IP

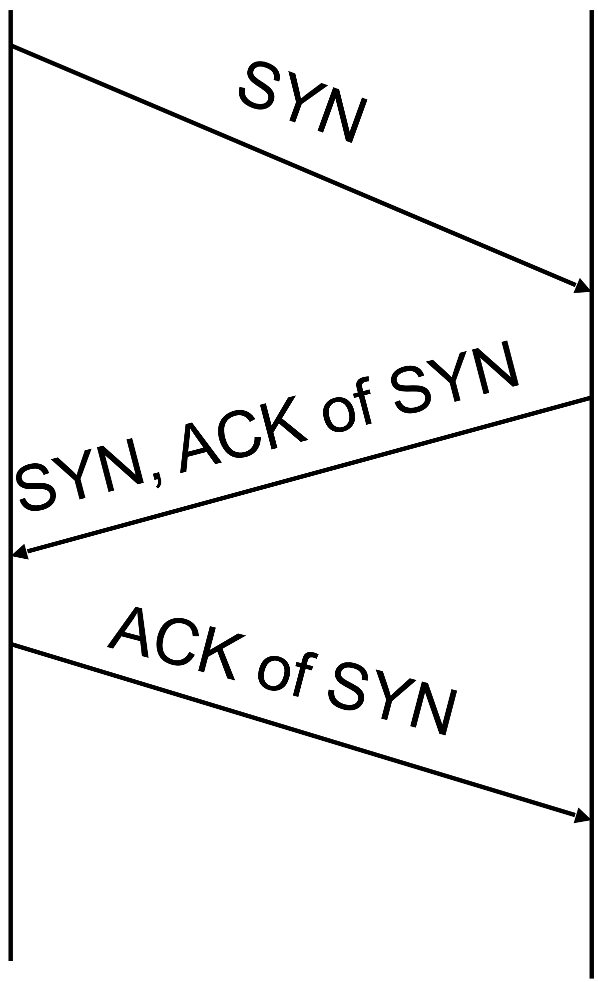
TCP segment structure



TCP Packets

- Each packet carries a sequence number
  - Initial number chosen randomly
  - Number incremented by the data length
- Each packet carries an acknowledgment
  - Can acknowledge a sequence of bytes by ack'ing latest byte received
- Reliable transport is implemented using these identifiers

TCP Connections



- TCP is *connection* oriented
- A connection is initiated with a *three-way handshake*
- Three-way handshake agrees on initial sequence numbers
- Takes 3 packets, 1.5 RTT (Round Trip Time)

SYN = Synchronize

ACK = Acknowledgement

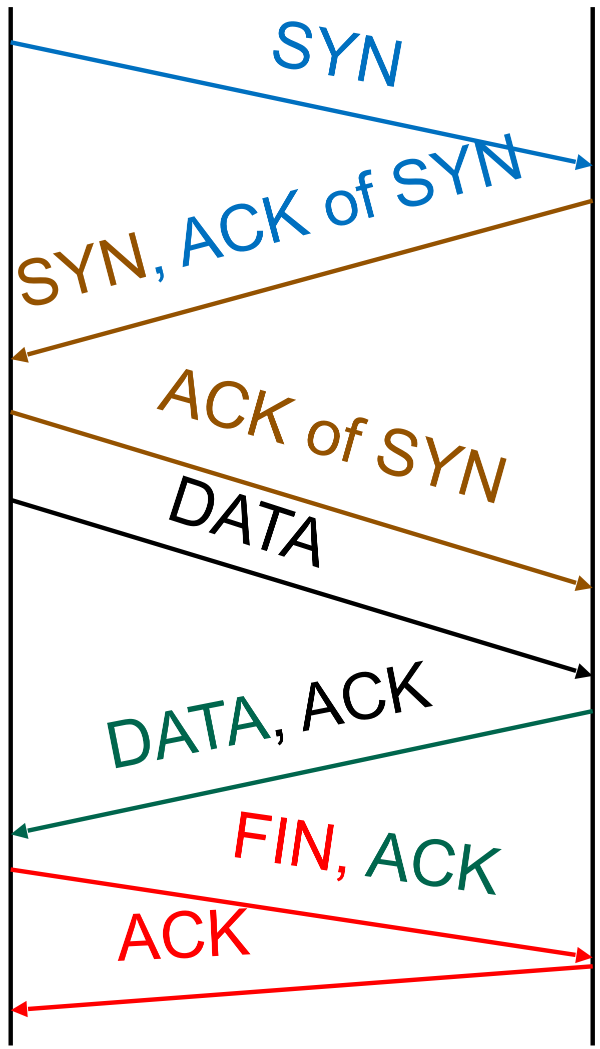
I would tell you a joke about TCP… If only to be acknowledged 😢

TCP Handshakes

The three-way handshake establishes common state on both sides of a connection

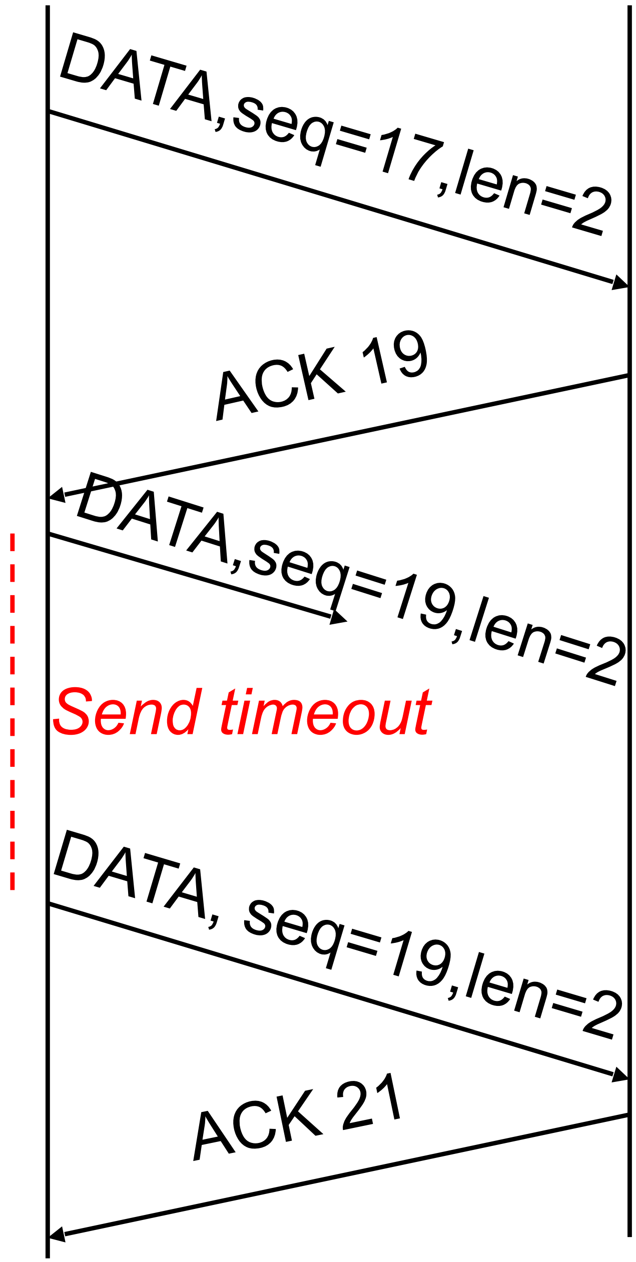
- Both sides will have seen one packet from the other side, thus know what the first seqno ought to be
- SYN-ACK also typically carries a new port for the server
- Both sides will know that the other side is ready to receive

Typical TCP Usage



- 3 round-trips to set up a connection, send a data packet, receive a response, tear down connection
- FINs work (mostly) like SYNs to tear down connection
  - Need to wait after a FIN for straggling packets

Reliable transport



- TCP keeps a copy of all sent, but unacknowledged packets
- If acknowledgment does not arrive within a "send timeout" period, packet is resent
- Send timeout adjusts to the round-trip delay
- ACKs can be *piggybacked*

Here's a joke about TCP.
Did you get it?
Did you get it?
Did you get it?
Did you get it?

TCP timeouts

What is a good timeout period ?

- Want improved throughput w/o unnecessary transmissions

$AverageRTT := (1 - \alpha)\ AverageRTT + \alpha\ LatestRTT$

$AverageVar := (1 - \beta)\ AverageVar + \beta\ LatestVar$

where $LatestRTT = (ack\_receive\_time - send\_time)$,

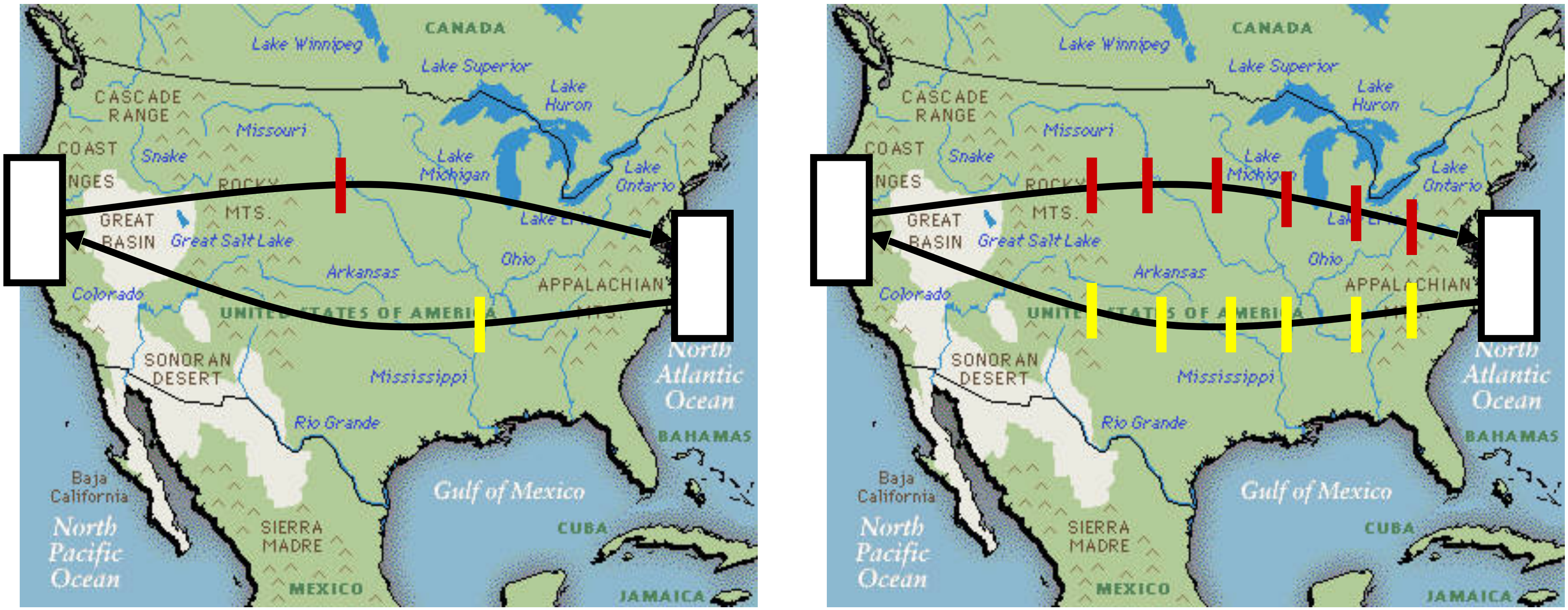
$LatestVar = |LatestRTT - AverageRTT|$,

$\alpha = 1/8$, $\beta = 1/4$ typically.

$Timeout := AverageRTT + 4*AverageVar$

→ Timeout is thus a function of RTT and variance

TCP Windows

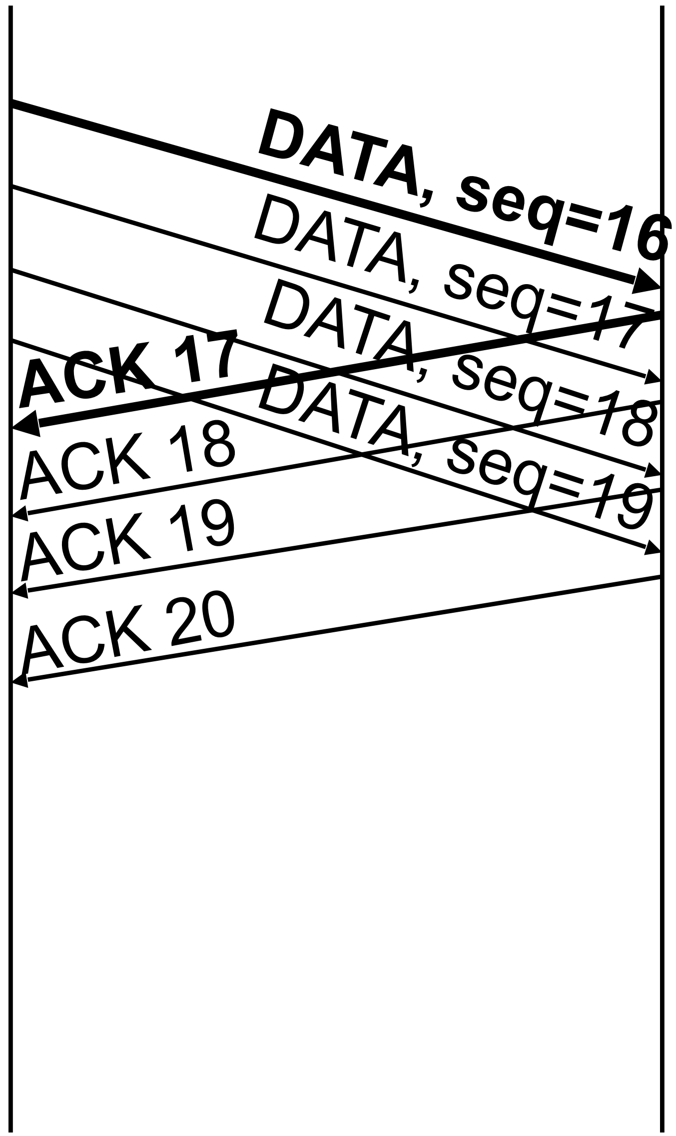


Multiple outstanding packets can increase throughput

How much data "fits" in a pipe?

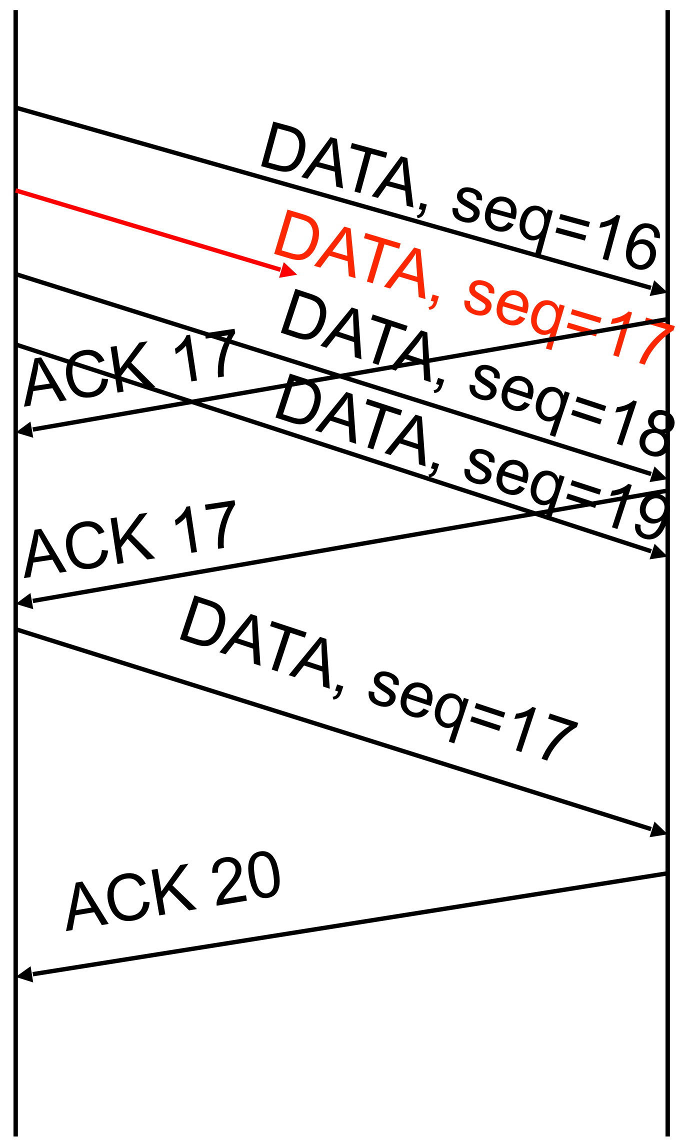
- Suppose the b/w is b bytes / second
- Suppose the RTT is r seconds
- Suppose an ACK is a small message
  - you can send b * r bytes before receiving an ACK for the first byte
- But b/w and RTT are both variable…

TCP Windows



- Can have more than one packet in transit
- Especially over fat pipes, e.g. satellite connection
- Need to keep track of all packets within the window
- Need to adjust window size

TCP Windows and Fast Retransmit

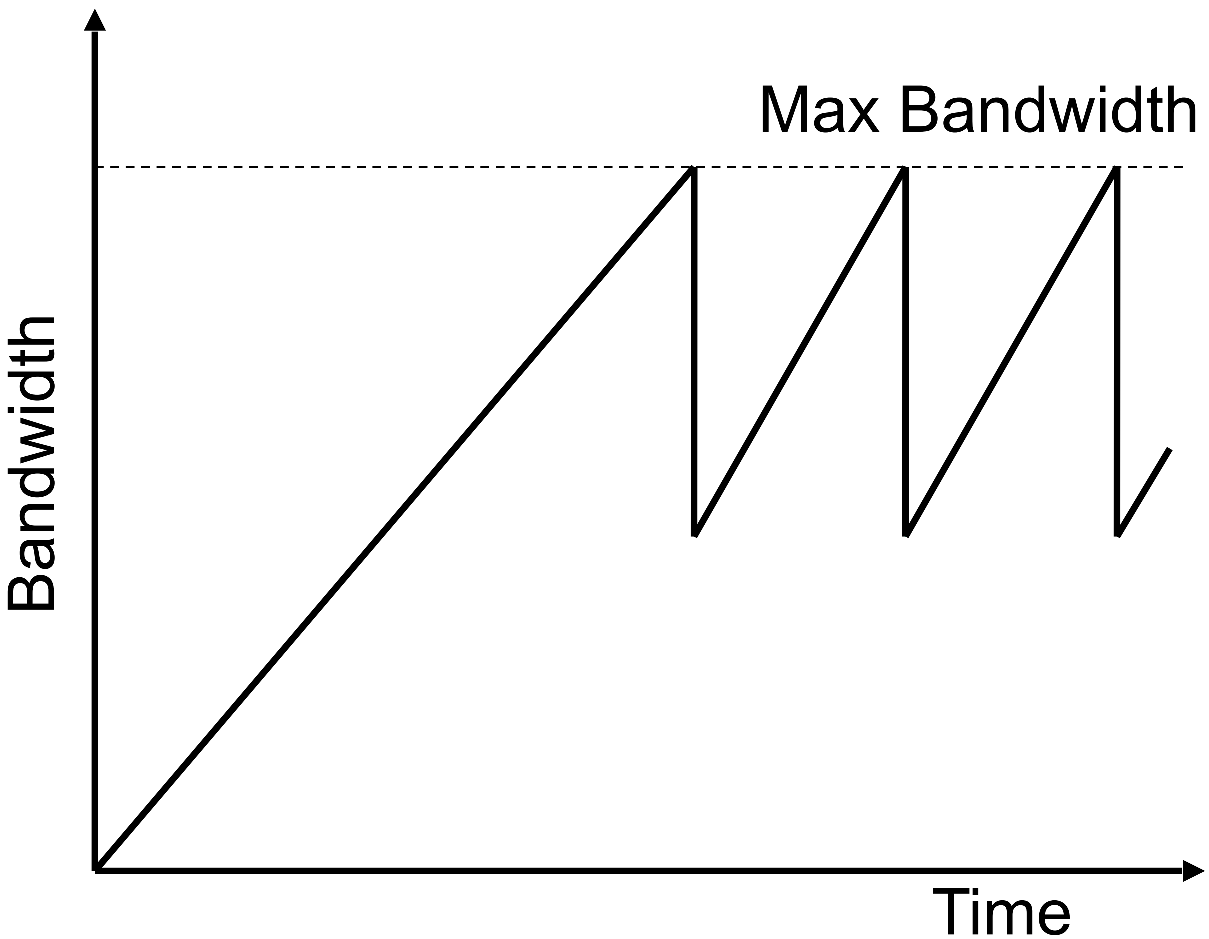


- When receiver detects a lost packet (i.e. a hole in the seqno space), it acks the last seqno it successfully received
- Sender can quickly detect that a loss occurred without waiting for a timeout

TCP Congestion Control

- TCP typically increases its window size by one MTU (Maximum Transmission Unit) every RTT
- It typically halves the window size when a packet drop occurs
  - A packet drop is evident from the acknowledgments
- Therefore, it will slowly build up to the max bandwidth, and hover around the max
  - It doesn't achieve the max possible though
  - Instead, it shares the b/w well with other TCP connections
- This linear-increase, exponential backoff in the face of congestion is termed *TCP-friendliness*

TCP Window Size



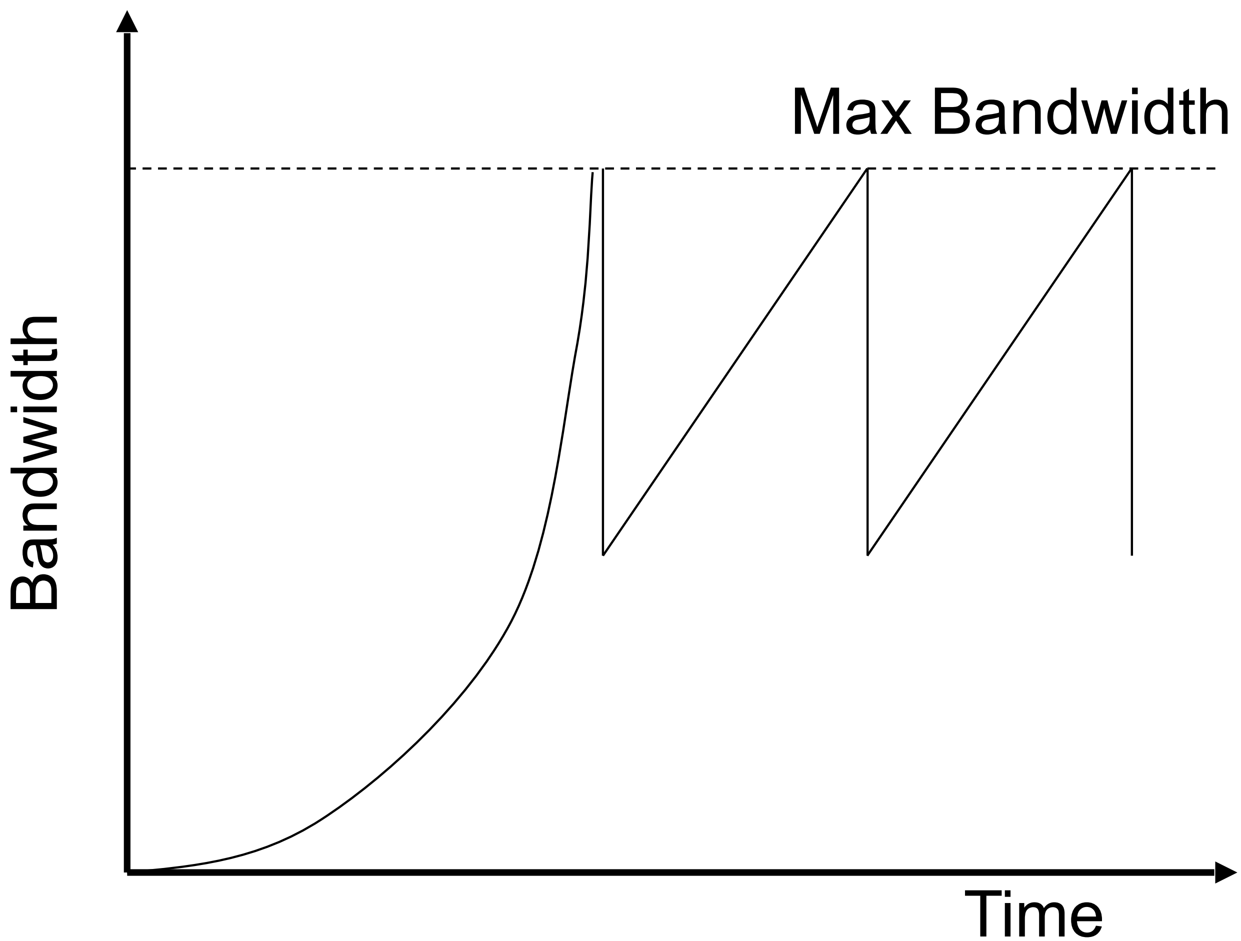
- Linear increase
- Exponential backoff

(Assumes no other losses in network except those due to b/w)

TCP Slow Start

- **Linear increase:**
  - takes a long time to build up a window size that matches the link bandwidth*delay
  - Most file transactions end before that happens
  - TCP spends a lot of time with small windows, never reaching a sufficiently large window size
- Better: **Exponential increase**
  - allow TCP to build up to a large window size initially by increasing the window size linearly for each ack received
  - Effectively doubling the window size until first loss

TCP w/ initial phase exponential



(Assumes no other losses in network except those due to b/w)

TCP Summary

- Reliable ordered message delivery
  - Connection oriented, 3-way handshake
- Transmission window for better throughput
  - Timeouts based on link parameters
- Congestion control
  - Linear increase, exponential backoff
- Fast adaptation
  - Exponential increase in the initial phase

Application Layer
Transport Layer
Network Layer
Link Layer
Physical Layer

Application Layer

DNS: domain name system

people: many identifiers:

- SSN, name, passport #

Internet hosts, routers:

- IP address (32 bit) - used for addressing datagrams
- "name", e.g., www.yahoo.com - used by humans

Q: how to map between IP address and name, and vice versa ?

Domain Name System:

- *distributed database* implemented in hierarchy of many *name servers*
- *application-layer protocol:* hosts, name servers communicate to *resolve* names (address/name translation)
  - note: core Internet function, implemented as application-layer protocol
  - complexity at network's "edge"

DNS: services, structure

DNS services

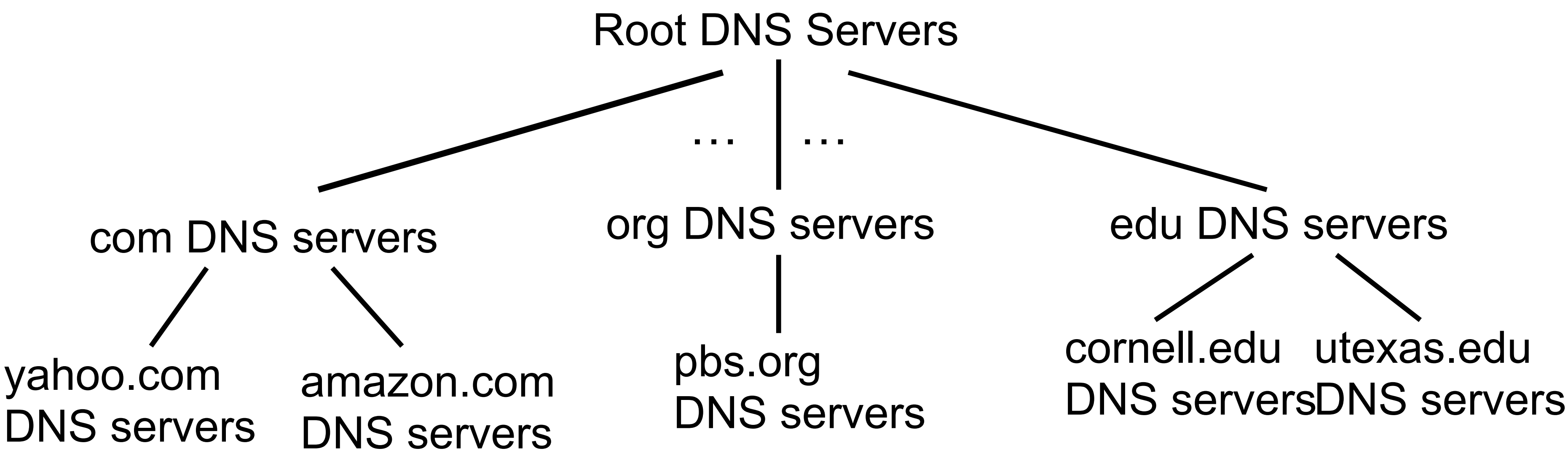
- hostname to IP address translation
- host aliasing
  - canonical, alias names
- mail server aliasing
- load distribution
  - replicated Web servers: many IP addresses correspond to one name

why not centralize DNS?

- single point of failure
- traffic volume
- distant centralized database
- maintenance

A: *doesn't scale!*

DNS: a distributed, hierarchical database

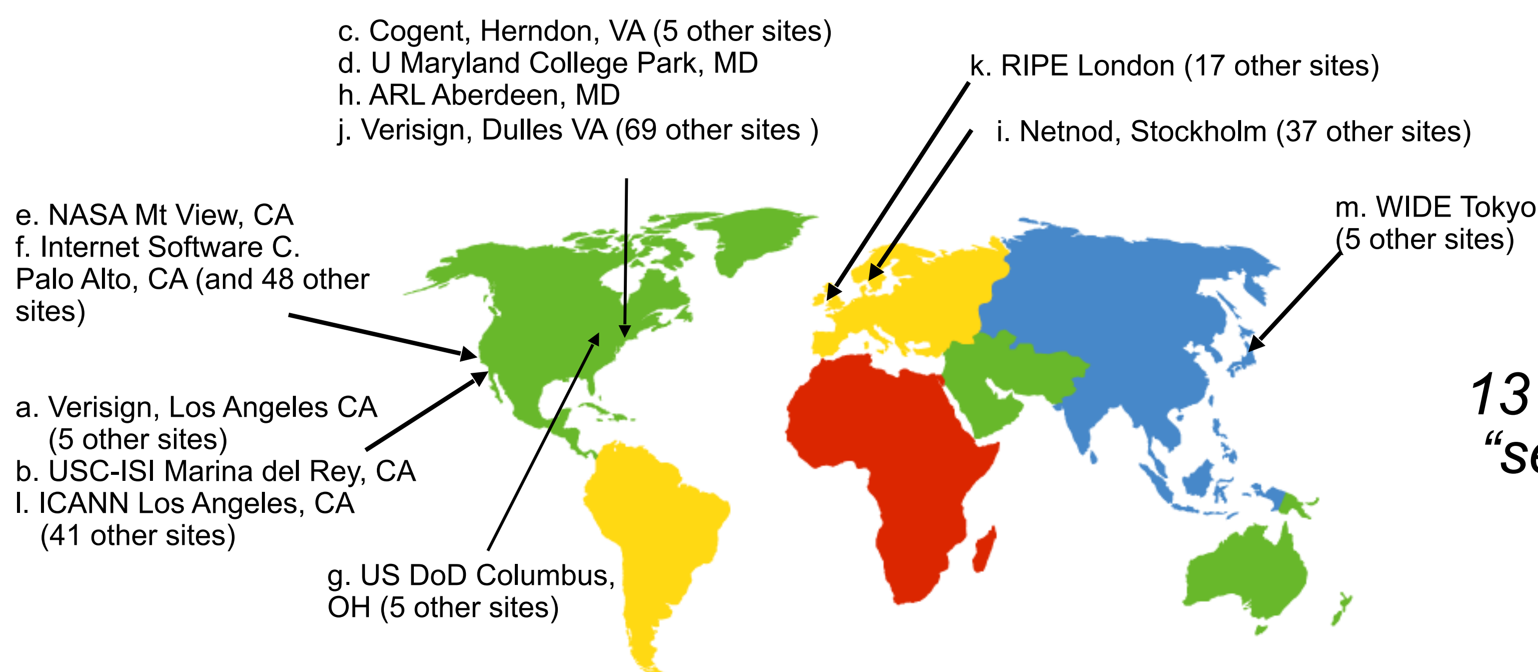


client wants IP for www.amazon.com; 1st approx:

- client queries root server to find com DNS server
- client queries .com DNS server to get amazon.com DNS server
- client queries amazon.com DNS server to get IP address for www.amazon.com

DNS: root name servers

- contacted by local name server that can not resolve name
- root name server:
  - contacts authoritative name server if name mapping not known
  - gets mapping
  - returns mapping to local name server



13 root name "servers" worldwide

TLD, authoritative servers

top-level domain (TLD) servers:

- responsible for com, org, net, edu, aero, jobs, museums, and all top-level country domains, e.g.: uk, fr, ca, jp
- Network Solutions maintains servers for .com TLD
- Educause for .edu TLD

authoritative DNS servers:

- organization's own DNS server(s), providing authoritative hostname to IP mappings for organization's named hosts
- can be maintained by organization or service provider

Local DNS name server

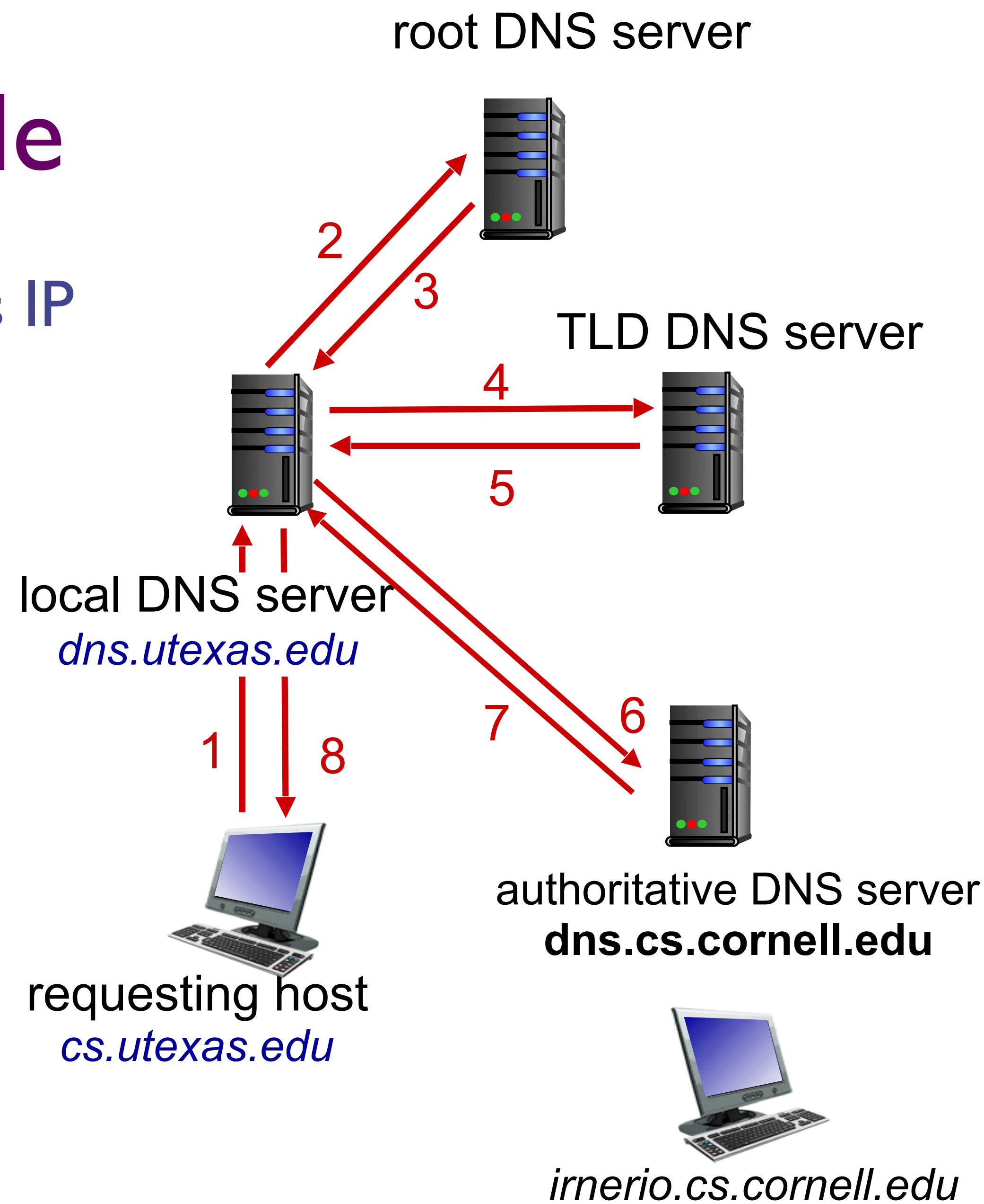
- does not strictly belong to hierarchy
- each ISP (residential ISP, company, university) has one
  - also called "default name server"
- when host makes DNS query, query is sent to its local DNS server
  - has local cache of recent name-to-address translation pairs (but may be out of date!)
  - acts as proxy, forwards query into hierarchy

DNS name resolution example

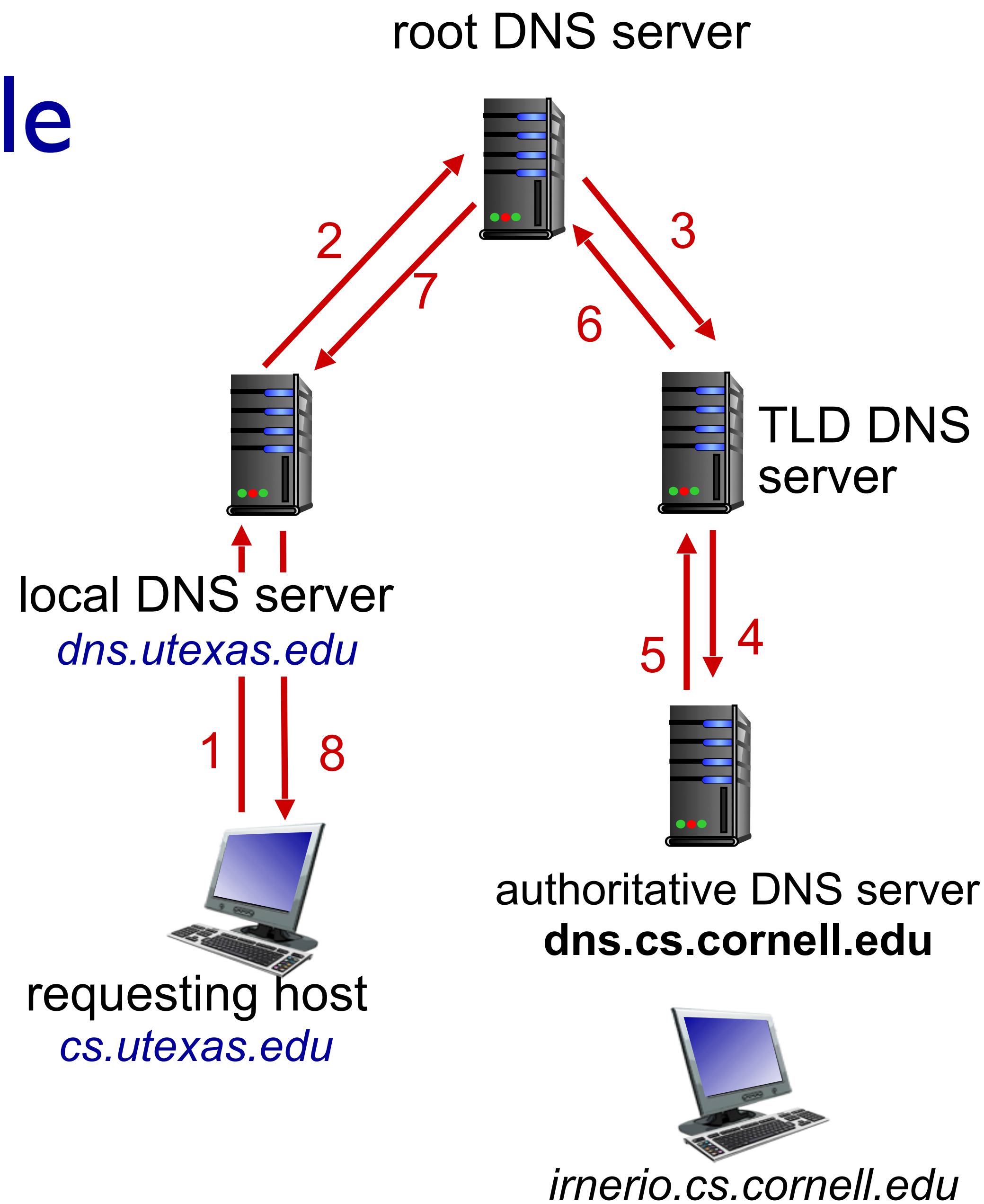
- host at cs.utexas.edu wants IP address for irnerio.cs.cornell.edu

iterated query:

- contacted server replies with name of server to contact
- "I don't know this name, but ask this server"



DNS name resolution example



recursive query:

- puts burden of name resolution on contacted name server
- heavy load at upper levels of hierarchy?

DNS: caching, updating records

- once (any) name server learns mapping, it *caches* mapping
  - cache entries timeout (disappear) after some time (TTL)
  - TLD servers typically cached in local name servers
    - thus root name servers not often visited
- cached entries may be *out-of-date* (best effort name-to-address translation!)
  - if name host changes IP address, may not be known Internet-wide until all TTLs expire
- update/notify mechanisms proposed IETF standard
  - RFC 2136

Attacking DNS

DDoS attacks

- Bombard root servers with traffic
  - Not successful to date
  - Traffic Filtering
  - Local DNS servers cache IPs of TLD servers, allowing root server bypass
- Bombard TLD servers
  - Potentially more dangerous

Redirect attacks

- Man-in-middle
  - Intercept queries
- DNS poisoning
  - Send bogus replies to DNS server, which caches

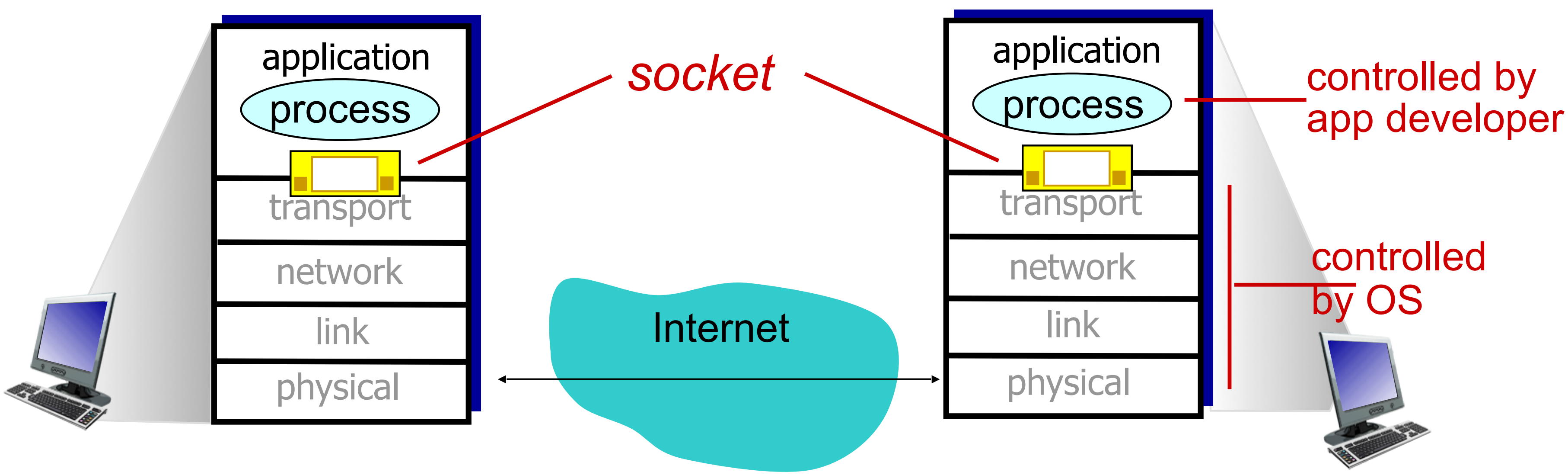
Exploit DNS for DDoS

- Send queries with spoofed source address: target IP
- Requires amplification

Sockets

socket: door between application process and end-end-transport protocol

- sending process shoves message out door
- sending process relies on transport infrastructure on other side of door to deliver message to socket at receiving process



Socket programming

Two socket types for two transport services:

- *UDP:* unreliable datagram
- *TCP:* reliable, byte stream-oriented

Application Example:

1. client reads a line of characters (data) from its keyboard and sends data to server
2. server receives the data and converts characters to uppercase
3. server sends modified data to client
4. client receives modified data and displays line on its screen

Socket programming *with UDP*

UDP: no "connection" between client & server

- no handshaking before sending data
- sender explicitly attaches IP destination address and port # to each packet
- receiver extracts sender IP address and port# from received packet

UDP: transmitted data may be lost or received out-of-order

Application viewpoint:

- UDP provides *unreliable* transfer of groups of bytes ("datagrams") between client and server

Client/server socket interaction: UDP

server (running on serverIP)

create socket, port= x:
serverSocket =
socket(AF_INET,SOCK_DGRAM)

read datagram from
serverSocket

write reply to
serverSocket
specifying
client address,
port number

client

create socket:
clientSocket =
socket(AF_INET,SOCK_DGRAM)

Create datagram with server IP and
port=x; send datagram via
clientSocket

read datagram from
clientSocket

close
clientSocket

Example app: UDP client

Python UDPClient

include Python's socket library

```
from socket import *
serverName = 'hostname'
serverPort = 12000
```

create UDP socket for server

```
clientSocket = socket(AF_INET,
                      SOCK_DGRAM)
```

get user keyboard input

```
message = raw_input('Input lowercase sentence:')
```

Attach server name, port to message; send into socket

```
clientSocket.sendto(message.encode(),
                    (serverName, serverPort))
```

read reply characters from socket into string

```
modifiedMessage, serverAddress =
                    clientSocket.recvfrom(2048)
```

print out received string and close socket

```
print modifiedMessage.decode()
clientSocket.close()
```

Example app: UDP server

Python UDPServer

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET, SOCK_DGRAM)
serverSocket.bind(('', serverPort))
print ("The server is ready to receive")
while True:
    message, clientAddress = serverSocket.recvfrom(2048)
    modifiedMessage = message.decode().upper()
    serverSocket.sendto(modifiedMessage.encode(),
                        clientAddress)
```

create UDP socket

bind socket to local port number 12000

loop forever

Read from UDP socket into message, getting client's address (client IP and port)

send upper case string back to this client

Socket programming *with TCP*

client must contact server

- server process must first be running
- server must have created socket (door) that welcomes client's contact

client contacts server by:

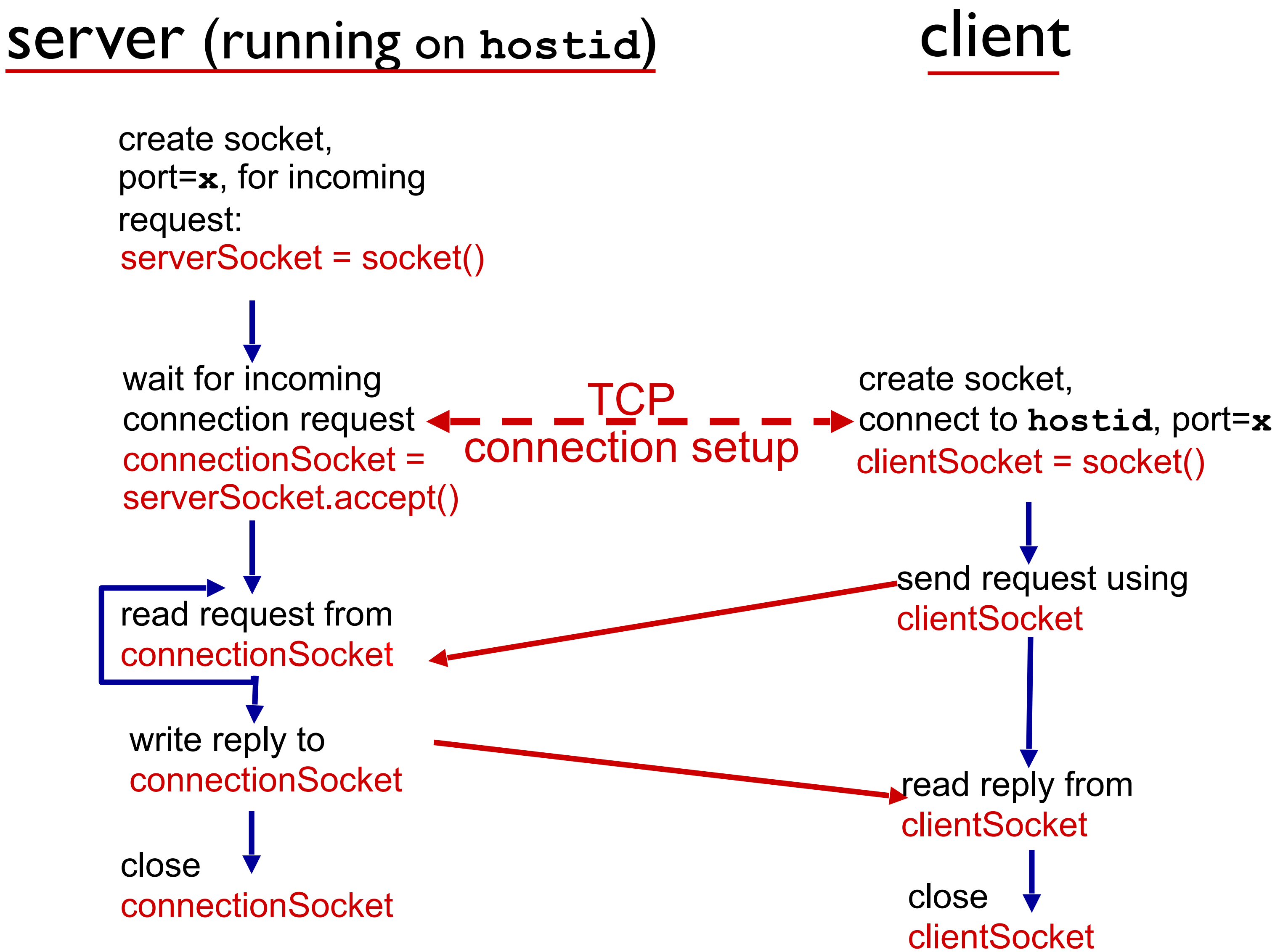
- Creating TCP socket, specifying IP address, port number of server process
- *when client creates socket:* client TCP establishes connection to server TCP

- when contacted by client, *server TCP creates new socket* for server process to communicate with that particular client
  - allows server to talk with multiple clients
  - source port numbers used to distinguish clients (more in Chap 3)

application viewpoint:

TCP provides reliable, in-order byte-stream transfer ("pipe") between client and server

Client/server socket interaction: TCP



Example app: TCP client

Python TCPClient

create TCP socket for server, remote port 12000

No need to attach server name, port

```
from socket import *
serverName = ’servername’
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName,serverPort))
sentence = raw_input(‘Input lowercase sentence:’)
clientSocket.send(sentence.encode())
modifiedSentence = clientSocket.recv(1024)
print (‘From Server:’, modifiedSentence.decode())
clientSocket.close()
```

Example app: TCP server

Python TCPServer

create TCP welcoming socket

server begins listening for incoming TCP requests

loop forever

server waits on accept() for incoming requests, new socket created on return

read bytes from socket (but not address as in UDP)

close connection to this client (but *not* welcoming socket)

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET,SOCK_STREAM)
serverSocket.bind((‘’,serverPort))
serverSocket.listen(1)
print ‘The server is ready to receive’
while True:
    connectionSocket, addr = serverSocket.accept()

    sentence = connectionSocket.recv(1024).decode()
    capitalizedSentence = sentence.upper()
    connectionSocket.send(capitalizedSentence.
                                              encode())
    connectionSocket.close()
```